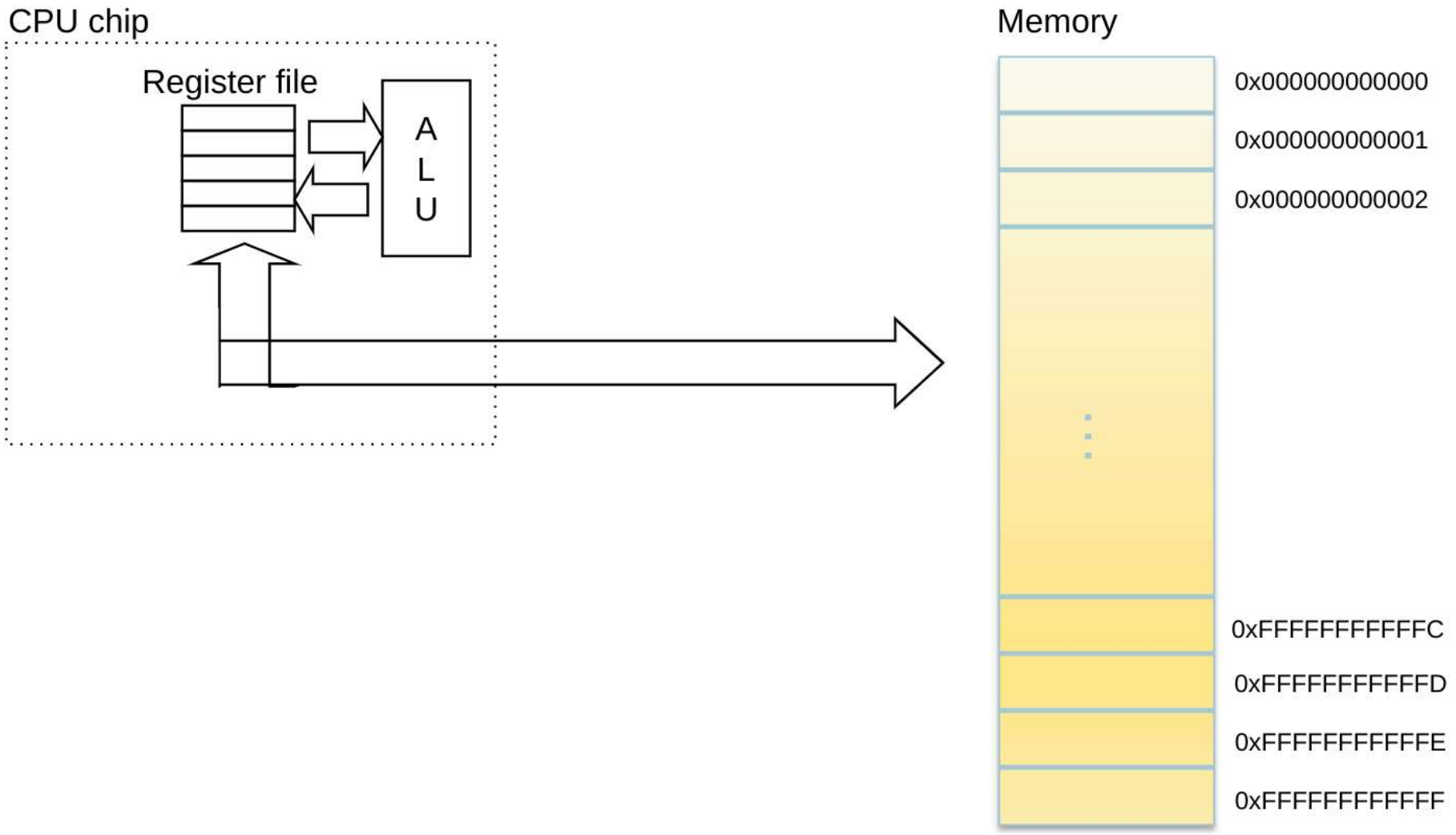
Operating Systems & C

Lecture 4: Memory

Willard Rafnsson
IT University of Copenhagen

ITU CPH

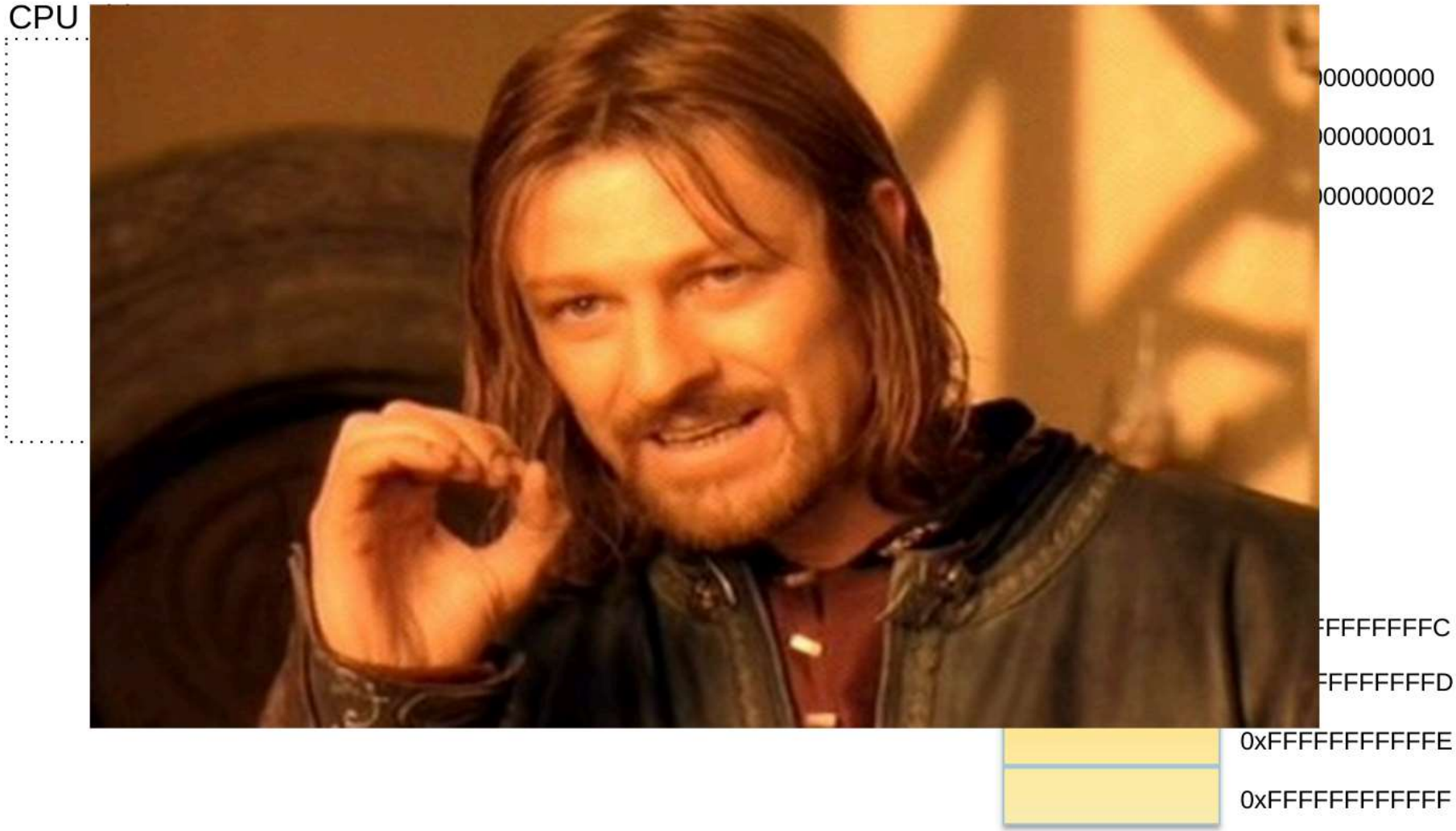
Our view of computers, so far



UNIVERSITY OF COPENHAGEN

ITU CPH

Our view of computers, so far

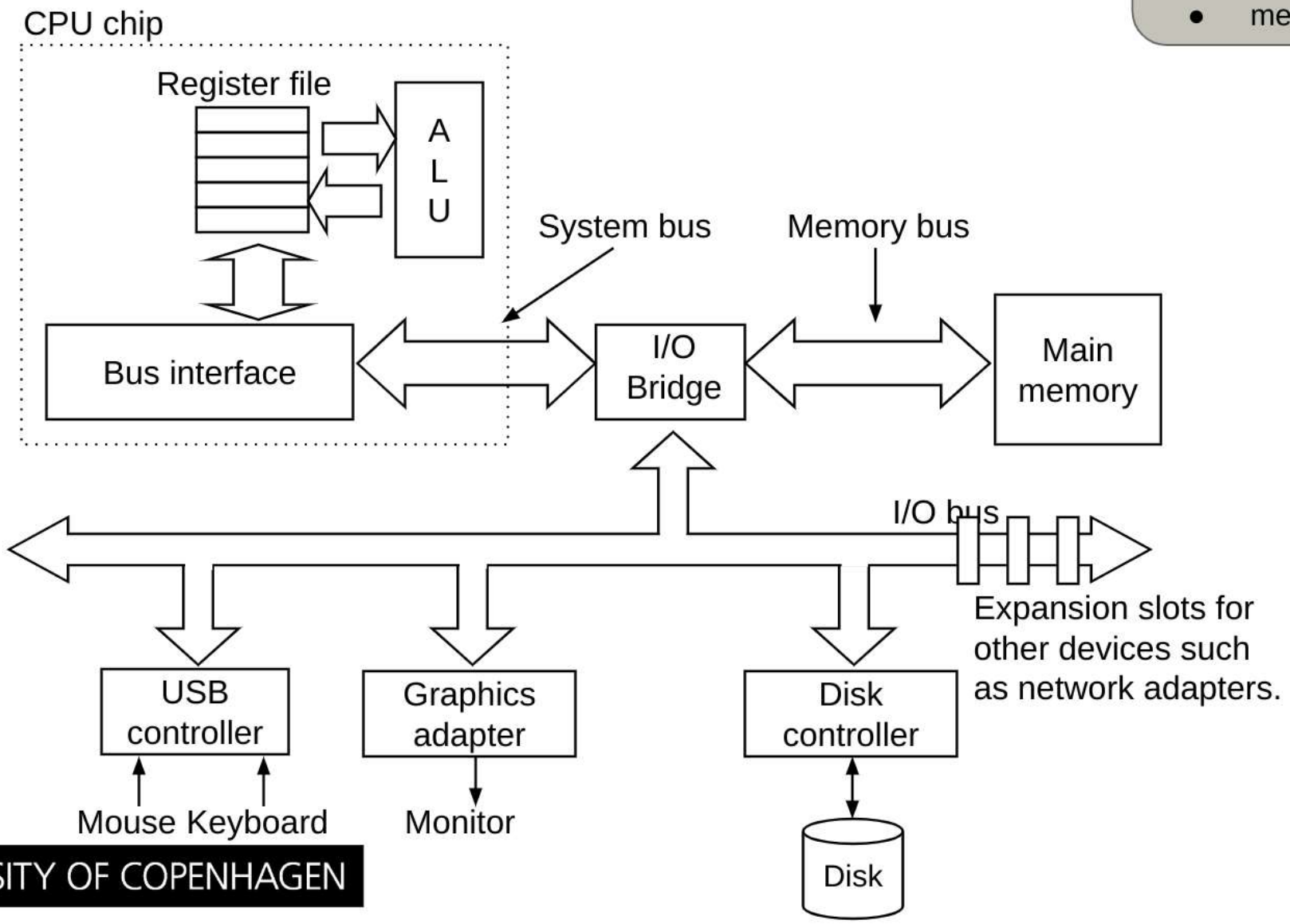


UNIVERSITY OF COPENHAGEN

ITU CPH

Computers are more like this

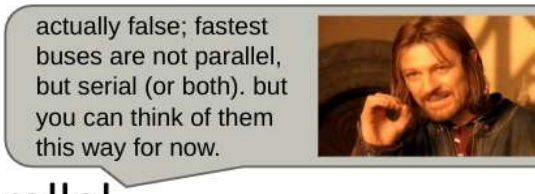
then, how does I/O work?
how is data transferred from
- secondary storage to memory, and
- memory to registers?



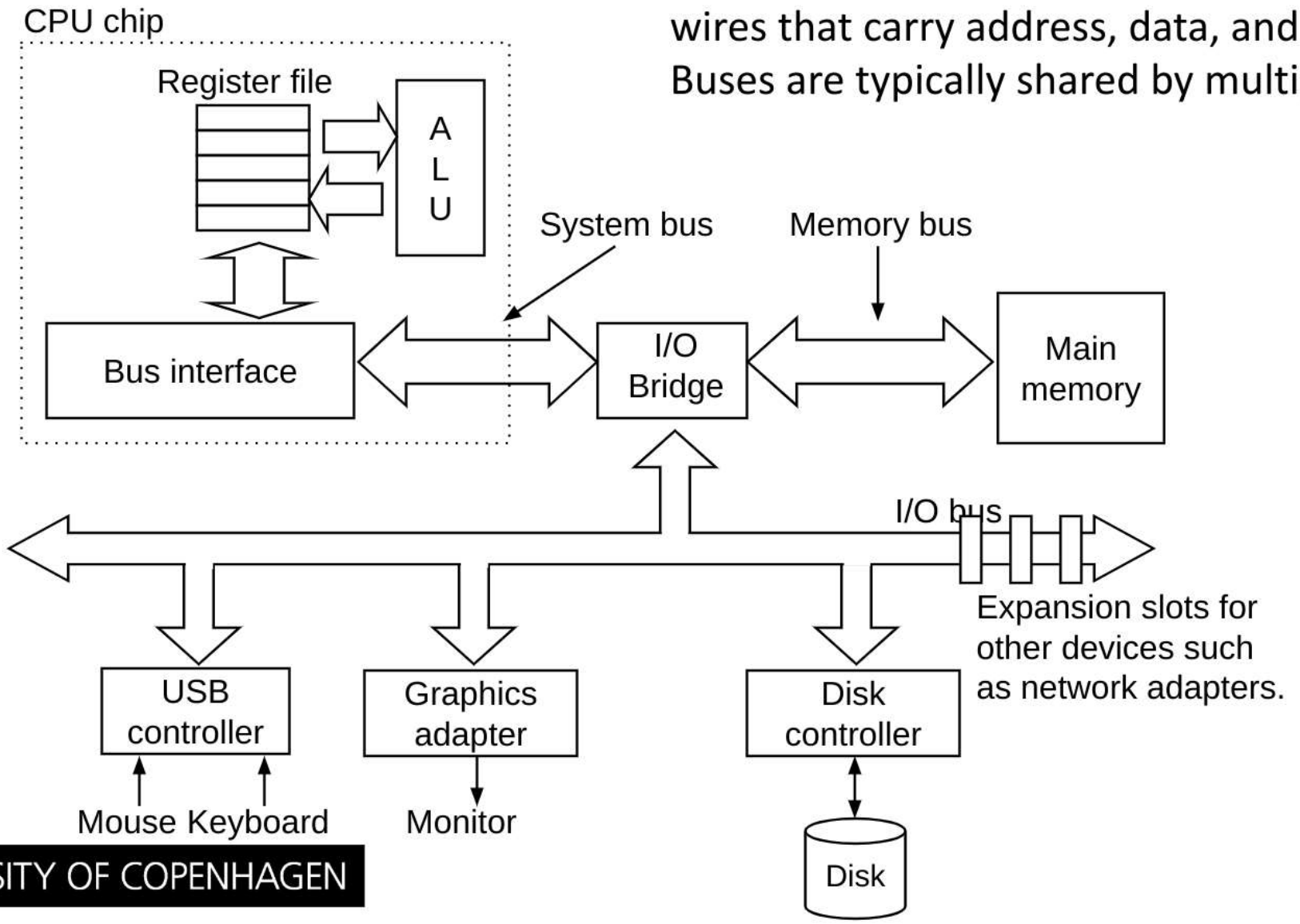
UNIVERSITY OF COPENHAGEN

ITU CPH

I/O Bus



A bus is a collection of parallel wires that carry address, data, and control signals
Buses are typically shared by multiple devices.

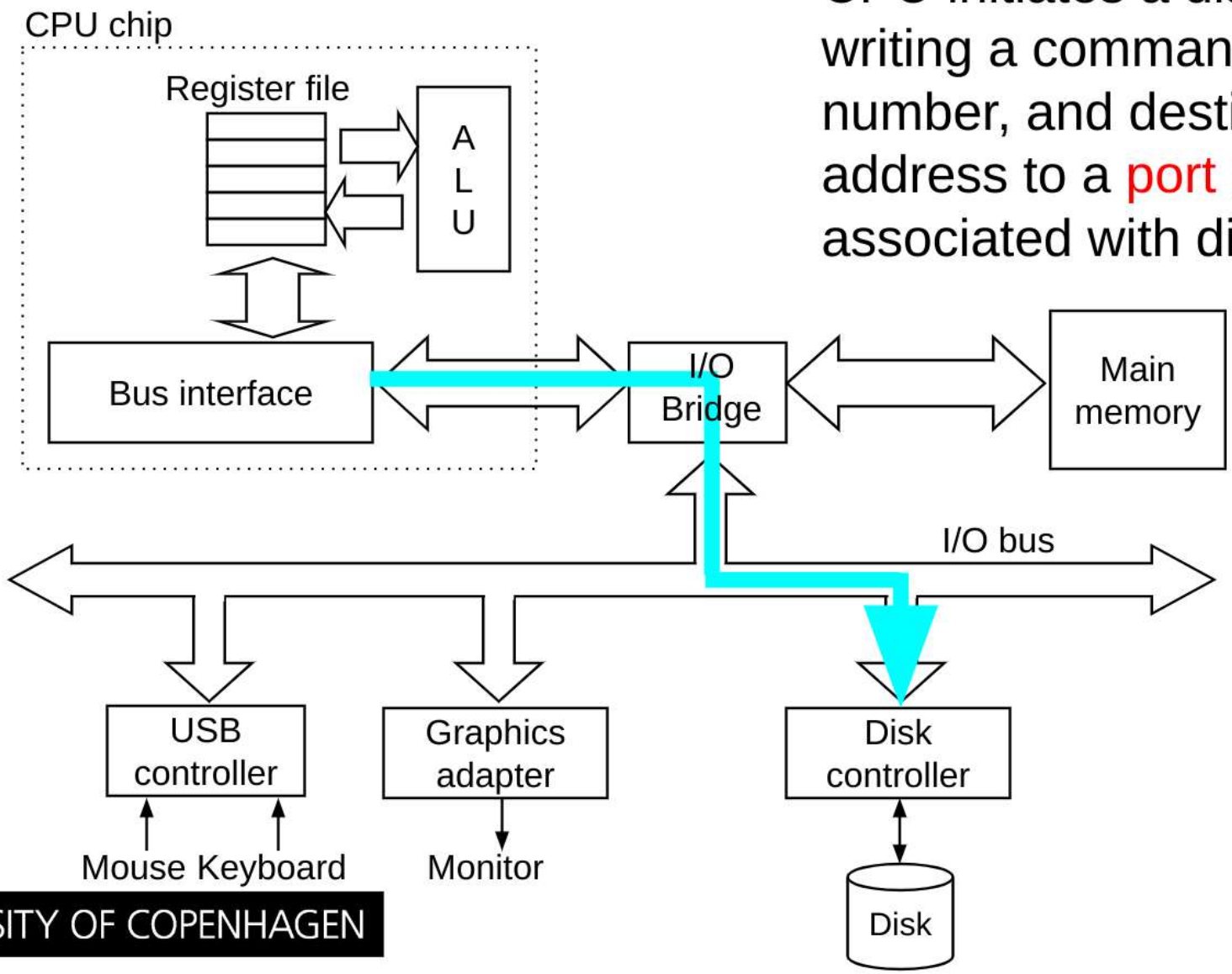


UNIVERSITY OF COPENHAGEN

ITU CPH

Reading a Disk Sector (1)

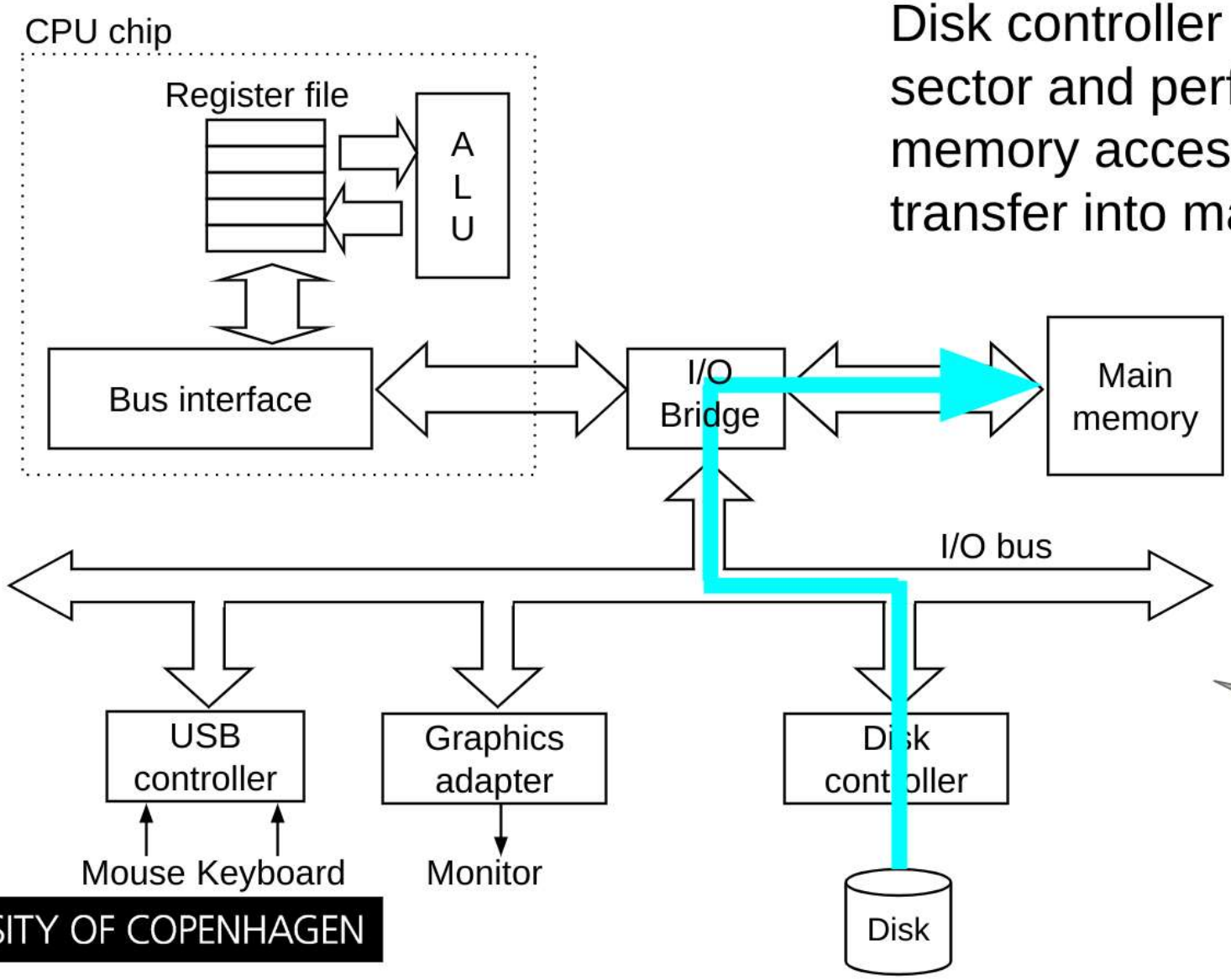
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.



UNIVERSITY OF COPENHAGEN

ITU CPH

Reading a Disk Sector (2)



Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

note: CPU is not involved in this.

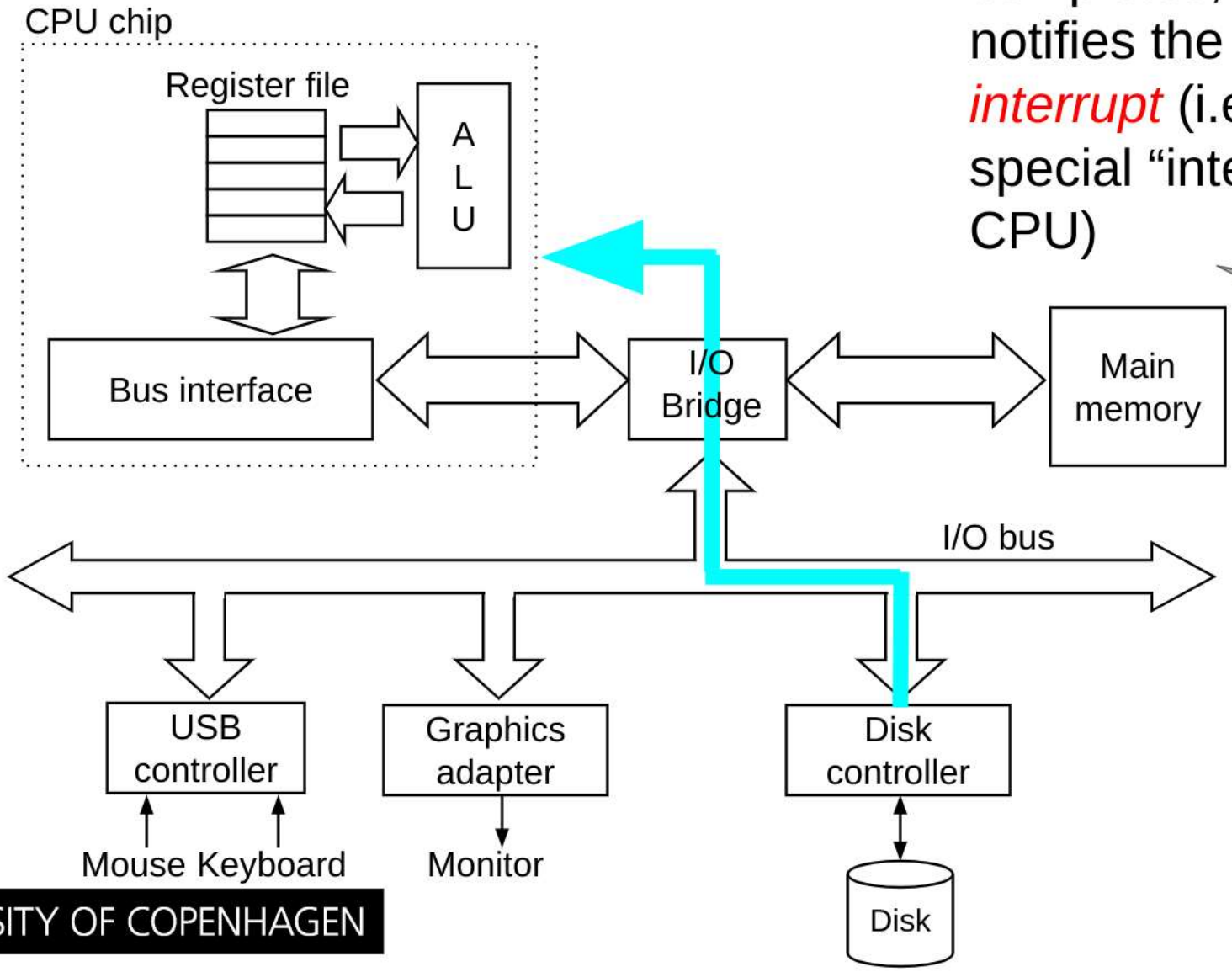
UNIVERSITY OF COPENHAGEN

ITU CPH

Reading a Disk Sector (3)

When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)

we'll learn about interrupts in future lecture

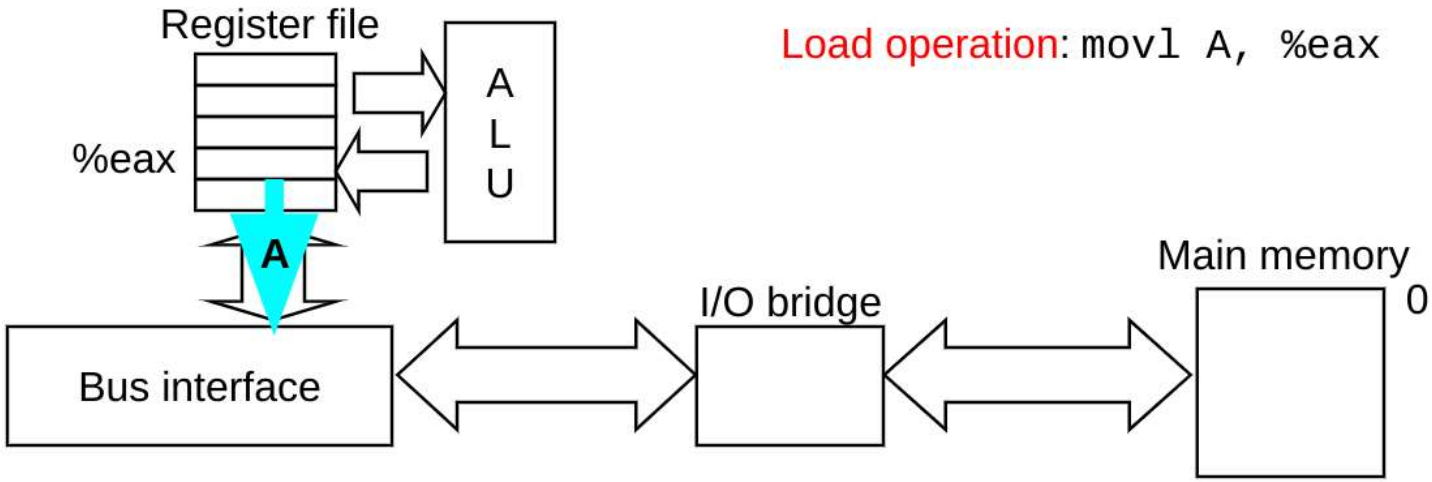


UNIVERSITY OF COPENHAGEN

ITU CPH

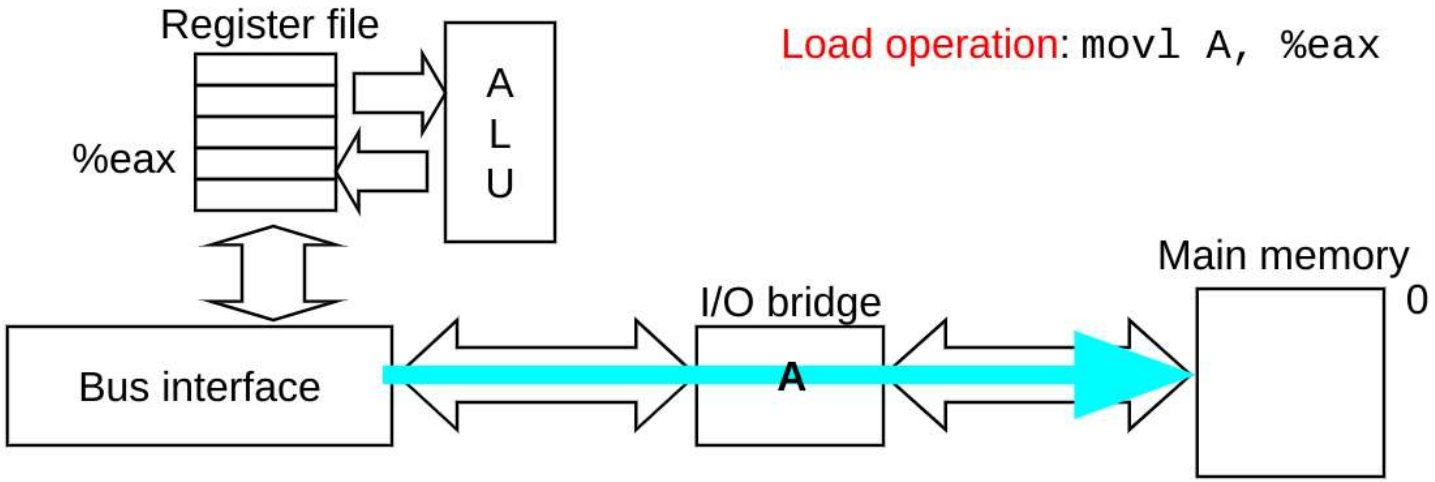
Memory Read Transaction (1)

CPU places address A on the memory bus.



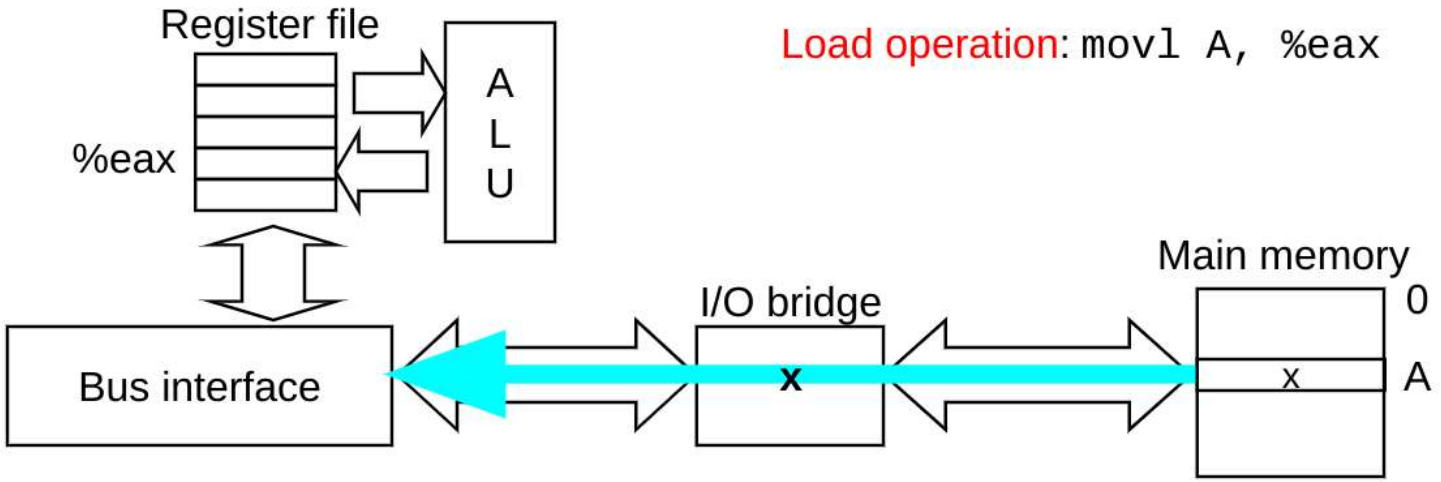
Memory Read Transaction (2)

Main memory reads A from the memory bus.



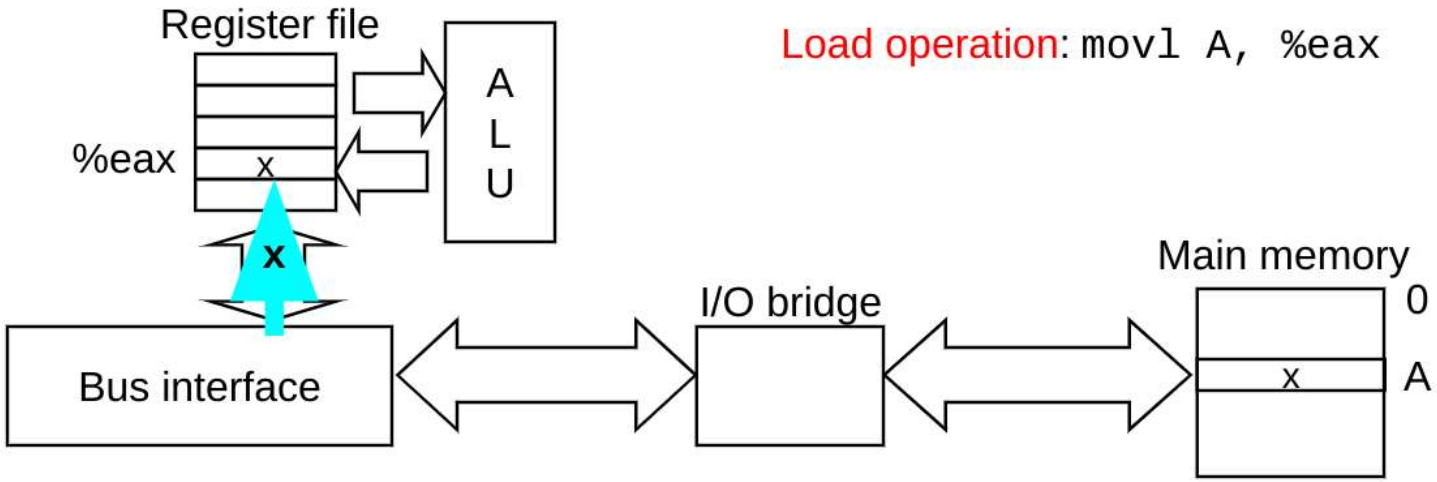
Memory Read Transaction (3)

Main memory retrieves word at address A (that's x), and places it on the bus.



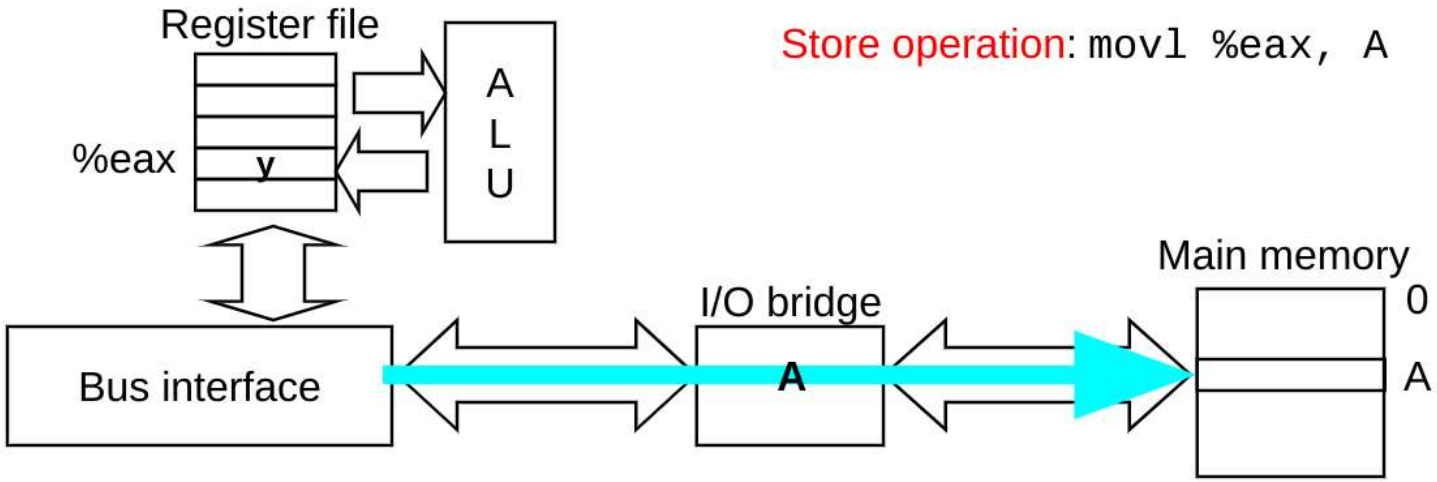
Memory Read Transaction (4)

CPU reads word from the bus (that's x), and copies it into %eax.



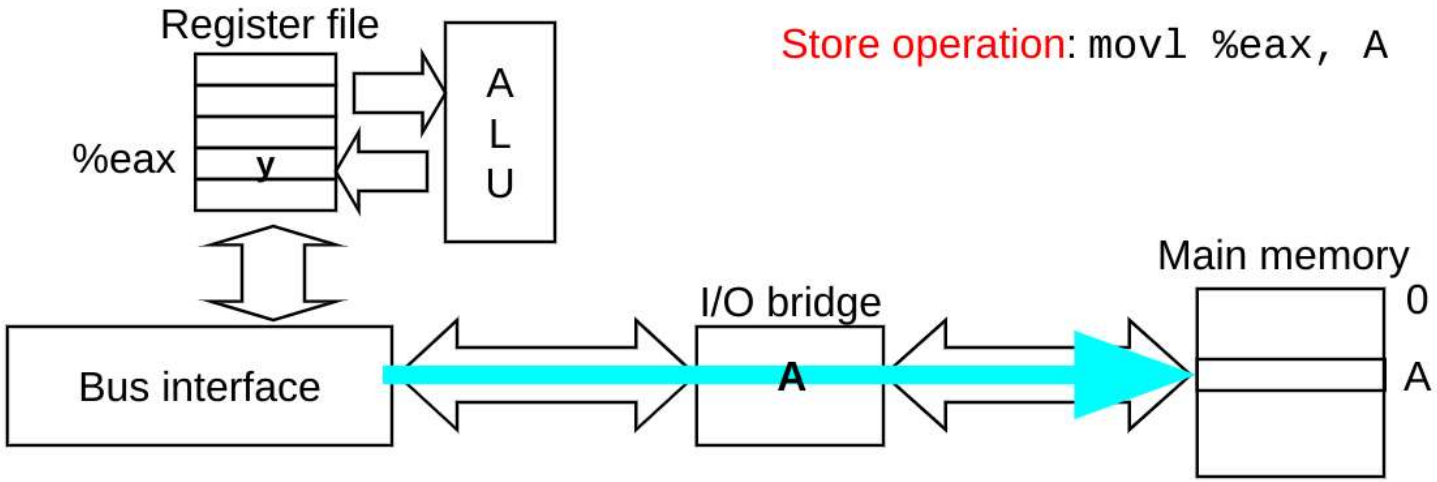
Memory Write Transaction (2)

Main memory reads A from the memory bus,
and waits for the corresponding data to arrive.



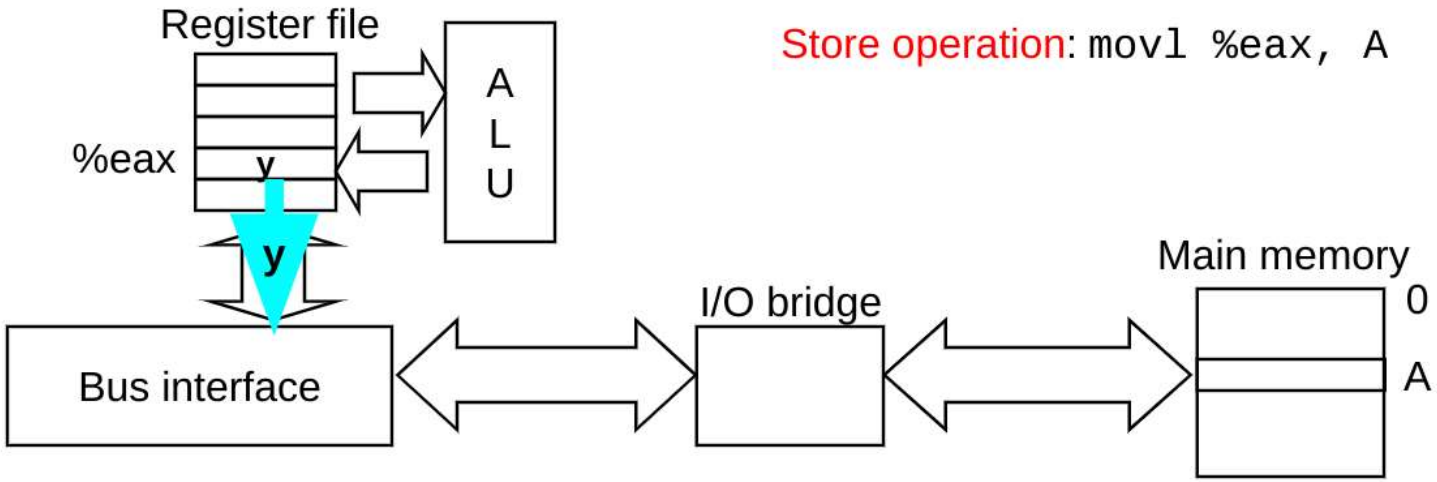
Memory Write Transaction (2)

Main memory reads A from the memory bus,
and waits for the corresponding data to arrive.



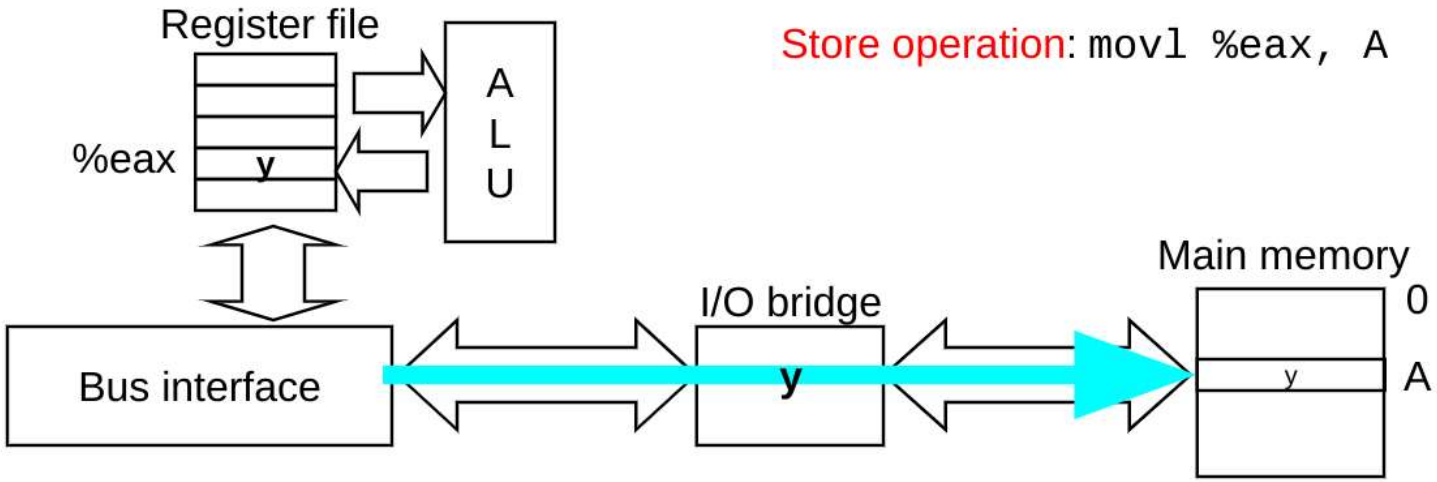
Memory Write Transaction (3)

CPU places contents of %eax on the memory bus (that's word y).



Memory Write Transaction (4)

Main memory reads y from the memory bus, and stores it at address A.

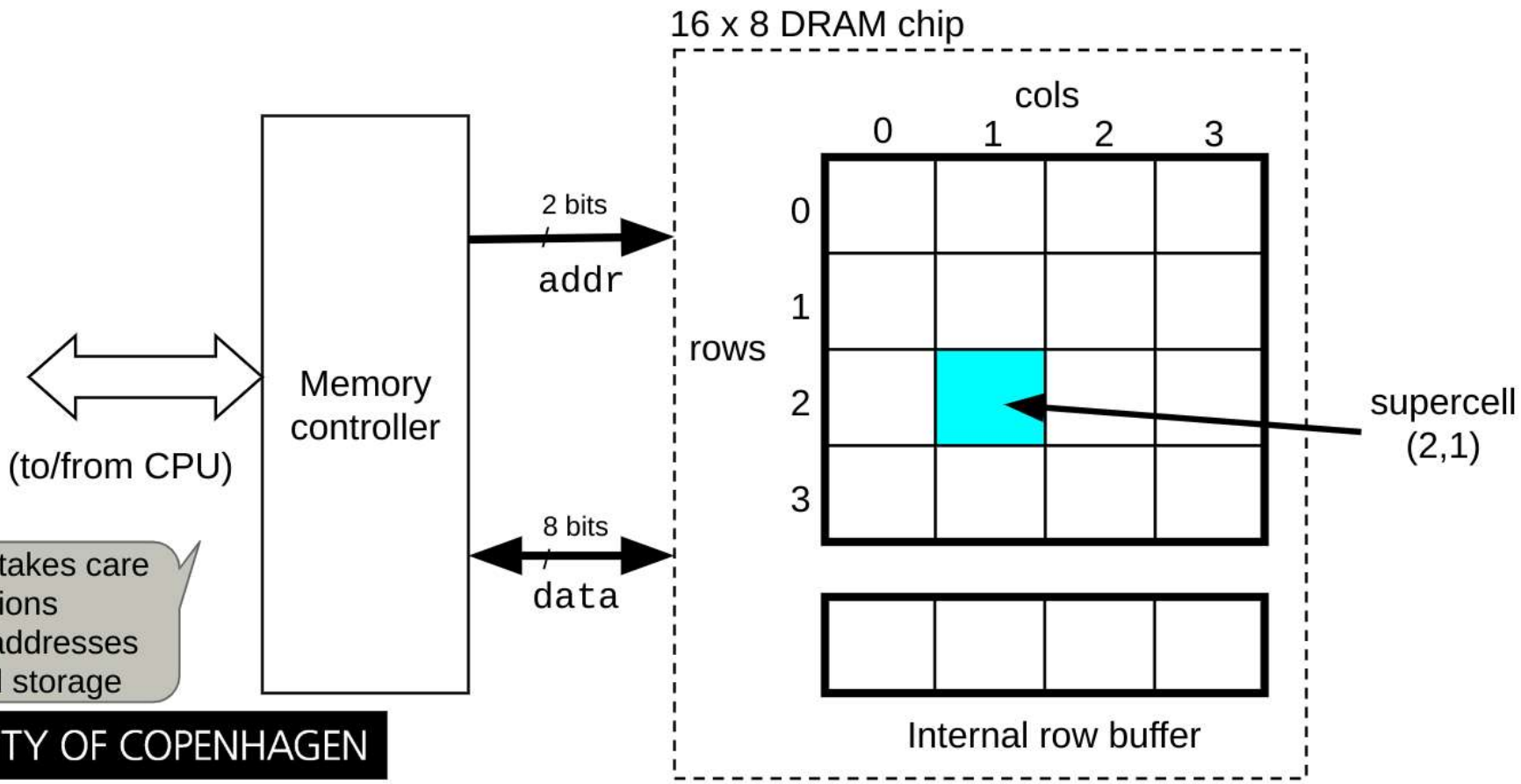


(Aside: Conventional DRAM Organization)

the way RAM is organized is as an array of "supercells

d * w DRAM:

dw total bits organized as d supercells of size w bits



controller takes care of translations between addresses & physical storage

UNIVERSITY OF COPENHAGEN

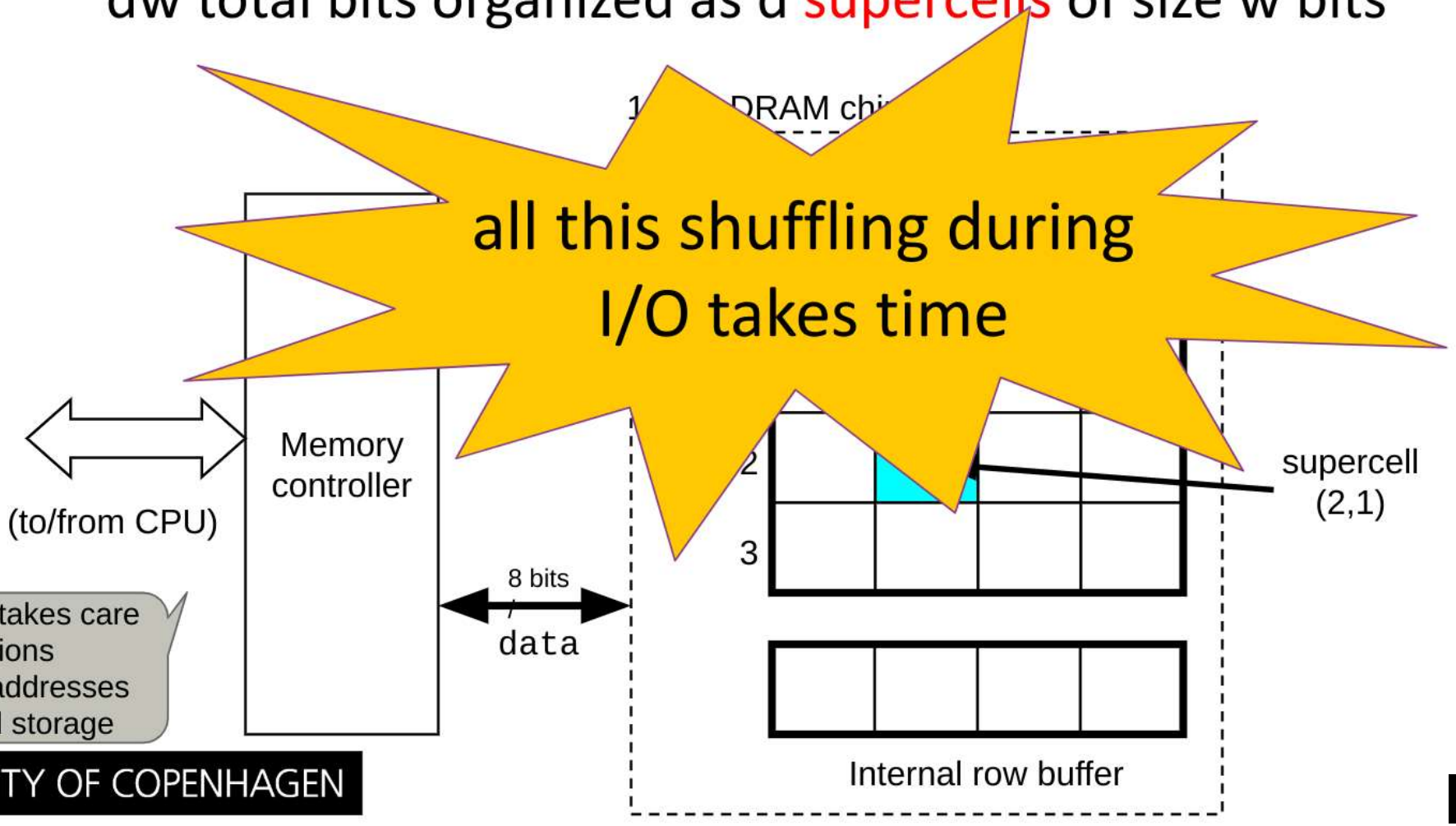
ITU CPH

(Aside: Conventional DRAM Organization)

the way RAM is organized is as an array of "supercells

d * w DRAM:

dw total bits organized as d supercells of size w bits



controller takes care of translations between addresses & physical storage

UNIVERSITY OF COPENHAGEN

ITU CPH

Today: Memory

- Locality
- Caching
- Virtual Memory
- Cache Coherence
- Prefetching

ITU CPH

Locality

ITU CPH

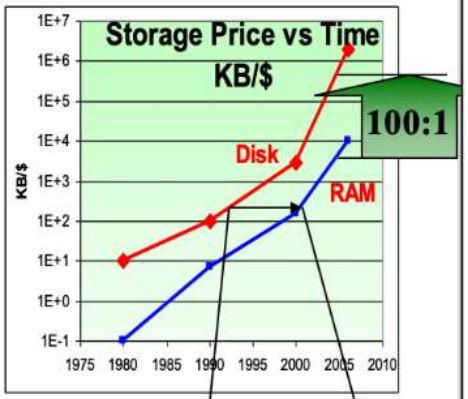
Jim Gray, 2006

http://jimgray.azurewebsites.net/jimgraytalks.htm

layout of data in memory

RAM Locality is King

- The cpu mostly waits for RAM
- Flash / Disk are 100,000 …1,000,000 clocks away from cpu
- RAM is ~100 clocks away unless you have locality (cache).
- If you want 1CPI (clock per instruction) you have to have the data in cache (program cache is "easy")
- This requires cache conscious data-structures and algorithms sequential (or predictable) access patterns
- Main Memory DB is going to be common.



today: caching.

from this talk, which has **motivated/driven a lot of systems research** since.
(side-note: Jim Gray vanished w/o a trace in 2007)

Tape is Dead
Disk is Tape
Flash is Disk
RAM Locality is King

Jim Gray
Microsoft
December 2006

UNIVERSITY OF COPENHAGEN

01. ITU CPH 7

Locality

(Principle: a rule-of-thumb ("do this; not that."))

Principle of Locality: use data & instructions with addresses near or equal to those used recently.

why is this a good principle: a read from memory (& disk) takes time.(despite buses being wide (⇒ a lot of data can be read at once), request-reply round trip takes time). **if** memory (& disk) access is *predictable*, **then** we can have data at hand before it is needed (i.e. *prefetch*). **programs that follow this principle have predictable memory (& disk) access**.

Locality types:

- **Temporal** locality:
  recently referenced items are
  likely to be referenced again in the near future
- **Spatial** locality:
  items with nearby addresses
  tend to be referenced close together in time

Locality Example

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

Data references

- Reference array elements in succession (stride-1 reference pattern).
- Reference variable `sum` each iteration.

Instruction references

- Reference instructions in sequence.
- Cycle through loop repeatedly.

Locality Example

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

Data references

- Reference array elements in succession (stride-1 reference pattern). — Spatial locality
- Reference variable `sum` each iteration. — Temporal locality

Instruction references

- Reference instructions in sequence. — Spatial locality
- Cycle through loop repeatedly. — Temporal locality

Qualitative Estimates of Locality

Claim: Being able to look at code and get a qualitative sense of its locality is **a key skill for a professional programmer**.

Question: Does this function have good locality wrt. array a?

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

UNIVERSITY OF COPENHAGEN

Q: which is faster?

ITU CPH

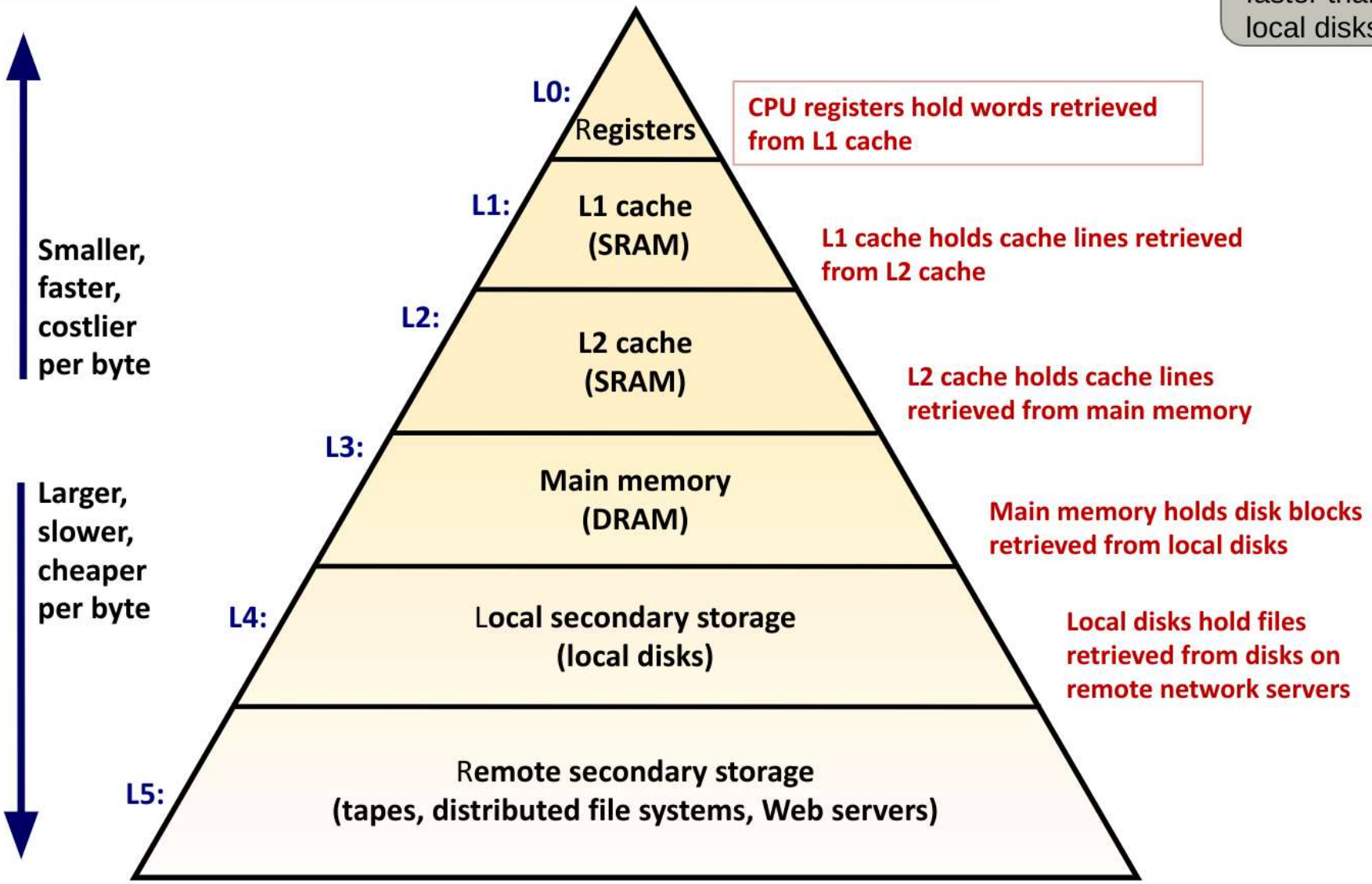
Memory Hierarchies

- Some fundamental and enduring properties of HW & SW:
  - Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
  - The gap between CPU and main memory *speed* is widening.
  - Well-written programs tend to exhibit good locality.

- These fundamental properties complement each other beautifully. They suggest an approach for organizing memory and storage systems known as a memory hierarchy.

An Example Memory Hierarchy

this is actually a myth; local network faster than some local disks.

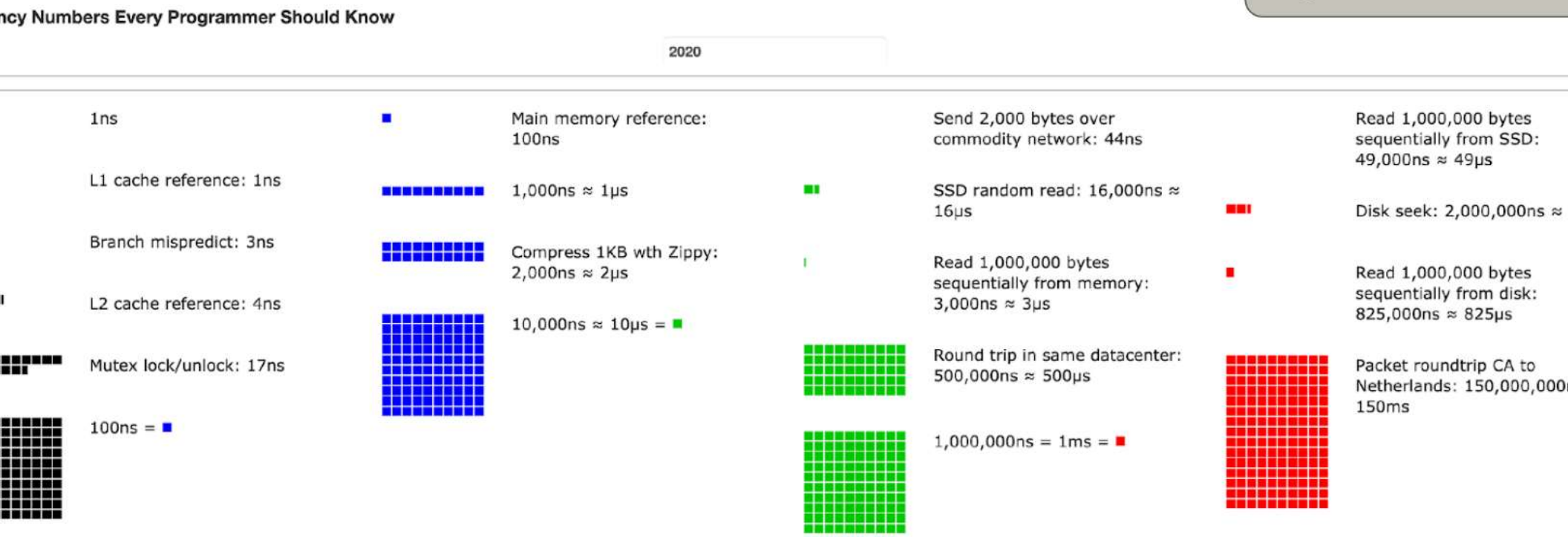


UNIVERSITY OF COPENHAGEN

ITU CPH

Latency Numbers Every Programmer Should Know

penalty: from 1 to 100.
huge difference.



https://colin-scott.github.io/personal_website/research/interactive_latency.html

UNIVERSITY OF COPENHAGEN

01. ITU CPH 42

Caching

ITU CPH

Caches

car mechanic analogy

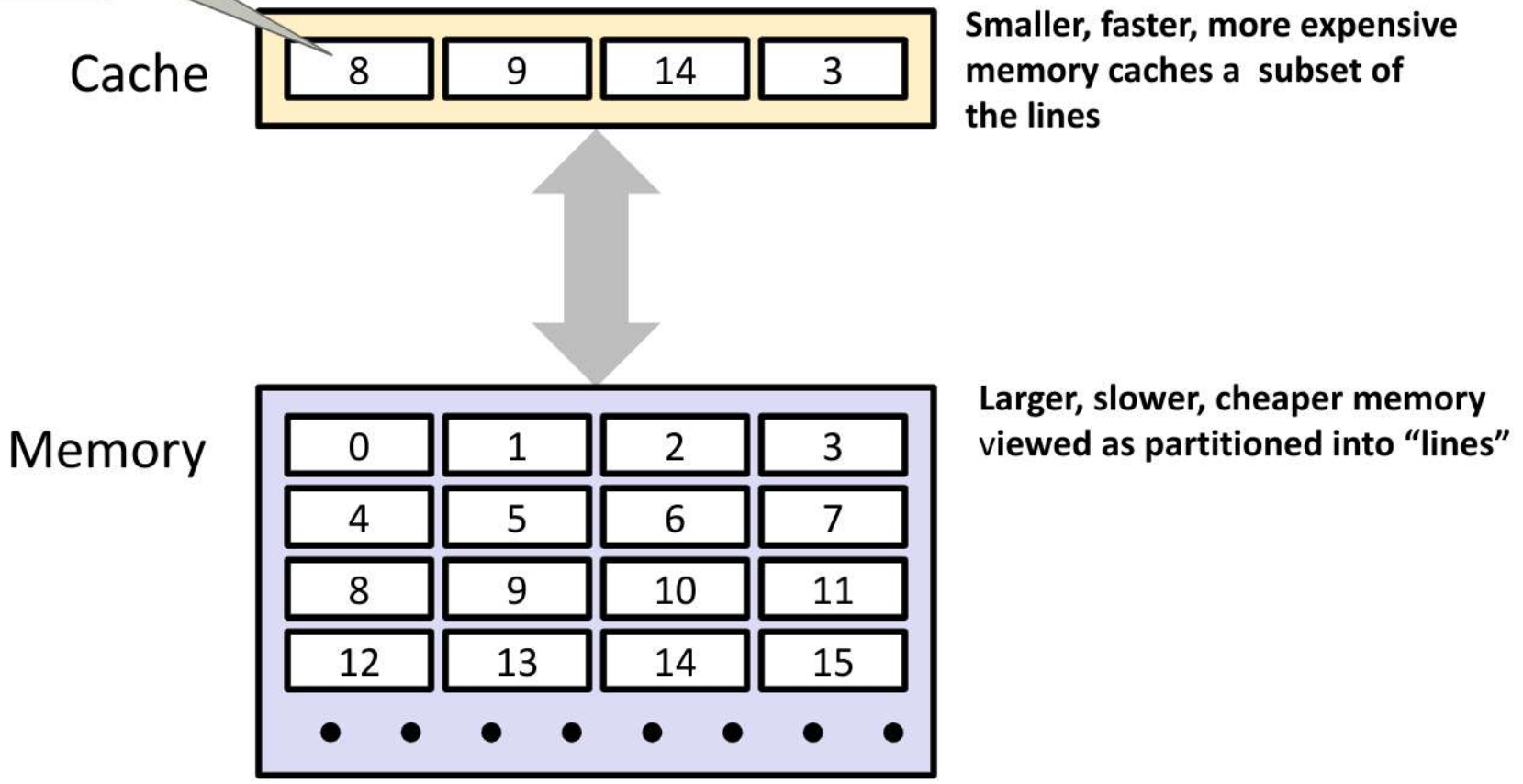
- *Cache:* A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.
- **Fundamental idea of a memory hierarchy:**
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- Why do memory hierarchies work? Because **locality**.
  - Programs tend to access the data at level k more often than they access the data at level k+1.
  - Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.
- *Big Idea:* Memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but serves data to programs at the rate of the fast storage near the top.

General Cache Concepts

unit of transfer: 1 line

memory is split into blocks, aka. lines, of size 64 bytes.

umber represents equested memory ocation



UNIVERSITY OF COPENHAGEN

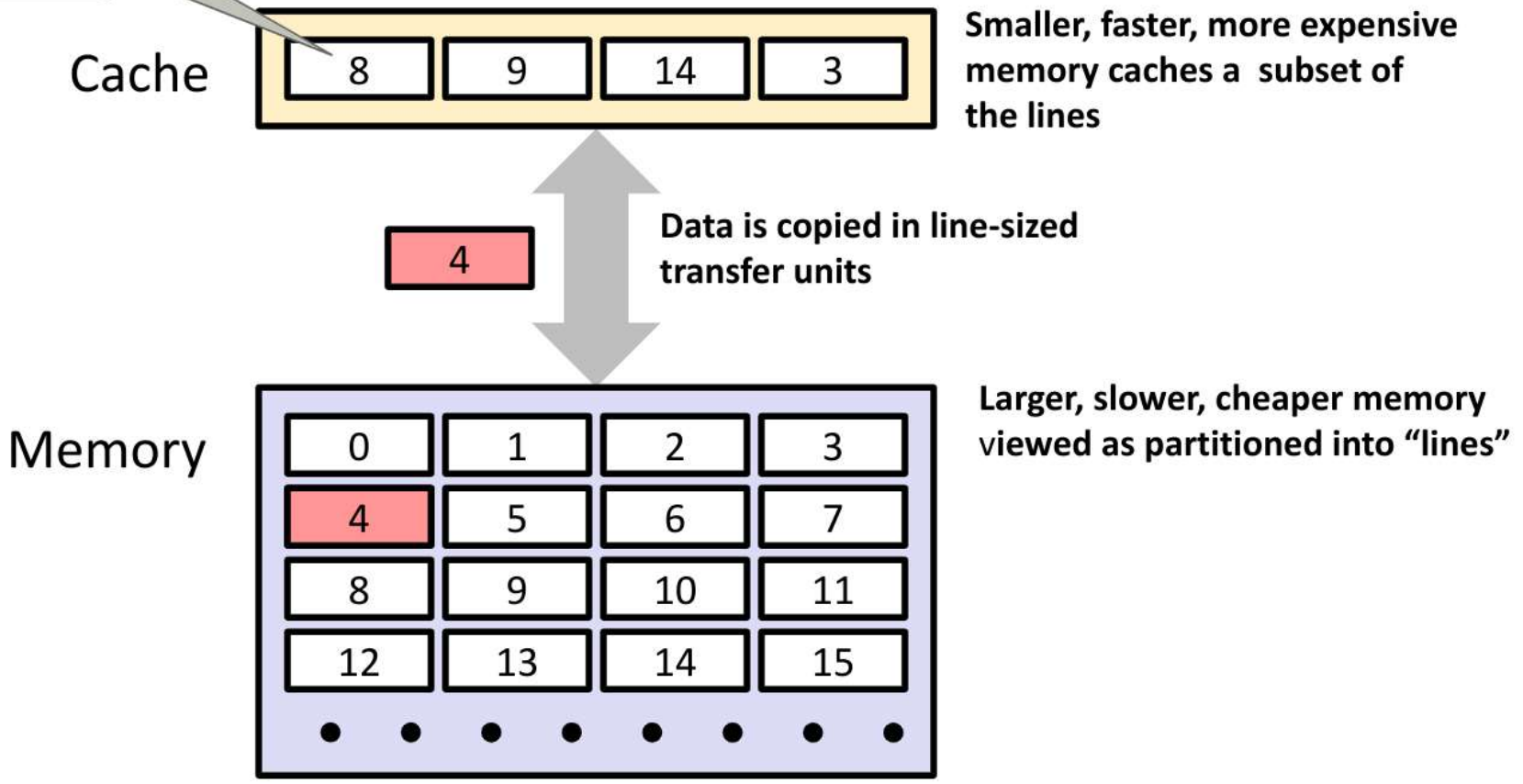
ITU CPH

General Cache Concepts

unit of transfer: 1 line

memory is split into blocks, aka. lines, of size 64 bytes.

umber represents equested memory ocation



UNIVERSITY OF COPENHAGEN

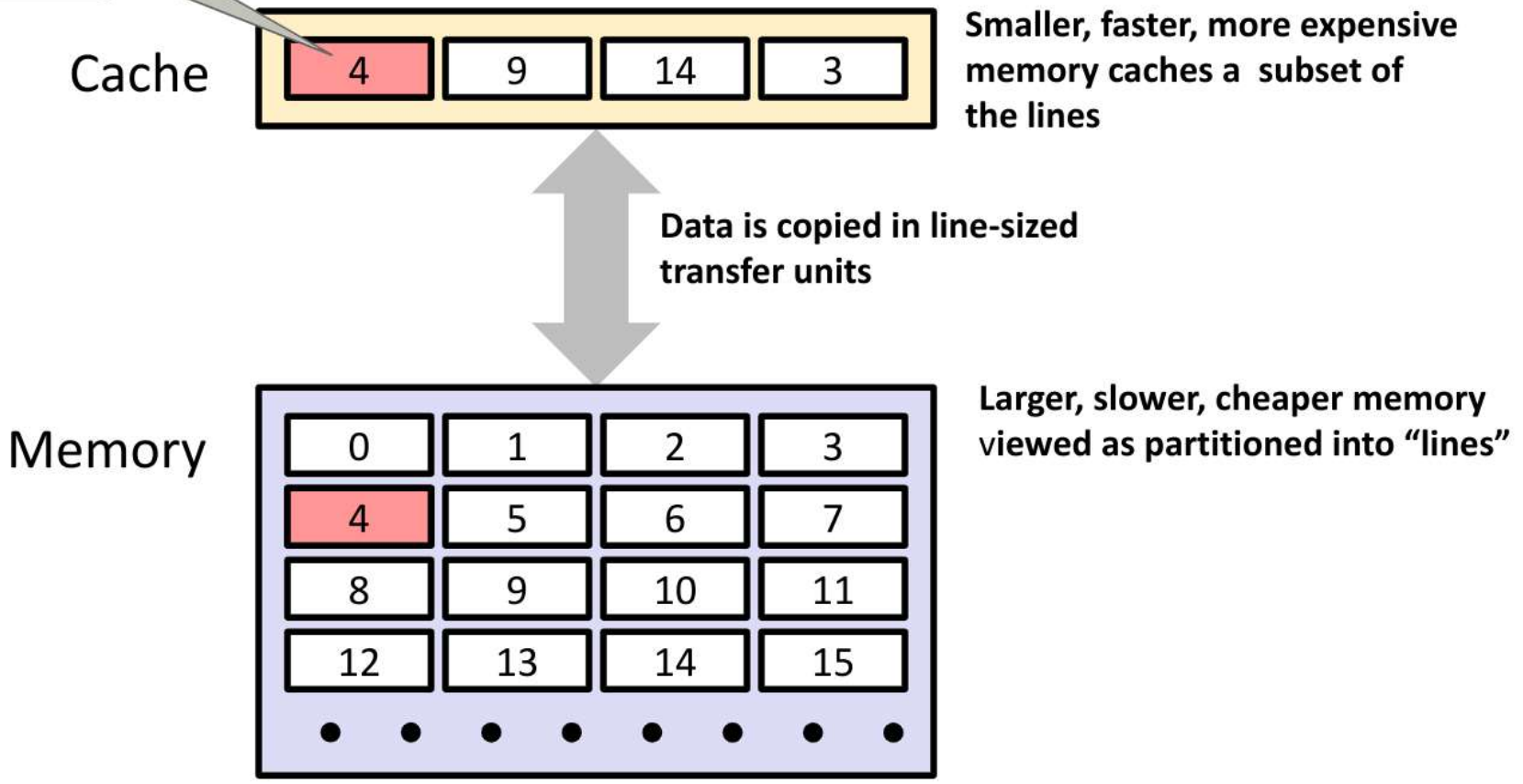
ITU CPH

General Cache Concepts

unit of transfer: 1 line

memory is split into blocks, aka. lines, of size 64 bytes.

umber represents equested memory ocation



UNIVERSITY OF COPENHAGEN

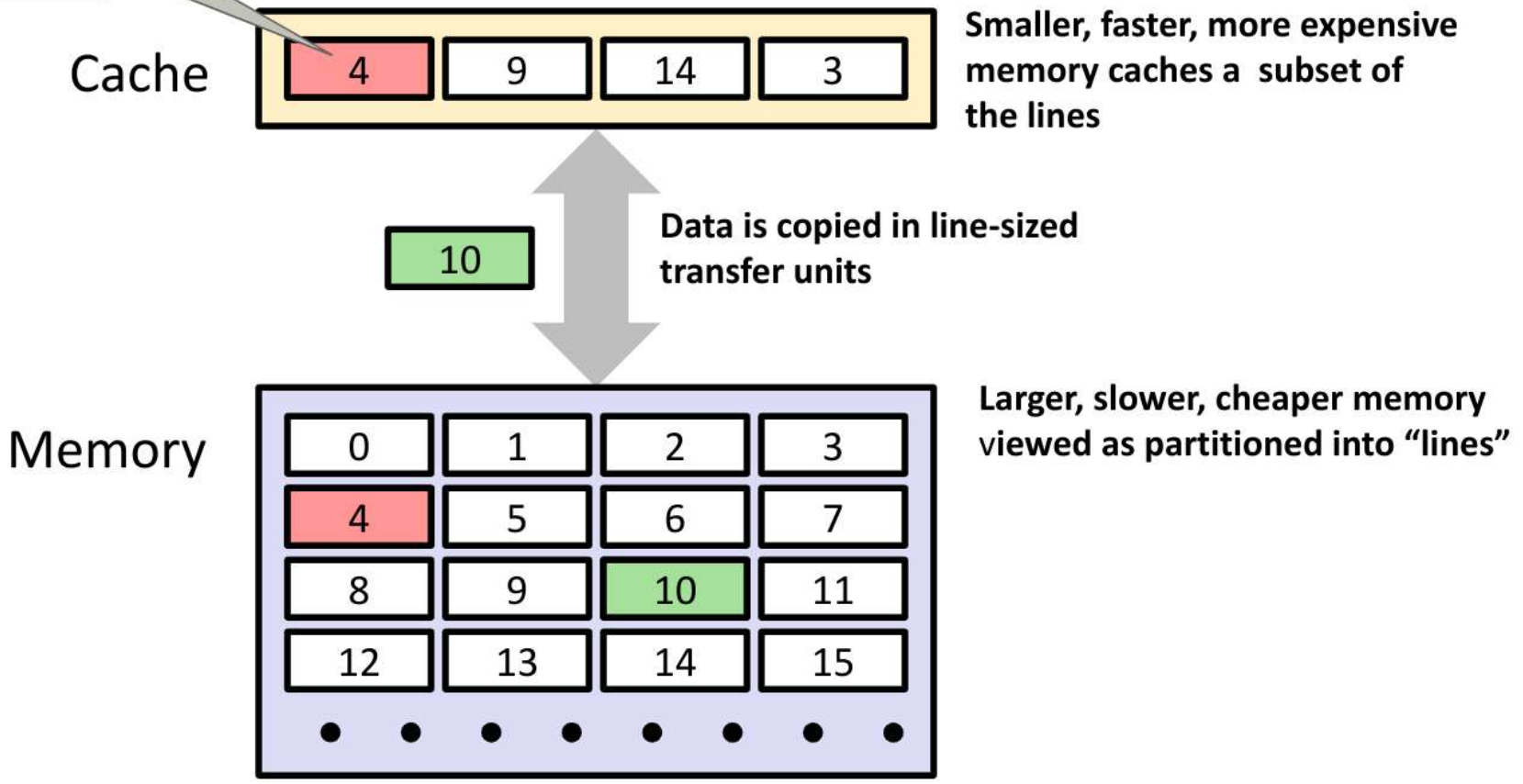
ITU CPH

General Cache Concepts

unit of transfer: 1 line

memory is split into blocks, aka. lines, of size 64 bytes.

umber represents equested memory ocation



UNIVERSITY OF COPENHAGEN

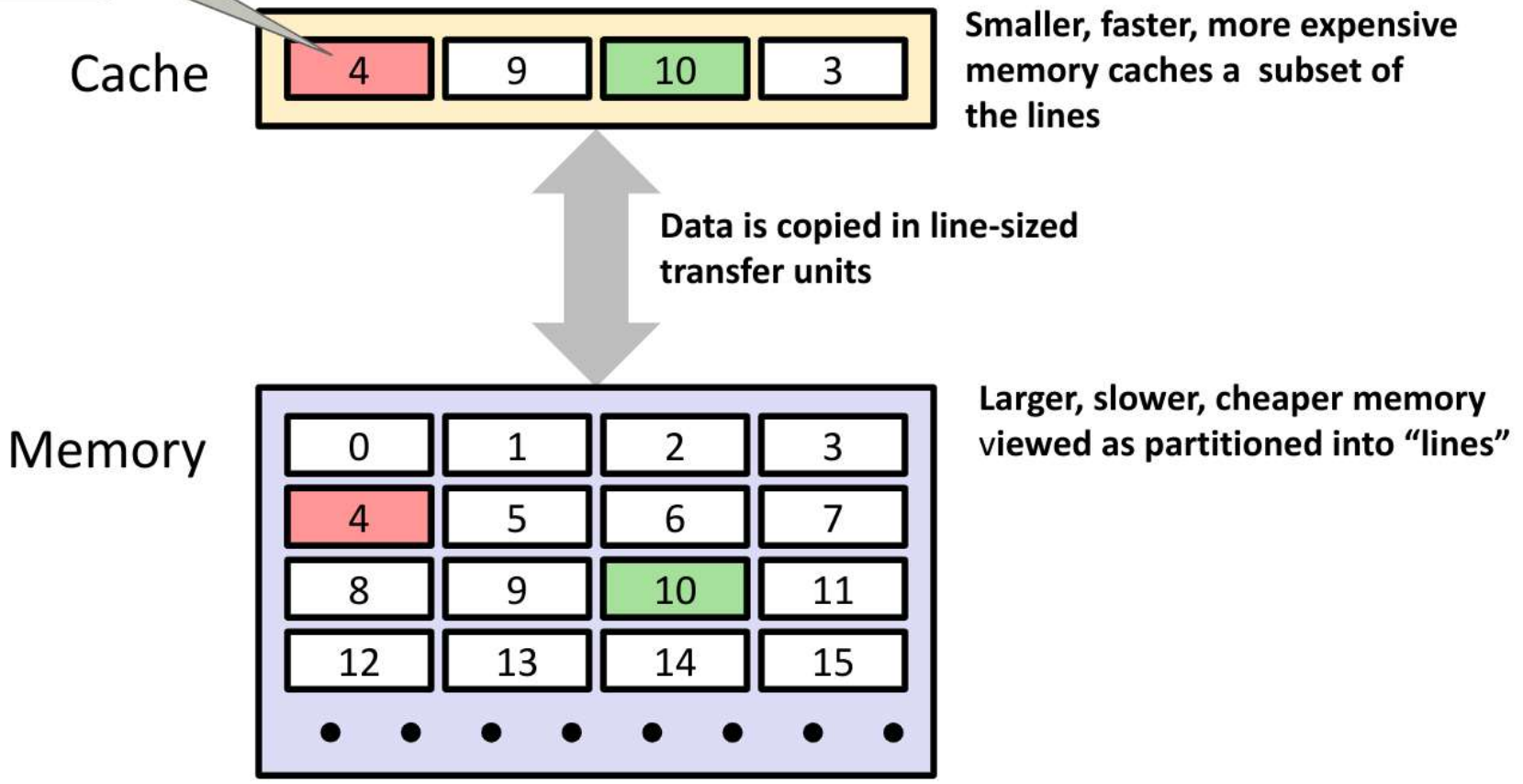
ITU CPH

General Cache Concepts

unit of transfer: 1 line

memory is split into blocks, aka. lines, of size 64 bytes.

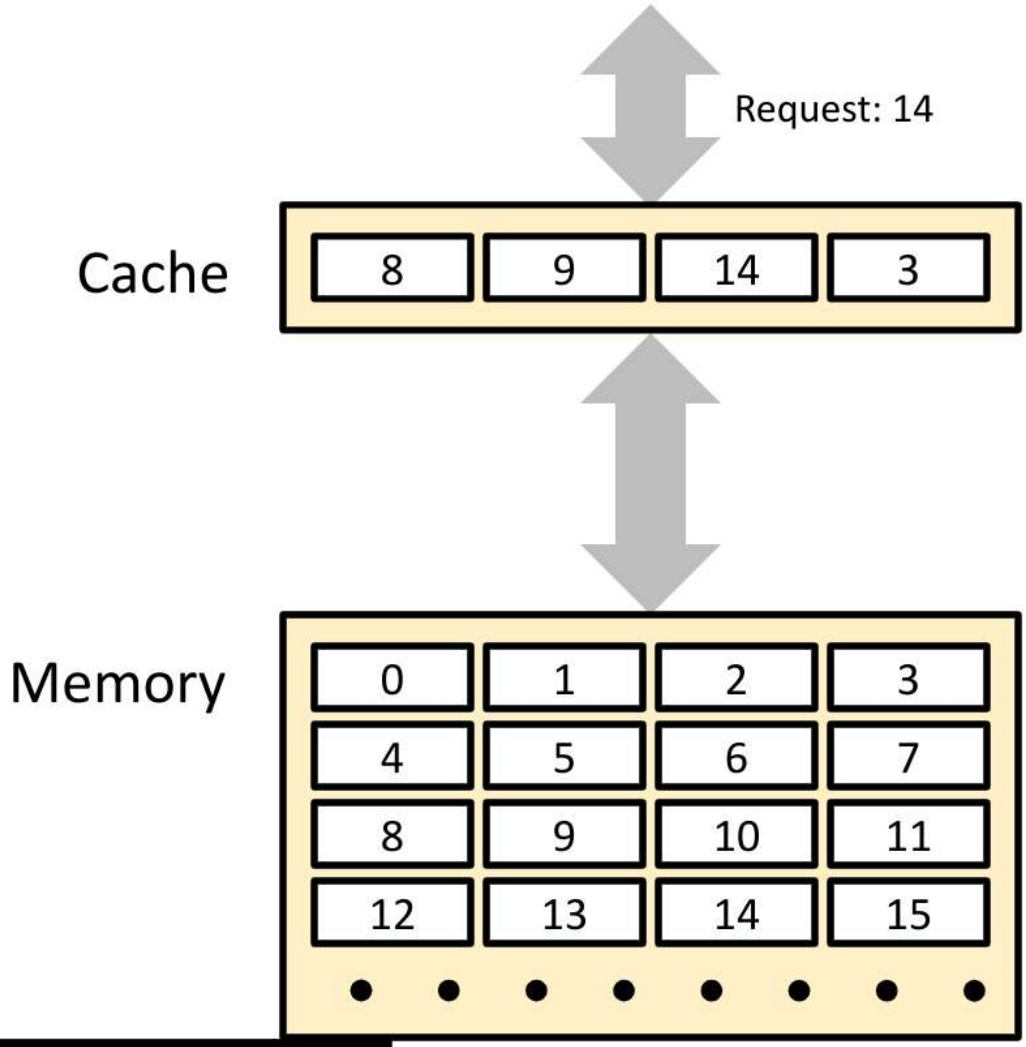
umber represents equested memory ocation



UNIVERSITY OF COPENHAGEN

ITU CPH

General Cache Concepts: Hit

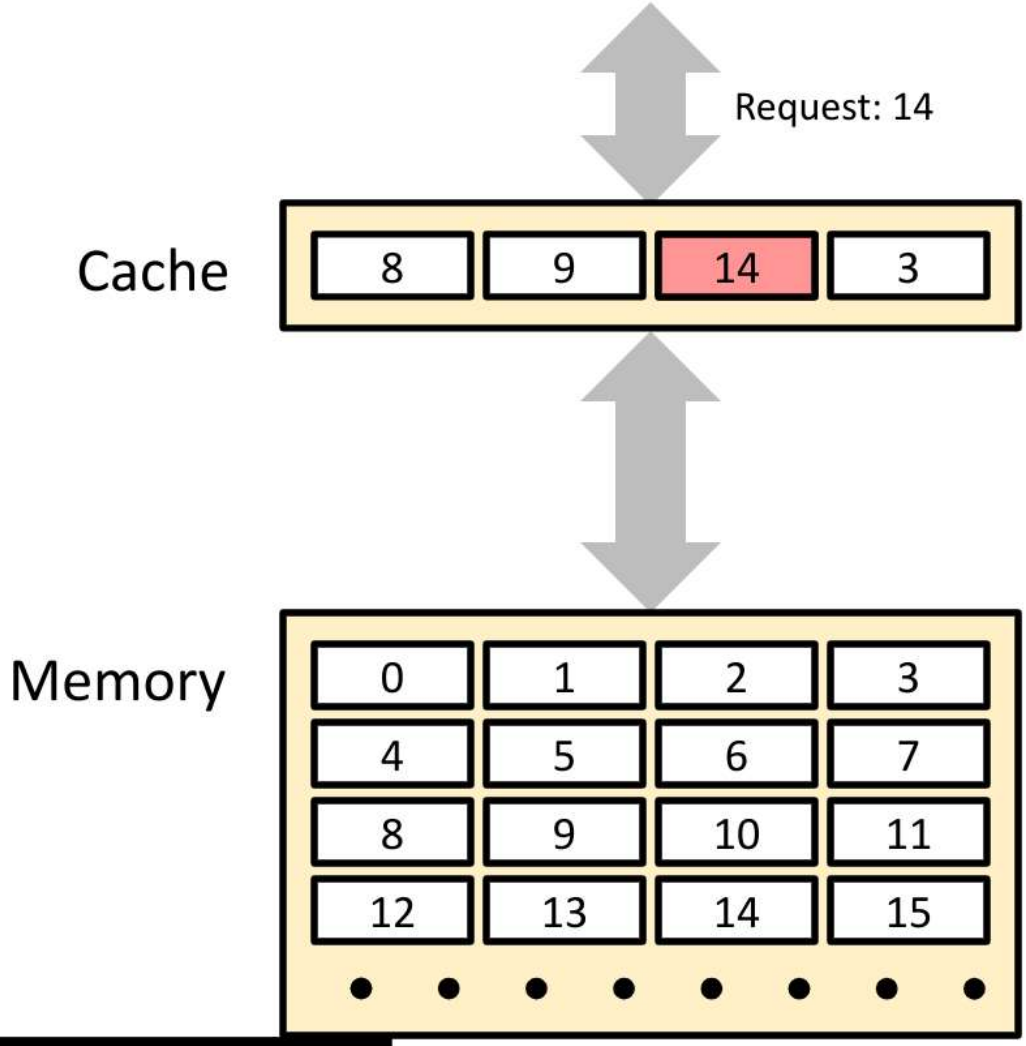


Data in line b is needed

UNIVERSITY OF COPENHAGEN

ITU CPH

General Cache Concepts: Hit



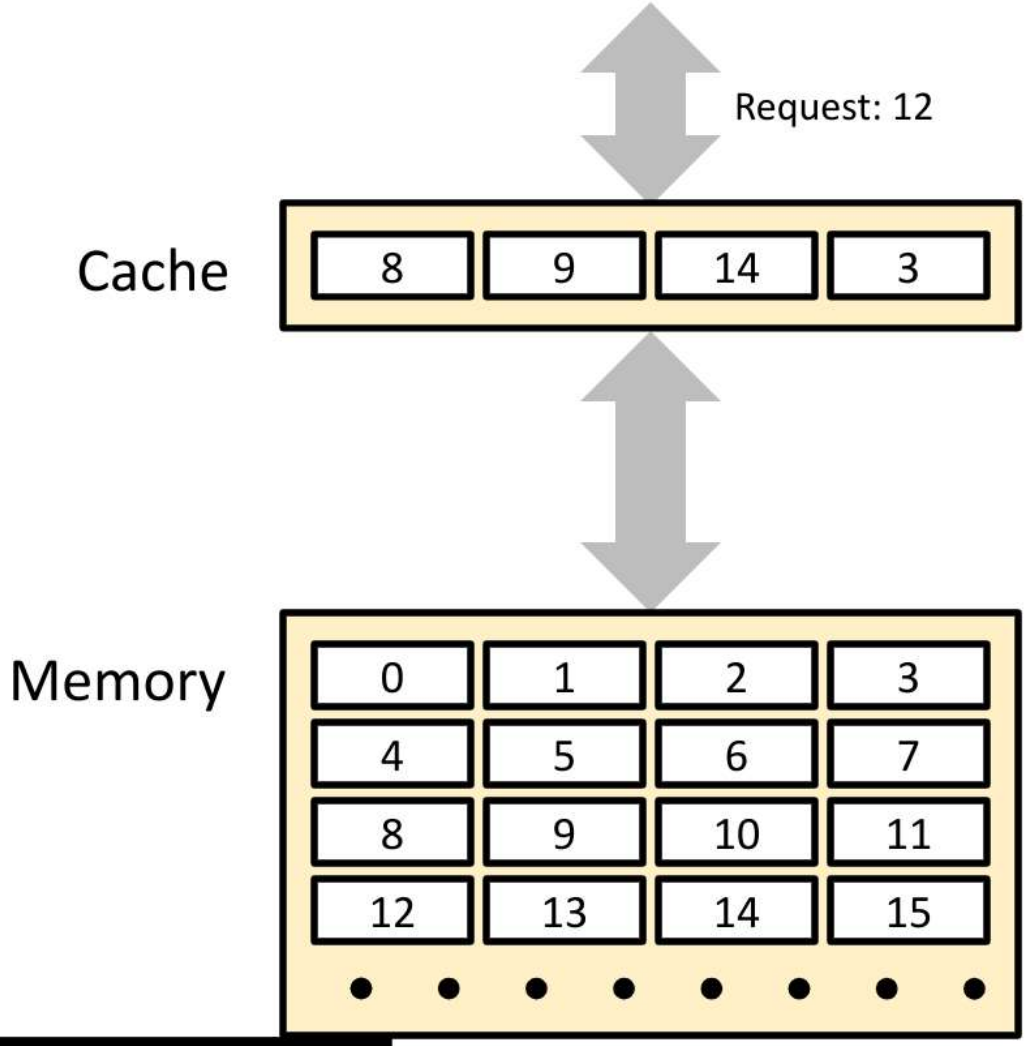
Data in line b is needed

line b is in cache:
Hit!

UNIVERSITY OF COPENHAGEN

ITU CPH

General Cache Concepts: Miss

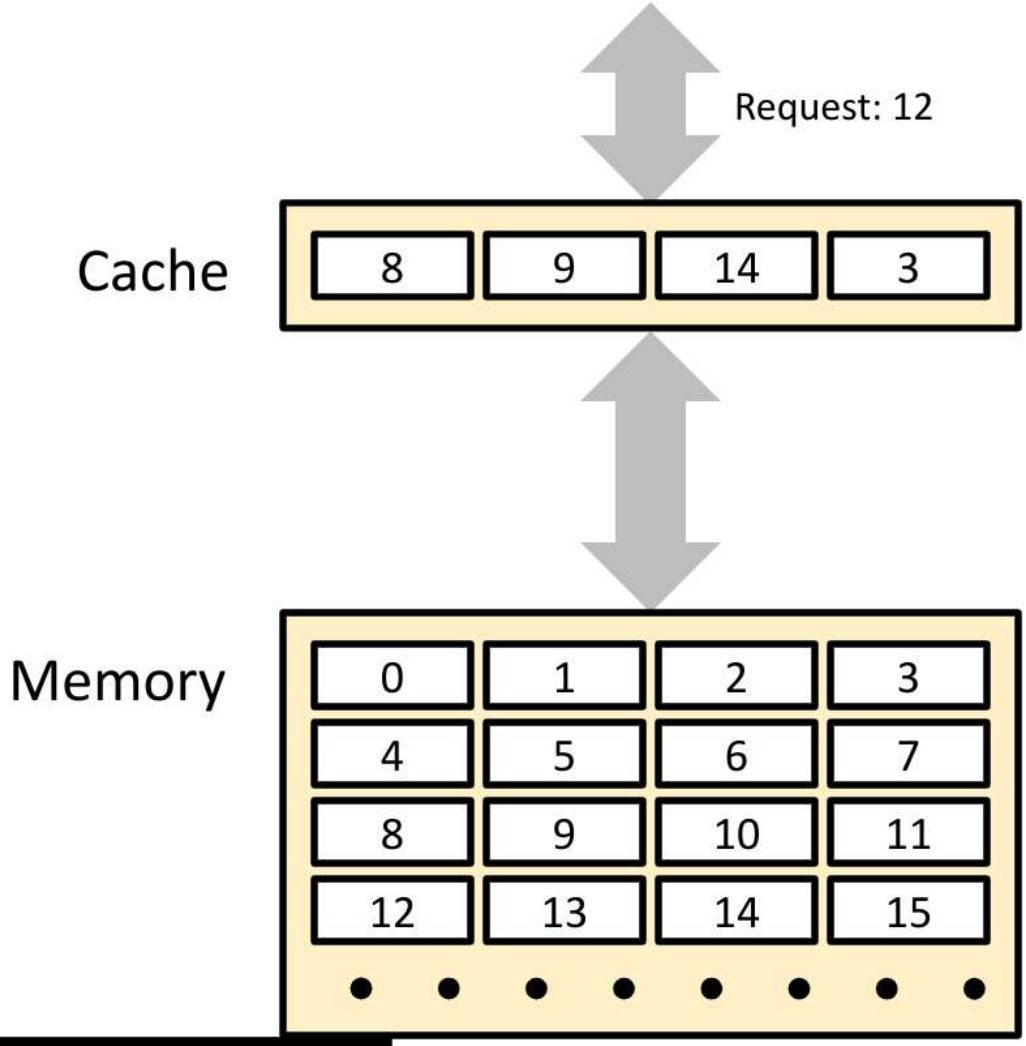


Data in line b is needed

UNIVERSITY OF COPENHAGEN

ITU CPH

General Cache Concepts: Miss



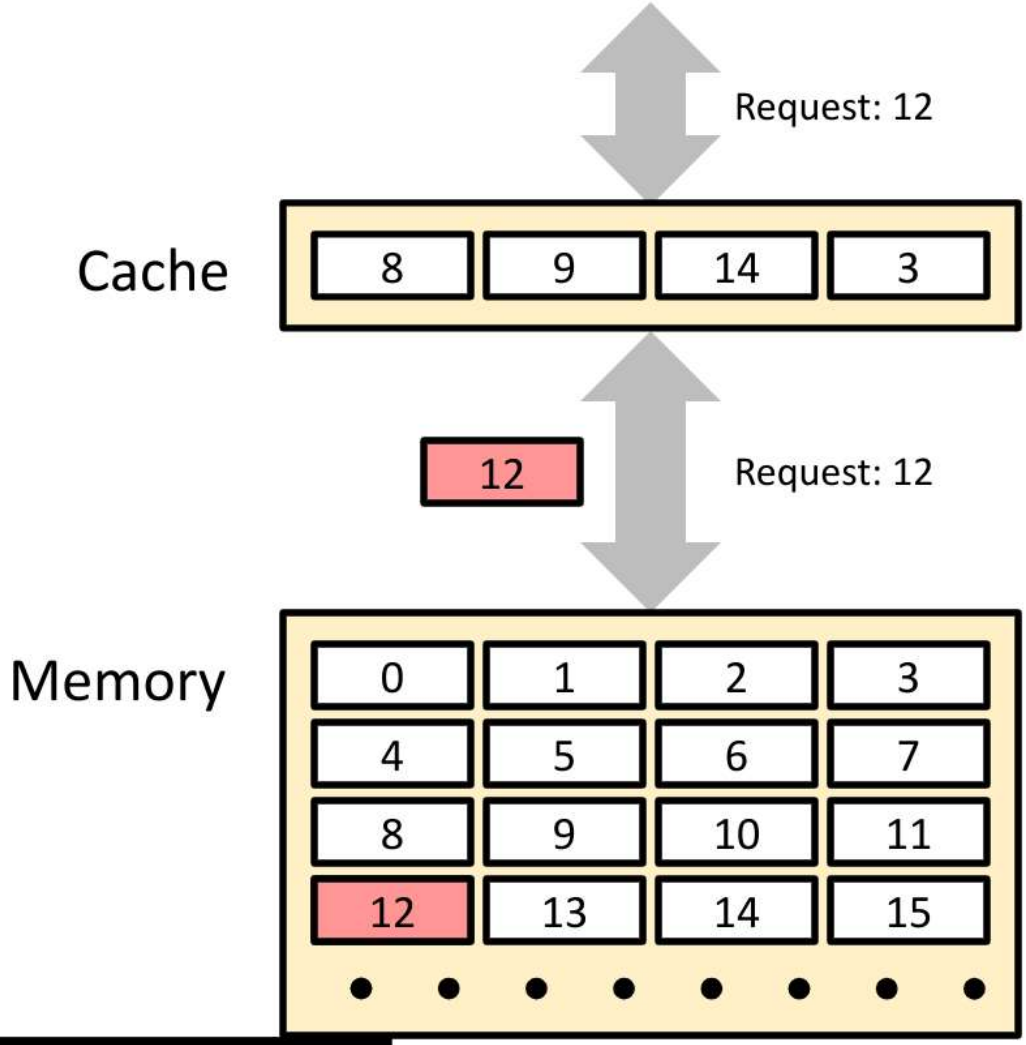
Data in line b is needed

Line b is not in cache:
Miss!

UNIVERSITY OF COPENHAGEN

ITU CPH

General Cache Concepts: Miss



Data in line b is needed

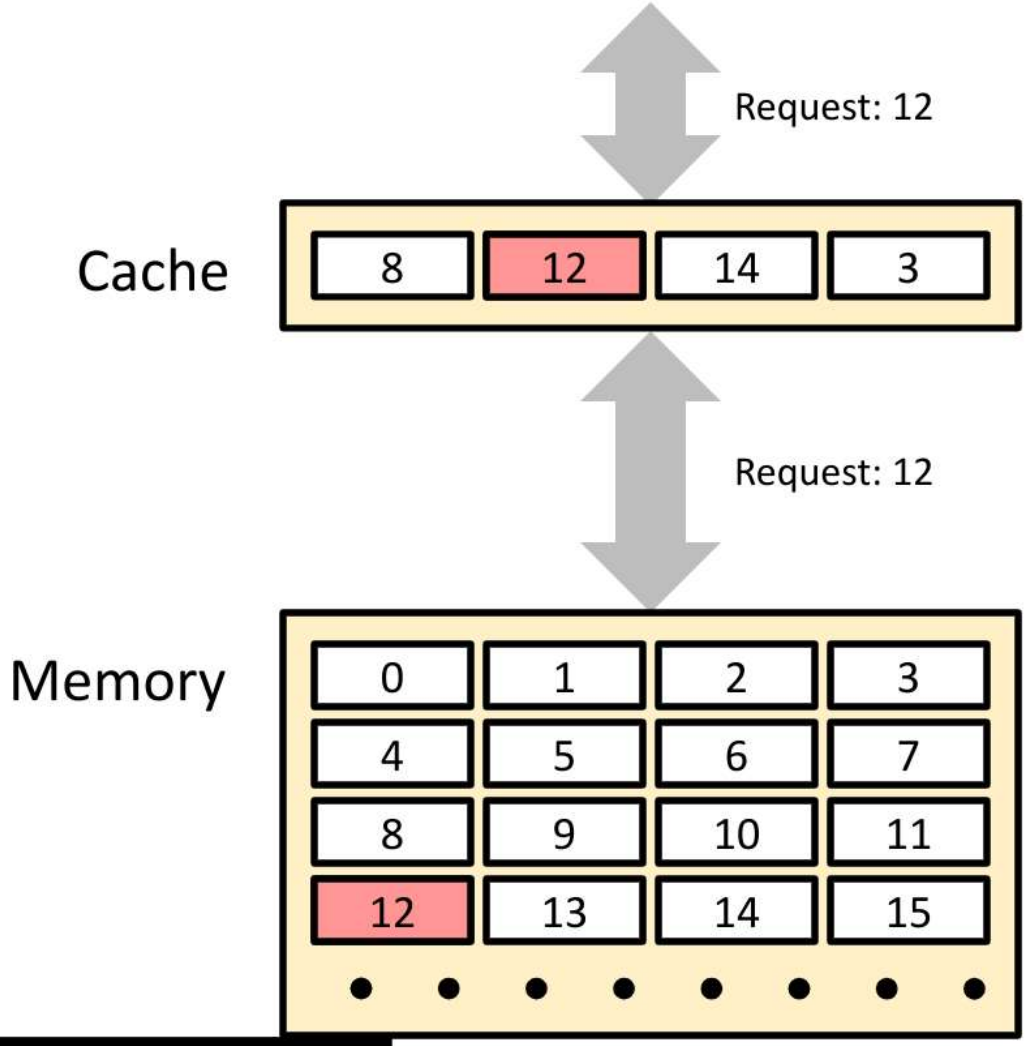
Line b is not in cache:
Miss!

Line b is fetched from *memory*

UNIVERSITY OF COPENHAGEN

ITU CPH

General Cache Concepts: Miss



Data in line b is needed

Line b is not in cache:
Miss!

Line b is fetched from *memory*

Line b is stored in cache

- Placement policy: determines where b goes
- Replacement policy: determines which block gets evicted (victim)

UNIVERSITY OF COPENHAGEN

ITU CPH

General Caching Concepts: Types of Cache Misses

Cold (compulsory) miss

Cold misses occur because the cache is empty.

Conflict miss

Most caches limit lines at level k+1 to a small subset (sometimes a singleton) of the line positions at level k.

- E.g. Line i at level k+1 must be placed in line (i mod 4) at level k.

Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k line.

- E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.

Capacity miss

Occurs when set of active cache lines (working set) is larger than the cache.

placement policy avoids conflict misses replacement policy care, to avoid capacity misses.

thrashing = every access miss

Examples of Caching in the Hierarchy

ex: access to memory i
100x more expensive
than L1 cache.

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	64-bytes line	On-Chip L1	1	Hardware
L2 cache	64-bytes line	On/Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

Cache Memories

given some data,
where is it going to be located?

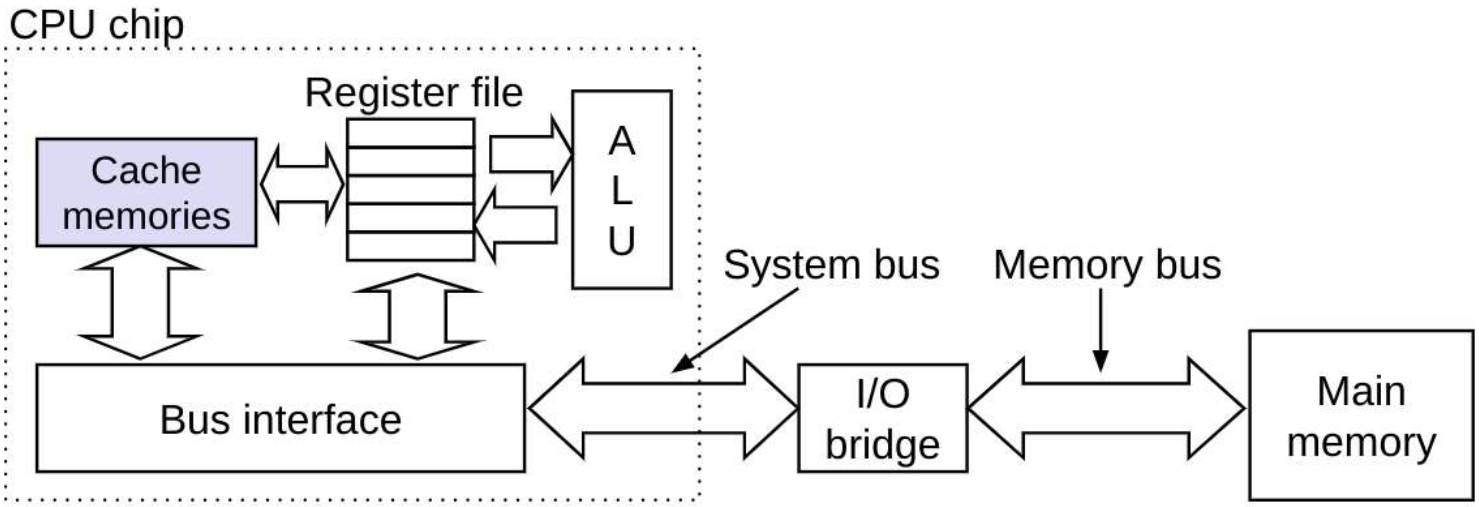
10s

Cache memories are small, fast SRAM-based memories managed automatically in hardware.

Hold frequently accessed blocks of main memory

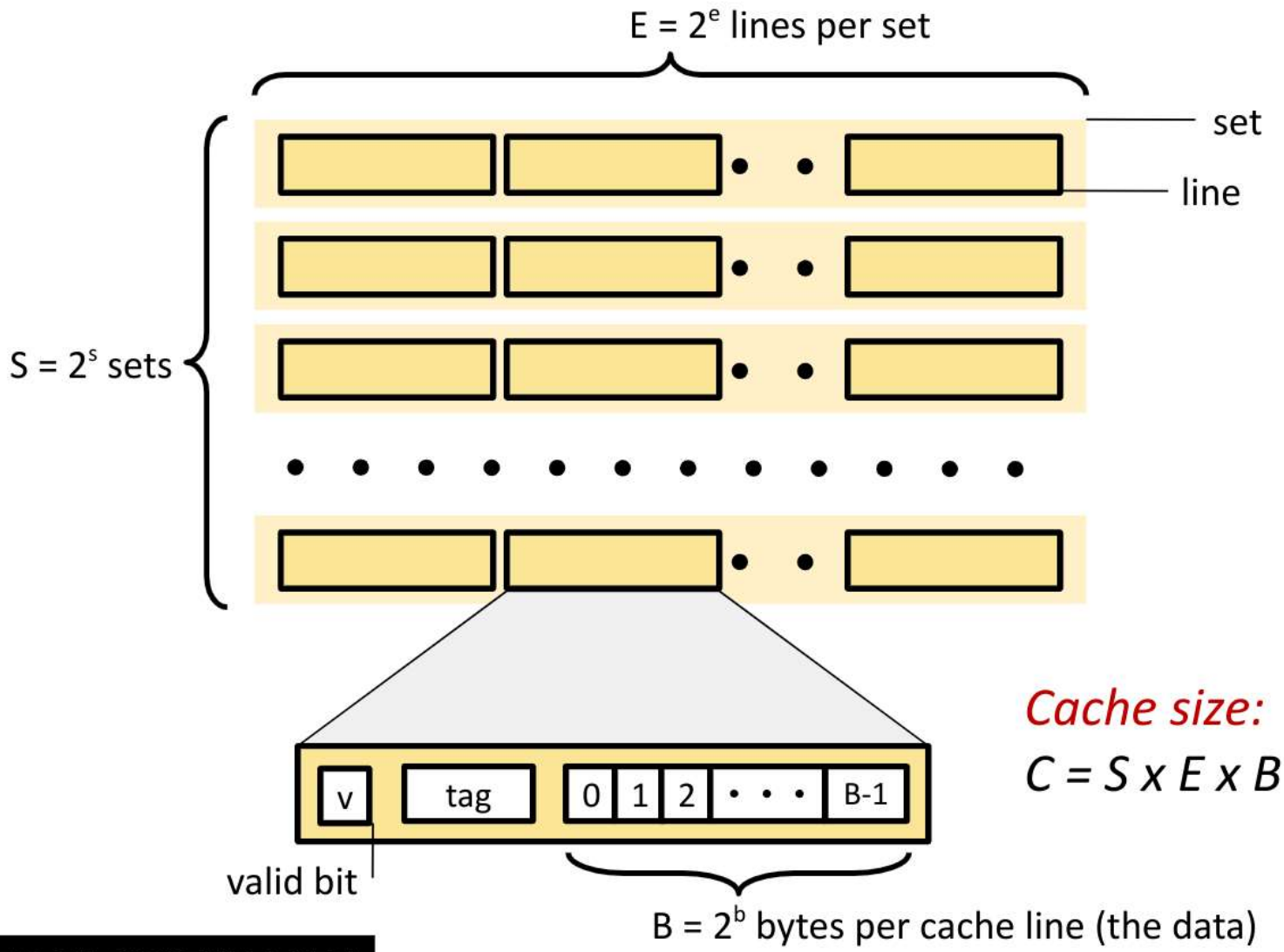
CPU looks first for data in caches (e.g., L1, L2, and L3), then in main memory

Typical system structure:



General Cache Organization (S, E, B)

organization of a cache.
have S sets, and E lines.

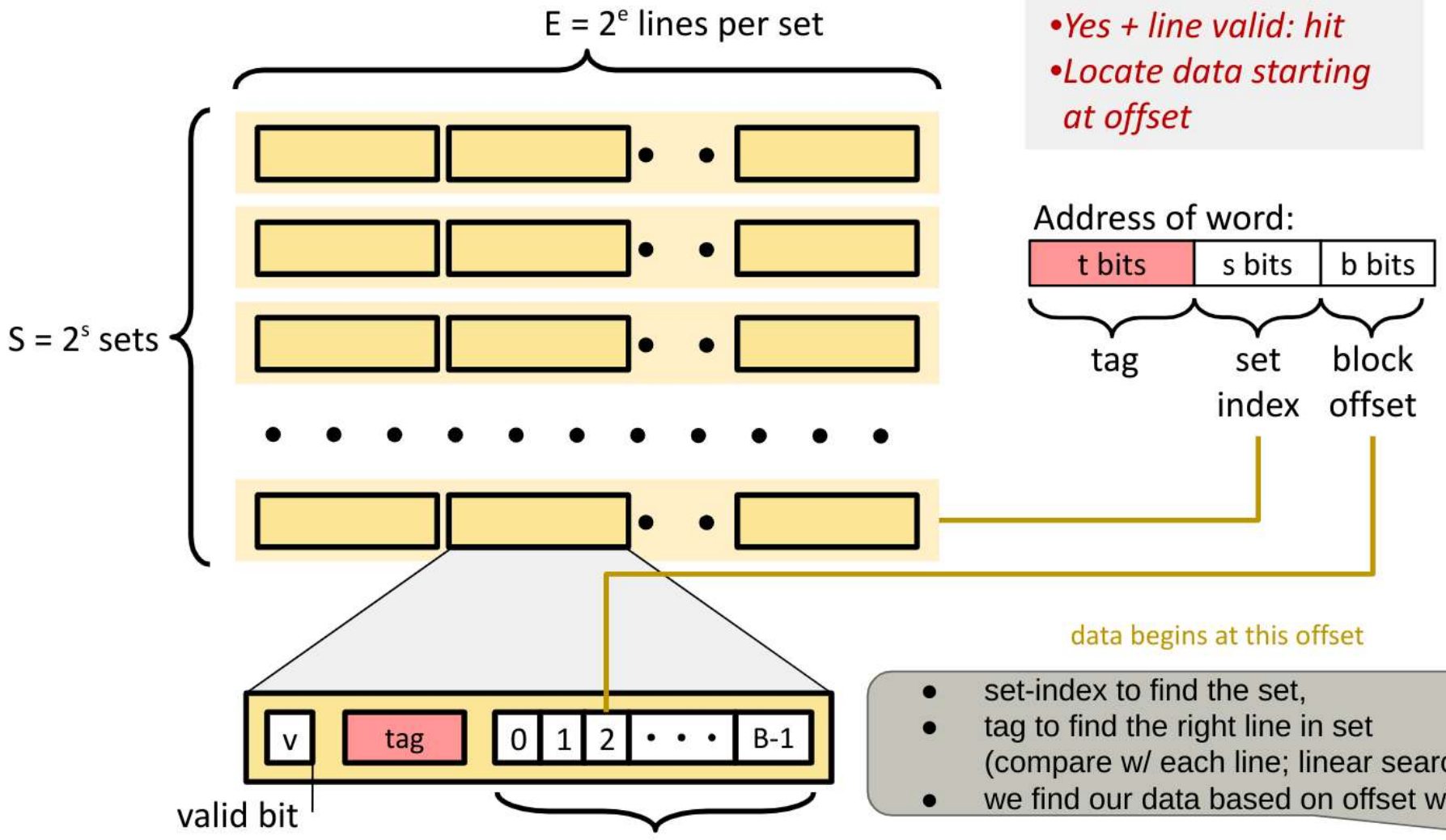


Cache size:

C = S x E x B data bytes

Cache Read

- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*
- *Locate data starting at offset*



- set-index to find the set,
- tag to find the right line in set (compare w/ each line; linear search)
- we find our data based on offset within line.

UNIVERSITY OF COPENHAGEN

ITU CPH

Direct-Mapped Cache Simulation

we ask for 1 byte.
but we transfer 1 line at a time.
here, a line is 2 bytes.

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

tag (0..1) set (0..3)

Address trace (reads, one byte per read):

0 [0$\underline{00}$0$_2$],
1 [0$\underline{00}$1$_2$],
7 [0$\underline{11}$1$_2$],
8 [1$\underline{00}$0$_2$],
0 [0$\underline{00}$0$_2$]

	v	Tag	Block
Set 0	0	?	?
Set 1			
Set 2			
Set 3			

UNIVERSITY OF COPENHAGEN

ITU CPH

Direct-Mapped Cache Simulation

we ask for 1 byte.
but we transfer 1 line at a time.
here, a line is 2 bytes.

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

tag (0..1)

set (0..3)

Address trace (reads, one byte per read):

0	[0$\underline{00}$0$_2$],	miss
1	[0$\underline{00}$1$_2$],	
7	[0$\underline{11}$1$_2$],	
8	[1$\underline{00}$0$_2$],	
0	[0$\underline{00}$0$_2$]	

	v	Tag	Block
Set 0	0	?	?
Set 1			
Set 2			
Set 3			

UNIVERSITY OF COPENHAGEN

ITU CPH

Direct-Mapped Cache Simulation

we ask for 1 byte.
but we transfer 1 line at a time.
here, a line is 2 bytes.

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

tag (0..1) set (0..3)

Address trace (reads, one byte per read):

0	[0$\underline{00}$0$_2$],	miss
1	[0$\underline{00}$1$_2$],	
7	[0$\underline{11}$1$_2$],	
8	[1$\underline{00}$0$_2$],	
0	[0$\underline{00}$0$_2$]	

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3			

UNIVERSITY OF COPENHAGEN

ITU CPH

Direct-Mapped Cache Simulation

we ask for 1 byte.
but we transfer 1 line at a time.
here, a line is 2 bytes.

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

tag (0..1) set (0..3)

Address trace (reads, one byte per read):

0	[0$\underline{00}$0$_2$],	miss
1	[0$\underline{00}$1$_2$],	hit
7	[0$\underline{11}$1$_2$],	
8	[1$\underline{00}$0$_2$],	
0	[0$\underline{00}$0$_2$]	

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3			

UNIVERSITY OF COPENHAGEN

ITU CPH

Direct-Mapped Cache Simulation

we ask for 1 byte.
but we transfer 1 line at a time.
here, a line is 2 bytes.

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

tag (0..1) set (0..3)

Address trace (reads, one byte per read):

0	[0$\underline{00}$0$_2$],	miss
1	[0$\underline{00}$1$_2$],	hit
7	[0$\underline{11}$1$_2$],	miss
8	[1$\underline{00}$0$_2$],	
0	[0$\underline{00}$0$_2$]	

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3			

UNIVERSITY OF COPENHAGEN

ITU CPH

Direct-Mapped Cache Simulation

we ask for 1 byte.
but we transfer 1 line at a time.
here, a line is 2 bytes.

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

tag (0..1) set (0..3)

Address trace (reads, one byte per read):

0	[0$\underline{00}$0$_2$],	miss
1	[0$\underline{00}$1$_2$],	hit
7	[0$\underline{11}$1$_2$],	miss
8	[1$\underline{00}$0$_2$],	
0	[0$\underline{00}$0$_2$]	

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

UNIVERSITY OF COPENHAGEN

ITU CPH

Direct-Mapped Cache Simulation

we ask for 1 byte.
but we transfer 1 line at a time.
here, a line is 2 bytes.

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

tag (0..1) set (0..3)

Address trace (reads, one byte per read):

0	[0$\underline{00}$0$_2$],	miss
1	[0$\underline{00}$1$_2$],	hit
7	[0$\underline{11}$1$_2$],	miss
8	[1$\underline{00}$0$_2$],	miss
0	[0$\underline{00}$0$_2$]	

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

UNIVERSITY OF COPENHAGEN

ITU CPH

Direct-Mapped Cache Simulation

we ask for 1 byte.
but we transfer 1 line at a time.
here, a line is 2 bytes.

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

tag (0..1) set (0..3)

Address trace (reads, one byte per read):

0	[0$\underline{00}$0$_2$],	miss
1	[0$\underline{00}$1$_2$],	hit
7	[0$\underline{11}$1$_2$],	miss
8	[1$\underline{00}$0$_2$],	miss
0	[0$\underline{00}$0$_2$]	

	v	Tag	Block
Set 0	1	1	M[8-9]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

UNIVERSITY OF COPENHAGEN

ITU CPH

Direct-Mapped Cache Simulation

we ask for 1 byte.
but we transfer 1 line at a time.
here, a line is 2 bytes.

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

tag (0..1) set (0..3)

Address trace (reads, one byte per read):

0	[0$\underline{00}$0$_2$],	miss
1	[0$\underline{00}$1$_2$],	hit
7	[0$\underline{11}$1$_2$],	miss
8	[1$\underline{00}$0$_2$],	miss
0	[0$\underline{00}$0$_2$]	miss

	v	Tag	Block
Set 0	1	1	M[8-9]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

UNIVERSITY OF COPENHAGEN

ITU CPH

Direct-Mapped Cache Simulation

we ask for 1 byte.
but we transfer 1 line at a time.
here, a line is 2 bytes.

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

tag (0..1) set (0..3)

Address trace (reads, one byte per read):

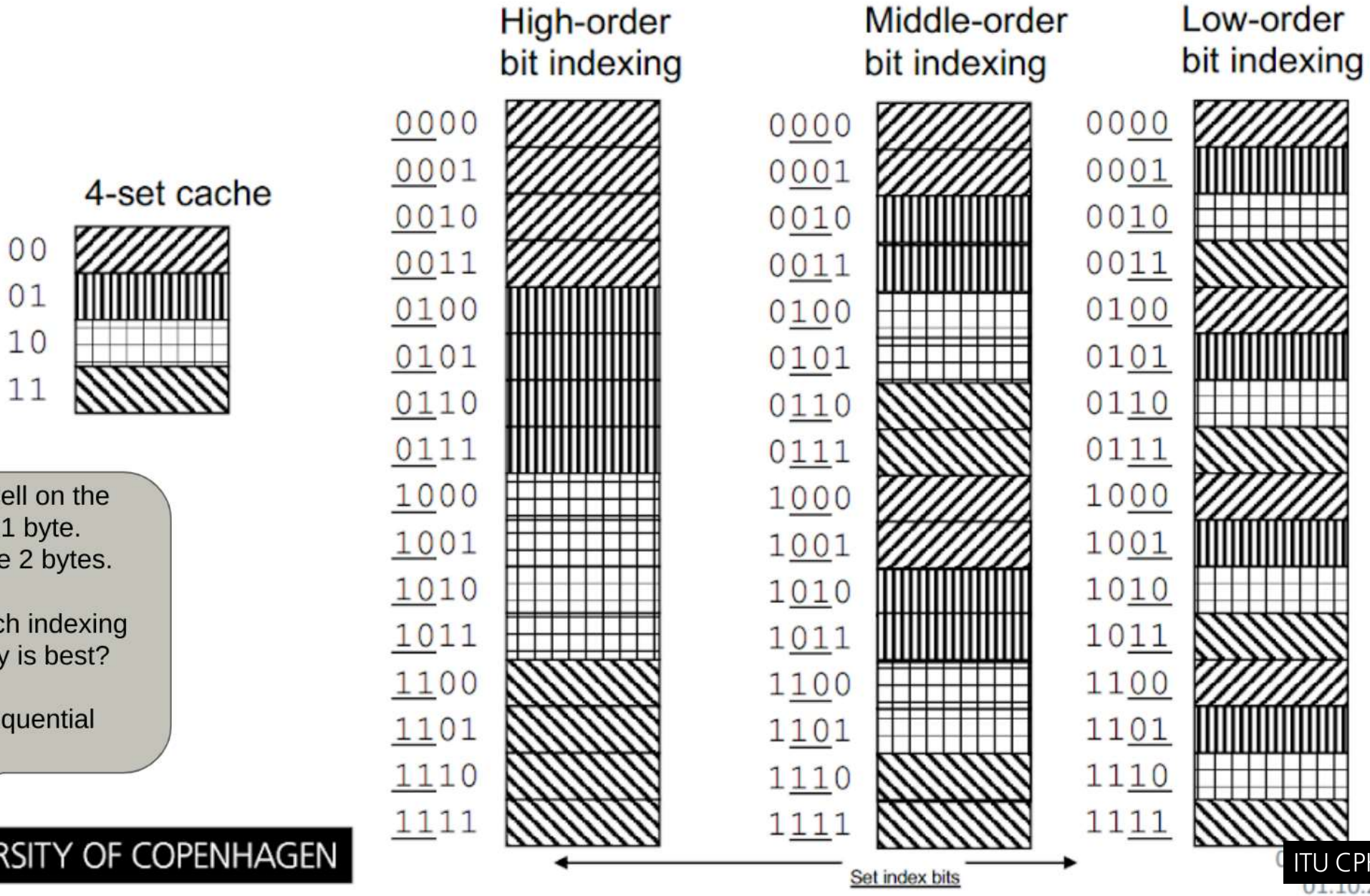
0	[0$\underline{00}$0$_2$],	miss
1	[0$\underline{00}$1$_2$],	hit
7	[0$\underline{11}$1$_2$],	miss
8	[1$\underline{00}$0$_2$],	miss
0	[0$\underline{00}$0$_2$]	miss

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

UNIVERSITY OF COPENHAGEN

ITU CPH

Why index with the middle bits?



every cell on the right is 1 byte.
line size 2 bytes.

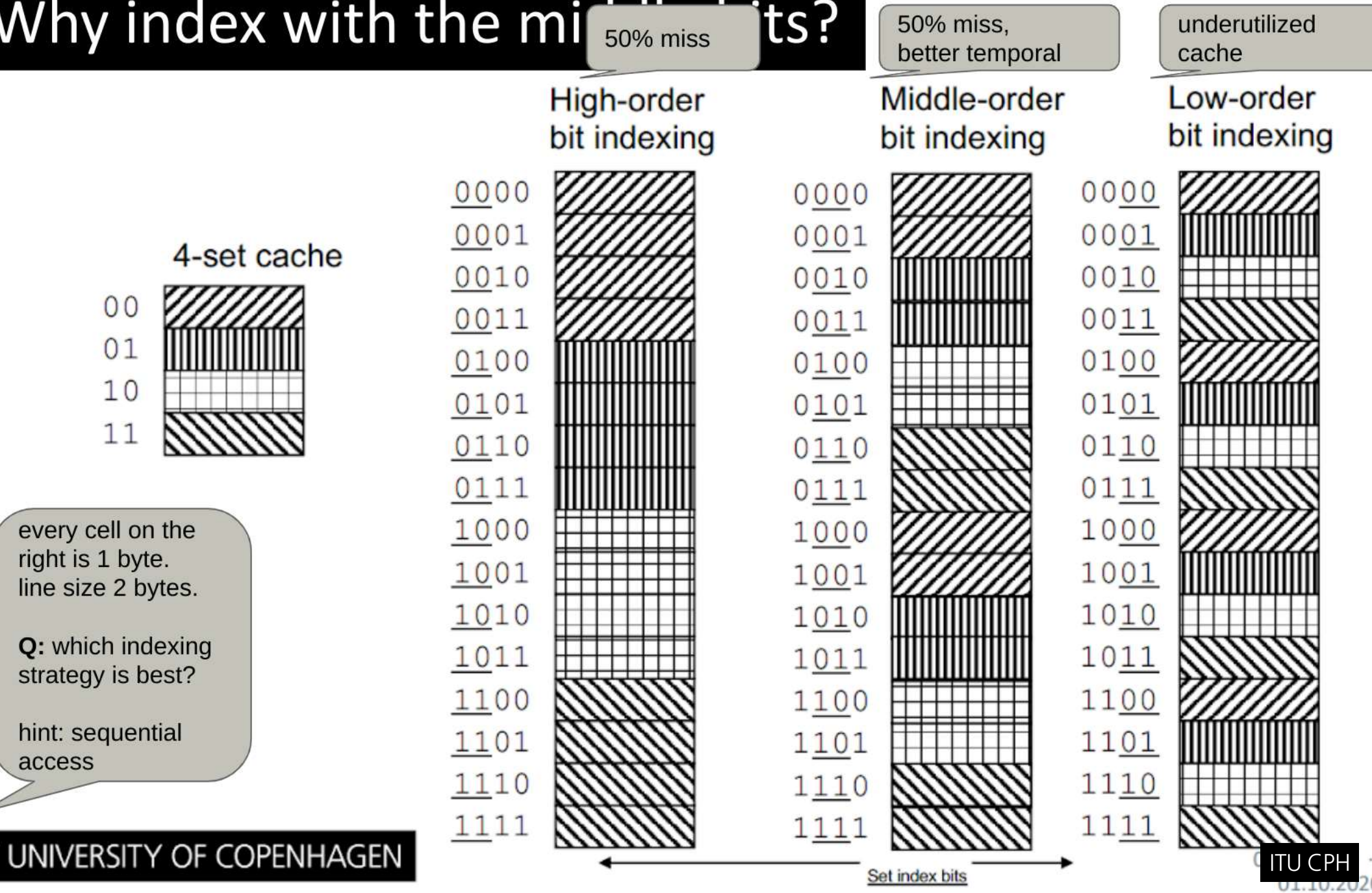
Q: which indexing strategy is best?

hint: sequential access

UNIVERSITY OF COPENHAGEN

ITU CPH

Why index with the mi ts?



UNIVERSITY OF COPENHAGEN

ITU CPH

What about writes?

Multiple copies of data exist:

- L1, L2, Main Memory, Disk

What to do on a write-hit?

- Write-through (write immediately to memory)
- Write-back (defer write to memory until replacement of line)
  - Need a dirty bit (line different from memory or not)

faster

What to do on a write-miss?

- Write-allocate (load into cache, update line in cache)
  - Good if more writes to the location follow
- No-write-allocate (writes immediately to memory)

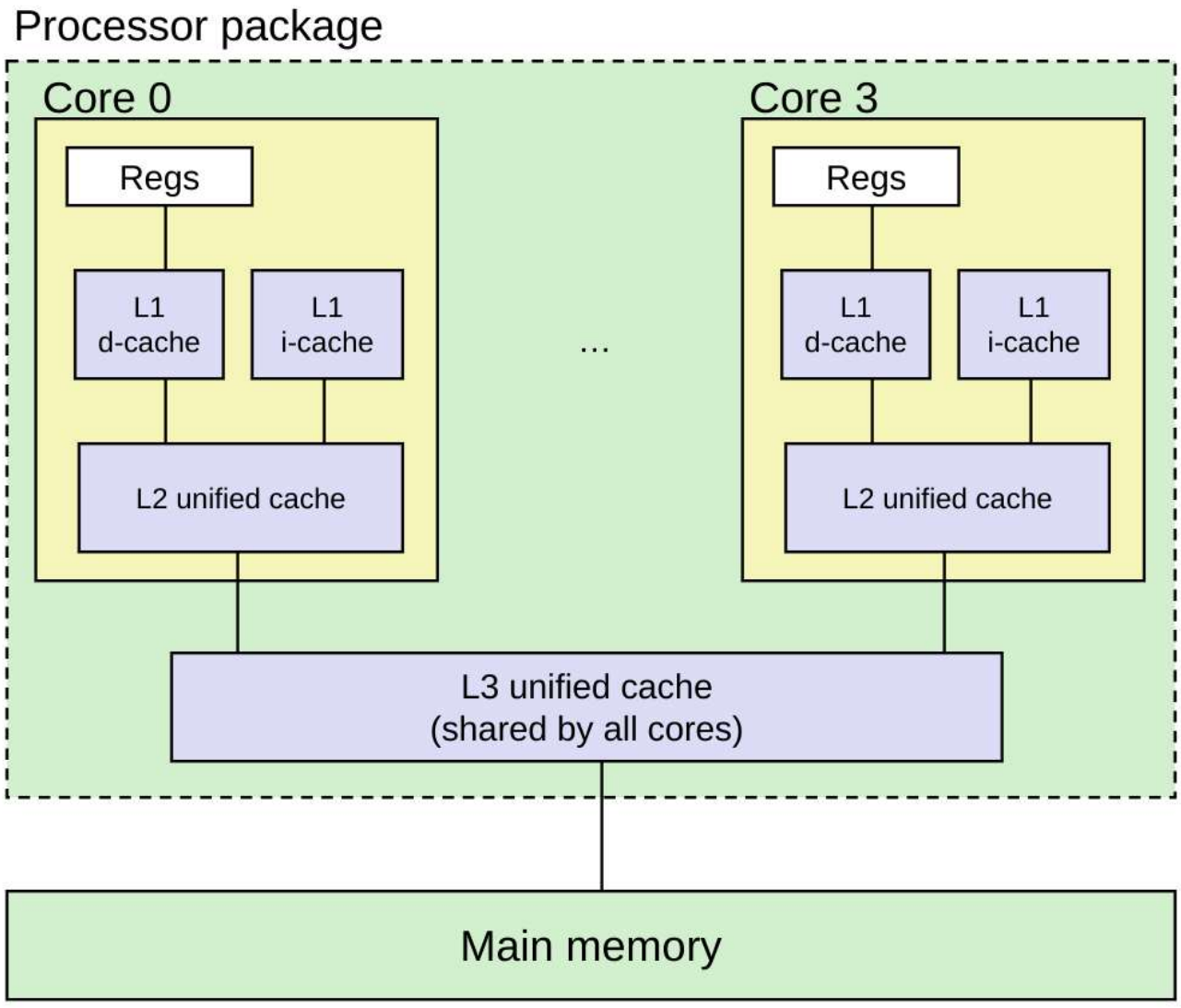
Typical

- Write-through + No-write-allocate
- **Write-back + Write-allocate**

UNIVERSITY OF COPENHAGEN

ITU CPH

Intel Core i7 Cache Hierarchy



L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Line size: 64 bytes for all caches.

UNIVERSITY OF COPENHAGEN

ITU CPH

Cache Performance Metrics

10s

- **Miss Rate**
  - Fraction of memory references not found in cache (misses / accesses) = **1 – hit rate**
  - Typical numbers (in percentages):
  - 3-10% for L1
  - can be quite small (e.g., < 1%) for L2, depending on size, etc.
- **Hit Time**
  - Time to deliver a line in the cache to the processor
  - includes time to determine whether the line is in the cache
  - Typical numbers:
  - 1-2 clock cycle for L1
  - 5-20 clock cycles for L2
- **Miss Penalty**
  - Additional time required because of a miss
  - typically 50-200 cycles for main memory (Trend: increasing!)

et's think about those numbers

Huge difference between a hit and a miss

Could be 100x, if just L1 and main memory

Would you believe 99% hits is twice as good as 97%?

Consider:

cache hit time of 1 cycle
miss penalty of 100 cycles

Average access time:

97% hits: 1 cycle + 0.03 * 100 cycles = **4 cycles**

99% hits: 1 cycle + 0.01 * 100 cycles = **2 cycles**

This is why "miss rate" is used instead of "hit rate"

Writing Cache Friendly Code

Make the common case go fast

Focus on the inner loops of the core functions

Minimize the misses in the inner loops

Repeated references to variables are good (temporal locality)

Stride-1 reference patterns are good (spatial locality)

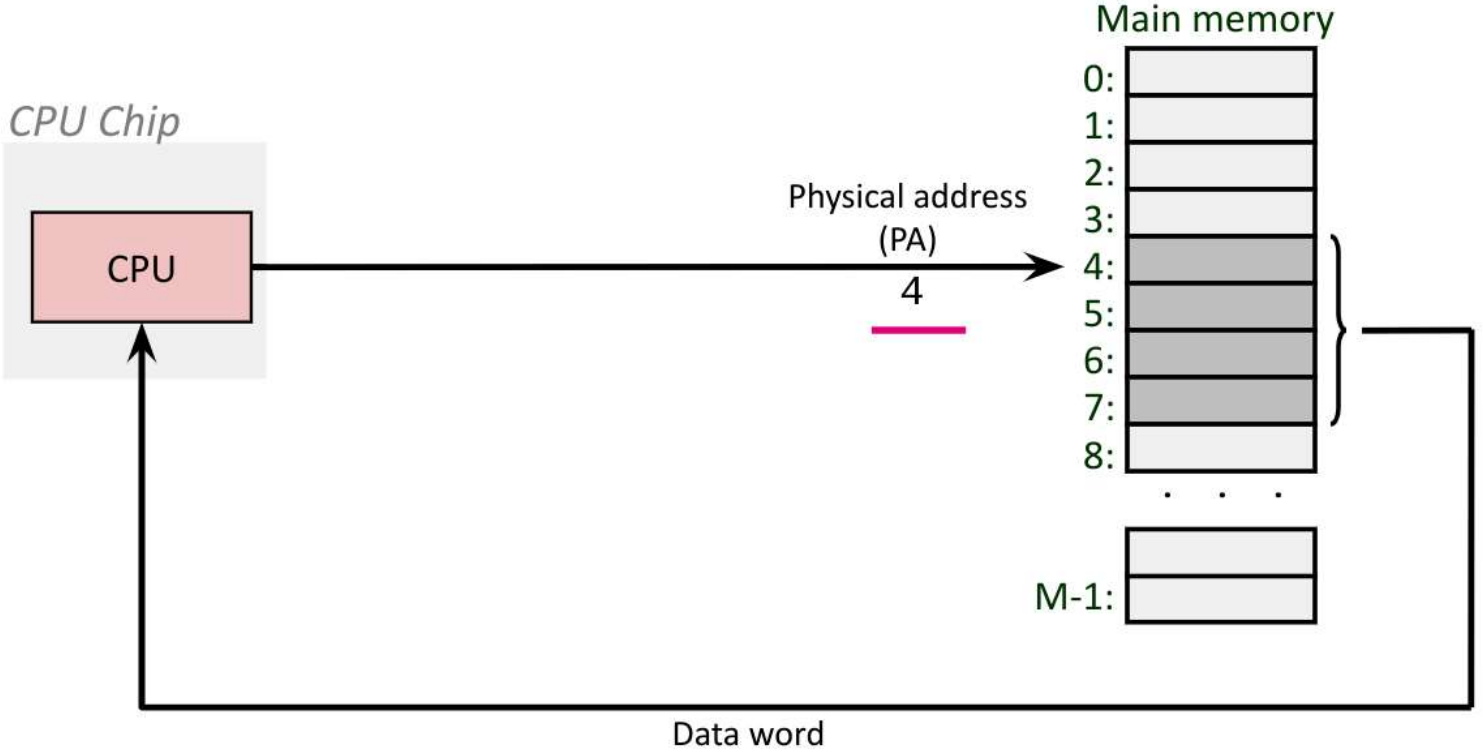
Key idea: Our qualitative notion of locality is quantified through our understanding of cache memories.

Virtual Memory

ITU CPH

A System Using **Physical** Addressing

address in CPU request is an actual, physical address.



Used in “simple” systems like embedded microcontrollers in devices like cars, elevators, and digital picture frames

UNIVERSITY OF COPENHAGEN

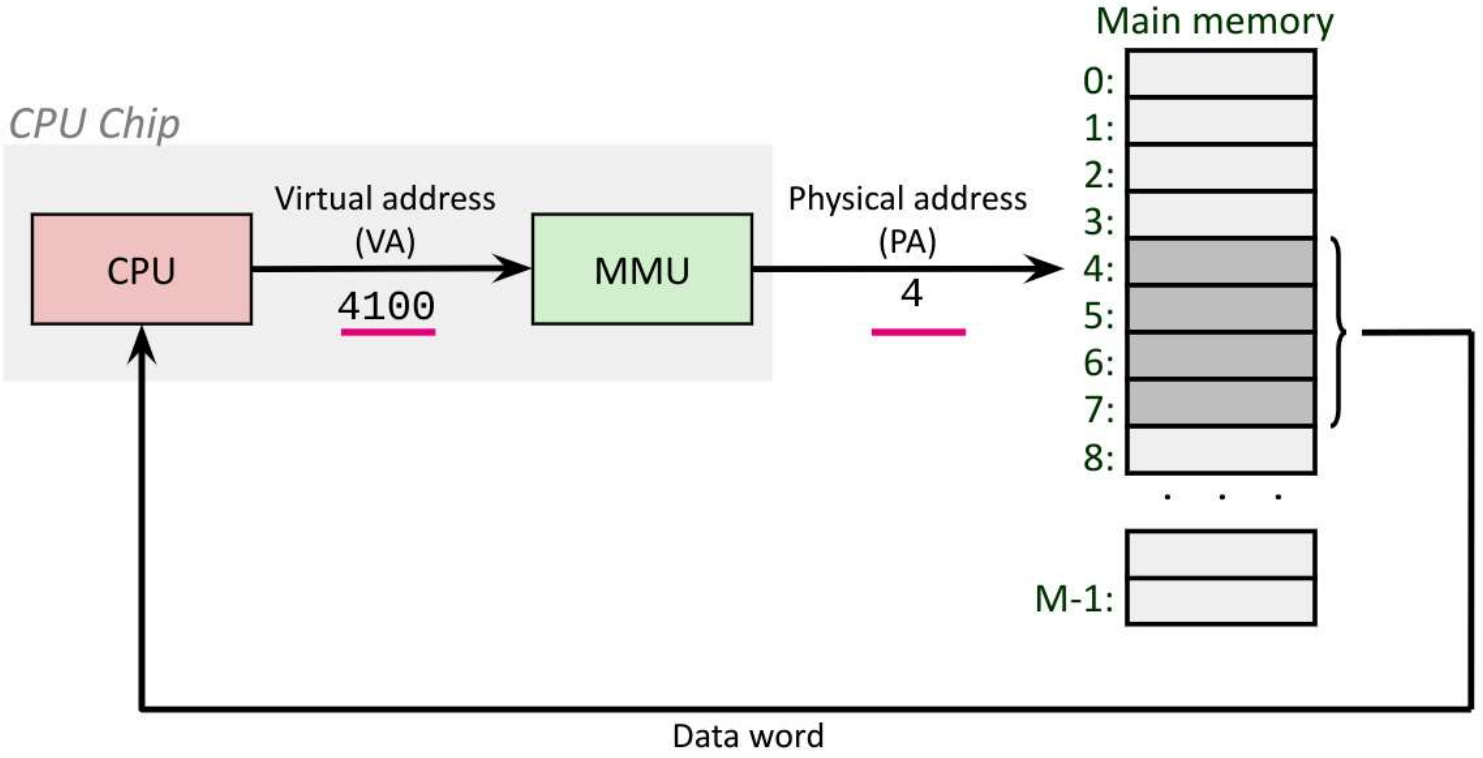
a **level of indirection** is useful (gives us options).

ITU CPH

A System Using **Virtual** Addressing

level of indirection: maps virtual addresses to physical addresses.

MMU: Memory Management Unit



Used in all modern servers, desktops, and laptops

One of the great ideas in computer science

MMU can do some key clever things. let's walk through that.

UNIVERSITY OF COPENHAGEN

ITU CPH

Virtual Memory

what does virtual memory give us? (i.e. why VM?)

- Uses main memory efficiently **performance**
  - Use DRAM as a cache for the parts of a virtual address space
- Simplifies memory management
  - Each process gets the same uniform linear address space (from 0 and up)
- Isolates address spaces **security**
  - One process can't interfere with another's memory
  - User program cannot access privileged kernel information

Address Space

10s

An address space is an ordered set of contiguous addresses (non-negative integers)

Physical address space ⇒ associated to RAM

Virtual address space ⇒ associated to each **process**

from *point of view* of **process**, it has access to whole memory, and that memory belongs to it.
in actuality, it has access to parts of memory, some shared w/ other pro

VM is a Tool for Caching

(i.e. for organizing how physical memory is used)

Virtual memory is an array of N contiguous bytes stored **on disk**.

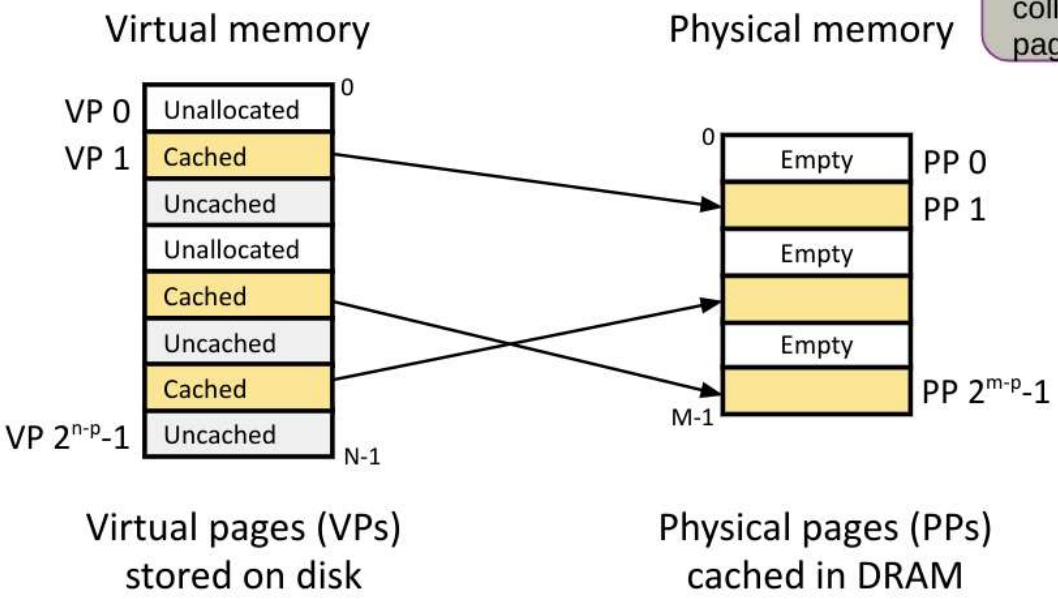
The contents of the array on disk are **cached** in *physical memory* (*DRAM cache*)

These cache blocks are called *pages* (size (quanta) P = 2^p bytes)

Unit of transfer between disk & virtual memory

4k, 8k, 16k
VM thus = a collection of pages

Q: how much VM can we have?



VM is a Tool for Caching

(i.e. for organizing how physical memory is used)

Virtual memory is an array of N contiguous bytes stored **on disk**.

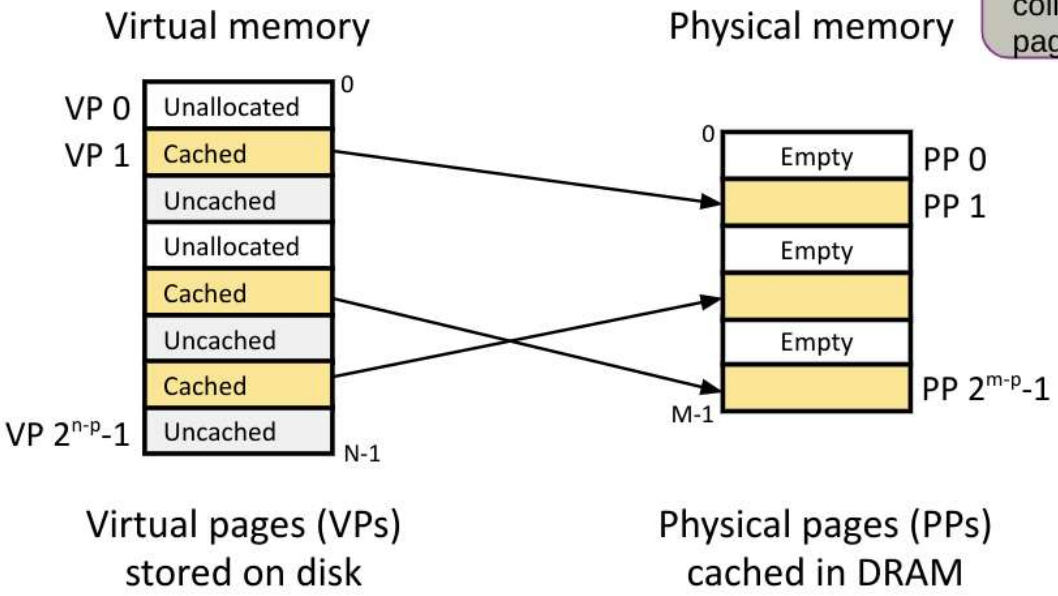
The contents of the array on disk are **cached** in *physical memory* (*DRAM cache*)

These cache blocks are called *pages* (size (quanta) P = 2^p bytes)

Unit of transfer between disk & virtual memory

4k, 8k, 16k
VM thus = a collection of pages

Q: how much VM can we have?
A: CPU issues address requests. VM size bounded by CPU word size



UNIVERSITY OF COPENHAGEN

ITU CPH

VM is a Tool for Caching

(i.e. for organizing how physical memory is used)

Virtual memory is an array of N contiguous bytes stored **on disk**.

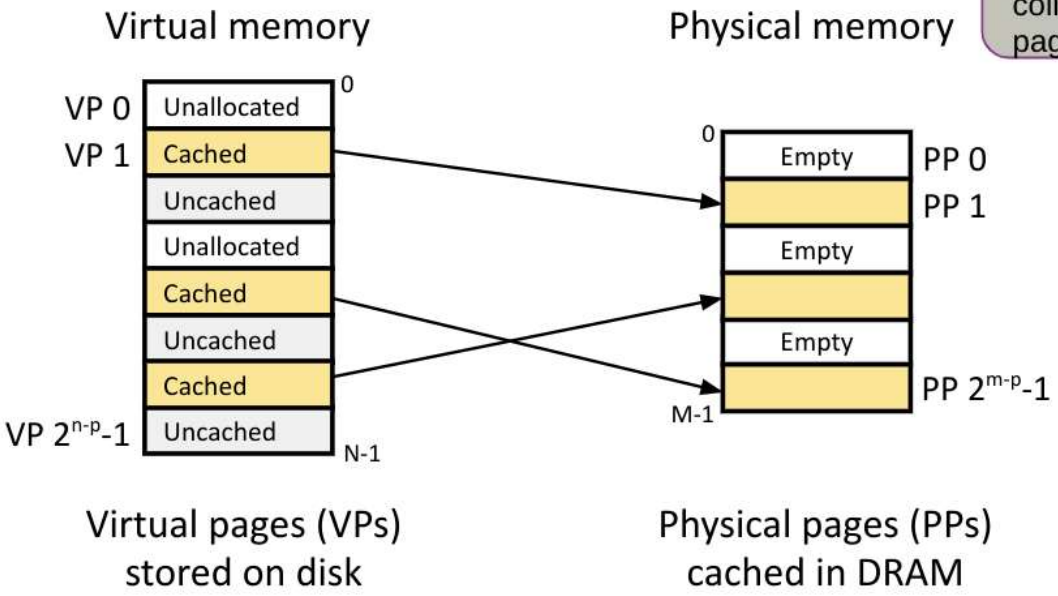
The contents of the array on disk are **cached** in *physical memory* (*DRAM cache*)

These cache blocks are called *pages* (size (quanta) P = 2^p bytes)

Unit of transfer between disk & virtual memory

4k, 8k, 16k
VM thus = a collection of pages

Q: how much VM can we have?
A: CPU issues address requests. VM size bounded by CPU word size



when you run out of memory, you don't run out of addresses, but actual phys mem

evel of indirection reates opportunities!

history; 2 GB limit

physical address extension (PAE): MMU "hack" enabling more than 2^32 physical mem despite CPU being 32-bit

today, address space size:
virtual mem ≫ physical mem
VM creates illusion (to proc) of more memory, using cac

UNIVERSITY OF COPENHAGEN

ITU CPH

DRAM Cache Organization

DRAM for phys. mem, SRAM for registers

DRAM cache organization driven by the enormous miss penalty

- DRAM is about ***10x*** slower than SRAM
- Disk is about ***10,000x*** slower than DRAM

Consequences:

- Large page (block) size: typically 4-8 KB, sometimes 4 MB
- Fully associative
  - Any VP can be placed in any PP
  - Need a "large" mapping function – different from CPU caches
- Highly sophisticated, expensive replacement algorithms
  - Too complicated and open-ended to be implemented in hardware
- Write-back rather than write-through

Cannot use the (simple) mechanism we saw in CPU caching

solution: page table (next slide)

UNIVERSITY OF COPENHAGEN

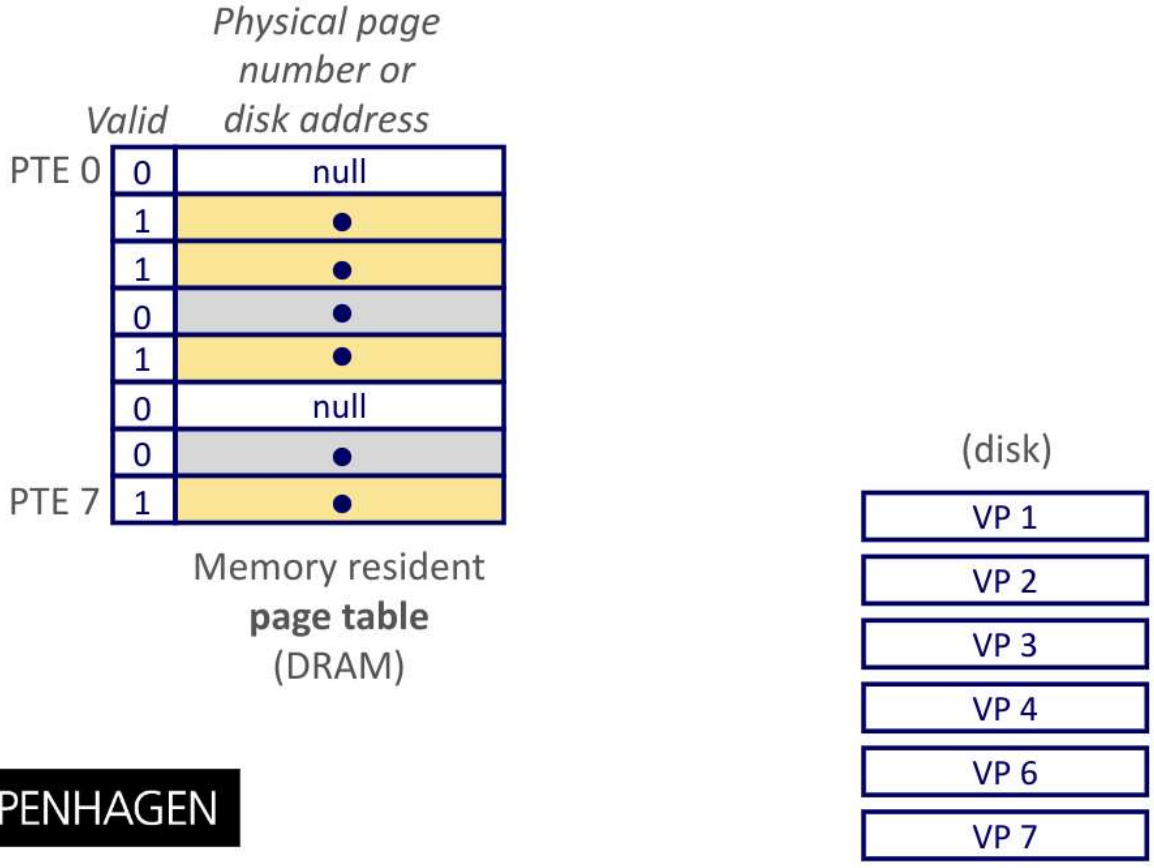
ITU CPH

Page Tables

serves as our virt-phys address mapping

A *page table* is an array of page table entries (PTEs) that maps **virtual** pages to **physical** pages.

A per-process kernel data structure in DRAM



UNIVERSITY OF COPENHAGEN

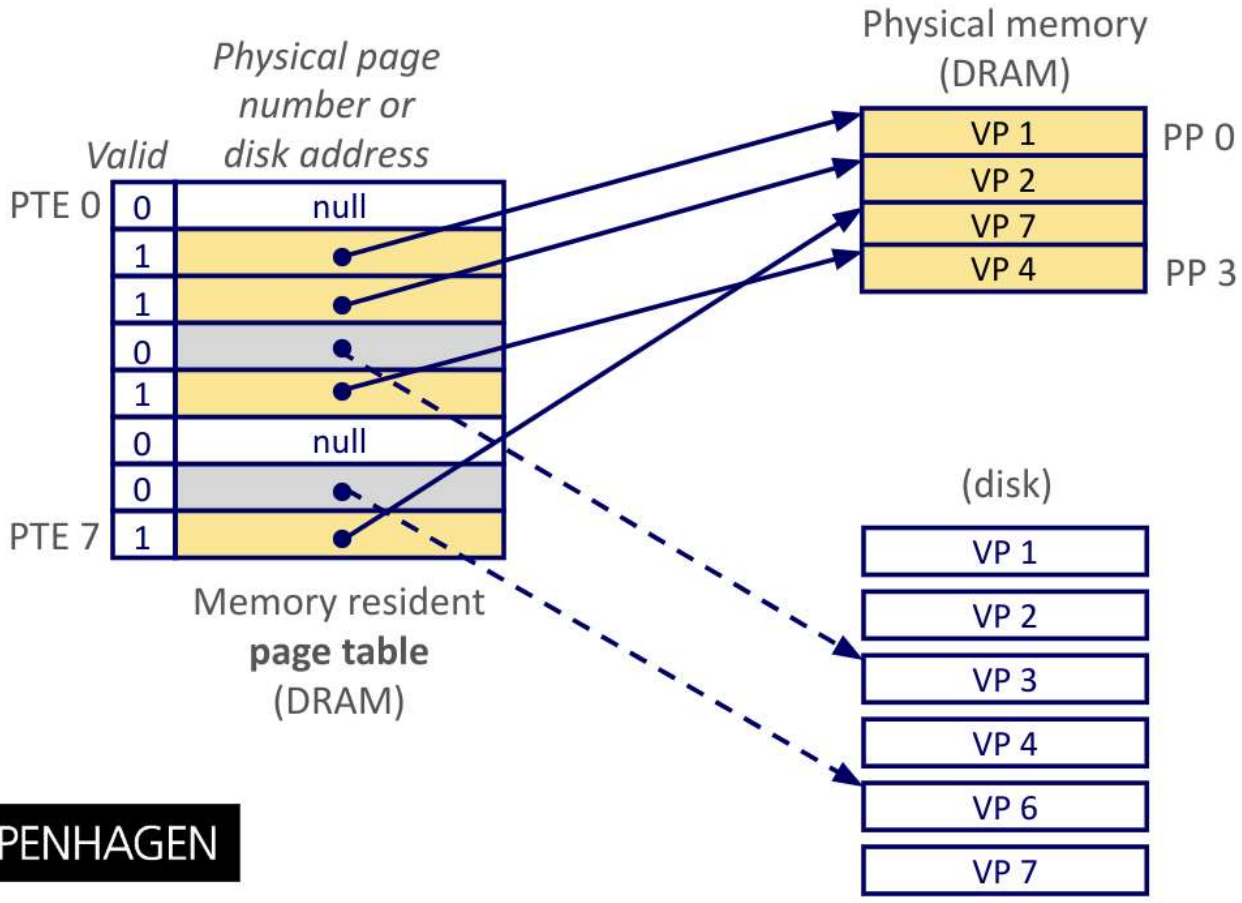
ITU CPH

Page Tables

serves as our virt-phys address mapping

A *page table* is an array of page table entries (PTEs) that maps **virtual** pages to **physical** pages.

A per-process kernel data structure in DRAM



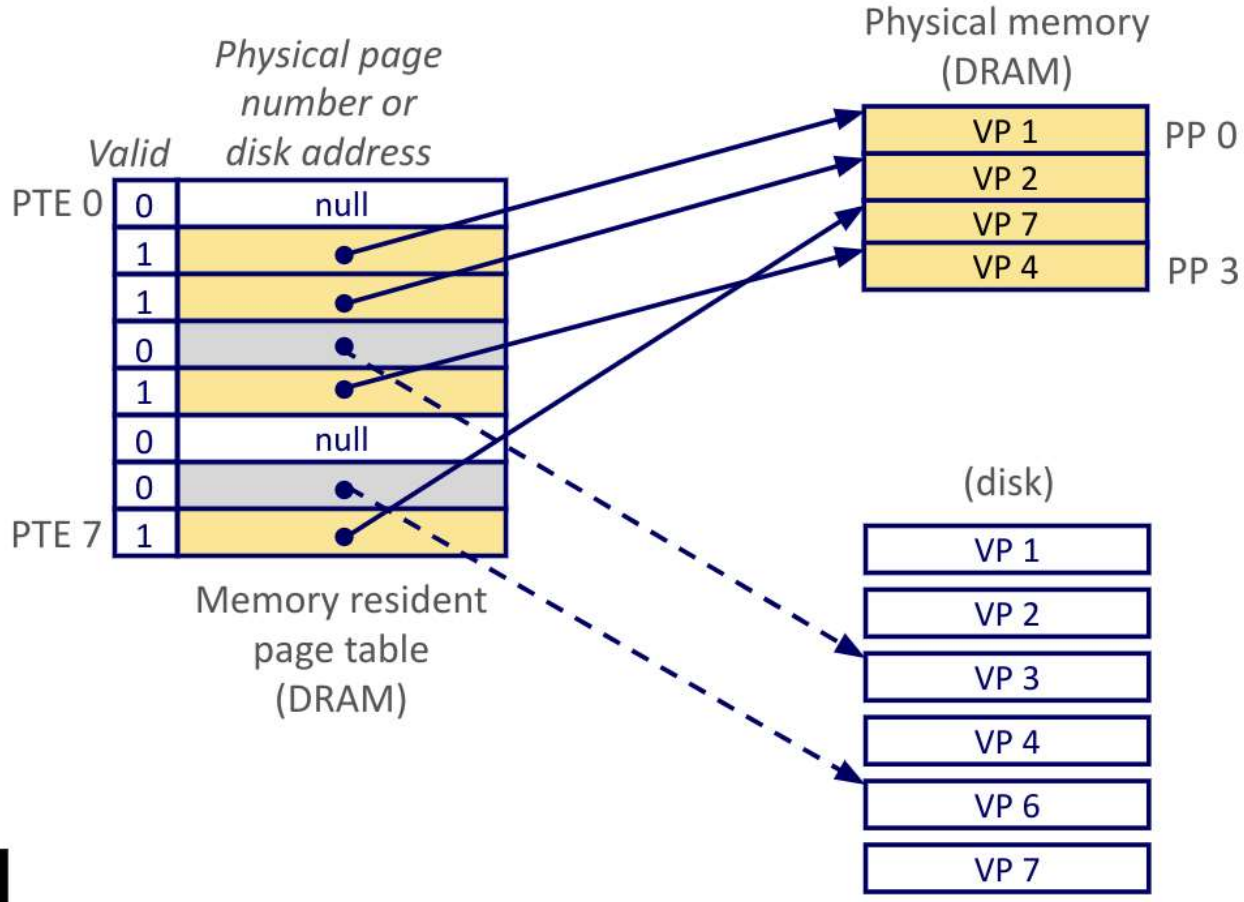
VM creates illusion (to proc) that data is always in DRAM. but in practice, there isn't space for all VM there. hence DRAM is cache

Page Hit

MMU gets address (virtual), looks up page table…

Page hit: reference to VM word that is in physical memory (DRAM cache hit)

return addr of page in DRAM



UNIVERSITY OF COPENHAGEN

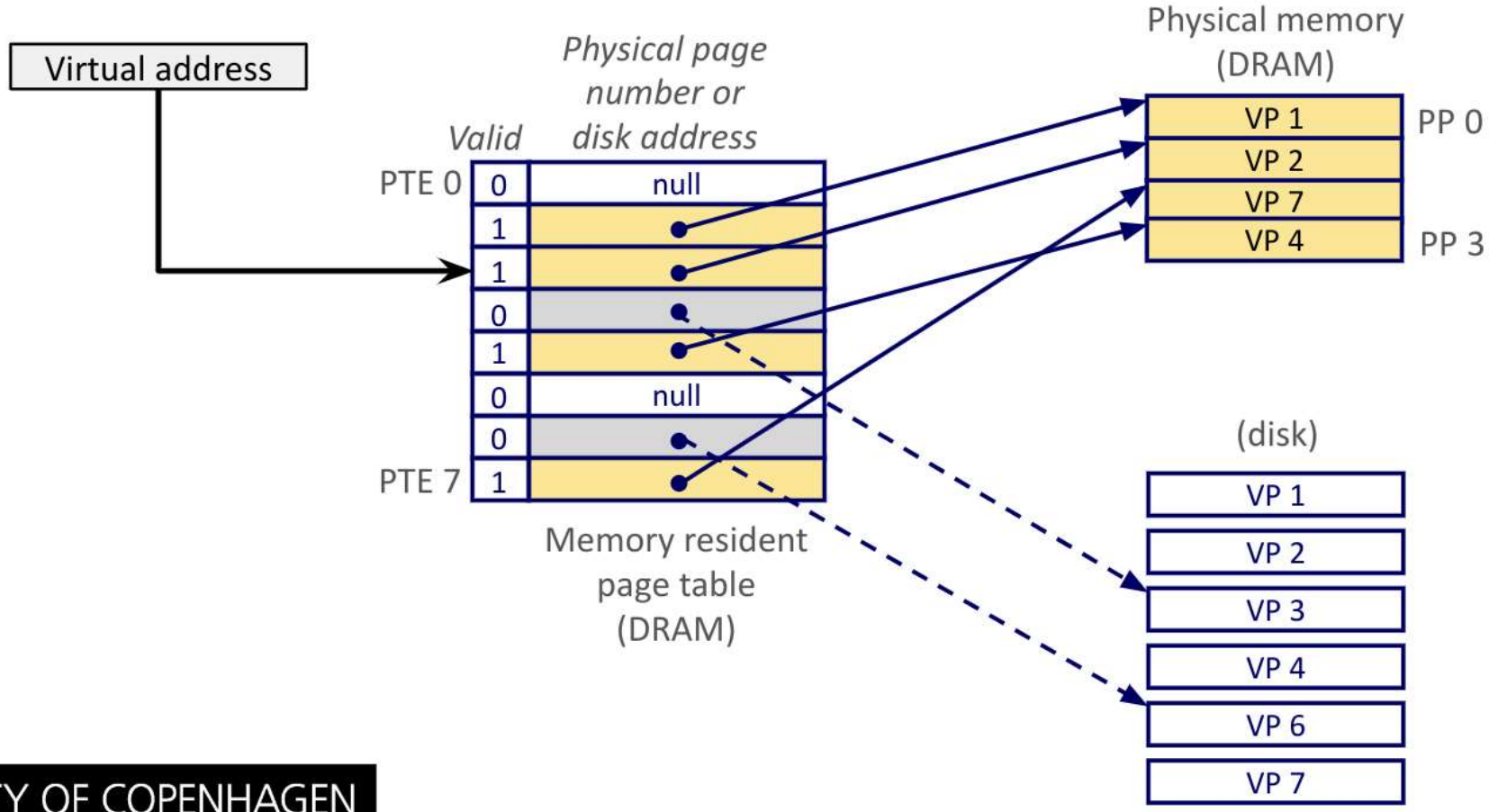
ITU CPH

Page Hit

MMU gets address (virtual), looks up page table…

Page hit: reference to VM word that is in physical memory (DRAM cache hit)

return addr of page in DRAM



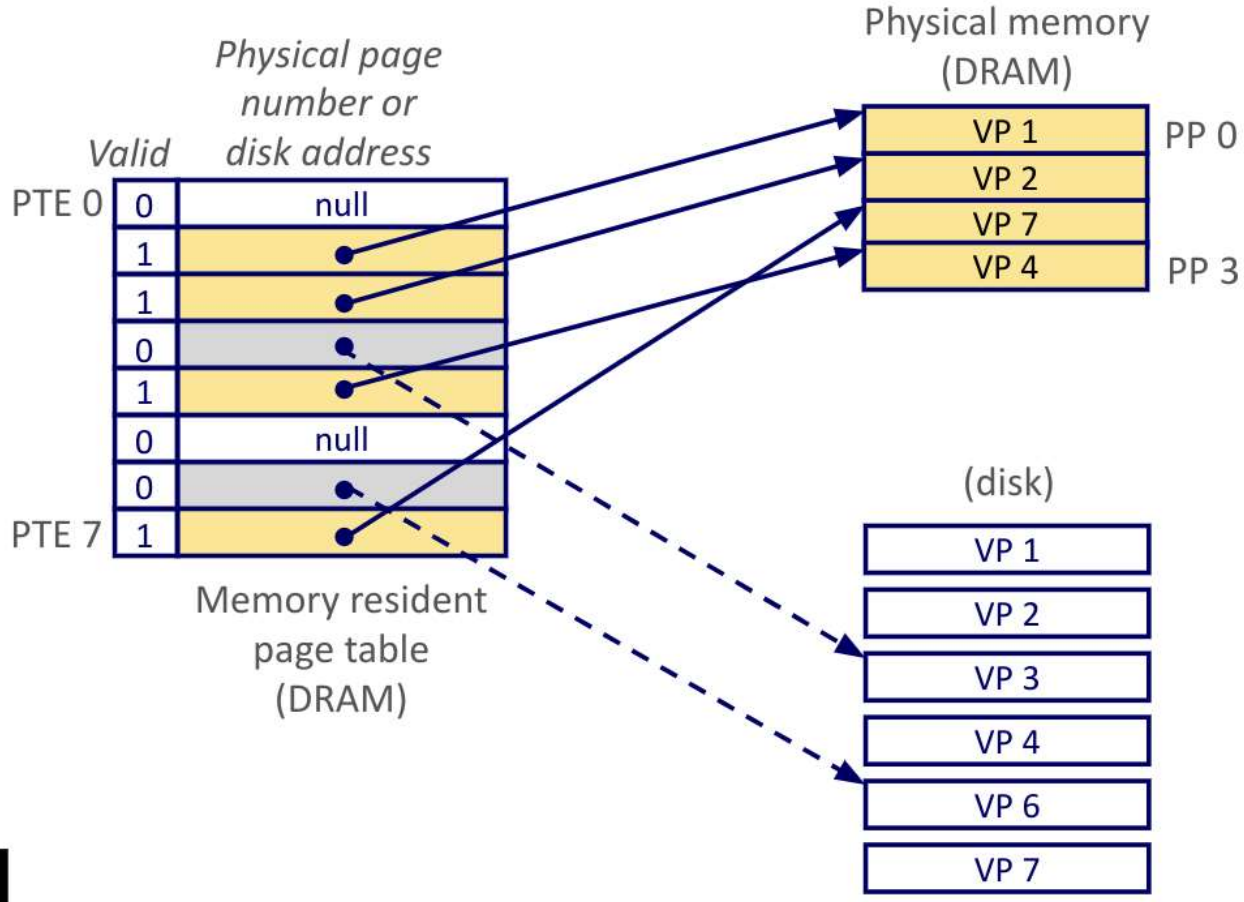
UNIVERSITY OF COPENHAGEN

ITU CPH

Page Fault

what happens on a miss?
(bring the page to the cache)
implemented w/ exceptions.

Page fault: reference to VM word that is not in physical memory (DRAM cache miss)



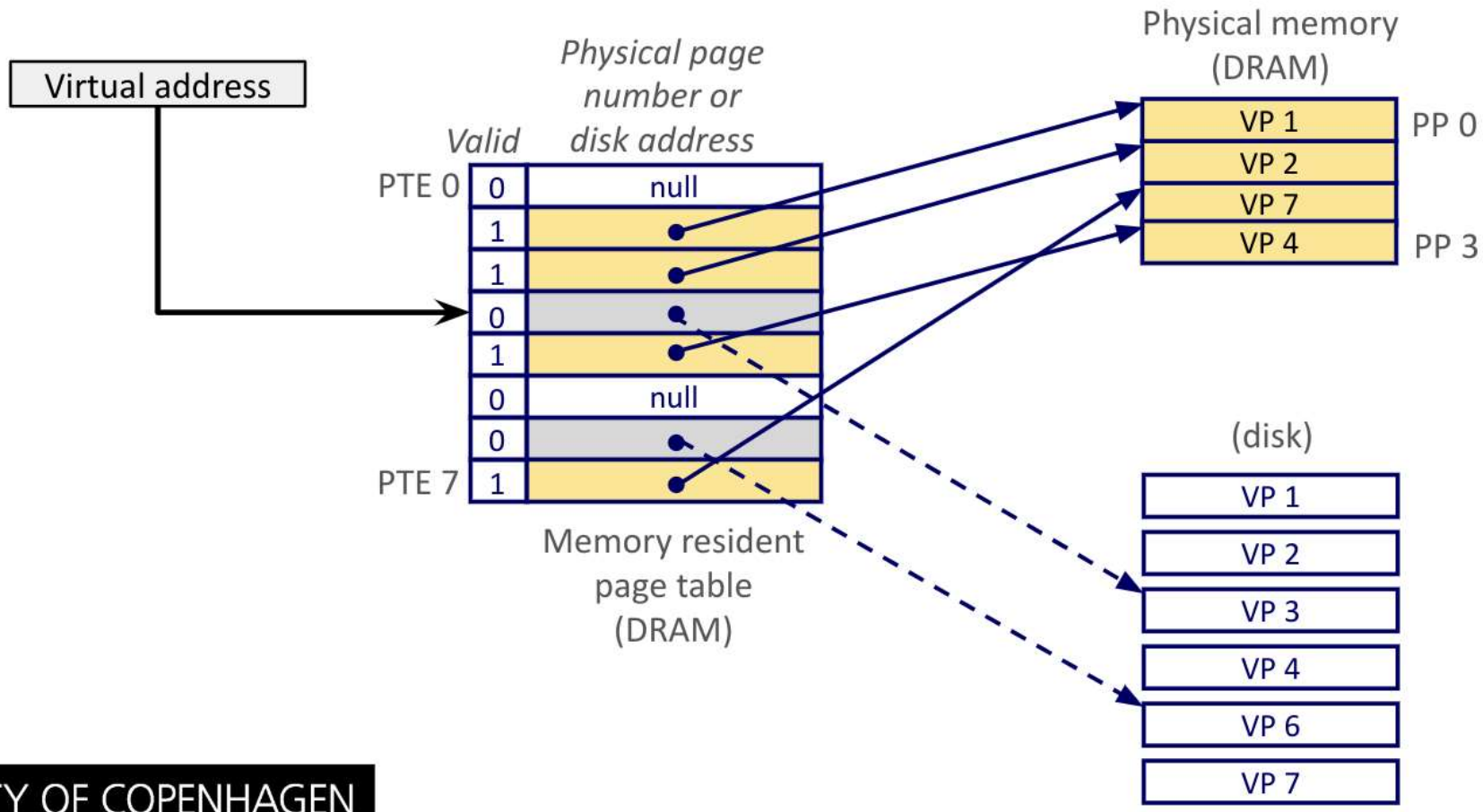
UNIVERSITY OF COPENHAGEN

ITU CPH

Page Fault

what happens on a miss?
(bring the page to the cache)
implemented w/ exceptions.

Page fault: reference to VM word that is not in physical memory (DRAM cache miss)

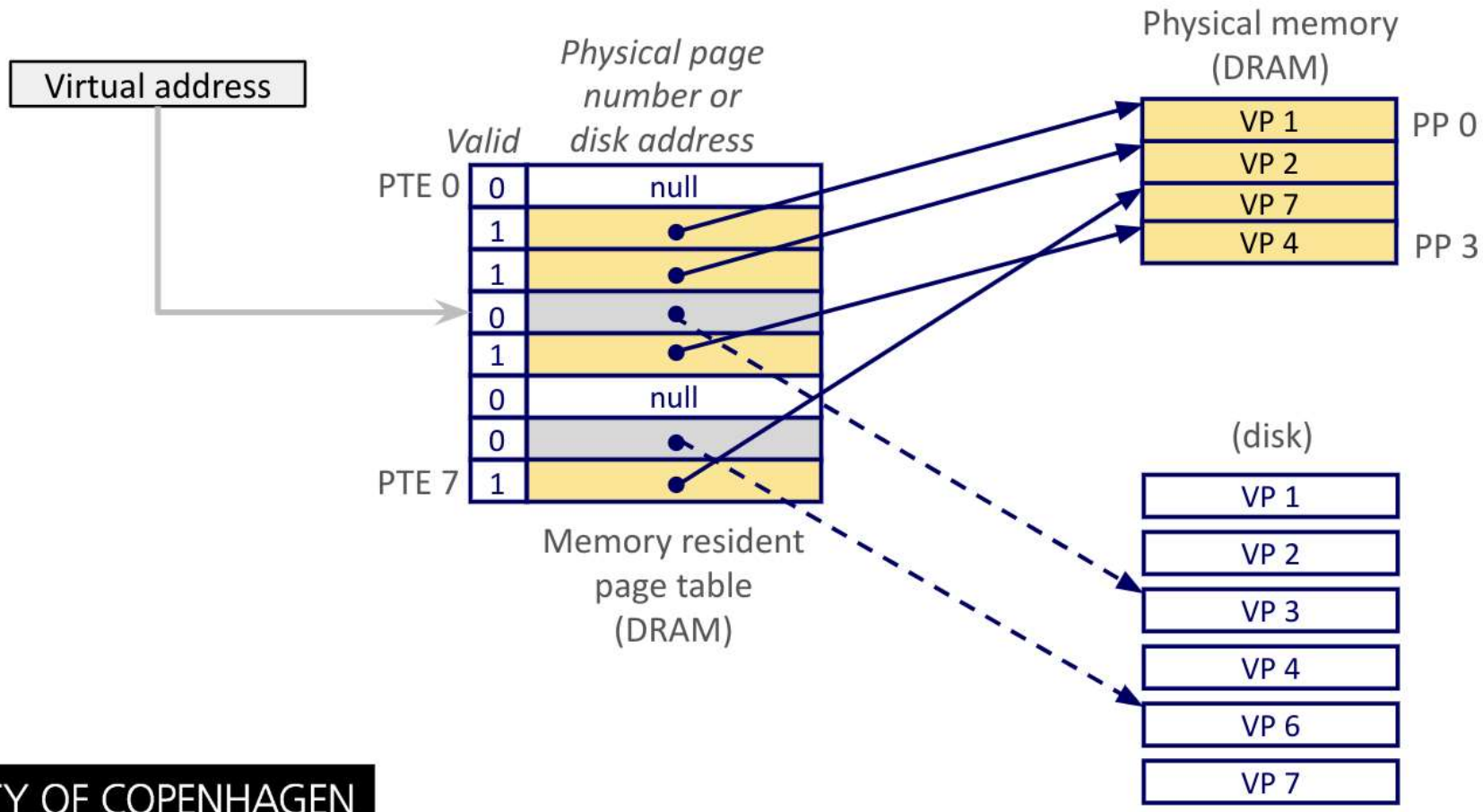


UNIVERSITY OF COPENHAGEN

ITU CPH

Handling Page Fault

Page miss causes page fault (an exception)



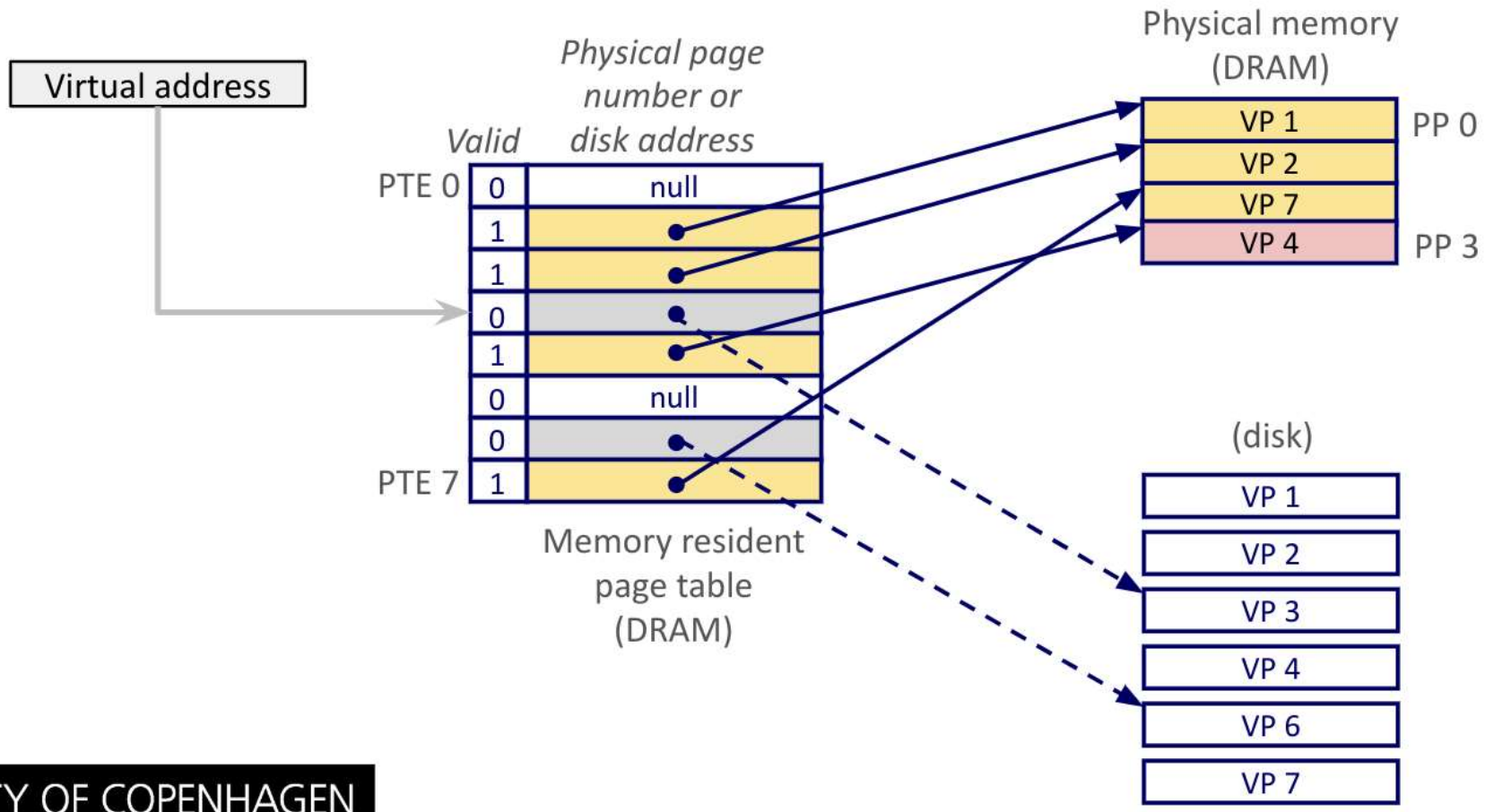
UNIVERSITY OF COPENHAGEN

ITU CPH

Handling Page Fault

(flip forth)

Page miss causes page fault (an exception)
Page fault handler selects a victim to be evicted (here VP 4)



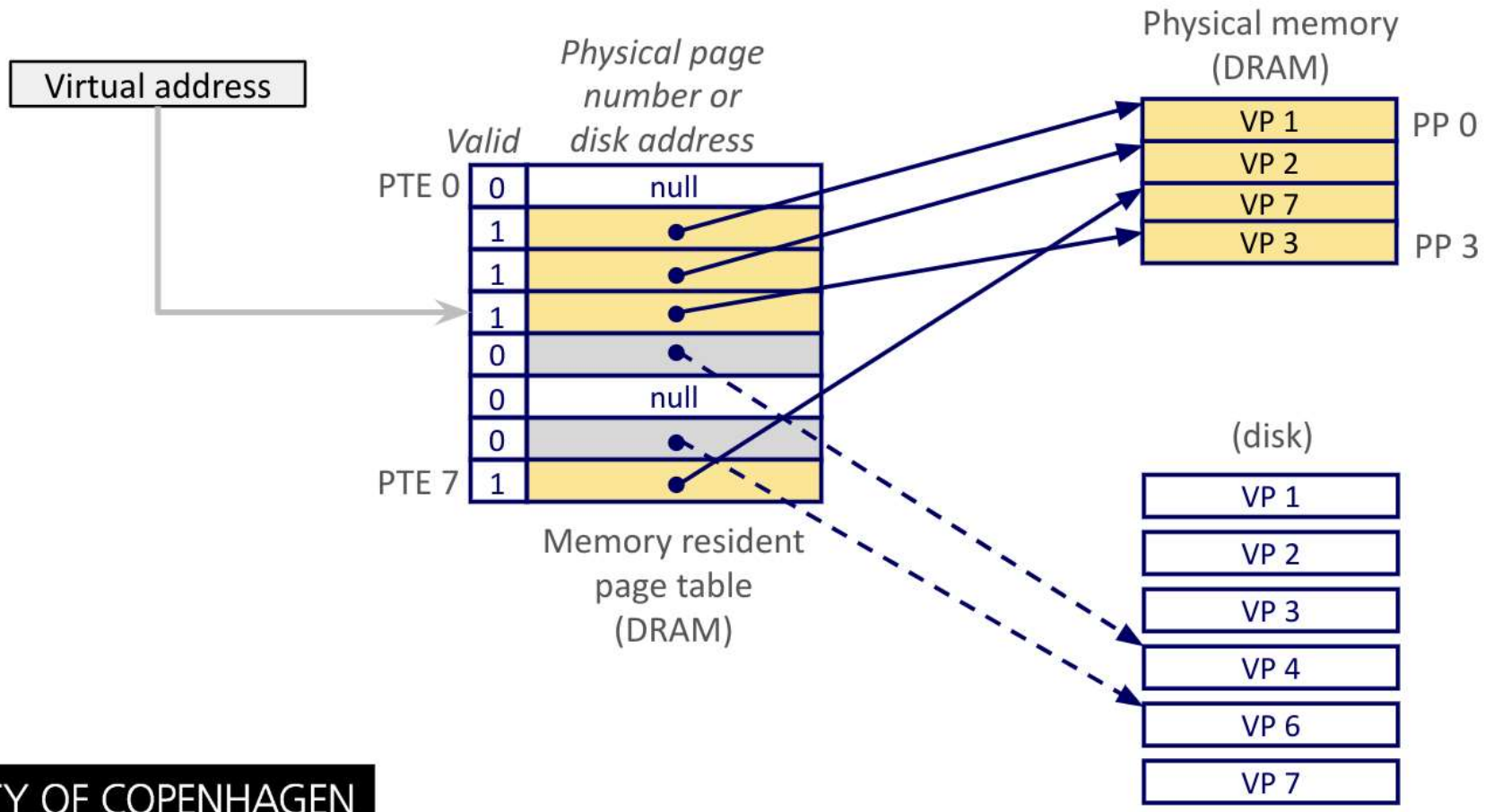
UNIVERSITY OF COPENHAGEN

ITU CPH

Handling Page Fault

(flip back)

Page miss causes page fault (an exception)
Page fault handler selects a victim to be evicted (here VP 4)



UNIVERSITY OF COPENHAGEN

ITU CPH

Handling Page Fault

this mechanism is making sure to bring data that process needs, to DRAM.

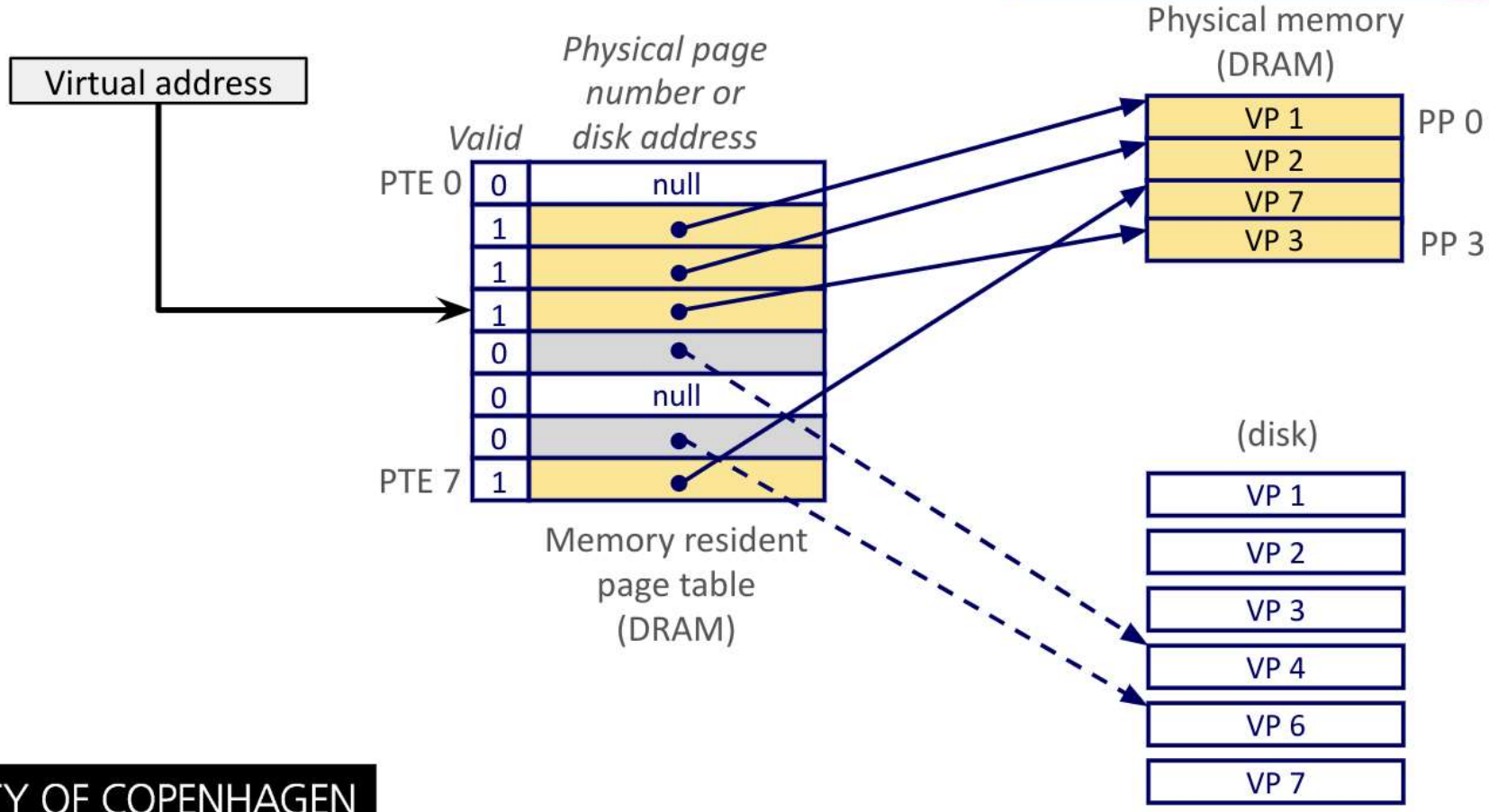
page fault is generated by hardware, bu handled by the OS

Page miss causes page fault (an exception)

Page fault handler selects a victim to be evicted (here VP 4)

Offending instruction is restarted: page hit!

(how: exception ⇒ control given to OS)



UNIVERSITY OF COPENHAGEN

ITU CPH

Locality to the Rescue Again!

Virtual memory **works** because of locality

At any point in time, programs tend to access a set of active virtual pages called the *working set*

- Programs with better temporal locality will have smaller working sets

If (working set size < main memory size)

- Good performance for one process after compulsory misses

If (SUM(working set sizes) > main memory size)

- *Thrashing:* Performance meltdown where pages are swapped (copied) in and out continuously (to/from disk; "swapping"; slooow)

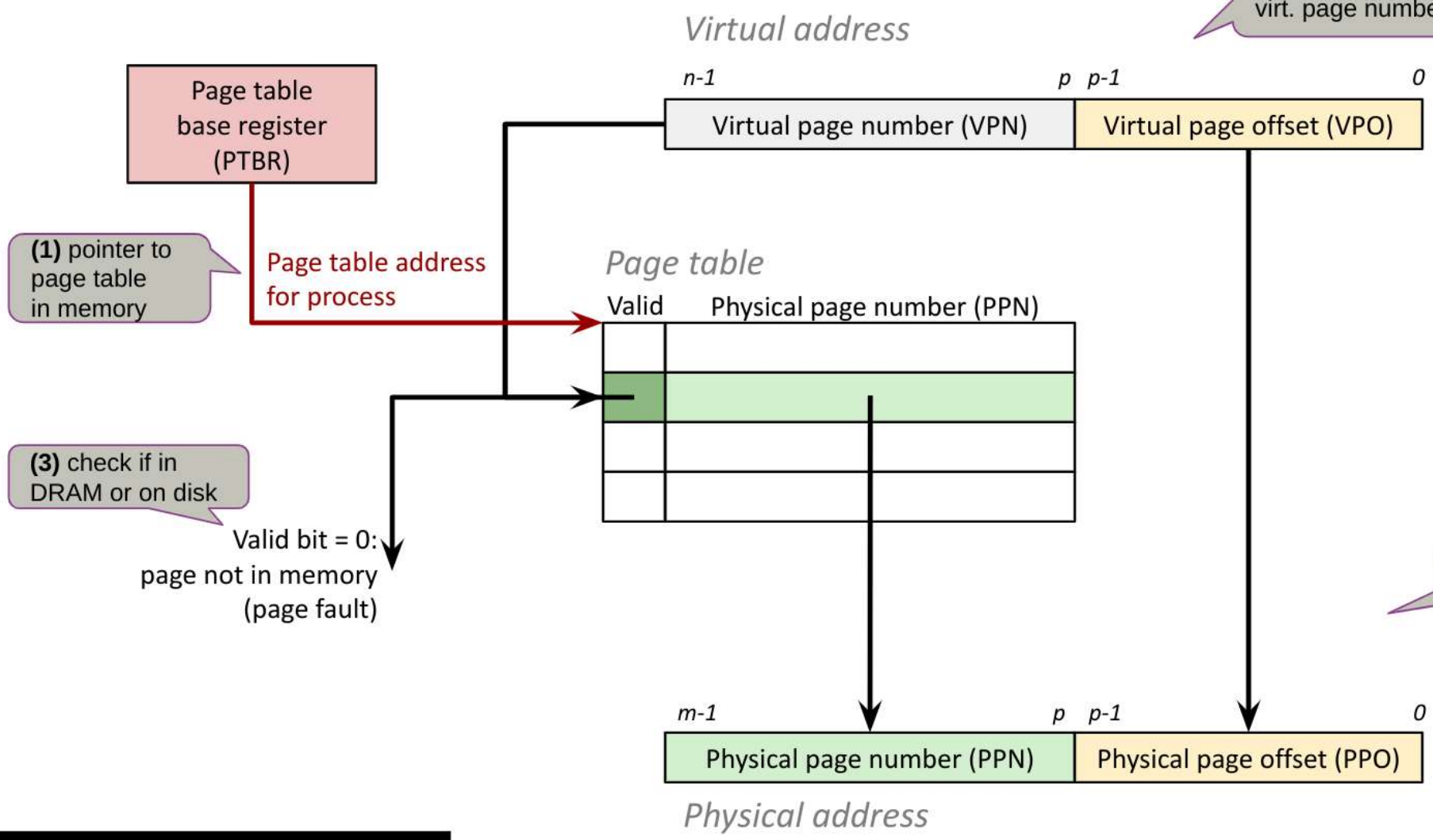
you can literally hear this in mechanical hard drives (a "grinding" sound)

UNIVERSITY OF COPENHAGEN

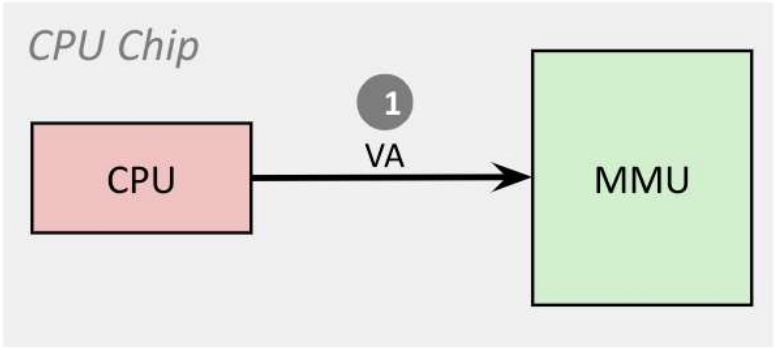
ITU CPH

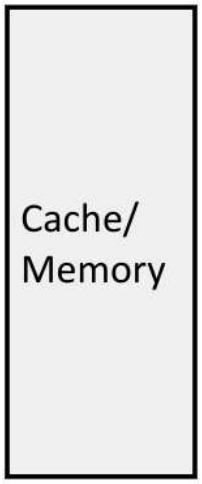
Address Translation With a Page Table

example now where we go through the whole translation.



Address Translation: Page Hit





1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to cache/memory

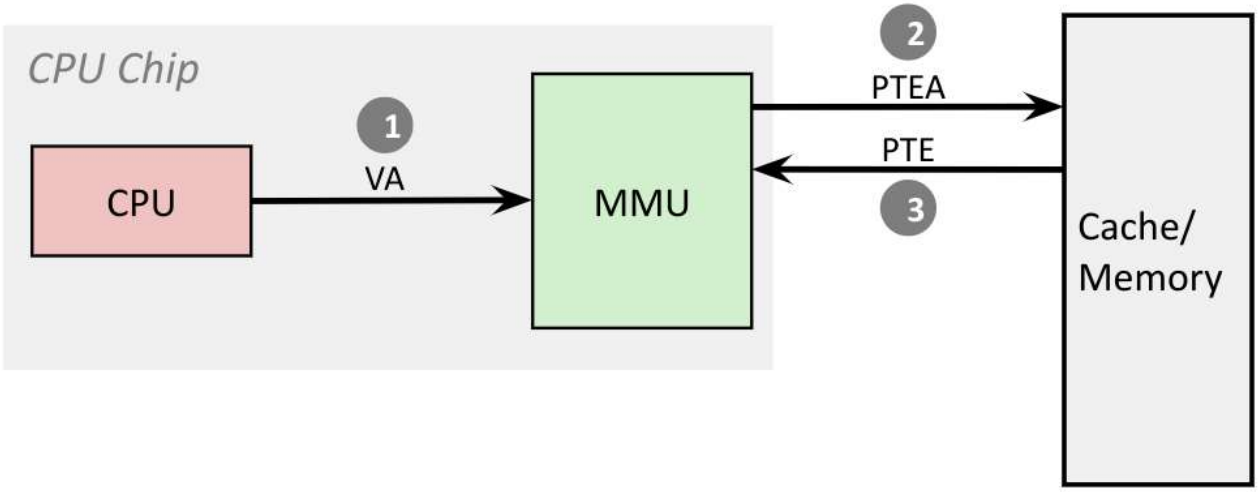
5) Cache/memory sends data word to processor

very flexible mechanism!
but, if we are not careful,
then we have 2 accesses
to DRAM for each VA.
(how to optimize that?
hold that thought)

UNIVERSITY OF COPENHAGEN

ITU CPH

Address Translation: Page Hit



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to cache/memory

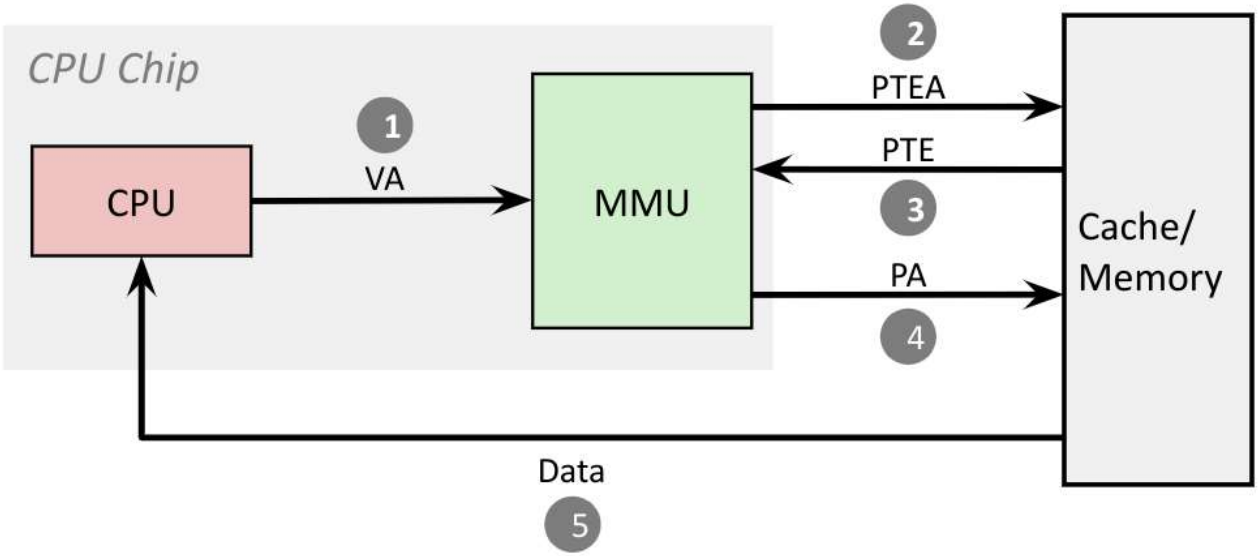
5) Cache/memory sends data word to processor

very flexible mechanism!
but, if we are not careful,
then we have 2 accesses
to DRAM for each VA.
(how to optimize that?
hold that thought)

UNIVERSITY OF COPENHAGEN

ITU CPH

Address Translation: Page Hit



1) Processor sends virtual address to MMU

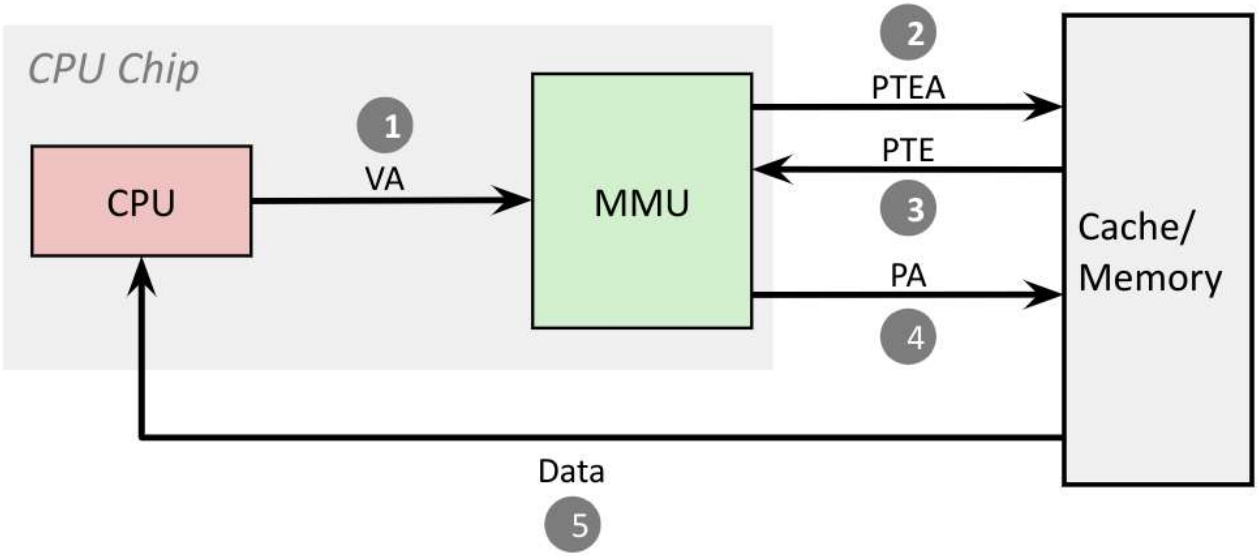
2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

very flexible mechanism! but, if we are not careful, then we have 2 accesses to DRAM for each VA. (how to optimize that? hold that thought)

Address Translation: Page Hit



1) Processor sends virtual address to MMU

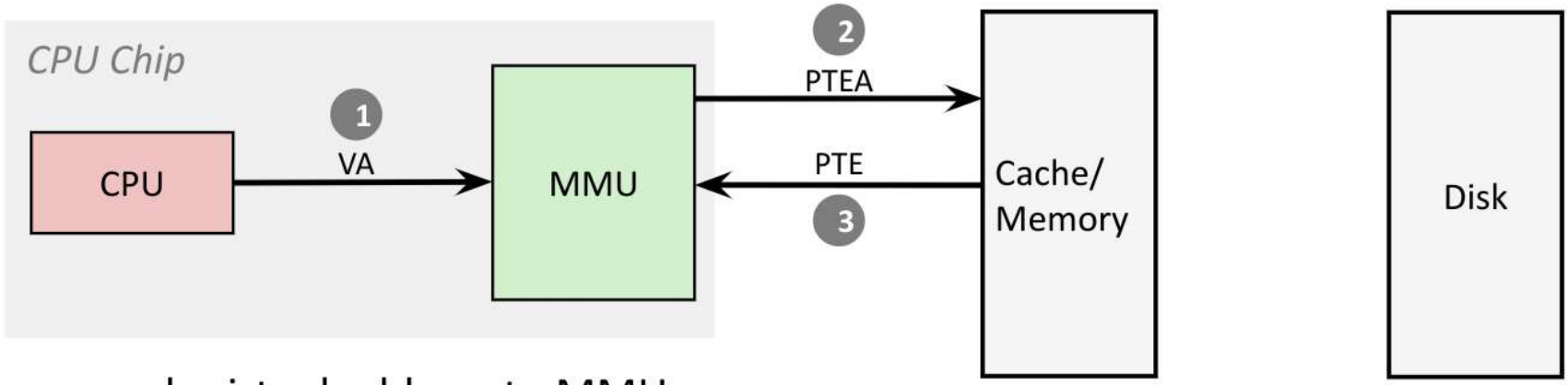
2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

very flexible mechanism! but, if we are not careful, then we have 2 accesses to DRAM for each VA. (how to optimize that? hold that thought)

Address Translation: Page Fault



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) Valid bit is zero, so MMU triggers page fault exception (page not in DRAM)

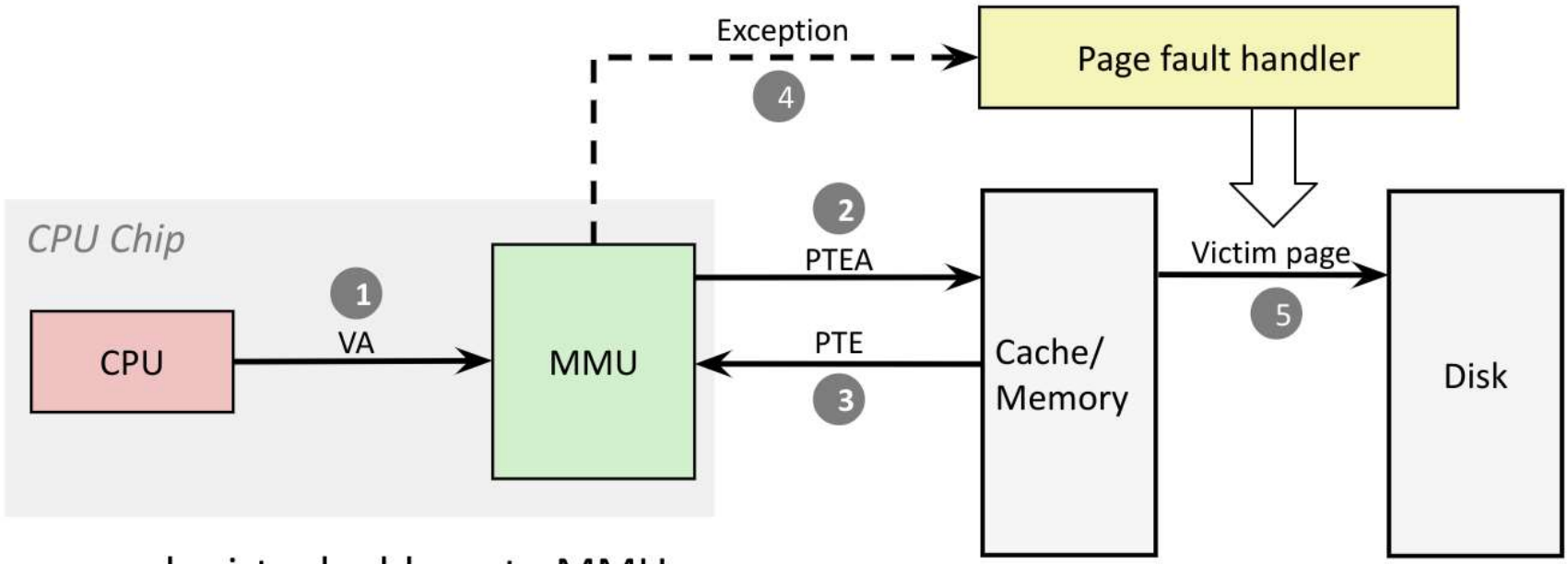
5) Handler identifies victim (and, if dirty, pages it out to disk)

6) Handler pages in new page and updates PTE in memory

7) Handler returns to original process, restarting faulting instruction

costly operation

Address Translation: Page Fault



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) Valid bit is zero, so MMU triggers page fault exception (page not in DRAM)

5) Handler identifies victim (and, if dirty, pages it out to disk)

6) Handler pages in new page and updates PTE in memory

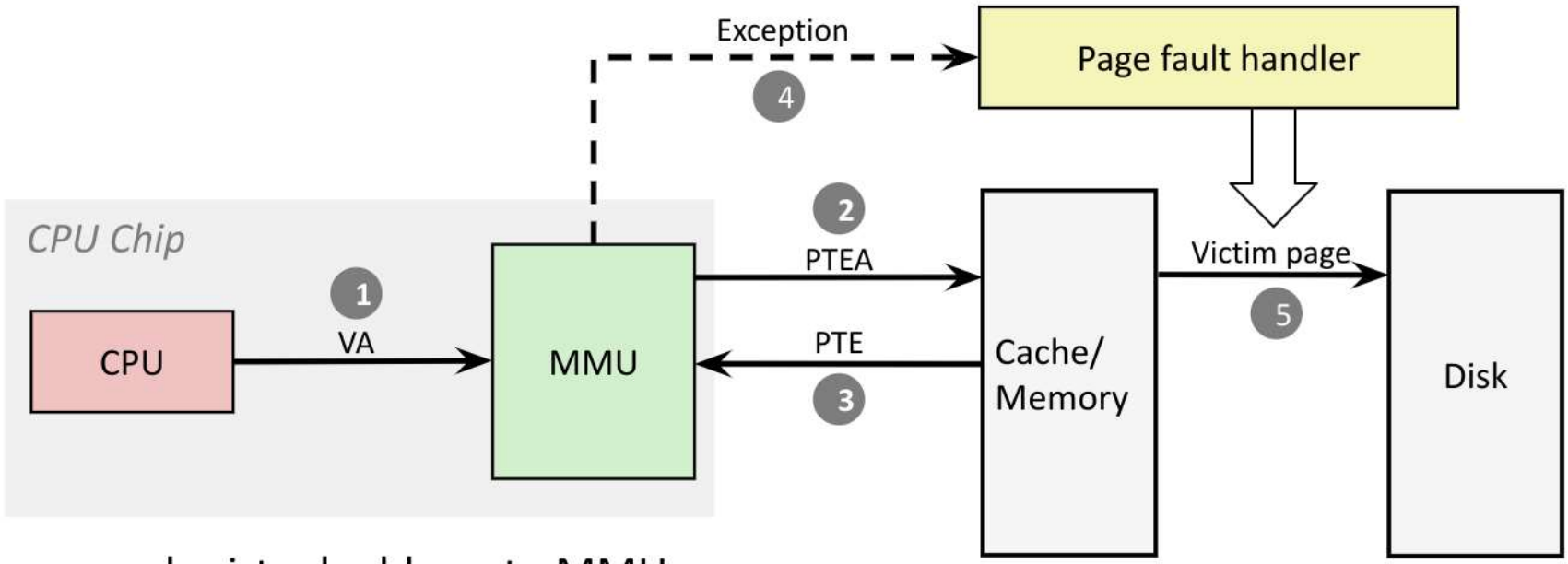
7) Handler returns to original process, restarting faulting instruction

UNIVERSITY OF COPENHAGEN

costly operation

ITU CPH

Address Translation: Page Fault



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) Valid bit is zero, so MMU triggers page fault exception (page not in DRAM)

5) Handler identifies victim (and, if dirty, pages it out to disk)

6) Handler pages in new page and updates PTE in memory

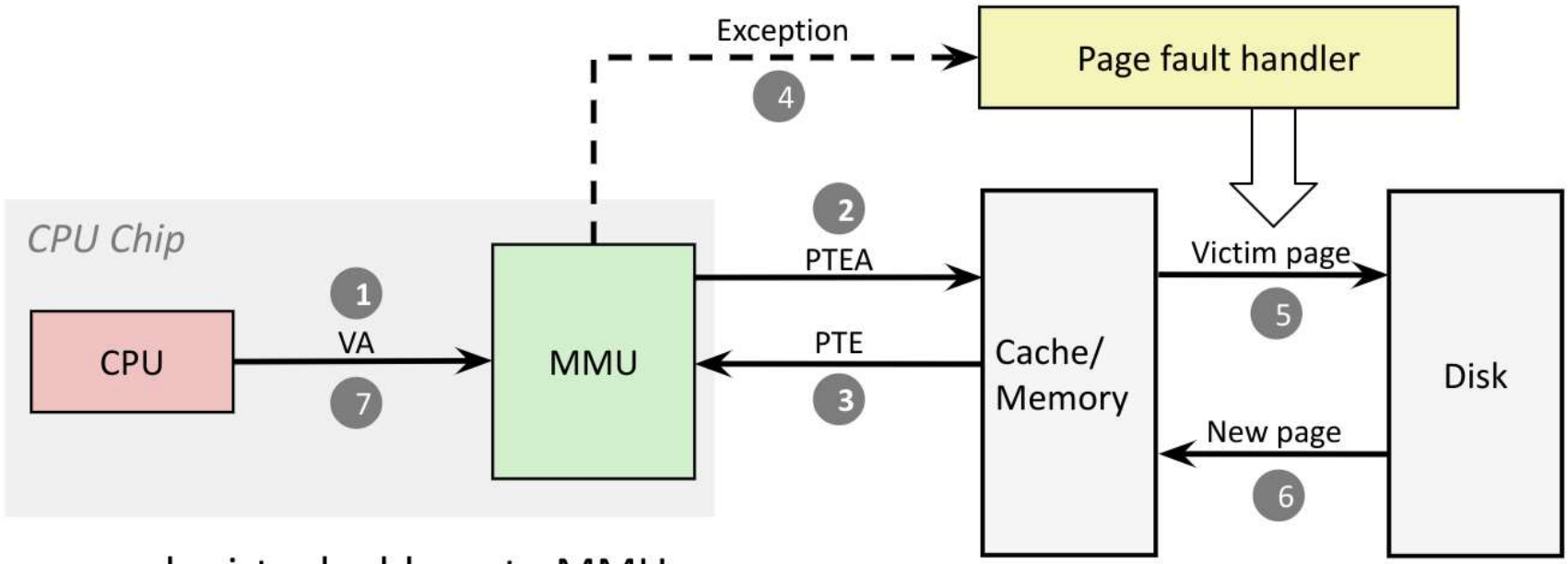
7) Handler returns to original process, restarting faulting instruction

UNIVERSITY OF COPENHAGEN

costly operation

ITU CPH

Address Translation: Page Fault



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) Valid bit is zero, so MMU triggers page fault exception (page not in DRAM)

5) Handler identifies victim (and, if dirty, pages it out to disk)

6) Handler pages in new page and updates PTE in memory

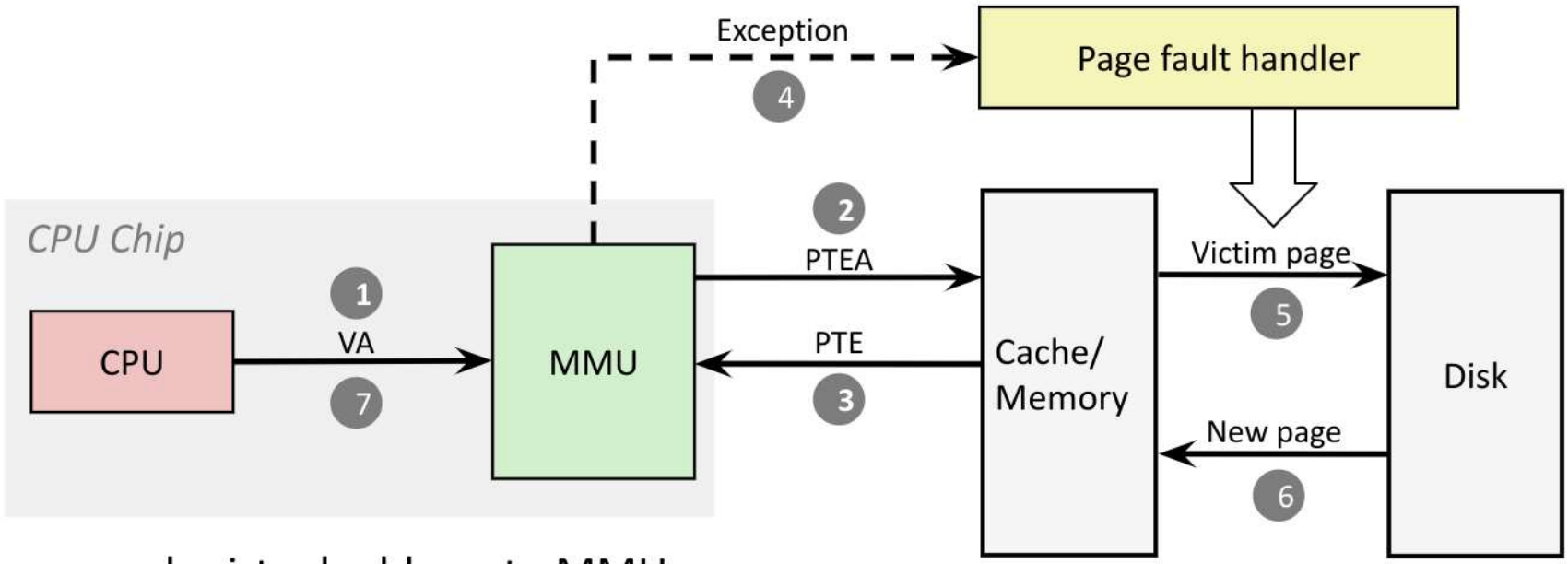
7) Handler returns to original process, restarting faulting instruction

UNIVERSITY OF COPENHAGEN

costly operation

ITU CPH

Address Translation: Page Fault



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) Valid bit is zero, so MMU triggers page fault exception (page not in DRAM)

5) Handler identifies victim (and, if dirty, pages it out to disk)

6) Handler pages in new page and updates PTE in memory

7) Handler returns to original process, restarting faulting instruction

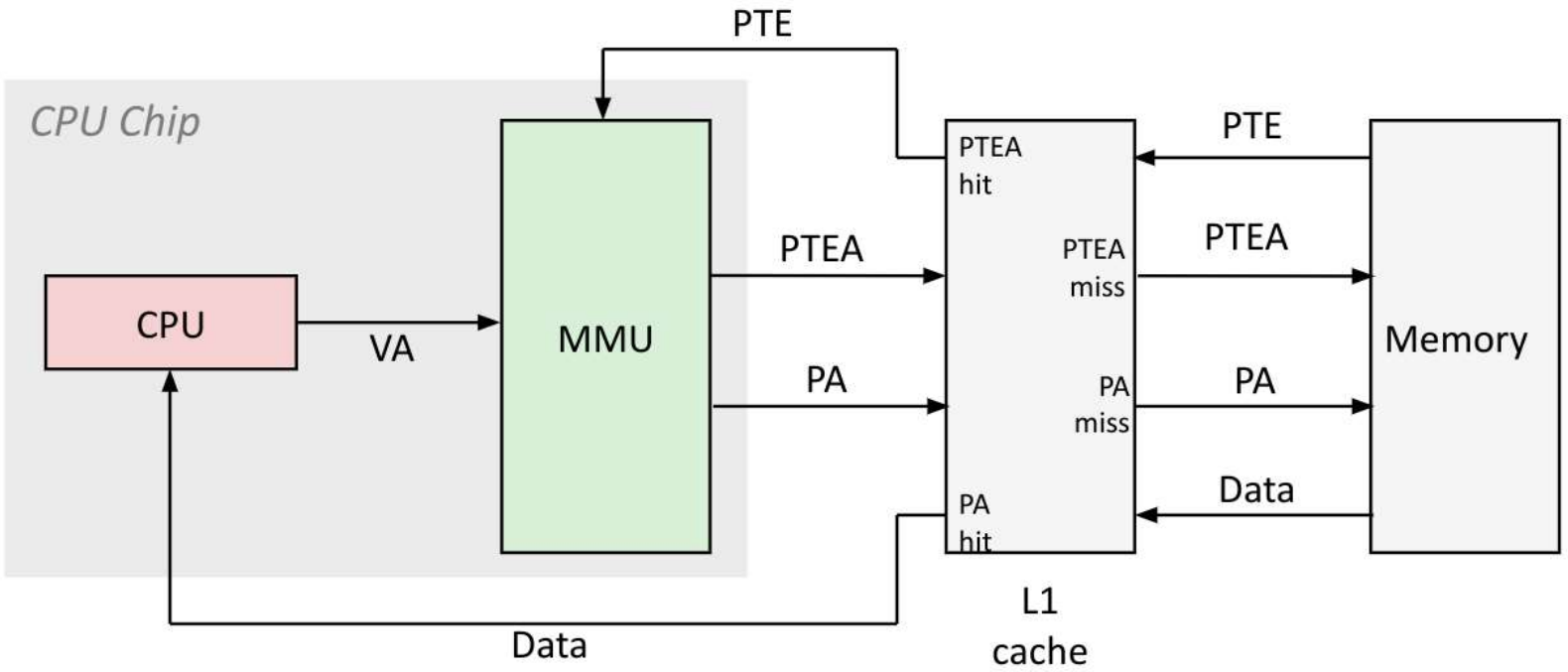
UNIVERSITY OF COPENHAGEN

costly operation

ITU CPH

Integrating VM and Cache

reality is different 😈
between CPU and DRAM,
we have caches.



VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

the further from CPU,
the more CPU must wait
(nanoseconds to 100s of micro

Speeding up Translation with a TLB

Page table entries (PTEs) are cached in L1 like any other memory word

- PTEs may be evicted by other data references
- PTE hit still requires a small L1 delay.

we don't want to pollute the L1 cache w/ PTEs

Solution:

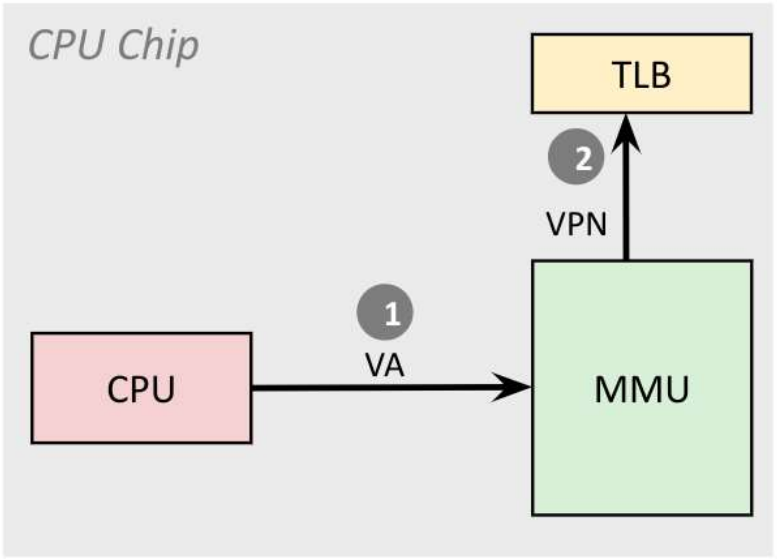
Speeding up Translation with a TLB

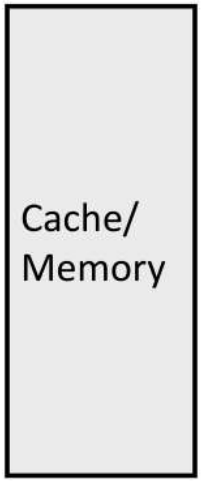
- Page table entries (PTEs) are cached in L1 like any other memory word
  - PTEs may be evicted by other data references
  - PTE hit still requires a small L1 delay.

we don't want to pollute the L1 cache w/ PTEs

- Solution: *Translation Lookaside Buffer* (TLB)
  - Small hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages

TLB Hit





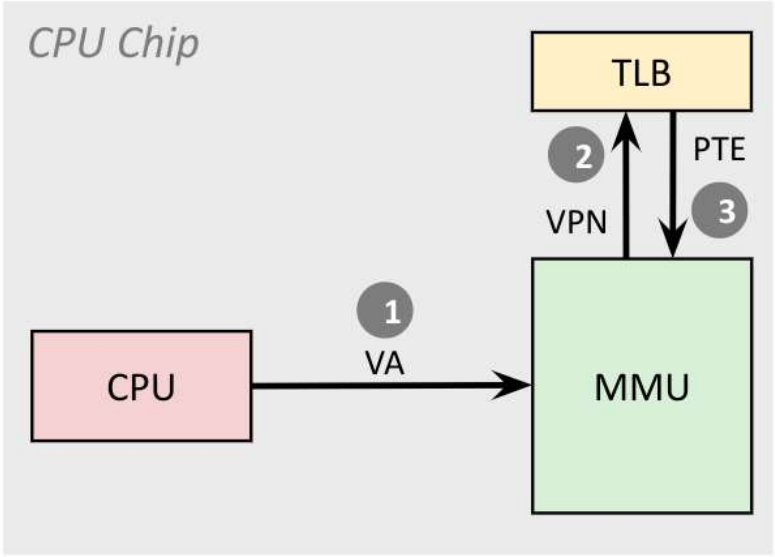
A TLB hit eliminates a memory access

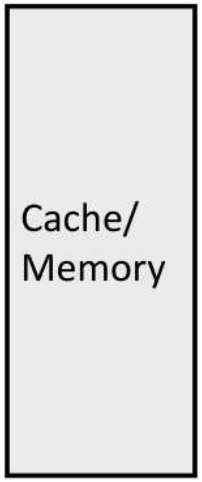
UNIVERSITY OF COPENHAGEN

nice and fast :-)

ITU CPH

TLB Hit





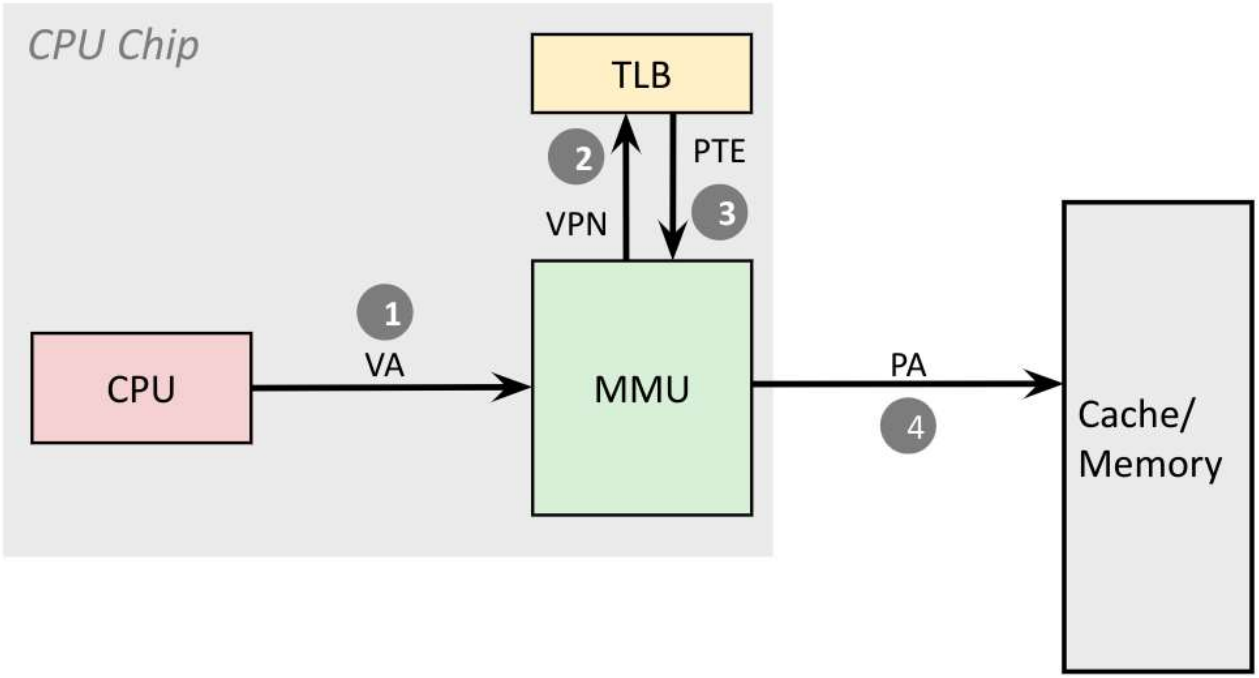
A TLB hit eliminates a memory access

UNIVERSITY OF COPENHAGEN

nice and fast :-)

ITU CPH

TLB Hit



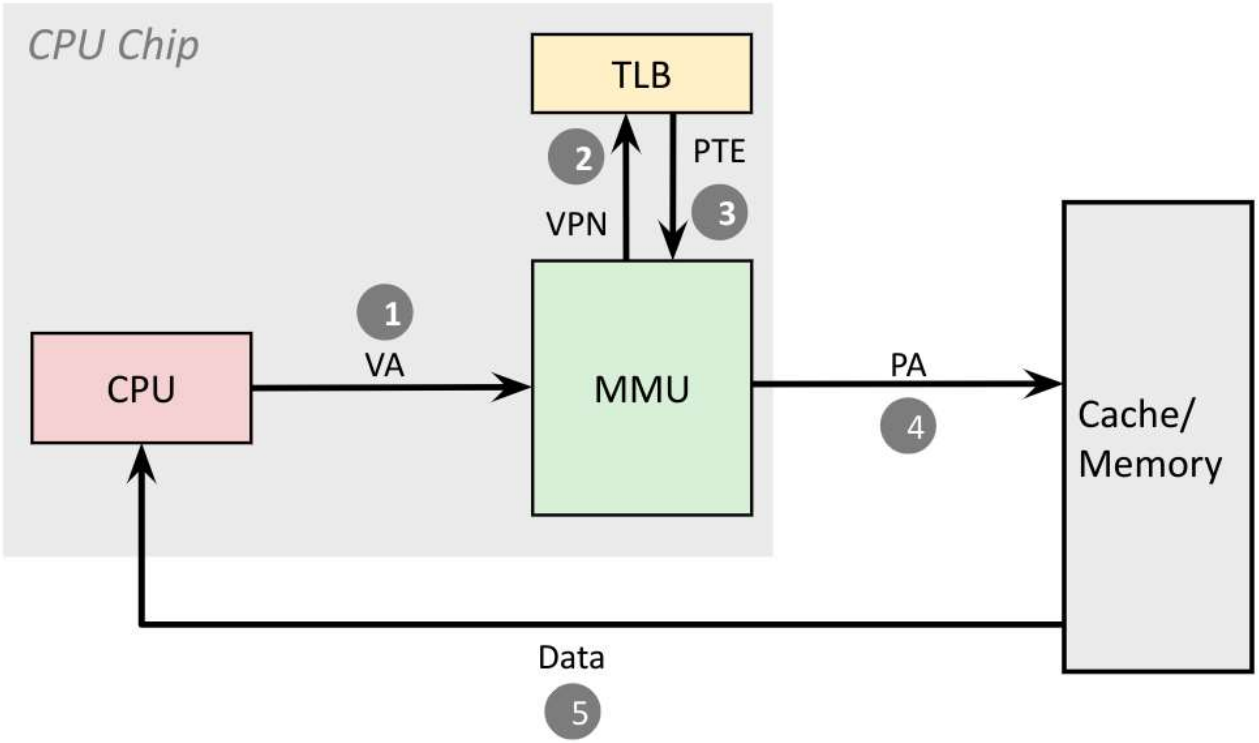
A TLB hit eliminates a memory access

UNIVERSITY OF COPENHAGEN

nice and fast :-)

ITU CPH

TLB Hit



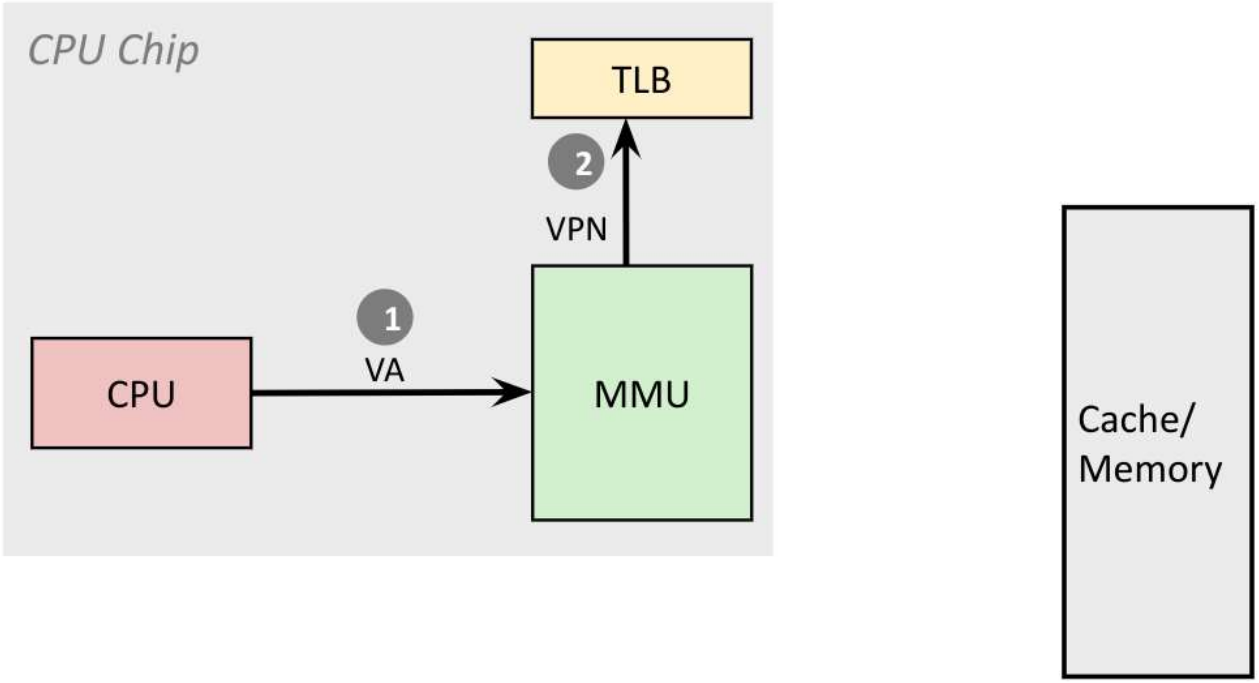
A TLB hit eliminates a memory access

UNIVERSITY OF COPENHAGEN

nice and fast :-)

ITU CPH

TLB Miss



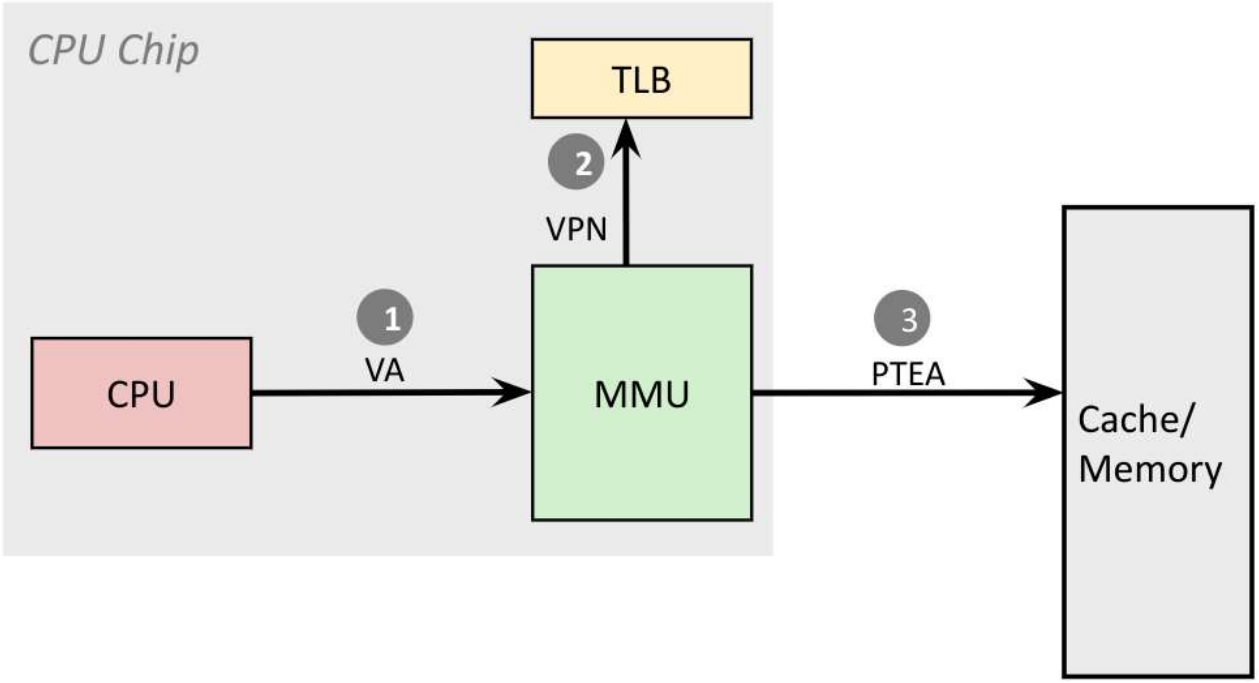
A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare. Why?

UNIVERSITY OF COPENHAGEN

ITU CPH

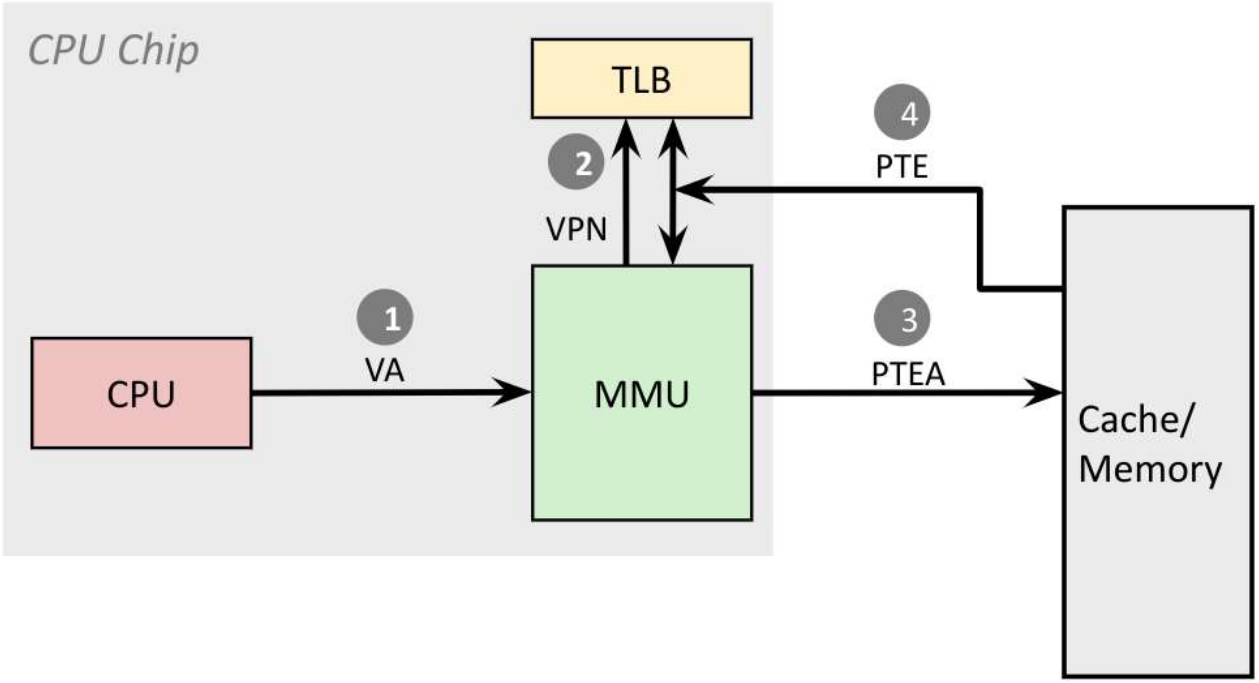
TLB Miss



A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare. Why?

TLB Miss



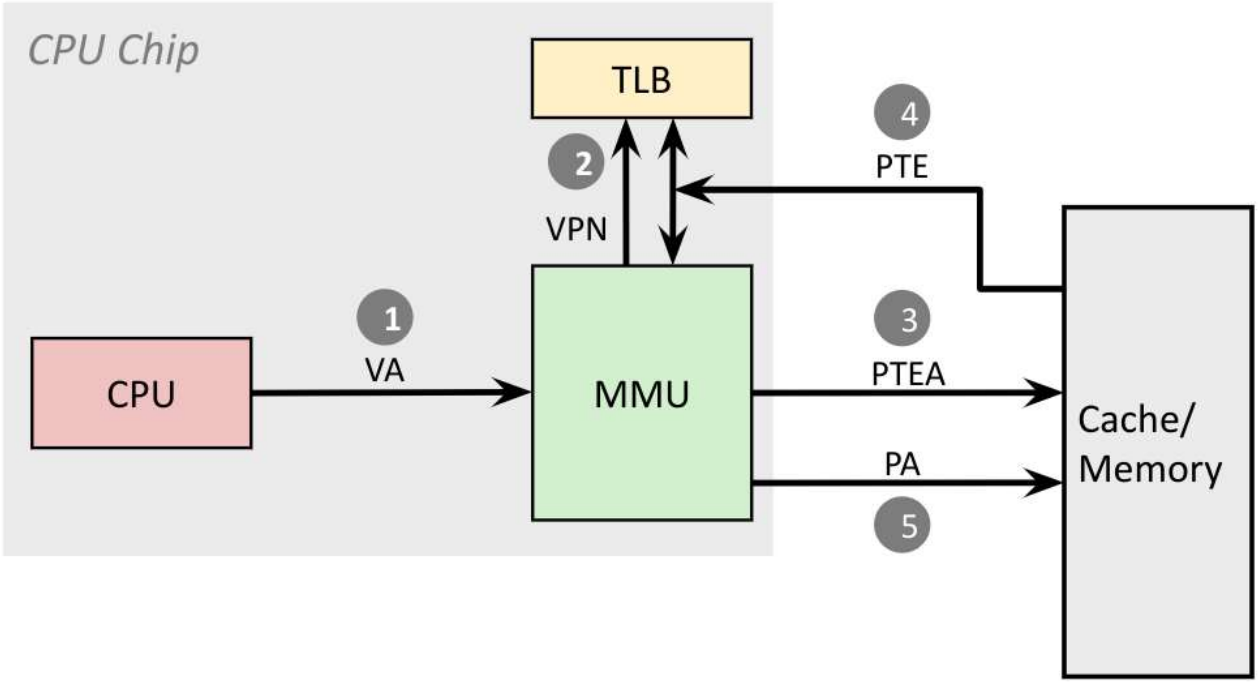
A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare. Why?

UNIVERSITY OF COPENHAGEN

ITU CPH

TLB Miss



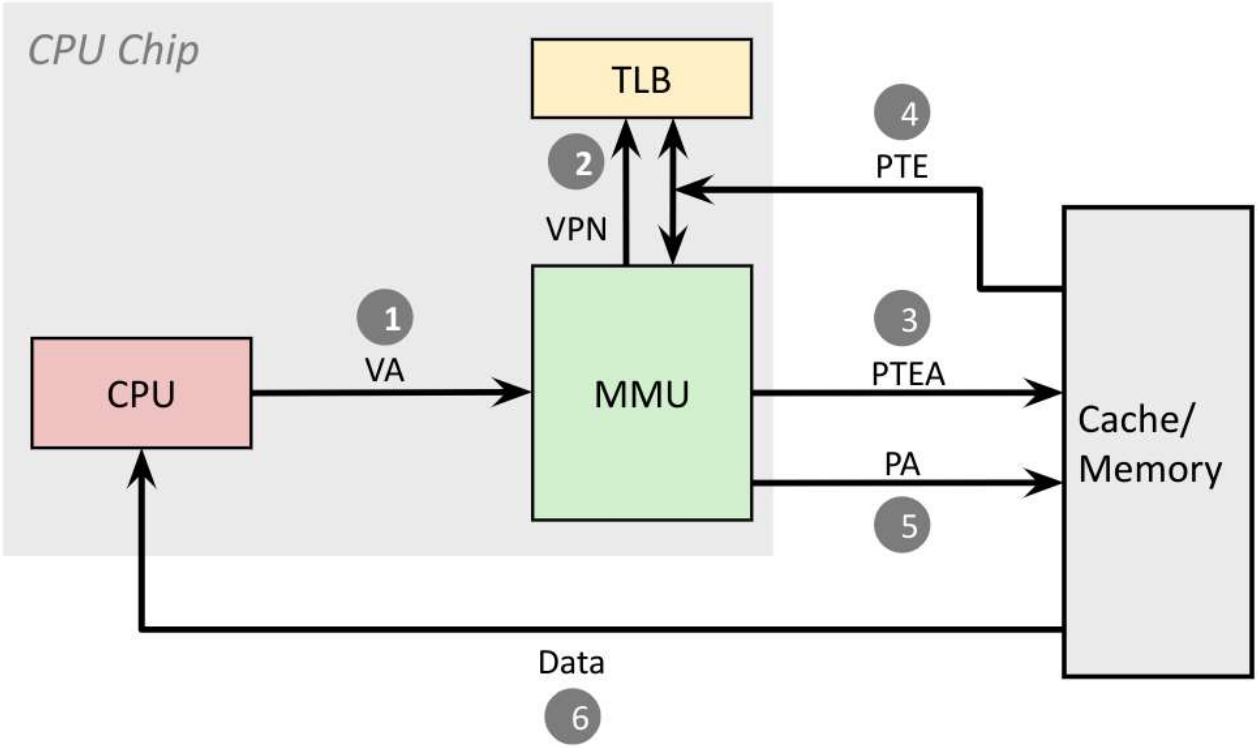
A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare. Why?

UNIVERSITY OF COPENHAGEN

ITU CPH

TLB Miss



A TLB miss incurs an additional memory access (the PTE)

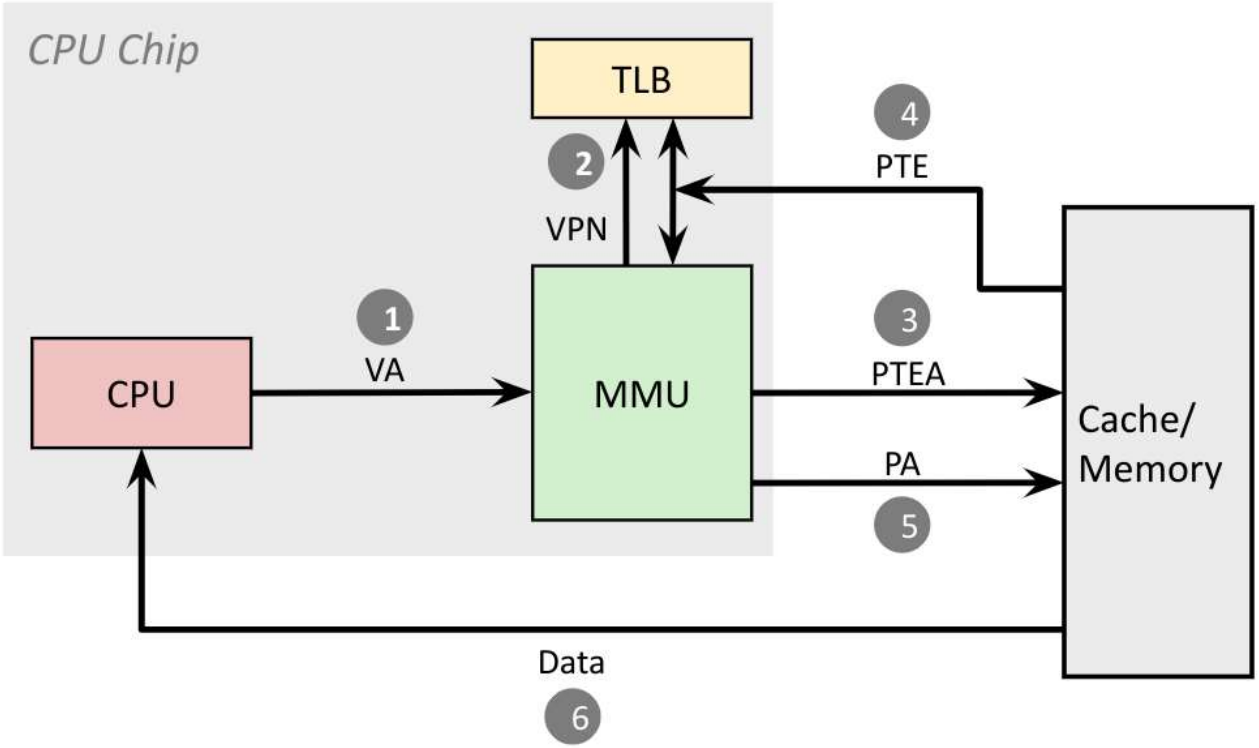
Fortunately, TLB misses are rare. Why?

UNIVERSITY OF COPENHAGEN

ITU CPH

TLB Miss

a way to quantify performance of your program is to monitor nr. of TLB misses. add locality to reduce



A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare. Why?

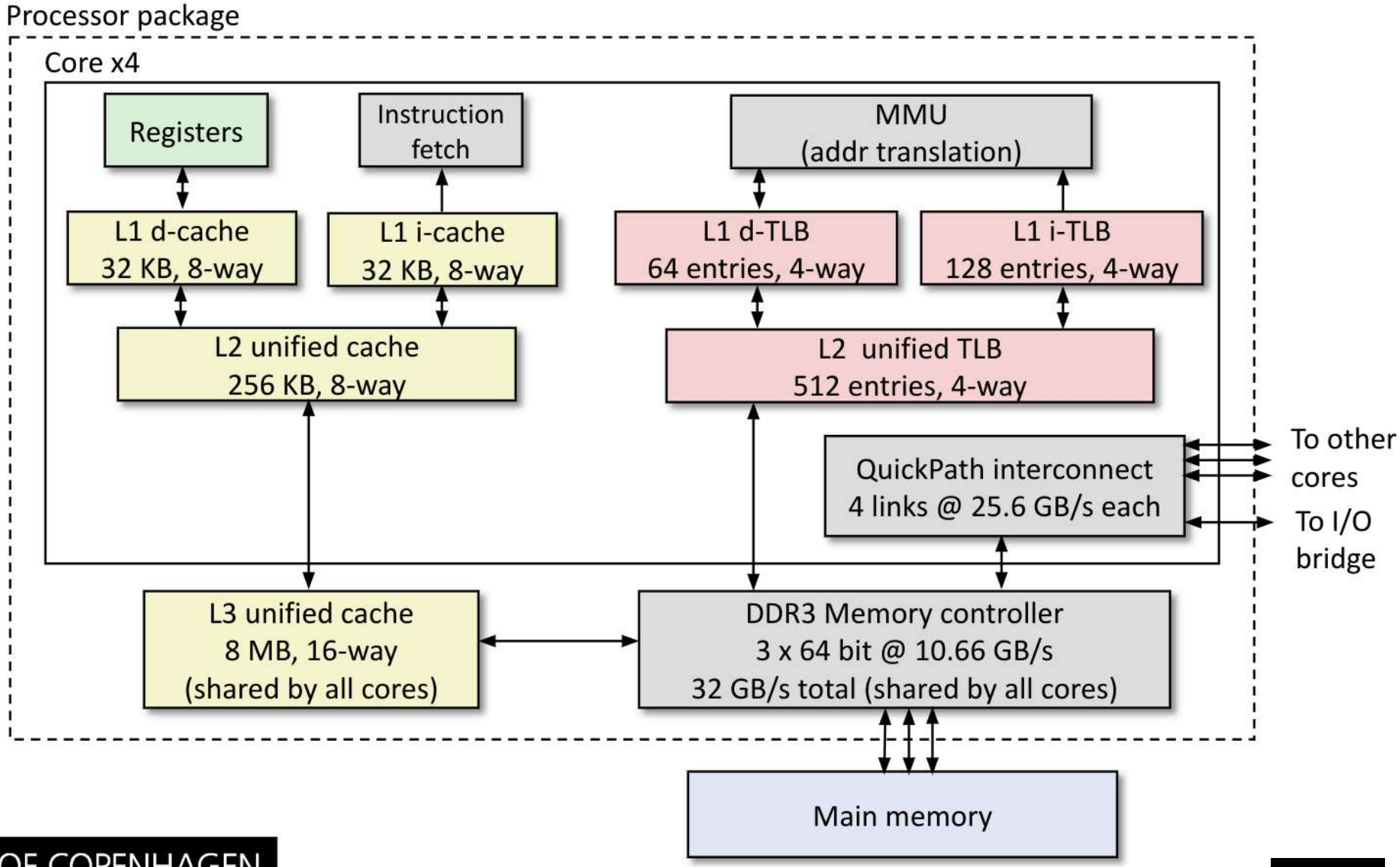
UNIVERSITY OF COPENHAGEN

TLB misses are rare! why

ITU CPH

Intel Core i7 Memory System

concrete example (20s)



UNIVERSITY OF COPENHAGEN

ITU CPH

Take-Aways

You should be able to describe:

- Address Space
- Physical and Virtual Memory
- MMU (its role, how work split between it and OS)
- Pages
- Page Table
- Translation Lookaside Buffer (TLB)

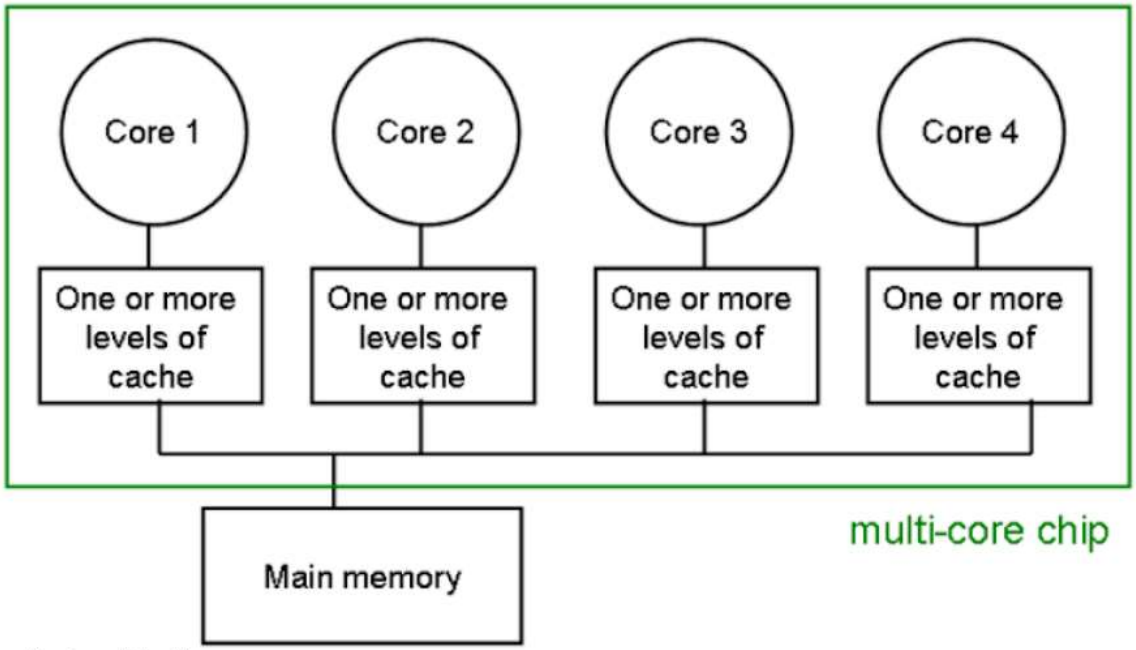
Virtual Memory as a tool for caching, memory management and memory protection.

Cache Coherence

ITU CPH

The Cache Coherence Problem

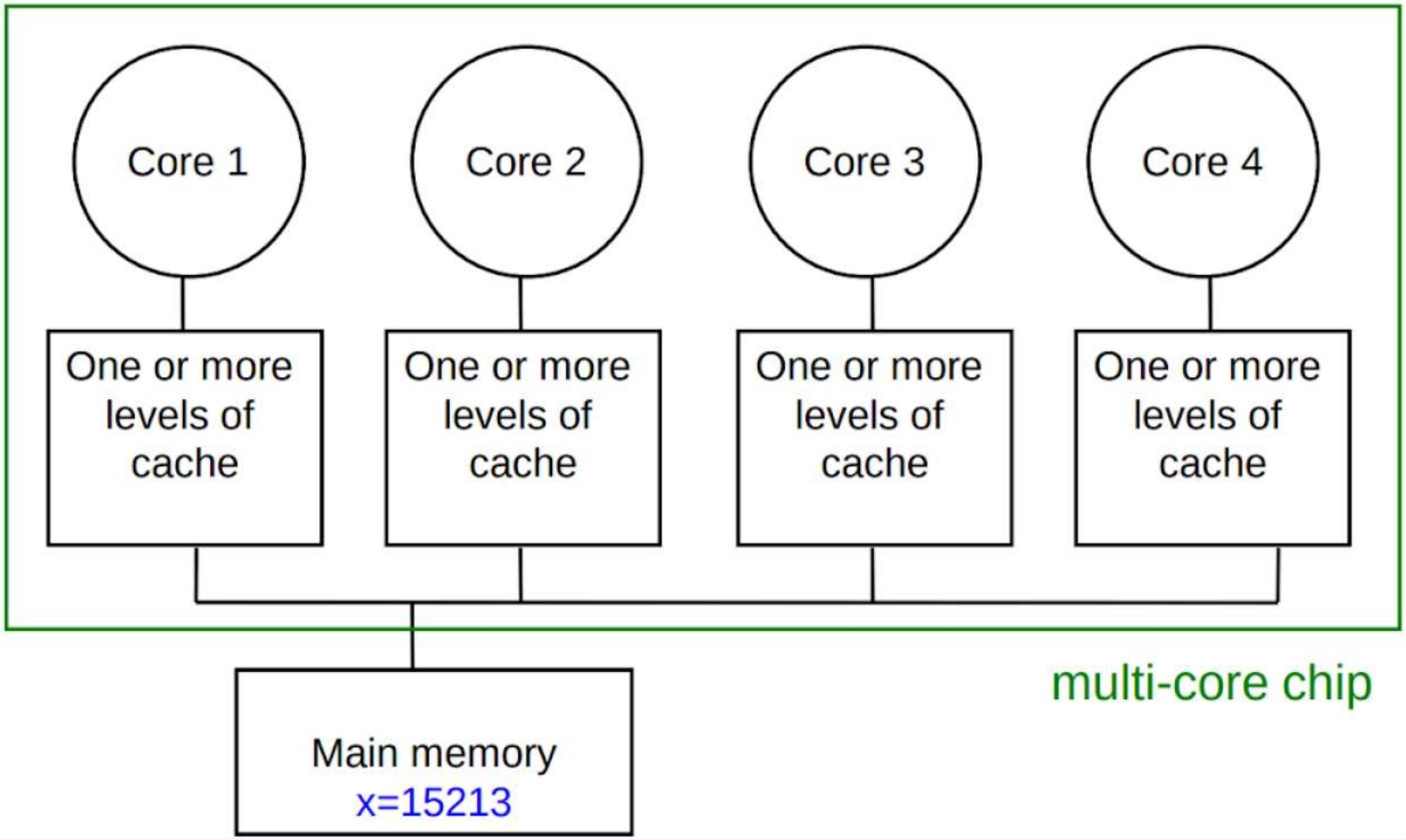
- Since we have private caches:
  How to keep the data consistent across caches?
- Each core should perceive the memory as a monolithic array, shared by all the cores



https://course.ece.cmu.edu/~ece600/lectures/lecture17.pdf

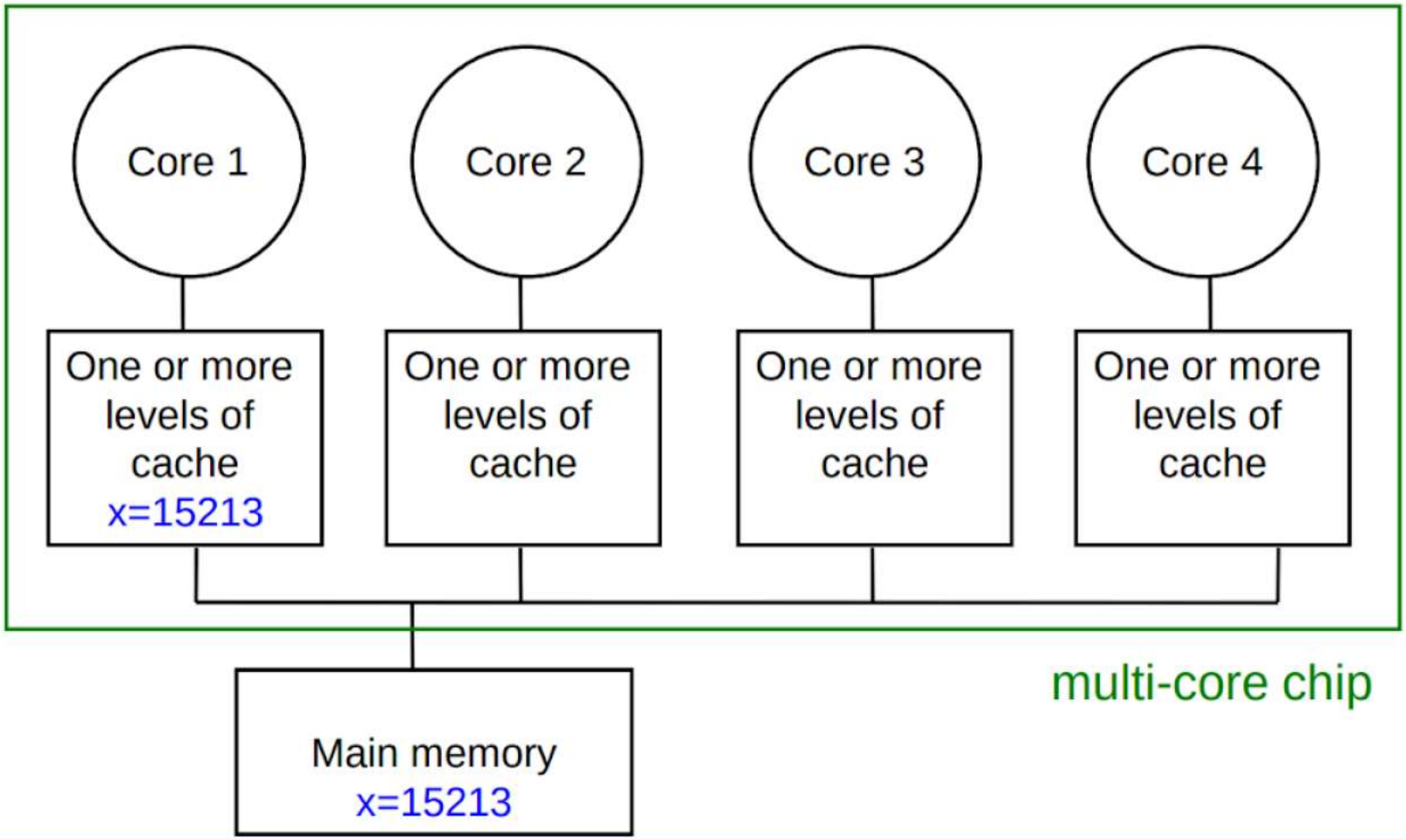
The Cache Coherence Problem

Suppose variable x initially contains 15213



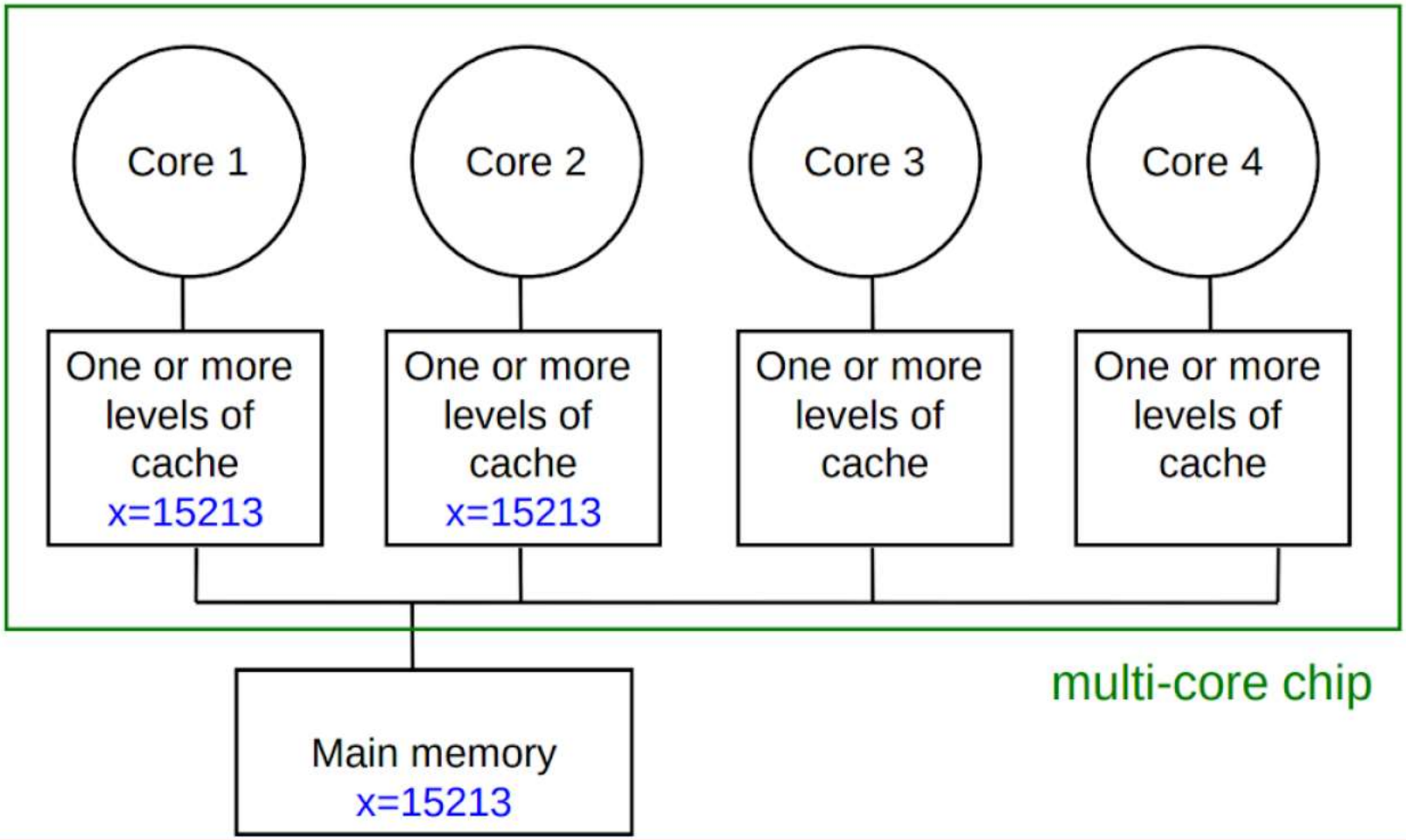
The Cache Coherence Problem

Core 1 reads x



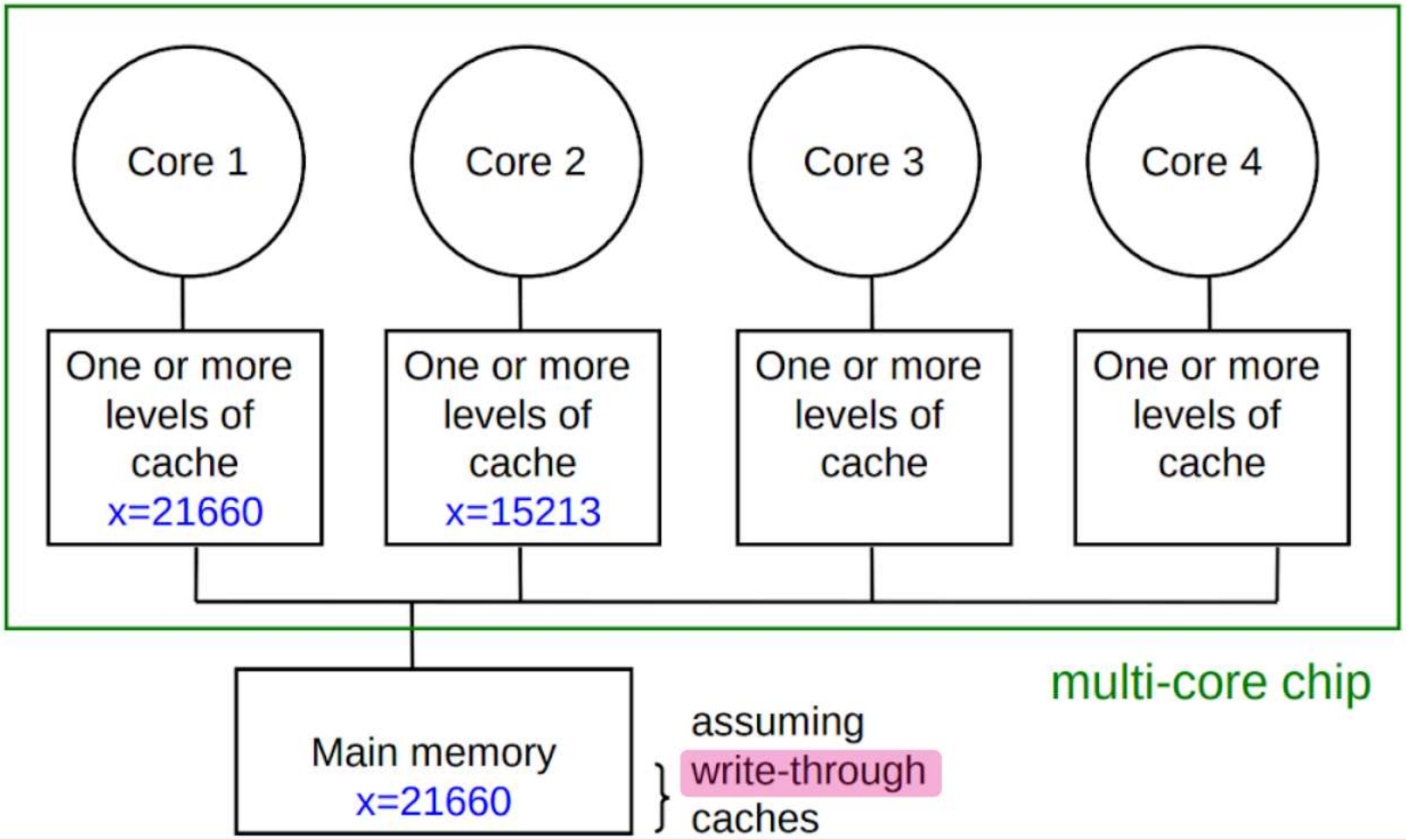
The Cache Coherence Problem

Core 2 reads x



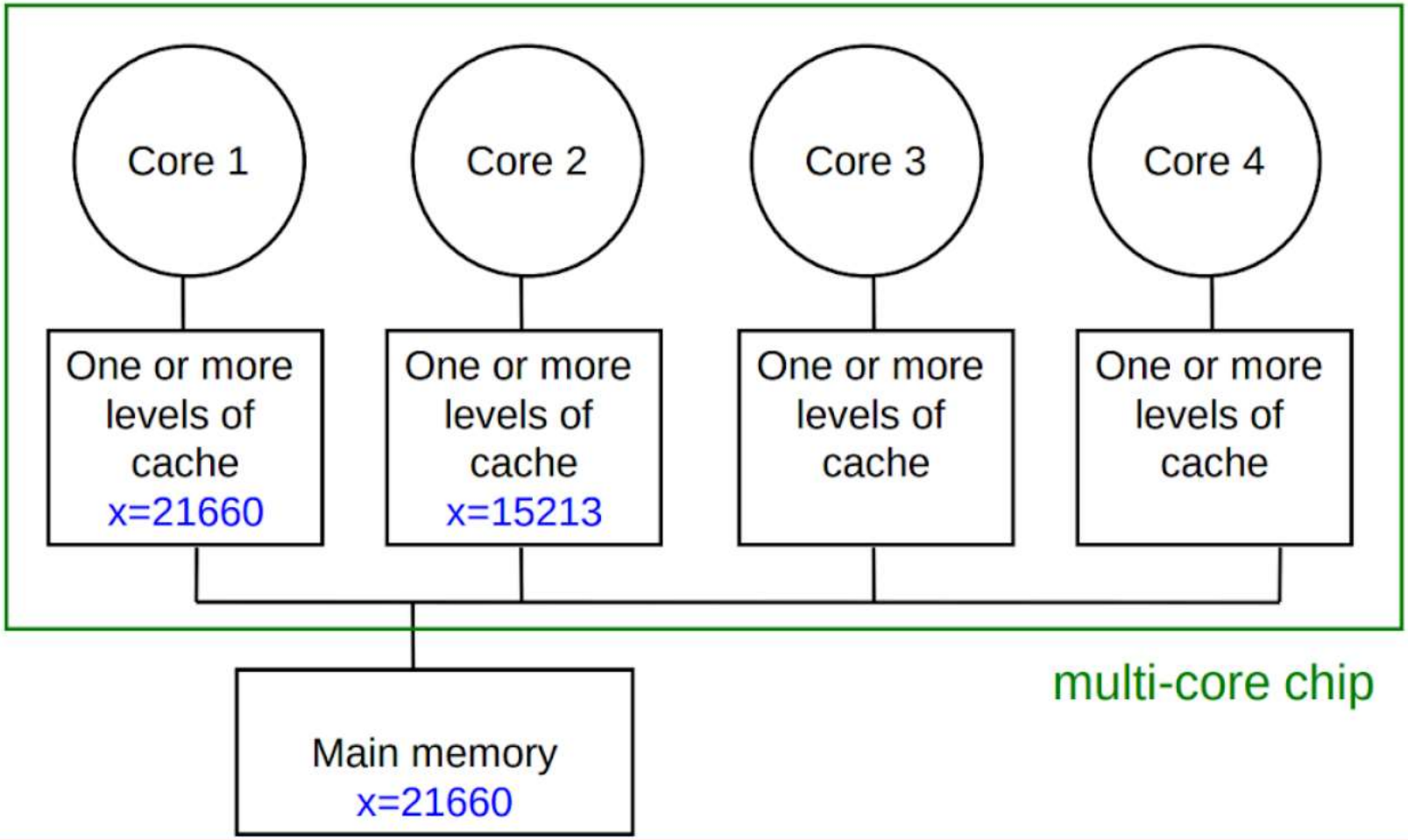
The Cache Coherence Problem

Core 1 writes to x, setting it to 21660



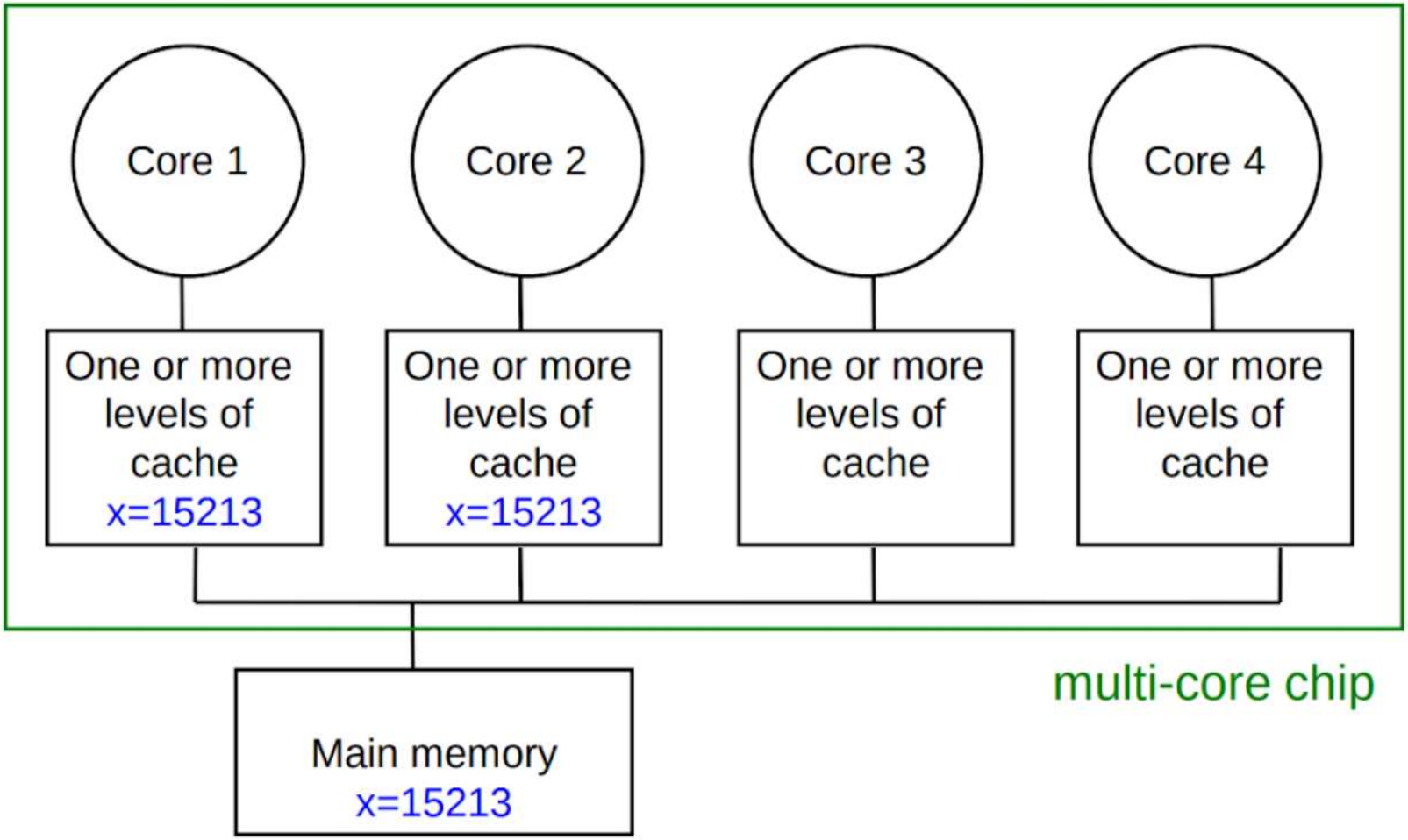
The Cache Coherence Problem

Core 2 attempts to read x… gets a stale copy



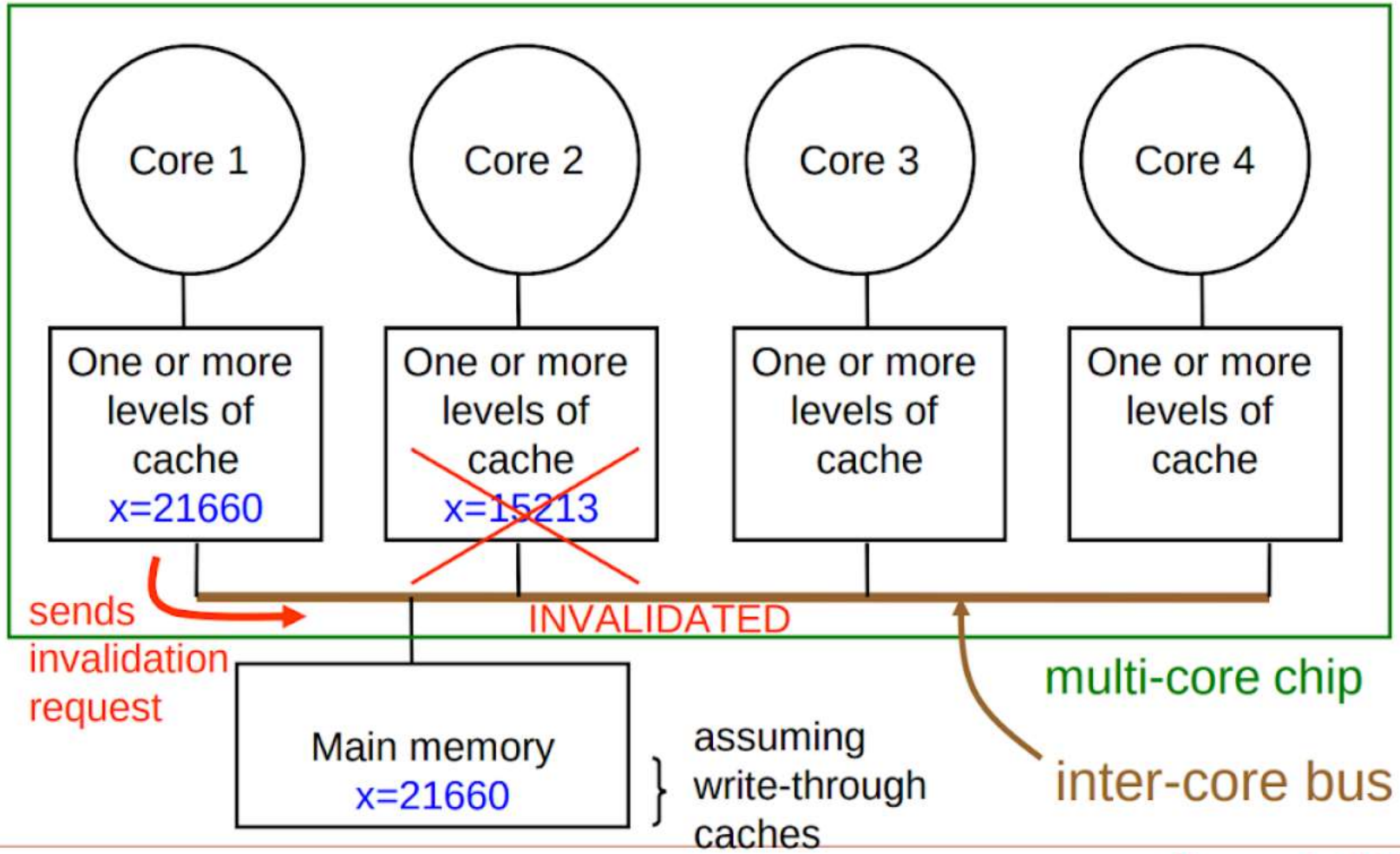
Invalidation Based Cache Coherence Protocol

Revisited: Cores 1 and 2 have both read x



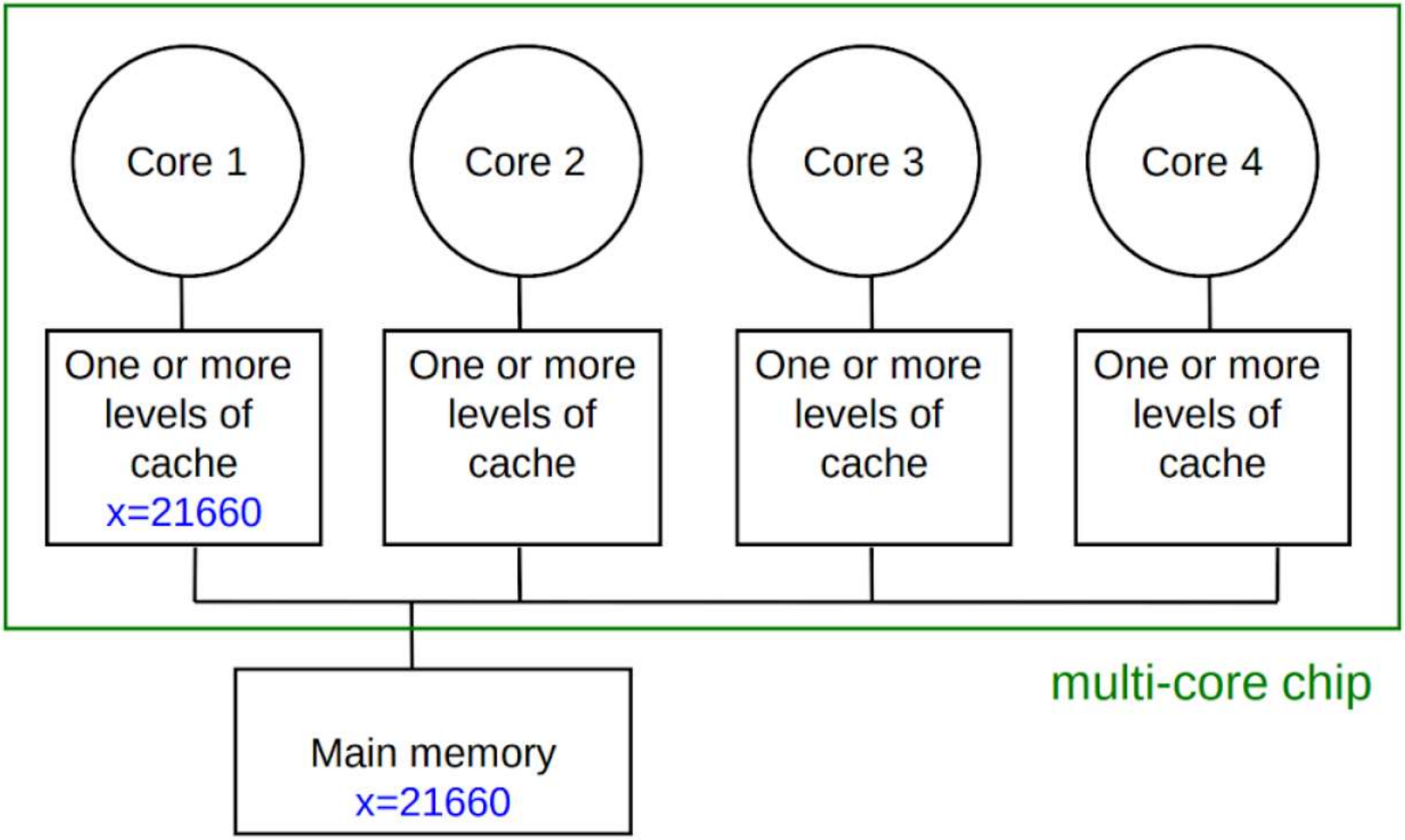
Invalidation Based Cache Coherence Protocol

Core 1 writes to x, setting it to 21660



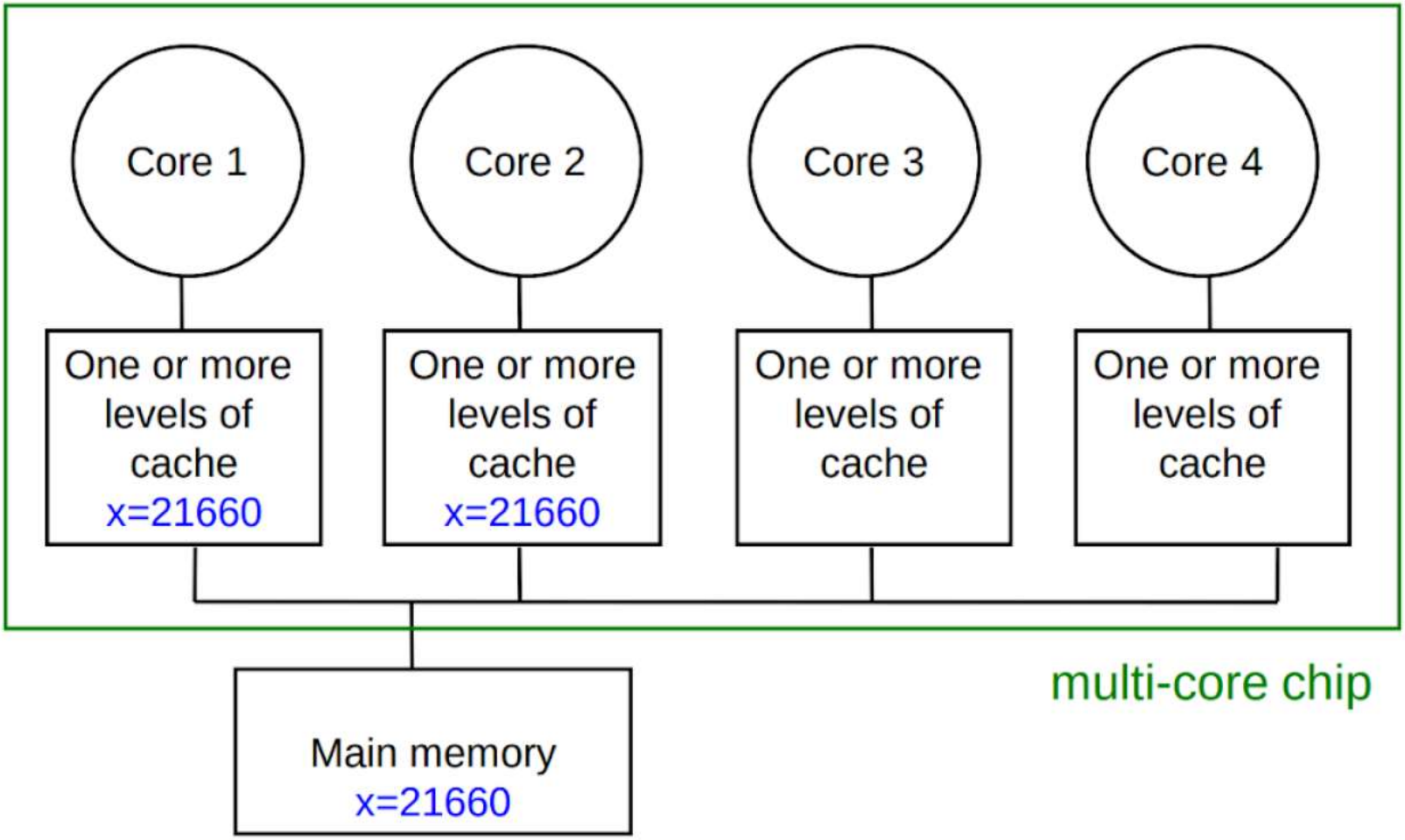
Invalidation Based Cache Coherence Protocol

After invalidation:



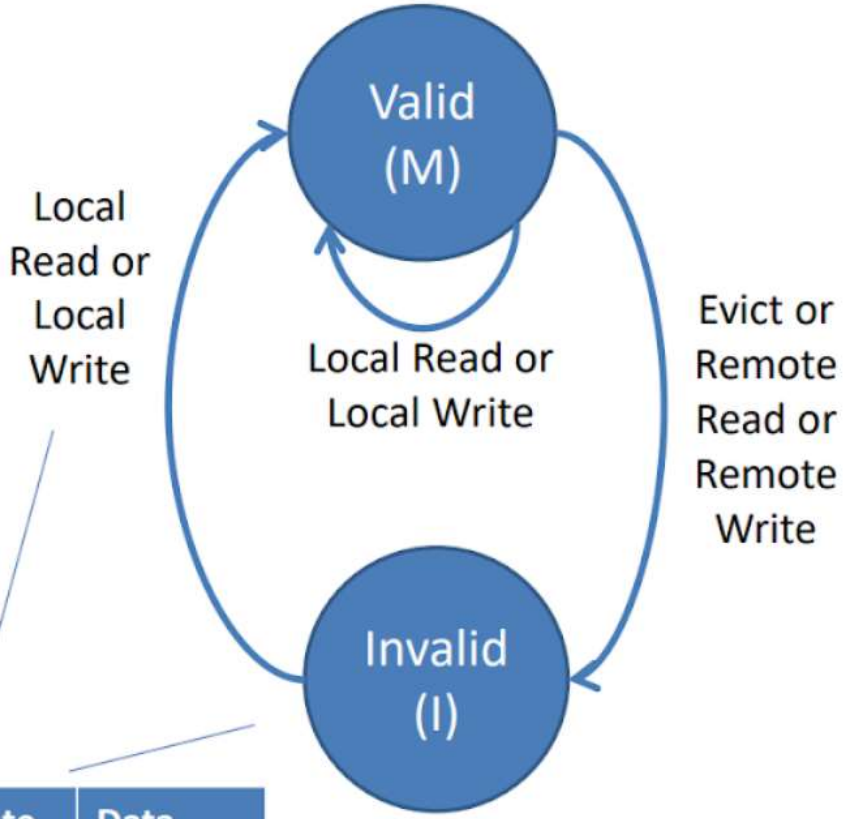
Invalidation Based Cache Coherence Protocol

Core 2 reads x. Cache misses, and loads the new copy.



Minimal Coherence Protocol (Write-Back Cache

- Blocks are always private or exclusive
- State transitions:
  - Local read: I->M, fetch, invalidate other copies
  - Local write: I->M, fetch, invalidate other copies
  - Evict: M->I, write back data
  - Remote read: M->I, write back data
  - Remote write: M->I, write back data



Cache

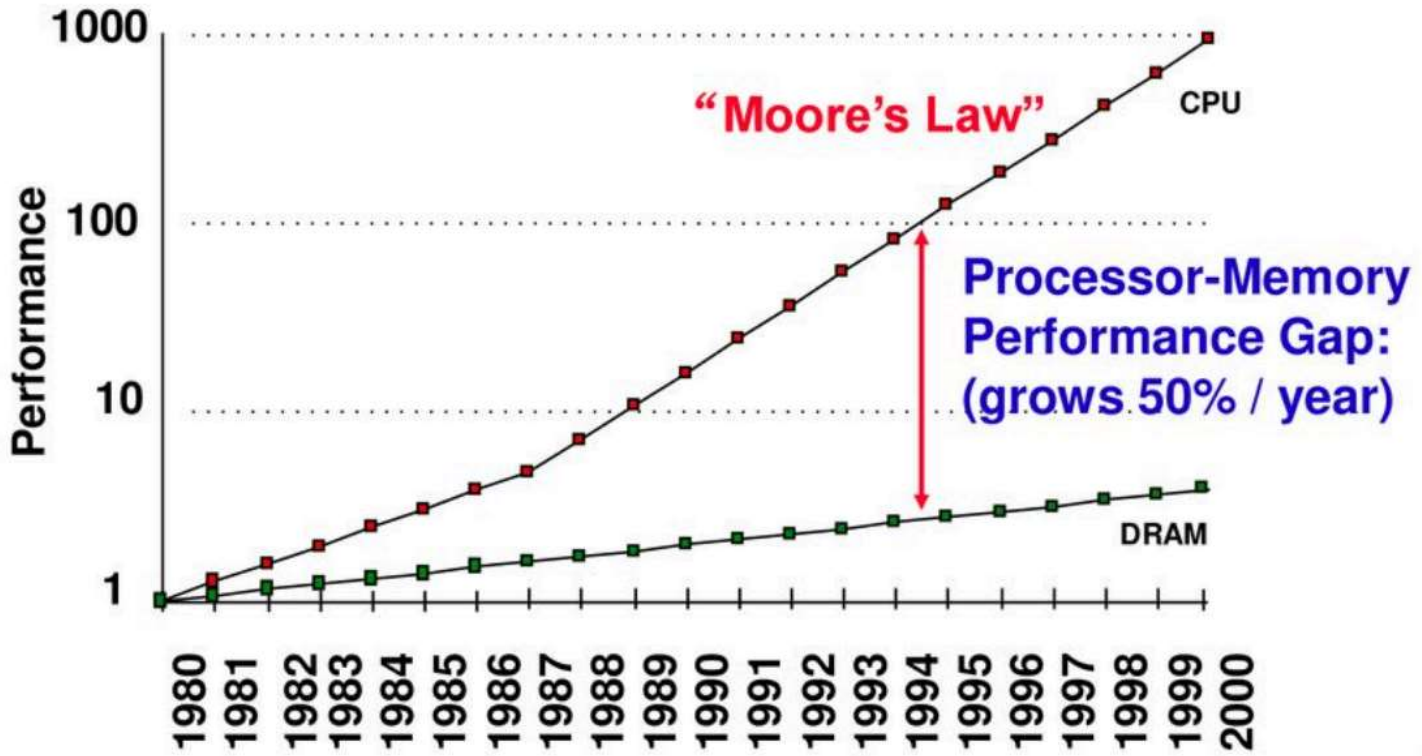
Tag	State	Data
A	M	...
B	I	...

modern protocols have more state (MESI) enable core 0 to read cache line from

Prefetching

ITU CPH

"Memory Wall"



Hide the Memory Latency

- **Many techniques have been proposed to further hide/tolerate the increasing memory latency.**
  - For example
    - Caches
    - Locality optimization
    - Pipelining
    - Out-of-order execution
    - Multithreading
- **Prefetching is one of the well studied techniques to hide memory latency.**
  - Some prefetching schemes have been adopted in commercial processors.

How Prefetching Works?

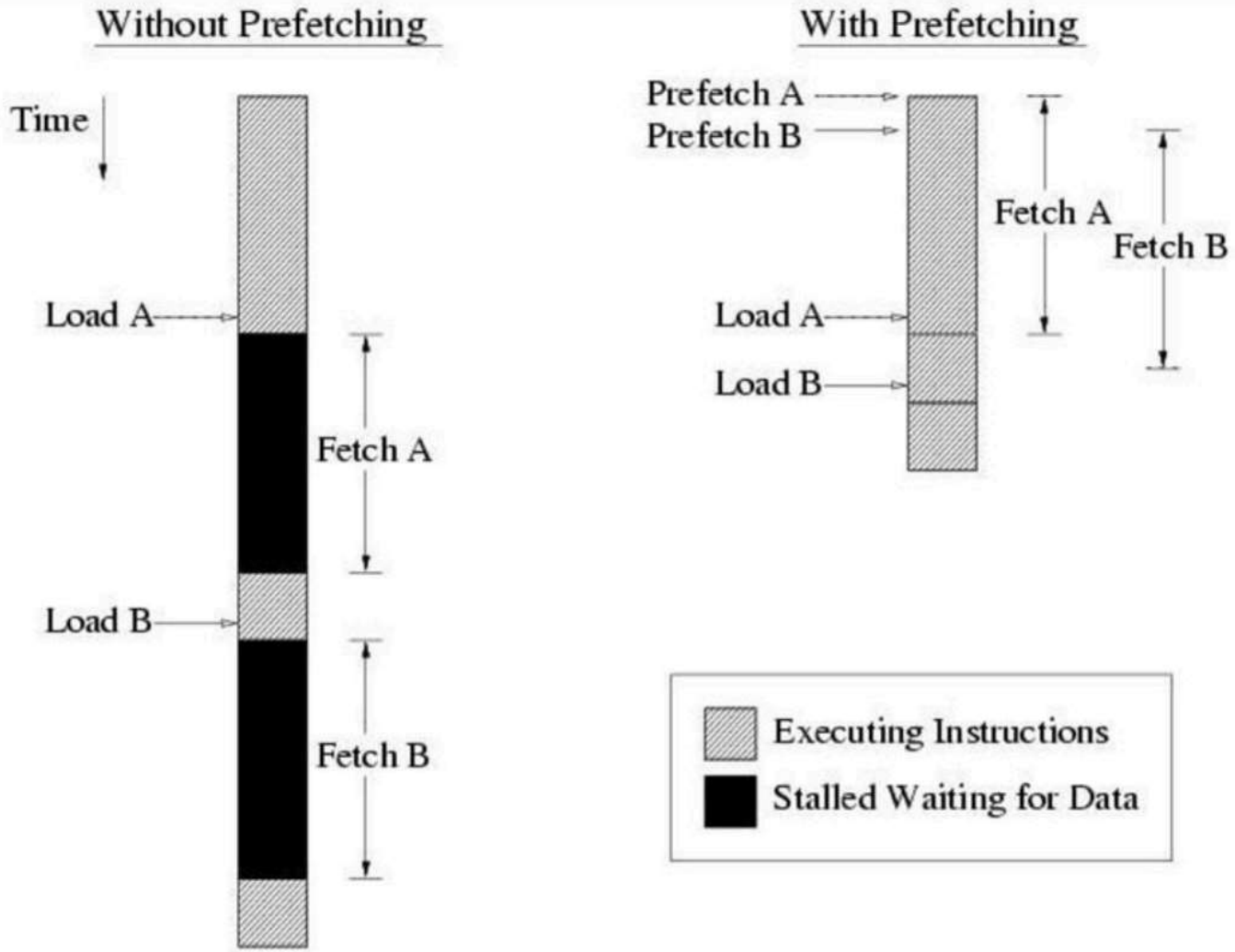


Figure 1.4: Illustration of how prefetching tolerates memory latency.

Prefetching Classification

- **Various prefetching techniques have been proposed.**
  - Instruction Prefetching vs. Data Prefetching
  - Software-controlled prefetching vs. Hardware-controlled prefetching.
  - Data prefetching for different structures in general purpose programs:
    - Prefetching for array structures.
    - Prefetching for pointer and linked data structures.

Basic Questions

1. ***When* to initiate prefetches?**
   - Timely
     - Too early → replace other useful data (cache pollution) or be replaced before being used
     - Too late → cannot hide processor stall
2. ***Where* to place prefetched data?**
   - Cache or dedicated buffer
3. ***What* to be prefetched?**

Array Prefetching Approaches

- **Software-based**
  - Explicit "fetch" instructions
  - Additional instructions executed
- **Hardware-based**
  - Special hardware
  - Unnecessary prefetchings (w/o compile-time information)

Side Effects and Requirements

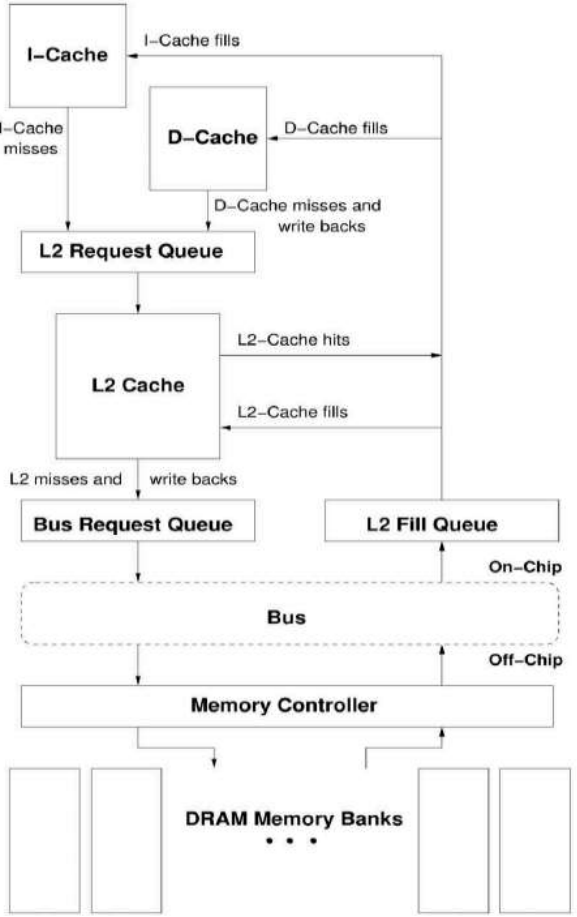
- **Side effects**
  - Prematurely prefetched blocks → possible "cache pollution"
  - Removing processor stall cycles (increase memory request frequency);
  - Unnecessary prefetchings → higher demand on memory bandwidth
- **Requirements**
  - Timely
  - Useful
  - Low overhead

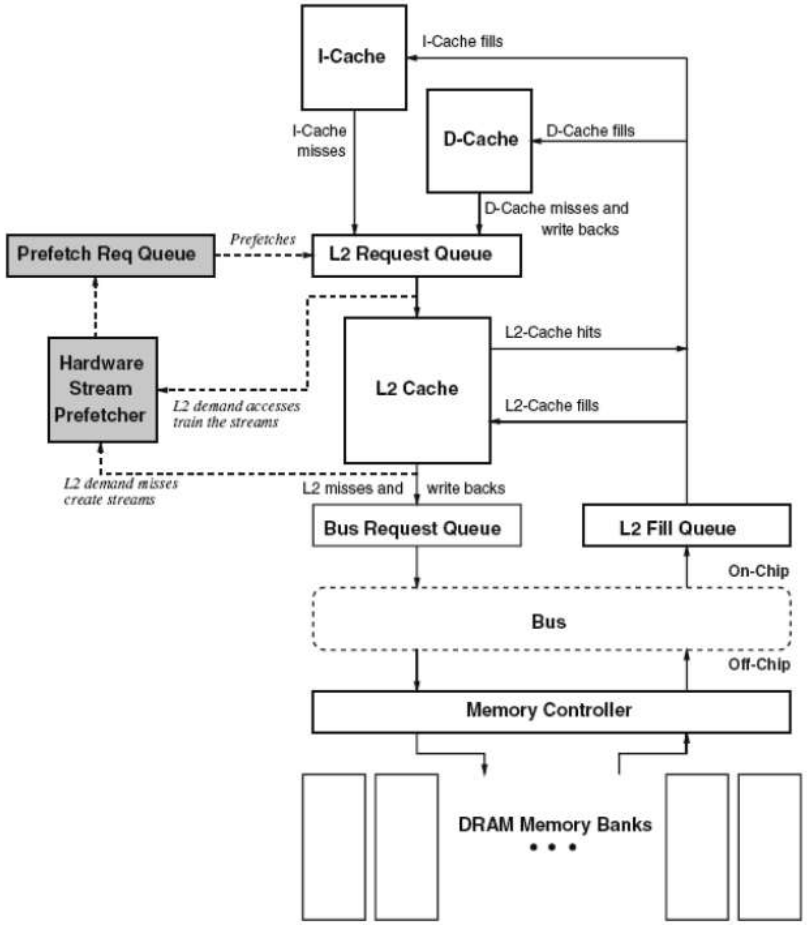
Hardware Data Prefetching

- **No need for programmer or compiler intervention**
- **No changes to existing executables**
- **Take advantage of run-time information**

- **E.g.,**
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer"
  - On miss check stream buffer
- **Works with data blocks too:**
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches

How a Prefetcher Fits in the Memory System





ITU CPH

Sequential Prefetching

- **Take advantage of spatial locality**
- ***One block lookahead* (OBL) approach**
  - Initiate a prefetch for block *b+1* when block *b* is accessed
  - Prefetch-on-miss
    - Whenever an access for block *b* results in a cache miss
  - Tagged prefetch
    - Associates a tag bit with every memory block
    - When a block is demand-fetched or a prefetched block is referenced for the first time next block is fetched.
    - Used in HP PA7200

OBL Approaches

- **Prefetch-on-miss**

- **Tagged prefetch**

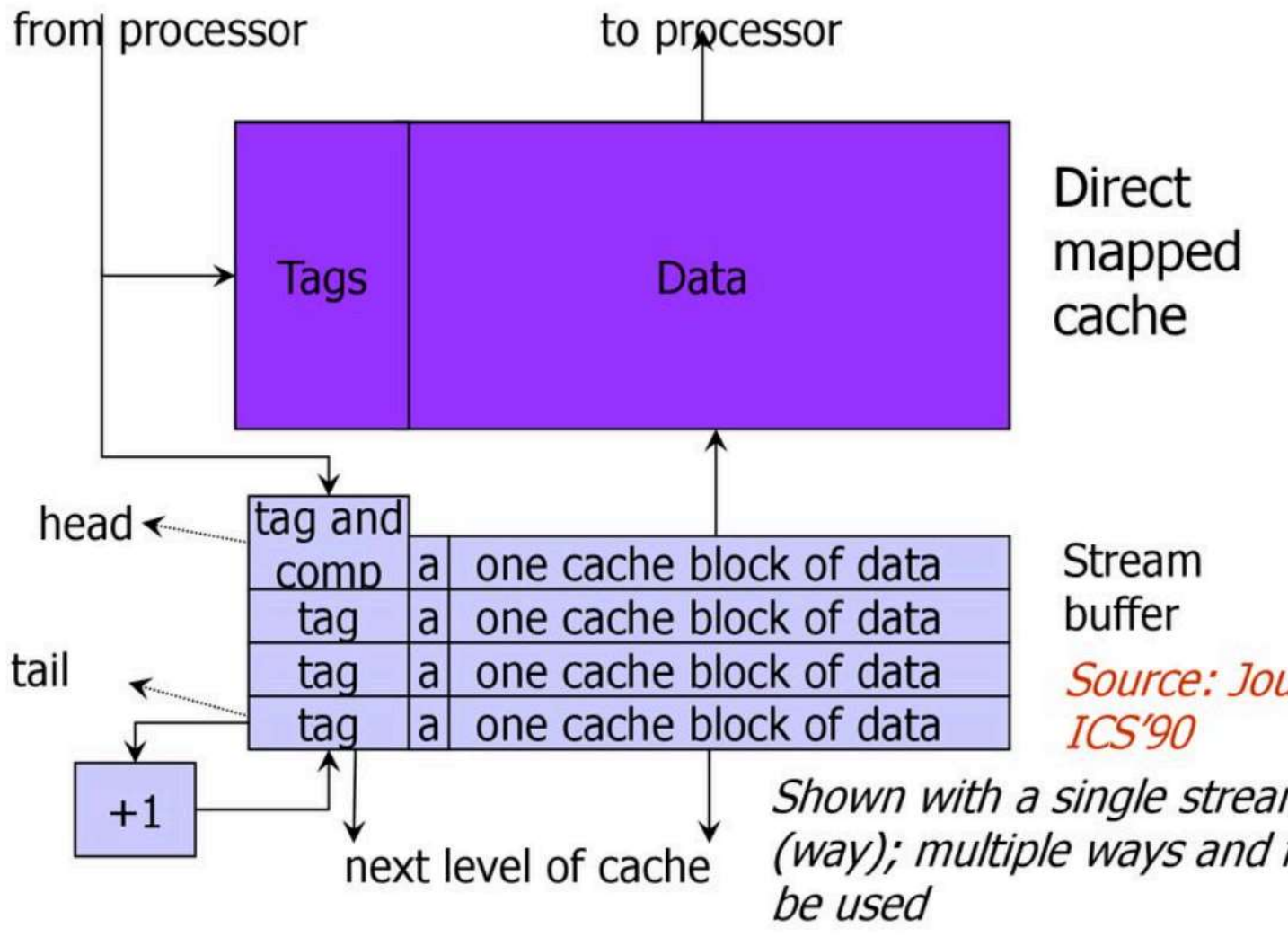
Degree of Prefetching

- **OBL may not initiate prefetch far enough to avoid processor memory stall**
- **Prefetch K > 1 subsequent blocks**
  - Additional traffic and cache pollution
- **Adaptive sequential prefetching**
  - Vary the value of *K* during program execution
  - High spatial locality → large K value
  - Prefetch efficiency metric
  - Periodically calculated
  - Ratio of useful prefetches to total prefetches

Stream Buffer

- ***K* prefetched blocks → FIFO stream buffer**
- **As each buffer entry is referenced**
  - Move it to cache
  - Prefetch a new block to stream buffer
- **Avoid cache pollution**

Stream Buffer Diagram



Sequential Prefetching

- **Pros:**
  - No changes to executables
  - Simple hardware
- **Cons:**
  - Only applies for good spatial locality
  - Works poorly for nonsequential accesses
  - Unnecessary prefetches for scalars and array accesses with large stride

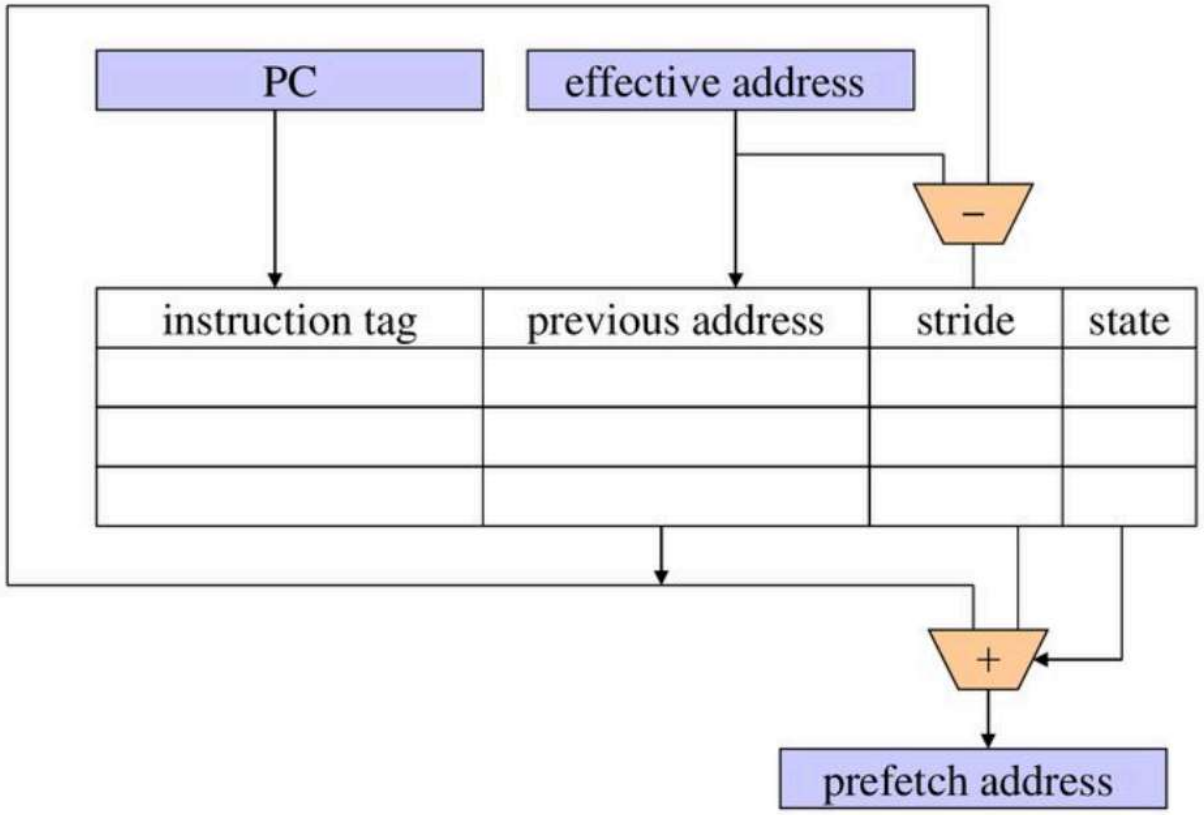
Prefetching with Arbitrary Strides

- Employ special logic to monitor the processor's address referencing pattern
- Detect constant stride array references originating from looping structures
- Compare successive addresses used by load or store instructions

Reference Prediction Table (RPT)

- **Hold information for the most recently used memory instructions to predict their access pattern**
  - Address of the memory instruction
  - Previous address accessed by the instruction
  - Stride value
  - State field

Organization of RPT



Software vs. Hardware Prefetching

- **Software**
  - Compile-time analysis, schedule fetch instructions within user program
- **Hardware**
  - Run-time analysis w/o any compiler or user support
- **Integration**
  - e.g. compiler calculates degree of prefetching (*K*) for a particular reference stream and pass it on to the prefetch hardware.