Linux Pundit

Open Source Opinion & Analysis

Business & Technical Consulting

Migrating to Linux for Device Software

Part I – Why Make the Move?

Bill Weinberg

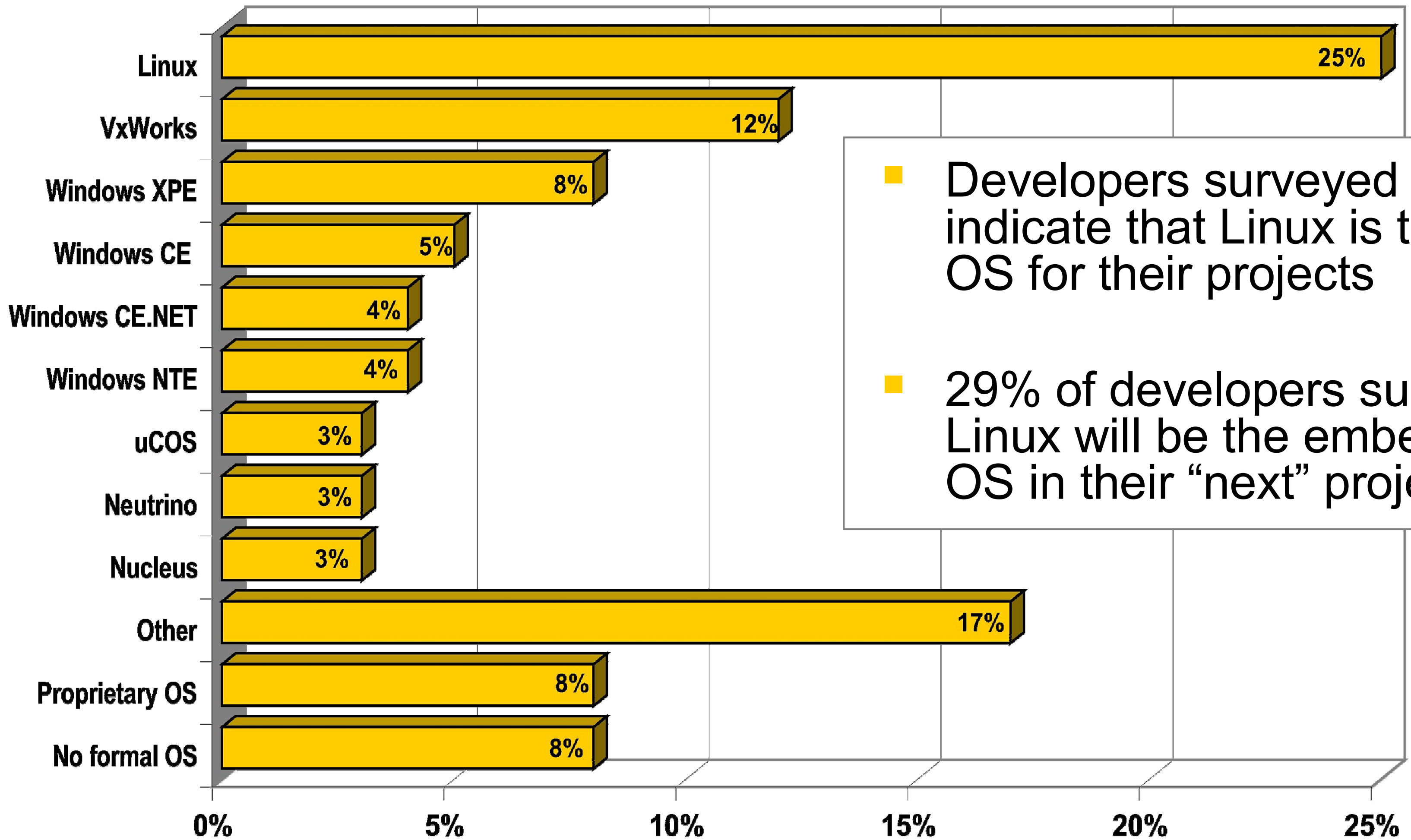
Wind River Seminars – Sunnyvale, Alameda, Seattle

August 2005

Agenda – Part I
Migrating to Linux for Device Software

- Device Software Landscape
- Migration Rationale
  - Business Drivers
  - Technical Benefits
- Getting Started

Target OSes for 32/64-bit Applications

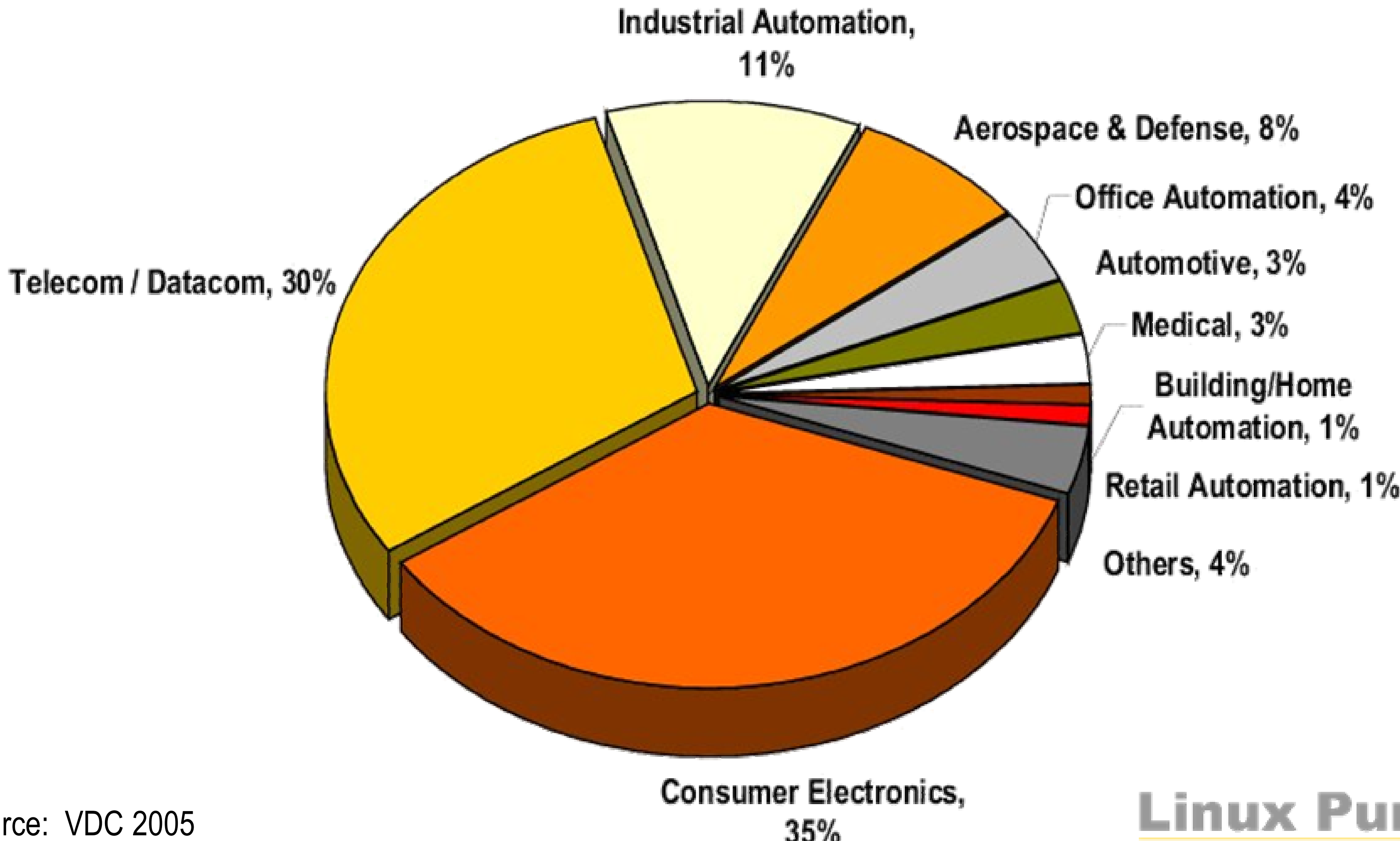


- Developers surveyed by VDC indicate that Linux is the top OS for their projects
- 29% of developers survey that Linux will be the embedded OS in their "next" project

Linux Application Space for Device Software

- Communications Infrastructure
  - Switches, routers, base stations, access points, hot spots, media gateways, SLAM, firewalls . . .
  - Wireless and wire-line
- Consumer Electronics
  - Handheld devices – cell phones, PDAs, media players, cameras
  - Automotive – in-car entertainment, navigation, satellite radio
  - Home entertainment – TV, HDTV, DVR/PVR, home gateways, home control
- Instrumentation and Control
  - Medical devices, industrial monitoring, manufacturing control, test equipment
- Aerospace and Defense
  - Secure networking, command and control, launch systems, simulation
- Office and Retail Automation
  - Printers, faxes., scanners, MFDs, voice mail, voice conferencing, POS, transaction terminals, thin clients
- Almost every other type of embedded design!

Linux for Device Software - Segmentation



Source: VDC 2005

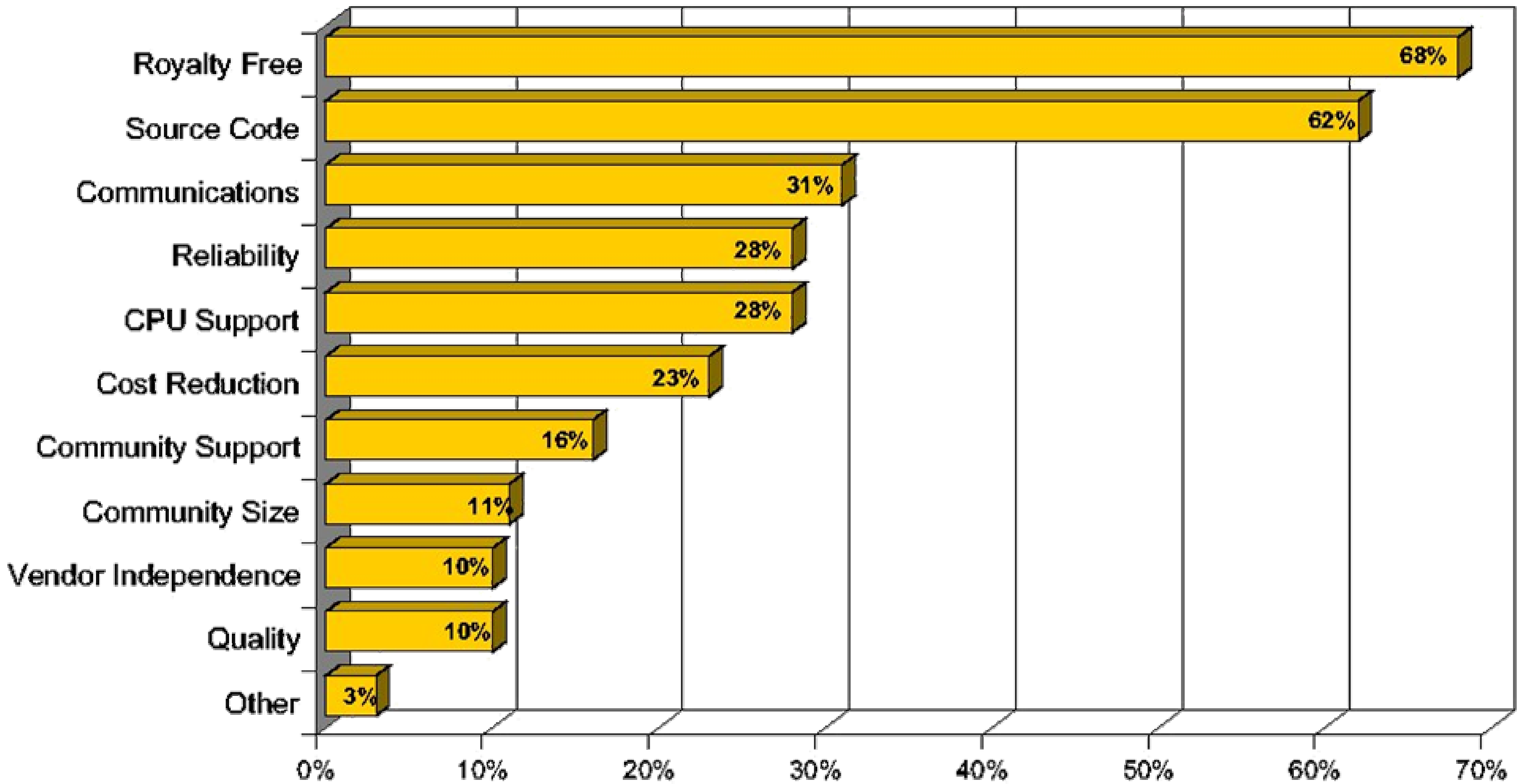
Where Linux Isn't in Embedded (Today)

- Applications Needing Certification
  - Aerospace, medical
- Software-based Real-time
  - Traditional industrial control
  - Network data plane
  - Software intensive signal processing
- Size- and Resource-sensitive Applications
  - 8, 16, and 32-bit SoCs and and MMU-less applications
  - Small memory footprint (<256KB)

Migration Motivators – Many and Varied

- Lower Total Costs
- Higher Reliability
- Vendor Independence / Self-determination
- "Gold Standard" Networking
- Broad CPU Support
- Tools Availability
- Enterprise Features make Embedded Mainstream

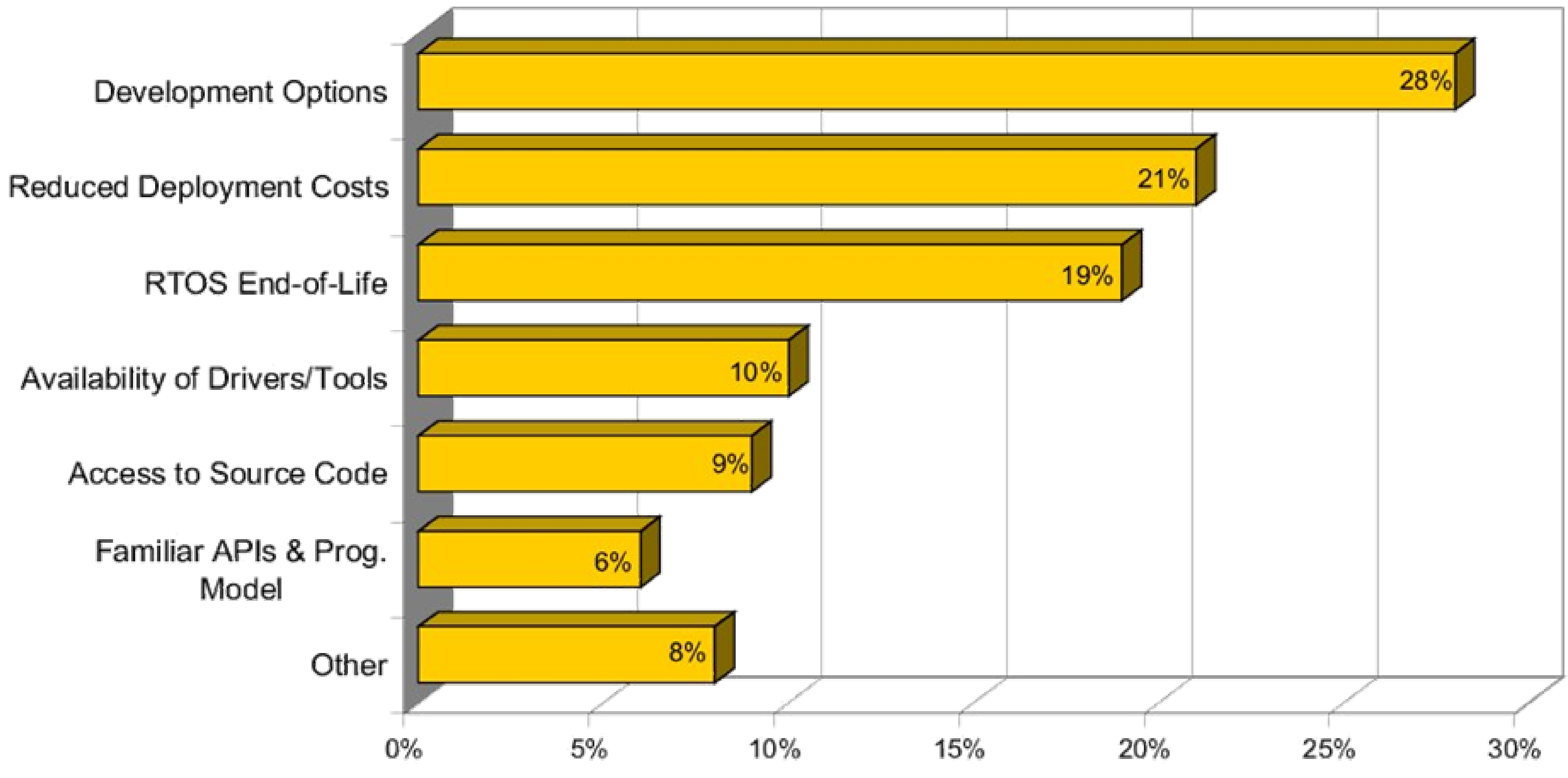
Linux for Device S/W - Adoption Factors



Source: VDC 2005

Linux Pundit
Open Source Opinion & Analysis
Business & Technical Consulting

RTOS Developers - Reasons for Migration



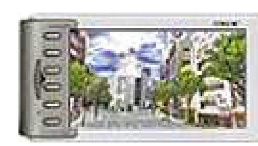
Source: LinuxPundit On-line Poll

Intelligent Device OEMs: Why They Look to Linux

- Platform consolidation
  – Strategic hardware and software platform
- Reduced bill of material costs
- Native platform for value-added services
- Synergy with deployment infrastructure







Platform Consolidation

- Historically, device OEMs support diverse, multi-tier product platforms
  - Entry-level, superior and deluxe product versions
  - Products developed by different subsidiaries or acquisitions
  - Legacy, current and next-generation development efforts
- Heterogeneous h/w and s/w raise costs
  - Multiple suppliers at lower volume/price points
  - Need to maintain separate teams for each platform type
    - Higher training, development and maintenance hosts

Consolidating Product Tiers and Technologies

High End
- Open High Level OS
- Open Environment
- Multimedia Enhanced
- High Performance Java

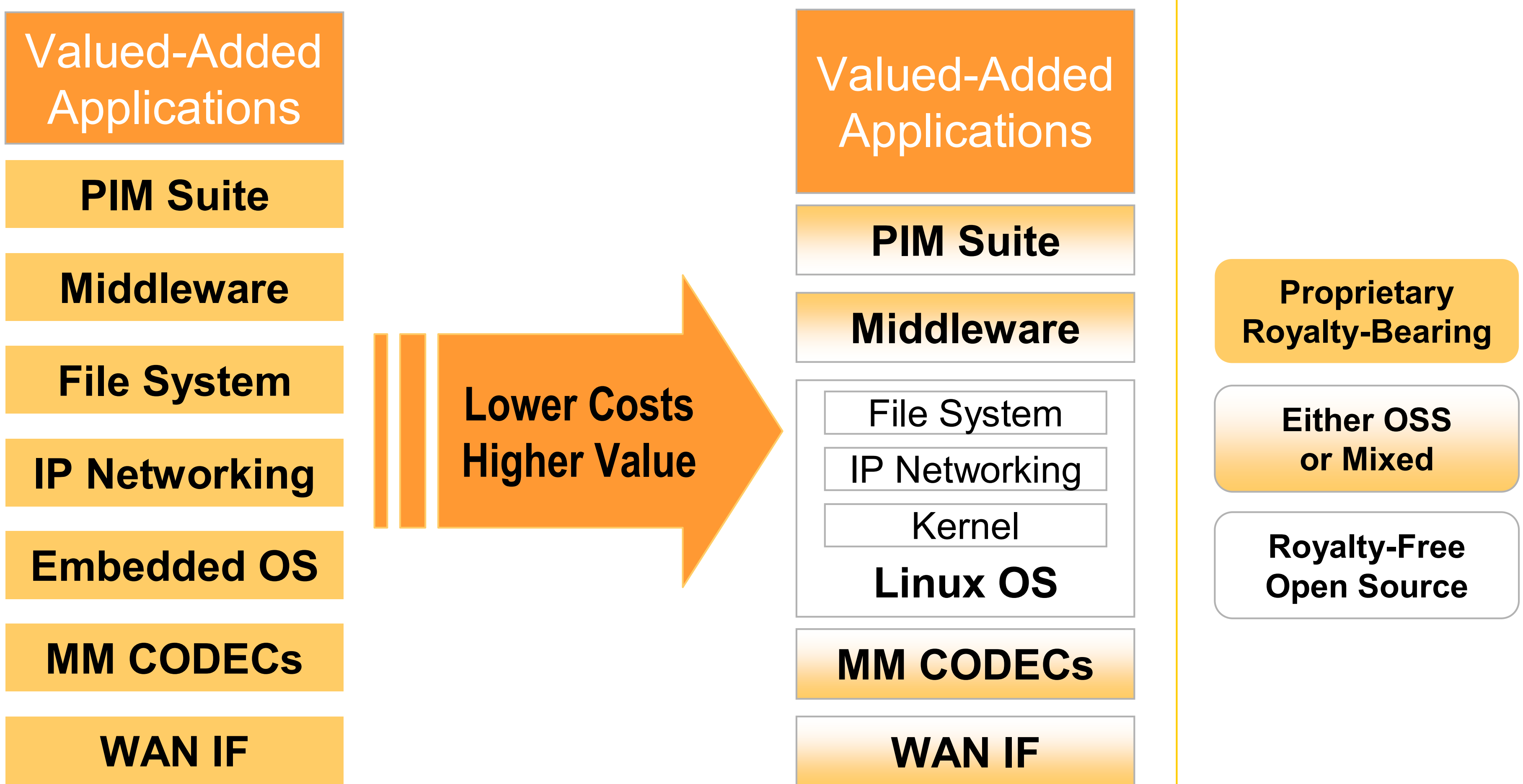
Middle Tier
- High-level OS
- Open Applications Environment
- Simple Multimedia
- Java Enabled

Low End
- Proprietary OS
- Closed application
- Basic Functionality

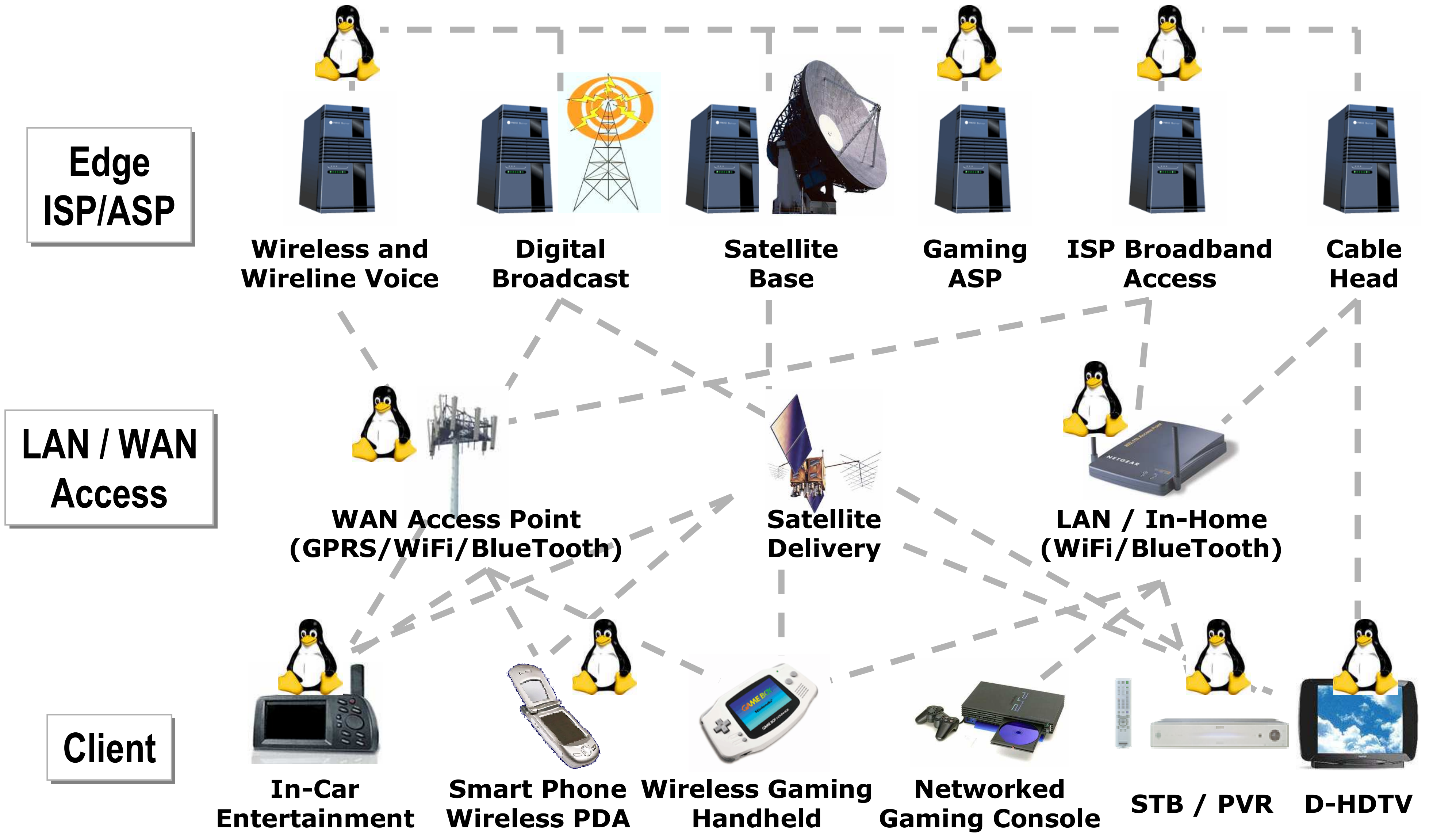
Unified Platform
- Linux at all tiers
- H/W and S/W MM Implementations
- More features enabled at higher product tiers

Linux Pundit
Open Source Opinion & Analysis
Business & Technical Consulting

Optimizing the Bill of Materials



Service/Content Delivery Infrastructure



Technical Benefits

- Networking Performance
- CPU Support
- Availability of Device Drivers
- Multi-Core and Multi-Processor Support
- Security
- Robustness

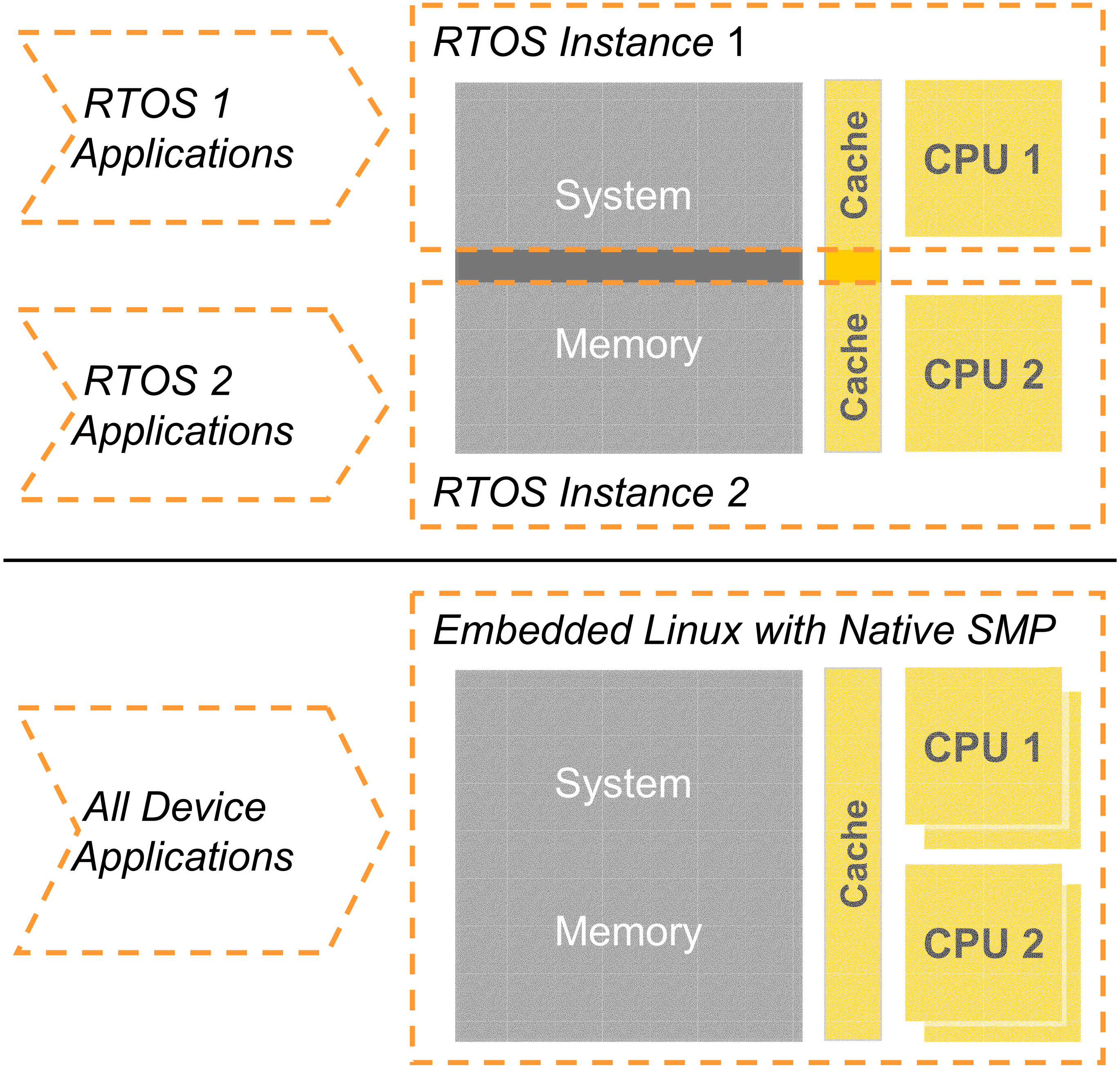
Linux CPU Support

- Not Just "Enterprise" CPUs
  - IA-32/x86 and Power Architecture
- Full Range of Embedded Processors
  - MPUs, MCUs, SoCs, FPGA
  - ARM, MIPS, M68K, PowerPC, SH, SPARC, Xtensa
- Integrated, Mature 64-bit support
  - Data and Instruction Sets

Multi-Processor and Multi-Core Support

- Next-generation CPUs deploying with multi-core architectures
  - Intel, AMD, FreeScale, and also ARM
- Requirements from device software?
  - Path to higher MIPS/watt on low-clock CPUs
  - Ability to leverage enterprise technology
  - Run increasing software workload

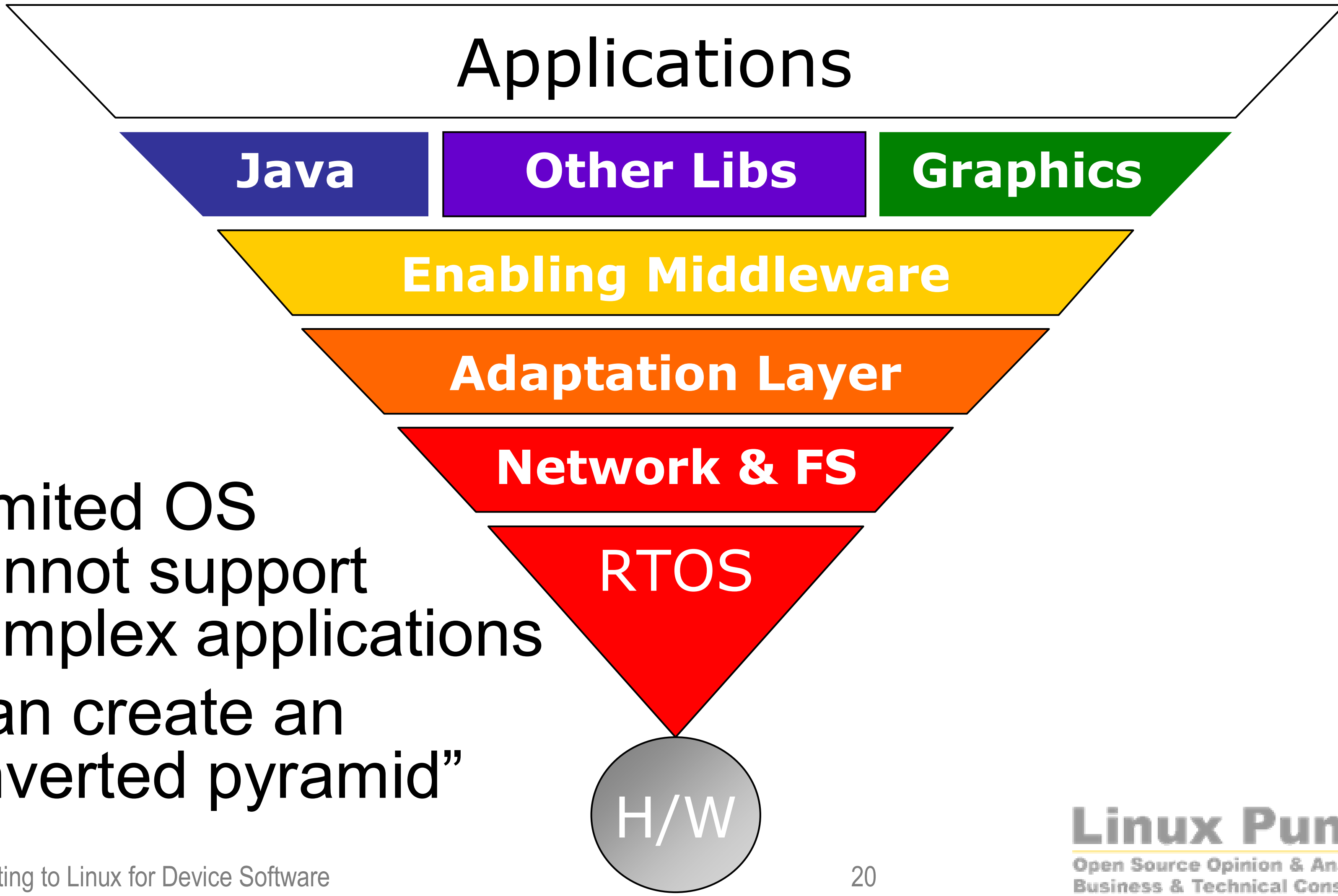
RTOS vs. Linux for Multi-Core and Multi-CPU Designs



Starting Point – Traditional RTOS

- Traditional “executive” RTOSes
  - VxWorks 5.0, pSOS, VRTX, eCOS, Nucleus
- Not Proprietary Embedded UNIX
  - ChorusOS, LynxOS, QNX
- Not enhanced RTOS
  - VxWorks 6.0/AE, Integrity, OSE

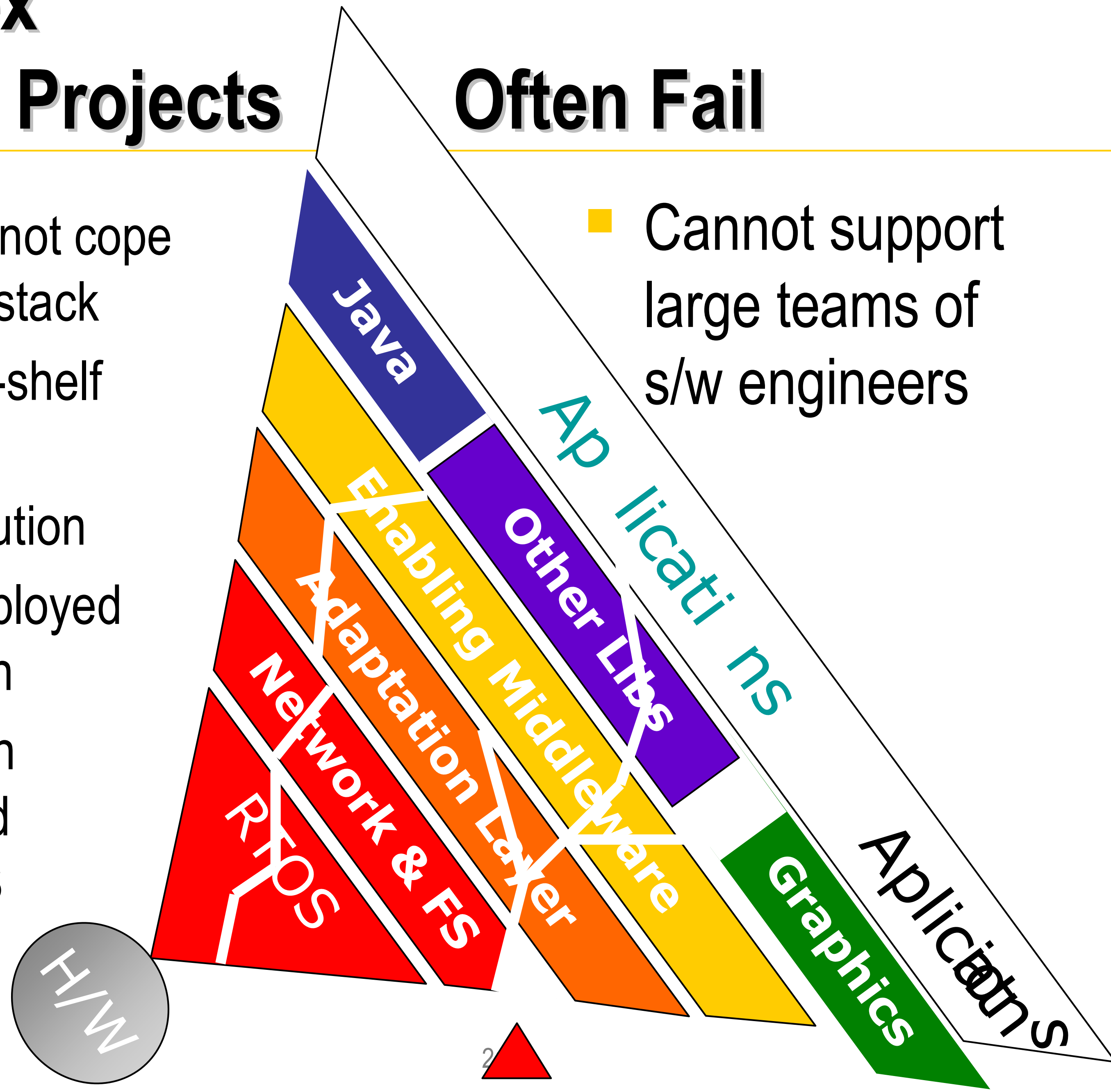
Building Complex Applications with a Vintage RTOS



- Limited OS cannot support complex applications
- Can create an "inverted pyramid"

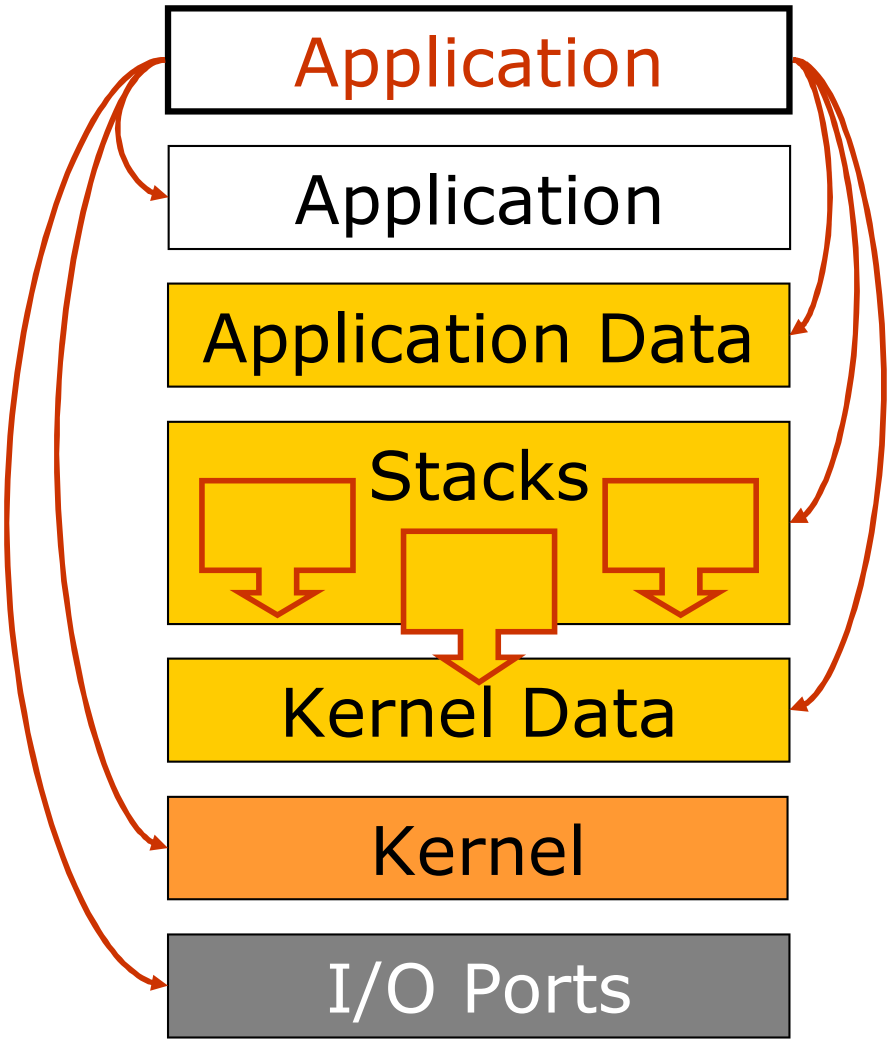
Why Complex RTOS-based Projects Often Fail

- Simple kernel cannot cope with complex s/w stack
- Little or no off-the-shelf s/w and m/w
- Single-vendor solution
- No support for deployed memory protection
- Mismatch between modern CPUs and 10-year old RTOS



- Cannot support large teams of s/w engineers

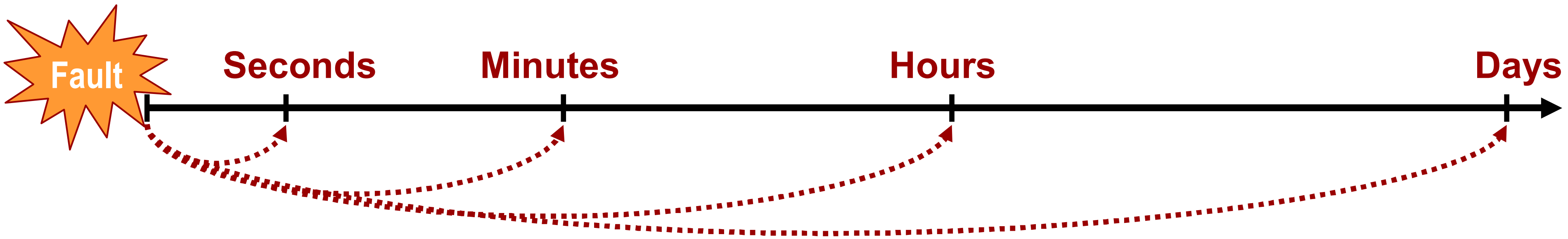
RTOS Applications Highly Exposed to Corruption



- Applications can corrupt
  – User and kernel data
  – Other applications
  – Kernel program code
  – I/O Device Ports
- Stacks freely underflow
  – Overwrite each other
  – Corrupt other data

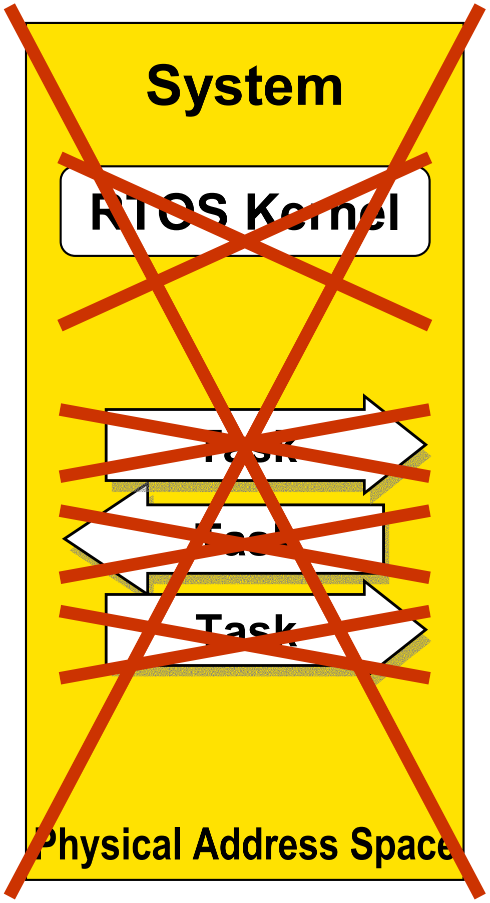
Fault Detection with a Vintage RTOS

- Effects of corrupt code, data, I/O Unpredictable
  - Minor glitch . . . catastrophic failure ??
- Fault-to-Detection time unbounded
  - Microseconds, seconds, minutes, days, months . . . years
- When detected, no association with cause!



Linux Pundit
Open Source Opinion & Analysis
Business & Technical Consulting

Fault Scope with a Vintage RTOS



- Failure almost always unrecoverable
  - Data and code overwritten
  - Stacks corrupted
  - Dynamic data leaks
  - Tasks not reliably restarted
- Failure is entire system!
  - Uses watchdogs to ensure integrity
  - Only recourse -- reboot system!

Fault Detection/Prevention with Linux

- Linux Catches Common Programming Errors
  - Writes to application or kernel code
  - Many accesses off “stray” pointers
  - Attempted writes to kernel data
  - Stack under-runs
  - Illegal accesses to I/O devices
- Immediate Process Termination : SEGFAULT
- Can Produce Core File
  - Can be parsed with debugger to determine and repair exact cause of error
- Failed Process Can Also Be Restarted

Getting Started

- Hardware Bring-Up
- Firmware Bring-Up
- Linux on Reference Platform
- Stage I Integration

Demo

- Application Migration

[redacted]

Linux Pundit
Open Source Opinion & Analysis
Business & Technical Consulting

Migrating to Linux for Device Software
Part II – Choosing the Path to Linux

Bill Weinberg
Wind River Seminars – Sunnyvale, Alameda, Seattle
August 2005

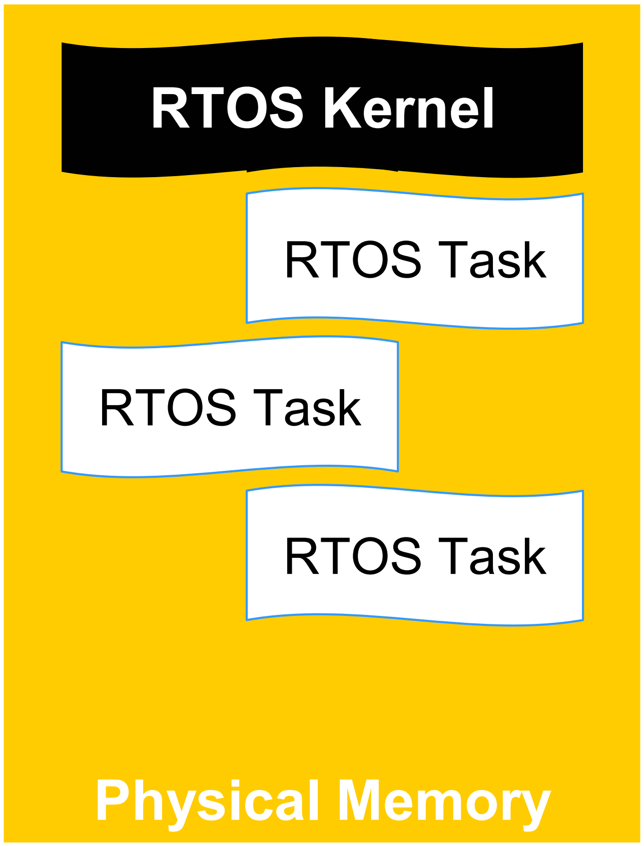
Agenda – Part II

Migrating to Linux for Device Software

- Comparing Legacy and Linux Architectures
- Run-time Architecture Options
- Migration Process
- Key Migration Challenges
- Resources
- Buy vs. Build

Comparing Legacy and Linux Architectures

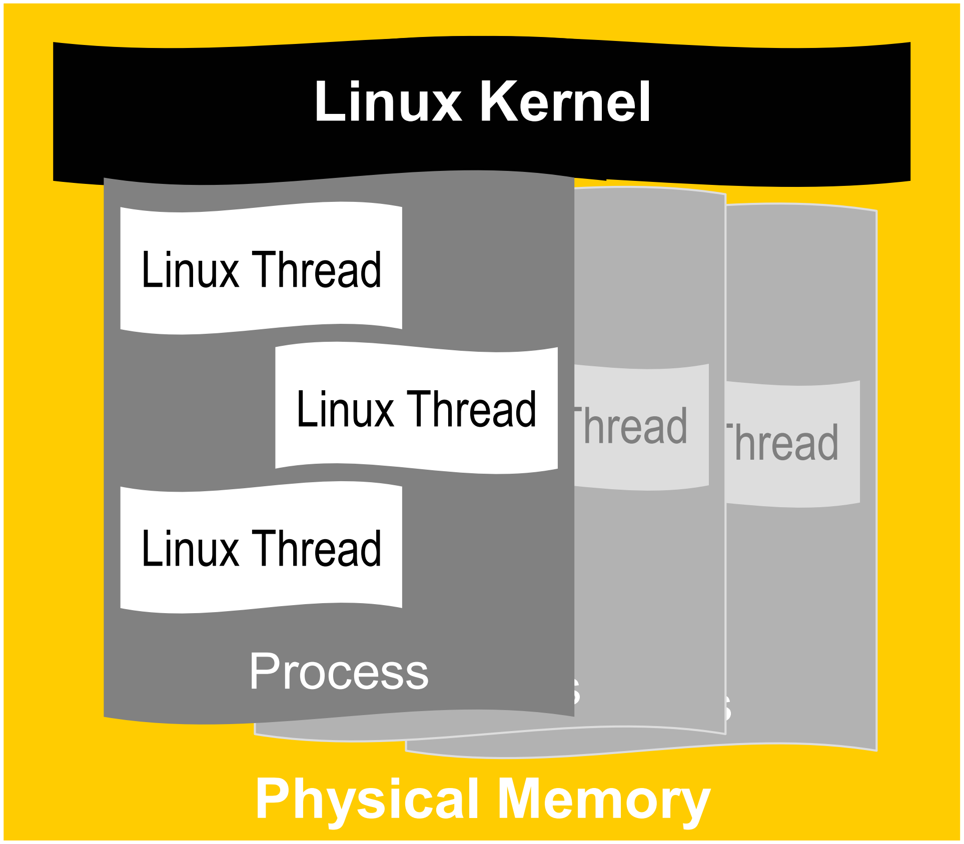
Vintage Legacy RTOS System Architecture



- Kernel & Applications run in physical memory
  - Copy from ROM to RAM at boot
  - Execute from RAM
- Little or no differentiation in user and system contexts
  - Tasks and kernel run in priveledged context, for performance
  - System calls are just “JSRs”

Linux System Software Architecture

- Kernel/Apps run in virtual address space(s)
  - Kernel copied from storage at boot time
  - Apps loaded as needed
  - Processes are containers
  - Tasks → Linux threads
  - All code, const data is RO
- Strongly differentiated execution contexts
  - User-space processes see only local address space
  - Only kernel context or root can access physical addresses

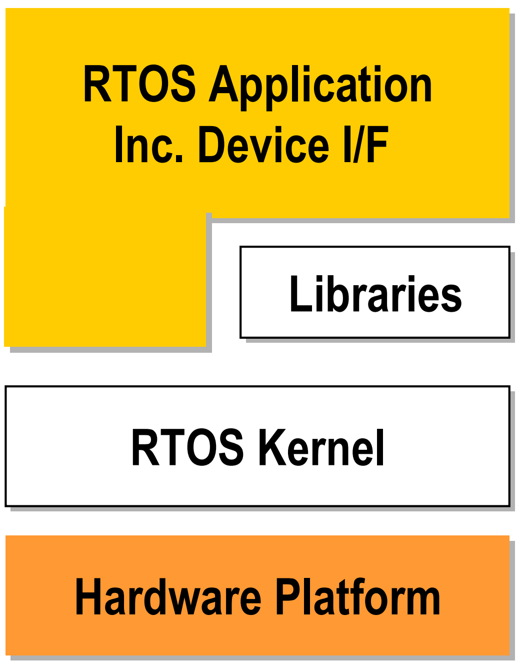


Linux Pundit
Open Source Opinion & Analysis
Business & Technical Consulting

Notions of Context

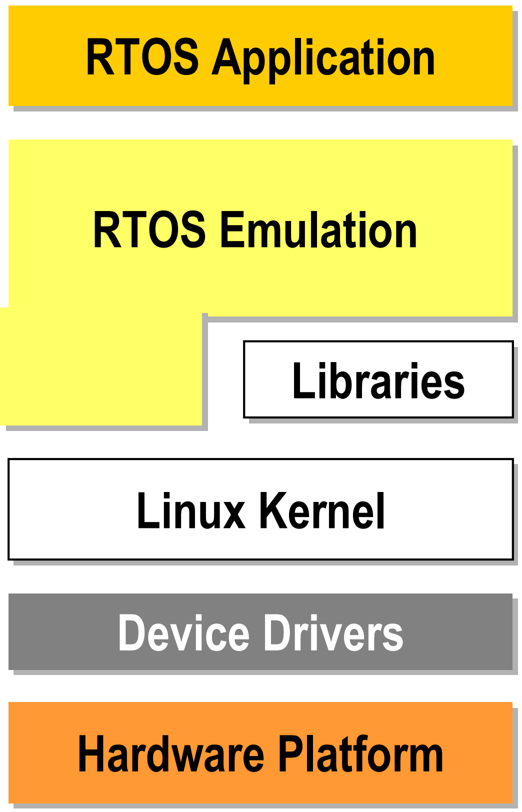
Scheduling Unit	Context	Addressing
RTOS Task	Registers, PC, SP	Physical
Linux* Thread	Registers, PC, SP	Logical (in process address space)
Linux Process	Context of 1st Thread, Memory Mapping	Logical
Linux Kernel Thread	Kernel address space	Logical

RTOS Run-time Architecture / Stack

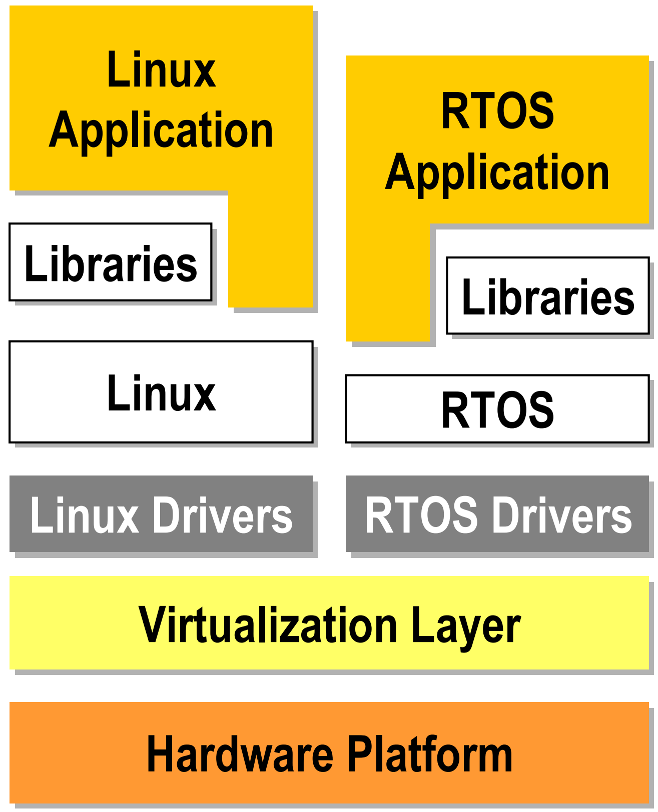


- Simple Architecture
  - S/W components linked as monolithic executable
- Key Question
  - Where and how does application code execute with Linux?

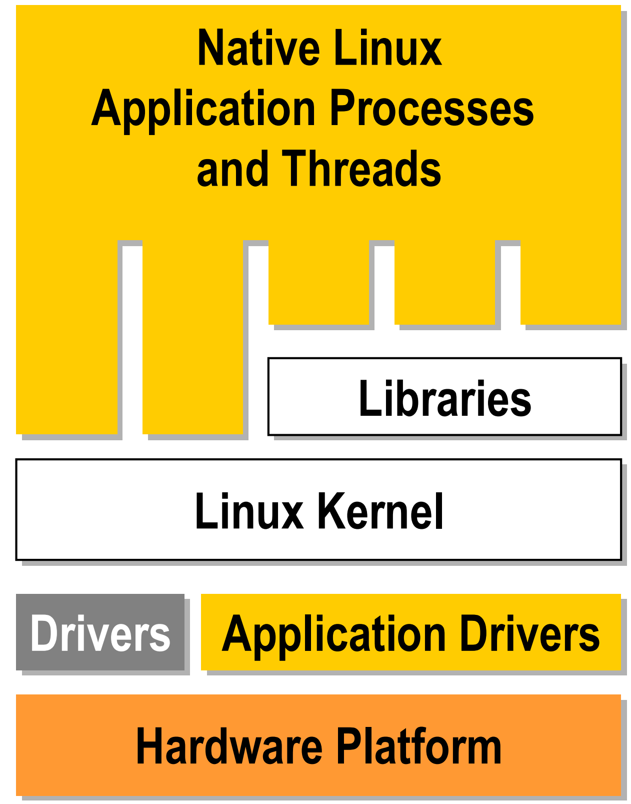
Migrated Run-time Architectures



RTOS Run-time Emulation over Linux

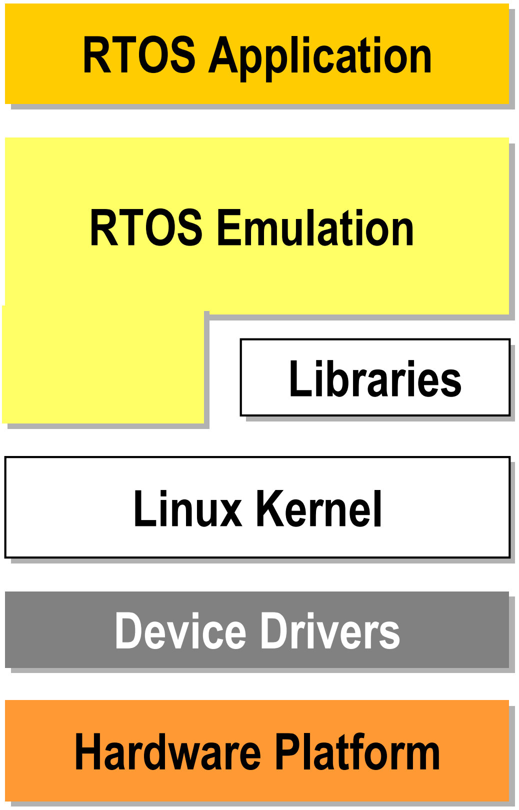


Partitioned Run-time with Virtualization



Complete Native Linux Port with Threads and/or Processes

RTOS Run-time Emulation

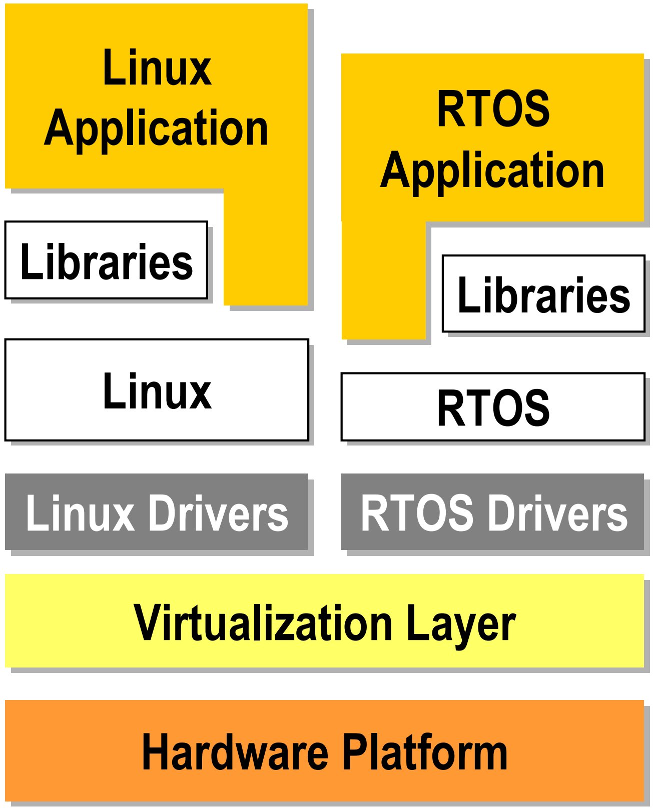


RTOS Run-time Emulation over Linux

- Two flavors of emulation
  - Library-based
    - Native Linux library functions emulate individual RTOS APIs and system calls
    - E.g., OSChanger, etc.
  - Whole-RTOS
    - RTOS code runs as sole linkable w/n a Linux process
    - Application code runs “on top” of process-based RTOS (cp. Java)
    - E.g., VxELL (VxSim)

Migrated Run-time Architectures

Partitioned / Virtualized Run-time



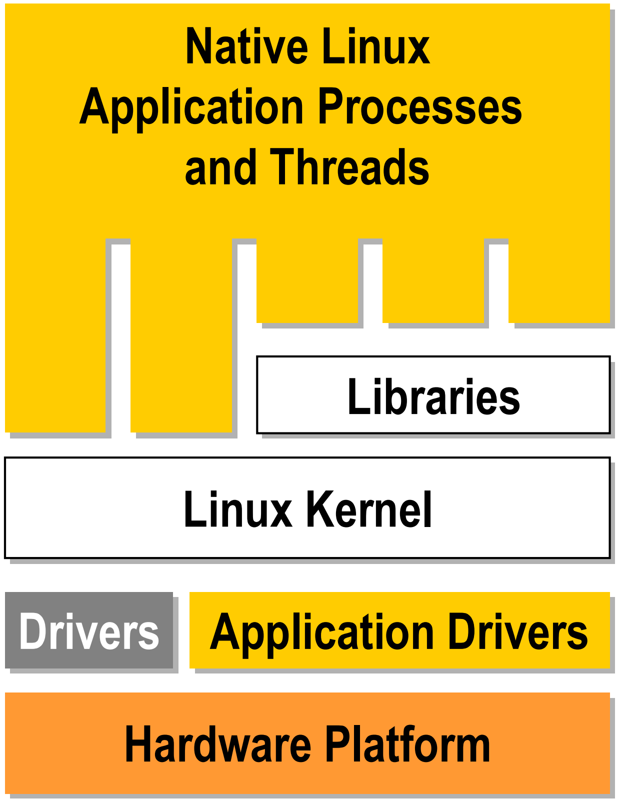
Partitioned Run-time with Virtualization

- Virtualize CPU, Memory, Interrupts into 2 or more partitions
- Partition “0”
  - Legacy RTOS for real-time and 100% API compatibility
- Partitions “1..N”
  - Instance(s) of Linux or other “application OS” for next-generation functionality
- E.g., Jaluna OSware, misc. OSS projects

Migrated Run-time Architectures

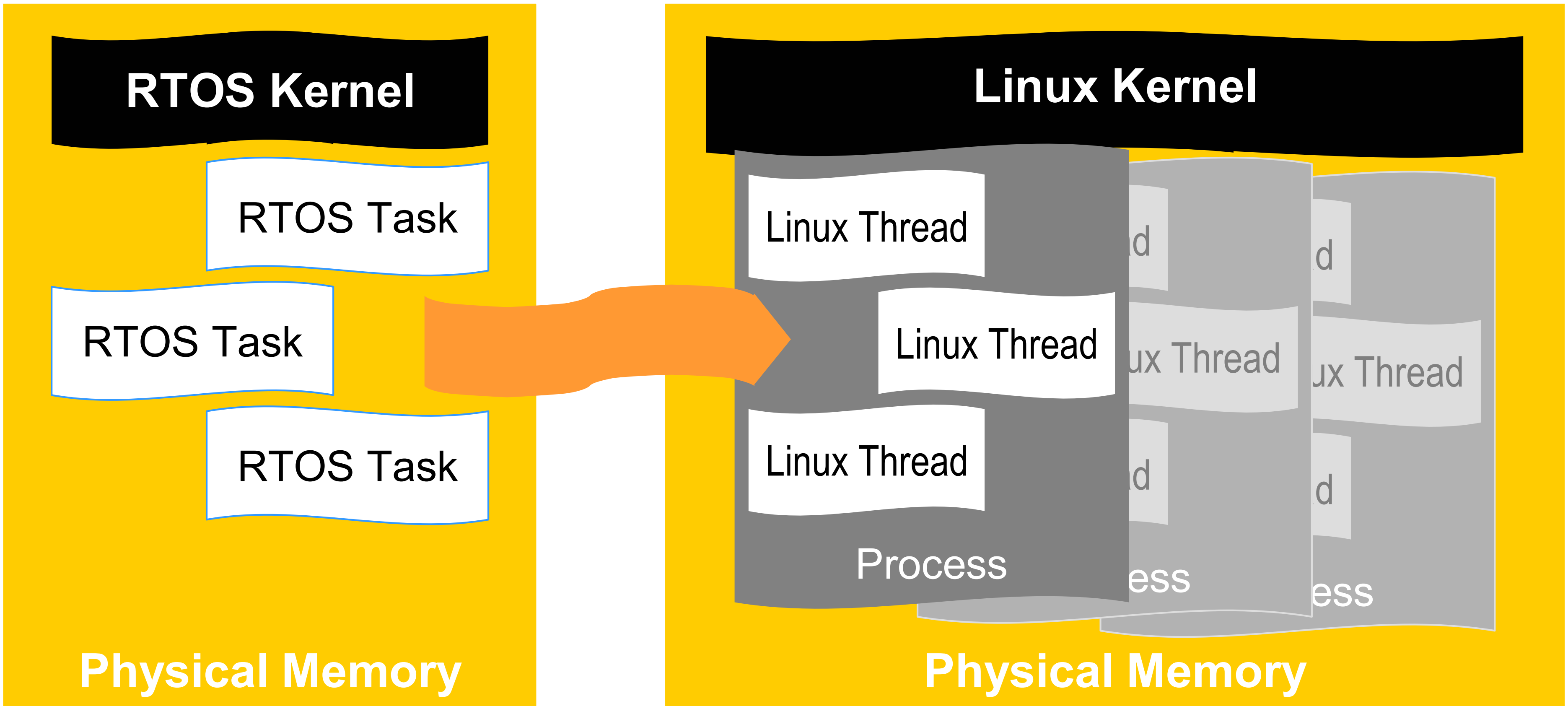
Native Linux Port with Processes/Threads

- Migrate legacy RTOS application to run as native Linux code
  - Map/emulate APIs
- Re-architect as needed
  - RTOS tasks migrate to multiple Linux processes & threads
  - Re-write I/O code as native Linux device drivers
- Greatest investment, greatest potential benefit
  - Resulting code is more portable with greater longevity



Complete Native Linux Port with Threads and/or Processes

Mapping Legacy RTOS Tasks to Linux Processes and Threads



Initial Porting Effort:
Single Process, Multi-Threaded

- Most logical migration architecture
  – Move legacy task code over to execute as Linux threads in a single Linux process
- Legacy-to-Target Application
  – One-to-one mapping of tasks to threads
  – Legacy and target both build in single C or C++ name space
  – Continued use, as needed, of data sharing through global variables
  – Options for migrating h/w i/f code to Linux drivers or in-line use
- Scales to multi-board legacy systems
  – Each legacy CPU board maps tos own process
  – Runs under Linux on a same or next-generation CPU

Example RTOS-to-Linux API Mapping: VxWorks to Linux

Call Type	VxWorks API	Linux Equivalent
Task Creation	taskSpawn()	pthread_create() or fork()
Instance Message Queue	msgQCreate()	mq_open()
Acquire Semaphore	semTake()	semget() and sem_wait()
Wait	taskDelay()	sleep() and nanosleep()

Mapping RTOS ITCs to Linux IPCs and Inter-thread Mechanisms

RTOS Inter-Task	Linux* Inter-Process	Linux Inter-Thread
Semaphores (Counting and Binary)	SVR4* Semaphores	
Mutexes		pthread Mutexes, Condition Variables, FUTEXes
Message Queues and Mailboxes	Pipes/FIFOs, SVR4 queues	
Shared Memory with formal mechanisms or through named data structures	Shared Memory with shmop() calls or with mmap()	Threads share name data structures in a process-wide namespace
Events and RTOS Signals	Signals, RT Signals	
Timers, Task Delay	POSIX timers/alarms, sleep() and nanosleep()	

Key Migration Challenges

- Real-time
- Time Management
- Footprint
- Device Interfacing
- Development Tools

Linux and Real-time

- Linux is not an RTOS, however . . .
- Linux does satisfy 87% of developer RT needs (VDC)
  - Soft real-time and preemption latencies
  - Interrupt response
  - Context switching
- Native Linux Real-time Capabilities
  - Preemptible Linux and the 2.6 kernel
  - FUTEXes and Robust Mutexes
  - New developments
- Linux Enhancement Technologies
  - Sub-kernels and third-party modules
  - Running an RTOS and Linux together with virtualization

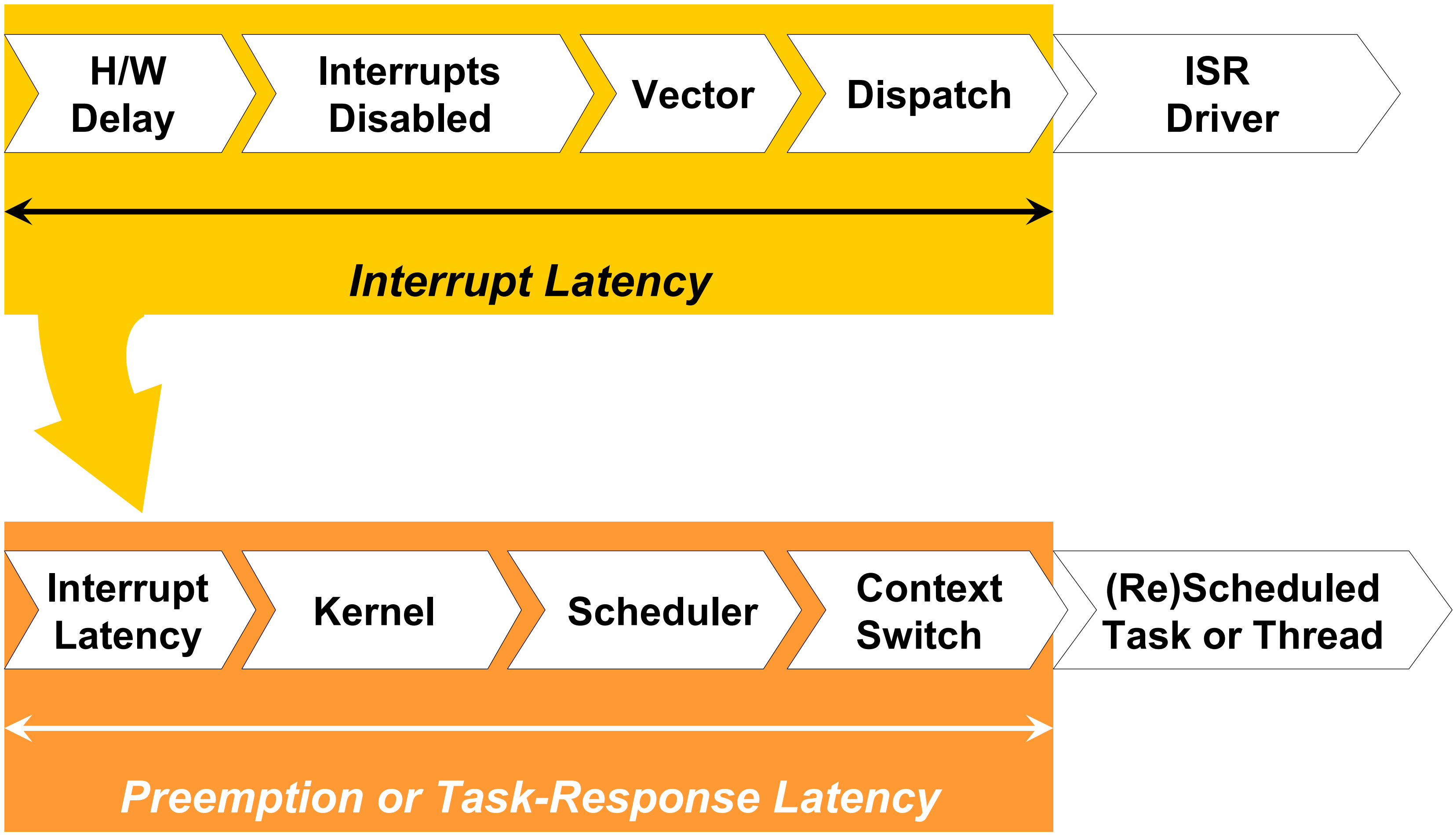
Real-time Requirements Satisfied by Linux for Respondents Using Linux, VDC, Natick MA

What is Real-time?

- Real-time vs. Real-fast
  - No absolute measure of real-time "speed"
  - Key are notions of worst case, determinism, and jitter
  - Some trade-off between responsiveness and throughput
- Hard vs. Soft Real-time
  - Hard real-time requires/provides guaranteed worst case
  - Soft real-time represents statistical results from best effort
    - System meets deadlines 99% or 99.999% of the time
- An OS is not "real-time"
  - OSes can enable applications to meet deadlines, or not
  - Most measurements are not of performance, but of how fast an OS can "get out of the way"

Real-time Terminology

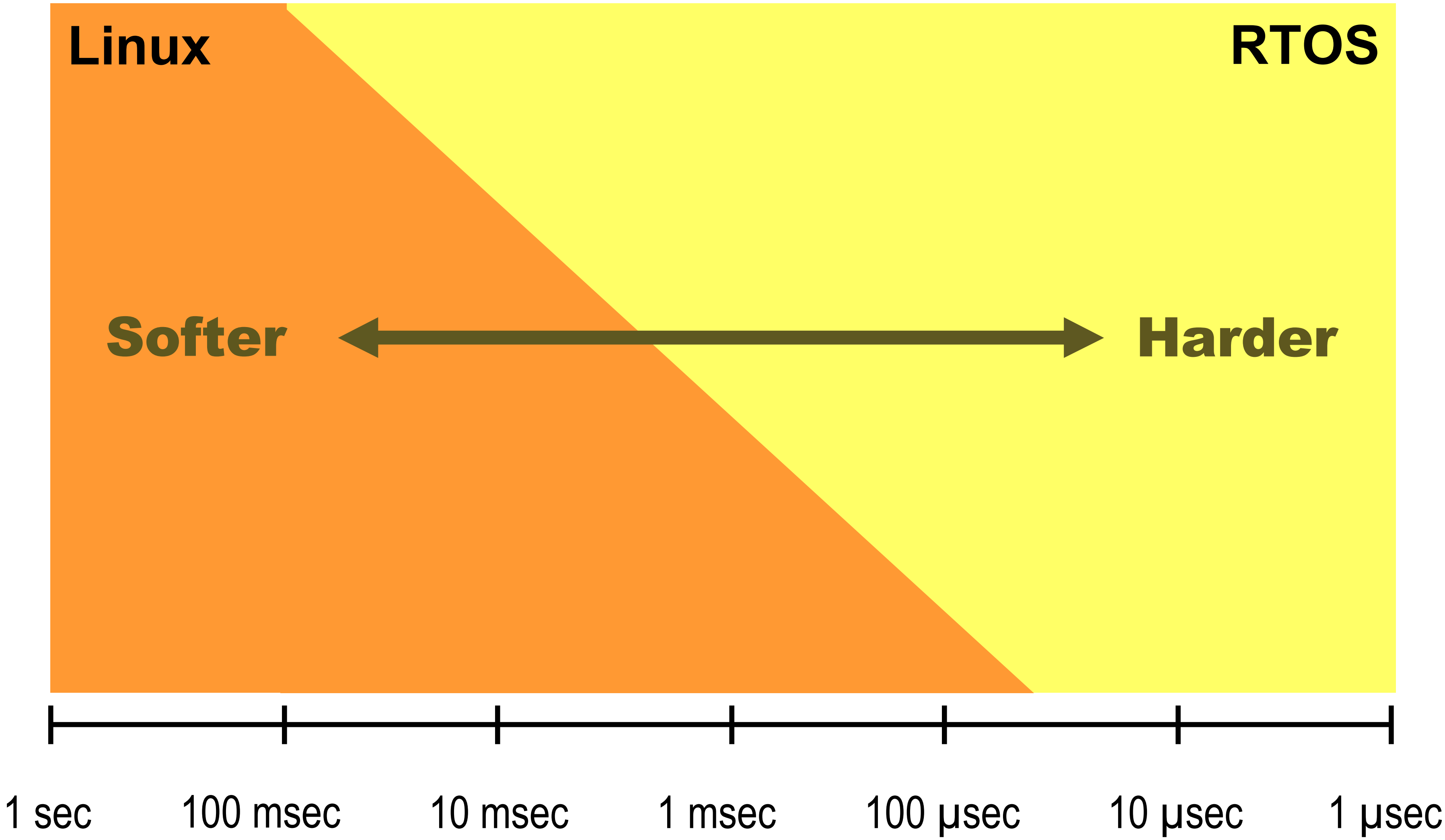
Interrupt and Preemption Latency Constituents



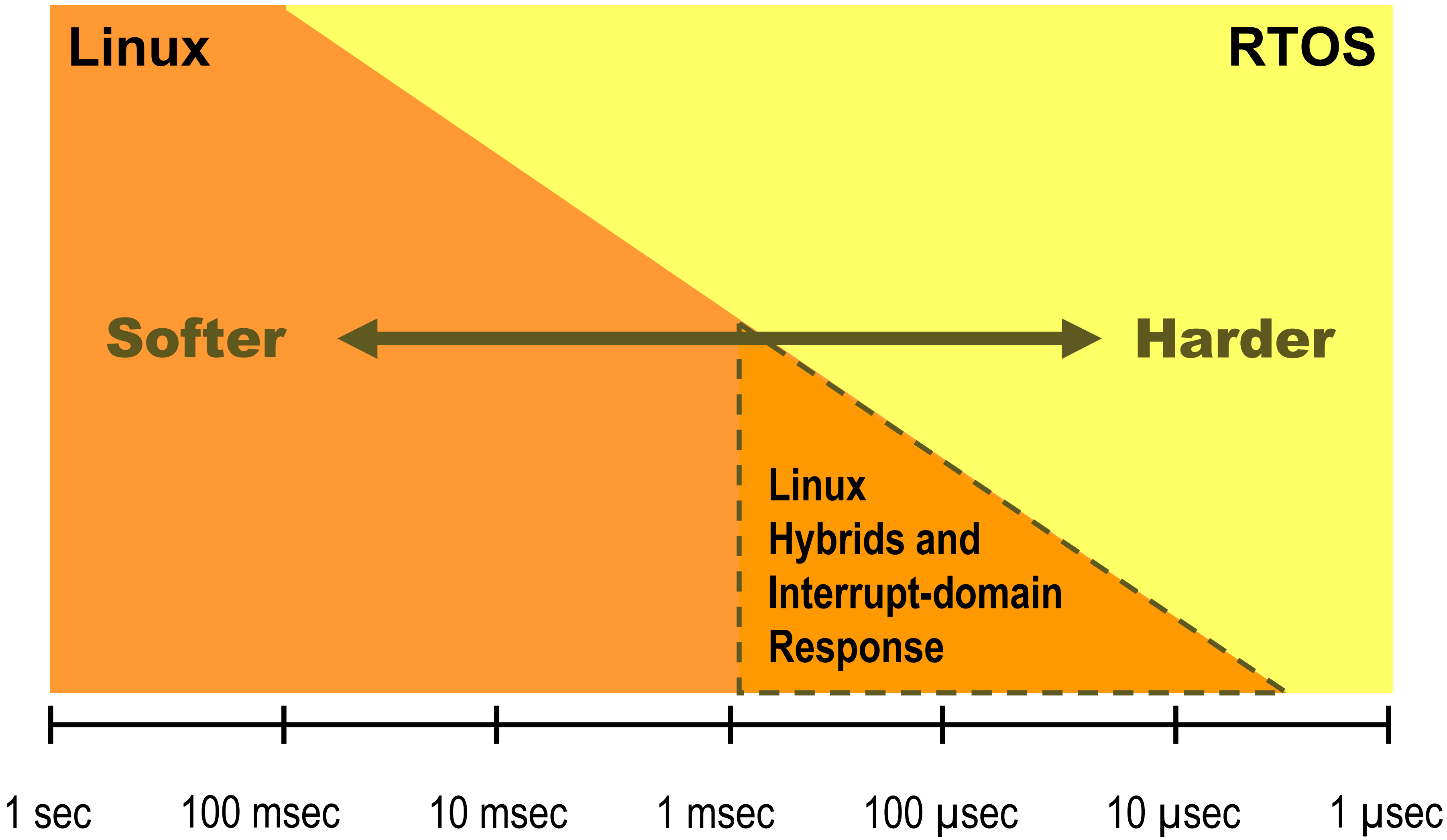
Linux Pundit
Open Source Opinion & Analysis
Business & Technical Consulting

Will Linux ever perform like an RTOS?

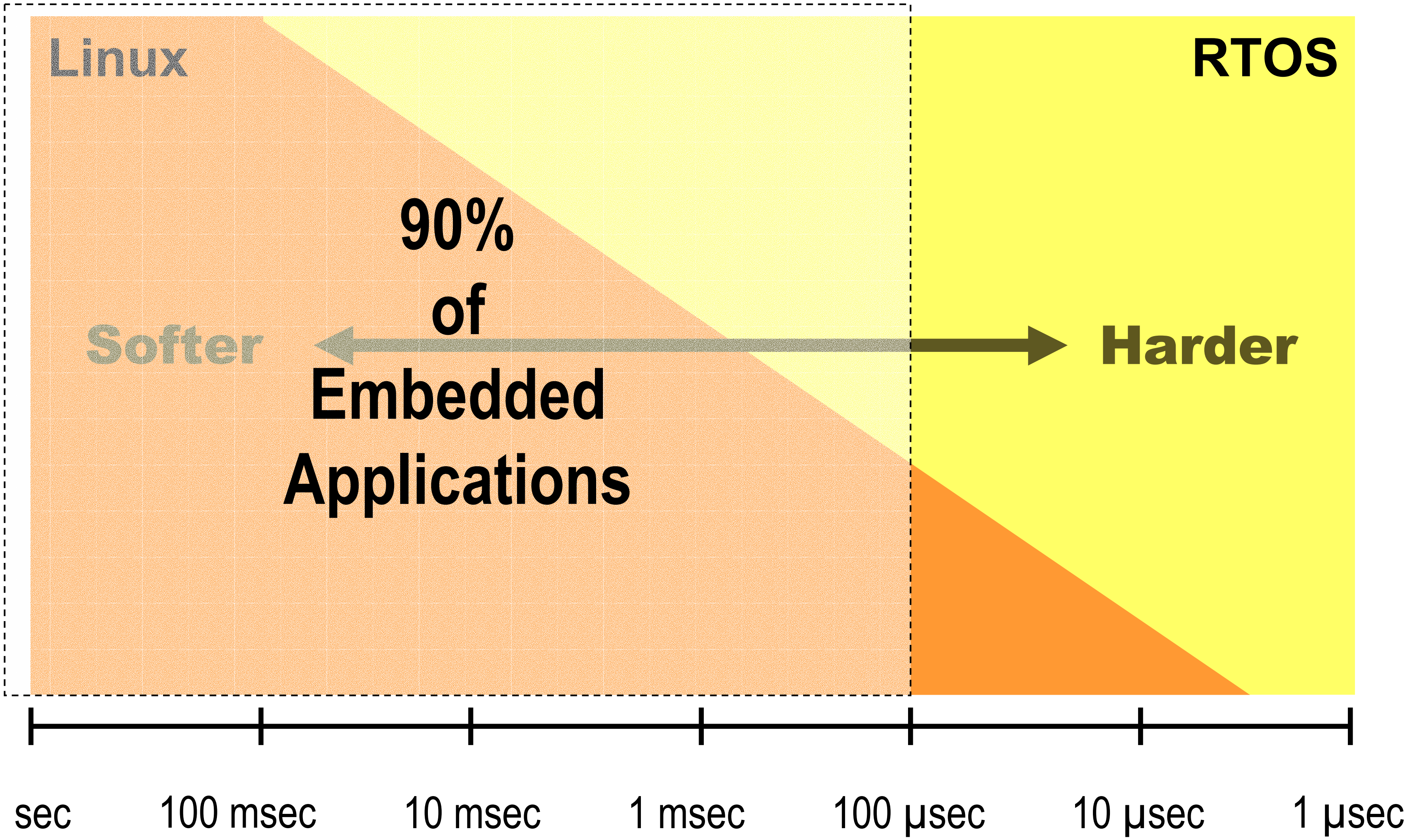
Linux Real-time Responsiveness



Linux Real-time Responsiveness (continued)



Linux Real-time Responsiveness (continued)

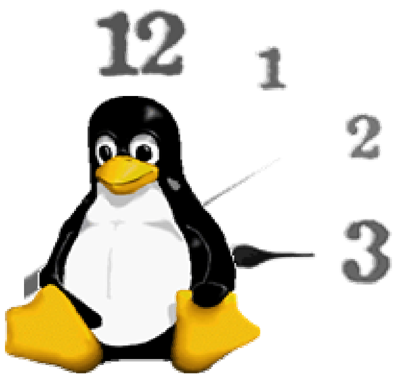


Clocks and Timers

- Timer resolution
  – VxWorks*/RTOS APIs quantify time in terms of system clock ticks, precise resolution
  – Linux* uses wall clock time and/or “jiffies”
- Auxiliary clock
  – VxWorks/RTOS offer auxiliary clocks
  – e.g., sysAuxClkConnect()
  – Allows connection to a second clock running at user-defined speed/resolution

Clocks and Timers Solution

- Timers
  - Linux kernels easily manage large numbers of timers with low overhead
  - Set interval timers via setitimer()
  - Use hardware-based timers
  - Use sub-kernel timers



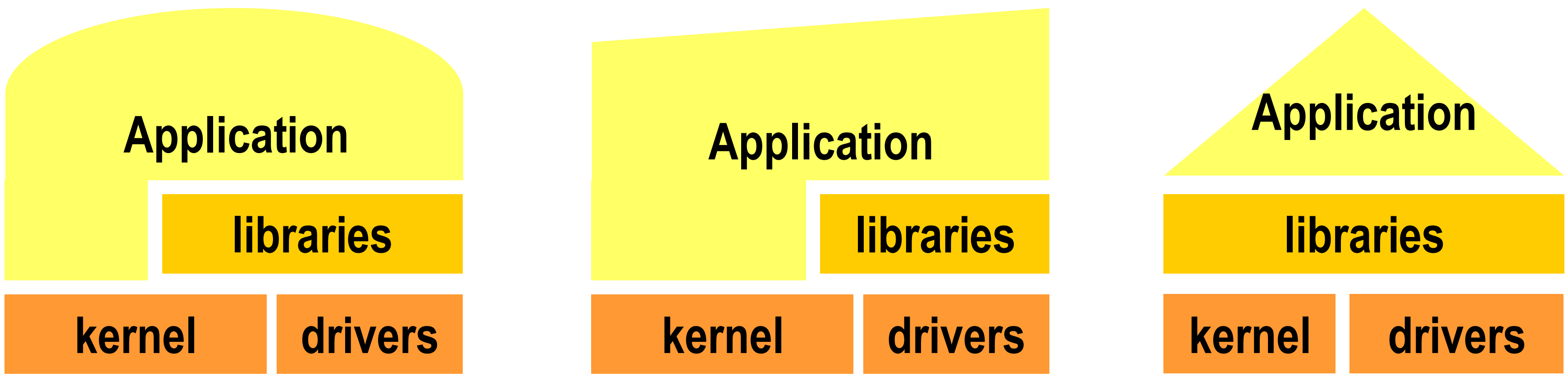
- Auxiliary Clock
  - Emulated via a thread looping on nanosleep()
  - Maximum resolution is ~1 ms
  - Finer resolution can be achieved using POSIX Timers
    - http://sourceforge.net/projects/high-res-timers/

Will Linux ever fit in embedded footprints?

Footprint

A legacy RTOS is not a platform

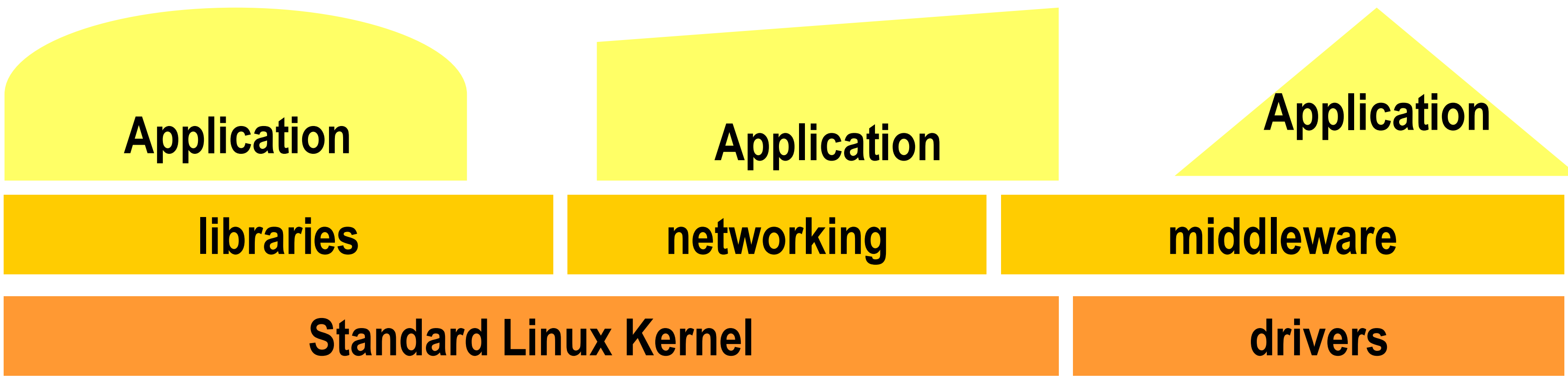
- RTOSes Scale DOWN well, because
  - RTOSes are collections of services
  - Each application only deploys services it needs
- Each RTOS-based design is *sui generis* – a one-off



Footprint

Linux is a platform, par excellence

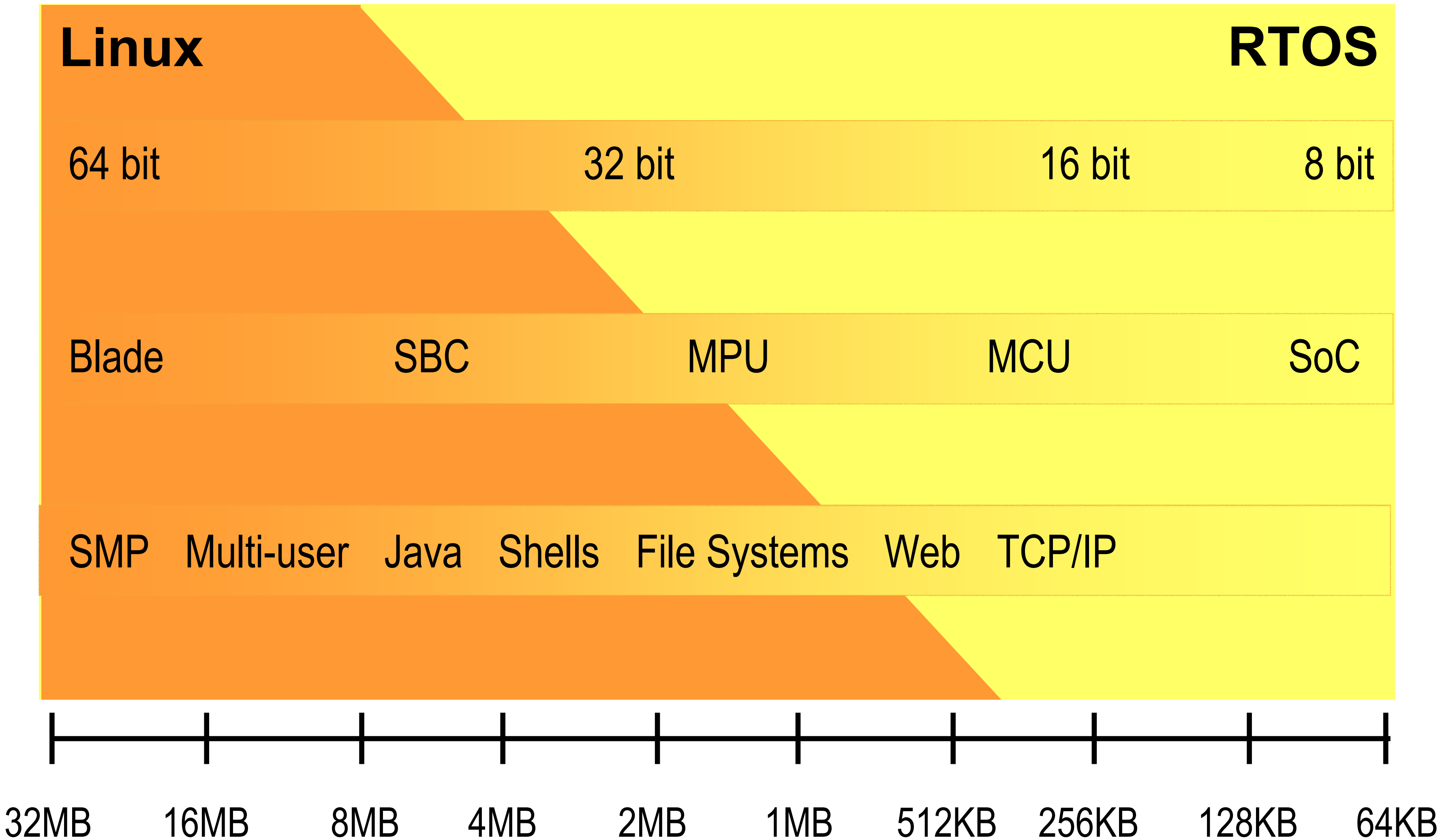
- All versions of Linux present same services, APIs
- Each application uses services it needs
- Any Linux implementation can run 1000s of applications



Embedded Linux = Enterprise Linux = Desktop Linux

Footprint

What you get out depends on what you put in

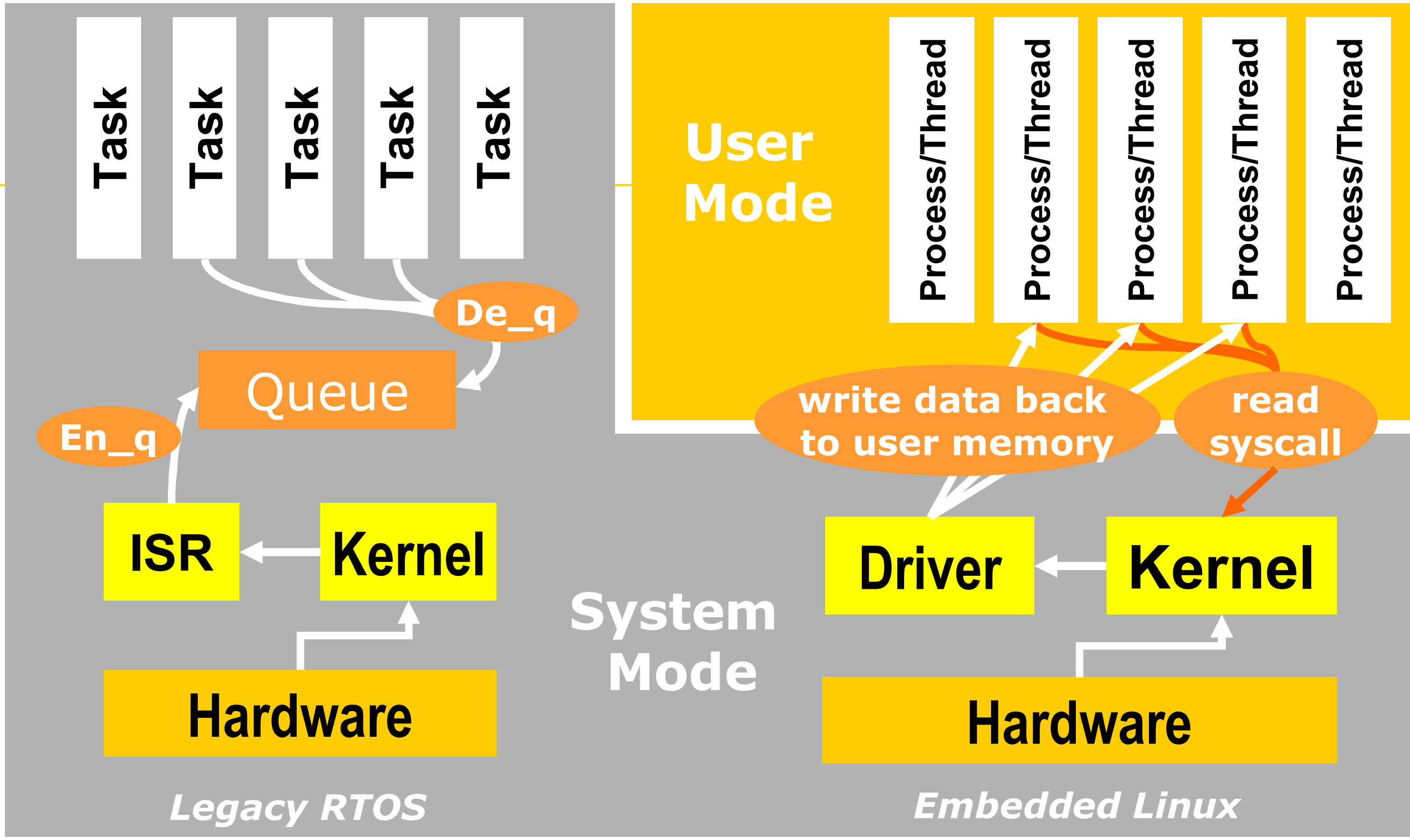


Vintage Legacy RTOS Device Code

- Device interface code tends toward informal
  - Read/write/catch inline with application code
  - No strong differentiation between system and “user code”

- Many legacy RTOSes have a driver model
  - Often ignored by device OEMS
  - Main audience is/was IHVs

RTOS vs. Linux H/W Inter-face



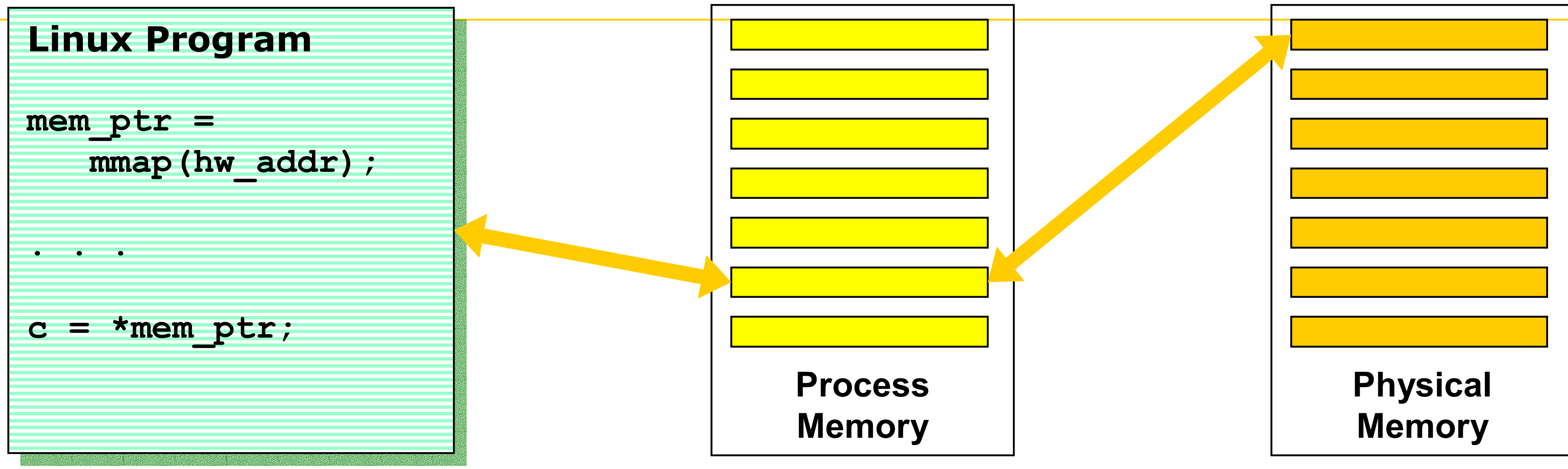
- RTOS application has access to machine address space, memory mapped devices and I/O instructions
- Linux systems use a device driver model for this functionality

Graphic from “Porting RTOS Device Drivers to Embedded Linux,” Linux Journal

Hardware Interface Solutions

- Convert legacy interface into Linux driver
  - Usually requires re-architecting
  - Can be involved process
- Port legacy interface using Linux mmap()
  - "Quick and dirty" – interface runs in user space process
  - Good for single read/write or polled device access
  - Not a good path for interrupt handling
- Use existing Linux user space subsystems
  - E.g, for USB

Direct Memory-Mapped I/O with Linux




```
#include <sys/mman.h>
#define REG_SIZE   0x4          /* device register size */
#define REG_OFFSET 0xFA400000   /* physical address of device */

void *mem_ptr;                  /* de-ref for memory-mapped access */
int fd;                         /* file descriptor */

fd=open("/dev/mem",O_RDWR);     /* open phys memory (must be root) */

mem_ptr = mmap((void *)0x0, REG_SIZE, PROT_READ+PROT_WRITE,
                 MAP_SHARED, fd, REG_OFFSET);
                                /* actual call to mmap */
```

Development Tools Challenges

- Good News
  - Linux and OSS offer developers thousands of tools and utilities
  - Many device software developers already know these tools
    - GCC, GDB, bash, TCP/IP utilities
- Less-good News
  - Not all are appropriate for device software
  - Most are CLI; few are integrated

Migration Resources

- Open Source Community
  - Projects around CPUs, tools, APIs
  - Informational web sites
- Semi-conductor & SBC Manufacturers
  - Maintain / Contribute to Architecture Trees
- Independent Software Vendors (ISVs)
- Peripheral & Chipset Providers (IHVs)
- Integrators & Professional Services Orgs.
- Linux Platform / Tools Providers
  - Distributions, Embedded Linux, and DSO

Migration Resources

Buy vs. Build Decisions with Open Source Linux