National Aeronautics and Space Administration

Core Flight System (cFS) Training

Core Flight Executive (cFE) Services

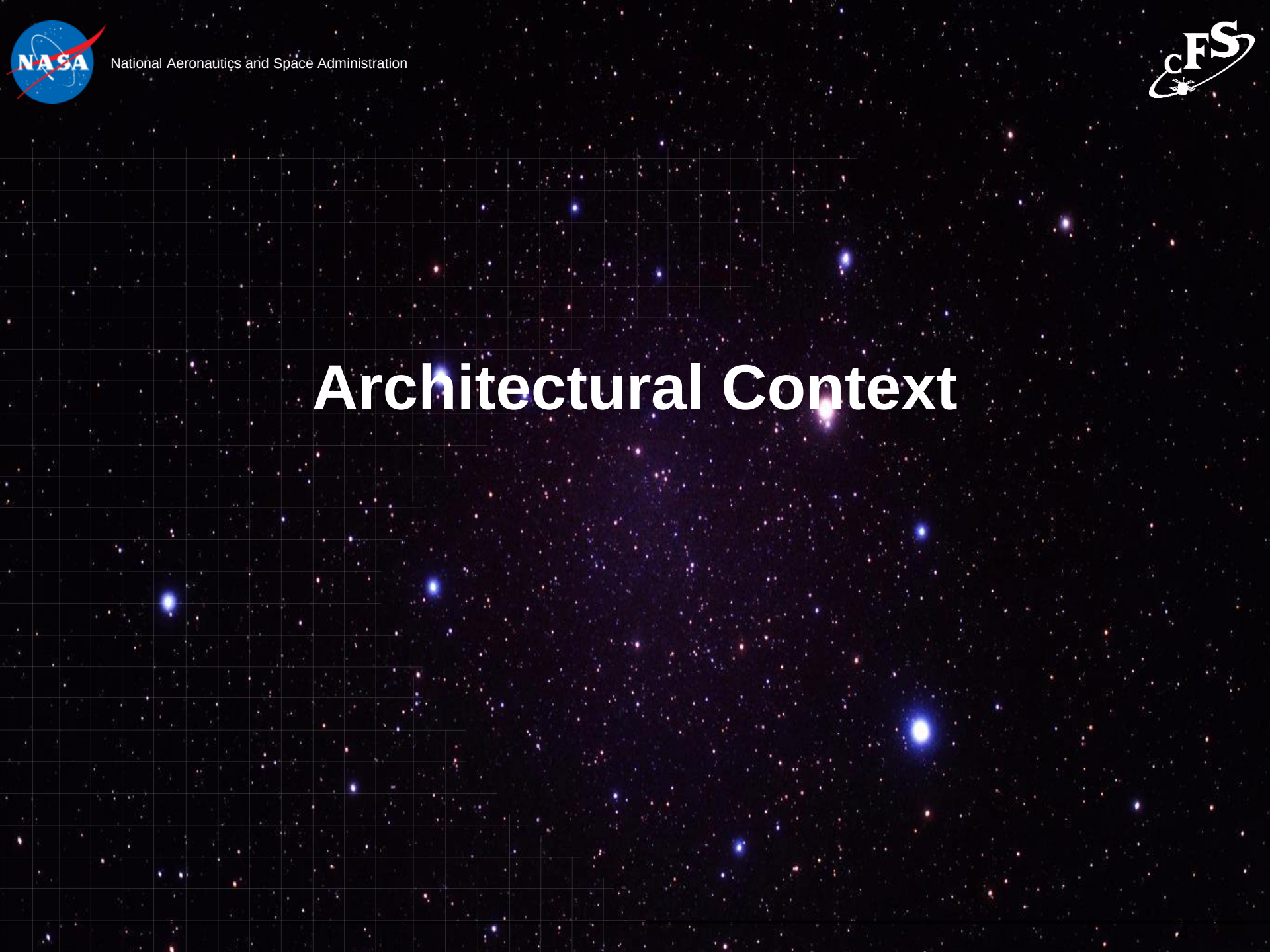
August 2019

Objectives and Intended Audience

- **Describe the core Flight Executive (cFE) services from a functional perspective**
- **Perform exercises using the ground command interface**
- **Intended Audience**
  - Spacecraft technical managers, systems engineers, discipline engineers, and software engineers

Agenda

1. cFE Architectural Context
2. cFE Services


National Aeronautics and Space Administration
NASA
cFS
Architectural Context

What are the cFE Services?

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

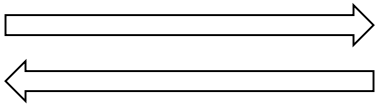
Software Bus (SB) Services

- Provide an application publish/subscribe messaging service

Table Services (TBL)

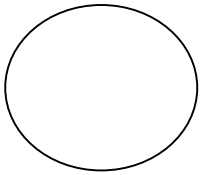
- Manage application table images

Diagram Notation



Software Bus (SB) Communications

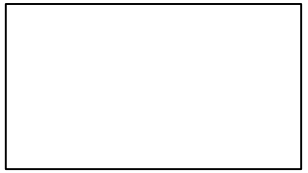
Non-Software Bus Information Flow



cFS Application

Internal Software Module, Library, or Data Store

File

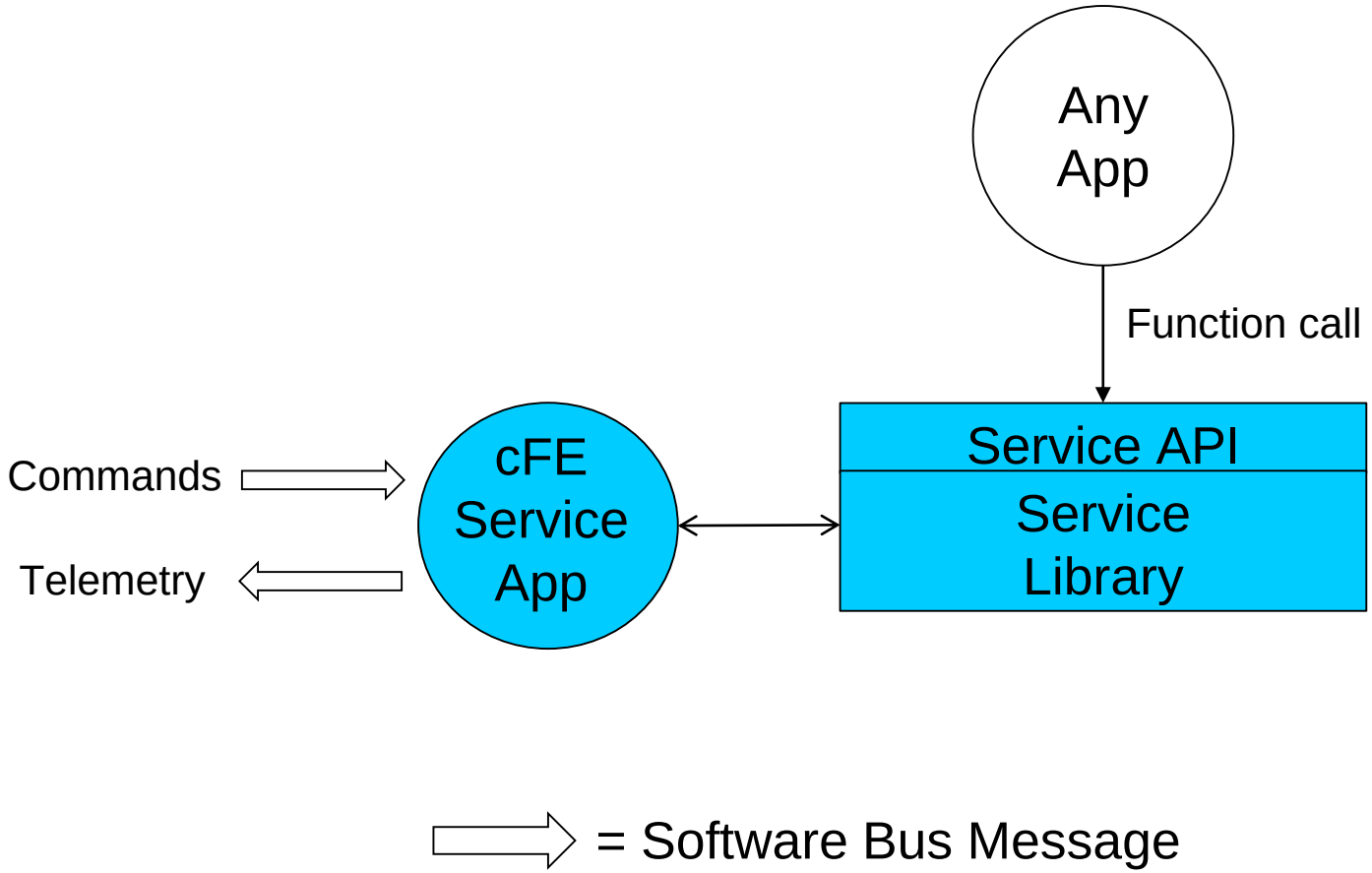


External Hardware Entity or Data Store (variable/table)

- 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 particular situation

Common cFE Service Design

- **Each cFE service has a**
  - A library that is used by applications
  - An application that provides a ground interface for operators to use to manage the service



Application Runtime Environment

- **cFE Services provide an Application Runtime Environment**
- **Applications are an architectural component that owns cFE and operating system resources**
- **The cFE service API provides a functional interface to use the services**
  - Very stable. No functional change since 2008
- **Resources are acquired during application initialization and released when it terminates**
  - Helps achieve the architectural goal for a loosely coupled system that is scalable, interoperable, testable (each app is unit tested), and maintainable
- **Obtaining information beyond the housekeeping packet**
  - Commands to send one time telemetry packets
  - Commands to write onboard service configuration data to files

Application-Centric Architecture

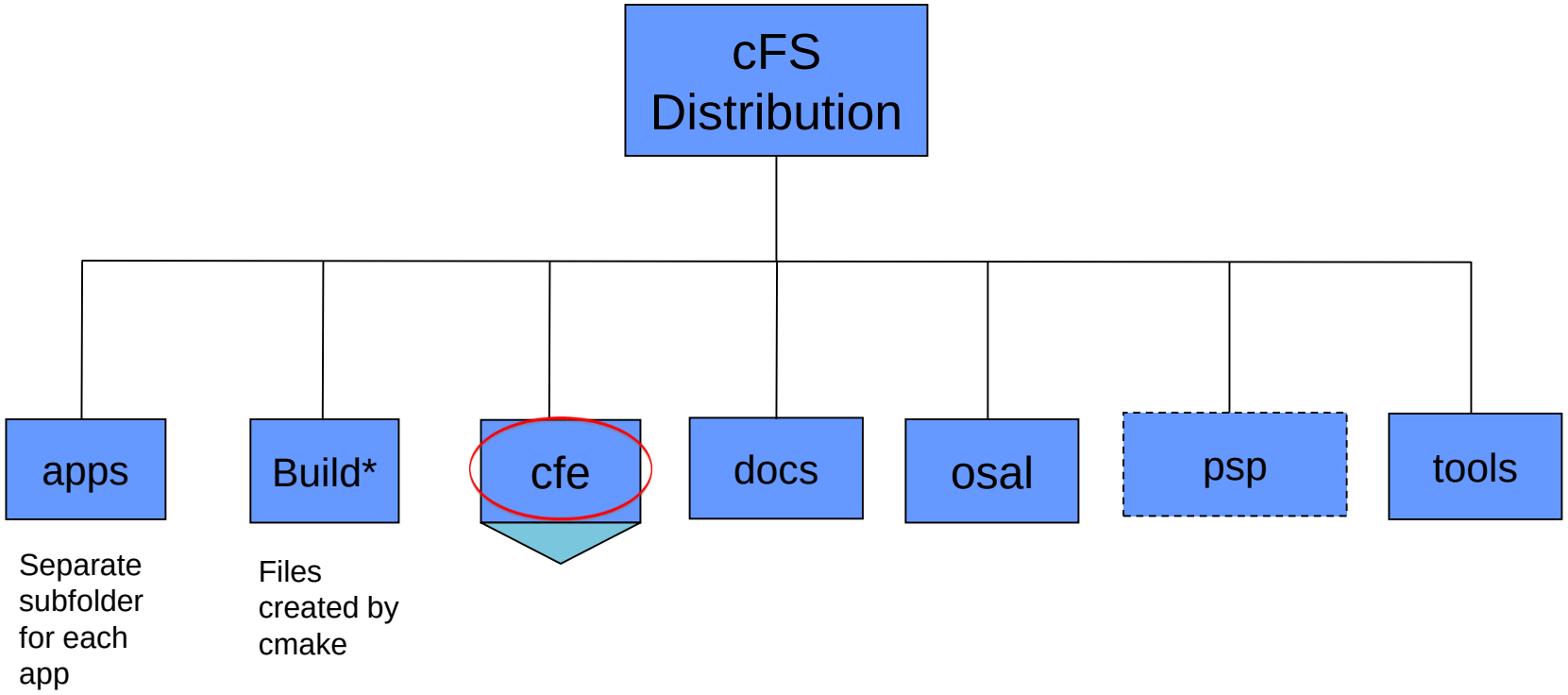
- **Applications are an architectural component that owns cFE and operating system resources**
- **Resources are acquired during initialization and released when an application terminates**
  - Helps achieve the architectural goal for a loosely coupled system that is scalable, interoperable, testable (each app is unit tested), and maintainable
- **Concurrent execution model**
  - Each app has its own execution thread and apps can spawn child tasks
- **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
  - Phone apps need to be rewritten for each platform
  - Framework provides seamless application transition from technology efforts to flight projects
- **Reload apps during operations without rebooting**

cFE Libraries

What is a library?

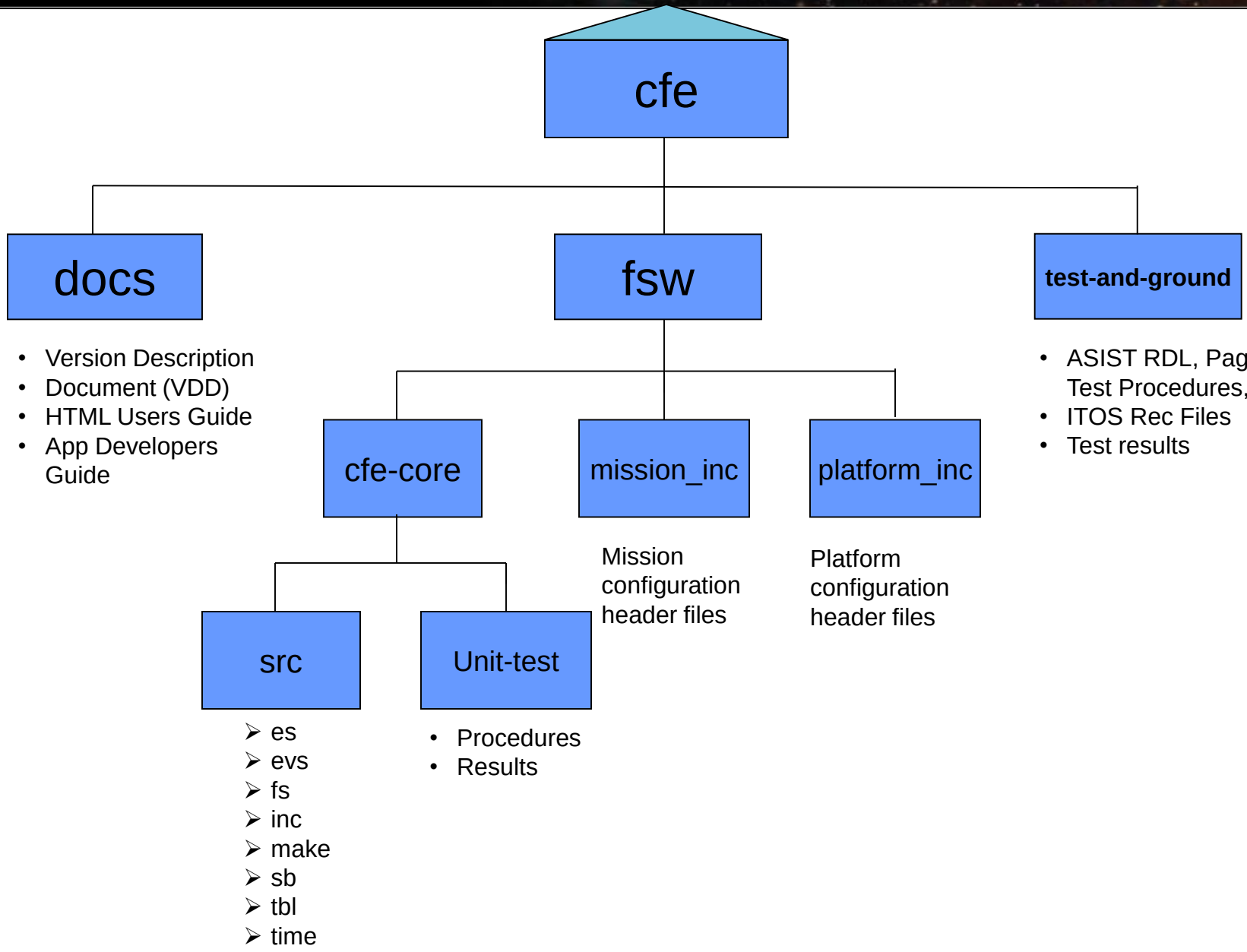
- Libraries can't use application services that require registration
- Static versus dynamic libraries

cFS Mission Directory Structure



* Files created by cmake

cFE Directory Structure



Configuration Parameter Scope

- **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\xx\fsw\mission_inc\xx_mission_cfg.h. xx_perfids.h

- **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_msgids_cfg.h
    - \apps\xx\fsw\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

Unique Identifier Configuration Parameters

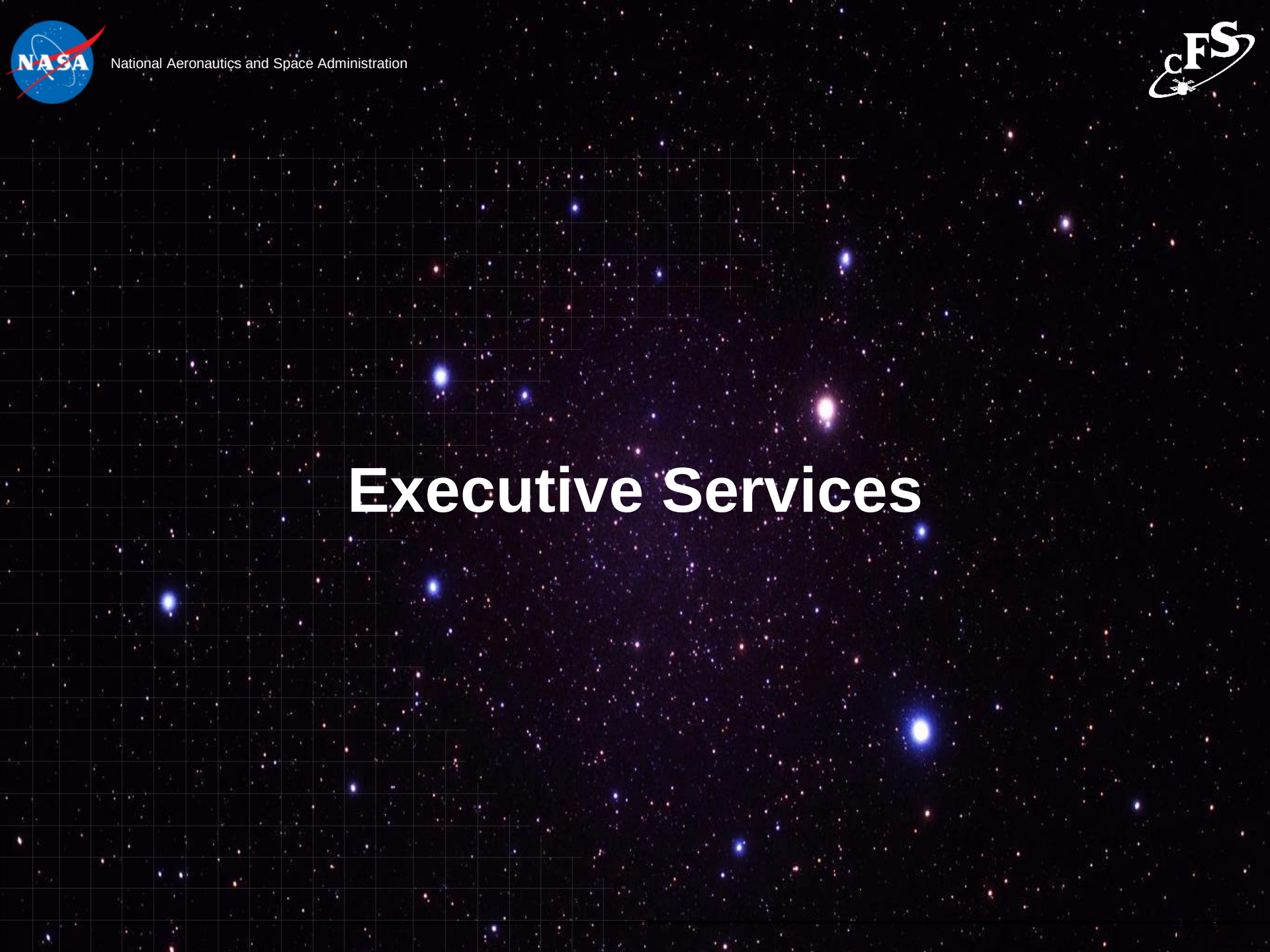
- **Software Bus Message Identifiers**
  - cfe_msgids.h (message IDs for the cFE should not have to change)
  - app_msgids.h (message IDs for the Applications) are platform configurations
- **Executive Service Performance Identifiers**
  - cFE performance IDs are embedded in the core
  - app_perfids.h (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**

cFS Application Mission and Platform Configuration Files

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	

cFE Service Training Template

- **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


National Aeronautics and Space Administration
Executive Services

Executive Services (ES) – Overview

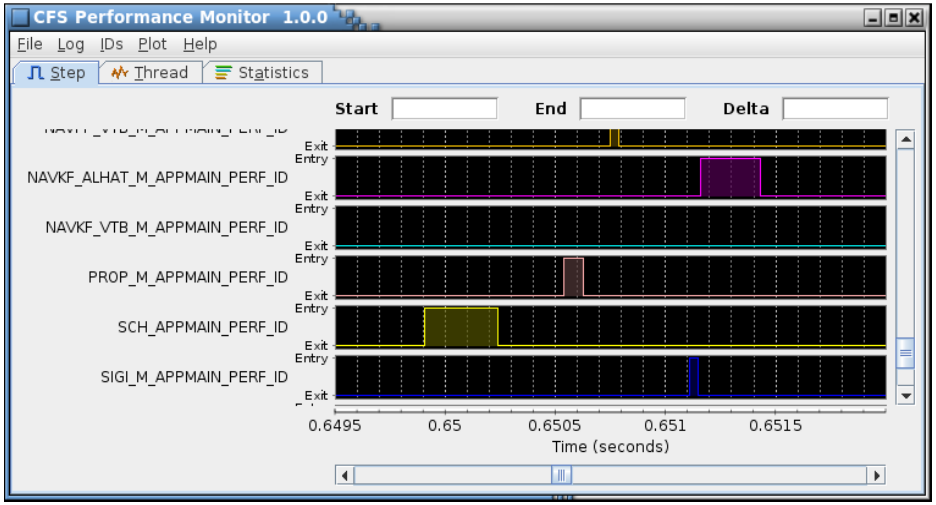
- **Initializes the cFE**
  - Reports reset type
  - Maintains an exception-reset log across processor resets
- **Creates the application runtime environment**
  - Primary interface to underlying operating system task services
  - Manages application resources
  - Supports starting, stopping, and loading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

Executive Services – Applications

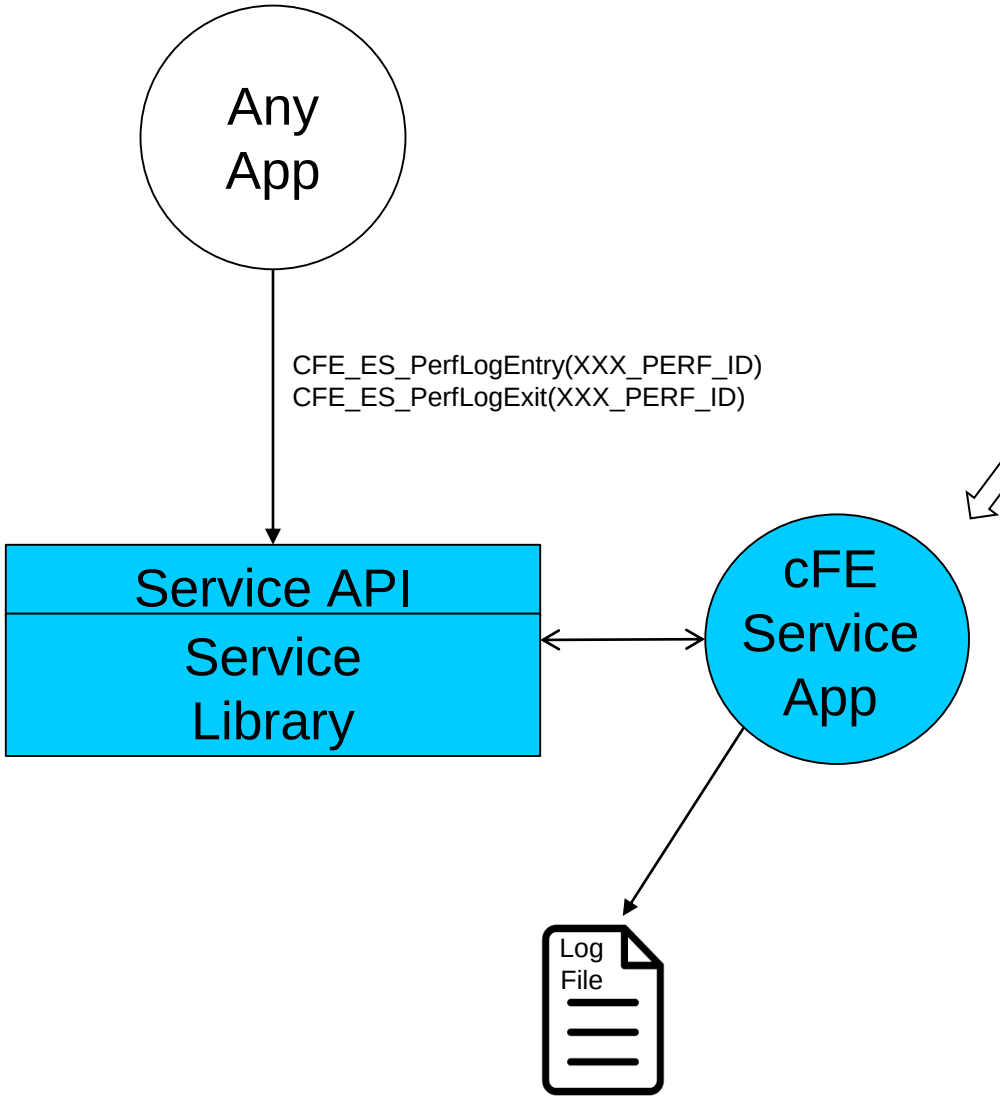
- **Applications are an architectural component that owns cFE and operating system resources**
- **Each application has a thread of execution in the underlying operating system (i.e. a task)**
- **Applications can create multiple child tasks**
  - Child tasks share the parent task's address space
- **Mission 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

Executive Services – Performance Analyzer

- **Provides a method to identify and measure code execution paths**
  - System tuning, troubleshooting, CPU loading
- **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**



Performance Monitor

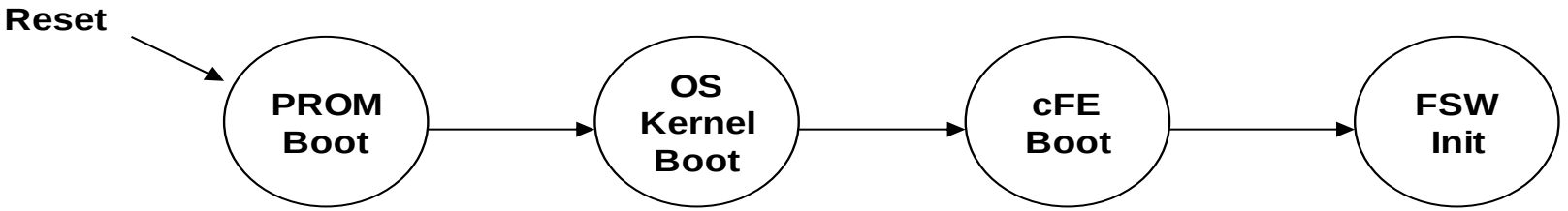


Executive Service Commands

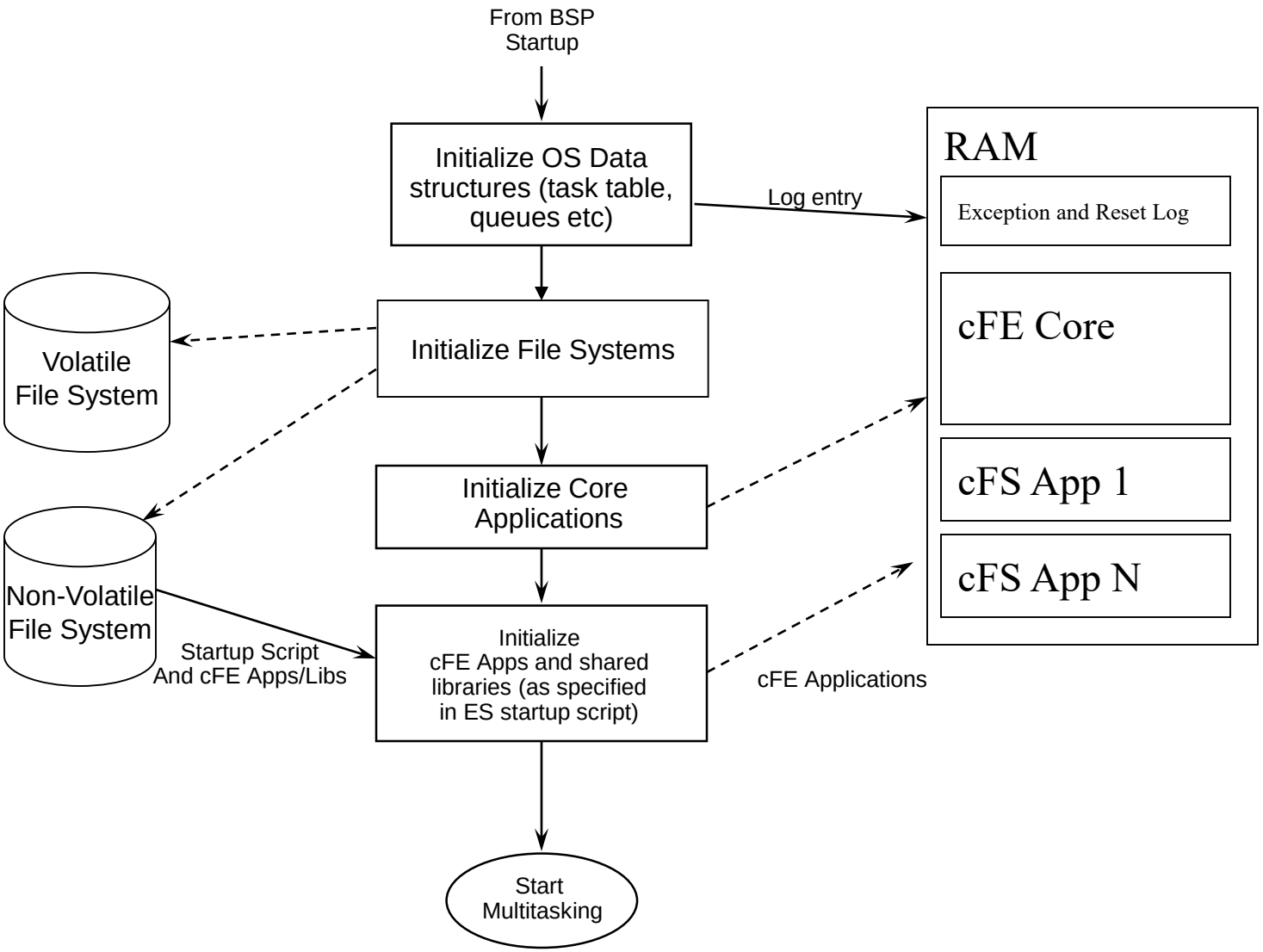
1. Set Masks
2. Set Triggers
3. Start Collection
4. Stop Collection

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

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 )
```

Executive Services – Logs

- **Exception-Reset**
  - Logs information related to resets and exceptions
- **System Log**
  - cFE apps use this log when errors are encountered during initialization before the Event Services is fully initialized
  - Mission apps can also use it during initialization
    - Recommended that apps should register with event service immediately after registering with ES so app events are captured in the EVS log
  - Implemented as an array of bytes that has variable length strings produced by printf() type statements

Executive Services – Reset Behavior

- **Power-on Reset**
  - Operating system loaded and started prior to cFE
  - Initializes file system
  - Critical data stores and logs cleared (initialized by hardware first)
  - ES starts each cFE service and then the mission applications
- **Processor Reset Preserves**
  - 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)
- **A power-on reset will be performed after a configurable number of processor resets**
  - Ground responsible for managing processor reset counter
- **Service initialization order: ES, EVS, SB, TIME, TBL, FS**
- **Service app initialization order: EVS, SB, ES, TIME, TBL**
- **Applications images are often compressed in non-volatile memory and decompressed by ES when they are loaded**

Executive Services – Retrieving Onboard State

- **Telemetry**
  - Housekeeping Status
    - Log file states, App, Resets, Performance Monitor, Heap Stats
- **Telemetry packets generated by command**
  - Single App Information
  - Memory Pool Statistics Packet
  - Shell command output packet
- **Files generated by command**
  - System Log
  - Exception-Reset Log
  - Performance Monitor
  - Critical Data Store Registry
  - All registered apps
  - All registered tasks

- **Child Tasks**
  - Recommend creating during app initialization
  - Relative parent priority depends on child's role
    - Performing lengthy process may be lower
    - Servicing short duration I/O may be higher

OS	Call	Notes
Posix/Linux	pthread_create()	
RTEMS	rtems_task_create()	
VxWorks	taskSpawn()	

Executive Services
System Integration and App Development (2 of 2)

- **Query startup type (Power On vs Processor)**
  - Not commonly used since CDS performs data preservation

- **Critical Data Store (CDS)**
  - E.g. Data Storage maintains open file management data in a CDS
  - Typical code idiom in app's initialization

```
Result = CFE_ES_RegisterCDS()
if (Result == CFE_SUCCESS)
   Populate CDS
else if (Result == CFE_ES_CDS_ALREADY_EXISTS)
   Restore CDS data
… Continually update CDS as application executes
```

- **Memory Pool**
  - Ideally apps would allocate memory pools during initialization but there aren't any restrictions
  - cFE Examples: Software Bus, Tables, and Events
  - App Examples: CFDP and Housekeeping

Executive Services – Configuration Parameters

List parameters that with higher probability of being tuned

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

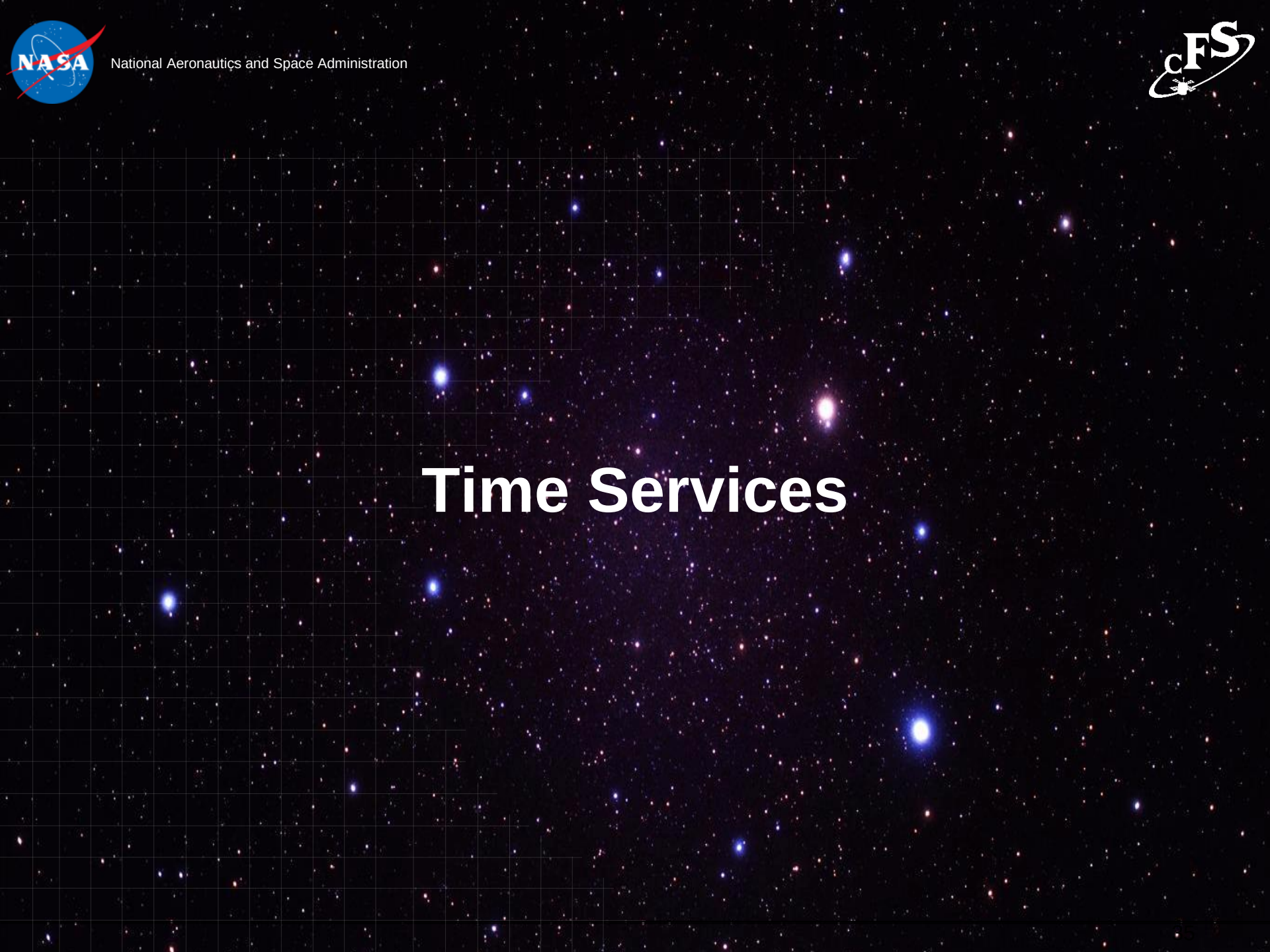
Performance Analysis Functions	Purpose
CFE_ES_PerfLogEntry	Entry marker for the performance analysis tool
CFE_ES_PerfLogExit	Exit marker for the performance analysis tool

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

Executive Services - Class Exercises

- See supplemental student material


National Aeronautics and Space Administration
Time Services

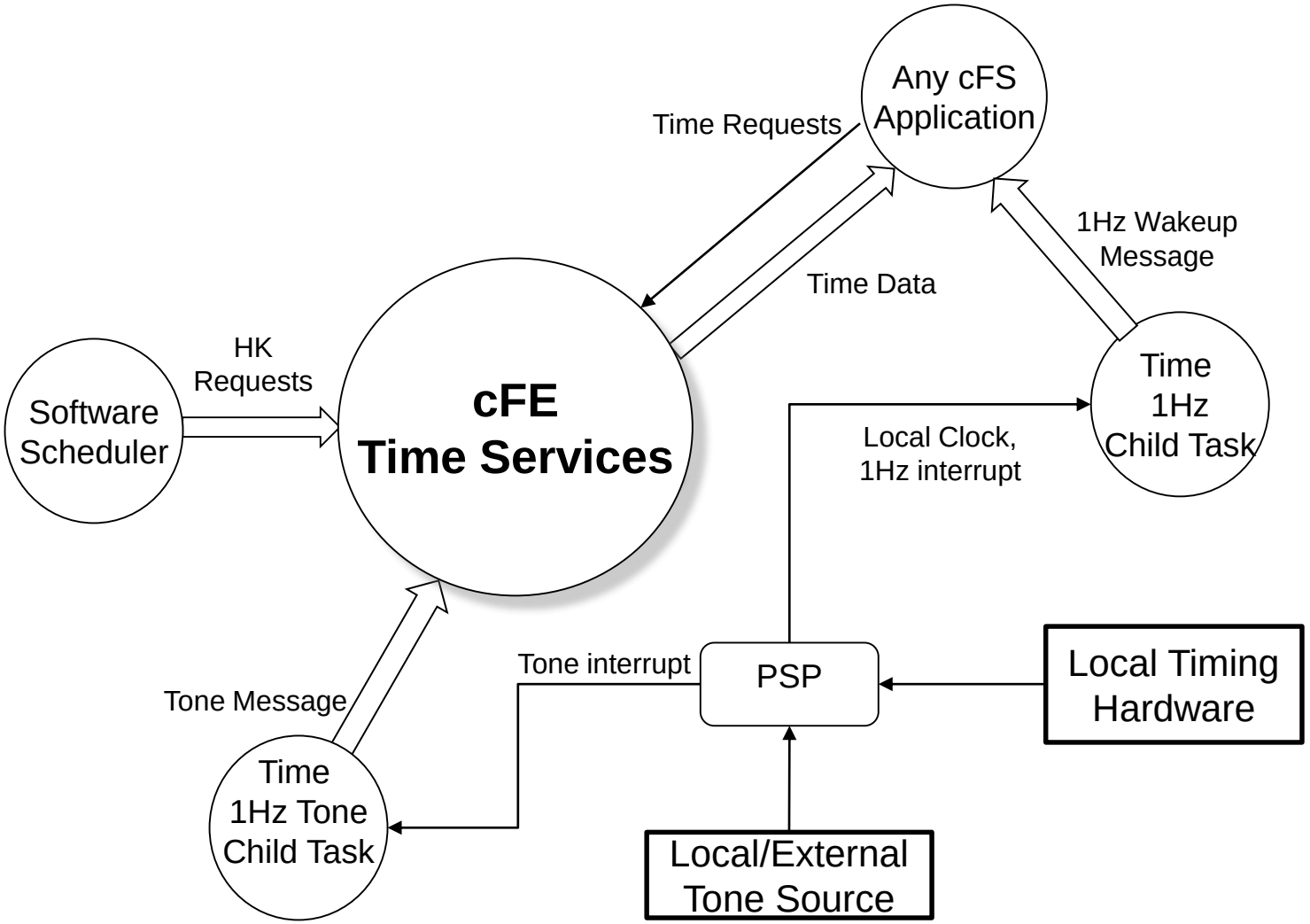
Time Services - Overview

- 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

Time Services – Time Formats

- **Supports two formats**
- **International Atomic Time (TAI)**
  - Number of seconds and sub-seconds elapsed since the ground epoch
  - TAI = MET + 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
  - UTC = TAI – Leap Seconds
  - Leap Seconds account for earth's slowing rotation

Time Services - Context



Time Services – “Flywheeling”

Flywheeling **occurs when TIME is not getting a valid tone signal or external "time at the tone" message. While this has minimal impact on internal operations, it can result in the drifting apart of times being stored by different spacecraft systems.**

- **Flywheeling occurs when at least one of the following conditions is true:**
  - loss of tone signal
  - loss of "time at the tone" data packet
  - signal and packet not within valid window
  - commanded into fly-wheel mode

Time Services – Reset Behavior

- **Power-On-Reset**
  - Initializes all counters in housekeeping telemetry
  - Validity state set to Invalid
  - STCF, Leap Seconds, and 1 Hz Adjustment to zero set to zero

- **Processor reset, preserves:**
  - MET
  - STCF
  - Leap Seconds
  - Clock Signal Selection
  - Current Time Client Delay (if applicable)
  - Uses ‘signature’ to determine validity of saved time. If signature fails then power-on-reset initialization is performed

Time Services – Retrieving Onboard State

- **Telemetry**
  - Housekeeping Status
    - Clock state, Leap Seconds, MET, STCF 1Hz Adjust
- **Telemetry packets generated by command**
  - Diagnostic Packet
- **Files generated by command**
  - None

Packet time stamps

Time Services – Configuration Parameters

List parameters that with higher probability of being tuned

System time is TAI or UTC

MET – Hardware register or software variable

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

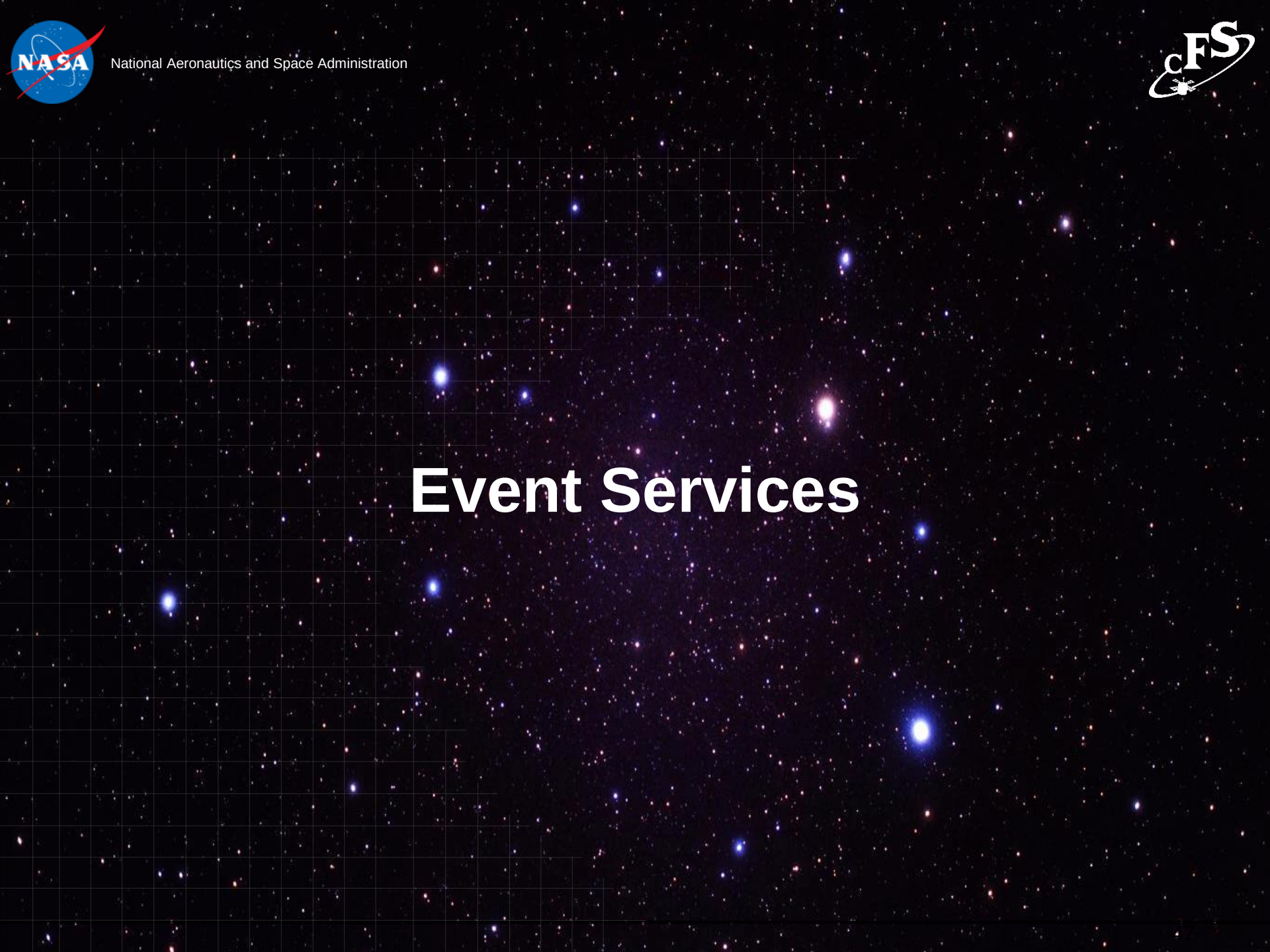
Time Services APIs

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

Time Services – Class Exercises

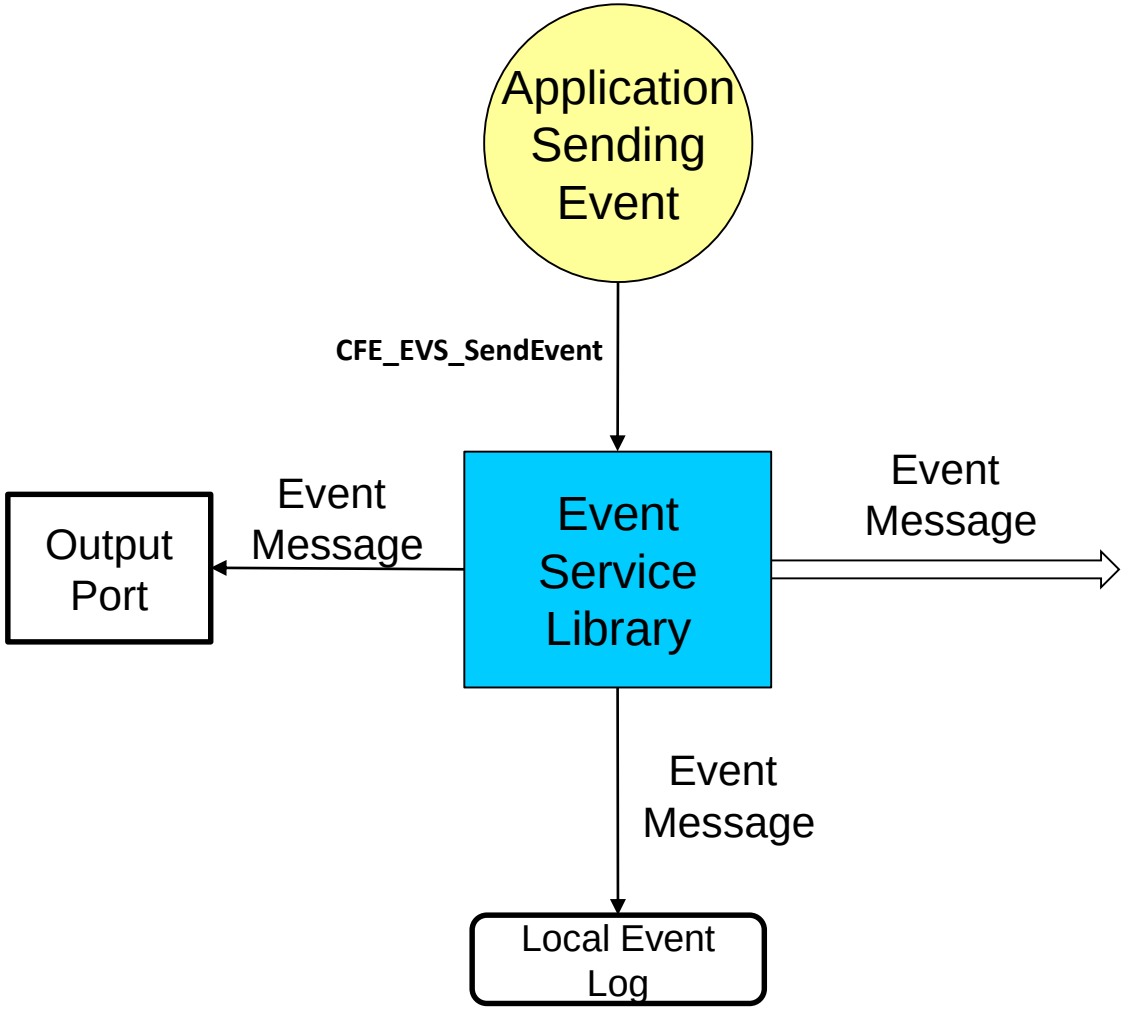
- See supplemental student material


National Aeronautics and Space Administration
Event Services

Event Services (EVS) - Overview

- **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

Event Services - Send Event



Event Services – Message Format

Example of an Event message:

- **Spacecraft time via CFE_TIME_GetTime()**
- **Spacecraft ID (not shown) defined in cfe_mission_cfg.h**
- **Processor ID defined in cfe_platform_cfg.h**
- **Event ID is unique within an application**
- **Event Text is created using printf() format options**
- **“Short Format” platform option allows messages to be sent without text portion**

Event Services – Event Filtering

- **Bit-wise AND "filter mask"**
  - 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

- **Software Bus 'No Subscriber' is a good example of first N message filter**
  - See *cfe_platform_cfg.h* CFE_SB_FILTERED_EVENT1

- **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

- **Applications register events for filtering during initialization**
  - Registering immediately after ES app registration allows events to be used rather than syslog writes

- I've asked about this same thing in the past; the answer seems to be that EVS ports were a concept/idea that was never fully realized. As you noted, it is implemented to the point where four outputs have some degree of command control, but it stops short of being fully useful - all go to OS_printf and no provisions are there to do anything else with the event strings.
-
- There is an existing enhancement request to do something more useful with this feature, or if there is no desire to have multiple outputs, it should be removed. My vote would be to pipe the output to a function in the PSP, and the platform support package can then decide what to do with the event strings. This would give some level of customization at least, without requiring one to modify EVS code directly.
-
- Open to suggestions on this -- it all depends on whether there is a driving interest/need for such a feature and how it would be used in a real deployment.
-
-
- On 06/11/2018 01:32 PM, Fleming, Thadeus wrote:
- > After a cursory look through the cFE docs, I didn't find any
- > discussion on the concept behind EVS ports. I saw requirements that
- > EVS support a specific number of ports, and be able to turn them on
- > and off, but no rationale about their meaning or intended usage.
- >
- > In the public cFE version, all 4 ports just pass a message string to
- > OS_printf. Furthermore, the port API defined in cfe_evs_utils.c seems
- > very limited, only accepting a pre-formatted message. Since this API
- > isn't defined in a header, I assume it isn't public anyway.
- >
- > Can anyone shed any light on what this feature is intended to do and
- > how it is intended to be used? Is anyone using it beyond the OS_printf
- > implementation?

Event Services – Message Control

- **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

Event Services – Reset Behavior

- **Power-on Reset**
  - No data preserved
  - Application initialization routines register with the service
  - If configured local event log enabled

- **Processor Reset**
  - If configured with an event log, preserves
    - Messages
    - Mode: Discard or Overwrite
    - Log Full and Overflow status

7/31/18 Email

The documentation for CFE_EVS_Unregister says "Applications must call this routine as part of their orderly shutdown process."

However, I've never actually seen this done, and grepping through the product line apps doesn't show it being used anywhere.

Are the docs incorrect, and the function doesn't need to be used? Or is it just neglected in most apps?

Event Services – Retrieving Onboard State

- **Housekeeping Telemetry**
  - Log Enabled, Overflow, Full, Enabled
  - For each App: AppID, Events Sent Count, Enabled

- **Write application data to file. For each app**
  - Active flag – Are events enabled
  - Event Count
  - For each filtered event
    - Event ID
    - Filter Mask
    - Event Count – Number of times Event ID has been issued

- **Local event log**
  - If enabled events are written to a local buffer
  - Log “mode” can be set to over write or discard
  - Serves as backup to onboard-recorder during initialization or error scenarios
  - Suitable for multi-processor architectures
  - Command to write log to file

- Debug
  - Typically disabled for flight
- Information
  - Missions should be consistent in usage
  - TBD – Add guidelines

Event Services – Configuration Parameters

List parameters that with higher probability of being tuned

Event Services Key Configuration Parameters

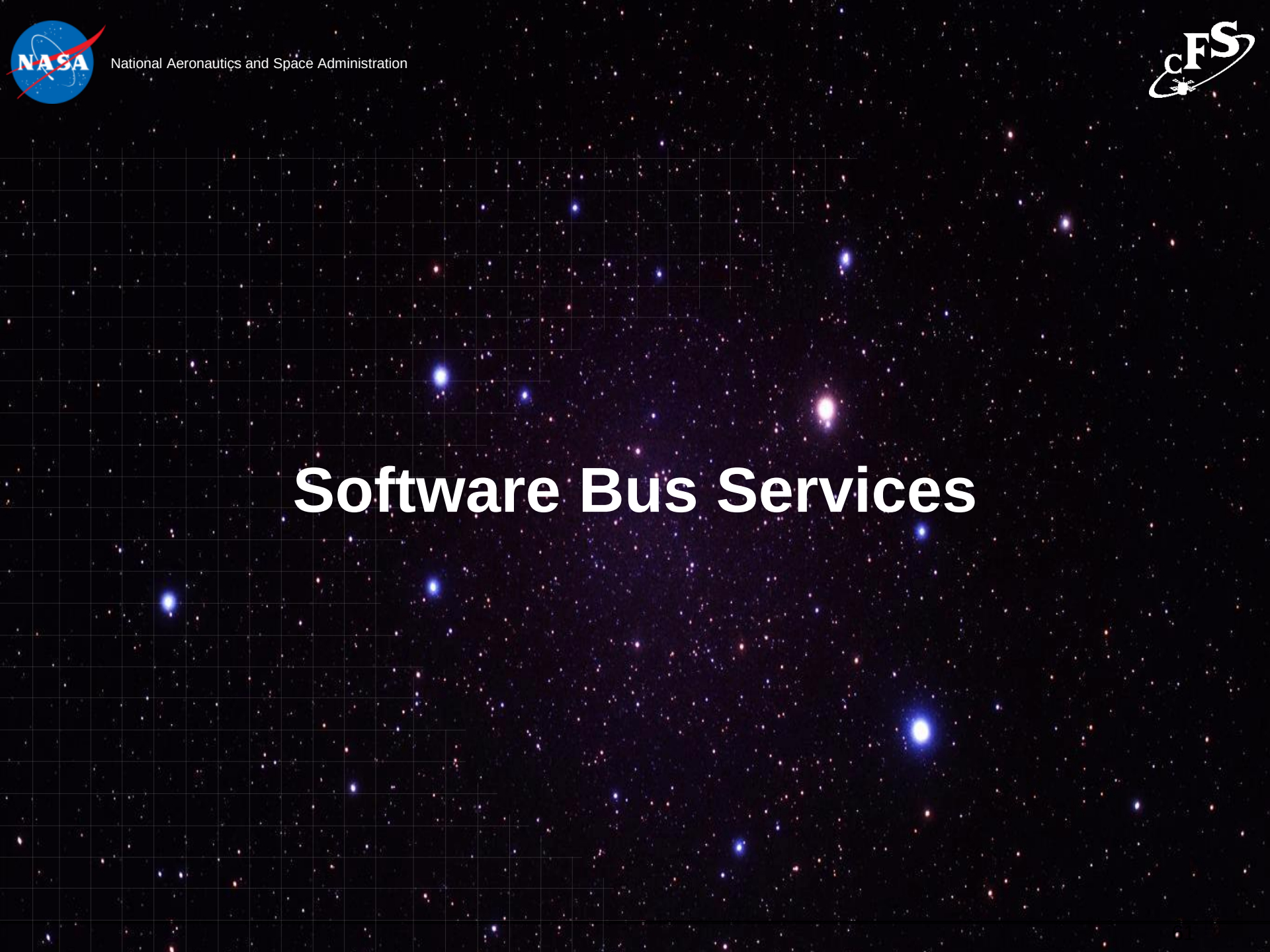
Parameter	Purpose	Scope	Notes
CFE_EVS_LOG_ON	cFE core platform message IDs	Platform	Defines the message IDs the cFE core will use on that Platform(CPU)
CFE_EVS_LOG_MAX	OSAL platform configuration	Platform	
CFE_EVS_DEFAULT_LOG_MODE	cFE core platform configuration	Platform	Most cFE parameters are here
XX_platform_cfg.h	Application platform wide configuration	Platform	
CFE_EVS_DEFAULT_TYPE_FLAG	Application message IDs	Platform	
CFE_EVS_MAX_EVENT_FILTERS	Define Maximum Number of Event Filters per Application	Platform	

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

Event Services - Class Exercises

- See supplemental student material


National Aeronautics and Space Administration
Software Bus Services

Software Bus (SB) Services - Overview

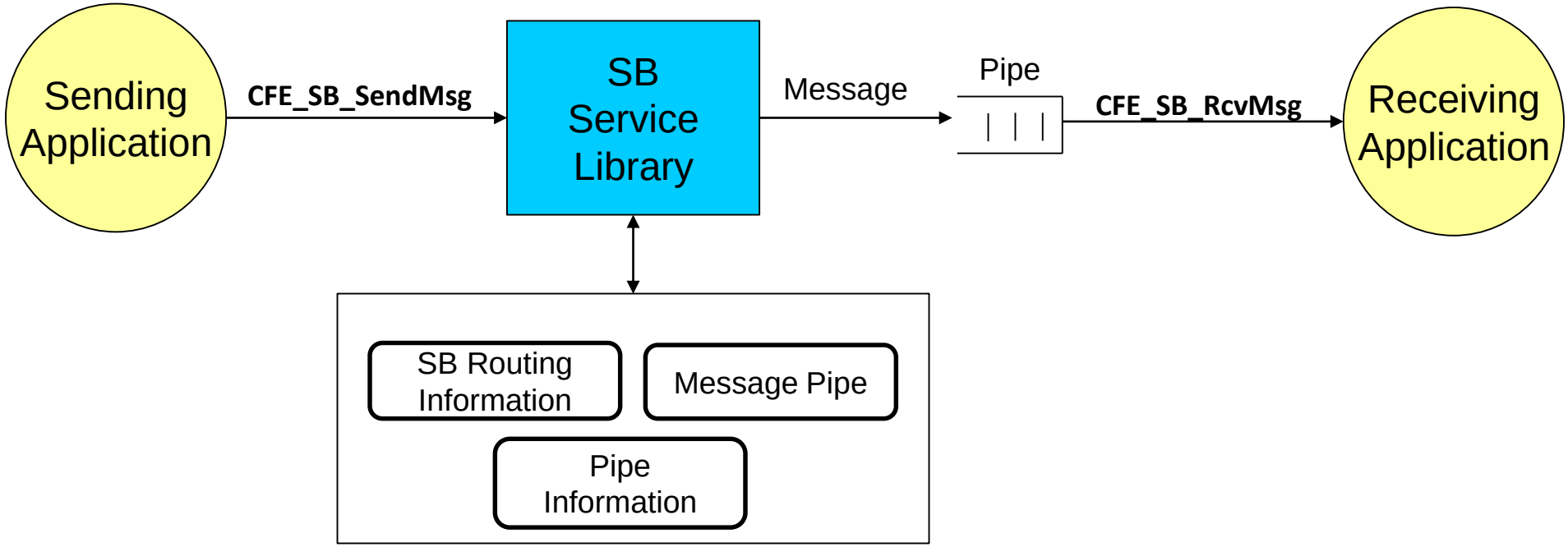
- **Provides a portable inter-application message service using a publish/subscribe model**

- **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**

Software Bus – Send & Receive



- **Applications create "pipes" and subscribe to receive messages**
  - Typically performed during application initialization although can be done at any time
  - Subscription specifies maximum pipe depth and maximum number of messages for each message ID

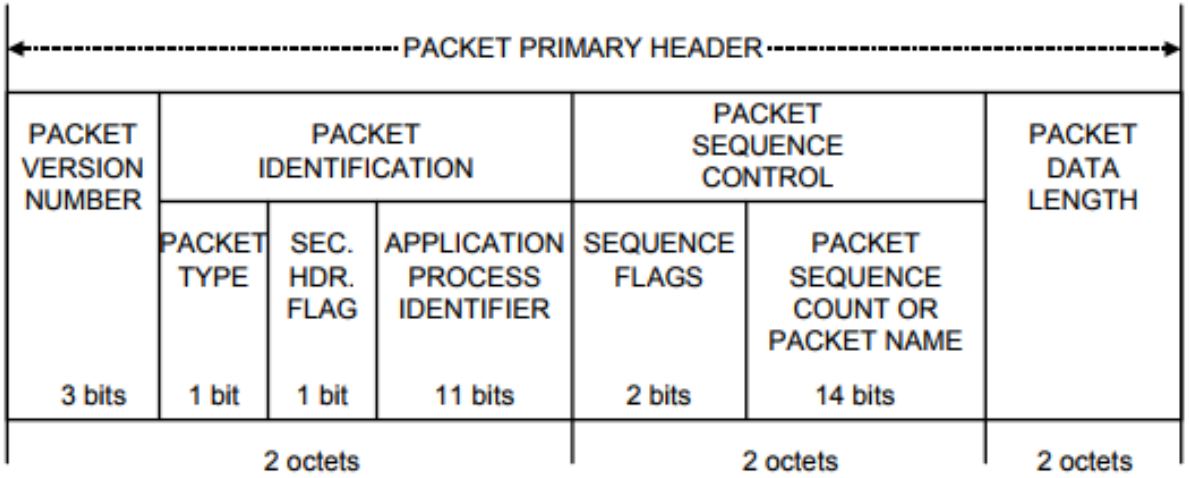
- **Apps can subscribe and unsubscribe to messages at any time**

Software Bus – Messages (1 of 2)

- **Messages**
  - Data structures used to transfer data between applications

- **By default Consultative Committee for Space Data Systems (CCSDS) packets used to implement messages**
  - In theory other formats could be used but has not occurred in practice
  - Simplifies data management since CCSDS standards used for flight-ground interfaces

- **CCSDS Primary Header (Always big endian)**



Software Bus – Messages (2 of 2)

- **"Packet" often used instead of "message" but not quite synonymous**
  - "Message ID" (first 16-bits) used to uniquely identify a message
  - "App ID" (11-bit) CCSDS packet identifier

Extended ApId

  - TBD describe concept and cFE 6.6 support

- **CCSDS Command Packets**
  - Secondary packet header contains a command function code
  - cFS apps typically define a single command packet and use the function code to dispatch a command processing function
  - Commands can originate from the ground or from onboard applications

- **CCSDS Telemetry Packets**
  - Secondary packet header contains a time stamp of when the data was produced
  - Telemetry is sent on the software bus by apps and can be ingested by other apps, stored onboard and sent to the ground

Software Bus – Message Formats

- **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;
```

Software Bus – Reset Behavior

- **No data is preserved for either a Power-On or Processor Reset**
  - All routing is reestablished as application create pipes and subscribe to messages
  - Any packet in transit at the time of the reset is discarded
  - All packet sequence counters reset to 1

Static SB?

Software Bus – Retrieving Onboard Stateate

- **Telemetry**
  - Housekeeping Status
    - Counters (No subscribers, send errors, pipe overflows, etc.), Memory Stats
- **Telemetry packets generated by command**
  - Statistics
  - Subscription Report
- **Files generated by command**
  - Routing Info
  - Pipe Info
  - Message ID to Route

TBD

Software Bus – Configuration Parameters

List parameters that with higher probability of being tuned

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	

Software Bus - Class Exercises

- See supplemental student material

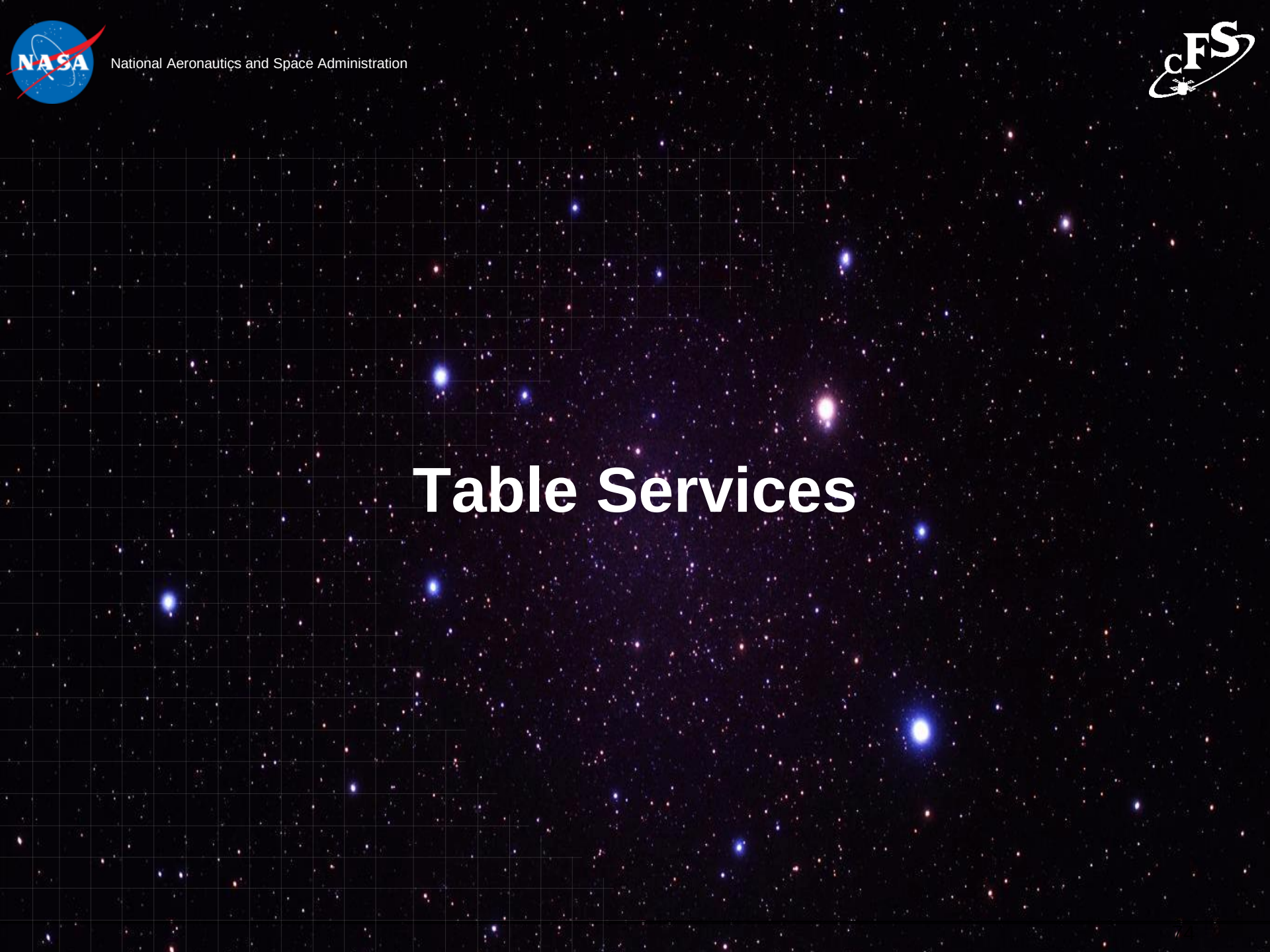
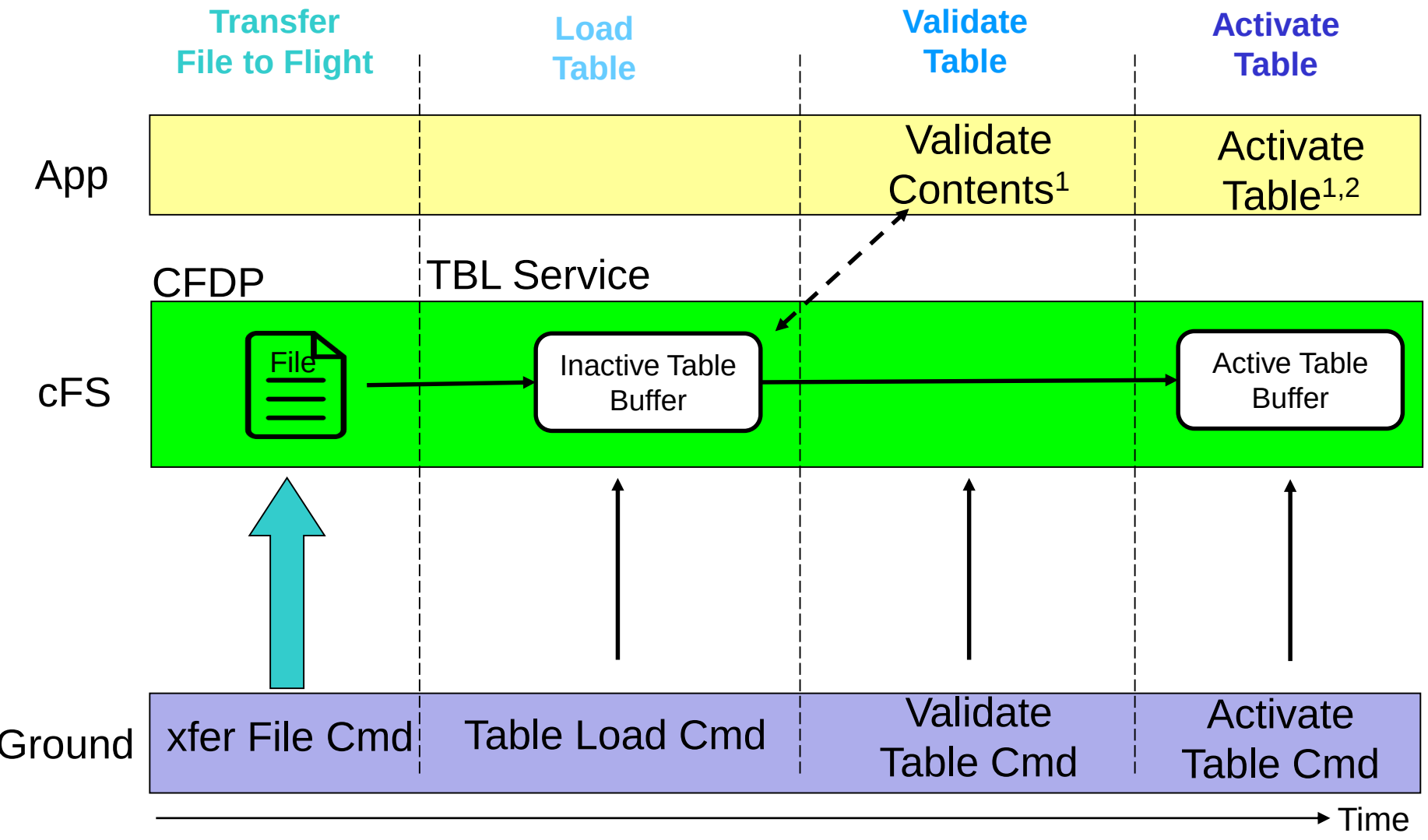

National Aeronautics and Space Administration
Table Services

Table Services (TBL) - Overview

- **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

Load Table



1. Apps typically validate & activate tables during their “housekeeping” execution cycle
2. In addition to instructing cFE to copy the contents, apps may have app-specific processing

Dump Table

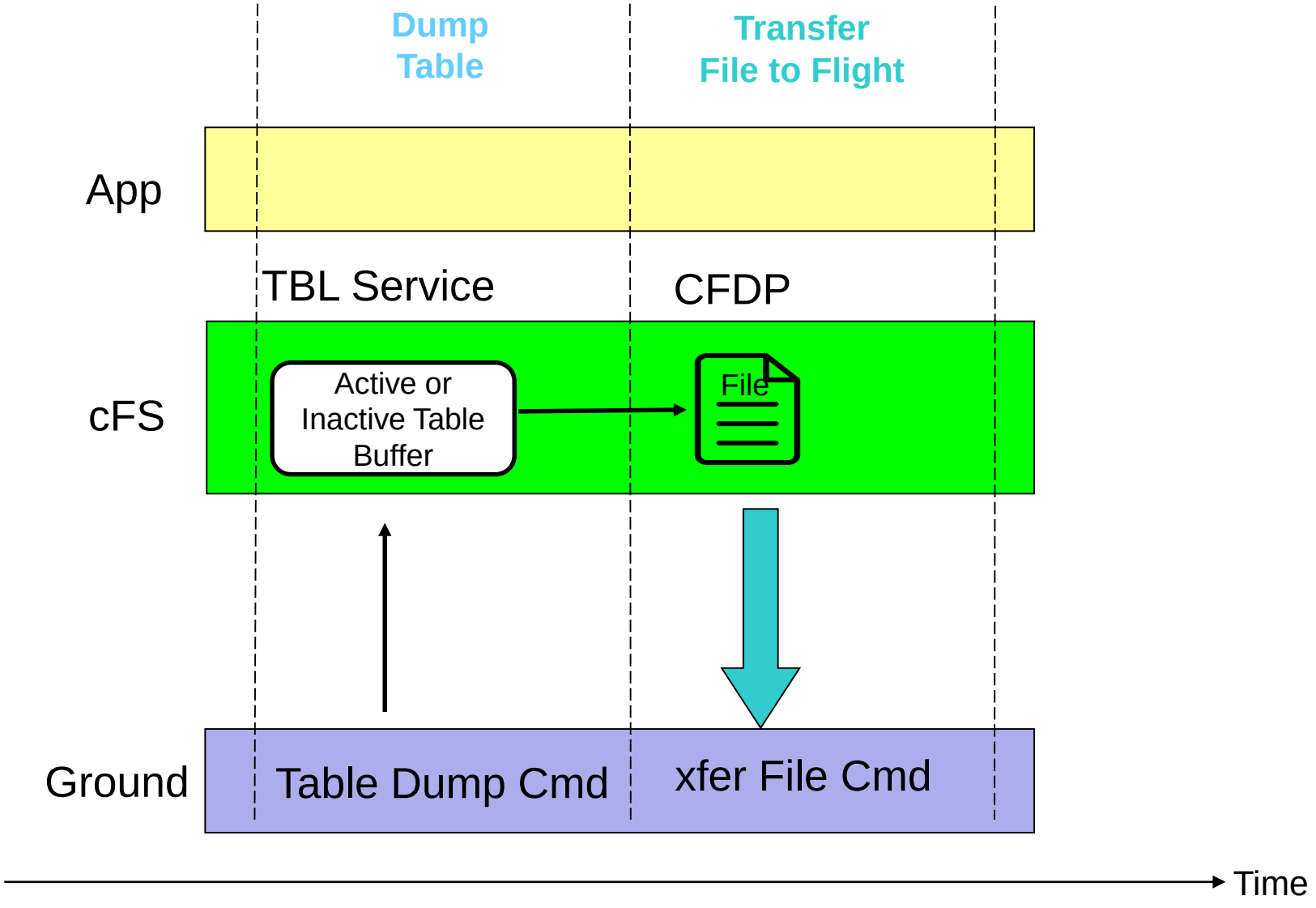
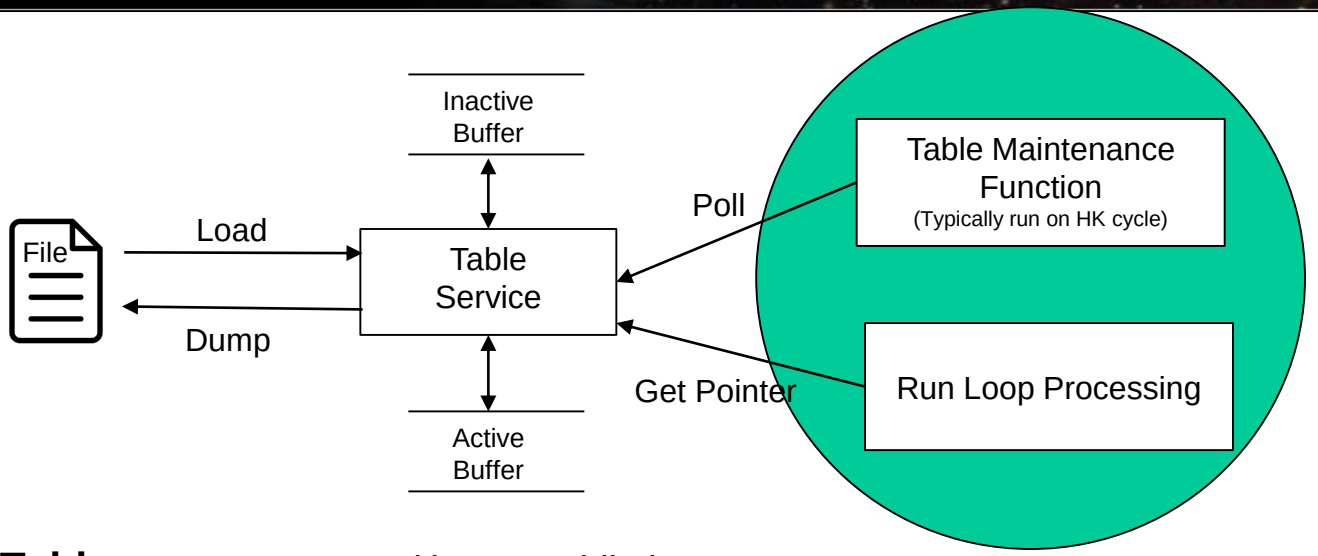

Dump Table
Transfer File to Flight
App
TBL Service
CFDP
cFS
Active or Inactive Table Buffer
File
Ground
Table Dump Cmd
xfer File Cmd
Time

Table Services – Managing Tables



- **Active Table** - Image accessed by app while it executes
- **Inactive Table** - Image manipulated by ops (could be stored commands)
- **Load → Validate → Activate**
  - Loads can be partial or complete
  - For partial loads current active contents copied to inactive buffer prior to updates from file
  - Apps can supply a "validate function" that is executed when commanded
- **Dump**
  - Command specifies whether to dump the active or inactive buffer to a file
- **Table operations are synchronous with the application that owns the table to ensure table data integrity**
- **Non-Blocking table updates allow tables to be used in Interrupt Service Routines**

Table Services –Table Buffer Types

- **Single Buffer**
  - The active buffer is the only buffer dedicated to the application's table
  - Table service shares inactive buffers to service multiple app's with single buffer tables
    - CFE_TBL_MAX_SIMULTANEOUS_LOADS defines the number of concurrent table load sessions
  - Most efficient use of memory and adequate for most situations
  - Since

```
#define CFE_TBL_OPT_DEFAULT (CFE_TBL_OPT_SNGL_BUFFER | CFE_TBL_OPT_LOAD_DUMP)
```

- **Double Buffer**
  - Dedicated inactive image for each double buffered table
  - Useful for fast table image swaps (.e.g. high rate app and/or very large table) and delayed activation of table's content (e.g. ephemeris)
  - E.g. Stored Command's Absolute Time Command table
- **Shared single buffer pool must be sized to accommodate the largest single buffer image**

Table Services –Table Attributes

- **Validation Function**
  - Applications register validation functions during initialization
  - Table activates for tables with validation functions will be rejected if the validation has not been performed
  - Mission critical data table values are usually verified

- **Message Notification**
  - Describe message notification API. SC is an example use case

- **Critical Tables**
  - Table data is stored in a Critical Data Store
  - Contents updated for each table active command

- **User Defined Address**
  - Application provides the memory address for the active table buffer
  - Typically used in combination with a dump-only table

- **Dump-Only**

Table Services – Reset Behavior

- **Table registry is cleared for power-on and processor resets**
  - Applications must register tables for any type of reset
  - Applications must initialize their table data for any type of reset

- **Critical Table Exception**
  - If a table is registered as critical then during a processor reset table service will locate and load the preserved table data from a critical data store

Table Services – Retrieving Onboard State

Housekeeping Telemetry

- TBD

- **Telemeter Application Registry**
  - Telemeter the Table Registry contents for the command-specified table

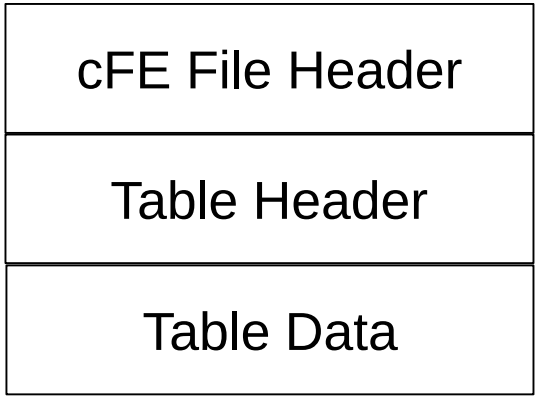
- **Dump Table Registry**
  - Write the pertinent table registry information to the command-specified file

- **Dump Table Registry**
  - The Application ID of the Application that Registered the table
  - The full name of the table
  - The size, in bytes, of the table
  - Pointers to the start addresses of the Table's image buffers, Active and Inactive (if appropriate)
  - A pointer to the start address of a Validation Function
  - A flag indicating whether a table image has been loaded into an Inactive buffer
  - A flag indicating whether the table is Critical and its associated CDS Handle if it is
  - A flag indicating whether the table has ever been loaded (initialized)
  - A flag indicating whether the table is Dump Only
  - A flag indicating whether the table has an Update Pending
  - A flag indicating whether the table is double buffered or not
  - The System Time when the Table was last Updated
  - The filename of the last file loaded into the table
  - The File Creation Time for the last file used to load the contents of the table

Table Services
System Integration and App Development (1 of 2)

- **Commands are typically used to initiate an action; not tables**
  - For example, change a control mode
- **Sometimes convenience commands are provided to change table elements**
  - For example, scheduler app provides an enable/disable scheduler table entry
- **Typically tables do not contain dynamic data computed by the FSW**
  - The cFE doesn't preclude this and it has been used as a convenient method to collect data, save to a file, and transfer it to the ground
  - These are defined as dump-only tables
  - Static tables can be checksummed
- **Tables can be shared between applications but this is rare**
  - Tables are not intended to be an inter-application communication mechanism

- **Load/dump files are binary files with the following sections:**



- **Table header defined in cfe_tbl_internal.h**

```
{
   uint32   Reserved;    /**< Future Use: NumTblSegments in File?   */
   uint32   Offset;      /**< Byte Offset at which load should commence */
   uint32   NumBytes;    /**< Number of bytes to load into table */
   char     TableName[CFE_TBL_MAX_FULL_NAME_LEN]; /**< Fully qualified name of table */

} CFE_TBL_File_Hdr_t;
```

Table Services – Configuration Parameters

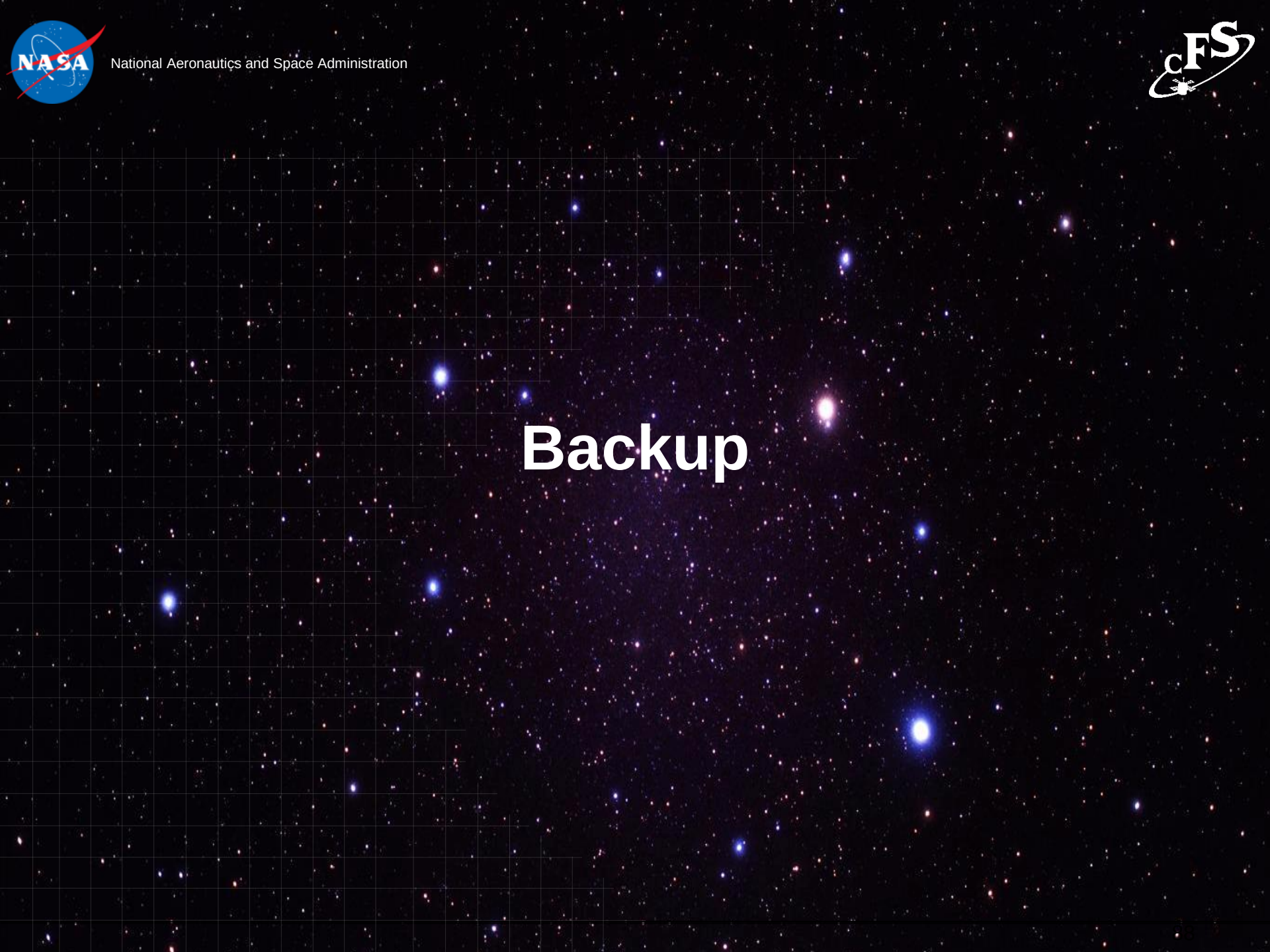
List parameters that with higher probability of being tuned

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.

Table Services - Class Exercises

- **See supplemental student material**


National Aeronautics and Space Administration
Backup

TBD

- TBD