EE 42
1:30

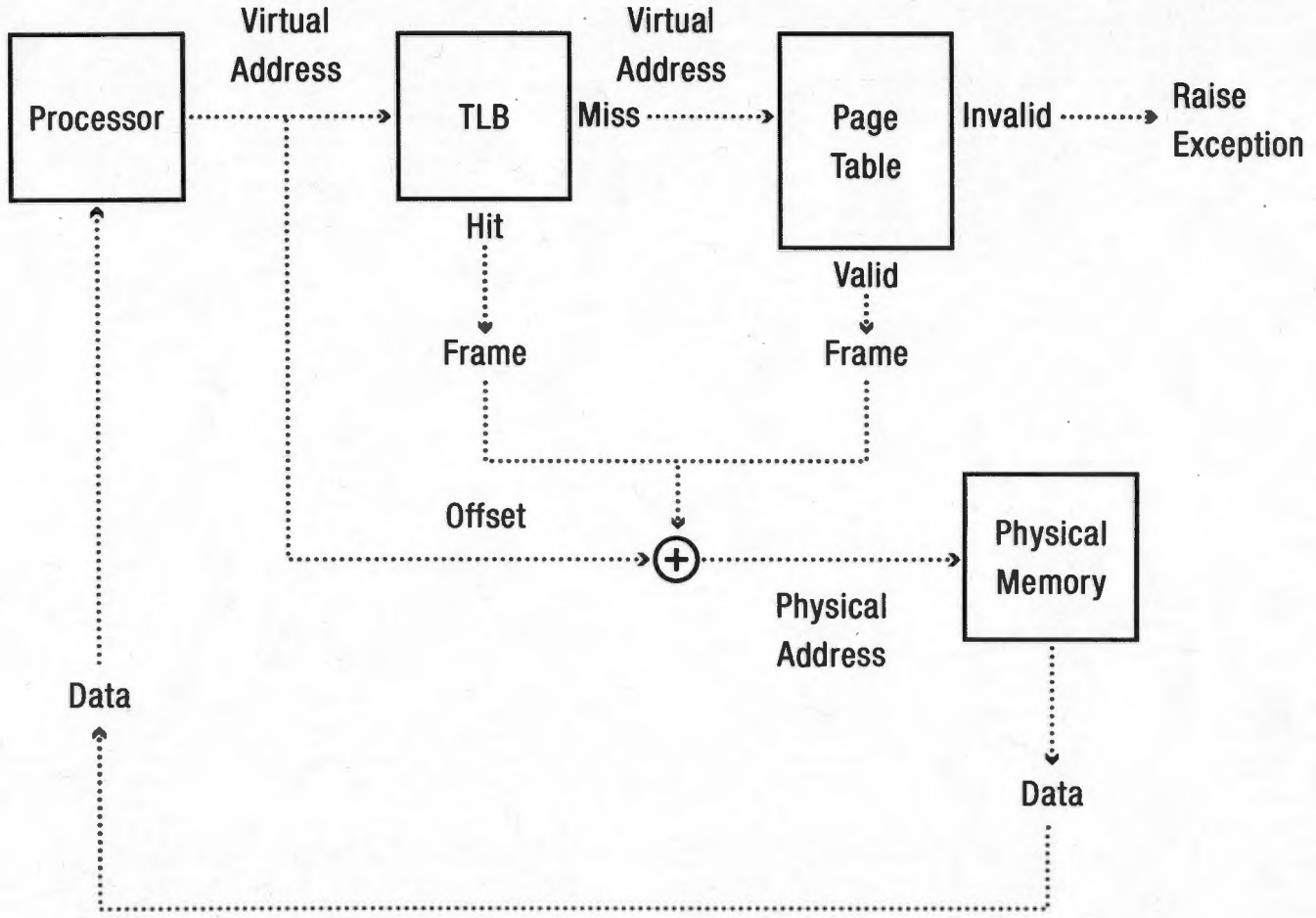
Efficient Address Translation

- Translation lookaside buffer (TLB)
  - Cache of recent virtual page -> physical page translations
  - If cache hit, use translation
  - If cache miss, walk multi-level page table
- Cost of translation =

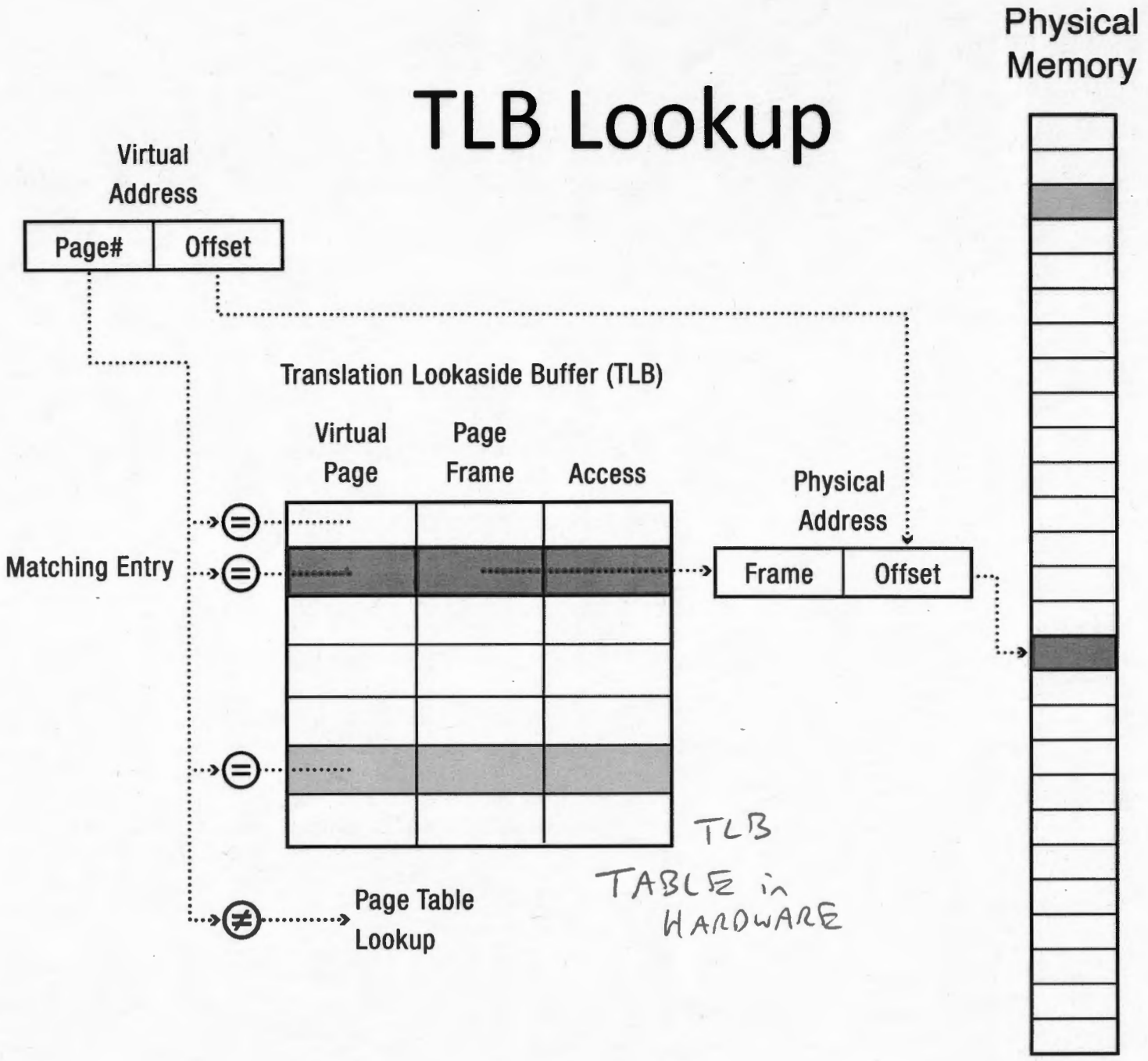
  Cost of TLB lookup +

  Prob(TLB miss) * cost of page table lookup

TLB and Page Table Translation

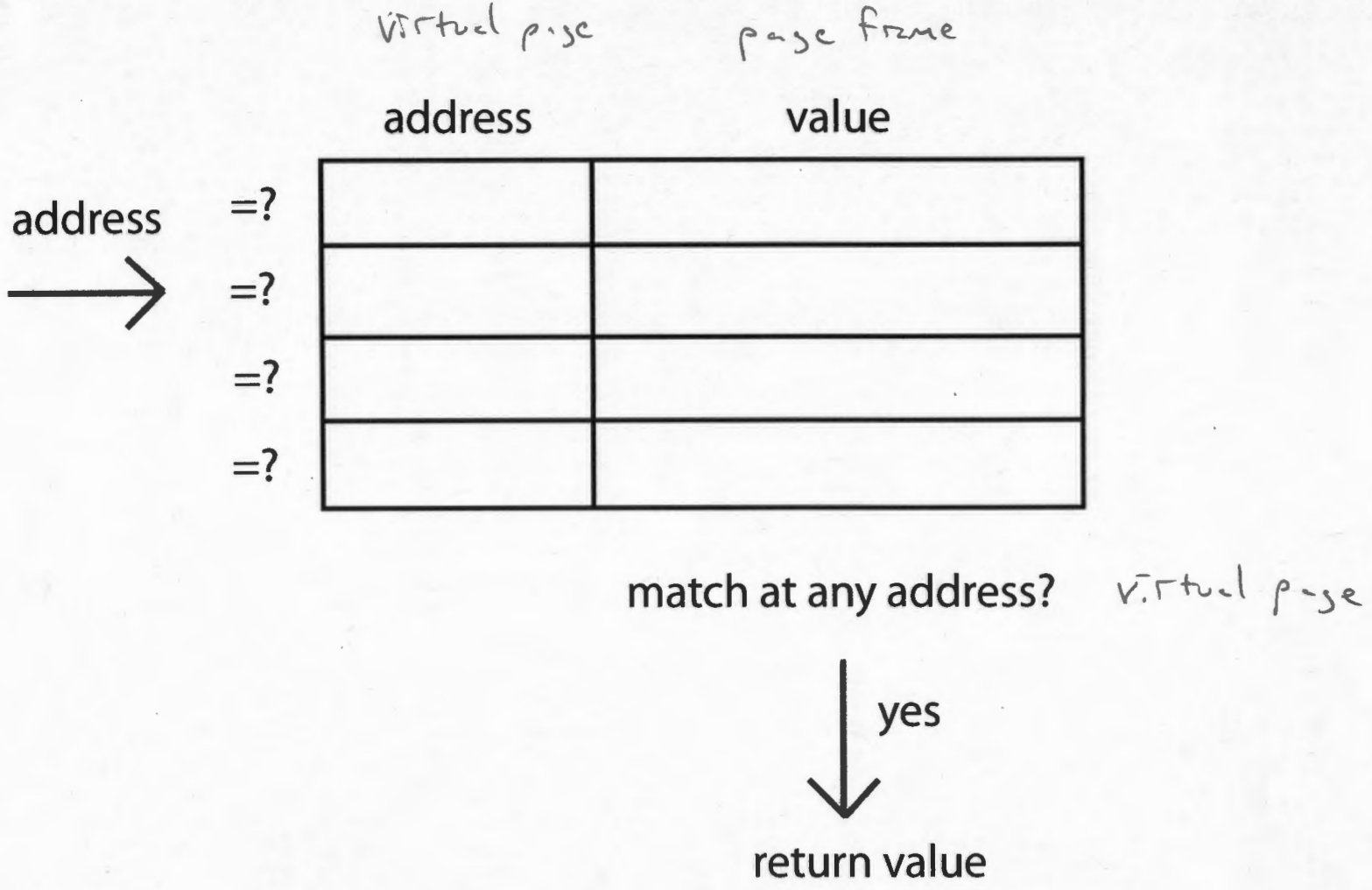


TLB Lookup


Physical Memory
Virtual Address
Page#
Offset
Translation Lookaside Buffer (TLB)
Virtual Page
Page Frame
Access
Physical Address
Frame
Offset
Matching Entry
Page Table Lookup
TLB
TABLE in HARDWARE

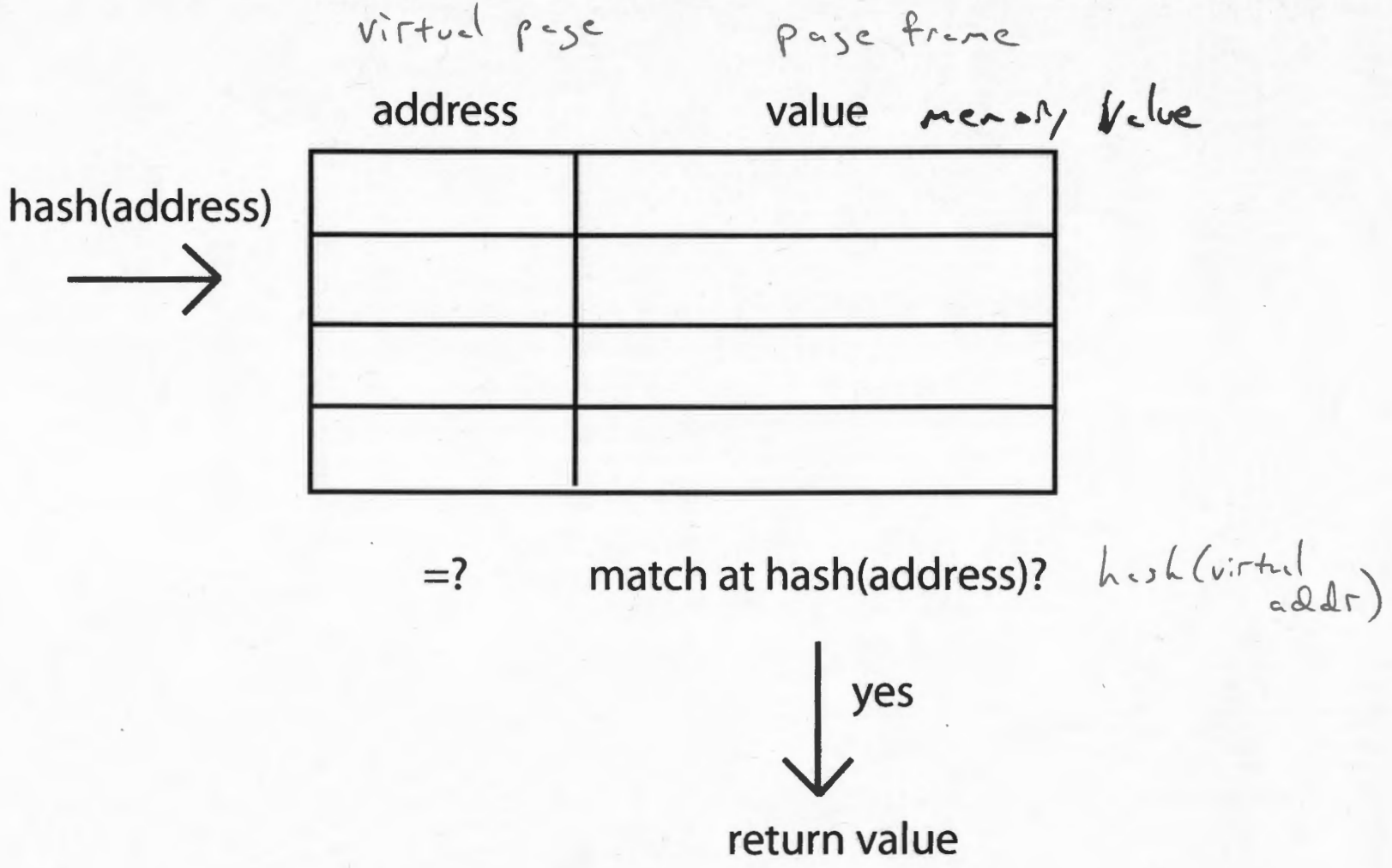
TLB OR

Cache Lookup: Fully Associative



TLB or

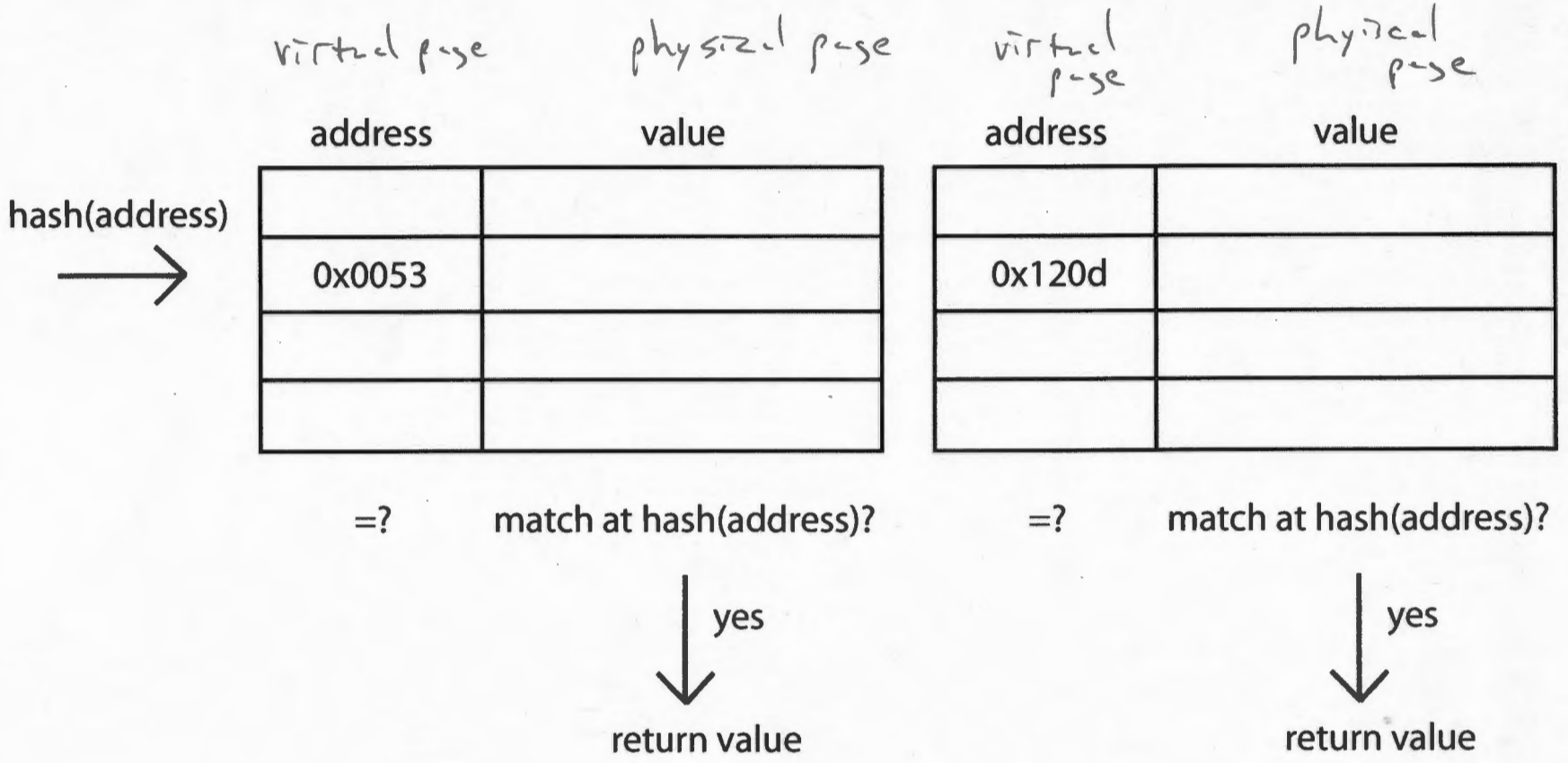
Cache Lookup: Direct Mapped



Why are TLB's not direct mapped?

TLB or

Cache Lookup: Set Associative



Most TLB's ~~are~~ (and most caches) are set associative

Question

- What happens on a context switch?
  - Reuse TLB?
  - Discard TLB? (xk resets TLB)

set CR3

- Solution: Tagged TLB
  - Each TLB entry has process ID
  - TLB hit only if process ID matches current process

AND ADDRESS MATCH

PROCESSID : ADDRESS

MIPS Address Translation

- Software-Loaded Translation lookaside buffer (TLB)
  - Cache of virtual page -> physical page translations
  - If TLB hit, physical address [As before]
  - If TLB miss, trap to kernel [New!]
  - Kernel fills TLB with translation and resumes execution
- Kernel can implement *any* page translation
  - Page tables
  - Multi-level page tables
  - Inverted page tables
  - ...

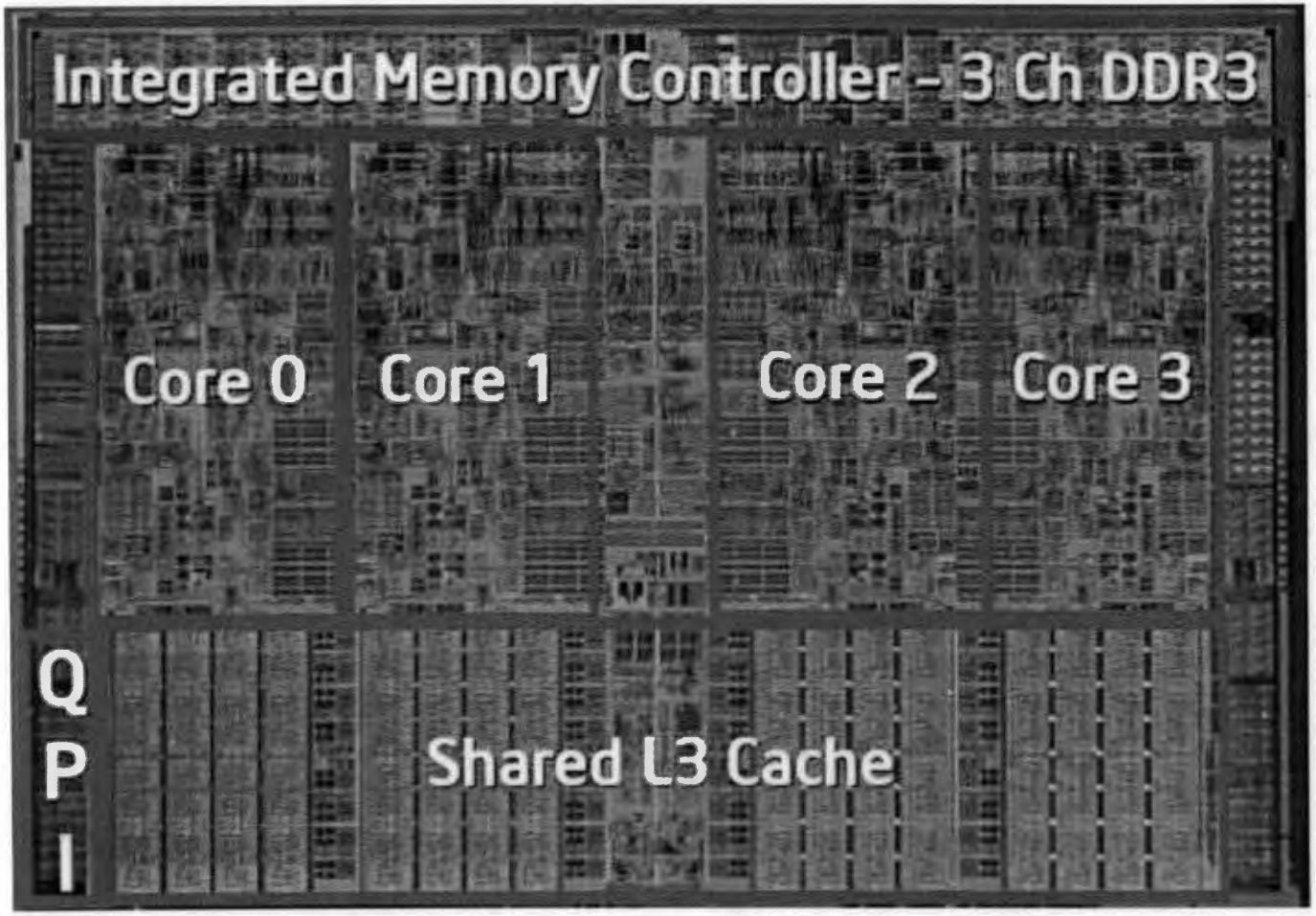
Question

- What is the cost of a TLB miss on a modern processor?
  - Cost of multi-level page table walk
  - MIPS: plus cost of trap handler entry/exit

Hardware Design Principle

The bigger the memory, the slower the memory

Intel i7



TLB L1
L2
ALU

L3

Memory Hierarchy

Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 µs	100 TB
Local non-volatile memory (FLASH)	100 µs	100 GB
~~Local disk~~	~~10 ms~~	~~1 TB~~
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

TLB
64 entries
per-core
1000 entries
per-core
shared
2 MB → 8 MB
10 TB
→ petabytes
NVRAM →
10µs, 10GB
100 GB → 10 TB
→ XB
→ 100 XB

i7 has 8MB as shared 3rd level cache; 2nd level cache is per-core

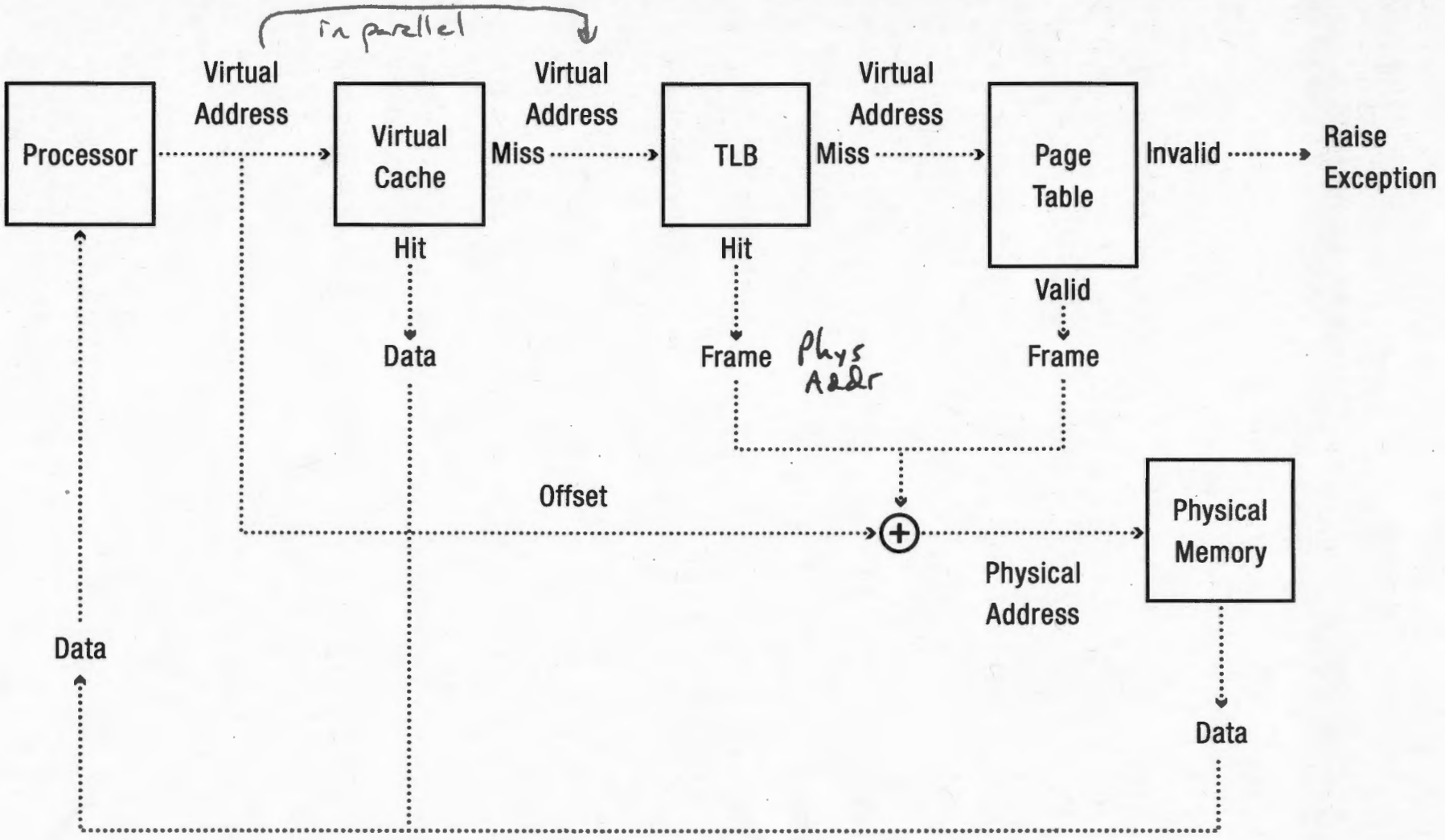
Question

- What is the cost of a first level TLB miss?
  - Second level TLB lookup
- What is the cost of a second level TLB miss?
  - 64 bit x86: 4-level page table walk
- How expensive is a 4-level page table walk?

Virtually Addressed vs. Physically Addressed Caches

- First level cache has at most a few cycles
  - Delays every instruction fetch and data reference
- Lookup TLB to get physical address, then lookup physical address in the cache?
  - Too slow!
- Instead, lookup virtual address in cache
- In parallel, lookup TLB in case of a cache miss

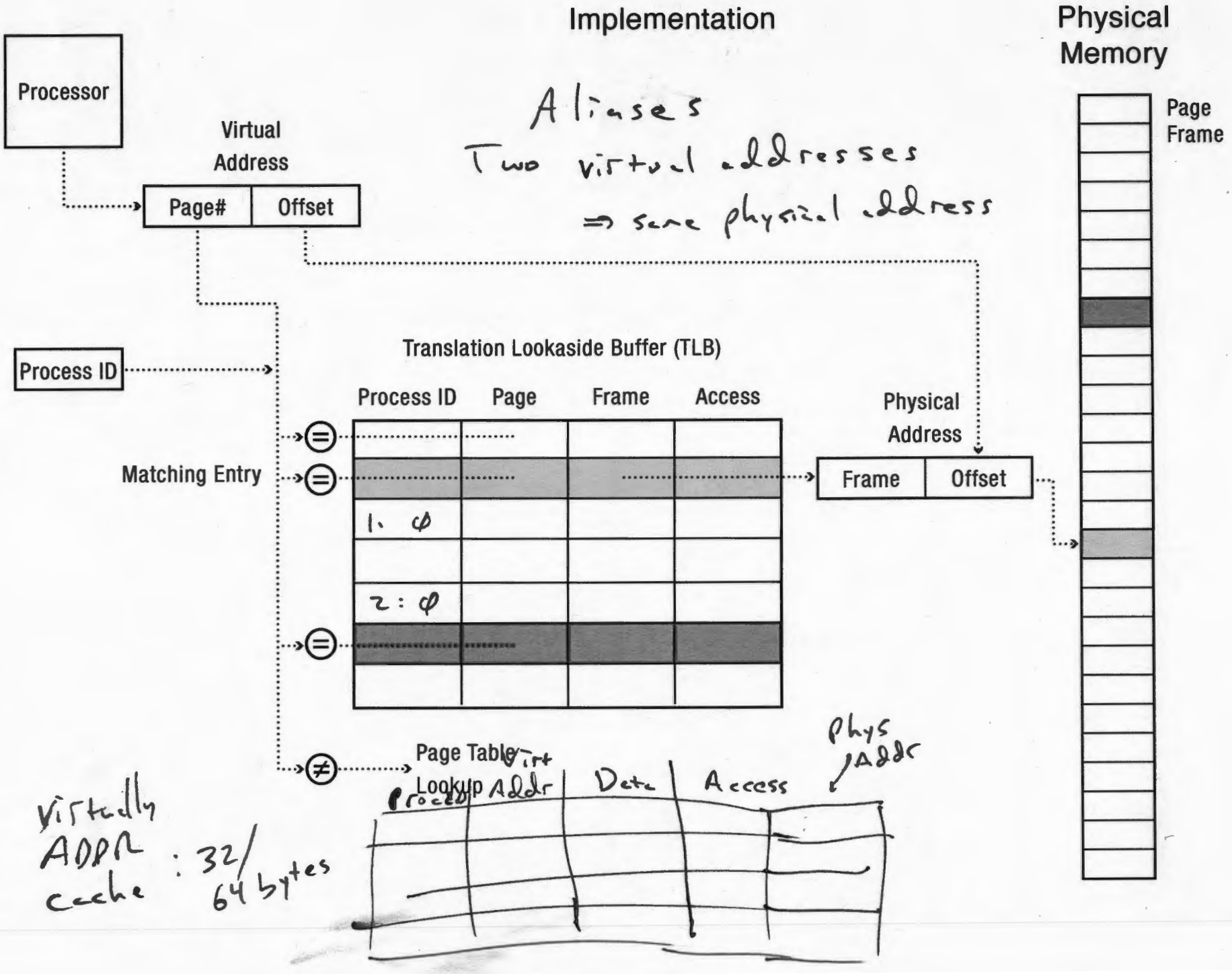
Virtually Addressed Caches



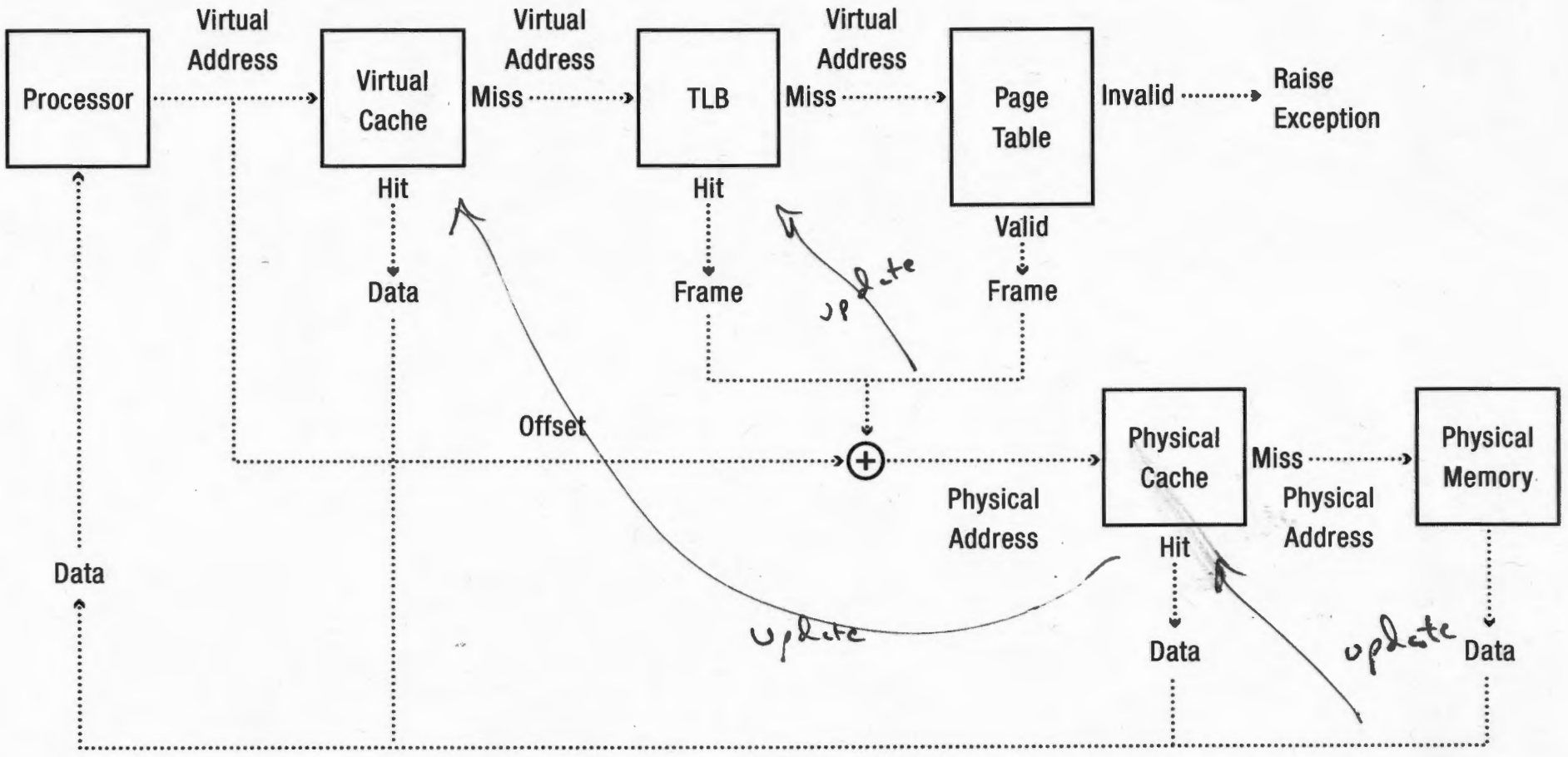
TLB & virtual cache: lookup virtual address

Question

- With a virtual cache, what do we need to do on a context switch?


Implementation
Physical Memory
Processor
Virtual Address
Page#
Offset
Aliases
Two virtual addresses
⇒ same physical address
Page Frame
Process ID
Translation Lookaside Buffer (TLB)
Process ID
Page
Frame
Access
Physical Address
Matching Entry
Frame
Offset
1: ϕ
2: ϕ
Page Table
Virt Addr
Lookup
Proceed
Data
Access
Phys Addr
Virtually Addr cache : 32/64 bytes

Physically Addressed Cache

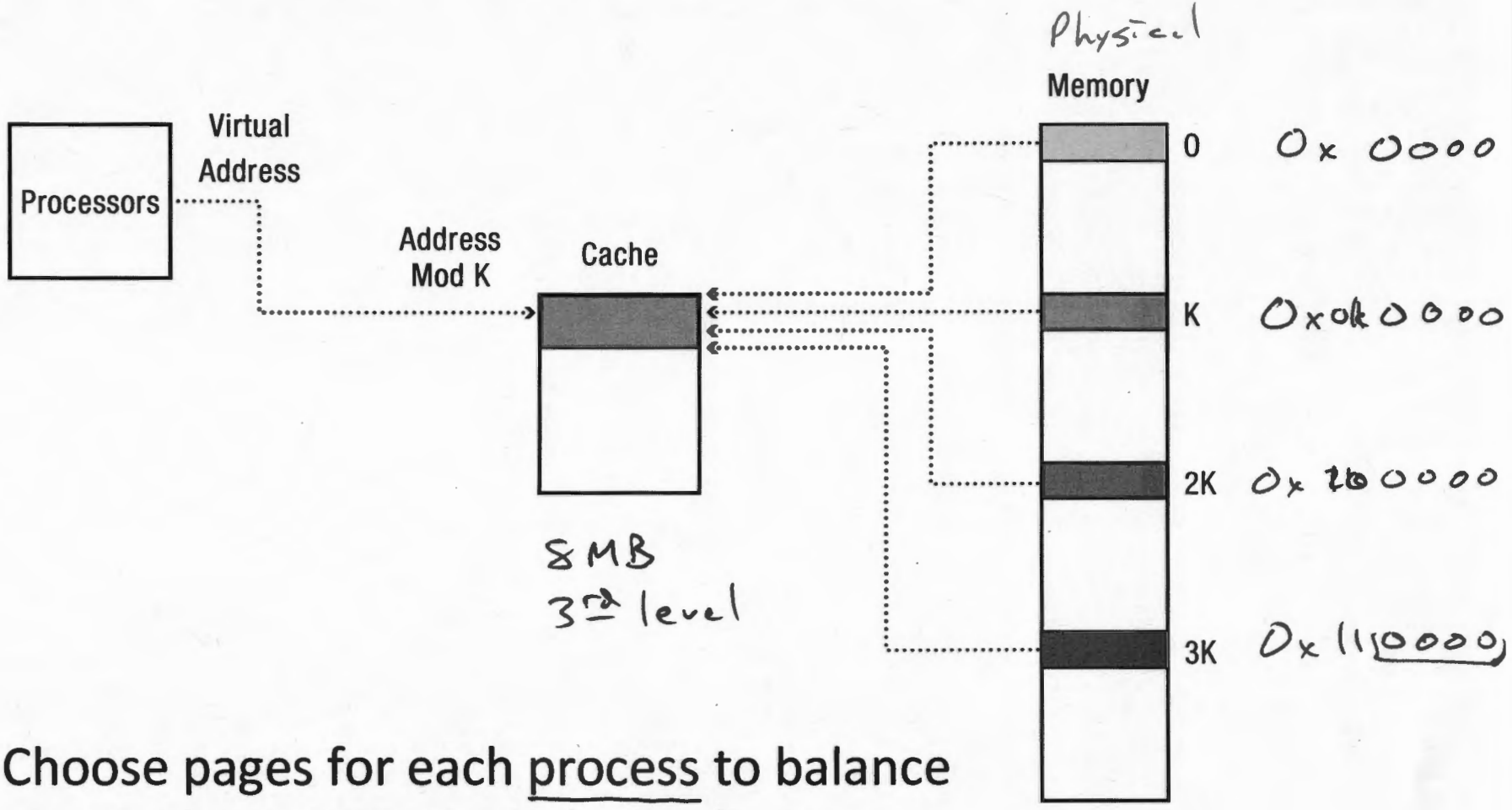


Physical cache: lookup physical address

Page Coloring

- What happens when cache size >> page size?
  - Direct mapped or set associative
  - Multiple pages map to the same cache line
- OS page assignment matters!
  - Example: 8MB cache, 4KB pages
  - 1 of every 2K pages lands in same place in cache
- What should the OS do?

Page Coloring



Choose pages for each process to balance usage across page types (colors)

TLB Size (Intel Kaby Lake, 2017)

First level TLB

- Instruction: 128 entries
- Data: 64 entries

Second level TLB

- 1536 entries

→ 2B page frames (!)

Modern server can have 10 TB (!) of DRAM

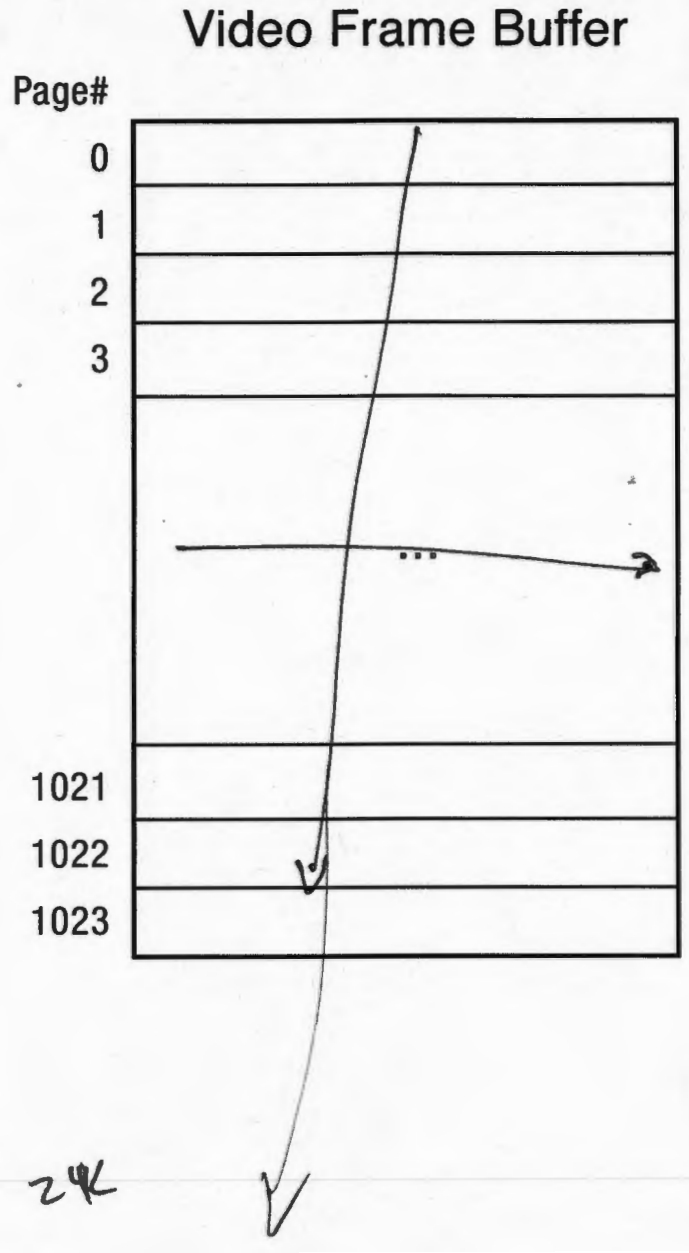
- 10-20% of server CPU time spent in TLB misses

When Do TLBs Work/Not Work?

Video Frame Buffer:
32 bits x 1K x 1K =
4MB = 1K pages

2017 laptop: 3K x 2K =
24MB = 6K pages

4K display: 4K x 3K =
48MB = 12K pages

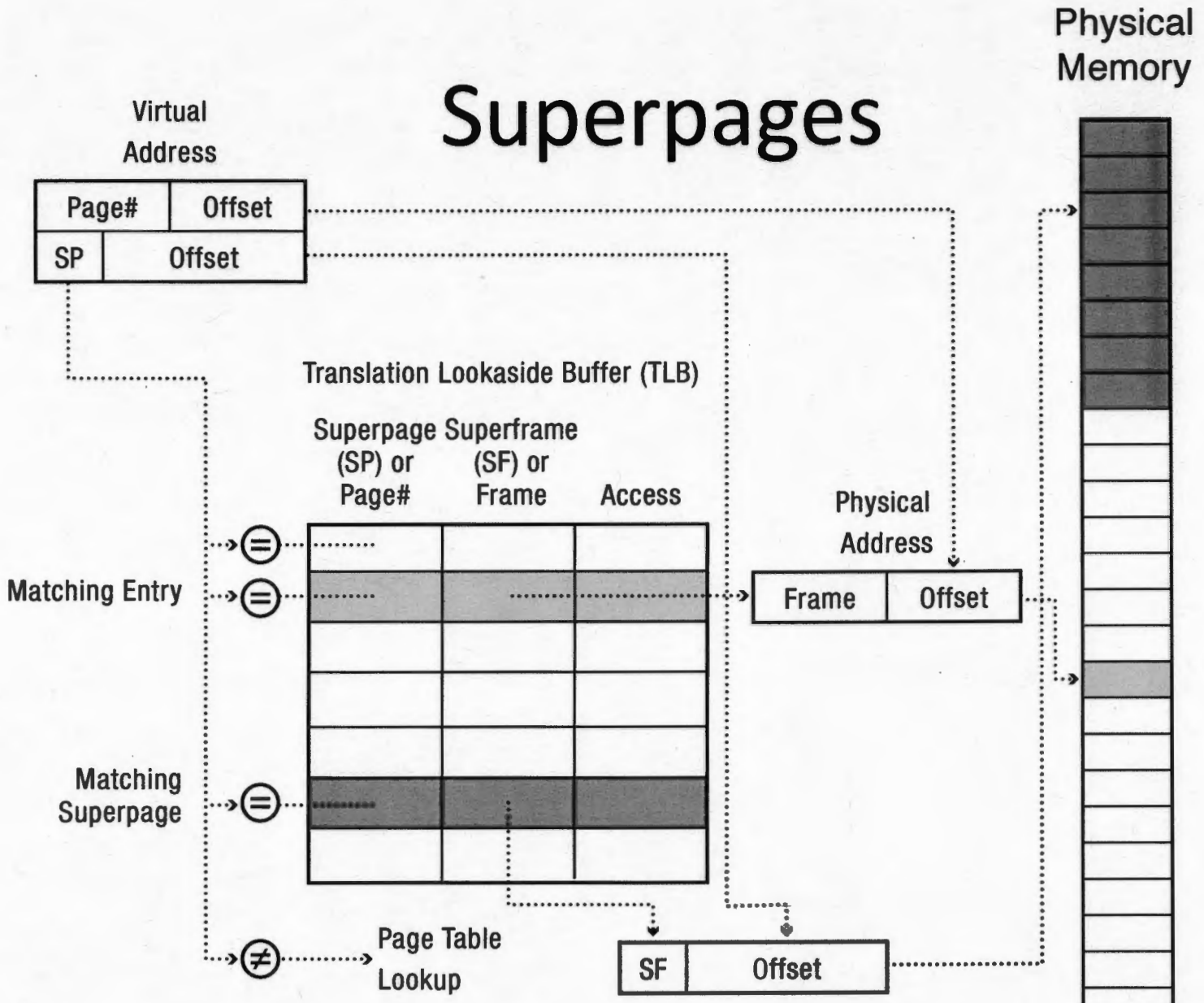


Superpages

- On x86 and ARM, TLB entry can be
  - A page
  - A superpage: a set of contiguous, aligned pages
- x86: superpage is set of pages in one page table
  - One page: 4KB
  - One page table: 2MB
  - One page table of page tables: 1GB
  - One page table of page tables of page tables: 0.5TB

ARM: superpages can be < full page table entry

Superpages



Kaby Lake has separate TLB for pages and superpages

When Do TLBs Work/Not Work, part 2

- What happens when the OS changes the permissions on a page?
  - For demand paging, copy on write, zero on reference, …
- TLB may contain old translation x86 → reset CR3
  - OS must ask hardware to purge TLB entry
- On a multicore: TLB shootdown
  - OS must ask each CPU to purge TLB entry

IPI

TLB Shootdown

Processor 1 TLB =

Process ID	VirtualPage	PageFrame	Access
0	0x0053	0x0003	R/W
1	0x40FF	0x0012	R/W

Processor 2 TLB =

0	0x0053	0x0003	R/W
0	0x0001	0x0005	Read

Processor 3 TLB =

1	0x40FF	0x0012	R/W
0	0x0001	0x0005	Read

Copy on write fork of process Φ ?

IPI - interprocessor interrupt

Virtual Cache Shootdown

- When permissions change for a page, we must shoot down the TLB entry on every CPU
- What about the contents of the virtual cache?
- Lazy shootdown of the virtual cache:
  - Lookup virtually addressed cache and TLB in *parallel*
  - Use the TLB to verify virtual address is still valid!
  - Evict entry from cache if not

Virtual Cache Aliases

- Alias: two (or more) virtual cache entries that refer to the same physical memory
  - A consequence of a tagged virtually addressed cache!
  - A write to one copy needs to update all copies
- Solution:
  - Virtual cache keeps both virtual and physical address for each entry
  - Lookup virtually addressed cache and TLB in *parallel*
  - Check if physical address from TLB matches any other entries, and update/invalidate those copies

x86 caches

- 64 byte line size
- Physically indexed
- Physically tagged
- Write buffer