CS 153
Design of Operating Systems

Fall 20

Lecture 19: Locality, Cache, and TLB

Instructor: Chengyu Song

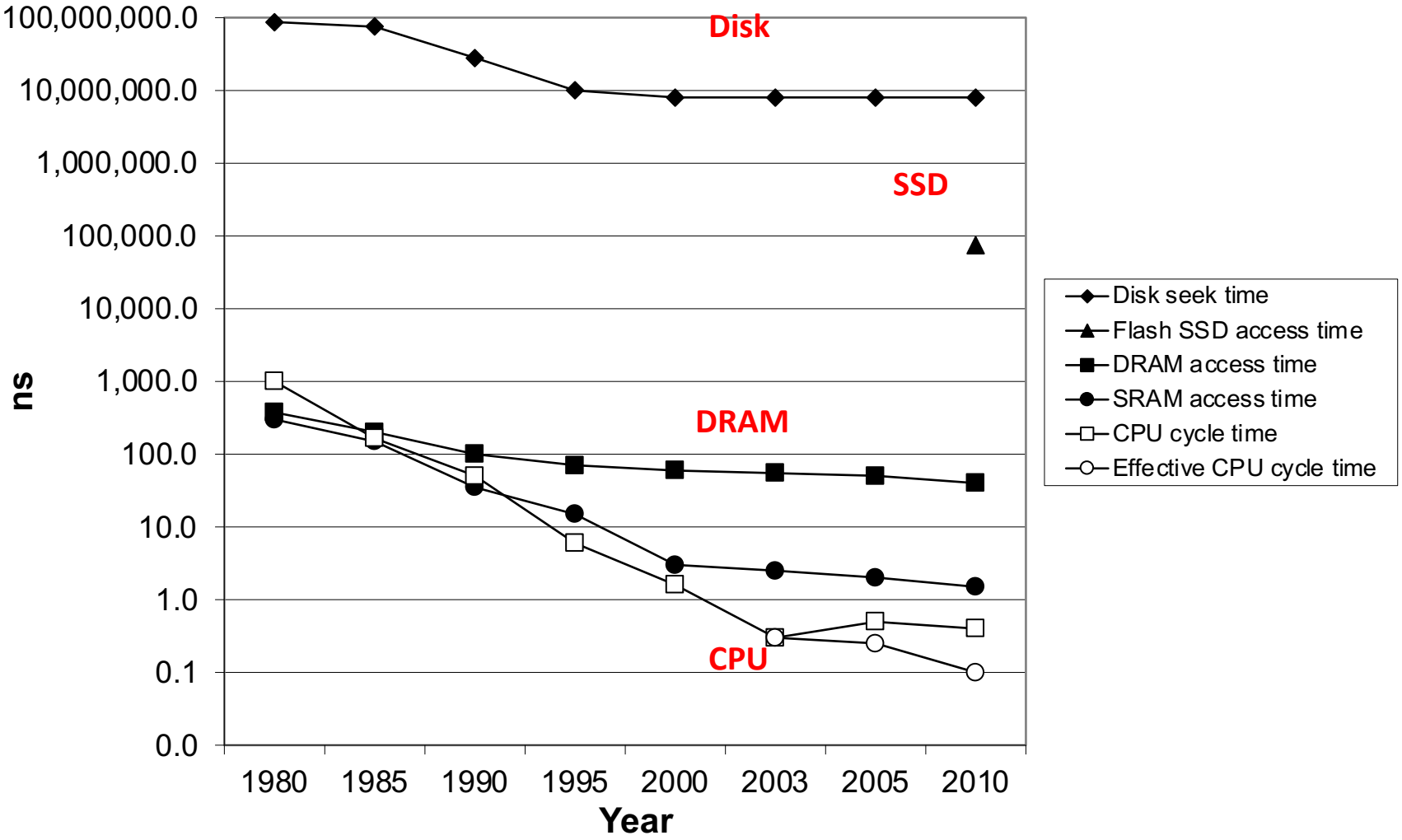
Some slides modified from originals by Dave O'hallaron

Efficient Translations

- Recall that our original page table scheme doubled the latency of doing memory lookups
  - One lookup into the page table, another to fetch the data
- Now two-level page tables triple the latency!
  - Two lookups into the page tables, a third to fetch the data
  - And this assumes the page table is in memory
- How can we use paging but also have lookups cost about the same as fetching from memory?
  - Cache (remember) translations in hardware
  - Translation Lookaside Buffer (TLB)
  - TLB managed by Memory Management Unit (MMU)

The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.



The Price-Speed Gap

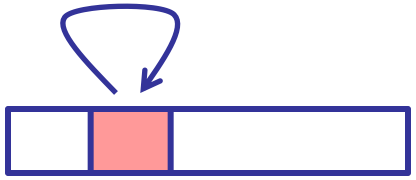
- Question: why don't we just use fast memory to do everything?
- SRAM
  - Latency: 0.5-2.5 ns, cost: ~$5000 per GB
- DRAM
  - Latency: 50-70 ns, cost: ~$20 - $50 per GB
- SSD/NVM
  - Latency: 70-150 ns, cost: ~$4 - $12 per GB
- Magnetic disk
  - Latency: 5-20 ms, cost: ~$0.02 - $2 per GB

Locality to the Rescue!

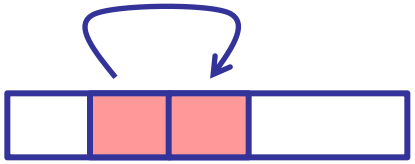
- The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as **locality**

Locality

- Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently



- 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). **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 with respect to 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;
}
```

Locality Example

- Question: Does this function have good locality with respect to array a?

```
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;
}
```

Locality Example

- Question: Can you permute the loops so that the function scans the 3-d array a with a stride-1 reference pattern (and thus has good spatial locality)?

```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

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

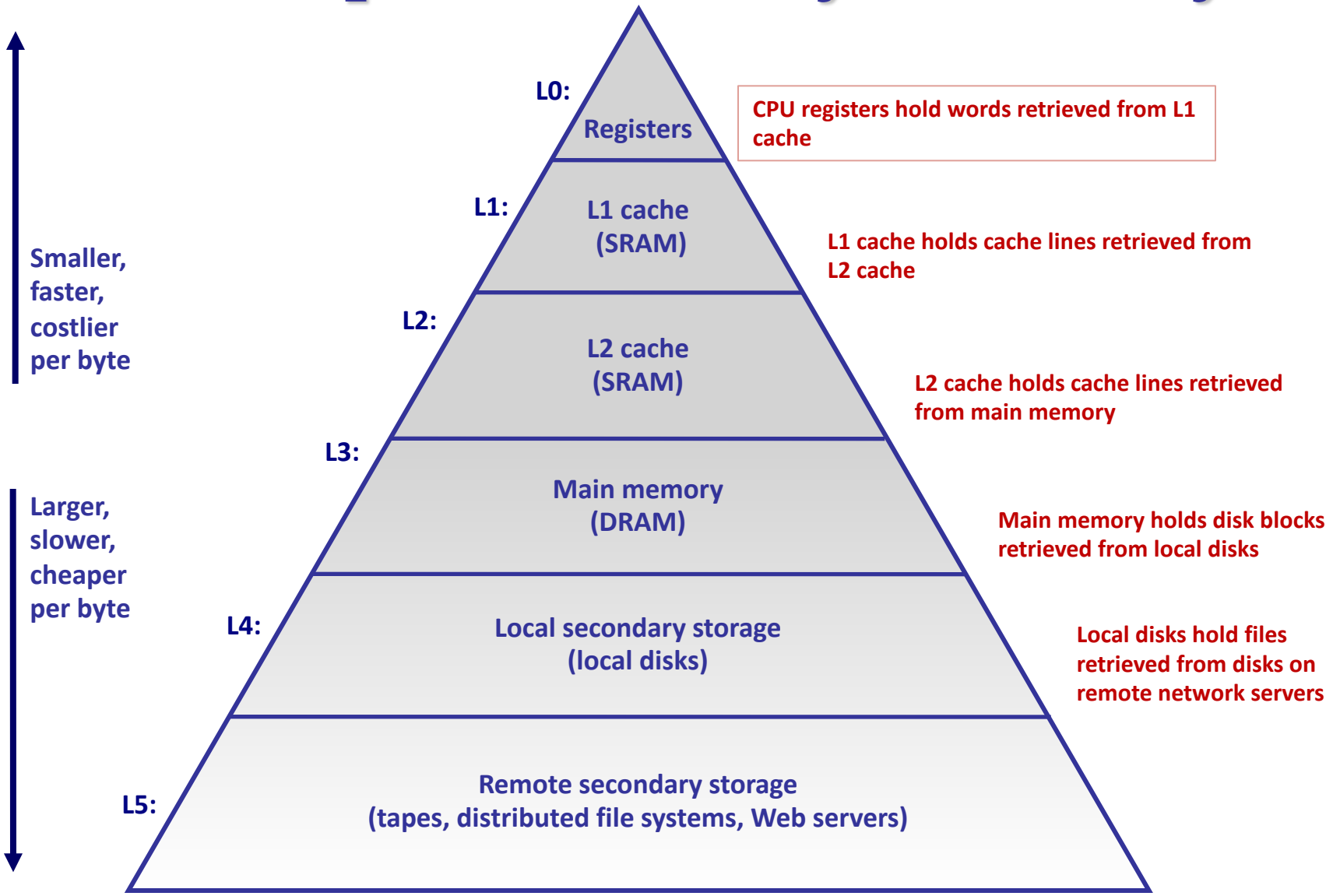
Memory Hierarchies

- Some fundamental and enduring properties of hardware and software:
  - 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.

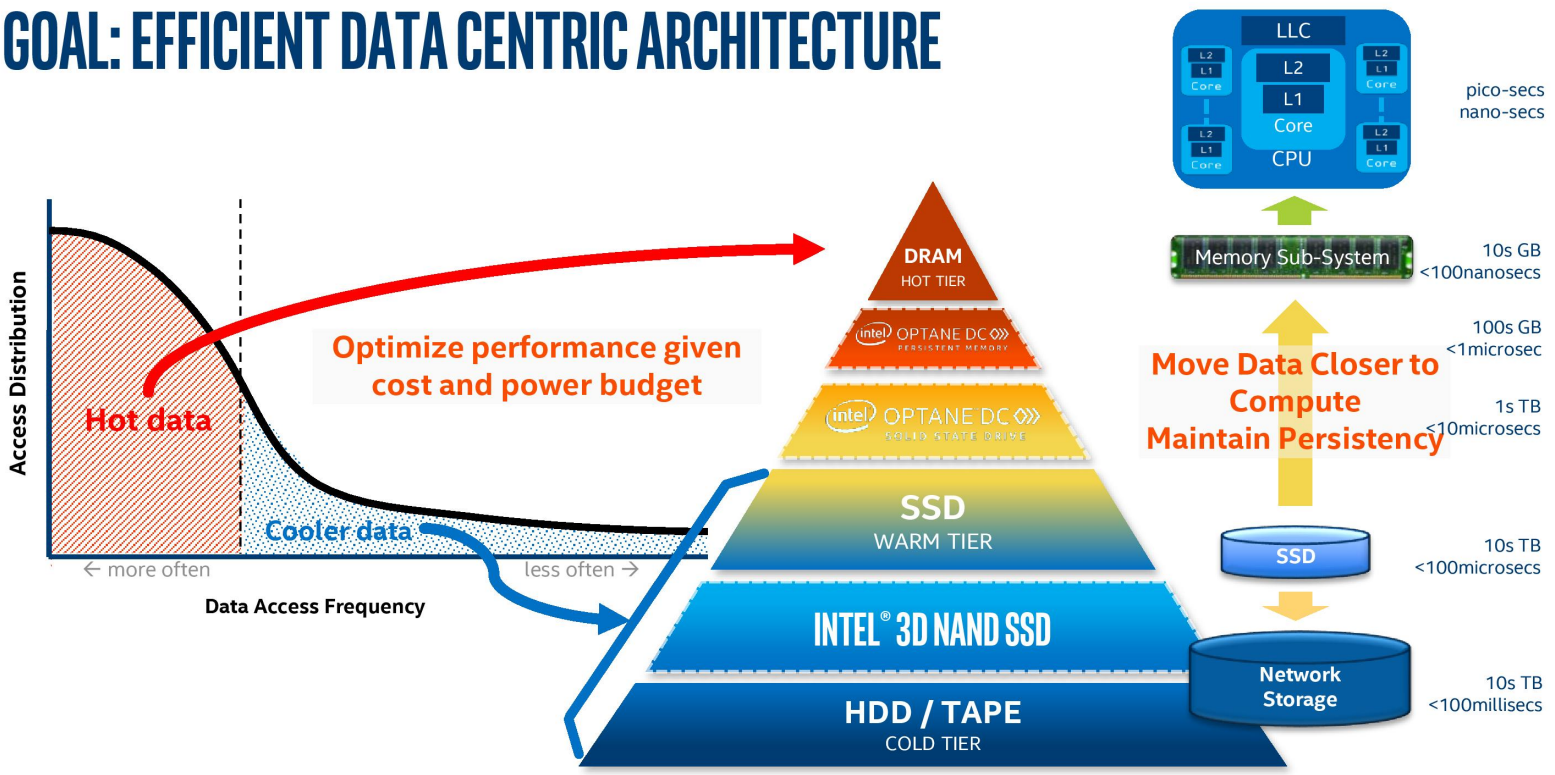
Cache

- **Cache**: A smaller, faster storage that acts as a staging area for a subset of the data in a larger, slower storage.
  - The storage could be a software data structure or a hardware device → memory hierarchy
- Why does cache work?
  - Because of locality!
    - » Hit fast storage much more frequently even though its smaller

An Example of Memory Hierarchy



Another Example



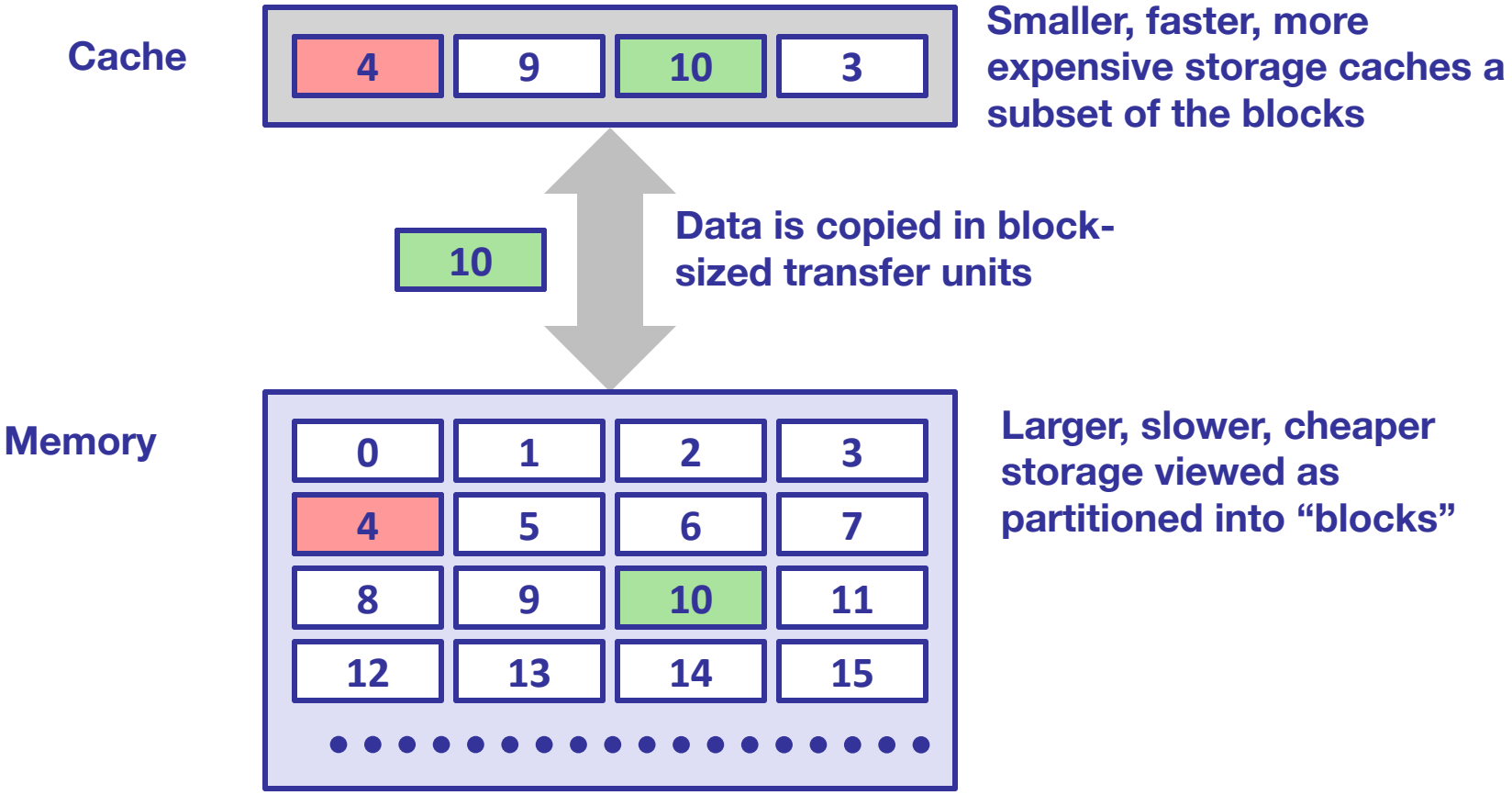
Memory hierarchy

- Fundamental idea of a memory hierarchy:
  - For each layer, faster, smaller device caches larger, slower device
- Why do memory hierarchies work?
  - Because of locality!
    - » Hit fast memory much more frequently even though its smaller
  - Thus, the storage at level k+1 can be slower (but larger and cheaper!)
- *Big Idea:* The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

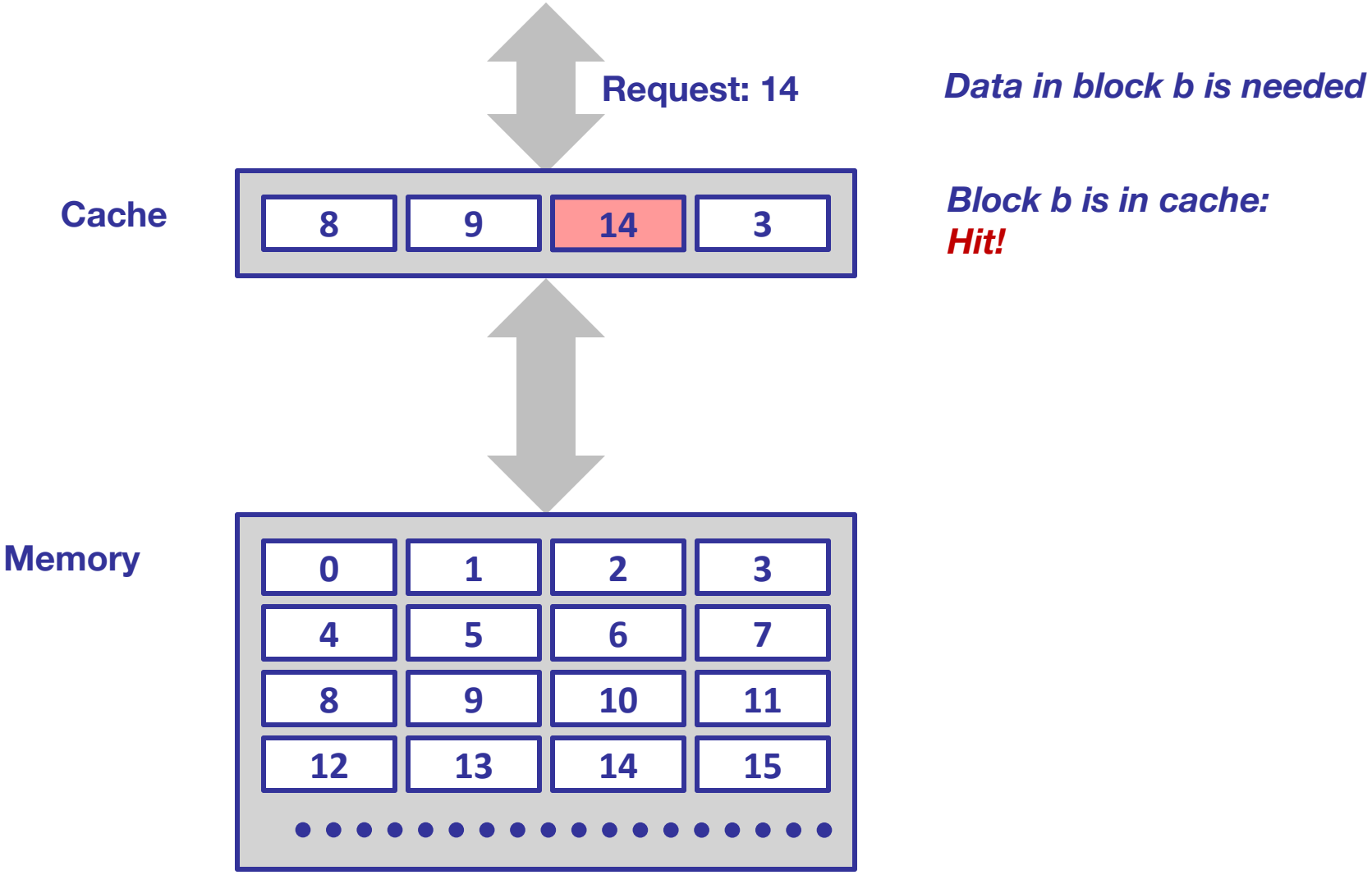
Examples of Caching in the Hierarchy

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
L1 cache	64-bytes block	On-Chip L1	1	Hardware
L2 cache	64-bytes block	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

General Cache Concepts



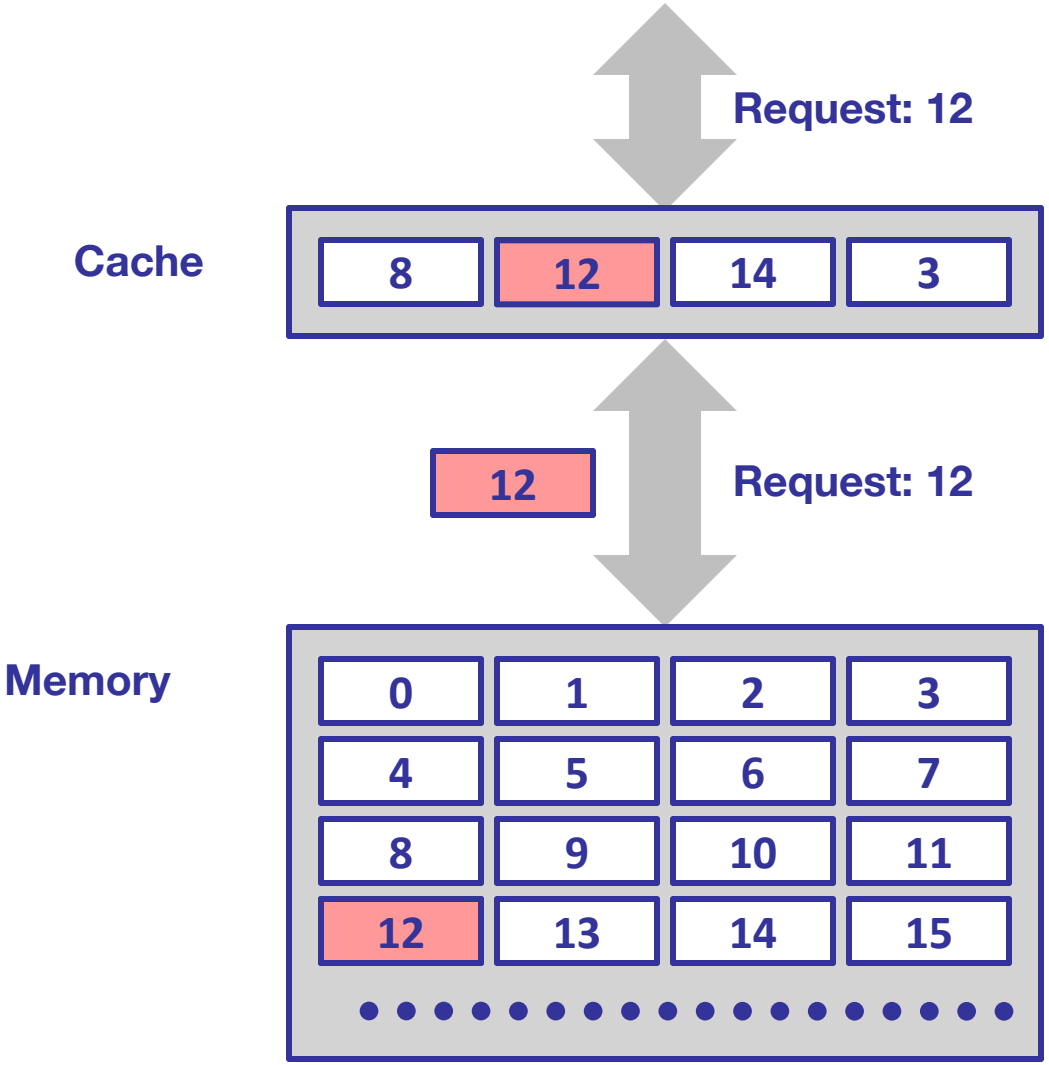
General Cache Concepts: Hit



Data in block b is needed

Block b is in cache:
Hit!

General Cache Concepts: Miss



Data in block b is needed

Block b is not in cache:
Miss!

Block b is fetched from memory

Block b is stored in cache

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

Types of Cache Misses

- Cold (compulsory) miss
  - Cold misses occur because the cache is empty.
- Conflict miss
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
    - » E.g. Block i at level k+1 must be placed in block (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 block.
    - » E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.
- Capacity miss
  - Occurs when the set of active cache blocks (working set) is larger than the cache.

Cache Replacement Policy

- Cache replacement policy: determine which data to remove when we need a victim

- Does it matter?
  - Yes! Cache filling is expensive
  - Getting the number down, can improve the performance of the system significantly

Considerations

- Cache replacement support has to be simple
  - They happen all the time, we cannot make that part slow
- But it can be complicated/expensive when a miss occurs – why?
  - Reason 1: if we are successful, this will be rare
  - Reason 2: when it happens we are paying the cost of loading
    - » Loading from lower layer is relatively slower: can afford to do some extra computation
    - » Worth it if we can save some future miss
- What makes a good cache replacement policy?

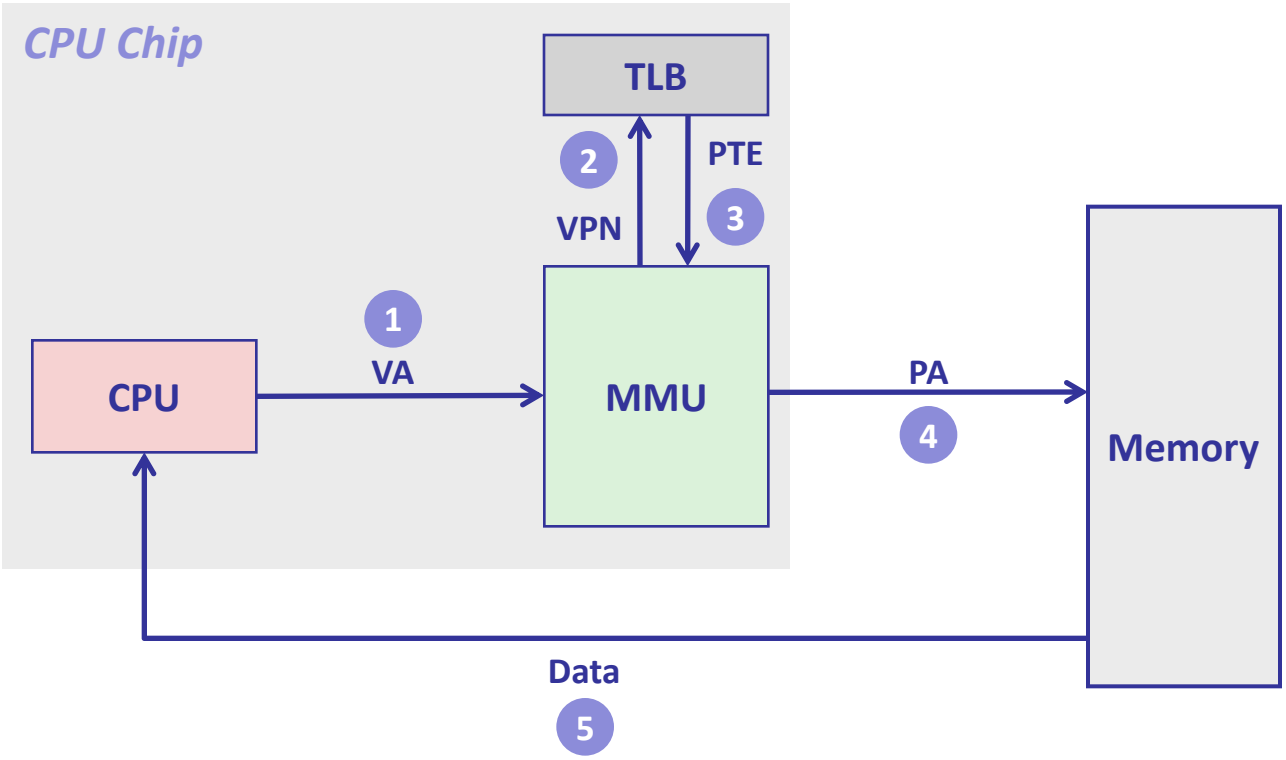
Evicting the Best Data

- Goal is to reduce the cache miss rate
- The best data to evict is the one never touched again
  - Will never have a cache miss on it
- Never is a long time, so picking the data closest to "never" is the next best thing
  - Evicting the data that won't be used for the longest period of time minimizes the number of cache misses
  - Proved by Belady
- We'll survey various replacement algorithms: Belady's, FIFO, LRU (least recently used)

TLBs

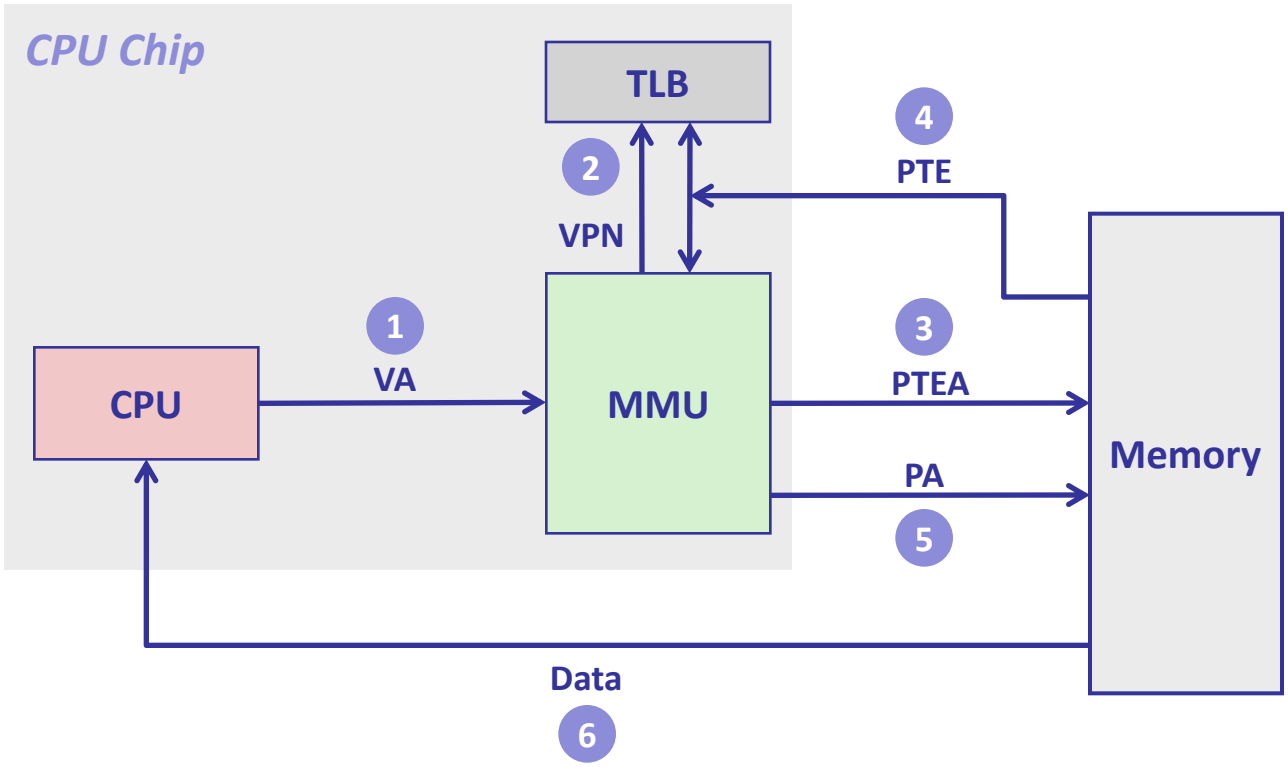
- Translation Lookaside Buffers
  - Actually a cache!
  - Translate virtual page #s into PTEs (not physical addrs)
  - Can be done in a single machine cycle
- TLBs implemented in hardware
  - Fully associative cache (all entries looked up in parallel)
    - » Keys are virtual page numbers
    - » Values are PTEs (entries from page tables)
  - With PTE + offset, can directly calculate physical address

TLB Hit



A TLB hit eliminates one or more memory accesses

TLB Miss



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

Why does TLB help?

- TLB reduces the translation type by remembering/caching the previous translation (page table walk) results
- However, in case of TLB (cache) miss, we still need to do the expensive page table walk
- Fortunately, TLB misses are rare. Why?
  - Because of an interesting property called program **locality**

Managing TLBs

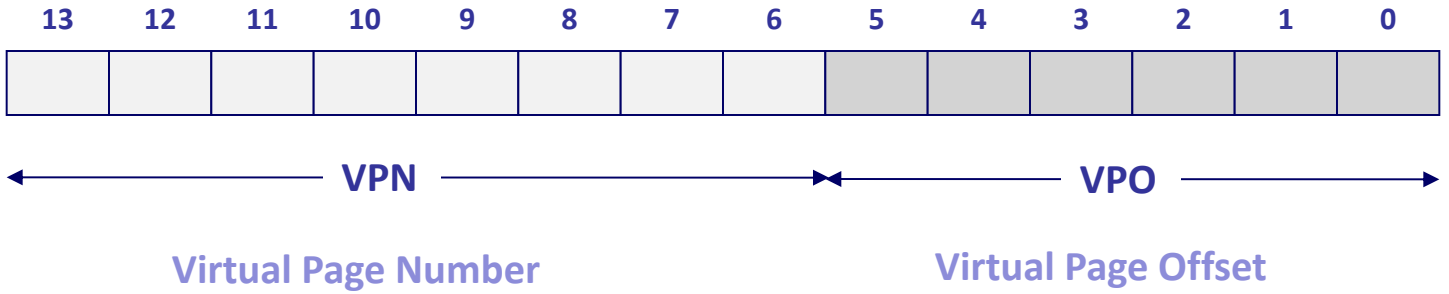
- Hit rate: address translations for most instructions are handled using the TLB
  - > 99% of translations, but there are misses (TLB miss)…
- Who places translations into the TLB (loads the TLB)?
  - Hardware (Memory Management Unit) [x86]
    - » Knows where page tables are in main memory
    - » OS maintains tables, HW accesses them directly
    - » Tables have to be in HW-defined format (inflexible)
  - Software loaded TLB (OS) [MIPS, Alpha, Sparc, PowerPC]
    - » TLB faults to the OS, OS finds appropriate PTE, loads it in TLB
    - » Must be fast (but still 20-200 cycles)
    - » CPU ISA has instructions for manipulating TLB
    - » Tables can be in any format convenient for OS (flexible)

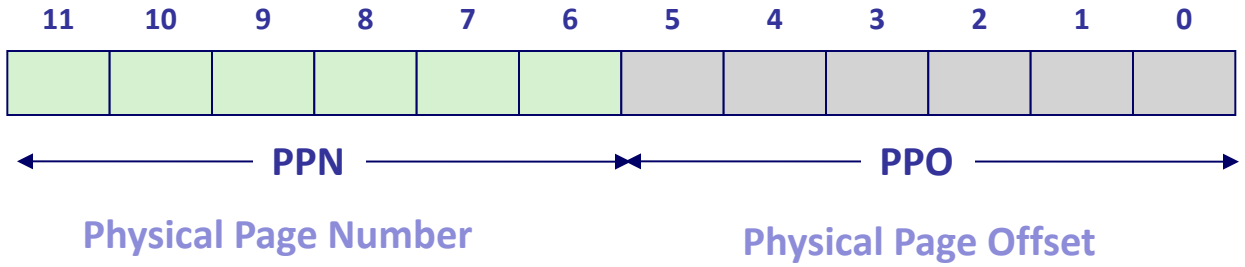
Managing TLBs (2)

- OS ensures that TLB and page tables are consistent
  - When it changes the protection bits of a PTE, it needs to invalidate the PTE if it is in the TLB (special hardware instruction)
- Reload TLB on a process context switch
  - Invalidate all entries
  - Why? Who does it?
- When the TLB misses and a new PTE has to be loaded, an existing PTE must be evicted
  - How? Replacement policy

Simple Memory System Example

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





Simple Memory System Page Table

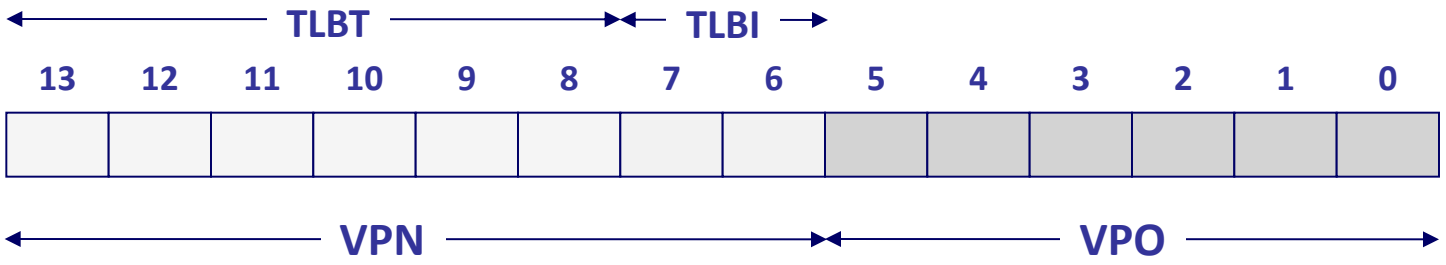
Only show first 16 entries (out of 256)

VPN	PPN	Valid
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

VPN	PPN	Valid
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	11	1
0F	0D	1

Simple Memory System TLB

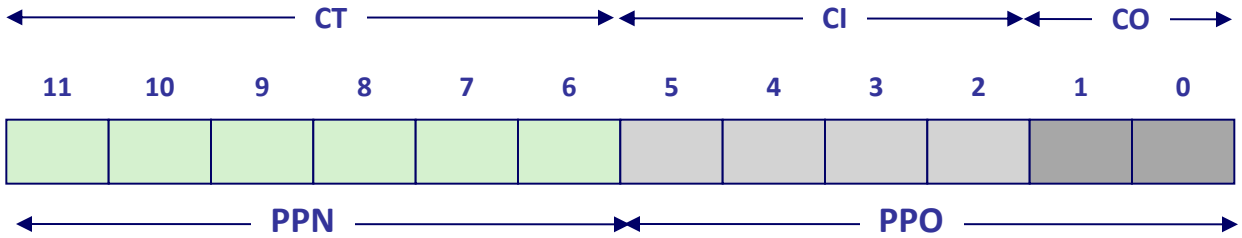
- 16 entries
- 4-way associative* (what is this?!)



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

Simple Memory System Cache

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

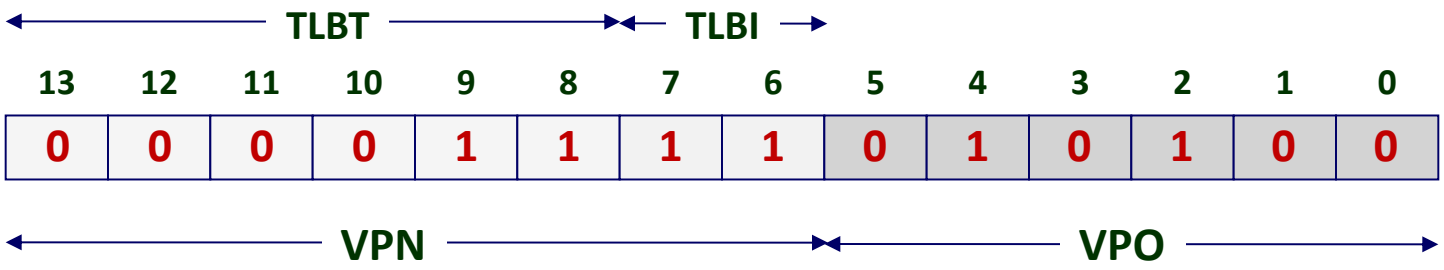


Idx	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

Idx	Tag	Valid	B0	B1	B2	B3
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	–	–	–	–

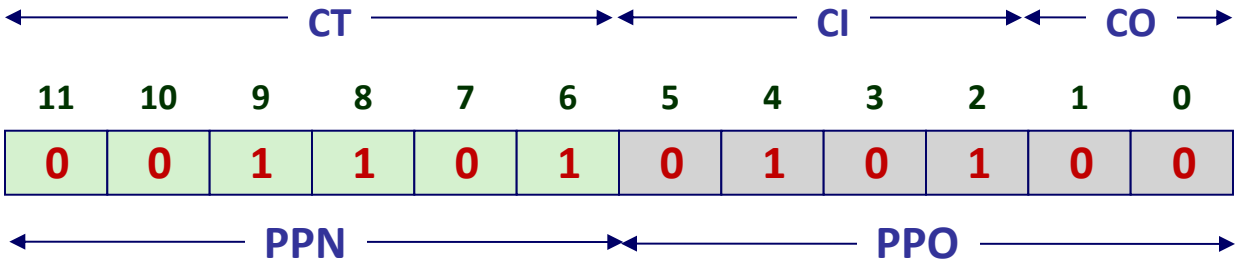
Address Translation Example #1

Virtual Address: `0x03D4`



VPN **0x0F** TLBI **0x3** TLBT **0x03** TLB Hit? **Y** Page Fault? **N** PPN: **0x0D**

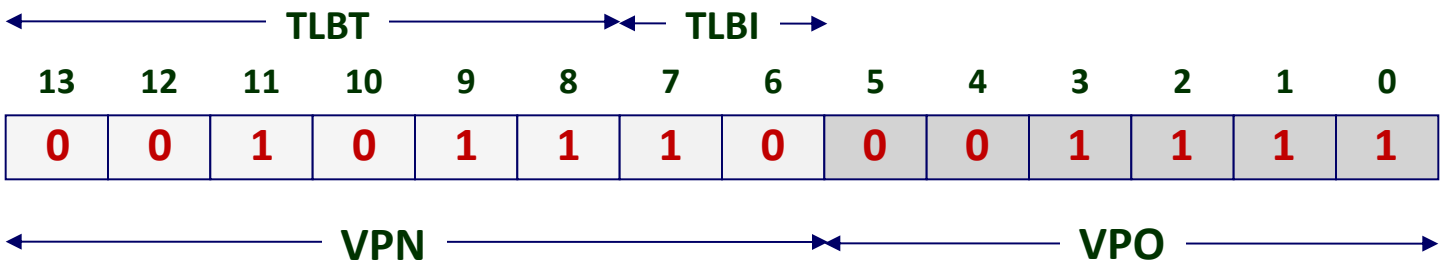
Physical Address



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

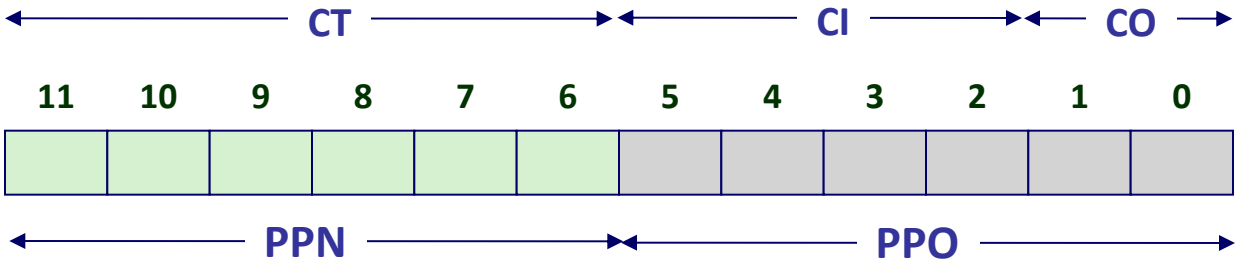
Address Translation Example #2

Virtual Address: `0x0B8F`



VPN 0x2E TLBI 2 TLBT 0x0B TLB Hit? N Page Fault? Y PPN: TBD

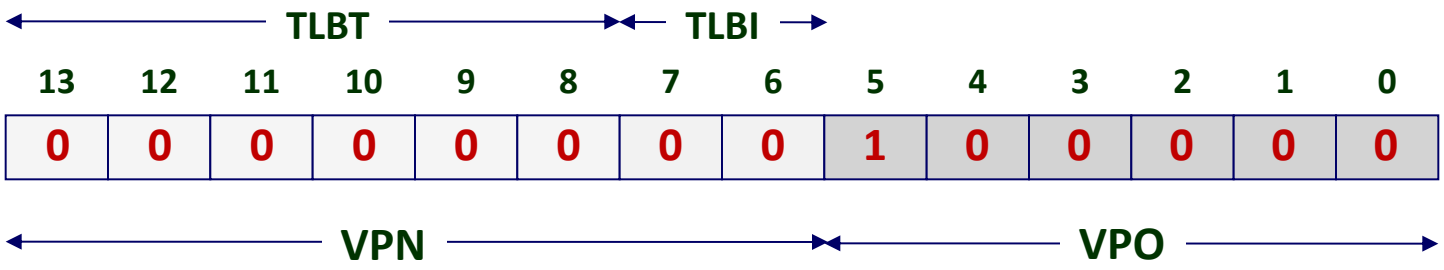
Physical Address



CO ___ CI___ CT ____ Hit? __ Byte: ____

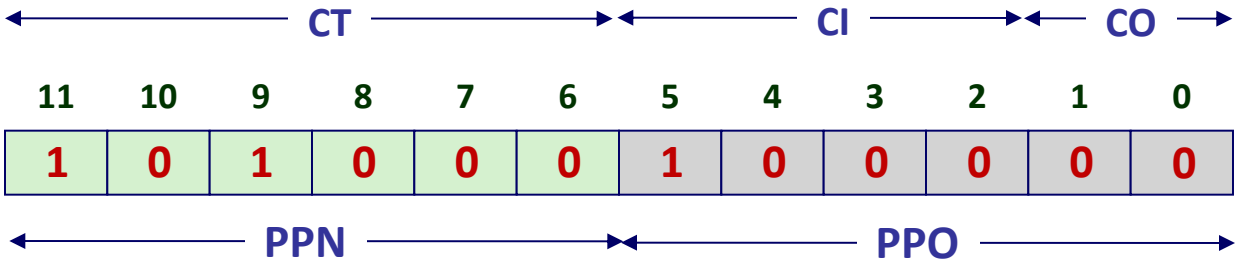
Address Translation Example #3

Virtual Address: 0x0020



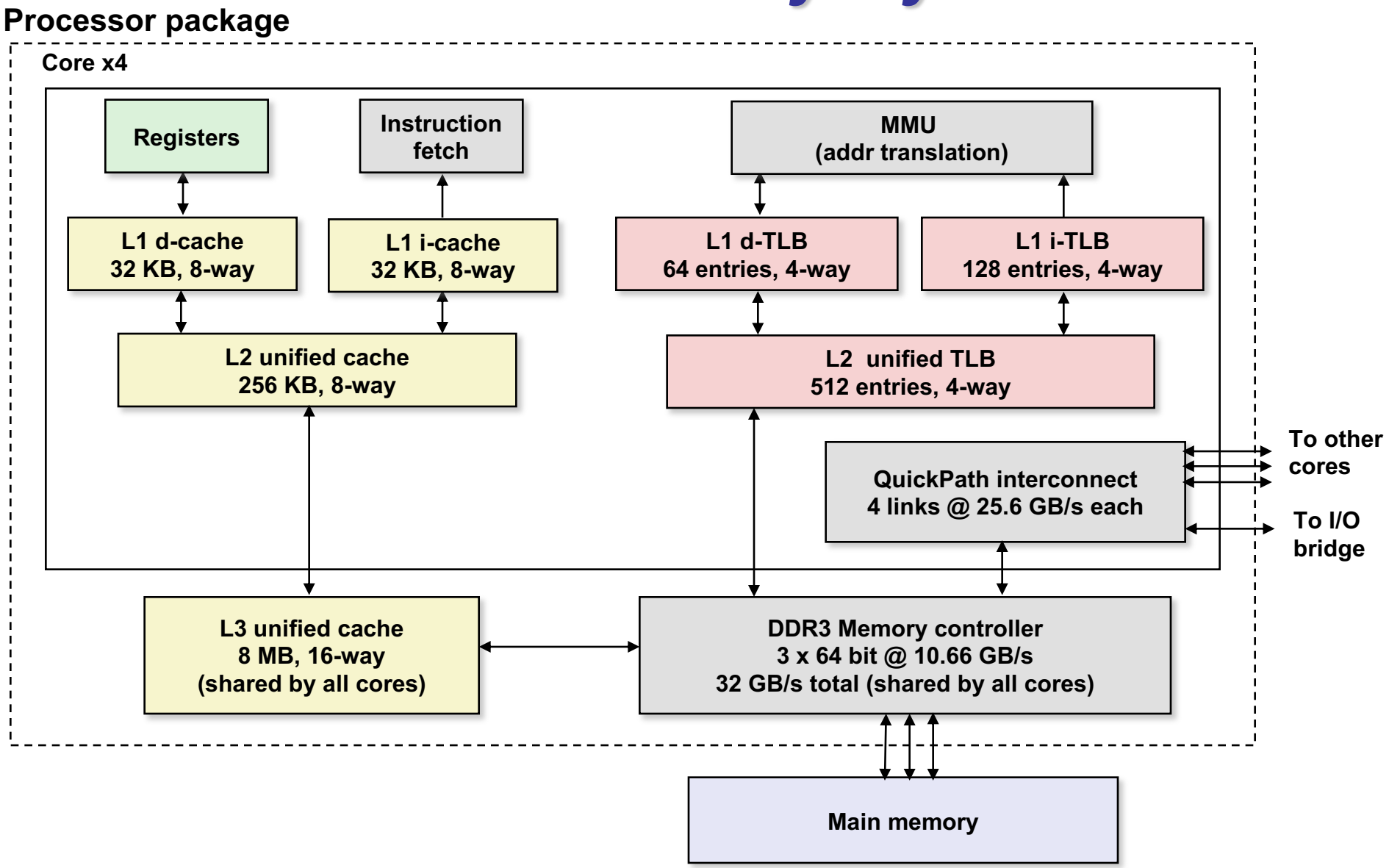
VPN 0x00 TLBI 0 TLBT 0x00 TLB Hit? N Page Fault? N PPN: 0x28

Physical Address



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

Intel Core i7 Memory System



End-to-end Core i7 Address Translation