Real-Time Systems

Hermann Härtig, Marcus Völp

Hardware

17/12/2012

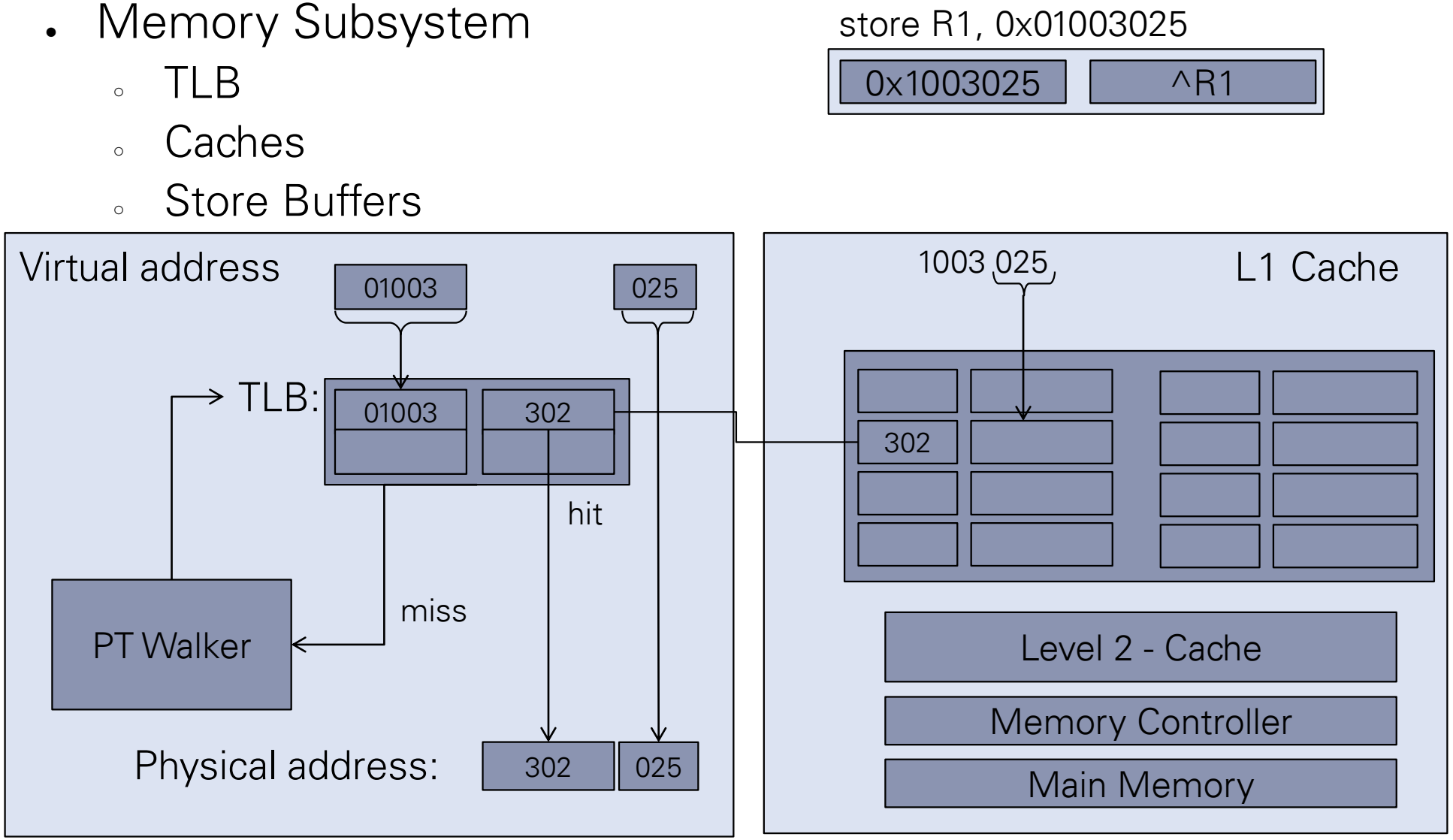
TECHNISCHE UNIVERSITÄT DRESDEN

Outline

- Hardware: Source of Unpredictability
  - Caches
  - Pipeline
  - Interrupt Latency
  - System Management Mode
- Special Purpose Hardware for "Embedded" Real-Time Systems
- Use unpredictable Hardware more predictable
- Real-Time Communication / Buses in separate Lecture

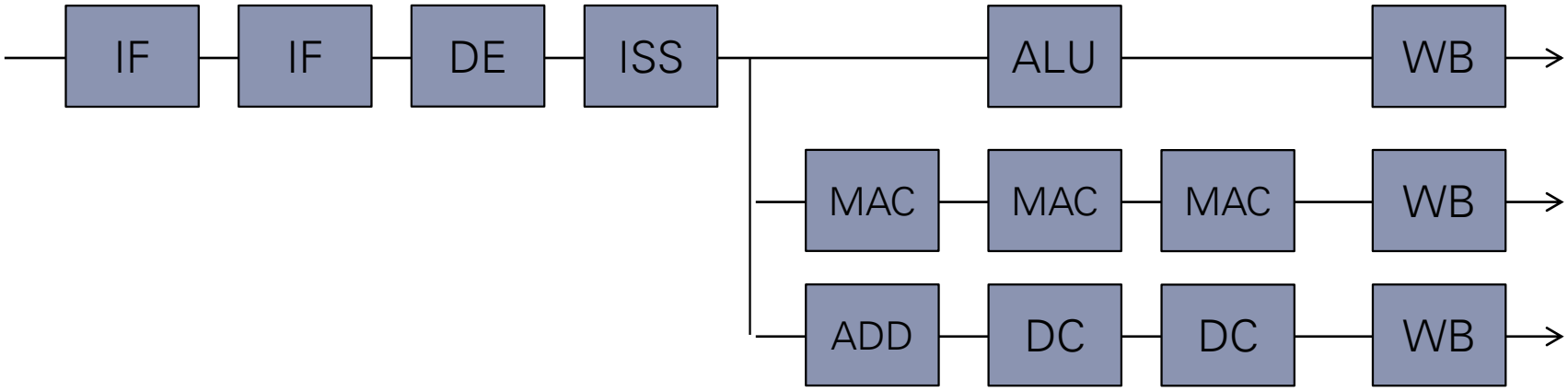
Sources of Unpredictability

- Memory Subsystem
  - TLB
  - Caches
  - Store Buffers



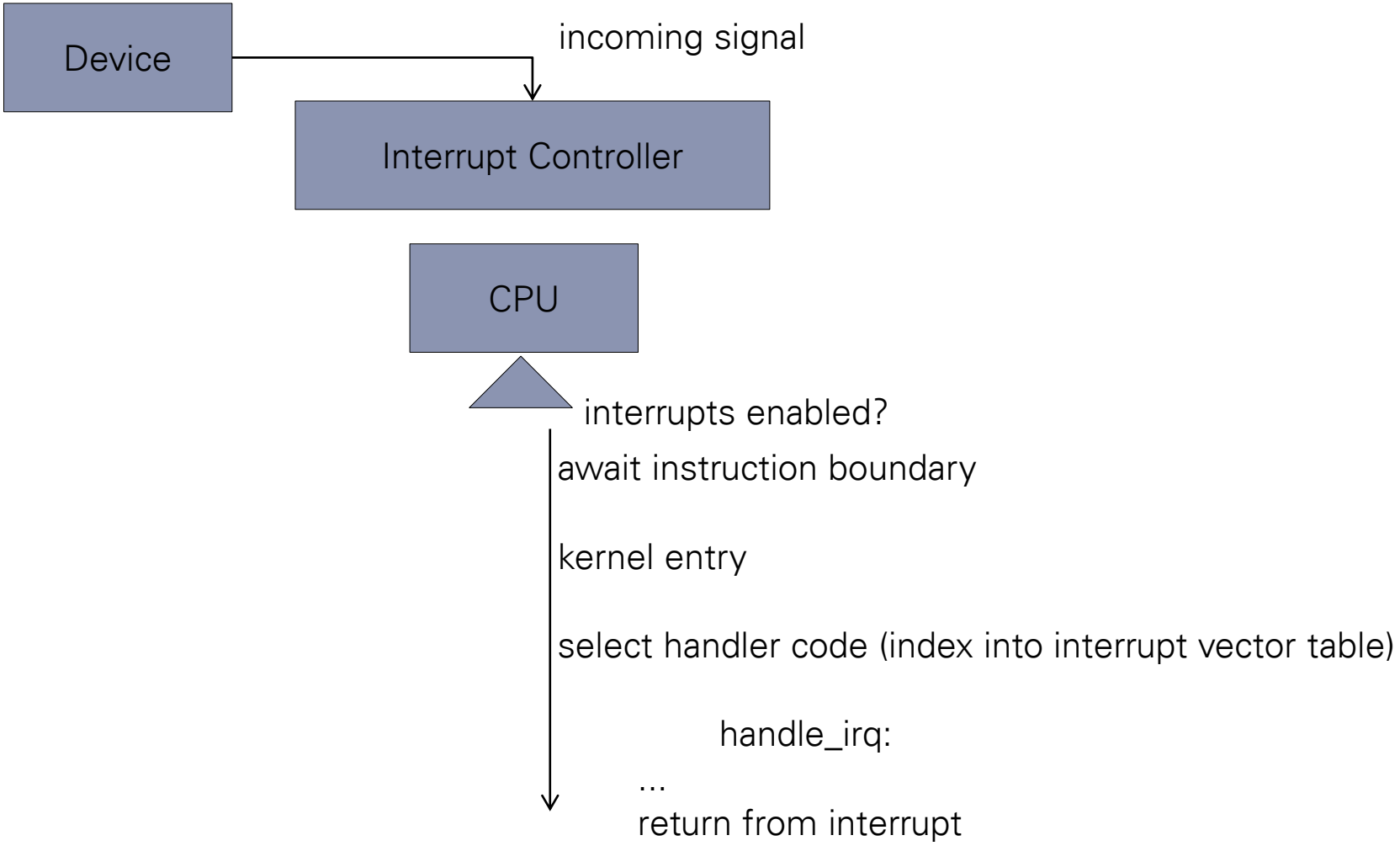
Sources of Unpredictability

- Processor Pipeline (ARM 11)

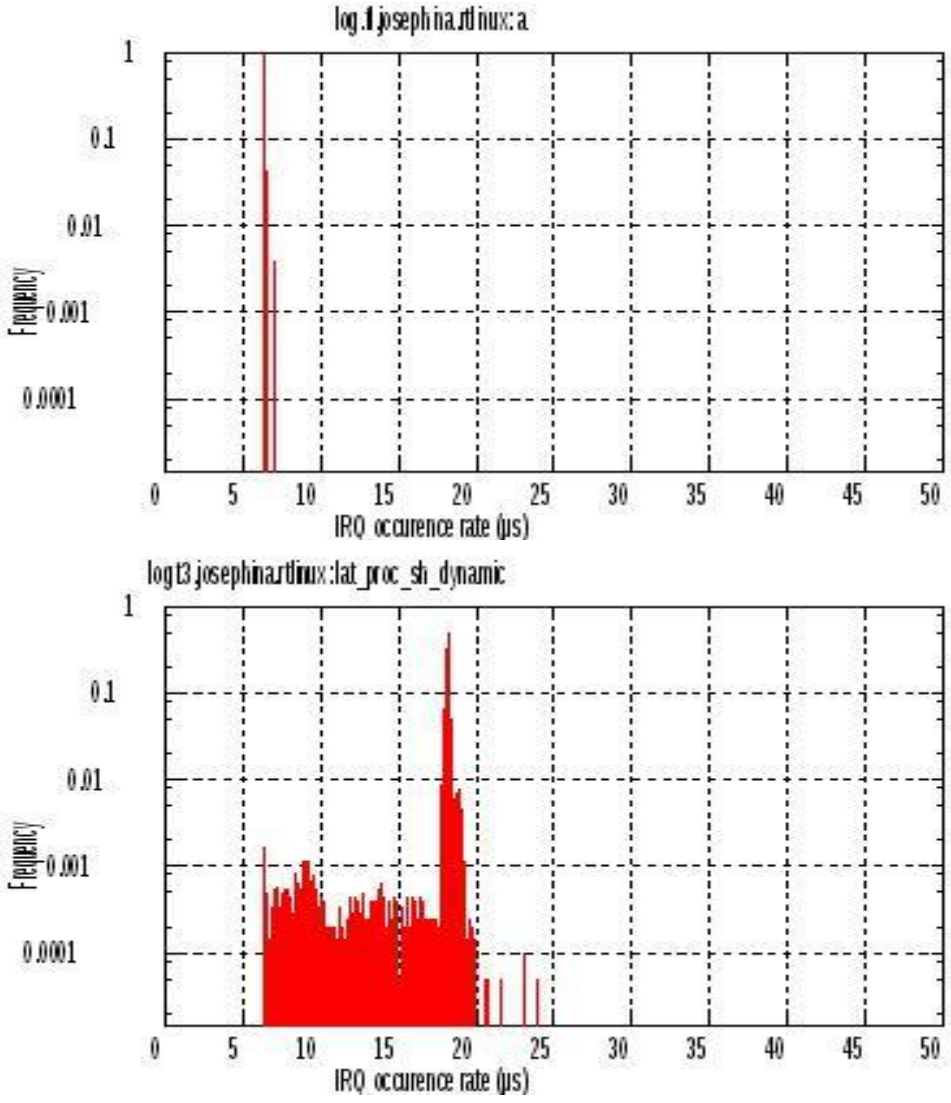


Sources of Unpredictability

- Interrupt Latency



Interrupt Response Time – RTLinux



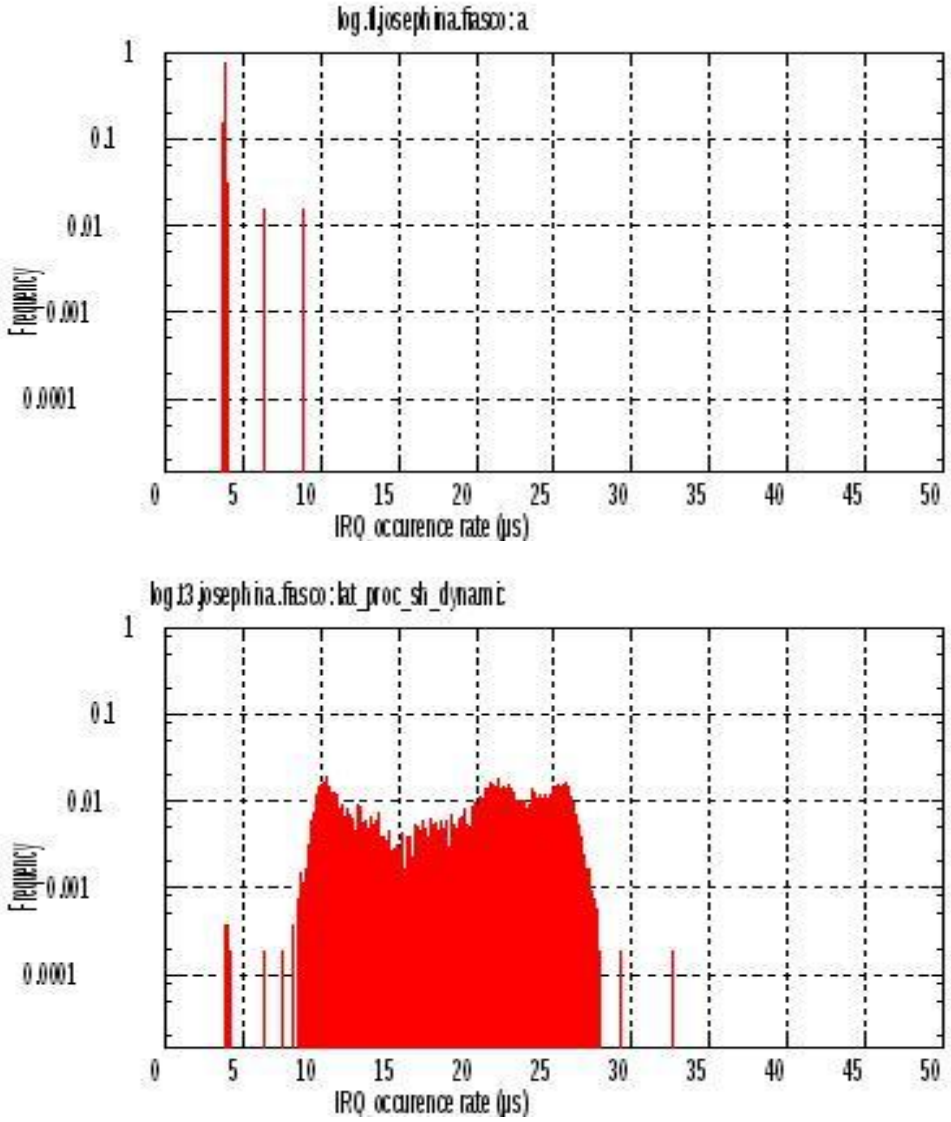
Interrupt response time:
Time from occurrence of interrupt to first instruction of RT Task

No parallel load (idle): **13 µs**

High parallel load: **68 µs**
(Benchmark, Cache-Flodder)

Measured on Intel P4 1.6 GHz

Interrupt Response Time – L4RTL (+AS switch)



Interrupt response time:
Time from occurrence of interrupt to first instruction of RT Task

No parallel load (idle): **43 µs**

High parallel load: **85 µs**
(Benchmark, Cache-Flodder)

Measured on Intel P4 1.6 GHz

System Management Mode (SMM)

- PC platforms
- “sits” underneath operating system
- Invoked using non-maskable interrupt
- Used for platform specifics, correction of design errors, thermal management
- Can be switched off, but better do not try
- Adds unpredictable delays

Outline

- Hardware is Source of Unpredictability
- Special Purpose Hardware for "Embedded" Real-Time Systems
  - Low Latency Interrupt Mode
  - Peripheral Event Controller
  - Capture – Compare Units
  - Scratchpad Memories
  - Real-Time Clocks
- Use unpredictable Hardware more predictable
- Real-Time Communication / Buses in separate Lecture

Low Latency Interrupts ...

… are considered of primary importance

- Sampling of Data in a High Rate (Sensors, Video, ...)
- Fast Response Times (Break Control, …)
- Part of context switch time .....

Low Latency Interrupts

- Several Interrupt Modi: ARM IRQ + FIQ
  - FIQ interrupts IRQ handler
  - 5 registers immediately available for FIQ handler
    no need to save register content
- ARM Low Interrupt Latency Configuration
  - Minimize worst case interrupt latency
    - disable Hit-under-Miss in Cache
    - abandon pending restartable memory OPs
    - restart memory OP on return from interrupt
    - do not use multi-word load / store instructions
    - avoid accesses to slow memory
      (device memory, strong ordering of memory accesses)
  - Worst Case FIQ Latency (PL190 VIC) :

Interrupt synchronization	3 cycles
Worst case execution time of current instruction	7 cycles
Entry to first instruction	2 cycles
SUM:	12 cycles

Low Latency Interrupts (2)

Peripheral Event Controller (PEC)

- Memory ↔ io-register move
- triggered by signal

- **Programmers Interface:**
  - counter — decrement on signal, trigger CPU interrupt on 0
  - byte / word select — select width of transfer (byte or word)
  - source / dest — increase update source / destination address
  - source / dest — address

-
```
signal () {
  if (counter--)
    trigger_interrupt();
  else
    *dest++ = *src;
}
```
=> minimal response time (~3 cycles)
=> PEC can execute at external clock rate

Example: SAB 80C166 (Siemens, 1990)

Timestamps / Event triggers

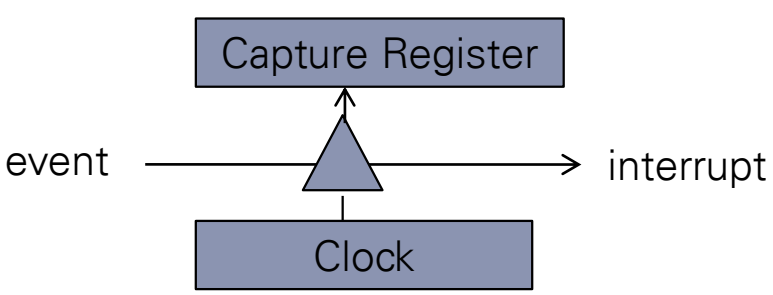
- Precise timestamps / trigger times
  with interrupt-based sampling is difficult
  - system load influences jitter in interrupt response time
  - atomic sections in OS delay interrupts
  - jitter in signal delivery (from signal source to CPU interrupt pin)
    - buses, interrupt controller

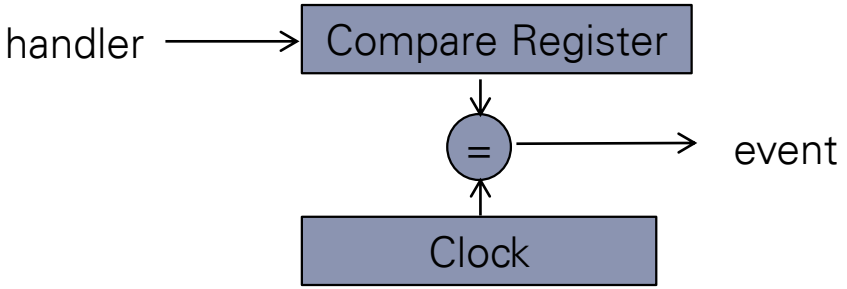
Capture + Compare Unit

- **Problem**
  - precise timestamp of event
  - (engine sensor triggered at cylinder position **= t**)
  - trigger events at precise points in time
  - (fire the engine at ***t + x***)

  - interrupt handlers have too high a jitter to meet the points in time

Capture Compare





Combination PEC, Capture-Compare

- Compare + PEC: highly efficient triggering of events
- Capture + PEC: highly efficient sampling

- no SW intervention

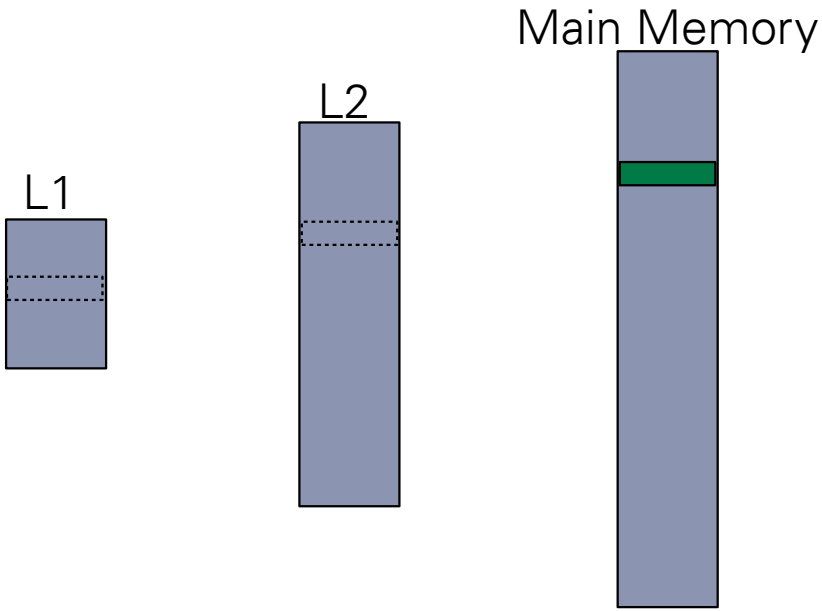
System on a Chip / Chip Multiprocessor

- Processor Chip contains entire system
- CPU Cores / Peripheral Components / Memory available as masks for weaver
- Configure system as required

- Examples:
  - Mobile Phones today:
    General Purpose Core (e.g., ARM) + Baseband DSP Processor
  - Game Console (Cell):
    General Purpose (PPC) + special purpose Graphics
  - Sensor + 8-bit Controller + CAN bus

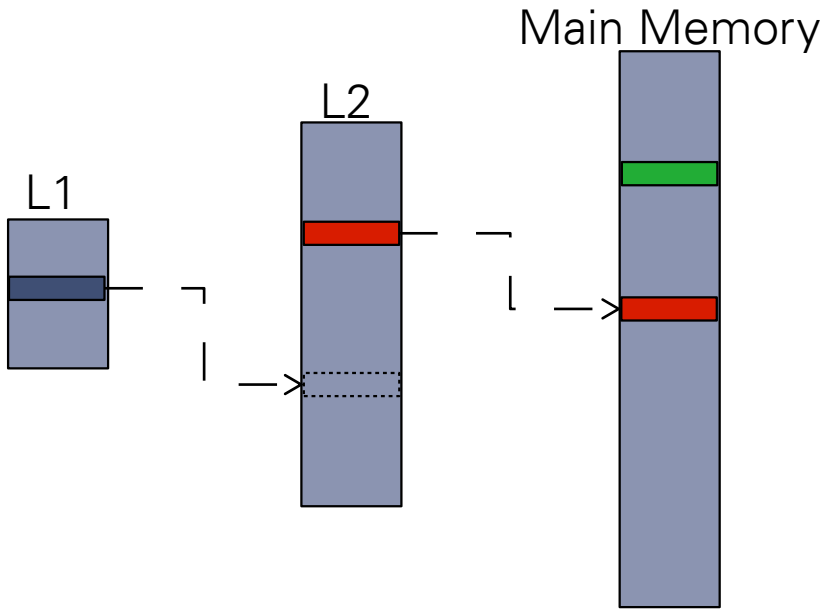
Scratch-pad Memory vs. Caches

- Synonyms:
  [Scratchpad / On-Chip / Tightly-Coupled] Memory
- Cache:
  - transparent addressing scheme
  - cache controller fills / replaces cachelines in parallel to CPU activity
  - difficult to predict worst-case execution time



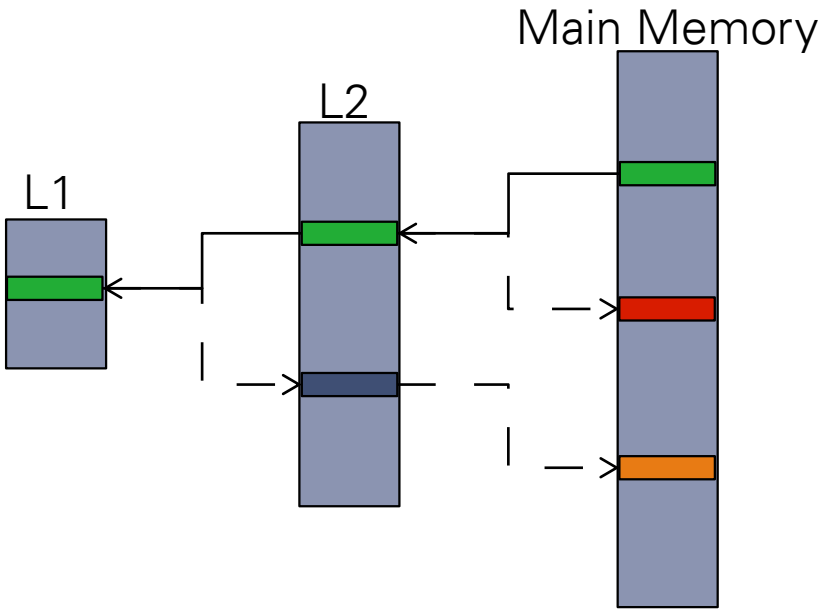
Scratch-pad Memory vs. Caches (2)

- Synonyms:
  [Scratchpad / On-Chip / Tightly-Coupled] Memory
- Cache:
  - transparent addressing scheme
  - cache controller fills / replaces cachelines in parallel to CPU activity
  - difficult to predict worst-case execution time



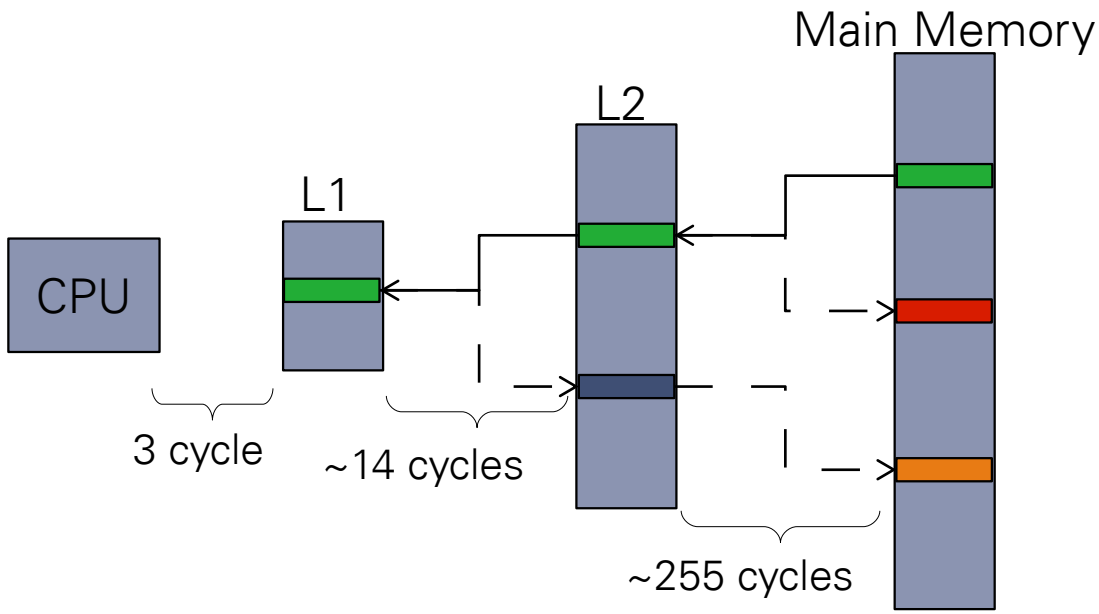
Scratch-pad Memory vs. Caches (3)

- Synonyms:
  [Scratchpad / On-Chip / Tightly-Coupled] Memory
- Cache:
  - transparent addressing scheme
  - cache controller fills / replaces cachelines in parallel to CPU activity
  - difficult to predict worst-case execution time



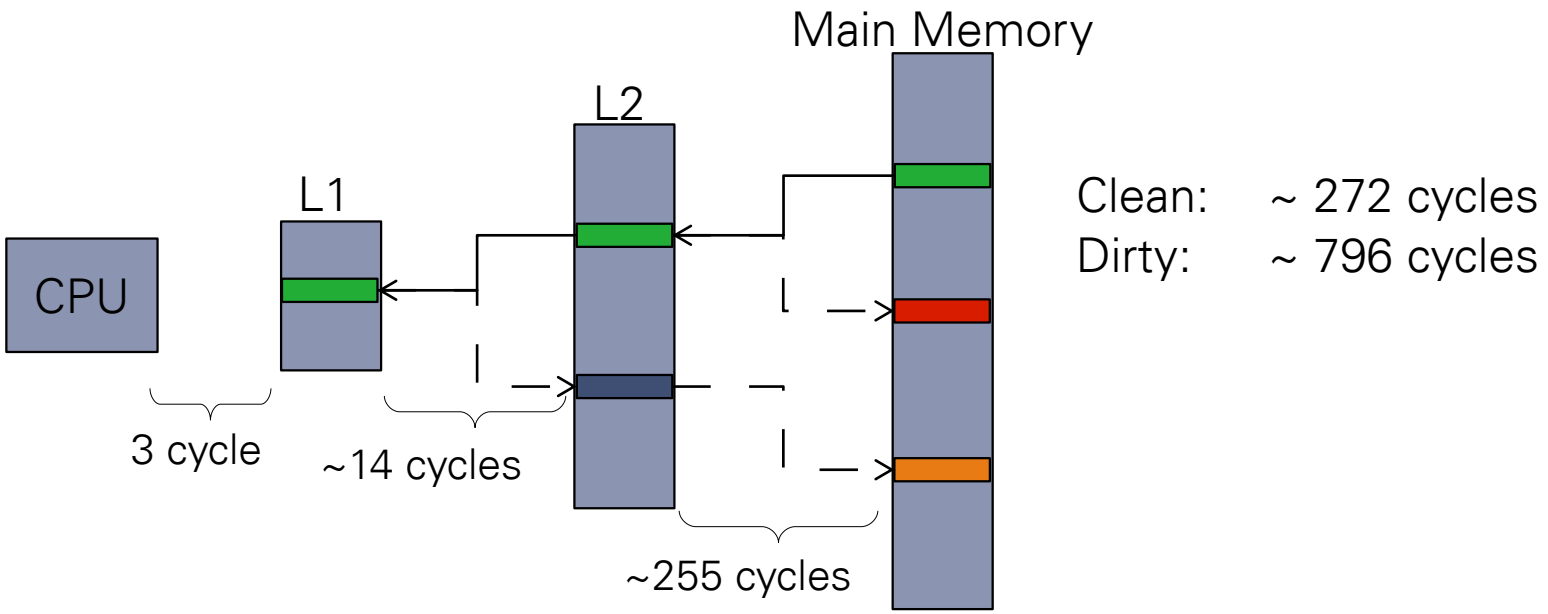
Scratch-pad Memory vs. Caches (4)

- Synonyms: [Scratchpad / On-Chip / Tightly-Coupled] Memory
- Cache:
  - transparent addressing scheme
  - cache controller fills / replaces cachelines in parallel to CPU activity
  - difficult to predict worst-case execution time

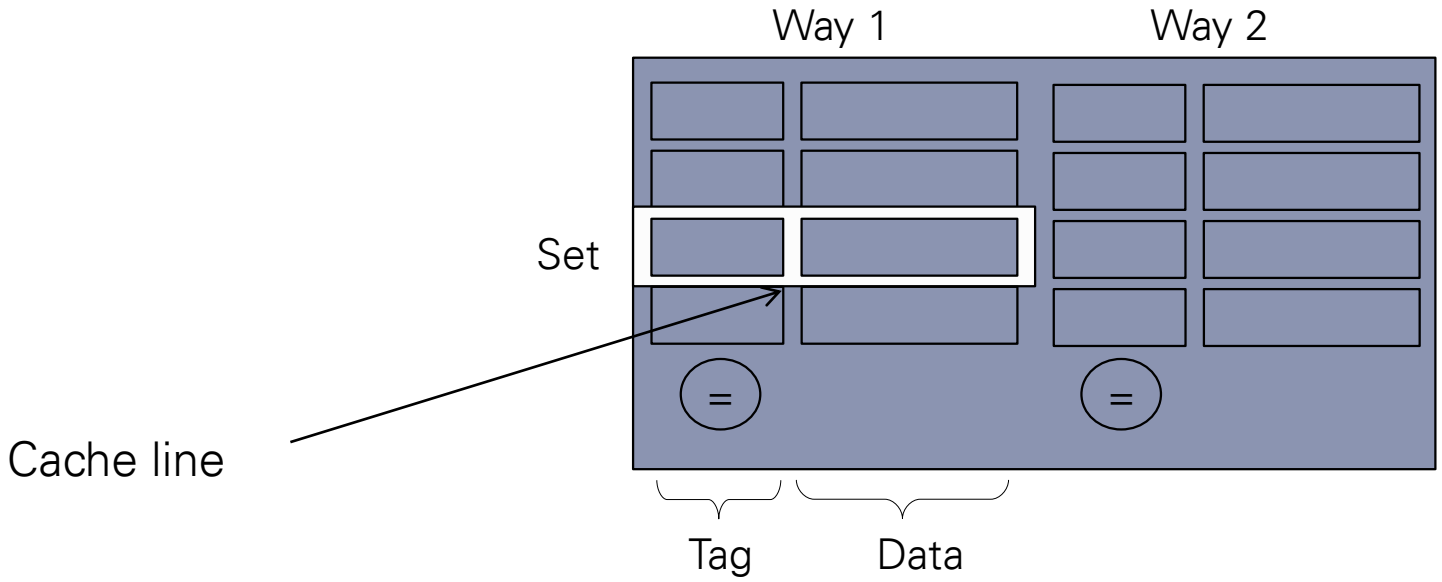


Scratch-pad Memory vs. Caches (5)

- Synonyms: [Scratchpad / On-Chip / Tightly-Coupled] Memory
- Cache:
  - transparent addressing scheme
  - cache controller fills / replaces cachelines in parallel to CPU activity
  - difficult to predict worst-case execution time



Scratch-pad Memory vs. Caches (6)



Cache lock:

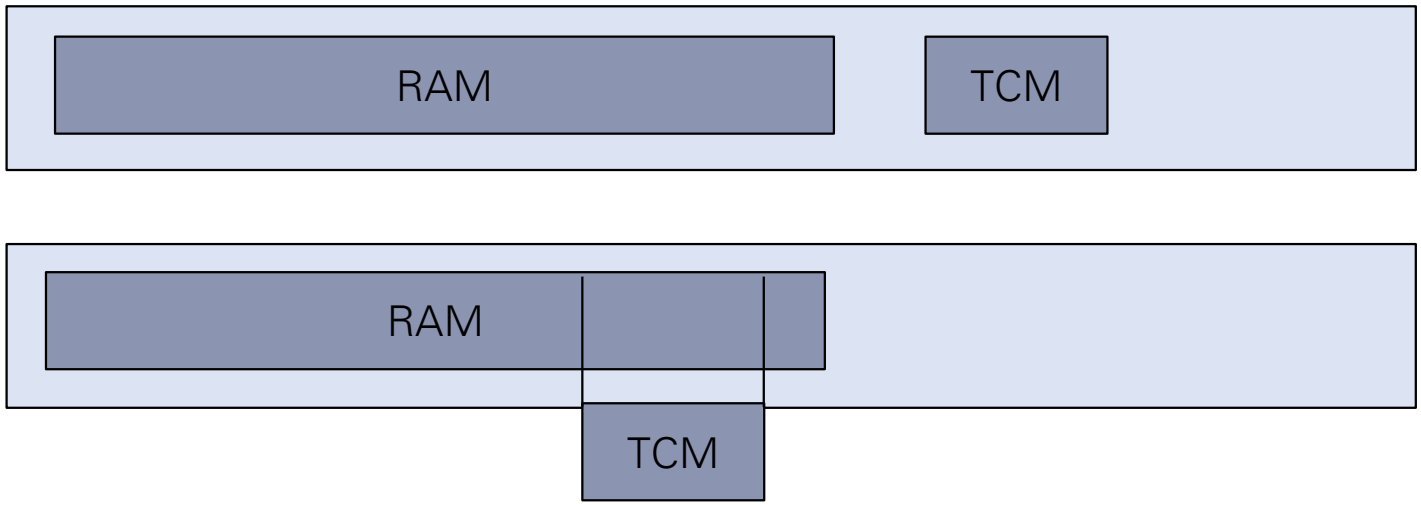
- Lock complete cache way
- Allocate to unlocked cache ways only

Scratch-pad Memory vs. Caches (7)

- Scratchpad Memory:
  - memory is addressed directly
    (device mapped to physical memory space)
  - 2 cycles latency / currently ~ 64 KB (more ~4 MB to come)
  - must explicitly manage data allocation
  - Compiler-based approaches:
    - static: determine code + data hot spots +
      allocate scratchpad memory for hot spots
    - dynamic: copy data to / from scratchpad memory

Scratch-pad Memory vs. Caches (8)

- ARM Tightly Coupled Memory:



  - overlay normal RAM with TCM
  - simplified cache logic for TCM memory
    - keeps track whether RAM or TCM holds current value
    - on miss: copies data from underlying RAM to TCM
    - What are the differences ??

Real Time Clocks

- Multiple Clocks in SoC
  - Core Cycle Count — 200 MHz fine grained
    - \+ fast to access
    - – accuracy ; clock skew ; subject to power management (dvfs)
  - Audio Codec
  - PCI — 33 MHz/ 66 MHz
  - Timer
  - RTC — 32.768 kHz; high precision; small clock skew

Outline

- Hardware is Source of Unpredictability
- Special Purpose Hardware for "Embedded" Real-Time Systems
- Use unpredictable Hardware more predictable
  - Cache Partitioning
- Real-Time Communication / Buses in separate Lecture

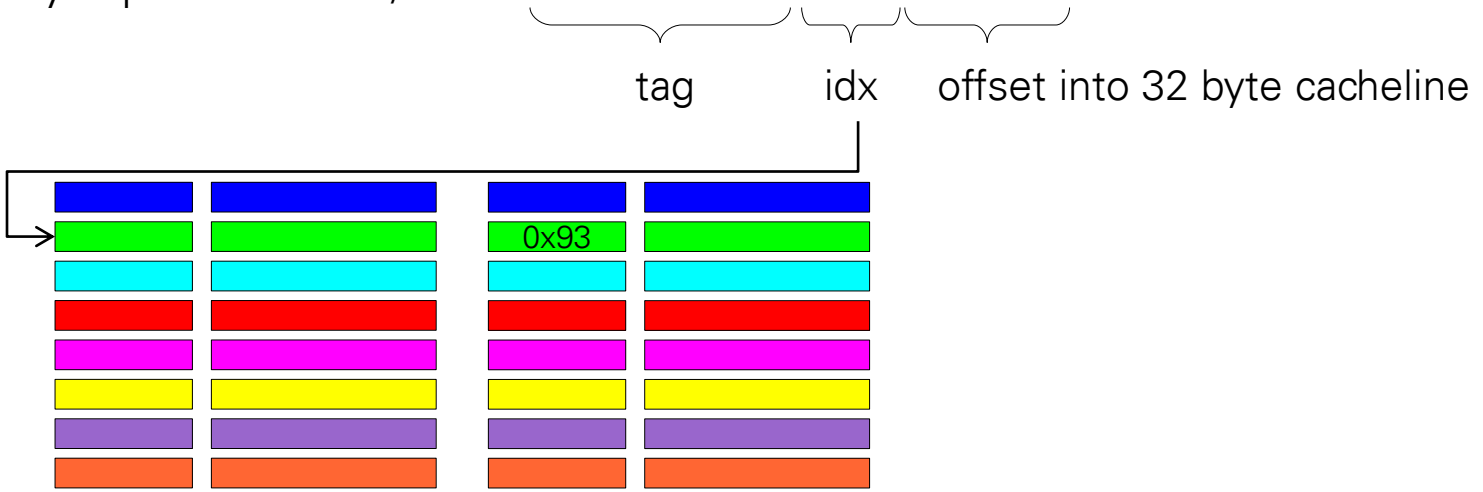
(OS Controlled) Cache Partitioning

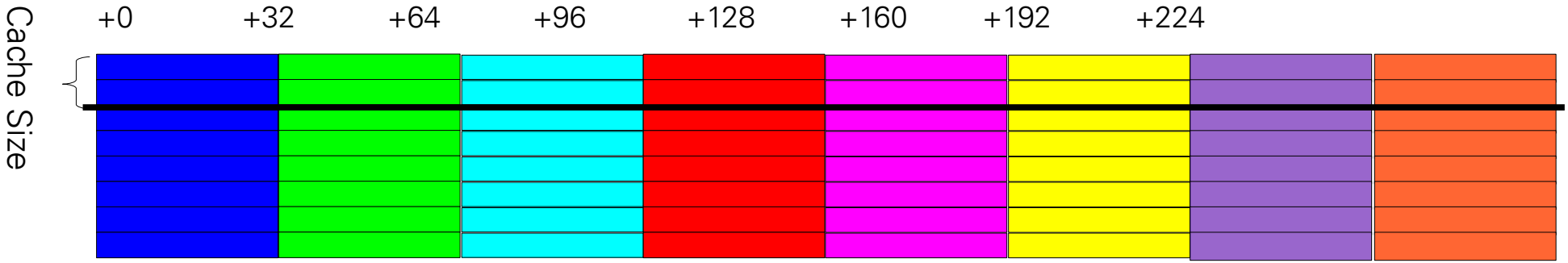
- Goal
  - partition cache to minimize interference between RT/RT or RT/NRT applications
- General method:
  - address regions map to cache sets
  - Control allocation to address regions
- Approaches
  - Compiler/linker
  - OS controlled
    - transparent to application
    - no need to rely on cooperating applications
    - isolates malicious, erroneous applications

OS Controlled Cache Partitioning (2)

Cache Coloring

16 bit address (binary representation): 1001 0011 0010 0100

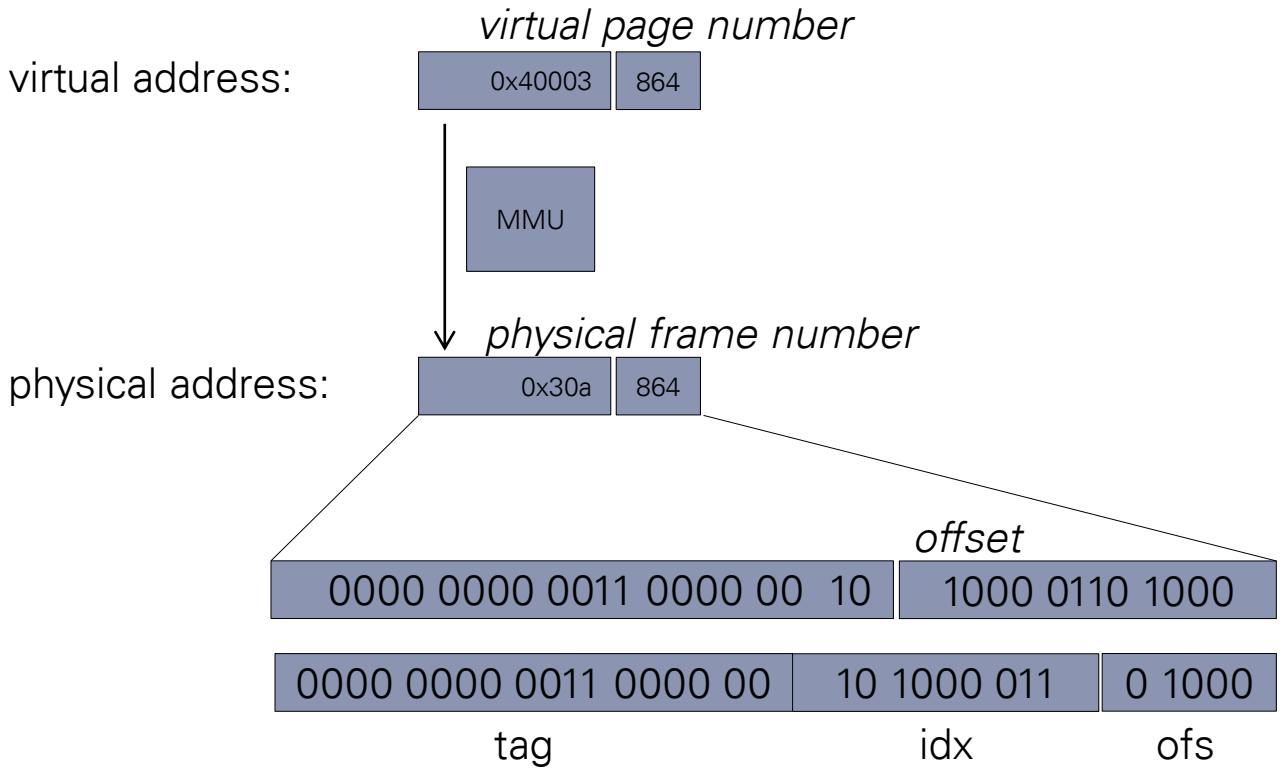




OS Controlled Cache Partitioning (3)

Cache Coloring

Address translation and caches

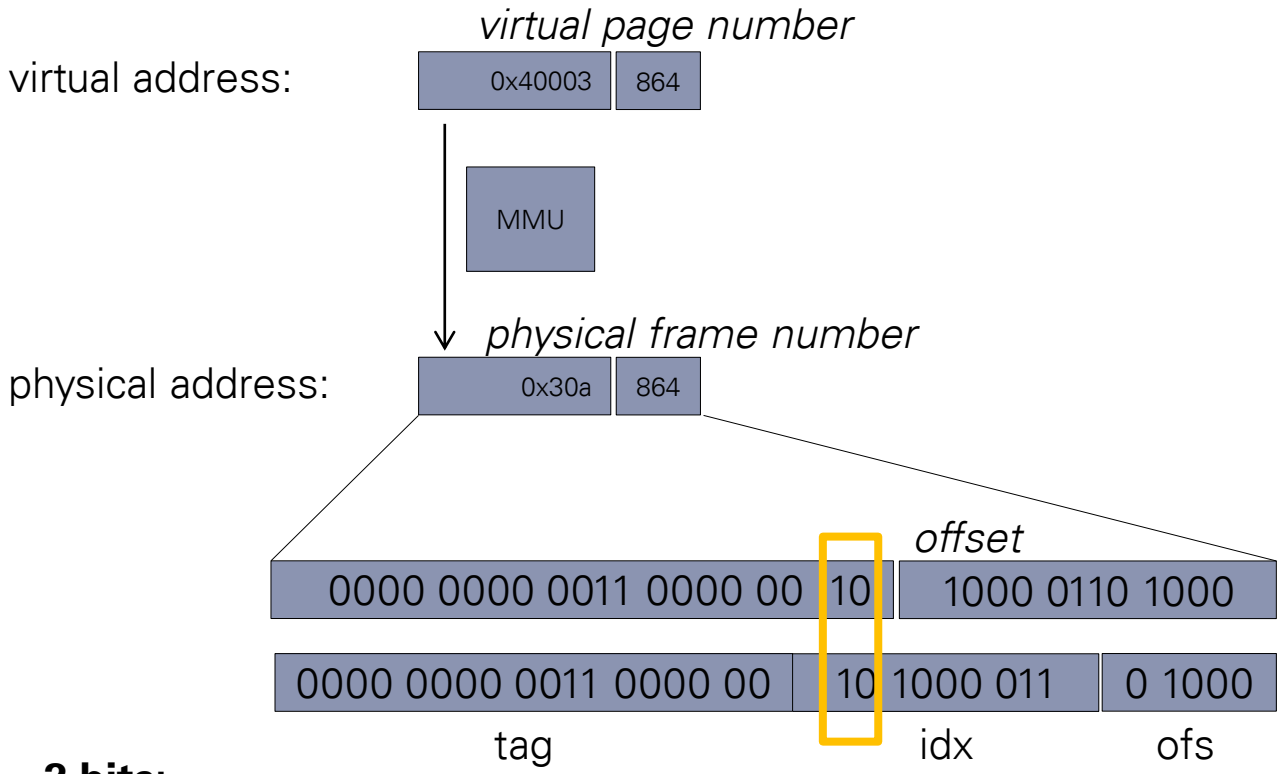


L1 Cache:
64KB, 4 Way, 32byte CL

OS Controlled Cache Partitioning (3)

Cache Coloring

Address translation and caches



L1 Cache:
64KB, 4 Way, 32byte CL

2 bits:

- subject to address translation
- evaluated to determine cache set

  => assign different colors to different RT + non RT Apps.

  => allocate in the OS memory frames of respective color

OS Controlled Cache Partitioning (5)

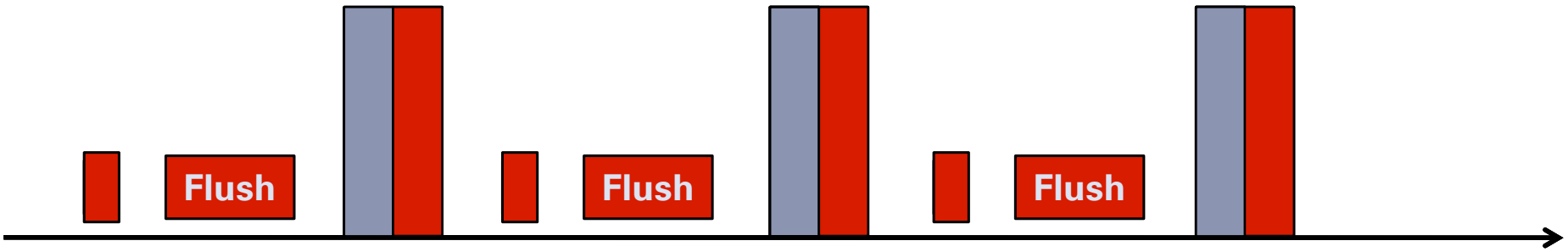
Scenario

- high priority real-time task (color = 00)
- runs in frequent short intervals

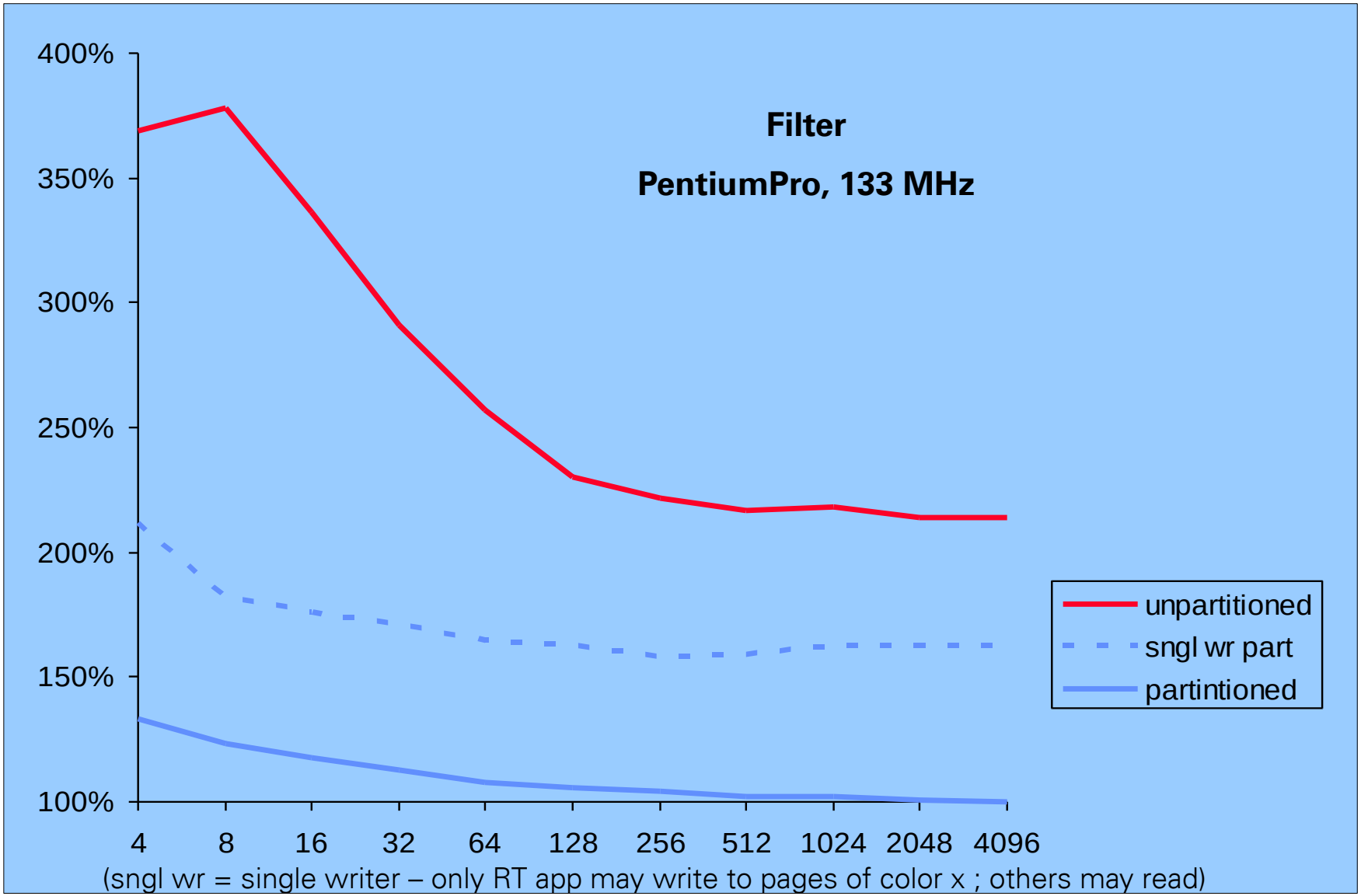
- other tasks in the backgound (e.g., cache flodder)

Example: Filter

- Worst case:
  - wo. cache partitioning:
    background tasks may evict all cachelines
    background tasks may load conflicting cachelines, which need writeback



OS Controlled Cache Partitioning (6)

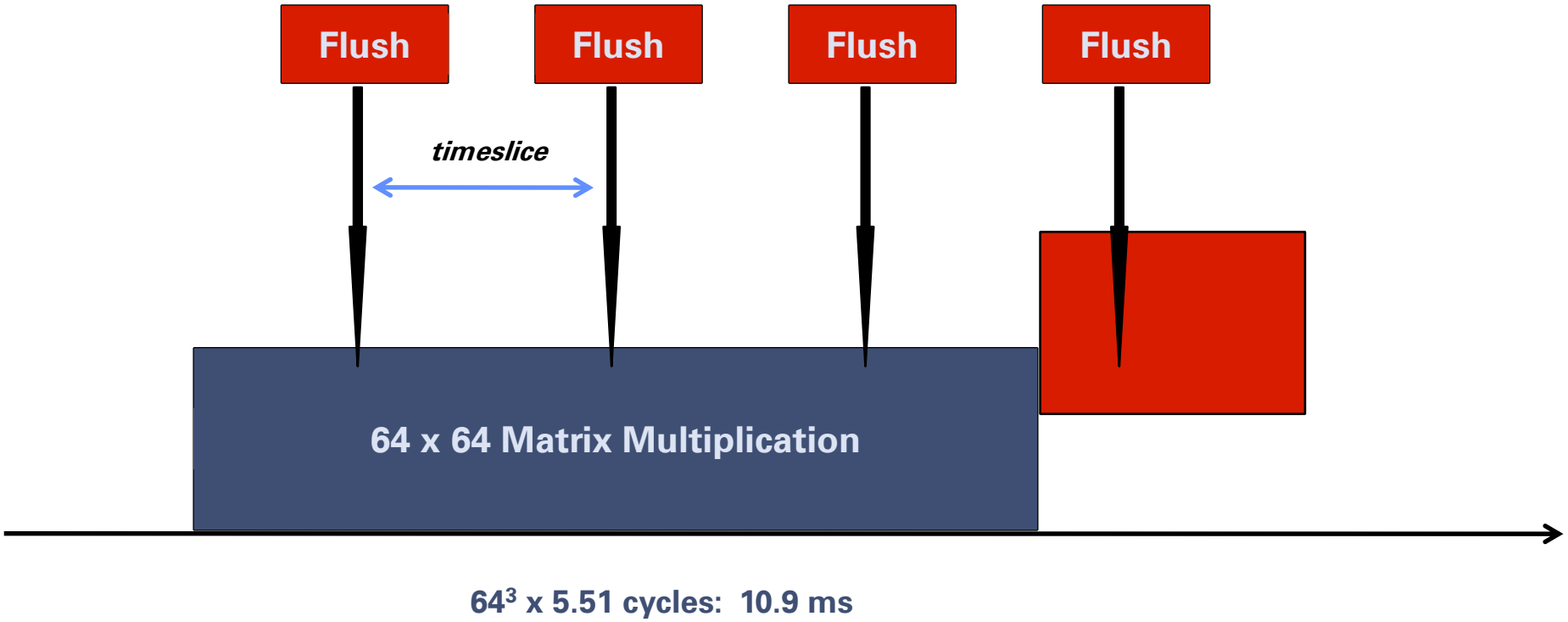


OS Controlled Cache Partitioning (7)

Example 2:

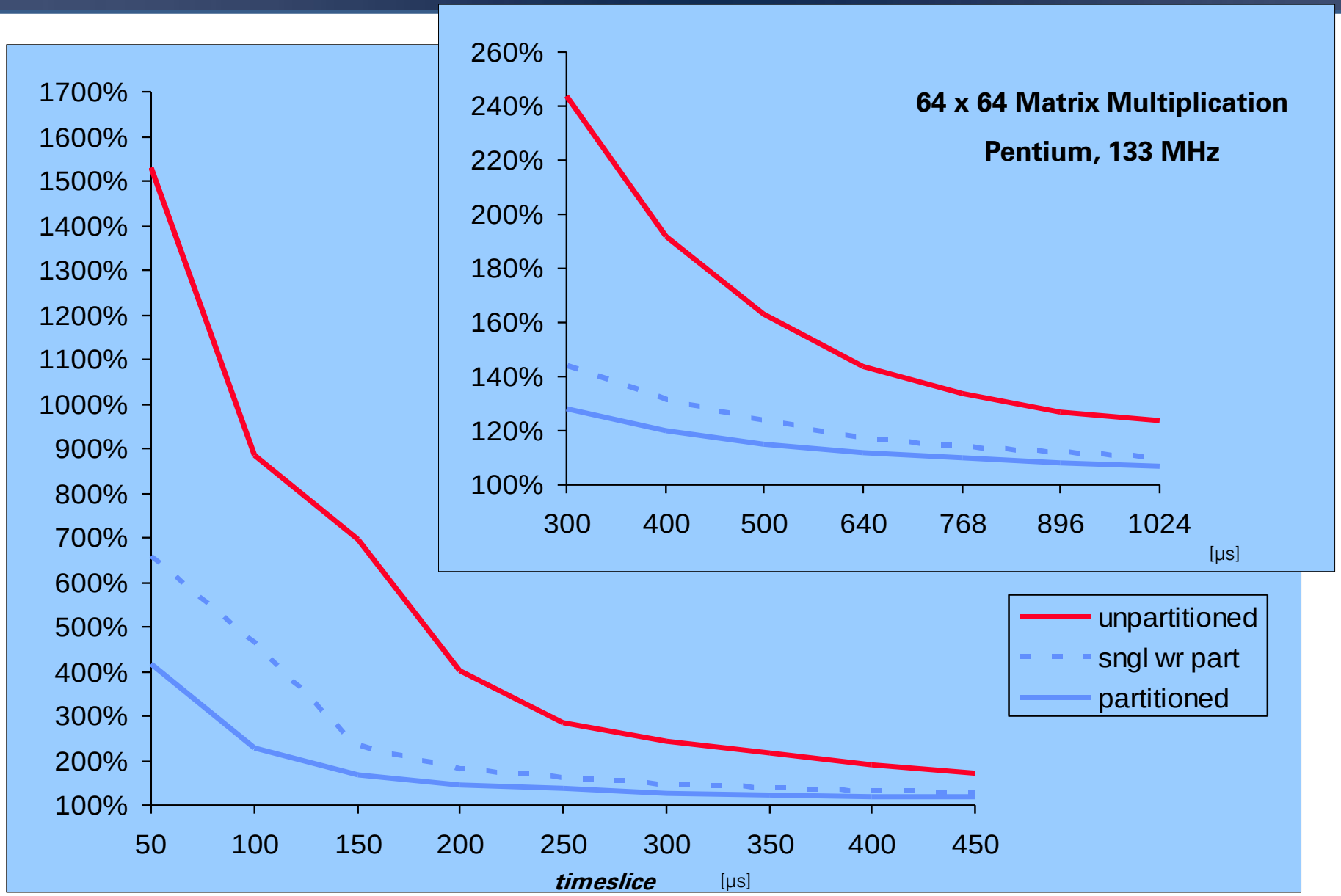
low priority task; frequently interrupted

e.g., matrix multiplication

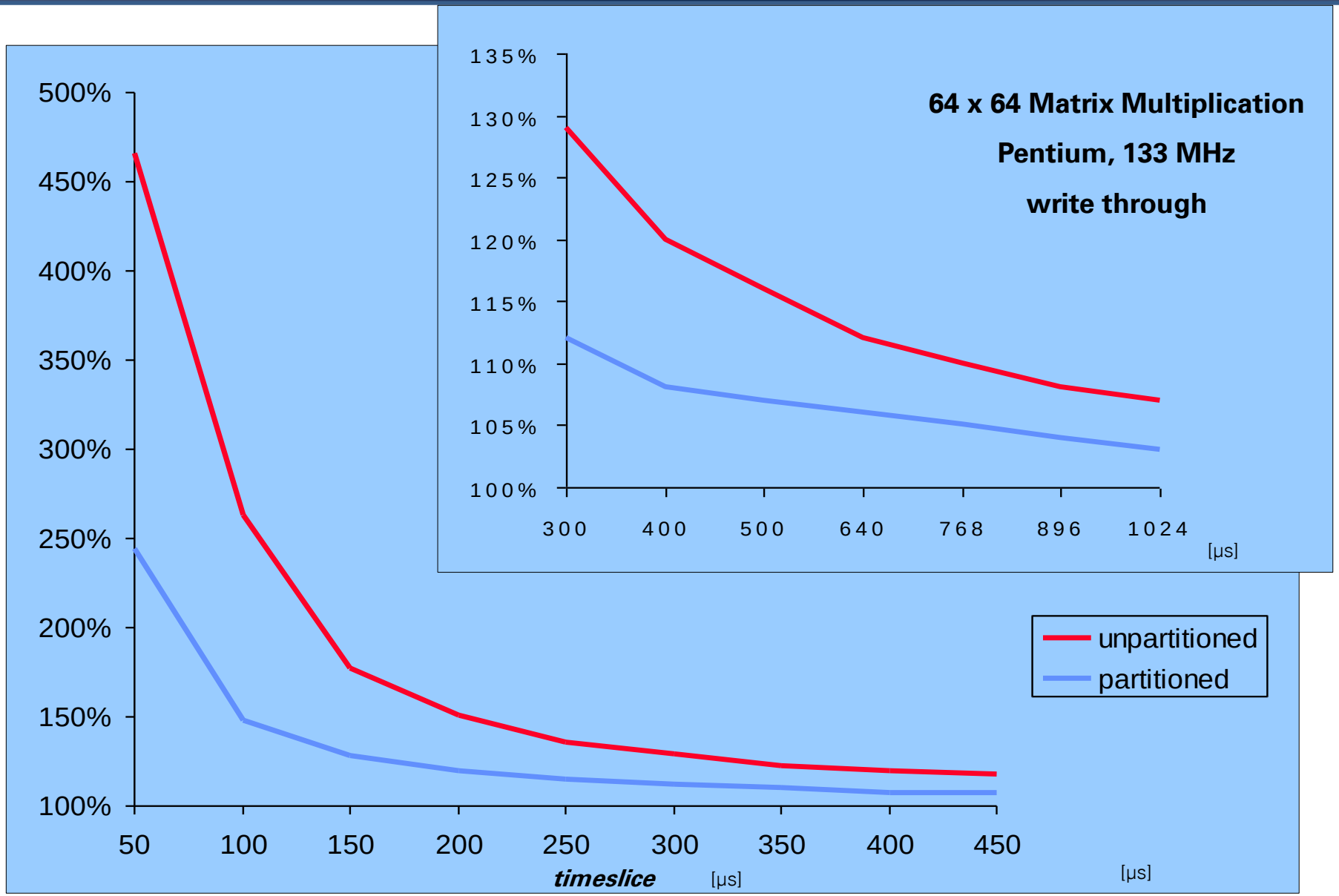


64^3 x 5.51 cycles: 10.9 ms

OS Controlled Cache Partitioning (8)



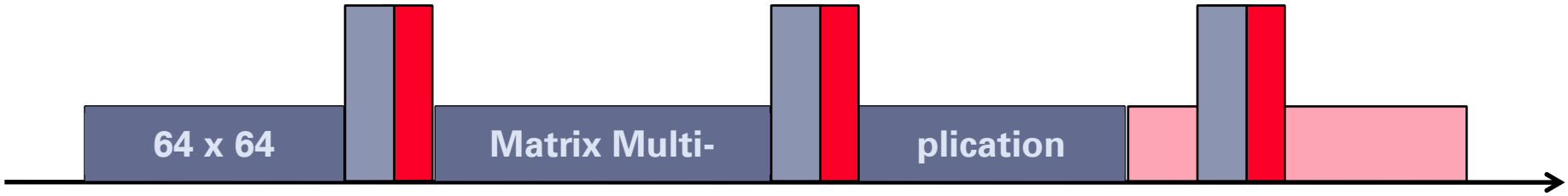
OS Controlled Cache Partitioning (9)

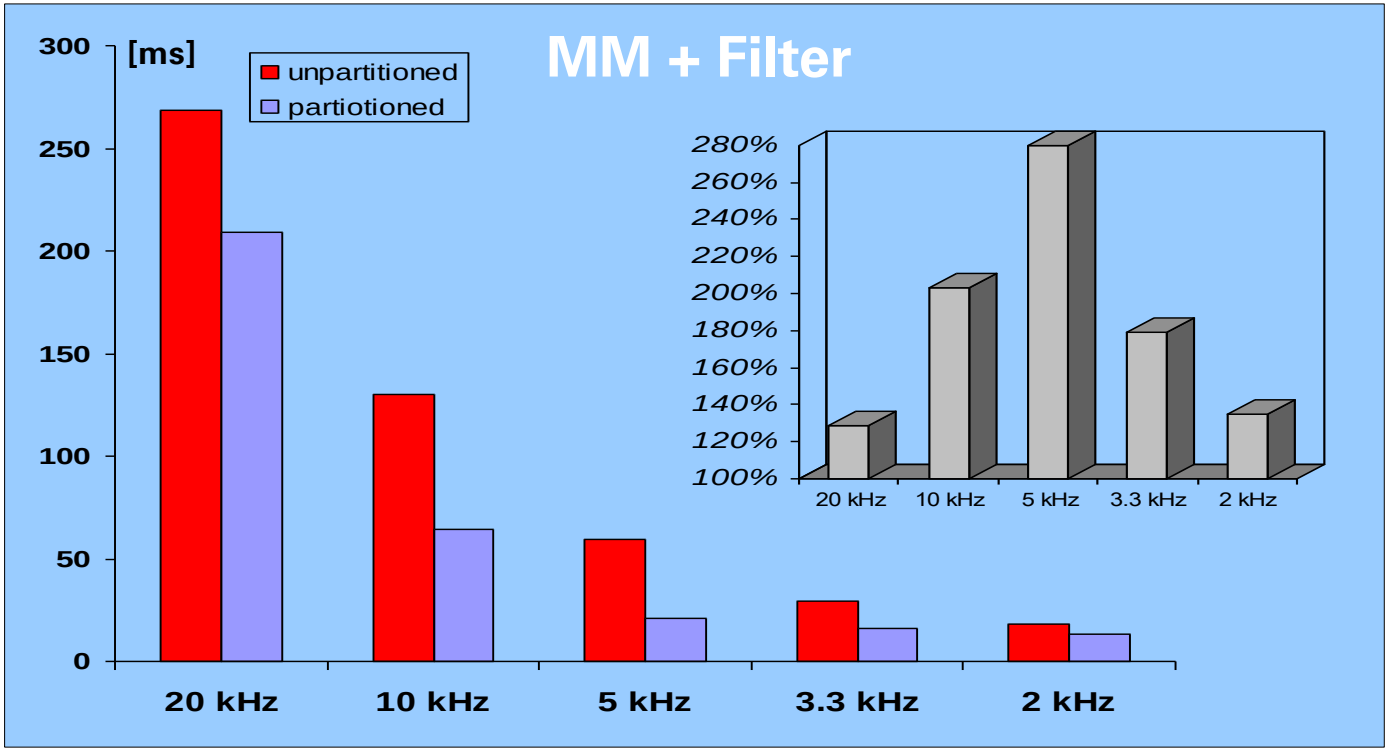


OS Controlled Cache Partitioning (10)

Experiment 3:

Combination: Matrix Multiplication + Filter





OS Controlled Cache Partitioning (11)

Caveat: application transparency

some applications require knowledge / control of physical addresses

e.g., drivers need physical addresses for direct memory address transfers (DMA)

- modify driver
- use recent hardware (e.g., Intel VT-d) with address translation for DMA

Conclusions

- High performance CPUs are rarely used in embedded systems
  - HW means to increase predictability
  - SW tweaks to use unpredictable HW in a more predictable way

- **References:**
  - Liedtke, Härtig, Hohmuth (RTAS '97):
    Operating system control cache predictability for real-time applications
  - ARM Real View Emulation Board User Guide
  - ARM 1176jzfs Manual
  - SAB 80C166 Handbook