

last time

- more page table tricks

 - loading from/unloading to files on disk

 - copy-on-write

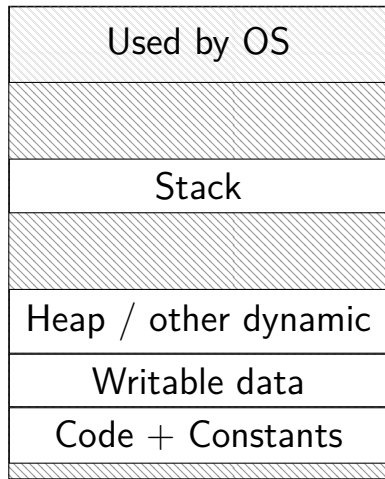
 - ...

- storing page tables in memory

 - page table base register

 - representing entries as integers

program memory



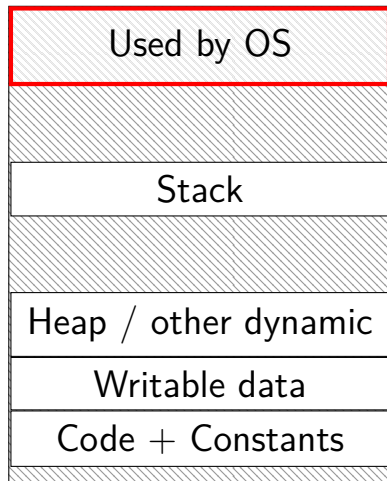
0xFFFF FFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F...

0x0000 0000 0040 0000

program memory



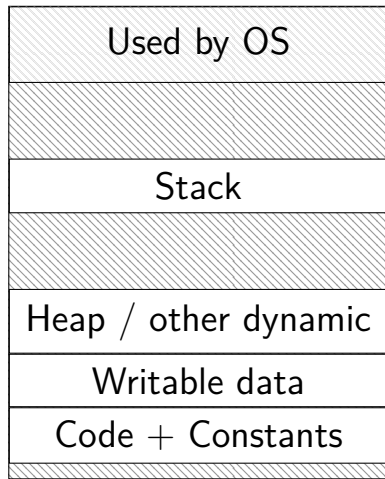
0xFFFF FFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F...

0x0000 0000 0040 0000

program memory



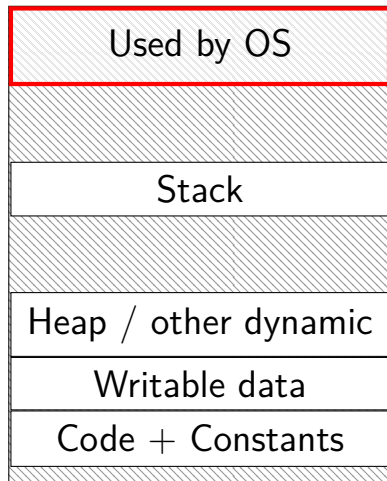
0xFFFF FFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F...

0x0000 0000 0040 0000

program memory



0xFFFF FFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F...

0x0000 0000 0040 0000

1-level exercise (2)

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	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	bytes
0x20-3	A0 E2 D1 F3
0x24-7	E4 E5 F6 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

1-level exercise (2)

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	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	bytes
0x20-3	A0 E2 D1 F3
0x24-7	E4 E5 F6 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

0x12 = 01 0010

PTE addr:

0x20 + 2 × 1 = 0x22

PTE value:

0xD1 = 1101 0001

PPN 110, valid 1

M[110 001] = M[0x32]

→ 0xBA

1-level exercise (2)

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	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	bytes
0x20-3	A0 E2 D1 F3
0x24-7	E4 E5 F6 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

0x12 = 01 0010

PTE addr:

$0x20 + 2 \times 1 = 0x22$

PTE value:

0xD1 = 1101 0001

PPN 110, valid 1

$M[110\ 001] = M[0x32]$

→ 0xBA

1-level exercise (2)

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	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	bytes
0x20-3	A0 E2 D1 F3
0x24-7	E4 E5 F6 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

0x12 = 01 0**010**

PTE addr:

$0x20 + 2 \times 1 = 0x22$

PTE value:

0xD1 = 1101 0001

PPN 110, valid 1

$M[110 \text{ } 001] = M[0x32]$

→ 0xBA

1-level exercise (2)

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	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	bytes
0x20-3	A0 E2 D1 F3
0x24-7	E4 E5 F6 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

0x12 = 01 0010

PTE addr:

$0x20 + 2 \times 1 = 0x22$

PTE value:

0xD1 = 1101 0001

PPN 110, valid 1

$M[110 \ 001] = M[0x32]$

→ 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 ...
#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:

VPN valid? physical)
0	0	—	
1	1	0x9999	
2	0	—	
3	1	0x3333	
...	

ptbr = GetPointerToTable(

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

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

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

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

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

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

translate()

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

VPN valid? physical)
0	0	—	
1	1	0x9999	
2	0	—	
3	1	0x3333	
...	

ptbr = GetPointerToTable(

translate(0x0**FFF**) == (void*) ~0L

translate(0x1**000**) == (void*) 0x9999**000**

translate(0x1**001**) == (void*) 0x9999**001**

translate(0x2**000**) == (void*) ~0L

translate(0x2**001**) == (void*) ~0L

translate(0x3**000**) == (void*) 0x3333**000**

page_allocate()

with POBITS=12, LEVELS=1:

ptbr == 0

page_allocate(0x1000) *or* page_allocate(0x1001) *or* ...

page_allocate()

with POBITS=12, LEVELS=1:

ptbr == 0

page_allocate(0x1000) or page_allocate(0x1001) or ...

ptbr *now* == GetPointerToTable(

VPN valid? physical

0	0	—
1	1	(new)
2	0	—
3	1	—
...

)

allocated with posix_memalign

page_allocate()

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

ptbr == 0

page_allocate(0x1**000**) or page_allocate(0x1**001**) or ...

ptbr *now* == GetPointerToTable(

VPN valid? physical

0	0	—
1	1	(new)
2	0	—
3	1	—
...

)

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 BEFORE LAB
- part 3 (two weeks after break): final submission
Friday after code review
most of grade based on this
will test previous parts again

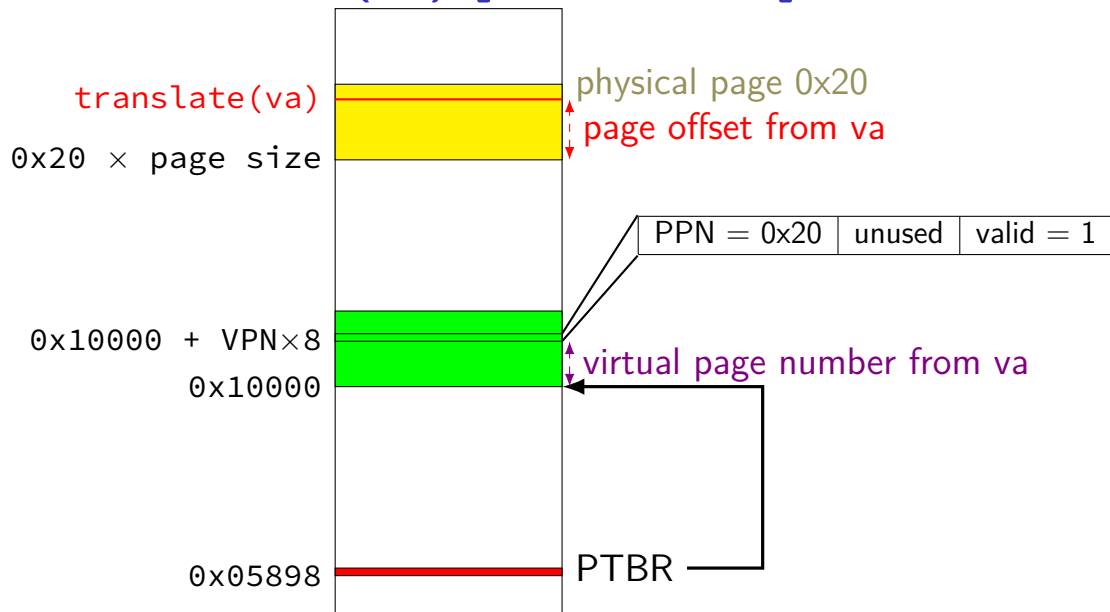
address/page table entry format

(with POBITS=12, LEVELS=1)

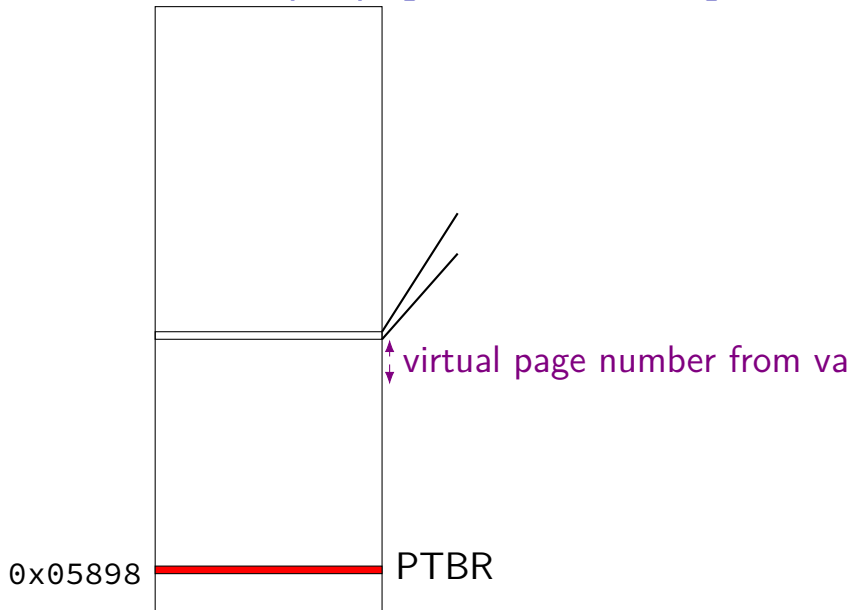
	bits 63–21	bits 20–12	bits 11–1	bit 0
page table entry	physical page number		unused	valid bit
virtual address	unused	virtual page number	page offset	
physical address	physical page number		page offset	

in assignment: value from `posix_memalign` = physical address

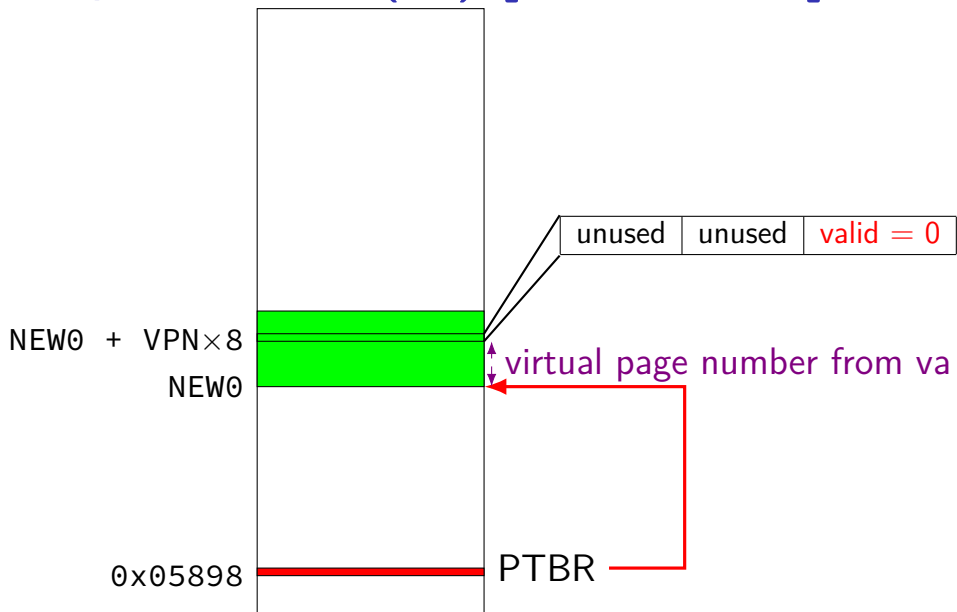
pa = translate(va) [LEVELS=1]



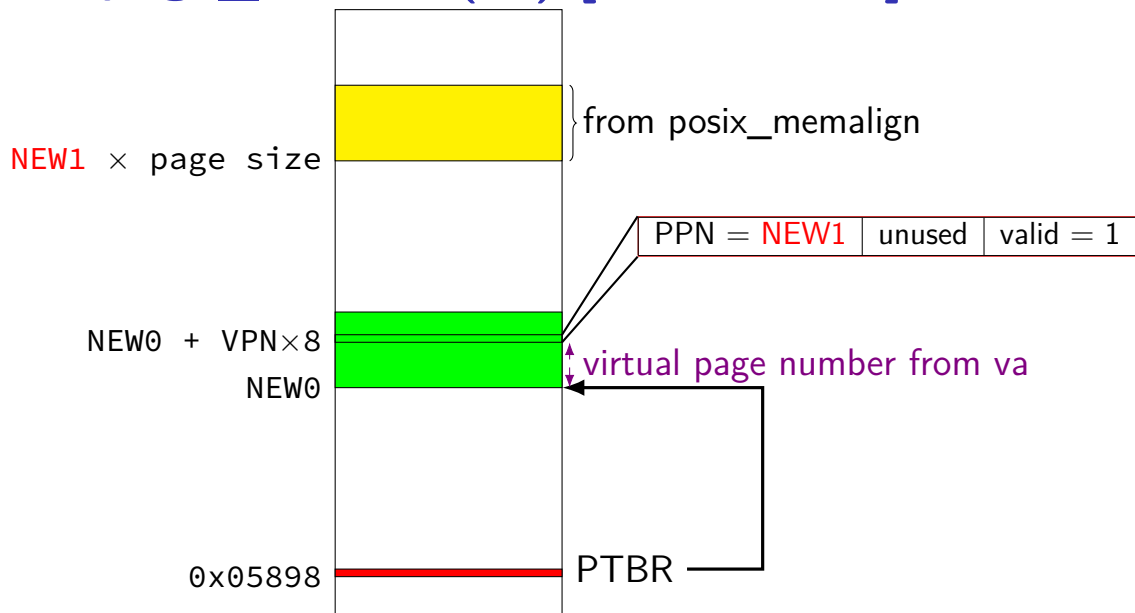
first_page_allocate(va) [LEVELS=1]



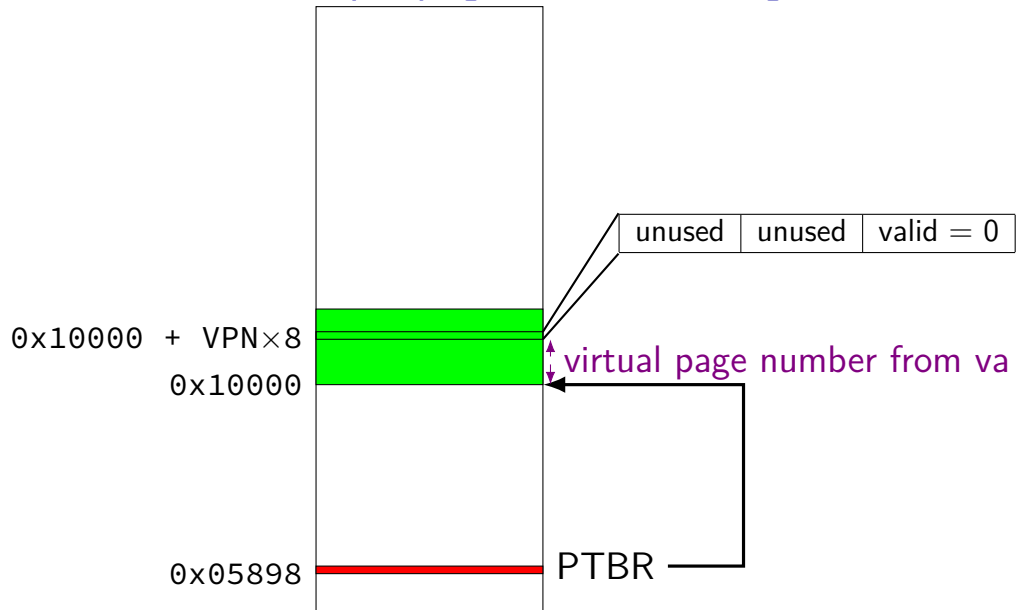
first_page_allocate(va) [LEVELS=1]



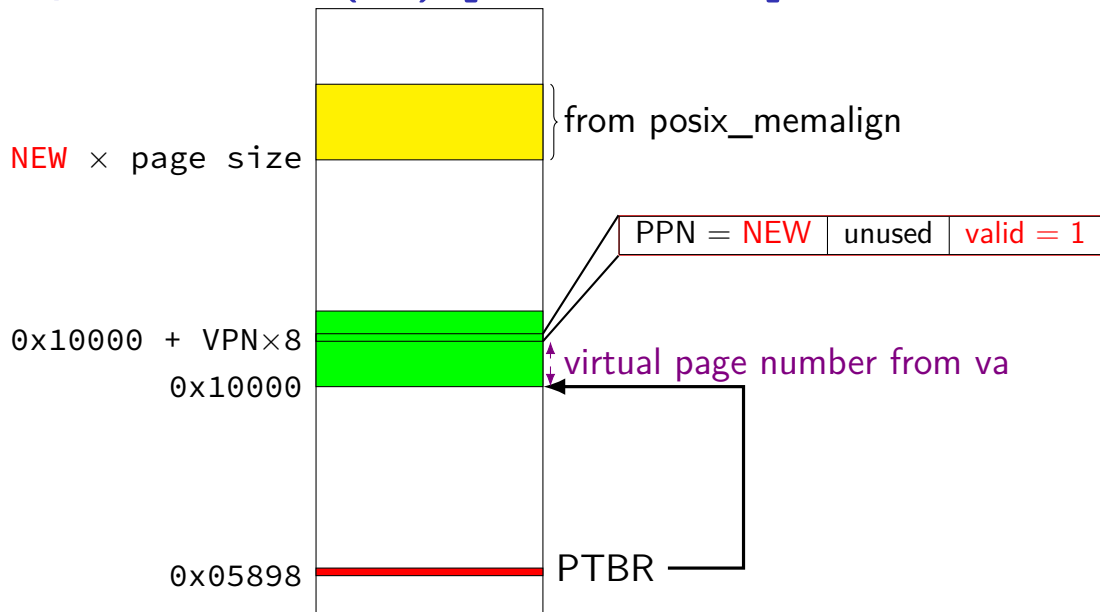
first page_allocate(va) [LEVELS=1]



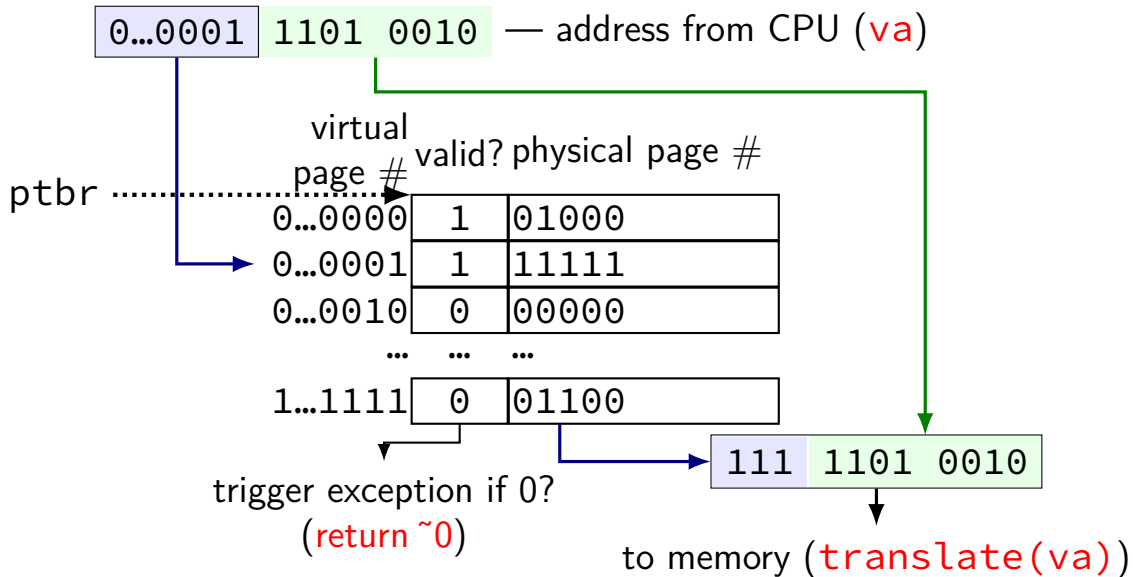
page_allocate(va) [LEVELS=1]



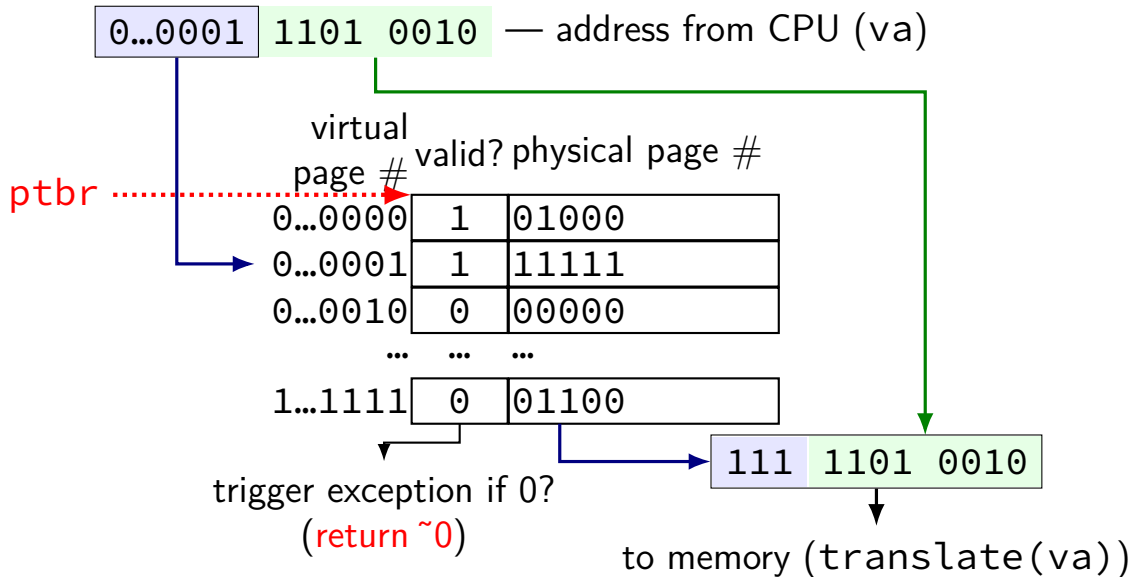
page_allocate(va) [LEVELS=1]



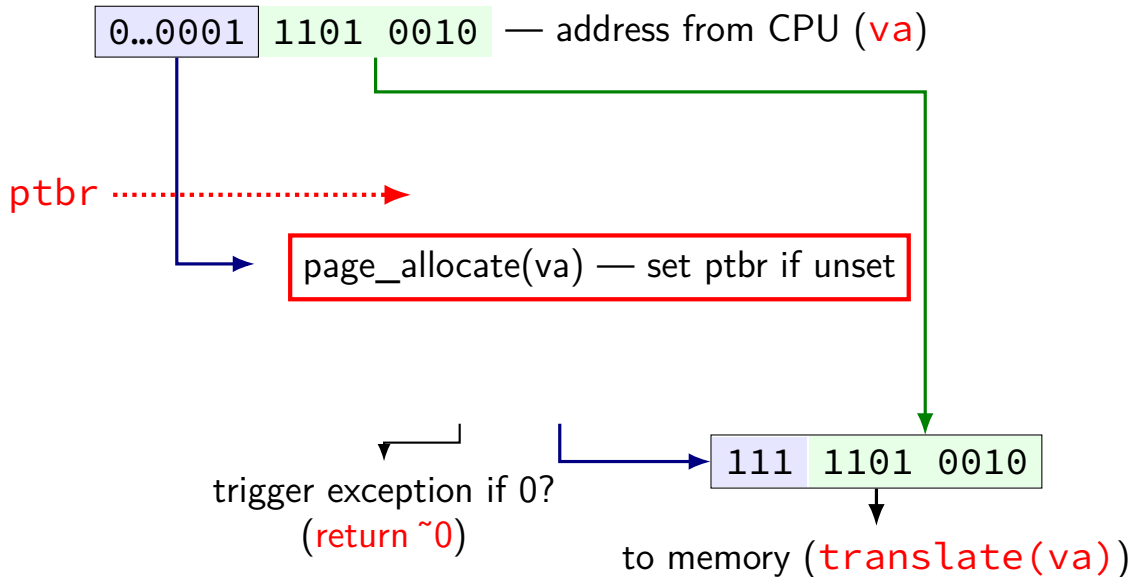
page table lookup (and translate())



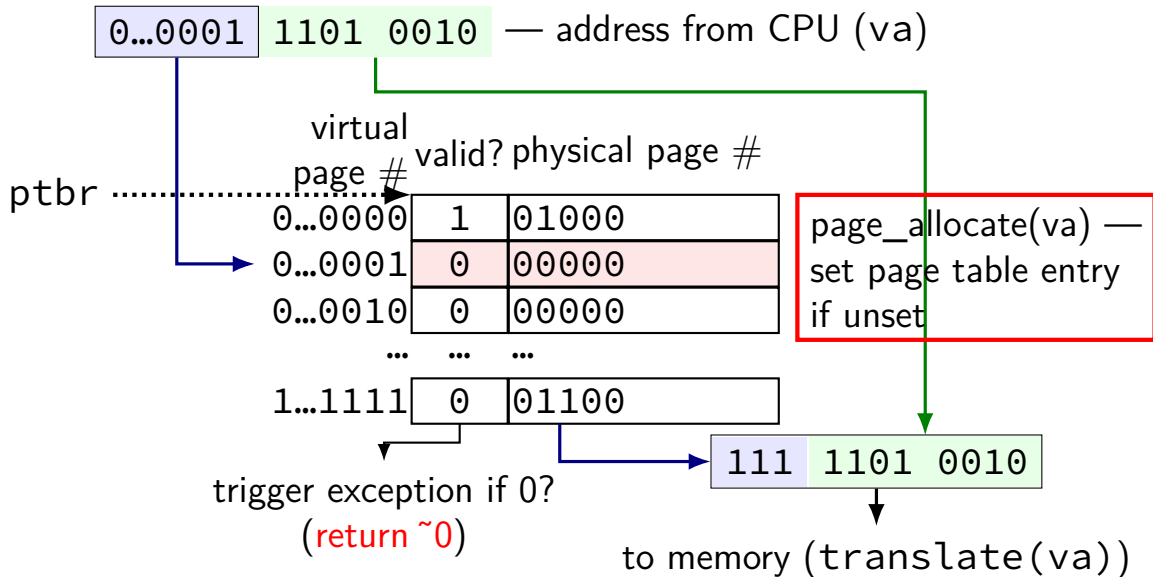
page table lookup (and translate())



page table lookup (and allocate)



page table lookup (and allocate)



exercise: 64-bit system

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

4096 byte pages

exercise: 64-bit system

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

4096 byte pages



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

exercise: 64-bit system

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?

exercise: 64-bit system

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?

exercise: 64-bit system

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

4096 byte pages

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

exercise: how large are physical page numbers?

page table entries are 8 bytes (room for expansion, metadata)

trick: power of two size makes table lookup faster

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

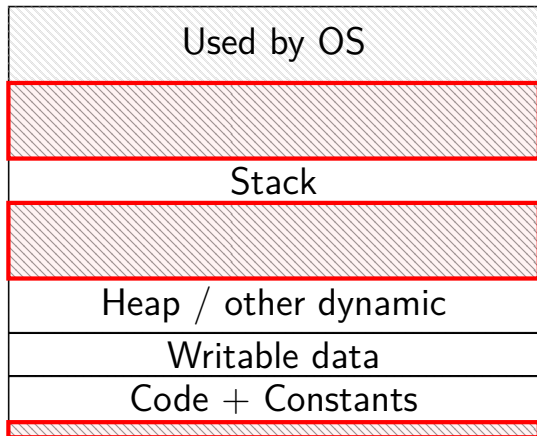
huge page tables

huge virtual address spaces!

impossible to store PTE for every page

how can we save space?

holes



most pages are **invalid**

saving space

basic idea: don't store (most) invalid page table entries

use a data structure other than a flat array

want a map — lookup key (virtual page number), get value (PTE)

options?

saving space

basic idea: don't store (most) invalid page table entries

use a data structure other than a flat array

want a map — lookup key (virtual page number), get value (PTE)

options?

hashtable

actually used by some historical processors

but never common

saving space

basic idea: don't store (most) invalid page table entries

use a data structure other than a flat array

want a map — lookup key (virtual page number), get value (PTE)

options?

hashtable

actually used by some historical processors
but never common

tree data structure

but not quite a search tree

search tree tradeoffs

lookup usually implemented in hardware

- lookup should be simple

- solution: lookup splits up address bits (no complex calculations)

lookup should not involve many memory accesses

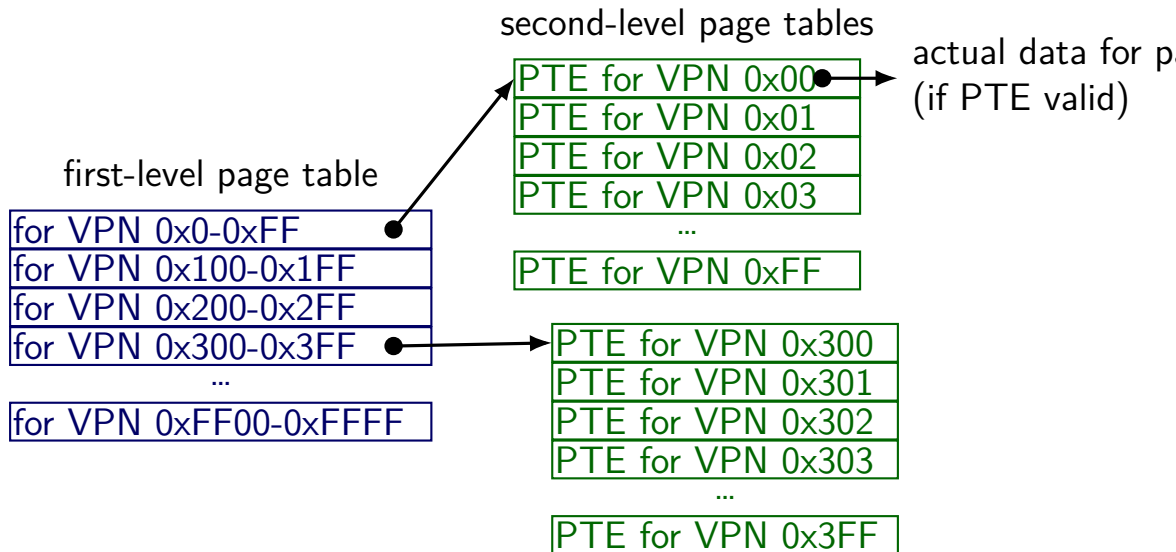
- doing two memory accesses is already very slow

- solution: tree with many children from each node

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

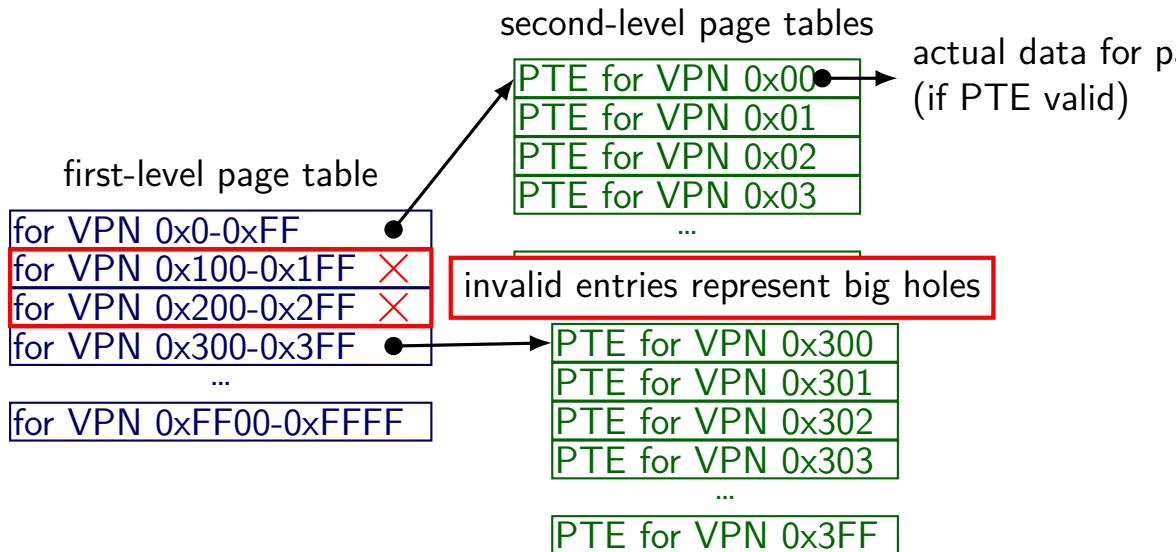
two-level page tables

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



two-level page tables

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



two-level page tables

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

first-level page table				for p d)
VPN range	valid	...	physical page # (of next page table)	
0x0000-0x00FF	1	...	0x22343	
0x0100-0x01FF	0	...	0x00000	
0x0200-0x02FF	0	...	0x00000	
0x0300-0x03FF	1	...	0x33454	
0x0400-0x04FF	1	...	0xFF043	
...	
0xFF00-0xFFFF	1	...	0xFF045	

first-level page table for VPN 0x000-0x00FF

first-level page table for VPN 0x100-0x10FF

first-level page table for VPN 0x200-0x20FF

first-level page table for VPN 0x300-0x30FF

...

first-level page table for VPN 0xFF00-0xFFFF

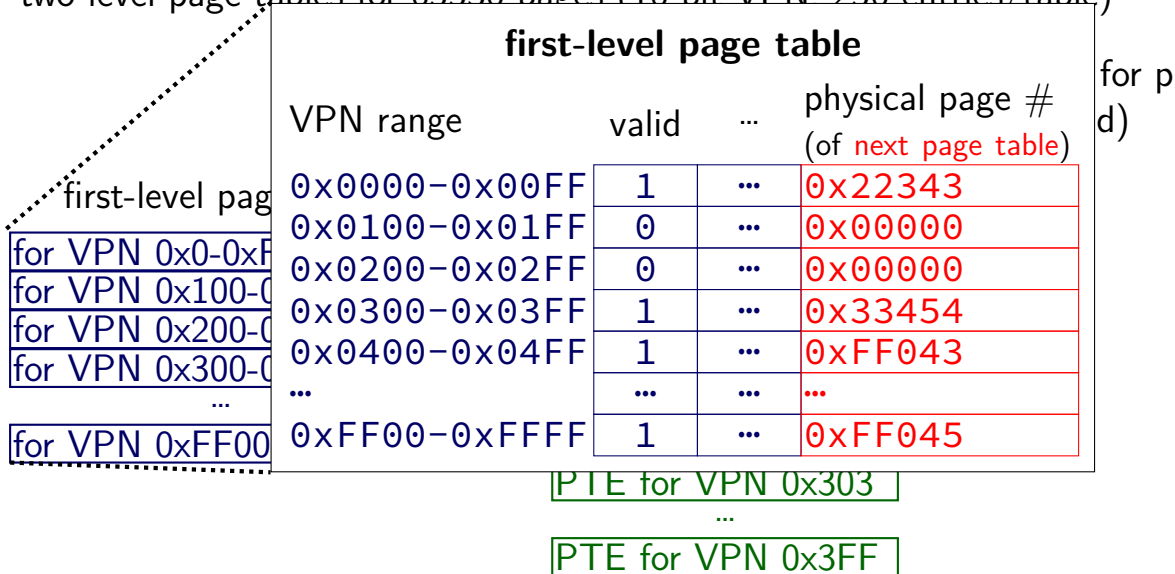
PTE for VPN 0x303

...

PTE for VPN 0x3FF

two-level page tables

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



two-level page tables

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

first-level page table				for p d)
VPN range	valid	...	physical page # (of next page table)	
0x0000-0x00FF	1	...	0x22343	
0x0100-0x01FF	0	...	0x00000	
0x0200-0x02FF	0	...	0x00000	
0x0300-0x03FF	1	...	0x33454	
0x0400-0x04FF	1	...	0xFF043	
...	
0xFF00-0xFFFF	1	...	0xFF045	

first-level page table for VPN 0x000-0x0FF

for VPN 0x100-0x1FF

for VPN 0x200-0x2FF

for VPN 0x300-0x3FF

...

for VPN 0xFF00-0xFFFF

PTE for VPN 0x303

...

PTE for VPN 0x3FF

two-level page tables

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

first-level page table

for VPN 0x0-0xFF
for VPN 0x100-0x1FF ✗
for VPN 0x200-0x2FF ✗
for VPN 0x300-0x3FF
...
for VPN 0xFF00-0xFFFF

a second-level page table

VPN	valid	...	physical page # (of data)
0x300	0	1	0x42443
0x301	0	1	0x4A9DE
0x302	0	1	0x5C001
0x303	0	1	0x00000
0x304	0	1	0x6C223
...
0x3FF	...	1	0x00000

PTE for VPN 0x303

...

PTE for VPN 0x3FF

or p
l)

two-level page tables

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

first-level page table

for VPN 0x0-0xFF	
for VPN 0x100-0x1FF	×
for VPN 0x200-0x2FF	×
for VPN 0x300-0x3FF	
...	
for VPN 0xFF00-0xFFFF	

a second-level page table

VPN	valid	...	physical page # (of data)
0x300	0	1	0x42443
0x301	0	1	0x4A9DE
0x302	0	1	0x5C001
0x303	0	1	0x00000
0x304	0	1	0x6C223
...
0x3FF	...	1	0x00000

PTE for VPN 0x303

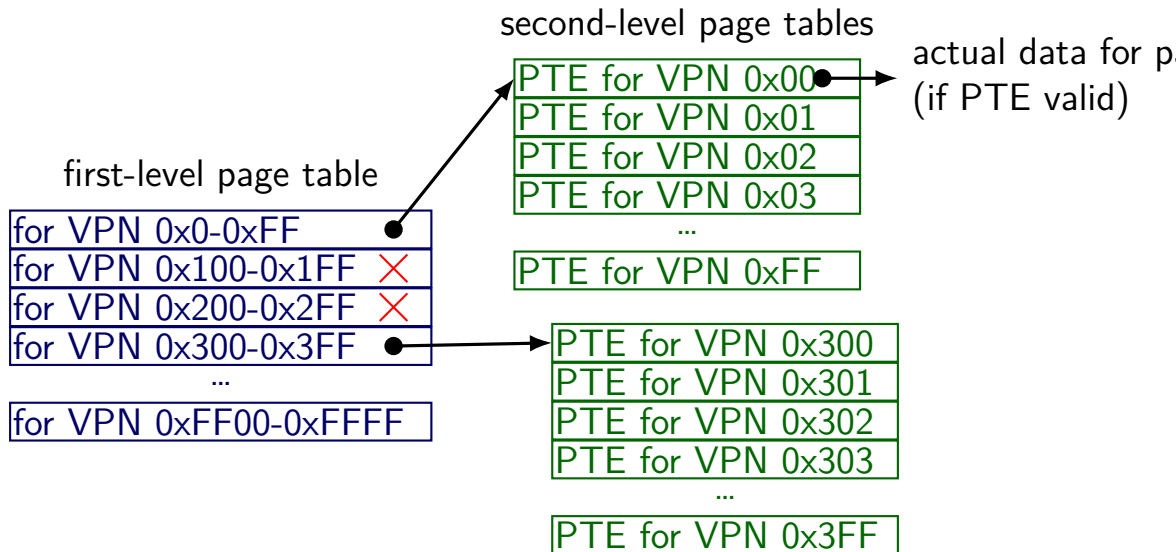
...

PTE for VPN 0x3FF

or p
l)

two-level page tables

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



two-level page table lookup

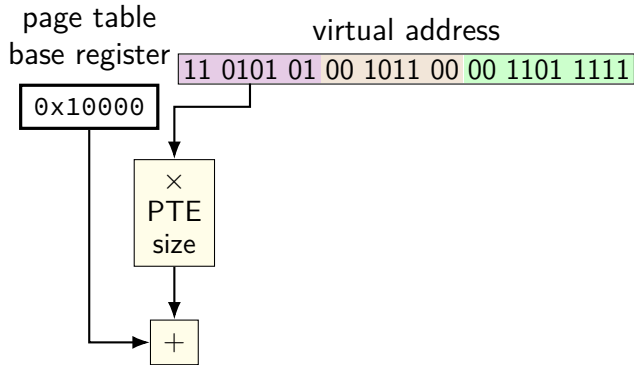
virtual address

11	0101	01	00	1011	00	00	1101	1111
----	------	----	----	------	----	----	------	------

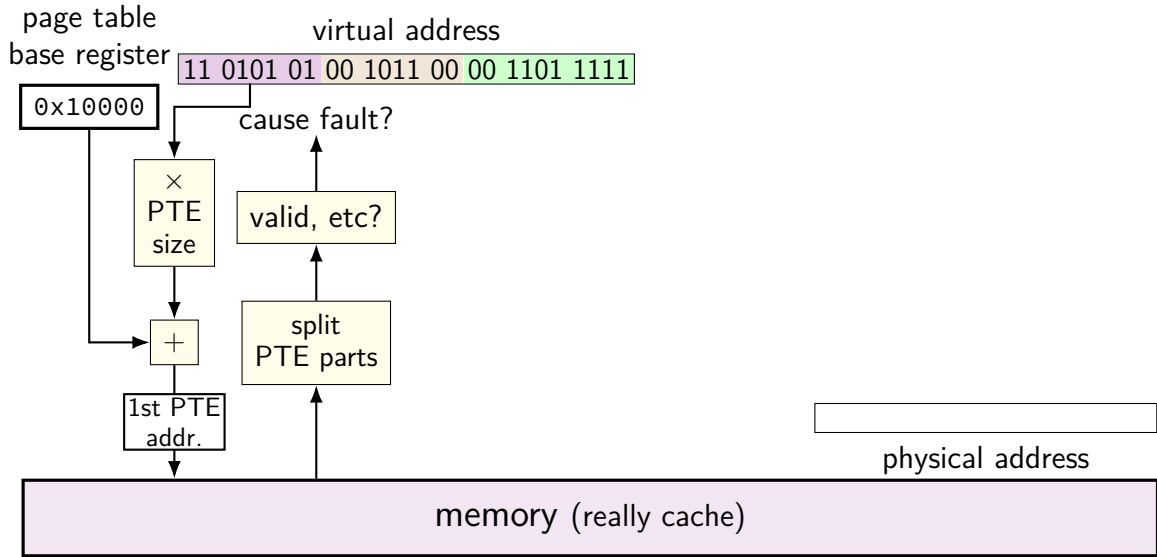
VPN — split into two parts (one per level)

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

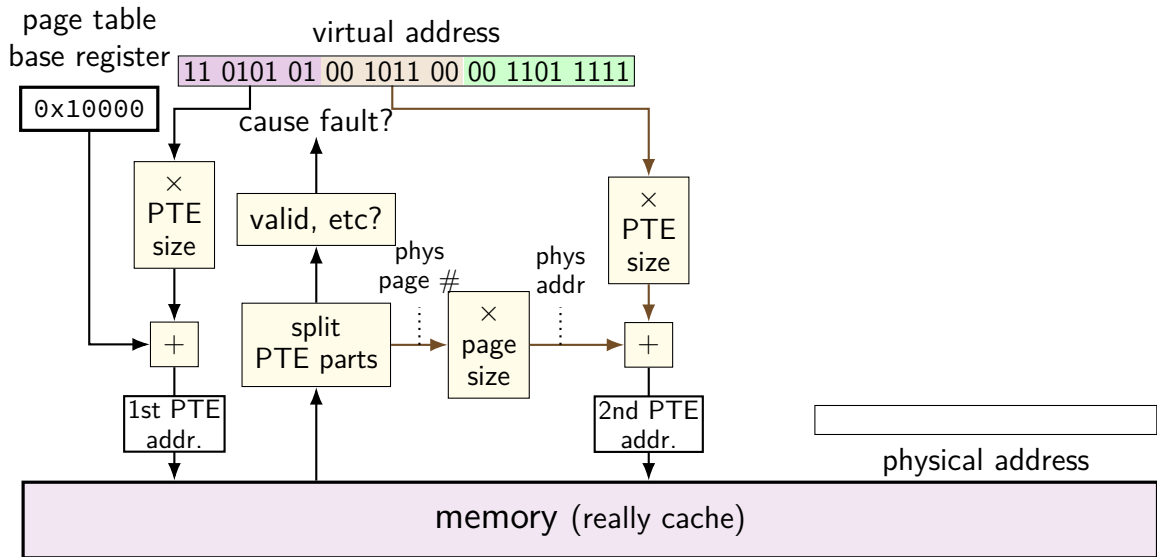
two-level page table lookup



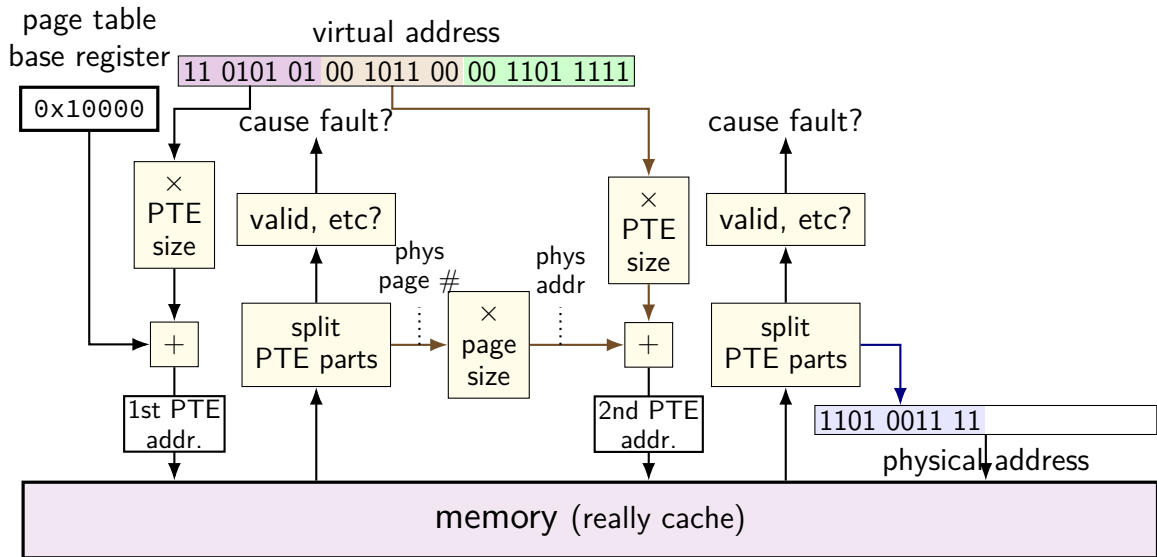
two-level page table lookup



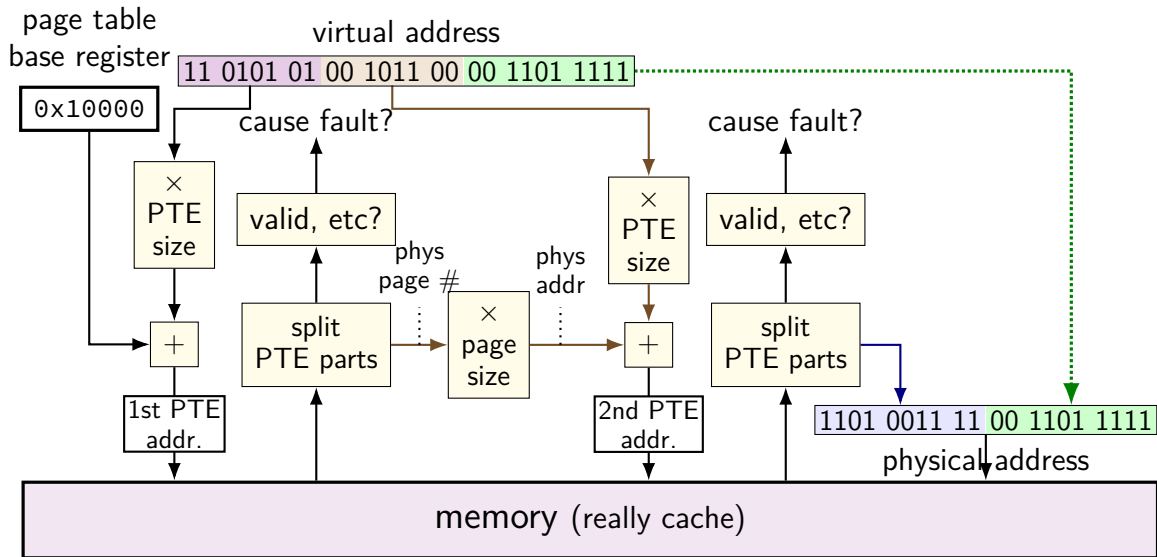
two-level page table lookup



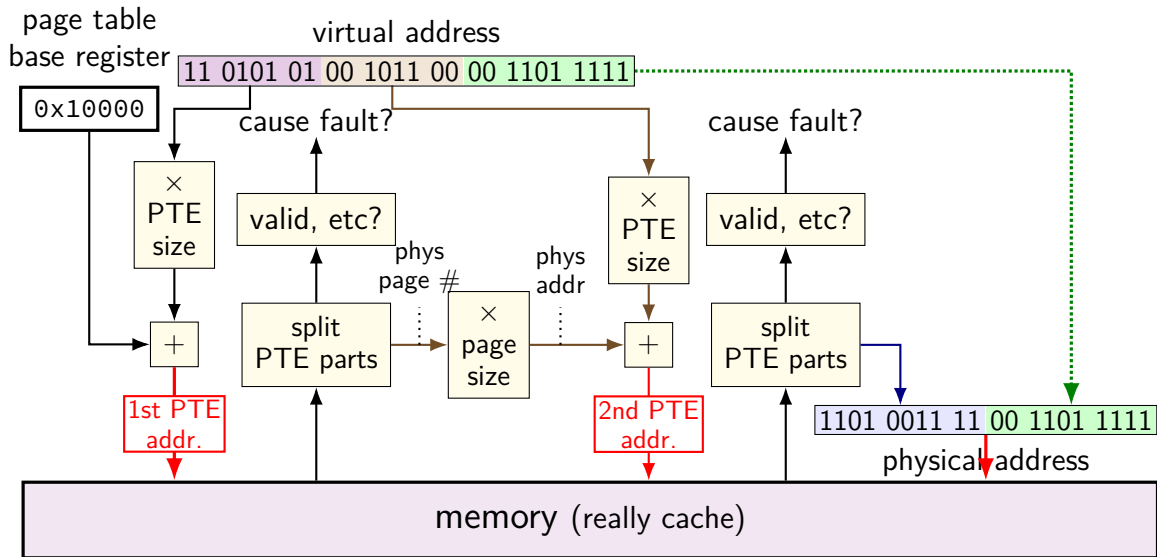
two-level page table lookup



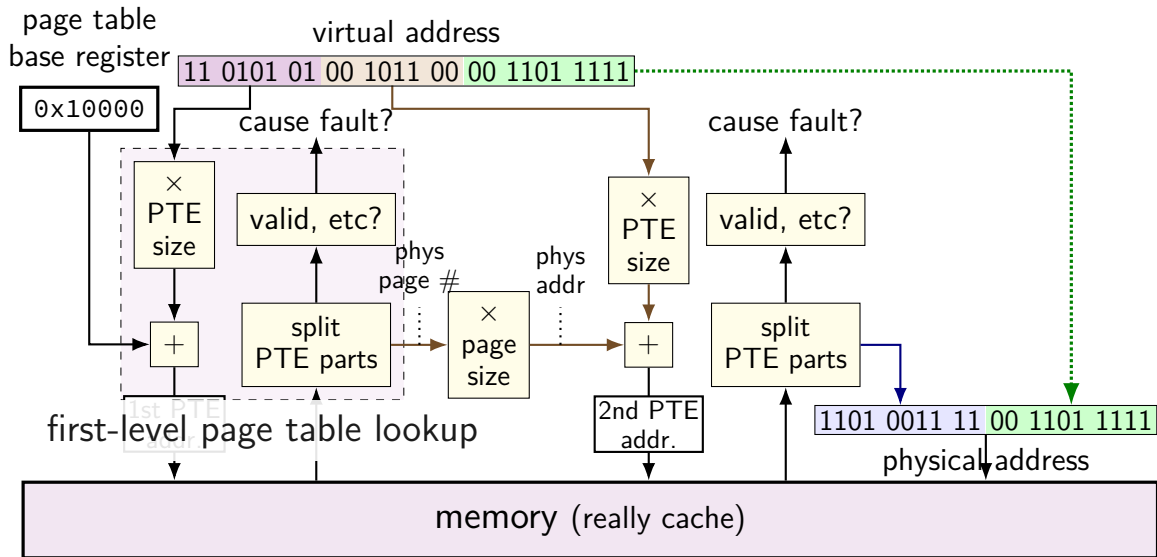
two-level page table lookup



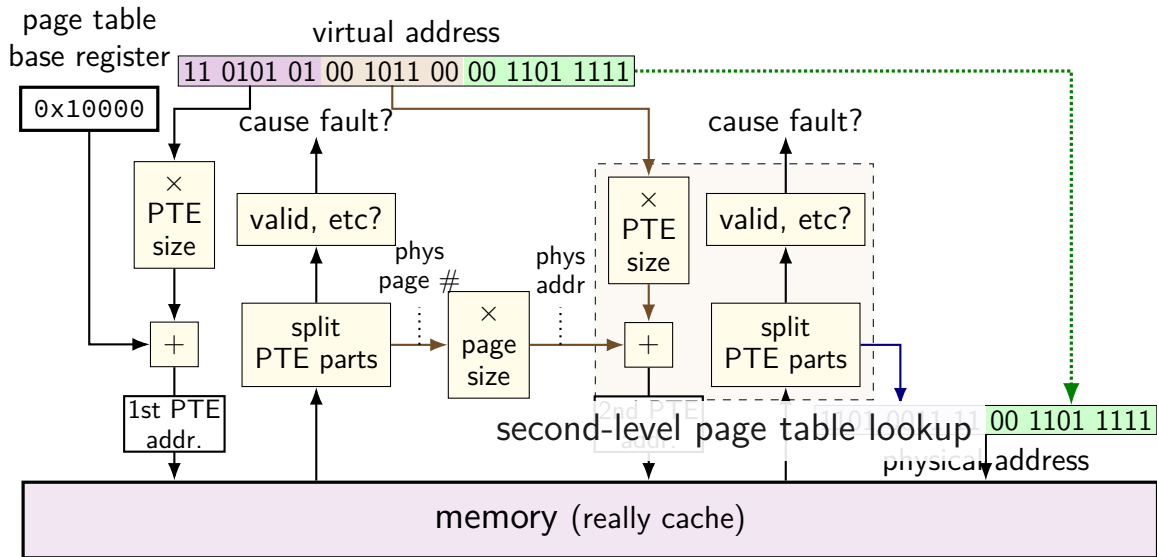
two-level page table lookup



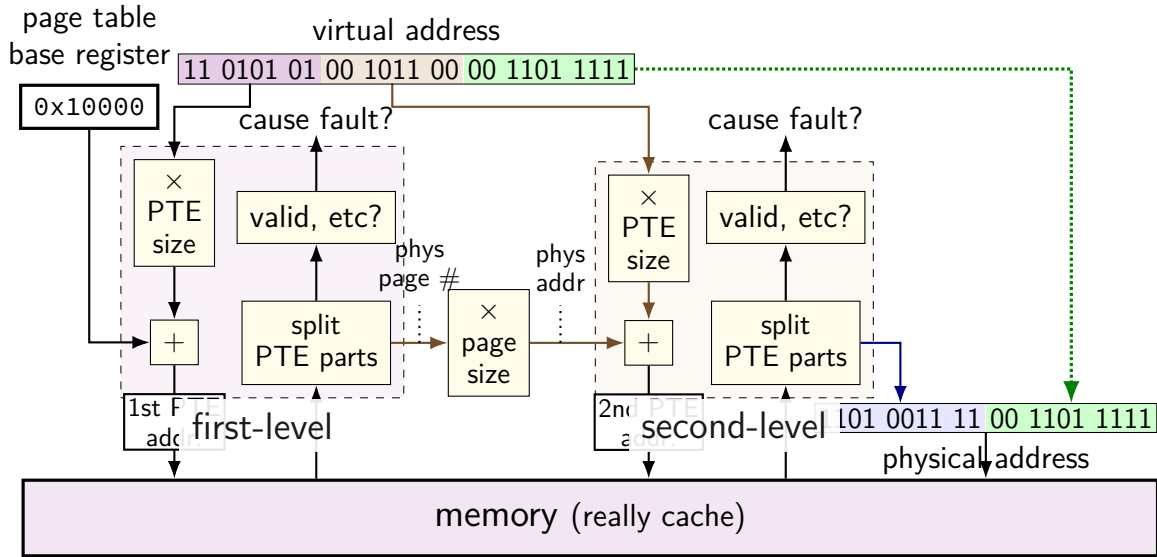
two-level page table lookup



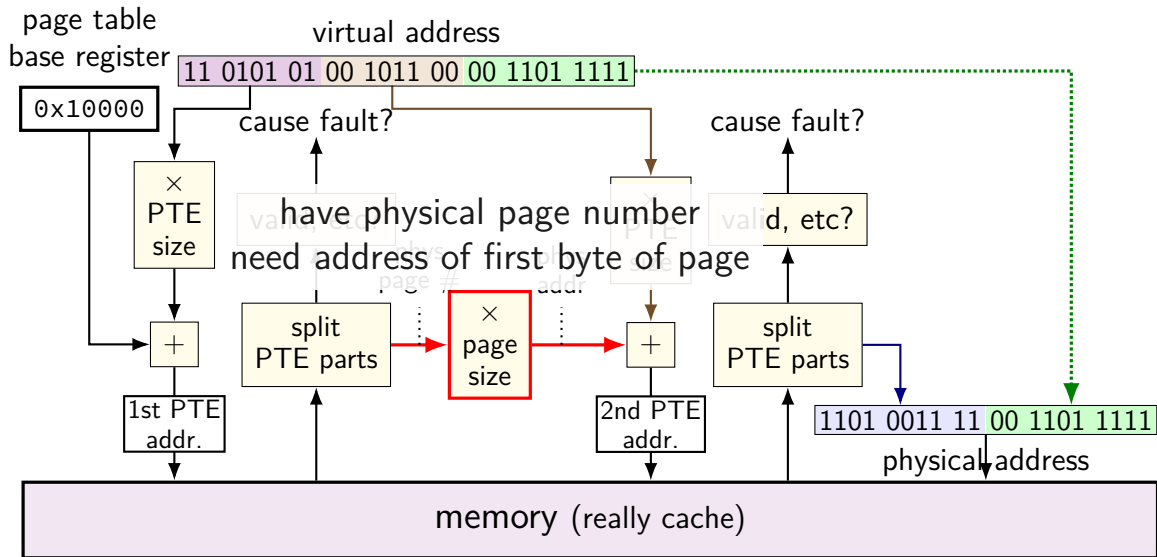
two-level page table lookup



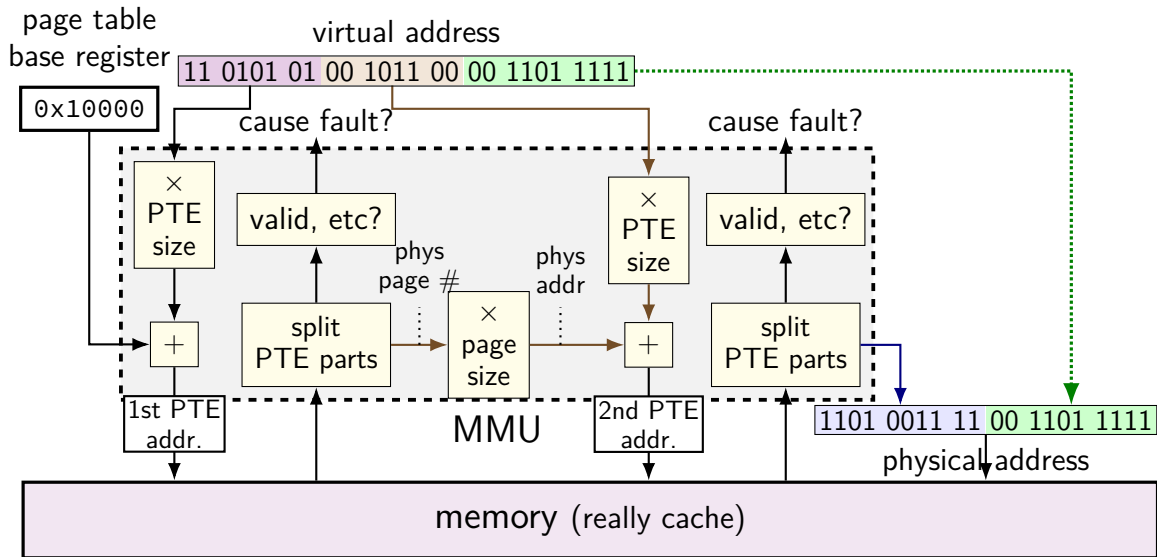
two-level page table lookup



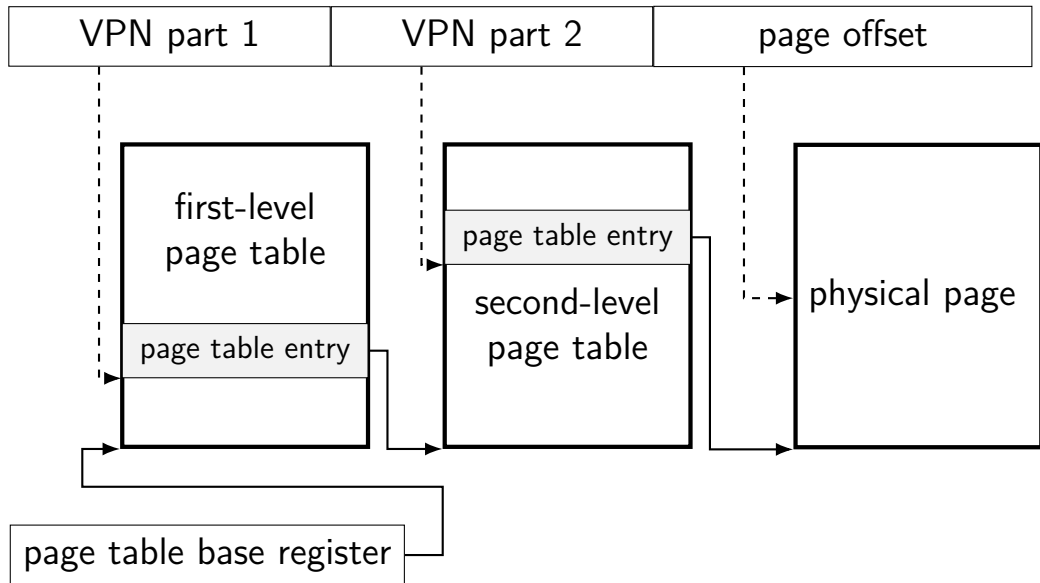
two-level page table lookup



two-level page table lookup



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

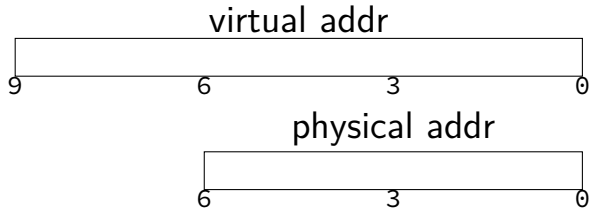
indexes used for lookup parts of the virtual page number
(there are not multiple VPNs)

assignment

2-level splitting

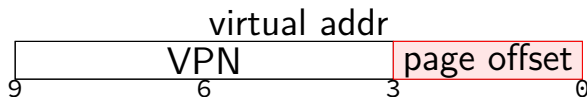
9-bit virtual address

6-bit physical address

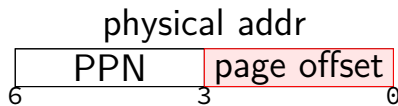


2-level splitting

9-bit virtual address



6-bit physical address



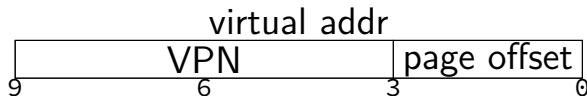
8-byte pages \rightarrow 3-bit page offset (bottom)

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

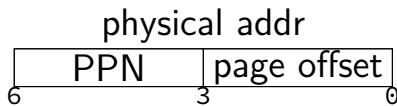
6-bit PA: 3 bit PPN + 3 bit PO

2-level splitting

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

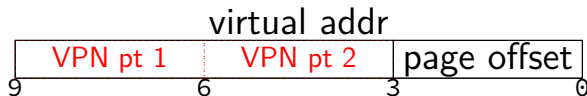
1 page page tables w/ 1 byte entry \rightarrow 8 entry PTs

page table (either level)

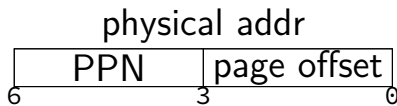
	valid? PPN	
0		
1		
2		
...
7		

2-level splitting

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	
0		
1		
2		
...
7		

8 entry page tables \rightarrow 3-bit VPN parts

9-bit VA: 3 bit VPN part 1; 3 bit VPN part 2

2-level example

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 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	bytes
0x20-3	00 91 72 13
0x24-7	F4 A5 36 07
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	AC DC DC 0C

2-level example

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 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	bytes
0x20-3	00 91 72 13
0x24-7	F4 A5 36 07
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	AC DC DC 0C

2-level example

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 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	bytes
0x20-3	00 91 72 13
0x24-7	F4 A5 36 07
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	AC DC DC 0C

2-level example

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 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	bytes
0x20-3	00 91 72 13
0x24-7	F4 A5 36 07
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	AC DC DC 0C

2-level example

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 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	bytes
0x20-3	00 91 72 13
0x24-7	F4 A5 36 07
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	AC 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 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	bytes
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

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 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	bytes
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

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 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	bytes
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

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 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	bytes
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

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 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	bytes
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

2-level exercise (2)

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

physical addresses	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 5A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
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

2-level exercise (3)

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

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
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	bytes
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

2-level exercise (3)

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

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
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	bytes
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

2-level exercise (3)

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

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
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	bytes
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

2-level exercise (4)

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	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	bytes
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

2-level exercise (5)

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

page table base register 0x10; translate virtual address 0x376

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	AC BC DC EC

physical addresses	bytes
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

2-level exercise (5)

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

page table base register 0x10; translate virtual address 0x376

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	AC BC DC EC

physical addresses	bytes
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

2-level exercise (5)

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

page table base register 0x10; translate virtual address 0x376

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	AC BC DC EC

physical addresses	bytes
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

2-level exercise (5)

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

page table base register 0x10; translate virtual address 0x376

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	AC BC DC EC

physical addresses	bytes
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

2-level exercise (5)

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

page table base register 0x10; translate virtual address 0x376

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	AC BC DC EC

physical addresses	bytes
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

2-level exercise (5)

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

page table base register 0x10; translate virtual address 0x376

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	AC BC DC EC

physical addresses	bytes
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

2-level exercise (5)

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

page table base register 0x10; translate virtual address 0x376

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	AC BC DC EC

physical addresses	bytes
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

2-level exercise (5)

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

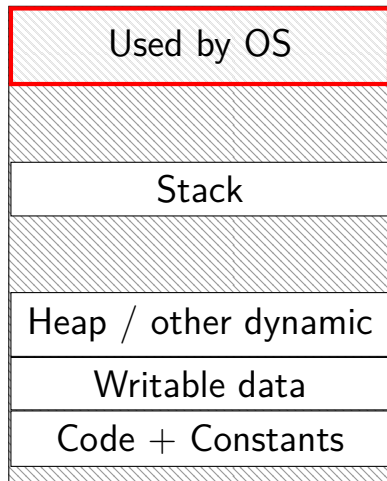
page table base register 0x10; translate virtual address 0x376

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	AC BC DC EC

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

backup slides

program memory



0xFFFF FFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F...

0x0000 0000 0040 0000

running OS code

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

running OS code

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

where in memory is this OS code?

running 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

running OS code

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

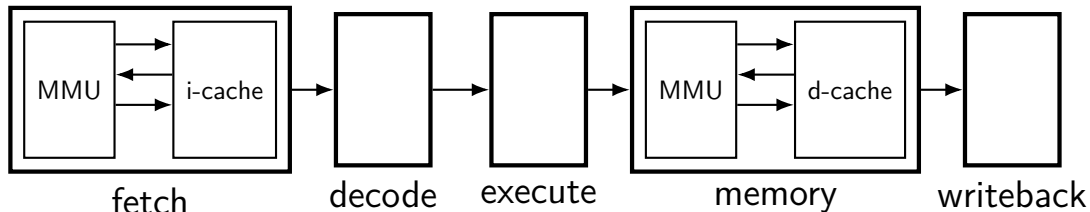
where in memory is this OS code?

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

code better not be modified by user program

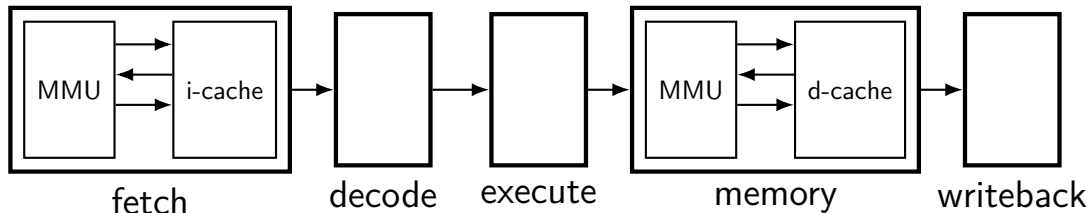
- otherwise: uncontrolled way to “escape” user mode

MMUs in the pipeline



up to four memory accesses per instruction

MMUs in the pipeline

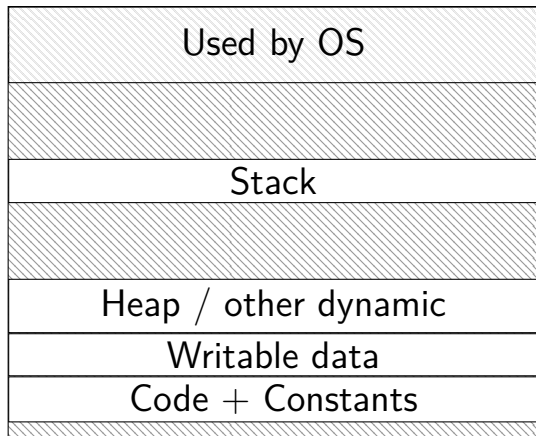


up to four memory accesses per instruction

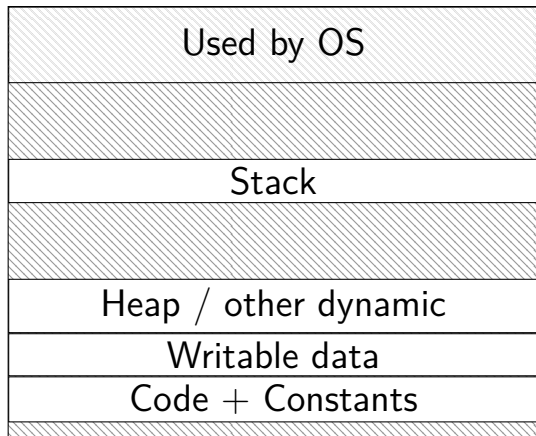
challenging to make this fast (topic for a future date)

do we really need a complete copy?

bash

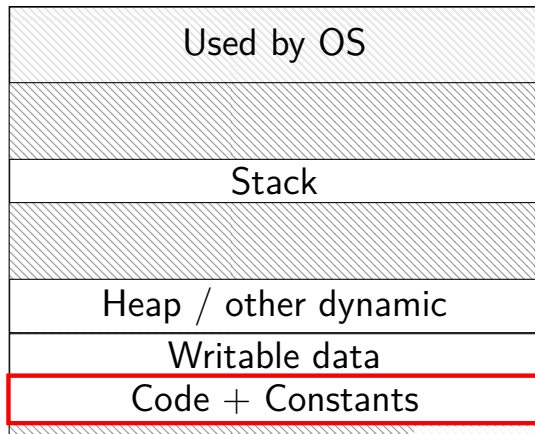


new copy of bash

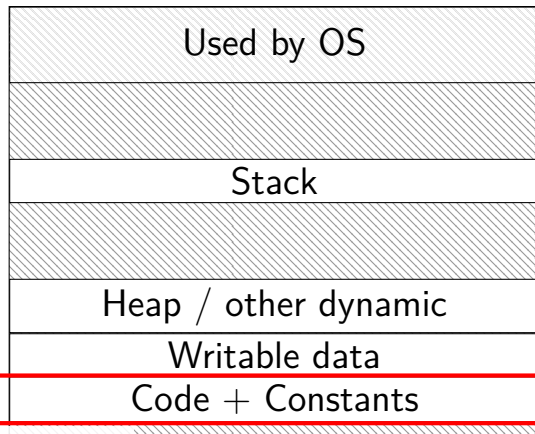


do we really need a complete copy?

bash



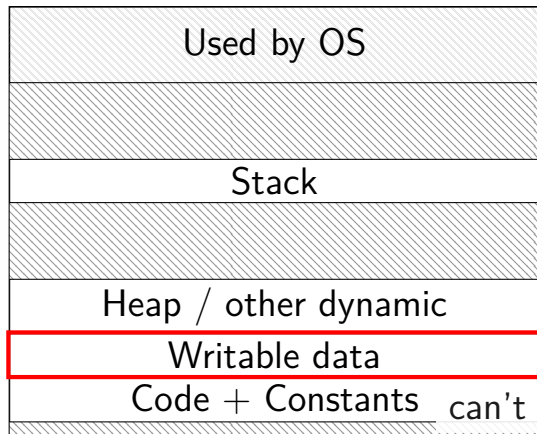
new copy of bash



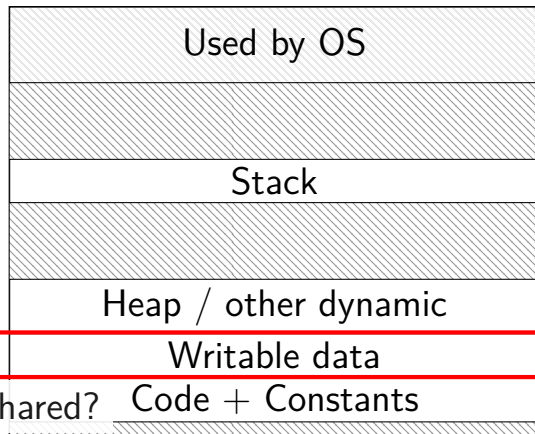
shared as read-only

do we really need a complete copy?

bash



new copy of bash



can't be shared?

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

copy-on-write and page tables

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

copy-on-write and page tables

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

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

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

copy-on-write and page tables

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

VPN	valid?	write?	physical 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

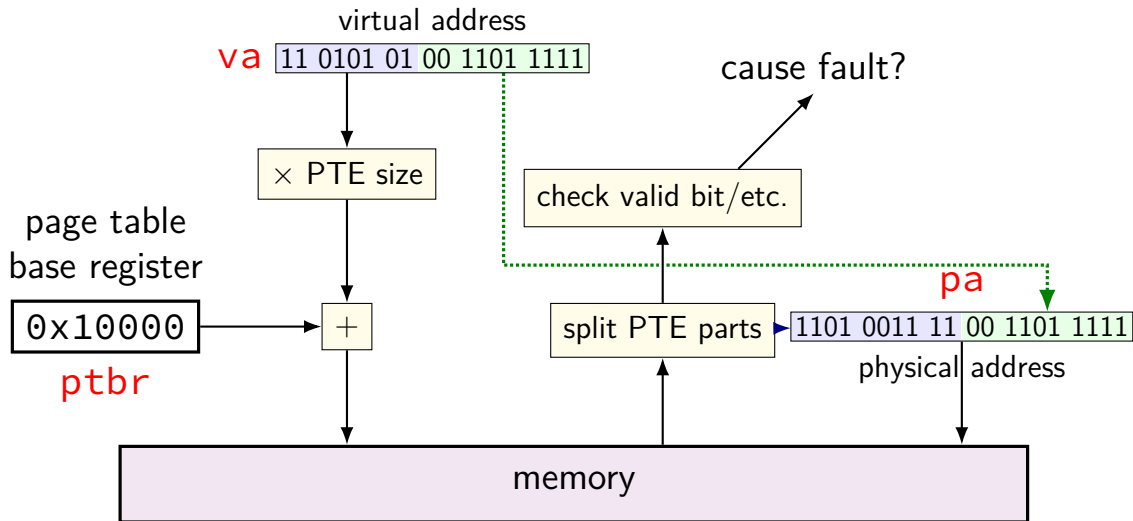
copy-on-write and page tables

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

VPN	valid?	write?	physical page
...
0x00601	1	0	0x12345
0x00602	1	0	0x12347
0x00603	1	0	0x12340
0x00604	1	0	0x200DF
0x00605	1	1	0x300FD
...

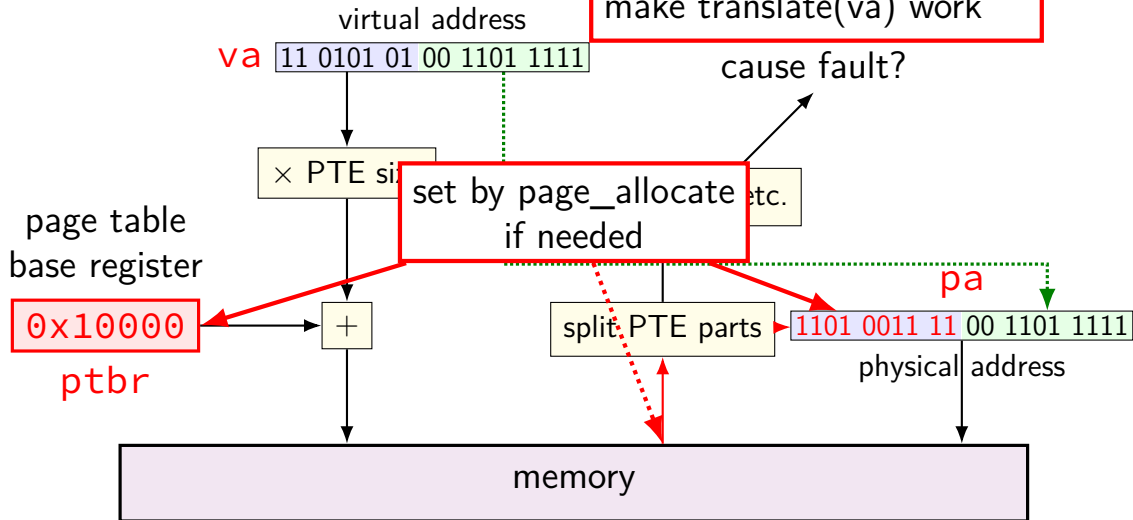
after allocating a copy, OS reruns the write instruction

pa=translate(va)



pa=translate(va)

page_allocate(va) needs to make translate(va) work



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/SDDs are slow

HDD reads and writes: milliseconds to tens of milliseconds

- minimum size: 512 bytes

- writing tens of kilobytes basically as fast as writing 512 bytes

SSD reads and writes: hundreds of microseconds

- designed for writes/reads of kilobytes (not much smaller)

HDD/SDDs are slow

HDD reads and writes: **milliseconds to tens of milliseconds**

- minimum size: 512 bytes

- writing tens of kilobytes basically as fast as writing 512 bytes

SSD reads and writes: **hundreds of microseconds**

- designed for reads/writes of kilobytes (not much smaller)

HDD/SDDs are slow

HDD reads and writes: **milliseconds to tens of milliseconds**

- minimum size: 512 bytes

- writing tens of kilobytes basically as fast as writing 512 bytes

SSD reads and writes: **hundreds of microseconds**

- designed for reads/writes of kilobytes (not much smaller)

HDD/SDDs are slow

HDD reads and writes: milliseconds to tens of milliseconds

minimum size: 512 bytes

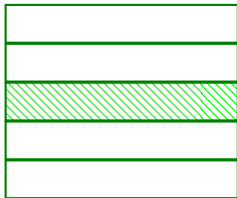
writing tens of **kilobytes** basically as fast as writing 512 bytes

SSD reads and writes: hundreds of microseconds

designed for reads/writes of **kilobytes** (not much smaller)

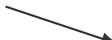
swapping timeline

program A pages

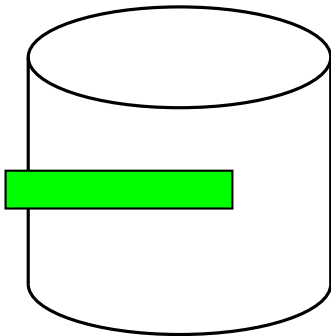


...

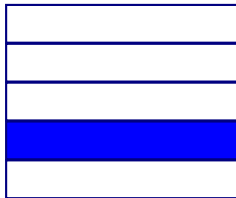
page fault



disk

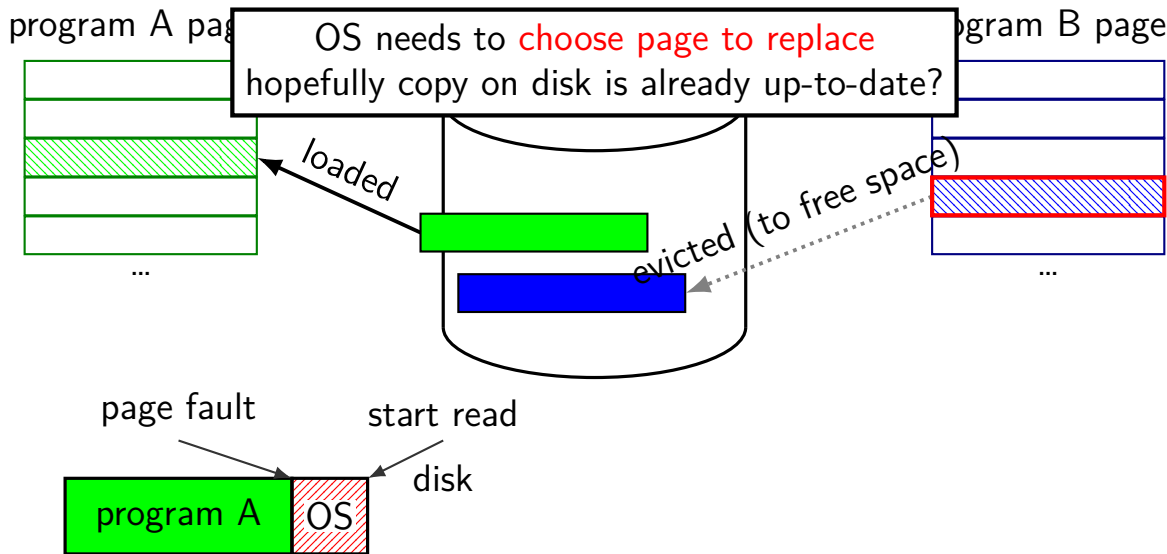


program B page

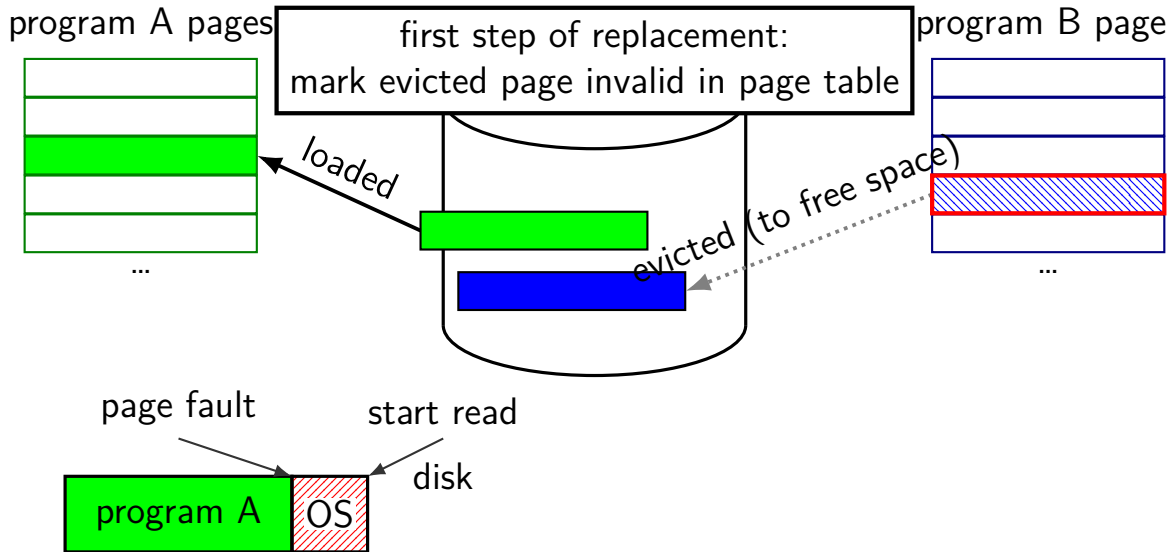


...

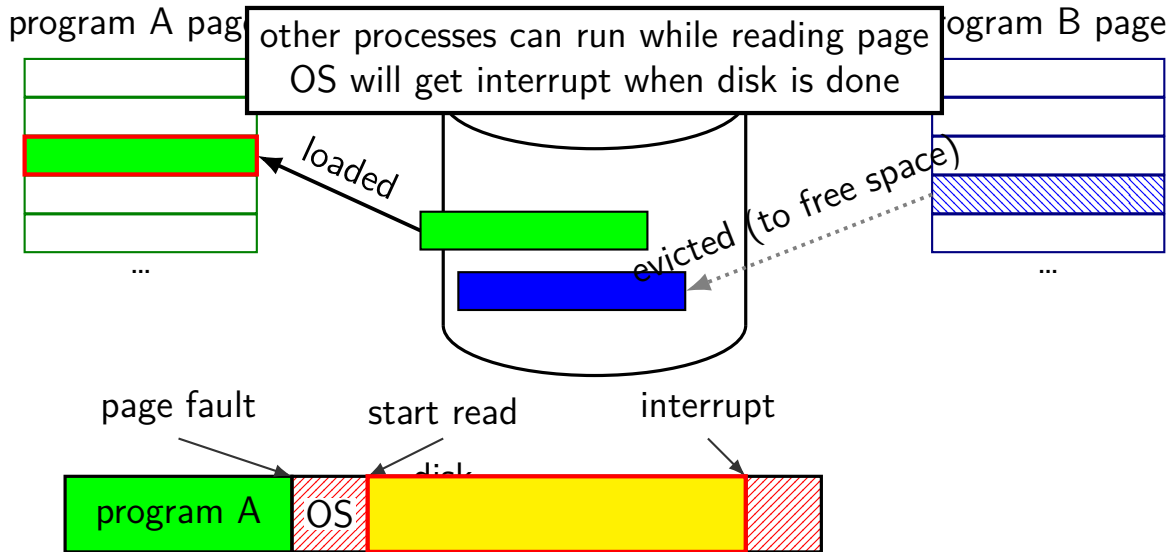
swapping timeline



swapping timeline

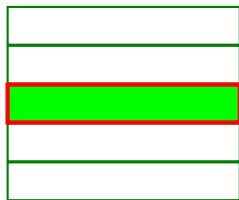


swapping timeline



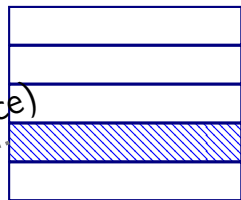
swapping timeline

program A pages



process A's page table updated
and restarted from point of fault

program B page



loaded

evicted (to free space)

page fault

start read

interrupt



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**

backup slides