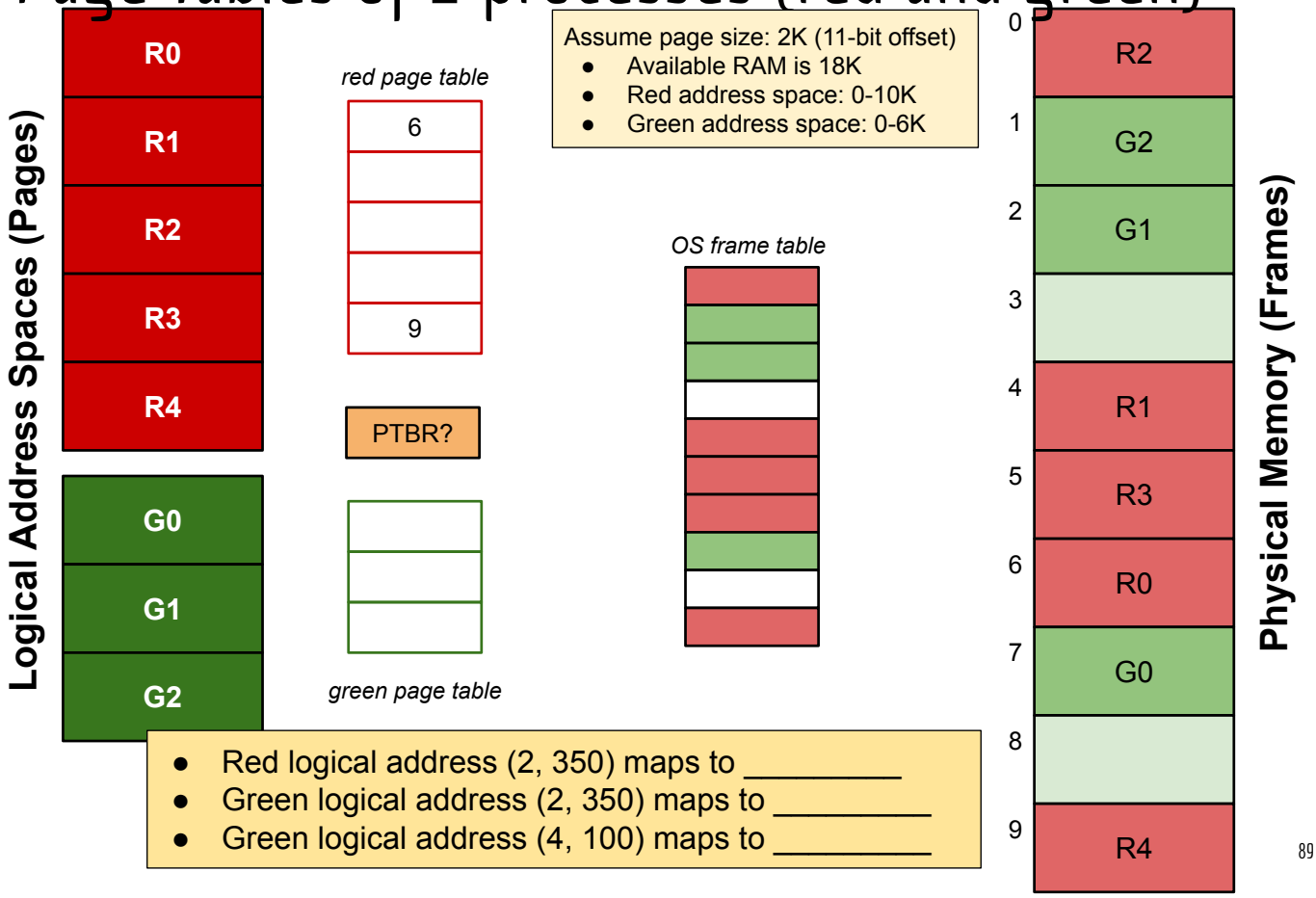
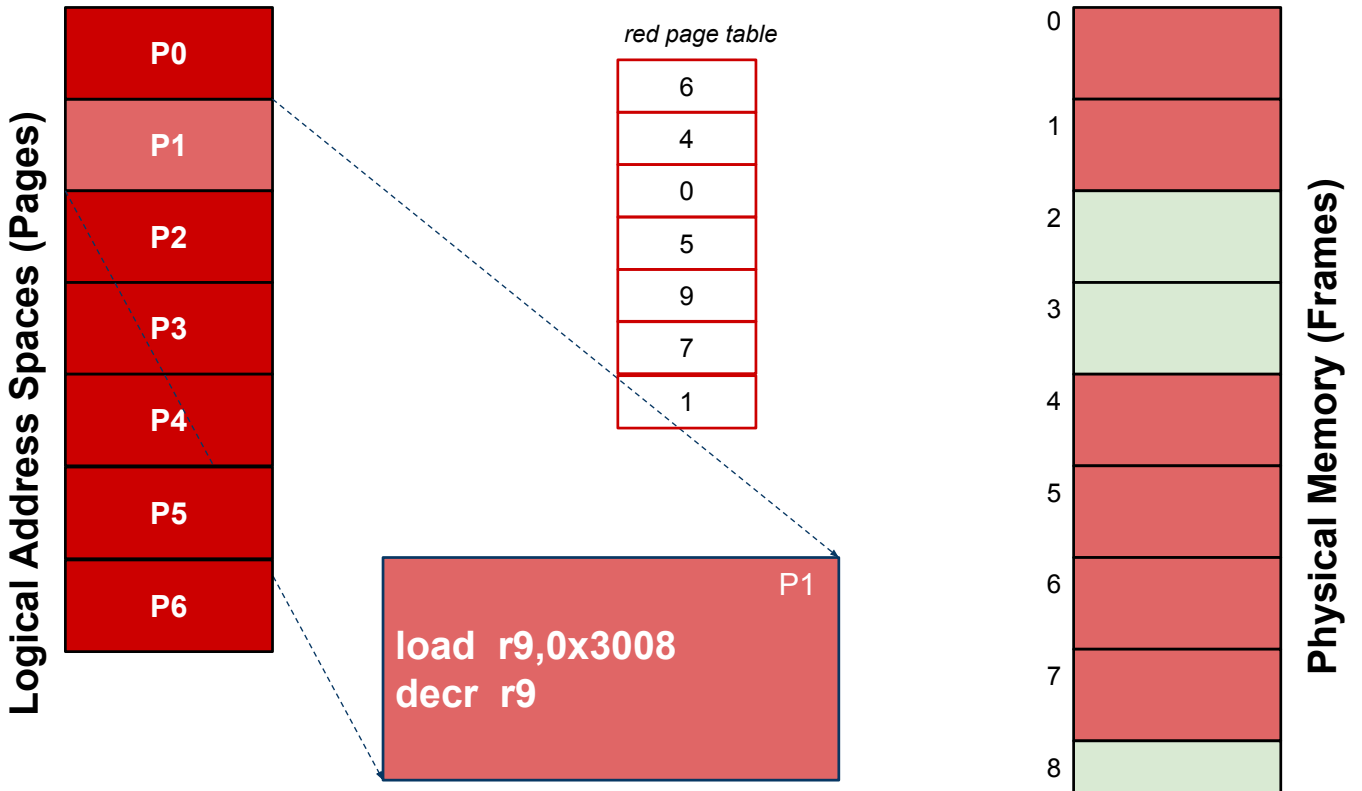
Page Tables of 2 processes (red and green)



- Red logical address (2, 350) maps to _____
- Green logical address (2, 350) maps to _____
- Green logical address (4, 100) maps to _____



Page size = 2K. Where is "0x3008"?
Page size = 4K. Where is “0x3008”?
Page size = 8K. Where is “0x3008”?

$2K = 2^{11}$
$4K = 2^{12}$
$8K = 2^{13}$

0x3008 (16-bit number 1 Hex => 4 bits)

- Page size 2K = 2^{11}
- lowest 11 bits are page offset
- higher 5 bits are page number

0x3008

0011 0000 0000 1000 **Page 6 Offset 008**

0x3008 (16-bit number 1 Hex => 4 bits)

- Page size 4K = 2^{12}
- lowest 12 bits are page offset
- higher 4 bits are page number

0x3008

0011 0000 0000 1000 **Page 3 Offset 008**

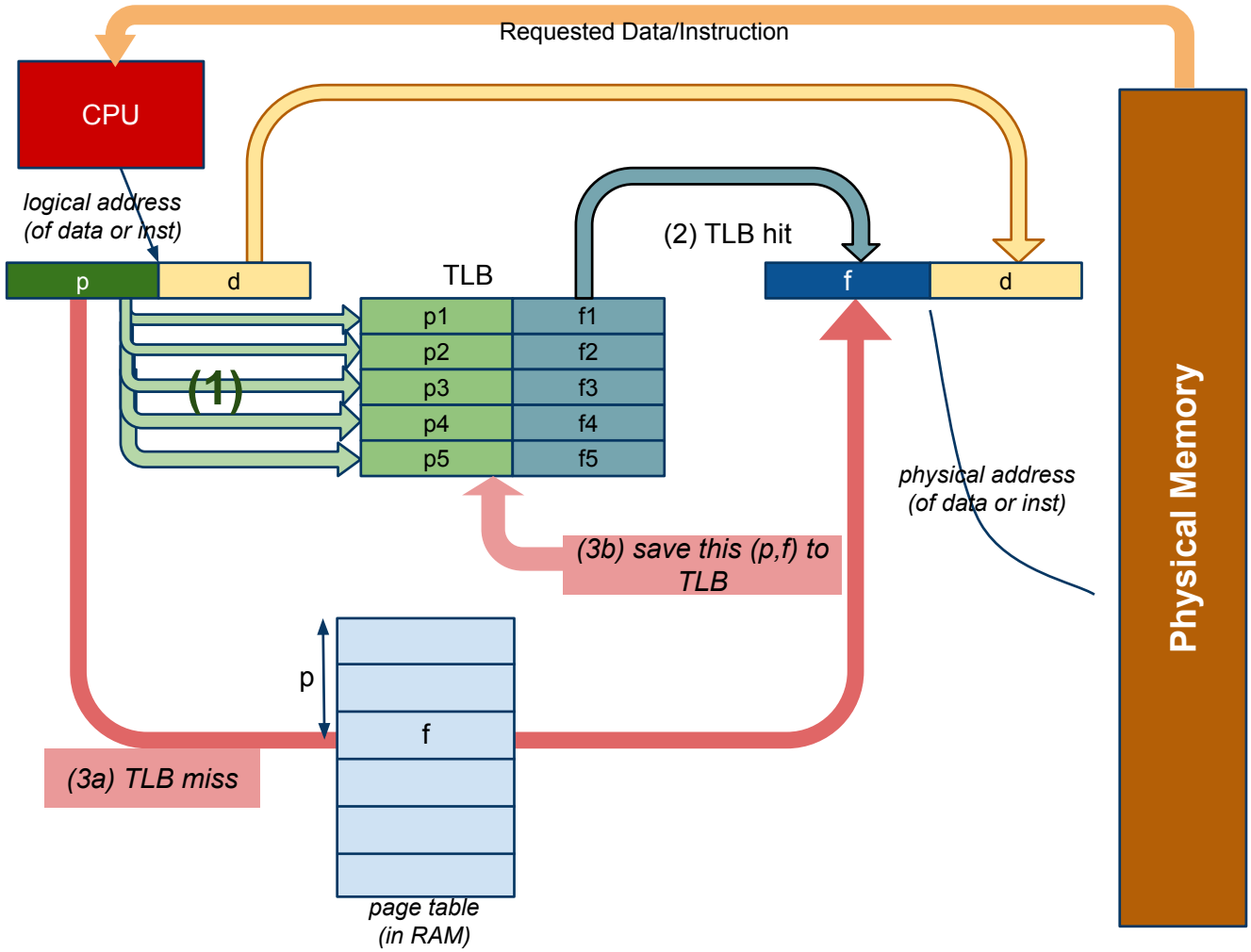
Paging hardware overhead:

access memory 2x

(*2x slower*)

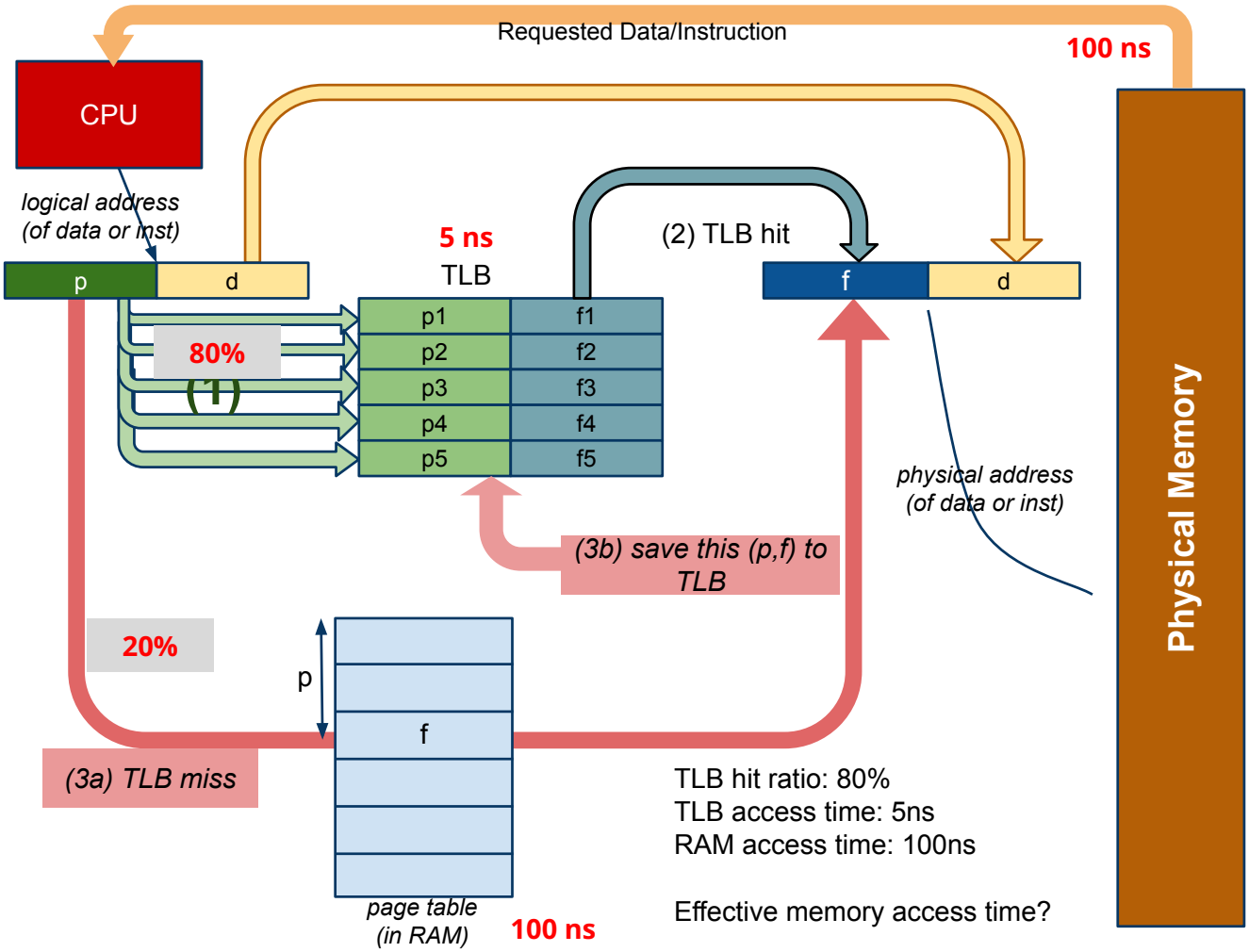
Translation Look-Aside Buffer (TLB)

- Mapping logical page numbers to physical frame address requires two RAM references
  - The first access to fetch the page table entry (for mapping logical page number to physical frame number)
  - The second access to fetch the target data/instruction
- To reduce overhead, use a hash map (implemented in hardware) to resolve mapping from page numbers to frame address
  - Associate Memory (searched by page numbers)
- TLB holds (pid, page number, frame addr) tuples of recently used addresses
- Issues: TLB hit and TLB miss



Effective Memory Access Time

- RAM access time is around hundreds of nanoseconds
- TLB access time is usually as fast as CPU registers
- Without TLB: Always two RAM references
- With TLB
  - TLB hit: One TLB access (read) + One RAM Access
  - TLB miss: Two RAM accesses + One TLB (write/update)
- On modern CPUs, TLB can hold up to 4096 entries
- Statistical/Probabilistic Formula (explained)



Sample Effective Memory Access Time

TLB hit ratio: 80%
TLB access time: 5ns
RAM access time: 100ns

What is the effective memory access time?

- 80% TLB hit: 1 TLB access + 1 RAM access = 0.8 (5 + 100) = 0.8 x 105 = 84
- 20% TLB miss: 1 TLB access + 2 RAM access = 0.2 (5 + 200) = 0.2 x 205 = 41

Analyzed differently:

- TLB is always accessed: 5ns
- 80% TLB hit: 1 RAM access: 0.8 x 100 = 80ns
- 20% TLB miss: 2 RAM accesses: 0.2 x 200 = 40ns

The total from either analysis: 125ns
Without TLB the access time is 200ns

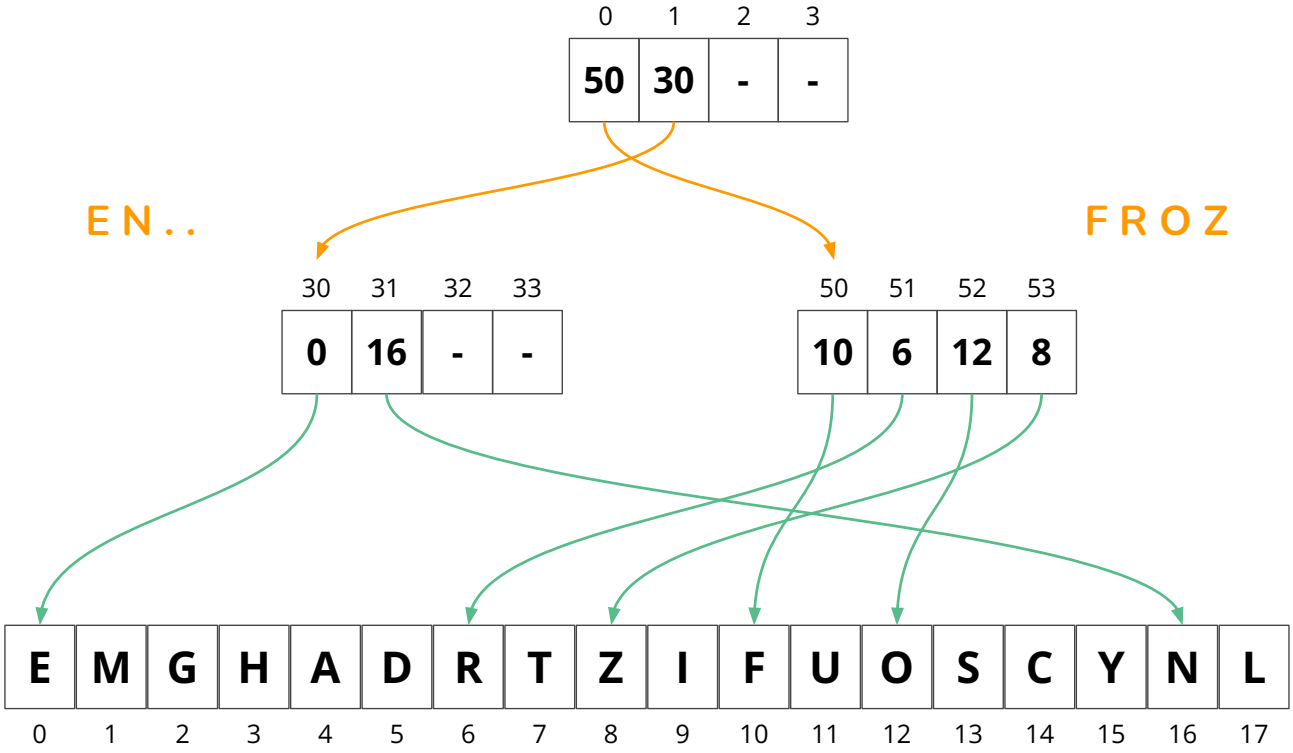
Managing (Huge) Page Tables

Page Table Management

- One page table (PT) per process
- The number of entries in PT is proportional to the process size
- The maximum number of entries in PT is determined by
  - The maximum size logical address space allowed by the architecture
  - The page size selected by the hardware
- Page tables must be stored in RAM (similar to memory for process)
- For MMU address translation hardware to work properly, page tables must be stored **contiguously in RAM**
  - Process can be stored non-contiguously but page tables cannot?
- It becomes an issue when the page table is **bigger** than the size of a **single page**

Issue: Giant & Contiguous Page Tables

Word Puzzle (with contiguous capacity 4)



Page Tables (like segment tables) are a *contiguous* "array" that may span multiple pages

Size of Page Table

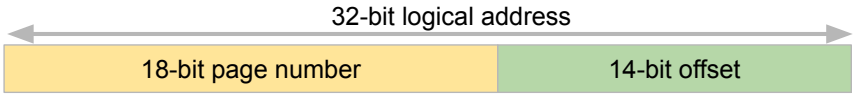
Assumptions:
- 20-bit logical address
- 4 bytes per page table entry

Page Size	Number of bits in		Max Number of pages	Max Size of Page Table	Explanation
	Offset	Page Number			
8K = 2^{13}	13	7	2^7	$2^2 \times 2^7 = 2^9 = 512$	OK: Page tables fit one page
4K	12	8	2^8	$2^2 \times 2^8 = 2^{10} = 1K$	OK: PT fit one page
2K	11	9	2^9	$2^2 \times 2^9 = 2^{11} = 2K$	OK: PT fit one page
1K	10	10	2^{10}	$2^2 \times 2^{10} = 2^{12} = 4K$	**Page tables require 4 frames**
512	9	11	2^{11}	8K	**PT requires 16 frames**
256	8	12	2^{12}	16K	**PT requires 64 frames**

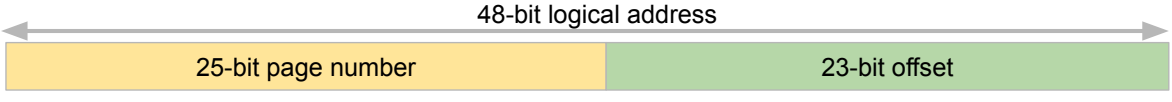
Logical Address Structure

Logical address space 1GB (2^{30} bytes) with page (or frame) size 8KB (2^{13} bytes)

Logical address space 4GB (2^{32} bytes) with page (or frame) size 16KB (2^{14} bytes)



Logical address space 256TB (2^{48} bytes) with page (or frame) size 8MB (2^{23} bytes)



Page Table Structure

- Main use: maps logical page numbers to physical frame “address”
- A page table is treated like a **contiguous** array
- Its base address is stored in PTBR (Page Table Base Register)
- Each row in the page table holds the following information
  - Frame number where the page is mapped to
  - Attributes/flags: reference bit, access mode: R/W, R/O, shared?
- Choices of implementation
  - Small “page tables” can be stored on special CPU registers
  - Large page tables must be stored **contiguously** in the RAM. The size of a page table should fit a **single** frame!
  - Problem: what if the page table itself is too big to fit single frame?

Advantages of Paging

- Improvements over variable-size partitions
  - No compaction needed
  - No external fragmentation
  - A small fraction of internal fragmentation
  - Processes can be loaded into non-contiguous frames
- Swapping can be performed per-page (not the entire process)

Page Sharing and Protection

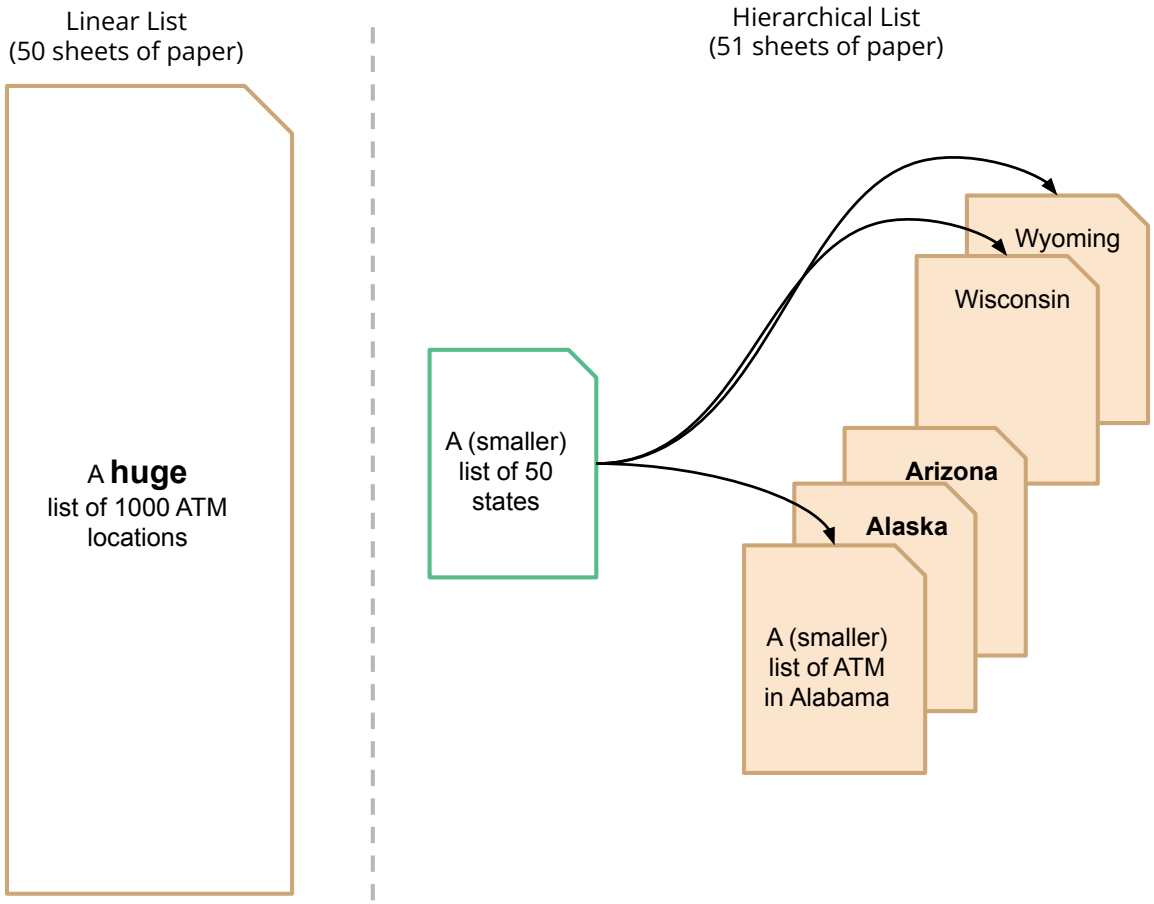
- Page Table Entries can store the following information
  - Protection/Access: R/W, R/O
  - Present/No-Present Bit: if a page is part of your process address space
- Implementation of “Shared Memory” on a paging-based CPU is straight forward
  - Assume two processes P1 and P2 share a physical frame #S
  - P1 maps the shared memory segment to its logical page #X
  - P2 maps the shared memory segment to its logical page #Y
  - The OS inserts
    - X => S for P1’s PTE
    - Y => S for P2’s PTE

Page Table Design Constraints

- The memory hardware views a page table as a ***contiguous*** 1D array
- Page tables must fit into **a single physical frame** in RAM
- The size of page tables is determined by the **size of the address space** supported by the CPU architecture (and NOT by the amount of RAM installed)
- When page tables exceed the size of a physical frame, the page tables themselves must be designed to be **pageable** (can be stored non-contiguously into several physical frames)

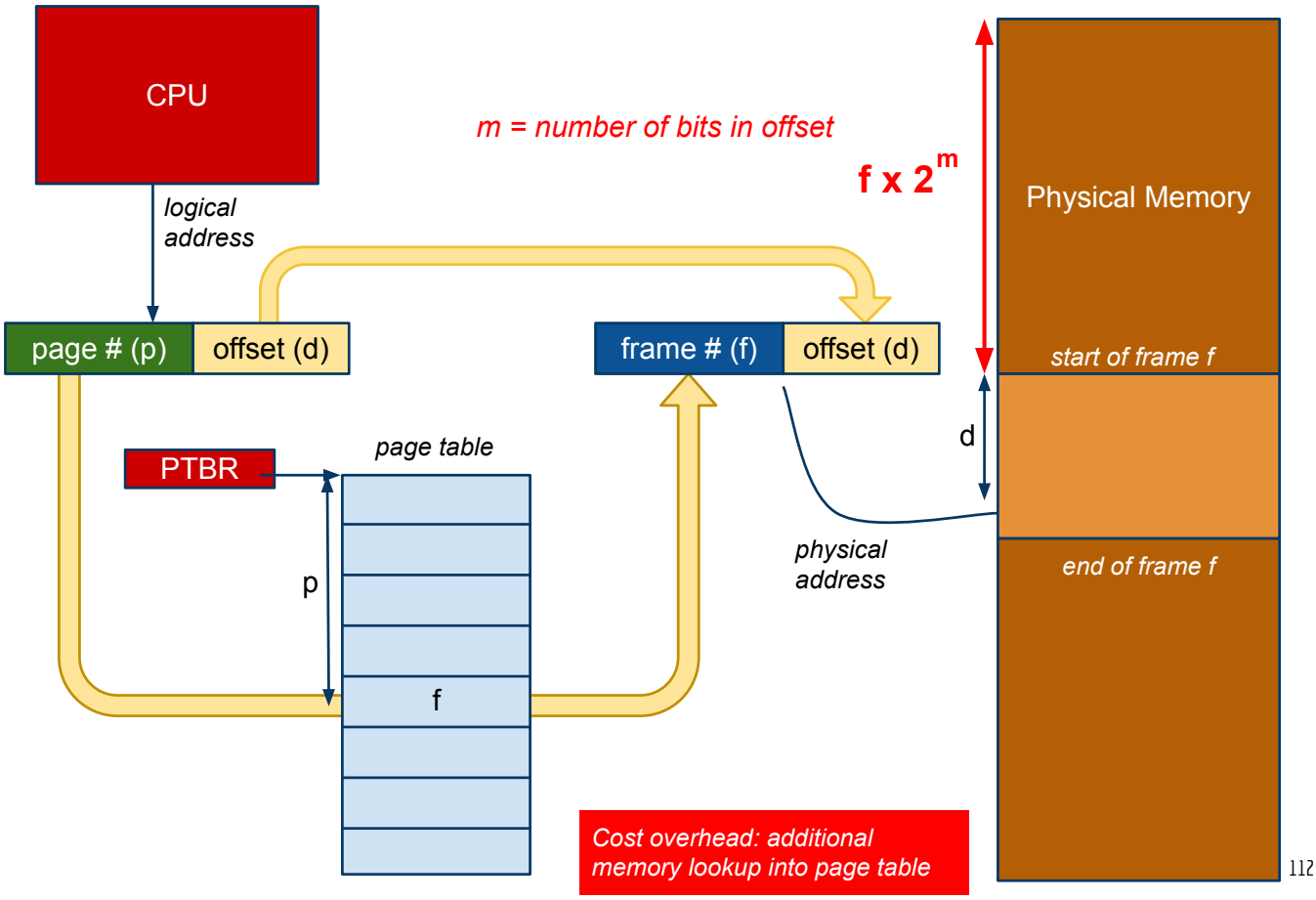
Managing Huge Page Tables

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables



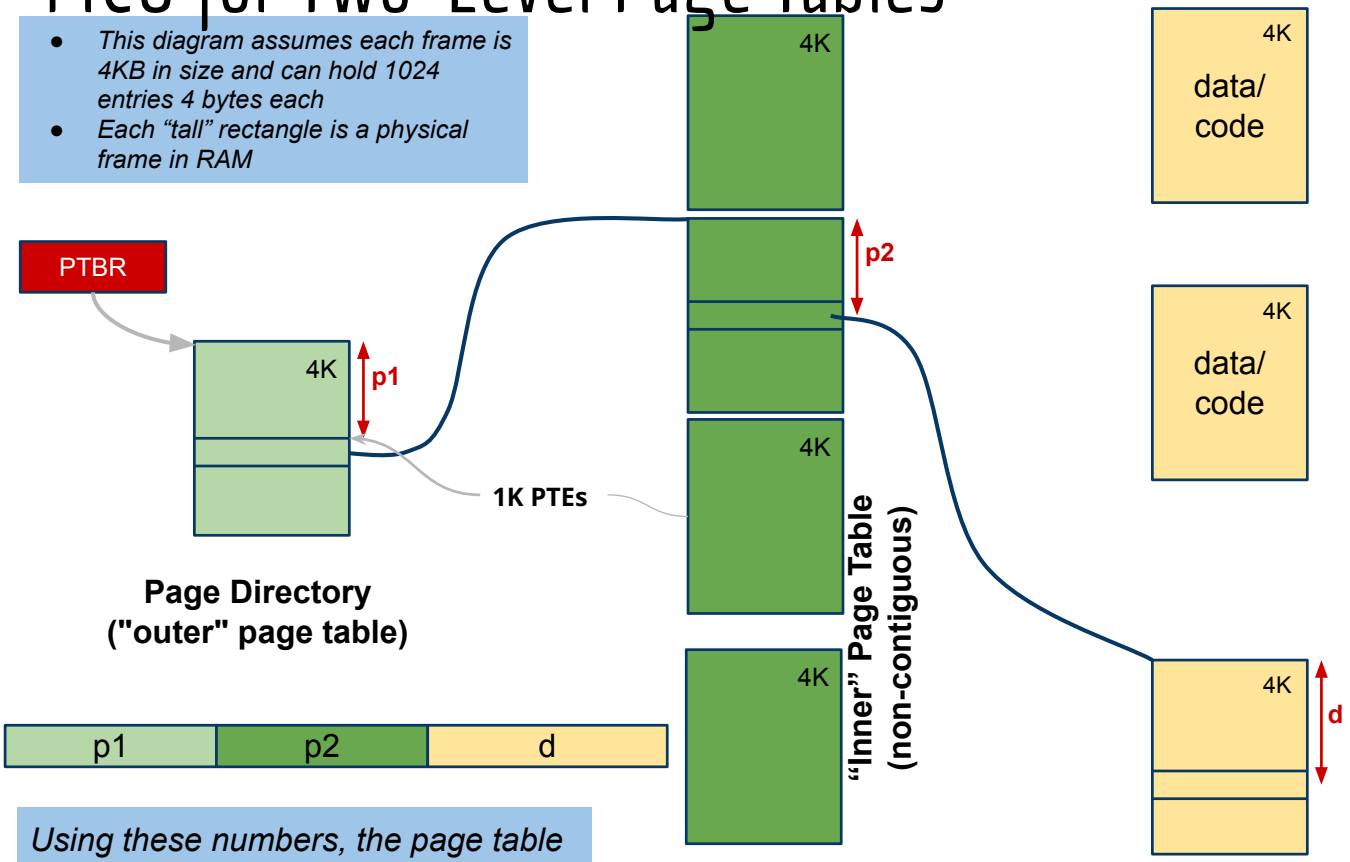
Page Table Size Considerations

- How many bytes required per page table entry (PTE)
  - What information stored in each PTE?
- How many entries per page table?
- How many PTEs can fit into a physical frame?



MCU for Two-Level Page Tables

- *This diagram assumes each frame is 4KB in size and can hold 1024 entries 4 bytes each*
- *Each "tall" rectangle is a physical frame in RAM*



Using these numbers, the page table can support an address space of size 1024x1024x4K = 4GB

"Outer" and "inner" page have similar structure

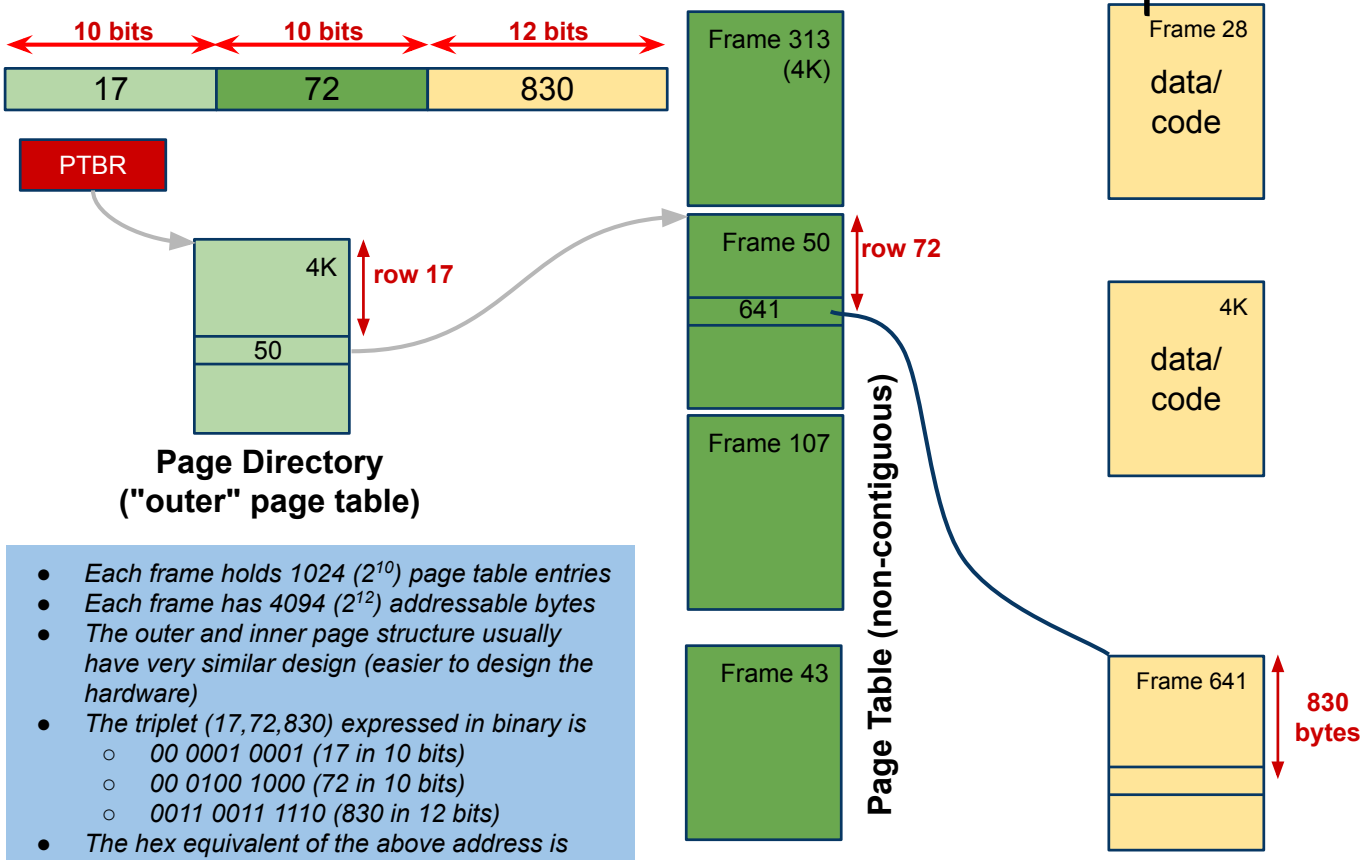
Paging hardware overhead for 2-level page tables

=

access memory 3x

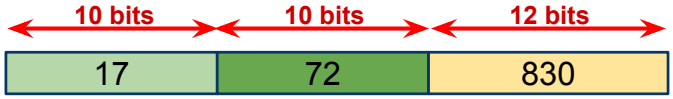
⅓ x slower

2-Level PTs: address translation example



- *Each frame holds 1024 (2^{10}) page table entries*
- *Each frame has 4094 (2^{12}) addressable bytes*
- *The outer and inner page structure usually have very similar design (easier to design the hardware)*
- *The triplet (17,72,830) expressed in binary is*
  - *00 0001 0001 (17 in 10 bits)*
  - *00 0100 1000 (72 in 10 bits)*
  - *0011 0011 1110 (830 in 12 bits)*
- *The hex equivalent of the above address is 0x0444 833E*

Address Format Details



Outer PT index 17 (0x11) ⇒ 00 0001 0001
Inner PT index 72 (0x48) ⇒ 00 0100 1000
Offset 830 (0x33E) ⇒ 0011 0011 1110

00 0001 0001 00 0100 1000 0011 0011 1110

0000 0100 0100 0100 1000 0011 0011 1110

0 4 4 4 8 3 3 E

The effect of frame size to logical address space size

Assume two-level paging

Assume 4-byte PTEs

Frame Size	PTEs per frame	Logical Address Structure	Theoretical Max Logical Address Space
2KB (2^{11} bytes)	2^9 entries	9-bit outer page, 9-bit inner page, 11-bit offset	2^{29} bytes (512 MB)
8KB (2^{13} bytes)	2^{11} entries	11-bit outer page, 11-bit inner page, 13-bit offset	2^{35} bytes (32 GB)
2MB (2^{21} bytes)	2^{19} entries	19-bit outer page, 19-bit inner page, 21-bit offset	2^{69} bytes (512 EB)

Assume 8-byte PTEs

Frame Size	PTEs per frame	Logical Address Structure	Theoretical Max Logical Address Space Size
8KB (2^{13} bytes)	2^{10} entries	10-bit outer page, 10-bit inner page, 13-bit offset	2^{33} bytes (8 GB)
32KB (2^{15} bytes)	2^{12} entries	12-bit outer page, 12-bit inner page, 15-bit offset	2^{39} bytes (512 GB)
2MB (2^{21} bytes)	2^{18} entries	18-bit outer page, 18-bit inner page, 21-bit offset	2^{47} bytes (128 TB)

Hierarchical Page Tables

- Modern CPUs support larger logical (and physical) address space
  - Giga (2^{30}) => Tera (2^{40}) => Peta (2^{50}) => Exa (2^{60})
  - 48-bit address space on Intel/AMD CPUs
- Larger address spaces require more bits stored in the PTE
  - Page Table Entries must grow from 4 bytes to 8 bytes
- Increasing PTE size implies **fewer entries** can be stored in a physical frame
- Fewer PTE entries implies fewer bits per page number field
- Fewer bits per page number field implies more fields (deeper hierarchy) to cover

Intel/AMD Examples

- 64-bit architecture
- Supported page sizes: 4KB (2^{12}), 2MB (2^{21}), 4MB (2^{22}), 1GB (2^{31})
- 52-bit physical address (2^{52} bytes = 4 Petabytes)
- Logical address spaces
  - 32-bit logical address (2^{32} bytes = 4 Gigabytes)
  - 48-bit logical address (2^{48} bytes = 256 Terabytes)
- Hierarchical Paging: 2-, 3-, and 4-level paging

Intel/AMD Examples

32-bit address, 4K pages, 4-byte page table entries (2-level paging)

10-bit PageDir	10-bit PageTab	12-bit Frame Offset

32-bit address, 4M pages, 4-byte page table entries

10-bit PageDir	22-bit Frame Offset

32-bit address, 4K pages, 8-byte page table entries (3-level paging)

2	9-bit offset	9-bit offset	12-bit Frame Offset

The leftmost 2-bit selects one of the four page directory pointer entries

32-bit address, 2M pages, 8-byte page table entries (2-level paging)

2	9-bit offset	21-bit Frame Offset

Notice the uniform number of bits (9 or 10) are used for "page number" in each format

48-bit address, 4K pages, 8-byte page table entries (4-level paging)

9-bit offset	9-bit offset	9-bit offset	9-bit offset	12-bit Frame Offset

48-bit address, 2M pages, 8-byte page table entries (3-level paging)

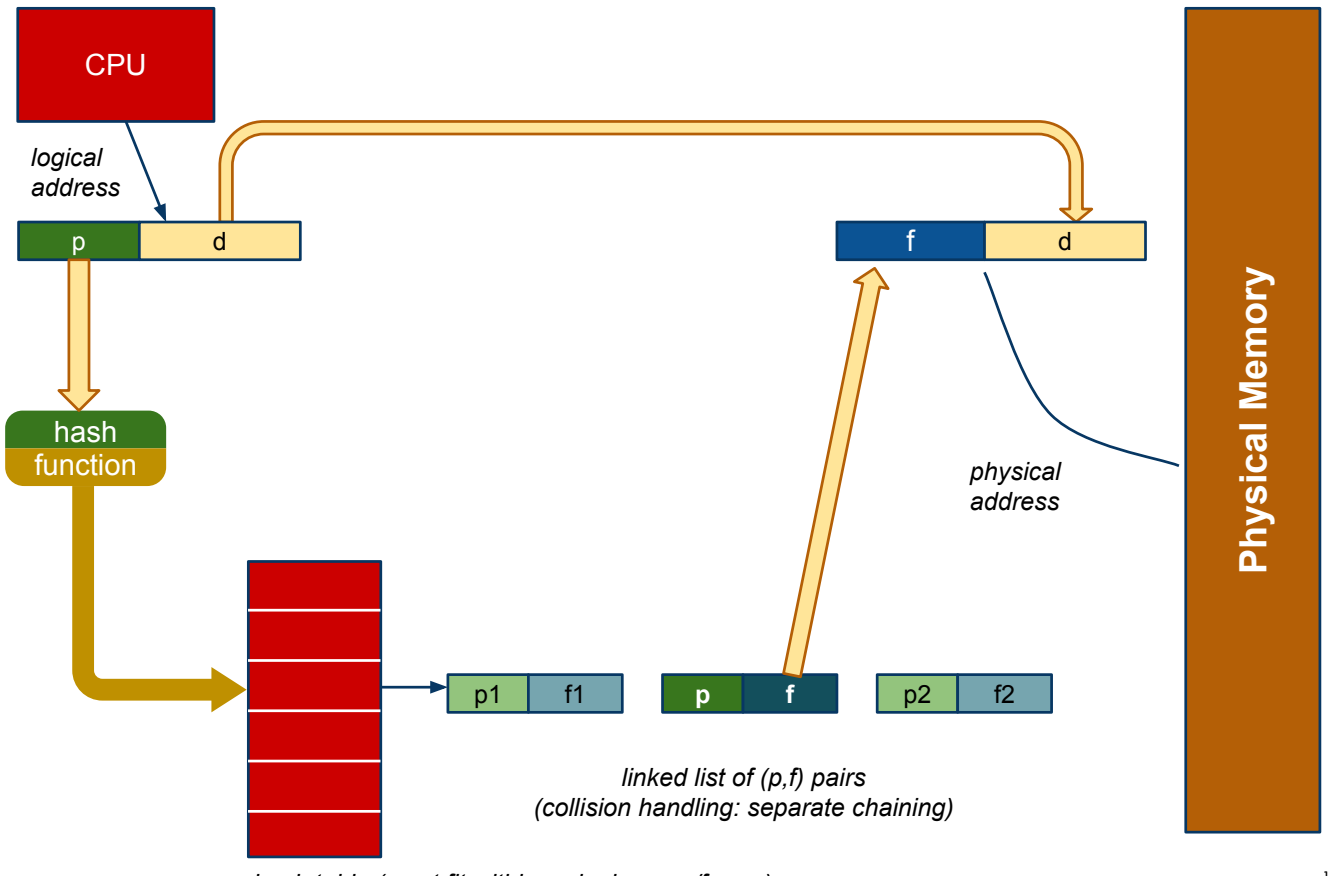
9-bit offset	9-bit offset	9-bit offset	21-bit Frame Offset

48-bit address, 1G pages, 8-byte page table entries (2-level paging)

9-bit offset	9-bit offset	30-bit Frame Offset

Hashed Page Tables

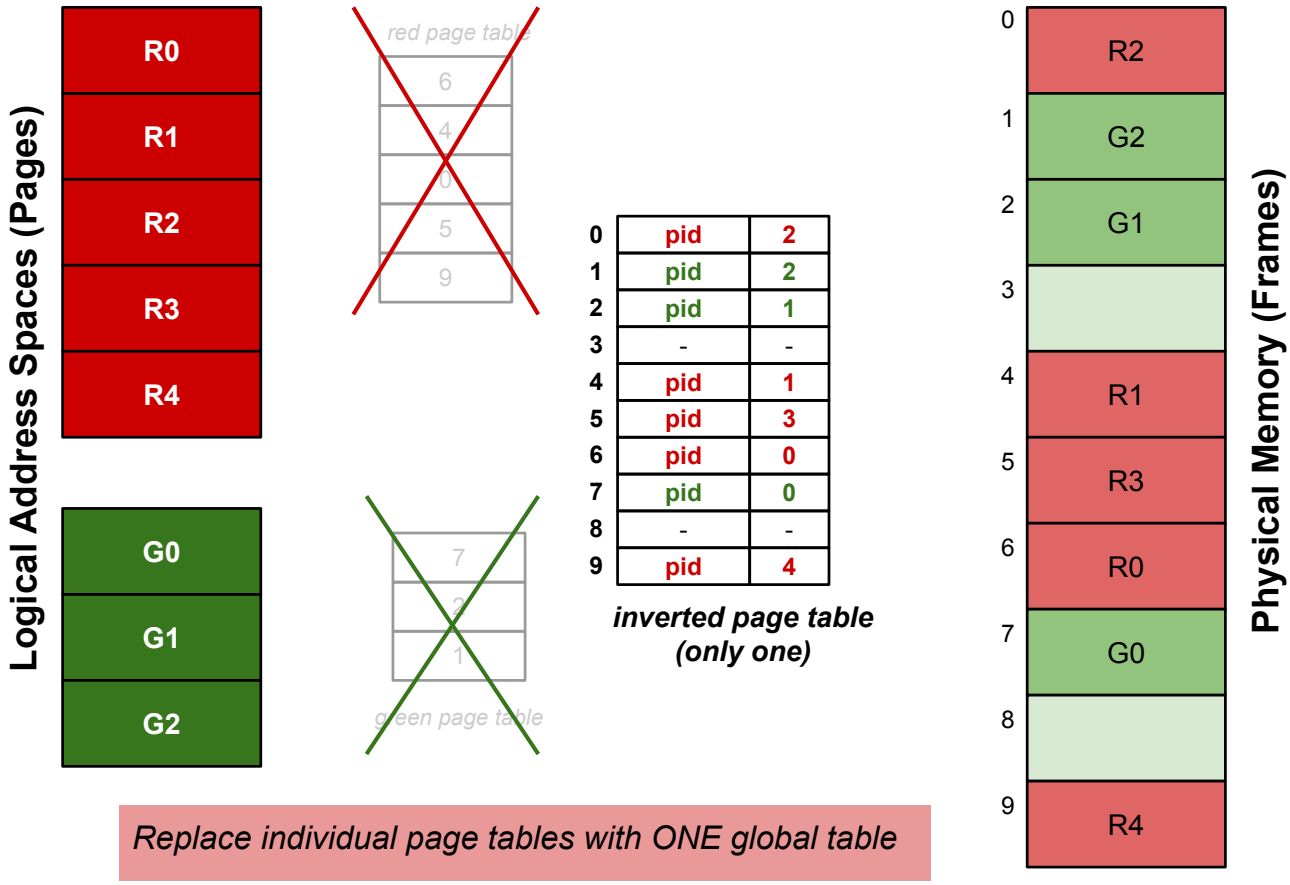
- Observations
  - Each level of hierarchical paging adds one more memory access
    - 1-level ⇒ 2x memory references (50% effective memory bandwidth)
    - 2-level ⇒ 3x memory references (33% effective memory bandwidth)
    - 3-Level ⇒ 4x memory references (25% effective memory bandwidth)
    - 4-Level ⇒ 5x memory references (20% effective memory bandwidth)
  - Page tables are treated as a linear contiguous array but the array is **mostly sparse**
- Proposed solution: replace the sparse array with a hash table (**one hash table per process**)



Inverted Page Tables

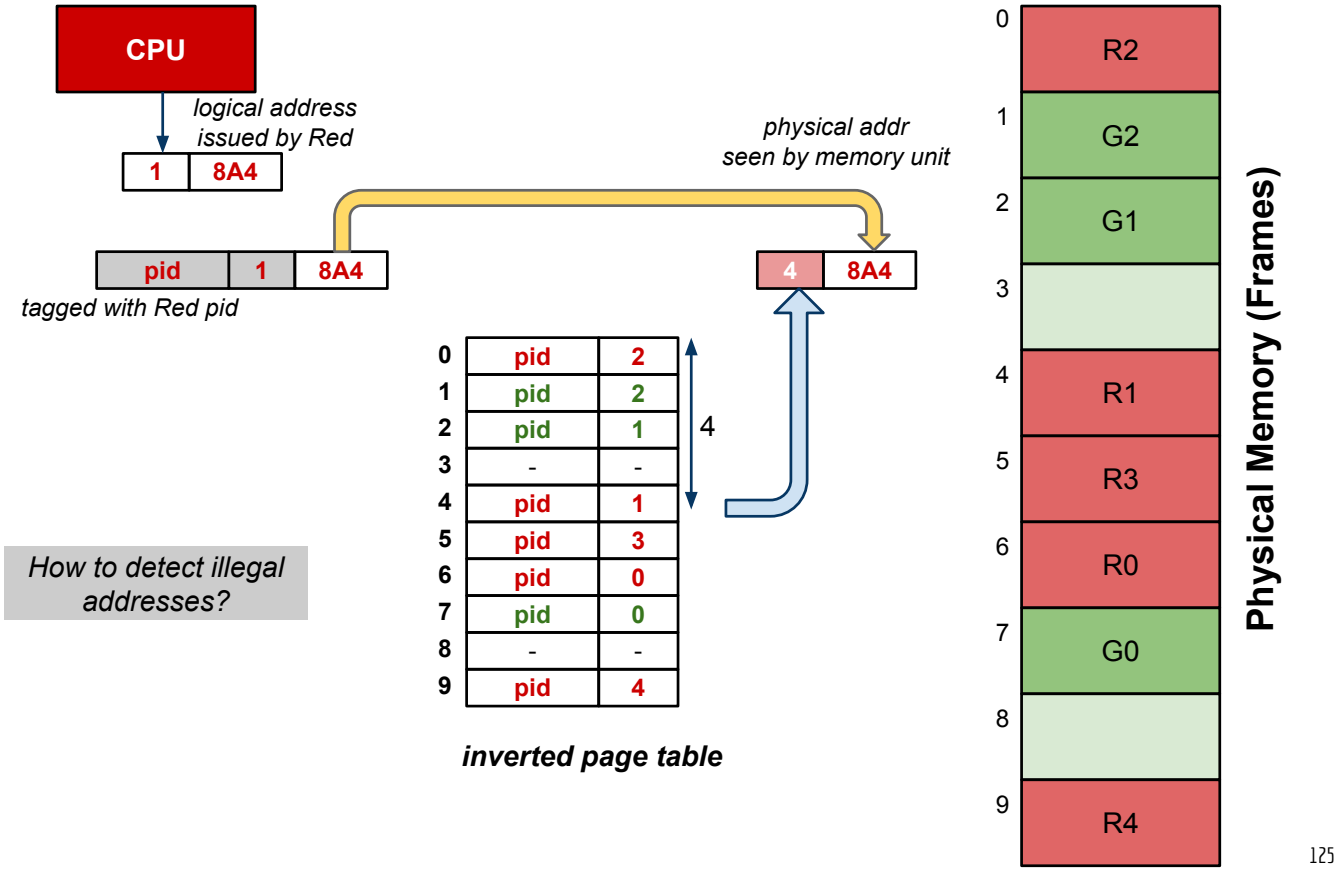
- Observations
  - Each page table may consume lots of memory space
    - Intel/AMD 48-bit 4-level paging consumes 2^{36} x 8 bytes = 2^{39} bytes = 512 GB
  - Each process must maintain its own page table
  - A page table maps a logical page address to a physical frame address
- Proposed Solution
  - Use a **global** "reverse" map that maps physical frame addresses to logical page addresses
  - The number of entries in the inverted table = number of frames in RAM
    - Each entry holds the pair (owner pid, logical page mapped to the frame)
  - Only one global table required (instead of one table per process)

Inverted Page Tables



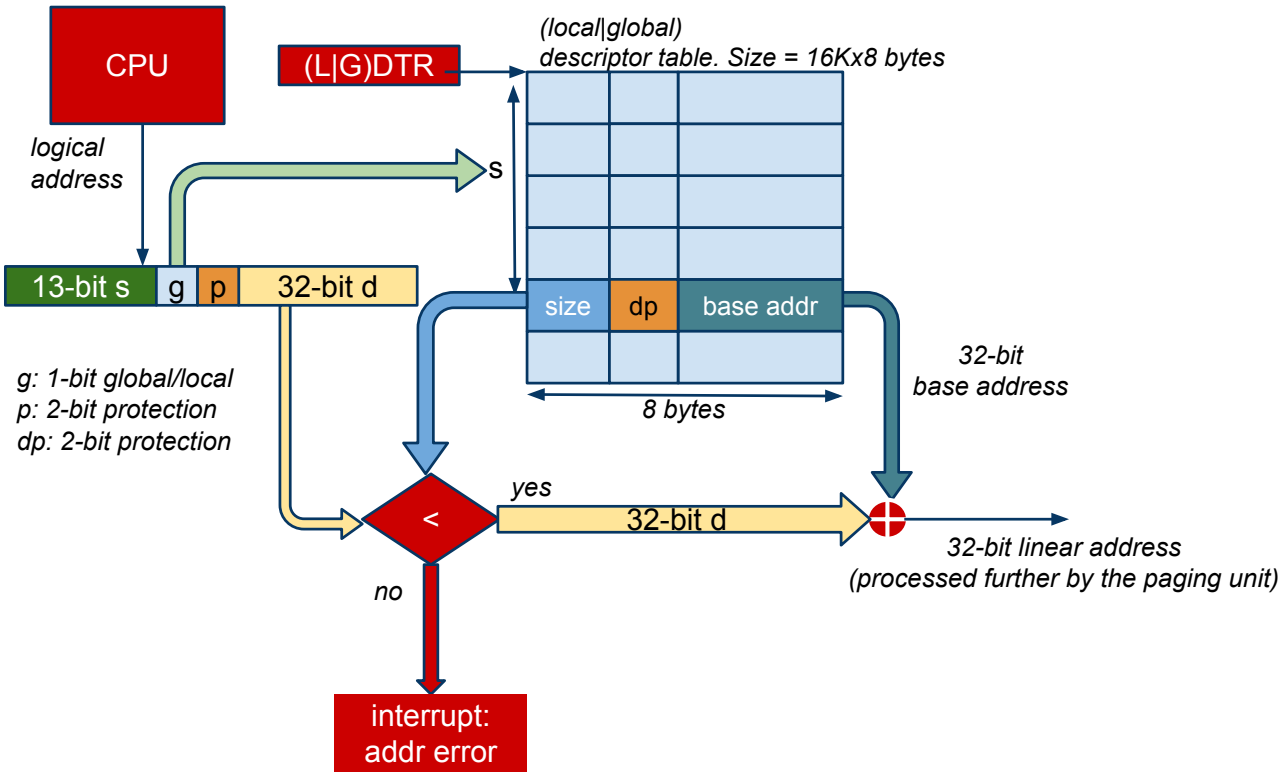
Replace individual page tables with ONE global table

Inverted Page Tables: One "global" page table



IA-32 Segmentation and Paging (Combined)

- IA-32 architecture supports both segmentation and paging
  - Segmentation: *Section 3.4.5 Intel System Programming Guide*
  - Paging: *Section 4.3 Intel System Programming Guide*
- Address translation is a 2-step process
  - The CPU generates the logical address (14-bit segment id, 32-bit offset) + 2-bit protection mode
  - Segmentation hardware translates the 46-bit address into **32-bit linear address (Figure 3-5)**
  - The paging hardware translates the 32-bit linear address to physical frame address (Figures 4-2, 4-3)
- Total 16K (2^{14}) segments
  - 8K private segments (local)
  - 8K shared segments (global)


CPU
(L|G)DTR
(local|global)
descriptor table. Size = 16Kx8 bytes
logical
address
s
13-bit s
g
p
32-bit d
size
dp
base addr
8 bytes
32-bit
base address
g: 1-bit global/local
p: 2-bit protection
dp: 2-bit protection
<
yes
32-bit d
32-bit linear address
(processed further by the paging unit)
no
interrupt:
addr error