

Address Translation

Main Points

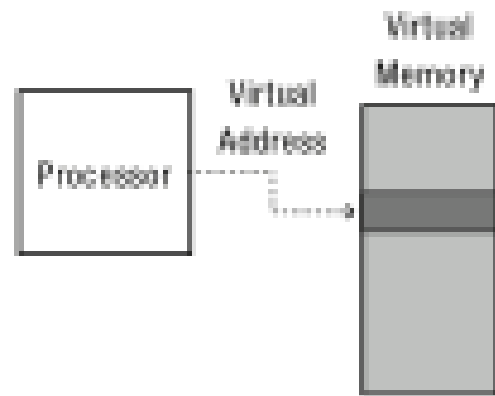
- Address Translation Concept
 - How do we convert a virtual address to a physical address?
- Flexible Address Translation
 - Base and bound
 - Segmentation
 - Paging
 - Multilevel translation
- Efficient Address Translation
 - Translation Lookaside Buffers
 - Virtually and physically addressed caches

Address Translation Goals

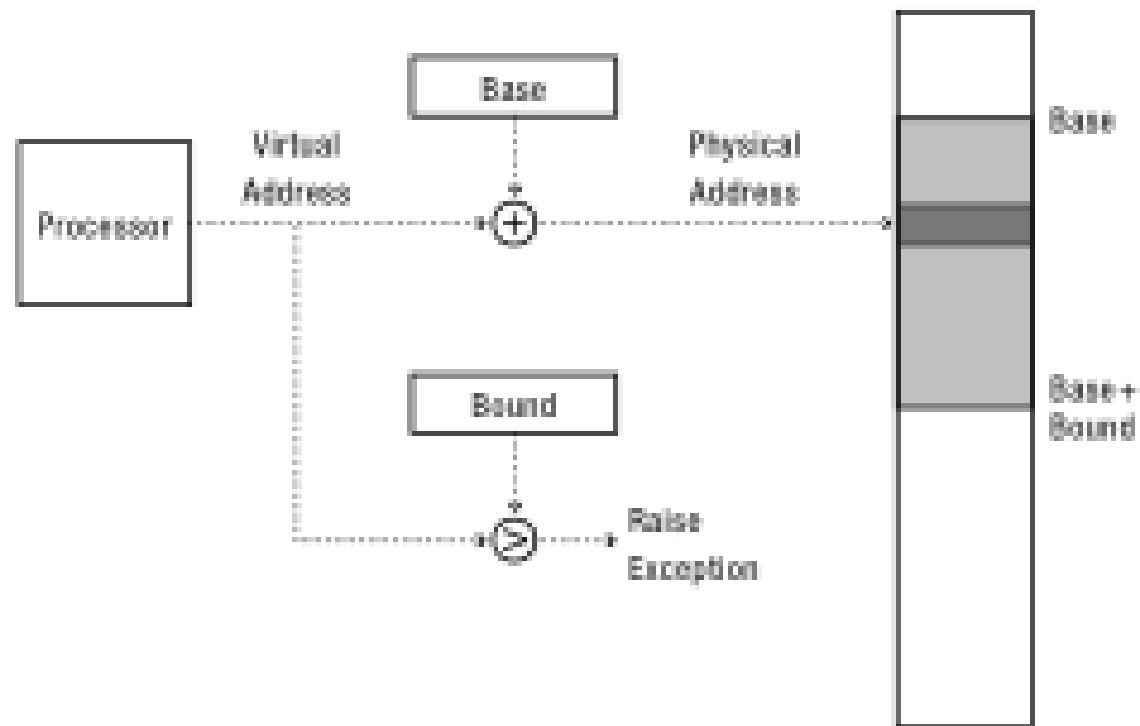
- Memory protection
- Memory sharing
 - Shared libraries, interprocess communication
- Sparse addresses
 - Multiple regions of dynamic allocation (heaps/stacks)
- Efficiency
 - Memory placement
 - Runtime lookup
 - Compact translation tables

Virtually Addressed Base and Bounds

Processor's View



Implementation



Activity #1

- Drawback of Base and Bounds

Virtually Addressed Base and Bounds

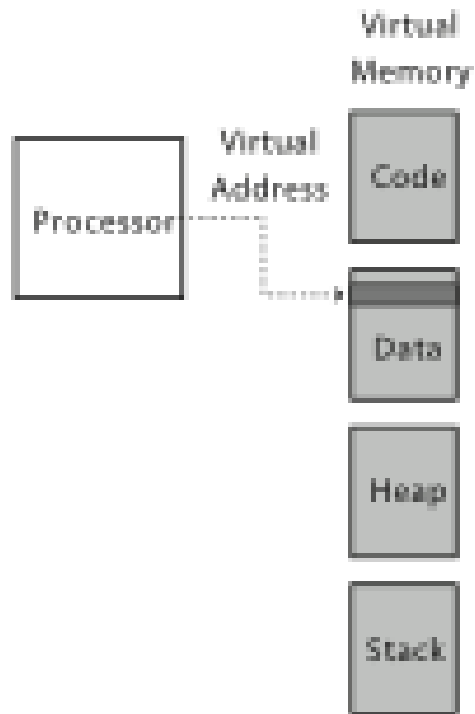
- Pros?
 - Simple
 - Fast (2 registers, adder, comparator)
 - Safe
 - Can relocate in physical memory without changing process
- Cons?
 - Can't keep program from accidentally overwriting its own code
 - Can't share code/data with other processes
 - Can't grow stack/heap as needed

Segmentation

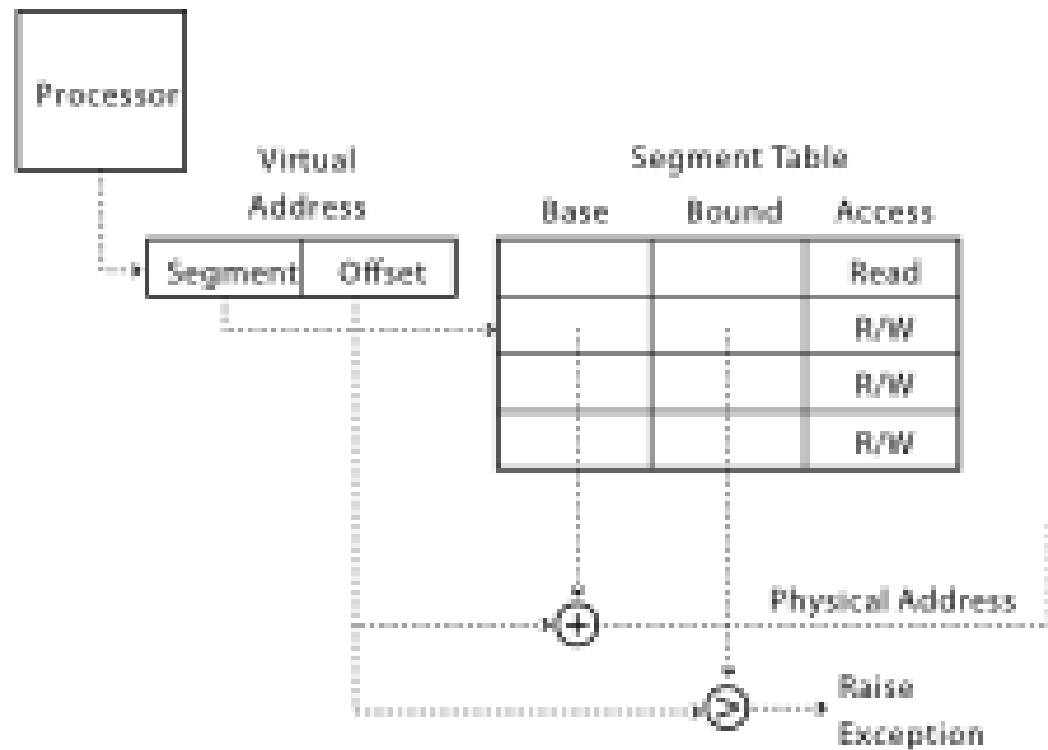
- Segment is a contiguous region of *virtual* memory
- Each process has a segment table (in hardware)
 - Entry in table = segment
- Segment can be located anywhere in physical memory
 - Each segment has: start, length, access permission
- Processes can share segments
 - Same start, length, same/different access permissions

Segmentation

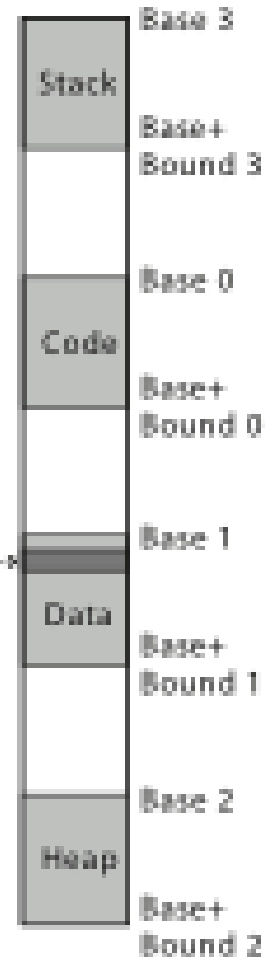
Processor's View



Implementation



Physical Memory

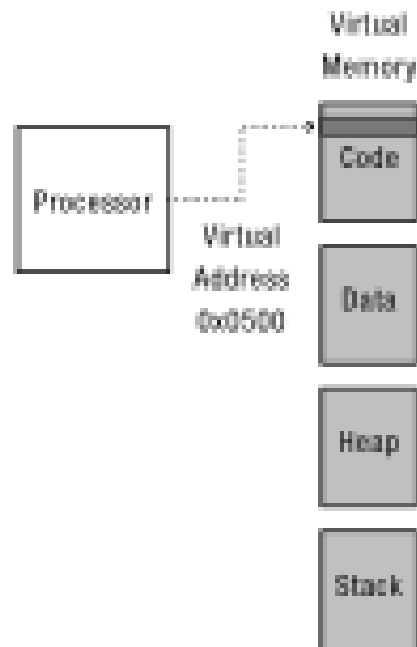


	Segment	start	length	
2 bit segment #	code	0x4000	0x700	
12 bit offset	data	0	0x500	
	heap	-	-	
Virtual Memory	stack	0x2000	0x1000	Physical Memory

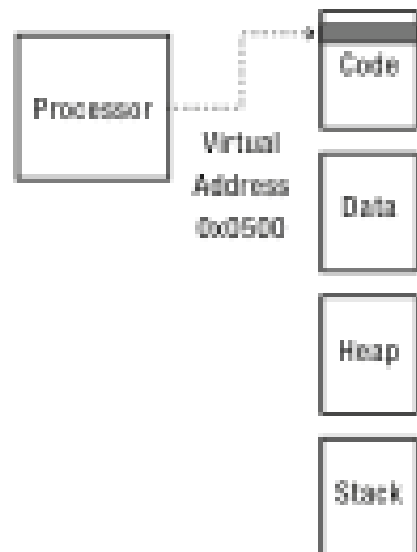
main: 240	store #1108, r2	x: 108	a b c \0
244	store pc+8, r31	...	
248	jump 360	main: 4240	store #1108, r2
24c		4244	store pc+8, r31
...		4248	jump 360
strlen: 360	loadbyte (r2), r3	424c	
...
420	jump (r31)	strlen: 4360	loadbyte (r2),r3
...		...	
x: 1108	a b c \0	4420	jump (r31)
...		...	

Processor's View

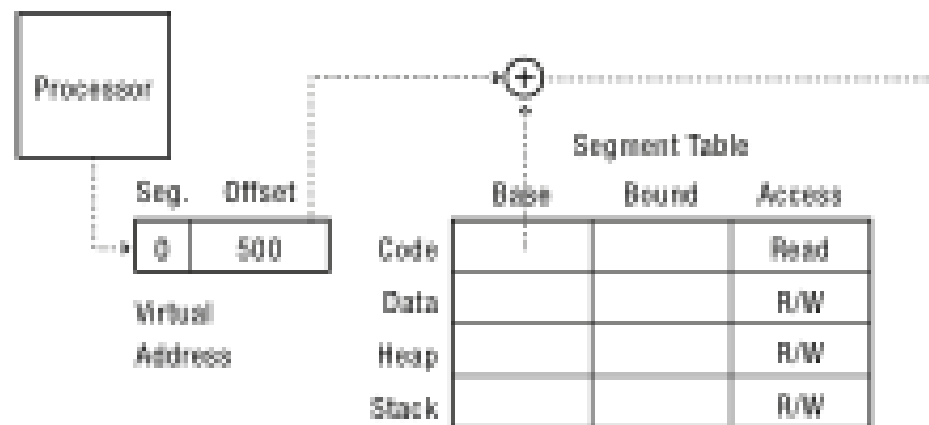
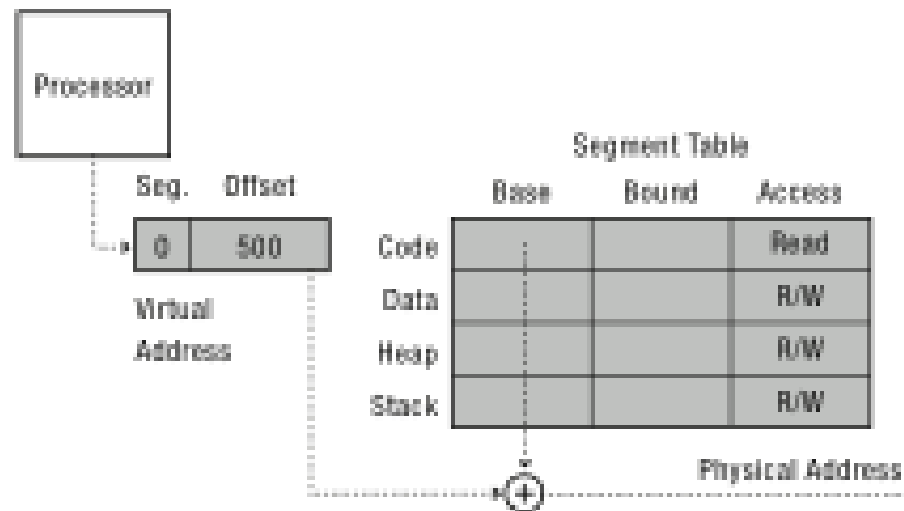
Process 1's View



Process 2's View



Implementation



Physical Memory



Activity #2

- Drawback of Segmentation

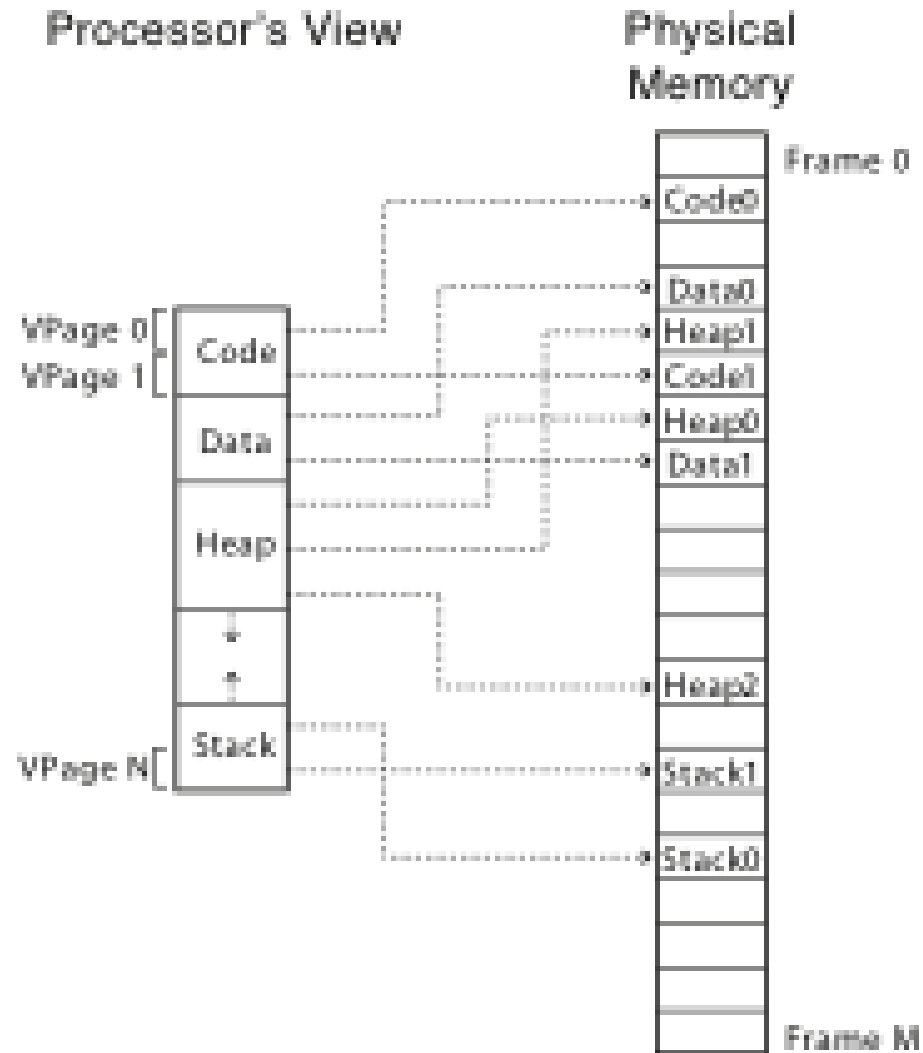
Segmentation

- Pros?
 - Can share code/data segments between processes
 - Can protect code segment from being overwritten
 - Can transparently grow stack/heap as needed
 - Can detect if need to copy-on-write
- Cons?
 - Complex memory management
 - Need to find chunk of a particular size
 - May need to rearrange memory from time to time to make room for new segment or growing segment
 - External fragmentation: wasted space between chunks

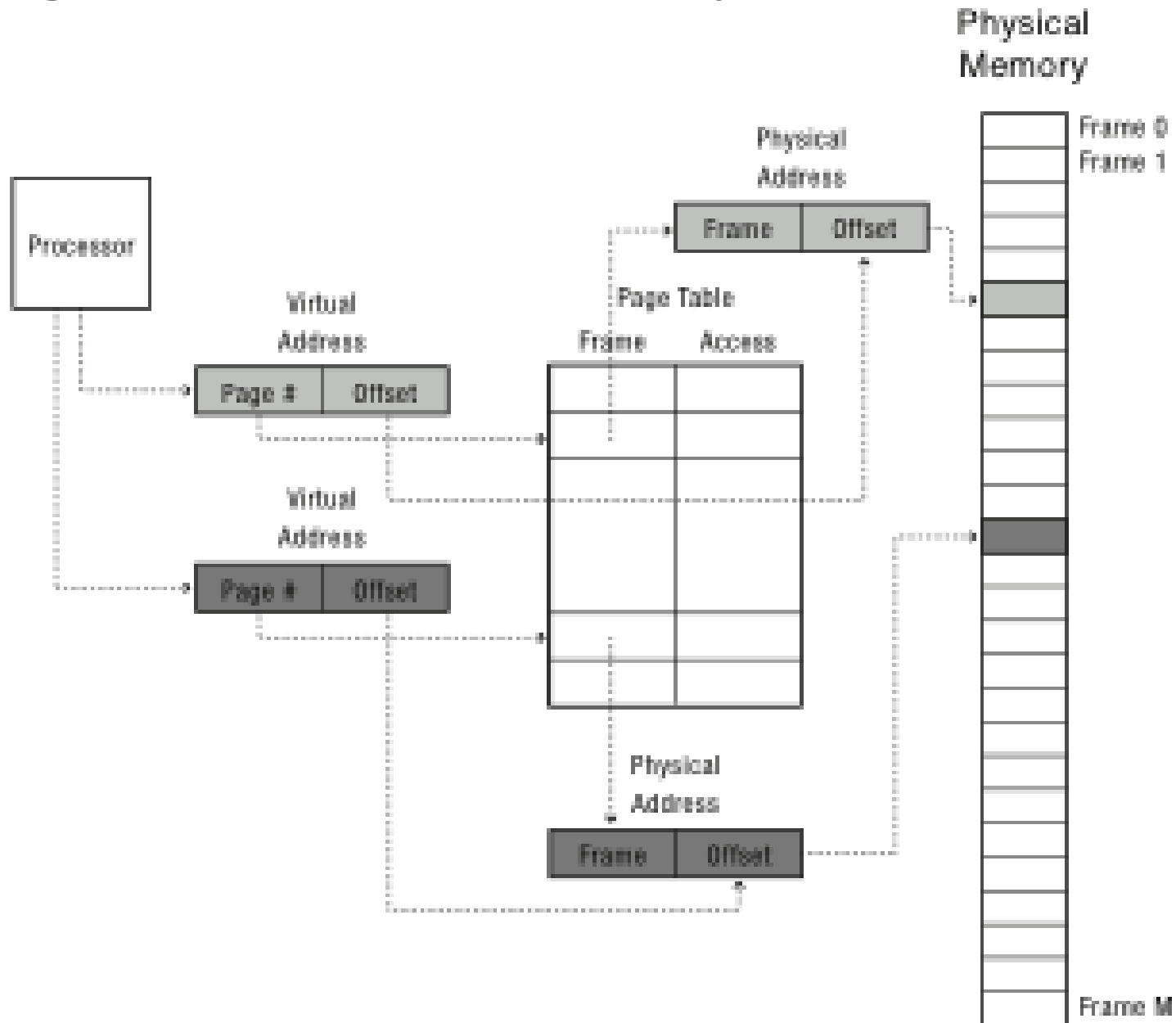
Paged Translation

- Manage memory in fixed size units, or pages
- Finding a free page is easy
 - Bitmap allocation: 00111111000000001100
 - Each bit represents one physical page frame
- Each process has its own page table
 - Stored in physical memory
 - Hardware registers
 - pointer to page table start
 - page table length

Paged Translation (Abstract)



Paged Translation (Implementation)



Process View

A
B
C
D
E
F
G
H
I
J
K
L

Physical Memory

I
J
K
L
E
F
G
H
A
B
C
D

Page Table

4
3
1

Activity #3

- Drawback of paging

Sparse Address Spaces

- Might want many separate dynamic segments
 - Per-processor heaps
 - Per-thread stacks
 - Memory-mapped files
 - Dynamically linked libraries
- What if virtual address space is large?
 - 32-bits, 4KB pages => 500K page table entries
 - 64-bits => 4 quadrillion page table entries

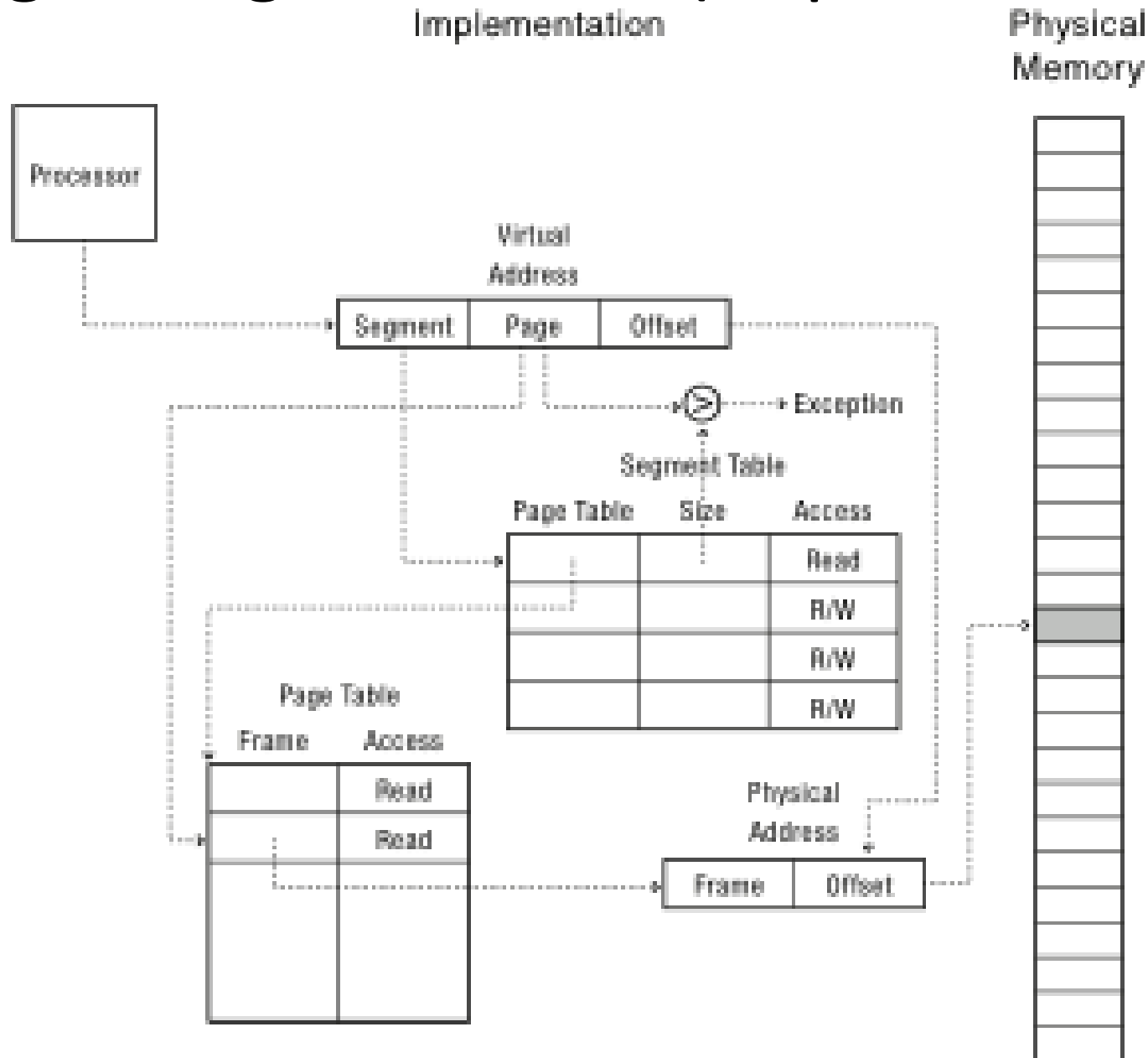
Multi-level Translation

- Tree of translation tables
 - Paged segmentation
 - Multi-level page tables
 - Multi-level paged segmentation
- Fixed-size page as lowest level unit of allocation
 - Efficient memory allocation (compared to segments)
 - Efficient for sparse addresses (compared to paging)
 - Efficient disk transfers (fixed size units)
 - Easier to build translation lookaside buffers
 - Efficient reverse lookup (from physical -> virtual)
 - Variable granularity for protection/sharing

Paged Segmentation

- Process memory is segmented
- Segment table entry:
 - Pointer to page table
 - Page table length (# of pages in segment)
 - Access permissions
- Page table entry:
 - Page frame
 - Access permissions
- Share/protection at either page or segment-level

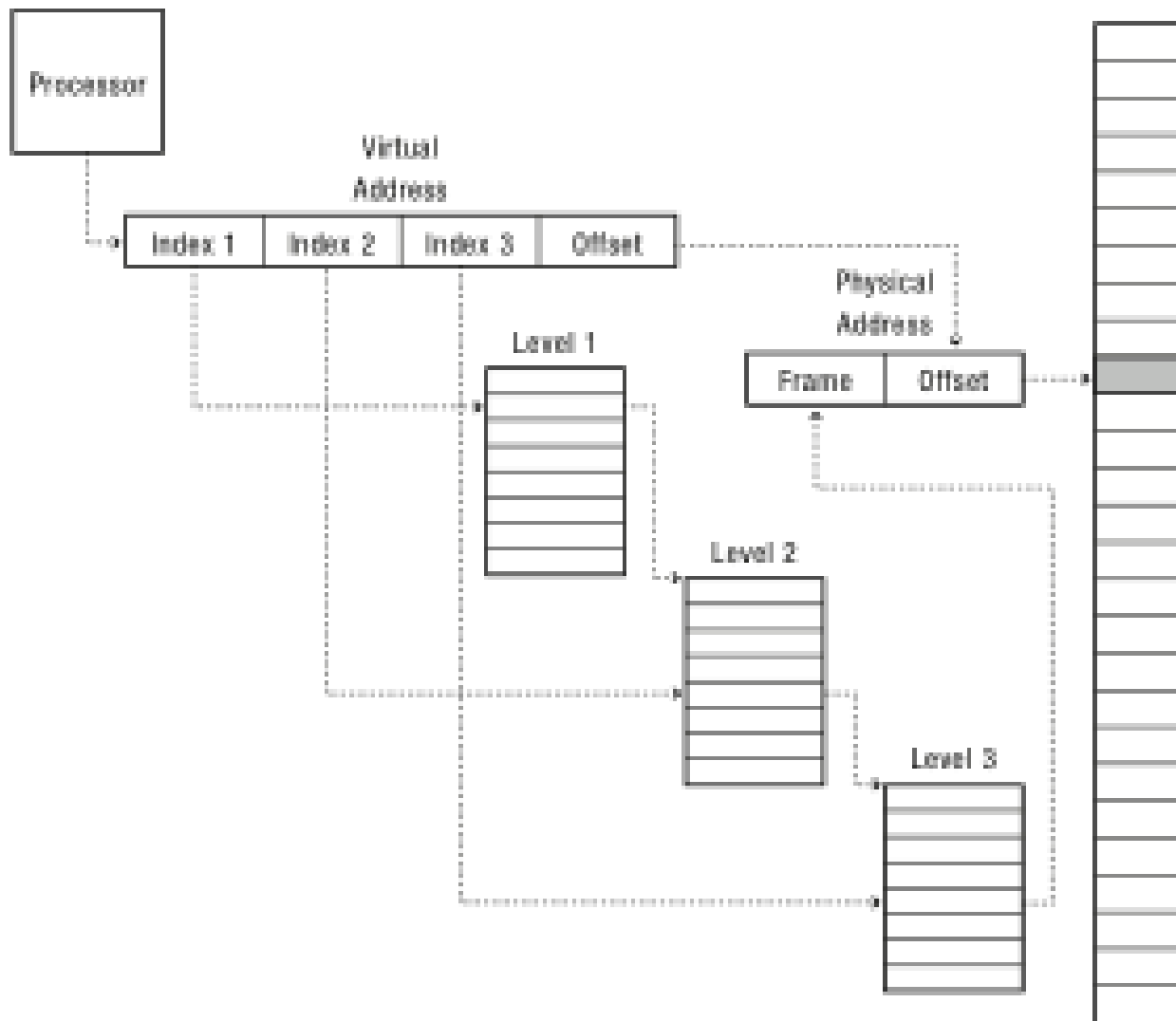
Paged Segmentation (Implementation)



Multilevel Paging

Implementation

Physical
Memory



Activity #4

- Drawback of Multilevel translation

Multilevel Translation

- Pros:
 - Allocate/fill only page table entries that are in use
 - Simple memory allocation
 - Share at segment or page level
- Cons:
 - Space overhead: one pointer per virtual page
 - Two (or more) lookups per memory reference

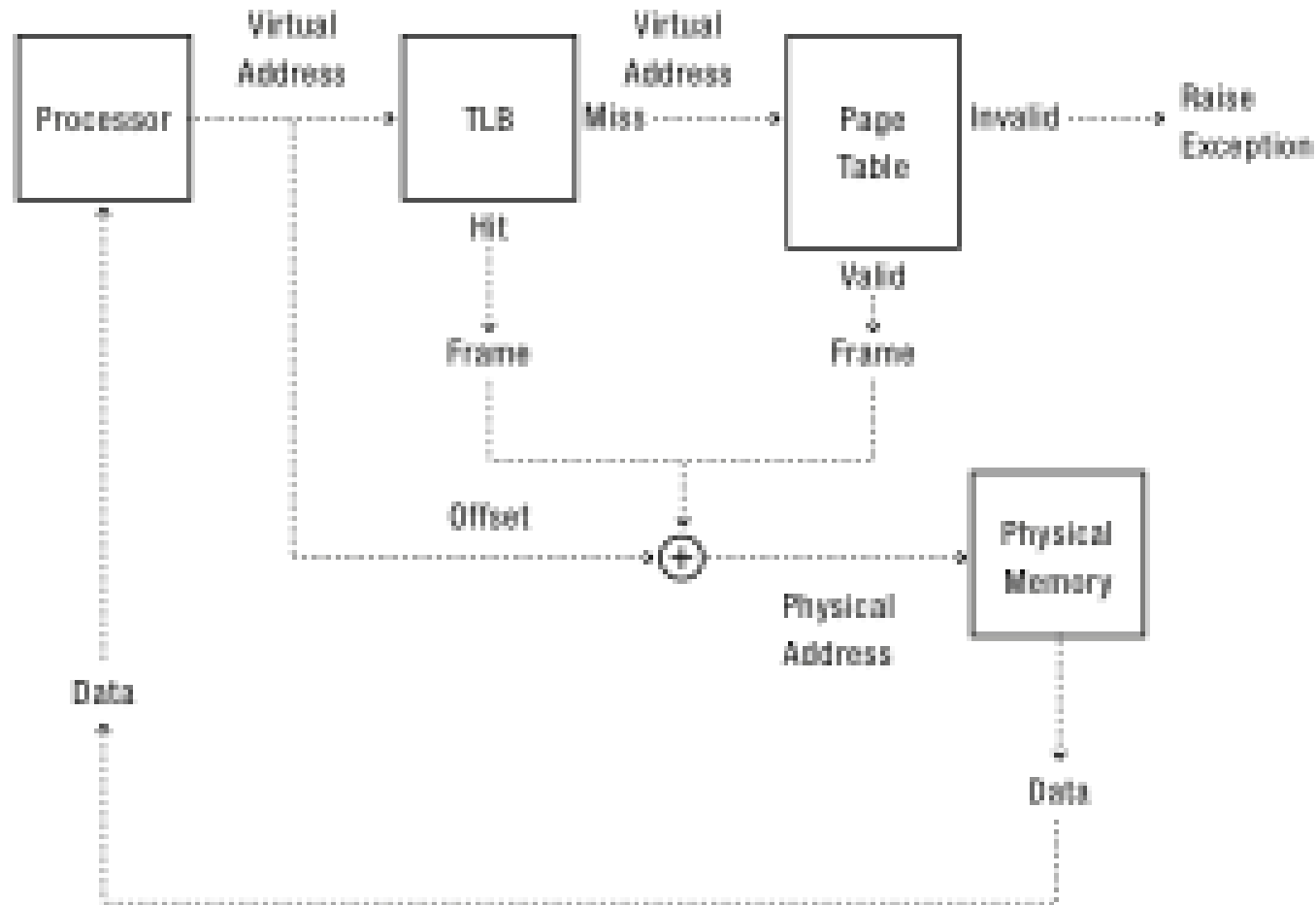
Activity #5

- Accelerate multilevel translation
 - Possible?
 - How?

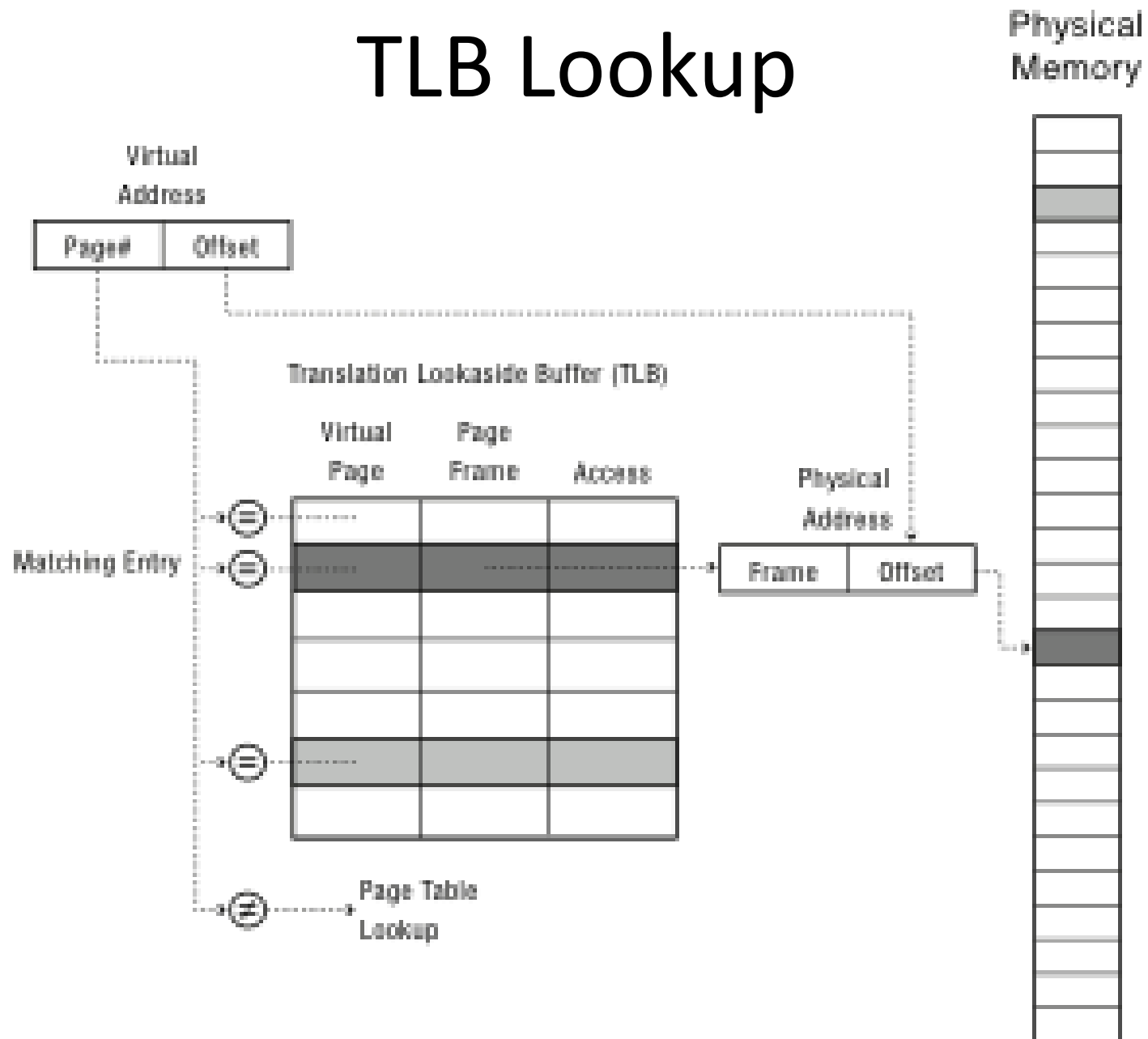
Efficient Address Translation

- Translation lookaside buffer (TLB)
 - Cache of recent virtual page \rightarrow physical page translations
 - If cache hit, use translation
 - If cache miss, walk multi-level page table
- Cost of translation =
Cost of TLB lookup +
 $\text{Prob}(\text{TLB miss}) * \text{cost of page table lookup}$

TLB and Page Table Translation



TLB 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 (more)

- Cache management
 - Page coloring
- Program debugging
 - Data breakpoints when address is accessed
- Zero-copy I/O
 - Directly from I/O device into/out of user memory
- 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