

Core Flight System Framework



Version 1.10
December 2023

- **Objectives**

- Describe the core Flight Executive (cFE) from a functional perspective

- **Intended audience**

- Mostly targeted at software engineers, but systems engineers, non-FSW spacecraft discipline engineers, and technical project managers could also benefit.

- **Prerequisites**

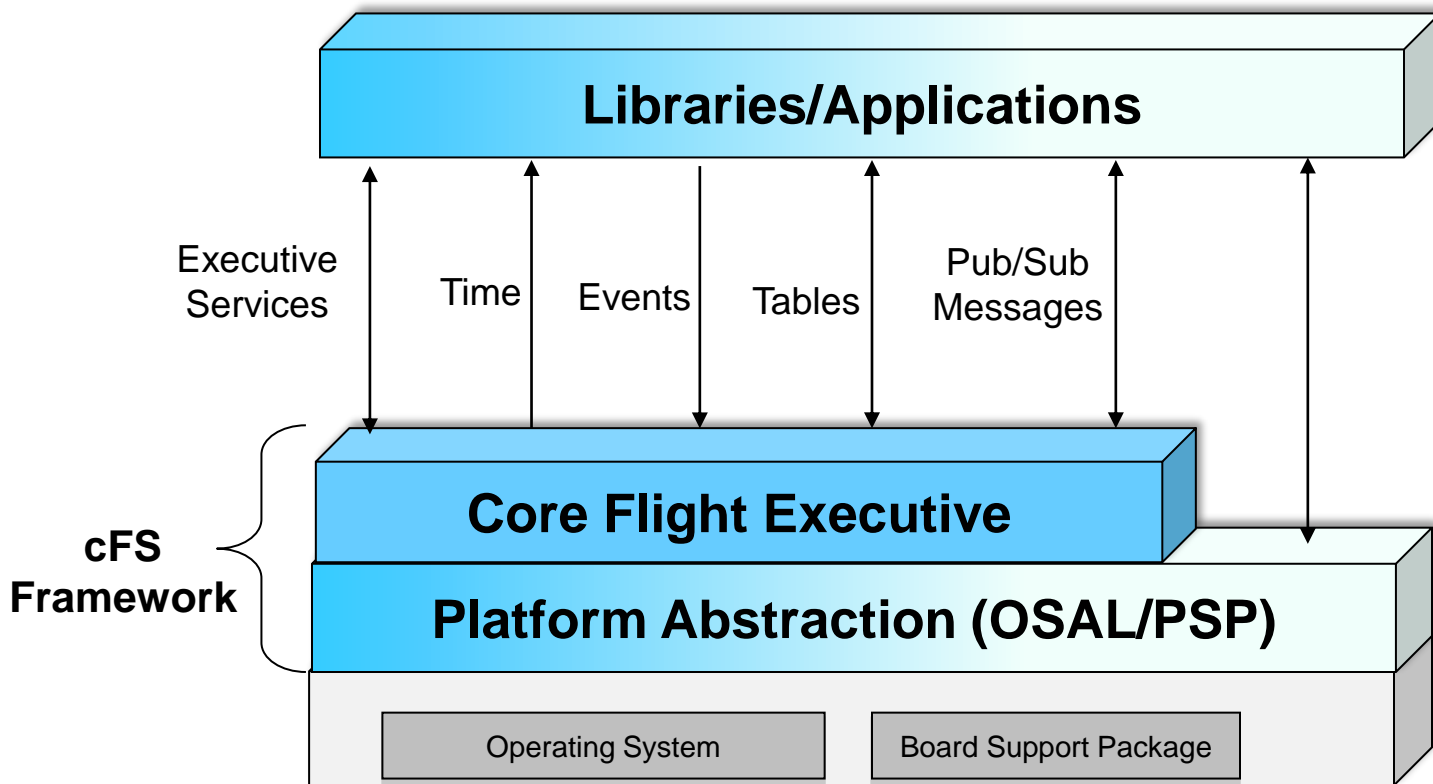
- Course Introductory material provided in the cFS overview slides and video

- **Hands-on cFS Basecamp Tutorials**

- cFS Basecamp, <https://github.com/cfs-tools/cfs-basecamp> , is a lightweight environment with built-in tutorials for learning the cFS
- Slides marked with the cFS Basecamp logo  provide instructions for launching the tutorials

1. cFS Layered Architecture
2. cFS Framework Services
3. Application Layer Architectural Components
4. cFS Framework Deployment

cFS Layered Architecture



OSAL: Operating System Abstraction Layer
PSP: Platform Support Package

Executive Services (ES)

- Manage the software system and create an application runtime environment

Time Services (TIME)

- Manage spacecraft time

Event Services (EVS)

- Provide a service for sending, filtering, and logging event messages

Software Bus (SB) Services

- Provide an application publish/subscribe messaging service

Table Services (TBL)

- Manage application table images

Operating System (OS)

- System software that manages computer hardware and software resources, and provides common services for computer programs
- Software services include scheduling different threads of execution, facilitating communication between threads, and managing memory for the entire system

Realtime Operating System (RTOS)

- Supports multi-tasking with preemptive scheduling so time critical tasks execute deterministically

Board Support Package (BSP)

- Contains hardware-specific boot firmware, device drivers and other routines to an operating system to function in a given hardware environment (i.e. a motherboard)

Operating System Abstraction Layer (OSAL)

- A software library that provides a single Application Program Interface (API) to the core Flight Executive (cFE) regardless of the underlying operating system

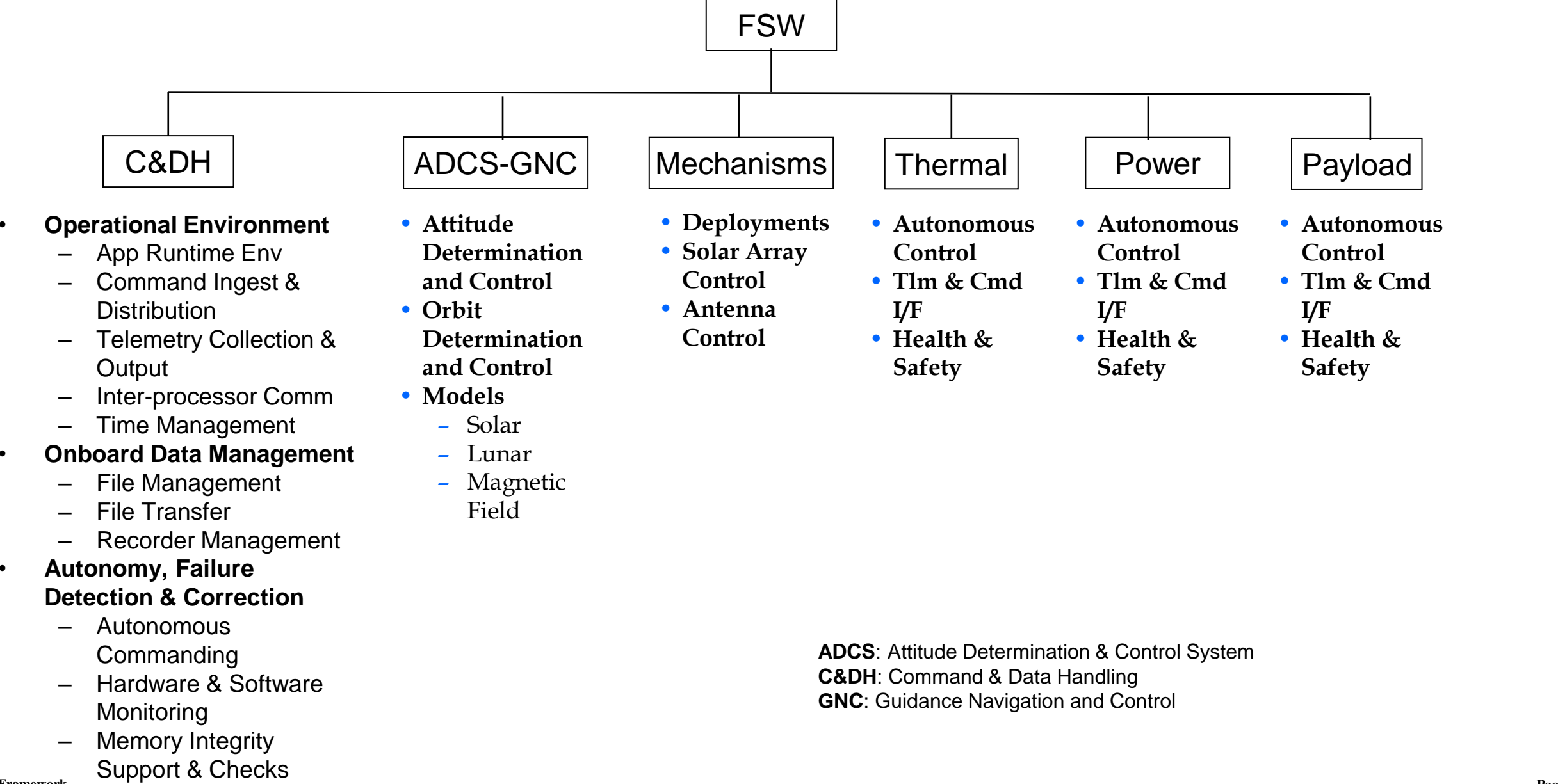
Platform Support Package (PSP)

- A software library that provides a single API to the underlying hardware board and BSP
- Library serves as the "glue" between the OS and the cFE
- During system initialization it performs BSP/OS specific setup and then calls cFE's entry point function

Notes

- The cFS Framework defaults to the Linux and can be run on a personal computer
- The cFS Platform List <https://github.com/cfs-tools/cfs-platform-list> provides links to cFS ports

- **Describe high level concept of using a combination reusable and mission specific apps to satisfy mission requirements**
- The Systems Engineering Document contain complete details
- **Applications are an architectural component that owns cFE and operating system resources via the cFE and OSAL Application Programmer Interfaces (APIs)**
- **cFE Services provide an Application Runtime Environment**
- **The cFE service and Platform Abstraction APIs provide a portable functional interface**
- **Write once run anywhere the cFS framework has been deployed**
 - Defer embedded software complexities due to cross compilation and target operating systems
 - Smartphone apps need to be rewritten for each platform
 - Provides seamless application transition from technology efforts to flight projects



- TODO

cFS Framework Services

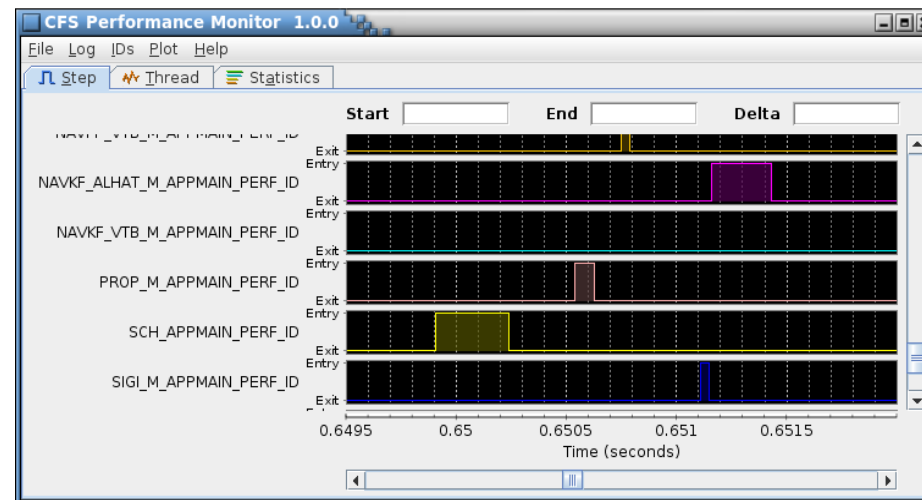
1. Executive Service (ES)
2. Event Service (EVS)
3. Software Bus Service (SB)
4. Table Service (TBL)
5. Time Service (TIME)
6. Operating System Abstraction Layer (OSAL)
7. Platform Support Package (PSP)

- **This section briefly introduces each cFE service's functionality**
 - **TBD** provides detailed material on each service with demos and self-guided tutorials

- **Initializes the cFE**
 - Reports reset type
 - Maintains an exception-reset log across processor resets
- **Creates the application runtime environment**
 - Primary interface to underlying operating system task services
 - Manages application resources
 - Supports starting, stopping, and loading applications during runtime
- **Memory Management**
 - Provides a dynamic memory pool service
 - Provides Critical Data Stores (CDS) which are memory blocks that are preserved across processor resets if the platform supports it

- Applications are an architectural component that owns cFE and operating system resources
- Each application has a thread of execution in the underlying operating system (i.e. a task)
- Applications can create multiple child tasks
 - Child tasks share the parent task's address space
- Mission applications are defined in *cfe_es_startup.scr* and loaded after the cFE applications are created
- Application Restarts and Reloads
 - Start, Stop, Restart, Reload commands
 - Data is not preserved; application run through their initialization
 - Can be used in response to
 - Exceptions
 - On-board Failure Detection and Correction response
 - Ground commands

- **Provides a method to identify and measure code execution paths**
 - System tuning, troubleshooting, CPU loading
- **Executive Service provides Developer inserts execution markers in FSW**
 - Entry marker indicate when execution resumes
 - Exit marker indicates when execution is suspended
 - CFE_ES_PerfLogExit() => CFE_SB_RcvMsg() => CFE_ES_PerfLogEntry()
- **Operator defines what markers should be captured via filters and defines triggers that determine when the filtered marker are captured**
- **Captured markers are written to a file that is transferred to the ground and displayed using the cFS Performance Monitor (CPM) tool**



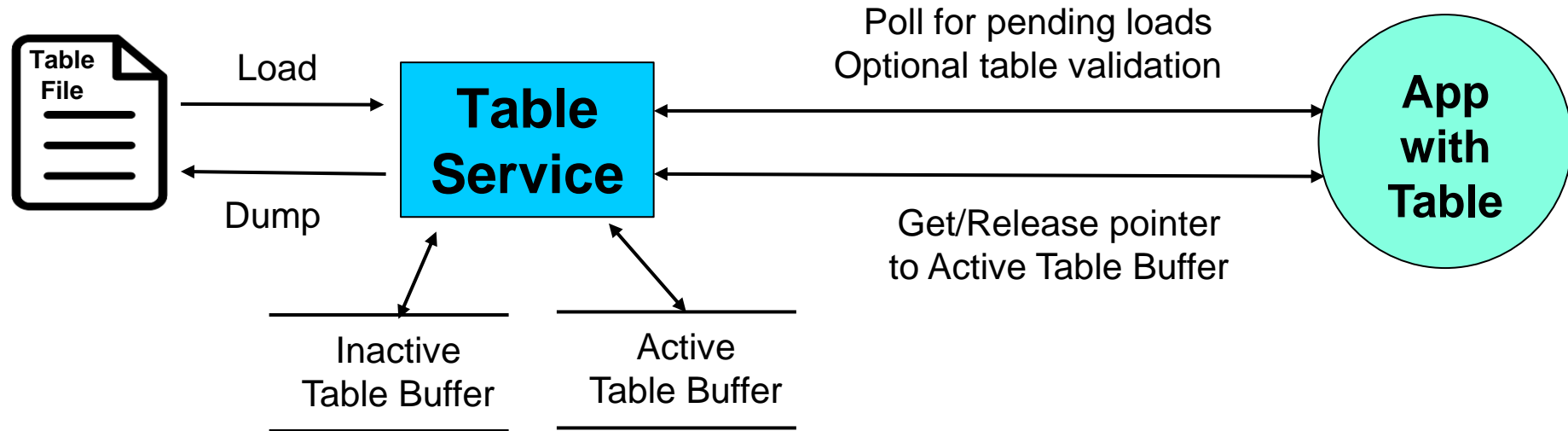
- **Provides an interface for sending time-stamped text messages on the software bus**
 - Considered asynchronous because they are not part of telemetry periodically generated by an application
 - Processor unique identifier
 - Optionally logged to a local event log
 - Optionally output to a hardware port
- **Four event types defined**
 - Debug, Informational, Error, Critical
- **Event message control**
 - Apps can filter individual messages based on identifier
 - Enable/disable event types at the processor and application scope

- **“Filter Mask”**
 - Bit-wise Boolean AND performed on event ID message counter, if result is zero then the event is sent
 - Mask applied before the sent counter is incremented
 - 0x0000 => Every message sent
 - 0x0003 => Every 4th message sent
 - 0xFFFE => Only first two messages sent
- **Reset filter**
 - Filters can be reset from an application or by command
- **Event filtering example**
 - Software Bus ‘No Subscriber’ event message, Event ID 14
 - See *cfe_platform_cfg.h* CFE_SB_FILTERED_EVENT1
 - Default configuration is to only send the first 4 events
 - Filter Mask = 0xFFFC
- **CFE_EVS_MAX_FILTER_COUNT (cfe_evs_task.h) defines maximum count for a filtered event ID**
 - Once reached event becomes locked
 - Prevents erratic filtering behavior with counter rollover
 - Ground can unlock filter by resetting or deleting the filter

- **Processor scope**
 - Enable/disable event messages based on type
 - Debug, Information, Error, Critical
- **Application scope**
 - Enable/disable all events
 - Enable/disable based on type
- **Event message scope**
 - During initialization apps can register events for filtering for up to CFE_EVS_MAX_EVENT_FILTERS defined in *cfe_platform_cfg.h*
 - Ops can add/remove events from an app's filter

- **Provides an inter-application message service using a publish/subscribe model**
- **Routes messages to all applications that have subscribed to the message (i.e. broadcast model)**
 - Subscriptions are done at application startup
 - Message routing can be added/removed at runtime
 - Sender does not know who subscribes (i.e. connectionless)
- **Reports errors detected during the transferring of messages**
- **Outputs Statistics Packet and the Routing Information when commanded**

- **What is a table?**
 - Tables are logical groups of parameters that are managed as a named entity
- **Parameters typically change the behavior of a FSW algorithm**
 - Examples include controller gains, conversion factors, and filter algorithm parameters
- **Tables service provides ground commands to load a table from a file and dump a table to a file**
 - Table loads are synchronized with applications
- **Tables are binary files**
 - Ground support tools are required to create and display table contents
- **The cFE can be built without table support**
 - Note the cFE applications don't use tables



- **Table service contains buffers that hold tables for all applications**
 - Active Table Buffer - Image accessed by app while it executes
 - Inactive Table Buffer - Image manipulated by ops (could be stored commands)
- **“Table Load” is a sequence of activities to transfer data from a file to the Active Table Buffer**
- **“Table Dump” is a sequence of activities to transfer data from a either Table Buffer to a file**
- **Table operations are synchronous with the application that owns the table to ensure table data integrity**

- **cFE Time Services provides time correlation, distribution and synchronization 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**
- **Designing and configuring time is tightly coupled with the mission avionics design**

- **Supports two formats**
- **International Atomic Time (TAI)**
 - Number of seconds and sub-seconds elapsed since the ground epoch
 - $\text{TAI} = \text{MET} + \text{STCF}$
 - Mission Elapsed Counter (MET) time since powering on the hardware containing the counter
 - Spacecraft Time Correlation Factor (STCF) set by ground ops
 - Note STCF can correlate MET to any time epoch so TAI is mandated
- **Coordinated Universal Time (UTC)**
 - Synchronizes time with astronomical observations
 - $\text{UTC} = \text{TAI} - \text{Leap Seconds}$
 - Leap Seconds account for earth's slowing rotation

- **TBD**

OpenSatKit Training osal-ng.ppt

- **TBD**

1. cFS Basecamp must be installed following the instructions at <https://github.com/cfs-tools/cfs-basecamp>
2. From the Tutorial dropdown list select “Basecamp Introduction” and do Lesson 1 “Basecamp Feature Overview”

1

Tutorials

Basecamp Introduction

Build and Run the cFS

cFS GitHub App Exchange

Create App Tool

Hello World

Hello Object

2

Basecamp Introduction

Objectives

Introduce users to Basecamp features and capabilities.

Lesson	Complete
<input checked="" type="radio"/> 1-Basecamp Feature Overview	No
<input type="radio"/> 2-Basecamp Demo App	No
<input type="radio"/> 3-Basecamp Next Steps	No

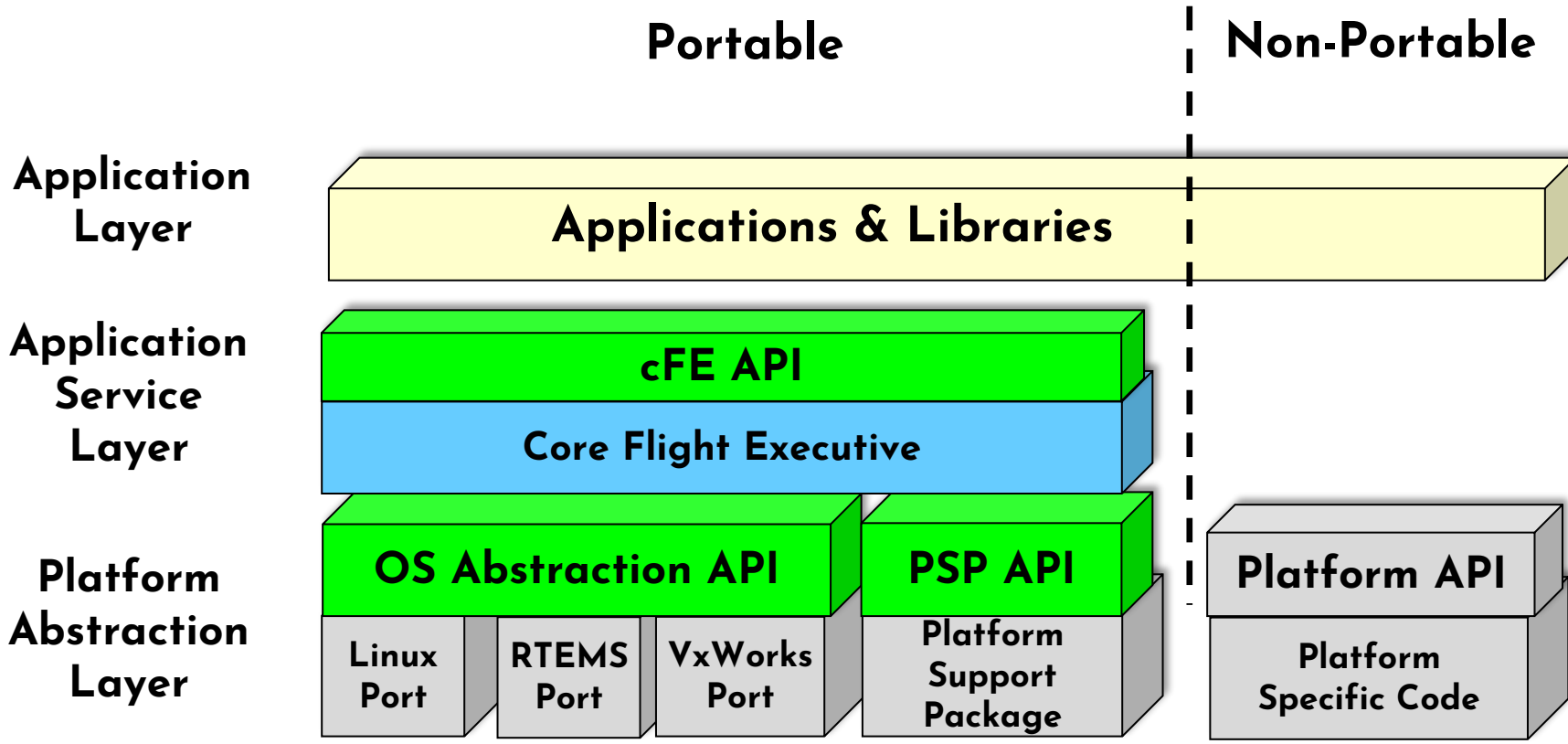
Start

Reset

Exit

Application Layer Architectural Components

- The cFS Application Layer contains Library and Application architectural components
- Architectural components have well defined context, interfaces, and relationships with other components
- A cFS library/application context is bounded by the following interfaces
 1. cFE, OSAL, and PSP services and Application Programmer Interfaces (APIs)
 2. The functional interfaces (APIs) defined by libraries
 3. The message interfaces defined by applications
 4. Platform-specific APIs
- This section discusses the first three interfaces
 - The cFS Systems Engineering document includes design patterns for platform-specific APIs
- An essential cFS architectural design concept is that libraries and apps can be designed with interdependencies that allow groups of libs/apps to provide mission functionality
 - This section introduces the “Operations Service App Suite” that must be included in every cFS target
 - The cFS Systems Engineering document contains more examples functional lib/app groups



- Libraries and apps are portable if they only use the cFS APIs shown in green
- Many mission specific apps may not be portable to a new mission unless the same platform is used
 - This does not make them “non-compliant” components, just non-portable
 - Organizational goals, budgets and schedules drive whether to develop reusable component decisions

- **What is an Application?**
 - A thread of execution managed by the platform's operating system
 - They acquire and own resources using the Platform Abstraction and Application Service APIs
- **Resources are typically acquired during initialization and released when an application terminates**
 - Helps create a deterministic steady-state system
 - Helps achieve the architectural goal for a loosely coupled system that is scalable, interoperable, testable (each app can be separately unit tested), and maintainable
- **Apps can be reloaded during operations without rebooting**

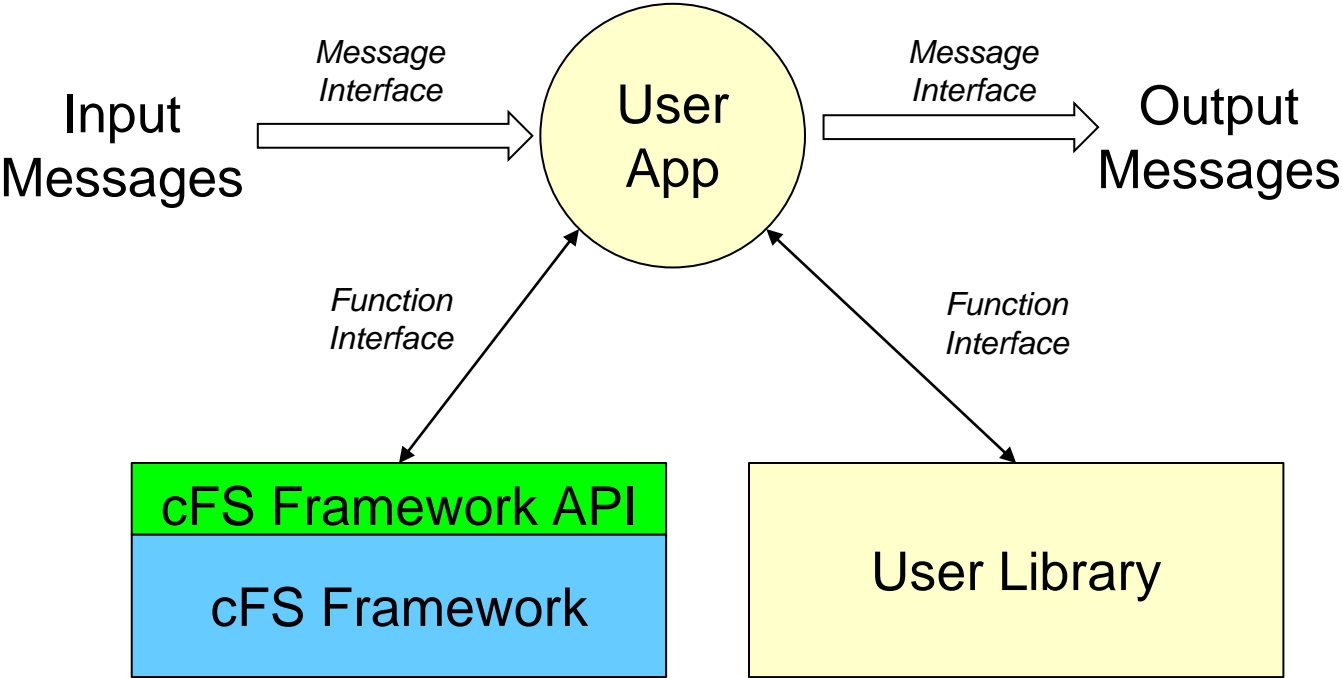
- **Concurrent execution model**
 - Each app has its own priority-based thread of execution and can spawn child tasks with their own priority-based thread of execution
 - Supports complex realtime mission requirements
- **Reusable apps only use the cFS APIs**
 - Write once run anywhere the cFS framework has been deployed
 - Can be written in a desktop environment deferring embedded software development complexities such as cross compilation and target operating systems
 - Provides seamless application transition from technology efforts to flight projects
 - More powerful than the Smartphone situation where different apps are written for each platform

- **What is a library?**
 - A collection of functions and data that are available for use by apps and other libraries
 - Architecturally they exist within the application layer
 - They cannot create tasks and they assume the AppID/TaskID of the caller
- **Libraries are not registered with Executive Services and do not have a thread of execution, so they have limited cFE API usage. For example,**
 - A library can't call CFE_EVS_Register() during initialization
 - The ES API does not provide a function for libraries analogous to CFE_ES_GetAppInfo()
- **Library functions execute within the context of the calling application**
 - CFE_EVS_SendEvent() will identify the calling app
 - Libraries can't register for cFE services during initialization and in general should not attempt to do so

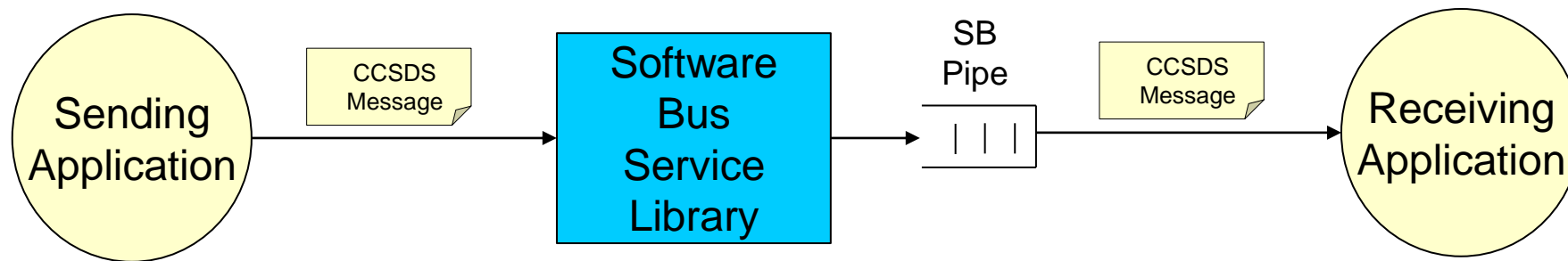


- **Libraries can either be statically or dynamically linked**
 - Dynamic linking requires support from the underlying operating system
- **No cFE API exists to retrieve library code segment addresses**
 - Prevents apps like Checksum from accessing library code space
- **Libraries are specified in the `cfe_es_startup.scr` and loaded during cFE initialization**
 - When using dynamic linking, libraries must be loaded prior to components that use them
- **For libraries that require a ground interface, or some other more complex runtime environment, a helper app is created to provide this support**
 - The cFE's service design uses this approach

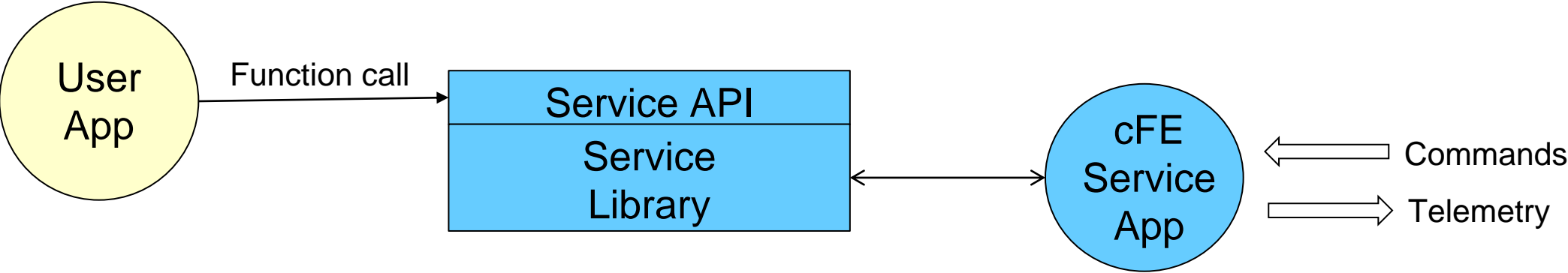




A standardize message interface allows portability



- **One-to-many message broadcast model**
 - Applications publish messages without knowledge of destinations
- **To receive messages, applications create an *SB Pipe* (a *FIFO queue*) and subscribe to messages**
 - Typically performed during application initialization
- **If needed, apps can subscribe and unsubscribe to messages at any time for runtime reconfiguration**
- ***SB Pipes* used for application data and control flow**
 - Poll and pend for messages



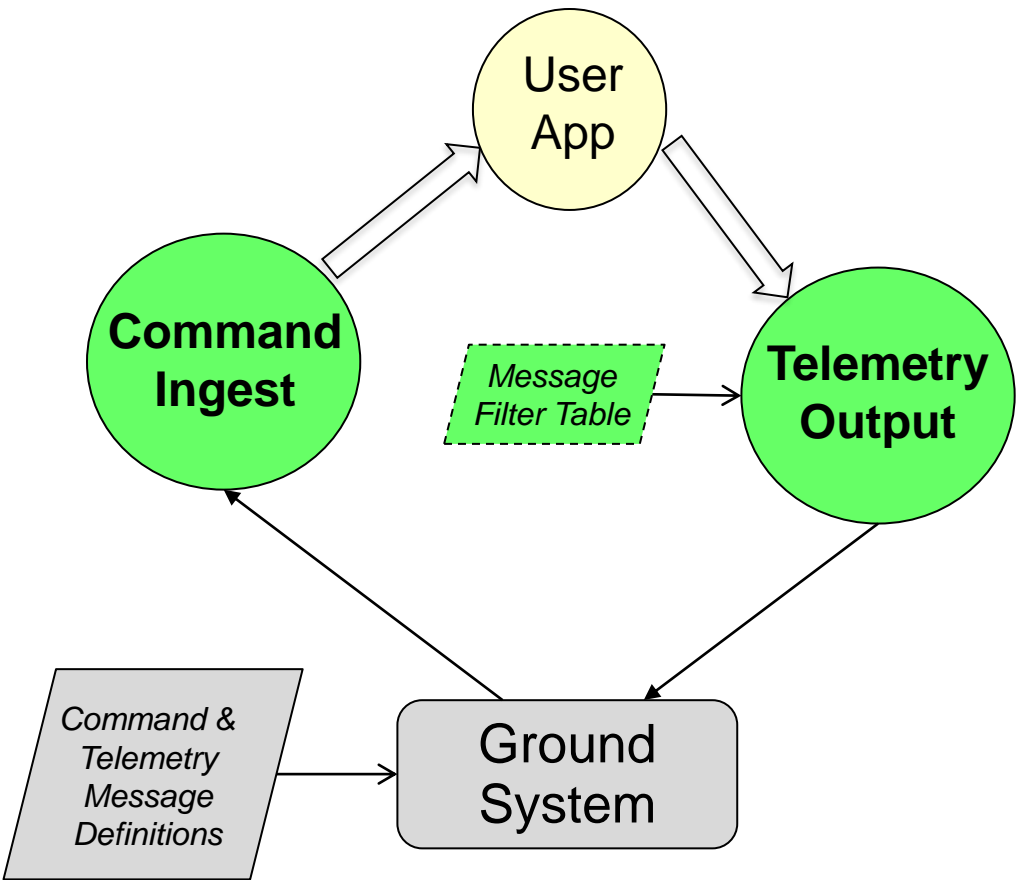
- **Each cFE service has**
 - A library that is used by applications
 - An application that provides a ground interface for operators to use to manage the service
- **Each cFE Service App periodically sends status telemetry in a “Housekeeping (HK) Packet”**
 - Housekeeping is an historical term that means an application’s status
- **You can obtain additional service information beyond the HK packet with commands that**
 - Send one-time telemetry packets
 - Write onboard service configuration data to files

 = Software Bus Message

- **The Operations Service APP Suite is a group of apps that provide functionality required by every cFS target in an operational system**
 - A target needs to communicate (receive commands and send telemetry) with at least one external system, typically a ground system
 - The cFS relies on files so a mechanism for transferring files between the target and an external system is needed, as well as remotely managing the target's directories and files
 - The cFS promotes designing synchronous systems so having an app synched with a 1Hz signal that sends periodic scheduling messages helps achieve this goal

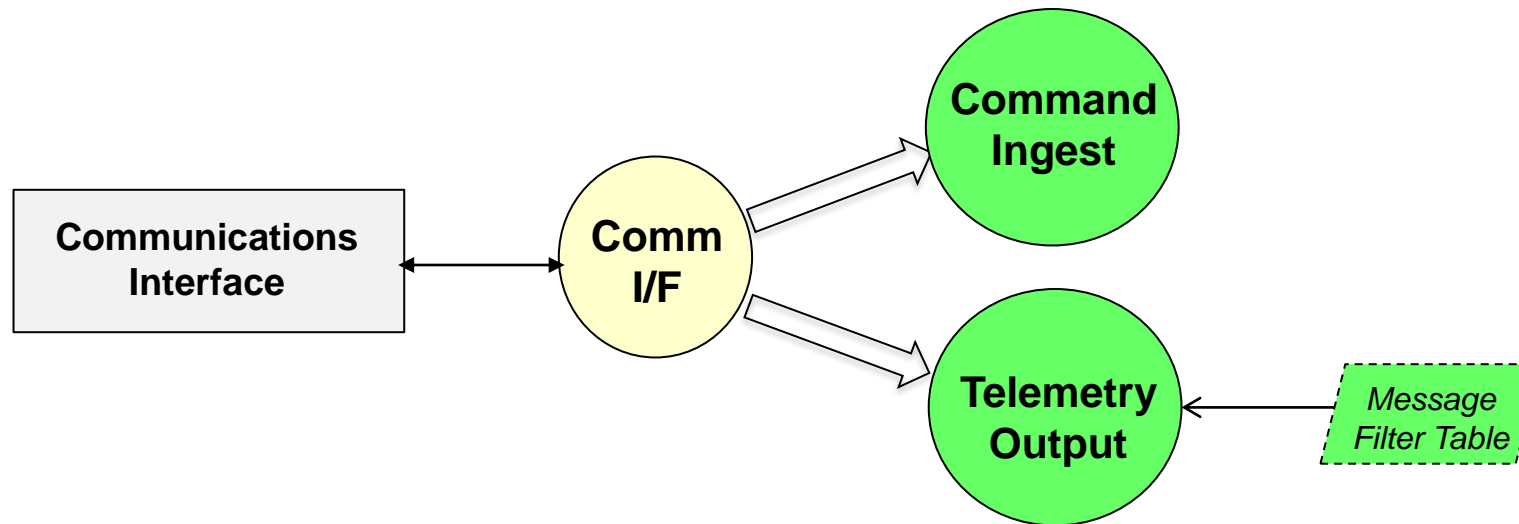
- **An Operations Service App Suite is included in Basecamp's default Target**

*** A cFS target is an instantiation of the cFE Framework on a platform with a set of library and apps. Not to be confused with a distribution.*

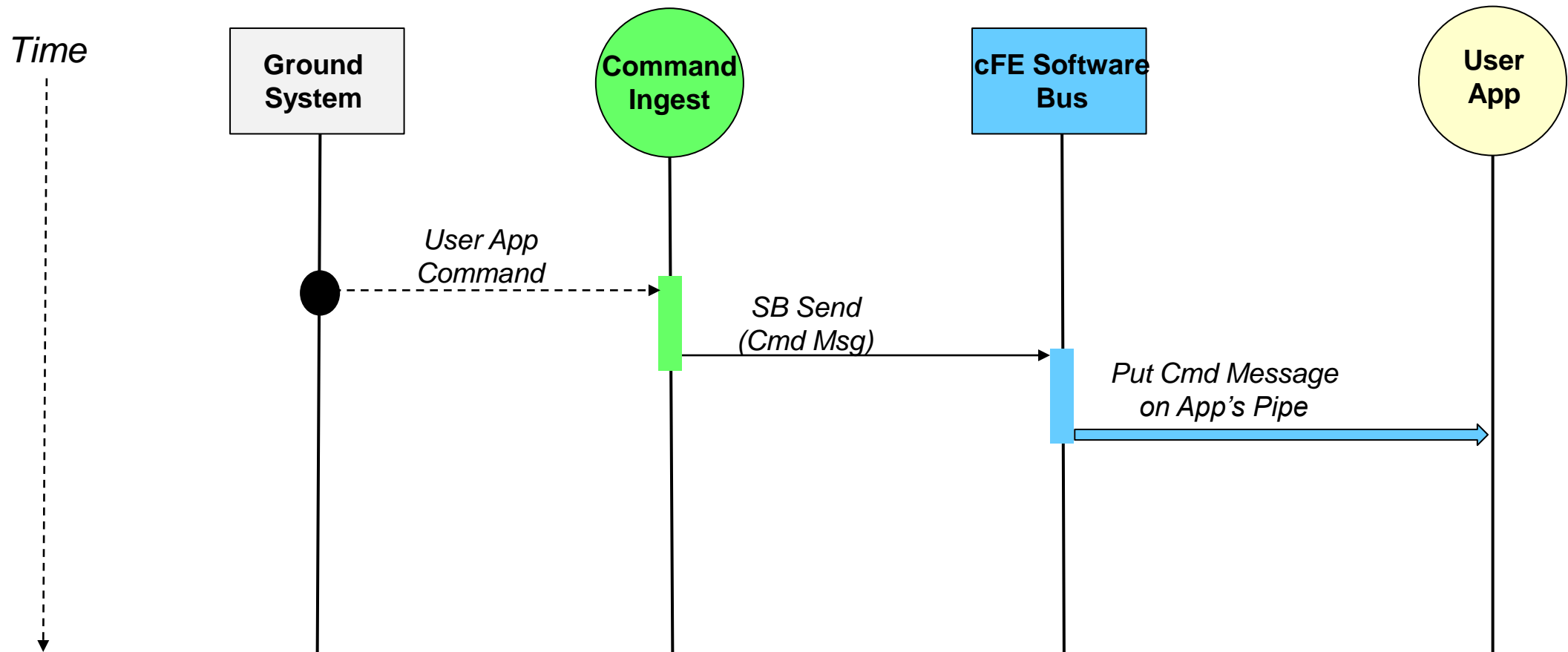


- **Command Ingest (CI) App**
 - Receives commands from an external source, typically the ground system, and sends them on the software bus
- **Telemetry Output (TO) App**
 - Receives telemetry packets from a the software bus and sends them to an external source, typically the ground system
 - Optional *Filter Table* that provides parameters to algorithms that select which messages should be output on the external communications link

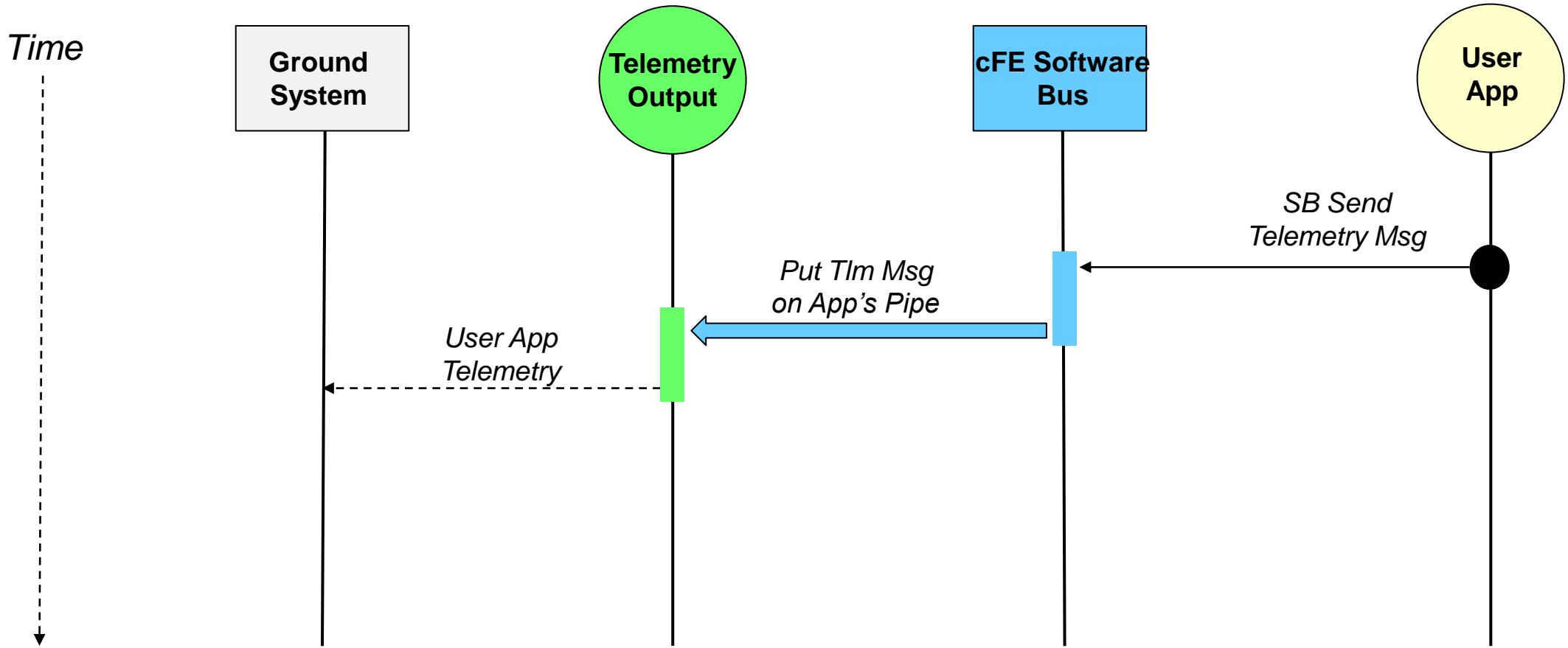
- **The ground and flight messages definitions must match**
 - Basecamp uses Electronic Data Sheets to define messages once and the EDS Toolchain creates ground and flight artifacts
 - In many situations, developers must manually implement separate ground and flight definitions and ensure that they match



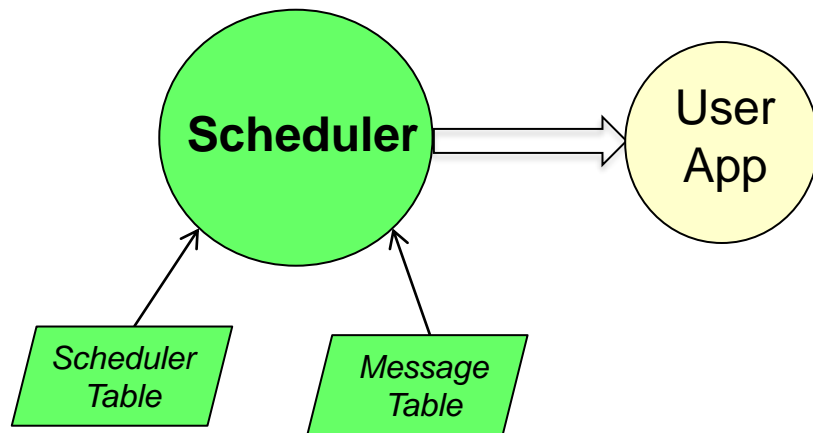
- **Mission external command and telemetry communications is more complicated for embedded systems**
 - An interface app is often used to manage the hardware interface and transferring messages between the Software Bus and the hardware interface
 - The Systems Engineering Document goes into more detail
- **The following versions of CI and TO are available as open source**
 - Basecamp versions use UDP and a JSON-defined filter table
 - cFS Bundle includes 'lab' versions that use UDP for the external comm
 - NASA's Johnson Space Center released versions that use a configurable I/O library for a different external comm links



● = Initial event



● = Initial event



- **Scheduler (SCH) App**
 - Synchronizes execution with clock’s 1Hz signal
 - Sends software bus messages defined in the *Message Table* at time intervals defined in the *Scheduler Table*

- **Application Control Flow Options**

- Pend indefinitely on a *SB Pipe* with subscriptions to messages from the Scheduler
 - This is a common way to synchronize the execution of most of the apps on a single processor
 - Many apps send periodic “Housekeeping” status packets in response to a “Housekeeping Request message from Scheduler
- Pend indefinitely on a message from another app
 - Often used when an application is part of a data processing pipeline
- Pend with a timeout
 - Used in situation with loose timing requirements and system synchronization is not required
 - The SB timeout mechanism uses the local oscillator so the wakeup time may drift relative to the 1Hz

Suspend execution until a message arrives on app's pipe

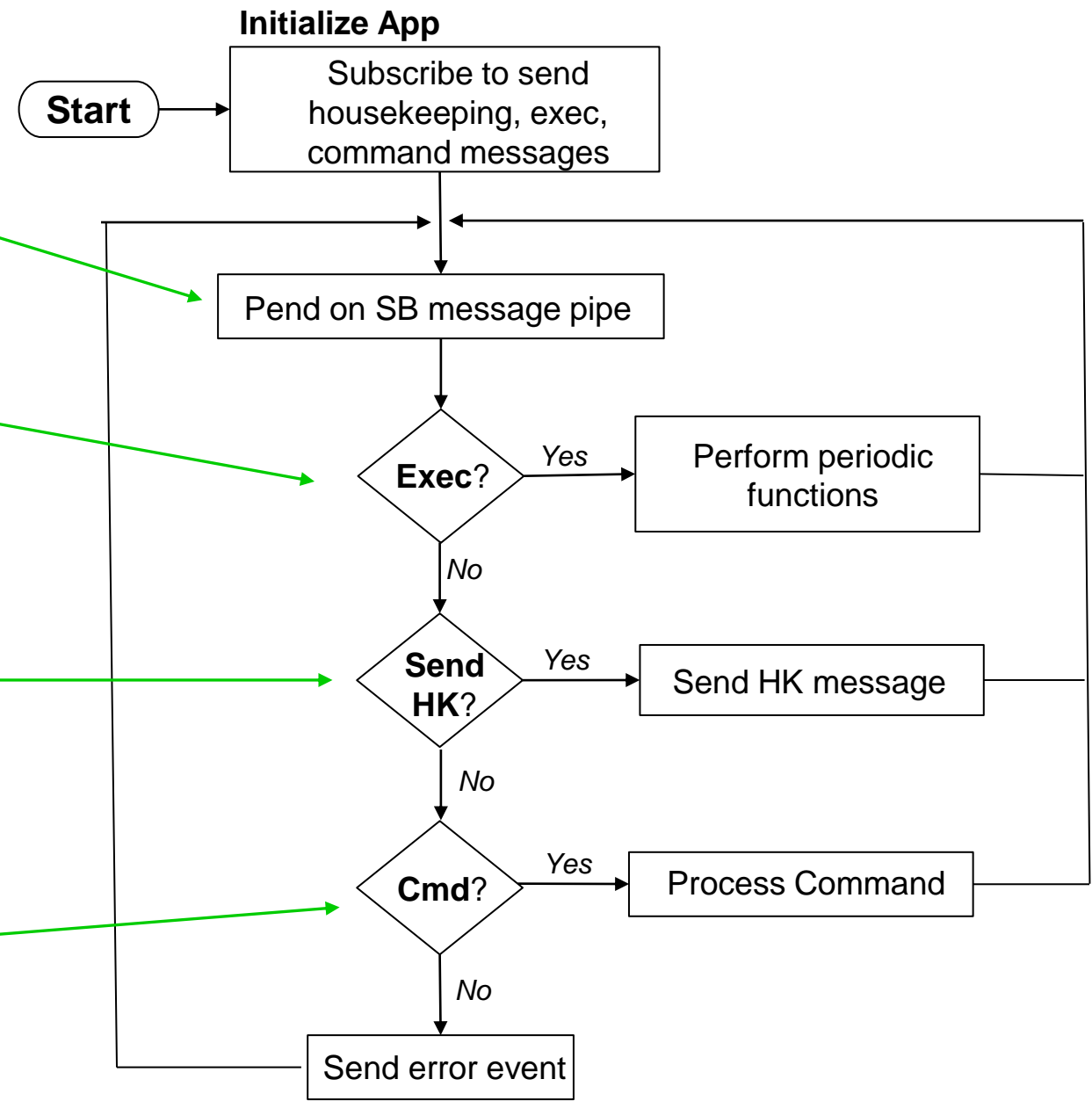
Periodic *execute* message from SCH app

Periodic *request housekeeping* message from SCH app

- Typically, on the order of seconds
- "Housekeeping cycle" convenient time to perform non-critical app functions

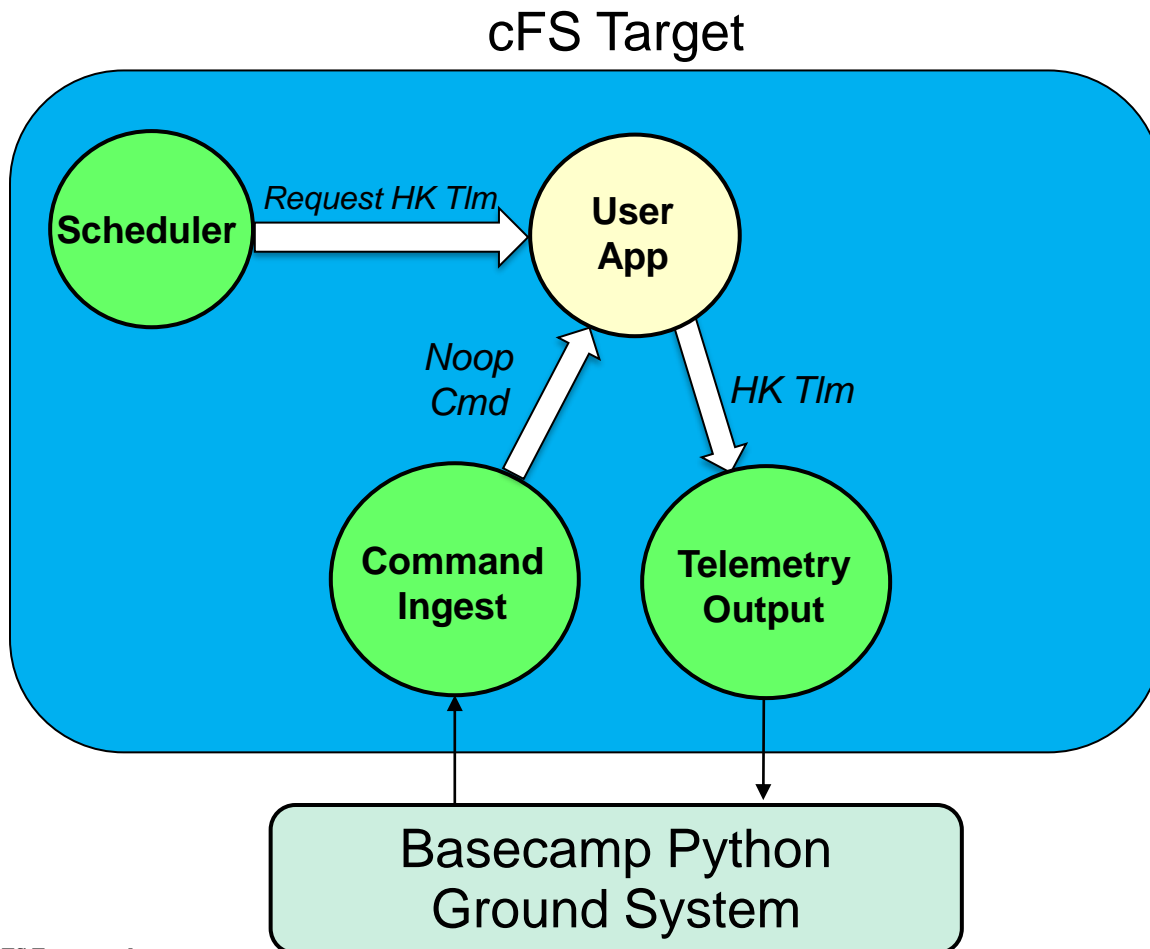
Process commands

- Commands can originate from ground or other onboard apps



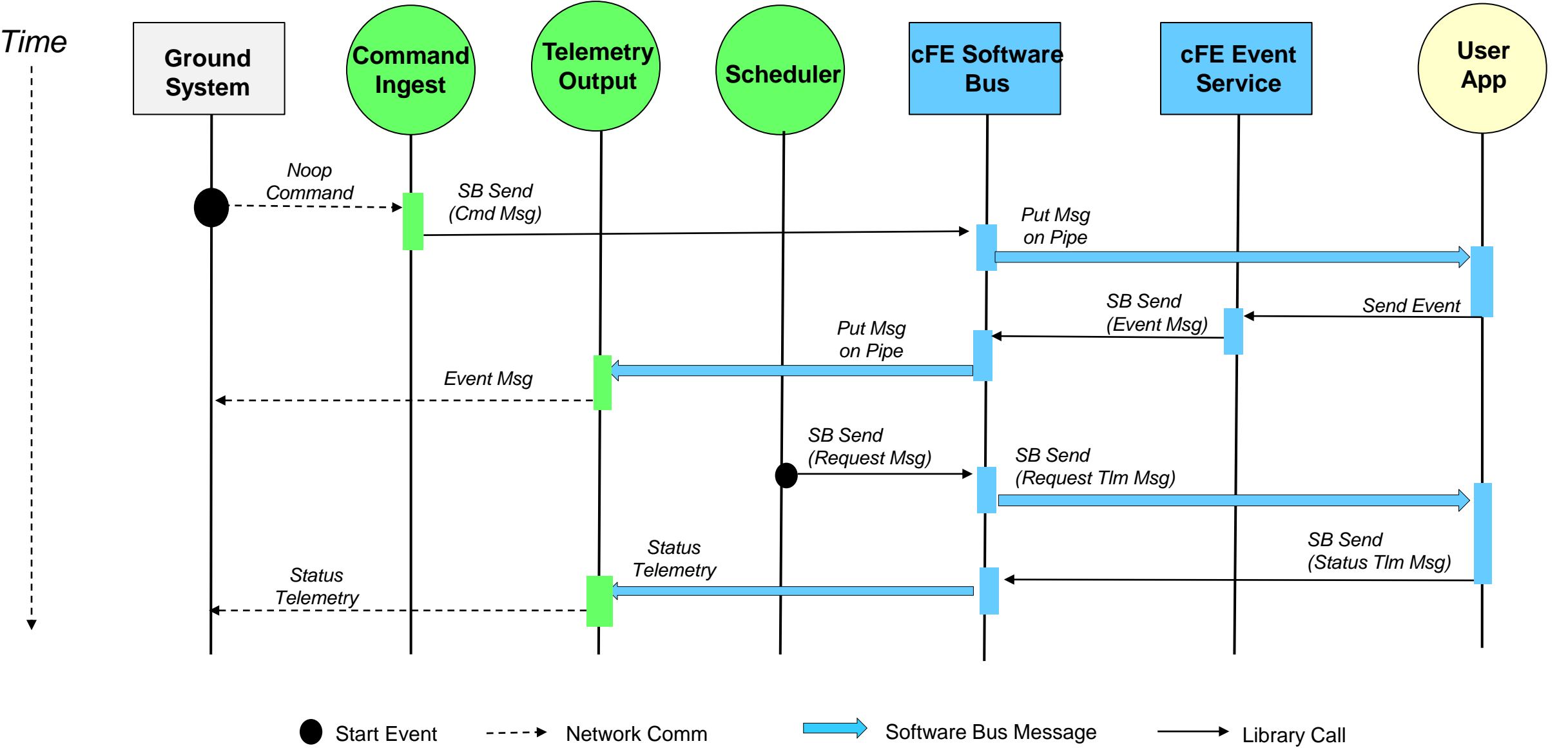
Application Runtime Environment Summary

- By convention every app contains a “No Operation (NOOP)” command
- Walking through the NOOP command execution flow is a good way to understand the runtime environment provided by three of operations service apps

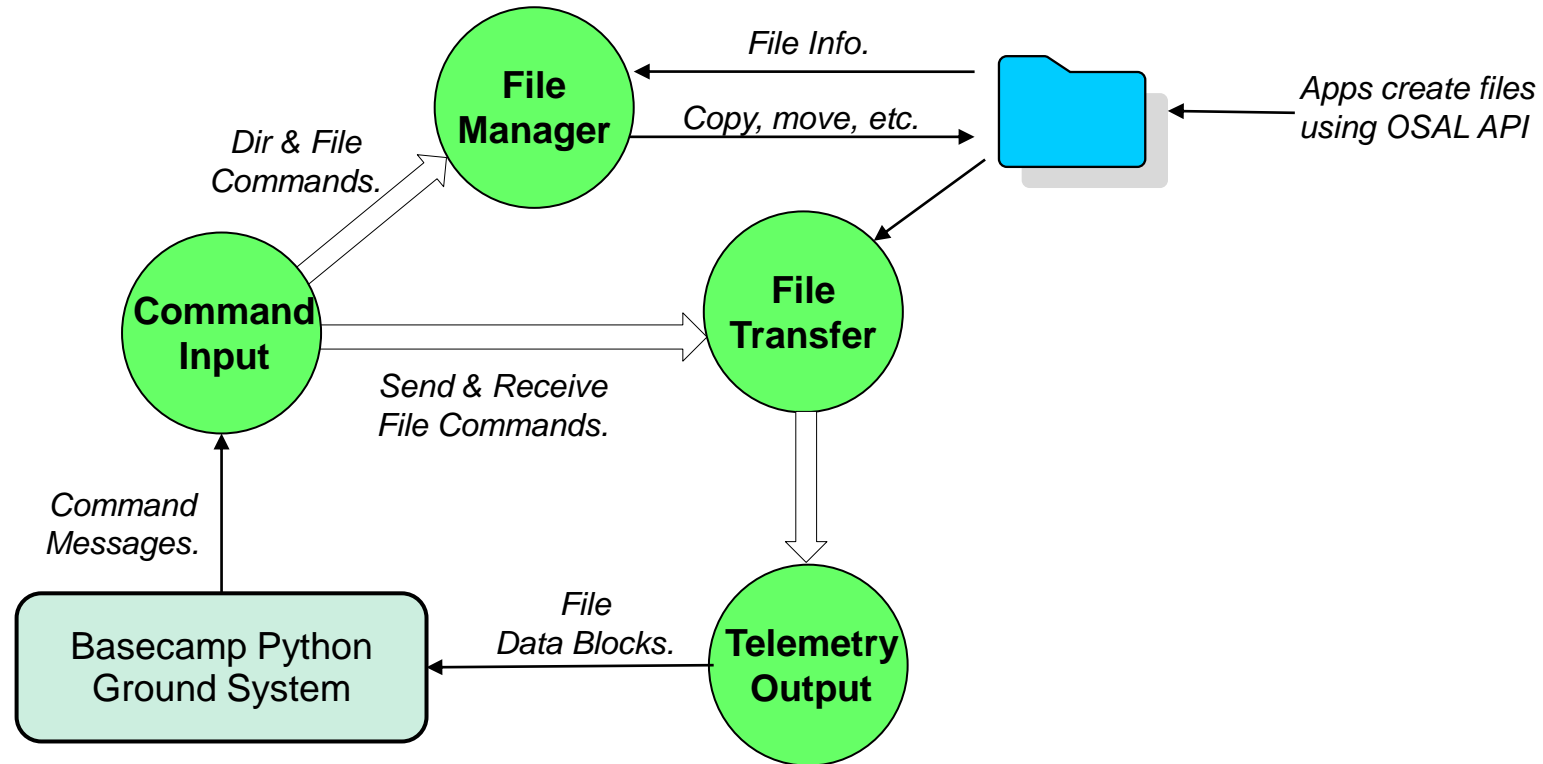


This sequence is illustrated on the next slide

1. When a user sends a NOOP command from the ground system an app responds with
 - An event message that contains the app's version number
 - Increments the command valid counter
2. The Scheduler app periodically sends a “Request Housekeeping Telemetry”
 - HK telemetry includes valid and invalid command counters

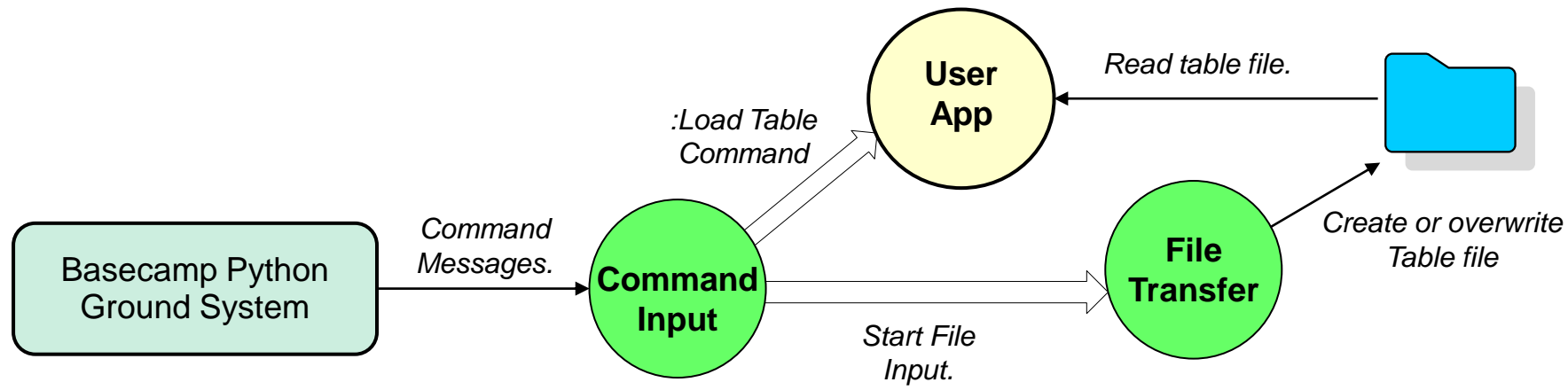


- **The cFS relies on a file system so Basecamp's Operations Service App Suite contains two apps that provide the required functionality**
- **File Manager provides a ground command/telemetry interface for managing onboard directories and files**
 - The NASA File Manager app design was refactored to use an object-based design and Basecamp's application framework
- **File Transfer transfers files between flight and ground using a custom file transfer protocol implemented in both the flight and ground systems**
 - The protocol is very similar to the Class 1 CCSDS File Delivery Protocol (CFDP)
 - The protocol messages are defined using EDS

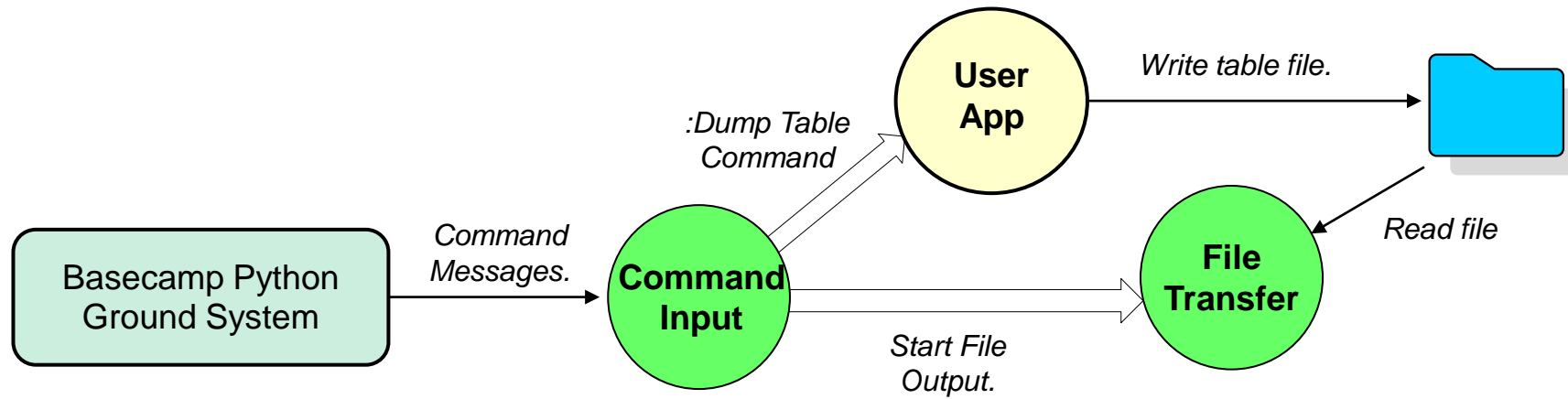


- **Apps create files using the OSAL API so the cFS Framework can manage files resource usage**
- **File Manager is used to manage directories and files**
 - Commands can originate from the ground or other onboard apps
- **File Transfer is used to transfer between the ground and flight**
 - Files are divided into data blocks that are transferred as CSSDS messages
 - The File Transfer to Telemetry Output message interface requires a mechanism to control the data rate

- **Basecamp apps do not use the cFS table service**
- **JSON files are used to define table parameters and values**
 - The cFS uses binary files
 - Onboard file management apps need to be present for either method
- **If a Basecamp app has a table then it provides table load and dump commands**
 - The Basecamp app framework provides table management services
- **All Basecamp apps have a JSON initialization file, but it is not a table**
- **The “Hello Table” code tutorial and Basecamp App Developer Guide describe how to create apps with tables**



1. **Use File Transfer to transfer the table file from the ground to an onboard directory**
 - The file can be located in any directory
2. **Send a Table Load command to the user app**
 - The command specifies the table filename and directory
 - The user app parses the JSON file using basecamp app framework utilities and optionally validates the contents



1. **Send a Table Dump command to the user app**
 - The command specifies the directory and table filename
2. **Use File Transfer to transfer the table file from an onboard directory to ground**

- **As previously described, messages are a key component of the cFS architecture**
 - They are used for transferring data and can be used to control an application's execution
- **The next few slides describe how Electronic Data Sheets are used to define messages**
- **The Deployment Model section describes how the EDS toolchain is used to build a target**

- The cFS uses the CCSDS

1. From the Tutorial dropdown list select “Basecamp Introduction” and do Lesson 2 “Basecamp Demo App”

1

Tutorials

Basecamp Introduction

Build and Run the cFS

cFS GitHub App Exchange

Create App Tool

Hello World

Hello Object

2

Tutorials

Basecamp Introduction

Build and Run the cFS

cFS GitHub App Exchange

Create App Tool

Hello World

Hello Child

Hello Object

Hello Table

Basecamp Introduction

Users to Basecamp features and

Complete

Basecamp Feature Overview

No

2-Basecamp Demo App

No

3-Basecamp Next Steps

No

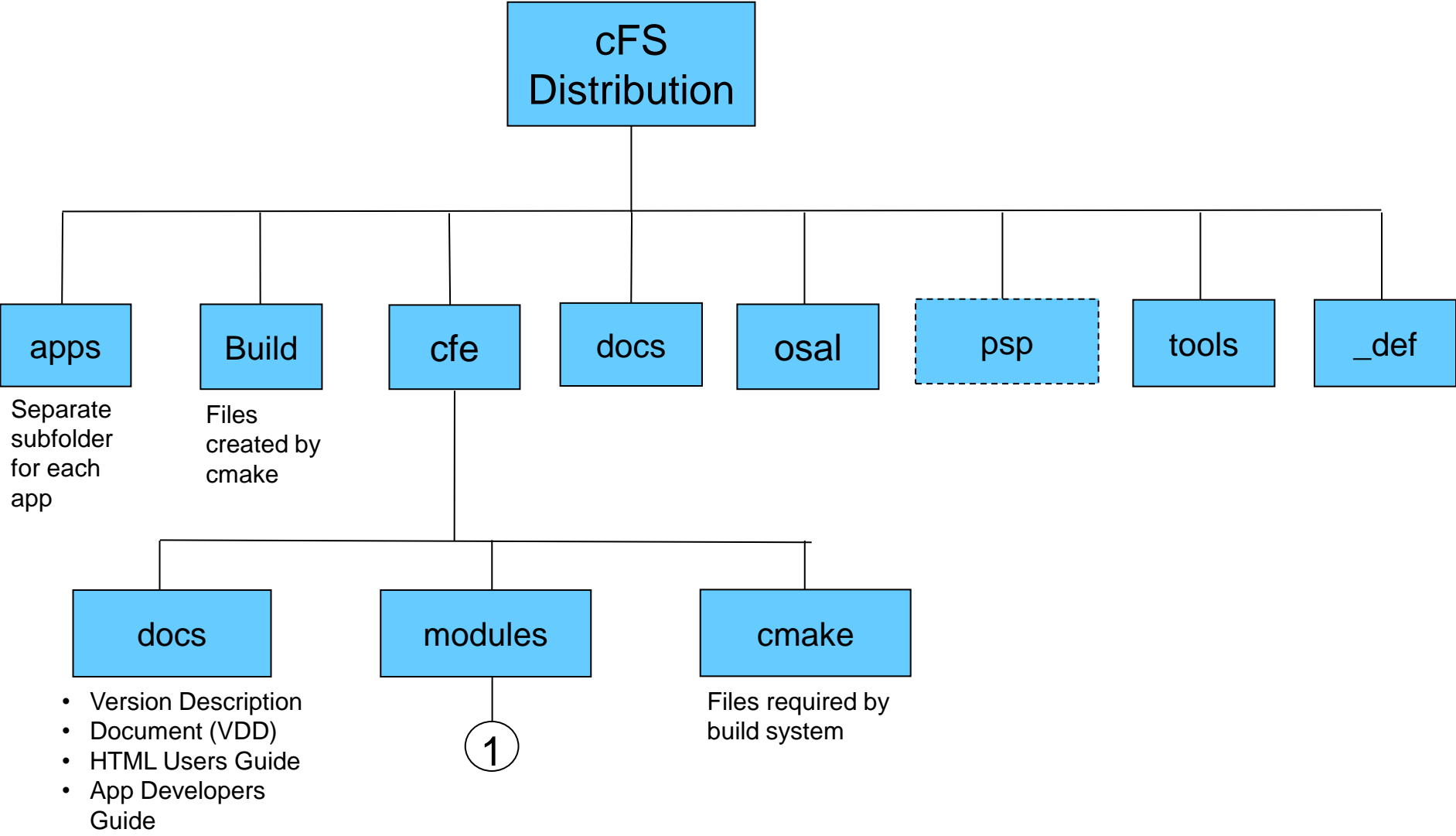
Start

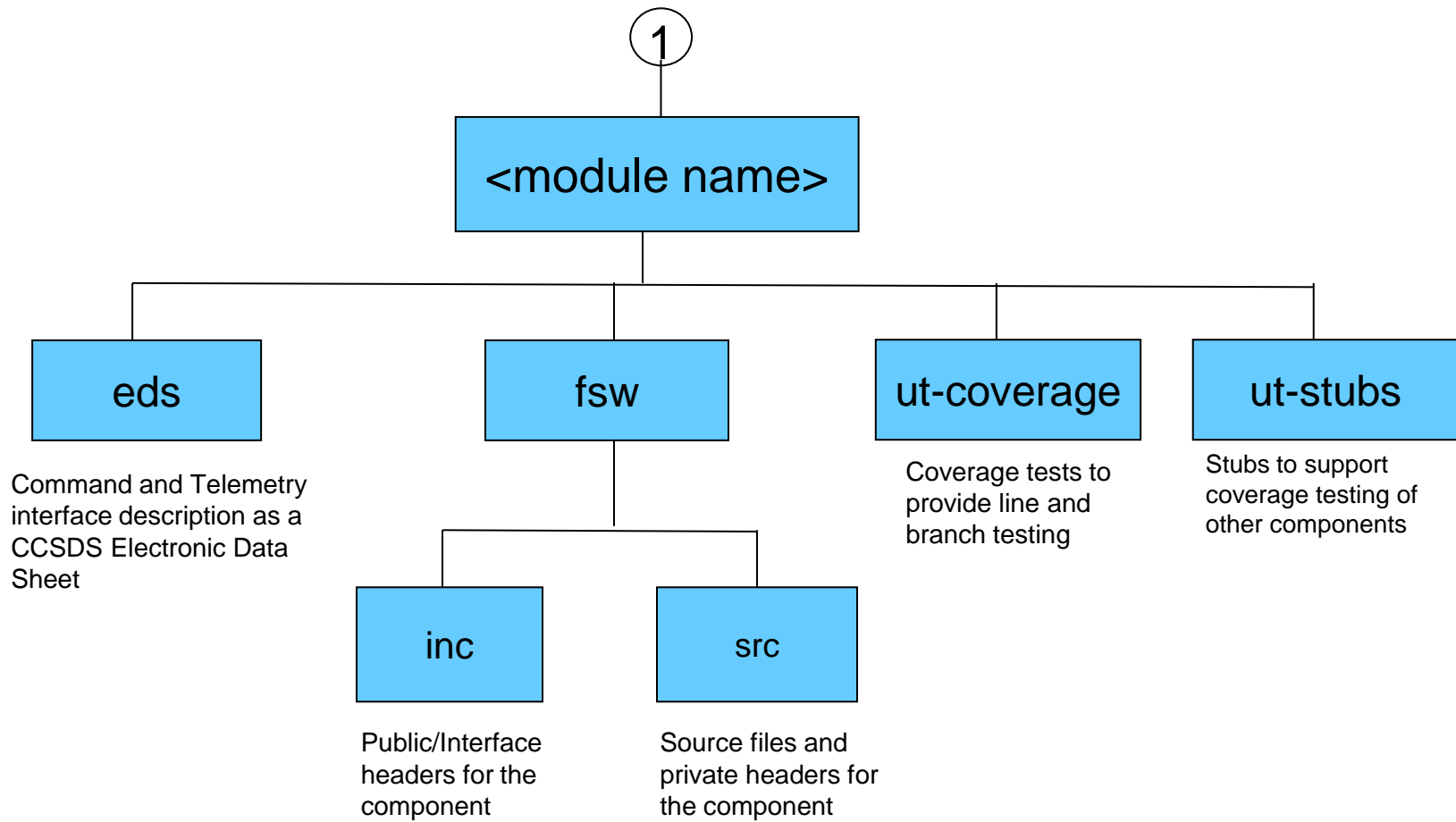
Reset

Exit

cFS Framework Deployment

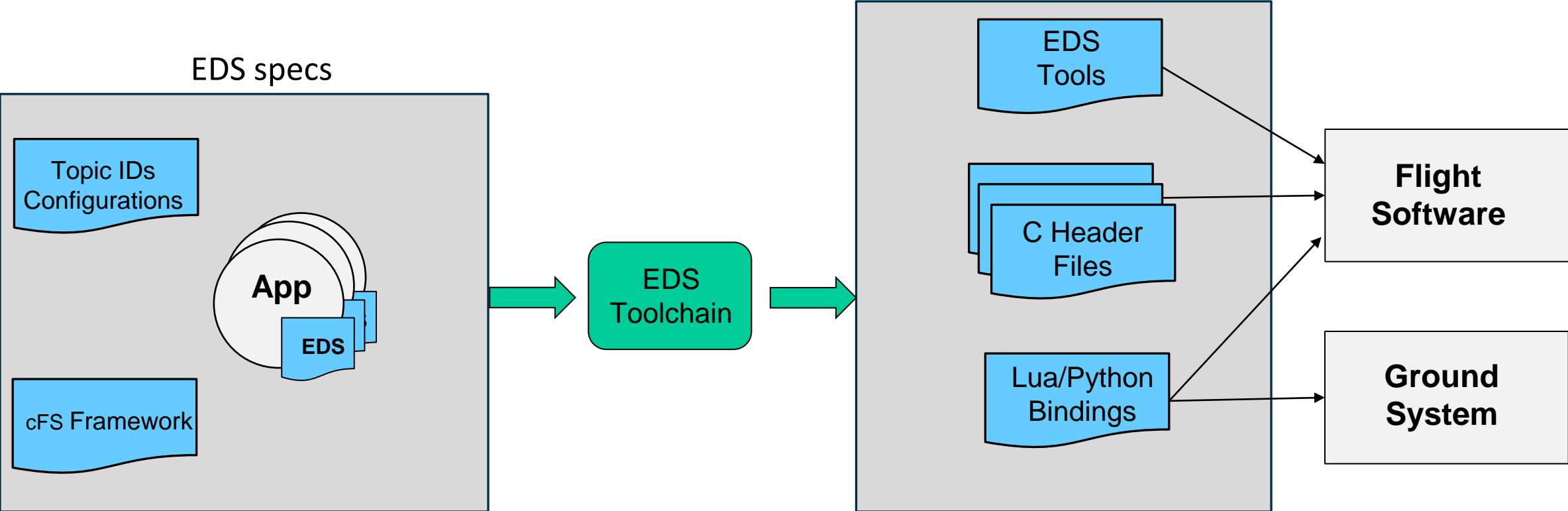
- **This section briefly introduces each cFE service's functionality**
 - OSK provides detailed material on each service with demos and self-guided tutorials
- **Section outline**
 1. TBD
- **cFS Framework is built as a single binary image**
 - Libraries and applications built as individual object files
 - A cFE startup script defines which lib/app object files are loaded





- **The Caelum release introduced the module structure**
- **cFE core components are organized as modules**
- **Modular structure allows advanced users to add, remove, or override entire core services as necessary to support their particular mission requirements**
- **cFE “out of the box” provides reference implementations that meet the needs of most missions**

Module	Purpose/Content
cfe_assert	A CFE-compatible library wrapping the basic UT assert library.
cfe_testcase	A CFE-compatible library implementing test cases for CFE core apps.
core_api	Contains the public interface definition of the complete CFE core - public API/headers only, no implementation.
core_private	Contains the inter-module interface definition of the CFE core - internal API/headers only, no implementation.
es	Implementation of the Executive Services (ES) core module.
evs	Implementation of the Event Services (EVS) core module.
fs	Implementation of the File Services (FS) core module.
msg	Implementation of the Message (MSG) core module.
resourceid	Implementation of the Resource ID core module.
sb	Implementation of the Software Bus (SB) core module.
sbr	Implementation of the Software Bus (SB) Routing module.
tbl	Implementation of the Table Services (TBL) core module.
time	Implementation of the Time Services (TIME) core module.



Single EDS definitions propagate to the ground and flight systems

- **Mission configuration parameters – used for ALL processors in a mission (e.g. time epoch, maximum message size, etc.)**
 - Default contained in:
 - \cfe\fs\mission_inc\cfe_mission_cfg.h
 - \apps\xx\fs\mission_inc\xx_mission_cfg.h. xx_perfids.h
- **Platform Configuration parameters – used for the specific processor (e.g. time client/server config, max number of applications, max number of tables, etc.)**
 - Defaults contained in:
 - \cfe\fs\platform_inc\cpuX\cfe_platform_cfg.h, cfe_msgids_cfg.h
 - \apps\xx\fs\platform_inc\xx_platform_cfg.h, xx_msgids.h
 - \osal\build\inc\osconfig.h
- **Just because something is configurable doesn't mean you want to change it**
 - E.g. CFE_EVS_MAX_MESSAGE_LENGTH

- **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
- **Task priorities are not configuration parameters but must be managed from a processor perspective**
- **Note cFE strings are case sensitive**

File	Purpose	Scope	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	

1. From the Tutorial dropdown list select “Build and Run the cFS” and do all of the lessons

1

Tutorials

Basecamp Introduction

Build and Run the cFS

cFS GitHub App Exchange

Create App Tool

Hello World

Hello Object

Build and Run the cFS

Objectives

Describes how to build the default Basecamp cFS target, how to run the target, and the next step options for adding apps.

Lesson	Complete
1-cFS Build Environment	No
2-Basecamp App Specs	No
3-Running the cFS	No

Start

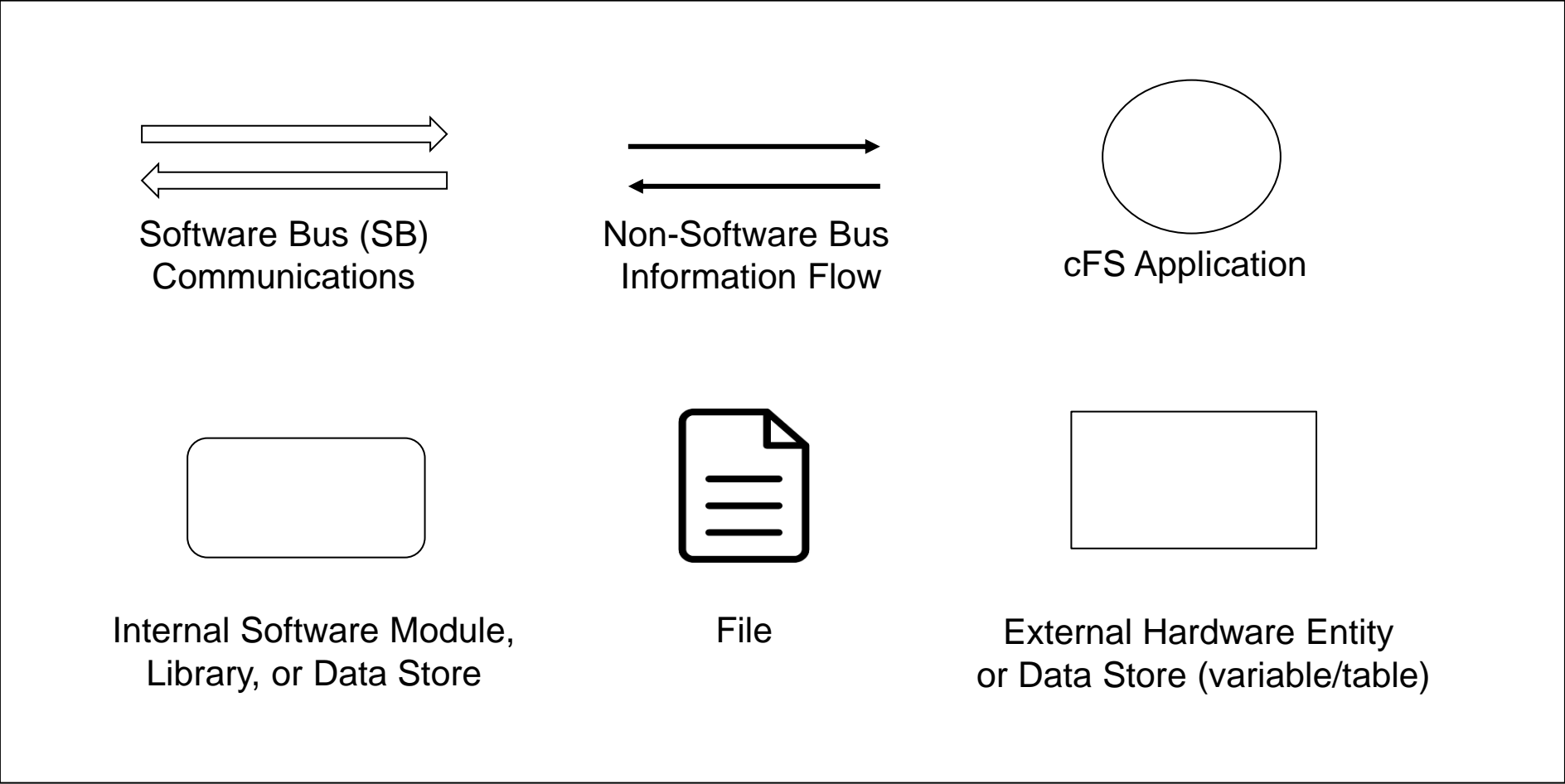
Reset

Exit

2

Appendix A

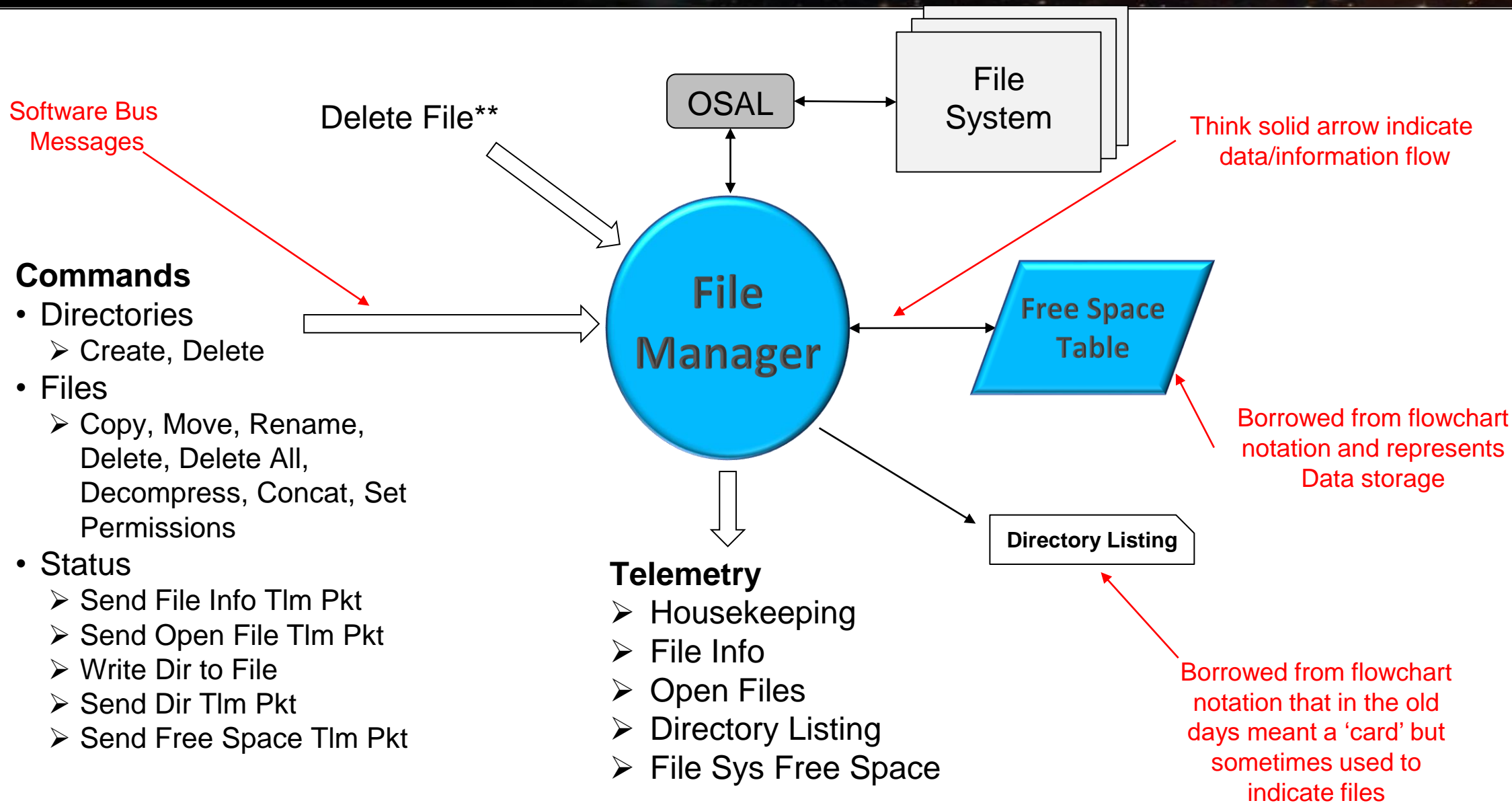
Architecture Design Notation



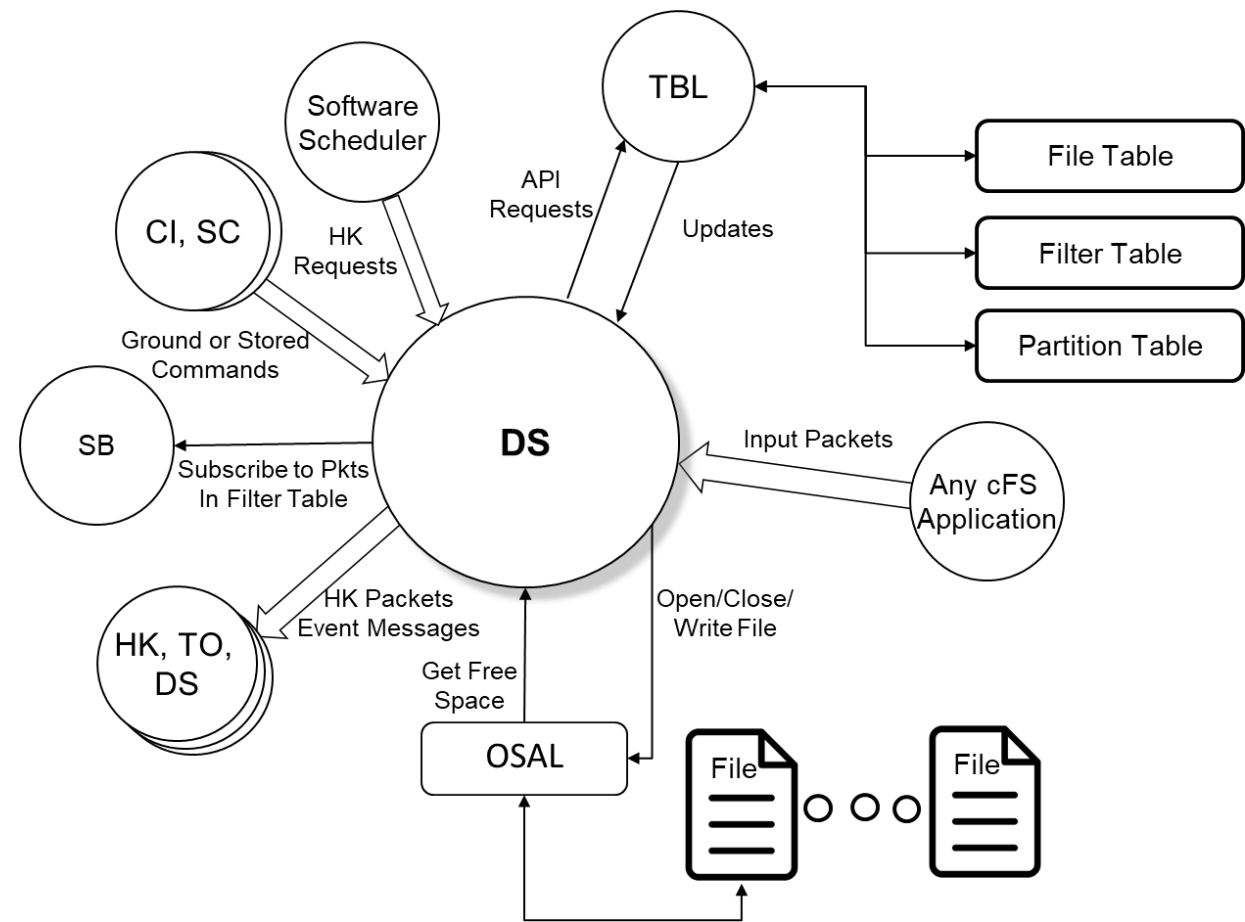
- Common data flows such as command inputs to an app and telemetry outputs from an app are often omitted from context diagrams unless they are important to the situation

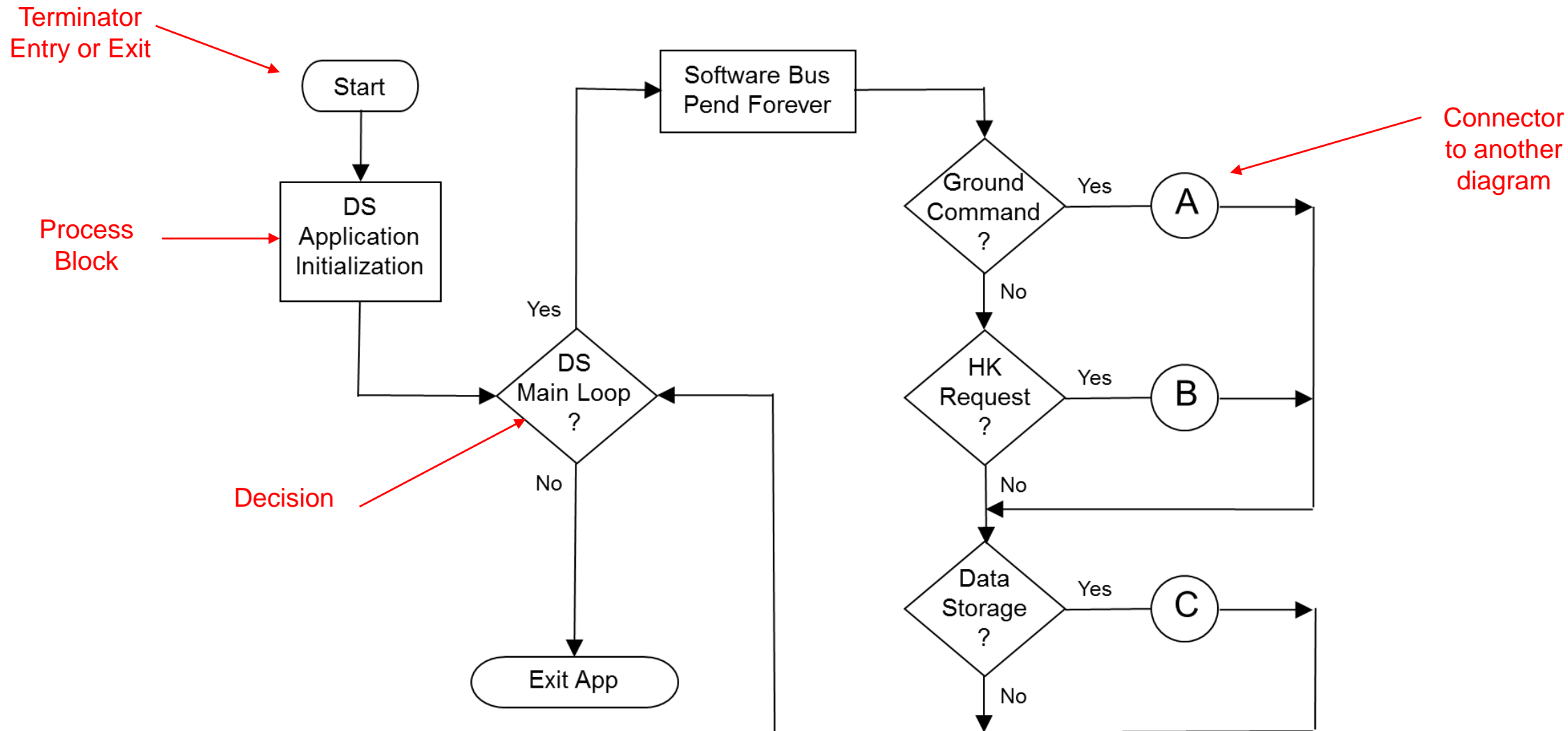
The following general outline is used in each of the cFE service documentation slides

- **Describe each service's main features from different perspectives**
 - System functions and operations
 - Feature Overview
 - Initialization and processor reset behavior
 - Onboard state retrieval
 - System integrator and developer
 - Configuration parameter highlights
 - Common practices
- **Student exercises are provided in a separate package**
 - Allows these slides to be maintained independent of the training platform and the training exercises can evolve independent of these slides



This is a more complete context diagram which is technically correct, but it can obscure the application-specific information that is most important. For example, HK request from the scheduler and outputting HK packet & event messages on the software bus are common design practice that may be omitted if people are comfortable with some assumptions. The important part of the diagram is showing interface boundaries to understand where control and data flow. Too much information is harder to maintain.





Often not show but if audience is unfamiliar with sequence diagram it can be helpful

Time

Applications, libraries and hardware devices shown across the top

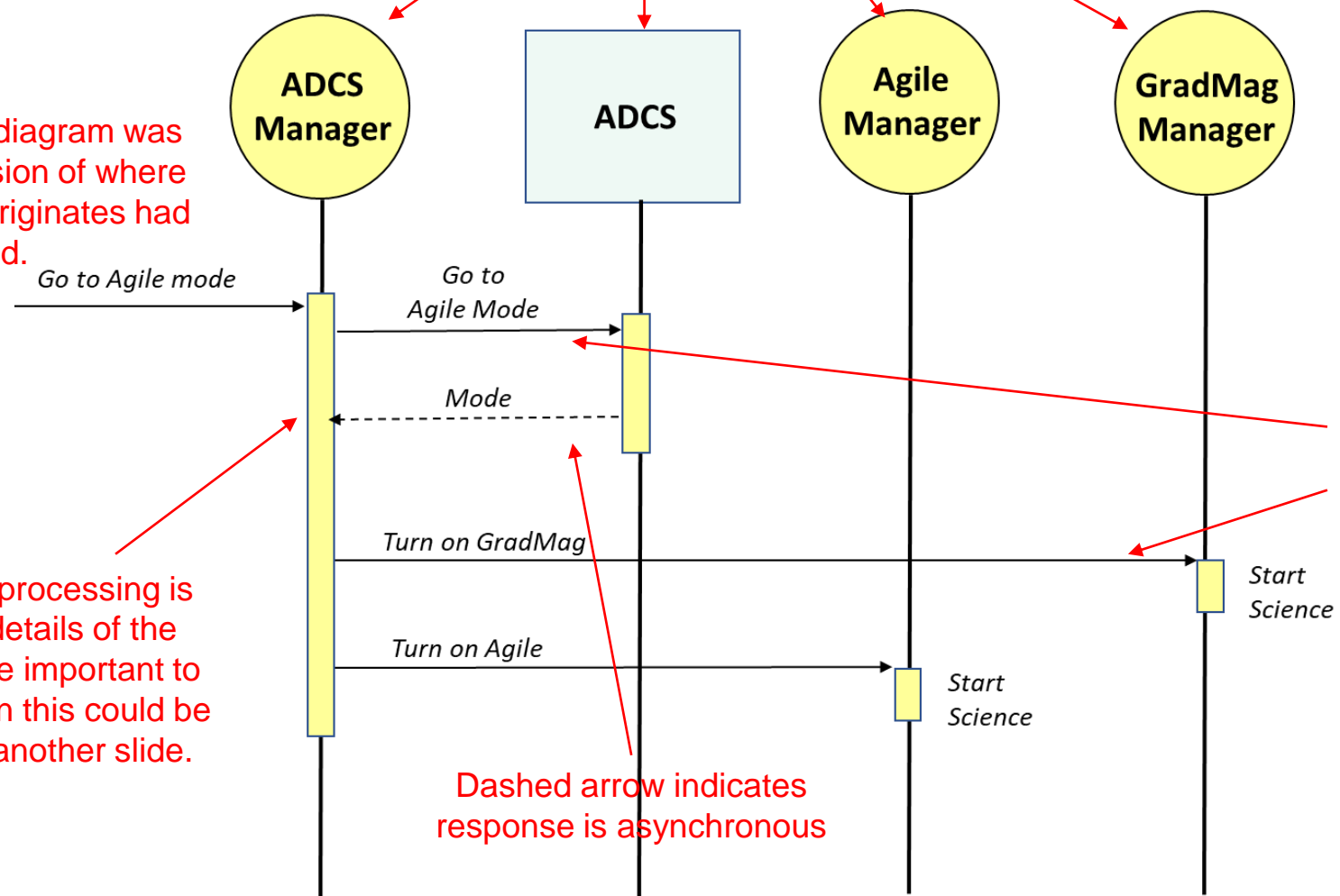
Level of detail shown is based on the goal of what is trying to be communicated. This diagram is showing a mode transition sequence, so it is not important to show that an ADCS request may go through a library or device driver to command the ADCS device.

At the time this diagram was written the decision of where this command originates had not been decided.

The thin line represents communication and how the communication occurs depends upon the item at the top of the subsystem line. App-to-app communication occurs via software bus messages. An app-to-device communication typically occurs through a device driver.

Blocks show processing is required. If details of the processing are important to document, then this could be contained in another slide.

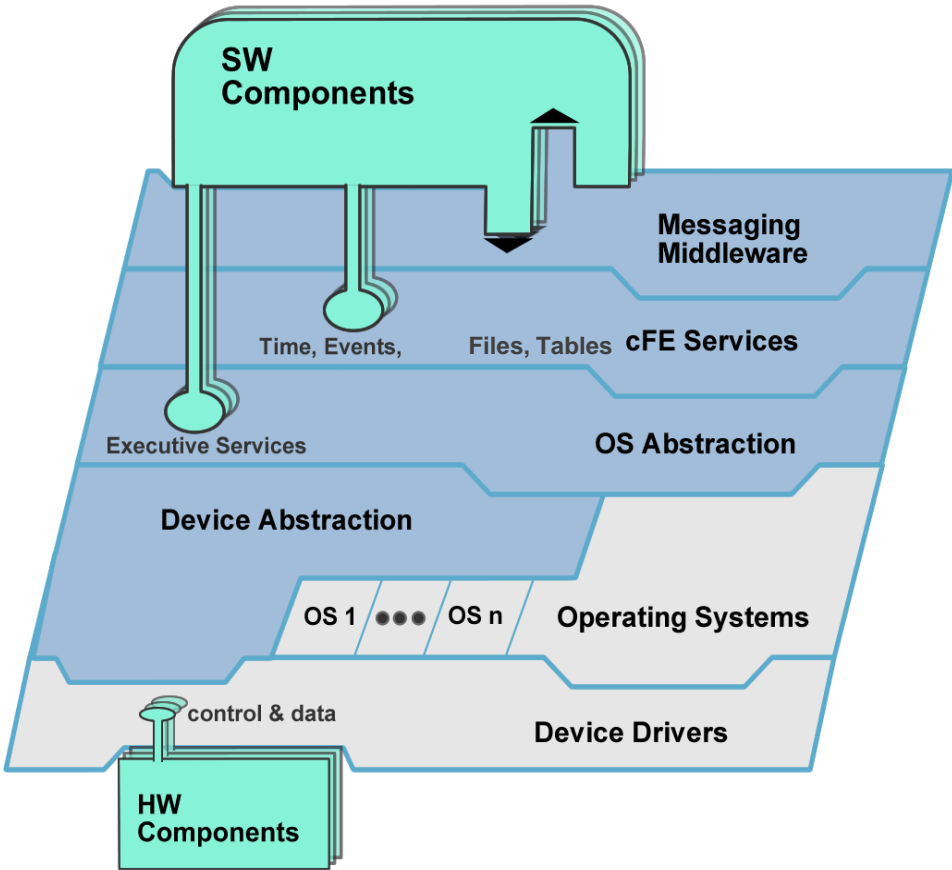
Dashed arrow indicates response is asynchronous



Appendix B

Supplemental Architectural Material

- 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.

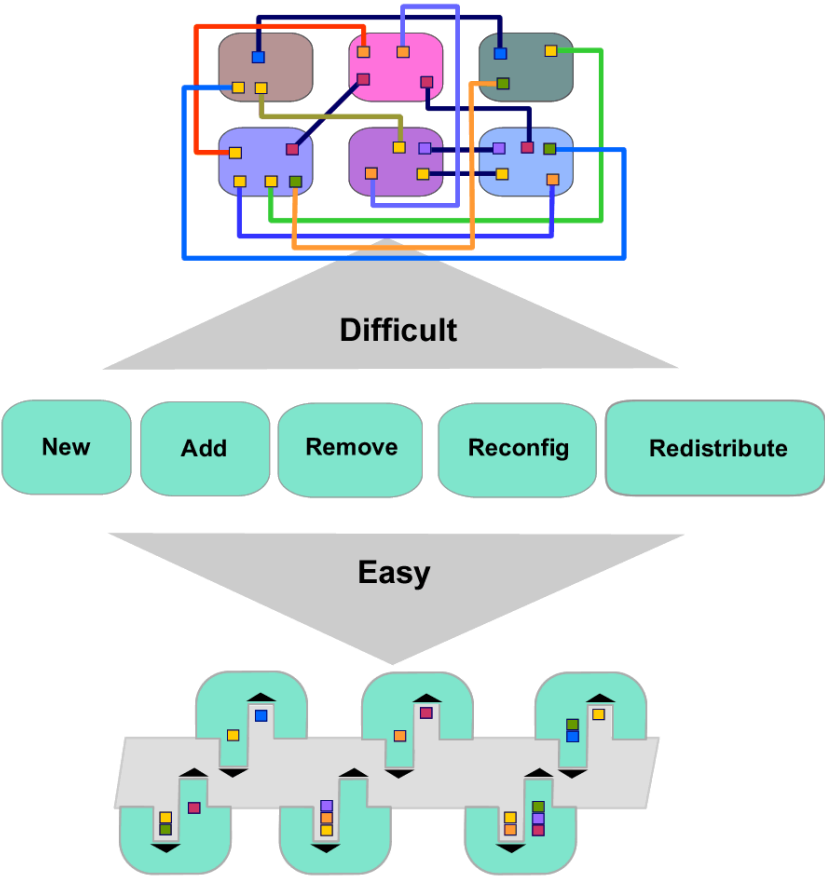


Plug and Play

- 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 (test apps, simulators)



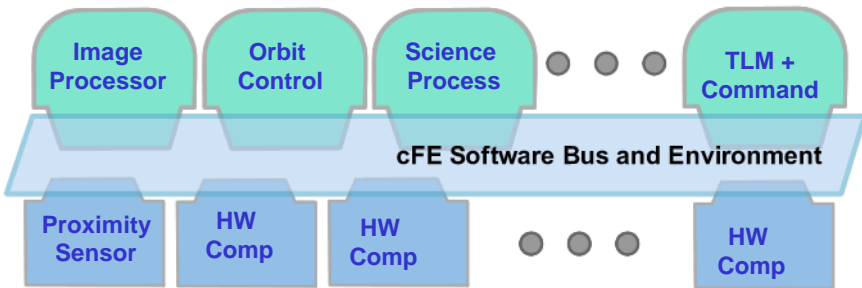
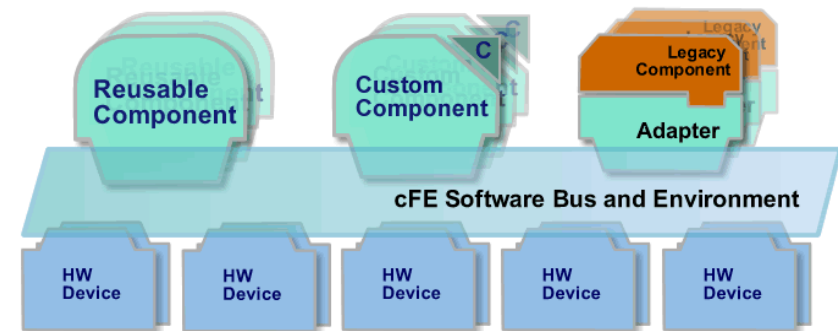
This powerful paradigm allows SW components to be switched in and out at runtime, without rebooting or rebuilding the system SW.

Reusable Components

- **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:

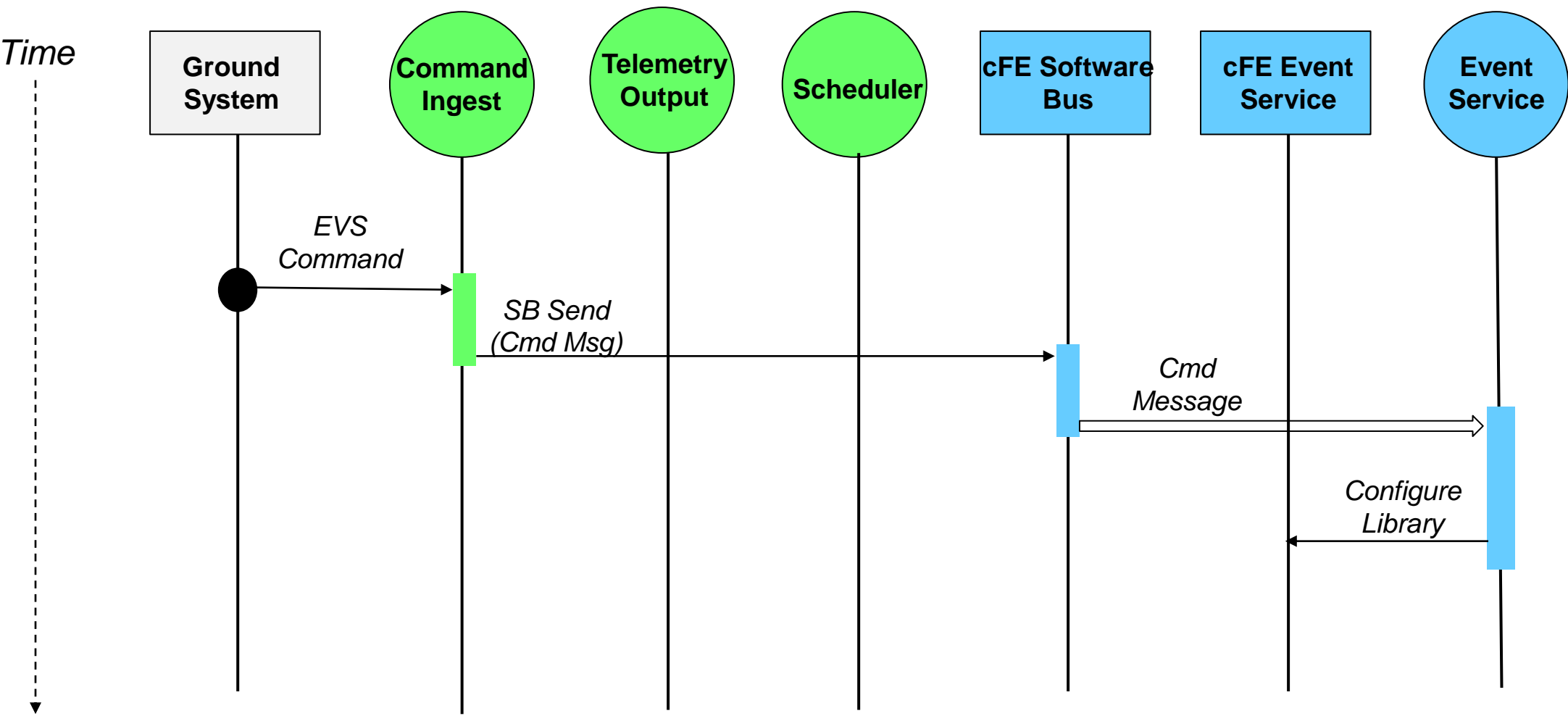
- **Reuse of tested, certified components supplies savings in each phase of the software development cycle**
- **Reduces risk**
- **Teams focus on the custom aspects of their project and don't "reinvent the wheel."**



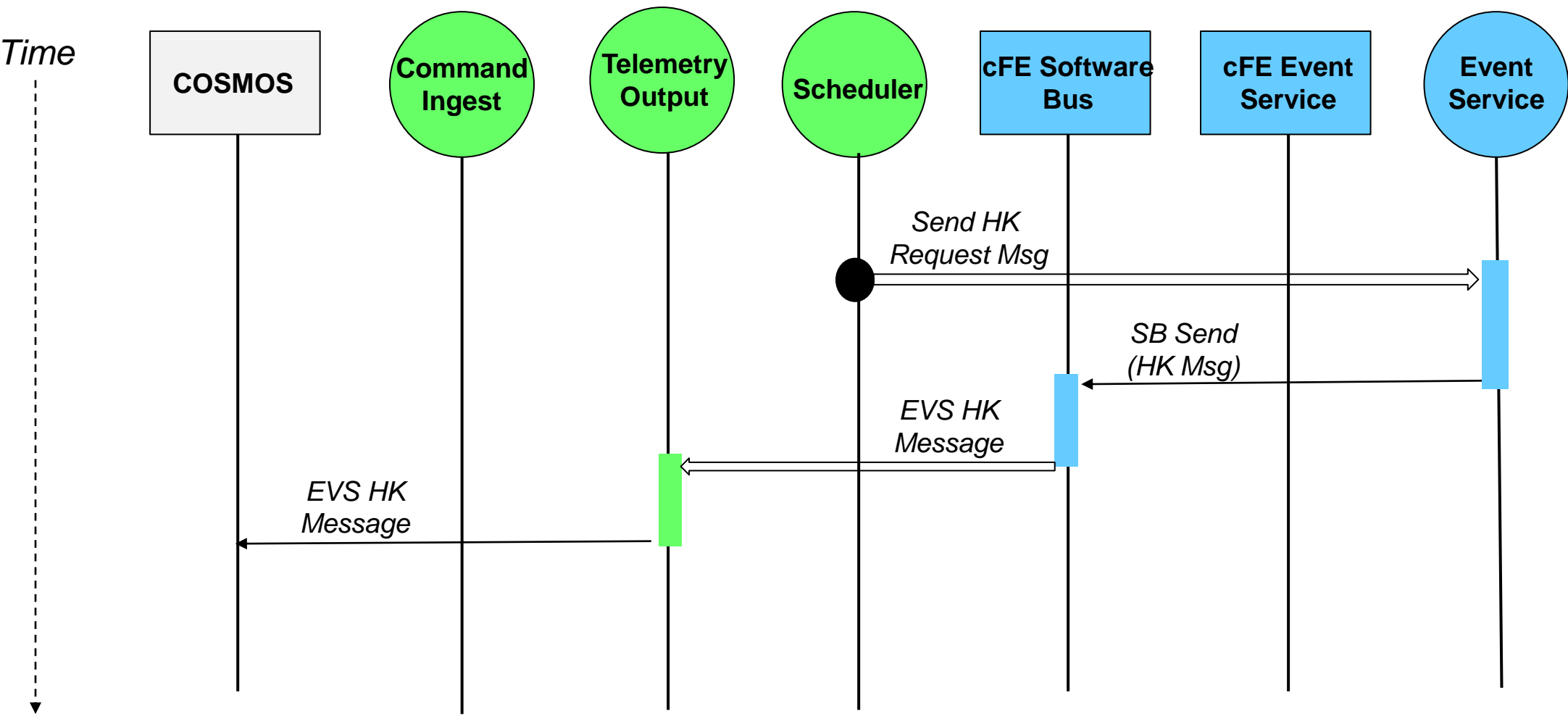
cFE/ App	Logical Lines of Code (non-table)	Config. Parameters	EEPROM (bytes)
cFE 6.4.0	12,930	General: 17 Executive Service: 46 Event Service: 5 Software Bus: 29 Table Service: 10 Time Service: 32	341,561
CFDP	8,559	33	85,812
Checksum	2,873	15	35,242
Data Storage	2,429	27	40,523
File Manager	1,853	22	16,272
Health & Safety	1,531	45	15071
House-Keeping	575	8	8,059
Limit Checker	2,074	13	31,026
Memory Dwell	1,035	8	8,617
Memory Manager	1,958	25	15,840
Scheduler	1,164	19	35,809
Stored Command (124 command sequences)	2,314	26	104,960

- **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
 - Addition of PSP changed build infrastructure midstream
- **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



● = Initial event



● = Initial event

Appendix C

Supplemental Application Material