

The background of the slide features a blue-tinted image of industrial robotic arms. Overlaid on this image is a network of glowing blue lines and dots, suggesting a digital or data-driven environment. The text is positioned on the left side of the image.

# **VIRTUALIZATION** AND REAL-TIME SYSTEMS FOR WORK LOAD CONSOLIDATION

Core and Visual Computing Group, Intel®

# LEGAL NOTICES AND DISCLAIMERS

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [www.intel.com](http://www.intel.com).

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Any forecasts of goods and services needed for Intel's operations are provided for discussion purposes only. Intel will have no liability to make any purchase in connection with forecasts published in this document.

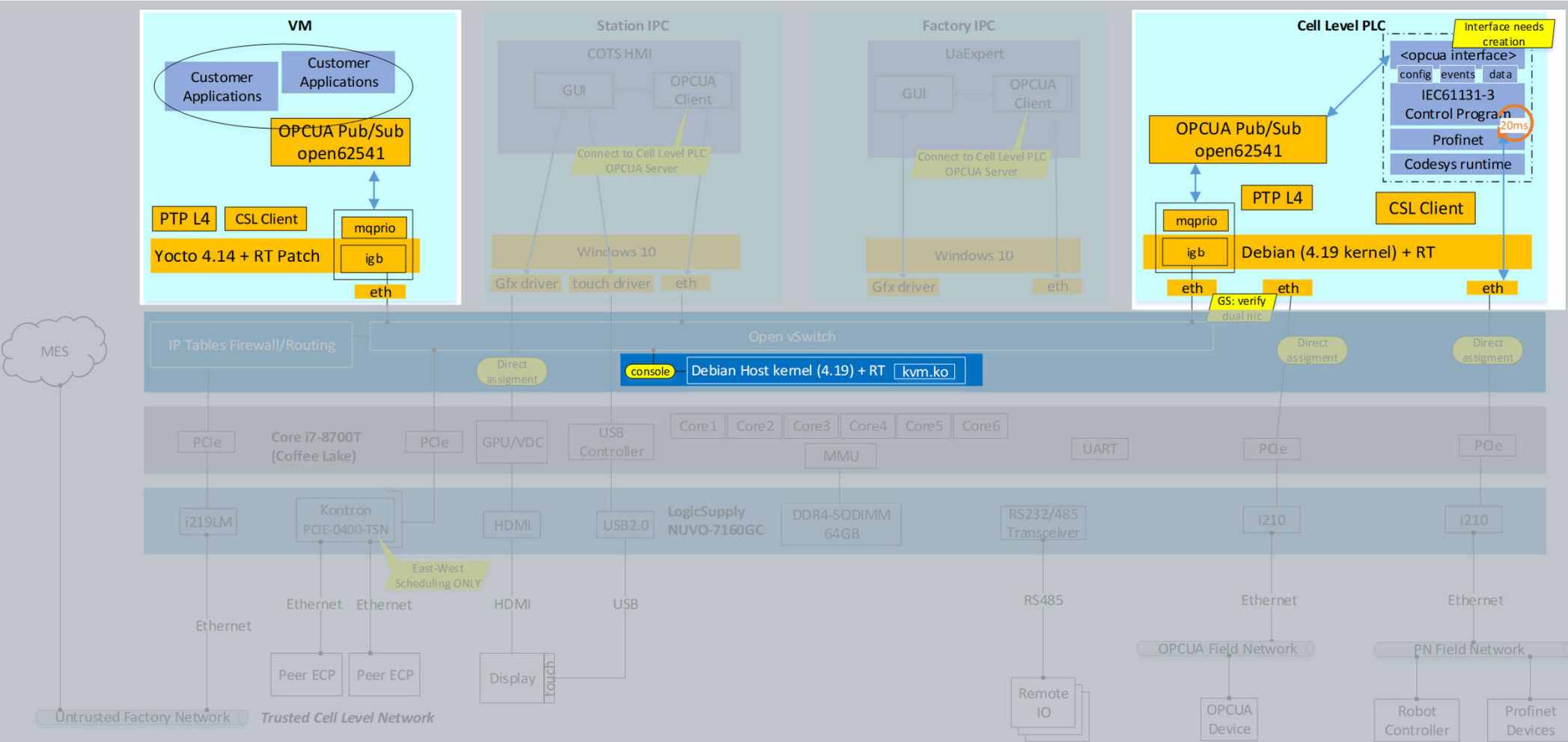
ARDUINO 101 and the ARDUINO infinity logo are trademarks or registered trademarks of Arduino, LLC.

Intel, the Intel logo, Intel Inside, the Intel Inside logo, OpenVINO, Intel Atom, Celeron, Intel Core, and Intel Movidius Myriad 2 are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

Copyright 2018 Intel Corporation.

# Real-Time Systems



How would you define a Real-Time system?





# HARD VS. SOFT REAL-TIME REQUIREMENTS

The primary difference between hard and software real-time systems is the consequences of missing a scheduling deadline.

- **Hard Real-Time:** scheduling deadlines must be met every single time. Missing the deadline means the system has failed, possibly with catastrophic consequences.
  - Robotic assembly line that require a high degree of timing accuracy
  - Software for dropping control rods in a nuclear power plant
- **Soft Real-Time:** can tolerate missing a deadline occasionally, if an average latency is maintained.
  - A system reporting on the current activity of the assembly line will not have catastrophic results if the information is slightly delayed

# PROPERTIES OF AN RTOS

A RTOS is predictable if the time necessary to acknowledge a request of an external event is known in advance. This predictability is called determinism, in the sense that the time is exactly known in advance.

Modern RTOS include in general the following features:

- fast switch context, small size, preemptive scheduling based
- on priorities, multitasking and multithreading, inter-task communication and synchronization mechanisms
- (semaphores, signals, events, shared memory, etc.), real-time timers, etc.

However, RTOS are similar to standard operating systems from a structural point of view, since functional components as interrupt handler, task manager, memory manager, I/O subsystem and inter-task communication are proper of both kind of operating systems.

# REAL-TIME SOFTWARE: THE CONTROL LOOP

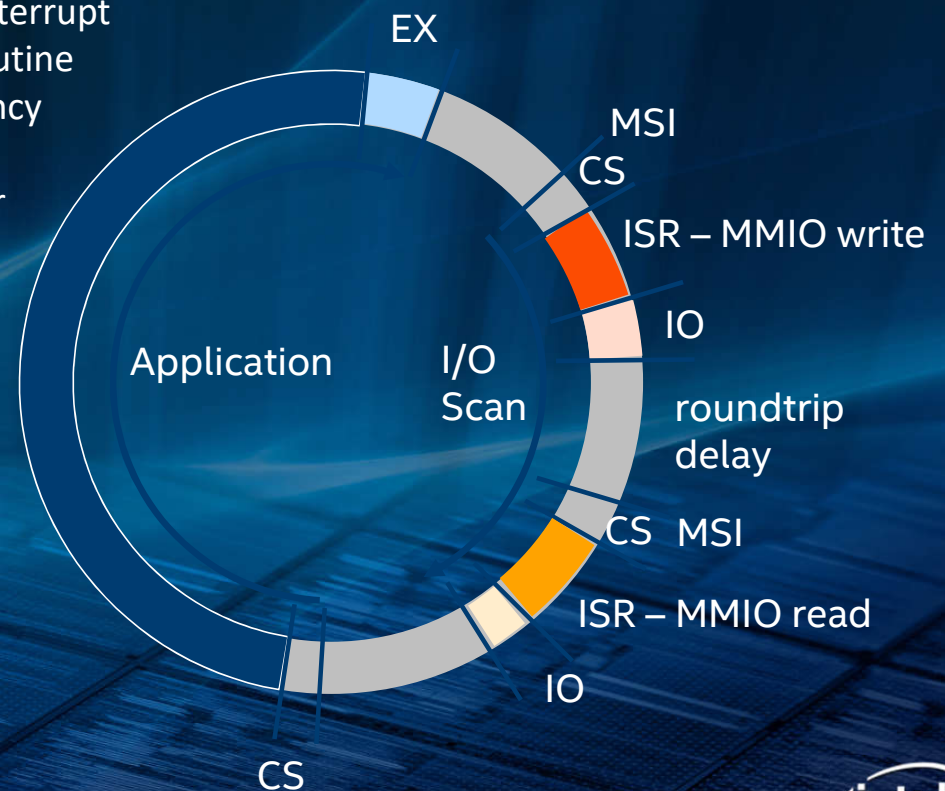
- Control Loop

- Acquire data
- Process
- Actuate

- Event Driven and Periodic

RTLinux applications consist of threads, interrupt handlers, “main” kernel processes, and user processes.

MSI: Message Signaled Interrupt  
ISR: Interrupt Service Routine  
CS: Context Switch Latency  
IO: I/O Jitter  
Ex: Execution Time Jitter





A blue industrial robotic arm is shown in a factory setting, performing a welding task. The robot's gripper is holding a metal piece, and a bright, intense light is visible at the point of contact, with a large spray of orange and yellow sparks erupting from the weld. The background is a blurred industrial environment with various structures and lights.

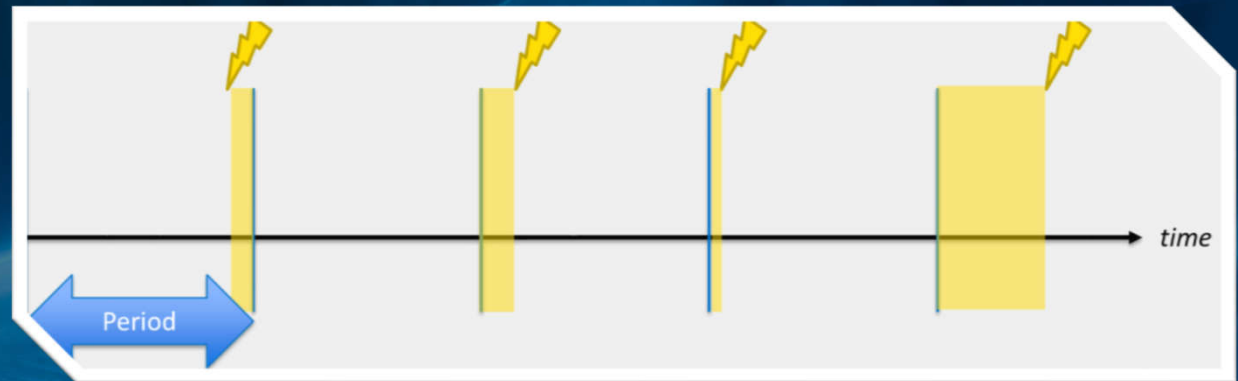
# Control Loops for Industrial Manufacturing

<https://www.intel.com/content/dam/www/public/us/en/documents/case-studies/real-time-iot-tracking-manufacturing-case-study.pdf>



# WHAT IS JITTER, TIMELINESS AND SYNCHRONIZATION?

- Time Constrained System
  - System must guarantee response within it
- Deadline components
  - Period
    - Interval at which events occur
  - Maximum latency
    - Largest time delay acceptable in servicing the event before causing a failure



According to the IEEE, jitter is “the time-related abrupt, spurious variation in the duration of any specified related interval”

# POSIX 1003.1B : A STANDARD FOR REAL-TIME UNIXES

- POSIX : family of IEEE/ISO standards (IEEE committee number 1003) aimed at standardizing services and interfaces offered by Unix-like OS. Eg, Posix 1003.1 (C library), Posix 1003.2 (Shell), Posix 1003.1b (real-time), Posix 1003.1c (threads)
- Benefit of Posix : portability at the source code level
- History : first draft in 1985, Posix 1003.1 in 1990, 1003.1b in 1993 (aka “posix.4”), 1003.1c in 1995, second version of Posix 1003.1b in 1996
- Posix 1003.1b defines “real-time” features : semaphores, shared memory, locking processes in RAM, memory-mapped files, asynchronous I/O, high-res clocks and timers, signals, scheduling

# CONTEXT SWITCHING

- A *context switch* (also sometimes referred to as a *process switch* or a *task switch*) is the switching of the [CPU](#) (central processing unit) from one [process](#) or *thread* to another.
- A *context* is the contents of a CPU's [registers](#) and *program counter* at any point in time. Switching involves saving the contents of the processes' CPU registers and loading the values of the next running process into those registers
- A context switch is sometimes described as the kernel suspending *execution of one process* on the CPU and resuming *execution of some other process* that had previously been suspended.



# THE DISPATCHER AND THE SCHEDULER

The **Dispatcher** carries out the context switch, which includes saving the stack of the outgoing task and the loading for the incoming task, and the CPU handing over to the task that is becoming active.

The Scheduler has the job of selecting the task that will obtain the processor.

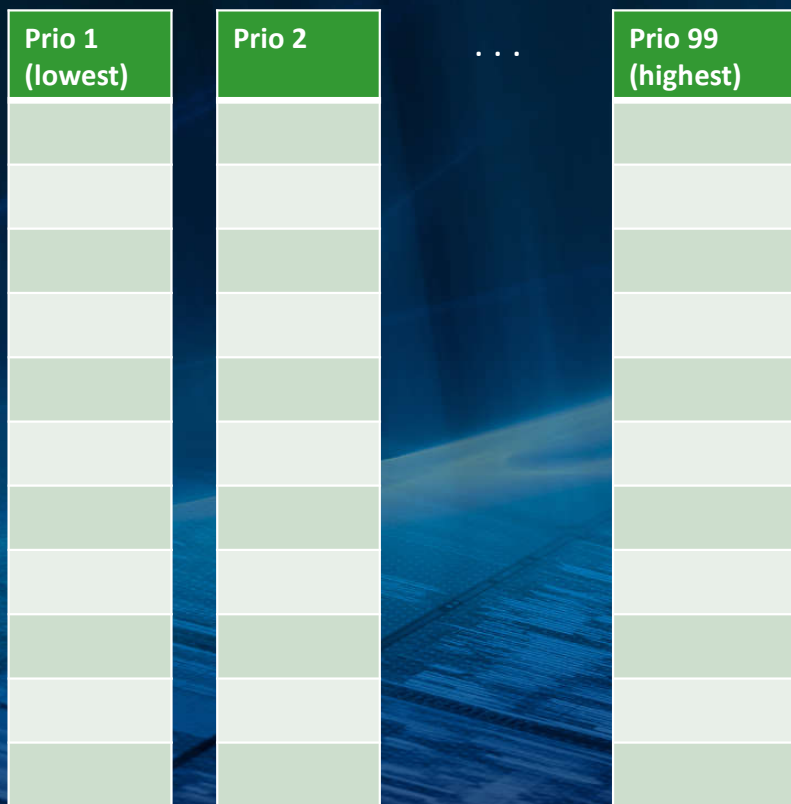
Scheduling algorithms can be either static or dynamic. A static scheduler knows all of the information about the current scheduling state i.e. number of tasks, deadlines, priorities, periods, etc.

A static scheduler solves the scheduling problem a priori, before execution occurs. This is why it is also called clairvoyant.

# PROCESS SCHEDULING


- Cooperative multitasking
  - A process or thread does not stop running until it voluntarily decides to yield
- Preemptive Multitasking
  - Preemption is involuntary. The scheduler forces the suspension of the running process
  - When a process' time slice reaches zero, it is preempted and the scheduler is called to select a new process
  - Three classes of threads for scheduling: `SCHED_FIFO`, `SCHED_RR`, `SCHED_OTHER`

# PRIORITY BASED PREEMPTIVE SCHEDULING (POSIX 1B FIFO)



- **SCHED\_FIFO**
  - FIFO ~ queue
  - 99 priorities (1-99)
  - One queue per priority
- **Tasks run until**
  - Preempted by higher prio
  - Blocked by I/O request
  - Hands-over the processor (sched\_yield)
- **Preempted by higher prio**
  - Will stay at the head of the list to resume as soon as all higher prio tasks are blocked or done
- **Blocked**
  - When becomes runnable will be inserted at the end of the list for its priority



The background of the slide is a high-resolution photograph of the Martian surface, showing reddish-brown soil, dark rocks, and parts of the Mars rover's solar panels and instruments. A large, semi-transparent blue speech bubble is centered on the slide, containing text about a software bug.

The mission was jeopardised by a concurrent software bug in the lander, which had been found in preflight testing but was deemed a glitch and therefore given a low priority as it only occurred in certain unanticipated heavy-load conditions, and the focus was on verifying the entry and landing code.

The problem, which was reproduced and corrected from Earth using a laboratory duplicate thanks to the logging and debugging functionality enabled in the flight software, was due to computer resets caused by priority inversion. [...]

Four resets occurred during the mission, before patching the software on July 21 to enable priority inheritance.

[https://en.wikipedia.org/wiki/Mars\\_Pathfinder](https://en.wikipedia.org/wiki/Mars_Pathfinder)

# Algorithms

## Priority Inversion

Low and High share a resource



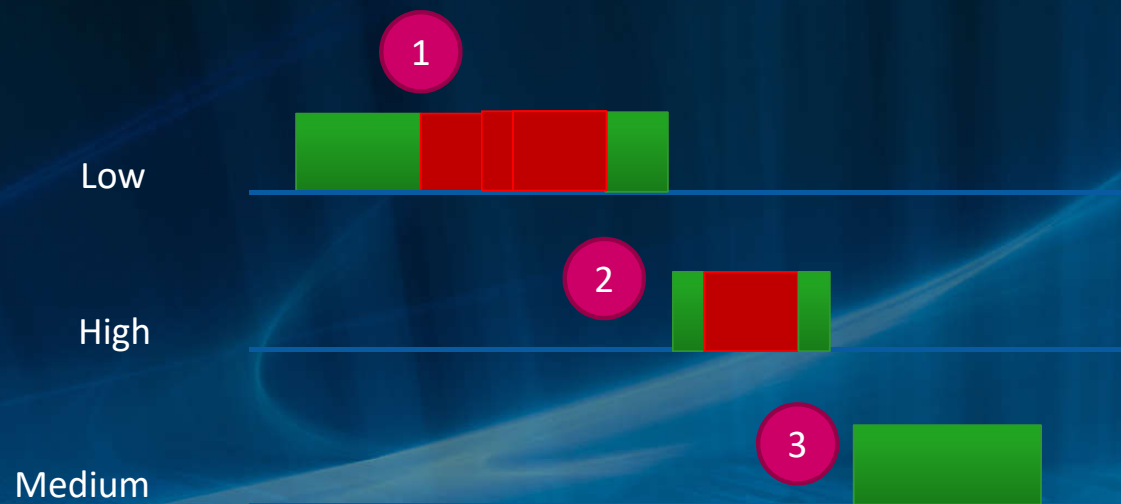
1. Low takes the resource
2. High preempts Low, cannot take the resource and is put to sleep
3. Medium preempts Low and executes completely before High!
4. High cannot take the resource and is put to sleep
5. Low uses and releases the resource, and finishes executing
6. High takes the resource and executes completely

# Algorithms

## Priority Inheritance

Low and High share a resource

1. Low takes the resource, and executes completely
2. High takes the resource and executes completely
3. Medium executes completely



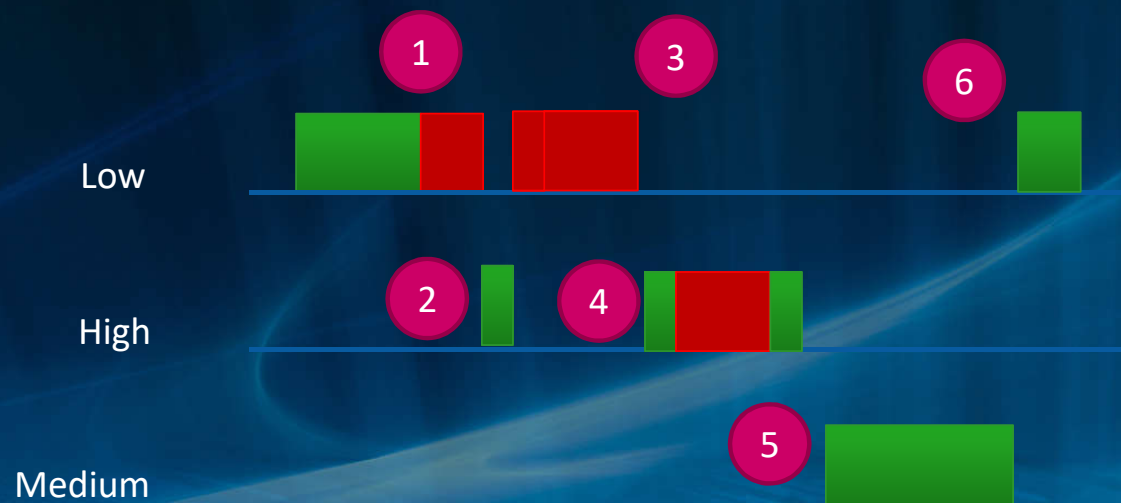


# Algorithms

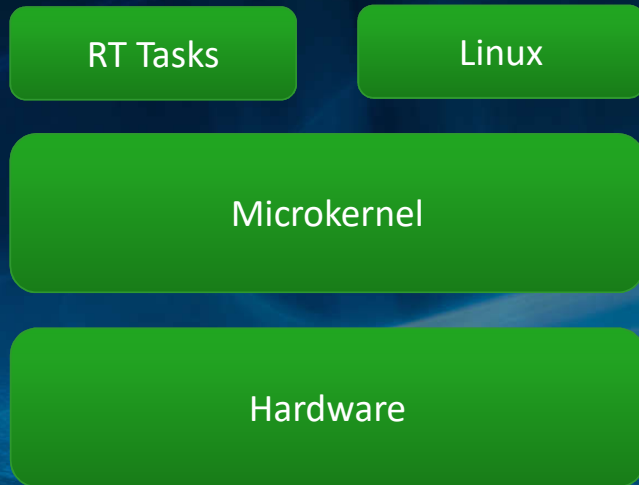
## Priority Inheritance Well Actually

Low and High share a resource

1. Low takes the resource, and executes completely
2. High tries to take the resource but it is taken. Low is given High priority
3. As soon as Low is done with the resource its priority is demoted
4. High takes the resource and executes completely
5. Medium executes completely
6. Low completes its execution



# REAL-TIME INTERRUPT LATENCY



- Typical worst case latency between a hardware interrupt and execution of the handler is 8 microseconds
- RTLinux consists of a real-time kernel called RTCore which runs the Linux operating system as a preemptible thread
- The general rule for RTLinux programmers is to move as much of the code as possible into the Linux context in order to take advantage of the sophisticated environment and tools available there so that in the real-time environment we can focus on getting the timing right.

# POSIX INTERRUPT-DRIVEN CONTROL LOOP

```
#include <stdio.h>
#include <unistd.h>
#include <sched.h>
```

```
unsigned int handler(unsigned int irq, struct
rtl_frame *regs)
{
    CONTROL_DEVICE();
    rtl_hard_enable_irq(irq);
    return 0;
}
```

```
unsigned int handler(unsigned int irq, struct rtl_frame
*regs);
int main(void) {
    if ( rtl_request_irq( IRQ, handler )) {
        printf("failed to get irq %d\n", IRQ);
        return -1;
    }
    rtl_main_wait();
    rtl_free_irq(IRQ);
    return 1;
}
```



# **HIGH** PRECISION TIMERS

# HIGH RESOLUTION TIMERS HARDWARE SUPPORT

```
$ cat /proc/timer_list
```

```
Timer List Version: v0.8
```

```
HRTIMER_MAX_CLOCK_BASES: 4
```

```
now at 199046423526 nsecs
```

```
cpu: 0
```

```
clock 0:
```

```
.base: ffff88017fc11640
```

```
.index: 0
```

```
.resolution: 1 nsecs
```

```
.get_time: ktime_get
```

```
.offset: 0 nsecs
```

```
active timers:
```

```
#0: <ffff88017fc11ae0>, tick_sched_timer, S:01, tick_nohz_restart, swapper/0/0
```

```
# expires at 199047000000-199047000000 nsecs [in 576474 to 576474 nsecs]
```

```
#1: def_rt_bandwidth, sched_rt_period_timer, S:01, enqueue_task_rt, ktimersoftd/0/4
```

```
# expires at 199107000000-199107000000 nsecs [in 60576474 to 60576474 nsecs]
```

```
. . .
```

## High Resolution Timers

- Report 1 ns resolution
- Approximation of accuracy
- Timer values rounded-up to this resolution value

# High Resolution Timers

## APIs allowing us or ns periods

**“POSIX interval timers”**  
Notification on expire via signal event sevp; alternatively  
polling

```
int timer_create(clockid_t clockid,  
                 struct sigevent *sevp,  
                 timer_t *timerid);
```

**itimers**  
Notification on expire via signal event determined by  
parameter “which” (e.g. SIGALRM for ITIMER\_REAL)

```
int setitimer(int which,  
              const struct itimerval *new_value,  
              struct itimerval *old_value);
```

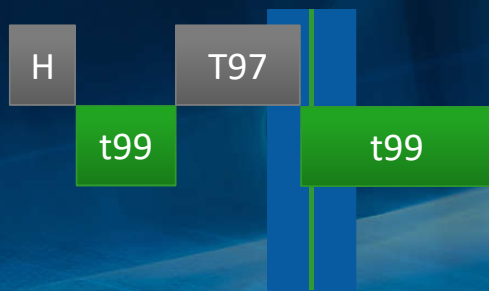
**clock\_nanosleep**  
No signal-based notification

```
int clock_nanosleep(clockid_t clock_id,  
                    int flags,  
                    const struct timespec *request,  
                    struct timespec *remain);
```

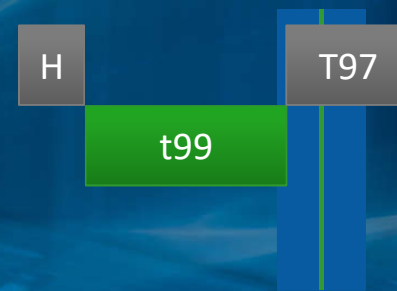
# High Resolution Timers

## Job Scheduling Implications

- All three APIs make use of High Resolution Timers
- However, due to Threaded IRQs...



`timer_create, setitimer`  
Signal delivered by T97 ISR in thread context  
**t99 misses the deadline**



`clock_nanosleep`  
t99 runs right after Hard IRQ handler  
**t99 meets the deadline**

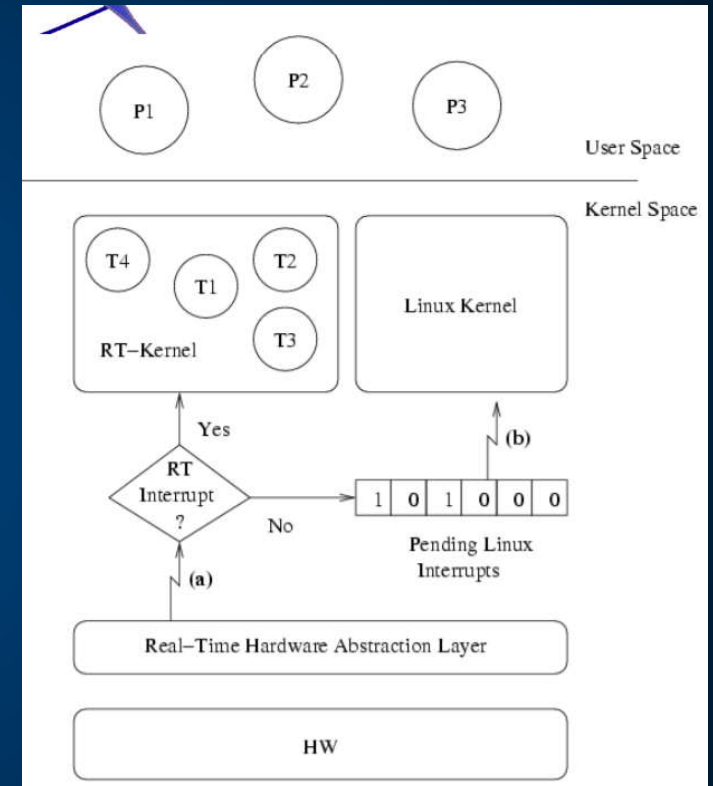


# STRATEGIES FOR SOFT REAL-TIME LINUX

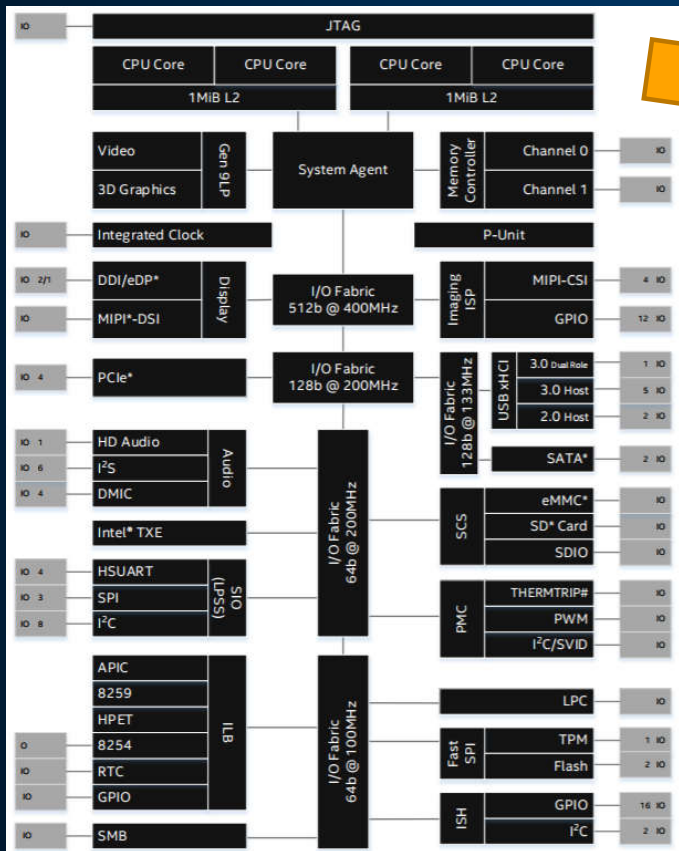
- Scheduling policy
- Preemption Improvement
  - No Forced Preemption:
  - Voluntary Kernel Preemption
  - Pre-emptible Kernel
- Disable support for memory paging
- PREEMPT\_RT Patch – a project whose goal is to implement hard real-time behavior in the Linux kernel. Implements Interrupt

# INTERRUPT ABSTRACTION

- Many processes are I/O bound rather than CPU bound.
- In many applications, only a small part of the process requires real-time performance.
- Run Linux as a low priority process on a small real-time hypervisor
- to run Linux as the lowest priority task (the idle task if you will) under a small real-time kernel. The real-time functions are handled by higher priority tasks running under this kernel. The non real-time stuff, like graphics, file management, and networking is handled by Linux.
- This approach is called “Interrupt Abstraction,” because the real-time kernel takes over interrupt handling from Linux.
- The real-time kernel intercepts hardware interrupts before they get to the Linux kernel.
- When a hardware interrupt occurs, the RT kernel first determines to whom it is directed.

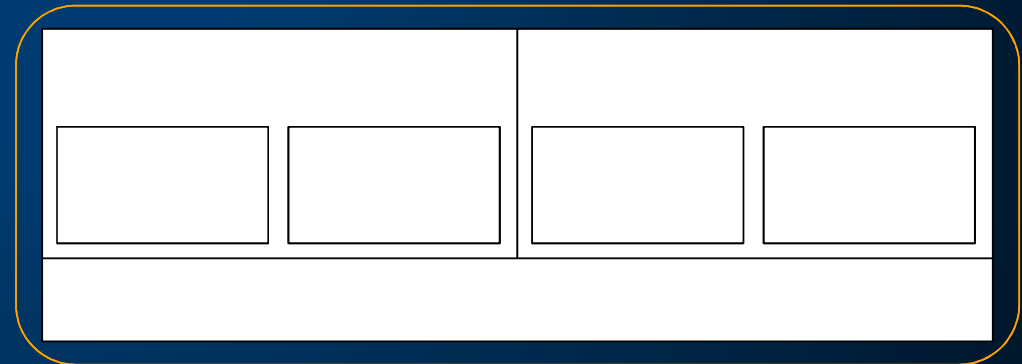


# INTEL® CACHE ALLOCATION TECHNOLOGY



Each module:

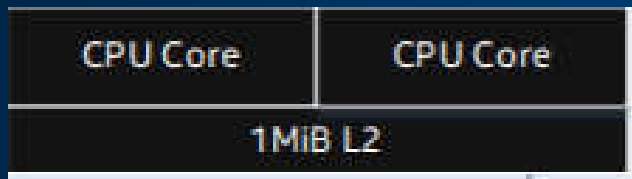
- 2 Cores
- 1 L2 Cache



# CACHE ALLOCATION TECHNOLOGY: “NOISY NEIGHBOR”

Module 0

- Core 0
- Core 1



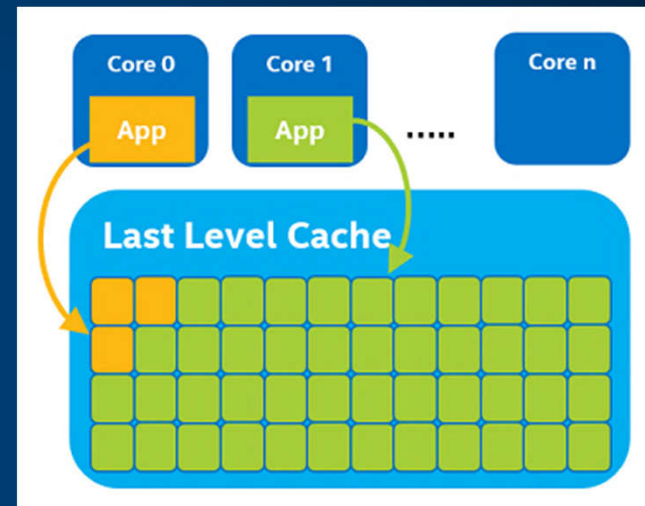
L2 Cache Shared

Most of it used by Core 1

Hosts a “noisy neighbor” process

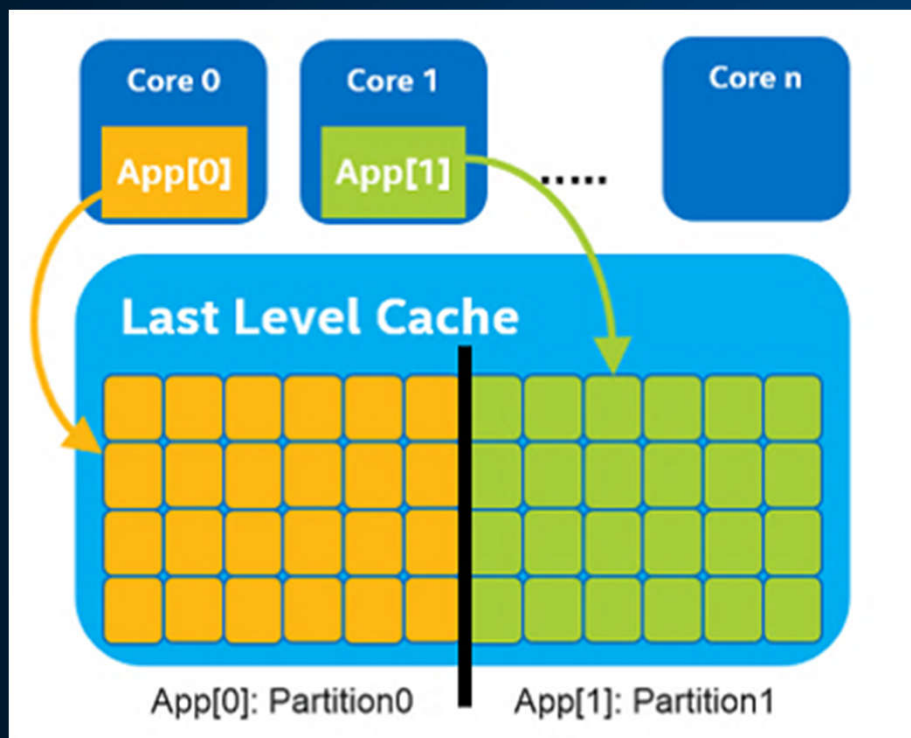
Impact for Core 0

- ↑ Cache misses
- ↑ Latency





# CACHE ALLOCATION TECHNOLOGY: LAST LEVEL CACHE PARTITIONING

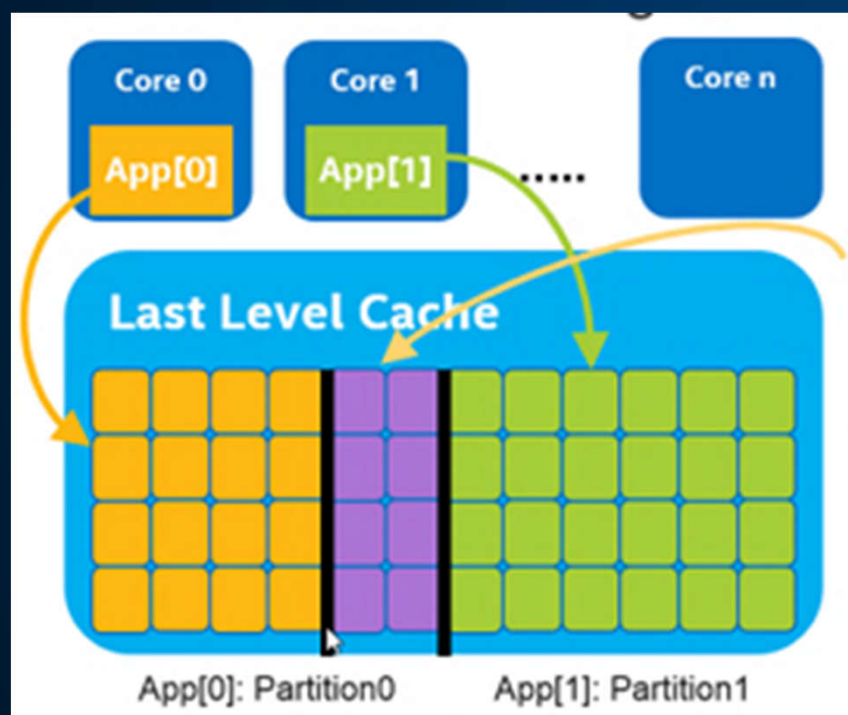


## Cache Allocation Technology

- LLC is partitioned
- Partitions are associated to cores
- Apollo Lake-i LLC
  - 1MB
  - 8-ways
  - 8 partitions
  - 128 KB each

# CACHE ALLOCATION TECHNOLOGY

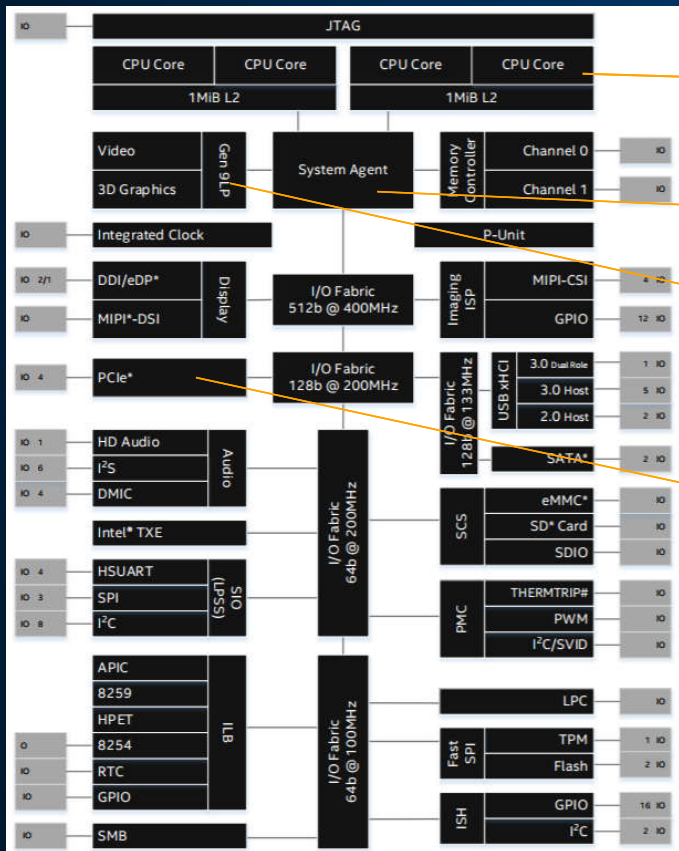
## ALLOCATE CACHE FOR AN APPLICATION (“PSEUDO-LOCKING”)



### “Pseudo-Locking”

- Prepare locked portions
  - Allow access from Core 1 only
- Load data and instructions
  - Run application on Core 1
- Prevent further flushes
  - Remove the portions where data and instructions from the real-time app where stored from Core 0 and Core 1

# POWER MANAGEMENT EVENTS



CPU

System Agent

Graphics

PCI Express

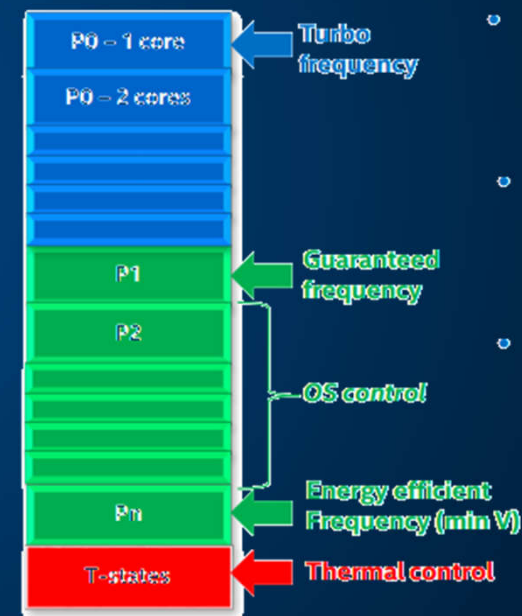
# POWER MANAGEMENT EVENTS: FIRMWARE TUNING

	Active state			
	C0	C1	C3	C6/C7
Core voltage*				
Core clock		off	off	off
PLL			off	off
L1/L2 caches				
LLC/L3 cache				
Wakeup time*	Active			
Idle power*	Active			

<https://www.intel.com/content/dam/doc/white-paper/energy-efficient-platforms-2011-white-paper.pdf>

Idle: Power Saving States

## Active: Frequency Scaling





# POWER MANAGEMENT EVENTS CPU: PROCESSOR STATES (C-STATES)

Optimize Power When Idle

Sample C-States (may vary)

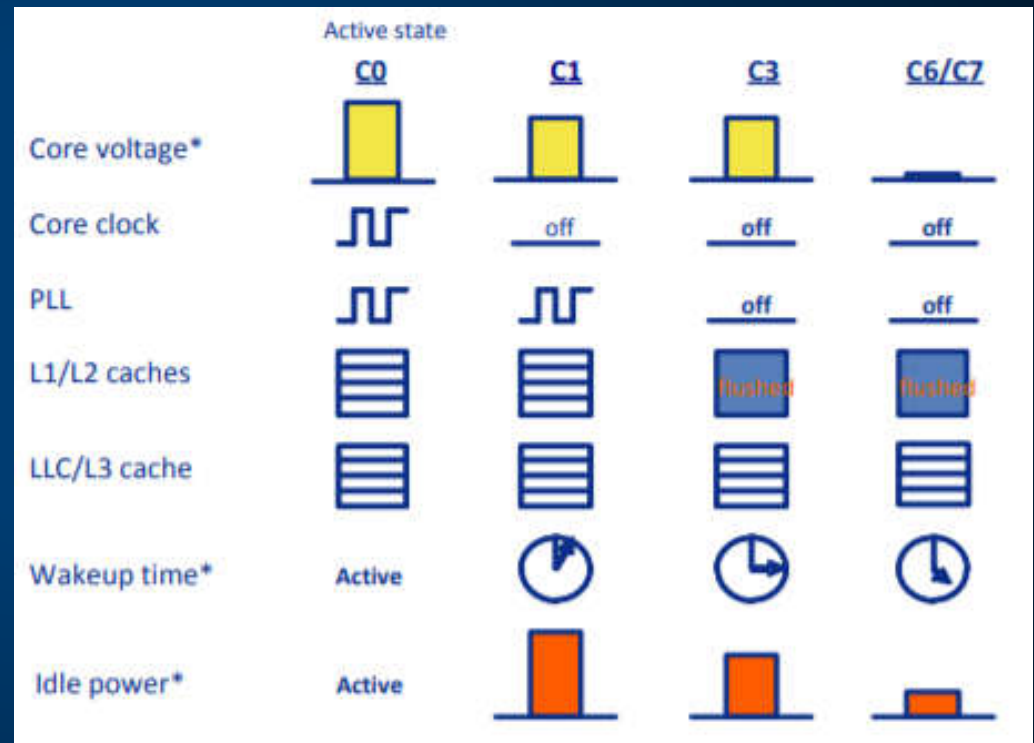
C0: operating

C1: halted

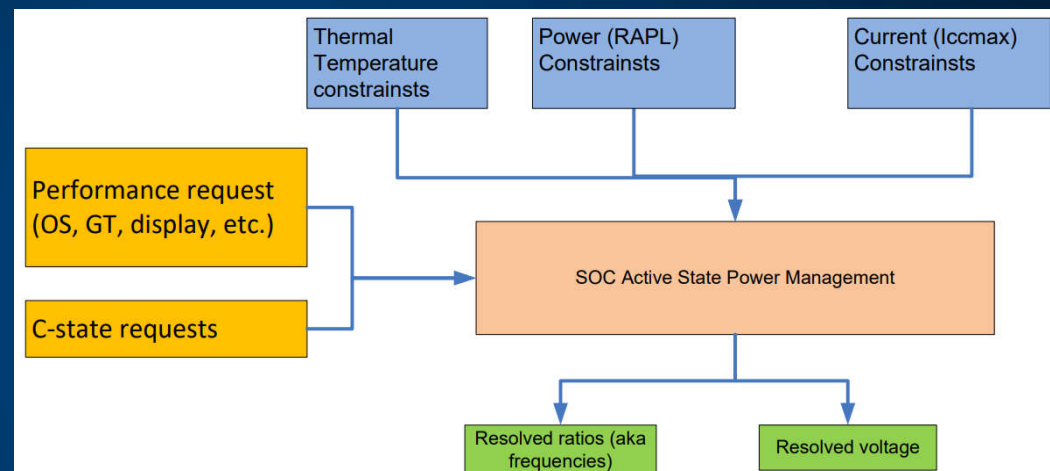
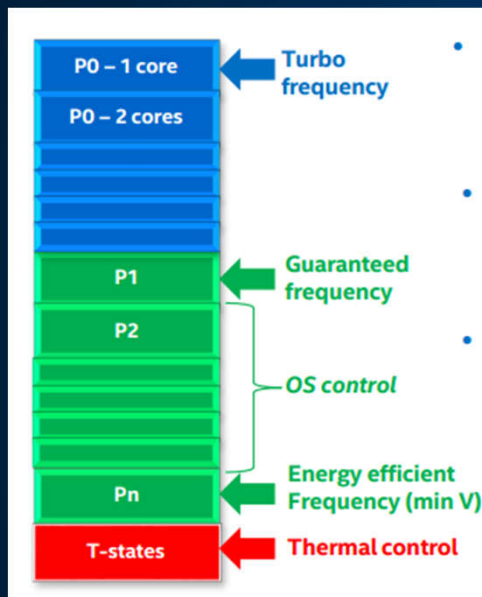
C3: caches flushed

Tuning

Disable in Firmware (BIOS Configuration)\*



# POWER MANAGEMENT EVENTS CPU: DYNAMIC VOLTAGE AND FREQUENCY SCALING



- Tuning
  - Disable in Firmware (BIOS Configuration)

# POWER MANAGEMENT EVENTS PCI EXPRESS (PCIE)

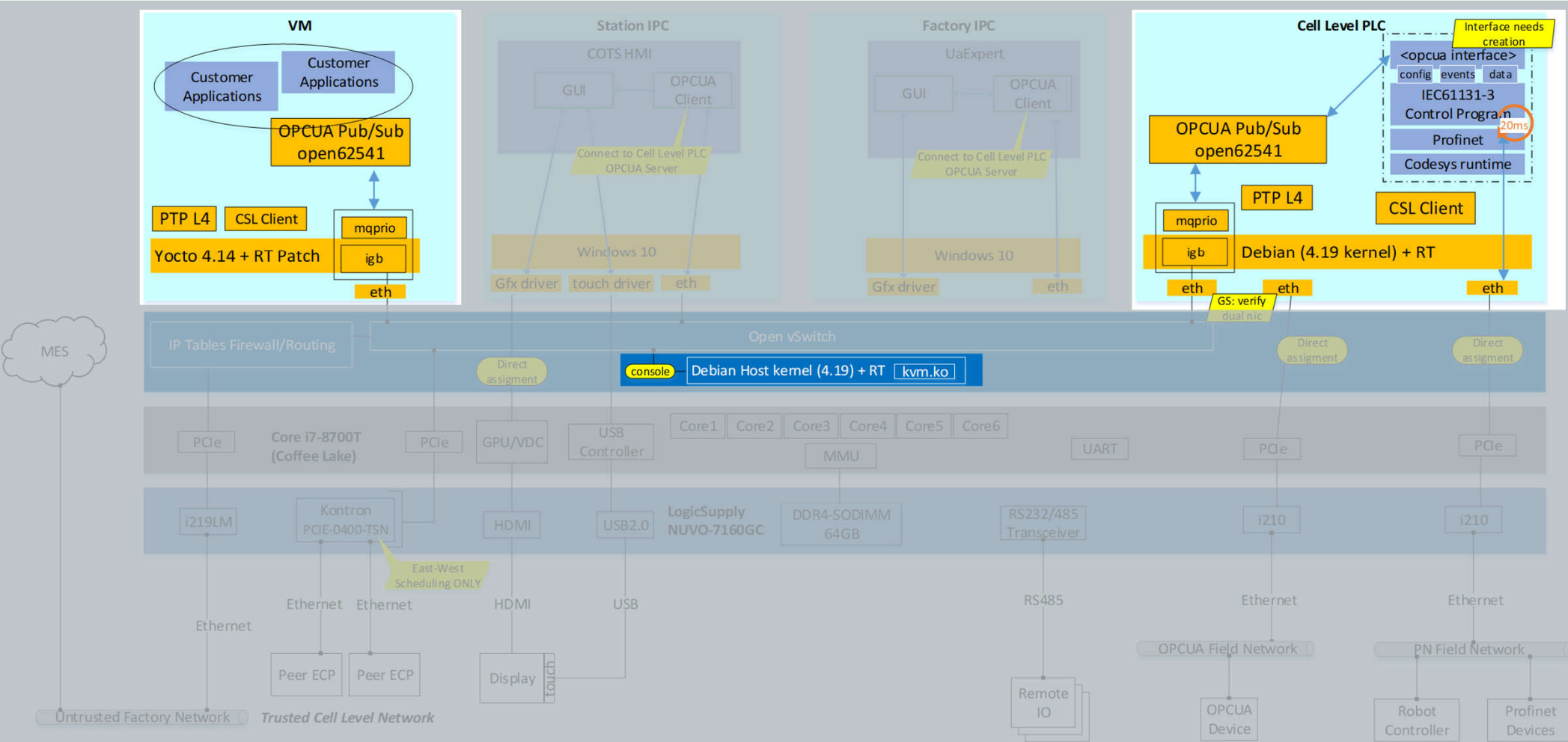
## Active State Power Management (ASPM)

- Triggered when link goes idle
- Manage serial link devices as they become less active over time
- Different sleep states to reduce overall power consumption
- Once link is active it will toggle different power states until it is fully active

## Tuning

- Disable in Firmware (BIOS Configuration)

# Real-Time Systems

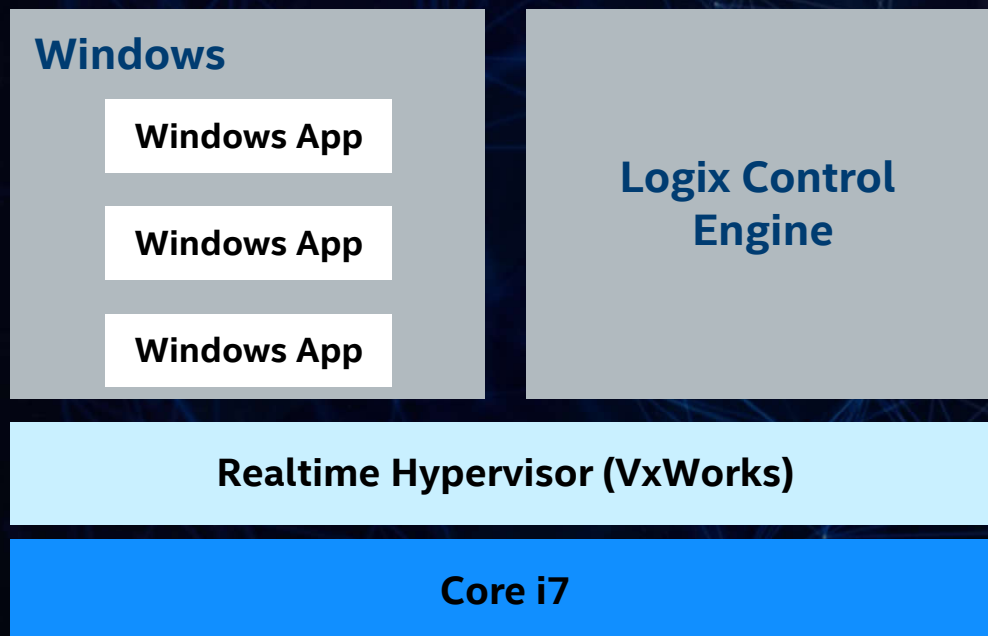




# MULTI-FUNCTION CONTROLLER WITH VXWORKS

## ROCKWELL COMPACT LOGIX 5480 CONTROLLER

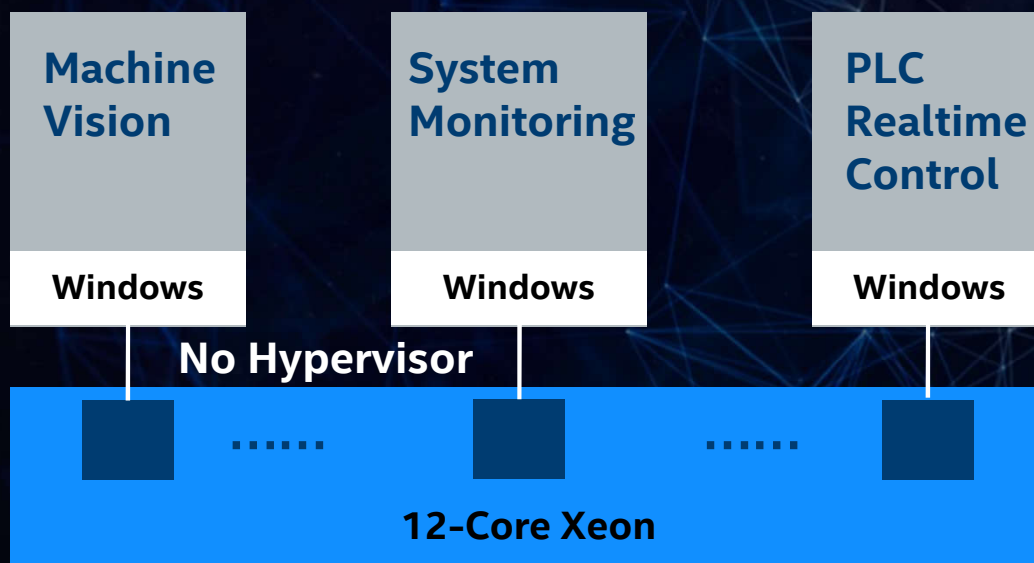
<https://www.youtube.com/watch?v=TovqAiZcCCk>



Opportunity	Problems Solved
Core CPU with real-time Hypervisor	<ul style="list-style-type: none"><li>▪ Ability to run Windows Applications alongside real-time Control Engine</li><li>▪ Leveraging existing PLC install base and extending value/revenue stream of assets</li></ul>

# WORKLOAD CONSOLIDATION WITHOUT VIRTUALIZATION

## BECKHOFF CX2072 CONTROLLER



<https://www.youtube.com/watch?v=miEOxPZ9IIA>

Opportunity	Problems Solved
High-end multi core Xeon CPUs	<ul style="list-style-type: none"><li>▪ Maximum performance by direct assignment of workloads to cores (no Hypervisor)</li><li>▪ Differentiation over competitor products</li><li>▪ Workloads assigned statically from Twincat environment</li></ul>

# INSTALLING RT-TESTS

**pmgtest** – start pairs of threads and measure the latency of the interprocess communication

**cyclictest** – test the latency of switching between processes

**ptsematest** – start pairs of threads and measure the latency of communications through POSIX mutexes.

**svsematest** – start pairs of threads and measure the latency of communications through SYSV mutexes.

**hackbench** – stress test the Linux scheduler using processes communication through sockets or pipes.

**sendme** – sends a signal from a driver to a user and measures the latency

**signaltest** – measures latency of POSIX signals

