Using Time and Resource Partitioning to Ensure Guaranteed Access and to Build Highly Reliable Embedded Systems

May 5, 2008

Edward Lee
Field Applications Engineer
QNX Software Systems

QNX SOFTWARE SYSTEMS

Survivable

- In an ideal world, all code is bug free
  - “I always write my code to be perfect”
  - Reality often has a way of intruding …
- ***Survivable*** platforms can recover from an unexpected circumstance or fault
  - Fault prevention
    - Design, debug, test
  - Fault detection & containment
  - Fault tolerance
    - Usually a system design issue, coupled with OS hooks
  - Fault recovery

High Availability: Basic Formula

$$\text{Availability} = \frac{\text{Mean Time Between Failure}}{\text{Mean Time Between Failure} + \text{Mean Time To Repair}}$$

- 99.999% availability: <5 minutes down time per year
- Increase in MTBF increases availability
  - Software MTBF is a function by defect density: defects / KLOC
  - MTBF improved by good software quality practices
- Decrease in MTTR increases availability
  - As MTTR approaches zero, availability of component approaches 100%

Other Availability Factors

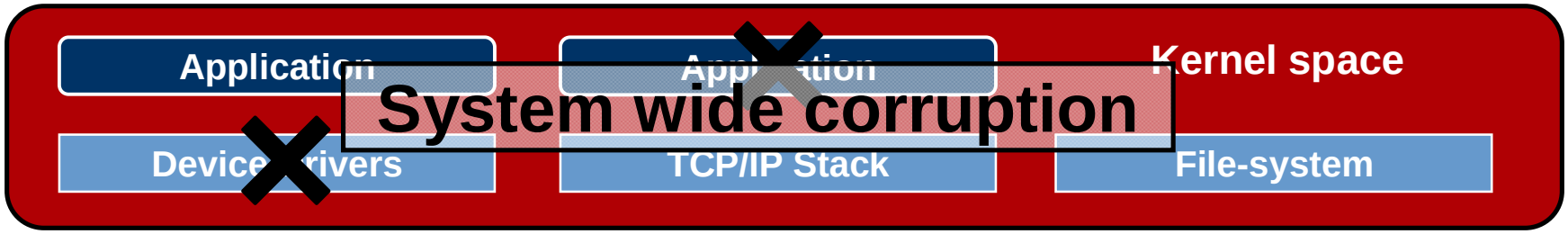
- Parallel (redundant) components increase availability
  - Parallel Availability = 1 – (1 – *component availability*)^2
  - If a component has 99% availability, a redundant configuration has an availability of 99.99%
- Reducing impact of failure also increases availability
- Serial dependencies where failure of one component causes unavailability of dependent components decreases availability
  - Serial Availability = A1 * A2
  - Two components with 99% availability arranged in a series results in 98.01% availability

Increasing Software Availability

- Software quality practices to reduce MTBF
- Decoupling software components in runtime system results in lower probability of failures affecting service
- Reduce MTTR of software components in runtime system
- Reduce serial dependencies in runtime system
  - Minimize impact of serial availability
- Use redundant software components where possible
  - Maximize benefit of parallel availability
- Reduce impact of failures in runtime system

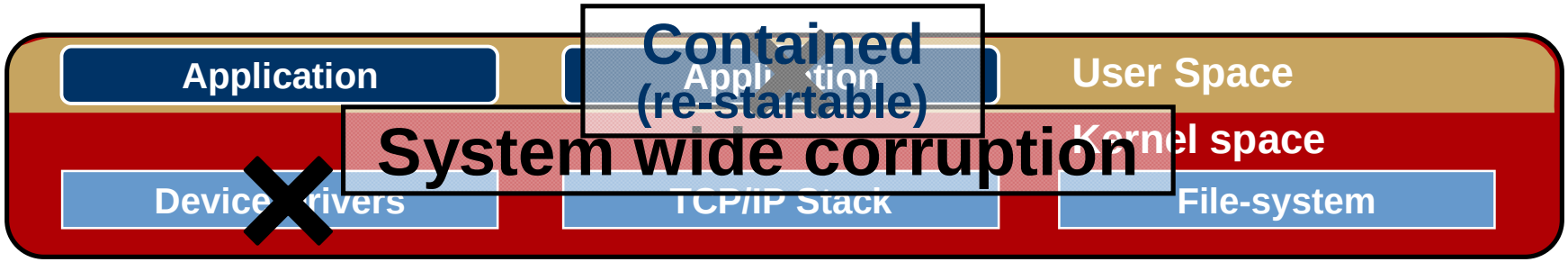
The Impact of the Operating System Architecture on Availability

Real-Time Executive Architecture



- One single piece of software
- Applications, Drivers, all software components are all built and loaded into the Kernel space
- If any software component fails, the entire system will reset
- High MTTR & Low Availability – as any software error will require a complete system restart to recover

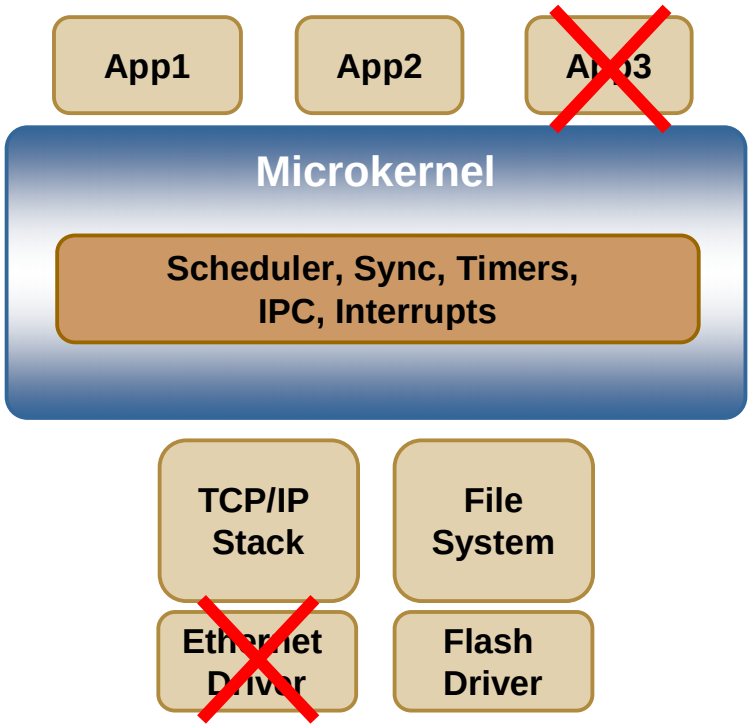
Monolithic Kernel Architecture



- Applications are independent software components
- Device Drivers and System Resources are still built into the Kernel space
- Application errors don't impact entire system, as they can be independently restarted and recovered
- Failure in any Device Driver or System Resource will still reset the Kernel, requiring a system restart to recover
- High MTTR & Low Availability – as any key software errors will require a complete system restart to recover

Micro-Kernel Architecture and Availability

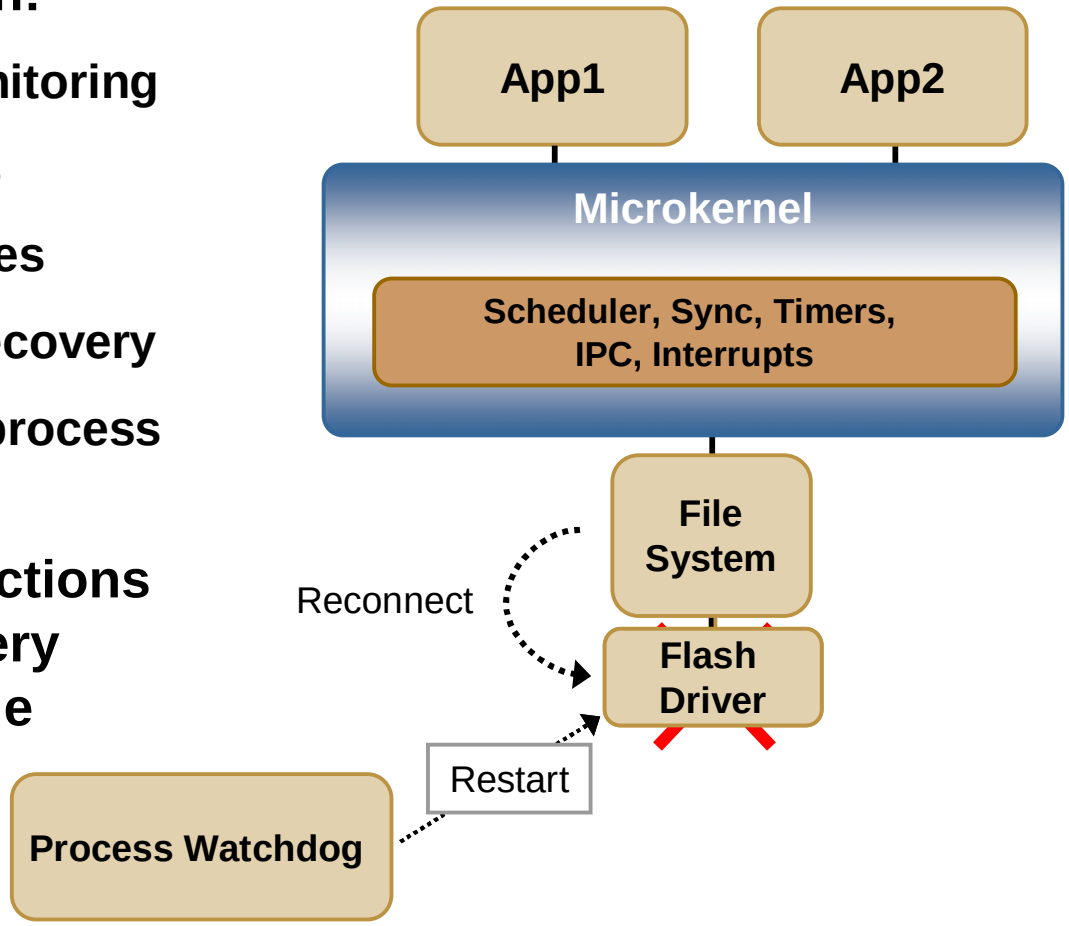
- **Applications and Device Drivers are all independent software components**
- **All software components can be restarted and recovered**
- **Low impact of driver and OS service failures – only applications using the failed service or driver are affected**



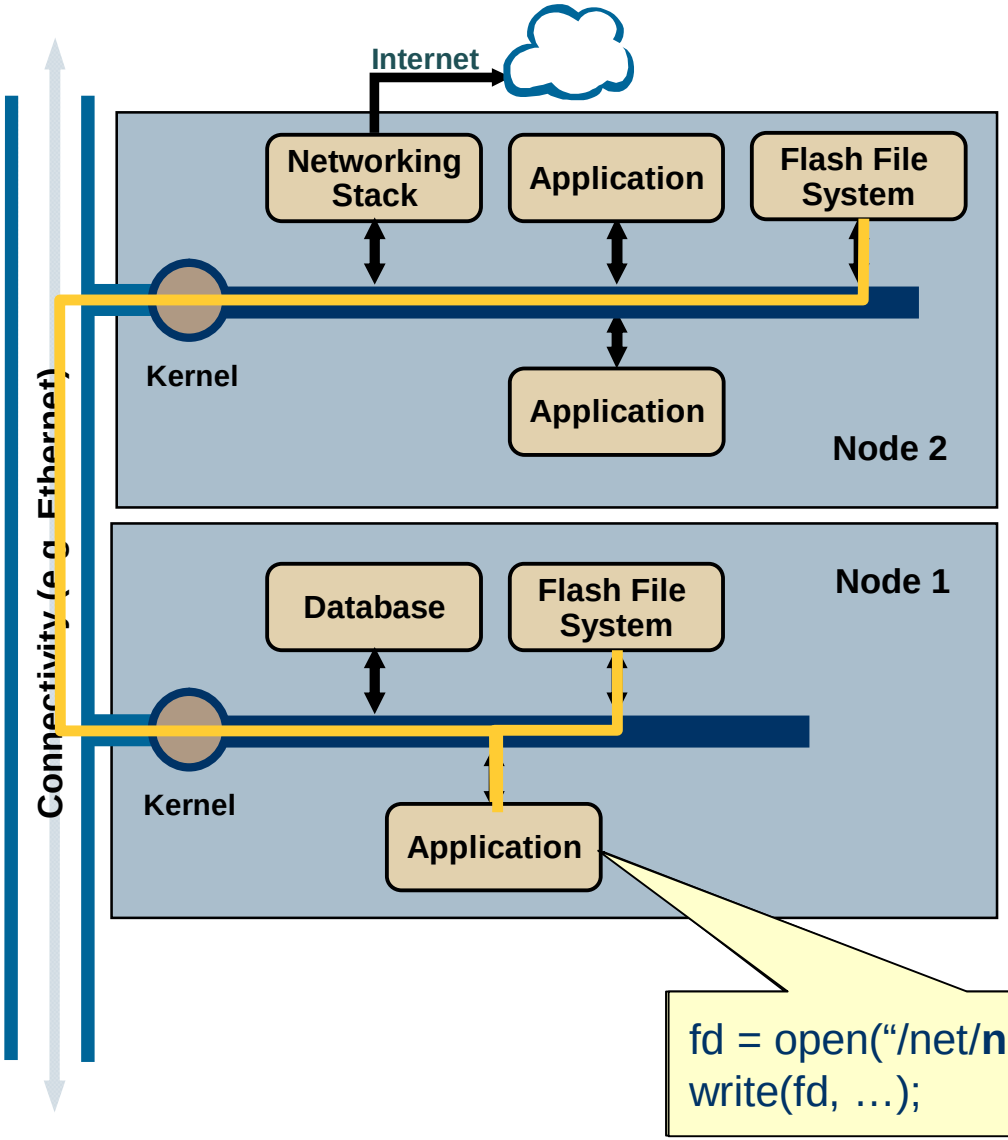
- **Low MTTR & High Availability – only the affected portion of system needs to be restarted**
- **By further reducing MTTR, Availability is further improved**

Reducing MTTR

- **Process watchdogs can:**
  - **Provide heartbeat monitoring**
  - **Detect process deaths**
  - **Restart failed processes**
  - **Perform multi-stage recovery**
  - **Send notifications of process failures**
- **High availability connections with connection recovery procedures are possible**



Distributed Processing for Redundant Services



- Distributed processing extends message passing over a transport layer
- Auto discovery of all nodes and published resources over Ethernet or other interconnect
- Global pathname space for networked resources
- Seamless sharing of I/O resources between nodes (e.g. use flash memory located on another node)
  - Networking stack
  - File systems
  - Hardware ports
- Increase availability through redundancy

```
fd = open("/net/node2/dev/ffs1",…);
write(fd, …);
```

Quick Summary

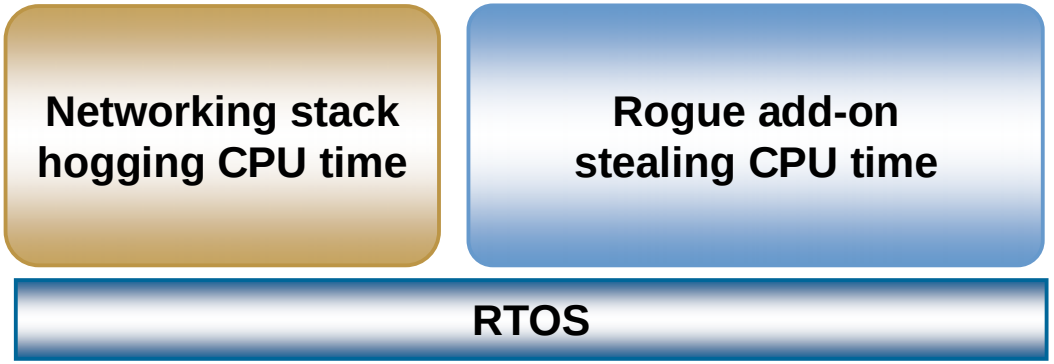
- A reliable system has high MTBF and MTTR
- QNX micro-kernel OS architecture promotes:
  - more reliable software → improves MTBF
  - fast and easy recover → reduces MTTR
- Process Watchdog provides automatic recovery, for both applications and dependencies, further reducing MTTR
- Distributed Processing provides redundancy

But is this Good Enough?

Other Problems

Security Threats

- Many embedded systems are network connected
  - Untrusted interfaces and network threats
  - Untrusted add-on software
- If appropriate measures aren't included by design, security and availability can be compromised
- Rogue software can launch denial of service (DOS) attack and starve core applications of CPU time, memory or other system resources
  - Need to contain untrusted, add-on software
- Distributed DOS attacks can busy your system with network processing



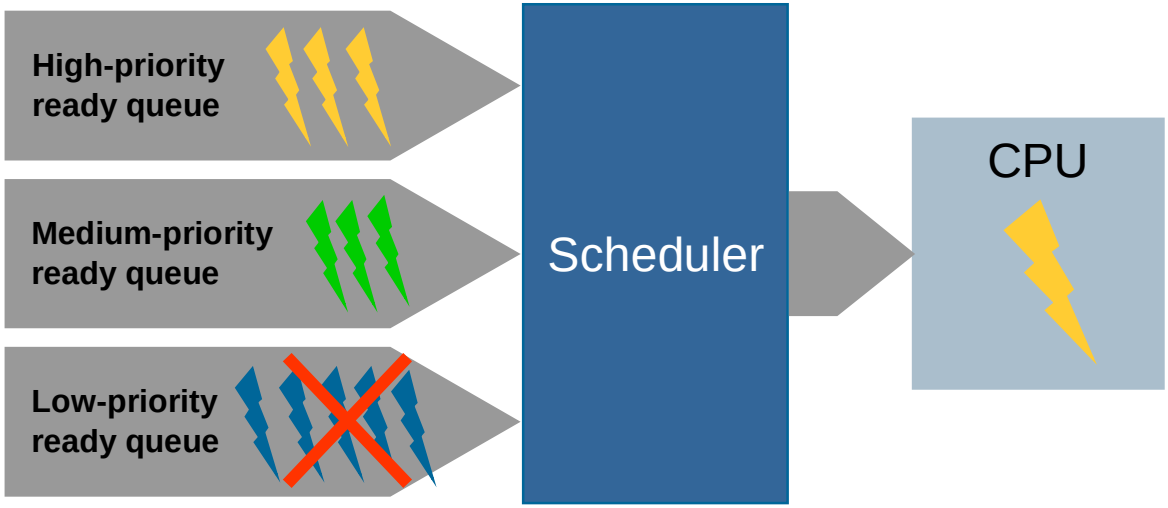
What Do Mean By Security?

- Security can have many meanings
  - Physical security
  - Network security
  - Cryptographic security
- In this presentation, “security” refers to the ability to protect applications running within a system from one another
- Applications can adversely affect another
  - By design — “hackers and crackers”
  - By circumstance — bugs, unexpected or unhandled situations, etc.
- Availability and security
  - If a system can be compromised by internal threats (either malicious or unintended), it is not secure

CPU Starvation

During Design and Implementation

- Difficult to determine how individual thread priorities will impact overall system
- Need to wait until system integration to test

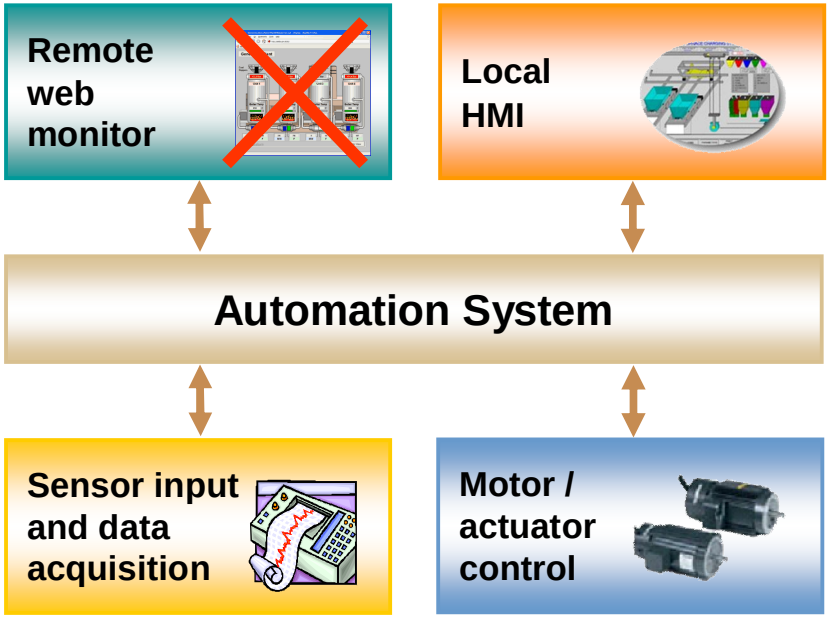


During System Integration

- All threads compete for CPU time
- Starvation of lower priority threads can occur
- Re-juggling priorities and re-writing code

Example

- Simple automation system
  - Remote web monitoring agent
    - low priority threads
  - Local Human Machine Interface (HMI)
    - medium and low priority threads
  - Sensor scanning & data acquisition
    - medium & low priority threads
  - Motor control
    - high priority threads
- During integration phase, remote monitoring agent works fine until the operator uses local HMI
  - Remote agent freezes and ceases to display updates
  - Troubleshooting reveals that when HMI commands result in a high level of motor control, remote agent stops getting CPU



CPU Starvation

- The question:
  - How many resource starvation problems do you discover in a project?
- How do you estimate this?
  - Many models for looking at defect densities, such as defects per KLOC or defects per function point
  - Provides an estimate for the number of defects, BUT how many would be of this type?
- Techniques for this are beyond scope of this session
  - Think about your project and your experience to determine how this applies to you

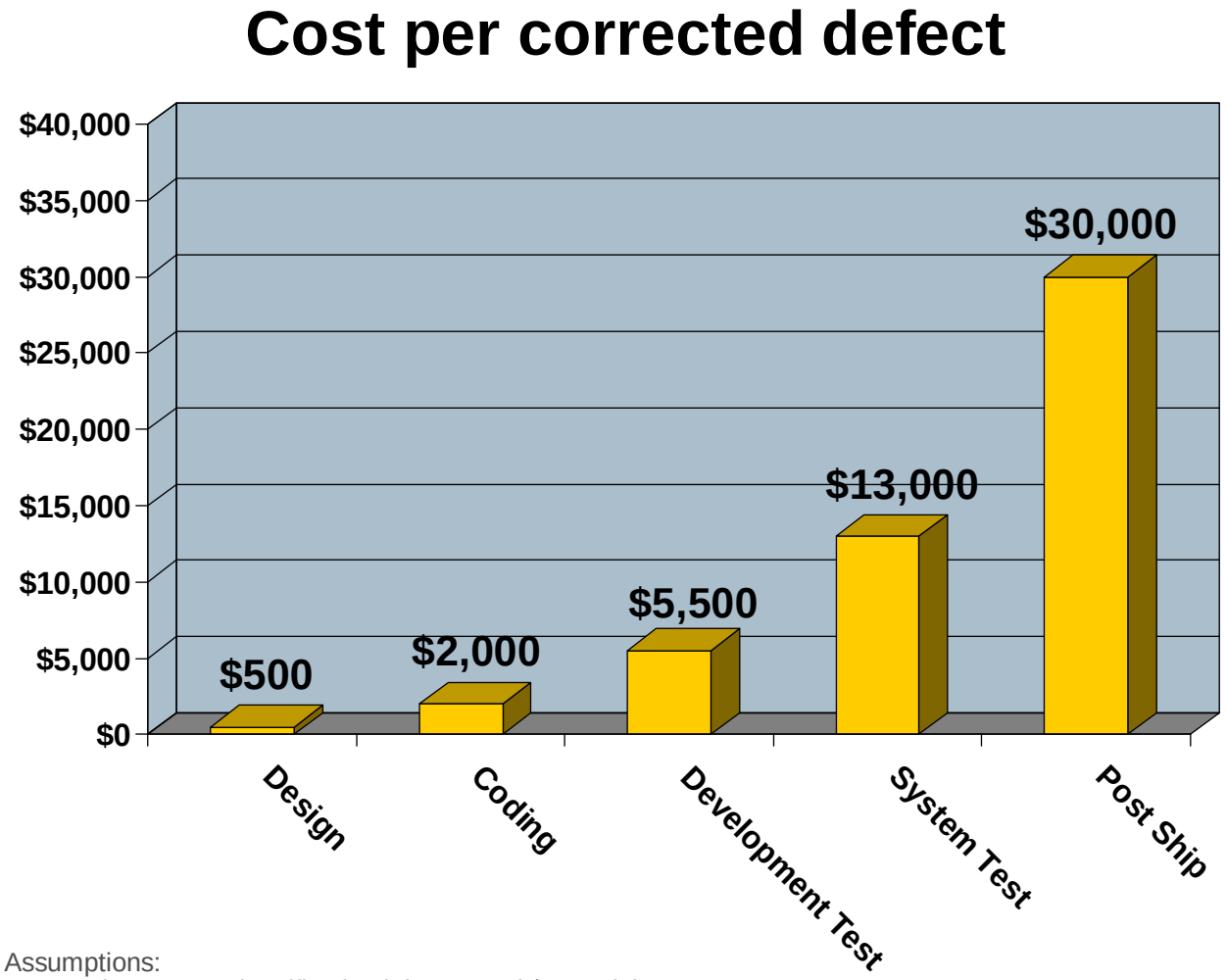
Software Defect Cost Model

	Relative cost of detecting and correcting defect				
	Design	Coding	Development Testing	System Testing	Post Ship
Pressman	1	6.5		15	60-100
Boehm & Basili	1	4	11	26	60

Sources:

- Pressman, Roger S., *Software Engineering, A Practitioner's Approach, 3rd Edition*, McGraw Hill, New York, 1992. p.559
- B. Boehm and V. Basili, "Software Defect Reduction Top 10 List," IEEE Computer, IEEE Computer Society, Vol. 34, No. 1, January 2001, pp. 135-137.

Software Defect Cost Model



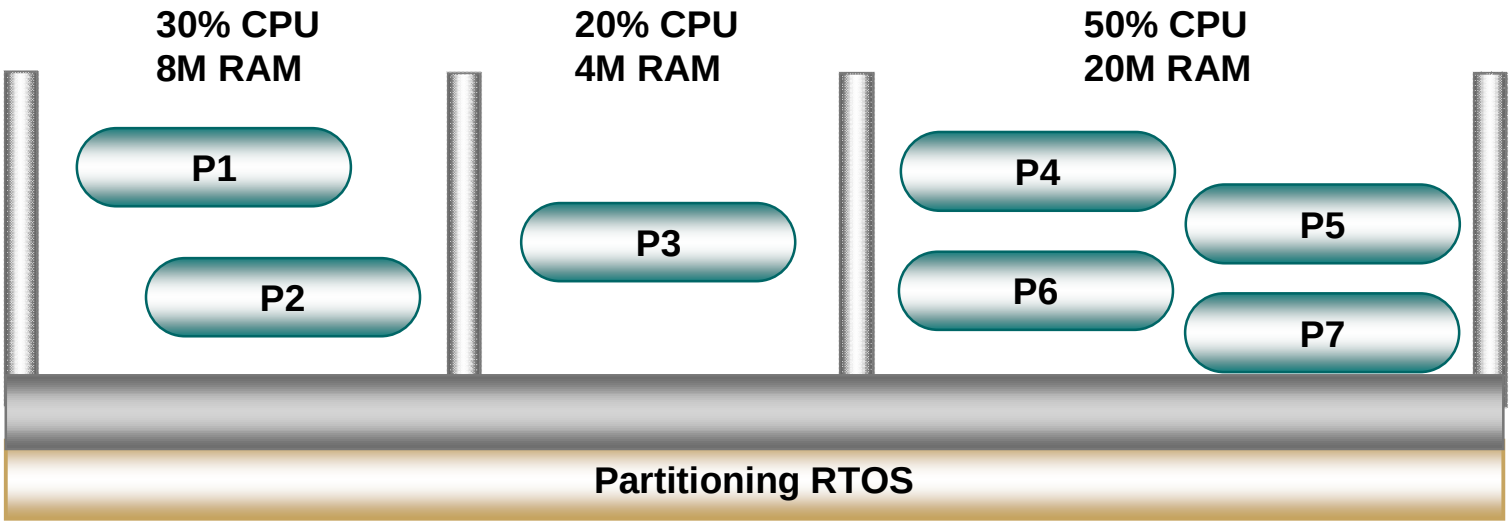
Assumptions:

- Development and verification labor rate of $1000 / day
- Industry average of finding / fixing a problem in the coding stage is ~2 days

Partitioning as a Solution

Partitioning Defined

- Partitioning provides a way for an OS to protect (or secure) compute resources
- Time partitioning — guaranteed CPU time for a partition
  - CPU time is shared among all processes/threads in the partition
  - Priority-based scheduling of threads within a partition
- Space partitioning — guaranteed memory for a partition
  - Memory is reserved for use by processes within the partition
  - Can explicitly cap memory consumption of a partition



Partitioning Approaches

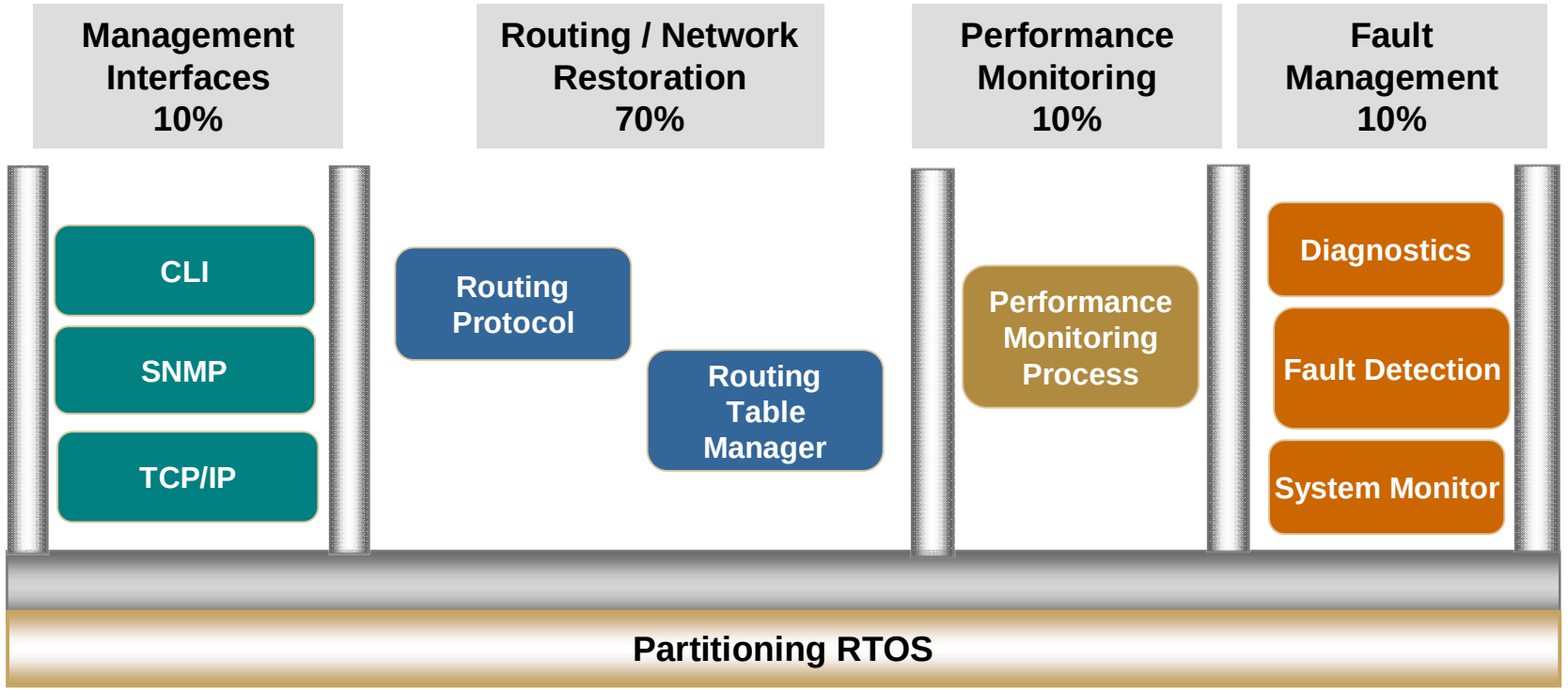
- Static time partitioning
  - Typically found in avionics, specified in ARINC 653
  - Partitions have fixed CPU time (for instance, 30% capacity)
  - When applications in a partition are idle, CPU time allocated to that partition goes unused
- Dynamic / adaptive time partitioning
  - CPU time not used by a partition is made available to other partitions
  - Reallocation of unused CPU cycles helps maximize performance of resource-constrained devices
  - Handles bursty CPU demands
- Memory partitioning is typically a fixed function where memory is reserved for a collection of processes

Introducing Adaptive Partitioning

- **What is Adaptive Partitioning?**
  - Adaptive partitioning is a new QNX product that extends the Neutrino RTOS
  - Allows you to build secure compartments or “partitions” around a set of applications or threads
  - Partitions enforce CPU guarantees for applications, controlled by easy to use budgets
- **Why is it Adaptive?**
  - Patent-pending design ensures all available CPU cycles are given to partitions that need processing time – no CPU cycles wasted
  - Provides performance advantage by permitting full processor utilization to accommodate spikes in demand
  - Critical processes can be configured to “borrow” CPU cycles in order to complete the required processing
- **Easy to get started**
  - ***No changes*** to how designers work today
    - POSIX programming model for the same, familiar design, programming & debugging techniques
  - ***No code changes*** are required to implement partitions

Beyond Security – Guaranteeing Remote Access

- Partitioning ensures remote monitoring and maintenance through the “Management Interfaces” partition are available when needed
  - Not susceptible to heavy CPU loads



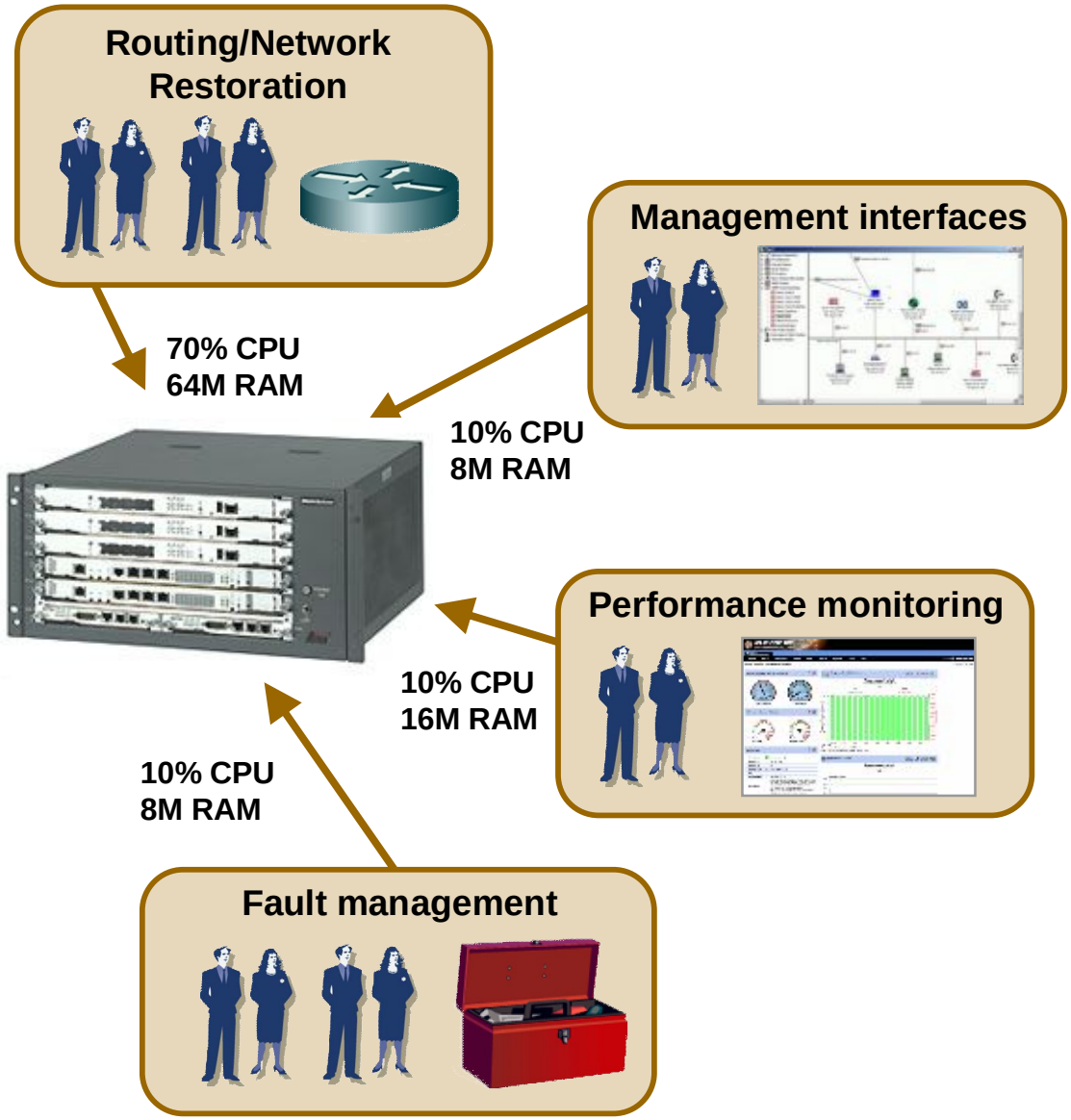
Designing with Partitions

System level design

- Time Partitioning
  - System designers allocate CPU time budget to subsystems / development teams
  - Eliminates need for system-wide priority schemes
  - Design teams can develop their own priority schemes within each partition
- Space partitioning
  - Memory limits established for each subsystem
  - Design teams must optimize within this constraint

Partitioning OS

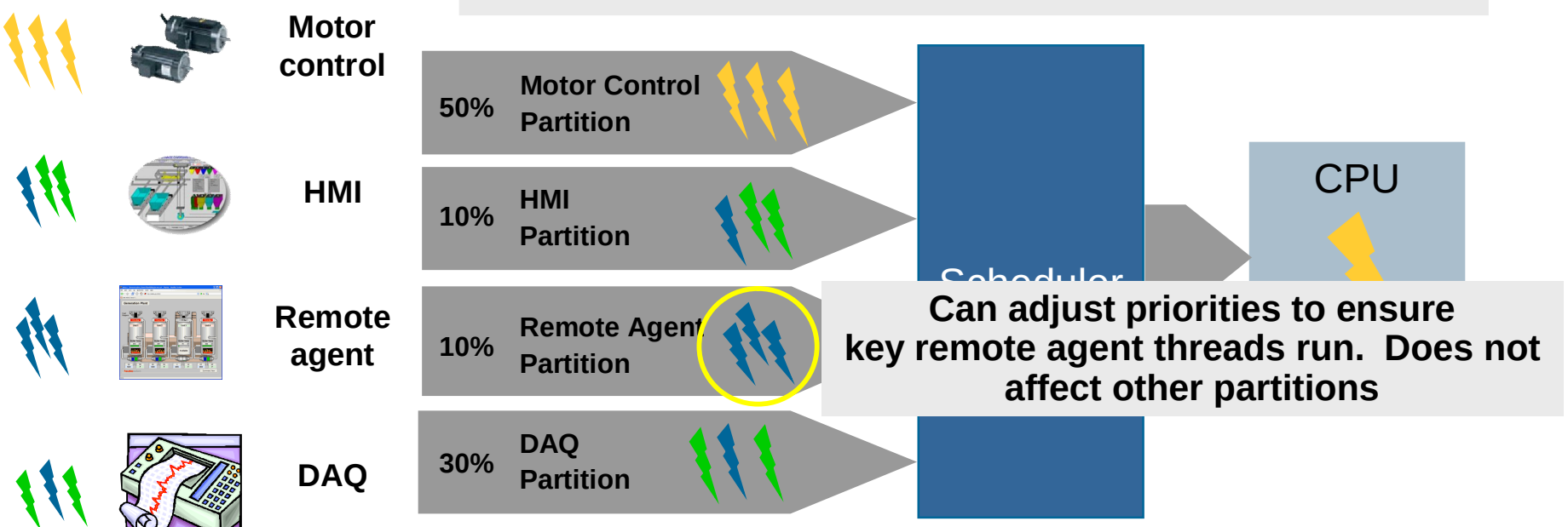
- RTOS enforces time and memory budgets



Partitioning to Guarantee Behavior

During Design and Implementation

- Budget appropriate CPU per partition
- Optimize priority assignment within partition
- Test using assigned budget



During System Integration

- Scheduler enforces CPU guarantees during overload
- Subsystems behave as designed and tested prior to integration – no re-juggling priorities and re-writing code
- Prioritize within partition to control behavior

Partitioning throughout the Development Cycle

- **Design / Architecture phase**
  - **Provides a method for managing CPU time & other resources**
  - **Removes “global prioritization policy”**
- **Unit Testing**
  - **Can configure system with appropriate budget and test for starvation issues on loaded system**
  - **Partition system to ensure test access**
- **Integration and system test**
  - **Avoids problems by design, allows detection earlier in development cycle**
- **Post ship**
  - **Partitions ensure critical system monitoring functions are guaranteed to run**
  - **Partitions ensure troubleshooting access to in-field systems**

Q&A

Ed Lee
elee@qnx.com