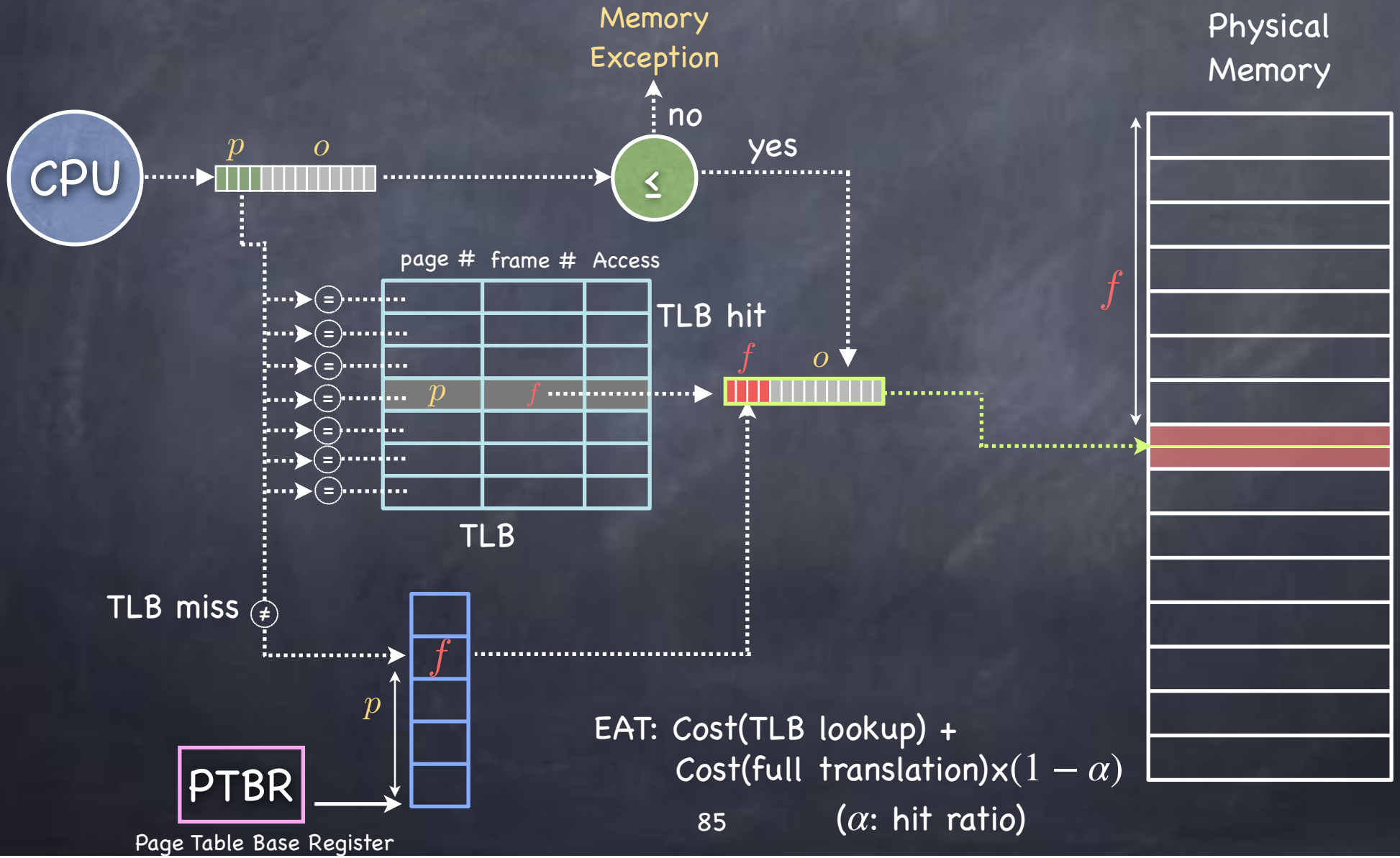
# Getting slooooower

- Every new level of paging
  - reduces the memory overhead for computing the mapping function...
  - ... but increases the time necessary to perform the mapping function

# Caching!

- Keep the results of recent VA-PA translations in a structure called Translation Lookaside Buffer (TLB)
  - TLB is a cache for page-to-frame mappings

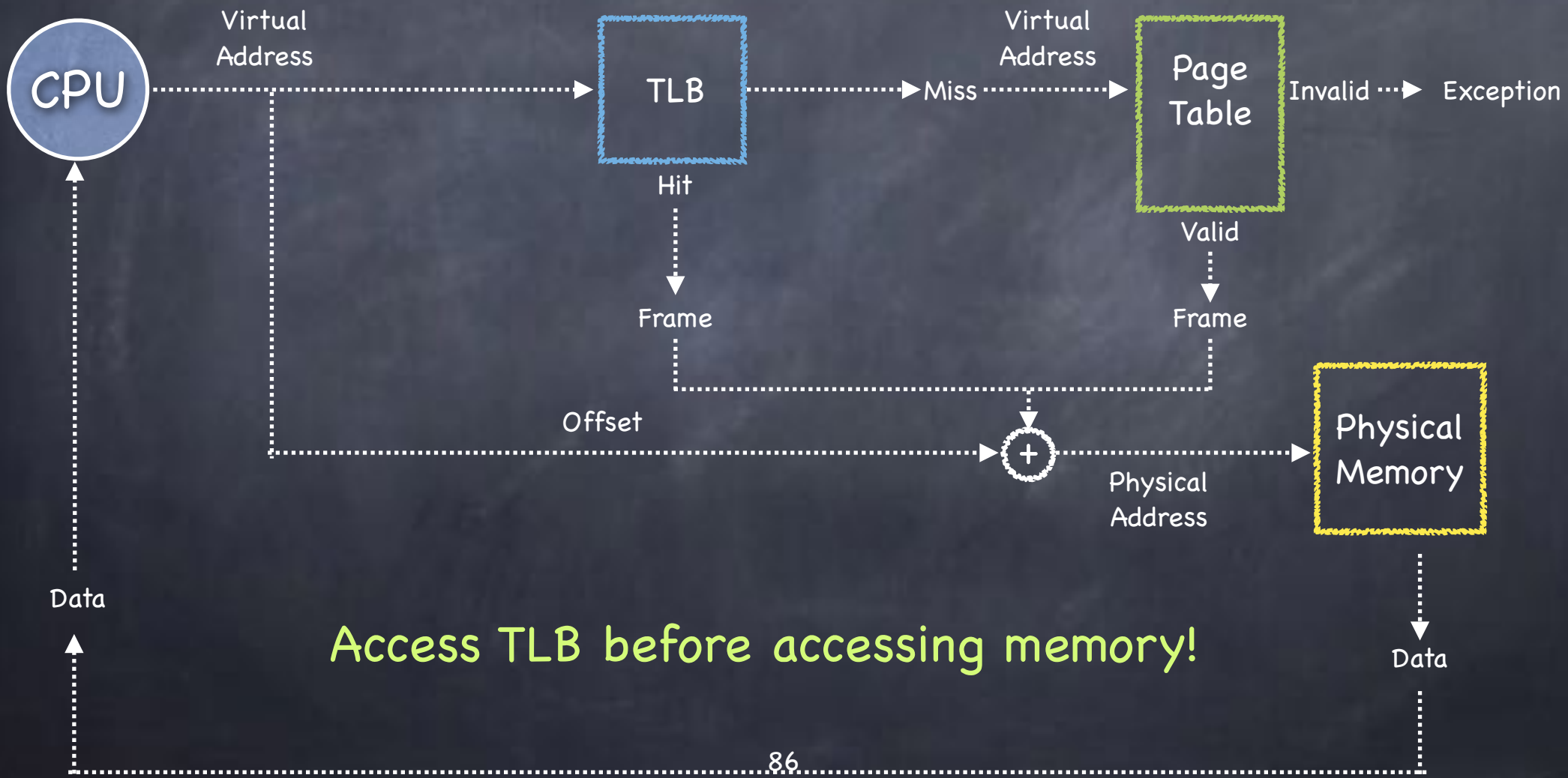
# Speeding things up: The TLB




EAT: Cost(TLB lookup) + Cost(full translation)x$(1-\alpha)$

($\alpha$: 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

- 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

# Handling TLB Misses: Software

- Software-managed (e.g., MIPS)
  - Hardware raises an exception, trap handler runs in kernel
  - Handler does the page walk, fetches PTE, and inserts/evicts entries in TLB
  - Handler must return to the same instruction that caused the trap!
  - Careful not to generate a TLB miss while running the handler!

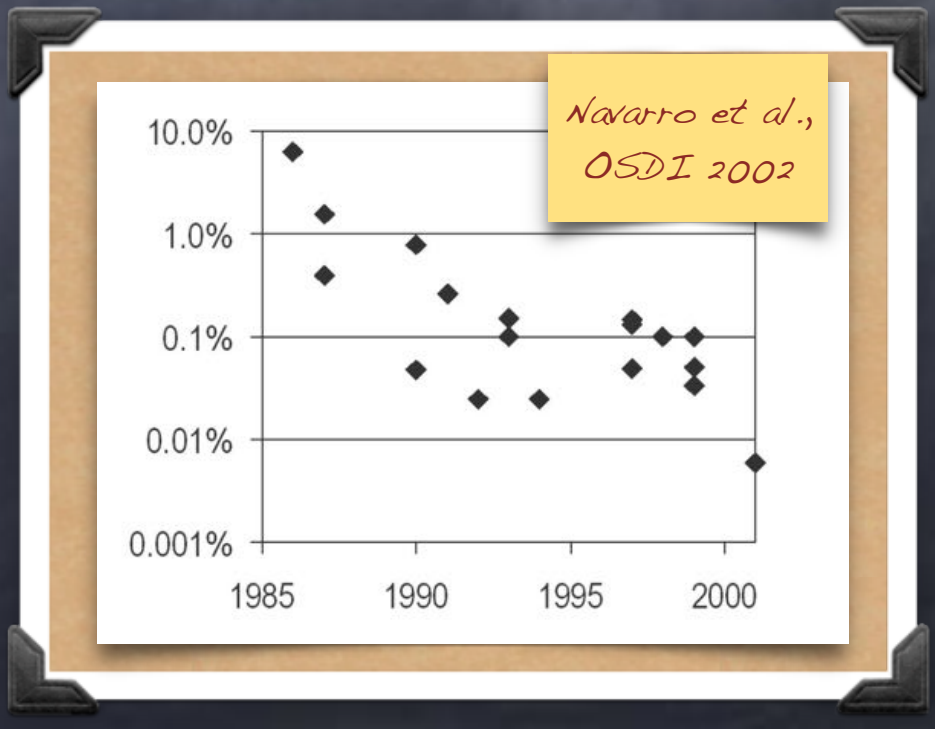
# Tradeoffs, Tradeoffs...

- 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: hardware must know location and format of PTEs
- 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 Coverage

- What fraction of memory can be accessed without TLB misses?
  - low TLB coverage can result in a large number of memory references

1000-fold decrease in 15 years!



# Superpages

- Wider TLB coverage by supporting page sizes that are multiples of the base page size: superpages
  - Pentium: 4KB base; 4MB Super
  - Itanium: 10 sizes, from 4KB (base) to 256 MB
- A set of contiguous base pages can be promoted to a superpage
- Demotion works the other way around

For more

Navarro et al.,
"Practical, transparent Operating systems support for superpages"

# Tradeoffs, Tradeoffs...

+ Improved TLB coverage! but...
- Larger internal fragmentation
- External fragmentation (?)
  - superpage of N base pages
  - N free base frames free, but not contiguous
- Less efficient reading
- Coarser granularity for dirty, reference, and protection bits

# 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 |

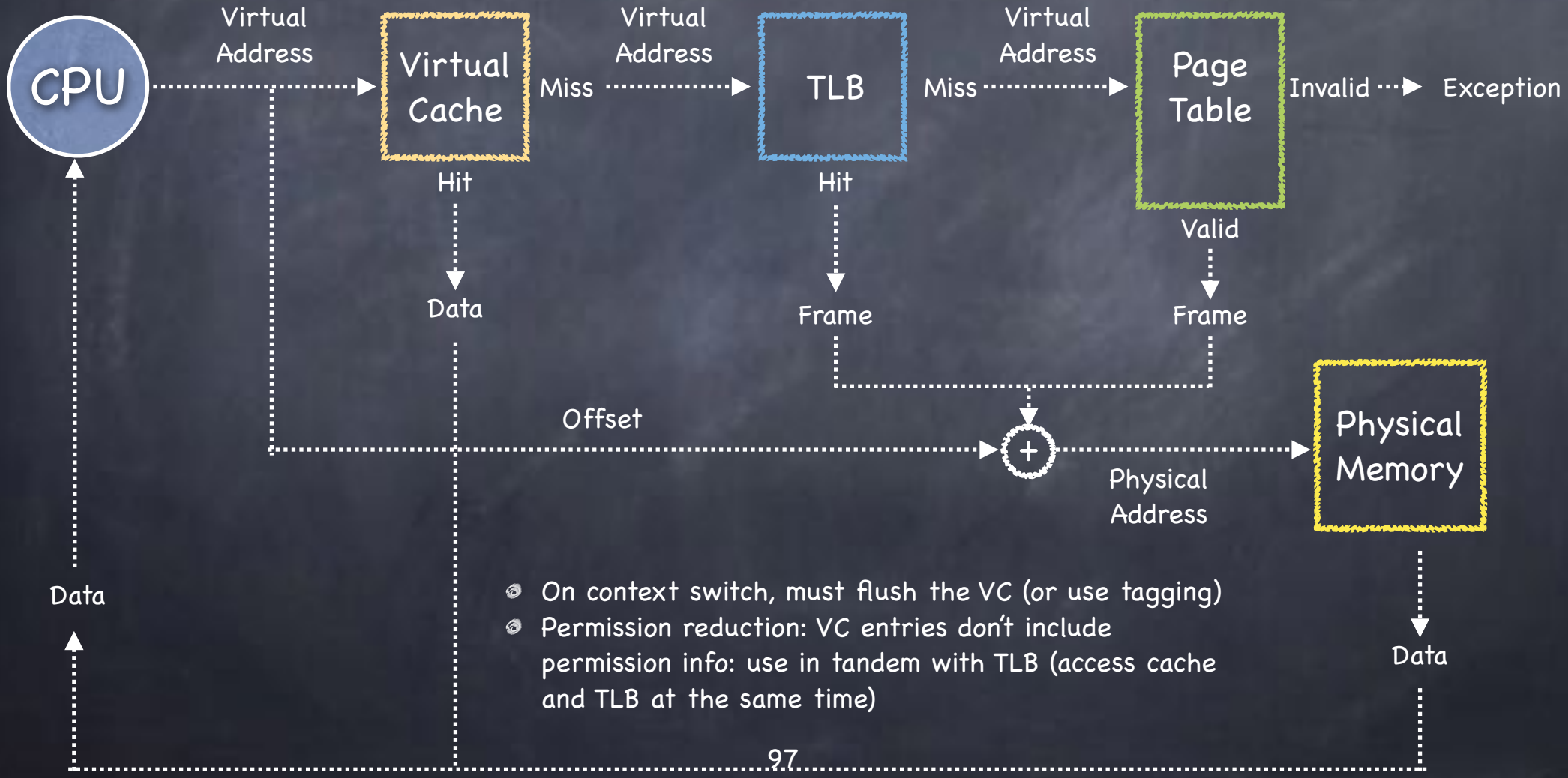
Ignore entries with wrong PIDs

# 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 hardware and OS will restore consistency

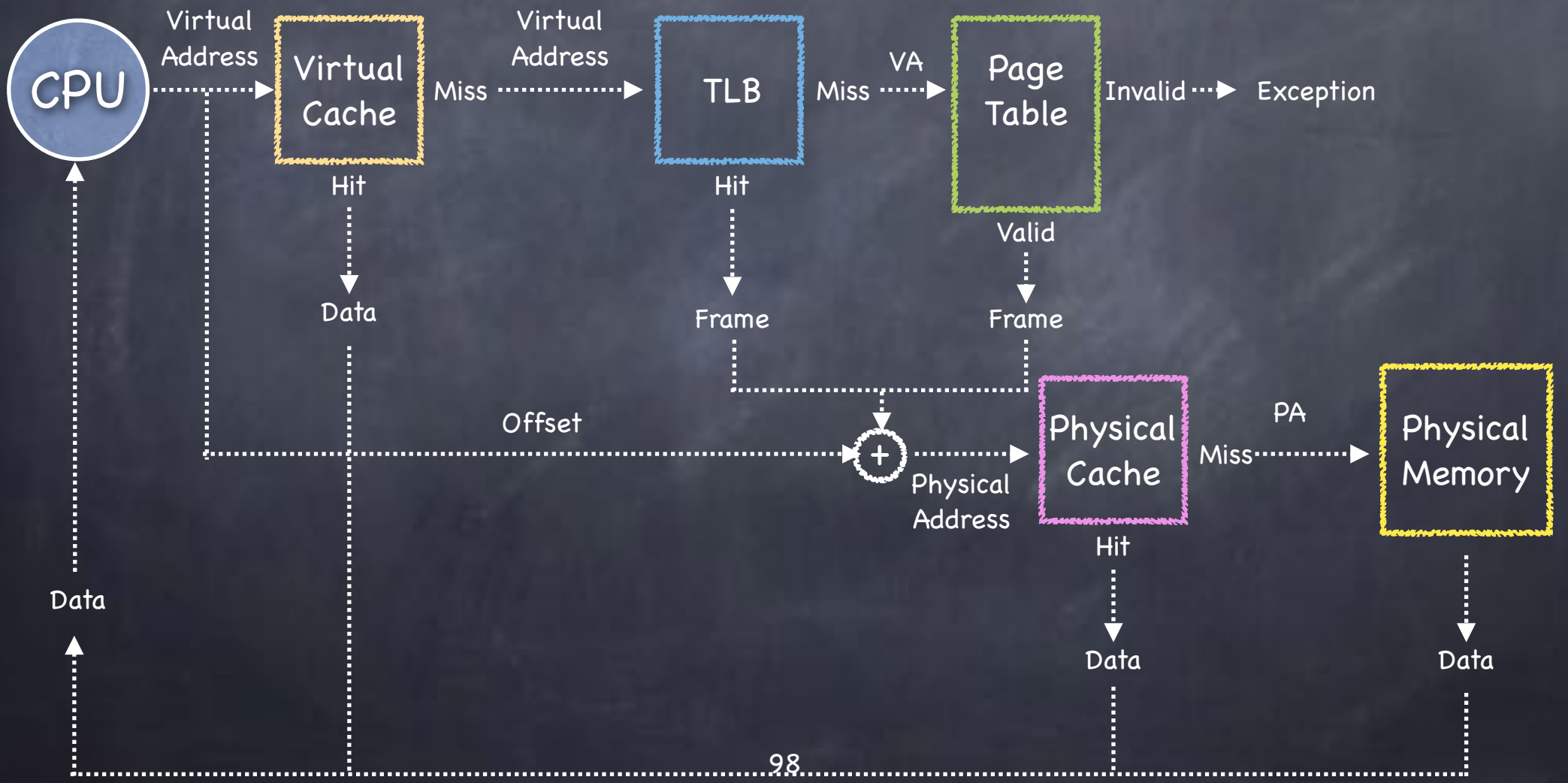
# Virtually Addressed Caches

A copy of the contents of physical memory, indexed by the virtual address




- On context switch, must flush the VC (or use tagging)
- Permission reduction: VC entries don't include permission info: use in tandem with TLB (access cache and TLB at the same time)

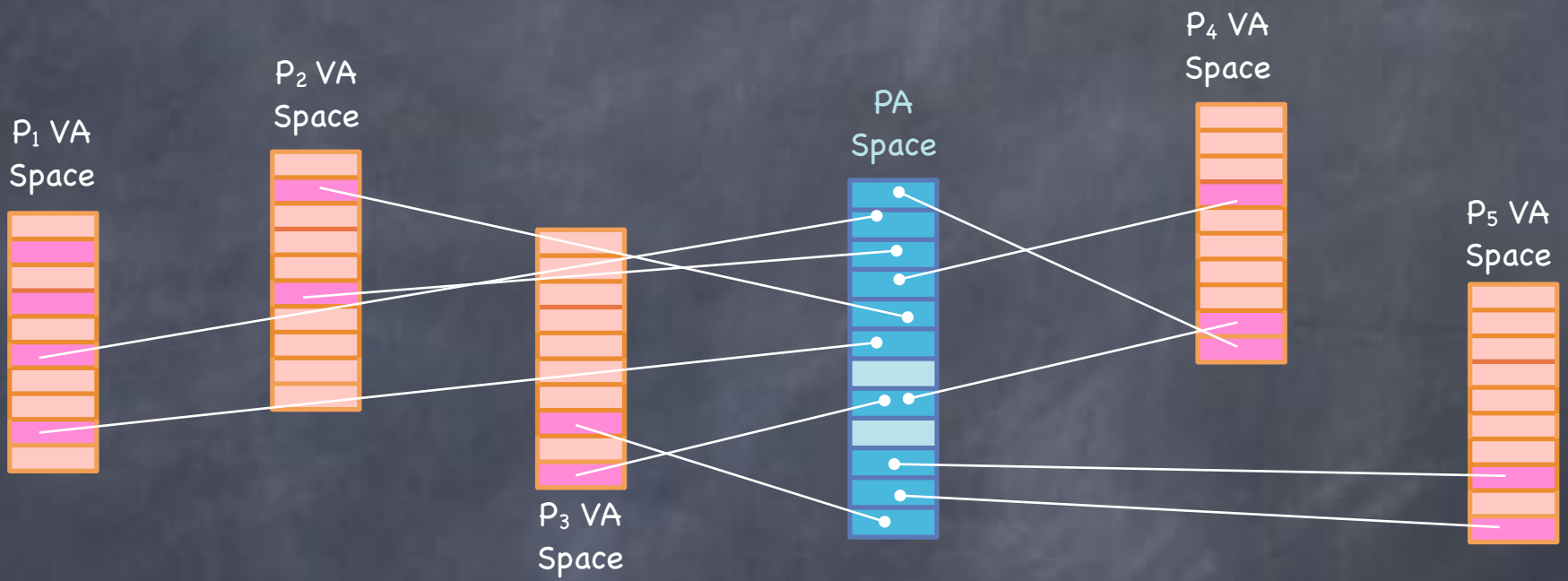
# Physically Addressed Caches



# Translation and Locality

- A physically indexed cache ensures that nearby frames are not mapped to the same cache line...
- ...but adding a layer of indirection disrupts the spatial locality of physical caching
- OS may map adjacent virtual pages to physically far away frames, sharing the same entry in physical addressed cache
  - cache appears smaller
  - performance is unpredictable
- Solution: page coloring
  - frames colored according to the bucket where they will be cached
  - When mapping pages to frames, OS spreads each process' pages across frames with 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?

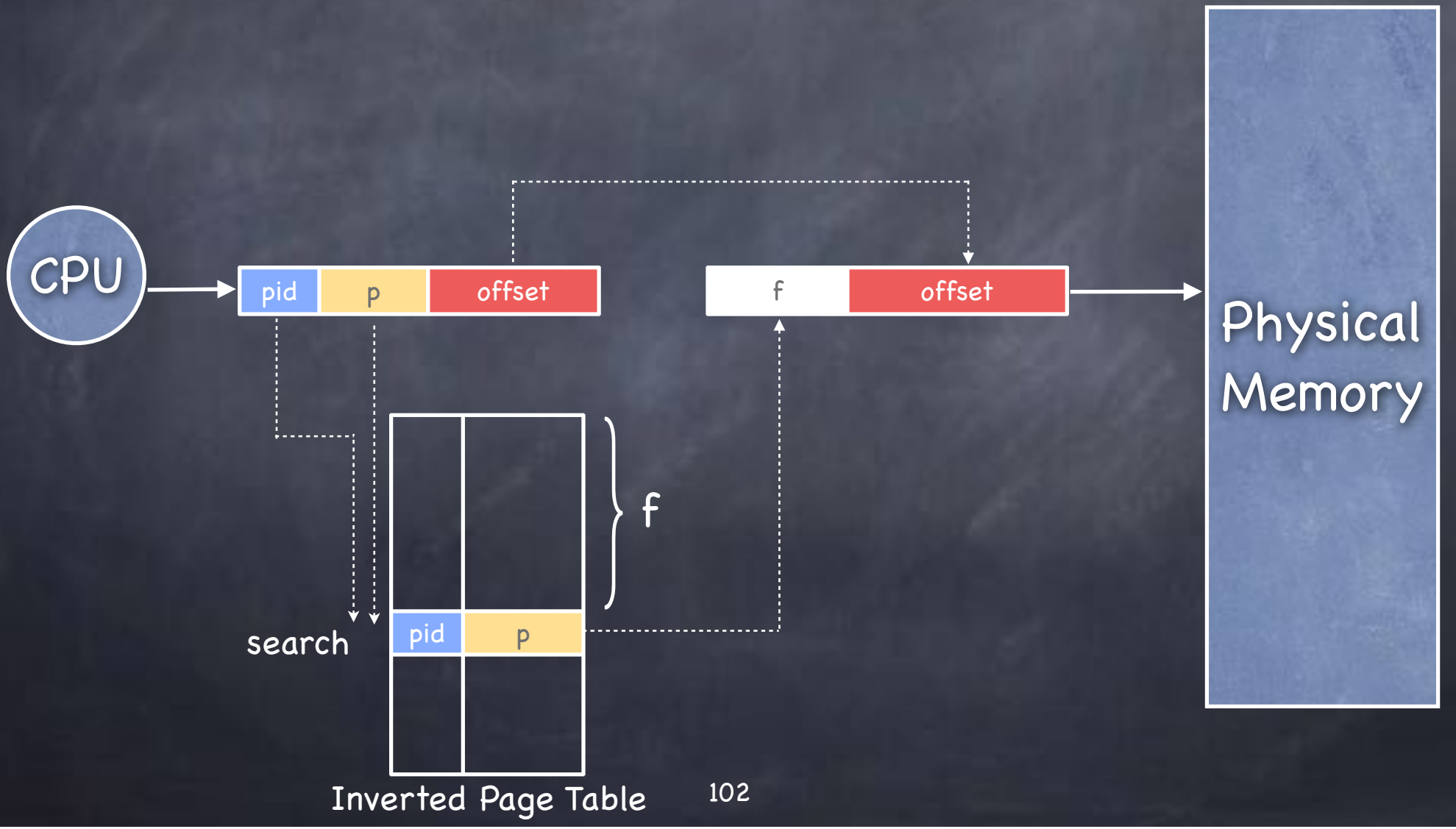
# Inverted Page Table

- 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

# Hashed Inverted Page Tables

- Add a Hash Anchor Table, mapping <pid, VP#> to an entry of the Inverted Page Table
- Collisions handled by chaining