



# Flight Software Workshop 2008 (FSW-08)

## 20 Years of Porting Flight Software

**Alan Cudmore**  
**Flight Software Branch**  
**NASA/GSFC**



# Contents

- **Flight Software Porting History**
- **Trends that Contribute to Flight Software Portability**
- **Tips for Porting Existing Flight Software**
- **Designed for Portability: The Core Flight Executive**
- **Lessons Learned**
- **Conclusion**

Term	Definition
API	Application Programmer Interface. A library function call.
RTOS	Real Time Operating System
OSAL	Operating System Abstraction Layer
cFE	NASA GSFCs Core Flight Executive
COTS	Commercial Off The Shelf
BSP	Board Support Package



# Flight Software Porting History

Mission	Processor Architecture	Operating System	Compiler	Dev. Host	FSW Architecture
SAMPEX	I386, 8086	VRTX	Metaware High C	MS-DOS!	GSFC Software Bus
XTE/ TRMM	I386	VRTX	Metaware High C	DOS/ Windows	GSFC Software Bus
HST Recorder	Mongoose I ( MIPS )	Nucleus	BSO Tasking C	DOS/ Windows	GSFC Software Bus
MAP	Mongoose V ( MIPS )	vxWorks	GNU C	Windows NT	GSFC Software Bus
SDO	Coldfire	RTEMS	GNU C	Linux	GSFC Software Bus
SDO	RAD750	vxWorks 5.5	GNU C	Solaris	GSFC Software Bus
LRO/GPM	RAD750	vxWorks 6.x	GNU C	Windows	cFE/CFS
MMS	Coldfire	RTEMS	GNU C	Linux	cFE/CFS



# Trends That Contribute To Flight Software Portability

- **High Level Languages**
  - 20+ years ago, most flight software was written in assembly language
  - The use of High Level Languages such as Ada, C, and C++ are the biggest aid to portability.
- **COTS Real Time Operating Systems**
  - We started using Commercial Off The Shelf ( COTS ) Real Time Operating Systems in 1989 with SAMPEX.
  - COTS Real Time Operating Systems offer:
    - Tasking/Context switching/Stack Management
    - Interrupt management
    - Message Queues
    - Semaphores
  - Most Real Time Operating Systems offer similar APIs, so porting is relatively easy
- **Standard Language Libraries ( C, C++ )**
  - Most embedded tool chains and RTOSs come with a C Standard Library
  - There are also standard math libraries available
- **Portable/Standard Operating System APIs**
  - The use of a standard Operating System API such as POSIX can help
  - We chose to create our own Operating System API
- **Standard device driver models**
  - Will start to play a bigger role in device driver portability
  - The standard Unix model of open/close/read/write/ioctl helps create portable device interfaces



# Tips on Porting Existing Flight Software (1)

- **Start with a working build of the code**
  - A build log serves as a good reference on how the software is compiled and linked
- **Understand the Build directories and Makefiles**
  - If the Makefiles are portable enough they can be reused and modified
- **Separate the code into high level and assembly code**
  - If possible just replace the assembly code functions with High Level Language stubs
  - Replace the Assembly with C ( or other High Level Language ) if possible
- **Try to get the code to compile without linking**
  - Isolate the Real time operating system interfaces and include files
    - They can be `#ifdef`'ed out to make sure the calls are not being compiled
  - Know where the include files are coming from
    - vxWorks, Standard C, Mission?
  - Try to compile the code with the workstation/PC compiler first
    - Compile with Linux GCC, or Windows MinGW, etc.
    - Worry about the new cross compiler or linker scripts later



# Tips on Porting Existing Flight Software (2)

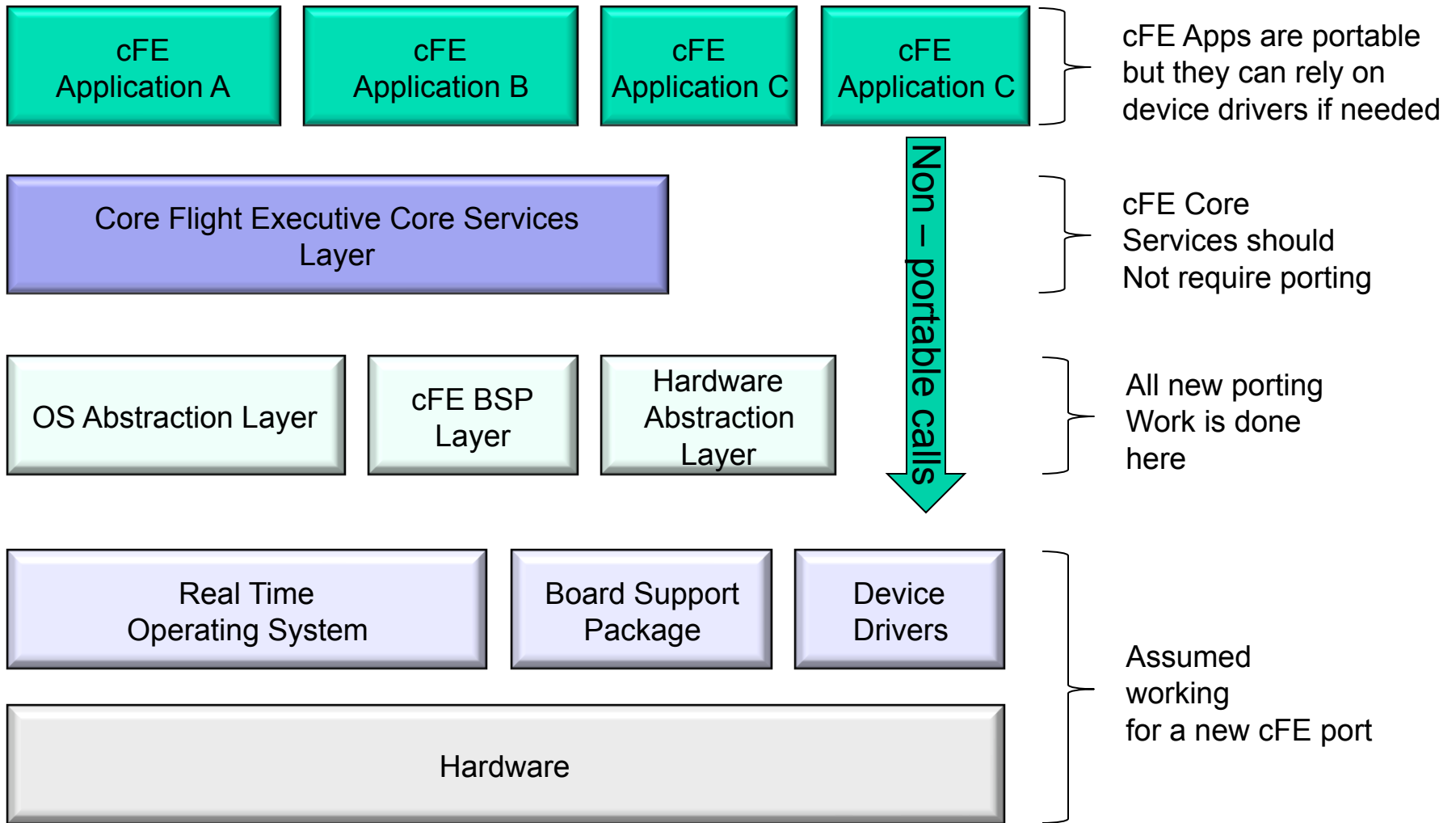
- **Watch out for endianness ( CPU byte ordering ) changes**
  - Will affect bit fields, structures, etc.
  - Big Endian CPUs: PowerPC, 68k/Coldfire, Mongoose V, Sparc
  - Little Endian CPUs: Some ARM, x86
- **Verify the pointer sizes**
  - x86 protected mode has 6 byte pointers.
- **Try to avoid Bit fields**
  - Can be a problem when changing compilers or CPU architectures
  - Has been a constant source of compiler bugs in the past
- **Know how the compiler is going to pad data structures**
  - The compiler may be aligning things without telling you
- **Look out for compiler specific pragmas and extensions**
  - Alignment, In-line assembly, structure packing are all commonly used
- **Data alignment**
  - Some compilers/architectures require the data structures to be aligned to avoid access exceptions



# Designed for Portability: The Core Flight Executive

- **The Core Flight Executive ( cFE ) was designed for portability**
  - Standard C with GSFC Coding Standards
  - Well thought out, layered APIs
  - Nearly everything is configurable through header files
  - OS Abstraction layer
  - Flexible and comprehensive build environment
- **Most cFE Applications require no porting at all**
  - As long as the cFE API is followed, no changes should be required
- **Some cFE Applications will have to be non-portable**
  - Hardware specific, or Device specific Applications have to be non-portable, but are fully supported.
  - There is nothing keeping a cFE application from directly calling OS specific APIs.
- **Much of the portability is handled in the build system**
  - The build system is often an overlooked part of development

# Designed for Portability: The Core Flight Executive



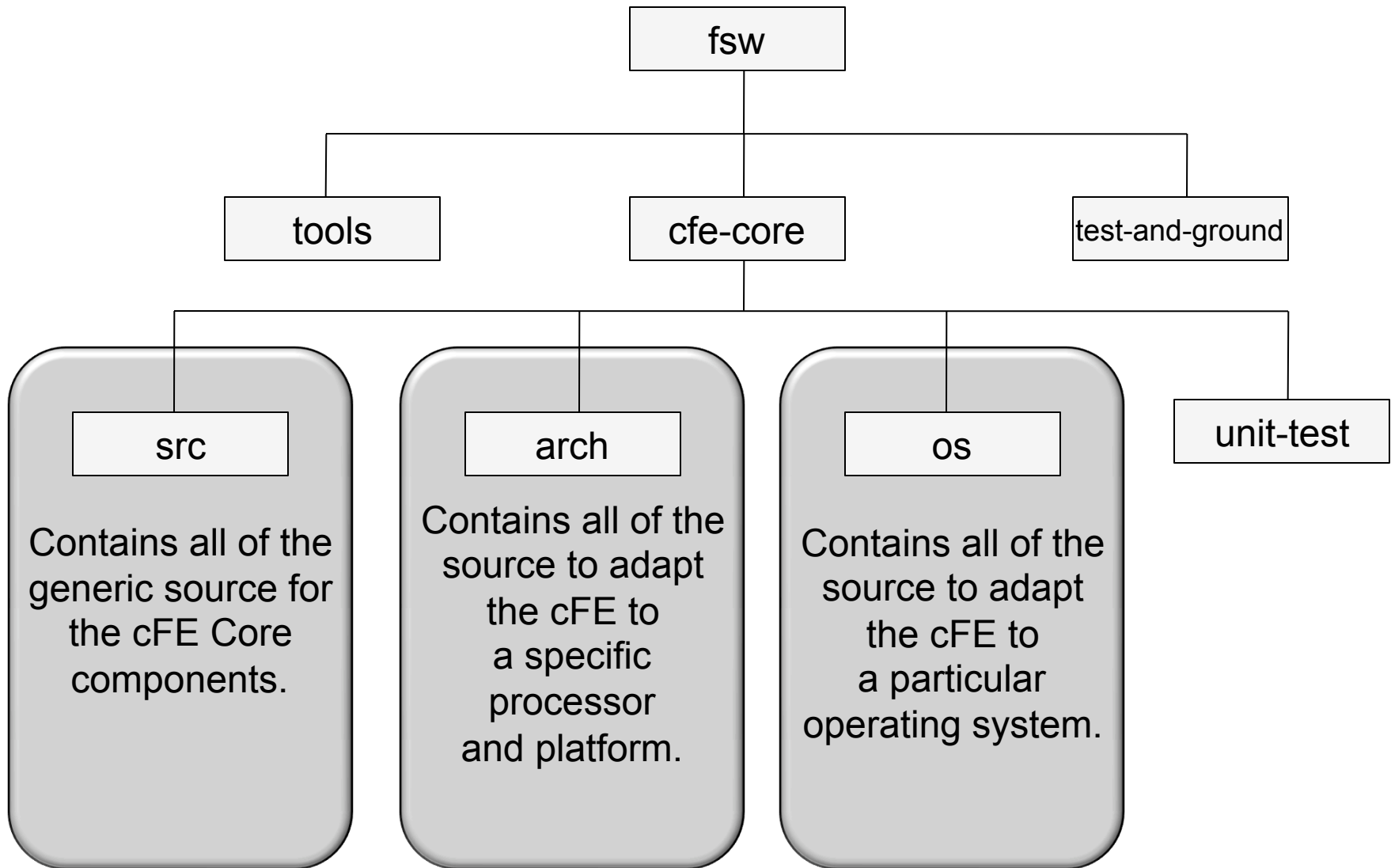


# Designed for Portability: The cFE Directory Structure

## The Directory Structure and Build Environment enhance the cFE's portability

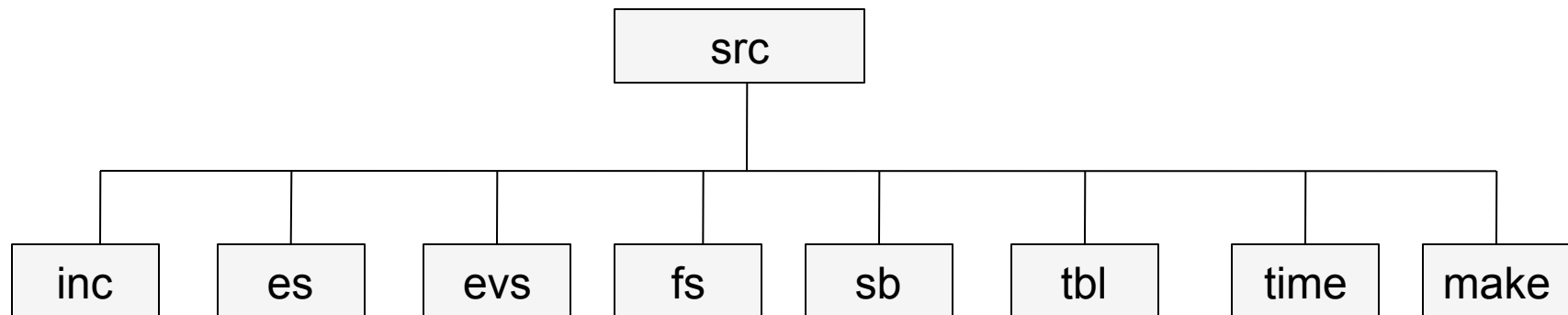
- **All non-portable source code is concentrated in specific directories**
  - OS Specific directories ( OS Abstraction Layer )
  - CPU Architecture Specific Directories
  - BSP Specific Directories
- **All source code is separated from the build products and configuration files**
  - cFE Core, OS Abstraction Layer, and cFE Applications are contained in source repositories that can reside almost anywhere on the development host.
  - The cFE Build directory structure can hold the build products and configuration files for any number of processors and configurations.
- **The directory structure and build environment are portable**
  - Many combinations of tools and development operating systems are supported
  - Relies on GNU make which is available on any system

# Designed for Portability: The cFE Source Tree (1)

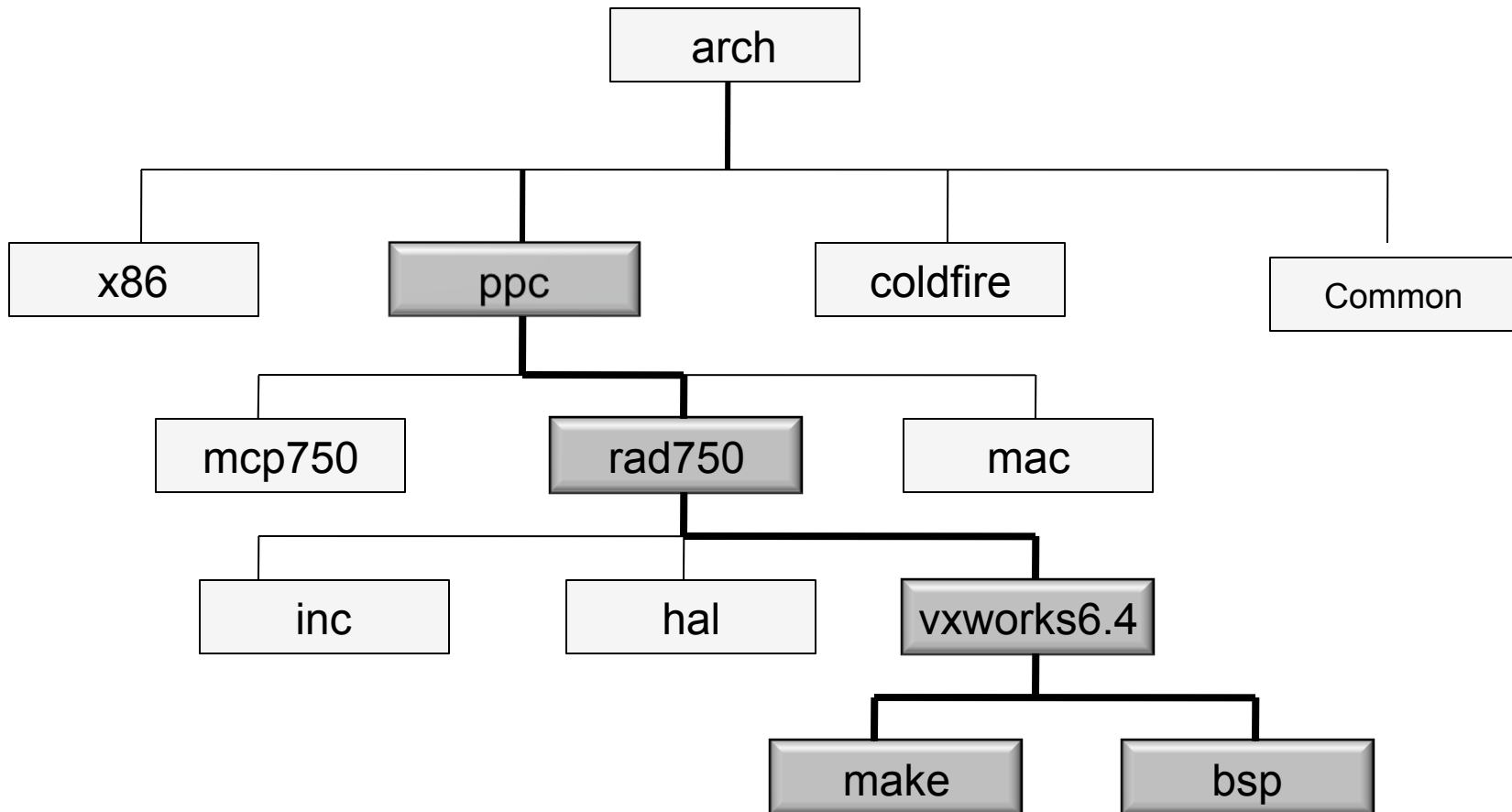


## Designed for Portability: The cFE Source Tree (2) – cFE Core

- The cFE Core “src” directory contains all of the generic components of the cFE core.
- There are no platform, processor, or architecture specific source files in these directories.



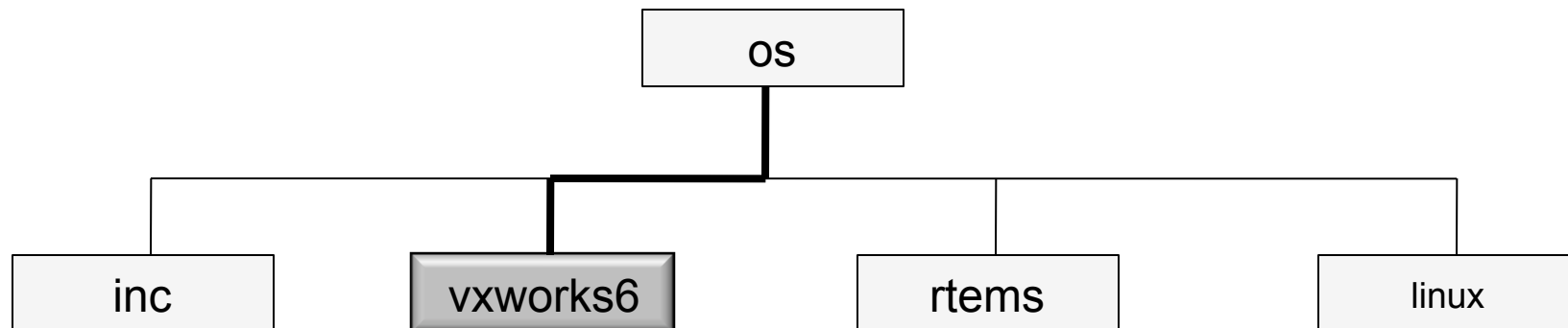
## Designed for Portability: The cFE Source Tree (3) – arch directory



- The “make” directory contains all of the RAD750/vxWorks6.4 specific compiler rules.
- The “bsp” directory contains all of the “Glue Code” to make the cFE work on the RAD750/vxWorks6.4 platform.

## Designed for Portability: The cFE Source Tree (4) – OS directory

- Each directory in the OS abstraction layer implements the generic OS API for a different “target” Operating System.
- The “vxworks6” directory, for example, implements the OS abstraction layer that the cFE uses for the vxWorks 6.4 Real Time Operating System.
- The cFE code uses the OSAL operating system services, which in turn call the correct “target” Operating System calls.



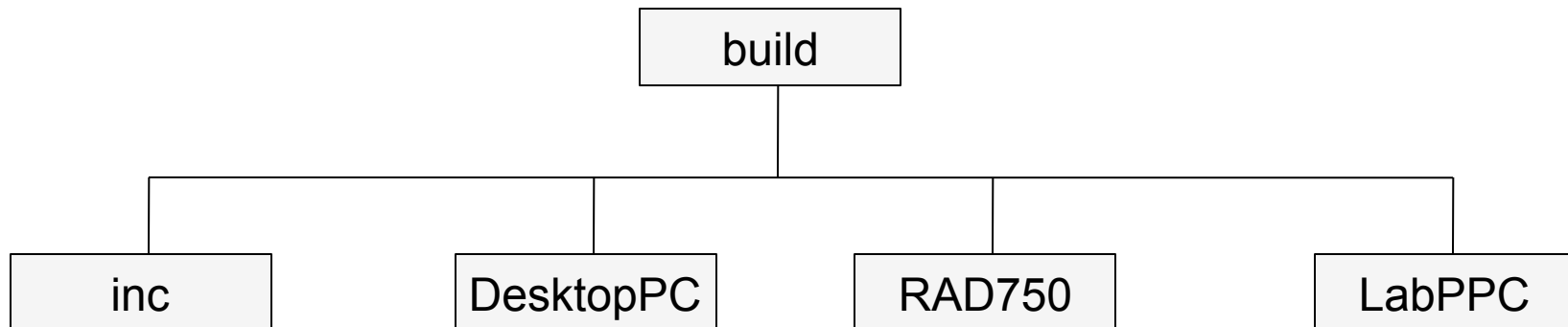
## Designed for Portability: The cFE Build Tree

The cFE build tree allows the configuration and building of any number of unique processor configurations. Each “cpu” can be:

- Any supported processor architecture
- Any supported platform
- Any combination of cFE applications
- Any supported cFE configuration

For example, a build directory can hold the configuration for:

- A Desktop PC/Linux prototype
- A lab Power PC COTS board
- A coldfire RTEMS board
- A RAD750/VxWorks flight configuration





# Lessons Learned During cFE / OSAL Development

- **The cFE cannot abstract everything**
  - It will take forever to abstract every single Device, API, etc.
  - A trade-off study needs to be done when adding a new abstracted API
  - Some Apps will have to use RTOS device drivers or APIs, but that is OK
- **From an OS to a middleware**
  - At first the cFE / OSAL was supposed to “contain” the RTOS.
  - In practice this became too hard to maintain for new releases of vxWorks and RTEMS
  - It is much easier to deploy the cFE on a already working RTOS/BSP combination
  - This makes it much easier to obtain standard hardware/RTOS products from a vendor ( i.e. RAD750/vxWorks )
- **File system layer of the OSAL was hard to get right.**
  - Original intention was to abstract the complete file system with Unix style paths and APIs
  - The implementation can be confusing to developers.
- **Sometimes the OSAL and cFE make things harder ( for me )**
  - As a developer and maintainer, putting out new cFE releases can be a lot of work due to the number of platforms that must be built and tested.
  - Since most ported platforms age quickly, the supported ports in the cFE can grow old and be less useful
  - The best way to combat this is to make the ports as generic as possible
    - i.e. Generic vxWorks, Generic RTEMS, Generic Linux, etc



# Conclusion

- **Portable flight software is a combination of best practices, solid engineering, and trade-offs ( speed vs. portability )**
- **In our opinion the cFE comes very close to achieving the “ultimate” portability, but there are still some trade-offs.**
  
- **Questions?**
  
- **Demonstration**