Translational Lookaside Buffer (TLB)

Recall that the page table is located in memory

1. First trip is to get PFN from VPN
2. Second trip is to get the actual data

Trips to memory are expensive, so we want to optimize this

Translation Lookaside Buffer stores a small portion of the page table in the processor itself

USER/KERNEL	VPN	PFN	VALID/INVALID
U	0	122	V
U	XX	XX	I
U	10	152	V
U	11	170	V
K	0	10	V
K	1	11	V
K	3	15	V
K	XX	XX	I

Figure 8.16: Translation Look-aside Buffer (TLB)

Components (per entry):

- User/Kernel flag
- VPN
- PFN
- Valid/Invalid flag

What happens if the entry isn't in the TLB?

- TLB miss
- If the page table entry hasn't been cached in the TLB, we need to go to the page table in memory
  - We can update the TLB on our way back!

Change in Process?

- Remember that each process has its own specific page table!

- What happens to the TLB when we switch processes?
  - The OS needs to know that the TLB exists
  - We need to flush the TLB entries for all user processes when we context switch

Thrashing

- What happens when overcommit our resources?
- Repeated page faults :(
  - Expensive (multiple trips to memory, plus swapping overhead)
    - Even more if we have to swap_write()
  - We would end up spending more time paging than doing actual work
- Process <> resource trade-off:
  - Less processes: little CPU work done
  - Too many processes: more context switches → greater overhead

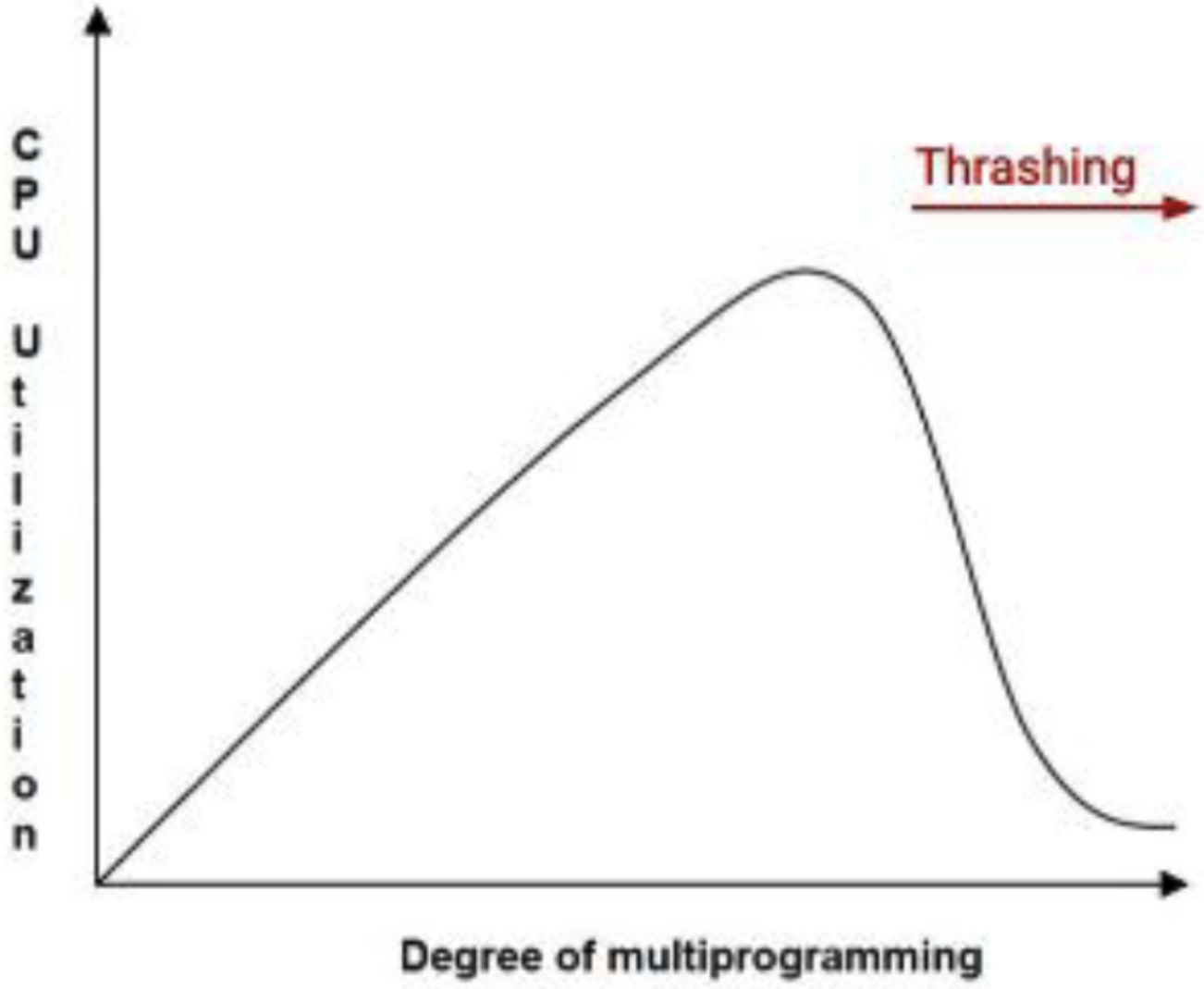


Figure 8.13: CPU Thrashing Phenomenon

Cacheing

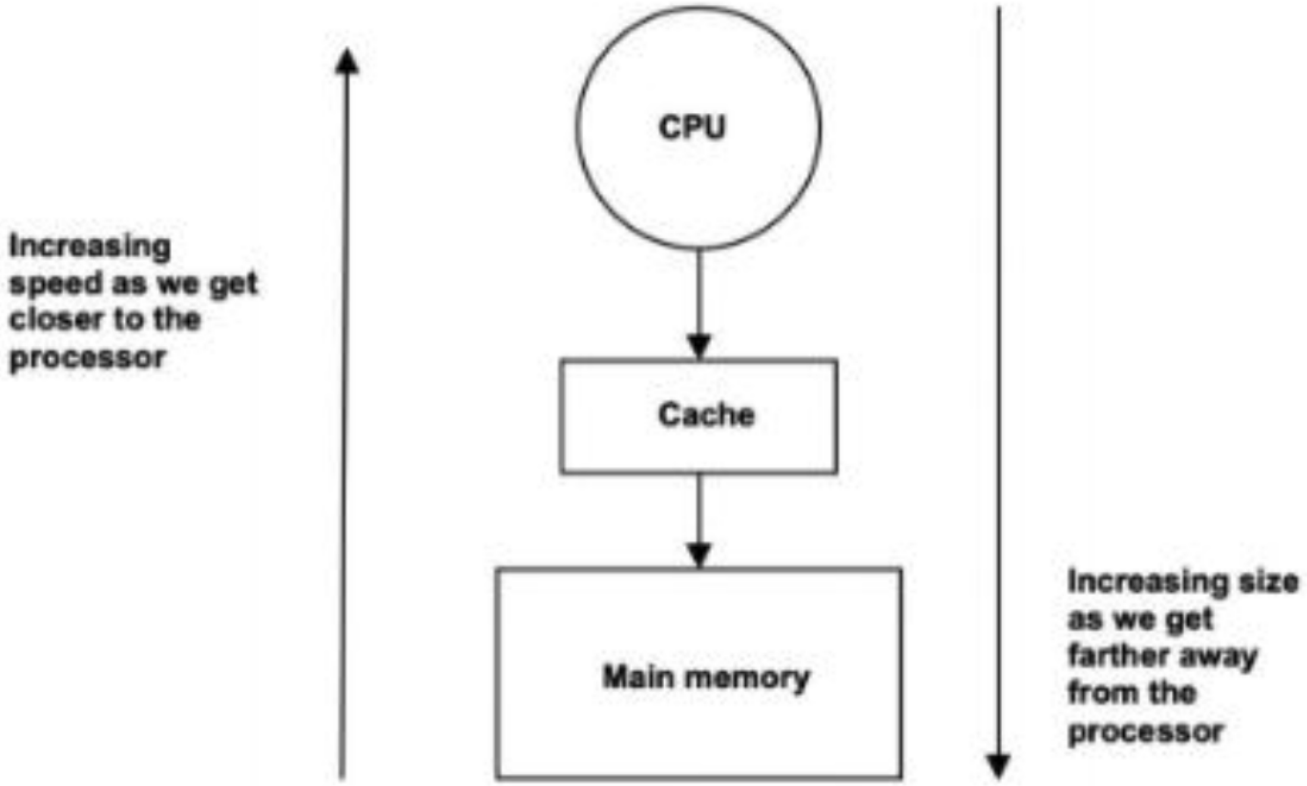
Memory Hierarchy

There are two main types of memory that we'll cover in this class:
SRAM	DRAM
very fast, difficult to scale	kinda slow, easy to scale
examples: TLB, register	main memory RAM

Key concept: you can't have both size and speed!

Best of Both Worlds

- Between the CPU and memory, include caches of SRAM that we can check and access quickly
- Use main memory (DRAM) as a last resort if requested value is not in SRAM

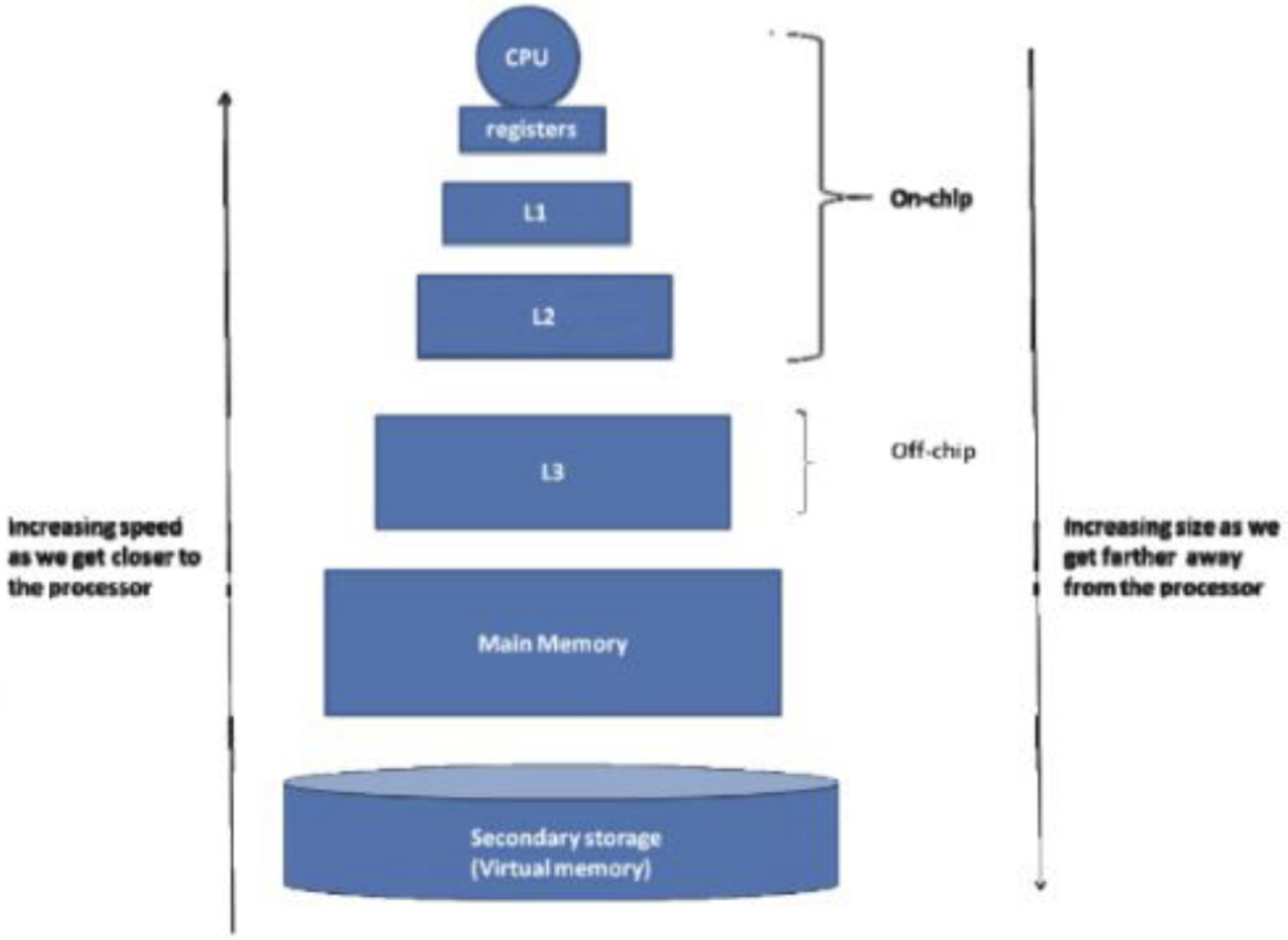


Some Definitions

Term	Defintion
Caching	General Concept of keeping a smaller stash of items closer to the point of use than the permanent location
Spatial Locality	We will likely access memory addresses near the ones we just accessed
Temporal Locality	If we access something, we will likely access it again soon
Hit	When we look for something in the cache and it's there
Miss	When we look for something in the cache and it's NOT there
Miss penalty	Time penalty for a miss (to go get the right value and update the cache)
Effective Memory Access Time (EMAT)	The effective time CPU sees to access memory (including cache access)

Multi-Level Cacheing

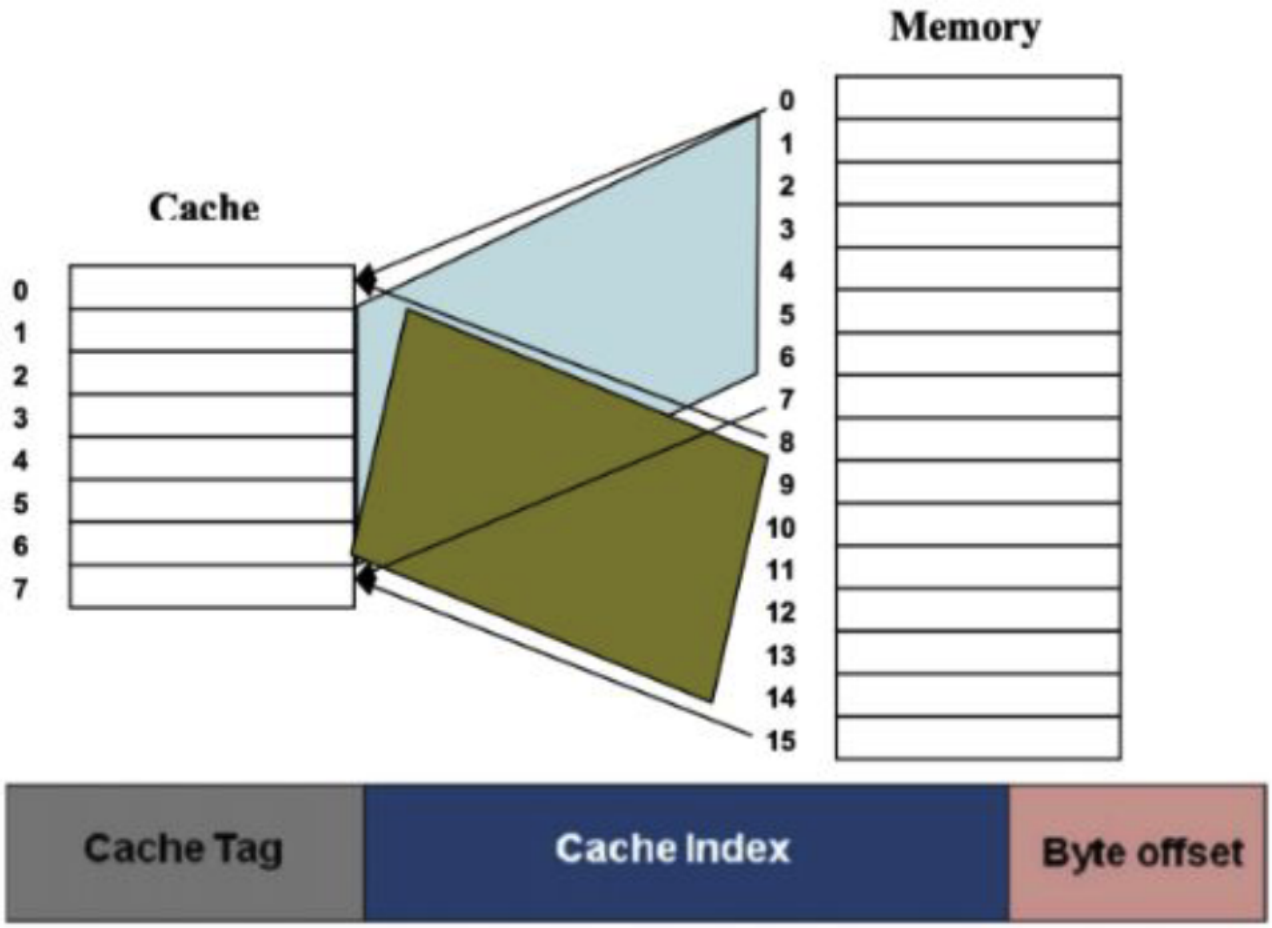
- Multiple levels of caches, each of them progressively bigger and slower
- Can calculate EMAT for each level of caching i with access time T
  - $EMAT = T_{,} + m \cdot T_{i+1}$



Direct Mapped Cache

- One-to-one correspondence between memory and cache locations
- Map each memory address to address % cache size
- Cache tag:
  - Differentiate between two memory locations mapping to the same index in cache
  - To look up a value in the cache, we need a tag and check to see if `cache[tag] == index`
  - If valid, get the data
- Cache index:
  - The specific location in the cache

    - Memory address % cache size



Types of Misses

- Compulsory Misses
    - Misses because the data has never loaded into the cache before
    - Trying to retrieve memory location 8 for the very first time
- Conflict Misses
    - Misses because the cache entry has been overridden by another memory location
    - There might be unused spots in the cache
    - Example: 15 and 7 map to the same location, so if 7 is put in the cache, and 15 overrides 7 in the cache, we will have a conflict miss when trying to get 15
- Capacity Misses
    - All spots in the cache are taken
    - Unable to find data without a conflict

If there is more than one type of miss present when checking for and adding something to the cache, we will prefer a compulsory miss over another kind of miss.

Direct Mapped Cache

Direct mapped caching does not seem to be the best approach to caching

- There were lots of misses and swapping
- There's a lot of overhead due to the metadata that we need to store

We can resolve some of the issues

- Pick a bigger cache table
- Create a hardware solution to add a byte offset from CPU → address to store multiple bytes within the cache entry
    - Allows for us to abuse the principle of locality in that one load into the cache → multiple accesses

TLB	Page Table	Cache	Possible?	When?
H	H	M	Y	TLB hit, so the page table is never really checked
H	M	H	N	Page can't be in TLB if it isn't in memory
H	M	M	N	Page can't be in TLB if it isn't in memory
M	H	H	Y	TLB miss, retrieve from page table and update TLB
M	H	M	Y	TLB miss, retrieve from page table and update TLB
M	M	H	N	Page can't be in cache if it isn't in memory
M	M	M	Y	TLB miss → page fault → after retrying, miss in cache