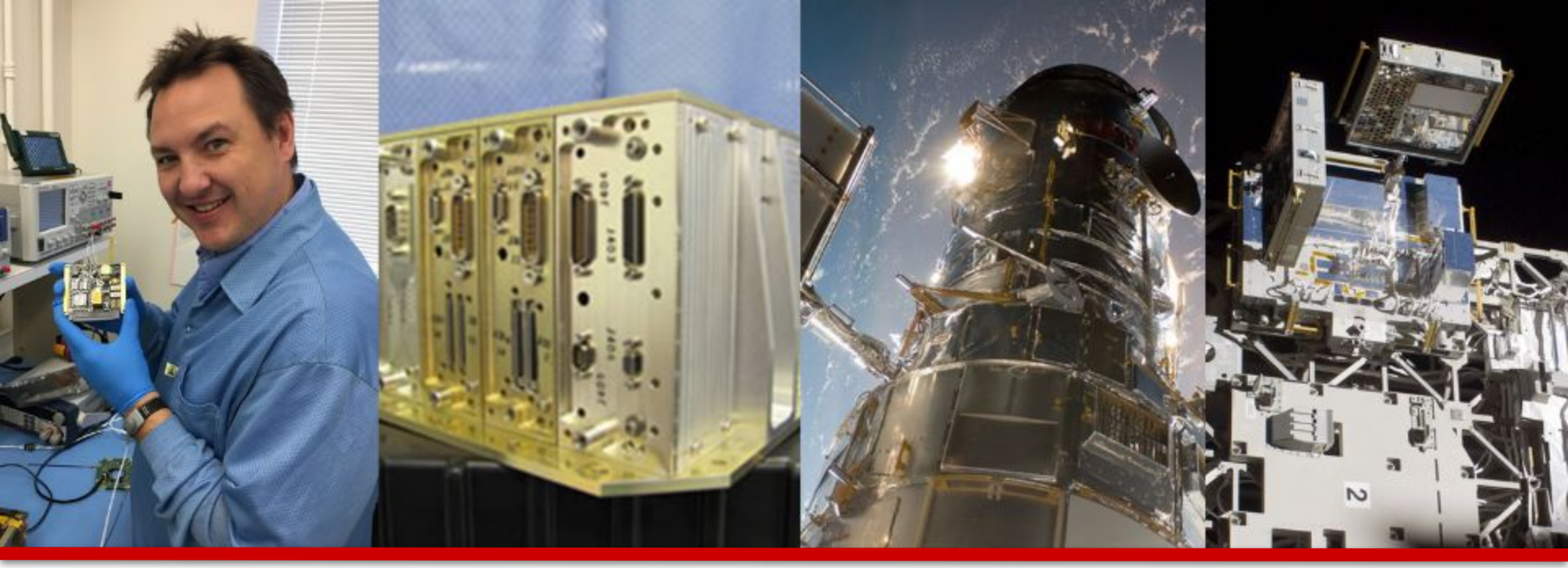
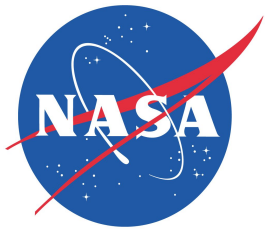
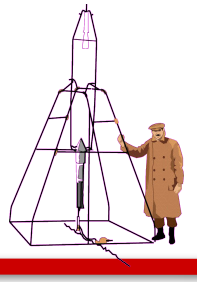


Multiple Process Support in cFS:
Enabling Safe On-Board Processing

James Marshall (james.marshall-1@nasa.gov)
NASA Goddard Space Flight Center
Science Data Processing Branch (587)
FSW 2021, February 11th 2021



Outline



What is cFS?

Goals and Motivation

Multiple Process Support

Language Support

RAISR and Ongoing Work

Contributors and Acknowledgements

What is cFS?

NASA's core Flight System

Flight software framework

Set of reusable applications

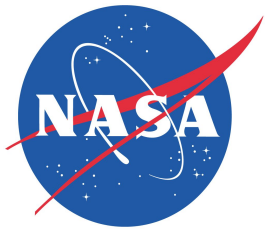
Layered architecture supports multiple OSes, hardware targets

Used in flight

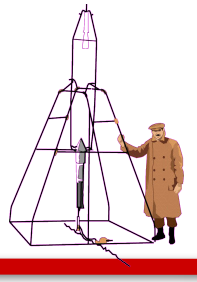
Open Source!
https://github.com/nasa/cfs

Mailing list:
https://lists.nasa.gov/mailman/listinfo/cfs-community

Image Credit: NASA/Ben Smegelsky
(https://www.nasa.gov/content/magnetospheric-multiscale-observatories-processed-for-launch)



Relevant cFS Design Considerations



Originally targeted “embedded” systems:

- Real-time OS
- Microcontroller without a memory management unit

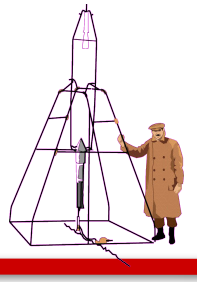
Single process, mutli-threaded model

‘pc-linux’ target originally used for development and debugging

Services and libraries bundled together

Active, focused development

Goals and Motivation



Enabling Safe Onboard Processing

Goal 1: Safety

Reduce fault propagation by taking advantage of existing OS protections

Goal 2: Processing

Support languages better suited to data processing

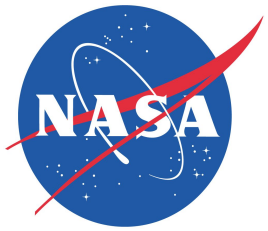
Requirements:

Target platform: ARM Cortex A running Linux

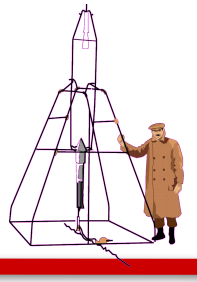
Launch a cFS application as a separate process

Use software bus, other cFS services, and library functions

Minimize changes to cFS: extend, don't modify



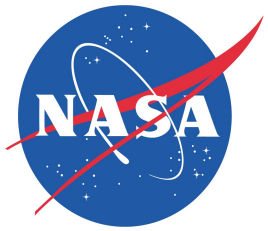
Motivation: Safety



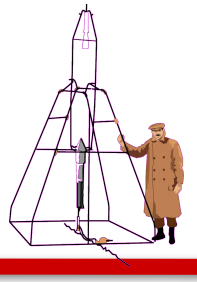
I’m sure everyone *here* writes perfect software

New Domains: SmallSats, CubeSats, ISS experiments

- Running Linux <- robust support for threads and processes
- High performance, commodity processors <- performance overheads okay
- More risk tolerant
- Schedule and budget constrained



Motivation: Processing



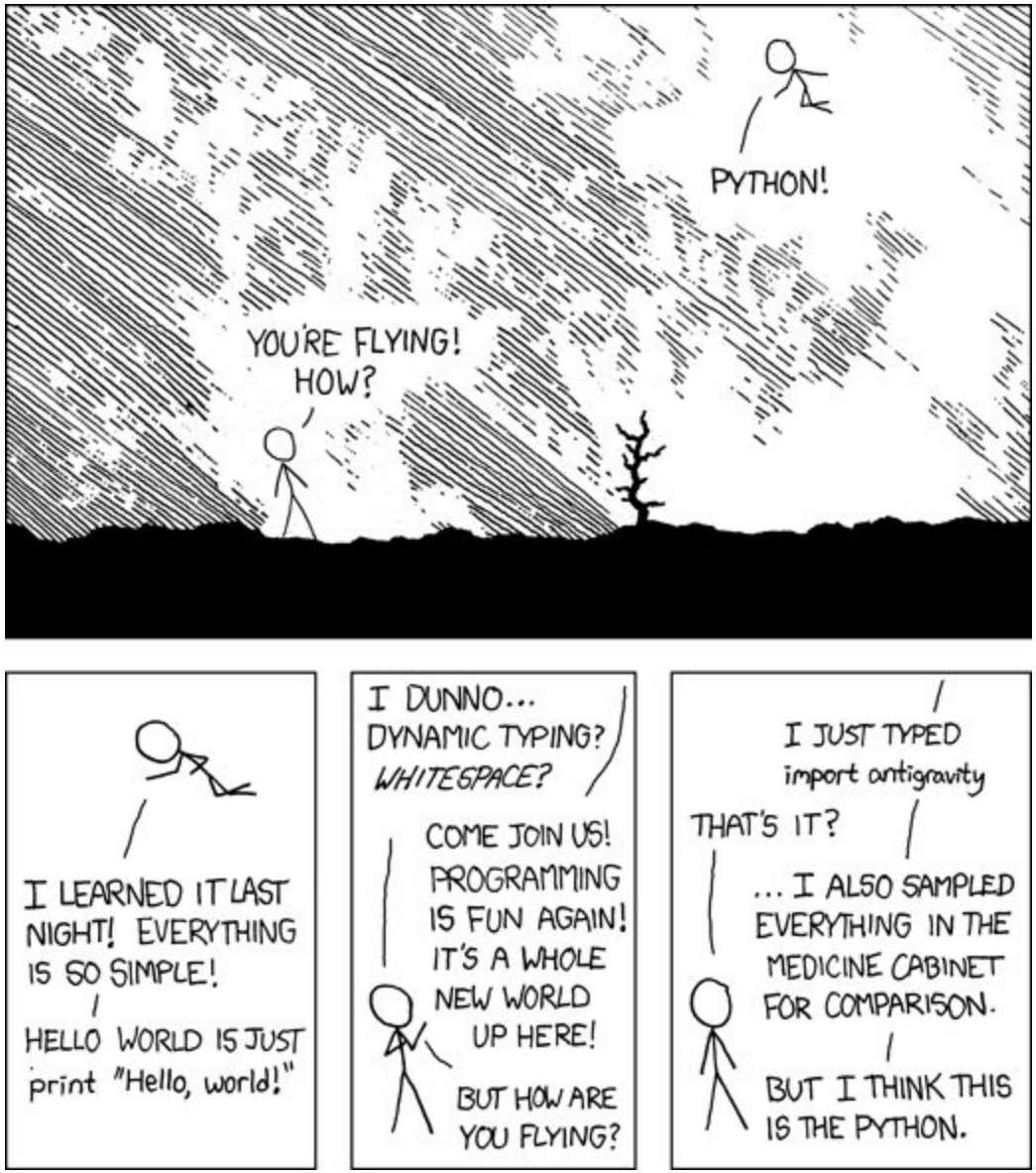
There is a need for on-board processing

- Instruments are generating tons of data
- Reduce bandwidth requirements
- Increase autonomy

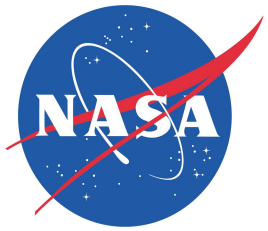
We have the hardware (SpaceCube!)

Also need software to take advantage

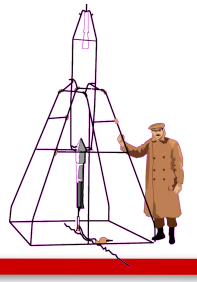
What language should we use?



Alt Text: I wrote 20 short programs in Python yesterday. It was wonderful. Perl, I'm leaving you.



Why Python?



Python would be great to use:

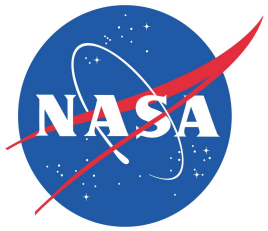
- Already used by scientific communities
- Has libraries available for almost everything
- Faster development, more expressive

C is a troublesome language…

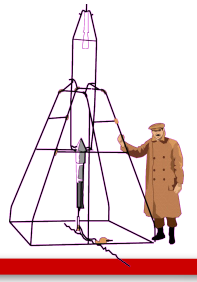
Enough Rope to Shoot Yourself in the Foot
Rules for C and C++ Programming

By Allen I. Holub

Multiple Process Support



Multiple Process Support



Small FY19 Internal Research and Development project

Focused on using an OS process instead of thread to encapsulate a cFS app

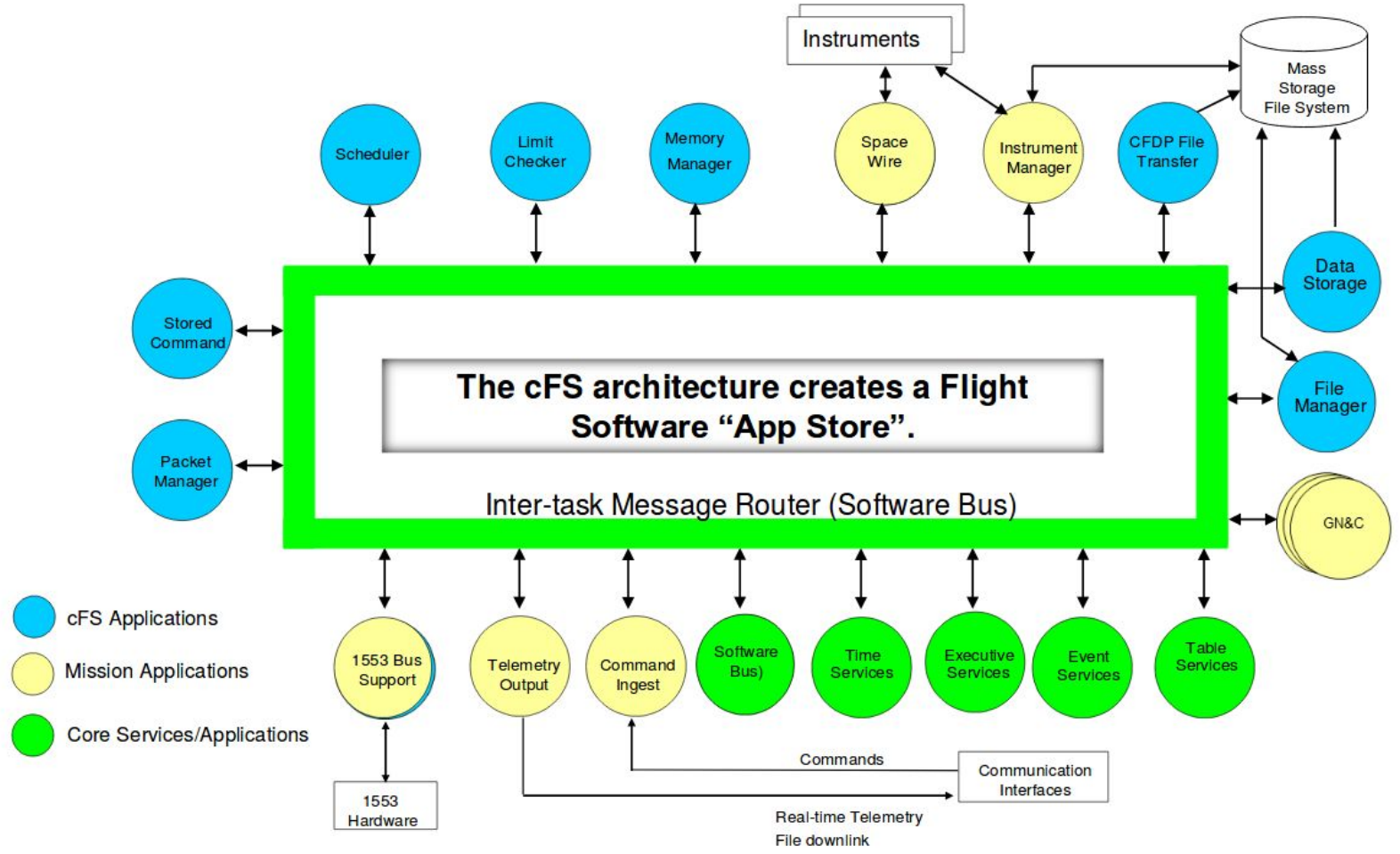
Main benefit: memory isolation

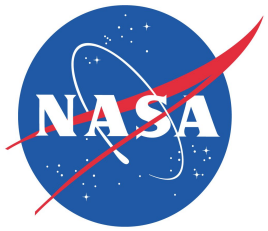
Main cost: performance

This work built the functionality we needed for Python support

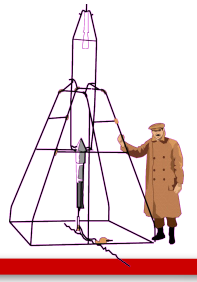
It also negates some of the risks of using Python

cFS From App Developer Point of View

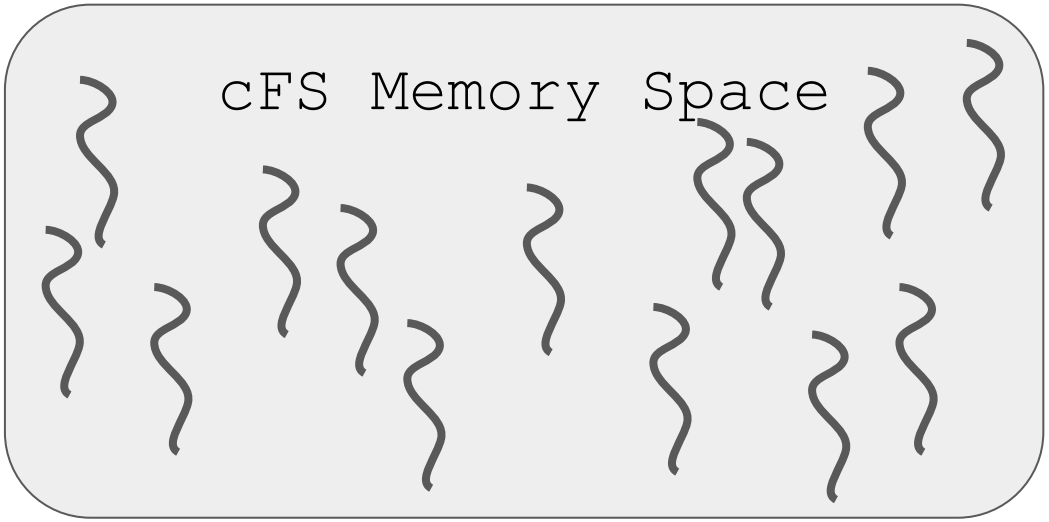




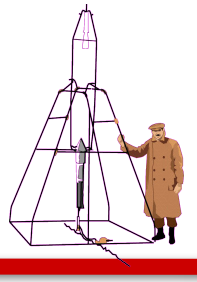
cFS From the OS Point of View



cFS typically has a single memory space

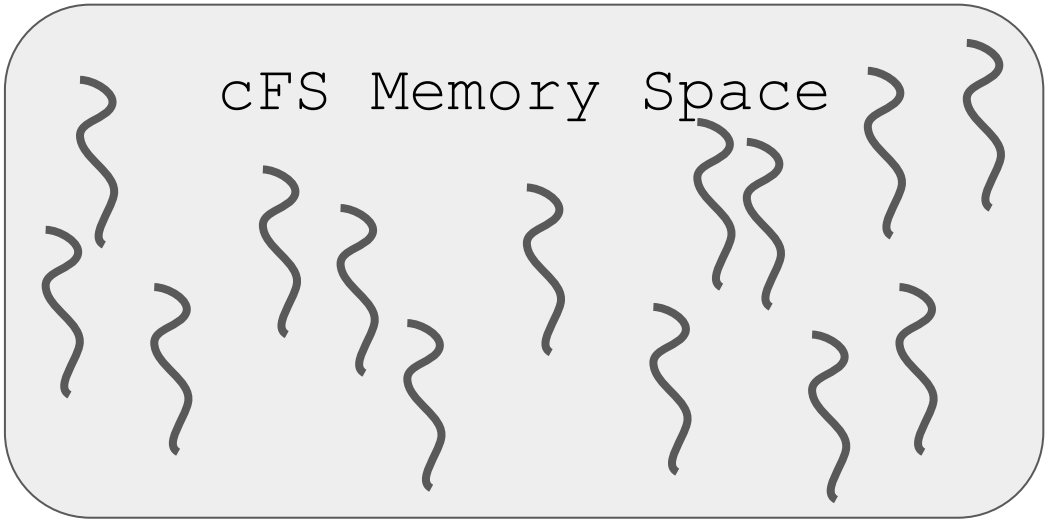


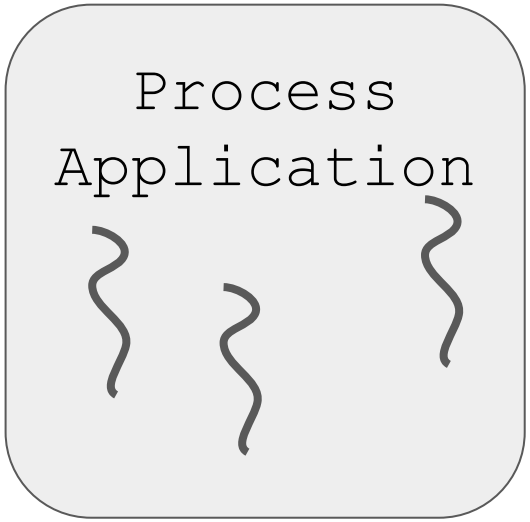
cFS From the OS Point of View



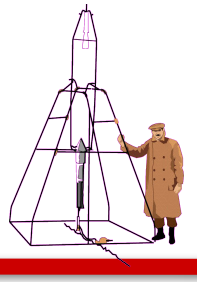
cFS typically has a single memory space

Our process application has an isolated memory space





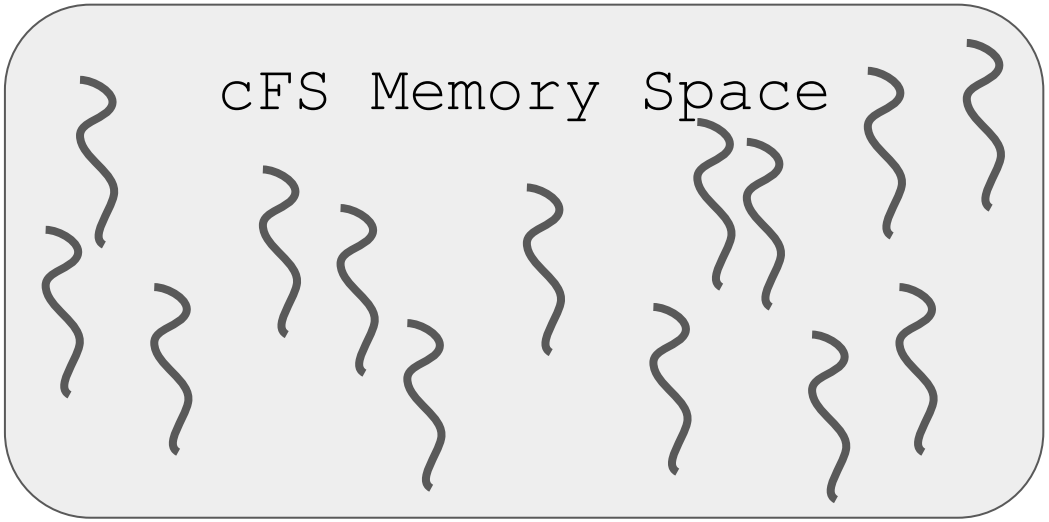
cFS From the OS Point of View



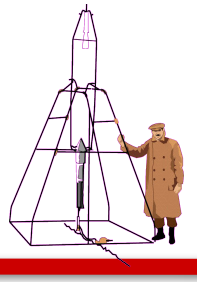
cFS typically has a single memory space

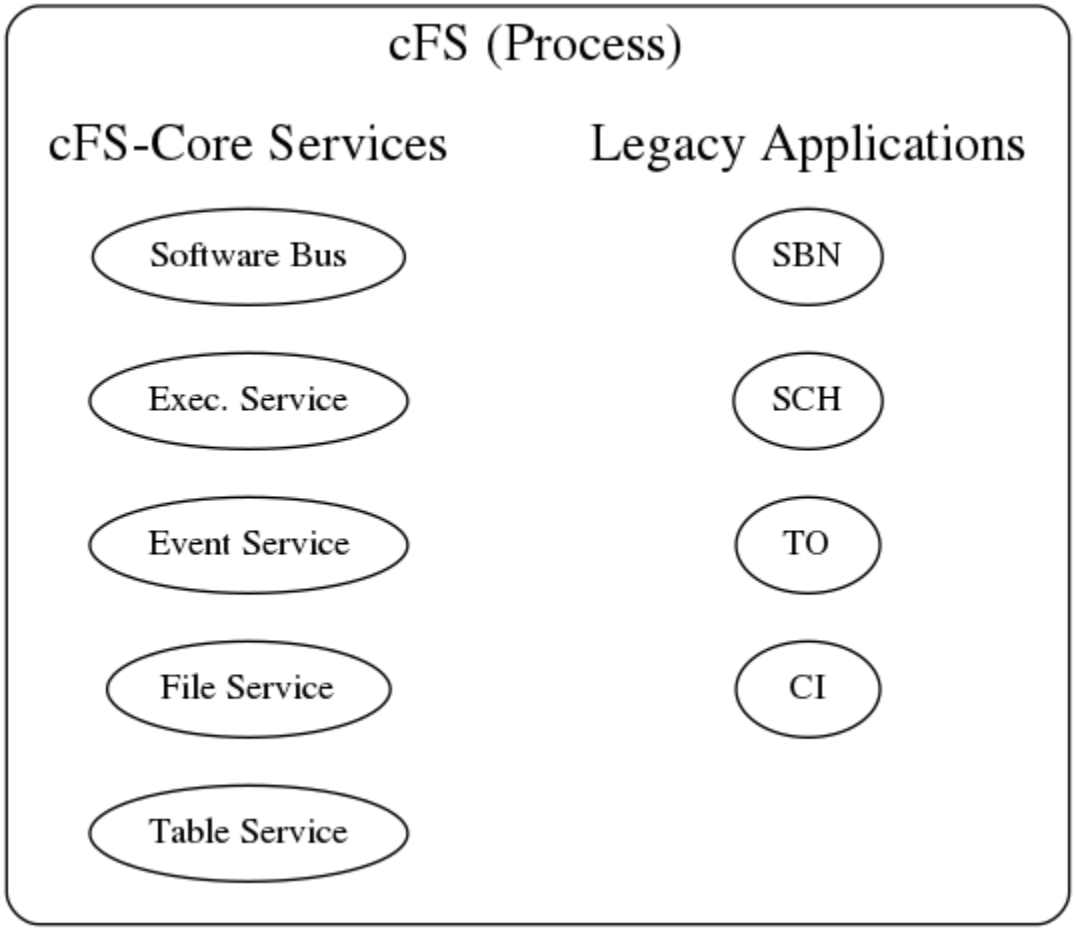
Our process application has an isolated memory space

This prevents faults from propagating to the rest of cFS



Current cFS



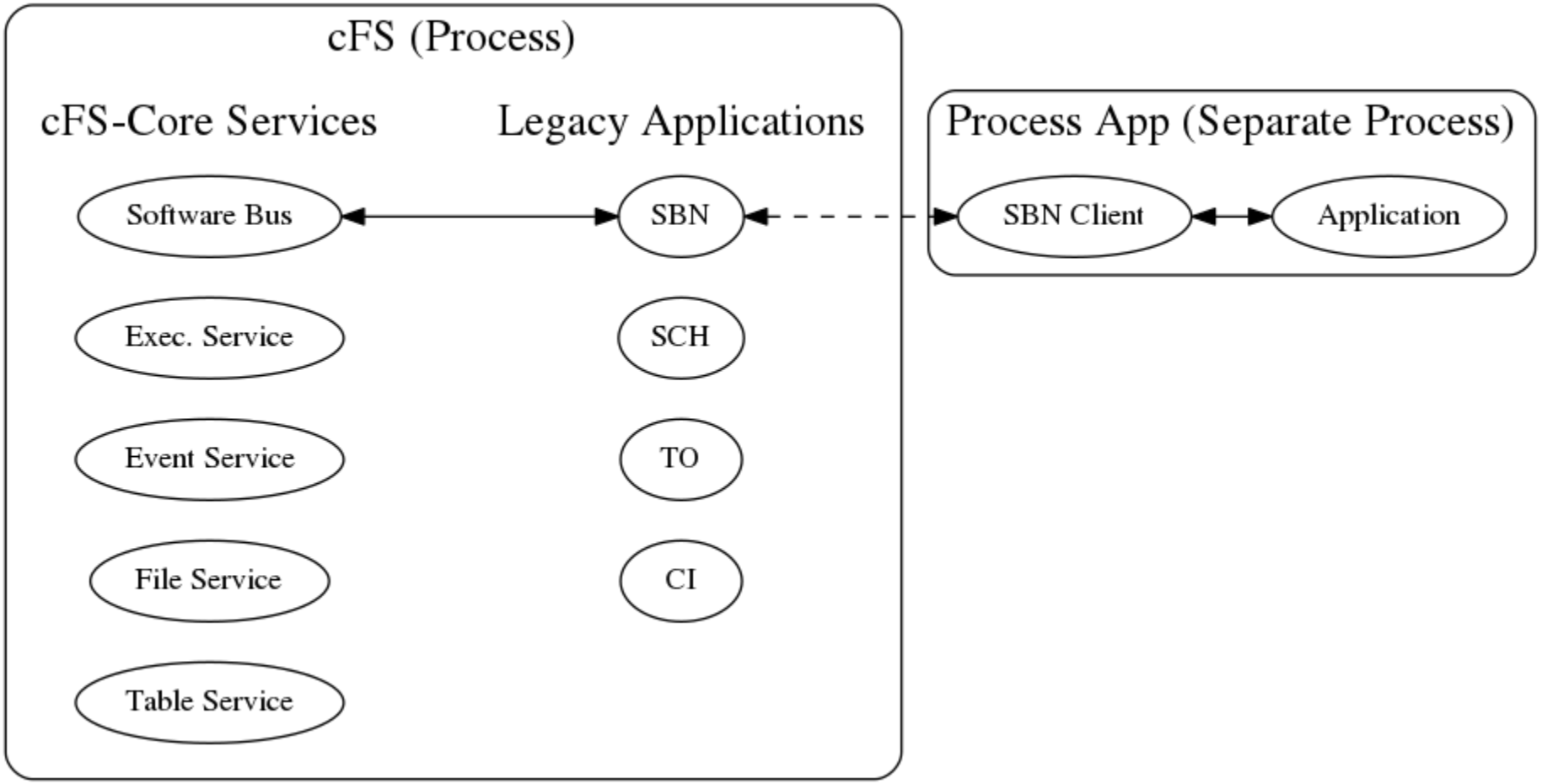


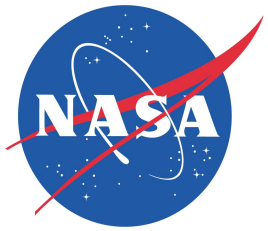
Single memory space

Function calls between applications and services

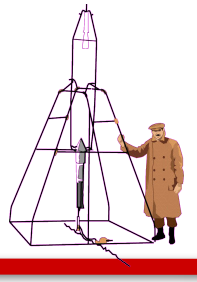
Inter-Application communication via Software Bus

Software Bus Access





Software Bus Network

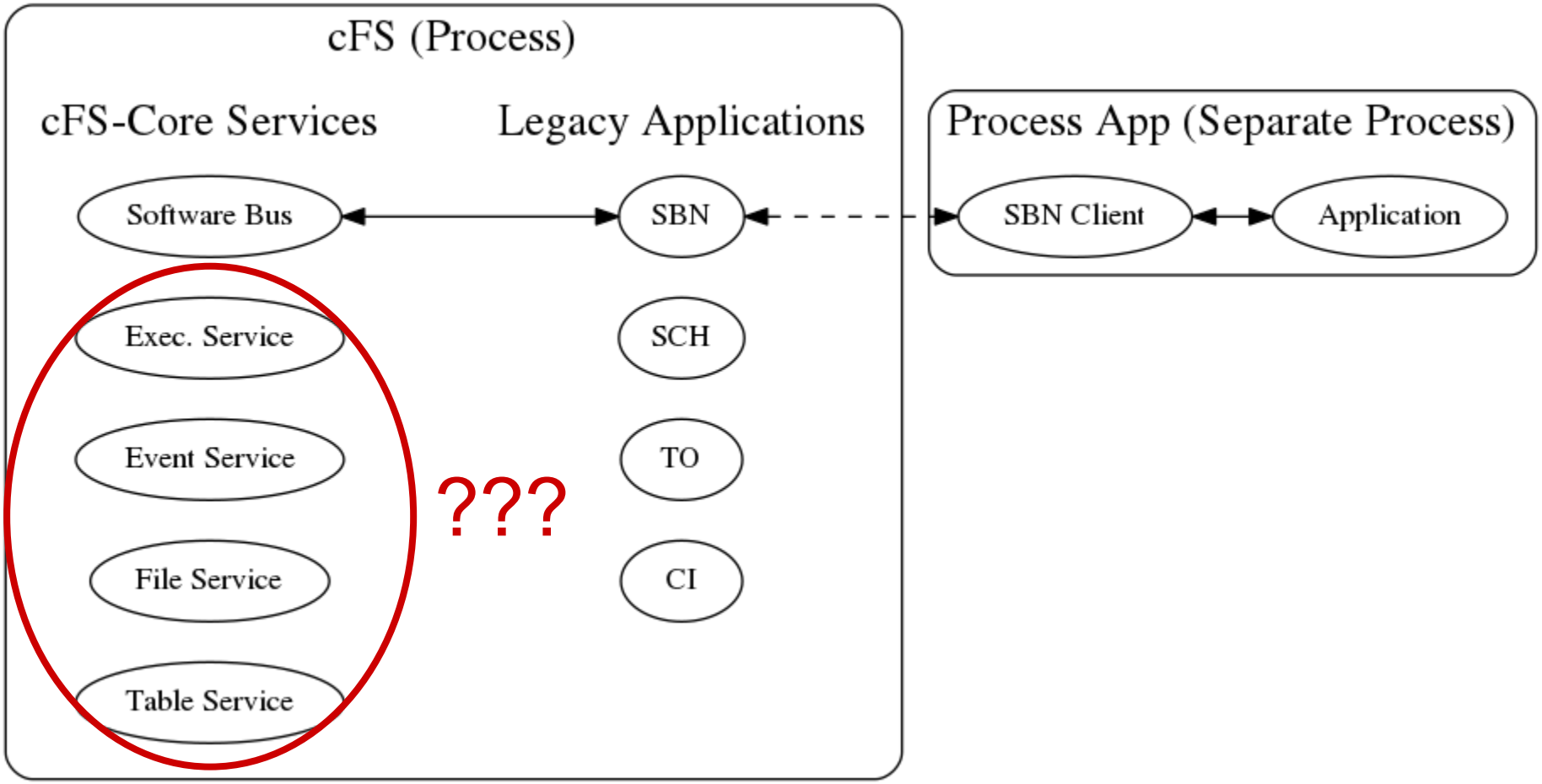


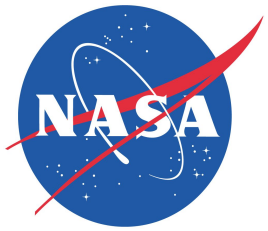
Software Bus Network is an existing application (thanks Chris Knight!)

The client connects to it over a local socket

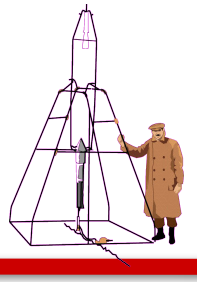
Allows for inter-application communication with Publish/Subscribe architecture

Access to Other Services?





Proxy Application



Core services are usually called via functions

Some functions do not use global data: safely called as a library

- Example: utilities functions like `CFE_FS_InitHeader`

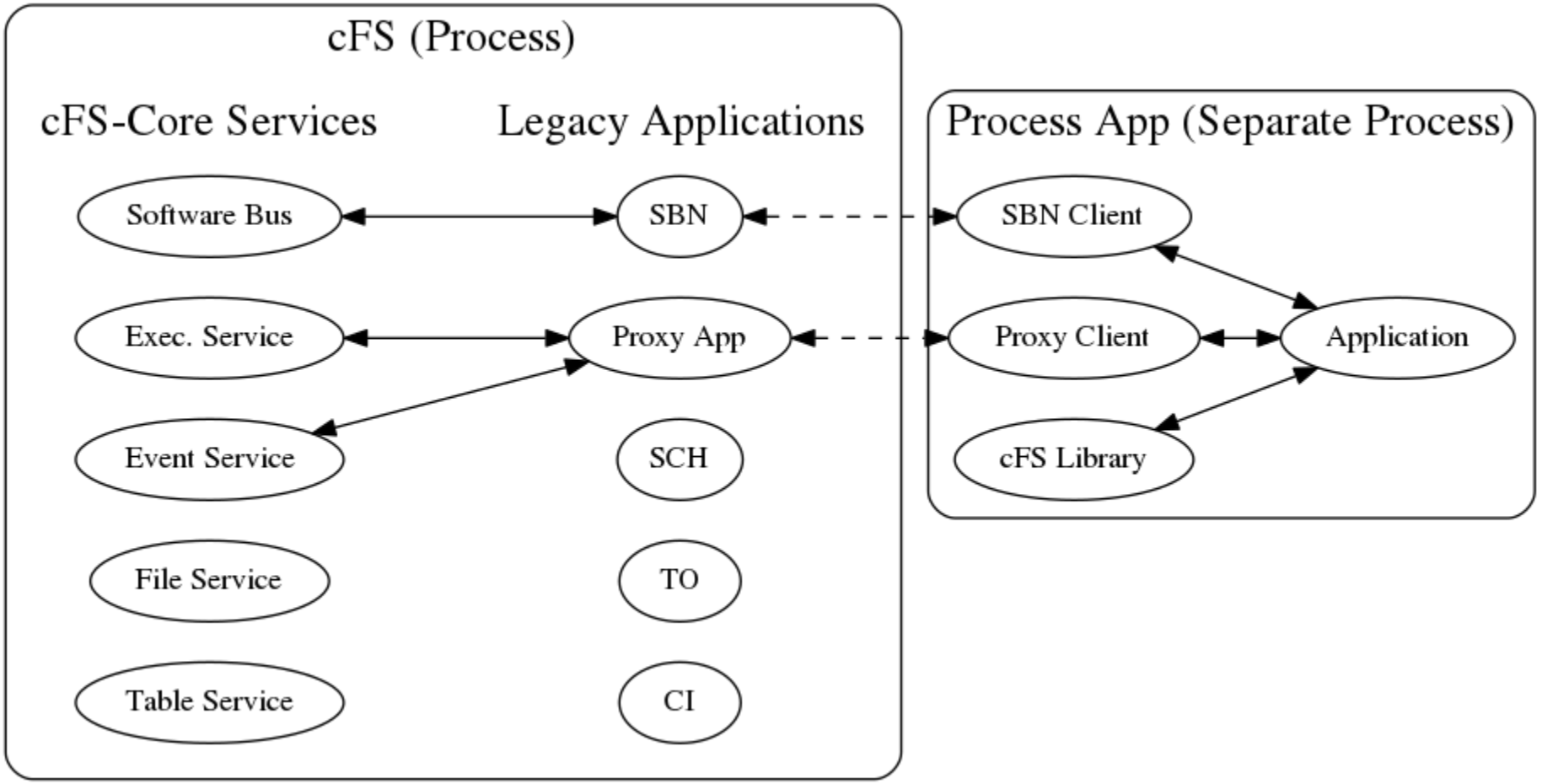
For functions that do use global data, the primary cFS instance must be called

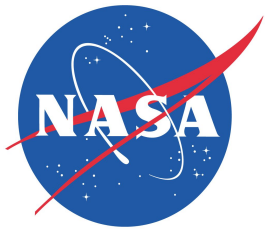
- This is an external process to the Process Application
- Examples: `CFE_ES_RunLoop` `CFE_EVS_SendEvent`

Needs to implement expected behavior of a normal application

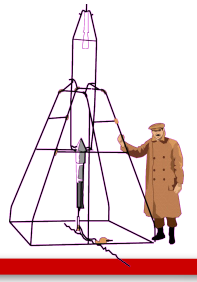
- Example: kill the Process Application when commanded to exit

Proxy Application





Proxy Implementation



Remote Procedure Calls

- Process Application -> Proxy Client -> FlatBuff -> NNG -> Socket -> ...

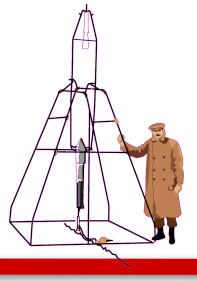
Wrapping functions

- All functions already exist in cFS Library, so must wrap with the linker
- cFS Library defines `main`, redefine for application

Proxy cFS Application

- Fork/Exec Process Application
- Currently handles process management
- Could be expanded with features like rate limiting, limit checks, voting

Language Support



Language Support

Small FY20 Internal Research and Development project

Built upon multiple process work

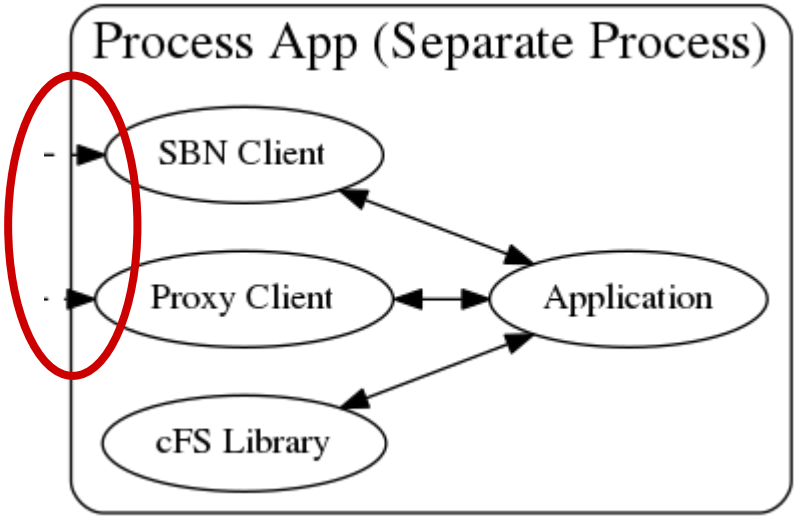
Some development work required

Large focus on finding a suitable project to use Python/cFS Integration

Implementation

So far, everything has been in C

I *thought* the best place to support other languages were the sockets



Implementation

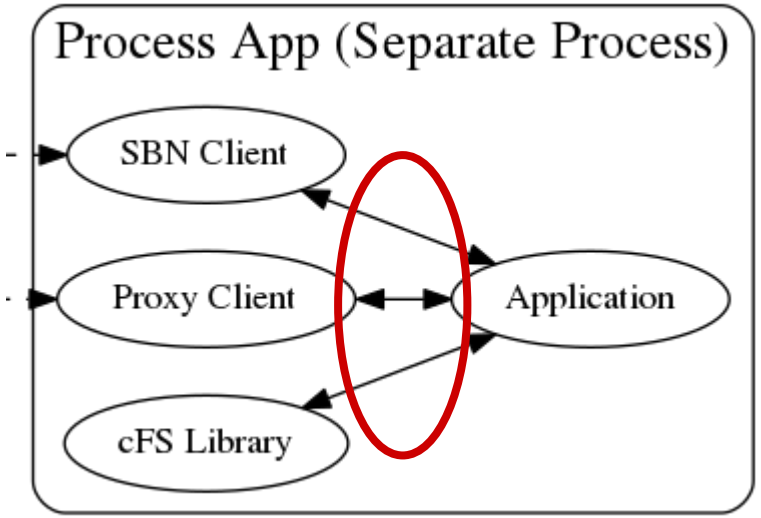
So far, everything has been in C

I *thought* the best place to support other languages were the sockets

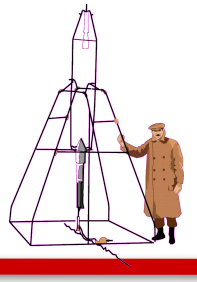
But we already have the C libraries

Most languages support calling C

Have to write the foreign function interface



Proofs of Concept



Python works!

Python command line interface ->

Rust also works!

Uses bindgen for function interface

Everything is `unsafe` (for now)

python

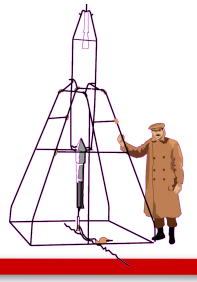
```
cFS> help

Documented commands (type help <topic>):
========================================
EOF   load_proxy            proxy_evs_register
exit  load_sbn              proxy_evs_sendevent
help  proxy_es_registerapp  sbn_recv

cFS> load_sbn
SBN Client library loaded: '<CDLL 'sbn_client.so'
910>'
SBN_Client Connecting to 127.0.0.1, 1234
SBN Client init: 0
SBN Client command pipe: 0
SBN Client subscribe msg (id 6366): 0
cFS>
```

RAISR and Ongoing Work

RAISR



Research in Artificial Intelligence for Spacecraft Resiliency

FY20 and FY21 Internal Research and Development projects

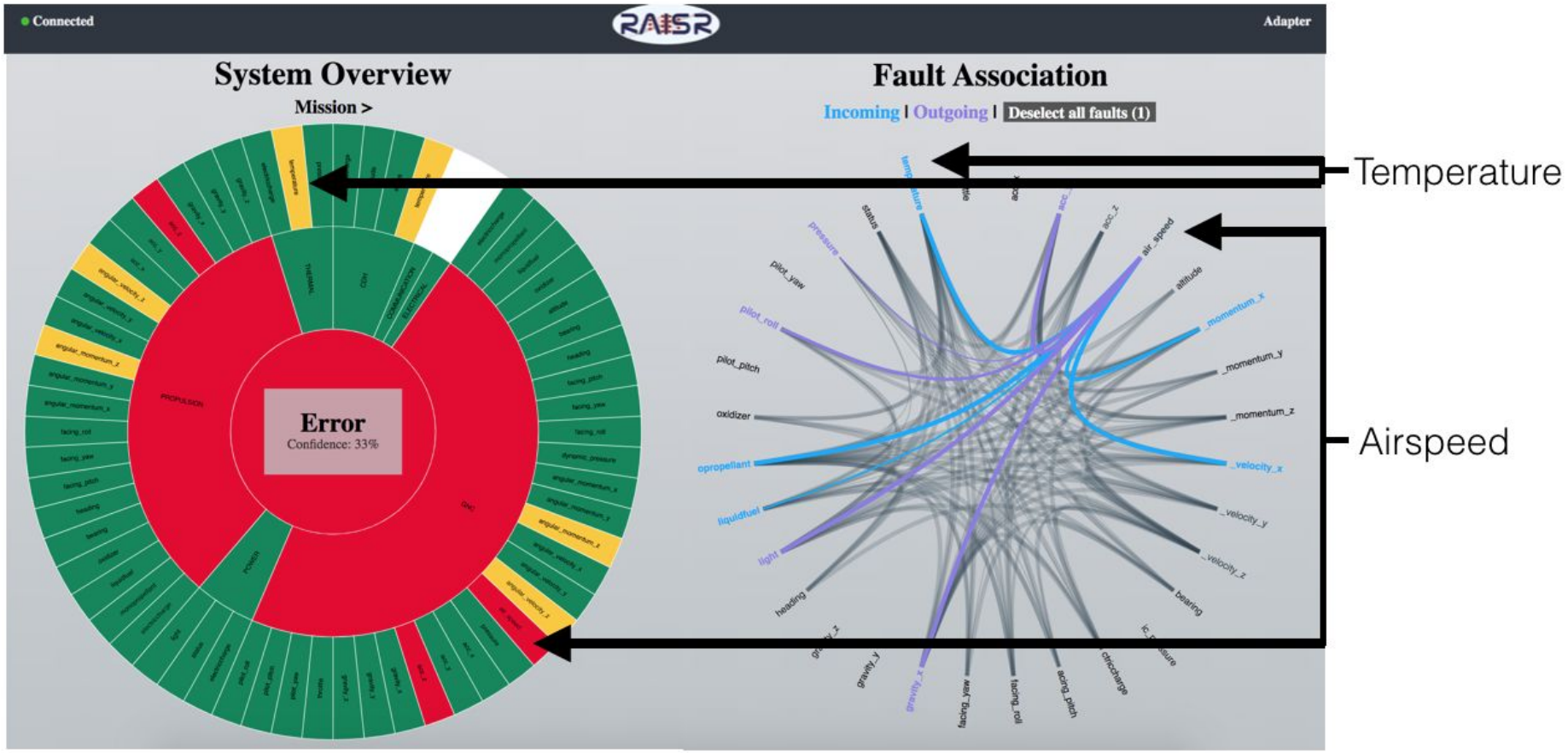
Evana Gizzi (Evana.Gizzi@nasa.gov) is the Principal Investigator

Uses flight telemetry to detect anomalies using Artificial Intelligence

Developing new AI-based algorithms

Python

RAISR in Action



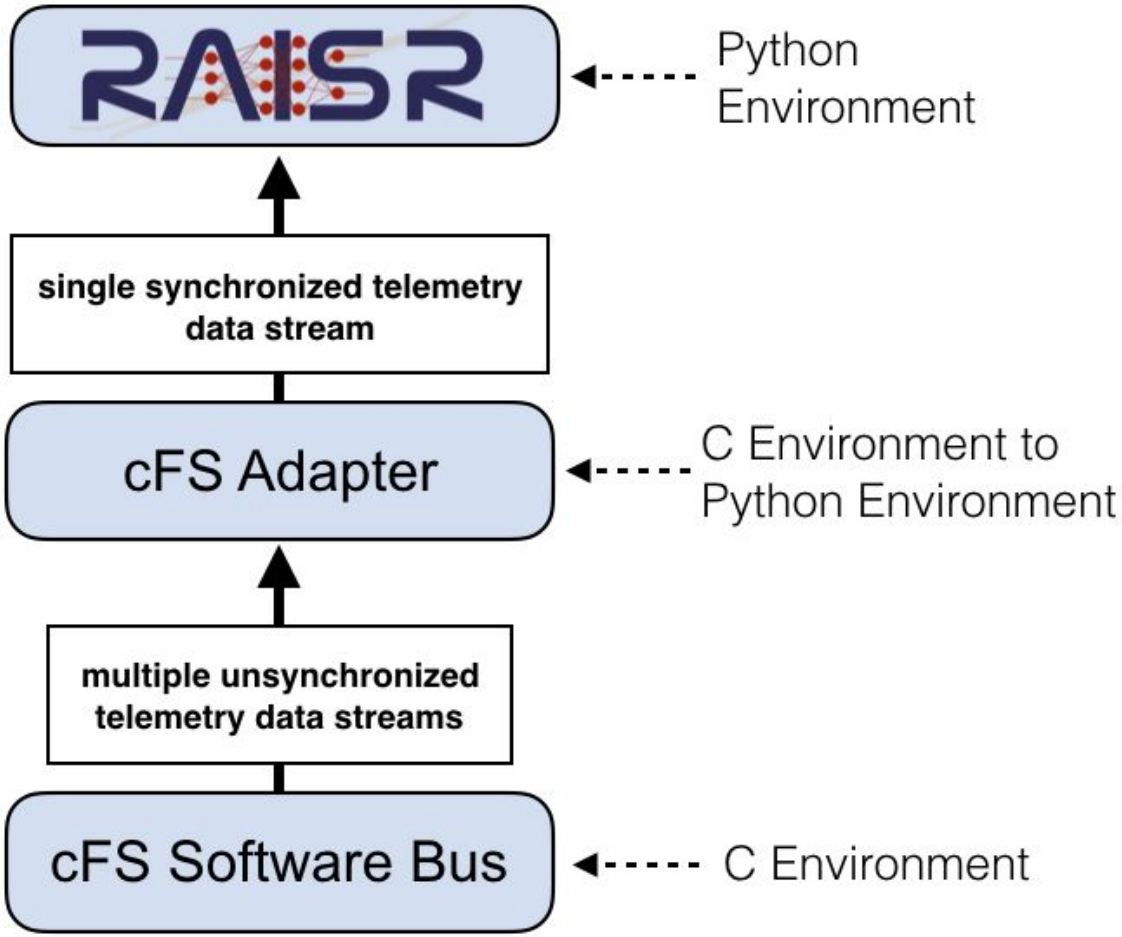
Mission status is shown on the left, telemetry associations on the right

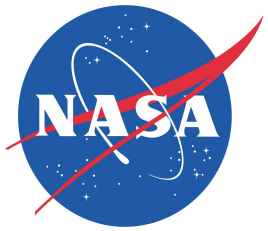
RAISR with cFS

Demonstrated RAISR running on a simulated satellite

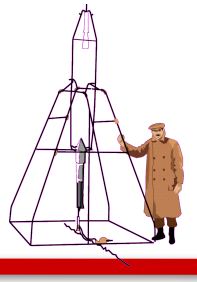
NOS3: https://github.com/nasa/nos3

Currently porting to flight-like hardware





Plenty Left TODO



Continue development with RAISR (funded for FY21!)

Improving and automating integration steps

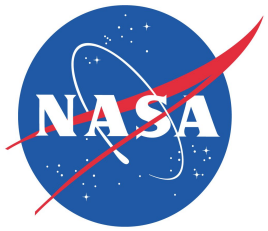
Better sandboxing of the Python application

Open Source

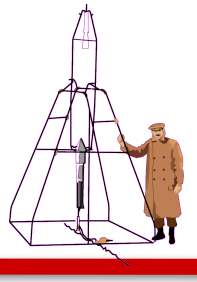
Tangential: exploring integrating Rust with cFS

Questions?

Contributors & Acknowledgments



Contributors & Acknowledgements



James Marshall (james.marshall-1@nasa.gov): Principal Investigator for Multi-Process and language work

Evana Gizzi (Evana.Gizzi@nasa.gov): Principal Investigator for Research in Artificial Intelligence for Spacecraft Resiliency

Alan Gibson and Nathan Riolo: Goddard employees who contributed to the multi-process and language support work

Christopher Chapman: summer intern contributed to the language work

Internal Research and Development program

cFS developers and maintainers