Introduction to Operating Systems

CPSC/ECE 3220 Spring 2020

Lecture Notes
OSPP Chapter 8 – Part B

(adapted by Mark Smotherman from publisher's web site)

Efficient Address Translation

- Translation Lookaside Buffer (TLB)
  - Cache of recent virtual page -> physical page translations with permissions, in hardware
  - If cache hit, use that translation
  - If cache miss, walk multi-level page table
- Caches are expensive
- Static on-chip memory
  - Performance gain should be worth it
  - TLB lookup needs to be faster than full address translation
  - Some use multiulevel TLBs

What is Virtual Memory?

https://www.youtube.com/watch?v=2quKyPnUShQ

Cost of address Translation

Cost of translation =

Cost of TLB lookup (whether in TLB or not +

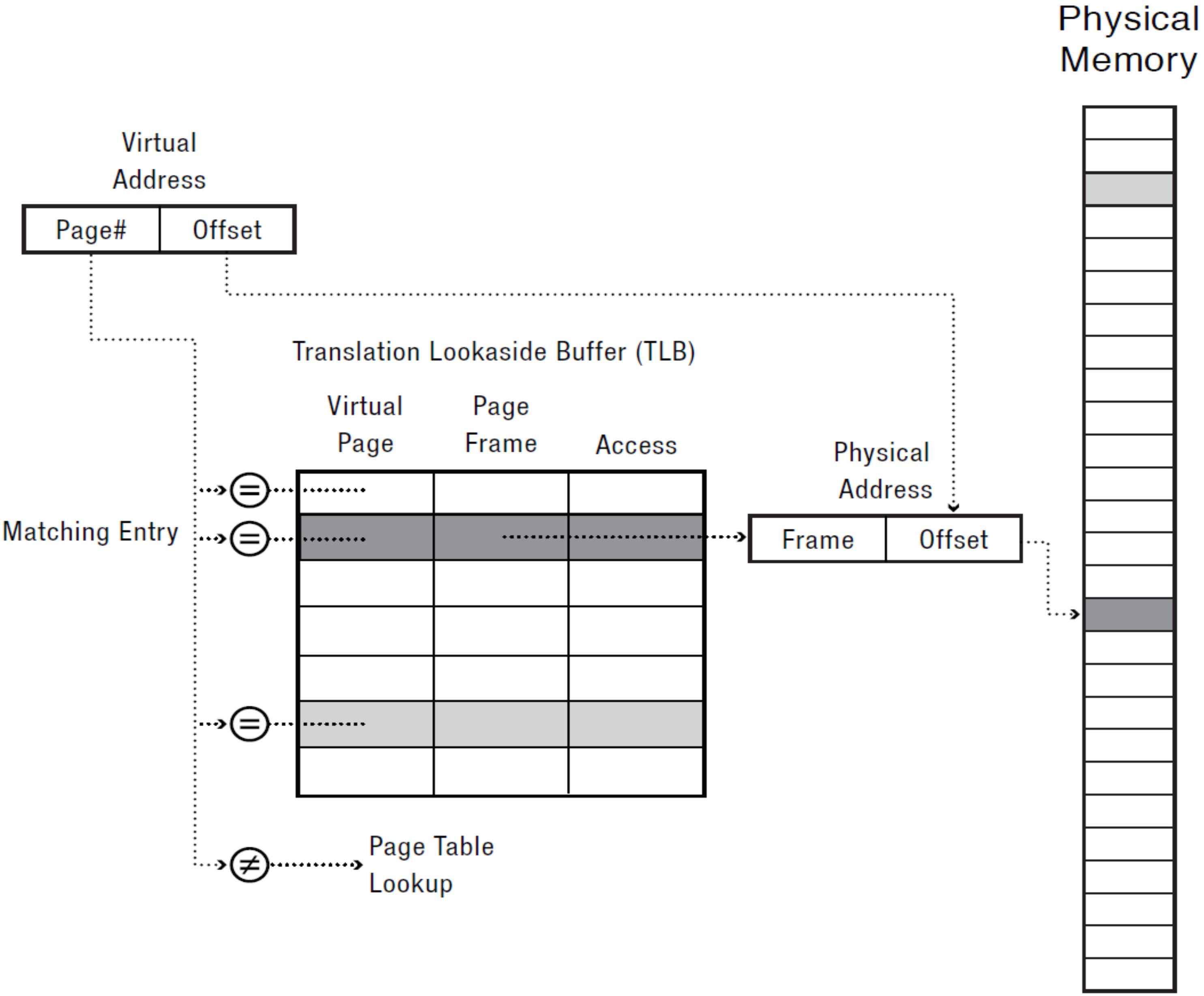
Cost of full translation (if not in TLB) *

Prob(TLB miss)

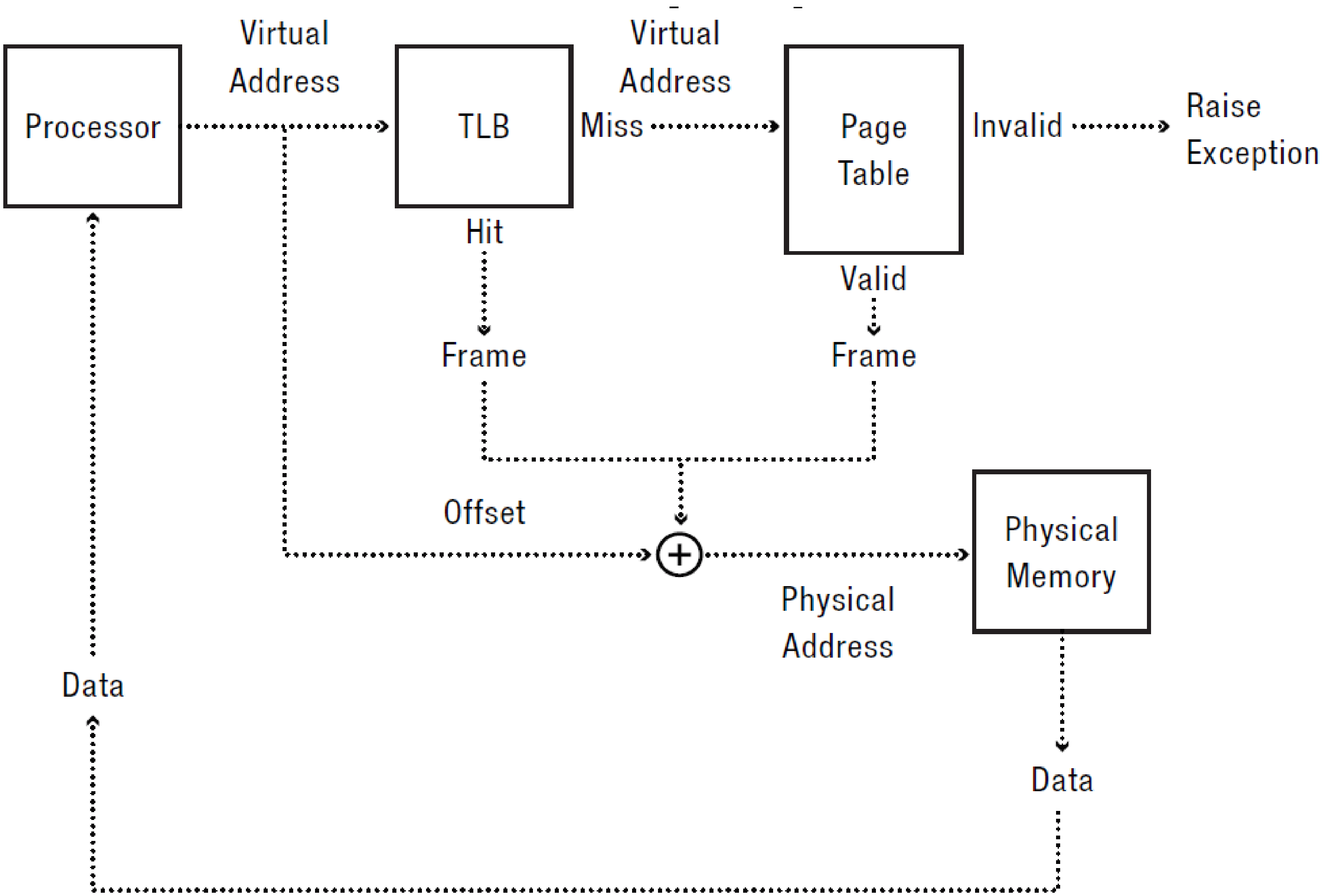
Prob(TLB miss) = 1 - Prob(TLB hit)

Design consideration: full address translation should be a rare event -> TLB should be sufficiently large

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 and Page Table



Hardware Design Principle

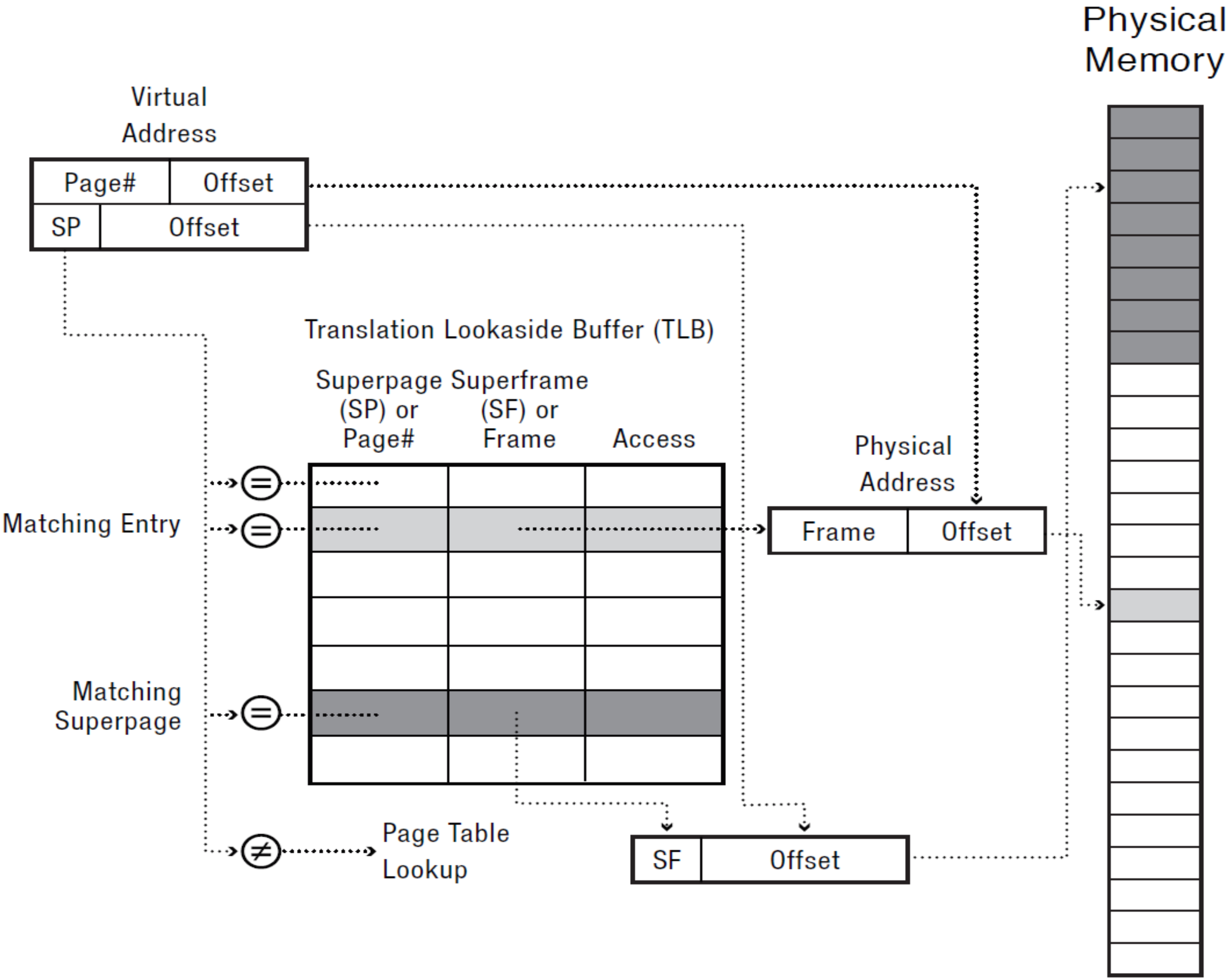
The bigger the memory, the slower the memory.

Why?

Superpages

- Improve TLB hit rate
- On many systems, TLB entry can be
  - A page
  - A superpage:
    - a set of contiguous pages in VM that map to contiguous pages in PM, aligned to share same high order address
  - TLB will have a flag to indicate a superpage
- Superpages are at the discretion of the OS.
- x86: superpage is a set of pages in one page table
  - x86 TLB entries
    - 4KB, 2MB, 1GB

Superpages


Physical Memory
Virtual Address
Page#
Offset
SP
Offset
Translation Lookaside Buffer (TLB)
Superpage (SP) or Page#
Superframe (SF) or Frame
Access
Matching Entry
Matching Superpage
Physical Address
Frame
Offset
Page Table Lookup
SF
Offset

TLBs Consistency

- TLB needs to be in the consistent state
- TLB may contain old translation at context switch
- On every context switch – change hw page table register to point to new process’ page table
- And *flush* TLB entry
  - Carries penalty

Tagged TLBs

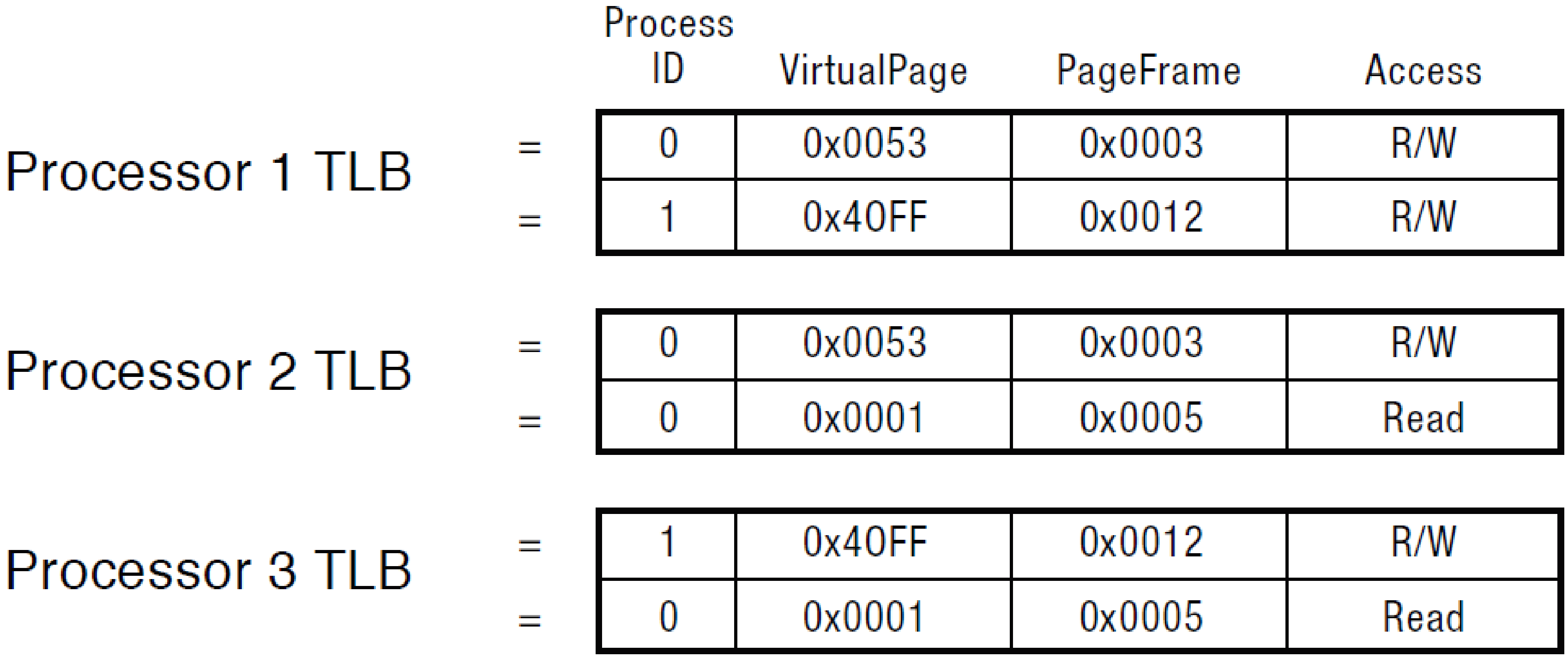
- What happens on a context switch?
  - Reuse TLB?
  - Discard TLB?

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

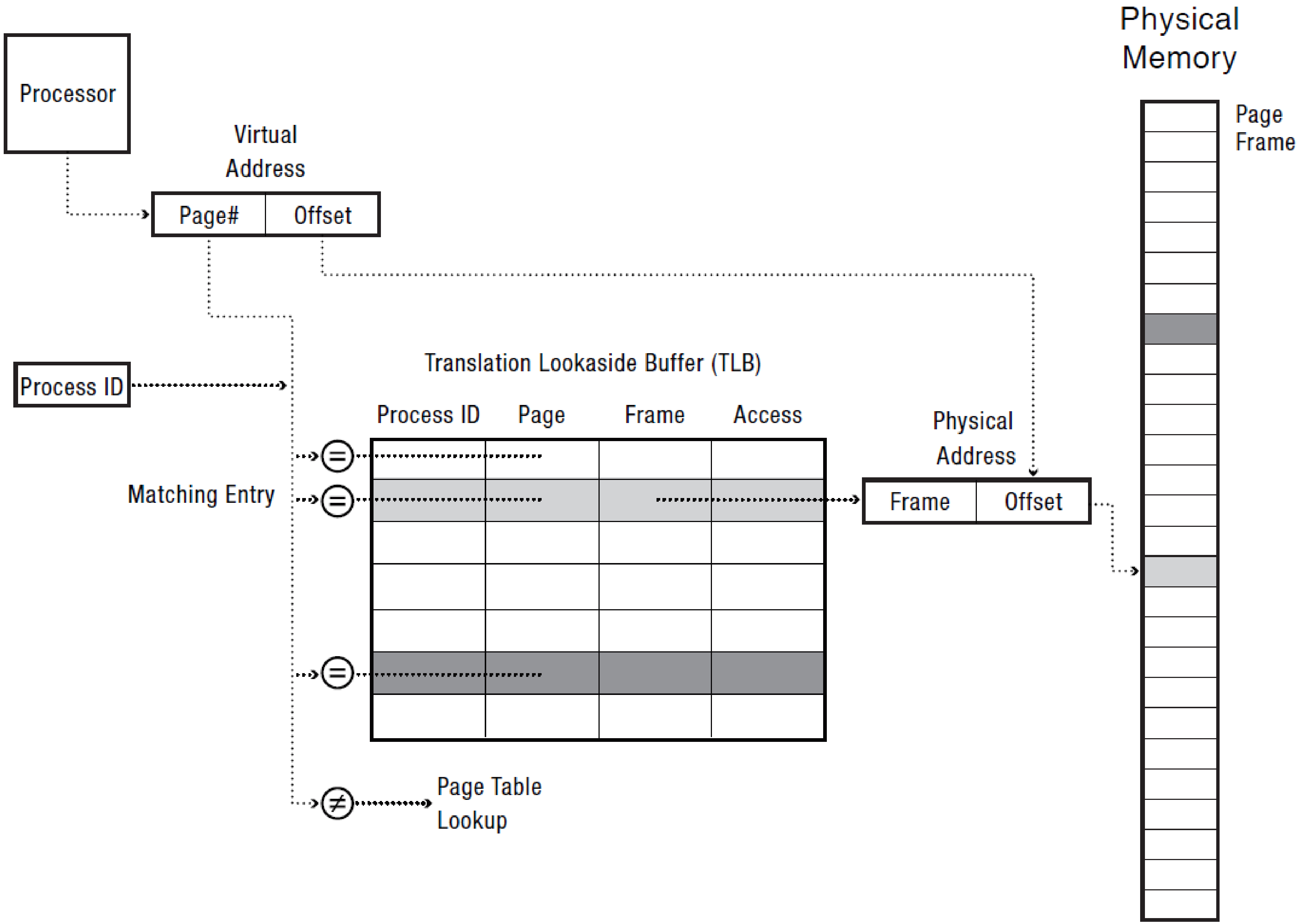
Multiprocessor TLB

- Any or all processors may have a cached copy of TLB
- Every time entry is modified, need to remove TLB from every processor
- OS interrupts every processor to remove old TLB entry – TLB shootdown
- Original processor continue executing *after* other processors remove entry

TLB Shootdown



		Process ID	VirtualPage	PageFrame	Access
Processor 1 TLB	=	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



Physical
Memory
Page
Frame
Processor
Virtual
Address
Page#
Offset
Process ID
Translation Lookaside Buffer (TLB)
Process ID
Page
Frame
Access
Physical
Address
Frame
Offset
Matching Entry
=
=
=
≠
Page Table
Lookup

Address Translation Uses

- Process isolation
  - Keep a process from touching anyone else's memory, or the kernel's
- Efficient interprocess communication
  - Shared regions of memory between processes
- Shared code segments
  - E.g., common libraries used by many different programs
- Program initialization
  - Start running a program before it is entirely in memory
- Dynamic memory allocation
  - Allocate and initialize stack/heap pages on demand

Address Translation Uses (2)

- Program debugging
  - Data breakpoints when address is accessed
- Memory mapped files
  - Access file data using load/store instructions
- Demand-paged virtual memory
  - Illusion of near-infinite memory, backed by disk or memory on other machines
- Zero-copy I/O
  - Directly from I/O device into/out of user memory
- Efficient support of virtual machines

Address Translation Uses (3)

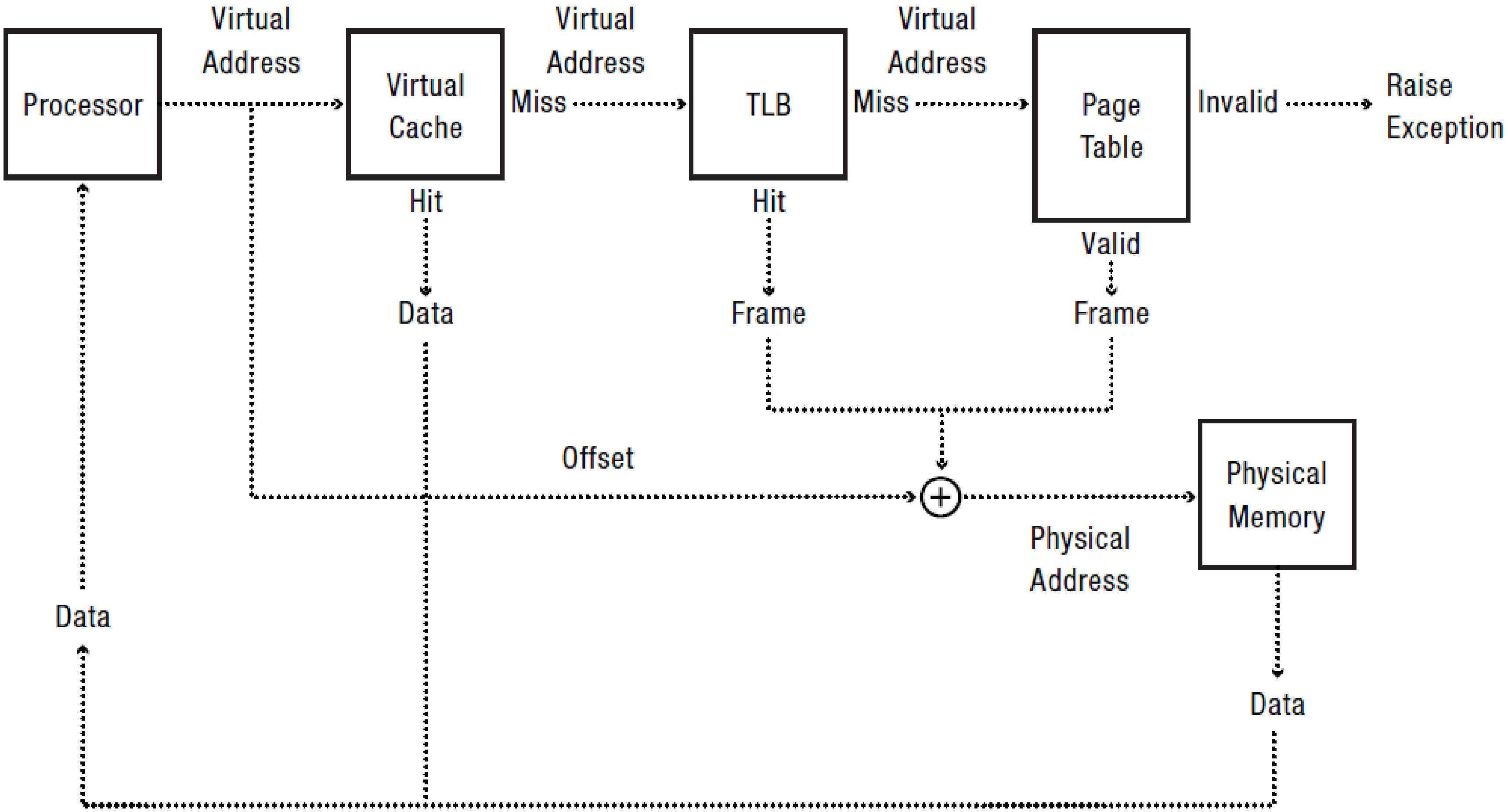
- Checkpointing/restart
  - Transparently save a copy of a process, without stopping the program while the save happens
- Persistent data structures
  - Implement data structures that can survive system reboots
- Process migration
  - Transparently move processes between machines
- Information flow control
  - Track what data is being shared externally
- Distributed shared memory
  - Illusion of memory that is shared between machines

(if time permits)

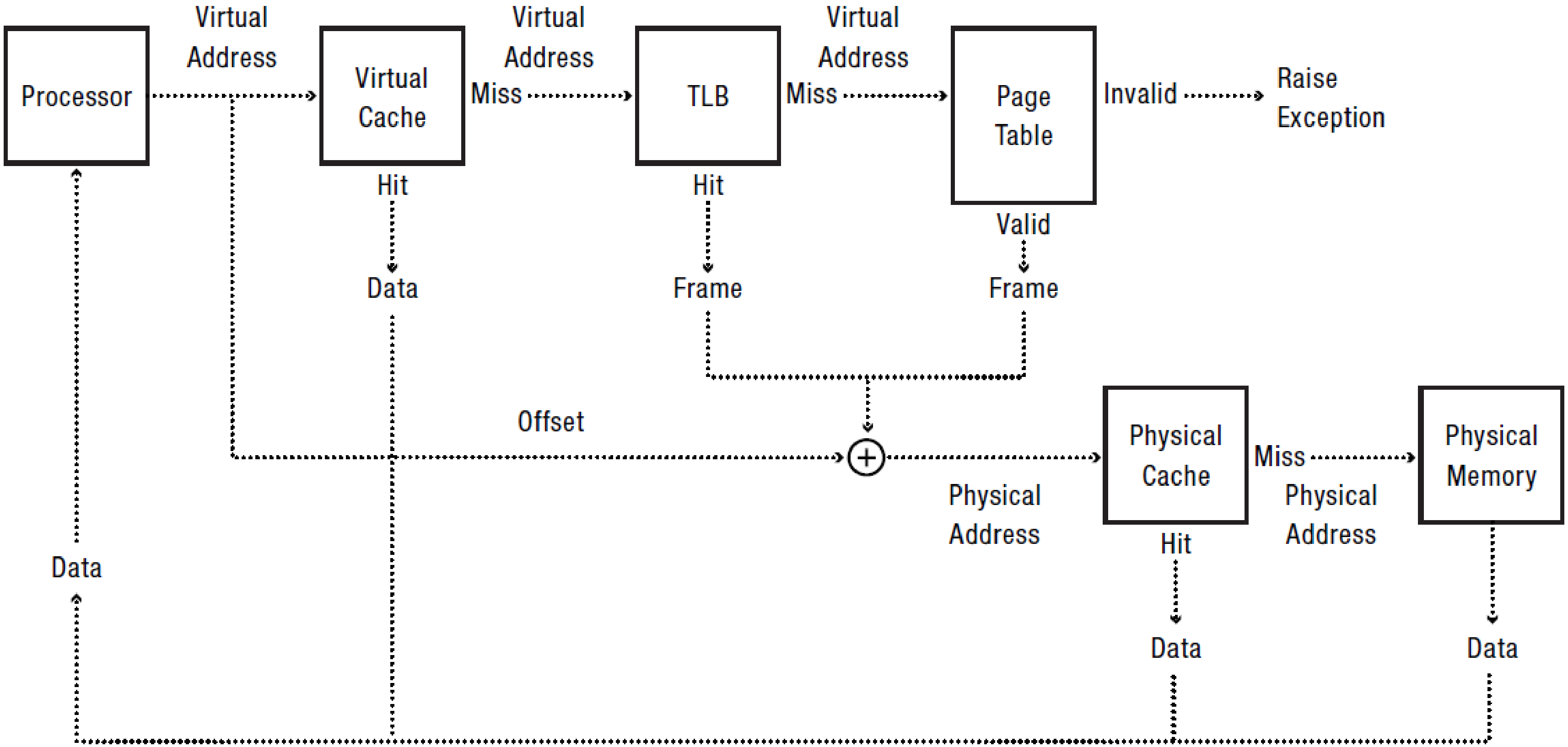
Virtually Addressed vs. Physically Addressed Caches

- Too slow to first access TLB to find physical address, then look up address in the cache
- Instead, first level cache is virtually addressed
- In parallel, access TLB to generate physical address in case of a cache miss

Virtually Addressed Caches



Physically Addressed Cache



Question

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