cFS Overview

Version 1.18

August 2024

Audience & Prerequisites

- **Objectives**
  - Introduce flight software (FSW) concepts and core Flight System (cFS) architectural features
- **Intended audience**
  - Spacecraft technical managers, systems engineers, discipline engineers, and software engineers
- **Prerequisites**
  - An interest in learning about FSW and the cFS
  - No programming skills required

Outline

1. Flight Software Introduction
2. Core Flight System(cFS) Introduction

Appendix A: Acronyms

Appendix B: cFS Architectural Goals

Appendix C: Online Resources

Flight Software Introduction

What is Flight Software?

Flight Software (FSW) runs on one or more embedded computers on a spacecraft

- In general, a spacecraft is a remotely operated robotic system so FSW can be applied to a wider range of applications than spacecrafts

Ground System

- Provides a user interface to the spacecraft
- Contains tools for command generation, autonomous operations, data analysis, etc.

Telecommands

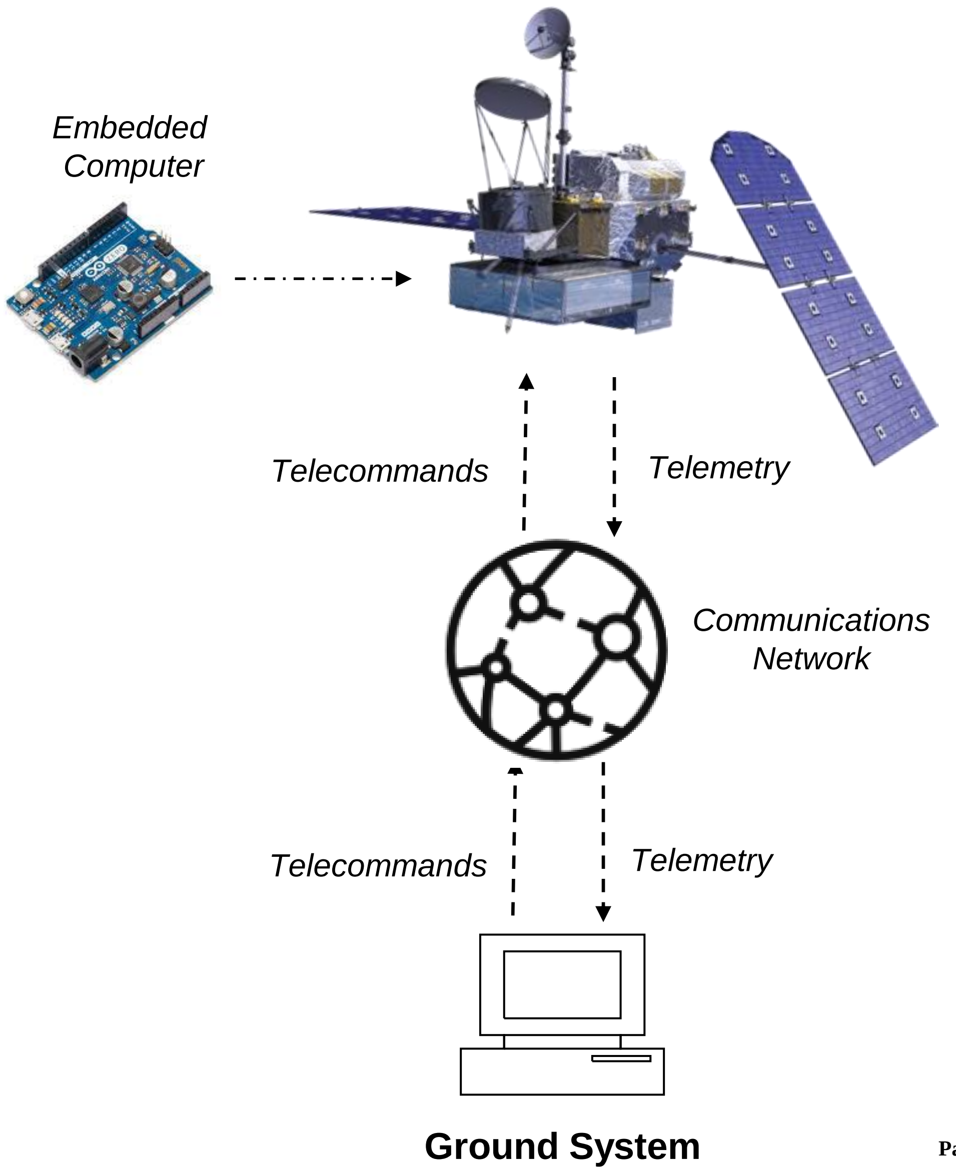
- Commands sent from a ground system to the spacecraft

Telemetry

- Data sent from the spacecraft to the ground system

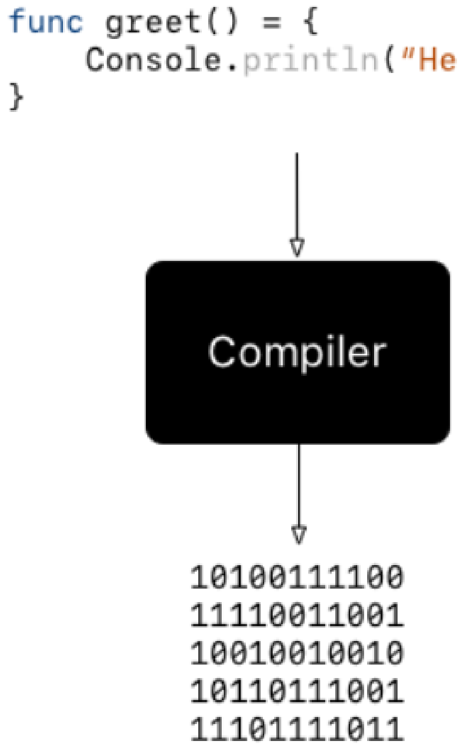
Communications Network

- Provides communitions between a ground system and a spacecraft
- This can be architected in many different ways and is rapidly changing with the commercialization of Low Earth Orbiting (LEO) space



Let’s start at the beginning for those no experience with developing software…

- **Microcode are instructions and data processed by a microprocessor (i.e. computer)**
- **Programming languages are formal languages (i.e. grammatical rules) that are translated into microcode by a compiler**



Different compilers are required for different processors

Some programming languages such as Python, Ruby, and JavaScript are translated into bytecode that is interpreted and run on virtual machines (VMs)

- VMs are ported to different processors
- Programs can run on platforms that have the VM and any required support libraries installed

What is an Embedded Computer?

- An *Embedded Computer* is a computer that is integrated in a product
- Embedded Computers do not usually have a keyboard, mouse, or monitor interface

Embedded Software is software that that runs on an *Embedded Computer* in order to make a device or product work

"Embedded software is the opposing thumb of hardware"

Embedded Software Characteristics

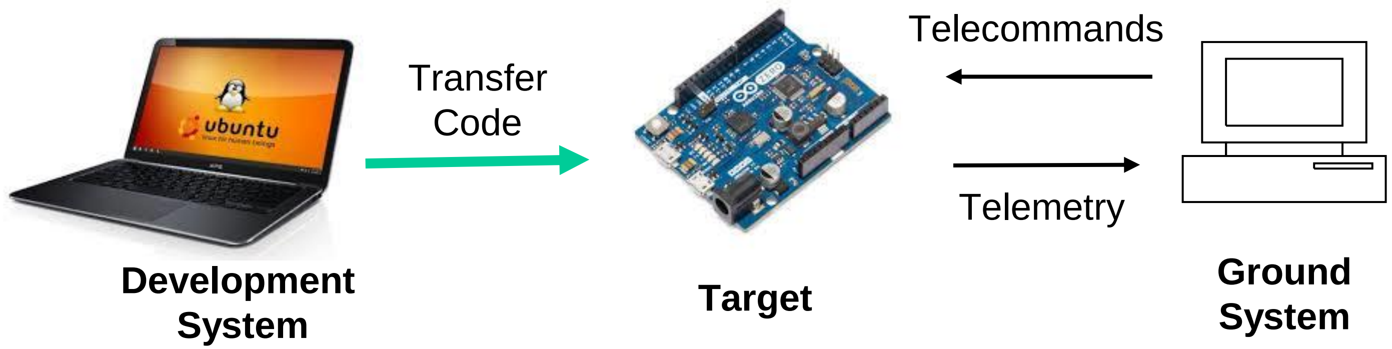
- **Microcontroller**
  - A microprocesor with limited resources: low power, small memory, and slower clock speeds
  - Typically, no operating system
  - For example, a processor embedded with a momentum wheel or a car’s antilock brake system

- **Embedded Operating System**
  - Real-time preemptive multi-tasking
  - Typically, much smaller than Windows or OS X
  - Examples include VxWorks, RTEMS, and FreeRTOS

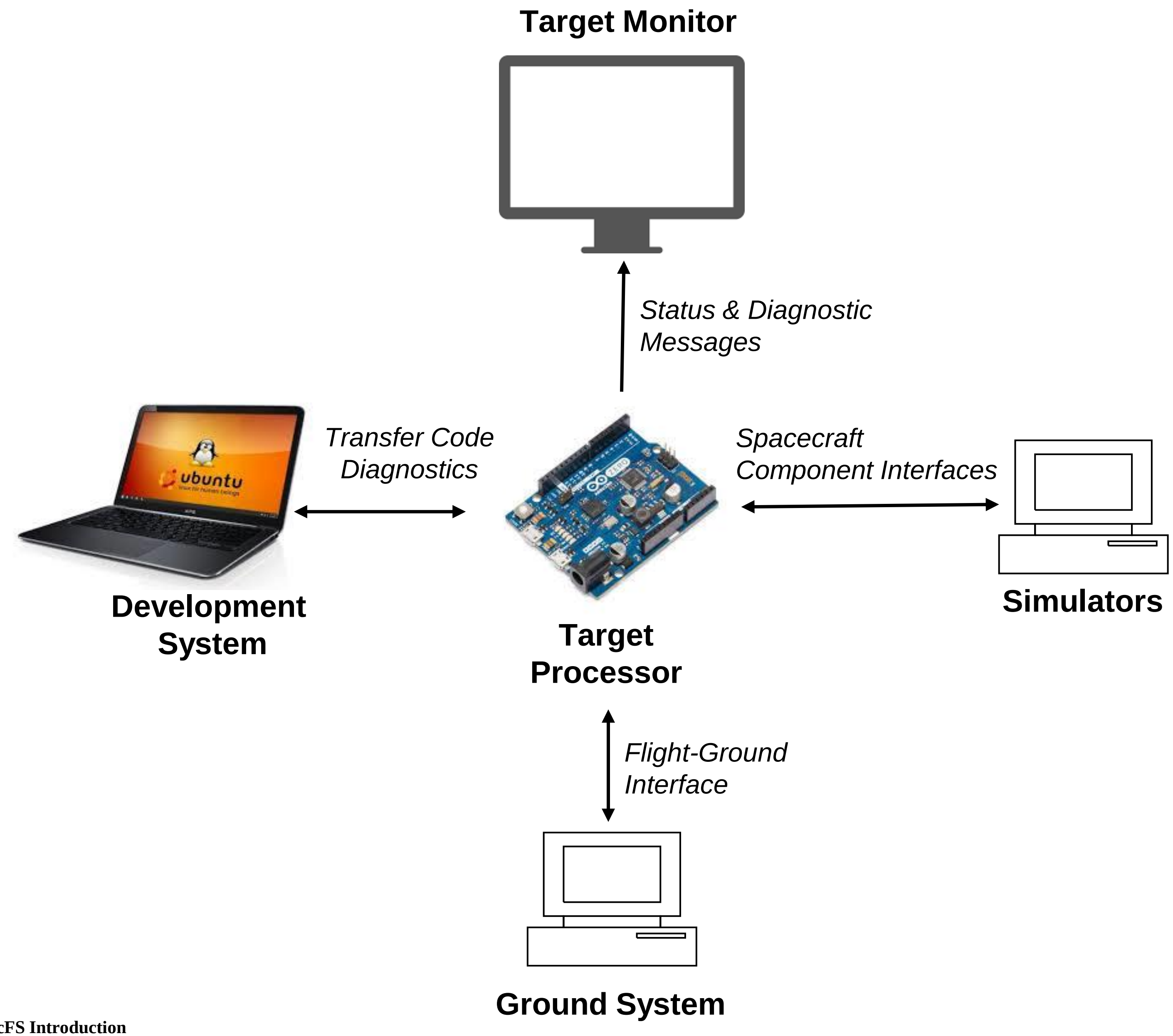
- **Embedded System**
  - The combination of an *Embedded Computer* and *Embedded Software*

How do we Develop Embedded Software?

- **Use a Windows, Mac, or Linux computer to write the code**
  - Programming Languages used
    - For Flight Software: C, C++, Assembly Language, sometimes Ada
    - For Ground and test software: C, C++, Python, Java, Ruby, etc.
- **Write code, cross-compile for target processor, transfer object code to embedded computer, control embedded system with a ground system**
  - Cross compilers are required when the target processor is different than development system's processor
- **This is known as Cross Development**



Cross Development Environment



Target Processor
- A processor that runs the cFS target image

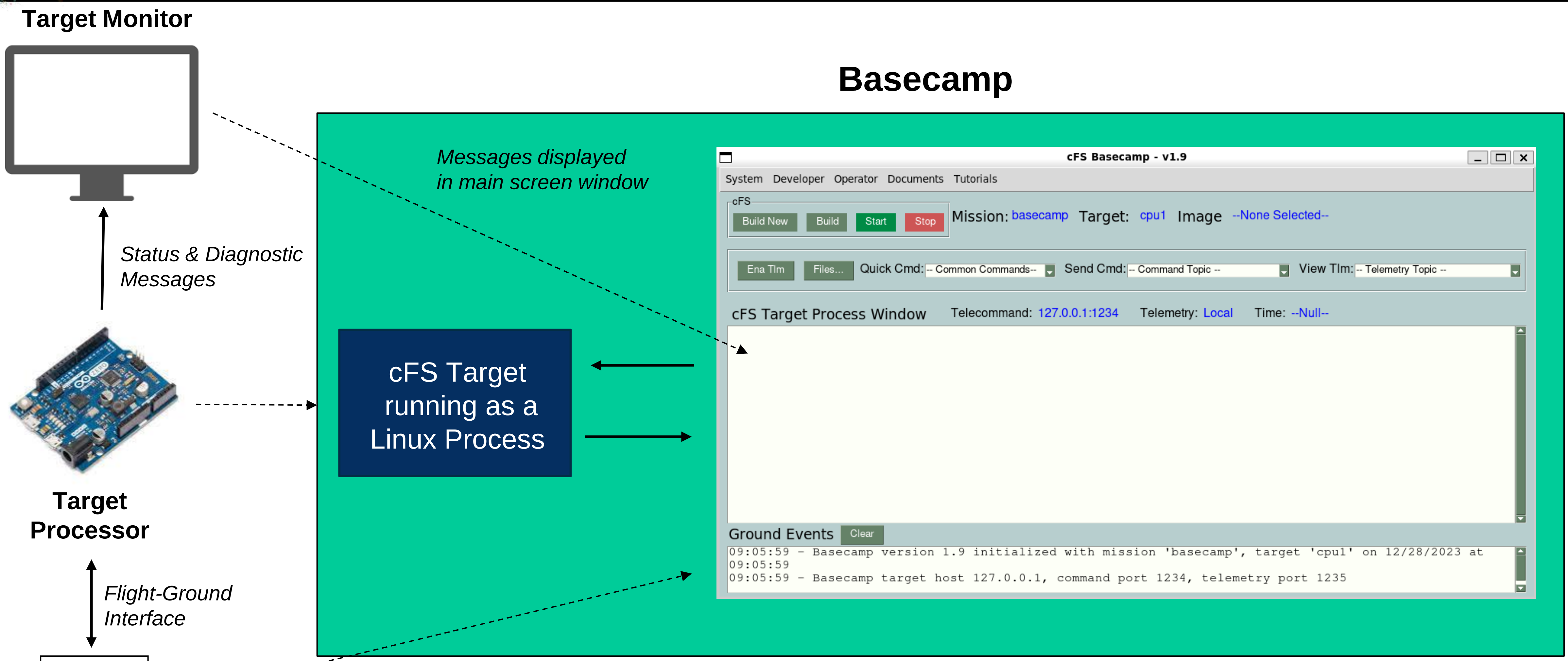
Development System
- Used to build and transfer the cFS target image to the target processor
- Requires a *Cross Compiler* if the target process is different than the development system
- May include runtime diagnostic tools

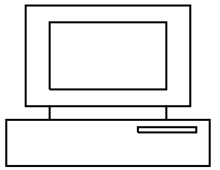
Target Monitor
- A common diagnostic tool used to help verify the embedded system is operating correctly
- Monitors are often connected over serial ports

Simulators
- Simulate spacecraft components, space environment, and spacecraft attitude and orbit dynamics
- Different environments can be created to develop and verify subsets of the flight software

Ground System
- An application that sends telecommands to the target and receives telemetry messages from the target
- The command & telemetry communications link may vary between test configurations and operations

Basecamp's Relationship to Cross Development Environment





Python menus and tools serve as the ground system

- *Basecamp provides enough functionality for cFS education and is not intended to be a fully functional ground system*

Spacecraft Definitions

- **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**

FSW Challenges – Limited Resources

- **Flight processor clock speeds are in the MHz range while our laptops are in the GHz range**
  - NICER[1] (6/12/17) 83Mhz Broad Reach 440 Power PC

- **Non-volatile memory**
  - GPM[2] (2/27/14) 2MB banks of EEPROM for each FSW image
  - Compressed operating system and apps

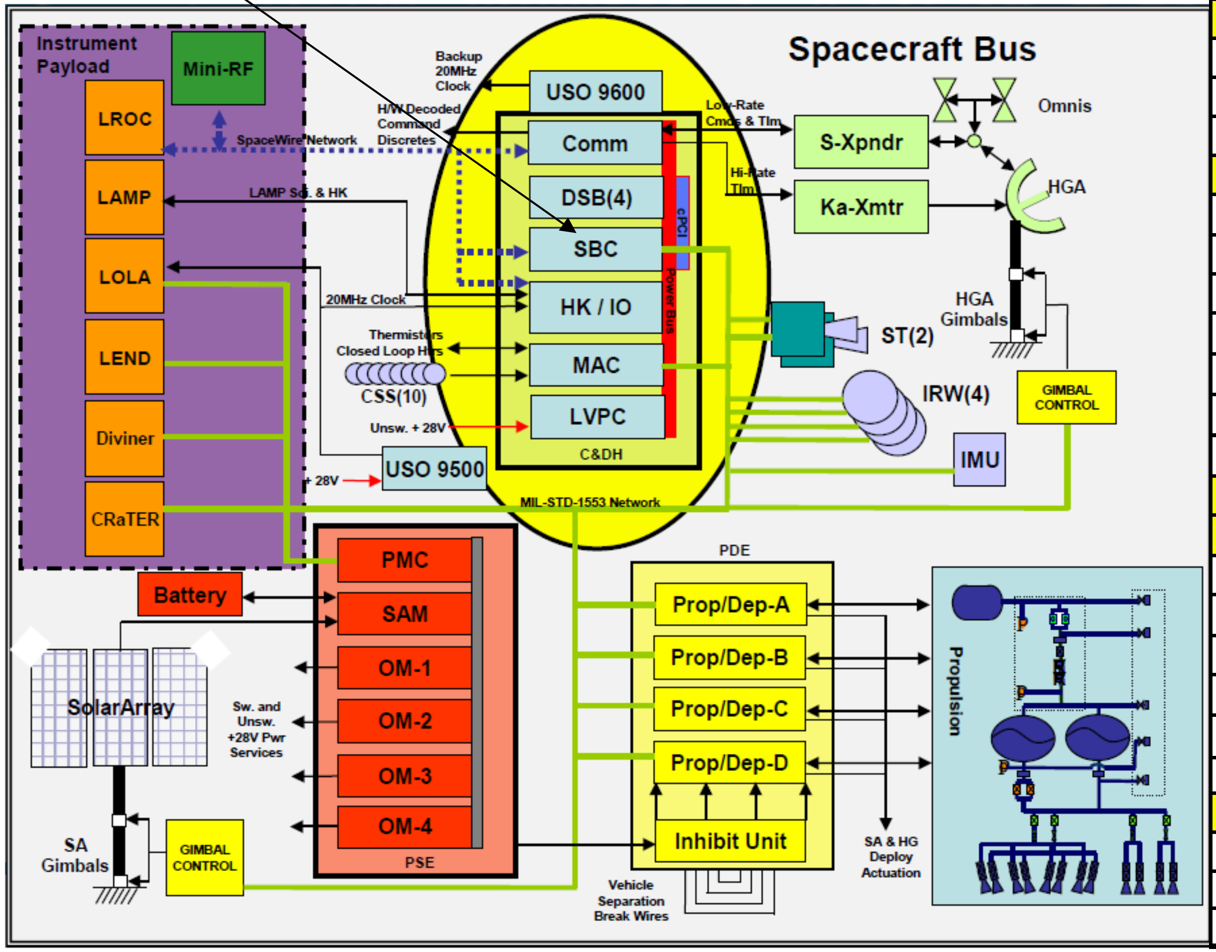
- **Flight RAM in Megabyte range while our laptops are in the Gigabyte range**
  - GPM (2/27/14) 36MB RAM

1. NICER - NASA Science
2. The Global Precipitation Measurement Mission (GPM) | NASA Global Precipitation Measurement Mission

FSW Challenges – Large number of Custom Interfaces

More than 20 Interfaces controlled by FSW on this single string example

FSW on
Single Board Computer



C&DH	Command & Data Handling
Comm	Communication
CRaTER	Cosmic Ray Telescope for the Effects of Radiation
CSS	Coarse Sun Sensor
DSB	Data Storage Board
EVD	Engine Valve Driver
HGA	High-Gain Antenna
H/W	Hardware
HK	Housekeeping
IO	Input/Output
IMU	Inertial Measurement Unit
IRW	Integrated Reaction Wheel
LAMP	Lyman-Alpha Mapping Project
LEND	Lunar Exploration Neutron Detector
LOLA	Lunar Orbiter Laser Altimeter
LROC	Lunar Reconnaissance Orbiter Camera
LVPC	Low Voltage Power Converter
MAC	Multi-Function Analog Card
OM	Output Module
PMC	Power Management Controller
PSE	Power System Electronics
SA	Solar Array
SAM	Solar Array Module
ST	Star Tracker
SBC	Single Board Computer
SSR	Solid State Recorder
Xmtr	Transmitter
Xpndr	Transponder

- **Curiosity Probe's Computer Reset (2/25/19)**
  - “Last week, its computer rebooted without warning. Now, NASA engineers are trying to figure out what caused the unprompted restart.”

- **Israeli Lunar Lander Suffers Glitch on Way to the Moon (2/27/19)**

  - "During the pre-maneuver phase the spacecraft computer reset unexpectedly, causing the maneuver to be automatically cancelled,“
  - "The engineering teams … are examining the data and analyzing the situation. At this time, the spacecraft's systems are working well, except for the known problem in the star tracker.“
  - https://www.space.com/israel-moon-lander-suffers-glitch.html

FSW Challenges – Robustness & Remote Diagnostics (2 of 2)

Intuitive Machines Marks a New Lunar Era with Help from NASA Goddard's Core Flight System (cFS)[1]

February 2024

Reflecting on the mission's success and the help of cFE, Intuitive Machine's Chief Technology Officer, Tim Crain, said "We stand on the shoulders of Giants. The work we were doing was built upon work people had done before us. NASA's core Flight software is a big part of what we do on the vehicle, and it has a lot of capabilities to reload and reinitialize software built into it, and we were able to take advantage of foresight of the people who had done space missions before had invested in that piece of technology and we use it to great effectiveness"

1. [A New Lunar Era with Help from Goddard's Core Flight System | Goddard Engineering and Technology Directorate (nasa.gov)]

FSW Challenges – Manage & Protect Hardware

- **<u>Detect</u> flight hardware anomalies and failures**
  - Sensors, Actuators, Clocks, CPU, Memory, Power, Thermal, Communications

- **<u>Respond</u> to onboard anomalies/failures – pre-planned actions, e.g.,**
  - Capture Diagnostic Data
  - Use “numerically safe” data for current control law cycle
  - Reconfigure onboard hardware and/or software – minimize the problem
  - ‘Safe’ all Flight Hardware (Science Instruments, Spacecraft Hardware)
  - Ensure Sun is on the Solar Array Panels -- Maintain Power Positive State
  - Notify Ground of problems

- **FSW can often compensate for hardware problems found during spacecraft I&T or post-launch**

FSW Challenges – Serving Multiple Customers

Principle Investigator

- “I want all science data in a timely manner”

Other spacecraft subsystems

- “It would be really nice if I could see data X, Y, and Z at rate N”

FSW Test Team

- “We want to see all of the data all of the time”

Spacecraft Integration and Test team

- “We want to see or at least log all of the data all of the time”
- “We also need to test as we will fly”

Operational Managers and Spacecraft Operators

- “Make my life simple and cheap”

FSW On Orbit Maintenance Team

- “Can we access/view data/code X and change it to Y?”

FSW Challenges – Complex Functionality

- FSW systems often have a large number of functional requirements as shown on the next slide

- Functions need to operate concurrently, at different rates, and with different real-time response constraints

- Functions have different levels of criticality and lower critical functions must not impact mission critical functions

Example Spacecraft Flight Software Top-Level Requirements

FSW

C&DH | ADCS-GNC | Mechanisms | Thermal | Power | Payload

C&DH

- **Operational Environment**
  - App Runtime Env
  - Command Ingest
  - Telemetry Output
  - Inter-processor Comm
  - Time Management
- **Onboard Data Management**
  - File Management
  - File Transfer
  - Recorder Management
- **Autonomy, Failure Detection & Correction**
  - Stored Commanding
  - Hardware & Software Monitoring
  - Memory Integrity Support & Checks

ADCS-GNC

- **Attitude Determination and Control**
- **Orbit Determination and Control**
- **Models**
  - Solar
  - Lunar
  - Magnetic Field

Mechanisms

- **Deployments**
- **Solar Array Control**
- **Antenna Control**

Thermal

- **Autonomous Control**
- **Tlm & Cmd I/F**
- **Health & Safety**

Power

- **Autonomous Control**
- **Tlm & Cmd I/F**
- **Health & Safety**

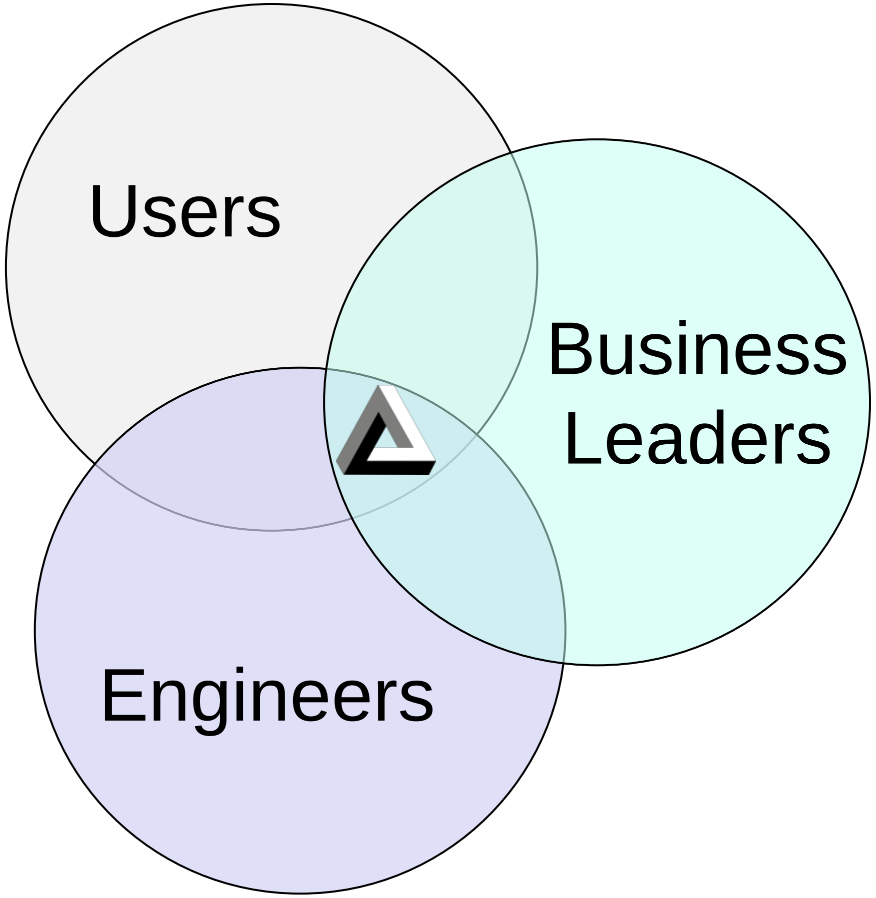
Payload

- **Autonomous Control**
- **Data Management**
- **Tlm & Cmd I/F**
- **Health & Safety**

ADCS: Attitude Determination & Control System
C&DH: Command & Data Handling
GNC: Guidance Navigation and Control

Software Architectural Economy (1 of 2)

- In addition to the FSW challenges, architecting sotware has general challenges that should be considered
- Conceptually these challenges can be thought of as a Software Architectural Economy that is graphically shown on the next slide
- The participants (stakeholders) within this economy are Users, Business Leaders and Engineers
- Each participant has has goals and pbjectives that they wish to maximize, i.e. an objective function.



"Systems should be designed with consideration for the user, the system (the IT infrastructure), and the business goals." [1]

- Users are the FSW customers listed earlier
- Engineers design/implement the technical system
- Business Leaders define and fund business goals

"Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while <u>optimizing common quality attributes</u> such as performance, security, and manageability." [1]

"Architectural Economics" – Each stakeholder strives to maximize their own objective function

1. Microsoft Application Architecture Guide, 2nd Edition, https://msdn.microsoft.com/en-us/library/ee658098.aspx

Architectural Economics Example – Mission X (1 of 2)

- **Original Mission X Ops Concept had continuous downlink and a ~15-minute uplink per 90-minute orbit**
  - Science data and spacecraft engineering data collected in separate 5-minute data files
  - Data must be distributed to global science data repo within 90 minutes of onboard sampling
- **Mission X uses CCSDS File Deliver Protocol(CFDP) Class 2 for file transfers**
  - Complete files can be transferred to the ground without an uplink
  - Autonomous onboard file deletion occurs after final ACK or NACK (retransmission of missing blocks) received during an uplink
- **Original file management strategy**
  - Continuously start transactions and downlink files regardless of whether uplink available
  - At the start of each uplink, ~36 file transferred need completion ACKs and onboard file deletion
  - Several meetings and reviews occurred where the downsides of this strategy were discussed and engineers and FSW system testers lobbied for a change
  - Business leaders insisted 2 uplinks per orbit could not be funded

Architectural Economics Example – Mission X (2 of 2)

- After the first mission sim, operators complained loudly about confusion caused

- About six months prior to launch money was “found” for a second contact and all file transfers occurred during a contact period

Where to Next?

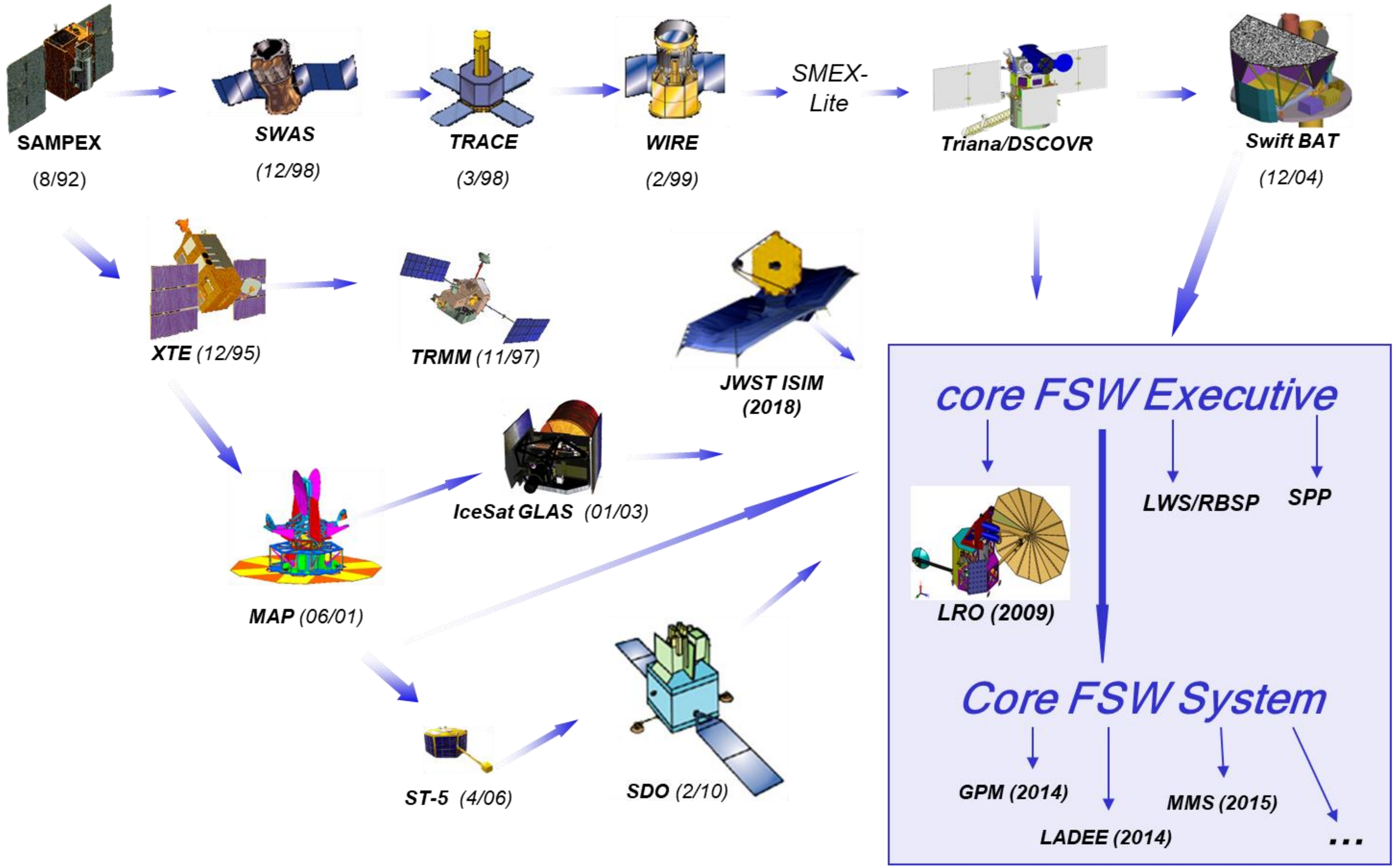
- The next section introduces the cFS architecture and the business model that manages and releases the architectural components

- The Basecamp cFS Framework slide deck describes architectural technical details

Core Flight System Introduction

Core Flight System Motivation

For decades the NASA Goddard Space Flight Center has been designing, building, and operating spacecraft customized for specific science goals



cFS Architectural Goals

- **A hertitage analysis was performed to identify heritage mission requirement/design commonality and variability**

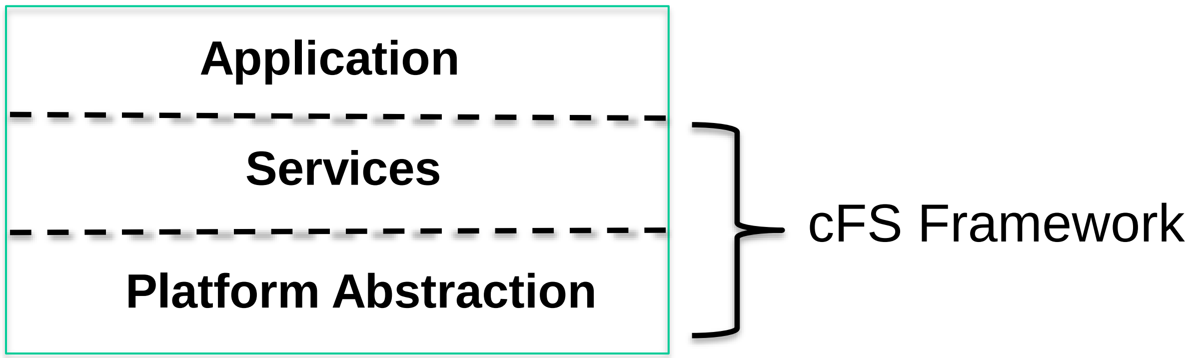
- **Software quality attributes[1] were analyzed and prioritized (see Appendix B)**
  - Provided as input for technical trade decisions
  - Often challenging because of the different FSW customer(user) goals
  - Consider the FSW maintainability attribute requiring the ability to observe and patch any memory location versus the reliability attribute for preventing a software exception/reset

- **These efforts resulted in a project-independent FSW product line**
  - Architectural components are selected, customed, tuned and integrated into an operational system

1. Software quality attributes help to measure the quality of the software from different perspectives

Core Flight System Architecture

- **The cFS uses a three-tiered software architecture that provides a <u>portable</u> and <u>extendable</u> framework with a <u>product line deployment model</u>**
  - Platform Abstraction Layer supports portability
  - The Service Layer provides an Application Programming Interface (API) enabling application reuse across missions
  - Compile-time configuration parameters and run-time command/table defined parameters support adaptability and scalability



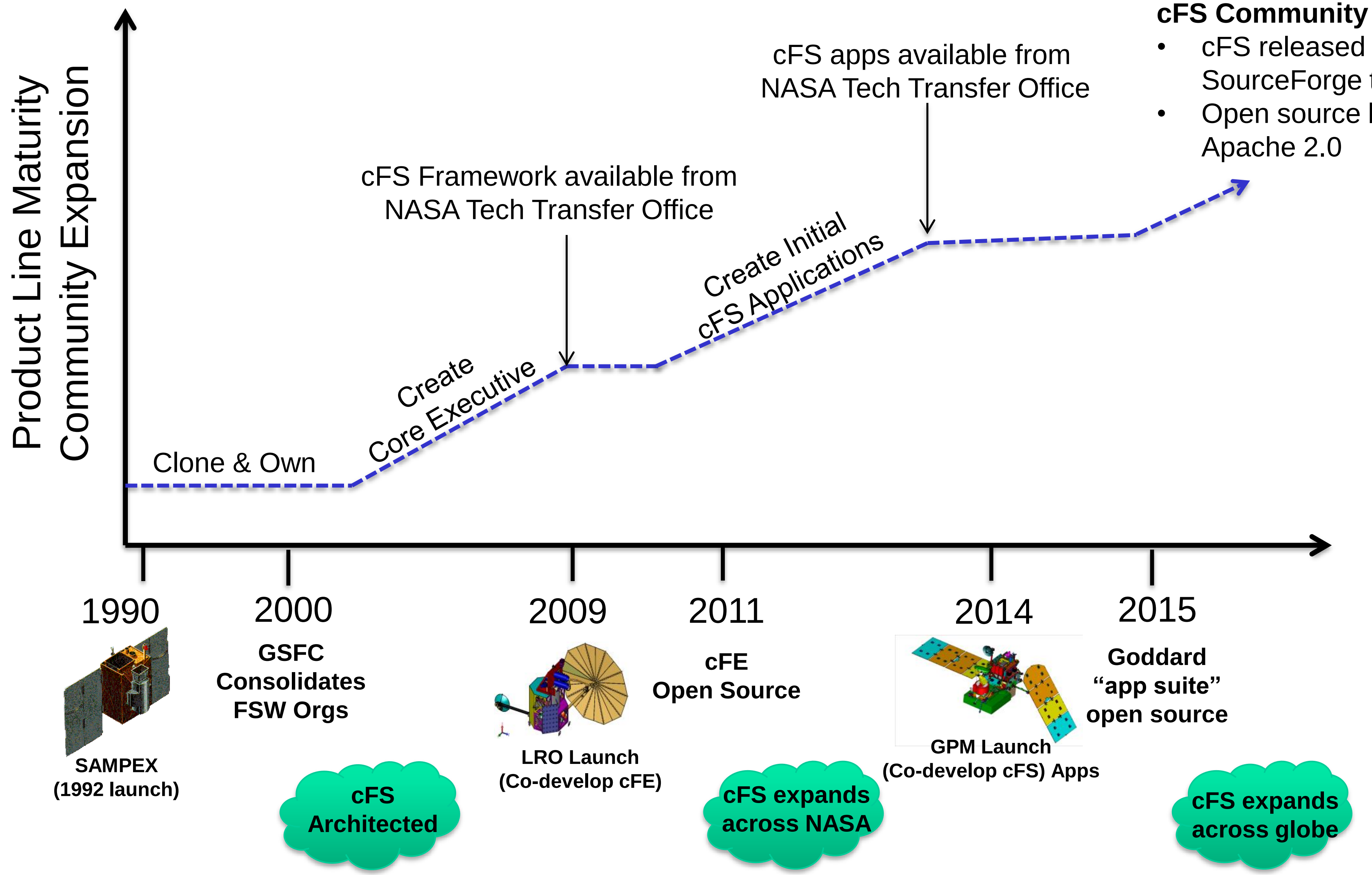
cFS Benefits

- **The cFS Framework and reusable app suite designs are based on lessons learned over decades of experience**
  - These lessons include the entire spacecraft lifecycle that includes development, operations, and maintenance
  - This perspective is very important because each phase may impose design requirements and/or constraints that are not always complimentary

- **The cFS component designs and parameterization address many of the FSW challenges previously listed, for example…**
  - The cFS framework provides services to preserve data across process resets
  - Mission functionality is distributed across apps that each have their own execution thread and priority
  - Message filter tables allow telemetry downlink content and rates to be tuned for specific testing and operational use cases

- **FSW engineering is a systems engineering endeavor because the FSW is integrating multiple subsystems into a single system that must operate as a single system. In order to design the best cFS system, or any FSW system, a FSW expert should be involved from early formulation stages to…**
  - Participate in trades: ground/flight, hardware/software, build/buy, budget options, etc.
  - Champion requirements, design, and interface features with knowledge of all of the customers and spacecraft lifecycle needs

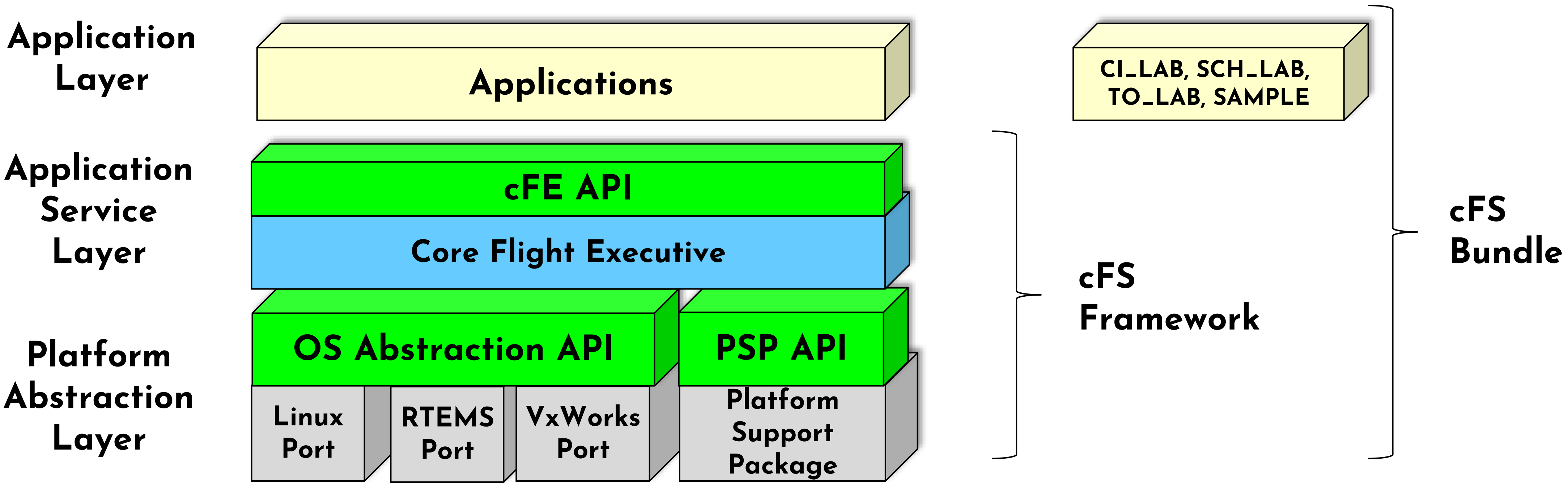
cFS Historical Timeline



Establish & Sustain Collaborative cFS Community

- cFS released on public servers, SourceForge then GitHub
- Open source license changed to Apache 2.0

cFS Layered Architecture



API: Application Programming Interface
OS: Operating System
PSP: Platform Support Package

Key Definitions

Framework

– The set of individual services, applications, tools, and infrastructure managed by the open source community Configuration Control Board (CCB).

Bundle

– An executable version of the framework configured for a nominal Linux system. Links compatible versions of the framework elements as a recommended starting point for new cFS-based systems.

Component

– An individual application, service, or tool that can be used in a cFS-based system

Distribution

– A set of custom components packaged together with the framework; generally created and provided by a cFS user (individual or group) with specific needs (e.g. a NASA center, the GSFC SmallSat Project Office)

cFE vs cFS

– cFE is the Core Flight Executive services and API – cFS is a general collective term for the framework and the growing set of components

Application Layer Benefits

- **Write once run anywhere the cFS framework has been deployed**

- **NASA Goddard has released 15 applications that provide common command and data handling functionality such as**
  - Stored command management and execution
  - Onboard data storage file management

- **Reduce project cost and schedule risks**
  - High quality flight heritage applications
  - Focus resources on mission-specific functionality

- **Framework provides seamless application transition from technology efforts to flight projects**

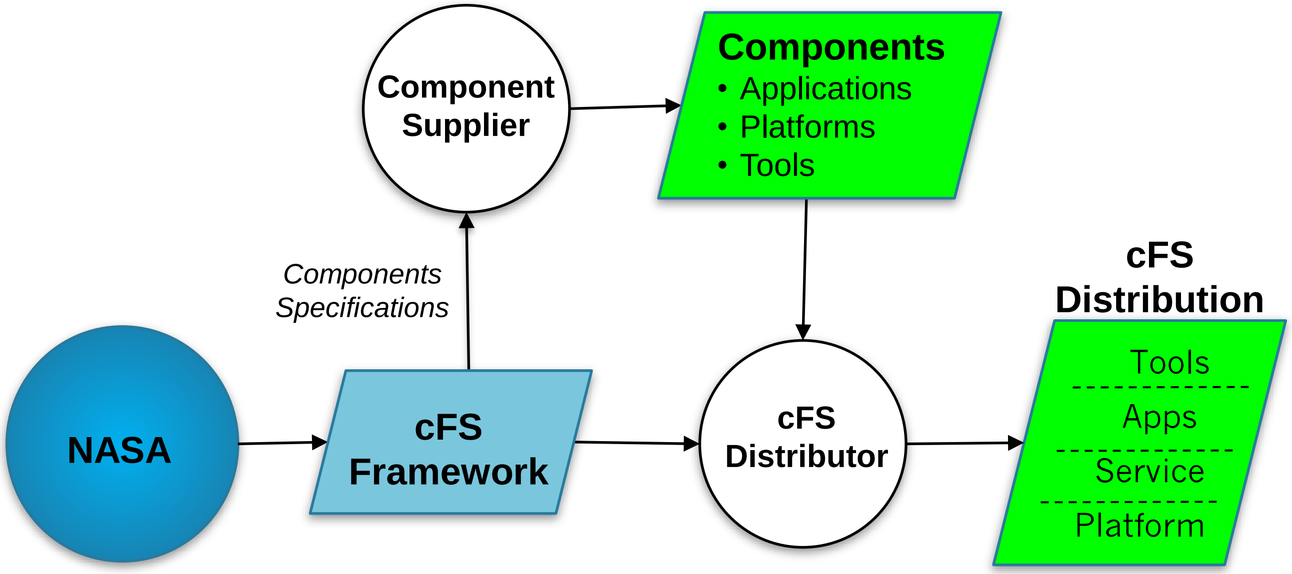
Current State of cFS Open Source Program

- **cFS Framework funding primarily comes from projects**
  - In 2023 the Artemis Gateway program is the primary funding (cFS is in their interoperability spec)
  - cFS strategic decisions are driven by NASA projects which may or may not align with the open source community's priorities
- **A NASA multi-center Configuration Control Board (CCB) manages releases of the open source cFS Framework**
  - The CCB works with projects to determine which issues and changes are in each release
- **Cumbersome cFS Framework release process (discussed later)**
- **Applications are maintained by the organization that created the apps**
  - Currently NASA Ames, Goddard, and Johnson maintain individual app repositories
  - The organization is responsible for funding and for maintaining compatibility with different cFS Framework versions
  - Other organizations have created apps that are included in a cFS Distribution (discussed later)

cFS Open Source Community Challenges

- **Hard for organizations to adopt the cFS**
  - Steep learning curve with some online training material but no courses offered
  - Requires highly motivated teams to train themselves
  - On small teams, there may only be one cFS expert so they become critical to the project's success
- **Successfully using the cFS on a mission requires disciplined systems engineering**
  - Most of the available online resources do not address cFS systems engineering
  - The cFS repo list available components and distributions but it's up to the user to figure which components they may need and how to use them
  - Apps often work together to achieve goals and implement functional requirements
    - The reusable apps are documented as separate components so it is often unclear how the apps can be used collectively
- **Github discussions are the primary technical support mechanism**
  - They're helpful but are responses are not always timely
- **Some artifacts were developed for Goddard-specific ground systems and have not been redesigned**
  - Table tools – Generate binary table files and display binary table files
  - "Build Test" scripts that verify an app's requirements in an operational environment. These have been replaced by "Unit Tests".

Conceptual cFS Product Model



- **Organizations, not just NASA, supply components (applications, platforms and tools) and cFS Distributions**
  - What would motivate an organization to be a supplier or distributor?
  - Would organizations spend resources on assets given the current NASA cFS Framework strategic path is driven by NASA projects?
- **One component supplier value proposition**
  - As the number of supported platforms increases then apps become more valuable
  - As the number of apps increases then supporting a cFS platform become more valuable

What Components and Distributions have been produced?

- **2019: Vendors start to offer processor boards integrated with the cFS**
  - AI Tech partnered with Embedded Flight Systems to offer the cFS integrated on the SP0-S Single Board Computer
  - Genesis Engineering developing an integrated GEN6000 (SpaceCube 2.0) cFS product
  - Genesis pursuing a Space Act Agreement (SAA) that would include the creation of a platform certification test suite
  - I am not aware of anything since 2019
- **2023: cFS Basecamp supports an "app store" model for cFS target integration**
  - Developed for cFS-based educational courses and projects
  - Long term goal is to provide an app migration path from cFS Basecamp to flight missions
  - Relies on a NASA cFS Electronic Data Sheet toolchain that is not part of the official cFS Framework
  - Currently does not use any NASA apps
- **2024: Github "Awesome core Flight System" list of cFS links, apps, tools, documentation, training, misions/projects**
  - mbacch/awesome-cfs: A curated list of resources for NASA's core Flight System (cFS) (github.com)
  - Individual apps produced by different organizations are listed, however system integration is up to each individual (cFS distributions are listed on the next slide)
- **A list of operating system ports is maintained at https://github.com/cfs-tools/cfs-platform-list**

cFS Distributions

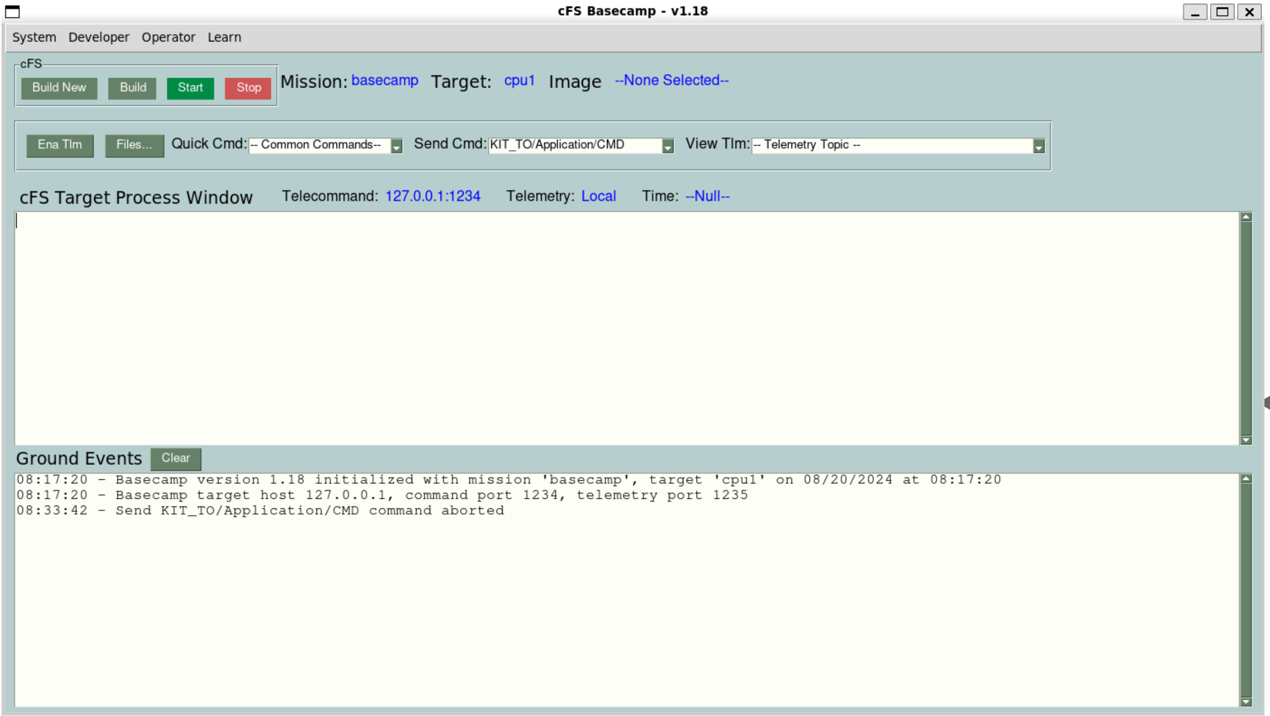
Name/Link	Provider	Status	Description
cFS Basecamp	Open STEMware	Use a cFS tech branch that is not current with the latest cFS Framework.	Basecamp provides a lightweight environment to help you learn NASA's core Flight System (cFS) and create app-based solutions for your projects. Basecamp's default cFS target runs on Linux and includes an app suite that provides a complete operational environment including support for onboard file management and transferring files between the ground and flight systems.
cFS Bundle	NASA GSFC	Current with latest cFS Framework	Contains the cFS Framework packaged with additional components to create a system that can easily be built, executed, and unit tested on a Linux platform. The bundle is not fully verified as an operational system, and is provided as a starting point vs an end product.
cFS Framework-101	NASA MSFC	Functional but outdated. Last updates in 2019.	A training tool for individuals to learn how to develop software with NASA-developed Core Flight software (CFS) framework.
NASA Operational Simulator for Small Satellites (NOS3)	NASA IV&V	Not current with latest cFS Framework	NOS3 provides a complete cFS system designed to support satellite flight software development throughout the project life cycle. It includes •42 Spacecraft dynamics and visualization, NASA GSFC • cFS – core Flight System, NASA GSFC • COSMOS – Ball Aerospace • ITC Common – Loggers and developer tools, NASA IV&V ITC • NOS Engine – Middleware bus simulator, NASA IV&V ITC
OpenSatKit (OSK)	Open STEMware	Runs on Ubuntu 18.04 and earlier. No longer supported	OSK provides a complete cFS system to simplify the cFS learning curve, cFS deployment, and application development. The kit combines three open source tools to achieve these goals: • cFS – core Flight System, NASA GSFC • COSMOS – command and control platform for embedded systems, Ball Aerospace • 42 dynamic simulator, NASA GSFC

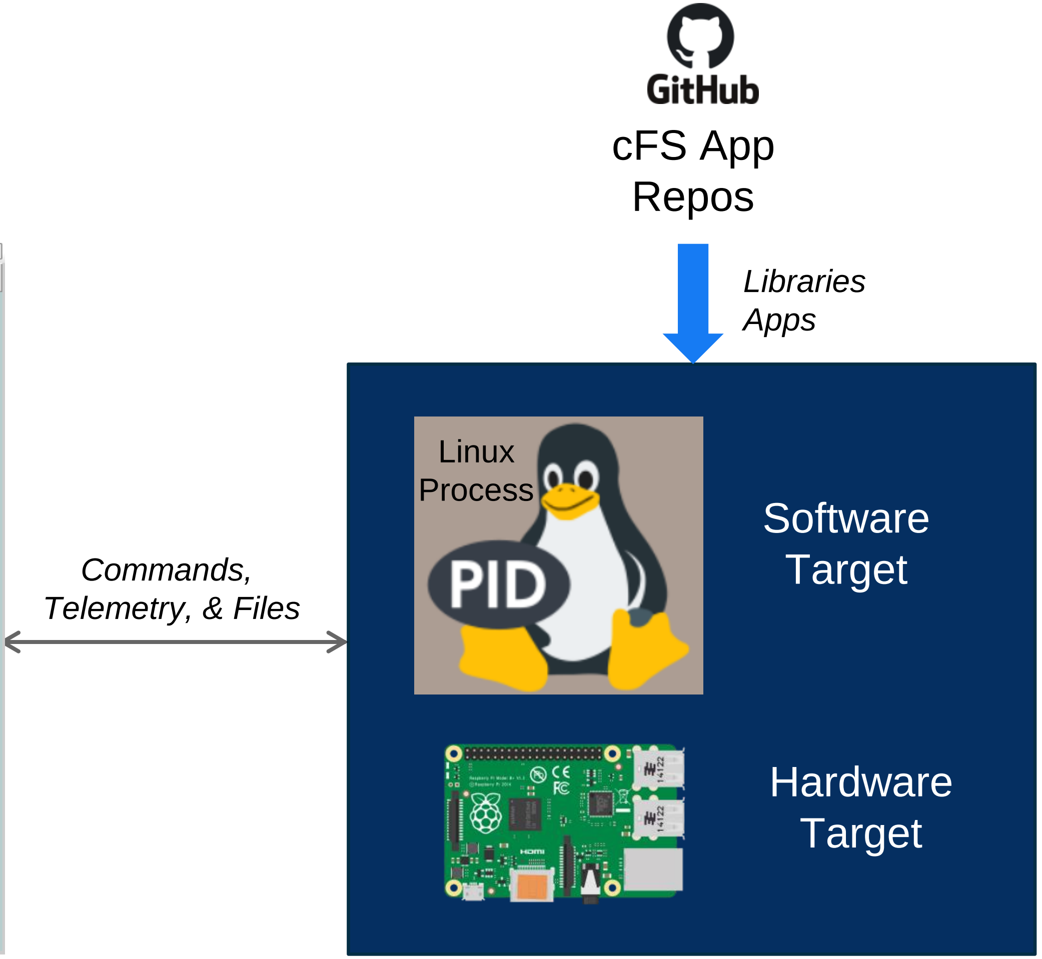
cFS Basecamp (1 of 2)

- **cFS educational platform designed to address the issues on the earlier *"cFS Open Source Community Challenges"* slide**
- **Default cFS target includes a fully functional app suite including a file transfer app**
- **Documentation and tutorials include hands-on exercises for an immersive learning experience**
  - Developers can learn the cFS Framework and app development basics with the built-in resources
- **Goal-oriented projects take developers to the next level of cFS development**
  - Online educational resources coupled with Basecamp's App Store allow developers to create projects designed to teach advanced topics
- **[future] Automated transition from Basecamp to OpenC3 ground system**
  - CCSDS SOIS EDS Tools and Library https://github.com/nasa/EdsLib provide automated EDS-to-OpenC3 conversions, but they are not integrated with NASA cFS repo

- Send commands, display telemetry, and transfer files between the lightweight Python graphical user interface and a cFS Target
- Supports local and remote cFS targets

Python GUI





Concluding Remarks: Version Control

- **A NASA multi-center board controls cFS Framework releases**
  - “Official Releases” have gone through NASA Goddard’s technology release process
- **Release Candidates (RCs) are verified software versions including updated supporting artifacts**
  - Many projects use RCs because they are stable and reliable, however each organization./project must evaluate their particular situation
  - Sometimes they are referred to as “Checkpoints”
  - All RCs are submitted to NASA Goddard’s technology release organization and they determine which RCs become an official releases
  - Historically the technology release process can be lengthy (greater than a year) so the RC process was created to allow the next releases to be developed without waiting for an official release
- **Versions Control**
  - In theory "breaking" changes are indicated by major version number updates (6.9.1 to 7.0.0)
  - A best-effort attempt is made to keep changes within a major version number backwards compatible
  - Sometimes changes sneak in where using open source cFS components across minor versions or RC's does break (using Draco-rc1 osal with Draco-rc4 cfe might not actually work), but the impacts to external apps should be at most minimal
  - Occasionally custom config a project may be using can break if they do something unanticipated or uncovered by open source CI, but again best effort is made to keep things backwards compatible until there's a major version number update
- **Feature Deprecation Process**
  - Mark feature as deprecated on any release
  - Provide tools/process that will warning applications when a feature is marked as deprecated
  - Only deprecate on major versions

Concluding Remarks: User Responsibilities

- **User Contributions**
  - Community Contribution process and Contributor License Agreement (CLA)

- **The cFS Framework has a NASA NPR-7150.2C Class E classification**
  - Class states Design Concept, Research, Technology and General-Purpose Software
  - The cFS Framework provides artifacts to support Class B missions and a subset of artifacts to support Class A missions
  - End-users are responsible for classifying the software system that uses the cFS Framework
  - This is consistent with the Apache license's "Disclaimer of Warranty" that states, *"…provides the Work…on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of … FITNESS FOR A PARTICULAR PURPOSE."*

- **End-users are responsible for complying with International Traffic in arms Regulations (ITAR)**

Appendix A

Acronyms

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
I/F	Interface
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

Appendix B

cFS Architectural Goals

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 (On-Orbit FSW modifications) for long duration missions
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

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 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

Appendix C

Online Resources

cFS Resources

- cFE Framework, http://github.com/nasa/cFE
  - Contains the OSALs, PSPs and cFE that are managed and released by the NASA CCB
- cFS Bundle (cFE Framework), http://github.com/nasa/cFS
  - Contains the cFE Framework packaged with additional components to create a system that can be built, run and unit tested on a Linux platform
  - Website also contains a list of apps, tools and distributions
    - Other components may exist
- cFS Apps, https://github.com/nasa/**
  - "**" is the app abbreviation such as FM for File Manager
- cFS External Code Interface (ECI) library, https://github.com/nasa/ECI

cFS Basecamp Resources

- cFS Basecamp, https://github.com/cfs-tools/cfs-basecamp
- cFS Basecamp App Repositories, https://github.com/cfs-apps
- cFE EDS Framework Toolchain, https://github.com/jphickey/cfe-eds-framework
- Open Mission Stack FSW Learning Resources, https://openmissionstack.com/
- YouTube Educational Videos, https://www.youtube.com/@OpenSTEMware
- OpenSatKit, https://github.com/OpenSatKit/OpenSatKit/wiki
    - Combines cFS, COSMOS Ground System, https://cosmosrb.com/, and NASA 42 Simulator, https://github.com/ericstoneking/42
    - Nolonger supported