last time

more page table tricks

generalized idea: on exception, maybe fix page table OS tracking memory as list of mappings (separate from page table) page table from HW for hardware

loading from/unloading to files on disk copy-on-write

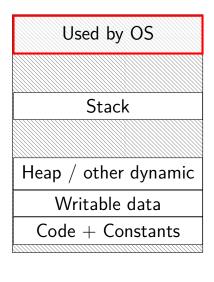
...

storing page tables in memory page table base register

representing entries as integers

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

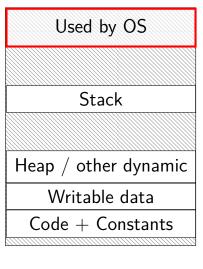
0xffff Ffff Ffff Ffff
0xffff 8000 0000 0000
0x7f...



0x7F...

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

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



0xffff ffff ffff ffff

0xFFFF 8000 0000 0000

0x7F...

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

physical addresses	byt	es			physica addresse	al s s	es		
0x00-3			22	33	0x20-			D1	F3
0x04-7	44	55	66	77	0x24-	7 E 4	E5	F6	07
0x08-B	88	99	AA	ВВ	0x28-	B <mark>89</mark>	9A	AB	ВС
0x0C-F	CC	DD	EE	FF	0x2C-	FCD	DE	EF	F0
0x10-3	1A	2A	ЗА	4A	0x30-	3ВА	0A	ВА	0A
0x14-7	1B	2B	3B	4B	0x34-	7 CB	0B	СВ	0B
0x18-B	1C	2C	3C	4C	0x38-	BDC	0C	DC	0C
0x1C-F	1C	2C	3C	4C	0x3C-	FEC	0C	EC	0C

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

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

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

 $0 \times 12 = 01 \quad 0010$ $PTE \ addr$: $0 \times 20 + 2 \times 1 = 0 \times 22$ $PTE \ value$: $0 \times D1 = 1101 \quad 0001$ $PPN \ 110, \ valid \ 1$ $M[110 \ 001] = M[0 \times 32]$ $\rightarrow 0 \times BA$

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

physical bytes physical bytes addresses addresses 0x20-3 A0 E2 D1 F3 0x00-3|00 11 22 33 0x04-7|44 55 66 77 0x24-7|E4 E5 F6 07 0x08-B|88 99 AA BB 0x28-B|89 9A AB BC 0x2C-FCD DE EF F0 0x0C-FICC DD EE FF 0x10-3|1A 2A 3A 4A 0x30-3|BA 0A BA 0A 0x14-7|1B 2B 3B 4B 0x34-7|CB 0B CB 0B 0x38-BDC 0C DC 0C 0x18-B|1C 2C 3C 4C 0x1C-F|1C 2C 3C 4C 0x3C-FEC 0C EC 0C

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

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

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

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

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

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

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

physical bytes 0x12 = 01 0010addresses PTE addr: 0x20-3 A0 E2 D1 F3 $0x20 + 2 \times 1 = 0x22$ 0x24-7|E4 E5 F6 07 0x28-B|89 9A AB BC PTE value: 0x2C-FCD DE EF F0 0xD1 = 1101 00010x30-3|BA 0A BA 0A PPN 110, valid 1 0x34-7|CB 0B CB 0B $M[110 \ 001] = M[0x32]$ 0x38-BDC 0C DC 0C \rightarrow 0xBA 0x3C-FEC 0C EC 0C

pagetable assignment

pagetable assignment

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

in assignment:

virtual address \sim arguments to your functions

physical address \sim your program addresses (normal pointers)

pagetable assignment API

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

translate()

with POBITS=12, LEVELS=1:

ptbr = GetPointerToTable(

vana.	p, 5.0a.	
0		
1	0×9999	١
0		,
1	0x3333	
	0 1 0 1 	0 —

VPN valid? physical

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

translate()

with POBITS=12, LEVELS=1:

ptbr =	GetPointerToTable(

 VPN valid? physical

 0
 0

 1
 1
 0×9999

 2
 0
 —

 3
 1
 0×3333

 ...
 ...
 ...

```
\begin{array}{l} translate(0x0 \mbox{FFF}) == (void^*) \mbox{ $^{\circ}$UL} \\ translate(0x1000) == (void^*) \mbox{ $0x9999000$} \\ translate(0x1001) == (void^*) \mbox{ $0x9999001$} \\ translate(0x2000) == (void^*) \mbox{ $^{\circ}$UL} \\ translate(0x2001) == (void^*) \mbox{ $^{\circ}$UL} \\ translate(0x3000) == (void^*) \mbox{ $0x3333000$} \\ \end{array}
```

page_allocate()

page_allocate()

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

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

VF IV Vallu! physical						
0	0					
1	1	(new))	١		
2	0)		
3	1					

VPN valid2 physical

allocated with posix_memalign

page_allocate()

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

ptbr now == GetPointerToTable(

VIIV	vanu:	Pilysi	Cui	
0	0			
1	1	(new))	١
2	0	_ ^)
3	1			

VPN valid2 physical

allocated with posix_memalign

posix_memalign

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

posix_memalign

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

posix_memalign

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

parts

```
part 1 (next week): LEVELS=1, POBITS=12 and
    translate() OR
    page_allocate()
part 2 (two weeks after break): all LEVELS, both functions
    in preparation for code review
    due Weds BFFORF LAB
part 3 (two weeks after break): final submission
    Friday after code review
    most of grade based on this
    will test previous parts again
```

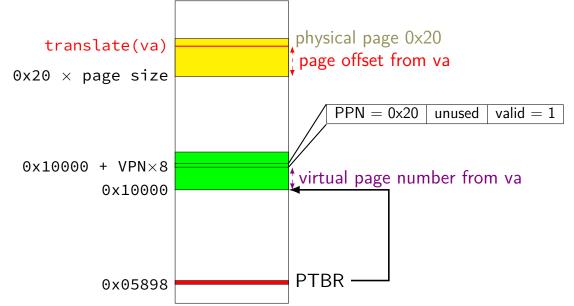
address/page table entry format

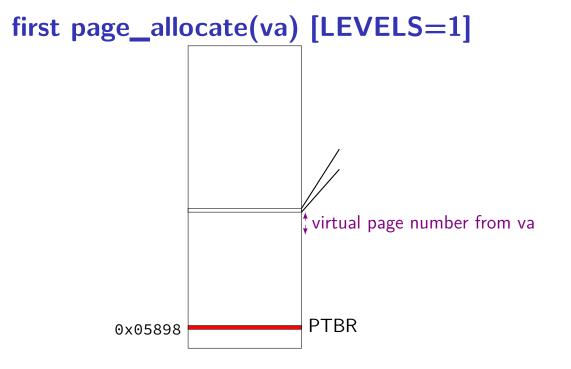
(with POBITS=12, LEVELS=1)

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

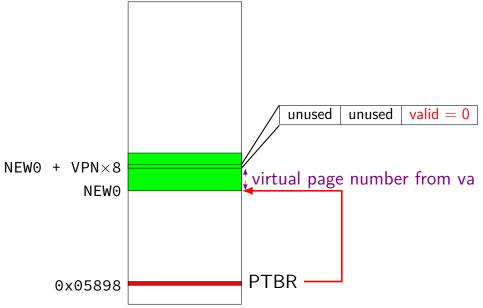
in assignment: value from posix_memalign = physical address

pa = translate(va) [LEVELS=1]

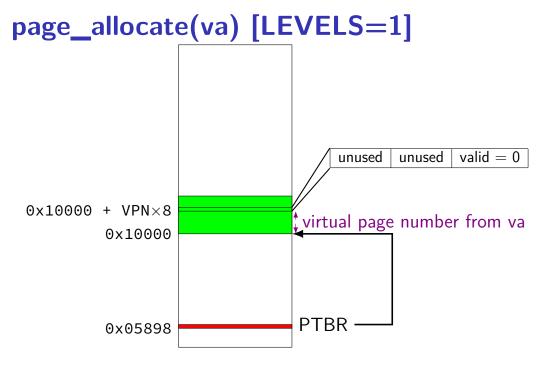


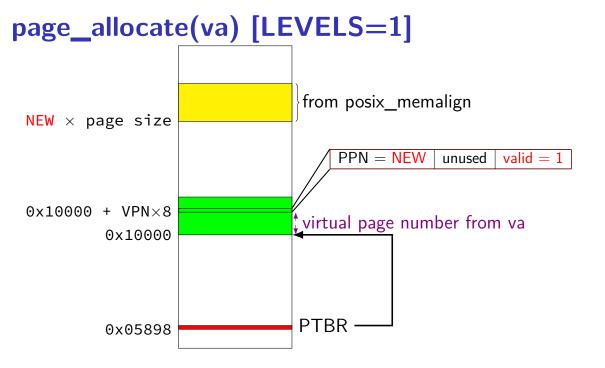


first page_allocate(va) [LEVELS=1]

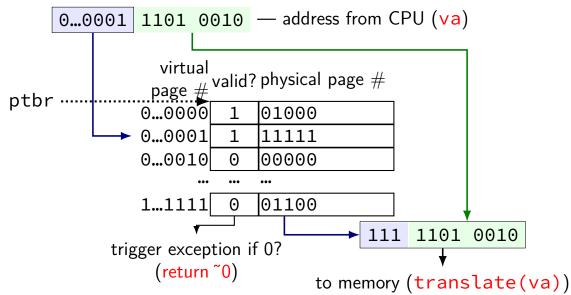


first page_allocate(va) [LEVELS=1] from posix_memalign $NEW1 \times page size$ PPN = NEW1valid = 1unused NEW0 + VPN×8 ‡virtual page number from va NEW₀ **PTBR** 0x05898

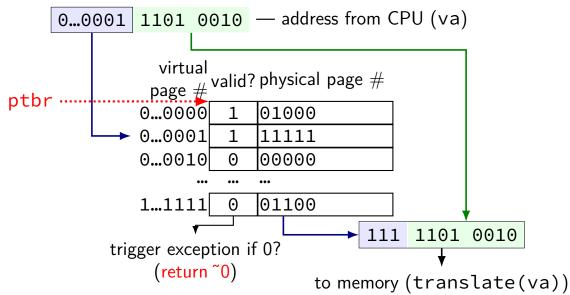




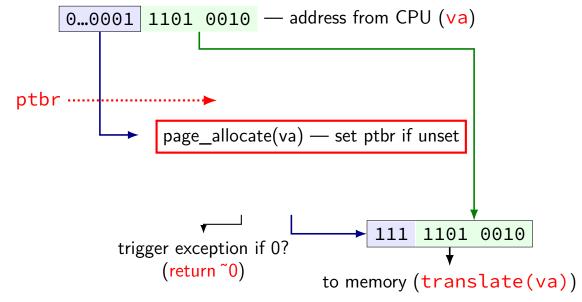
page table lookup (and translate())



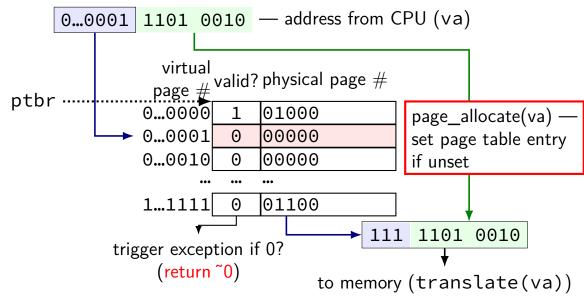
page table lookup (and translate())



page table lookup (and allocate)



page table lookup (and allocate)



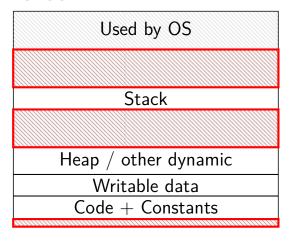
huge page tables

huge virtual address spaces!

impossible to store PTE for every page

how can we save space?

holes



most pages are invalid

saving space

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

saving space

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

hashtable

actually used by some historical processors but never common

saving space

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

hashtable

actually used by some historical processors but never common

tree data structure

but not quite a search tree

search tree tradeoffs

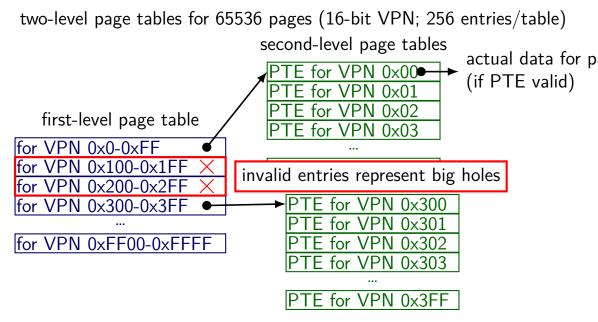
lookup usually implemented in hardware

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

lookup should not involve many memory accesses

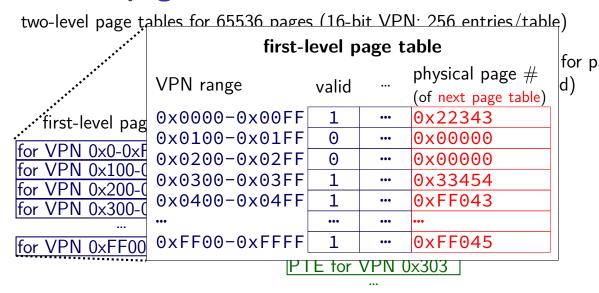
doing two memory accesses is already very slow solution: tree with many children from each node (far from binary tree's left/right child)

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



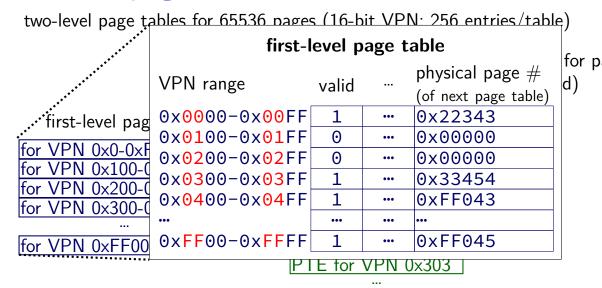
two-level page tables for 65536 pages (16-bit VPN: 256 entries/table) first-level page table for p physical page # VPN range valid d) (of next page table) 0x0000-0x00FF 0x22343 first-level pag $0 \times 0100 - 0 \times 01 FF$ 0 0×00000 VPN 0x0-0xF $0 \times 0200 - 0 \times 02FF$ 0 0×00000 VPN 0x100-0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

PTE for VPN 0x3FF

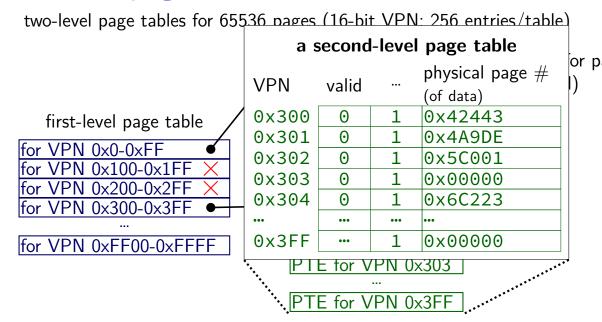


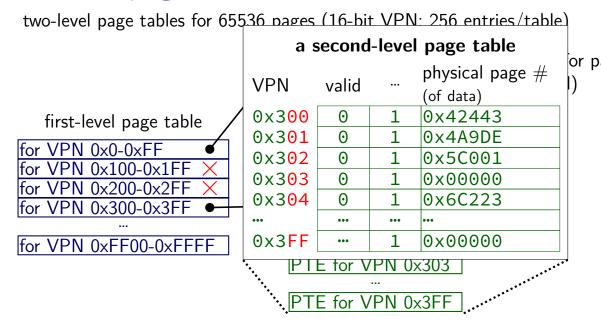
TE for VPN 0x3FF

23



PTE for VPN 0x3FF





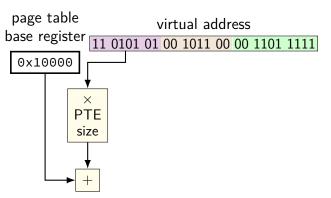
two-level page tables for 65536 pages (16-bit VPN; 256 entries/table) second-level page tables actual data for p for VPN 0x00 (if PTE valid) first-level page table for VPN $0 \times 0 - 0 \times FF$ tor VPN $0 \times 100 - 0 \times 1$ FF IPTE for VPN 0xFF VPN 0x200-0x2FF for VPN 0x300-0x3FF VPN 0x300 for VPN 0xFF00-0xFFFF VPN 0x302 TE for VPN 0x303 for VPN 0x3FF

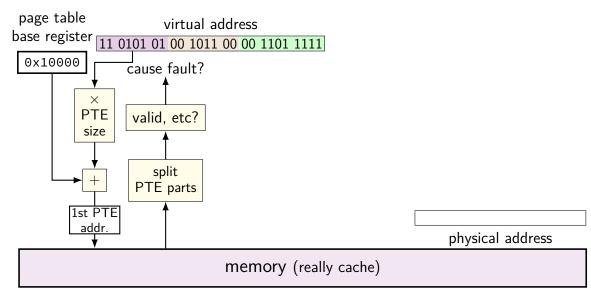
virtual address

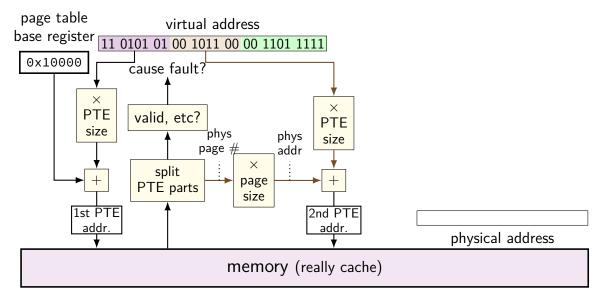
11 0101 01 00 1011 00 00 1101 1111

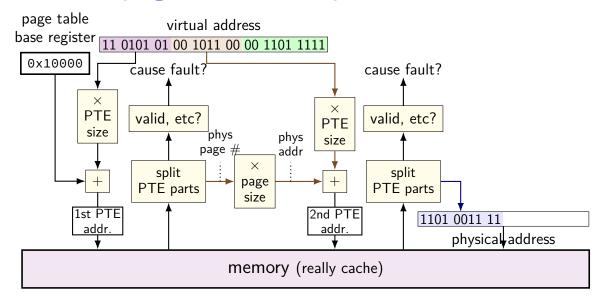
VPN — split into two parts (one per level)

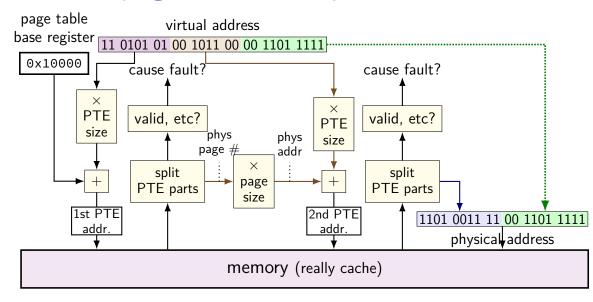
this example: parts equal sized — common, but not required

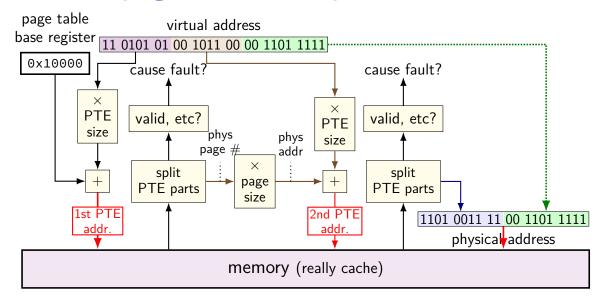


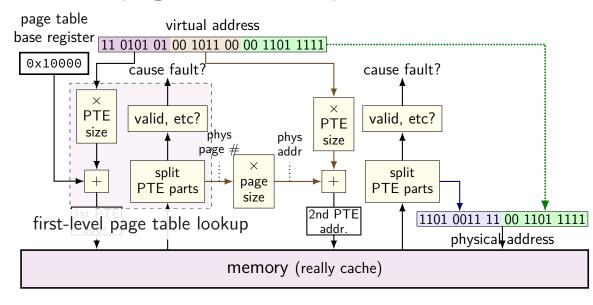


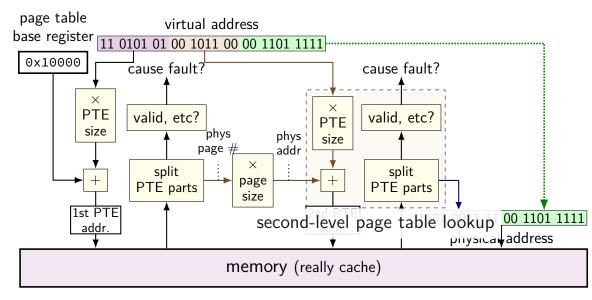


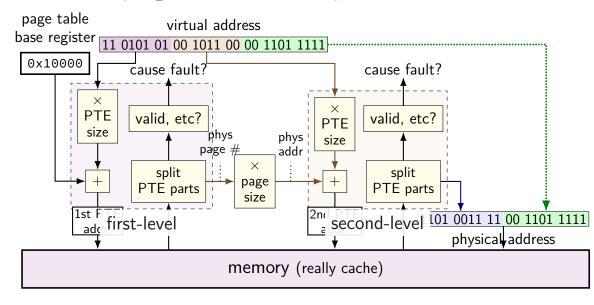


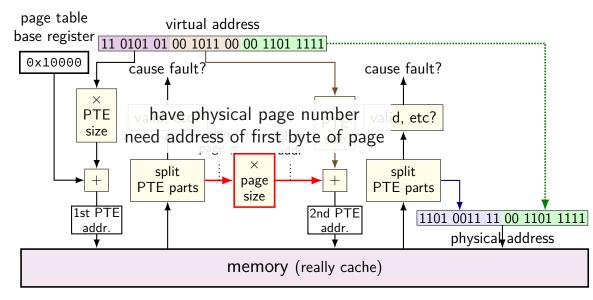


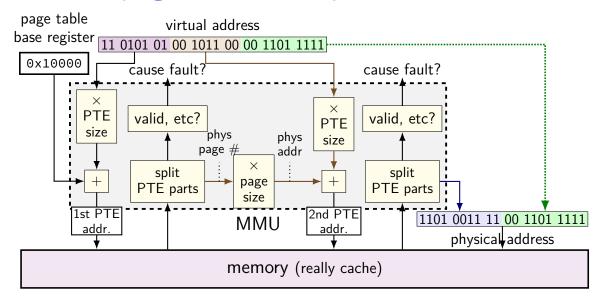












another view



multi-level page tables

VPN split into pieces for each level of page table

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

bottom level: page table entry points to destination page

validity checks at each level

note on VPN splitting

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

assignment

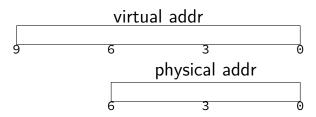
assignment: variable number of LEVELS

page tables always one page size

means upper bits of virtual address values not used

9-bit virtual address

6-bit physical address



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

virtual addr

PPN

page offset

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

9-bit virtual address

virtual addr
VPN page offset
6 3

6-bit physical address

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

page table (either level)

- 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

9-bit virtual address

virtual addr page offset VPN pt 1 VPN pt 2

PPN

- 6-bit physical address
- 8-byte pages \rightarrow 3-bit page offset (bottom) ⁶
 - page offset page table (either level)

physical addr

- 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

valid? PPN

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

physical addresses	byt	es			physical addresses	byt	es		
0x00-3			22	33	0x20-3			72	13
0x04-7	44	55	66	77	0x24-7	F4	Α5	36	07
0x08-B	88	99	AA	ВВ	0x28-B	89	9A	AB	ВС
0x0C-F	CC	DD	ΕE	FF	0x2C-F	CD	DE	EF	F0
0x10-3	1A	2A	ЗА	4A	0x30-3	ВА	0A	ВА	0Α
0x14-7	1В	2B	3B	4B	0x34-7	DB	0B	DB	0B
0x18-B	10	2C	3C	4C	0x38-B	EC	0C	EC	0C
0x1C-F	1C	2C	3C	4C	0x3C-F	AC	DC	DC	0C

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

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

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

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

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

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

physical by addresses_	tes		physical byte	S
0x00-300		2 33	0x20-300	
0x04-74	1 55 66	6 77	0x24-7F4	A5 36 07
0x08-B8	3 99 A <i>A</i>	A BB	0x28-B89	9A AB BC
0x0C-FC	DD EE	FF	0x2C-FCD	DE EF F0
0x10-31	A 2A 3A	4 4 A	0x30-3BA	<mark>0A</mark> BA 0A
0x14-71	3 2B 3E	3 4B	0x34-7DB	0B DB 0B
0x18-B1	C 2C 30	C 4C	0x38-BEC	0C EC 0C
0x1C-F1	C 2C 30	C 4C	0x3C-FAC	DC DC 0C

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

physical bytes addresses physical bytes addresses	D3
auui csscs auui esses	D3
0x00-300 11 22 33 0x20-3D0 D1 D2	
0x04-744 55 66 77 0x24-7 D4 D5 D6	D7
0x08-B88 99 AA BB	ВС
0x0C-FCC DD EE FF 0x2C-FCD DE EF	F0
0x10-3 1A 2A 3A 4A 0x30-3 BA 0A BA	0Α
0x14-7 1B 2B 3B 4B 0x34-7 DB 0B DB	0B
0x18-B1C 2C 3C 4C	0C
0x1C-F 1C 2C 3C 4C	0C

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

physical addresses	byte	es			physical bytes addresses
0x00-3	00	11	22	33	0x20-3D0 D1 D2 D3
0x04-7	44	55	66	77	0x24-7D4 D5 D6 D
0x08-B	88	99	AΑ	ВВ	0x28-B89 9A AB B
0x0C-F	CC	DD	EE	FF	0x2C-FCD DE EF F
0x10-3	1A	2A	3A	4A	0x30-3BA 0A BA 0
0x14-7	1В	2B	3B	4B	0x34-7DB 0B DB 0I
0x18-B	1C	2C	3C	4C	0x38-BEC 0C EC 00
0x1C-F	1C	2C	3C	4C	0x3C-FFC 0C FC 00

physical addresses	byte	es			physical addresses	byt	es		
0x00-3			22	33	0x20-3	D0	D1	D2	D3
0x04-7	44	55	66	77	0x24-7	D4	D5	D6	D7
0x08-B	88	99	AA	ВВ	0x28-B	89	9A	AB	ВС
0x0C-F	CC	DD	EE	FF	0x2C-F	CD	DE	EF	F0
0x10-3	1A	2A	ЗА	4A	0x30-3	ВА	0A	ВА	0A
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	1C	2C	3C	4C	0x3C-F	FC	0C	FC	0C

2 D3
5 D7
3 BC
F (
\ 0 <i>F</i>
3 OE
00
00

physical addresses	byt	es			physica addresse	l byt	es		
0x00-3			22	33	0x20-3			D2	D3
0x04-7	44	55	66	77	0x24-	7 D4	D5	D6	D7
0x08-B	88	99	AΑ	ВВ	0x28-l	389	9A	ΑB	ВС
0x0C-F	CC	DD	ΕE	FF	0x2C-I	-CD	DE	EF	F0
0x10-3	1A	2A	3A	4A	0x30-3	BA	0A	ВА	0Α
0x14-7	1B	2B	3B	4B	0x34-	7 DB	0B	DB	0B
0x18-B	10	2C	3C	4C	0x38-l	3 EC	0C	EC	0C
0x1C-F	1C	2C	3C	4C	0x3C-I	FC	0C	FC	0C

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

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

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

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

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

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

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

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

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

physical bytes addresses				
0x00-3				
0x04-7	44	55	66	77
0x08-B	88	99	AΑ	ВВ
0x0C-F				
0x10-3	1A	2A	3A	4A
0x14-7	1B	2B	3B	4B
0x18-B	1C	2C	3C	4C
0x1C-F	AC	ВС	DC	EC

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

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

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

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

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

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

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

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

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

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

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

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

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

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

```
physical addresses

0x20-3 D0 E1 D2 D3

0x24-7 D4 E5 D6 E7

0x28-B 89 9A AB BC

0x2C-F CD DE EF F0

0x30-3 BA 0A BA 0A

0x34-7 DB 0B DB 0B

0x38-B EC 0C EC 0C

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

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

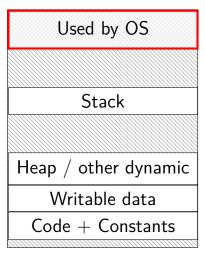
page tables 1 page; PTE 1st byte: (MSB) 2-bit PPN, valid bit; rest unused

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

```
physical bytes
addresses
0x20-3D0 E1 D2 D3
0x24-7D4 E5 D6 E7
0x28-Bl89 9A AB BC
0x2C-FCD DE EF F0
0x30-3|BA 0A BA 0A
0x34-7DB 0B DB 0B
0x38-B|EC 0C EC 0C
0x3C-FIFC 0C FC 0C
```

backup slides

program memory



0xFFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F...

0x0000 0000 0040 0000

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

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

where in memory is this OS code?

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

where in memory is this OS code?

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

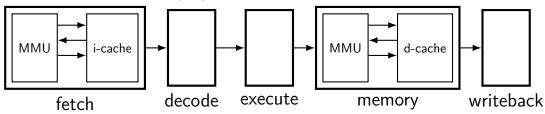
system calls, I/O events, etc. run OS code in kernel mode

where in memory is this OS code?

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

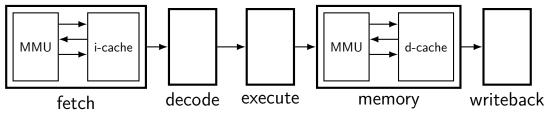
code better not be modified by user program otherwise: uncontrolled way to "escape" user mode

MMUs in the pipeline



up to four memory accesses per instruction

MMUs in the pipeline



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

do we really need a complete copy?

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

ما م ما

new c	opy of bash
Use	ed by OS
	Stack
Heap / o	other dynamic
Writ	able data
Code -	+ Constants

do we really need a complete copy?

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

shared as read-only

do we really need a complete copy?

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

trick for extra sharing

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

can we detect modifications?

trick for extra sharing

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

can we detect modifications?

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

VPN

valid? write?

•••

0x00601 0x00602 0x00603 0x00604 0x00605

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

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

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

V 1 1 4
0x00601
0x00602
0x00603
0x00604
0x00605
•••

VPN

valid? write? page				
•••	•••	•••		
-1	\sim	012241		

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

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

physical

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

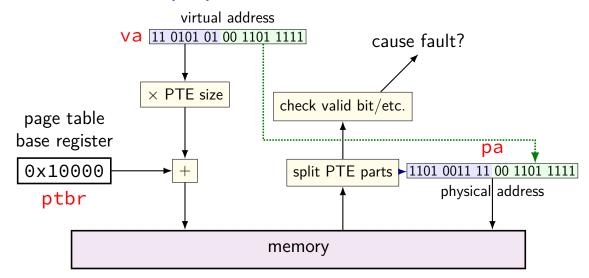
VPN	valid?	write	pnysicai page
V	vana.	*******	page
•••	•••	•••	•••
0x00601	1	0	0x12345
0x00602	1	0	0x12347
0x00603	1	0	0x12340
0x00604	1	0	0x200DF
0x00605	1	0	0x200AF
•••	•••	•••	•••

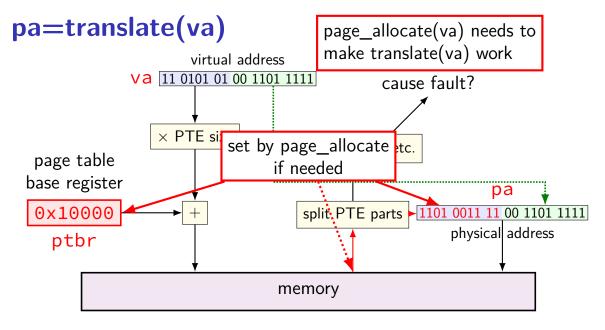
when either process tries to write read-only page triggers a fault — OS actually copies the page

\/DN	VPN valid? write? page		₂ physical	VPN	physical valid? write? page		
VIIN			page		valiu: 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	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

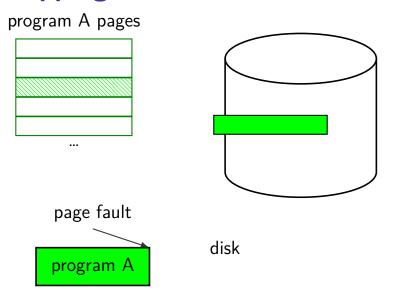
"swap out" a page
OS marks as invalid in the page table(s)
copy to disk (if modified)
```

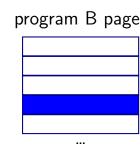
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

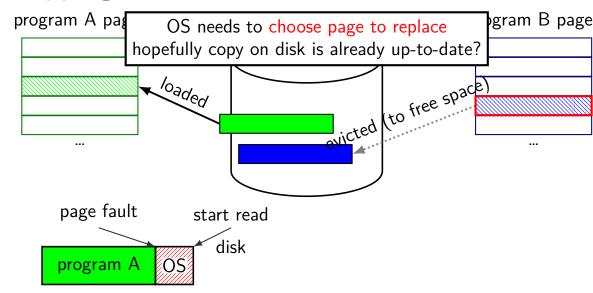
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

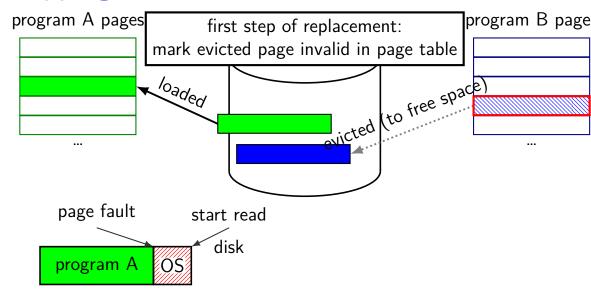
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

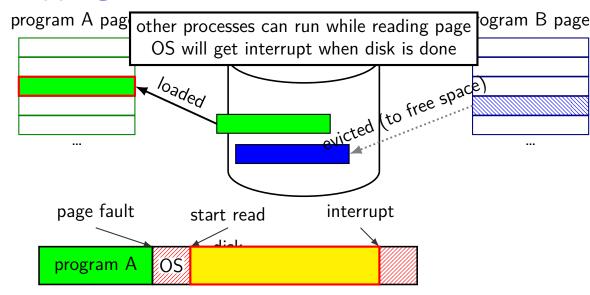
HDD reads and writes: milliseconds to tens of milliseconds minimum size: 512 bytes writing tens of kilobytes basically as fast as writing 512 bytes

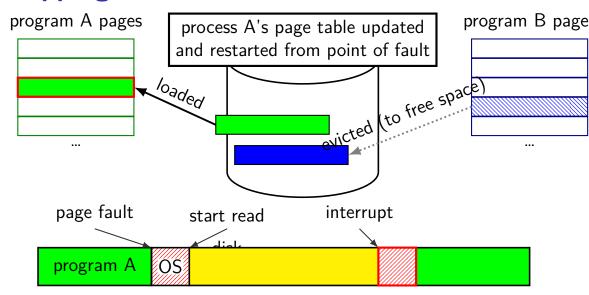












swapping almost mmap

```
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

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

4096 byte pages

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

4096 byte pages

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

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

4096 byte pages

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

exercise: how large are physical page numbers?

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

4096 byte pages

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

exercise: how large are physical page numbers?

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

4096 byte pages

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

exercise: how large are physical page numbers?

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

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

backup slides