Hierarchical Page Tables

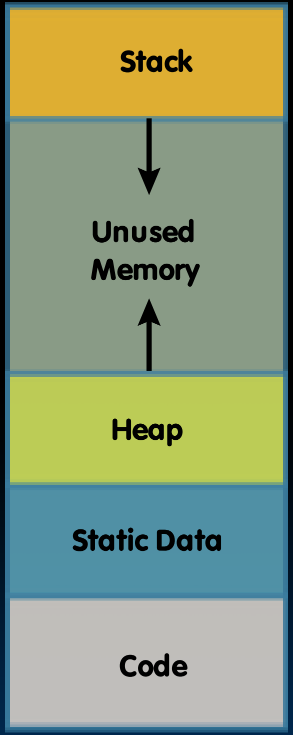
Size of Page Tables

- E.g., 32-Bit virtual address, 4-KiB pages
  - Single page table size:
    - 4 x 2^{20} Bytes = 4-MiB
    - 0.1% of 4-GiB memory
  - Total size for 256 processes (each needs a page table)
    - 256 x 4 x 2^{20} Bytes = 256 x 4-MiB = 1-GiB
    - 25% of 4-GiB memory!
- What about 64-bit addresses?

How can we keep the size of page tables "reasonable"?

Options for Page Tables

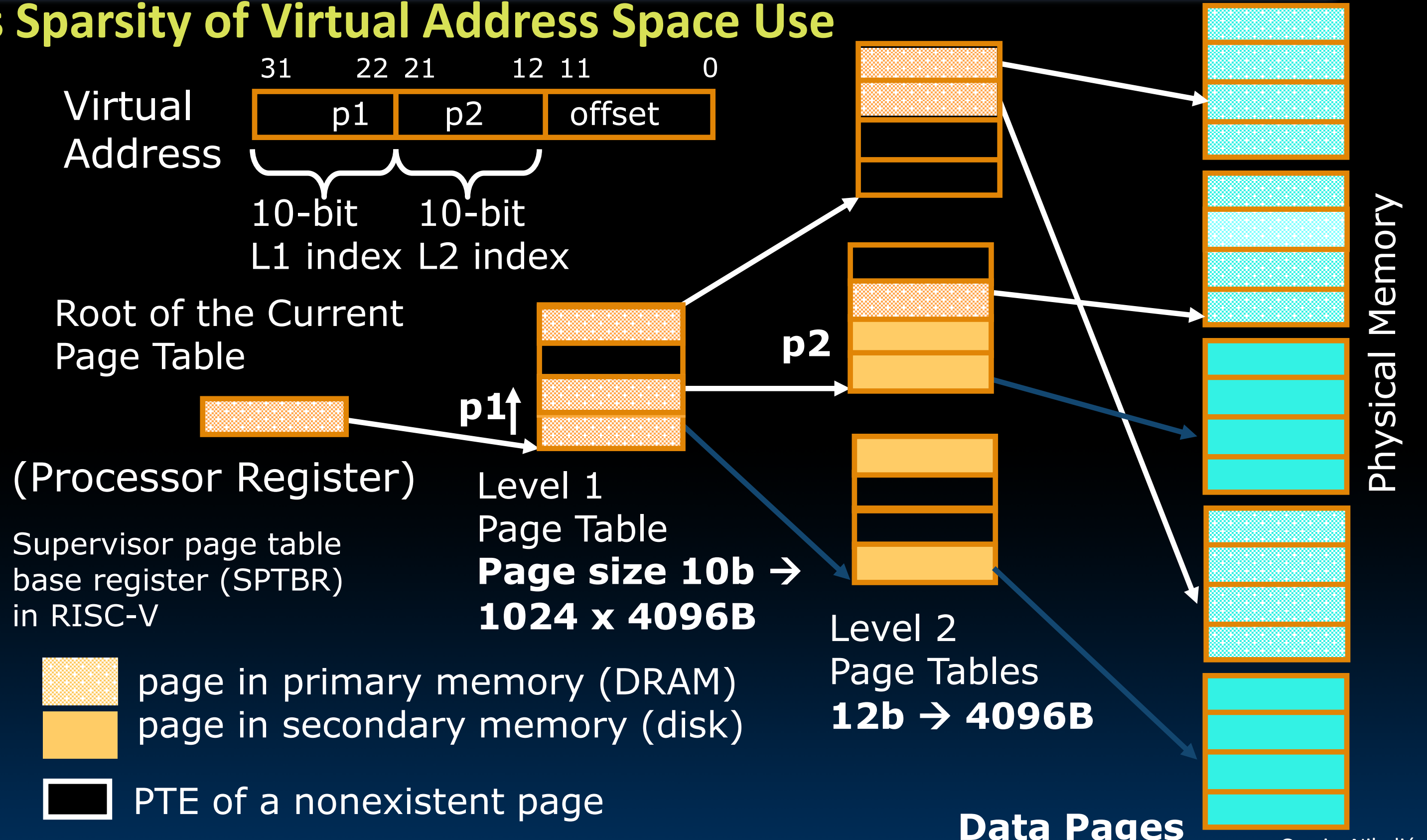
- Increase page size
  - E.g., doubling page size cuts PT size in half
  - At the expense of potentially wasted memory
- Hierarchical page tables
  - With decreasing page size
- Most programs use only fraction of memory
  - Split PT in two (or more) parts
  - This is done in RISC-V



Garcia, Nikolić

Hierarchical Page Table

Exploits Sparsity of Virtual Address Space Use



Virtual Address

10-bit L1 index

10-bit L2 index

Root of the Current Page Table

(Processor Register)

Supervisor page table base register (SPTBR) in RISC-V

Level 1 Page Table

Page size 10b → 1024 x 4096B

Level 2 Page Tables

12b → 4096B

Data Pages

▨ page in primary memory (DRAM)

🟨 page in secondary memory (disk)

□ PTE of a nonexistent page

Garcia, Nikolić

Example: 32-b RISC-V

Virtual address (32 bits)	VPN[1]	VPN[0]	offset
	10	10	12
Physical address(34 bits)	PPN[1]	PPN[0]	offset
	12	10	12

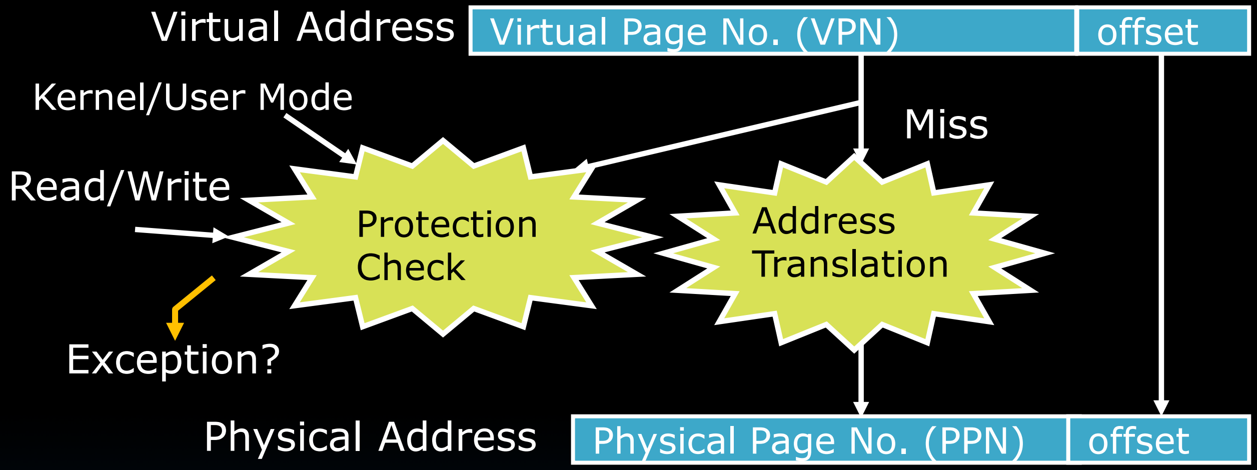
- VPN: Virtual Page Number
- PPN: Physical Page Number
- Page Table Entry (PTE) is 32b and contains:
  - PPN[1], PPN[0]
  - Status bits for protection and usage (read, write, exec), validity, etc.

PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
12	10	2	1	1	1	1	1	1	1	1

R= 0, W=0, X = 0 points to next level page table; otherwise it is a leaf PTE

Translation Lookaside Buffers

Address Translation and Protection



- Every instruction and data access needs address translation and protection checks

Good VM design should be fast (~one cycle) and space efficient

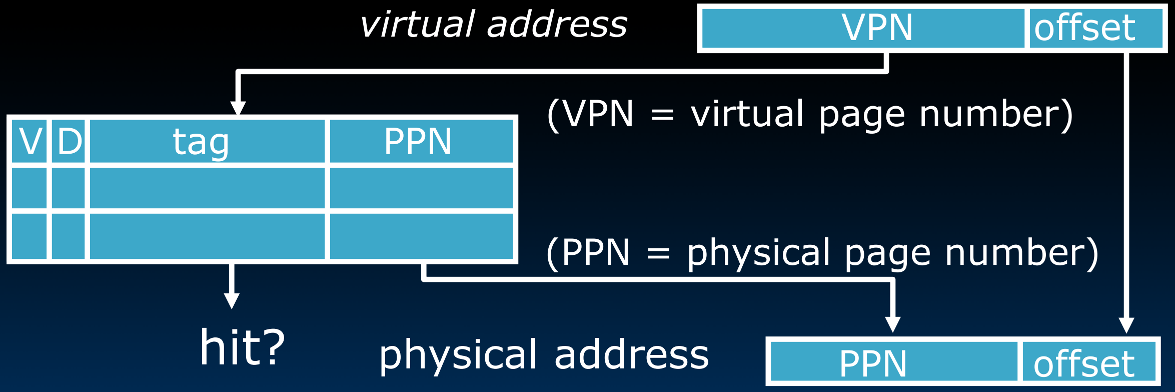
Translation Lookaside Buffers (TLB)

Address translation is very expensive!

- In a single-level page table, each reference becomes two memory accesses
- In a two-level page table, each reference becomes three memory accesses

Solution: *Cache some translations in TLB*

TLB hit	→ *Single-Cycle Translation*
TLB miss	→ *Page-Table Walk to refill*



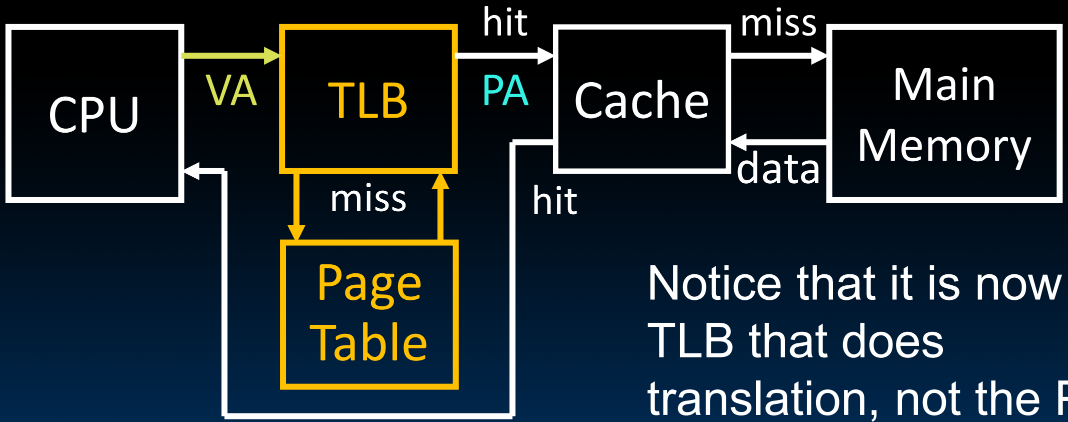
Garcia, Nikolić

TLB Designs

- Typically 32-128 entries, usually fully associative
  - Each entry maps a large page, hence less spatial locality across pages → more likely that two entries conflict
  - Sometimes larger TLBs (256-512 entries) are 4-8 way set-associative
  - Larger systems sometimes have multi-level (L1 and L2) TLBs
- Random or FIFO replacement policy
- "TLB Reach": Size of largest virtual address space that can be simultaneously mapped by TLB

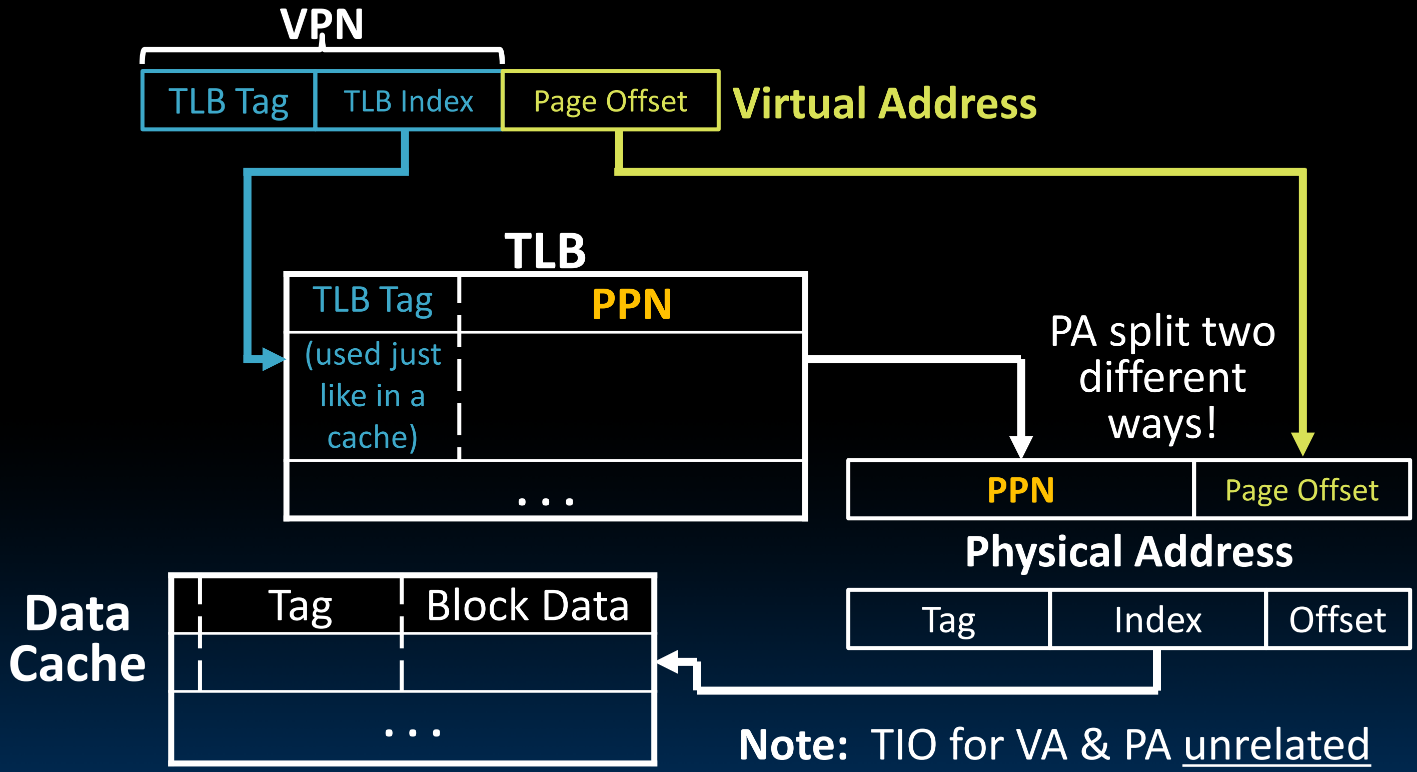
Where Are TLBs Located?

- Which should we check first: Cache or TLB?
  - Can cache hold requested data if corresponding page is not in physical memory? **No**
  - With TLB first, does cache receive VA or PA? **PA**



Notice that it is now the TLB that does translation, not the Page Table!

Address Translation Using TLB

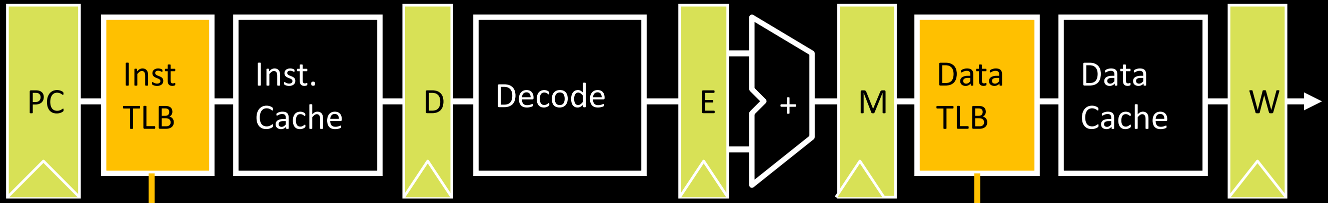


Note: TIO for VA & PA unrelated

Garcia, Nikolić

TLBs in Datapath

VM-related Events in Pipeline



TLB miss? Page Fault? Protection violation?

TLB miss? Page Fault? Protection violation?

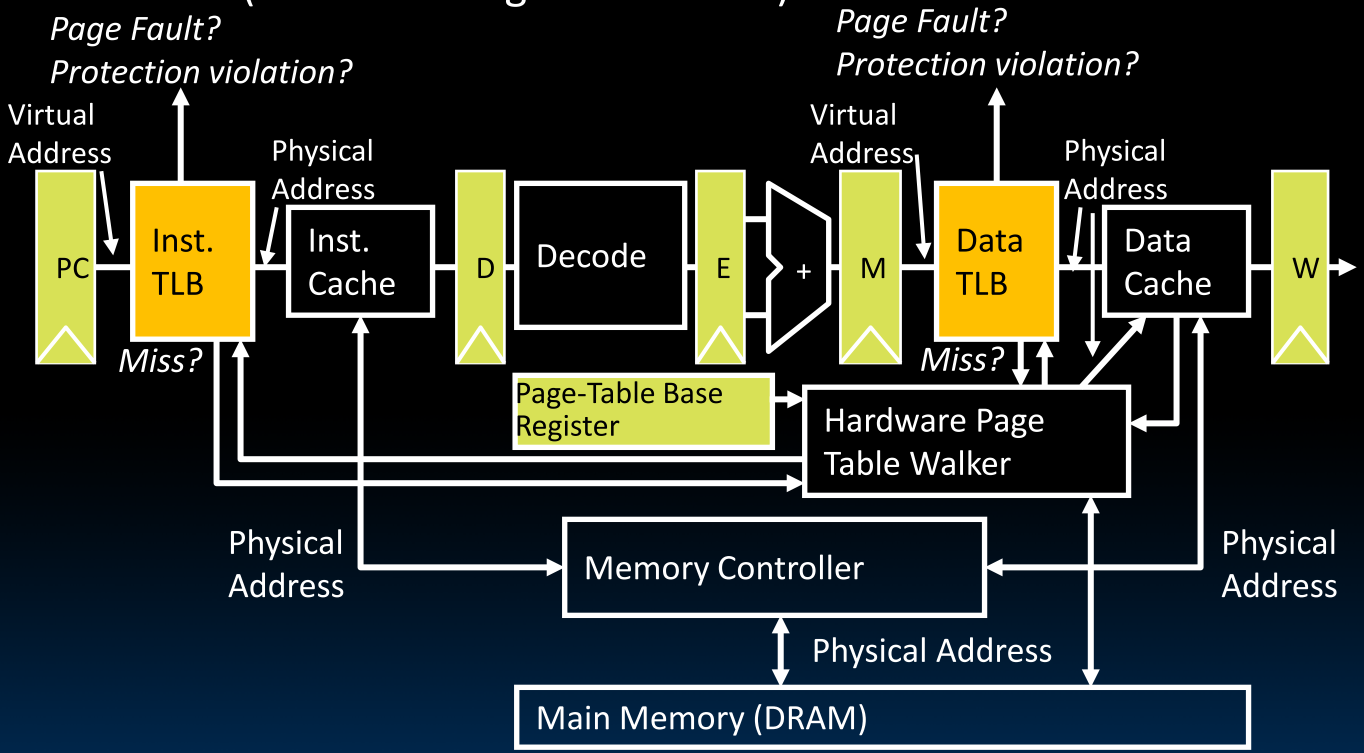
- Handling a TLB miss needs a hardware or software mechanism to refill TLB
  - Usually done in hardware
- Handling a page fault (e.g., page is on disk) needs a *precise trap* so software handler can easily resume after retrieving page
- Protection violation may abort process

Berkeley UNIVERSITY OF CALIFORNIA

Garcia, Nikolić

Page-Based Virtual-Memory Machine

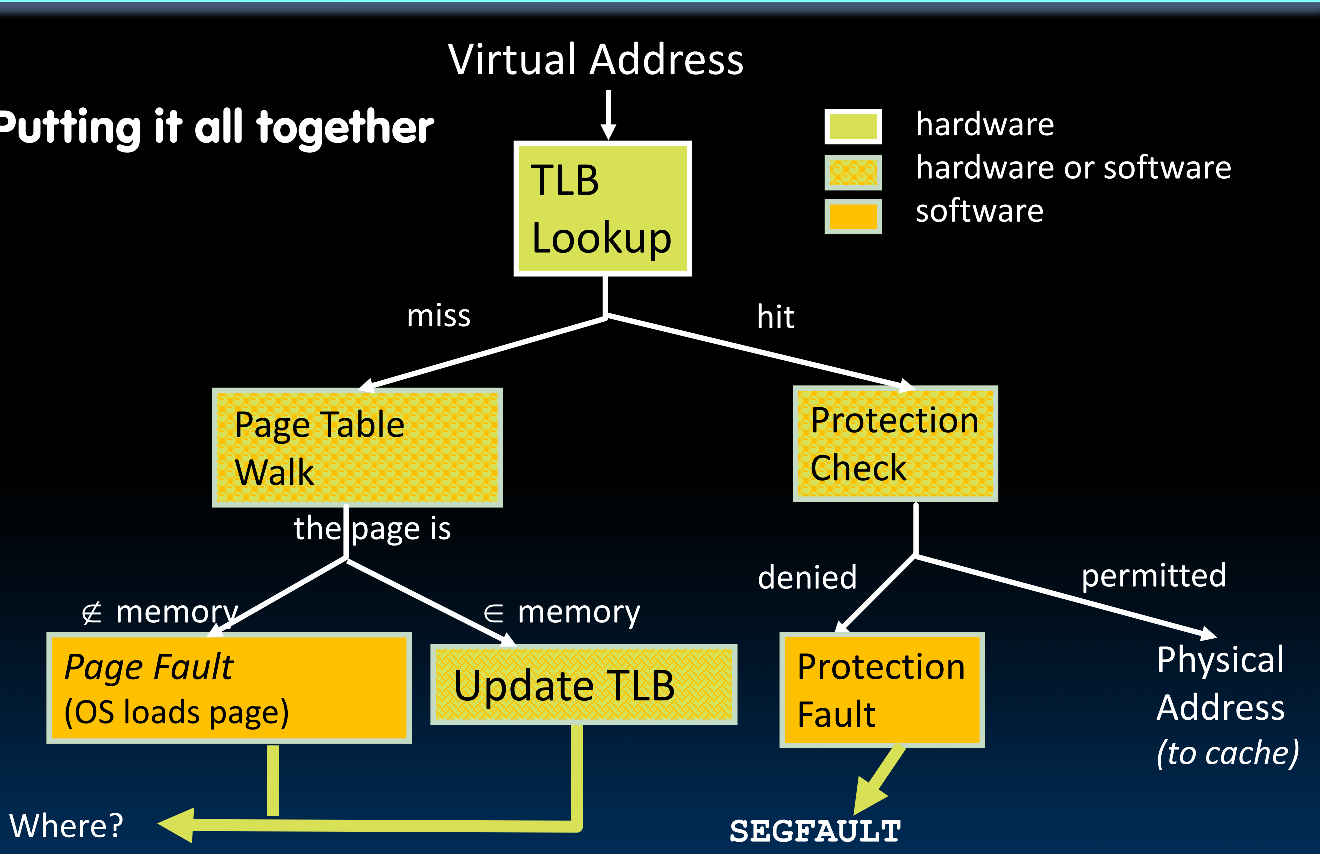
(Hardware Page-Table Walk)



- Assumes page tables held in untranslated physical memory

Address Translation

Putting it all together



Modern Virtual Memory Systems

Illusion of a large, private, uniform store

Protection & Privacy

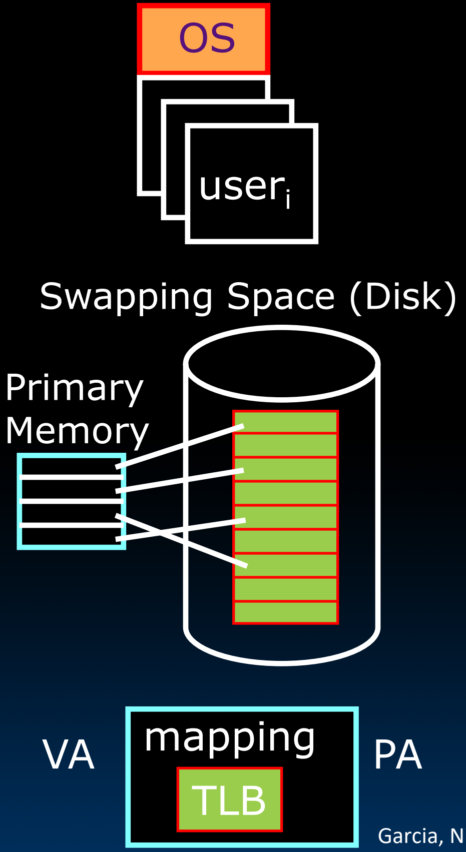
Several users/processes, each with their private address space

Demand Paging

Provides the ability to run programs larger than the primary memory

Hides differences in machine configurations

The price is address translation on each memory reference



Garcia, Nikolić

Review: Context Switching

- How does a single processor run many programs at once?
- *Context switch:* Changing of internal state of processor (switching between processes)
  - Save register values (and PC) and change value in Supervisor Page Table Base register (SPTBR)
- What happens to the TLB?
  - Current entries are for different process
  - Set all entries to invalid on context switch

VM Performance

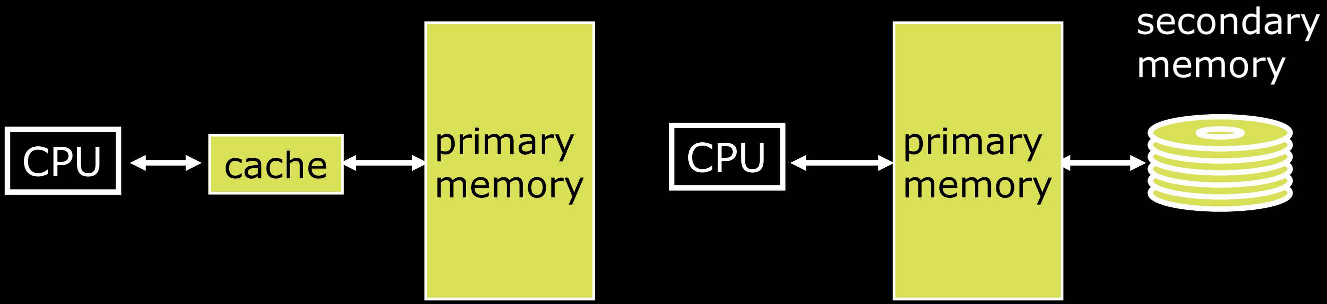
Comparing the Cache and VM

Cache version	Virtual Memory version
Block or Line	Page
Miss	Page Fault
Block Size: 32-64B	Page Size: 4K-8KiB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU), FIFO, random
Write Thru or Back	Write Back

VM Performance

- Virtual Memory is the level of the memory hierarchy that sits *below* main memory
  - TLB comes *before* cache, but affects transfer of data from disk to main memory
  - Previously we assumed main memory was lowest level, now we just have to account for disk accesses
- Same CPI, AMAT equations apply, but now treat main memory like a mid-level cache

Typical Performance Stats



Caching

cache entry

cache block (≈32-64 bytes)

cache miss rate (1% to 20%)

cache hit (≈1 cycle)

cache miss (≈100 cycles)

Demand paging

page frame

page (≈4Ki bytes)

page miss rate (<0.001%)

page hit (≈100 cycles)

page miss (≈5M cycles)

Impact of Paging on AMAT (1/2)

- Memory Parameters:
  - L1 cache hit = 1 clock cycles, hit 95% of accesses
  - L2 cache hit = 10 clock cycles, hit 60% of L1 misses
  - DRAM = 200 clock cycles (≈100 nanoseconds)
  - Disk = 20,000,000 clock cycles (≈10 milliseconds)
- Average Memory Access Time (no paging):
  - 1 + 5% × 10 + 5% × 40% × 200 = 5.5 clock cycles
- Average Memory Access Time (with paging):
  - 5.5 (AMAT with no paging) + ?

Impact of Paging on AMAT (2/2)

- Average Memory Access Time (with paging) =
  - $5.5 + 5\% \times 40\% \times (1\text{-}HR_{Mem}) \times 20{,}000{,}000$
- AMAT if $HR_{Mem} = 99\%$?
  - $5.5 + 0.02 \times 0.01 \times 20{,}000{,}000 = 4005.5$ (≈728x slower)
  - 1 in 20,000 memory accesses goes to disk: 10 sec program takes 2 hours!
- AMAT if $HR_{Mem} = 99.9\%$?
  - $5.5 + 0.02 \times 0.001 \times 20{,}000{,}000 = 405.5$
- AMAT if $HR_{Mem} = 99.9999\%$
  - $5.5 + 0.02 \times 0.000001 \times 20{,}000{,}000 = 5.9$