Lecture #23

Real Time Operating Systems

18-348 Embedded System Engineering

Philip Koopman

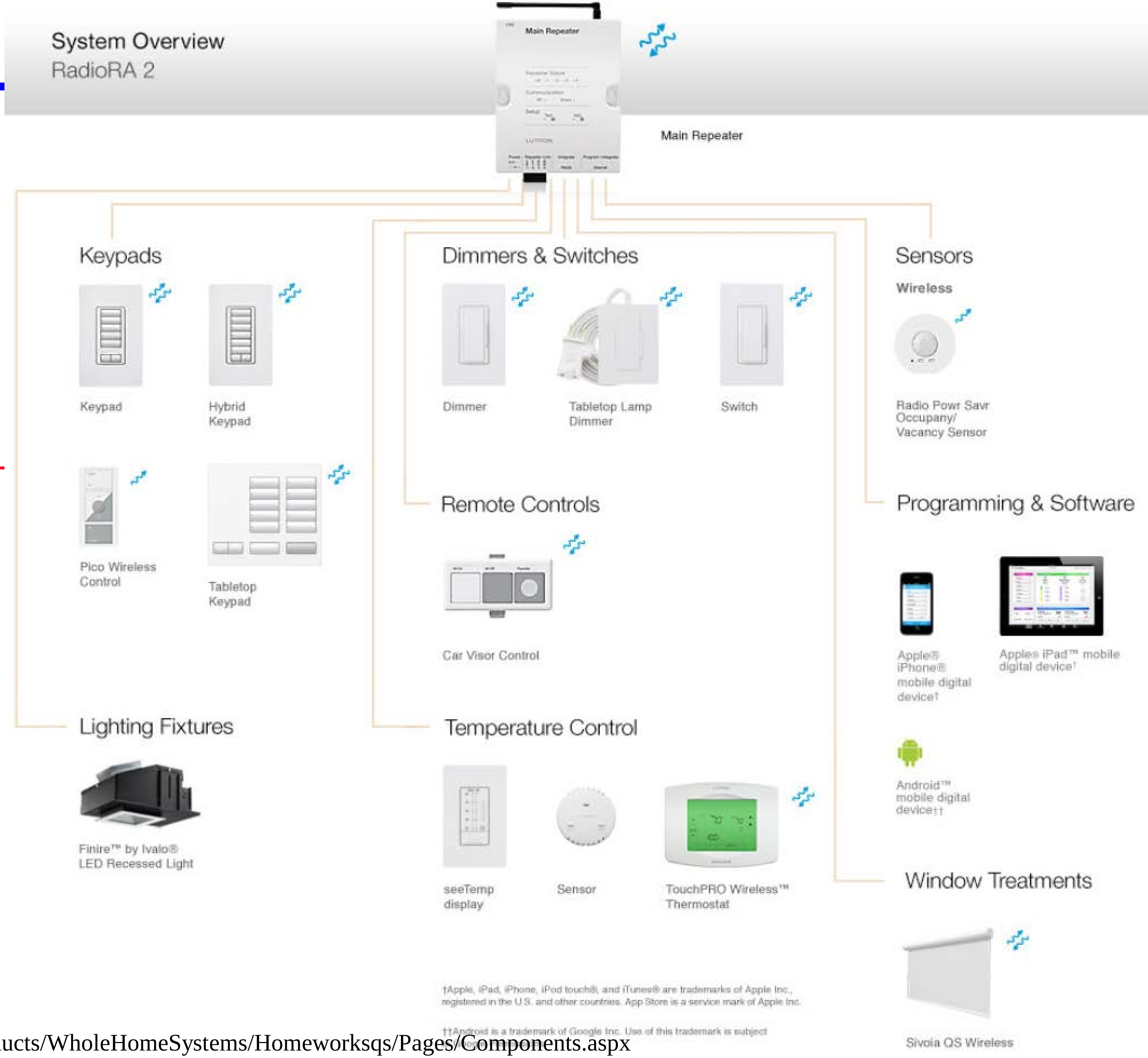
Monday, April 11, 2016

Electrical & Computer ENGINEERING

Carnegie Mellon

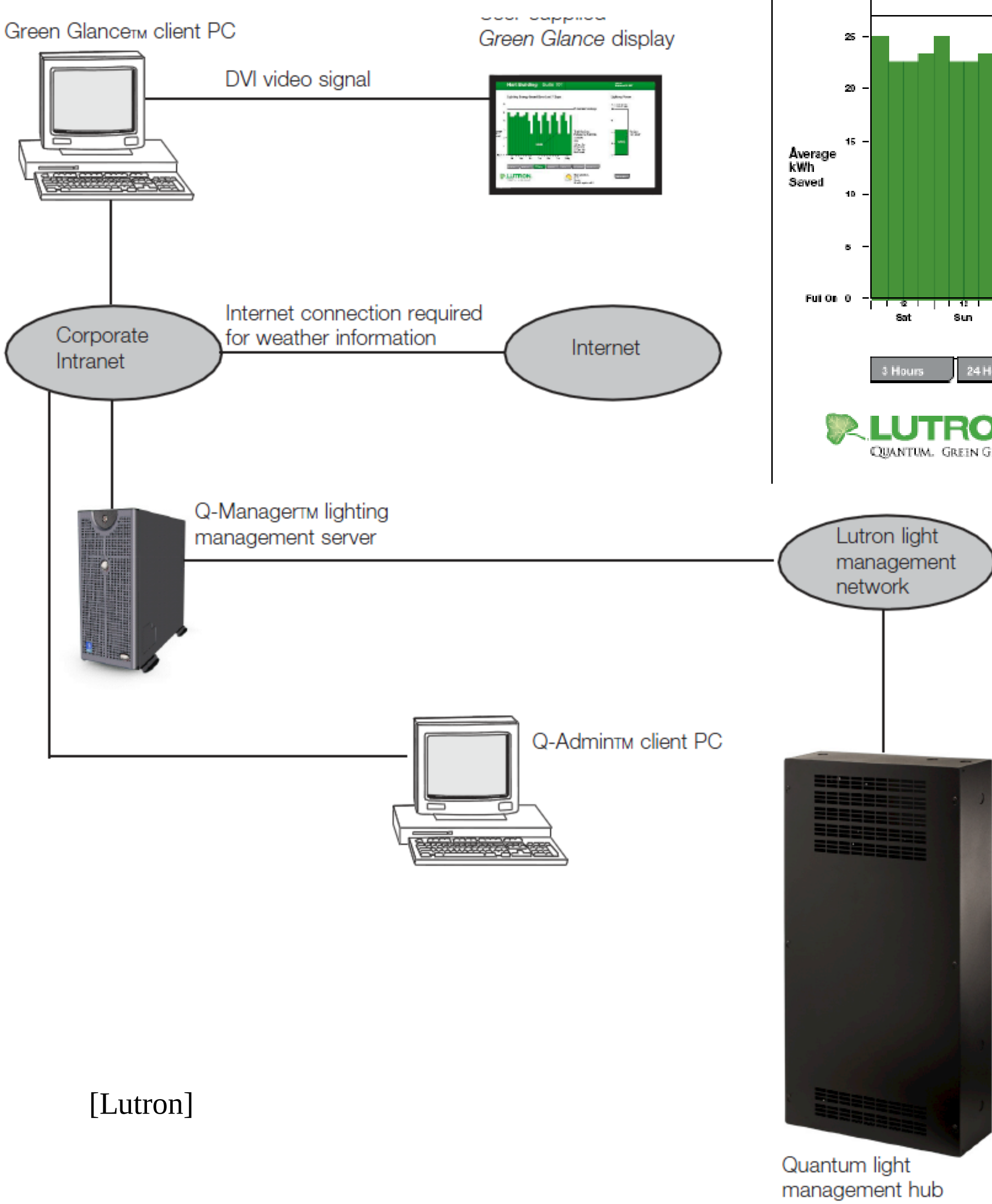
Lighting

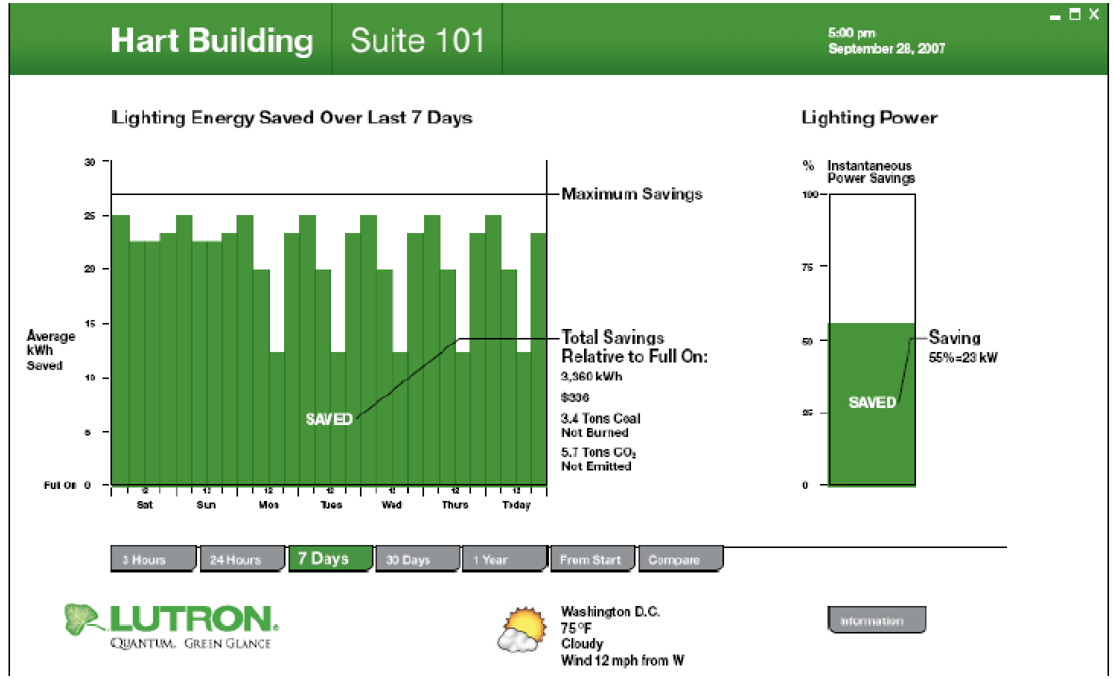
- **It's just a light switch – *how hard can it be?***



http://www.lutron.com/Products/WholeHomeSystems/Homeworksqs/Pages/Components.aspx

Energy Savings





[Lutron]

Real time light dimming

- Occupancy Sensors
- Daylight Sensors
- Wireless & wired networking
- Provide constant illumination across building
- Save by avoiding 100%-on

Where Are We Now?

- **Where we've been:**
  - Interrupts
  - Context switching and response time analysis
  - Concurrency
  - Scheduling

- **Where we're going today:**
  - RTOS and other related topics
  - Priority inversion
  - Why software quality matters (safety & security)

- **Where we're going next:**
  - Intro to embedded networks
  - System booting, control, safety
  - Test #2 on Wednesday April 20th, 2016

Preview

- **Priority Inversion**
  - Combining priorities with a mutex leads to complications
  - Priority inheritance & priority ceiling as solutions

- **RTOS overview**
  - What to look for in an RTOS
  - Market trends in RTOS
  - General embedded design trends

Remember the Major Scheduling Assumptions?

- **Five assumptions throughout this lecture**
  1. **Tasks $\{T_i\}$ are perfectly periodic**
  2. **B=0**
  3. **$P_i = D_i$**
  4. **Worst case C_i**
  5. **Context switching is free**

Overcoming Assumptions

- **WHAT IF:**

1. Tasks $\{T_i\}$ are NOT periodic
   - Use Sporadic techniques

2. Tasks are NOT completely independent
   - Worry about dependencies
     (lets talk about this one)

3. Deadline NOT = period
   - Use Deadline monotonic

4. Worst case computation time c_i isn't known
   - Use worst case computation time, if known
   - Build or buy a tool to help determine Worst Case Execution Time (WCET)
   - Turn off caches and otherwise reduce variability in execution time

5. Context switching is free (zero cost)
   - Gets messy depending on assumptions
   - Might have to include scheduler as task
   - Almost always need to account for blocking time B

Reminder: Basic Hazards

- **Deadlock**
  - Task A needs resources X and Y
  - Task B needs resources X and Y

  - Task A acquires mutex for resource X
  - Task B acquires mutex for resource Y

  - Task A waits forever to get mutex for resource Y
  - Task B waits forever to get mutex for resource X

- **Livelock**
  - Tasks release resources when they fail to acquire both X and Y, but… just keep deadlocking again and again

- **We're not to solve these here… desktop OS designers have these too**
  - But there are related priority problems specific to real time embedded systems

Mutex + Priorities Leads To Problems

- **Scenario: Higher priority task waits for release of shared resource**
  - Task L (low prio) acquires resource X via mutex
  - Task H (high prio) wants mutex for resource X and waits for it

- **Simplistic outcome with no remedies to problems (don't do this!)**
  - Task H hogs CPU in an infinite test-and-set loop waiting for resource X
  - Task L never gets CPU time, and never releases resource X

  - Strictly speaking, this is "starvation" rather than "deadlock"

[Renwick04] *modified*

Bounded Priority Inversion

- **An possible approach (BUT, this has problems…)**
  - Task H returns to scheduler every time mutex for resource X is busy
  - Somehow, scheduler knows to run Task L instead
    - If it is a round-robin preemptive scheduler, this will help
    - In prioritized scheduler, task H will have to reschedule itself for later
      - » Can get fancy with mutex release re-activating waiting tasks, whatever ….
  - Priority inversion is bounded – Task L will eventually release Mutex
    - And, if we keep critical regions short, this blocking time B won't be too bad

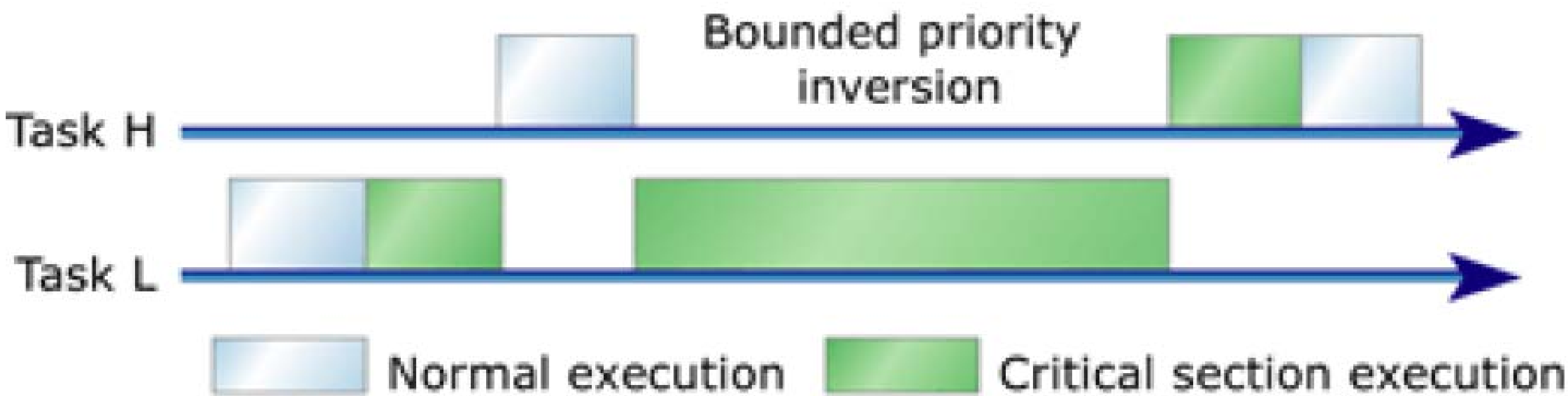


Figure 1: Bounded priority inversion

[Renwick04]

Unbounded Priority Inversion

- **But, simply having Task H relinquish the CPU isn't enough**
  - Task L acquires mutex X
  - Task H sees mutex X is busy, and goes to sleep for a while; Task L resumes
  - Task M preempts task L, and runs for a long time
  - Now task H is waiting for task M ➔ Priority Inversion
    - Task H is *effectively* running at the priority of task L because of this inversion

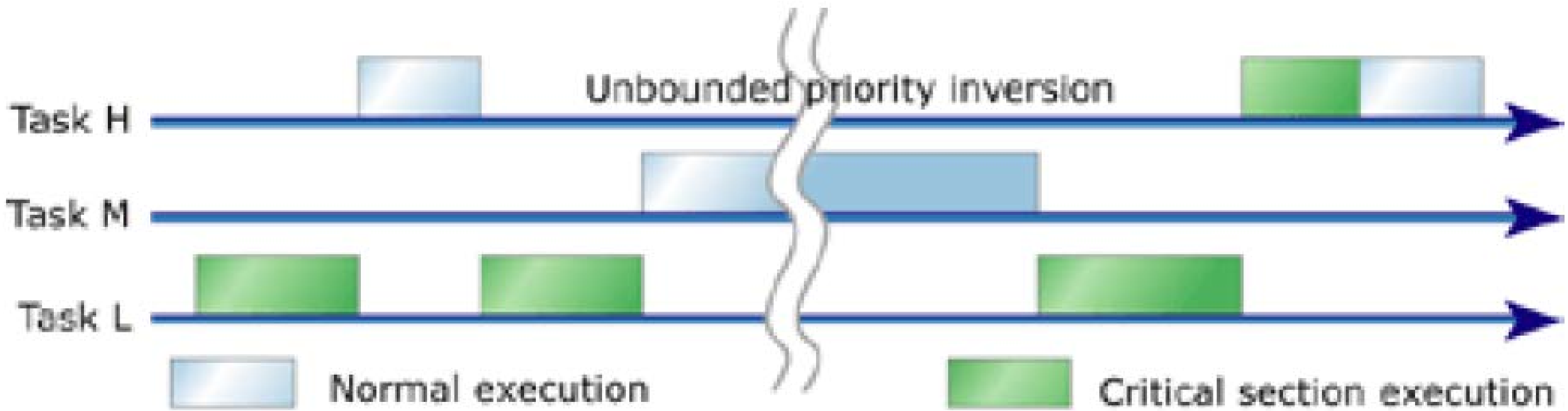


Figure 2: Unbounded priority inversion

[Renwick04]

Solution: Priority Inheritance

- **When task H finds a lock occupied:**
  - It elevates task L to at least as high a priority as task H
  - Task L runs until it releases the lock, but with priority of at least H
  - Task L is demoted back to its normal priority
  - Task H gets its lock as fast as possible; lock release by L ran at prio H
- **Idea: since mutex is delaying task H, free mutex as fast as you can**
  - Without suspending tasks having higher priority than H!
  - For previous slide picture, L would execute with higher prio than M

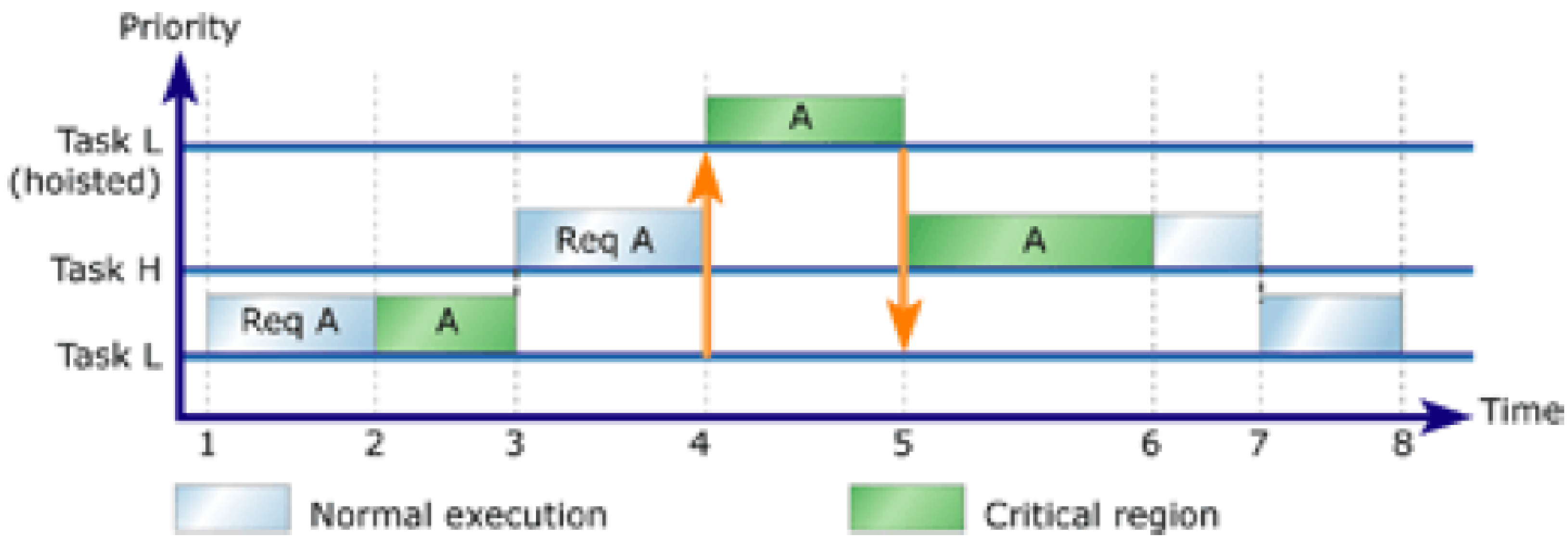


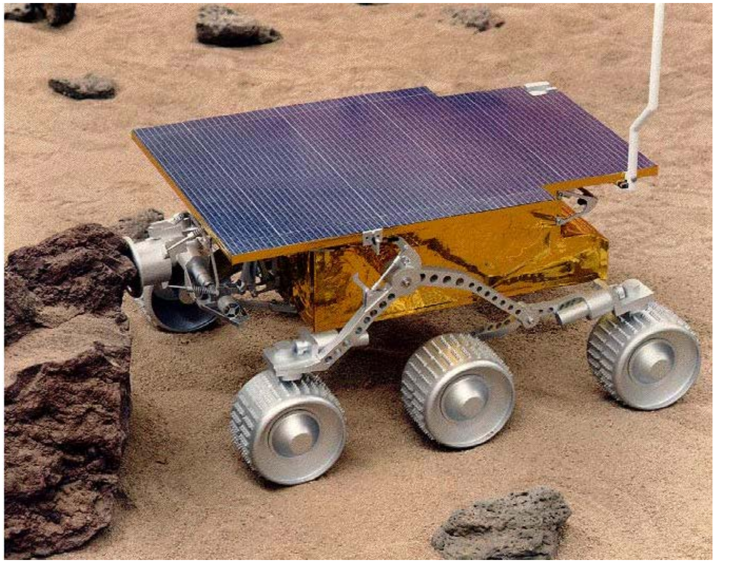
Figure 5: Simple priority inheritance

[Renwick04]

Priority Inheritance Pro/Con

- **Pro: it avoids many deadlocks and starvation scenarios!**
  - Only elevates priority when needed (only when high prio task wants mutex)
- **Run-time scheduling cost is perhaps neutral**
  - Task H burns up extra CPU time to run Task L at its priority
  - Blocking time B costs per the scheduling math are:
    - L runs at prio H, which effectively increases H's CPU usage
    - But, H would be "charged" with blocking time B regardless, so no big loss
- **Con: complexity can be high**
  - Almost-static priorities, not fully static
    - But, only changes when mutex encountered, not on every scheduling cycle
  - Nested priority elevations can be tricky to unwind as tasks complete
  - Multi-resource implementations are even trickier
- **If you can avoid need for a mutex, that helps a lot**
  - But sometimes you need a mutex; then you need priority inheritance too!

Mars Pathfinder Incident (Sojourner Rover)



- **July 4, 1997 – Pathfinder lands on Mars**
  - First US Mars landing since Vikings in 1976
  - First rover to land (vs. crash) on Mars
  - Uses VxWorks RTOS
- **But, a few days later…**
  - Multiple system resets occur
    - Watchdog timer saves the day!
    - System reset to safe state instead of unrecoverable crash
  - Reproduced on ground; patch uploaded to fix it
    - Developers didn't have Priority Inheritance turned on!
    - Scenario pretty much identical to H/M/L picture a couple slides back
    - Rough cause: "The data bus task executes very frequently and is time-critical -- we shouldn't spend the extra time in it to perform priority inheritance" [Jones07]

RTOS Selection

- **RTOS = Real Time Operating System**
  - An OS specifically intended to support real time scheduling
    - Usually, this means ability to meet deadlines
  - Can support any scheduling approach, but often is preemptive & prioritized
  - Usually designed to have low blocking time B

- **Why isn’t plain Windows an RTOS?**
  - Example – Win NT (in all fairness, it was never supposed to be an RTOS!)
  - 31 priority levels (not enough if you need one per task and one per resource)
    - Round robin execution to all threads at same priority
    - Probably want 256 or more for an RTOS
  - Didn’t support priority inheritance
  - Long blocking times on simple system calls (e.g., 670 usec+ on WinNT)
  - Device drivers aren’t designed to guarantee minimum blocking time
  - Virtual memory is assumed active (swap to disk is a timing problem!)
  - It’s expensive for mass market products at $186+ per license
  - Source: [http://www.dedicated-systems.com/magazine/97q2/winntasrtos.htm]

So What Do You Need In An RTOS?

Source: [Hawley03] Selecting a Real-Time Operating System, Embedded.com

- **Build vs. buy**
  - Don’t build it if you can buy it (“free” = “buy” for right now)
  - More on this later

- **Footprint**
  - How much memory does the RTOS take?
  - Tasker can be very small, but there is more to an RTOS than that
  - Libraries
    - If you use one math function, does linker drag in all math functions?
    - Or can linker just link functions you actually use?
  - Feature subsetting
    - Can you get RTOS to include only features you need to minimize footprint?

RTOS Features – 2

- **Performance**
  - Real Time != Real Fast … but Real Slow is no fun either
  - Blocking time B is key!
  - What is task switching time?
  - What is maximum blocking time within supplied code?
  - Does it get things such as device driver blocking right?
  - Boot time – does your customer want to wait 5 minutes to boot a flashlight?
  - Make sure you compare apples to apples – comparable CPUs and clock speeds
- **Add-ons**
  - Does it come with support for web connectivity?
  - Does it support domain-specific needs (e.g., MISRA C compiler for automotive?)
- **Tool support – comes with or supports other tools you need**
  - Compilers
  - Debuggers
  - Simulators, ICE, etc.

RTOS – 3

- **Standards support**
  - Windows?
  - POSIX (“Unix”)?
    - Watch out for subsetting! Might support some functions but not even a command prompt
    - QNX and RT-Linux have a command prompt
    - VxWorks is Posix compliant, but doesn’t support “fork”
  - Safety certification, if required (domain specific)
    - This is becoming more common for major players

- **Technical support**
  - Will they answer the phone at 3 AM if your biggest customer is down?
  - Training
  - Examples

- **Source code**
  - Some will provide you with source code outright so you can self-support
  - Some will put source code in escrow in case they go out of business

RTOS – 4

RTOS features you need

- Mutex / semaphore
  - Priority inheritance or priority ceiling
- Scheduling support: RMS (big RTOS) or static multi-rate (medium RTOS) or single-rate cyclic exec (small RTOS)
- Processes (big RTOS) or just tasks (medium/small RTOS)
- Memory protection and memory management

Licensing – how much does it cost?

- Bulk license – flat fee for unlimited copies
- Per-copy license – usually “runtime only” license is “cheap”
  - Development license may be expensive
- Free software isn’t really free
  - Support comes from somewhere – internal or 3rd party

Reputation

- Will the company be there for you?
  - Will it still be there tomorrow (is it one guy in a garage?)
- Does its software actually work?

THREADX

ThreadX is Field Proven!

With over a billion deployments, ThreadX is industry proven and ready for your most demanding requirements.

Small Footprint

ThreadX is implemented as a C library. Only the features used by the application are brought into the final image. The minimal footprint of ThreadX is under 2KB on Microcontrollers.

- Minimal Kernel Size: Under 2K bytes
- Queue Services: 900 bytes
- Semaphore Services: 450 bytes
- Mutex Services: 1200 bytes
- Block Memory Services: 550 bytes
- Minimal RAM requirement: 500 bytes
- Minimal ROM requirement: 2K bytes

* Measurements based on ThreadX V5.1, configured for minimal size

Fast Response

ThreadX helps your application respond to external events faster than ever before. ThreadX is also deterministic. A high priority thread starts responding to an external event on the order of the time it takes to perform a highly optimized ThreadX context switch.

- Boot Time: 300 cycles
- Context Switch Time: 20 cycles
- Semaphore Get: 30 cycles

* timing based on ThreadX V5.1, configured for maximum performance and minimal size.

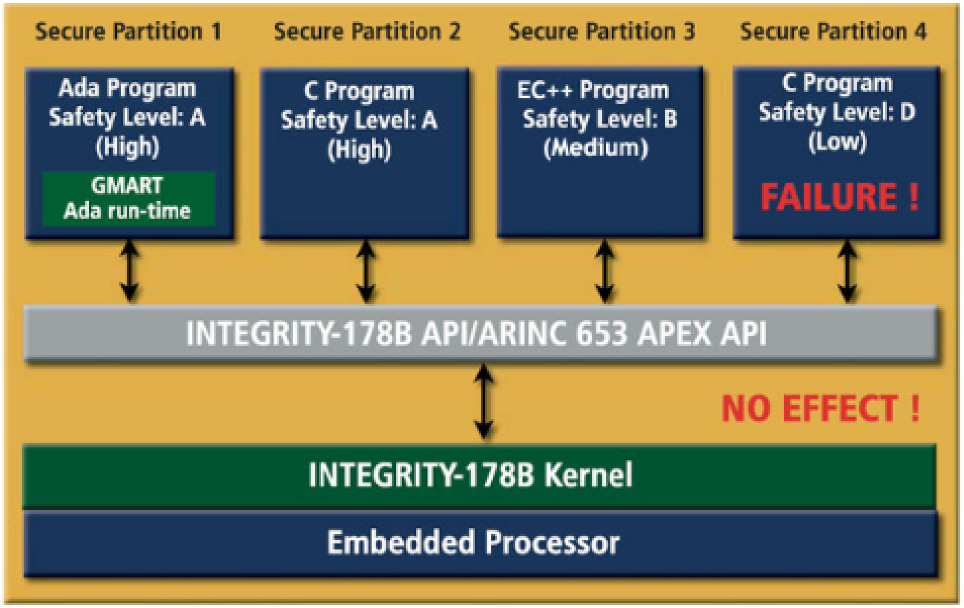
Instant On

ThreadX requires as little as 300 cycles to initialize and start scheduling application threads. This is hugely important for consumer and medical devices that simply can't afford a long boot time.

Safety Critical Products: INTEGRITY®-178B RTOS

» Download INTEGRITY®-178B RTOS Datasheet (PDF)

The INTEGRITY®-178B operating system is the most secure operating system in the world to have been certified by the NSA-managed NIAP lab to EAL6+ High Robustness. No other commercial operating system has attained this level of security. No other commercial operating system has entered into an evaluation at EAL6+ High Robustness.



Related Articles

- INTEGRITY Security Overview
- The Gold Standard for Operating System Security: SKPP
- Secure Separation Architecture

INTEGRITY-178B

Safety critical runtime options

- Securely partitioned real-time operating system
- Protection in both the time and space domains
- Resource/IO protection
- ARINC-653-1 compliant APEX interface
- Support for multiple levels of safety criticality
- Support for Ada 95, C, and Embedded C++
- Support for Rate Monotonic Analysis (RMA)
- DO-178B Level A certification package

Adopting A Free RTOS Can Be Tricky

- **Example: Adopt a "free" RTOS**
  - Assume it's "free" (source code available), popular, and pretty good
  - Local engineers learn it and make some tweaks
  - Now you have your own local code base and some expert engineers

- **Is it really "free?"**
  - Engineers invested time learning it, but they'd do that for any RTOS
  - Local code base has to be maintained – this is *not* free
    - If bug fixes are published for initial code, have to adapt them to your version
    - Maybe no big deal if a small fraction of engineer's time
    - Engineer was good at RTOS design already, so it's a "free" skill

- **But what is the organizational cost?**
  - If that engineer leaves, you need to hire someone else with RTOS skills!
    - And convince them to move to whatever little town that company is in
  - May or may not be able to benefit from later add-on tools
    - May or may not be able to migrate to later major upgrades

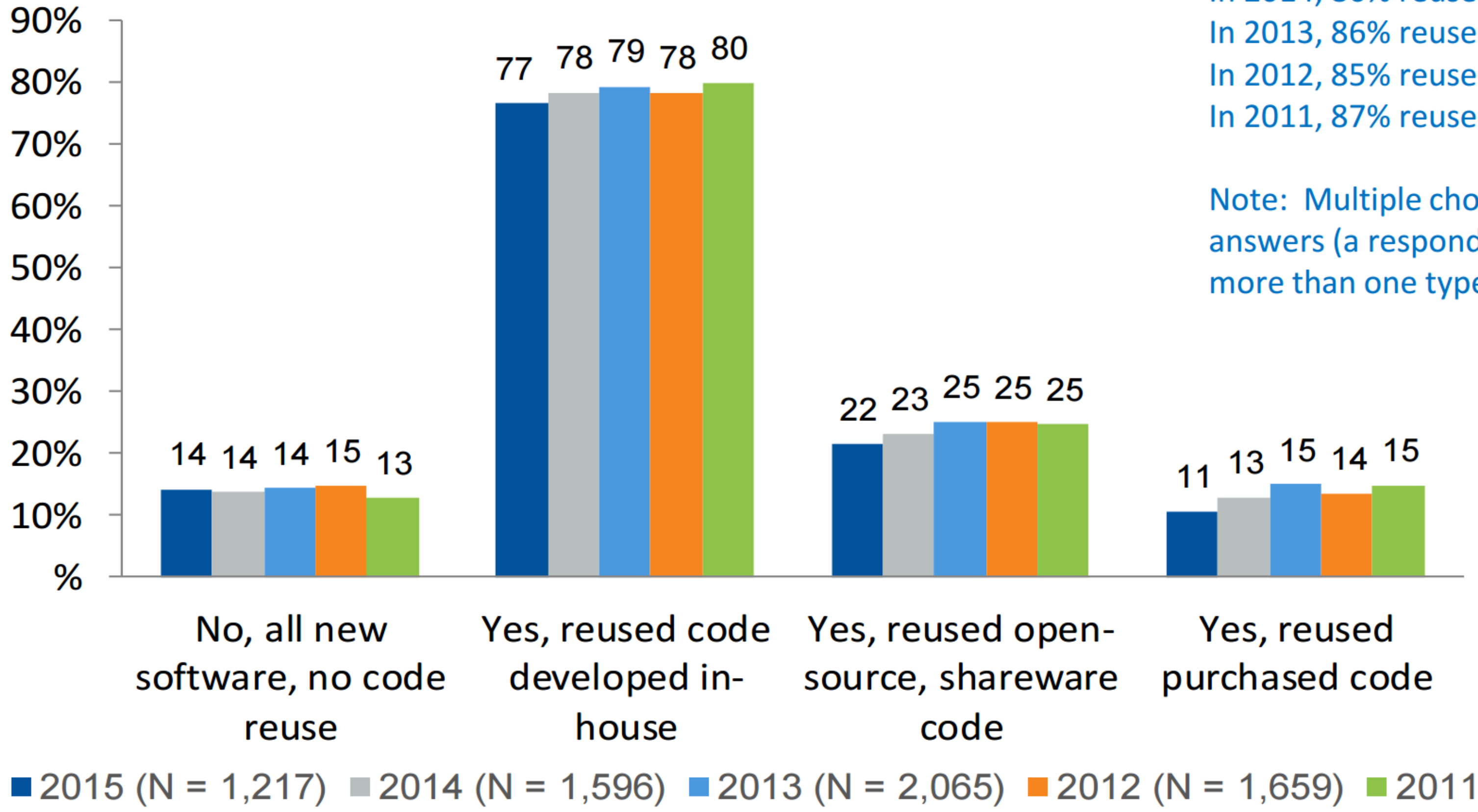
Industry Concern: Open Source "Poisoning"

- **Industry projects have to be *very* careful about open source**
  - Some open source licenses are no big deal (probably Berkeley)
  - Some open source licenses are *toxic* (especially GPL)
    - GPL library code and using compilers is OK; rest can be a problem
  - Some are in between
- **Common concerns with open source**
  - Requirement to publish source code of "derivative works"
  - Prohibition for fixed-function product "Tivo-ization" prohibited
  - Tracking and publishing copyright attribution (an annoyance)
  - Possibility of being sued for patent infringement by open source code
- **How do you manage the risks?**
  - Use open source tracking tools that sniff out all open source code in a build
  - Have explicit legal department sign-off on every open source component
    - Sometimes you can't use them because the legal issues are too tough
    - And sometimes it's OK … depends upon product & company

Few Projects Are "Clean Sheet of Paper"

2015 UBM Electronics Embedded Markets Study

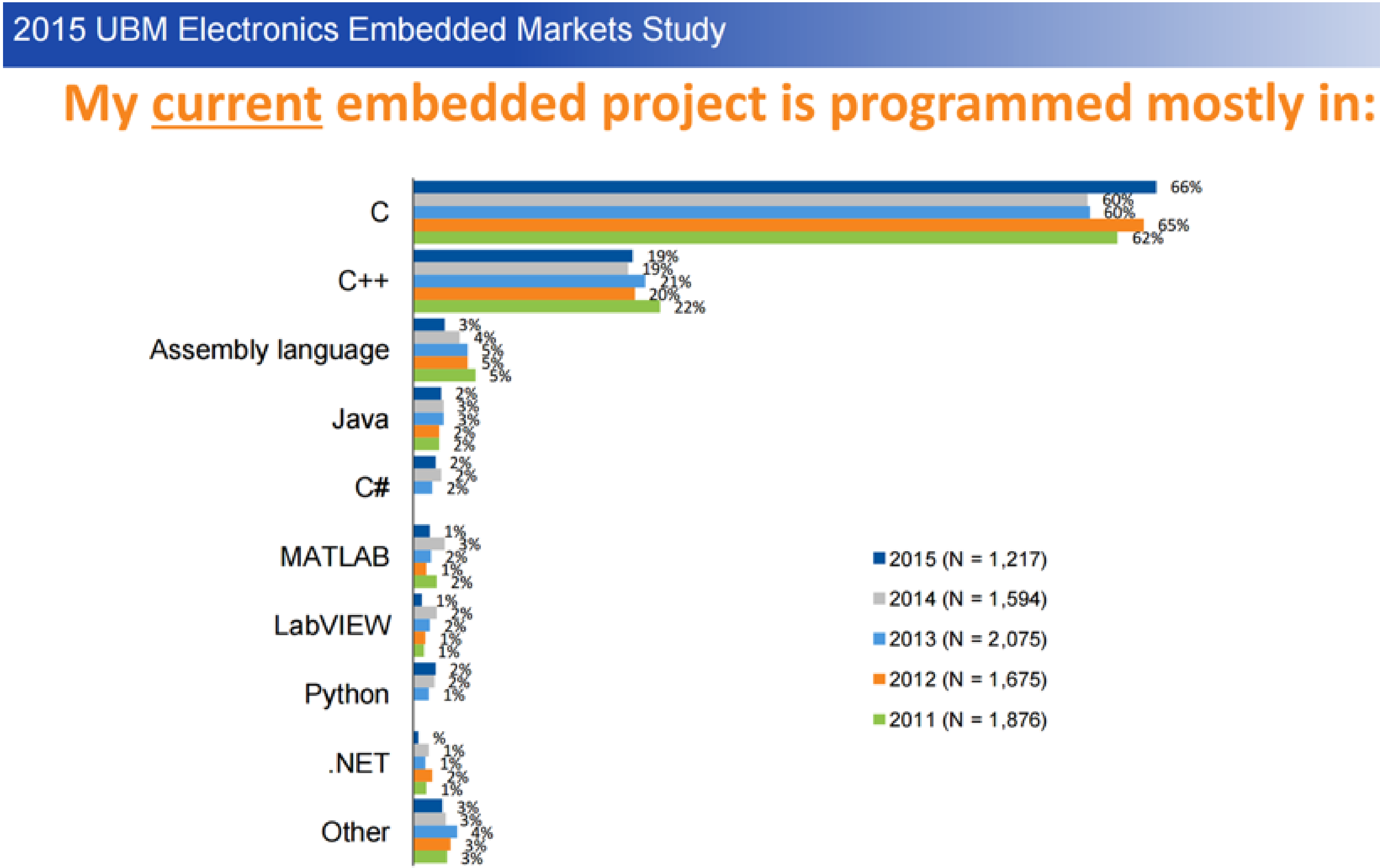
Does your current project reuse code from a previous embedded project?



In 2015, 86% reused code.
In 2014, 86% reused code.
In 2013, 86% reused code.
In 2012, 85% reused code.
In 2011, 87% reused code.

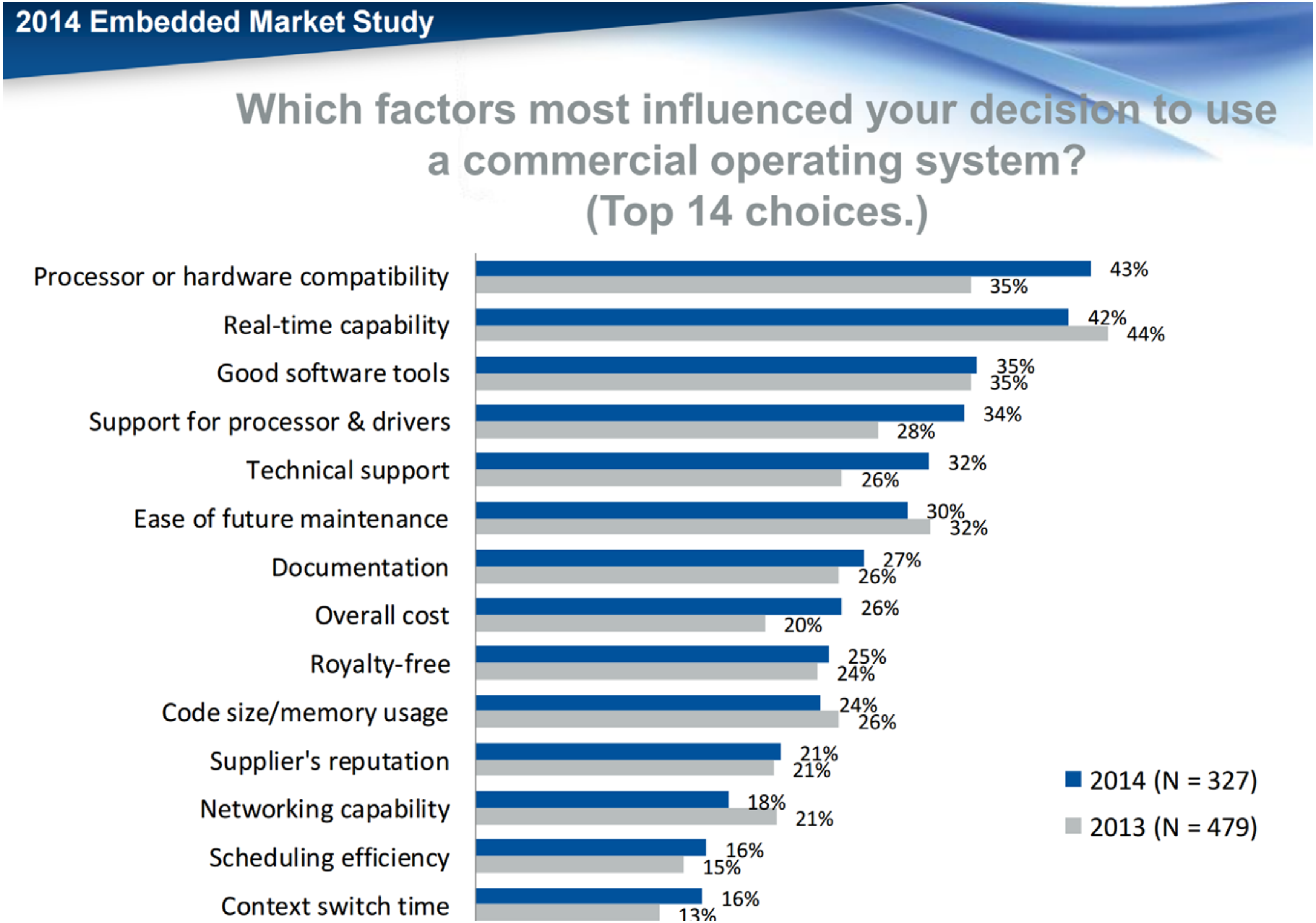
Note: Multiple choice for "Yes" answers (a respondents can select more than one type of reused code).

C & C++ Are Prevalent



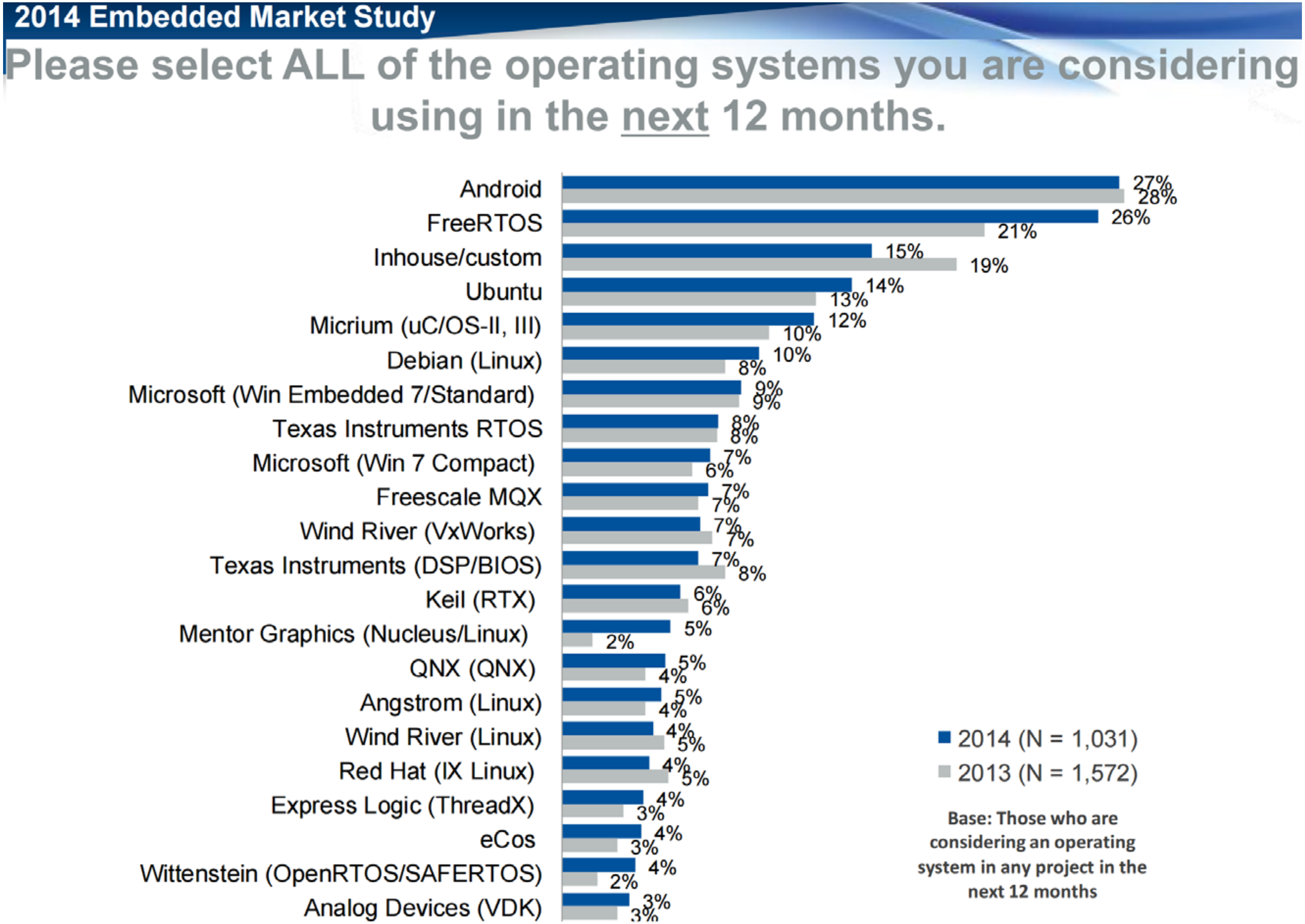
http://webpages.uncc.edu/~jmconrad/ECGR4101-2015-08/Notes/UBM%20Tech%202015%20Presentation%20of%20Embedded%20Markets%20Study%20World%20Day1.pdf

RTOS Selection Factors:



http://bd.eduweb.hhs.nl/es/2014-embedded-market-study-then-now-whats-next.pdf

RTOS Popularity



http://bd.eduweb.hhs.nl/es/2014-embedded-market-study-then-now-whats-next.pdf