Real-Time Systems

Hermann Härtig, Marcus Völp

Hardware

TECHNISCHE UNIVERSITÄT DRESDEN

Outline

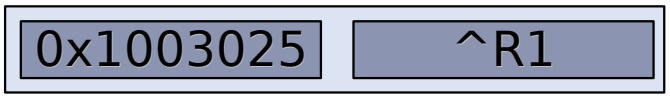
- Hardware: Source of Unpredictability
  - Caches
  - Pipeline
  - Interrupt Latency
  - System Management Mode
- Special Purpose Hardware for “Embedded” Real-Time Systems
- Use unpredictable Hardware more predictably

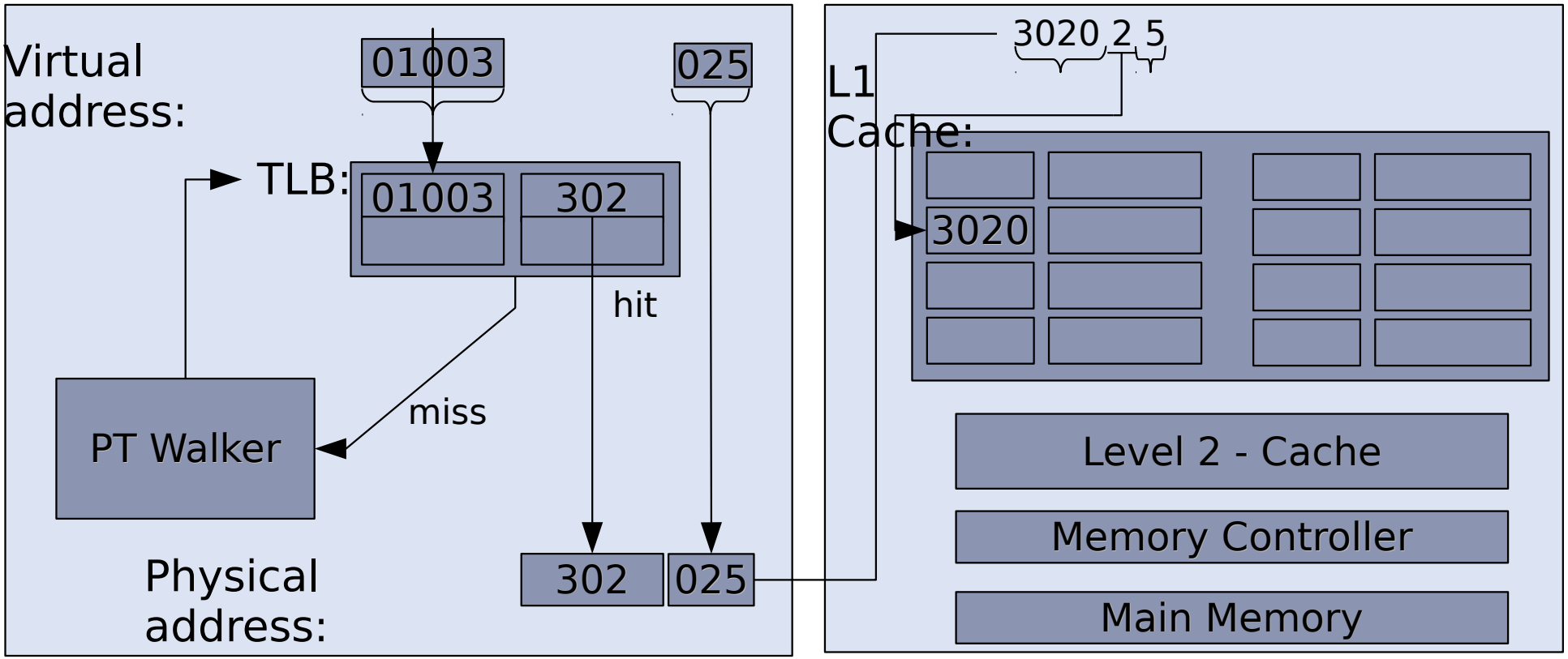
- Real-Time Communication / Buses in separate Lecture

Sources of Unpredictability

- Memory Subsystem
  - TLB
  - Caches
  - Store Buffers

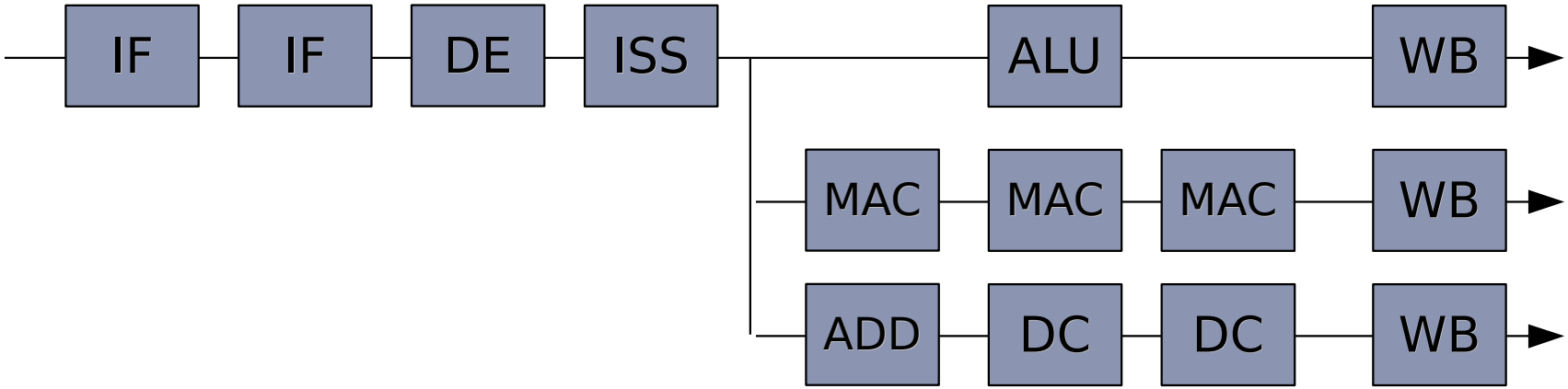
store R1, 0x01003025





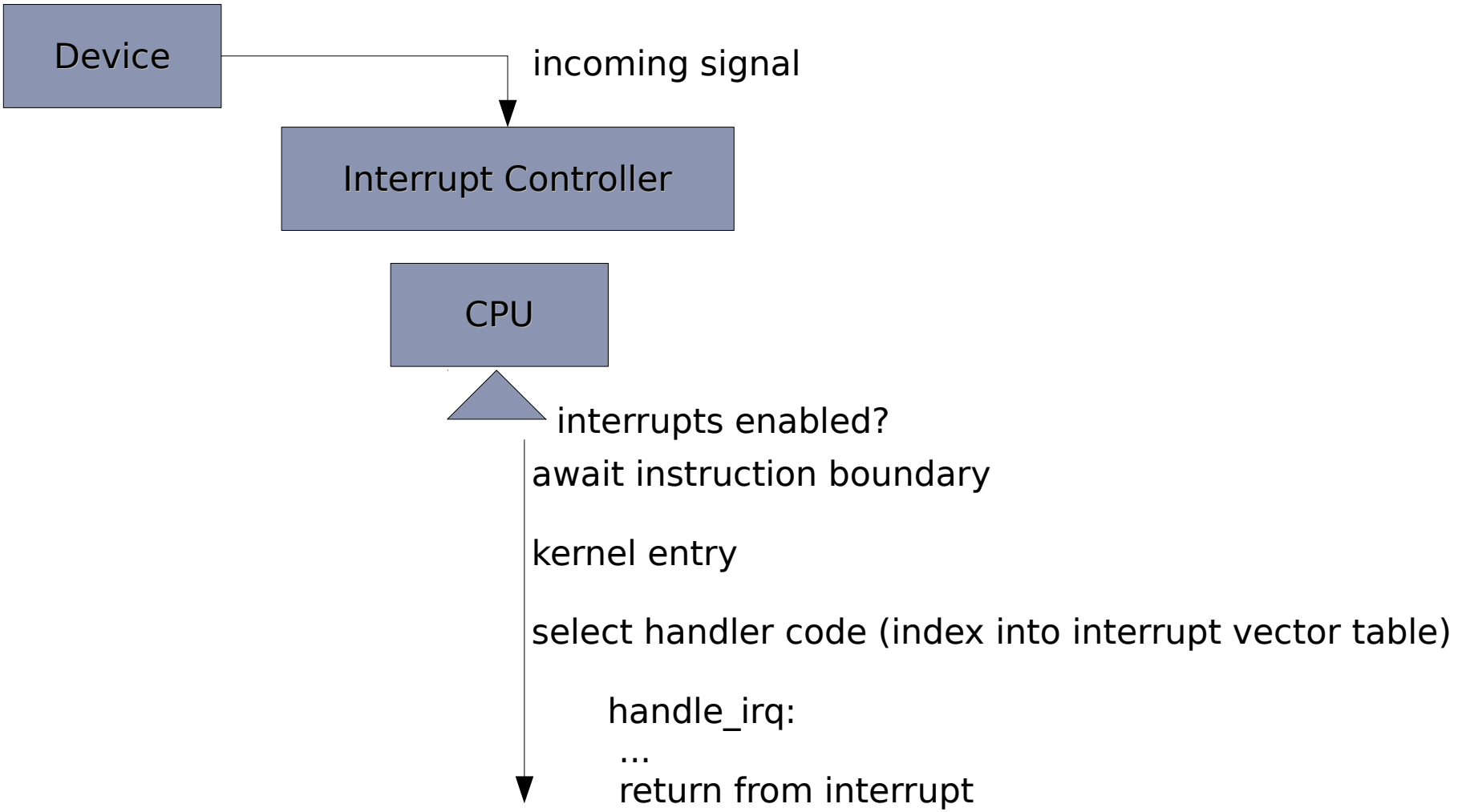
Sources of Unpredictability

- Processor Pipeline (ARM 11)

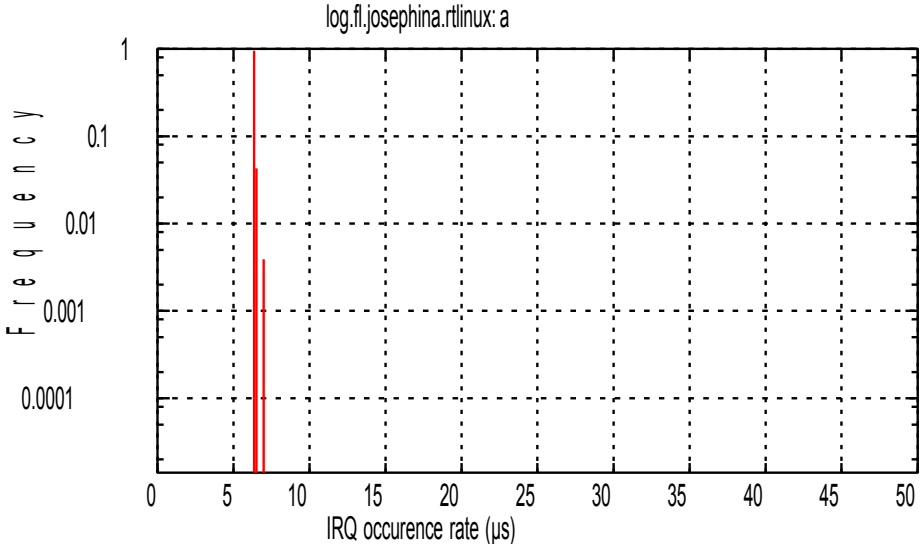


Sources of Unpredictability

- Interrupt Latency

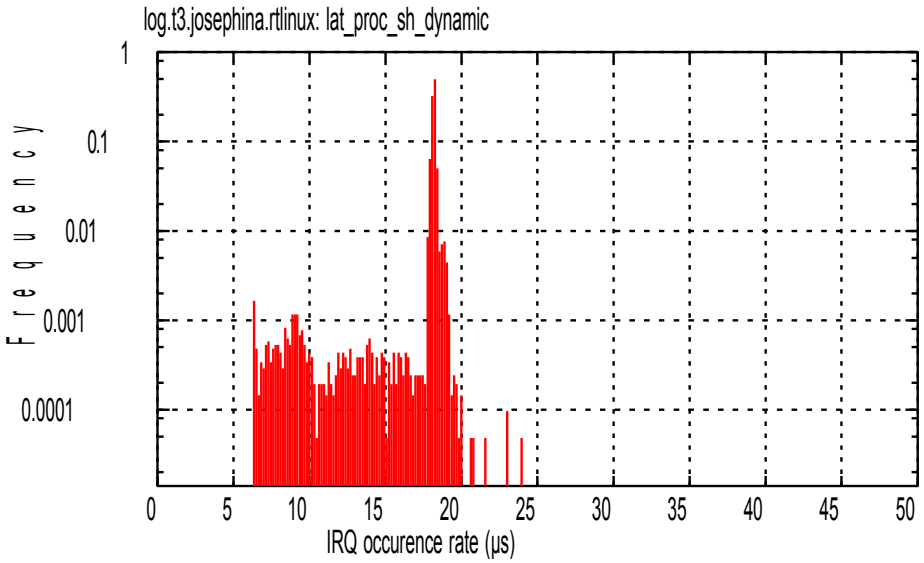


Interrupt Response Time – RTLinux



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

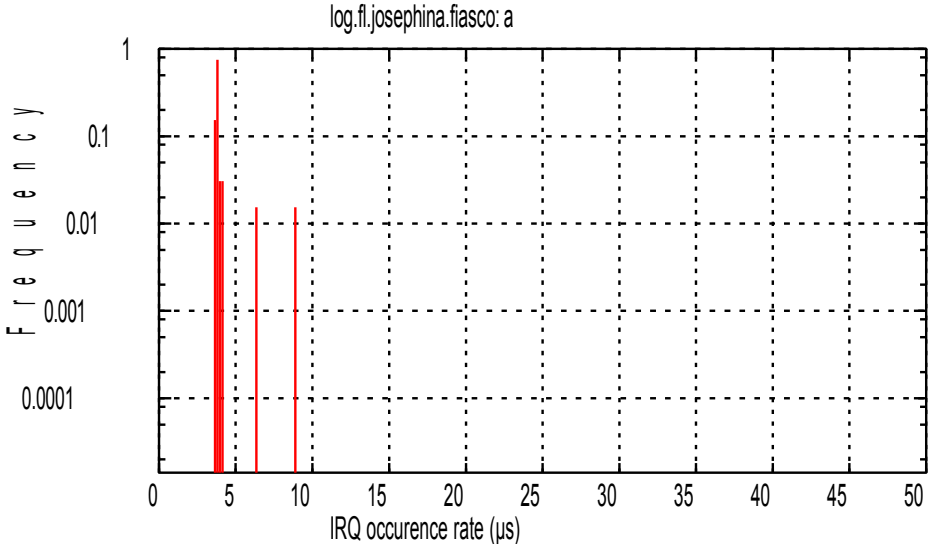
No parallel load (idle): **13 µs**

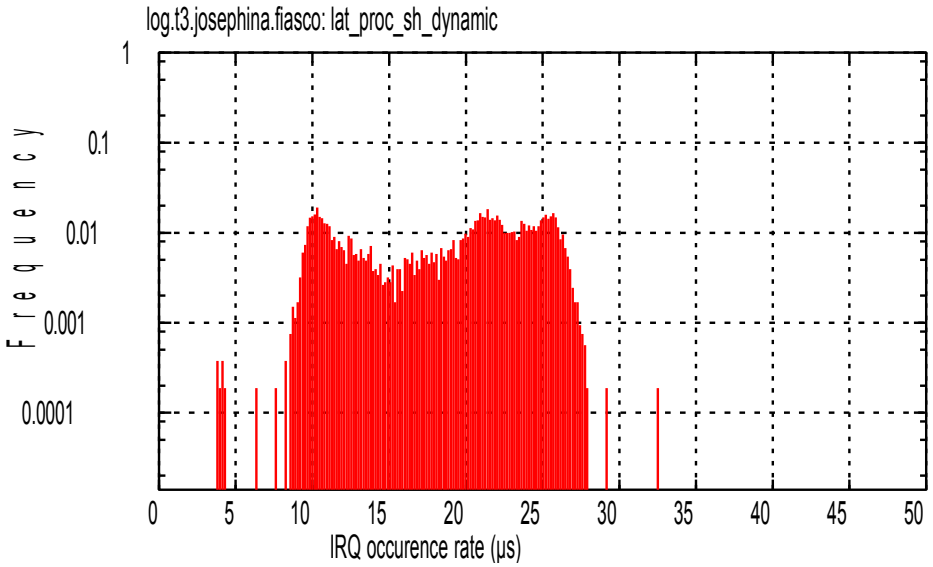


High parallel load: **68 µs**
(Benchmark, Cache-Flooder)

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-Flooder)

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
- How to use unpredictable Hardware more predictably

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

- Two 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, recommendations:
    - 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 “counter==0”
  - byte / word select — select width of transfer (byte or word)
  - source / dest — increase update source / destination address
  - source / dest — address

- **Semantics expressed as Signal-Handler:**

```
signal () {
   if (counter--)
      *dest++ = *src;
   else
      trigger_interrupt();
}
```

=> minimal response time (~3 cycles)

=> PEC can execute at external clock rate

Example: SAB 80C166 (Siemens, 1990)

Timestamps / Event triggers

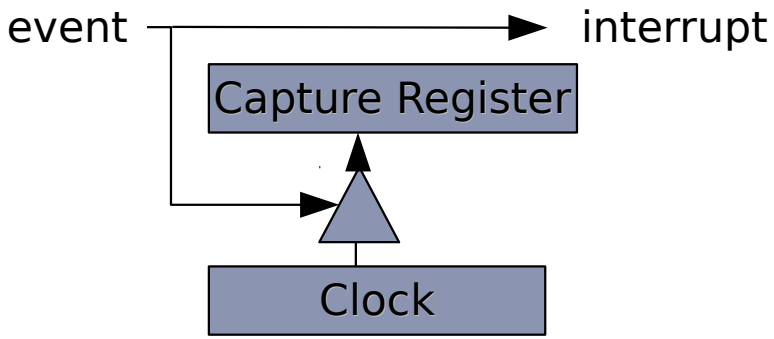
- Precise timestamps / trigger times with interrupt-based sampling is difficult
  - system load influences jitter in interrupt response time
  - atomic sections in OS delay interrupts
  - jitter in signal delivery (from signal source to CPU interrupt pin)
    - buses, interrupt controller

Capture + Compare Unit

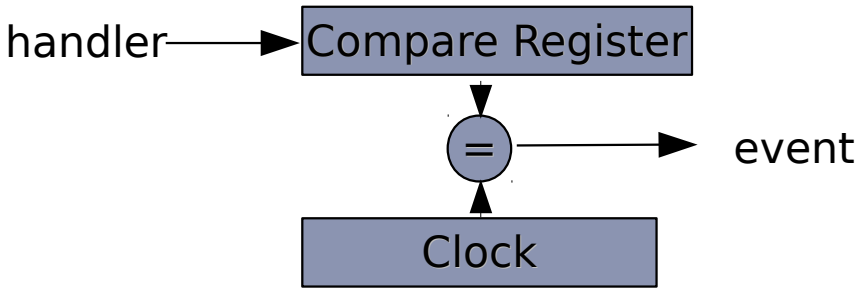
- Problem
  - precise timestamp of event
    (engine sensor triggered at cylinder position = t)
  - trigger events at precise points in time
    (fire the engine at $t + x$)

  - interrupt handlers' jitter too high 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

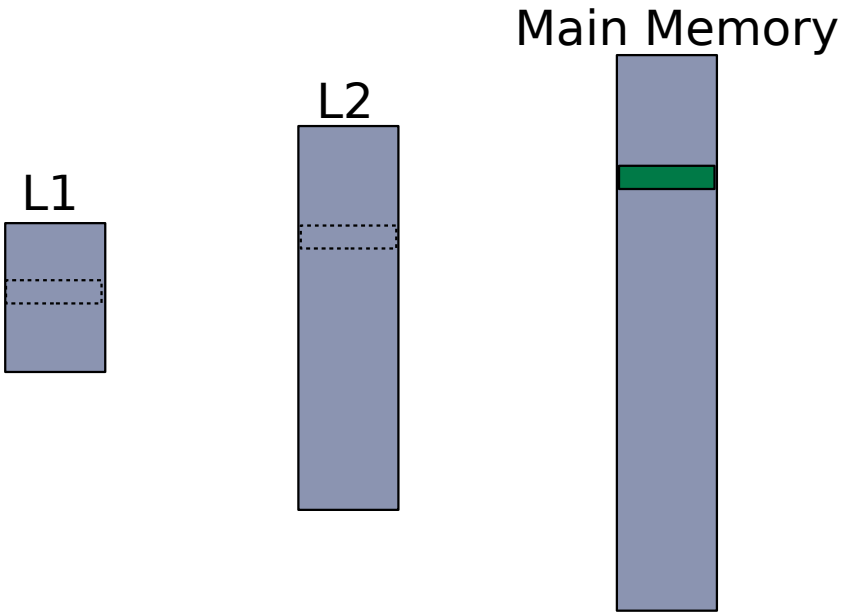
System on Chip / Chip Multiprocessor

- 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

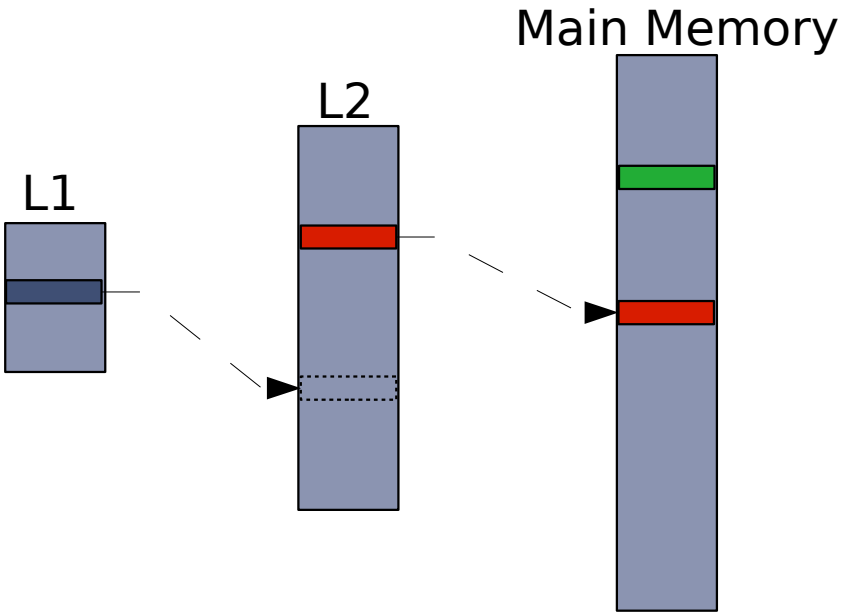
Scratch-pad Memory vs. Caches

- 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



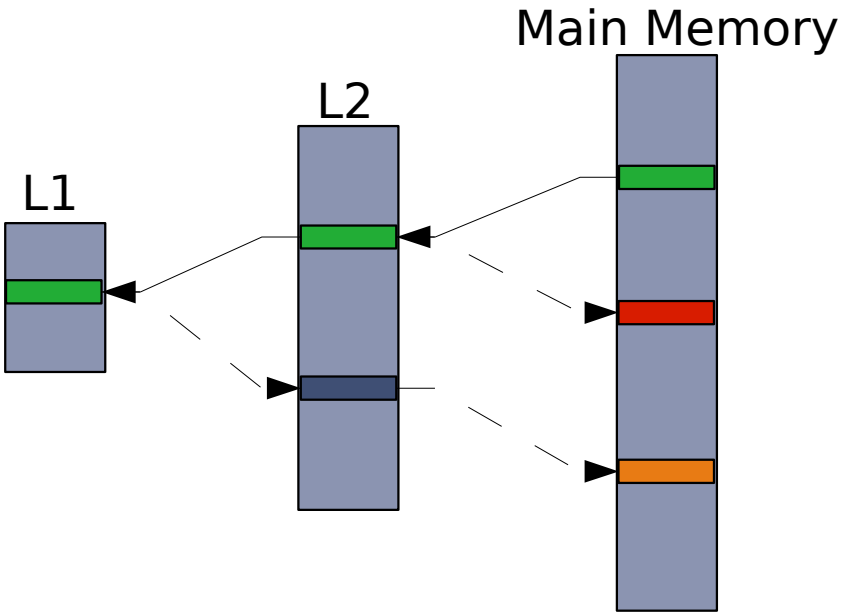
Scratch-pad Memory vs. Caches (2)

- 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



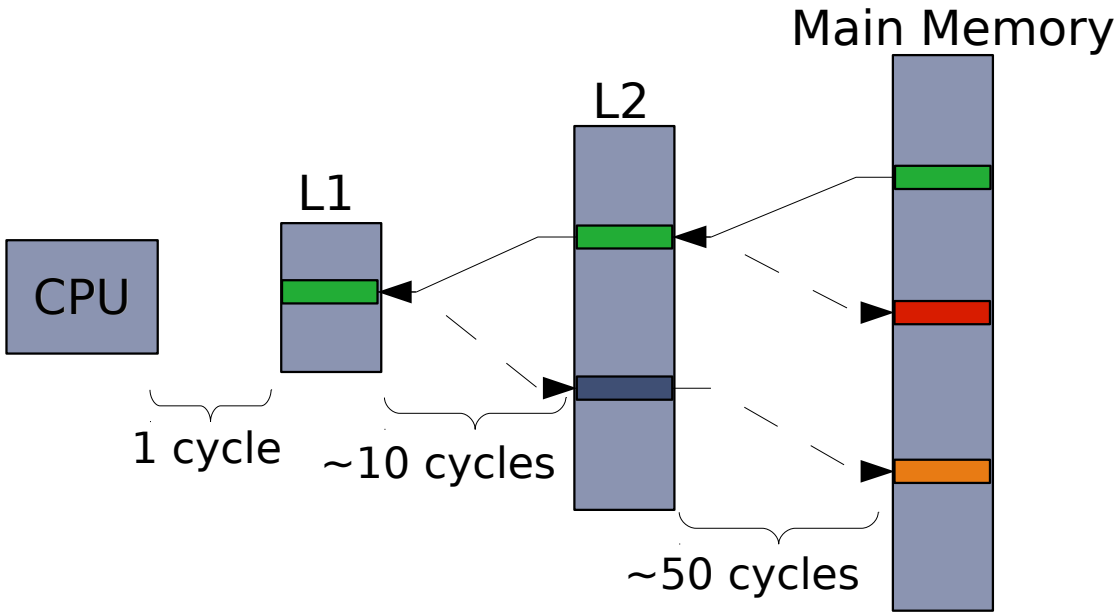
Scratch-pad Memory vs. Caches (3)

- 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



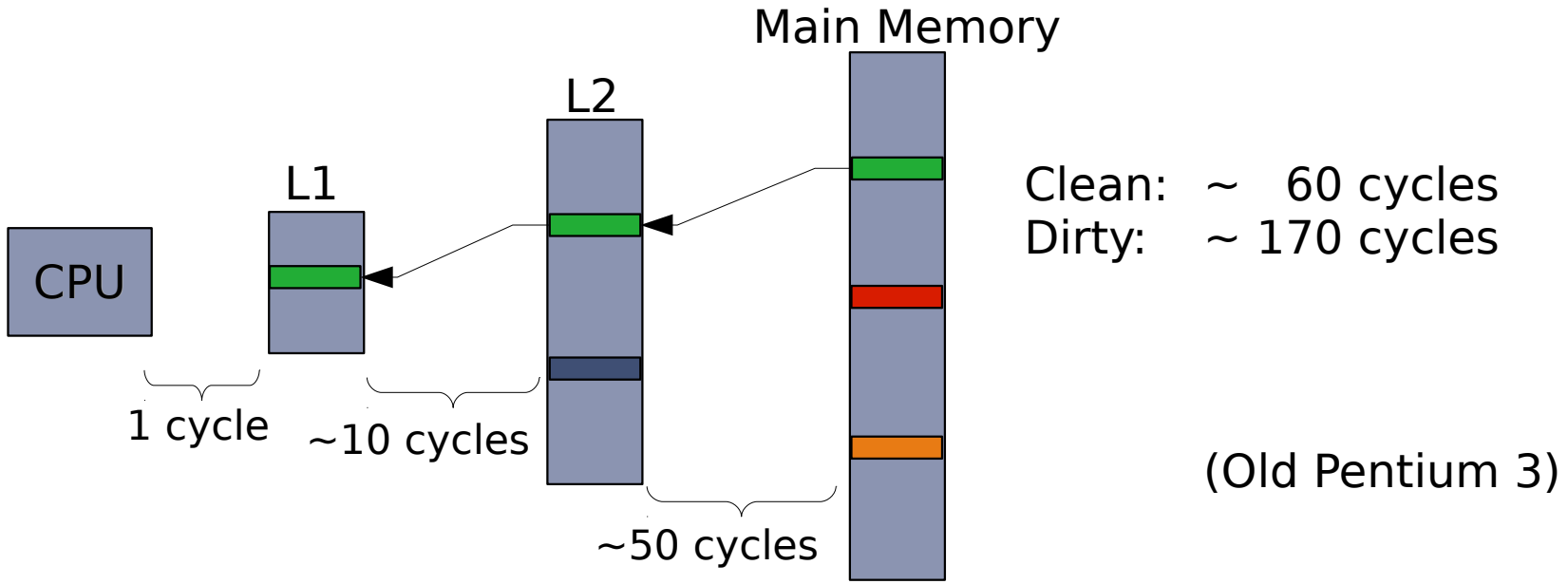
Scratch-pad Memory vs. Caches (4)

- 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)

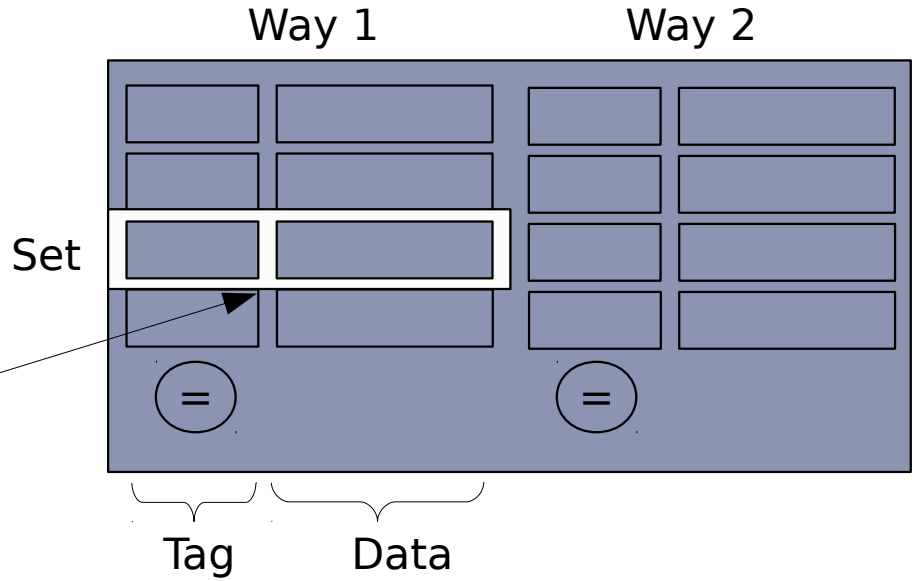
- Synonyms:
  [Scratchpad / On-Chip / Tightly-Coupled] Memory
- Cache:
  - transparent addressing scheme
  - cache controller fills / replaces cachelines in parallel to CPU activity
  - difficult to predict worst-case execution time



Scratch-pad Memory vs. Caches (6)

Address example needed ..... for explanation

Terminology:
* set
* tag
* way
* cache line



Cache lock:

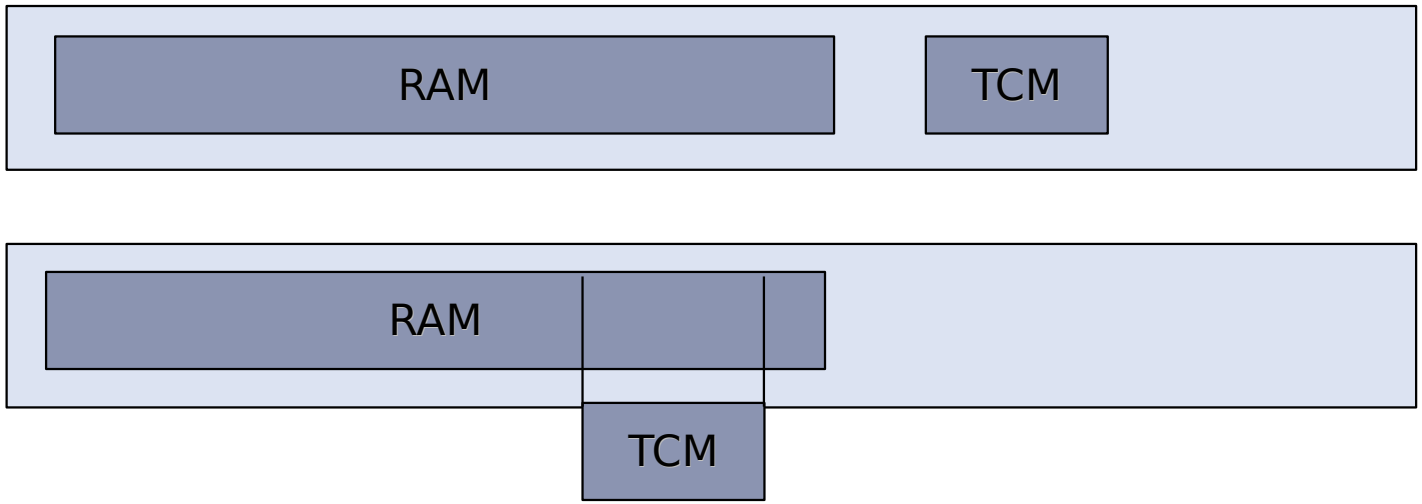
- Lock complete cache way
- Allocate to unlocked cache ways only

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
    - Which 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

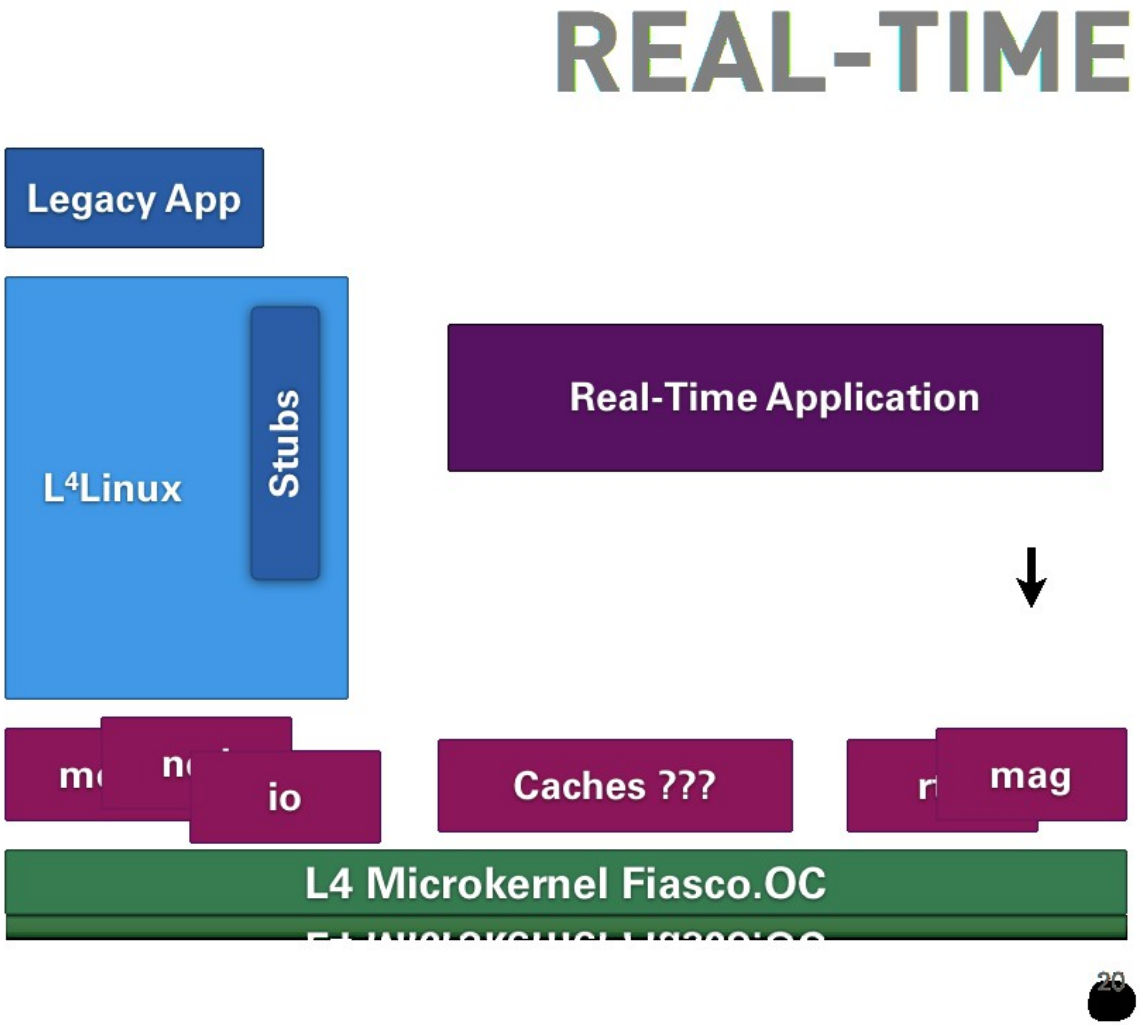
- Hardware is Source of Unpredictability
- Special Purpose Hardware for “Embedded” Real-Time Systems
- Use unpredictable Hardware more predictable
  - Cache Partitioning
- Real-Time Communication / Buses in separate Lecture

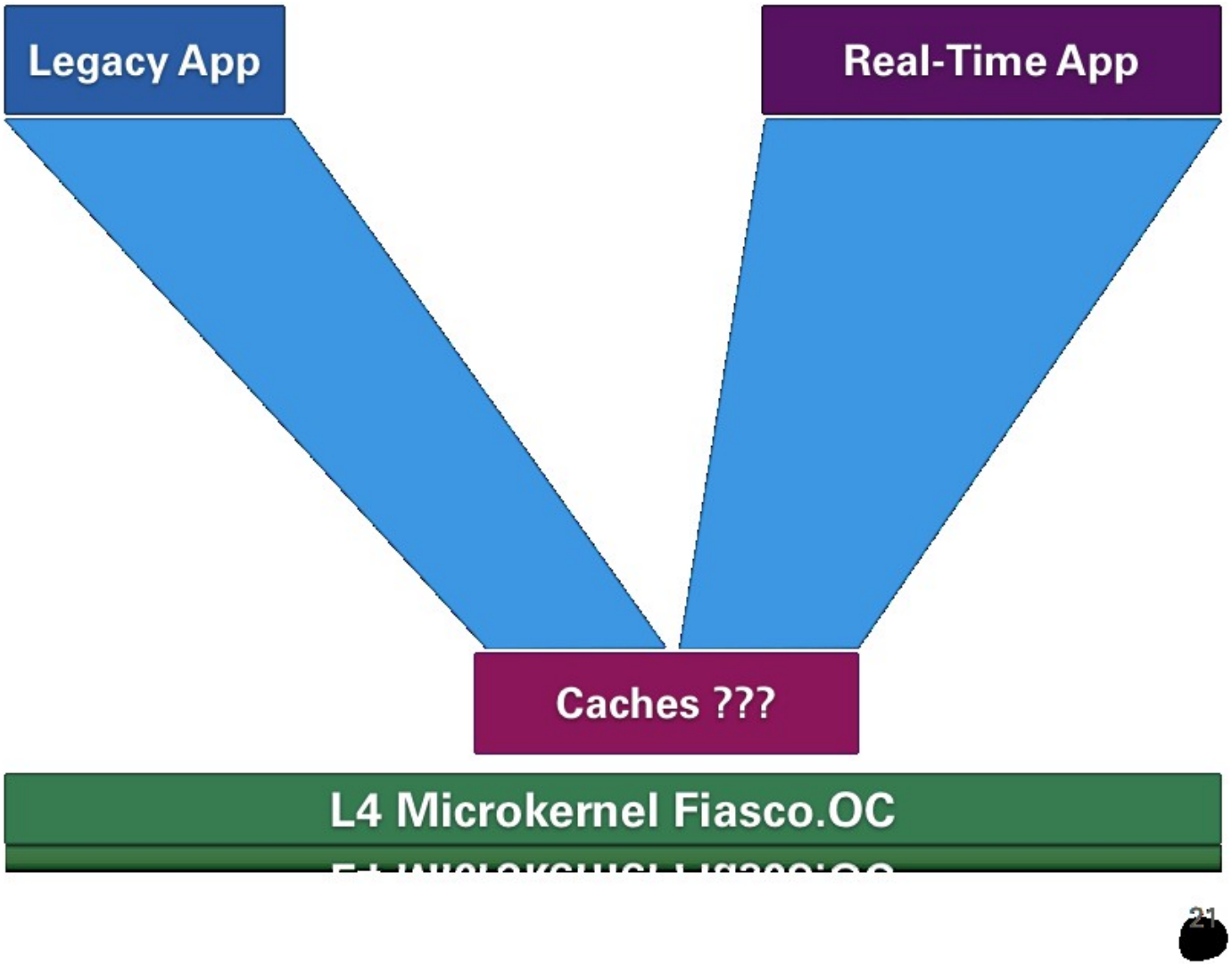
(OS-Controlled) Cache Partitioning

- 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

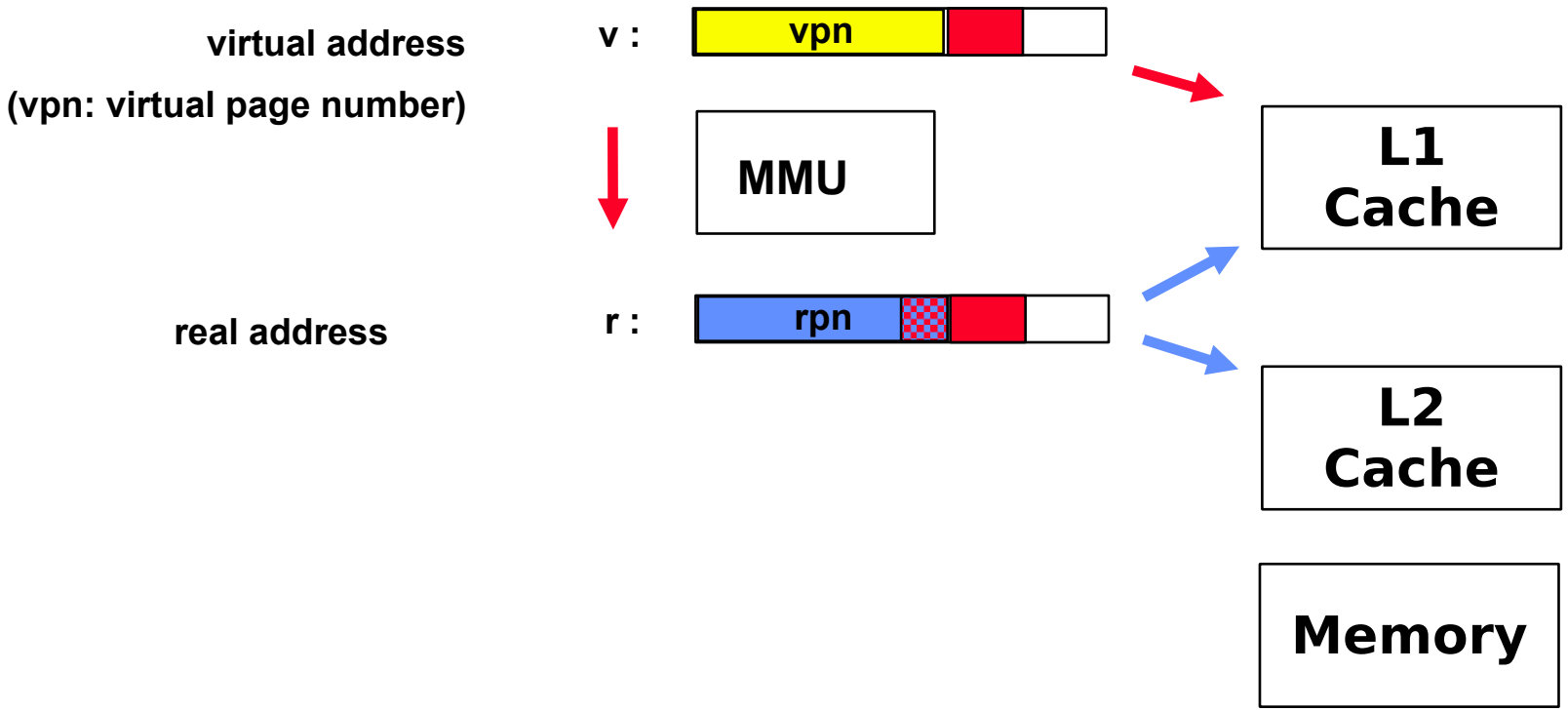
      isolates malicious, erroneous applications

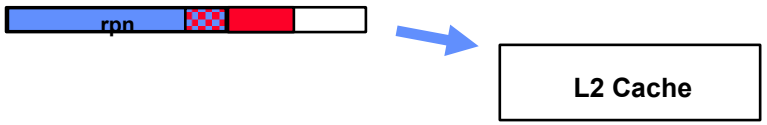
RT and Non-RT

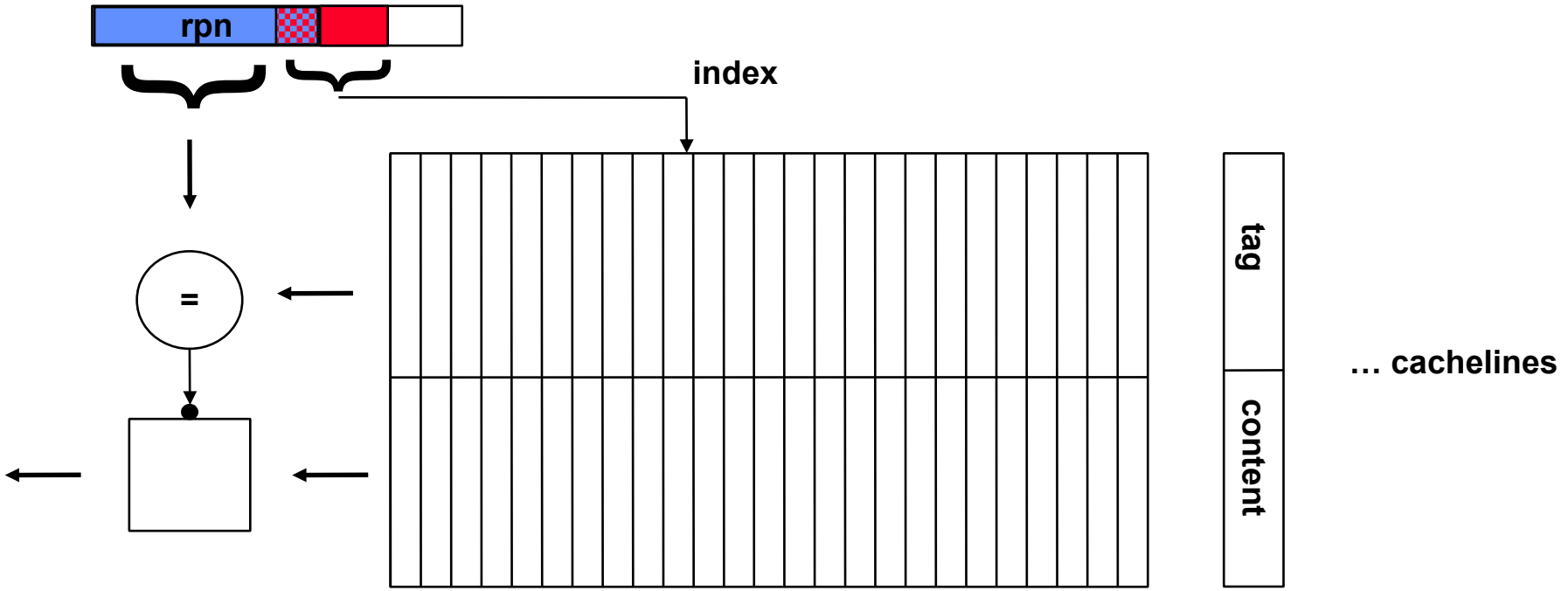

REAL-TIME
Legacy App
L4Linux
Stubs
Real-Time Application
io
Caches ???
mag
L4 Microkernel Fiasco.OC

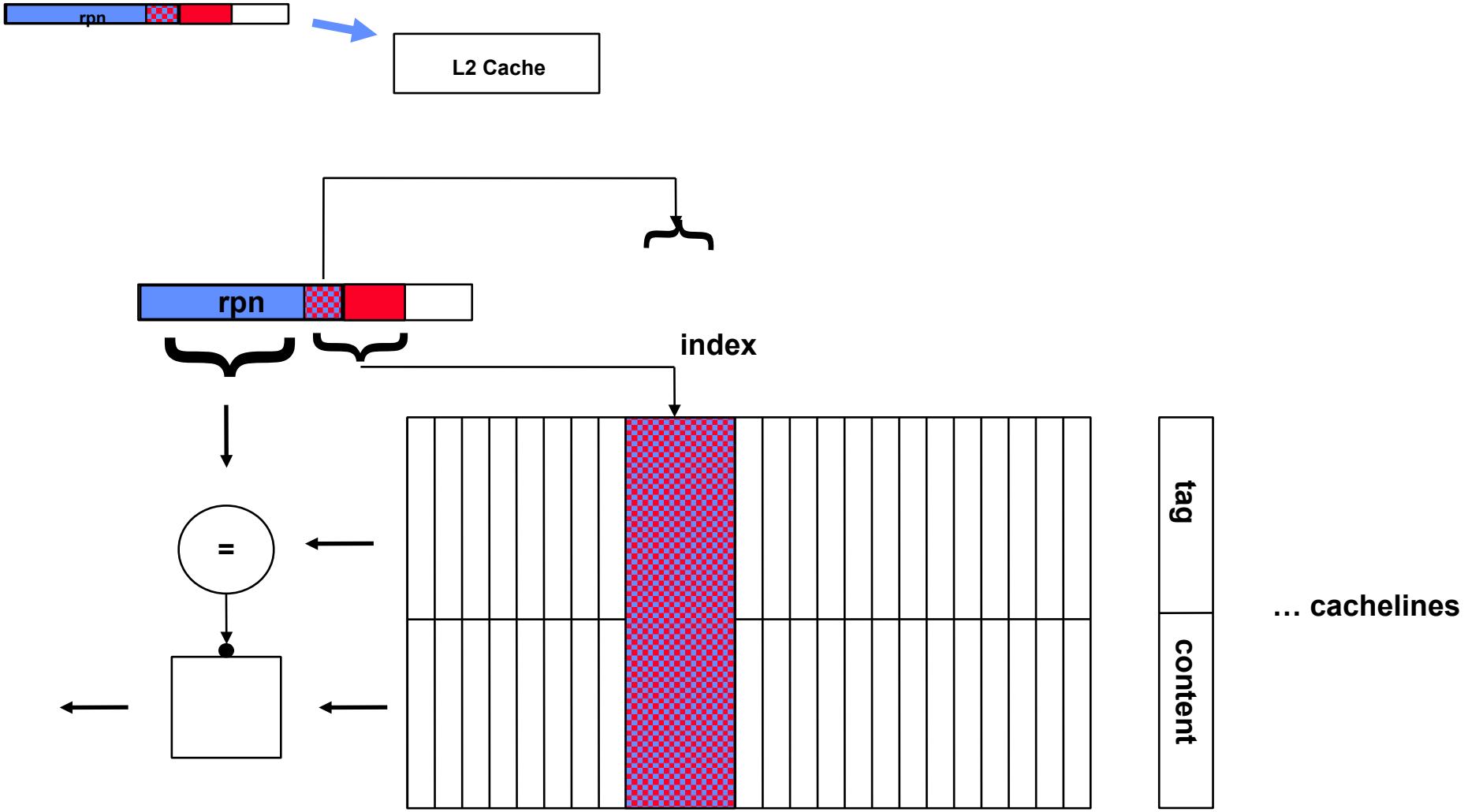

Legacy App
Real-Time App
Caches ???
L4 Microkernel Fiasco.OC
21

Cache Partitioning

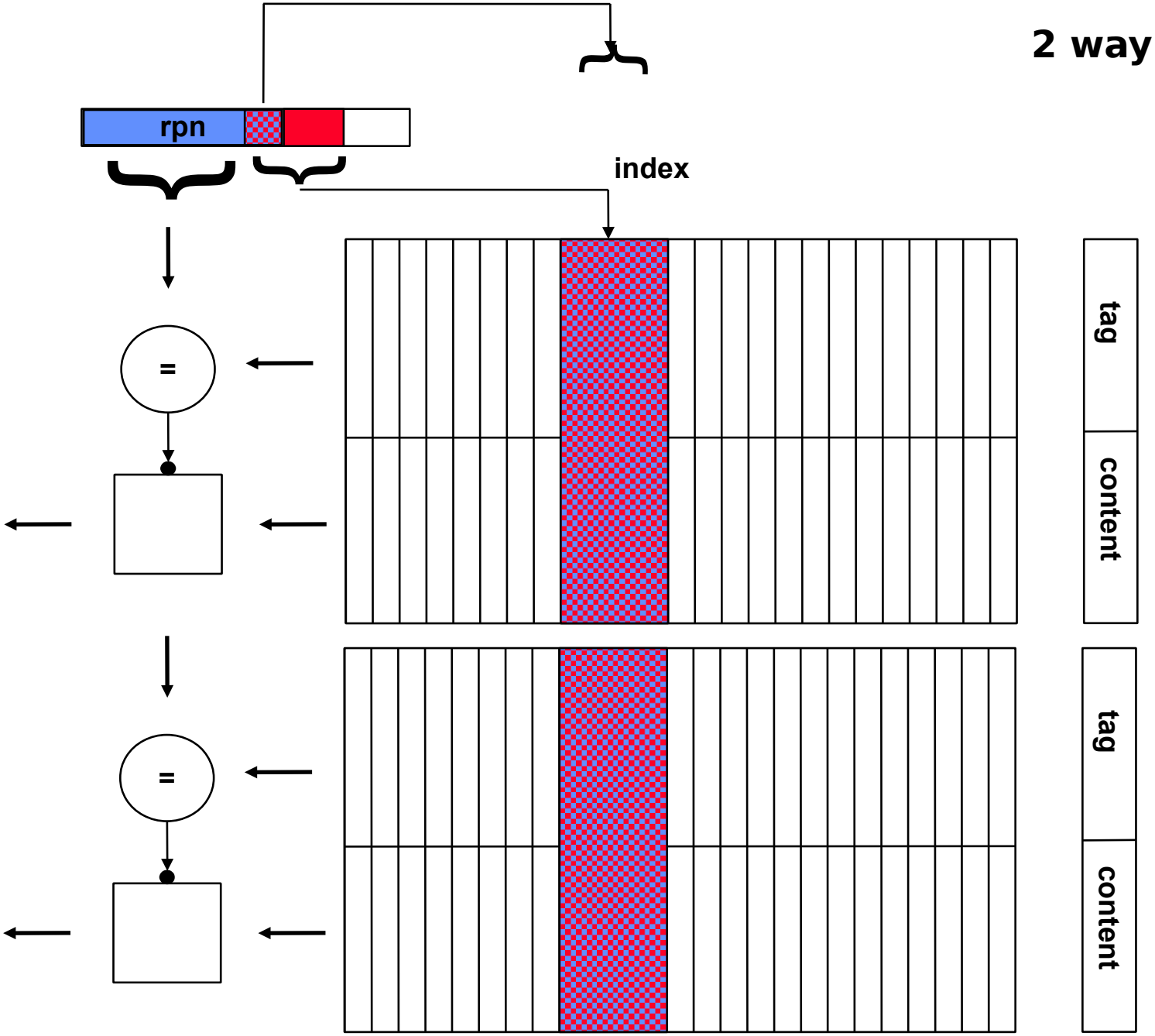


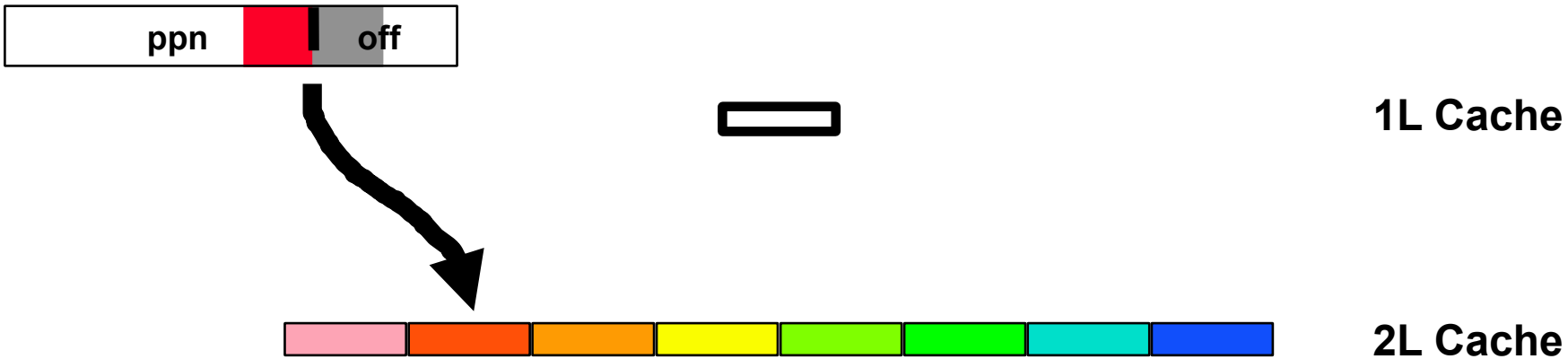

rpn
L2 Cache


rpn
index
=
tag
content
… cachelines


rpn
L2 Cache
rpn
index
=
tag
content
… cachelines

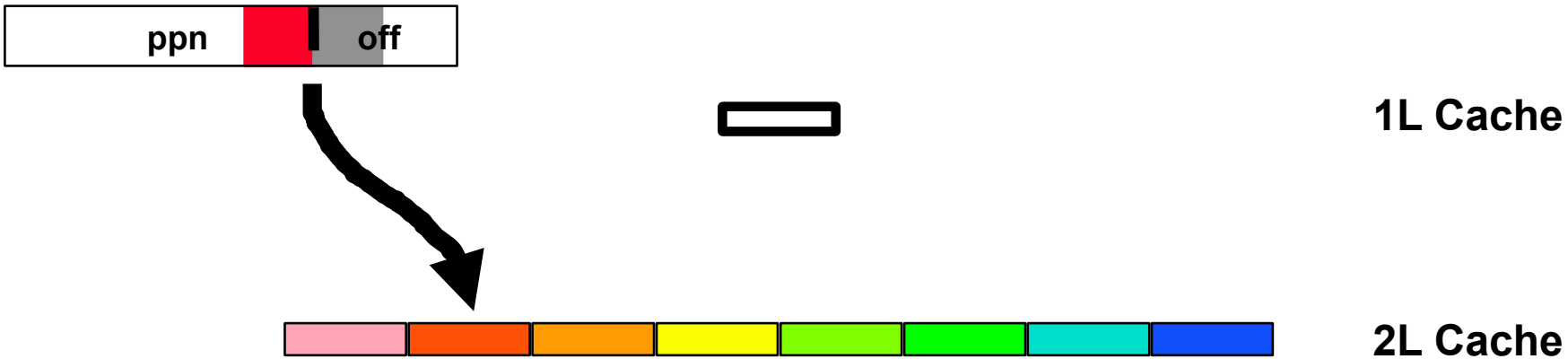
2 way associativity





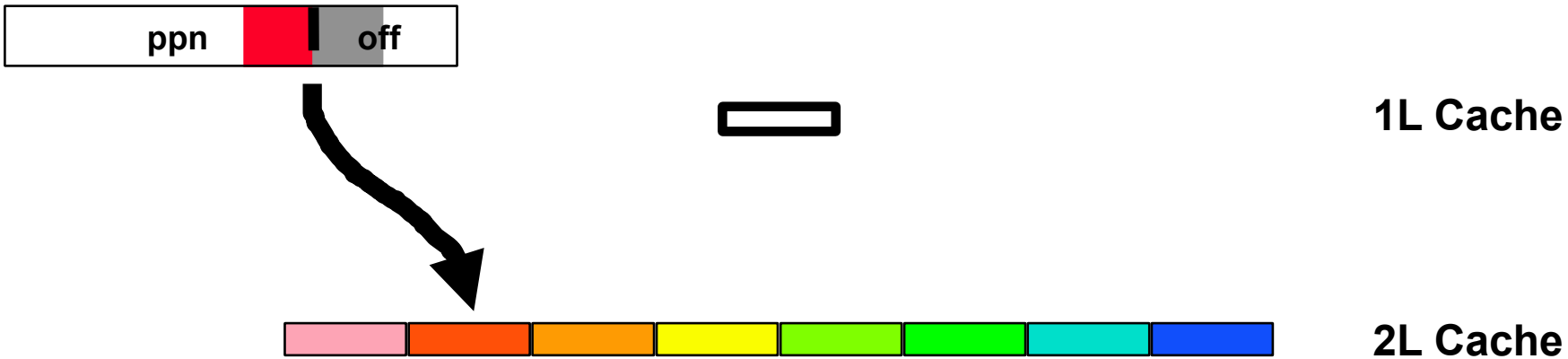
000000	001000	002000	003000	004000	005000	006000	007000
010000	011000	012000	013000	014000	015000	016000	017000
020000	021000	022000	023000	024000	025000	026000	027000
030000	031000	032000	033000	034000	035000	036000	037000
040000	041000	042000	043000	044000	045000	046000	047000
050000	051000	052000	053000	054000	055000	056000	057000
060000	061000	062000	063000	064000	065000	066000	067000
070000	071000	072000	073000	074000	075000	076000	077000
100000	101000	102000	103000	104000	105000	106000	107000
110000	111000	112000	113000	114000	115000	116000	117000
120000	121000	122000	123000	124000	125000	126000	127000
130000	131000	132000	133000	134000	135000	136000	137000
140000	141000	142000	143000	144000	145000	146000	147000
150000	151000	152000	153000	154000	155000	156000	157000
160000	161000	162000	163000	164000	165000	166000	167000
170000	171000	172000	173000	174000	175000	176000	177000

Main Memory



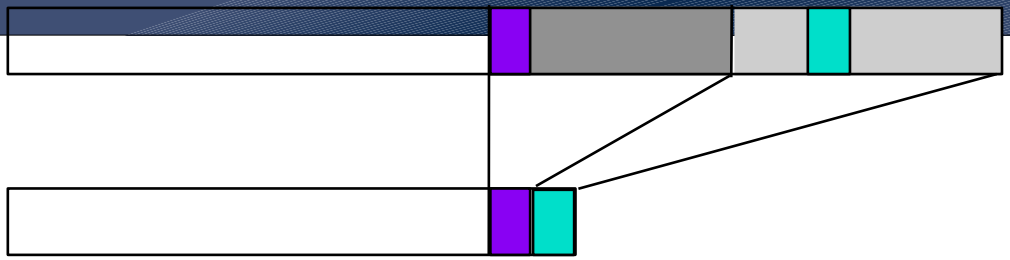
000000	001000	002000	003000	004000	005000	006000	007000
010000	011000	012000	013000	014000	015000	016000	017000
020000	021000	022000	023000	024000	025000	026000	027000
030000	031000	032000	033000	034000	035000	036000	037000
040000	041000	042000	043000	044000	045000	046000	047000
050000	051000	052000	053000	054000	055000	056000	057000
060000	061000	062000	063000	064000	065000	066000	067000
070000	071000	072000	073000	074000	075000	076000	077000
100000	101000	102000	103000	104000	105000	106000	107000
110000	111000	112000	113000	114000	115000	116000	117000
120000	121000	122000	123000	124000	125000	126000	127000
130000	131000	132000	133000	134000	135000	136000	137000
140000	141000	142000	143000	144000	145000	146000	147000
150000	151000	152000	153000	154000	155000	156000	157000
160000	161000	162000	163000	164000	165000	166000	167000
170000	171000	172000	173000	174000	175000	176000	177000

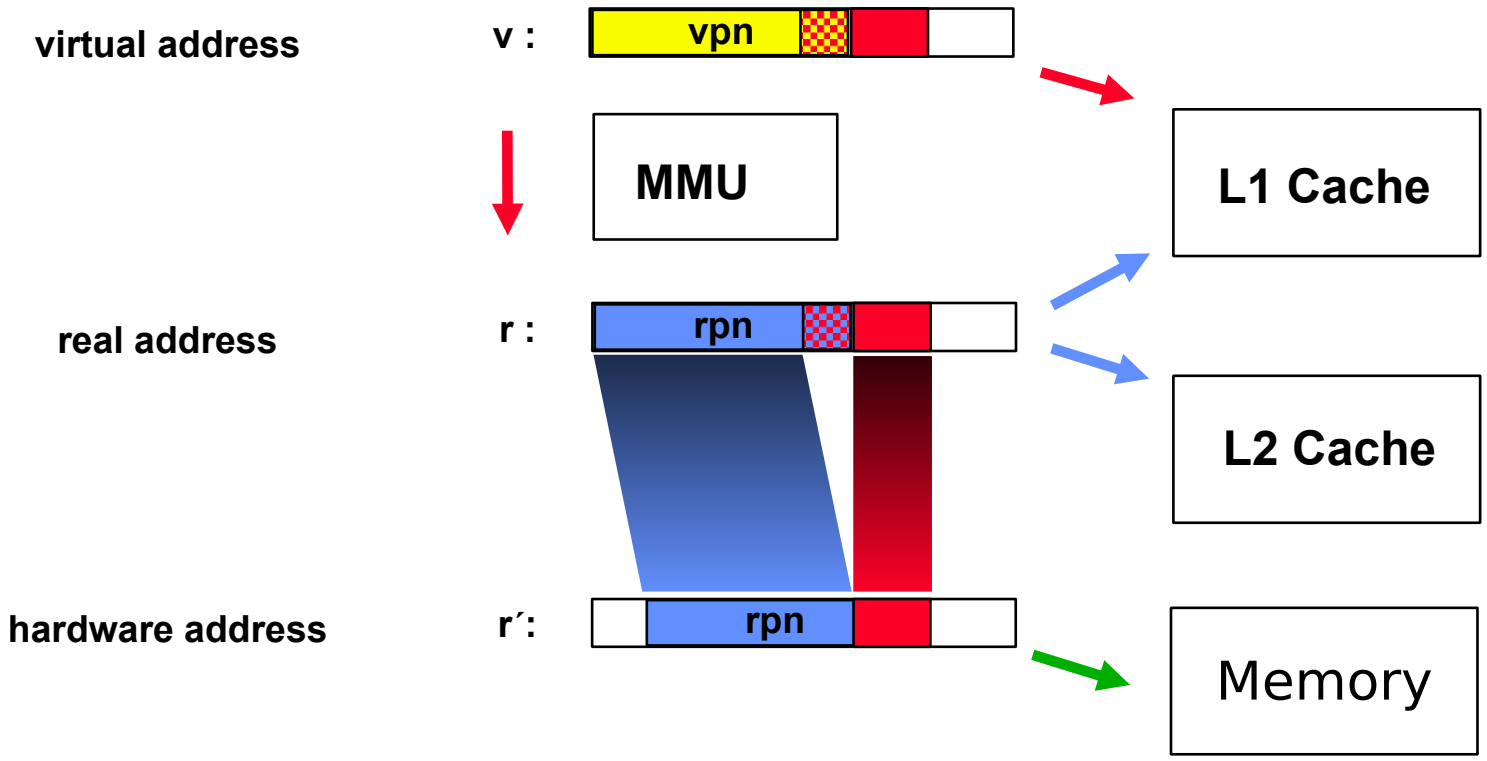
Main Memory



000000	001000	002000	003000	004000	005000	006000	007000
010000	011000	012000	013000	014000	015000	016000	017000
020000	021000	022000	023000	024000	025000	026000	027000
030000	031000	032000	033000	034000	035000	036000	037000
040000	041000	042000	043000	044000	045000	046000	047000
050000	051000	052000	053000	054000	055000	056000	057000
060000	061000	062000	063000	064000	065000	066000	067000
070000	071000	072000	073000	074000	075000	076000	077000
100000	101000	102000	103000	104000	105000	106000	107000
110000	111000	112000	113000	114000	115000	116000	117000
120000	121000	122000	123000	124000	125000	126000	127000
130000	131000	132000	133000	134000	135000	136000	137000
140000	141000	142000	143000	144000	145000	146000	147000
150000	151000	152000	153000	154000	155000	156000	157000
160000	161000	162000	163000	164000	165000	166000	167000
170000	171000	172000	173000	174000	175000	176000	177000

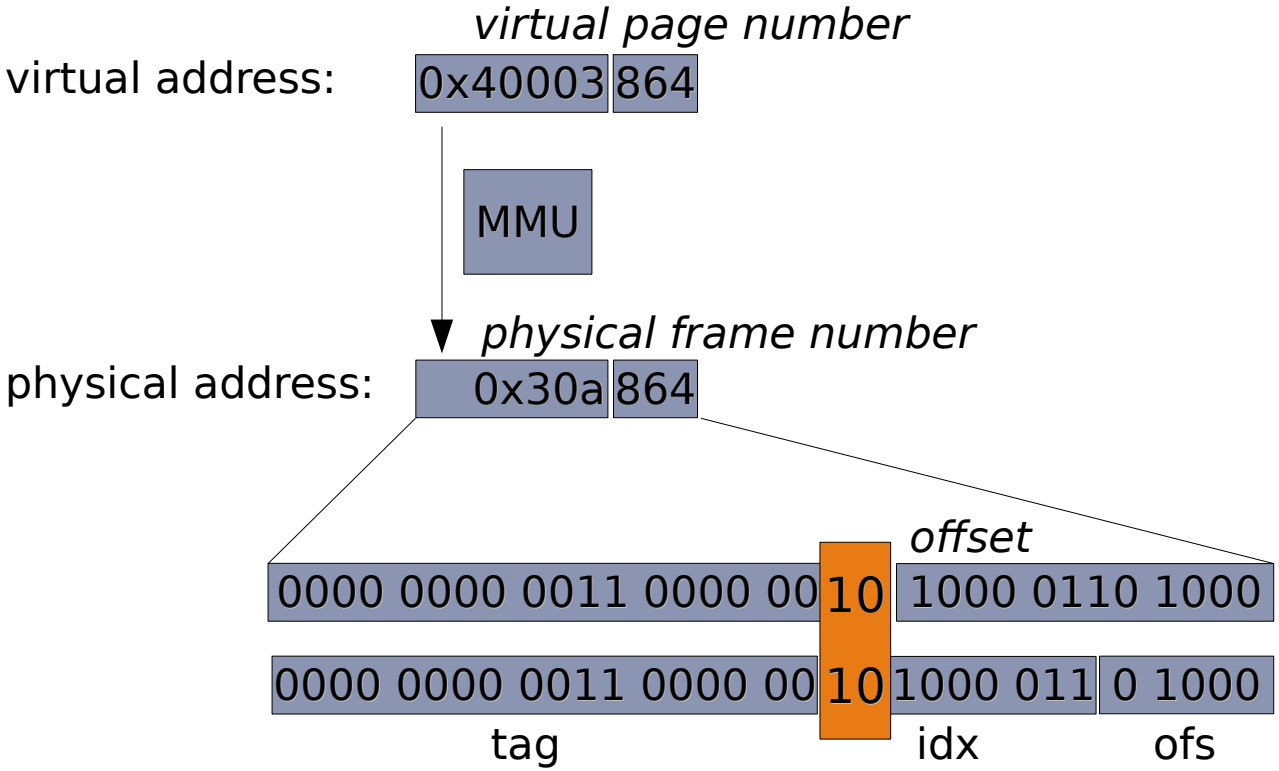
Main Memory




virtual address
v :
vpn
MMU
L1 Cache
real address
r :
rpn
L2 Cache
hardware address
r´:
rpn
Memory

OS-Controlled Cache Partitioning

Address translation and caches



OS-Controlled Cache Partitioning

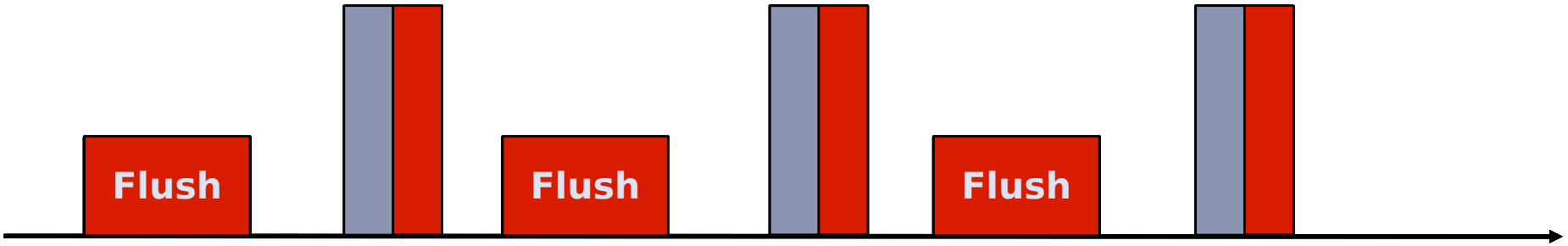
Scenario

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

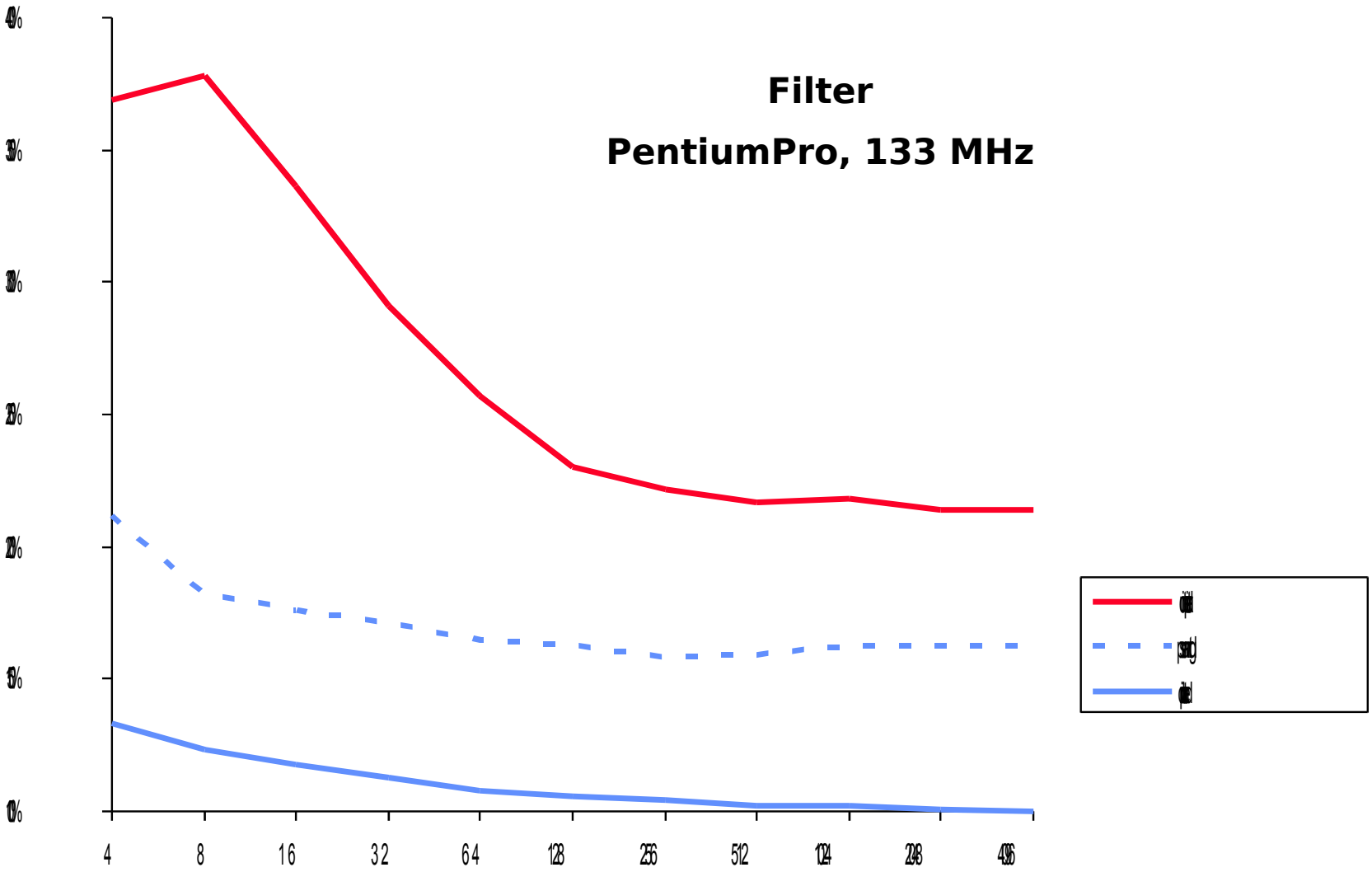
- other tasks in the backgound (e.g., cache flodder)

Example: Filter

- Worst case:
  - without cache partitioning:
    background tasks may evict all cachelines
    background tasks may load conflicting cachelines, which need writeback



OS-Controlled Cache Partitioning

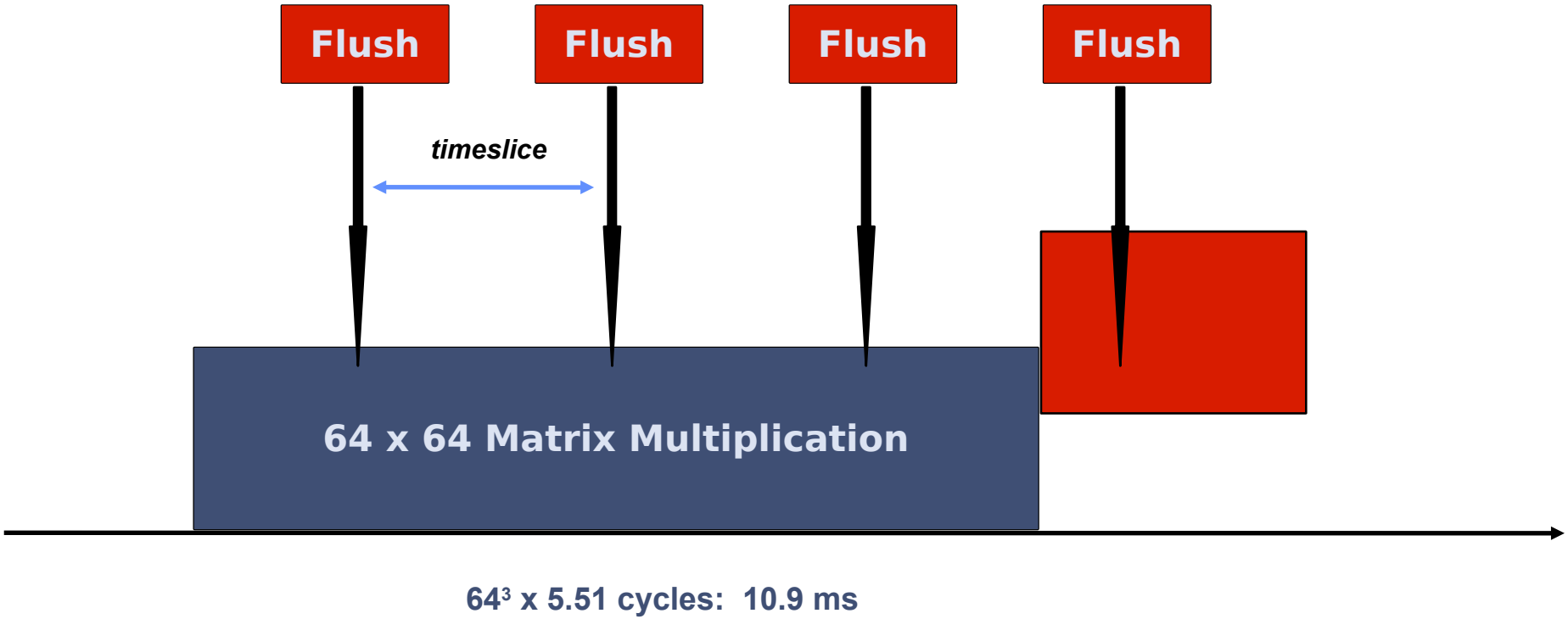


(sngl wr = single writer – only RT app may write to pages of color x ; others may read)

OS-Controlled Cache Partitioning

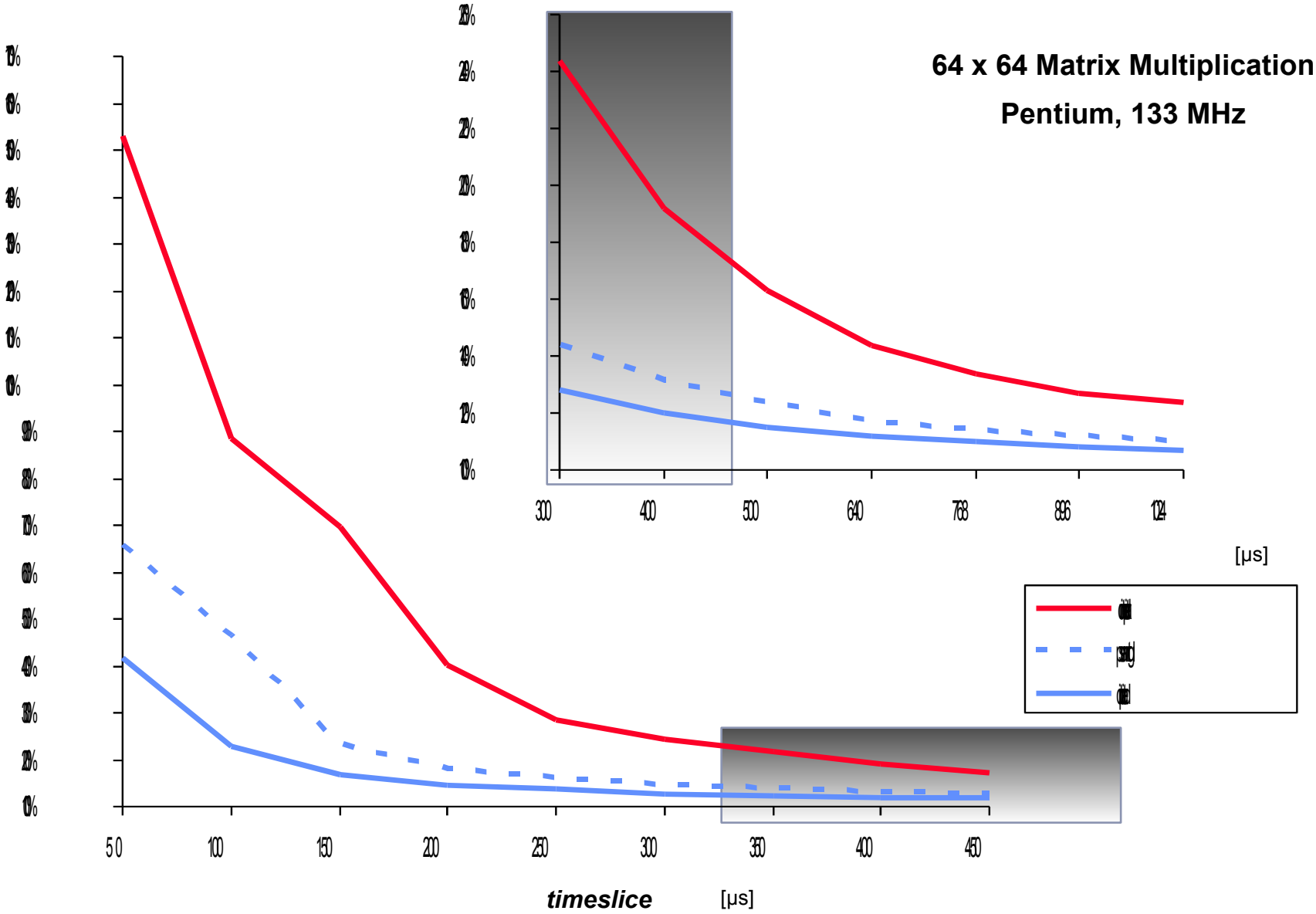
Example 2:

low priority task; frequently interrupted
e.g., matrix multiplication

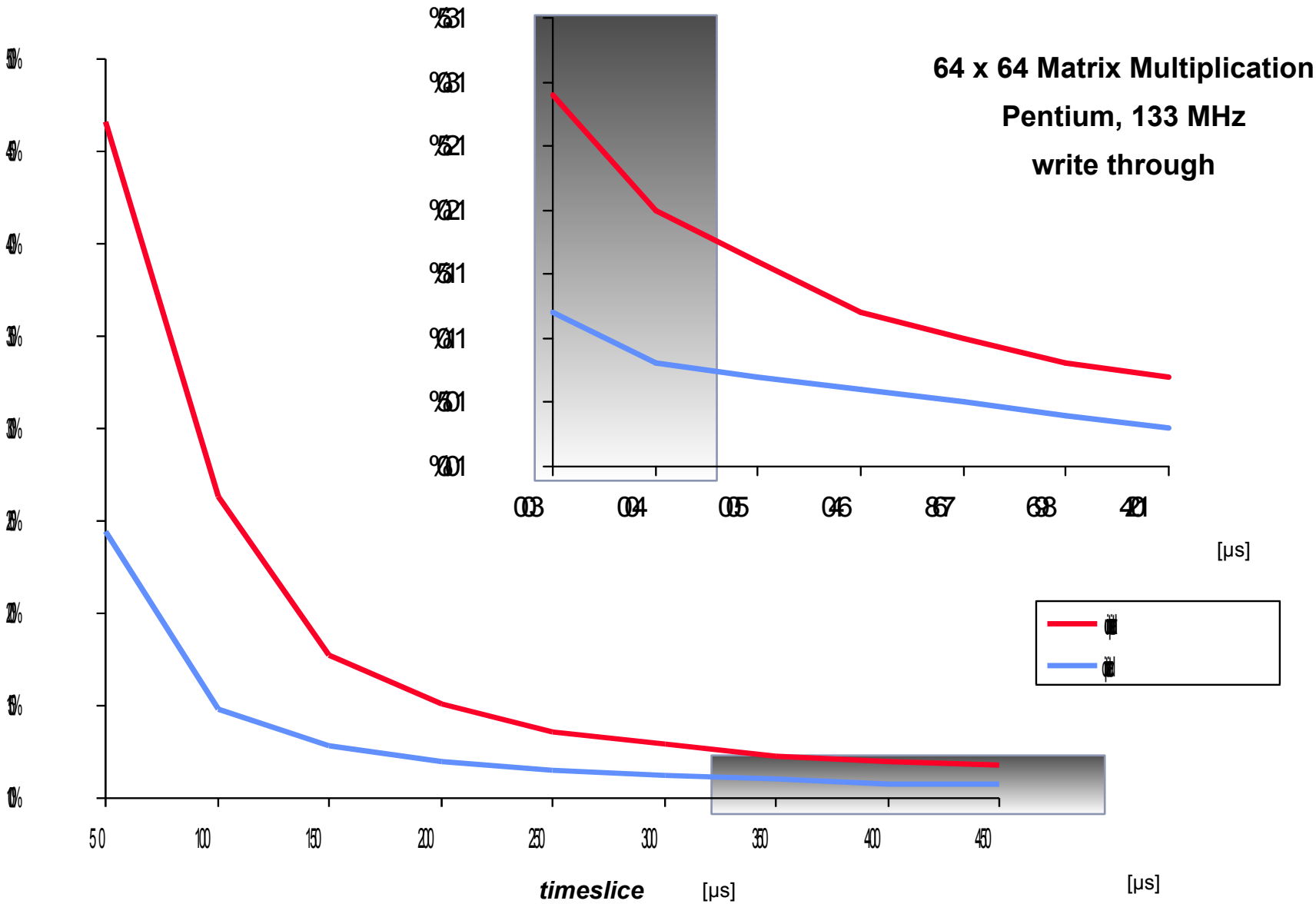


64^3 x 5.51 cycles: 10.9 ms

OS-Controlled Cache Partitioning



OS-Controlled Cache Partitioning



OS-Controlled Cache Partitioning

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

Caveat:

caches (especially in SMP) have become more complicated

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