

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.

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.

AGENDA

I. Overview

1. Industrial Real-Time Systems
 1. Cyber-Physical Systems
 2. The Control Loop
 3. Cycle Time
 4. Defining Real-Time
 5. Timeliness and Time Sync
2. Timeliness
 1. Deadlines
 2. Concurrency
 3. Jitter
 4. Histograms
 5. Lab: Getting Started

II. Determinism: Concurrency

1. Algorithms
 1. Rate Monotonic Scheduling
 2. Preemption and Priorities
 3. POSIX 1.b Schedulers
 4. Lab: Scheduling
 5. Main Obstacles to Preemption
 6. Priority Inversion
 7. Priority Inheritance
2. Interrupts
 1. Threaded IRQs
 2. Priority Tuning
 3. Interrupt Abstraction
 4. System Management Interrupts
 5. Lab: Shielding and Affinity Tuning
3. Power Management Events
 1. ACPI States
 2. Dynamic Voltage and Frequency Scaling
 3. Graphics
 4. PCI Express
 5. Lab: Power Management Tuning

AGENDA

II. Determinism: Timing

1. High Resolution Timers
 1. Timer APIs
 2. Job Scheduling Implications
 3. Lab: Timer Scheduling BAH5
2. Time Synchronization
 1. Scalable Base Synchronization
 2. Central Platform Time
 3. Precise Time Measurement

II. Determinism: Memory & I/O

1. Real Time Application Tuning
 1. Virtual Memory
 2. Stack and Heap
2. Cache Allocation Technology
 1. Noisy Neighbor
 2. LLC Partitioning
 3. Waymarks
 4. Allocate Cache for an Application
 5. Lab: CAT MSR interface
3. Fabric Virtual Channels
 1. Traffic Classes
 2. Real-Time Upstream Traffic
 3. Real-Time Upstream and Downstream Traffic (Experimental)

Slide 4

BAH5

Review once the content is settled

Blanco Alcaine, Hector, 3/29/2019

How would you define a Real-Time system?



PROPERTIES OF AN RTOS

A RTOS is predictable if the time necessary to acknowledge a request of an external event is known in advance. The end point of this predictability scale is called determinism, in sense that this 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.

WHY LINUX IS NOT A REAL-TIME OS

- Coarse-grained Synchronization
 - kernel system calls are not preemptible
- Paging
 - swapping pages in and out of virtual memory is unbounded
 - Latency to external storage is not predictable
- Fairness
 - scheduler may give the processor to a low priority process that has been waiting a long time
- Request Reordering
- Batching

REAL-TIME SOFTWARE: THE CONTROL LOOP

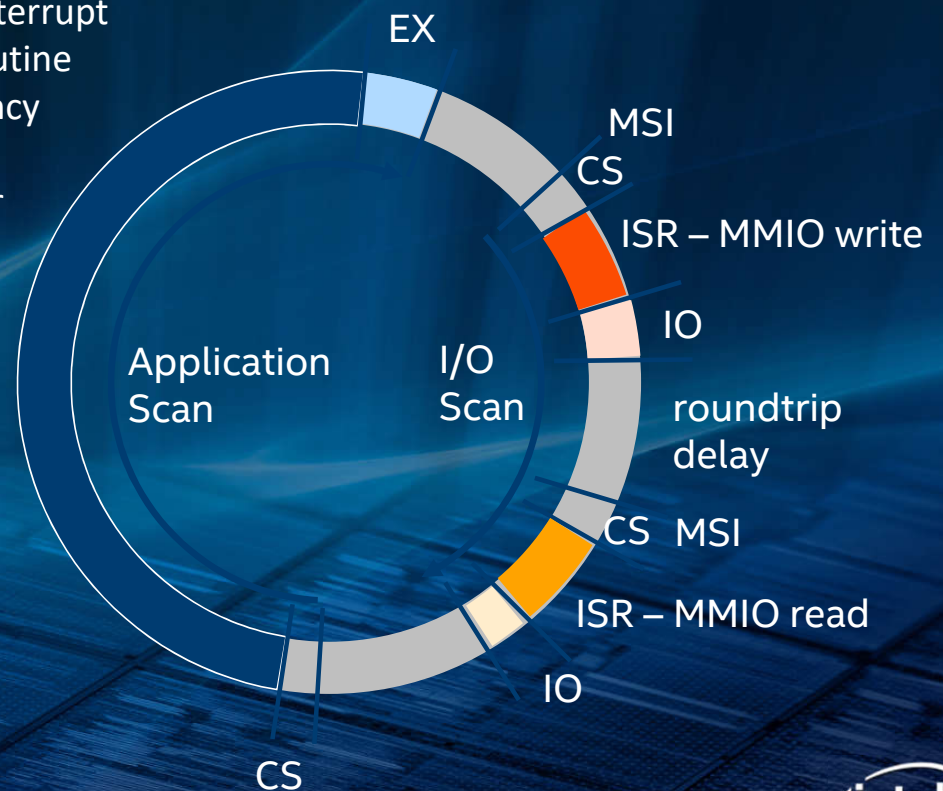
- Control Loop

- 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

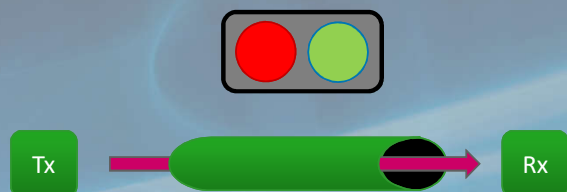


SYNCHRONIZATION AND TIMELINESS

Synchronized

“Occur [...] with coordinated timing”

<https://en.wiktionary.org/wiki/synchronize>

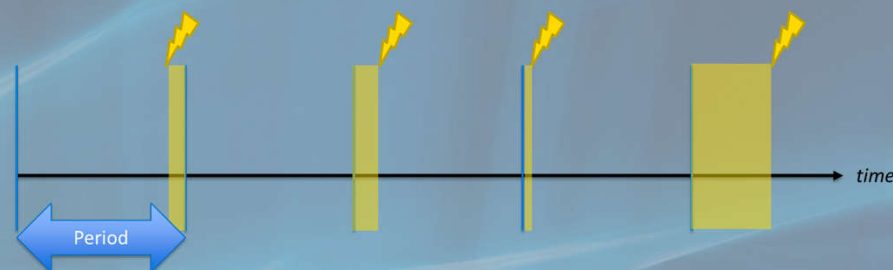


Shared understanding of time

Timely

“Done at the proper time”

<https://en.wiktionary.org/wiki/timely>

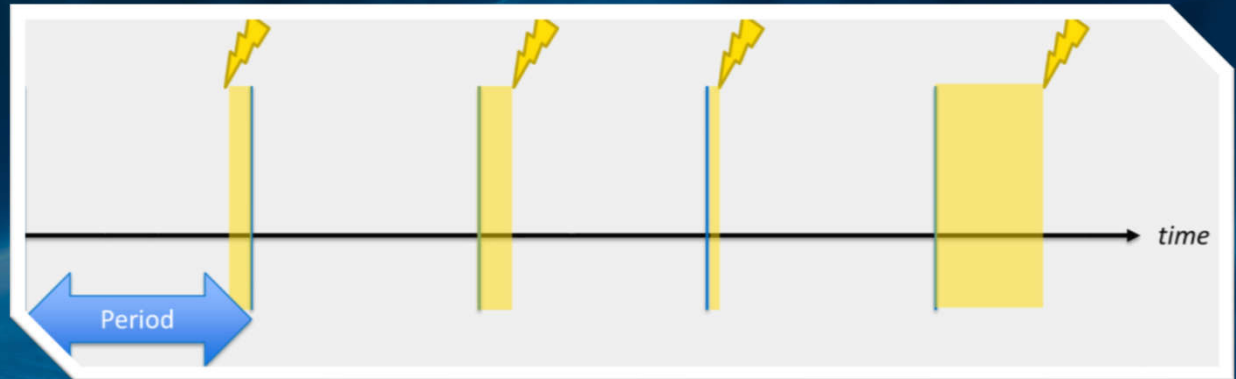


Relevant in this context:

- Deadlines, cycle period, cycle time, jitter
- Determinism, predictability, bounded latency

WHAT IS JITTER AND TIMELINESS?

- 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”

CONTEXT SWITCHING (1)

- 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.

CONTEXT SWITCHING (2)

- Context switching occurs only in kernel mode
 - The contents of all registers used by a process in user mode have already been saved on the Kernel Mode stack before performing the process switch.
- Hardware context:
 - A processes data must be loaded into the CPU registers before the process can resume execution.
 - Stored in task_struct and kernel-mode stack
 - Changing the stack point changes the running process.

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.

RATE-MONOTONIC SCHEDULING

- Given

- n tasks
- T_i : unique period per task
- C_i : computation time per task

- Assuming

- No shared resources (e.g. semaphore)
- Periods equal to deterministic deadlines
- Context switch takes no time
- **Rate-Monotonic priority assignment**
 - **Shorter deadlines are given higher priorities**
- Static priorities

- Guarantees

- Tasks can be scheduled...
- ... as long as CPU utilization is less than a threshold

Process	Execution Time	Period	Priority
P1	1	8	Medium
P2	2	5	
P3	2	10	Highest
			Lowest

$$\frac{1}{8} + \frac{2}{5} + \frac{2}{10} = 0.725$$

$$U = 3(2^{\frac{1}{3}} - 1) = 0.77976 \dots$$

$$0.725 < 0.77976 \dots$$

The system is schedulable

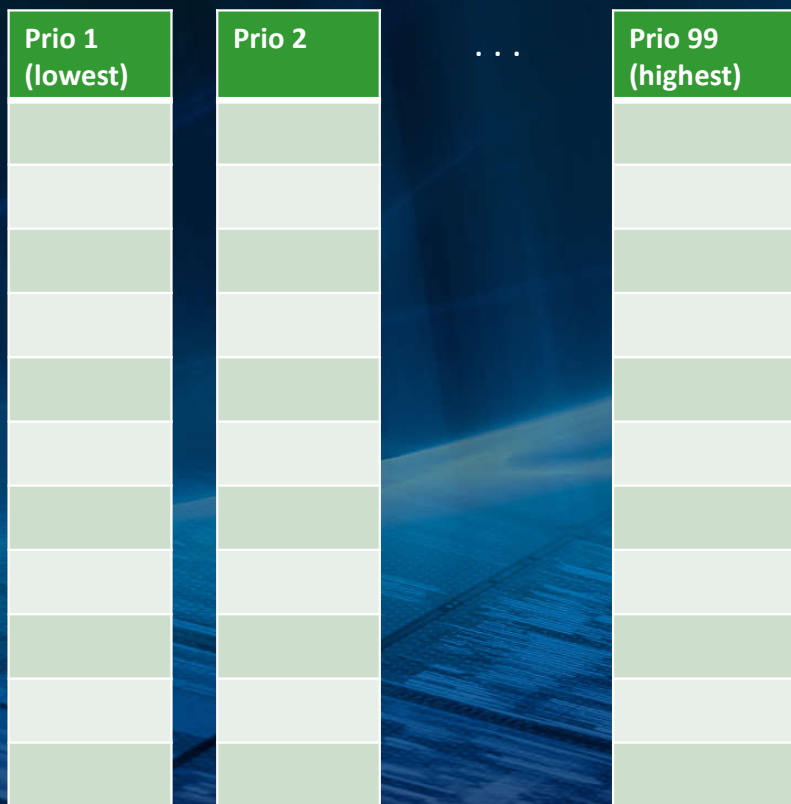
Slide 14

BAH29 Add disclaimer as example was taken from Wikipedia
Blanco Alcaine, Hector, 4/1/2019

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

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

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`

ALGORITHMS MAIN OBSTACLES TO PREEMPTION

Synchronization

- Priority Inversion: usage of a shared resource
- Synchronization primitives
 - May be designed for throughput instead of latency
 - E.g. non-pre-emptable spinlocks, Read-Copy-Update choices

Interrupt Handling

- How much “pre-emptible” is the handling of an interrupt

Algorithms

Priority Inversion

Low and High share a resource



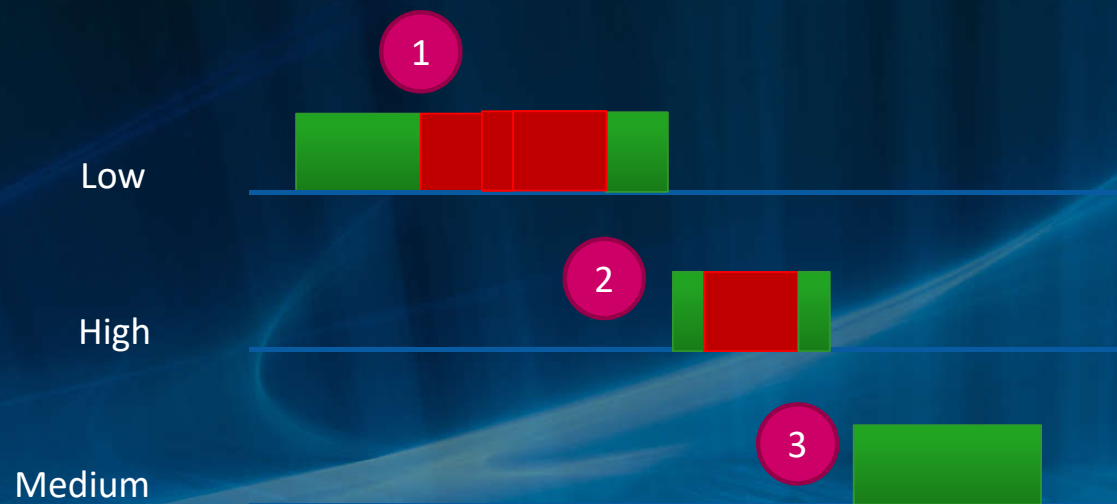
1. Low takes the resource
2. High preempts Low, cannot take the resource and is put to sleep
3. Medium preempts Low and executes completely before High!
4. High cannot take the resource and is put to sleep
5. Low uses and releases the resource, and finishes executing
6. High takes the resource and executes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

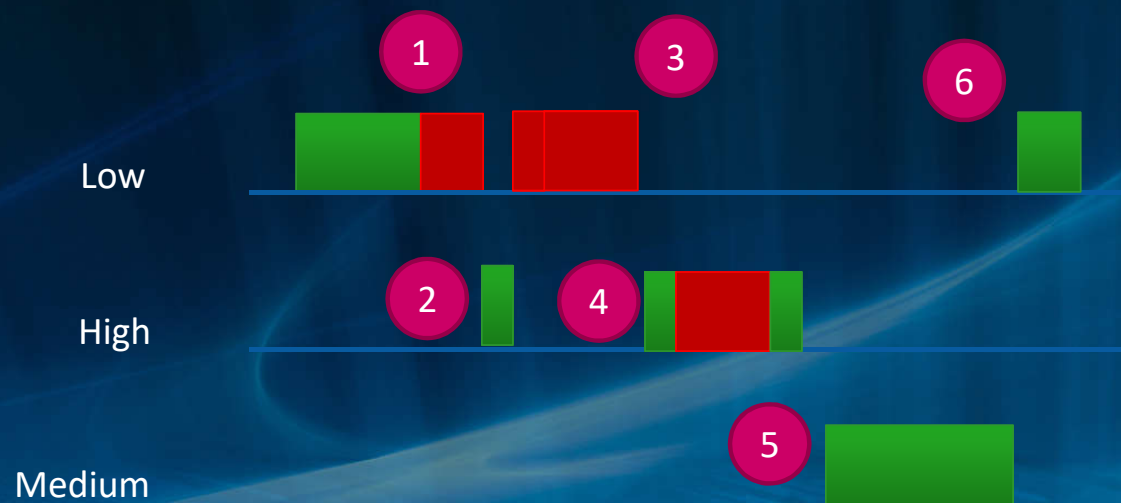



Algorithms

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



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

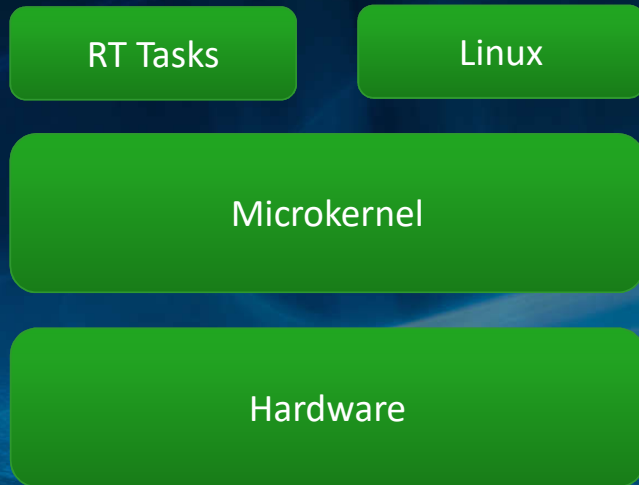
The mission was jeopardised by a concurrent software bug in the lander, which had been found in preflight testing but was deemed a glitch and therefore given a low priority as it only occurred in certain unanticipated heavy-load conditions, and the focus was on verifying the entry and landing code.

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

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

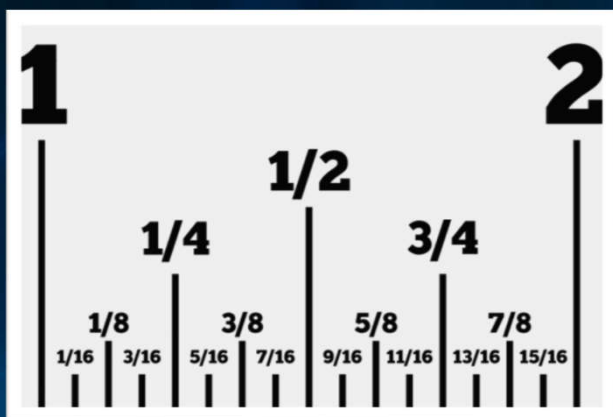
```
#include <stdio.h>
#include <unistd.h>
#include <sched.h>
```

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

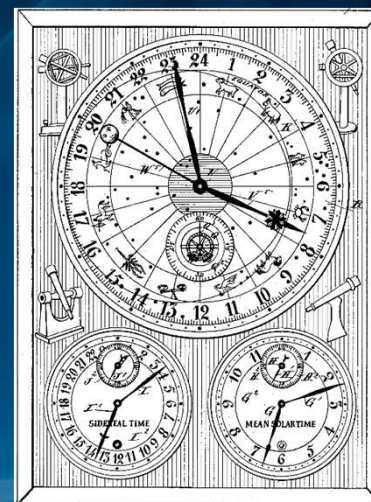
```
unsigned int handler(unsigned int irq, struct rtl_frame
*regs);
int main(void) {
    if ( rtl_request_irq( IRQ, handler )) {
        printf("failed to get irq %d\n", IRQ);
        return -1;
    }
    rtl_main_wait();
    rtl_free_irq(IRQ);
    return 1;
}
```

LAB: HIGH PRECISION TIMERS

OVERVIEW



High Resolution Timers



Time Synchronization

https://en.wikipedia.org/wiki/File:Measuring_-_Fractions_of_an_inch.svg [Public Domain]
https://en.wikipedia.org/wiki/Sidereal_time#/media/File:ConantClock.png [Public Domain]



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

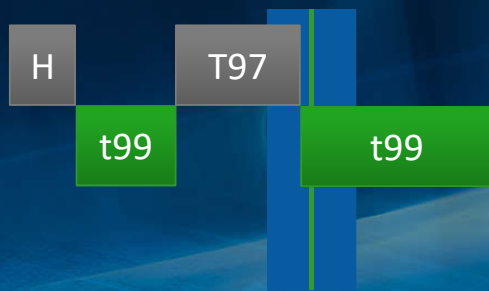
```
int clock_nanosleep(clockid_t clock_id,  
                    int flags,  
                    const struct timespec *request,  
                    struct timespec *remain);
```



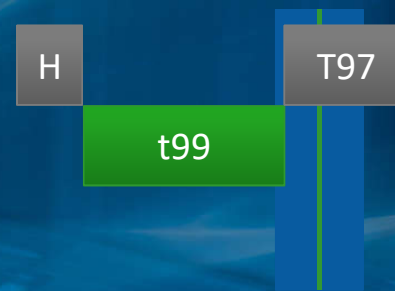
High Resolution Timers

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



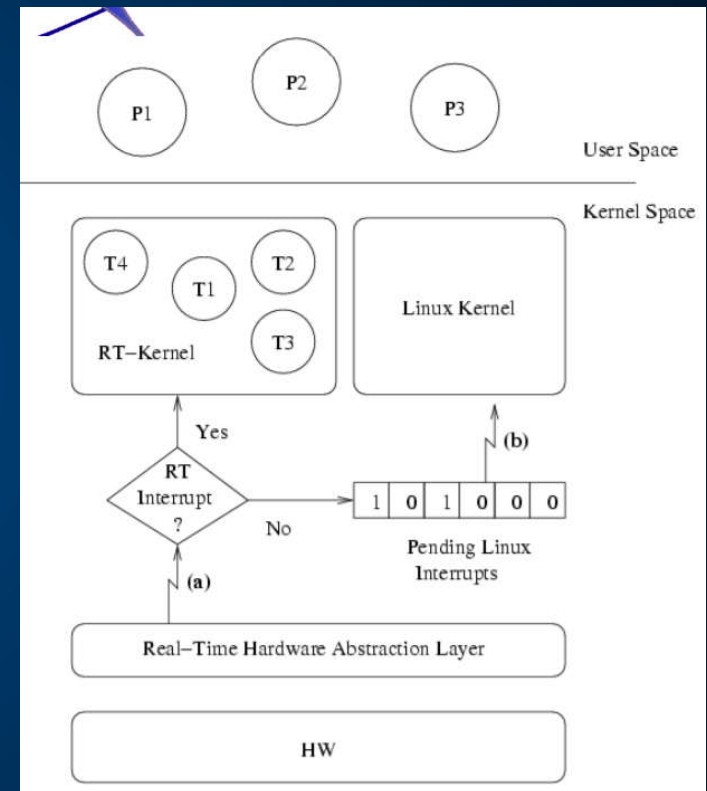
`clock_nanosleep`
t99 runs right after Hard IRQ handler
t99 meets the deadline

STRATEGIES FOR SOFT REAL-TIME LINUX

- Scheduling policy
- Preemption Improvement
 - No Forced Preemption (Server):
 - Voluntary Kernel Preemption (Desktop)
 - Pre-emptible Kernel (Low-Latency Desktop)
- 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

- 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



OVERVIEW

Processor

L1 cache

L2 cache

Main memory

Devices (e.g. disk, NIC)

Operating System
Virtual Memory Management

Real-Time Application
Stack and Heap Allocation

FI

REAL-TIME APPLICATION TUNING: OVERVIEW

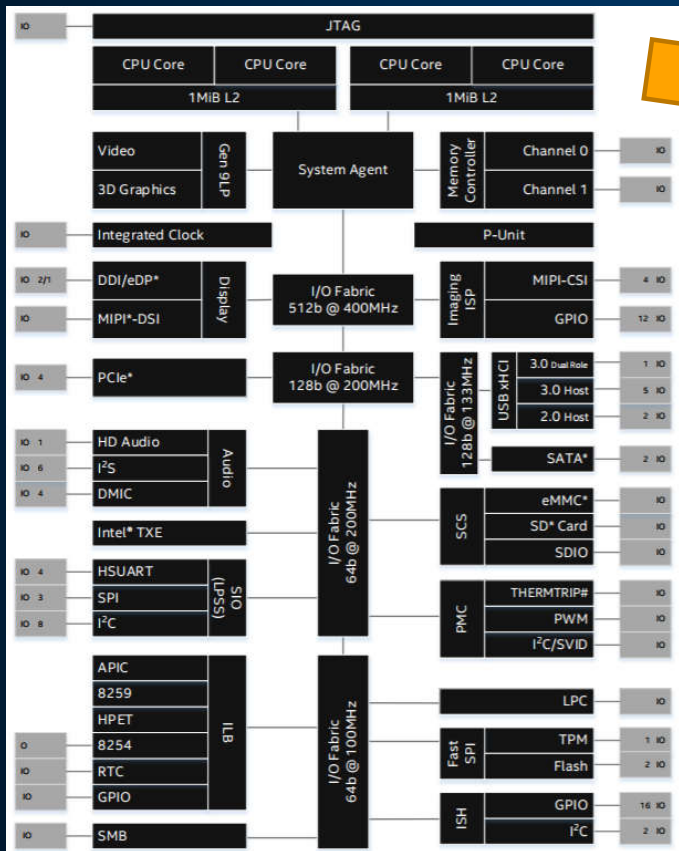
Virtual Memory Management

- Memory Locking

Stack and Heap Allocation

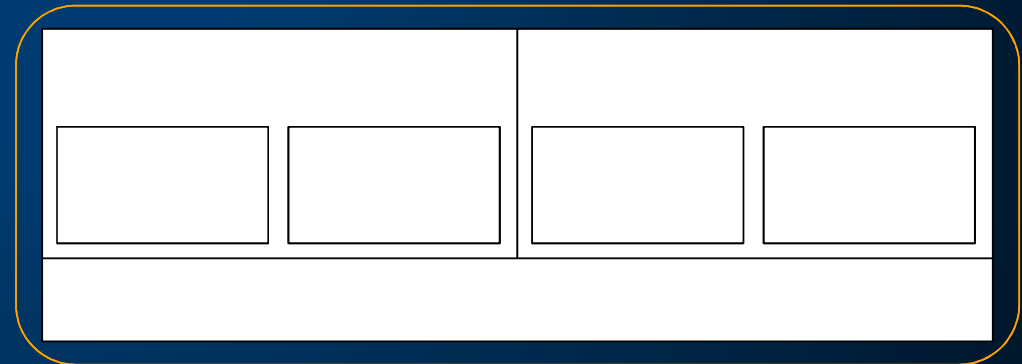
- Stack Memory for RT-Threads
- Dynamic Memory Allocation in RT-Threads

CACHE ALLOCATION TECHNOLOGY: CACHE HIERARCHY IN APL-I



Each module:

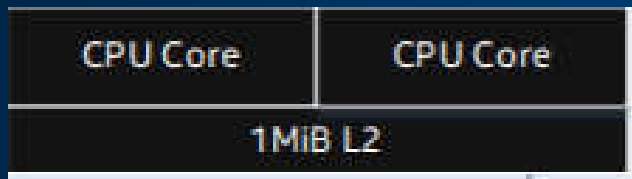
- 2 Cores
- 1 L2 Cache



CACHE ALLOCATION TECHNOLOGY: “NOISY NEIGHBOR”

Module 0

- Core 0
- Core 1



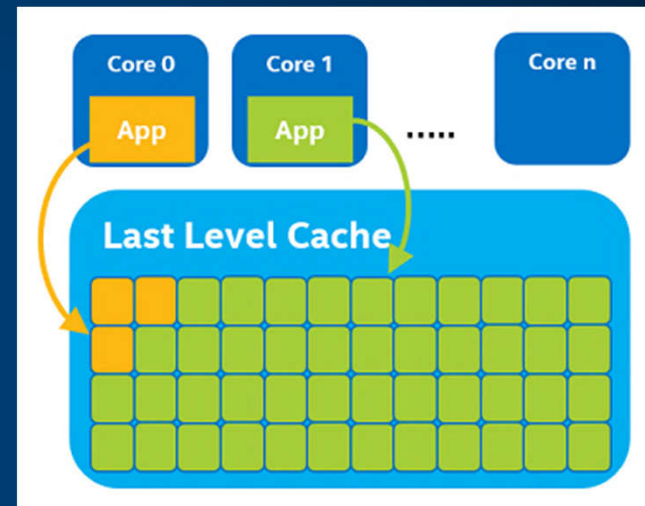
L2 Cache Shared

Most of it used by Core 1

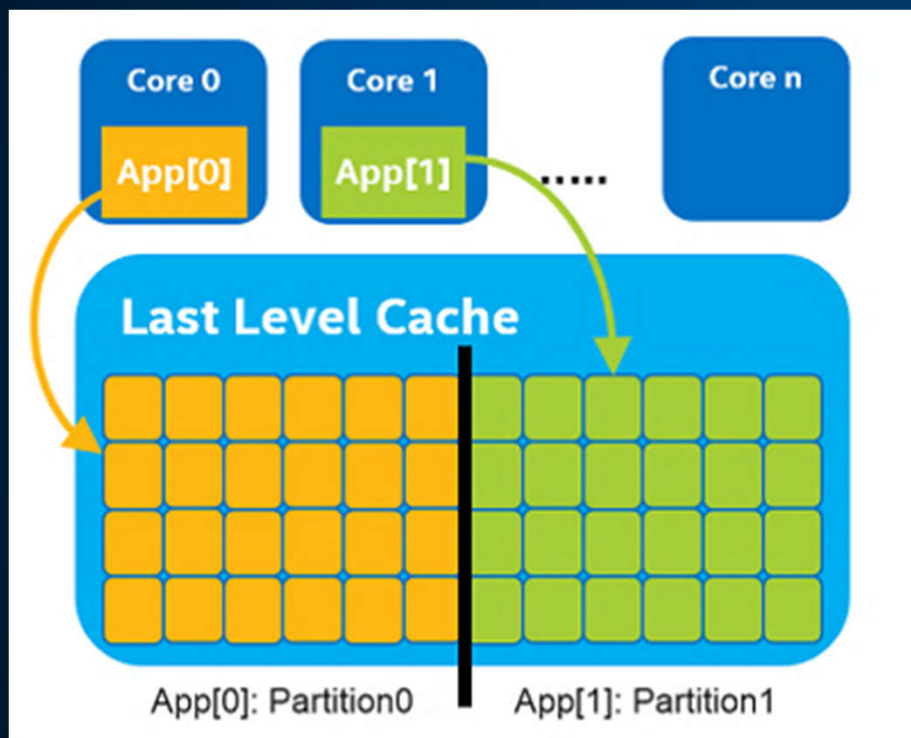
Hosts a “noisy neighbor” process

Impact for Core 0

- ↑ Cache misses
- ↑ Latency



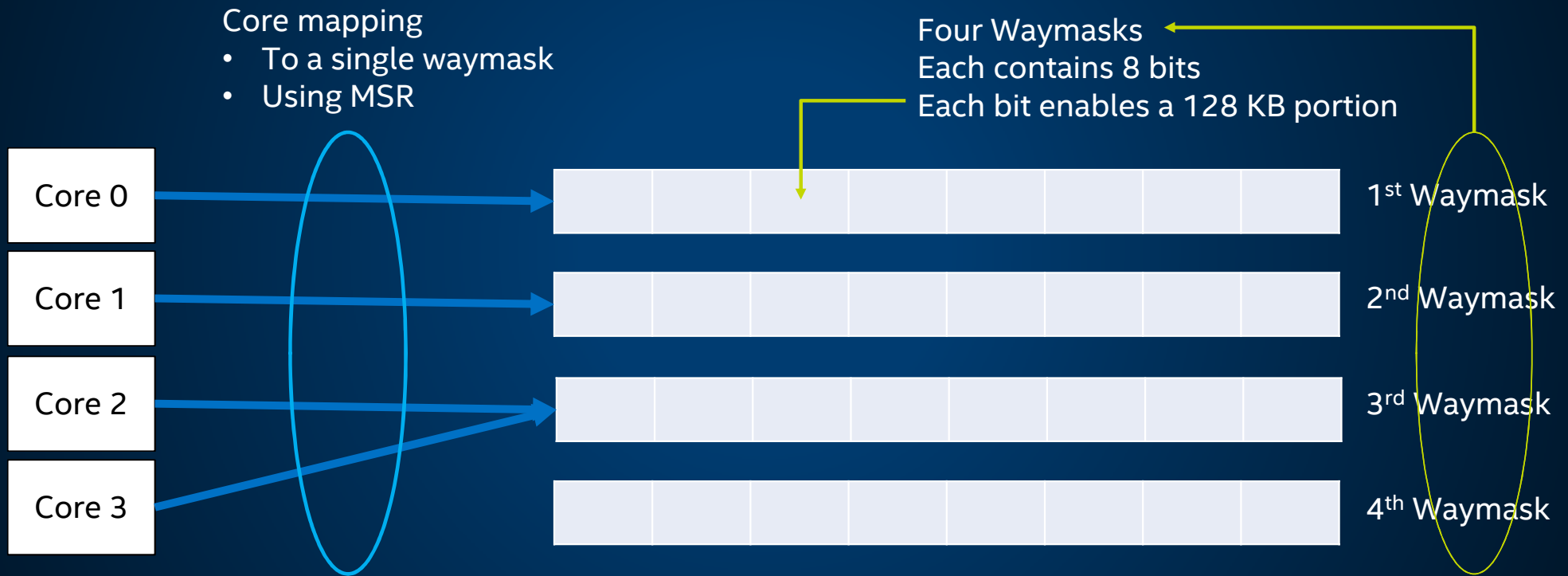
CACHE ALLOCATION TECHNOLOGY: LAST LEVEL CACHE PARTITIONING



Cache Allocation Technology

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

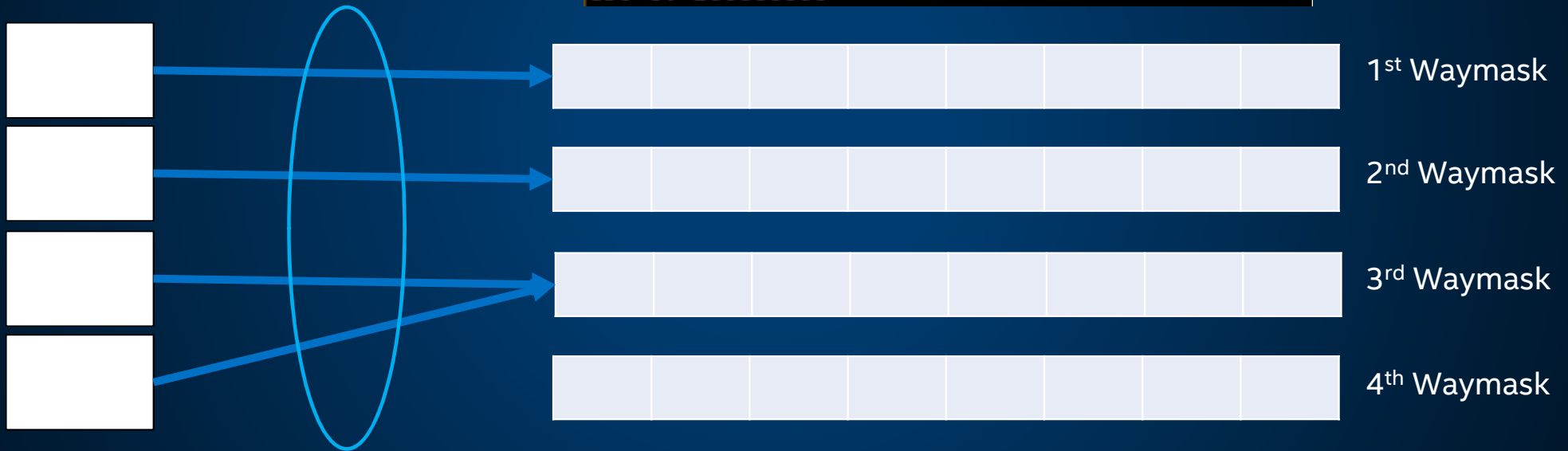
CACHE ALLOCATION TECHNOLOGY: CACHES, CORES AND WAYMASKS



CACHE ALLOCATION TECHNOLOGY READING CORE WAYMASKS

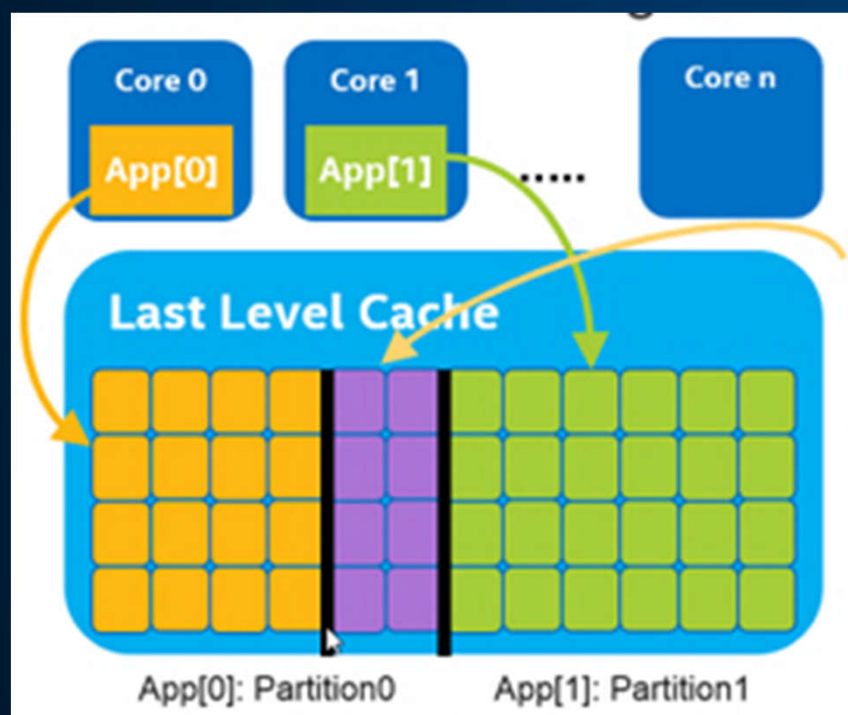
Read MSR
0xC8F
For all CPUs

```
root@intel-corei7-64-cavs-hda:~# rdmsr -a 0xc8f
CPU 0: 0
CPU 1: 100000000
CPU 2: 200000000
CPU 3: 200000000
```



CACHE ALLOCATION TECHNOLOGY

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

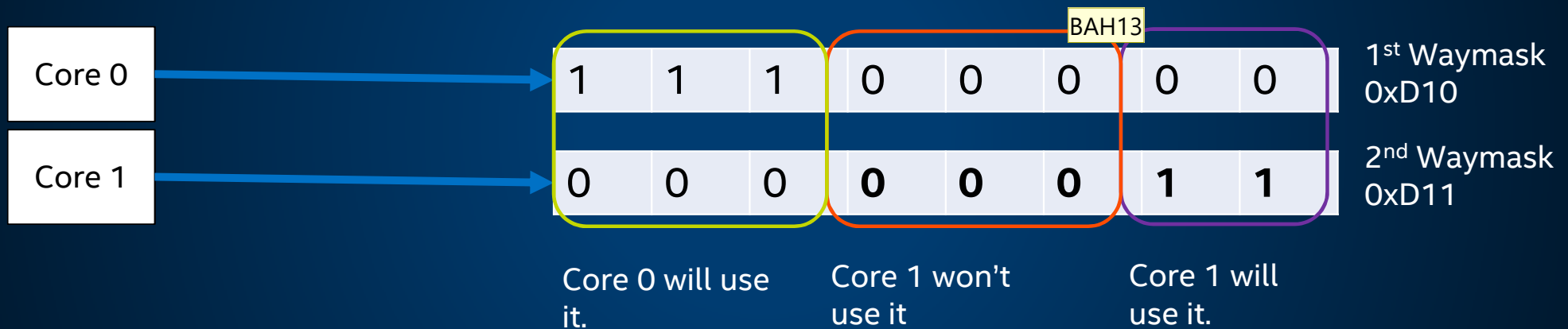


“Pseudo-Locking”

- 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

CACHE ALLOCATION TECHNOLOGY: "PSEUDO-LOCKING": ALLOCATE CACHE

```
$ wrmsr 0xD10 0xD0  
$ wrmsr 0xD11 0x3
```



After allocating cache for Core 1, run your RT-App in isolation....

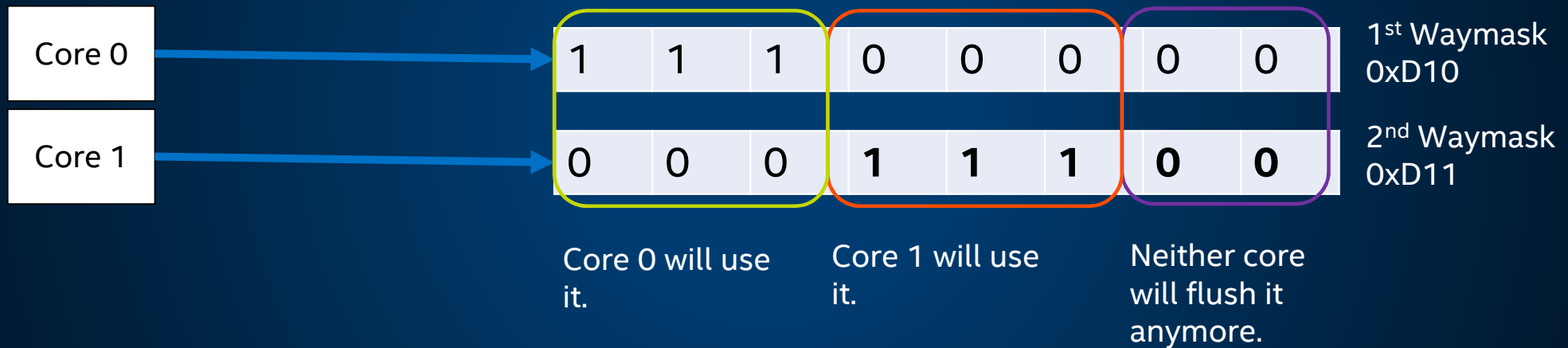
Slide 40

BAH13 IIRC there are restrictions about where the 1s go in the waymask, the need to be consecutive, etc. Cross-check it.
Blanco Alcaine, Hector, 4/6/2018

CACHE ALLOCATION TECHNOLOGY

“PSEUDO-LOCKING”: PREVENT FURTHER FLUSHES

```
$ wrmsr 0xD10 0xD0  
$ wrmsr 0xD11 0x1C
```



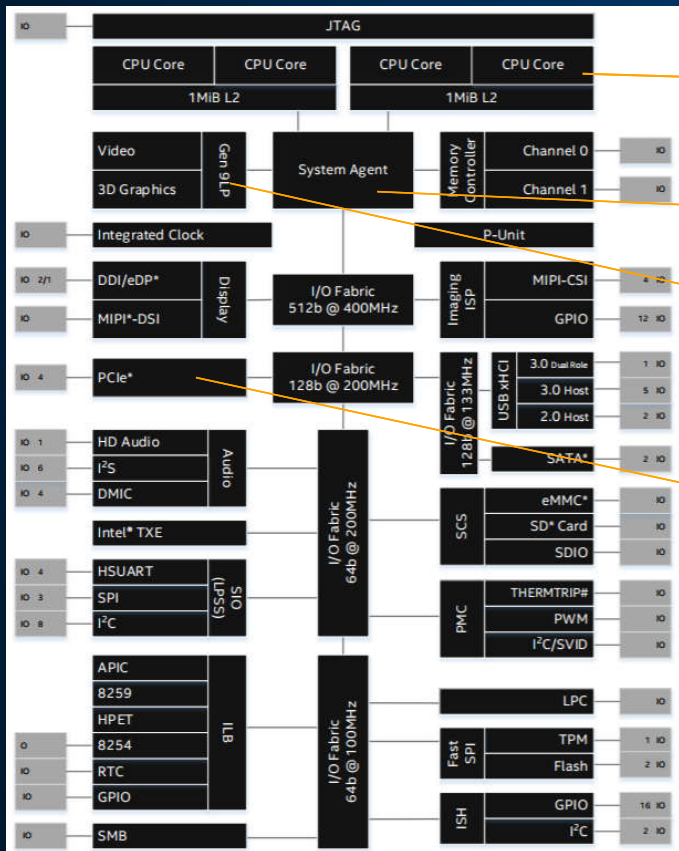
CACHE ALLOCATION TECHNOLOGY: HIGH LEVEL TOOLS AND LIBRARIES

- Resource CKernel interface for CPU resource allocation

PQoS / Intel® RDT utility

- Linux command line utility to configure CAT
- Also supports ontlol
- Linux related “Resource Director Technologies”

POWER MANAGEMENT EVENTS



CPU

System Agent

Graphics

PCI Express

POWER MANAGEMENT EVENTS ADVANCED CONFIGURATION AND POWER INTERFACE (ACPI) STATES

Computer System (Gx/Sx)

Package (Cx / PCx)
Module (MCx)

Core/CPU (Cx / CCx)

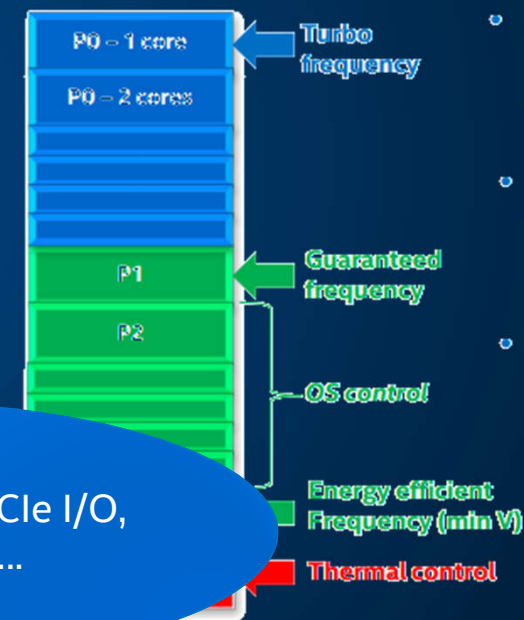
Device (Dx)

Power-Performance (Px)
When operating (C0/PC0)

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			

Active: Frequency Scaling



For CPU, graphics, PCIe I/O,
System Agent....

Idle: Power Saving States

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

Optimize Power When Idle

Sample C-States (may vary)

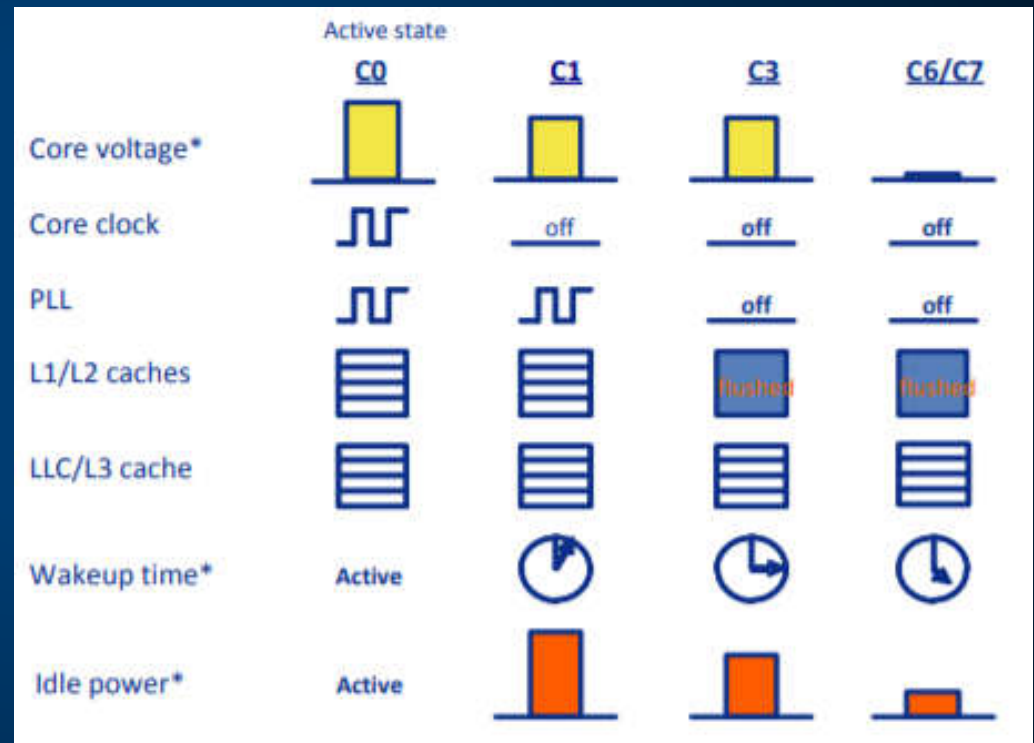
C0: operating

C1: halted

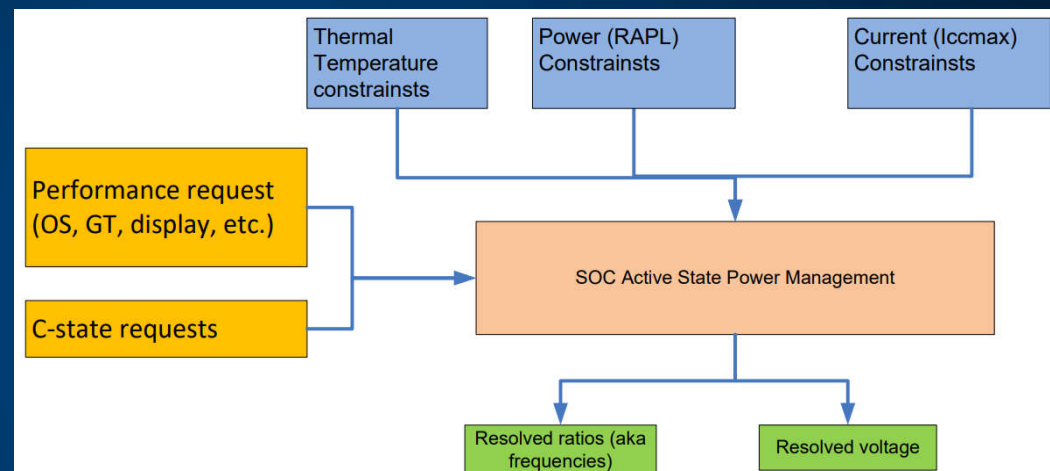
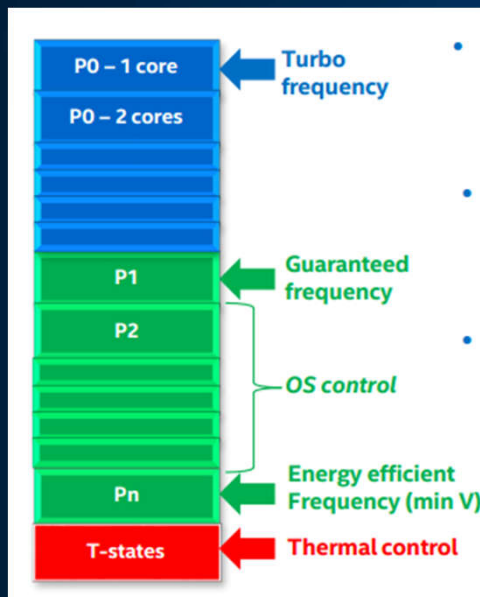
C3: caches flushed

Tuning

Disable in Firmware (BIOS Configuration)*



POWER MANAGEMENT EVENTS CPU: DYNAMIC VOLTAGE AND FREQUENCY SCALING



- Tuning
 - Disable in Firmware (BIOS Configuration)

POWER MANAGEMENT EVENTS GRAPHICS

Render C-State 6 (RC6)

- Very low voltage mode when idle

- Different RC6 modes available, differing in entry and exit latencies and voltage

GT PM (Graphics Technology Power Management)

- Various Power Management features supported by the CPU

Frequency Scaling

- Can be set via software

Tuning

- RC6 and GT PM: Disable in Firmware (BIOS Configuration)

- Kernel boot parameters also available

- Frequency Scaling

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

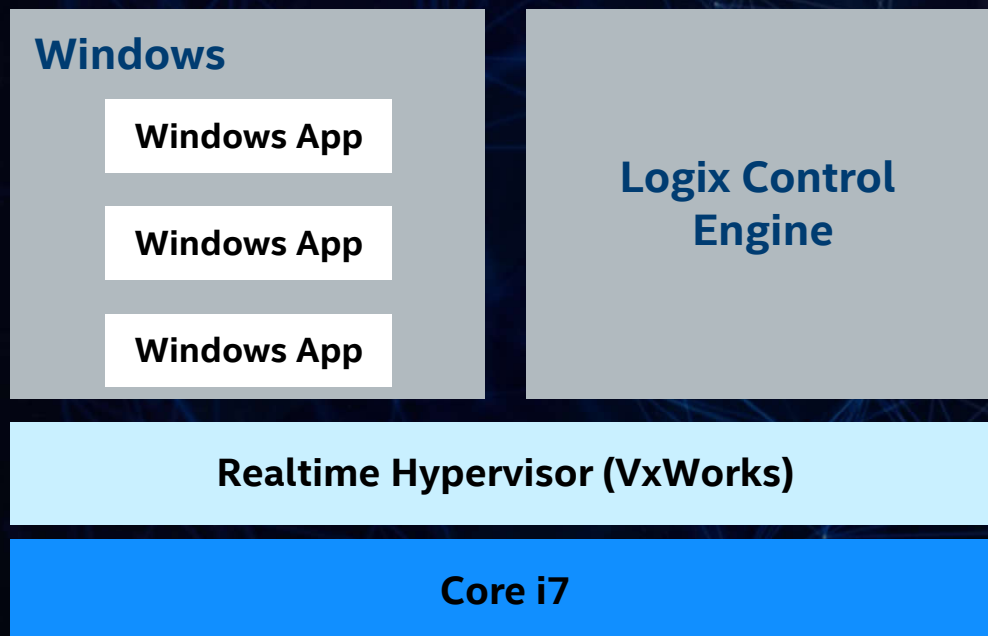
Tuning

- Disable in Firmware (BIOS Configuration)

MULTI-FUNCTION CONTROLLER WITH VXWORKS

ROCKWELL COMPACT LOGIX 5480 CONTROLLER

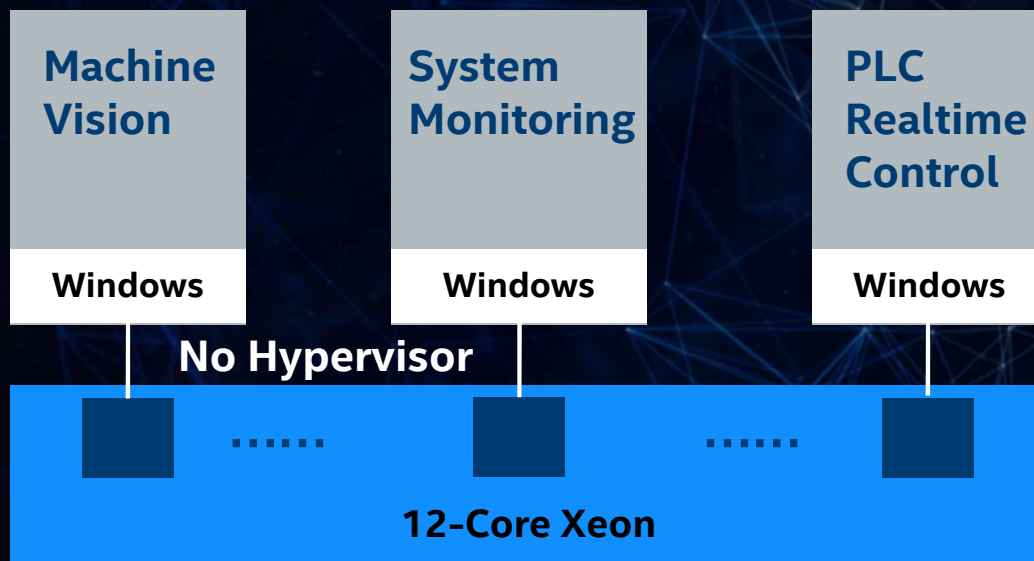
<https://www.youtube.com/watch?v=TovqAiZcCCk>



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

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">▪ Maximum performance by direct assignment of workloads to cores (no Hypervisor)▪ Differentiation over competitor products▪ Workloads assigned statically from Twincat environment

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

SUMMARY

- Workload Consolidation trends are driving down capital equipment and maintenance prices while increasing factory flexibility and efficiency.
- Intel® Virtualization Technologies including Intel® VT-x, VT-d, VT-c and VT-g allow multiple workloads to be consolidated into a single machine.
- Intel® Virtualization Technology is being adopted by real-time hypervisors, operating systems and applications enabling real-time workload to be consolidated.
- Intel supports and Industrial Ecosystem of Commercial and Open Source partners.

RESOURCES

- [Intel® Virtualization Technology \(Intel® VT\)](#)
- [Intel® Virtualization Technology for Directed I/O \(VT-d\): Enhancing Intel platforms for efficient virtualization of I/O devices](#)
- [PCI-SIG SR-IOV Primer](#)
- [Intel® Data Direct I/O Technology](#)
- [Data Plane Development Kit \(DPDK\)](#)
- [Does My Processor Support Intel® Virtualization Technology?](#)

