# [537] Smaller Page Tables

Tyler Harter
9/24/14

# Worksheets

Problem 1: how many accesses with TLB?

Problem 2: how large are PTEs?

- tip, use indexes to save memory

Problem 3: how large are PTs?

## Smaller Page Tables

## Paging Problems

Too slow [last time]

Too big [today’s focus]

## Motivation

Why do we want big virtual address spaces?

# Motivation

Why do we want big virtual address spaces?

- programming is easier
- applications need not worry (as much) about fragmentation

## Motivation

Why do we want big virtual address spaces?

- programming is easier
- applications need not worry (as much) about fragmentation

Paging goals:

- space efficiency (don’t waste on invalid data)
- simplicity (no bookkeeping should require contiguous pages)

# Motivation

Why do we want big virtual address spaces?

- programming is easier
- applications need not worry (as much) about fragmentation

Paging goals:

- **space efficiency (don't waste on invalid data)**
- simplicity (no bookkeeping should require contiguous pages)

## Approach 1: Change Page Size

Make pages bigger

Worksheet: Problem 4

# Approach 1: Change Page Size

Make pages bigger

Worksheet: Problem 4

Why are 4 MB pages bad?

## Approach 1: Change Page Size

Make pages bigger

Worksheet: Problem 4

Why are 4 MB pages bad?  Internal fragmentation.

# Mixed Page Sizes

Some systems support multiple page sizes
- better TLB is bigger motivation, though

Mechanisms: what are implications for
- PTs?
- TLBs?

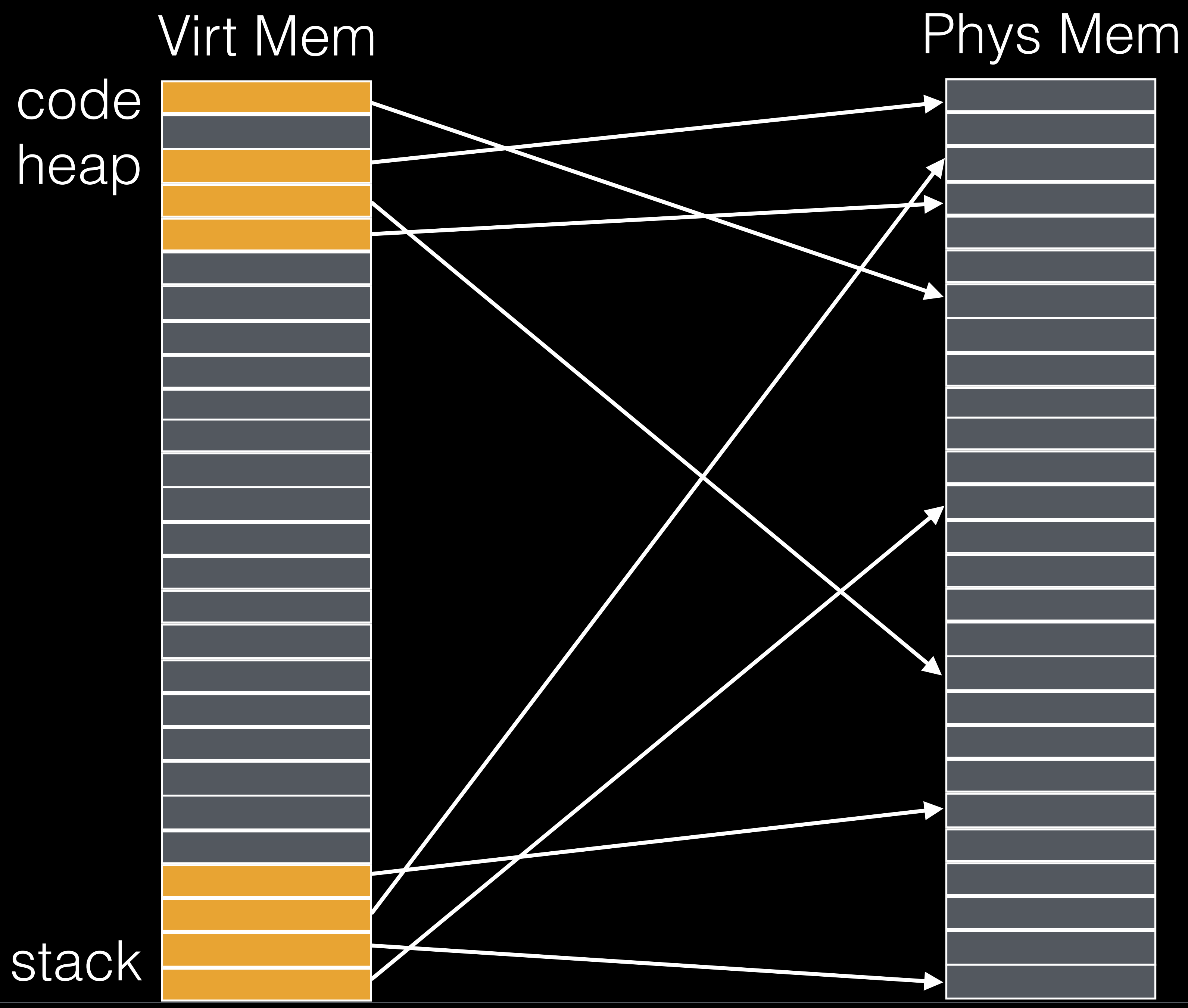
Policy: when to use large pages?

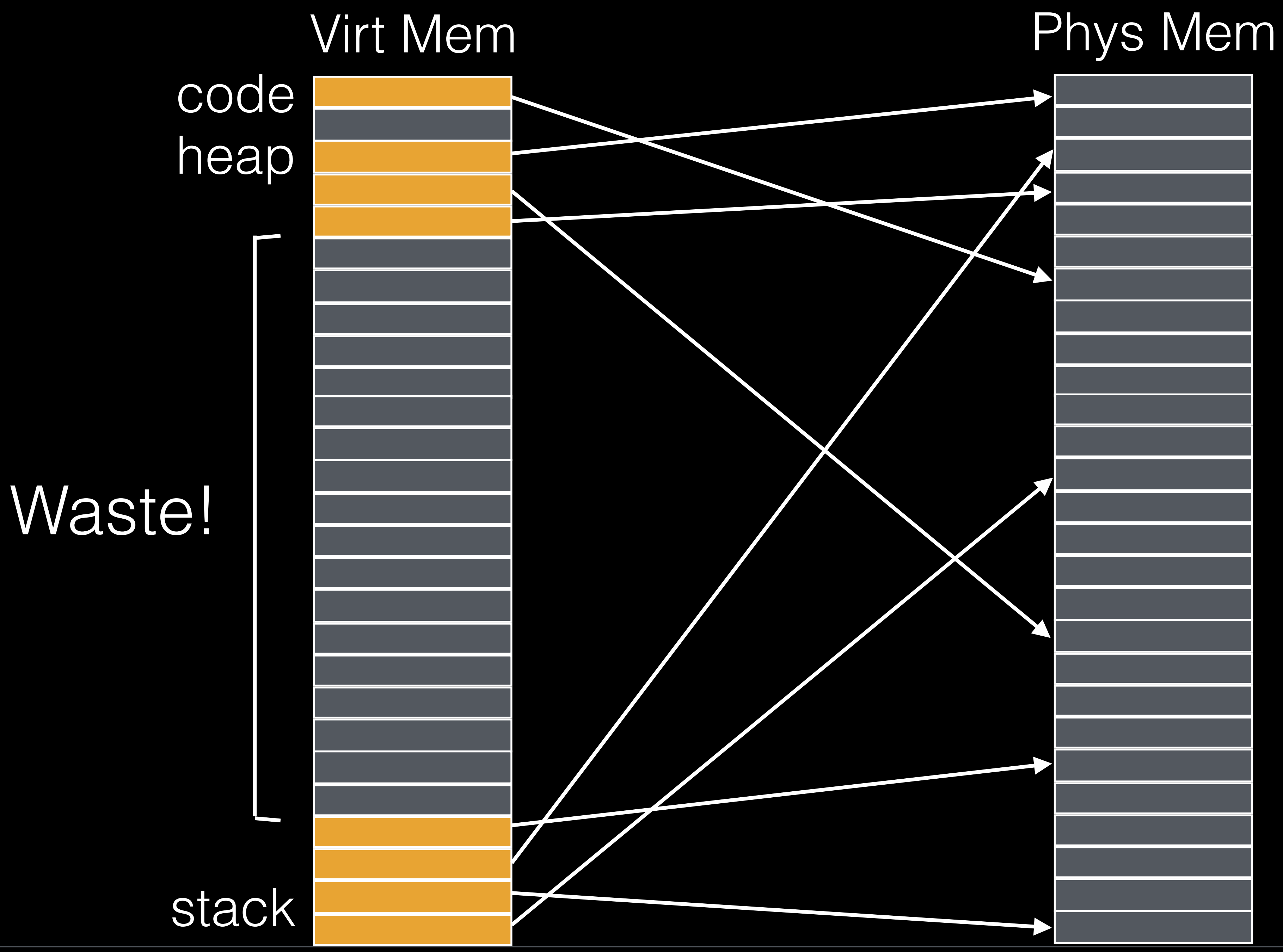
## Approach 2: abandon simple linear page tables

Use more complex PTs, instead of just a big array.

Suggestions?

Look at problem more closely…


Virt Mem
Phys Mem
code
heap
stack


Virt Mem
Phys Mem
code
heap
Waste!
stack

# Many invalid PT entries

| PFN | valid | prot |
|---|---|---|
| 10 | 1 | r-x |
| - | 0 | - |
| 23 | 1 | rw- |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| *…many more invalid…* | | |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| 28 | 1 | rw- |
| 4 | 1 | rw- |

# Many invalid PT entries

| PFN | valid | prot |
|---|---|---|
| 10 | 1 | r-x |
| - | 0 | - |
| 23 | 1 | rw- |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| *…many more invalid…* | | |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| 28 | 1 | rw- |
| 4 | 1 | rw- |

how to avoid storing these?

## Approach 2a: hash-table lookup

Called an inverted page table.

Pros/Cons?

## Approach 2a: hash-table lookup

Called an inverted page table.

Pros/Cons?

Nice if we trapped on TLB misses…

# Many invalid PT entries

| PFN | valid | prot |
|---|---|---|
| 10 | 1 | r-x |
| - | 0 | - |
| 23 | 1 | rw- |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| *…many more invalid…* | | |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| 28 | 1 | rw- |
| 4 | 1 | rw- |

how to avoid storing these?

# Many invalid PT entries

| PFN | valid | prot |
|---|---|---|
| 10 | 1 | r-x |
| - | 0 | - |
| 23 | 1 | rw- |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| *…many more invalid…* | | |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| 28 | 1 | rw- |
| 4 | 1 | rw- |

how to avoid storing these?

Note there is a big "hole" in our addr space: invalids are clustered.

# Many invalid PT entries

| PFN | valid | prot |
|---|---|---|
| 10 | 1 | r-x |
| - | 0 | - |
| 23 | 1 | rw- |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| *…many more invalid…* | | |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| 28 | 1 | rw- |
| 4 | 1 | rw- |

how to avoid storing these?

Note there is a big "hole" in our addr space: invalids are clustered.

How did we fix holes in phys memory before?

# Many invalid PT entries

| PFN | valid | prot |
|---|---|---|
| 10 | 1 | r-x |
| - | 0 | - |
| 23 | 1 | rw- |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| *…many more invalid…* | | |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| - | 0 | - |
| 28 | 1 | rw- |
| 4 | 1 | rw- |

how to avoid storing these?

Note there is a big “hole” in our addr space: invalids are clustered.

How did we fix holes in phys memory before?
- segmentation
- paging

# Approach 2

**Approach 2a**: hashtable

**Approach 2b**: segments over PTs

**Approach 2c**: PTs over PTs

# Approach 2

**Approach 2a**: hashtable

**Approach 2b**: segments over PTs

**Approach 2c**: PTs over PTs

**Approach 2d**: PTs over PTs over PTs over PTs

**for fun!**

## Approach 2

**Approach 2a**: hashtable

**Approach 2b**: segments over PTs

**Approach 2c**: PTs over PTs

**Approach 2d**: PTs over PTs over PTs

## Approach 2

**Approach 2a**: hashtable

**Approach 2b**: segments over PTs

**Approach 2c**: PTs over PTs

**Approach 2d**: PTs over PTs over PTs

## Segmentation/Paging Hybrid

Idea: use different page tables for heap, stack, etc

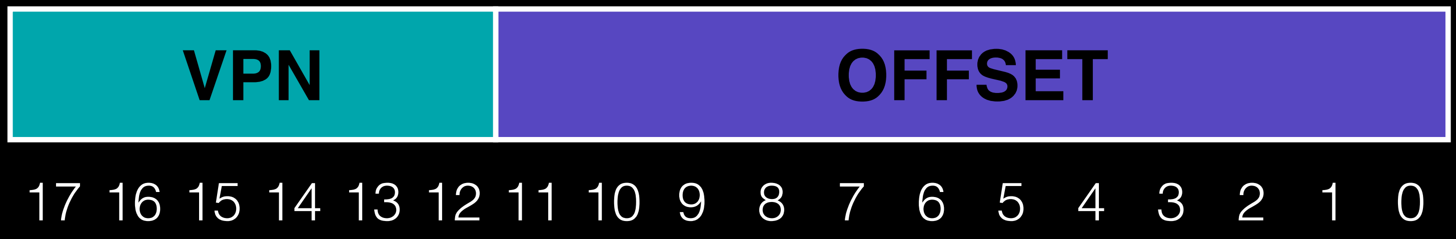
# Segmentation/Paging Hybrid

Idea: use different page tables for heap, stack, etc
- each PT can be a different size
- each PT has a base/bounds (where?)

# Segmentation/Paging Hybrid

Idea: use different page tables for heap, stack, etc
- each PT can be a different size
- each PT has a base/bounds (where?)

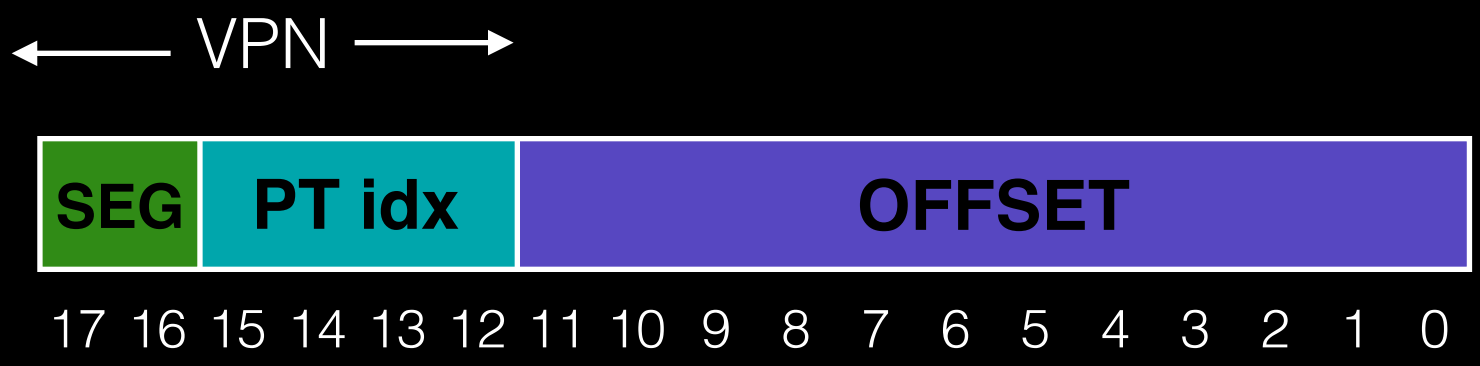
Before:



# Segmentation/Paging Hybrid

Idea: use different page tables for heap, stack, etc
- each PT can be a different size
- each PT has a base/bounds (where?)

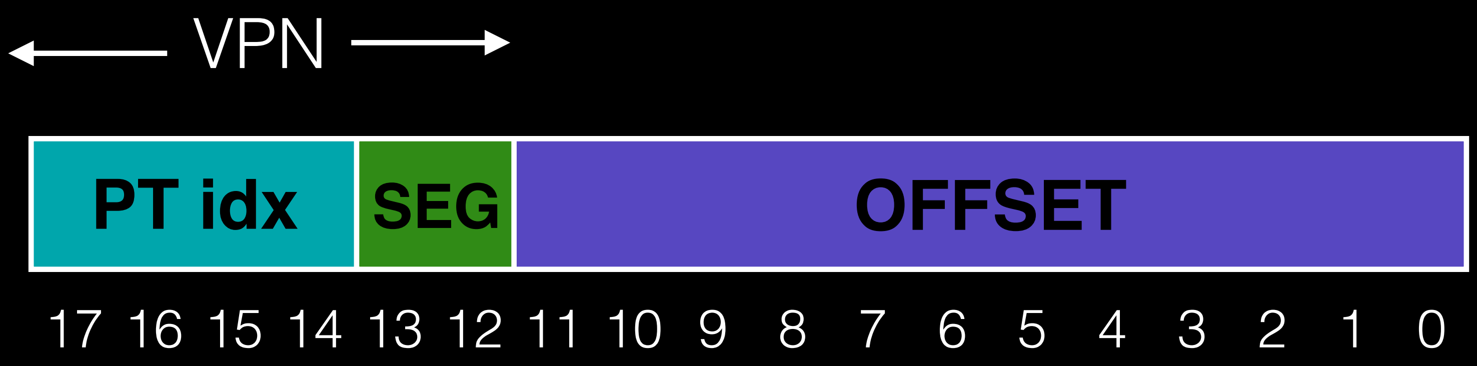
Now:



# Segmentation/Paging Hybrid

Idea: use different page tables for heap, stack, etc
- each PT can be a different size
- each PT has a base/bounds (where?)

Now:

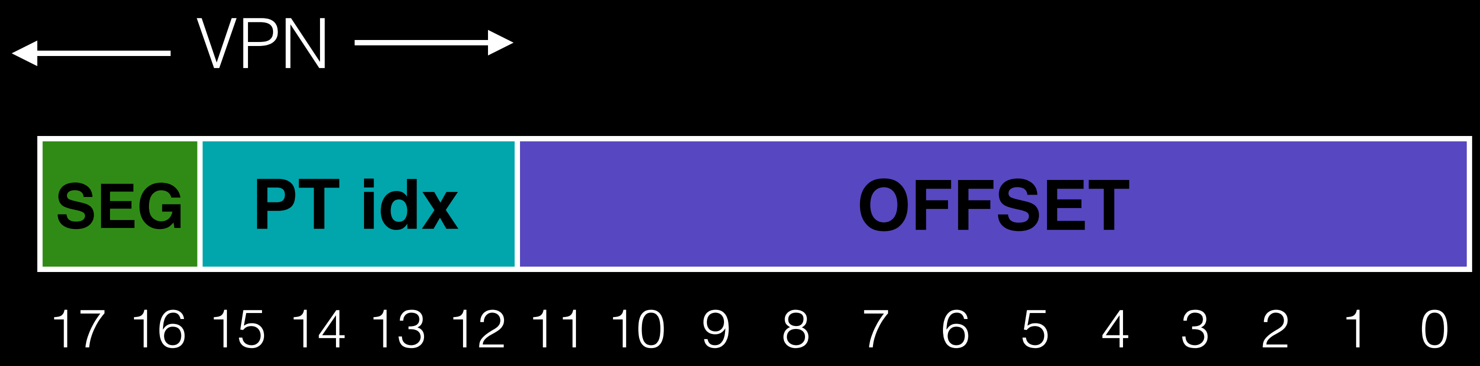


why not?

# Segmentation/Paging Hybrid

Idea: use different page tables for heap, stack, etc
- each PT can be a different size
- each PT has a base/bounds (where?)

Now:



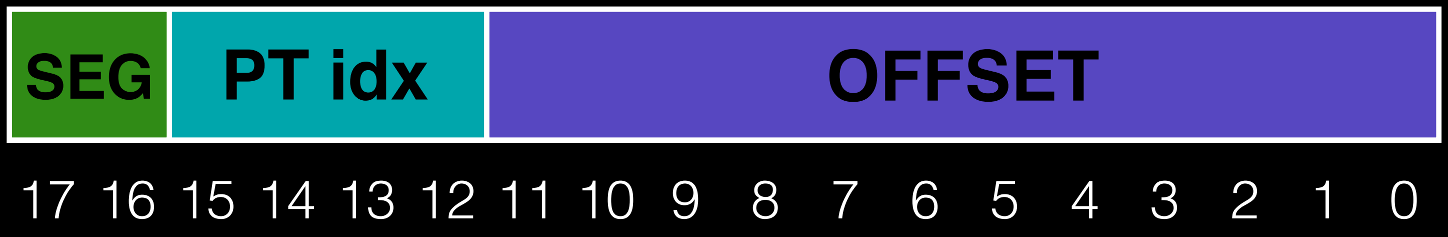
# Segmentation/Paging Hybrid

segment 00: code

| PFN | valid | prot |
|---|---|---|
| 0x10 | 1 | r-x |
| 0x15 | 1 | r-x |
| 0x12 | 1 | r-x |
| … | | |

segment 01: heap

| PFN | valid | prot |
|---|---|---|
| 0x22 | 1 | rw- |
| 0x02 | 1 | rw- |
| 0x04 | 1 | rw- |
| … | | |



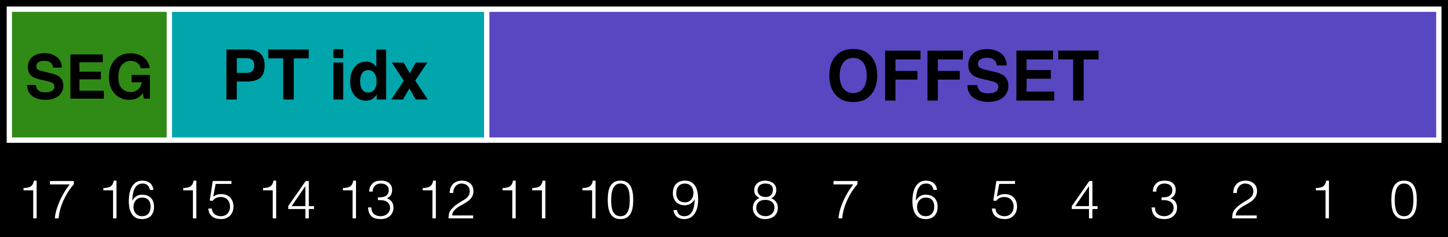
# Segmentation/Paging Hybrid

segment 00: code

| PFN | valid | prot |
|---|---|---|
| 0x10 | 1 | r-x |
| 0x15 | 1 | r-x |
| 0x12 | 1 | r-x |
| | … | |

segment 01: heap

| PFN | valid | prot |
|---|---|---|
| 0x22 | 1 | rw- |
| 0x02 | 1 | rw- |
| 0x04 | 1 | rw- |
| | … | |



Problem 5
(worksheet)

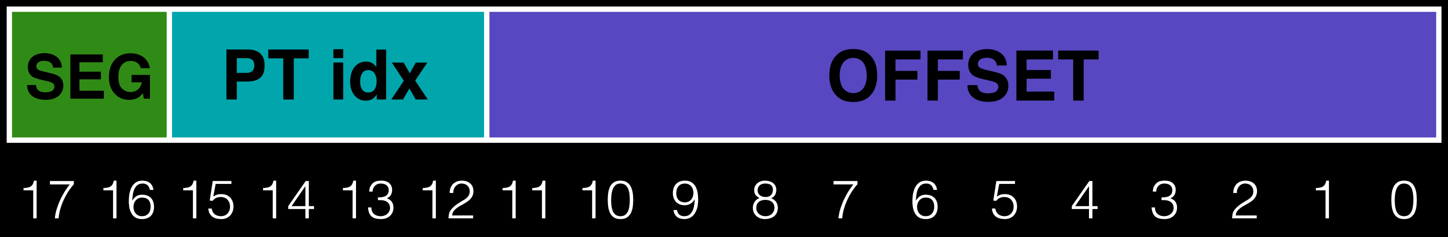
# Segmentation/Paging Hybrid

segment 00: code

| PFN | valid | prot |
|---|---|---|
| 0x10 | 1 | r-x |
| 0x15 | 1 | r-x |
| 0x12 | 1 | r-x |
| … | | |

segment 01: heap

| PFN | valid | prot |
|---|---|---|
| 0x22 | 1 | rw- |
| 0x02 | 1 | rw- |
| 0x04 | 1 | rw- |
| … | | |



What about the stack?
(OSTEP skips this)

## Approach 2

**Approach 2a**: hashtable

**Approach 2b**: segments over PTs

**Approach 2c**: PTs over PTs

**Approach 2d**: PTs over PTs over PTs

# Multi-Level Page Tables

Idea: break PT itself into pages

# Multi-Level Page Tables

Idea: break PT itself into pages
- a page directory refers to pieces
- only have pieces with >0 valid entries

## Multi-Level Page Tables

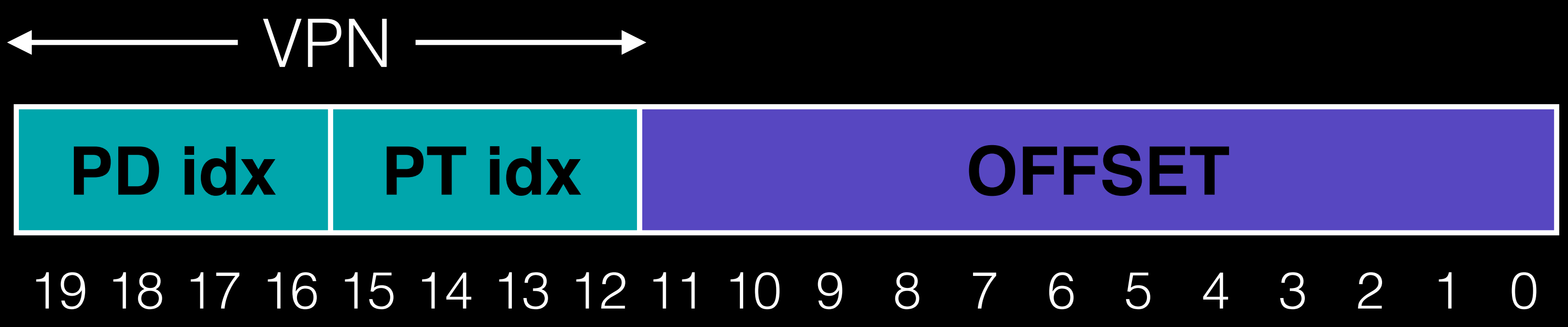
Idea: break PT itself into pages
 - a page directory refers to pieces
 - only have pieces with >0 valid entries

Used by x86.

# Multi-Level Page Tables

Idea: break PT itself into pages
- a page directory refers to pieces
- only have pieces with >0 valid entries

Used by x86.



**page directory**

| PFN | valid |
|---|---|
| 0x3 | 1 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| 0x92 | 1 |

**page of PT (@PFN:0x3)**

| PFN | valid |
|---|---|
| 0x10 | 1 |
| 0x23 | 1 |
| - | 0 |
| - | 0 |
| 0x80 | 1 |
| 0x59 | 1 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |

**page of PT (@PFN:0x92)**

| PFN | valid |
|---|---|
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| 0x55 | 1 |
| 0x45 | 1 |

Problem 6
(worksheet)

assume 20-bit virtual addrs

## Motivation

Why do we want big virtual address spaces?
- programming is easier
- applications need not worry (as much) about fragmentation

Paging goals:
- space efficiency (don't waste on invalid data)
- simplicity (no bookkeeping should require contiguous pages)

# Motivation

Why do we want big virtual address spaces?

- programming is easier
- applications need not worry (as much) about fragmentation

Paging goals:

- space efficiency (don’t waste on invalid data)
- **simplicity (no bookkeeping should require contiguous pages)**
  - page directories are too big!

# Approach 2

**Approach 2a**: hashtable

**Approach 2b**: segments over PTs

**Approach 2c**: PTs over PTs

**Approach 2d**: PTs over PTs over PTs

## >2 Levels

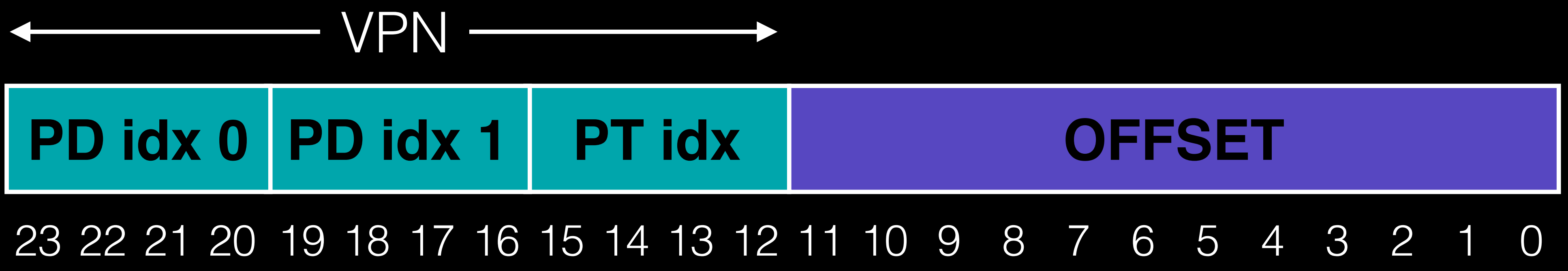
Problem: page directories may not fit in a page

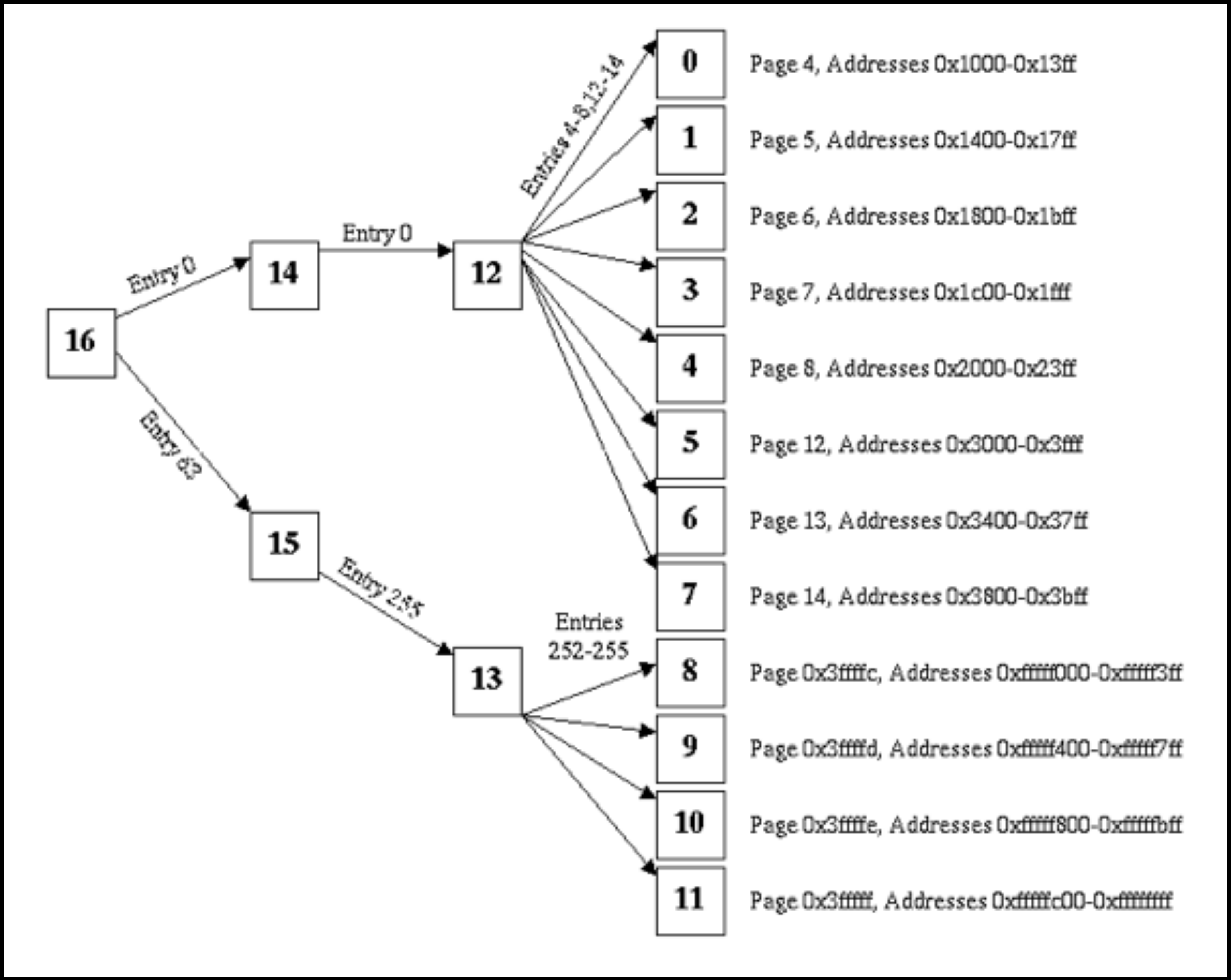
Solution: split page directories into pieces.
Use another page dir to refer to the page dir pieces.

# >2 Levels

Problem: page directories may not fit in a page

Solution: split page directories into pieces.
Use another page dir to refer to the page dir pieces.




16
Entry 0
14
Entry 0
12
Entries 4-8,12-14
Entry 63
15
Entry 255
13
Entries 252-255
0
Page 4, Addresses 0x1000-0x13ff
1
Page 5, Addresses 0x1400-0x17ff
2
Page 6, Addresses 0x1800-0x1bff
3
Page 7, Addresses 0x1c00-0x1fff
4
Page 8, Addresses 0x2000-0x23ff
5
Page 12, Addresses 0x3000-0x3fff
6
Page 13, Addresses 0x3400-0x37ff
7
Page 14, Addresses 0x3800-0x3bff
8
Page 0x3ffffc, Addresses 0xfffff000-0xfffff3ff
9
Page 0x3ffffd, Addresses 0xfffff400-0xfffff7ff
10
Page 0x3ffffe, Addresses 0xfffff800-0xfffffbff
11
Page 0x3fffff, Addresses 0xfffffc00-0xffffffff

http://web.eecs.utk.edu/~mbeck/classes/cs560/560/oldtests/t2/2003/Answers.html

## >2 Levels

Problem: page directories may not fit in a page

Solution: split page directories into pieces.
Use another page dir to refer to the page dir pieces.

# >2 Levels

Problem: page directories may not fit in a page

Solution: split page directories into pieces.
Use another page dir to refer to the page dir pieces.

How many levels do we need? (Problem 7)

## Approach 2

**Approach 2a**: hashtable

**Approach 2b**: segments over PTs

**Approach 2c**: PTs over PTs

**Approach 2d**: PTs over PTs over PTs

# What about TLBs?

Lookups in multiple levels more expensive.

How much does a miss cost? (problem 8)

Time/Space tradeoffs.

## Summary

Many PT options are possible.

Time/Space/Complexity tradeoffs.

OS traps on TLB misses would be ideal.

x86 walks multi-level PTs.

## Announcements

P2a due in 9 days!

Discussion tomorrow…

FB tech talk tonight.