Lessons from 30 Years of Flight Software

David McComas
NASA Goddard Space Flight Center
Software Engineering Division
Flight Software Systems Branch

Finding the Best Practices in R&D Software

- What is R&D?
  - R&D is developing new solutions for a specific problem domain
- Flight software at Goddard has two relationships with R&D
  - Part of the spacecraft and instruments that serve as tools for scientists to perform their R&D
  - Advance the state of technology in components with embedded processors and in the software technology itself
- This presentation looks back on 30 years of flight software development at the Goddard Space Flight Center observing trends and capturing lessons learned

My 30 Year Perspective

29 Years

Flight Projects

1.5 Years

Line Management

NASA Perspective
Answering the “Big” questions

- **Earth**
  - How is the global earth system changing?
  - How will the Earth system change in the future?
- **Heliophysics**
  - What causes the sun to vary?
  - How do the Earth and Heliosphere respond?
  - What are the impacts on humanity?
- **Planets**
  - How did the sun's family of planets and minor bodies originate?
  - How did the solar system evolve to its current diverse state?
  - How did life begin and evolve on Earth, and has it evolved elsewhere in the Solar System?
  - What are the characteristics of the Solar System that lead to the origins of life?
- **Astrophysics**
  - How does the Universe Work?
  - How did we get here?
  - Are we alone?

To help answer these questions NASA builds spacecraft, vehicles and instruments with lots of software on board

Goddard Perspective

NASA's Goddard Space Flight Center in Greenbelt, Maryland, is home to the nation's largest organization of scientists, engineers and technologists who build spacecraft, instruments and new technology to study Earth, the sun, our solar system and the universe.

Fight Software Perspective

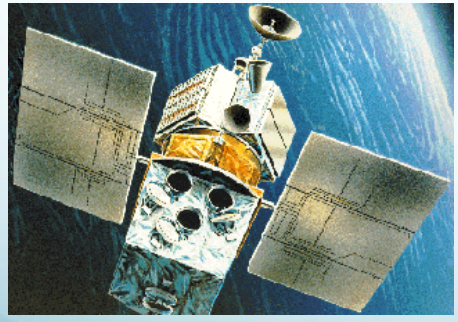
- The Flight Software Systems Branch provides on-board, embedded software products that enable spacecraft hardware, science instruments and flight components to operate as an integrated on-orbit science observatory.
- Embedded software complicates the software development lifecycle
- High fidelity simulators are critical to verify the flight software prior to flight hardware integration
- High reliability and fault tolerant to ensure spacecraft safety even with a failure
- Deterministic real-time performance required for attitude control, high speed data, etc.
- Remote maintenance and system updates
  - Subsystems fail over time (mission lifetimes are 3-20 years or more)
- Autonomous operations (i.e. In a broad sense, every spacecraft is a robot)

Flight Hardware Constraints

- Harsh space and launch environments as well as power, size and weight limitations constrain hardware options.
  - Minimize **S**ize, **W**eight, **a**nd **P**ower (SWaP)
  - Power increases cause mass and size increases
  - Radiation tolerant hardware
- Speed of processor
  - Example: LRO and Curiosity use a 166 MHz processor , my laptop uses 2.5 GHz processor
- Memory and storage
  - LRO has 2MB of instruction memory, my laptop has 4GB of RAM for instructions and data
  - Curiosity has 2GB of Flash and 256 MB DRAM (Huge by spacecraft norms and needed a RTG)

My First Spacecraft

- **Hubble Space Telescope (HST)**
  - Launched on April 24, 1990 by Space Shuttle Discovery from Kennedy Space Center
  - Telescope observes in the near ultraviolet, visible, and near infrared spectra
  - Operation for 25 years and counting



- **Extreme Ultraviolet Explorer (EUVE)**
  - Launched on June 7, 1992 on a Delta II rocket from Cape Canaveral
  - Perform an all-sky survey in the extreme ultraviolet band
  - Observe EUV sources such as white dwarfs and coronal stars

1985 Unconstrained Commercial Technology

- 1984 Sony introduced the discman to replace the tape-based walkman
- 1984 – Nokia introduced the world's first portable phone weighing in at 11lbs and requiring a car to charge the batteries.
- Macintosh, Windows 1.0, Amiga 1000, Commodore 128 and Consumer reports citing the "mouse" and "icons" as major advancements

HST & EUVE FSW Development

- EUVE Spacecraft bus based on the Multi-mission Modular Spacecraft (MMS) architecture
  - NASA Standard Spacecraft Computer (NSSC-i)
    - 18-bit data words with 64K of core memory, 33 instructions Fixed point arithmetic, 1Mhz clock
  - MIL-STD-1750A Co-processor
  - 64K of 32-bit word shared memory between the processors
- HST used the NSSC-I with a DF-224 co-processor with 32K 24-bit words available at one time
- Custom NSSC-I assembler and debugger
- Custom real-time operating system
- For NSSC-I development we counted CPU cycles for functions
- EUVE's NSSC-I FSW exceeded available memory
  - Default launch image included onetime autonomous deployment app that was replaced after deployments
- HST certified the lab prior to each acceptance test run

My last Spacecraft - GPM

- Global Precipitation Measurement (GPM) Launched on February 27, 2014 on an H-IIA rocket from Tanegashima Space Center
- Follow on to the Tropical Rainfall Measurement Mission (TRMM) that moderate and heavy rainfall in the tropics
  - TRMM lasted ~17.5 years, Launched 11/27/97 and deactivated 4/9/15
- GPM extends TRMM's measurements by measuring light rain and falling snow in middle and high latitudes which account for significant fractions of precipitation occurrences

GPM FSW Development

- RAD750 Processor based on the commercial PowerPC 750 operating at 132 MHz with 36MB RAM, 4MB EEPROM, and 4GB of Solid State Recorder memory
- Open Source GNU compiler and debugger
- Commercial VxWorks Operating System
- GPM did not use code generation but other contemporary in house missions are using it
- cFS core, Co-developed cFS applications

1985 - 2015

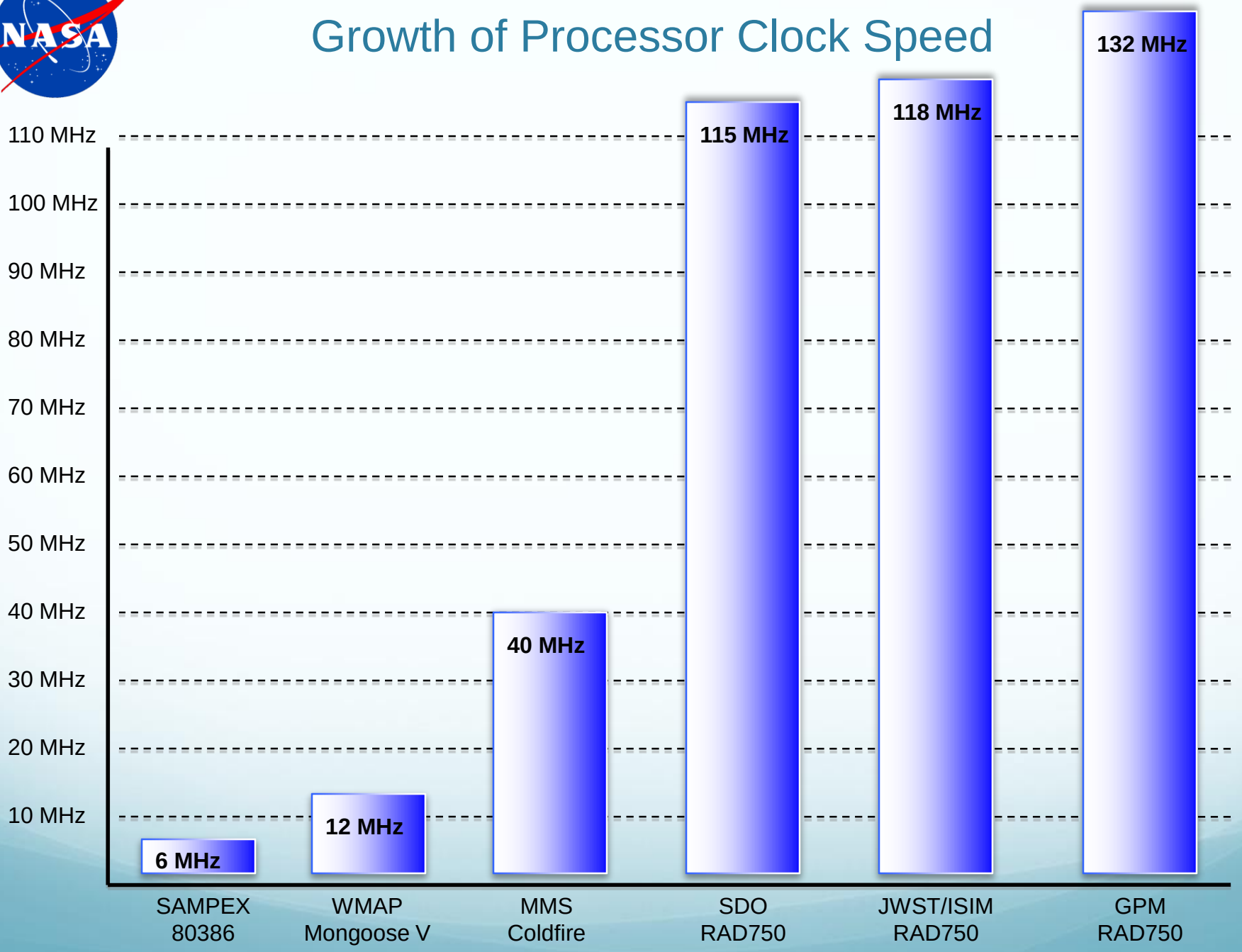
Trends

Recent and Upcoming Processors

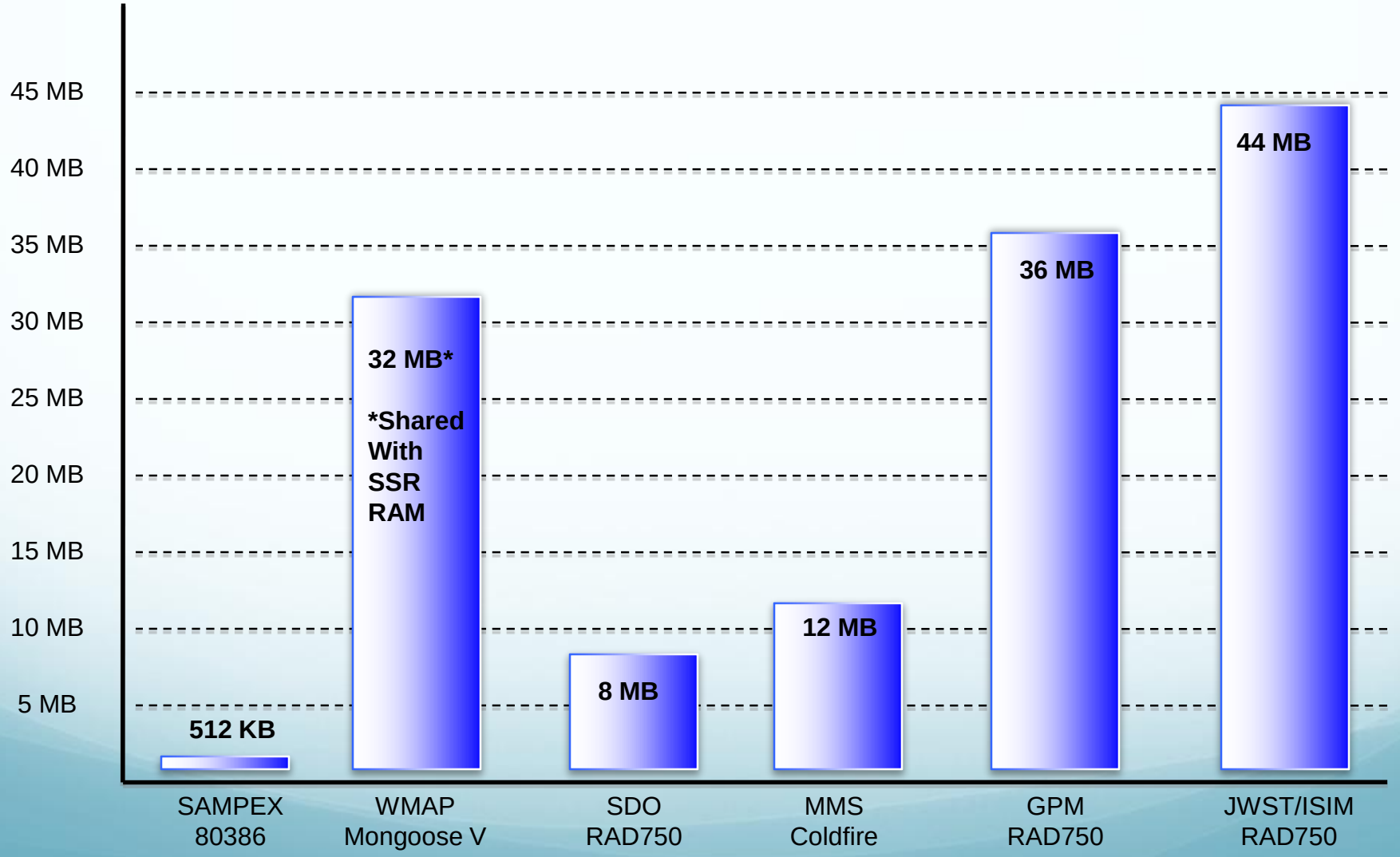
Mission	Processor	Clock Speed	EEPROM	Local RAM	SSR RAM
SAMPEX	80386	6 MHz	256 KB (1 bank)	512 Kbytes	48 Mbytes
MAP	Mongoose V	12 MHz	2 MB (2 banks)	32 MB shared	224 MB shared
LRO	RAD750	132 MHz	4 MB (2 banks)	36 MB	16 GB
SDO	RAD750	115 MHz	4 MB (1 bank)	8 MB	128 MB
GPM	RAD750	132 MHz	4 MB (2 banks)	36 MB	4 GB
MMS	Coldfire	40/20 MHz	4 MB (2 banks)	12 MB	600 MB
JWST ISIM	RAD750	118 Mhz	4 MB	44 MB	N/A
RNS…	SpaceCube dual core	250 Mhz	512MB flash	256 MB	960 GB (Hard drive)
SpaceCube 2.0	PPC 440 2 To 6 cores	250 Mhz	variable	variable	NA
Maestro (Lite)	Tilera 49 core	300Mhz	variable	variable	NA
	Tilera 16 core	300 Mhz	variable	variable	NA
BAE RAD750 (new)	PPC 750	200 Mhz	variable	variable	NA
Leon3 FT GR712RC	SPARC 8 Dual core	100 Mhz	variable	variable	NA

Like the desktop, most next generation flight processors are multicore

Growth of Processor Clock Speed



Growth of Processor RAM Size



FSW Functional Changes from EUVE to GPM

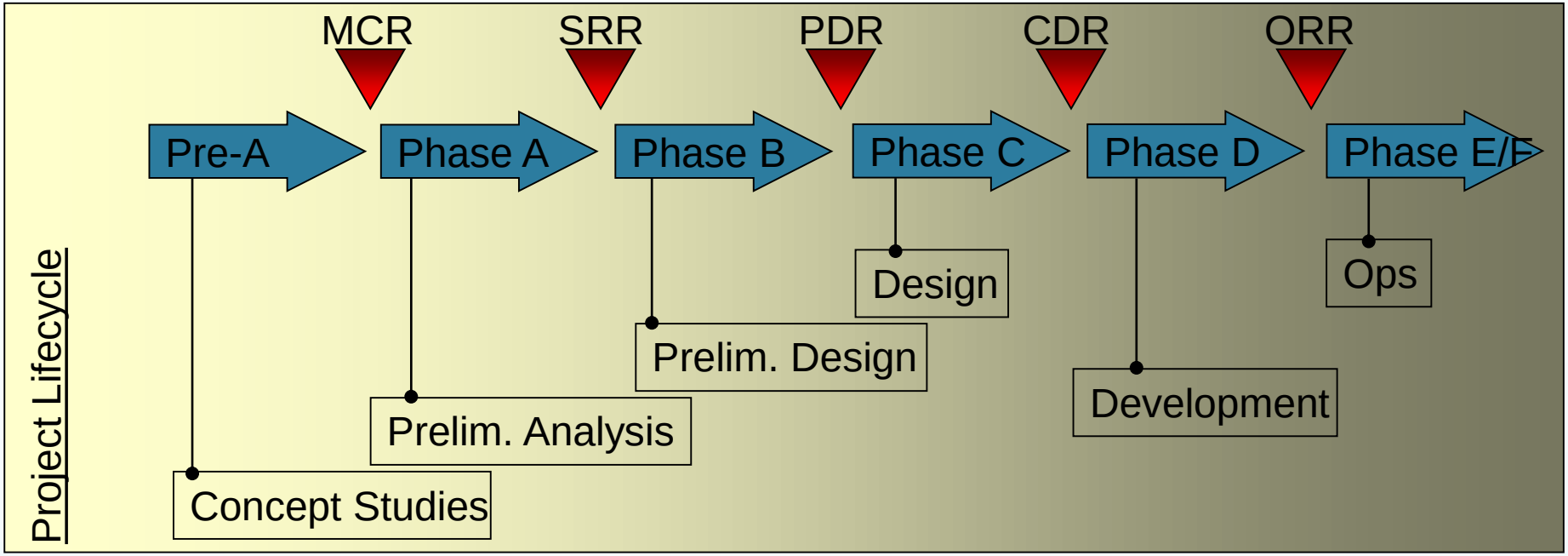
- Spacecraft Ephemeris
  - EUVE: Interpolation using Hermitian Polynomials, required daily coefficient uploads
  - EUVE: Experimental TDRSS Onboard Navigation System (TONS) that measured spacecraft-to-TDRSS signal Doppler shift to determine spacecraft position and velocity
  - Onboard propagator using Runge Kutta integration, required weekly position-velocity vector updates
  - GPM: GPS receiver, requires no operational intervention and minimal FSW data processing
- Star Tracker
  - EUVE: Optical camera and FSW processed images using onboard star catalog
  - GPM: Digital tracker that outputs quaternions
- Telemetry formats
  - EUVE: Time Division Multiplex data streams
  - GPM: CCSDS standard compliant packets
- File System
  - EUVE: None
  - GPM: Onboard file system (FS), custom EEPROM FS, VxWorks RAM FS
  - GPM: CCSDS File Delivery Protocol (CFDP) used for flight-ground file transfers

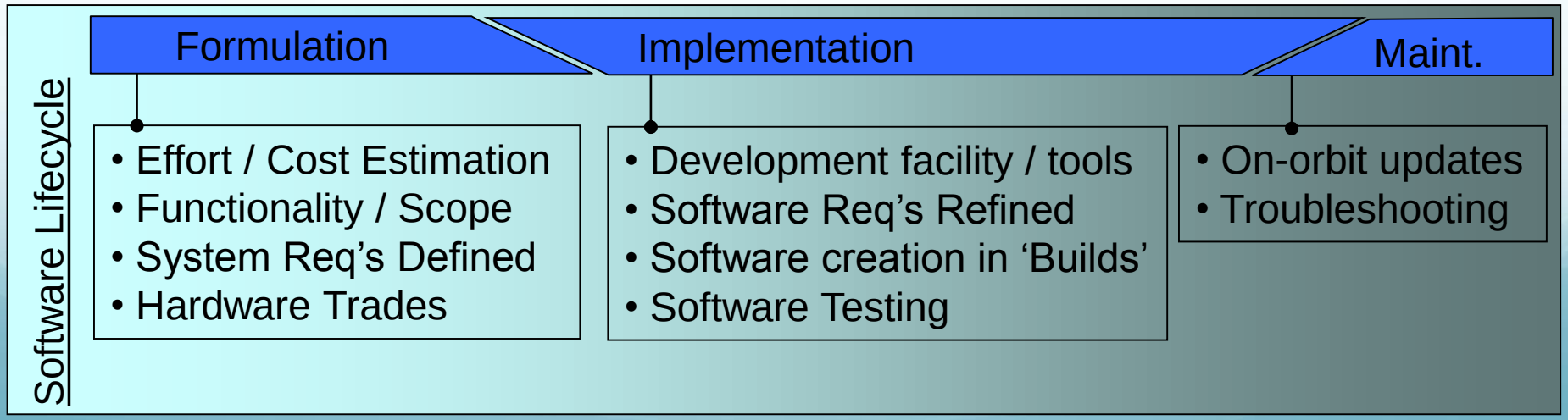
Observations and Lessons

- Using a commercial based processor allows space industry to leverage broad user base and wide variety of tools available in the commercial marketplace.
- More horsepower does not mean you get sloppy on performance.
  - Manage your resource margins
  - Keep experts around
  - Proper diagnostic tools
- More horsepower does mean you can apply good software engineering design principles
  - Design first and optimize second. Should have always done but harder when counting CPU cycles
- Hardware changes over time
  - Use layered software architecture with object oriented components to insulate and encapsulate

Flight Software Development Process

Lifecycle


MCR
SRR
PDR
CDR
ORR
Pre-A
Phase A
Phase B
Phase C
Phase D
Phase E/F
Project Lifecycle
Concept Studies
Prelim. Analysis
Prelim. Design
Design
Development
Ops

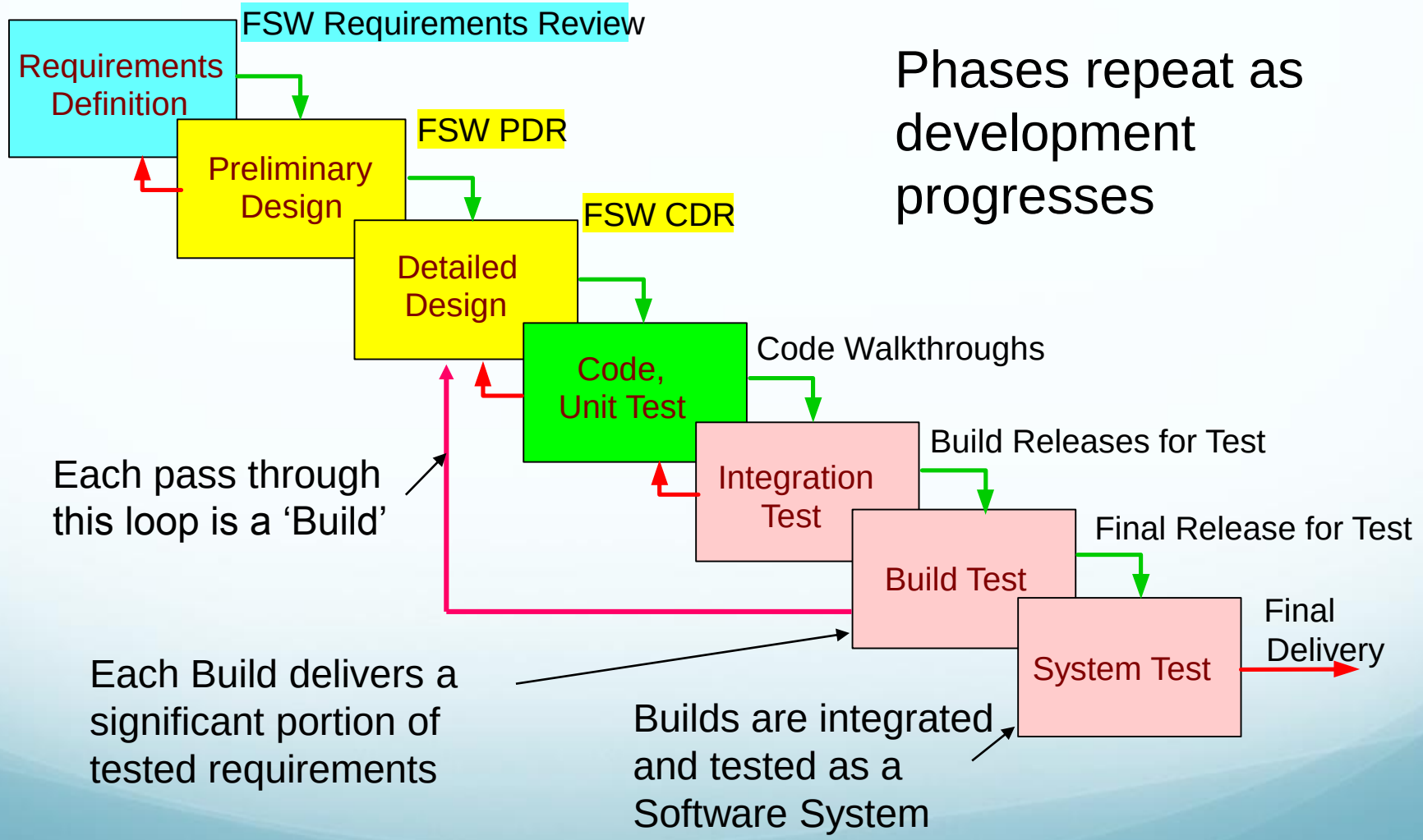

Software Lifecycle
Formulation
Implementation
Maint.
• Effort / Cost Estimation
• Functionality / Scope
• System Req's Defined
• Hardware Trades
• Development facility / tools
• Software Req's Refined
• Software creation in 'Builds'
• Software Testing
• On-orbit updates
• Troubleshooting

Software Costs

- Most Software costs are labor
  - People generating code or testing software
  - Some costs required for hardware and software development tools
    - Compiler seats, software licenses, configuration management software, etc.

- Estimating software costs is not perfect
  - “How long does it take to create the software for 10 requirements?”
  - “How long does it take to test 10 requirements?”
  - Best estimates are a result of keeping good records from previous endeavors

Software Development Process



FSW Branch Templates & Standards

- Based on FSW Lessons Learned
- Avoids ‘Reinventing’ Documents
- Guides Team Leads to Consider all Issues up-front
- Enables Rapid Mission Progress

FSW Development Automation

- Integrated Tools Set aid in:
  - Requirements Management
  - Configuration Management
  - Defect and Change Tracking
  - Test Status Management

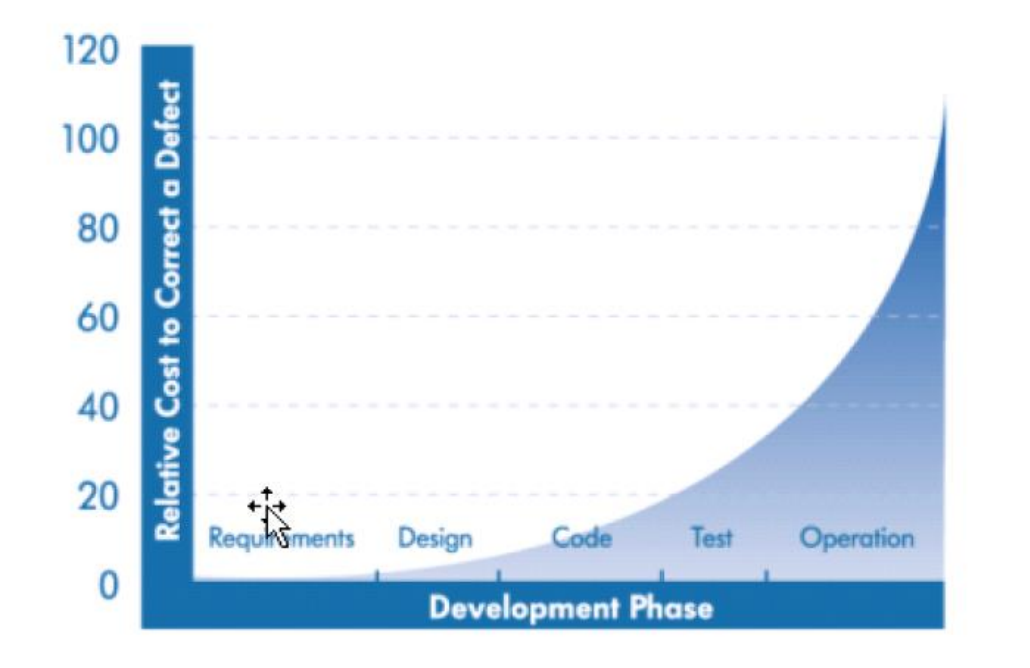
Why Test?

Project Mgr:	**Sally, you look troubled are you ok?**
Sally:	**I'm worried about the software. The more tests I do, the more problems I find.**
Project Mgr:	**Well maybe you should stop testing the software…**

- FSW is a critical element for mission success
  - Software that crashes regularly may lose science data or observation time
  - Incorrect computations can jeopardize spacecraft or payload safety
- The only way to know for sure the software works is to test it in a flight-like environment

Relative Cost of Software Defect

- The best time to find an error in any system is right after it is introduced
  - Finding a problem late in development can be expensive to solve.



Software Engineering Economics (Prentice Hall,1981), Barry Boehm

Test Staff

- Testers develop automated procedures to exercise the software and look for expected output.

- Goal is to find problems before software is used by other subsystems:
  - Need to check performance during boot-up, normal execution and under ‘stress’ conditions
  - Need to verify all software interfaces work as expected
  - Need to verify GN&C algorithms, science processing, power monitoring, etc.

- The procedures are ‘scripts’ written for a ground system environment.
  - Can be re-executed for each build to make sure previous functionality is not lost.
  - Allows for the software procedures to be used at spacecraft integration

Test Staff

- The test procedure development process is a software development effort in itself.
  - Usually about 1/3 to 1/2 the development staff
  - Testers often specialize in specific areas. Examples:
    - GN&C
    - Timing
    - Command handling
    - etc

Test Types (1)

- **Unit Tests**
  - **Performed on non-flight-like hardware, usually by the developer**
  - **Verify Code Logic -- “Does it do what I intended?”**
    - **Exercise full range of inputs and outputs**
    - **Exercise all paths**
- **Integration Tests -- Repeated each build**
  - **Performed on Flight-like Hardware**
  - **Focus on hardware/software interfaces**
  - **Exercise all threads of capabilities in the build**

Test Types (2)

- Build Verification Tests
  - Were FSW Requirements implemented correctly?
- System Validation Tests
  - Does the FSW System meet all Intended On-orbit Operations Capabilities?
- Acceptance Test – a Test Event
  - Does the full set of System Validation Tests execute properly on the final Flight Software Build?

Test Beds

"Test Like You Fly…"

- A flight like test environment emulates hardware and software interfaces
  - Attitude sensors, memory, A2D registers, …
- Allows all nominal data and off-nominal data to be injected at the interfaces
- Not having a flight-like test environment is like:
  - Asking a mechanical engineer to test their hardware with half the launch loads
  - Or, asking an electrical engineer to test their circuits with half the voltage

Test Beds

- Flight-like test beds allow trouble shooting for on-orbit problems in electronics and software without experimenting on the flight hardware.
  - Also allows for trouble shooting during I&T in parallel to other processing.

FSW Testbed Elements

(1) Flight Data System

(2) Simulators & Tools

(3) T&C Ground System

- ASIST
- ITOS

FSW Testbed Requirements

- Exercise FSW on target hardware
  - find any problems before I&T
- Self-documenting FSW Tests
  - FSW Test Results Review/Analysis
- Repeatable Tests and Expected Results
- Enable FSW, Simulator Troubleshooting

FSW Testbed
Flight Electronics Options

- Commercial
  - An electronic system that incorporates the same functionality as the flight system but is built of commercial (off-the-shelf) hardware.
    - Not rad-hard
    - Allows software development team to start working early before more flight-like hardware is available.
- Breadboard
  - An electronic system that uses flight components, but is not processed or packaged in a flight-like manner
    - Example: FPGAs are mounted via sockets to allow reprogramming
- ETU (Engineering Test Unit)
  - An electronic system that uses flight components and is processed and packaged in a flight-like manner

Spacecraft GN&C Simulator

- High fidelity GN&C hardware interfaces
  - Direct I/O or 1553 bus
- Models the on-orbit space environment
  - Disturbances, Orbital relationships, Physics phenomenon
- Models behavior of actuator hardware due to FSW commands
  - Reaction Wheels, Torquer Bars, Thrusters
- Models sensor hardware inputs
  - Gyroscopes, Star Trackers, Sun Sensors, Magnetometers, …

Spacecraft GN&C Simulator (Con't.)

- Configurable for FSW test purposes
  - Synchronize simulator time to FSW time
  - Set initial orbit parameters
  - Set initial GN&C flight hardware statuses
  - Inject on-orbit hardware anomalies

FSW Sustaining Engineering

- What is FSW Sustaining Engineering?
  - Modifying embedded software while it's executing in space.
- Why change FSW in orbit?
  - Compensate for a hardware problems
  - Maintain science program
    - Increase science capabilities
    - Implement new mission requirements
    - Increase mission lifetime
  - Correct FSW bugs
  - Simplify or automate operations

FSW Post-Launch Changes Examples:

- Rossi X-Ray Timing Experiment (RXTE): star trackers lost track on guide stars
  - Just after launch it was determined that the star trackers would drop lock on guide stars intermittently
  - Developed and installed software that modified the star tracker management code to perform a new directed search for guide stars whenever a guide star is lost

- GPM: correct default magnetic torque rod parameters
  - The default magnetic torque rod data processing parameters used by the safehold algorithm were incorrect resulting in increased momentum during safehold
  - EEPROM defaults were changed for operational CPU as well as the cold spare CPU

Observations and Lessons Learned

- FSW expert should be involved from early mission formulation stages
  - Participate in ground/flight trades, hardware/software trades, mission cost estimates
- Formal Development and Test Processes do pay-off
- Detailed FSW Requirements are tremendously critical
  - 'Communicate' exactly what FSW will do
  - Create clear agreement among developers, testers, Systems Engineers, Ground Operators, Hardware subsystem engineers
- Interface Control Documents are critical
  - Must get detailed hardware and software interface definitions in writing and signed-off
- Formal and Informal Review of FSW Requirements, Designs, Code, Test Scenarios, Test Results are all critical
  - FSW specialists, Project Systems Engineers, Hardware Subsystem Analysts, Operations
  - Walkthroughs find errors
  - Formal Reviews (Standup Presentations) facilitate Project level resolution of FSW risks
  - Avoid including design information in the requirements
- High Fidelity FSW Testbed is non-negotiable
  - FSW must execute on flight-like hardware
  - Simulations must accommodate ground validation of FSW
  - Essential for post-launch maintenance of FSW

core Flight System (cFS)

Applied Lessons Learned

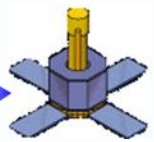
History and Motivation

- Several years ago, Goddard Space Flight Center (GSFC) was tasked two large in-house missions with concurrent development schedules (Solar Dynamics Observatory (SDO), and Global Precipitation Measurement (GPM)
- GSFC was to design and build the spacecraft bus, avionics and flight software and integrate these components with the spacecraft
- Without the staff for both projects and reduced budgets, we needed to find a better way
  - We had about a year to figure it out before staffing up

GSFC Flight Software Heritage

SWAS

(launched 12/98)



TRACE

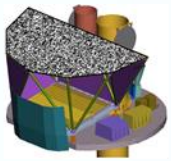
(launched 3/98)

WIRE

(launched 2/99)

SMEX-Lite

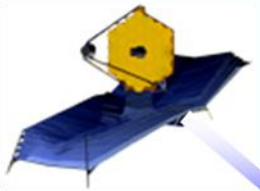
Triana/DSCOVR



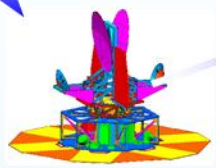
Swift BAT (launched 12/04)

XTE (launched 12/95)

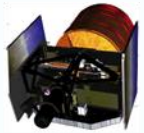
TRMM (launched 11/97)



JWST ISIM



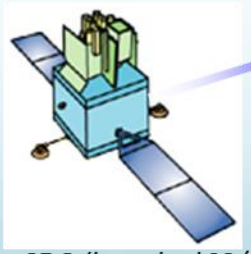
MAP (launched 06/01)



IceSat GLAS

(launched 01/03)

ST-5 (launched 5/06)



SDO (launched 02/10)

Architecture Goals

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

These goals were written in 2006 and have remained essentially unchanged over the years!

History - Re-use in the Past

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

Heritage - What Worked Well

- Message bus
  - All software applications use message passing (internal and external)
  - CCSDS standards for messages (commands and telemetry)
  - Applications were processor agnostic (distributed processing)
- Layering
- Packet based stored commanding (AKA Mission Manager)
- Vehicle FDIR based on commands and telemetry packets
- Table driven applications
- Critical subsystems synchronized to the network schedule
- Clean application interfaces
  - Component based architecture (The Lollipop Diagram)

Heritage - What Worked Well

- Lots of innovation
  - Constant pipeline of new and varied missions
  - Teams worked full life cycle
    - Requirements through launch + 60days
    - Maintenance teams in-house and in contact with engineers early in development
  - Teams keep trying different approaches
    - Rich heritage to draw from
- Keep the little “c” in the architecture
  - A little core framework, as in low footprint, optimized for flight systems
    - Can we fit in a cubesat with 800KB flash and 2MB RAM?

Heritage - What Didn't Work So Well

- Statically configured Message bus and tables
  - Scenario: GN&C needs a new diagnostic packet
  - How do I add a new one on orbit? (FAST mission example)
- Monolithic load (The "Amorphous Blob")
  - Raw memory loads and byte patching needed to keep bandwidth needs down
  - Modeling tools did not support loadable objects
- Reinventing the wheel
  - Mission specific "common" services ("Look , I've got a new and improved version!")
    - Need to "optimize" for each mission
- Application rewrites for different operating systems
- Claims of high reuse, but it still took the same effort on each mission
- Any changes rippled through all the tests, documents and development artifacts
  - All the development artifacts were also clone and own

Key Trades

Architecture Trades: Pub/Sub messaging

- Publish - just send data packets
  - Destination agnostic
  - Components can be configured to limit command sources
- Subscribe - any a component can receive/listen to any packet
- Peer to Peer network
  - No master, stateless
    - Component /node stops and data is un-subscribed automatically
  - Robust/Fault tolerant (No master, GPM network example)
- Ground systems, and simulation applications look like any other component/node
  - External interfaces can be gatewayed and firewalled
- Consultative Committee for Space Data Systems (CCSDS) packet format
  - All the pieces (Identifier, time, sequence number, length) and extensible
  - Works well with our existing ground systems
- Evaluated CCSDS Asynchronous Message Service (AMS) and COTS Network Data Distribution Service (NDDS)

Architecture Trades: File Systems

- No GSFC missions had flown a file system
  - Triana hadn't launched
    - Deep Space Climate Observatory, (DSCOVR) (Launch ~ January 2015)
- File systems are a well supported abstraction for data storage
- Standard file transfer mechanisms (TFTP, FTP, CFDP)
- Operating system support across most vendors
- Lots of resistance to added complexity
  - VxWorks DOS and MER
- Result:
  - Use file for code, data and recorder
  - LRO used VxWorks file system with work arounds (stat example)
  - Looking at JPL file system
    - RAMFS – A Volatile Memory Filesystem
      - POSIX compliant, SPIN® checked
  - Funding RTEMS robust file system work

Architecture Trades: Linking

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

Architecture

Concepts and Standards

- Layered Architecture
- Standard Middleware/Bus
- Standard Application Programmer Interface for a set of core services

} Core Flight Executive (cFE)

- Plug and Play Reusable Applications
- Command & Telemetry database

} cFS Applications

- Reuse Requirements Management
- Reuse Standards
- Reuse Repository

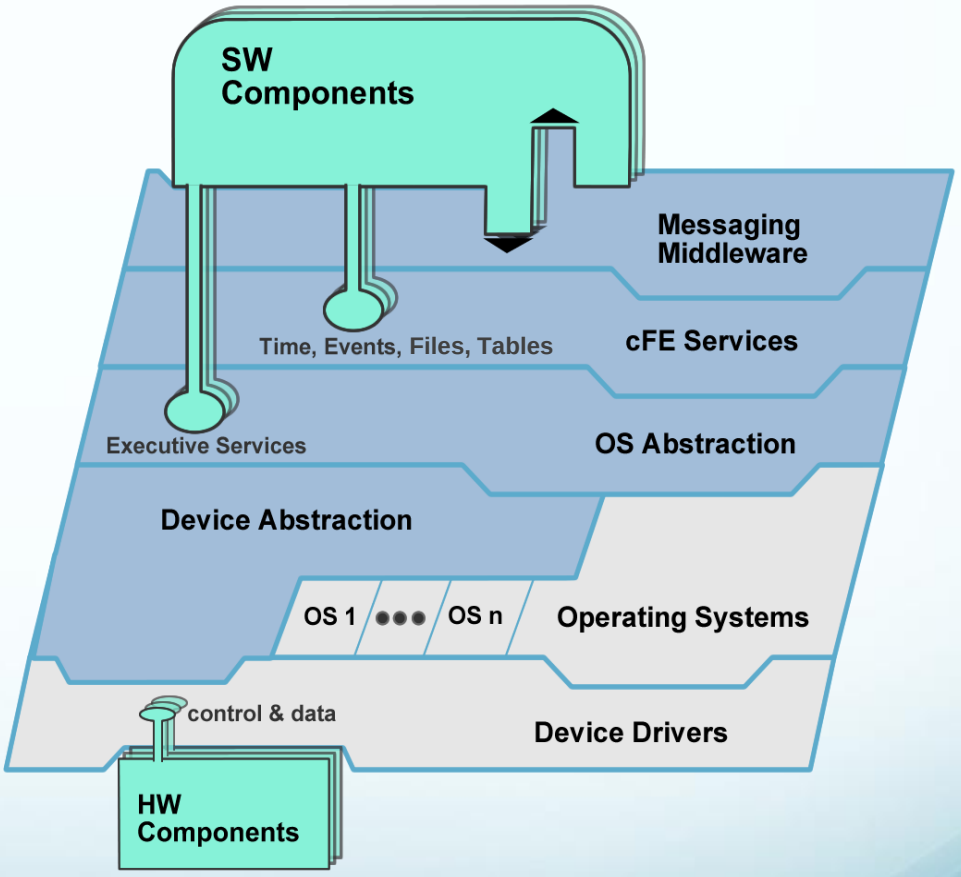
} Library & CM

- Configuration Tool for Mission Users
- Development Tools

} Integrated Development Environment

Layered Services

- 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.



Standard Middleware Bus

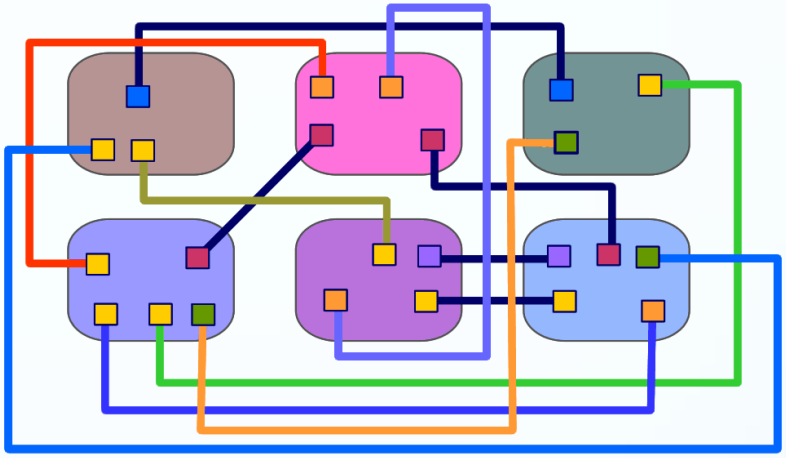
Publish/Subscribe

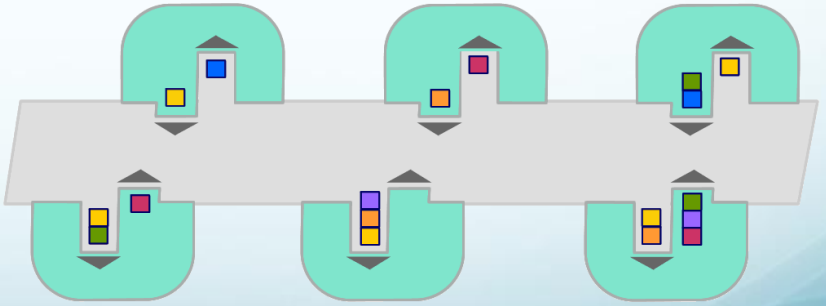
- **Components communicate over a standards-based Message-oriented Middleware/Software Bus**.
- The Middleware/ Software Bus uses a run-time Publish/Subscribe model. Message source has no knowledge of destination.
- No inherent component start up dependencies

Impact:

- Minimizes interdependencies
- Supports HW and SW runtime “plug and play”
- Speeds development and integration.
- Enables dynamic component distribution and interconnection.

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





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

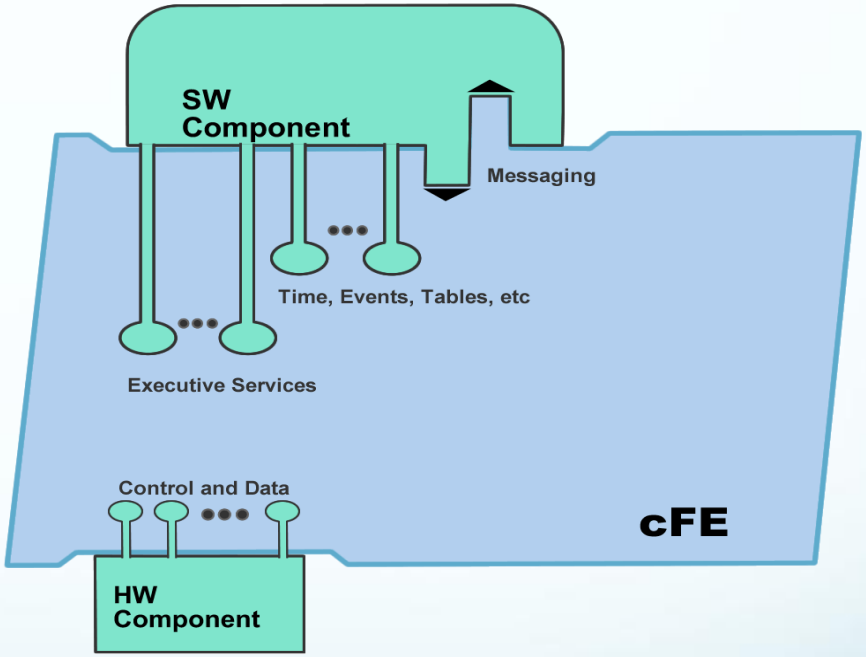
Standard Application Programmer Interface (API)

Application Programmer Interfaces

- cFS services and middleware communication bus has a **standardized, well-documented API**
- An **abstracted HW component API enables standardized interaction** between SW and HW components.

Impact:

- Allows development and testing using **distributed teams**
- With the framework already in place, a**pplications can be started earlier** in the development process
- **Can do early testing and prototyping on desktops and commercial components**
- **Simplifies integration**



API supplies all functions and data components developers need.

cFS Overview

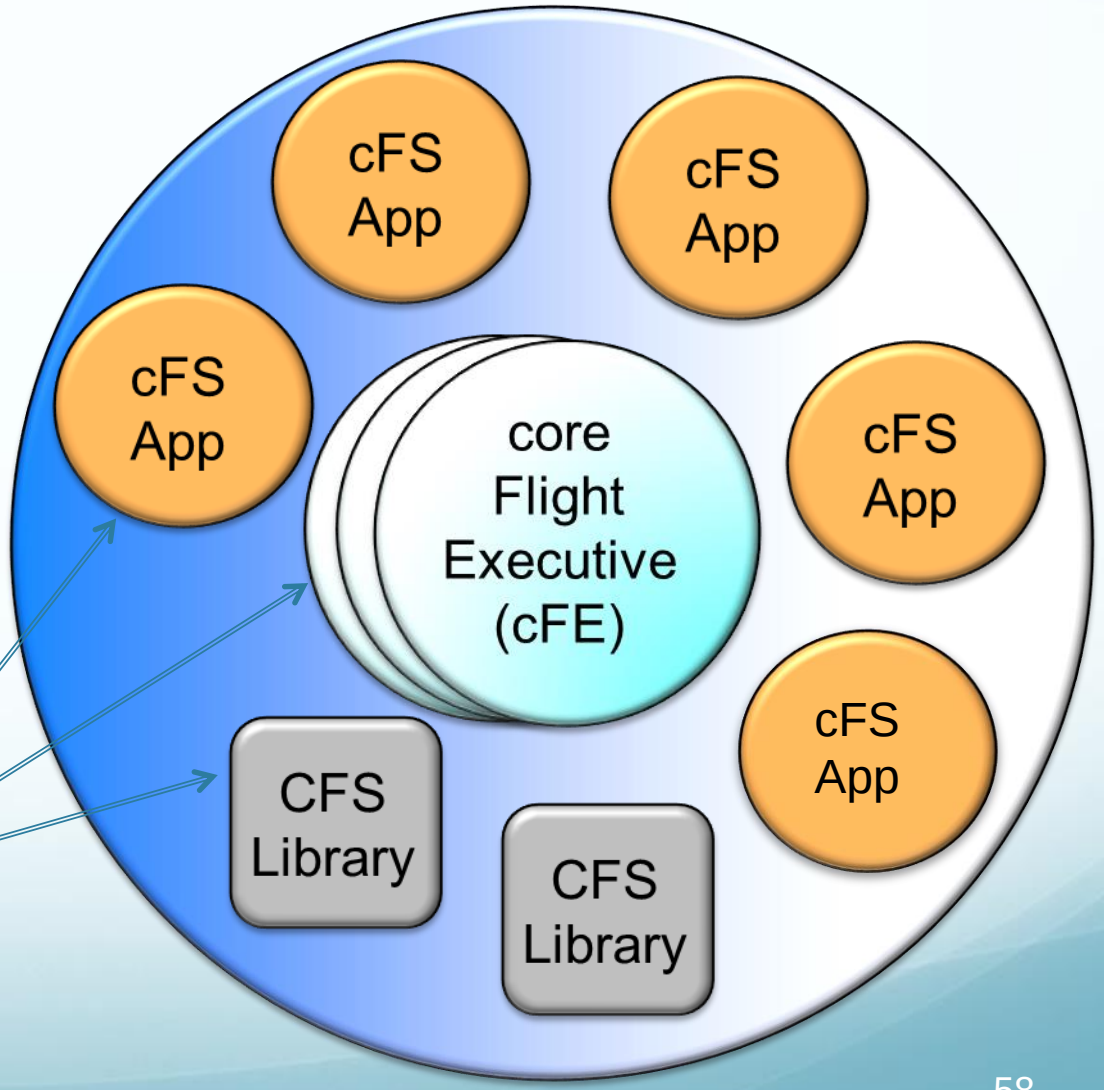
Core Flight System (cFS)

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

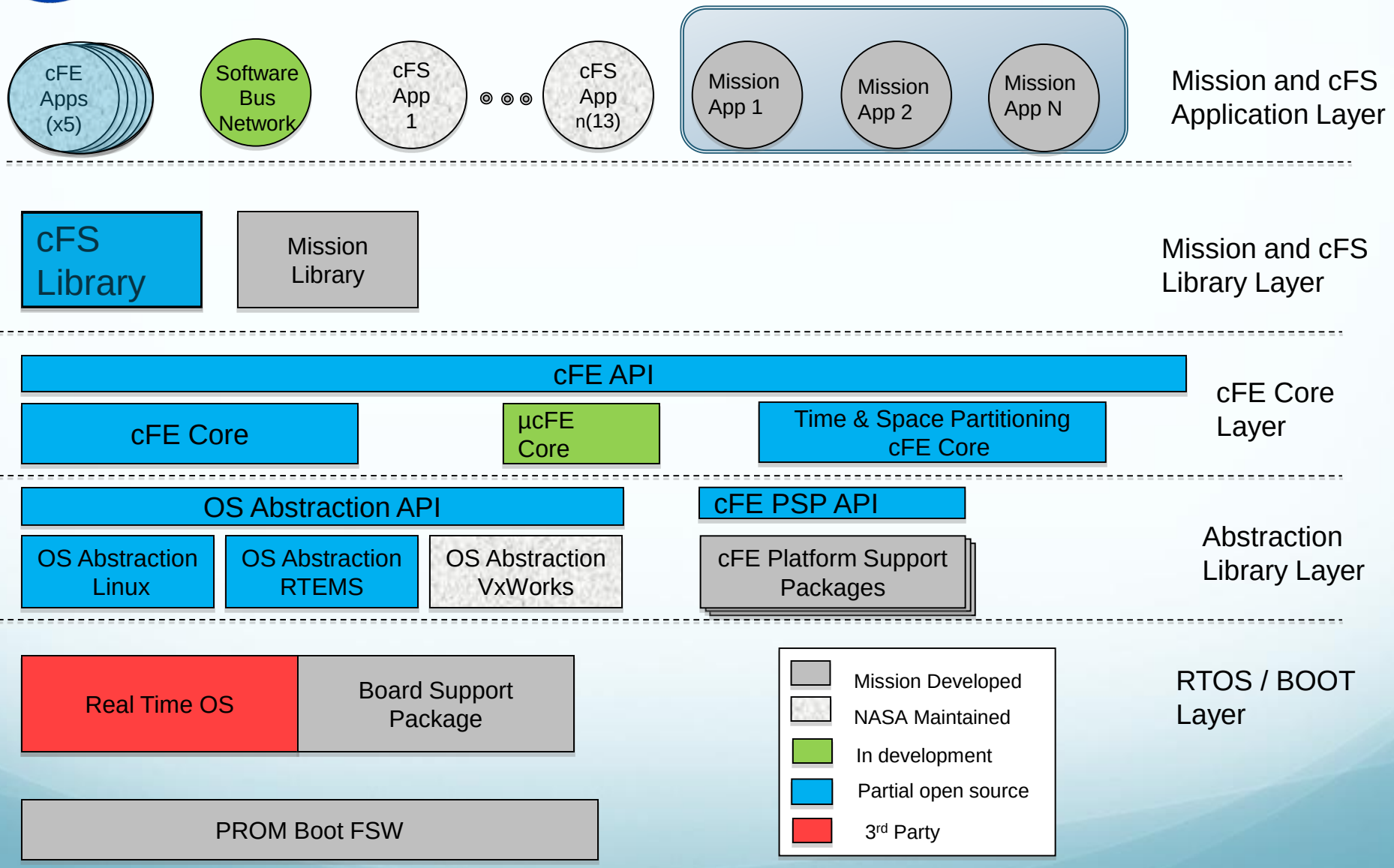
core Flight Executive (cFE)

- A framework of *mission independent, re-usable, core* flight software services and operating environment
- Layered on top of the OSAL and PSP

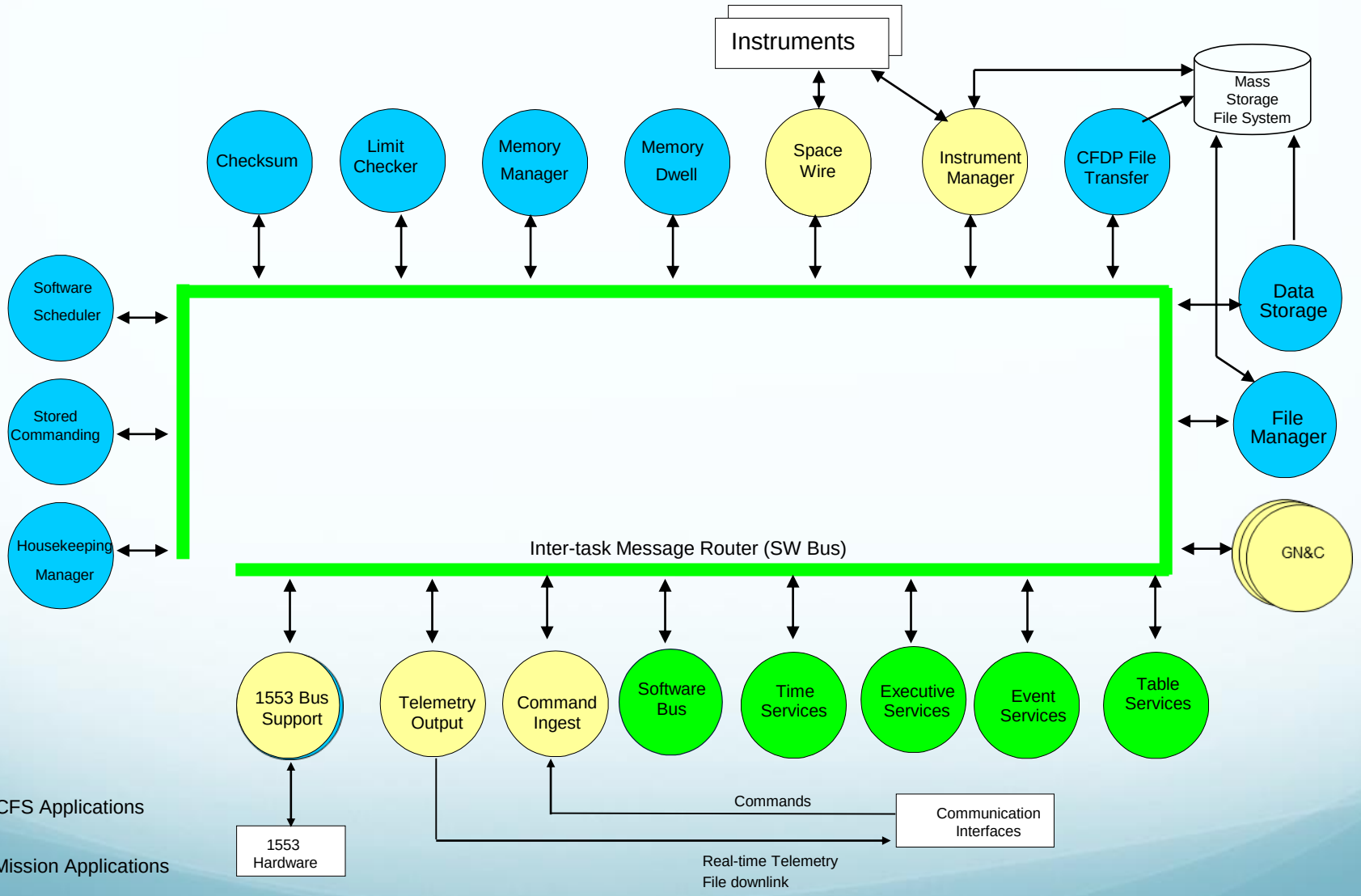
- **Each element is a separate loadable file**



cFS Software Layers and Components

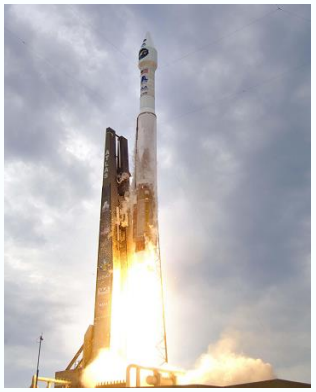


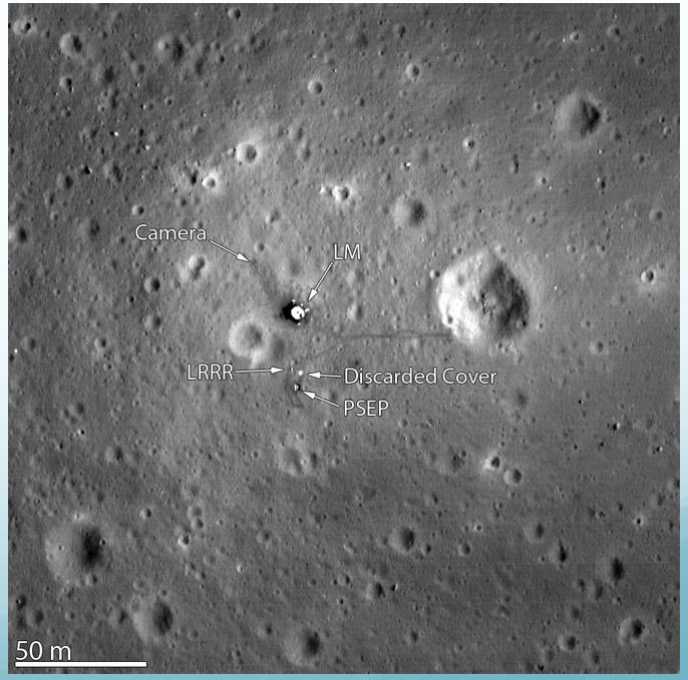
Flight Software "App" Store



Flight Missions (Class B)

- LRO, first mission to use cFE
  - Launched June 18, 2009
- GPM, most recent to use cFS
  - Launched February 14, 2014





A Recent Success

Observatory for Planetary Investigations from the Stratosphere (OPIS)

- Baseline command and data handling software was up an running on the target platform within a month
- OPIS launched 6 months later! (Class D mission)

10/08/2014 10:02am EST

10/08/2014 12:36pm EST

Lessons

- **Even in space we can use product line concepts**
- Code reuse is not enough!
- Most of the software artifacts must be reusable
  - Requirements
  - Documentation
    - Software user's guides
    - Operational user's guides
  - Test procedures
    - Automated Unit tests
    - Automatized Functional tests
  - All of the above need to be parameterized!
- Ground systems components should have a similar architecture
  - Many flight software components have a corresponding ground system component

Lessons

- Social aspects:
  - Project engineers must see value in it
  - Informal engineer to engineer interactions worked well
  - Stakeholders were engaged early
  - Engineers across projects helped shape the architecture
  - Resistance to change is hard to overcome
    - Personnel with less flight system experience quickly embraced the architecture and product line
  - When everyone is working with similar software and tools, people can and do help each other
    - Individual engineers started writing tools and wanted to share them
  - It's important to say why the architecture is the way it is
    - Heritage analysis is documented and a formal process
  - As the originator of the cFS, we had to get out of a "local" mindset, give up some control and let it be a community effort
  - **Attending conferences and workshops is important**

Lessons

- Need a well defined set of Quality attributes to evaluate future architectural decisions
  - There are too many good suggestions for enhancements, we needed an **objective** way to evaluate them
  - Defined quality attributes have:
    - Description
    - Aspect of
    - Requirement
    - Rationale
    - Evidence of verification
    - Tactic to achieve
    - Project specified
      - Prioritization
      - Intended Variation
  - The available literature did not seem to address this issue

Lessons

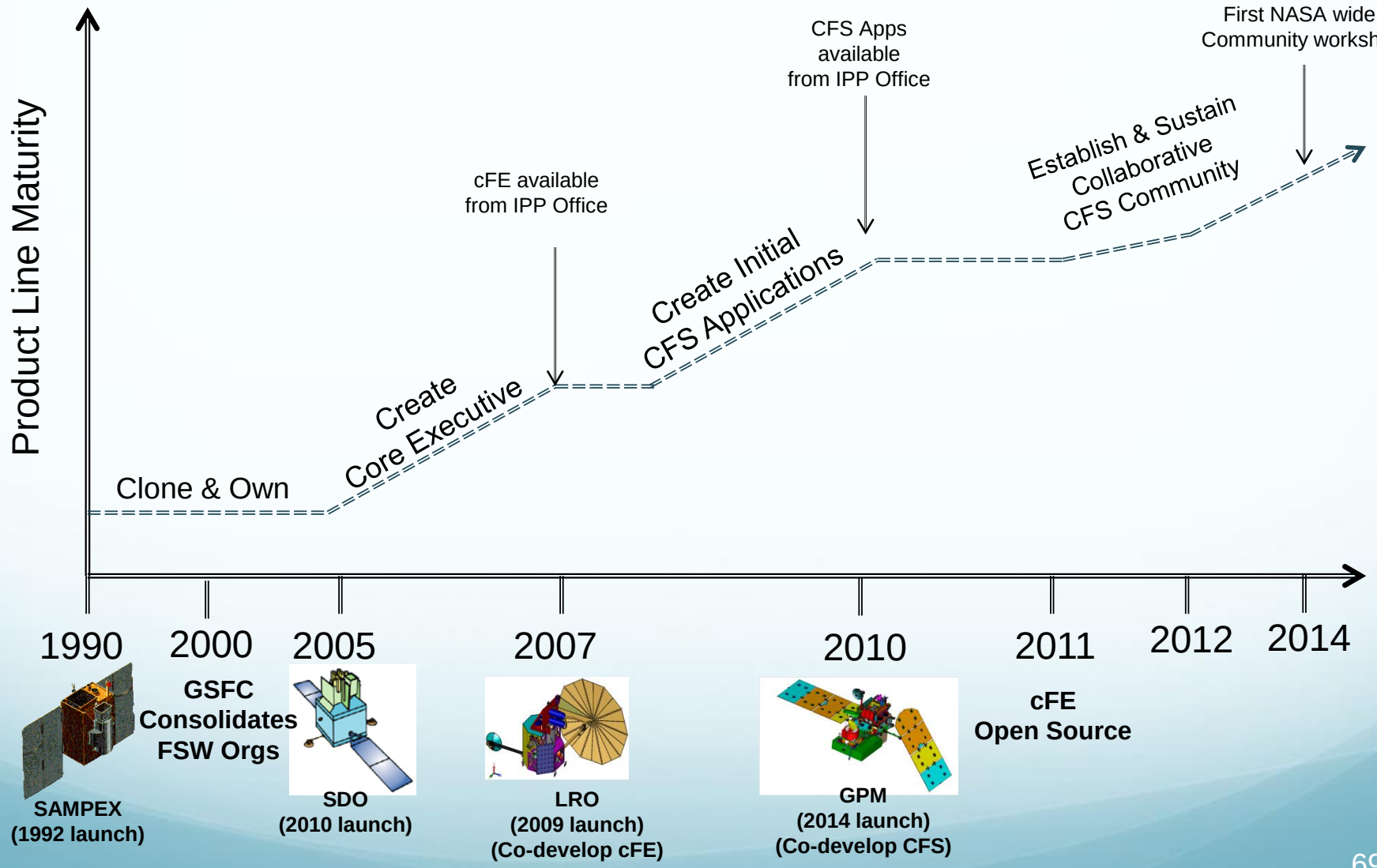
- The product line naturally incorporates best practices and mission experience
- Lots of support tools independently developed
  - Each user created an electronic data sheet tool
    - Tool to generate XML interface description from header files
    - Tool to generate header files from the XML
  - Code templates for reuseable components
    - Auto generated from IDE or command line
    - */* Put your code here */* Comments
- Auto generation of components from software models
- Variability analysis and designs seems to take a few projects to get it right (So plan for it)
  1. Try to write it for reuse
  2. Deploy it and find out what you did wrong
  3. Update and release it again

Lessons

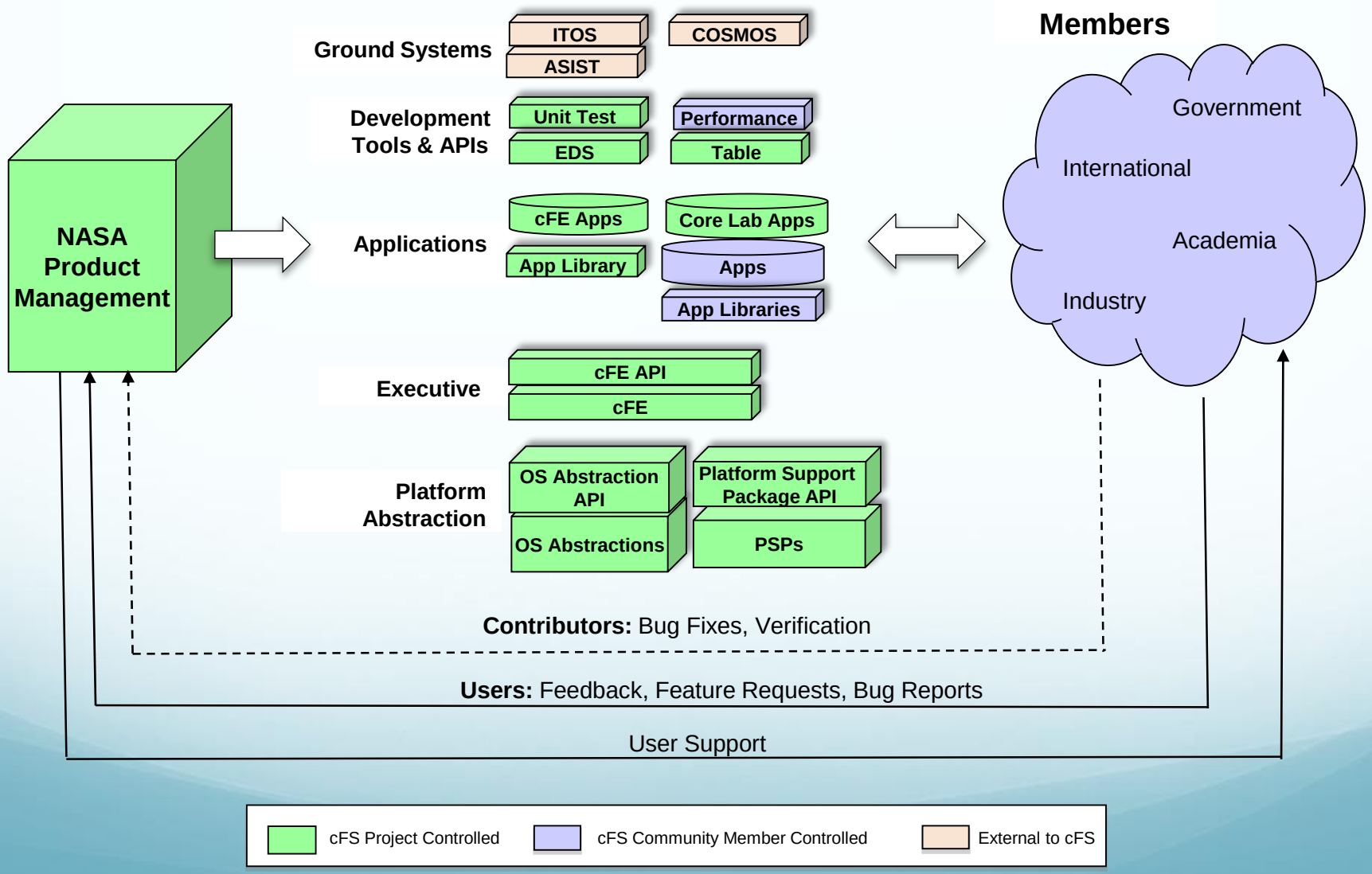
- Training and help must be available early
  - If they don't understand it, people will avoid it
- Easy to get a new developer started
  - Deploying virtual machines (VM's) with the environment and all the tools installed
- Support open source software (Linux, GNU tools, …)
- Whenever possible, make it open source!
  - OSAL at http://sourceforge.net/projects/osal/
  - cFE at http://sourceforge.net/projects/coreflightexec/
  - Plans to release suite of cFS applications
  - GSFC, Spring 2015)
  - Plans to release additional tools and the Integrated Development Environment (Eclipse based)
  - Plans to release AR Drone Quadricopter software kit
  - JSC, Summer 2015)

Building a cFS Community

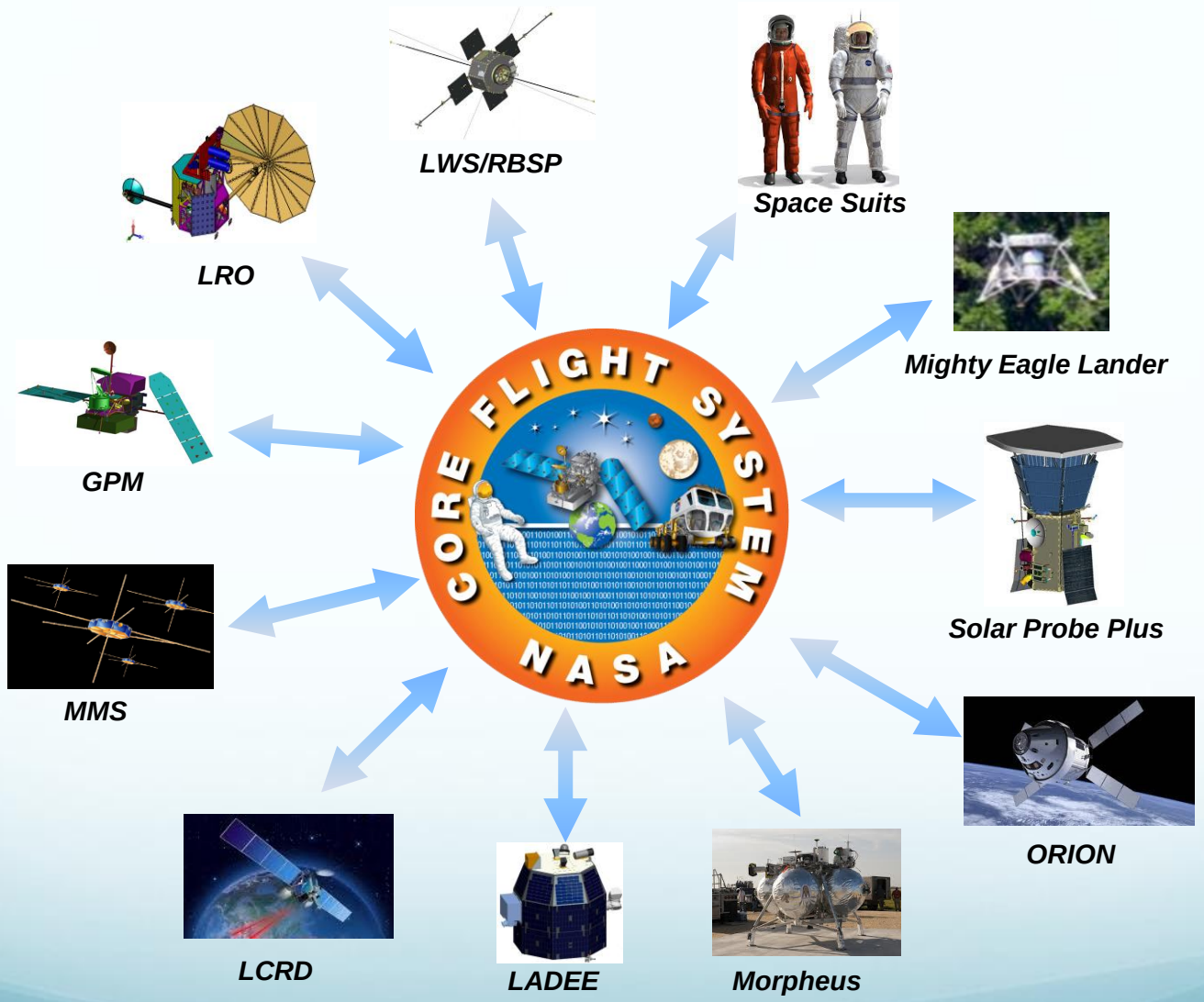
CFS Product Line Timeline



cFS Community



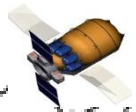
cFS use at NASA



The CFS architecture, originally developed to promote GSFC project collaboration and cost savings, has now become an Agency wide collaboration and cost savings resource

CFS Users Across the Globe

DOD and US industry
- Potential for standardization though the CCSDS and the Space Universal MOdular Architecture (SUMO) team sponsored by Office of the Director of National Intelligence



GRC – CPST and Advanced suit

APL - RBSP. Proposing use on Solar Probe, DoD programs.

LRO, MMS, GPM, NICER, OPIS and many others.

ARC- LADEE

Commercial - Moon Express (Lunar X-Prize)

Kirtland AFB – Onboard Autonomous Planning System

JPL – Evaluating architecture for robotic missions and ESTO missions

JSC-Used Successfully on Morpheus. Using on AES projects , Habitats, Waypoint, Certified for Class A (human rated).

MSFC- Mighty Eagle Lander, AES RESOLVE

KSC-Evaluating for AES, sounding rockets and UAV's

JAXA's Engineering Digital Innovation Center
Next generation software architecture research

Korea Aerospace Research Institute
Lunar program

European Space Research and Technology Centre

cFS Use Outside of NASA

- Korean Aerospace Research Institute (KARI) is planning to use cFS for South Korea's Lunar program
  - Working to create cFS DTN application to use between orbiters and rovers
  - Coordination through NASA HQ
- JAXA's Engineering Digital Innovation (JEDI) Center using cFS for prototyping next generation flight software architecture.
- AFRL plan to use cFS as baseline for "development of a common platform for the control architecture of small/medium UAVs"
- CCSDS Management Council has proposed creating a Reference Architecture Orange Book based on cFS
- Moon Express, Masten Space Systems, and Astrobotic Tech. using cFS for the Google Lunar X Prize

cFS Contributions From Other Organizations

Organization	Contribution	Notes
Johnson Space Center	Trick Simulator integration, Enhanced Build environment, Training materials, ITOS integration, multiple new platforms	
Johnson Space Center	Class A certification of OSAL, cFE and selected cFS applications	Use in Orion Backup flight computer, video processing unit, and Advanced Space Suit
Johnson Space Center	Enhanced Unit tests and increased code coverage, new performance analysis tool	
Glenn Research Center	Code Improvements, modern build environment (cmake), Electronic Data Sheet integration	
Ames Research Center	cFS community configuration management services, continuous integration build services	
Ames Research Center	Simulink Interface Layer for auto-coding cFS applications	
JHU/APL	Multi-Core cFE/OSAL port	Joint IRAD with GSFC, will be used for GSFC MUSTANG flight processor card
DARPA/Emergent	Fractionated Spacecraft / Distributed Mission cFS applications Formation Flying	Part of DARPA F6 project, they hope to make the apps available as open source
Interns and misc contributors	cFS development tools are being created and shared by many organizations Miscellaneous bug fixes reported via open source sites.	

Ongoing Activities

Technical Enhancements

- Integrated Development Environment (IDE)
- Automated tests (unit, functional, build…)
- CCSDS EDS specifications for cFS components
- Integrate Multi-core support into OSAL and cFE
- Integrate/Merge ARINC653 port into OSAL and cFE
- Integrate Dellingr Cubesat FreeRTOS OSAL Port
- Improve scheduler time synchronization
- Expand SB namespace beyond 2^{11}
- Lab upgrades
  - RTEMS 4.11 updates
  - VxWorks 6.9 updates
  - RAD750 simulator
  - MPC8377E: PowerQUICC II Pro Processor test beds
  - LEON3 test bed
  - MCP750 test bed

Operational Enhancements

- Formalize cFS user community
- Web based app store