Virtual Memory

page tables

program memory = virtual; real memory = physical

each memory divided into fixed-sizes pages

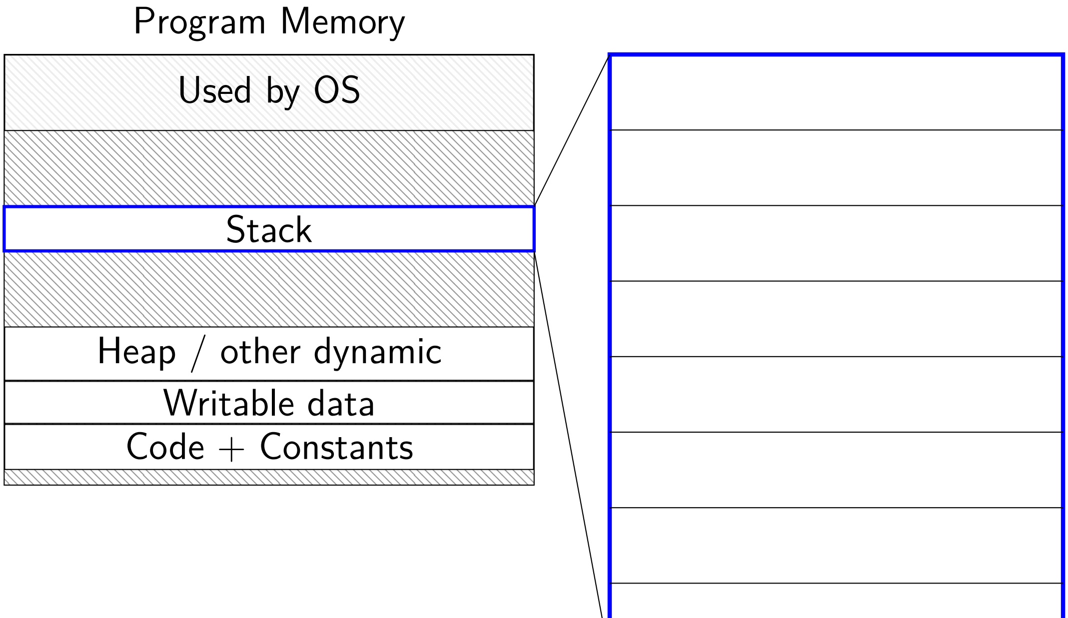
each virtual page has a page table entry:

- has location of physical page (if any)
- has valid bit
- has permission bits

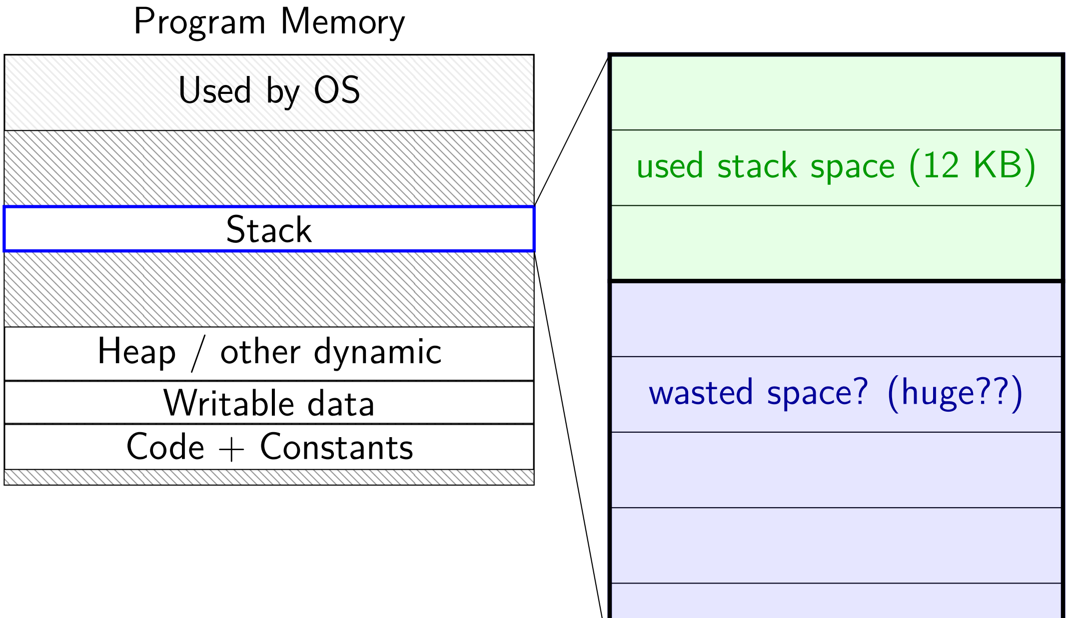
all page table entries stored in page table

permission or validity error triggers exception

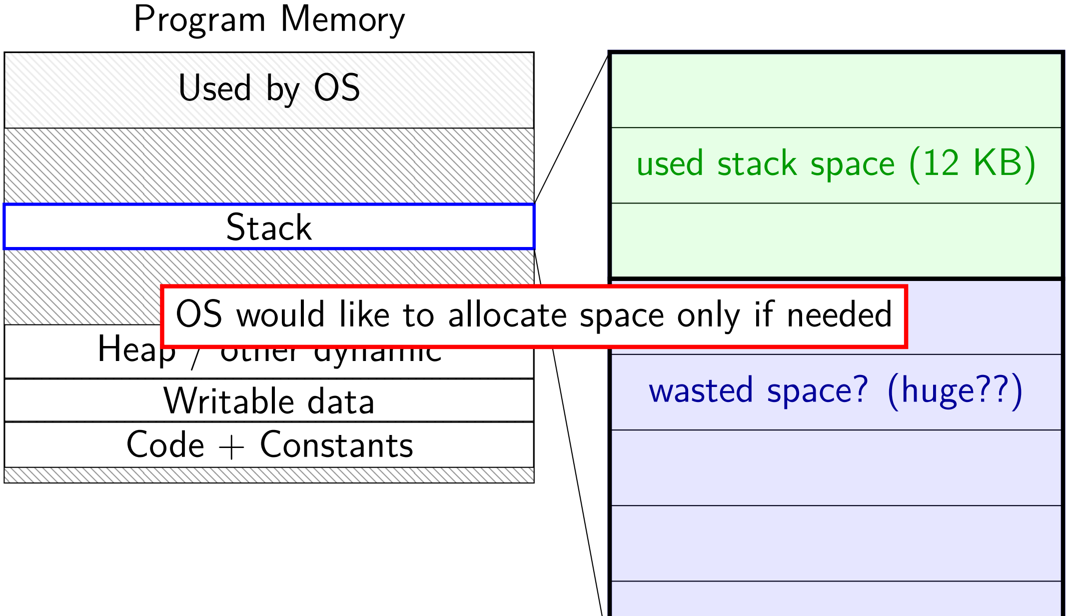
space on demand



space on demand



space on demand



OS would like to allocate space only if needed

allocating space on demand

`%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
…	…	…

allocating space on demand

```
%rsp = 0x7FFFC000
```

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

page fault!

VPN	valid?	physical page
...	...	...
0x7FFFB	0	---
0x7FFFC	1	0x200DF
0x7FFFD	1	0x12340
0x7FFFE	1	0x12347
0x7FFFF	1	0x12345
...	...	...

pushq triggers exception

hardware says "accessing address 0x7FFFBFF8"

OS looks up what's should be there — "stack"

allocating space on demand

%rsp = 0x7FFFC000

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

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

restarted

VPN	valid?	physical 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

allocating space on demand

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

fast copies

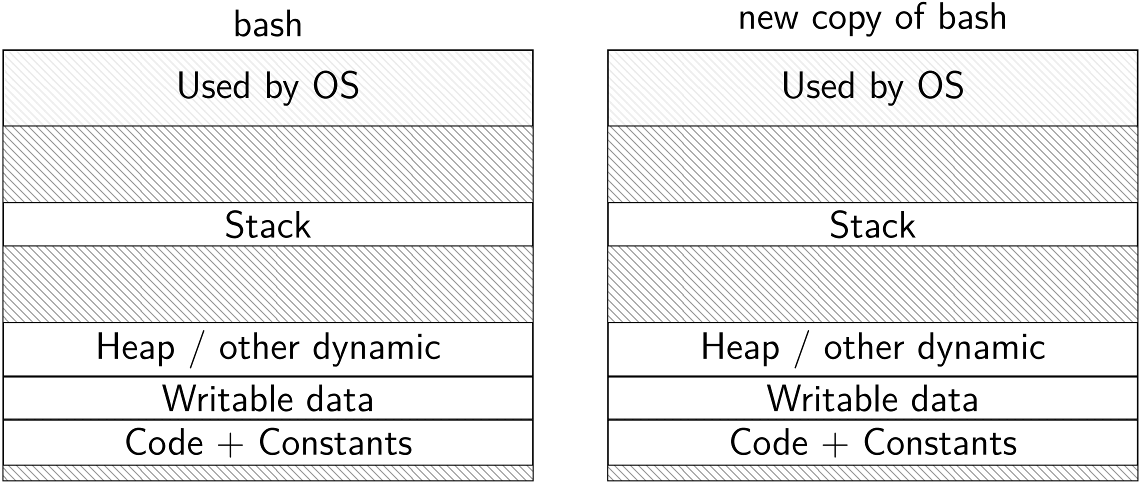
Unix mechanism for starting a new process: `fork()`

creates a copy of an entire program!

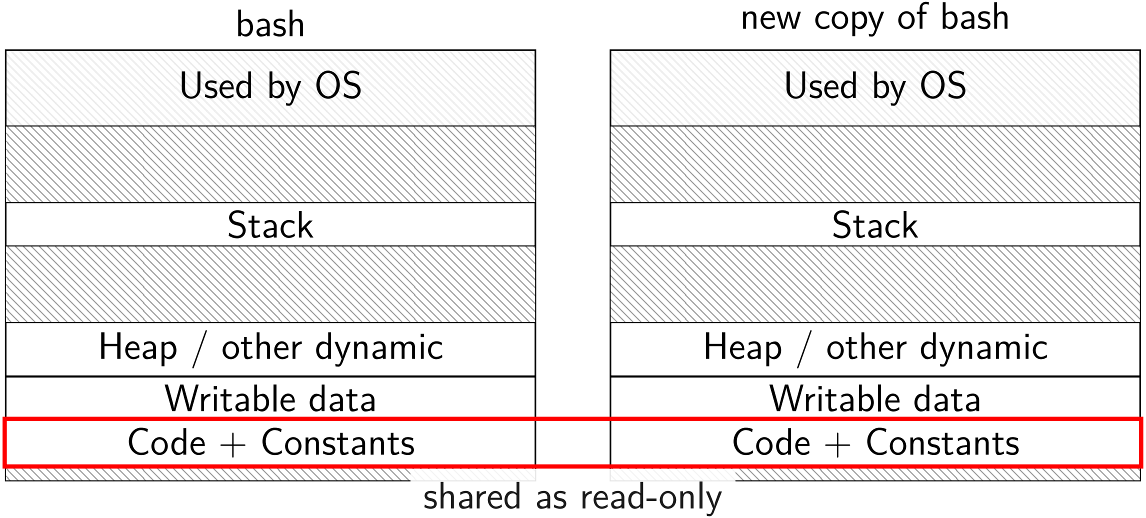
(usually, the copy then calls `execve` — replaces itself with another program)

how isn't this really slow?

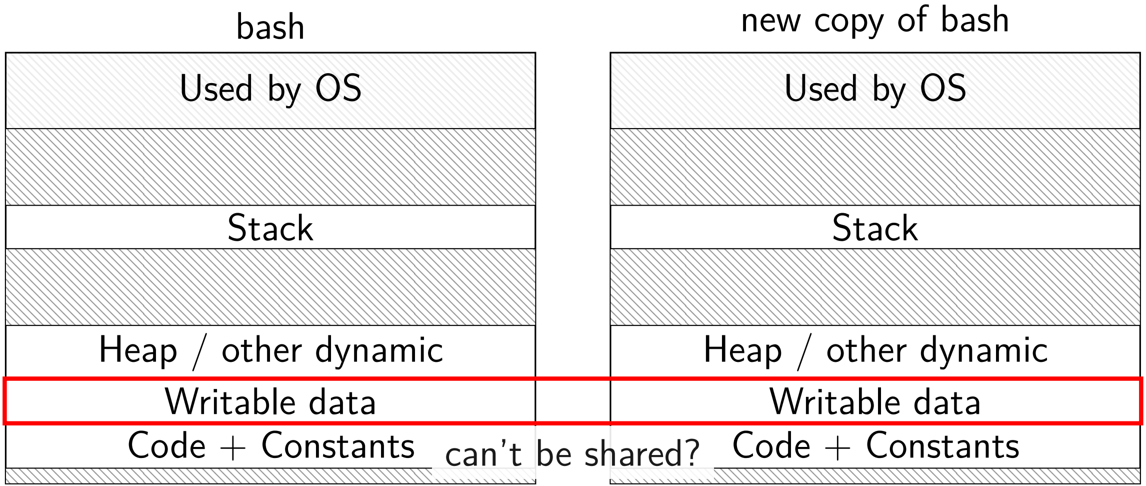
do we really need a complete copy?



do we really need a complete copy?



do we really need a complete copy?



trick for extra sharing

sharing writeable data is fine — until either process modifies the copy

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

swapping

our textbook presents virtual memory to support swapping

using disk (or SSD, ...) as the next level of the memory hierarchy

OS allocates space on disk

DRAM is a cache for disk

swapping versus caching

"cache block" $\approx$ physical page

fully associative

- every virtual page can be stored in any physical page

replacement is managed by the OS

normal cache hits happen without OS

- common case that needs to be fast

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 writes 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 writes and writes: hundreds of microseconds

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

HDD/SDDs are slow

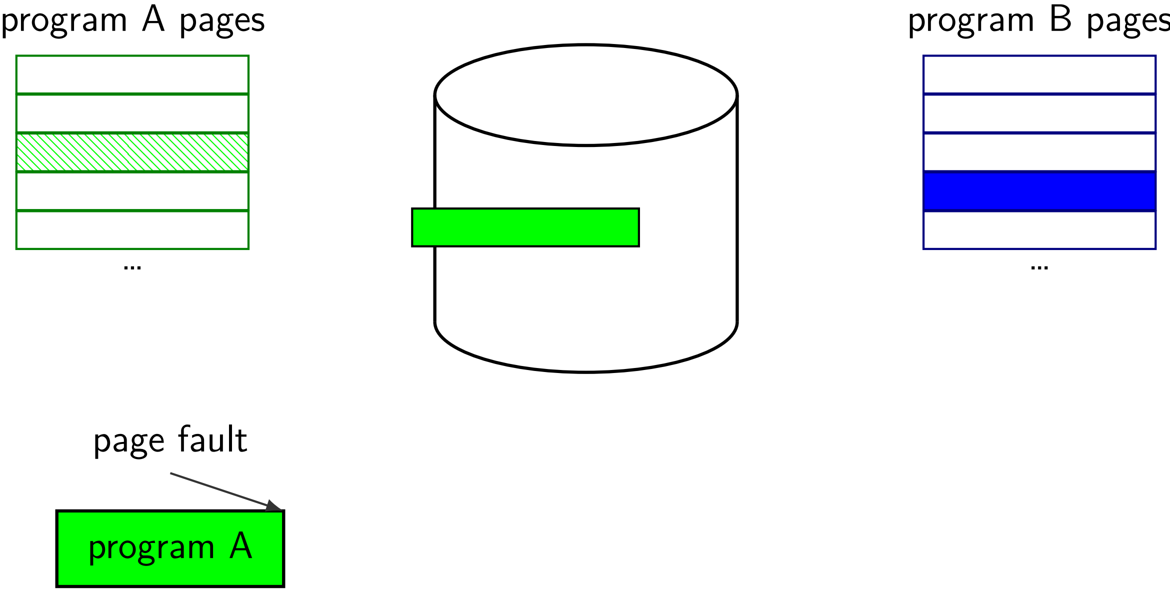
HDD reads and writes: milliseconds to tens of milliseconds

- minimum size: 512 bytes
- writing tens of kilobytes basically as fast as writing 512 bytes

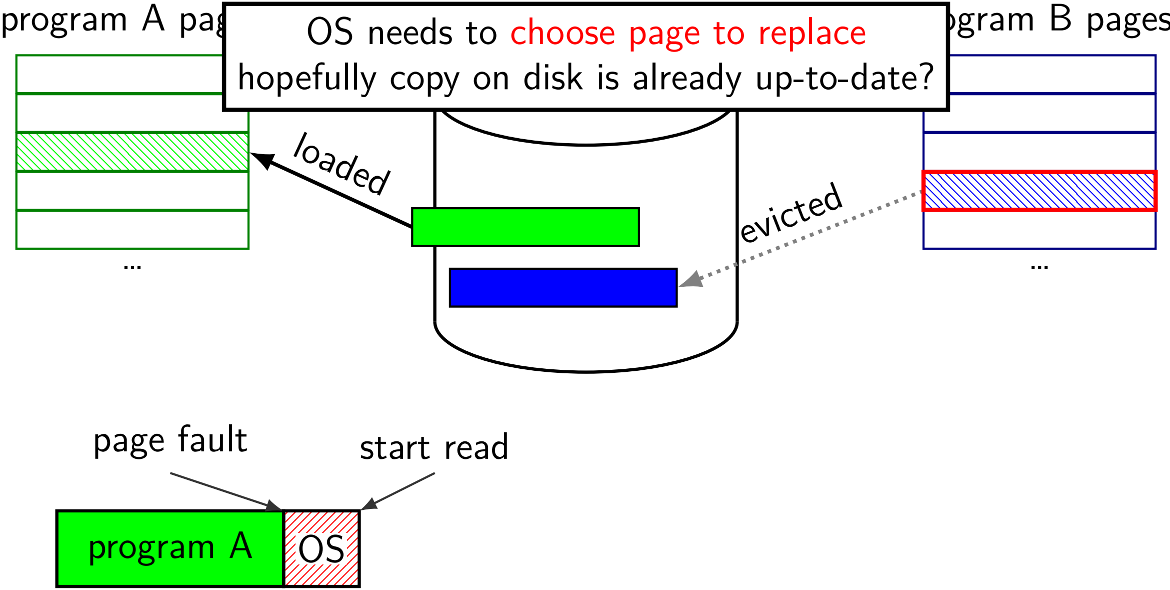
SSD writes and writes: hundreds of microseconds

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

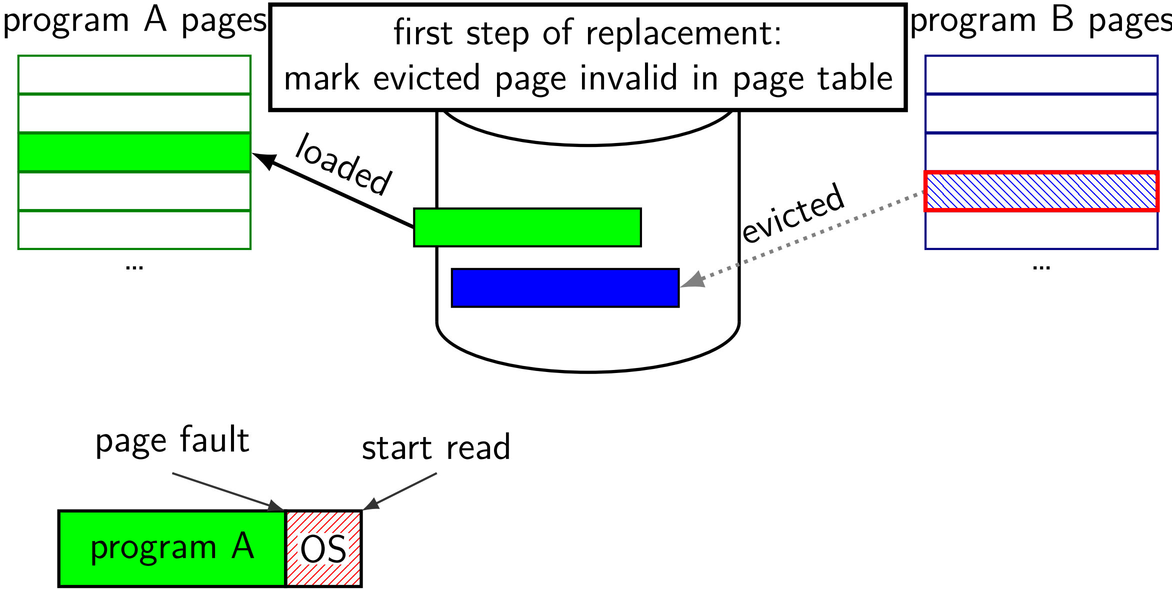
swapping timeline



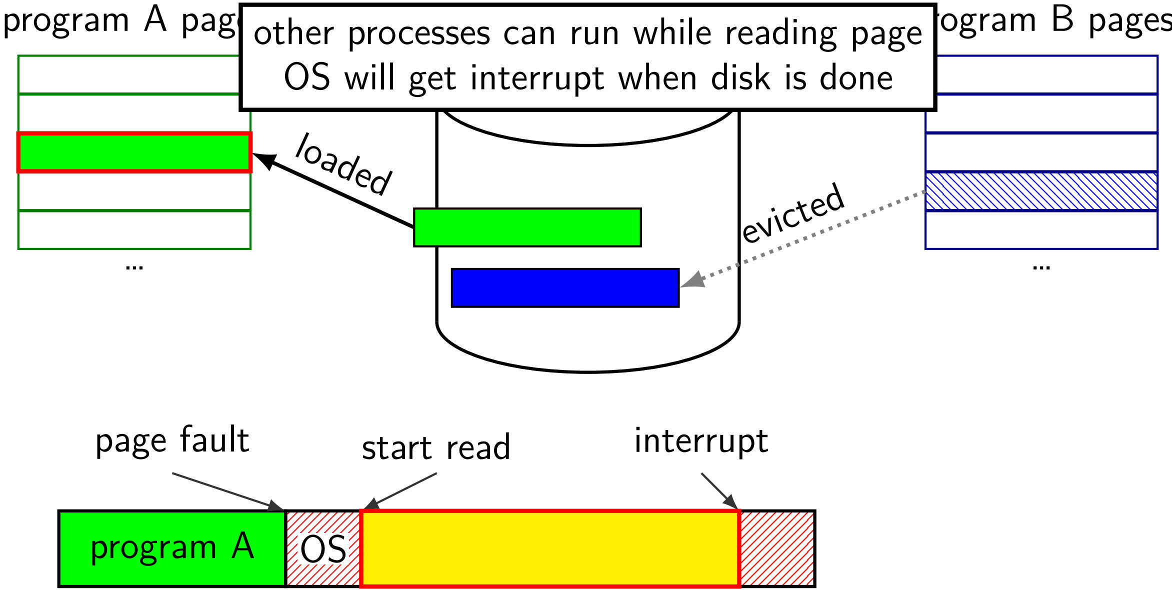
swapping timeline



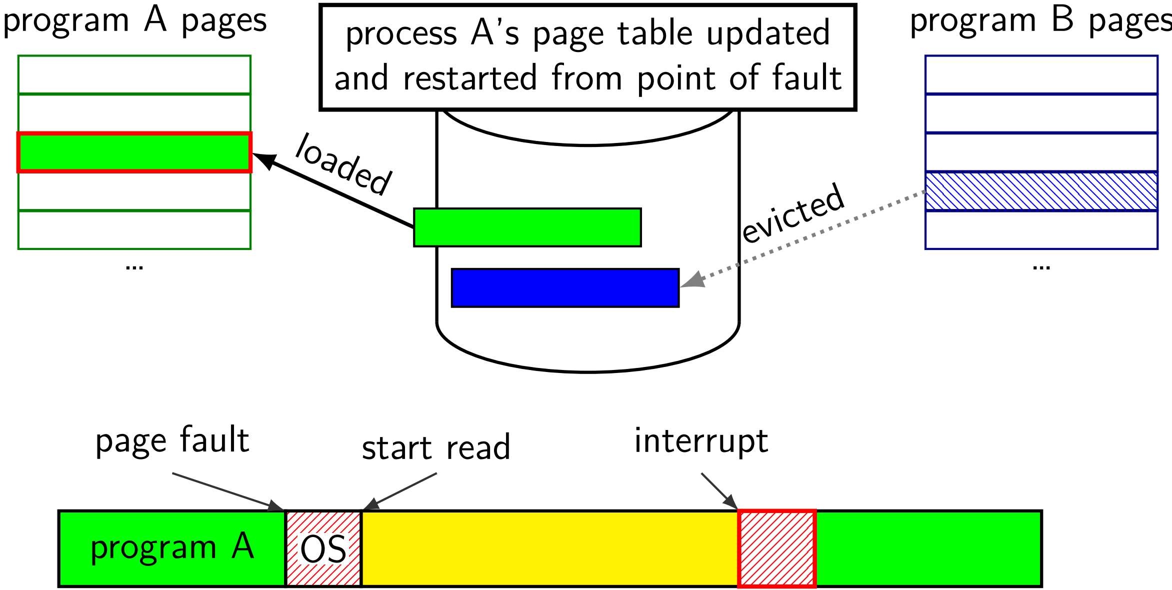
swapping timeline



swapping timeline



swapping timeline



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 address 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?

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

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

exercise: 64-bit system

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

4096 byte pages

exercise: how many page table entries? $2^{48}/2^{12} = 2^{36}$ entries

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

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

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

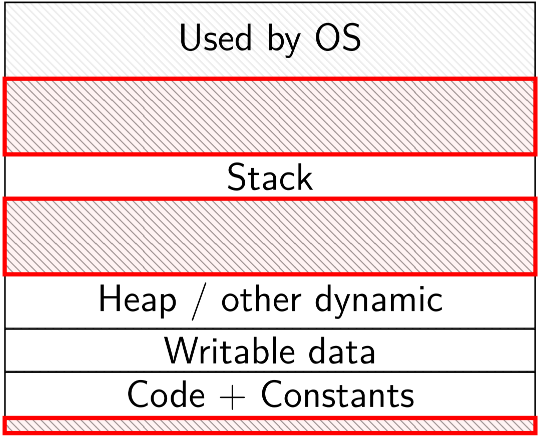
huge page tables

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

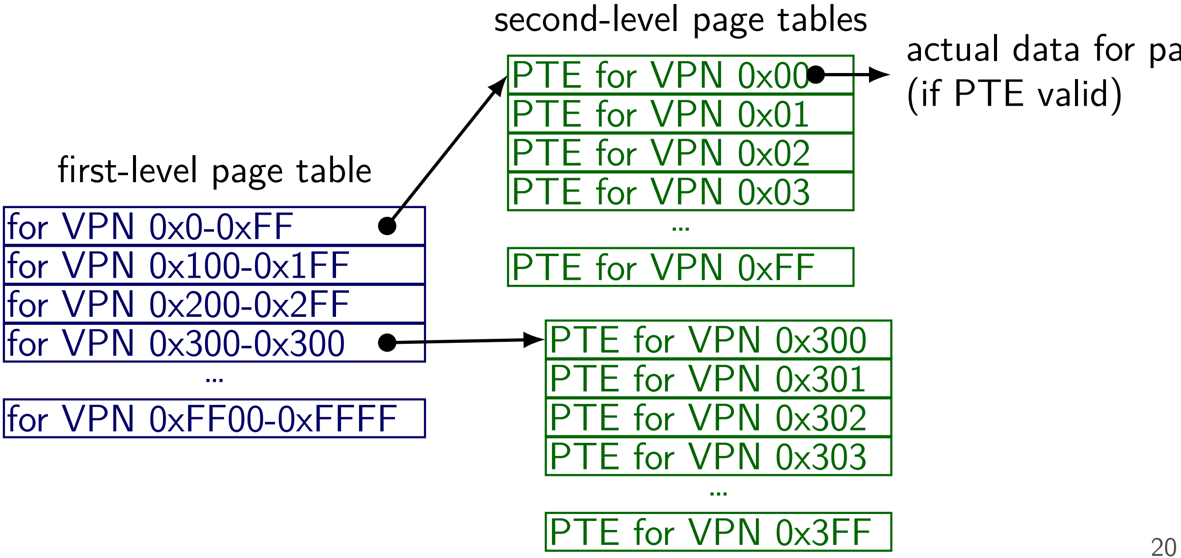
- lookup should be simple

lookup should not involve many memory accesses

- doing two memory accesses is already very slow

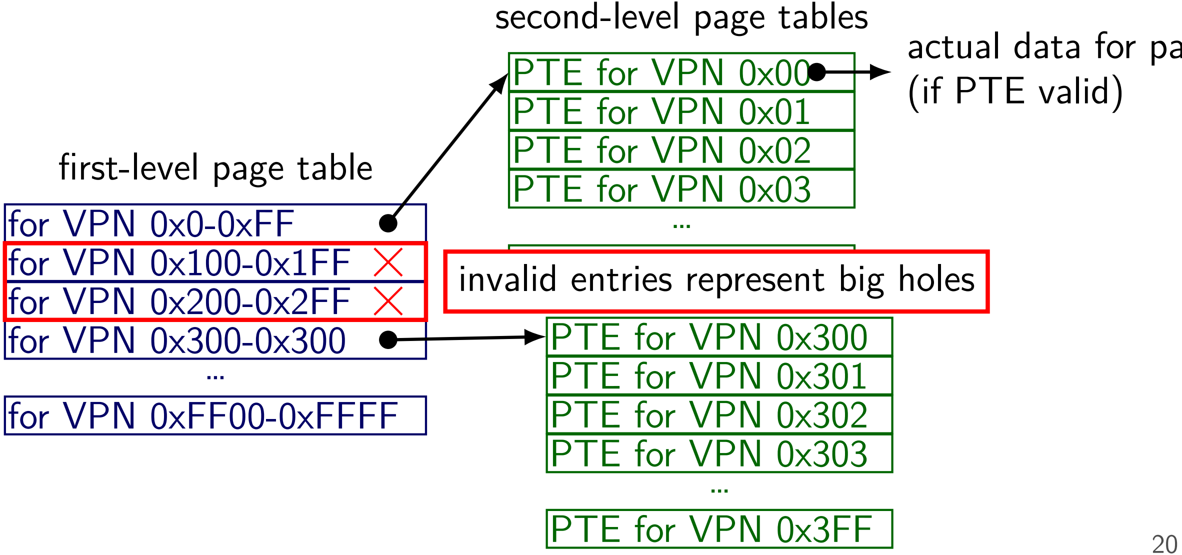
two-level page tables

two-level page table for 65536 pages (16-bit VPN)



two-level page tables

two-level page table for 65536 pages (16-bit VPN)



two-level page tables

two-level page table for 65536 pages (16-bit VPN)

first-level pag

for VPN 0x0-0xF
for VPN 0x100-0
for VPN 0x200-0
for VPN 0x300-0
...
for VPN 0xFF00

first-level page table

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

PTE for VPN 0x303
...
PTE for VPN 0x3FF

two-level page tables

two-level page table for 65536 pages (16-bit VPN)

first-level pag

for VPN 0x0-0xF

for VPN 0x100-0

for VPN 0x200-0

for VPN 0x300-0

...

for VPN 0xFF00

first-level page table

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

PTE for VPN 0x303

...

PTE for VPN 0x3FF

two-level page tables

two-level page table for 65536 pages (16-bit VPN)

first-level page table

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

first-level pag

for VPN 0x0-0xF

for VPN 0x100-0

for VPN 0x200-0

for VPN 0x300-0

...

for VPN 0xFF00

PTE for VPN 0x303

...

PTE for VPN 0x3FF

two-level page tables

two-level page table for 65536 pages (16-bit VPN)

first-level page table

for VPN 0x0-0xFF ●
for VPN 0x100-0x1FF ✕
for VPN 0x200-0x2FF ✕
for VPN 0x300-0x300 ●
...
for VPN 0xFF00-0xFFFF

a second-level page table

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

PTE for VPN 0x303

...

PTE for VPN 0x3FF

two-level page tables

two-level page table for 65536 pages (16-bit VPN)

first-level page table

- for VPN 0x0-0xFF ●
- for VPN 0x100-0x1FF ✕
- for VPN 0x200-0x2FF ✕
- for VPN 0x300-0x300 ●
- …
- for VPN 0xFF00-0xFFFF

a second-level page table

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

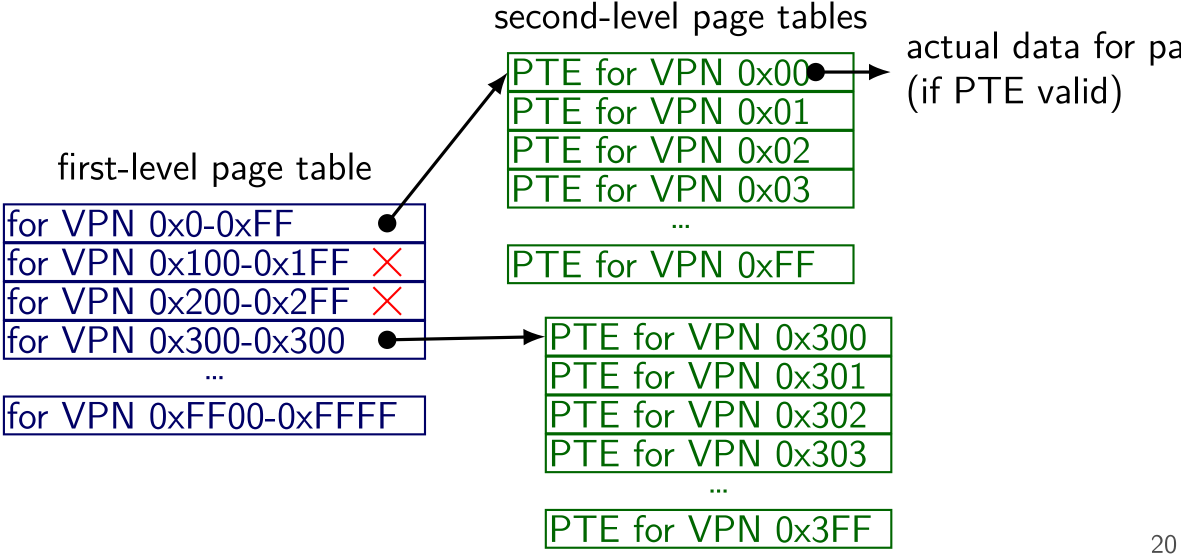
PTE for VPN 0x303

…

PTE for VPN 0x3FF

two-level page tables

two-level page table for 65536 pages (16-bit VPN)



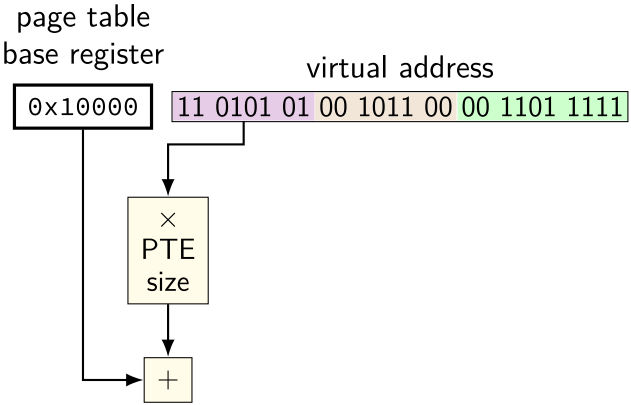
two-level page table lookup

virtual address

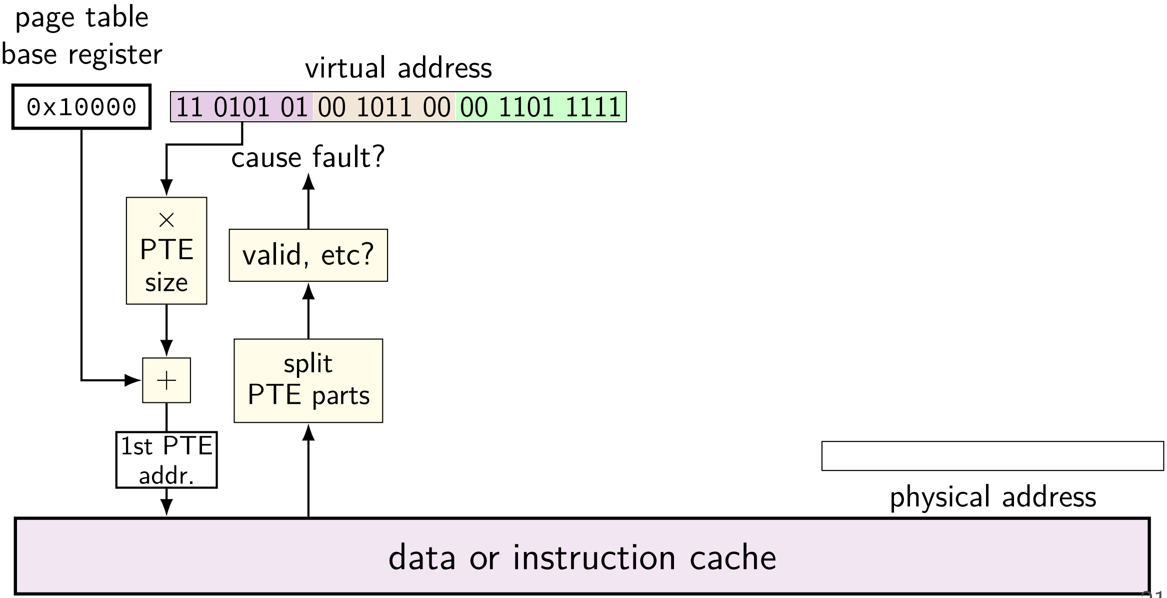
11 0101 01 | 00 1011 00 | 00 1101 1111

VPN — split into two parts (one per level)

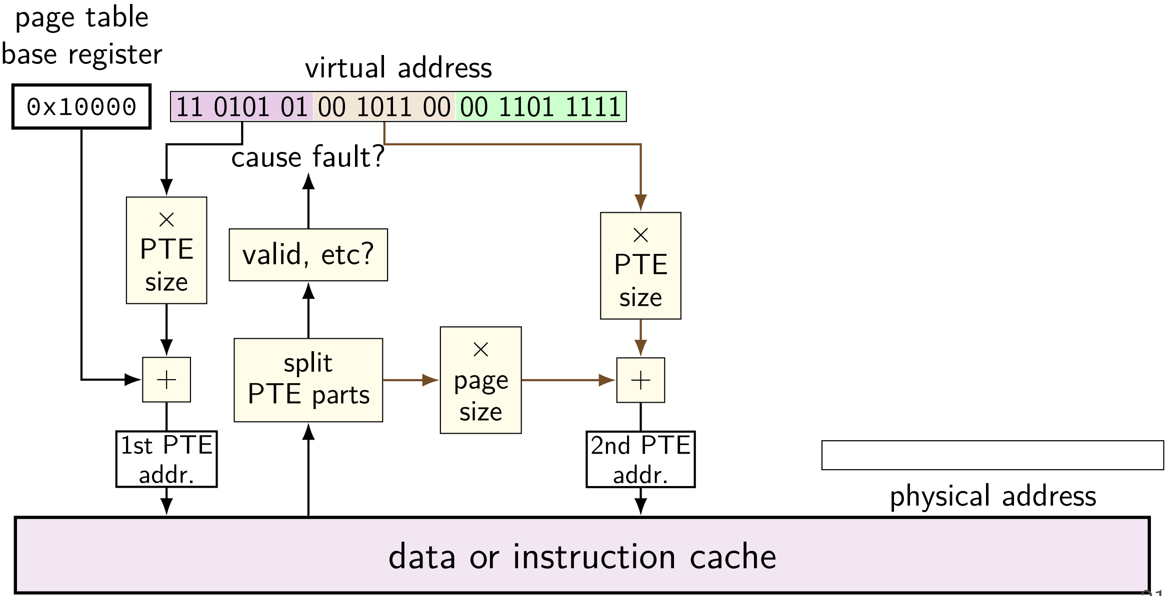
two-level page table lookup



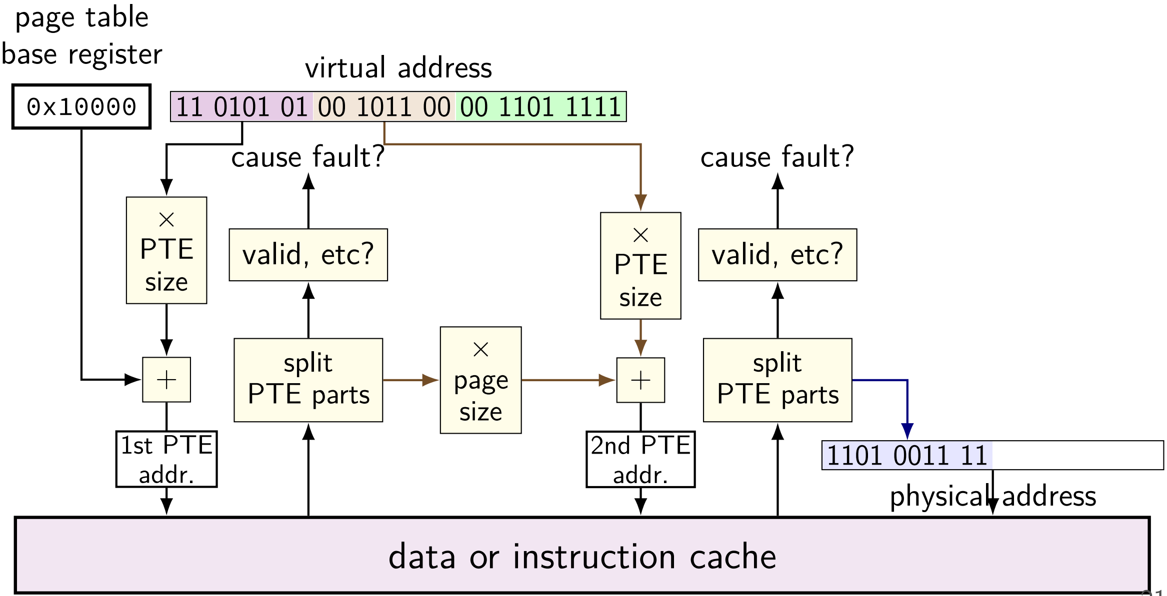
two-level page table lookup



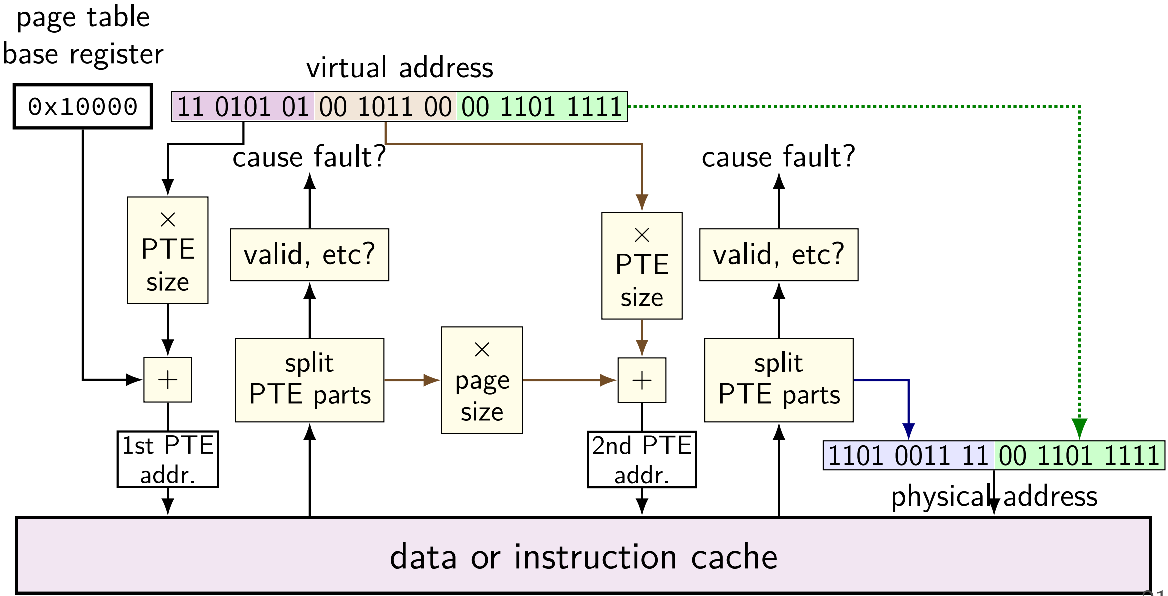
two-level page table lookup



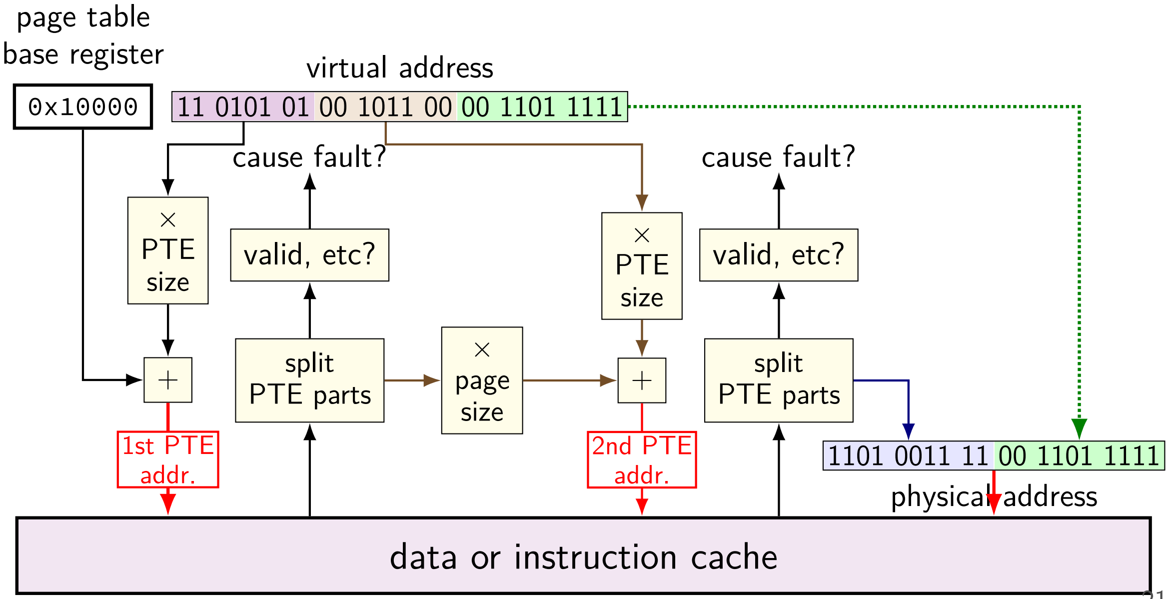
two-level page table lookup



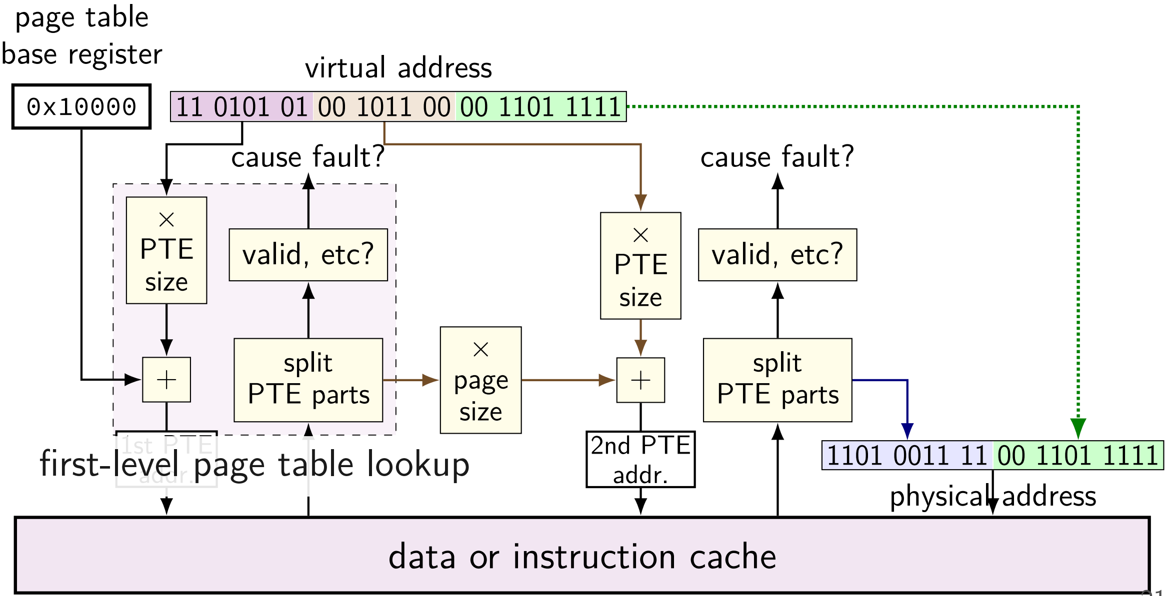
two-level page table lookup



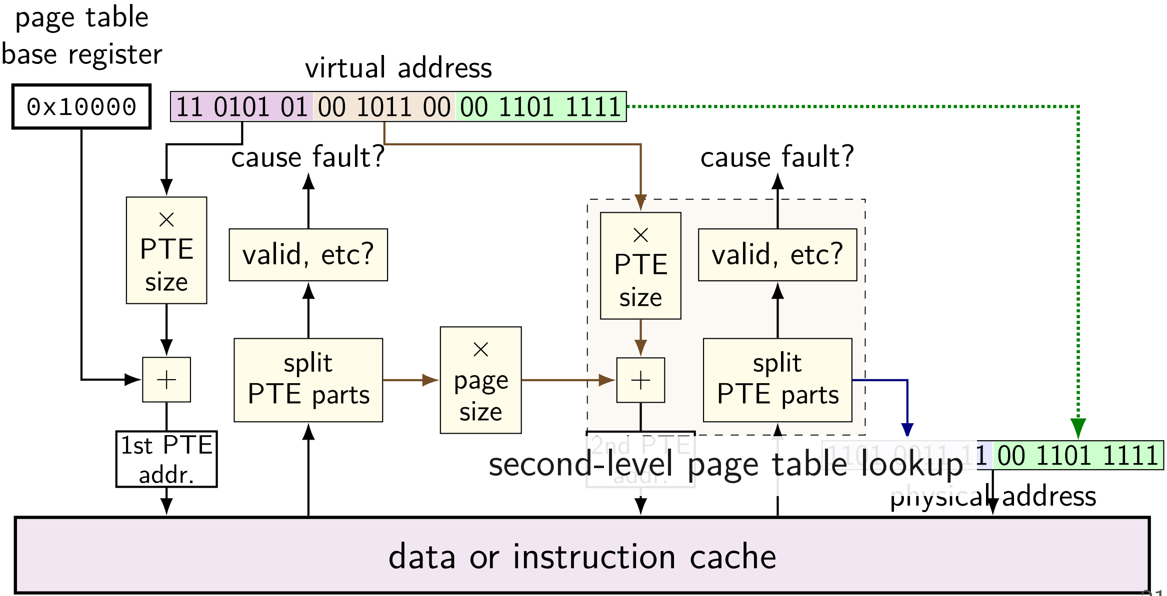
two-level page table lookup



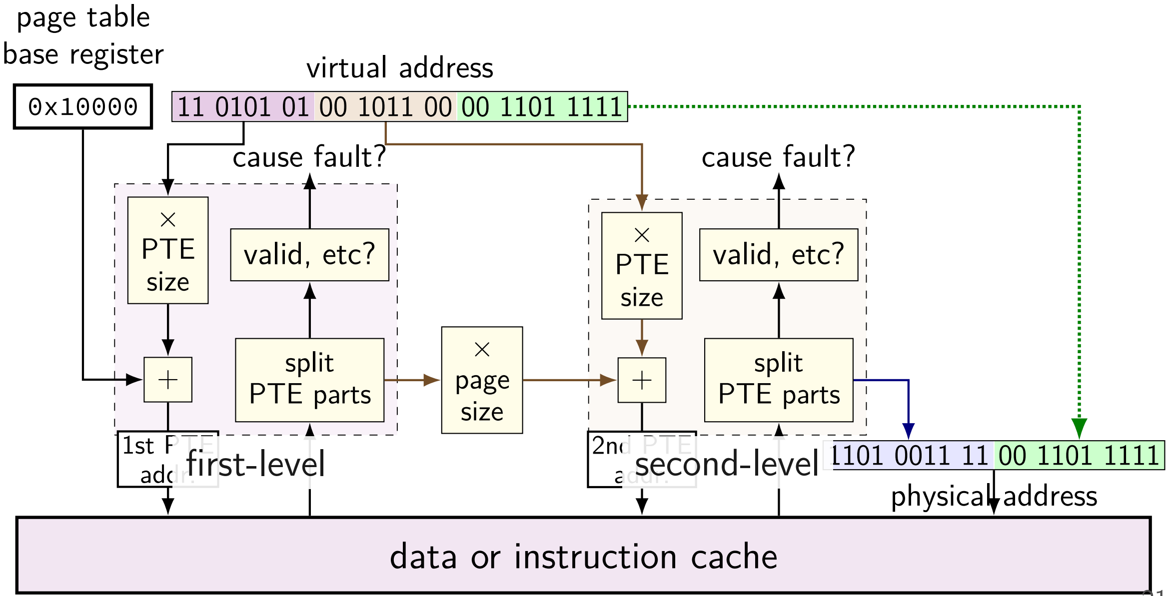
two-level page table lookup



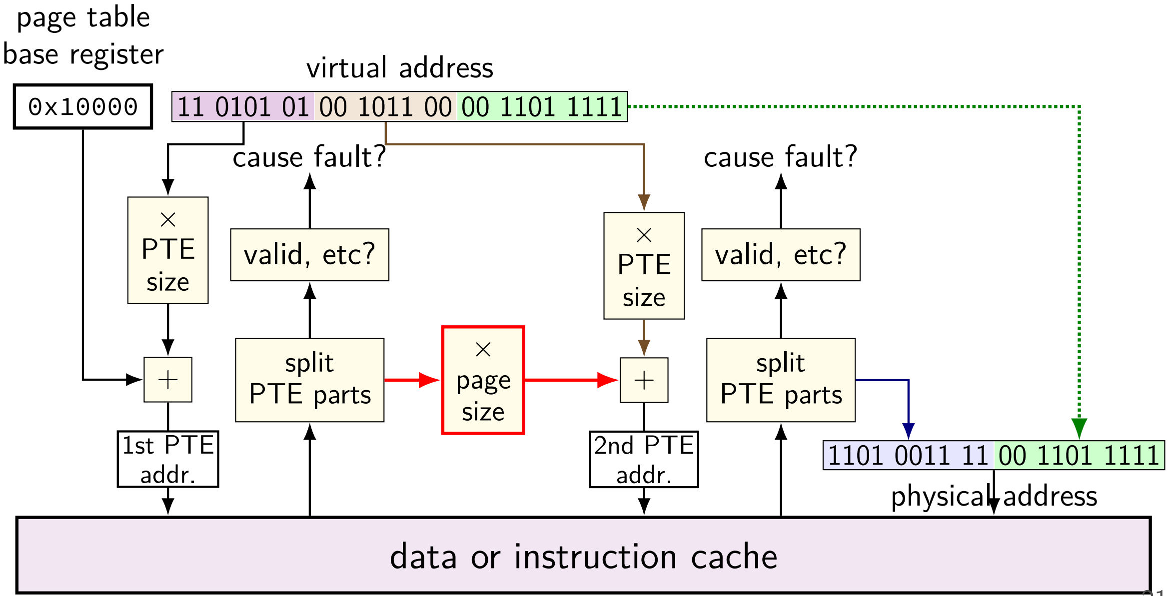
two-level page table lookup



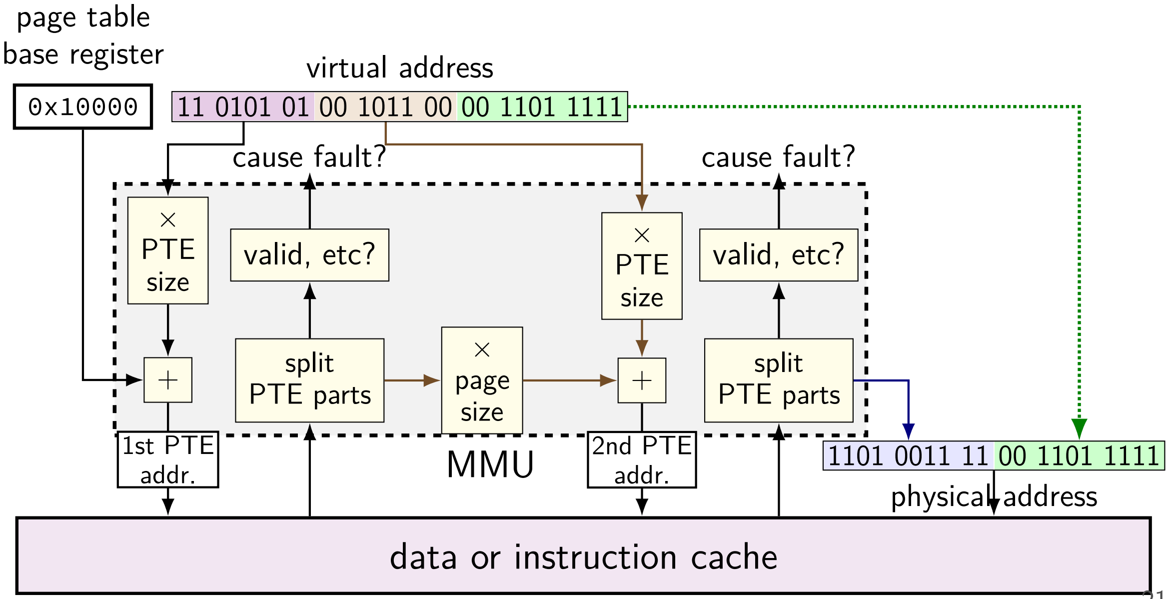
two-level page table lookup



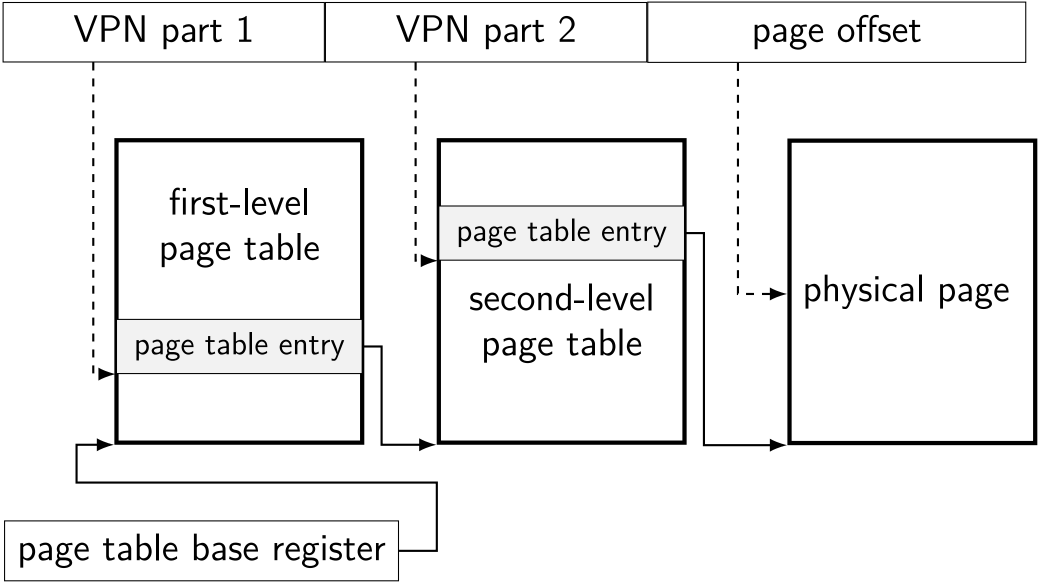
two-level page table lookup



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 and permission checks at each level

note on VPN splitting

textbook labels it ‘VPN 1’ and ‘VPN 2’ and so on

these are parts of the virtual page number

(there are not multiple VPNs)

splitting addresses for levels

x86-32

32-bit physical address; 32-bit virtual address

2^{12} byte page size

2-levels of page tables; each page table is one page

4 byte page table entries

how is address `0x12345678` split up?

splitting addresses for levels

x86-32

32-bit physical address; 32-bit virtual address

2^{12} byte page size

12-bit page offset

2-levels of page tables; each page table is one page

4 byte page table entries

how is address `0x12345678` split up?

splitting addresses for levels

x86-32

32-bit physical address; 32-bit virtual address

2^{12} byte page size

- *12-bit page offset*

2-levels of page tables; each page table is one page

4 byte page table entries

- $2^{12}/4 = 2^{10}$ *PTEs/page table; 10-bit VPN parts*

how is address `0x12345678` split up?

splitting addresses for levels

x86-32

32-bit physical address; 32-bit virtual address

2^{12} byte page size

12-bit page offset

2-levels of page tables; each page table is one page

4 byte page table entries

$2^{12}/4 = 2^{10}$ *PTEs/page table; 10-bit VPN parts*

how is address `0x12345678` split up?

10-bit VPN part 1: `0001 0010 00 (0x48);`

10-bit VPN part 2: `11 0100 0101 (0x345);`

12-bit page offset: `0x678`

1-level example

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 bits;

page table base register `0x20`; translate virtual address `0x30`

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	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

1-level example

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 bits;

page table base register `0x20`; translate virtual address `0x30`

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	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

`0x30` = `11 0000`

PTE addr:

`0x20` + 6 × 1 = `0x26`

PTE value:

`0xD6` = `1101 0110`

PPN `110`, valid `1`

M[`110 000`] = **M[**`0x30`]

1-level example

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 bits;

page table base register `0x20`; translate virtual address `0x30`

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	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

`0x30` = `11 0000`

PTE addr:

`0x20` + 6 × 1 = `0x26`

PTE value:

`0xD6` = `1101 0110`

PPN `110`, valid 1

M[`110 000`] = **M[**`0x30`]

1-level example

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 bits;

page table base register `0x20`; translate virtual address `0x30`

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	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

`0x30` = `11 0000`

PTE addr:

`0x20` + 6 ×1 = `0x26`

PTE value:

`0xD6` = `1101 0110`

PPN `110`, valid `1`

M[`110 000`] = **M[**`0x30`]

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 `0x131`

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 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 `0x131`

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

`0x131` = `1 0011 0001`

`0x20` + 4 × 1 = `0x24`

PTE 1 value:

`0xD4` = `1101 0100`

PPN `110`, valid `1`

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 `0x131`

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

0x131 = 1 0011 0001

0x20 + 4 × 1 = 0x24

PTE 1 value:

0xD4 = 1101 0100

PPN 110, valid 1

PTE 2 addr:

110 000 + 110 = 0x36

PTE 2 value: 0xDB

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 `0x131`

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

`0x131` = `1 0011 0001`

`0x20` + 4 ×`1` = `0x24`

PTE 1 value:

`0xD4` = `1101 0100`

PPN `110`, valid `1`

PTE 2 addr:

`110 000` + `110` = `0x36`

PTE 2 value: `0xDB`

PPN `110`; valid `1`

M[`110 001` (`0x31`)] = `0x0A`

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 `0x131`

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

`0x131` = `1 0011 0001`

`0x20` + 4 ×1 = `0x24`

PTE 1 value:

`0xD4` = `1101 0100`

PPN `110`, valid 1

PTE 2 addr:

`110 000` + `110` = `0x36`

PTE 2 value: `0xDB`

PPN `110`; valid 1

M[`110 001` (`0x31`)] = `0x0A`

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 `0x131`

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

0x131 = 1 0011 0001

0x20 + 4 ×1 = 0x24

PTE 1 value:

0xD4 = 1101 0100

PPN 110, valid 1

PTE 2 addr:

110 000 + 110 = 0x36

PTE 2 value: 0xDB

PPN 110; valid 1

M[110 001 (0x31)] = 0x0A

2-level splitting

9-bit virtual address

6-bit physical address

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

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

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

8 entry page tables $\rightarrow$ 3-bit VPN parts

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

pages and page table base pointer

page table base pointer — only for first-level lookup

- zeroth page table entry

1st-level page table entry contains physical page number

multiply page number by page size to get byte address of page

- (then same process as using page table base pointer)

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

`0x0FB` = `011 111 011`

PTE 1 addr:

`0x08` + 3 ×1 = `0x0B`

PTE 1: `0xBB` at `0x0B`

PTE 1: PPN `101` (5) valid 1

PTE 2 addr:

`101 000` + `111` = `0x2F`

PTE 2: `0xF0` at `0x2F`

PTE 2: PPN `111` (7) valid 1

`111 011` = `0x3B` → `0x0C`

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

`0x0FB` = `011 111 011`

PTE 1 addr:

`0x08` + 3 × 1 = `0x0B`

PTE 1: `0xBB` at `0x0B`

PTE 1: PPN `101` (5) valid 1

PTE 2 addr:

`101 000` + `111` = `0x2F`

PTE 2: `0xF0` at `0x2F`

PTE 2: PPN `111` (7) valid 1

`111 011` = `0x3B` → `0x0C`

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

0x0FB = 011 111 011

PTE 1 addr:

0x08 + 3 × 1 = 0x0B

PTE 1: 0xBB at 0x0B

PTE 1: PPN 101 (5) valid 1

PTE 2 addr:

101 000 + 111 = 0x2F

PTE 2: 0xF0 at 0x2F

PTE 2: PPN 111 (7) valid 1

111 011 = 0x3B → 0x0C

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

0x0FB = 011 111 011

PTE 1 addr:

0x08 + 3 ×1 = 0x0B

PTE 1: 0xBB at 0x0B

PTE 1: PPN 101 (5) valid 1

PTE 2 addr:

101 000 + 111 = 0x2F

PTE 2: 0xF0 at 0x2F

PTE 2: PPN 111 (7) valid 1

111 011 = 0x3B → 0x0C

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 `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 (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 `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

`0x00B` = `000 001 011`

PTE 1: `0x88` at `0x08`

PTE 1: PPN `100` (5) valid `0`

page fault!

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 `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

`0x00B` = `000 001 011`

PTE 1: `0x88` at `0x08`

PTE 1: PPN `100` (5) valid `0`

page fault!

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 `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 (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 `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

`0x1CB` = `111 001 011`

PTE 1: `0xFF` at `0x0F`

PTE 1: PPN `111` (7) valid `1`

PTE 2: `0x0C` at `0x39`

PTE 2: PPN `000` (0) valid `0`

page fault!

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 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

0x1CB = 111 001 011

PTE 1: 0xFF at 0x0F

PTE 1: PPN 111 (7) valid 1

PTE 2: 0x0C at 0x39

PTE 2: PPN 000 (0) valid 0

page fault!

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 `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

`0x1CB` = `111 001 011`

PTE 1: `0xFF` at `0x0F`

PTE 1: PPN `111` (7) valid `1`

PTE 2: `0x0C` at `0x39`

PTE 2: PPN `000` (0) valid `0`

page fault!

2-level exercise (4)

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

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

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

physical addresses	bytes	physical addresses	bytes
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 AA BB	0x28-B	89 9A AB BC
0x0C-F	CC DD EE FF	0x2C-F	CD DE EF F0
0x10-3	1A 2A 3A 4A	0x30-3	BA 0A BA 0A
0x14-7	1B 2B 3B 4B	0x34-7	DB 0B DB 0B
0x18-B	1C 2C 3C 4C	0x38-B	EC 0C EC 0C
0x1C-F	AC BC DC EC	0x3C-F	FC 0C FC 0C

2-level exercise (4)

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

page tables 1 page; PTE: 2 bit PPN (MSB of first byte), 1 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

`0x376` $=$ `110 111 0110`

PTE 1: $0x10 + 6 \times 2 =$ `0x1C`: `AC BC`

PTE 1: PPN `10` valid `1`

PTE 2: $0x20 + 7 \times 2 =$ `0x2E`: `EF F0`

PTE 2: PPN `11` valid `1`

`11 0110` $=$ `0x36` $\rightarrow$ `DB`

2-level exercise (4)

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

page tables 1 page; PTE: 2 bit PPN (MSB of first byte), 1 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

`0x376` = 110 111 0110

PTE 1: $0x10 + 6 \times 2 =$ `0x1C`: AC BC

PTE 1: PPN 10 valid 1

PTE 2: $0x20 + 7 \times 2 =$ `0x2E`: EF F0

PTE 2: PPN 11 valid 1

11 0110 = 0x36 → DB

2-level exercise (4)

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

page tables 1 page; PTE: 2 bit PPN (MSB of first byte), 1 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

`0x376` $=$ `110 111 0110`

PTE 1: $0x10 + 6 \times 2 =$ `0x1C`: `AC BC`

PTE 1: PPN `10` valid `1`

PTE 2: $0x20 + 7 \times 2 =$ `0x2E`: `EF F0`

PTE 2: PPN `11` valid `1`

`11 0110` $=$ `0x36` $\rightarrow$ `DB`

2-level exercise (4)

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

page tables 1 page; PTE: 2 bit PPN (MSB of first byte), 1 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

`0x376` $=$ `110 111 0110`

PTE 1: $0x10 + 6 \times 2 =$ `0x1C`: `AC BC`

PTE 1: PPN `10` valid `1`

PTE 2: $0x20 + 7 \times 2 =$ `0x2E`: `EF F0`

PTE 2: PPN `11` valid `1`

`11 0110` $=$ `0x36` $\rightarrow$ DB

2-level exercise (4)

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

page tables 1 page; PTE: 2 bit PPN (MSB of first byte), 1 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

`0x376` $=$ `110 111 0110`

PTE 1: $0\text{x}10 + 6 \times 2 =$ `0x1C`: `AC BC`

PTE 1: PPN `10` valid `1`

PTE 2: $0\text{x}20 + 7 \times 2 =$ `0x2E`: `EF F0`

PTE 2: PPN `11` valid `1`

`11 0110` $=$ `0x36` $\rightarrow$ `DB`

2-level exercise (4)

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

page tables 1 page; PTE: 2 bit PPN (MSB of first byte), 1 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

0x376 = 110 111 0110

PTE 1: $0x10 + 6 \times 2 =$ 0x1C: AC BC

PTE 1: PPN 10 valid 1

PTE 2: $0x20 + 7 \times 2 =$ 0x2E: EF F0

PTE 2: PPN 11 valid 1

11 0110 = 0x36 → DB

2-level exercise (4)

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

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

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

physical addresses	bytes	physical addresses	bytes
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 AA BB	0x28-B	89 9A AB BC
0x0C-F	CC DD EE FF	0x2C-F	CD DE EF F0
0x10-3	1A 2A 3A 4A	0x30-3	BA 0A BA 0A
0x14-7	1B 2B 3B 4B	0x34-7	DB 0B DB 0B
0x18-B	1C 2C 3C 4C	0x38-B	EC 0C EC 0C
0x1C-F	AC BC DC EC	0x3C-F	FC 0C FC 0C

`0x376` $=$ `110 111 0110`

PTE 1: $0x10 + 6 \times 2 =$ `0x1C`: `AC BC`

PTE 1: PPN `10` valid `1`

PTE 2: $0x20 + 7 \times 2 =$ `0x2E`: `EF F0`

PTE 2: PPN `11` valid `1`

`11 0110` $=$ `0x36` $\rightarrow$ `DB`

current x86-64 page tables

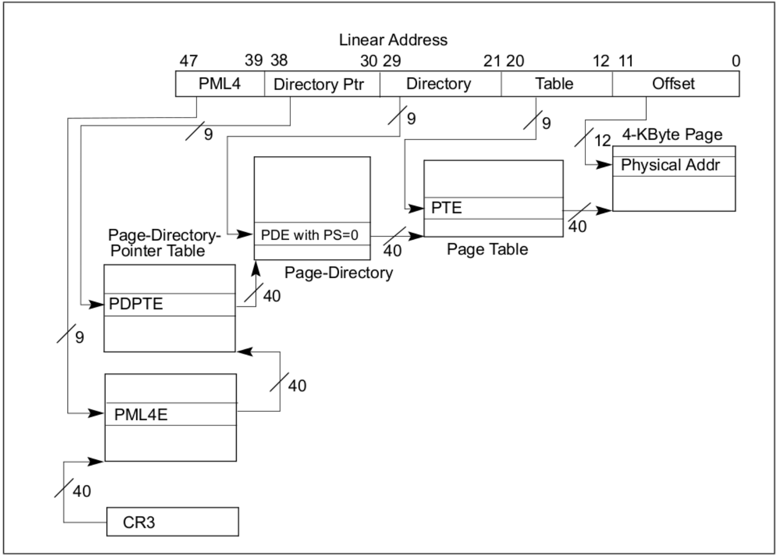


Figure 4-8. Linear-Address Translation to a 4-KByte Page using 4-Level Paging

current x86-64 page tables

4-level page table

512 PTEs of 8 bytes each for each page table

choice: exactly one page per page table

allows OS to allocate new page table space in one page units

(just like program memory)

page table space exercise (1)

4-level page table

512 PTEs of 8 bytes each for each page table

suppose a process has exactly one page allocated

how much space for page tables?

page table space exercise (1)

4-level page table

512 PTEs of 8 bytes each for each page table

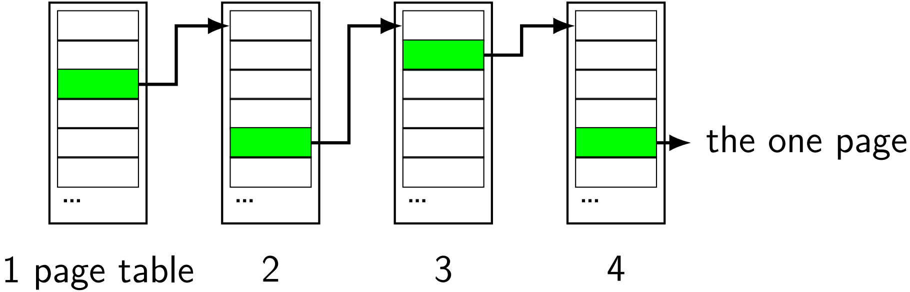
suppose a process has exactly one page allocated

how much space for page tables?

1 page at each level (4KB each)

exactly one valid entry in each of them

page table space exercise (1)



4 page tables at 1 page/page table
plus 1 page of data
5 pages total

page table space exercise (2)

4-level page table

512 PTEs of 8 bytes each for each page table

suppose a process has exactly two pages allocated:

- one at address 0x0, one at address `0x200000000000`

how much space for page tables?

page table space exercise (2)

4-level page table

512 PTEs of 8 bytes each for each page table

suppose a process has exactly two pages allocated:

- one at address 0x0, one at address `0x20000000000`

how much space for page tables?

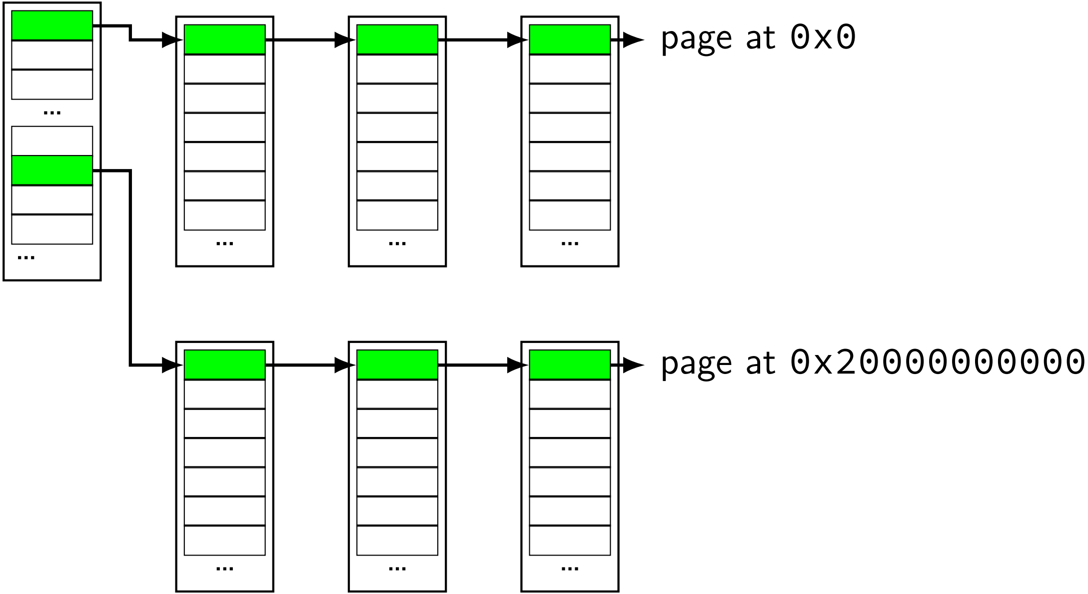
1 shared first-level PT, with two valid entries

two second-level PTs, each with one valid entry

two third-level PTs, each with one valid entry

two fourth-level PTs, each with one valid entry

page table space exercise (2)



page table space exercise (3)

4-level page table; each PT: 512 PTEs of 8 bytes

suppose a process has 100 pages of stack, 100 pages of code+constants (contiguous)

- stack and code+constants far apart

how much space for page tables?

page table space exercise (3)

4-level page table; each PT: 512 PTEs of 8 bytes

suppose a process has 100 pages of stack, 100 pages of code+constants (contiguous)

- stack and code+constants far apart

how much space for page tables? — *minimum:*

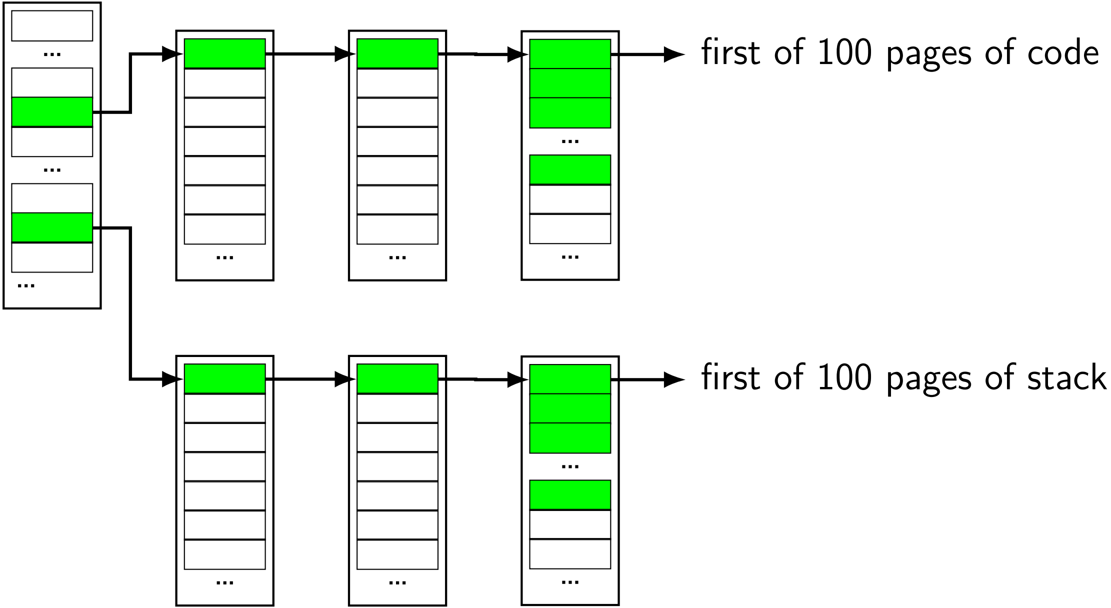
1 shared first-level PT, with two valid entries

two second-level PT, each with one valid entry

two third-level PT, each with one valid entry

two fourth-level PT, each with 100 valid entries

page table space exercise (3)



page table space exercise (3)

4-level page table; each PT: 512 PTEs of 8 bytes

suppose a process has 100 pages of stack, 100 pages of code+constants (contiguous)

how much space for page tables?

page table space exercise (3)

4-level page table; each PT: 512 PTEs of 8 bytes

suppose a process has 100 pages of stack, 100 pages of code+constants (contiguous)

how much space for page tables? — *maximum:*

1 shared first-level PT, with four valid entries

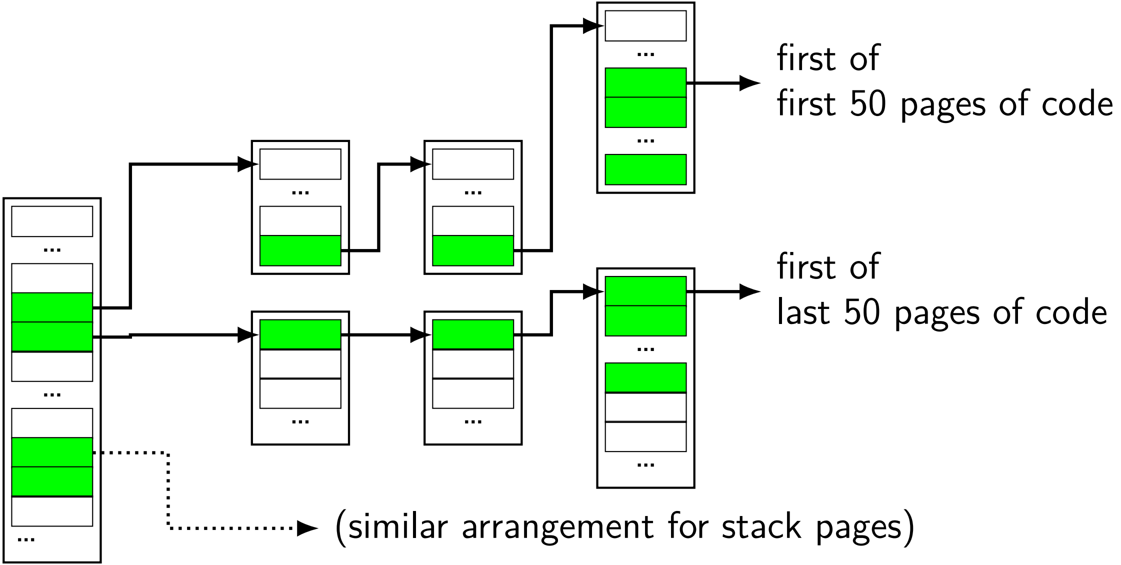
four second-level PT, each with one valid entry

- two for stack, two for code+constants

four third-level PT, each with one valid entry

four fourth-level PT, each with 50 valid entries

page table space exercise (3)



page table space exercise (4)

4-level page table; each PT: 512 PTEs of 8 bytes

suppose a process has 200 pages, randomly distributed in PT

about how much space for page tables?

page table space exercise (4)

4-level page table; each PT: 512 PTEs of 8 bytes

suppose a process has 200 pages, randomly distributed in PT

about how much space for page tables?

about 165 ($\pm \sim 8$) entries in first-level PT
(some pages randomly share first-level PT entries)

about 165 second-level PTs, 200 third-level, 200 fourth-level

a bit less than 600 page tables — almost 2400 KB