QNX SOFTWARE SYSTEMS

Portability Made Possible:
Creating Reusable Software Assets Through POSIX

Steve Furr, QNX Software Systems
Jason Clarke, QNX Software Systems

QNX Highlights

- **QNX: provider of realtime operating system (RTOS) software, development tools and services for mission-critical embedded applications**
  - 24 years of realtime embedded experience
  - Millions of installations worldwide
  - Reputation for reliability and scalability
- **Leader in innovative embedded technology**
  - First multitasking RTOS running with MMU support
  - First RTOS to implement distributed processing
  - First RTOS to implement symmetric multi-processing
  - First POSIX certified RTOS
  - First microGUI windowing system for embedded systems

Agenda

POSIX: Standard APIs

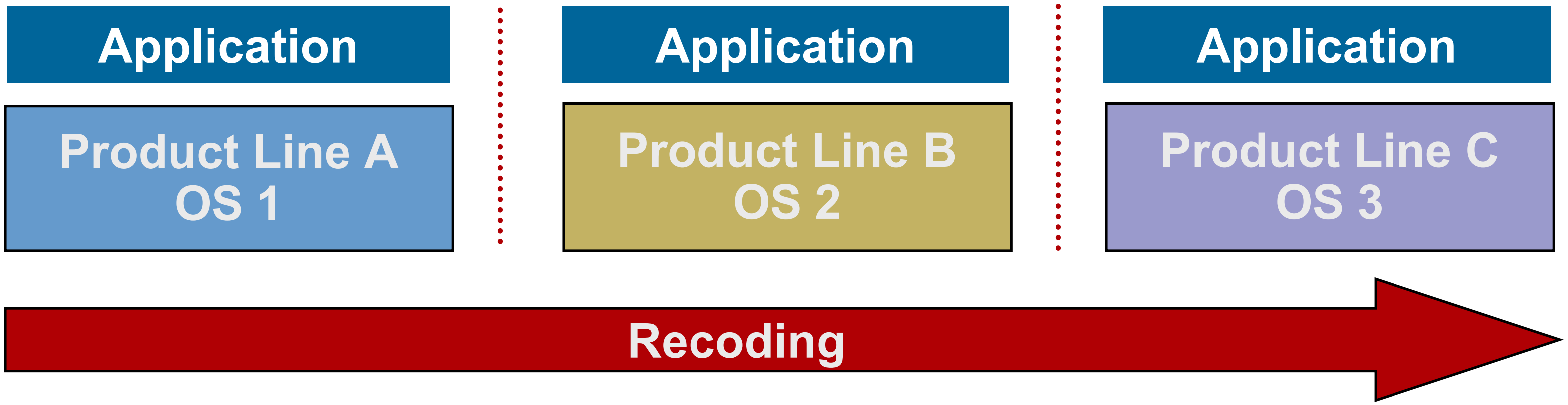
- Divergent Environments
- Portability vs. Conformance
- POSIX Overview and Evolution
- POSIX Comparison
  - QNX Neutrino, Linux & VxWorks

Application Portability

- VxWorks & QNX Neutrino
- Migration Roadmap
- Examples and Q&A

Today's Environment

- Typical company has multiple product lines and limited interoperability
- Vendors are locked into a single OS solution or
- Applications need to be recoded or ported to deploy on different product lines
  - Takes time, adds costs
  - Increases delays to product deployment



Standards & Portability

- Standard APIs preserve software investments
- Portability lowers time, cost and risk associated with integrating new technology across product lines
- Common standard maximizes application base for development environments
- Developers familiar with standard become productive more quickly

Portability vs. Conformance

- **Portability**
  - Degree to which a software/application base is reusable
    - Between different versions of the same vendor's environment
    - Between different vendors environments
  - Measurement
    - Difficult to verifiably measure
    - Portability from one environment to another is *not* a reliable metric of how portable it will be to other environments, *except* under constrained circumstances
- **Conformance**
  - Provides verifiable metric of portability on an application by application basis (pass/fail)
  - Two Sides:
    - Vendor conformance: conformant implementation
    - Consumer conformance: conforming application

The POSIX Specification

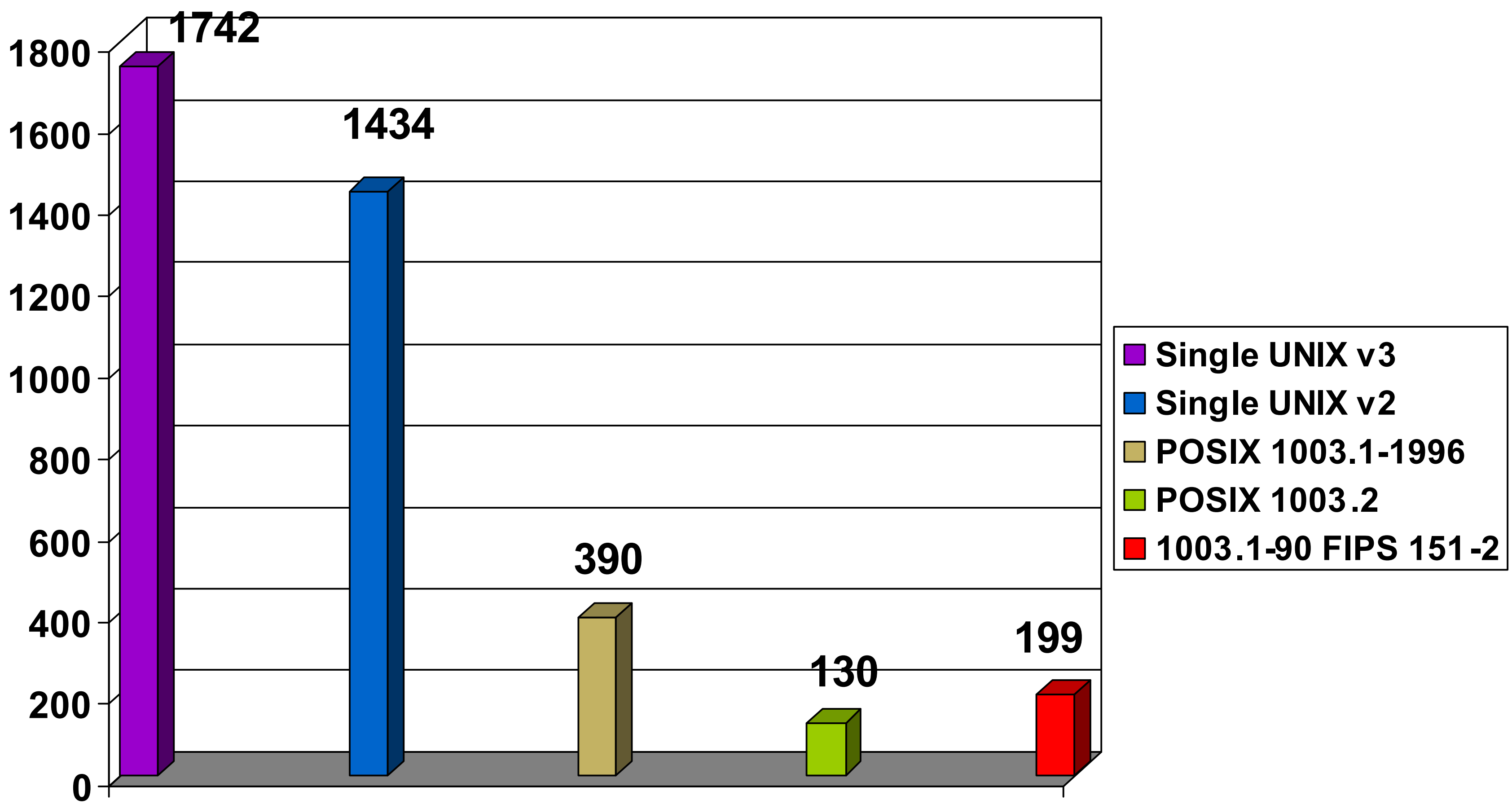
POSIX Overview

- Portable Operating System Interface
- Family of standards that define an interface, using the C programming language, a command interpreter, and common utility programs
- Developed by industry organizations
  - IEEE
  - ISO/IEC
  - The Open Group
- Introduced in 1980s to define standard way to interact with multiple UNIX derivatives
- POSIX1003.1-2001: current version of standard
  - Used by Linux Standard Base and Embedded Linux Consortium

Where Is it Used?

- POSIX can be broadly implemented across a wide range of systems, including:
  - Current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
  - Compatible systems that are not derived from the original UNIX system code
  - Emulations hosted on entirely different operating systems
  - Networked systems
  - Distributed systems
  - Systems running on a broad range of hardware

API Evolution

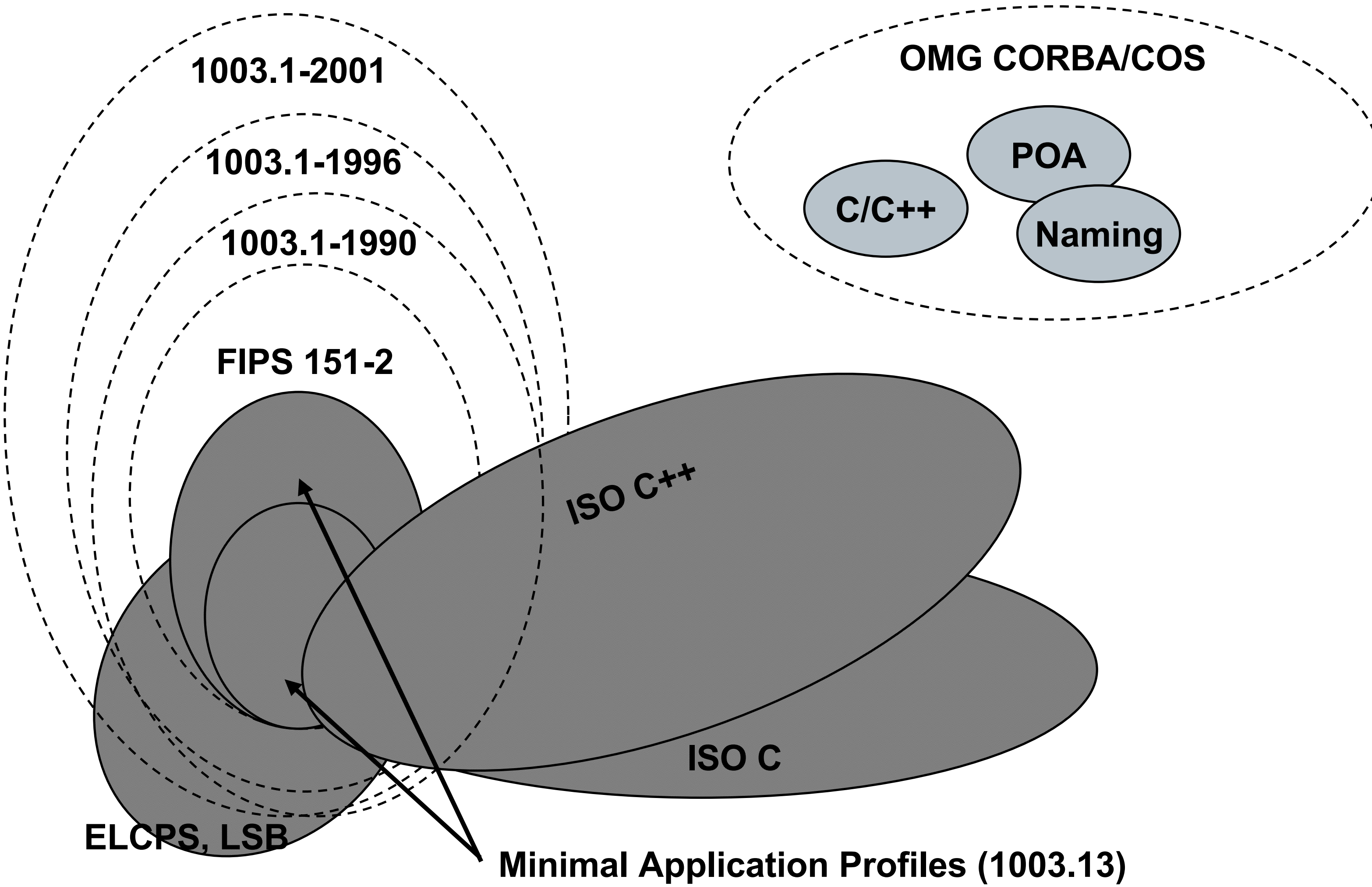


Key Benefits

QNX SOFTWARE SYSTEMS

- **Source-level compatibility of applications**
  - Can choose the best OS for the job at hand, without having to rewrite entire code base or change programming models
- **Portability of applications and programmers**
  - Lowers the time, cost and risks associated with integrating new technology across the enterprise
- **Shifts focus from incompatible system product (RTOS) implementations to compliance to single set of APIs**
- **If an OS meets the specification and commonly available applications run on it then it is open**
  - Which specification (i.e. profile) do I need?

Application Environment Profile (Landscape)



POSIX Compliance

OS Comparison

POSIX Feature Matrix

Feature	PSE 51	PSE 52	PSE 53	PSE 54	POSIX 1003.1-2001
1003.1-90 Processes	-	-	x	x	x
1003.1-90 Pipes	-	-	x	x	x
1003.1-90 Files & Directories	-	x	-	x	x
1003.1-90 Users & Groups	-	-	-	x	x
1003.1b-93 Memory Protection	-	-	x	x	x
1003.1b-93 Hi Res. Clocks & Timers	x	x	x	x	x
1003.1b-93 Realtime Signals	x	x	x	x	x
1003.1b-93 Semaphores	x	x	x	x	x
1003.1b-93 Shared Memory	x	x	x	x	x
1003.1b-93 IPC Message Passing	x	x	x	x	x

Check for Advanced Realtime

POSIX Feature Matrix

Feature	PSE 51	PSE 52	PSE 53	PSE 54	POSIX 1003.1-2001
1003.1c-95 Threads	x	x	x	x	x
1003.1c-95 Thread Safe Functions	x	x	x	x	x
1003.1c-95 Thread Attribute Stack Address	x	x	x	x	x
1003.1c-95 Thread Attribute Stack Size	x	x	x	x	x
1003.1c-95 Thread Process Shared	-	-	x	x	x
1003.1c-95 Thread Priority Scheduling	x	x	x	x	x
1003.1c-95 Thread Priority Inheritance	x	x	x	x	x
1003.1c-95 Thread Priority Protection	x	x	x	x	x
POSIX2_SW_DEV	-	-	-	x	x

POSIX Profiles – OS Compliance

POSIX Standard	QNX Neutrino	Linux	VxWorks
Specification Base	**1003.1-2001**	**1003.1-1996***	**PSE 51/PSE 52**
Realtime (.1b)	**x**	**x**	**x**
Realtime Threads (.1c)	**x**	**x**	**-**
1003.1d-1999 Additional Realtime (Sporadic server scheduling, execution timers, …)	**x**	**-**	**-**
1003.1j-200*x* Advanced Realtime (Barriers, spin-locks, …)	**x**	**-**	**-**
Best practices (development)	**Configure, GCC, perl, …**	**Configure, GCC, perl, …**	**-**

**Newer versions of the Linux kernel are moving toward conformance with the 2001 specification.*

Linux-QNX Portability

- Application portability between Linux and QNX Neutrino can be easily accomplished
  - Both Linux and QNX Neutrino share large POSIX feature set
- Linux developers can retain programming model and existing APIs while porting applications to QNX Neutrino

Bottom line:

- Porting applications between Linux and QNX Neutrino is relatively simple
- Standard POSIX APIs are key

Application Porting Example – Lynx Text Browser

Untar the source: Go to the source directory.	tar xz lynx2.8.4.tar.gz cd lynx2-8-4
Make a host directory: Go to the host directory.	mkdir x86-pc-nto-qnx cd x86-pc-nto-qnx
Configure the source: Make.	../configure make
Install the browser: Install.	make install

Sample 3rd-Party Applications

QNX SOFTWARE SYSTEMS

TAO – CORBA ORB	**PVM** – Distributed processing system	**mySQL** – Open-source database	**Postgres** – Open-source database
GDBM – Database	**Apache** – Web server	**LibXML2** – XML Database	**LibXSLT** – XSL Processor
Xerces – XML Processor	**Python** – Scripting language	**Perl** – Scripting language	**Ruby** – Scripting language
Open SSH/SSL – Secure sockets and shells	**Zebra router** – Manages TCP/IP based protocols	**Samba** – Shared access to resources on Windows networks	**Mozilla** – Web browser based on Netscape source code
Doxygen – Source code documentation tool	**Open LDAP** – Light weight Directory Access Protocol	**Sendmail** – Email server	**GNU EMACS** – Programmers' editor
GDB – GNU debugger	**GCC** – GNU C/C++ compiler	**CVS** – Source code Version-control System	**VIM** — Vi IMproved, a programmers' editor

POSIX Summary

QNX SOFTWARE SYSTEMS

- Standard interface increases software portability for all embedded systems
- Some markets, such as military, moving toward using POSIX as their base specification
- OS conformance a matter of degree
  - QNX Neutrino provides conformance with 1003.1-2001
  - Linux moving toward 2001.3-2001 with latest versions
  - VxWorks only conforms with minimal profiles – PSE 51/PSE 52
- Migration of legacy VxWorks code to POSIX RTOS increases software portability

OS Migration: VxWorks to QNX Neutrino

Jason Clarke, QNX Software Systems

OS Migration – Business Drivers

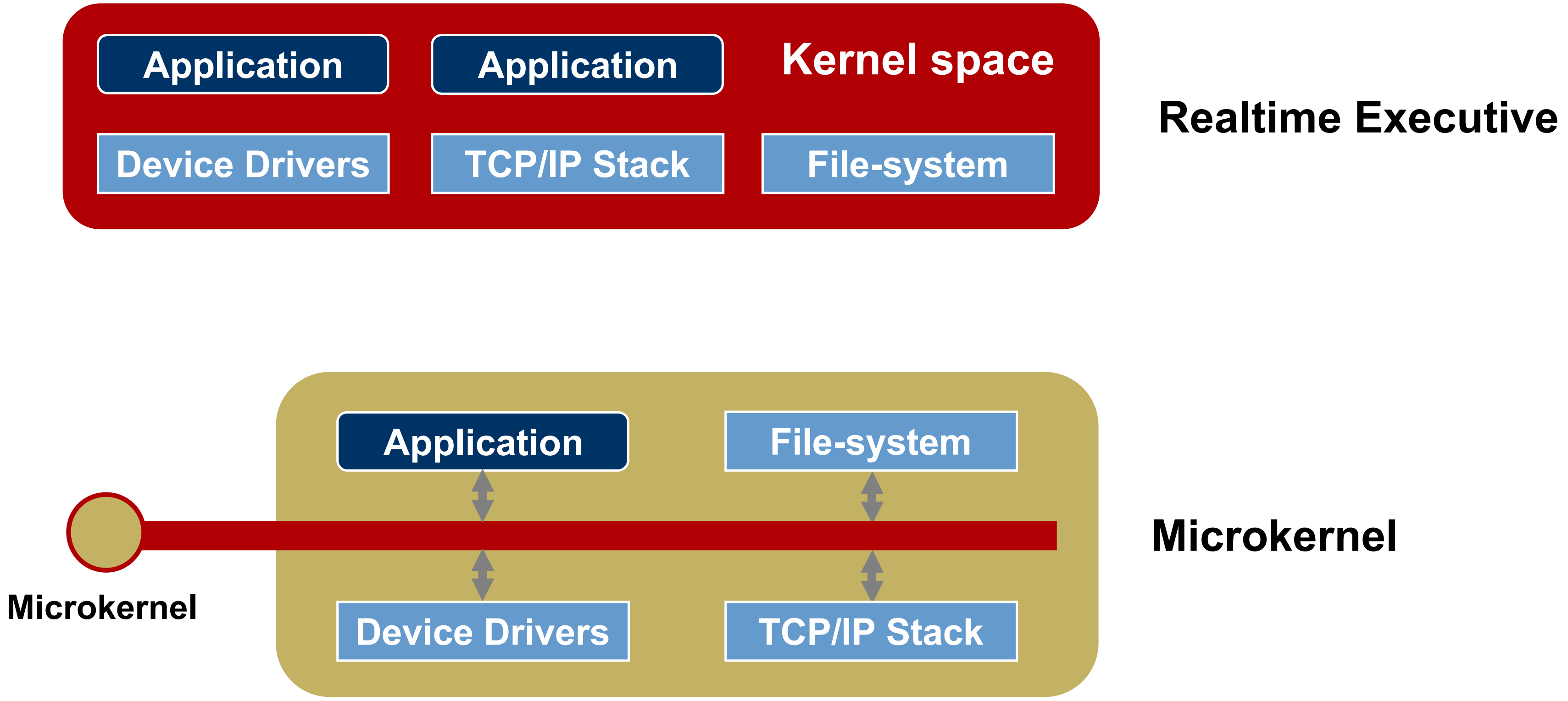
QNX SOFTWARE SYSTEMS

- **Supplier limitations**
  - Proprietary API locks customer to OS vendor
  - High cost of developer training
  - Limited software choices
- **Product capabilities**
  - Product stability – OS reliability, performance
  - Support for latest technologies – SMP, HA, 3D graphics
  - Dynamic upgradeability – modularity, software hotswap
- **Development costs**
  - High cost of new feature development and deployment
  - Soaring bug identification and bug fix costs
  - Third-party software porting and integration costs
  - Need to employ specialized kernel experts

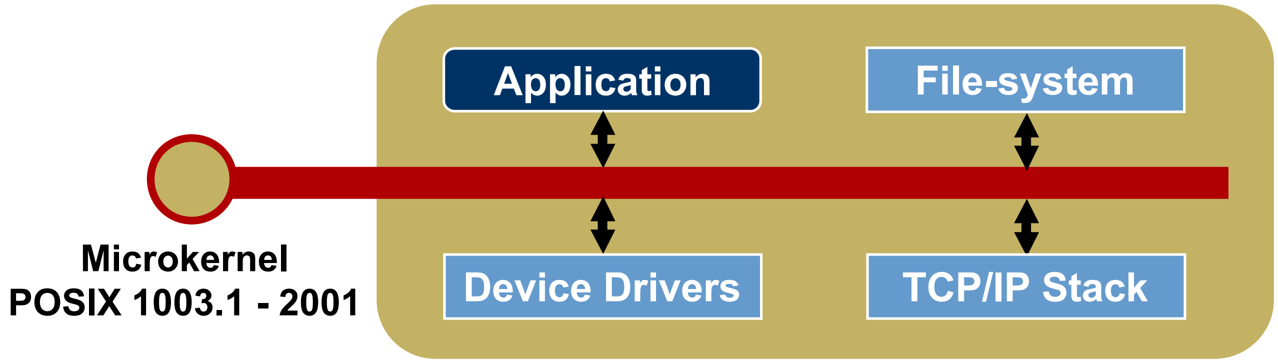
Porting Issues

- **Software architecture**
  - Memory accessibility (process vs. single-address model)
  - Tight coupling between OS, system, and user tasks
- **Differences at system and application level API**
  - POSIX vs. minimal profile + proprietary
  - Physical memory vs. virtual memory addressing

Architectural Comparison

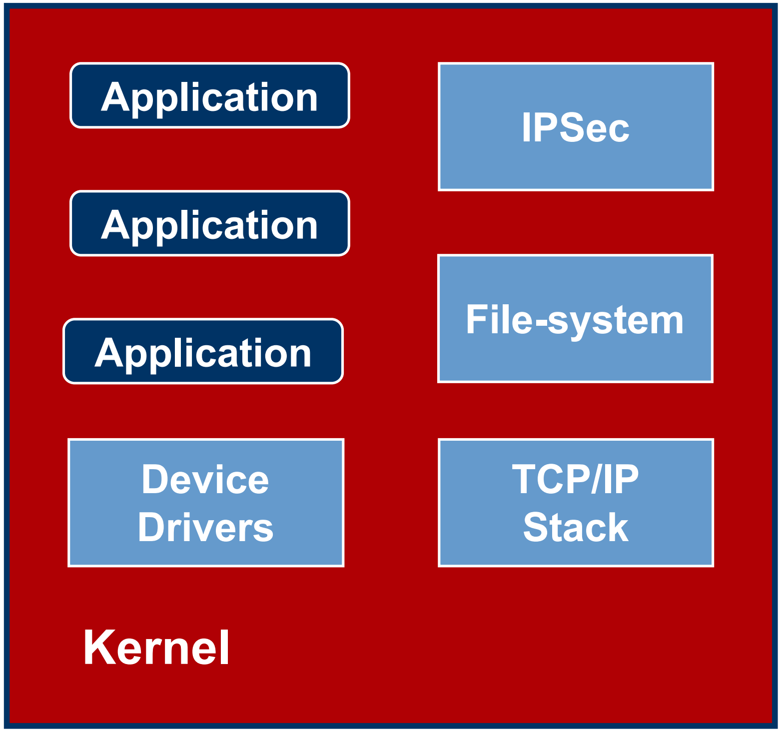


Architecture – Microkernel



- POSIX is bred in the bone
- Applications, drivers, stacks coded to the same APIs

Legacy VxWorks Environment



The Migration Goal:

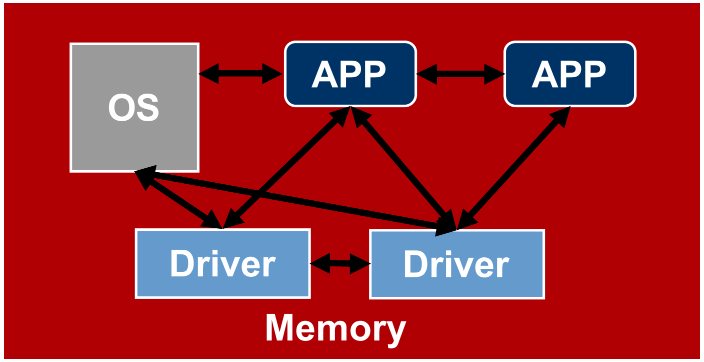
- Re-use existing legacy software
- Future-proof, unified, scalable software architecture

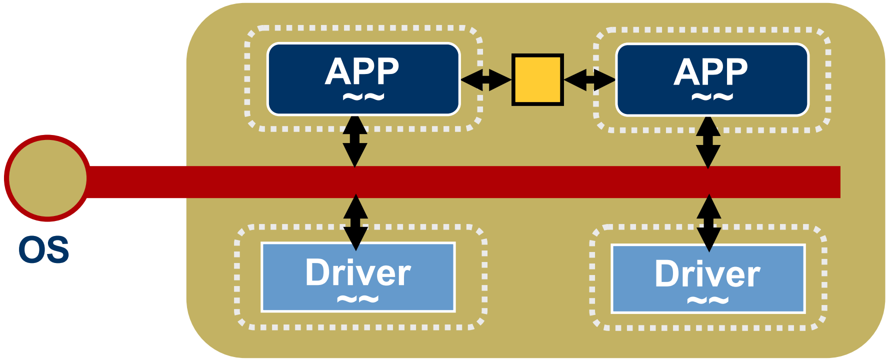
The Migration Challenge:

- Poorly defined coupling between components
- Implicit sharing of memory
- Several assumptions about legacy architecture creep into code

Memory Accessibility

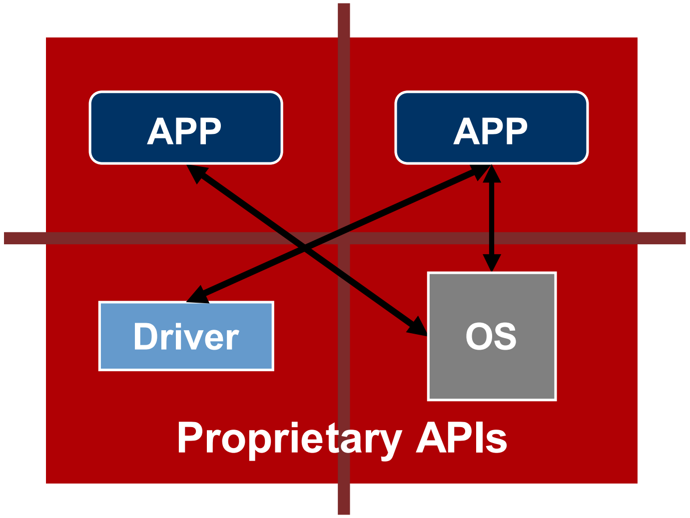
- **Realtime executive model: All tasks have access to complete system memory**
  - Tight coupling between OS, system and user tasks
  - Difficult to separate components without re-design
  - Re-architecting parts of the system is non-trivial
- **Process model: Threads within a process have access to the same memory**
  - Separate processes can share memory using explicitly defined shared memory regions

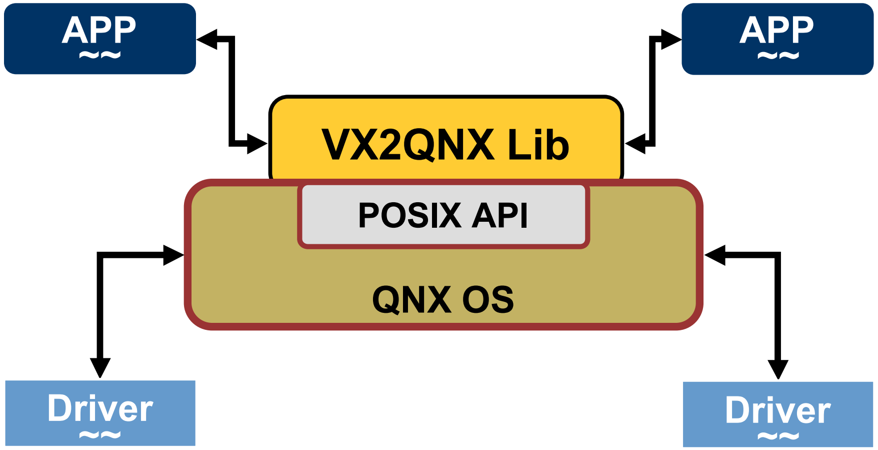




APIs

- Differences at system and application level API
  - Legacy applications often use proprietary APIs
  - Interface to system is also proprietary (possibly different)
  - While porting, a mapping may be necessary between the proprietary API and the underlying POSIX API of the new OS
    - Can encapsulate most of mapping in a "porting library"

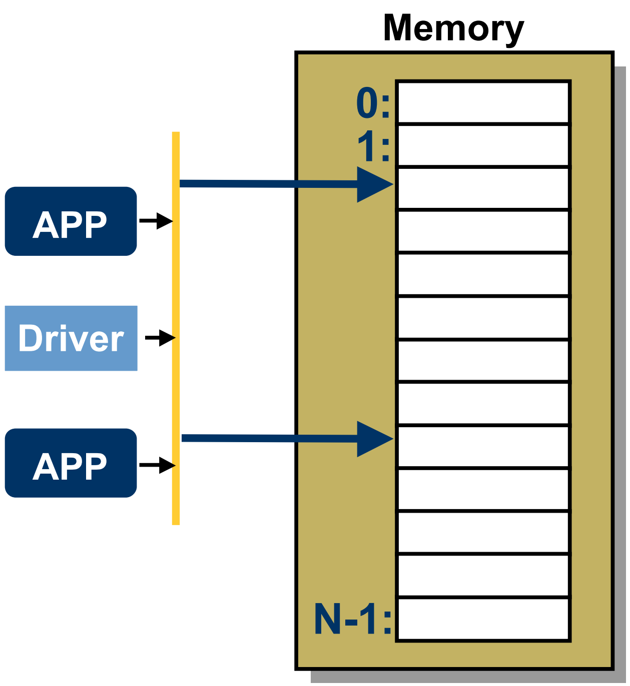




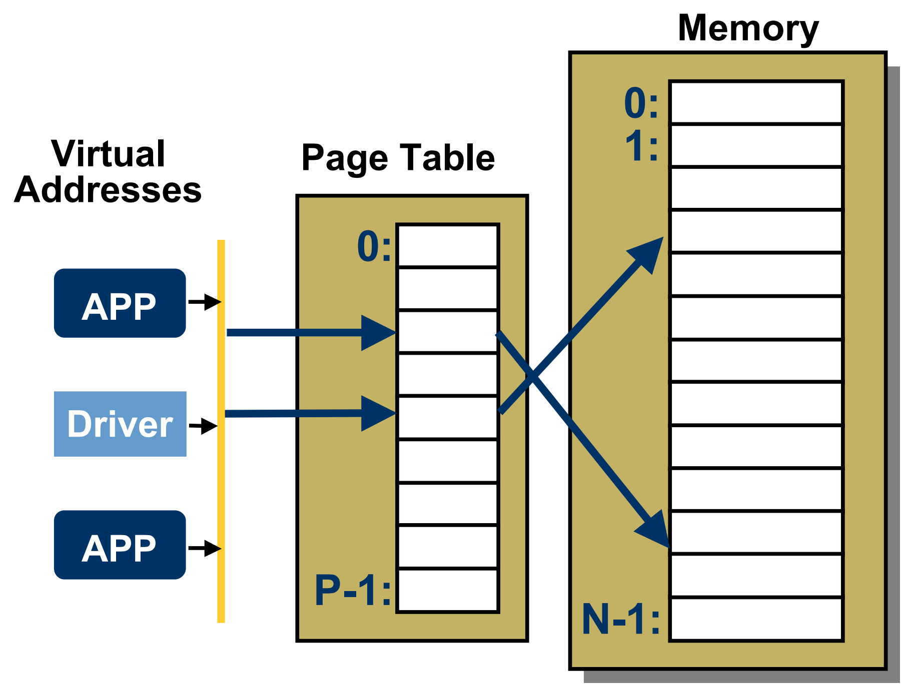
Physical vs. Virtual Memory

QNX SOFTWARE SYSTEMS

Physical Addresses



Physical Addresses



- OS uses hardware to translate virtual addresses to physical addresses using a maintained table
- Must ensure all memory accesses performed via “mapped-in” variables instead of #define

Software Builds

- **Build scripts/makefiles must be re-worked**
  - Adapt/adopt build infrastructure
    - Re-work macros to point to QNX tools + re-write link sections
    - Import code base using the QNX Momentics IDE (automatically sets up make infrastructure)
- **Compiler differences**
  - Different compiler vendors (Diab, Metrowerks, GCC, etc.)
  - Different GCC variants: 2.7.2 (VxWorks 5.4) vs. 2.95 (QNX / VxWorks 5.5)
  - Code changes required to remove compiler errors
- **Linker**
  - Different linkers/linker options (change makefiles / macros)
  - “main” function for each separate process: Equivalent to VxWorks “usrApplInit() “ function
  - Shared library concept used to reduce memory footprint

Other Considerations

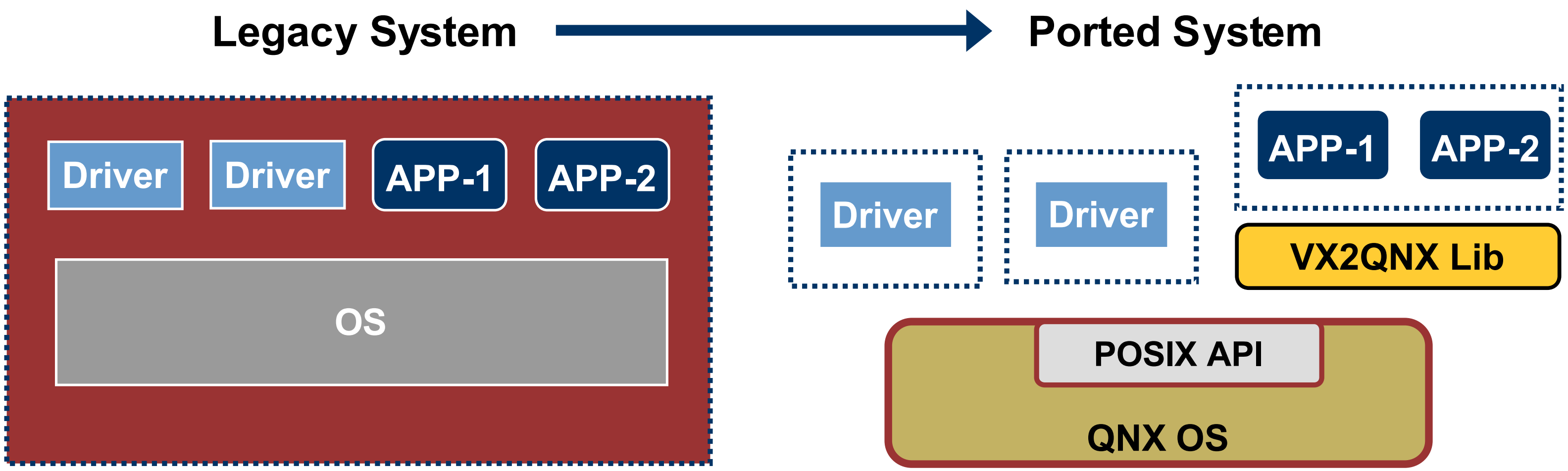
- Significant legacy software base
  - Millions to tens of millions of LOC
  - Large number of protocols and applications to be re-used
- Most software written for real-time executives
  - Specific assumptions about underlying RTOS
- Code may depend on specific tools
  - C++ especially fragile
- Modularity not always enforced
  - May complicate “from the ground up” re-architecting
- Device driver infrastructure
- Software build infrastructure, capabilities

Porting Strategies

A phased approach

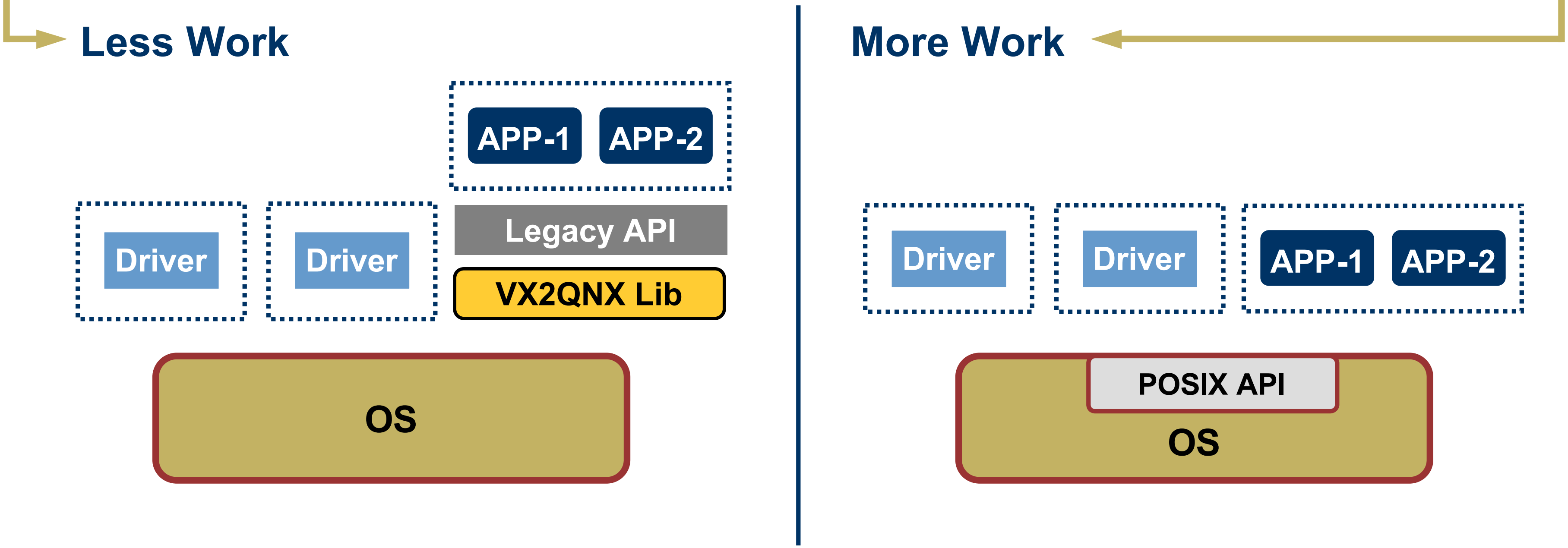
Porting Process

- Two main aspects: porting driver/hardware related code, and porting application code
  - Typically porting driver code will be done manually by inspection ("do-once" operation)
  - Porting application code would likely be most significant portion of effort associated with porting



Porting Strategies

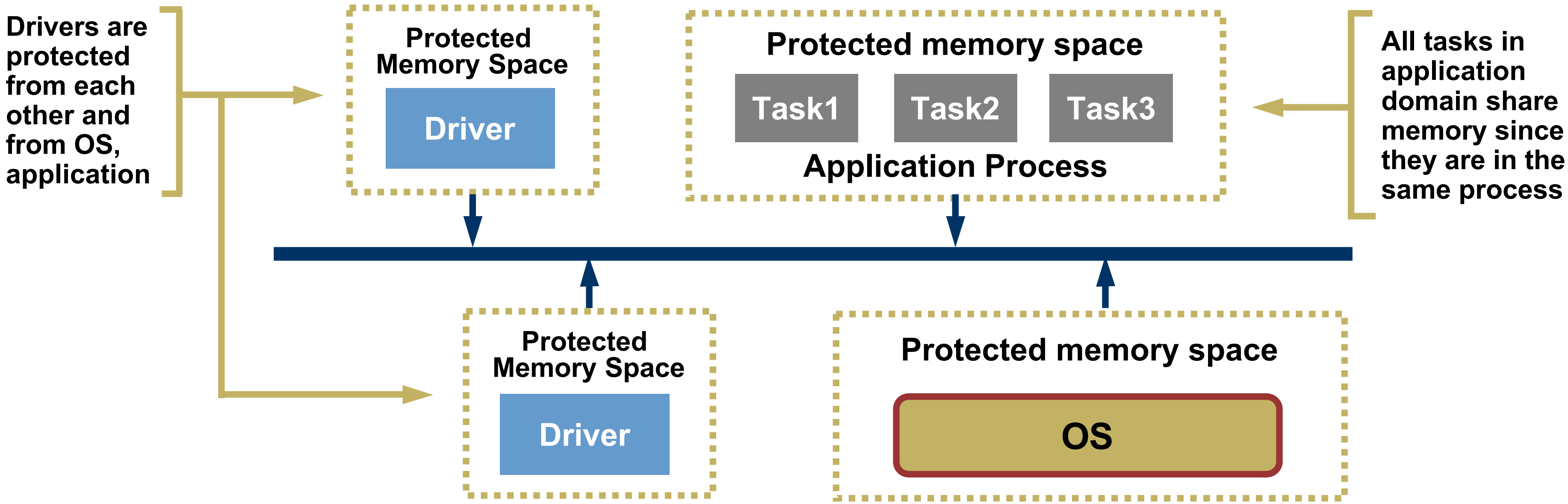
- Two main strategies to deal with legacy application code
  - Develop porting library that provides legacy API while implementing it using underlying API calls of new OS
  - Replace legacy functions with appropriate native OS calls for the new OS. Can be done manually or automatically through use of code parsing tools



Execution Model

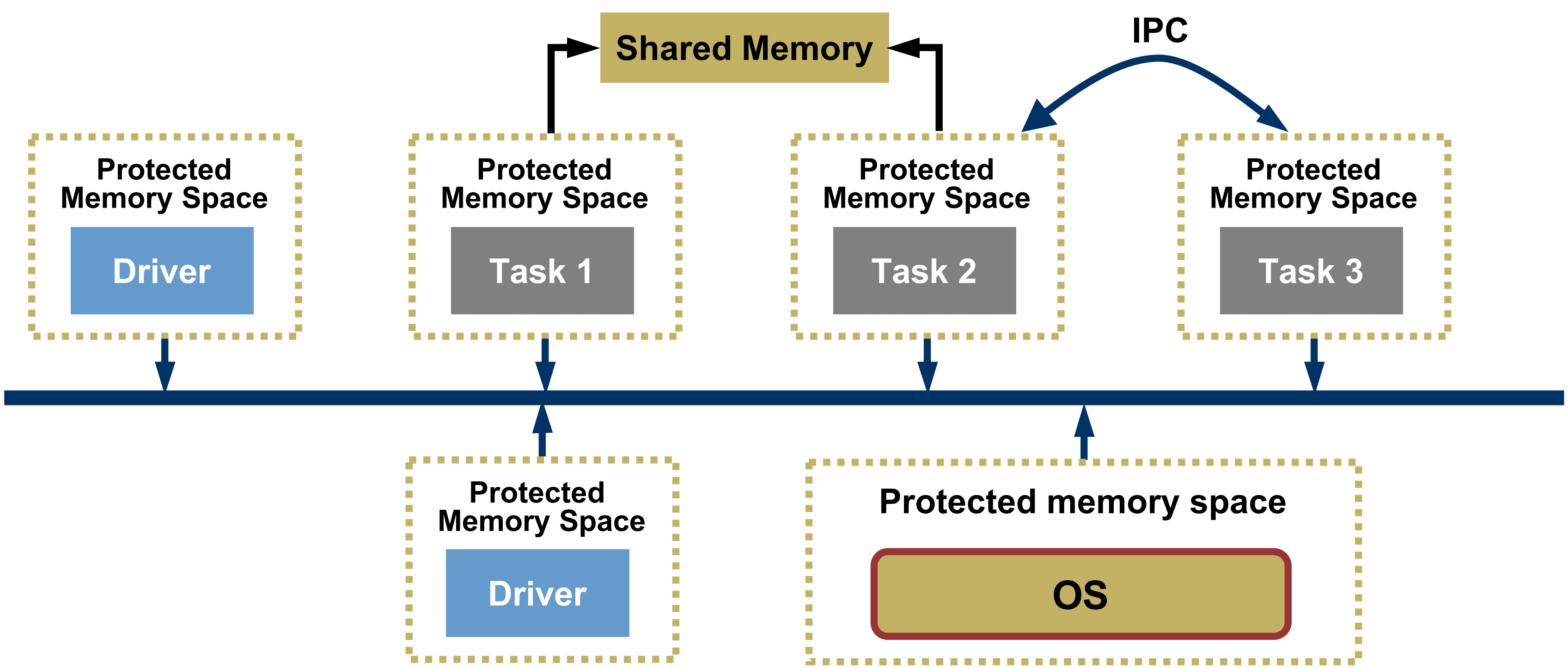
➔ Mapping tasks – Option #1

> Run application as single process under the new OS. Every task in the original legacy application becomes a thread in the new application process. Drivers run in their own protected memory spaces. Application is protected from driver and OS

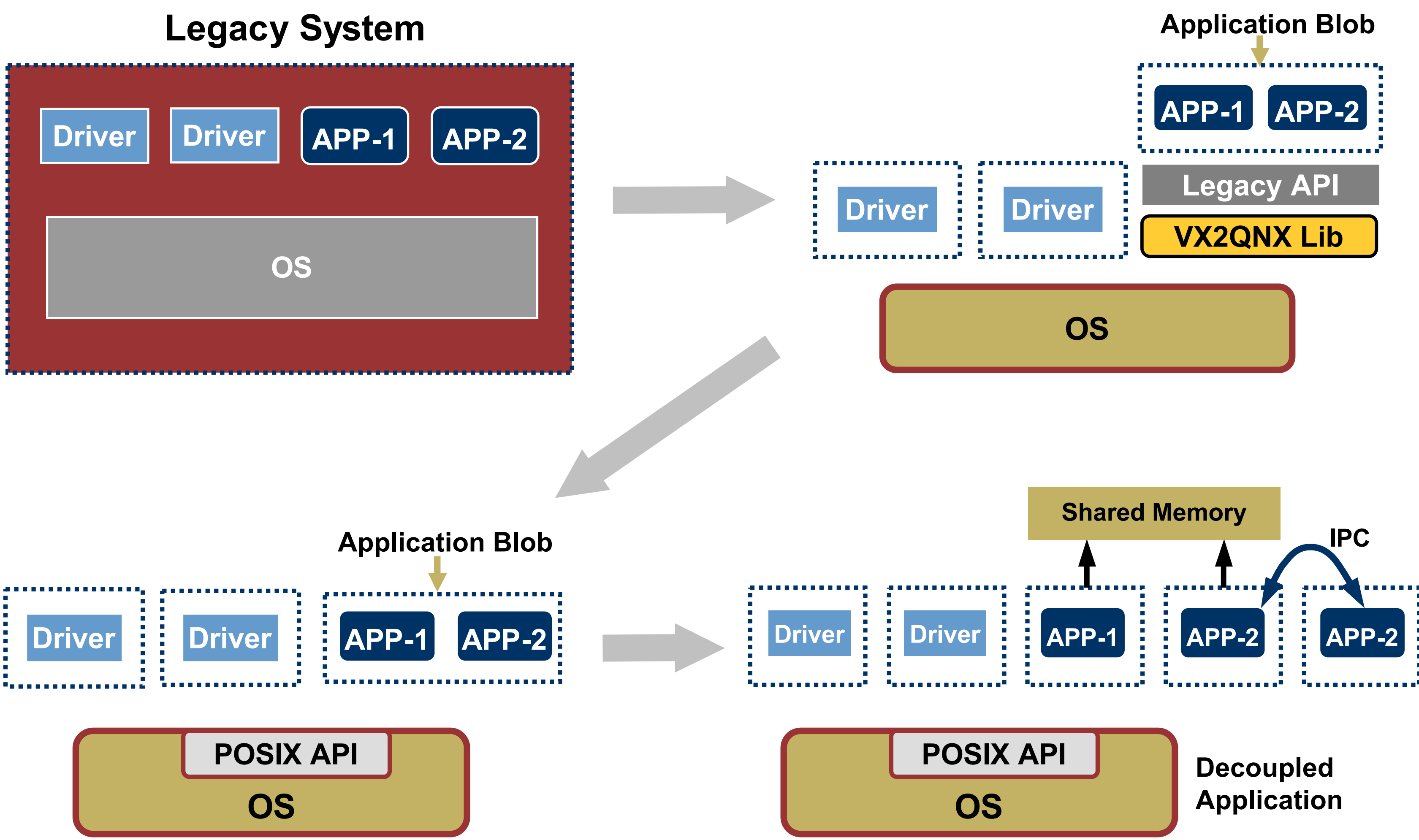


Execution Model Cont.

- Option #2: Break application down into separate processes that communicate using process IPC mechanisms and shared memory to share data (far more robust)



Porting Roadmap



Porting Library

- Implements key VxWorks functions
  - Functionally equivalent implementation for the VxWorks API calls
  - Provides code compatibility with legacy code at the application layer
- Complete VxWorks system is encapsulated inside one process under the QNX® Neutrino® RTOS
  - Task in VxWorks ® Thread in QNX Neutrino
- taskLib, msgQLib, semLib, semCLib, semBLib, semMLib, wdLib, errnoLib, taskInfoLib, kernelLib, lstLib, schedPxLib, mqPxLib, clockLib, semPxLib, sigLib, timerLib, …
- Covers majority of core VxWorks API
- Networking API coverage also being added into library
- Library provided as source:
  - Use as reference for porting and/or deployment as a compatibility layer

Migration Scope

- Adapt/adopt build infrastructure
  - Examine on per-environment basis
- Port hardware-related code
  - Drivers and OS
- Port application code
  - Porting library solves a major issue (API compatibility)
- Phased effort lets you evolve system over time
  - No system downtime
  - More manageable

Porting Examples

Inter-task Synchronization

VxWorks Using Semaphores

```
SEM_ID mSem;

void func1(void) {
...
sem_take(mSem,...);
...critical section
sem_give(mSem);
...
}

void func2(void){
...
sem_take(mSem,...);
...critical section
sem_give(mSem);
...
}

void init(void) {
...
mSem = semMCreate(...)
...
taskSpawn("Task1", ..., func1, ...);
taskSpawn("Task2", ..., func2, ...);
...
}
```

QNX Neutrino Using Mutexes

```
pthread_mutex_t *mmtx;

void func1(void) {
...
pthread_mutex_lock(mmtx,...);
...critical section
pthread_mutex_unlock(mmtx);
...
}

void func2(void){
...
pthread_mutex_lock(mmtx, ...);
...critical section
pthread_mutex_unlock(mmtx);
...
}

void init(void) {
...
mmtx = malloc(sizeof(pthread_mutex_t));
pthread_mutex_init(mmtx, ...);
...
pthread_create(..., func1, ...);
pthread_create(..., func2, ...);
...
}
```

QNX Neutrino Using Semaphores

```
sem_t *msem;

void func1(void) {
...
sem_wait(msem,...);
...critical section
sem_post(msem);
...
}

void func2(void){
...
sem_wait(msem, ...);
...critical section
sem_post(msem);
...
}

void init(void) {
...
msem = malloc(sizeof(sem_t));
sem_init(msem, ...);
...
pthread_create(..., func1, ...);
pthread_create(..., func2, ...);
...
}
```

Memory Accessibility

VxWorks

```
#define DEVICEADDR 0x8000abcd
 void *daddr;

void func1(void) {
int val;
…
// read value
val = *daddr;
// modify value
...
// write value
*daddr = val;
}

void init(void) {
…
daddr = DEVICEADDR;
taskSpawn("Task1", ..., func1, ...);
...
}
```

QNX Neutrino

```
#define DEVICEADDR 0x8000abcd
 void *daddr;

void func1(void) {
int val;
…
// read value
val = *daddr;
// modify value
...
// write value
*daddr = val;
}

void init(void) {
...
daddr = mmap (… len, ..., DEVICEADDR );
pthread_create( ..., func1, ...);
...
}
```

Migration Summary

- **Business and technical needs drive migration**
  - **Reliability, modularity, cost to add new features, software reuse…**
- **Moving to POSIX RTOS provides greater application portability, increases software ROI**
- **Phased approach to migration enables continued revenue stream and manageable migration path**
- **QNX offers program designed to accelerate migration**
  - **Porting tutorial and library**
  - **Extended QNX Momentics evaluation**
  - **Migration support and services packages**

Questions?

Q&A Session

Thanks for Your Time!

Steve Furr (furr@qnx.com)
Jason Clarke (jclarke@qnx.com)

Porting Tutorial: “Migrating Legacy Code from WindRiver’s VxWorks to the QNX Neutrino RTOS”
http://www.qnx.com/mailings/vxporting/