Flight Software Branch

Flight Software Workshop 2007 (FSW-07)

Current and Future
Flight Operating Systems

Alan Cudmore

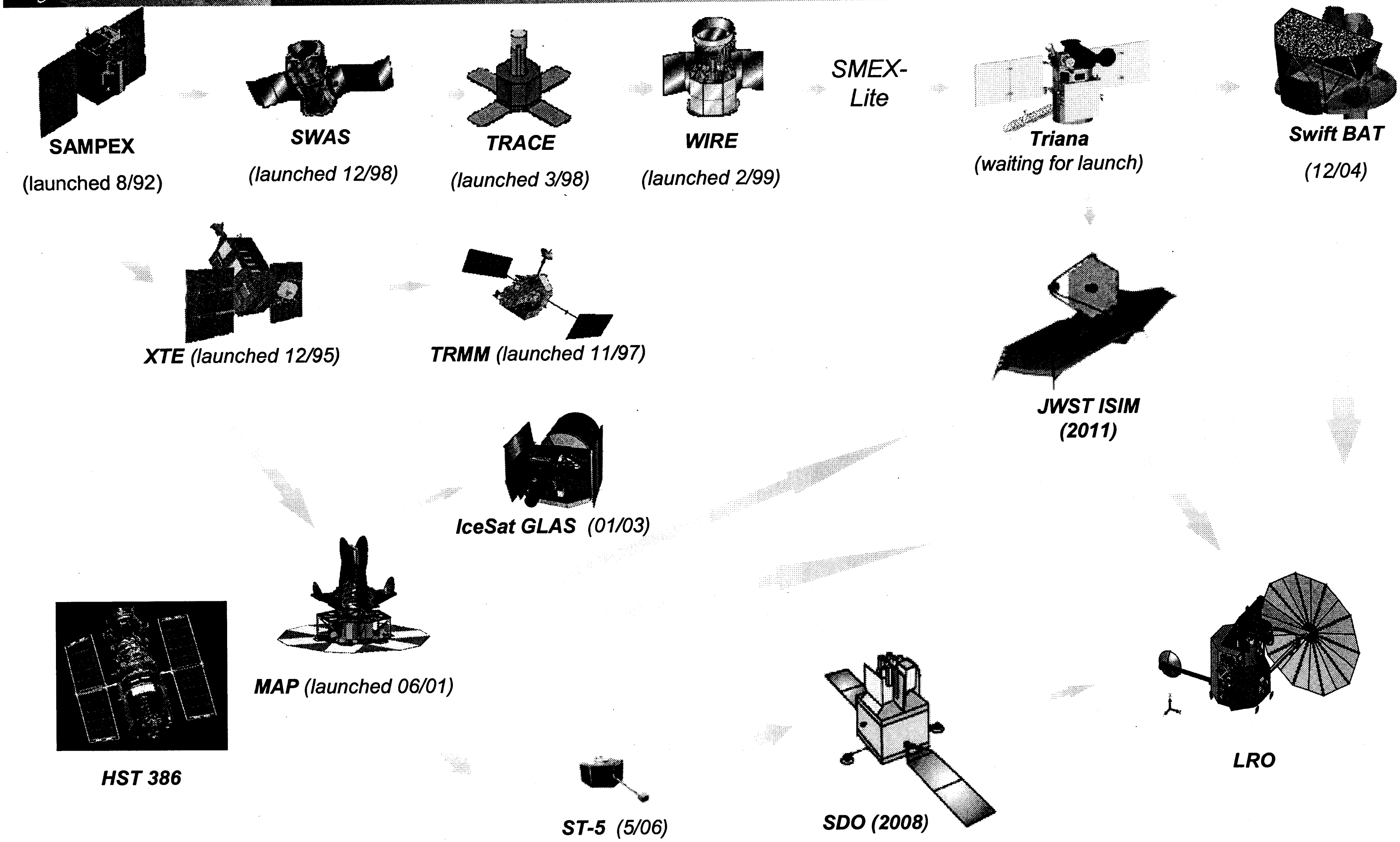
Flight Software Branch

NASA/GSFC

Outline

- **Types of Real Time Operating Systems**
  - Classic Real Time Operating Systems
  - Hybrid Real Time Operating Systems
  - Process Model Real Time Operating Systems
  - Partitioned Real Time Operating Systems
- **Is the Classic RTOS Showing it's Age?**
- **Process Model RTOS for Flight Systems**
- **Challenges of Migrating to a Process Model RTOS**
- **Which RTOS Solution is Best?**
- **Conclusion**

GSFC Satellites with COTS Real Time Operating Systems



Classic Real Time OS

- **What is a "Classic" RTOS?**
  - **Developed for easy COTS development on common 16 and 32 bit CPUs.**
  - **Designed for systems with single address space, and low resources**
  - **Literally Dozens of choices with a wide array of features.**

Terms:

OS	= Operating System
RTOS	= Real Time Operating System
COTS	= Commercial, Off the Shelf
CPU	= Central Processing Unit
MMU	= Memory Management Unit
Kernel	= An Operating System Core
POSIX	= Portable Operating System Interface
GSFC	= Goddard Space Flight Center
cFE	= GSFC's core Flight Executive

Classic RTOS - VRTX

- **Ready Systems VRTX**
- **Size: Small - 8KB RTOS Kernel**
- **Provides: Very basic RTOS services**
- **Used on:**
  - **Small Explorer Missions**
    - Used from 1992 to 1999
    - 8086 and 80386 Processors
  - **Medium Explorer Missions**
    - XTE (1995) TRMM (1997)
    - 80386 Processors
  - **Hubble Space Telescope**
    - 80386 Processors
- **Advantages:**
  - **Small, fast**
  - **Uses 80386 memory protection -- A feature we have missed since we stopped using it!**
- **Current use:**
  - **Only being maintained, not used for new development**

Classic RTOS - Nucleus

- Accelerated Technology Nucleus RTOS
- Size: Small < 64Kbyte RTOS Kernel
- Provides: Very basic RTOS services
- Used on:
  - Hubble Space Telescope Solid State Recorder
    - Mongoose 1 processor
- Advantages:
  - Small
  - Written in C
  - Source Code included
  - Add-ons available for Network, File system, etc
- Current use:
  - Used for some GSFC Rad Hard Coldfire GPS applications

Classic RTOS - vxWorks

- **Wind River Systems vxWorks RTOS**
- **Size: Medium - Large > 100Kbyte RTOS Kernel**
- **Provides: RTOS Services, DOS file system, Network Stack, Debugging features**
- **Used on:**
  - **MAP, EO-1, GLAS**
    - Mongoose 5 processor
    - Static memory map
  - **Triana, Swift/BAT**
    - RAD6000 processor
    - C++ Flight Software, Dynamic loading, file systems
  - **SDO, LRO**
    - RAD750 Processor
    - SDO using vxWorks 5.x, static memory map
    - LRO using vxWorks 6.x, dynamic loading, file systems
- **Advantages:**
  - **"Standard" RTOS**
  - **Wide support for debug tools, BSPs, add-ons**
  - **Dynamic loading, File Systems, Network Stack**
  - **Migration path to Memory Protected Process Model**
- **Current Use:**
  - **Baseline for all RAD750 Missions**

Flight Software Branch

Classic RTOS - RTEMS

- **OAR Inc - Real Time Executive for Multiprocessor Systems**
- **Size: Medium - Large > 100Kbyte RTOS Kernel**
- **Provides: RTOS Services, DOS file system, Network Stack**
- **Used on:**
  - **ST-5**
    - Mongoose 5 processor
    - Static Memory Map
  - **Themis**
    - Coldfire RH-5208 Processor
    - Static Memory Map
  - **SDO**
    - 5 Coldfire RH-5208 Processors
    - Static Memory Map
- **Advantages:**
  - **Open Source (free to download and use)**
  - **Written in C**
  - **Source Code included**
  - **POSIX APIs**
  - **Very Similar to vxWorks kernel**
- **Current Use:**
  - **Being used for RH-5208 Coldfire and SPARC/Leon applications**
  - **Used in labs where license fees are prohibitive**

Hybrid Real Time OS

- **What is a "Hybrid" Real Time OS?**
  - A Hybrid Real Time OS is an Operating System that has features of both the Classic RTOS and the Process Based Operating System.

- **vxWorks 6.x**
  - vxWorks 5.x features + Memory Protected "Real Time Process"
  - Backwards compatibility with vxWorks 5.x and RTOS Tasks
  - Single Physical Address space for Real Time Process
  - Growing number of POSIX Programmer interfaces

- **Real Time Linux**
  - RTAI Linux, Wind River Real Time Core for Linux (RT Linux)
  - Modified Linux Kernel running on top of a Classic RTOS. The underlying RTOS will schedule the Linux Kernel as a task.
  - Hard Real Time tasks run on the RTOS and can communicate with the standard Linux Processes.

- **Current or Planned Use:**
  - vxWorks 6.x is being used on LRO and JWST. Use of Real Time Processes are being considered.

Flight Software Branch

Process Model Real Time OS

- **What is a Process Model RTOS?**
  - **Implements a POSIX/Unix Style Process with memory protected virtual address space.**
    - Processes run in the CPU non-privileged user mode.
    - Device drivers and kernel code run in the privileged kernel mode
  - **Requires a CPU with Memory Management Unit**
    - PPC, x86, ARM, etc.
  - **Provides POSIX Programming Interfaces**
  - **Provides a Real Time Scheduler**
  - **Typically require more Memory and CPU power than a Classic RTOS**

- **Examples of Process Model RTOSs**
  - **Lynx OS**
  - **QNX Neutrino**
  - **Green Hills Integrity**
  - **Linux - Near Real Time variants: TimeSys, RedHawk**

Flight Software Branch

Partitioned Real Time OS

- What is a Partitioned Real Time OS?
  - System is split into multiple virtual partitions to isolate critical tasks/processes
  - Memory and CPU time can be bound for each partition
  - Critical applications in one partition cannot be affected by applications in another partition
- ARINC 653 Standard
  - The ARINC 653 standard specifies the interface and services for safety critical partitioned operating systems
  - Most Partitioned RTOSs follow the ARINC 653 standard
- DO-178B Standard
  - Many partitioned systems are also DO-178B certifiable for safety critical systems.
  - DO-178B is a standard for software development for safety critical systems.
  - A DO-178B certifiable system does not have to be an ARINC 653 system.
- Examples of Partitioned RTOSs
  - LynxOS 178B
  - LynxOS SE (Non 178B)
  - BAE CsLEOS
  - Green Hills Integrity 178B
  - Wind River Platform for Safety Critical ARINC 653

Is the Classic RTOS showing it's age?

- Classic Real Time Operating Systems with shared memory space have been used successfully in flight missions for decades.
- But now we are adding:
  - TCP/IP Stacks
  - File Systems
  - File Transfer Agents
  - Middleware/OO Frameworks
  - Dynamic Loaders
  - Scripting languages
  - On-Board Science Data Processing
- As the size and complexity increase, so will the:
  - Chance for a bug or stray pointer to kill the system
  - Chance for a memory leak
  - Amount of time needed to find a bug
  - Amount of time it takes to start and reboot the system

- **How can we try to maintain reliability as these systems grow?**

Process Model RTOS for Flight Systems

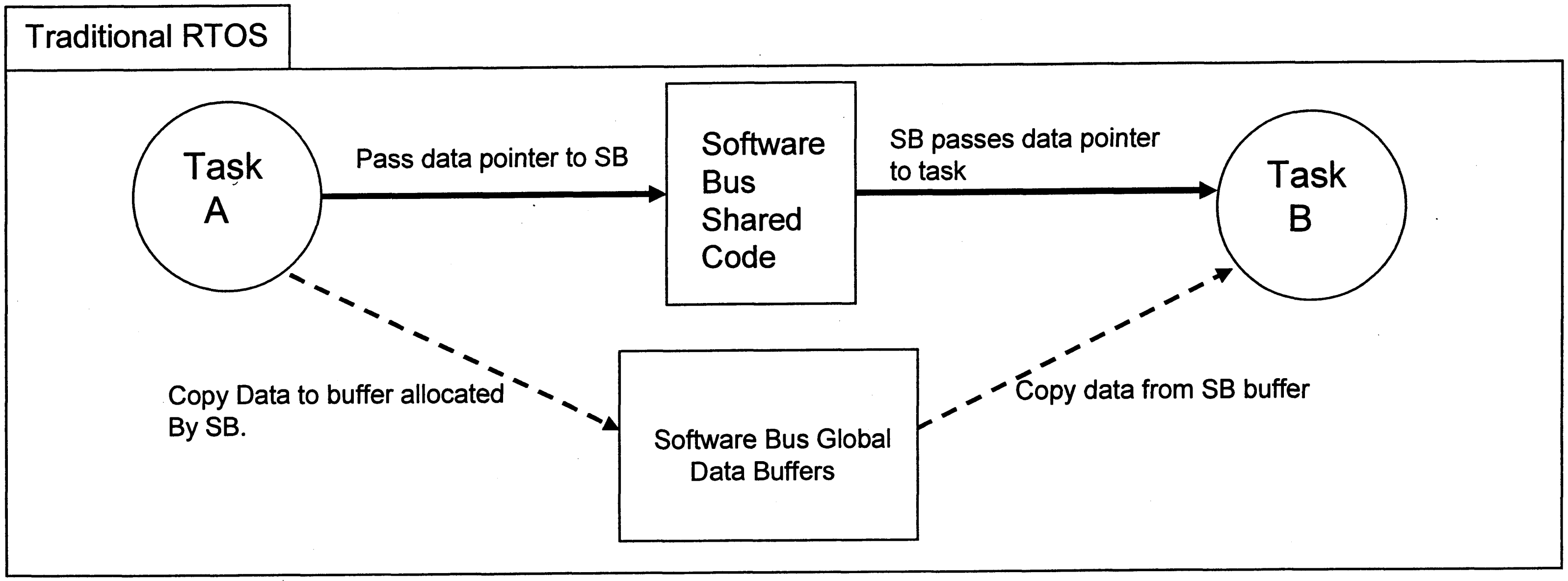
- **A Process Model RTOS can take advantage of the features in advanced CPUs to increase the reliability of flight software.**

- **Advantages of a Process Model RTOS**
  - Process based Memory Protection
  - Ability to map around bad memory
  - Page based dynamic memory allocation/deallocation
  - Forced application / device driver separation
  - Explicit code/data sharing and encapsulation

- **Given some advantages, what are the challenges of migrating flight software to a Process Model RTOS?**

Challenges of Migrating to a Process Model RTOS

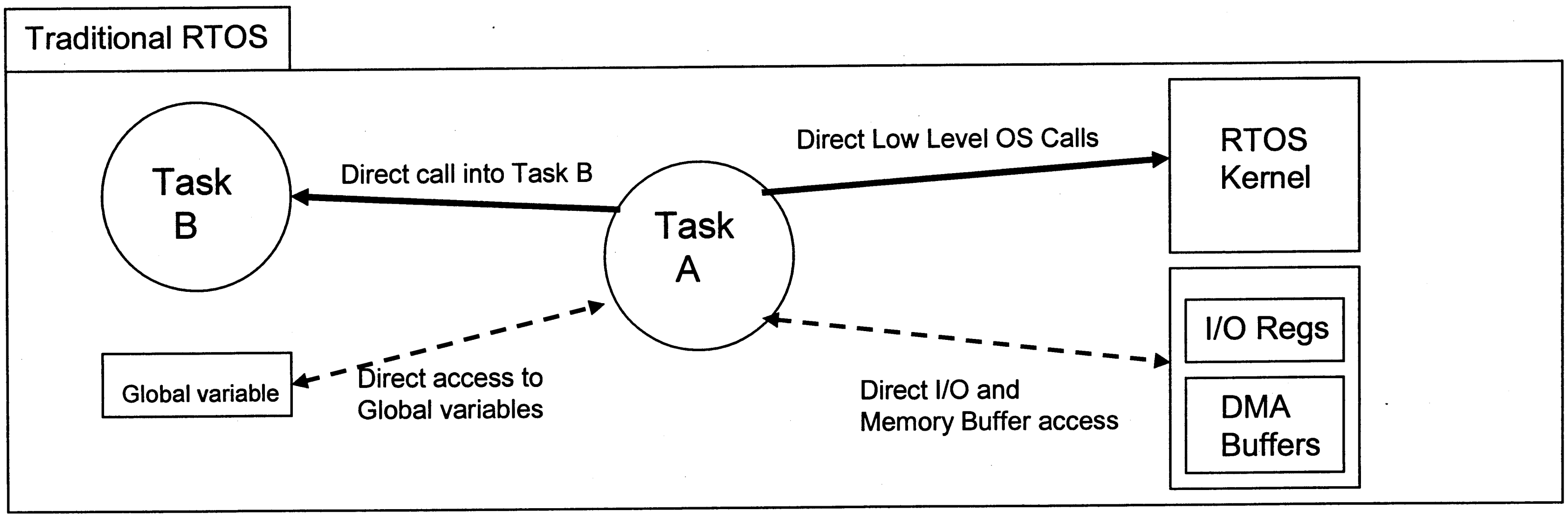
- **Inter-process Communication and shared memory**
  - **Example : GSFC Software Bus**



- **Potential solutions:**
  - **Create Shared memory segments for Software Bus Global Memory and Buffers**
    - Cannot use pointers with absolute addresses, must use offsets
  - **Send the entire message via SB / Inter-process Communication**
    - Overhead in copying the data, but less chance for pointer corruption issues

Challenges of Migrating to a Process Model RTOS

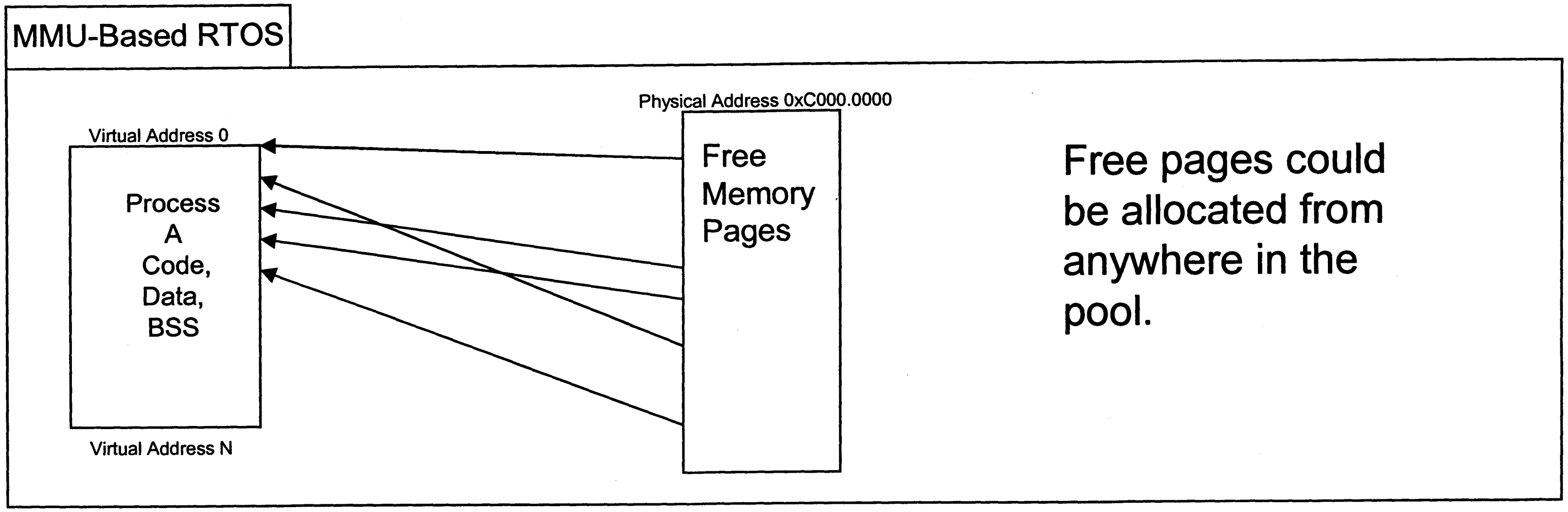
- **Device Drivers, I/O, and Memory Access**



- **Potential Solutions**
  - **Low level device access through device drivers**
    - Applications use device driver API to access hardware
  - **I/O remapping calls**
    - Some Operating Systems have calls to map I/O space into the process memory map
  - **Shared memory segments, Shared Libraries**
    - Better way to share code and data

Challenges of Migrating to a Process Model RTOS

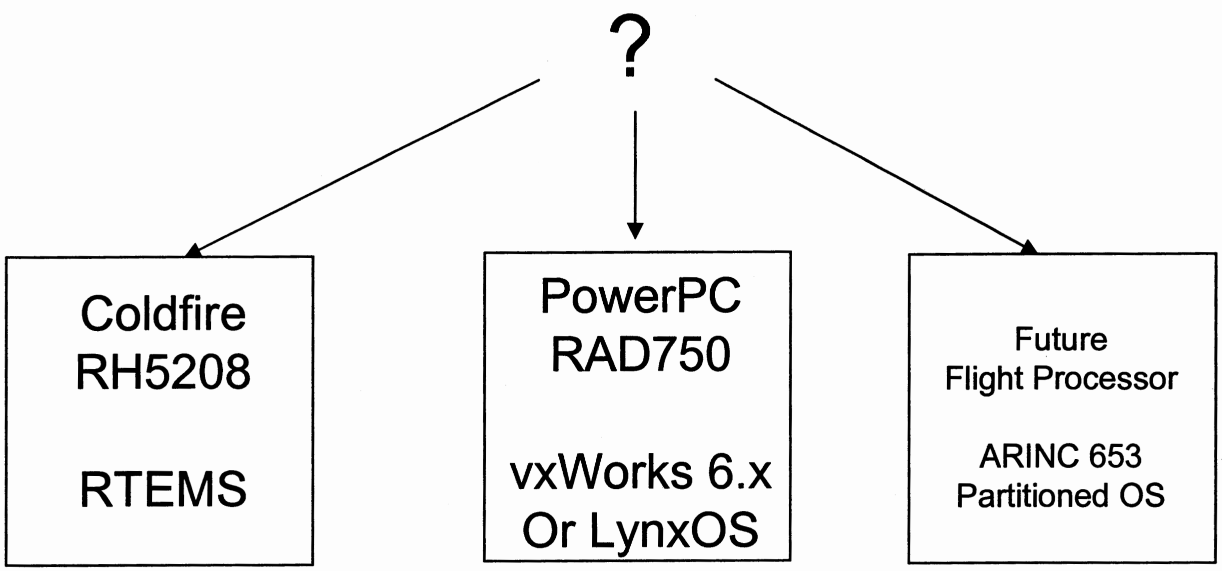
- **Memory Map Issues**
  - **FSW Maintenance teams patch software by using memory maps and absolute addresses.**
  - **A process running in a protected virtual address space may have it's memory pages allocated from anywhere in the pool of available pages using the MMU.**



  - **Options for patching memory?**
    - It should be possible to get a page map for a process in memory and determine what pages it has allocated.
    - Safer options include patching on disk executable and restarting the process.

Which RTOS solution is best?

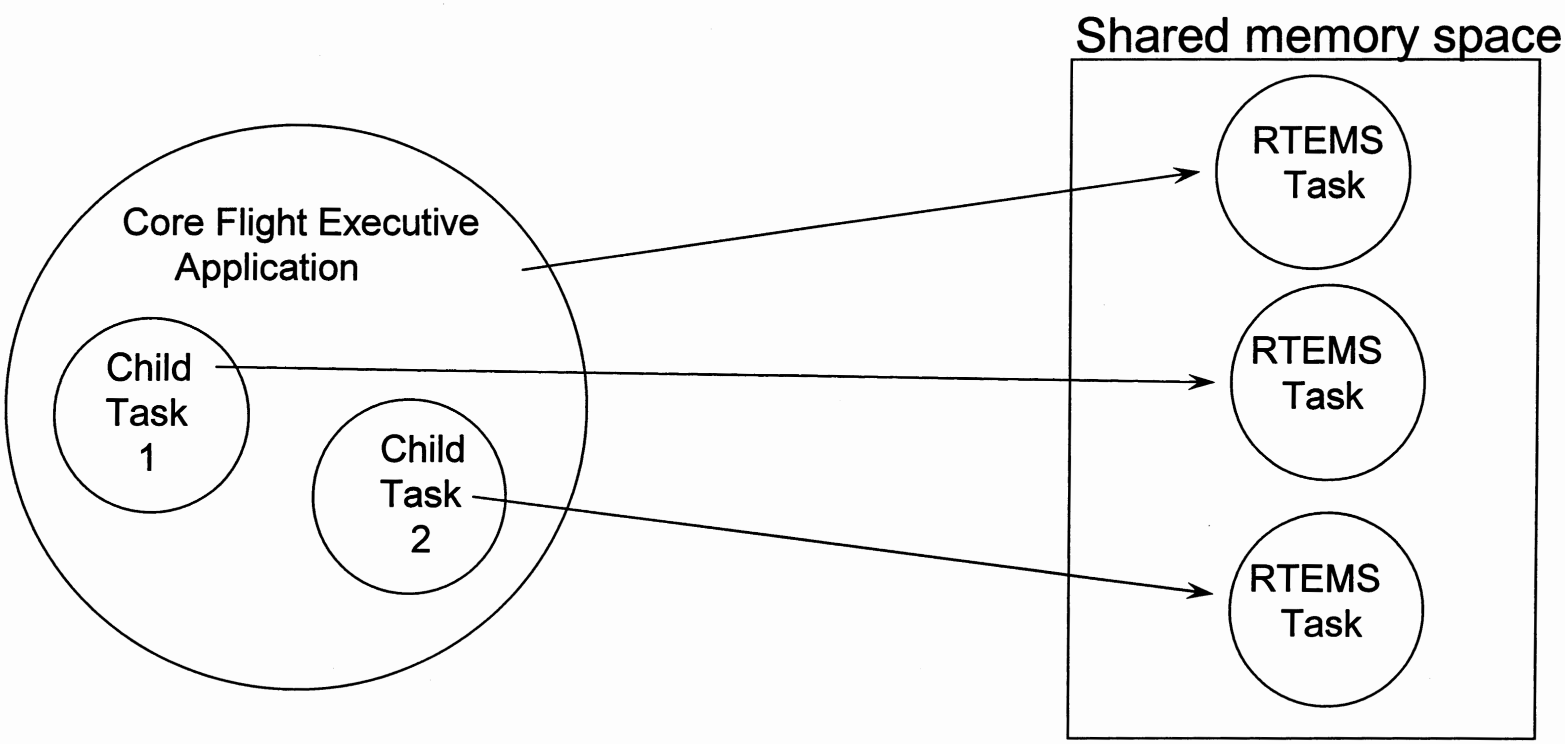
- **For the foreseeable future, it looks like we will need all three types of Real Time Operating Systems**
  - **Classic RTOS for CPUs without a MMU - Small Instrument, Low Power applications**
  - **Process Model RTOS for more powerful CPUs - C&DH Systems, “Flight Server”**
  - **Partitioned RTOS for Safety Critical / Manned Applications**



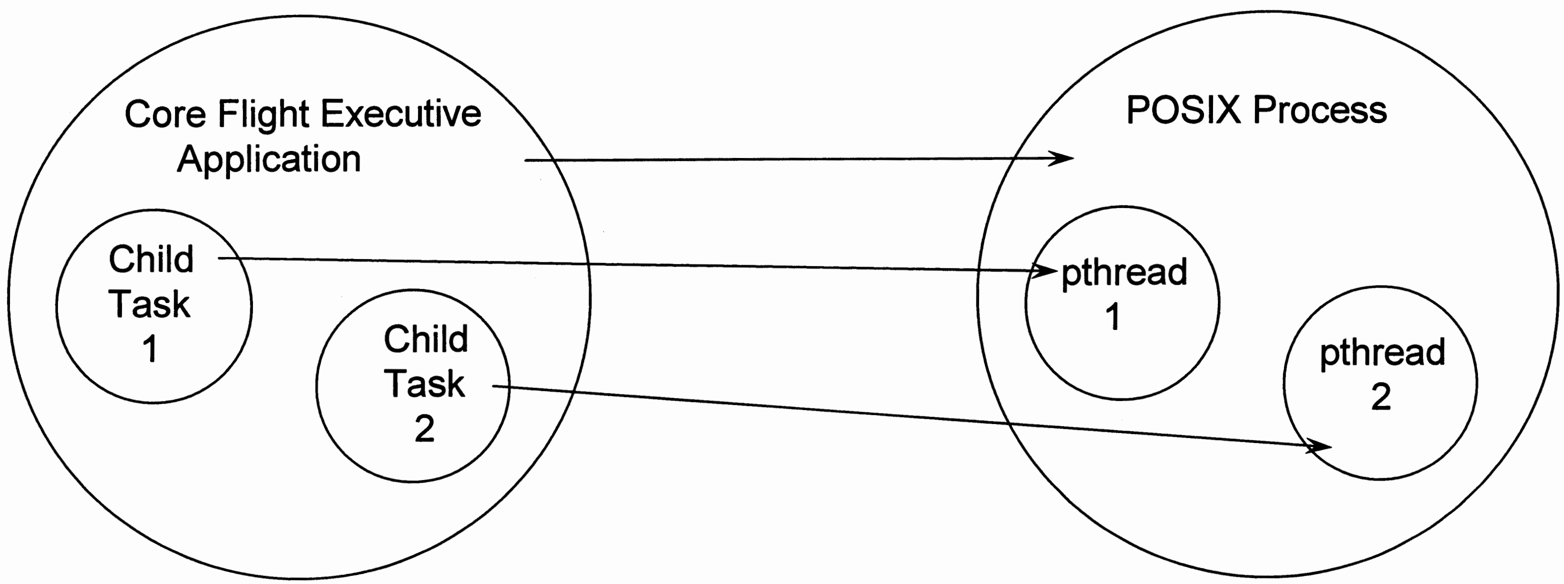
- **How do we manage the Flight Software for these three RTOS models?**

Core Flight Executive App on a Classic RTOS

- The GSFC core Flight Executive (cFE) uses an OS Abstraction Layer to isolate it from the RTOS.
- The cFE maps the Application's main thread to an RTOS task
- The cFE maps each Child task to an RTOS thread
- There is no protection from the rest of the tasks in the system

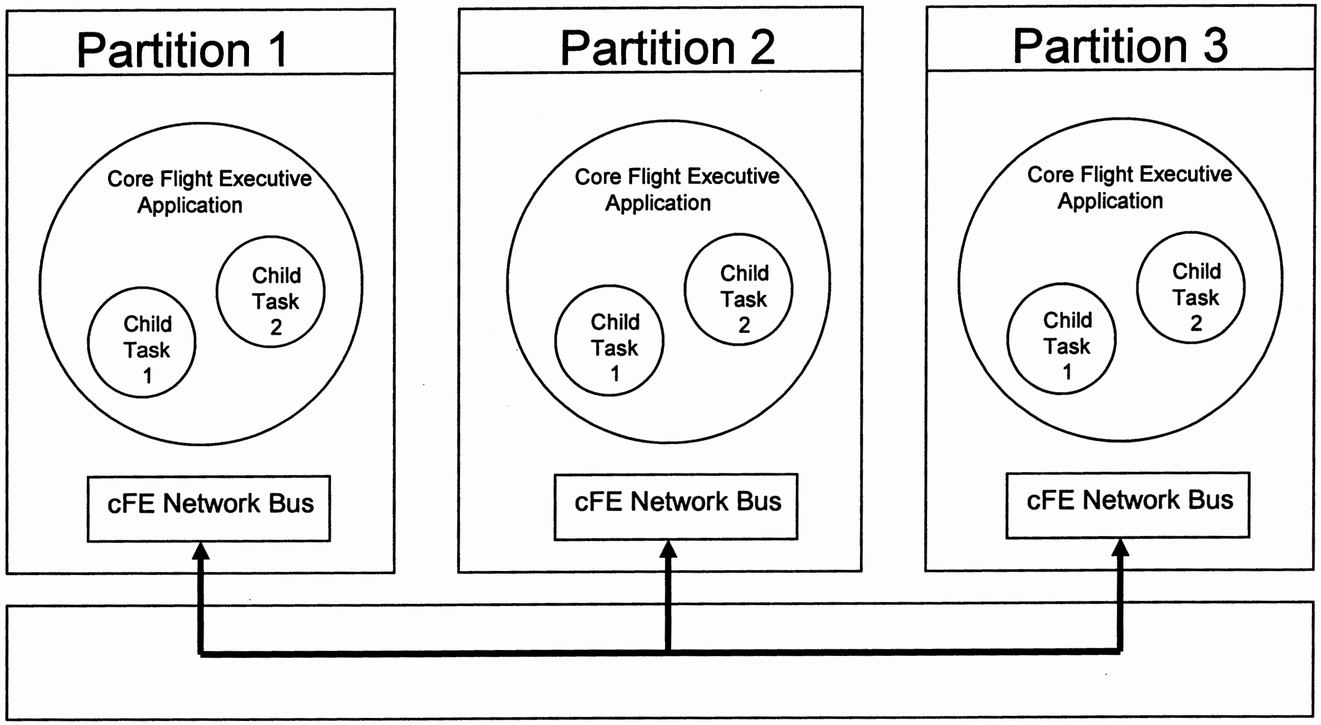


Core Flight Executive App on a Process Model RTOS



- **On a Process Model RTOS, a Core Flight Executive Application maps to a memory protected process**
- **Each cFE child task maps to a thread within the process**
- **The cFE process is isolated from the rest of the memory in the system**

Core Flight Executive App on a Partitioned RTOS



- **On a Partitioned RTOS, each partition looks like a separate processor to the core Flight Executive.**
- **This model could have one cFE Core per partition communicating via the Network Bus application.**

Conclusion

- Although the future is in the use of Process Based RTOSs in flight software, we still need to use Classic RTOSs for small/low power processors.
- The use of an OS abstraction layer and a portable Flight Software architecture such as the core Flight Executive can help ease the transition from one type of RTOS to another and promote software reuse.

- Questions?