last time

```
dividing memory into pages
     page numbers (0-based index of page) and page offsets (byte in page)
address space sizes
     pointer size v. number of bits actually used
     more physical addresses than installed memory
page tables entries
     lookup using virtual page number
     valid bit — is something there? (if no, exception)
     physical page number
     permission bits — what accesses to allow? (if no, exception)
```

allocate-on-demand

anonymous feedback (1)

"people in this class whine too much in the anonymous feedback and I think there is no problem with telling them to stop whining... and I'll start with myself... I need to stop whining in the anonymous feedback....."

"I think class time in managed well for content, and honestly others' complaints are better voiced offline, such as on Piazza."

anonymous feedback (2)

"Hi Professor, if it's not too much trouble, would you mind nudging the class about not talking super loudly while you're lecturing? I get a lot out of attending lecture for this class, but sometimes it's even difficult to focus or hear you over the people around me..."

anonymous feedback (2)

"I think that the quiz questions with only one right answer being worth 4 points is very steep, especially since there is only one correct one. If we are between two choices and choose the wrong one, our grade drops by 15 points no questions asked. If possible, I would like to suggest that in the comments we can provide a "backup answer" for half credit that can help justify and explain our thought process. Personally, I just think that it is a very steep penalty. Thank you for your consideration."

in some cases it'll make sense to have partial credit for some wrong answers

in some cases we can give partial credit based on reasoning in comments (when it shows understanding of concept the question is meant to test) but main mitigation in value of quiz questions is meant to be number of quizzes

anonymous feedback (3)

"I learn better by seeing and understanding examples. I'd like to see more of class time dedicated to livecoding, examples, and interactive questions. This helps me visualize concepts in the way that we'll see them applied. I typically learn the best in CS classes that are structured to have a short conceptual lesson in the beginning of class followed by livecoding and examples to help prepare me for labs and homework. So far, it's taken me a lot longer to complete our HW/labs because it is my first time applying the concepts we use."

somewhat intentional that HW/labs are meant to be practice applying concepts more than assessment/etc.

course design that doesn't do this probably needs students to prepare for lecture more

agree that I should have more interactive questions

livecoding isn't a teaching technique I'm as comfortable with, but...
...I don't think it's a good fit to for most of our rather non-algorithmic

anonymous feedback (4)

"Would you be able to share the median, mean, and standard deviation of the final from this class last semester, as well as the curve that was applied to everyone's grades at the end?"

approx. 25th/median/75th percentile on study materials page not a curve (since it's not based on relative performance of students)

	S 2023	F 2023
A+	96.9	96.5
Α	92.7	92.5
A-	89.6	89.7
B+	86.9	86.5
В	83.0	82.5
B-	80.0	79.5
C+	77.0	76.9
C	73.0	73.0
C-	70.0	69.0
D+	67.0	66.0
D	63.0	62.0
D-	60.0	59.0

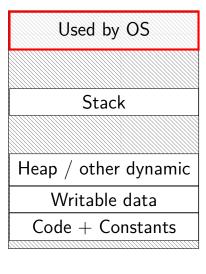
program memory

Used by OS		
Stack		
Heap / other dynamic		
Writable data		
Code + Constants		

0xffff Ffff Ffff Ffff
0xffff 8000 0000 0000
0x7f...

0x0000 0000 0040 0000

program memory



0xFFFF FFFF FFFF

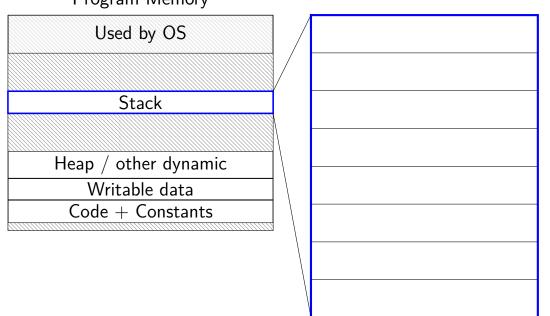
0xFFFF 8000 0000 0000

0x7F...

0x0000 0000 0040 0000

space on demand

Program Memory



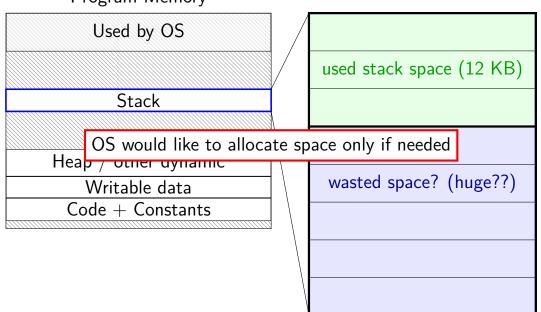
space on demand

Program Memory

Used by OS used stack space (12 KB) Stack Heap / other dynamic wasted space? (huge??) Writable data Code + Constants

space on demand

Program Memory



%rsp = 0x7FFFC000

```
// requires more stack space
A: pushq %rbx

B: movq 8(%rcx), %rbx
C: addq %rbx, %rax
...
```

VPN	valid?	pnysical page
VIIN	valiu:	page
•••	•••	•••
0x7FFFB	0	
0x7FFFC	1	0x200DF
0x7FFFD	1	0x12340
0x7FFFE	1	0x12347
0x7FFFF	1	0x12345
•••	•••	•••

المسامين عالما

%rsp = 0x7FFC000

```
// requires more stack space
A: pushq %rbx
page fault!
B: movq 8(%rcx), %rbx
C: addq %rbx, %rax
...
```

VPN	valid?	physical page
VIIN	valiu!	page
•••	•••	•••
0x7FFFB	0	
0x7FFFC	1	0x200DF
0x7FFFD	1	0x12340
0x7FFFE	1	0x12347
0x7FFFF	1	0x12345
•••	•••	•••

pushq triggers exception hardware says "accessing address 0x7FFBFF8" OS looks up what's should be there — "stack"

%rsp = 0x7FFC000

```
// requires more stack space
A: pushq %rbx restarted

B: movq 8(%rcx), %rbx
C: addq %rbx, %rax
...
```

VPN	valid?	physical page
VFIN	valiu!	page
•••	•••	•••
0x7FFFB	1	0x200D8
0x7FFFC	1	0x200DF
0x7FFFD	1	0x12340
0x7FFFE	1	0x12347
0x7FFFF	1	0x12345
•••	•••	•••

in exception handler, OS allocates more stack space OS updates the page table then returns to retry the instruction

note: the space doesn't have to be initially empty

only change: load from file, etc. instead of allocating empty page

loading program can be merely creating empty page table everything else can be handled in response to page faults no time/space spent loading/allocating unneeded space

mmap

```
Linux/Unix has a function to "map" a file to memory
int file = open("somefile.dat", O_RDWR);
    // data is region of memory that represents file
char *data = mmap(..., file, 0);
   // read byte 6 from somefile.dat
char seventh_char = data[6];
   // modifies byte 100 of somefile.dat
data[100] = 'x';
    // can continue to use 'data' like an array
```

Linux maps: list of maps

```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 48328831
                                                         /bin/cat
0060a000-0060b000 r-p 0000a000 08:01 48328831
                                                         /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 48328831
                                                         /bin/cat
01974000-01995000 rw-p 00000000 00:00 0
                                                         [heap]
7f60c718b000-7f60c7490000 r-p 00000000 08:01 77483660
                                                         /usr/lib/locale/locale—archive
7f60c7490000-7f60c764e000 r-xp 00000000 08:01 96659129
                                                         /lib/x86_64—linux—gnu/libc-2.1
7f60c764e000-7f60c784e000 ----p 001be000 08:01 96659129
                                                         /lib/x86_64—linux—gnu/libc-2.1
7f60c784e000-7f60c7852000 r-p 001be000 08:01 96659129
                                                         /lib/x86_64—linux—gnu/libc-2.1
7f60c7852000—7f60c7854000 rw—p 001c2000 08:01 96659129
                                                         /lib/x86 64—linux—gnu/libc-2.1
7f60c7854000-7f60c7859000 rw-p 00000000 00:00 0
7f60c7859000-7f60c787c000 r-xp 00000000 08:01 96659109
                                                         /lib/x86_64—linux—gnu/ld-2.19.s
7f60c7a39000-7f60c7a3b000 rw-p 00000000 00:00 0
7f60c7a7a000—7f60c7a7b000 rw—p 00000000 00:00 0
7f60c7a7b000-7f60c7a7c000 r-p 00022000 08:01 96659109
                                                         /lib/x86_64—linux—gnu/ld-2.19.s
7f60c7a7c000-7f60c7a7d000 rw-p 00023000 08:01 96659109
                                                         /lib/x86_64—linux—gnu/ld-2.19.s
7f60c7a7d000—7f60c7a7e000 rw—p 00000000 00:00 0
7ffc5d2b2000-7ffc5d2d3000 rw-p 00000000 00:00 0
                                                         [stack]
7ffc5d3b0000-7ffc5d3b3000 r---p 00000000 00:00 0
                                                         [vvar]
7ffc5d3b3000-7ffc5d3b5000 r-xp 00000000 00:00 0
                                                          vdsol
fffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0
                                                         [vsyscall]
```

Linux maps: list of maps

```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 48328831
                                                        /bin/cat
0060a000-0060b000 r-p 0000a000 08:01 48328831
                                                        /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 48328831
                                                        /bin/cat
01974000-01995000 rw-p 00000000 00:00 0
                                                        [heap]
7f60c718b000_7f60c7490000
                                                        <u>usr/lib/locale/lo</u>cale—archive
7f60c74900 OS tracks list of struct vm_area_struct with:
                                                                         gnu/libc-2.1
7f60c764e0
                                                                         gnu/libc-2.1
          (shown in this output):
7f60c784e0
                                                                         gnu/libc-2.1
7f60c78520
                                                                         gnu/libc-2.1
             virtual address start, end
7f60c78540
                                                                         gnu/ld-2.19.s
7f60c78590
             permissions
7f60c7a390
7f60c7a7a0
             offset in backing file (if any)
7f60c7a7b0
                                                                         gnu/ld-2.19.s
7f60c7a7c0
             pointer to backing file (if any)
                                                                         gnu/ld-2.19.s
7f60c7a7d0
7ffc5d2b20
7ffc5d3b00
           (not shown):
7ffc5d3b30
ffffffffff
             info about sharing of non-file data
```

do we really need a complete copy?

	bash
	Used by OS
	Stack
	eap / other dynamic
	. ,
	Writable data

ما م ما

nev	w copy of bash
	Used by OS
	Stack
Неар	/ other dynamic
V	Vritable data
Coc	le + Constants

do we really need a complete copy?

bash	new copy of bash	
Used by OS	Used by OS	
Stack	Stack	
Heap / other dynamic	Heap / other dynamic	
Writable data	Writable data	
Code + Constants	Code + Constants	

shared as read-only

do we really need a complete copy?

bash	new copy of bash	
Used by OS	Used by OS	
Stack	Stack	
Heap / other dynamic	Heap $/$ other dynamic	
Writable data	Writable data	
Code + Constants $Code + Constants$ $Code + Constants$		

trick for extra sharing

```
sharing writeable data is fine — until either process modifies it example: default value of global variables might typically not change (or OS might have preloaded executable's data anyways)
```

can we detect modifications?

trick for extra sharing

```
sharing writeable data is fine — until either process modifies it example: default value of global variables might typically not change (or OS might have preloaded executable's data anyways)
```

can we detect modifications?

trick: tell CPU (via page table) shared part is read-only processor will trigger a fault when it's written

VPN

valid? write?

•••

0x00601 0x00602 0x00603 0x00604 0x00605

		Page
•••	•••	•••
1		0x12345
1	1	0x12347
1	1	0x12340
1	1	0x200DF
1	1	0x200AF
•••	•••	•••

VPN
•••
0x00601
0x00602
0x00603
0x00604
0x00605
•••

valid? write? page		
•••	•••	•••
1	0	0x12345
1	0	0x12347
1	0	0x12340
1	0	0x200DF
1	0	0x200AF
•••	•••	•••

• • • • •
•••
0x00601
0x00602
0x00603
0x00604
0x00605

VPN

valid?	write?	physical page

•••	•••	•••
1	0	0x12345
1	0	0x12347
1	0	0x12340
1	0	0x200DF
1	0	0x200AF
•••	•••	•••

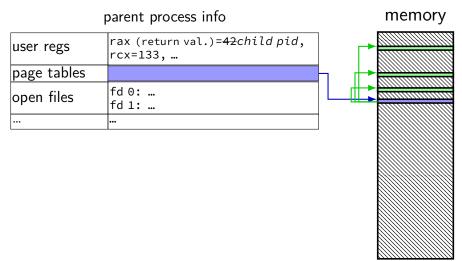
copy operation actually duplicates page table both processes share all physical pages but marks pages in both copies as read-only

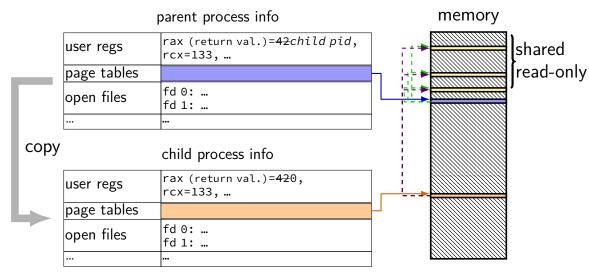
VPN	valid? write? page			VPN	valid? write? page			
VEIN			page		VEIN	valiu	page	
•••	•••	•••	•••		•••	•••	•••	•••
0x00601	1	0	0x12345		0x00601	1	0	0x1234
0x00602	1	0	0x12347		0x00602	1	0	0x1234
0x00603	1	0	0x12340		0x00603	1	0	0x1234
0x00604	1	0	0x200DF		0x00604	1	0	0x200[
0x00605	1	0	0x200AF		0x00605	1	0	0x200A
•••	•••	•••	•••		•••	•••	•••	•••

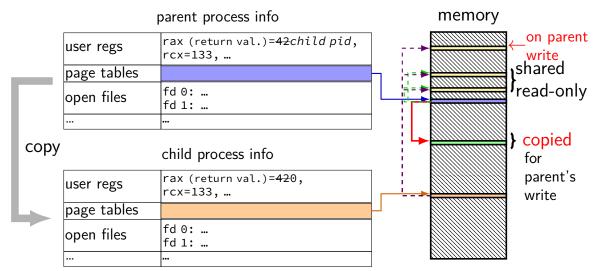
when either process tries to write read-only page triggers a fault — OS actually copies the page

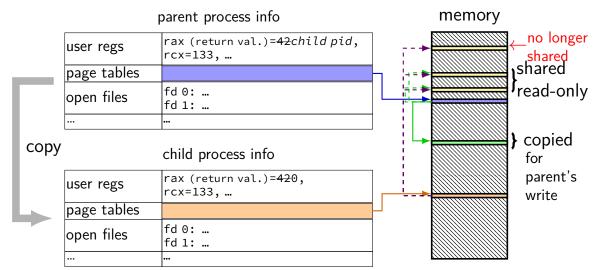
VPN	valid? write? page		physical	VPN	valid? write? page			
viii valid: write:		page	VIIV	valiu:	page			
•••	•••	•••	•••	•••	•••	•••	•••	
0x00601	1	0	0x12345	0x00601	1	0	0x12345	
0x00602	1	0	0x12347	0x00602	1	0	0x12347	
0x00603	1	0	0x12340	0x00603	1	0	0x12340	
0x00604	1	0	0x200DF	<u>0x00604</u>	1	0	0x200DF	
0x00605	1	0	0x200AF	0x00605	1	1	0x300FD	
•••	•••	•••	•••	•••	•••	•••	•••	

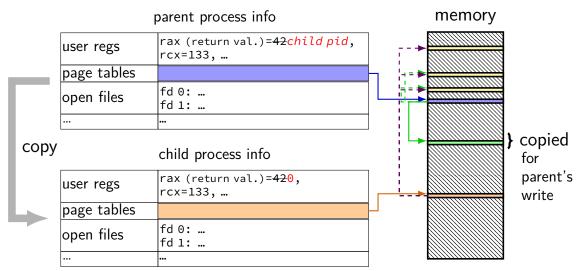
after allocating a copy, OS reruns the write instruction











page tricks generally

deliberately make program trigger page/protection fault

but don't assume page/protection fault is an error

have seperate data structures represent logically allocated memory e.g. "addresses 0x7FFF8000 to 0x7FFFFFFF are the stack"

page table is for the hardware and not the OS

example page table tricks

allocating space on demand

loading code/data from files on disk on demand

copy-on-write

saving data temporarily to disk, reloading to memory on demand "swapping"

detecting whether memory was read/written recently stopping in a debugger when a variable is modified sharing memory between programs on two different machines

example page table tricks

allocating space on demand

loading code/data from files on disk on demand

copy-on-write

saving data temporarily to disk, reloading to memory on demand "swapping"

detecting whether memory was read/written recently stopping in a debugger when a variable is modified sharing memory between programs on two different machines

example page table tricks

allocating space on demand

loading code/data from files on disk on demand

copy-on-write

saving data temporarily to disk, reloading to memory on demand "swapping"

detecting whether memory was read/written recently

stopping in a debugger when a variable is modified

sharing memory between programs on two different machines

example page table tricks

allocating space on demand

loading code/data from files on disk on demand

copy-on-write

saving data temporarily to disk, reloading to memory on demand "swapping"

detecting whether memory was read/written recently

stopping in a debugger when a variable is modified

sharing memory between programs on two different machines

example page table tricks

allocating space on demand

loading code/data from files on disk on demand

copy-on-write

saving data temporarily to disk, reloading to memory on demand "swapping"

detecting whether memory was read/written recently

stopping in a debugger when a variable is modified

sharing memory between programs on two different machines

hardware help for page table tricks

information about the address causing the fault
e.g. special register with memory address accessed
harder alternative: OS disassembles instruction, look at registers

(by default) rerun faulting instruction when returning from exception

precise exceptions: no side effects from faulting instruction or after e.g. pushq that caused did not change %rsp before fault e.g. can't notice if instructions were executed in parallel

exercise setup

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

page table

virtual	valid?	physical
page #	valiu!	page #
00	1	010
01	1	111
10	0	000
11	1	000

physical bytes addresses 0x00-3 00 11 22 33
0x04-744 55 66 77
0x08-B88 99 AA BB
0x0C-FCC DD EE FF
0x10-3 1A 2A 3A 4A
0x14-7 1B 2B 3B 4B
0x18-B1C2C3C4C
0x1C-F1C 2C 3C 4C

physical bytes addresses					
0x20-3	D0	D1	D2	D3	
0x24-7	D4	D5	D6	D7	
0x28-B					
0x2C-F	CD	DE	EF	F0	
0x30-3					
0x34-7	СВ	0B	СВ	0B	
0x38-B					
0x3C-F	EC	0C	EC	0C	

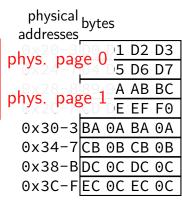
exercise setup

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

page table

virtual	valid?	physical
page #	valiu!	page #
00	1	010
01	1	111
10	0	000
11	1	000

physical bytes addresses					
0x00-3	00 11 22 33				
0x04-7	44 55 66 77				
0x08-B	88 99 AA BB				
	CC DD EE FF				
0x10-3	1A 2A 3A 4A				
	1B 2B 3B 4B				
0x18-B	1C 2C 3C 4C				
0x1C-F	1C 2C 3C 4C				



5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

```
(virtual addresses) 0x18 = ???; 0x03 = ???; 0x0A = ???; 0x13 = ???
```

page table

```
page # valid? ___
            010
    001
    01
             111
            000
    10
    11
            000
```

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical bytes addresses_					
0x20-3	D0	D1	D2	D3	
0x24-7					
0x28-B	89	9A	ΑB	ВС	
0x2C-F					
0x30-3	ВА	0A	ВА	0A	
0x34-7					
0x38-B	DC	0C	DC	0C	
0x3C-F	EC	0C	EC	0C	

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

page table

```
page # valid? _
    00
            010
    01
            111
            000
    10
            000
    11
```

physical addresses	byt	es		
0x00-3	00	11	22	33
0x04-7	44	55	66	77
0x08-B	88	99	AA	ВВ
0x0C-F				
0x10-3	1A	2A	ЗА	4A
0x14-7	1B	2B	3B	4B
0x18-B	1C	2C	3C	4C
0x1C-F	1C	2C	3C	4C

physical bytes addresses 0x20-3 D0 D1 D2 D3 0x24-7 D4 D5 D6 D7 0x28-B|89 9A AB BC 0x2C-FCD DE EF F0 0x30-3|BA 0A BA 0A 0x34-7 CB 0B CB 0B 0x38-BDC 0C DC 0C 0x3C-FEC 0C EC 0C

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

```
(virtual addresses) 0x18 = ; 0x03 = ; 0x0A = ???; 0x13 = ??? page table
```

physical bytes addresses					
0x00-3	00	11	22	33	
0x04-7	44	55	66	77	
0x08-B	88	99	AA	ВВ	
0x0C-F	CC	DD	EE	FF	
0x10-3	1A	2A	ЗА	4A	
0x14-7	1B	2B	3B	4B	
0x18-B	1C	2C	3C	4C	
0x1C-F	1C.	2C	3C	4C	

physical bytes addresses 0x20-3 D0 D1 D2 D3 0x24-7 D4 D5 D6 D7 0x28-B|89 9A AB BC 0x2C-FCD DE EF F0 0x30-3|BA 0A BA 0A 0x34-7 CB 0B CB 0B 0x38-BDC 0C DC 0C 0x3C-F|EC 0C EC 0C

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

```
(virtual addresses) 0x18 = ; 0x03 = ; 0x0A = ; 0x13 = ??? page table
```

physical bytes addresses					
0x00-3	00	11	22	33	
0x04-7	44	55	66	77	
0x08-B	88	99	AA	ВВ	
0x0C-F	CC	DD	EE	FF	
0x10-3	1A	2A	ЗА	4A	
0x14-7	1B	2B	3B	4B	
0x18-B	1C	2C	3C	4C	
0×1C-E	10	20	30	10	

physical addresses	byt	es		
0x20-3	D0	D1		
0x24-7	D4	D5	D6	D7
0x28-B	89	9A	ΑB	ВС
0x2C-F	CD	DE	EF	F0
0x30-3				
0x34-7				
0x38-B				
0x3C-F	EC	0C	EC	0C

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

```
(virtual addresses) 0x18 = ; 0x03 = ; 0x0A = ; 0x13 = page table
```

virtual page # valid? physical page # 00 1 010 011 111 10 0 000 11 1 1 1000

physical addresses	bvt	es		
0x00-3				
0x04-7	44	55	66	77
0x08-B	88	99	AA	ВВ
0x0C-F	CC	DD	EE	FF
0x10-3	1A	2A	ЗА	4A
0x14-7	1B	2B	3B	4B
0x18-B	1C	2C	3C	4C
0x1C-F	1C	2C	3C	4C

physical bytes addresses 0x20-3 D0 D1 D2 D3 0x24-7 D4 D5 D6 D7 0x28-B|89 9A AB BC 0x2C-FCD DE EF F0 0x30-3|BA 0A BA 0A 0x34-7 CB 0B CB 0B 0x38-BDC 0C DC 0C 0x3C-FEC 0C EC 0C

motivation: page tables are big

real systemns: huge number of virtual pages not enough space to store page table in the processor core

trick one: store in memory processor core just has pointer to place in memory

trick two: avoid storing most invalid entries tree-like data structure omit nodes of tree that would only have invalid leafs

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15) physical page # (bits 4–14) other bits and/or unused (bit 0-3)

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15) physical page # (bits 4–14) other bits and/or unused (bit 0-3)

page table base register

0x00010000

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor) valid (bit $\overline{15}$) physical page # (bits 4–14) other bits and/or unused (bit 0-3) physical memory page table base register addresses bytes 0x00000000-1 00000000 00000000 0x00010000 0x00010000-1 00000000 00000000 $0 \times 00010002 - 3 10100010$ $0 \times 00010004 - 5\Gamma$ 0x00010006-7 10110000 0x000101FE-F 10001110 0x00010200-1 10100010 001

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor) valid (bit 15) physical page # (bits 4–14) other bits and/or unused (bit 0-3) physical memory page table base register addresses bytes 0x00000000-1 00000000 00000000 0x00010000 $0 \times 00010002 - 3 10100010$ $0 \times 00010004 - 5\Gamma$ 10000010 0x00010006-7 10110000 0x000101FE-F 10001110 0x00010200-1 10100010 0011101

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor) valid (bit 15) physical page # (bits 4–14) other bits and/or unused (bit 0-3) physical memory page table base register addresses bytes 0x00000000-1 00000000 00000000 0x00010000 0x00010000-1 00000000 $0 \times 00010002 - 3 10100010$ $0 \times 00010004 - 5\Gamma$ 0x00010006-7 0x000101FE-F 10001110 0x00010200-1 10100010 0011101

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor) valid (bit $\overline{15}$) physical page # (bits 4–14) other bits and/or unused (bit 0-3) physical memory page table base register addresses bytes 0x00000000-1 00000000 00000000 0x00010000 0x00010000-1 00000000 00000000 $0 \times 00010002 - 3 10100010$ $0 \times 00010004 - 5\Gamma$ 10000010 0x00010006-7 10110000 0x000101FE-F 10001110 0x00010200-1 10100010 0011101

where can processor store megabytes of page tables? in memory

valid (bit 15) physical page # (bits 4–14) other bits and/or unused (bit 0-3) page table physical memory base register addresses bytes 0x0000000-1 00000000 00000000 0x00010000 0x00010000-1 00000000 00000000 $0 \times 00010002 - 3$ page table (logically) 0x00010004-5 10000010 0x00010006-7 10110000 00110000 virtual page # valid? physical page # 0000 0000 0000 0000 0x000101FE-F 10001110 0000 0001 0x00010200-1 10100010 00111010 0000 0010 0000 0011 0000 0011 1111 1111 00 1110 1000

page table entry layout (chosen by processor)

where can processor store megabytes of page tables? in memory

valid (bit 15) physical page # (bits 4–14) other bits and/or unused (bit 0-3) page table physical memory base register addresses bytes 0x0000000-1 00000000 00000000 0x00010000 0x00010000-1 00000000 00000000 0x00010002-3 page table (logically) 0x00010004-5 0000010 0x00010006-7 0110000 00110000 virtual page # valid? physical page # 0000 0000 0000 0000 0x000101FE-F 10001110 0000 0001 0x00010200-1 10100010 00111010 0000 0010 0000 0011 0000 0011 1111 1111 00 1110 1000

page table entry layout (chosen by processor)

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor) valid (bit 15) physical page # (bits 4–14) other bits and/or unused (bit 0-3) page table physical memory base register addresses bytes 0x0000000-1 00000000 00000000 0x00010000 0x00010000-1 00000000 0x00010002-3 page table (logically) 0x00010004-5 0x00010006-7 virtual page # valid? physical page # 0000 0000 0x000101FE-F 10001 0000 0001 0x00010200-1 10100010 001 0000 0010 0000 0011 0000 001 1111 1111 1110 1000

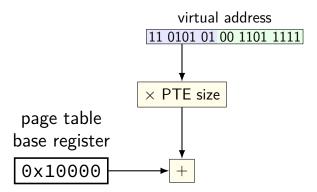
where can processor store megabytes of page tables? in memory

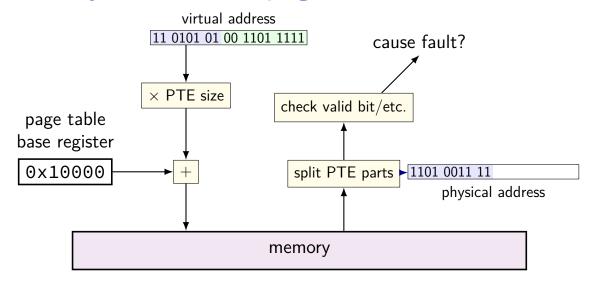
valid (bit 15) physical page # (bits 4–14) other bits and/or unused (bit 0-3) page table physical memory base register addresses bytes 0x0000000-1 00000000 00000000 0x00010000 0x00010000-1 00000000 000000000 $0 \times 00010002 - 3$ page table (logically) 0x00010004-5 10000010 0x00010006-7 10110000 001 virtual page # valid? physical page # 0000 0000 0000 0000 0x000101FE-F 10001110 0000 0001 0x00010200-1 10100010 00111010 0000 0010 0000 0011 0000 0011 1111 1111 00 1110 1000

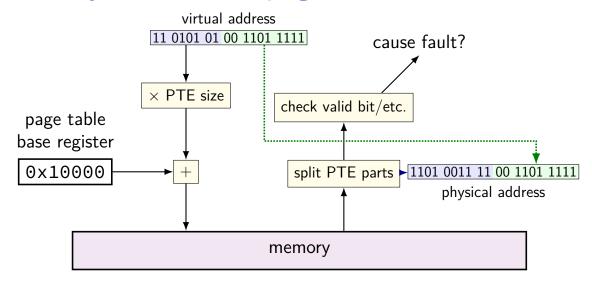
page table entry layout (chosen by processor)

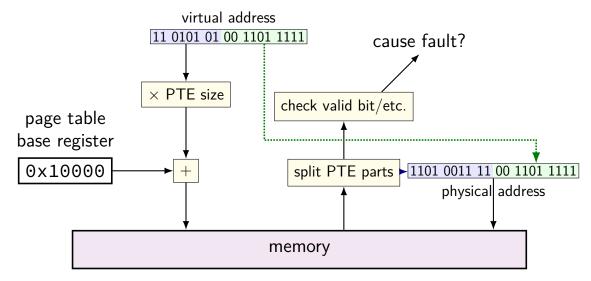
virtual address

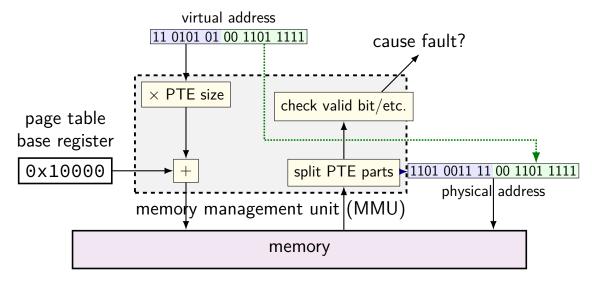
11 0101 01 00 1101 1111

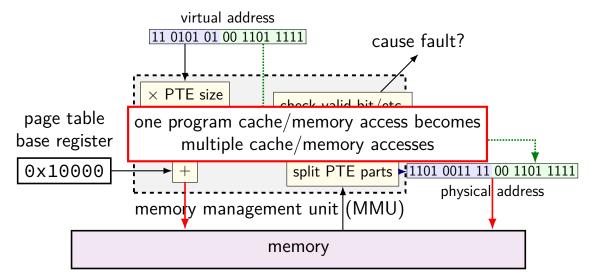


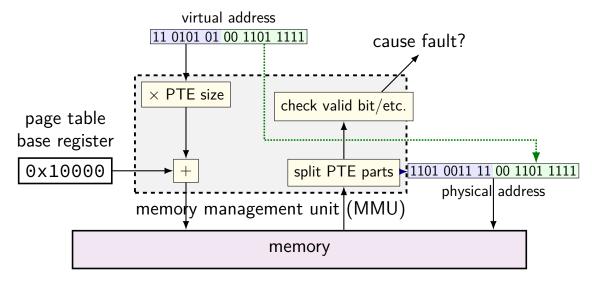












6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other; page table base register 0x20; translate virtual address 0x31

physical addresses	byt	es			physic address	al es	byt	es		
0x00-3			22	33	0x20-	-3	D0	D1	D2	D3
0x04-7	44	55	66	77	0x24-	-7	E4	E5	F6	07
0x08-B	88	99	AΑ	ВВ	0x28-	-B	89	9Α	ΑB	ВС
0x0C-F	CC	DD	EE	FF	0x2C-	-F	CD	DE	EF	F0
0x10-3	1A	2A	ЗА	4A	0x30-	-3	ВА	0Α	ВА	0A
0x14-7	1В	2B	3B	4B	0x34-	-7	СВ	0B	СВ	0B
0x18-B	1C	2C	3C	4C	0x38-	-B	DC	0C	DC	0C
0x1C-F	1C	2C	3C	4C	0x3C-	- F	EC	0C	EC	0C

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other;

```
physical bytes
                       physical bytes
                                           0x31 = 11 0001
                      addresses
addresses
                                           PTE addr:
0x00-3|00 11 22 33
                      0x20-3|D0 D1 D2 D3
                                           0x20 + 110 \times 1 = 0x26
0x04-7|44 55 66 77
                      0x24-7|E4 E5 F6 07
0x08-B|88 99 AA BB
                      0x28-B|89 9A AB BC
                                          PTE value:
                      0x2C-FCD DE EF F0
0x0C-FICC DD EE FF
                                           0xF6 = 1111 0110
0x10-3|1A 2A 3A 4A
                      0x30-3|BA 0A BA 0A
                                           PPN 111, valid 1
0x14-7|1B 2B 3B 4B
                      0x34-7|CB 0B CB 0B
                                           M[111 \ 001] = M[0x39]
                      0x38-BDC 0C DC 0C
0x18-B|1C 2C 3C 4C
                                           \rightarrow 0x0C
0x1C-F|1C 2C 3C 4C
                      0x3C-FEC 0C EC 0C
```

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other;

```
physical bytes
                       physical bytes
                                           0x31 = 11 0001
                      addresses
addresses
                                           PTE addr:
0x00-3|00 11 22 33
                      0x20-3|D0 D1 D2 D3
                                           0x20 + 110 \times 1 = 0x26
0x04-7|44 55 66 77
                      0x24-7|E4 E5 F6 07
0x08-B|88 99 AA BB
                      0x28-B|89 9A AB BC
                                          PTE value:
                      0x2C-FCD DE EF F0
0x0C-FICC DD EE FF
                                           0xF6 = 1111 0110
0x10-3|1A 2A 3A 4A
                      0x30-3|BA 0A BA 0A
                                           PPN 111, valid 1
0x14-7|1B 2B 3B 4B
                      0x34-7|CB 0B CB 0B
                                           M[111 \ 001] = M[0x39]
                      0x38-BDC 0C DC 0C
0x18-B|1C 2C 3C 4C
                                           \rightarrow 0x0C
0x1C-F|1C 2C 3C 4C
                      0x3C-FEC 0C EC 0C
```

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other;

```
physical bytes
                       physical bytes
                                           0x31 = 11 \ 0001
                      addresses
addresses
                                           PTE addr:
0x00-3|00 11 22 33
                      0x20-3|D0 D1 D2 D3
                                           0x20 + 110 \times 1 = 0x26
0x04-7|44 55 66 77
                      0x24-7|E4 E5 F6 07
0x08-B|88 99 AA BB
                      0x28-B|89 9A AB BC
                                          PTE value:
                      0x2C-FCD DE EF F0
0x0C-FICC DD EE FF
                                           0xF6 = 1111 0110
0x10-3|1A 2A 3A 4A
                      0x30-3|BA 0A BA 0A
                                           PPN 111, valid 1
0x14-7|1B 2B 3B 4B
                      0x34-7|CB 0B CB 0B
                                           M[111 \ 001] = M[0x39]
                      0x38-BDC 0C DC 0C
0x18-B|1C 2C 3C 4C
                                           \rightarrow 0x0C
0x1C-F|1C 2C 3C 4C
                      0x3C-FEC 0C EC 0C
```

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other;

```
physical bytes
                       physical bytes
                                           0x31 = 11 0001
                      addresses
addresses
                                           PTE addr:
0x00-3|00 11 22 33
                      0x20-3|D0 D1 D2 D3
                                           0x20 + 110 \times 1 = 0x26
0x04-7|44 55 66 77
                      0x24-7|E4 E5 F6 07
0x08-B|88 99 AA BB
                      0x28-B|89 9A AB BC
                                          PTE value:
                      0x2C-FCD DE EF F0
0x0C-FICC DD EE FF
                                           0xF6 = 1111 0110
0x10-3|1A 2A 3A 4A
                      0x30-3|BA 0A BA 0A
                                           PPN 111, valid 1
0x14-7|1B 2B 3B 4B
                      0x34-7|CB 0B CB 0B
                                           M[111 \ 001] = M[0x39]
                      0x38-BDC 0C DC 0C
0x18-B|1C 2C 3C 4C
                                           \rightarrow 0x0C
0x1C-F|1C 2C 3C 4C
                      0x3C-F|EC 0C EC 0C
```

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other page table base register 0x20; translate virtual address 0x12

physical addresses	byte	ytes				phy addr	/sical esses	bytes				
0x00-3			22	33		0x2	0-3	Α0	E2	D1	F3	
0x04-7	44	55	66	77		0x2	4-7	E4	E5	F6	07	
0x08-B	88	99	AΑ	ВВ		0x2	8-B	89	9A	ΑB	ВС	
0x0C-F	CC	DD	EE	FF		0x2	C-F	CD	DE	EF	F0	
0x10-3	1A	2A	ЗА	4A		0x3	80-3	ВА	0Α	ВА	0A	
0x14-7	1B	2B	3B	4B		0x3	4-7	СВ	0B	СВ	0B	
0x18-B	1C	2C	3C	4C		0x3	8-B	DC	0C	DC	0C	
0x1C-F	1C	2C	3C	4C		0x3	C-F	EC	0C	EC	0C	

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other page table base register 0x20; translate virtual address 0x12

physical bytes addresses 0x00-3|00 11 22 33 0x04-7|44 55 66 77 0x08-B|88 99 AA BB 0x0C-FICC DD EE FF 0x10-3|1A 2A 3A 4A 0x14-7|1B 2B 3B 4B 0x18-B|1C 2C 3C 4C 0x1C-F|1C 2C 3C 4C

```
physical bytes
addresses
0x20-3|A0 E2 D1 F3
0x24-7|E4 E5 F6 07
0x28-B|89 9A AB BC
0x2C-FCD DE EF F0
0x30-3|BA 0A BA 0A
0x34-7|CB 0B CB 0B
0x38-BDC 0C DC 0C
                     \rightarrow 0xBA
0x3C-FEC 0C EC 0C
```

0x12 = 01 0010 $PTE \ addr$: $0x20 + 2 \times 1 = 0x22$ $PTE \ value$: 0xD1 = 1101 0001 $PPN 110, \ valid 1$ M[110 001] = M[0x32] $\rightarrow 0xBA$

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other page table base register 0x20; translate virtual address 0x12

physical bytes addresses 0x00-3|00 11 22 33 0x04-7|44 55 66 77 0x08-B|88 99 AA BB 0x0C-FICC DD EE FF 0x10-3|1A 2A 3A 4A 0x14-7|1B 2B 3B 4B 0x18-B|1C 2C 3C 4C 0x1C-F|1C 2C 3C 4C

physical bytes $0 \times 12 = 01 \ 0.010$ addresses PTE addr: 0x20-3|A0 E2 D1 F3 $0x20 + 2 \times 1 = 0x22$ 0x24-7|E4 E5 F6 07 0x28-B|89 9A AB BC PTE value: 0x2C-FCD DE EF F0 0xD1 = 1101 00010x30-3|BA 0A BA 0A PPN 110, valid 1 0x34-7|CB 0B CB 0B $M[110 \ 001] = M[0x32]$ 0x38-BDC 0C DC 0C \rightarrow 0xBA 0x3C-FEC 0C EC 0C

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other page table base register 0x20; translate virtual address 0x12

```
physical bytes
                       physical bytes
                                            0 \times 12 = 01 \ 0010
                      addresses
addresses
                                            PTE addr:
0x00-3|00 11 22 33
                       0x20-3|A0 E2 D1 F3
                                            0x20 + 2 \times 1 = 0x22
0x04-7|44 55 66 77
                       0x24-7|E4 E5 F6 07
0x08-B|88 99 AA BB
                       0x28-B|89 9A AB BC
                                           PTE value:
                       0x2C-FCD DE EF F0
0x0C-FICC DD EE FF
                                            0xD1 = 1101 0001
0x10-3|1A 2A 3A 4A
                       0x30-3|BA 0A BA 0A
                                            PPN 110, valid 1
0x14-7|1B 2B 3B 4B
                       0x34-7|CB 0B CB 0B
                                            M[110 \ 001] = M[0x32]
                       0x38-BDC 0C DC 0C
0x18-B|1C 2C 3C 4C
                                            \rightarrow 0xBA
0x1C-F|1C 2C 3C 4C
                       0x3C-FEC 0C EC 0C
```

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other page table base register 0x20; translate virtual address 0x12

physical addresses	bvt	es			
					, a
0x00-3] (
0x04-7	44	55	66	77] (
0x08-B	88	99	AΑ	ВВ] (
0x0C-F	CC	DD	EE	FF] (
0x10-3	1A	2A	3A	4A	(
0x14-7	1В	2B	3B	4B	(
0x18-B	1C	2C	3C	4C	(
0x1C-F	1C	2C	3C	4C] (

```
physical bytes
                    0x12 = 01 0010
addresses
                    PTE addr:
0x20-3 A0 E2 D1 F3
                    0x20 + 2 \times 1 = 0x22
0x24-7 E4 E5 F6 07
0x28-Bl89 9A AB BC
                   PTE value:
0x2C-FCD DE EF F0
                    0xD1 = 1101 0001
0x30-3|BA 0A BA 0A
                    PPN 110, valid 1
0x34-7 CB 0B CB 0B
                    M[110 \ 001] = M[0x32]
0x38-BDC 0C DC 0C
                    \rightarrow 0xBA
0x3C-F|EC 0C EC 0C
```

pagetable assignment

```
pagetable assignment
```

simulate page tables (on top of normal program memory) alternately: implement another layer of page tables on top of the existing system's

in assignment:

virtual address \sim arguments to your functions

physical address \sim your program addresses (normal pointers)

pagetable assignment API

```
/* configuration parameters */
#define POBITS ...
#define LEVELS /* later /
size_t ptbr; // page table base register
    // points to page table (array of page table entries)
// lookup "virtual" address 'va' in page table ptbr points to
// return (void*) (~0L) if invalid
void *translate(size t va);
// make it so 'va' is valid, allocating one page for its data
// if it isn't already
void page_allocate(size_t va)
```

translate()

with POBITS=12, LEVELS=1:

ptbr = GetPointerToTable(

٧r	ıv vand	a i priysicai	
0	0		
1	1	0×9999	١,
2	0	_	7
3	1	0x3333	

VDN valid2 physical

```
\begin{array}{l} translate(0x0FFF) == (void^*) ~0L \\ translate(0x1000) == (void^*) ~0x9999000 \\ translate(0x1001) == (void^*) ~0x9999001 \\ translate(0x2000) == (void^*) ~0L \\ translate(0x2001) == (void^*) ~0L \\ translate(0x3000) == (void^*) ~0x3333000 \\ \end{array}
```

translate()

with POBITS=12, LEVELS=1:

ptbr = GetPointerToTable(

VPN Valid! physical				
0	0			
1	1	0×9999	١	
2	0)	
3	1	0x3333		

1/DM - 1:12 mby/size1

```
\begin{array}{l} translate(0x0\text{FFF}) == (void^*) ~0L \\ translate(0x1000) == (void^*) ~0x9999000 \\ translate(0x1001) == (void^*) ~0x9999001 \\ translate(0x2000) == (void^*) ~0L \\ translate(0x2001) == (void^*) ~0L \\ translate(0x3000) == (void^*) ~0x3333000 \\ \end{array}
```

page_allocate()

```
with POBITS=12, LEVELS=1:  ptbr == 0 \\ page\_allocate(0 \times 1000) \ \textit{or} \ page\_allocate(0 \times 1001) \ \textit{or} \ ... \\
```

page_allocate()

```
with POBITS=12, LEVELS=1: 
 ptbr == 0 page\_allocate(0x1000) \ or \ page\_allocate(0x1001) \ or \ ...
```

 $\mathsf{ptbr}\ \mathit{now} == \mathsf{GetPointerToTable}($

VI IV	vanu:	Pilysi	cai	
0	0			
1	1	(new))	١
2	0)
3	1			

VPN valid2 physical

allocated with posix_memalign

page_allocate()

```
with POBITS=12, LEVELS=1: ptbr == 0 page_allocate(0x1000) or page_allocate(0x1001) or ...
```

 $ptbr\ \mathit{now} == \mathsf{GetPointerToTable}($

VI IN Valla: Physical					
0		0			
1		1	(new))	١
2		0)
3		1			

VPN valid? physical

allocated with posix_memalign

posix_memalign

```
void *result;
error code =
     posix_memalign(&result, alignment, size);
allocate size bytes
choosing address that is multiple of alignment
    can make sure allocation starts at beginning of page
error_code indicates if out-of-memory, etc.
fills in result (passed via pointer)
```

posix_memalign

```
void *result;
error code =
     posix_memalign(&result, alignment, size);
allocate size bytes
choosing address that is multiple of alignment
    can make sure allocation starts at beginning of page
error_code indicates if out-of-memory, etc.
fills in result (passed via pointer)
```

posix_memalign

```
void *result;
error code =
     posix_memalign(&result, alignment, size);
allocate size bytes
choosing address that is multiple of alignment
    can make sure allocation starts at beginning of page
error_code indicates if out-of-memory, etc.
fills in result (passed via pointer)
```

parts

```
part 1 (next week): LEVELS=1, POBITS=12 and
    translate() OR
     page_allocate()
part 2: all LEVELS, both functions
    in preparation for code review
     originally scheduled for lab on the 27th
    will move to lab just after reading day
     (might mean I need to cancel lab one week)
part 3: final submission
     Friday after code review
     most of grade based on this
     will test previous parts again
```

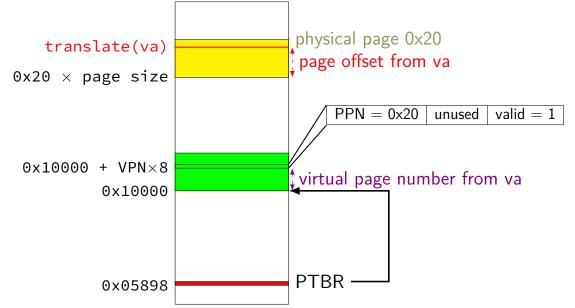
address/page table entry format

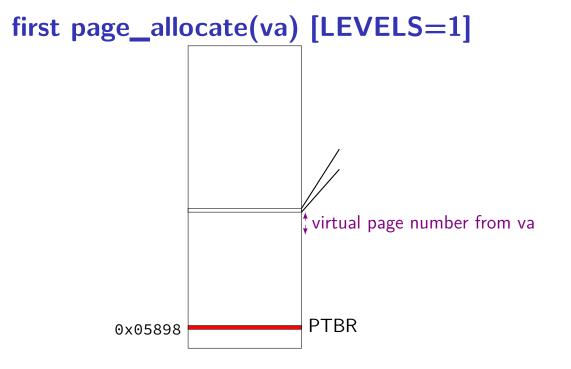
(with POBITS=12, LEVELS=1)

	bits 63–21	bits 20–12	bits 11–1	bit 0
page table entry	physical page number		unused	valid bit
virtual address	unused virtual page number		page o	offset
physical address	physical page number		page o	offset

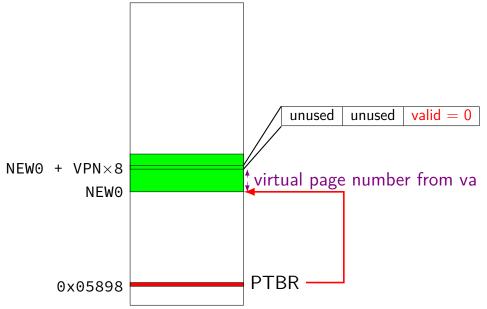
in assignment: value from posix_memalign = physical address

pa = translate(va) [LEVELS=1]

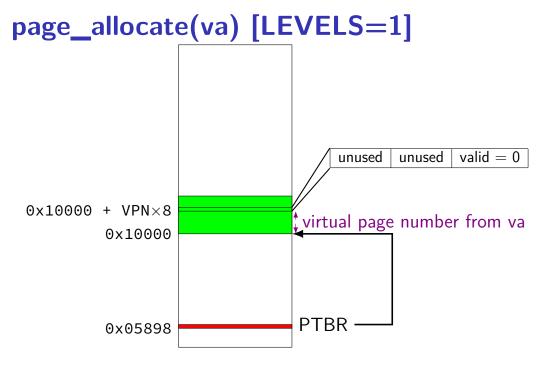


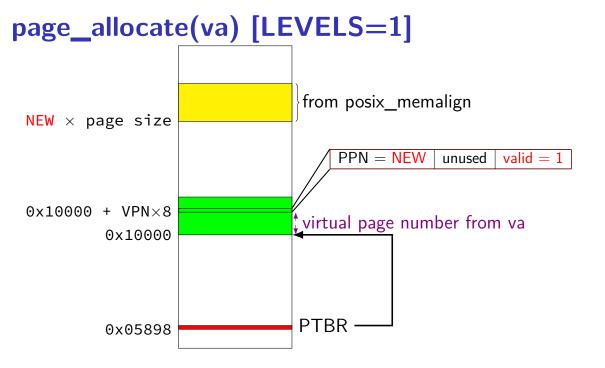


first page_allocate(va) [LEVELS=1]

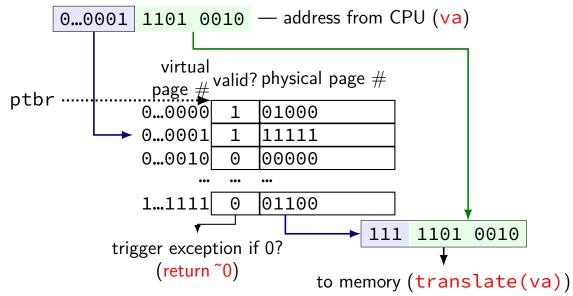


first page_allocate(va) [LEVELS=1] from posix_memalign $NEW1 \times page size$ PPN = NEW1valid = 1unused NEW0 + VPN×8 ‡virtual page number from va NEW₀ **PTBR** 0x05898

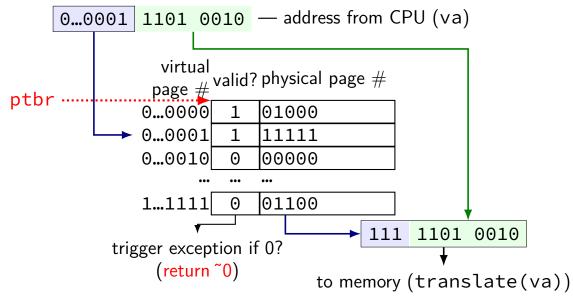




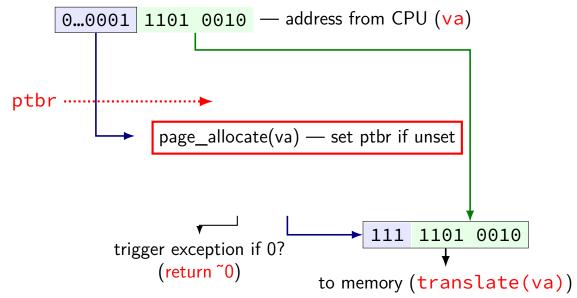
page table lookup (and translate())



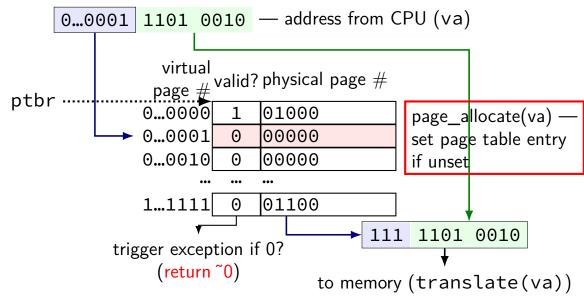
page table lookup (and translate())



page table lookup (and allocate)



page table lookup (and allocate)



my desktop: 39-bit physical addresses; 48-bit virtual addresses

4096 byte pages

my desktop: 39-bit physical addresses; 48-bit virtual addresses

4096 byte pages

top 16 bits of 64-bit addresses not used for translation

my desktop: 39-bit physical addresses; 48-bit virtual addresses

4096 byte pages

exercise: how many page table entries? (assuming page table like shown before)

exercise: how large are physical page numbers?

my desktop: 39-bit physical addresses; 48-bit virtual addresses

4096 byte pages

exercise: how many page table entries? (assuming page table like shown before)

exercise: how large are physical page numbers?

```
my desktop: 39-bit physical addresses; 48-bit virtual addresses
```

4096 byte pages

exercise: how many page table entries? (assuming page table like shown before)

exercise: how large are physical page numbers?

page table entries are 8 bytes (room for expansion, metadata) trick: power of two size makes table lookup faster

would take up 2^{39} bytes?? (512GB??)

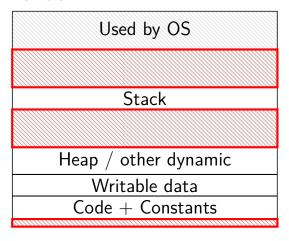
huge page tables

huge virtual address spaces!

impossible to store PTE for every page

how can we save space?

holes



most pages are invalid

saving space

basic idea: don't store (most) invalid page table entries
use a data structure other than a flat array
 want a map — lookup key (virtual page number), get value (PTE)
options?

saving space

basic idea: don't store (most) invalid page table entries
use a data structure other than a flat array
want a map — lookup key (virtual page number), get value (PTE)
options?

hashtable

actually used by some historical processors but never common

saving space

basic idea: don't store (most) invalid page table entries
use a data structure other than a flat array
 want a map — lookup key (virtual page number), get value (PTE)
options?

hashtable

actually used by some historical processors but never common

tree data structure

but not quite a search tree

search tree tradeoffs

lookup usually implemented in hardware

lookup should be simple solution: lookup splits up address bits (no complex calculations)

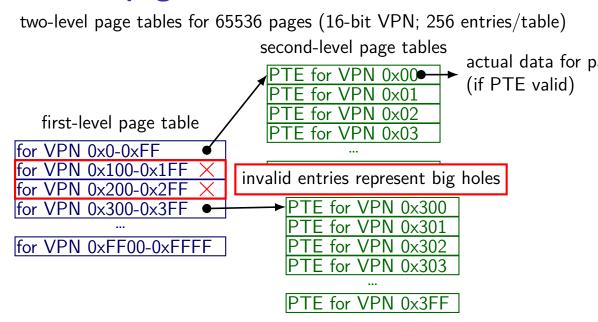
lookup should not involve many memory accesses

doing two memory accesses is already very slow solution: tree with many children from each node

(far from binary tree's left/right child)

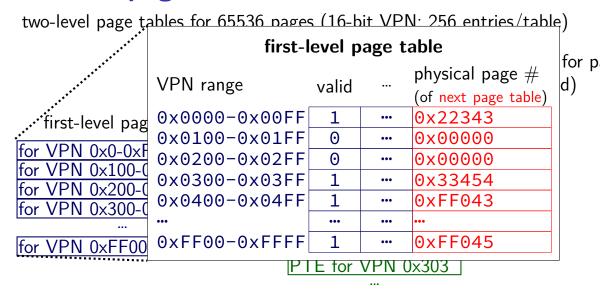
two-level page tables for 65536 pages (16-bit VPN; 256 entries/table) second-level page tables actual data for p for VPN 0x00 (if PTE valid) first-level page table for VPN $0 \times 0 - 0 \times FF$ for VPN 0x100-0x1FF PTE for VPN 0xFF VPN 0x200-0x2FF VPN 0x300 for VPN 0x300-0x3FF for VPN 0xFF00-0xFFFF ΓE for VPN 0x302 TE for VPN 0x303

for VPN 0x3FF



two-level page tables for 65536 pages (16-bit VPN: 256 entries/table) first-level page table for p physical page # VPN range valid d) (of next page table) 0x0000-0x00FF 0x22343 first-level pag $0 \times 0100 - 0 \times 01 FF$ 0 0×00000 VPN 0x0-0xF $0 \times 0200 - 0 \times 02FF$ 0 0×00000 VPN 0x100-0 $0 \times 0300 - 0 \times 03FF$ 0x33454 VPN 0x200- $0 \times 0400 - 0 \times 04FF$ 0xFF043 0xFF045 $0 \times FF00 - 0 \times FFFF$ •••

PTE for VPN 0x3FF

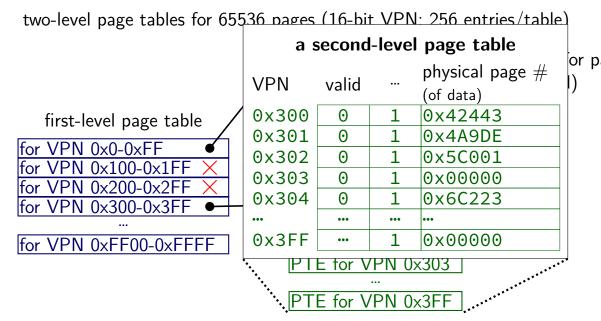


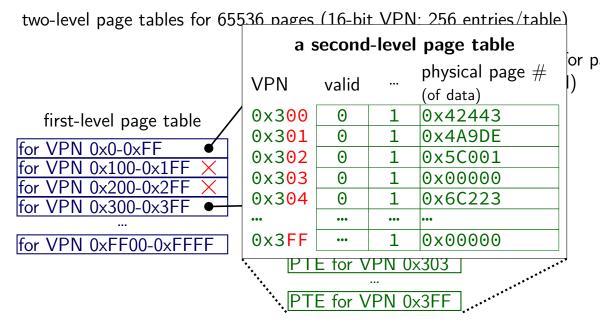
PTE for VPN 0x3FF

two-level page tables for 65536 pages (16-bit VPN: 256 entries/table) first-level page table for p physical page # VPN range valid d) (of next page table) $0 \times 0 0 0 0 - 0 \times 0 0 FF$ 0x22343 first-level pag $0 \times 0100 - 0 \times 01FF$ 0 0×00000 VPN 0x0-0xF 0 0×00000 VPN 0x100-0 $0 \times 0300 - 0 \times 03FF$ 0x33454 VPN 0x200- $0 \times 0400 - 0 \times 04FF$ 0xFF043 0xFF045 $0 \times FF00 - 0 \times FFFF$ •••

TE for VPN 0x3FF

46





two-level page tables for 65536 pages (16-bit VPN; 256 entries/table) second-level page tables actual data for p for VPN 0x00 (if PTE valid) first-level page table for VPN $0 \times 0 - 0 \times FF$ tor VPN 0x100-0x1FFIPTE for VPN 0xFF VPN 0x200-0x2FF for VPN 0x300-0x3FF VPN 0x300 for VPN 0xFF00-0xFFFF VPN 0x302 TE for VPN 0x303

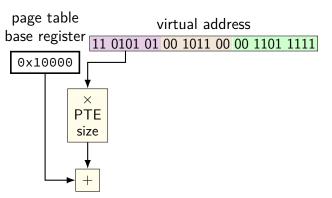
for VPN 0x3FF

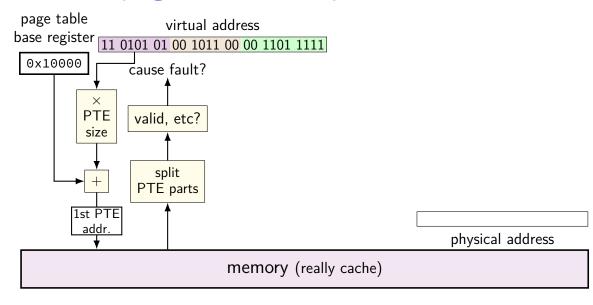
virtual address

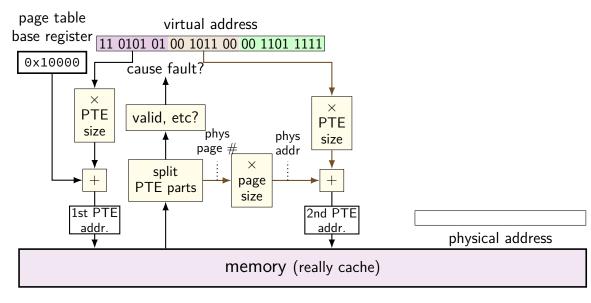
11 0101 01 00 1011 00 00 1101 1111

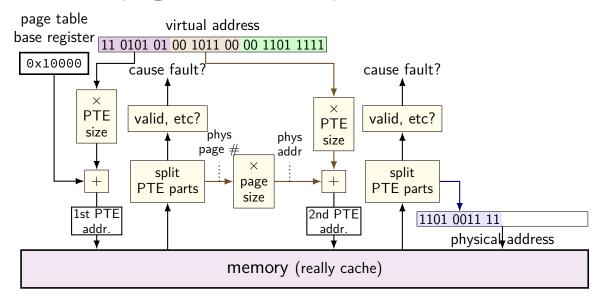
VPN — split into two parts (one per level)

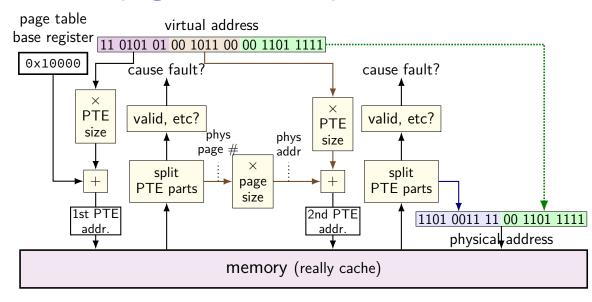
this example: parts equal sized — common, but not required

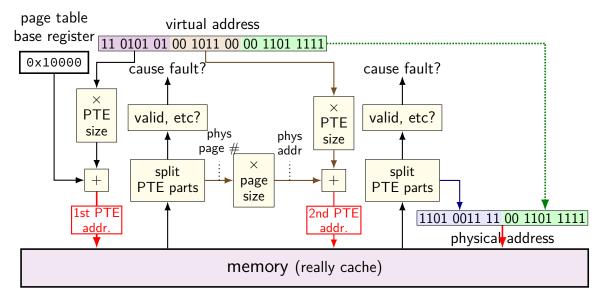


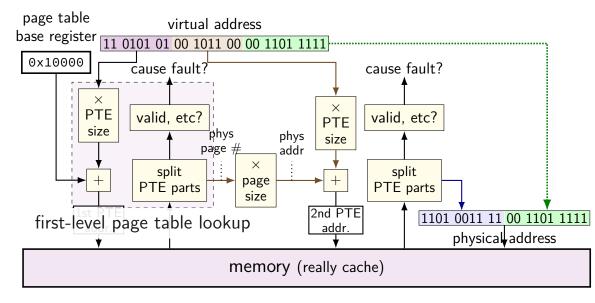


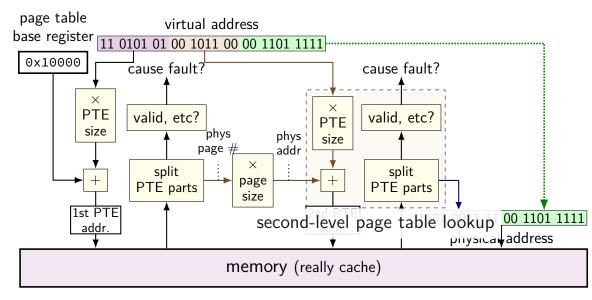


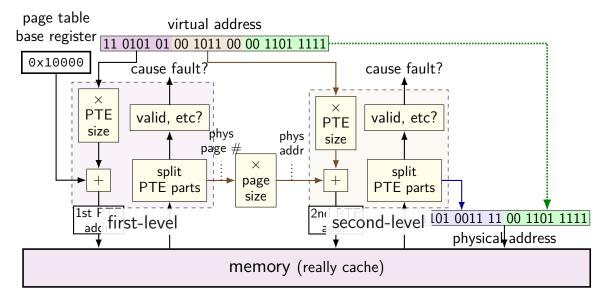


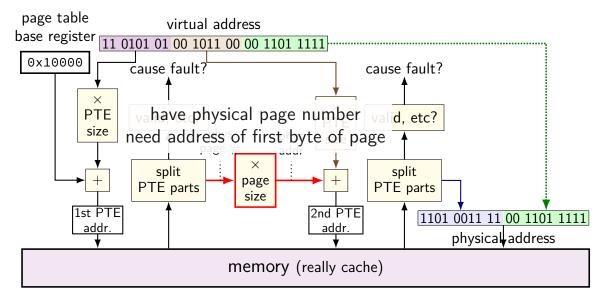


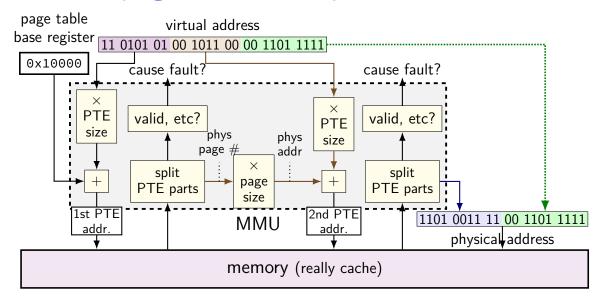




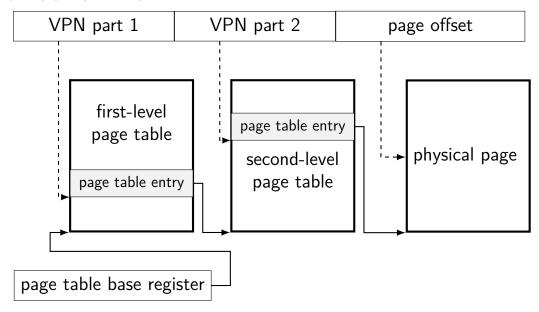








another view



multi-level page tables

VPN split into pieces for each level of page table

top levels: page table entries point to next page table usually using physical page number of next page table

bottom level: page table entry points to destination page

validity checks at each level

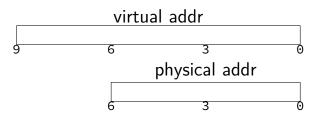
note on VPN splitting

indexes used for lookup parts of the virtual page number (there are not multiple VPNs)

assignment

9-bit virtual address

6-bit physical address



- 9-bit virtual address
- page offset VPN physical addr

virtual addr

PPN

page offset

6-bit physical address

8-byte pages \rightarrow 3-bit page offset (bottom) ⁶

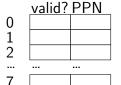
- 9-bit VA: 6 bit VPN + 3 bit PO
- 6-bit PA: 3 bit PPN + 3 bit PO

9-bit virtual address

virtual addr page offset VPN

- 6-bit physical address
- physical addr PPN8-byte pages \rightarrow 3-bit page offset (bottom) ⁶
 - page offset

- 9-bit VA: 6 bit VPN + 3 bit PO
- 6-bit PA: 3 bit PPN + 3 bit PO
- 1 page page tables w/ 1 byte entry \rightarrow 8 entry PTs



page table (either level)

9-bit virtual address

virtual addr

VPN pt 1 VPN pt 2 page offset
9 6 3

PPN

physical addr

page offset

valid? PPN

- 6-bit physical address
- 8-byte pages \rightarrow 3-bit page offset (bottom) ⁶
- 9-bit VA: 6 bit VPN + 3 bit PO page table (either level)
- 6-bit PA: 3 bit PPN + 3 bit PO
- 1 page page tables w/ 1 byte entry ightarrow 8 entry PTs
- 8 entry page tables \rightarrow 3-bit VPN parts
- 9-bit VA: 3 bit VPN part 1; 3 bit VPN part 2

physical bytes addresses
0x20-300 91 72 13
0x24-7F4 A5 36 07
0x28-B89 9A AB BC
0x2C-FCD DE EF F0
0x30-3BA 0A BA 0A
0x34-7DB 0B DB 0B
0x38-BEC 0C EC 0C
0x3C-FAC DC DC 0C

physical bytes addresses								
0x00-3	00	11	22	33				
0x04-7	44	55	66	77				
0x08-B								
0x0C-F	CC	DD	EE	FF				
0x10-3	1A	2A	3A	4A				
0x14-7								
0x18-B								
0x1C-F	1 <u>C</u>	2C	3C	4C				

physical ddresses	byt	es		
0x20-3	00	91	72	13
0x24-7	F4	Α5	36	07
0x28-B	89	9A	ΑB	ВС
0x2C-F	CD	DE	EF	F0
0x30-3	ВА	0A	ВА	0A
0x34-7	DΒ	0B	DB	0B
0x38-B	EC	0C	EC	0C
0x3C-F	AC	DC	DC	0C

physical bytes addresses									
0x00-3									
0x04-7	44	55	66	77					
0x08-B	88	99	AΑ	ВВ					
0x0C-F	CC	DD	EE	FF					
0x10-3	1A	2A	3A	4A					
0x14-7	1B	2B	3B	4B					
0x18-B	1C	2C	3C	4C					
0x1C-F	1C	2C	3C	4C					

physical	byt	es		
addresses				
0x20-3				
0x24-7				
0x28-B	89	9A	ΑB	ВС
0x2C-F	CD	DE	EF	F0
0x30-3	ВА	0A	ВА	0A
0x34-7	DB	0B	DB	0B
0x38-B	EC	0C	EC	0C
0x3C-F	AC	DC	DC	0C

physical bytes addresses									
0x00-3									
0x04-7	44	55	66	77					
0x08-B	88	99	AΑ	ВВ					
0x0C-F	CC	DD	EE	FF					
0x10-3									
0x14-7	1B	2B	3B	4B					
0x18-B	1C	2C	3C	4C					
0x1C-F	1C	2C	3C	4C					

physical addresses	byt	es		
0x20-3	00	91	72	13
0x24-7	F4	Α5	36	07
0x28-B				
0x2C-F				
0x30-3	ВА	0A	ВА	0A
0x34-7	DB	0B	DB	0B
0x38-B	EC	0C	EC	0C
0x3C-F	AC.	DC	DC	0C

physical bytes addresses									
0x00-3									
0x04-7	44	55	66	77					
0x08-B	88	99	AΑ	ВВ					
0x0C-F	CC	DD	EE	FF					
0x10-3	1A	2A	3A	4A					
0x14-7	1В	2B	3B	4B					
0x18-B	1C	2C	3C	4C					
0x1C-F	1C	2C	3C	4C					

physical addresses	byt	es		
0x20-3	00	91	72	13
0x24-7	F4	Α5	36	07
0x28-B	89	9A	ΑB	ВС
0x2C-F	CD	DE	EF	F0
0x30-3	ВА	0A	ВА	0A
0x34-7				
0x38-B	EC	0C	EC	0C
0x3C-F	AC.	DC	DC	0C

2-level exercise (1)

physical addresses	byte	es			phy addr	/sical esses	byt	es		
0x00-3			22	33		20-3			D2	D3
0x04-7	44	55	66	77	0x2	4-7	D4	D5	D6	D7
0x08-B	88	99	AA	ВВ	0x2	28-B	89	9A	AB	ВС
0x0C-F	CC	DD	EE	FF	0x2	C-F	CD	DE	EF	F0
0x10-3	1A	2A	ЗА	4A	0x3	80-3	ВА	0A	ВА	0A
0x14-7	1В	2B	3B	4B	0x3	84-7	DB	0B	DB	0B
0x18-B	1C	2C	3C	4C	0x3	88-B	EC	0C	EC	0C
0x1C-F	1C	2C	3C	4C	0x3	C-F	FC	0C	FC	0C

2-level exercise (1)

physical addresses	byt	es			phy addr	ysical esses	byt	es		
0x00-3			22	33	0x2	20-3	D0	D1	D2	D3
0x04-7	44	55	66	77	0x2	24-7	D4	D5	D6	D7
0x08-B	88	99	AA	ВВ	0x2	28-B	89	9A	AB	ВС
0x0C-F	CC	DD	EE	FF	0x2	2C-F	CD	DE	EF	F0
0x10-3	1A	2A	ЗА	4A	0x3	80-3	ВА	0A	ВА	0A
0x14-7	1B	2B	3B	4B	0x3	34-7	DB	0B	DB	0B
0x18-B	10	2C	3C	4C	0x3	88-B	EC	0C	EC	0C
0x1C-F	1C	2C	3C	4C	0x3	BC-F	FC	0C	FC	0C

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused; page table base register 0x08; translate virtual address 0x0FB

physical addresses	byt	es			a	physical ddresses	byt	es		
0x00-3	00	11	22	33	(9x20-3	D0	D1	D2	D3
0x04-7	44	55	66	77	(9x24-7	D4	D5	D6	D7
0x08-B	88	99	AΑ	BB	(9x28-B	89	9A	AΒ	ВС
0x0C-F	CC	DD	ΕE	FF	(9x2C−F	CD	DE	EF	F0
0x10-3	1A	2A	3A	4A	(9x30-3	ВА	0Α	ВА	ΘΑ
0x14-7	1В	2B	3B	4B	(9x34-7	DB	0B	DB	0B
0x18-B	1C	2C	3C	4C	(9х38-В	EC	0C	EC	00
0x1C-F	1C	2C	3C	4C	(0x3C-F	FC	0C	FC	0C

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused; page table base register 0x08; translate virtual address 0x0FB

physical addresses	byt	es			p ad	hysical dresses	byt	es		
0x00-3			22	33		x20-3			D2	D3
0x04-7	44	55	66	77	0)	x24-7	D4	D5	D6	D7
0x08-B	88	99	AΑ	ВВ	0)	x28-B	89	9A	AΒ	ВС
0x0C-F	CC	DD	ΕE	FF	0)	x2C-F	CD	DE	EF	F0
0x10-3	1A	2A	3A	4A	0)	x30-3	ВА	0A	ВА	0A
0x14-7	1B	2B	3B	4B	0)	x34-7	DΒ	0B	DB	0B
0x18-B	1C	2C	3C	4C	0)	x38-B	EC	0C	EC	00
0x1C-F	1C	2C	3C	4C	0)	x3C-F	FC	0C	FC	00

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused; page table base register 0x08; translate virtual address 0x0FB

physical addresses	byte	es			physic address	al es	byt	es		
0x00-3			22	33	0x20-				D2	D3
0x04-7	44	55	66	77	0x24-	-7[D4	D5	D6	D7
0x08-B	88	99	AΑ	ВВ	0x28-	-B[89	9A	ΑB	ВС
0x0C-F	CC	DD	EE	FF	0x2C-	-F[CD	DE	EF	F0
0x10-3	1A	2A	3A	4A	0x30-	-3[ВА	0Α	ВА	0Α
0x14-7	1B	2B	3B	4B	0x34-	-7[DB	0B	DB	0B
0x18-B	1C	2C	3C	4C	0x38-	-в[EC	0C	EC	0C
0x1C-F	1C	2C	3C	4C	0x3C-	-F[FC	0C	FC	0C

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused; page table base register 0x10; translate virtual address 0x109

physical addresses	byt	es		
0x00-3	00	11	22	33
0x04-7	44	55	66	77
0x08-B				
0x0C-F				
0x10-3				
0x14-7			3B	
0x18-B			3C	
0x1C-F	1C	2C	3C	4C

physical addresses	byt	es		
0x20-3	D0	D1	D2	D3
0x24-7	D4	D5	D6	D7
0x28-B	89	9A	ΑB	ВС
0x2C-F	CD	DE	EF	F0
0x30-3	ВА	0A	ВА	0A
0x34-7	DB	0B	DB	0B
0x38-B	EC	0C	EC	0C
0x3C-F	FC	0C	FC	0C

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused page table base register 0x08; translate virtual address 0x00B

physical addresses	byte	es			physic address	cal ses	byte	es		
0x00-3			22	33	0x20-				D2	D3
0x04-7	44	55	66	77	0x24-	-7	D4	D5	D6	D7
0x08-B	88	99	AΑ	ВВ	0x28-	-в	89	9A	ΑB	ВС
0x0C-F	CC	DD	EE	FF	0x2C-	-F	CD	DE	EF	F0
0x10-3	1A	2A	3A	4A	0x30-	-3	ВА	0Α	ВА	0Α
0x14-7	1B	2B	3B	4B	0x34-	-7	DB	0B	DB	0B
0x18-B	1C	2C	3C	4C	0x38-	-В	EC	0C	EC	0C
0x1C-F	1C	2C	3C	4C	0x3C-	-F	FC	0C	FC	0C

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused page table base register 0x08; translate virtual address 0x00B

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical ddresses	byt	es		
0x20-3			D2	D3
0x24-7	D4	D5	D6	D7
0x28-B	89	9A	ΑB	ВС
0x2C-F	CD	DE	EF	F0
0x30-3	ВА	0A	ВА	0A
0x34-7	DB	0B	DB	0B
0x38-B	EC	0C	EC	0C
0x3C-F	FC	0C	FC	0C

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused page table base register 0x08; translate virtual address 0x00B

physical addresses	byt	es		
0x00-3	00	11	22	33
0x04-7	44	55	66	77
0x08-B				
0x0C-F	CC	DD	EE	FF
0x10-3				
			3B	
0x18-B				
0x1C-F	1C	2C	3C	4C

physical ddresses	byt	es		
0x20-3	D0	D1	D2	D3
0x24-7	D4	D5	D6	D7
0x28-B	89	9A	ΑB	ВС
0x2C-F	CD	DE	EF	F0
0x30-3	ВА	0A	ВА	0A
0x34-7	DB	0B	DB	0B
0x38-B	EC	0C	EC	0C
0x3C-F	FC	0C	FC	0C

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused page table base register 0x08; translate virtual address 0x1CB

physical addresses	byte	es			physical addresses	byt	es		
0x00-3			22	33	0x20-3			D2	D3
0x04-7	44	55	66	77	0x24-7	D4	D5	D6	D7
0x08-B	88	99	AΑ	ВВ	0x28-B	89	9A	ΑB	ВС
0x0C-F	CC	DD	EE	FF	0x2C-F	CD	DE	EF	F0
0x10-3	1A	2A	3A	4A	0x30-3	ВА	0A	ВА	0Α
0x14-7	1B	2B	3B	4B	0x34-7	DB	0B	DB	0B
0x18-B	1C	2C	3C	4C	0x38-B	EC	0C	EC	0C
0x1C-F	1C	2C	3C	4C	0x3C-F	FC	0C	FC	0C

10-bit virtual addresses, 6-bit physical; 16 byte pages, 2 byte PTE

page tables 1 page; PTE 1st byte: (MSB) 2-bit PPN, valid bit; rest unused

physical bytes addresses										
addresses										
0x00-3	00	11	22	33						
0x04-7	44	55	66	77						
0x08-B										
0x0C-F	CC	DD	EE	FF						
0x10-3	1A	2A	3A	4A						
0x14-7	1В	2B	3B	4B						
0x18-B										
0x1C-F	ΑC	ВС	DC	EC						

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

0x3C-F FC 0C FC 0C
```

10-bit virtual addresses, 6-bit physical; 16 byte pages, 2 byte PTE

page tables 1 page; PTE 1st byte: (MSB) 2-bit PPN, valid bit; rest unused

physical addresses	byt	<u> </u>		
addresses	<i></i>			
0x00-3				
0x04-7	44	55	66	77
0x08-B				
0x0C-F	CC	DD	EE	FF
0x10-3	1A	2A	3A	4A
0x14-7	1B	2B	3B	4B
0x18-B			3C	
0x1C-F	AC	ВС	DC	EC

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

0x3C-F FC 0C FC 0C
```

10-bit virtual addresses, 6-bit physical; 16 byte pages, 2 byte PTE

page tables 1 page; PTE 1st byte: (MSB) 2-bit PPN, valid bit; rest unused

physical addresses	byt	<u> </u>		
addresses	<i></i>			
0x00-3				
0x04-7	44	55	66	77
0x08-B				
0x0C-F	CC	DD	EE	FF
0x10-3	1A	2A	3A	4A
0x14-7	1B	2B	3B	4B
0x18-B			3C	
0x1C-F	AC	ВС	DC	EC

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

0x3C-F FC 0C FC 0C
```

10-bit virtual addresses, 6-bit physical; 16 byte pages, 2 byte PTE

page tables 1 page; PTE 1st byte: (MSB) 2-bit PPN, valid bit; rest unused

physical addresses	bvt	es		
addresses,	-) -			
0x00-3				
0x04-7	44	55	66	77
0x08-B	88	99	AΑ	ВВ
0x0C-F	CC	DD	EE	FF
0x10-3	1A	2A	ЗА	4A
0x14-7	1B	2B	3B	4B
0x18-B	1C	2C	3C	4C
0x1C-F	AC	ВС	DC	EC

```
physical bytes addresses 0x20-3 D0 E1 D2 D3 0x24-7 D4 E5 D6 E7 0x28-B 89 9A AB BC 0x2C-F CD DE EF F0 0x30-3 BA 0A BA 0A 0x34-7 DB 0B DB 0B 0x38-B EC 0C EC 0C 0x3C-F FC 0C FC 0C
```

10-bit virtual addresses, 6-bit physical; 16 byte pages, 2 byte PTE

page tables 1 page; PTE 1st byte: (MSB) 2-bit PPN, valid bit; rest unused

physical addresses	byt	<u> </u>		
addresses	<i></i>			
0x00-3				
0x04-7	44	55	66	77
0x08-B				
0x0C-F	CC	DD	EE	FF
0x10-3	1A	2A	3A	4A
0x14-7	1B	2B	3B	4B
0x18-B			3C	
0x1C-F	AC	ВС	DC	EC

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

0x3C-F FC 0C FC 0C
```

10-bit virtual addresses, 6-bit physical; 16 byte pages, 2 byte PTE

page tables 1 page; PTE 1st byte: (MSB) 2-bit PPN, valid bit; rest unused

physical addresses	byt	es		
0x00-3			22	33
0x04-7	44	55	66	77
0x08-B				
0x0C-F	CC	DD	EE	FF
0x10-3	1A	2A	3A	4A
0x14-7				
0x18-B	1C	2C	3C	4C
0x1C-F	AC	ВС	DC	EC

```
physical bytes addresses 0x20-3 D0 E1 D2 D3 0x24-7 D4 E5 D6 E7 0x28-B 89 9A AB BC 0x2C-F CD DE EF F0 0x30-3 BA 0A BA 0A 0x34-7 DB 0B DB 0B 0x38-B EC 0C EC 0C 0x3C-F FC 0C FC 0C
```

10-bit virtual addresses, 6-bit physical; 16 byte pages, 2 byte PTE

page tables 1 page; PTE 1st byte: (MSB) 2-bit PPN, valid bit; rest unused

physical addresses	byt	<u> </u>		
addresses	<i></i>			
0x00-3				
0x04-7	44	55	66	77
0x08-B				
0x0C-F	CC	DD	EE	FF
0x10-3	1A	2A	3A	4A
0x14-7	1B	2B	3B	4B
0x18-B			3C	
0x1C-F	AC	ВС	DC	EC

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

0x3C-F FC 0C FC 0C
```

10-bit virtual addresses, 6-bit physical; 16 byte pages, 2 byte PTE

page tables 1 page; PTE 1st byte: (MSB) 2-bit PPN, valid bit; rest unused

physical addresses	byt	es			
0x00-3	00	11	22	33	
0x04-7	44	55	66	77	
0x08-B	88	99	AΑ	ВВ	
0x0C-F	CC	DD	EE	FF	
0x10-3	1A	2A	3A	4A	
0x14-7	1B	2B	3B	4B	
0x18-B	1C	2C	3C	4C	
0x1C-F	AC	ВС	DC	EC	

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

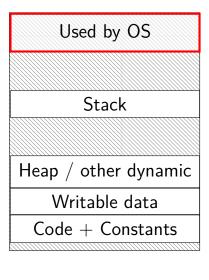
0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

0x3C-F FC 0C FC 0C
```

backup slides

program memory



0xFFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F...

0x0000 0000 0040 0000

system calls, I/O events, etc. run OS code in kernel mode

system calls, I/O events, etc. run OS code in kernel mode

where in memory is this OS code?

system calls, I/O events, etc. run OS code in kernel mode

where in memory is this OS code?

probably have a page table entry pointing to it marked not accessible in user mode

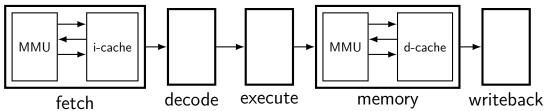
system calls, I/O events, etc. run OS code in kernel mode

where in memory is this OS code?

probably have a page table entry pointing to it marked not accessible in user mode

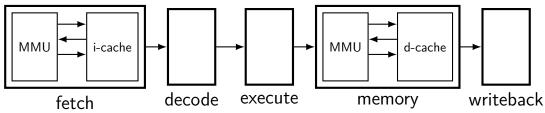
code better not be modified by user program otherwise: uncontrolled way to "escape" user mode

MMUs in the pipeline



up to four memory accesses per instruction

MMUs in the pipeline



up to four memory accesses per instruction challenging to make this fast (topic for a future date)

do we really need a complete copy?

bash	new copy of bash
Used by OS	Used by OS
Stack	Stack
Heap / other dynamic	Heap / other dynamic
Writable data	Writable data
Code + Constants	Code + Constants

do we really need a complete copy?

bash	new copy of bash		
Used by OS	Used by OS		
Stack	Stack		
Heap / other dynamic	Heap / other dynamic		
Writable data	Writable data		
Code + Constants	Code + Constants		

shared as read-only

do we really need a complete copy?

bash	new copy of bash
Used by OS	Used by OS
Stack	Stack
Heap / other dynamic	Heap $/$ other dynamic
Writable data	Writable data
Code + Constants can't b	e shared? Code + Constants

trick for extra sharing

```
sharing writeable data is fine — until either process modifies it example: default value of global variables might typically not change (or OS might have preloaded executable's data anyways)
```

can we detect modifications?

trick for extra sharing

```
sharing writeable data is fine — until either process modifies it example: default value of global variables might typically not change (or OS might have preloaded executable's data anyways)
```

can we detect modifications?

trick: tell CPU (via page table) shared part is read-only processor will trigger a fault when it's written

VPN

valid? write?

•••

0x00601 0x00602 0x00603 0x00604 0x00605

		F-0-
•••	•••	•••
1	1	0x12345
1	1	0x12347
1	1	0x12340
1	1	0x200DF
1	1	0x200AF
•••	•••	•••

VPN
•••
0x00601
0x00602
0x00603
0x00604
0x00605
•••

physical valid? write? page					
•••	•••	•••			
1	0	0x12345			
1	0	0x12347			
1	0	0x12340			
1	0	0x200DF			
1	0	0x200AF			
•••	•••	•••			

•••
0x00601
0x00602
0x00603
0x00604
0x00605
•••

VPN

•••	•••	•••			
valid? write? page					

•••	•••	•••
1	0	0x12345
1	0	0x12347
1	0	0x12340
1	0	0x200DF
1	0	0x200AF
•••	•••	•••

copy operation actually duplicates page table both processes share all physical pages but marks pages in both copies as read-only

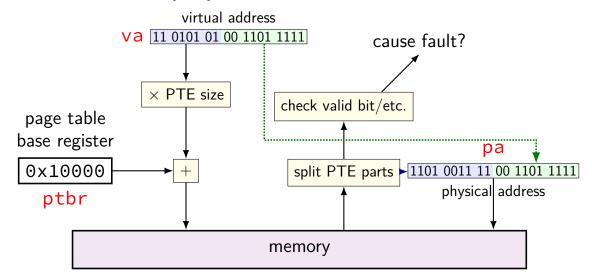
VPN	valid? write? page		physical page	VPN	VPN valid? write		physical page
•••	•••	•••	•••	•••	•••	•••	•••
0x00601	1	0	0x12345	0x00601	1	0	0x12345
0x00602	1	0	0x12347	0x00602	1	0	0x12347
0x00603	1	0	0x12340	0x00603	1	0	0x12340
0x00604	1	0	0x200DF	0x00604	1	0	0x200DF
0x00605	1	0	0x200AF	0x00605	1	0	0x200AF
•••	•••	•••	•••	•••	•••	•••	•••

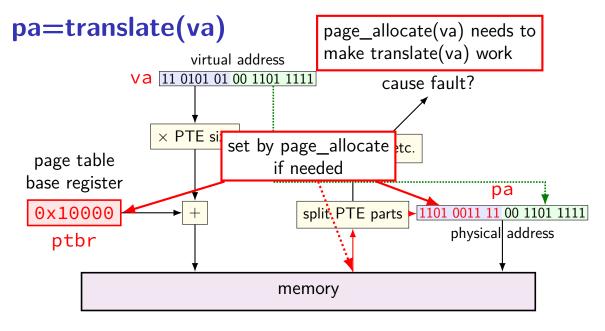
when either process tries to write read-only page triggers a fault — OS actually copies the page

VPN	valid? write? page		physical	VPN	physical valid? write? page			
valid: write: pa		[:] page	VIIN	valiu:	wille:	page		
•••	•••	•••	•••	•••	•••	•••	•••	
0x00601	1	0	0x12345	0x00601	1	0	0x12345	
0x00602	1	0	0x12347	0x00602	1	0	0x12347	
0x00603	1	0	0x12340	0x00603	1	0	0x12340	
0x00604	1	0	0x200DF	<u>0x00604</u>	1	0	0x200DF	
0x00605	1	0	0x200AF	0x00605	1	1	0x300FD	
•••	•••	•••	•••	•••	•••	•••	•••	

after allocating a copy, OS reruns the write instruction

pa=translate(va)





swapping

early motivation for virtual memory: swapping

using disk (or SSD, ...) as the next level of the memory hierarchy how our textbook and many other sources presents virtual memory

OS allocates program space on disk own mapping of virtual addresses to location on disk

DRAM is a cache for disk

swapping

early motivation for virtual memory: swapping

using disk (or SSD, ...) as the next level of the memory hierarchy how our textbook and many other sources presents virtual memory

OS allocates program space on disk own mapping of virtual addresses to location on disk

DRAM is a cache for disk

swapping components

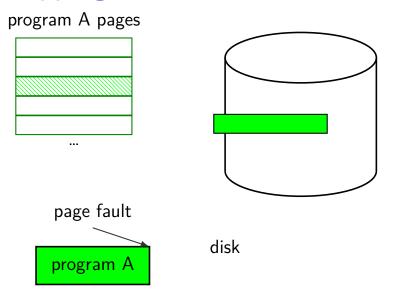
```
"swap in" a page — exactly like allocating on demand!
    OS gets page fault — invalid in page table
    check where page actually is (from virtual address)
    read from disk
    eventually restart process
"swap out" a page
    OS marks as invalid in the page table(s)
    copy to disk (if modified)
```

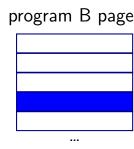
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

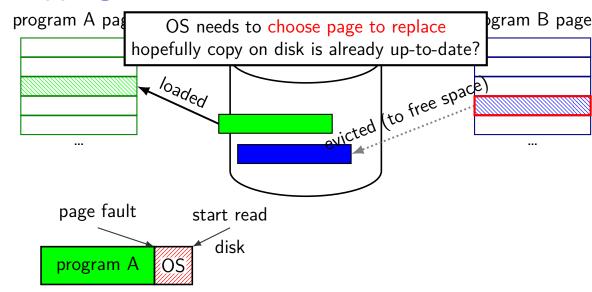
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

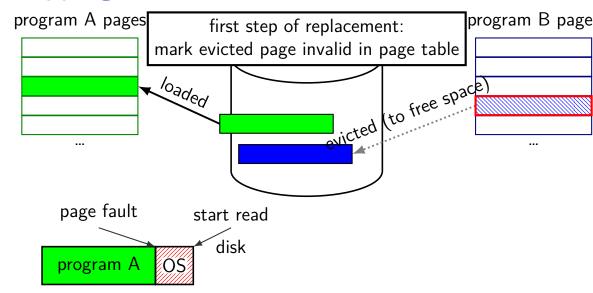
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

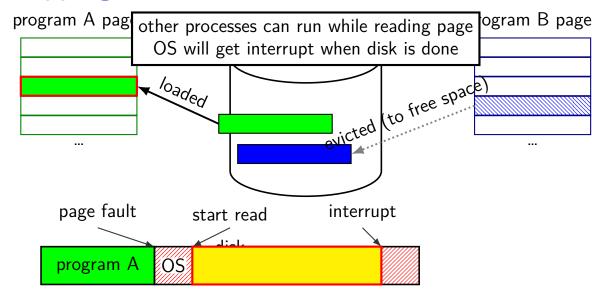
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

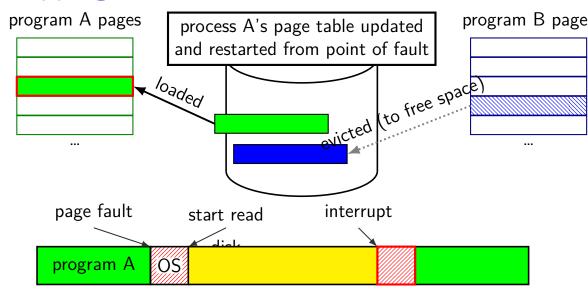












swapping almost mmap

```
access mapped file for first time, read from disk (like swapping when memory was swapped out)
```

```
write "mapped" memory, write to disk eventually (like writeback policy in swapping) use "dirty" bit
```

extra detail: other processes should see changes all accesses to file use same physical memory