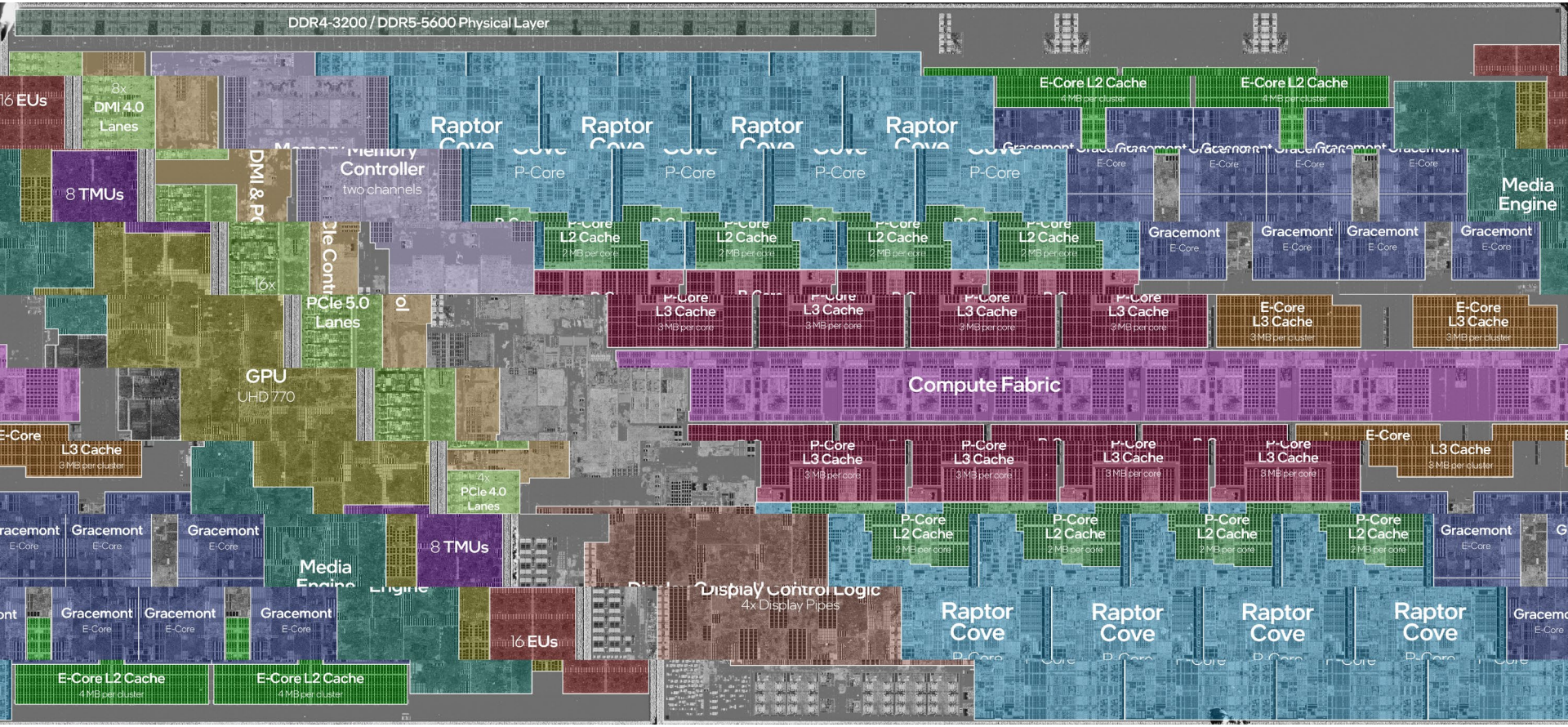
Contemporary Computer Architecture TDSN13

LECTURE 2 – INTRODUCTION CONT

ANDREAS AXELSSON (ANDREAS.AXELSSON@JU.SE)

Intel Core i9-13900K die shot

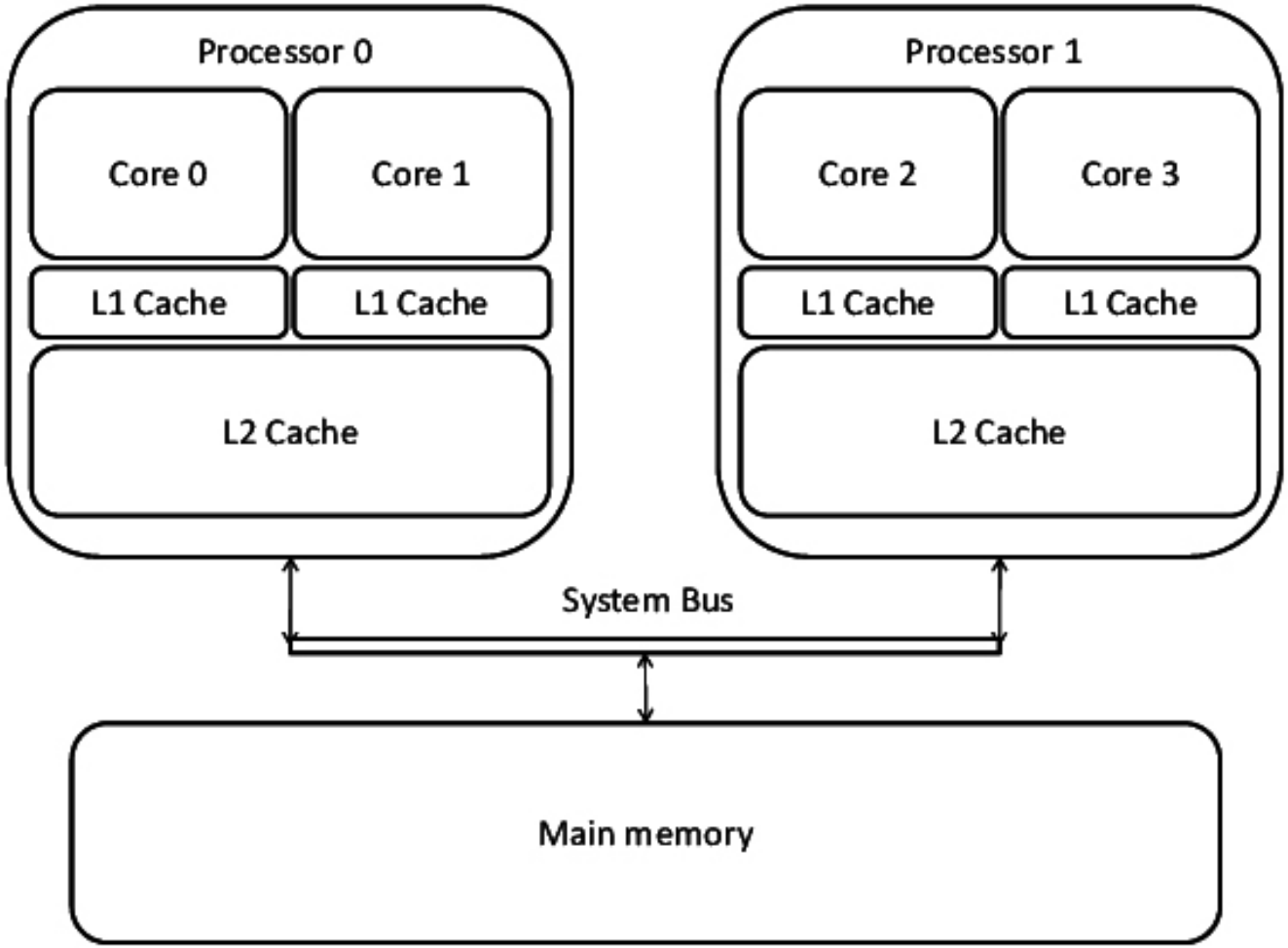


By JmsDoug and Fritzchens Fritz - Own work based on: Intel@intel7(10nmESF)@RaptorLake@RPL(8P+16E)@i9-13900K@ES DSCx05 poly@5xExt.jpg

Flynn's Taxonomy

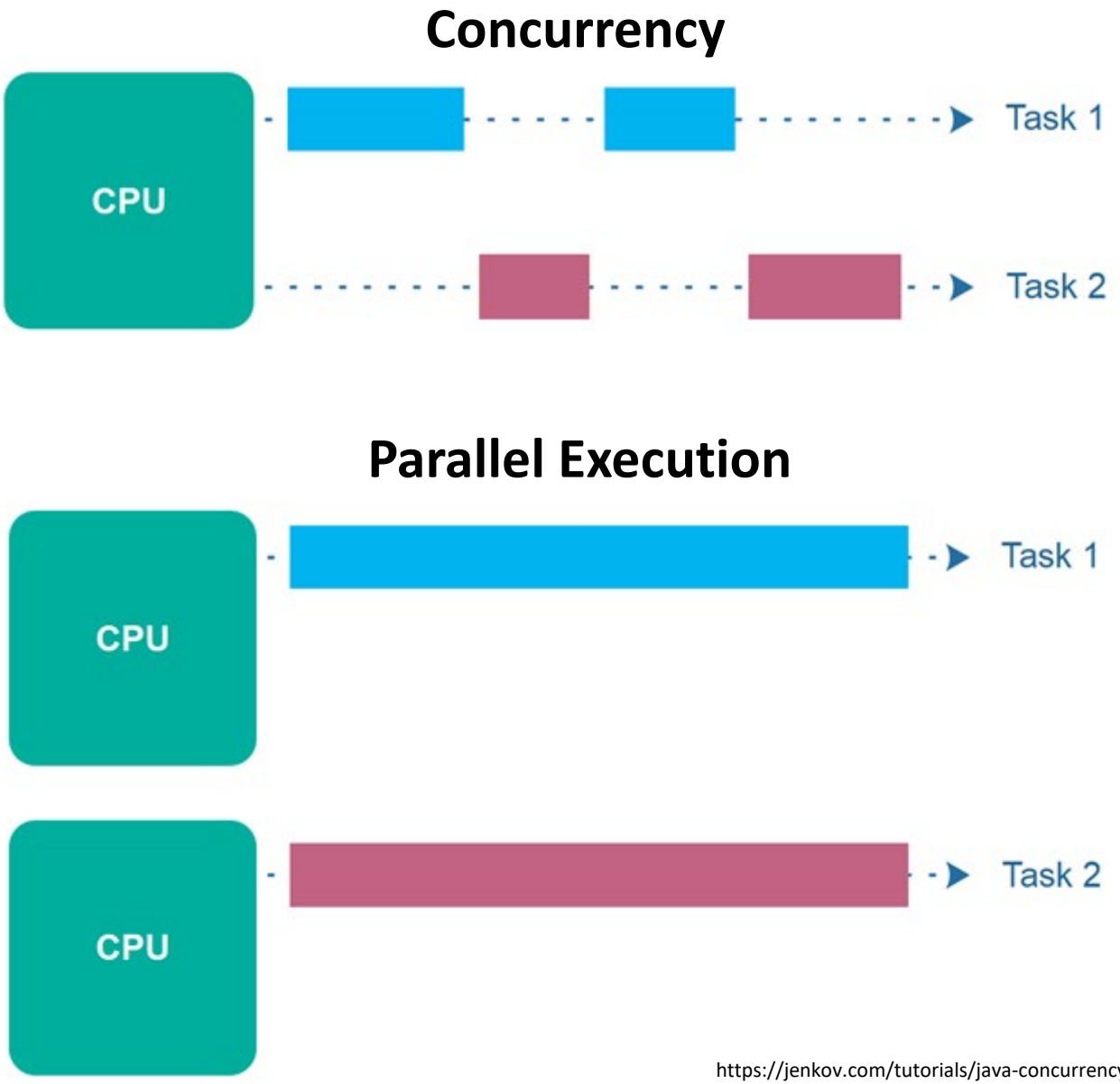
		Instruction	
		Single	Multiple
Data	Single	SISD Single instruction single data	MISD Multiple instruction single data
	Multiple	SIMD Single instruction multiple data	MIMD Multiple instruction multiple data

Multi-core / Multi-processor



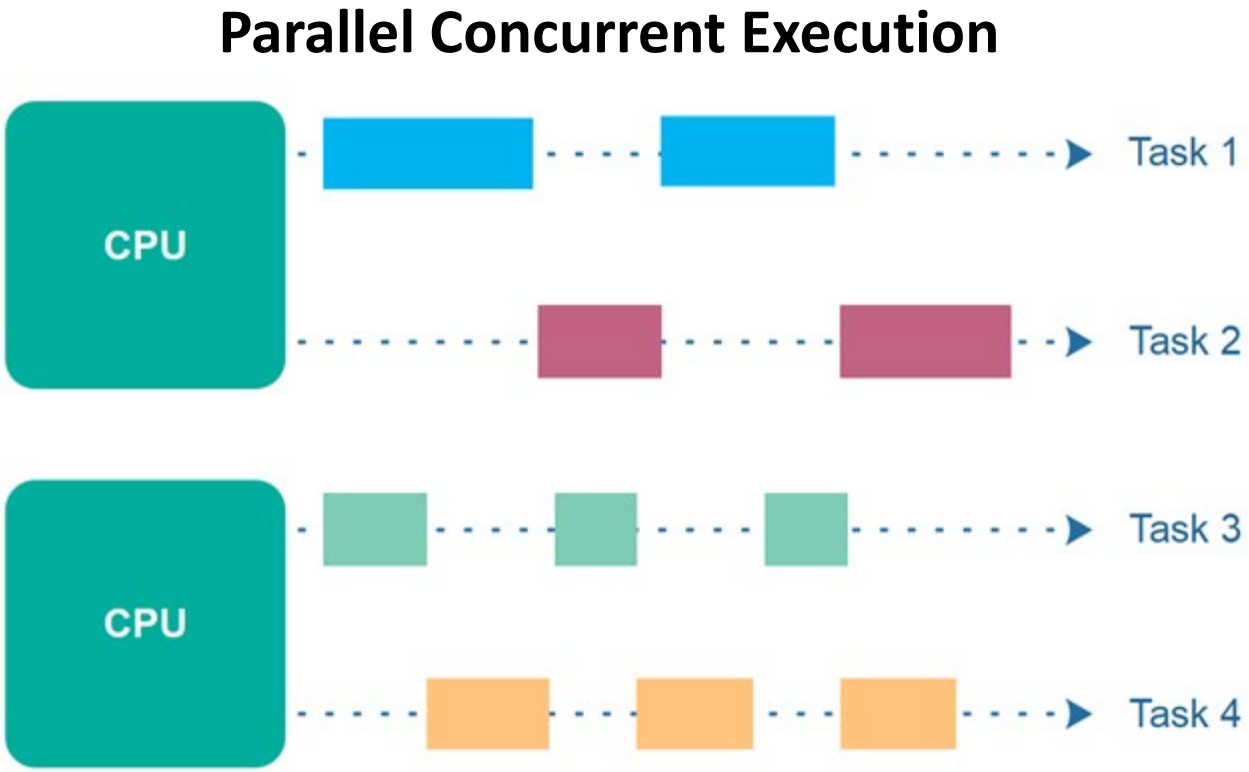
https://www.itrelease.com/2020/07/advantages-and-disadvantages-of-multi-core-processors/

Concurrency vs Parallelism



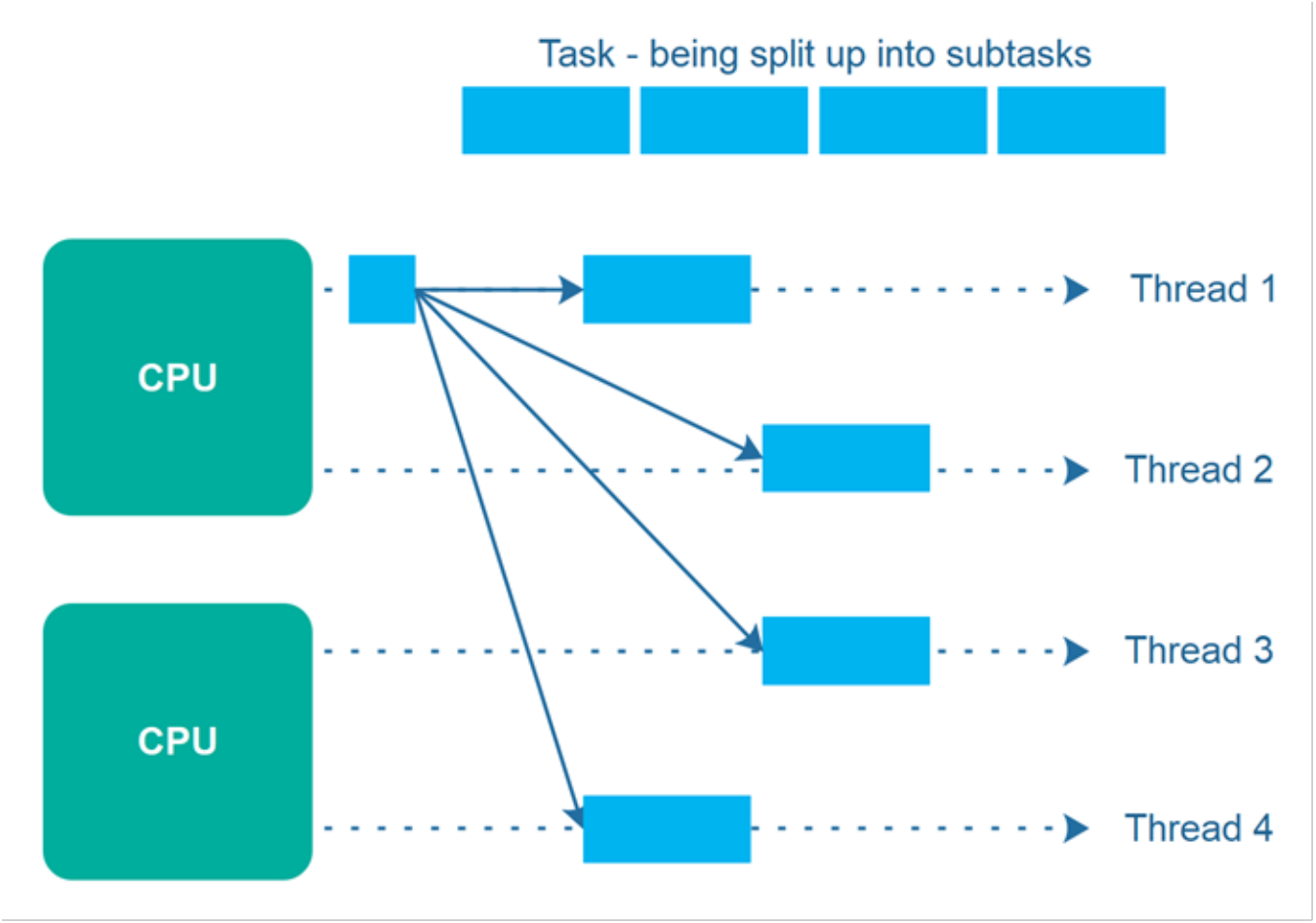
https://jenkov.com/tutorials/java-concurrency/concurrency-vs-parallelism.html

Concurrency vs Parallelism



https://jenkov.com/tutorials/java-concurrency/concurrency-vs-parallelism.html

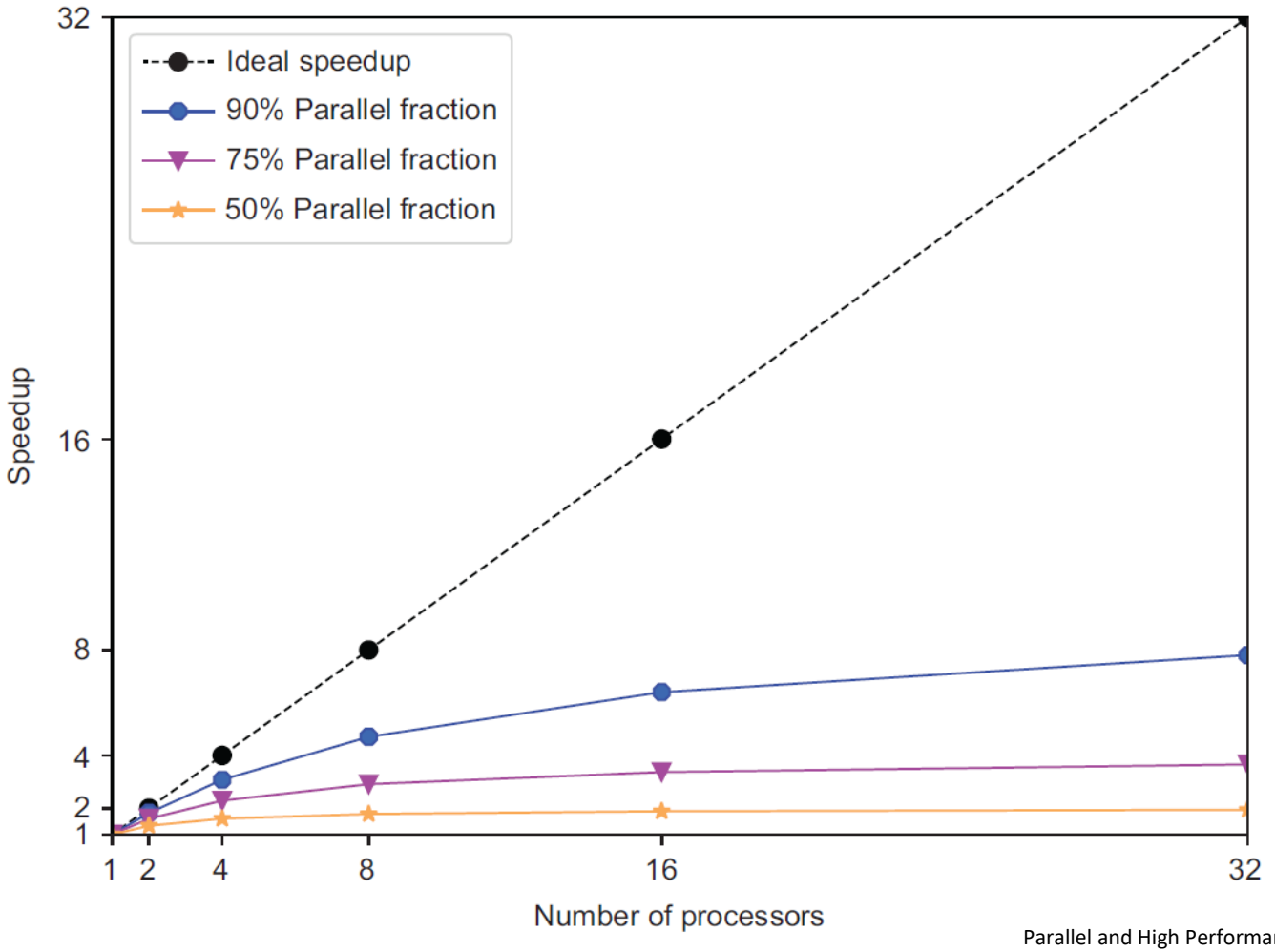
Concurrency vs Parallelism



” Concurrency is about **dealing** with lots of things at once. Parallelism is about **doing** lots of things at once.”

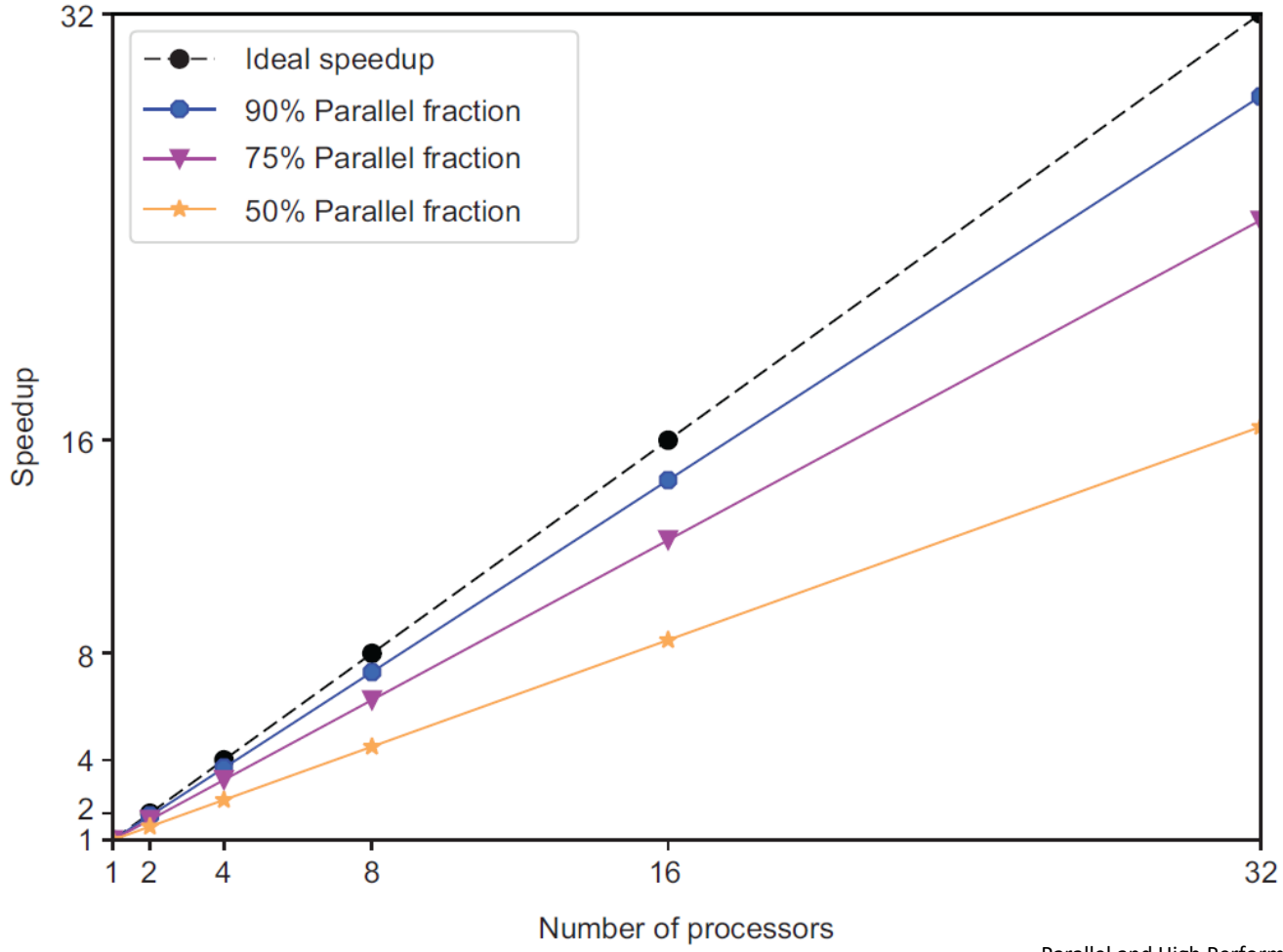
Amdahl's Law

$$SpeedUp(N) = \frac{1}{S+\frac{P}{N}} \; where \; \text{S+P=1}$$

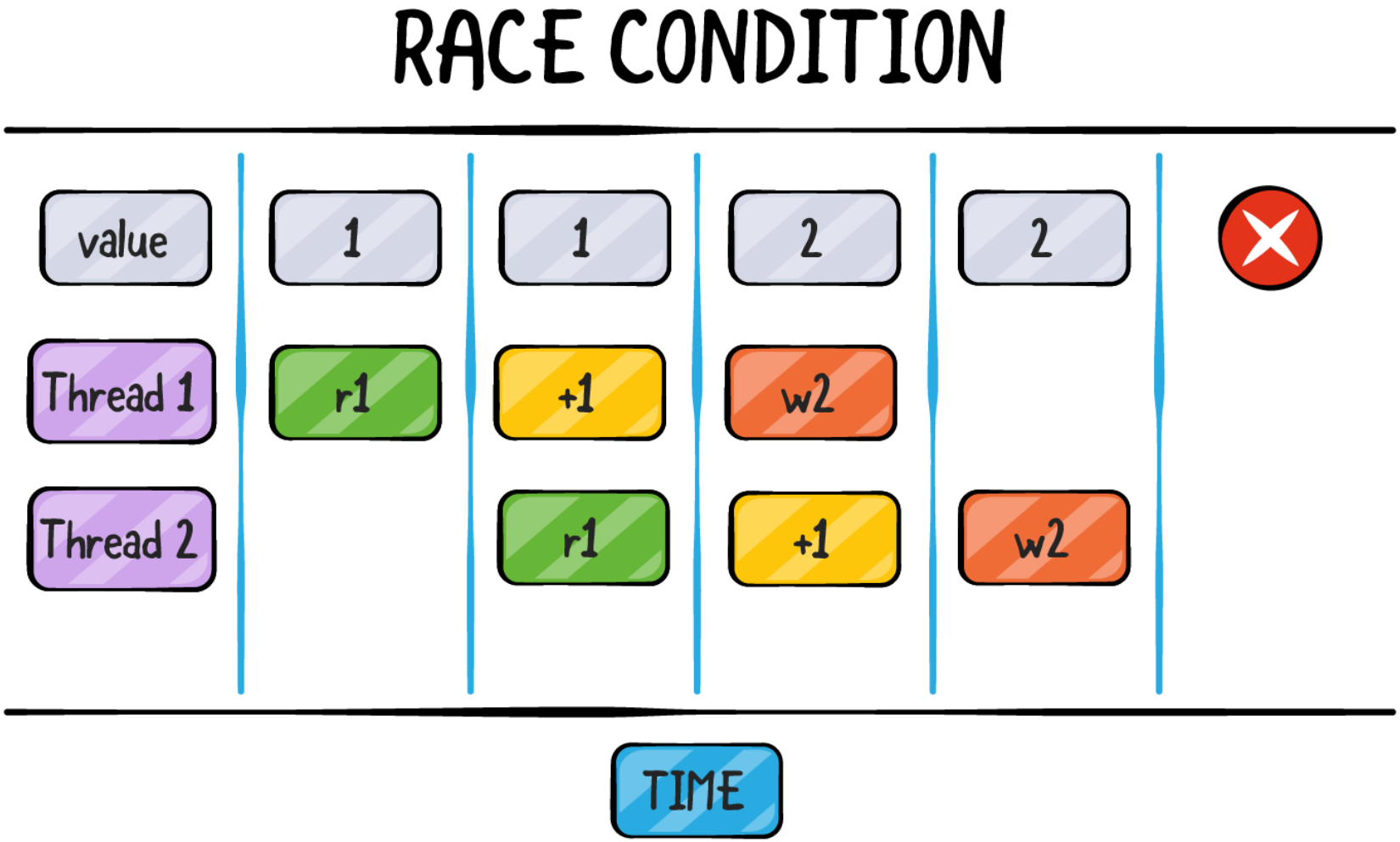


Gustafson-Barsis's Law

$$SpeedUp(N) = N - S * (N - 1) \ \ where \text{ S is a fraction}$$

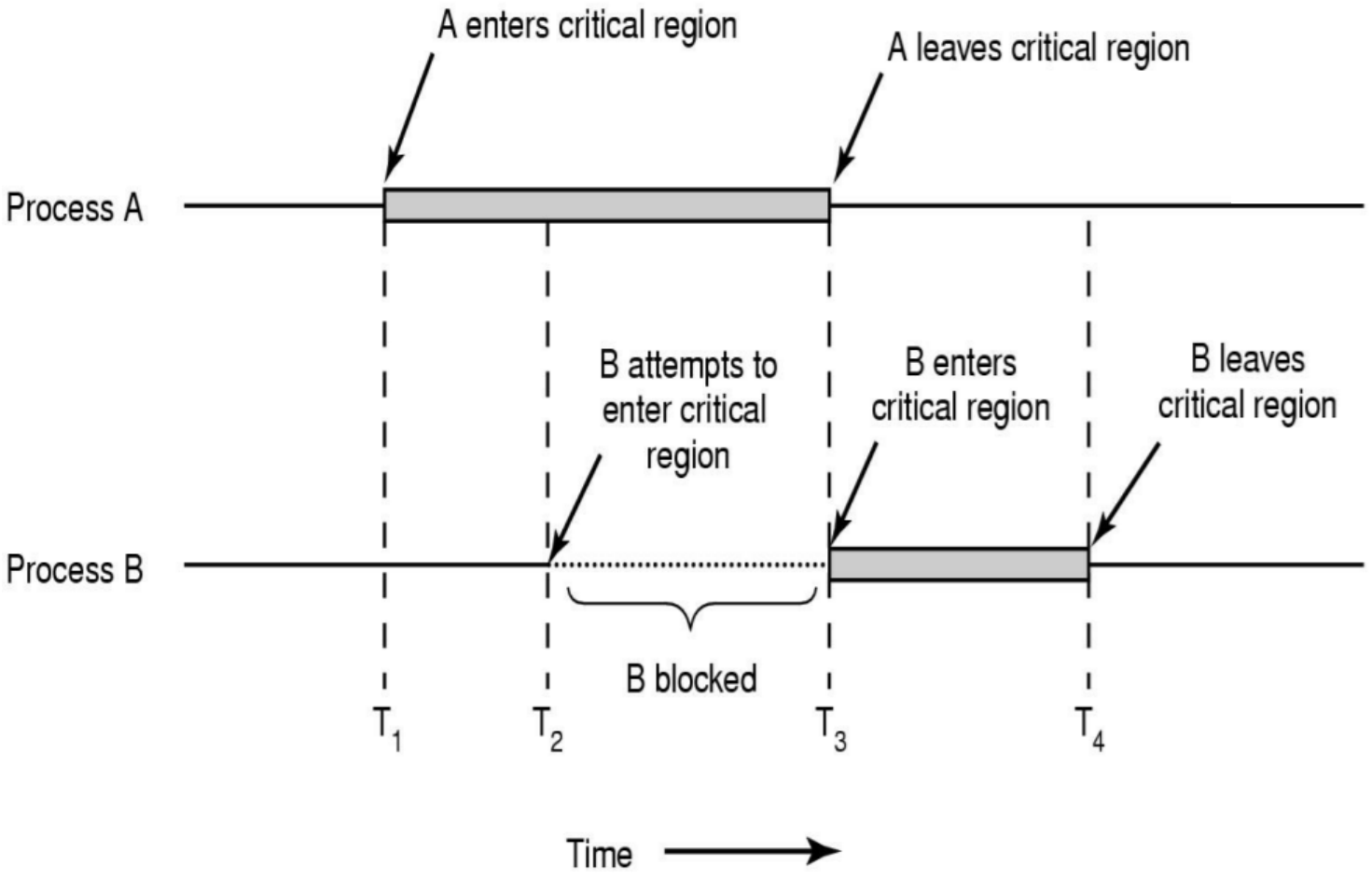


Race Conditions



https://www.raywenderlich.com/books/concurrency-by-tutorials/v2.0/chapters/5-concurrency-problems

Critical Regions



Mutual Exclusion

Mutex

- Tries to take a lock (mutex). If already taken, move thread to blocked list and run thread from ready list, otherwise lock the mutex. Do critical section. Return lock.

- Solutions often depends on ’Atomic instruction’ (Test and Set)

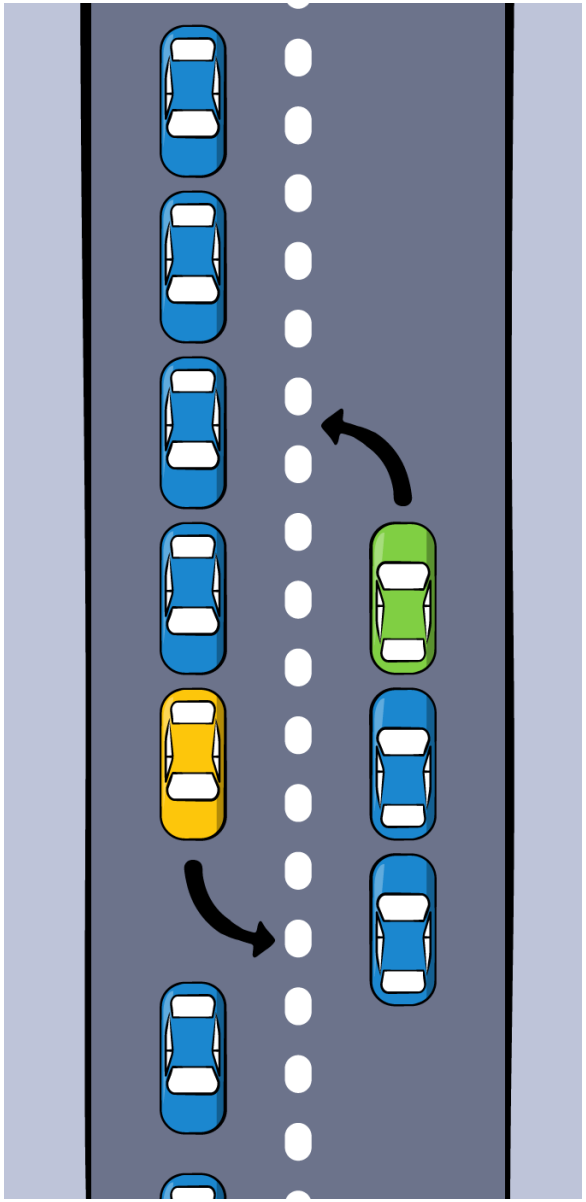
```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

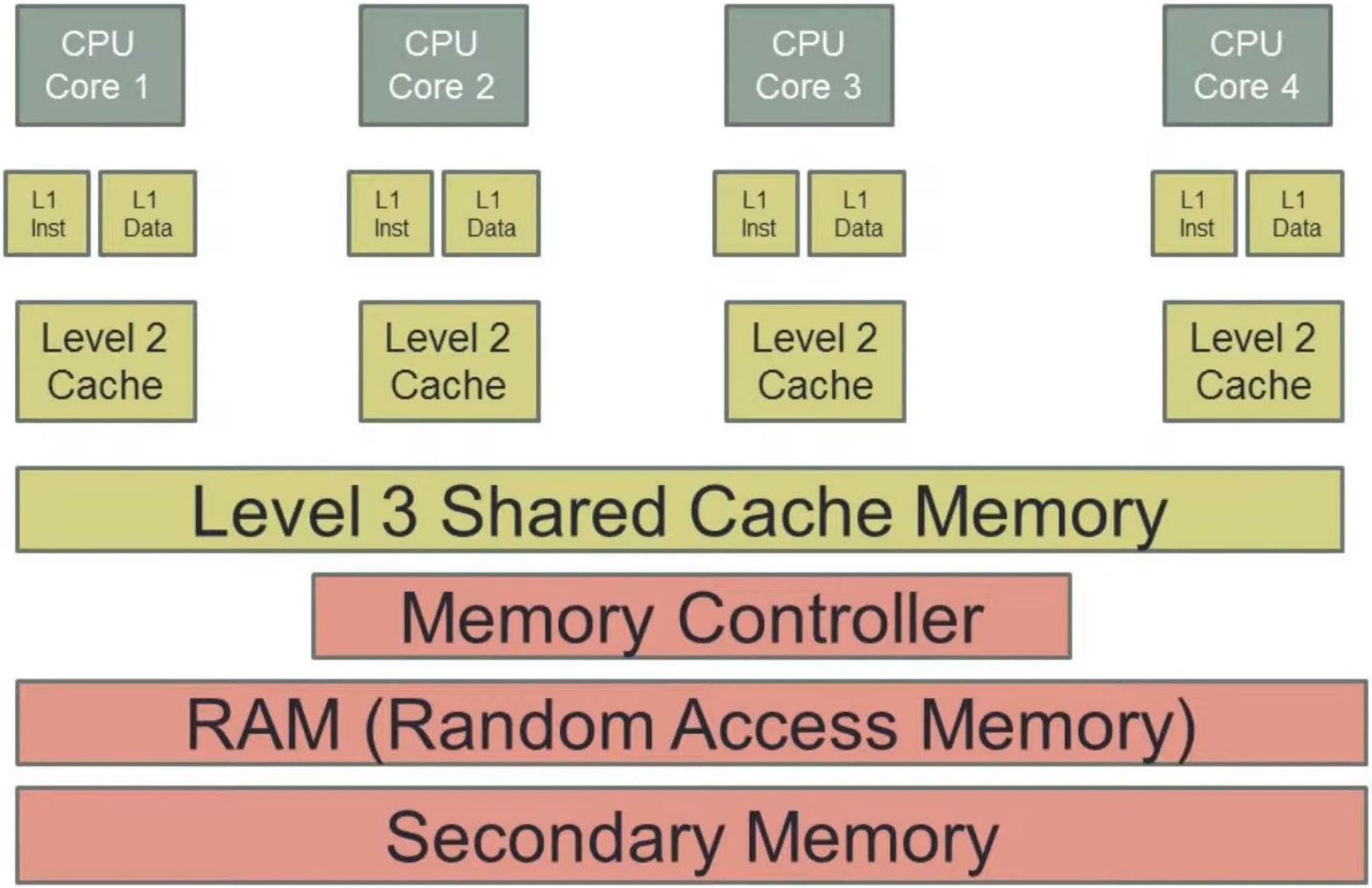
Deadlock

When two or more processes stop making progress *indefinitely* because they are all waiting for each other to do something.

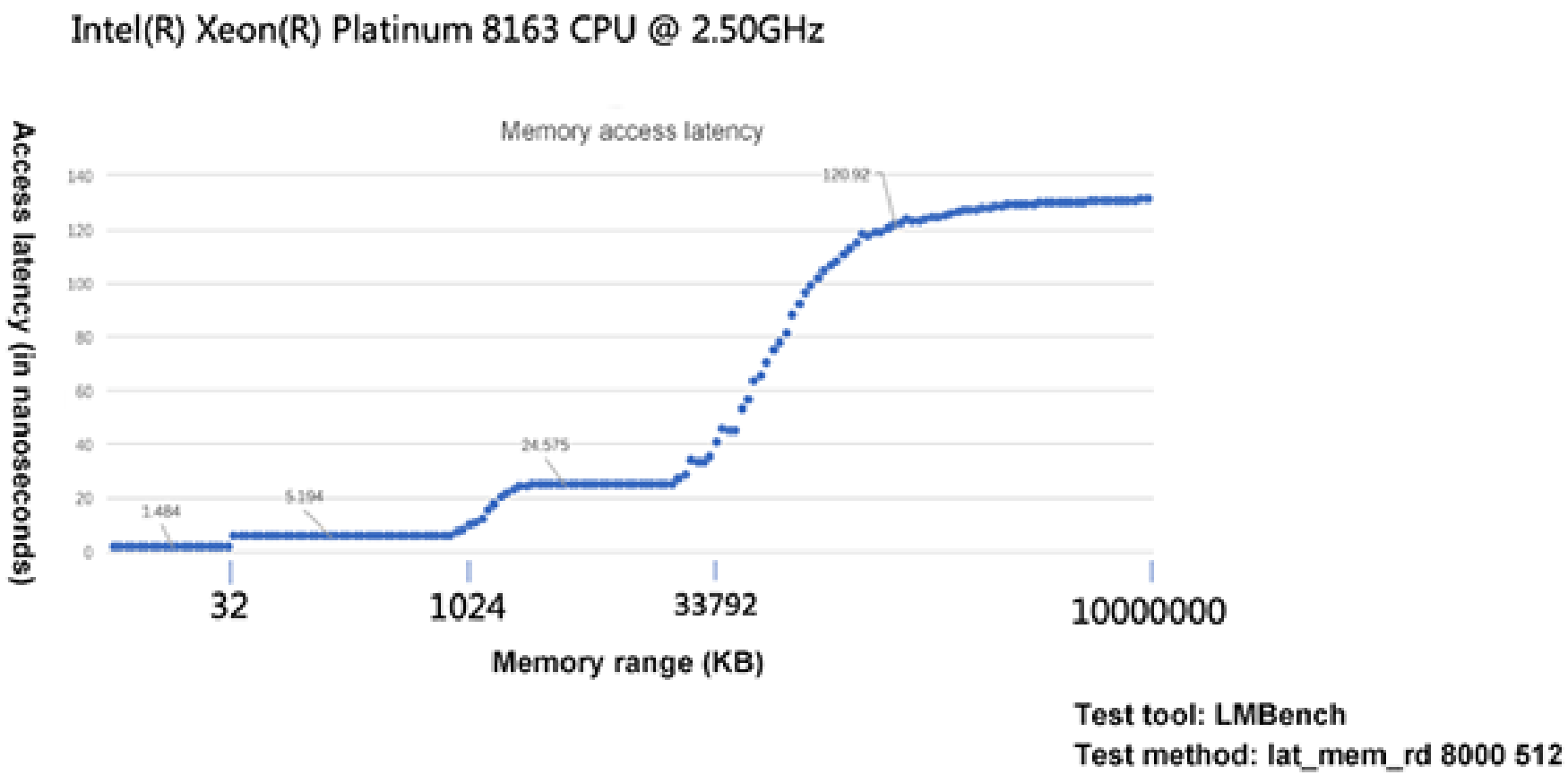
Deadlock



Cache memory



Cache performance differences



https://www.alibabacloud.com/blog/understanding-memory-performance_594708

Cache Organization

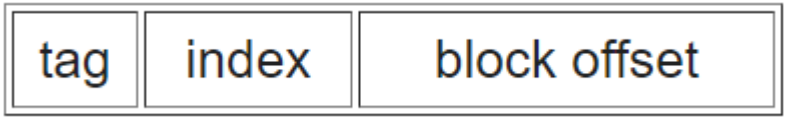
Cache memory divided into **cache lines** or **cache blocks**

tag – is part of the adress

data block – containt the actual memory content

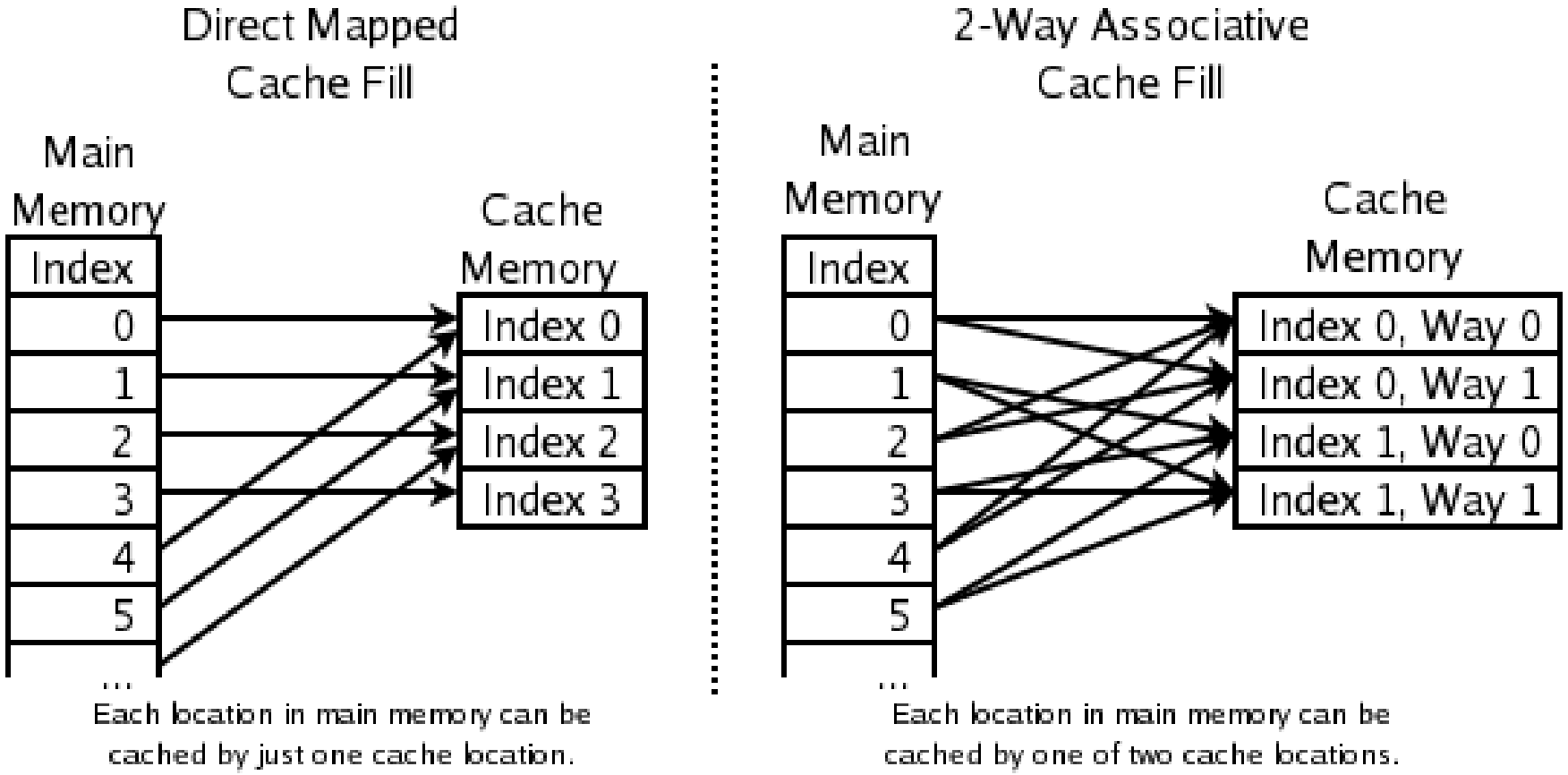
flag bits – containts flags such as if memory has been changed (dirty)

To find correct cache line/block the memory adress that is accessed by the CPU is used as a key.



https://en.wikipedia.org/wiki/CPU_cache

Direct Mapped/2-Way Associative



Cache Organization

- Direct mapped cache – good best-case time, but unpredictable in the worst case
- Two-way set associative cache
- Two-way skewed associative cache
- Four-way set-associative cache
- Eight-way set-associative cache, a common choice for later implementations
- 12-way set associative cache, similar to eight-way
- Fully associative cache – the best miss rates, but practical only for a small number of entries

Each of the methods have different rates of **cache hits** or **cache misses**

Replacement and Write Policies

Replacement Policies:

For caches that allow multiple possible locations (like fully associative and set-associative), there's a question of which line to replace when bringing in new data. Common policies include:

Least Recently Used (LRU): Replace the block that hasn't been accessed for the longest time.

FIFO (First-In, First-Out): Replace the oldest block in the set or cache.

Random: Randomly select a block to replace.

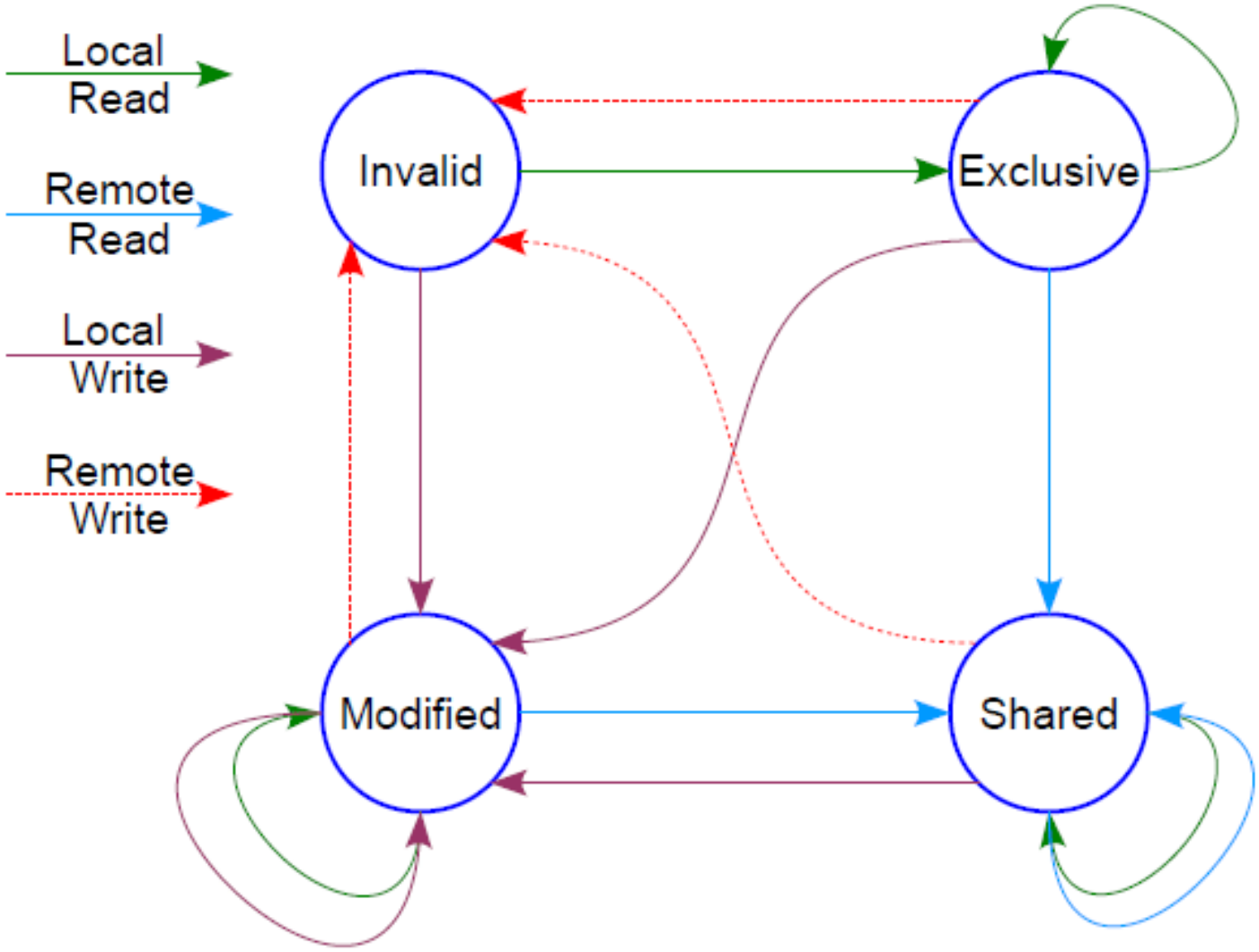
Write Policies:

Decide how to handle data writes:

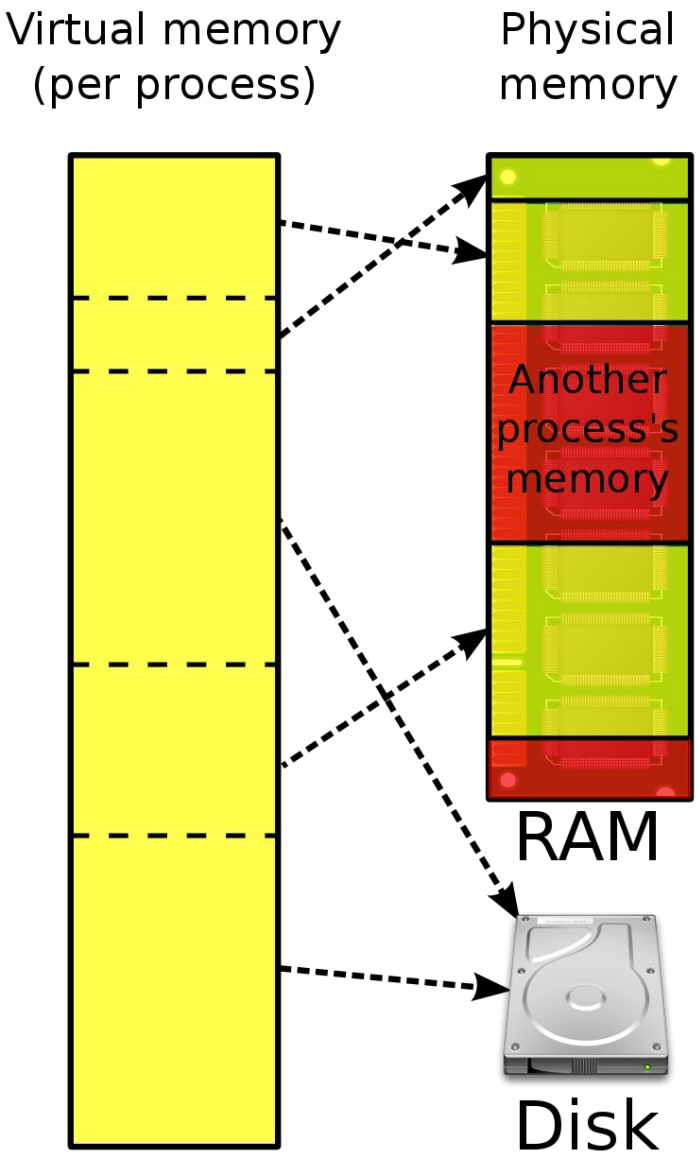
Write-Through: Every write to the cache immediately writes to main memory.

Write-Back (or Copy-Back): Writes are made to the cache alone, and data is only written back to main memory when the cache line is replaced.

MESI Cache Coherence Protocol

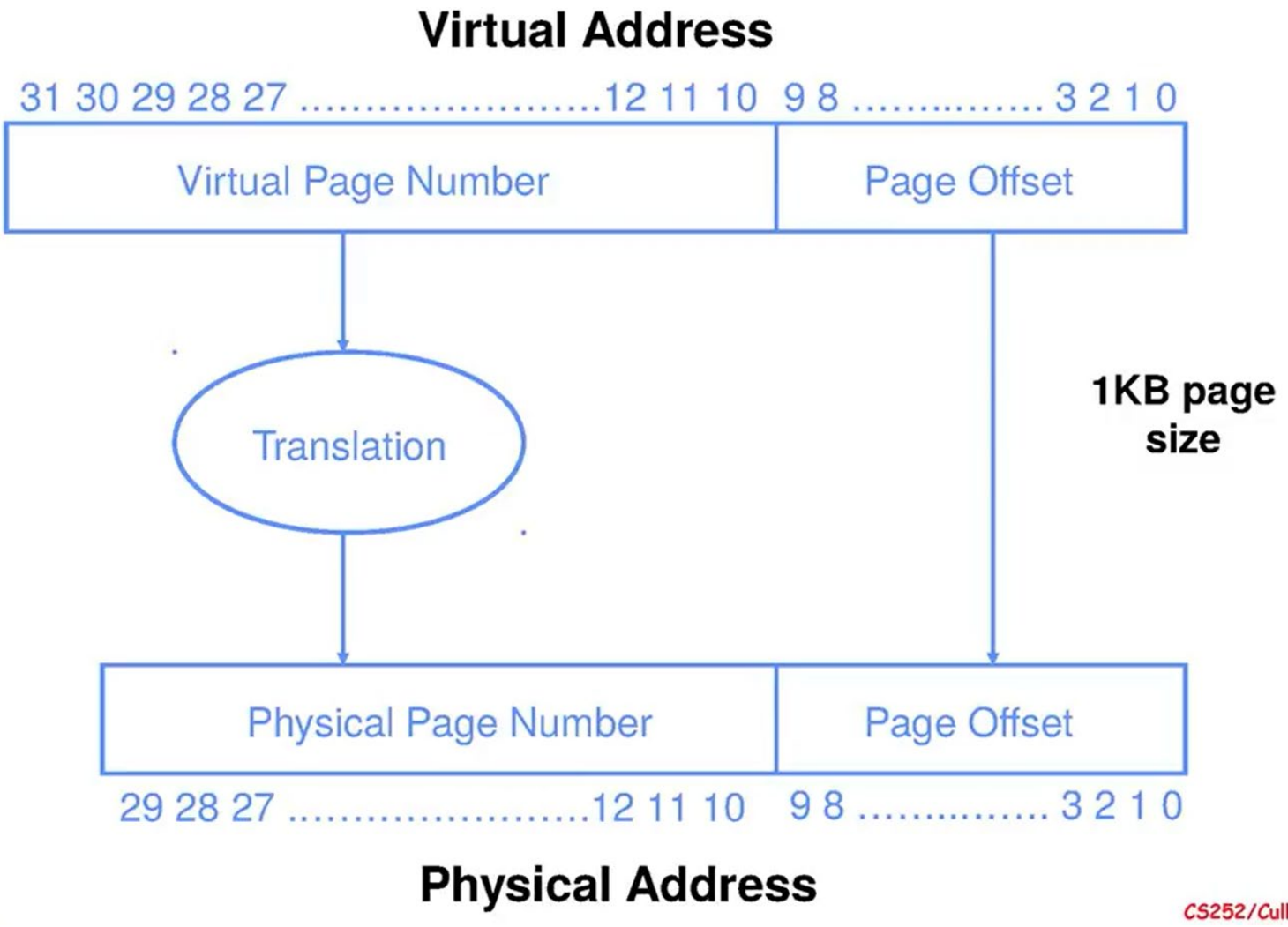


Virtual Memory



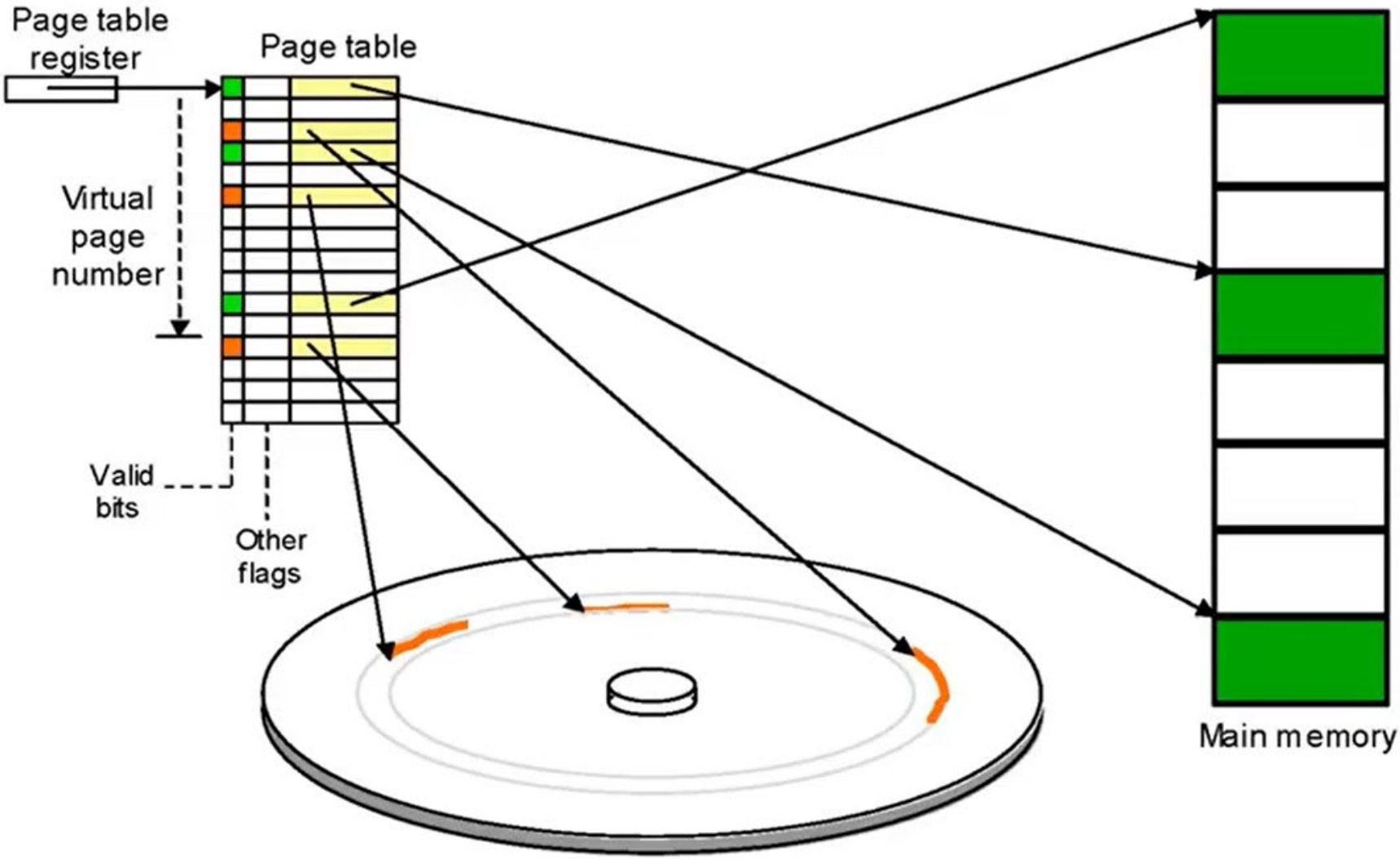
https://en.wikipedia.org/wiki/Virtual_memory

Virtual Memory



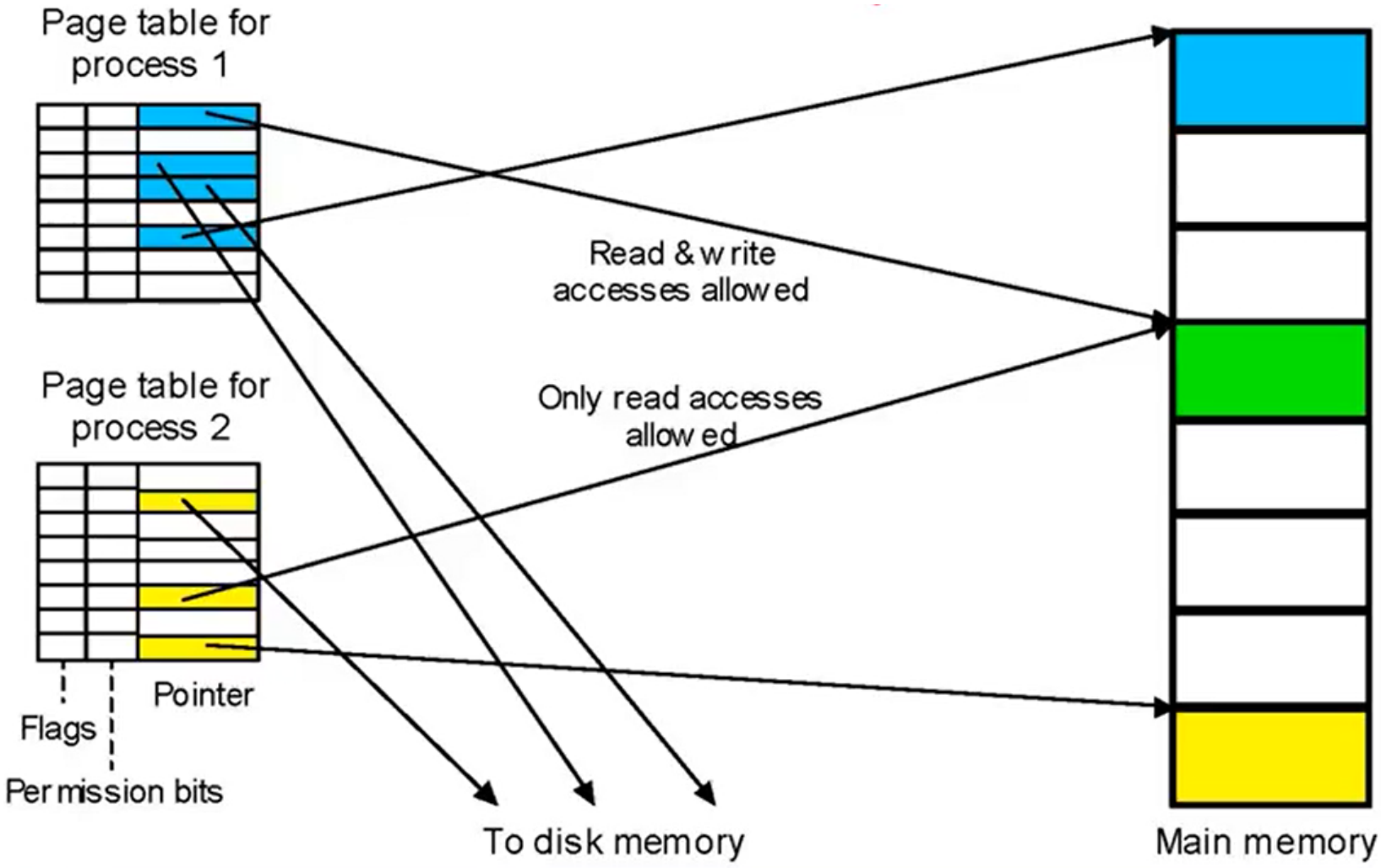
https://www.youtube.com/watch?v=ICnr87A5m7c

Virtual Memory



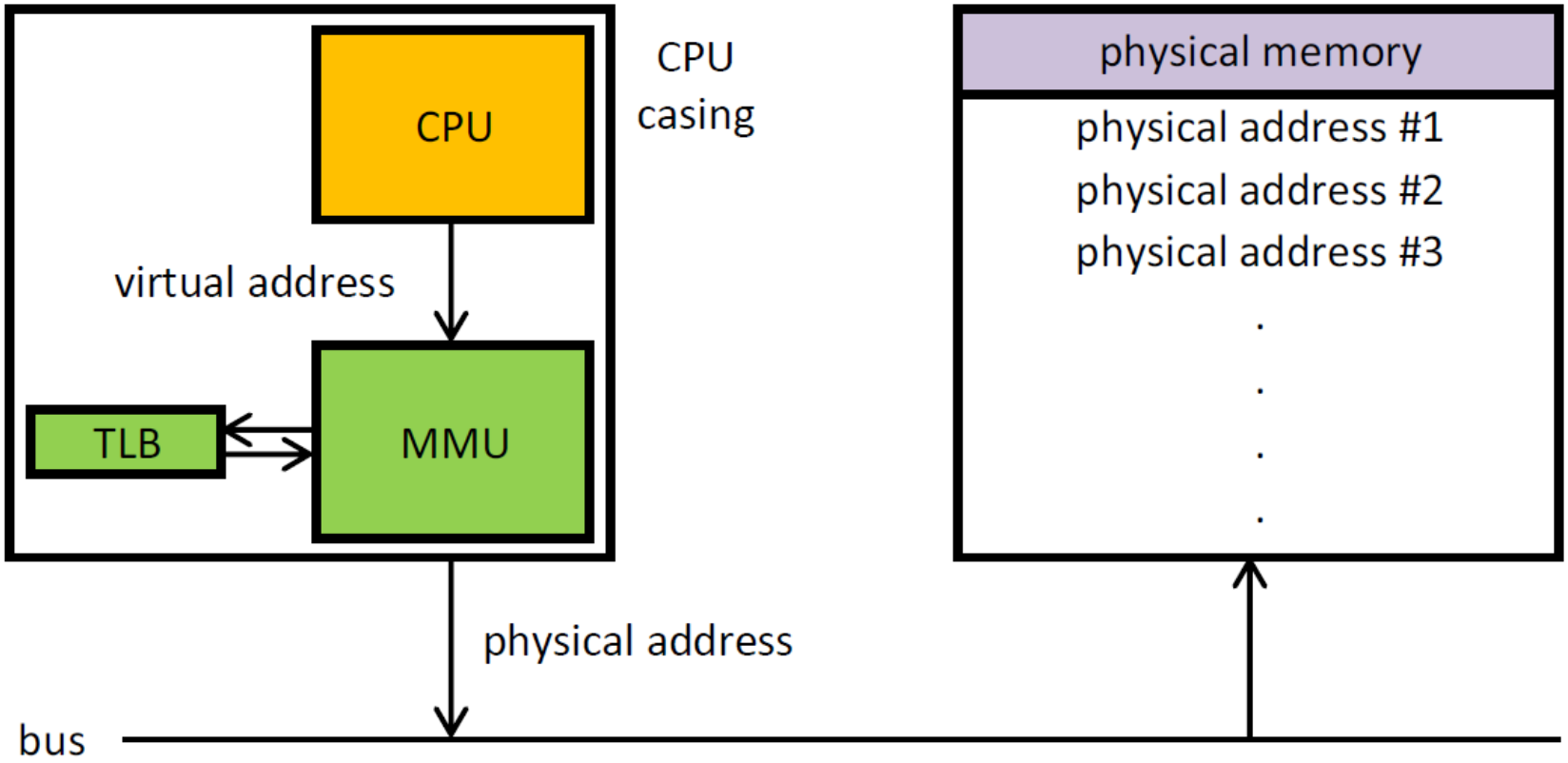
https://www.youtube.com/watch?v=ICnr87A5m7c

Virtual Memory



https://www.youtube.com/watch?v=ICnr87A5m7c

Virtual Memory



CPU: Central Processing Unit

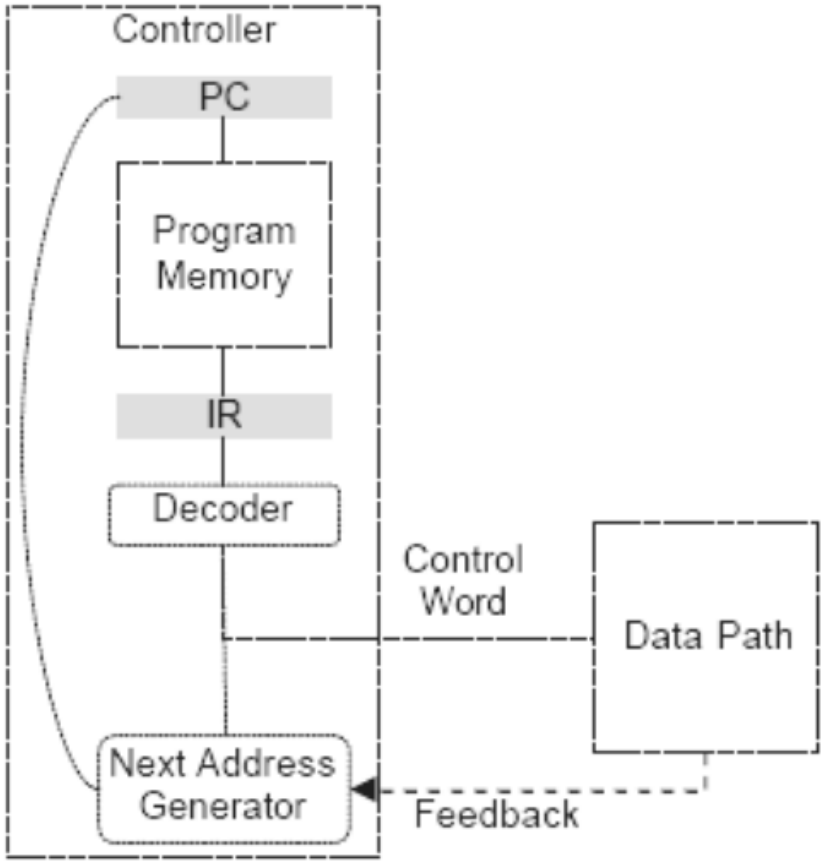
MMU: Memory Management Unit

TLB: Translation lookaside buffer

(TLB can be called an address-translation cache)

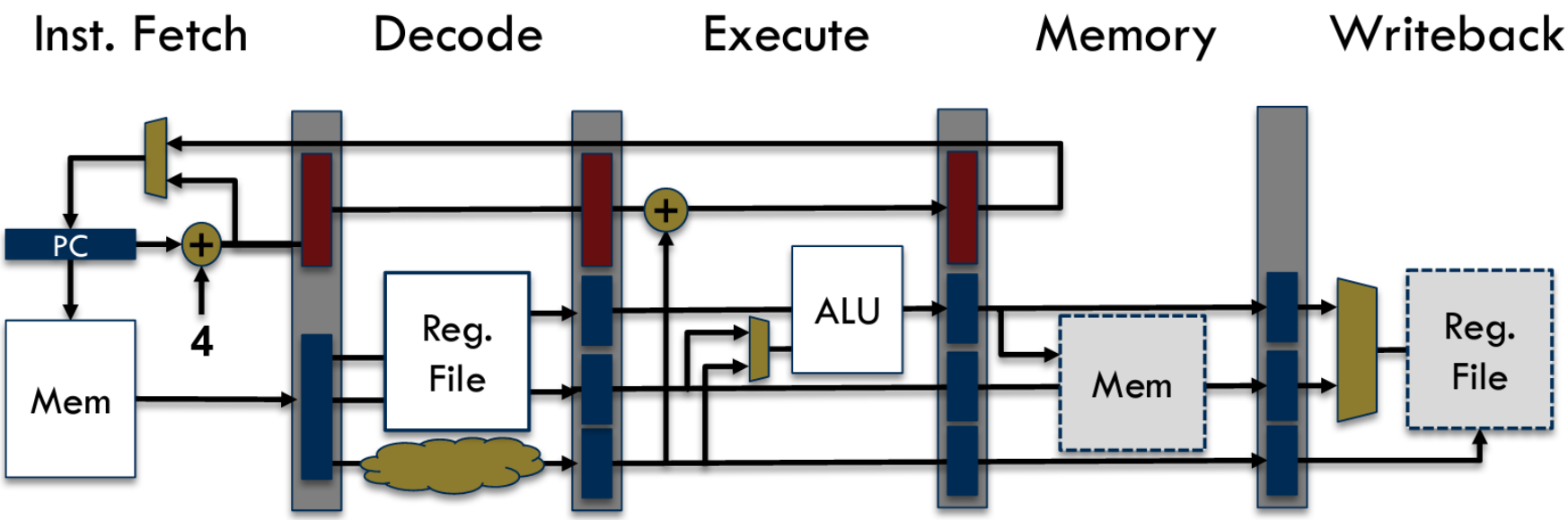
https://en.wikipedia.org/wiki/Memory_management_unit

RISC without pipelining



RISC
Simple Instructions
No μ-programming
RISC PM = N x CISC PM

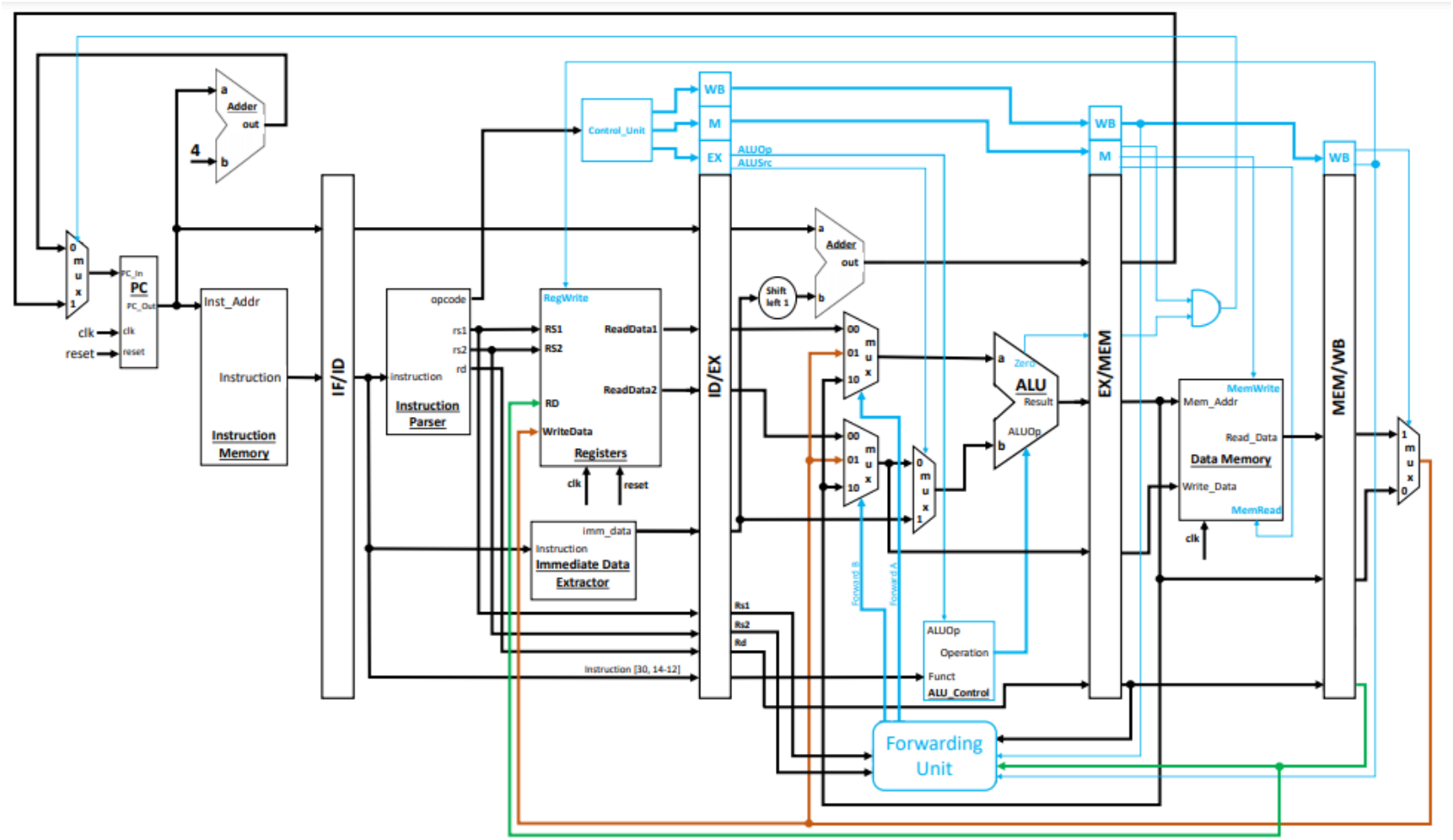
5 Stage Pipelining



1. Instruction Fetch
2. Decode Instruction
3. Execute Instruction
4. Memory Access
5. Writeback to Registers

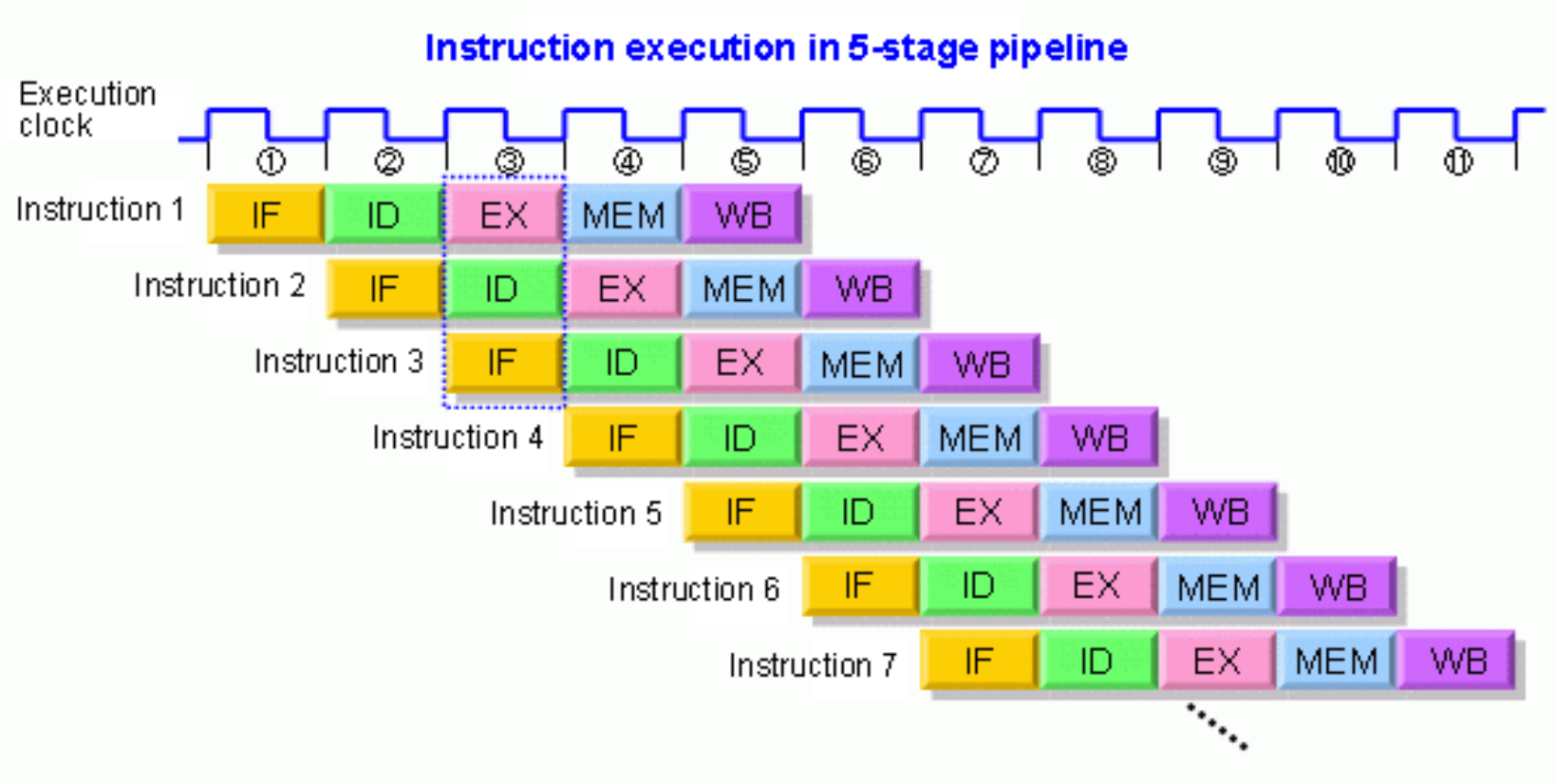
https://users.cs.utah.edu/~bojnordi/classes/6810/s18/slides/05-pipelining.pdf

5 Stage Pipelining (Risc-V)



https://github.com/AkeelMedina22/RISC-V-Pipelined-Processor

5 Stage Pipelining



http://www.cs.iit.edu/~cs561/cs350/CPU/5stage.html

Pipeline Hazards

Structural hazards: Multiple instructions compete for the same resource

Data hazards: A dependent instruction cannot proceed because it needs a value that hasn't been produced

Control hazards: The next instruction cannot be fetched because the outcome of an earlier branch is unknown

Data Hazards

Consider the following set of instructions in a 5-stage pipeline.
Operands are read in ID.
MEM is memory Write for result; RW is Register Write for result

R3 is accessed in READ mode; expect the result of ADD to be available in **R3**

ADD **R3**, R6, R5	- Results to be written in R3
SUB R4, **R3**, R5	- R3 has one of the operand
OR R6, **R3**, R7	- R3 has one of the operand
AND R8, **R3**, R7	- R3 has one of the operand
XOR R12, **R3**, R10	- R3 has one of the operand

But result of ADD written in R3 at t5

	t1	t2	t3	t4	t5	t6	t7	t8	t9
ADD R3, R6, R5	IF	ID	IE	MEM	RW R3	--	--	--	--
SUB R4, R3, R5	--	IF	ID R3	IE	MEM	RW	--	--	--
OR R6, R3, R7	--	--	IF	ID R3	IE	MEM	RW	--	--
AND R8, R3, R9	--	--	--	IF	ID R3	IE	MEM	RW	--
XOR R10, R3, R11	--	--	--	--	IF	ID R3	IE	MEM	RW

Note when each instruction is accessing R3

https://witscad.com/course/computer-architecture/chapter/pipeline-hazards

Introduction

Questions?

Contact information

Andreas Axelsson

Email: andreas.axelsson@ju.se

Mobile: 0709-467760