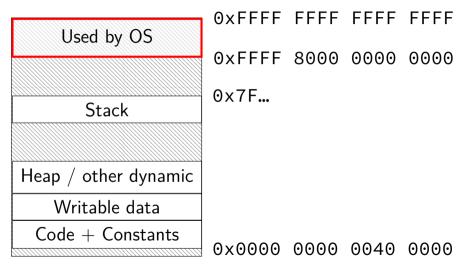
program memory

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

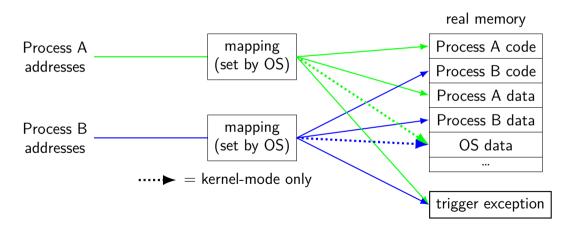
0x0000 0000 0040 0000

program memory



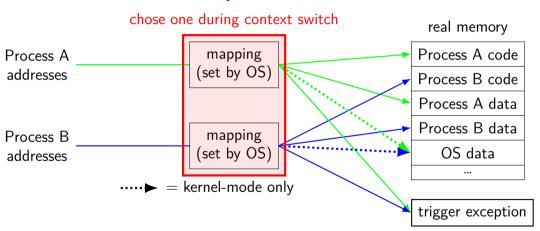
address spaces

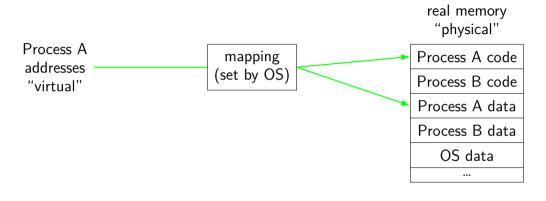
illuision of dedicated memory

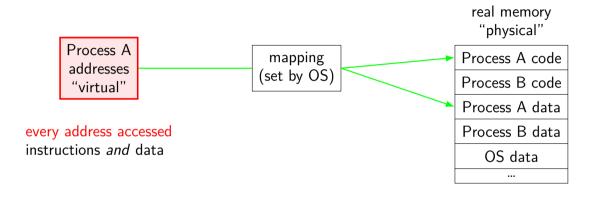


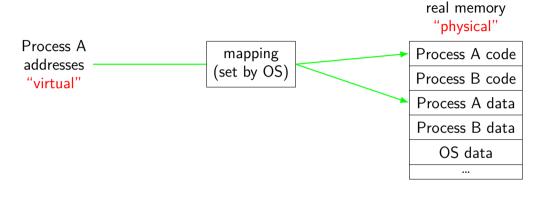
address spaces

illuision of dedicated memory

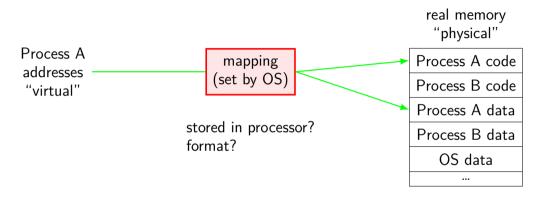


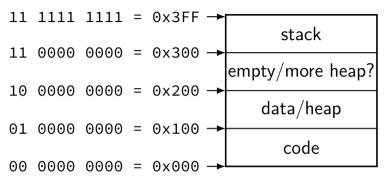


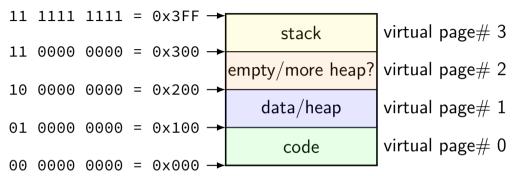


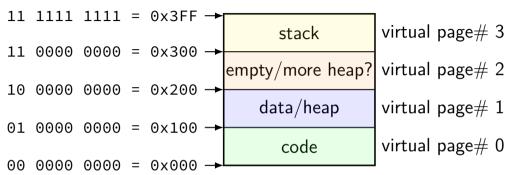


program addresses are 'virtual' real addresses are 'physical' can be different sizes!

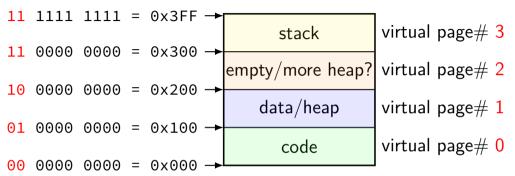




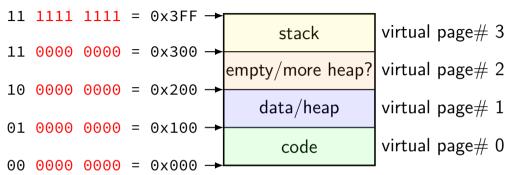




divide memory into pages (2^8 bytes in this case) "virtual" = addresses the program sees



page number is upper bits of address (because page size is power of two)



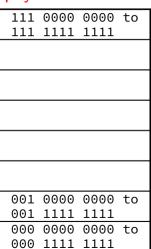
rest of address is called page offset

toy physical memory

program memory virtual addresses

11	0000	0000	to
11	1111	1111	
10	0000	0000	to
10	1111	1111	
01	0000	0000	to
01	1111	1111	
00	0000	0000	to
00	1111	1111	

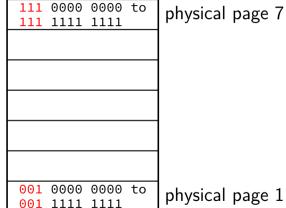
real memory physical addresses



toy physical memory

real memory physical addresses

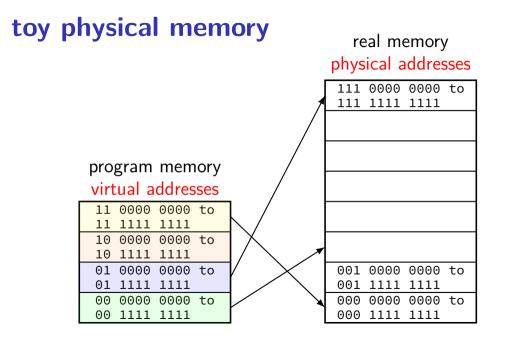
program memory						
virtual addresses						
11	0000	0000	to			
11	1111	1111				
10	0000	0000	to			
10	1111	1111				
01	0000	0000	to			
01	1111	1111				
00	0000	0000	to			
00	1111	1111				

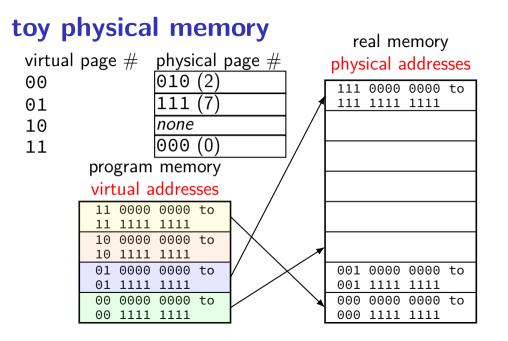


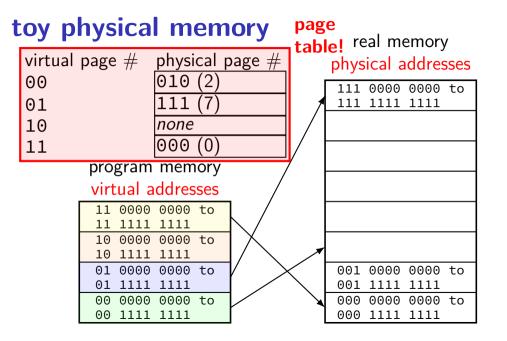
0000

0000 to

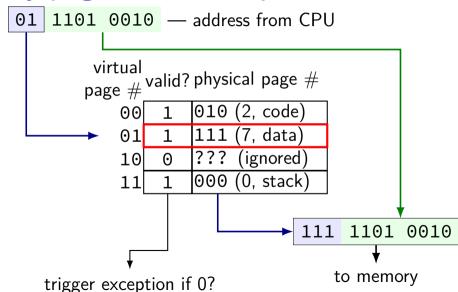
physical page 1 physical page 0



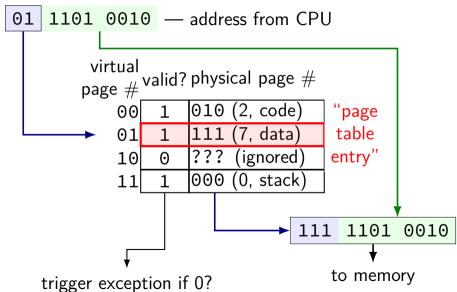




```
virtual page # valid? physical page # 00 1 010 (2, code) 01 1 111 (7, data) 10 0 ??? (ignored) 11 1 000 (0, stack)
```



7



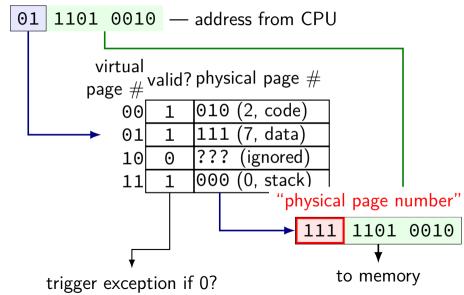
7

t "virtual page number" ookup 1101 0010 — address from CPU virtual page # valid? physical page #010 (2, code) 00data 01 10 0 (ignored) 000 (0, stack) 1101 0010

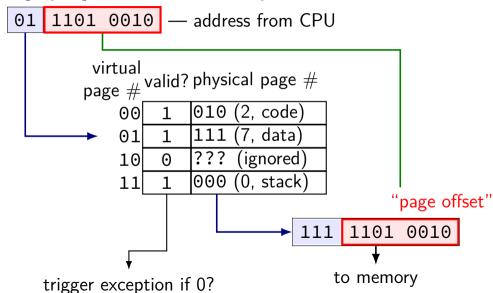
trigger exception if 0?

to memory

7



toy pag "page offset" ookup



on virtual address sizes

virtual address size = size of pointer?

often, but — sometimes part of pointer not used

example: typical x86-64 only use 48 bits rest of bits have fixed value

virtual address size is amount used for mapping

address space sizes

amount of stuff that can be addressed = address space size based on number of unique addresses

e.g. 32-bit virtual address = 2^{32} byte virtual address space

e.g. 20-bit physical addresss $=2^{20}$ byte physical address space

address space sizes

amount of stuff that can be addressed = address space size based on number of unique addresses

- e.g. 32-bit virtual address = 2^{32} byte virtual address space
- e.g. 20-bit physical addresss $=2^{20}$ byte physical address space

what if my machine has 3GB of memory (not power of two)?

not all addresses in physical address space are useful
most common situation (since CPUs support having a lot of memory)

exercise: page counting

suppose 32-bit virtual (program) addresses

and each page is 4096 bytes (2^{12} bytes)

how many virtual pages?

exercise: page counting

suppose 32-bit virtual (program) addresses and each page is 4096 bytes (2^{12} bytes)

how many virtual pages?

$$2^{32}/2^{12} = 2^{20}$$

exercise: page table size

suppose 32-bit virtual (program) addresses suppose 30-bit physical (hardware) addresses each page is 4096 bytes (2^{12} bytes)

pgae table entries have physical page #, valid bit, bit

how big is the page table (if laid out like ones we've seen)?

exercise: page table size

suppose 32-bit virtual (program) addresses suppose 30-bit physical (hardware) addresses each page is 4096 bytes (2^{12} bytes)

pgae table entries have physical page #, valid bit, bit

how big is the page table (if laid out like ones we've seen)?

 2^{20} entries $\times (18+1)$ bits per entry issue: where can we store that?

exercise: address splitting

and each page is 4096 bytes (2^{12} bytes)

split the address 0x12345678 into page number and page offset:

exercise: address splitting

and each page is 4096 bytes (2^{12} bytes)

split the address 0x12345678 into page number and page offset:

page #: 0x12345; offset: 0x678

exercise: page table lookup

suppose 64-byte pages (= 6-bit page offsets), 9-bit virtual addresses

VPN	valid	PPN
000	1	0010
001	1	1010
010	0	
011	0	
100	1	1110
101	1	0100
110	1	0001
111	0	

virtual address 0x024 (0 0010 0100) = physical address ???

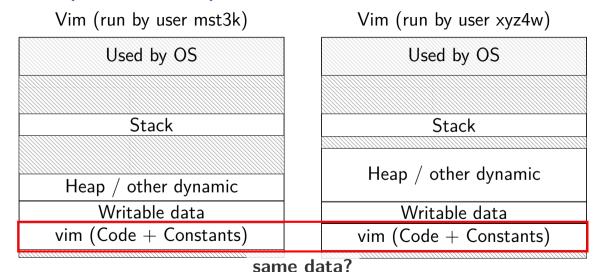
vim (two copies)

Vim (run by user mst3k)

Used by OS Stack Heap / other dynamic Writable data vim (Code + Constants) Vim (run by user xyz4w)

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

vim (two copies)



two copies of program

would like to only have one copy of program

what if mst3k's vim tries to modify its code?

would break process abstraction:

"illusion of own memory"

permissions bits

```
page table entry will have more permissions bits can access in user mode? can read from? can write to? can execute from?
```

checked by hardware like valid bit

page table (logically)

virtual page #		user?	write?	exec?	physical page $\#$
0000 0000	0	0	0	0	00 0000 0000
0000 0001	1	1	1	0	10 0010 0110
0000 0010	1	1	1	0	00 0000 1100
0000 0011	1	1	0	1	11 0000 0011
•••					
1111 1111[1	0	1	0	00 1110 1000

running a program

Some program

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

running a program

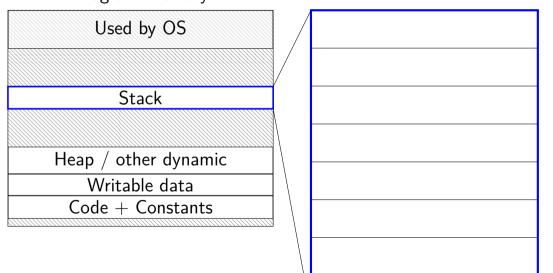
Some program

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

OS's memory

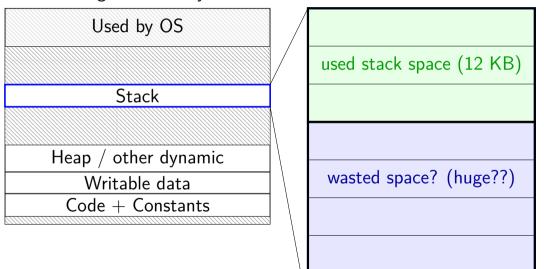
space on demand

Program Memory



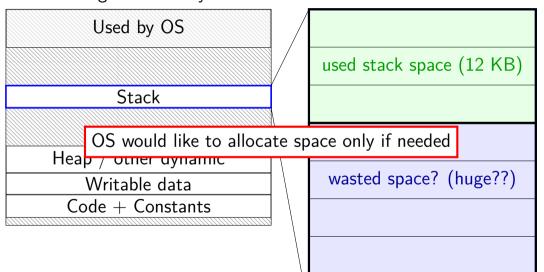
space on demand

Program Memory



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?	physical page
•••	•••	•••
0x7FFFB	0	
0x7FFFC	1	0x200DF
0x7FFFD	1	0x12340
0x7FFFE	1	0x12347
0x7FFFF	1	0x12345
•••	•••	•••

%rsp = 0x7FFFC000

```
...

// requires more stack space

A: pushq %rbx page fault!

B: movq 8(%rcx), %rbx

C: addq %rbx, %rax

...

VPN valid? physical valid? page

...

0x7FFFB
0x7FFFC
1 0x200DF
0x7FFFD
1 0x12340
0x7FFFE
0x7FFFE
1 0x12347
0x7FFFF
1 0x12345
...
```

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

%rsp = 0x7FFFC000

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

B: movq 8(%rcx), %rbx
C: addq %rbx, %rax
... physical valid? page
... ... page
... physical valid? page
... ... 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

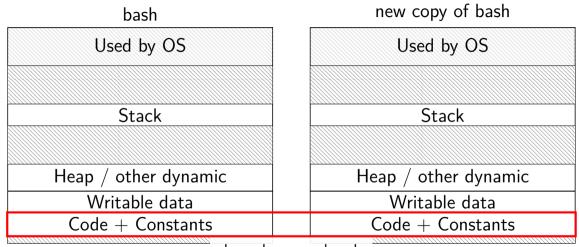
only change: load from file, etc. instead of allocating empty page

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

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?



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 be 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

•••

... 0x00601 0x00602 0x00603 0x00604 0x00605 valid? write?

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

physical

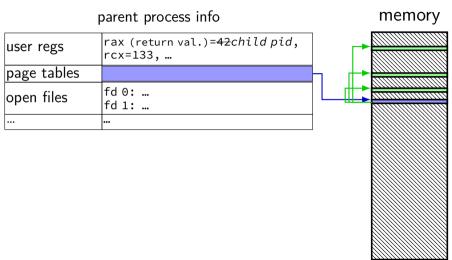
copy operation actually duplicates page table both processes share all physical pages but marks pages in both copies as read-only

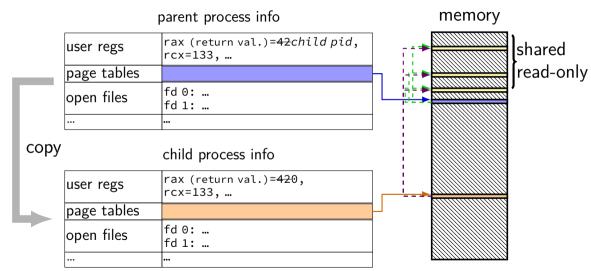
VPN	valid? write?			VPN	valid? write?		
VIIV	page page		page		valiu:	wille:	page
•••	•••	•••	•••	•••	•••	•••	•••
0x00601	1	0	0x12345	0x00601	1	0	0x12345
0x00602	1	0	0x12347	0x00602	1	0	0x12347
0x00603	1	0	0x12340	0x00603	1	0	0x12340
0x00604	1	0	0x200DF	<u>0x00604</u>	1	0	0x200DF
0x00605	1	0	0x200AF	0x00605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

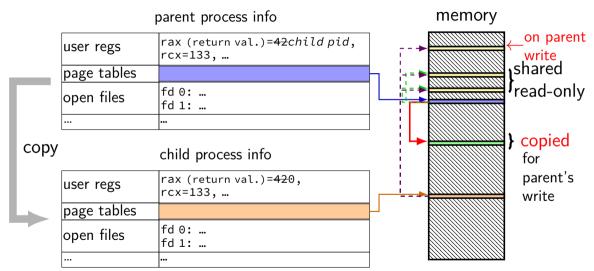
when either process tries to write read-only page triggers a fault — OS actually copies the page

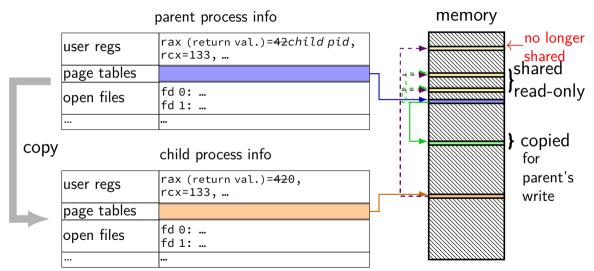
VPN	valid? write?			VPN	valid? write?		
VIIN	valid! Write!		page	VIIN	valiu: write:		page
•••	•••	•••	•••	•••	•••	•••	•••
0x00601	1	0	0x12345	0x00601	1	0	0x12345
0x00602	1	0	0x12347	0×00602	1	0	0x12347
0x00603	1	0	0x12340	0x00603	1	0	0x12340
0x00604	1	0	0x200DF	0x00604	1	0	0x200DF
0x00605	1	0	0x200AF	0x00605	1	1	0x300FD
•••	•••	•••	•••	•••	•••	•••	•••

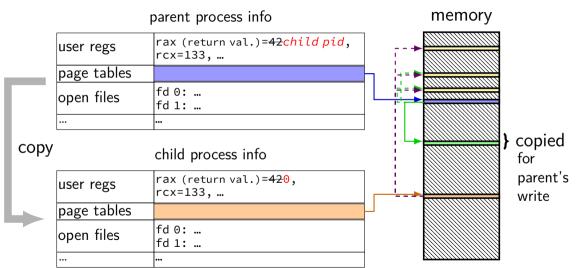
after allocating a copy, OS reruns the write instruction



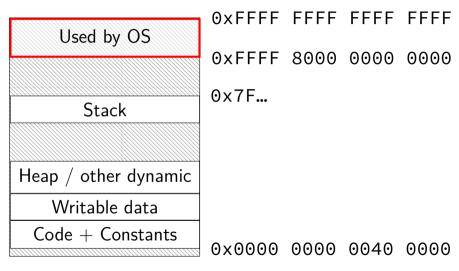








program memory



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

mmap

```
Linux/Unix has a function to "map" a file to memory
int file = open("somefile.dat", 0 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.
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.
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
                                                         vvarl
7ffc5d3b3000-7ffc5d3b5000 r-xp 00000000 00:00 0
                                                         vdsol
fffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0
                                                         [vsvscall]
```

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
                                                        /bin/cat
0060b000—
          OS tracks list of struct vm area struct with:
01974000 -
7f60c718b0
                                                                         cale—archive
          (shown in this output):
7f60c74900
                                                                         gnu/libc-2.1
             virtual address start, end
7f60c764e0
                                                                         gnu/libc-2.1
7f60c784e0
                                                                         gnu/libc-2.1
             permissions
7f60c78520
                                                                         gnu/libc-2.1
7f60c78540
             offset in backing file (if any)
7f60c78590
                                                                         gnu/ld-2.19.s
7f60c7a390
             pointer to backing file (if any)
7f60c7a7a0
7f60c7a7b0
                                                                         gnu/ld-2.19.
7f60c7a7c0
                                                                         gnu/ld-2.19.
          (not shown):
7f60c7a7d0
7ffc5d2b20
             info about sharing of non-file data
7ffc5d3b00
7ffc5d3b30
fffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsvscall]
```

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 0x7FFFFFFFF 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

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

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

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

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

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 #	valid!	page #
00	1	010
01	1	111
10	0	000
11	1	000

physical addresses	hvt	25		
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	1В	2B	3B	4B
0x18-B	1C	2C	3C	4C
0×1C-F	1C	2C	3C	4C

physical addresses	byt	es		
0x20-3	D0	D1	D2	D3
0x24-7	D4	D5	D6	D7
0x28-B 0x2C-F	89	9A	ΑB	ВС
0x2C-F	С	DE	EF	F0
0x30-3	ВА	0Α	BA	0A
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 #	valid!	page #
00	1	010
01	1	111
10	0	000
11	1	000

physical addresses	byt	es		
0x00-3			22	33
0x04-7	44	55	66	77
0x08-B	88	99	ΑА	ВВ
0x0C-F				
0x10-3				
0×14-7	1В	2B	3B	4B
0x18-B	1C	2C	3C	4C
0x1C-F	1C	2C	3C	4C

physical bytes addresses phys. page 1 0x30-3BA 0A BA 0A 0x34-7CB 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

```
page # valid? _
    00
            1010
    01
            111
    10
         0
            000
            000
```

addresses		pnysical bytes addresses	
0x00-300 1	L 22 33	0x20-3D0 D1 D	2 D3
0x04-7 <mark>44 5</mark> 5	66 77	0x24-7D4 D5 D	6 D7
0x08-B88 99	AA BB	0x28-B <mark>89 9A A</mark>	в вс
0×0C-F CC DI	EE FF	0x2C-FCD DE E	F F0
0x10-3 1A 2A	3A 4A	0x30-3BA 0A B	A 0 A
0x14-7 1B 2E	3 3B 4B	0x34-7 CB 0B C	B 0B
0x18-B1C 20	3C 4C	0x38-BDC 0C D	C 0C
0x1C-F1C 20	3C 4C	0x3C-FEC 0C E	C OC

physical bytes

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

```
(virtual addresses) 0x18 = 00; 0x03 = ???; 0x0A = ???; 0x13 = ???
```

page table

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

addresses
0x00-3 <mark>00</mark> 11 22 33
0x04-744 55 66 77
0×08-B88 99 AA BB
0×0C-FCC DD EE FF
0×10-3 1A 2A 3A 4A
0×14-7 1B 2B 3B 4B
0×18-B 1C 2C 3C 4C
$0 \times 10 = 10 20 30 40$

physical bytes

0x20-3	D0	D1	D2	D3
0x24-7	D4	D5	D6	D7
0x28-B	89	9A	AB	ВС
0x2C-F	CD	DE	EF	F0
0x30-3	ВА	0A	ВА	0A
0x34-7	СВ	0B	СВ	0B
0x38-B	DC	0C	DC	0C

0x3C-FEC 0C EC 0C

physical bytes addresses____

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

(virtual addresses) 0x18 = 00; 0x03 = 0x4A; 0x0A = ???; 0x13 = ???

physical bytes

page table

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

addresses	Dy les
	00 11 22 33
0x04-7	14 55 66 77
0x08-B	38 99 AA BB
0x0C-F	CC DD EE FF
0x10-31	LA 2A 3A <mark>4A</mark>
0×14-7	LB 2B 3B 4B
0x18-B1	LC 2C 3C 4C
0×1C-E	1C 2C 3C 4C

addicasca				
0x20-3	<u>0</u>	D1	D2	D3
0x24-7	D4	D5	D6	D7
0x28-B	89	9A	ΑB	ВС
0x2C-F	D	DE	EF	F0
0x30-3	ВА	0A	ВА	0A
0x34-7	СВ	0B	СВ	0B
0x38-B	DC	0C	DC	0C

0x3C-FEC 0C EC 0C

physical bytes

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

(virtual addresses) 0x18 = 00; 0x03 = 0x4A; 0x0A = 0xDC; 0x13 = ???

page table

```
virtual page # valid? physical page # 00 1 010 010 01 111 110 0 000 11 1 000
```

physical bytes addresses____ $0 \times 00 - 3 \mid 00 \mid 11 \mid 22 \mid 33 \mid$ $0 \times 04 - 7 | 44 55 66 77$ 0x08-B|88 99 AA BB 0x0C-FCC DD EE FF 0x10-3 1A 2A 3A 4A $0 \times 14 - 7 | 1B 2B 3B 4B$ 0x18-B1C 2C 3C 4C 0x1C-F1C 2C 3C 4C physical bytes addresses_ 0x20-3 D0 D1 D2 D3 0x24-7D4 D5 D6 D7 0x28-B|89 9A AB BC 0x2C-FCD DE EF F0 0x30-3BA 0A BA 0A 0x34-7CB 0B CB 0B 0x38-BDC 0C DC 0C 0x3C-FIEC 0C EC 0C

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

(virtual addresses) 0x18 = 00; 0x03 = 0x4A; 0x0A = 0xDC; 0x13 = fault page table

physical butos

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

Dytes
addresses
0x00-3 00 11 22 33
0×04-744 55 66 77
0x08-B88 99 AA BB
0x0C-FCC DD EE FF
0×10-3 1A 2A 3A 4A
0×14-7 1B 2B 3B 4B
0x18-B 1C 2C 3C 4C
0x1C-F1C 2C 3C 4C

0x20-3	<u>0</u>	D1	D2	D3
0x24-7	D4	D5	D6	D7
0x28-B	89	9A	ΑB	ВС
0x2C-F	С	DE	EF	F0
0x30-3	ВА	0A	ВА	0A
0x34-7	СВ	0B	СВ	0B
0x38-B	DC	0C	DC	0C
0x3C-F	EC	0C	EC	0C

addresses

where can processor store megabytes of page tables? in memory

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 base register

0x00010000

where can processor store megabytes of page tables? in memory

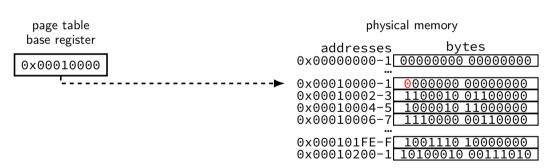
page table entry layout (chosen by processor)

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

physical memory page table base register bvtes addresses 0×00010000 $0 \times 00010000 - 1$ 0000000 00000000 $0 \times 00010002 - 3$ 0x00010004-5 1000010 $0 \times 00010006 - 7$ 0x000101FE-F 1001110 0x00010200-1 10100010 001

where can processor store megabytes of page tables? in memory

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)

where can processor store megabytes of page tables? in memory

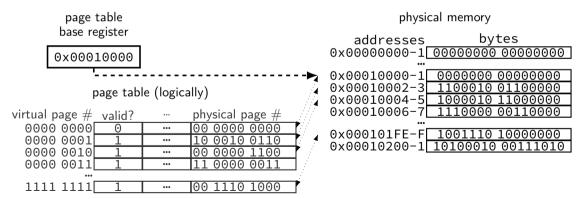
page table entry layout (chosen by processor)

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

physical memory page table base register addresses bvtes 0×00010000 $0 \times 00010000 - 1$ 0000000 00000000 $0 \times 00010002 - 3$ 0x00010004-5 1000010 $0 \times 00010006 - 7$ 0x000101FE-F 1001110 0x00010200-1 10100010 001

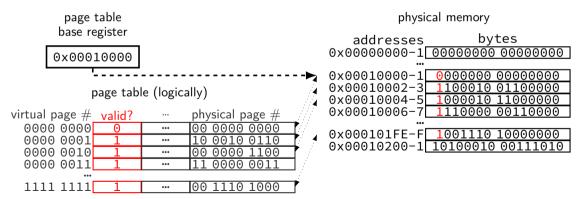
where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)



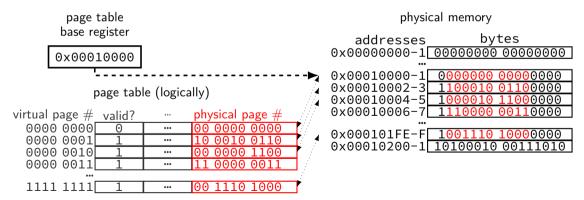
where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)



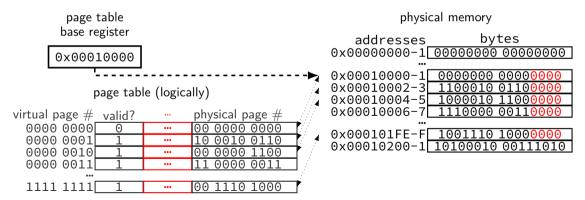
where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)



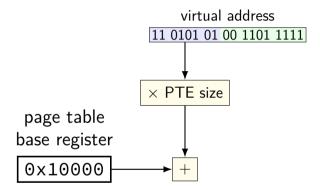
where can processor store megabytes of page tables? in memory

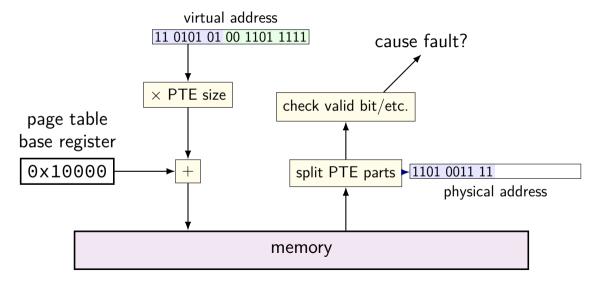
page table entry layout (chosen by processor)

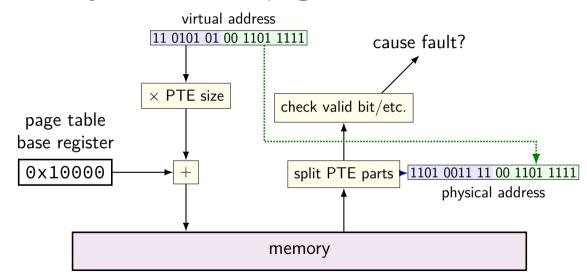


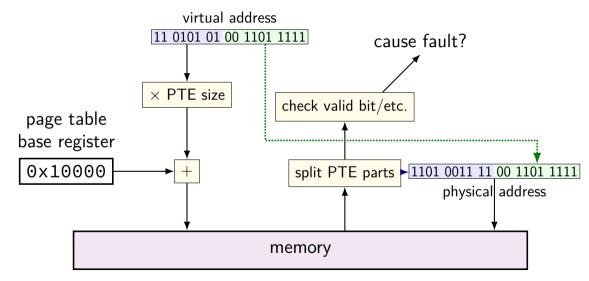
virtual address

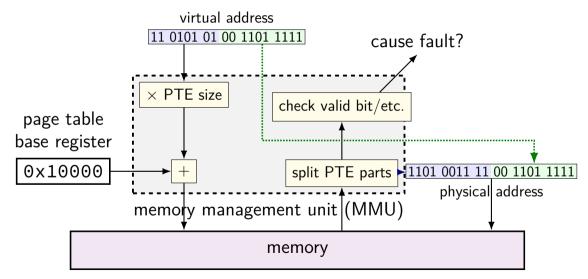
11 0101 01 00 1101 1111

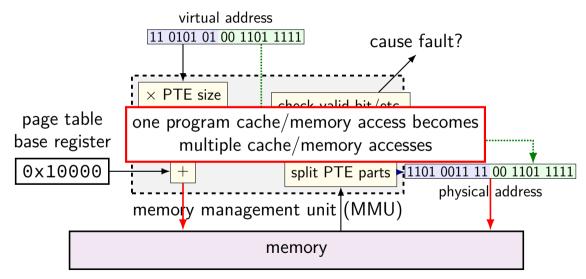


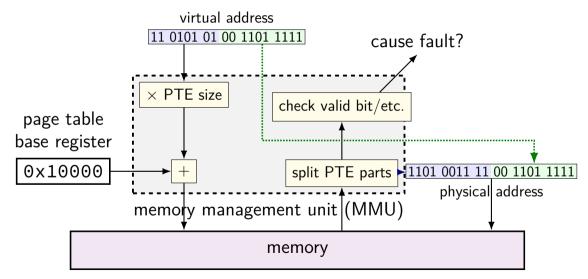












6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other; page table base register 0x20; translate virtual address 0x31

physical addresses	byte	es			physical addresses	byt	es		
0x00-3	00	11	22	33	0x20-3	D0	D1	D2	D3
0x04-7	44	55	66	77	0x24-7	E4	E5	F6	07
0x08-B	88	99	AA	ВВ	0x28-B	89	9A	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;

```
page table base register 0x20; translate virtual address 0x31
   nhysical.
                          nhysical.
```

addresses	addresses	0x31 = 11 0001
0x00-300 11 22 33		<i>□ PTE addr:</i>
0×04-744 55 66 77		$\frac{1}{1} 0 \times 20 + \frac{110}{1} \times 1 = \frac{1}{1}$

			0x28-B	0×08-B88 99 AA BB
☐ ₀	EF F0	CD DE	0x2C-F	0x0C-FCC DD EE FF
7 6	BA 0A	BA 0A	0x30-3	0×10-3 1A 2A 3A 4A

PTF value

 $0 \times F6 = 1111 \ 0110$ PPN 111, valid 1 0x34-7 CB 0B CB 0B 0x14-7|1B 2B 3B 4B $M[111 \ 001] = M[0x39]$ 0x38-BDC 0C DC 0C 0x18-Bl1C 2C 3C 4C \rightarrow 0x0C 0x1C-F|1C 2C 3C 4C 0x3C-FIEC 0C EC 0C

36

0x26

0x04-7|44 55 66 77

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 bytes
                                               0 \times 31 = 11 \ 0001
  addresses
                                               PTE addr:
  0x00-300 11 22 33
                          0x20-3 D0 D1 D2 D3
                                               0x20 + 110 \times 1 = 0x26
```

0x24-7|E4 E5 F6 07

0x28-B|89 9A AB BC 0x08-B|88 99 AA BB 0x0C-FCC DD EE FF 0x2C-FCD DE EF F0 0x10-3|1A 2A 3A 4A

0x30-3|BA 0A BA 0A 0x34-7CB 0B CB 0B 0x14-7|1B 2B 3B 4B0x18-Bl1C 2C 3C 4C 0x38-BlDC 0C DC 0C

PTE value: 0xF6 = 1111 0110PPN 111, valid 1

 $M[111 \ 001] = M[0x39]$

 \rightarrow 0x0C 0x3C-FIEC 0C EC 0C 0x1C-F|1C 2C 3C 4C 36

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 tables 1 page, 1 TE. 3 bit 1 TN (MSB), 1 valid bit, 4 other,

 0x00-3
 00 11 22 33
 0x20-3
 00 D1 D2 D3
 0x20+110 x1 = 0x26

 0x08-B88 99 AA BB
 0x28-B89 9A AB BC
 0x28-B89 9A AB BC
 0x20+110 x1 = 0x26

 0x0C-FCC DD EE FF
 0x2C-FCD DE EF F0
 0xF6 = 1111 0110

 0x30-3 BA 0A BA 0A
 0x30-3 BA 0A BA 0A
 0xF6 = 1111 0110

36

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 bytes
                                           0x31 = 11 0001
addresses
                                           PTE addr:
0x00-300 11 22 33
                      0x20-3 D0 D1 D2 D3
```

0x0C-FCC DD EE FF 0x2C-FCD DE EF F0 0x10-3|1A 2A 3A 4A 0x30-3|BA 0A BA 0A

0x34-7CB 0B CB 0B

 $0x20 + 110 \times 1 = 0x26$ 0x04-7|44 55 66 77 0x24-7E4 E5 F6 07 0x28-B|89 9A AB BC 0x08-B|88 99 AA BB PTE value: 0xF6 = 1111 0110

PPN 111, valid 1 0x14-7|1B 2B 3B 4B $M[111 \ 001] = M[0x39]$ 0x18-Bl1C 2C 3C 4C 0x38-BlDC 0C DC 0C \rightarrow 0x0C 0x3C-FIEC 0C EC 0C 0x1C-F|1C 2C 3C 4C

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
0 \times 00 - 3 | 00 \ 11 \ 22 \ 33
                         0x20-3 A0 E2 D1 F3
0x04-7|44 55 66 77
                         0x24-7E4 E5 F6 07
                         0x28-Bl89 9A AB BC
0x08-Bl88 99 AA BB
0x0C-FCC DD EE FF
                         0x2C-FCD DE EF F0
                         0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
                         0x34-7CB 0B CB 0B
0 \times 14 - 7 | 1B 2B 3B 4B
0x18-Bl1C 2C 3C 4C
                         0x38-BlDC 0C DC 0C
0x1C-F|1C 2C 3C 4C
                         0x3C-FIEC 0C EC 0C
```

0x1C-F|1C 2C 3C 4C

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
                                            0 \times 12 = 01 \quad 0010
addresses
                                            PTE addr:
0x00-300 11 22 33
                       0x20-3 A0 E2 D1 F3
                                            0x20 + 2 \times 1 = 0x22
0x04-7|44 55 66 77
                       0x24-7E4 E5 F6 07
                       0x28-B|89 9A AB BC
0x08-B|88 99 AA BB
                                           PTE value:
0x0C-FCC DD EE FF
                       0x2C-FCD DE EF F0
                                            0 \times D1 = 1101 \ 0001
0x10-3|1A 2A 3A 4A
                       0x30-3|BA 0A BA 0A
                                            PPN 110, valid 1
                       0x34-7CB 0B CB 0B
0x14-7|1B 2B 3B 4B
                                           M[110 \ 010] = M[0x32]
0x18-Bl1C 2C 3C 4C
                       0x38-BlDC 0C DC 0C
```

0x3C-FIEC 0C EC 0C

 \rightarrow 0xBA

0x1C-F|1C 2C 3C 4C

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 tables 1 page, FTE. 3 bit FFN (M3B), 1 valid bit, 4 other page table base register 0x20; translate virtual address 0x12

```
physical bytes
                                           0 \times 12 = 01 \ 0010
addresses
                                           PTE addr:
0x00-300 11 22 33
                      0x20-3 A0 E2 D1 F3
                                           0x20 + 2 \times 1 = 0x22
0x04-7|44 55 66 77
                      0x24-7E4 E5 F6 07
                      0x28-B|89 9A AB BC
0x08-B|88 99 AA BB
                                           PTE value:
0x0C-FCC DD EE FF
                      0x2C-FCD DE EF F0
                                           0 \times D1 = 1101 \ 0001
0x10-3|1A 2A 3A 4A
                      0x30-3|BA 0A BA 0A
                                           PPN 110, valid 1
                      0x34-7CB 0B CB 0B
0x14-7|1B 2B 3B 4B
                                           M[110 \ 010] = M[0x32]
0x18-Bl1C 2C 3C 4C
                      0x38-BlDC 0C DC 0C
```

0x3C-FIEC 0C EC 0C

 \rightarrow 0xBA

0x1C-F|1C 2C 3C 4C

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
                                           0 \times 12 = 01 \ 0010
addresses
                                           PTE addr:
0x00-300 11 22 33
                      0x20-3 A0 E2 D1 F3
                                           0x20 + 2 \times 1 = 0x22
0x04-7|44 55 66 77
                      0x24-7E4 E5 F6 07
                      0x28-B|89 9A AB BC
0x08-B|88 99 AA BB
                                          PTE value:
0x0C-FCC DD EE FF
                      0x2C-FCD DE EF F0
                                           0xD1 = 1101 0001
0x10-3|1A 2A 3A 4A
                      0x30-3|BA 0A BA 0A
                                           PPN 110, valid 1
                      0x34-7CB 0B CB 0B
0x14-7|1B 2B 3B 4B
                                           M[110 \ 010] = M[0x32]
0x18-Bl1C 2C 3C 4C
                      0x38-BlDC 0C DC 0C
                                           \rightarrow 0xBA
```

0x3C-FIEC 0C EC 0C

0x18-Bl1C 2C 3C 4C

0x1C-F|1C 2C 3C 4C

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
                                               0 \times 12 = 01 \ 0010
  addresses
                                               PTE addr:
  0x00-300 11 22 33
                         0x20-3 A0 E2 D1 F3
                                               0x20 + 2 \times 1 = 0x22
  0x04-7|44 55 66 77
                         0x24-7E4 E5 F6 07
```

0x38-BlDC 0C DC 0C

0x3C-FIEC 0C EC 0C

0x28-B|89 9A AB BC 0x08-B|88 99 AA BB PTE value: 0x0C-FCC DD EE FF 0x2C-FCD DE EF F0 0x10-3|1A 2A 3A 4A 0x30-3|BA 0A BA 0A 0x34-7CB 0B CB 0B 0x14-7|1B 2B 3B 4B

0xD1 = 1101 0001PPN 110, valid 1 $M[110 \ 010] = M[0x32]$ \rightarrow 0xBA

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 ... /* page offset bits */
#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 (~OL) if invalid
size t 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:

```
\begin{array}{c|c} & VPN\ valid?\ physical\\ 0 & 0 & ---\\ ptbr = GetPointerToTable( \begin{array}{c|c} 1 & 0x9999\\ 2 & 0 & ---\\ 3 & 1 & 0x3333\\ \cdots & \cdots & \cdots \end{array} \end{array} )
```

```
translate(0x0FFF) == (\text{void*}) \, ^{\circ}0L

translate(0x1000) == (\text{void*}) \, 0x9999000

translate(0x1001) == (\text{void*}) \, 0x9999001

translate(0x2000) == (\text{void*}) \, ^{\circ}0L

translate(0x2001) == (\text{void*}) \, ^{\circ}0L

translate(0x3000) == (\text{void*}) \, ^{\circ}0L
```

translate()

```
with POBITS=12, LEVELS=1:
```

```
translate(0x0FFF) == (void^*) ~0L

translate(0x1000) == (void^*) 0x9999000

translate(0x1001) == (void^*) 0x9999001

translate(0x2000) == (void^*) ~0L

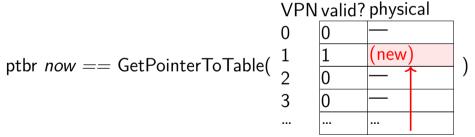
translate(0x2001) == (void^*) ~0L

translate(0x3000) == (void^*) 0x3333000
```

page_allocate()

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

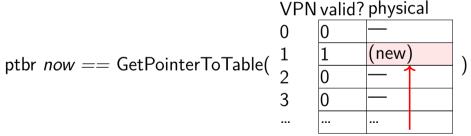
page_allocate()



allocated with posix_memalign

page_allocate()

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



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 (week after break): all LEVELS, both functions
    in preparation for code review
    due Weds BFFORF LAB
part 3 (week after break): final submission
     Friday after code review
    most of grade based on this
    will test previous parts again
```

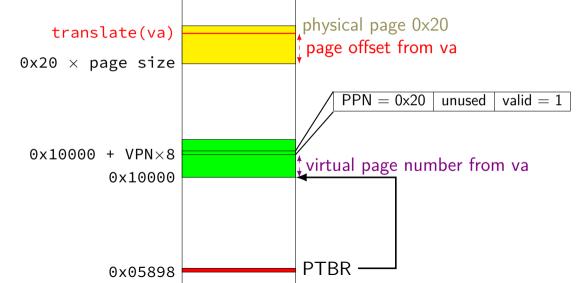
address/page table entry format

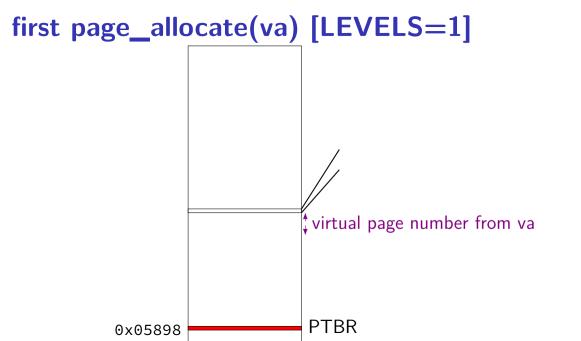
(with POBITS=12, LEVELS=1)

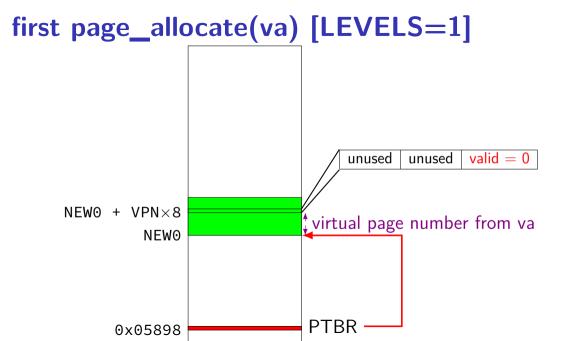
	bits 63–21	bits 20–12	bits 11–1	bit 0
page table entry	physical page number		unused	valid bit
virtual address	unused	virtual page number	page offset	
physical address	physical page number		page offset	

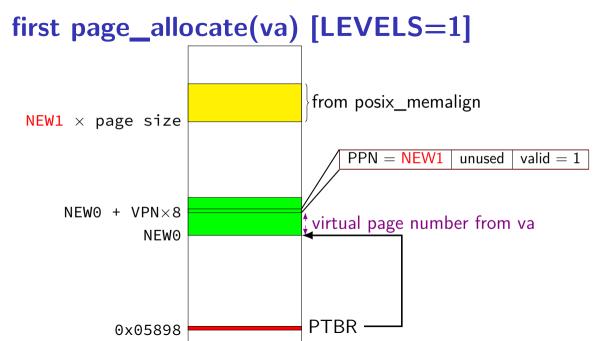
in assignment: value from posix_memalign = physical address

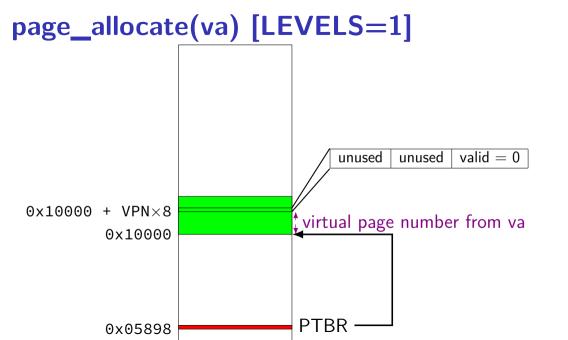
pa = translate(va) [LEVELS=1]

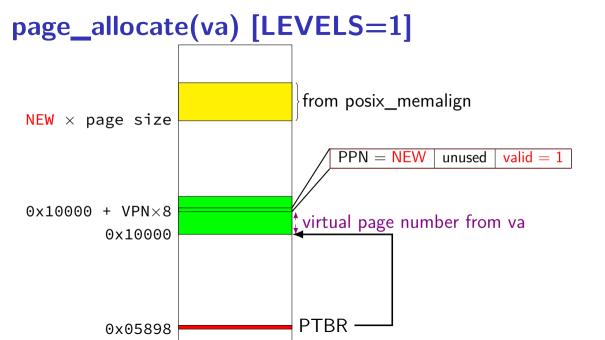




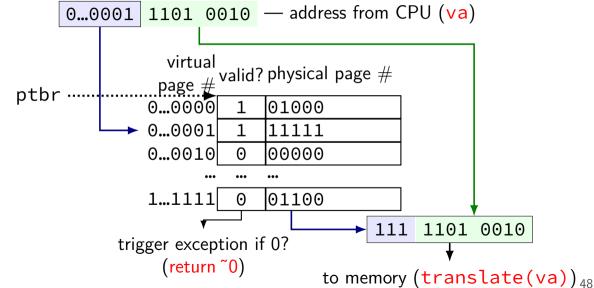




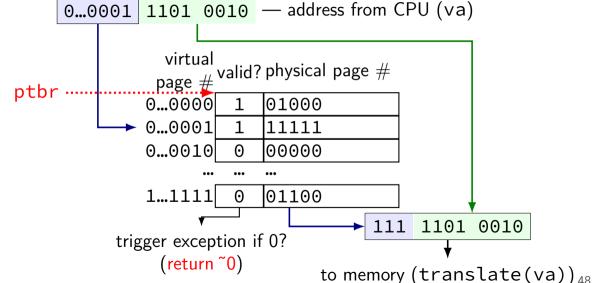




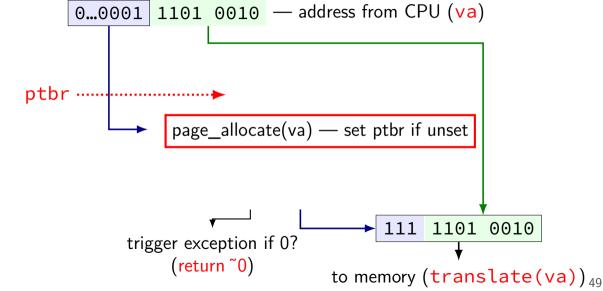
page table lookup (and translate())



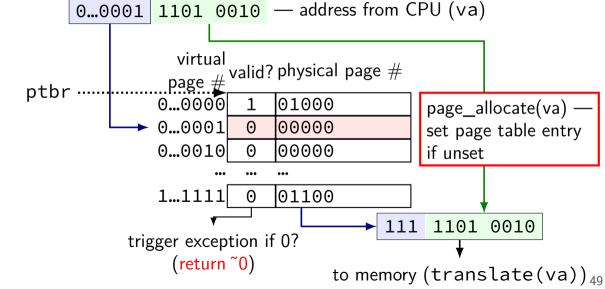
page table lookup (and translate())



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) $2^{48}/2^{12}=2^{36}$ entries

exercise: how large are physical page numbers? 39 - 12 = 27 bits

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) $2^{48}/2^{12}=2^{36}$ entries

exercise: how large are physical page numbers? 39 - 12 = 27 bits 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??)

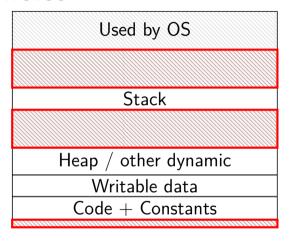
huge page tables

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

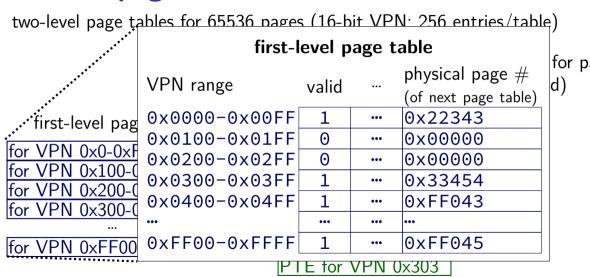
doing two memory accesses is already very slow solution: tree with many children from each node (far from binary tree's left/right child)

F for VPN 0x02 first-level page table F for VPN 0x03 for VPN 0x0-0xFF VPN 0×100-0×1FF PTE for VPN 0xFF for VPN 0x200-0x2FF for VPN 0x300 for VPN 0x300-0x3FF VPN 0x301 for VPN 0xFF00-0xFFFF LONG OFF

55

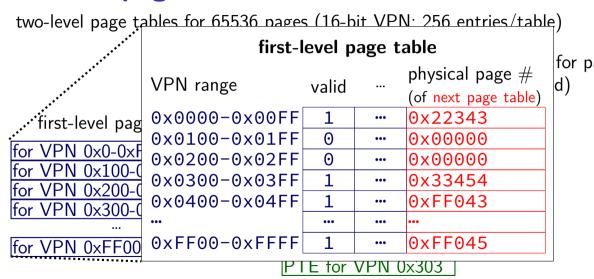
two-level page tables for 65536 pages (16-bit VPN; 256 entries/table) second-level page tables actual data for p PTE for VPN 0x00 (if PTE valid) F for VPN 0x02 first-level page table for VPN 0x0-0xFF

VPN 0x100-0x1FF invalid entries represent big holes VPN 0x300 for VPN 0x300-0x3FF VPN 0x301 for VPN 0xFF00-0xFFFF LONG OFF



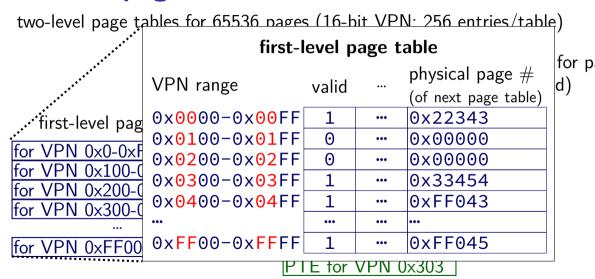
V/DNI A AFE

55

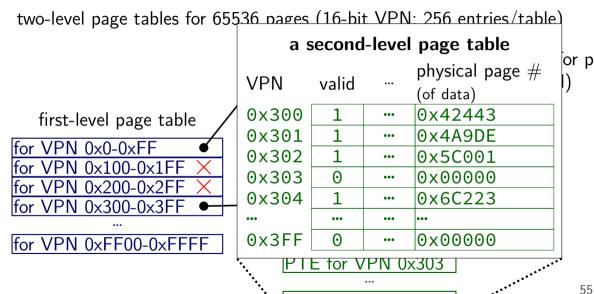


V/DNI A AFE

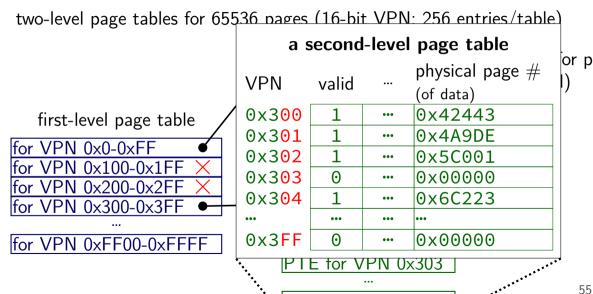
55



V/DNI A AFE

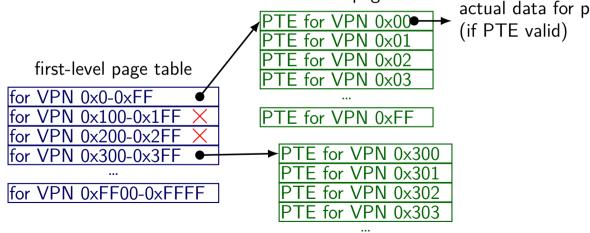


V/DNI A AFE



V/DNI A AFE

two-level page tables for 65536 pages (16-bit VPN; 256 entries/table) second-level page tables



LONG OFF

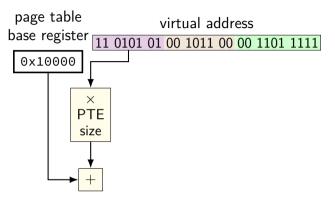
55

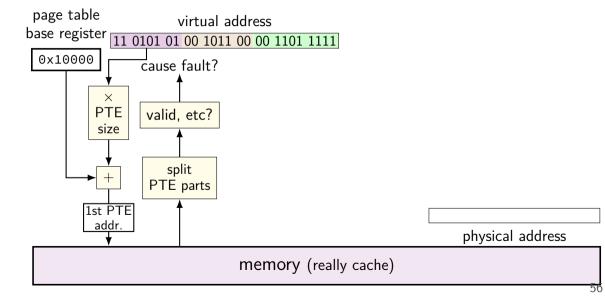
virtual address

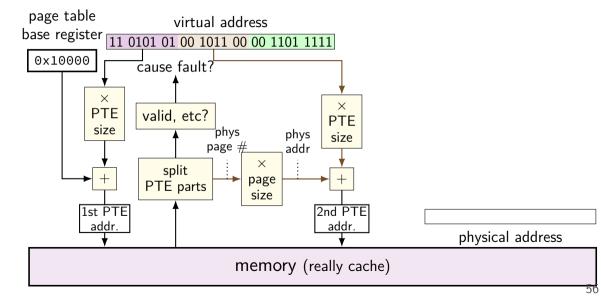
11 0101 01 00 1011 00 00 1101 1111

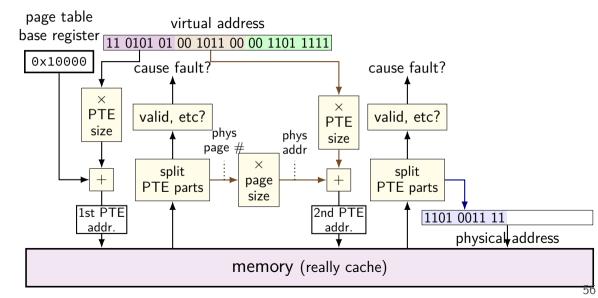
VPN — split into two parts (one per level)

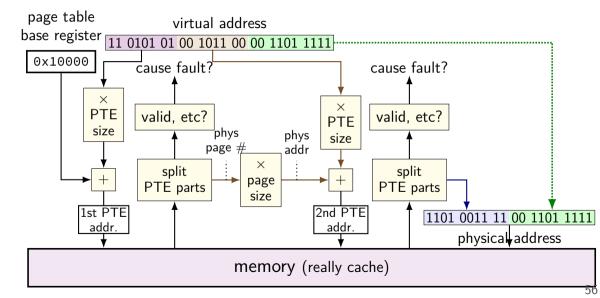
this example: parts equal sized — common, but not required

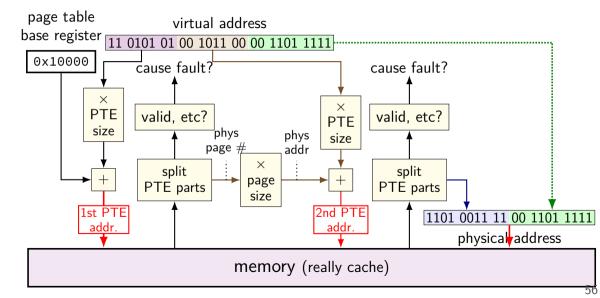


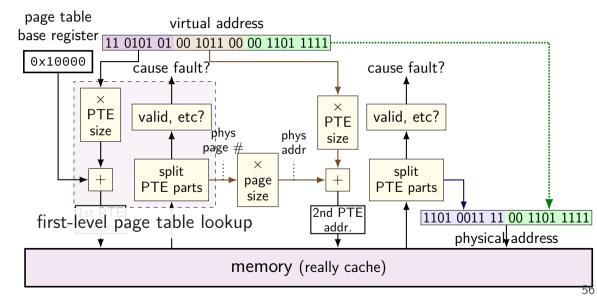


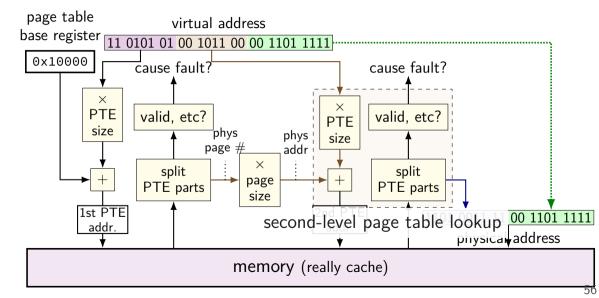


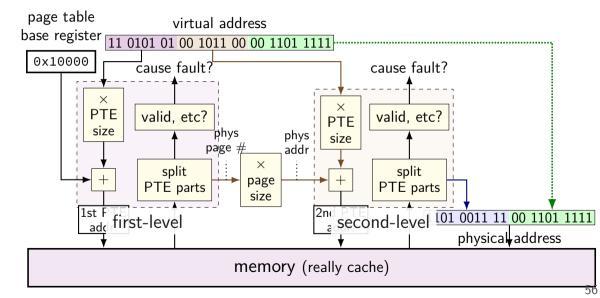


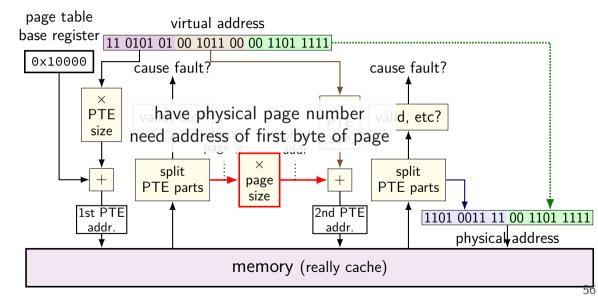


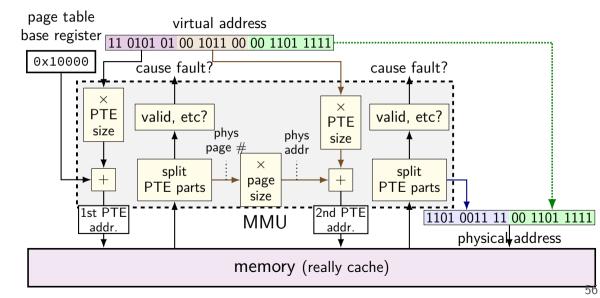




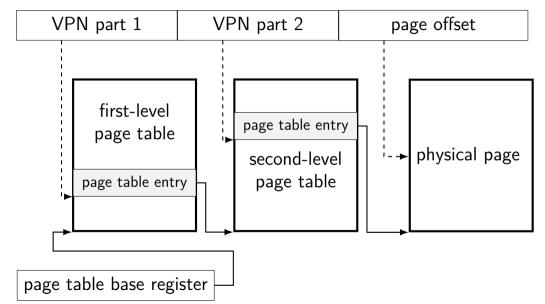








another view



57

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

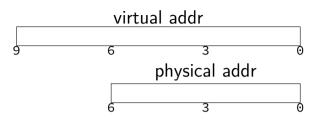
validity checks at each level

note on VPN splitting

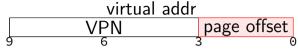
indexes used for lookup parts of the virtual page number (there are not multiple VPNs)

assignment

9-bit virtual address6-bit physical address



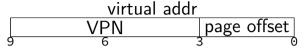
- 9-bit virtual address
- 6-bit physical address
- 8-byte pages \rightarrow 3-bit page offset (bottom) $\frac{1}{6}$
- 9-bit VA: 6 bit VPN + 3 bit PO
- 6-bit PA: 3 bit PPN + 3 bit PO

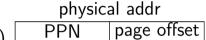


physical addr

PPN page offset

- 9-bit virtual address
- 6-bit physical address
- 8-byte pages \rightarrow 3-bit page offset (bottom)
- 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

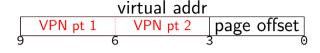


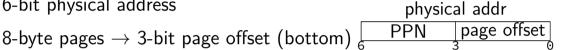




- page table (either level)
 - valid? PPN

- 9-bit virtual address
- 6-bit physical address
- 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
- 8 entry page tables \rightarrow 3-bit VPN parts
- 9-bit VA: 3 bit VPN part 1; 3 bit VPN part 2





page table (either level)

valid? PPN

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 0x20; translate virtual address 0x129

physical baddresses_		physical bytes addresses							
0×00-3		22 3	33		20-3			72	13
$0 \times 04 - 7$	14 55	66 7	77	0 x	24-7	F4	Α5	36	07
0x08-B	38 99	AA E	3B	0 x	28-B	89	9A	ΑB	ВС
0x0C-F	CC DD	EE F	FF	0 x	2C-F	CD	DE	EF	F0
0x10-3	LA 2A	3A 4	4A	0 x	30-3	ВА	0Α	ВА	0A
0x14-7	LB 2B	3B 4	4B	0 x	34-7	DΒ	0B	DB	0B
0x18-B1	LC 2C	3C 4	4C	0 x	38-B	EC	0C	EC	0C
0x1C-F	LC 2C	3C 4	4C	0 x	3C-F	AC	DC	DC	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 0x20; translate virtual address 0x129

```
physical bytes
                                                0x129 = 1 0010 1001
addresses
                                                0x20 + 0x4 \times 1 = 0x24
0 \times 00 - 3 | 00 \ 11 \ 22 \ 33
                         0x20-3|00 91 72 13
                                                PTE 1 value:
0 \times 04 - 7 | 44 55 66 77
                         0x24-7|_{F4} A5 36 07
                                                0xF4 = 1111 0100
                         0x28-Bl89 9A AB BC
0x08-Bl88 99 AA BB
                                                PPN 111. valid 1
0x0C-FCC DD EE FF
                         0x2C-FCD DE EF F0
                         0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
0x14-7|1B 2B 3B 4B
                         0x34-7DB 0B DB 0B
0x18-Bl1C 2C 3C 4C
                         0x38-BIEC 0C EC 0C
0x1C-F|1C 2C 3C 4C
                         0x3C-FIAC DC DC 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 0x20; translate virtual address 0x129

```
physical bytes
                                                  0 \times 129 = 1 \ 0010 \ 1001
addresses
                                                  0x20 + 0x4 \times 1 = 0x24
0 \times 00 - 3 | 00 \ 11 \ 22 \ 33
                          0x20-3|00 91 72 13
                                                  PTE 1 value:
                          0x24-7|F4 A5 36 07
0 \times 04 - 7 | 44 55 66 77
                                                  0xF4 = 1111 0100
                          0x28-Bl89 9A AB BC
0x08-Bl88 99 AA BB
                                                  PPN 111. valid 1
0x0C-FCC DD EE FF
                          0x2C-FCD DE EF F0
                                                  PTE 2 addr:
                          0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
                                                   111 \ 000 + 101 \times 1 = 0 \times 3D
                          0x34-7DB 0B DB 0B
0 \times 14 - 7 | 1B 2B 3B 4B
                                                   PTE 2 value: 0xDC
0x18-Bl1C 2C 3C 4C
                          0x38-BIEC 0C EC 0C
0x1C-F|1C 2C 3C 4C
                          0x3C-FIAC DC DC 0C
```

0x1C-F|1C 2C 3C 4C

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 tables I page, I IL. 3 bit I IN (IVISB), I valid bit, 4 dilused

```
page table base register 0x20; translate virtual address 0x129
   physical bytes
                               physical <sub>bytes</sub>
                                                         0 \times 129 = 1 \ 0010 \ 1001
  addresses
                              addresses
                                                         0x20 + 0x4 \times 1 = 0x24
   0x00-3|00 11 22 33
                               0 \times 20 - 3 \mid 00 \ 91 \ 72 \ 13
                                                         PTE 1 value:
   0 \times 04 - 7 | 44 55 66 77
                               0x24-7|F4 A5 36 07
                                                         0 \times F4 = 1111 \ 0100
                               0x28-B|89 9A AB BC
   0x08-Bl88 99 AA BB
                                                         PPN 111. valid 1
   0x0C-FCC DD EE FF
                               0x2C-FCD DE EF F0
                                                         PTE 2 addr:
   0 \times 10 - 3 | 1A 2A 3A 4A
                               0x30-3|BA 0A BA 0A
                                                         111 \ 000 + 101 \times 1 = 0x3D
                                                         PTE 2 value: 0xDC
   0 \times 14 - 7 | 1B 2B 3B 4B
                               0 \times 34 - 7 \mid DB \mid 0B \mid DB \mid 0B
                              0x38-BIEC 0C EC 0C
                                                         PPN 110; valid 1
   0x18-Bl1C 2C 3C 4C
```

0x3C-FAC DC DC 0C

 $M[110 \ 001 \ (0x31)] = 0x0A_{62}$

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 0x20; translate virtual address 0x129

```
physical bytes
                             physical <sub>bytes</sub>
                                                       0 \times 129 = 1 \ 0010 \ 1001
addresses
                            addresses
                                                       0x20 + 0x4 \times 1 = 0x24
0x00-3|00 11 22 33
                            0 \times 20 - 3 \mid 00 \ 91 \ 72 \ 13
                                                       PTE 1 value:
0 \times 04 - 7 | 44 55 66 77
                            0x24-7|F4 A5 36 07
                                                       0 \times F4 = 1111 \ 0100
                            0x28-B|89 9A AB BC
0x08-Bl88 99 AA BB
                                                       PPN 111. valid 1
0x0C-FCC DD EE FF
                            0x2C-FCD DE EF F0
                                                       PTE 2 addr:
0 \times 10 - 3 | 1A 2A 3A 4A
                            0x30-3|BA 0A BA 0A
                                                       111 \ 000 + 101 \times 1 = 0 \times 3D
                                                       PTE 2 value: 0xDC
0 \times 14 - 7 | 1B 2B 3B 4B
                            0 \times 34 - 7 \mid DB \mid 0B \mid DB \mid 0B
                            0x38-BIEC 0C EC 0C
                                                       PPN 110; valid 1
0x18-Bl1C 2C 3C 4C
                                                       M[110 \ 001 \ (0x31)] = 0x0A
0x1C-F|1C 2C 3C 4C
                            0x3C-FAC DC DC 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 0x20; translate virtual address 0x129

```
physical bytes
                            physical <sub>bytes</sub>
                                                     0x129 = 1 0010 1001
addresses
                           addresses
                                                     0x20 + 0x4 \times 1 = 0x24
0x00-3|00 11 22 33
                           0 \times 20 - 3 | 00 91 72 13
                                                     PTE 1 value:
0 \times 04 - 7 | 44 55 66 77
                           0x24-7|F4 A5 36 07
                                                     0 \times F4 = 1111 \ 0100
                           0x28-B|89 9A AB BC
0x08-Bl88 99 AA BB
                                                     PPN 111. valid 1
0x0C-FCC DD EE FF
                           0x2C-FCD DE EF F0
                                                     PTE 2 addr:
0 \times 10 - 3 | 1A 2A 3A 4A
                           0x30-3|BA 0A BA 0A
                                                     111 000 + 101 \times 1 = 0 \times 3D
                                                     PTE 2 value: 0xDC
0 \times 14 - 7 | 1B 2B 3B 4B
                           0 \times 34 - 7 \mid DB \mid 0B \mid DB \mid 0B
                           0x38-BIEC 0C EC 0C
                                                     PPN 110; valid 1
0x18-Bl1C 2C 3C 4C
                                                     M[110 \ 001 \ (0x31)] = 0x0A
0x1C-F|1C 2C 3C 4C
                           0x3C-FAC DC DC 0C
```

2-level exercise (1)

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 bytes addresses					physical bytes addresses					
addresses						addresses	S			
0x00-3			22	33		0x20-3			D2	D3
0x04-7	44	55	66	77		0x24-7	7D4	D5	D6	D7
0x08-B	88	99	AA	ВВ		0x28-E	89	9A	AB	ВС
0x0C-F	CC	DD	EE	FF		0x2C-F	CD	DE	EF	F0
0x10-3	1A	2A	ЗА	4A		0x30-3	BA	0A	ВА	0A
0x14-7	1B	2B	3B	4B		0x34-7	DB	0B	DB	0B
0x18-B	1C	2C	3C	4C		0x38-E	BEC	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;

```
physical bytes
                         physical <sub>bytes</sub>
addresses
                                                0 \times 0 = 011 \ 111 \ 011
0x00-3|00 11 22 33
                         0x20-3|D0 D1 D2 D3
                                                (PTE 1 addr: 0x08 +
                         0x24-7D4 D5 D6 D7
0x04-7|44 55 66 77
                                                PTE size times 011 (3))
0x08-B|88 99 AA BB
                         0x28-B|89 9A AB BC
                                                PTE 1: 0xBB at 0x0B
0x0C-FCC DD EE FF
                         0x2C-FCD DE EF F0
                                                PTE 1: PPN 101 (5) valid 1
0x10-3|1A 2A 3A 4A
                         0x30-3|BA 0A BA 0A
                                                PTE 2: 0xF0 at 0x2F
                         0 \times 34 - 7 | DB | 0B | DB | 0B
0x14-7|1B 2B 3B 4B
                                                PTE 2: PPN 111 (7) valid 1
0x18-Bl1C 2C 3C 4C
                         0x38-BIEC 0C EC 0C
                                                111 \ 011 = 0x3B \rightarrow 0x0C
0x1C-F|1C 2C 3C 4C
                         0x3C-FIFC 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;

```
physical bytes
                         physical <sub>bytes</sub>
addresses
                                                0 \times 0 = 011 \ 111 \ 011
0x00-3|00 11 22 33
                         0x20-3|D0 D1 D2 D3
                                                (PTE 1 addr: 0x08 +
                         0x24-7D4 D5 D6 D7
0x04-7|44 55 66 77
                                                PTE size times 011 (3))
0x08-B|88 99 AA BB
                         0x28-B|89 9A AB BC
                                                PTE 1: 0xBB at 0x0B
0x0C-FCC DD EE FF
                         0x2C-FCD DE EF F0
                                                PTE 1: PPN 101 (5) valid 1
0x10-3|1A 2A 3A 4A
                         0x30-3|BA 0A BA 0A
                                                PTE 2: 0xF0 at 0x2F
                         0 \times 34 - 7 | DB | 0B | DB | 0B
0x14-7|1B 2B 3B 4B
                                                PTE 2: PPN 111 (7) valid 1
0x18-Bl1C 2C 3C 4C
                         0x38-BIEC 0C EC 0C
                                                111 \ 011 = 0x3B \rightarrow 0x0C
0x1C-F|1C 2C 3C 4C
                         0x3C-FIFC 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;

```
physical bytes
                         physical <sub>bytes</sub>
addresses
                                                0 \times 0 = 011 \ 111 \ 011
0x00-3|00 11 22 33
                         0x20-3|D0 D1 D2 D3
                                                (PTE 1 addr: 0x08 +
                         0x24-7D4 D5 D6 D7
0x04-7|44 55 66 77
                                                PTE size times 011 (3))
0x08-B|88 99 AA BB
                         0x28-B|89 9A AB BC
                                                PTE 1: 0xBB at 0x0B
0x0C-FCC DD EE FF
                         0x2C-FCD DE EF F0
                                                PTE 1: PPN 101 (5) valid 1
0x10-3|1A 2A 3A 4A
                         0x30-3|BA 0A BA 0A
                                                PTE 2: 0xF0 at 0x2F
                         0 \times 34 - 7 | DB | 0B | DB | 0B
0x14-7|1B 2B 3B 4B
                                                PTE 2: PPN 111 (7) valid 1
0x18-Bl1C 2C 3C 4C
                         0x38-BIEC 0C EC 0C
                                                111 \ 011 = 0x3B \rightarrow 0x0C
0x1C-F|1C 2C 3C 4C
                         0x3C-FIFC 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;

```
physical bytes
                          physical <sub>bytes</sub>
addresses
                                                0 \times 0 = 011 \ 111 \ 011
0x00-3|00 11 22 33
                         0x20-3|D0 D1 D2 D3
                                                (PTE 1 addr: 0x08 +
                         0x24-7|D4 D5 D6 D7
0x04-7|44 55 66 77
                                                PTE size times 011 (3))
0x08-B|88 99 AA BB
                         0x28-B|89 9A AB BC
                                                PTE 1: 0xBB at 0x0B
0x0C-FCC DD EE FF
                         0x2C-FCD DE EF F0
                                                PTE 1: PPN 101 (5) valid 1
0x10-3|1A 2A 3A 4A
                         0x30-3|BA 0A BA 0A
                                                PTE 2: 0xF0 at 0x2F
                         0 \times 34 - 7 | DB | 0B | DB | 0B
0x14-7|1B 2B 3B 4B
                                                PTE 2: PPN 111 (7) valid 1
0x18-Bl1C 2C 3C 4C
                         0x38-BIEC 0C EC 0C
                                                111 \ 011 = 0x3B \rightarrow 0x0C
0x1C-F|1C 2C 3C 4C
                         0x3C-FIFC 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 byt addresses	es	physical addresses	bytes
		addresses	
0×00-300	11 22 33	0x20-3	D0 D1 D2 D3
0x04-744	55 66 77	0x24-7	D4 D5 D6 D7
0x08-B88	99 AA BB	0x28-B	89 9A AB BC
0x0C-FCC	DD EE FF	0x2C-F	CD DE EF F0
0×10-31A	2A 5A 4A	0x30-3	BA 0A BA 0A
0x14-71B	2B 3B 4B	0x34-7	DB 0B DB 0B
0x18-B1C	2C 3C 4C	0x38-B	EC 0C EC 0C
0x1C-F1C	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 0 \times 10; translate virtual address 0 \times 109
```

physical bytes physical _{bytes} $0 \times 109 = 100 \ 011 \ 001$ addresses (PTE 1 at:

0x00-3|00 11 22 33 0x20-3|D0 D1 D2 D3 0x10 + PTE size times 4 (100)) 0x04-7|44 55 66 77 0x24-7D4 D5 D6 D7

PTF 1: 0x1B at 0x14 0x08-B|88 99 AA BB 0x28-B|89 9A AB BC PTE 1: PPN 000 (0) valid 1 0x0C-FCC DD EE FF 0x2C-FCD DE EF F0 (second table at:

0x10-3|1A 2A 5A 4A 0x30-3|BA 0A BA 0A 0 (000) times page size = 0×00) $0 \times 14 - 7 \mid 1B \mid 2B \mid 3B \mid 4B \mid$ $0 \times 34 - 7 \mid DB \mid 0B \mid DB \mid 0B$ PTE 2: 0x33 at 0x03 0x18-Bl1C 2C 3C 4C 0x38-BIEC 0C EC 0C PTE 2: PPN 001 (1) valid 1

0x1C-F|1C 2C 3C 4C 0x3C-FIFC 0C FC 0C $001 \ 001 = 0x09 \rightarrow 0x99$

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 bytes 0x109 = 100 011 001
```

physical bytes addresses addresses $0 \times 0 - 3 | 00 \ 11 \ 22 \ 33$ physical bytes $0 \times 109 = 100 \ 011 \ 00 \ 0 \times 109 = 100 \ 011 \ 000 \ 0 \times 1000 = 100 \ 0 \times 1000 = 1000 \ 0 \times$

0x08-B 88 99 AA BB 0x28-B 89 9A AB BC 0x0C-F CC DD EE FF 0x10-3 1A 2A 5A 4A 0x30-3 BA 0A BA 0A 0x 0A 14 (second table at: 0x00)

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 0 \times 10; translate virtual address 0 \times 109
    physical bytes
                                  physical <sub>bytes</sub>
                                                              0 \times 109 = 100 \ 011 \ 001
```

addresses (PTE 1 at:

0x00-3|00 11 22 33 0x20-3|D0 D1 D2 D3 0x10 + PTE size times 4 (100))

0x04-7|44 55 66 77 0x24-7D4 D5 D6 D7 PTF 1: 0x1B at 0x14 0x08-B|88 99 AA BB 0x28-B|89 9A AB BC PTE 1: PPN 000 (0) valid 1

0x0C-FCC DD EE FF 0x2C-FCD DE EF F0 (second table at: 0x10-3|1A 2A 5A 4A 0x30-3|BA 0A BA 0A 0 (000) times page size = 0×00) $0 \times 14 - 7 \mid 1B \mid 2B \mid 3B \mid 4B \mid$ $0 \times 34 - 7 \mid DB \mid 0B \mid DB \mid 0B$ PTF 2: 0x33 at 0x03

0x18-Bl1C 2C 3C 4C 0x38-BIEC 0C EC 0C PTE 2: PPN 001 (1) valid 1 0x1C-F|1C 2C 3C 4C 0x3C-FIFC 0C FC 0C $001 \ 001 = 0x09 \rightarrow 0x99$

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 0 \times 10; translate virtual address 0 \times 109

physical bytes

physical bytes

ox 109 = 100 \ 011 \ 001
```

physical bytes addresses bytes $0 \times 109 = 100 \ 011 \ 010 \ 011 \$

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 bytes addresses $0 \times 00 - 3 | 00 \ 11 \ 22 \ 33$ 0x20-3|D0 D1 D2 D3 $0 \times 04 - 7 | 44 55 66 77$ 0x24-7D4 D5 D6 D7 0x28-Bl89 9A AB BC 0x08-Bl88 99 AA BB 0x0C-FCC DD EE FF 0x2C-FCD DE EF F0 0x30-3|BA 0A BA 0A $0 \times 10 - 3 | 1A 2A 3A 4A$ 0x34-7DB 0B DB 0B $0 \times 14 - 7 | 1B 2B 3B 4B$ 0x18-Bl1C 2C 3C 4C 0x38-BIEC 0C EC 0C 0x3C-F|FC 0C FC 0C 0x1C-F|1C 2C 3C 4C

65

0x1C-F|1C 2C 3C 4C

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 0×08 ; translate virtual address $0 \times 00B$ physical bytes addresses 0x00-3|00 11 22 33 0x20-3 D0 D1 D2 D3 0x04-7|44 55 66 77 0x24-7D4 D5 D6 D7 $0 \times 0 = 000 001 011$ 0x08-Bl88 99 AA BB 0x28-Bl89 9A AB BC PTE 1: 0x88 at 0x08 0x0C-FCC DD EE FF 0x2C-FCD DE EF F0 PTE 1: PPN 100 (5) valid 0 0x30-3|BA 0A BA 0A $0 \times 10 - 3 | 1A 2A 3A 4A$ page fault! 0x34-7DB 0B DB 0B $0 \times 14 - 7 | 1B 2B 3B 4B$ 0x18-Bl1C 2C 3C 4C 0x38-BIEC 0C EC 0C

0x3C-FIFC 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 bytes
addresses
0x00-3|00 11 22 33
                        0x20-3 D0 D1 D2 D3
0x04-7|44 55 66 77
                        0x24-7D4 D5 D6 D7
                                              0 \times 0 = 000 001 011
0x08-B|88 99 AA BB
                        0x28-Bl89 9A AB BC
                                              PTE 1: 0x88 at 0x08
0x0C-FCC DD EE FF
                        0x2C-FCD DE EF F0
                                              PTE 1: PPN 100 (5) valid 0
                        0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
                                              page fault!
                        0x34-7DB 0B DB 0B
0 \times 14 - 7 | 1B 2B 3B 4B
0x18-Bl1C 2C 3C 4C
                        0x38-BIEC 0C EC 0C
0x1C-F|1C 2C 3C 4C
                        0x3C-FIFC 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 bytes addresses $0 \times 00 - 3 \mid 00 \ 11 \ 22 \ 33$ 0x20-3|D0 D1 D2 D3 0x04-7|44 55 66 77 0x24-7D4 D5 D6 D7 0x28-Bl89 9A AB BC 0x08-Bl88 99 AA BB 0x0C-FCC DD EE FF 0x2C-FCD DE EF F0 0x30-3|BA 0A BA 0A $0 \times 10 - 3 | 1A 2A 3A 4A$ 0x34-7DB 0B DB 0B $0 \times 14 - 7 | 1B 2B 3B 4B$ 0x18-Bl1C 2C 3C 4C 0x38-BIEC 0C EC 0C 0x1C-F|1C 2C 3C 4C 0x3C-FIFC 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 bytes
                         physical <sub>bytes</sub>
addresses
0x00-3|00 11 22 33
                         0x20-3|D0 D1 D2 D3
                                                0 \times 1 CB = 111 001 011
                         0x24-7D4 D5 D6 D7
0x04-7|44 55 66 77
                                                PTE 1: 0xFF at 0x0F
0x08-B|88 99 AA BB
                         0x28-B|89 9A AB BC
                                                PTE 1: PPN 111 (7) valid 1
0x0C-FCC DD EE FF
                         0x2C-FCD DE EF F0
                                                PTE 2: 0x0C at 0x39
0 \times 10 - 3 | 1A 2A 3A 4A
                         0x30-3|BA 0A BA 0A
                                                PTE 2: PPN 000 (0) valid 0
                         0x34-7DB 0B DB 0B
0 \times 14 - 7 | 1B 2B 3B 4B
                                                page fault!
                         0x38-BIEC 0C EC 0C
0x18-Bl1C 2C 3C 4C
                         0x3C-F|FC 0C FC 0C
0x1C-F|1C 2C 3C 4C
```

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 bytes
                         physical <sub>bytes</sub>
addresses
0x00-3|00 11 22 33
                         0x20-3|D0 D1 D2 D3
                                                0 \times 1 CB = 111 001 011
                         0x24-7D4 D5 D6 D7
0x04-7|44 55 66 77
                                                PTE 1: 0xFF at 0x0F
0x08-B|88 99 AA BB
                         0x28-B|89 9A AB BC
                                                PTE 1: PPN 111 (7) valid 1
0x0C-FCC DD EE FF
                         0x2C-FCD DE EF F0
                                                PTE 2: 0x0C at 0x39
0 \times 10 - 3 | 1A 2A 3A 4A
                         0x30-3|BA 0A BA 0A
                                                PTE 2: PPN 000 (0) valid 0
                         0x34-7DB 0B DB 0B
0 \times 14 - 7 | 1B 2B 3B 4B
                                                page fault!
                         0x38-BIEC 0C EC 0C
0x18-Bl1C 2C 3C 4C
                         0x3C-F|FC 0C FC 0C
0x1C-F|1C 2C 3C 4C
```

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 bytes
                          physical <sub>bytes</sub>
addresses
0x00-3|00 11 22 33
                          0x20-3|D0 D1 D2 D3
                                                 0 \times 1 CB = 111 001 011
                          0x24-7D4 D5 D6 D7
0x04-7|44 55 66 77
                                                 PTE 1: 0xFF at 0x0F
0x08-B|88 99 AA BB
                          0x28-B|89 9A AB BC
                                                 PTE 1: PPN 111 (7) valid 1
0x0C-FCC DD EE FF
                          0x2C-FCD DE EF F0
                                                 PTE 2: 0 \times 0 C at 0 \times 39
0 \times 10 - 3 | 1A 2A 3A 4A
                          0x30-3|BA 0A BA 0A
                                                 PTE 2: PPN 000 (0) valid 0
                          0x34-7DB 0B DB 0B
0 \times 14 - 7 | 1B 2B 3B 4B
                                                 page fault!
                          0x38-BIEC 0C EC 0C
0x18-Bl1C 2C 3C 4C
                          0x3C-F|FC 0C FC 0C
0x1C-F|1C 2C 3C 4C
```

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

0			,	_	•				
physical addresses	byte	25			physical addresses	byt	P S		
addresses					addresses				
0x00-3	00	11	22	33	0x20-3	D0	E1	D2	D3
0x04-7	44	55	66	77	0x24-7	D4	E5	D6	E7
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	ЗА	4A	0x30-3	ВА	0Α	ВА	0Α
0x14-7	1В	2B	3B	4B	0x34-7	DB	0B	DB	0B
0x18-B	1C	2C	3C	4C	0x38-B	EC	0C	EC	0C
0x1C-F	AC	ВС	DC	EC	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
                                                    0 \times 376 = 110 \ 111 \ 0110
0 \times 00 - 3 \mid 00 \ 11 \ 22 \ 33
                           0x20-3|D0 E1 D2 D3
                                                    PTE 1: 0x10 + 6 \times 2 = 0x1C:
0 \times 04 - 7 | 44 55 66 77
                           0x24-7D4 E5 D6 E7
                                                    AC BC
0x08-Bl88 99 AA BB
                           0x28-Bl89 9A AB BC
                                                    PTF 1: PPN 10 valid 1
0x0C-FCC DD EE FF
                           0x2C-FCD DE EF F0
                                                     PTE 2: 0x20 + 7 \times 2 = 0x2E:
                           0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
                                                    FF F0
                           0 \times 34 - 7 | DB | 0B | DB | 0B
0 \times 14 - 7 | 1B 2B 3B 4B
                                                     PTE 2: PPN 11 valid 1
0x18-Bl1C 2C 3C 4C
                           0x38-BIEC 0C EC 0C
                                                     11 0110 = 0x36 \rightarrow DB
0×1C-FAC BC DC EC
                           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
                                                    0 \times 376 = 110 \ 111 \ 0110
0 \times 00 - 3 \mid 00 \ 11 \ 22 \ 33
                           0x20-3|D0 E1 D2 D3
                                                    PTE 1: 0x10 + 6 \times 2 = 0x1C:
0 \times 04 - 7 | 44 55 66 77
                           0x24-7D4 E5 D6 E7
                                                    AC BC
0x08-Bl88 99 AA BB
                           0x28-Bl89 9A AB BC
                                                    PTF 1: PPN 10 valid 1
0x0C-FCC DD EE FF
                           0x2C-FCD DE EF F0
                                                    PTE 2: 0x20 + 7 \times 2 = 0x2E:
                           0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
                                                    FF F0
                           0 \times 34 - 7 | DB | 0B | DB | 0B
0 \times 14 - 7 | 1B 2B 3B 4B
                                                    PTE 2: PPN 11 valid 1
0x18-Bl1C 2C 3C 4C
                           0x38-BIEC 0C EC 0C
                                                    11 0110 = 0x36 \rightarrow DB
0×1C-FAC BC DC EC
                           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
                                                   0 \times 376 = 110 \ 111 \ 0110
0 \times 00 - 3 | 00 \ 11 \ 22 \ 33
                          0x20-3|D0 E1 D2 D3
                                                   PTE 1: 0x10 + 6 \times 2 = 0x1C:
0 \times 04 - 7 | 44 55 66 77
                          0x24-7D4 E5 D6 E7
                                                   AC BC
                          0x28-Bl89 9A AB BC
0x08-Bl88 99 AA BB
                                                   PTF 1: PPN 10 valid 1
0x0C-FCC DD EE FF
                          0x2C-FCD DE EF F0
                                                   PTE 2: 0x20 + 7 \times 2 = 0x2E:
                          0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
                                                   EF F0
0 \times 14 - 7 | 1B 2B 3B 4B
                          0x34-7|DB 0B DB 0B
                                                   PTE 2: PPN 11 valid 1
0x18-Bl1C 2C 3C 4C
                          0x38-BIEC 0C EC 0C
                                                   11 0110 = 0x36 \rightarrow DB
0x1C-FAC BC DC EC
                          0x3C-FIFC 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
                                                    0 \times 376 = 110 \ 111 \ 0110
0 \times 00 - 3 \mid 00 \ 11 \ 22 \ 33
                           0x20-3|D0 E1 D2 D3
                                                    PTE 1: 0x10 + 6 \times 2 = 0x1C:
0 \times 04 - 7 | 44 55 66 77
                           0x24-7D4 E5 D6 E7
                                                    AC BC
0x08-Bl88 99 AA BB
                           0x28-Bl89 9A AB BC
                                                    PTF 1: PPN 10 valid 1
0x0C-FCC DD EE FF
                           0x2C-FCD DE EF F0
                                                     PTE 2: 0x20 + 7 \times 2 = 0x2E:
                           0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
                                                    FF F0
                           0 \times 34 - 7 | DB | 0B | DB | 0B
0 \times 14 - 7 | 1B 2B 3B 4B
                                                     PTE 2: PPN 11 valid 1
0x18-Bl1C 2C 3C 4C
                           0x38-BIEC 0C EC 0C
                                                     11 0110 = 0x36 \rightarrow DB
0×1C-FAC BC DC EC
                           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
                                                    0 \times 376 = 110 \ 111 \ 0110
0 \times 00 - 3 | 00 \ 11 \ 22 \ 33
                           0x20-3|D0 E1 D2 D3
                                                    PTE 1: 0x10 + 6 \times 2 = 0x1C:
0 \times 04 - 7 | 44 55 66 77
                           0x24-7D4 E5 D6 E7
                                                    AC BC
0x08-Bl88 99 AA BB
                           0x28-Bl89 9A AB BC
                                                    PTF 1: PPN 10 valid 1
0x0C-FCC DD EE FF
                           0x2C-FCD DE EF F0
                                                    PTE 2: 0x20 + 7 \times 2 = 0x2E:
                           0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
                                                    FF FO
                           0 \times 34 - 7 | DB | 0B | DB | 0B
0 \times 14 - 7 | 1B 2B 3B 4B
                                                    PTE 2: PPN 11 valid 1
0x18-Bl1C 2C 3C 4C
                           0x38-BIEC 0C EC 0C
                                                    11 0110 = 0x36 \rightarrow DB
0×1C-FAC BC DC EC
                           0x3C-FIFC 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
                                                    0 \times 376 = 110 \ 111 \ 0110
0 \times 00 - 3 | 00 \ 11 \ 22 \ 33
                           0x20-3|D0 E1 D2 D3
                                                    PTE 1: 0x10 + 6 \times 2 = 0x1C:
0 \times 04 - 7 | 44 55 66 77
                           0x24-7D4 E5 D6 E7
                                                    AC BC
                           0x28-Bl89 9A AB BC
0x08-Bl88 99 AA BB
                                                    PTF 1: PPN 10 valid 1
0x0C-FCC DD EE FF
                           0x2C-FCD DE EF F0
                                                    PTE 2: 0x20 + 7 \times 2 = 0x2E:
                           0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
                                                    FF F0
                           0 \times 34 - 7 | DB | 0B | DB | 0B
0 \times 14 - 7 | 1B 2B 3B 4B
                                                    PTE 2: PPN 11 valid 1
0x18-Bl1C 2C 3C 4C
                           0x38-BIEC 0C EC 0C
                                                    11 0110 = 0x36 \rightarrow DB
0×1C-FAC BC DC EC
                           0x3C-FIFC 0C FC 0C
```

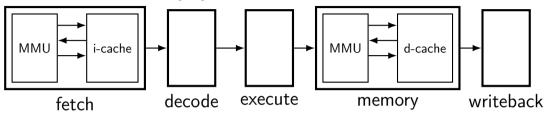
10-bit virtual addresses, 6-bit physical; 16 byte pages, 2 byte PTE

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

```
physical bytes
addresses
                                                  0 \times 376 = 110 \ 111 \ 0110
0 \times 00 - 3 | 00 \ 11 \ 22 \ 33
                          0x20-3|D0 E1 D2 D3
                                                  PTE 1: 0x10 + 6 \times 2 = 0x1C:
0 \times 04 - 7 | 44 55 66 77
                          0x24-7D4 E5 D6 E7
                                                  AC BC
0x08-Bl88 99 AA BB
                          0x28-Bl89 9A AB BC
                                                  PTF 1: PPN 10 valid 1
0x0C-FCC DD EE FF
                          0x2C-FCD DE EF F0
                                                   PTE 2: 0x20 + 7 \times 2 = 0x2E:
                          0x30-3|BA 0A BA 0A
0 \times 10 - 3 | 1A 2A 3A 4A
                                                  FF F0
                          0x34-7DB 0B DB 0B
0 \times 14 - 7 | 1B 2B 3B 4B
                                                  PTE 2: PPN 11 valid 1
0x18-Bl1C 2C 3C 4C
                          0x38-BIEC 0C EC 0C
                                                   11 0110 = 0x36 \rightarrow DB
0×1C-FAC BC DC EC
                          0x3C-FIFC 0C FC 0C
```

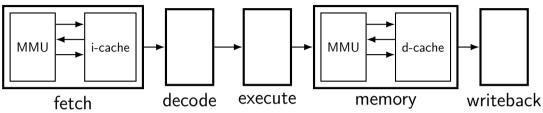
backup slides

MMUs in the pipeline



up to four memory accesses per instruction

MMUs in the pipeline

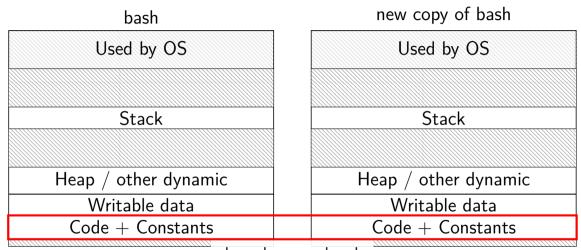


up to four memory accesses per instruction challenging to make this fast (topic for a future date)

do we really need a complete copy?

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

do we really need a complete copy?



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

0x00601 0x00602

0x00603 0x00604

0x00605

•••

valid? write?

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

ا مه نصا ما

VPN	valid? write?					
VIII	valiu:	wiite:	page			
•••	•••	•••	•••			
0x00601	1	0	0x12345			
0x00602	1	0	0x12347			
0x00603	1	0	0x12340			
0x00604	1	0	0x200DF			
0x00605	1	0	0x200AF			
•••	•••	•••	•••			

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

V/DN

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

physical

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

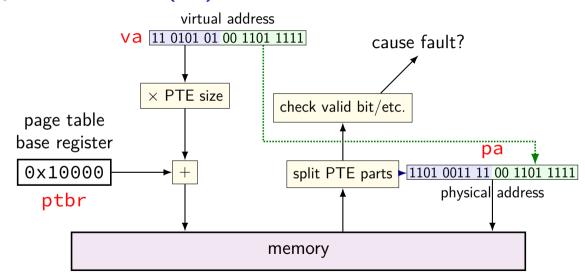
VPN	valid?	writo'	physical page	VPN	valid? write?		
VITIN	pag		page	page		write	page
•••	•••	•••	•••	•••	•••	•••	•••
0x00601	1	0	0x12345	0x00601	1	0	0x12345
0x00602	1	0	0x12347	0x00602	1	0	0x12347
0x00603	1	0	0x12340	0x00603	1	0	0x12340
0x00604	1	0	0x200DF	<u>0x00604</u>	1	0	0x200DF
0x00605	1	0	0x200AF	0x00605	1	0	0x200AF
•••	•••	•••	•••	•••	•••	•••	•••

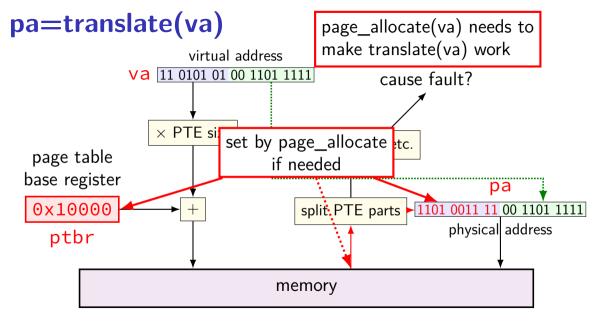
when either process tries to write read-only page triggers a fault — OS actually copies the page

VPN	physical valid? write? page		VPN	valid?	physical valid? write?		
VIIN			page	V I IV	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	0x00604	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

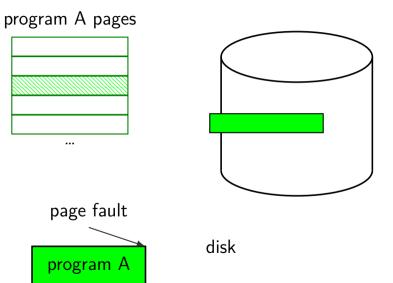
```
"swap in" a page — exactly like allocating on demand!
     OS gets page fault — invalid in page table
     check where page actually is (from virtual address)
     read from disk
    eventually restart process
"swap out" a page
     OS marks as invalid in the page table(s)
     copy to disk (if modified)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

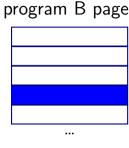
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

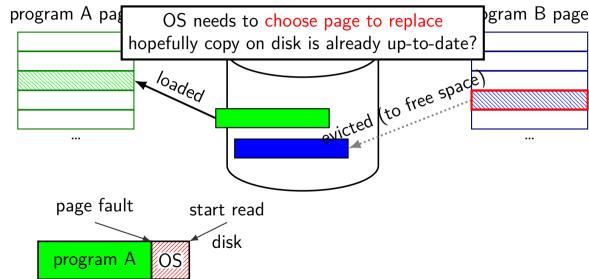
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

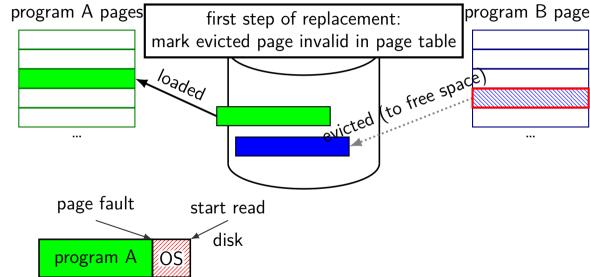
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

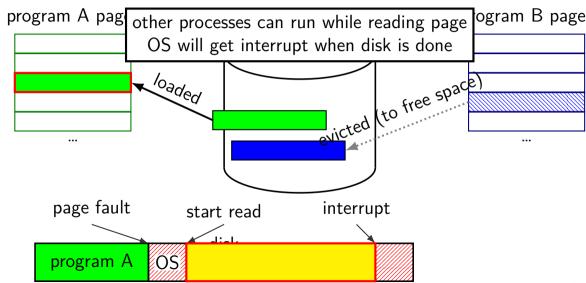
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes



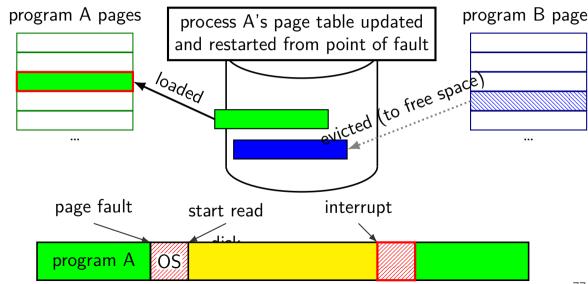








7



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