TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science, Institute for System Architecture, Operating Systems Group

# Real-Time Systems '08 / '09

## Hardware

*Marcus Völp*

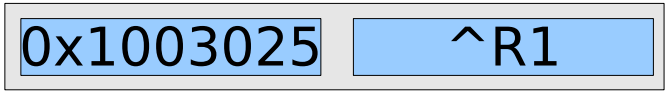
# Outlook

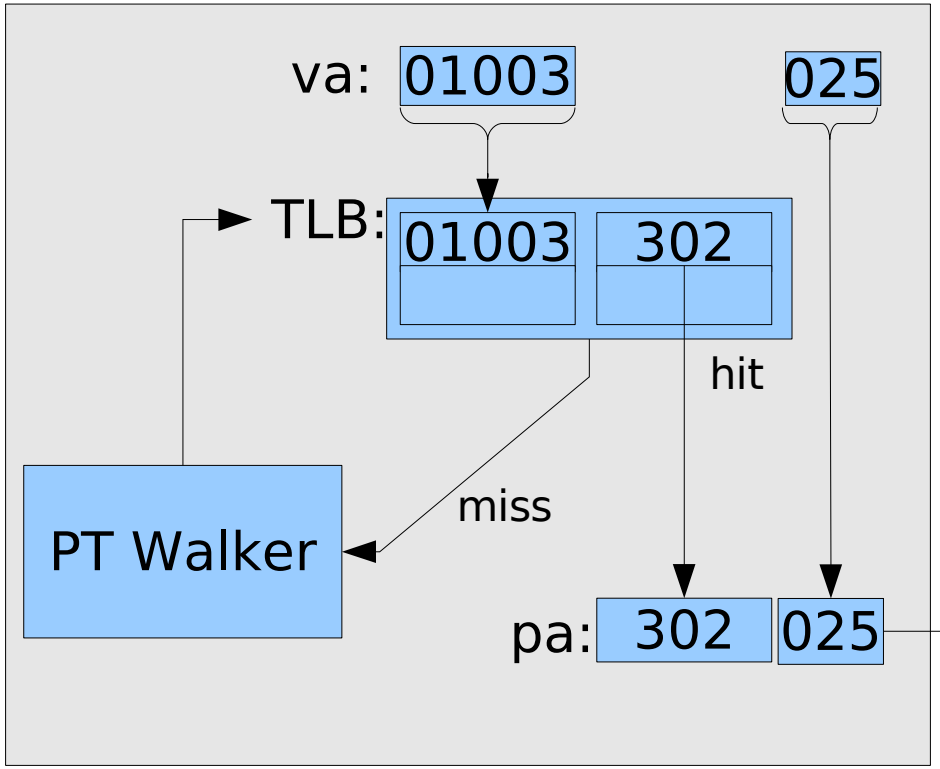
- Hardware is Source of Unpredictability
  - Caches
  - Pipeline
  - Interrupt Latency
- Special Purpose Hardware for “Embedded” Real-Time Systems
  - Low Latency Interrupt Mode
  - Peripheral Event Controller
  - Capture – Compare Units
  - Scratchpad Memories
  - Real-Time Clocks
- Use unpredictable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

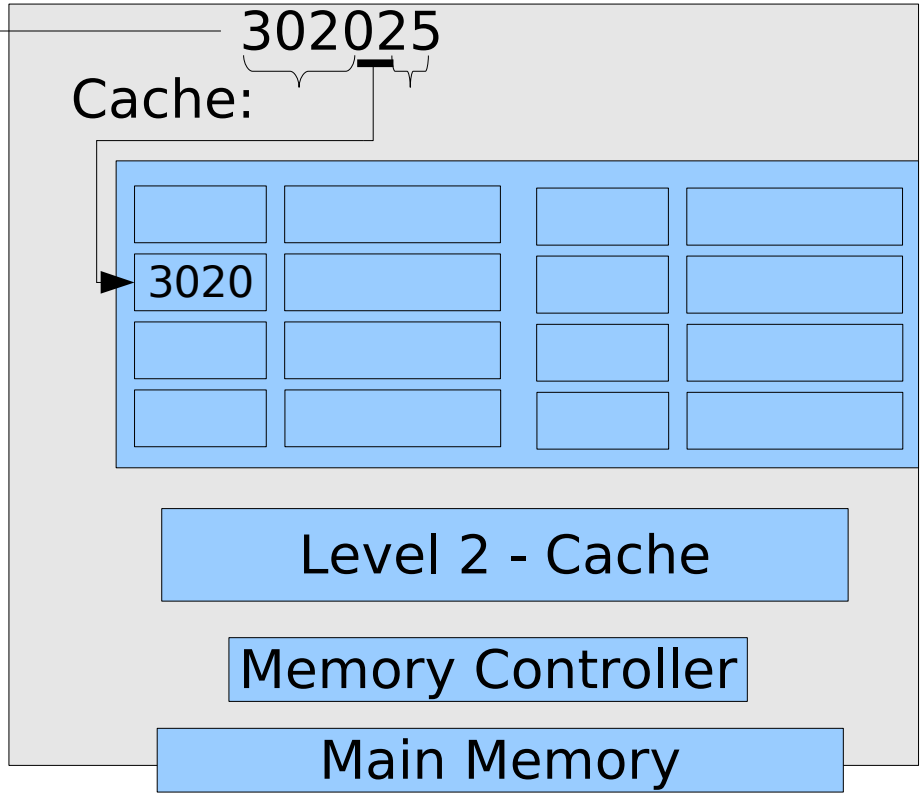
# Sources of Unpredictability

- Memory Subsystem
  - TLB
  - Caches
  - Store Buffers

store R1, 0x01003025

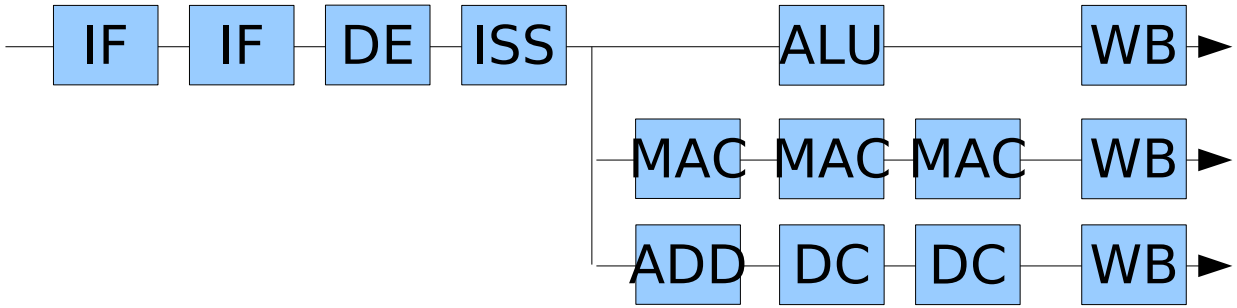


# Sources of Unpredictability

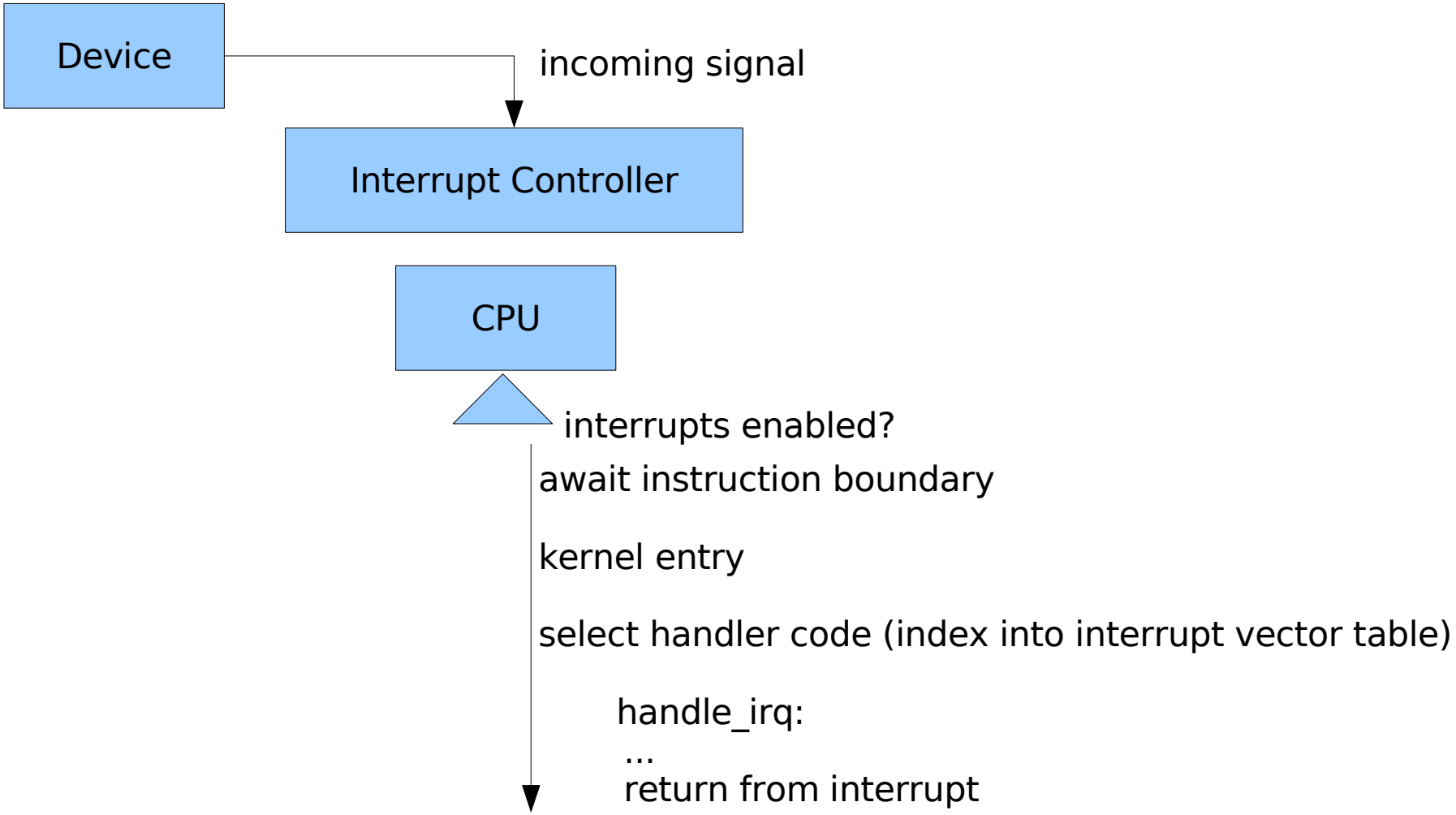
- Processor Pipeline



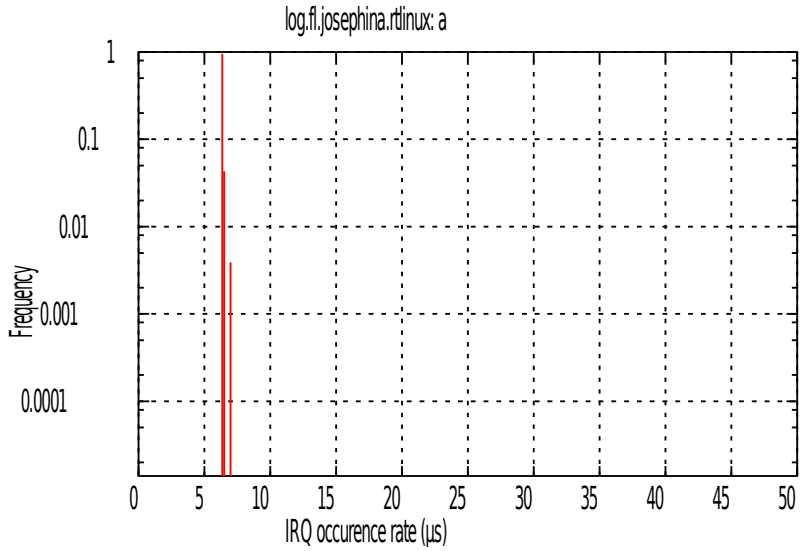

(ARM 11 Microarchitecture)

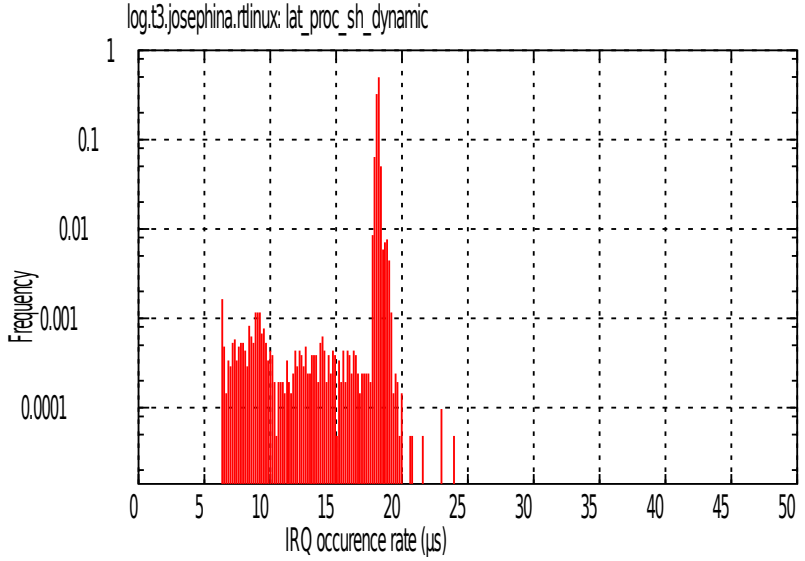
# Sources of Unpredictability

- Interrupt Latency



# Interrupt Response Time – RTLinux



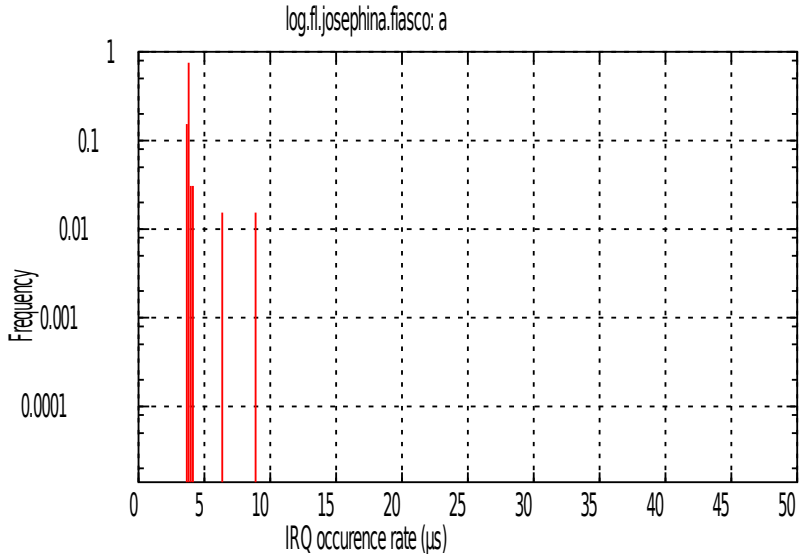

***Interrupt response time:***
*Time from occurrence of interrupt to first instruction of RT Task*

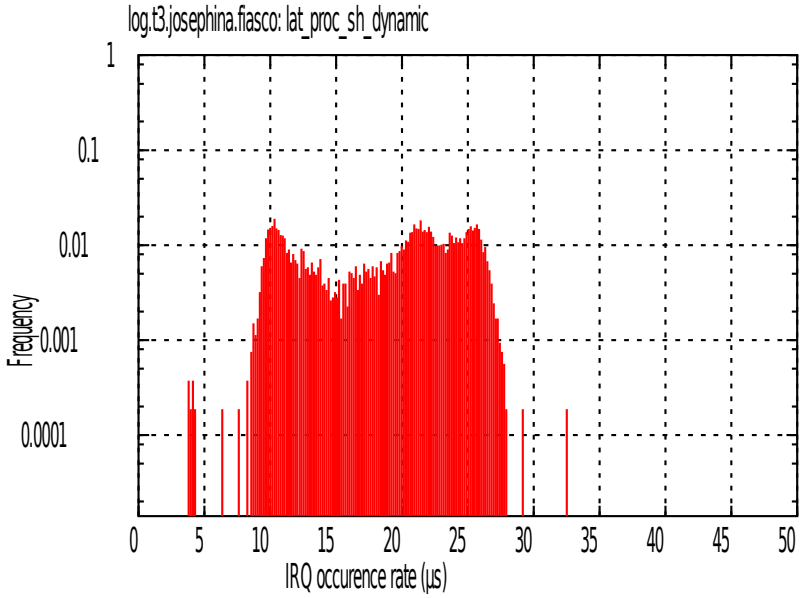
No parallel load (idle): 13 µs

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

# Outlook

- Hardware is Source of Unpredictability
  - Caches
  - Pipeline
  - Interrupt Latency
- 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
  - Cache Partitioning
- Real-Time Communication / Buses in separate Lecture

# Low Latency Interrupts

- Sampling of Data in a High Rate (Sensors, Video, ...)
- Fast Response Times (Break Control, ...)

- ARM IRQ + FIQ
  - FIQ interrupts IRQ handler
  - 5 immediately available registers 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
    - software must not use multi-word load / store instructions
    - software should 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 / 12 cycles

# Low Latency Interrupts

- **Obtaining precise timestamps with interrupt-based sampling is difficult**
  - system load increases 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

- **Peripheral Event Controller (PEC)**
  - executes one memory-to-memory move
  - triggered by arriving signal

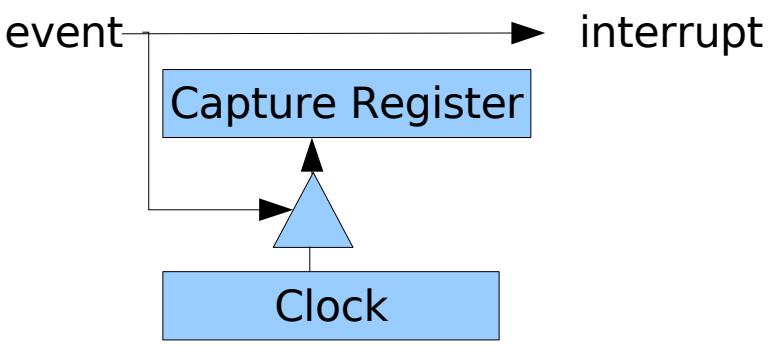
  - **Programmers Interface:**
    - counter — decrement on signal, trigger CPU interrupt on 0
    - byte / word select — select width of transfer (byte or word)
    - source / dest increase — update source / destination address
    - source / dest address

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

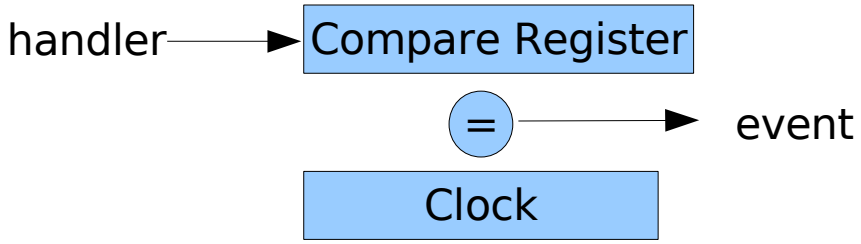
# Capture + Compare Unit

- **Problem**
  - precise timestamp of event
    - (engine sensor triggered at cylinder position **= t**)
  - generate precise events
    - (fire the engine at ***t + x***)
  - interrupt handlers have too high a jitter to meet the points in time

**Capture**




**Compare**



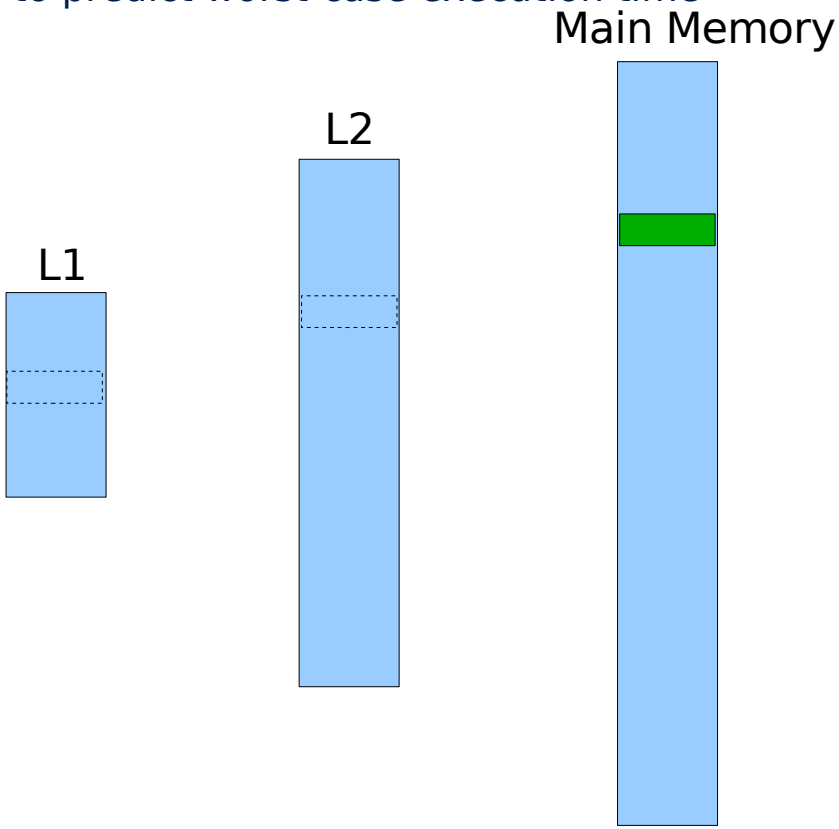
# System on a Chip / Chip Multiprocessor

- 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

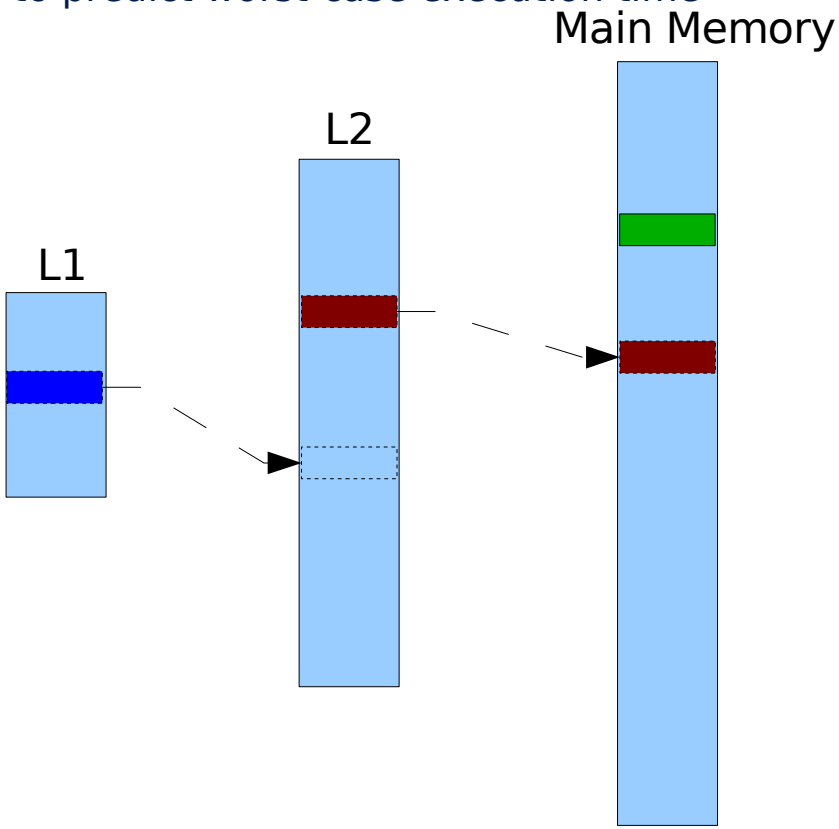
# Scratch-pad Memory vs. Caches

- 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



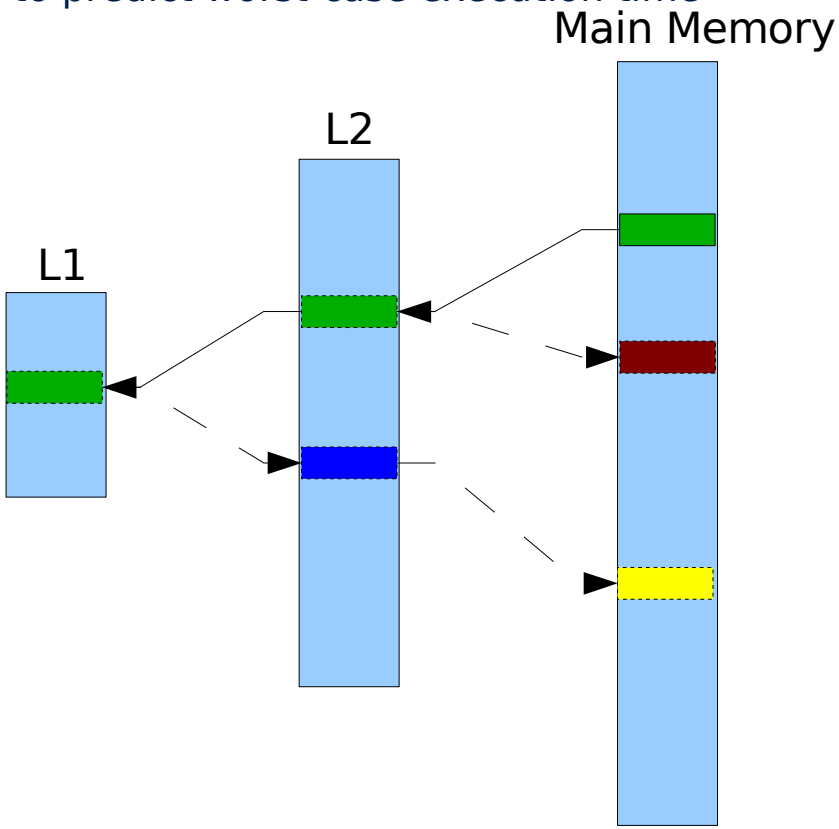
# Scratch-pad Memory vs. Caches

- 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



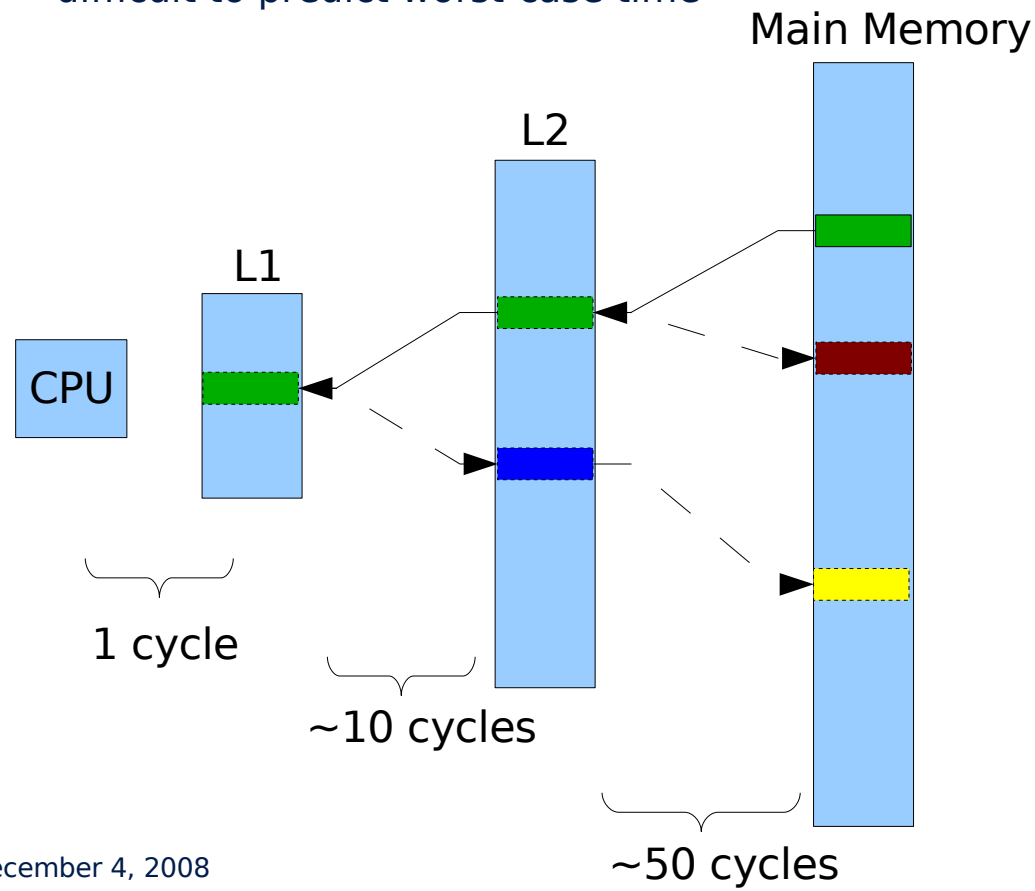
# Scratch-pad Memory vs. Caches

- 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

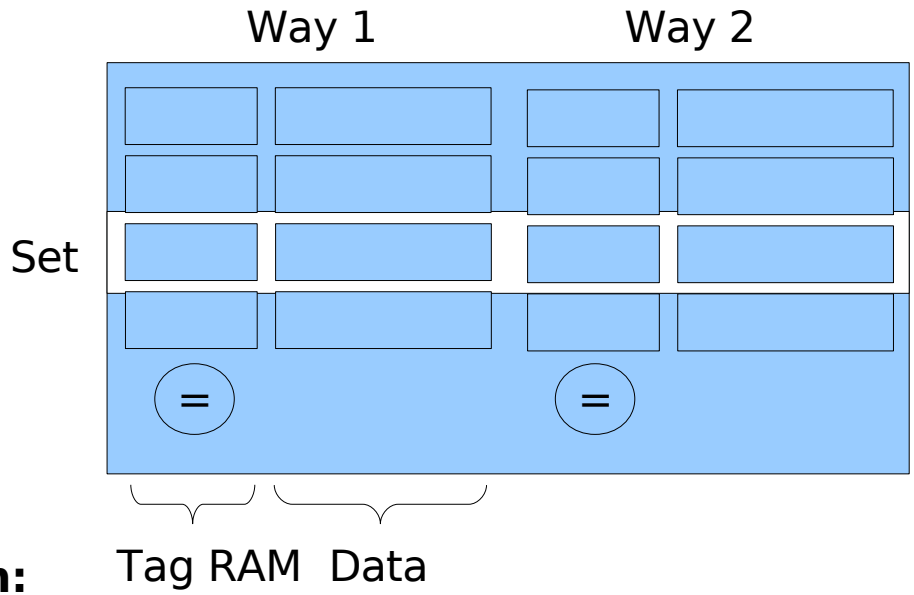


# Scratch-pad Memory vs. Caches

- 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 time



# Scratch-pad Memory vs. Caches




**Cache lockdown:**

- Lock cacheway
- Allocate only to unlocked ways

# Scratch-pad Memory vs. Caches

- Scratchpad Memory:
  - memory can be addressed directly
    (device mapped to physical memory space)
  - 2 cycles latency / currently ~ 64 KB (more ~4 MB to come)
  - must relocate data explicitly

  - Compiler-based approaches:
    - static: determine code + data hot spots +
      allocate scratchpad memory for hot spots
    - dynamic: copy data to / from scratchpad memory

```
int a[20];

for (int j = 0; j < 40; j++)
  for (int i = 5; i < 10; i++)
      a[i] *= 42;
```

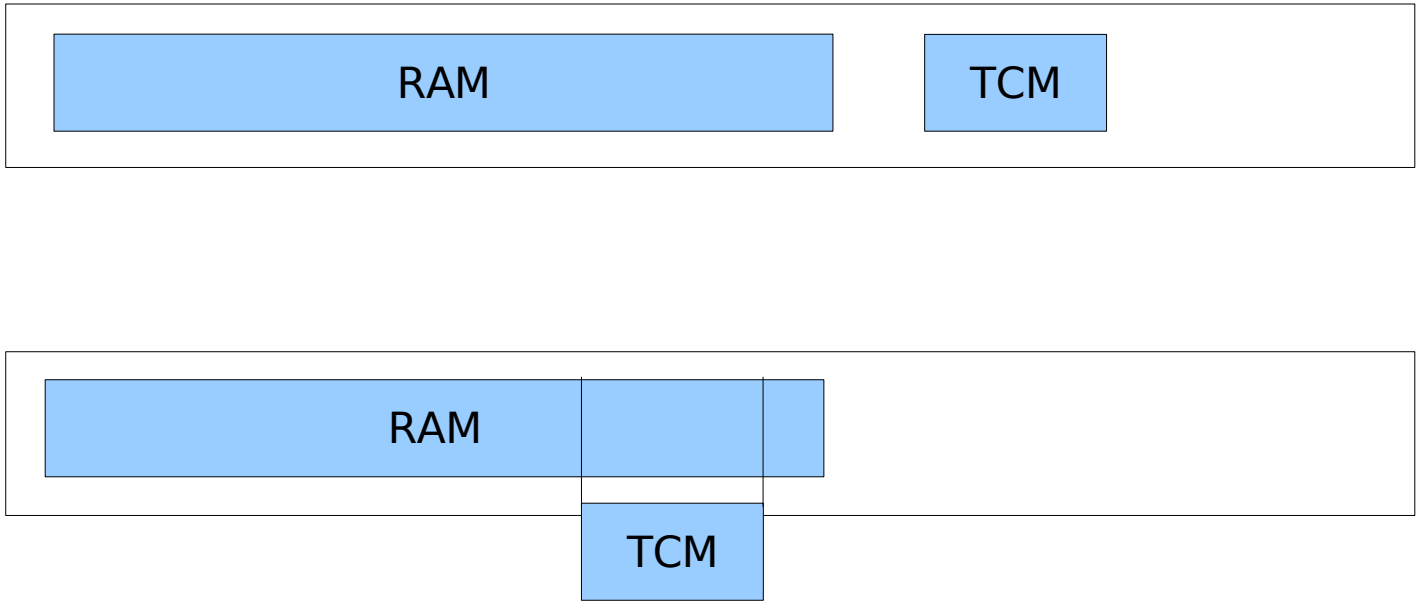
=>

```
int a[20];
int b[5] = {a[5], a[6], a[7], a[8], a[9]};

    for (int j = 0; j < 40; j++)
        for (int i = 0; i < 10; i++)
            b[i] *= 42;
```

# Scratch-pad Memory vs. Caches

- 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

# Real Time Clocks

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

need to synchronize clocks within core / system wide

~> separate lecture

# Outlook

- Hardware is Source of Unpredictability
  - Caches
  - Pipeline
  - Interrupt Latency
- 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
  - Cache Partitioning
- Real-Time Communication / Buses in separate Lecture

# OS Controlled Cache Predictability

- goals
  - partition cache to minimize interference between RT Applications
  - OS controlled
    - transparent to application
    - no need to rely on cooperating applications
    - survives malicious, erroneous applications

- **Cache Partition**

  Address regions that map to a given set of cachelines (a partition)

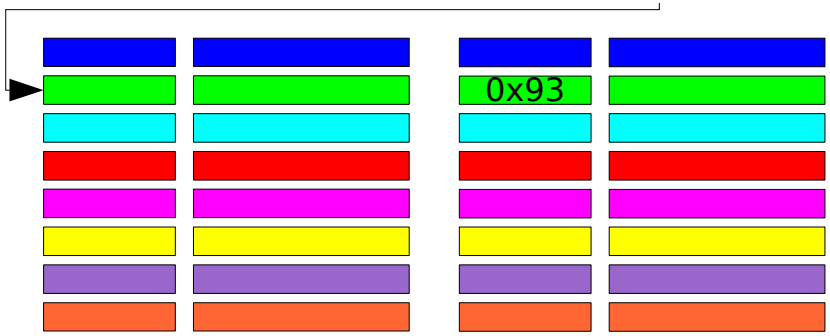
  - tool support: Compiler / Linker

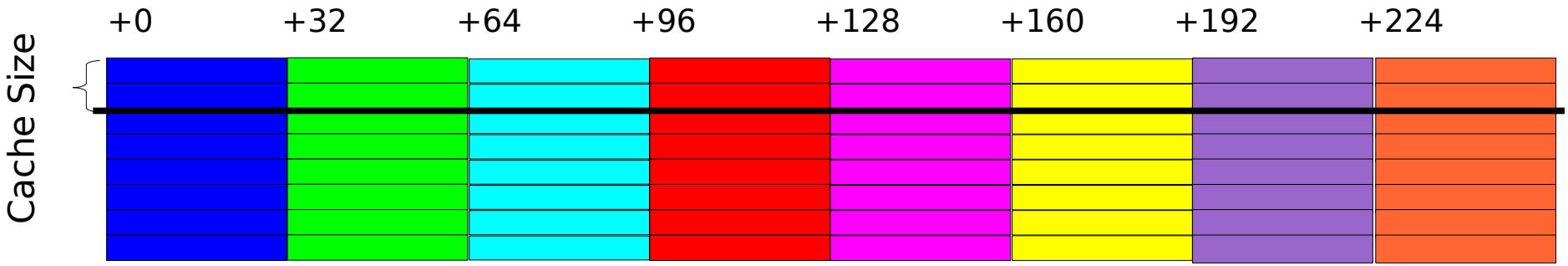
# OS Controlled Cache Predictability

**Cache Coloring**

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

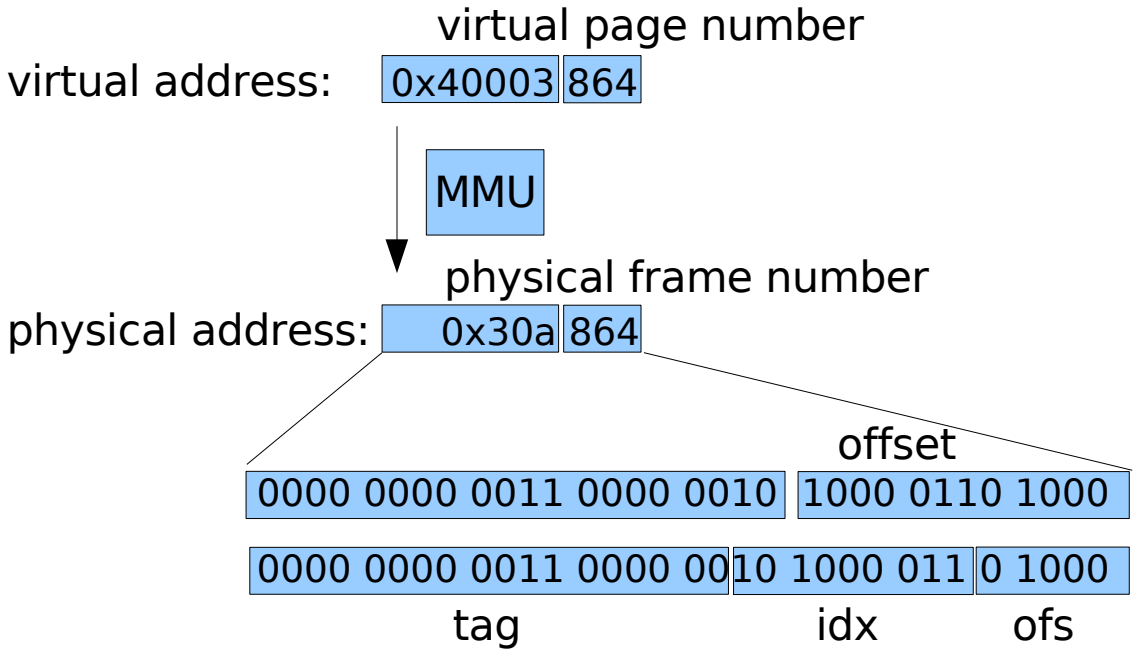
tag idx offset into 32 byte cacheline



# OS Controlled Cache Predictability

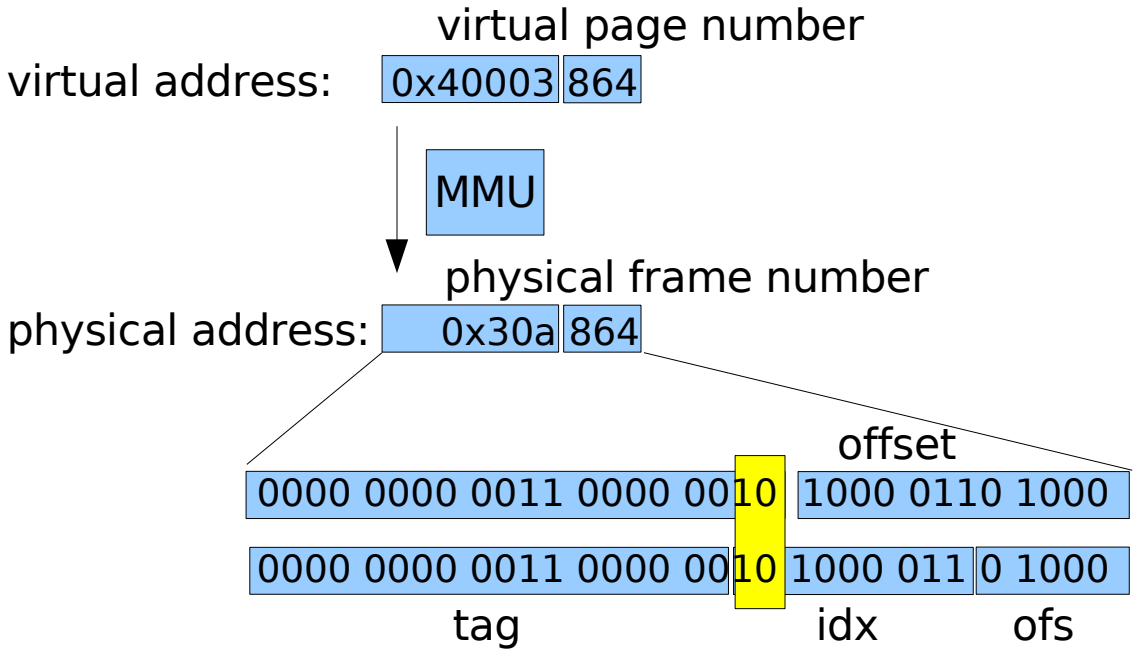
**Cache Coloring**
Address translation and caches




L2 Cache:
4MB, 4 Way, 32byte CL

# OS Controlled Cache Predictability

**Cache Coloring**
Address translation and caches




L2 Cache:
4MB, 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

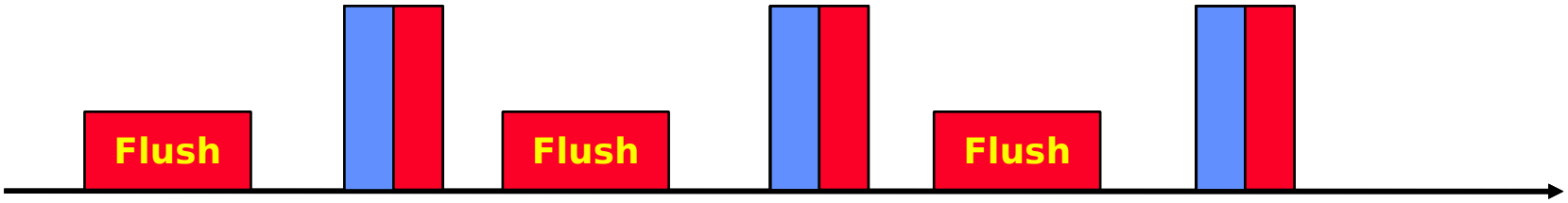
# OS Controlled Cache Predictability

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

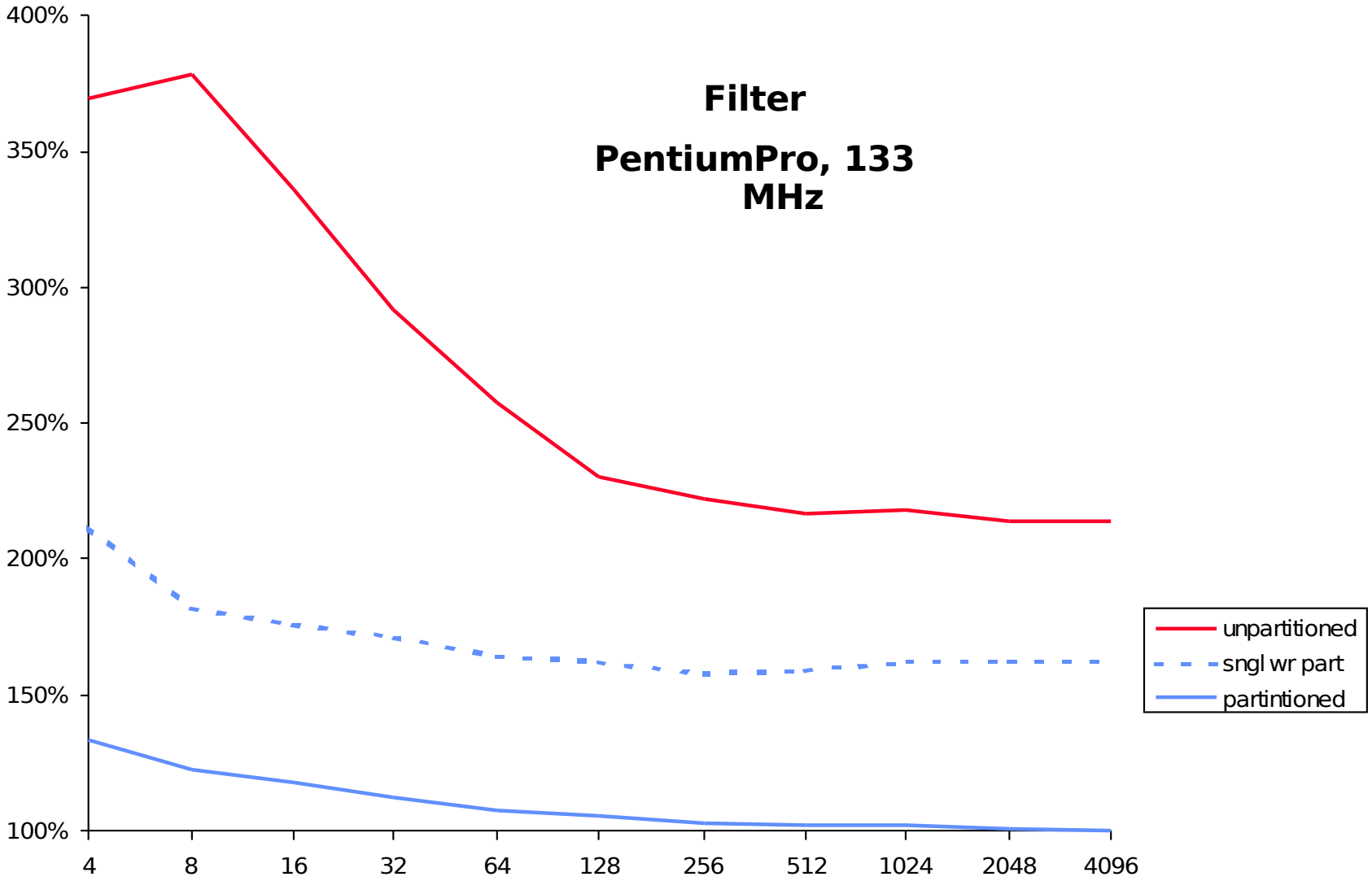
- other tasks in the backgound (e.g., cache flodder)

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 Predictability

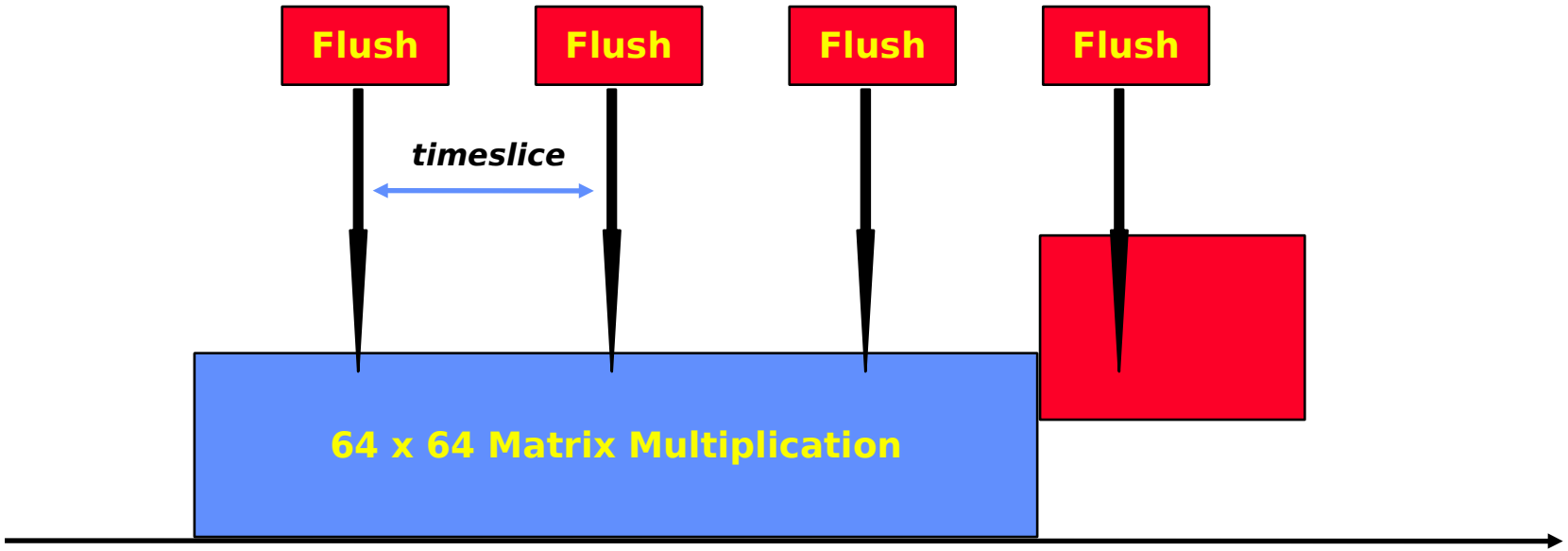



(sngl wr = single writer – only RT app may write to pages of color x ; others may read)

# OS Controlled Cache Predictability

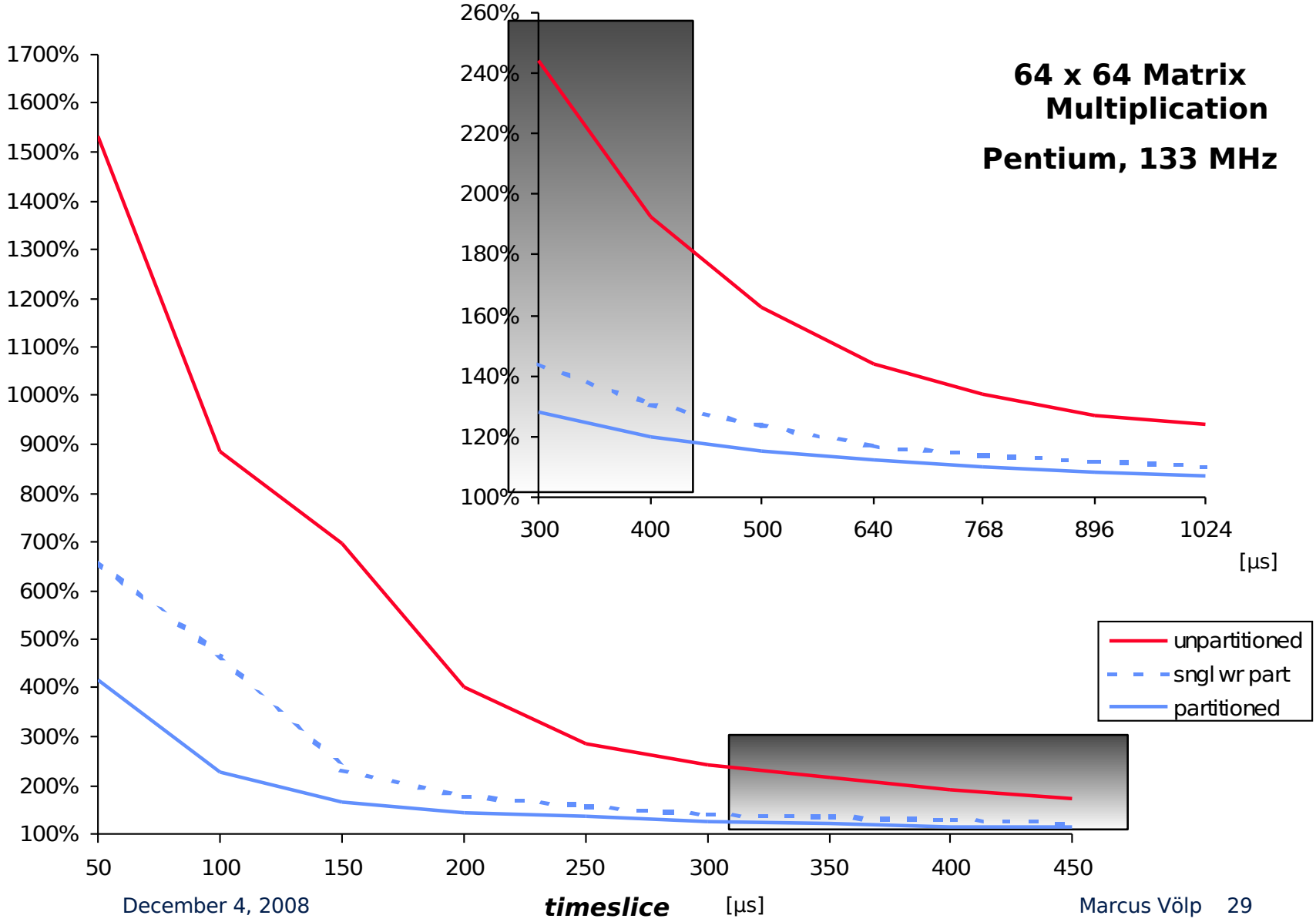
**Example 2:**

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

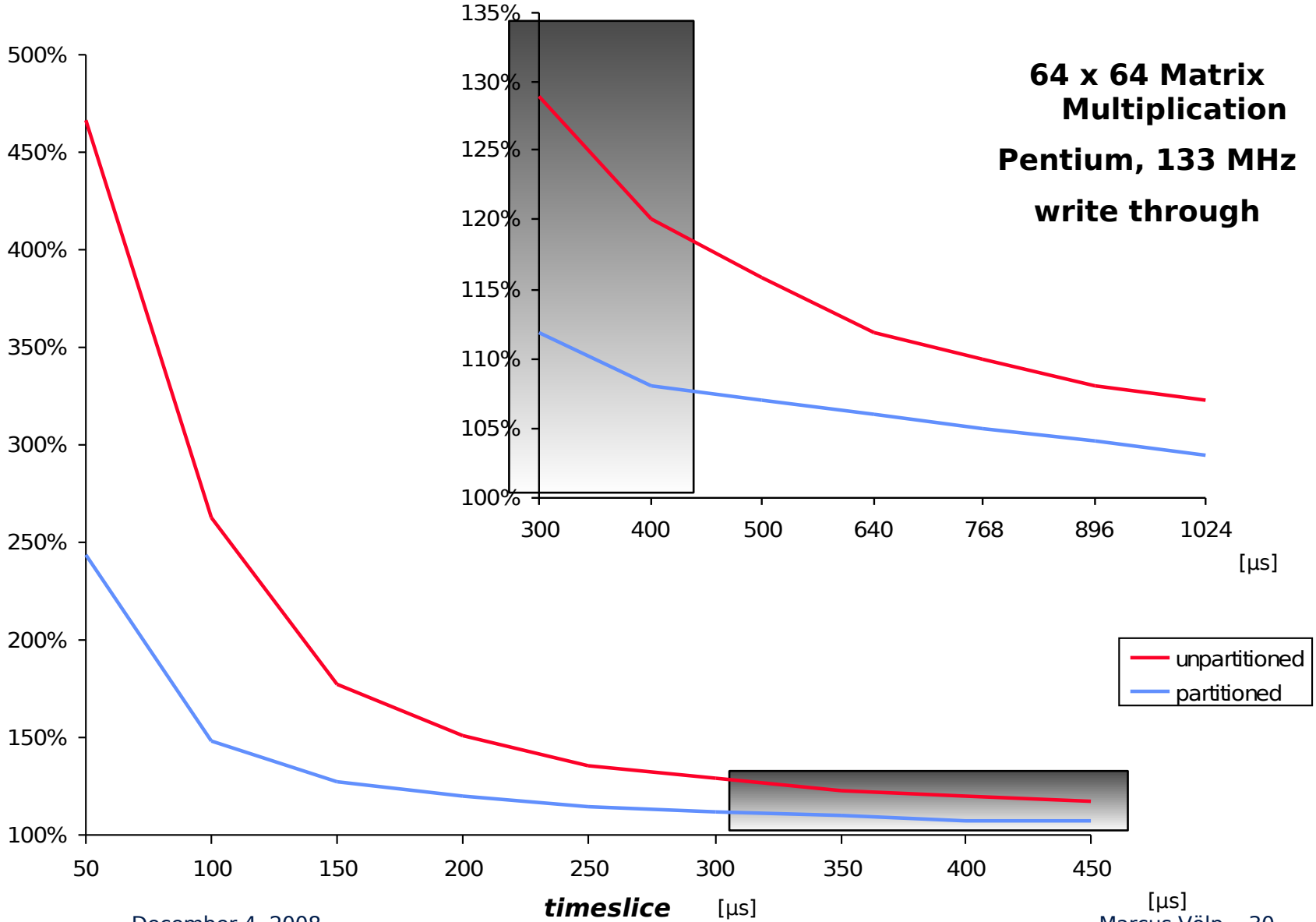



**$64^3$ x 5.51 cycles: 10.9 ms**

# OS Controlled Cache Predictability

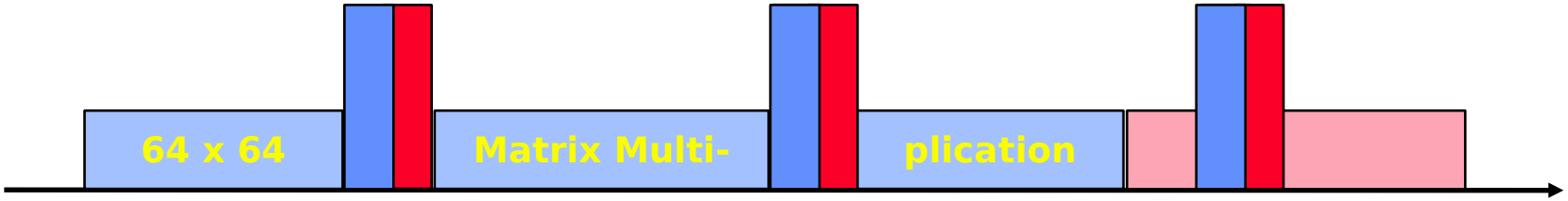


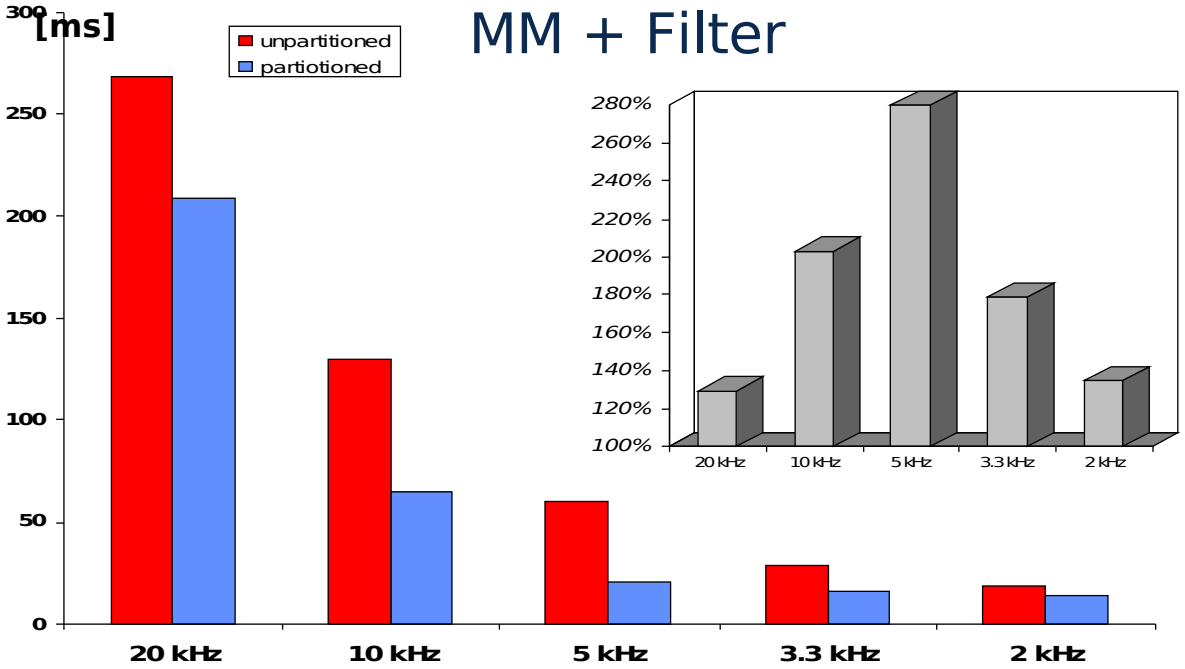
# OS Controlled Cache Predictability



# OS Controlled Cache Predictability

**Experiment 3:**
Combination: Matrix Multiplication + Filter



# OS Controlled Cache Predictability

**Caveat: application transparency**

some applications require knowledge about 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