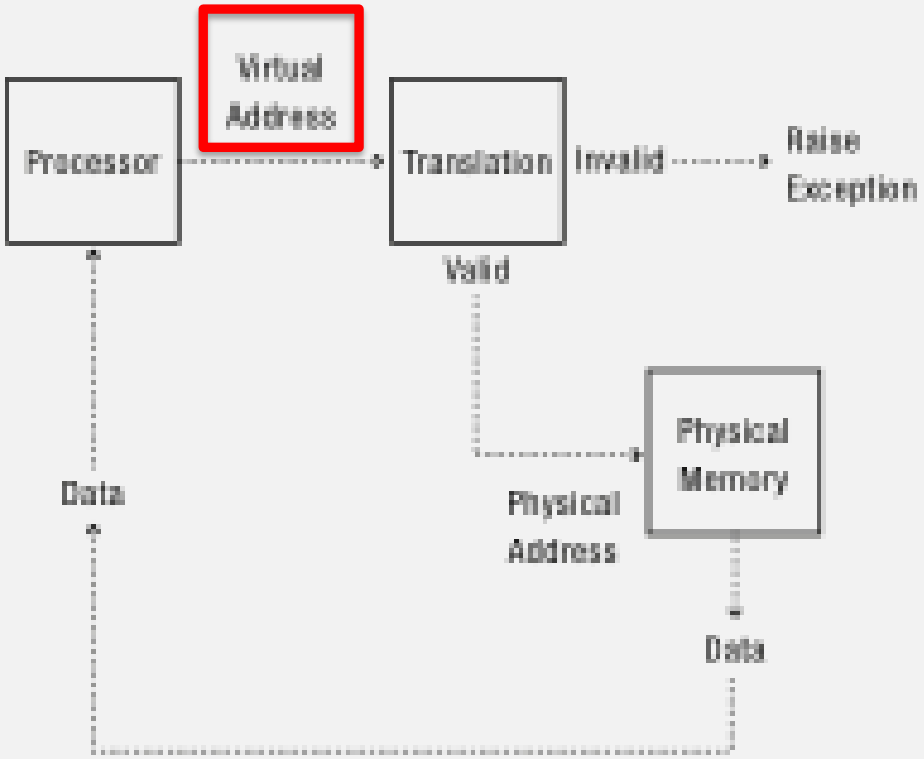
# Address Translation

# 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 Concept



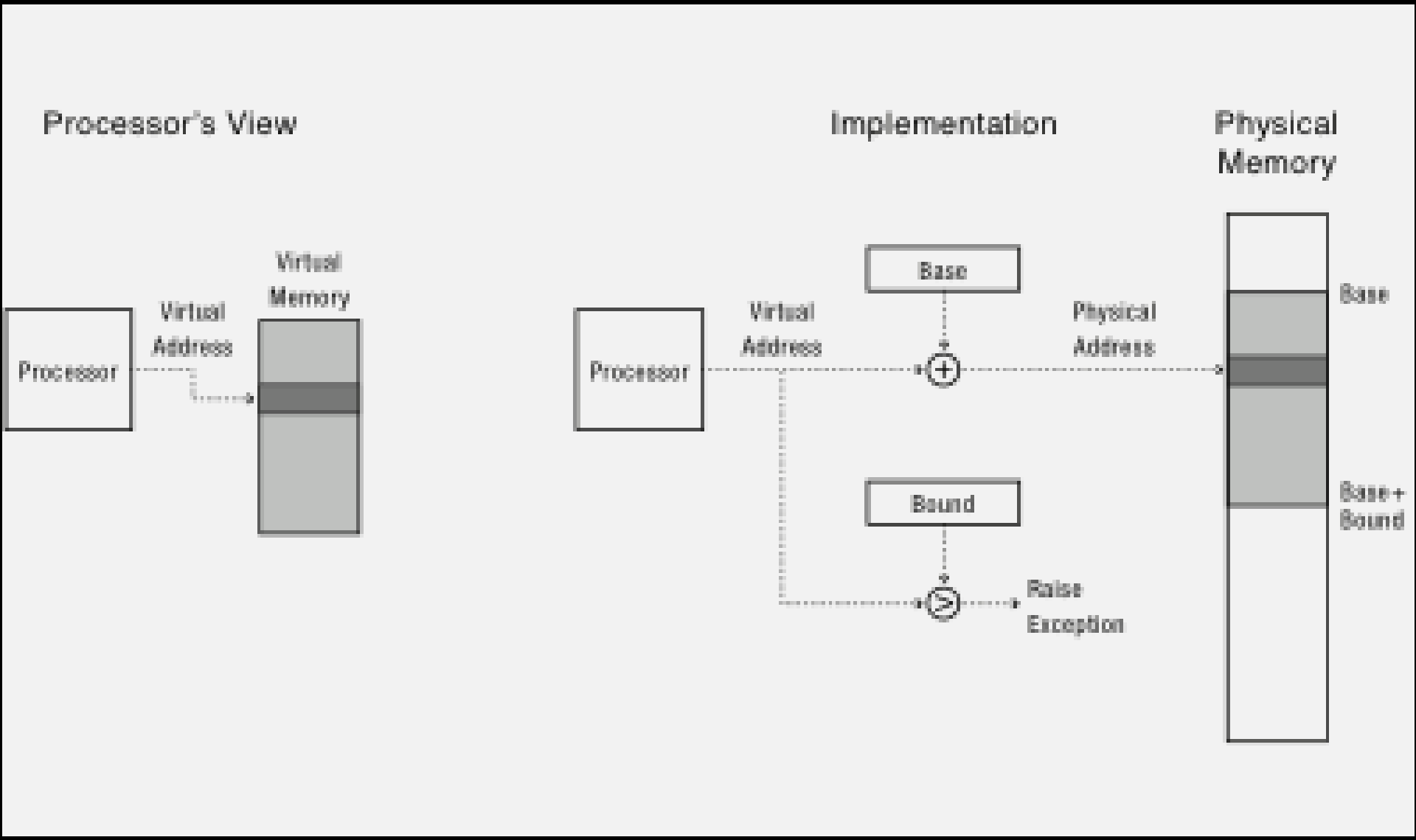
# Address Translation Goals

- 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

# Bonus Feature

- What can you do if you can (selectively) gain control whenever a program reads or writes a particular virtual memory location?
- Examples:
  - Copy on write
  - Fill on demand
  - Demand paging
  - Memory mapped files
  - ...

# Virtually Addressed Base and Bounds



# Activity #1

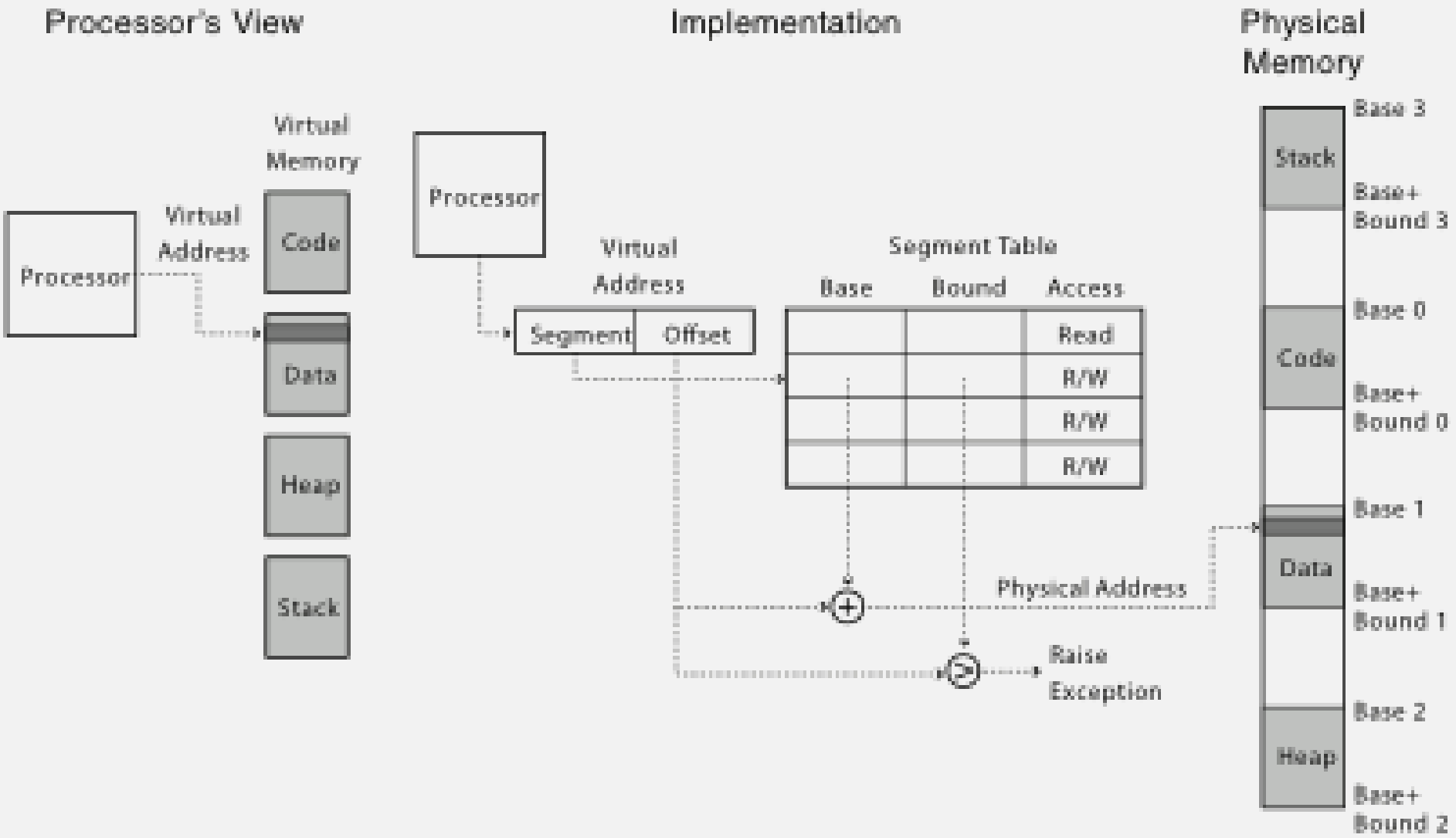
- Drawback of Base and Bounds

# Virtually Addressed Base and Bounds

# 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



2 bit segment #
12 bit offset

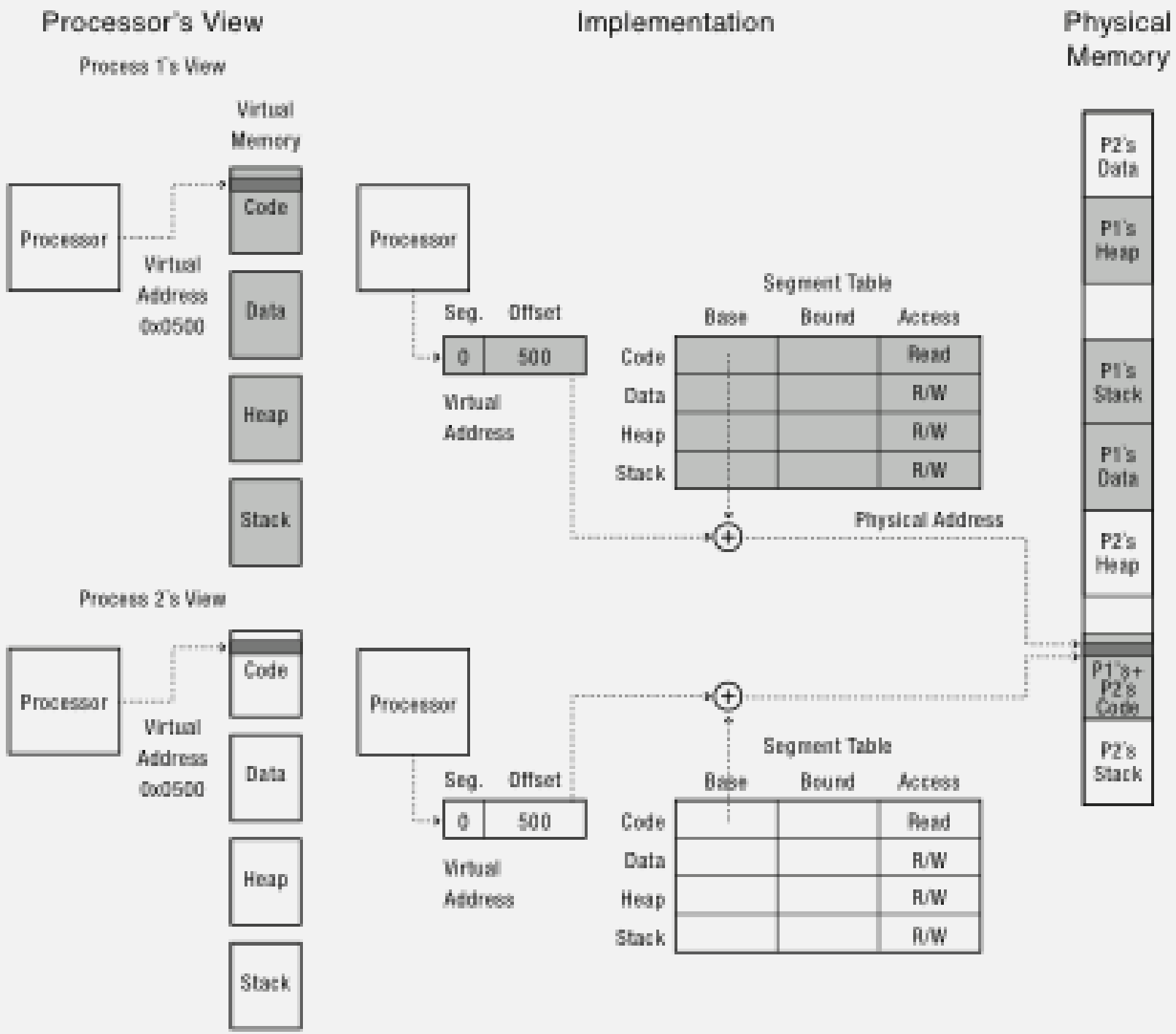
| Segment | start | length |
|---|---|---|
| **code** | 0x4000 | 0x700 |
| **data** | 0 | 0x500 |
| **heap** | – | – |
| **stack** | 0x2000 | 0x1000 |

Virtual Memory

| | |
|---|---|
| main: 240 | store #1108, r2 |
| 244 | store pc+8, r31 |
| 248 | jump 360 |
| 24c | |
| … | |
| strlen: 360 | loadbyte (r2), r3 |
| … | … |
| 420 | jump (r31) |
| … | |
| x: 1108 | a b c \0 |
| … | |

Physical Memory

| | |
|---|---|
| x: 108 | a b c \0 |
| … | |
| main: 4240 | store #1108, r2 |
| 4244 | store pc+8, r31 |
| 4248 | jump 360 |
| 424c | |
| … | … |
| strlen: 4360 | loadbyte (r2),r3 |
| … | |
| 4420 | jump (r31) |
| … | |


Processor's View
Implementation
Physical Memory
Process 1's View
Virtual Memory
Processor
Virtual Address 0x0500
Code
Data
Heap
Stack
Process 2's View
Processor
Virtual Address 0x0500
Code
Data
Heap
Stack
Processor
Seg.
Offset
0
500
Virtual Address
Segment Table
Base
Bound
Access
Code
Data
Heap
Stack
Read
R/W
R/W
R/W
Physical Address
Processor
Seg.
Offset
0
500
Virtual Address
Segment Table
Base
Bound
Access
Code
Data
Heap
Stack
Read
R/W
R/W
R/W
P2's Data
P1's Heap
P1's Stack
P1's Data
P2's Heap
P1's+ P2's Code
P2's Stack

# Activity #2

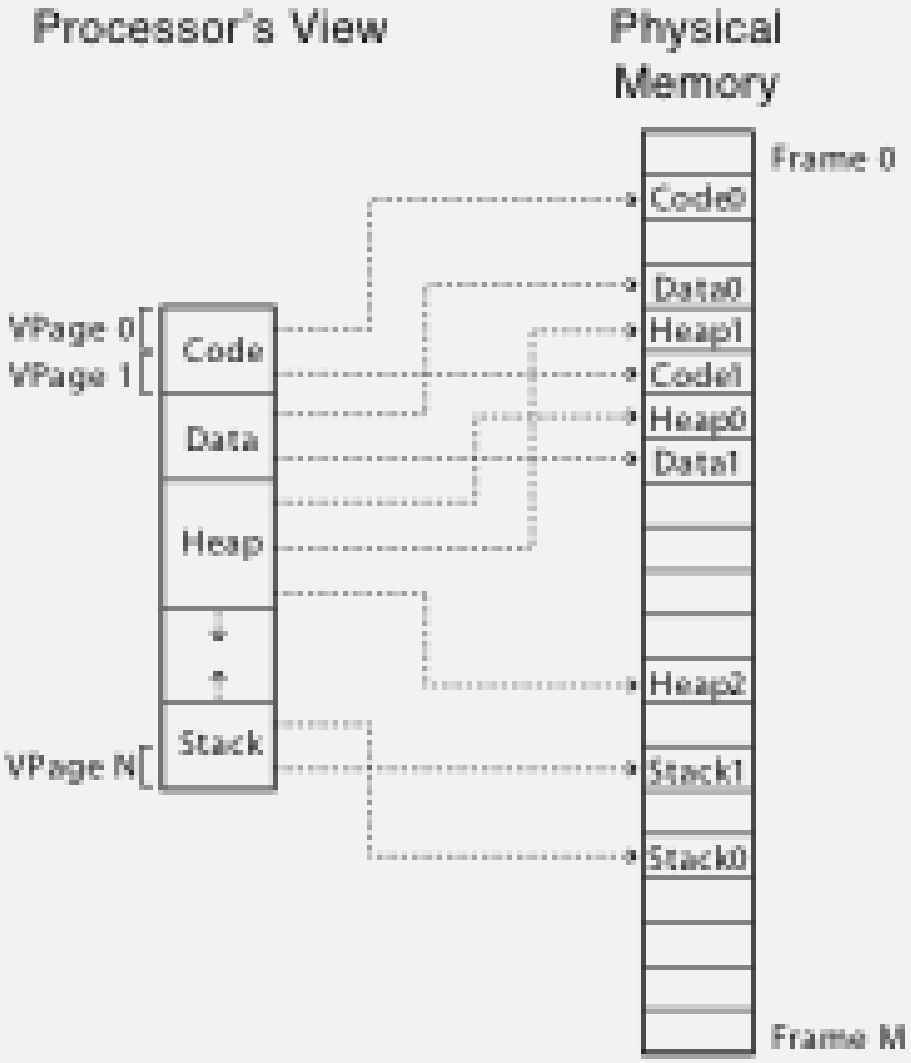
- Drawback of Segmentation

# Segmentation

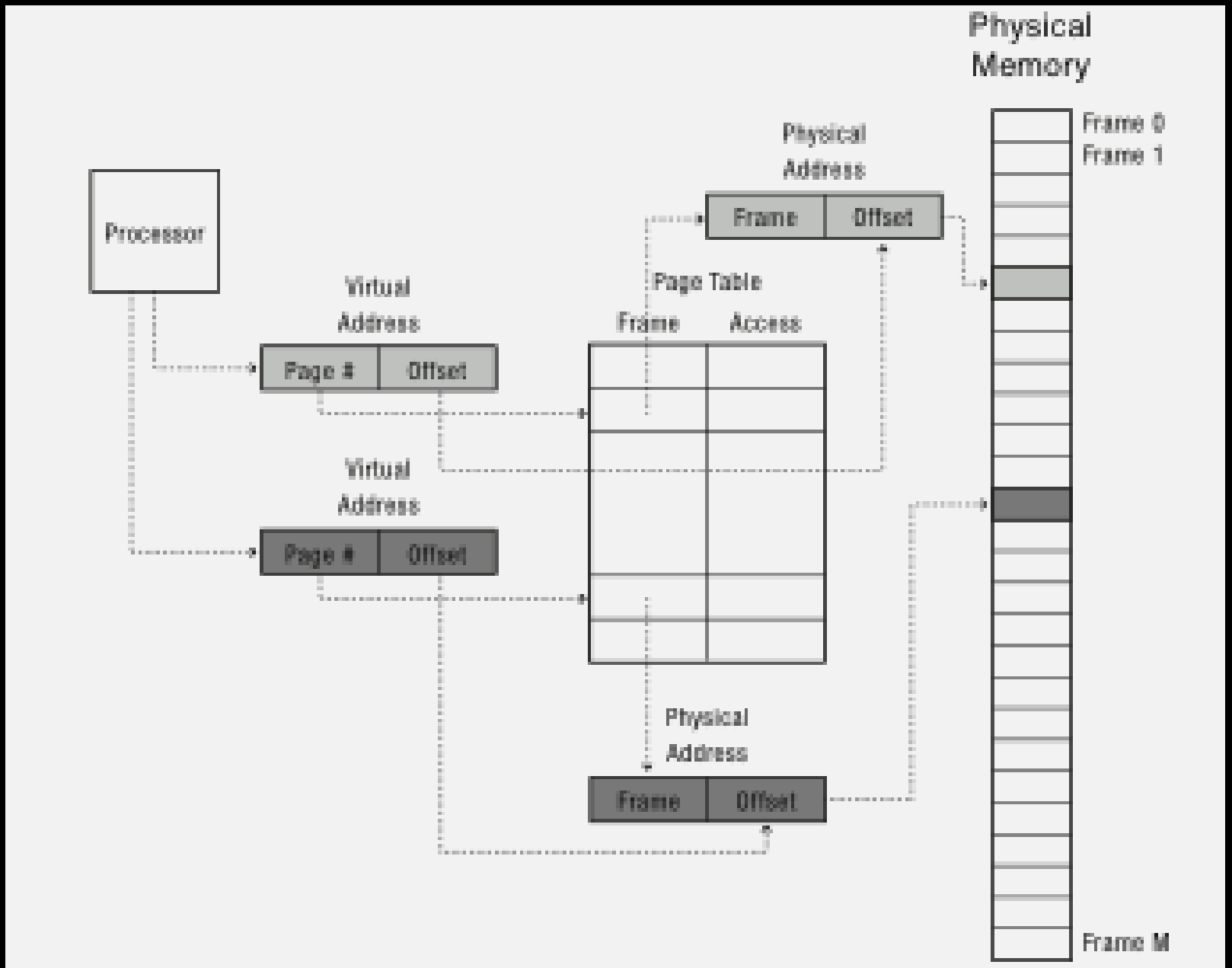
# Paged Translation

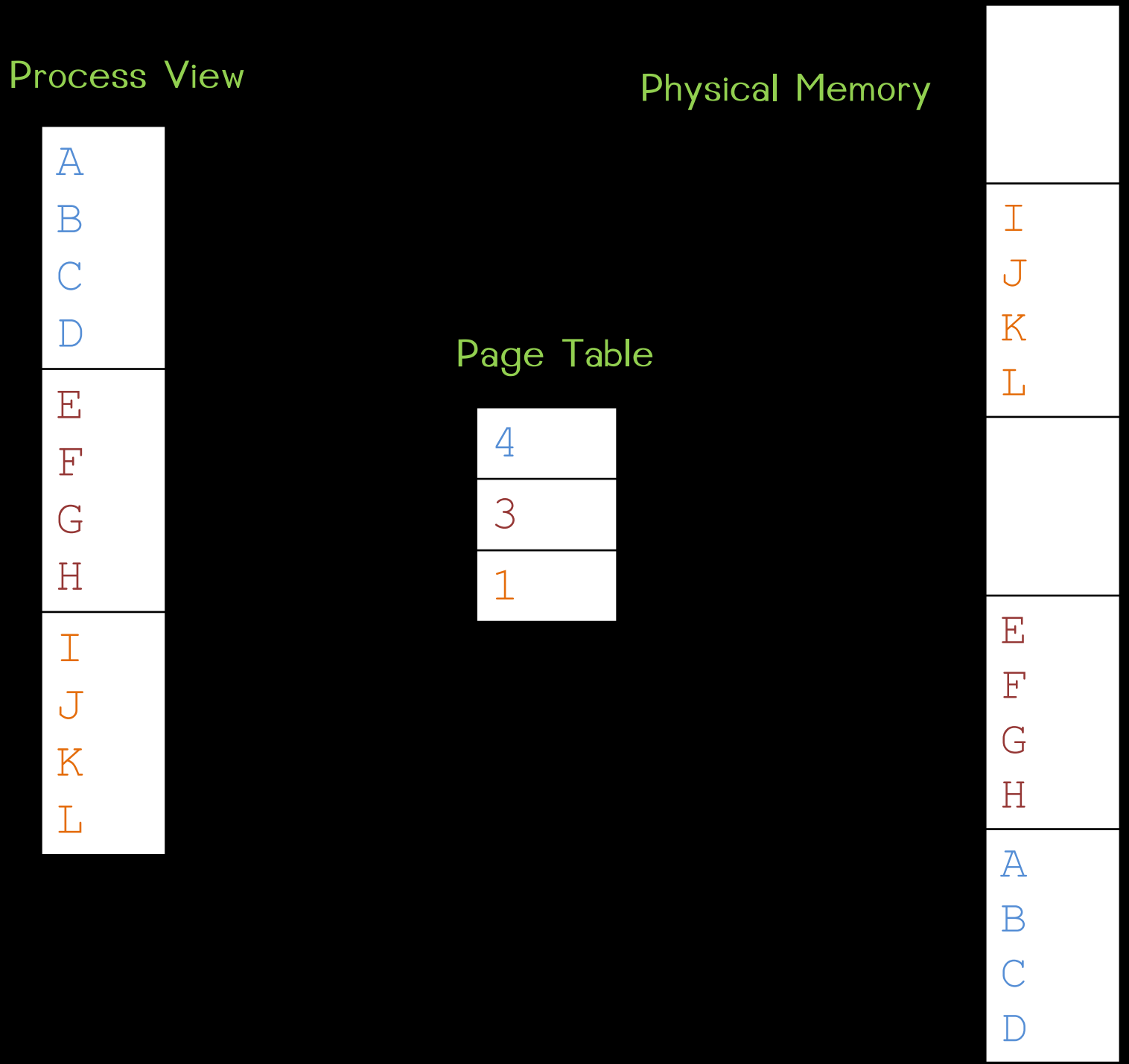
- Manage memory in fixed size units, or pages
- Finding a free page is easy
  - Bitmap allocation: 0011111100000001100
  - Each bit represents one physical page frame
- Each process has its own page table
  - Stored in physical memory
  - Hardware registers
    - pointer to page table start
    - page table length

# Paged Translation (Abstract)



# Paged Translation (Implementation)




Process View
A
B
C
D
E
F
G
H
I
J
K
L
Page Table
4
3
1
Physical Memory
I
J
K
L
E
F
G
H
A
B
C
D

# 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

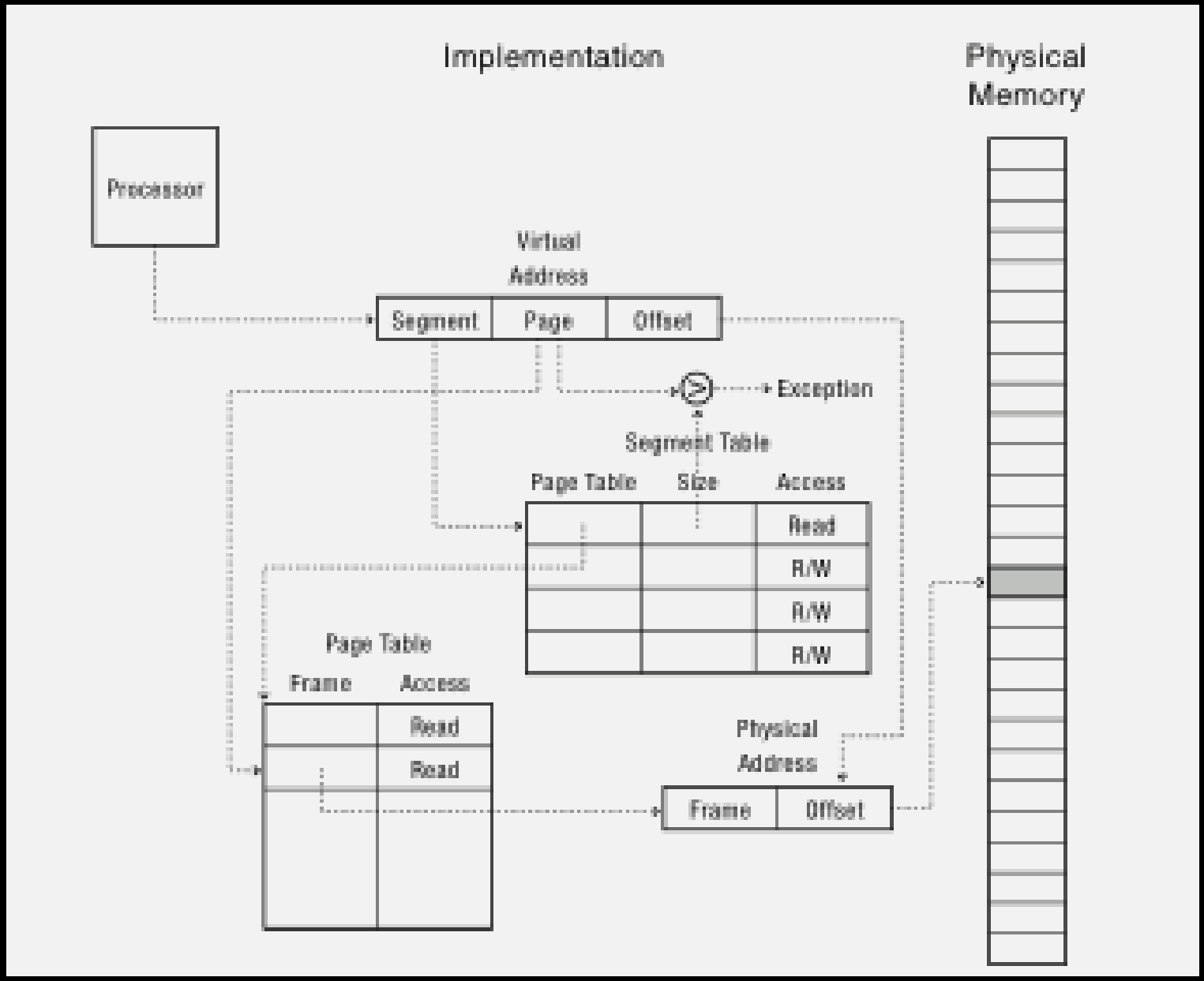
# Multi-level Translation

- 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

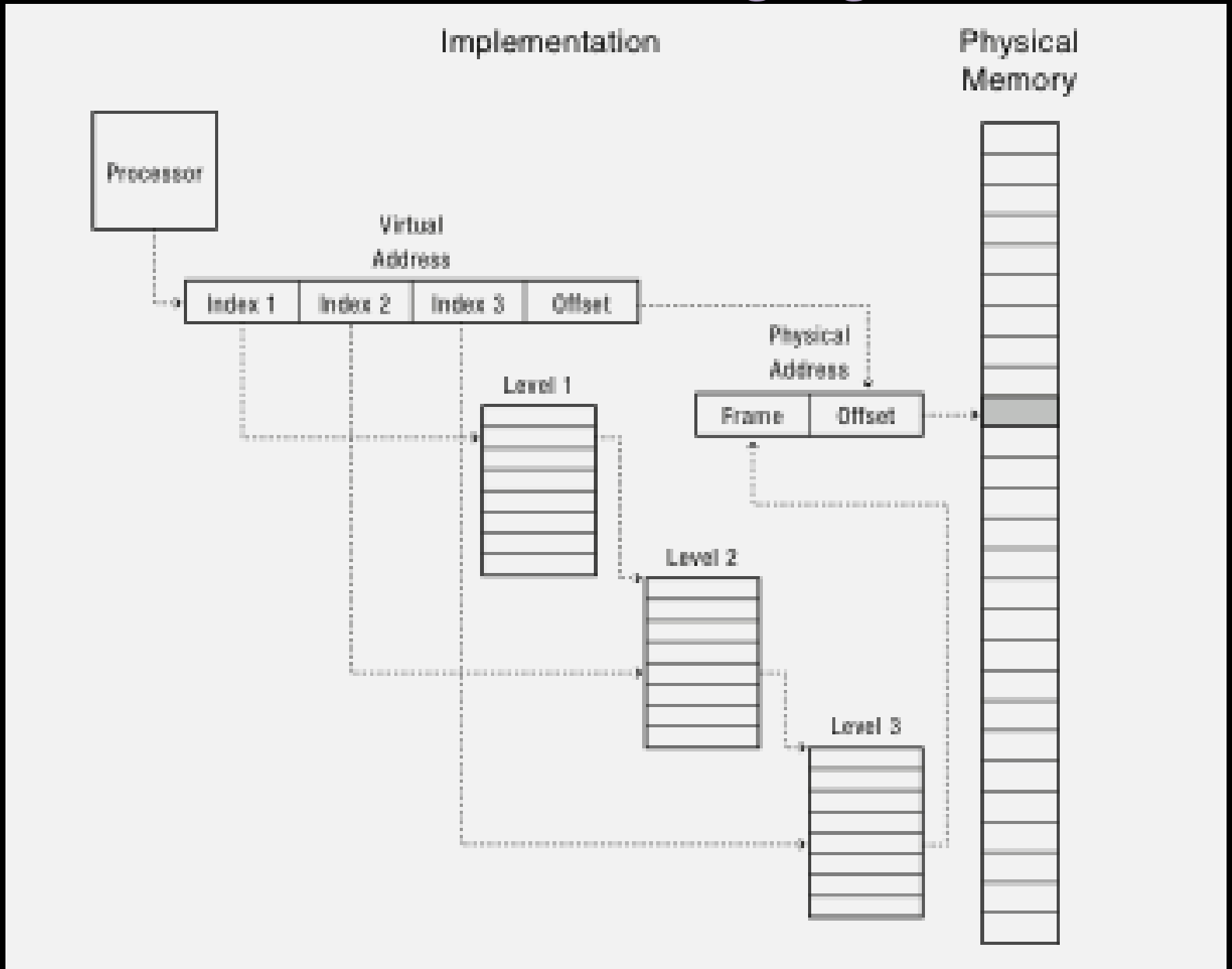
# Paged Segmentation

- 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



# Activity #4

- Drawback of Multilevel translation

# Multilevel Translation

# Activity #5

- Accelerate multilevel translation
  - Possible?
  - How?

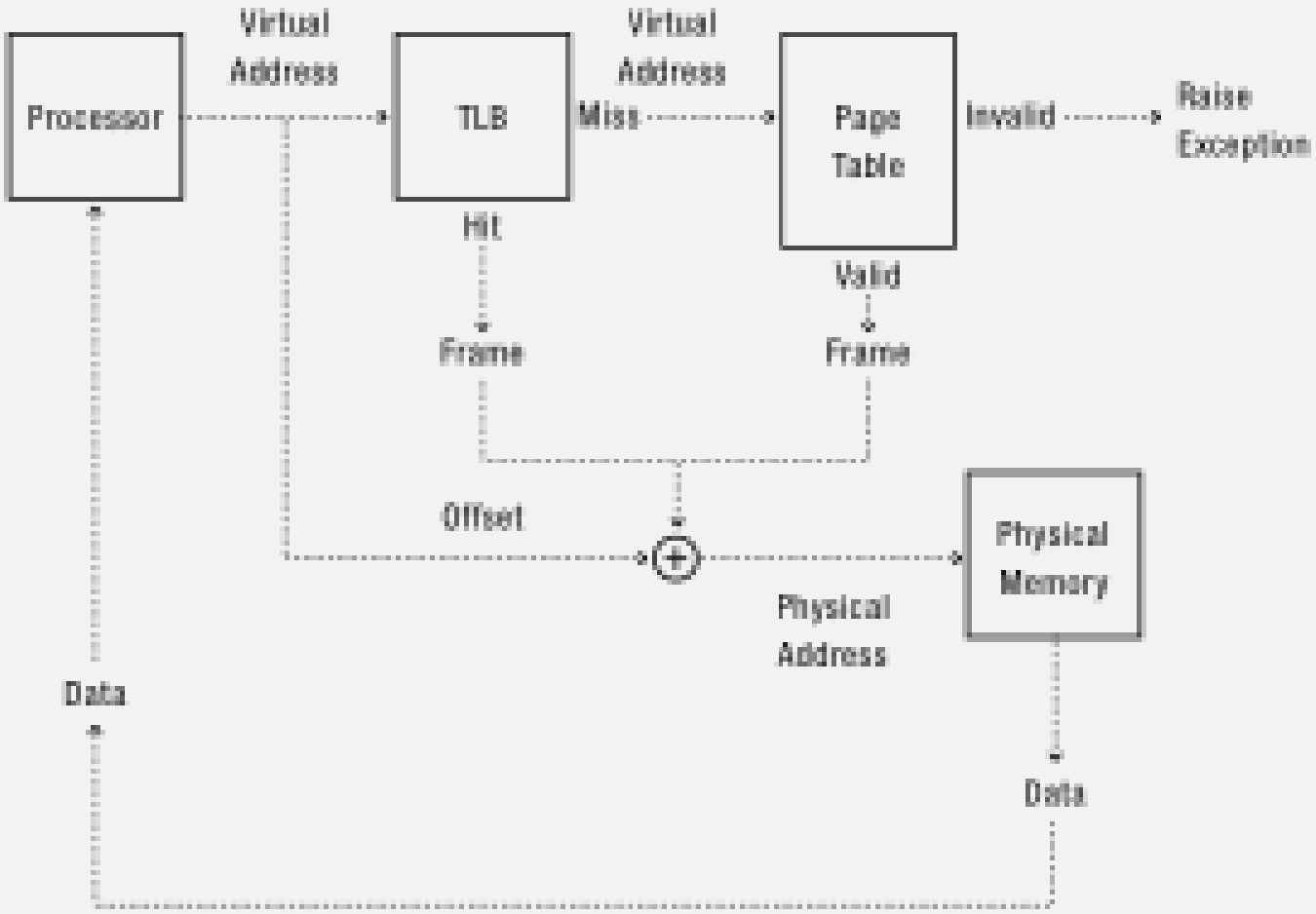
# Efficient Address Translation

- 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 =

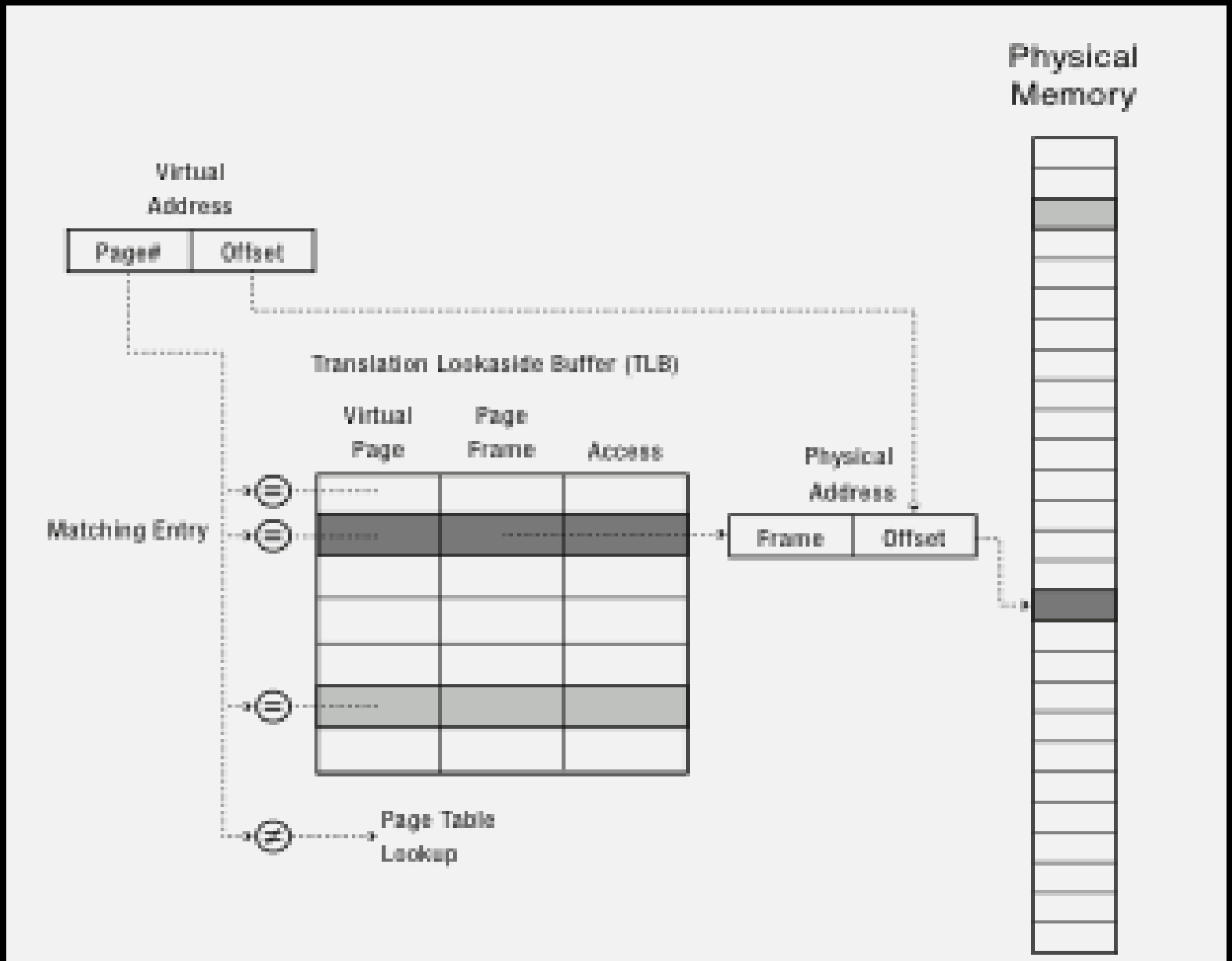
  Cost of TLB lookup +

  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
Matching Entry
Frame
Offset
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 (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