

# Week 7

## Memory Management: Demand Paging

*Slides 94-101 updated on Feb 21*

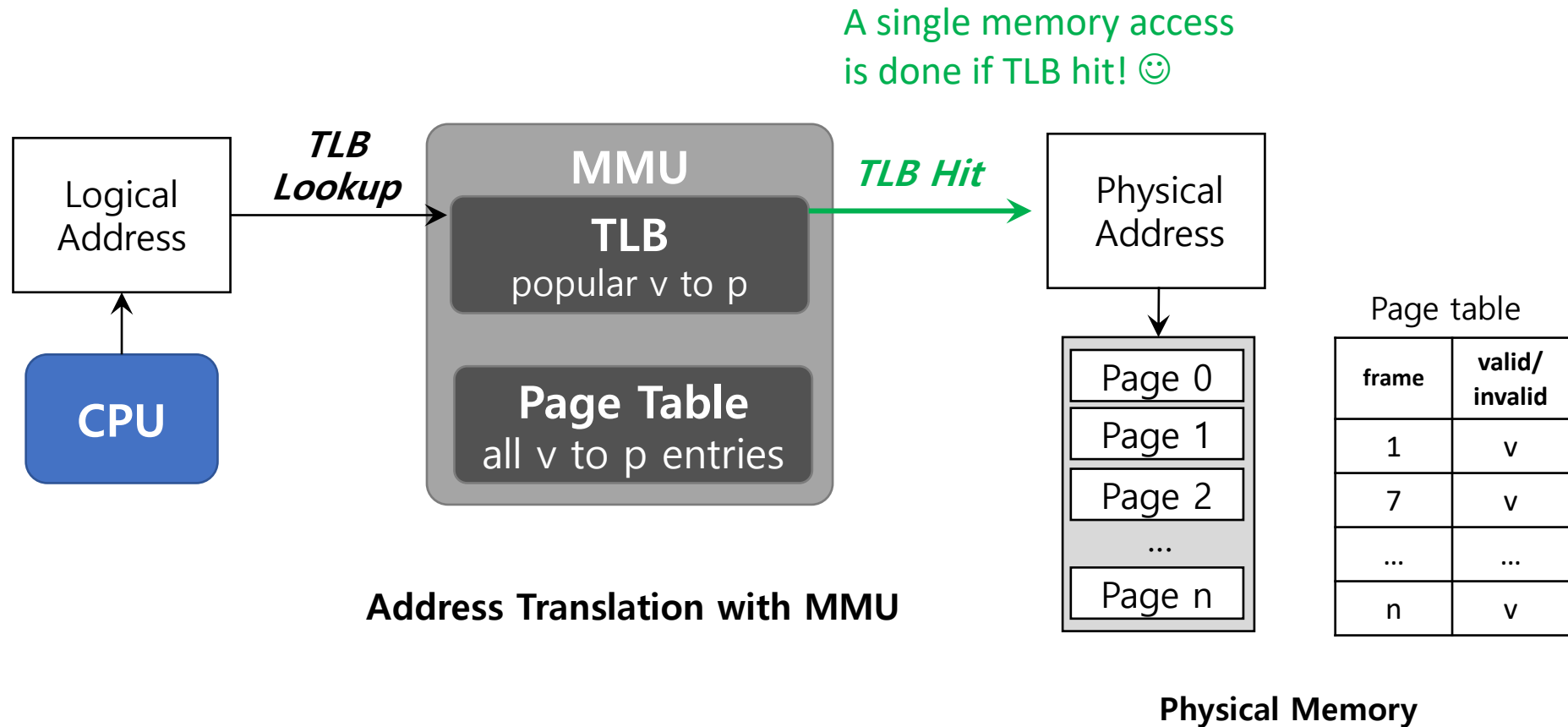
Oana Balmau

February 16, 2023

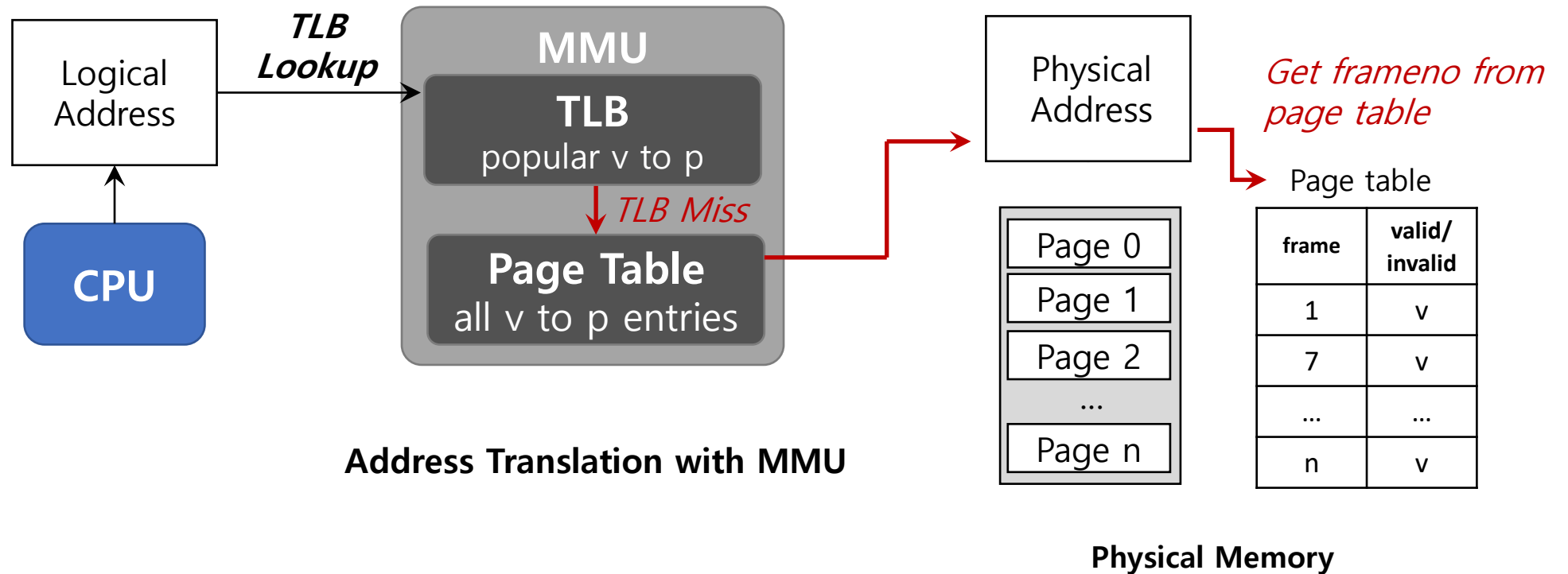
# Recap: Translation Lookaside Buffer (TLB)

- Small fast **hardware cache** of **popular (pageno, frameno) maps**.
- **Part of MMU**
- If mapping for pageno found in TLB
  - Use frameno from TLB
  - Abort mapping using page table
- If not
  - Perform mapping using page table
  - Insert (pageno, frameno) in TLB

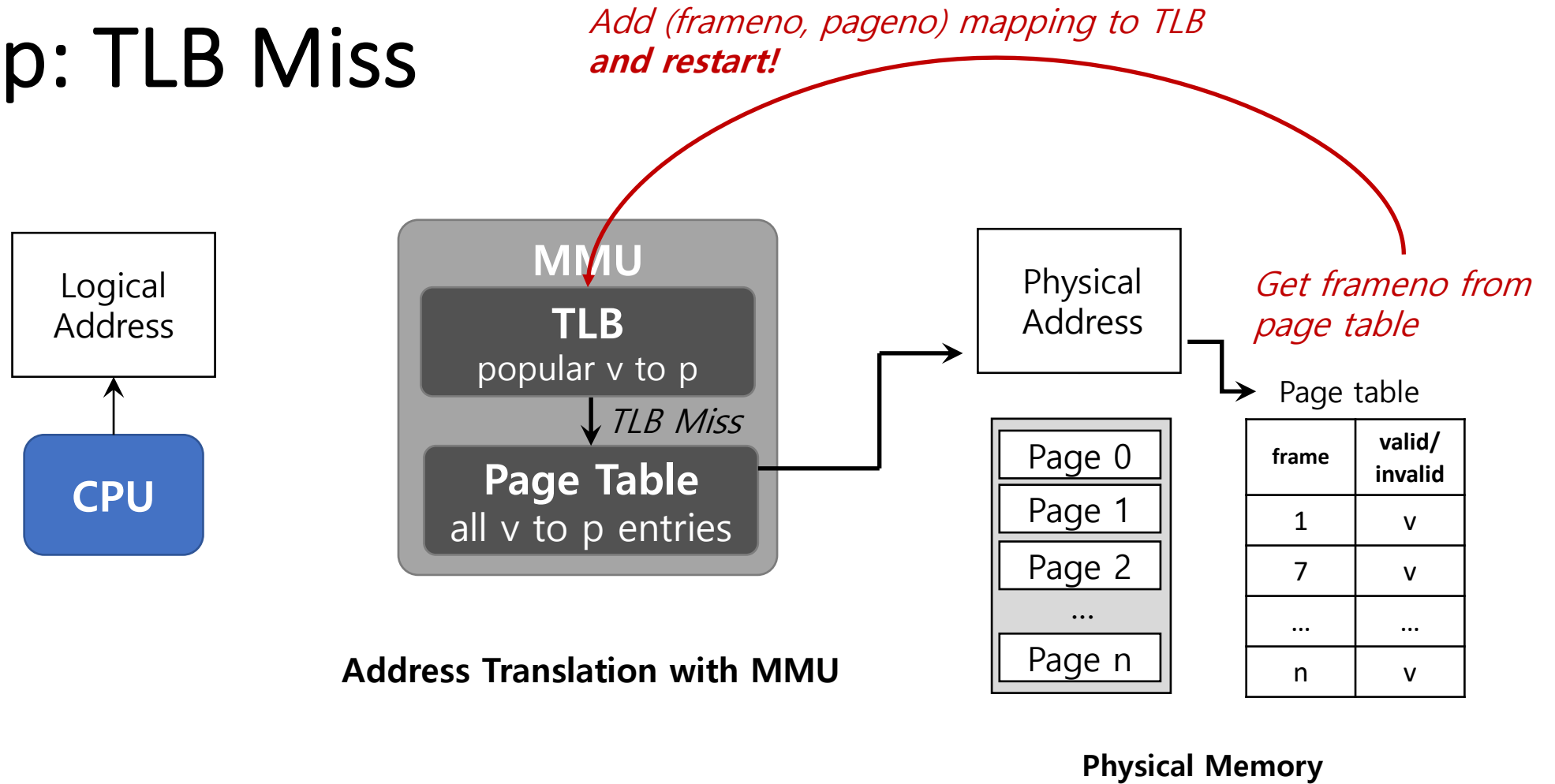
# Recap: TLB Hit



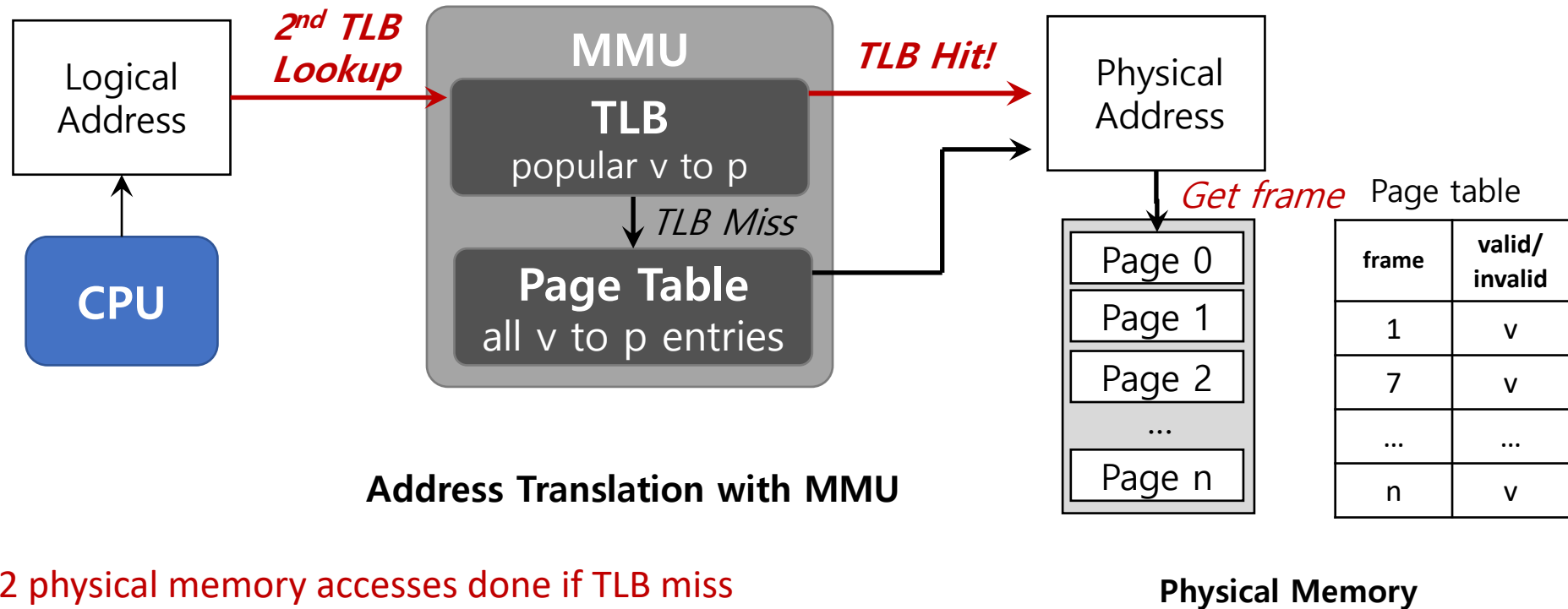
# Recap: TLB Miss



# Recap: TLB Miss



# Recap: TLB Miss



2 physical memory accesses done if TLB miss  
→ Want high TLB hit rate!

# Q&A from Tuesday

- Internal vs External fragmentation
- Associative memory vs fully associative cache
- Size of the PID in the TLB
- Valid/Invalid bit in page table

# Dealing with Large Virtual Address Spaces



# Dealing with Large Virtual Address Spaces

**Example: 64-bit virtual address space (64-bit CPU instructions)**

4kB pages  $\rightarrow$  12-bit page offset

Leaves  $64 - 12 = 52$  bits for pageno  $\rightarrow 2^{52}$  page table entries

Let's say every page table entry 4B

$\rightarrow$  Page table size for one process =  $4B * 2^{52}$  PTEs =  $2^{54}B$

=  $2^4 \times 2^{50} =$  **16 Petabytes (More than main memory!)**

# Dealing with Large Virtual Address Spaces

**Example: 64-bit virtual address space (64-bit CPU instructions)**

4kB pages  $\rightarrow$  12-bit page offset

Leaves 64 – 12 = 52 bits for virtual page number

- Why 4kB pages ?
- How do we get to 12-bit page offset?

Let's say every page table entry 4B

$\rightarrow$  Page table size for one process = 4B \*  $2^{52}$  PTEs =  $2^{54}$ B

=  $2^4 \times 2^{50}$  = **16 Petabytes (More than main memory!)**

# Dealing with Large Virtual Address Spaces

- Why 4kB pages ?

*Typical value for page size; normally, this value is given as part of the problem statement, or you'd have enough information to deduce it.*

# Dealing with Large Virtual Address Spaces

- How do we get to 12-bit page offset?

*Remember paging virtual address:*

| Virtual page number (bits) | Offset (bits) |
|----------------------------|---------------|
|----------------------------|---------------|

*Page size =  $2^{\text{Offset}}$  Bytes . Why?*

- *Every Byte needs to have an address and*
- *We can represent  $2^{\text{Offset}}$  addresses on Offset bits*

00

01

10

11

*Page size = 4KB =  $2^2 \times 2^{10}$  B =  $2^{12}$  B  $\rightarrow$  Offset = 12.*

# Dealing with Large Virtual Address Spaces

**Example: 64-bit virtual address space (64-bit CPU instructions)**

~~4kB pages → 12-bit page offset~~

Leaves  $64 - 12 = 52$  bits for pageno →  $2^{52}$  page table entries

Let's say every page table entry 4B

→ Page table size for one process =  $4B * 2^{52} \text{ PTEs} = 2^{54}B$

=  $2^4 \times 2^{50} =$  **16 Petabytes (More than main memory!)**

# Dealing with Large Virtual Address Spaces

**Example: 64-bit virtual address space (64-bit CPU instructions)**

~~4kB pages → 12-bit page offset~~

Leaves  $64 - 12 = 52$  bits for pageno →  $2^{52}$  page table entries

Let's say even

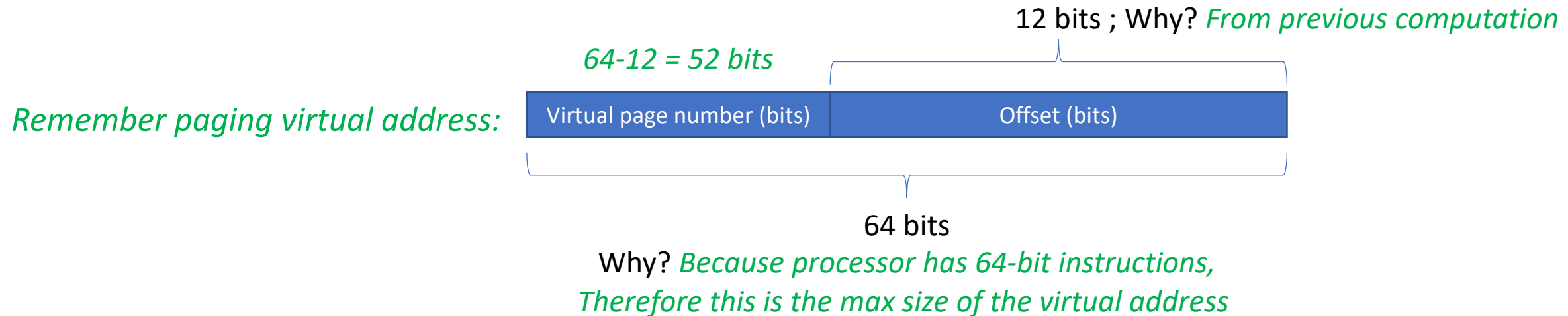
→ Page table

- Why 52 bits for pageno ?
- Why  $2^{52}$  page table entries?

$= 2^4 \times 2^{50} = 16$  Petabytes (More than main memory!)

# Dealing with Large Virtual Address Spaces

Why 52 bits for virt pageno?



# Dealing with Large Virtual Address Spaces

Why  $2^{52}$  Page table entries?

*Remember paging virtual address:*



*If the virtual page number is represented on 52 bits, we have  $2^{52}$  possible virtual page numbers.*

*The page table needs to keep track of all the virtual pages, therefore, it needs to be as big as the total number of virtual pages.*



# Dealing with Large Virtual Address Spaces

~~Example: 64-bit virtual address space (64-bit CPU instructions)~~

~~4kB pages → 12-bit page offset~~

~~Leaves  $64 - 12 = 52$  bits for pageno →  $2^{52}$  page table entries~~

Let's say every page table entry 4B

→ Page table size for one process =  $4B * 2^{52} \text{ PTEs} = 2^{54}B$

=  $2^4 \times 2^{50} =$  **16 Petabytes (More than main memory!)**

# Dealing with Large Virtual Address Spaces

~~Example: 64-bit virtual address space (64-bit CPU instructions)~~

~~4kB pages  $\rightarrow$  12-bit page offset~~

~~Leaves  $64 - 12 = 52$  bits for pageno  $\rightarrow 2^{52}$  page table entries~~

Let's say every page table entry 4B

$\rightarrow$  Page table

$= 2^4 \times$

- **Why?** *This is an assumption*

*The page table entry size is the amount of memory occupied by a page table entry. Depending on what metadata the page table entry stores, the size can range from a few Bytes to a few KB.*

# Dealing with Large Virtual Address Spaces

~~Example: 64-bit virtual address space (64-bit CPU instructions)~~


~~4kB pages → 12-bit page offset~~

~~Leaves  $64 - 12 = 52$  bits for pageno →  $2^{52}$  page table entries~~

~~Let's say every page table entry 4B~~

→ Page table size for one process =  $4B * 2^{52} \text{ PTEs} = 2^{54}B$

=  $2^4 \times 2^{50} B =$  **16 Petabytes (More than main memory!)**

 *1PB =  $2^{50}$  Bytes*

# Dealing with Large Virtual Address Spaces

**Less extreme example: 32-bit virtual address space (32-bit CPU instructions)**

4kB pages → 12-bit page offset, 20 bits for pageno →  $2^{20}$  page table entries

Again assuming every page table entry 4B

→ Page table size for one process =  $2^{20} \times 4\text{B} =$

$$= 4 \times 2^{20} = 4\text{MB}$$

Let's say we run 100 processes in parallel (typical on a standard machine)

**→ 400 MB of main memory used only for PCBs.**

# QUIZ: How big are page Tables?

1. PTEs are **2 bytes**, and **32** possible virtual page numbers

$$32 * 2 \text{ bytes} = 64 \text{ bytes}$$

2. PTEs are **2 bytes**, virtual addrs are **24 bits**, pages are **16 bytes**

$$2 \text{ bytes} * 2^{(24 - \log_2 16)} = 2^{21} \text{ bytes} = 2 * 2^{20} \text{ bytes (2 MB)}$$

3. PTEs are **4 bytes**, virtual addrs are **32 bits**, and pages are **2KB**

$$4 \text{ bytes} * 2^{(32 - \log_2 2K)} = 4 * 2^{21} \text{ bytes (8 MB)}$$

**How big is each page table?**

# How to make Page Table smaller?

- Big pages
- Segmentation + Paging
- Multi-level page tables

# Use Bigger Pages

## 32-bit virtual address space

~~4kB pages~~ **16kB pages**

- 14-bit page offset, 18 bits for pageno
- $2^{18}$  page table entries. Assuming 4B page table entry:
- Page table size =  $4\text{B} \times 2^{18} = \mathbf{1\text{MB}}$

*Factor of 4 reduction size compared to previous example.*

# Use Bigger Pages

- **Advantage:** Easy to implement.
- **Disadvantage:** Larger pages have **higher internal fragmentation**.
- Most systems use 4KB or 8KB pages in common case.



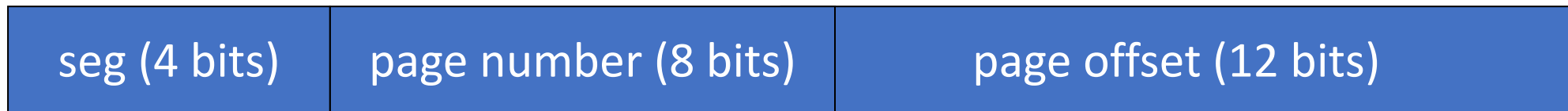
# Segmentation + Paging

Divide address space into segments (code, heap, stack)

- Segments can be variable length

Divide each segment into fixed-sized pages

Virtual address divided into three portions:



# Segmentation + Paging

## Implementation:

- Each segment has a page table
- Each segment track base (physical address) and bounds of **page table** for that segment

# Quiz: Segmentation + Paging

| seg (4 bits) | page number (8 bits) | page offset (12 bits) |
|--------------|----------------------|-----------------------|
|--------------|----------------------|-----------------------|

| seg | base     | bounds | R W |
|-----|----------|--------|-----|
| 0   | 0x002000 | 0xff   | 1 0 |
| 1   | 0x000000 | 0x00   | 0 0 |
| 2   | 0x001000 | 0x0f   | 1 1 |

0x002070 read: **0x004070**

0x202016 read: **0x003016**

0x104c84 read: **error**

0x010424 write: **error**

0x210014 write: **error**

0x203568 read: **0x02a568**

|       |
|-------|
| ...   |
| 0x01f |
| 0x011 |
| 0x003 |
| 0x02a |
| 0x013 |
| ...   |
| 0x00c |
| 0x007 |
| 0x004 |
| 0x00b |
| 0x006 |
| ...   |

0x001000

0x002000

Page Table  
addresses not ordered,  
but segments are contiguous

# Quiz: Segmentation + Paging

Every HEX digit represents  
4 bits

| seg (4 bits) | page number (8 bits) | page offset (12 bits) |
|--------------|----------------------|-----------------------|
|--------------|----------------------|-----------------------|

| seg | base     | bounds | R W |
|-----|----------|--------|-----|
| 0   | 0x002000 | 0xff   | 1 0 |
| 1   | 0x000000 | 0x00   | 0 0 |
| 2   | 0x001000 | 0x0f   | 1 1 |

0x002070 read: 0x004070

0x202016 read: 0x003016

0x104c84 read: error

0x010424 write: error

0x210014 write: error

0x203568 read: 0x02a568

|       |
|-------|
| ...   |
| 0x01f |
| 0x011 |
| 0x003 |
| 0x02a |
| 0x013 |
| ...   |
| 0x00c |
| 0x007 |
| 0x004 |
| 0x00b |
| 0x006 |
| ...   |

0x001000

0x002000

Page Table  
addresses not ordered,  
but segments are contiguous

# Quiz: Segmentation + Paging

Every HEX digit represents 4 bits

| seg (4 bits) | page number (8 bits) | page offset (12 bits) |
|--------------|----------------------|-----------------------|
|--------------|----------------------|-----------------------|

| seg | base     | bounds | R W |
|-----|----------|--------|-----|
| 0   | 0x002000 | 0xff   | 1 0 |
| 1   | 0x000000 | 0x0    |     |
| 2   | 0x001000 | 0x0f   | 1 1 |

within bounds

0x002070 read: 0x004070

0x202016 read: 0x003016

0x104c84 read: error

0x010424 write: error

0x210014 write: error

0x203568 read: 0x02a568

|       |      |            |
|-------|------|------------|
| ...   |      |            |
| 0x01f |      | 0x001000   |
| 0x011 |      |            |
| 0x003 |      |            |
| 0x02a |      |            |
| 0x013 |      |            |
| ...   |      |            |
| 0x00c | 0x00 | 0x002000   |
| 0x007 | 0x01 |            |
| 0x004 | 0x02 | 0x004000 + |
| 0x00b |      | 0x000070   |
| 0x006 |      |            |
| ...   |      |            |

Page Table

# Quiz: Segmentation + Paging

Every HEX digit represents 4 bits

| seg (4 bits) | page number (8 bits) | page offset (12 bits) |
|--------------|----------------------|-----------------------|
|--------------|----------------------|-----------------------|

| seg | base     | bounds | R W |
|-----|----------|--------|-----|
| 0   | 0x002000 | 0xff   | 1 0 |
| 1   | 0x000000 | 0x00   | 0 0 |
| 2   | 0x001000 | 0x0f   | 1 1 |

within bounds

0x002070 read: 0x004070

0x202016 read: 0x003016

0x104c84 read: error

0x010424 write: error

0x210014 write: error

0x203568 read: 0x02a568

|            |
|------------|
| ...        |
| 0x01f 0x00 |
| 0x011 0x01 |
| 0x003 0x02 |
| 0x02a      |
| 0x013      |
| ...        |
| 0x00c      |
| 0x007      |
| 0x004      |
| 0x00b      |
| 0x006      |
| ...        |

0x001000

0x003000 +  
0x000016

0x002000

Page Table

# Quiz: Segmentation + Paging

Every HEX digit represents 4 bits

| seg (4 bits) | page number (8 bits) | page offset (12 bits) |
|--------------|----------------------|-----------------------|
|--------------|----------------------|-----------------------|

| seg | base     | bounds | R W |
|-----|----------|--------|-----|
| 0   | 0x002000 | 0xff   | 1 0 |
| 1   | 0x000000 | 0x00   | 0 0 |
| 2   | 0x001000 | 0x0f   | 1 1 |

0x002070 read: 0x004070  
 0x202016 read: 0x003016 No permission  
 0x104c84 read: error  
 0x010424 write: error  
 0x210014 write: error  
 0x203568 read: 0x02a568

|       |          |
|-------|----------|
| ...   |          |
| 0x01f | 0x001000 |
| 0x011 |          |
| 0x003 |          |
| 0x02a |          |
| 0x013 |          |
| ...   |          |
| 0x00c | 0x002000 |
| 0x007 |          |
| 0x004 |          |
| 0x00b |          |
| 0x006 |          |
| ...   |          |

Page Table

# Quiz: Segmentation + Paging

Every HEX digit represents 4 bits

| seg (4 bits) | page number (8 bits) | page offset (12 bits) |
|--------------|----------------------|-----------------------|
|--------------|----------------------|-----------------------|

| seg | base     | bounds | R W |
|-----|----------|--------|-----|
| 0   | 0x002000 | 0xff   | 1 0 |
| 1   | 0x000000 | 0x00   | 0 0 |
| 2   | 0x001000 | 0x0f   | 1 1 |

0x002070 read: 0x004070  
 0x202016 read: 0x003016  
 0x104c84 read: error No permission  
 0x010424 write: error  
 0x210014 write: error  
 0x203568 read: 0x02a568

|       |          |
|-------|----------|
| ...   |          |
| 0x01f | 0x001000 |
| 0x011 |          |
| 0x003 |          |
| 0x02a |          |
| 0x013 |          |
| ...   |          |
| 0x00c | 0x002000 |
| 0x007 |          |
| 0x004 |          |
| 0x00b |          |
| 0x006 |          |
| ...   |          |

Page Table



# Quiz: Segmentation + Paging

Every HEX digit represents 4 bits

| seg (4 bits) | page number (8 bits) | page offset (12 bits) |
|--------------|----------------------|-----------------------|
|--------------|----------------------|-----------------------|

| seg | base     | bounds | R W |
|-----|----------|--------|-----|
| 0   | 0x002000 | 0xff   | 1 0 |
| 1   | 0x000000 | 0x00   | 0 0 |
| 2   | 0x001000 | 0x0f   | 1 1 |

**Out of bounds**

0x002070 read: 0x004070

0x202016 read: 0x003016

0x104c84 read: error

0x010424 write: error

0x210014 write: error

0x203568 read: 0x02a568

|       |          |
|-------|----------|
| ...   |          |
| 0x01f | 0x001000 |
| 0x011 |          |
| 0x003 |          |
| 0x02a |          |
| 0x013 |          |
| ...   |          |
| 0x00c | 0x002000 |
| 0x007 |          |
| 0x004 |          |
| 0x00b |          |
| 0x006 |          |
| ...   |          |

Page Table

# Quiz: Segmentation + Paging

Every HEX digit represents 4 bits

| seg (4 bits) | page number (8 bits) | page offset (12 bits) |
|--------------|----------------------|-----------------------|
|--------------|----------------------|-----------------------|

| seg | base     | bounds | R W |
|-----|----------|--------|-----|
| 0   | 0x002000 | 0xff   | 1 0 |
| 1   | 0x000000 | 0x00   | 0 0 |
| 2   | 0x001000 | 0x0f   | 1 1 |

within bounds

0x002070 read: 0x004070

0x202016 read: 0x003016

0x104c84 read: error

0x010424 write: error

0x210014 write: error

0x203568 read: 0x02a568

|            |
|------------|
| ...        |
| 0x01f 0x00 |
| 0x011 0x01 |
| 0x003 0x02 |
| 0x02a 0x03 |
| 0x013      |
| ...        |
| 0x00c      |
| 0x007      |
| 0x004      |
| 0x00b      |
| 0x006      |
| ...        |

0x001000

0x02a000 +

0x000568

0x002000

Page Table

# Advantages of Segmentation + Paging

## + Supports sparse address spaces

- Decreases size of page tables
- If segment not used, not need for page table

## + Sharing

## + No external fragmentation

# Disadvantages of Segmentation + Paging

- ☹ Potentially large page tables (for each segment)
- ☹ Must allocate each page table contiguously.
  - Can get tricky with large page table

# Multi-level Page Tables

Turns the linear page table we've seen so far into a tree structure.

# Multi-level Page Tables

Turns the linear page table we've seen so far into a tree structure.

## 2-level Page Table:

- Chop up the page table into page-sized units.
- If an entire page of page-table entries is invalid, don't allocate that page of the page table at all.
- To track if a page of page table is valid, use a **page directory** (new structure).

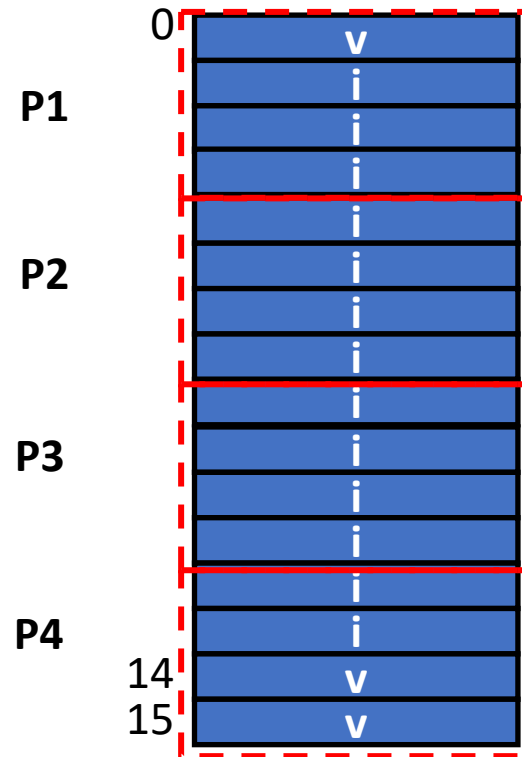
# Flat (1-Level) Page Table

Page table size:  
 $2^4 = 16$  entries

|    |   |
|----|---|
| 0  | v |
|    | i |
|    | i |
|    | i |
|    | i |
|    | i |
|    | i |
|    | i |
|    | i |
|    | i |
|    | i |
|    | i |
|    | i |
|    | i |
|    | i |
| 14 | v |
| 15 | v |

# 1-Level Page Table → 2-Level Page Table

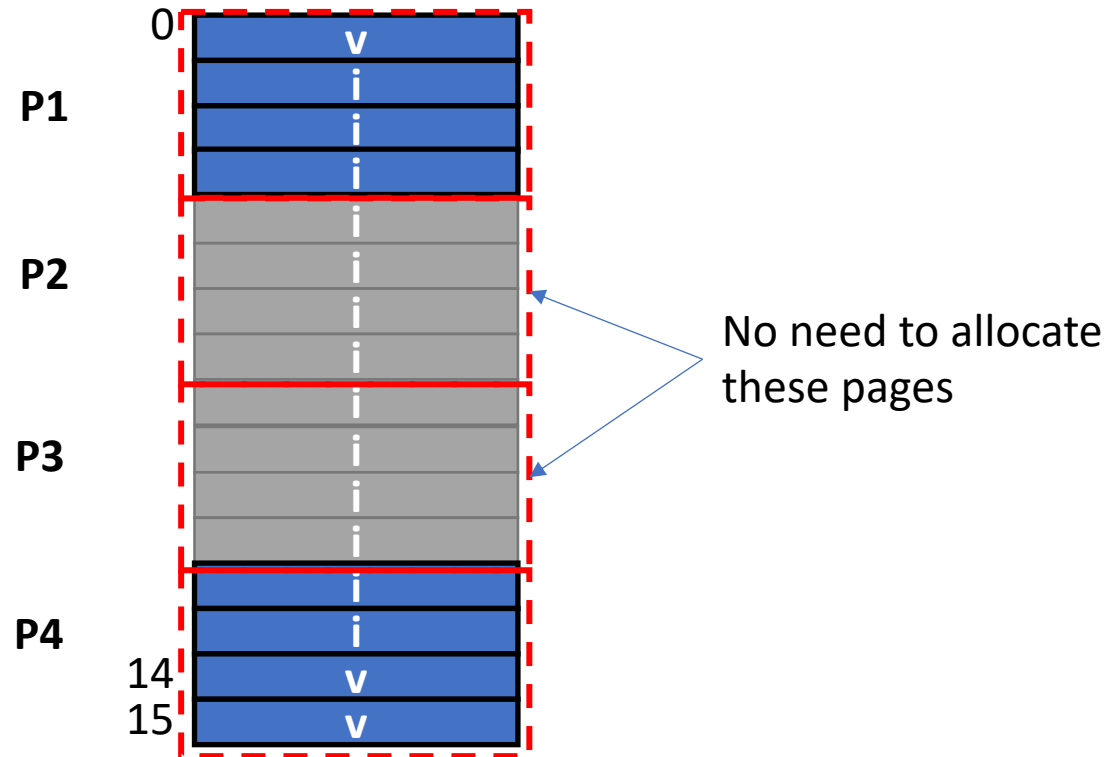
Page table size:  
 $2^4 = 16$  entries





# 1-Level Page Table → 2-Level Page Table

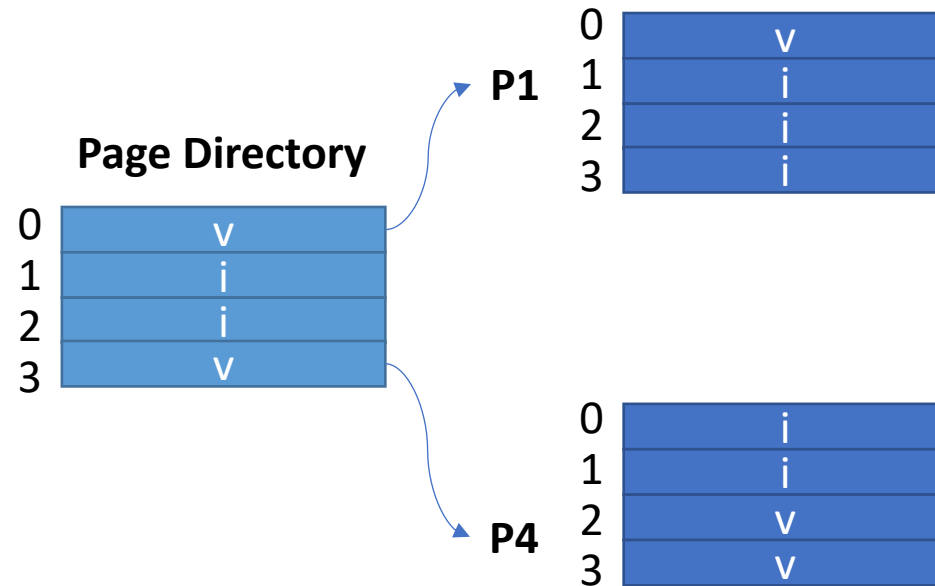
Page table size:  
 $2^4 = 16$  entries



# 2-Level Page Table

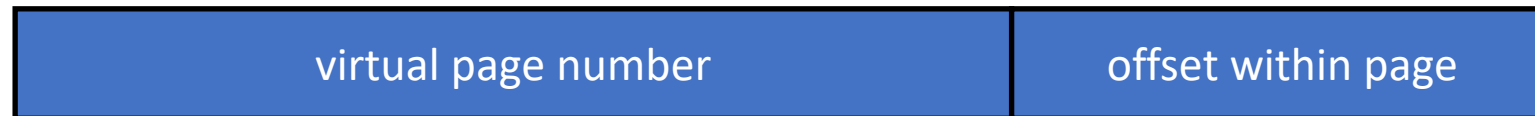
**Page table size (page directory + 2 page tables)**

$2^2 + 2 \times 2^2 = 12$  entries ( $< 16$ )

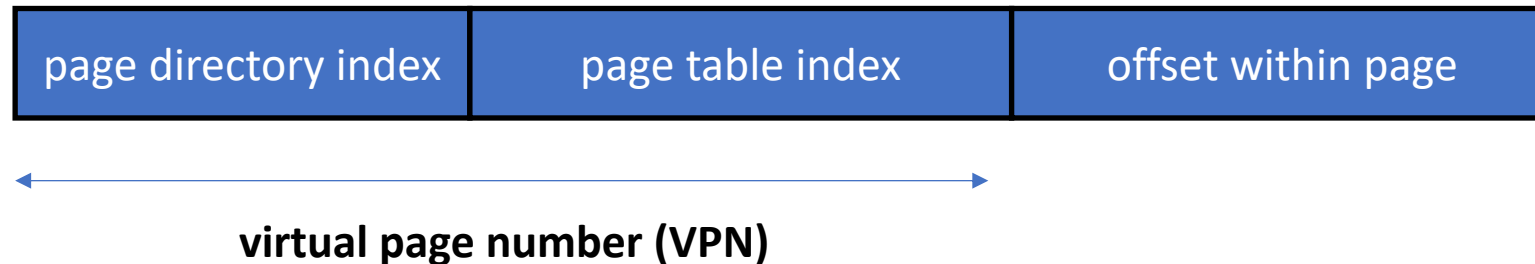


# Virtual Address

- Single-level page table



- 2-level page table



# Why Useful?

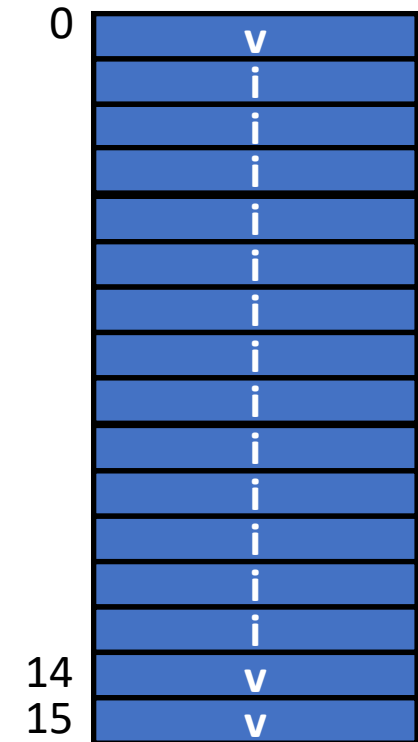
- **For sparse address spaces**
  - Most address spaces are sparsely populated

## One-level page table:

- Need page table for entire address space

## Two-level page table:

- Need top-level page table for entire address space
- Need only second-level page tables for populated parts of the address space



# Let's practice!

- Virtual address – 32 bits
- Only low 20MB and upper 2MB valid
- Page size – 4k or offset = 12 bits, so VPN = 20 bits

What is the size of a 1-level page table (#PTEs)?

What is the size of a 2-level page table with 8-bit page directory and 12-bit level 2 pages (#PTEs)?

# Size of a 1-level page table (#PTEs)?

- Virtual address – 32 bits
- Only low 20MB and upper 2MB valid
- Page size – 4k or offset = 12 bits, so VPN = 20 bits

What is the size of a 1-level page table (#PTEs)?

| Virtual page number (bits) | Offset (bits) |
|----------------------------|---------------|
| 20 bits                    | 12 bits       |

*For a 1-level page table, this is irrelevant because we need to represent all the virtual pages.*

*The VPN # of bits and the offset are computed like in the first example, with the exception that this time the CPU has 32-bit instructions, so the total number of bits in a virtual address is 32.*

*The #PTEs is computed like in the first example, by using the number of bits in the VPN:*

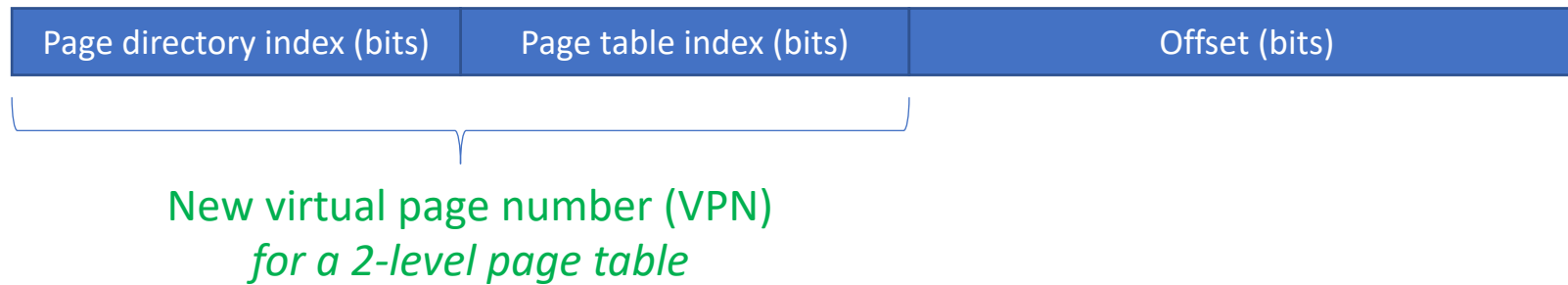
*#PTEs =  $2^{20}$  = 1,048,576 page table entries*

*Note that in this example we stop at the number of page table entries, without computing the size of the Page table. If we wanted to compute the size of the page table we would need to multiply  $2^{20}$  by the Size of a page table entry (in the first example, we estimated it at 4B)*

# Size of a 2-level page table (#PTEs)?

What is the size of a 2-level page table with 8-bit page directory and 12-bit level 2 pages (#PTEs)?

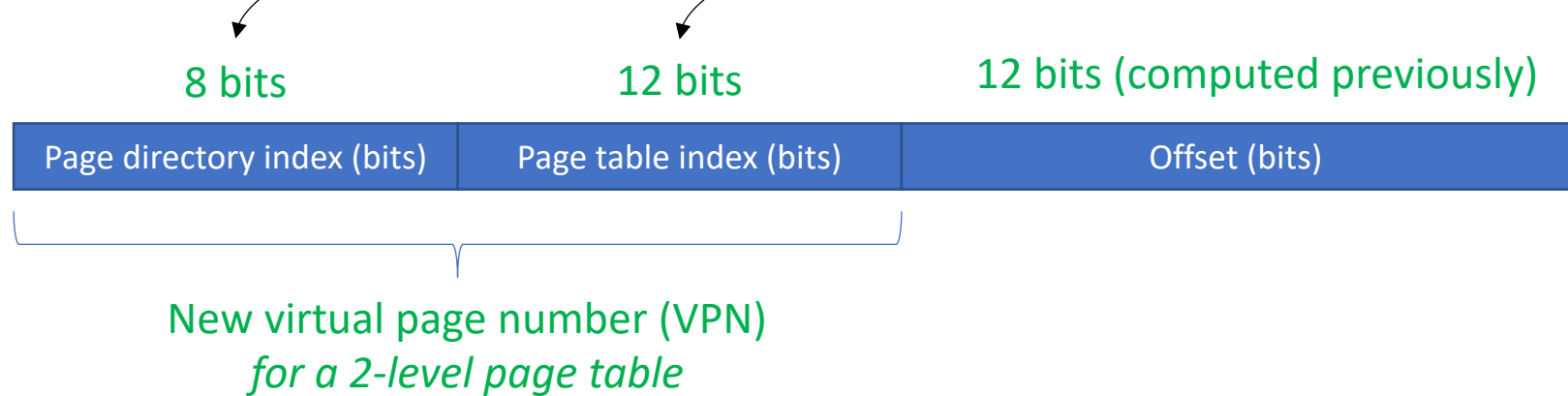
*Virtual address structure changes with 2-level page table:*



# Size of a 2-level page table (#PTEs)?

What is the size of a 2-level page table with 8-bit page directory and 12-bit level 2 pages (#PTEs)?

*Virtual address structure changes with 2-level page table:*

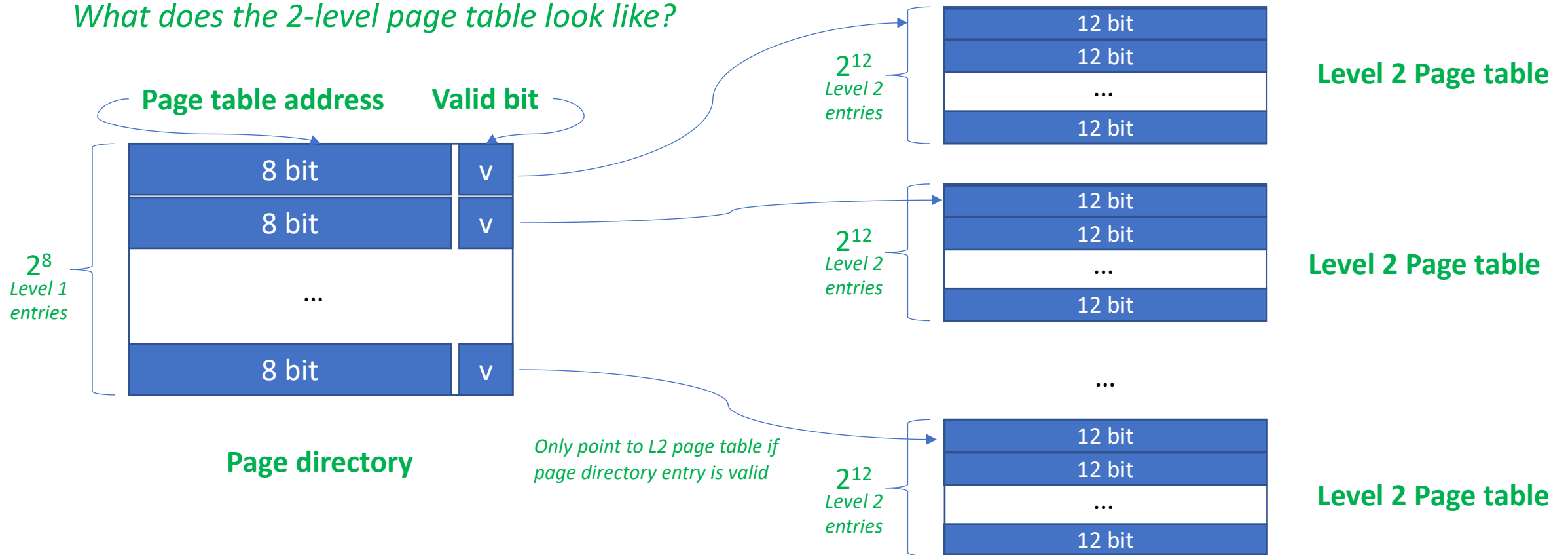




# Size of a 2-level page table (#PTEs)?

What is the size of a 2-level page table with 8-bit page directory and 12-bit level 2 pages (#PTEs)?

*What does the 2-level page table look like?*



# Size of a 2-level page table (#PTEs)?

Only low 20MB and upper 2MB valid

- *How many page directory entries are valid?*
- *How many level 2 page tables are allocated?*
- *How many total page table entries with a 2-level page table and a program with sparse address space?*

# Size of a 2-level page table (#PTEs)?

- *How many level 2 page tables are allocated?*

*Sparse address space → we will need at least one 2<sup>nd</sup> level page table for low 20MB and one 2<sup>nd</sup> level page table for upper 2MB*

*How large of an address space can we reference with a single level2 page table?*

- *we know the size of one PTE on level 2 is 12 bits → we can reference  $2^{12}$  pages*
- *each page has 4KB → one level2 page table can reference  $4KB * 2^{12} = 2^2 * 2^{10} * 2^{12} B = 2^4 * 2^{20} B$   
 $= 16MB$*

*→ We need 2 level2 page tables to reference low 20MB + 1 level2 page table to reference upper 2MB.*

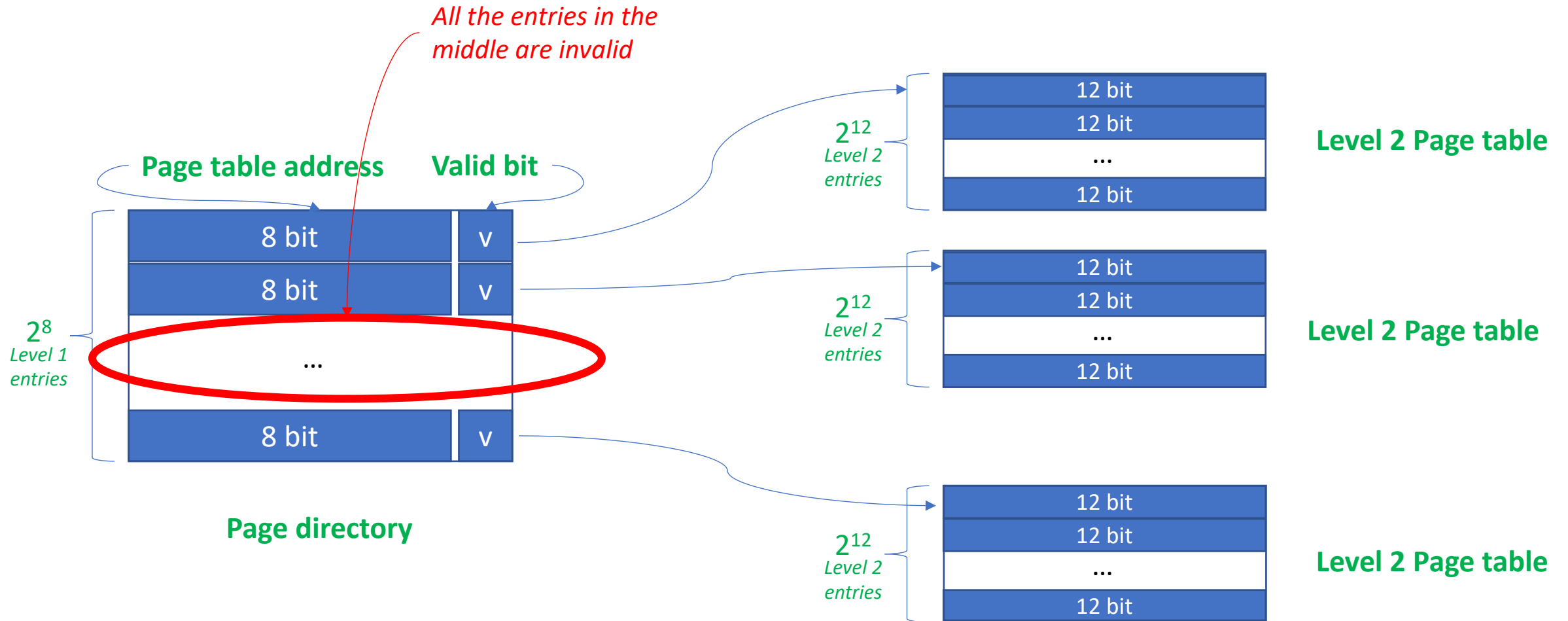
# Size of a 2-level page table (#PTEs)?

- *How many page directory entries are valid?*

*Only 3 page directory entries are valid.*

*Why? Because we need 2 level2 page tables to reference low 20MB + 1 level2 page table to reference upper 2MB.*

# Size of a 2-level page table (#PTEs)?



# Size of a 2-level page table (#PTEs)?

- *How many total page table entries with a 2-level page table and a program with sparse address space?*
- $2^8$  PTEs for 1<sup>st</sup> level
- $2 \times 2^{12} + 1 \times 2^{12}$  PTEs for 2<sup>nd</sup> level;
- Total =  $2^8 + 3 \times 2^{12} = 12,544$  PTEs

# 2-Level Page Tables for Dense Address Spaces?

- Not useful
- In fact, counter-productive (Why?)
- But most address spaces are sparse

# Dense Address Space 1-Level Page Table

Page table size:  
 $2^4 = 16$  entries

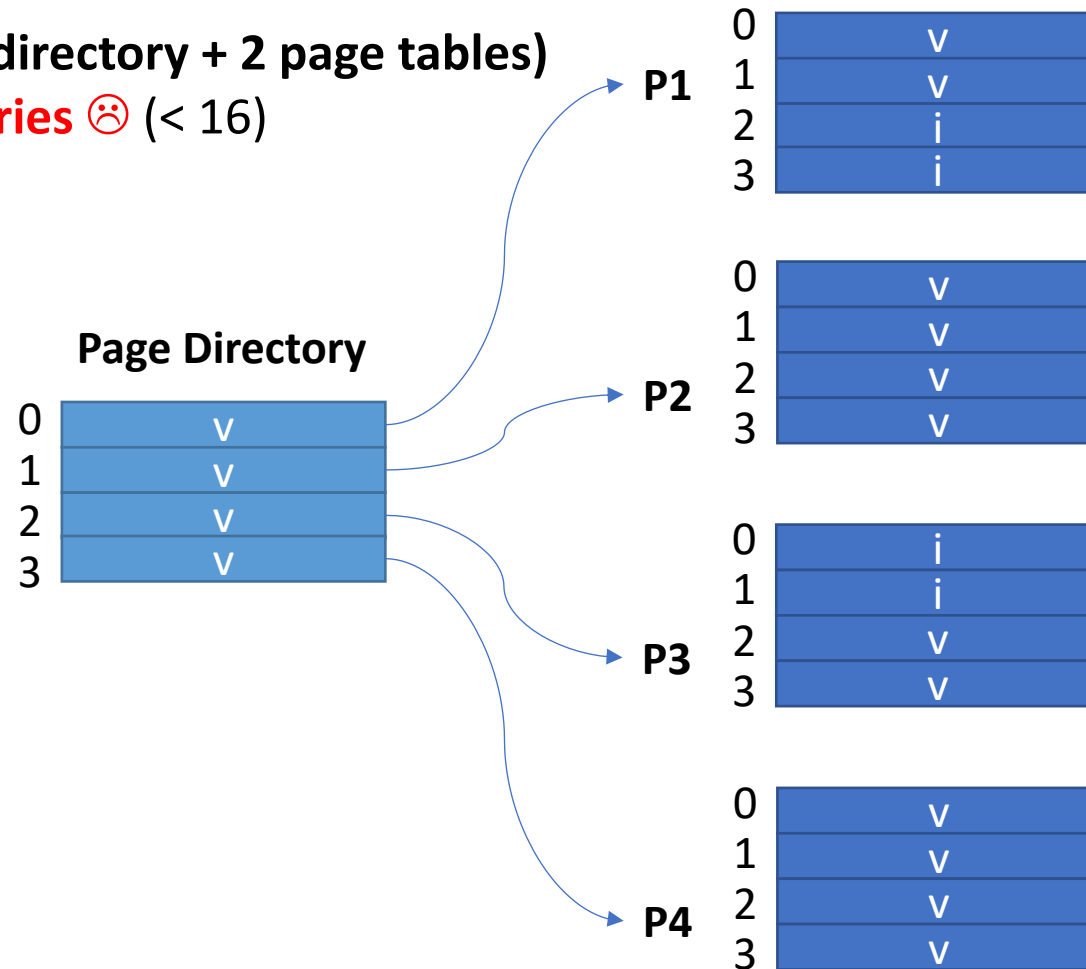
|    |   |
|----|---|
| 0  | v |
|    | v |
|    | i |
|    | i |
|    | v |
|    | v |
|    | v |
|    | v |
|    | i |
|    | i |
|    | v |
|    | v |
|    | v |
|    | v |
| 14 | v |
| 15 | v |



# Dense Address Space 2-Level Page Table

Page table size (page directory + 2 page tables)

$2^2 + 4 \times 2^2 = \mathbf{20 \text{ entries}}$  😞 ( $< 16$ )



# Are Two Levels Enough?

- Need top-level page table (page directory) for entire address space.
- Assume Size second-level page table == size of page
  - Why? Easy to allocate

**Problem:** Top-level can get too large if large address space (64 bits)

**Solution (?): More levels.**

# More Levels: The Price to Be Paid

- Each level adds another memory access
- N-level page table
  - 1 memory access → N+1 memory accesses ( N for page table + 1 for phys addr)
- But, TLB still works
  - If TLB hit, 1 memory access → 1 memory accesses
  - If miss, 1 memory access → N+1 memory accesses

→ TLB hit rate must be very high (99+ %)

# Further Reading

## **Operating Systems: Three Easy Pieces by R. & A. Arpaci-Dusseau**

Chapters 19–22

<https://pages.cs.wisc.edu/~remzi/OSTEP/>

### **Credits:**

Some slides adapted from the OS courses of Profs. Remzi and Andrea Arpaci-Dusseau (University of Wisconsin-Madison), Prof. Willy Zwaenepoel (University of Sydney), and Prof. Youjip Won (Hanyang University).