National Aeronautics and Space Administration

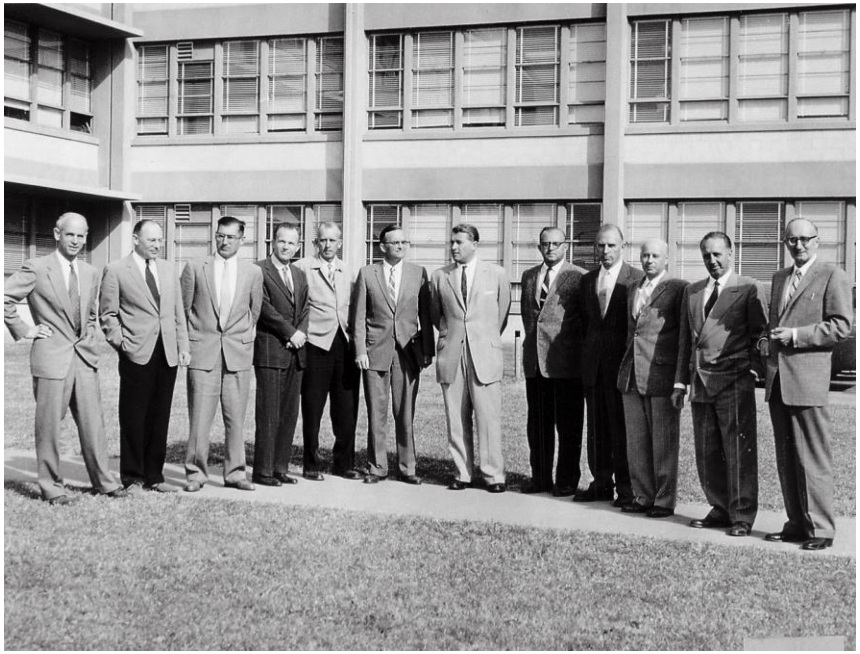
core Flight System (cFS) Background and Overview

Agenda

- **What is Flight Software**
- **cFS Overview**
- **cFS History and Motivation**
- **cFS Architecture and Design**
  - Quality Analysis
  - Key Trades
  - Concepts and Standards
  - Layers and Components
    - Real Time OS
    - OS Abstraction
    - Platform Support Package
    - Core Services
    - Applications
    - Tools
    - Documentation
- **Operational Scenarios**
- **Deployment**
- **Run Time**
- **Mission Examples**
- **Lessons Learned**

Audience Background

- **Familiarity with the Spacecraft Domain**
  - LEO, MEO, GEO, Deep Space
  - Radiation environment
  - Size, Mass, and Power (SWaP)
  - Limited memory
  - Limited processing power
  - Autonomy
  - Fault management



- **Software**
  - Languages?
  - Tools?
  - Operating systems?

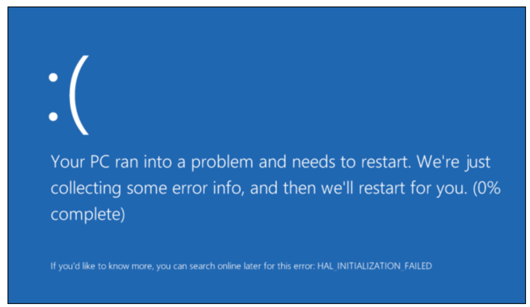
What is Flight Software (FSW)?

Definitions

- **A few terms:**
  - **Spacecraft Bus** – usually refers to the fundamental systems of a spacecraft, i.e.
    - Mechanical Structure
    - Electrical System
    - Power System
    - Command and Data Handling System (C&DH)
    - Attitude Control System/ Propulsion System
    - RF System
    - Thermal System
  - **Payloads** – refers to the instruments on board, i.e.
    - Cameras, Telescopes, Radars, etc
  - **Observatory** – Usually refers to the entire system, i.e. the combination of the Spacecraft Bus and the Payloads

What is Flight Software?

- **First, What's Software?**
  - A general term primarily used for digitally stored data such as computer programs and other kinds of information read and written by computers (Wikipedia)
  - You really know what it is when it doesn't work!!



- **Flight Software is**
  - Software that flies (for us at NASA, that typically means on a spacecraft)
  - Could be part of the Spacecraft Bus, or an Instrument
  - Hosted within flight electronics CPU; e.g., embedded in the C&DH
  - Starts when Spacecraft Power is applied to the CPU
  - The “Brains” of the on-orbit mission
  - Major enabler to support technology capabilities of future missions

Flight Software is Unique

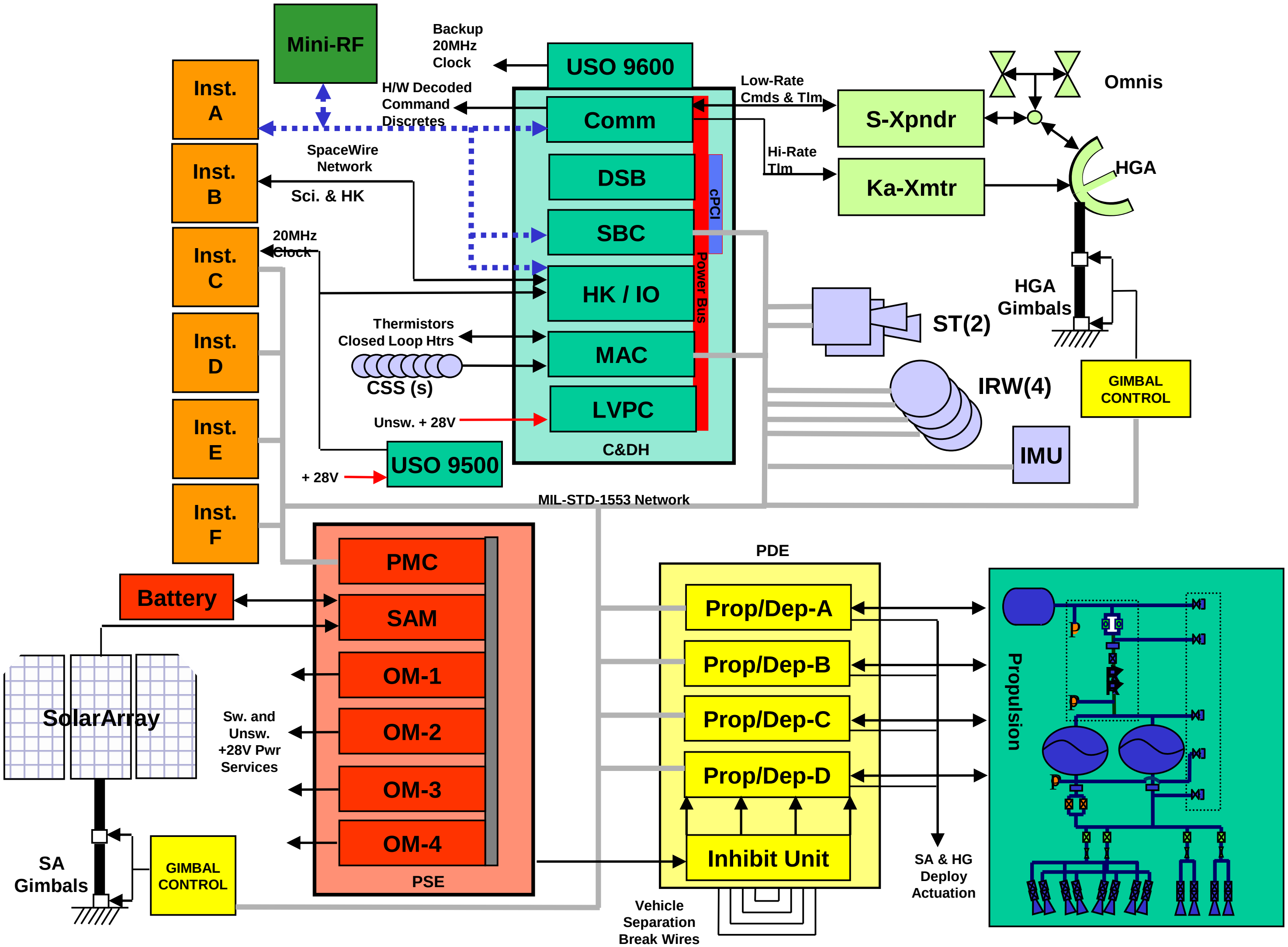
- **FSW is embedded" software**
  - "computer software, written to control machines or devices that are not typically thought of as computers. It is typically specialized for the particular hardware that it runs on and has time and memory constraints." - (Wikipedia)

- **FSW must handle things in "real time"**
  - Guarantee a response within required time constraint or deadline
  - Deterministic
  - Reliable

- **FSW is Mission Critical**
  - Must keep the spacecraft safe through an anomaly (i.e. solar arrays pointed to sun, antenna pointed to ground)
  - Spacecraft is not always in contact with the control center(s) and therefore must be able to act "autonomously"
  - Must be maintainable

Typical Block Diagram

Simplified Avionics Systems/ The "Observatory"



Typical Flight Software Requirements

- **C&DH System:**
  - *Establish the startup configuration*
  - *Manage command and telemetry*
    - *Distribute commands/ Format telemetry for downlink*
    - *Store engineering and science data onboard*
  - *Control the flow of on-board operations*
  - *Time Management*
    - *Manage and distribute on-board time*
    - *Time-tag data*
  - *Allow for upload and execution of new software*
  - *Manage Fault Detection and Correction (FDC)**

- ***Power System:***
  - *Ensure solar arrays point to the Sun*
  - *Ensure batteries are charged*
  - *Control the distribution of power to onboard subsystems*

Typical Flight Software Requirements

- **GN&C System:**
  - *Determine current attitude*
  - *Control momentum build-up*
  - *Determine current orbit position/velocity*
  - *Control Delta-V maneuvers*

- **RF System:**
  - *Manage the Downlink*
  - *Accept the Uplink*
  - *Manage antennae pointing for ground contacts (GN&C)*

- ***Instruments:***
  - *Configure science instruments*
  - *Capture science data (may process data)*

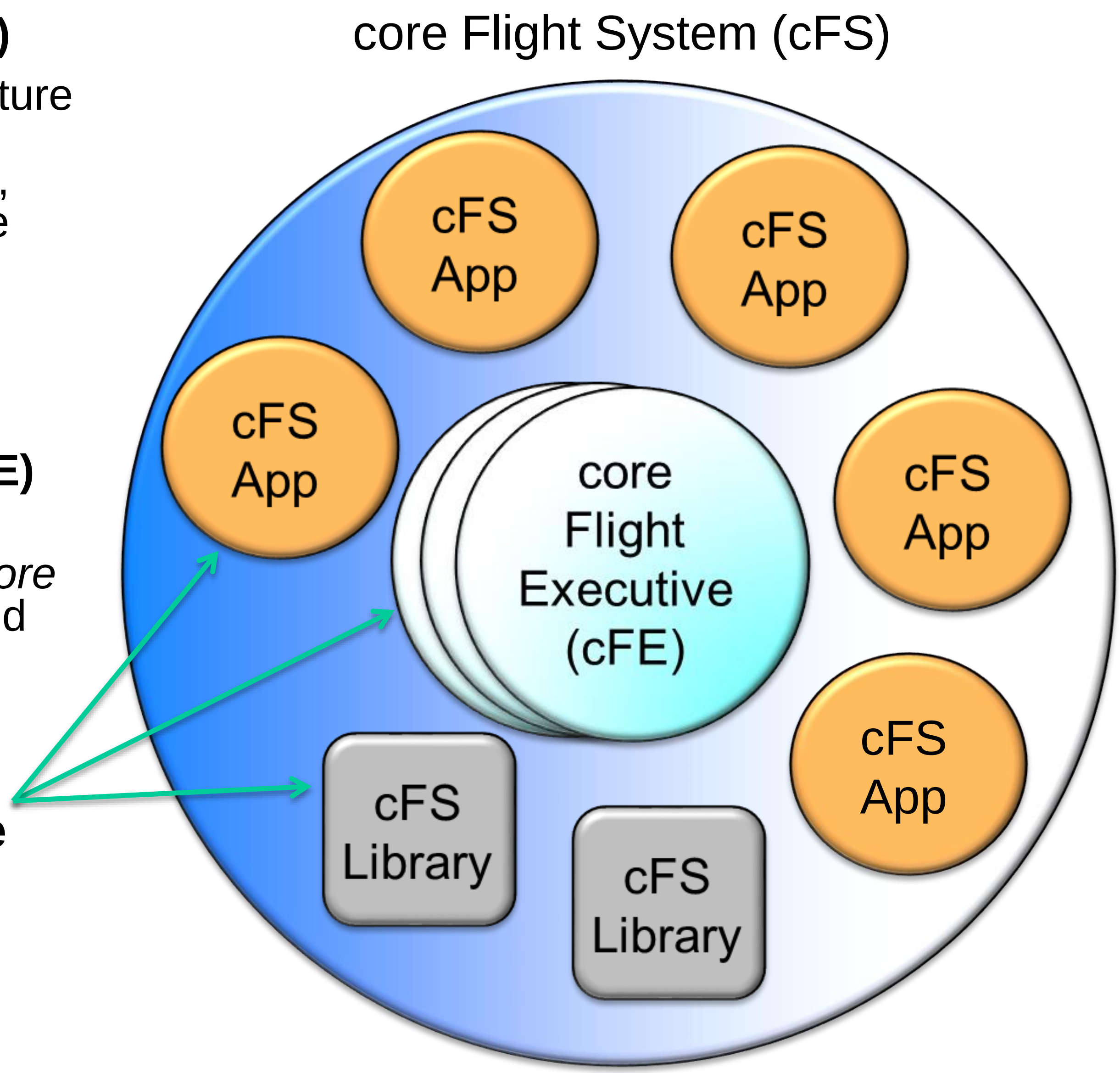
cFS Overview

cFS Overview

- **core Flight System (cFS)**
  - A Flight Software Architecture consisting of an OS Abstraction Layer (OSAL), Platform Support Package (PSP), cFE Core, cFS Libraries, and cFS Applications

- **core Flight Executive (cFE)**
  - A framework of *mission independent, re-usable, core* flight software services and operating environment

- **Each element is a separate loadable file**

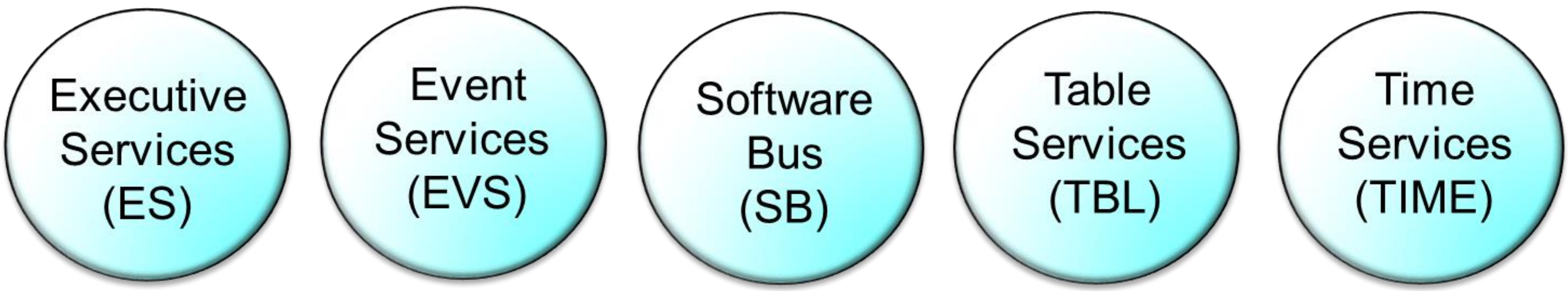


cFS Overview

- **A set of** *mission independent, re-usable, core* **flight software services, applications, and operating environment**
  - Layered architecture
    - Supports a variety of hardware platforms
  - Provides standardized Application Programmer Interfaces (API)
  - Supports and hosts flight software applications
    - *Applications can be added and removed at run-time (eases system integration and FSW maintenance*)
  - Supports software development for on-board FSW, desktop FSW development and simulators
  - Contains platform and mission configuration parameters that are used to tailor to a specific platform and mission.

cFS Overview – cFE Services

- **cFE services include:**



- **Support services include:**
  - File utilities

cFS Overview - Applications

Application	Function
CFDP	Transfers/receives file data to/from the ground
Checksum	Performs data integrity checking of memory, tables and files
Command Ingest Lab	Accepts CCSDS telecommand packets over a UDP/IP port
Data Storage	Records housekeeping, engineering and science data onboard for downlink
File Manager	Interfaces to the ground for managing files
Housekeeping	Collects and re-packages telemetry from other applications.
Health and Safety	Ensures that critical tasks check-in, services watchdog, detects CPU hogging, and calculates CPU utilization
Limit Checker	Provides the capability to monitor values and take action when exceed threshold
Memory Dwell	Allows ground to telemeter the contents of memory locations. Useful for debugging
Memory Manager	Provides the ability to load and dump memory
Software Bus Network	Passes Software Bus messages over various “plug-in” network protocols
Scheduler	Schedules onboard activities via (e.g. HK requests)
Scheduler Lab	Simple activity scheduler with a one second resolution
Stored Command	Onboard Commands Sequencer (absolute and relative)
Telemetry Output Lab	Sends CCSDS telemetry packets over a UDP/IP port

History and Motivation

The Aerospace Domain is Unique

- **Missions require use of specialized, radiation tolerant hardware**
  - Complete COTS solutions do not exist
  - Fixed and constrained environment
    - Speed of processor
      - Example: LRO uses 166 MHz processor, my laptop uses 2.5 GHz processor
    - Amount of memory and storage
      - Example: LRO has 2MB of code memory, my laptop has 4GB of RAM

- **Complex software system**
  - High speed science operations
  - High reliability, fault tolerant
  - Autonomous operations
  - On orbit maintenance

$ These challenges increase the cost of satellite software $

History - Re-use in the Past

- **In the past, little cost saving has been realized via FSW reuse**
  - No product line. Instead heritage missions were used as starting point (Clone & Own)
  - Changes made to the heritage software for the new mission were not controlled
    - New flight hardware or Operating System required changes throughout FSW
    - FSW Requirements were sometimes re-written which affects FSW and tests.
    - FSW changes were made at the discretion of developer
    - FSW test procedure changes were made at the discretion of the tester
    - Extensive documentation changes were made for style
  - Not all Products from heritage missions were available
  - Reuse was not an formal part of development methods
  - Reuse was not enforced

History and Motivation

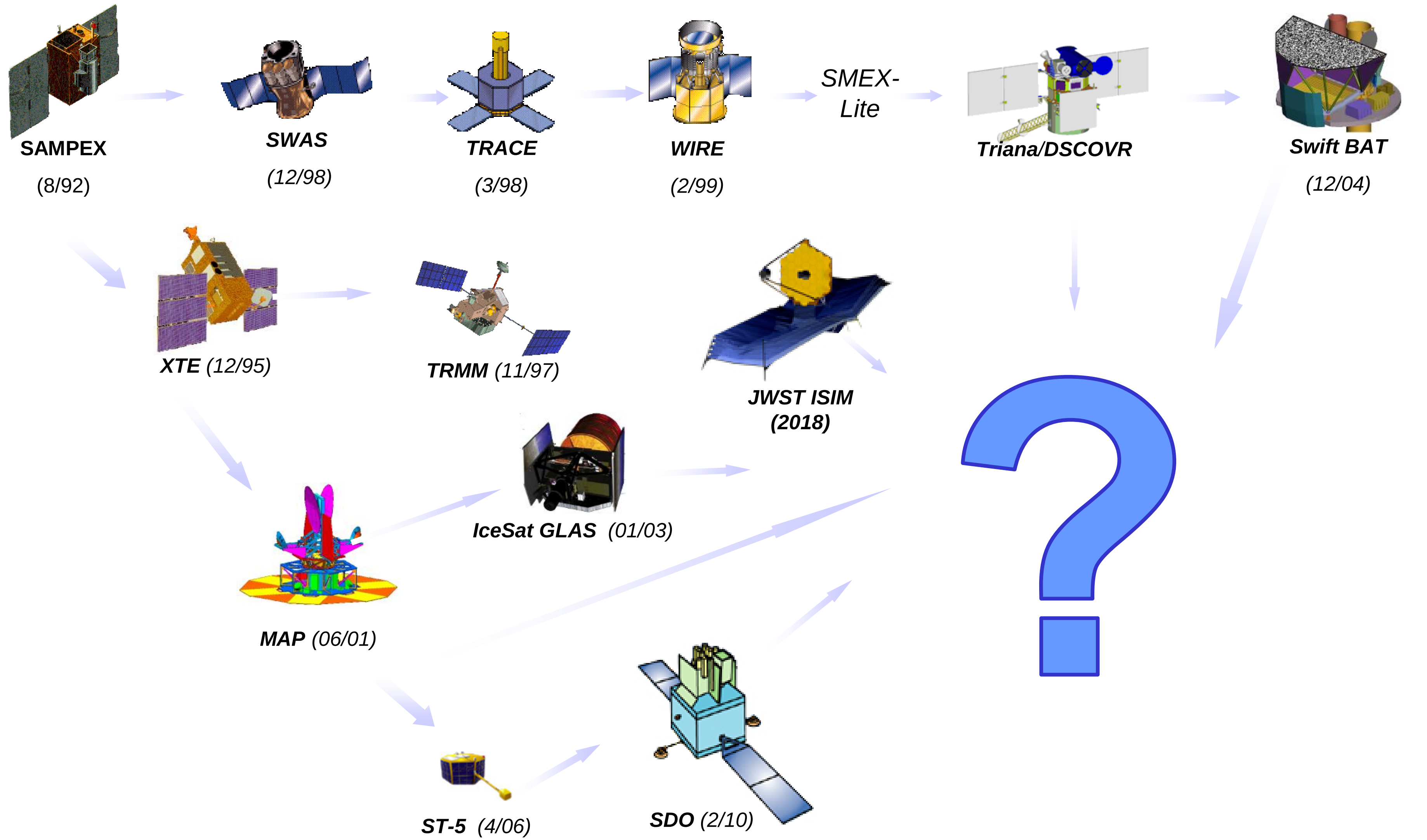
- **Several years ago, GSFC was tasked two large in-house missions with concurrent development schedules (SDO, GPM)**
- **GSFC Code 582 was to design and build the spacecraft bus, avionics and flight software and integrate these components with the spacecraft**
- **Without the staff for both projects and a reduced budget, we needed to find a better way**
  - We had about a year to figure it out before staffing up

History and Motivation
582's Approach

- **Formed a team of senior FSW engineers**
  - Management isolated team engineers from short term mission schedules
  - Each had experience on different missions and saw commonality across the missions

- **Team then decided to:**
  - Determine impediments to good flight software reuse
  - Perform heritage analysis
    - Utilize best concepts from missions ranging from Small Explorer class to the Great Observatories
    - Utilize commonality across missions
  - Design with reusability and flexibility in mind
  - Take advantage of software engineering advances
  - Establish architecture goals

cFS Flight Software Architecture Heritage



Heritage - What Worked Well

- **Message bus**
  - All software applications use message passing (internal and external)
  - CCSDS standards for messages (commands and telemetry)
  - Applications were processor agnostic (distributed processing)
- **Layering**
- **Packet based stored commanding (AKA Mission Manager)**
  - Absolute Time Sequence (ATP), Relative Time Sequence (RTP)
- **Vehicle Failure Detection Isolation and Recovery (FDIR) based on commands and telemetry packets**
- **Table driven applications**
- **Critical subsystems synchronized to the network schedule**
  - 1553 bus master Time Division Multiple Access (TDMA)
- **Clean application interfaces**
  - Component based architecture (The Lollipop Diagram)

Heritage - What Worked Well

- **Lots of innovation**
  - Constant pipeline of new and varied missions
  - Teams worked full life cycle
    - Requirements through launch + 60days
    - Maintenance teams in-house and in contact with engineers early in development
  - Teams keep trying different approaches
    - Rich heritage to draw from
- **The little “c” in cFE and cFS**
  - A little core framework, as in low footprint, optimized for flight systems
    - Full cFS suite with FreeRTOS in 800KB flash with 2MB RAM for cubesats

Heritage - What Didn't Work So Well

- **Statically configured Message bus**
  - Scenario: GN&C needs a new diagnostic packet
    - Give the C&DH team your new packet definition file
    - Wait a week for a new interim build
    - Rinse and Repeat
  - How do I add a new one on orbit?
- **Monolithic load (The "Amorphous Blob")**
  - Raw memory loads and byte patching needed to keep bandwidth needs down
- **Reinventing the wheel**
  - Mission specific "common" services ("Look , I've got a new and improved version!")
- **Application rewrites for different OSes**

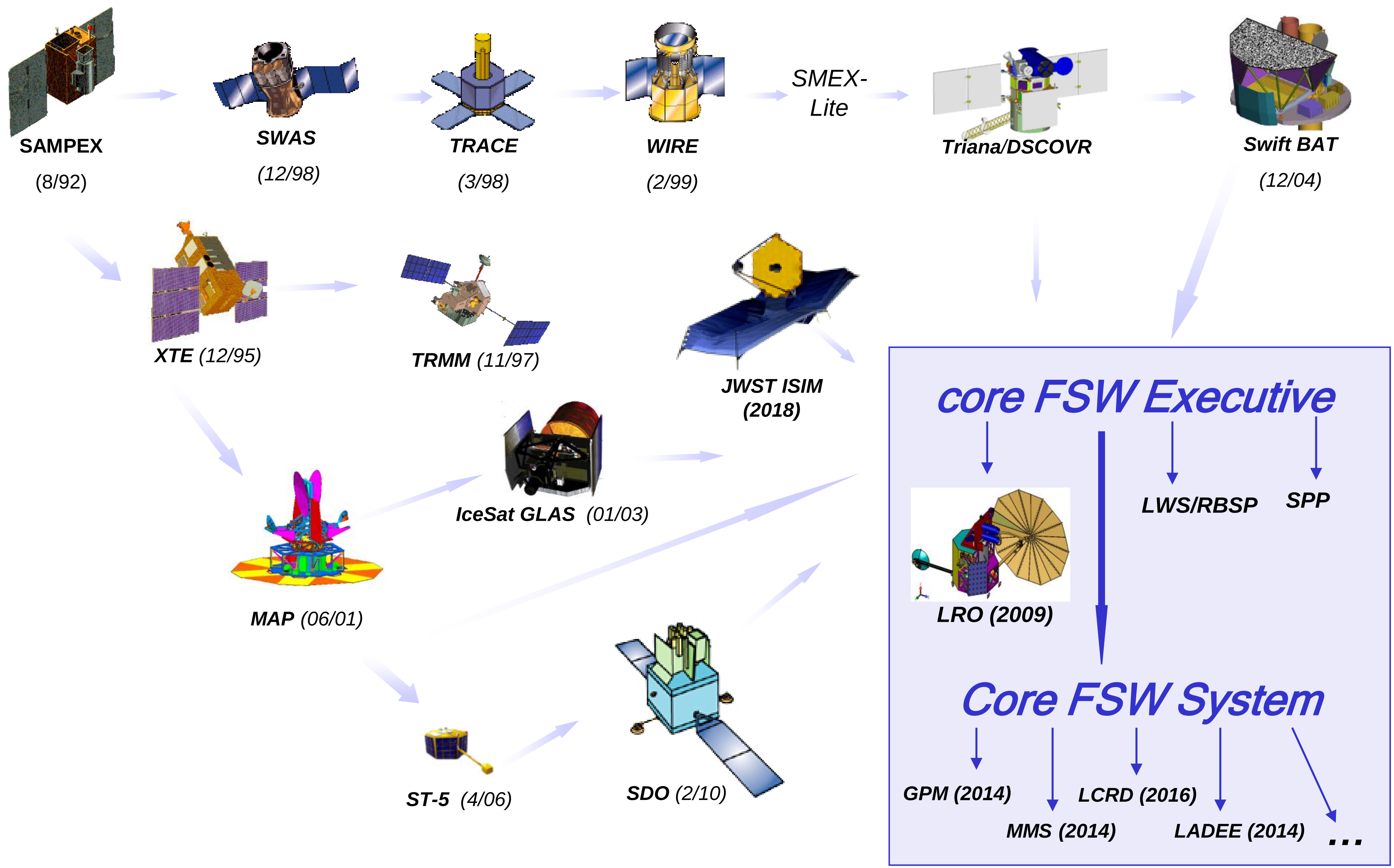
Heritage - What Didn't Work So Well

- **Statically configured tables**
  - Scenario: GN&C needs a gyro scale factor table ….
- **Tool directory structure coupling**
  - Some of your application files go here, some there, and some over there
- **Modeling tools and the Amorphous Blob**
  - Tools did not support component loadable objects
- **Implementing device drivers in C++ (1553 example)**
- **Claims of high reuse, but it still took the same effort on each mission**

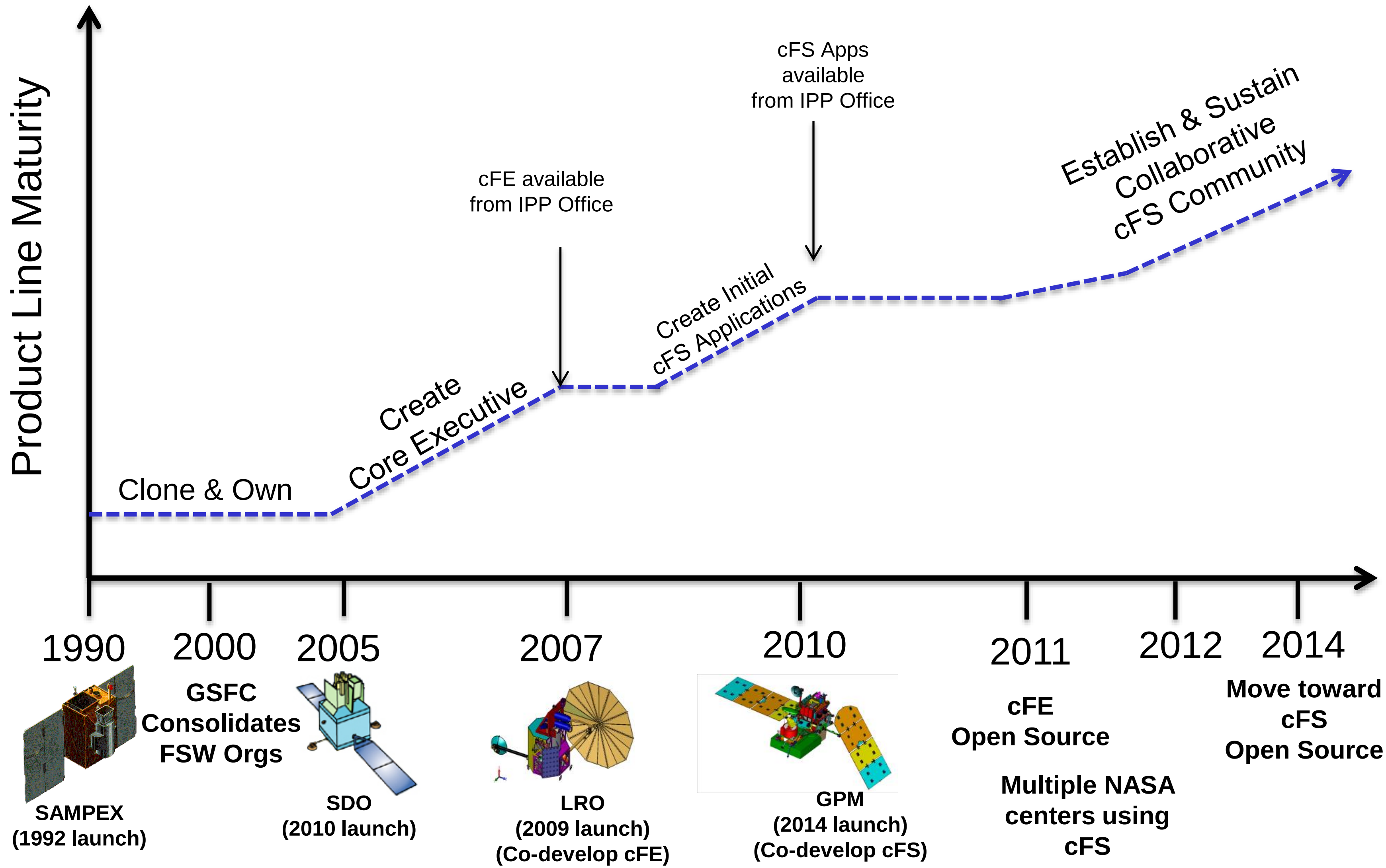
Architecture Goals

1. Reduce time to deploy high quality flight software
2. Reduce project schedule and cost uncertainty
3. Directly facilitate formalized software reuse
4. Enable collaboration across organizations
5. Simplify sustaining engineering (AKA. On Orbit FSW maintenance) Missions last 10 years or more
6. Scale from small instruments to Hubble class missions
7. Build a platform for advanced concepts and prototyping
8. Create common standards and tools across the center

cFS Architecture Heritage



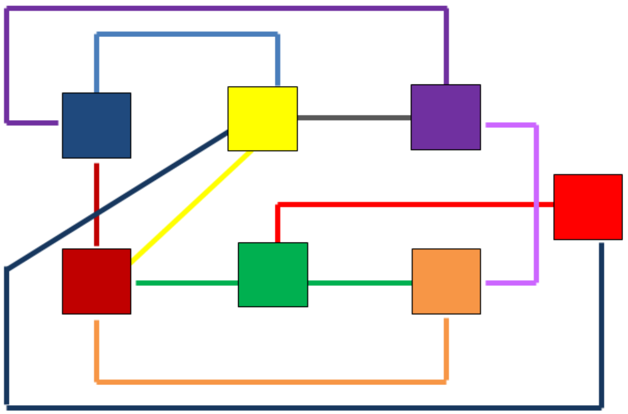
cFS Timeline

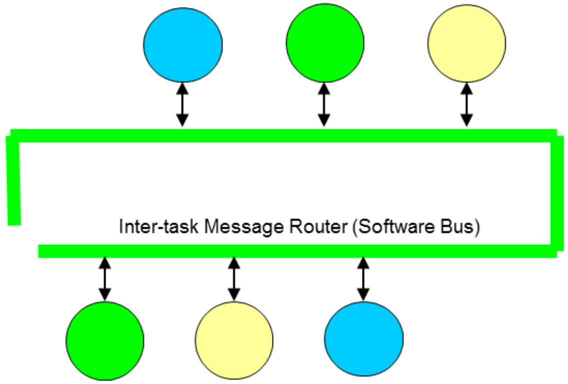


Paradigm Shift in Development

- **Traditional approach**
  - Each mission had its own solution typically based on a previous mission
    - Requires extensive, laborious and error prone requirement, design and code changes.
- **cFS-based approach**
  - Baseline code infrastructure and artifacts already completed, tested, and qualified
  - Standardizes flight software development
  - Applications can be added and removed at runtime

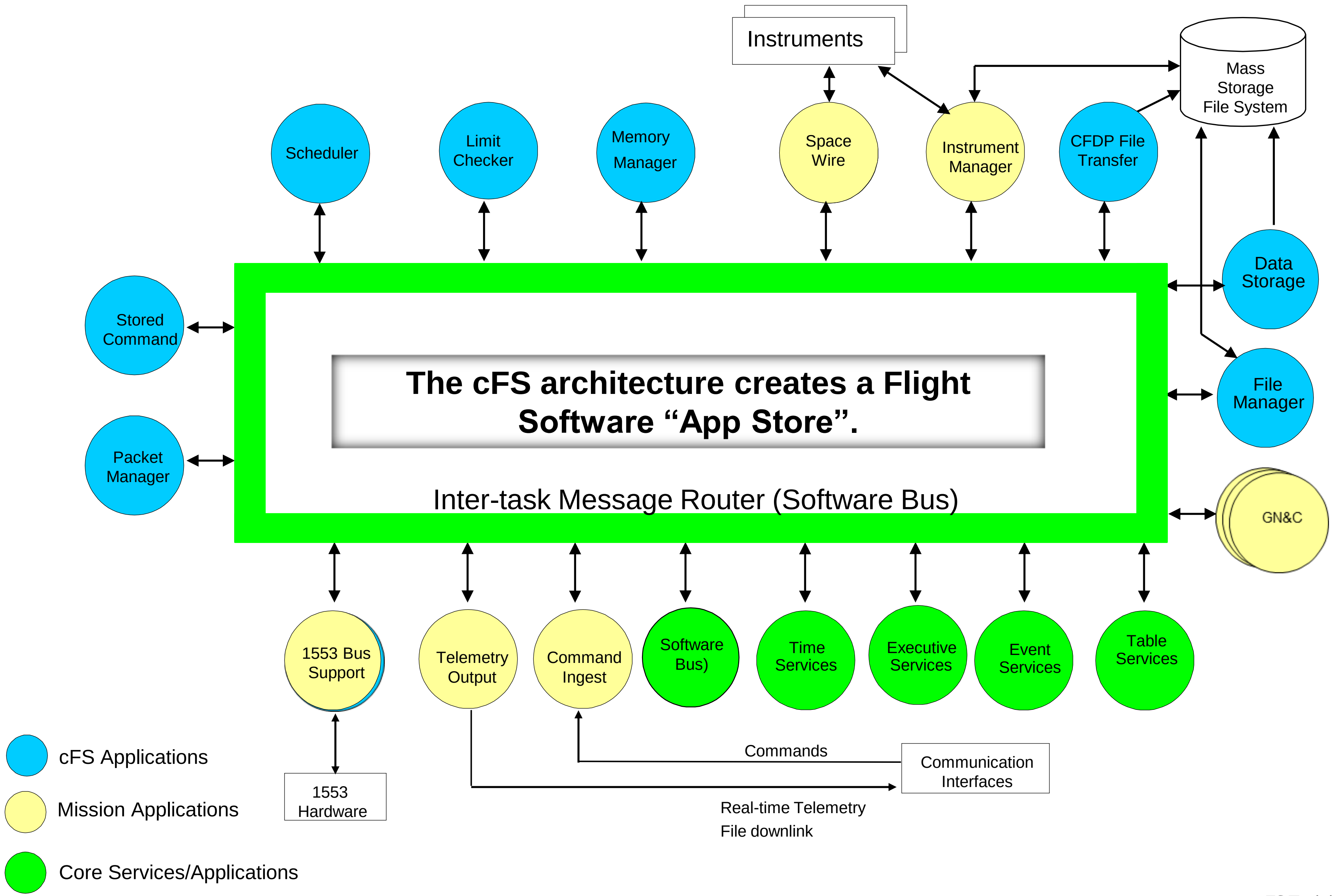
Legacy: tightly-coupled, custom interfaces - data formats, protocols, internal knowledge & component interdependence





Publish/Subscribe: loosely-coupled, standard interface - data formats, protocols, & component Independence

Lollipop/Bubble Diagram - Example



A Complete Engineering Solution

Includes reusable:

- Requirements
- Source Code
- Design Documentation
- Development Standards
- Test Artifacts
- Tools
  - Unit Test Framework
  - Software Timing Analyzer
- User's Guides
  - Application Developers Guide
  - API Reference Guides
  - Deployment Guides
  - Flight Operations Guides
- Simple Ground system

The cFS architecture reduces Non-Recurring Engineering (NRE) up to 90%

cFS Use

- **In use at seven NASA centers:**
  - Ames Research Center
  - Glenn Research Center
  - Goddard Space Flight Center
  - Johnson Space Center
  - Kennedy Space Center
  - Marshall Space Flight Center
  - Langley Research Center

- **Used/In Development on:**
  - Landers
  - Orbiters
  - Unmanned Aerial Vehicles
  - Space Suits
  - Crew Habitats
  - Rovers
  - Spacecubes
  - SmallSats
  - PiSats

Architecture and Design Quality Analysis

Quality Analysis - 1

- **Operability**
  - The architecture must enable the flight system to operate in an efficient and understandable way
- **Reliability**
  - The architecture implementation must be known to behave correctly in nominal and expected off-nominal situations
- **Robustness**
  - The architecture implementation must be predictable and safe in the presence of unexpected conditions
- **Performance**
  - The architecture implementation must be efficient in runtime resources given the targeted processing environments
- **Testability**
  - The architecture implementation must be easily and comprehensively testable in-situ in flight like scenarios
- **Maintainability**
  - The architecture implementation must be maintainable in the operational environment

Quality Analysis - 2

- **Effective Reuse**
  - The architecture must support an effective reuse approach. This includes the software and artifacts. Requirements, design, code, review presentations, test, operations guides, command and telemetry databases. The goal is to achieve 100% reuse of a software component with no code changes
- **Composability**
  - Properties established at the component level, such as interfaces, timeliness or testability, also hold at the system level. For an application or node to be composable the architecture and process must support:
    - Independent development of nodes
    - Integration of the node into a system should not invalidate services in the value and temporal domains
    - Integration of an additional node into a functioning system should not disturb the correct operation of the existing nodes
    - Replica determinism – identical copies of nodes must produce identical results in an identical order, within a specified time interval
- **Predicable Development Schedule**
  - Development estimates provided by the FSW team should be reliable

Quality Analysis - 3

- **Scalability**
  - The FSW must scale with mission requirements. (Example: instruments or subsystem processor may only need a small amount of message buffer space. This should be configurable to avoid wasting memory resources)
- **Adaptability**
  - The FSW must be capable of supporting a range of platforms and missions
- **Minimized Development Cost**
  - Costs for mission functions should be as low as possible. The teams must consider the difference between NRE and costs for a given mission
- **Technology infusion**
  - The FSW should support the infusion of new hardware and software technologies with minimal side effects

Architecture and Design
Key Trades

Architecture Trades - Pub/Sub messaging

- **Publish - just send data packets**
  - Destination agnostic
  - Components can be configured to limit command sources
- **Subscribe – any component can receive/listen**
- **Peer to Peer network**
  - No master, stateless
    - Component/node stops and data is un-subscribed automatically
  - Robust/Fault tolerant
- **Ground systems, models look like any other component/node**
  - External interfaces can be controlled and firewalled
- **CCSDS packet format**
  - All the pieces (Id, time, seq#, length) and extensible
  - Works well with existing GSFC ground systems
- **Looked at CCSDS Asynchronous Message Service (AMS)**
- **Looked at COTS Network Data Distribution Service (NDDS)**

Architecture Trades - File System

- **File systems are a well supported abstraction for data storage**
- **Standard file transfer mechanisms (TFTP, FTP, CFDP)**
- **General operating system support**
- **No GSFC missions had flown a file system**
  - Triana never launched
- **Lots of resistance to added complexity**
  - VxWorks DOS file system issues on spacecraft
- **Result:**
  - Use file for code, data and recorder
  - LRO used VxWorks file system with work-arounds (stat example)
  - Looking at JPL file system
    - RAMFS – A Volatile Memory Filesystem
      - POSIX compliant, SPIN® checked
  - Funding RTEMS robust file system work

Architecture Trades - “C” language (cFE)

- **Small footprint**
- **Links and binds with other languages C++, ADA, scripts**
- **Not required for all components, just the cFE**
- **Component interfaces are standard not the implementation**
  - Issue becomes the supporting language library
- **Most modeling tools can interface to “C”**

Architecture Trades – Development Tools

- **Rational Rose UML (2004) time frame**
  - Did not support MMU
  - Single binary image
  - High cost for small projects
- **Rational Rose UML (2010) time frame**
  - Adding support for MMU and VxWorks RTP
  - RTPs can be separate loadable object files
- **GNU based compilers and linkers**
  - Supports multiple platforms
  - Non proprietary
  - Tool chain can be modified
  - Long-term tool use without issues of licenses and vendor obsolescence
- **Result: GNU tool chain**

Architecture Trades - Linking

- **Dynamic linking**
  - Requires symbols tables on board
  - Executable Linkable Format (ELF) code files about double in size
  - More efficient use of memory (No “spacers” required)
  - Can map around bad memory blocks (MMU required)
- **Static linking**
  - No on board symbols
  - Small code files (stripped ELF)
  - Absolute location for each software component
  - Need to add margin around component memory space
- **Trade result:**
  - The architecture will support both
  - Open source RTEMS now has support for both (GSFC funded)

Architecture Trades - UML

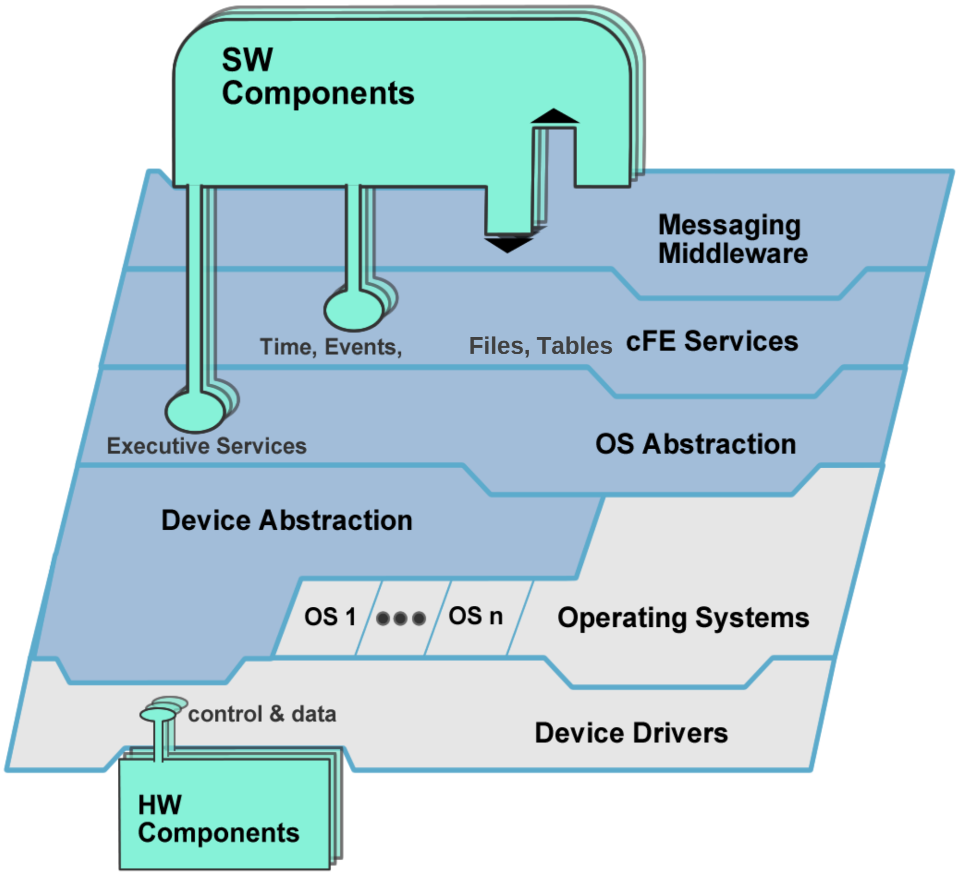
- With well defined cFE interfaces and services, it was always envisioned that cFS components could be created with modeling tool auto generated code and linked with the GNU tool chain
- This has been done somewhat with Matlab/Simulink for GN&C and is the topic for the first cFS workshop

Concepts and Standards

• Layered Architecture • Standard Middleware/Bus • Standard Application Programmer Interface for a set of core services	Core Flight Executive (cFE)
• Plug and Play Reusable Applications • Command & Telemetry database	cFS Applications
• Reuse Requirements Management • Reuse Standards • Reuse Repository	Library & CM
• Configuration Tool for Mission Users • Development Tools	Integrated Development Environment

Layered Service Architecture

- Each layer and service has a standard API
- Each layer "hides" its implementation and technology details.
- Internals of a layer can be changed -- without affecting other layers' internals and components.
- Provides Middleware, OS and HW platform-independence.



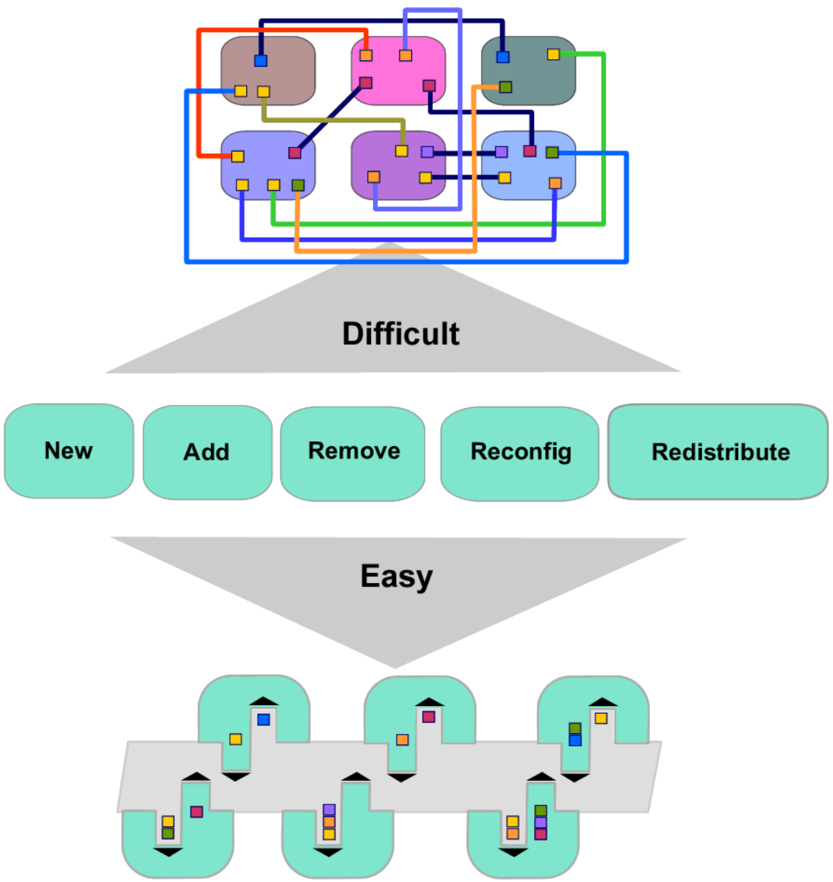
Plug and Play

Plug and Play

- cFE API's support add and remove functions
- SW components can be switched in and out at runtime, without rebooting or rebuilding the system SW.
- Qualified Hardware and cFS-compatible software both "plug and play."

Impact:

- Changes can be made dynamically during development, test and on-orbit even as part of contingency management
- Technology evolution/change can be taken advantage of later in the development cycle.
- Testing flexibility (GSE, test apps, simulators)

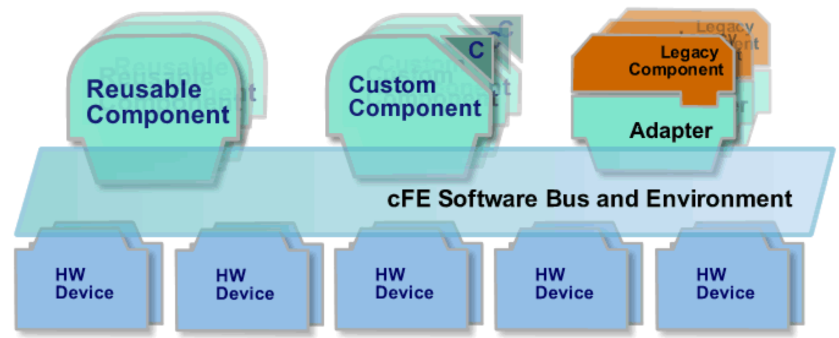


This powerful paradigm allows SW components to be switched in and out at runtime, without rebooting or rebuilding the system SW.

Reusable Components

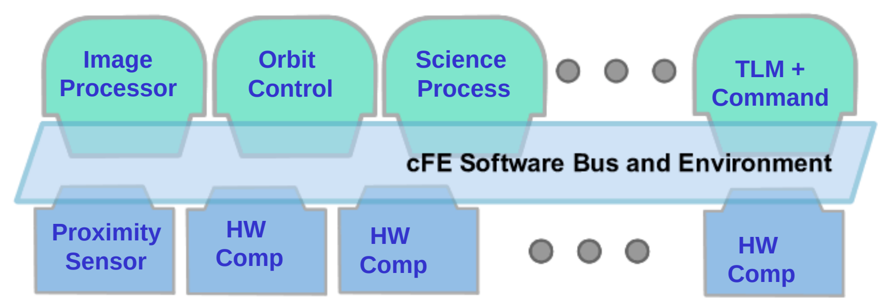
Reusable Components

- Common FSW functionality has been abstracted into a library of reusable components and services.
- Tested, Certified, Documented
- A system is built from:
  - Core services
  - Reusable components
  - Custom mission specific components
  - Adapted legacy components



Impact:

- Reuse of tested, certified components supplies savings in each phase of the software development cycle
- Reduces risk
- Teams focus on the custom aspects of their project and don't "reinvent the wheel."



Component Example

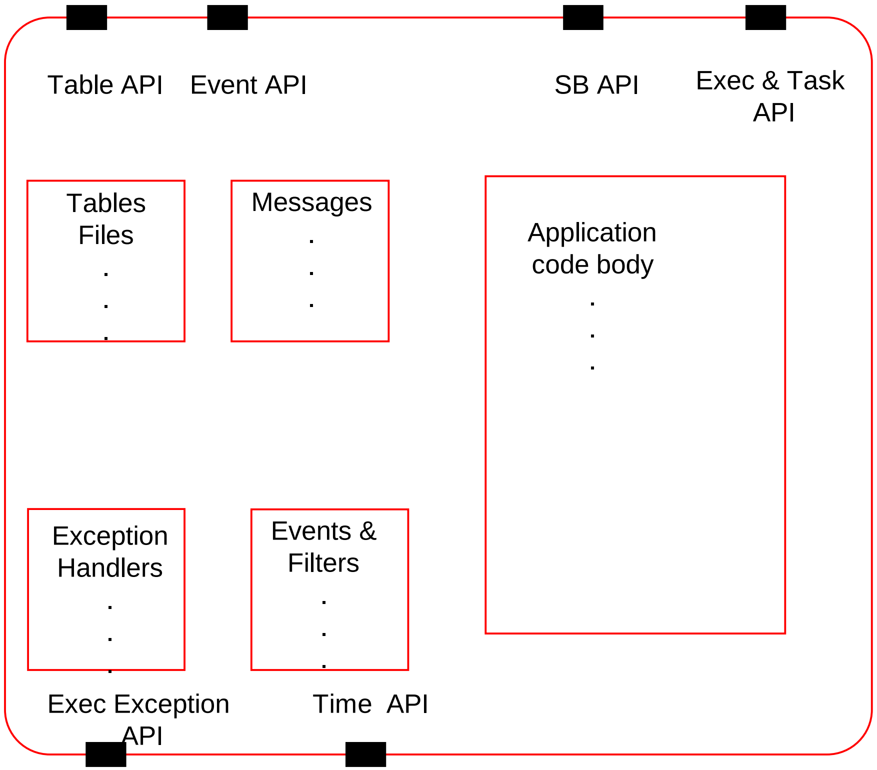
• Interface only through core API's.

• A component contains all data needed to define it's operation.

• Components register for services
- Register exception handlers
- Register Event counters and filter
- Register Tables
- Publish messages
- Subscribe to messages

• Component may be added and removed at runtime. (Allows rapid prototyping during development)

• Configuration Parameters



Configuration Parameters

- **Mission configuration parameters – used for ALL processors in a mission (eg. time epoch, maximum message size, etc)**
  - Default contained in:
    - \cfe\fsw\mission_inc\cfe_mission_cfg.h
    - \apps\app\fsw\mission_inc\app_mission_cfg.h
    - \apps\app\fsw\mission_inc\app_perfids.h
  - Mission version maintained in \build\mission_inc

- **Platform Configuration parameters – used for the specific processor (eg. time client/server config, max number of applications, max number of tables, etc)**
  - Defaults contained in:
    - \cfe\fsw\platform_inc\cpuX\cfe_platform_cfg.h
    - \cfe\fsw\platform_inc\cpuX\cfe_msgids_cfg.h
    - \apps\app\fsw\platform_inc\app_platform_cfg.h
    - \apps\app\fsw\platform_inc\app_msgids.h
    - \osal\build\inc\osconfig.h
  - Mission versions contained in \build\cpuX\inc

Configuration Parameter Types

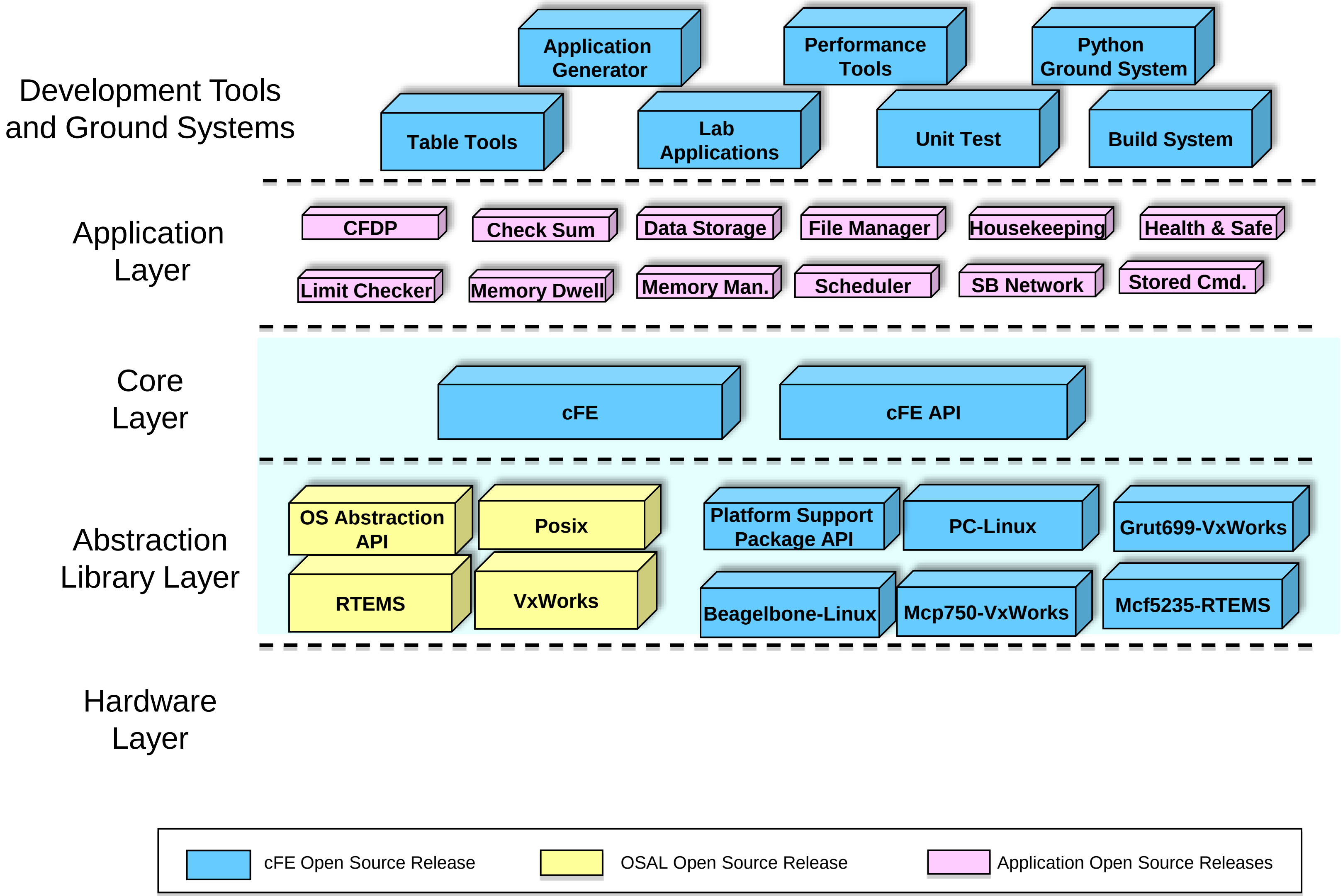
- **Software Bus Message Identifiers**
  - cfe_msgids.h (message IDs for the cFE should not have to change)
  - app_msgids.h (message IDs for the Applications) are platform configurations
- **Executive Service Performance Identifiers**
  - cFE performance IDs are embedded in the core
  - app_perfids.h (performance IDs for the applications) are mission configuration
- **cFE, osal, and application specific configurations**
  - xxx_mission_cfg.h, xxx_platform_cfg, and osconfig.h
- **Makefiles**
  - Specifies the platforms and their applications and tables to be built for a mission

cFS Application Mission and Platform Configuration Files

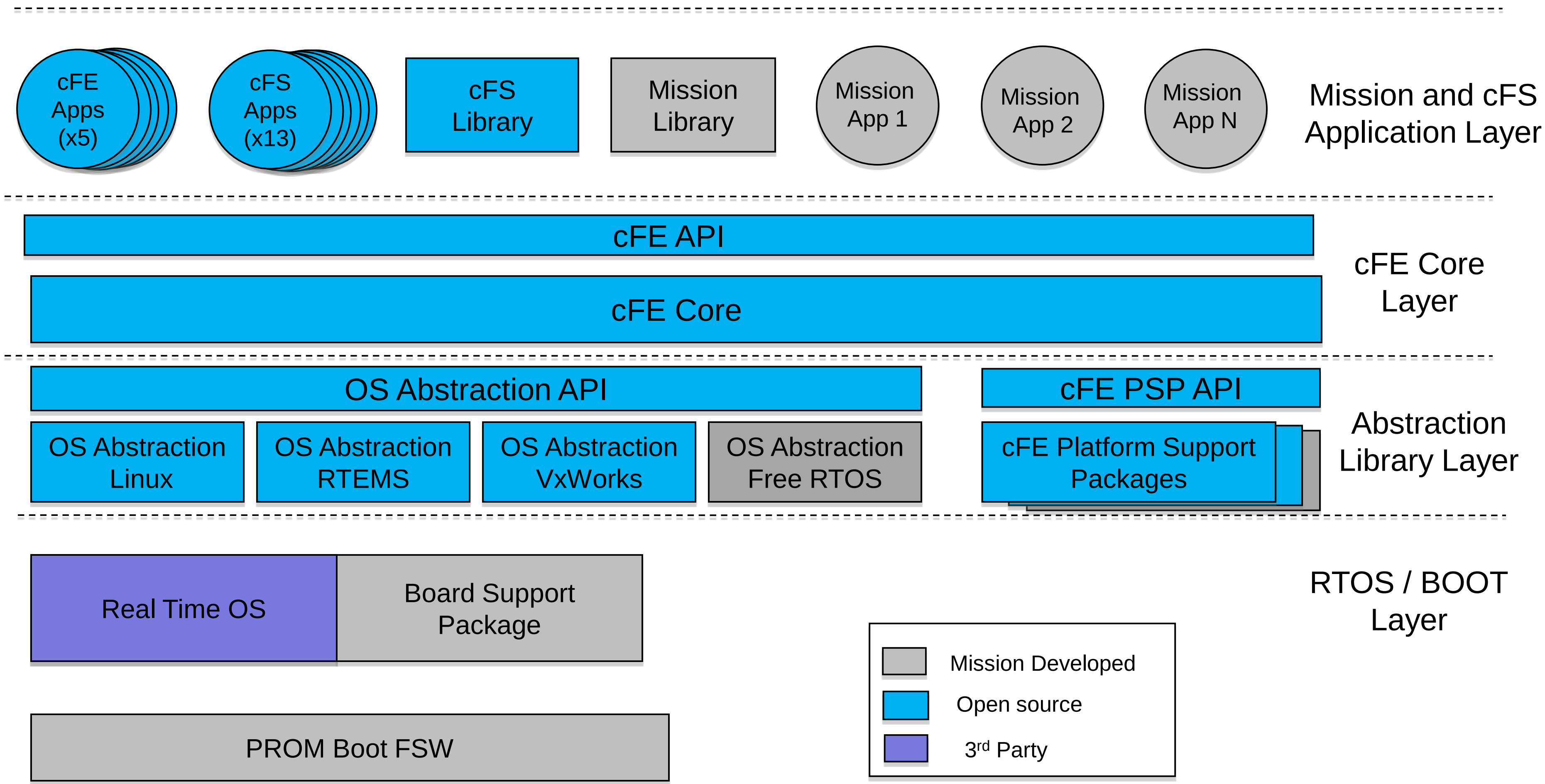
File	Purpose	Mission or Platform	Notes
cfe_mission_cfg.h	cFE core mission wide configuration	Mission	
cfe_platform_cfg.h	cFE core platform configuration	Platform	Most cFE parameters are here
cfe_msgids.h	cFE core platform message IDs	Platform	Defines the message IDs the cFE core will use on that Platform(CPU)
osconfig.h	OSAL platform configuration	Platform	
XX_mission_cfg.h	A cFS Application's mission wide configuration	Mission	Allows a single cFS application to be used on multiple CPUs on one mission
XX_platform_cfg.h	Application platform wide configuration	Platform	
XX_msgids.h	Application message IDs	Platform	
XX_perfids.h	Application performance IDs	Platform	

Architecture Layers

cFS Architecture Layers



cFS Software Layers and Components

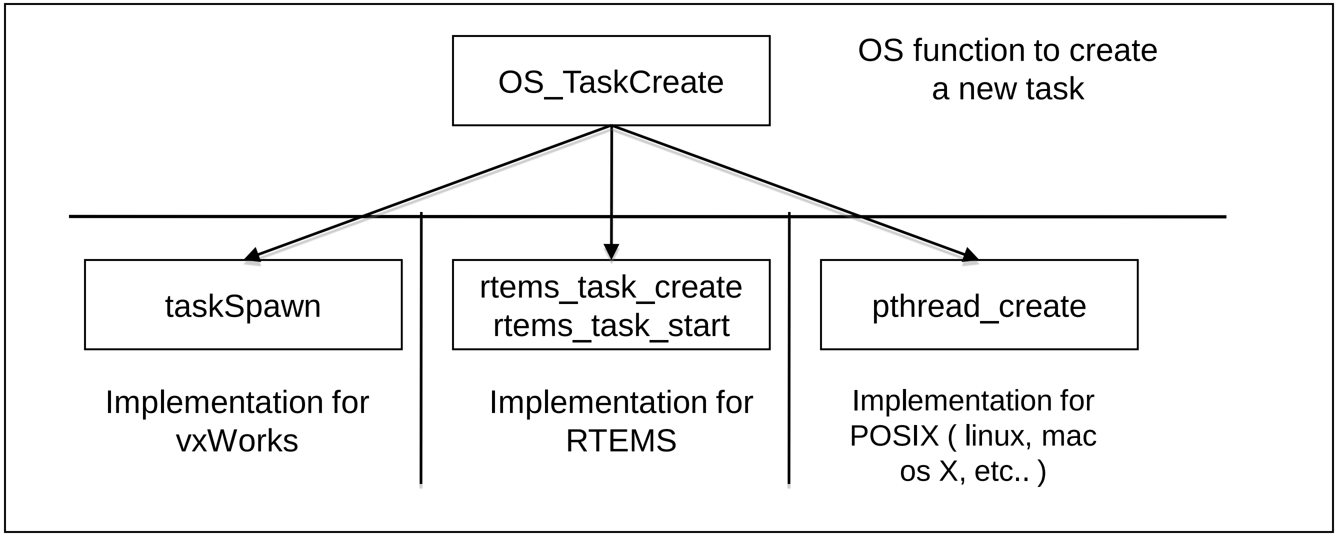


RTOS / Boot Layer

- **PROM Boot Software**
  - PROM resident software that does early initialization and bootstraps the Operating System
  - Provides ground based EEPROM/Flash loader
  - Keep it as simple as possible to minimize PROM changes
  - Commonly used Boot Software
    - RAD750 – BAE SUROM
    - Coldfire – Custom GSFC developed
    - LEON3 – uBoot – or Gaisler MKPROM
- **Real Time Operating System**
  - Pre-emptive priority based multi-tasking
  - Message Queues, Semaphores
  - Interrupt handling, Exception Handling
  - File systems, and shell
  - Supported Real Time Operating Systems
    - VxWorks
    - RTEMS
    - Linux
    - Linux with Xenomai Real-time extensions (In work)
    - ARINC 653 (Green Hills, and VxWorks-653)

Abstraction Library Layer
OSAL - 1

- The Operating System Abstraction layer (OSAL) is a small software library that isolates our Flight Software from the Real Time Operating System
- With the OS Abstraction Layer, flight software such as the Core Flight Executive can run on several operating systems without modification
  - Allows execution of FSW on simulators and desktop computers
- Current Implementations of the OSAL include:
  - RTEMS - Used on the RHCF 5208 Coldfire CPU
  - vxWorks - Used on RAD750
  - Linux / x86 - Used to run software on Desktop PC with Linux



Abstraction Library Layer
OSAL - 2

- **A standalone project, separate from the cFE**
  - The cFE is built on the OSAL to provide portability
- **Available as Open Source on NASA's Open Source Website**
  - http://opensource.gsfc.nasa.gov

Abstraction Library Layer
Platform Support Package

- **Platform Support Package (PSP)**
  - A Platform Support Package is all of the software that is needed to adapt the cFE Core to a particular Processor Card.
  - A Platform Support Package also includes all of the tool chain specific make rules and options
  - Each mission is expected to customize a Platform Support Package
- **Functions include:**
  - Startup code
  - EEPROM and Memory read, write, copy, and protection functions
  - Processor card reset functions
  - Exception handler functions
  - Timer functions
- **Current Implementations of the PSPs include:**
  - Desktop Linux for prototyping and Class “D”
  - Power PC MCP750 / RAD750 – vxWorks 6.x
  - Coldfire - RTEMS

cFE Core Layer Overview

- **A set of** ***mission independent, re-usable, core*** **flight software services, applications, and operating environment**
  - Layered architecture
    - Supports a variety of hardware platforms
  - Provides standardized Application Programmer Interfaces (API)
  - Supports and hosts flight software applications
    - *Applications can be added and removed at run-time (eases system integration and FSW maintenance*)
  - Supports software development for on-board FSW, desktop FSW development and simulators
  - Contains platform and mission configuration parameters that are used to tailor to a specific platform and mission.

Executive Services (ES)

Event Services (EVS)

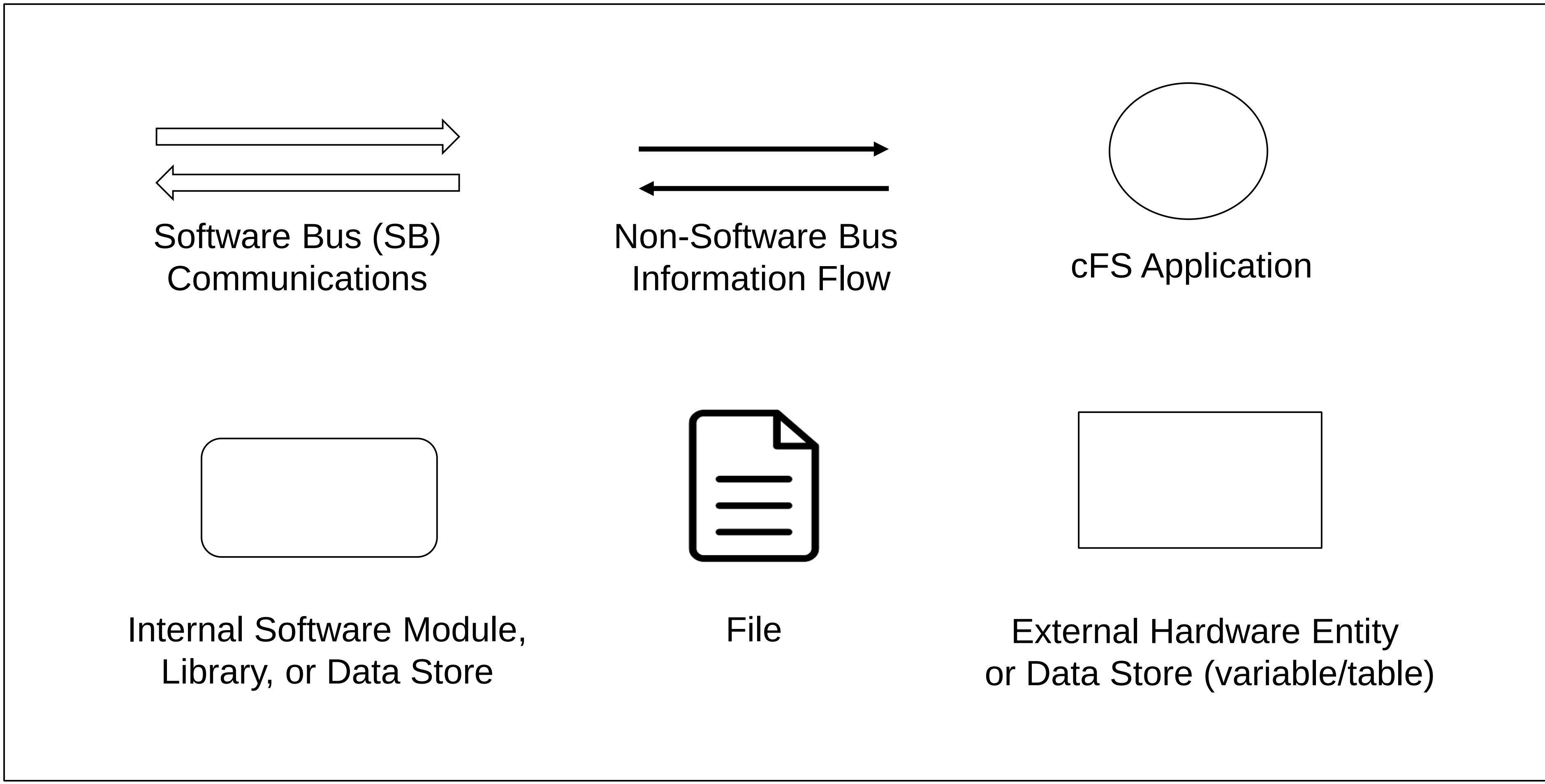
Software Bus (SB)

Table Services (TBL)

Time Services (TIME)

Open source release at, http://sourceforge.net/projects/coreflightexec/

Context Diagram Legend



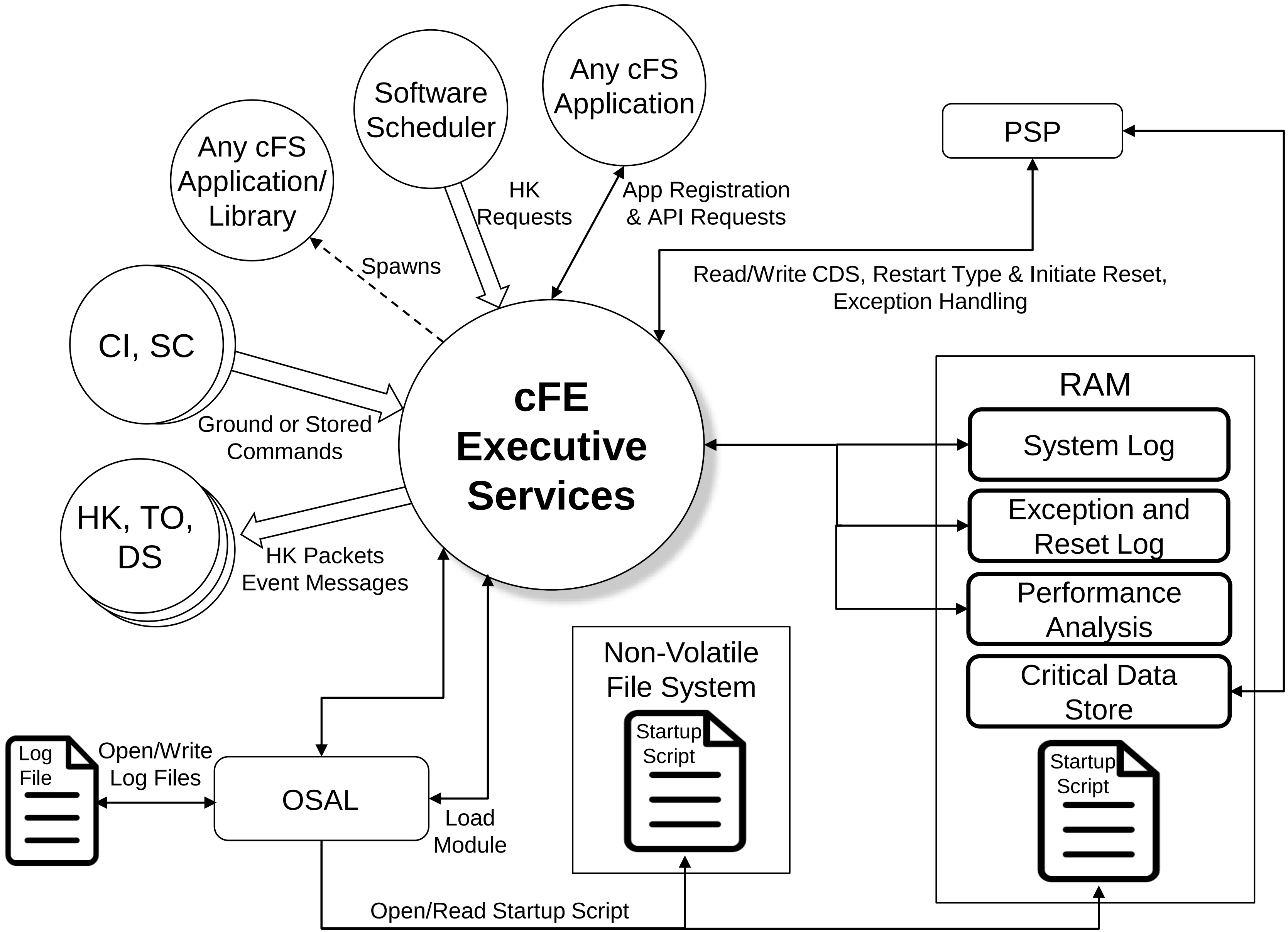
cFE Core Layer
Executive Services

- **Manages the startup of the cFE**
  - Power-on reset – cFE core, cFS Apps, file system, critical data store and logs are initialized
    - Decompresses cFS Applications
  - Processor reset – cFE core and cFE Apps are initialized. The following is preserved:
    - File system
    - Critical Data Store (CDS)
    - ES System Log
    - ES Exception and Reset (ER) log
    - Performance Analysis data
    - ES Reset info (i.e.reset type, boot source, number of processor resets)
    - Time Data (i.e. MET, SCTF, Leap Seconds)

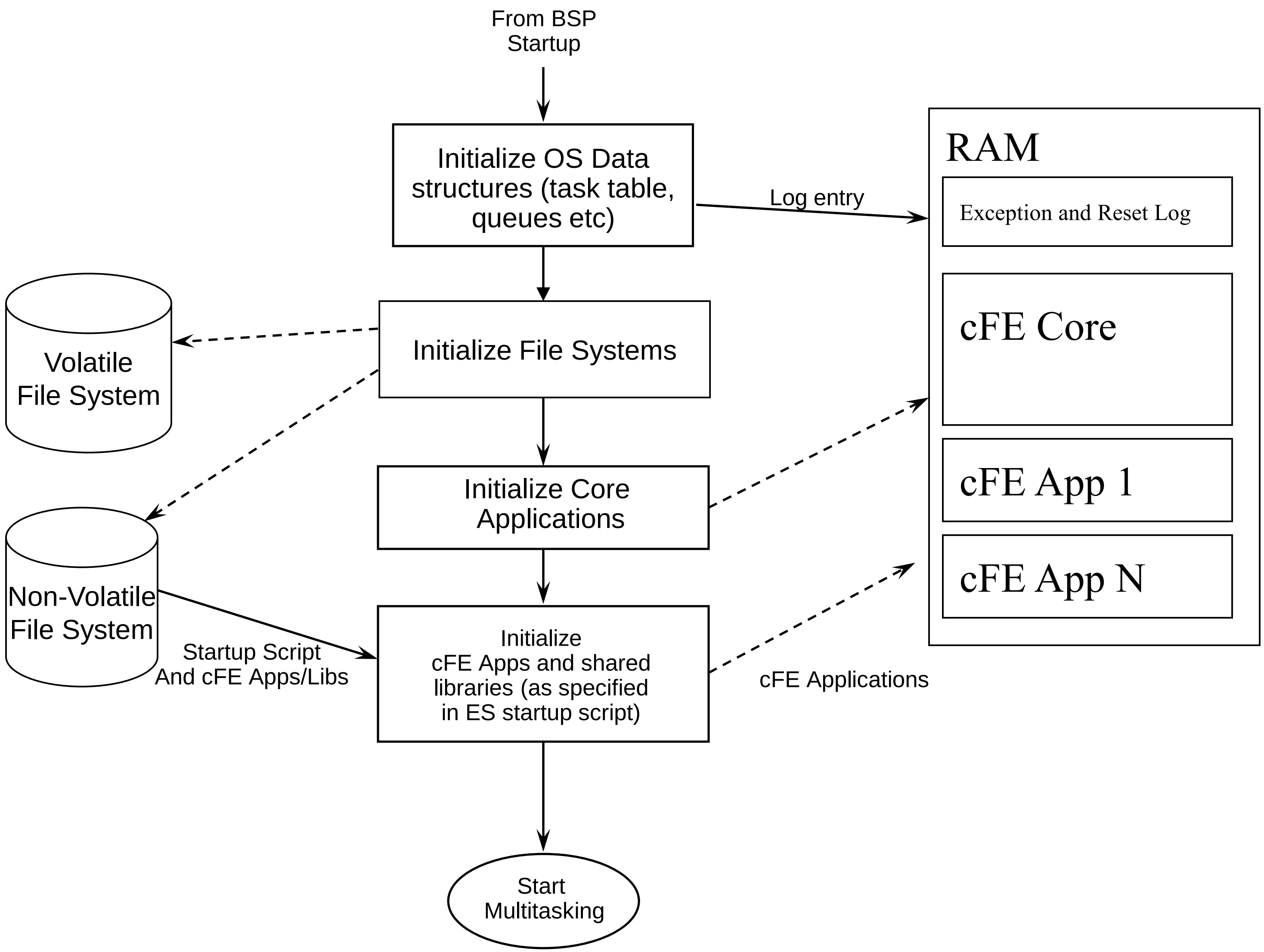
cFE Core Layer
Executive Services

- **Provides ability to start, restart and delete cFS Applications**
  - On startup
  - During runtime
- **Manages a Critical Data Store which can be used to preserve data (except in the case of a power-on reset)**
- **Provides ability to load shared libraries**
- **Provides support for device drivers**
- **Logs information related to resets and exceptions**
- **Manages a system log for capturing information and errors**
- **Provides Performance Analysis support**

Executive Services
Software Context Diagram



Executive Services Startup



The cFE core is started as one unit. The cFE Core is linked with the RTOS and support libraries and loaded into system EEPROM as a static executable.

Executive Services Restart

- **Restart cFE Core (and Applications)**
  - This is a full restart of the cFE Core
  - It is equivalent to the traditional Cold Restart
- **Restart Application**
  - This will effectively delete and start an Application
  - It can be used in response to
    - Exceptions
    - On-board FDC applications
    - Ground commands
  - Critical data can be stored in a Critical Data Store (CDS)

Executive Services APIs

Utility Functions	Purpose
CFE_ES_GetBlockInCDS	Allocate a block of space in the critical data store
CFE_ES_WriteToSysLog	Write to provided string to the System Log
CFE_ES_CalculateCRC	Calculate a data integrity value on a block of memory.

Critical Data Store (CDS) Functions	Purpose
CFE_ES_RegisterCDS	Allocates a block of memory in the Critical Data Store for a cFE Application
CFE_ES_CopyToCDS	Saves a block of data to the CDS
CFE_ES_RestoreFromCDS	Recover a block of data from the CDS

Memory Pool Functions	Purpose
CFE_ES_PoolCreate	Manages a memory pool created by an application
CFE_ES_GetPoolBuf	Gets a buffer from the memory pool created by CFE_ES_CreatePool
CFE_ES_PutPoolBuf	Releases a buffer from the memory pool

Performance Analysis Functions	Purpose
CFE_ES_PerfLogEntry	Entry marker for the performance analysis tool
CFE_ES_PerfLogExit	Exit marker for the performance analysis tool

Executive Services APIs

Application and Task Control Functions	Purpose
CFE_ES_GetResetType	Identifies the type of the last reset the processor most recently underwent
CFE_ES_ResetCFE	Perform a reset of the cFE Core and all of the cFE Applications
CFE_ES_RestartApp	Perform a restart of the specified cFE Application
CFE_ES_ReloadApp	Stops and then Starts a cFE Application from the specified file
CFE_ES_DeleteApp	Deletes a cFE Application
CFE_ES_ExitApp	Provides an exit point for a cFE Application's run loop
CFE_ES_RegisterApp	Register the cFE Application
CFE_ES_GetAppIDByName	Returns the cFE Application ID corresponding to the given cFE Application name
CFE_ES_GetAppID	Returns the cFE Application ID of the calling cFE Application
CFE_ES_GetAppName	Returns the cFE Application Name of the calling cFE Application
CFE_ES_GetTaskInfo	Returns info about the specified child task ID including Task name, Parent task etc.
CFE_ES_RegisterChildTask	Register a child task (note each cFE Application has a main task)
CFE_ES_CreateChildTask	Create a child task
CFE_ES_DeleteChildTask	Delete a child task
CFE_ES_ExitChildTask	Exits a child task

cFE Core Layer
Event Services

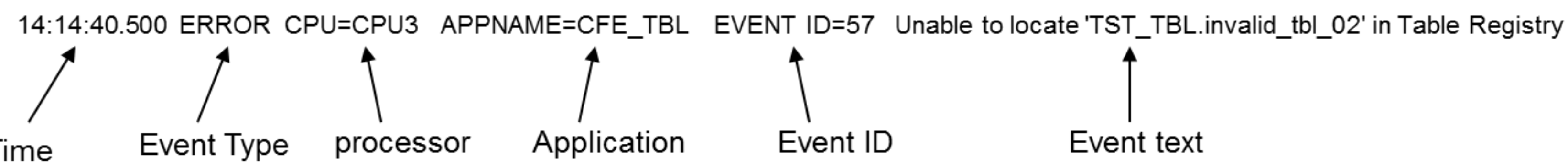
- **Provides an interface for sending asynchronous debug, informational, or error message telemetry to ground**
  - Provide a processor unique software bus event message containing the processor ID, Application ID, Event ID, timestamp, and the request-specified event data (text string including parameters)
  - Provide ability to send messages via hardware message ports
  - Provide ability to send long or short message format

- **Provide an interface for filtering event messages**
  - Provide event filtering via:
    - Event Filtering algorithm
    - Event Type (Debug, Information, Error, Critical)
    - Application
    - Application Event Type

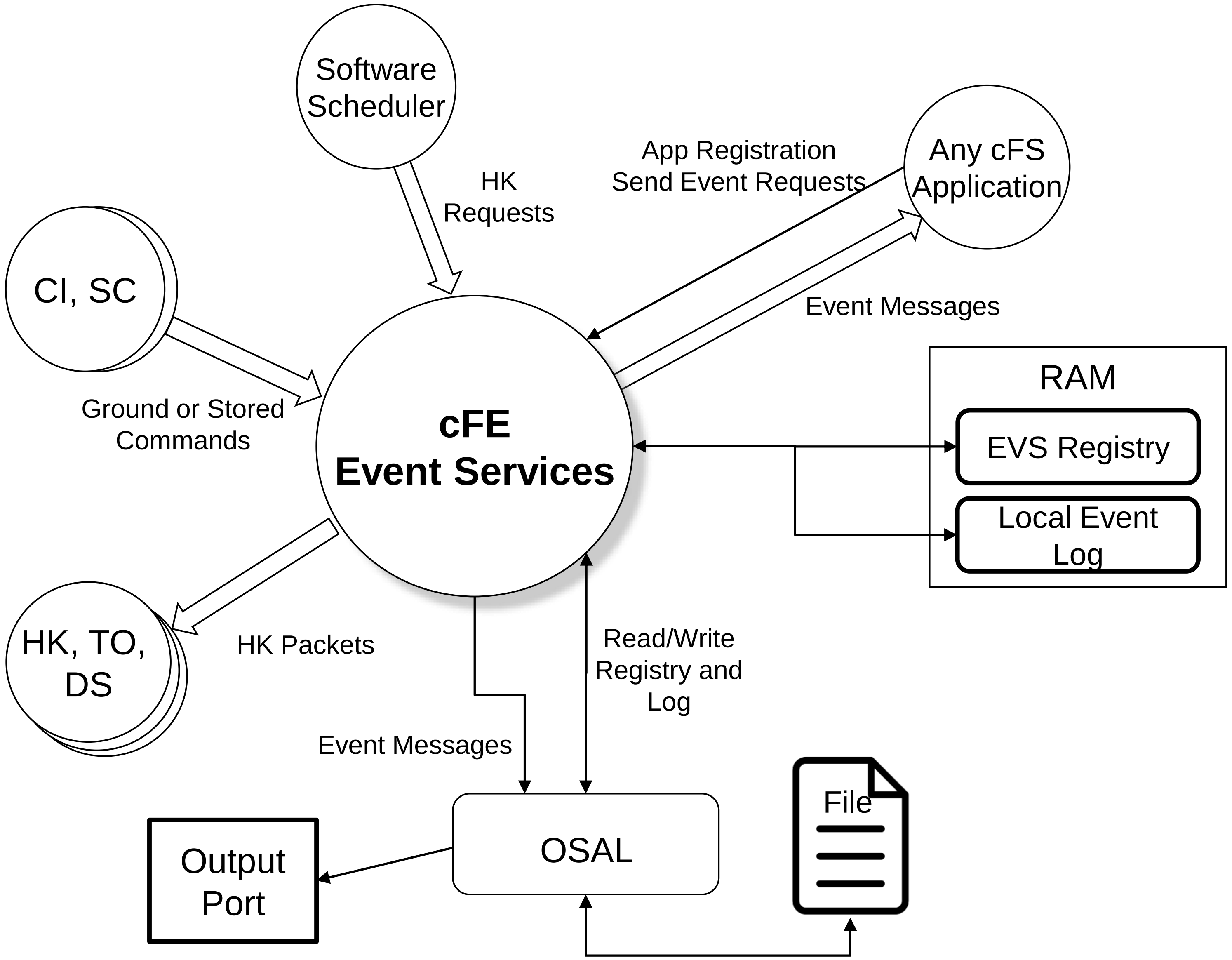
Event Services (EVS)

- Provide an interface for registering an application’s event filter masks, types, and type enable status
- Provide an interface for un-registering an application from using event services
- Provide an interface for enabling/disabling an application’s event filtering
- <optional> Provide an interface for logging event into a local event log

Example of an Event message:



Event Services Software Context Diagram

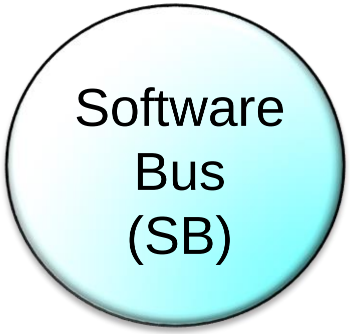


Event Services APIs

Application Functions	Purpose
CFE_EVS_Register	Register the application with event services. All Applications must register with EVS
CFE_EVS_Unregister	Cleanup internal structures used by the event manager
CFE_EVS_SendEvent	Request to generate a software event. Event message will be generated based on filter settings
CFE_EVS_SendEventWithAppID	Generate a software event as though it came from the specified cFE Application
CFE_EVS_SendTimedEvent	Generate a software event with a specific time tag
CFE_EVS_ResetFilter	Resets the calling application's event filter for a single event ID
CFE_EVS_ResetAllFilters	Resets all of the calling application's event filters

cFE Core Layer
Software Bus (SB)

- **Provides a portable inter-application message service**
- **Routes messages to all applications that have subscribed to the message**
  - Subscriptions are done at application startup
  - Message routing can be added/removed at runtime
- **Reports errors detected during the transferring of messages**
- **Outputs Statistics Packet and the Routing Information when commanded**



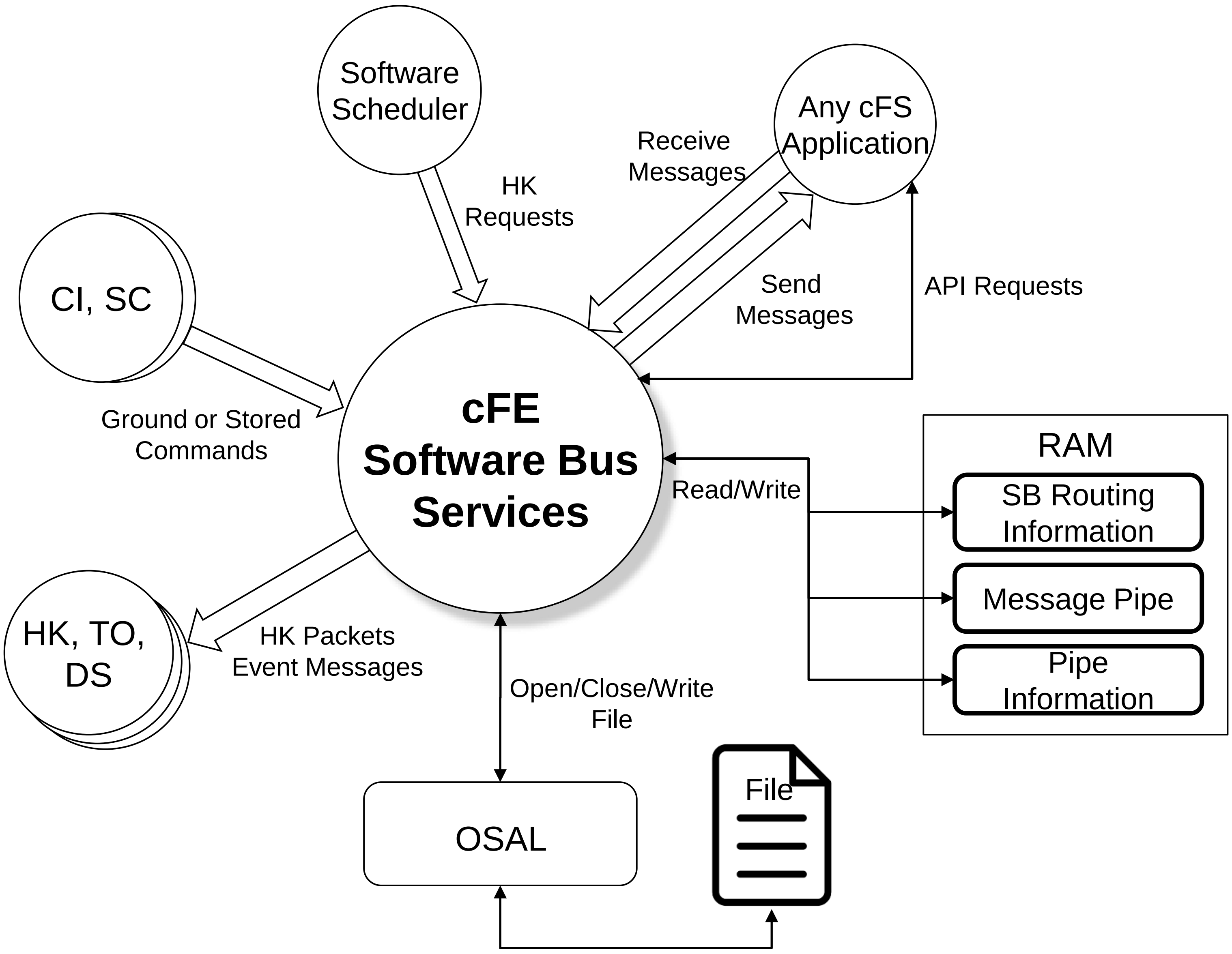
Software Bus Messages

- cFE abstracts the message format
- Implementation currently includes CCSDS format
- Software Bus provides functions to access message header (eg. CFE_SB_SetCmdCode, CFE_SB_SetMsgTime etc)

```
typedef struct{
    CCSDS_PriHdr_t          Pri;
    CCSDS_CmdSecHdr_t       Sec;
}CFE_SB_CmdHdr_t;

typedef struct{
    CCSDS_PriHdr_t     Pri;
    CCSDS_TlmSecHdr_t  Sec;
}CFE_SB_TlmHdr_t;
```

cFE Software Bus Services Software Context Diagram



cFE Software Bus APIs

Application Functions	Purpose
CFE_SB_CreatePipe	Creates and initializes an input pipe that the calling application can use to receive software bus messages
CFE_SB_DeletePipe	Deletes specified input pipe
CFE_SB_InitMsg	Initialize a buffer for a software bus message
CFE_SB_SubscribeEx	Adds the specified pipe to the destination list for the specified Message ID
CFE_SB_Subscribe	Same as CFE_SB_SubscribeEx except uses default Quality and Message Limit parameters
CFE_SB_SubscribeLocal	Same as CFE_SB_Subscribe except the subscription is local to the processor
CFE_SB_Unsubscribe	Removes specified pipe from destination list for the specified Message ID
CFE_SB_UnsubscribeLocal	Removes specified pipe from destination list for the specified Message ID (local subscription)
CFE_SB_SendMsg	Sends the specified Message to all subscribers
CFE_SB_RcvMsg	Retrieves the next message from the specified pipe - Can poll, pend with a timeout or pend forever - Data not copied. Function sets the Receiver’s pointer to the address of the actual message
CFE_SB_GetLastSenderId	Retrieve the application ID of the sender of the last message
CFE_SB_ZeroCopyGetPtr	Get a SB buffer for sending a Message via CFE_SB_ZeroCopySend
CFE_SB_ZeroCopySend	Send a Message that has been created via cFE_SB_ZeroCopyGetPtrbuffer
CFE_SB_ZeroCopyReleasePtr	Releases the Software Bus buffer created by cFE_SB_ZeroCopyGetPtrbuffer (On error condition)

cFE Software Bus APIs

Application Functions	Purpose
CFE_SB_MsgHdrSize	Provide access to construct and interpret software bus message fields (instead of accessing elements of the structure directly - portability)
CFE_SB_GetUserData	
CFE_SB_GetMsgId	
CFE_SB_SetMsgId	
CFE_SB_GetUserDataLength	
CFE_SB_SetUserDataLength	
CFE_SB_GetTotalMsgLength	
CFE_SB_SetTotalMsgLength	
CFE_SB_GetMsgTime	
CFE_SB_SetMsgTime	
CFE_SB_TimeStampMsg	
CFE_SB_GetCmdCode	
CFE_SB_SetCmdCode	
CFE_SB_GetChecksum	
CFE_SB_GenerateChecksum	
CFE_SB_ValidateChecksum	

cFE Core Layer
Table Services

- **Manages all cFS table images**

- **API provided for Applications to simplify Table Management**
  - Applications do not need to contain code for managing their own tables
  - Registering of Tables at run time allows for scalable system integration

- **Table of Tables (a.k.a - Table Registry) is populated at run-time eliminating cross coupling of Applications with flight executive at compile time**

Table
Services
(TBL)

cFE Core Layer
Table Services

- **All table updates are performed synchronously with the Application that owns the table to ensure table data integrity**

- **Tables can be shared between Applications**

- **Non-Blocking table updates allow tables to be used in Interrupt Service Routines**
  - Single buffer tables – uses shared inactive buffer for table updates (4)
  - Double buffer tables – uses dedicated inactive buffer for table updates

- **Common ground/user interface to all tables via Table Services**
  - Load/Dump/Validate and Activate tables
  - Table Registry Dump
  - Files are used to load and dump tables

Table Services
Software Context Diagram

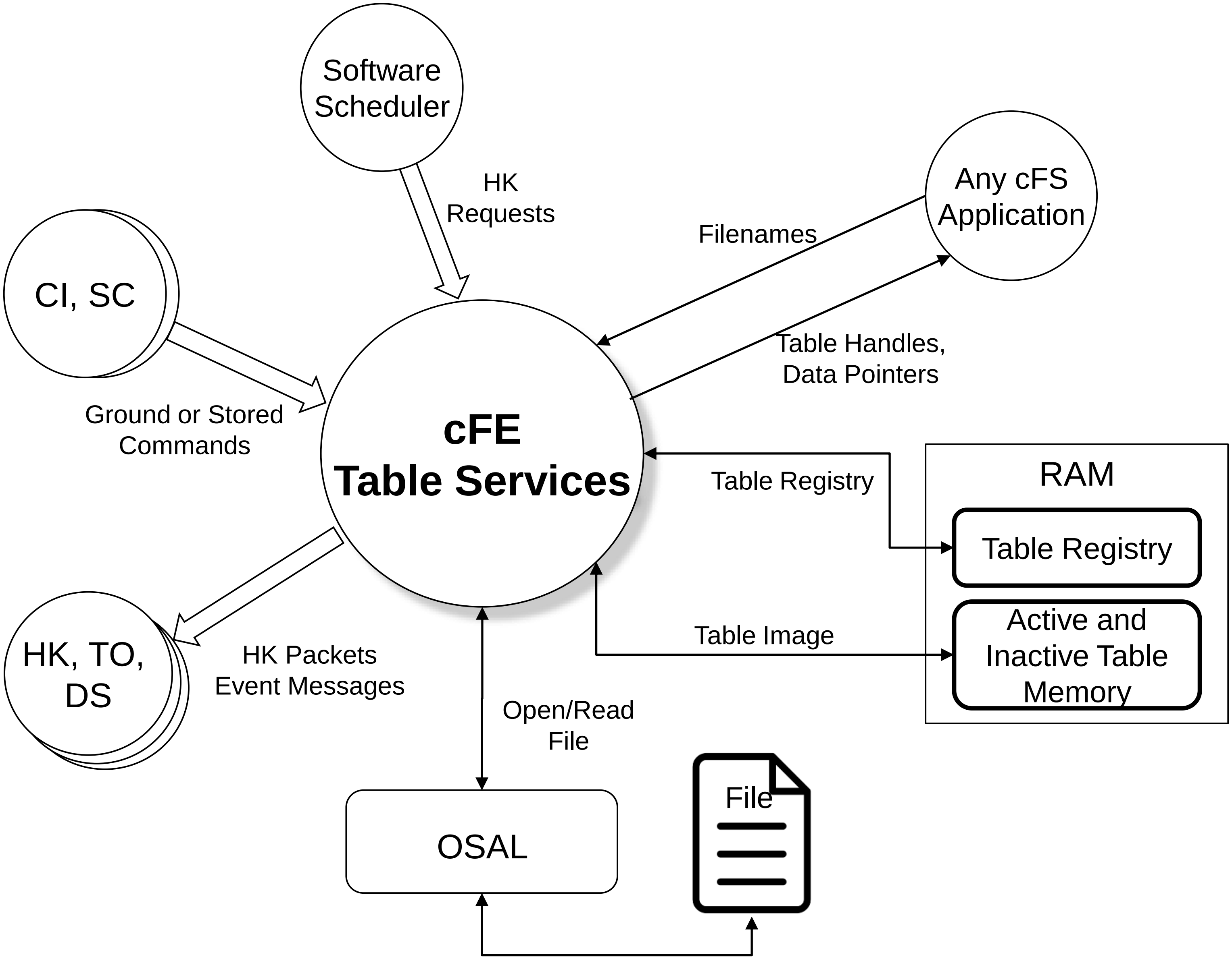


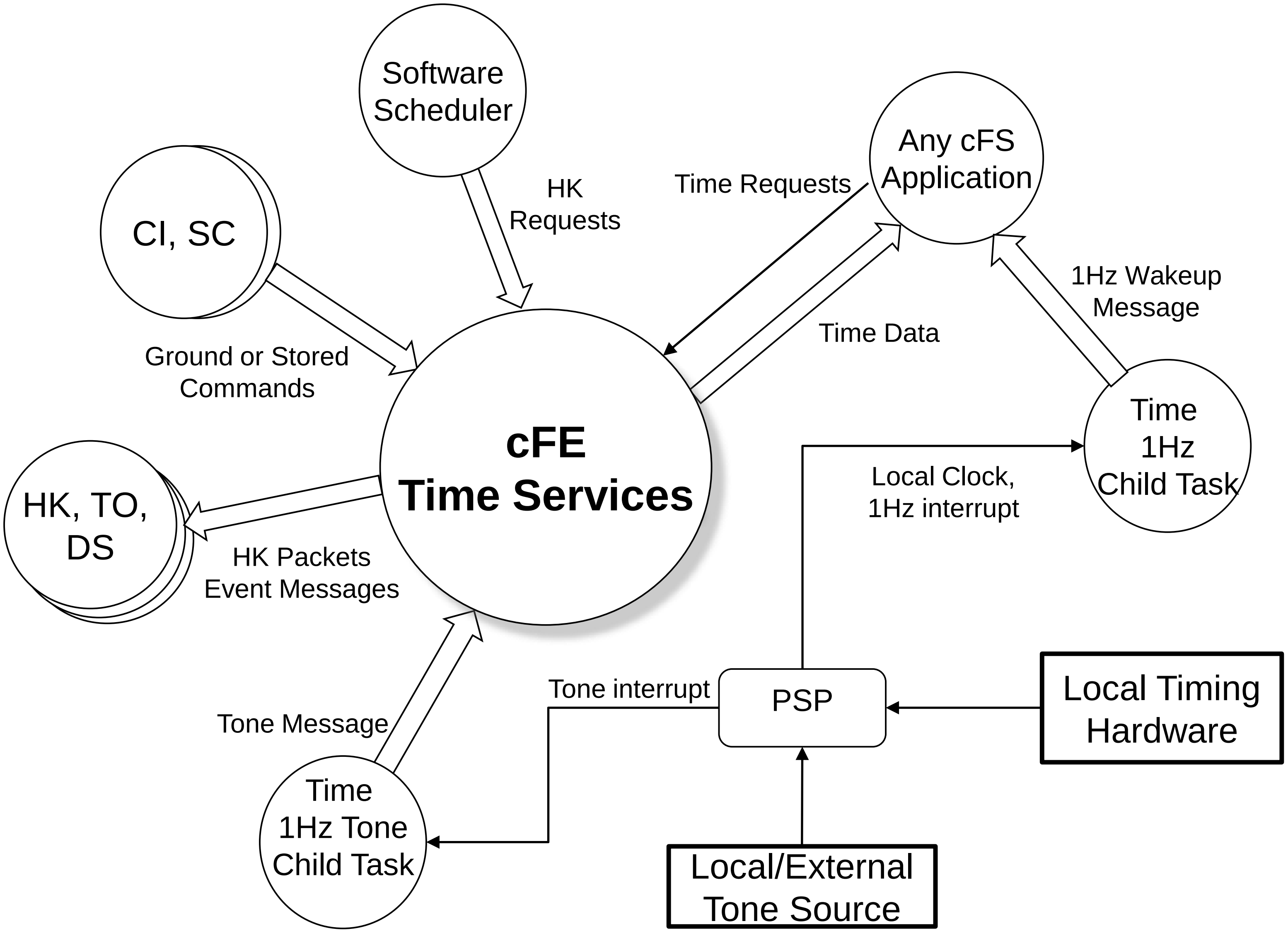
Table Services APIs

Application Functions	Purpose
CFE_TBL_Register	Registers a new table
CFE_TBL_Unregister	Unregister a table and release its resources
CFE_TBL_Load	Initialize or update the contents of a table from memory or a file
CFE_TBL_Share	Get a handle to a table that was created by another application
CFE_TBL_GetAddress	Get the address of a table (locks the table)
CFE_TBL_GetAddresses	Get the address of a collection of tables (locks the tables)
CFE_TBL_ReleaseAddress	Release a table address (unlocks the table). Must be done periodically by the cFE Application that owns the table in order to allow updates to the tables
CFE_TBL_ReleaseAddresses	Release an array of table address (unlocks the tables)
CFE_TBL_GetStatus	Returns the status on the specified table regarding validation or update requests
CFE_TBL_Validate	Performs the registered validation function for the specified table and reports the success/failure to the operator via Table Services Housekeeping Telemetry and Event Messages.
CFE_TBL_Update	Update table contents with new data if an update is pending
CFE_TBL_Manage	Performs routine actions to manage the specified table. This includes performing any necessary table updates or table validations
CFE_TBL_GetInfo	Provides information about the specified table including size, last time updated etc.

cFE Core Layer
TIME Services

- **Provides a user interface for correlation of spacecraft time to the ground reference time (epoch)**
- **Provides calculation of spacecraft time, derived from mission elapsed time (MET), a spacecraft time correlation factor (STCF), and optionally, leap seconds**
- **Provides a functional API for cFE applications to query the time**
- **Distributes a “time at the tone” command packet, containing the correct time at the moment of the 1Hz tone signal**
- **Distributes a “1Hz wakeup” command packet**
- **Forwards tone and time-at-the-tone packets**

Time Services
Software Context Diagram



Time Services APIs

Time Conversion Functions	Purpose
CFE_TIME_Sub2MicroSecs	Convert a sub-seconds count to an equivalent number of microseconds
CFE_TIME_Micro2SubSecs	Convert a number of microseconds to an equivalent sub-seconds count
CFE_TIME_CFE2FSSeconds	Convert cFE seconds to File System Seconds
CFE_TIME_FS2CFESeconds	Convert File System seconds to cFE seconds

Basic Clock Functions	Purpose
CFE_TIME_GetTime	Get the current spacecraft time
CFE_TIME_GetUTC	Get the current UTC time
CFE_TIME_GetTAI	Get the current TAI time
CFE_TIME_MET2SCTIME	Converts MET to Spacecraft time
CFE_TIME_GetMET	Get the current value of the mission-elapsed time
CFE_TIME_GetMETseconds	Get the current seconds count of the mission-elapsed time
CFE_TIME_GetMETsubsecs	Get the current sub-seconds count of the mission-elapsed time
CFE_TIME_GetSTCF	Get the current value of the spacecraft time correction factor (STCF)
CFE_TIME_GetLeapSeconds	Get the current value of the leap seconds counter
CFE_TIME_GetClockState	Get the current state of the spacecraft clock
CFE_TIME_GetClockInfo	Get clock information

Time Services APIs

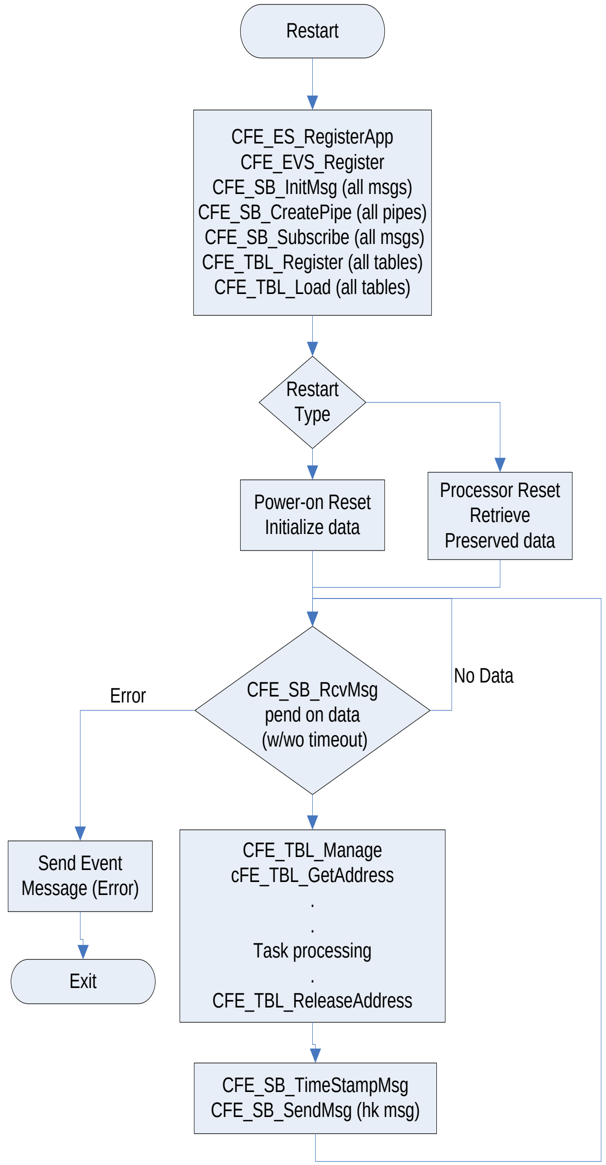
Time Manipulation Functions	Purpose
CFE_TIME_Add	Add two time values
CFE_TIME_Subtract	Subtract one time value from another
CFE_TIME_Compare	Compare two time values
CFE_TIME_Print	Print a time value as a string

External Time Sources	Purpose
CFE_TIME_ExternalTone	Latch the local time at the 1Hz tone signal
CFE_TIME_ExternalMET	Provide the MET from an external source
CFE_TIME_ExternalGPS	Provide the time from an external source that has data common to GPS receiver
CFE_TIME_ExternalTime	Provide the time from an external source that measures time relative to a known epoch

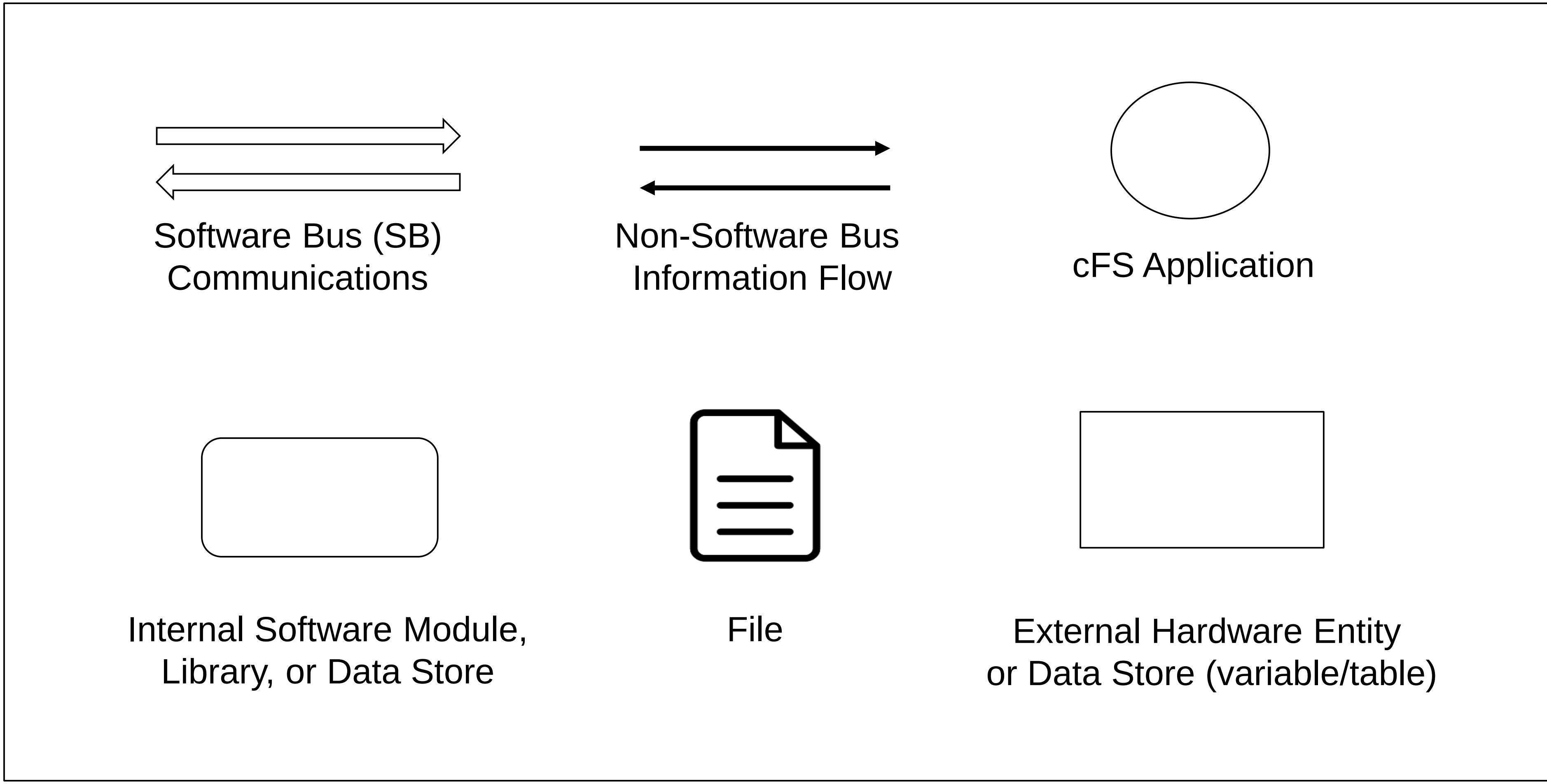
cFS Applications

Application	Function
CFDP	Transfers/receives file data to/from the ground
Checksum	Performs data integrity checking of memory, tables and files
Command Ingest Lab	Accepts CCSDS telecommand packets over a UDP/IP port
Data Storage	Records housekeeping, engineering and science data onboard for downlink
File Manager	Interfaces to the ground for managing files
Housekeeping	Collects and re-packages telemetry from other applications.
Health and Safety	Ensures that critical tasks check-in, services watchdog, detects CPU hogging, and calculates CPU utilization
Limit Checker	Provides the capability to monitor values and take action when exceed threshold
Memory Dwell	Allows ground to telemeter the contents of memory locations. Useful for debugging
Memory Manager	Provides the ability to load and dump memory.
Software Bus Network	Passes Software Bus messages over various “plug-in” network protocols
Scheduler	Schedules onboard activities via (e.g. HK requests)
Scheduler Lab	Simple activity scheduler with a one second resolution
Stored Command	Onboard Commands Sequencer (absolute and relative).
Telemetry Output Lab	Sends CCSDS telemetry packets over a UDP/IP port

Example cFS Application Template



Context Diagram Legend



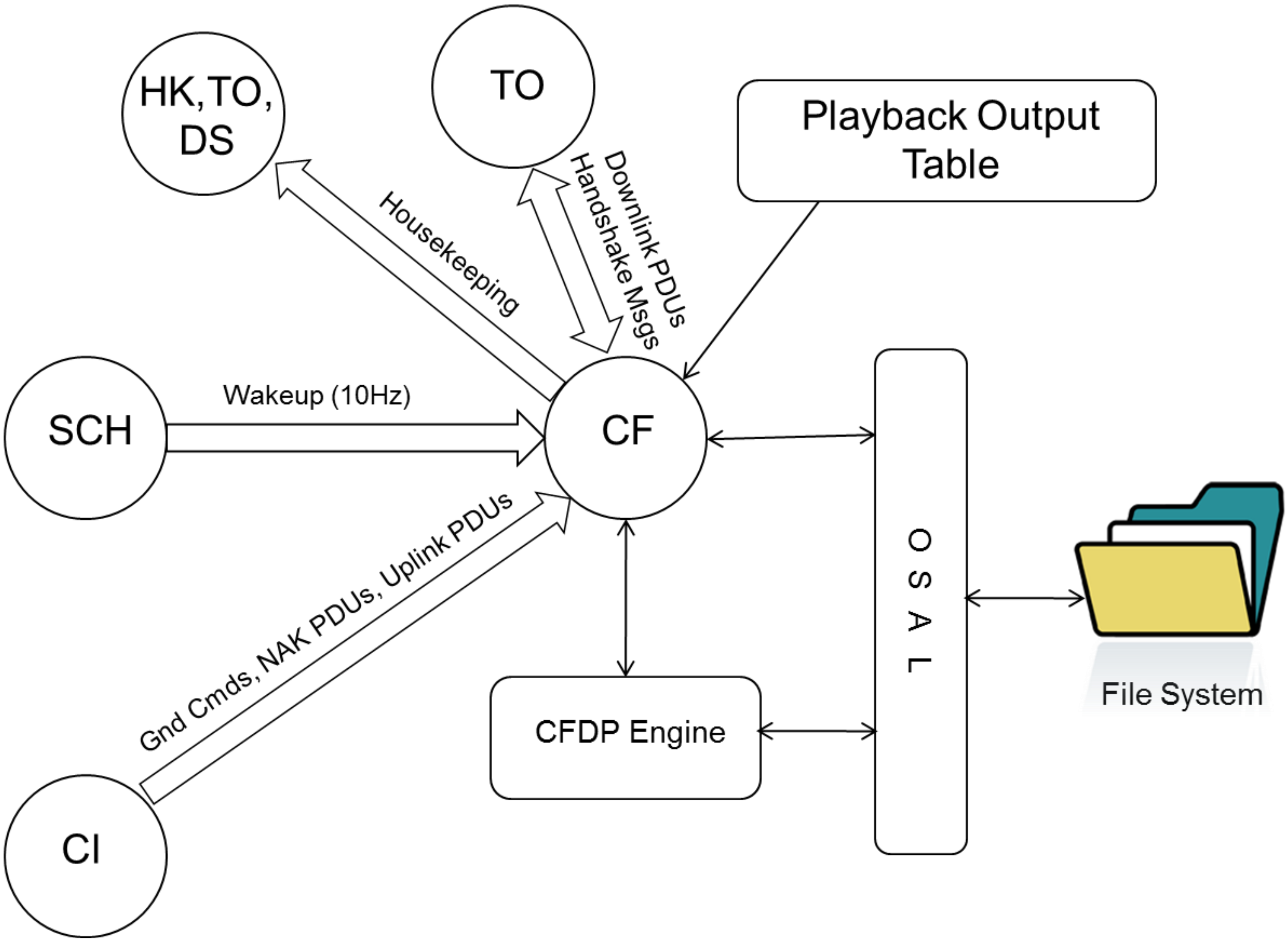
CFDP (CF)

CF Overview

- **Transmits and receives files to and from the ground**
  - Typically interfaces to ground through CI and TO applications

- **Utilizes CCSDS File Delivery Protocol (CFDP)**

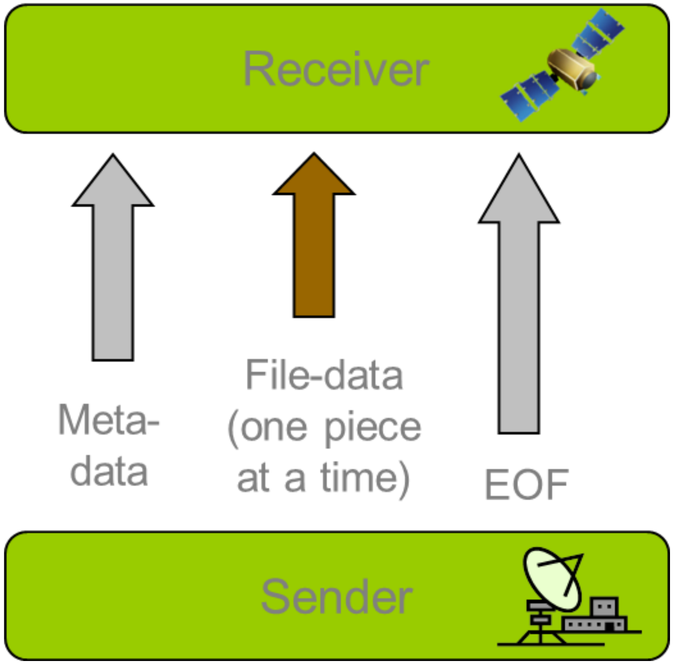
CF Context Diagram



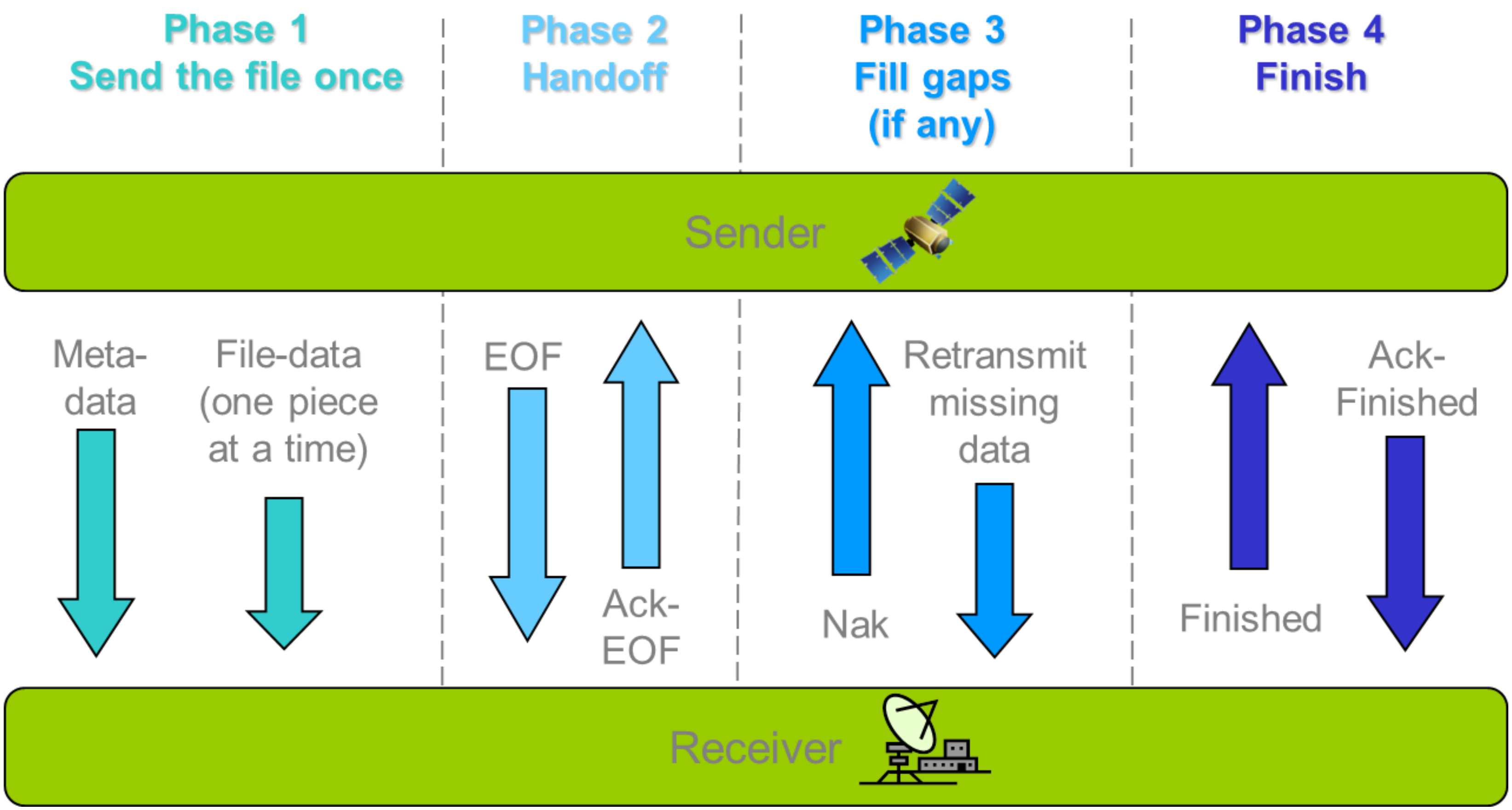
CF CFDP Engine

- Provides compile-time and run-time configuration parameters
- Packages file data and protocol messages in PDUs as defined in CCSDS Blue Book CCSDS 727.0-B-4
- Capable of transmitting and receiving class 1(Unreliable) or class 2(Reliable) transfers
- Handles simultaneous transactions
- Provides “Indications” to inform the CF application of significant occurrences
- Receives “Put Requests” to start file transfer
- Verifies file checksum for class 1 and 2

CF CFDP Class 1 Uplink Example



CF Class 2 Playback Example



CF File Uplink

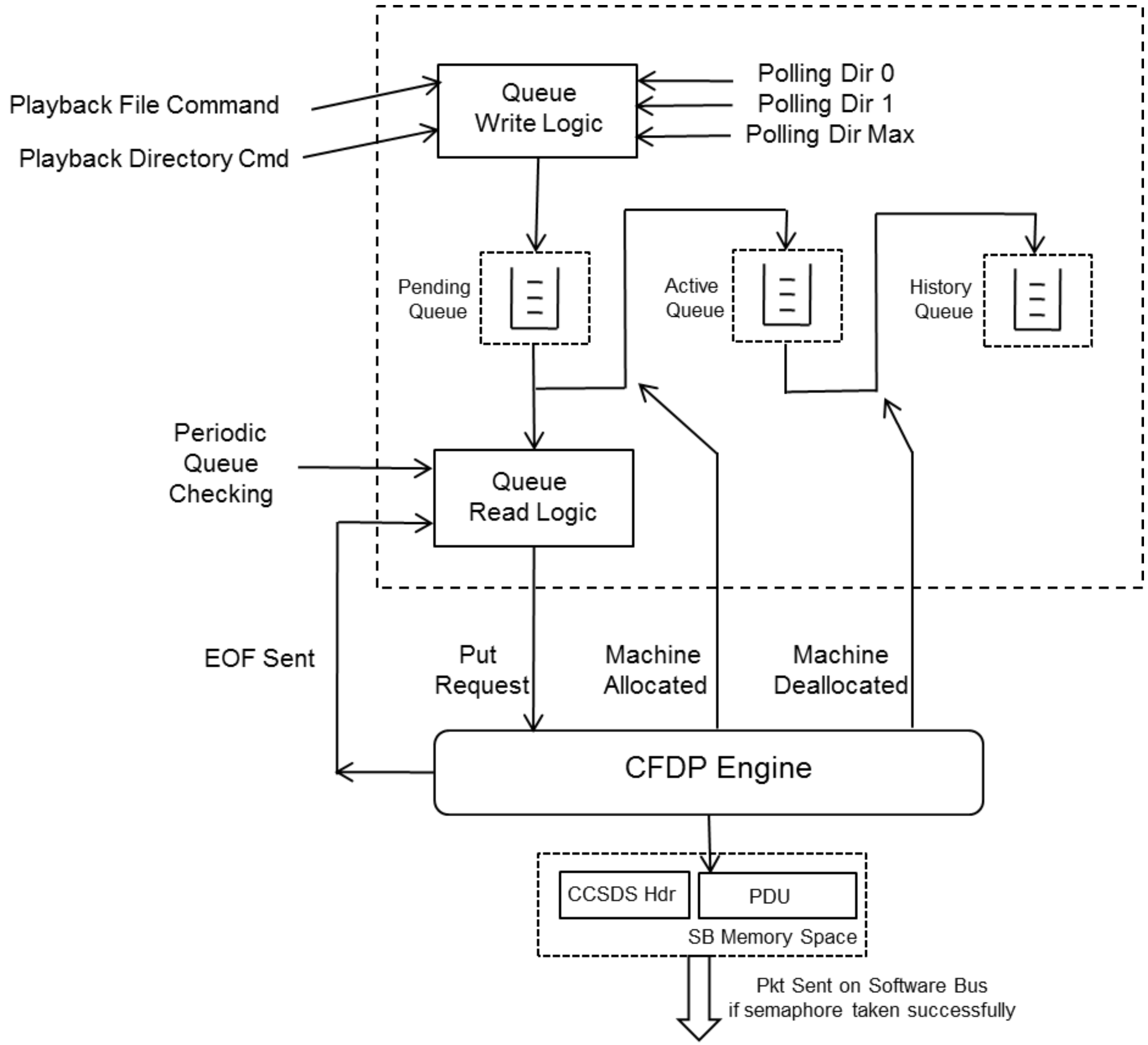
- **Subscribes to Uplink PDUs. MsgId is specified in the configuration table**
  - CF application does not do much processing in the uplink direction. The engine does most of the work.

- **Receives PDUs wrapped in CCSDS packet from Command Ingest application (CI).**

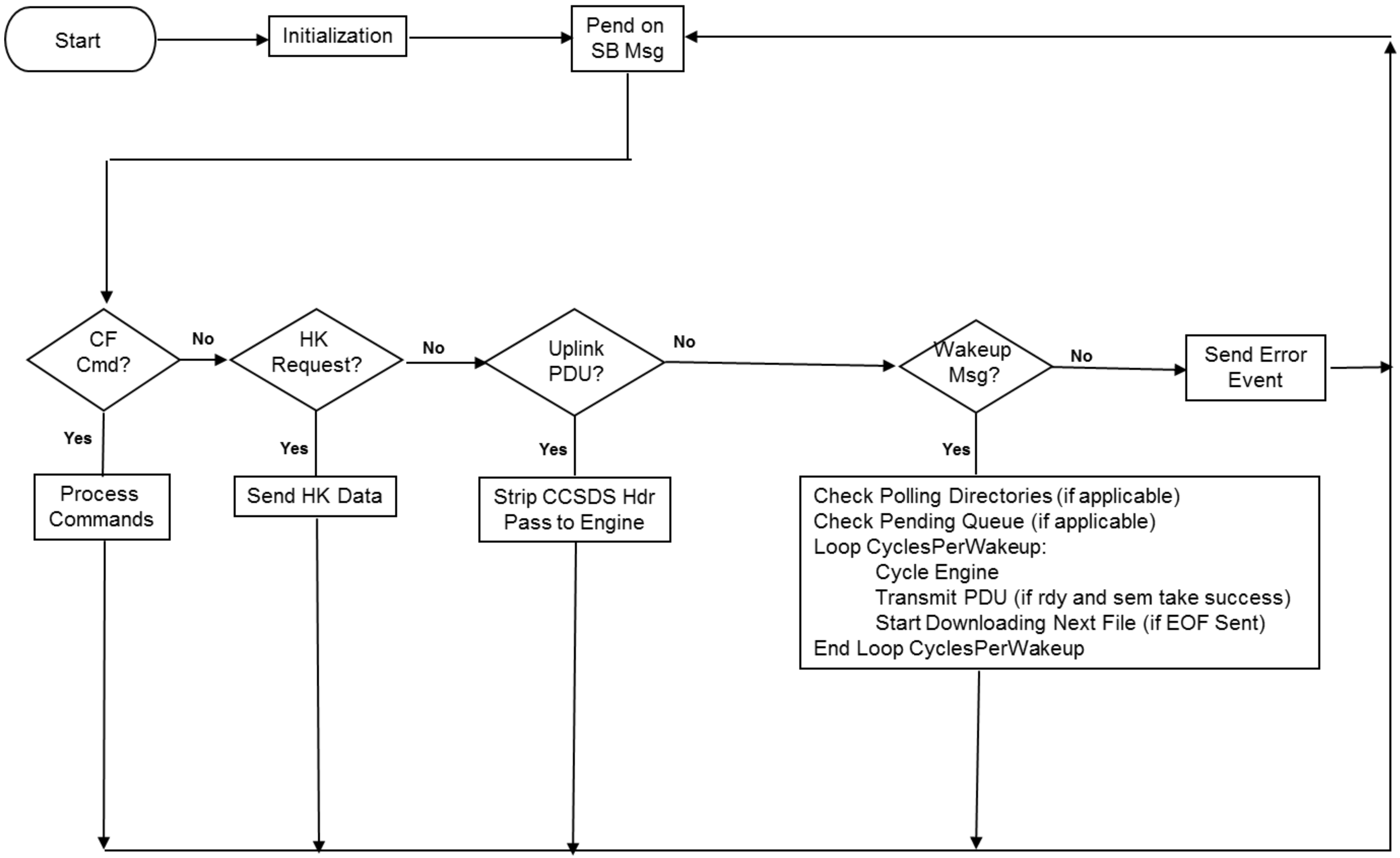
- **Strips away the CCSDS header and passes the raw PDU to the engine.**

- **Keeps track of uplinked files through the use of an active queue and a history queue**

CF – File Playback



CF Flow Control



CF Compile Time Configuration

- Max Simultaneous Transactions
- Pipe Name and Depth
- Max File Data in Playback PDU
- Max File Data in Uplink PDU
- Engine Temp File Prefix
- Configuration Table
- Configuration Table Filename
- Max Restricted Directories
- Max Playback Channels
- Max Polling Directories Per Channel
- Memory Pool Bytes
- Default Queue Info Filename

CF Commands - 1

Command	Description	Parameters
Noop	Increment command counter, display CF version number	None
Reset Counters	Reset one or all – command, fault, uplink, downlink counters	None
Playback File	Adds file to playback pending queue.	class, channel preserve, priority, SrcFilename, DstFilename
Playback Directory	Adds files in a given directory to playback pending queue.	class, channel preserve, priority, SrcDir, DstDir
Purge Queue	Purges the playback pending queue	channel
Write Queue	Writes the queue contents to a file.	type (uplink, playback), channel, queue, path/filename
Write Active Trans	Writes the transaction information (for all active transactions) to a file.	path/filename
Enable/Disable Dequeue	Enable/Disable Dequeue of playback pending queue.	channel
Dequeue Node	Dequeue a file on the pending or history queue.	type (uplink, playback), channel, queue, path/filename
Set Engine MIB Param	Set the engine configuration parameter specified in the command.	The configuration parameter to be set, value
Get Engine MIB Param	Display the given engine parameter in an event.	The configuration parameter to be displayed
Dump Config Params	Displays entire configuration contents via tlm packet. Includes run-time and compile-time.	None

CF Commands - 2

Command	Description	Parameters
Suspend Transaction	Pauses timers and counters for a single (or all) transaction(s)	String formatted as SrcEntityId_TransSeqNum (ex.0.24_3) or filename or “all”
Resume Transaction	Resume timers and counters for a single (or all) transaction(s)	String formatted as SrcEntityId_TransSeqNum (ex.0.24_3) or filename or “all”
Cancel Transaction	Cancels a single (or all) transaction(s)	String formatted as SrcEntityId_TransSeqNum (ex.0.24_3) or filename or “all”
Abandon Transaction	Abandons a single (or all) transactions(s)	String formatted as SrcEntityId_TransSeqNum (ex.0.24_3) or filename or “all”
Freeze	Pauses timers and counters for all transactions	None
Thaw	Resumes timers and counters for all transactions	None
Enable/Disable Polling Directory	Enable or disable polling directory	Channel, poll, directory number (get number from config table)
Set Poll Directory Param	Change class, priority, preserve, SrcPath, or DstPath of given polling directory	Channel, poll directory number, class, priority, preserve, source pathname, destination pathname
Send Transaction Diag	Send diagnostic packet for a single transaction	String formatted as SrcEntityId_TransSeqNum (ex.0.24_3) or filename

CF Commands – 3

Command	Description	Parameters
Kickstart	Start the transmission of the next file on the pending queue	Channel
Quick Status	Display high level status of the specified transaction	String formatted as SrcEntityId_TransSeqNum (ex.0.24_3) or filename
GiveTake	Adjust the handshake semaphore in the unexpected case that the semaphore value lost or gained a count when viewed during idle time	Channel
Auto Suspend Enable	enable or disable auto suspend mode	0 to disable and 1 to enable

CF Housekeeping Telemetry Message - 1

Telemetry Point	Description
Command Counter	Commands executed successfully
Command Error Counter	Commands that failed to execute
Memory In Use	Number of queue node bytes in use
Peak Memory In Use	Peak queue node bytes in use
Max Memory Needed	Memory needed if all queues were full
Memory Allocated	Memory allocated for queue nodes
Queue Nodes Allocated	Number of queue nodes currently allocated
Queue Nodes Released	Number of queue nodes returned to heap
Num Uplink PDUs Received	Number of uplink PDUs received
Num Files Uplinked Successfully	Number of uplink transactions succeeded
Num files failed uplink	Number of uplink transactions failed
Num uplink files in progress	Number of uplink transactions in progress
Last file uplinked	Filename of last file uplinked
Positive ACK Limit Counter	Number of ack timeout faults
Keep Alive Limit Counter	Number of keep alive faults
Invalid Transmission Mode Counter	Number of Inval transmission mode faults
FileStore Rejection Counter	Number of filestore rejection faults
File Checksum Failure Counter	Number of checksum failure faults
Filesize Error Counter	Number of filesize error faults

CF Housekeeping Telemetry Message - 2

Telemetry Point	Description
NAK Limit Counter	Number of NAK limit faults
Inactivity Counter	Number of inactivity faults
Invalid File Structure Counter	Number of invalid file structure faults
Suspend Request Counter	Number of suspend requests
Cancel Request Counter	Memory cancel requests
Flight Entity ID	Flight Entity ID
Frozen/Thawed Status	Transactions frozen or thawed
Machines Allocated by Engine	Number of machines allocated
Machines Deallocated	Number of machines deallocated
Frozen Partners	Any partners frozen — yes/no
Active Playback Files	Number of active playback files
Active Uplink Files	Number of active uplink files
Total Files Sent	Number of playback files sent
Total Files Received	Number of uplink files received
Total Transactions Frozen	Number of transactions frozen
Total Transactions Suspended	Number of transactions suspended
Total unsuccessful files sent	Number of unsuccessful playback files
Total unsuccessful files received	Number of unsuccessful uplink files

CF Housekeeping Telemetry Message - 3

Channel Telemetry (repeated for each channel)

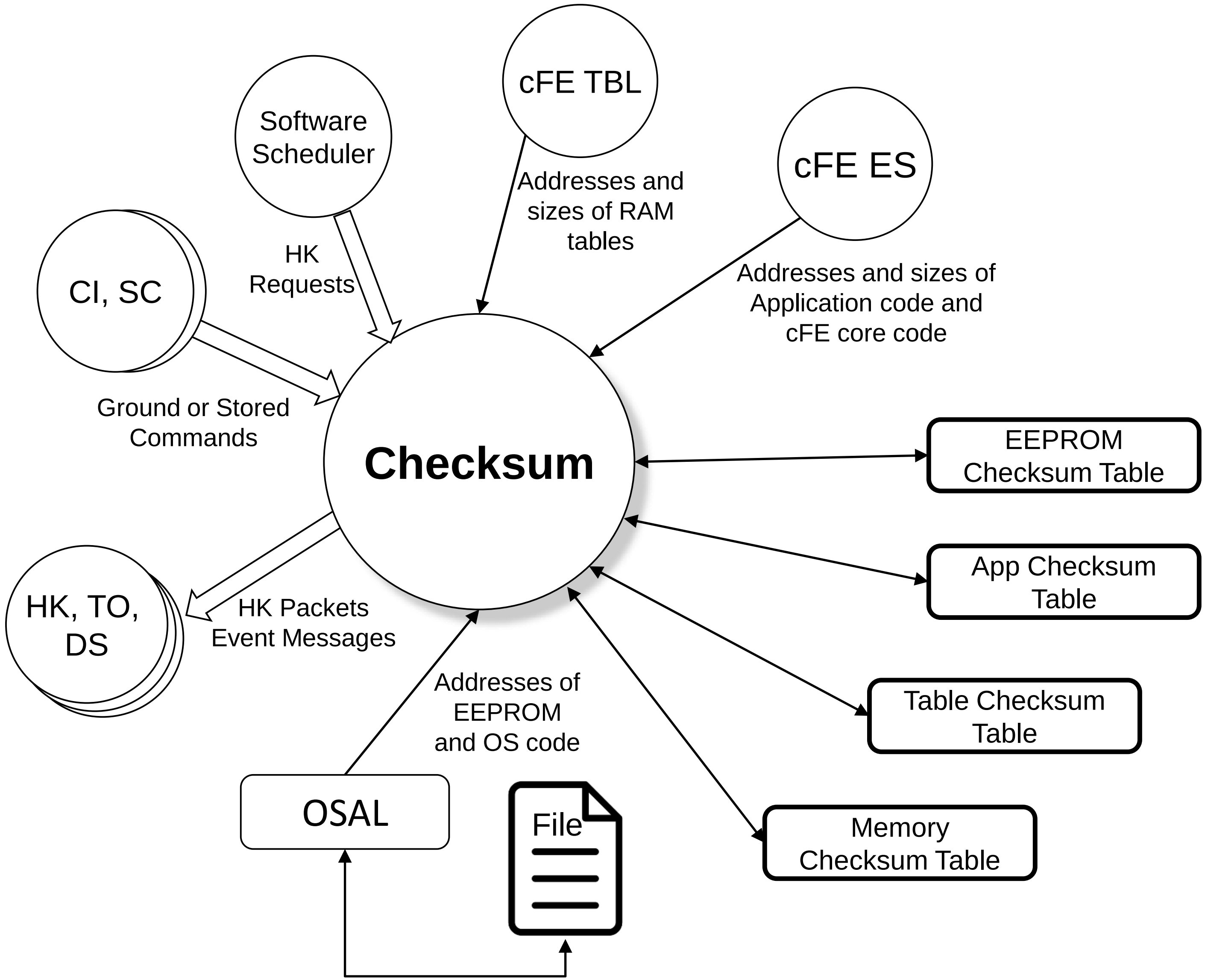
Telemetry Point	Description
PDUs Sent	Number of PDUs sent
Files Sent	Number of files sent
Files Sent successfully	Number of files sent successfully
Files Sent unsuccessfully	Number of files sent unsuccessfully
Files on Pending Queue	Number of files on pending queue
Files on Active queue	Number of files on active queue
Files on History queue	Number of files on history queue
Naks Received	Number of NAKs received
Dequeue Enable State	Pending queue, dequeue state
Poll Directory Enable State	One flag for each polling directory
Red Light Counter	Number of times TO has denied the request to send a pdu
Green Light Counter	Number of times TO has accepted the request to send a pdu
Poll Directory Check Counter	Number of times poll directories checked
Pending Queue Checked Counter	Number of times pending queue checked

Checksum (CS)

CS Overview

- **Monitors the static code/data specified by the users and the OS and cFE code segments.**
- **Uses four different user defined tables**
  - Table of Apps to be checkummed
  - Table of Tables to be checksummed
  - Table of EEPROM to be checksummed
  - Table of other memory areas (“Memory”) to be checksummed
- **Reports all checksum miscompares as errors.**
- **Scheduled to wakeup on a 1Hz schedule**
- **Byte-limited per cycle to prevent CPU hogging**

CS Context Diagram



CS Checksum Activities

- **Background Cycle**
  - On wake-up from a “background cycle” command, CS continues checksum calculations through the four definitions tables, OS code segment, and cFE core.
- **Recompute**
  - On receipt of a “recompute” command, CS spawns a child task to compute a new baseline checksum for the selected area
  - Only one child task may be active at a time
    - Includes One-Shot
- **One Shot**
  - On receipt of a “one-shot” command, CS spawns a child task to compute a checksum on the specified memory area.
  - Only one child task may be active at a time
    - Includes Recomputes

CS Checksum Algorithm

- **The term 'checksum' is historical and does not actually mean we are using a checksumming algorithm, as it has been proven they are not safe enough.**

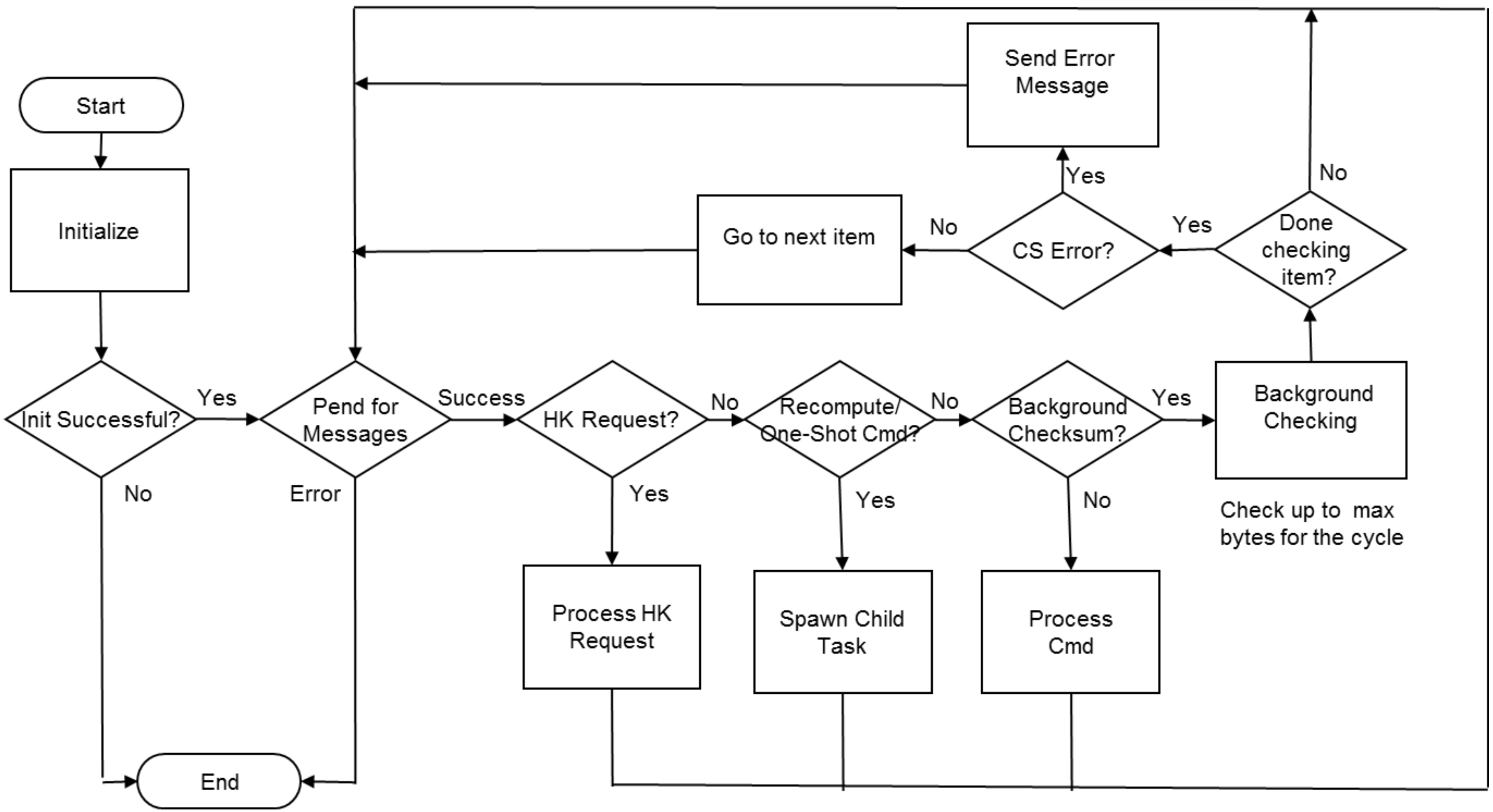
- **The algorithm that CS will use will be a Cyclical Redundancy Check (CRC) algorithm. It will be handled by a cFE ES function which specifies 8, 16, or 32 bit polynomial.**

- **By default, CS will use the cFE default CRC algorithm, but can be changed via a configuration parameter**

CS Checksum Regions

- **EEPROM**
  - Everything in EEPROM (file system, OS, bootstrap, etc)
    - Split up by user-defined regions
- **RAM**
  - OS code segment
  - cFE core code segment
  - Application code segments
  - Tables
  - User defined memory segments

CS Flow Control



CS Telemetering Checksums

- **CS maintains a dump-only checksum working table for each checksum region defined by table in CS**
  - Updates checksum results for each checksum region on each checksum cycle
  - Users can obtain current checksum results by performing a table dump via a Table Services command

CS Configuration Parameters

Parameter	Description	Default Value
Default EEPROM Table Name	--	/cf/apps/cs_eepromtbl.tbl
Default Memory Table Name	--	/cf/apps/cs_memorytbl.tbl
Default Tables Table Name	--	/cf/apps/cs_tablestbl.tbl
Default Apps Table Name	--	/cf/apps/cs_apptbl.tbl
Pipe Depth	Command pipe depth	12
Max # of EEPROM Entries	Maximum number of entries in the table to checksum	16
Max # of Memory Entries	Maximum number of entries in the table to checksum	16
Max # of Tables Entries	Maximum number of entries in the table to checksum	24
Max # of Apps Entries	Maximum number of entries in the table to checksum	24
Default Bytes per Cycle	# of bytes checksummed in a single cycle	16384 (16KB)
Child Task Priority	1 is highest priority. Child cannot be higher than CS.	200
Child Task Delay	Delay to prevent CPU hogging.	1000 ms
Startup Timeout	Time for CS to wait for other apps to start	60000 ms
Mission Revision	Mission-level revision number	0

CS Commands

Command	Description
No-op	Increments the Command Accepted Counter and sends an info event message
Reset Counters	Initializes housekeeping counters to zero
Disable Checksumming	Stop CS background checking
Enable Checksumming	Restart background checking
OneShot Checksum	Start at given address, compute checksum over size
Cancel Oneshot checksum	If a one shot CS is in progress, stop it
Report Baseline of cFE Core	Reports the baseline of the cFE Core code segment
Recompute Baseline of cFE Core	Recomputes the baseline of the cFE Core code segment
Report Baseline of OS	Reports the baseline of the OS code segment
Recompute Baseline of OS	Recomputes the baseline of the OS code segment
Disable Checksumming for cFE Core	Stop background checking cFE Core code segment
Enable Checksumming for cFE Core	Restart background checking cFE Core code segment
Disable Checksumming for OS	Stop background checking OS code segment
Enable Checksumming for OS	Restart background checking OS code segment

CS EEPROM Commands

Command	Description
Get Region ID for EEPROM Address	Retrieves EEPROM table entry ID for region that covers given address
Recompute baseline for EEPROM Region	Recompute the baseline checksum for the given EEPROM region ID
Report Baseline for EEPROM Region	Sends event message with baseline checksum for given EEPROM region ID
Disable Checksumming for EEPROM Region	Stop background checking for the given EEPROM region ID
Enable Checksumming for EEPROM Region	Restart background checking for the given EEPROM region ID
Disable Checksumming for EEPROM	Stop background checking entire EEPROM table
Enable Checksumming for EEPROM	Restart background checking entire EEPROM table

CS User Defined Memory Commands

Command	Description
Get Region ID for Memory Address	Retrieves Memory table entry ID for region that covers given address
Recompute baseline for Memory Region	Recompute the baseline checksum for the given Memory region ID
Report Baseline for Memory Region	Sends event message with baseline checksum for given Memory region ID
Disable Checksumming for Memory Region	Stop background checking for the given Memory region ID
Enable Checksumming for Memory Region	Restart background checking for the given Memory region ID
Disable Checksumming for Memory	Stop background checking entire Memory table
Enable Checksumming for Memory	Restart background checking entire Memory table

CS Application Commands

Command	Description
Recompute baseline for Application	Recompute the baseline checksum for the given App name
Report Baseline for Application	Sends event message with baseline checksum for given App name
Disable Checksumming for Application	Stop background checking for the given App name
Enable Checksumming for Application	Restart background checking for the given App name
Disable Checksumming for Apps	Stop background checking entire App table
Enable Checksumming for Apps	Restart background checking entire App table

CS Table Commands

Command	Description
Recompute baseline for Table	Recompute the baseline checksum for the given Table name
Report Baseline for Table	Sends event message with baseline checksum for given Table name
Disable Checksumming for Table	Stop background checking for the given Table name
Enable Checksumming for Table	Restart background checking for the given Table name
Disable Checksumming for Tables	Stop background checking entire Table table
Enable Checksumming for Tables	Restart background checking entire Table table

CS Housekeeping Telemetry Message - 1

Telemetry	Description
CmdCounter	Number of accepted commands
CmErrCounter	Number of rejected commands
ChecksumState	Enable/Disable status of background checksumming
EepromCSState	Enable/Disable status of EEPROM checksumming
MemoryCSState	Enable/Disable status of user-defined Memory checksumming
AppCSState	Enable/Disable status of Apps checksumming
TablesCSState	Enable/Disable status of Tables checksumming
OSCSState	Enable/Disable status of OS code segment checksumming
CfeCoreCSState	Enable/Disable status of cFE core checksumming
EepromCSErrCounter	Number of checksum errors reported in EEPROM
MemoryCSErrCounter	Number of checksum errors reported in checksummed area of memory
AppsCSErrCounter	Number of checksum errors reported in checksummed apps
TablesCSErrCounter	Number of checksum errors reported in checksummed tables
CfeCoreCSErrCounter	Number of checksum errors reported in cFE core code
OSCSErrCounter	Number of checksum errors reported in OS code

CS Housekeeping Telemetry Message - 2

Telemetry	Description
CurrentCSTable	Current table being checksummed (cFE Core, OS, EEPROM, Memory, Apps, Tables)
CurrentEntryInTable	Current entry ID in the table currently being checksummed
EepromBaseline	Current baseline checksum of entire EEPROM
OSBaseline	Current baseline checksum of OS code segment
CfeCoreBaseline	Current baseline checksum of cFE Core code segment
LastOneShotAddress	Start address used in the last One Shot checksum command
LastOneShotSize	Number of bytes used in the last One Shot checksum command
LastOneShotChecksum	Calculated checksum by the last One Shot checksum command
PassCounter	Number of times CS has passed through all of its tables

Data Storage (DS)

DS Overview

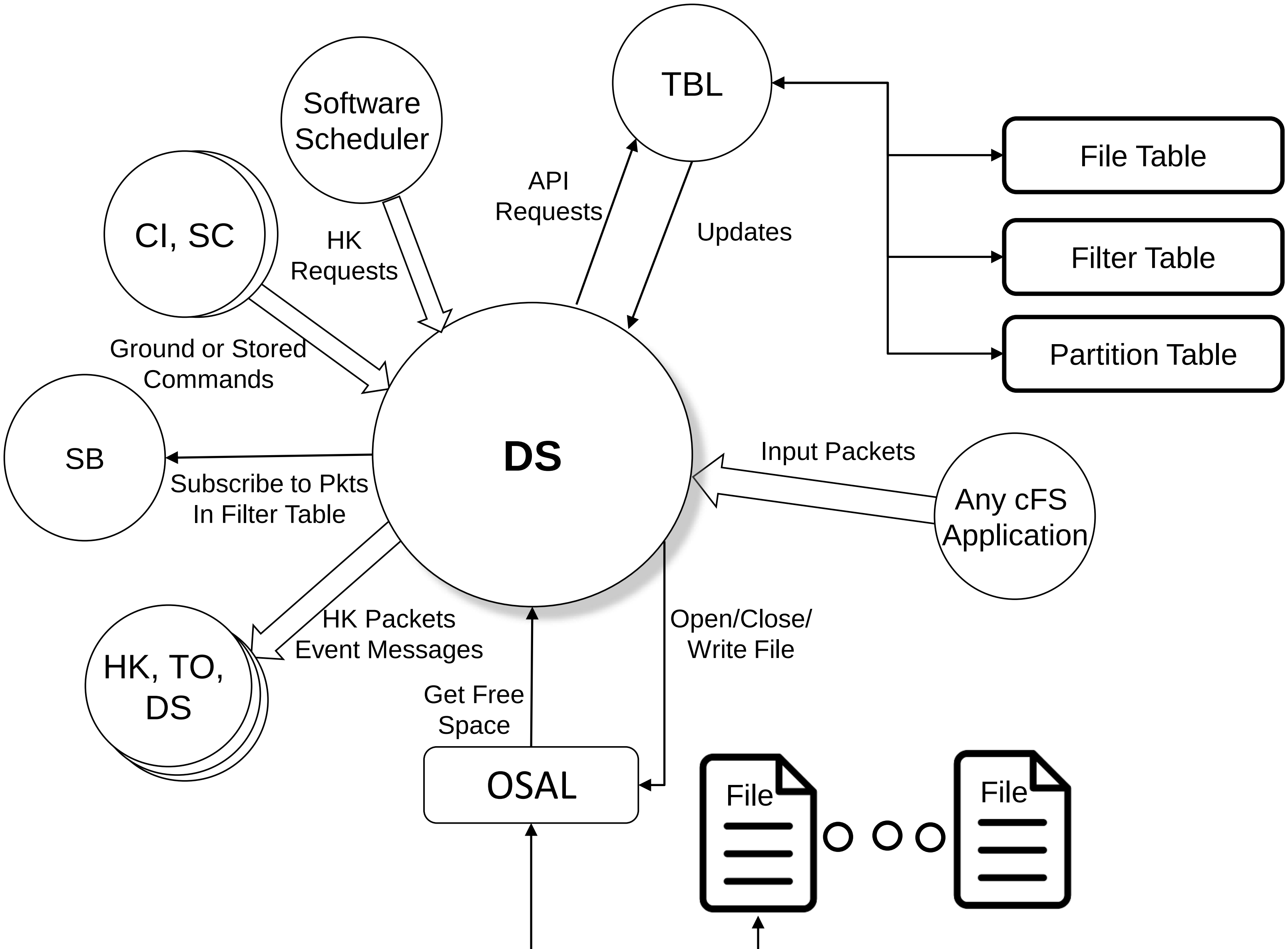
- **DS receives messages from the software bus and writes them to files**
  - Messages to be stored in each file are specified in tables
    - provides time and sequence based filtering of message packet
  - Files may be size-based or time-based (table defined)

- **DS cycle through the following actions:**
  - create a file, write data pkts to the file, then close file based on file size or time

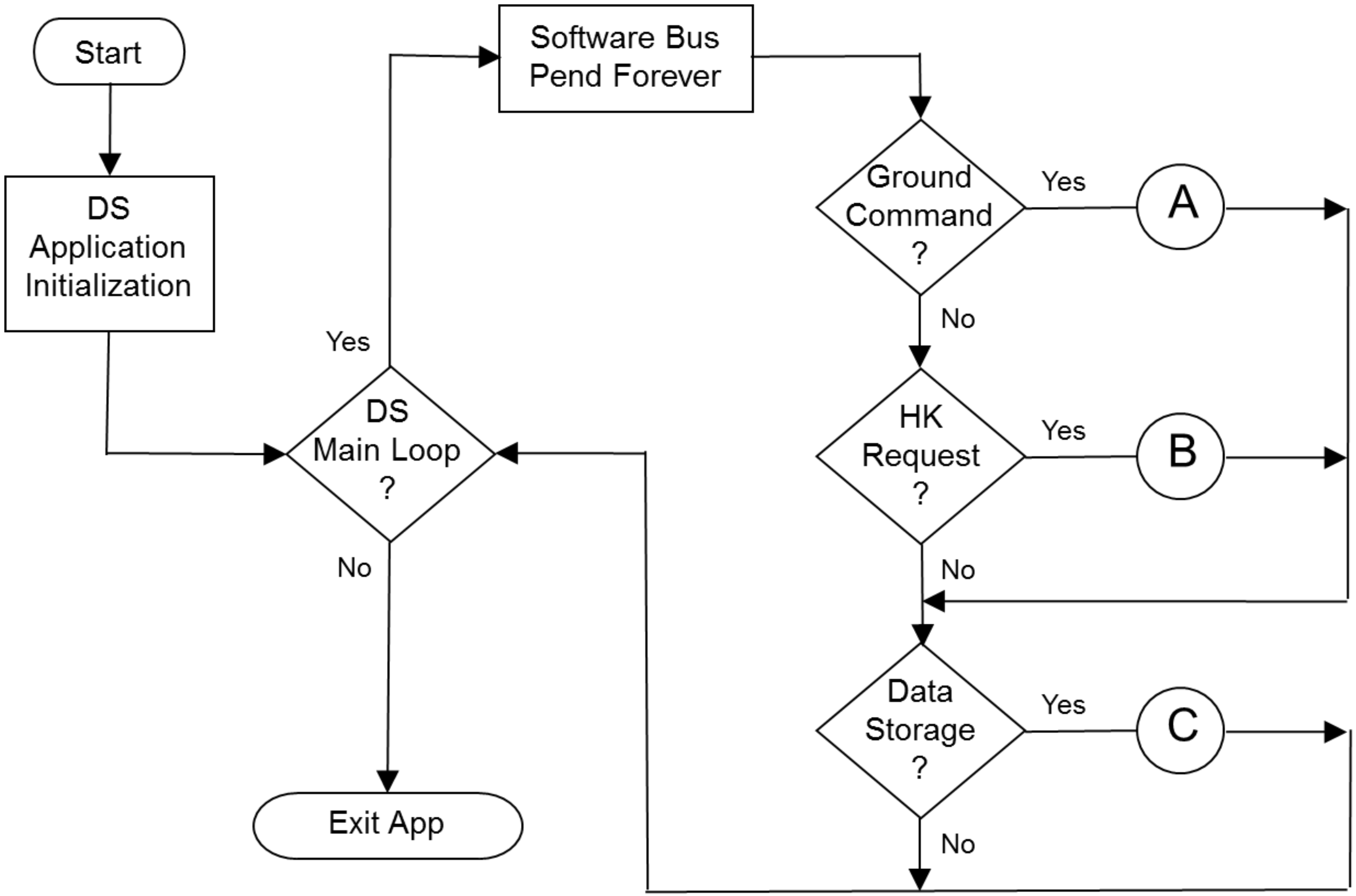
- **DS uses two tables**
  - Filter Table - one entry per input message id
  - File Table - one entry per file basename

- **DS has no download or playback capabilities**

DS Context Diagram



DS Flow Control



DS Timed Based Files

- **Time-Based Files:**
  - Filename = Basename + YYYYDDDHHMMSS + extension
  - Time in filename is the time the file was created.
  - Files are created when the first input pkt is received
  - File Table tells DS how long (in seconds) the file should be open
    - File closed when time reached or reset occurs
  - Next file created when next input pkt received

DS Size Based Files

- **Size-Based Files:**
  - Filename = Basename + 8 Digit Sequence + extension
  - Sequence count in filename starts at zero after a power-on reset.
  - If input packet would cause file size to be > table defined “max file size”, then
    - Current open file is closed
    - File sequence count is incremented
    - New file created and input packet is written

DS Filter Table

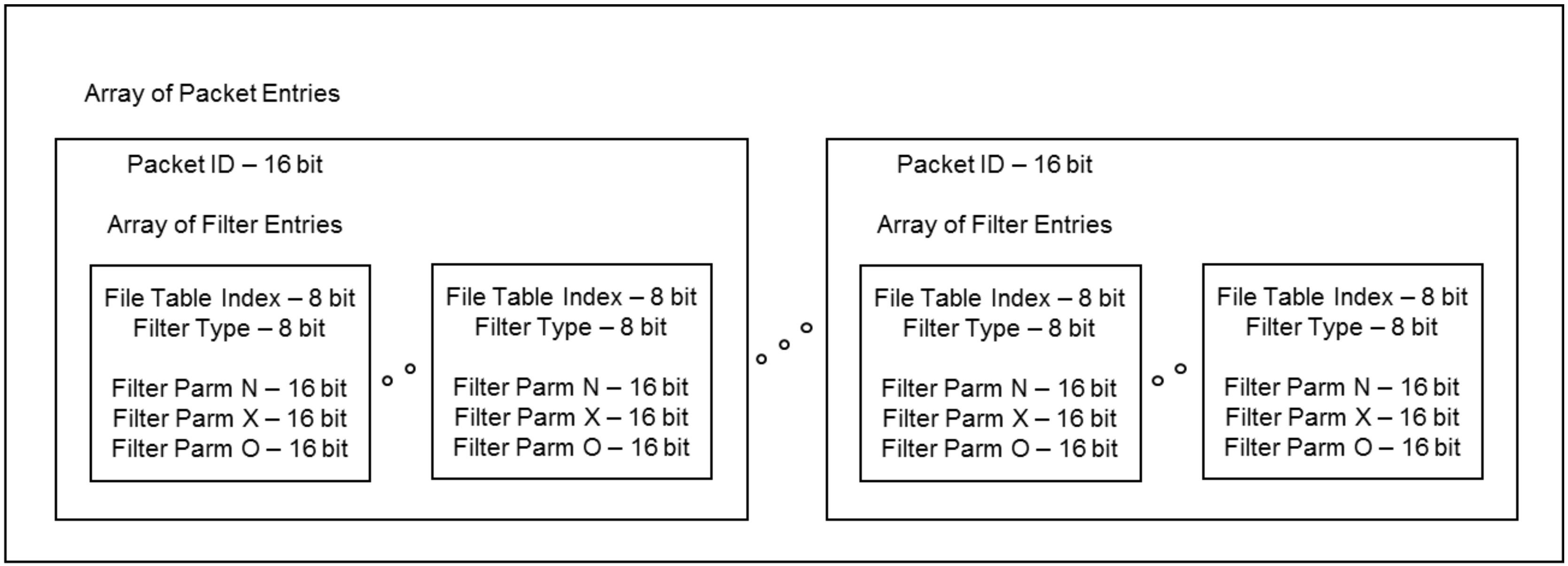


Table Size Configuration Parameters:

DS_PACKETS_IN_FILTER_TABLE: default value = 256

DS_FILTERS_PER_PACKET: default value = 4

DS File Tables

Table Descriptor - string

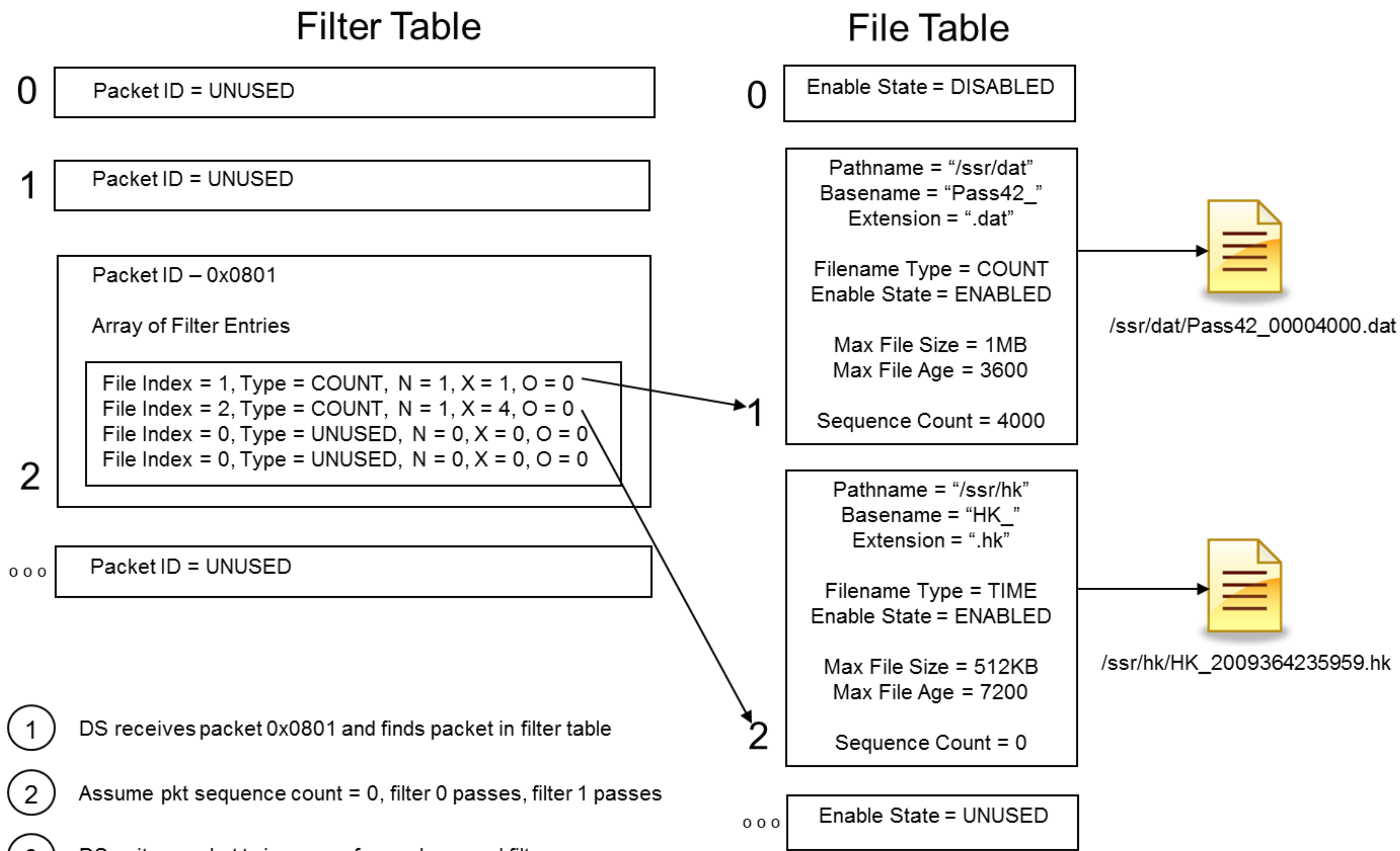
Array of File Entries

Pathname - string Basename - string Extension - string Filename Type – 16 bit Enable State – 16 bit Max File Size – 32 bit Max File Age – 32 bit Sequence Count – 32 bit	Pathname - string Basename - string Extension - string Filename Type – 16 bit Enable State – 16 bit Max File Size – 32 bit Max File Age – 32 bit Sequence Count – 32 bit

Table Size Configuration Parameters:

DS_DEST_FILE_CNT: default value = 16

DS Input Packet Flow



1. DS receives packet 0x0801 and finds packet in filter table
2. Assume pkt sequence count = 0, filter 0 passes, filter 1 passes
3. DS writes packet twice, once for each passed filter

DS Configuration Parameters - 1

Configuration Parameter	Description	Default Value
DS_DESTINATION_TBL_NAME	Logical name for the Destination File Table	"FILE _TBL"
DS_DEF_DEST_FILENAME	Default table filename — loaded at startup	"/cf/appsids_file_tbl.tbl"
DS_DEST_FILE_CNT	Number of file entries in Destination File Table	16
DS_PATHNAME_BUFSIZE	Size of pathname buffer in cmds, tlm, tables	OS_MAX_PATH_LEN (64)
DS_BASENAME_BUFSIZE	Size of basename buffer in cmds, tlm, tables	OS_MAX_PATH_LEN (64)
DS_EXTENSION_BUFSIZE	Size of extension buffer in cmds, tlm, tables	8
DS_FILTER_TBL_NAME	Logical name for the Packet Filter Table	"FILTER _TBL"
DS_DEF_FILTER_FILENAME	Default table filename — loaded at startup	"/cf/apps/ds_filter_tbl.tbl"
DS_PACKETS_IN_FILTER_TABLE	Number of packet entries in Packet Filter Table	256
DS_FILTERS_PER_PACKET	Number of filters per packet table entry	4

DS Configuration Parameters - 2

Configuration Parameter	Description	Default Value
DS_SEQUENCE_DIGITS	Number of digits in sequence portion of filename	8
DS_MAX_SEQUENCE_COUNT	Max filename sequence count before rollover	99999999
DS_TOTAL_FNAME_BUFSIZE	Size of buffer to contain fully qualified filename	OS_MAX_PATH_LEN (64)
DS_FILE_HDR_SUBTYPE	Common cFE file header subtype identifier for DS files	12345
DS_FILE_HDR_DESCRIPTION	Descriptive text for DS file secondary header	"DS data storage file"
DS_FILE_MIN_SIZE_LIMIT	Smallest amount that may be set for file max size limit	1024 (bytes)
DS_FILE_MIN_AGE_LIMIT	Smallest amount that may be set for file max age limit	60 (seconds)
DS_APP_PIPE_NAME	Logical name for DS application input pipe	"DS _ CMD _PIPE"
DS_APP_PIPE_DEPTH	Size of DS application input pipe	256 (packets)
DS_MAKE_TABLES_CRITICAL	If "1", cFE Table Services will store DS tables in CDS	0
DS_SECS_PER_HK_CYCLE	DS measures file age by counting HK cycles	5 (seconds)

DS Commands

Command	Description
No-op	General DS aliveness test — verifies command handler and event generation
Reset Counters	Reset DS application housekeeping telemetry counters
Set Enable State For Packet Processor	Set enable/disable state for data storage packet processor
Set Destination File For Packet Filter	Modify packet filter table entry — set destination file
Set Filter Type For Packet Filter	Modify packet filter table entry — set filter type (sequence count vs time)
Set Filter Parms For Packet Filter	Modify packet filter table entry — set filter parms (N, X, 0)
Set Filename Type For Destination File	Modify destination file table entry — set filename type (sequence count vs time)
Set Enable State For Destination File	Modify destination file table entry — set enable/disable state
Set Path Portion of Destination Filename	Modify destination file table entry — set path portion of filename (string)
Set Base Portion of Destination Filename	Modify destination file table entry — set base portion of filename (string)
Set Extension Portion of Destination Filename	Modify destination file table entry — set extension portion of filename (string)
Set Max File Size For Destination File	Modify destination file table entry — set max file size (bytes)
Set Max File Age For Destination File	Modify destination file table entry — set max file age (seconds)
Set Filename Sequence Count For Destination File	Modify destination file table entry — set filename sequence counter value
Close Destination File	Close data storage file, file re-opened when next packet written to file

DS Housekeeping Telemetry Message

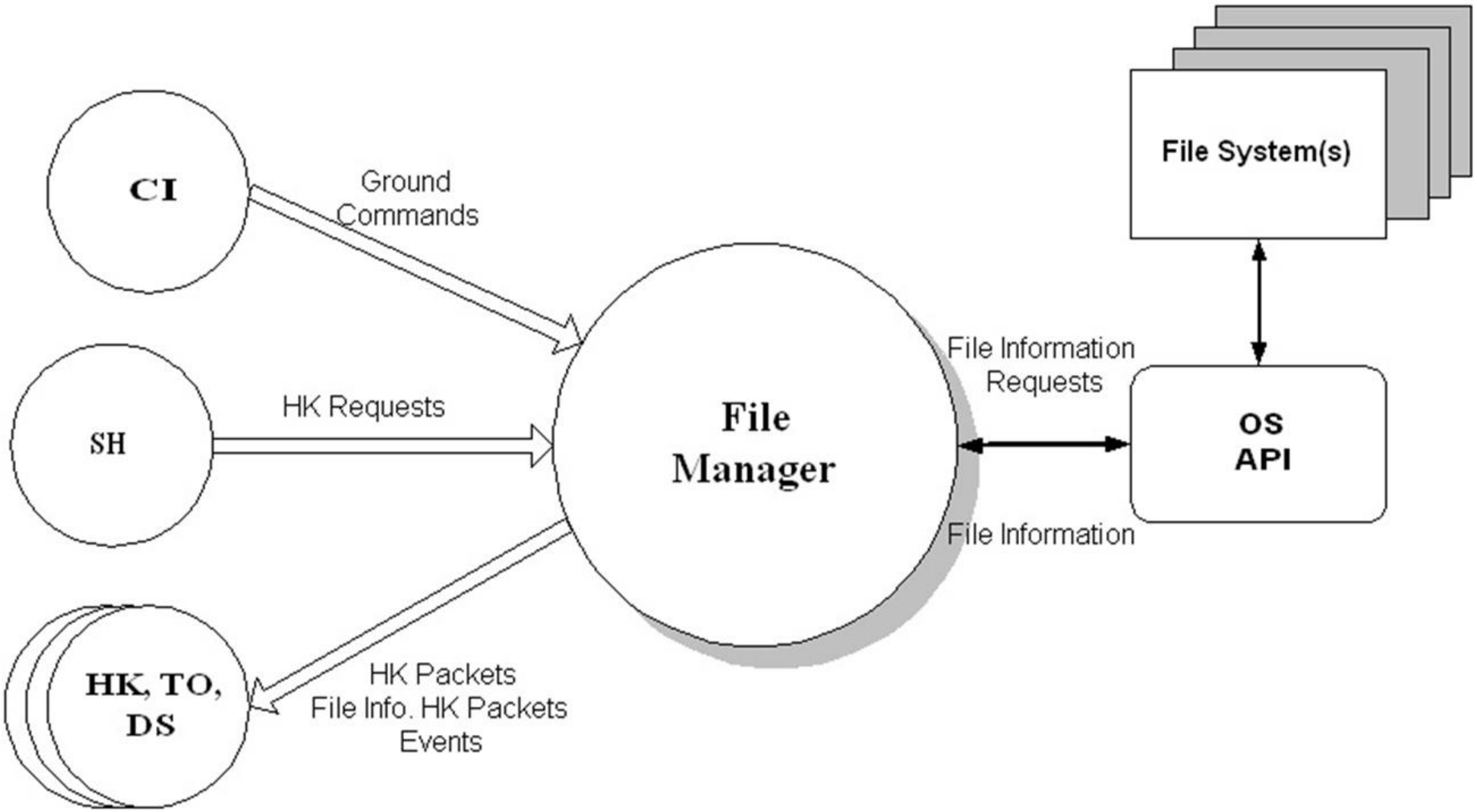
Telemetry Point	Description
Commands Accepted Counter	Number of successful ground commands (includes commands from on board sources)
Commands Rejected Counter	Number of commands with process errors
Disabled Packet Counter	Packets not processed because the packet processor was disabled
Ignored Packet Counter	Packets not processed because tables were not loaded or the packet was not in the filter table
Filtered Packet Counter	Packets processed that did not pass any filter tests (includes disabled destination files)
Passed Packet Counter	Packets processed that passed at least one filter test
File Write Counter	Total number of successful file writes
File Write Error Counter	Total number of file write errors
File Update Counter	Number of files with secondary header successfully updated prior to being closed
File Update Error Counter	Number of errors trying to update the secondary file header
Destination Table Load Counter	Number of times that cFE Table Services signaled new destination file table data
Destination Table Error Counter	Number of times that cFE Table Services signaled no destination file table data was available
Filter Table Load Counter	Number of times that cFE Table Services signaled new packet filter table data
Filter Table Error Counter	Number of times that cFE Table Services signaled no packet filter table data was available
Packet Processor Enable State	Current enable/disable state for the data storage packet processor
Array data (per destination file)	
-- File age, size, rate of growth	File age in seconds, file size in bytes, file growth rate in bytes per second
-- Filename sequence count	Filename sequence counter (this value will be used when the next file is created)
-- Enable state, open state, filename	File enable state, file open state, current filename (if open)

File Manager (FM)

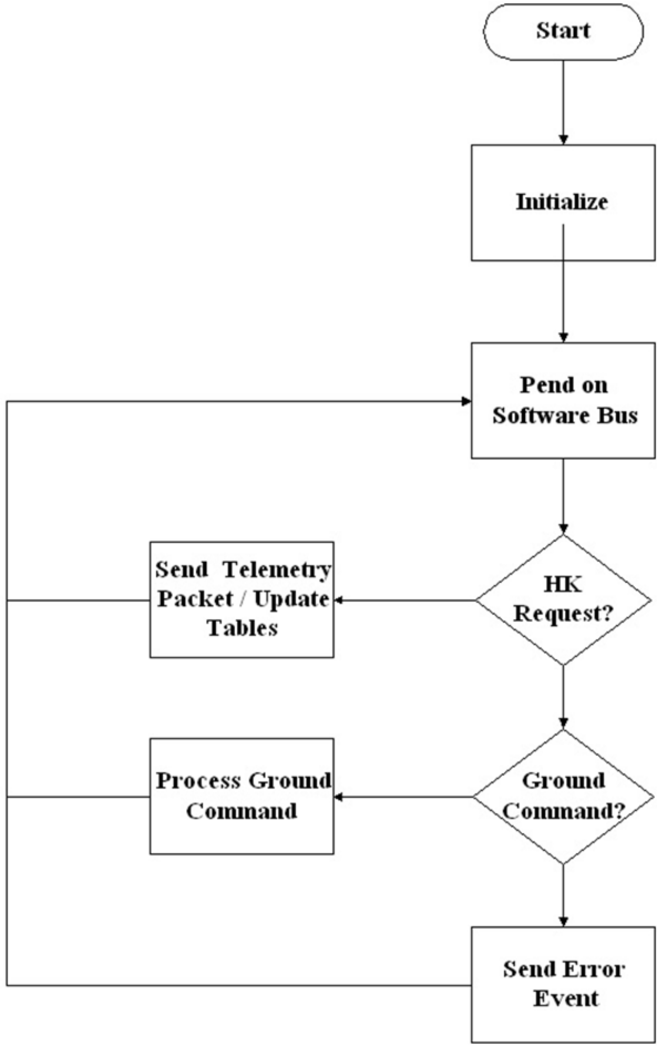
FM Overview

- **Provides a ground interface for:**
  - The management of onboard files
    - Copying files, Moving files, Renaming files, Deleting files, Closing files Decompressing files, and Concatenating files
    - Providing file information
    - Providing open file listings
  - The management of onboard directories
    - Creating directories
    - Deleting directories
    - Providing directory listings
  - Device free space reporting

FM Context Diagram



FM Flow Control



FM Configuration Parameters

- **Mission**
  - Max number of onboard file systems (defaults to 3)
  - Onboard File System Device Names
    - Defaults
      - o "/ram"
      - o "/eep0"
      - o "/eep1"
- **Platform**
  - Command default output filenames
    - Directory Listing File (defaults to "/ram/fm_dirlist_file.dat")
  - Max full path specification character length (defaults to OS_MAX_PATH_LEN = 64)
  - Max files in an open file listing (defaults to OS_MAX_NUM_OPEN_FILES = 128)
  - Max files in a directory listing message (defaults to 20)

FM Commands

Command	Description
Noop	Increments the Command Accepted Counter and sends a debug event message
Reset Command Counters	Initializes the following FM counters to 0: Command Rejected Counter, Command Accepted Counter
File Copy	Copies the command-specified file to the command-specified destination file or directory
File Move	Moves the command-specified file to the command-specified destination file or directory
Rename File	Renames the command-specified file to the command-specified file
Delete File	Deletes the command-specified file, if and only if, the file is closed
Delete All Files	Deletes all files in the command-specified directory, if and only if, the files are closed.
Decompress File	Decompresses the command-specified file creating the command-specified destination file
Concatenate Files	Concatenates the command-specified source files creating the command-specified destination file
File Information	Creates and sends a software bus message containing the file size, last modification time, and file status (Open, Closed) of a given file, if and only if, the file exists

FM File Information Message

Telemetry Point	Description
FileStatus	Status indicating whether the file is Open or Closed
CRC_Computed	Flag indicating if a CRC was computed on the command specified file
<OPTIONAL> CRC	Computed CRC of file contents
FileSize	Size of file in bytes
LastModifiedTime	System time the file was last modified
Filename	Echo of command specified filename

- CRC ground tool provided

FM Commands - 2

Command	Description
List Open Files	Creates and sends a software bus message containing the number of open files, the name/path of each open file, and application identifier associated with each open file
Create Directory	Creates the command-specified directory
Delete Directory	Removes the command-specified directory, if and only if, the command-specified directory is empty
Directory Listing via File	Writes to a file the complete listing of the command-specified directory
Directory Listing via Message	Creates and sends a software bus message containing the contents of a directory (up to <PLATFORM_DEFINED> filenames, starting at the command-specified offset)

FM Open File Listing Message

Telemetry Point	Description
NumOpenFiles	Number of open files in the FSW system
FileNames[1..n] where n = <PLATFORM_DEFINED> FM_MAX_OPEN_FILE_LIST_MSG_FILES	Names of open files in the FSW system
AppNames[1..n] where n = <PLATFORM_DEFINED> FM_MAX_OPEN_FILE_LIST_MSG_FILES	Names of applications that have files open in the FSW system

FM Directory Listing File

- **File Format**
  - Binary
- **File Content**
  - cFE file header
    - Header length
    - Spacecraft ID
    - Processor ID
    - Application ID
    - Creation Time (seconds and subseconds)
    - File Description
  - Echo of command-specified directory name
  - Directory size in bytes
  - Total number of files in the directory
  - For each file contained in the directory:
    - File Name
    - File Size
    - Last Modification Time

FM Directory Listing Message

Telemetry Point	Description
DirSize	Directory size in bytes
DirOffset	Echo of command specified directory offset
TotalFiles	Total number of files contained in the command specified directory
FileSizes[1..n] where n = <PLATFORM_DEFINED> FM_MaxDirListMsgFiles	Sizes of the files contained within the command-specified directory starting at the command specified offset
FileLastModTimes[1..n] where n = <PLATFORM_DEFINED> FM_MaxDirListMsgFiles	Last modification times of the files contained within the command-specified directory starting at the command specified offset
DirName	Echo of command specified directory name
FileNames[1..n] where n = <PLATFORM_DEFINED> FM_MaxDirListMsgFiles	Names of files contained within the command-specified directory starting at the command-specified offset

FM – Housekeeping Telemetry Message

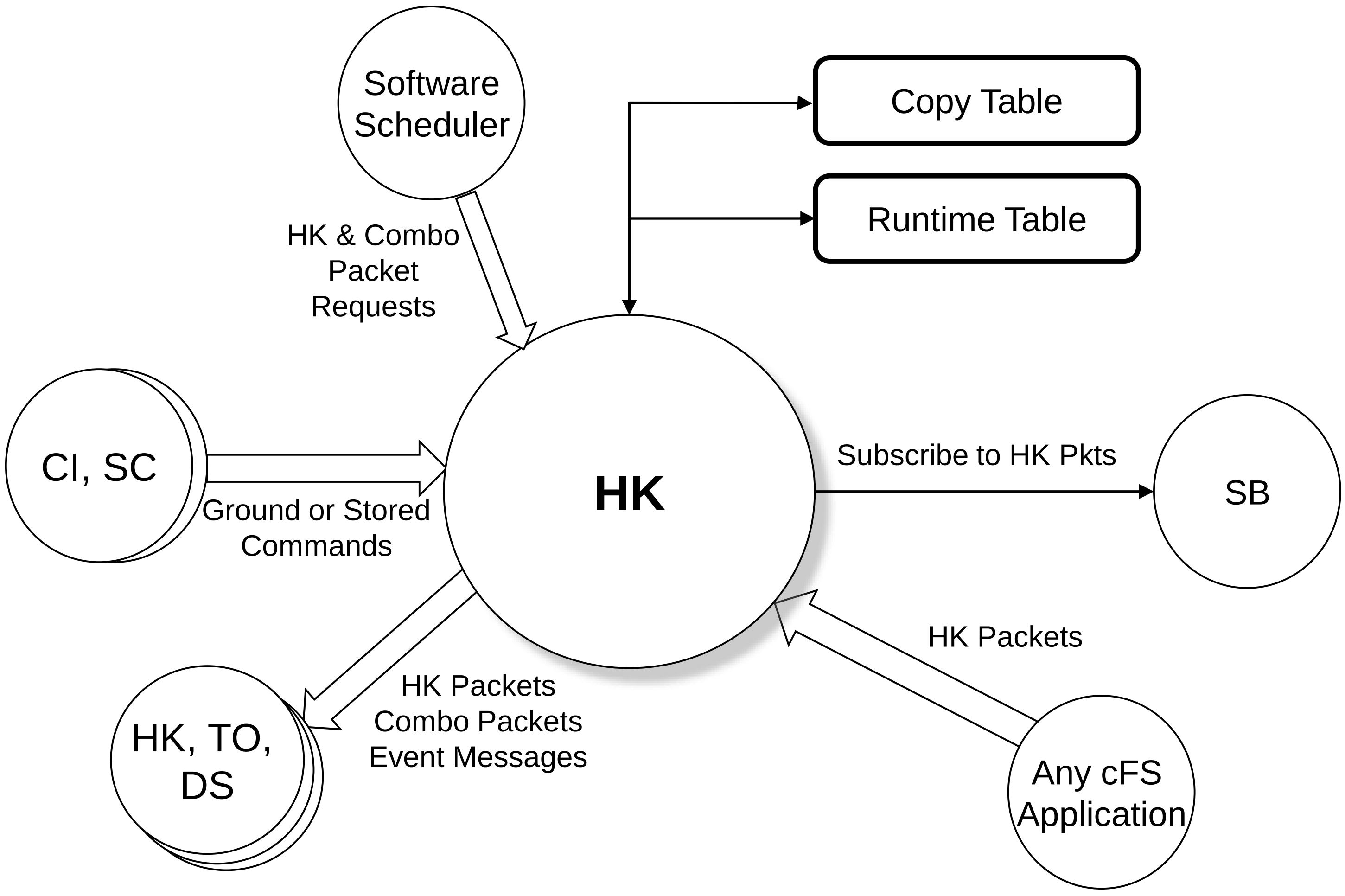
Telemetry Point	Description
CommandCounter	Number of rejected commands
CommandErrCounter	Number of accepted commands
NumOpenFiles	Number of open files in the entire FSW system
BlockSize[1..n]	Block size of drive n
NumBlocks[1..n] where n = <MISSION_DEFINED> FMMaxNumDevices	Number of available blocks on drive n

Housekeeping (HK)

HK Overview

- **Builds combined telemetry messages containing data from system applications**
  - Sends notification when expected data is not received.
    - Expected data is specified by table

HK Context Diagram



HK Copy Table File Format

Static Copy Table - Loaded by Ground

	InMid	InOffset	OutMid	OutOffset	#bytes
0	0x805	12	0x812	12	8
1	0x810	12	0x812	20	4
2	0x805	20	0x814	16	12
3	0	0	0	0	0

o
o
o

124	0x851	12	0x812	46	8
125	0	0	0	0	0
126	0	0	0	0	0
127	0	0	0	0	0

Runtime Table - dump only

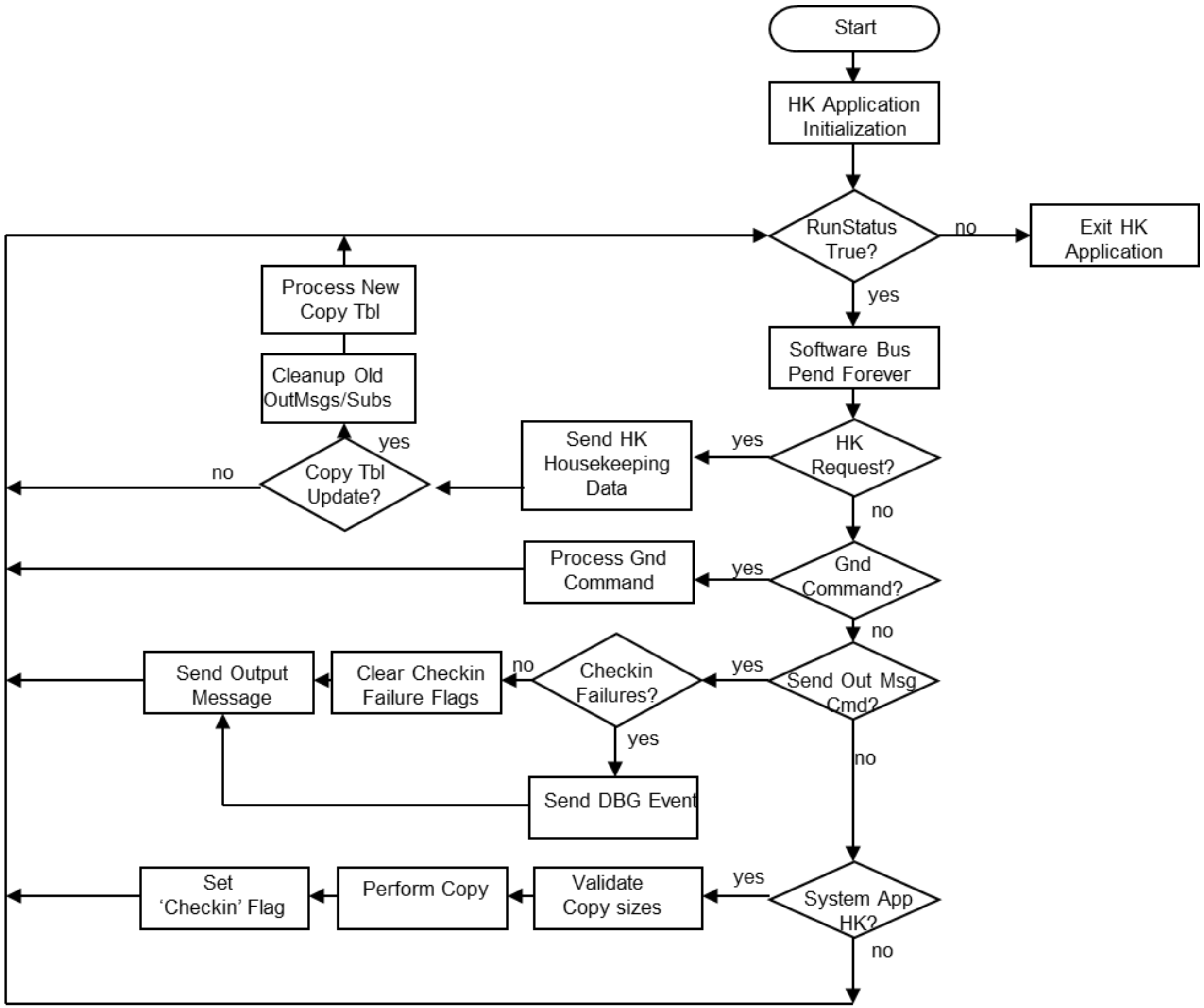
InMid Subscribed	InMid Present	OutMid Address
Yes	No	0x42004589
Yes	No	0x42004589
Yes	No	0x420046FE
No	No	NULL

o
o
o

Yes	No	0x42004589
No	No	NULL
No	No	NULL
No	No	NULL

Number of elements in table is an HK config parameter

HK Flow Control



HK Configuration Parameters

Parameter	Description	Default Value
Pipe Depth	Depth of HK command pipe	40
# of Copy Table Entries	Number of elements in the HK copy table to process	128
# bytes in memory pool	Number of bytes to allocate in the HK memory 6144 pool (needed for the HK output packets)	6144
Default HK Copy Table Name	--	CopyTable
Default HK Runtime Table Name	--	RuntimeTable
Default HK Copy Table Filename	--	/cf/apps/hk_cpy_tbl.tbl
Mission Revision	Mission-level revision number	0

HK Commands

Command	Description
No-Op	Increments the HK Command Accepted Counter and sends an info event message
Reset Counters	Initializes the following counters to 0: - Command Counter - Command Error Counter - Output Messages Sent - Missing Data Counter

HK Housekeeping Telemetry Message

Telemetry Point	Description
Command Counter	Number of accepted commands
Command Error Counter	Number of rejected commands
Output Messages Sent	Number of output messages sent
Missing Data Counter	Number of times missing data from other apps was detected
Memory Pool Handle	Used to get memory pool statistics. The memory pools is used to allocate memory for output messages.

National Aeronautics and Space Administration

Health & Safety (HS)

HS Overview

- **Performs Application Monitoring**
  - Detects when critical applications are not running and take a table defined action
- **Performs Event Monitoring**
  - Detects critical events and take a table defined action
- **Manages Watchdog**
  - Initializes and service the watchdog.
  - Withholds servicing of the watchdog if certain conditions are not met.
- **Manages CPU**
  - Reports CPU Utilization
  - Detects CPU Hogging and take appropriate action
  - Provides CPU Aliveness Indication
- **Reports Table Defined Execution Counters**
  - Can include Application Main Tasks, Child Tasks, ISRs, and Device Drivers

HS Application Monitoring

- **Monitors the health of table specified applications**
  - Both cFE core applications and any cFS application
- **How are they monitored?**
  - Use the counters maintained by ES in the CFE_ES_RunLoop function.
    - Applications must call CFE_ES_RunLoop to increment the execution counter and let the system know they are active.
- **What are the response options for an application not running?**
  1. Perform Processor reset
     - Sets Service_watchdog flag to FALSE
  2. Restart Application
  3. Send an Event message
  4. Send Software Bus Message

- **What happens if an app goes away or is restarted?**
  - There should be sufficient time to restart an app before it's flagged as missing
  - Application monitoring can be disabled during application updates/maintenance
  - Application monitoring table can be reloaded if an application is permanently deleted

HS – Event Monitoring

- **Subscribe to all event messages**
  - Monitored events are table specified
- **HS can take one of the following actions on events:**
  - Processor Reset
  - Reset Application
  - Delete Application
  - Send Software Bus Message
- **Event monitoring can be turned on or off by command**

HS Watchdog Management

- **Watchdog must be initialized at startup**
  - BSP will program watchdog to a reasonable value to allow system to start
- **Watchdog will be serviced as long as "Service_watchdog" flag is TRUE**
- **If HS is not running, the watchdog will expire causing a CPU reset.**
- **The "Service_watchdog" flag is set to FALSE if**
  - There is CPU hogging for more than <TBD, Configuration parameter > seconds
  - There is a Critical Application Monitoring failure
- **One of the above conditions should be enough to restart the system before the watchdog expires**
  - Why bother with the watchdog then?
    - If the software gets "stuck" then the watchdog will make sure it is reset.
- **OS API will supply get timeout, set timeout, and service functions separately**
  - HS will set the timeout to the default value when initializing, and service every cycle

HS CPU Management

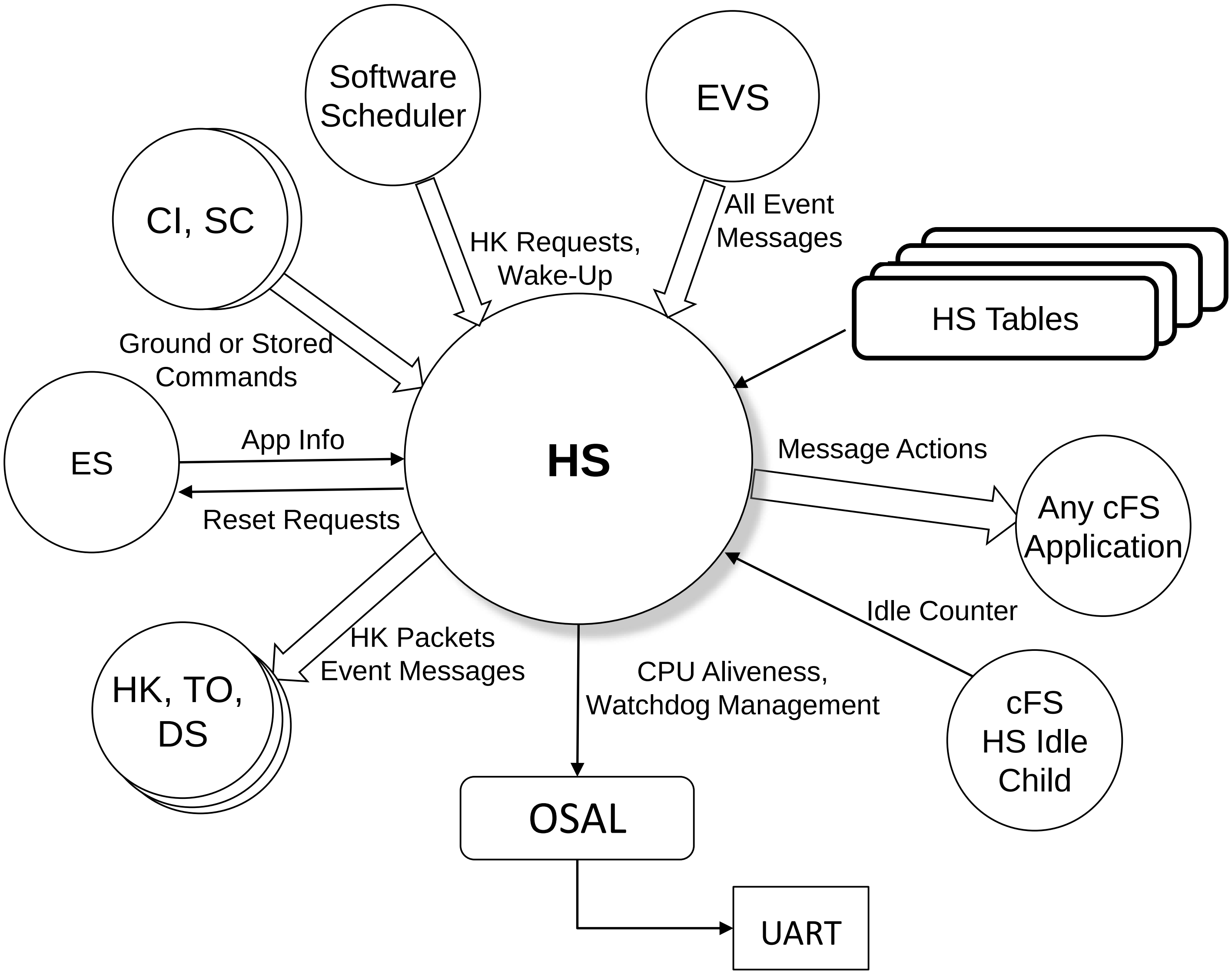
- **HS will perform the following CPU related functions:**
  - Reports CPU utilization information
    - The CPU information comes from OS/BSP
      - May have different implementations on different platforms
    - Collects and report average CPU utilization over <TBD, Configuration Parameter> time
    - Collects and report peak CPU utilization over <TBD, Configuration Parameter> time
  - Provides CPU hogging indication
    - If the CPU is at 100% start the “hogging” counter
    - If hogging counter reaches <TBD, Configuration parameter > limit
      - Send event / make syslog entry
      - Set “Service_watchdog flag” to FALSE
      - Processor Reset
  - Provides CPU aliveness indication (output characters to UART)
    - Enable/disable Command to output periodic heartbeat to UART
    - Output characters are <TBD, Configuration parameter >

HS –Execution Counter Reporting

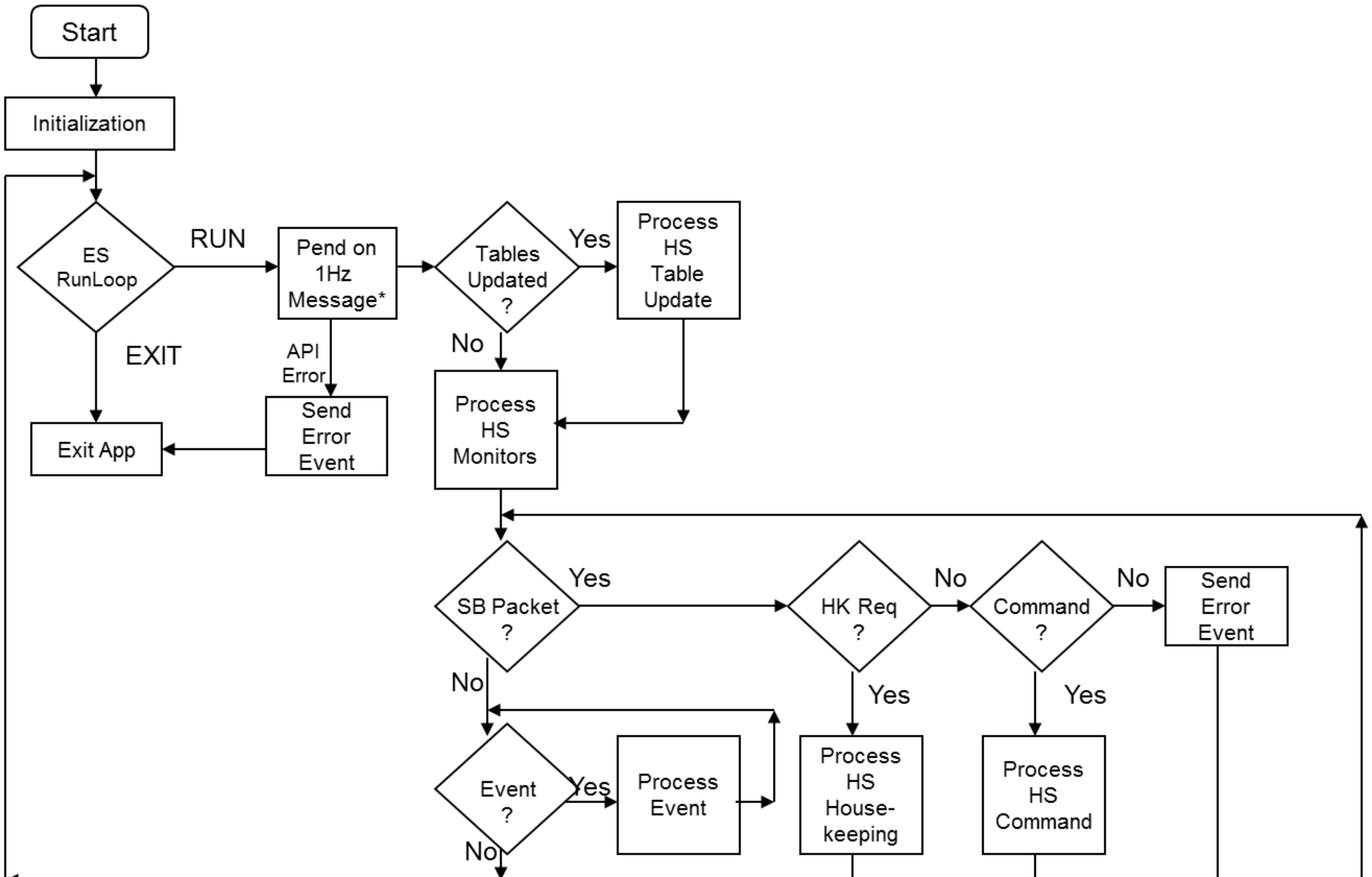
- **HS has a configurable number of execution counter entries in its housekeeping telemetry message**
  - A Table specifies the counters that will be reported in in the housekeeping telemetry message.
  - On every housekeeping request, HS copies the requested execution counters into the packet

- **Where does HS get the execution counter status?**
  - ES Maintains execution counters for:
    - cFE Core Apps
    - cFS Apps
    - Child Tasks
    - Device Drivers/ISRs
  - App Main Counters are incremented by calling CFE_ES_RunLoop
    - The counters are different from heritage counters that incremented before and after the software bus call.
  - Child task counters are incremented by using an ES counter API
  - Device Driver and ISR counters can be incremented with an ES counter API
  - Counters are fetched by calling ES GetAppInfo and GetTaskInfo API functions

HS Context Diagram



HS – Flow Control



*1Hz message is the only message that shows up on that pipe, and is a pend with timeout.

HS Configuration Parameters - 1

Parameter	Description	Default Value
HS_MAX_EXEC_CNT_SLOTS	Maximum Number of Execution Counters to be Reported	32
HS_MAX_MSG_ACT_TYPES	Maximum Number of Message Action types	8
HS_MAX_MSG_ACT_SIZE	Maximum Size of Message Action Message	16
HS_MAX_CRITICAL_APPS	Maximum Number of Critical Applications to Monitor	32
HS_MAX_CRITICAL_EVENTS	Maximum Number of Critical Events to Monitor	16
HS_WATCHDOG_TIMEOUT_VALUE	Default Watchdog timeout value in milliseconds to be set when initializing	10000
HS_CPU_ALIVE_STRING	String to output on UART	"."
HS_CPU_ALIVE_PERIOD	How often to output CPU aliveness indicator	5
HS_MAX_RESTART_ACTIONS	How many times a Processor Reset can be performed by a monitor failure	3
HS_CMD_PIPE_DEPTH	Software bus command pipe depth	12
HS_IDLE_TASK_PRIORITY	Priority of the Idle Task being used for CPU Utilization Monitoring	252

HS Configuration Parameters - 2

Parameter	Description	Default Value
HS_UTIL_CALLS_PER_MARK	Number of (1 Hz) calls between capturing the Idle Task Count	1
HS_UTIL_CYCLES_PER_INTERVAL	Number of HS cycles between calculating CPU Utilization	1
HS_UTIL_PER_INTERVAL_TOTAL	Number that signifies full utilization during one period	10000
HS_UTIL_PER_INTERVAL_HOGGING	Number that signifies CPU is being hogged in terms of full utilization	9900
HS_UTIL_CONV_MULT1 **HS_UTIL_CONV_DIV** **HS_UTIL_CONV_MULT2**	Utilization = Full Utilization —(((Idle Task Cycles * MULT1) / DIV) * MULT2)	Determined by Calibration
HS_UTIL_HOGGING_TIMEOUT	Number of Intervals for which hogging threshold must be exceeded to result in hogging event message	5
HS_UTIL_PEAK_NUM_INTERVAL	Number of intervals over which to report the peak value	64
HS_UTIL_AVERAGE_NUM_INTERVAL	Number of intervals over which to report the average value	4
HS_UTIL_DIAG_MASK	Used for calibration (how frequently to record time)	0xFFFFFFFF
HS_UTIL_DIAG_ARRAY_POWER	Used for calibration (how many time recordings are stored)	4

HS Commands - 1

Command	Description
Noop	Increment commands accepted counter and send event message
Reset Counters	Reset housekeeping telemetry counters
Disable Critical Application Monitor	Disables the monitoring and actions related to the critical application monitor.
Enable Critical Application Monitor	Enables and reinitializes the monitoring and actions related to the critical application monitor.
Disable Critical Event Monitor	Disables the critical event monitor function in HS
Enable Critical Event Monitor	Enables the critical event monitor function in HS
Disable CPU Aliveness Indicator	Stops the periodic output of characters to the UART.
Enable CPU Aliveness Indicator	Starts the periodic output of characters to the UART.
Set Max Processor Resets	Sets the max number of processor resets HS can perform to provided parameter value
Reset Processor Resets Counter	Resets the current count of HS performed Processor Resets.
Disable CPU Hogging Indicator	Stops the Hogging event from being sent
Enable CPU Hogging Indicator	Allows the Hogging event to be sent

HS Commands - 2

Command	Description
Report Utilization Diagnostics	Reports the current Utilization Diagnostics information in an event message
Set Utilization Parameters	Sets the calibration parameters used for Utilization Monitoring to the specified parameters
Set Utilization Diagnostics Mask	Sets the mask value being used for collecting Utilization Diagnostics information to a specified parameter

HS Housekeeping Telemetry Message

Telemetry Point	Description
HS CMDPC	Count of valid commands received
HS CMDEC	Count of invalid commands received
HS APPMONSTATE	Status of Critical Application Monitor (enabled, disabled)
HS EVTMONSTATE	Status of Event Monitor (enabled, disabled)
HS CPUALIVESTATE	Status of Aliveness Indicator output (enabled, disabled)
HS CPUHOGSTATE	State of CPU Hogging Indicator output (enable, disabled)
HS STATUSFLAGS	Status flags for table loaded and CDS available states
HS PRRESETCNT	Number of resets HS has performed so far
HS MAXRESETCNT	Max number of resets HS is allowed to perform
HS EVTMONCNT	Number of events monitored by the Event Monitor
HS INVALIDEVTAPPCNT	Number of entries in Event Monitor Table that have unresolvable task names
HS APPMONENABLE[TBD]	Application Monitor enable status by table entry
HS MSGACTCTR	Number of message actions sent
HS CPUUTILAVG	Average CPU Utilization
HS CPUUTILPEAK	Peak CPU Utilization
HS EXECOUNT[TBD]	Execution Counter Array

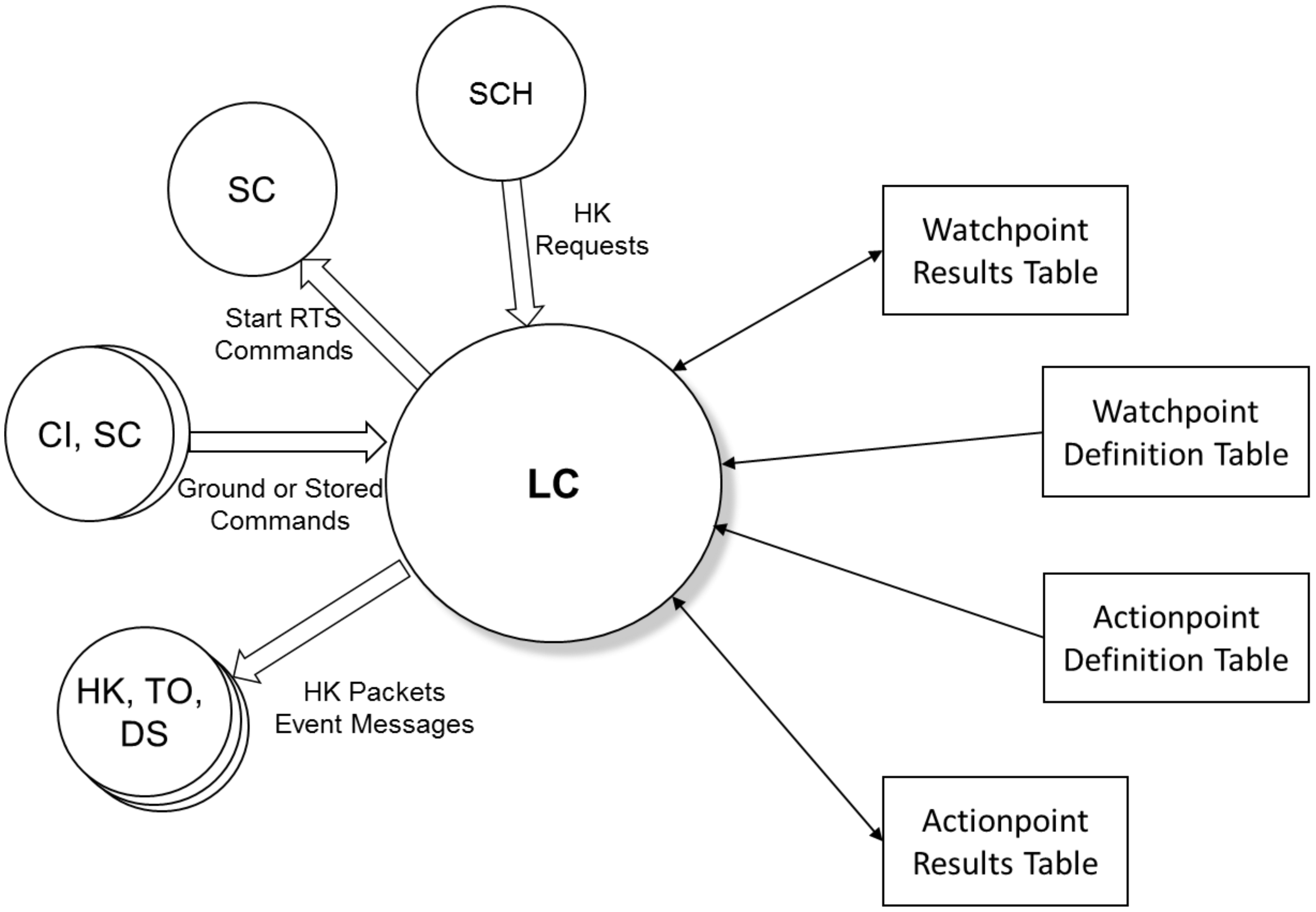
Limit Checker (LC)

LC Overview

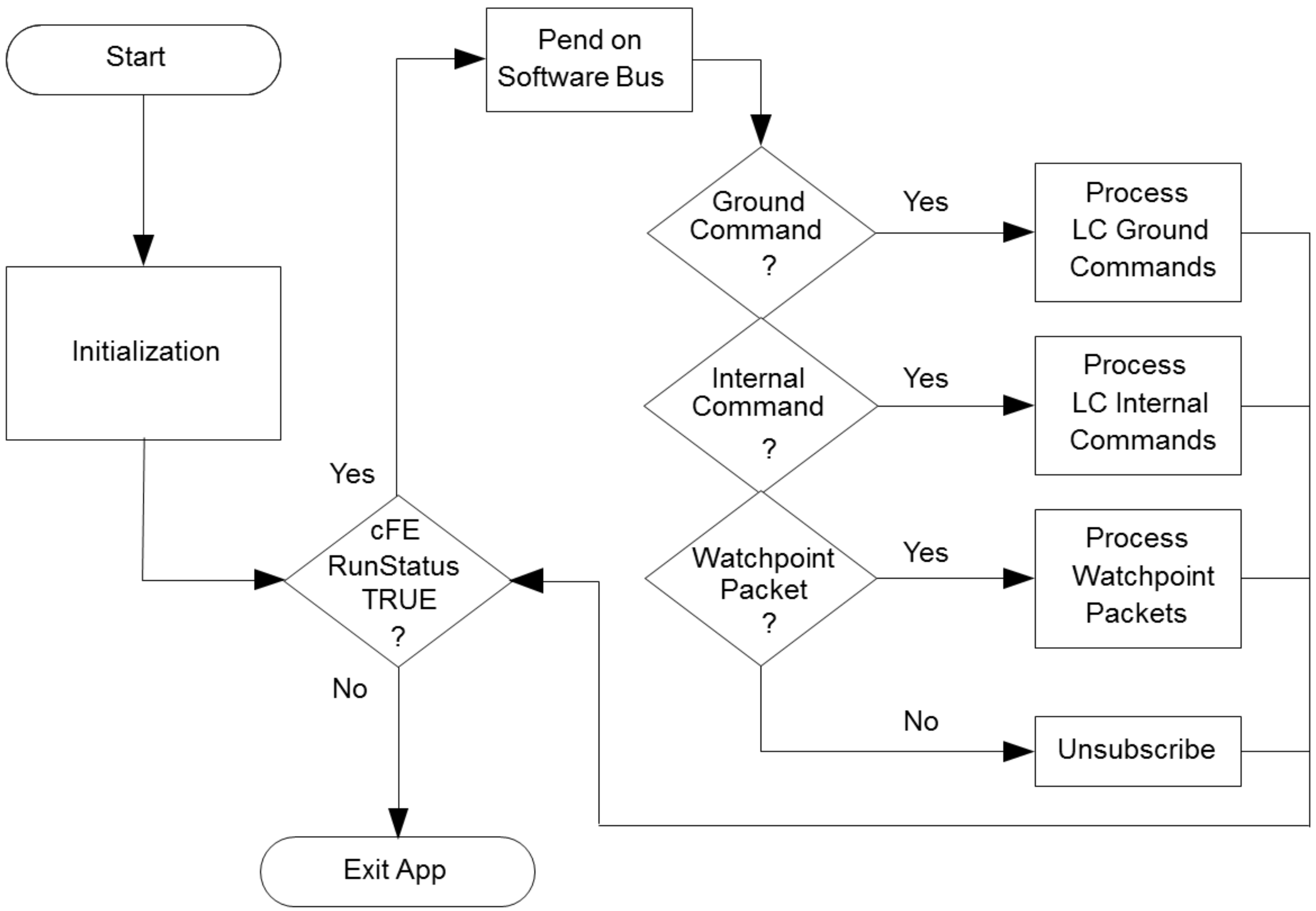
- **Monitors table defined Watchpoints**
  - Each watchpoint compares a telemetry data value with a constant threshold value
  - Comparison result may be True, False, Error, or Stale
  - Watchpoint results are stored in a dump-only table

- **Evaluates table defined Actionpoints**
  - Each action point analyzes the results of one (or more) watchpoints
  - Analysis result may be Pass, Fail, Error, or Stale
    - If number of consecutive fails exceeds limit then send event and optionally invoke RTS
  - Actionpoint results are stored in a dump-only table

LC Context Diagram



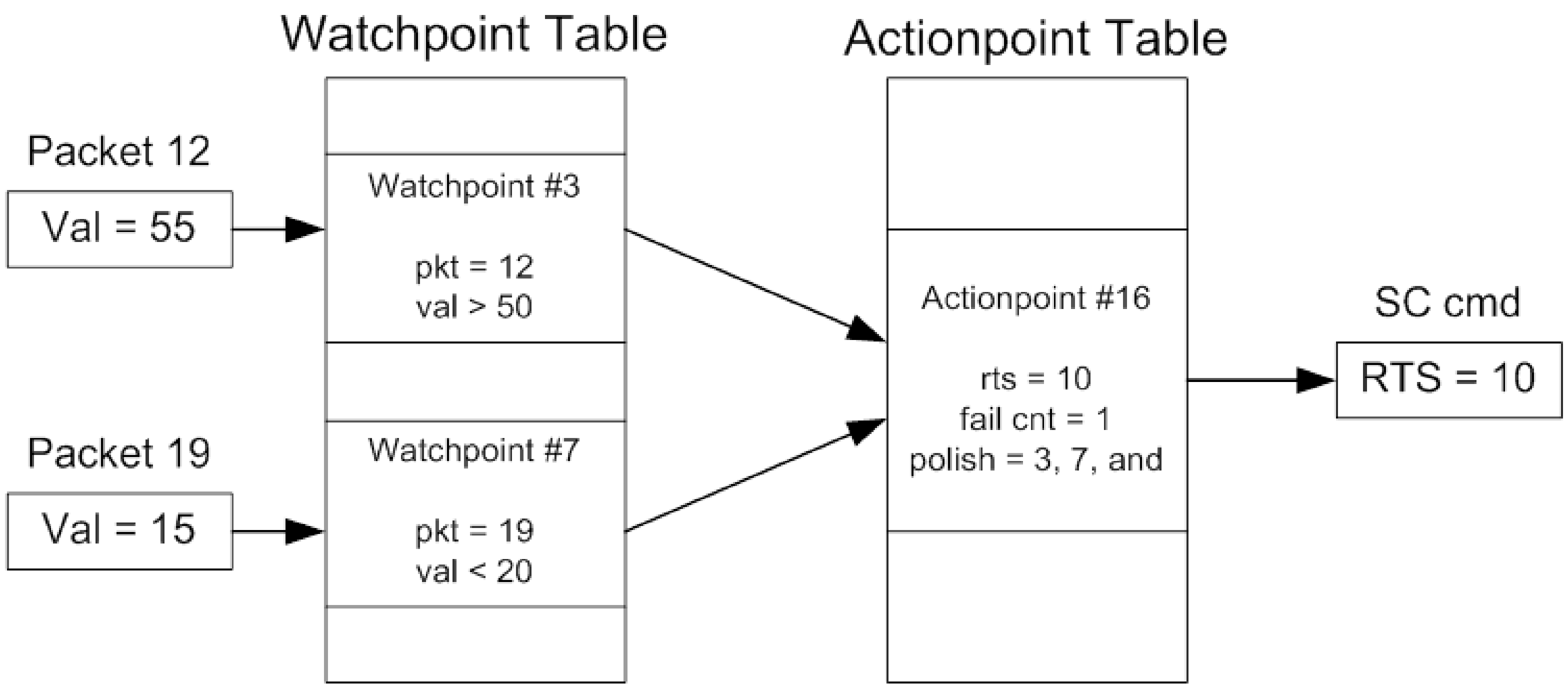
LC Flow Control



LC Scheduling

- **Watchpoints**
  - Watchpoints are evaluated whenever a packet containing them arrives
- **Actionpoints**
  - Actionpoints are processed only when an Actionpoint Sample Request is received
  - A Sample Request may target one or all actionpoints
  - The Sample Request is an internal message and will not increment the ground command counter

LC Monitor Process



1) Telemetry packet #12 arrives – watchpoint #3 results set to TRUE
2) Telemetry packet #19 arrives – watchpoint #7 results set to TRUE
3) Action command arrives – actionpoint #16 evaluates to FAIL
4) Actionpoint #16 triggers – LC sends command to start RTS #10

LC Modes of Operation

Supports three operating modes that can be set by command:

1. **Active**

   Normal Operation Mode. Performs all limit tests defined in the watchpoint definition table and invokes stored command sequences as defined in the actionpoint definition table when an actionpoint fails

2. **Passive**

   Performs all limit tests, but no stored command sequences are invoked as the result of actionpoint failures

3. **Disabled**

   No watchpoint or actionpoint evaluations take place

LC Custom Functions

- **Custom functions can be used in place of a standard comparison operator in watchpoint definitions**
  - If the comparison is designated “Custom” (instead of <, <=, !=, =, >, or >=), the function stub LC_CustomFunction is called and passed the following parameters
    - WatchIndex: The ID of the watchpoint for this call
    - ProcessedWPData: Watchpoint data read from message. Sized as a uint32, masked, and adjusted for any platform endian difference.
    - MessagePtr: Pointer to the message. If the function needs raw watchpoint data it can use this pointer to extract it.
    - WDTCustomFuncArg: Custom function argument for this watchpoint from the watchpoint definition table.
  - LC_CustomFunction returns a True or False that is used as the result of the comparison for the watchpoint that triggered the call
  - Custom functions are added by modifying the source for LC_CustomFunction
- **LC can have as many custom functions as monitor points**

LC Configuration Parameters

- Command Pipe Depth
- Maximum number of watchpoints
  - Dictates the size of the Watchpoint Definition and Results Tables
- Maximum number of actionpoints
  - Dictates the size of the Actionpoint Definition and Results Tables
- LC application state after power-on reset
- LC application state when CDS has been restored
- Default Watchpoint Definition Table (WDT) filename
- Default Actionpoint Definition Table (ADT) filename
- Maximum ADT reverse polish (RPN) equation size (operators and operands)
- Maximum ADT actionpoint event text string length
- Maximum RTS ID allowed during ADT validation
- Floating Point Comparison Tolerance

LC Commands

Command	Description
No-op	Increments the Command Accepted Counter and sends an event message with application version information
Reset Counters	Initializes housekeeping counters
Set LC State	Sets the LC application state (Active, Passive, Disabled)
Set AP State	Sets the state of one or all actionpoints (Active, Passive, Disabled)
Set AP Permanently Off	Sets the state of a single actionpoint to permanently off (requires table load to restore)
Reset AP Statistics	Reset statistics in the Actionpoint Results Table (ART) for one or all actionpoints
Reset WP Statistics	Reset statistics in the Watchpoint Results Table (WRT) for one or all watchpoints

LC Housekeeping Telemetry Message

Telemetry Point	Description
CmdCount	Number of accepted ground commands
CmdErrCount	Number of rejected ground commands
APSampleCount	Total count of actionpoints sampled
MonitoredMsgCount	Total count of messages monitored
RTSExecCount	Total count of RTS sequences initiated
PassiveRTSExecCount	Total count of RTS sequences not initiated because either the LC application state or the state of the actionpoint that failed is set to Passive
WPsInUse	How many watchpoints are currently defined
ActiveAPs	How many actionpoints are currently set active
CurrentLCState	Current LC application operating state (Active, Passive, Disabled)
WPResults	Packed subset of Watchpoint Results Table (see next slide)
APResults	Packed subset of Actionpoint Results Table (see next slide)

LC Housekeeping Results Data

- **WPResults**
  - Byte array with 2 bits per watchpoint (aligned to nearest longword boundary)
  - Most recent watchpoint comparison result (2 bits)
    - 0 = False, 1 = True, 2 = Error, 3 = Stale
  - Ordering: (R_{wp3}, R_{wp2}, R_{wp1}, R_{wp0}), (R_{wp7}, R_{wp6}, R_{wp5}, R_{wp4}), etc...
- **APResults**
  - Byte array with 4 bits per actionpoint (aligned to nearest longword boundary)
  - Actionpoint current state (2 bits)
    - 0 = Unused or Permanently Off, 1 = Active, 2 = Passive, 3 = Disabled
  - Most recent actionpoint analysis result (2 bits)
    - 0 = Pass, 1 = Fail, 2 = Error, 3 = Stale
  - Ordering: (S_{ap1}, R_{ap1}, S_{ap0}, R_{ap0}), (S_{ap3}, R_{ap3}, S_{ap2}, R_{ap2}), etc...

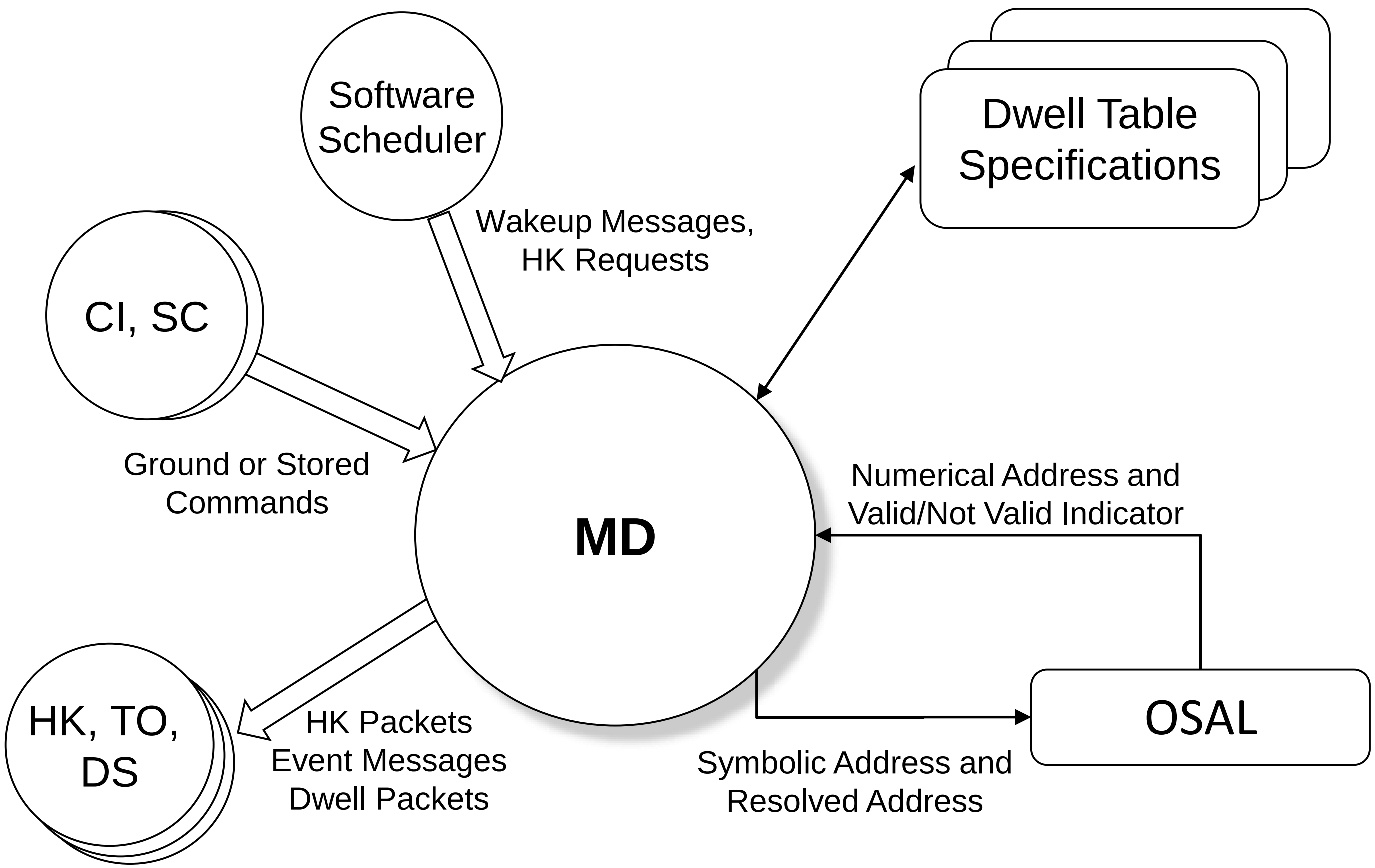
National Aeronautics and Space Administration

Memory Dwell (MD)

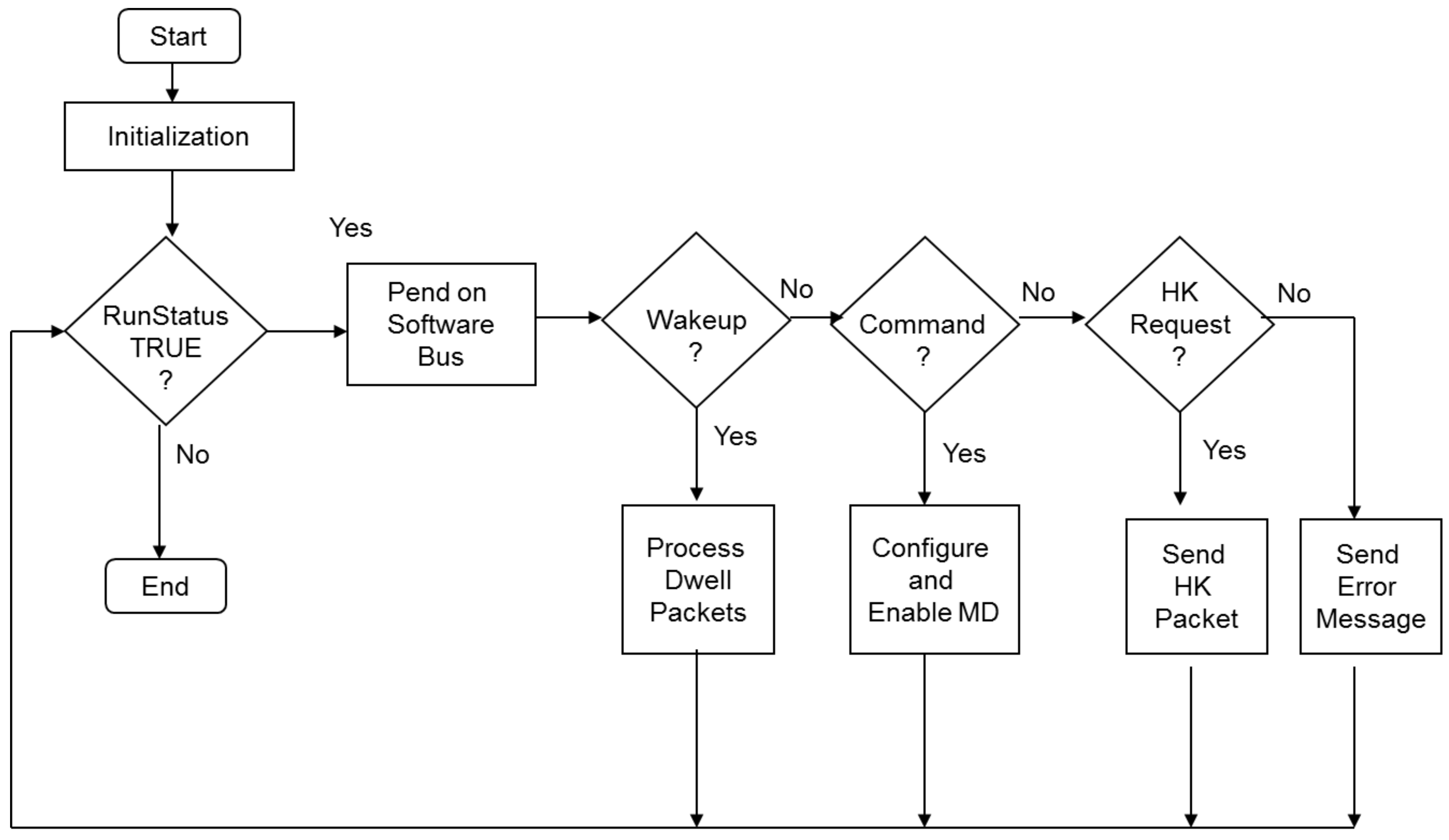
MD Overview

- **Samples and reports data from any memory address**
  - Used to augment telemetry stream provided during development
  - Supports debugging efforts
- **Dwell packet streams are specified by Dwell Tables**
  - Up to four active Dwell Tables
    - The size of the Dwell Tables is defined by a configuration parameter
    - Each entry contains:
      - Memory Address
        - <OPTIONAL>Provide Support for Symbolic Addressing
      - # Bytes to Read (1..4)
      - Delay until Next Dwell (in multiples of wake-up call rate)
    - Dwell Tables can be populated either by Table Loads or via Jam Commands
  - Each active table generates a telemetry message at the scheduled wake-up frequency (dwell rate)
    - Size of the message is based on the number of table specified memory addresses

MD Context Diagram



MD Flow Control



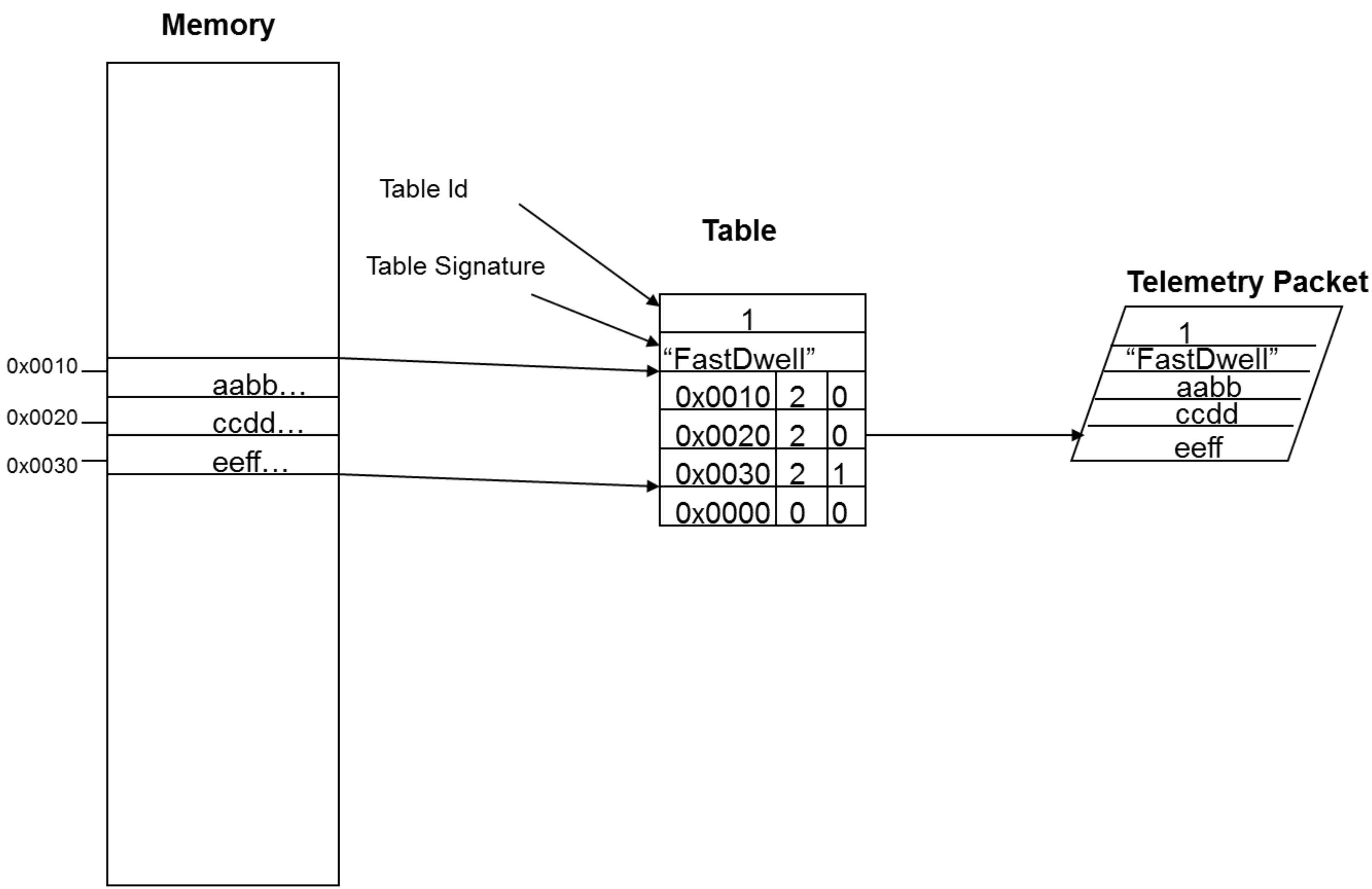
MD – Dwell Tables

Table Id =1		
Table Signature="FastDwell"		
Address 1 =0x0010	Field Length 1= 2	Delay 1=0
Address 2 =0x0020	Field Length 2= 2	Delay 2=0
Address 3 =0x0030	Field Length 3 = 2	Delay 3=1
Address 4 =0x0000	Field Length 4 = 0	Delay 4=0
…	…	…
Address 63	Field Length 63	Delay 63
Address 64	Field Length 64	Delay 64

This Dwell Table is configured to output 1 dwell output packet with each wakeup call, containing values for memory addresses 0x0010, 0x0020, and 0x0030.

Note that "Field Length 4=0" designates end of active table, making Address 3 the last address that will be read.

MD Dwell Example



MD – Configuration Parameters

- Command Pipe Depth
- Number of Dwell Tables
- Size of Dwell Tables
- Enforce Double Word Alignment
- Signature Option
- Signature Length

MD Commands

Command	Description
Noop	Increment commands accepted counter and send event message
Reset Counters	Reset housekeeping telemetry counters
Start (Individual) Table Dwell	Enable processing of the specified dwell table. Populates 1 pkt immediately; then initiates countdown timer.
Stop (Individual) Table Dwell	Disable processing of the specified dwell table
Jam Dwell Entry	Modify one address in the specified dwell table
Set Dwell Table Signature	Define a signature for a specified dwell table which will be inserted in dwell packets derived from the table

MD Housekeeping Telemetry Message

Telemetry Point	Description
MD_CMDPC	Count of valid commands received
MD_CMDEC	Count of invalid commands received
MD_ENABLEMASK	Dwell status bits for global dwell enable and all dwell table active flags (0x1=TBL1, 0x2=TBL2, 0x4=TBL3, 0x8=TBL4, 0x10 TBL5, 0x20 TBL6, etc… up to TBL16)
MD_ADDRCNT[NUM_DWELL_TBLS]	Number of dwell addresses in table
MD_RATE[NUM_DWELL_TBLS]	Total of delay counts (in active entries) in table
MD_DATASIZE [NUM_DWELL_TBLS]	Number of bytes of data specified by table
MD_DWPKTOFFSET[NUM_DWELL_TBLS]	Current write offset within dwell packet data region
MD_DWTBLENTRY[NUM_DWELL_TBLS]	Next dwell table entry to be processed
MD_COUNTDOWN[NUM_DWELL_TBLS]	Current value of countdown timer

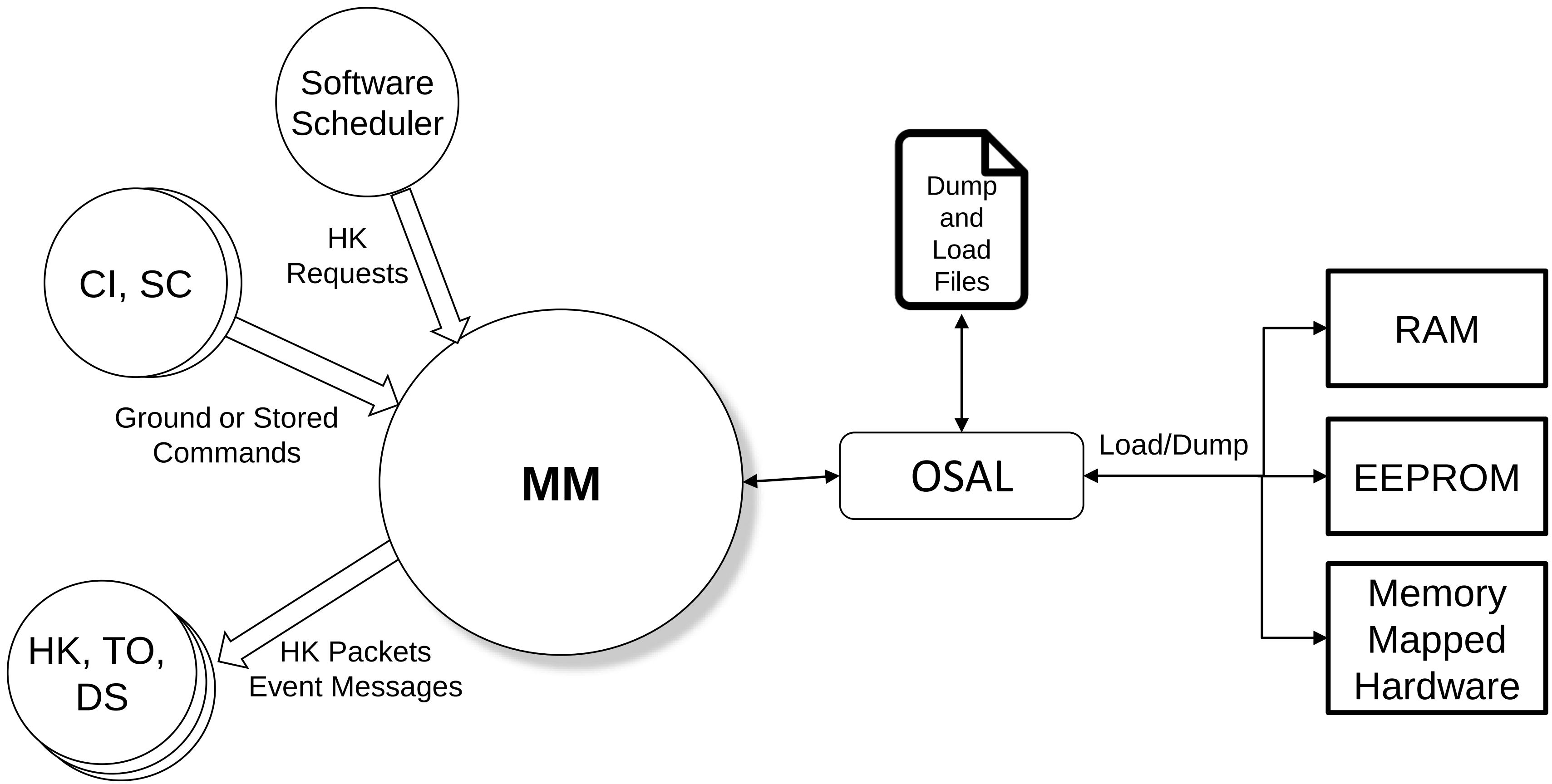
Memory Manager (MM)

MM Overview

- **Performs Memory Read and Write (Peek and Poke) Operations**
  - Peek 8, 16, or 32 bits of data
  - Poke 8, 16, or 32 bits of data
- **Performs Memory Load and Dump Operations**
  - Memory load with interrupts disabled
  - Memory load from a file
  - Memory dump to a file

    Segmented to prevent CPU hogging
  - Dump memory in an event message
- **Performs Diagnostic Operations**
  - Memory fill
  - <OPTIONAL> Load or dump address range with forced 8, 16, or 32 bit wide access
- **<OPTIONAL> Provide Support for Symbolic Addressing**
  - Any address can be referenced via a symbol name
  - Dump system symbol table or lookup symbol address

MM Context Diagram



MM Memory Types - 1

- **Standard Memory Types**
  - RAM
  - EEPROM
    - Any write alignment requirements for EEPROM are handled inside the OS abstraction layer for the target platform in question and are transparent to MM
- **Special Types (can be conditionally compiled in when needed)**
  - MEM8
    - Forces memory read and writes in 8 bit chunks only
  - MEM16
    - Forces memory read and writes in 16 bit chunks only
  - MEM32
    - Forces memory read and writes in 32 bit chunks only
  - Specified byte counts and addresses must be properly aligned for these memory types or an error event message will be generated and the operation aborted
- **Memory mapped I/O is accessed as RAM (or as MEM32/16/8 when special data alignment is required)**

MM Memory Types - 2

Type[3]	Calls Used	Address and Data Size Alignment Forced by MM	Peek/Poke via command	File support (load/dump from file?)	Multiple bytes with interrupts disabled?	Symbolic Addressing (if applicable)[4]	Fill
RAM	CFE_PSP_MemCpy[1]	No	Yes	Yes	Yes	Yes	Yes
EEPROM	CFE_PSP_MemCpy[2]	No	Yes	Yes	No	Yes	Yes
MEM32	OS_MemRead32 OS_MemWrite32	Yes	Yes	Yes	No	Yes	Yes
MEM16	OS_MemReadl6 OS_MemWrite16	Yes	Yes	Yes	No	Yes	Yes
MEM8	OS_MemRead8 OS_MemWrite8	Yes	Yes	Yes	No	Yes	Yes

[1] Memory mapped I/O that is byte addressable and requires no special code support will be accessed as standard RAM

[2] The CFE_PSP_MemCpy routine handles hardware-specific data alignment requirements for EEPROM writes. MM enables/disables EEPROM write protection (from ground command request) via calls to the cFE PSP.

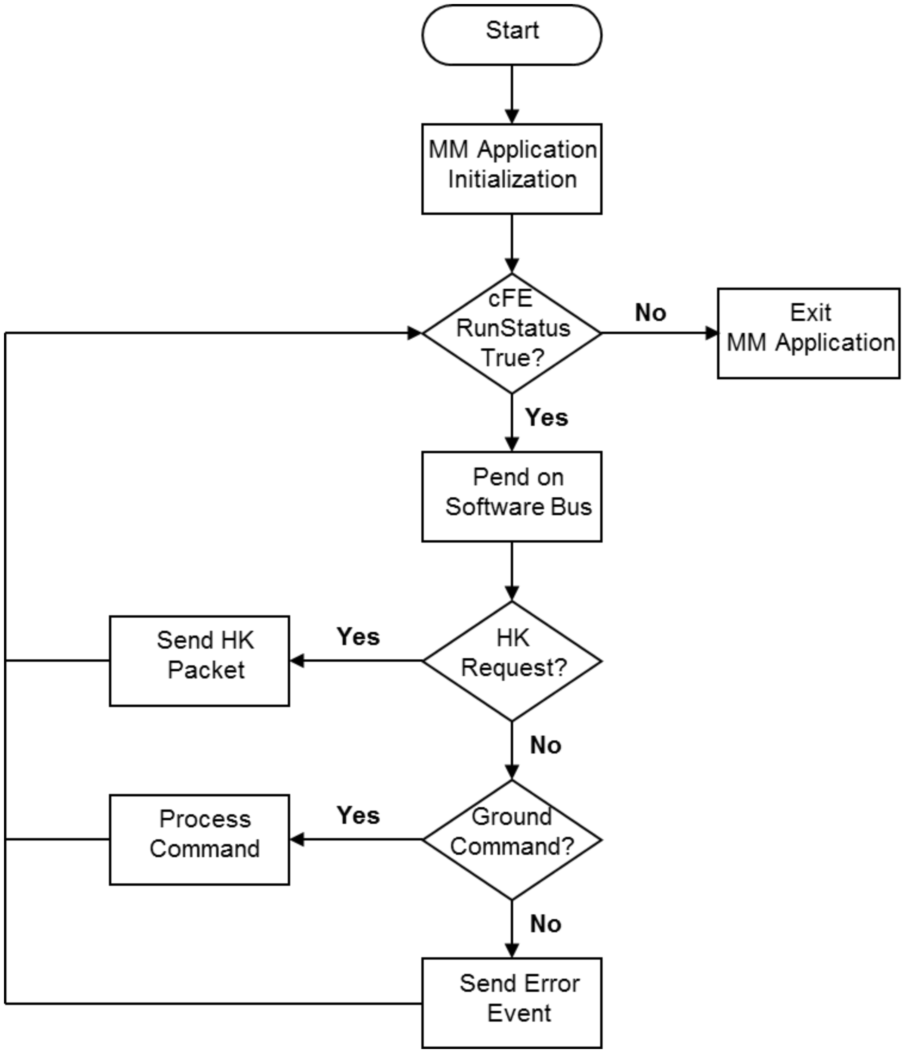
[3] MEM32, MEM16, and MEM8 memory types are optional and can be compiled out of MM.

[4] Symbolic addressing is optional. In order to simplify the code, the infrastructure will be there within commands/telemetry definitions regardless of whether a platform supports it or not. However, if a platform does not support symbolic addressing and someone tries to use symbolic addressing, MM will report the error.

MM File Formats

- **Memory Load File**
  - Binary File
    - Includes cFE file header with secondary file header containing:
      - o Destination Address Symbolic Name
      - o Destination Address Offset
      - o Destination Memory Type
      - o Number of Load Bytes
      - o Data Integrity Value (CRC on load data)
- **Memory Dump File**
  - Binary File
    - Includes cFE file header with secondary file header containing:
      - o Address Symbolic Name (NUL string)
      - o Source Address (fully resolved, absolute address)
      - o Source Memory Type
      - o Number of Bytes Dumped
      - o Data Integrity Value (CRC on dumped data)
- **Dump and load files use the same format so dump files can be loaded back into memory if desired**

MM Flow Control



MM – Configuration Parameters

- **CRC type**
  - interrupt disabled loads
  - File loads
  - File dumps
- **Load Parameters**
  - Maximum number of bytes for a file load to:
  - Maximum number of bytes for an uninterruptable load
  - Maximum number of bytes to segment load operations into to avoid CPU hogging
- **Dump Parameters**
  - Maximum number of bytes for a file dump from:
  - Maximum number of bytes to segment dump operations into to avoid CPU hogging
  - The number of bytes that can be dumped in an event message is based on the maximum event message string length specified by the cFE configuration parameter CFE_EVS_MAX_MESSAGE_LENGTH
- **Fill Parameters**
  - Maximum number of bytes for a memory fill operation to:
  - Maximum number of bytes to segment fill operations into to avoid CPU hogging
- **Optional Memory Types**
  - Include/Exclude MEM32, MEM16, or MEM8 optional memory types

MM Commands - 1

Command	Description
Noop	Increments the Command Accepted Counter and sends a debug event message
Reset Counters	Initializes housekeeping counters to zero
Memory Peek	Reads 8, 16, or 32 bits of data from any address and reports the data in an event message
Memory Poke	Writes 8, 16, or 32 bits of data to any address
Load Memory With Interrupts Disabled	Loads data into memory with CPU interrupts disabled during the load
Memory Load From File	Loads memory contents from a file
Memory Dump To File	Dumps memory contents from memory to a file
Dump Memory In Event	Dumps a series of data bytes from memory into telemetry as ASCII characters in an event message. The maximum number of bytes that can be transferred is limited to the maximum event message string length specified in the cFE configuration parameters

MM Commands - 2

Command	Description
Memory Fill	Loads memory with a 32 bit fill pattern. For MEM16 and MEM8 memory types only the least significantword (MEM16) or byte (MEM8) of the fill pattern is used.
Lookup Symbol	Looks up a symbol name in the system symbol table and reports the resolved address in an event message and housekeeping
Save Symbol Table To File	Saves the system symbol table to an onboard file if the operation is supported by the target operating system

MM Housekeeping Telemetry Message

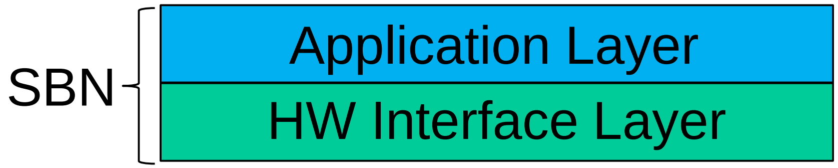
Telemetry Point	Description
CmdCounter	Number of accepted ground commands
ErrCounter	Number of rejected ground commands
LastAction	Last command action executed
MemType	Memory type for last command
Address	Fully resolved address used for last command
FillPattern	Fill pattern used if memory fill command was issued
BytesProcessed	Bytes processed for last command
FileName[OS MAX PATH LEN]	Name of the data file used for last command, when applicable

Software Bus Network (SBN)

SBN Overview

- **Transparently connects Software Bus local systems**
  - Multi-processor, multi-core, partitions, distributed systems
  - Applications have no knowledge of destination
  - Requires a single namespace (unique message IDs across system)
- **Peer to Peer, no bus master**
- **Heartbeat algorithm to detect failed nodes**
- **Optional recognition and retransmission of missed packets**
- **Latest version with plug-in network interface software modules**
  - Currently supports UDP/IP, Shared memory, and RS422 serial
    - JSC has ARINC 653 partition support
    - Planned support for SpaceWire, Time-Triggered Ethernet and TTP/C
  - In process of being released
- **Supports communications scheduling for synchronous system behaviors**
- **Latest version available on babelfish**

Module/Plug-In Architecture



- **Application layer is interface independent, handles message protocol: message routing, heartbeating, etc.**
- **Hardware interface layer uses libraries to handle interface-specific functions**
- **All hardware interaction is done via interface "modules" (implemented as libraries)**
- **Module/plug-in architecture allows:**
  - Easy customization for different platforms
  - Easy to add support for new hardware interfaces
  - Possible in-flight modification of peers and/or modules (not currently supported, but a future expansion)

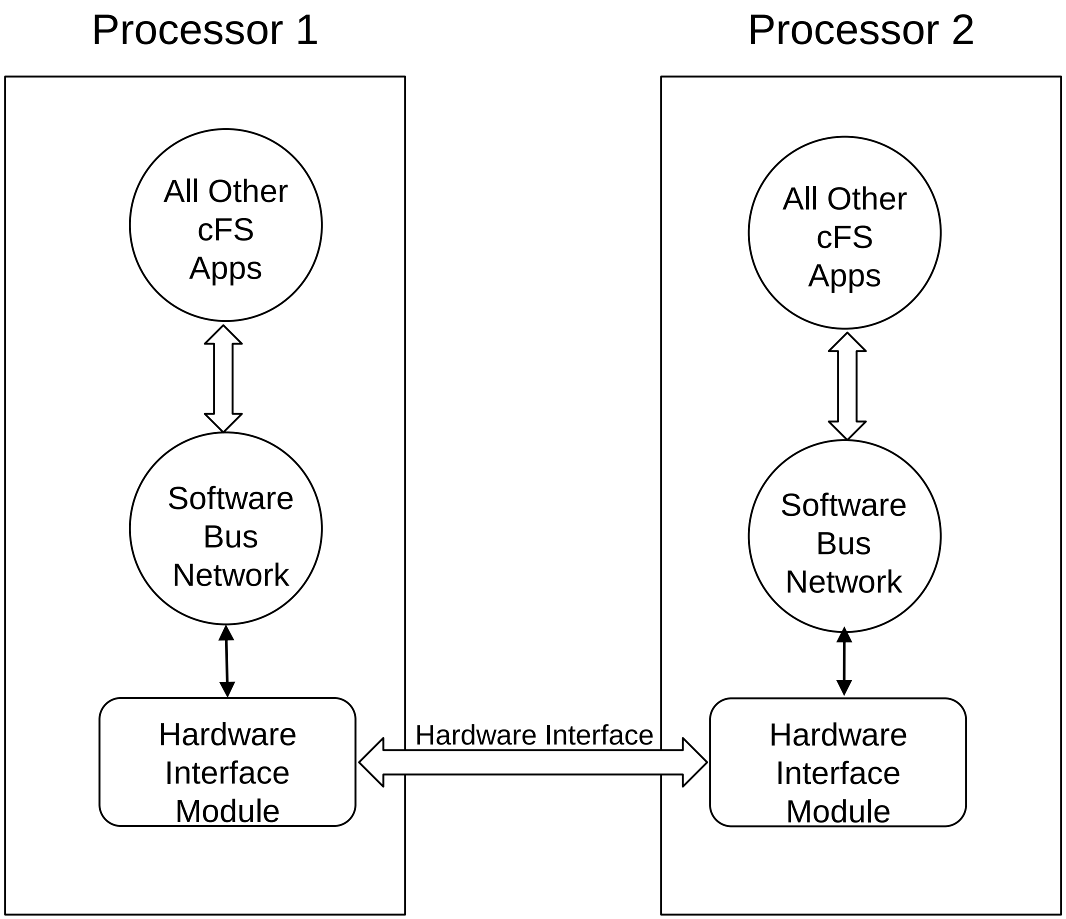
Module/Plug-In Architecture

- **All interface modules support a common API**
- **Each module can specify custom configuration parameters and custom housekeeping telemetry**
- **Modules are loaded by SBN on application startup**
  - Requires dynamic loading

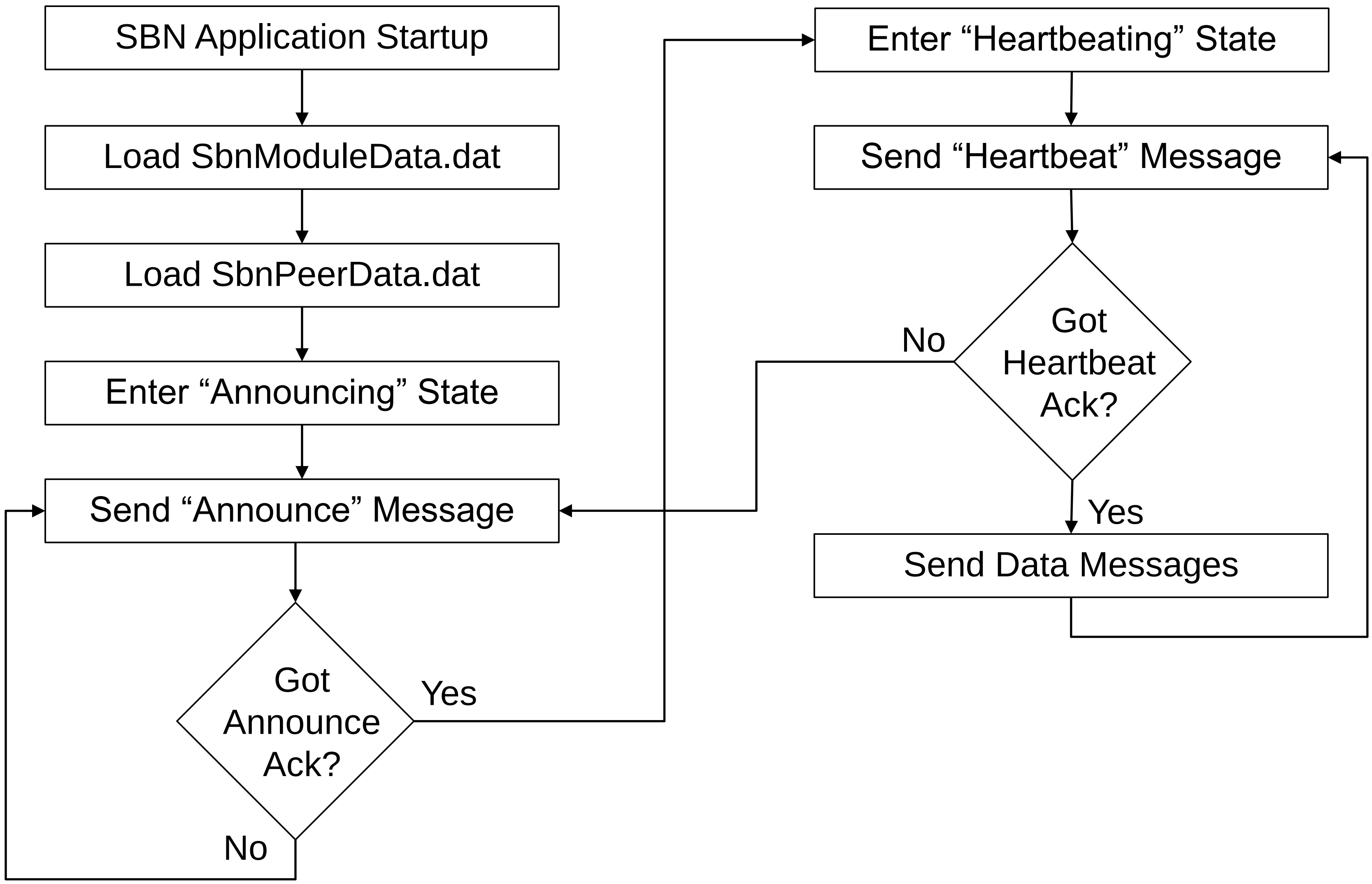
Module API

- **All interface modules must support the following functions:**
  - Parse File Entry
  - Initialize Peer Interface
  - Send Net Message
  - Check for Net Protocol Message
  - Receive Message
  - Verify Peer Interface
  - Verify Host Interface
  - Report Module Status
  - Reset Peer
  - Delete Host Resources
  - Delete Peer Resources
- **Function pointers are stored in a structure in the module source code.**
- **The name of the structure is included in the SbnModuleData.dat file**
- **The SBN application does an object load to get the operations structure**

SBN Context


Processor 1
Processor 2
All Other cFS Apps
Software Bus Network
Hardware Interface Module
Hardware Interface
All Other cFS Apps
Software Bus Network
Hardware Interface Module

SBN Startup


SBN Application Startup
Load SbnModuleData.dat
Load SbnPeerData.dat
Enter “Announcing” State
Send “Announce” Message
Got Announce Ack?
Yes
No
Enter “Heartbeating” State
Send “Heartbeat” Message
Got Heartbeat Ack?
No
Yes
Send Data Messages

SBN Message Types

- **Protocol Messages – Messages sent to establish and maintain link between peers.**
  - Announce – Announces presence to peers to establish link
  - Announce Acknowledge – Acknowledgement sent to a peer after receiving an announce message from that peer
  - Heartbeat – Aliveness message sent to all peers to maintain link
  - Heartbeat Acknowledge – Acknowledgement sent to a peer after receiving a heartbeat message from that peer
- **Data Messages – Actual Software Bus messages routed between peers**

SBN Peer Configuration

- **Peer configuration has 2 parts – peers and interface types**
- **Peer and module configurations are specified in text files**
- **SbnModuleData.dat specifies interface modules in use by SBN**
  - Protocol ID, Module Location, Interface Structure
- **SbnPeerData.dat specifies peers connected to this processor and correlates each peer with an interface type**
  - Peer Name, Processor ID, Protocol ID, SpaceCraft ID, Protocol Parameters
    - Each Protocol ID must match a protocol ID listed in SbnModuleData.dat
    - Protocol parameters vary according to the Protocol ID

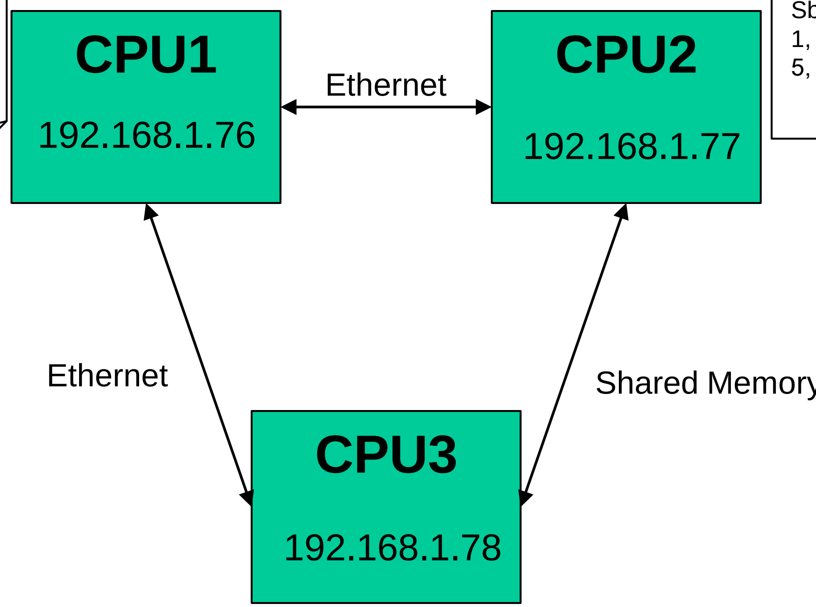
SBN Peer Configuration Example

SbnPeerData.dat
CPU1, 1, 1, 0, 192.168.1.76, 15820, 5821;
CPU2, 2, 1, 0, 192.168.1.77, 15820, 5822;
CPU3, 3, 1, 0, 192.168.1.78, 15820, 5823;

SbnModuleData.dat
1, IPv4, /cf/apps/ipv4.so, IPv4Ops;

SbnPeerData.dat
CPU1, 1, 1, 0, 192.168.1.76, 15820, 5821;
CPU2, 2, 1, 0, 192.168.1.77, 15820, 5822;
CPU2, 2, 5, 0, 0xfffe9000, 0x1000, 0xfffe8000, 0x1000, 0xfffeb000, 0x1000, 0xfffea000, 0x1000;
CPU3, 3, 5, 0, 0xfffe8000, 0x1000, 0xfffe9000, 0x1000, 0xfffea000, 0x1000, 0xfffeb000, 0x1000;

SbnModuleData.dat
1, IPv4, /cf/apps/ipv4.so, IPv4Ops;
5, ShMem, /cf/apps/shmem.so, ShMemOps;



SbnPeerData.dat
CPU1, 1, 1, 0, 192.168.1.76, 15820, 5821;
CPU3, 3, 1, 0, 192.168.1.78, 15820, 5823;
CPU2, 2, 5, 0, 0xfffe9000, 0x1000, 0xfffe8000, 0x1000, 0xfffeb000, 0x1000, 0xfffea000, 0x1000;
CPU3, 3, 5, 0, 0xfffe8000, 0x1000, 0xfffe9000, 0x1000, 0xfffea000, 0x1000, 0xfffeb000, 0x1000;

SbnModuleData.dat
1, IPv4, /cf/apps/ipv4.so, IPv4Ops;
5, ShMem, /cf/apps/shmem.so, ShMemOps;

SBN Configuration Parameters

Parameter	Default	Description
SBN_SUB_PIPE_DEPTH	256	Depth of SBN subscription pipe
SBN_VOL_PEER_FILENAME	"/ram/apps/SbnPeerData.dat"	Location of peer configuration file in volatile memory
SBN_NONVOL_PEER_FILENAME	"/cf/apps/SbnPeerData.dat"	Location of peer configuration file in nonvolatile memory
SBN_PEER_FILE_LINE_SIZE	128	Maximum line length in peer configuration file
SBN_MAX_NETWORK_PEERS	4	Maximum number of peers
SBN_VOL_MODULE_FILENAME	"/ram/apps/SbnModuleData.dat"	Location of module configuration file in volatile memory
SBN_NONVOL_MODULE_FILENAME	"/cf/apps/SbnModuleData.dat"	Location of module configuration file in nonvolatile memory
SBN_MODULE_FILE_LINE_SIZE	128	Maxmum line length in peer configuration file
SBN_MAX_INTERFACE_TYPES	6	Maximum number of interface types
SBN_MOD_STATUS_MSG_SIZE	128	Maximum number of bytes in a module status message
SBN_MAX_MSG_RETRANSMISSIONS	3	Number of times the SBN will try to retransmit a missed message

SBN Commands

Command	Description
Noop	Increments the Command Accepted Counter and sends a debug event message
Reset Command Counters	Initializes the following SBN counters to 0: Command Rejected Counter, Command Accepted Counter, Peer Send/Receive counters
Get Peer List	Gets a list of all peers recognized by the SBN.
Get Peer Status	Get status information on the specified peer. Information format is based on the interface type of the peer.
Reset Peer	Resets a specified peer.

SBN Housekeeping Telemetry

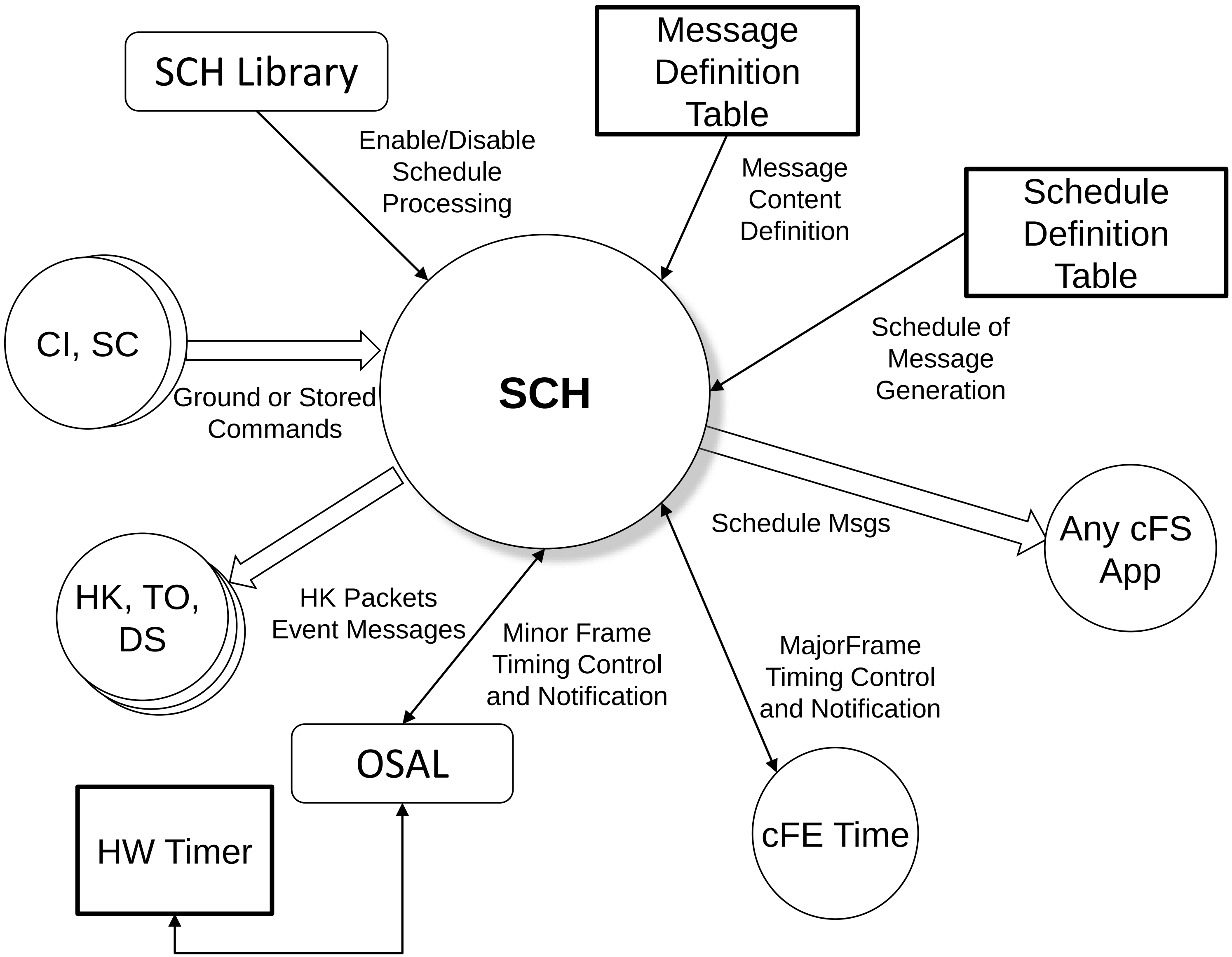
Telemetry Point	Description
CmdCount	Number of commands accepted by the SBN application
CmdErrCount	Number of commands rejected by the SBN application
PeerAppMsgRecvCount[SBN_MAX_NETWORK_PEERS]	Number of application messages received by each peer
PeerAppMsgSendCount[SBN_MAX_NETWORK_PEERS]	Number of application messages sent by each peer
PeerAppMsgRecvErrCount[SBN_MAX_NETWORK_PEERS]	Number of application message receive errors for each peer
PeerAppMsgSendErrCount[SBN_MAX_NETWORK_PEERS]	Number of application message send errors for each peer
PeerProtoMsgRecvCount[SBN_MAX_NETWORK_PEERS]	Number of protocol messages received by each peer
PeerProtocolMsgSendCount[SBN_MAX_NETWORK_PEERS]	Number of protocol messages sent by each peer
PeerProtocolMsgRecvErrCount[SBN_MAX_NETWORK_PEERS]	Number of protocol message receive errors for each peer
PeerProtocolMsgSendErrCount[SBN_MAX_NETWORK_PEERS]	Number of protocol message send errors for each peer

Scheduler (SCH)

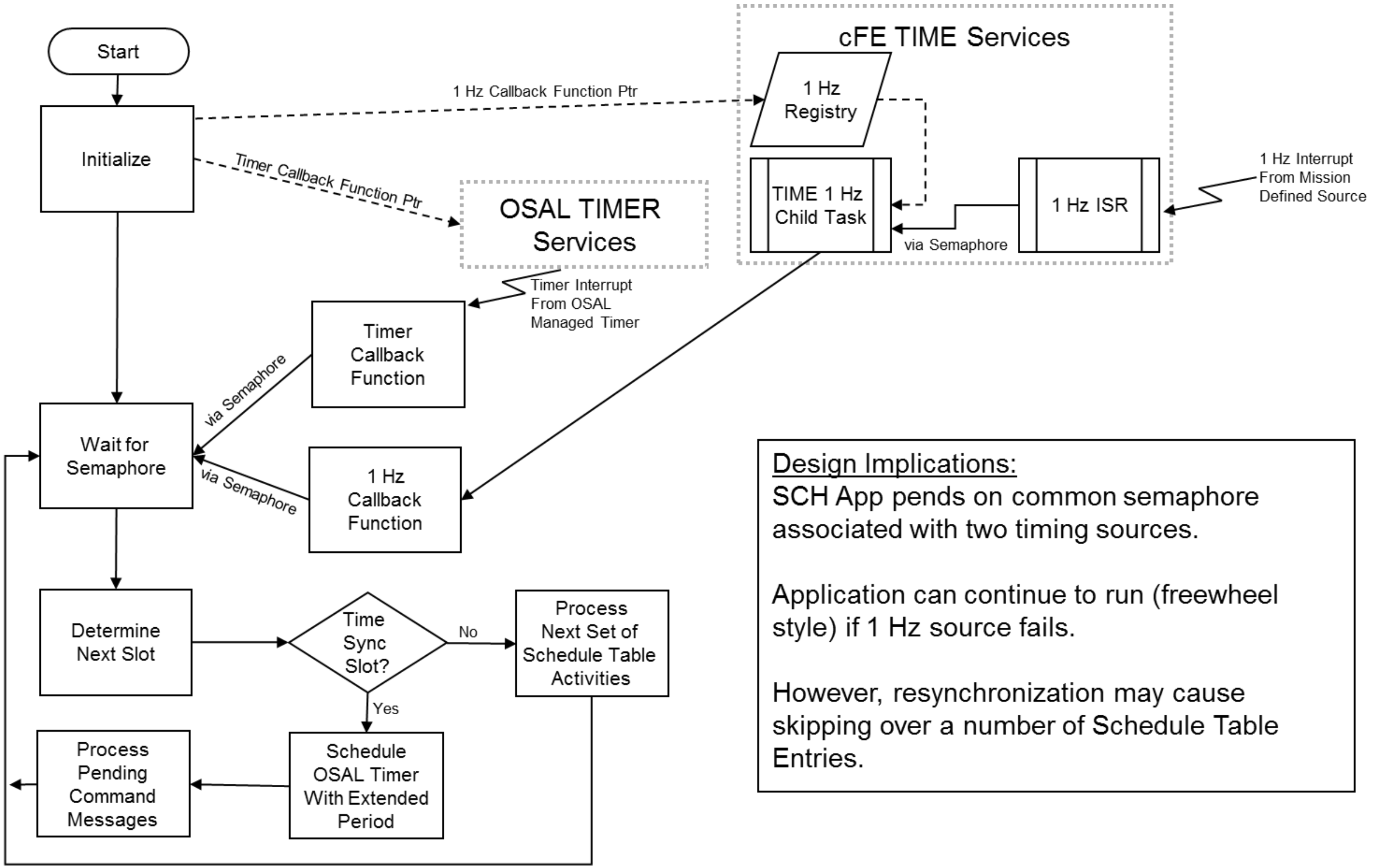
SCH Overview

- **Provides method of generating messages at pre-determined timing intervals**
  - Operates in a Time Division Multiplexed (TDM) fashion with deterministic behavior
    - Synchronized to external Major Frame cFE TIME 1 Hz signal
    - Each Major Frame split into a platform configuration number of smaller slots (typically 100 slots of 10 milliseconds each)
      - Each slot can contain a platform configuration number of software bus messages that can be issued within that slot

SCH Context Diagram



SCH Flow Control



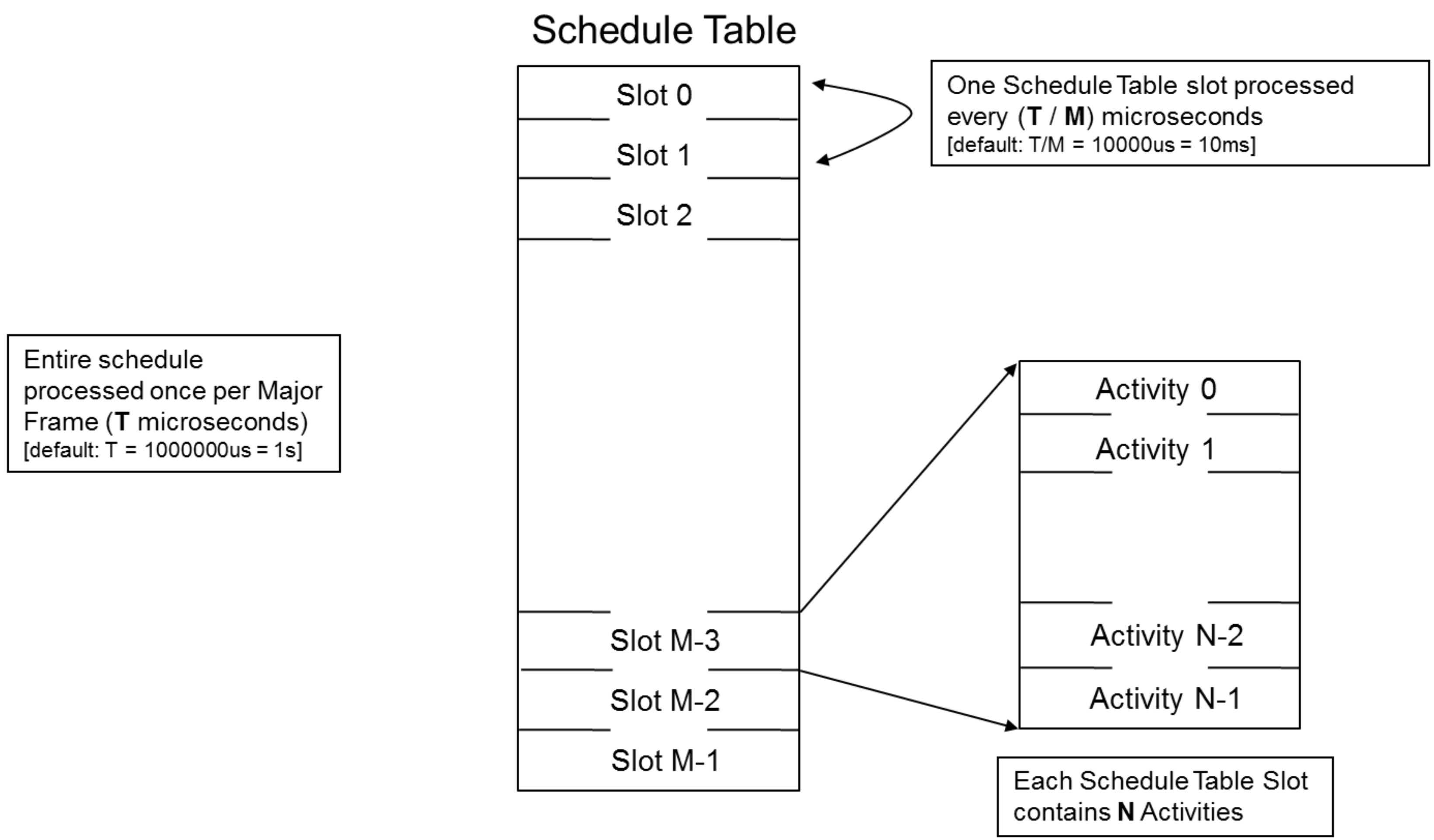
<u>Design Implications:</u>

SCH App pends on common semaphore associated with two timing sources.

Application can continue to run (freewheel style) if 1 Hz source fails.

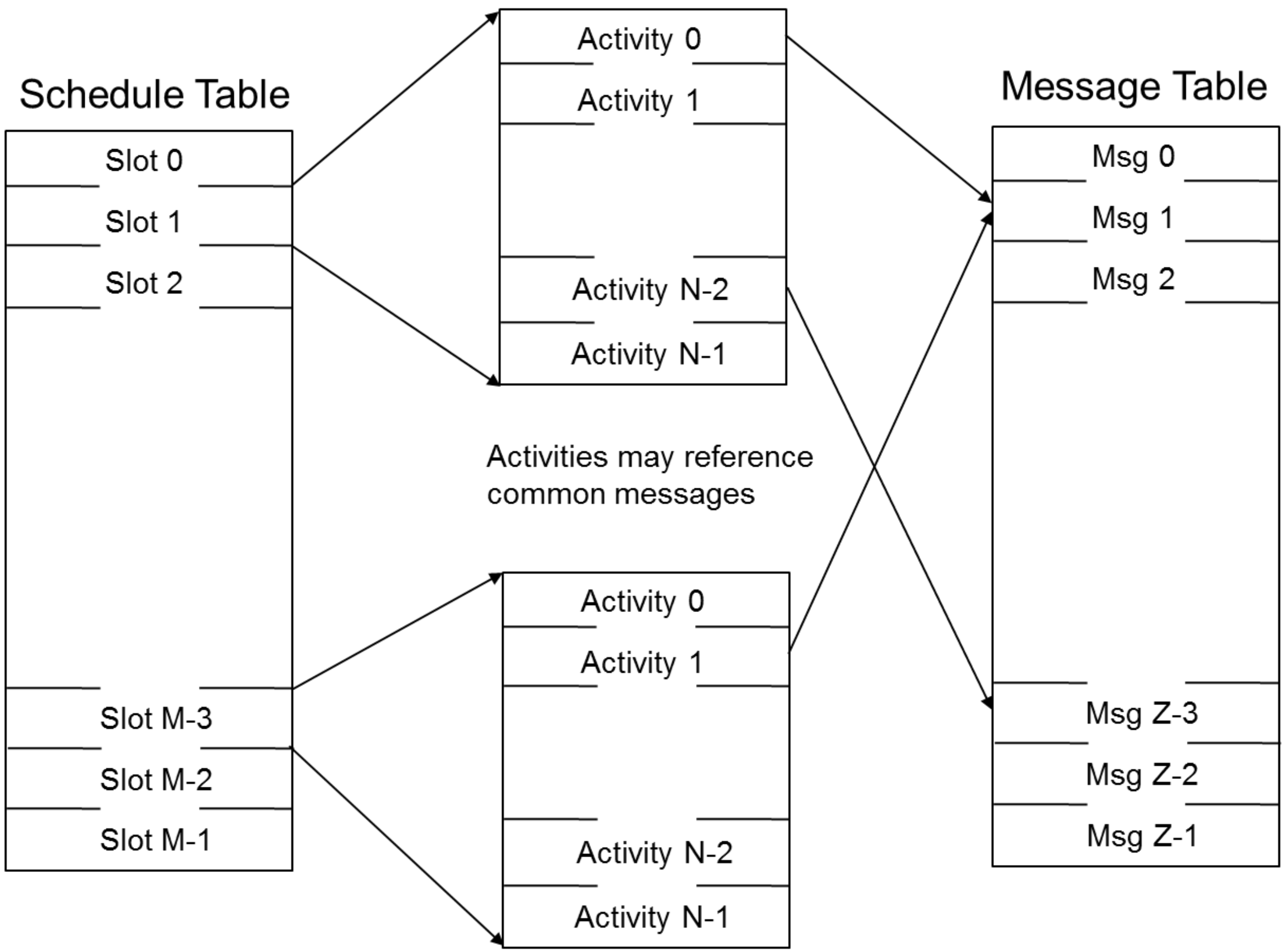
However, resynchronization may cause skipping over a number of Schedule Table Entries.

SCH Schedule Table Processor



- SCH maintains a count of number of times entire table is processed and reports it in housekeeping telemetry

SCH – Activity Messages



SCH Configuration Parameters - 1

Parameter	Description	Default Value
SCH_PIPE_DEPTH	Software bus command pipe depth	12
SCH_TOTAL_SLOTS	Minor Frame Frequency (in Hz)	100
SCH_ENTRIES_PER_SLOT	Maximum Activities per slot	5
SCH_MAX_MESSAGES	Maximum Number of Message Definitions in Message Table	128
SCH_MDT_MIN_MSG_ID	Minimum Message ID allowed in Message Definition Table	0
SCH_MDT_MAX_MSG_ID	Maximum Message ID allowed in Message Definition Table	CFE_SB_HIGHEST_VALID_MSGID
SCH_MAX_MSG_WORDS	Maximum Length, in Words, of a Message in the message table	64
SCH_MAX_LAG_COUNT	Maximum Number of slots allowed for catch-up before skipping	(SCH_TOTAL_SLOTS/2)
SCH_MAX_SLOTS_PER_WAKEUP	Maximum Number of Slots to be processed when in "Catch Up" mode	5
SCH_MICROS_PER_MAJOR_FRAME	Conversion factor for how many microseconds in a wake-up period	10000000

SCH Configuration Parameters - 2

Parameter	Description	Default Value
SCH_SYNC_SLOT_DRIFT_WINDOW	Additional time allowed in Sync Slot to wait for Major Frame Sync	5000
SCH_STARTUP_SYNC_TIMEOUT	Timeout on waiting for all applications to start at initialization	50000
SCH_STARTUP_PERIOD	Number of microseconds to attempt major frame synchronization	(5*SCH_MICROS_PER_MAJOR_FRAME)
SCH_MAX_NOISY_MAJORF	Maximum noisy major frames prior to desynchronization	2
SCH_LIB_PRESENCE	Presence of SCH Library	1
SCH_LIB_DIS_CTR	Processing disabled counter at startup	0
SCH_SCHEDULE_FILENAME	Default schedule table filename to load at startup	“/cf/apps/sch _def_schtbl.tbl”
SCH_MESSAGE_FILENAME	Default message table filename to load at startup	“/cf/apps/sch _def_msgtbl.tbl"
SCH_MISSION_REV	Mission revision number	0

SCH Commands

Command	Description
No-op	Increments the Command Accepted Counter and sends a debug event message
Reset Counters	Initializes housekeeping counters to zero
Enable Entry	Enables an entry in the Schedule Definition Table
Disable Entry	Disables an entry in the Schedule Definition Table
Enable Group and/or Multi-Group(s)	Enables a group and/or multi-group(s) of entries in the Schedule Definition Table
Disable Group and/or Multi-Group(s)	Disables a group and/or multi-group(s) of entries in the Schedule Definition Table
Enable Sync	Enables usage of Major Frame Signal if previously autonomously disabled for being "noisy"
Send Diagnostic Tim	Generates and sends the SCH Diagnostic Telemetry Packet that contains the current state of all activities defined in the Schedule Definition Table

SCH Housekeeping Telemetry Message - 1

Telemetry Point	Description
CommandCounter	Number of accepted ground commands
CommandErrCounter	Number of rejected ground commands
ScheduleActivitySuccessCounter	Number of scheduled activities processed
ScheduleActivityFailureCounter	Number of scheduled activities failed due to error
SlotsProcessedCounter	Number of schedule slots processed
SlotsSkippedCounter	Number of instances when one or more slots were skipped
MultipleSlotsCounter	Number of instances when two or more slots were processed at once
SameSlotCounter	Number of instances when SCH woke up in the same time slot as previously
BadTableDataCount	Number of table entries with an error that have been encountered
TableVerifySuccessCount	Number of successful table verifications performed
TableVerifyFailureCount	Number of failed table verifications performed
TablePassCounter	Number of times Schedule Table was completely processed
ValidMajorFrameCount	Number of Valid Major Frame Signals received
MissedMajorFrameCount	Number of Major Frame Signals that did not occur when expected
UnexpectedMajorFrameCount	Number of Major Frame Signals that occurred when nor expected to occur
MinorFramesSinceTone	Number of Minor Frames processed since last Major Frame

SCH Housekeeping Telemetry Message - 2

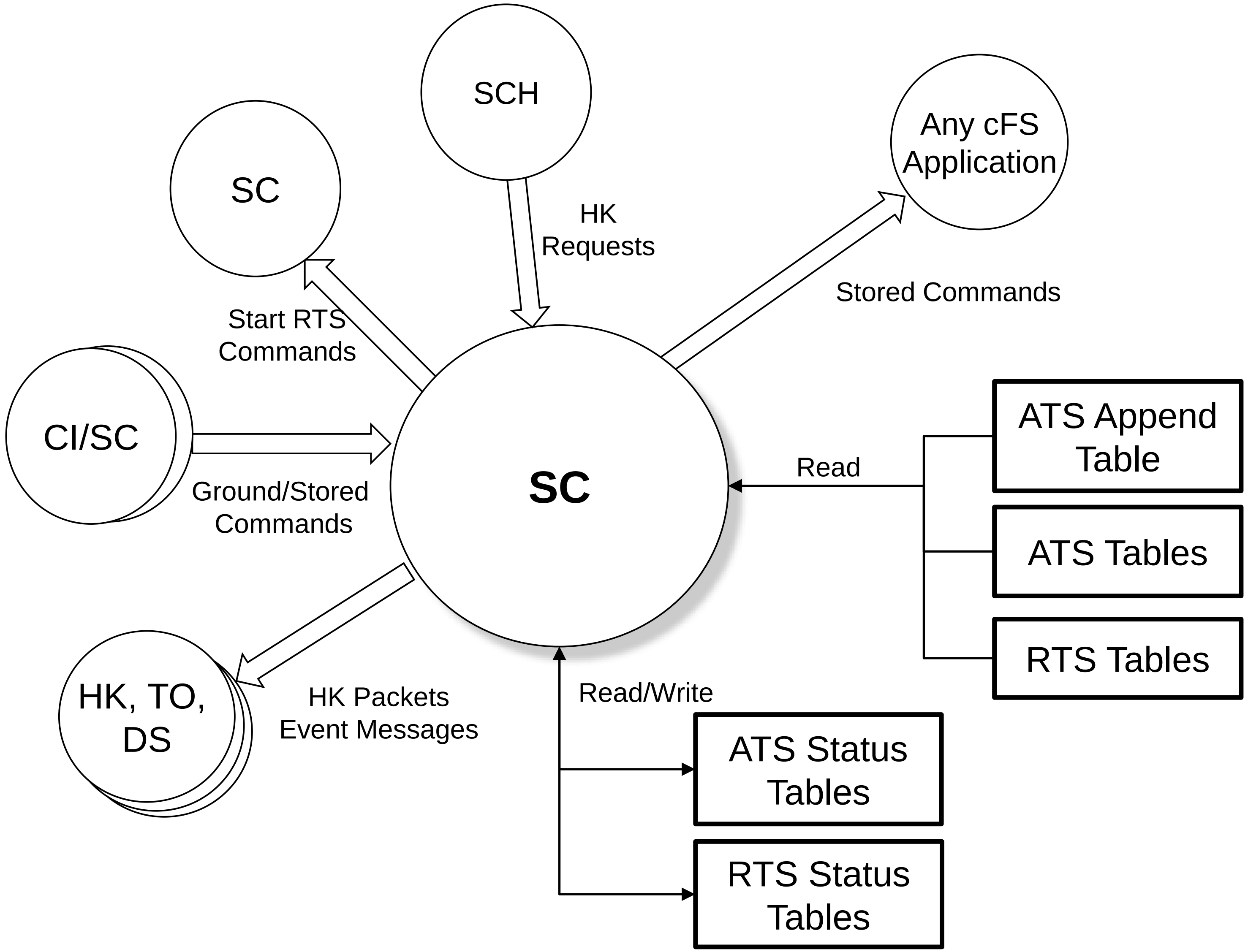
Telemetry Point	Description
NextSlotNumber	The next slot to be processed in the Schedule Definition Table
LastSyncMETSlot	Slot Number when last Time Synchronization occurred
IgnoreMajorFrame	Major Frame Signals are ignored because they are deemed "noisy"
UnexpectedMajorFrame	Last Major Frame Signal occurred when not expected
SyncToMET	Minor Frames are synchronized to MET.
MajorFrameSource	Identifies the source of the Major Frame Signal (timer, MET, etc)

Stored Command (SC)

SC Overview

- **Provides services to execute preloaded, table defined command sequences at predetermined absolute or relative time intervals**
  - Supports two types of time tagged command sequences
    - ATSs are command sequences timed to execute at some absolute point in time, as measured by the configured time
      - Supports 2 ATSs
      - One second granularity
      - Records table processing status in a dump-only status table
    - RTSs are command sequences which execute at some point in time, relative to the previous command in the relative time command sequence
      - Supports <platform defined> RTSs
      - One second granularity
      - Records table processing status in a dump-only status table

SC Context Diagram



SC ATS Table Processor

- **The ATS Processor manages the execution of the Absolute Time Sequences**
- **The ATS Processor manages two buffers of ATSs**
- **Only one ATS can be active at one time**
- **The ATS Processor can be controlled by requests from the ground**
  - Start an ATS
  - Stop an ATS
  - Switch the ATS buffer
  - Jump within an ATS
  - Continue on ATS Failure
  - Append ATS

SC RTS Table Processor

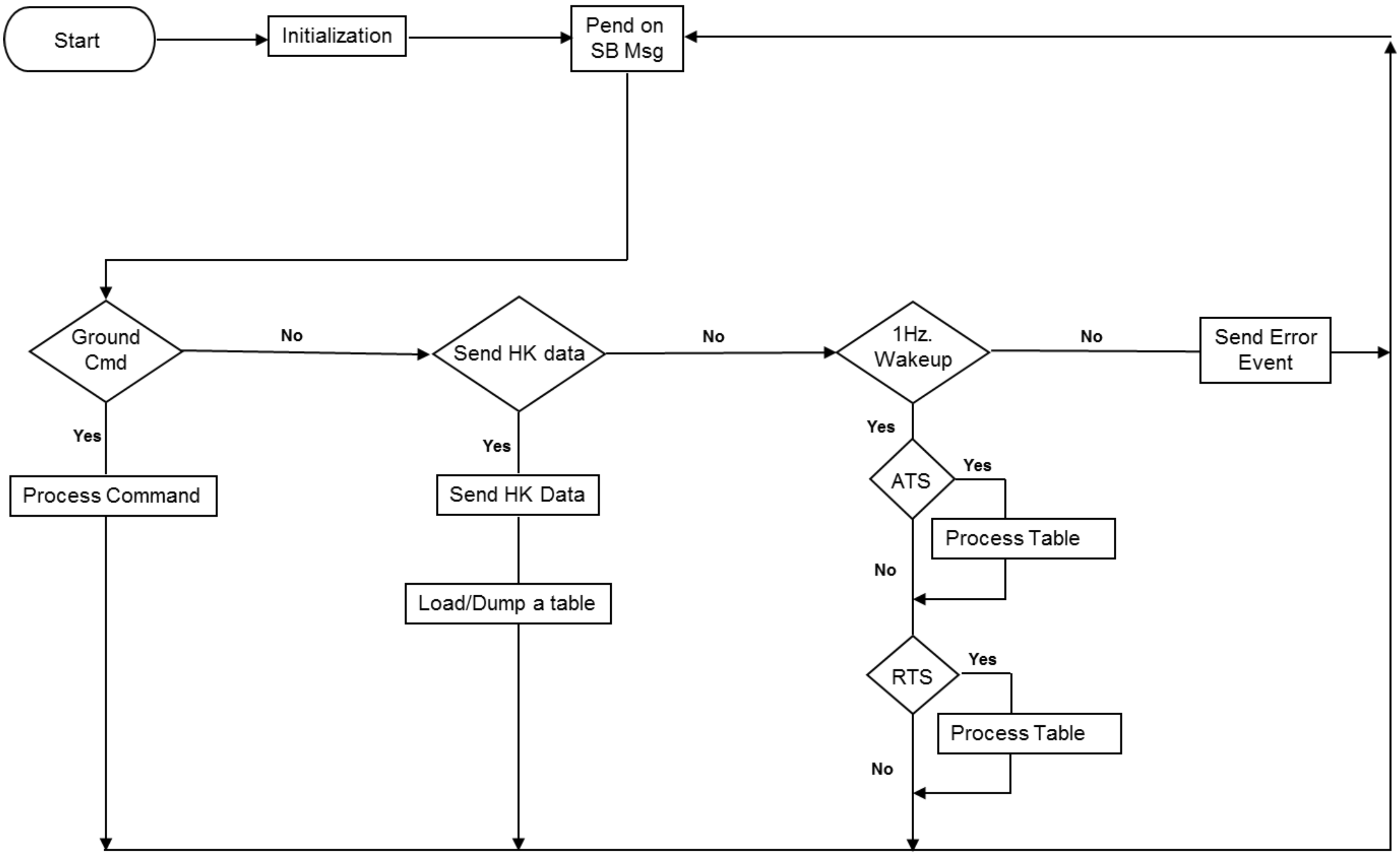
- **The RTS Processor is controlled by requests from:**
  - Ground
  - FSW applications (LC, HK)
- **The RTS Processor can be commanded to:**
  - Start an RTS
  - Stop an RTS
  - Enable an RTS
  - Disable an RTS
- **RTS #1 is executed at initialization**
  - Contain the startup sequence

SC Status Tables

ATS Status Tables	Parameters
SC AtsCmdStatusIndexTable	Array of unsigned bytes which are the number of ATS buffers multiplied by the number of ATS commands. LOADED, EMPTY, EXECUTED etc
SC_AtpControlBlock_t	ATP execution state of the ATP ATS number currently running if any ATS Command number to run if any Time index pointer for the current command Switch Pend Flag
SC_AtsInfoTable_t	Number of commands in the ATS Size of the ATS How many time the ATS has been used

RTS Status Tables	Parameters
SC RtpControlBlock t	Number of RTSs currently active Next RTS number to execute
SC RtsInfoEntry t	Status of the RTS Disabled/Enabled flag for the current RTS Next command time for an RTS Where the next RTS command is in the buffer Number of Errors in the current RTS How many times an RTS ran

SC Flow Control



SC Configuration Parameters - 1

Parameter	Description	Default Value
SC_MAX_CMDS_PER_SEC	Maximum number of commands that can be sent out by SC in any given second	8
SC_NUMBER_OF_RTS	The number of RTS's allowed in the system	64
SC_ATS_BUFF_SIZE	The max sizeof an ATS buffer in words (not bytes)	8000
SC_APPEND_BUFF_SIZE	The max sizeof an Append ATS buffer in words (not bytes)	4000
SC_RTS_BUFF_SIZE	The max size of an RTS buffer in WORDS (not bytes)	150
SC_MAX_ATS_CMDS	The maximum number of commands that are allowed in each ATS	1000
SC_LAST_RTS_WITH_EVENTS	When all RTS's are started, the SC_RTS_START_INF_EID event message is sent out. This parameter suppresses that message for all RTS's over this number	20
SC_PACKET_MIN_SIZE	This parameter specifies the maximum size for an ATS or RTS command	250
SC_PIPE_DEPTH	Maximum number of messages that will be allowed in the SC command pipe at one time	12
SC_ATS_FILE_NAME	Base filename for the ATS tables loaded at startup	/cf/apps/sc_ats
SC_APPEND_FILE_NAME	Default append ATS filename loaded at startup	/cf/apps/sc_append.tbl
SC_RTS_FILE_NAME	Base filename for the RTS tables loaded at startup	/cf/apps/sc_rts

SC Configuration Parameters - 2

Parameter	Description	Default Value
SC_ATS_TABLE_NAME	Base name for unique ATS table object names	ATS_TBL
SC_APPEND_TABLE_NAME	Unique table object name for the Append ATS table	APPEND_TBL
SC_RTS_TABLE_NAME	Base name for unique RTS table object names	RTS_TBL
SC_RTSINFO_TABLE_NAME	Name of the RTS Information Table	RTSINF_TBL
SC_RTP_CTRL_TABLE_NAME	Name of the RTP Control Block Table	RTPCTR_TBL
SC_ATSINFO_TABLE_NAME	Name of the ATS Information Table	ATSINF_TBL
SC_APPENDINFO_TABLE_NAME	Name of the Append ATS Information Table	APPINF_TBL
SC_ATS_CTRL_TABLE_NAME	Name of the ATP Control Block Table	ATPCTR_TBL
SC_ATS_CMD_STAT_TABLE_NAME	The prefix of the ATS Command Status table names	ATSCMD_TBL
SC_CONT_ON_FAILURE_START	Specifies the default state to continue an ATS when a command in the ATS fails checksum validation	TRUE
SC_TIME_TO_USE	Defines the TIME SC should use for its commands	SC_USE_CFE_TIME
SC_ENABLE_GROUP_COMMANDS	Specifies the inclusion state RTS group commands	TRUE
SC_MISSION_REV	Mission specific revision number	0

SC Commands - 1

Command	Description
No-op	Increments the command counter and generates an informational event
Reset counters	Resets telemetry counters to zero
Start ATS	Start the specified ATS
Stop ATS	Stop the current executing ATS
Switch ATS	Switch from the currently executing ATS to the alternate ATS
ATS Jump	Jump to a specified time in the currently running ATS. All commands prior to the specified jump time will not be executed
Continue ATS Execution On A Checksum Failure	Sets the status of the CONTINUE ATS ON FAILURE flag. When the SC Flight Software encounters a failure in the execution of ATS it shall continue or abort the ATS execution based on the status of CONTINUE ATS ON FAILURE
Append ATS	Append the contents of the Append Table to the specified ATS

SC Command - 2

Command	Description
Enable RTS	Enable the specified RTS for execution
Disable RTS	Disable the specified RTS
Start RTS	Start the specified RTS
Stop RTS	Stop the specified RTS

SC Housekeeping Telemetry Message - 1

Telemetry Point	Description
CmdErrCtr	Number of ground commands aborted
CmdCtr	Number of ground commands successfully executed
AtpFreeBytes[0]	Number of free bytes in ATS A
AtpFreeBytes[1]	Number of free bytes in ATS B
AtsNumber	Currently executing ATS (none, A , B)
AtpState	Current ATS state: IDLE, EXECUTING
AtpCmdNumber	Next ATS command number
AtsNumber	Current ATS Number: NONE , ATS A , ATS B

SC Housekeeping Telemetry Message - 2

Telemetry Point	Description
SwitchPendFlag	Indication of ATS switch pending
NextAtsTime	Next ATS command time in seconds
ContinueAtsOnFailure	When the SC Flight Software encounters a failure in the execution of ATS it shall continue or abort the ATS execution based on the status of this flag
RtsActivErrCtr	Total count of all failed RTS activation attempts
RtsActivCtr	Total count of all RTSs successfully activated
RtsNumber	The next RTS command will come from this RTS
NextRtsTime	The configured when the next RTS command will execute
RtsExecutingStatus	This is a bit map consisting of an even number of unsigned words with one bit for each RTS. There are <platform defined>/16 or 16 unsigned words. The least significant bit of word 0 represents the bit for RTS 1, the MSB of word 0 is for RTS 16
AtpCommandNumber	Next ATS command number
AtpCommandCtr	The number of commands sent out by all ATSs. This value reflects the cumulative error count for all ATS commands sent from the ATS processor, until the counter rolls over or is reset. It is not reset by starting a new ATS

SC Housekeeping Telemetry Message - 3

Telemetry Point	Description
AtsCmdErrCtr	The number of commands with errors for all ATSs. This value reflects the cumulative error count of all ATSs run, until the counter rolls over or is reset. It is not reset by starting a new ATS
LastAtsErrCmd	The ID of the last ATC which caused an error. This value is not reset by stopping the current ATS, starting a new ATS or sending the reset command
LastAtsErrSeq	The ATS that contained the last error (none, A , B)
LastRtsErrCmd	The word offset of the last RTS command which caused an error. This value is not reset by stopping the current RTS, starting a new RTS or sending the reset command.
RtsCmdErrCtr	The number of commands with errors for ALL RTSs. This value reflects the cumulative count for all RTS commands with errors, until the counter rolls over or is reset
RtsCmdCtr	The number of commands sent out by ALL RTSs. This value reflects the cumulative count for all RTS commands sent, until the counter rolls over or is reset
LastRtsErrSeq	The RTS sequence number of the last RTS command which caused an error. This value is not reset by stopping the current RTS, starting a new RTS or sending the reset command

SC Housekeeping Telemetry Message - 3

Telemetry Point	Description
AtsCmdErrCtr	The number of commandswith errors for all ATSs. This value reflects the cumulative error count of all ATSs run, until the counter rolls over or is reset. It is not reset by starting a new ATS
LastAtsErrCmd	The ID of the lastATC which caused an error. This value is not reset by stopping the current ATS, starting a new ATS or sending the reset command
LastAtsErrSeq	The ATS that contained the last error (none, A , B)
LastRtsErrCmd	The word offset of the last RTS command which caused an error. This value is not reset by stopping the current RTS, starting a new RTS or sending the reset command.
RtsCmdErrCtr	The number of commands with errors for ALL RTSs. This value reflects the cumulative count for all RTS commands with errors, until the counter rolls over or is reset
RtsCmdCtr	The number of commands sent out by ALL RTSs. This value reflects the cumulative count for all RTS commands sent, until the counter rolls over or is reset
LastRtsErrSeq	The RTS sequence number of the last RTS command which caused an error. This value is not reset by stopping the current RTS, starting a new RTS or sending the reset command

SC Housekeeping Telemetry Message - 4

Telemetry Point	Description
RtsDisabledStatus	This is the same as the executing bit map where 0 = ENABLED and 1 = DISABLED.
NumRtsActive	Number of active RTSs
AppendAtsiD	The last ATS that was appended (none, A , B)
AppendSize	The size (in bytes) of the commands loaded in the Append Table
AppendCount	The number of commands in the Append Table
AppendLoads	The total number of loads performed to the Append Table

cFS Components Metrics

Component	Version	Logical Lines of Code	Configuration Parameters
Core Flight Executive	6.4.0	12930	General: 17, Executive Service: 46 Event Service: 5, Software Bus: 29 Table Service: 10, Time Service: 32
CFDP	2.2.1	8559	33
Checksum	2.2.0	2873	15
Data Storage	2.3.0	2429	27
File Manager	2.3.1	1853	22
Health & safety	2.2.0	1531	45
Housekeeping	2.4.0	575	8
Limit Checker	2.0.0	2074	13
Memory Dwell	2.3.0	1035	8
Memory Manager	2.3.0	1958	25
Stored Commanding	2.3.0	2314	26
Scheduler	2.2.0	1164	19

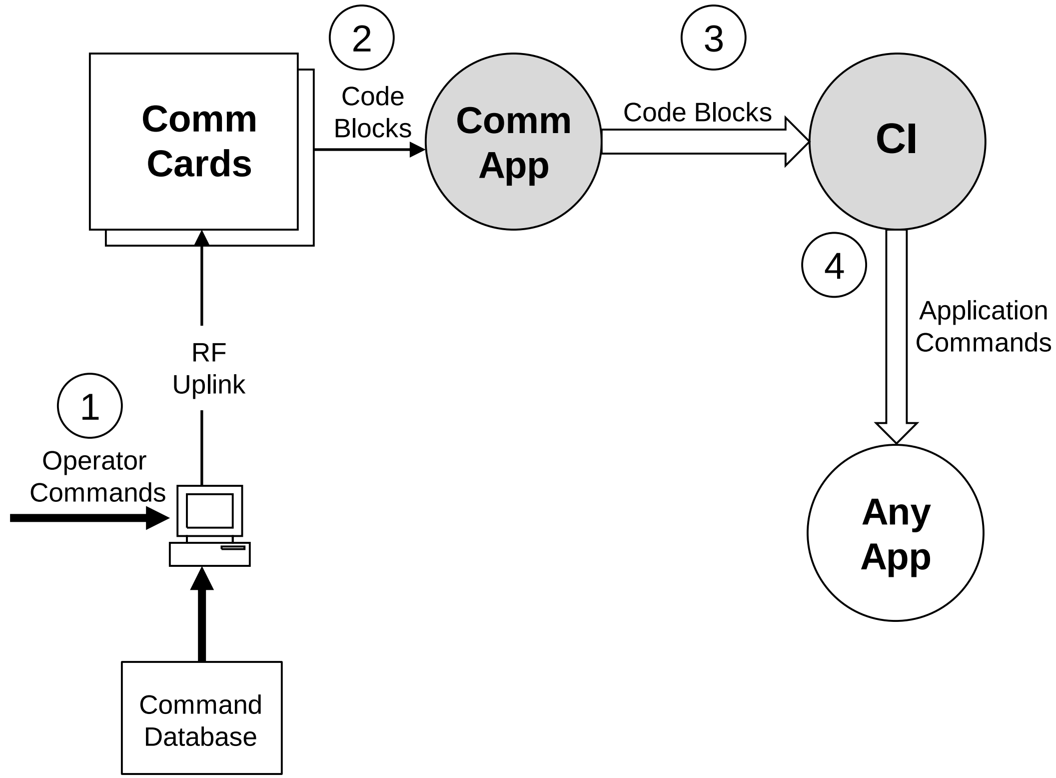
- **Two scopes of configuration parameters: mission or processor**
- **Configuration parameters span a large functional range from a simple default file name to a system behavioral definition like the time client/server configuration**

Operational Scenarios

Operational Scenarios
Uplink

* **Basic uplink from GSFC Mission Perspective**

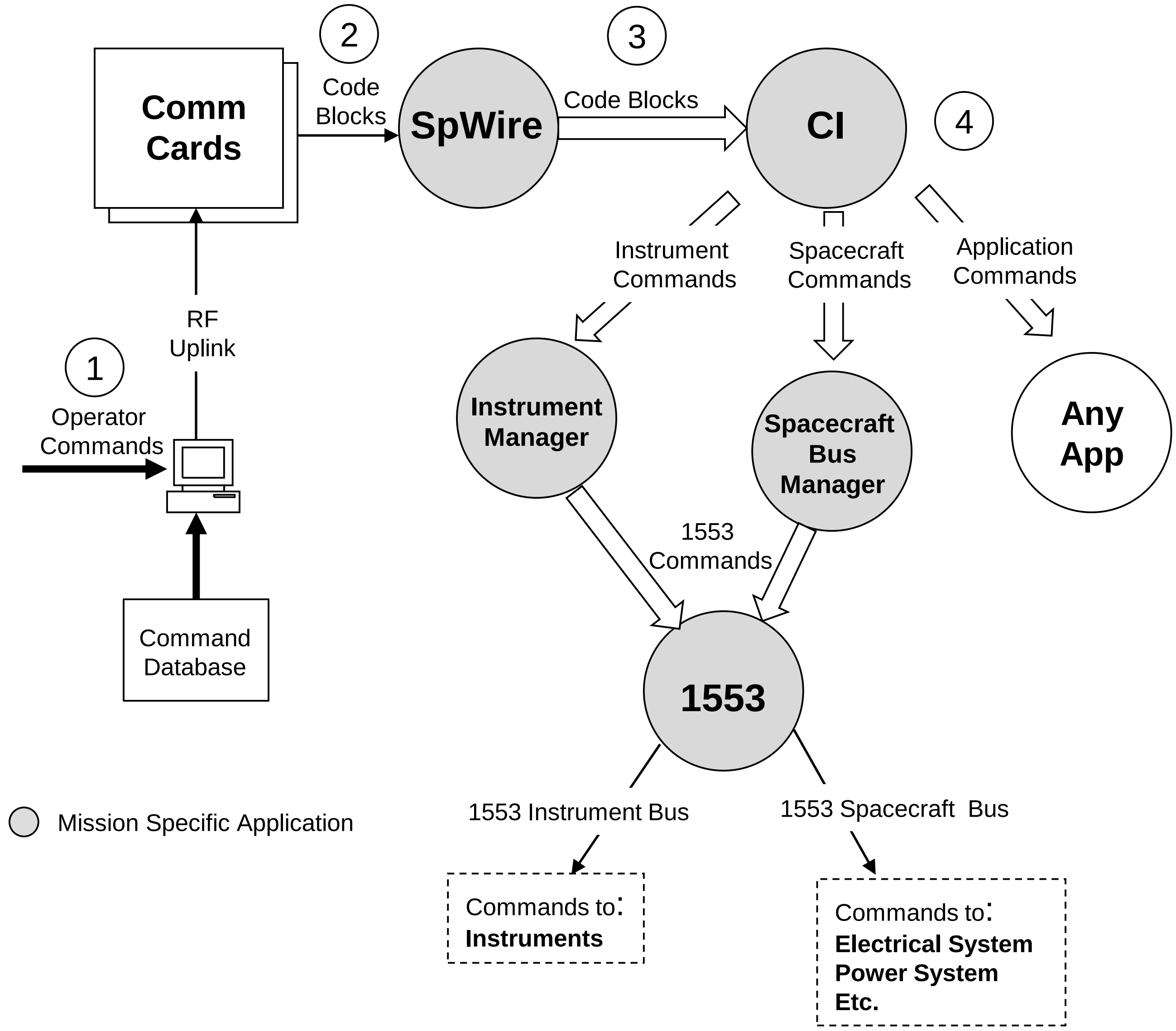
1) **Commands sent from ground system are received by communication hardware**
2) **Communication hardware processes commands received and sends code blocks to receiving application.**
3) **Communication application strips off any hardware protocol wrappers, packages Code Blocks for transfer over software bus , and forwards Code Blocks to CI application**
4) **CI assembles command packets, performs command authentication, and sends commands to subscribed applications**



Mission Specific Application

Operational Scenarios
Uplink – Command Routing Example 1

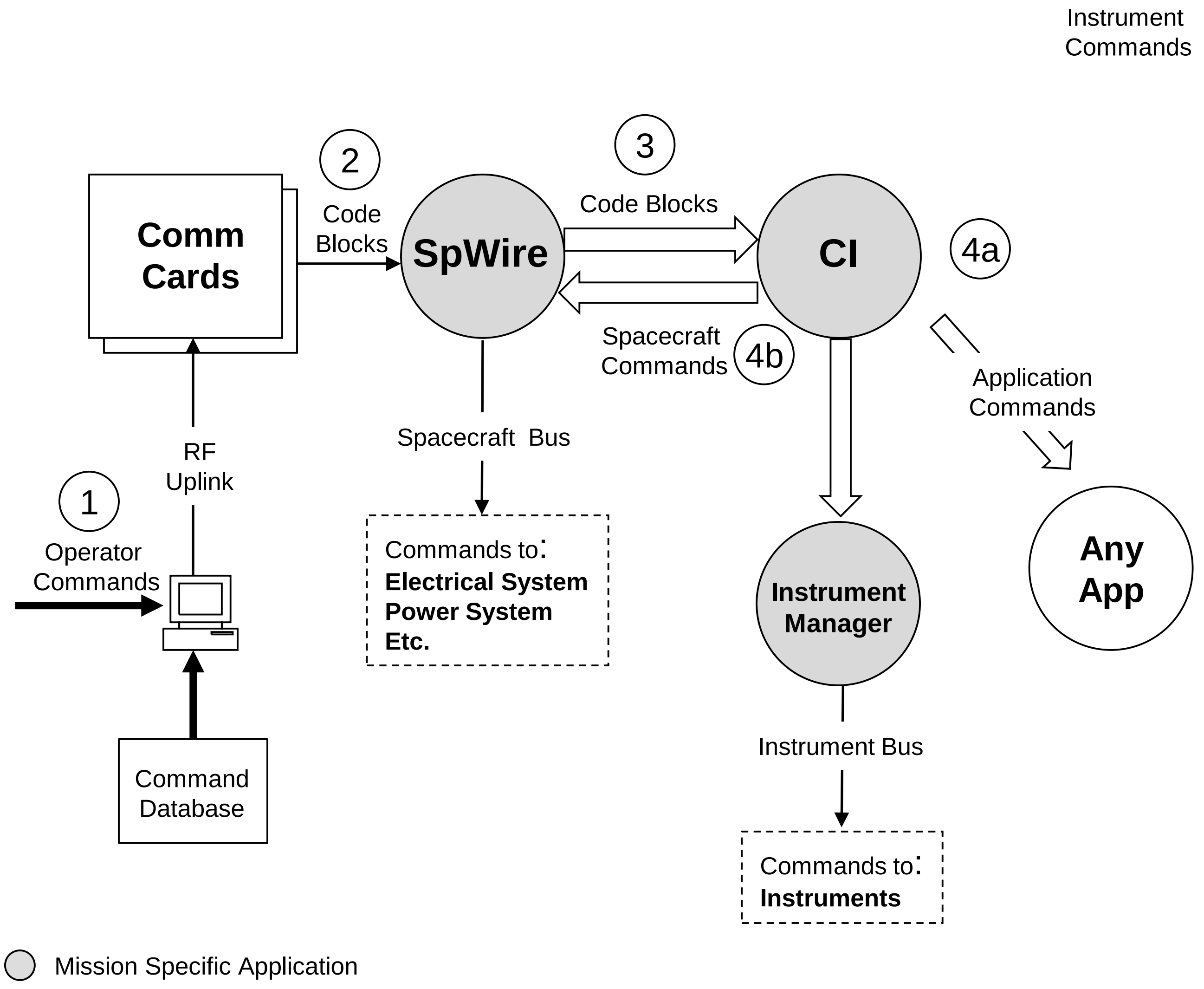
1) Commands sent from ground system are received by communication hardware
2) Communication hardware processes commands received and sends code blocks to receiving application.
3) SpWire application forwards Code Blocks to CI application
4) CI assembles command packets, performs command authentication, and sends commands to subscribed applications



*** Detailed uplink/command routing from GSFC Mission Perspective**

Operational Scenarios
Uplink – Command Routing Example 2

1) Commands sent from ground system are received by communication hardware
2) Communication hardware processes commands received and sends code blocks to receiving application.
3) SpWire application forwards Code Blocks to CI application
4) CI assembles command packets, performs command authentication, and sends commands to subscribed applications
   a) Assembled command routed directly to applications
   b) Assembled command routed back to SpWire application for distribution to Spacecraft bus across spacewire network

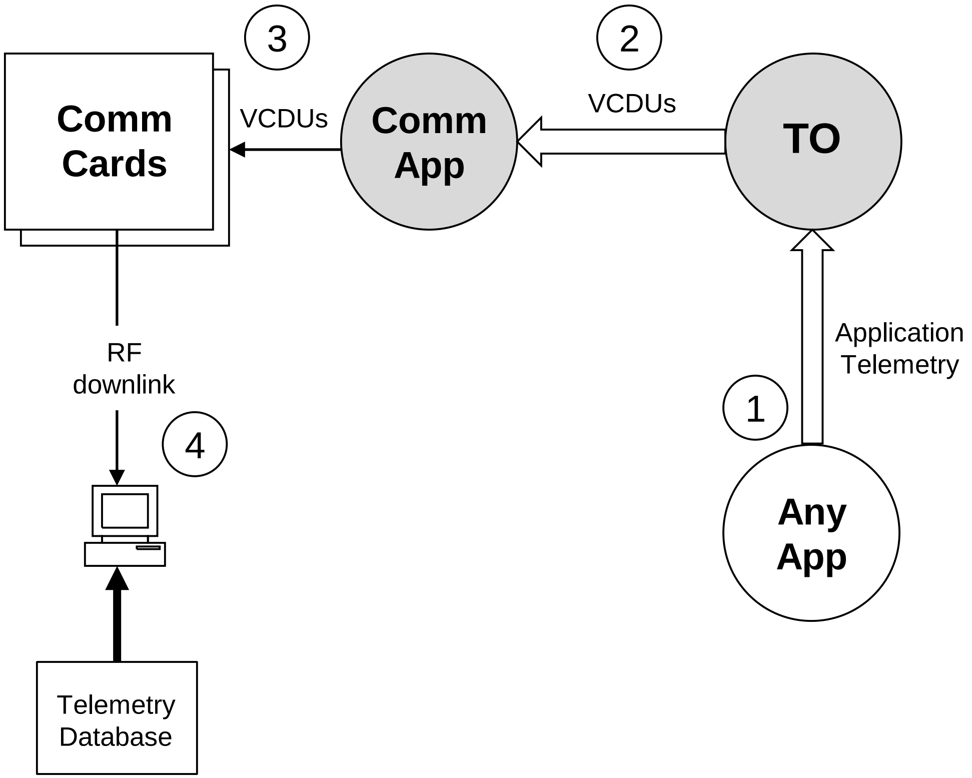


* Detailed uplink/command routing from GSFC Mission Perspective

Operational Scenarios
Telemetry Packet Downlink

* Basic downlink from GSFC Mission Perspective

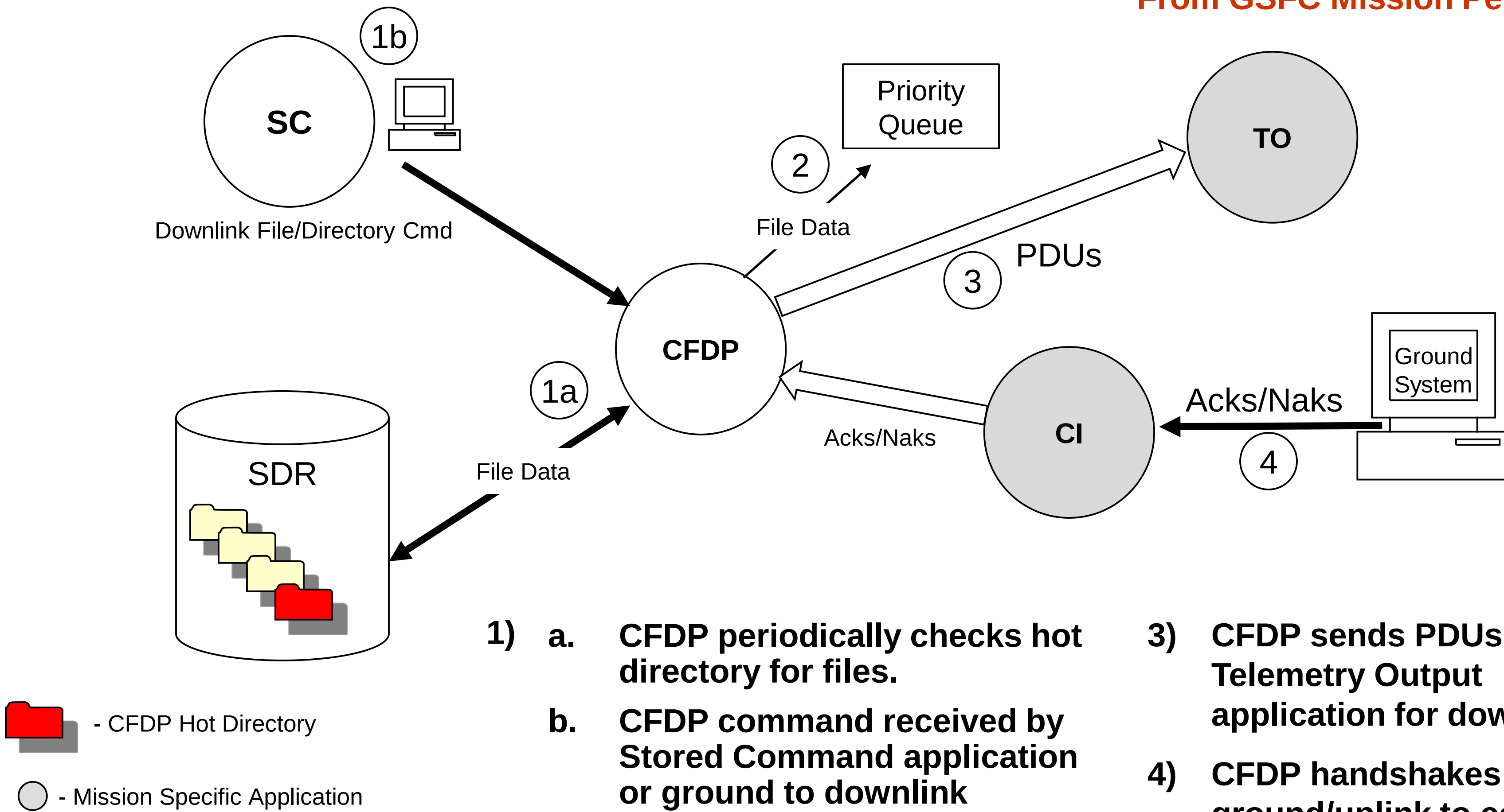
1) Telemetry is collected from the various applications in the system and routed to TO application
2) TO collects, filters, and builds real-time VCDUs for downlink. The VCDU's are packaged and routed over the software bus
3) Communication application strips off software bus headers, packages VCDUs in hardware protocol wrappers and outputs VCDUs across hardware link.
4) Telemetry is received by the ground system from communication hardware



Mission Specific Application

Telemetry and Science File Downlink

* From GSFC Mission Perspective



1) a. **CFDP periodically checks hot directory for files.**

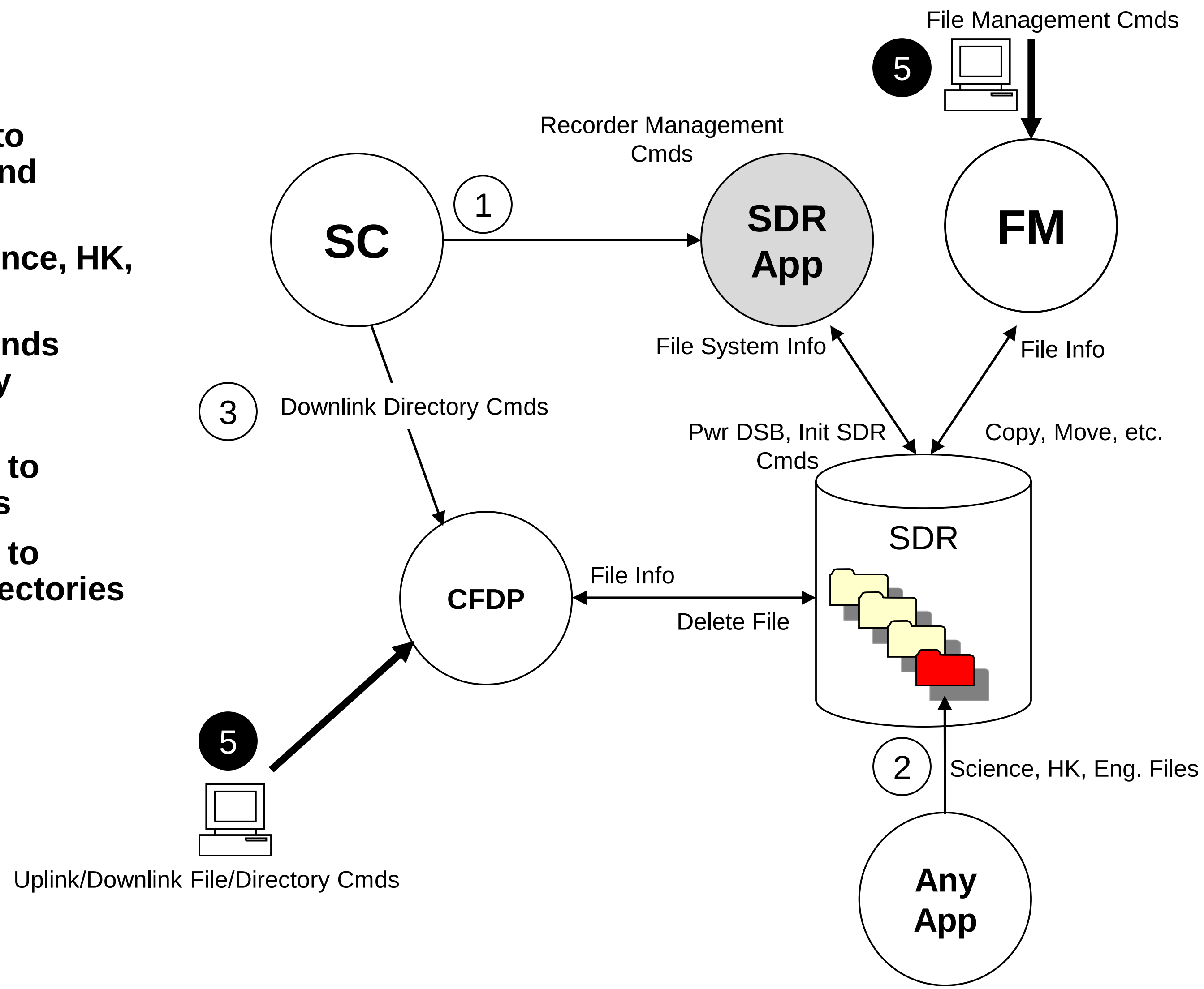
   b. **CFDP command received by Stored Command application or ground to downlink file/directory**

2) **CFDP copies file data to priority queue and begins file transfer:**
   - Opens next file from queue
   - Creates and sends meta-data PDUs

3) **CFDP sends PDUs to Telemetry Output application for downlink.**

4) **CFDP handshakes with ground/uplink to complete file transfer**

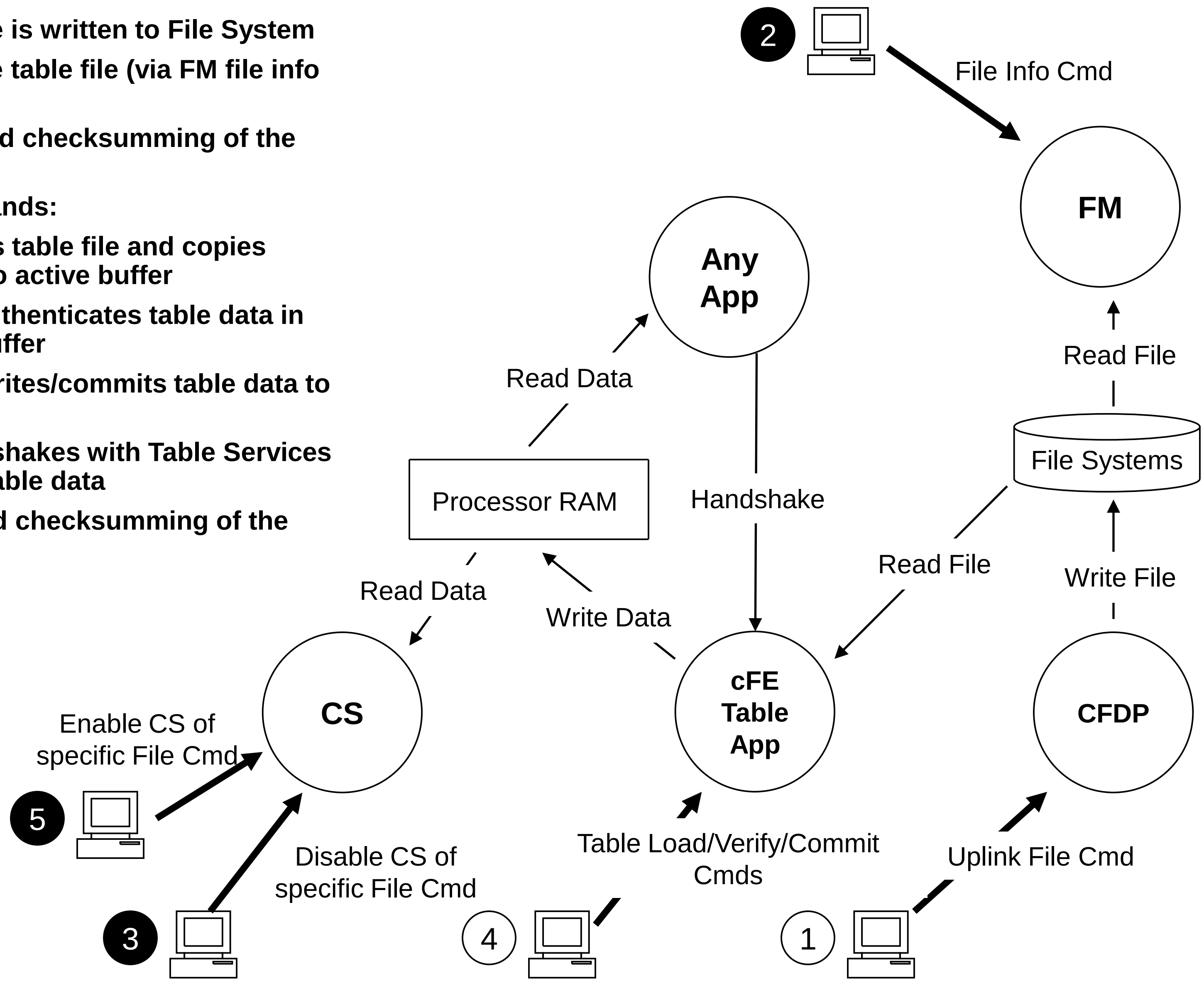
1) Stored commands sent to initialize file system(s) and create partitions
2) Applications create Science, HK, and/or Engineering files
3) SC (typically via ATS) sends CFDP downlink directory commands
4) Ground commands sent to uplink and downlink files
5) Ground commands sent to manage the files and directories in the file system(s).



- CFDP Hot Directory
- Mission Specific Application
- Optional Step

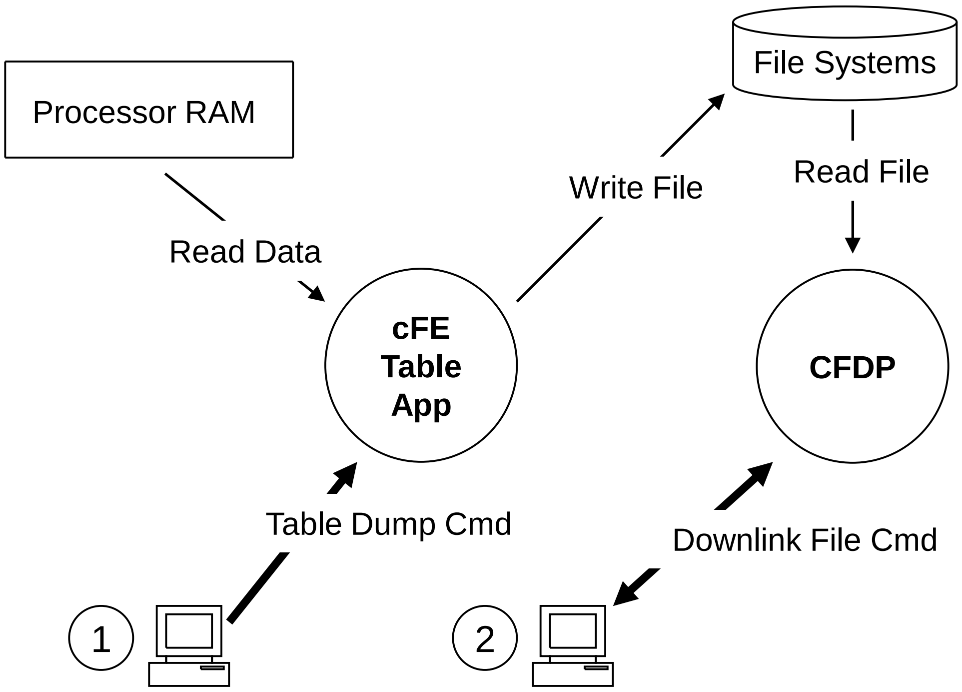
1) Uplink table – table is written to File System
2) Optionally CRC the table file (via FM file info command)
3) Disable background checksumming of the table
4) Send Table commands:
   - Load – reads table file and copies contents into active buffer
   - Validate – authenticates table data in the active buffer
   - Activate – writes/commits table data to RAM

   Application handshakes with Table Services to read updated table data
5) Enable background checksumming of the table



● - Optional Step

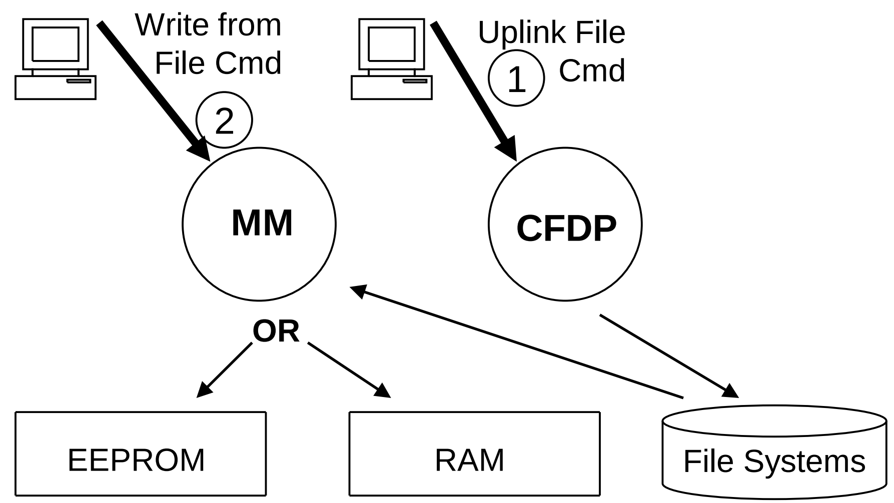
1) **Send Table dump command – table file is written to File System**
2) **Downlink file – table is written to ground File System.**



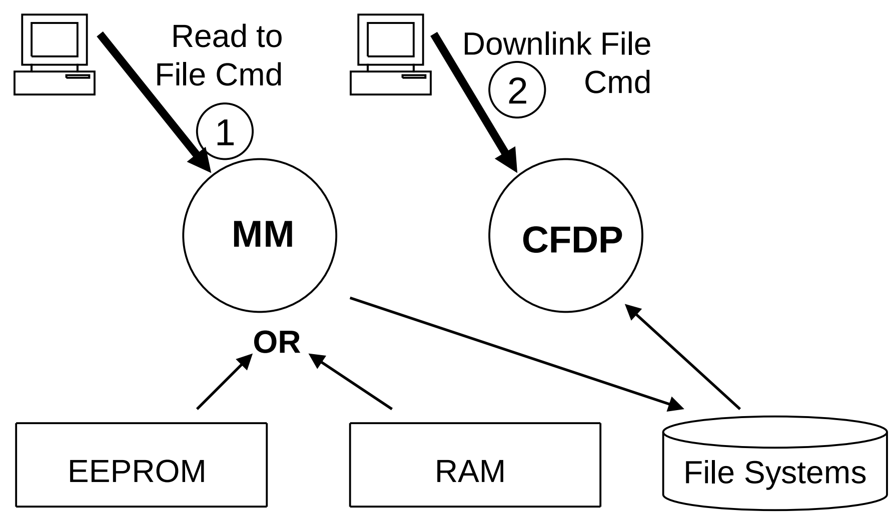
Operational Scenarios
Load/Dump Memory

- **MM Features**
  - Commanded Writes (peek and poke)
  - Commanded Reads via event messages
  - File Reads and Write (show in diagram)
- **Upload to Memory from Ground**
  1. Uplink File using CFDP
  2. Write the data from a file into EEPROM or RAM
- **Download from Memory to Ground**
  1. Read the data from EEPROM or RAM into a file
  2. Downlink File using CFDP

Upload to Memory from Ground

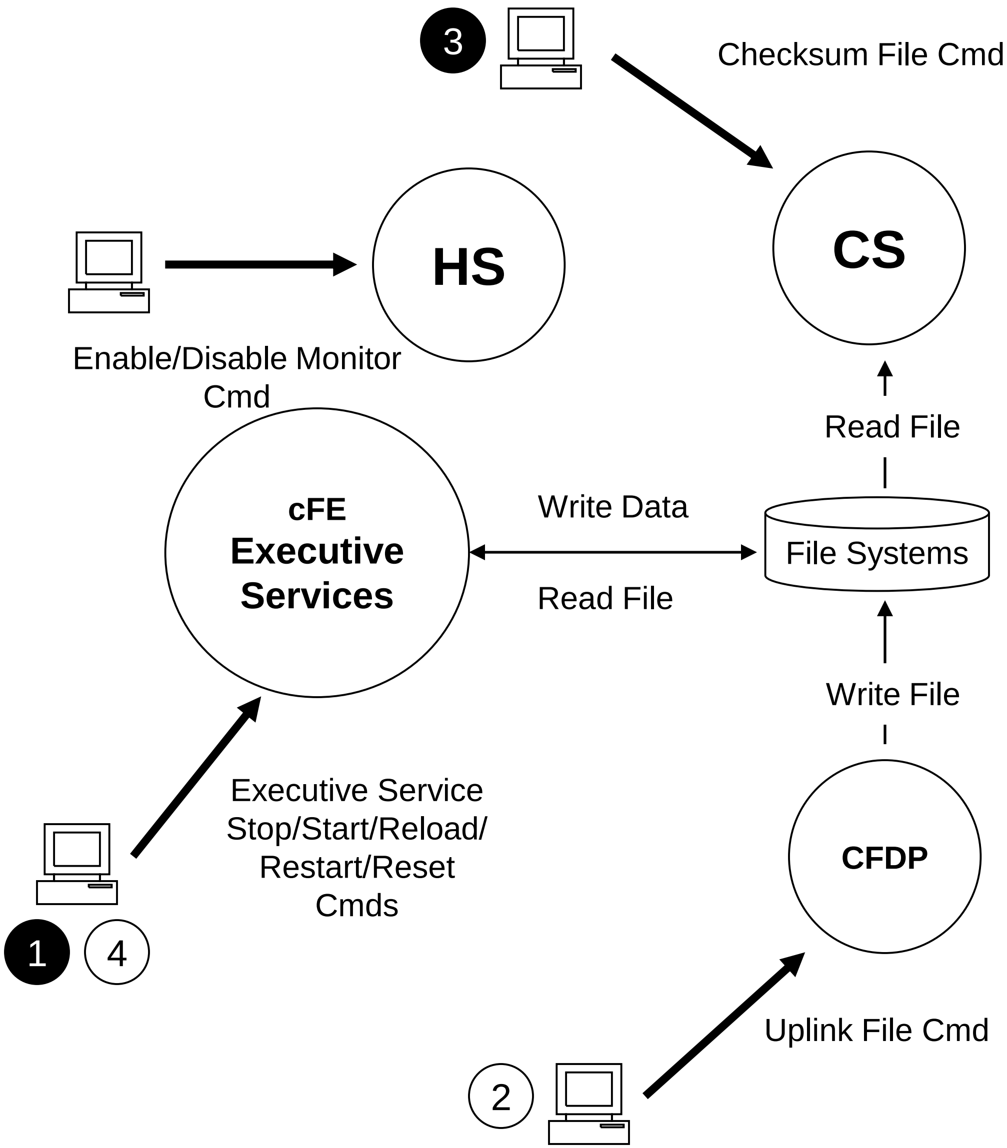


Download from Memory to Ground



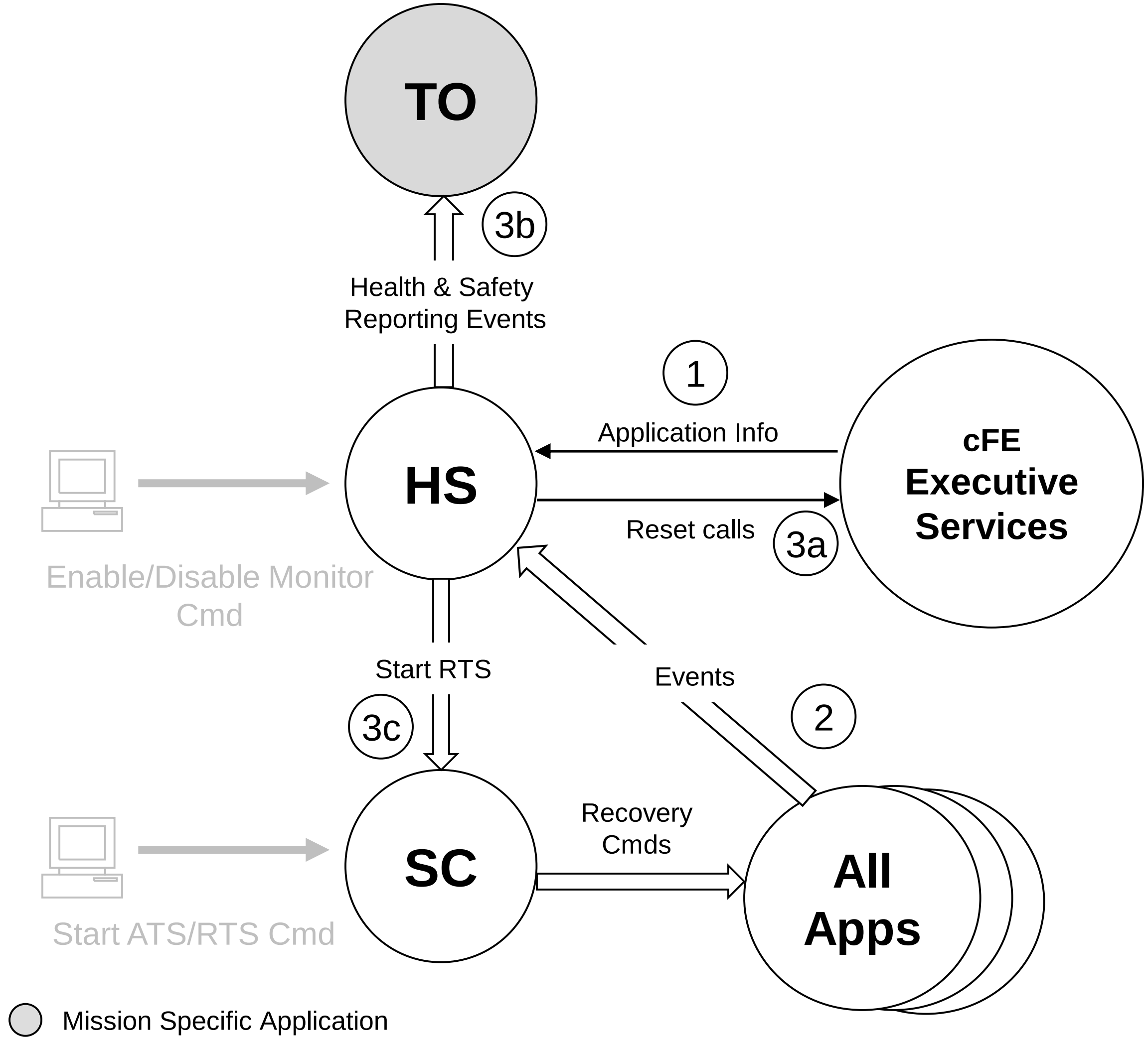
Operational Scenarios
Load and Execute Application Updates

1) **Send Executive Service command to stop application**
2) **Uplink file – file containing code update(s) is written to File System**
3) **Checksum the file**
4) **Send Executive Service commands to:**
   - Reload application
   - Start application
   - Restart application
   - Perform Processor reset



● - Optional Step

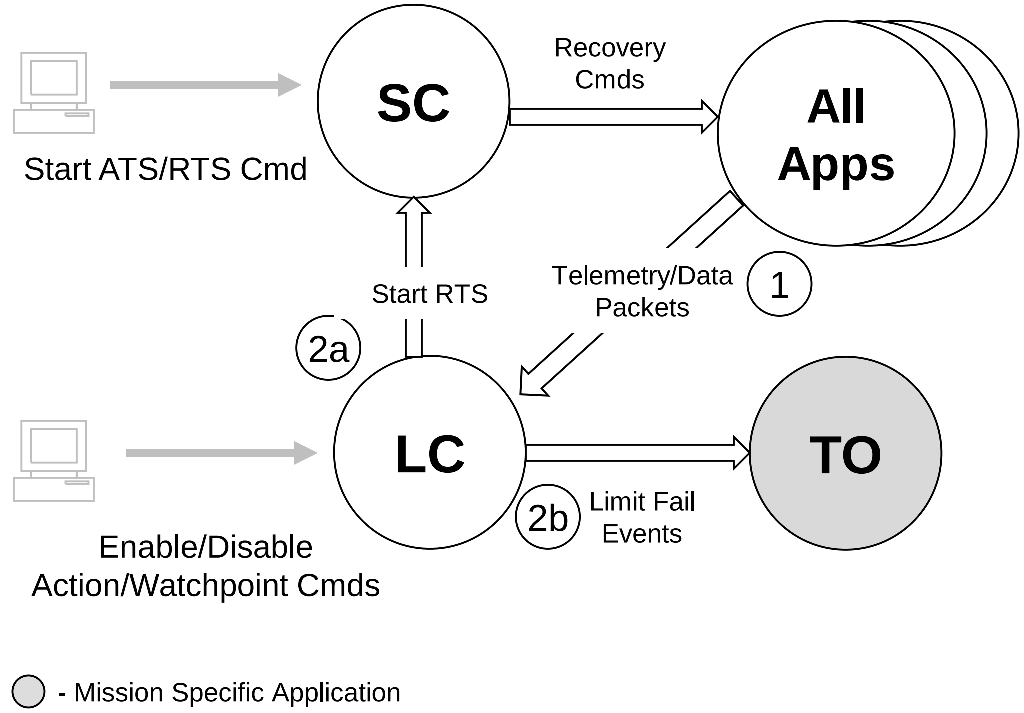
1) **HS monitors applications**
2) **HS monitors event messages**
3) **HS Table specified actions are taken in response to application and event monitoring:**
   a) **Reset applications or the processor**
   b) **Send Event message**
   c) **Initiate Stored Command (SC) recovery sequence**



Not pictured: HS manages watchdog, reports CPU utilization & detects hogging, and outputs aliveness heartbeat to UART.

Operational Scenarios
Fault Detection

1) **LC monitors table specified telemetry and data (watchpoints)**
2) **LC evaluates actionpoints and takes action upon detected failure condition:**
   a) **Initiate Stored Command (SC) recovery sequence**
   b) **Send failure event messages**



Tools

Unit Test Framework (UTF)

- **Tool is used to unit test (full path coverage) cFS Applications**
  - All cFE APIs and OSAL APIs are simulated
  - Allows for return codes to be forced in order to exercise the error path in the code undergoing unit testing
  - Delivered with each cFE release

UT-Assert Library

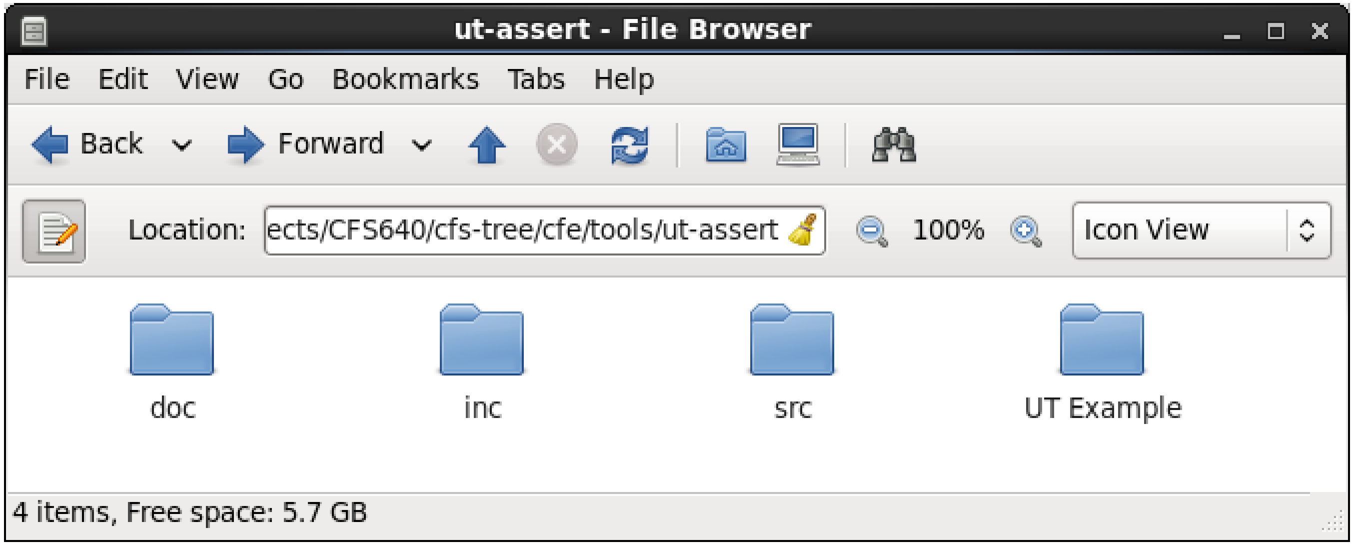
- **Used for unit testing cFS applications and tasks through the use of assert statements**
  - An assert statement evaluates whether a condition is true or false and returns PASS or FAIL.
  - Each test case should be self-verifying, rather than needing to be manually verified after running

- **Used to test functionality and code coverage of every function in an application, one at a time**
  - Each test should be completely independent from other tests

- **Each test case should test ONLY its designated function/operation**
  - Results of sub-functions do not need to be tested – tested in separate test case for each sub-function

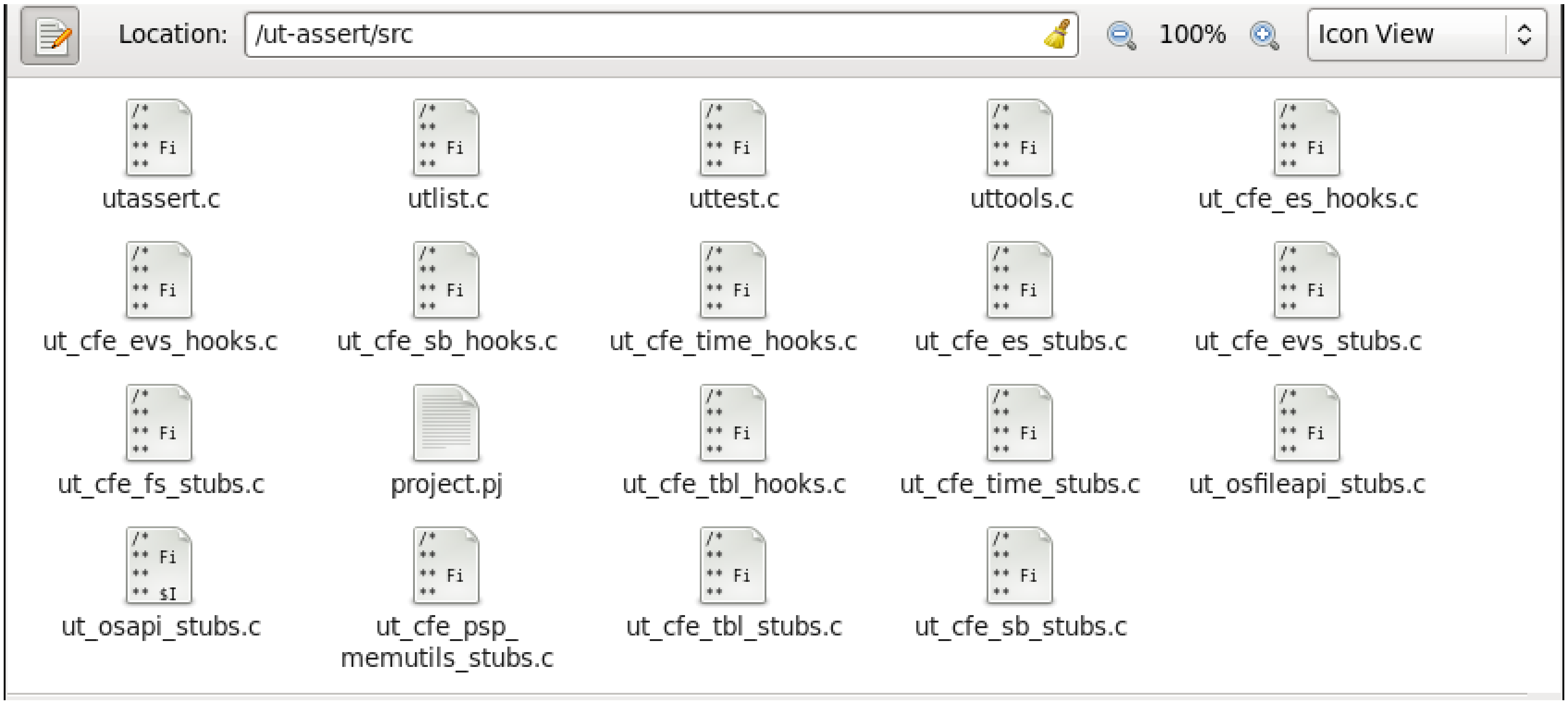
- **All cFS API library functions (OSAL, PSP, and cFE) are automatically substituted with UT-Assert stub and hook functions**
  - Every cFS API library function has a corresponding UT-Assert stub function
  - Tests can set individual stub functions to behave in 3 different ways:
    1.) Return its default return value (usually CFE_SUCCESS)
    2.) Return a specified custom value
    3.) Execute a specified custom hook function and then return the resulting return value
  - Some cFS API library functions have corresponding hook functions that are called by default
    - Can be substituted for a custom hook function or a custom return value

What's Included in the Library

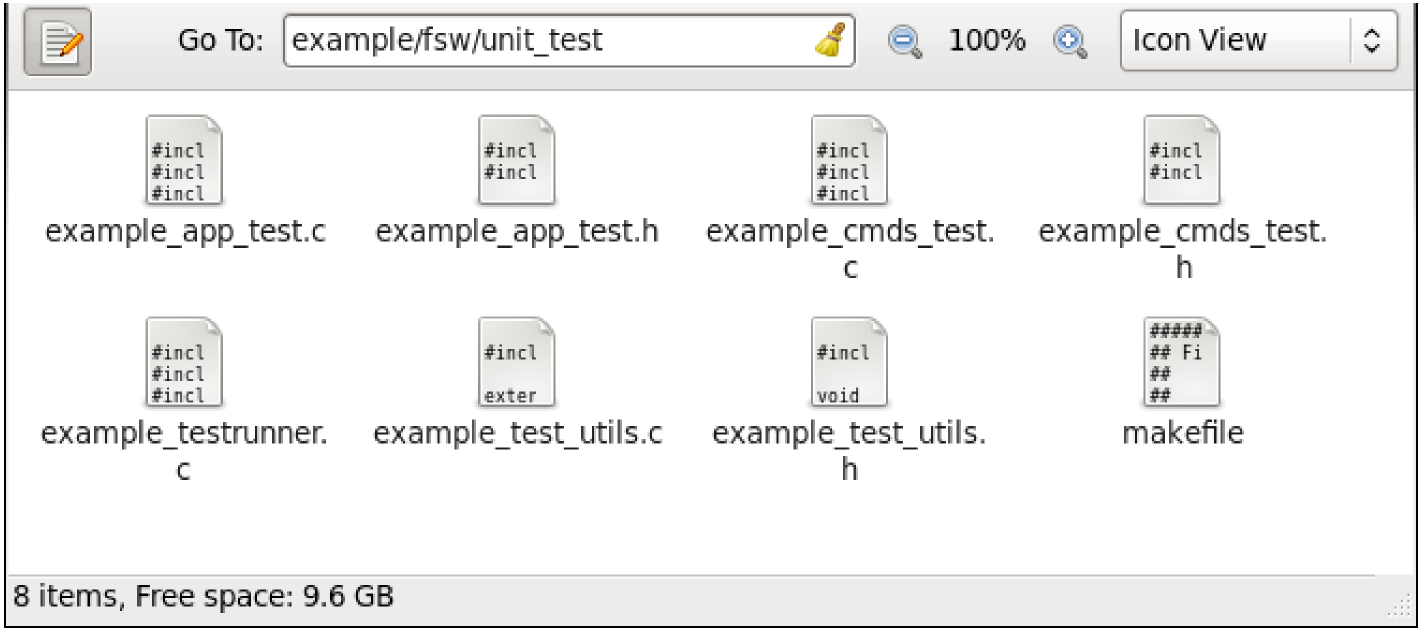


- **/src: Contains the library source files**
- **/inc: Contains the header files for the library source files**
- **/doc: Contains documentation about UT-Assert**
  - Note: current documentation is incomplete and outdated
    - Will be replaced by this presentation
- **/UT Example: Contains an example UT-Assert unit test suite (and the app it tests)**

What's Included in the Library

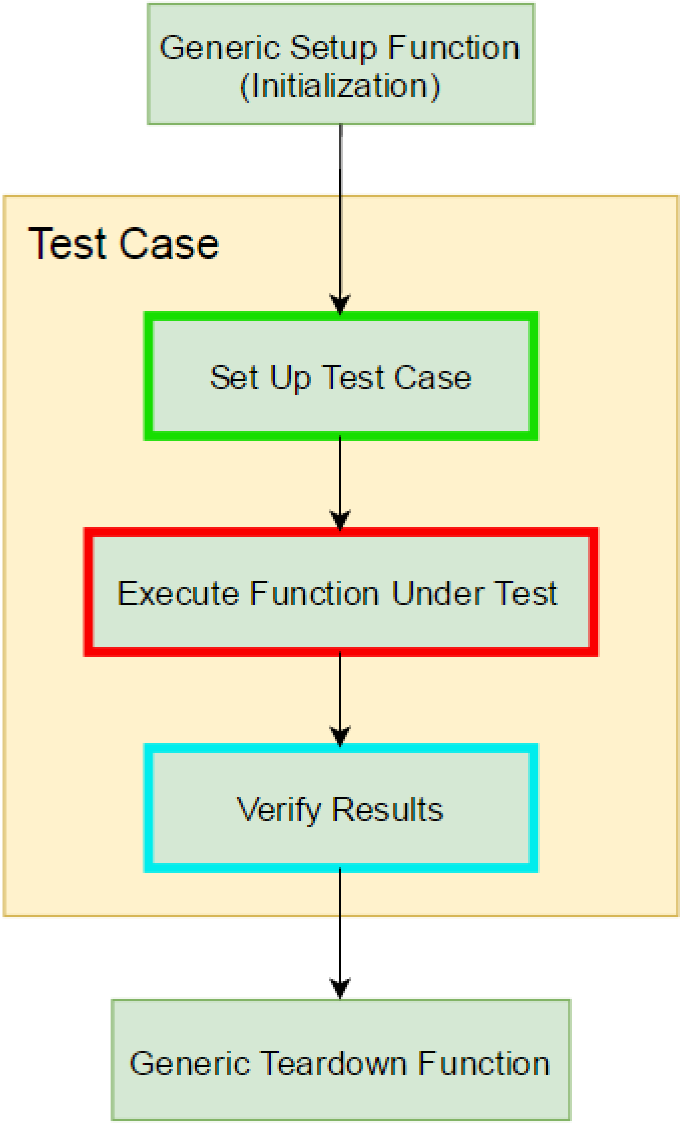


- **Contents:**
  - utassert.c /.h – defines the standard assert function, along with a few related functions
  - utlist.c – defines functions to create linked lists, which are used elsewhere in the library
  - uttest.c – defines the functions used to add and run test cases
  - uttools.c – defines miscellaneous functions that are useful for unit testing
  - ut_cfe_***_stubs.c – defines the stub functions for a particular cFS component, and supporting functions
  - ut_cfe_***_hooks.c – defines the default hook functions for a particular cFS component



- **For the example application:**
  - <source file name>_test.c / .h: defines the unit test cases for all functions in a particular source file
  - <app name>_test_utils.c / .h: defines miscellaneous test functions (Setup, Teardown, etc)
  - <app name>_testrunner.c: defines the main function, which adds all test cases and runs them
  - makefile: standard makefile functionality

Test Structure: Test Case Flowchart



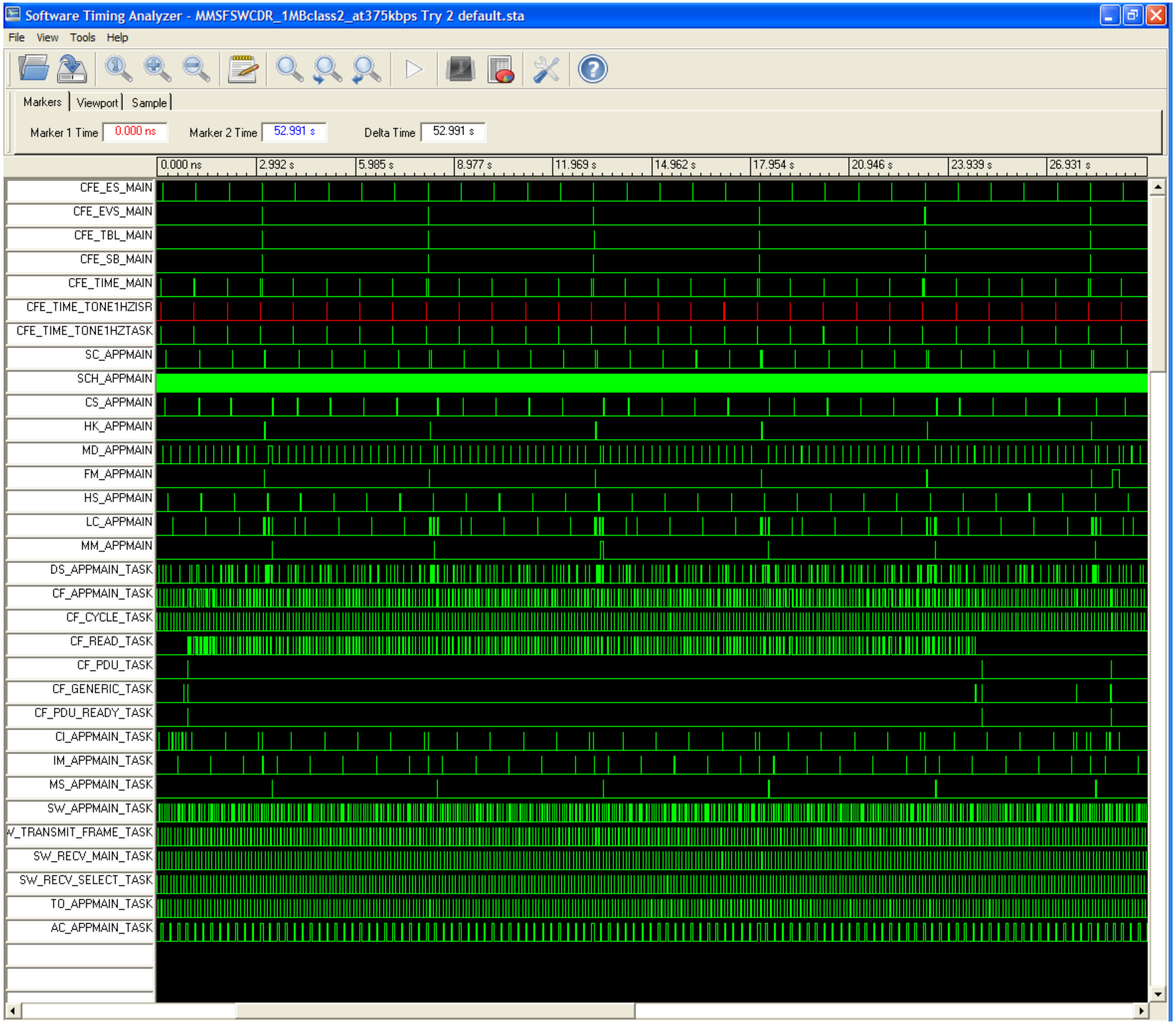
How to Get the UT-Assert Library

- **cFS NASA wide community Babelfish git repository**
- **Sourceforge**
  - https://sourceforge.net/projects/cfs-ut-assert/

Software Timing Analyzer - 1

- **Microsoft Windows program that provides visibility into the real-time performance of embedded systems software**
- **The software has the following key features:**
  - Graphically displays task execution as waveforms
    - Rising edge indicates that a task is running, Falling edge indicates that a task is pending
    - Can display the execution state of multiple tasks simultaneously
  - Calculates Statistics
    - Measures task execution interval (how often a task runs) and execution width (how long it takes to run)
    - Measures Min, Max, and Average CPU Utilization.
  - Analyzes/searches timing data for user specified conditions

Software Timing Analyzer - 2



Additional Tools - 1

- **Command Ingest (CI) Lab Tool**
  - Application that accepts CCSDS telecommand packets over a UDP/IP port
- **Telemetry Output (TO) Lab Tool**
  - Application that sends CCSDS telemetry packets over a UDP/IP port
- **Ground System GUI**
  - Python / QT4 based Command/Telemetry GUI
  - Designed for use with CI Lab and TO Lab tools
  - Uses C program to send commands over a UDP socket

Additional Tools - 2

- **Scheduler Lab Tool**
  – Application that schedules activities with a one second resolution
- **Generate Scheduler Table Tool**
  – Python script used to generate the scheduler definition table (sch_def_schtbl.c) used by the cFS Scheduler (SCH) application
- **Generate Application Template Tool**
  – Python script used to generate the base code, including the table definitions, for the new applications listed in the command
- **Table CRC Tool**
  – C program designed to calculate the CRC of a given table file (.tbl) using the same algorithm as the cFE Table Services flight software
- **Elf to cFE Table Tool**
- **Message ID Print Tool**
  – Prints the Message IDs used by the cFE

cFS Documentation

- **cFE Documentation in the “docs” directory**
  - cFE Requirements Document
  - cFE Application Developer’s Guide
  - cFE Deployment Guide
  - cFE User’s Guide
  - OSAL Library API Document
  - Tools documentation in the “tools” directory
    - Performance Analysis
    - Cmd/Tlm utils
    - Elf2cfetab
    - UTF
- **Build Verification Testing results in the “test-and-ground/test-review-packages” directory**
- **Each cFS application has ability to generate a Doxygen users guide (html format)**
- **Mission “docs” directory**
  - cFS Deployment Guide
  - cFS Tlm and Cmd Mnemonic Naming Convention

Deployment

Where is the cFS?

- **cFE open Internet access at http://sourceforge.net/projects/coreflightexec/**
  - Source code
  - Requirements and user guides
  - Tools

- **OSAL open Internet access at http://sourceforge.net/projects/osal/**
  - Source code
  - Requirements and user guides
  - Tools

- **cFS application suite is also available on sourceforge**
  - Links are available from https://cfs.gsfc.nasa.gov

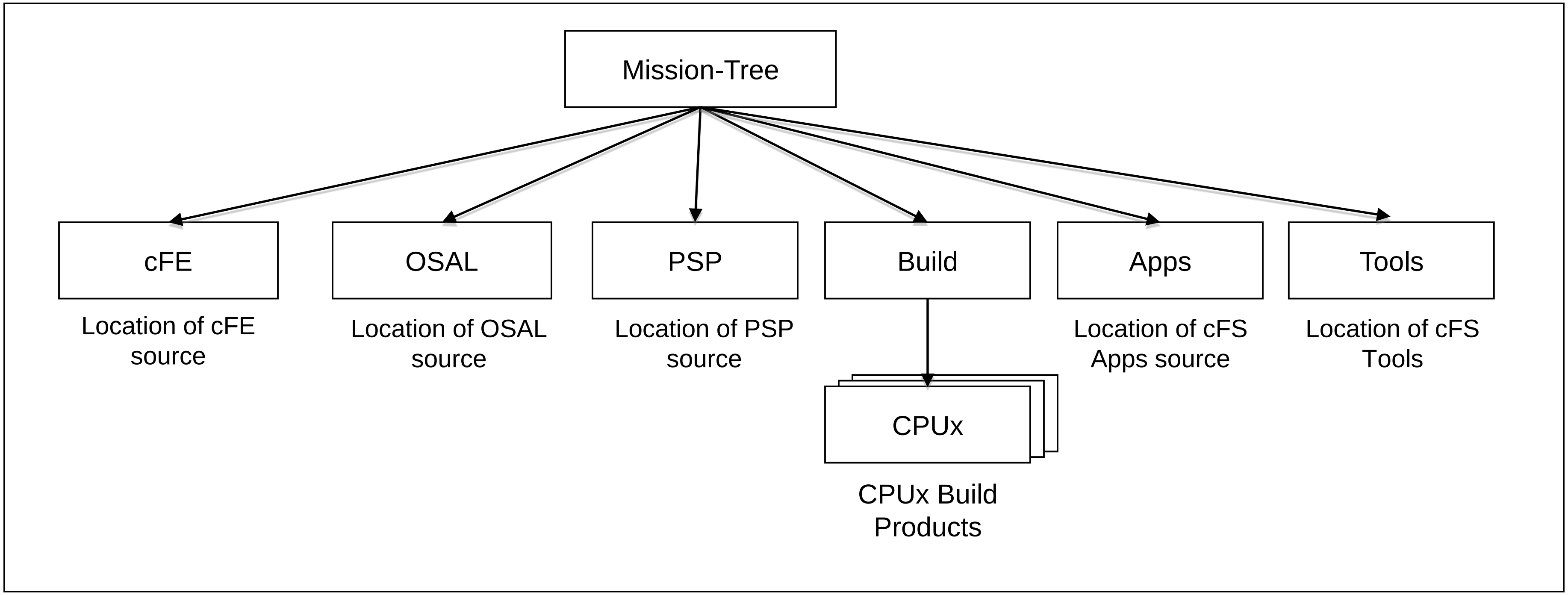
- **cFS Public Website at http://www.coreflightsystem.org**

cFS Community Projects on Babelfish

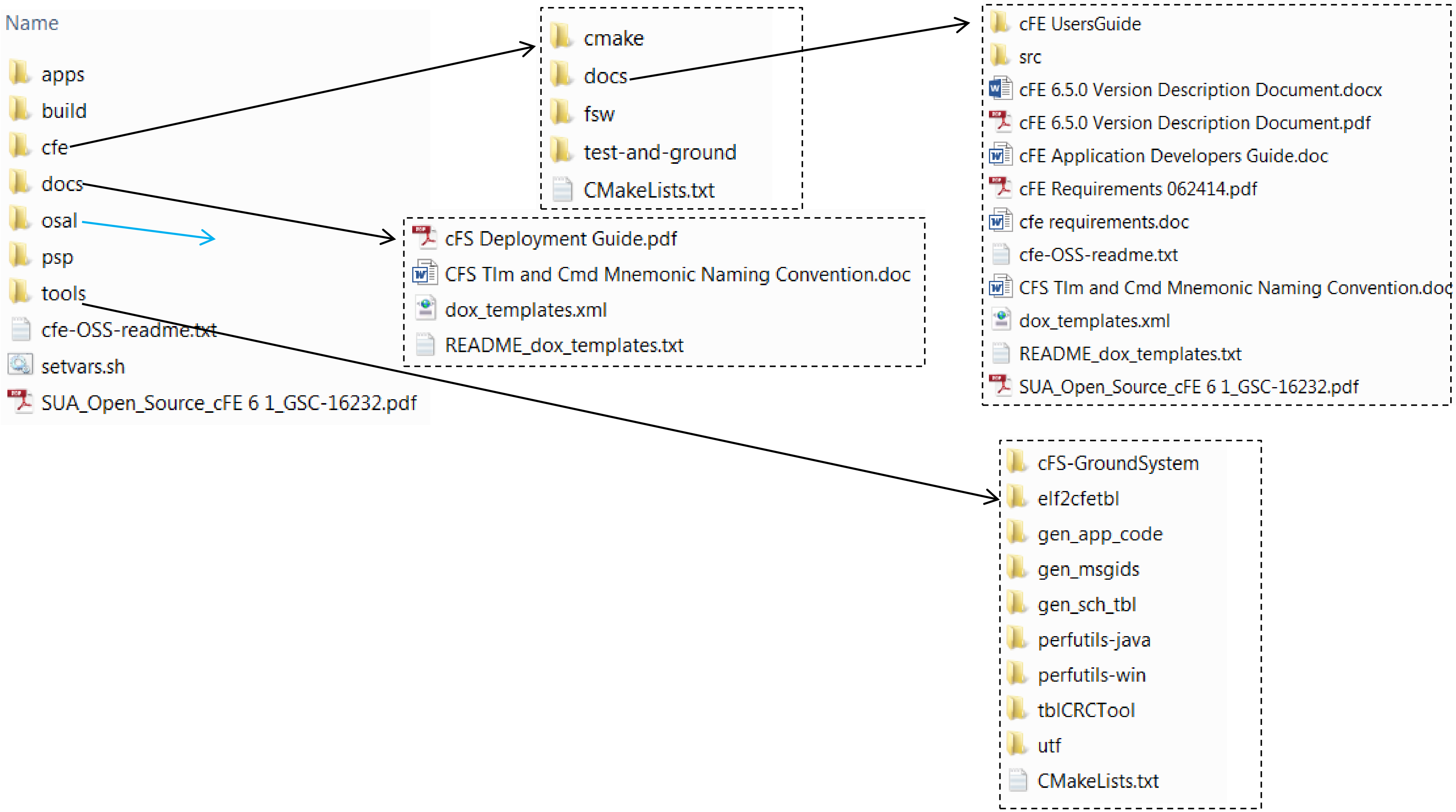
- **Babelfish provides two services for each project:**
  - Git repository
  - Trac system
    - Provides issue tracking and Wiki services
- **Babelfish hosts six separate cFS projects/repos:**
  - cfs_cfe
  - cfs_osal
  - cfs_psp
  - cfs_tools
  - cfs_apps
  - cfs_test
- **Anyone with an NDC account can acquire access**
  - Contact Greg Limes (gregory.limes@nasa.gov) for an account

Development Environment

- **The cFS has a complete development environment that is designed to manage:**
  - Builds of images for multiple processors
  - Multiple processor architectures
  - Multiple operating systems
  - Different application loads on each processor
  - As little duplication of code as possible

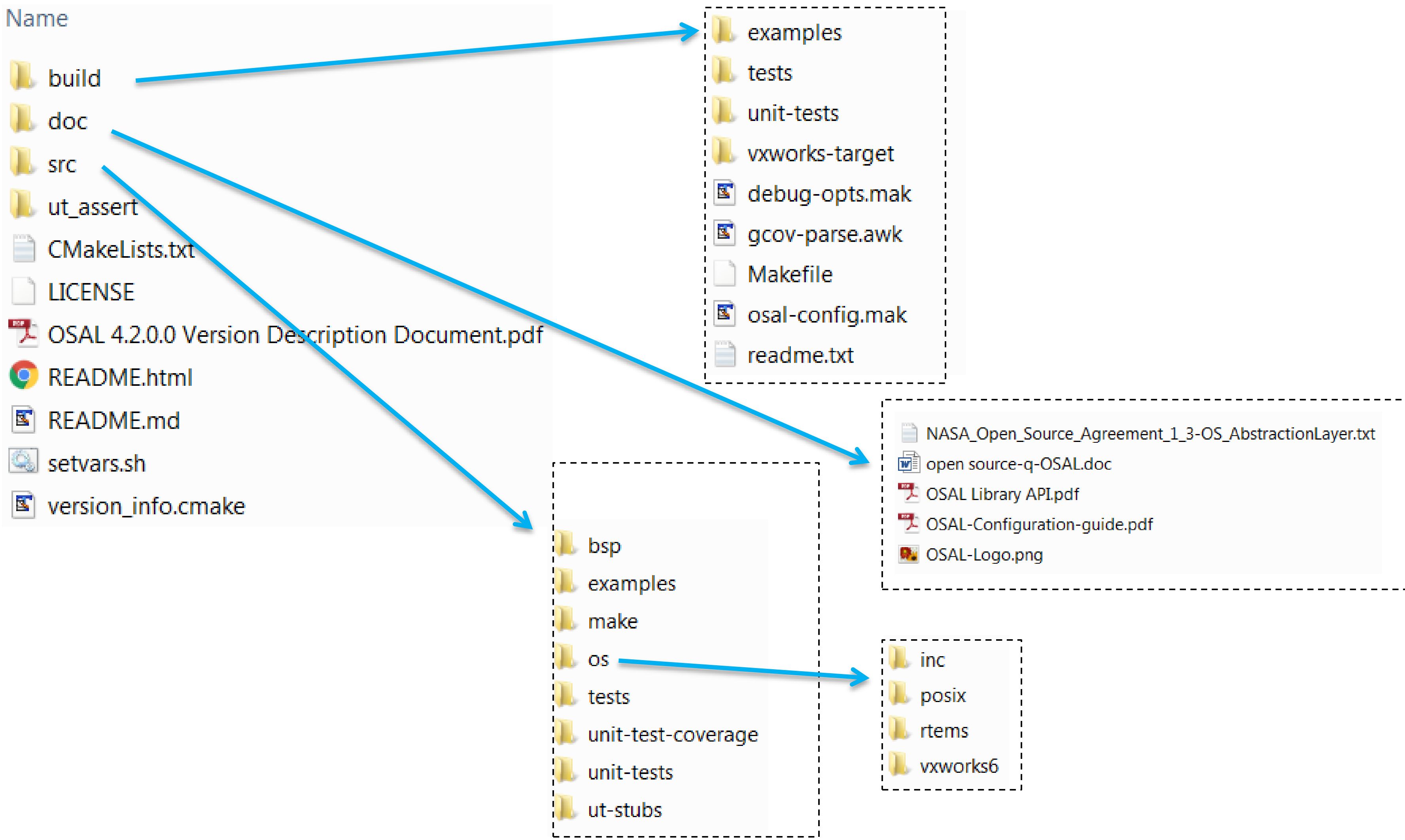


What's in the cFE Open Source Tarball



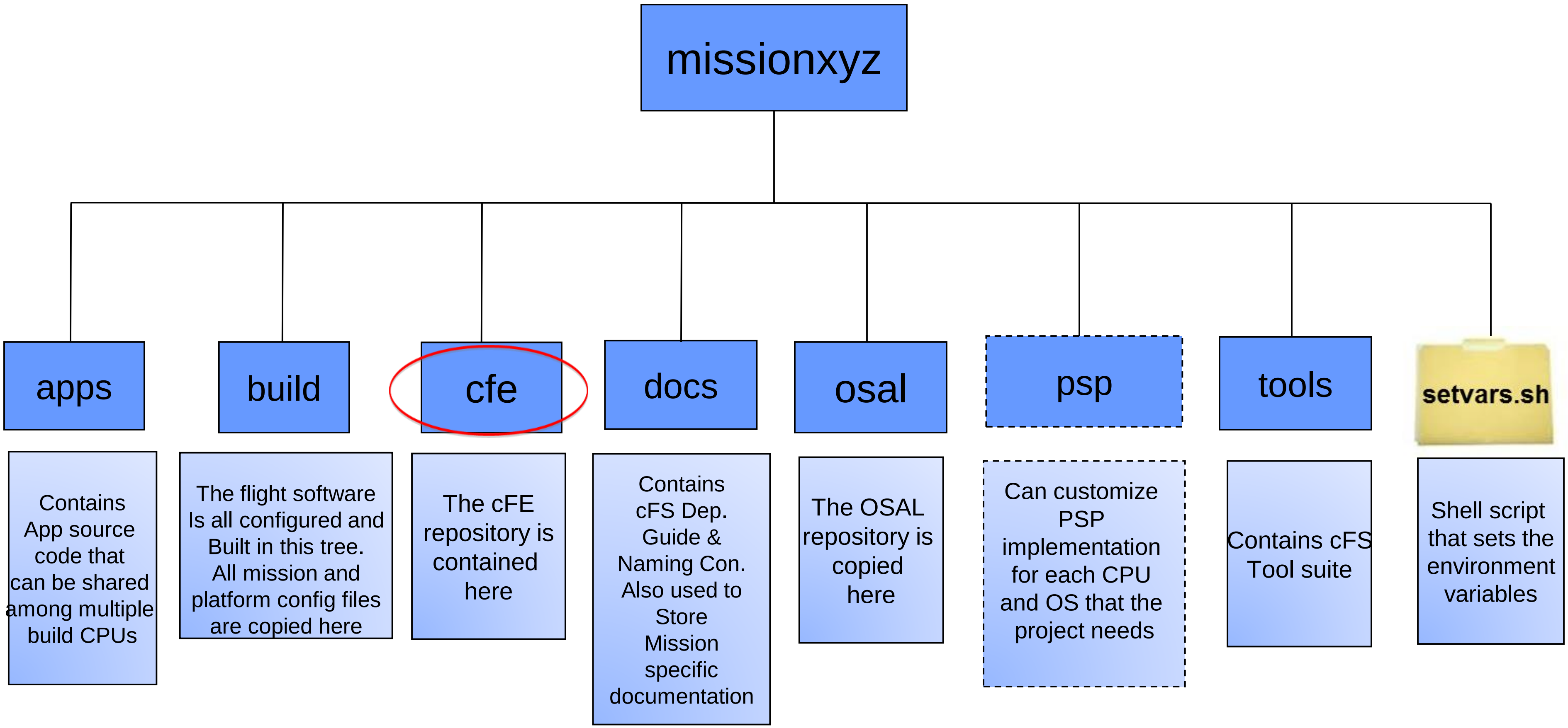
Note: There are other PSPs at each center that are not open source

What's in the OSAL Open Source Tarball

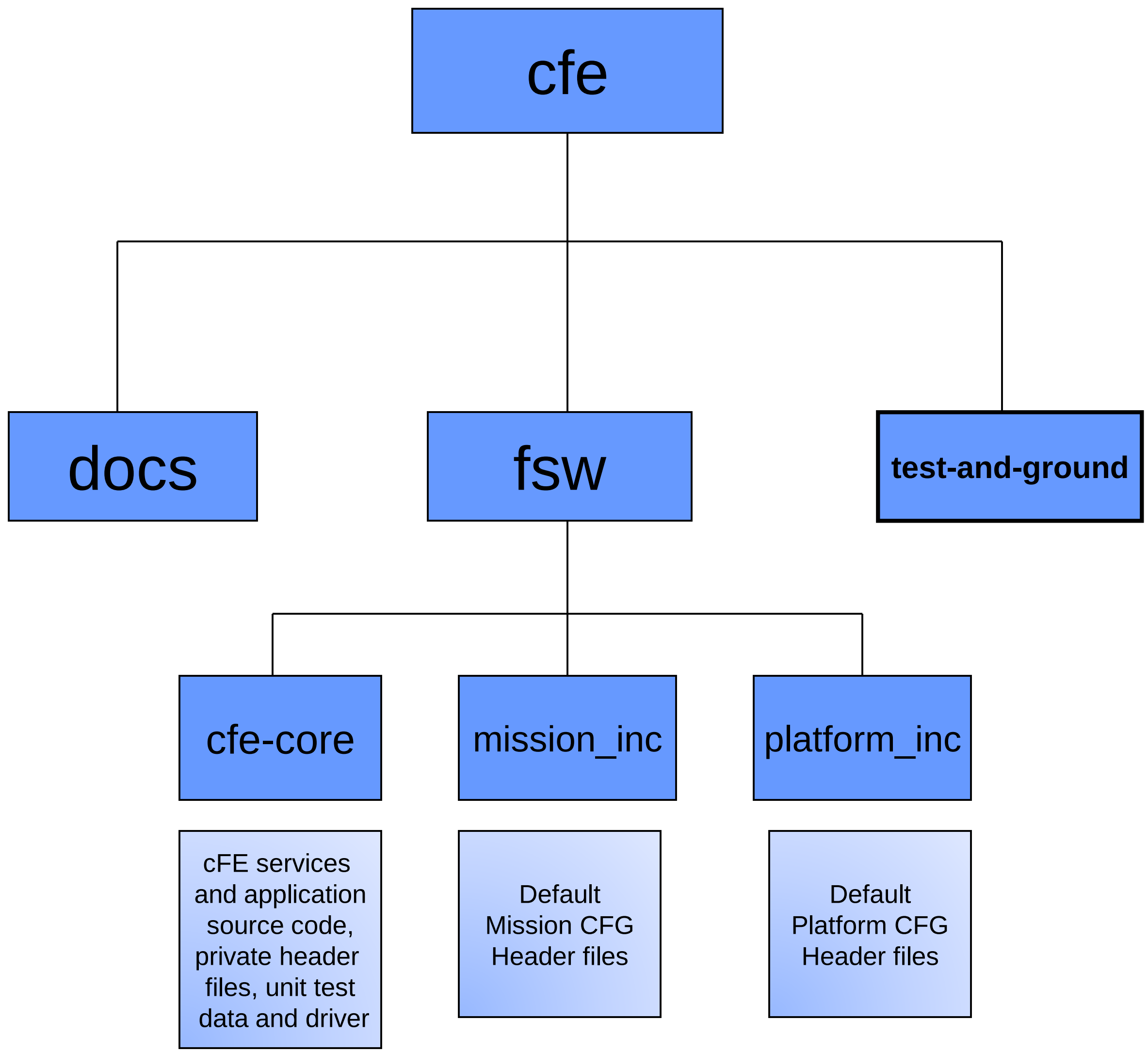


Note: There are other OS implementations at each center that are not open source

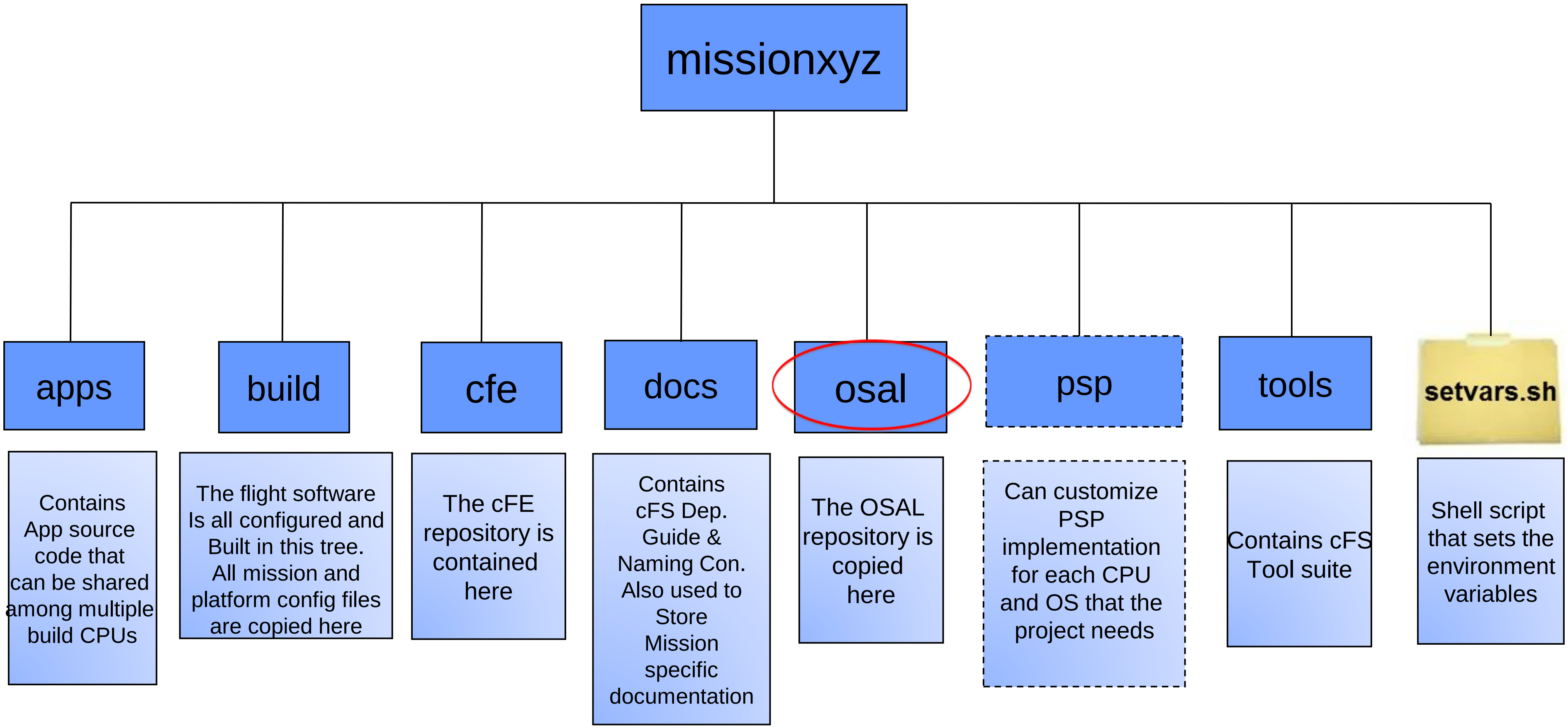
cFS Mission Directory Structure



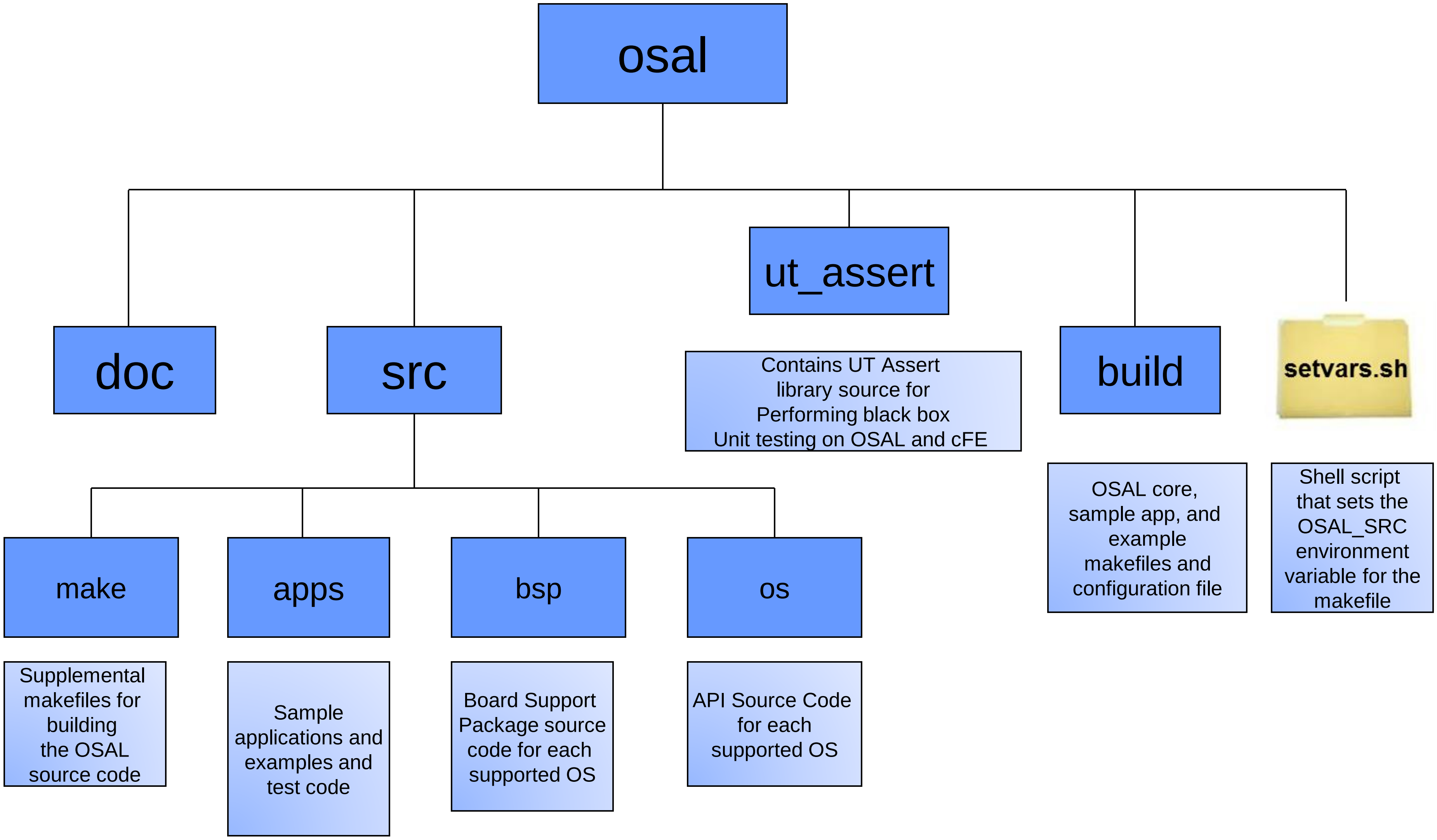
cFE Directory Structure



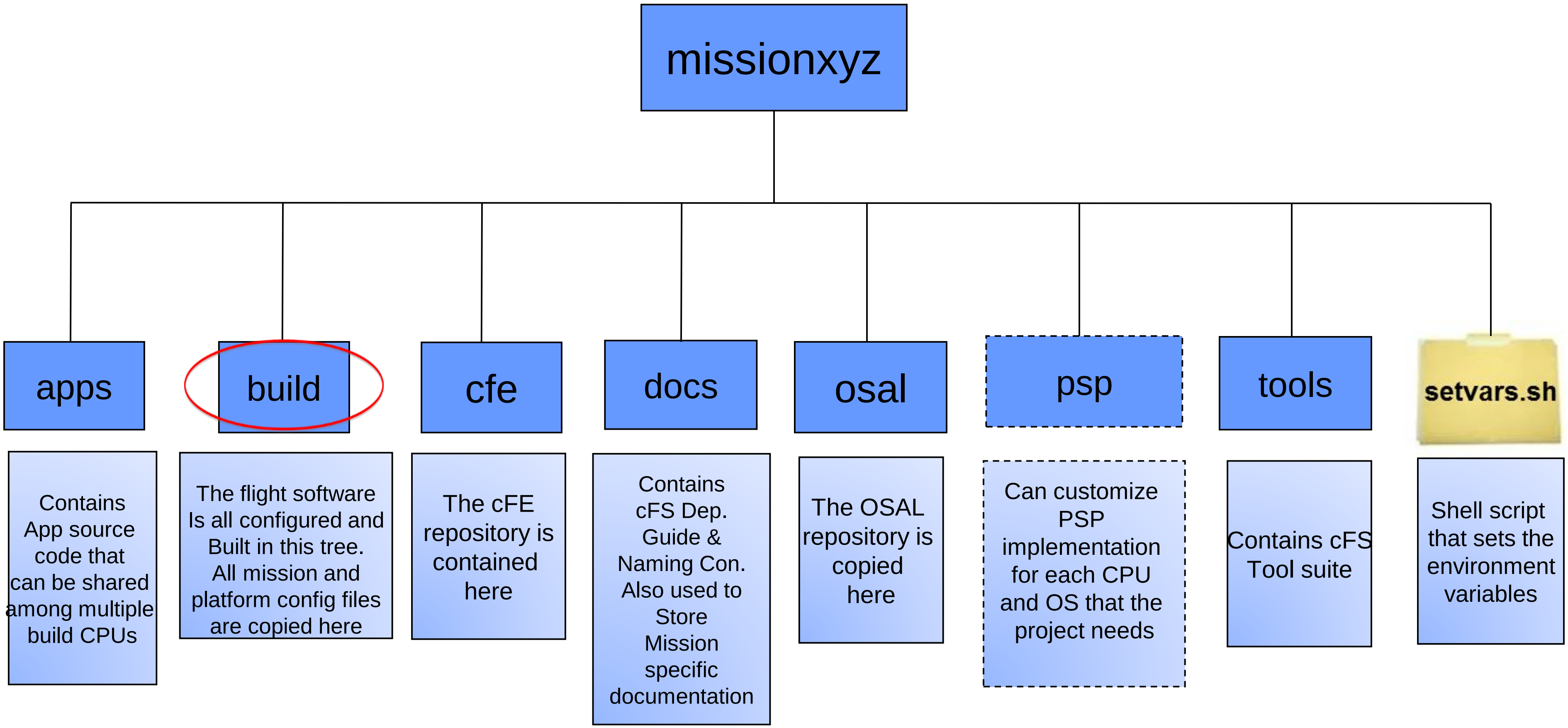
cFS Mission Directory Structure



OSAL Directory Structure



cFS Mission Directory Structure

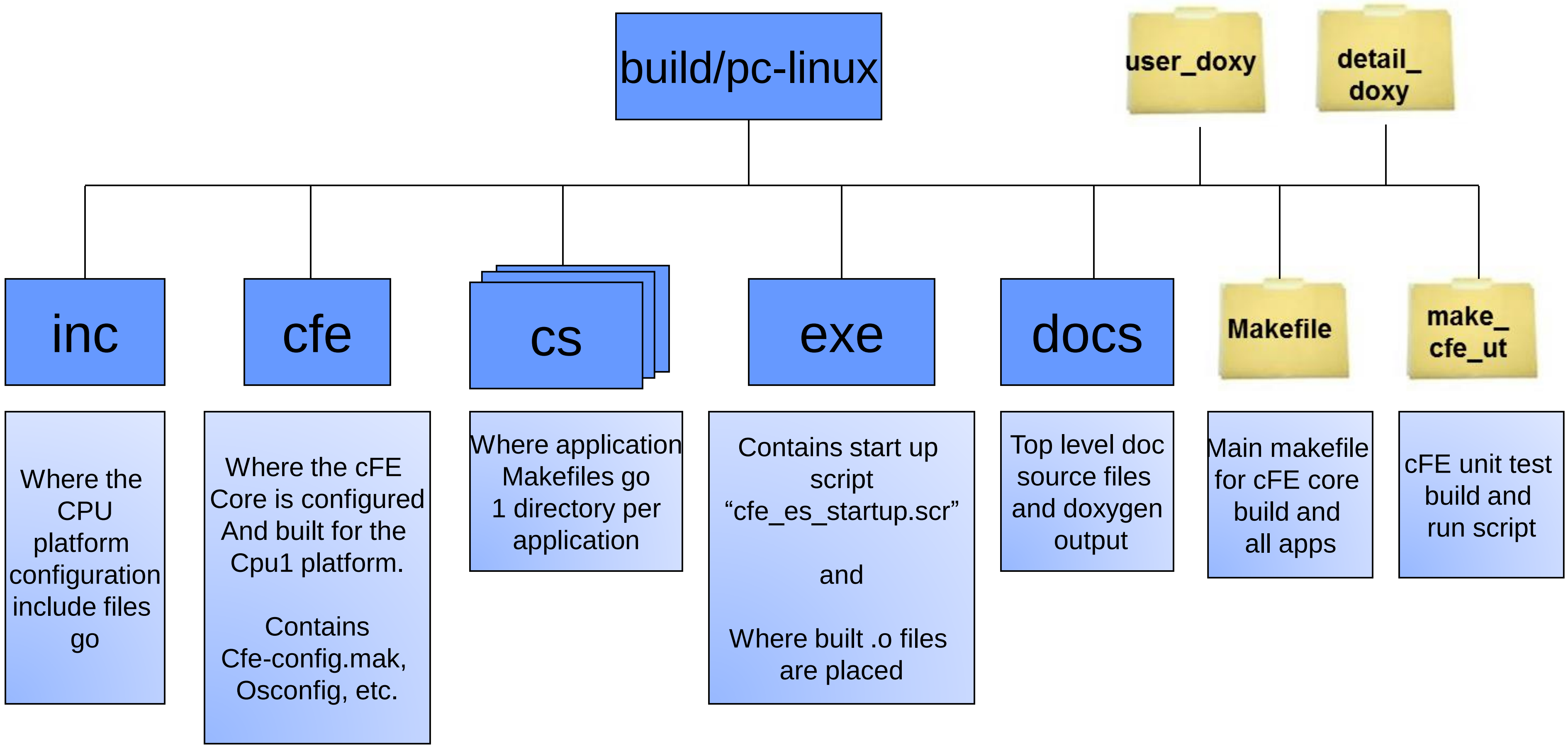


cFS Mission “build” Directory Structure

- build
  - mission_inc: Where the mission-wide configuration Include files go
  - pc-linux: Build directory and configuration for CPU 1
  - mcp750-vxworks6: Build directory and configuration for CPU 2
  - …
  - cpuN: Build directory and configuration for CPU N
  - Makefile: Main makefile for all Mission builds
  - cfs.mak: Supplemental makefile for applications for all Mission builds
  - cfs_tst.mak: Supplemental makefile for unit tests for all Mission builds

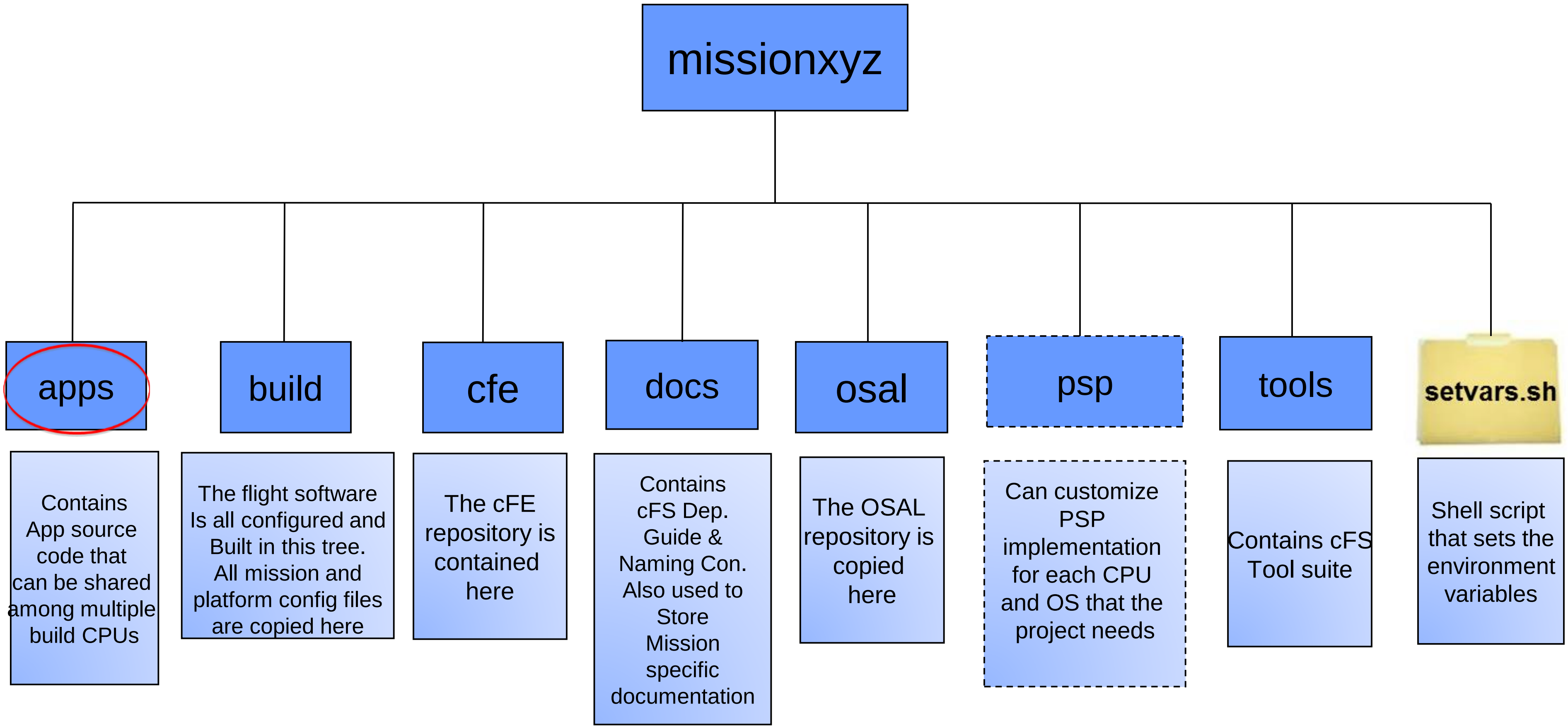
The “build” directory is where the cFS (cFE Core + cFS Apps) is built for a mission. This directory contains all configured mission and platform configuration files.

cFS “build/pc-linux” Directory Structure

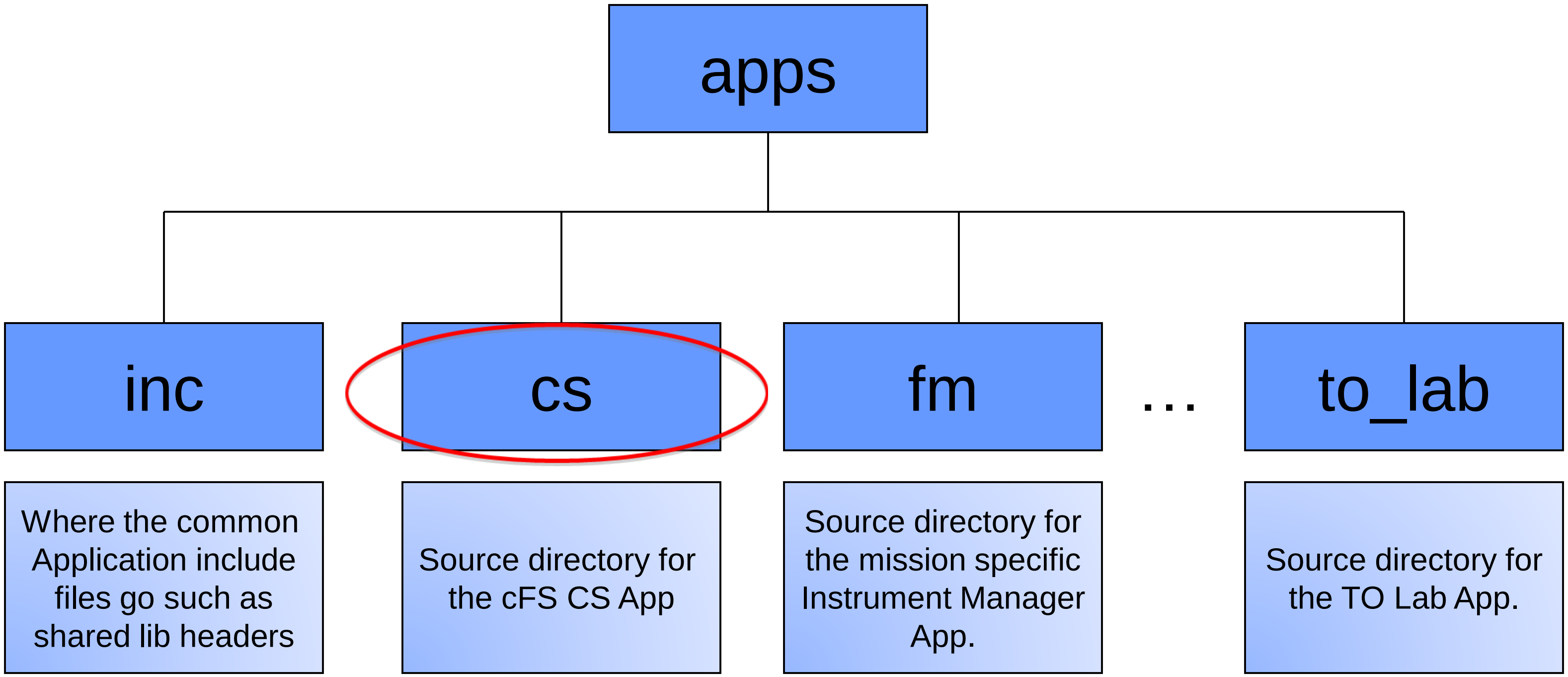


Each Platform (CPU) directory can have a custom cFE core configuration which is built for a specific architecture, platform, and operating system. It can have a unique mix of cFS applications.

cFS Mission Directory Structure

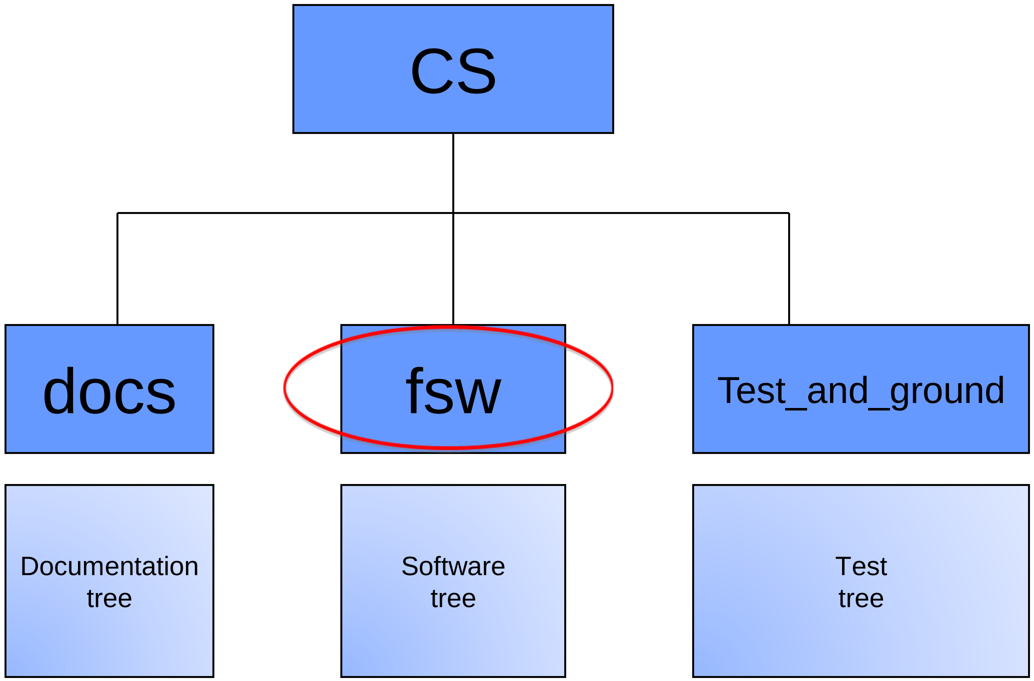


cFS “apps” Directory Structure



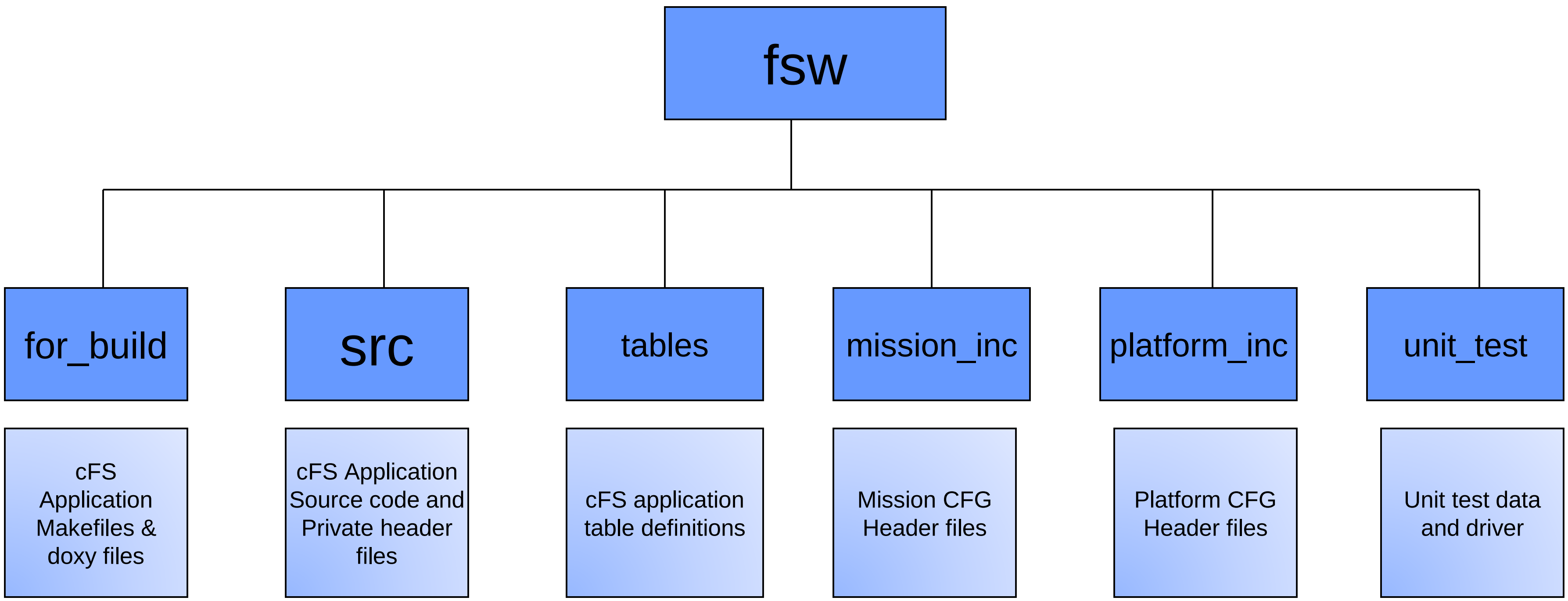
The “apps” directory is where all of the cFS applications and mission unique applications are stored. There are no build products stored here.

cFS Application Directory Structure



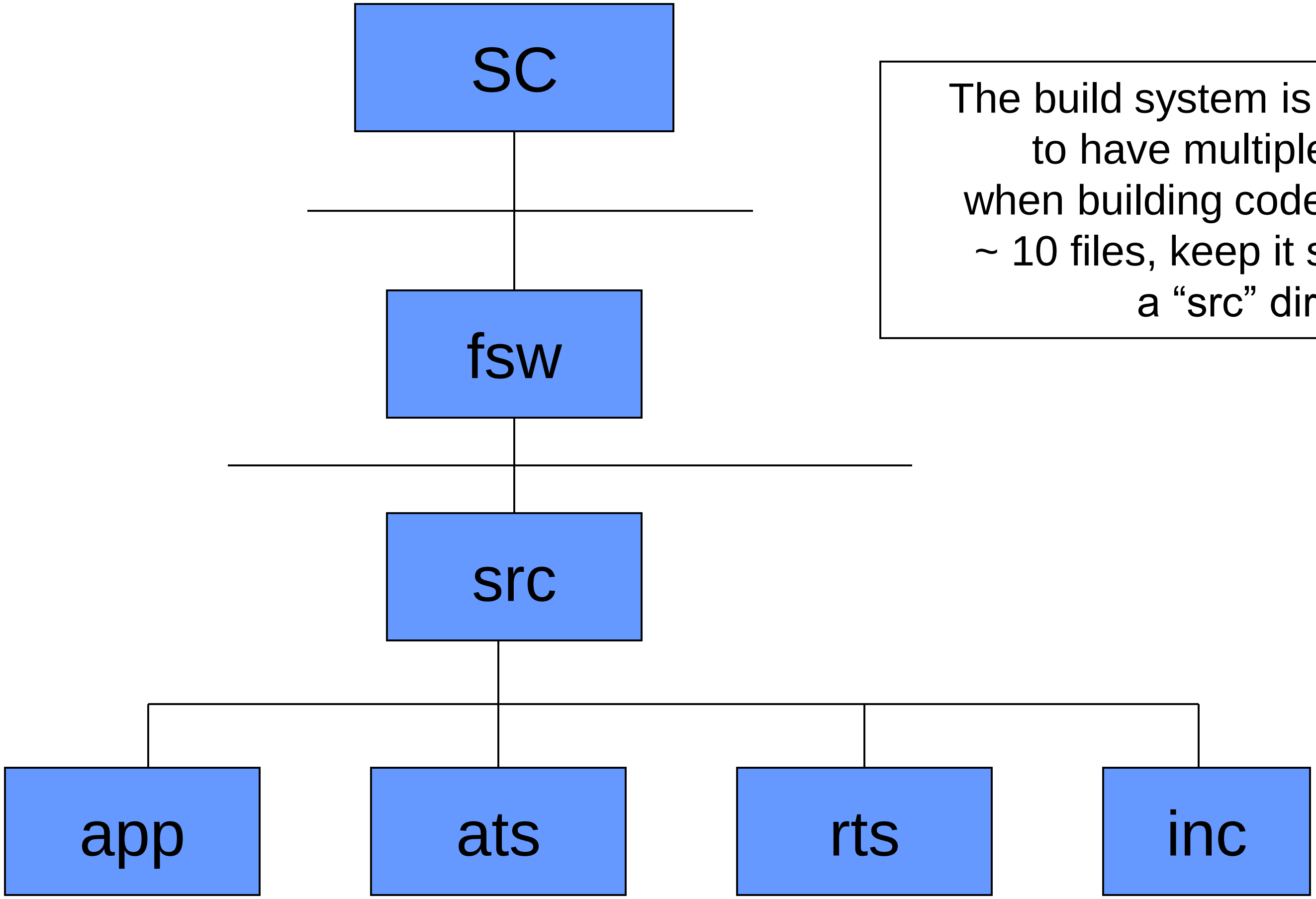
The cFS App directory is where a single cFS Application is stored. It includes all software products, documentation, tests (unit tests and test procedures) and miscellaneous utilities.

cFS Application “fsw” Directory Structure



The “fsw” directory is where all of the software for the cFS Application is stored. The “src” directory includes all private header files and C source files. The remaining directories have public and configuration header files to be installed in the cFS Mission Directory Structure.

cFS Application “fsw/src” Directory Structure



The build system is flexible enough to have multiple directories when building code. For apps with ~ 10 files, keep it simple with just a “src” directory

Installation and Configuration

- cFE open source release
- OSAL open source release
- cFS Application releases (optional)
- Build machine and Target platform

Installation and Configuration Steps - 1

- **Setup The Mission Directory Structure**
  - Unpack cFE 6.4.2 open source release package
    - Packaged within the cFS Mission Directory Structure Template
  - Unpack OSAL open source release package and locate it in the “misson-xyz/osal” directory
  - Unpack mission apps in the “mission-xyz/apps” directory

Installation and Configuration Steps - 2

- **Install Configuration files and makefiles**
  - In the missionxyz/build/cpuX directory run “make config” to:
    - Install cFS application mission configuration and platform configuration header files in the appropriate mission tree directory locations
      - Xx_mission_cfg.h files go in “mission-xyz/build/inc”
      - Xx_platform_cfg.h files go in “mission-xyz/build/cpuX/inc”
      - This is not an all inclusive list
    - Install cFE Core mission configuration and platform configuration headers in the appropriate mission tree directory locations
      - cfe_mission_cfg.h goes in “mission-xyz/build/inc”
      - cfe_platform_cfg.h goes in “mission-xyz/build/cpuX/inc”
      - cfe_msgids.h goes in “mission-xyz/build/cpuX/inc”
    - OSAL osconfig.h platform config header in the appropriate places.
      - osconfig.h goes in “mission-xyz/build/cpuX/inc”
    - Install application makefiles in the appropriate mission tree directory locations

Installation and Configuration Steps - 3

- **Edit Configuration Files**
  - The following configuration files need to be tailored for the mission:
    - cfe_mission_cfg.h -- cFE mission configuration header file
    - Any application mission configuration header files (xx_mission_cfg.h)
  - For each platform (CPU) config:
    - cfe_platform_cfg.h -- cFE platform configuration header file
    - osconfig.h -- OSAL platform configuration header file
    - Any application platform configuration header files (xx_platform_cfg.h)
  - Makefiles
    - All top level makefiles have to be edited to reflect the applications that are being used
  - Other files:
    - Startup scripts, edit Applications for subscription and table info, etc.

Installation and Configuration Steps - 4

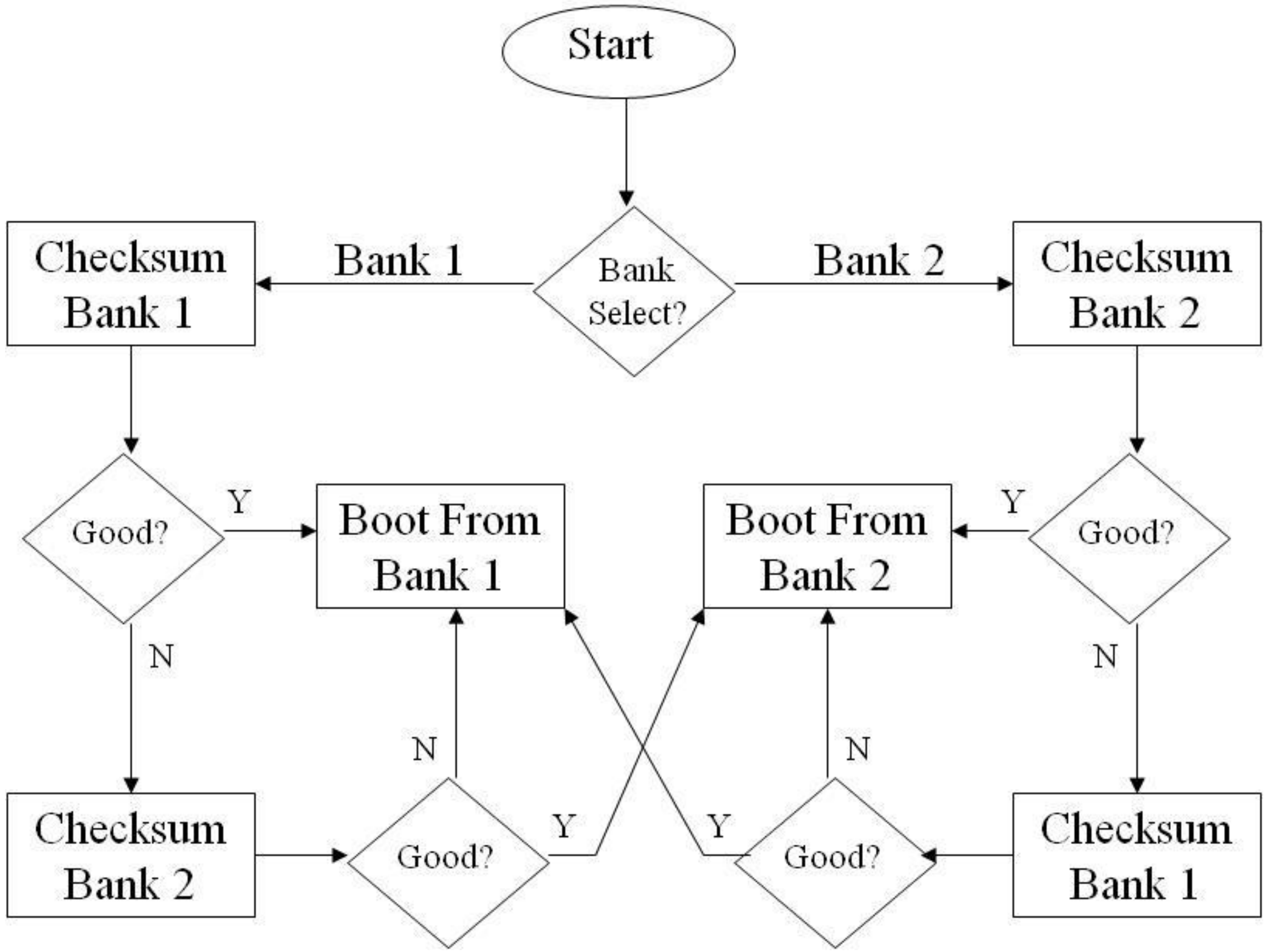
- **Build the system**
  – In the missionxyz/build/cpuX directory run “make”
    - Compiles, links, and installs cFE core, OSAL, and PSP
      - Linked into one core binary file
      - Copies core binary to /build/cpuX/exe directory
    - Compiles and installs each application
      - Unique application object files
      - Copies application object files to /build/cpuX/exe directory and to the PROM location specified in the makefile
        - The PSP defines where the cFS build will load/run from
    - Builds and installs application table files
      - Compiles table source files
      - Creates .tbl files
      - Copies .tbl files to /build/cpuX/exe directory and to the PROM location specified in the makefile

Run Time

Boot Sequence

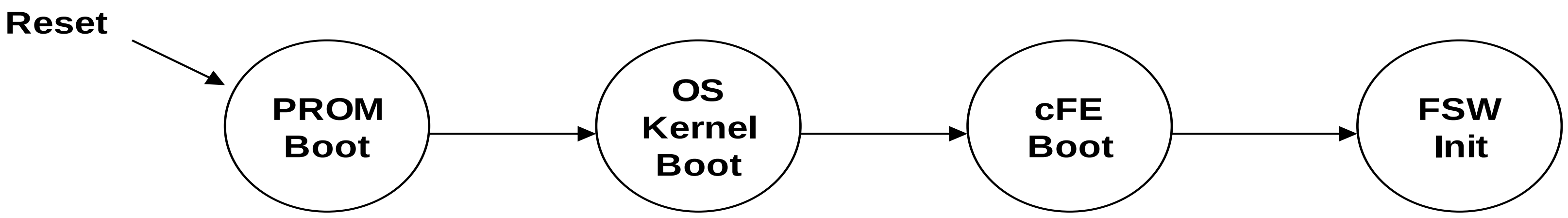
- **For VxWorks and RTEMS the OSAL and boot logic supports two memory allocation models for software startup, static and dynamic. Linux platforms only support dynamic. Regardless of which model is used, system initialization is very similar**
- **Reference mission boot requirements**
  - On a Power-On reset, initialize all processor SRAM
  - Validate the CRC of EEPROM Bank 1
  - Boot EEPROM bank 1 if it passes CRC validation
  - Boot EEPROM bank 2 if bank 1 fails CRC validation
  - Store the boot code in non-volatile storage devices which cannot be modified in flight
  - Provide two copies of the FSW code image in EEPROM
  - Adhere to the margin requirements of GSFC-STD-1000, Rule 3.07

Boot EEPROM Bank selection logic

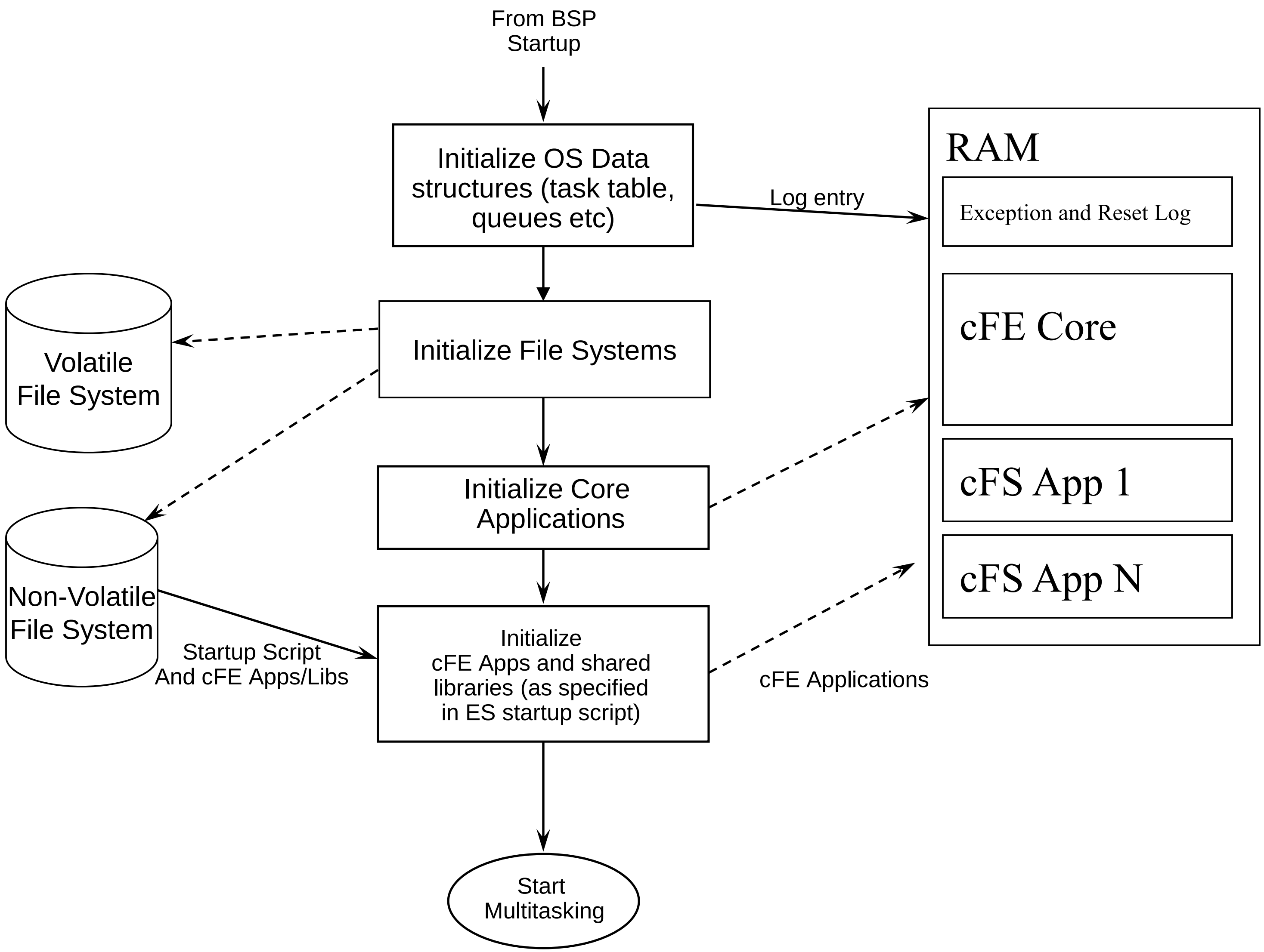


Boot Sequence

- The PROM boots the OS kernel linked with the BSP, loader and EEPROM file system.
  - Accesses simple file system
  - Selects primary and secondary images based on flags and checksum validation
  - Copies OS image to RAM
- The OS kernel boots the cFE
  - Performs self – decompression (optional)
  - Attaches to EEPROM File System
  - Starts up cFE
- cFE boots cFE interface apps and mission components (C&DH, GNC, Science applications)
  - Creates/Attaches to Critical Data Store (CDS)
  - Creates/Attaches to RAM File System
  - Starts cFE applications (EVS, TBL, SB, & TIME)
  - Starts the C&DH and GNC applications based on “cfe_es_startup.scr”



cFE Executive Services Startup



The cFE core is started as one unit. The cFE Core is linked with the RTOS and support libraries and loaded into system EEPROM as a static executable.

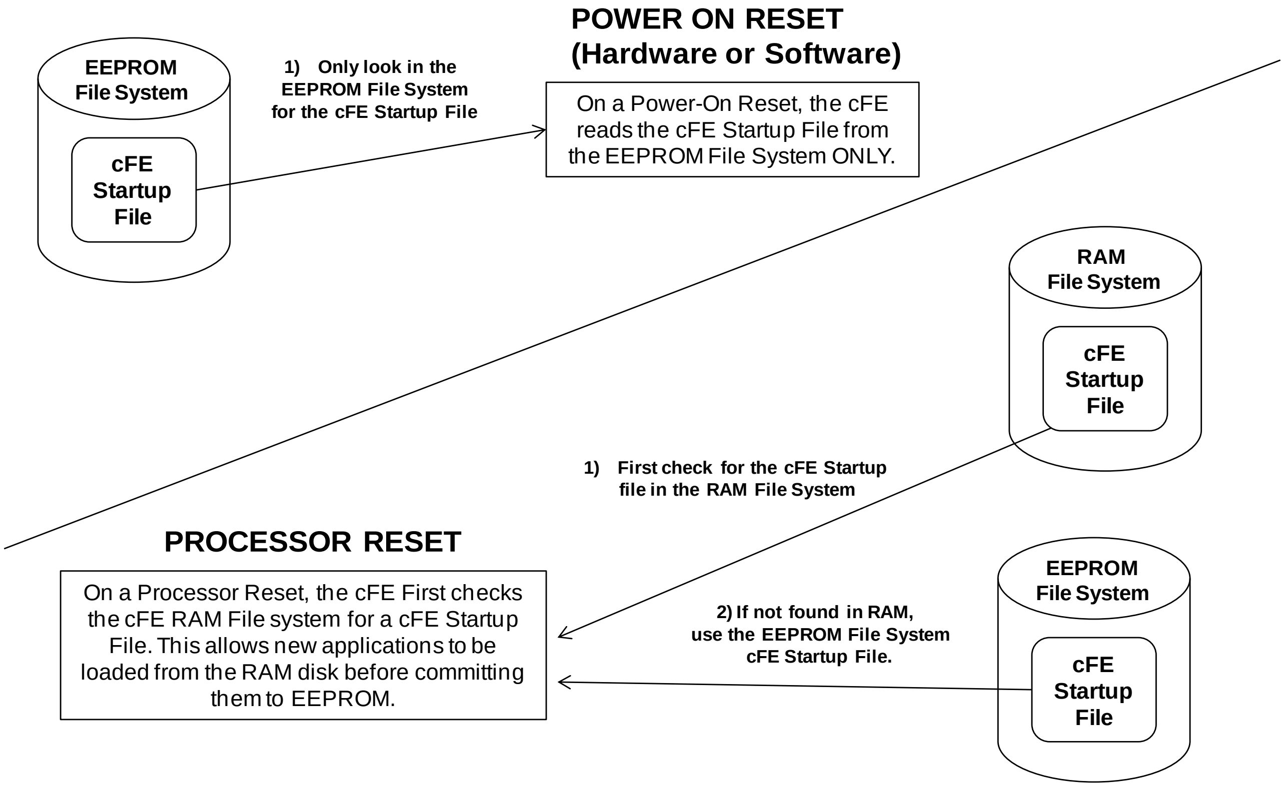
Startup Script

- **The startup script is a text file, written by the user that contains a list of entries (one entry for each application)**
  - Used by the ES application for automating the startup of applications.
  - ES application allows the use of a volatile and nonvolatile startup scripts. The project may utilize zero, one or two startup scripts.

Object Type	`CFE_APP` for an Application, or `CFE_LIB` for a library.
Path/Filename	This is a cFE Virtual filename, not a vxWorks device/pathname
Entry Point	This is the name of the "main" function for App.
CFE Name	The cFE name for the APP or Library
Priority	This is the Priority of the App, not used for a Library
Stack Size	This is the Stack size for the App, not used for a Library
Load Address	This is the Optional Load Address for the App or Library. It is currently not implemented so it should always be 0x0.
Exception Action	This is the Action the cFE should take if the Application has an exception. • 0 = Do a cFE Processor Reset • Non-Zero = Just restart the Application

Component Startup

- **Immediately after the cFE completes its initialization, the ES Application first looks for the volatile startup script referenced by configuration parameter CFE_ES_VOLATILE_STARTUP_FILE.**
- **If ES does not find the file, it attempts to open the file referenced by configuration parameter CFE_ES_NONVOL_STARTUP_FILE.**

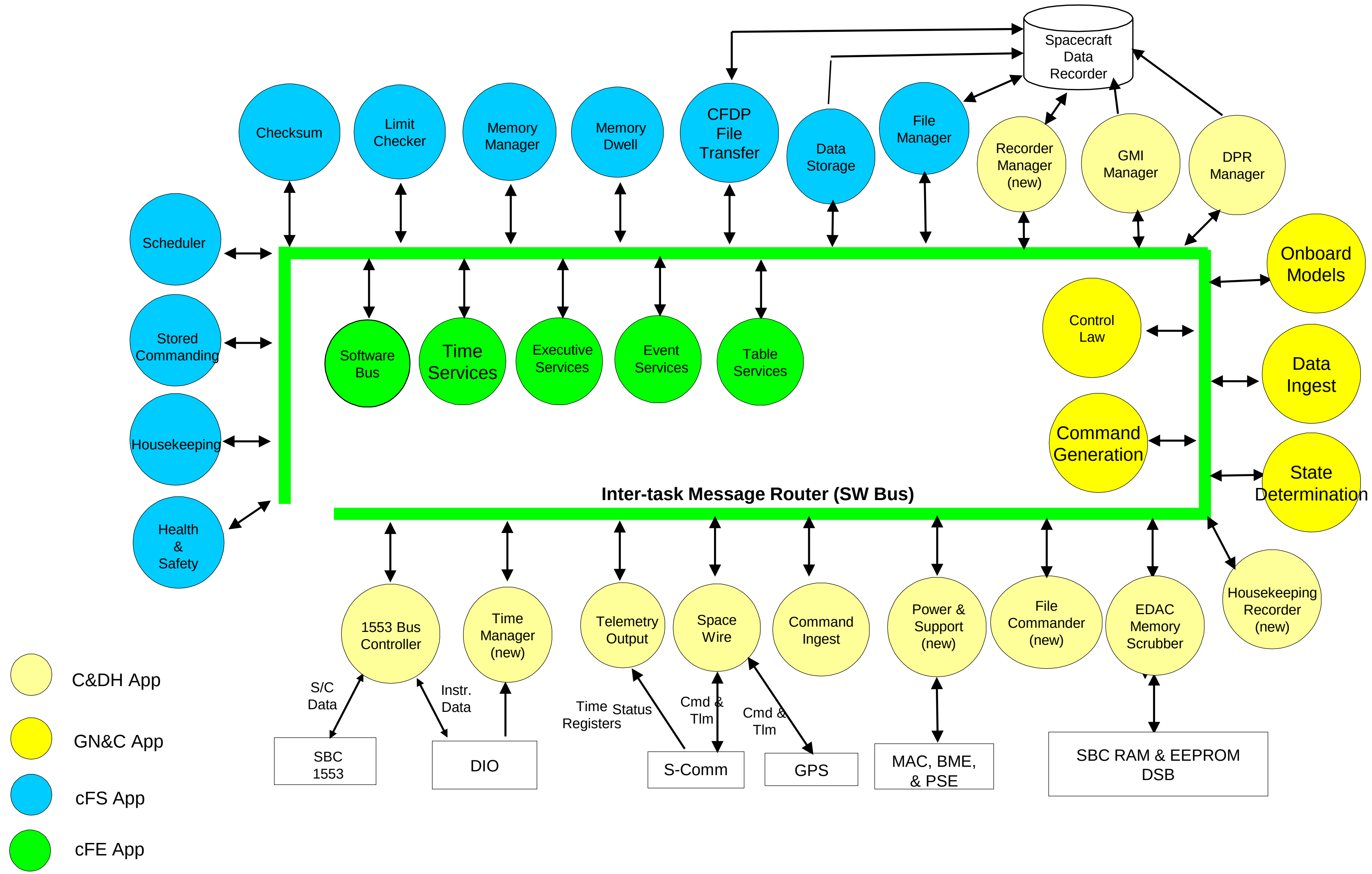


Example cfe_es_startup.scr

```
CFE_APP, /cf/apps/ci_lab.o,   CI_Lab_AppMain,  CI_LAB_APP,     70,   4096, 0x0, 0;
CFE_APP, /cf/apps/sch_lab.o,  SCH_Lab_AppMain, SCH_LAB_APP,   120,   4096, 0x0, 0;
CFE_APP, /cf/apps/to_lab.o,   TO_Lab_AppMain,  TO_LAB_APP,     74,   4096, 0x0, 0;
CFE_LIB, /cf/apps/cfs_lib.o,  CFS_LibInit,     CFS_LIB,         0,      0, 0x0, 0;
!
! Startup script fields:
! 1. Object Type      -- CFE_APP for an Application, or CFE_LIB for a library.
! 2. Path/Filename    -- This is a cFE Virtual filename, not a vxWorks device/pathname
! 3. Entry Point      -- This is the "main" function for Apps.
! 4. CFE Name         -- The cFE name for the the APP or Library
! 5. Priority         -- This is the Priority of the App, not used for Library
! 6. Stack Size       -- This is the Stack size for the App, not used for the Library
! 7. Load Address     -- This is the Optional Load Address for the App or Library. Currently
not implemented
!                        so keep it at 0x0.
! 8. Exception Action -- This is the Action the cFE should take if the App has an exception.
!                        0        = Just restart the Application
!                        Non-Zero = Do a cFE Processor Reset
!
! Other  Notes:
! 1. The software will not try to parse anything after the first '!' character it sees. That
!    is the End of File marker.
! 2. Common Application file extensions:
!    Linux = .so  ( ci.so )
!    OS X  = .bundle  ( ci.bundle )
!    Cygwin = .dll ( ci.dll )
!    vxWorks = .o ( ci.o )
!    RTEMS with S-record Loader = .s3r ( ci.s3r )
!    RTEMS with CEXP Loader = .o ( ci.o )
```

Mission Examples

Example Mission 1 - Software Architecture



Example Mission 1 - Noteworthy Items

- cFE was very reliable and stable
- Easy rapid prototyping with heritage code that was cFE compliant
- Layered architecture has allowed COTS lab to be maintained through all builds
- Lines of Code Percentages:

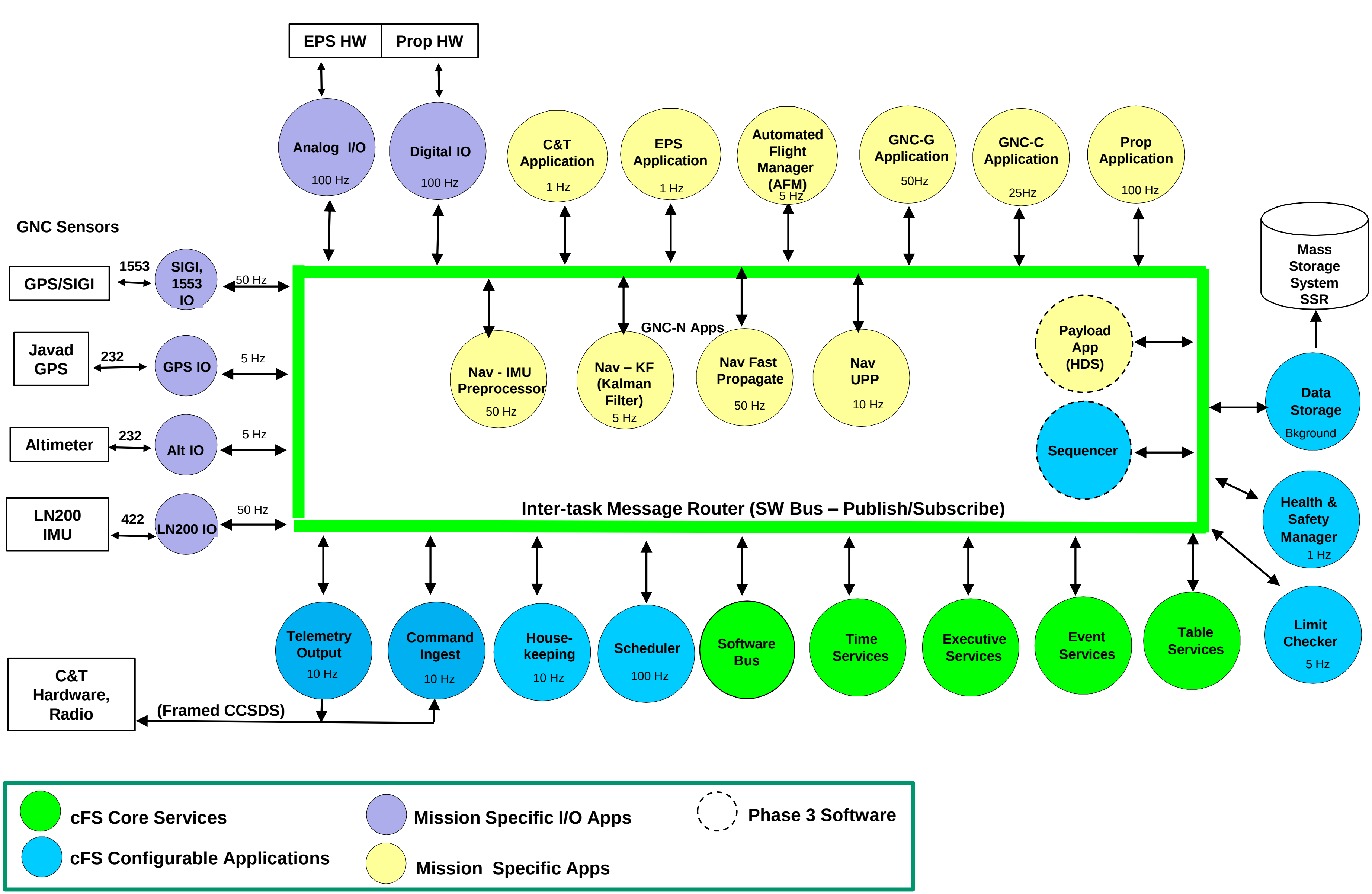
Source	Percentage
BAE	0.3
EEFS	1.7
OSAL	2.1
PSP	1.0
cFE	12.4
GNC Library	1.6
cFS Applications	23.5
Heritage Clone & Own	38.9
New Source	18.5

Example Mission 1
SLOC Count with SEER-SEM Estimates

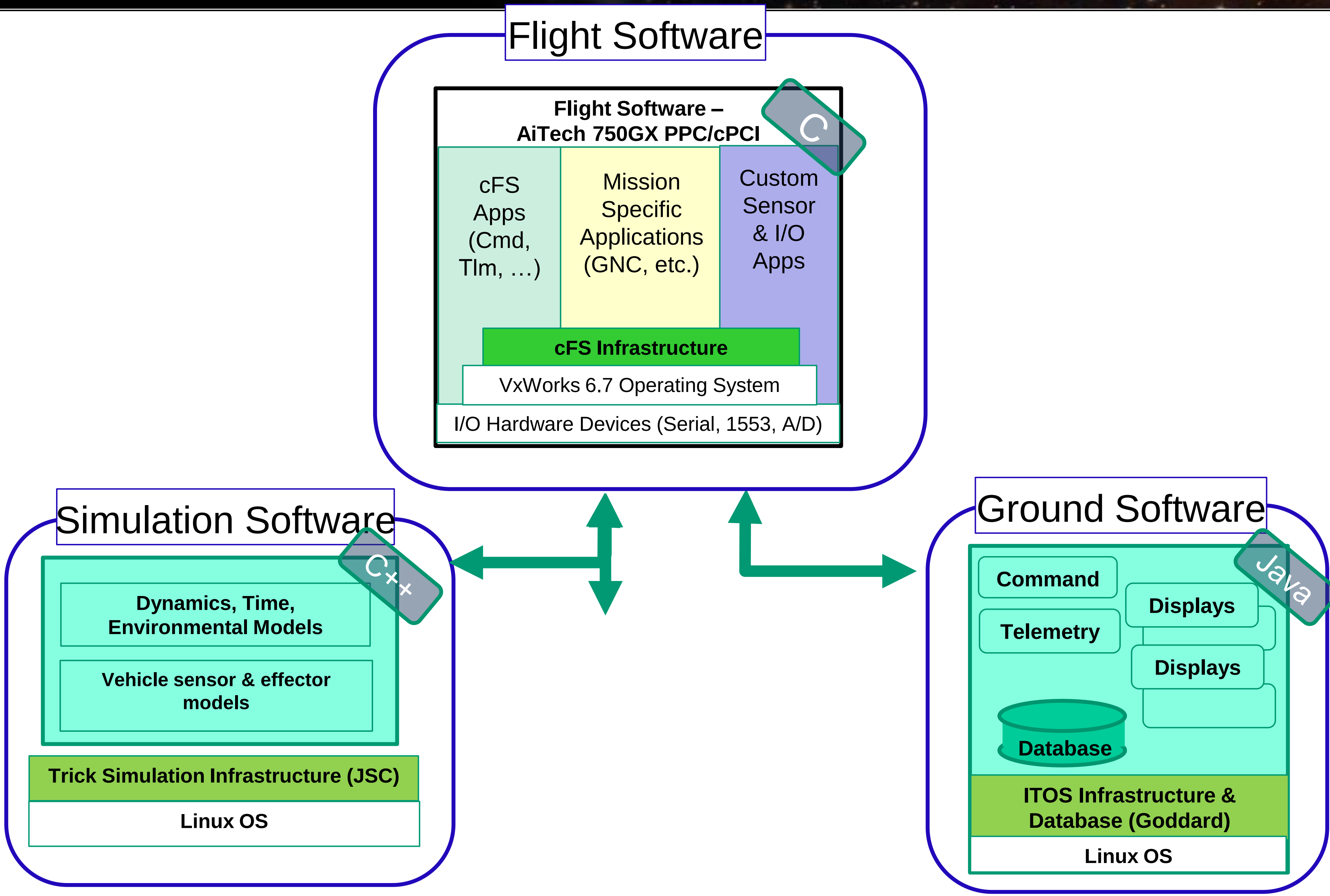
Module Name	SLOC (Logical)	New Do-178b Class A		New DO-178b Class B		GSFC Clone & Own Class B		CFS Use Class B	
		Effort hours	Cost	Effort hours	Cost	Effort hours	Cost	Effort hours	Cost
OS API & OSAL	2,338	6,205		5,291		901	$ -	278	
BSP	1,492	3,960		3,376		575	$ -	178	
Executive Services	4,737	12,572		10,720		1,826	$ -	564	
Event Service	1,429	3,793		3,234		551	$ -	170	
File System	763	2,025		1,727		294	$ -	91	
Mission Config Include Files	1,857	4,928		4,202		716	$ -	221	
Software Bus	2,017	5,353		4,564		777	$ -	240	
Table Service	2,182	5,791		4,938		841	$ -	260	
Time Service	1,941	5,151		4,392		748	$ -	231	
cFE Configuration (hdr files)	226	600		511		87	$ -	27	
cFE platform Support Pkg	827	2,195		1,872		319	$ -	98	
CFS Library	166	441		376		64	$ -	20	
Checksum	2,811	7,460		6,361		1,083	$ -	335	
File Manager	1,664	4,416		3,766		641	$ -	198	
File Commanding	447	1,186		1,012		172	$ -	53	
Health & Safety	1,433	3,803		3,243		552	$ -	171	
Memory Manager	1,927	5,114		4,361		743	$ -	229	
Scheduler	1,067	2,832		2,415		411	$ -	127	
Mode Manager	2,175	5,772		4,922		838	$ -	259	
Housekeeping	554	1,470		1,254		214	$ -	66	
Stored Commands	2,033	5,396		4,601		784	$ -	242	
Limit Checker	1,812	4,809		4,101		698	$ -	216	
Memory Dwell	1,003	2,662		2,270		387	$ -	119	
cFDP	8,286	21,991		18,751		3,193	$ -	986	
Total SLOC	***45,187***	**119,746**		**102,258**		**17,415**	**$ -**	**5,377**	
Defect Prediction		***30***		***52***		***8***		***2***	
		57.8 FTE		**49.6 FTE**		**8.2 FTE**		**2.6 FTE**	

(JPL SLiC tool)

Example Mission 2 - Software Architecture



Example Mission 2 - Software Components



National Aeronautics and Space Administration

Flight Software Lessons Learned

Plan Ahead!

FSW Lessons Learned

- **FSW involvement starts during early mission formulation stages**
  - Participate in ground/flight trades, hardware/software trades, mission cost estimates
- **Formal Development and Test Processes do pay-off**
  - NPR 7150.2
- **Detailed FSW Requirements are critical**
  - ‘Communicate’ exactly what FSW will do
  - Create clear agreement among developers, testers, Systems Engineers, Ground Operators, Hardware subsystem engineers
- **Interface Control Documents are critical**
  - Must get detailed hardware and software interface definitions in writing and signed-off

Lessons Learned

- **Do not compromise on Testing!**
- **Formal and Informal Review of FSW Test Scenarios, Test Scripts and Test Results should be held**
  - FSW Engineers, Project Systems Engineers, Hardware Subsystem Analysts, Operations
  - Walkthroughs find errors
  - Formal Reviews (Standup Presentations) facilitate Project level resolution of FSW risks
- **High Fidelity FSW Testbed is very important**
  - FSW must execute on flight-like hardware
  - Simulations must accommodate ground validation of FSW
  - Essential for post-launch maintenance of FSW

Backup

Acronym List - 1

API	Application Programmer Interface
APL	Applied Physics Lab
ASIST	Advanced Spacecraft Integration and System Testing
ATS	Absolute Time Sequence
BC	Bus Controller
BT	Build Test
bps	bits-per seconds
Bps	Bytes-per seconds
BSP	Board Support Package
C&DH	Command and Data Handling
CCSDS	Consultative Committee for Space Data Systems
CDS	Critical Data Store
CESE	Center for Experimental Software Engineering
CFDP	CCSDS File Delivery Protocol
cFE	Core Flight Executive
cFS	Core Flight Software System
CM	Configuration Management
CMD	Command
COTS	Commercial Off The Shelf
cPCI	Compact PCI
CRC	Cyclic Redundancy Check
CS	Checksum
DMA	Direct Memory Access
DS	Data Storage
EEPROM	Electrically Erasable Programmable Read-Only Memory
EOF	End of File
ES	Executive Services
EVS	Event Services
FDC	Failure Detection and Correction
FDIR	Failure Detection, Isolation, and Recovery
FM	File Management, Fault Management
FSW	Flight Software

Acronym List - 2

GNC	Guidance Navigation and Control
GSFC	Goddard Space Flight Center
GOTS	Government Off The Shelf
GPM	Global Precipitation Measurement
GPS	Global Positioning System
Hi-Fi	High-Fidelity Simulation
HK	Housekeeping
HS	Health & Safety
HW	Hardware
Hz	Hertz
I&T	Integration and Test
ICD	Interface Control Document
IPP	Innovative Partnership Program Office
IRAD	Internal Research and Development
ITAR	International Traffic in Arms Regulations
ISR	Interrupt Service Routine
ITOS	Integration Test and Operations System
IV&V	Independent Verification and Validation
JHU	Johns Hopkins University
KORI	Korean Aerospace Research Institute
LADEE	Lunar Atmosphere and Dust Environment Explorer
LC	Limit Checker
LDS	Local Data Storage
LRO	Lunar Reconnaissance Orbiter
Mbps	Megabits-per seconds
MD	Memory Dwell
MET	Mission Elapsed Timer
MM	Memory Manager
MS	Memory Scrub
NACK	Negative-acknowledgement
NASA	National Aeronautics Space Agency

Acronym List - 3

NESC	NASA Engineering and Safety Center
NOOP	No Operation
OS	Operating System
OSAL	Operating System Abstraction Layer
PCI	Peripheral Component Interconnect
PSP	Platform Support Package
RAM	Random-Access Memory
RM	Recorder Manager
ROM	Read-Only Memory
RT	Remote Terminal
R/T	Real-time
RTOS	Real-Time Operating System
RTS	Relative Time Sequence
SARB	Software Architecture Review Board
S/C	Spacecraft
SB	Software Bus
SBC	Single-Board Computer
SC	Stored Command
SCH	Scheduler
S-COMM	S-Band Communication Card
SDO	Solar Dynamic Observatory
SDR	Spacecraft Data Recorder
SIL	Simulink Interface Layer
SpW	Spacewire
SRAM	Static RAM
SSR	Solid State Recorder
STCF	Spacecraft Time Correlation Factor
SUROM	Start-Up Read-Only Memory
SW	Software, Spacewire
TAI	International Atomic Time
TBD	To be determined

Acronym List - 4

TBL	Table Services
TLM	Telemetry
TDRS	Tracking Data Relay Satellite
TM	Time Manager
TO	Telemetry Output
TRMM	Tropical Rainfall Measuring System
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UMD	University of Maryland
UT	Unit Test
UTC	Coordinated Universal Time
VCDU	Virtual Channel Data Unit
XB	External Bus
XBI	Instrument 1553 External Bus
XBS	Spacecraft 1553 External Bus