ELISA
Enabling Linux in Safety Applications

WORKSHOP

NASA Goddard

core Flight System (cFS) Overview

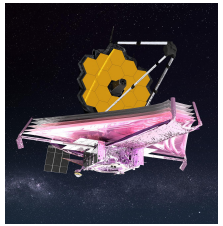
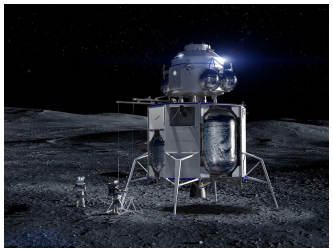
Rich Landau
NASA GSFC
Flight Software Systems Branch (Code 582)

What is Flight Software?

Flight Software (FSW) is:

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

FSW


Not FSW

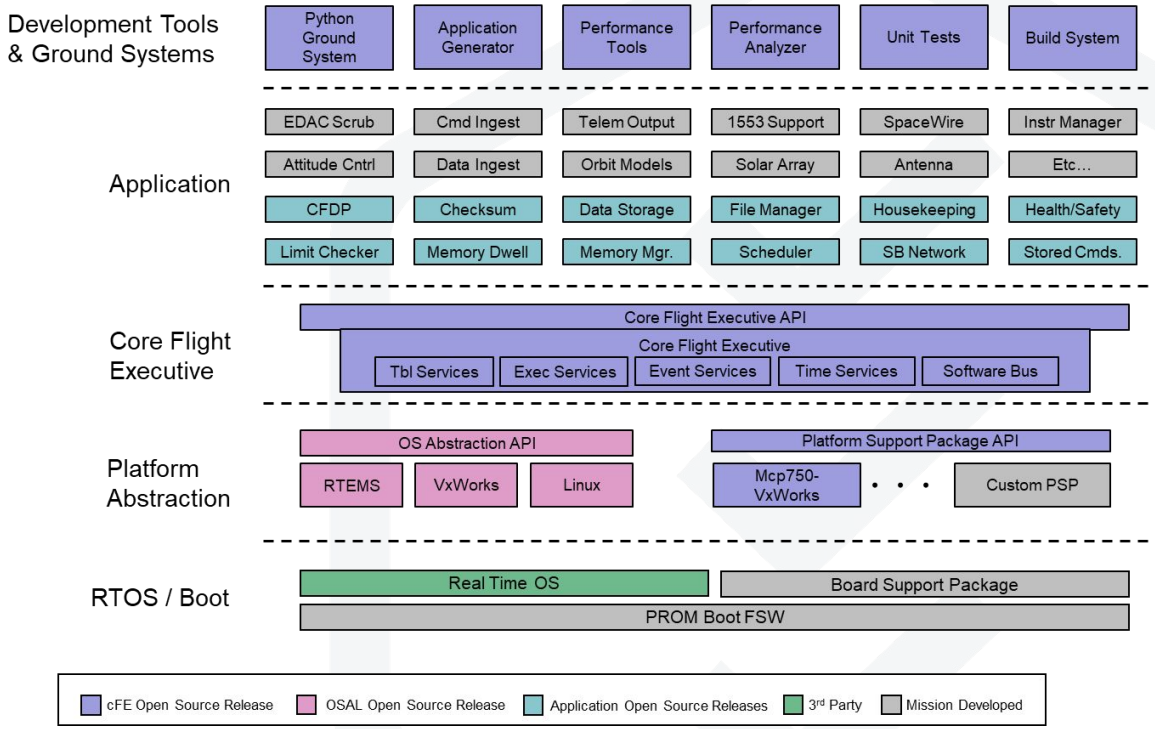

What is cFS?

cFS named in the NASA International Software System Interoperability Standards as the standard FSW framework

NASA 2020 Software of the Year Award winner!

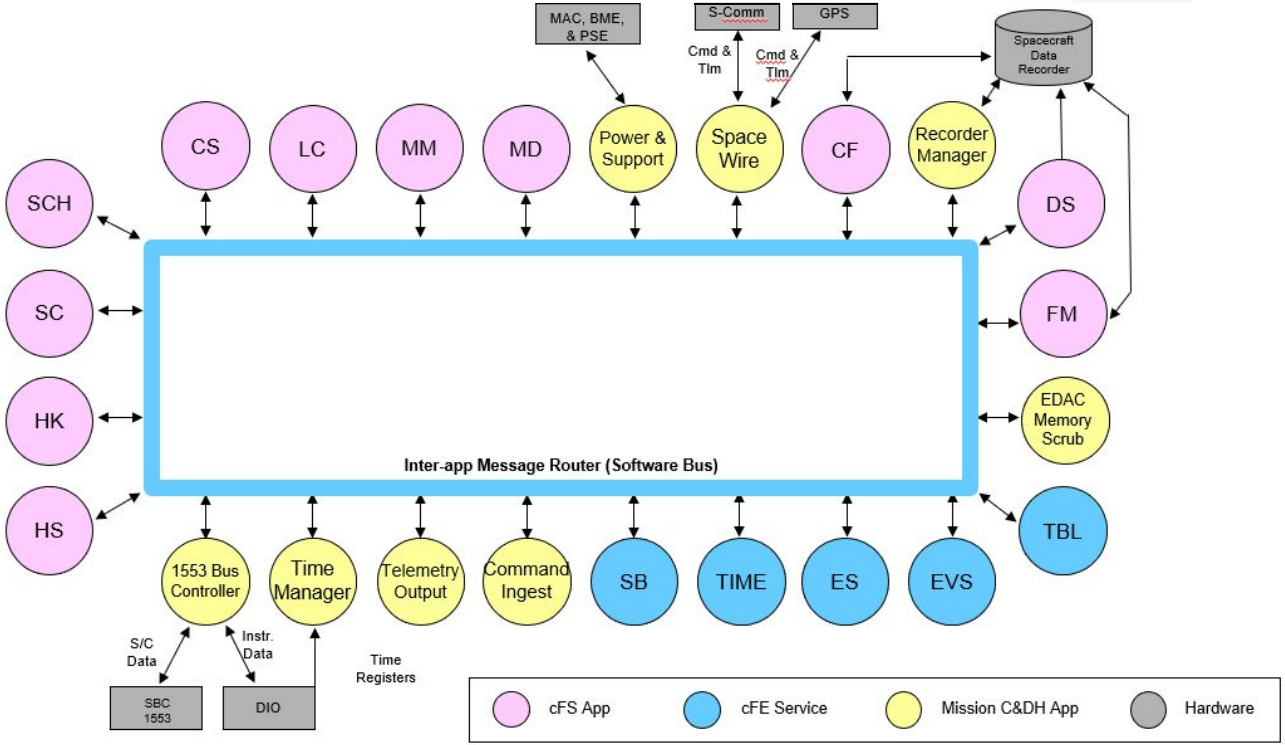
A flight-proven FSW framework, providing:

- 12+ modular software applications for common functions (limit checker, file transfer, memory management, etc)
- Core services required by all flight missions
- Portability via processor and operating system abstraction
- Supporting tools (unit test framework, performance monitoring, ground system simulator, etc)



Open source at: https://github.com/nasa/cFS

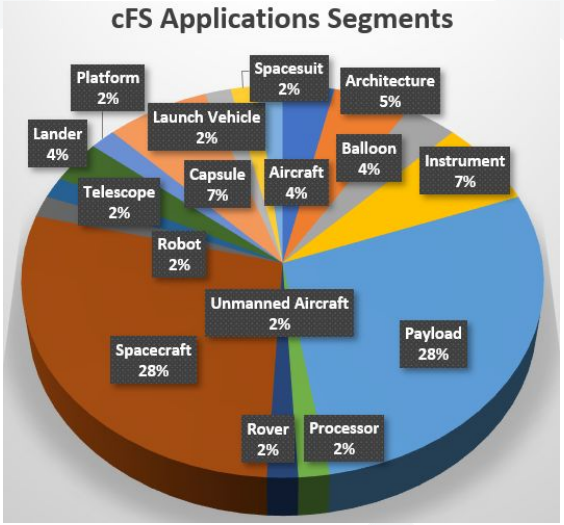
Typical cFS Deployment



cFS Usage at NASA and Beyond

cFS has been used by:

- 40+ NASA missions across 6 NASA Centers
- Flagships such as Gateway, Roman Space Telescope, and Mars Sample Return
- 12+ private companies and universities *that we know of*, most recently on the Intuitive Machines (IM) Odysseus Lunar Lander
    - *IM CTO publicly credited cFS for the quick-turnaround reprogramming of the landing LiDAR configuration that saved their mission*
- 5+ foreign space agencies *that we know of*
- A Roomba



Benefits of cFS

Quality: cFS reuse reduces defects significantly. We haven't found all the bugs, but we're pretty close.

Cost: cFS reuse reduces cost by at least 50% over clone-and-own.

Sustainment: Code "feel" is shared across projects, making it easier to move between projects and sustain older ones

Risk and Schedule: Starting from cFS substantially lowers project risk and makes schedules more predictable.

Standards: cFS is now a standard for the space community defined by the Human Exploration and Operations Mission Directorate in 2020 as the standard FSW framework for NASA, ESA, CSA, JAXA.

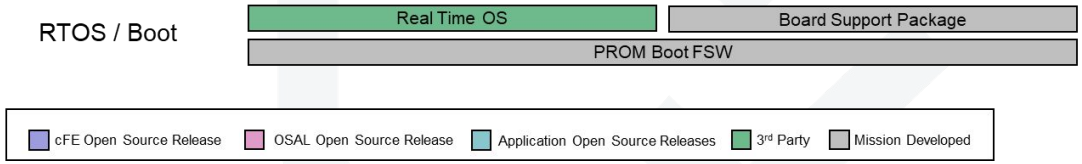
Layers of cFS

Operating System / Boot layer

- The bootloader is responsible for bringing up the system and starting the operating system

- Historically we've focused on real-time operating systems (RTOS) with Linux often (but not always) relegated to desktop testing

- We hope this is changing!

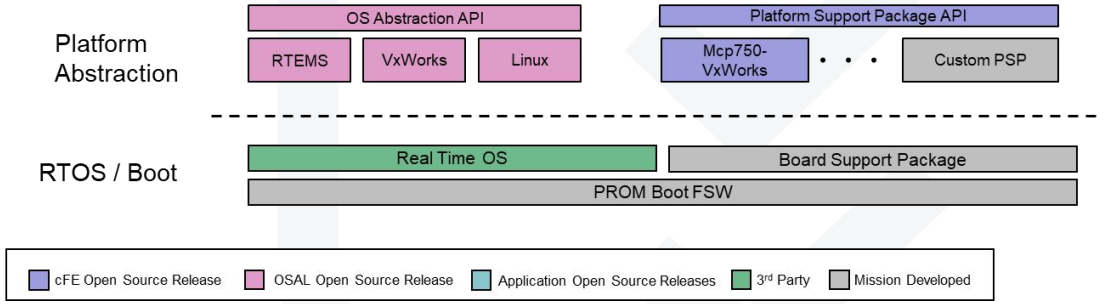


Platform Abstraction layer

- Provides a single API for higher levels regardless of OS/platform

- Built-in support for RTEMS, VxWorks, and Linux (desktop)
  - Projects have implemented OSALs for many other OSs

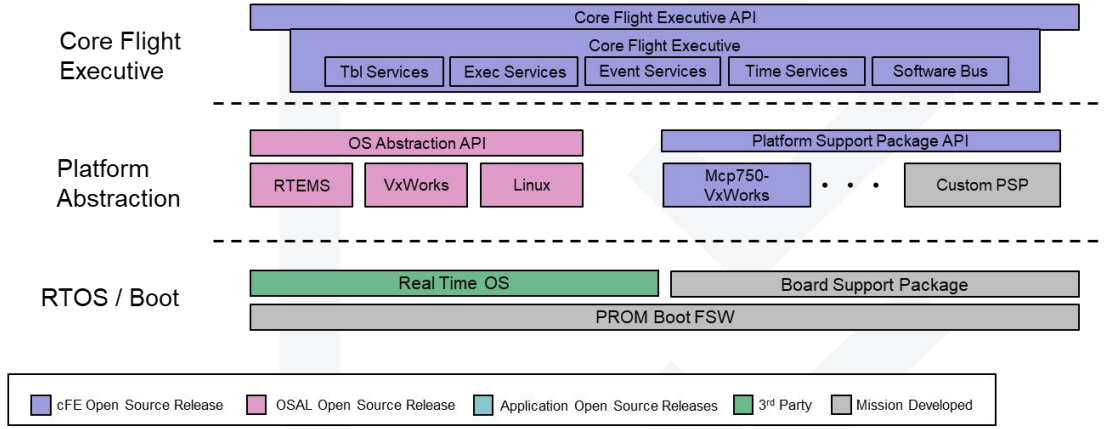
- These days many OSs support POSIX, simplifying the work here



core Flight Executive (cFE) layer

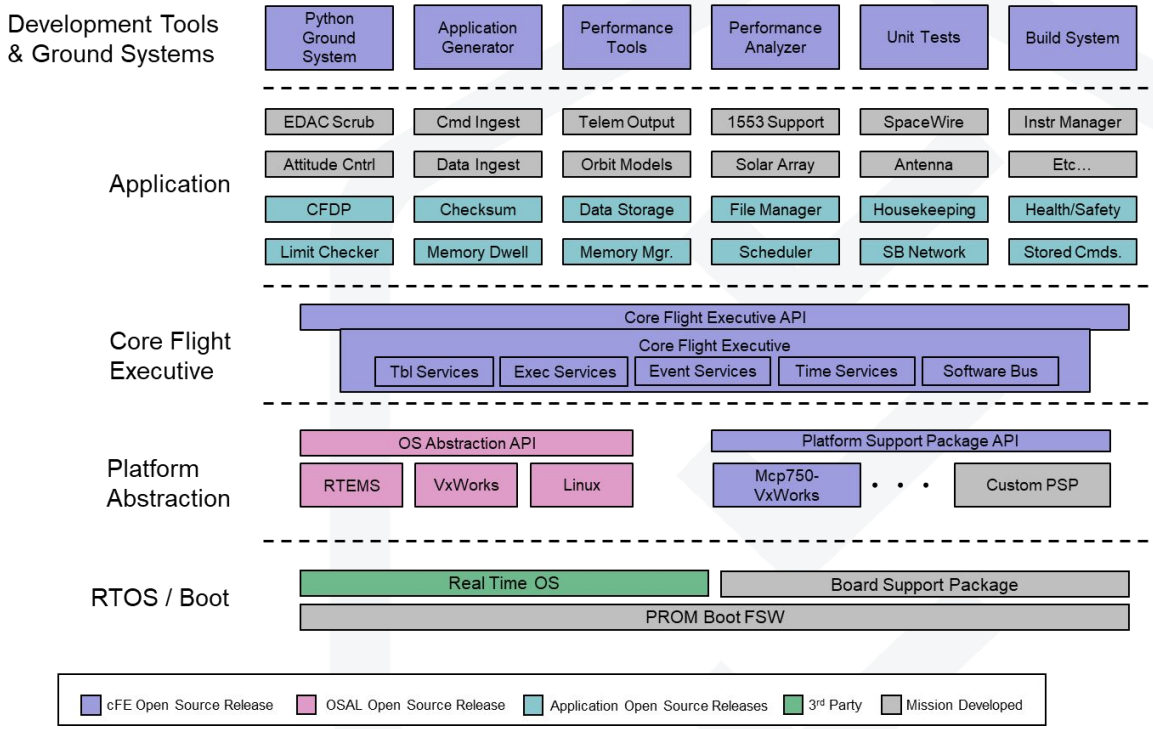
Provides core services that all missions require:

- Executive Services: Start, manage, and stop services and apps, provide data and utilities such as memory pools
- Event Services: Log and downlink events that occur onboard
- Software Bus: Route messages between applications using a publish/subscribe paradigm
- Table Services: Provide dynamic configuration to apps
- Time Services: Provides the time and associated features



Application layer + tools

- Modular applications available on GitHub for common functions
- Missions spend most of their time developing custom apps for their mission use cases (talking to hardware, processing data, etc)
- Applications are written independent of OS/platform using cFE and platform abstraction layers
- Included tools simplify development and verification



The Future

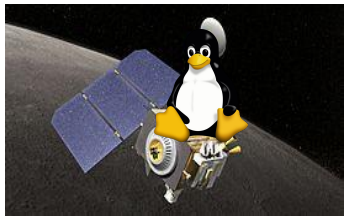
Linux's Role in cFS

Historically reserved for development and prototyping, two major developments may bring Linux into the forefront of cFS architectures:

Linux as a first-class FSW operating system

With PREEMPT_RT in the mainline, Linux may get more adoption for flight software OSs, especially for lower mission classes (i.e. unmanned), or on those with more resources

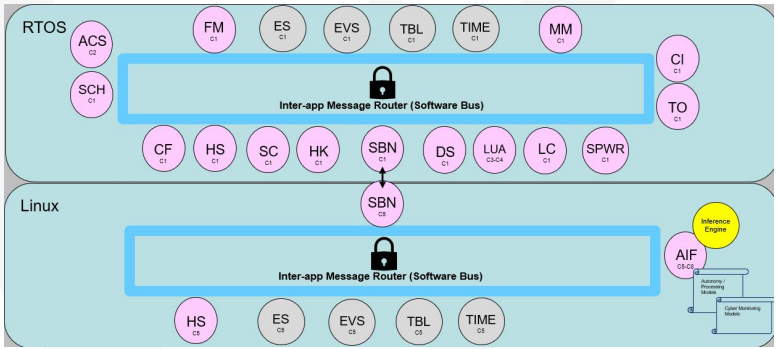
The challenge: Many of our missions still run with ~16 MB of RAM and ~8 MB storage, the kernel needs to be small



Penguin image credit: lewing@isc.tamu.edu Larry Ewing and The GIMP

Linux for parallel processing

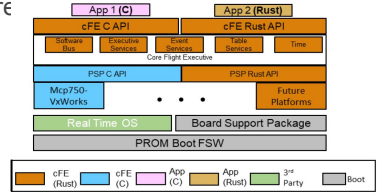
Significantly faster flight processors are being developed with advanced partitioning functionality. This would let us run an RTOS for real-time spacecraft control and Linux for non-safety-critical processing



What else are we excited about?

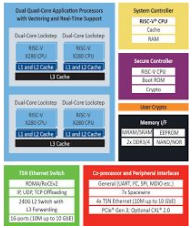
Memory Safe Programming Languages

- IRAD project is in work to convert portions of cFS to the Rust language, providing a Rust API for apps while remaining backwards compatible with C
- Will provide memory and concurrency safety and allow the use of a more modern, expre



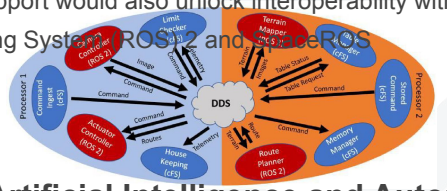
More Powerful Spaceflight Hardware

- NASA/JPL/Microchip collaboration to build a next-gen RISC-V spaceflight processor
- Eight-core processor for parallelized workloads
- Integrated vector engines for enhanced AI processing
- Hypervisor for running parallel OSs in tandem
- Much more



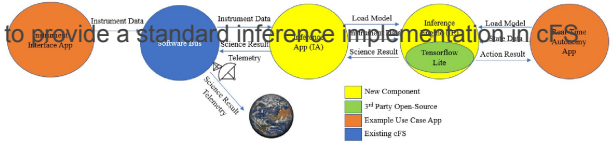
Distributed Communication and Computing

- IRAD project is in work to support the Data Distribution Service (DDS) within cFS's Software Bus
- Project would enable streamlined cross-processor and cross-subsystem communication without explicit bridges
- DDS support would also unlock interoperability with the Robot Operating System (ROS) 2 and NASA ROS



Onboard Artificial Intelligence and Autonomy

- Advances in hardware enable onboard AI and autonomy
- Science processing can be done onboard
- Onboard LLMs for low-risk decision making and manned expert advice
- Desire to provide a standard inference implementation in cFS



ELISA Enabling Linux in Safety Applications

WORKSHOP

Licensing of Workshop Results

All work created during the workshop is licensed under Creative Commons Attribution 4.0 International (CC-BY-4.0) [https://creativecommons.org/licenses/by/4.0/] by default, or under another suitable open-source license, e.g., GPL-2.0 for kernel code contributions.

You are free to:

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.