
vm

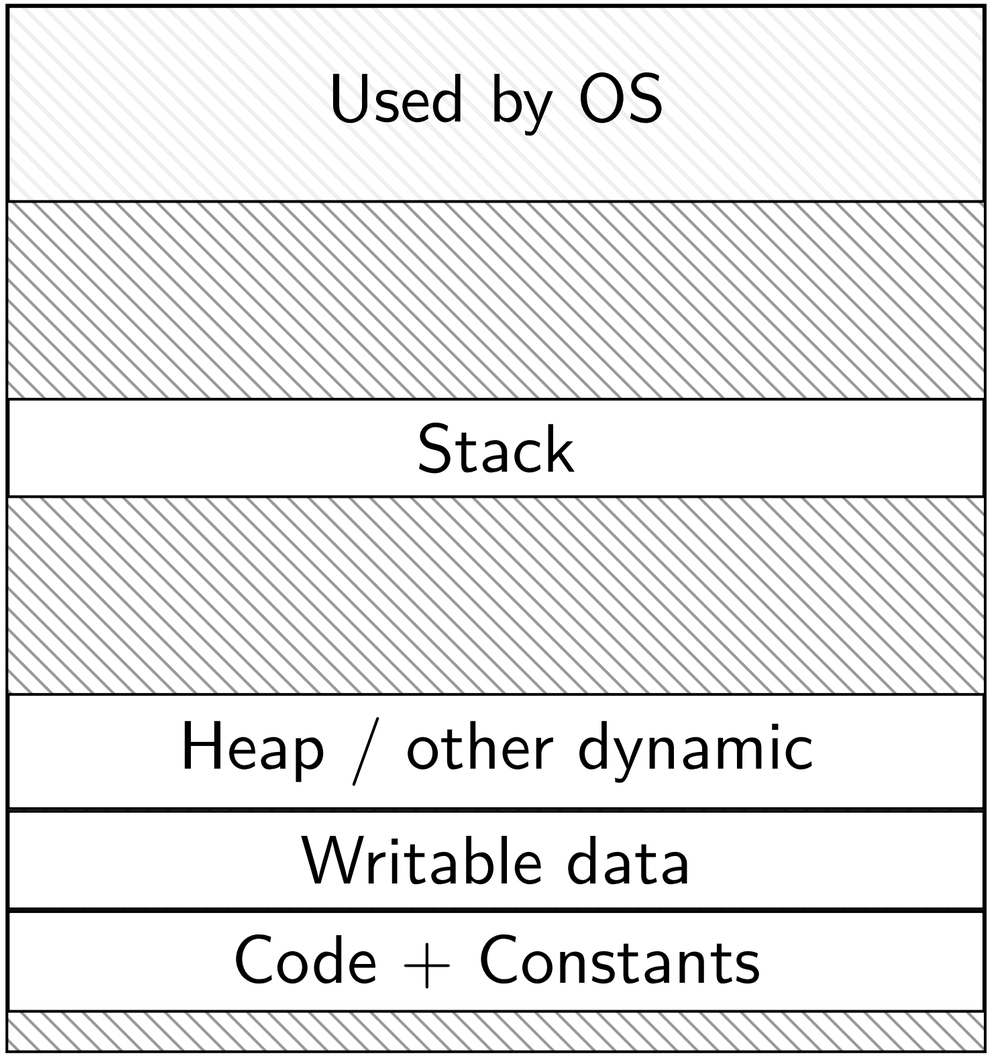
changelog

early Feb: adjust formatting on page table exercises; add some new exercises about general page table layout

12 Feb 2026: fixup errors in several of the 2-level exercises where physical memory contents did not agree with the given solution; correct parens on page able counting exercises

14 Feb 2026: correct erroneous conversion of 16384 to 2^{14} in exercises about how many levels of page table are needed; adjusted one instance to 65536 instead of 16384 and

program memory



0xFFFF FFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F…

0x0000 0000 0040 0000

this layout is what we want

programs know their addresses when compiled

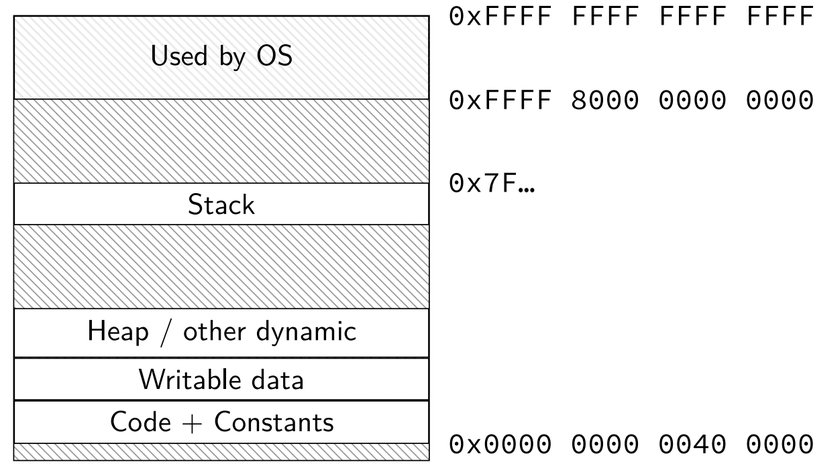
- compiler can generate smaller, more efficient code

programs have (essentially) unlimited memory

- or at least it appears that way

programs are not machine-specific (within same ISA)

- i.e., don't need to know how much memory is installed

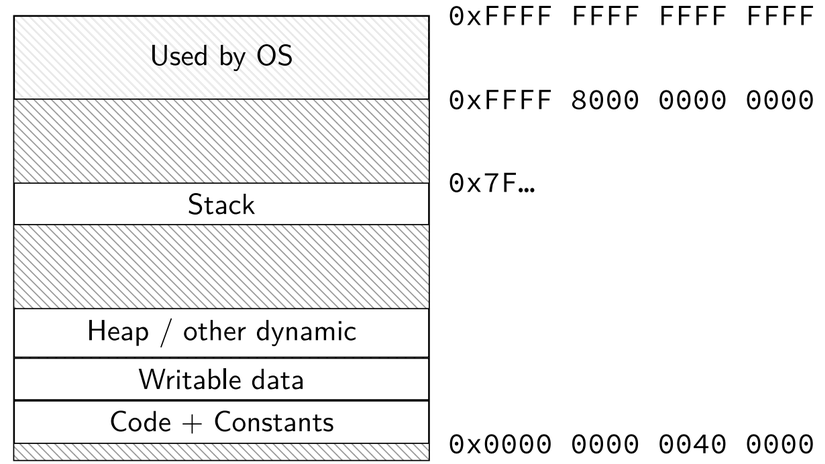


but, this has a number of issues!

every program uses the same addresses

could only run one program at a time

our machines don't actually have unlimited memory



virtualization allows us to address this

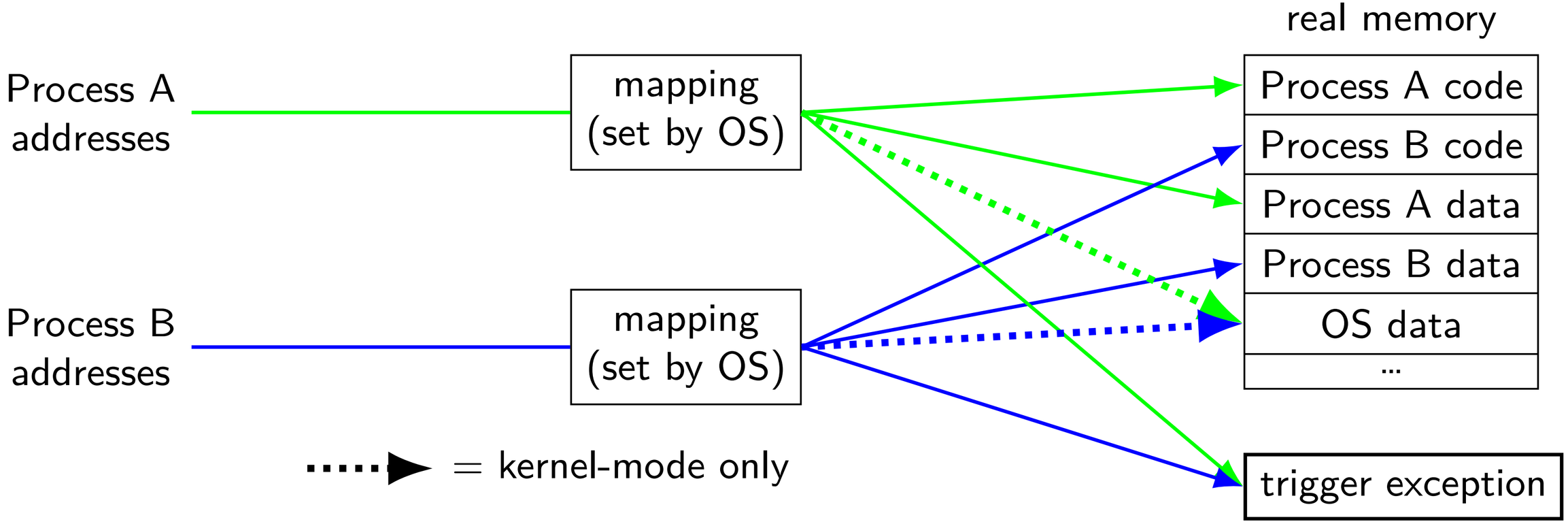
virtualization: allow many users to access the same computing hardware resources as if they were the only user

examples:

- virtual machine: multiple OSes running on same hardware
- virtual memory: multiple processes using same physical memory

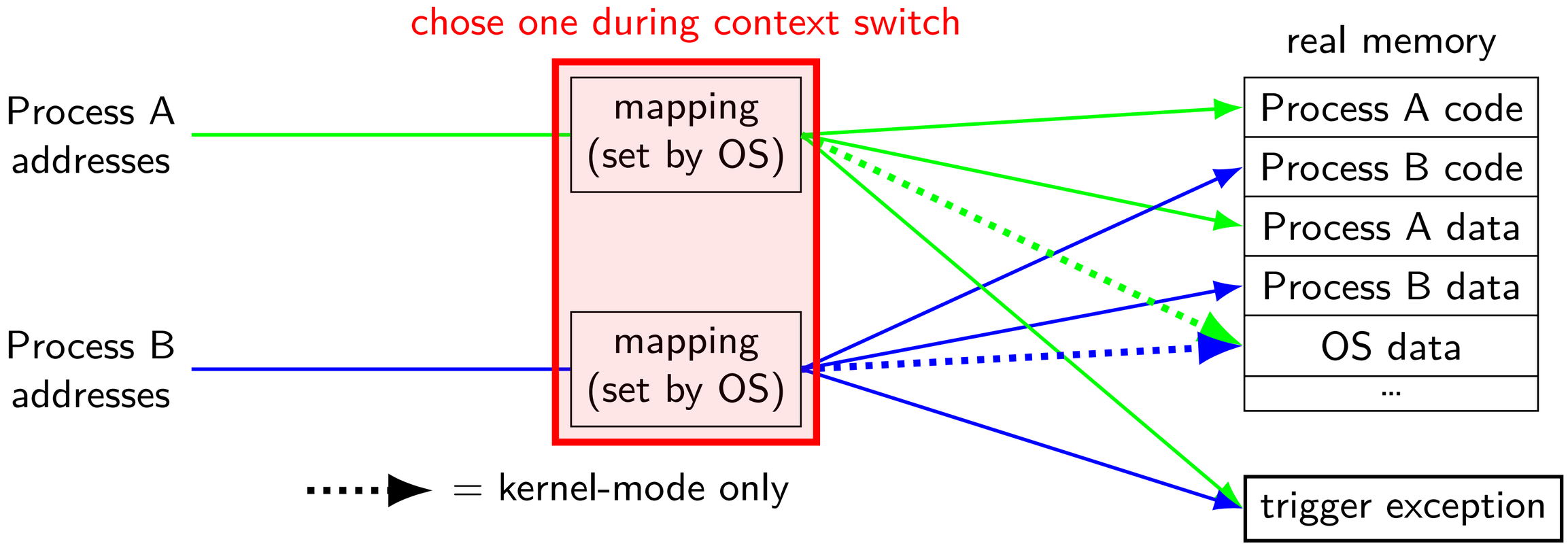
address spaces

illusion of *dedicated memory*



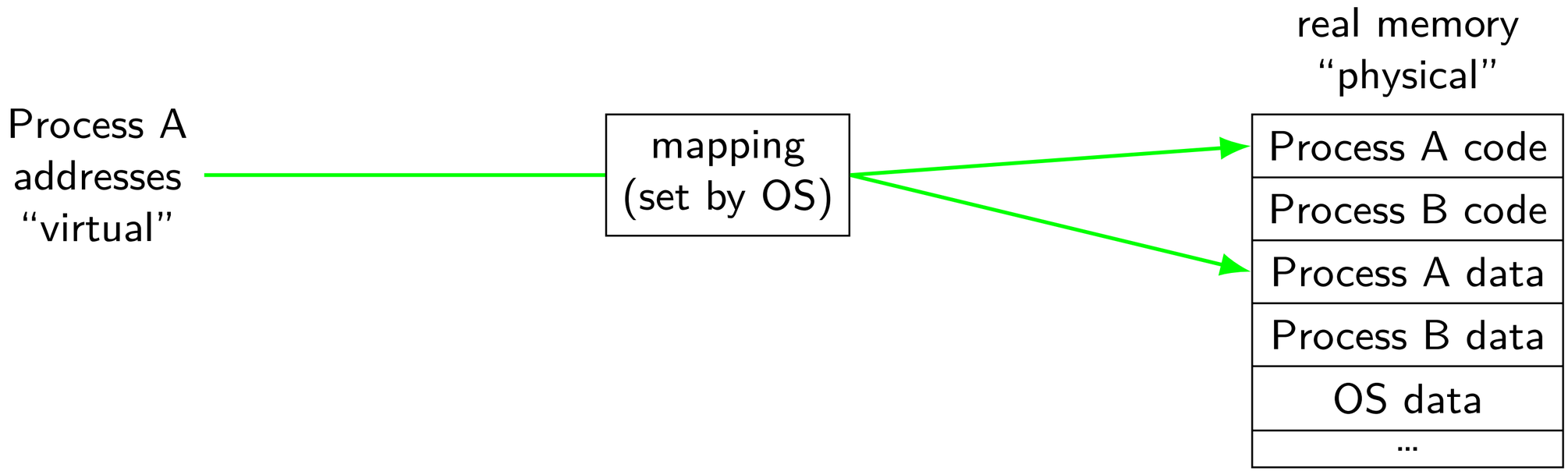
address spaces

illusion of *dedicated memory*

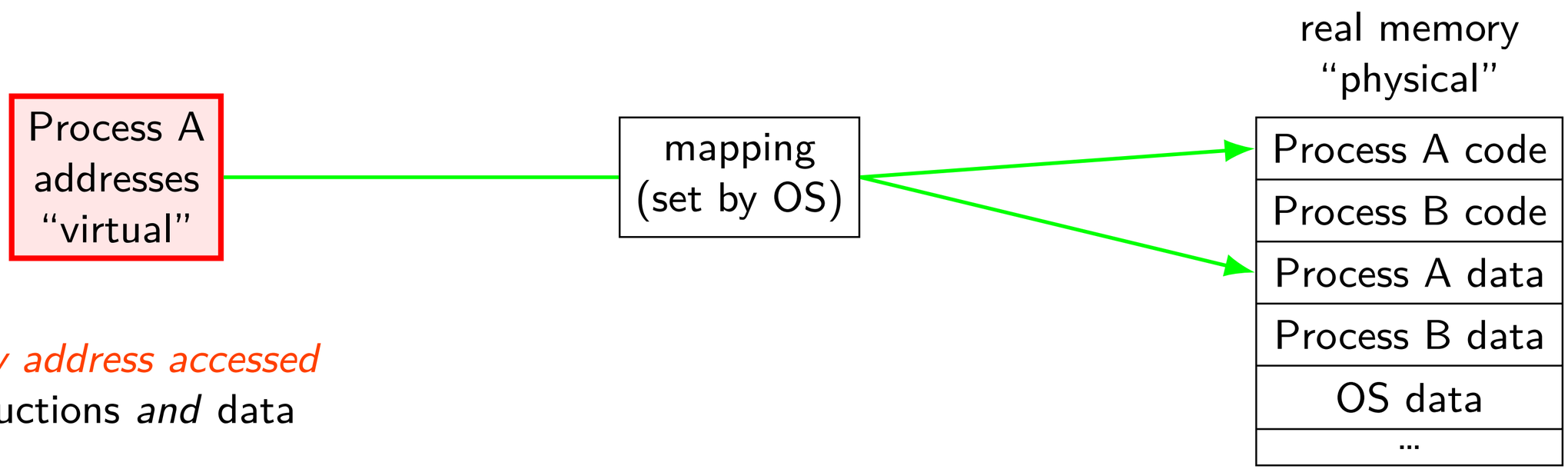


address translation

address translation

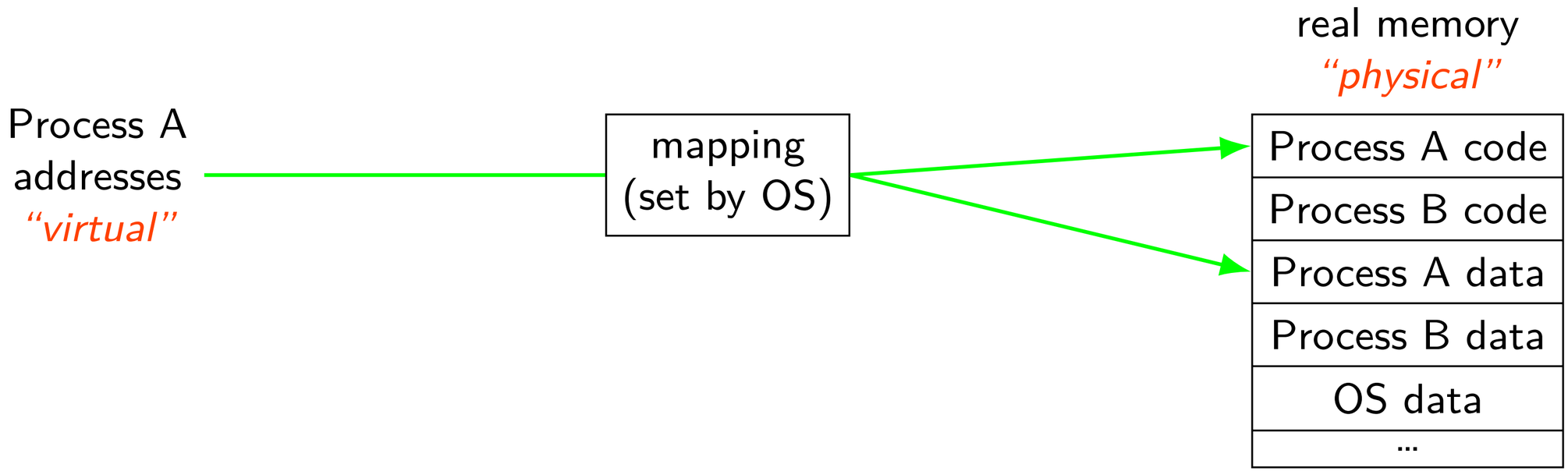


address translation



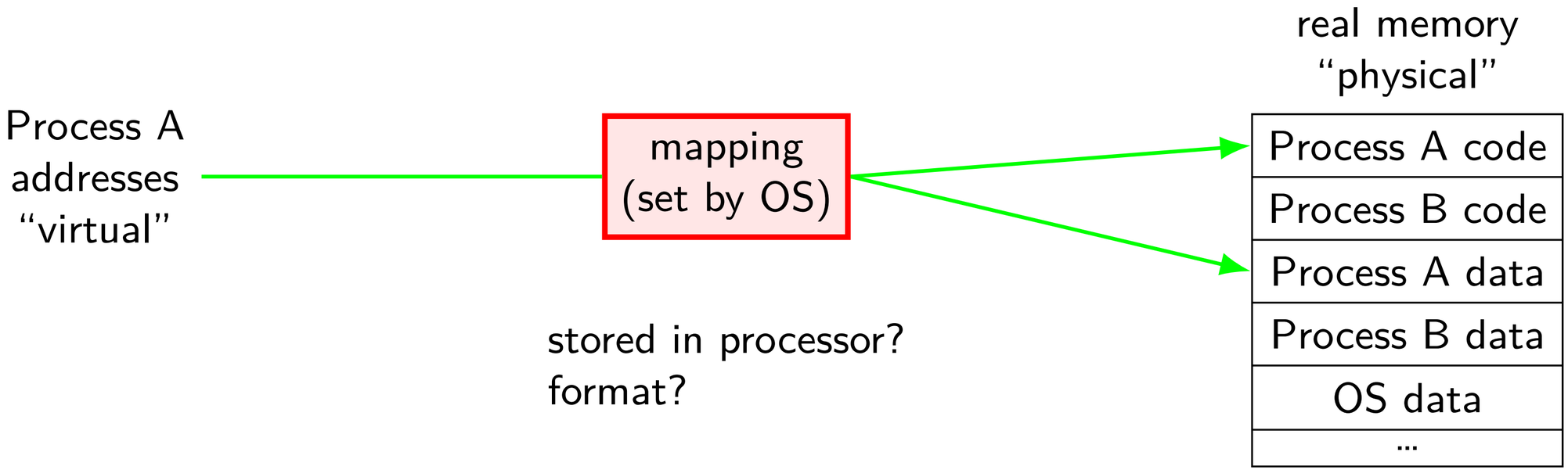
every address accessed
instructions *and* data

address translation

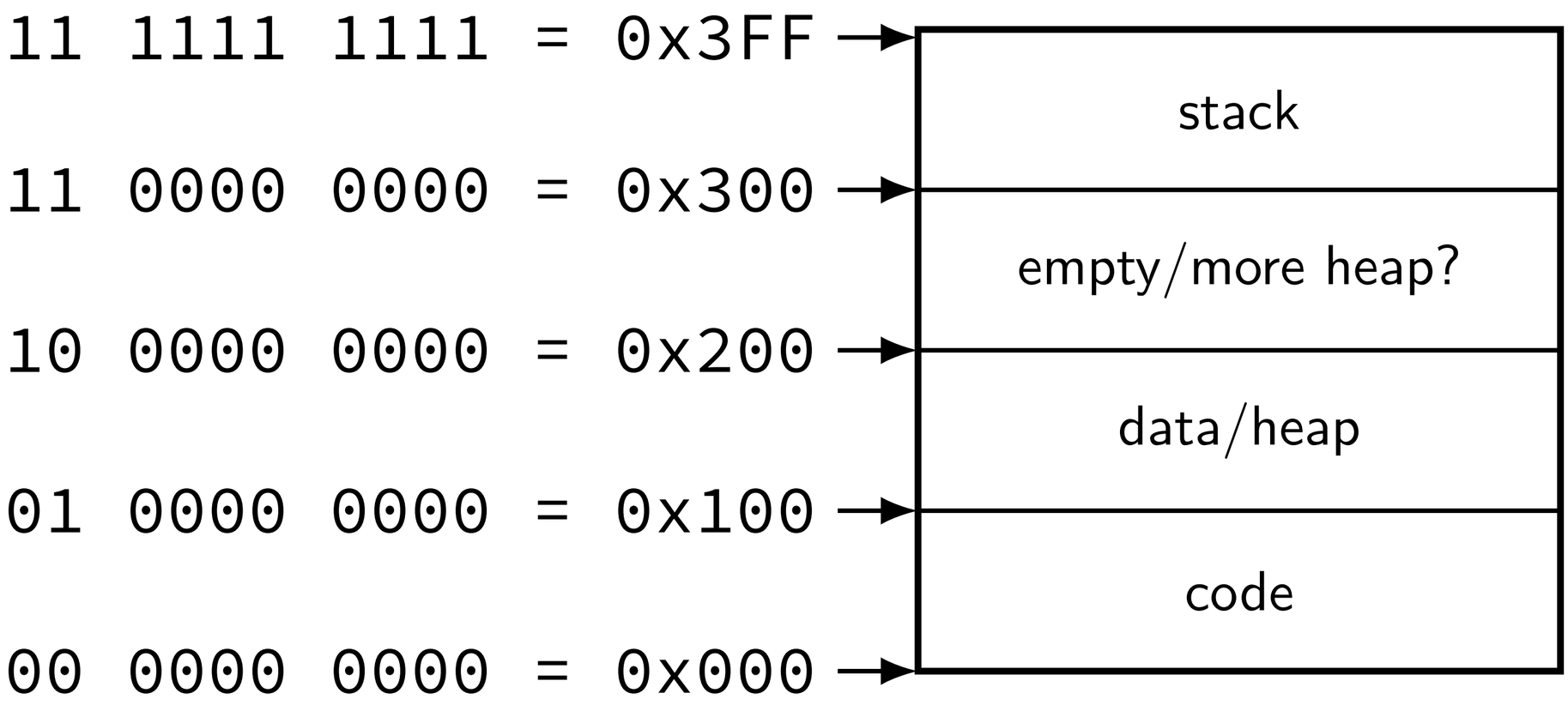


program addresses are 'virtual'
real addresses are 'physical'
can be *different sizes*!

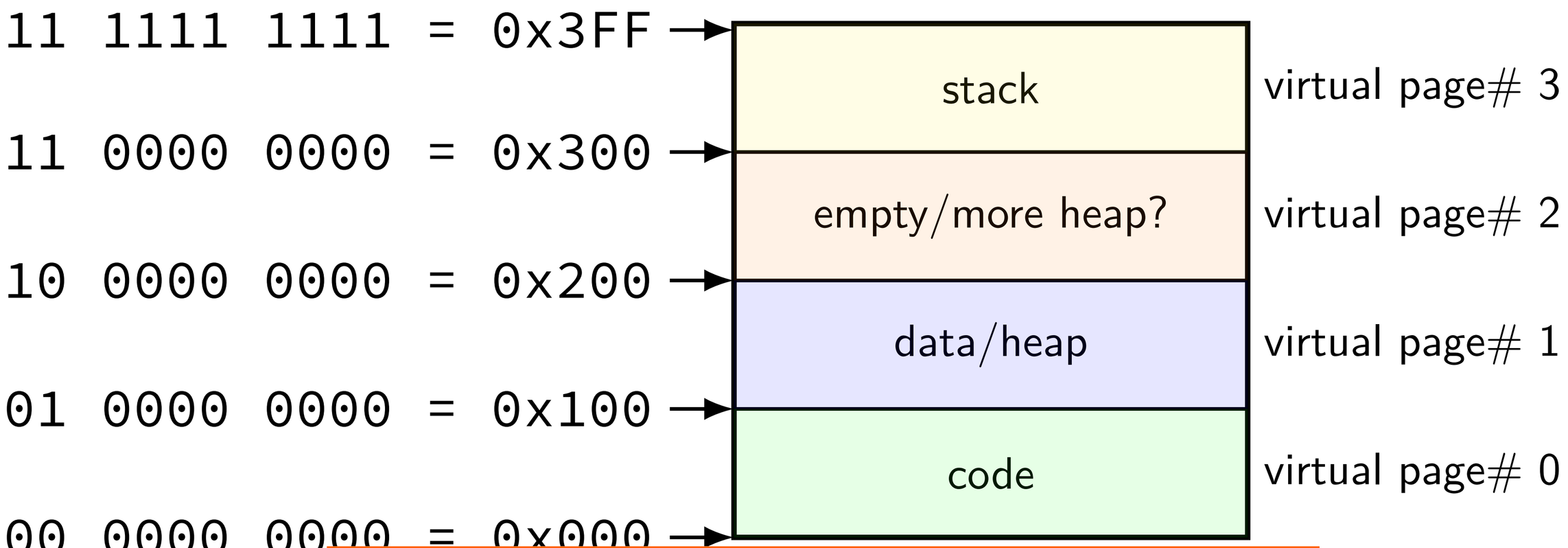
address translation



toy program memory



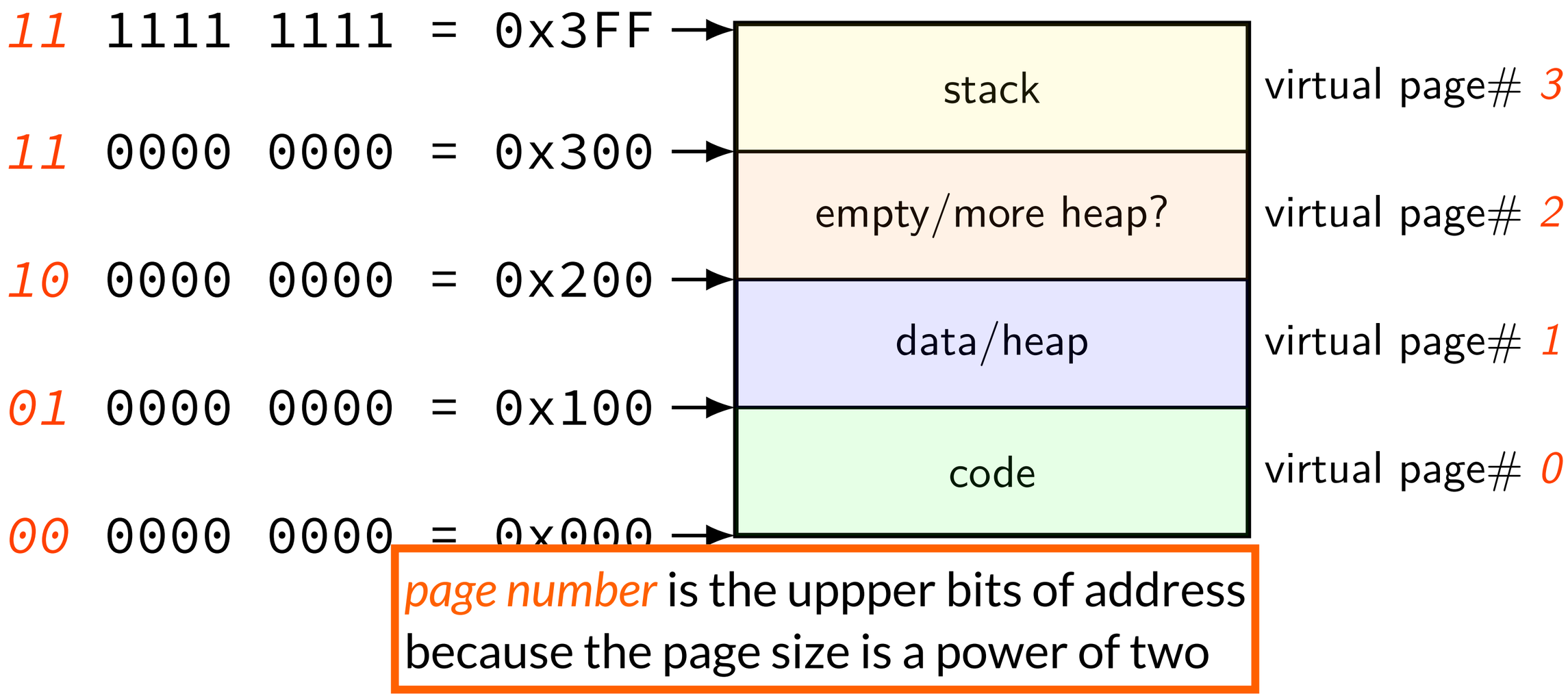
toy program memory



divide memory into *pages* (2^8 bytes in this case)

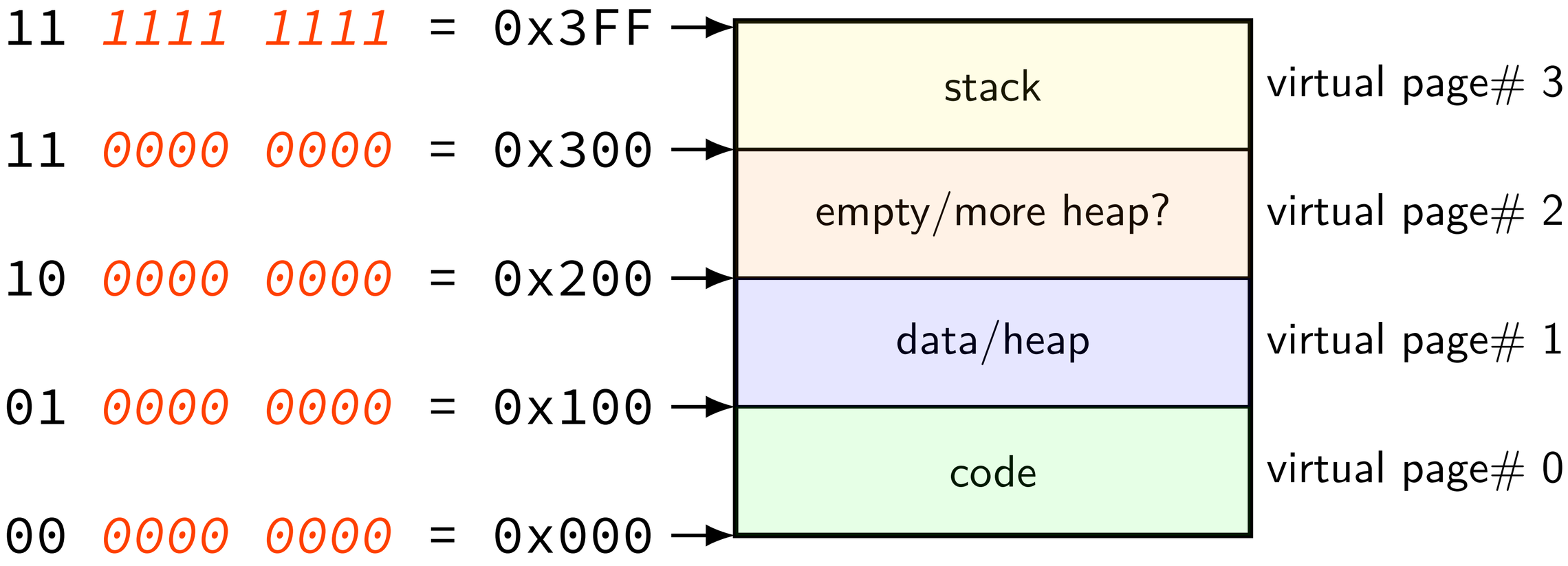
"virtual" = addresses the program sees

toy program memory



page number is the uppper bits of address because the page size is a power of two

toy program memory

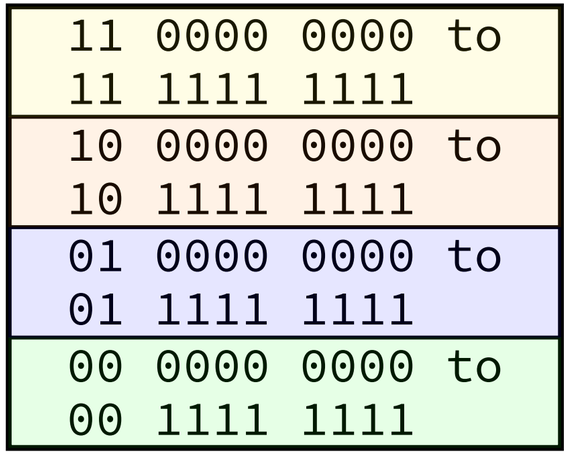


the rest of the address is called the *page offset*

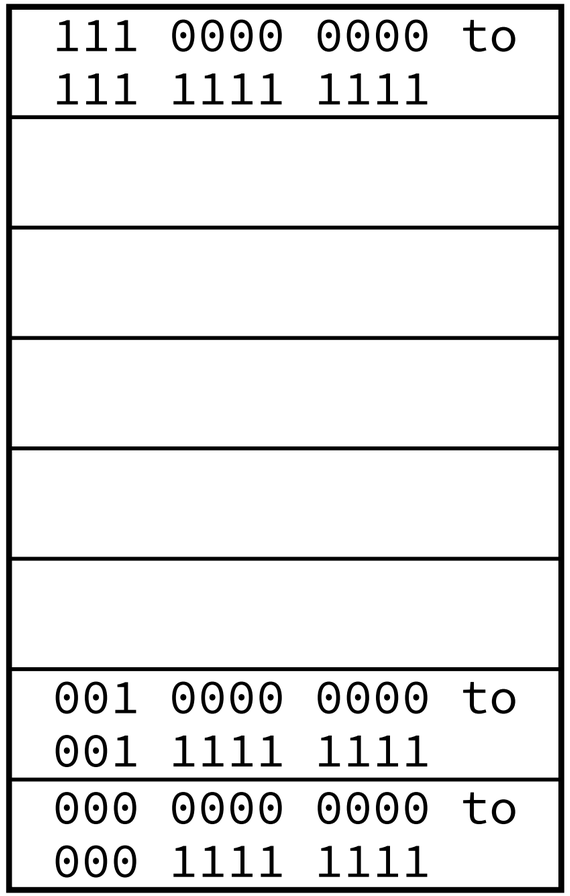
toy physical memory

toy physical memory

program memory
virtual addresses



real memory
physical addresses



toy physical memory

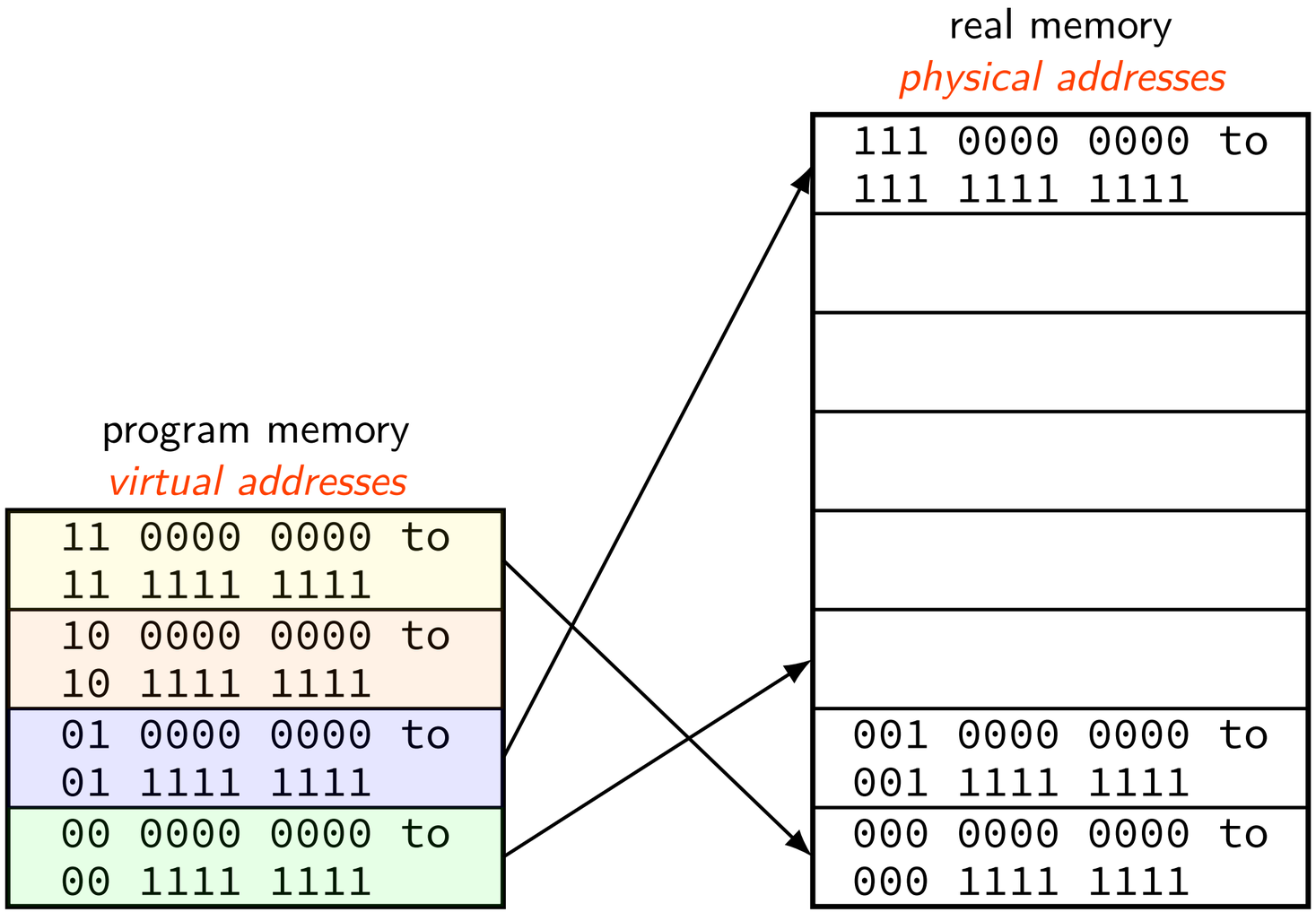
program memory
virtual addresses

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

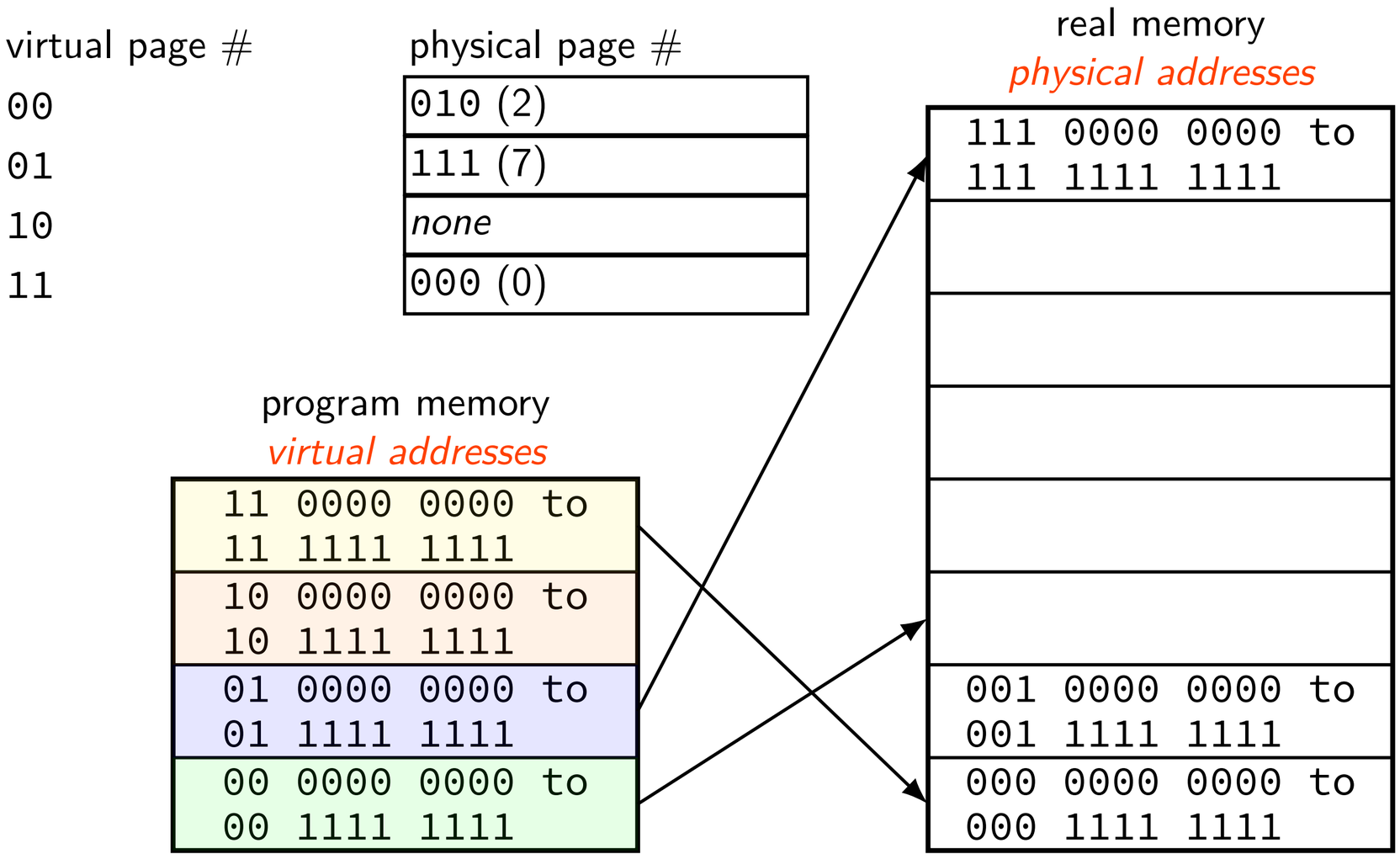
real memory
physical addresses

111 0000 0000 to *111* 1111 1111	physical page 7
001 0000 0000 to *001* 1111 1111	physical page 1
000 0000 0000 to *000* 1111 1111	physical page 0

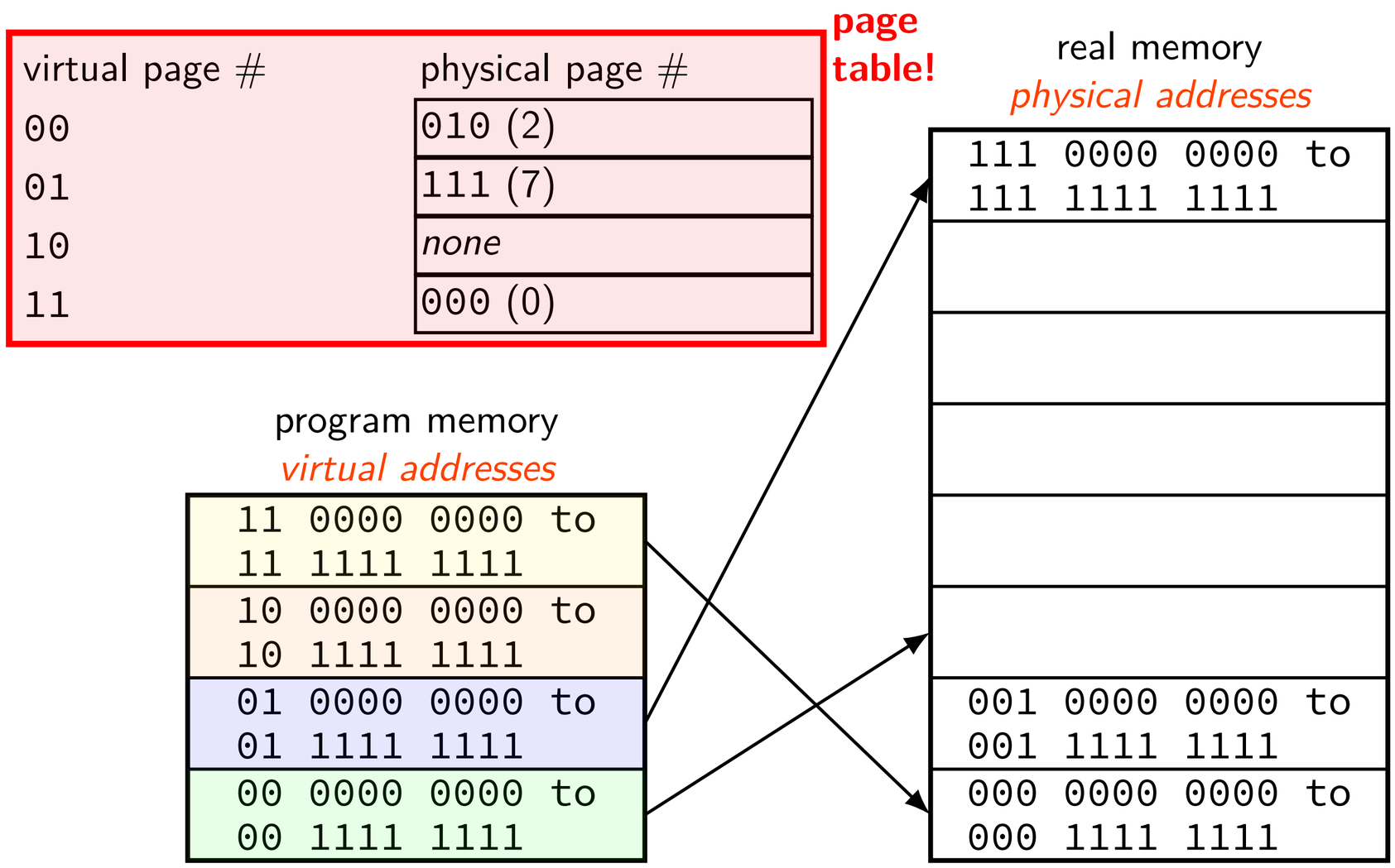
toy physical memory



toy physical memory



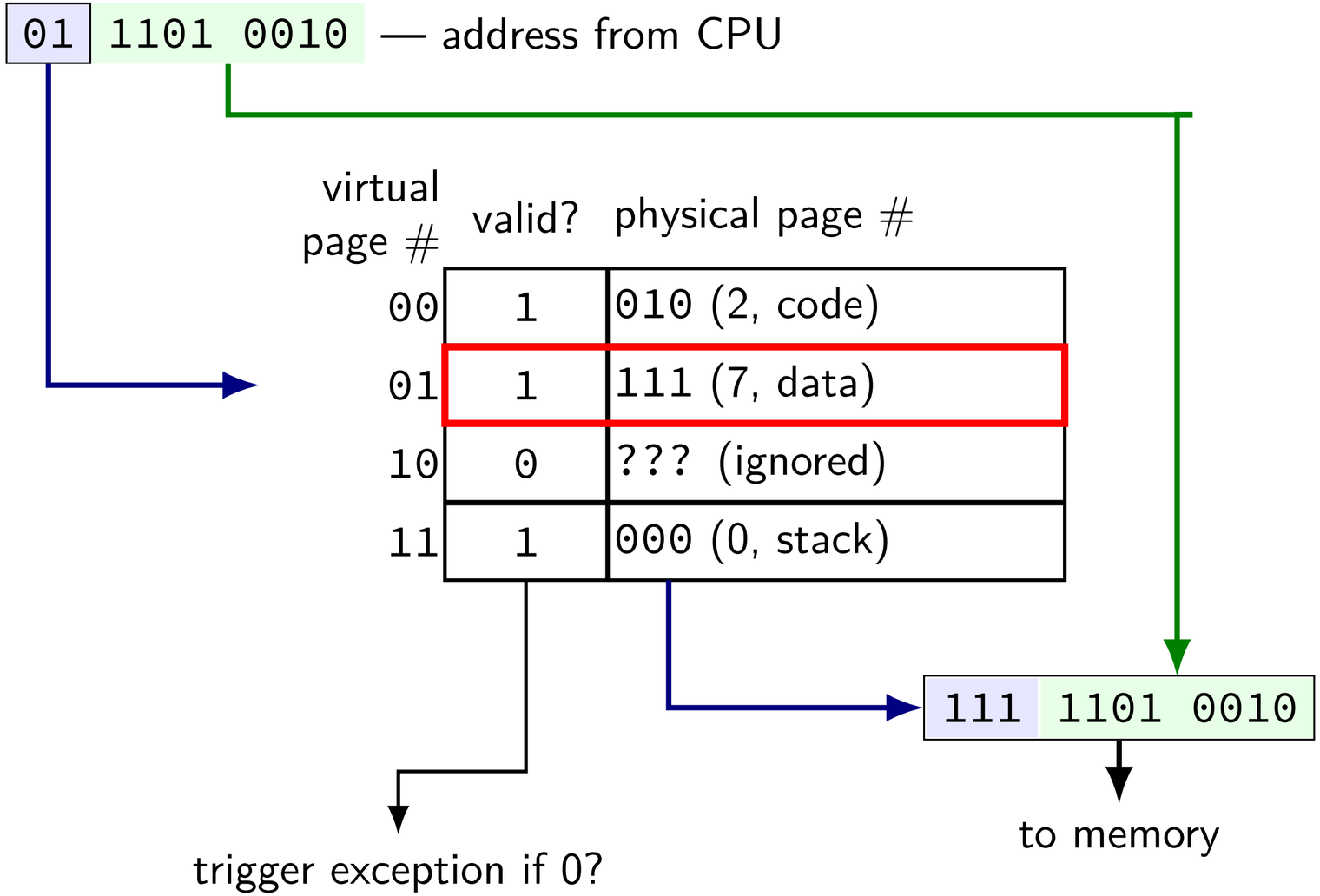
toy physical memory



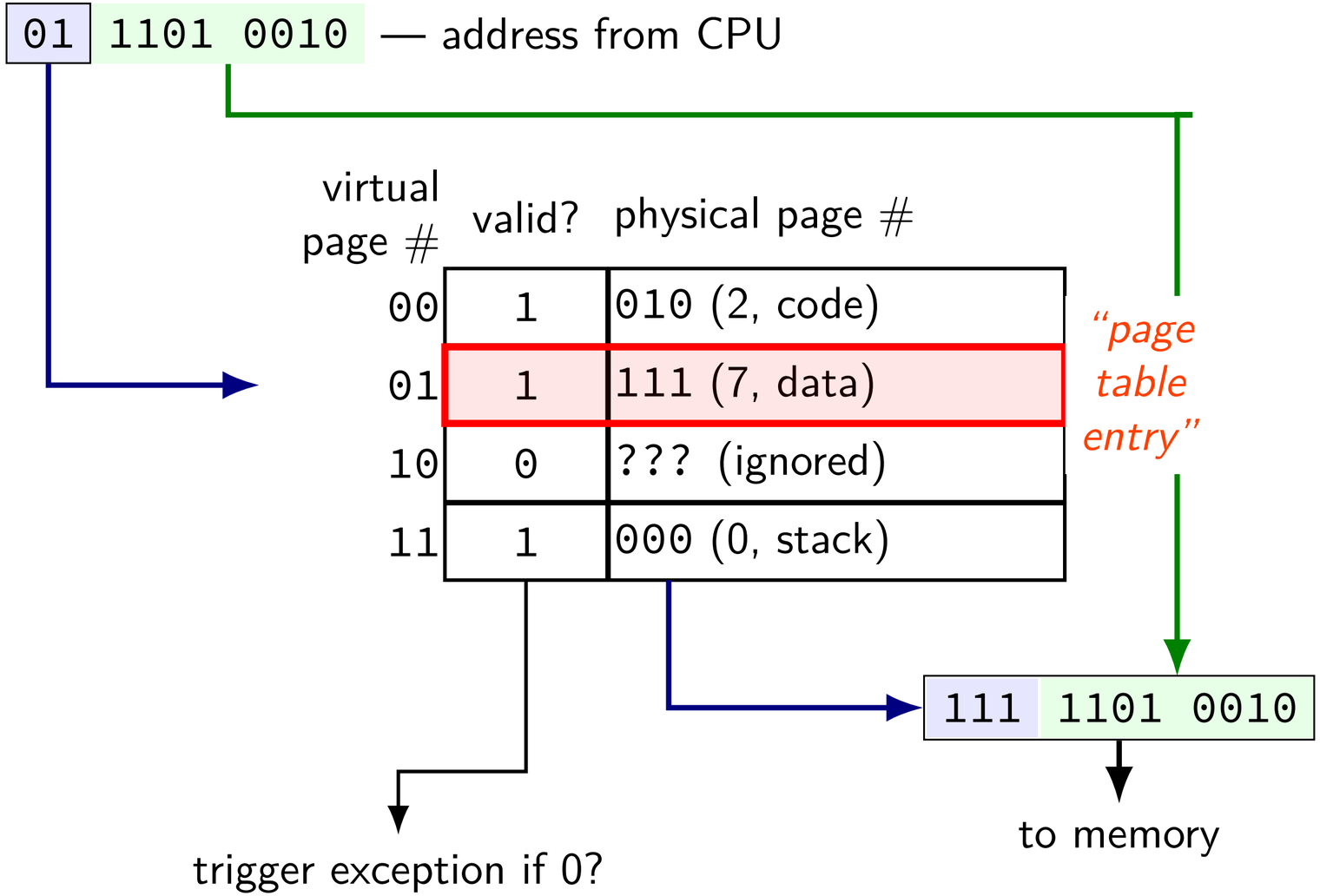
toy page table lookup

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

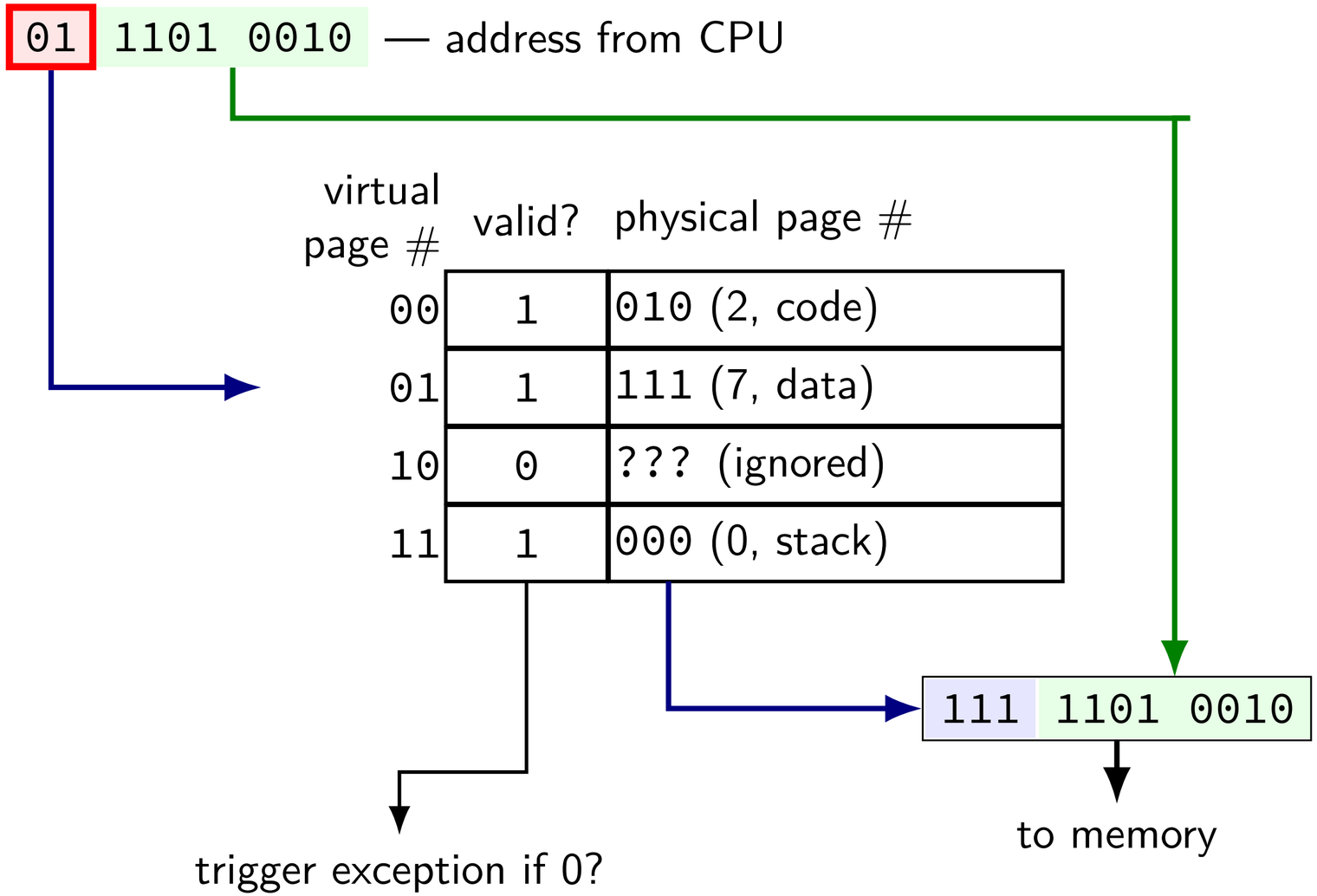
toy page table lookup



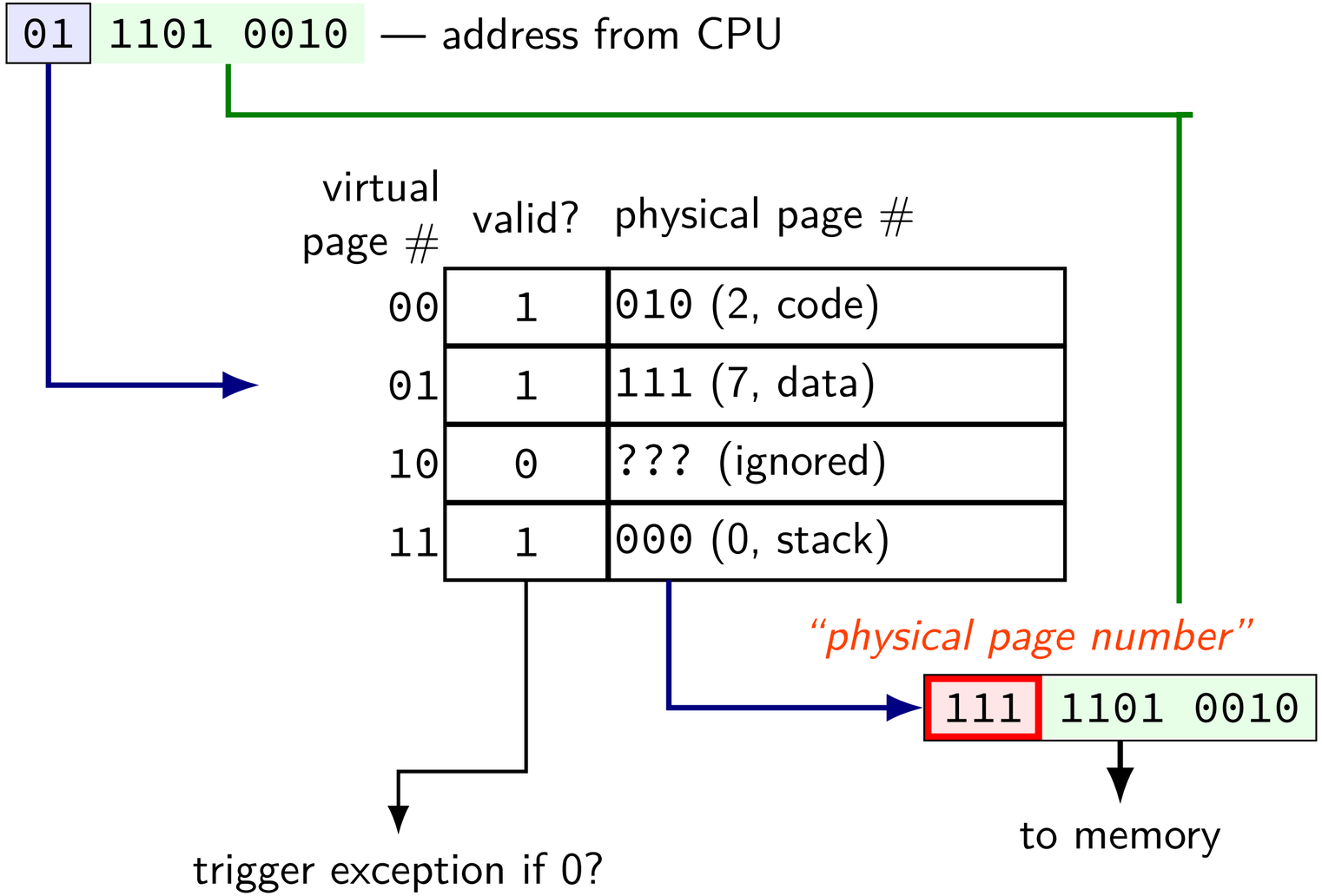
toy page table lookup



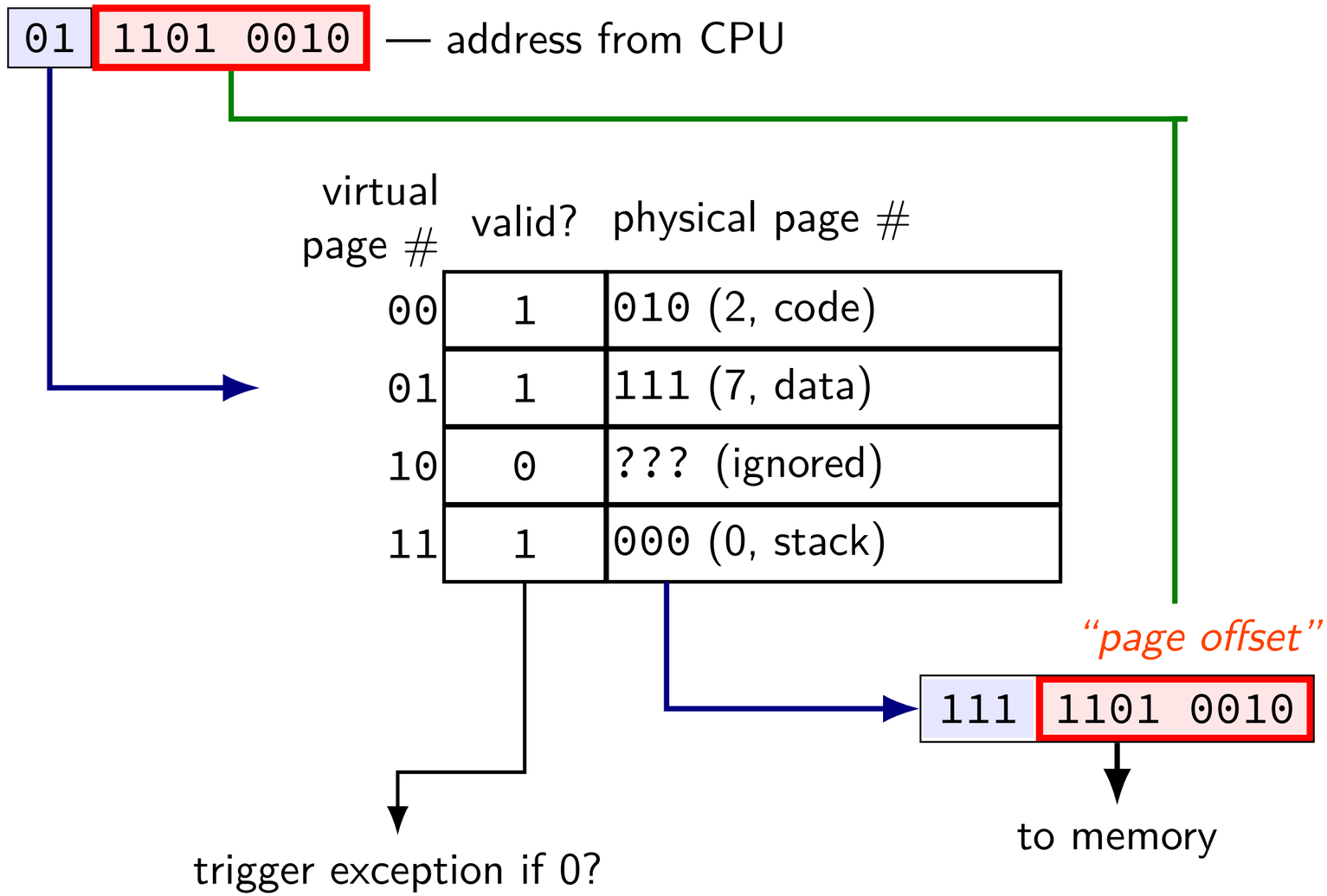
toy page table lookup



toy page table lookup



toy page table lookup



on virtual address sizes

virtual address size = size of pointer?

often, but — sometimes part of pointer not used

example: typical x86-64 only use 48 bits

- rest of bits have fixed value

virtual address size is amount used for mapping

address space sizes

amount of stuff that can be addressed = address space size

- based on number of unique addresses

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

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

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

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

exercise: page counting

suppose 32-bit virtual (program) addresses

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

how many virtual pages?

exercise: page counting

suppose 32-bit virtual (program) addresses

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

how many virtual pages?

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

exercise: page table size

suppose 32-bit virtual (program) addresses

suppose 30-bit physical (hardware) addresses

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

page table entries have physical page #, valid bit

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

exercise: page table size

suppose 32-bit virtual (program) addresses

suppose 30-bit physical (hardware) addresses

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

page table entries have physical page #, valid bit

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

2^{20} entries $\times (18 + 1)$ bits per entry

exercise: page table size

suppose 32-bit virtual (program) addresses

suppose 30-bit physical (hardware) addresses

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

page table entries have physical page #, valid bit

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

2^{20} entries $\times (18 + 1)$ bits per entry

19922944 bits = 2490368 bytes

exercise: page table size

suppose 32-bit virtual (program) addresses

suppose 30-bit physical (hardware) addresses

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

page table entries have physical page #, valid bit

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

2^{20} entries $\times (18 + 1)$ bits per entry

19922944 bits = 2490368 bytes

issue: where can we store that?

exercise: address splitting

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

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

exercise: address splitting

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

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

page number: `0x12345`; offset: `0x678`

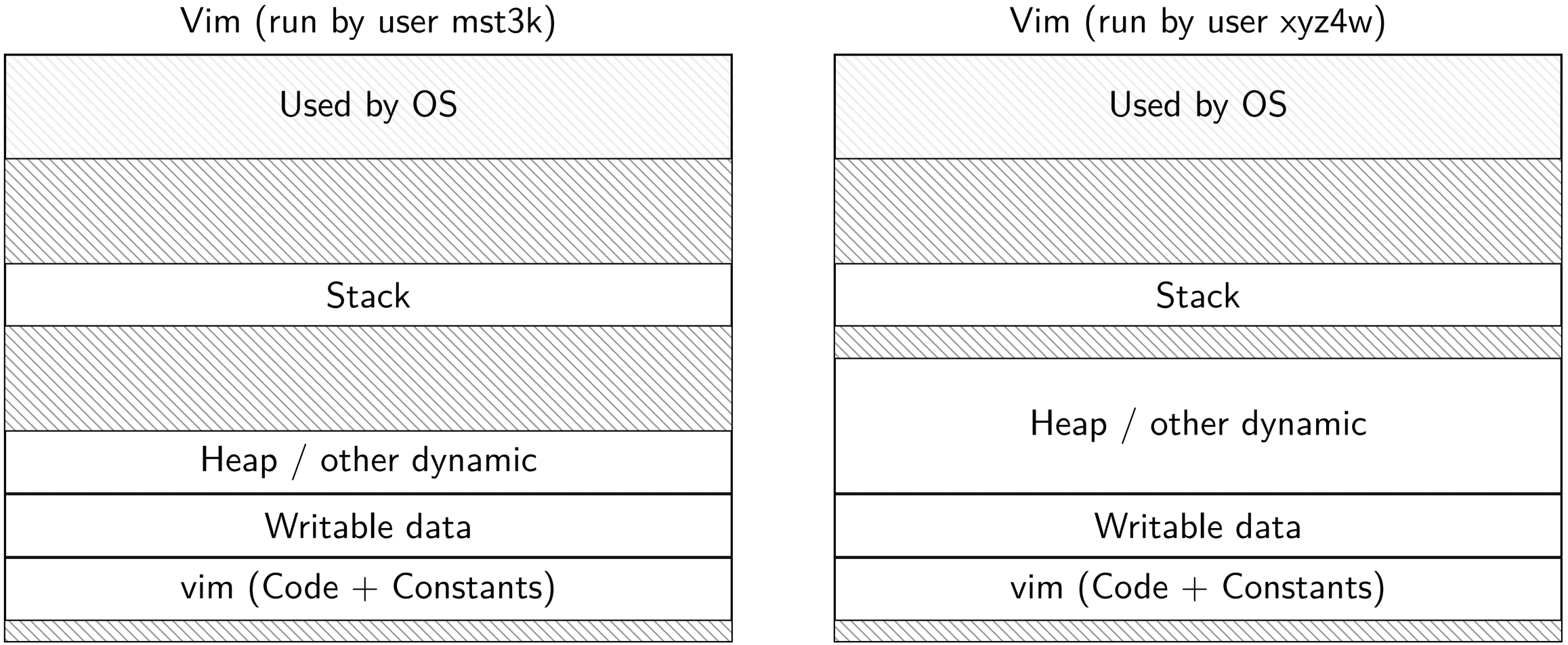
exercise: page table lookup

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

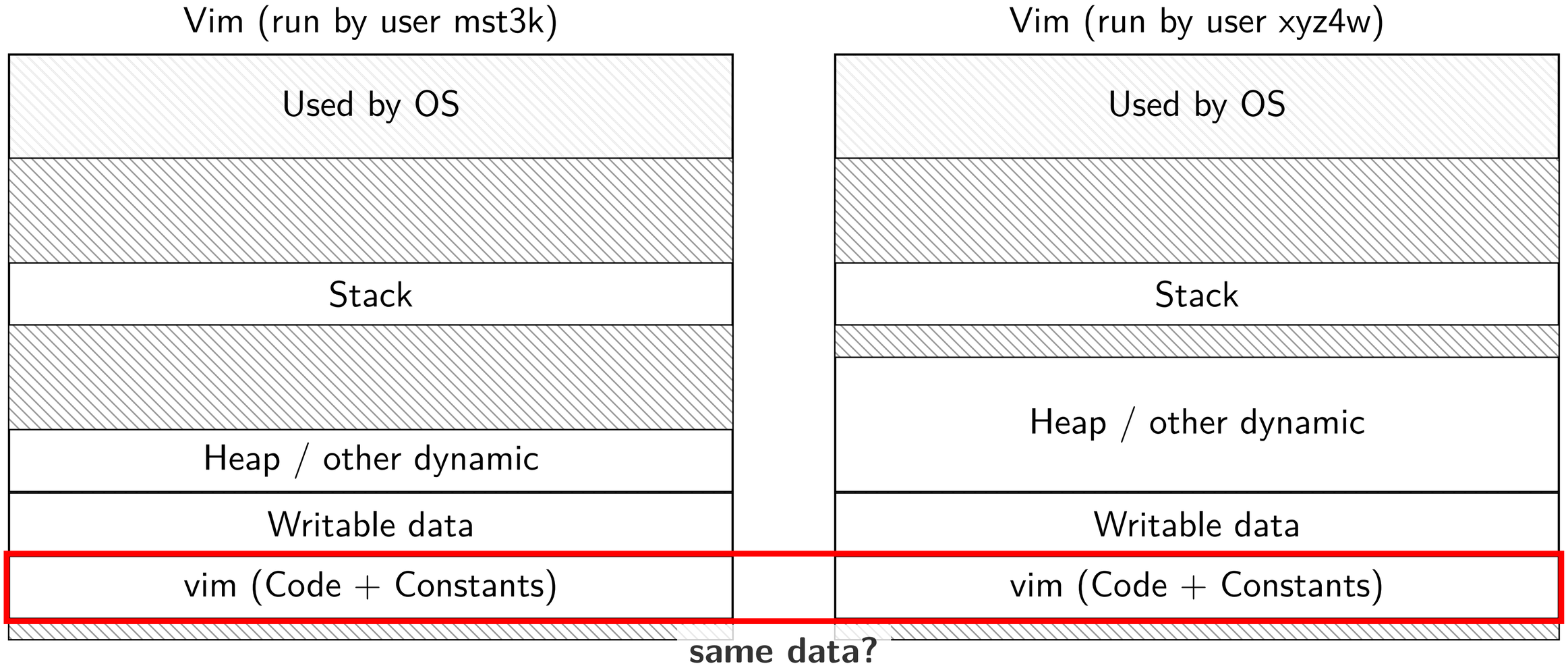
VPN	valid	PPN
000	1	0010
001	1	1010
010	0	—
011	0	—
100	1	1110
101	1	0100
110	1	0001
111	0	—

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

vim (two copies)



vim (two copies)



two copies of program

would like to only have one copy of program code

could setup both page tables to point to same code, but…

what if `mst3k`'s vim has bug and modifies its code

expect this to break `mst3k`'s vim, but…

would also break `xyz4w`'s vim

violates process abstraction:

"illusion of own memory"

permissions bits

page table entry will have more *permissions bits*

- can access in user mode?
- can read from?
- can write to?
- can execute from?

checked by hardware like valid bit

page table (logically)

virtual page #	valid?	user?	write?	exec?	physical page #
0000 0000	0	0	0	0	00 0000 0000
0000 0001	1	1	1	0	10 0010 0110
0000 0010	1	1	1	0	00 0000 1100
0000 0011	1	1	0	1	11 0000 0011
...					
1111 1111	1	0	1	0	00 1110 1000

running 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

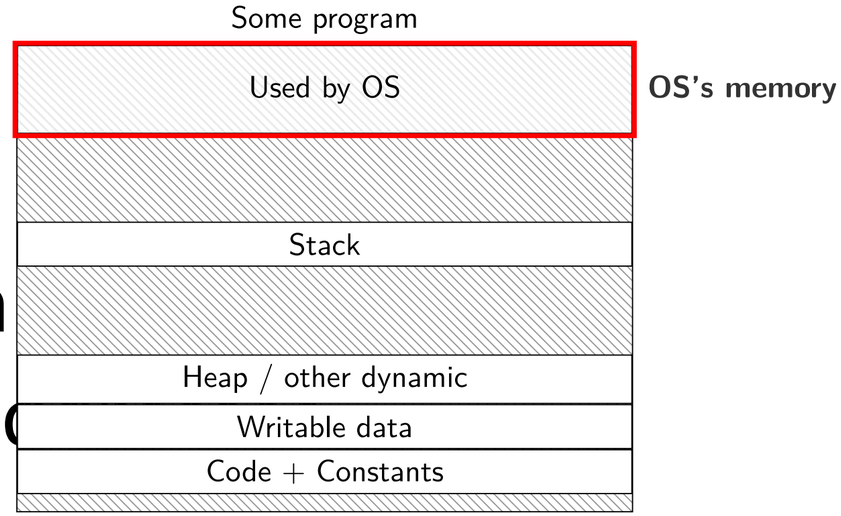
OS memory in programs

most OSes: kernel runs using

normal programs' page tables:

avoids page table change on exception

$\Longrightarrow$ kernel code/data needs virtual ac



typically: high-numbered virtual pages reserved for this

those page table entries marked non-user-accessible

(can't allow programs to mess with OS's stuff)

usually copied in every process's page table

invalid pages

model so far: page tables = logical memory of program

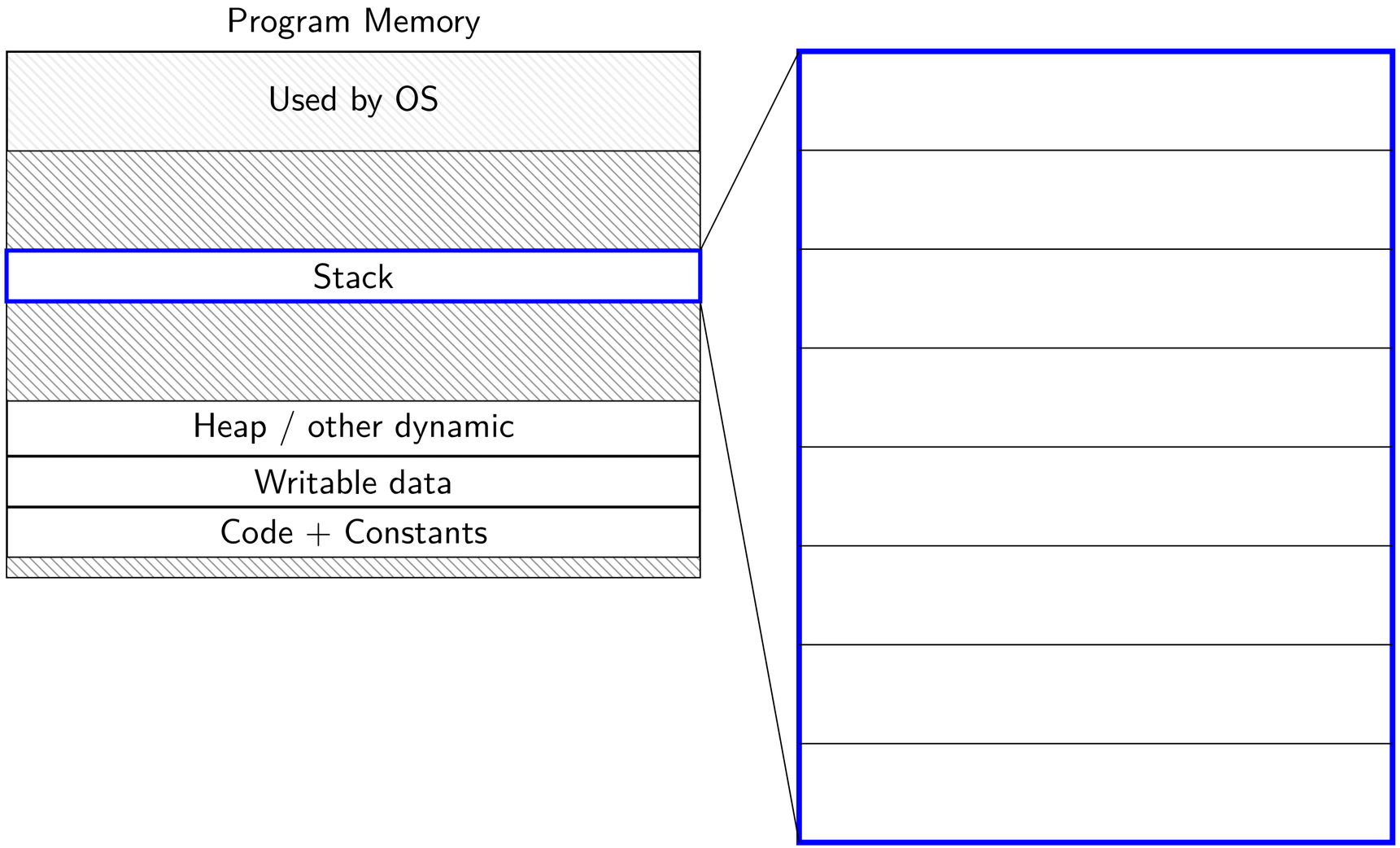
- invalid page $\implies$ process accessing it should crash
- wrong permissions $\implies$ process accessing it should crash

but it doesn't need to work that way

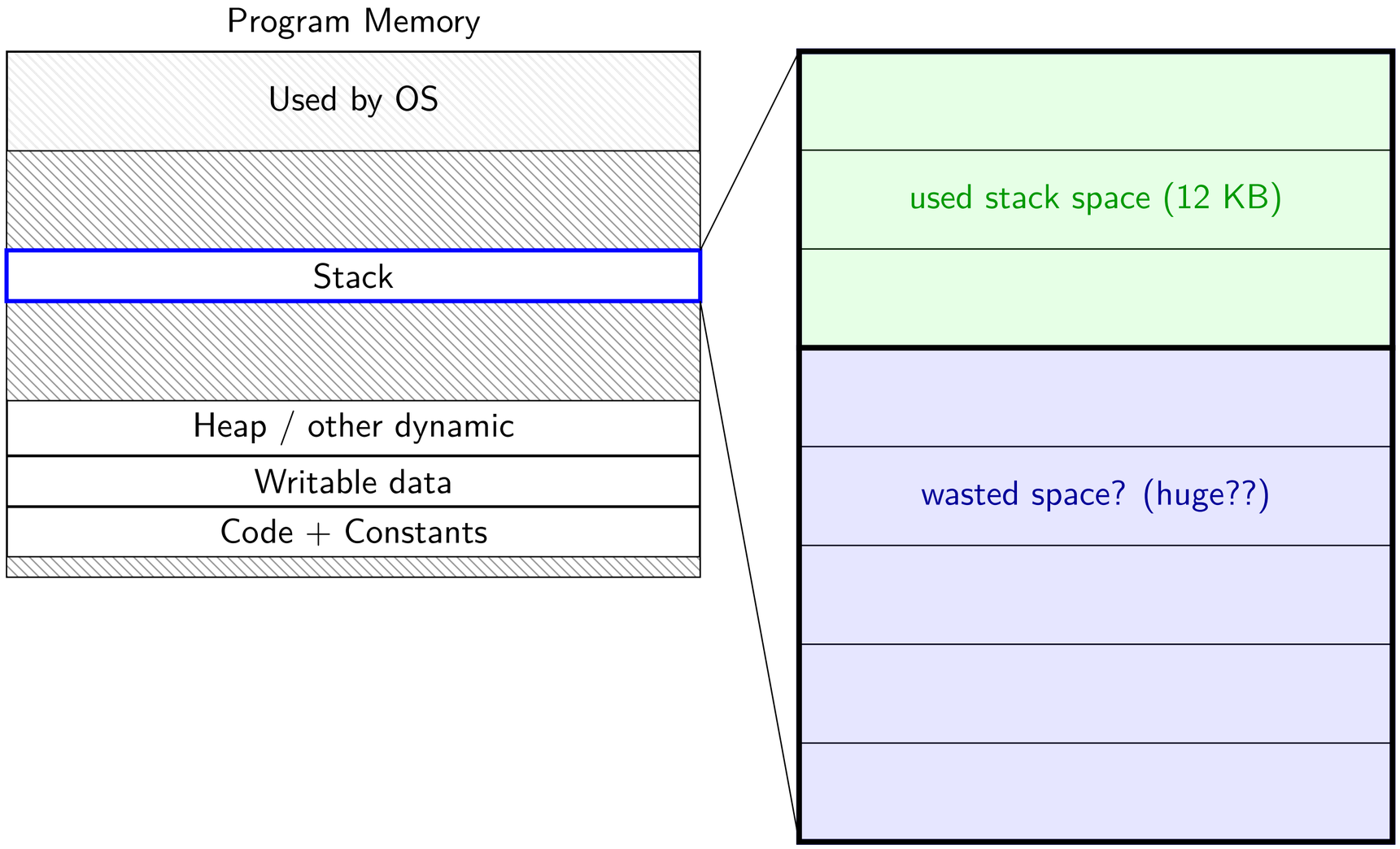
- and doesn't work that way in most OSes

gives operating system more flexibility!

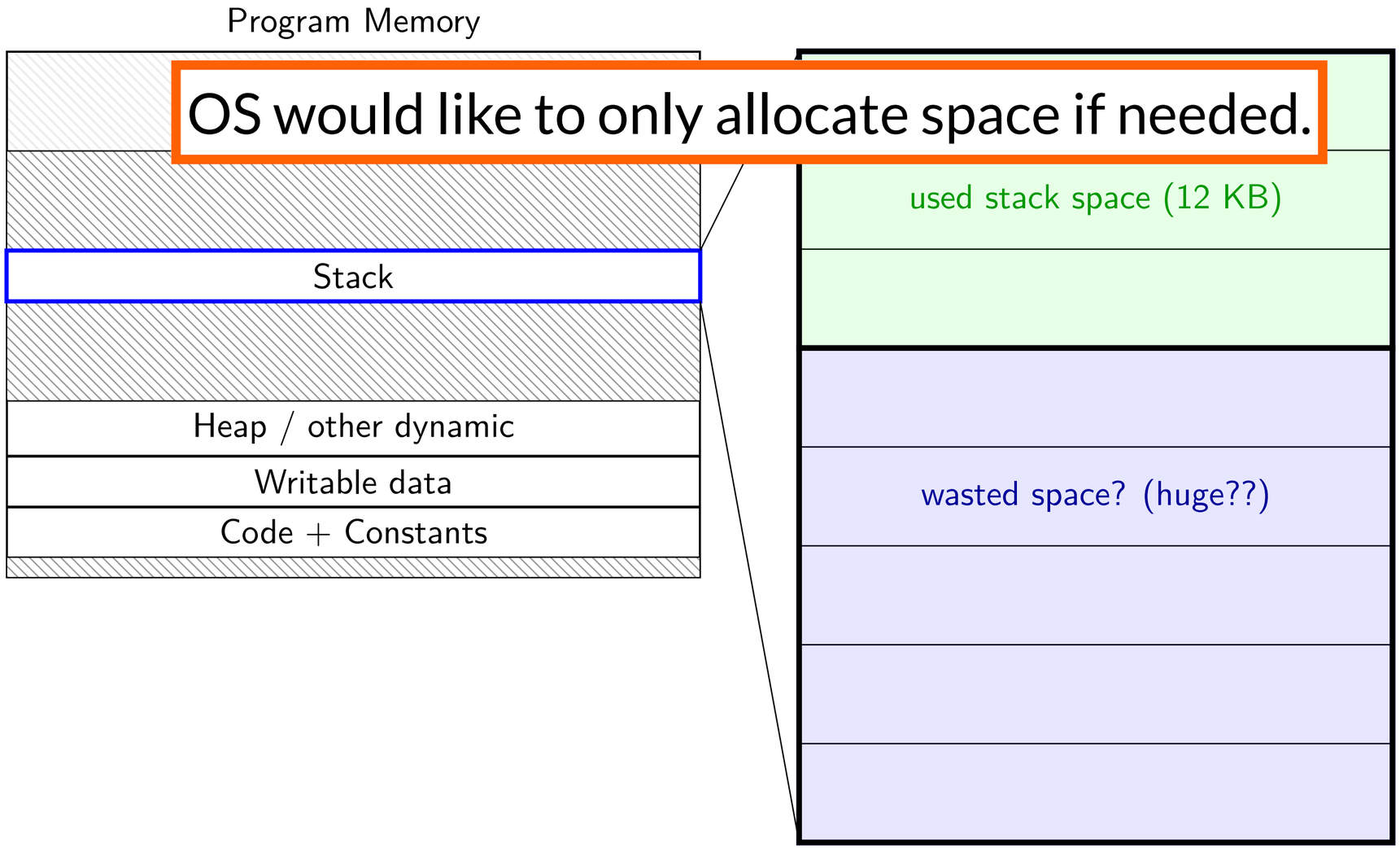
space on demand



space on demand



space on demand



allocating space on demand

`%rsp = 0x7FFFC000`

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

VPN
…
0x7FFFB
0x7FFFC
0x7FFFD
0x7FFFE
0x7FFFF
…

valid?	physical page
…	…
0	---
1	0x200DF
1	0x12340
1	0x12347
1	0x12345
…	…

allocating space on demand

`%rsp = 0x7FFFC000`

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

fault!

VPN
…
0x7FFFB
0x7FFFC
0x7FFFD
0x7FFFE
0x7FFFF
…

valid?	physical page
…	…
0	---
1	0x200DF
1	0x12340
1	0x12347
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

load from file, etc. instead of allocating empty page

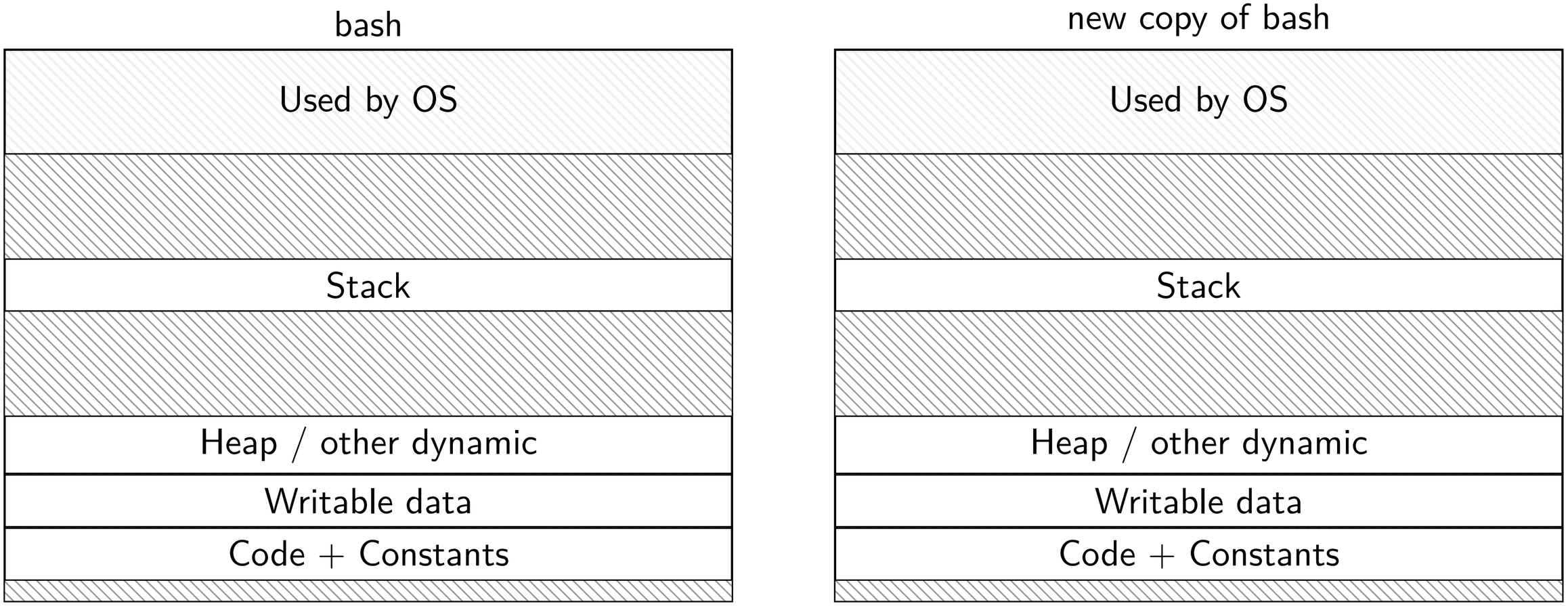
loading program can be *merely creating empty page table*

everything else can be handled *in response to page faults*

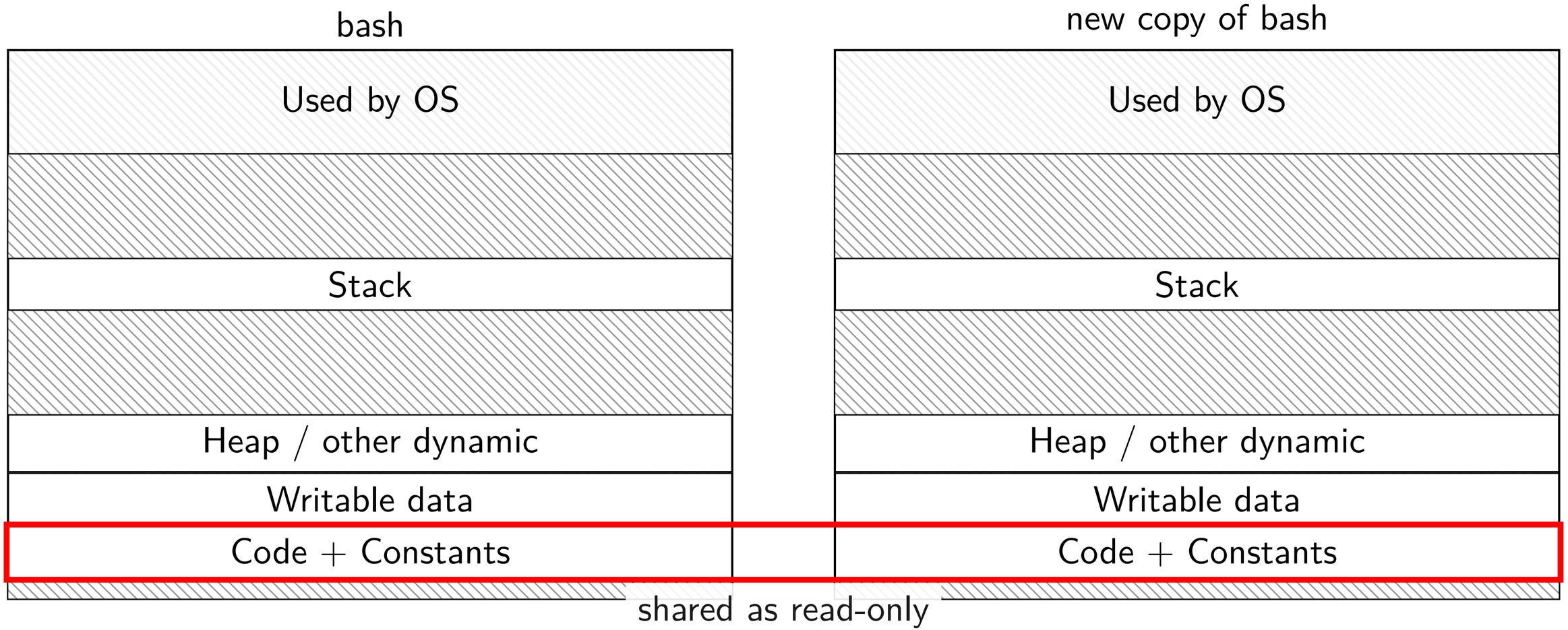
page fault = exception triggered for invalid/unperissions page

no time/space spent loading/allocating unneeded space

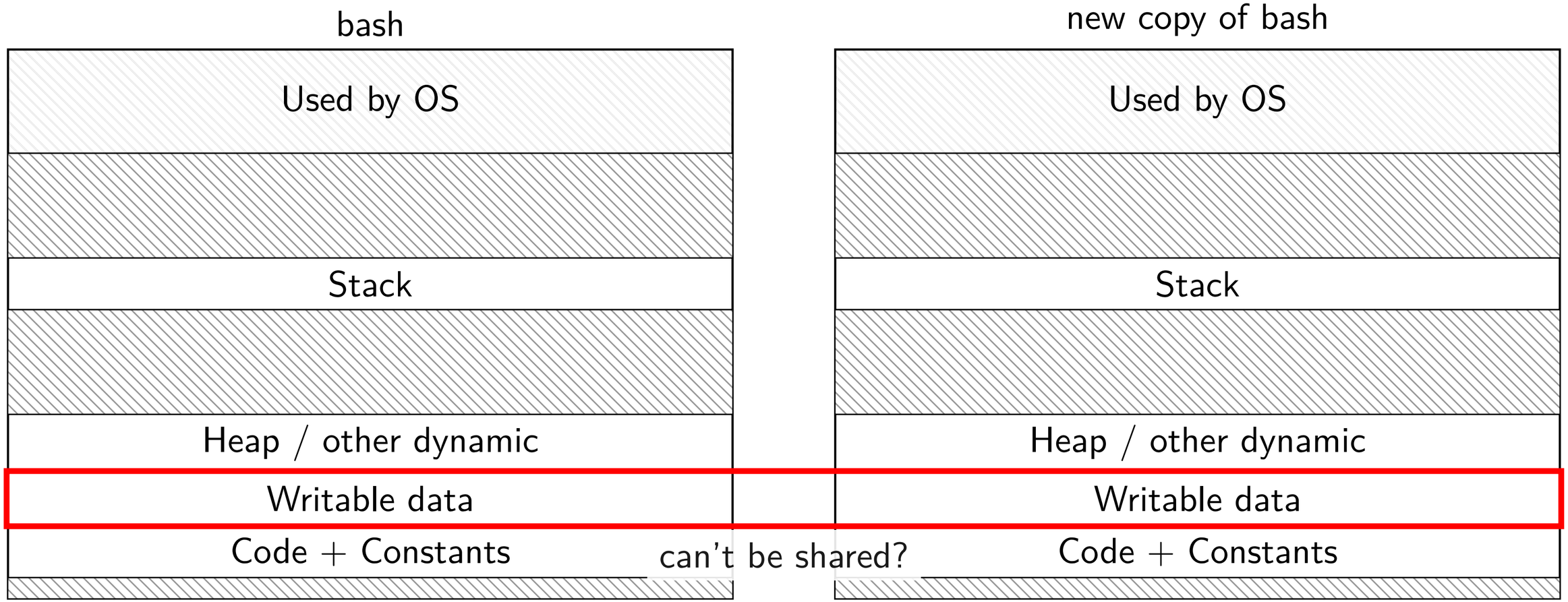
do we really need a complete copy?



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 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
			0x200DF
			0x200AF
			…

copy operation actually duplicates page table

both processes *share all physical pages*

but marked *read-only in both tables*

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 a read-only page triggers a page 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
…	…	…	…

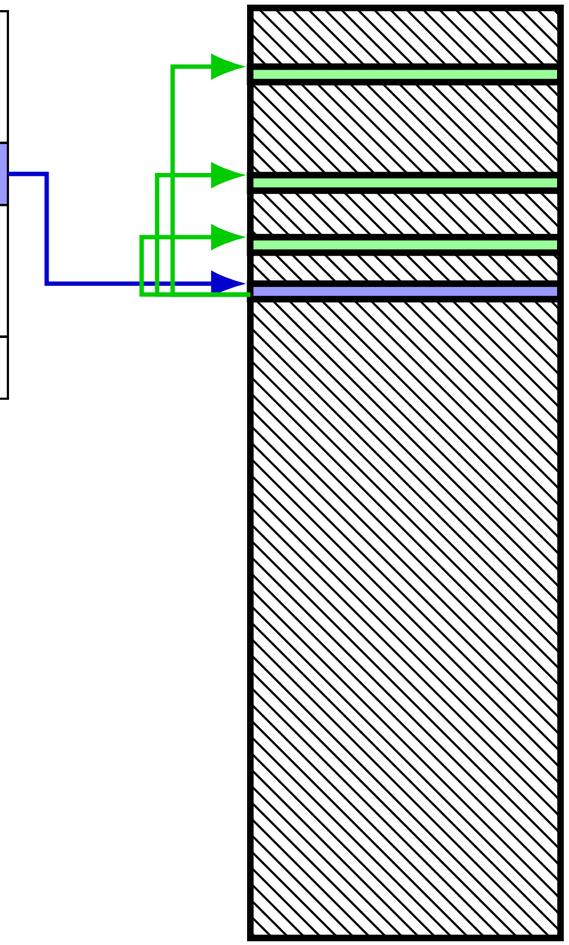
after allocating a copy, OS reruns the write instruction

fork (w/ copy-on-write, if parent writes first)

parent process info

user regs	`rax (return val.)=`~~`42`~~*child pid*, `rcx=133, …`
page tables	
open files	`fd 0: …` `fd 1: …`
…	…

memory



fork (w/ copy-on-write, if parent writes first)

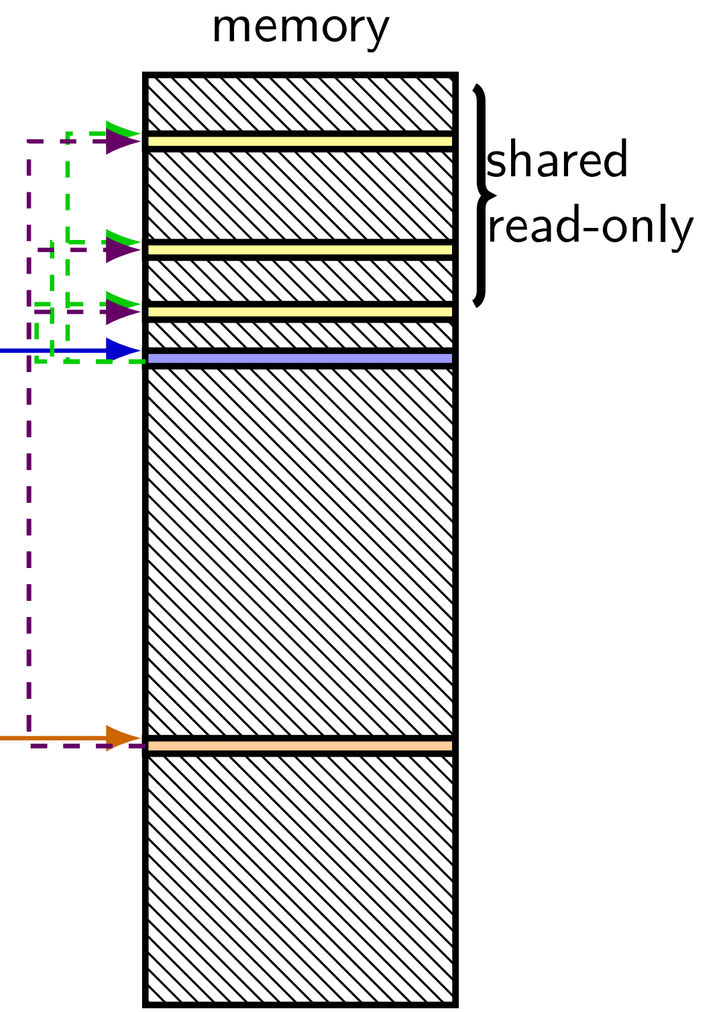
parent process info

user regs	`rax (return val.)=~~42~~`*child pid*`, rcx=133, …`
page tables	
open files	`fd 0: … fd 1: …`
…	…

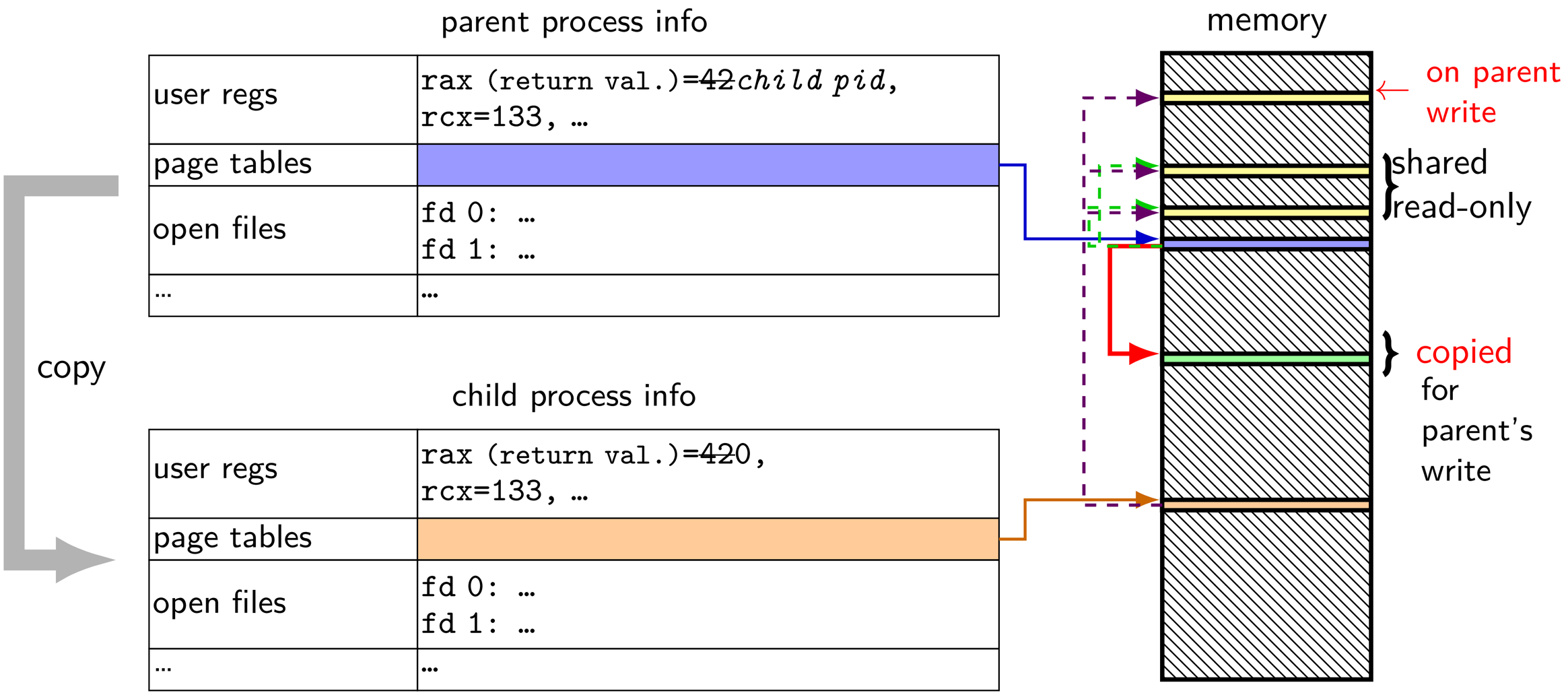
copy

child process info

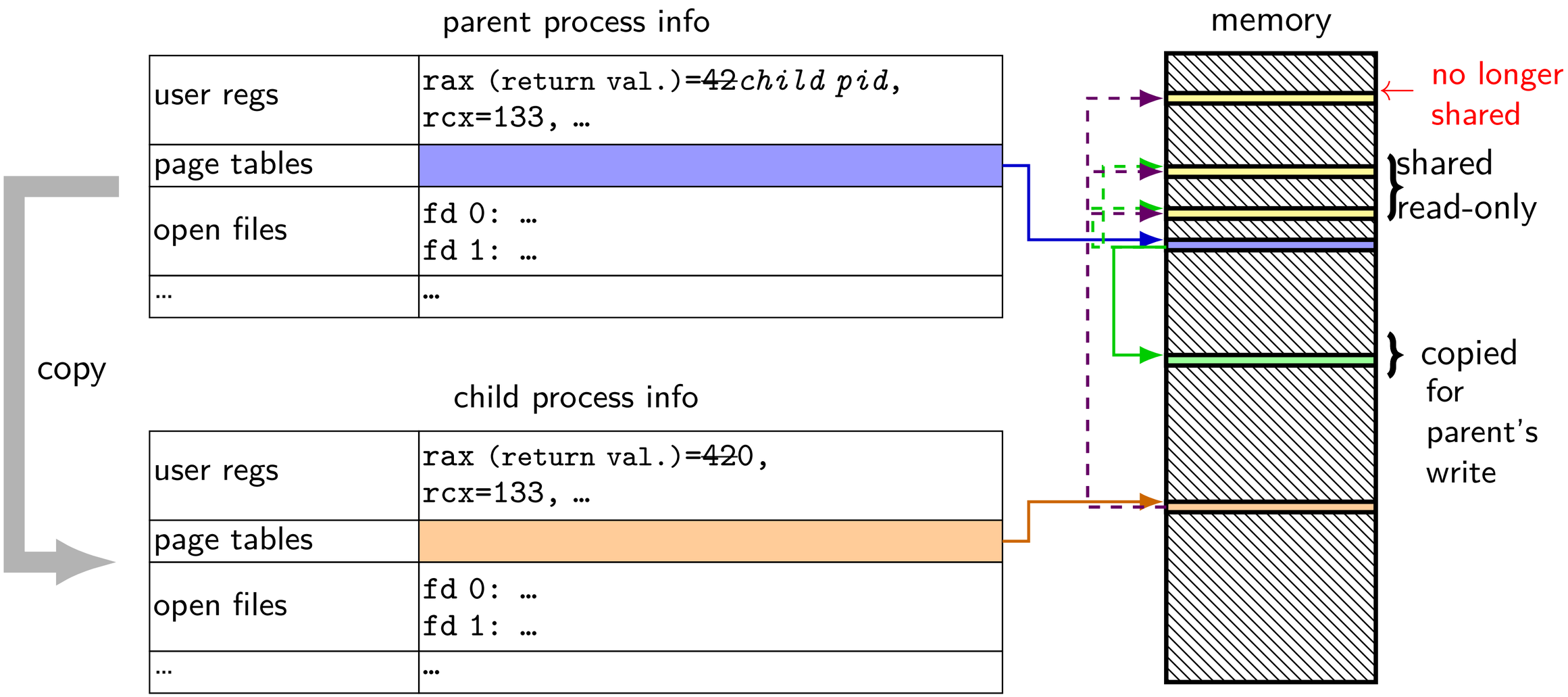
user regs	`rax (return val.)=~~42~~0, rcx=133, …`
page tables	
open files	`fd 0: … fd 1: …`
…	…



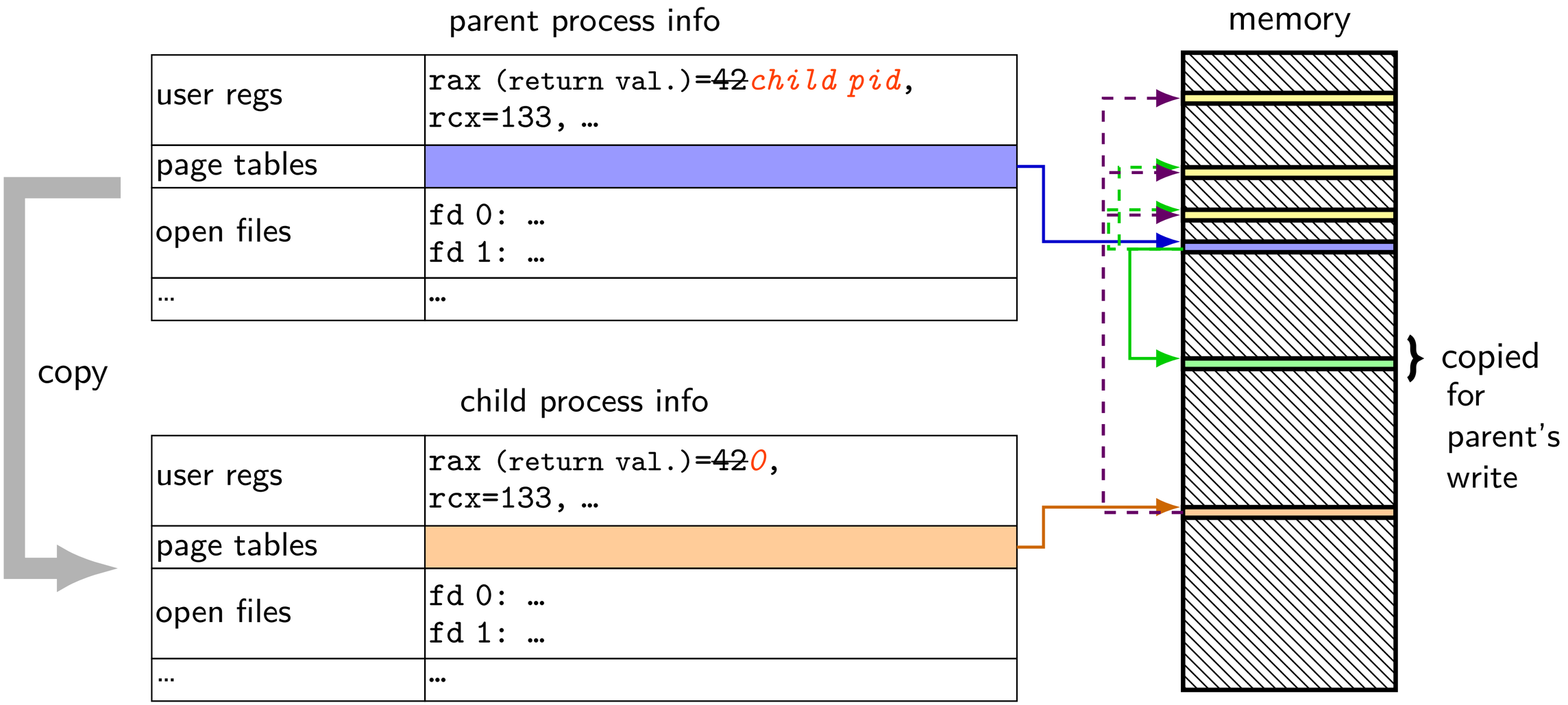
fork (w/ copy-on-write, if parent writes first)



fork (w/ copy-on-write, if parent writes first)



fork (w/ copy-on-write, if parent writes first)



mmap

Linux/Unix has a function to “map” a file to memory

```
int file = open("somefile.dat", O_RDWR);

    // data is region of memory that represents file
char *data = mmap(..., file, 0);

    // read byte 6 from somefile.dat
char seventh_char = data[6];

   // modifies byte 100 of somefile.dat
data[100] = 'x';
    // can continue to use 'data' like an array
```

Linux memory tracking

Linux kernel treats all program memory as special case of mmap'd file

two major features to help

'private' mappings = copy-on-write from original file

- supports 'load program from disk on demand'
- pages not shared with original file after writes

original file = all zeroes (instead of file from disk)

- "anonymous mapping"
- handles 'allocate-on-demand' scenario

Linux maps: list of maps

```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 48328831                           /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 48328831                           /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 48328831                           /bin/cat
01974000-01995000 rw-p 00000000 00:00 0                                  [heap]
7f60c718b000-7f60c7490000 r--p 00000000 08:01 77483660  /usr/lib/locale/locale-archive
7f60c7490000-7f60c764e000 r-xp 00000000 08:01 96659129  /lib/x86_64-linux-gnu/libc-2.19.so
7f60c764e000-7f60c784e000 ---p 001be000 08:01 96659129  /lib/x86_64-linux-gnu/libc-2.19.so
7f60c784e000-7f60c7852000 r--p 001be000 08:01 96659129  /lib/x86_64-linux-gnu/libc-2.19.so
7f60c7852000-7f60c7854000 rw-p 001c2000 08:01 96659129  /lib/x86_64-linux-gnu/libc-2.19.so
7f60c7854000-7f60c7859000 rw-p 00000000 00:00 0
7f60c7859000-7f60c787c000 r-xp 00000000 08:01 96659109  /lib/x86_64-linux-gnu/ld-2.19.so
7f60c7a39000-7f60c7a3b000 rw-p 00000000 00:00 0
7f60c7a7a000-7f60c7a7b000 rw-p 00000000 00:00 0
7f60c7a7b000-7f60c7a7c000 r--p 00022000 08:01 96659109  /lib/x86_64-linux-gnu/ld-2.19.so
7f60c7a7c000-7f60c7a7d000 rw-p 00023000 08:01 96659109  /lib/x86_64-linux-gnu/ld-2.19.so
7f60c7a7d000-7f60c7a7e000 rw-p 00000000 00:00 0
7ffc5d2b2000-7ffc5d2d3000 rw-p 00000000 00:00 0                          [stack]
7ffc5d3b0000-7ffc5d3b3000 r--p 00000000 00:00 0                          [vvar]
7ffc5d3b3000-7ffc5d3b5000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
```

Linux maps: list of maps

```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 48328831           /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 48328831           /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 48328831           /bin/cat
01974000-01995000 rw-p 00000000 00:00 0                  [heap]
7f60c718b000-7f60c7490000 r--p 00000000 08:01 77483660   /usr/lib/locale/locale-archive
7f60c7490000-7f60c764e000 r-xp 00000000 08:01 96659129   /lib/x86_64-linux-gnu/libc-2.19.so
7f60c764e000-7f60c784e000 ---p 001be000 08:01 96659129   /lib/x86_64-linux-gnu/libc-2.19.so
7f60c784e000-7f60c7852000 r--p 001be000 08:01 96659129   /lib/x86_64-linux-gnu/libc-2.19.so
7f60c7852000-7f60c7854000 rw-p 001c2000 08:01 96659129   /lib/x86_64-linux-gnu/libc-2.19.so
7f60c7854000-7f60c7859000 rw-p 00000000 00:00 0
7f60c7859000-7f60c787c000 r-xp 00000000 08:01 96659109   /lib/x86_64-linux-gnu/ld-2.19.so
7f60c7a39000-7f60c7a3b000 rw-p 00000000 00:00 0
7f60c7a7a000-7f60c7a7b000 rw-p 00000000 00:00 0
7f60c7a7b000-7f60c7a7c000 r--p 00022000 08:01 96659109   /lib/x86_64-linux-gnu/ld-2.19.so
7f60c7a7c000-7f60c7a7d000 rw-p 00023000 08:01 96659109   /lib/x86_64-linux-gnu/ld-2.19.so
7f60c7a7d000-7f60c7a7e000 rw-p 00000000 00:00 0
7ffc5d2b2000-7ffc5d2d3000 rw-p 00000000 00:00 0          [stack]
7ffc5d3b0000-7ffc5d3b3000 r--p 00000000 00:00 0          [vvar]
7ffc5d3b3000-7ffc5d3b5000 r-xp 00000000 00:00 0          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
```

Linux maps: list of maps

```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 48328831                           /bin/cat
0060a000-0060b000 r--p 0000a000 08
0060b000-0060c000 rw-p 0000b000 08
01974000-01995000 rw-p 00000000 00
7f60c718b000-7f60c7490000 r--p 000                                       archive
7f60c7490000-7f60c764e000 r-xp 000                                       ibc-2.19.so
7f60c764e000-7f60c784e000 ---p 001                                       ibc-2.19.so
7f60c784e000-7f60c7852000 r--p 001                                       ibc-2.19.so
7f60c7852000-7f60c7854000 rw-p 001                                       ibc-2.19.so
7f60c7854000-7f60c7859000 rw-p 000
7f60c7859000-7f60c787c000 r-xp 000                                       d-2.19.so
7f60c7a39000-7f60c7a3b000 rw-p 000
7f60c7a7a000-7f60c7a7b000 rw-p 000
7f60c7a7b000-7f60c7a7c000 r--p 000                                       d-2.19.so
7f60c7a7c000-7f60c7a7d000 rw-p 00023000 08:01 96659109                   /lib/x86_64-linux-gnu/ld-2.19.so
7f60c7a7d000-7f60c7a7e000 rw-p 00000000 00:00 0
7ffc5d2b2000-7ffc5d2d3000 rw-p 00000000 00:00 0                          [stack]
7ffc5d3b0000-7ffc5d3b3000 r--p 00000000 00:00 0                          [vvar]
7ffc5d3b3000-7ffc5d3b5000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

OS tracks list of `struct vm_area_struct` with:

- virtual address start, end
- permissions
- offset in backing file (if any)
- pointer to backing file (if any)

(not shown):

- info about sharing of non-file data

page tricks generally

deliberately *make program trigger page/protection fault*

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

have *seperate data structures* represent logically allocated memory

e.g. "addresses `0x7FFF8000` to `0x7FFFFFFF` are the stack"

page table is for the hardware and not the OS

example page table tricks

allocating space on demand

loading code/data from files on disk on demand

copy-on-write

saving data temporarily to disk, reloading to memory on demand

"swapping"

detecting whether memory was read/written recently

stopping in a debugger when a variable is modified

sharing memory between programs on two different machines

example page table tricks

allocating space on demand

loading code/data from files on disk on demand

copy-on-write

saving data temporarily to disk, reloading to memory on demand

“swapping”

detecting whether memory was read/written recently

stopping in a debugger when a variable is modified

sharing memory between programs on two different machines

example page table tricks

allocating space on demand

loading code/data from files on disk on demand

copy-on-write

saving data temporarily to disk, reloading to memory on demand

"swapping"

detecting whether memory was read/written recently

stopping in a debugger when a variable is modified

sharing memory between programs on two different machines

example page table tricks

allocating space on demand

loading code/data from files on disk on demand

copy-on-write

saving data temporarily to disk, reloading to memory on demand

"swapping"

detecting whether memory was read/written recently

stopping in a debugger when a variable is modified

sharing memory between programs on two different machines

example page table tricks

allocating space on demand

loading code/data from files on disk on demand

copy-on-write

saving data temporarily to disk, reloading to memory on demand

"swapping"

detecting whether memory was read/written recently

stopping in a debugger when a variable is modified

sharing memory between programs on two different machines

hardware help for page table tricks

information about the address causing the fault

- e.g. special register with memory address accessed
- harder alternative: OS disassembles instruction, look at registers

(by default) rerun faulting instruction when returning from exception

precise exceptions: no side effects from faulting instruction or after

- e.g. `pushq` that caused did not change `%rsp` before fault
- e.g. can't notice if instructions were executed in parallel

exercise setup

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

page table

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

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

exercise setup

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

page table

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

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

phys. page 0

phys. page 1

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

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages (virtual addresses) 0x18 = ; 0x03 = ; 0x0A = ; 0x13 =

page table

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

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

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages (virtual addresses) *0x18* = ; `0x03` = ; `0x0A` = ; `0x13` =

page table

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

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

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages (virtual addresses) 0x18 = *00*; `0x03` = ; `0x0A` = ; `0x13` =

page table

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

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

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages (virtual addresses) 0x18 =00; `0x03` = *0x4A*; `0x0A` = ; `0x13` =

page table

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

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

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages (virtual addresses) 0x18 =00; `0x03` = 0x4A; `0x0A` = *0xDC*; `0x13` =

page table

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

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

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages (virtual addresses) 0x18 =00; `0x03` = 0x4A; `0x0A` = 0xDC; `0x13` = fault

page table

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

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

page tables in memory

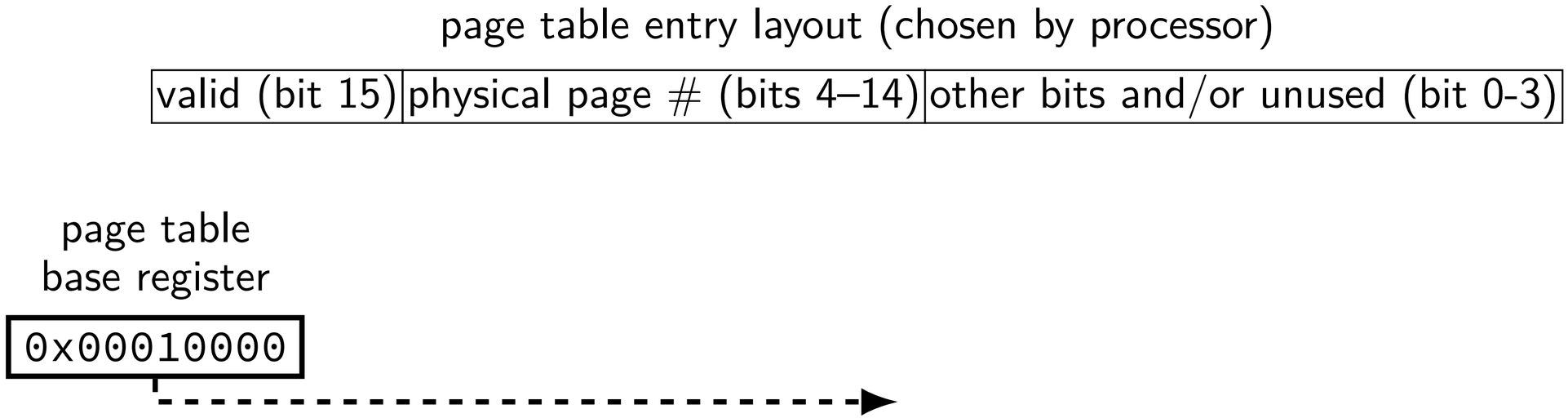
where can processor store megabytes of page tables? *in memory*

page table entry layout (chosen by processor)

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

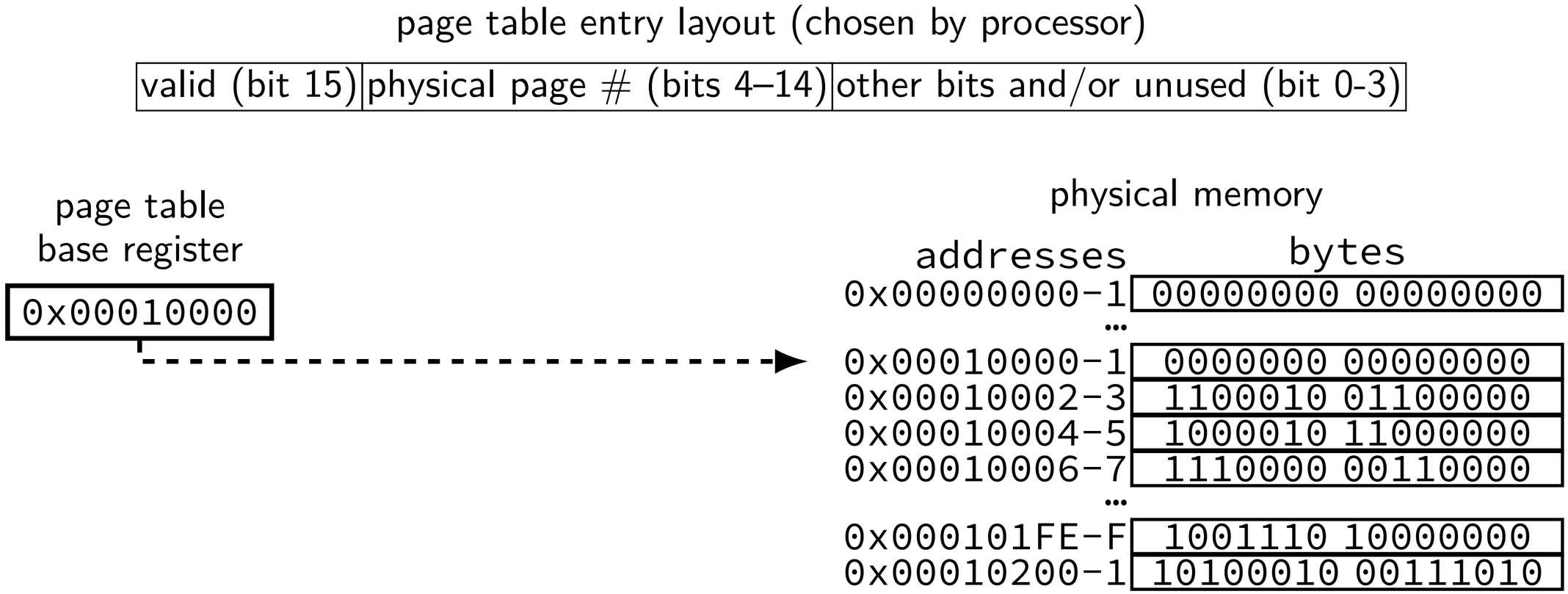
page tables in memory

where can processor store megabytes of page tables? *in memory*



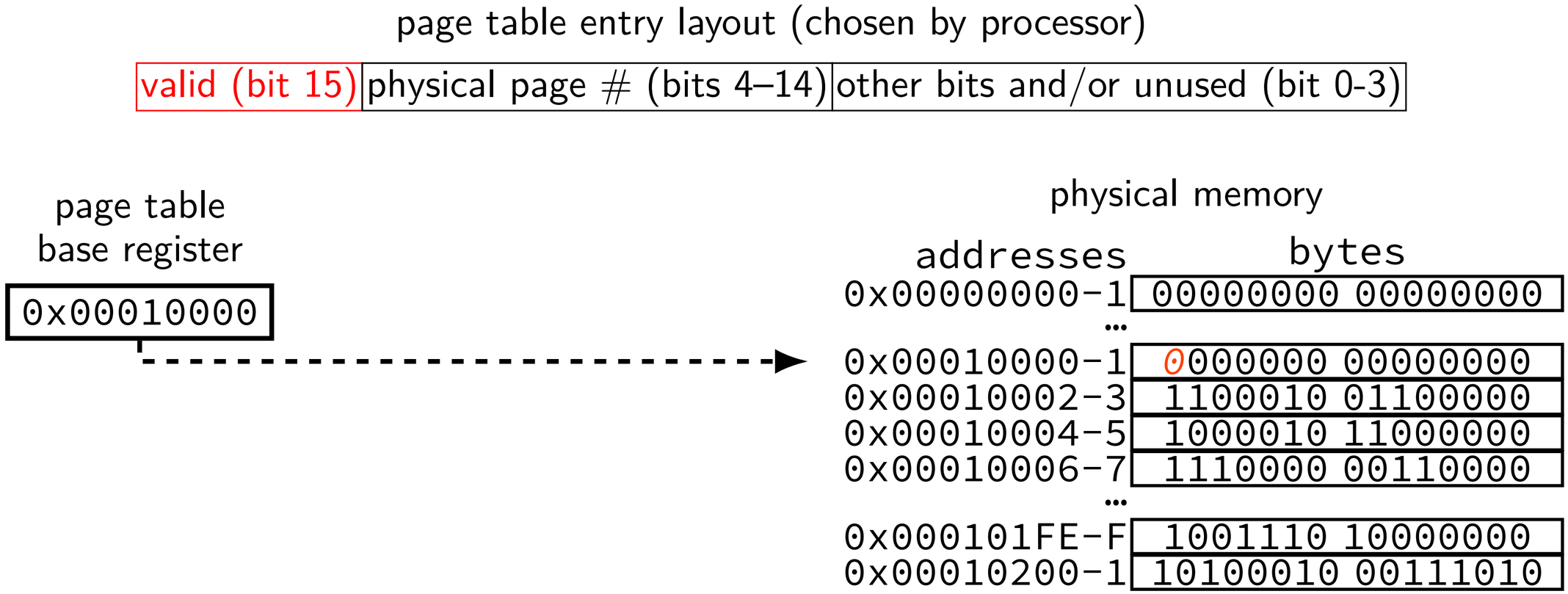
page tables in memory

where can processor store megabytes of page tables? *in memory*



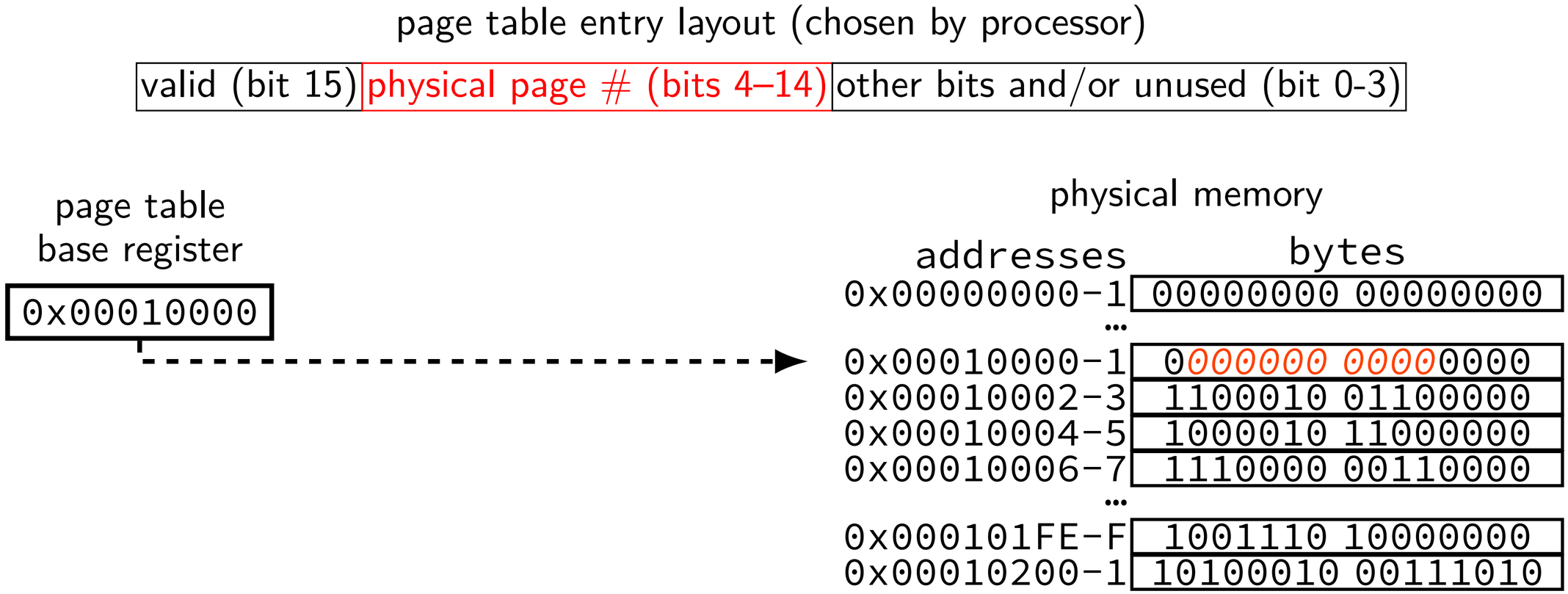
page tables in memory

where can processor store megabytes of page tables? *in memory*



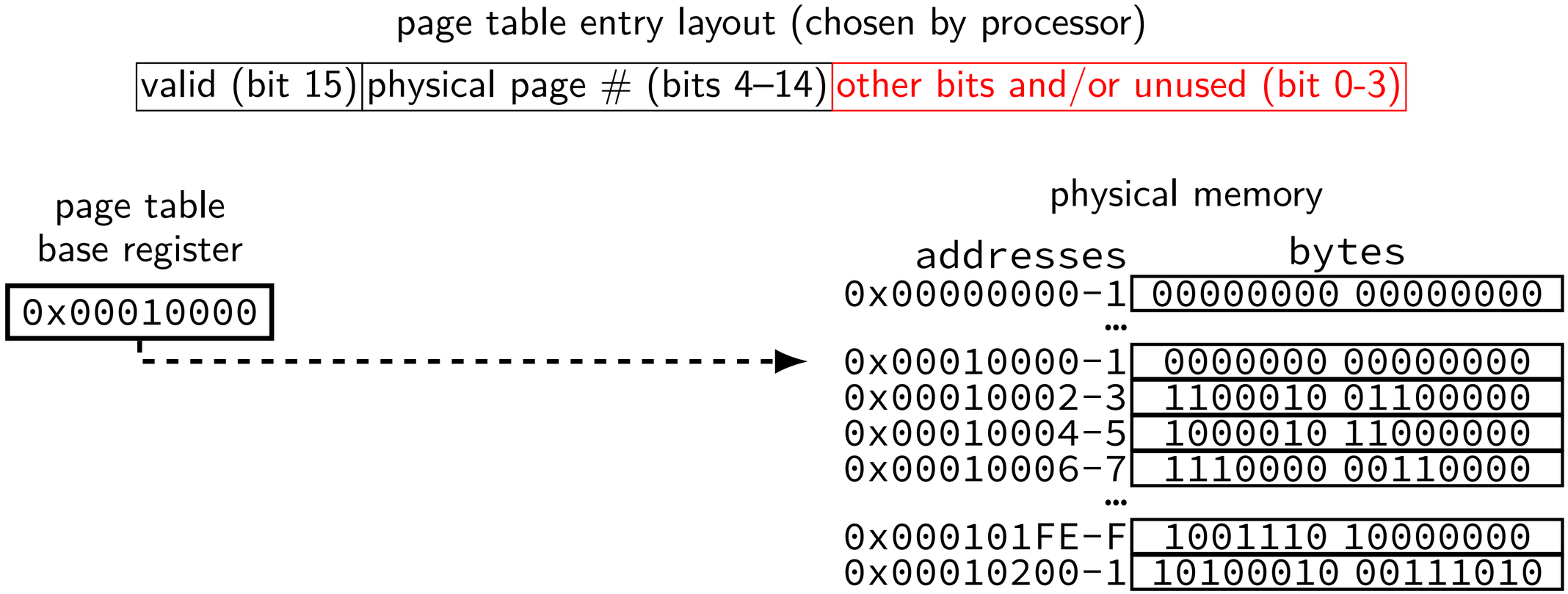
page tables in memory

where can processor store megabytes of page tables? *in memory*



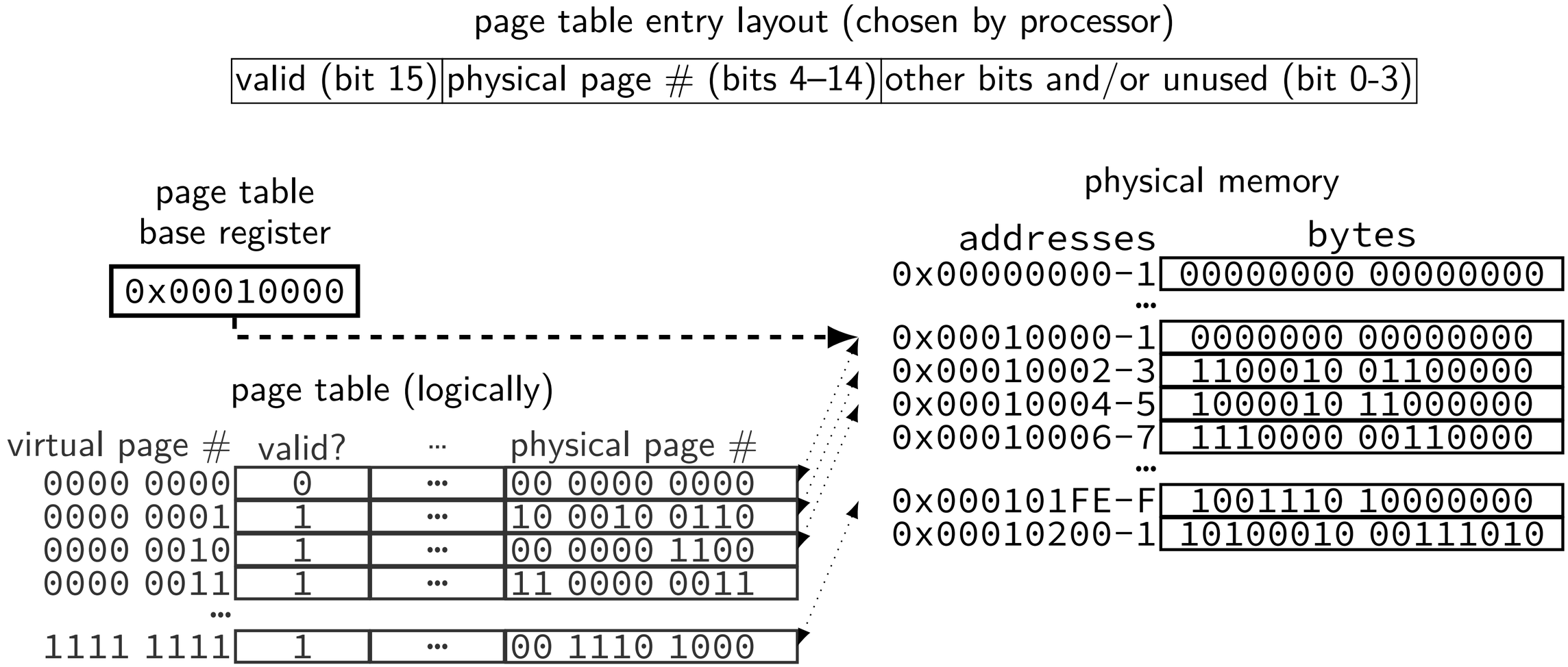
page tables in memory

where can processor store megabytes of page tables? *in memory*



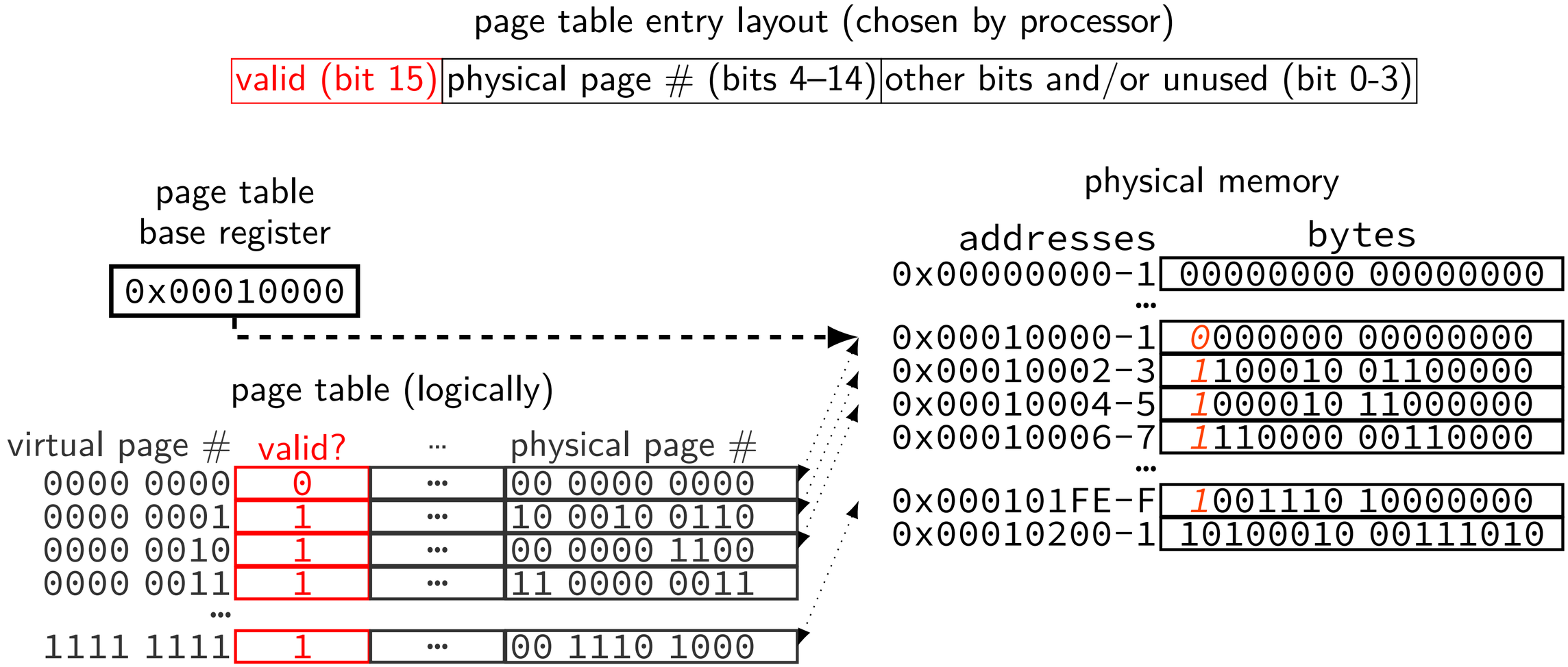
page tables in memory

where can processor store megabytes of page tables? *in memory*



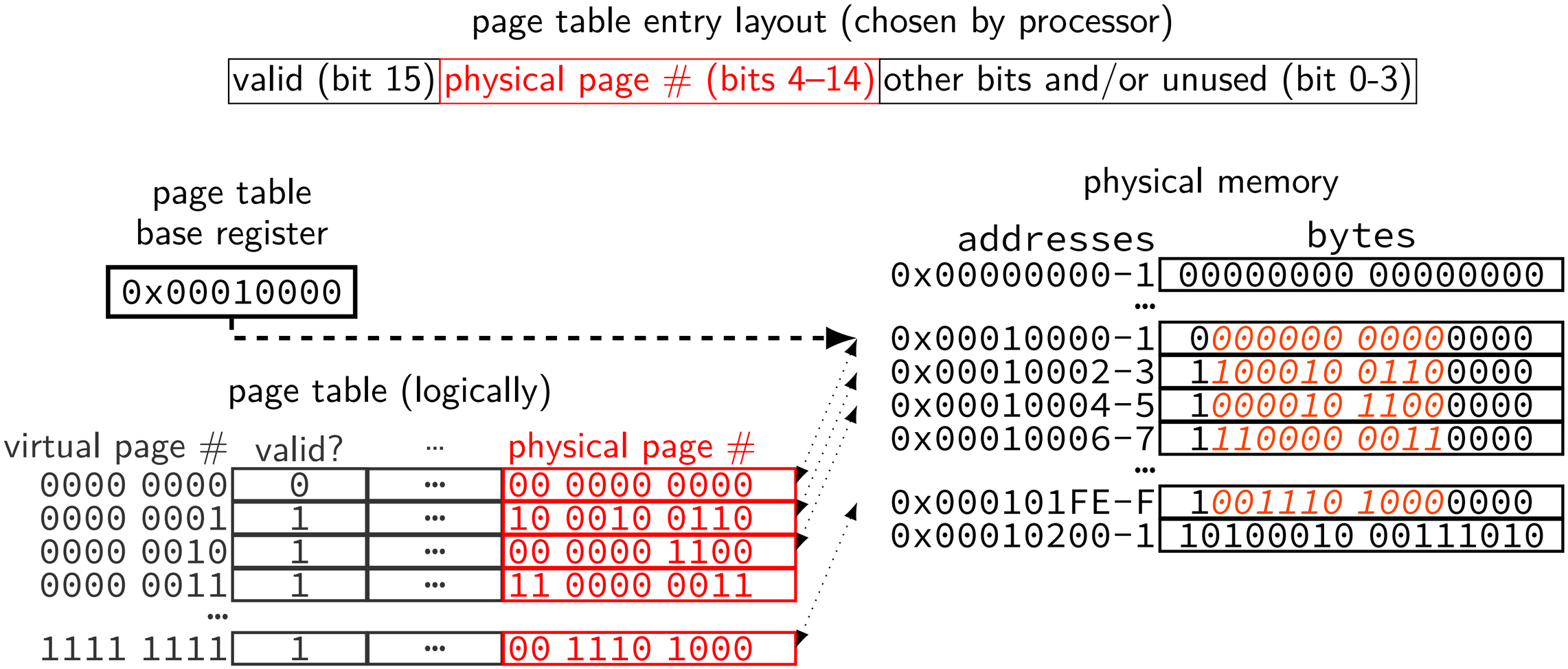
page tables in memory

where can processor store megabytes of page tables? *in memory*



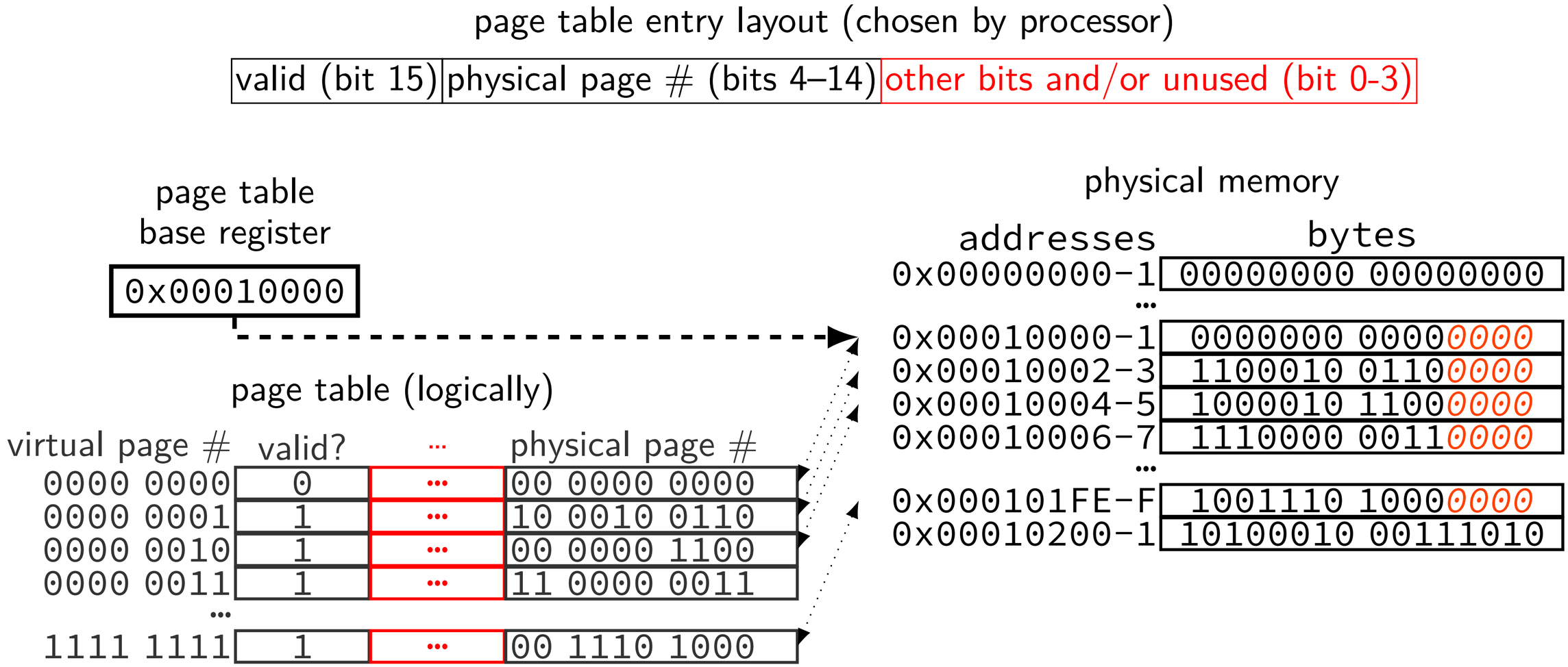
page tables in memory

where can processor store megabytes of page tables? *in memory*



page tables in memory

where can processor store megabytes of page tables? *in memory*

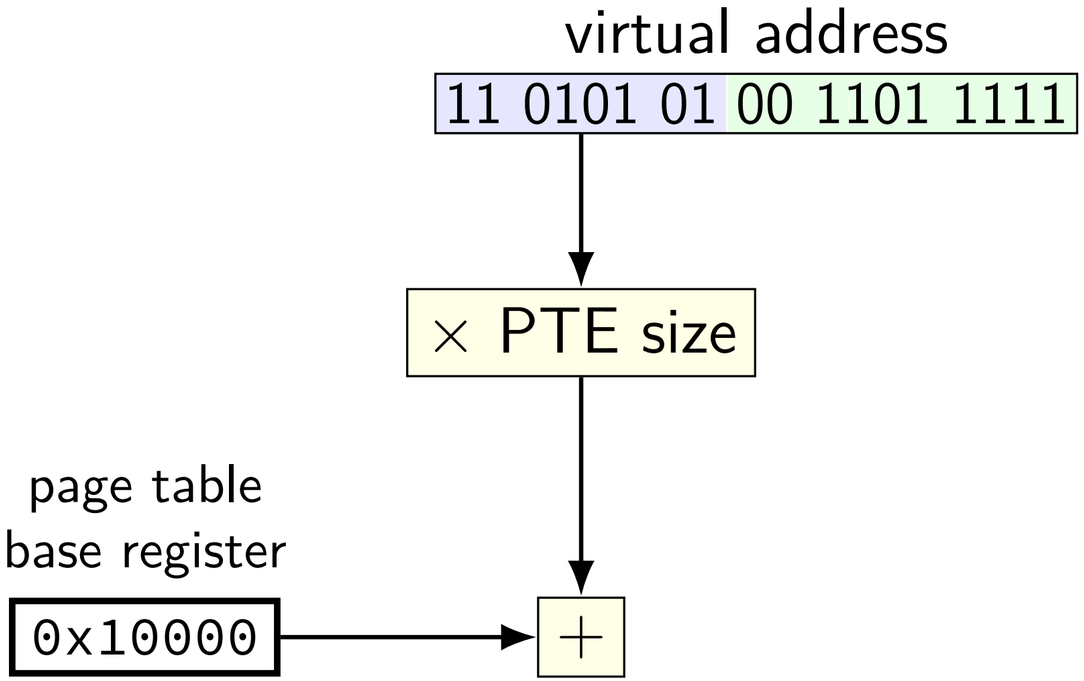


memory access with page table

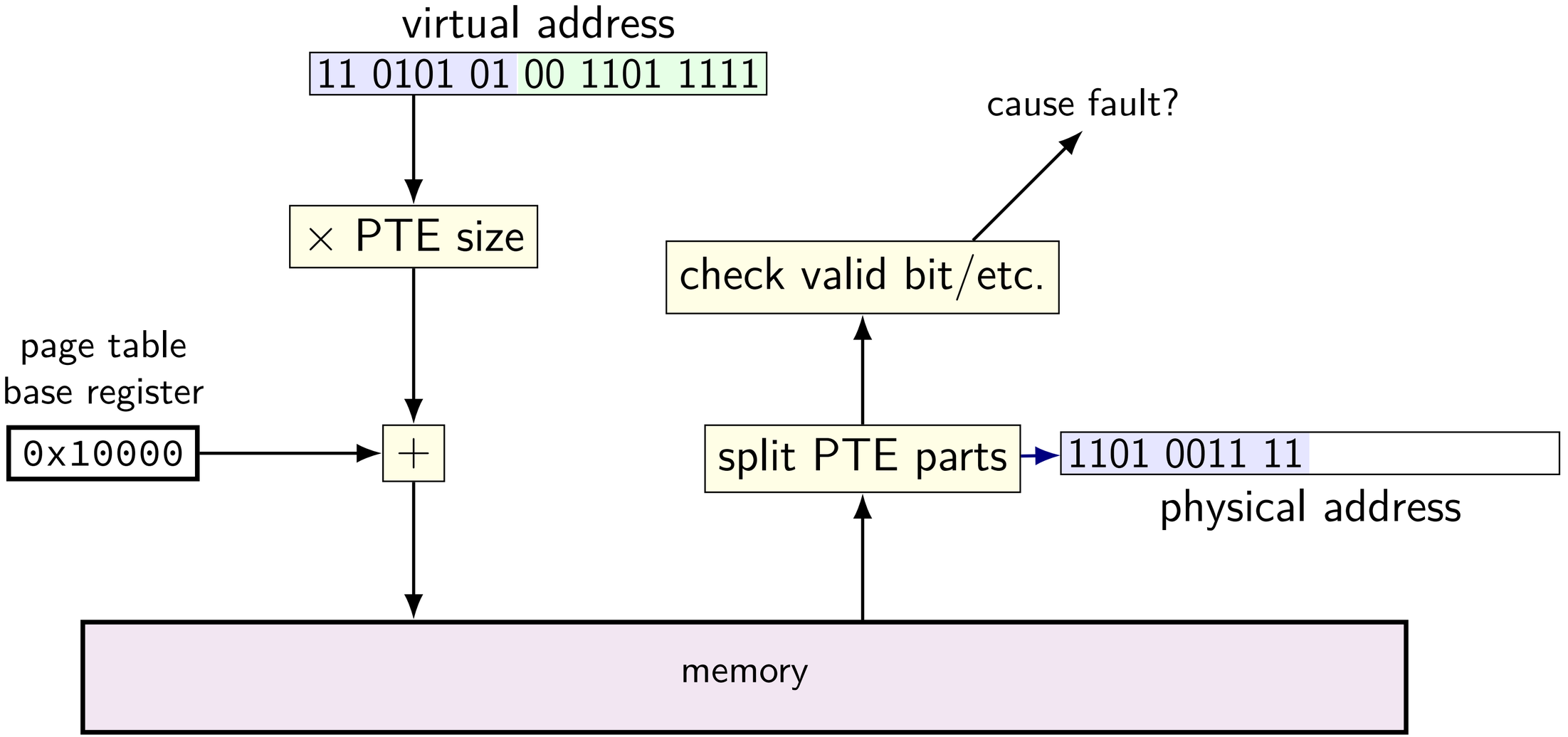
virtual address

11 0101 01	00 1101 1111

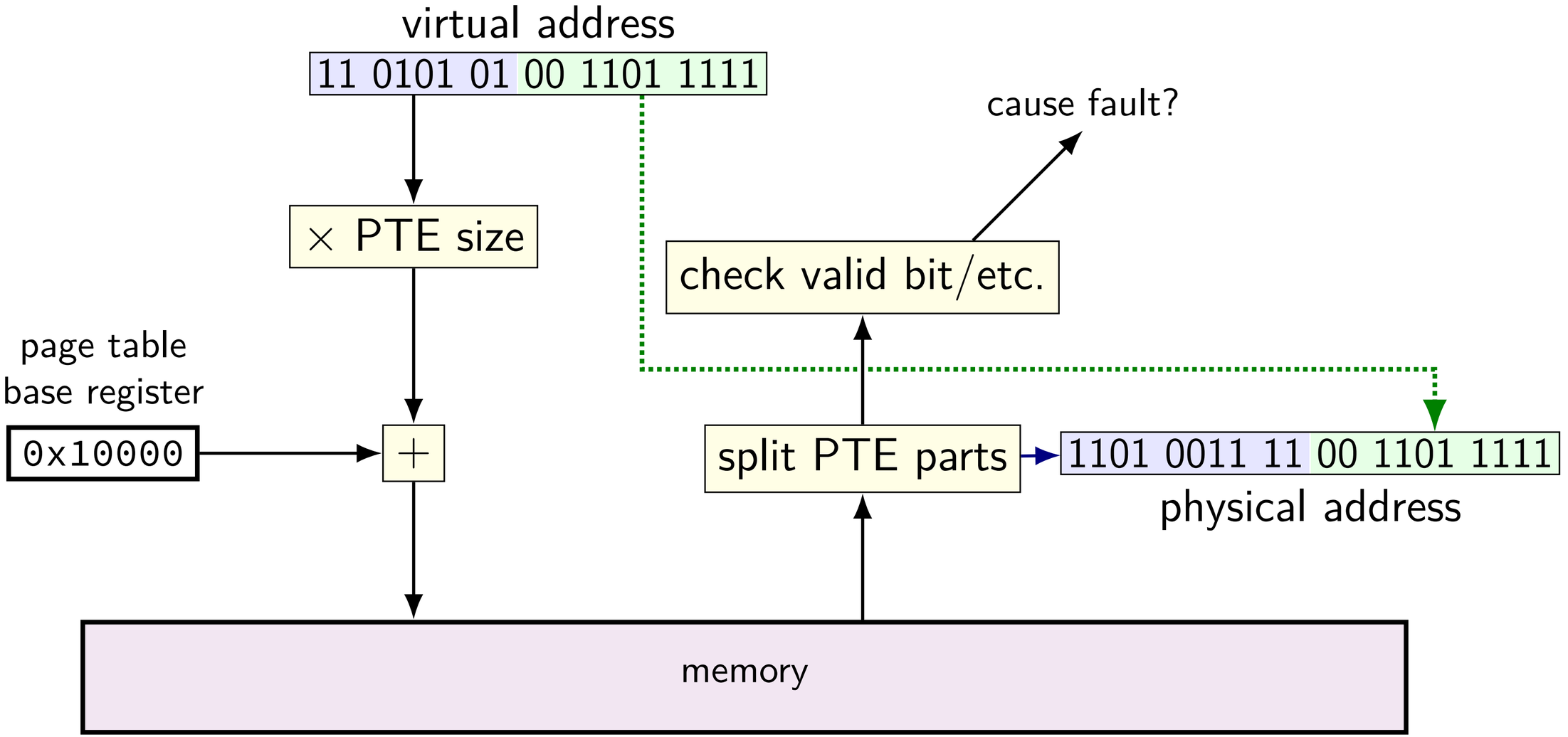
memory access with page table



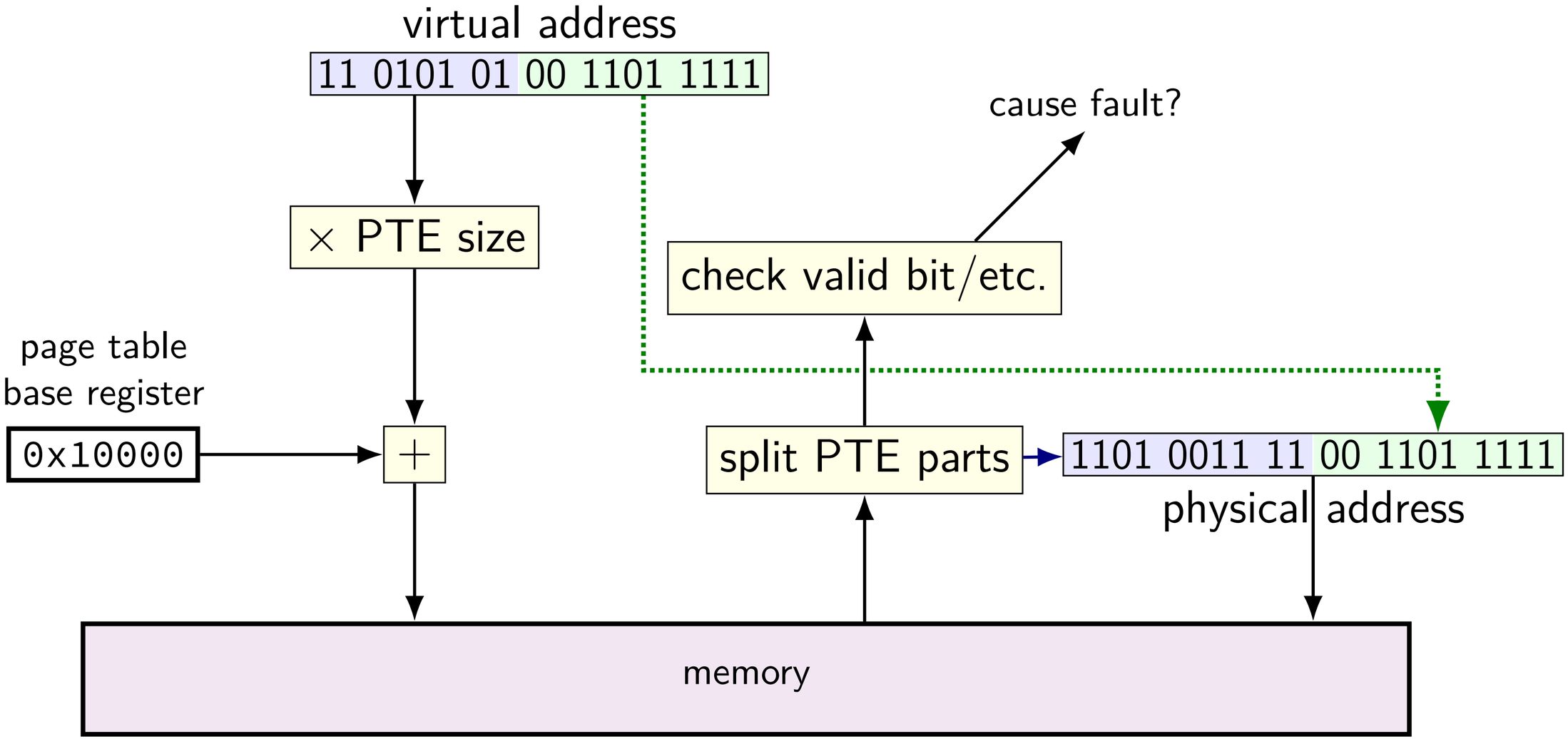
memory access with page table



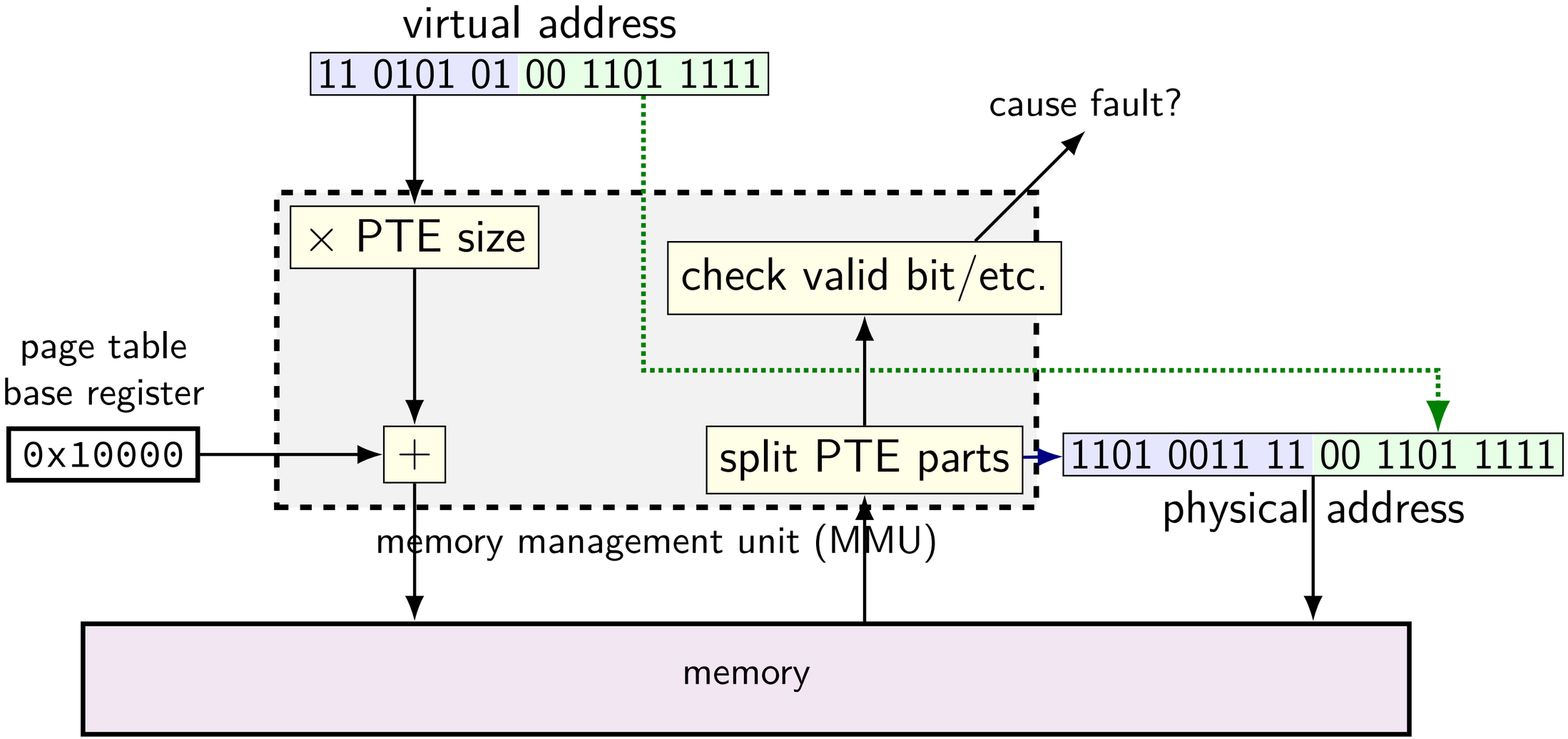
memory access with page table



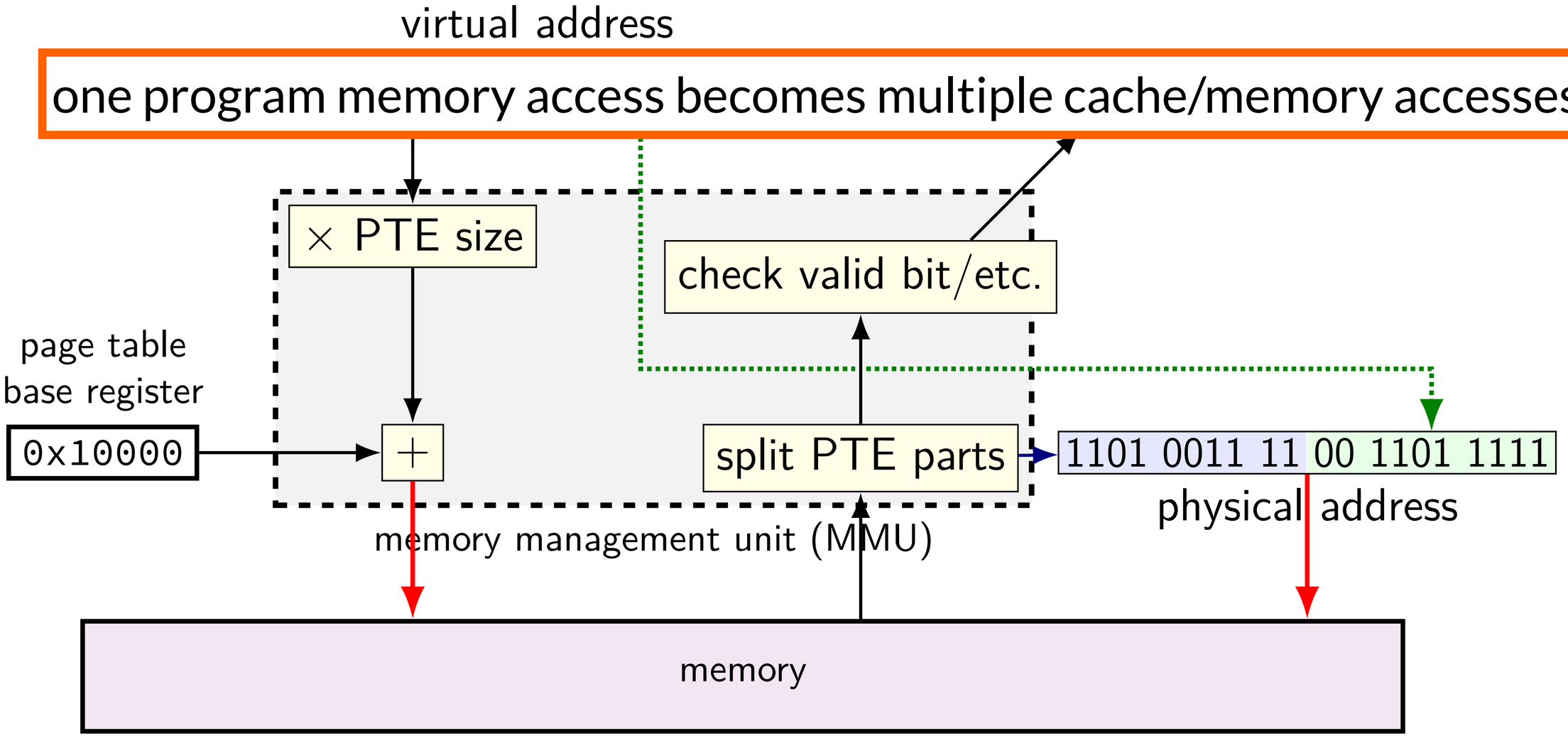
memory access with page table



memory access with page table



memory access with page table



1-level exercise (1)

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

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

1-level exercise (1)

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

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$0x31 = 11\ 0001_{TWO}$

PTE addr:

$0x20 + 110_{TWO} \times 1 = 0x26$

PTE value:

$0xF6 = 1111\ 0110_{TWO}$

PPN 111, valid 1

$M[111\ 001_{TWO}] = M[0x39]$

$\rightarrow$ 0x0C

1-level exercise (1)

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

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$0x31 = 11\ 0001_{TWO}$

PTE addr:

$0x20 + 110_{TWO} \times 1 = 0x26$

PTE value:

$0xF6 = 1111\ 0110_{TWO}$

PPN 111, valid 1

$M[111\ 001_{TWO}] = M[0x39]$

$\rightarrow$ 0x0C

1-level exercise (1)

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other;

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

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	*0x24–27*	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x31 = 11 0001_{TWO}

PTE addr:

0x20 + 110_{TWO} × 1 = *0x26*

PTE value:

0xF6 = 1111 0110_{TWO}

PPN 111, valid 1

M[111 001_{TWO}] = M[0x39]

→ 0x0C

1-level exercise (1)

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

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–27	E4 E5 *F6* 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$0x31 = 11\ 0001_{TWO}$

PTE addr:

$0x20 + 110_{TWO} \times 1 = 0x26$

PTE value:

$0xF6 = 1111\ 0110_{TWO}$

PPN 111, valid 1

$M[111\ 001_{TWO}] = M[0x39]$

$\rightarrow$ 0x0C

1-level exercise (1)

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

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$0x31 = 11\ 0001_{TWO}$

PTE addr:

$0x20 + 110_{TWO} \times 1 = 0x26$

PTE value:

$0xF6 = 1111\ 0110_{TWO}$

PPN 111, valid 1

$M[111\ 001_{TWO}] = M[0x39]$

$\rightarrow$ 0x0C

1-level exercise (1)

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

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$0x31 = 11\ 0001_{TWO}$

PTE addr:

$0x20 + 110_{TWO} \times 1 = 0x26$

PTE value:

$0xF6 = 1111\ 0110_{TWO}$

PPN 111, valid 1

$M[111\ 001_{TWO}] = M[0x39]$

$\rightarrow$ 0x0C

1-level exercise (1)

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

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$0x31 = 11\ 0001_{TWO}$

PTE addr:

$0x20 + 110_{TWO} \times 1 = 0x26$

PTE value:

$0xF6 = 1111\ 0110_{TWO}$

PPN 111, valid 1

$M[111\ 001_{TWO}] = M[0x39]$

$\rightarrow$ 0x0C

1-level exercise (1)

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

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	*0x38–3B*	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$0x31 = 11\ 0001_{TWO}$

PTE addr:

$0x20 + 110_{TWO} \times 1 = 0x26$

PTE value:

$0xF6 = 1111\ 0110_{TWO}$

PPN 111, valid 1

$M[111\ 001_{TWO}] = M[0x39]$

$\rightarrow$ 0x0C

1-level exercise (1)

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

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC *0C* DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$0x31 = 11\ 0001_{TWO}$

PTE addr:

$0x20 + 110_{TWO} \times 1 = 0x26$

PTE value:

$0xF6 = 1111\ 0110_{TWO}$

PPN 111, valid 1

$M[111\ 001_{TWO}] = M[0x39]$

$\rightarrow$ *0x0C*

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 E2 D1 F3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 E2 D1 F3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 01 0010_{TWO}
PTE addr:
0x20 + 010_{TWO} × 1 = 0x22
PTE value:
0xD1 = $1101\ 0001_{TWO}$
PPN 110, valid 1
M[$110\ 010_{TWO}$] = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 E2 D1 F3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = *01* *0*010$_{TWO}$
PTE addr:
0x20 + *010*$_{TWO}$ × 1 = 0x22
PTE value:
0xD1 = 1101 0001$_{TWO}$
PPN 110, valid 1
M[110 010$_{TWO}$] = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	*0x20–23*	A0 E2 D1 F3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$0x12 = 01\ 0010_{TWO}$

PTE addr:

$0x20 + 010_{TWO} \times 1 = 0x22$

PTE value:

$0xD1 = 1101\ 0001_{TWO}$

PPN 110, valid 1

$M[110\ 010_{TWO}] = M[0x32]$

$\rightarrow$ 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 E2 *D1* F3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 01 0010_{TWO}
PTE addr:
0x20 + 010_{TWO} × 1 = 0x22
PTE value:
0xD1 = 1101 0001_{TWO}
PPN 110, valid 1
M[110 010_{TWO}] = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 E2 D1 F3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 01 0010_{TWO}
PTE addr:
0x20 + 010_{TWO} × 1 = 0x22
PTE value:
0xD1 = $1101\ 0001_{TWO}$
PPN 110, valid 1
M[$110\ 010_{TWO}$] = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 E2 D1 F3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 01 0010_{TWO}
PTE addr:
0x20 + 010_{TWO} × 1 = 0x22
PTE value:
0xD1 = $1101\ 0001_{TWO}$
PPN 110, valid 1
M[$110\ 010_{TWO}$] = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 E2 D1 F3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 01 0010_{TWO}
PTE addr:
0x20 + 010_{TWO} × 1 = 0x22
PTE value:
0xD1 = $1101\ 0001_{TWO}$
PPN 110, valid 1
M[$110\ 010_{TWO}$] = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 E2 D1 F3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	*0x30–33*	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 01 0010_{TWO}
PTE addr:
0x20 + 010_{TWO} × 1 = 0x22
PTE value:
0xD1 = $1101\ 0001_{TWO}$
PPN 110, valid 1
M[$110\ 010_{TWO}$] = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 E2 D1 F3
0x04–07	44 55 66 77	0x24–27	E4 E5 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A *BA* 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 01 0010_{TWO}
PTE addr:
0x20 + 010_{TWO} × 1 = 0x22
PTE value:
0xD1 = $1101\ 0001_{TWO}$
PPN 110, valid 1
M[$110\ 010_{TWO}$] = M[0x32]
→ *0xBA*

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, 2 byte PTE
page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;
page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	0x24–27	70 00 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, *2 byte PTE*

page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;

page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	0x24–27	70 00 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, 2 byte PTE
page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;
page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	0x24–27	70 00 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 1 0010_{TWO}
PTE addr:
0x20 + 10_{TWO} × 2 = 0x24
PTE value: 70 00
PPN 011, valid 1
M[$011\ 010_{TWO}$] = M[0x1A]
→ 0x3C

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, 2 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;

page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	0x24–27	70 00 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = *1* *00*10$_{\text{TWO}}$

PTE addr:

0x20 + *10*$_{\text{TWO}}$ × 2 = 0x24

PTE value: 70 00

PPN 011, valid 1

M[011 010$_{\text{TWO}}$] = M[0x1A]

→ 0x3C

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, 2 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;

page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	*0x24–27*	70 00 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 1 0010_{TWO}

PTE addr:

0x20 + 10_{TWO} × 2 = *0x24*

PTE value: 70 00

PPN 011, valid 1

M[$011\ 010_{TWO}$] = M[0x1A]

→ 0x3C

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, 2 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;

page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	0x24–27	*70 00* F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 1 0010_{TWO}

PTE addr:

0x20 + 10_{TWO} × 2 = 0x24

PTE value: *70 00*

PPN 011, valid 1

M[$011\ 010_{TWO}$] = M[0x1A]

→ 0x3C

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, 2 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;

page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	0x24–27	70 00 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$0x12 = 1\ 0010_{TWO}$

PTE addr:

$0x20 + 10_{TWO} \times 2 = 0x24$

PTE value: 70 00

PPN ***011***, valid 1

$M[011\ 010_{TWO}] = M[0x1A]$

$\rightarrow$ 0x3C

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, 2 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;

page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	0x24–27	70 00 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 1 0010_{TWO}

PTE addr:

0x20 + 10_{TWO} × 2 = 0x24

PTE value: 70 00

PPN 011, valid 1

M[$011\ 010_{TWO}$] = M[0x1A]

→ 0x3C

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, 2 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;

page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	0x24–27	70 00 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 1 0010_{TWO}

PTE addr:

0x20 + 10_{TWO} × 2 = 0x24

PTE value: 70 00

PPN 011, valid 1

M[011 010_{TWO}] = M[0x1A]

→ 0x3C

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, 2 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;

page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	0x24–27	70 00 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 1 0010_{TWO}

PTE addr:

0x20 + 10_{TWO} × 2 = 0x24

PTE value: 70 00

PPN 011, valid 1

M[$011\ 010_{TWO}$] = M[*0x1A*]

→ 0x3C

1-level exercise (3)

5-bit virtual addresses, 6-bit physical; 8 byte pages, 2 byte PTE
page tables 1 page; PTE: 3 bit PPN (MSB of first byte), 1 valid bit, 12 other;
page table base register `0x20`; translate virtual address `0x12`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	A0 00 D1 00
0x04–07	44 55 66 77	0x24–27	70 00 F6 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C *3C* 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x12 = 1 0010_{TWO}
PTE addr:
0x20 + 10_{TWO} × 2 = 0x24
PTE value: 70 00
PPN 011, valid 1
M[$011\ 010_{TWO}$] = M[0x1A]
→ *0x3C*

pagetable assignment

pagetable assignment

simulate page tables (on top of normal program memory)

> alternately: implement another layer of page tables
> on top of the existing system's

in assignment:

virtual address $\sim$ arguments to your functions

physical address $\sim$ your program addresses (normal pointers)

pagetable assignment API

```
/* configuration parameters */
#define POBITS ... /* page offset bits */
#define LEVELS /* later */
```

```
size_t ptbr; // page table base register
    // points to page table (array of page table entries)

// lookup "virtual" address 'va' in page table ptbr points to
// return (~0L) if invalid
size_t translate(size_t va);

// allocating virtaul page which starts at address 'va'
// if it isn't already
int allocate_page(size_t va);
```

translate()

with POBITS=12, LEVELS=1:

ptbr = GetPointerToTable(

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

)

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:

ptbr = GetPointerToTable(

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

)

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*

allocate_page()

with POBITS=12, LEVELS=1:

ptbr $==$ 0

allocate_page(0x1000)

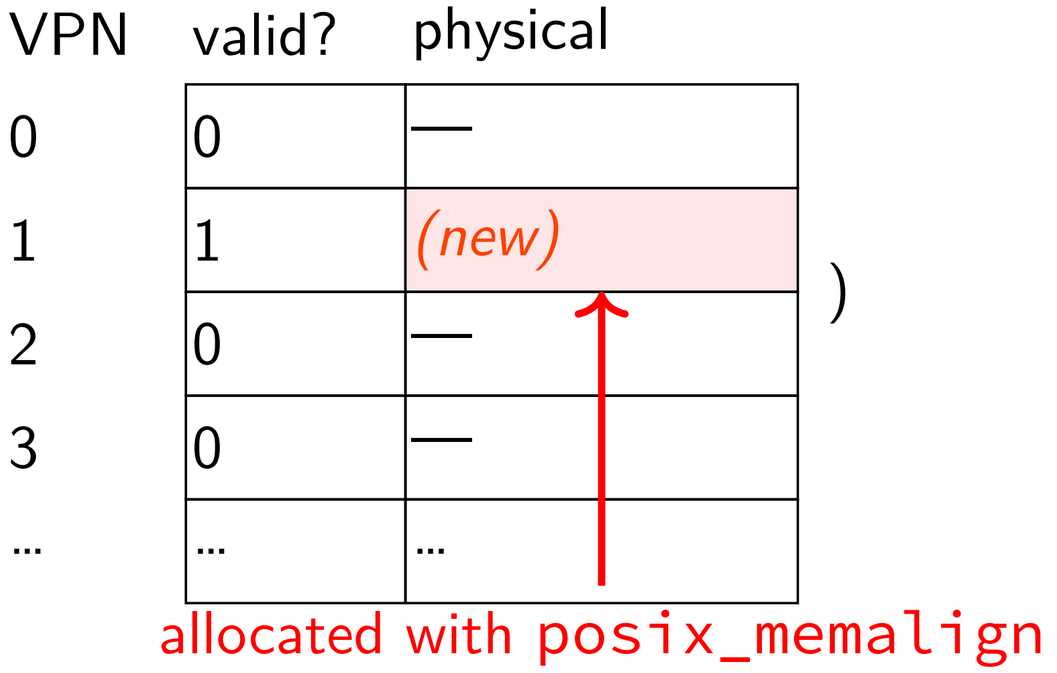
allocate_page()

with POBITS=12, LEVELS=1:

ptbr $==$ 0

allocate_page(0x1000)

ptbr *now* $==$ GetPointerToTable(



)

allocate_page()

with POBITS=*12*, LEVELS=1:

ptbr $==$ 0

allocate_page(0x1*000*)

ptbr *now* $==$ GetPointerToTable(

VPN	valid?	physical
0	0	—
1	1	(new)
2	0	—
3	0	—
...	...	...

)

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: LEVELS=1, POBITS=12 and

translate() OR

allocate_page()

part 2: all LEVELS, both functions

in preparation for code review

due Weds BEFORE LAB

part 3: final submission

Friday after code review

most of grade based on this

will test previous parts again

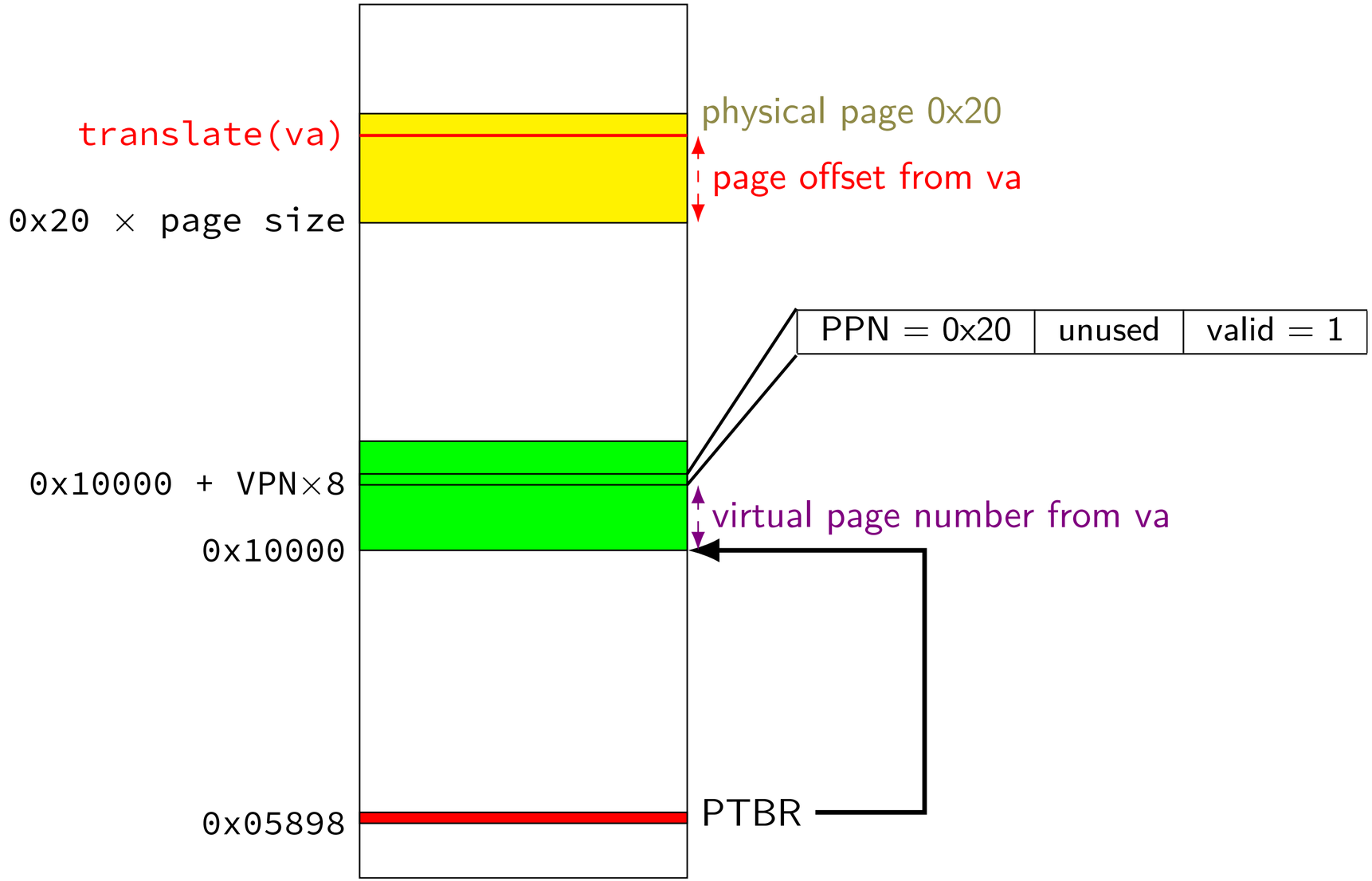
address/page table entry format

(with POBITS=12, LEVELS=1)

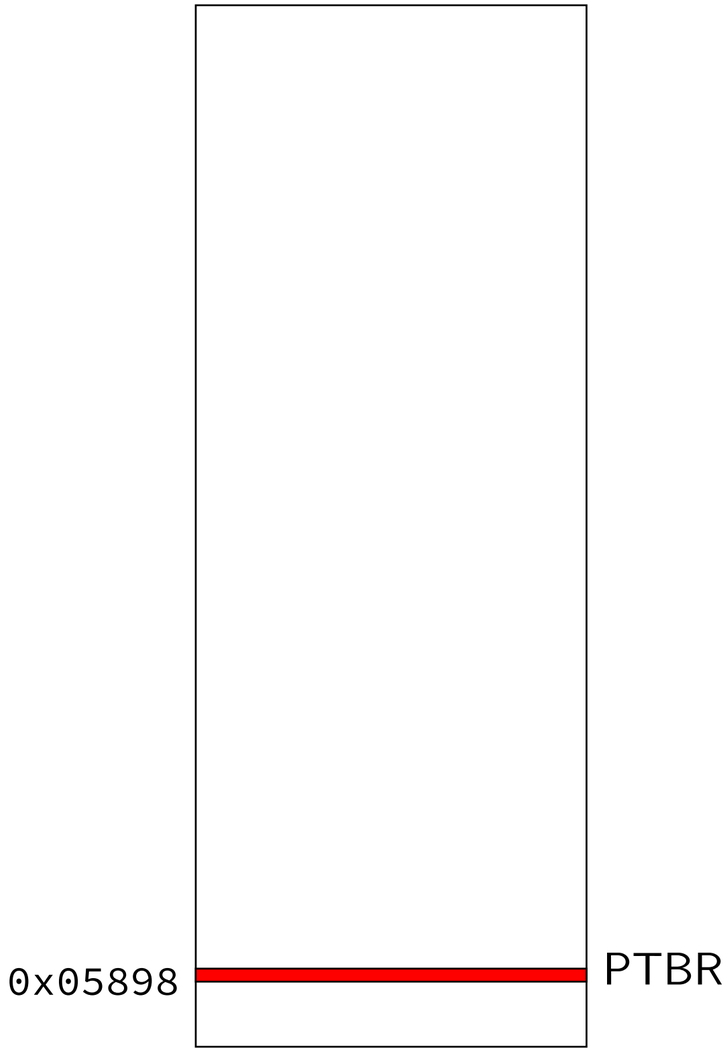
	bits 63–21	bits 20–12	bits 11–1	bit 0
page table entry	physical page number		unused	valid bit
virtual address	unused	virtual page number	page offset	
physical address	physical page number		page offset	

in assignment: value from posix_memalign = physical address

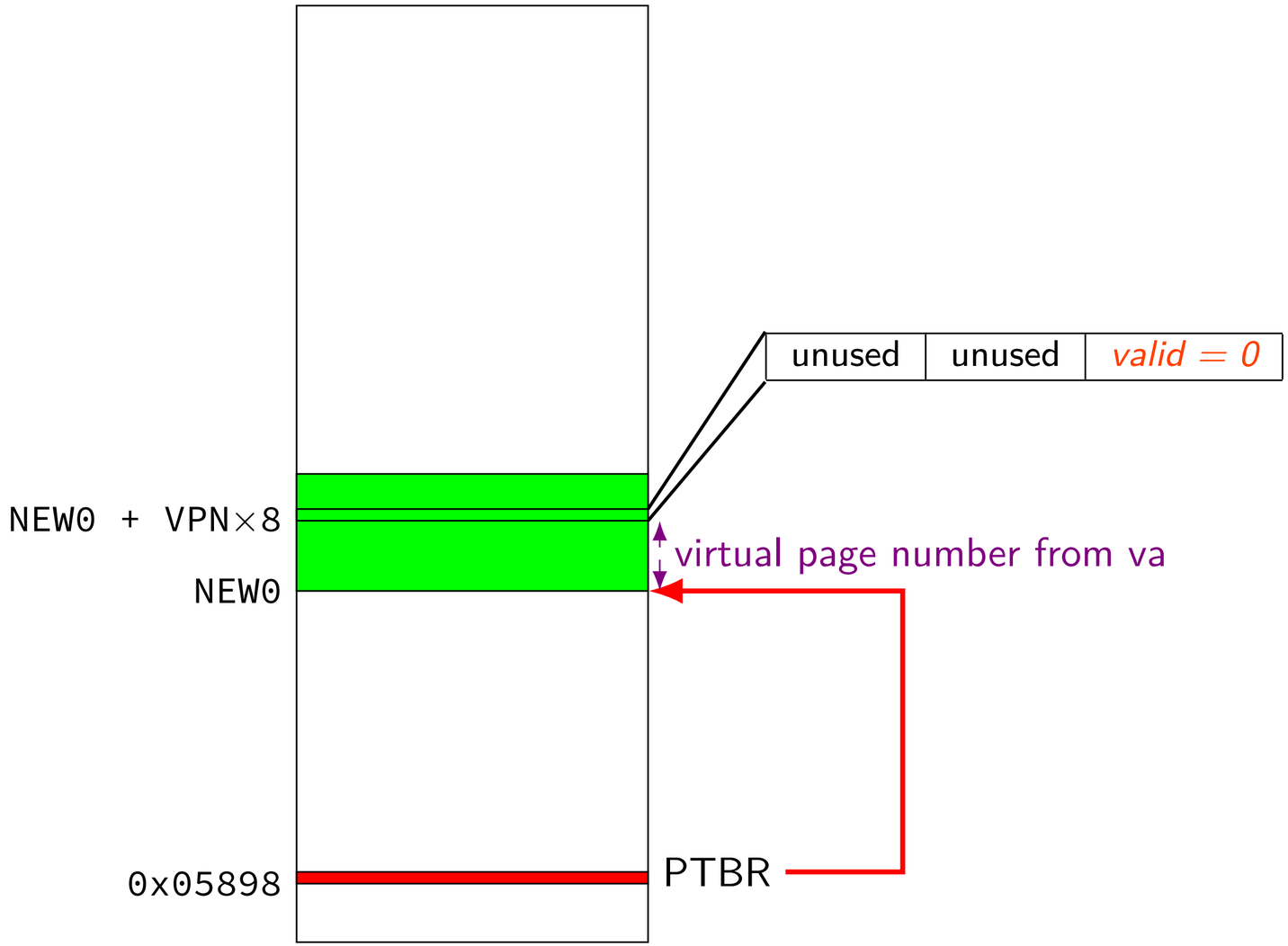
pa = translate(va) [LEVELS=1]



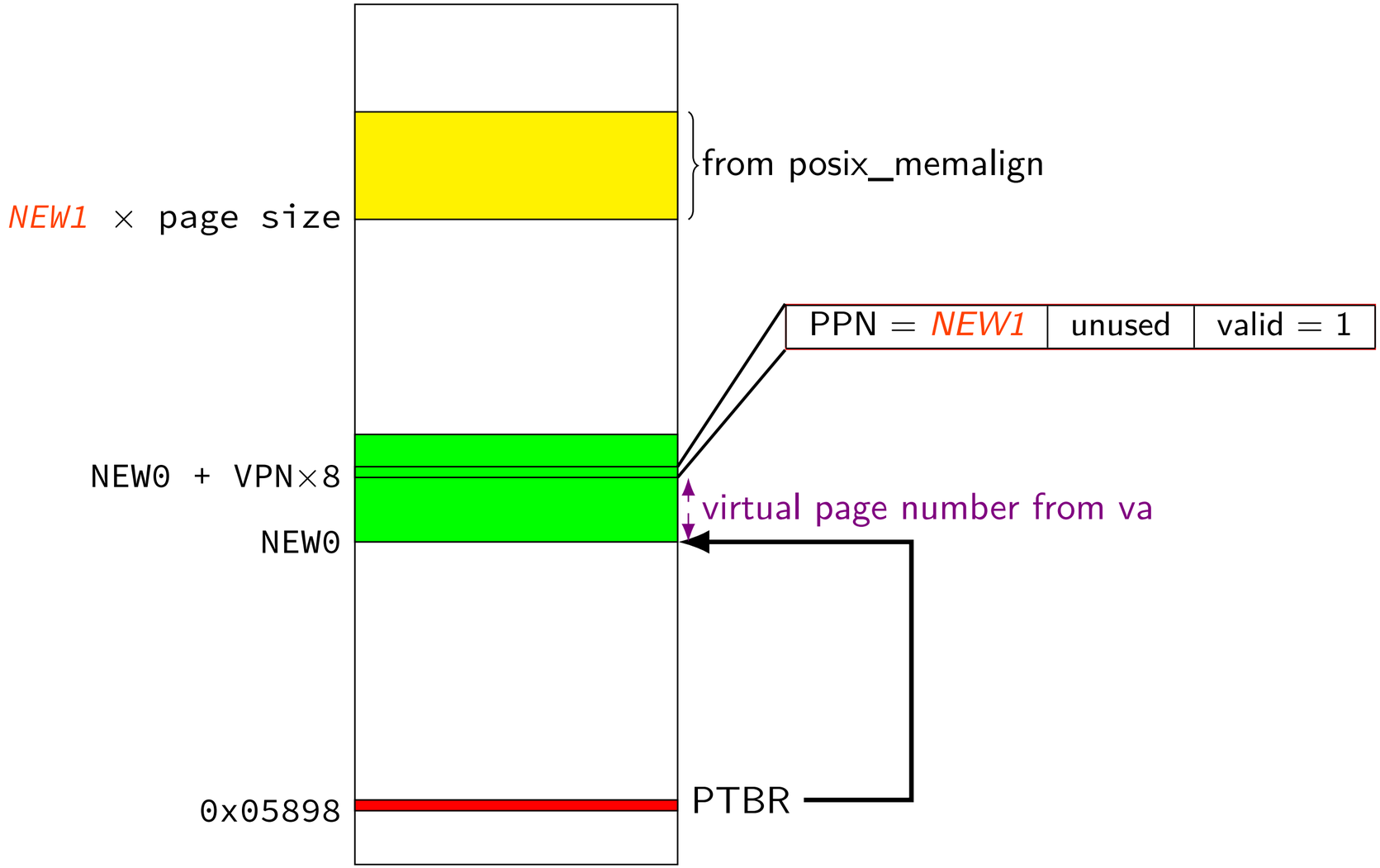
first allocate_page(va) [LEVELS=1]



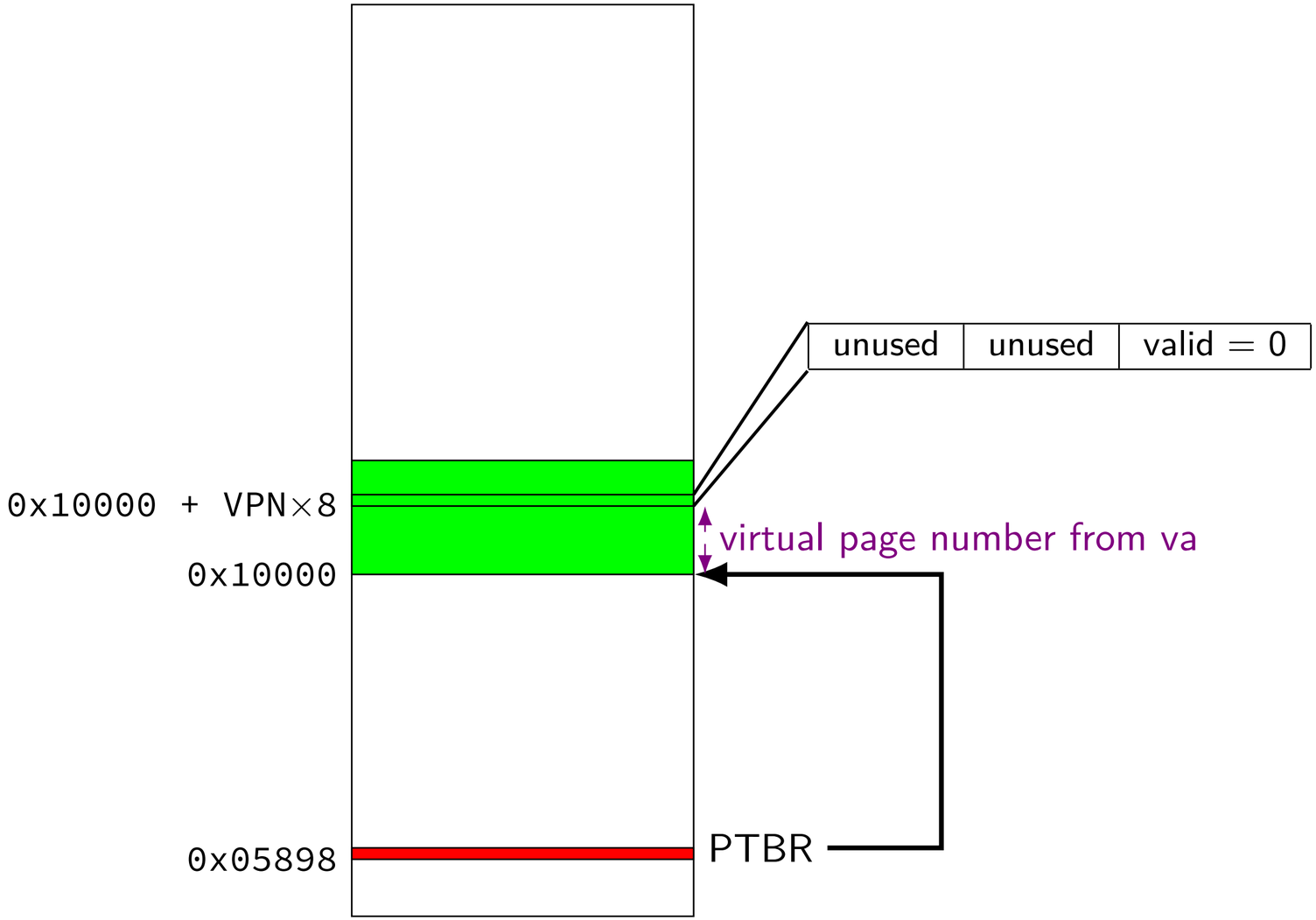
first allocate_page(va) [LEVELS=1]



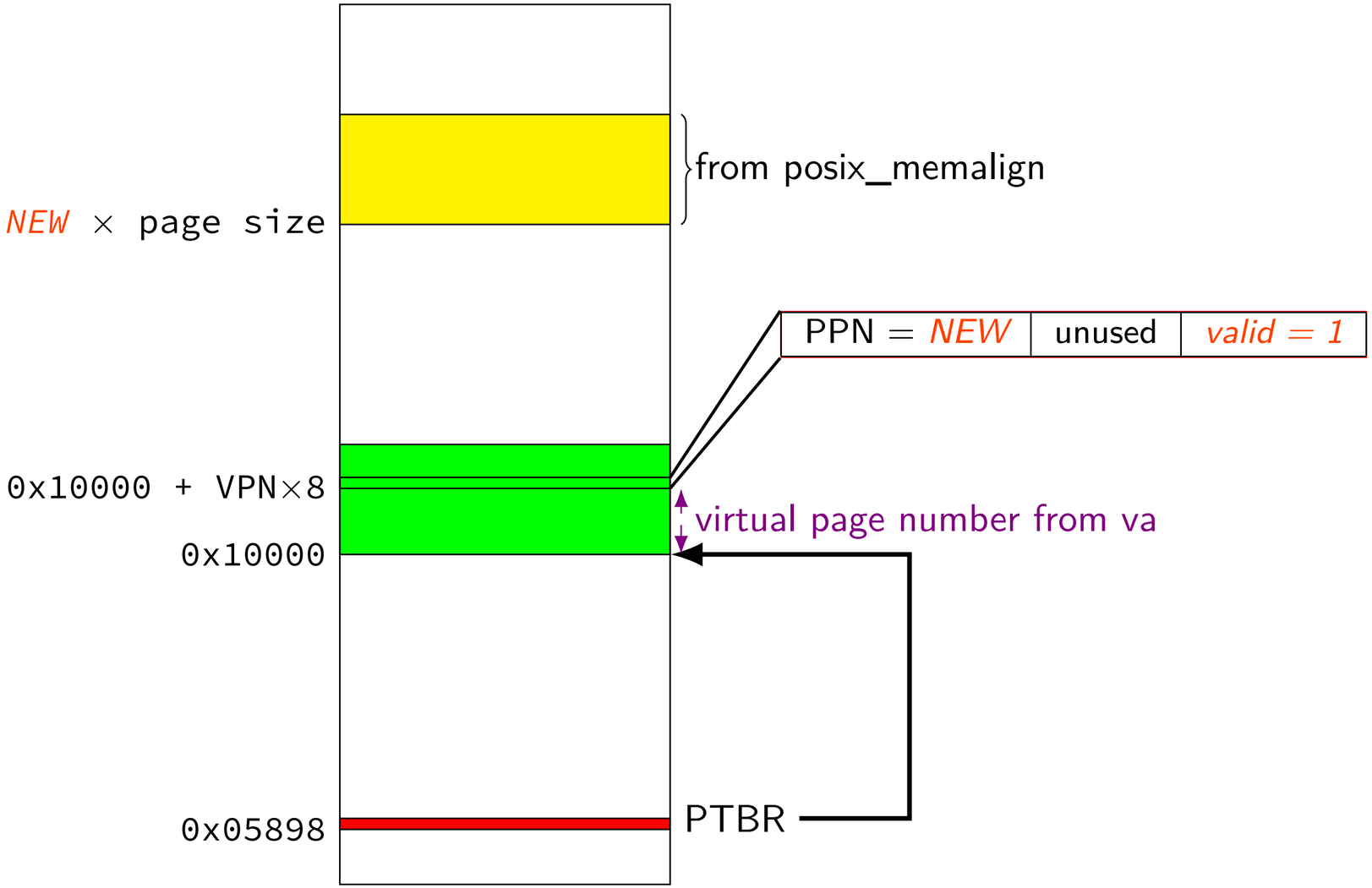
first allocate_page(va) [LEVELS=1]



allocate_page(va) [LEVELS=1]

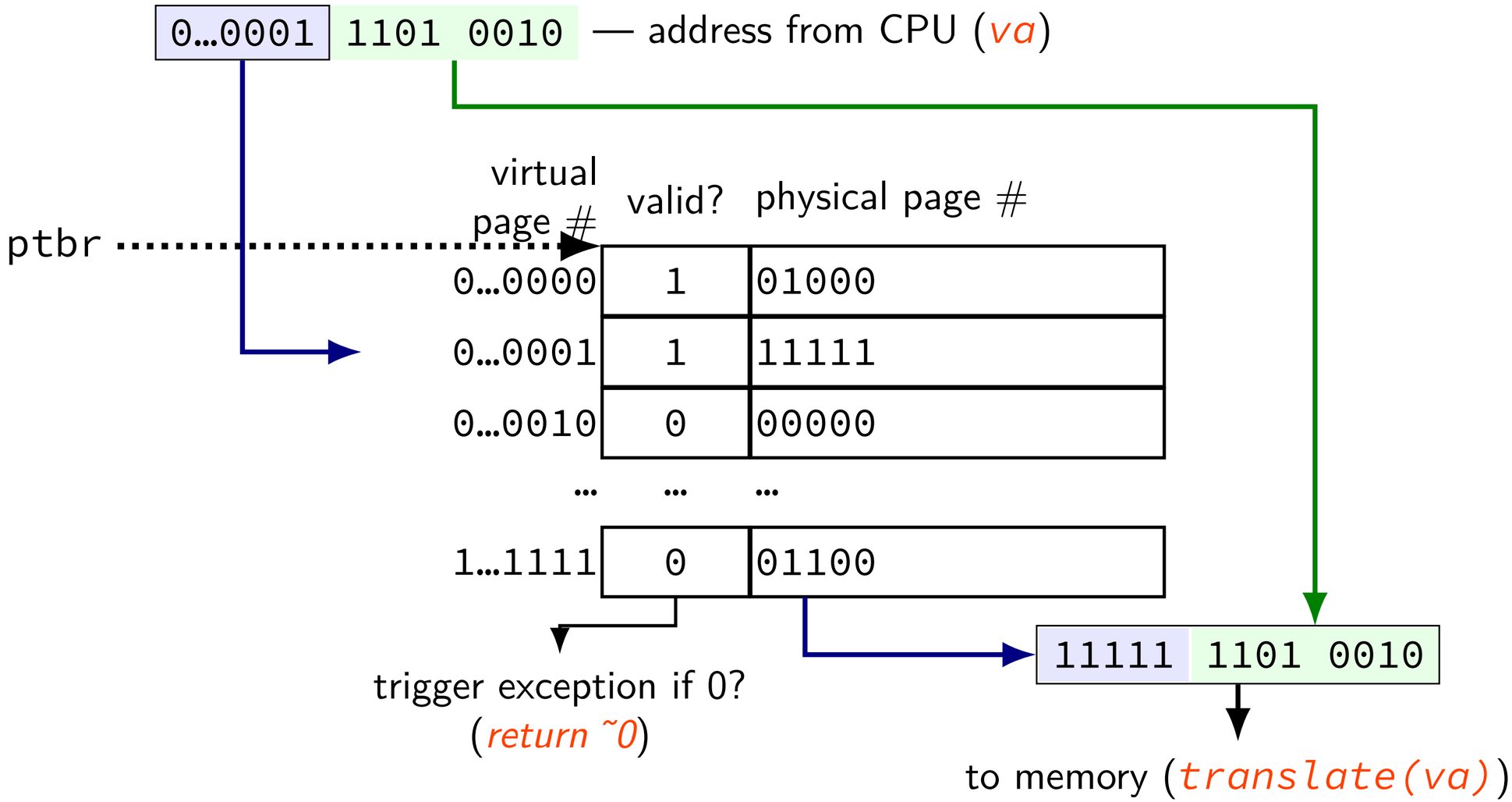


allocate_page(va) [LEVELS=1]

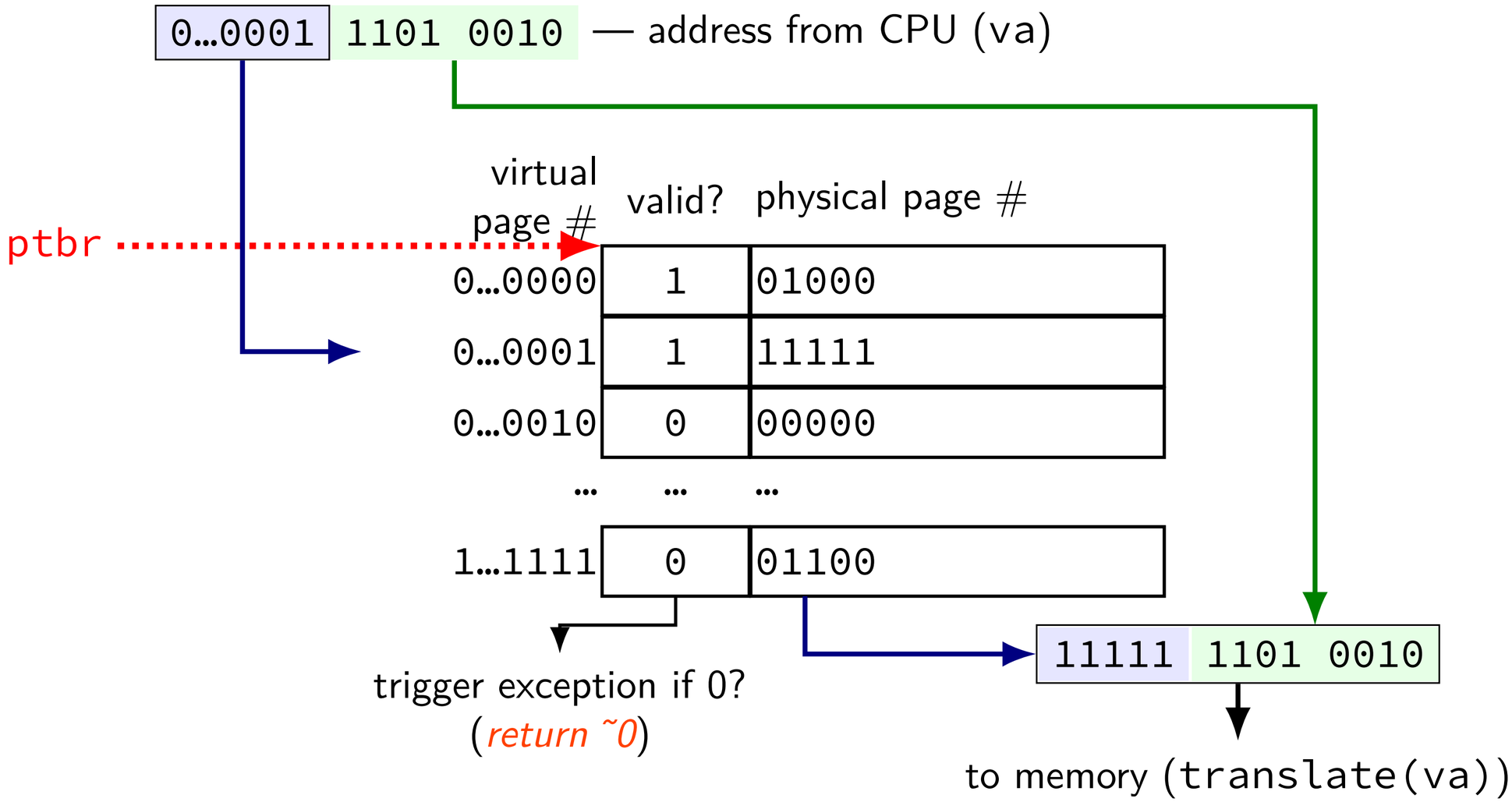


page table lookup (and translate())

page table lookup (and translate())

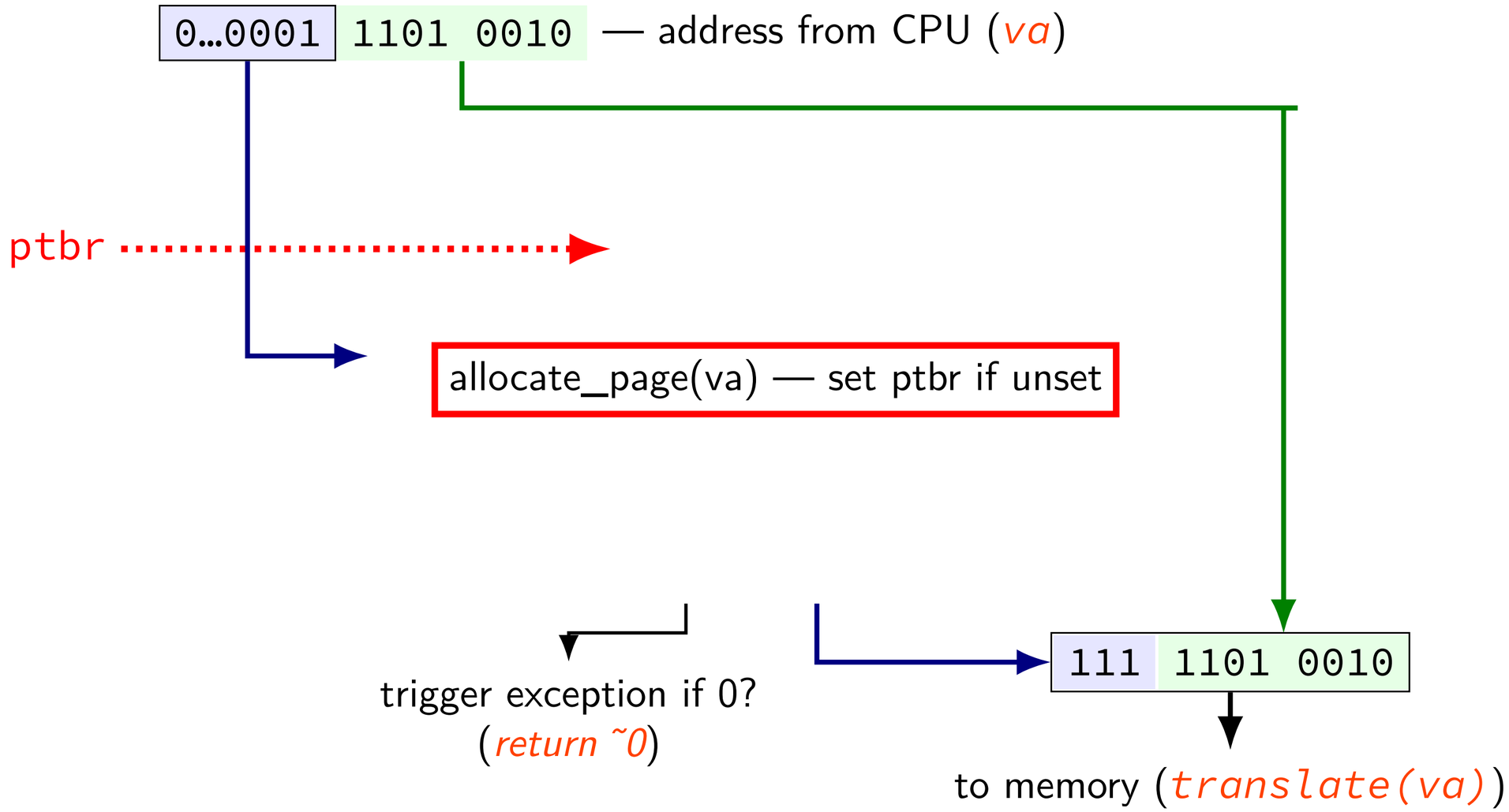


page table lookup (and translate())

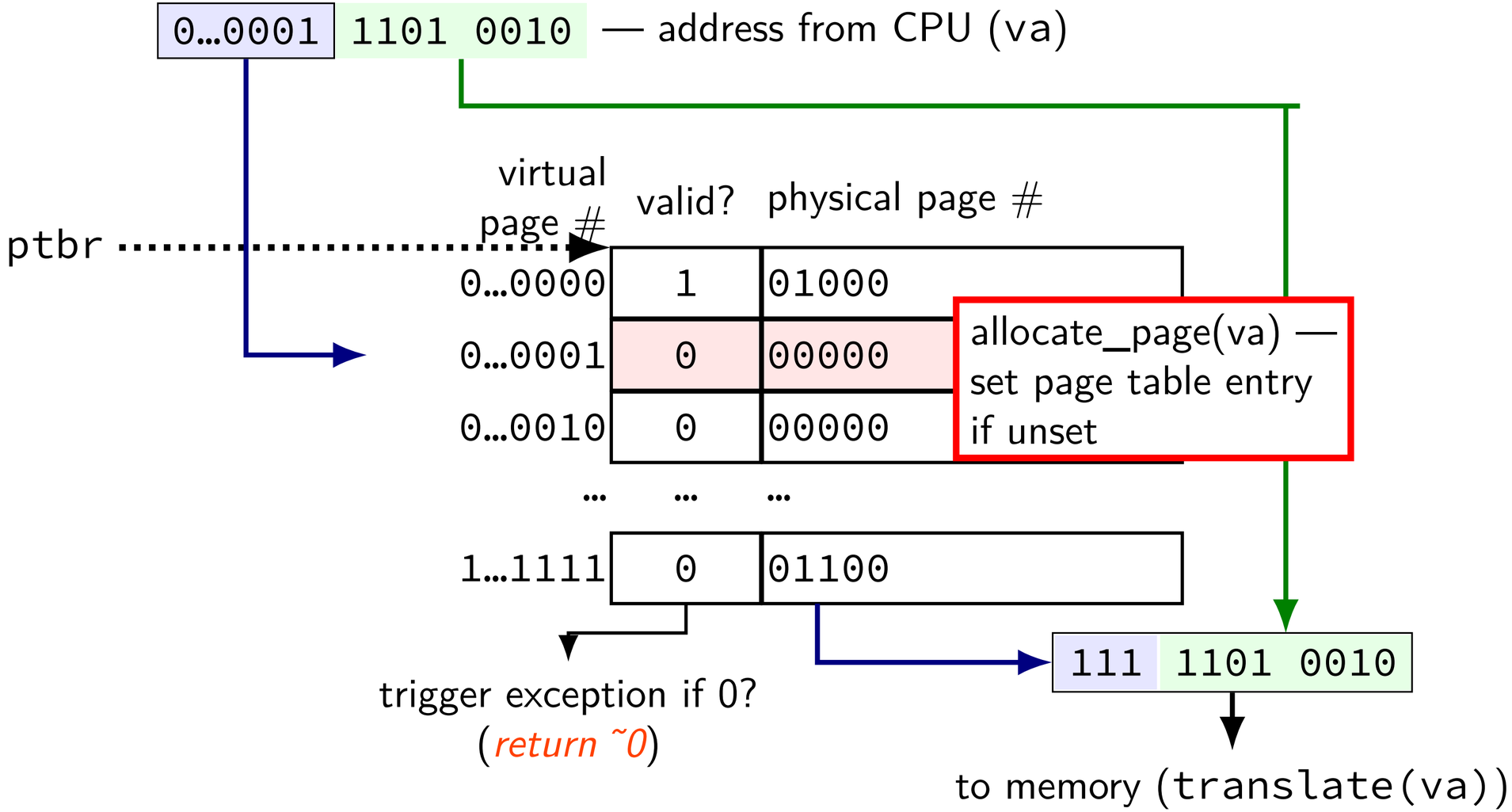


page table lookup (and allocate)

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: 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

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? $39 - 12 = 27$ bits

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

trick: power of two size makes table lookup faster

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? $39 - 12 = 27$ bits

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

trick: power of two size makes table lookup faster

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

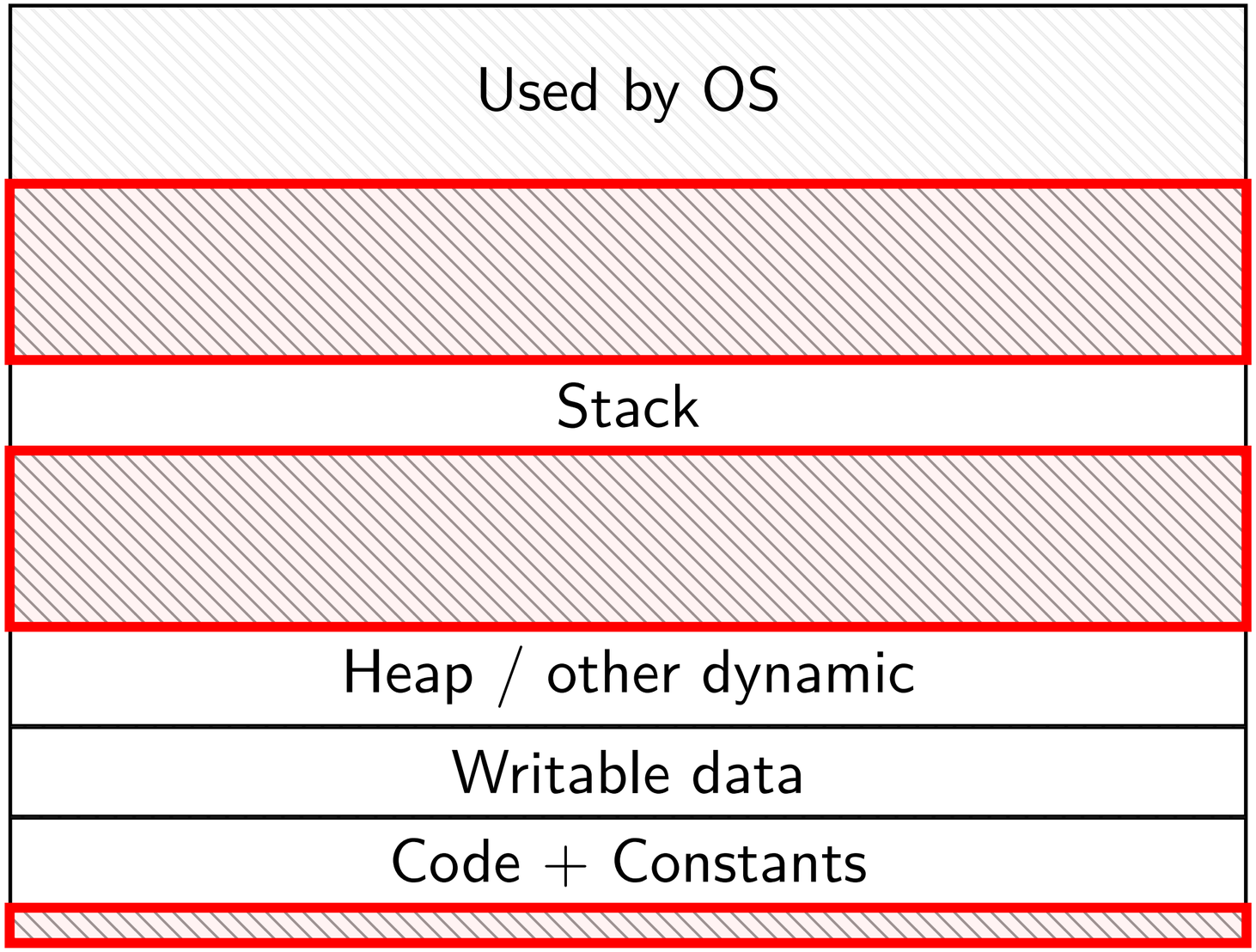
huge page tables

huge virtual address spaces!

impossible to store PTE for every page

how can we save space?

holes



most pages are *invalid*

saving space

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

use a data structure other than a flat array

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

options?

hashtable

- actually used by some historical processors
- but never common

tree data structure

- but not quite a search tree

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

saving space

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

use a data structure other than a flat array

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

options?

hashtable

- actually used by some historical processors
- but never common

tree data structure

- but not quite a search tree

search tree tradeoffs

lookup usually implemented *in hardware*

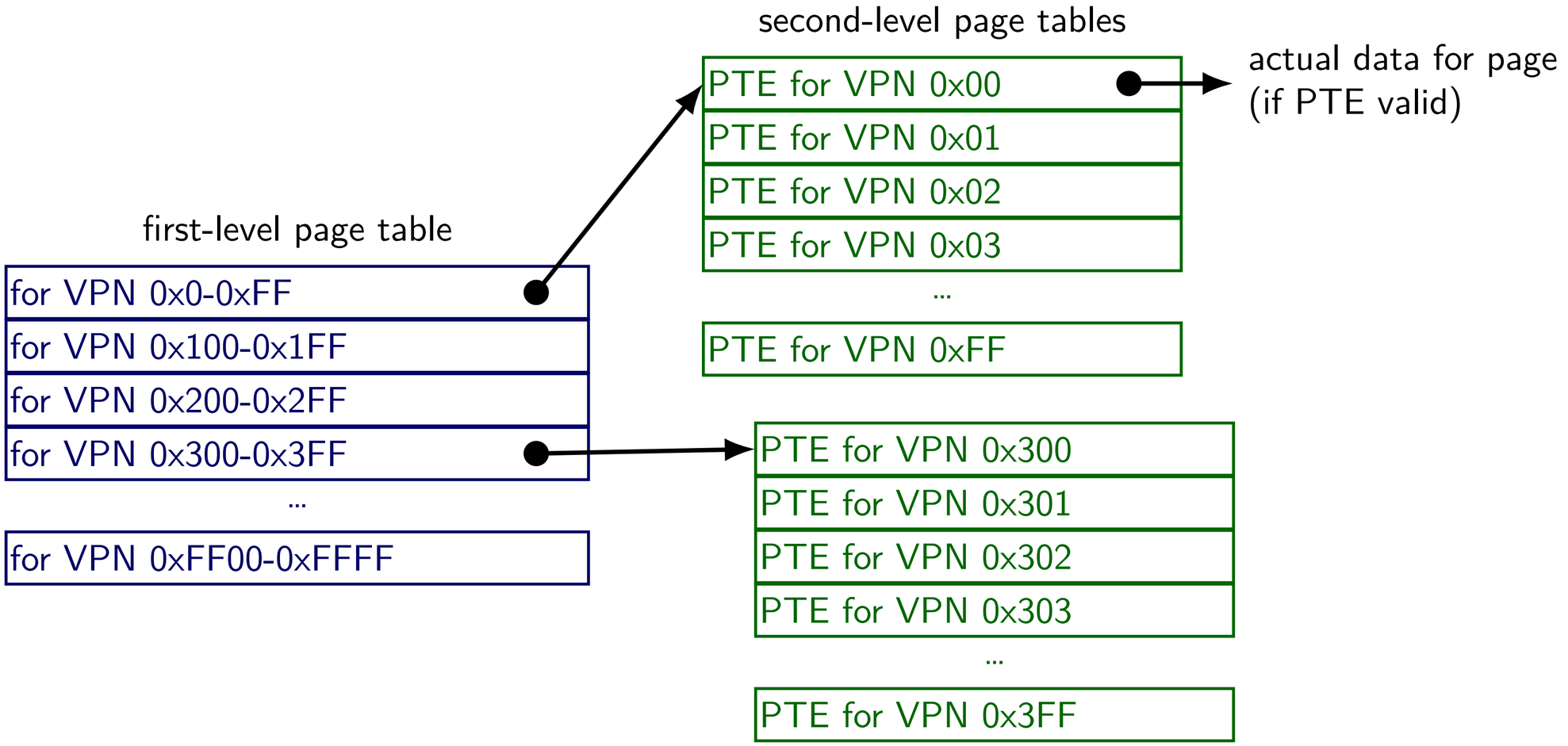
- lookup should be simple
- solution: lookup splits up address bits (no complex calculations)

lookup should not involve many memory accesses

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

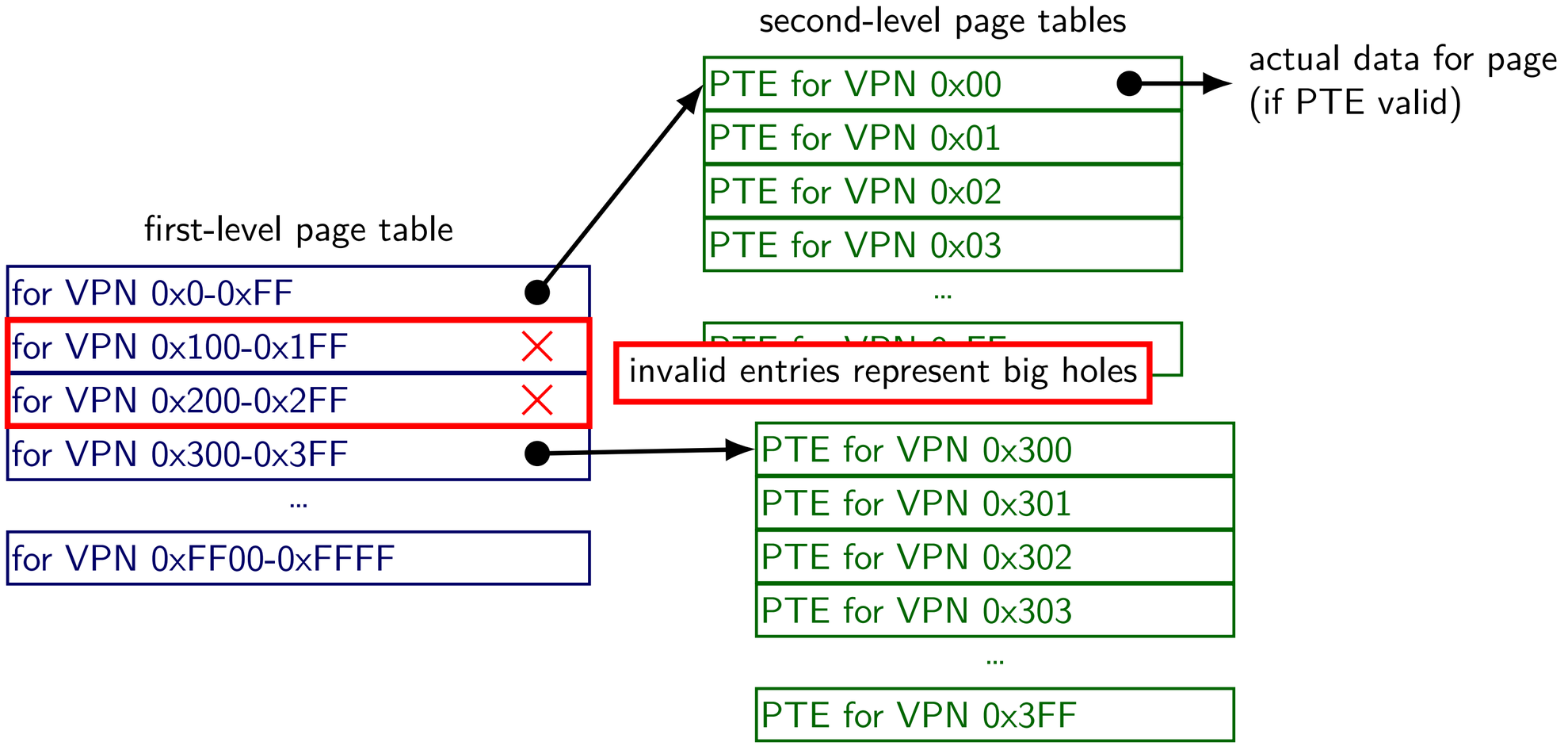
two-level page tables

for 16-bit VPN, 256 entries/table



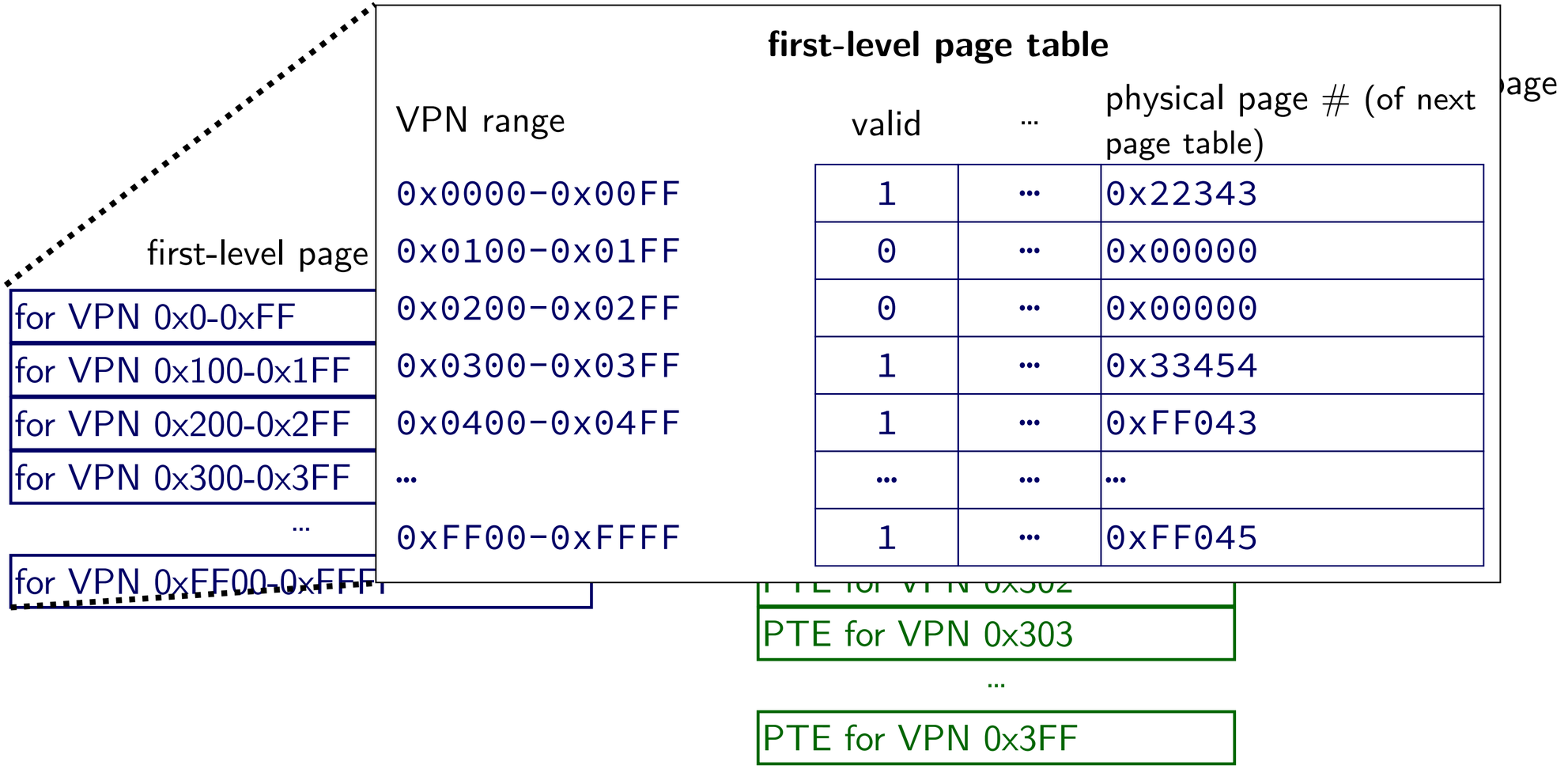
two-level page tables

for 16-bit VPN, 256 entries/table



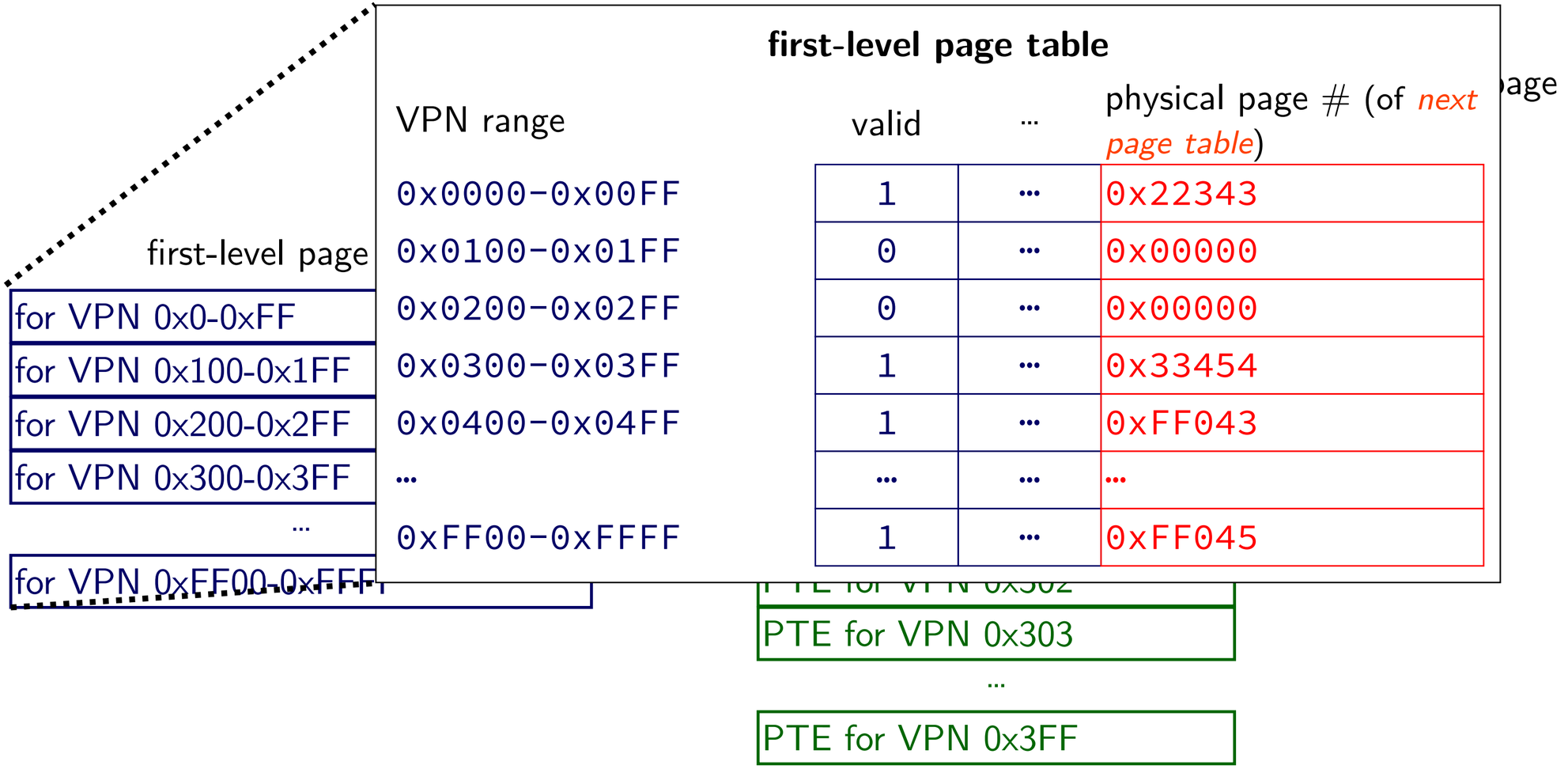
two-level page tables

for 16-bit VPN, 256 entries/table



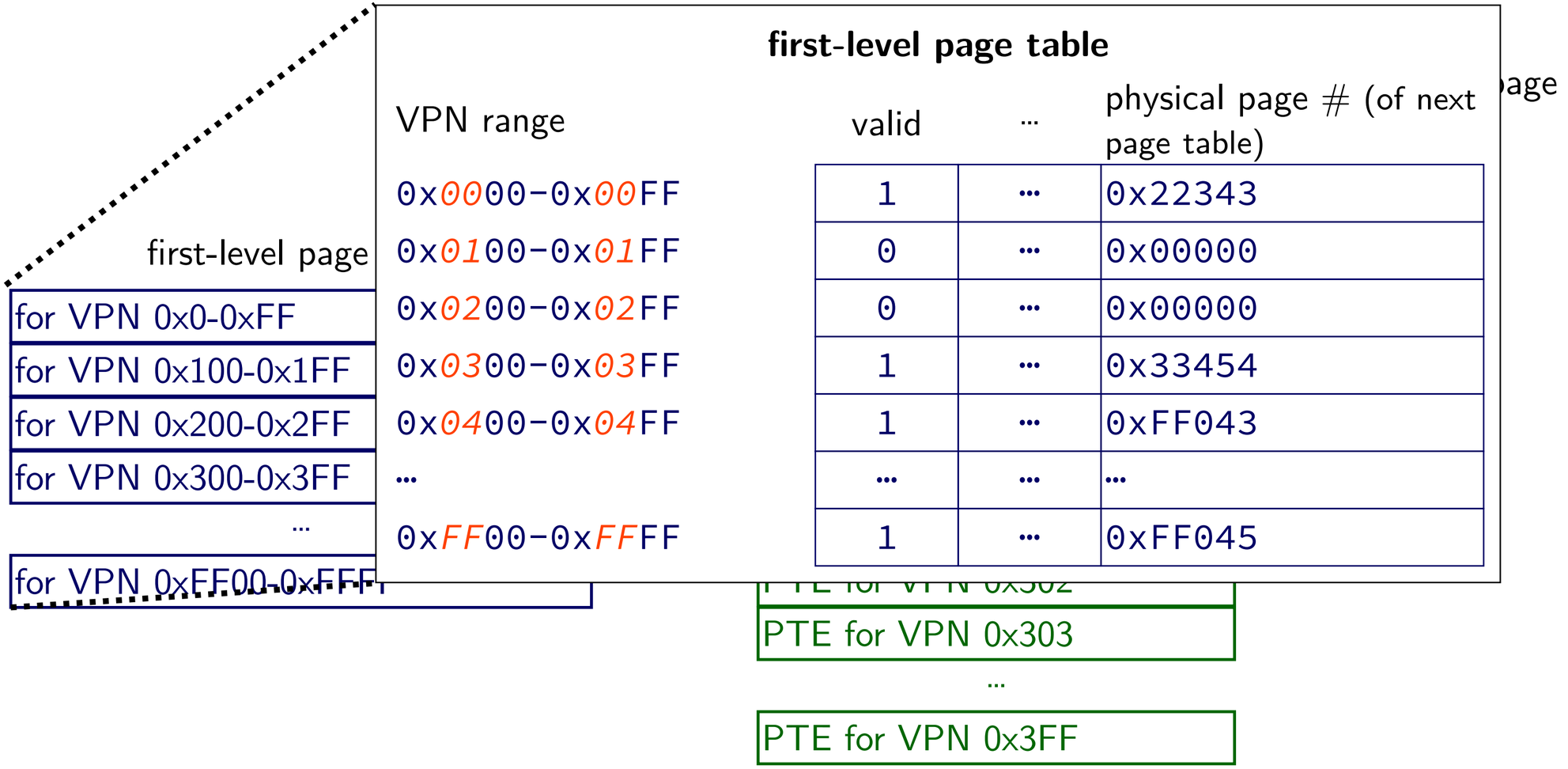
two-level page tables

for 16-bit VPN, 256 entries/table



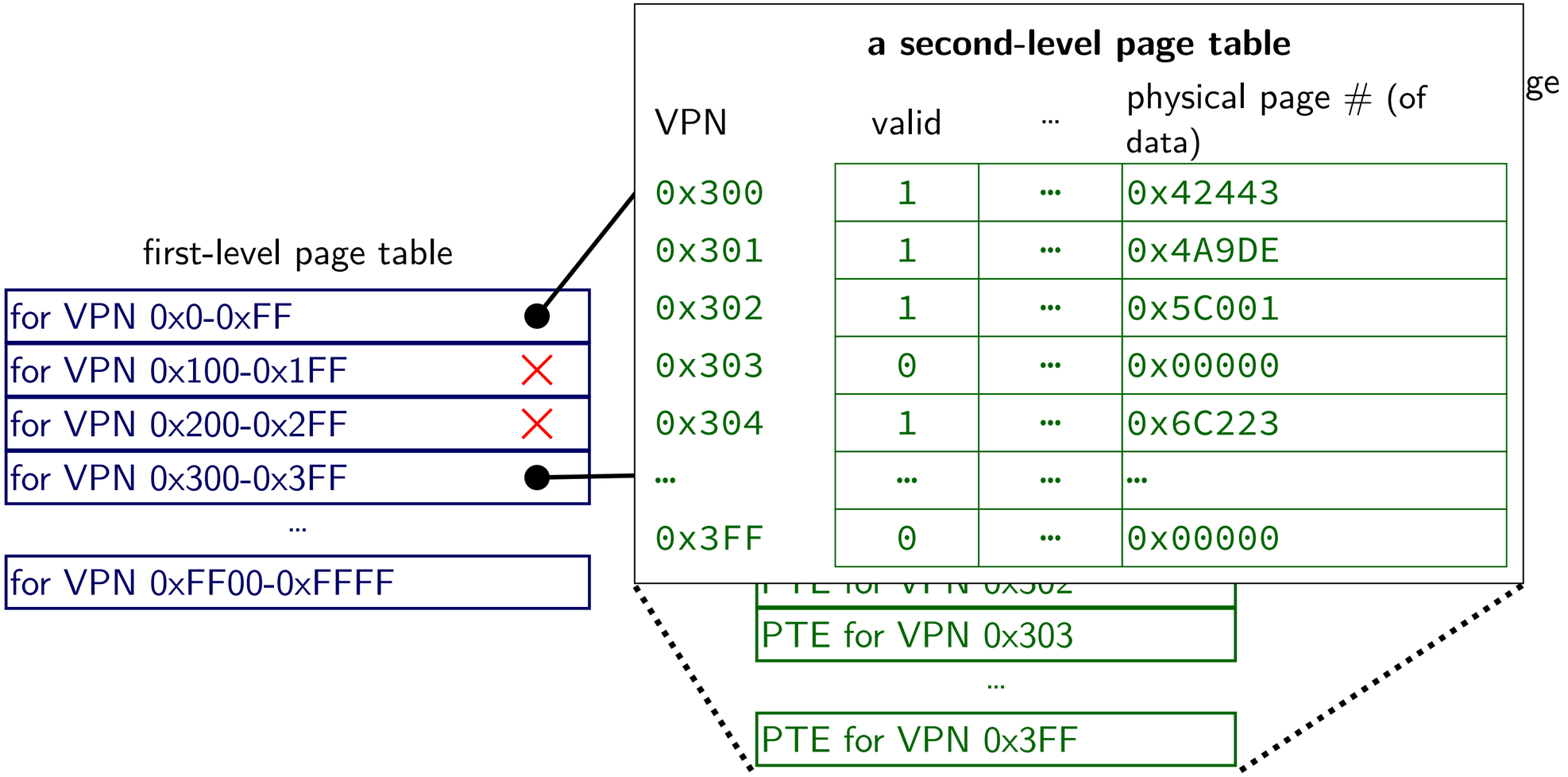
two-level page tables

for 16-bit VPN, 256 entries/table



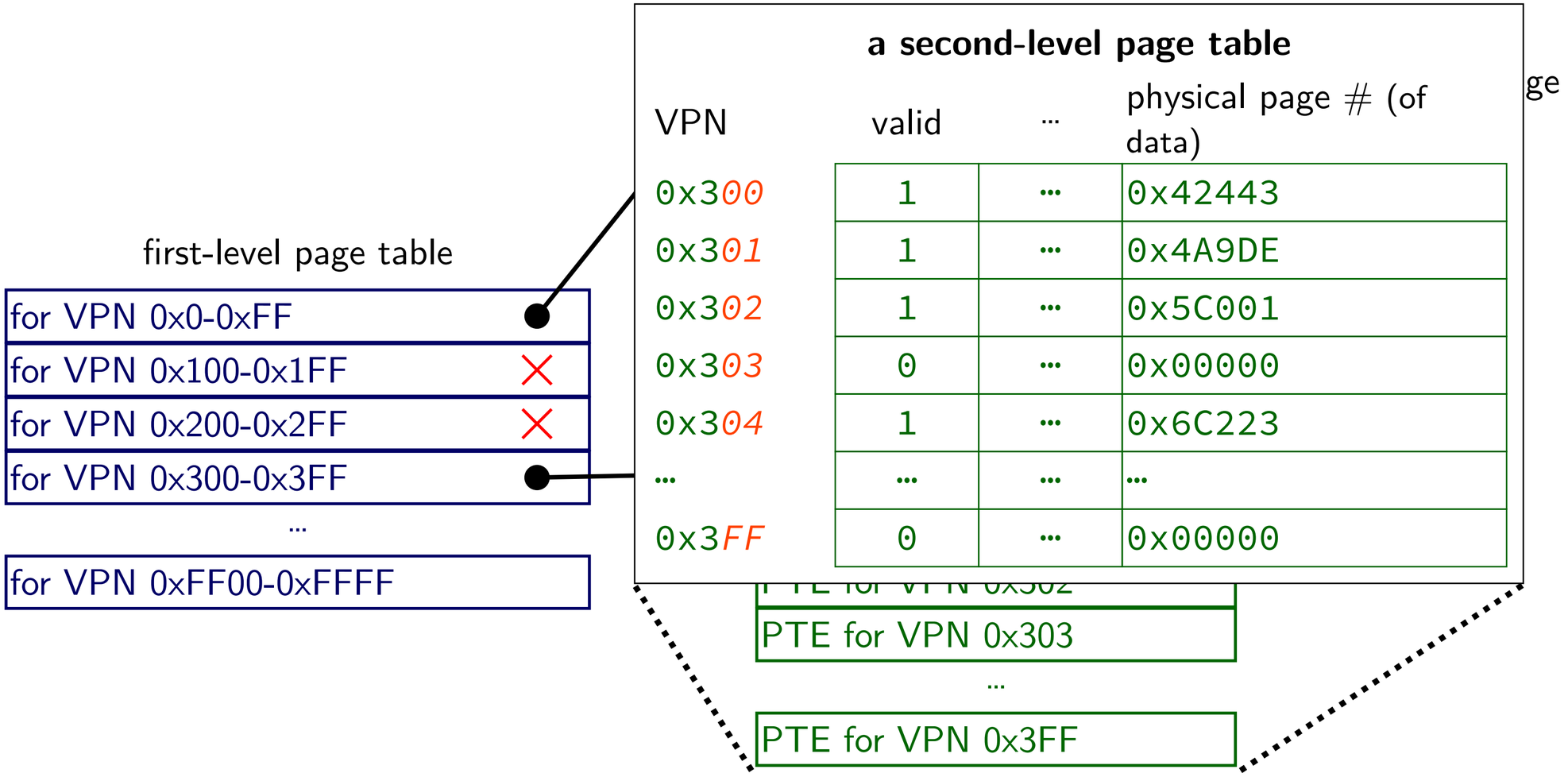
two-level page tables

for 16-bit VPN, 256 entries/table



two-level page tables

for 16-bit VPN, 256 entries/table



two-level page table lookup

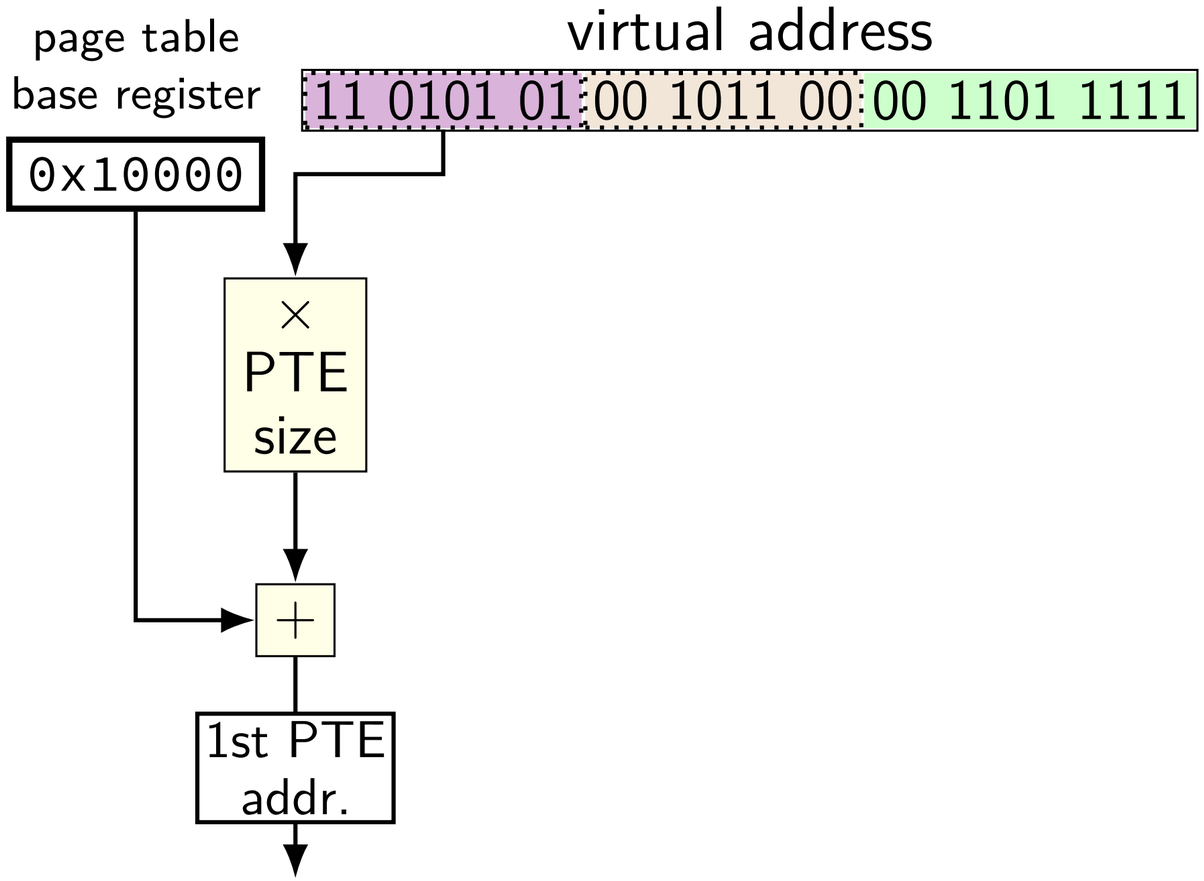
two-level page table lookup

virtual address

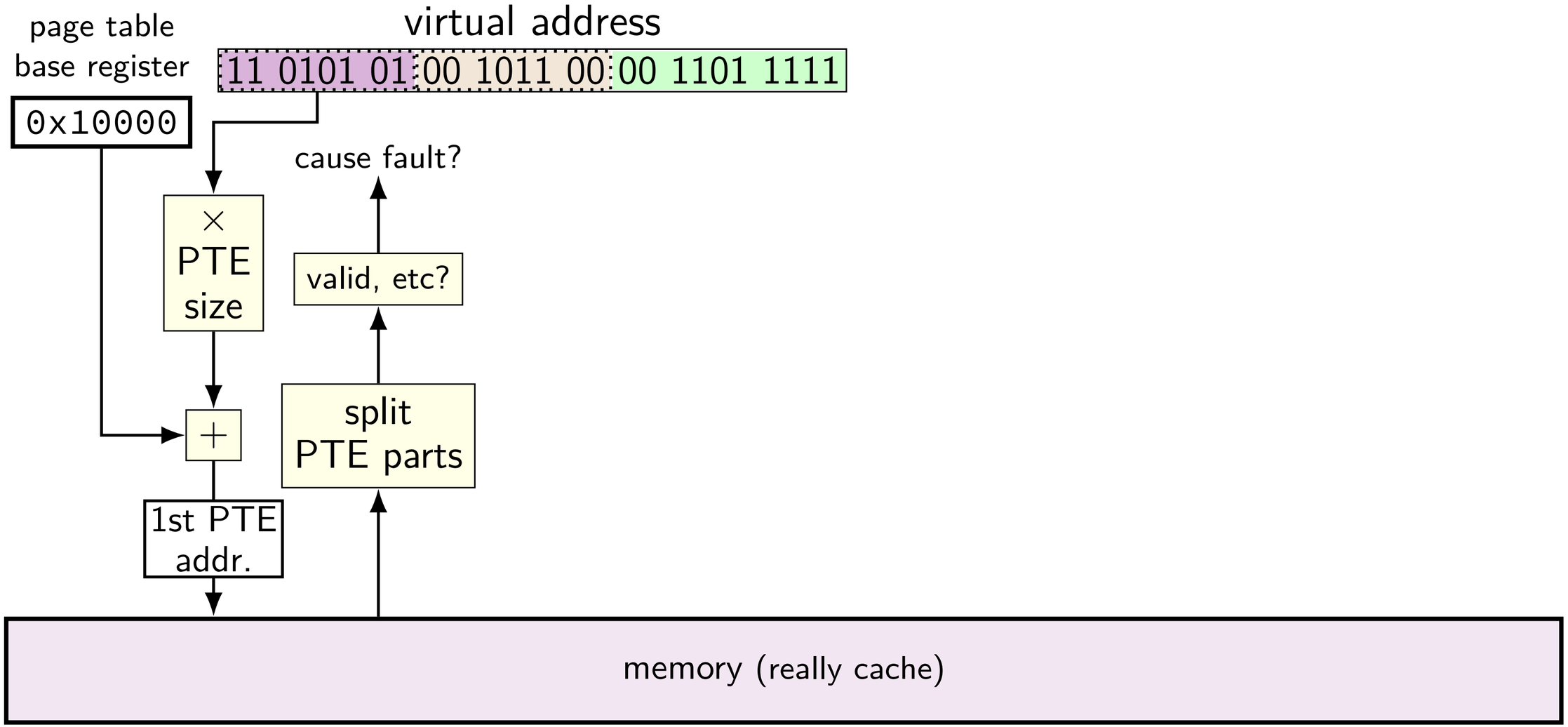
VPN — split into two parts (one per level)

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

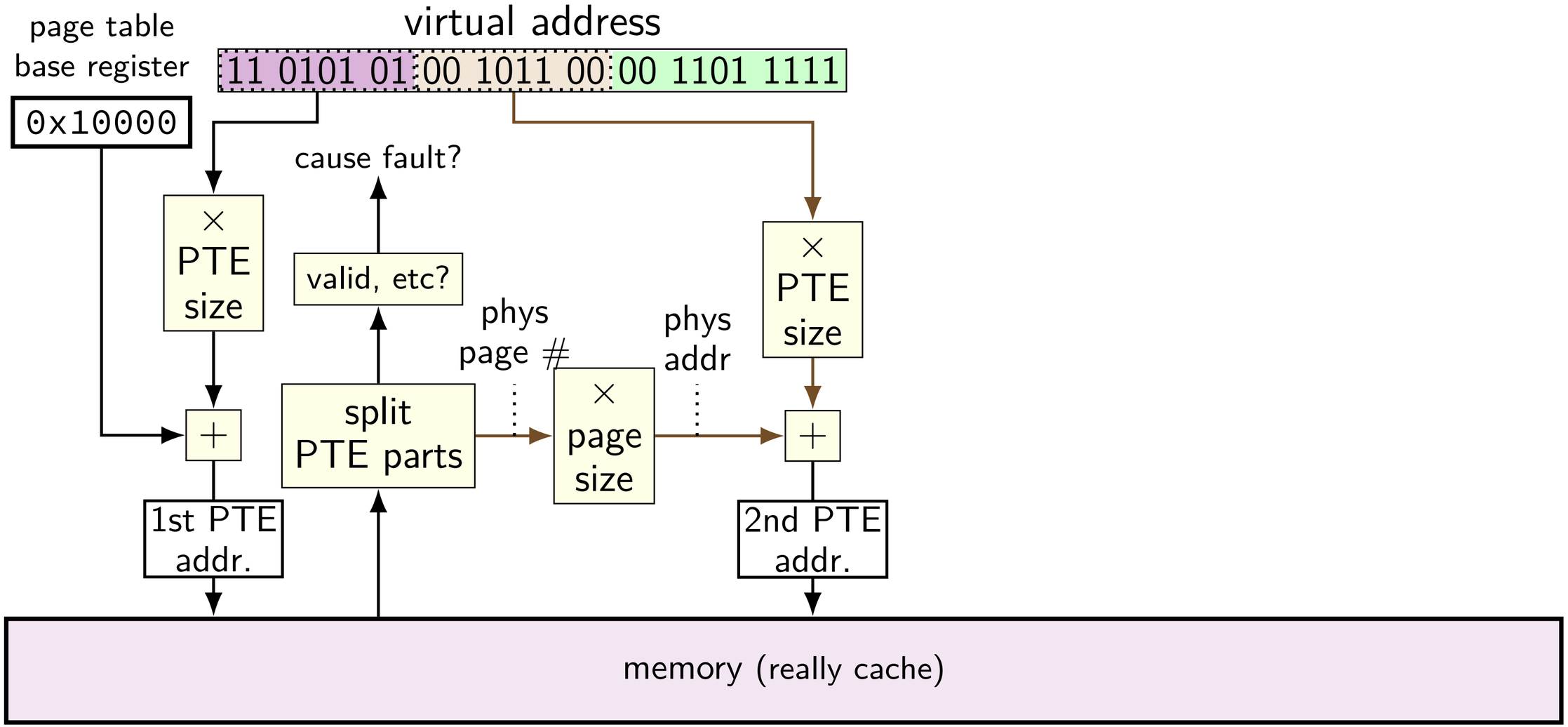
two-level page table lookup



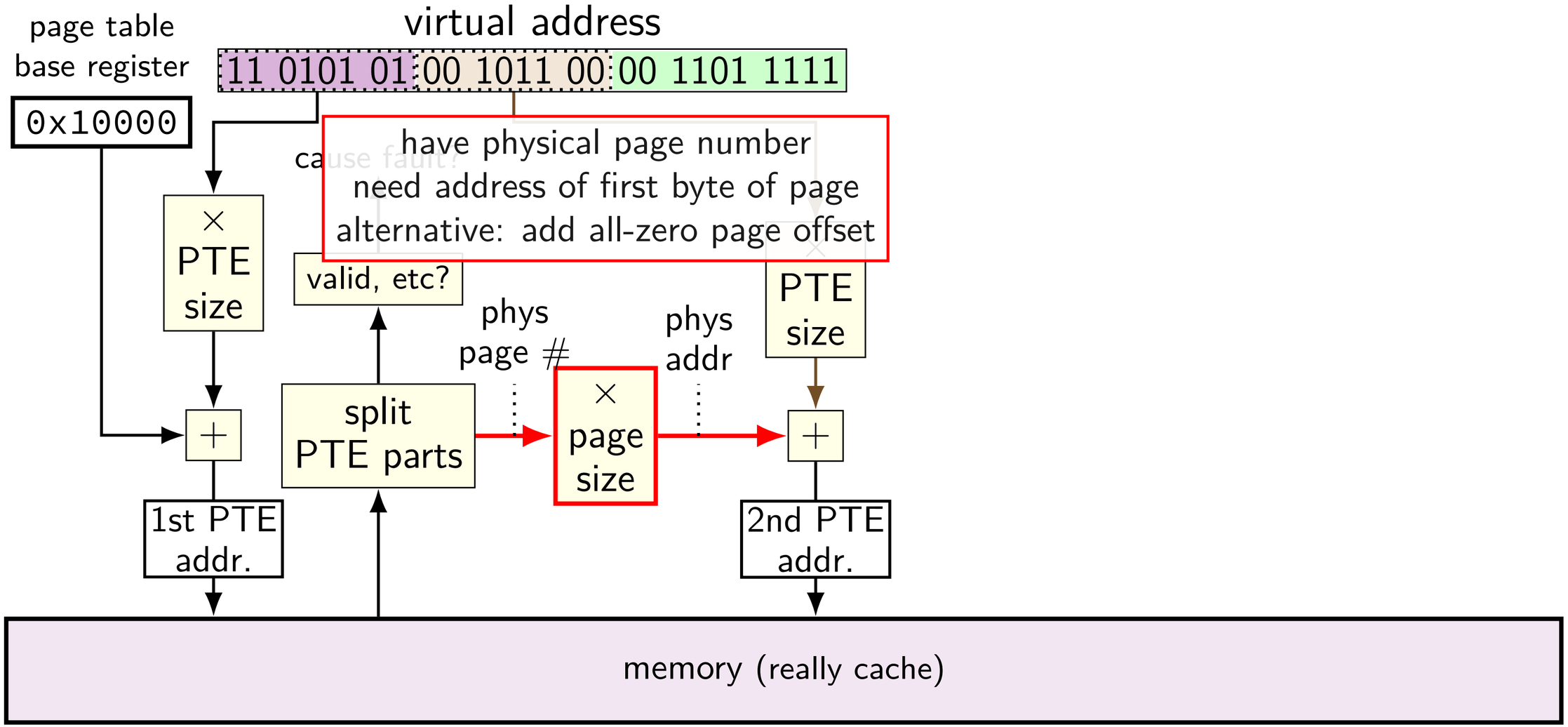
two-level page table lookup



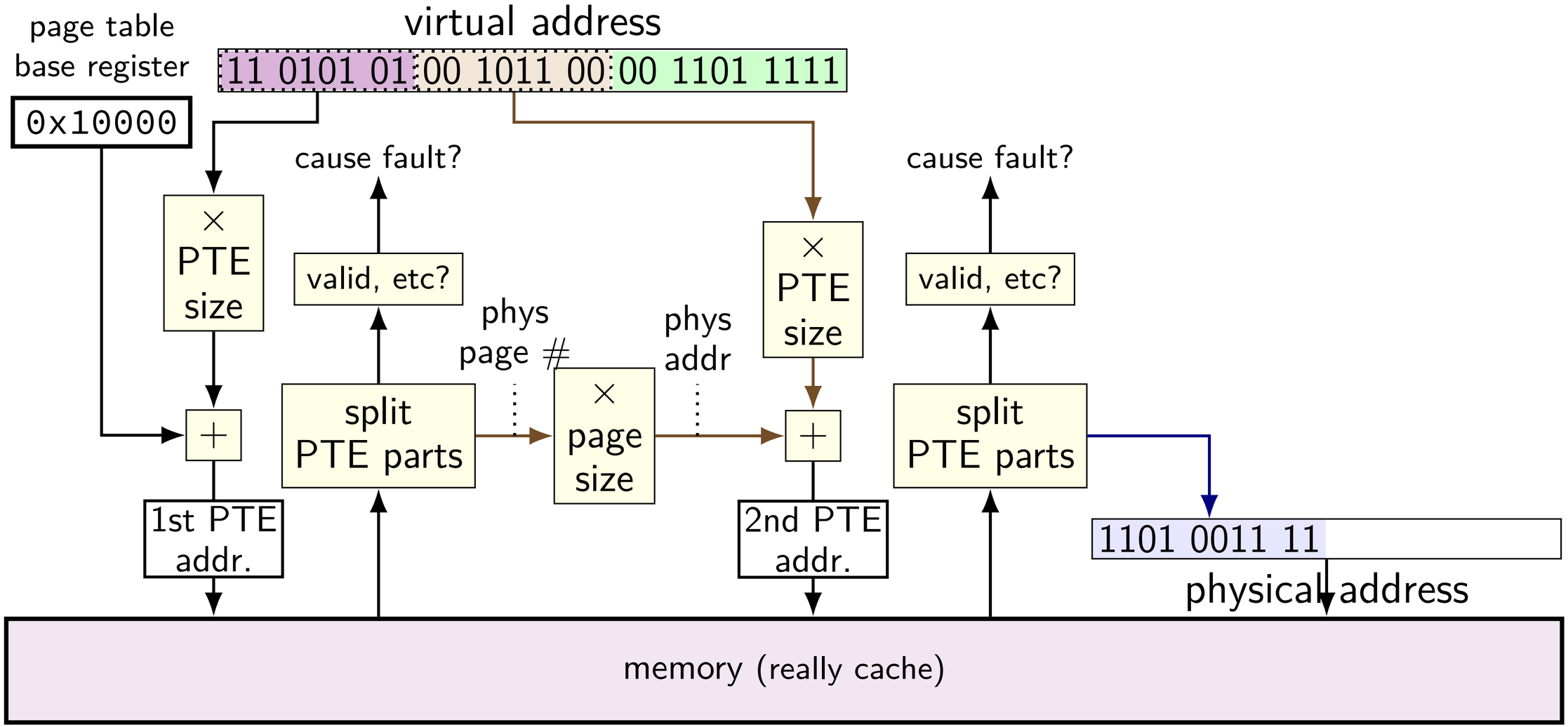
two-level page table lookup



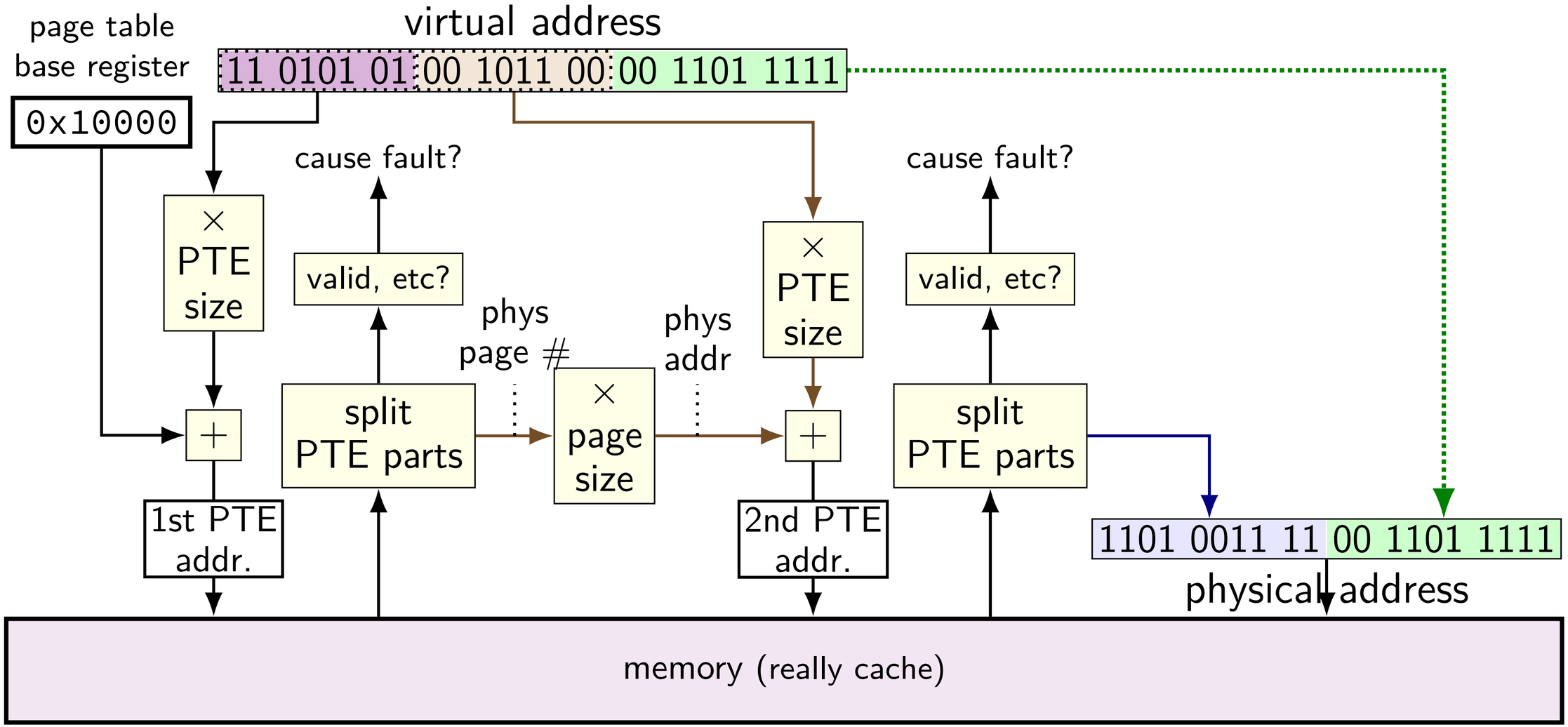
two-level page table lookup



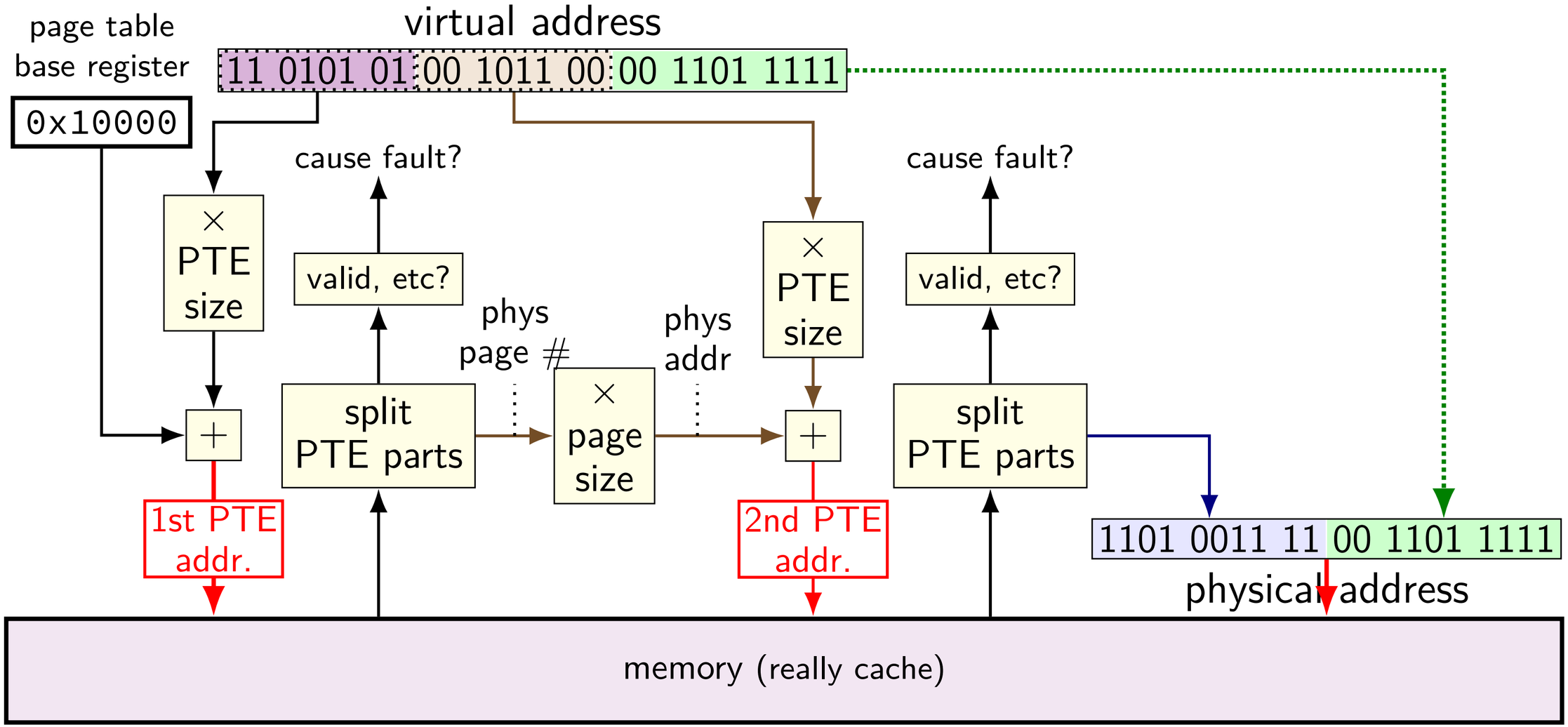
two-level page table lookup



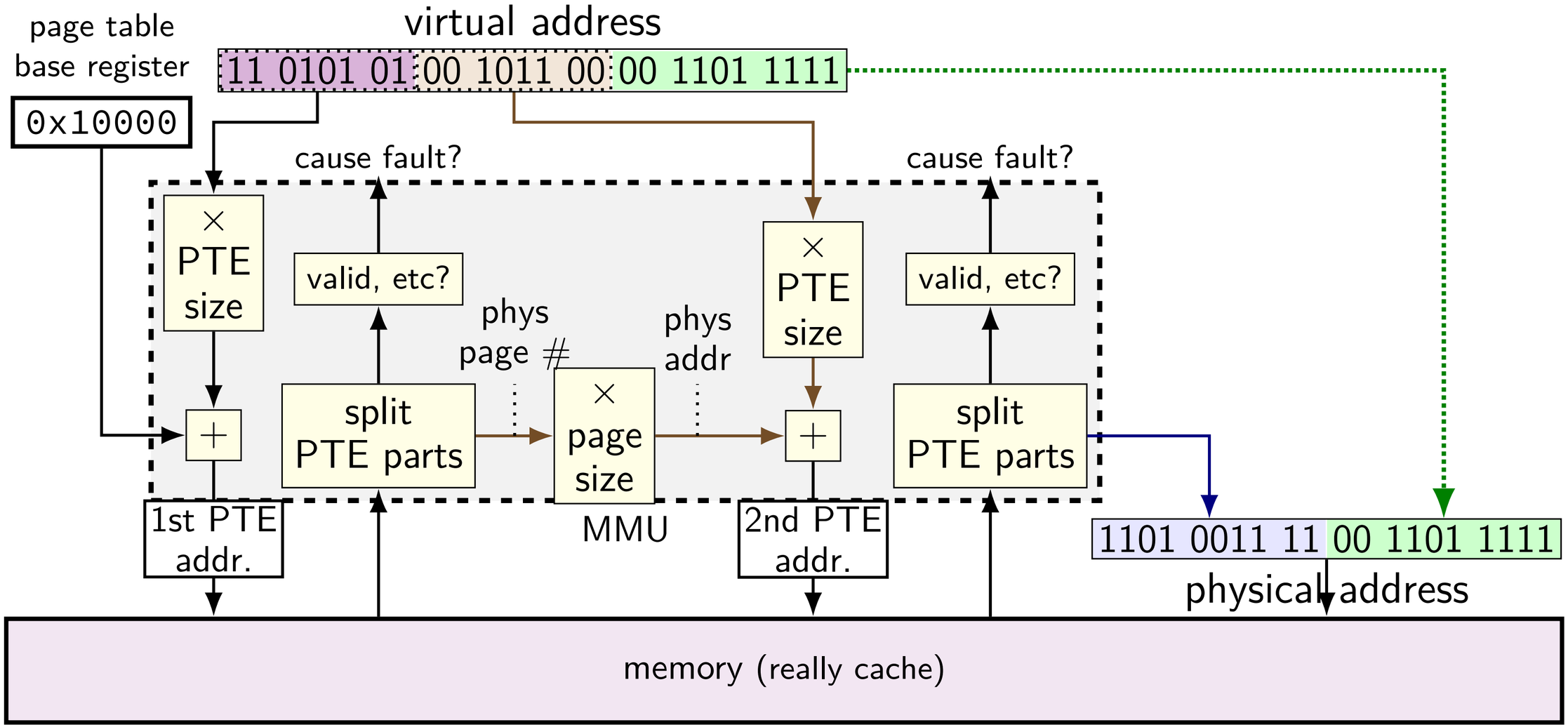
two-level page table lookup



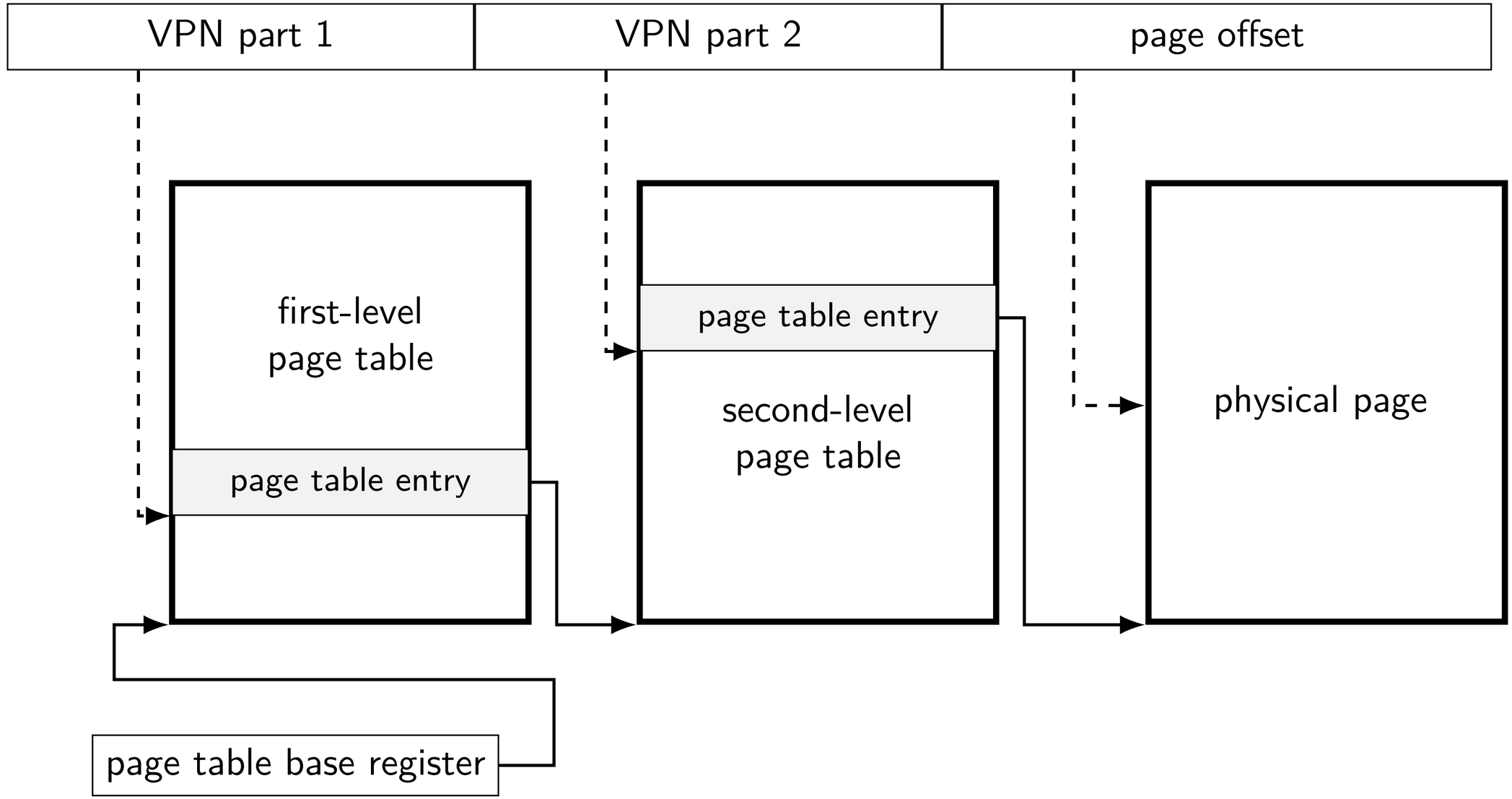
two-level page table lookup



two-level page table lookup



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)

page table counting

counting page tables

3-level page tables, 4096 (2^{12}) byte pages

256 entries (2^{8}) per table

process can access:

- `0x0` (VPN parts `0`, `0`, `0`; PO `0`) - `0x7FFFFF` (VPN parts `0`, `0x7`, `0xFF`; PO `0xFFF`)
- `0x880000` (`0`, `0x8`, `0x80`; `0`)- `0x88FFFF` (`0`, `0x8`, `0x8F`, `0xFFF`)
- `0x100000000` (`0x10`, `0`, `0`; PO `0`) - `0x100000FFF` (`0x10`, `0`, `0`; `0xFFF`)

how many page tables does it need?

page table counting (2)

3-level page tables, 1024 (2^{10}) byte pages

256 entries (2^8) per table

4 byte page table entries

if process can access 10MiB ($10 \cdot 2^{20}$ bytes) of memory (without page faults):

- minimum size of page tables?
- maximum size of page tables — assuming no all-invalid page tables/duplicate physical pages

page table counting (2, soln, minimum)

minimum: all together, starting at beginning of tables at each level (to avoid invalid entries)

- each third-level table: at most 256KB (if all valid)

use 40 third-level tables (to point to 10MiB of data)

can point to all those with 40 of 256 entries in second-level table

need 1st-level table to point to that second level table

$40 + 1 + 1 = 42$ page tables at $256 \times 4 = 1024$ bytes per table = 42KiB

- $42 + 10240 = 10282$ KiB including data; $\sim 0.4\%$ overhead

page table counting (2, soln, maximum)

worst case: as many tables with just one valid entry as possible

maximum number of third-level tables = $256 \times 256 = 65536$

- but can't have this many, because if each is non-empty, would imply 65536 data pages

can have 10240 third-level page tables, each pointing to one data page

can only have up to 256 second-level tables

- each second-level table needs to be pointed to by 1st-level table

can only hvae one 1st level table

$10240 + 256 + 1 = 10497$ page tables at 1KiB per table = 10497KiB

- $10497 + 10240 = 20737$ KiB including data; $\sim 103\%$ overhead

choosing page table sizes

42-bit physical addresses

4 permission bits in page table entries

and want page tables should be <= 1 page in size

if we want 64-bit virtual addresses, how many levels…

with 4096 (2^{12} byte) pages?

with 65536 (2^{16} byte) pages?

choosing page table sizes (4096)

12-bit page offset; 42-bit physical; 64-bit virtual

page table entries: at least 35 bits

valid (1) + permissions (4) + PPN (42 - 12 = 30)

likely choosing 8 byte PTE (closest power of two)

largest page table that fits in one page:

4096/8 = 512 (2^9) entries

9-bit index per level

want 64-12=52 bits of virtual page number

$\lceil 52/9 \rceil = 6$ levels

choosing page table sizes (16384)

16-bit page offset; 42-bit physical; 64-bit virtual

page table entries: at least 31 bits

valid (1) + permissions (4) + PPN (42 - 16 = 26)

can choose 4 byte PTE

largest page table that fits in one page:

65536/4 = 16384 (2^{14}) entries

14-bit index per level

want 64-16=48 bits of virtual page number

$\lceil 48/14 \rceil = 4$ levels

exercise setup: 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

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

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

exercise setup: 2-level splitting

9-bit virtual address

6-bit physical address

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

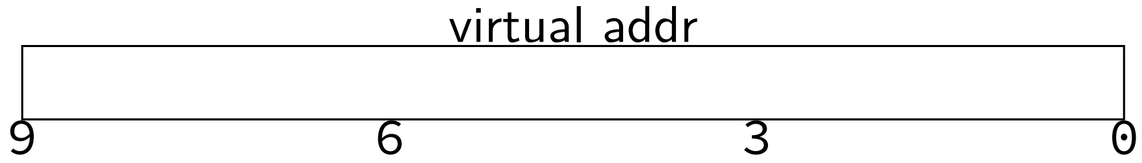
9-bit VA: 6 bit VPN + 3 bit PO

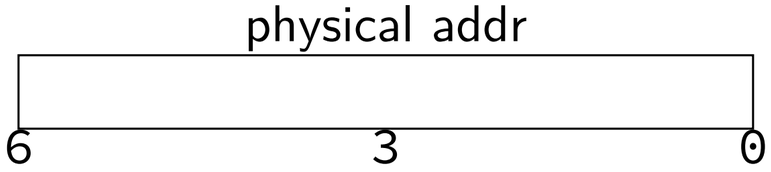
6-bit PA: 3 bit PPN + 3 bit PO

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

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

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





exercise setup: 2-level splitting

9-bit virtual address

6-bit physical address

8-byte pages → 3-bit page offset (bottom)

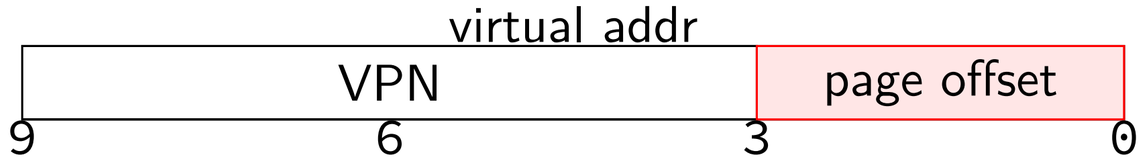
9-bit VA: 6 bit VPN + 3 bit PO

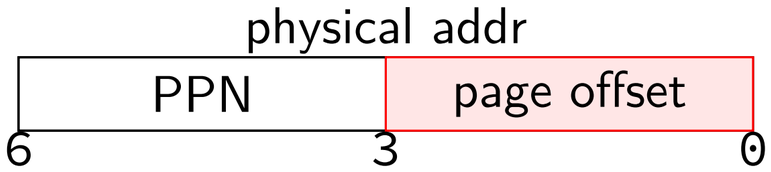
6-bit PA: 3 bit PPN + 3 bit PO

1 page page tables w/ 1 byte entry → 8 entry PTs

8 entry page tables → 3-bit VPN parts

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





exercise setup: 2-level splitting

9-bit virtual address

6-bit physical address

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

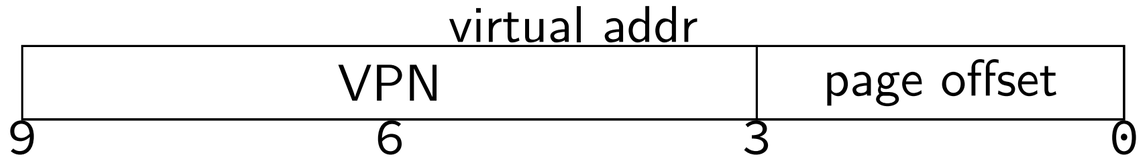
9-bit VA: 6 bit VPN + 3 bit PO

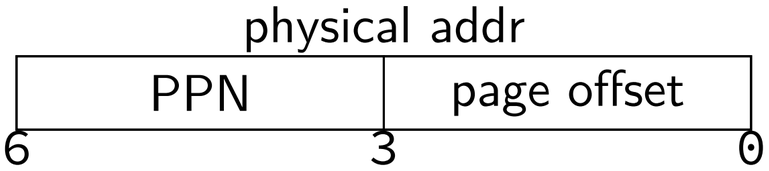
6-bit PA: 3 bit PPN + 3 bit PO

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

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

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





page table (either level)

	valid?	PPN
0		
1		
2		
...	...	...
7		

exercise setup: 2-level splitting

9-bit virtual address

6-bit physical address

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

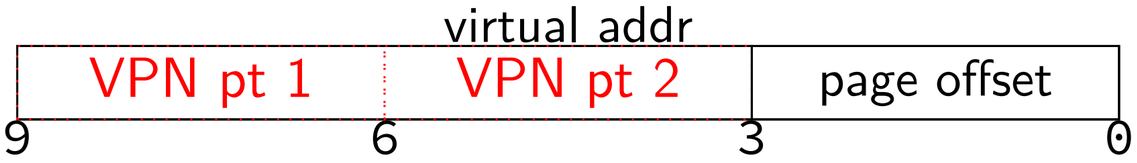
9-bit VA: 6 bit VPN + 3 bit PO

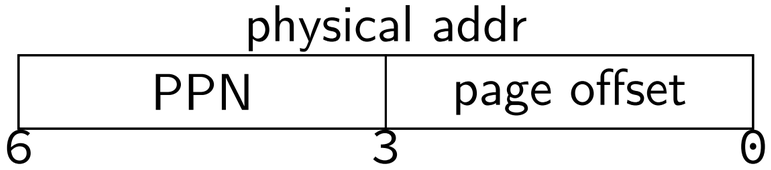
6-bit PA: 3 bit PPN + 3 bit PO

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

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

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





page table (either level)

	valid?	PPN
0		
1		
2		
...	...	...
7		

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	0x24–27	F4 A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC DC EC FC

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	0x24–27	F4 A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC DC EC FC

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	0x24–27	F4 A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC DC EC FC

`0x129 = 1 0010 1001`

`0x20 + 100`$_{\text{TWO}}$ $\times$ `1 = 0x24`

PTE 1 value:

`0xF4 = 1111 0100`

`PPN 111, 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 `0x129`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	0x24–27	F4 A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC DC EC FC

`0x129 = 1 0010 1001`

`0x20 + 100`$_{\text{TWO}}$ $\times$ `1 = 0x24`

PTE 1 value:

`0xF4 = 1111 0100`

`PPN 111, 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 `0x129`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	*0x24–27*	F4 A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC DC EC FC

`0x129 = 1 0010 1001`

`0x20 + 100`$_{\text{TWO}}$ $\times$ `1 = 0x24`

PTE 1 value:

`0xF4 = 1111 0100`

`PPN 111, 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 `0x129`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	0x24–27	*F4* A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	*0x3C–3F*	EC DC EC FC

0x129 = 1 0010 1001

0x20 + 100_{TWO} × 1 = 0x24

PTE 1 value:

0xF4 = *111*1 0100

PPN *111*, valid 1

PTE 2 addr:

111 000_{TWO} + 101_{TWO} × 1 = 0x3D

(111000_{TWO} = 111×page size)

PTE 2 value: 0xDC

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 `0x129`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	0x24–27	F4 A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC *DC* EC FC

`0x129 = 1 00`*`10`*` `*`1`*`001`

`0x20 + 100`$_{TWO}$ $\times$ `1 = 0x24`

PTE 1 value:

`0xF4 = 1111 0100`

`PPN 111, valid 1`

PTE 2 addr:

`111 000`$_{TWO}$ `+` *`101`*$_{TWO}$ $\times$ `1 = 0x3D`

(`111000`$_{TWO}$ `= 111`$\times$ page size)

PTE 2 value: `0xDC`

`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 `0x129`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	0x24–27	F4 A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC DC EC FC

`0x129 = 1 0010 1001`

`0x20 + 100`$_{\text{TWO}}$ $\times$ `1 = 0x24`

PTE 1 value:

`0xF4 = 1111 0100`

`PPN 111, valid 1`

PTE 2 addr:

`111 000`$_{\text{TWO}}$ `+ 101`$_{\text{TWO}}$ $\times$ `1 = 0x3D`

(`111000`$_{\text{TWO}}$ `= 111`$\times$ page size)

PTE 2 value: `0xDC`

`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 `0x129`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	0x24–27	F4 A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC DC EC FC

`0x129 = 1 0010 1001`

`0x20 + 100`$_{\text{TWO}}$ $\times$ `1 = 0x24`

PTE 1 value:

`0xF4 = 1111 0100`

`PPN 111, valid 1`

PTE 2 addr:

`111 000`$_{\text{TWO}}$ `+ 101`$_{\text{TWO}}$ $\times$ `1 = 0x3D`

(`111000`$_{\text{TWO}}$ = 111$\times$page size)

PTE 2 value: `0xDC`

`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 `0x129`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	0x24–27	F4 A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	*0x30–33*	BA *0A* BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC DC EC FC

`0x129 = 1 0010 1001`

`0x20 + 100`$_{\text{TWO}}$ $\times$ `1 = 0x24`

PTE 1 value:

`0xF4 = 1111 0100`

`PPN 111, valid 1`

PTE 2 addr:

`111 000`$_{\text{TWO}}$ `+ 101`$_{\text{TWO}}$ $\times$ `1 = 0x3D`

(`111000`$_{\text{TWO}}$ = 111$\times$page size)

PTE 2 value: `0xDC`

`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 `0x129`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	00 91 72 13
0x04–07	44 55 66 77	0x24–27	F4 A5 36 07
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC DC EC FC

0x129 = 1 0010 1001

0x20 + 100_{TWO} × 1 = 0x24

PTE 1 value:

0xF4 = 1111 0100

PPN 111, valid 1

PTE 2 addr:

111 000_{TWO} + 101_{TWO} × 1 = 0x3D

(111000_{TWO} = 111×page size)

PTE 2 value: 0xDC

PPN 110; valid 1

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

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x0FB = *011* 111 011_{TWO}

PTE 1 addr: 0x08 + 011_{TWO} × 1

PTE 1: 0xBB at 0x0B
0xBB = 1011 1011_{TWO}

PTE 1: PPN 101 (5) valid 1

PTE 2 addr:
101 000_{TWO} + 111_{TWO} × 1
(101000_{TWO} = 101×page size)

PTE 2: 0xF0 at 0x2F

PTE 2: PPN 111 (7) valid 1
111 011_{TWO} = 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 address	bytes			
0x00–03	00	11	22	33
0x04–07	44	55	66	77
0x08–0B	88	99	AA	*BB*
0x0C–0F	CC	DD	EE	FF
0x10–13	1A	2A	3A	4A
0x14–17	1B	2B	3B	4B
0x18–1B	1C	2C	3C	4C
0x1C–1F	1C	2C	3C	4C

physical address	bytes			
0x20–23	D0	D1	D2	D3
0x24–24	D4	D5	D6	D7
0x28–2B	89	9A	AB	BC
0x2C–2F	CD	DE	EF	F0
0x30–33	BA	0A	BA	0A
0x34–37	CB	0B	CB	0B
0x38–3B	DC	0C	DC	0C
0x3C–3F	EC	0C	EC	0C

0x0FB = 011 111 011_{TWO}

PTE 1 addr: 0x08 + 011_{TWO} × 1

PTE 1: *0xBB* at *0x0B*

0xBB = 1011 1011_{TWO}

PTE 1: PPN 101 (5) valid 1

PTE 2 addr:

101 000_{TWO} + 111_{TWO} × 1

(101000_{TWO} = 101×page size)

PTE 2: 0xF0 at 0x2F

PTE 2: PPN 111 (7) valid 1

111 011_{TWO} = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x0FB = 011 111 011_{TWO}

PTE 1 addr: 0x08 + 011_{TWO} × 1

PTE 1: 0xBB at 0x0B
0xBB = *1011* 1011_{TWO}

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

PTE 2 addr:
101 000_{TWO} + 111_{TWO} × 1
(101000_{TWO} = *101*×page size)

PTE 2: 0xF0 at 0x2F

PTE 2: PPN 111 (7) valid 1
111 011_{TWO} = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x0FB = 011 *111* 011$_{TWO}$

PTE 1 addr: 0x08 + 011$_{TWO}$ × 1

PTE 1: 0xBB at 0x0B
0xBB = 1011 1011$_{TWO}$

PTE 1: PPN 101 (5) valid 1

PTE 2 addr:
101 000$_{TWO}$ + *111*$_{TWO}$ × 1
(101000$_{TWO}$ = 101×page size)

PTE 2: 0xF0 at 0x2F

PTE 2: PPN 111 (7) valid 1
111 011$_{TWO}$ = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	*0x2C–2F*	CD DE EF *F0*
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x0FB = 011 111 011_{TWO}

PTE 1 addr: 0x08 + 011_{TWO} × 1

PTE 1: 0xBB at 0x0B
0xBB = 1011 1011_{TWO}

PTE 1: PPN 101 (5) valid 1

PTE 2 addr:
101 000_{TWO} + 111_{TWO} × 1
(101000_{TWO} = 101×page size)

PTE 2: *0xF0* at *0x2F*

PTE 2: PPN 111 (7) valid 1
111 011_{TWO} = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x0FB = 011 111 011_{TWO}

PTE 1 addr: 0x08 + 011_{TWO} × 1

PTE 1: 0xBB at 0x0B
0xBB = 1011 1011_{TWO}

PTE 1: PPN 101 (5) valid 1

PTE 2 addr:
101 000_{TWO} + 111_{TWO} × 1
(101000_{TWO} = 101×page size)

PTE 2: 0xF0 at 0x2F

PTE 2: PPN *111* (7) valid 1
111 011_{TWO} = 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x0FB = 011 111 011_{TWO}

PTE 1 addr: 0x08 + 011_{TWO} × 1

PTE 1: 0xBB at 0x0B
0xBB = $1011\ 1011_{TWO}$

PTE 1: PPN 101 (5) valid 1

PTE 2 addr:
$101\ 000_{TWO}$ + 111_{TWO} × 1
(101000_{TWO} = 101×page size)

PTE 2: 0xF0 at 0x2F

PTE 2: PPN 111 (7) valid 1
$111\ 011_{TWO}$ = 0x3B → 0x0C

exercise interlude: parameters

multi-level parameter exercise (1)

8192 byte (2^{13} byte) pages

8-byte page table entries

each page table takes up 8192 bytes (one page)

four-level page tables

size of virtual page number? virtual address?

multi-level parameter exercise (2)

16384-byte (2^{14} byte) pages

16-byte page table entries

want page tables to be at most one page in size

how many levels do we need to support 64-bit virtual address?

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 `0x10B`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE 0F	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 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 `0x10B`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE 0F	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x10B = *100* 001 011_{TWO}

PTE 1 addr: 0x10 + 100_{TWO} × 1

PTE 1: 0x1B at 0x14

0x1B = 0001 1011_{TWO}

PTE 1: PPN 000 (0) valid 1

PTE 2 addr:

000 000_{TWO} + 001_{TWO} × 1

PTE 2: 0x11 at 0x01

PTE 2: PPN 000 (0) valid 1

000 011_{TWO} = 0x03 → 0x33

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 `0x10B`

physical address	bytes			
0x00–03	00	11	22	33
0x04–07	44	55	66	77
0x08–0B	88	99	AA	BB
0x0C–0F	CC	DD	EE	0F
0x10–13	1A	2A	3A	4A
0x14–17	*1B*	2B	3B	4B
0x18–1B	1C	2C	3C	4C
0x1C–1F	1C	2C	3C	4C

physical address	bytes			
0x20–23	D0	D1	D2	D3
0x24–24	D4	D5	D6	D7
0x28–2B	89	9A	AB	BC
0x2C–2F	CD	DE	EF	F0
0x30–33	BA	0A	BA	0A
0x34–37	CB	0B	CB	0B
0x38–3B	DC	0C	DC	0C
0x3C–3F	EC	0C	EC	0C

0x10B = $100\ 001\ 011_{TWO}$

PTE 1 addr: 0x10 + 100_{TWO} × 1

PTE 1: *0x1B* at *0x14*

0x1B = $0001\ 1011_{TWO}$

PTE 1: PPN 000 (0) valid 1

PTE 2 addr:

$000\ 000_{TWO}$ + 001_{TWO} × 1

PTE 2: 0x11 at 0x01

PTE 2: PPN 000 (0) valid 1

$000\ 011_{TWO}$ = 0x03 → 0x33

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 `0x10B`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE 0F	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$\texttt{0x10B} = \texttt{100 001 011}_{\texttt{TWO}}$

PTE 1 addr: $\texttt{0x10} + \texttt{100}_{\texttt{TWO}} \times 1$

PTE 1: 0x1B at 0x14

$\texttt{0x1B} = \textit{0001}\ \texttt{1011}_{\texttt{TWO}}$

PTE 1: PPN *000* (0) valid *1*

PTE 2 addr:

$\textit{000}\ \texttt{000}_{\texttt{TWO}} + \texttt{001}_{\texttt{TWO}} \times 1$

PTE 2: 0x11 at 0x01

PTE 2: PPN 000 (0) valid 1

$\texttt{000}\ \texttt{011}_{\texttt{TWO}} = \texttt{0x03} \rightarrow \texttt{0x33}$

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 `0x10B`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE 0F	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

$\texttt{0x10B} = \texttt{100}\ \textit{001}\ \texttt{011}_{\text{TWO}}$

PTE 1 addr: $\texttt{0x10} + \texttt{100}_{\text{TWO}} \times 1$

PTE 1: 0x1B at 0x14

$\texttt{0x1B} = \texttt{0001 1011}_{\text{TWO}}$

PTE 1: PPN 000 (0) valid 1

PTE 2 addr:

$\texttt{000 000}_{\text{TWO}} + \textit{001}_{\text{TWO}} \times 1$

PTE 2: 0x11 at 0x01

PTE 2: PPN 000 (0) valid 1

$\texttt{000}\ \texttt{011}_{\text{TWO}} = \texttt{0x03} \rightarrow \texttt{0x33}$

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 `0x10B`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE 0F	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

`0x10B` = $100\ 001\ 011_{TWO}$

PTE 1 addr: `0x10` + $100_{TWO} \times 1$

PTE 1: `0x1B` at `0x14`
`0x1B` = $0001\ 1011_{TWO}$

PTE 1: PPN 000 (0) valid 1

PTE 2 addr:
$000\ 000_{TWO} + 001_{TWO} \times 1$

PTE 2: *0x11* at *0x01*

PTE 2: PPN 000 (0) valid 1
$000\ 011_{TWO}$ = 0x03 → 0x33

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 `0x10B`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE 0F	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

`0x10B` = $100\ 001\ 011_{TWO}$

PTE 1 addr: `0x10` + $100_{TWO} \times 1$

PTE 1: `0x1B` at `0x14`
`0x1B` = $0001\ 1011_{TWO}$

PTE 1: PPN 000 (0) valid 1

PTE 2 addr:
$000\ 000_{TWO} + 001_{TWO} \times 1$

PTE 2: `0x11` at `0x01`

PTE 2: PPN *000* (0) valid 1
$000\ 011_{TWO}$ = 0x03 → 0x33

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 `0x10B`

physical address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE 0F	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x10B = 100 001 011_{TWO}

PTE 1 addr: 0x10 + 100_{TWO} × 1

PTE 1: 0x1B at 0x14
0x1B = 0001 1011_{TWO}

PTE 1: PPN 000 (0) valid 1

PTE 2 addr:
000 000_{TWO} + 001_{TWO} × 1

PTE 2: 0x11 at 0x01

PTE 2: PPN 000 (0) valid 1
000 011_{TWO} = 0x03 → 0x33

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

`0x0F3 = 000 001 011`$_{TWO}$

PTE 1: `0x88 at 0x08`
`(0x08 = base +` 000_{TWO}`)`

PTE 1: `PPN 100 (4) valid 0`
page fault!

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 0A 0B	0x28–2B	89 9A AB BC
0x0C–0F	0C 0D 0E 0F	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 0A 0B	0x28–2B	89 9A AB BC
0x0C–0F	0C 0D 0E 0F	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	1C 2C 3C 4C	0x38–3B	DC 0C DC 0C
0x1C–1F	1C 2C 3C 4C	0x3C–3F	EC 0C EC 0C

0x1CB = 111 001 011_{TWO}

PTE 1: 0xFF at 0x0F

  (= base + 111_{TWO})

PTE 1: PPN 111 (7) valid 1

PTE 2: 0x0C at 0x39 PTE 2: PPN 000 (0) valid *0 page fault!*

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	3D 5C 7C 9C	0x38–3B	DC 0C DC 0C
0x1C–1F	AC BC 13 57	0x3C–3F	EC 0C EC 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	3D 5C 7C 9C	0x38–3B	DC 0C DC 0C
0x1C–1F	AC BC 13 57	0x3C–3F	EC 0C EC 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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	3D 5C 7C 9C	0x38–3B	DC 0C DC 0C
0x1C–1F	AC BC 13 57	0x3C–3F	EC 0C EC 0C

0x76 = 110 111 0110

PTE 1 address:

0x10 + $110_{TWO} \times 2$ = 0x1C

PTE 1:

AC BC → PPN 10 valid 1

PTE 2 address:

$10\ 0000_{TWO}$ + $111_{TWO} \times 2$ = 0x2E

(100000_{TWO} = 10×page size)

PTE 2:

EF F0 → PPN 11 valid 1

11 0110 = 0x36 → 0xCB

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	3D 5C 7C 9C	0x38–3B	DC 0C DC 0C
0x1C–1F	AC BC 13 57	0x3C–3F	EC 0C EC 0C

0x76 = *110* 111 0110

PTE 1 address:

0x10 + $110_{\text{TWO}} \times 2$ = 0x1C

PTE 1:

AC BC → PPN 10 valid 1

PTE 2 address:

$10\ 0000_{\text{TWO}}$ + $111_{\text{TWO}} \times 2$ = 0x2E

(100000_{TWO} = 10×page size)

PTE 2:

EF F0 → PPN 11 valid 1

11 0110 = 0x36 → 0xCB

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	3D 5C 7C 9C	0x38–3B	DC 0C DC 0C
0x1C–1F	AC BC 13 57	0x3C–3F	EC 0C EC 0C

0x76 = 110 111 0110

PTE 1 address:

0x10 + $110_{TWO} \times 2$ = 0x1C

PTE 1:

AC BC → PPN 10 valid 1

PTE 2 address:

$10\ 0000_{TWO}$ + $111_{TWO} \times 2$ = 0x2E

(100000_{TWO} = *10*×page size)

PTE 2:

EF F0 → PPN 11 valid 1

11 0110 = 0x36 → 0xCB

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	3D 5C 7C 9C	0x38–3B	DC 0C DC 0C
0x1C–1F	AC BC 13 57	0x3C–3F	EC 0C EC 0C

0x76 = 110 *111 0110*

PTE 1 address:

$0x10 + 110_{TWO} \times 2 = 0x1C$

PTE 1:

AC BC → PPN 10 valid 1

PTE 2 address:

$10\ 0000_{TWO} + 111_{TWO} \times 2 =$ 0x2E

$(100000_{TWO} = 10 \times \text{page size})$

PTE 2:

EF F0 → PPN 11 valid 1

11 *0110* = 0x36 → 0xCB

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	0x34–37	CB 0B CB 0B
0x18–1B	3D 5C 7C 9C	0x38–3B	DC 0C DC 0C
0x1C–1F	AC BC 13 57	0x3C–3F	EC 0C EC 0C

0x76 = 110 111 0110

PTE 1 address:

0x10 + $110_{TWO} \times 2$ = 0x1C

PTE 1:

AC BC → PPN 10 valid 1

PTE 2 address:

$10\ 0000_{TWO}$ + $111_{TWO} \times 2$ = 0x2E

(100000_{TWO} = 10×page size)

PTE 2:

EF F0 → PPN 11 valid 1

11 0110 = *0x36* → 0xCB

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 address	bytes	physical address	bytes
0x00–03	00 11 22 33	0x20–23	D0 D1 D2 D3
0x04–07	44 55 66 77	0x24–24	D4 D5 D6 D7
0x08–0B	88 99 AA BB	0x28–2B	89 9A AB BC
0x0C–0F	CC DD EE FF	0x2C–2F	CD DE EF F0
0x10–13	1A 2A 3A 4A	0x30–33	BA 0A BA 0A
0x14–17	1B 2B 3B 4B	*0x34–37*	CB 0B *CB* 0B
0x18–1B	3D 5C 7C 9C	0x38–3B	DC 0C DC 0C
0x1C–1F	AC BC 13 57	0x3C–3F	EC 0C EC 0C

0x76 = 110 111 0110

PTE 1 address:

0x10 + $110_{\text{TWO}} \times 2$ = 0x1C

PTE 1:

AC BC → PPN 10 valid 1

PTE 2 address:

$10\ 0000_{\text{TWO}}$ + $111_{\text{TWO}} \times 2$ = 0x2E

(100000_{TWO} = 10×page size)

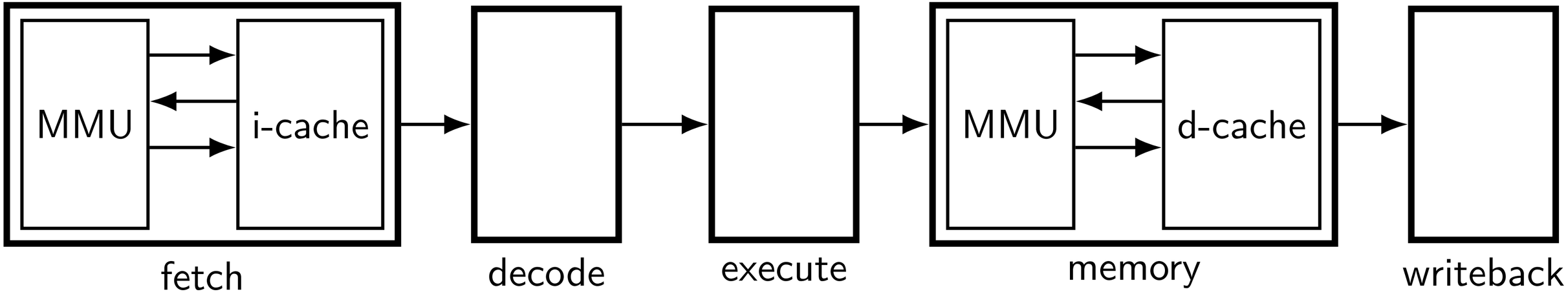
PTE 2:

EF F0 → PPN 11 valid 1

11 0110 = 0x36 → 0xCB

Backup slides

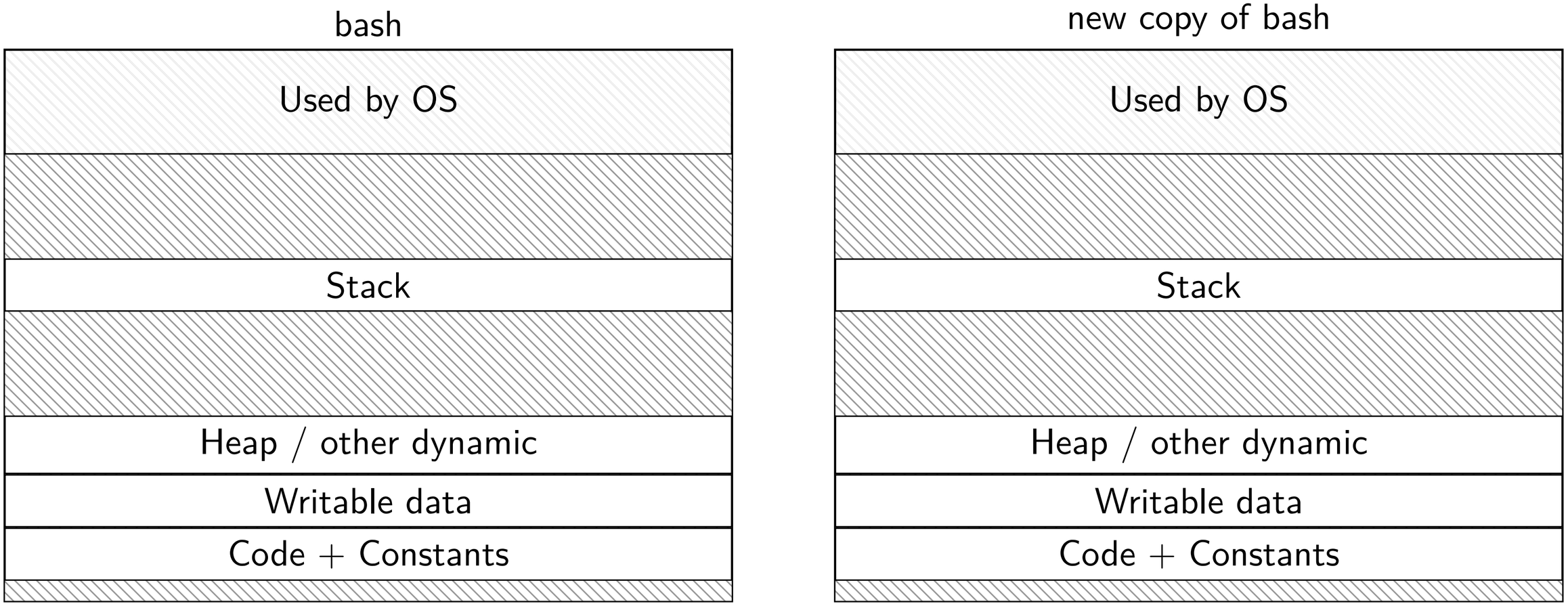
MMUs in the pipeline



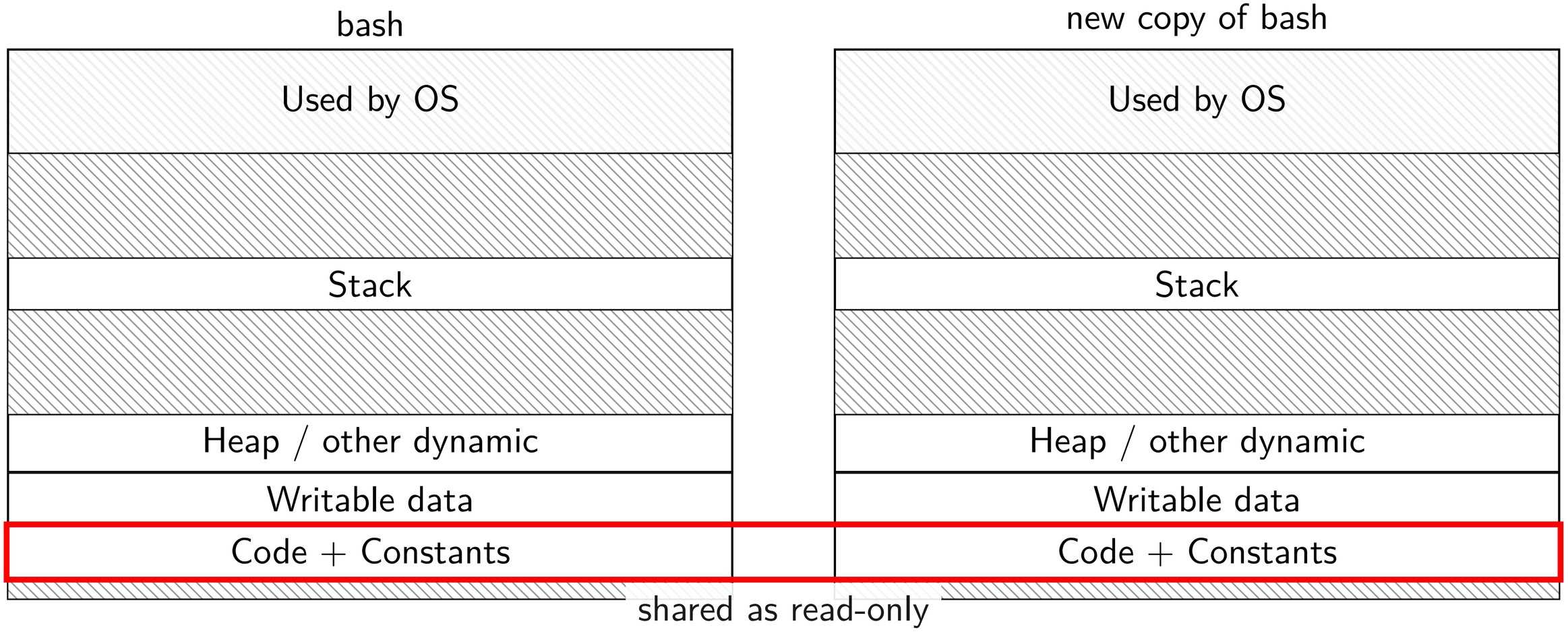
up to four memory accesses per instruction

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

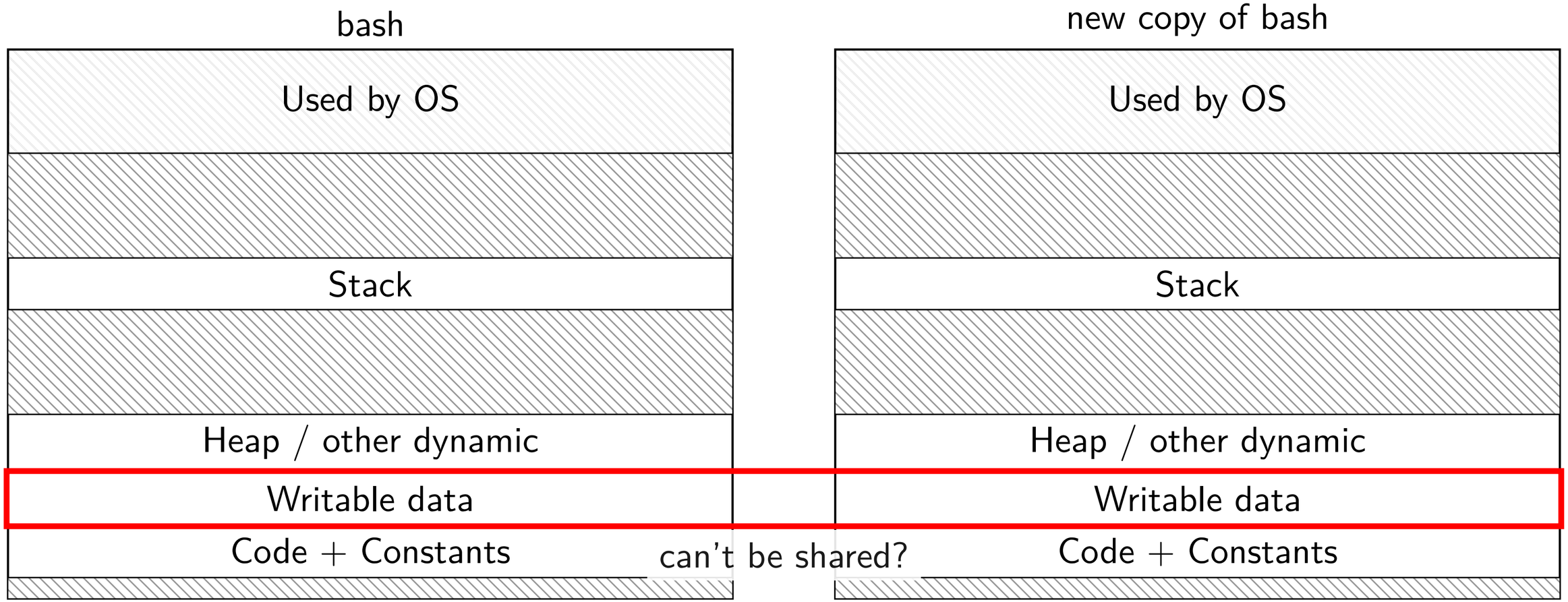
do we really need a complete copy?



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 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
			0x200DF
			0x200AF
			…

copy operation actually duplicates page table

both processes *share all physical pages*

but marked *read-only in both tables*

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 a read-only page triggers a page fault — OS actually copies the page

copy-on-write and page tables

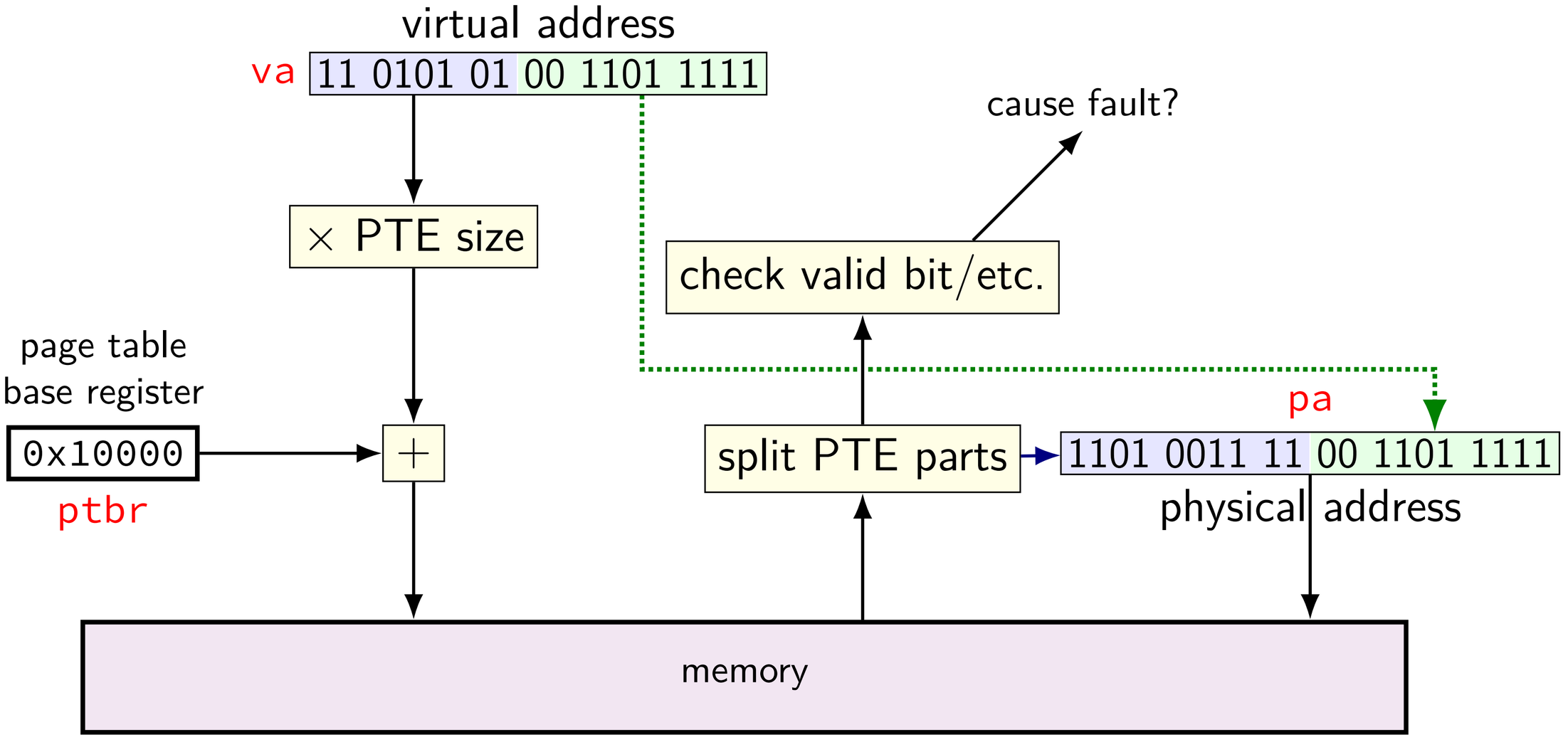
VPN	valid?	write?	physical page
…	…	…	…
0x00601	1	0	0x12345
0x00602	1	0	0x12347
0x00603	1	0	0x12340
0x00604	1	0	0x200DF
0x00605	1	0	0x200AF
…	…	…	…

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
…	…	…	…

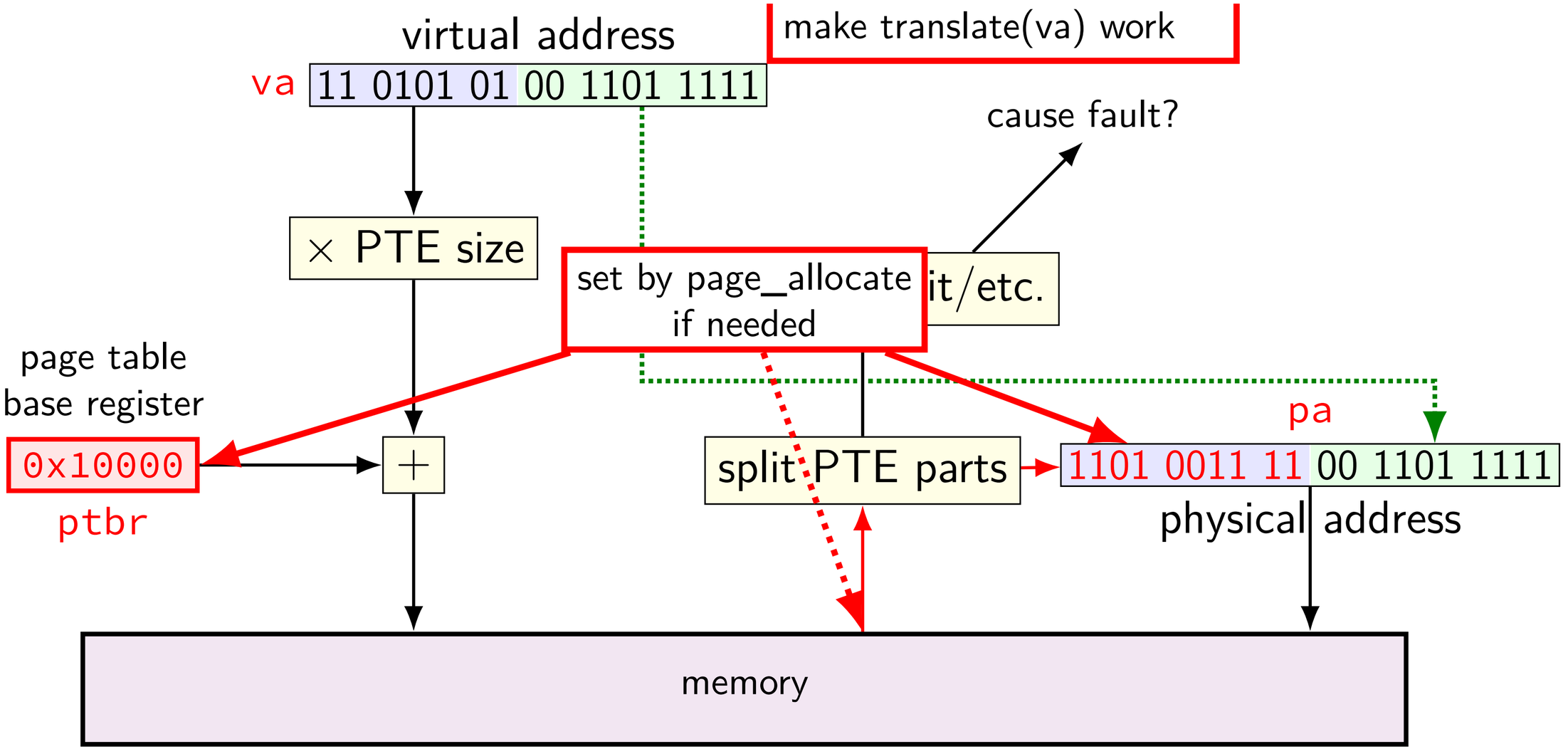
after allocating a copy, OS reruns the write instruction

pa=translate(va)

pa=translate(va)



pa=translate(va)



swapping

early motivation for virtual memory: *swapping*

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

- how our textbook and many other sources presents virtual memory

OS allocates *program space on disk*

- own mapping of virtual addresses to location on disk

DRAM is a cache for disk

swapping

early motivation for virtual memory: *swapping*

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

- how our textbook and many other sources presents virtual memory

OS allocates *program space on disk*

- own mapping of virtual addresses to location on disk

DRAM is a cache for disk

swapping components

"swap in" a page — exactly like allocating on demand!

- OS gets page fault — invalid in page table
- check where page actually is (from virtual address)
- read from disk
- eventually restart process

"swap out" a page

- OS marks as invalid in the page table(s)
- copy to disk (if modified)

HDD/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)

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*

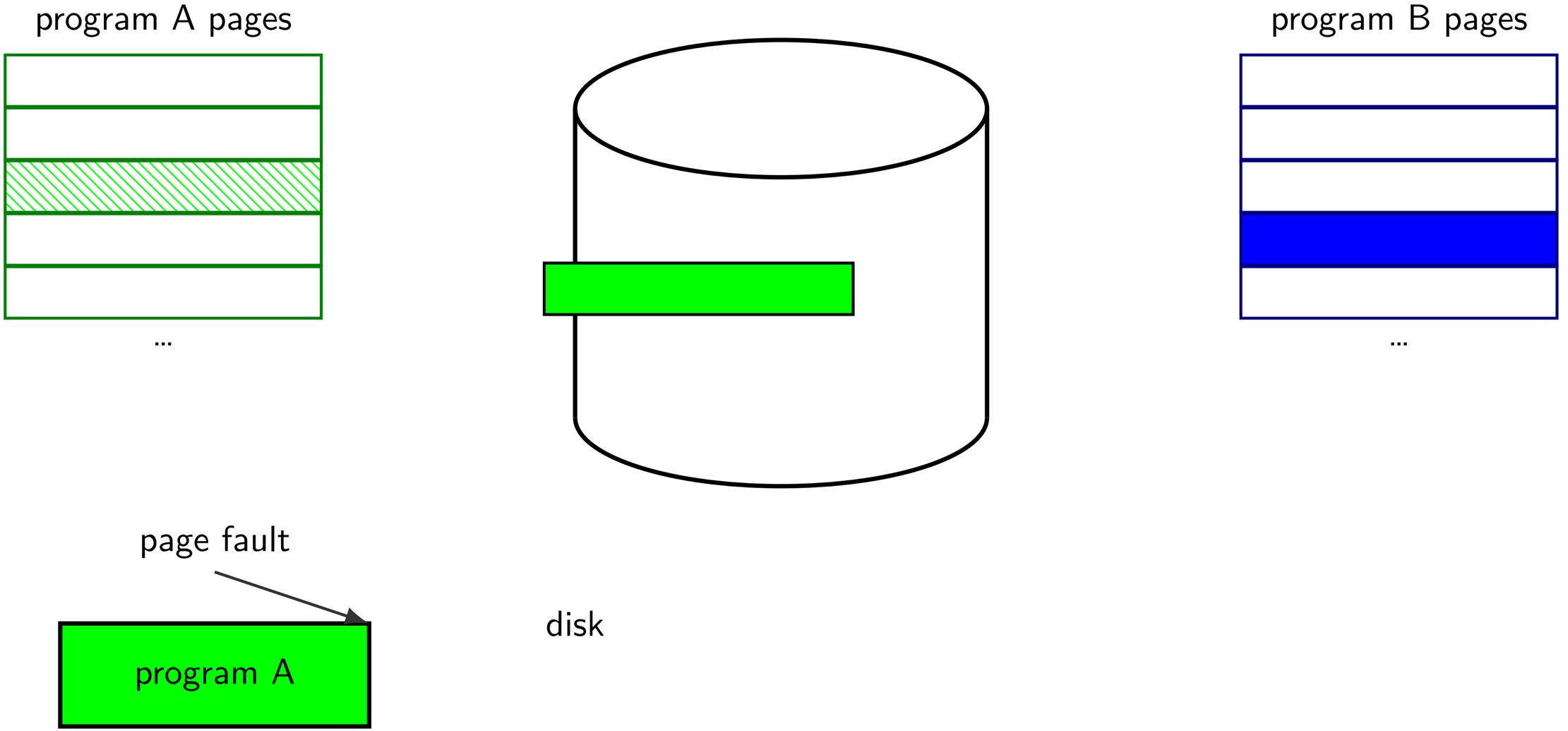
- minimum size: 512 bytes
- writing tens of kilobytes basically as fast as writing 512 bytes

SSD writes and writes: *hundreds of microseconds*

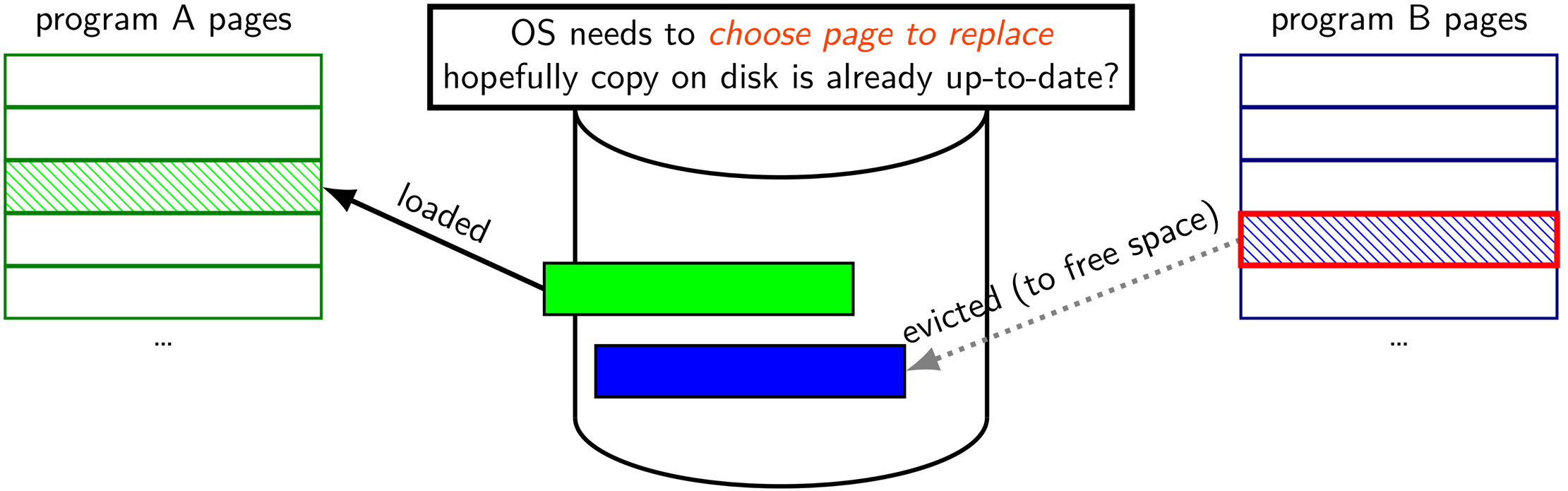
- designed for writes/reads of kilobytes (not much smaller)

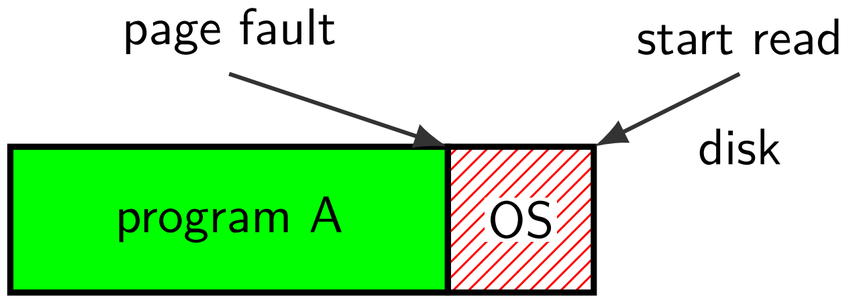
swapping timeline

swapping timeline

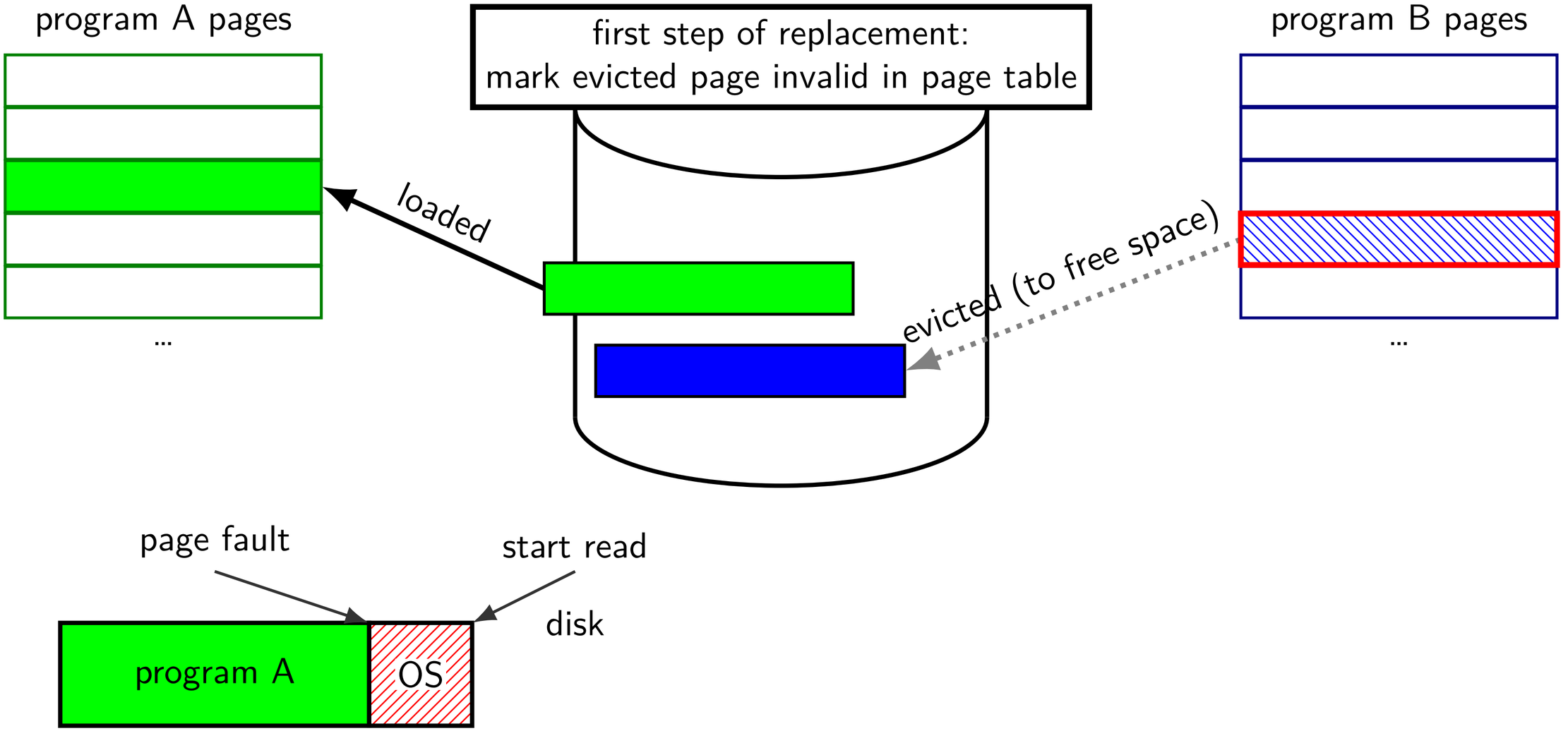


swapping timeline

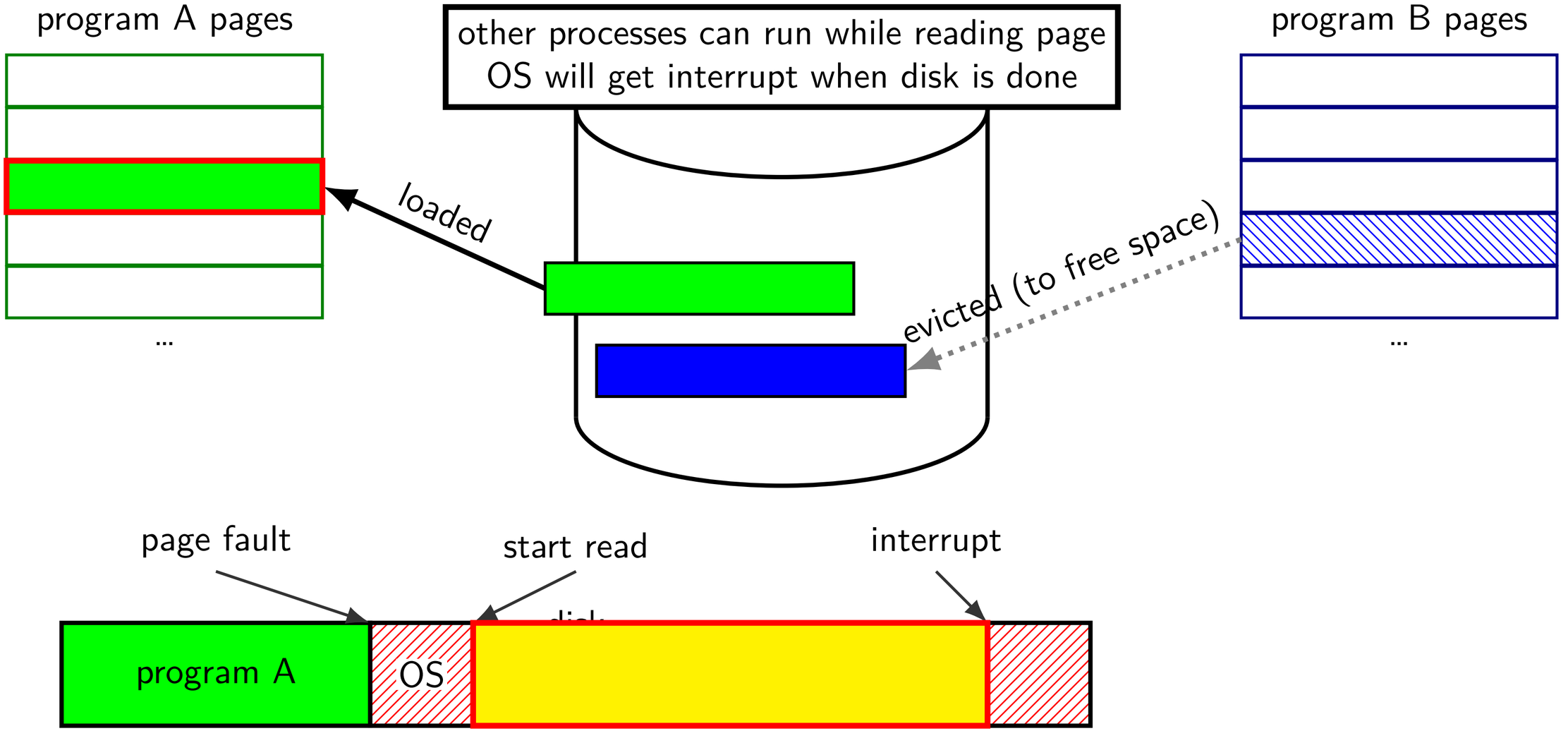




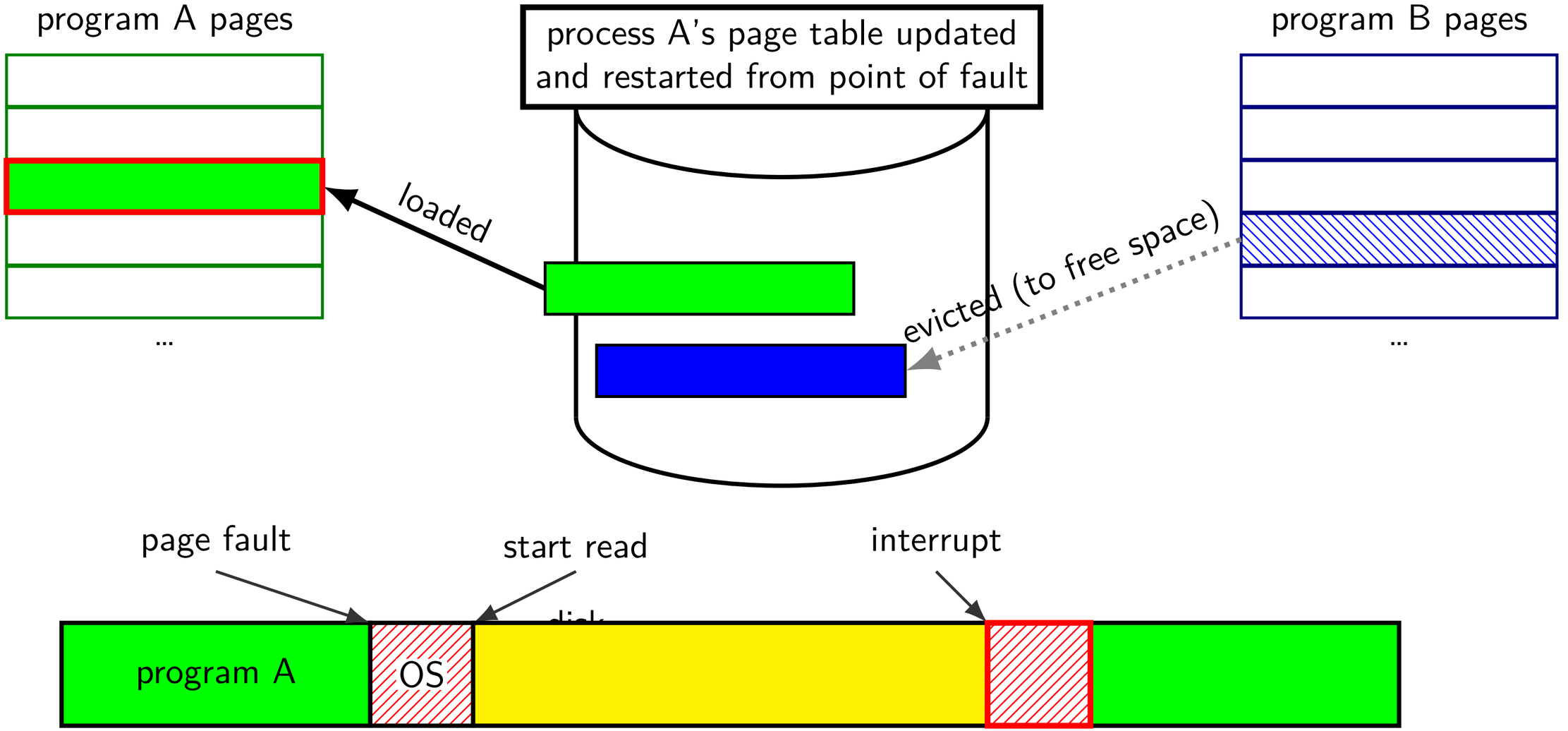
swapping timeline



swapping timeline



swapping timeline



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*

exercise

40-bit physical addresses, 36-bit virtual addresses, 2^{14} byte (16384 byte) pages

Q1: number of virtual pages?

Q2: size of physical page numbers?

exercise: page table lookup (2)

suppose 32-byte pages (= 5-bit page offsets), 8-bit virtual addresses

VPN	valid	PPN
`000`	`1`	`10`
`001`	`1`	`01`
`010`	`0`	—
`011`	`0`	—
`100`	`1`	`00`
`101`	`0`	—
`110`	`0`	—
`111`	`0`	—

virtual address `0x97` = ???