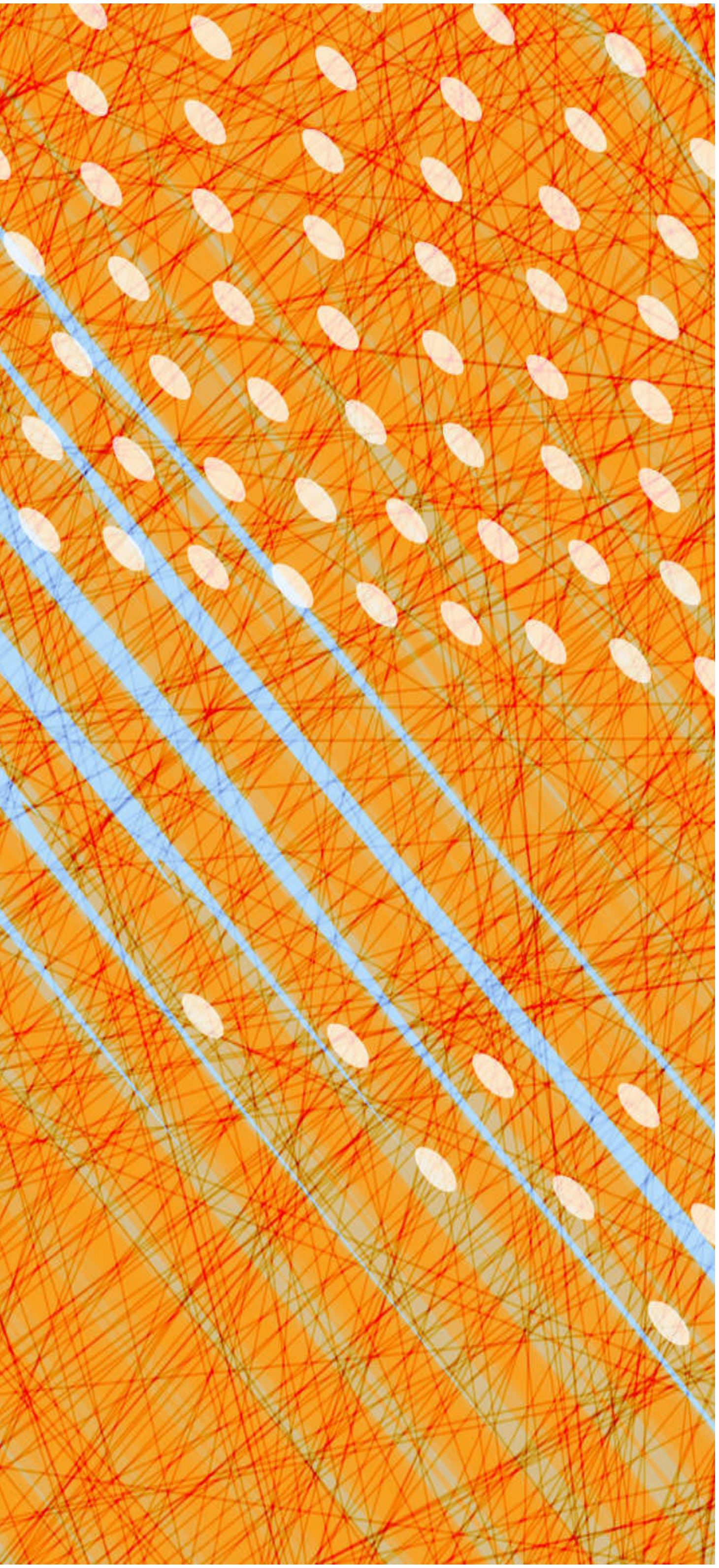


Research Night!!!

- If you are interested in undergraduate research

April 8 (This Thursday!!!)

5:30 to 7:30 pm

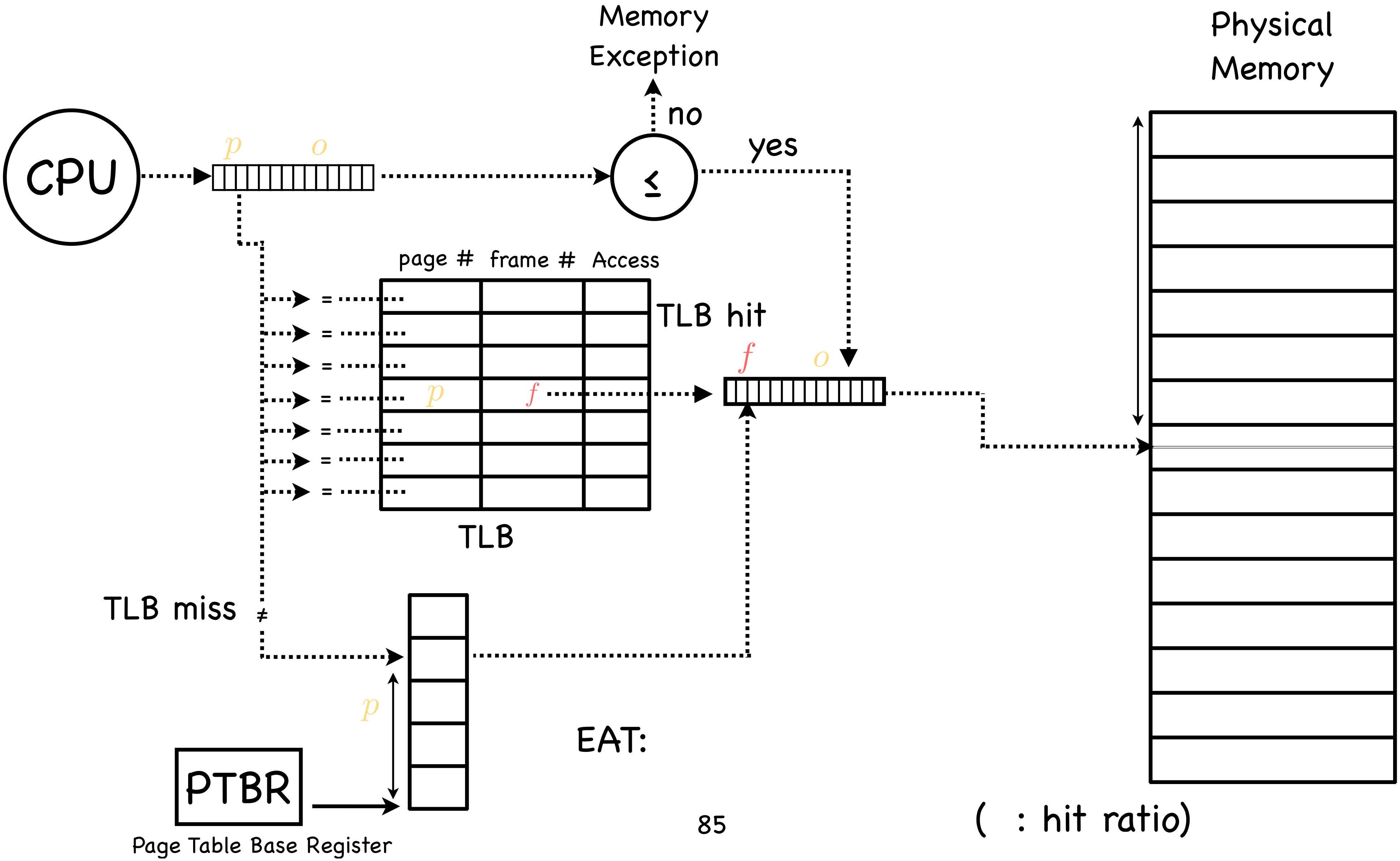
Over Discord:

https://discord.gg/cCM3QuGY3B

Caching!

- Keep the results of recent VA-PA translations in a structure called Translation Lookaside Buffer (TLB)

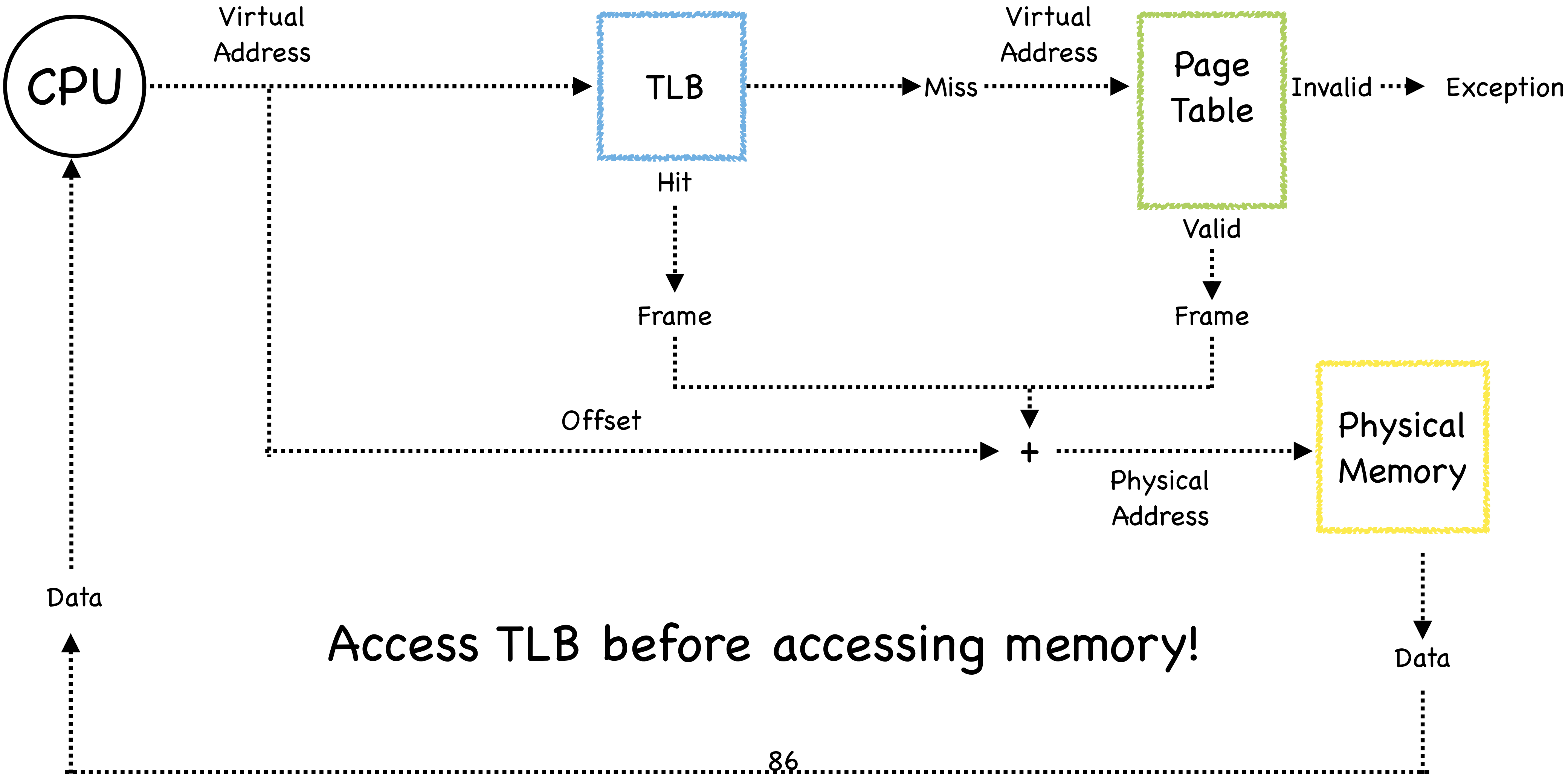
Speeding things up: The TLB



EAT:

(: hit ratio)

Address Translation with TLB



Access TLB before accessing memory!

Hit and Miss

- The TLB is small; it cannot hold all PTEs
    - it can be fast only if it is small!
  - Some translations will inevitably miss the TLB
  - Must access memory to find the appropriate PTE
    - called walking the page table
    - incurs large performance penalty

Handling TLB Misses

- Hardware-managed (e.g., x86)
    - The hardware does the page walk
    - Hardware fetches PTE and inserts it in TLB
        - If TLB is full, must replace another TLB entry
    - Done transparently to system software
- Software-managed (e.g., MIPS)
    - Hardware raises an exception
    - OS does the page walk, fetches PTE, and inserts evicts entries in TLB

Tradeoffs, Tradeoffs...

- Hardware-managed TLB
  + No exception on TLB miss. Instruction just stalls
  + No extra instruction/data brought into the cache
  - OS has no flexibility in deciding Page Table

- Software-managed TLB
  + OS can define Page Table organization
  + More flexible TLB entry replacement policies
  - Slower: exception causes to flush pipeline; execute handler; pollute cache

TLB Consistency - I

- On context switch
  - VAs of old process should no longer be valid
  - Change PTBR — but what about the TLB?

TLB Consistency - I

- On context switch
  - VAs of old process should no longer be valid
  - Change PTBR — but what about the TLB?
    - Option 1: Flush the TLB
    - Option 2: Add pid tag to each TLB entry

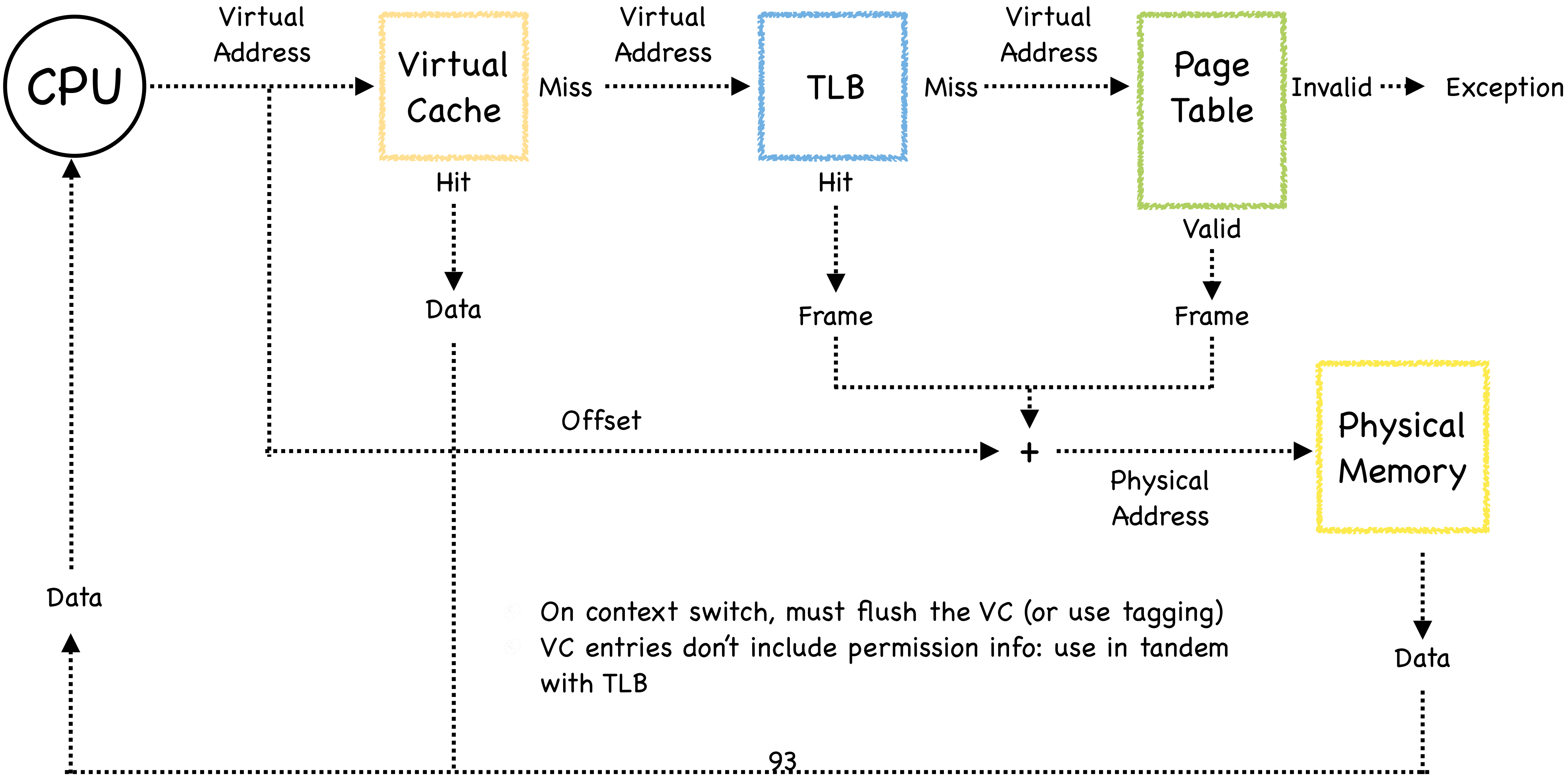
	PID	VirtualPage	PageFrame	Access
TLB Entry	1	0x0053	0x0012	R/W

Ignore entries with wrong PIDs

TLB Consistency - II

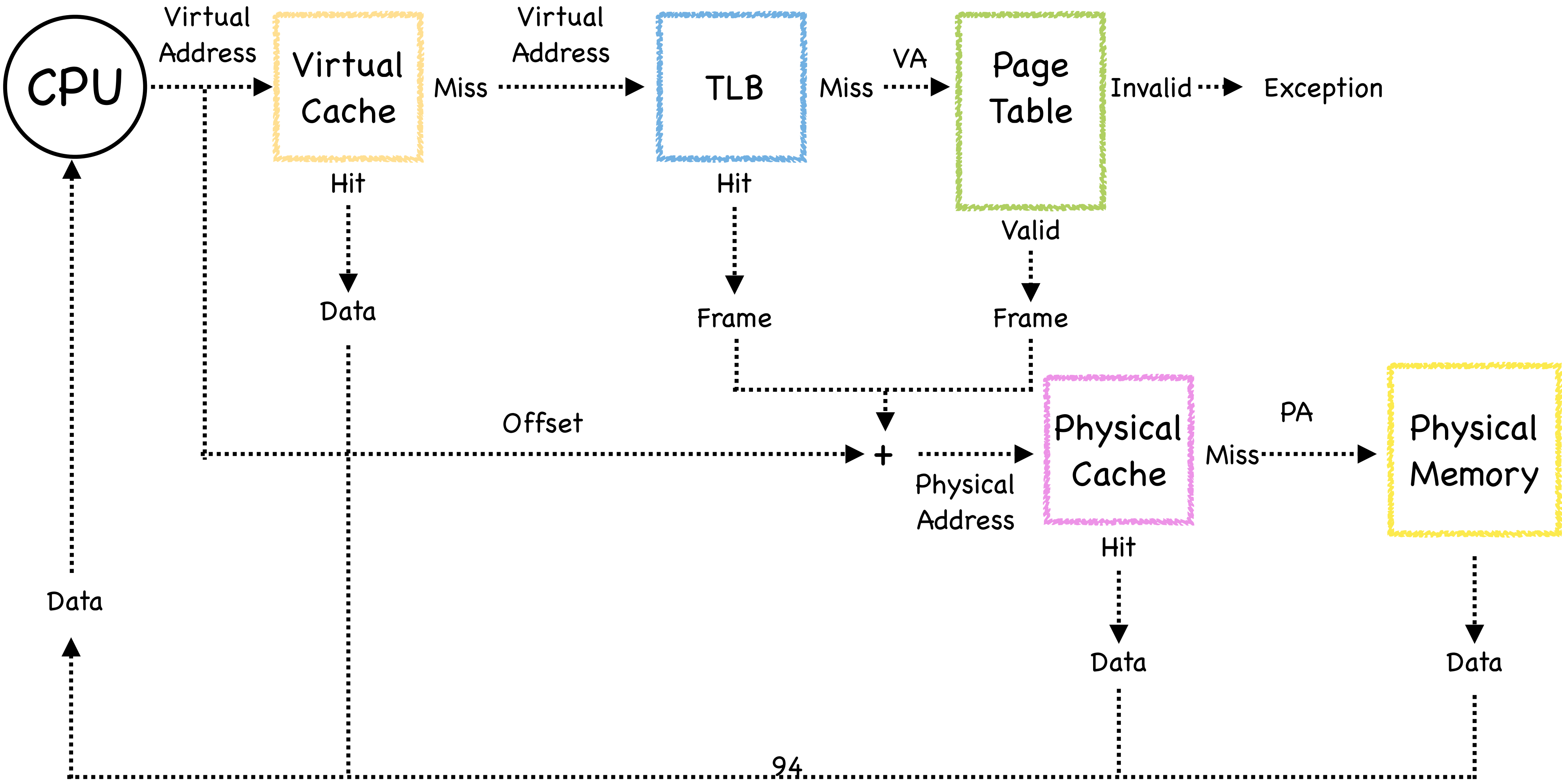
- What if OS changes permissions on page?
  - If permissions are reduced, OS must ensure affected TLB entries are purged
    - e.g., on copy-on-write
  - If permissions are expanded, no problem
    - new permissions will cause an exception and OS will restore consistency

Virtually Addressed Caches



- On context switch, must flush the VC (or use tagging)
- VC entries don't include permission info: use in tandem with TLB

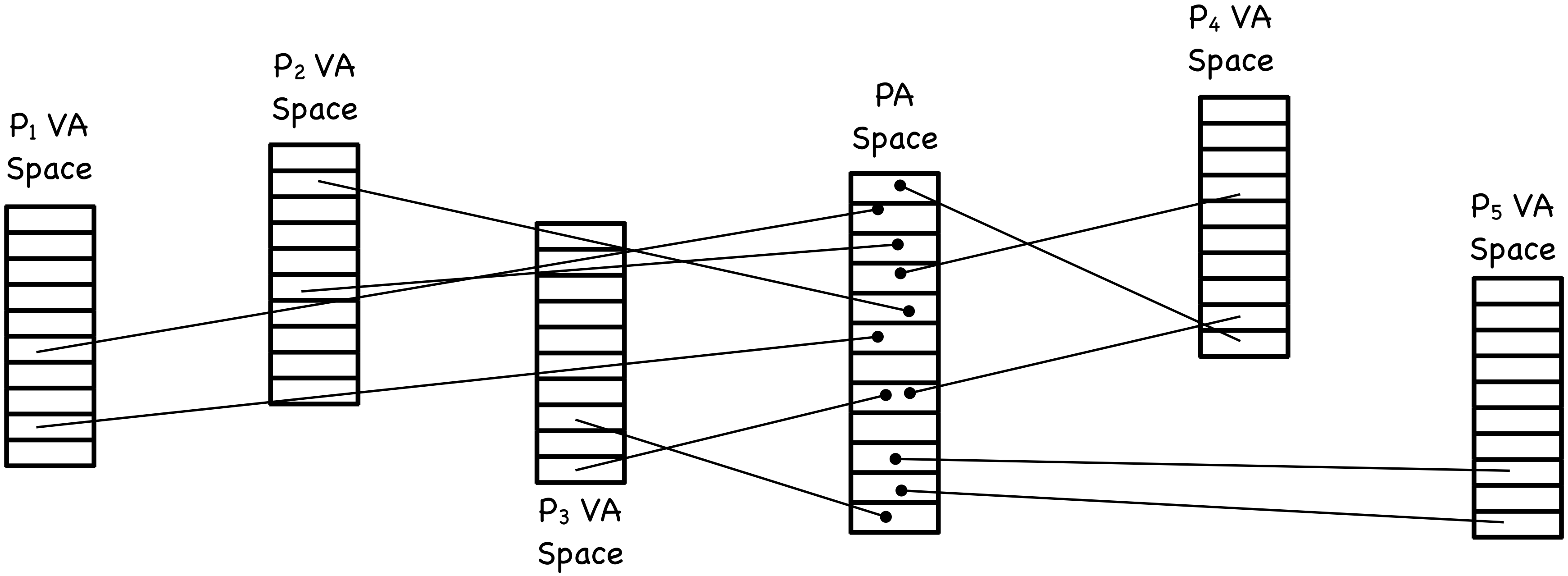
Physically Addressed Caches



Translation and Locality

- Adding a layer of indirection disrupts the spatial locality of caching
- OS may map adjacent virtual pages to frames sharing the same entry in physical addressed cache
  - cache appears smaller
  - performance is unpredictable
- Solution: page coloring
  - frames colored according to cache buckets they will use
  - OS spreads each process' pages across as many colors as possible

A different approach



- So many virtual pages…
- …comparatively few physical frames
- What if mapping size were proportional to the number of frames, not pages?

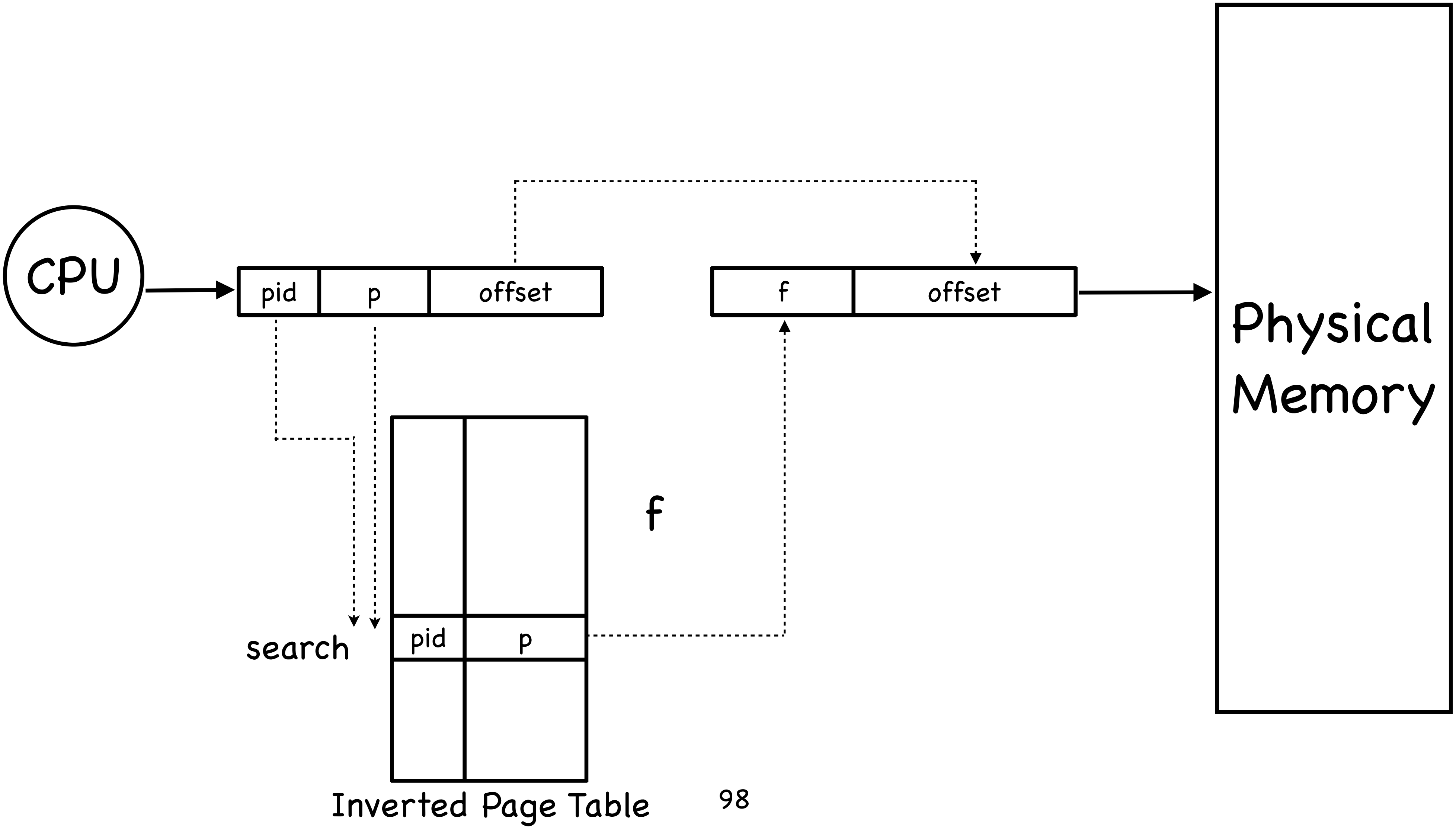
Inverted Page Table

- For each frame, a register containing
  - Residence bit
    - is the frame occupied?
  - Number of the occupying page
  - Protection bits
- Searched by page number

Catch?

- The VAS of different processes may map the same page number to different frames!
  - add pid to IPT entry

Basic Inverted Page Table Architecture

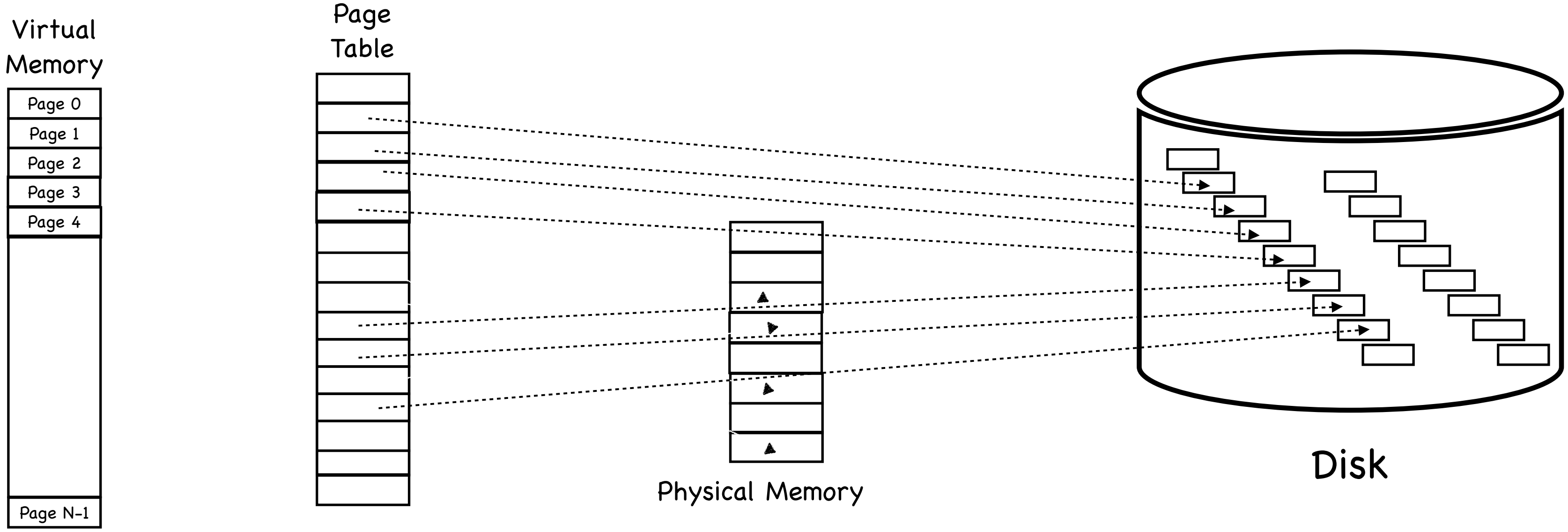


Discussion

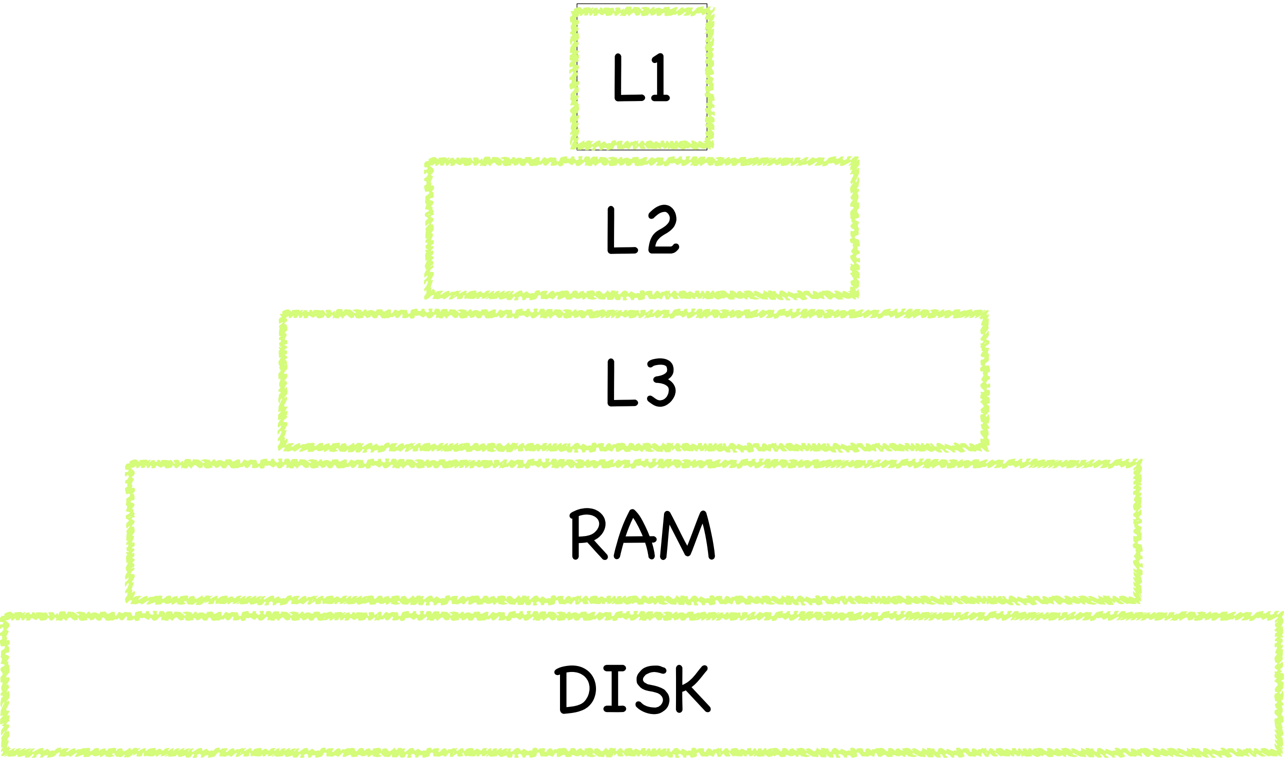
- Less memory to store page tables
- More time to search page tables
  - searching linearly a long list of entries is no fun
  - and using associative memory is too expensive
- Solution: hashing
  - hash(page, pid) -> PT entry (or chain of entries)
- Sharing frames does not work
  - In standard paging, multiple PTE could map to same frame, but now single page for each frame...
  - use “segment identifier” instead of pid in inverted PT
    - multiple processes can share the same segment

Virtual Memory

- Each process has the illusion of a large address space
  - 2^x bytes for x-bit addressing
- However, physical memory is usually much smaller
  - and we want to run multiple processes concurrently
- How do we give this illusion to multiple processes?
  - Virtual Memory: back every memory segment with a file on disk



Processes execute from disk!



L1

L2

L3

RAM

DISK

RAM is just another layer of cache!

Swapping vs. Paging

- Swapping
  - Loads entire process in memory
  - “Swap in” (from disk) or “Swap out” (to disk) a process
  - Slow (for large processes)
  - Wasteful (might not require everything)
  - Does not support sharing of code segments
  - Virtual memory limited by size of physical memory
- Paging
  - Runs all processes concurrently
  - A few pages from each process live in memory (the rest is on disk)
  - Finer granularity, higher performance
  - Large virtual memory supported by small physical memory
  - Certain pages (e.g., read-only ones) can be shared among processes

A Virtual Page can be…

- Mapped (present bit set in PTE) may trigger Page Fault

    to a physical frame, with certain r/w/x permissions

- Not mapped (present bit not set in PTE)

    in some physical frame, but not currently mapped

    or still in the original program file

    or needing to be zero-filled (heap, BSS, stack)

    or on backing store (paged or swapped out)

    or not part of one of the processes' segment

        Segmentation Fault!

Page Fault

Handling a Page Fault

- Identify page and reason
    - access inconsistent with segment access rights
        - terminate process
    - access a page currently on disk
        - does frame with the code/data already exist?
            - if not, allocate a frame and load page in
    - access of zero-initialized data (BSS) or stack
        - allocate a frame, initialize all bytes to zero
    - access of a COW page
        - allocate a frame and copy

When a page must be brought in…

- Find a free frame
  - evict one if there are no free frames
- Issue disk request to fetch data for page
- Move “current process” to disk queue
- Context switch to new process
- Update PTE when disk completes
  - frame number, present bit, RWX bits, etc.
- Move “current process” to ready queue

When a page must be swapped out…

- Find all page table entries that refer to old page
    - Frame might be shared
    - Access Core Map (frames —> pages)
- Set each page table entry to not present (invalid)
- Remove any TLB entries
    - "TLB Shootdown": in multiprocessors, TLB entry must be eliminated from the TLB of all processors
- Write page back to disk, if needed
    - Dirty bit in PTE indicates need

Demand Paging

MIPS Style

1. TLB Miss
2. Exception to kernel
3. Page Table walk
4. Page fault (present bit not set in Page Table)
5. Convert VA to file offset
6. Allocate page frame (evict page if needed)
7. Initiate disk block read into page frame
8. Disk interrupt when DMA completes
9. Mark page as present
10. Update TLB
11. Resume process at faulting instruction
12. TLB hit
13. Execute instruction

Software handling page fault

Demand Paging:

x86 Style

1. TLB Miss
2. Page Table walk
3. Page fault (page not present in Page Table)
4. Exception to kernel
5. Convert VA to file offset
6. Allocate page frame (evict page if needed)
7. Initiate disk block read into page frame
8. Disk interrupt when DMA completes
9. Mark page as present
10. Resume process at faulting instruction
11. TLB miss
12. Page Table walk – success!
13. TLB updated
14. Execute instruction

Software handling page fault