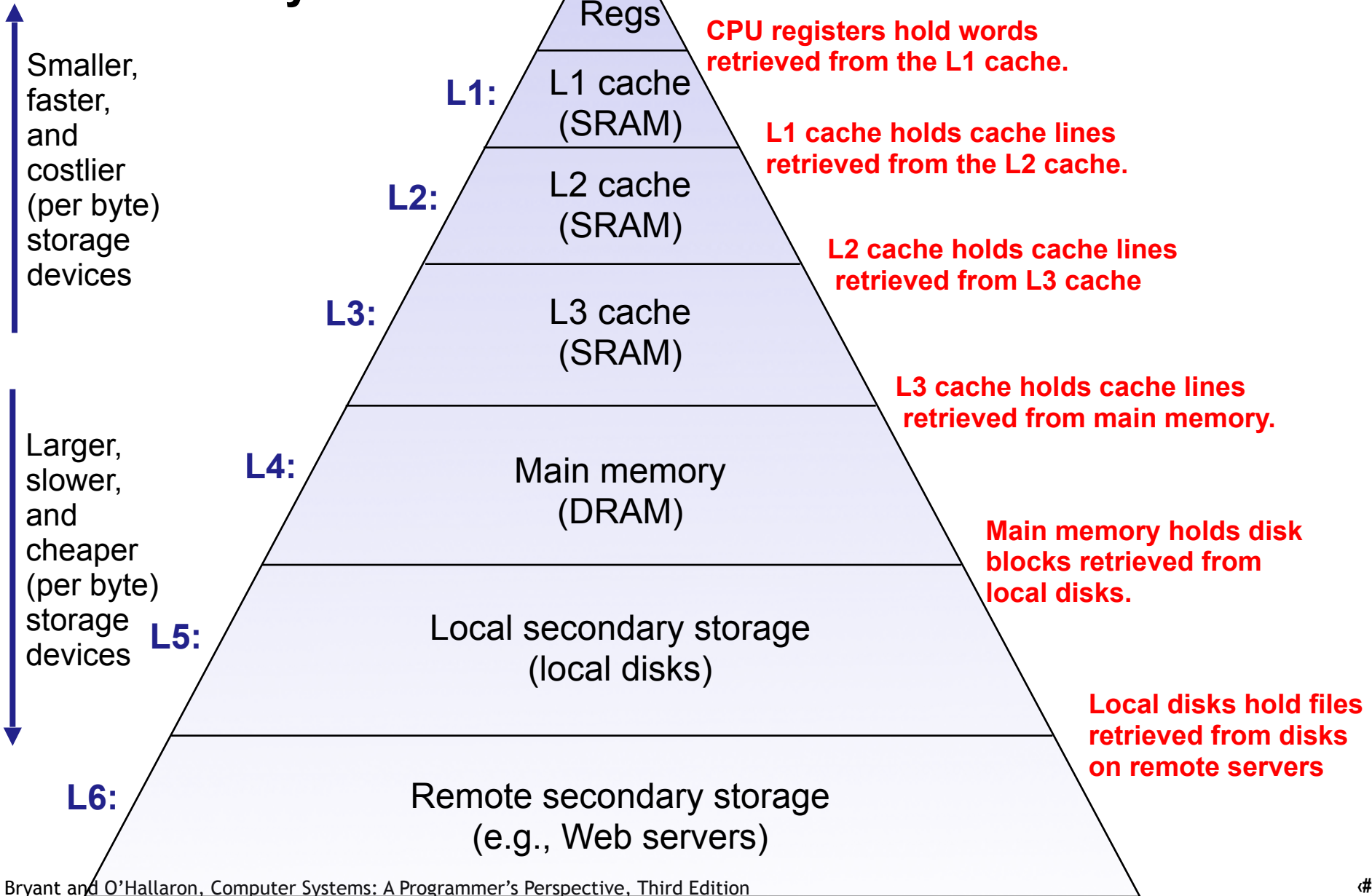


Cache Memories

Modified version of slides by –
Randal E. Bryant and David R. O'Hallaron

Example Memory Hierarchy



Examples of Caching in the Mem. Hierarchy

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 MMU
L1 cache	64-byte blocks	On-Chip L1	4	Hardware
L2 cache	64-byte blocks	On-Chip L2	10	Hardware
Virtual Memory	4-KB pages	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	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

Storage Trends

SRAM

Metric	1985	1990	1995	2000	2005	2010	2015	2015:1985
\$/MB	2,900	320	256	100	75	60	320	116
access (ns)	150	35	15	3	2	1.5	200	115

DRAM

Metric	1985	1990	1995	2000	2005	2010	2015	2015:1985
\$/MB	880	100	30	1	0.1	0.06	0.02	44,000
access (ns)	200	100	70	60	50	40	20	10
typical size (MB)	0.256	4	16	64	2,000	8,000	16,000	62,500

Disk

Metric	1985	1990	1995	2000	2005	2010	2015	2015:1985
\$/GB	100,000	8,000	300	10	5	0.3	0.03	3,333,333
access (ms)	75	28	10	8	5	3	3	25
typical size (GB)	0.01	0.16	1	20	160	1,500	3,000	300,000

CPU Clock Rates

Inflection point in computer history
when designers hit the “Power Wall”



	1985	1990	1995	2003	2005	2010	2015	2015:1985
CPU	80286	80386	Pentium	P-4	Core 2	Core i7(n)	Core i7(h)	
Clock rate (MHz)	6	20	150	3,300	2,000	2,500	3,000	500
Cycle time (ns)	166	50	6	0.30	0.50	0.4	0.33	500
Cores	1	1	1	1	2	4	4	4
Effective cycle time (ns)	166	50	6	0.30	0.25	0.10	0.08	2,075

(n) Nehalem processor
(h) Haswell processor

Today

■ Cache memory organization and operation

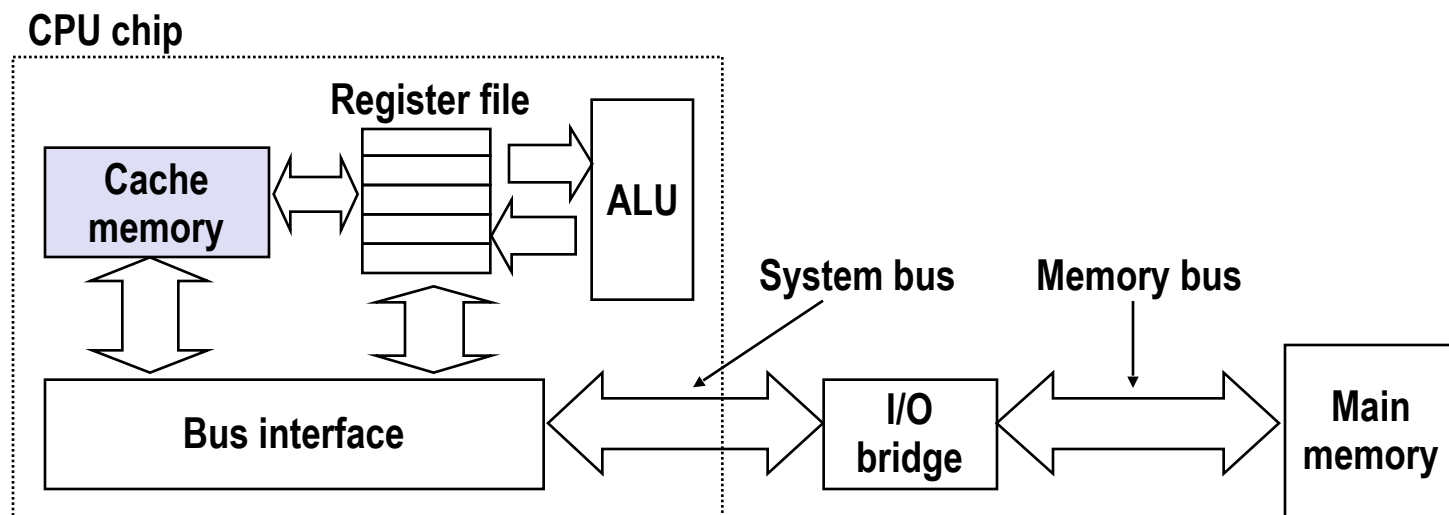
■ Performance impact of caches

- The memory mountain
- Rearranging loops to improve spatial locality
- Using blocking to improve temporal locality

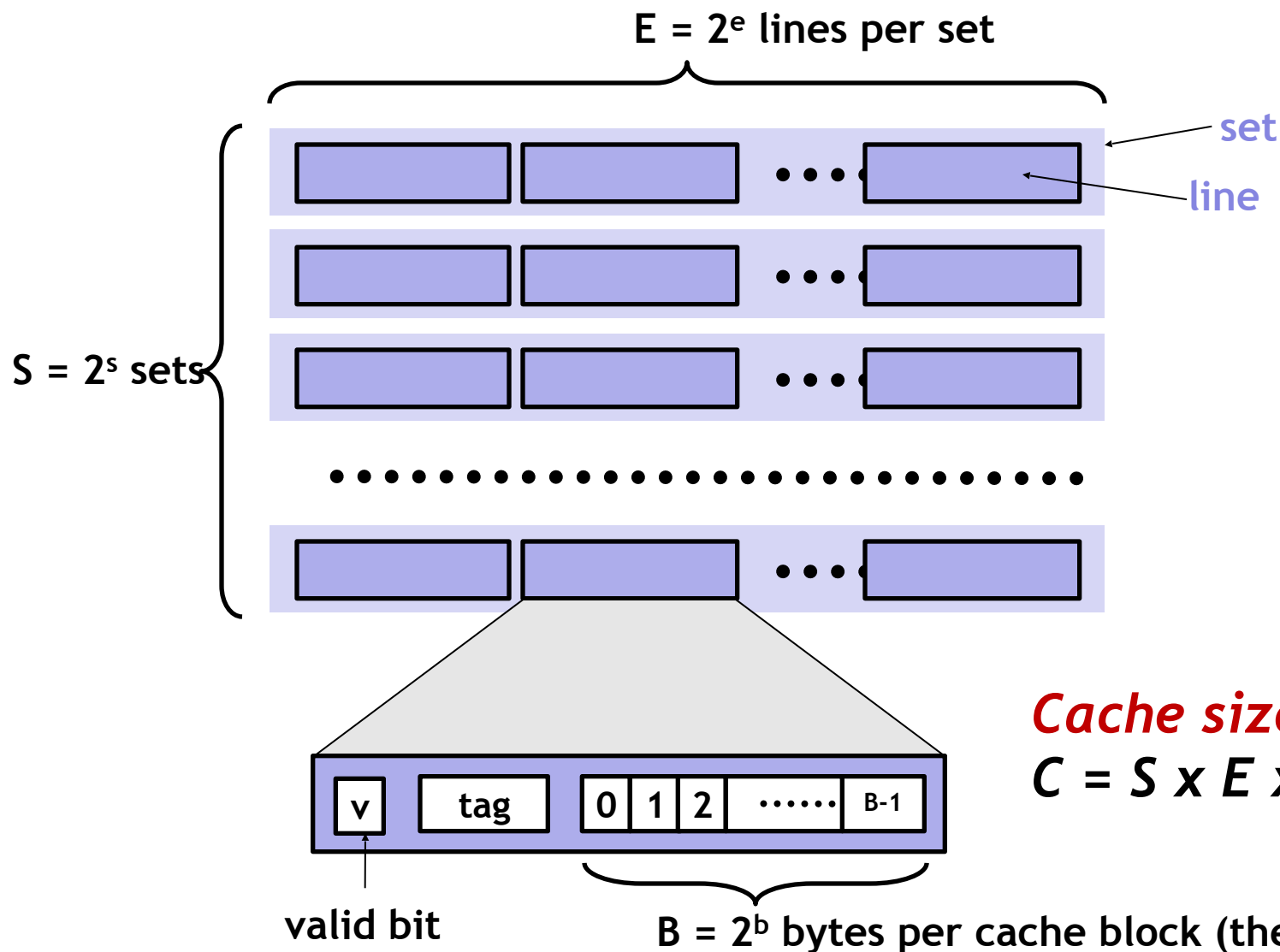


Cache Memories

- **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 cache
- Typical system structure:

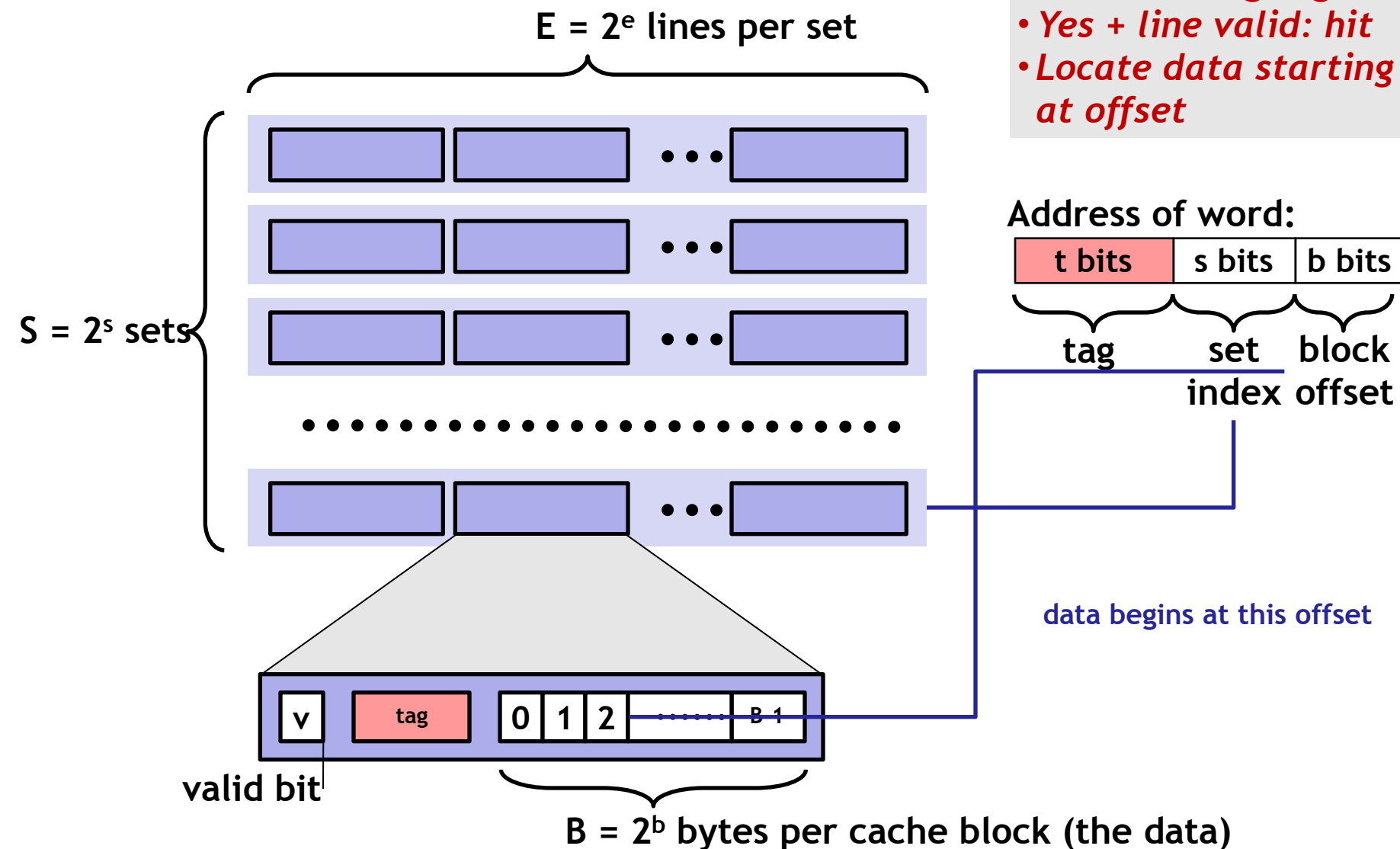


General Cache Organization (S, E, B)



Cache Read

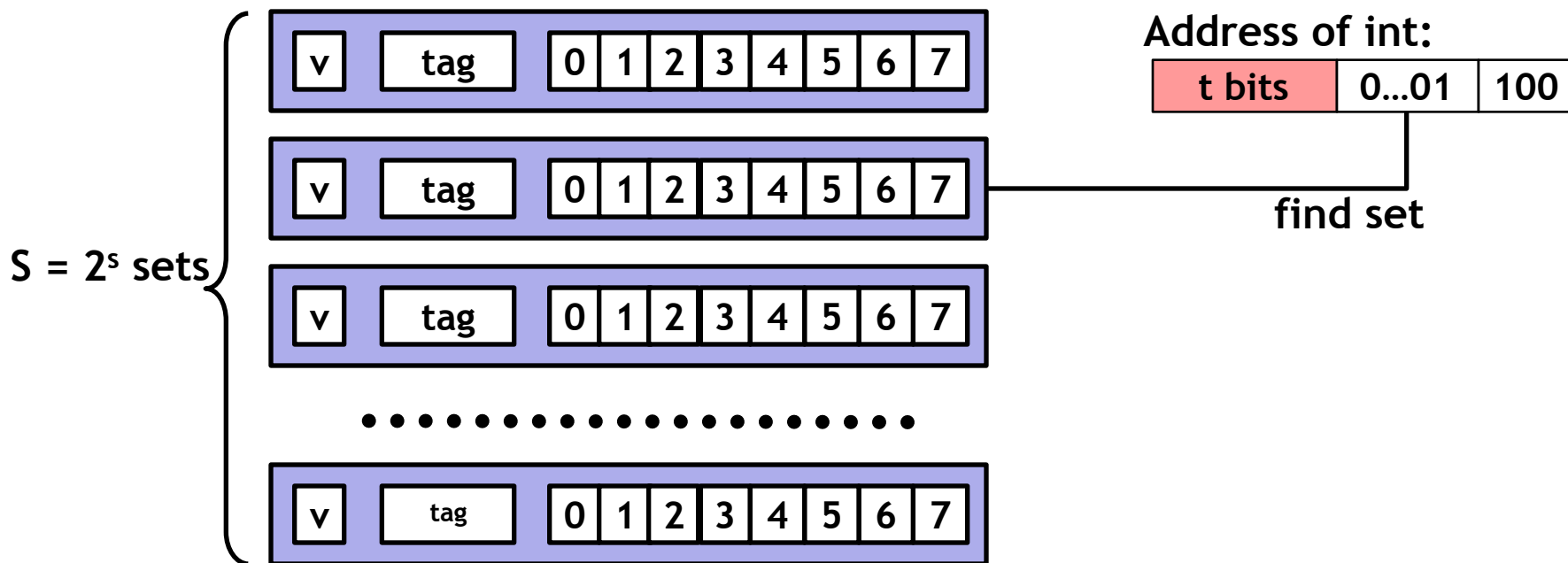
- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*
- *Locate data starting at offset*



Example: Direct Mapped Cache ($E = 1$)

Direct mapped: One line per set

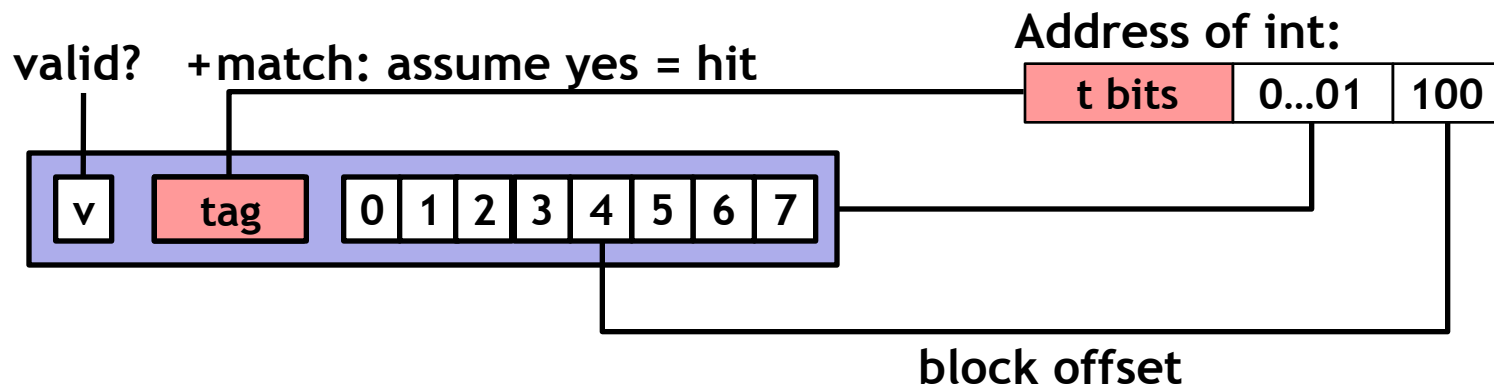
Assume: cache block size 8 bytes



Example: Direct Mapped Cache (E = 1)

Direct mapped: One line per set

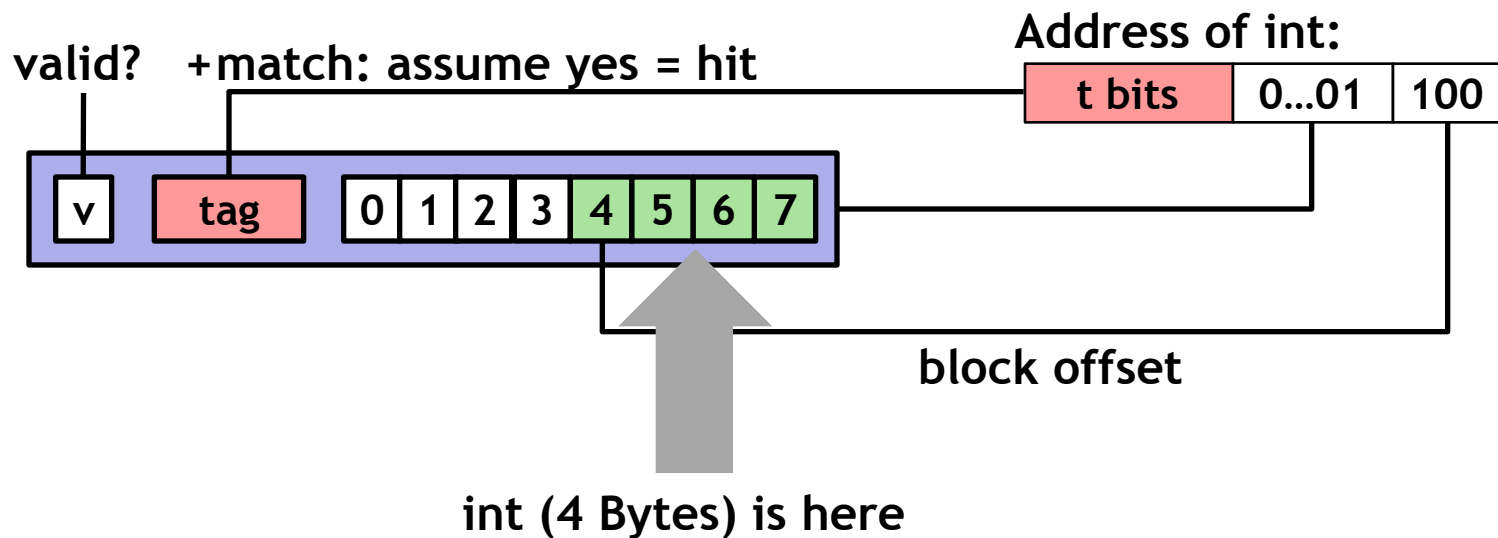
Assume: cache block size 8 bytes



Example: Direct Mapped Cache (E = 1)

Direct mapped: One line per set

Assume: cache block size 8 bytes



If tag doesn't match: old line is evicted and replaced

Direct-Mapped Cache Simulation

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

M=16 bytes (4-bit addresses), B=2 bytes/block,
S=4 sets, E=1 Blocks/set

Address trace (reads, one byte per read):

0	[<u>0000</u> ₂],	miss
1	[<u>0001</u> ₂],	hit
7	[<u>0111</u> ₂],	miss
8	[<u>1000</u> ₂],	miss
0	[<u>0000</u> ₂]	miss

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

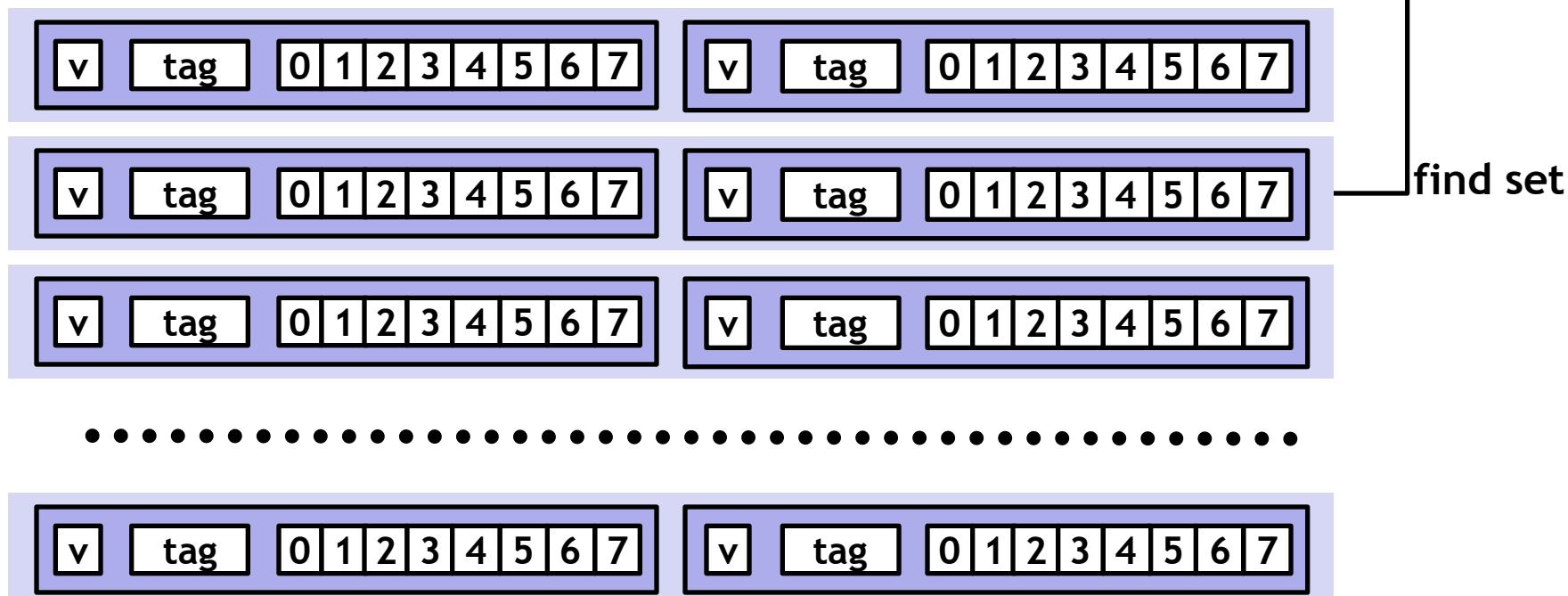
E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set

Assume: cache block size 8 bytes

Address of short int:

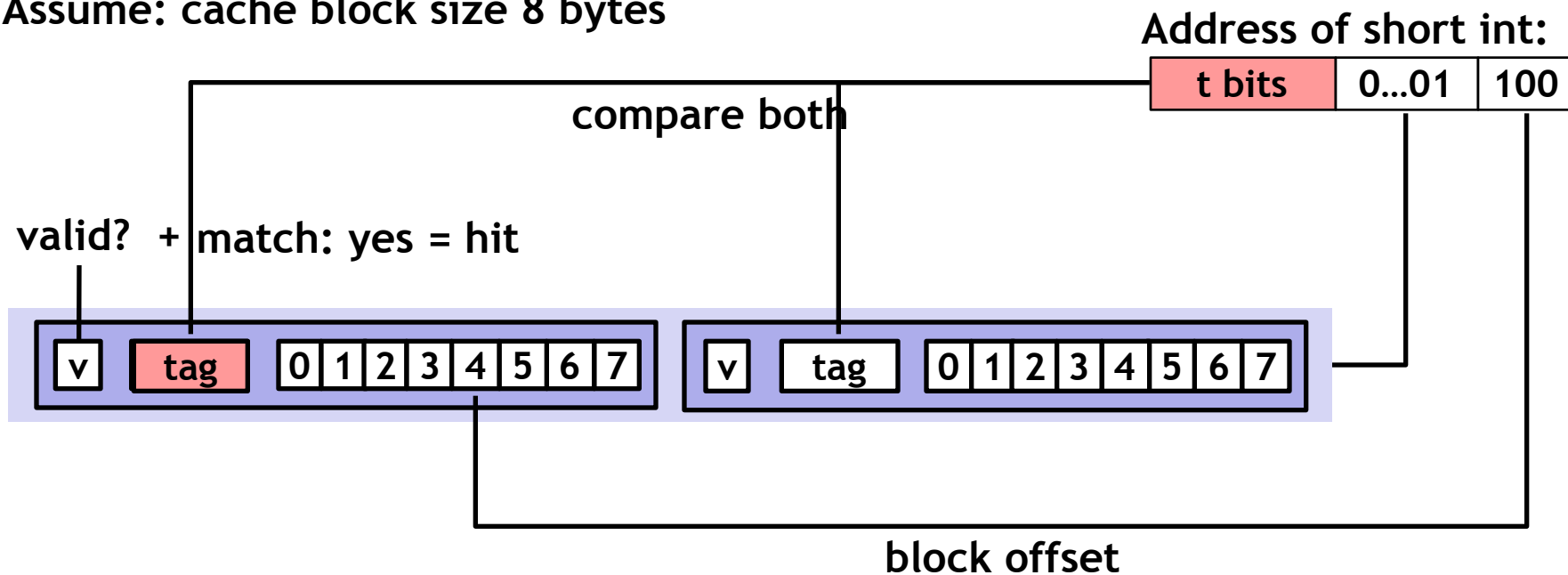
t bits	0...01	100
--------	--------	-----



E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set

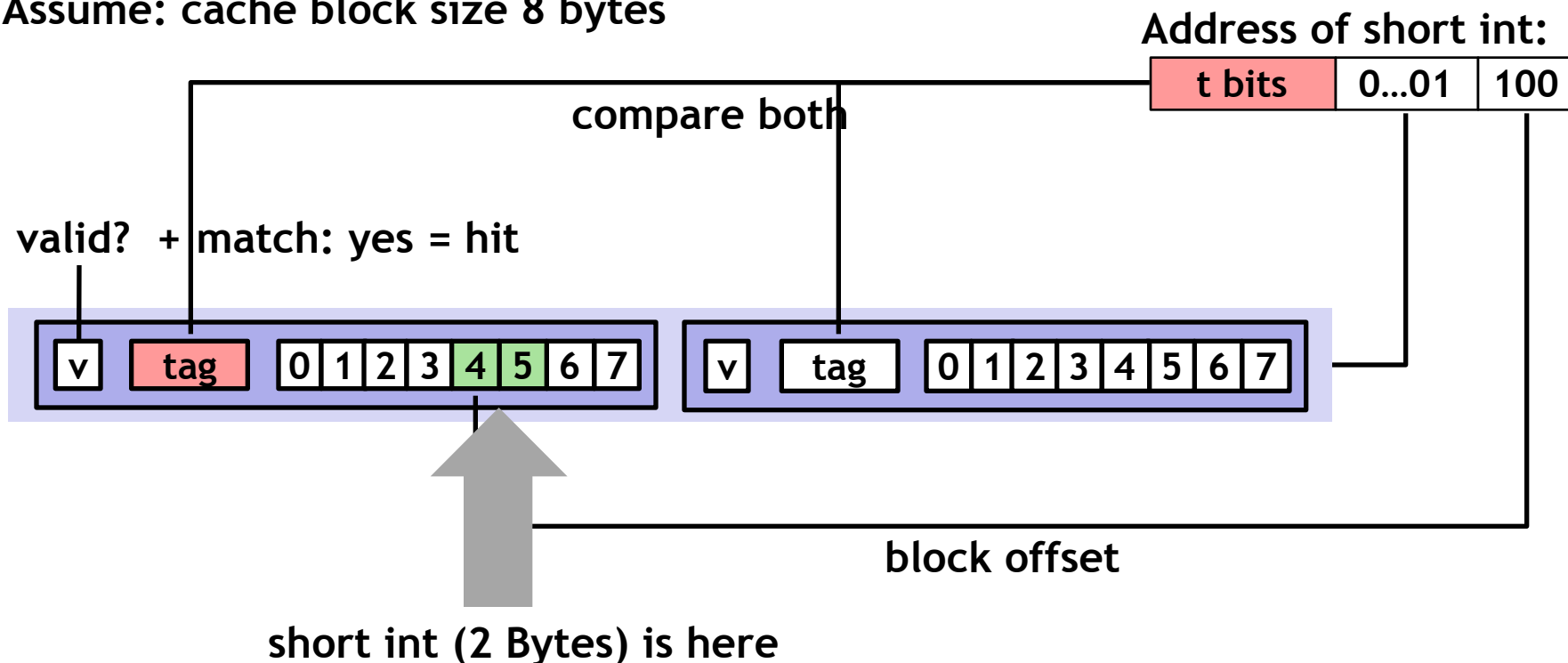
Assume: cache block size 8 bytes



E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set

Assume: cache block size 8 bytes



No match:

- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), .

2-Way Set Associative Cache Simulation

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

M=16 byte addresses, B=2 bytes/block,
S=2 sets, E=2 blocks/set

Address trace (reads, one byte per read):

0	[00 <u>0</u> 0 ₂],	miss
1	[00 <u>0</u> 1 ₂],	hit
7	[01 <u>1</u> 1 ₂],	miss
8	[10 <u>0</u> 0 ₂],	miss
0	[00 <u>0</u> 0 ₂]	hit

	v	Tag	Block
Set 0	1	00	M[0-1]
	1	10	M[8-9]
Set 1	1	01	M[6-7]
	0		

What about writes?

■ Multiple copies of data exist:

- L1, L2, L3, 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)

■ What to do on a write-miss?

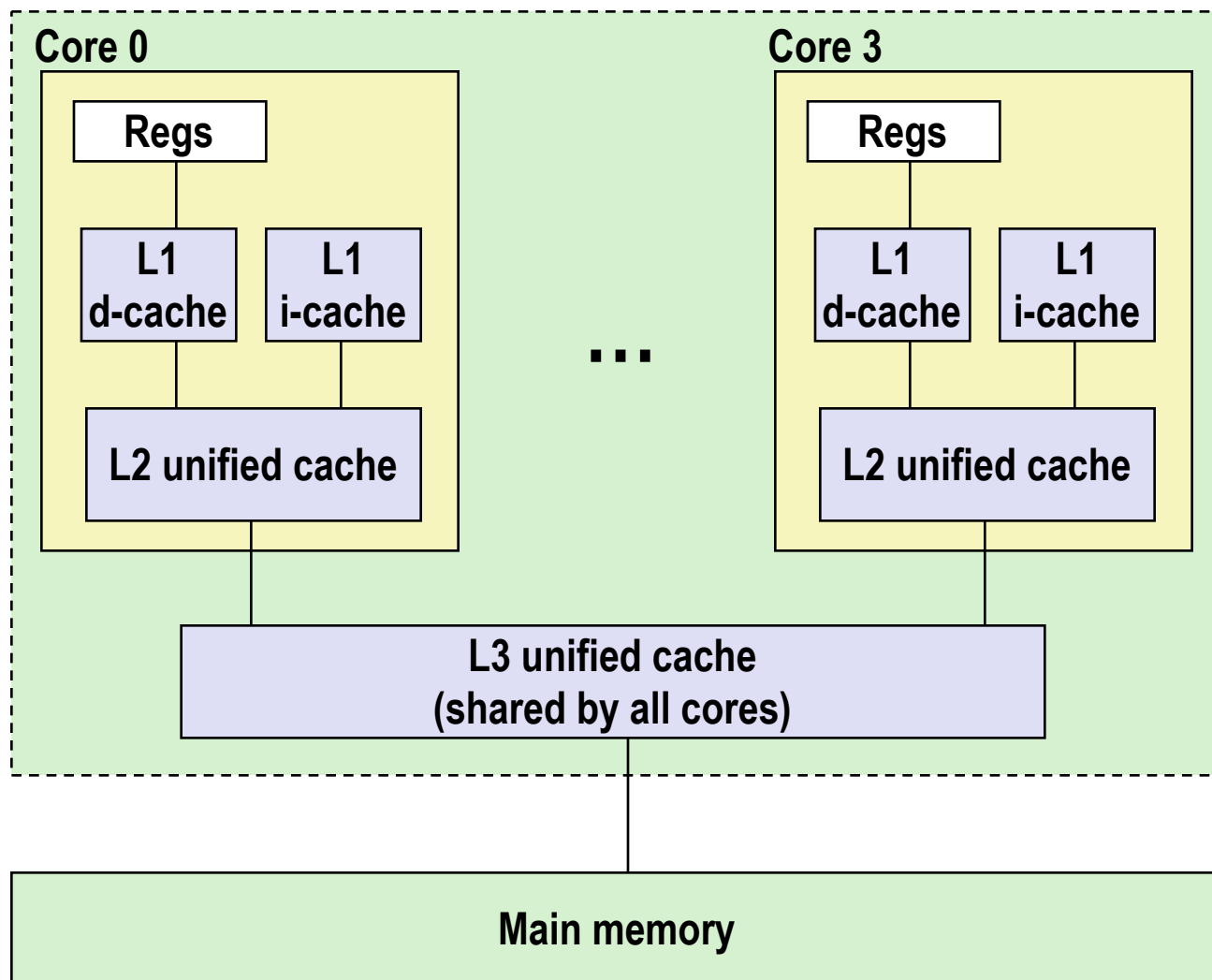
- **Write-allocate** (load into cache, update line in cache)
 - Good if more writes to the location follow
- **No-write-allocate** (writes straight to memory, does not load into cache)

■ Typical

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

Intel Core i7 Cache Hierarchy

Processor package



L1 i-cache and d-cache:

32 KB, 8-way,
Access: 4 cycles

L2 unified cache:

256 KB, 8-way,
Access: 10 cycles

L3 unified cache:

8 MB, 16-way,
Access: 40-75
cycles

**Block size: 64 bytes
for all caches.**

Cache Performance Metrics

■ 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:
 - 4 clock cycle for L1
 - 10 clock cycles for L2

■ Miss Penalty

- Additional time required because of a miss
 - typically 50-200 cycles for main memory (Trend: increasing!)

Let'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: $1 \text{ cycle} + 0.03 * 100 \text{ cycles} = 4 \text{ cycles}$
99% hits: $1 \text{ cycle} + 0.01 * 100 \text{ cycles} = 2 \text{ cycles}$

■ 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

Cache Summary

■ **Cache memories can have significant performance impact**

■ **You can write your programs to exploit this!**

- Focus on the inner loops, where bulk of computations and memory accesses occur.
- Try to maximize spatial locality by reading data objects with sequentially with stride 1.
- Try to maximize temporal locality by using a data object as often as possible once it's read from memory.

Back to Address Translation and Virtual Memory

- Simple memory system example
- Case study: Core i7/Linux memory system
- Memory mapping

Review of Symbols

■ Basic Parameters

- $N = 2^n$: Number of addresses in virtual address space
- $M = 2^m$: Number of addresses in physical address space
- $P = 2^p$: Page size (bytes)

■ Components of the virtual address (VA)

- TLBI: TLB index
- TLBT: TLB tag
- VPO: Virtual page offset
- VPN: Virtual page number

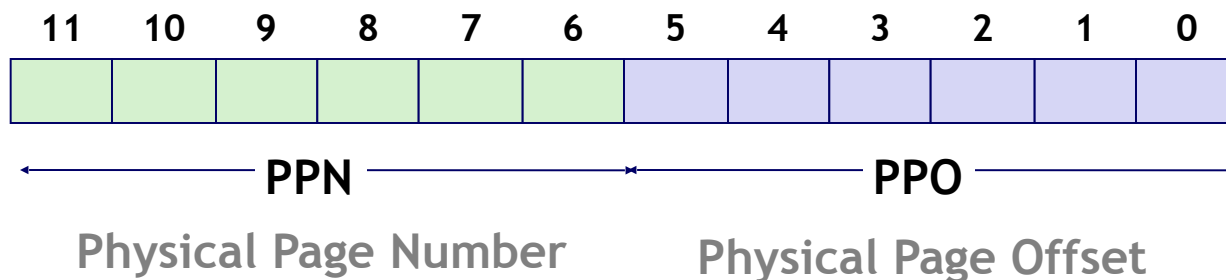
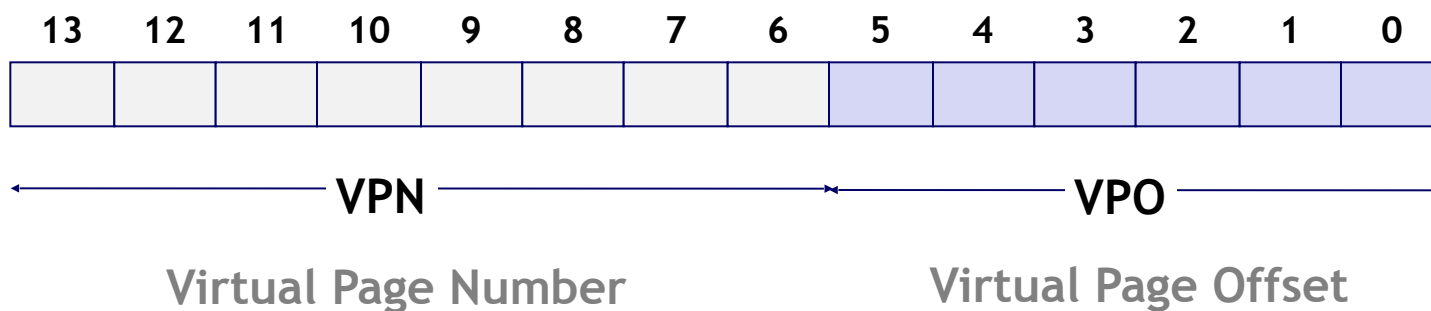
■ Components of the physical address (PA)

- PPO: Physical page offset (same as VPO)
- PPN: Physical page number
- CO: Byte offset within cache line
- CI: Cache index
- CT: Cache tag

Simple Memory System Example

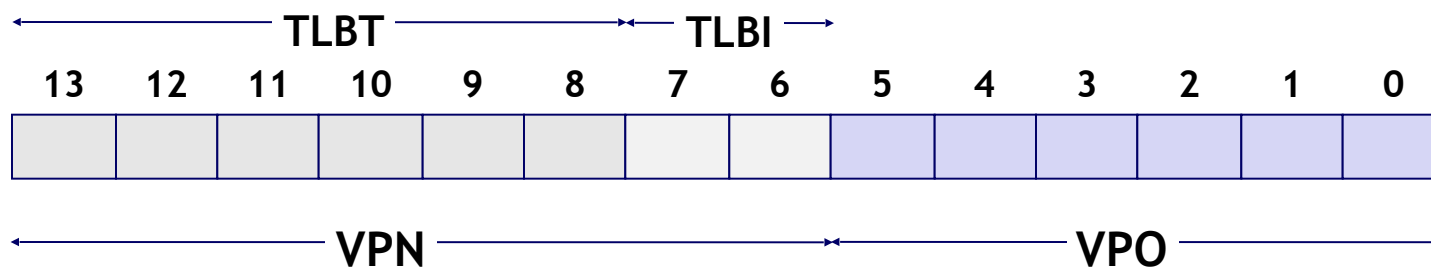
■ Addressing

- 14-bit virtual addresses
- 12-bit physical address
- Page size = 64 bytes



1. Simple Memory System TLB

- 16 entries
- 4-way associative



Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	-	0	09	0D	1	00	-	0	07	02	1
1	03	2D	1	02	-	0	04	-	0	0A	-	0
2	02	-	0	08	-	0	06	-	0	03	-	0
3	07	-	0	03	0D	1	0A	34	1	02	-	0

2. Simple Memory System Page Table

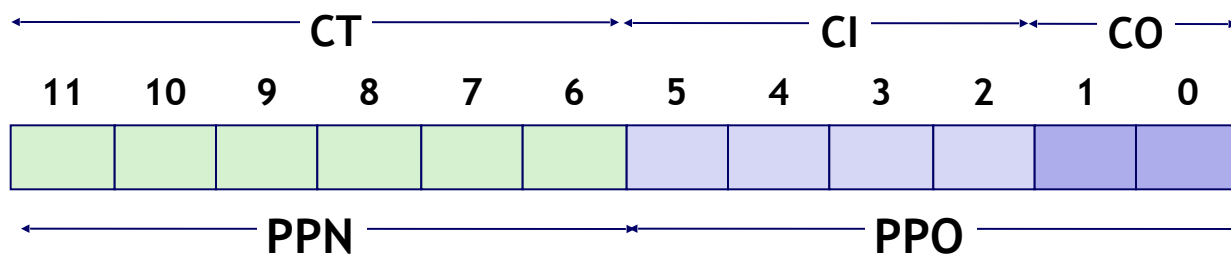
Only show first 16 entries (out of 256)

<i>VPN</i>	<i>PPN</i>	<i>Valid</i>
00	28	1
01	-	0
02	33	1
03	02	1
04	-	0
05	16	1
06	-	0
07	-	0

<i>VPN</i>	<i>PPN</i>	<i>Valid</i>
08	13	1
09	17	1
0A	09	1
0B	-	0
0C	-	0
0D	2D	1
0E	11	1
0F	0D	1

3. Simple Memory System Cache

- 16 lines, 4-byte block size
- Physically addressed
- Direct mapped

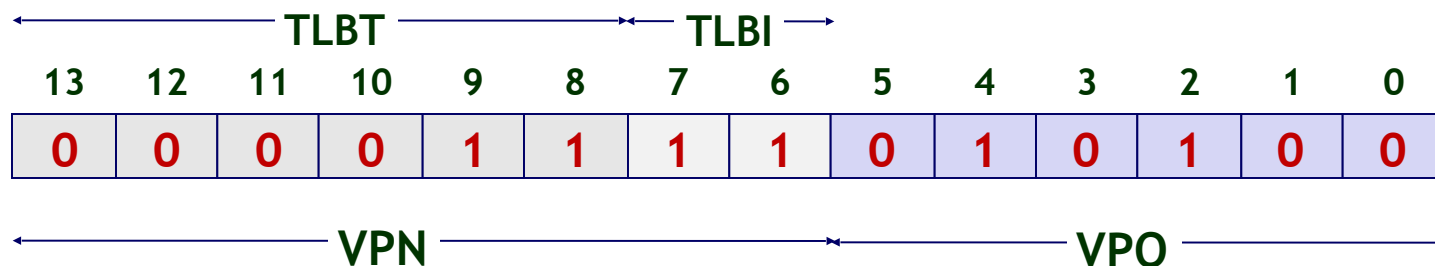


<i>Idx</i>	<i>Tag</i>	<i>Valid</i>	<i>B0</i>	<i>B1</i>	<i>B2</i>	<i>B3</i>
0	19	1	99	11	23	11
1	15	0	-	-	-	-
2	1B	1	00	02	04	08
3	36	0	-	-	-	-
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	-	-	-	-
7	16	1	11	C2	DF	03

<i>Idx</i>	<i>Tag</i>	<i>Valid</i>	<i>B0</i>	<i>B1</i>	<i>B2</i>	<i>B3</i>
8	24	1	3A	00	51	89
9	2D	0	-	-	-	-
A	2D	1	93	15	DA	3B
B	0B	0	-	-	-	-
C	12	0	-	-	-	-
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	-	-	-	-

Address Translation Example #1

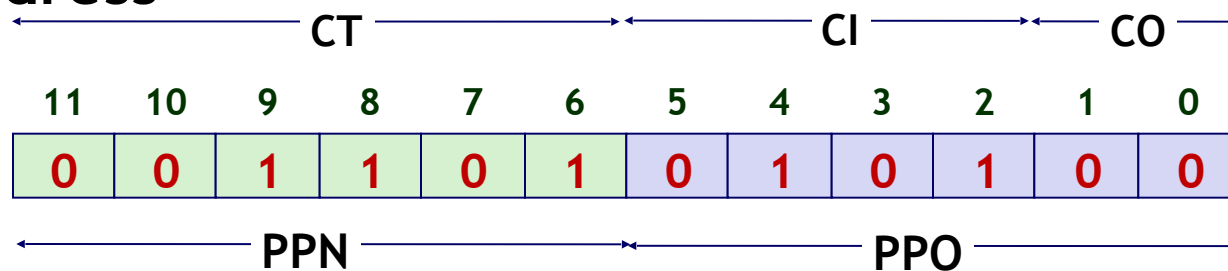
Virtual Address: 0x03D4



VPN 0x0F TLBI 0x3 TLBT 0x03
 PPN: ____

Y TLB Hit? __ Page Fault? 0x0D

Physical Address



CO 0 CI 0x5 CT 0x0D Hit? Y Byte: 0x36

Address Translation Example #2

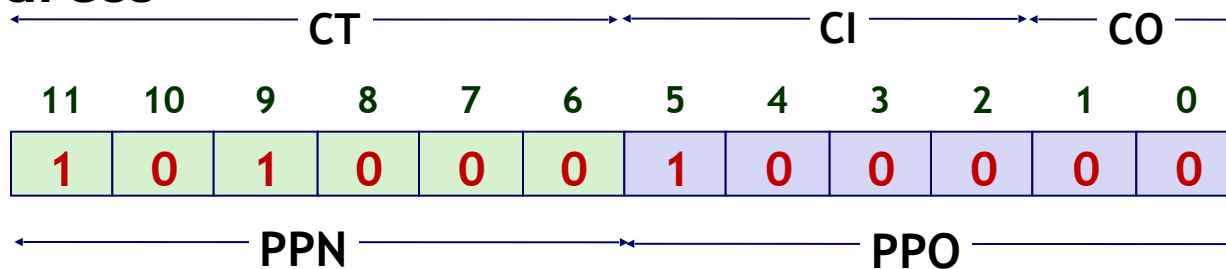
Virtual Address: 0x0020



VPN **0x00** TLBI **0** TLBT **0x00**
PPN: _____

N TLB Hit? **__** Page Fault? **0x28**

Physical Address



CO **0** CI **0x8** CT **0x28** Hit? **N** Mem Byte: _____

Address Translation Example #3

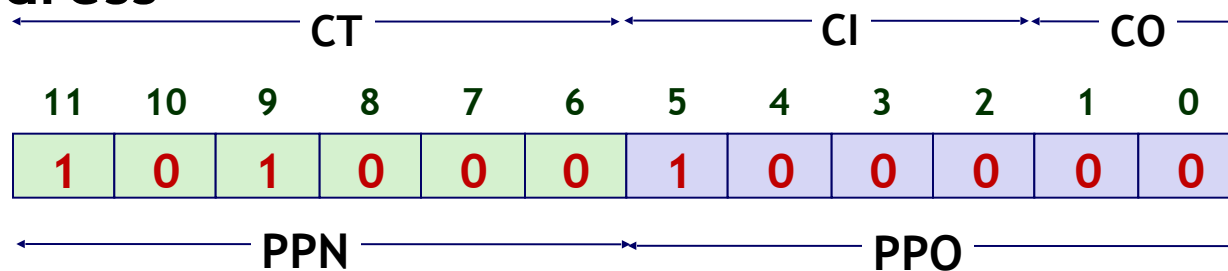
Virtual Address: 0x0020



VPN **0x00** TLBI **0** TLBT **0x00**
PPN: _____

N TLB Hit? **__** Page Fault? **0x28**

Physical Address



CO **0** CI **0x8** CT **0x28** Hit? **N** Mem Byte: _____

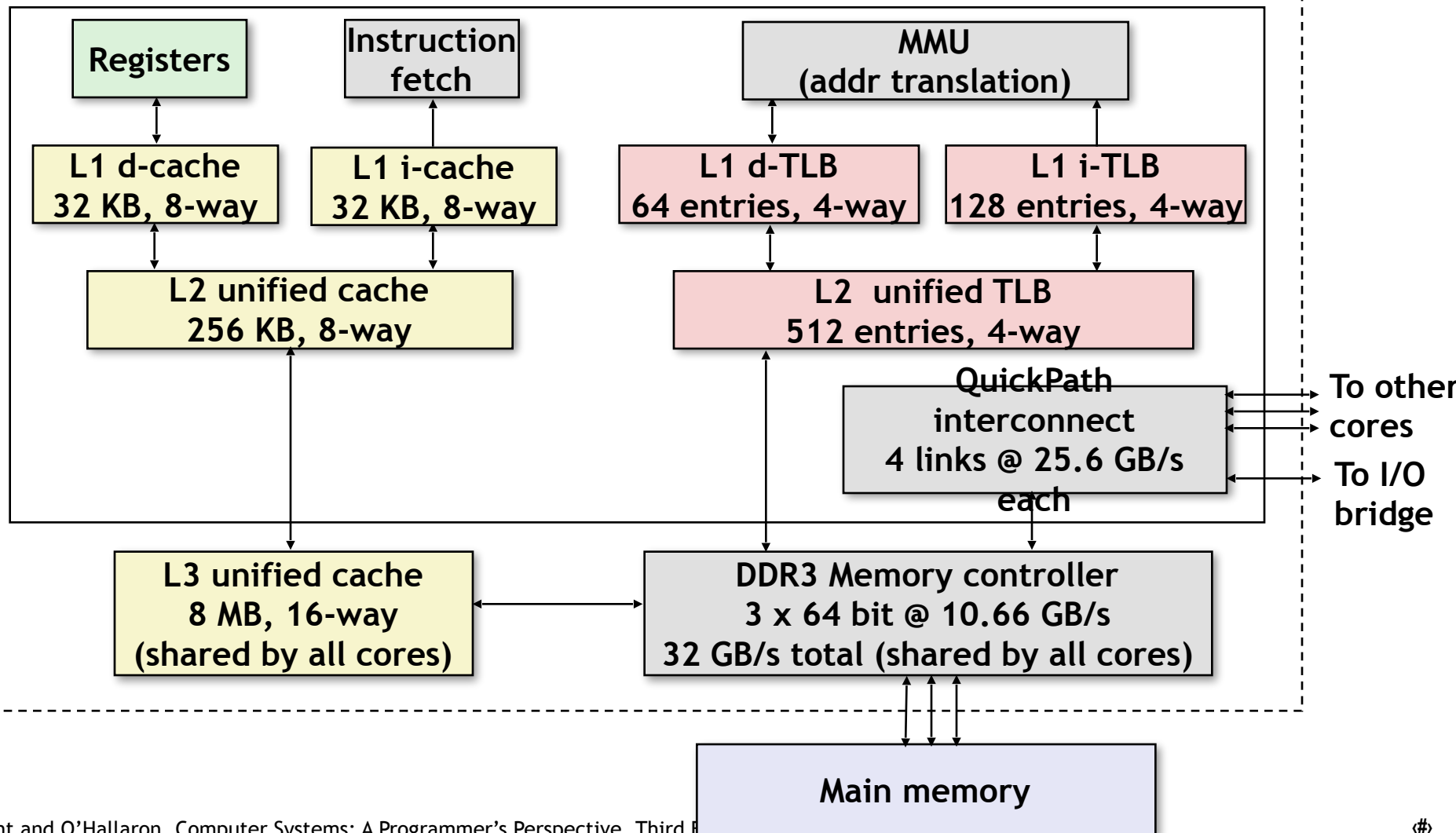
Today

- Simple memory system example
- **Case study: Core i7/Linux memory system**
- Memory mapping

Intel Core i7 Memory System

Processor package

Core x4



Review of Symbols

■ Basic Parameters

- $N = 2^n$: Number of addresses in virtual address space
- $M = 2^m$: Number of addresses in physical address space
- $P = 2^p$: Page size (bytes)

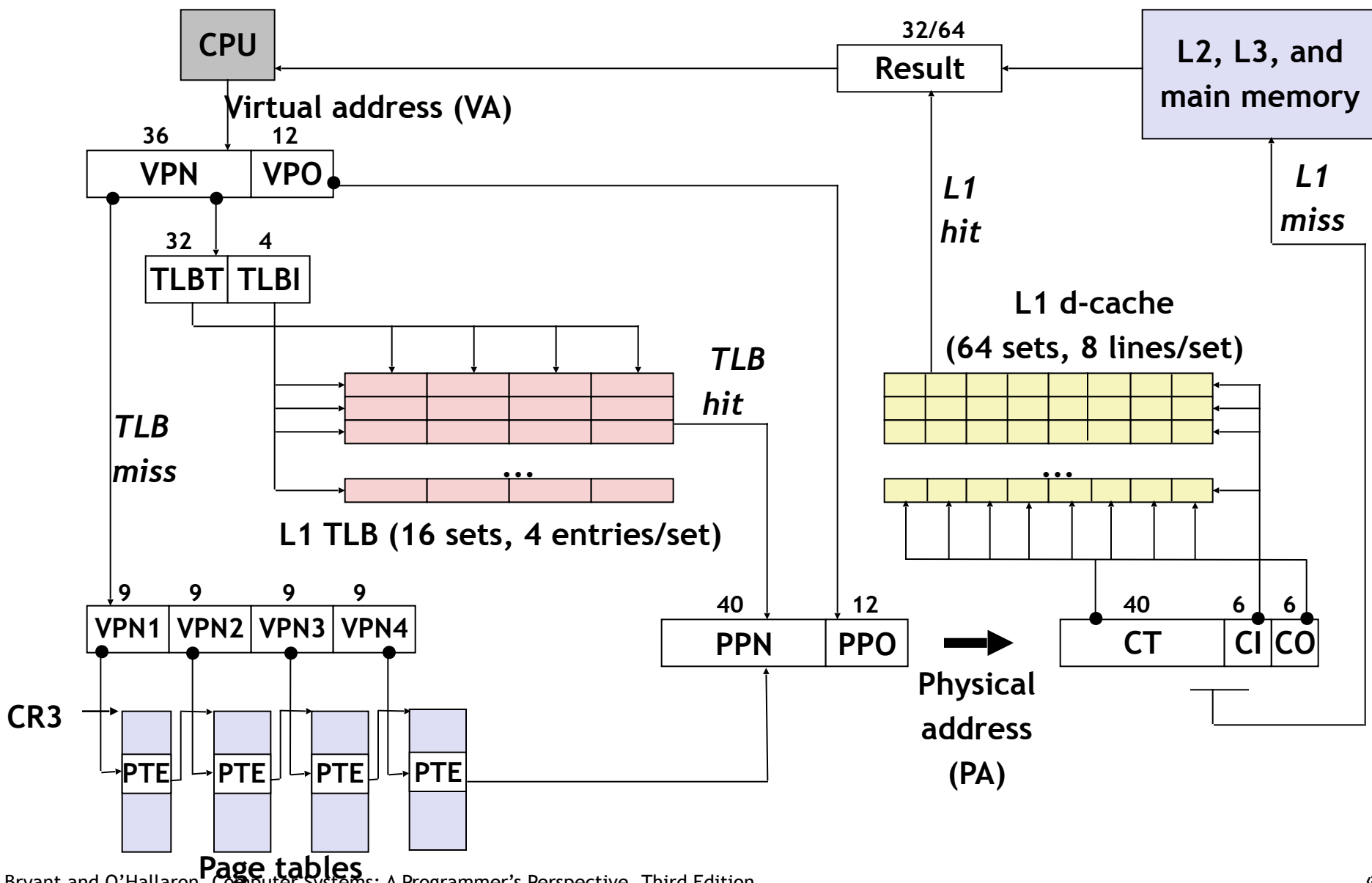
■ Components of the virtual address (VA)

- TLBI: TLB index
- TLBT: TLB tag
- VPO: Virtual page offset
- VPN: Virtual page number

■ Components of the physical address (PA)

- PPO: Physical page offset (same as VPO)
- PPN: Physical page number
- CO: Byte offset within cache line
- CI: Cache index
- CT: Cache tag

End-to-end Core i7 Address Translation



Core i7 Level 1-3 Page Table Entries

63	62	52	51	12	11	9	8	7	6	5	4	3	2	1	0
XD	Unused	Page table physical base address				Unused	G	PS		A	CD	WT	U/S	R/W	P=1

Available for OS (page table location on disk)														P=0
--	--	--	--	--	--	--	--	--	--	--	--	--	--	-----

Each entry references a 4K child page table. Significant fields:

P: Child page table present in physical memory (1) or not (0).

R/W: Read-only or read-write access access permission for all reachable pages.

U/S: user or supervisor (kernel) mode access permission for all reachable pages.

WT: Write-through or write-back cache policy for the child page table.

A: Reference bit (set by MMU on reads and writes, cleared by software).

PS: Page size either 4 KB or 4 MB (defined for Level 1 PTEs only).

Page table physical base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

XD: Disable or enable instruction fetches from all pages reachable from this PTE.

Core i7 Level 4 Page Table Entries

63	62	52	51	12	11	9	8	7	6	5	4	3	2	1	0
XD	Unused	Page physical base address				Unused	G		D	A	CD	WT	U/S	R/W	P=1
Available for OS (page location on disk)															P=0

Each entry references a 4K child page. Significant fields:

P: Child page is present in memory (1) or not (0)

R/W: Read-only or read-write access permission for child page

U/S: User or supervisor mode access

WT: Write-through or write-back cache policy for this page

A: Reference bit (set by MMU on reads and writes, cleared by software)

D: Dirty bit (set by MMU on writes, cleared by software)

Page physical base address: 40 most significant bits of physical page address (forces pages to be 4KB aligned)

XD: Disable or enable instruction fetches from this page.

Core i7 Page Table Translation

