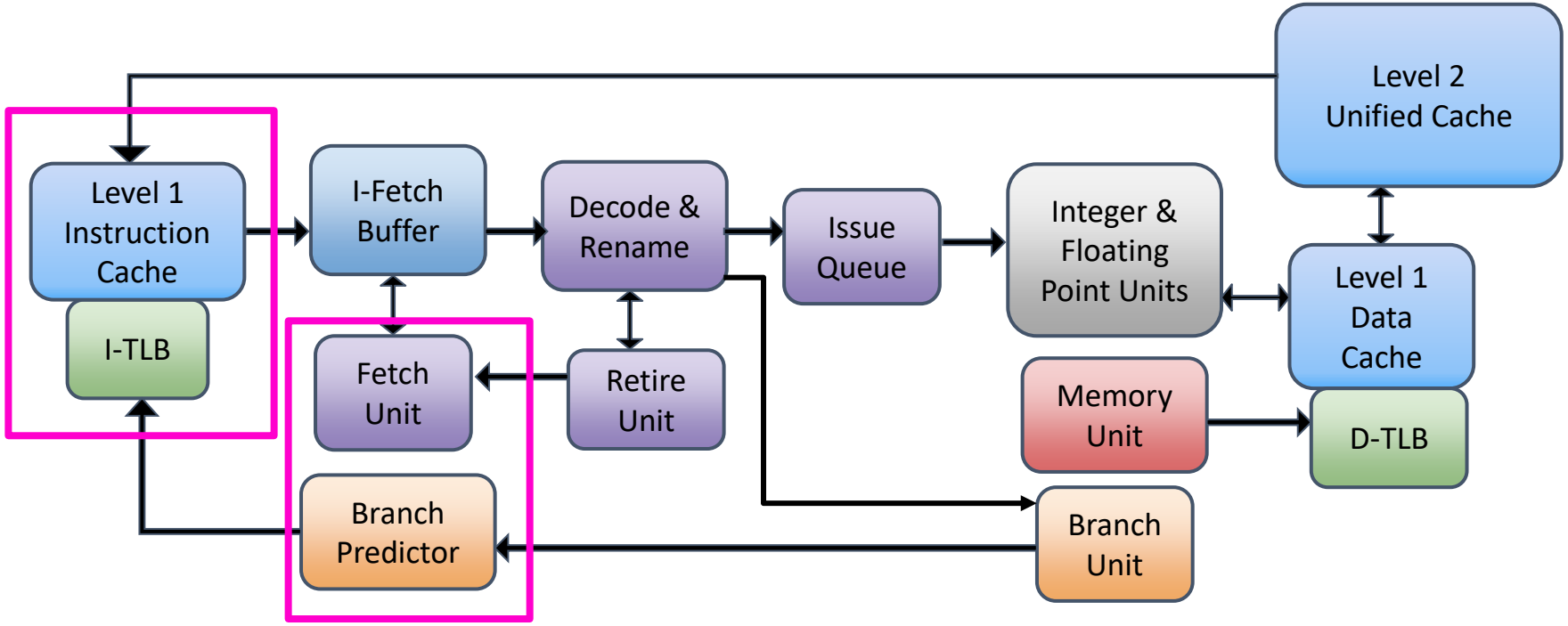
CASPER

# CS773: Computer Architecture for Performance and Security

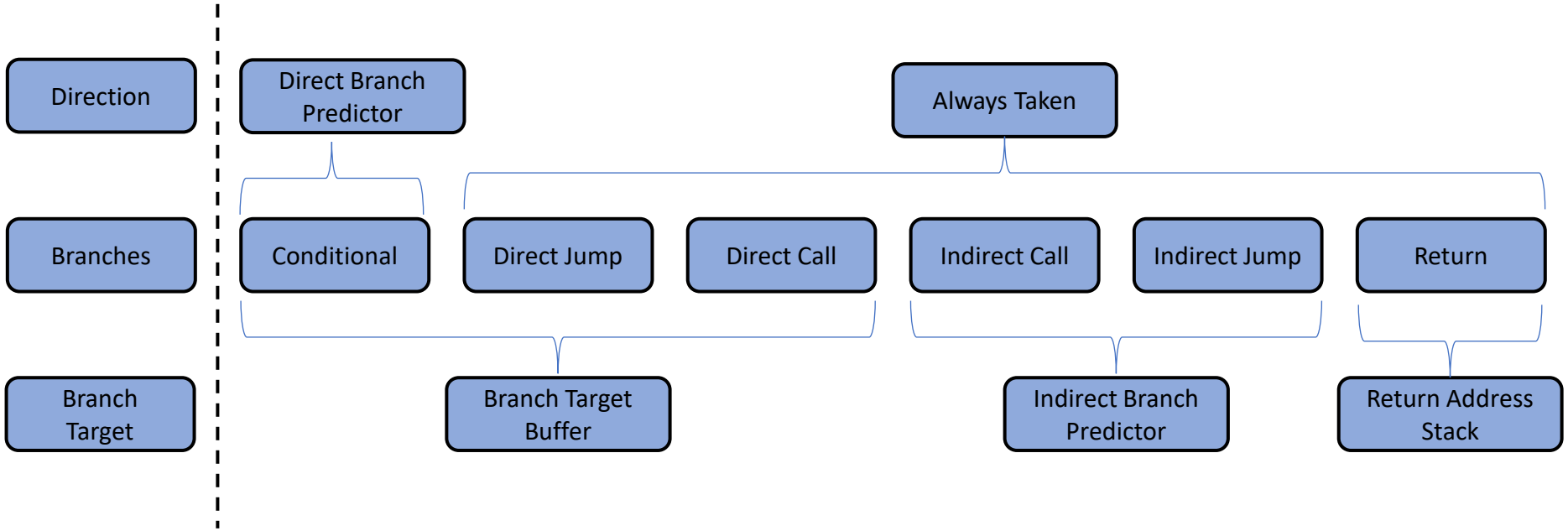
## Lecture 4: Microarchitecture for performance

https://www.cse.iitb.ac.in/~biswa/

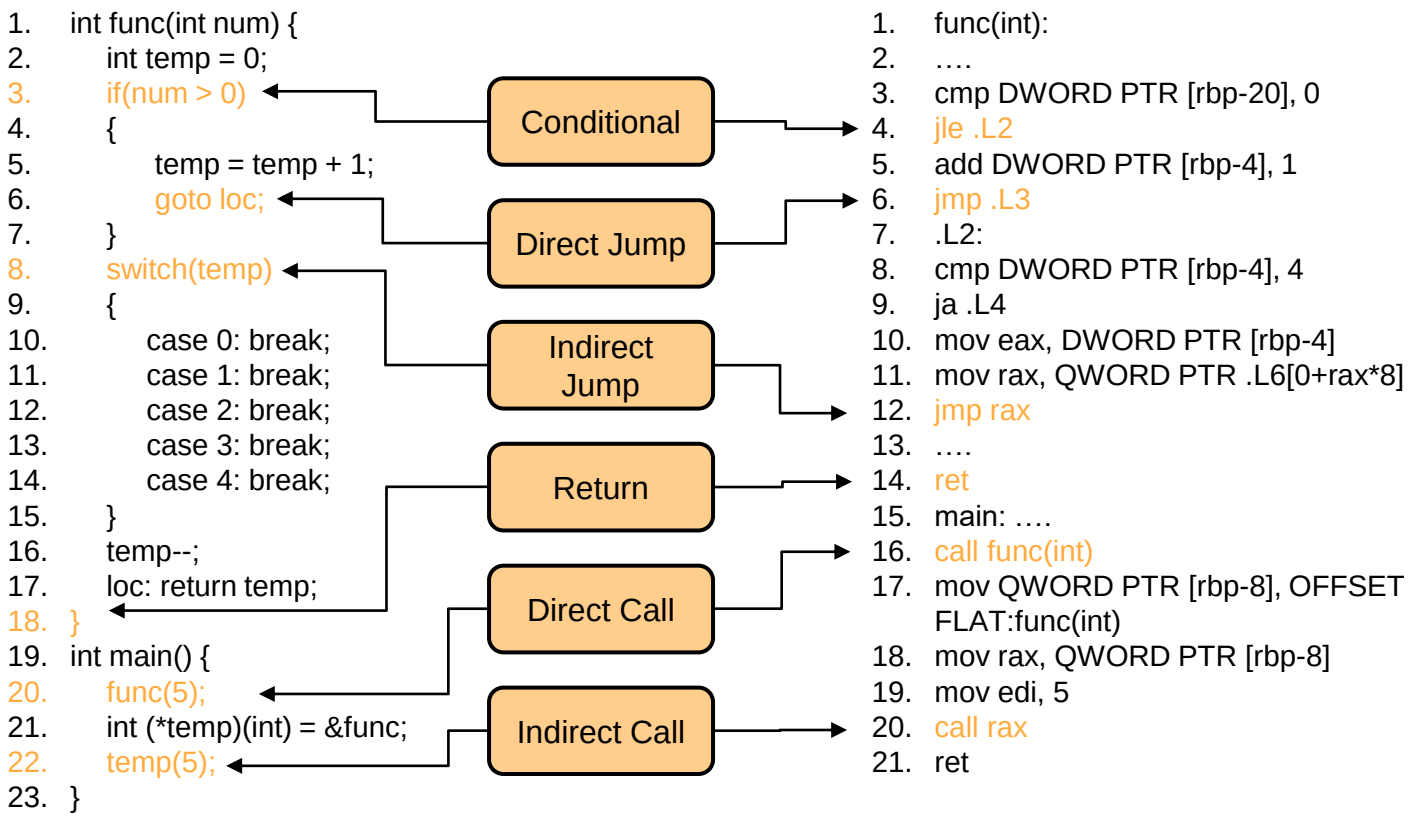
# Microarchitecture for Performance



# Front-end Bottleneck (Servers, mobile workloads)



# Branch instructions




(a) C++ Code

(b) Assembly Code

# BTB/L1I Capacity/latency Problem

## The AMD "Zen 2" Processor

The branch capacity in Zen 2 is nearly double that of Zen. The L0 BTB was increased from 8 to 16 entries. The L1 BTB was increased from 256 to 512 entries. The L2 BTB was increased from 4096 to 7168 entries. The indirect target array

## The IBM z15 High Frequency Mainframe Branch Predictor

to generation as shown in table 1. The z15 BTB1 can hold up to 16K branches and is organized as 2K logical rows with 8 ways per row. The BTB2 can hold 128K branches and is organized as 32K logical rows with 4 ways per row.

## Evolution of the Samsung Exynos CPU Microarchitecture

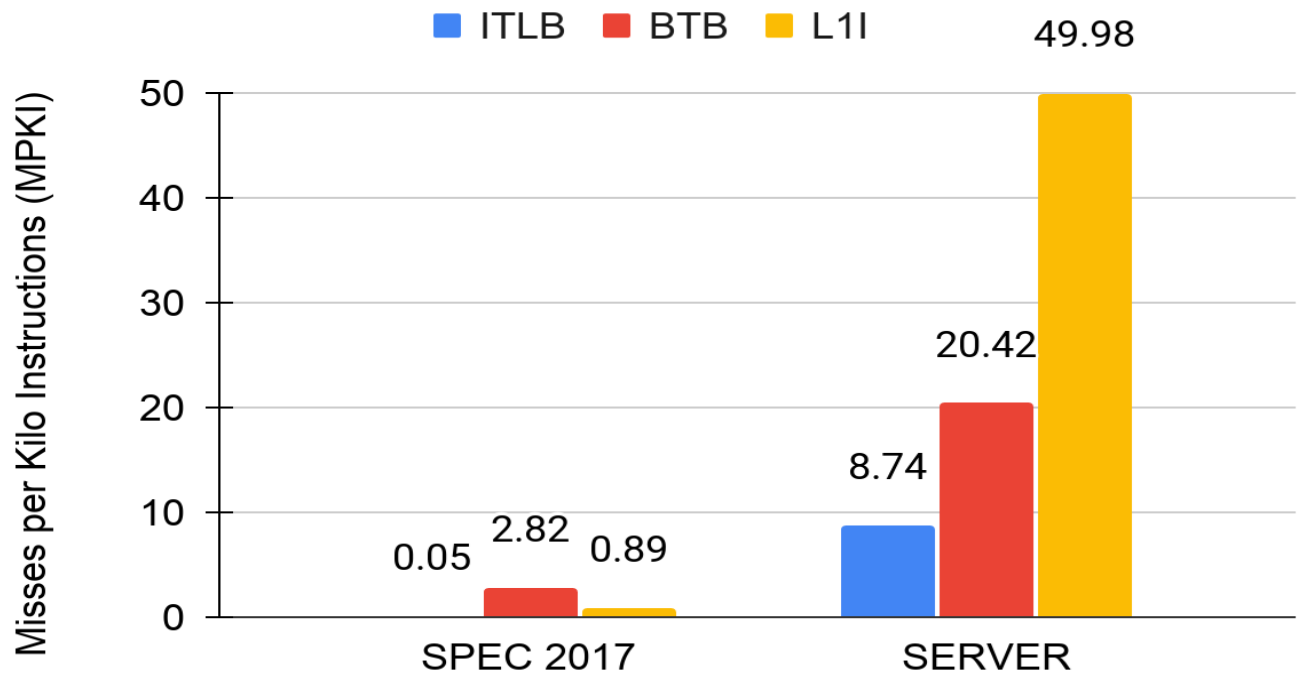
BRANCH PREDICTOR STORAGE, IN KBYTES

| Bit storage (KB) | SHP | L1BTBs | L2BTB | Total |
|---|---|---|---|---|
| M1/M2 | 8.0 | 32.5 | 58.4 | 98.9 |
| M3 | 16.0 | 49.0 | 110.8 | 175.8 |
| M4 | 16.0 | 50.5 | 221.5 | 288.0 |
| M5 | 32.0 | 53.3 | 225.5 | 310.8 |
| M6 | 32.0 | 78.5 | 451.0 | 561.5 |

## The Arm Neoverse N1 Platform: Building Blocks for the Next-Gen Cloud-to-Edge Infrastructure SoC

The branch predictor employs a large 6K-entry main branch target buffer with 3-cycle access latency to retrieve branches' target addresses without accessing the I-cache. Such a

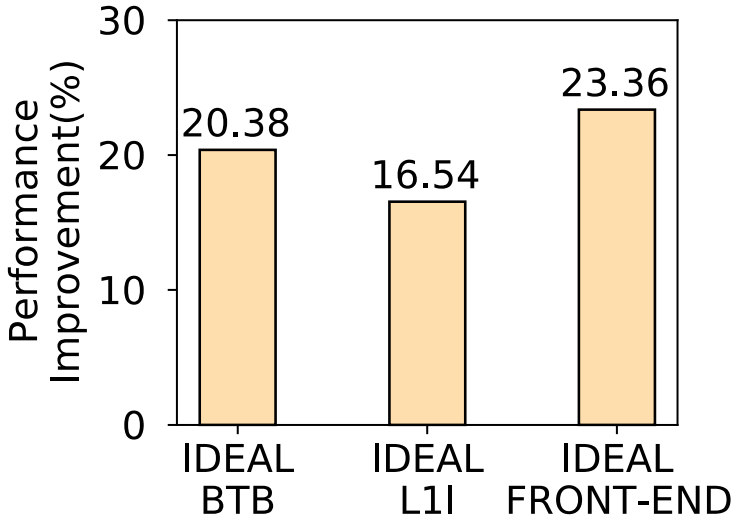
# An Example

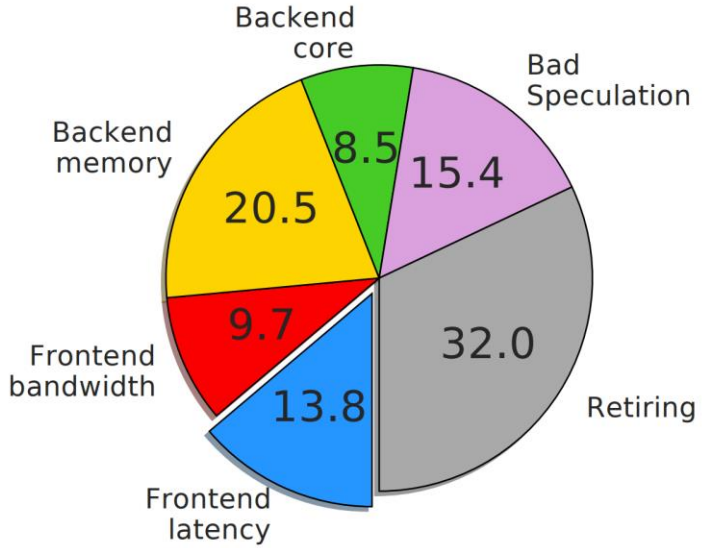



**High MPKI in front-end structures due to increase in codebase.**

# Ideal front-end: Google Web search

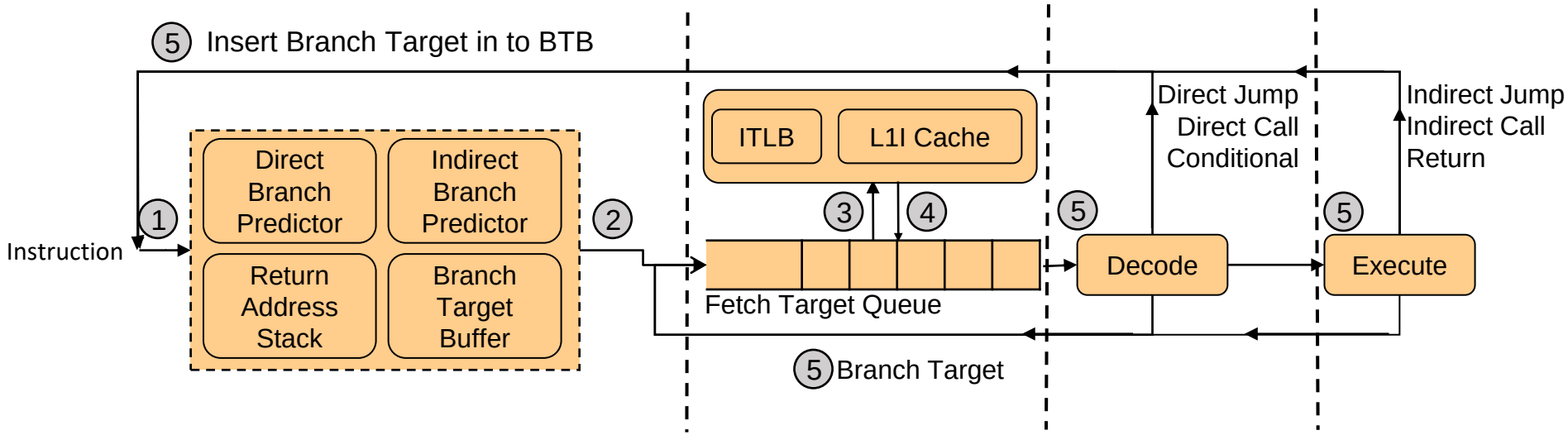
Ideal BTB :- All misses converted to hits.
Ideal L1I :- Misses converted to hits.




Ideal front-end does not provide significant performance improvement on top of ideal BTB (it has ideal ITLB too).

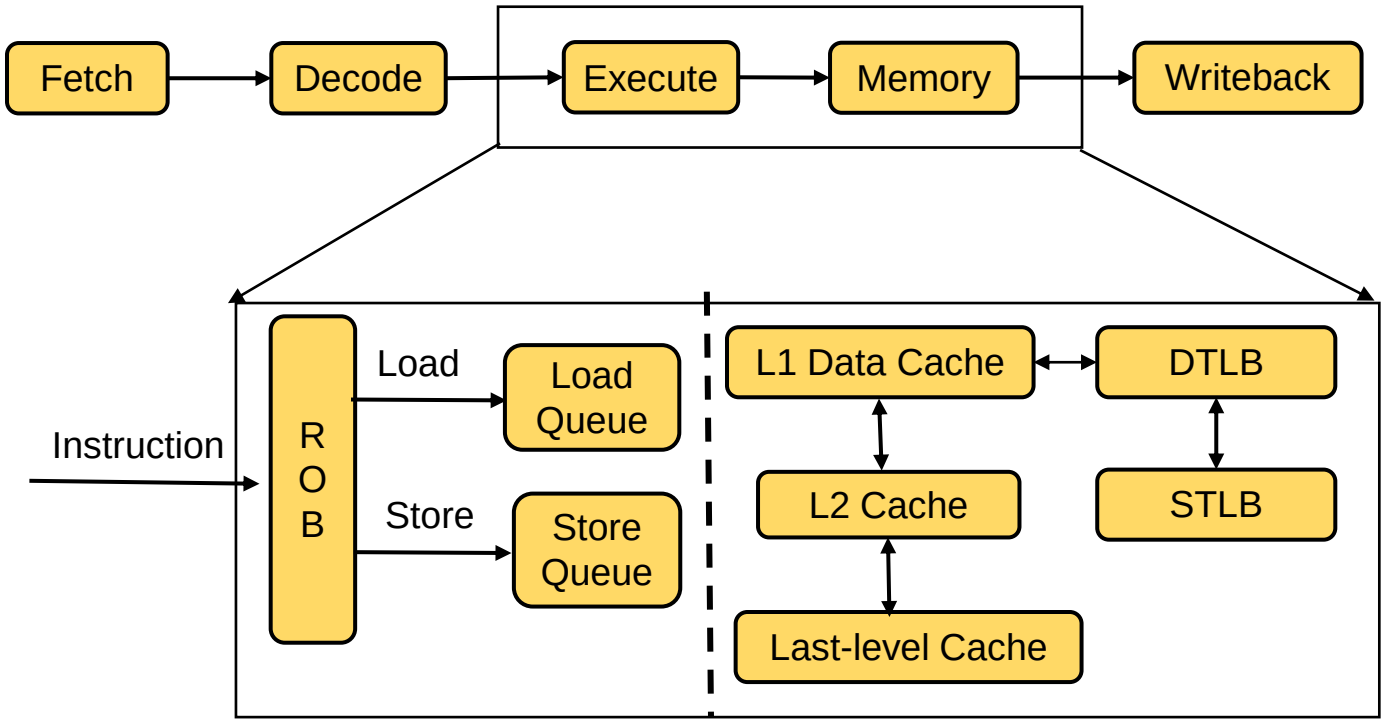
# Decoupled Front-end




Branch predictor looks ahead even if L1I misses

# PAUSE

# Front-end to Back-end



# Virtual Memory

App. 1

**Virtual address space**

Printf (“%d”, &a);

App. 2

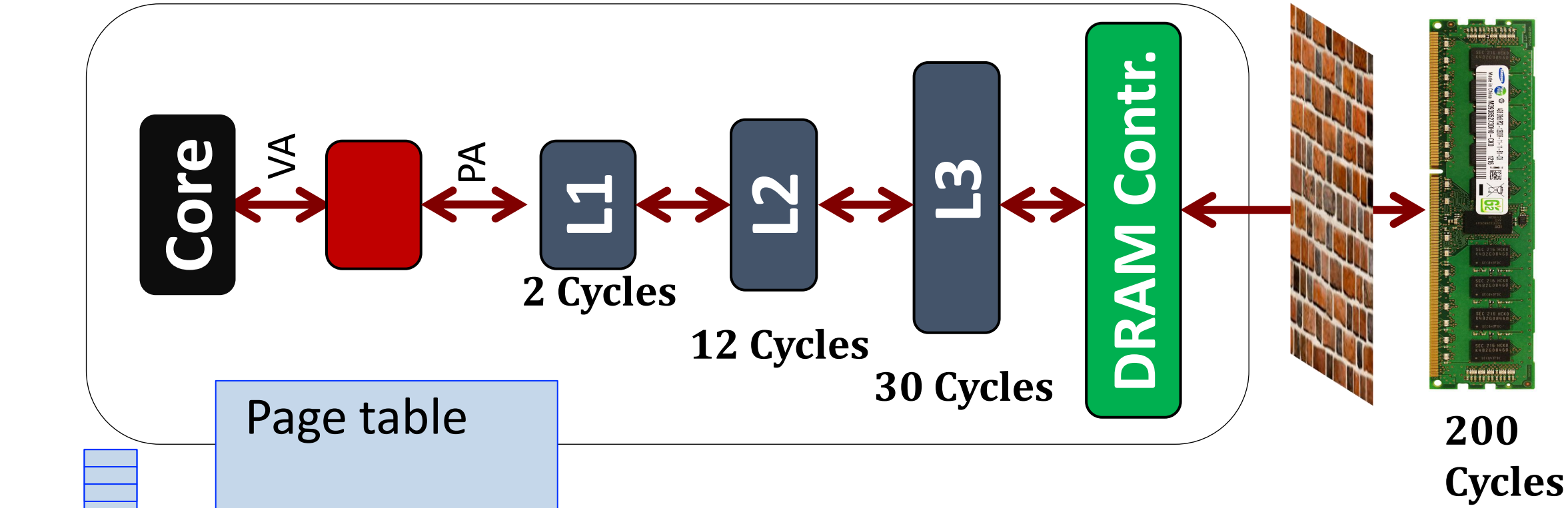
**Virtual address space**

Printf (“%d”, &a);

100

100

# Page Table



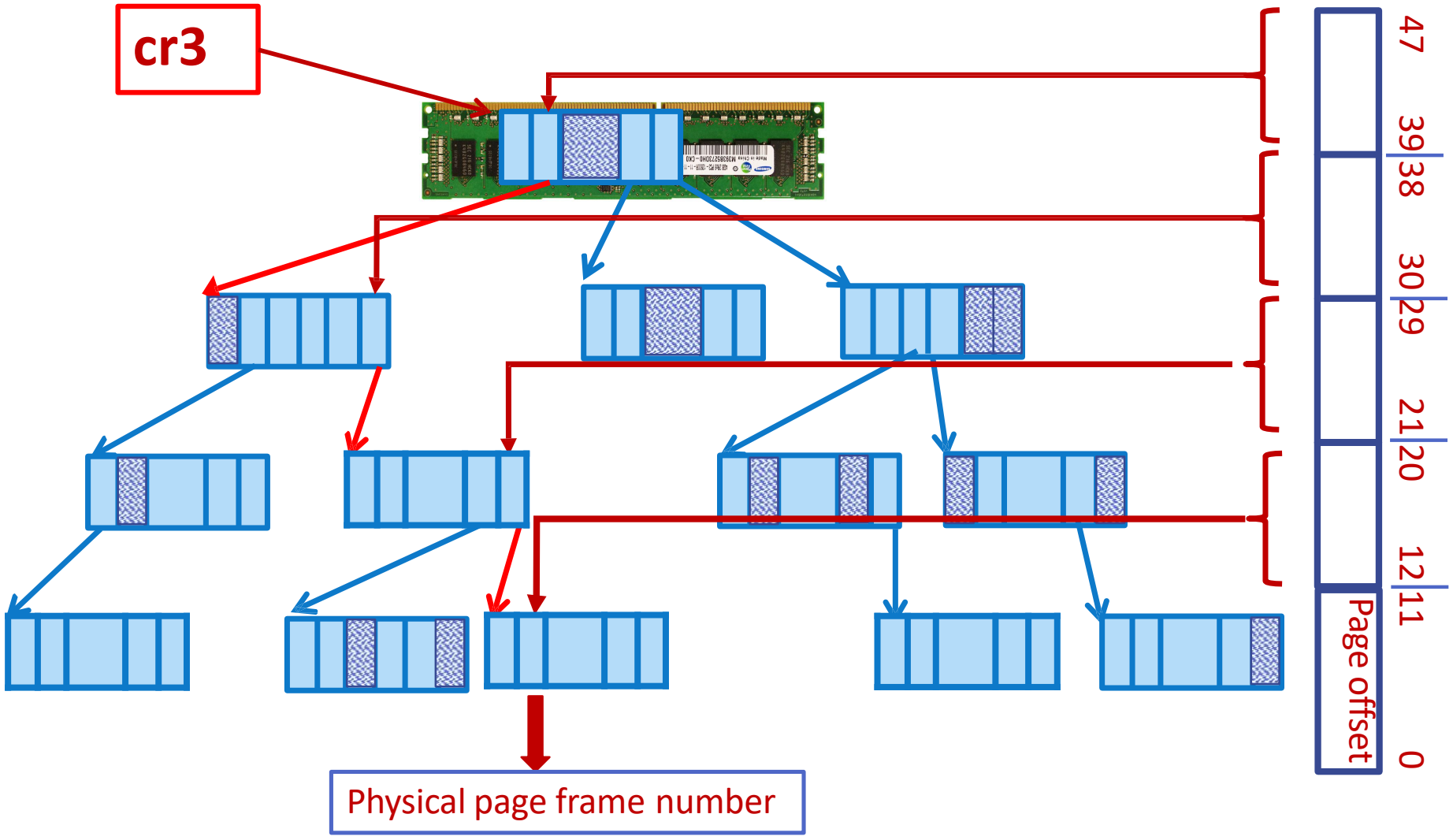

Page table

VP0 → PP8

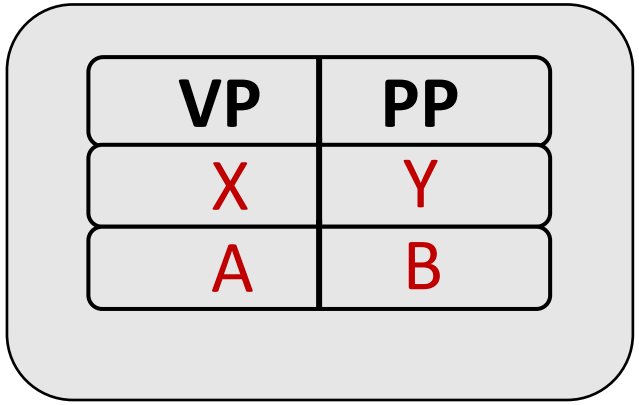
VP1 → PP3

VP2 → PP2

# Multi-level One (four to five-level Walk)

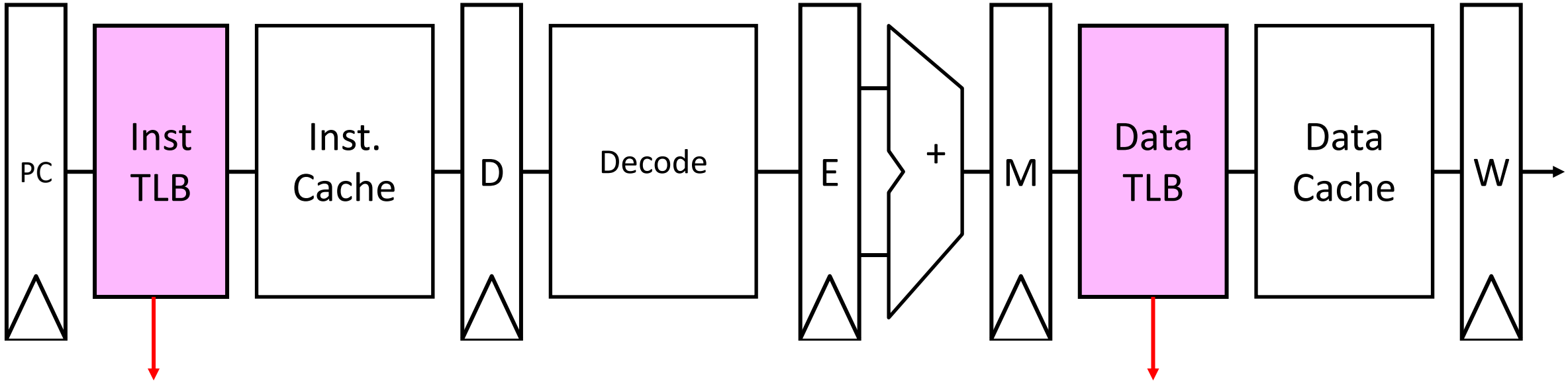


# Can We Cache Translations too? Why Not




Translation Look-aside Buffers (TLBs)

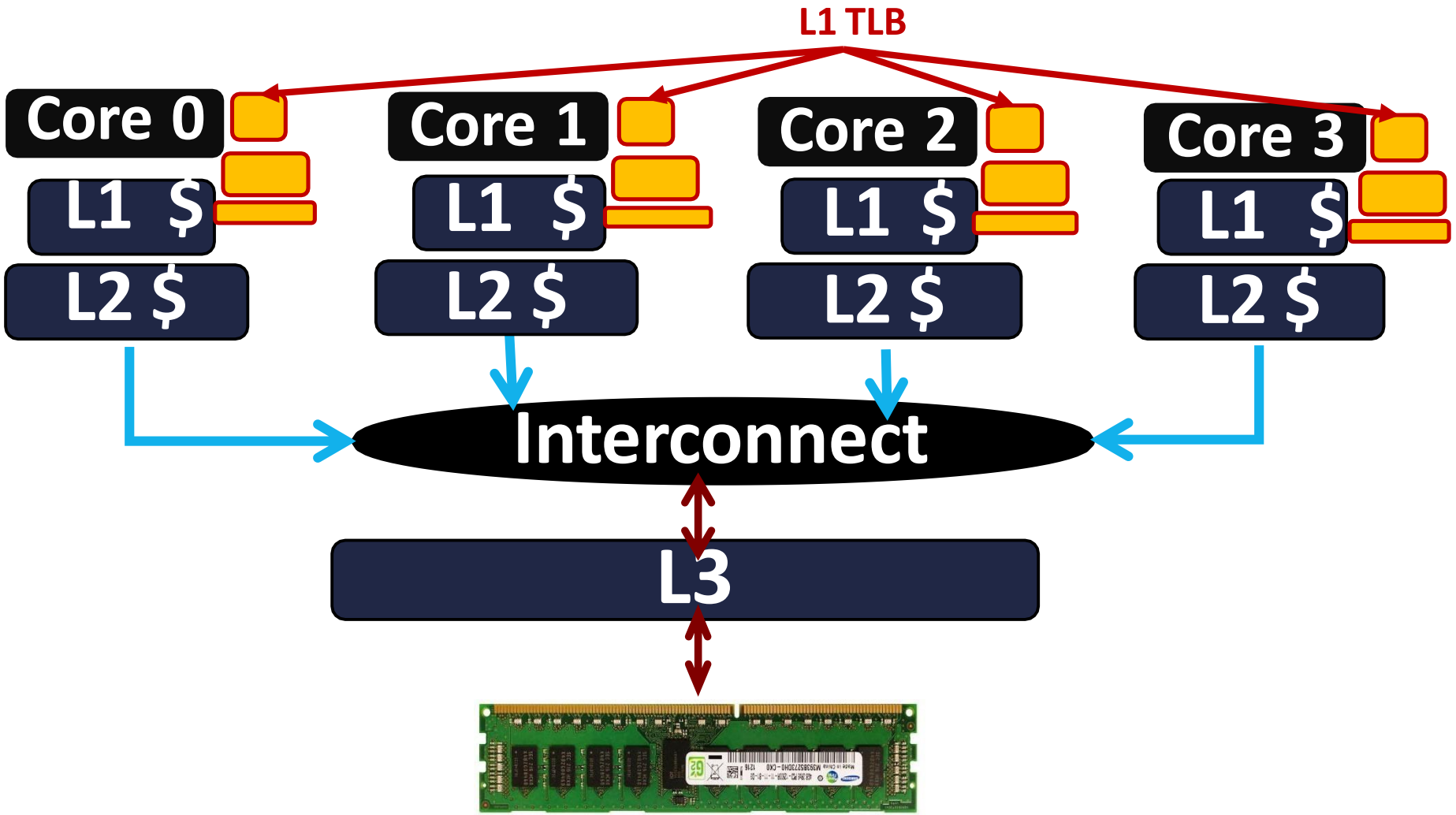
# TLB in your text-book Processor Pipeline



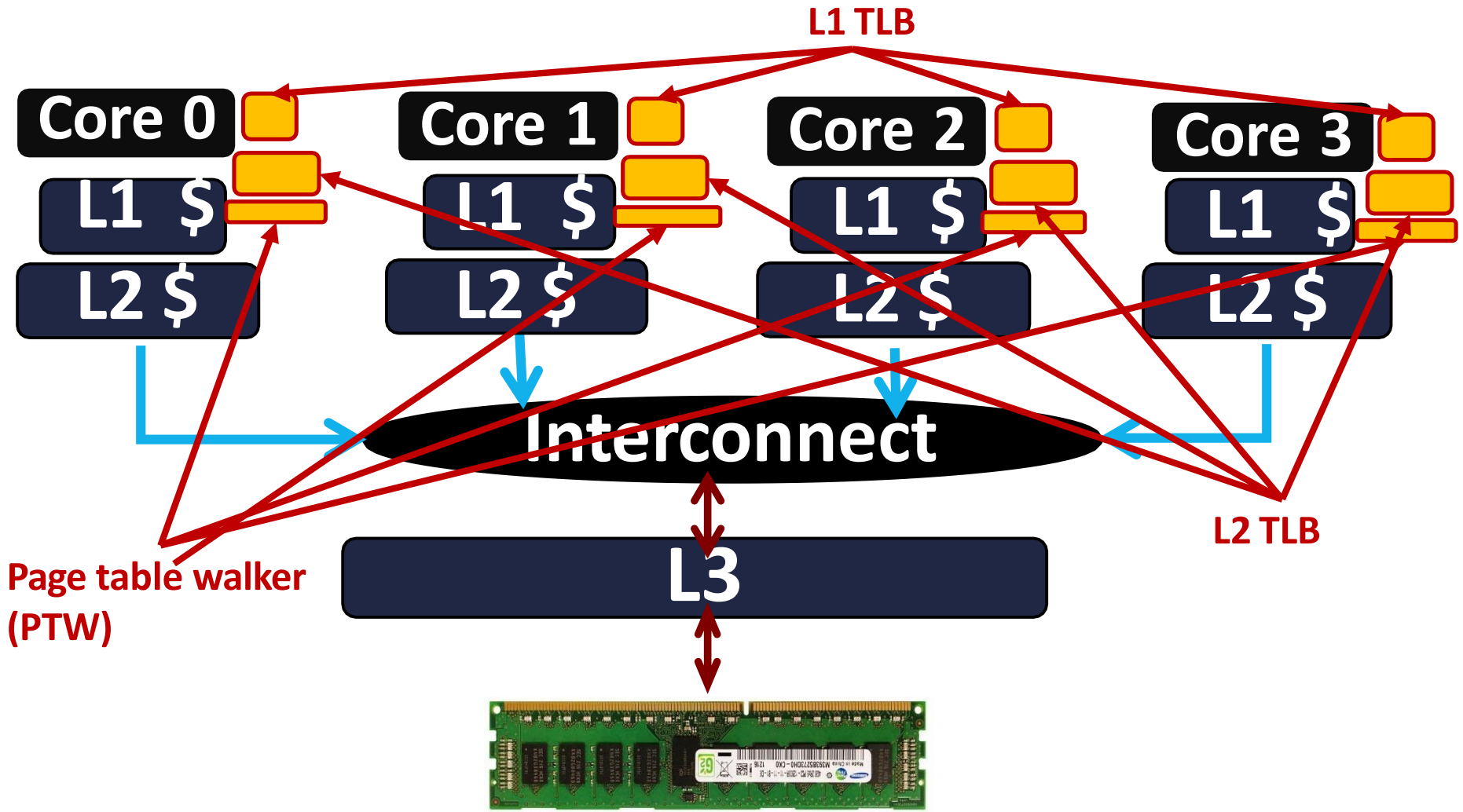

*TLB miss? Page Fault?*
*Protection violation?*

*TLB miss? Page Fault?*
*Protection violation?*

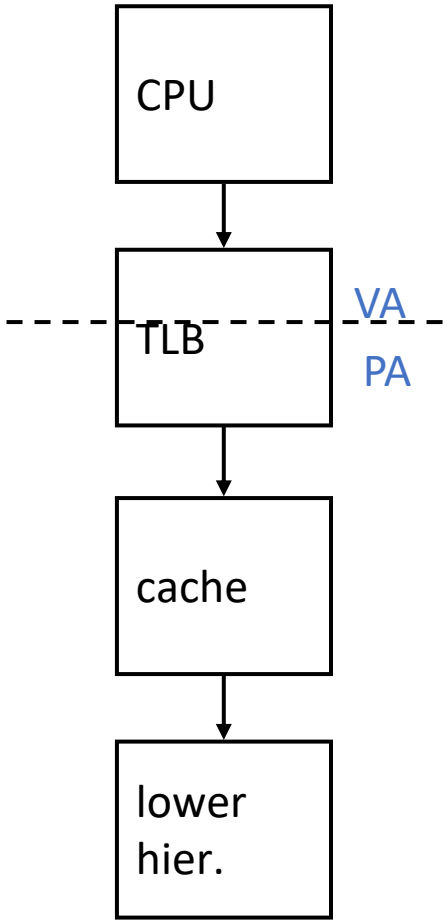
# TLBs and Page-table Walkers (PTWs)



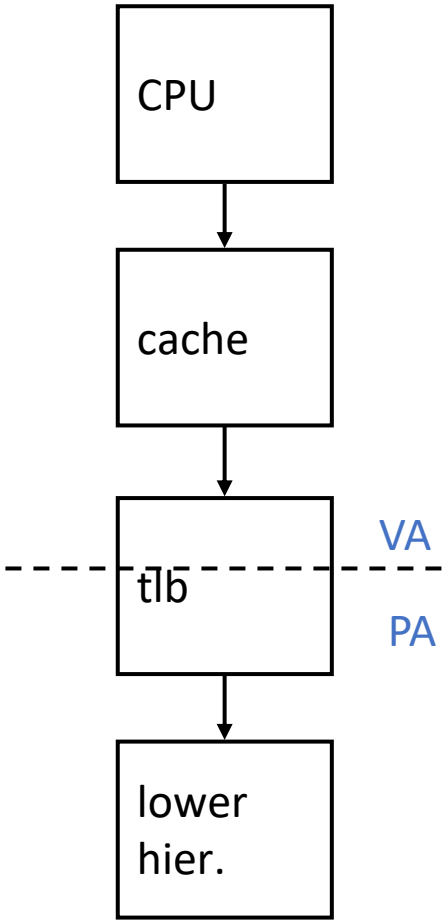
# TLBs and Page-table Walkers (PTWs)



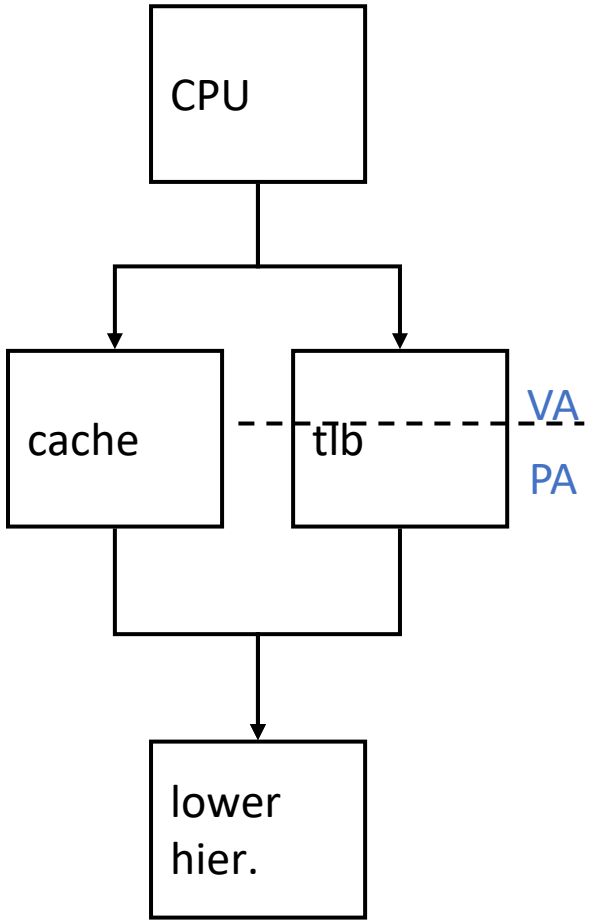
# Virtual and Physical Caches




physical cache




virtual (L1) cache




virtual-physical cache

# PAUSE

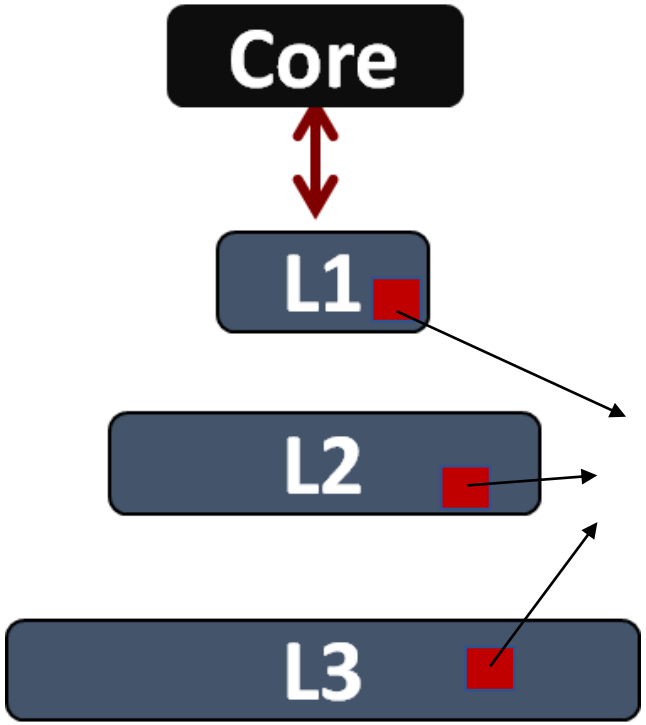
# Ideal Back-end (L1D hit rate of 100%)


L1 hit rate of 100% (a dream ☺)

RIP Memory wall ☺

60% performance gap ☹

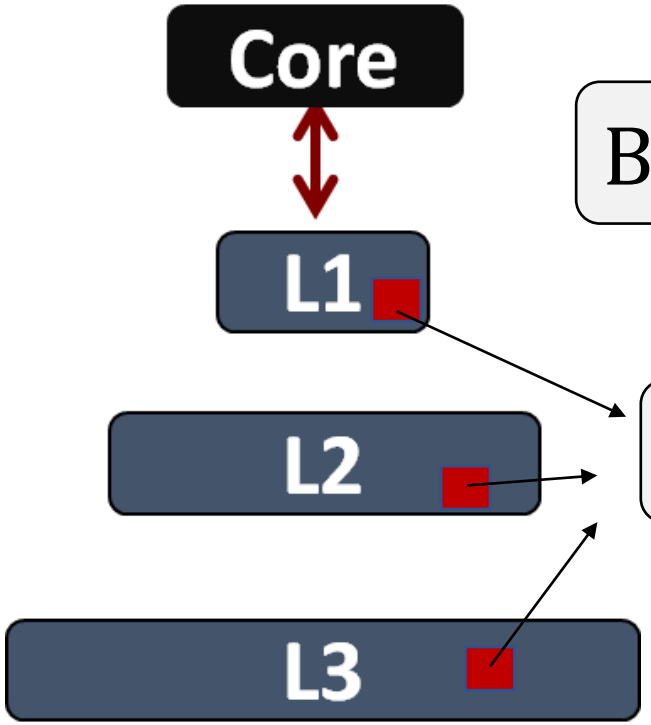
# High Performance Cache Hierarchy




60% performance gap

Better cache content management techniques

# High Performance TLB Hierarchy ☺




Better TLB content management techniques

50% away from the ideal TLB ☺

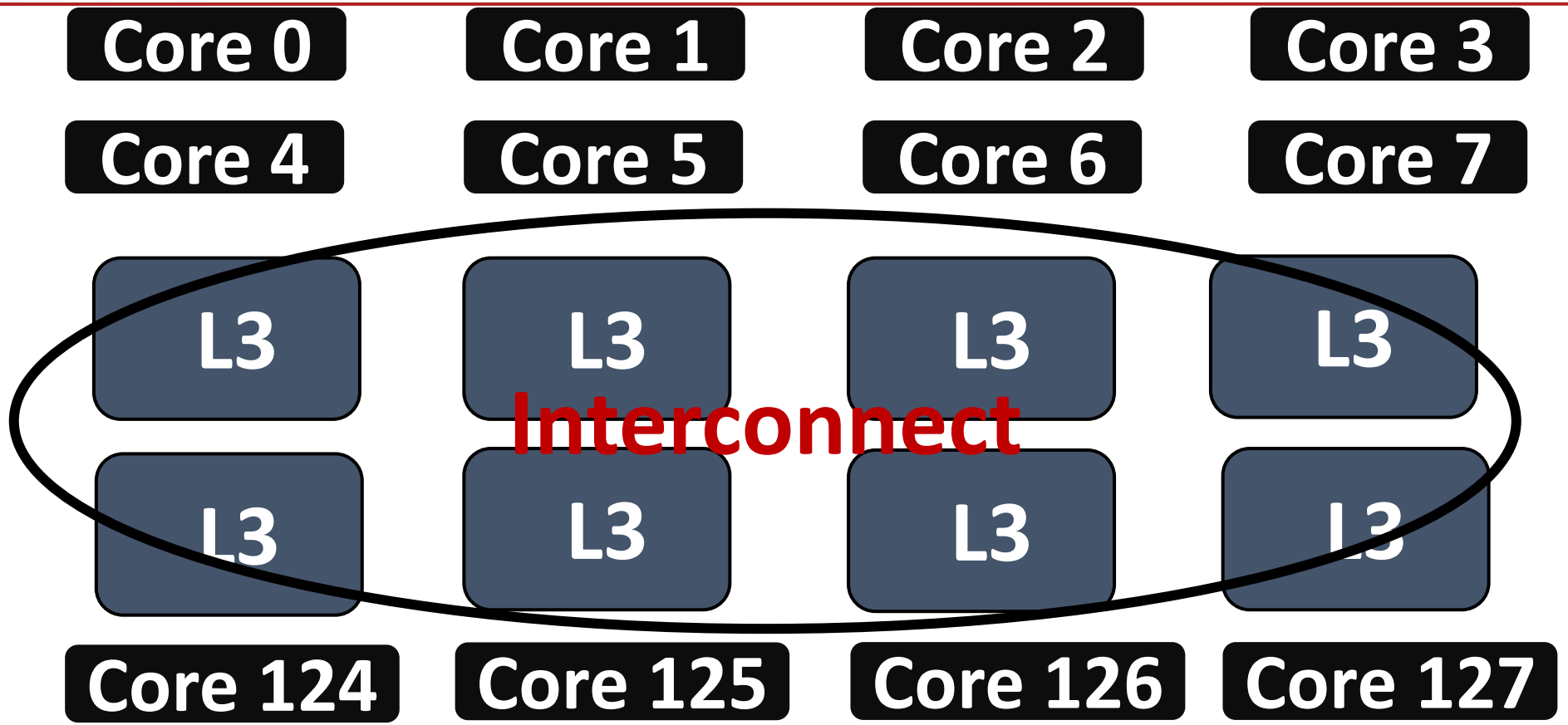
Extremely important for memory-intensive irregular application domains

# Domain Specific Memory Hierarchies

Graph processing bottlenecks

ML/CV application bottlenecks

# Microarchitecture for Many-core Systems




# Bottlenecks: Two Fundamental principles

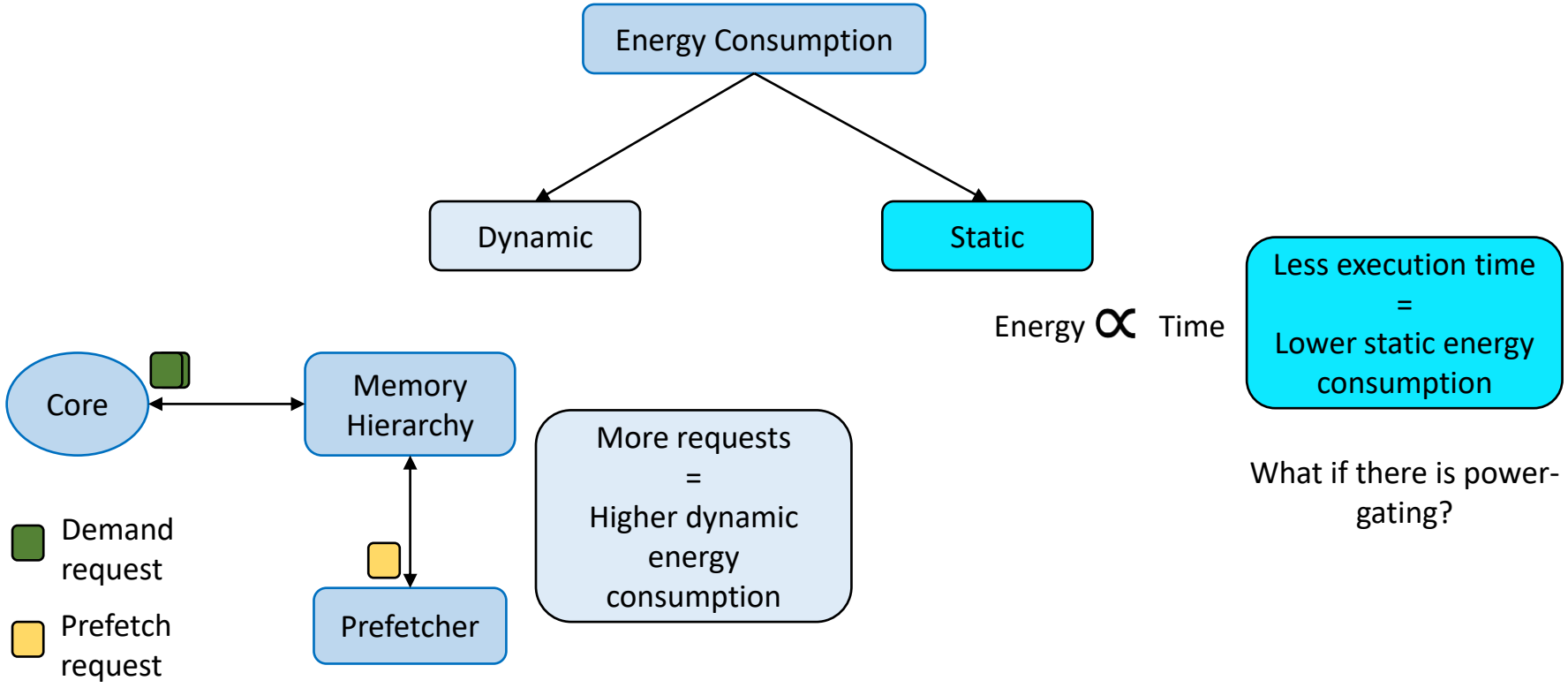
LATENCY

BANDWIDTH

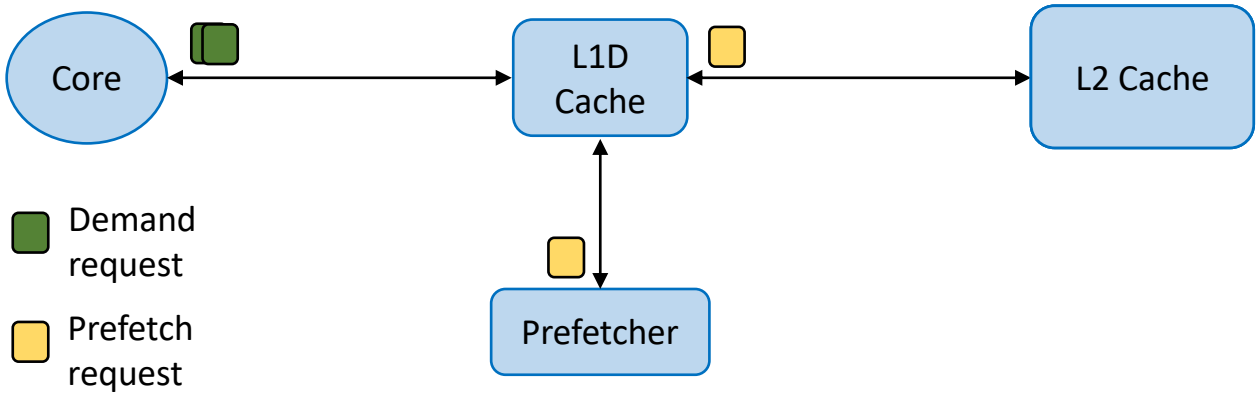
MAY BE CAPACITY?

# Fairness and QOS? Think about Cloud providers

# What about Power/Energy



# Power Gating




More requests in the memory hierarchy can lead to higher static energy consumption.

# summary

Latency

Bandwidth (may be capacity)

Fairness

QOS

Energy/Power

# Thanks