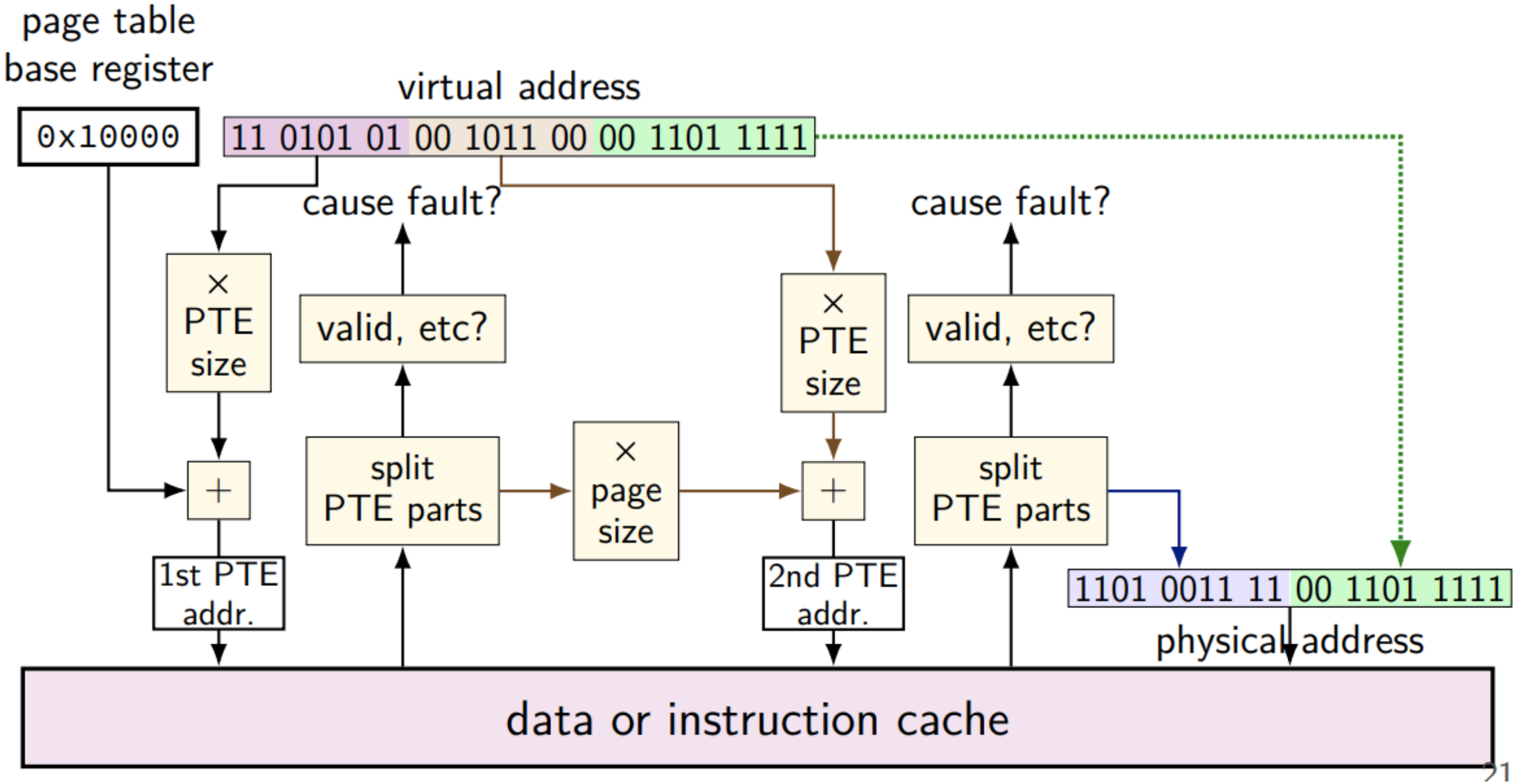
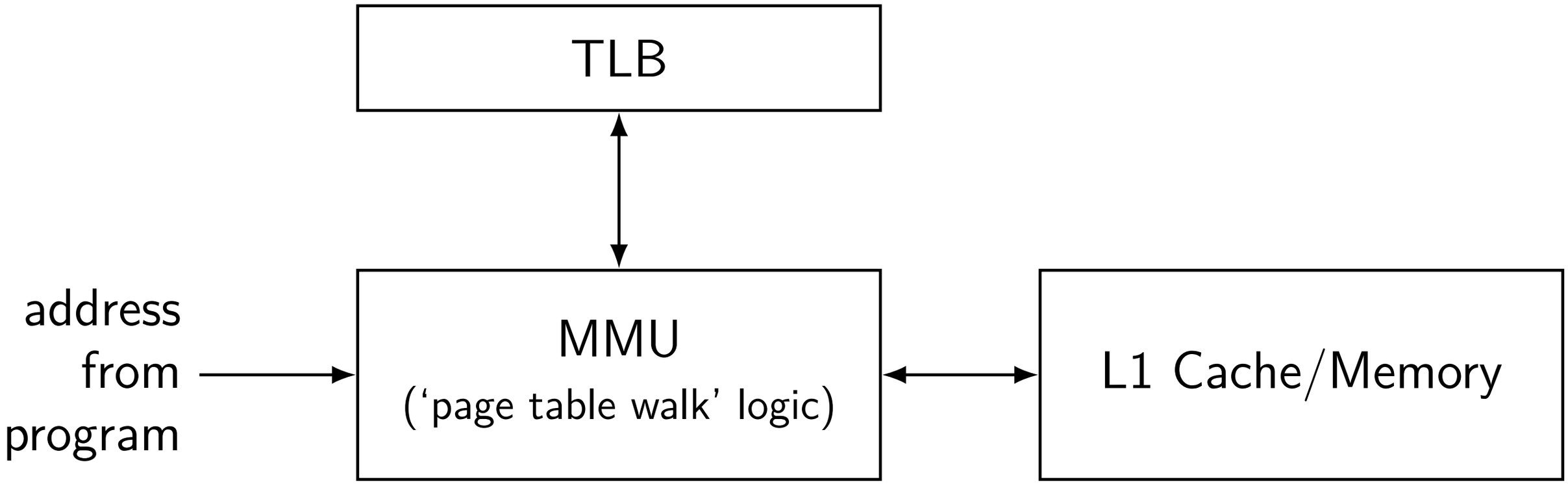
virtual memory 3

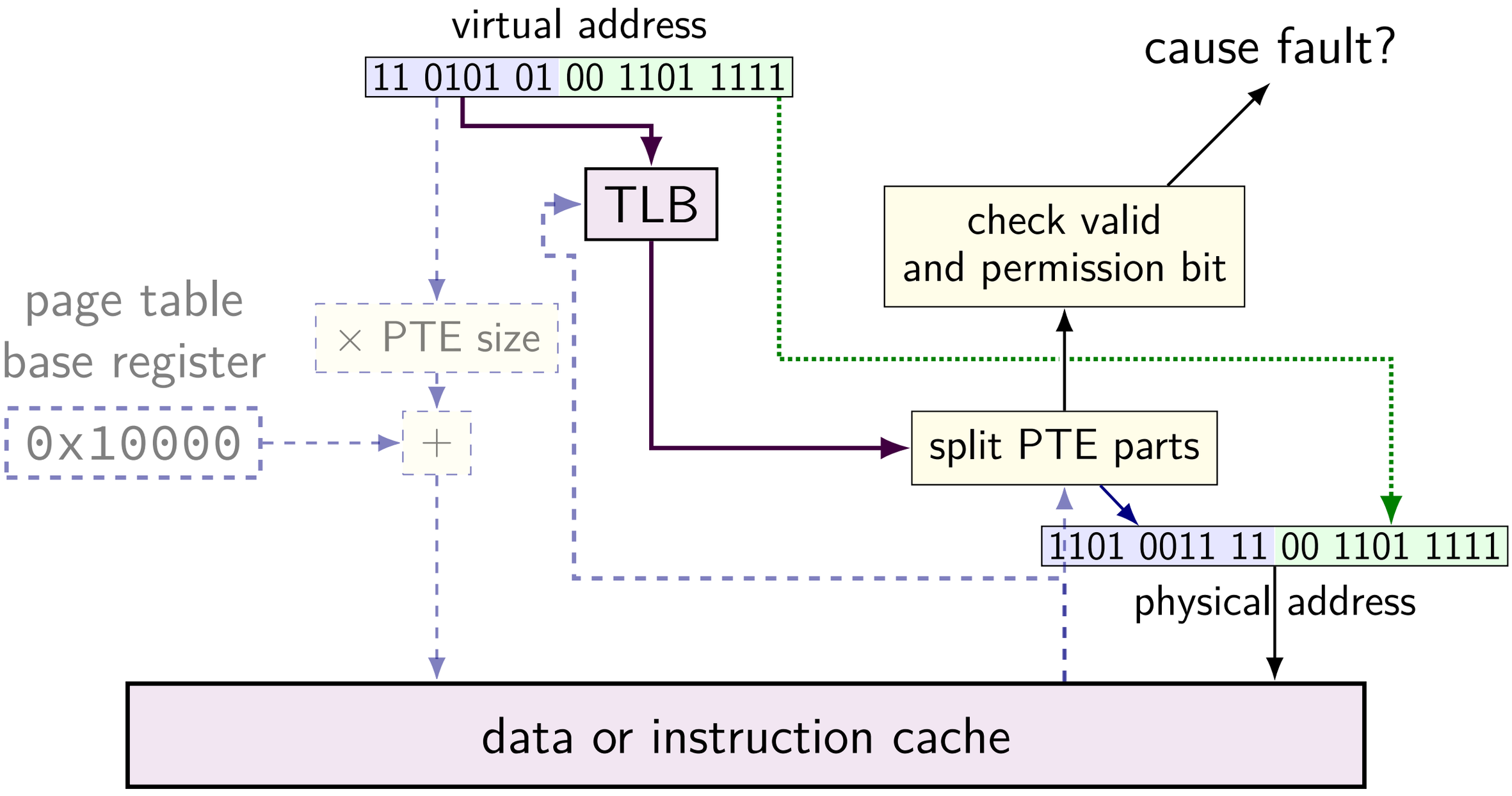
two-level page table lookup



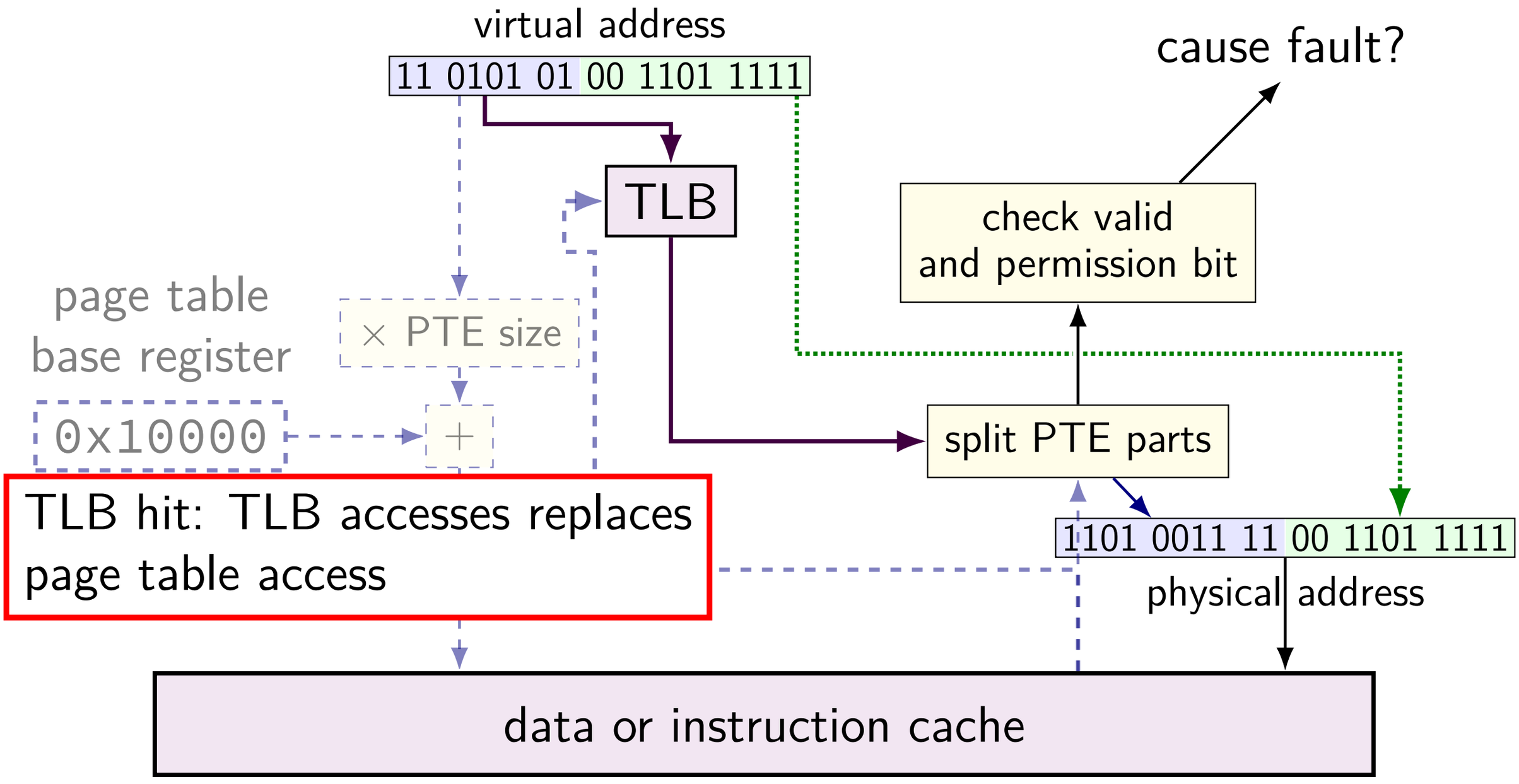
TLB and the MMU (1)



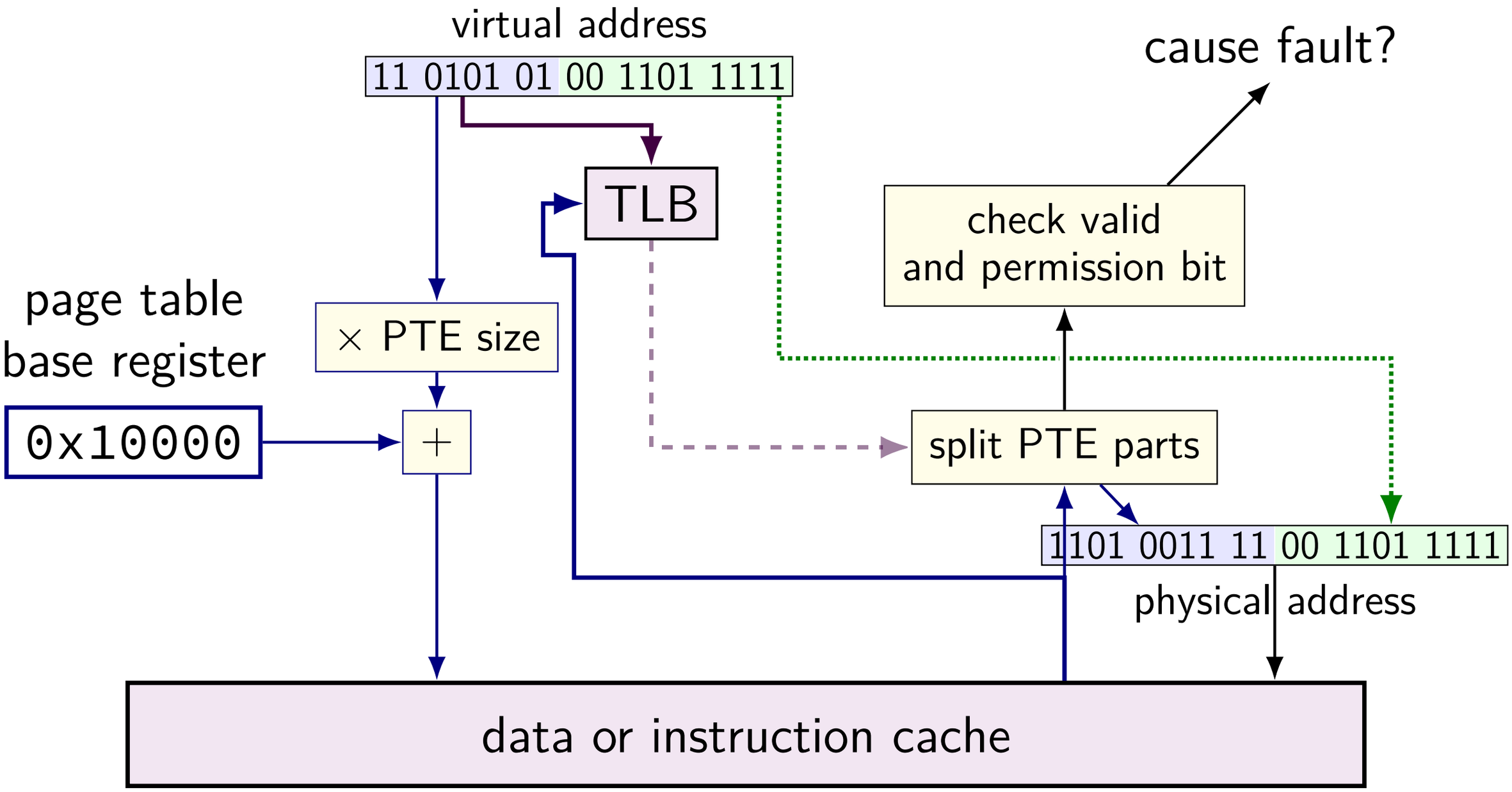
TLB and the MMU (2)



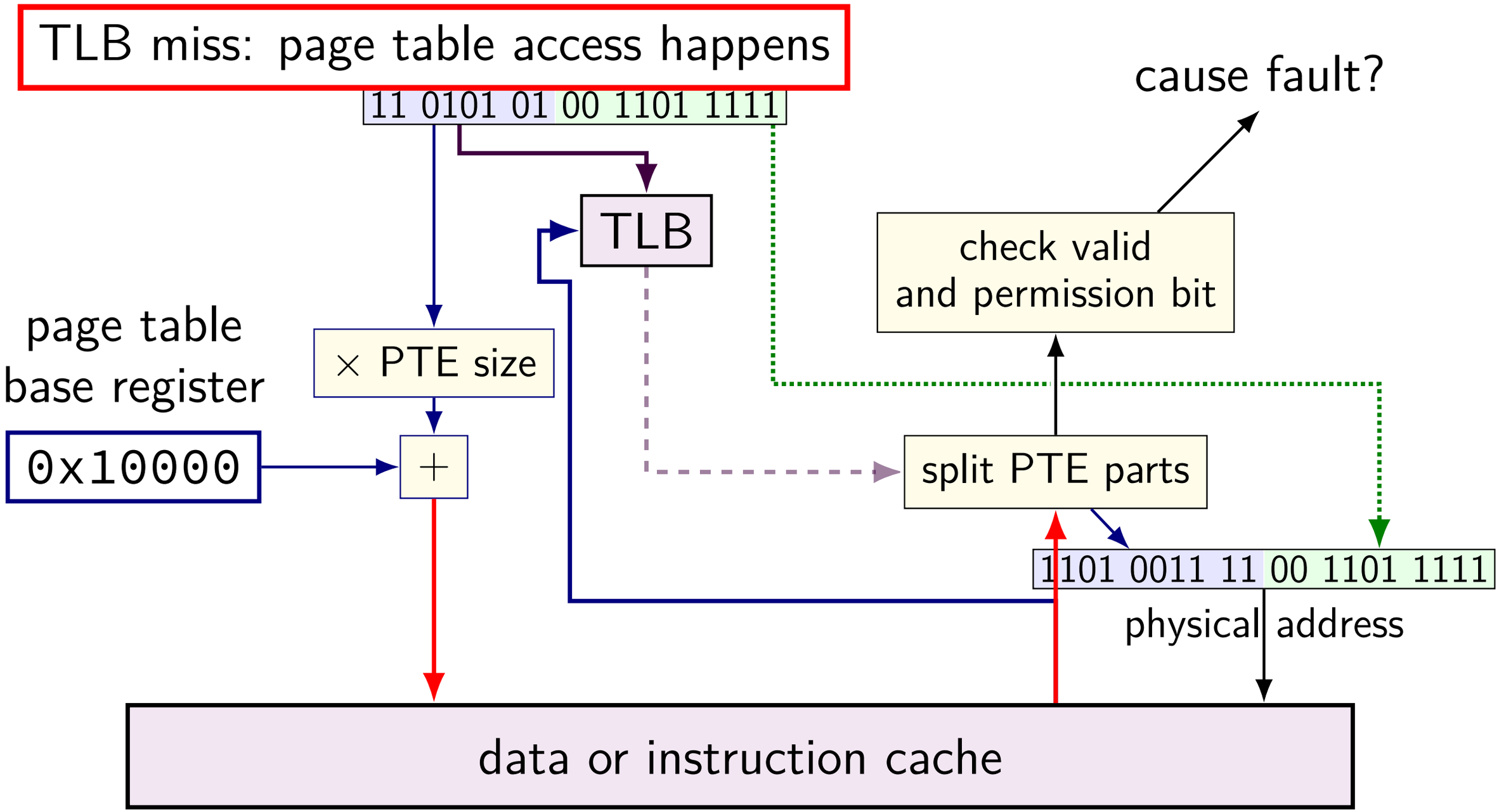
TLB and the MMU (2)



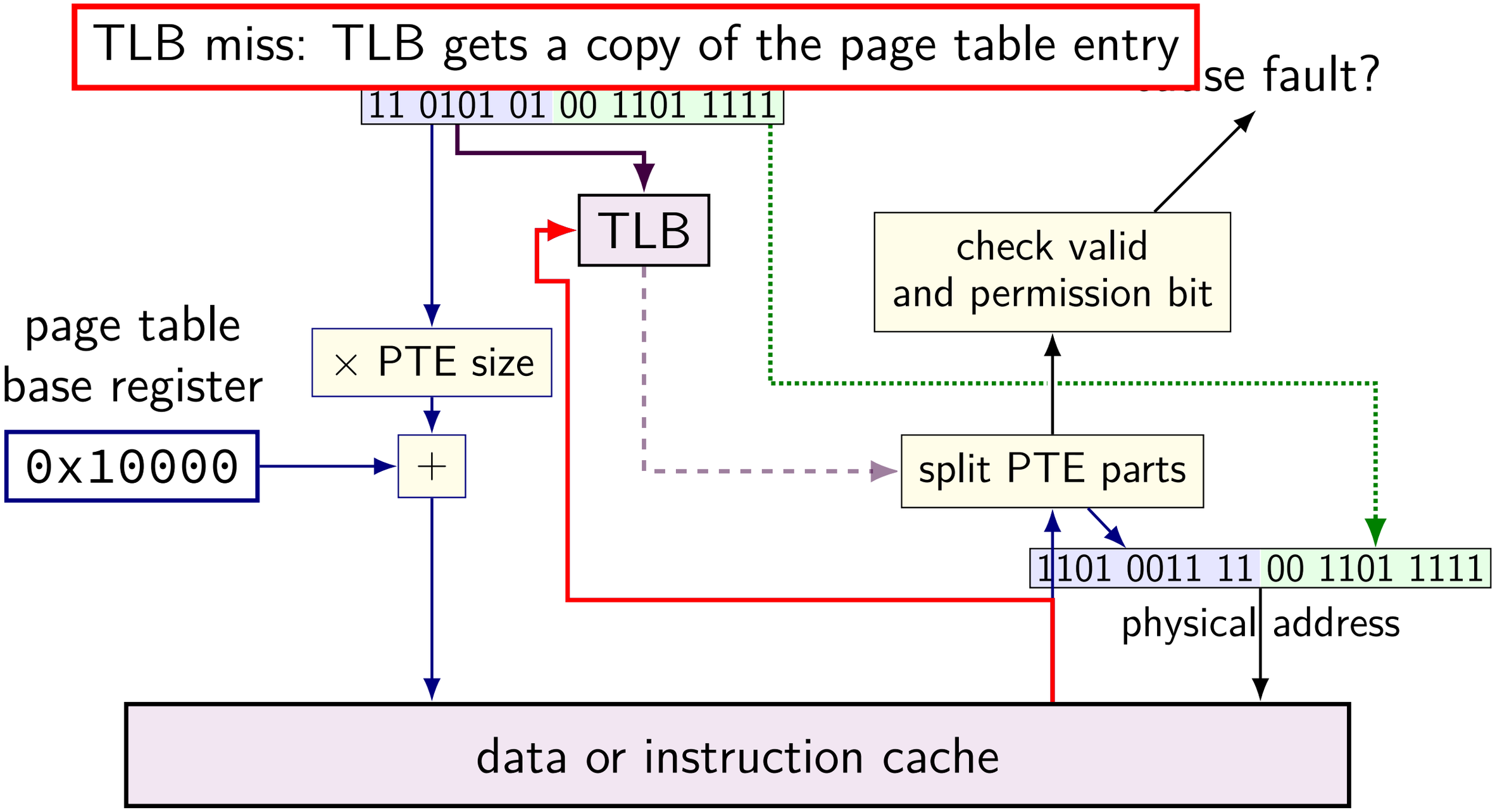
TLB and the MMU (2)



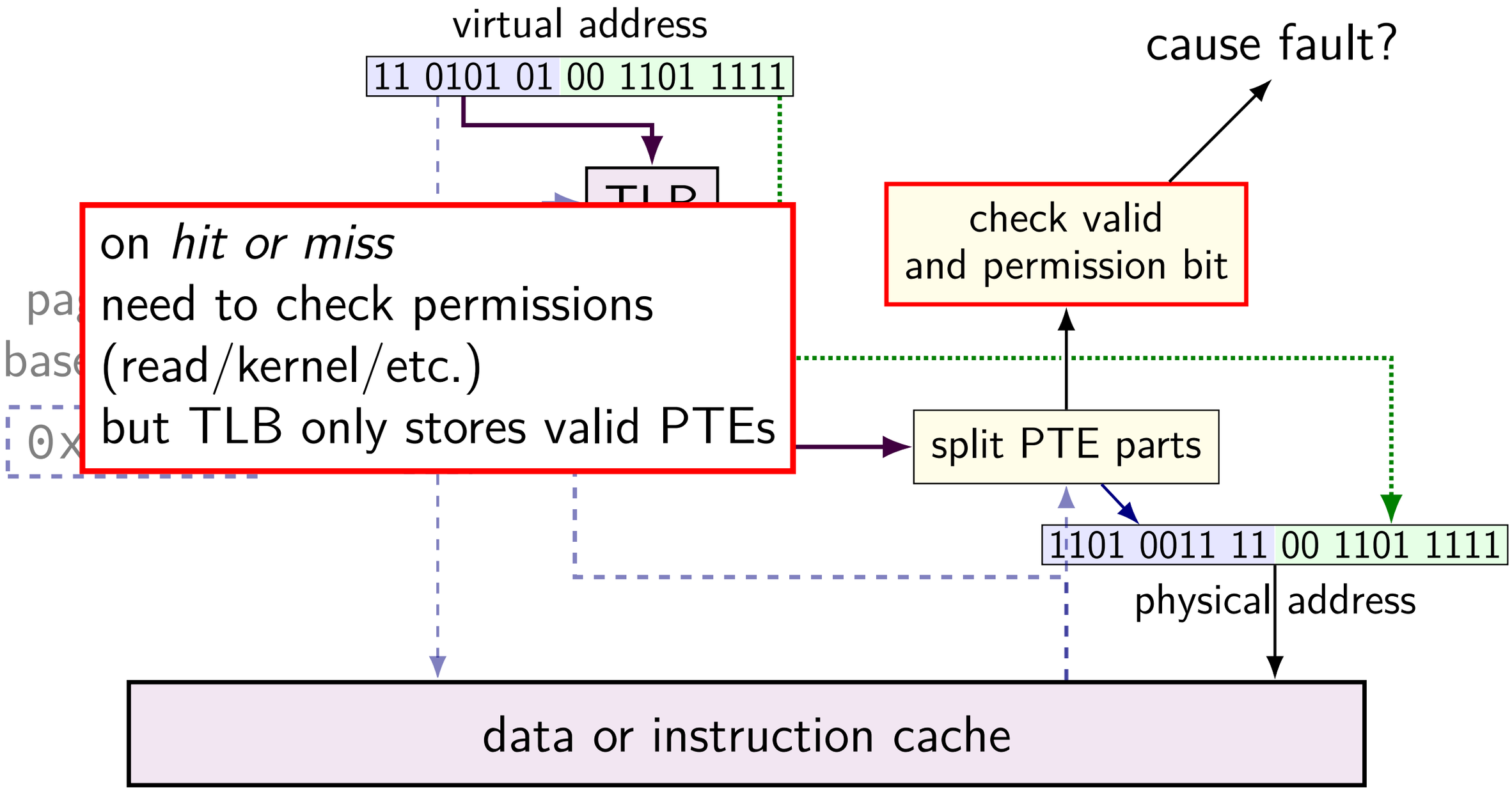
TLB and the MMU (2)



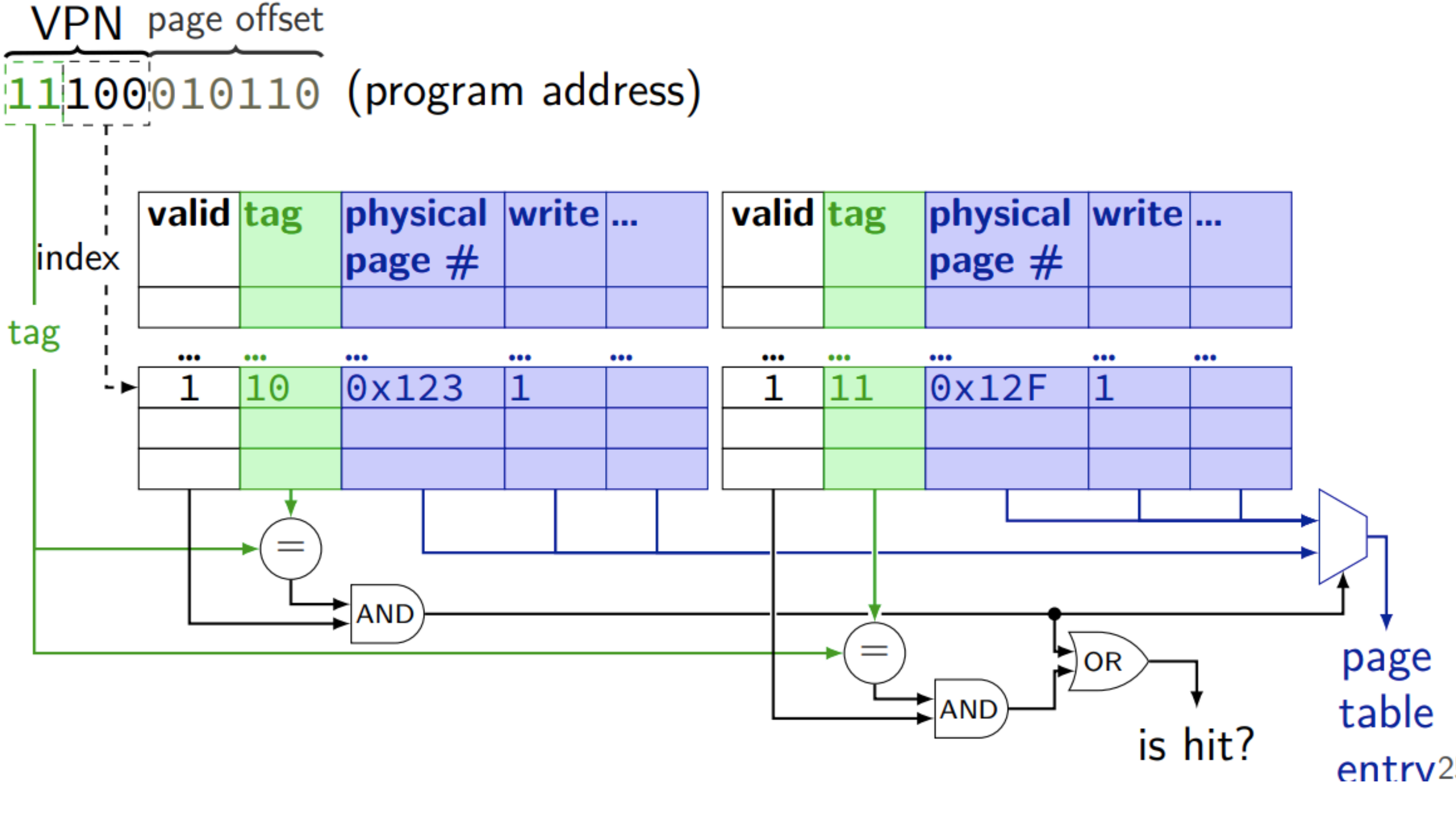
TLB and the MMU (2)



TLB and the MMU (2)



TLB organization (2-way set associative)



exercise: TLB access pattern (setup)

4-entry, 2-way TLB, LRU replacement policy, initially empty

4096 byte pages

how many index bits?

TLB index of virtual address 0x12345?

exercise: TLB access pattern

4-entry, 2-way TLB, LRU replacement policy, initially empty

4096 byte pages

type	virtual	physical
read	0x440030	0x554030
write	0x440034	0x554034
read	0x7FFFE008	0x556008
read	0x7FFFE000	0x556000
read	0x7FFFDFF8	0x5F8FF8
read	0x664080	0x5F9080
read	0x440038	0x554038
write	0x7FFFDFF0	0x5F8FF0

which are TLB hits? which are TLB misses? final contents of TLB?

exercise: TLB access pattern

4-entry, 2-way TLB, LRU replacement policy, initially empty

4096 byte pages

				VPNs of PTEs held in TLB	
type	virtual	physical	result	set 0	set 1
`read`	`0x440030`	`0x554030`	miss	0x440	
`write`	`0x440034`	`0x554034`	hit	0x440	
`read`	`0x7FFFE008`	`0x556008`	miss	0x440	
`read`	`0x7FFFE000`	`0x556000`	hit	0x440, 0x7FFFE	
`read`	`0x7FFFDFF8`	`0x5F8FF8`	miss	0x440, 0x7FFFE	0x7FFFD
`read`	`0x664080`	`0x5F9080`	miss	0x664, 0x7FFFE	0x7FFFD
`read`	`0x440038`	`0x554038`	miss	0x664, 0x440	0x7FFFD
`write`	`0x7FFFDFF0`	`0x5F8FF0`	hit	0x440, 0x7FFFE	0x7FFFD

which are TLB hits? which are TLB misses? final contents of TLB?

exercise: TLB access pattern

4-entry, 2-way TLB, LRU replacement policy, initially empty

4096 byte pages

type					
read	0x664080	0x5F9080	miss	0x664, 0x7FFE	0x7FFD
read	0x440038	0x554038	miss	0x664, 0x440	0x7FFD
write	0x7FFFDFF0	0x5F8FF0	hit	0x440, 0x7FFE	0x7FFD

set idx

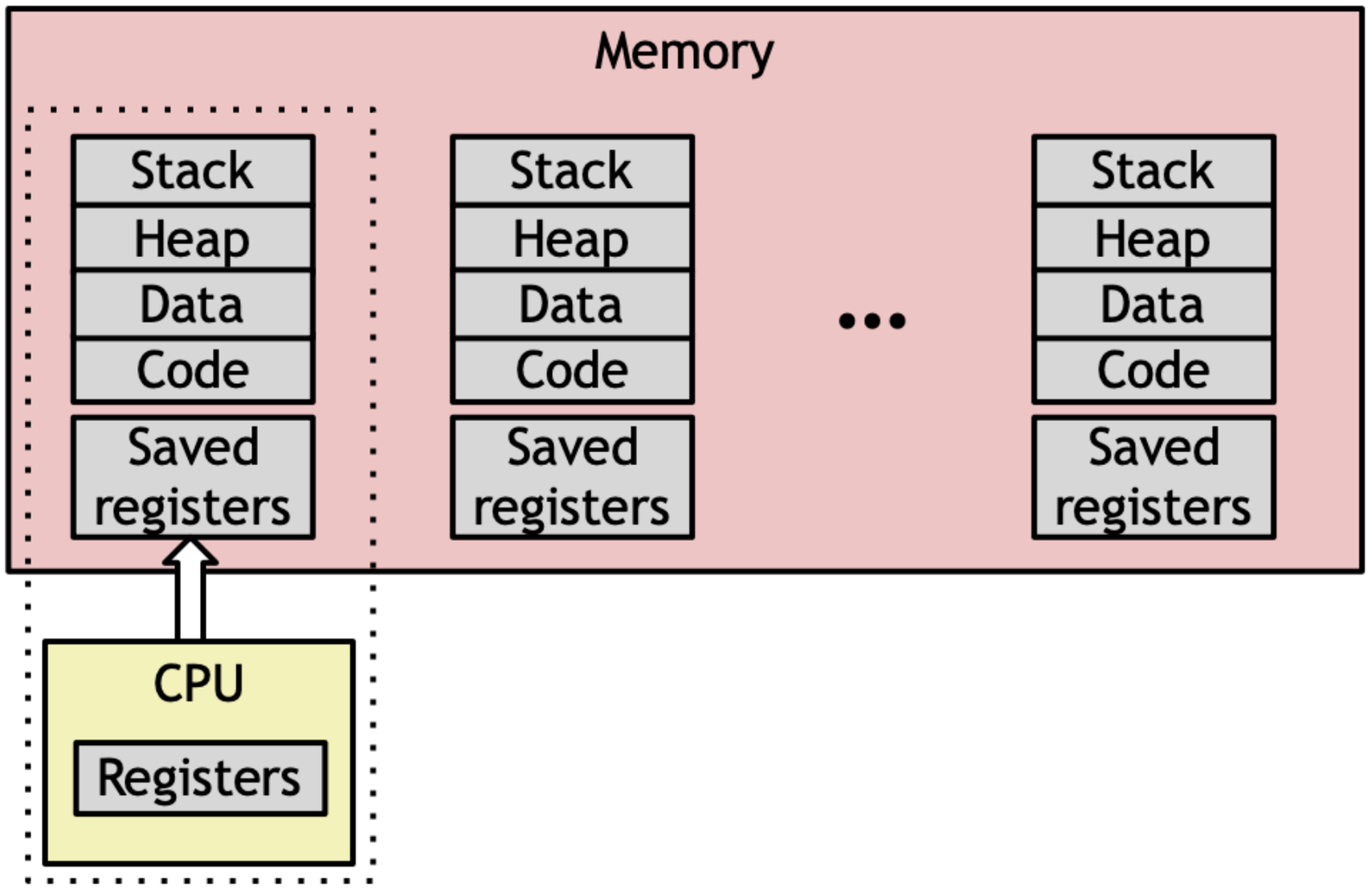
0

V	tag	physical page	write?	kernel?	...	LRU?
1	0x00220 (0x440 ≫ 1)	0x554	1	0	...	no
1	0x3FFFF (0x7FFFE ≫ 1)	0x556	1	0	...	yes

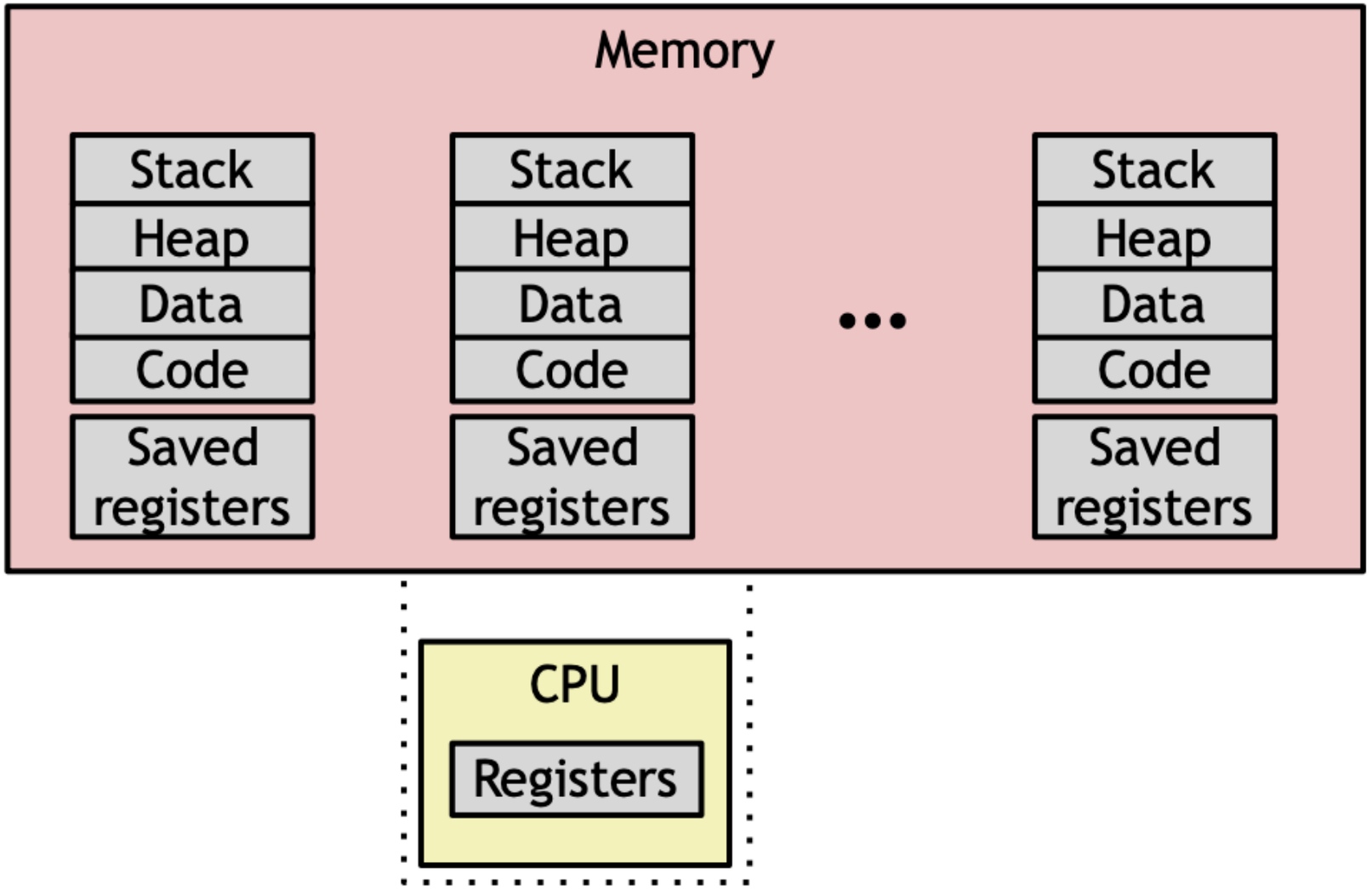
1

V	tag	physical page	write?	kernel?	...	LRU?
1	0x3FFFF (0x7FFFD ≫ 1)	0x5F9	1	0	...	no
0	---	---	-	-	...	yes

which are TLB hits? which are TLB misses? final contents of TLB?

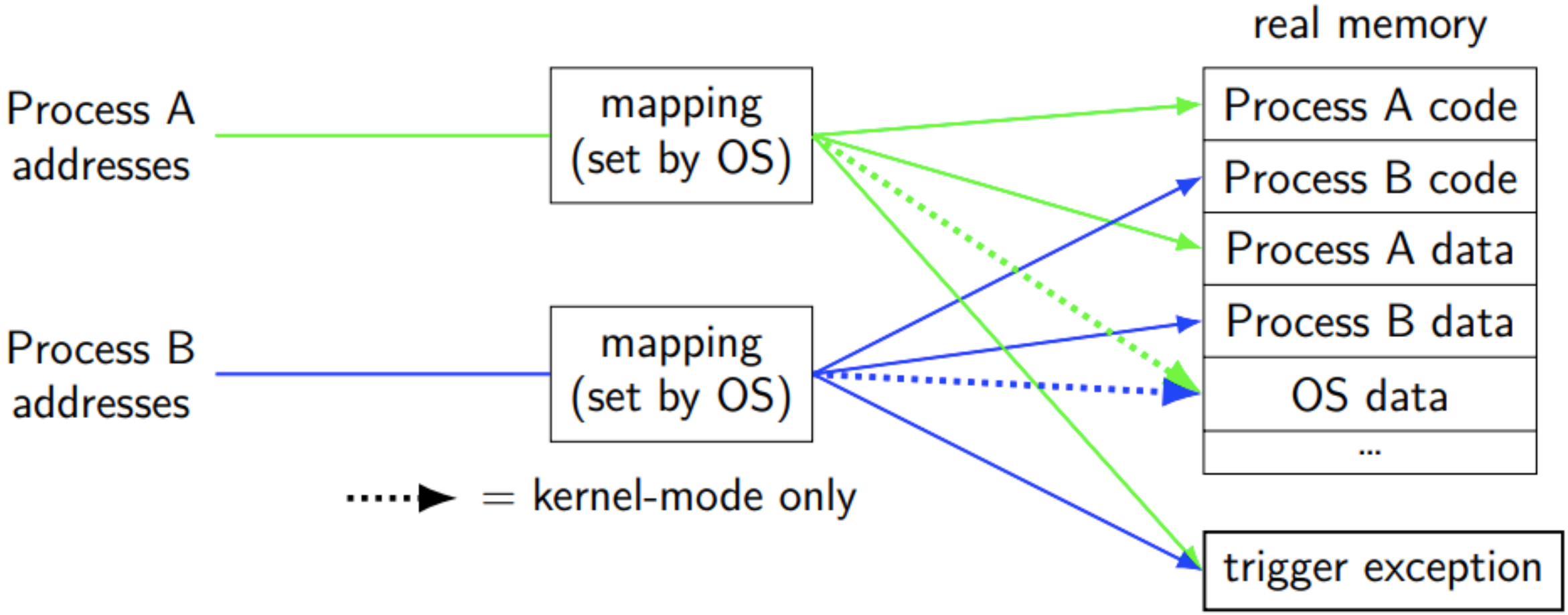


What is the state of the TLB?



What is the state of the TLB?

The TLB is going to cache the old mapping



Though virtual address 0x0 maps to different address
The TLB currently only caches a single mapping

changing page tables

what happens to TLB when page table base pointer is changed?

e.g. context switch

most entries in TLB refer to things from wrong process

oops — read from the wrong process's stack?

changing page tables

what happens to TLB when page table base pointer is changed?

e.g. context switch

most entries in TLB refer to things from wrong process

oops — read from the wrong process's stack?

option 1: invalidate all TLB entries

side effect on "change page table base register" instruction

changing page tables

what happens to TLB when page table base pointer is changed?

e.g. context switch

most entries in TLB refer to things from wrong process

oops — read from the wrong process's stack?

option 1: invalidate all TLB entries

side effect on "change page table base register" instruction

option 2: TLB entries contain process ID

set by OS (special register)
checked by TLB in addition to TLB tag, valid bit

editing page tables

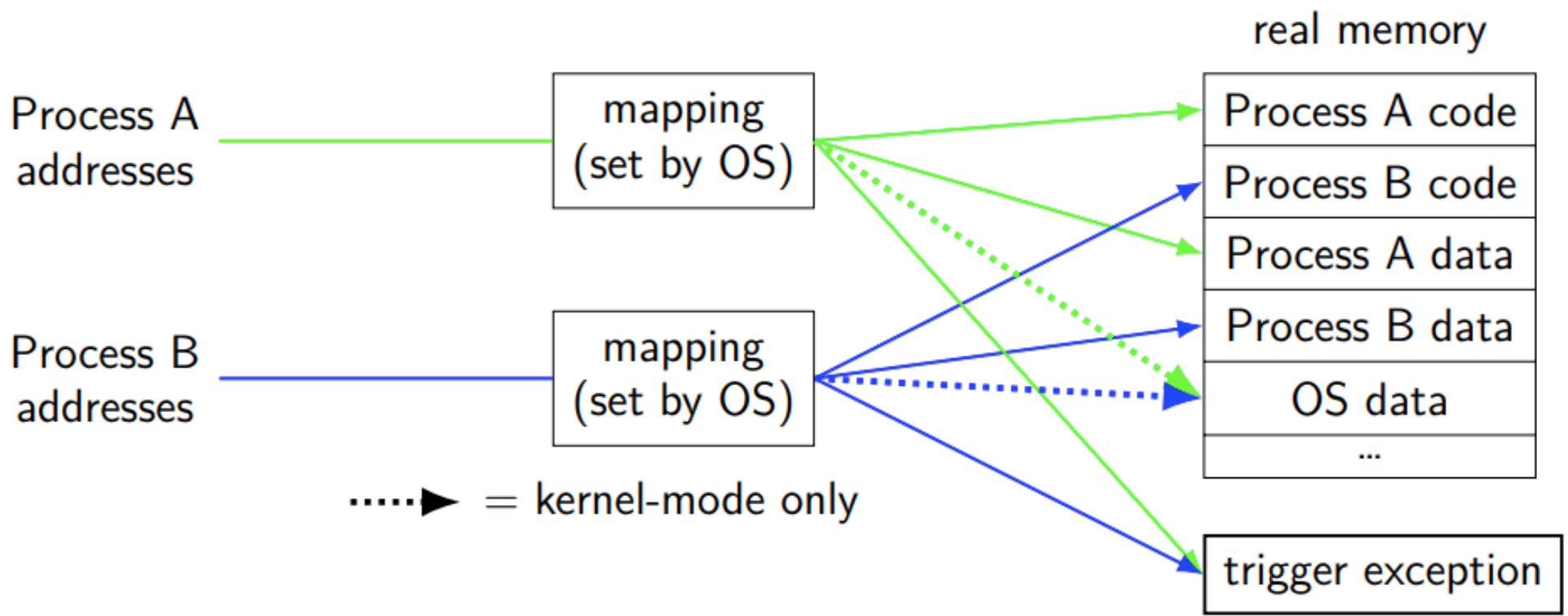
what happens to TLB when OS changes a page table entry?

invalid to valid — nothing needed

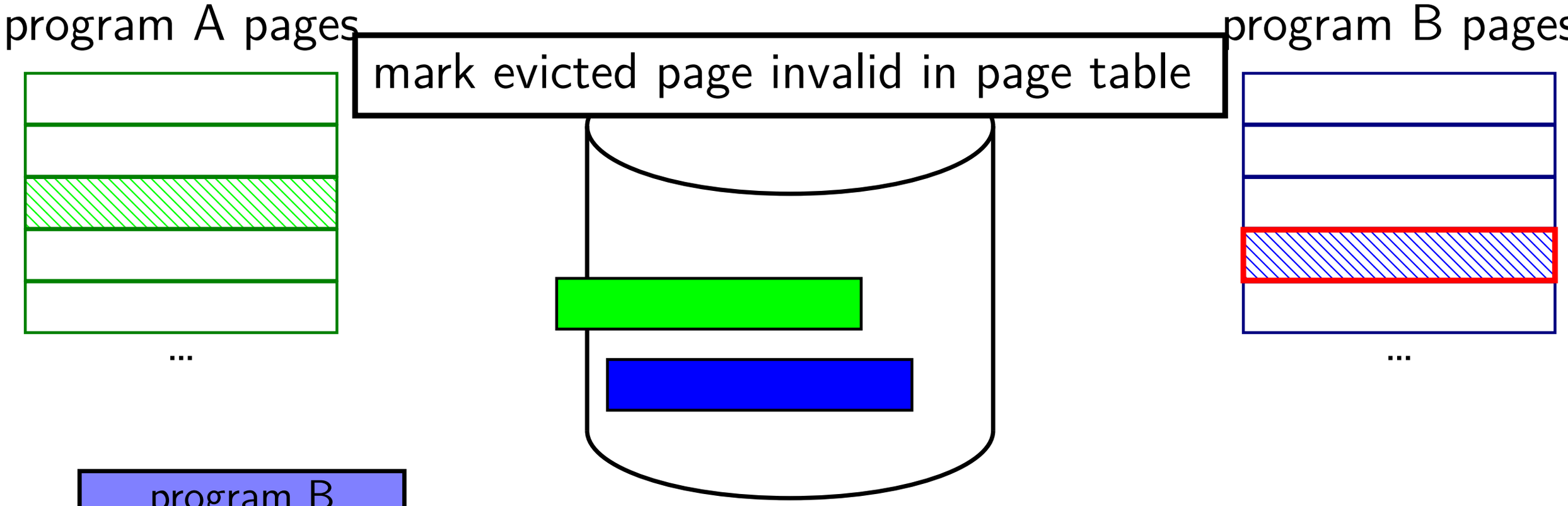
- TLB doesn't contain invalid entries
- MMU will check memory again

valid to invalid — OS needs to tell processor to invalidate it

- special instruction (x86: `invlpg`)

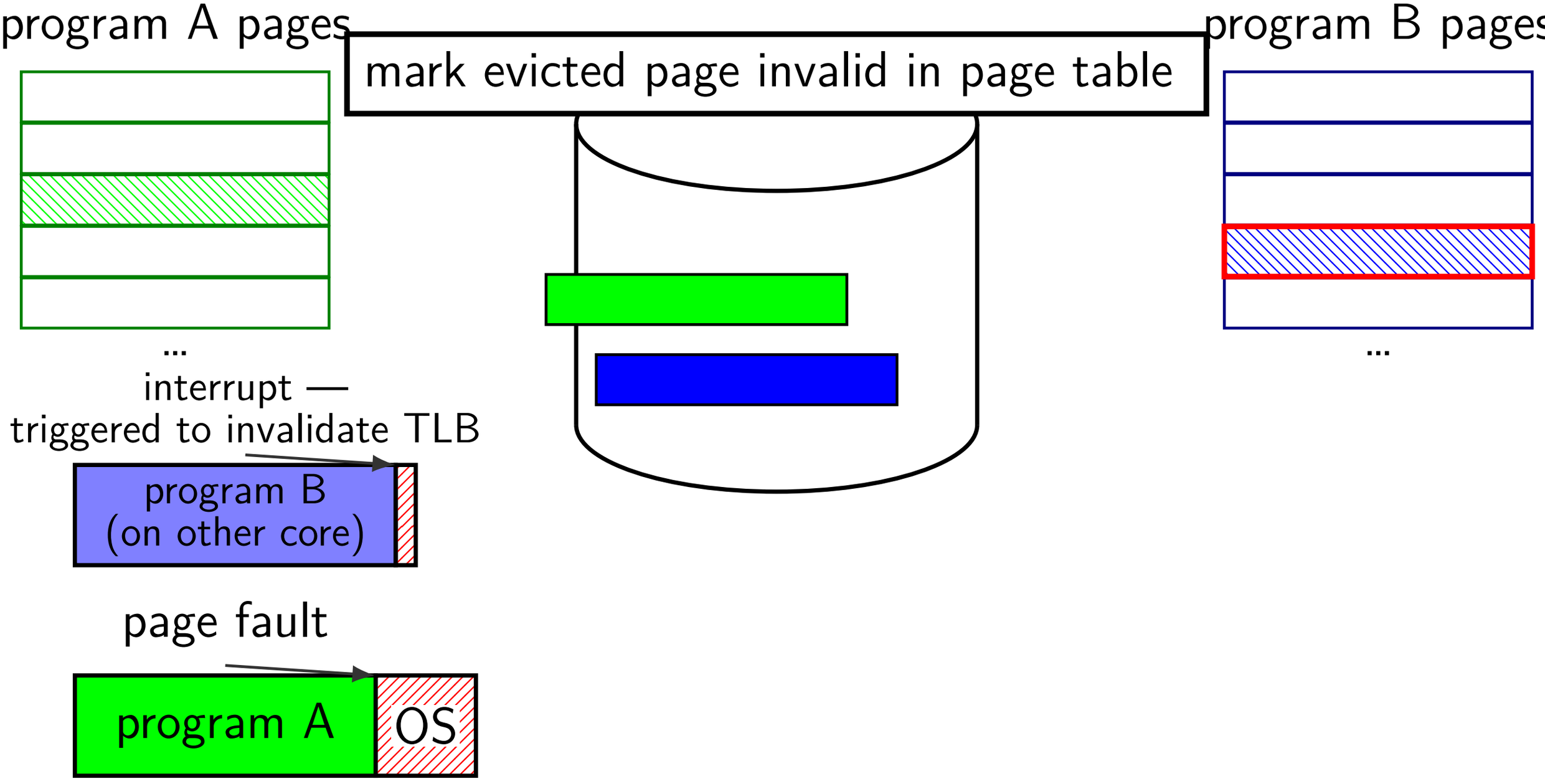


TLB shootdown

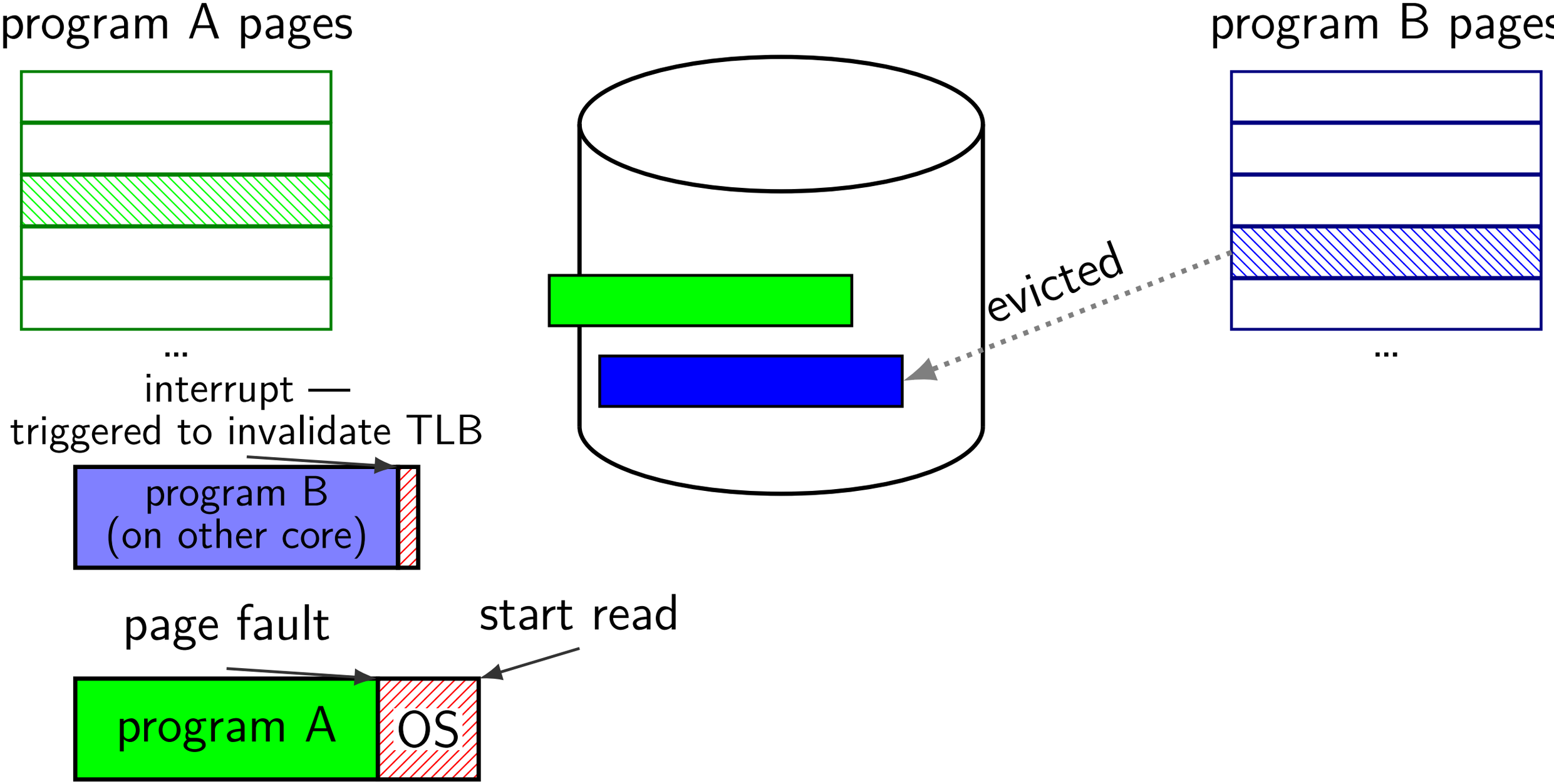


Each core has it's own TLB

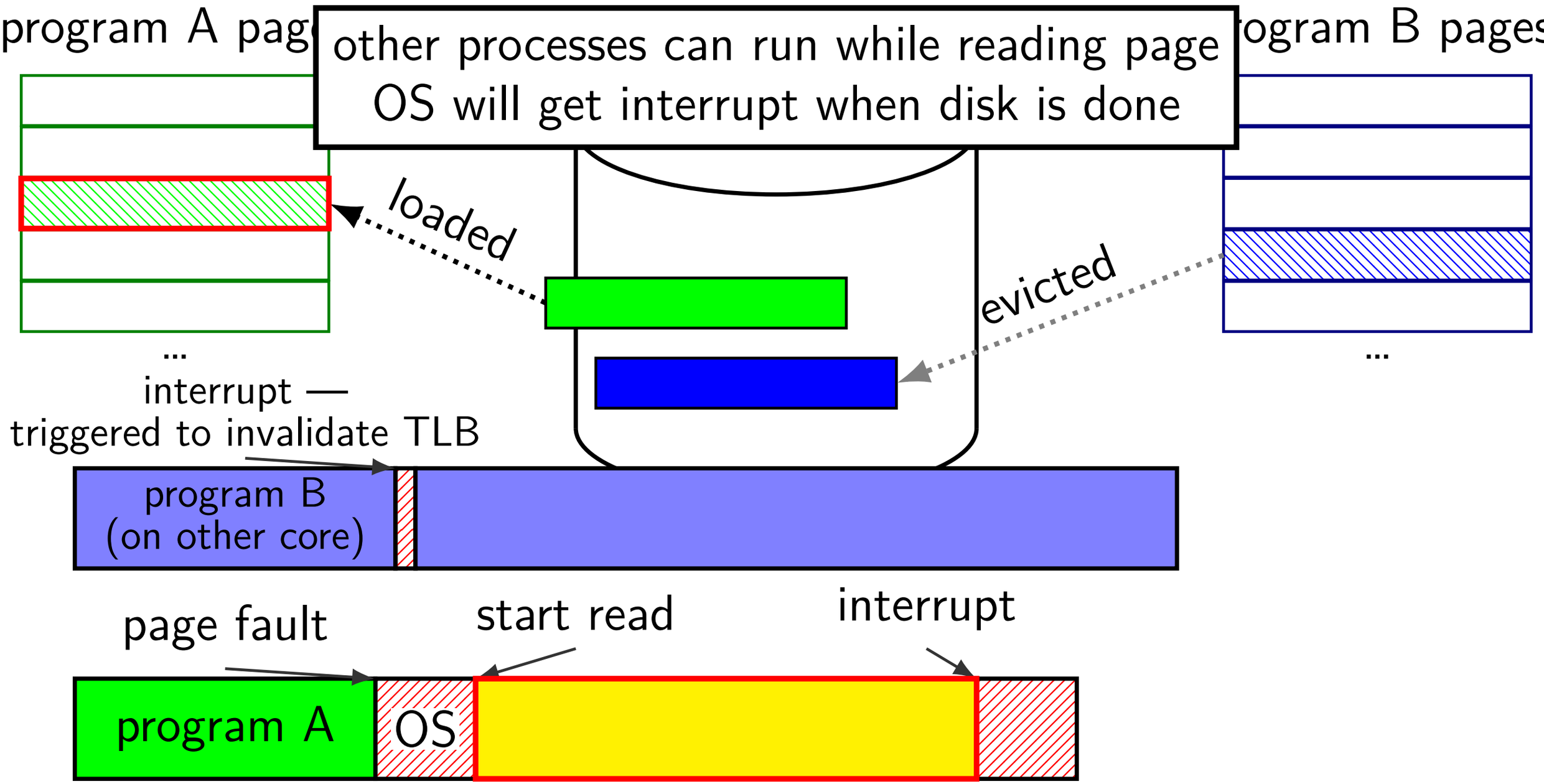
TLB shootdown



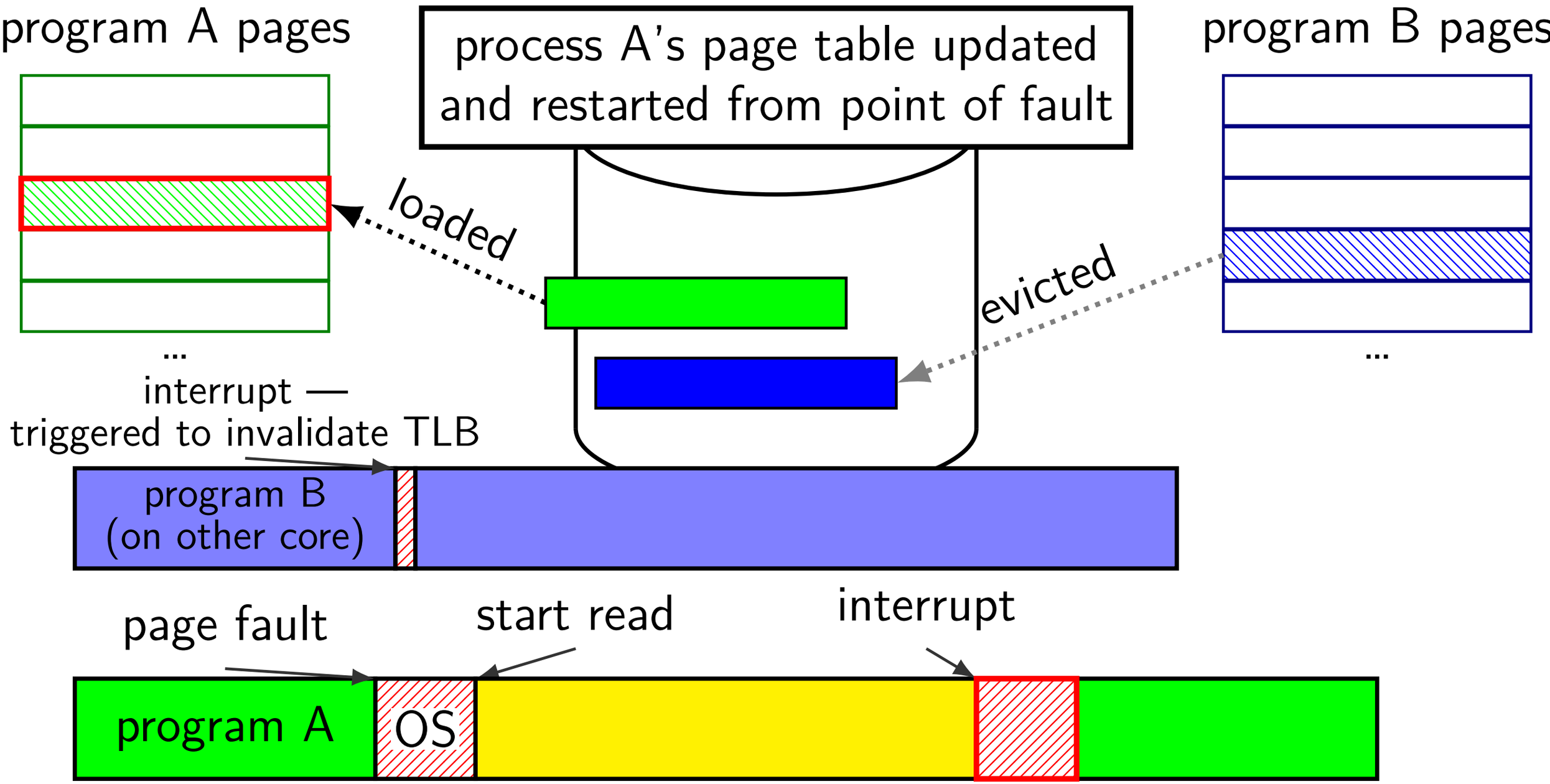
TLB shootdown



TLB shootdown



TLB shootdown



x86-64 page table entries (1)

63	62–61	60–59	58–52	51	M[1]–M-1	32–9	8	7	6	5	4	3	2	1	0	
XD	Prot. Key[4]	Ignored	Rsvd.		Address of 4KB page frame		Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored																0	PTE: not present

present = valid

R/W = writes allowed?

U/S = user-mode allowed? (“user/supervisor”)

XD = execute-disable?

A = accessed? (MMU sets to 1 on page read/write)

D = dirty? (MMU sets to 1 on page write)

x86-64 page table entries (1)

63	62–61	60–59	58–57	56–55	54–52	51–M	M-1	31–30	29–28	27–26	25–24	23–22	21–20	19–18	17–16	15–14	13–12	11–10	9–8	7	6	5	4	3	2	1	0	
XD	Prot. Key[4]	Ignored		Rsvd.		Address of 4KB page frame													Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored																											0	PTE: not present

present = valid

R/W = writes allowed?

U/S = user-mode allowed? (“user/supervisor”)

XD = execute-disable?

A = accessed? (MMU sets to 1 on page read/write)

D = helps support replacement policies for swapping

x86-64 page table entries (1)

63	62–61	60–59	58–57	56–55	54–52	51–M	M-1	31–9	8	7	6	5	4	3	2	1	0	
XD	Prot. Key[4]	Ignored	Rsvd.	Address of 4KB page frame					Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored																		0	PTE: not present

present = valid

R/W = writes allowed?

U/S = user-mode allowed? (“user/supervisor”)

XD = execute-disable?

A = accessed? (MMU sets to 1 on page read/write)

D = dirty? (MMU sets to 1 on page write)

helps support writeback policy for swapping

x86-64 page table entries (2)

63	62–61	60–59	58–52	51	M[1]	M-1	32–9	8	7	6	5	4	3	2	1	0	
XD	Prot. Key[4]	Ignored	Rsvd.	Address of 4KB page frame				Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored																	0	PTE: not present

G = global? (shared between all page tables)

PWT, PCD, PAT = control how caches work when accessing physical page:

- can disable using the cache entirely
- can disable write-back (use write-through instead)
- multicore-related cache settings
- (and some other settings)

Table 11-11. Selection of PAT Entries with PAT, PCD, and PWT Flags

PAT	PCD	PWT	PAT Entry
0	0	0	PAT0
0	0	1	PAT1
0	1	0	PAT2
0	1	1	PAT3
1	0	0	PAT4
1	0	1	PAT5
1	1	0	PAT6
1	1	1	PAT7

x86-64 page table entries (2)

63	62–61	60–59	58–52	51	M[1]	M-1	32–9	8	7	6	5	4	3	2	1	0	
XD	Prot. Key[4]	Ignored	Rsvd.				Address of 4KB page frame	Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored																	0	PTE: not present

G = global? (shared between all page tables)

P CPU won't evict TLB entries on most page table base registers changes

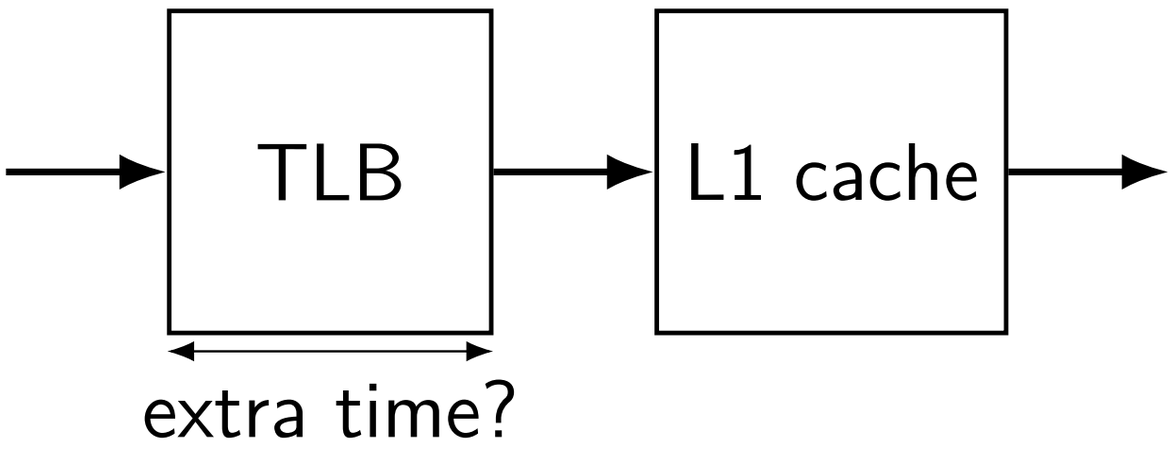
can disable using the cache entirely

can disable write-back (use write-through instead)

multicore-related cache settings

(and some other settings)

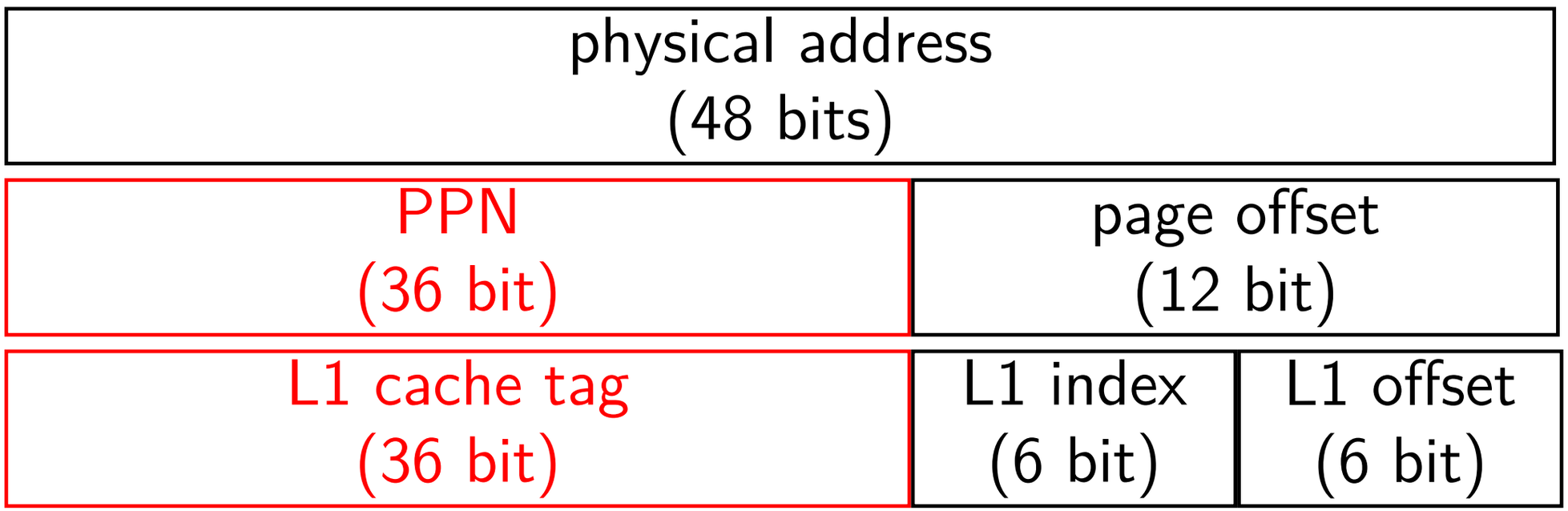
TLBs and performance



L1 caches and page numbers (Intel Skylake)

<table>
<tr><td colspan="3">physical address
(48 bits)</td></tr>
<tr><td colspan="2">PPN
(36 bit)</td><td>page offset
(12 bit)</td></tr>
<tr><td>L1 cache tag
(36 bit)</td><td>L1 index
(6 bit)</td><td>L1 offset
(6 bit)</td></tr>
</table>

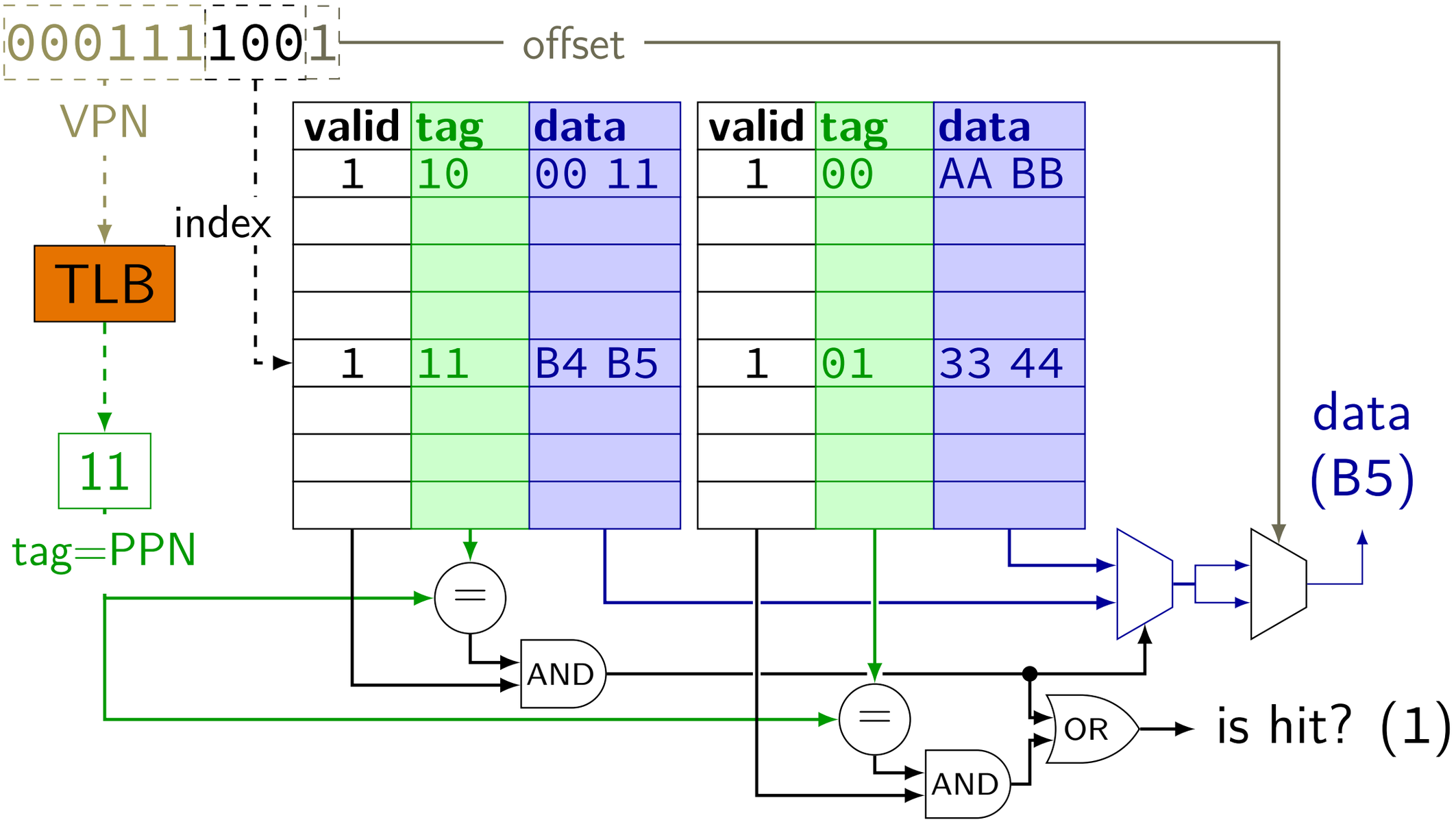
L1 caches and page numbers (Intel Skylake)



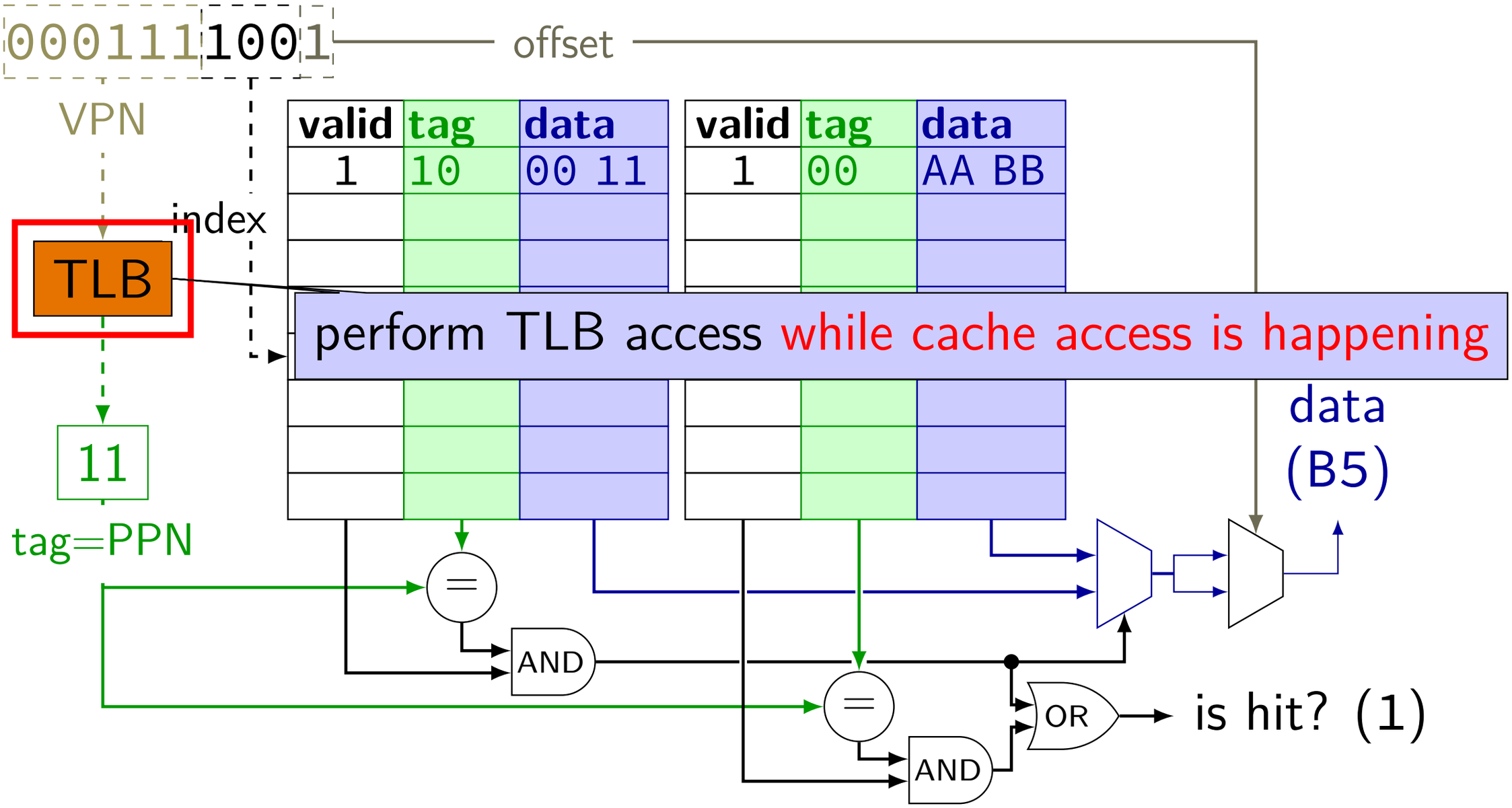
not a coincidence

why did Intel make this decision?

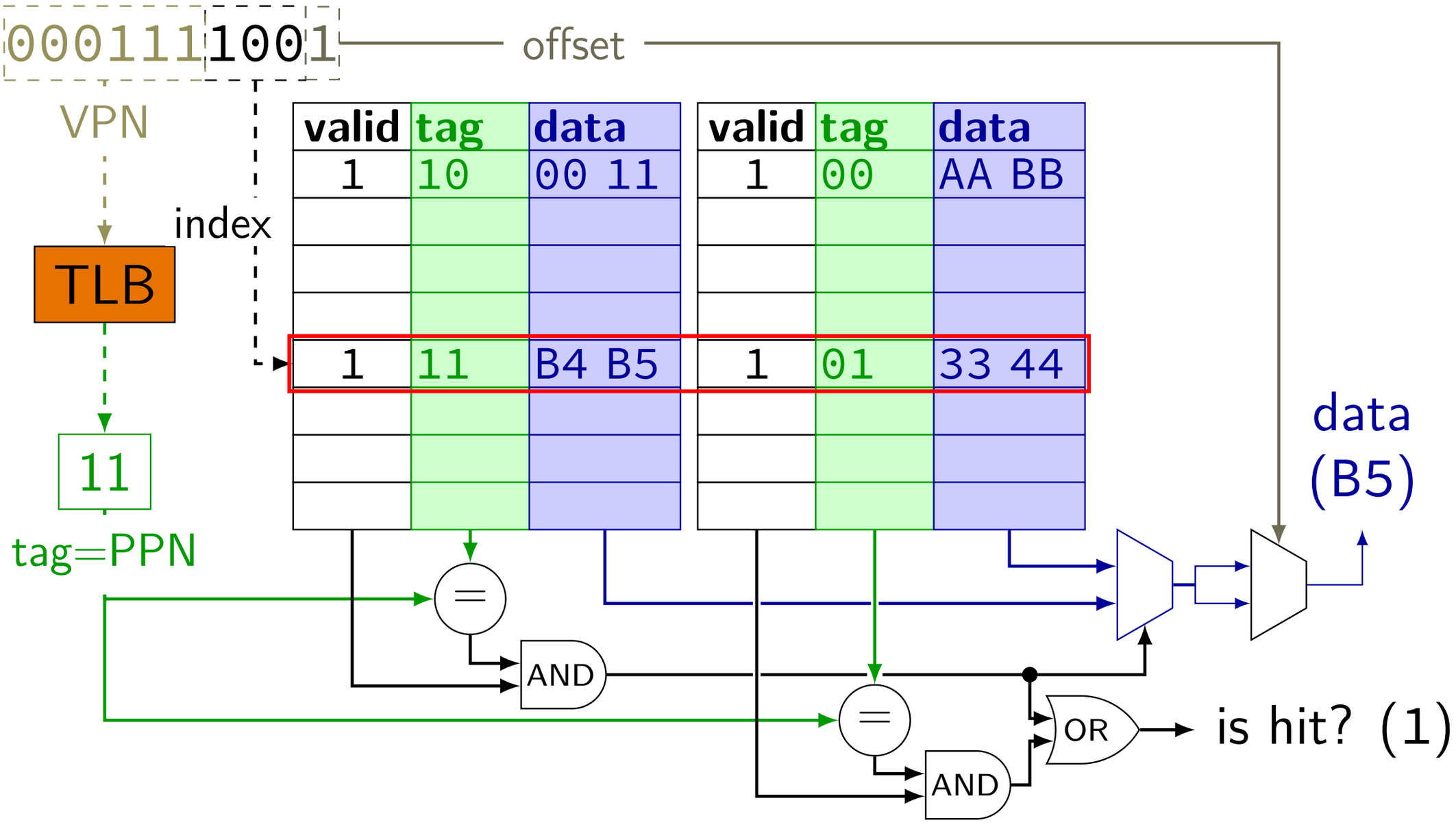
overlapping TLB and cache access



overlapping TLB and cache access



overlapping TLB and cache access



virtually-indexed, physically-tagged

called virtually-indexed, physically-tagged cache

requirement: index contained entirely in page offset

do not need to do translation to start cache access

tag overlaps with PPN

example: tag=PPN

do TLB access while retrieving cache set

most common design in current processors

reason for highly associative (e.g. 8-way) L1 caches

address splitting

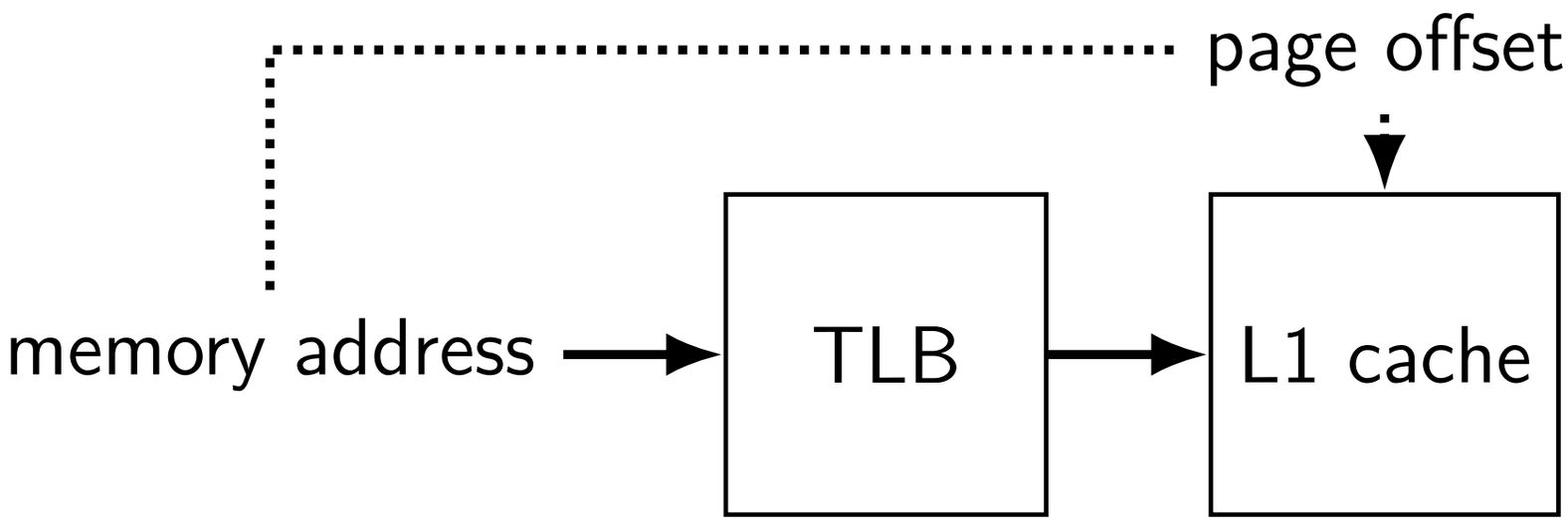
16-bit virtual addresses

64-byte pages

256B, 8-way L1 cache with 16B blocks

can TLB and cache access overlap?

physical caches

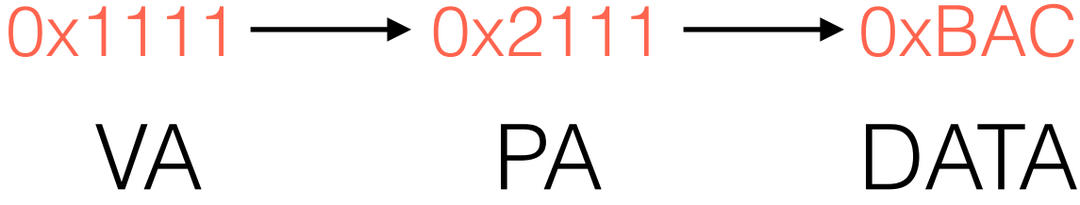


Exercise

Consider a machine with:

Description of Virtual Memory
1) 48 bit virtual address
2) Pages are 4KB
3) 4 Level Page Table, where each Page table is one Page
4) PTE are 8 bytes the first 40bits of which represent Physical Page Number
5) 4 way TLB with 64 entries

Description of Physical Memory
1)1TB Disk
2)16GB DRAM
3) 8-way associative L1 cache with 64 sets and 64 Byte blocks



Assume that the cache and TLB are originally empty

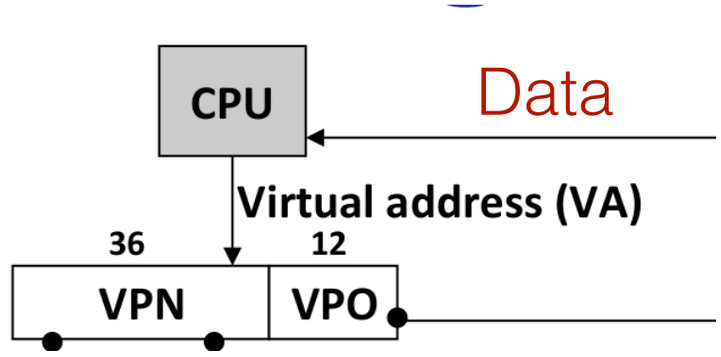
What are the contents TLB after 0x1111 access?

How many bits are in the physical address?

What are the contents Cache after 0x1111 access

Consider a machine with:

Description of Virtual Memory
1) 48 bit virtual address
2) Pages are 4KB
3) 4 Level Page Table, where each Page table is one Page
4) PTE are 8 bytes the first 40bits of which represent Physical Page Number
5) 4 way TLB with 64 entries

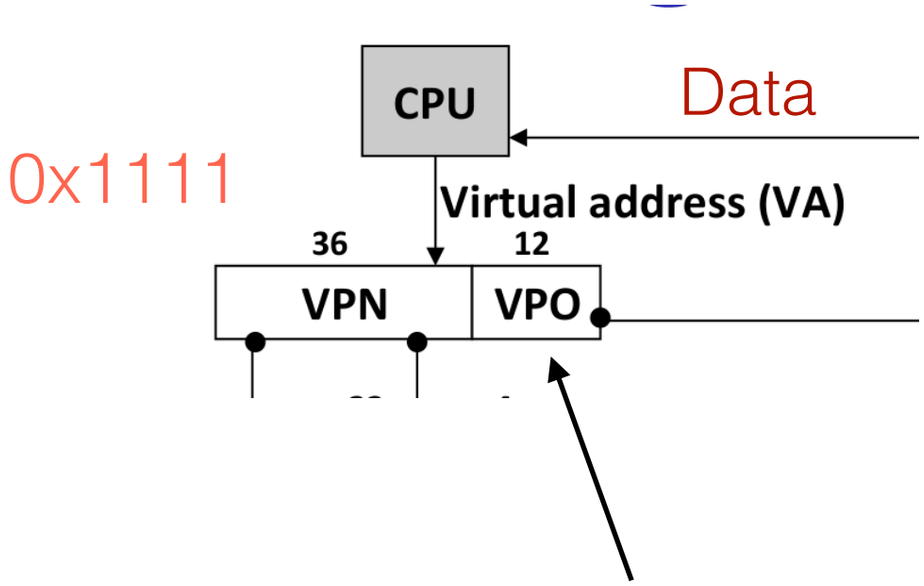


0x1111 ⟶ 0x2111 ⟶ 0xBAC

VA PA DATA

Consider a machine with:

Description of Virtual Memory
1) 48 bit virtual address
2) Pages are 4KB
3) 4 Level Page Table, where each Page table is one Page
4) PTE are 8 bytes the first 40bits of which represent Physical Page Number
5) 4 way TLB with 64 entries



$4KB = 4*2^{10} = 2^{12}$

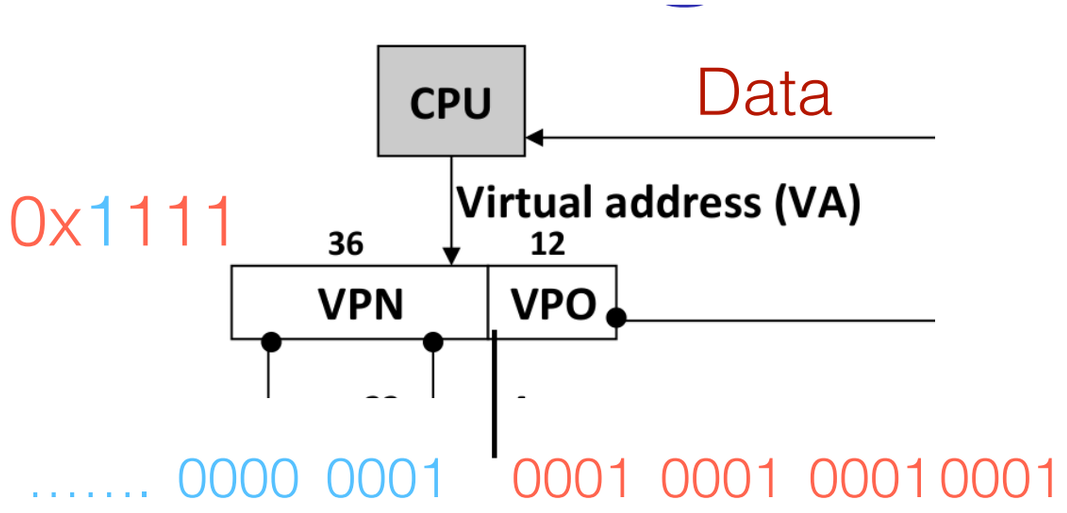
VPN 36 bits is just what is left over

0x1111 → 0x2111 → 0xBAC

VA PA DATA

Consider a machine with:

Description of Virtual Memory
1) 48 bit virtual address
2) Pages are 4KB
3) 4 Level Page Table, where each Page table is one Page
4) PTE are 8 bytes the first 40bits of which represent Physical Page Number
5) 4 way TLB with 64 entries

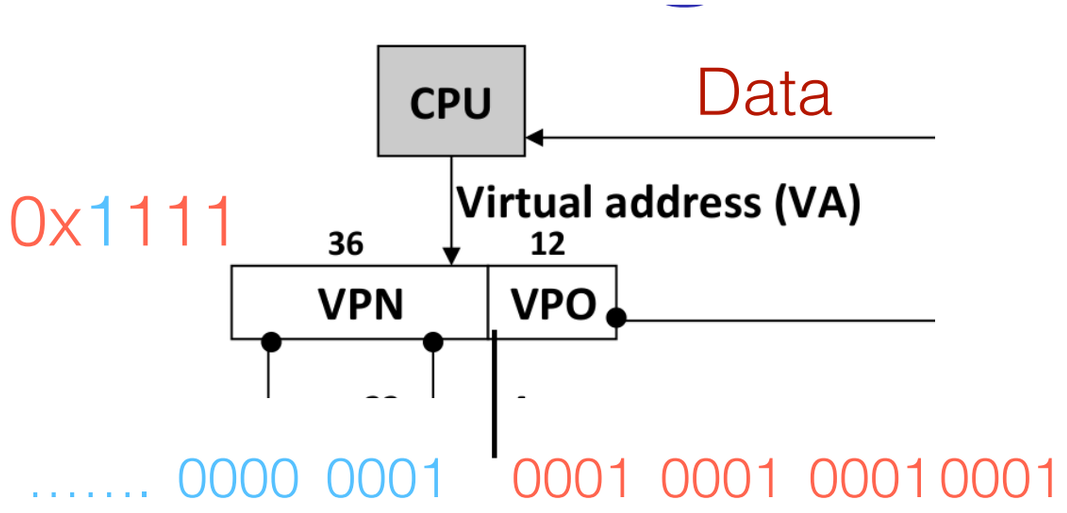


0x1111 ⟶ 0x2111 ⟶ 0xBAC

VA PA DATA

Consider a machine with:

Description of Virtual Memory
1) 48 bit virtual address
2) Pages are 4KB
3) 4 Level Page Table, where each Page table is one Page
4) PTE are 8 bytes the first 40bits of which represent Physical Page Number
5) 4 way TLB with 64 entries

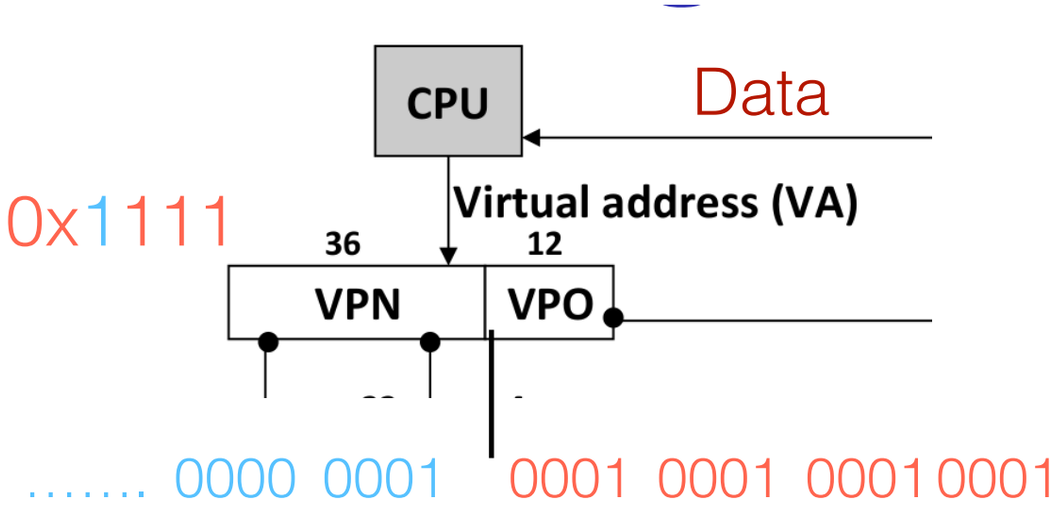


0x1111 ⟶ 0x2111 ⟶ 0xBAC

VA PA DATA

Consider a machine with:

Description of Virtual Memory
1) 48 bit virtual address
2) Pages are 4KB
3) 4 Level Page Table, where each Page table is one Page
4) PTE are 8 bytes the first 40bits of which represent Physical Page Number
5) 4 way TLB with 64 entries



How many entries

$$4KB/8 = 2^{12}/2^{3} = 2^{9}$$

How many bits needed
To address all the entries

9

How many levels

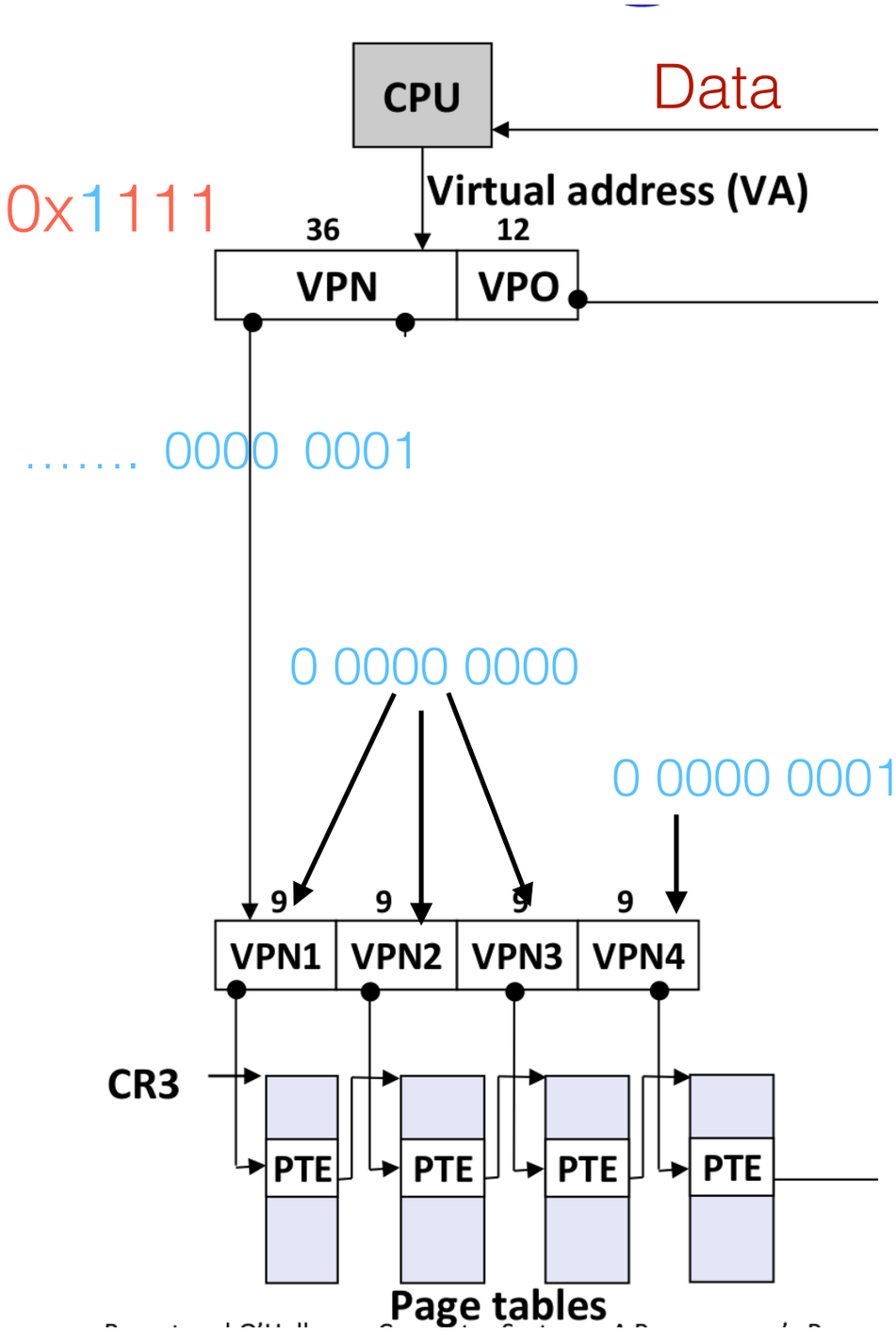
4

0x1111 → 0x2111 → 0xBAC

VA PA DATA

Consider a machine with:

Description of Virtual Memory
1) 48 bit virtual address
2) Pages are 4KB
3) 4 Level Page Table, where each Page table is one Page
4) PTE are 8 bytes the first 40bits of which represent Physical Page Number
5) 4 way TLB with 64 entries



0x1111 → 0x2111 → 0xBAC

VA PA DATA

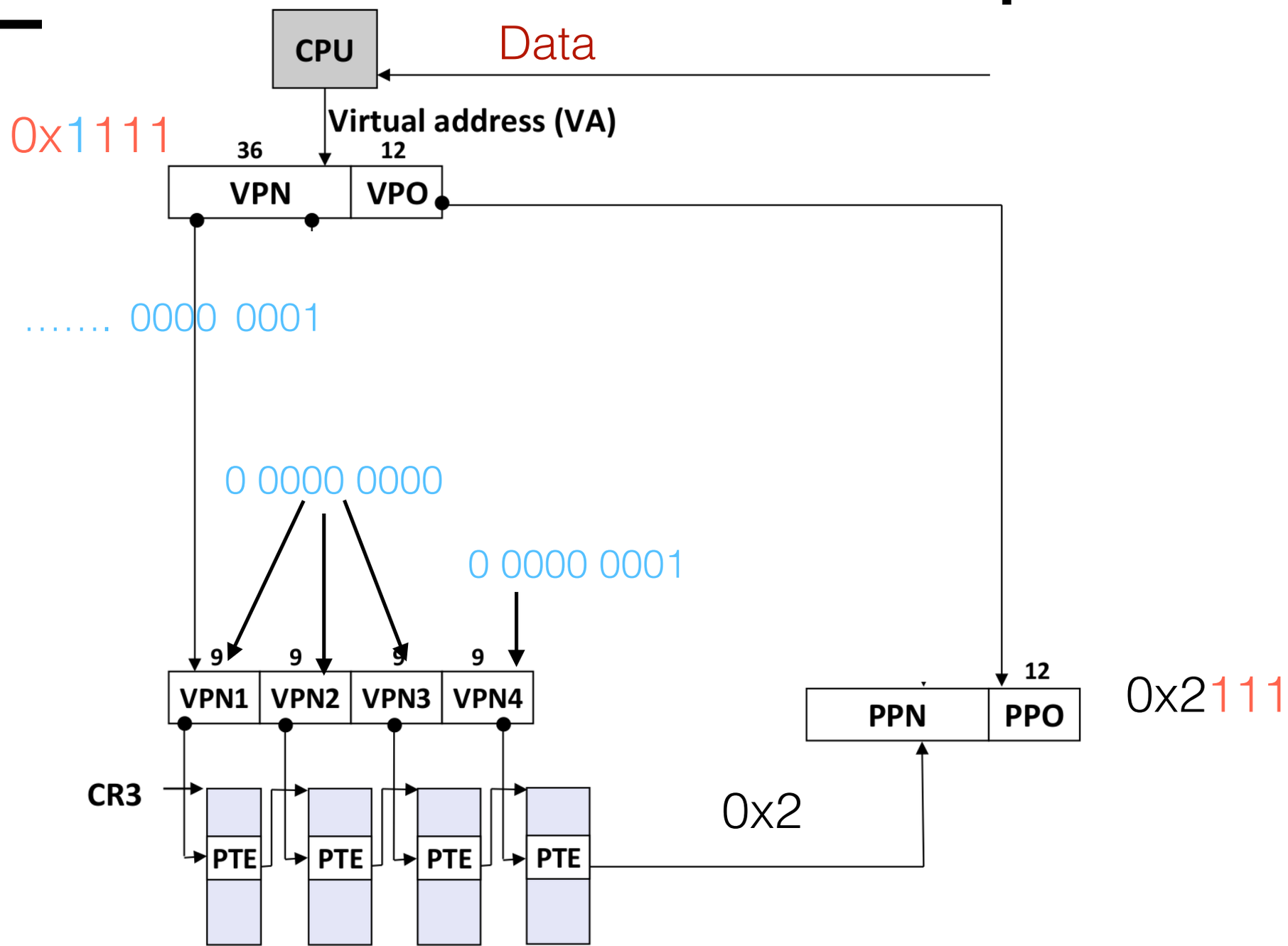
Consider a machine with:

4) PTE are 8 bytes the first 40bits of which represent Physical Page Number

5) 4 way TLB with 64 entries

0x1111 → 0x2111 → 0xBAC

VA PA DATA



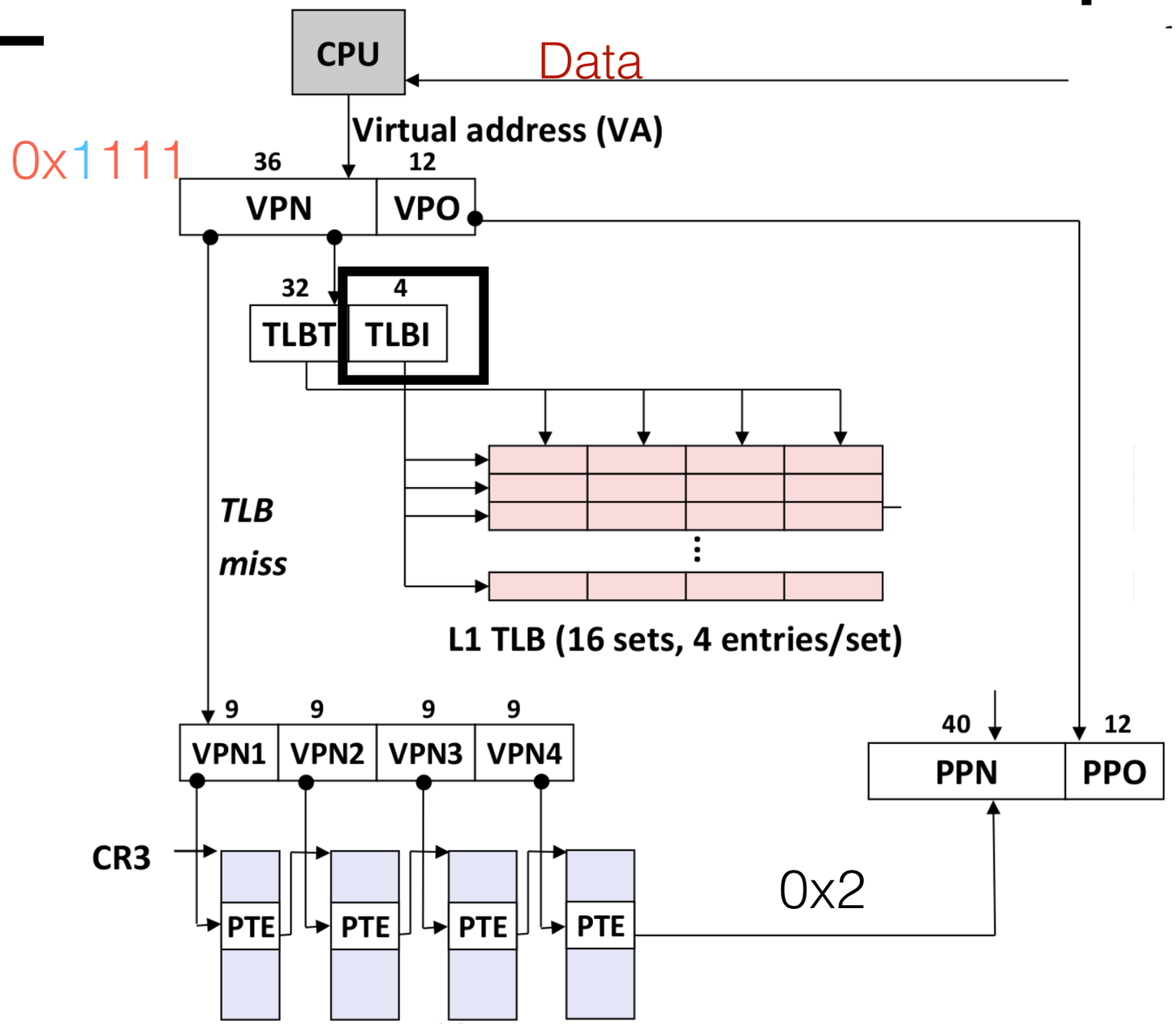
Consider a machine with:

4) PTE are 8 bytes the first 40bits of which represent Physical Page Number

5) 4 way TLB with 64 entries

0x1111 ⟶ 0x2111 ⟶ 0xBAC

VA PA DATA



Number of sets

64/4 =16

Number of Bit to Index TLB = 4

Because 16 $=2^4$

0x2111

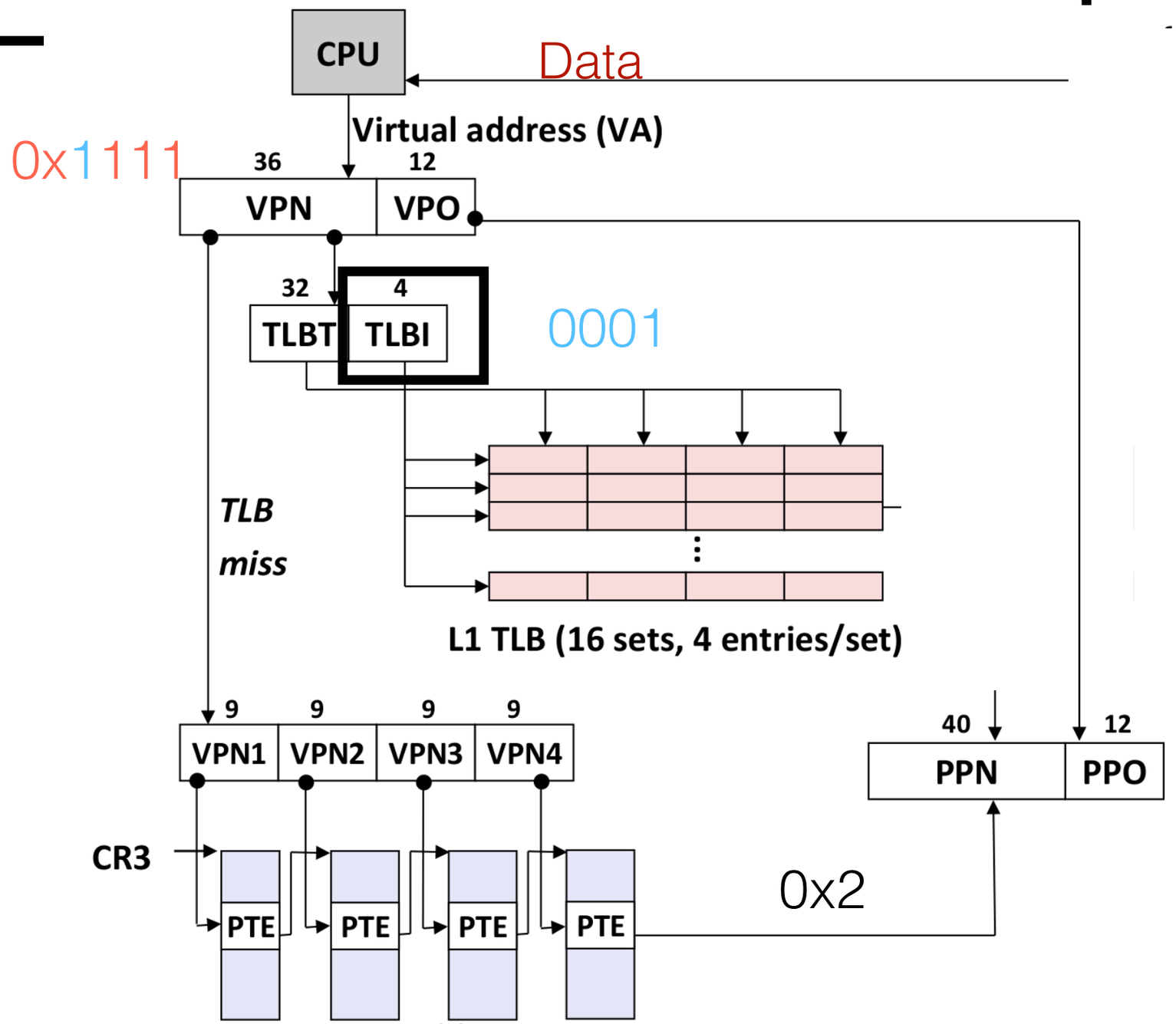
Consider a machine with:

4) PTE are 8 bytes the first 40bits of which represent Physical Page Number

5) 4 way TLB with 64 entries

0x1111 ⟶ 0x2111 ⟶ 0xBAC

VA PA DATA



Number of sets

64/4 =16

Number of Bit to Index TLB = 4

Because 16 $=2^4$

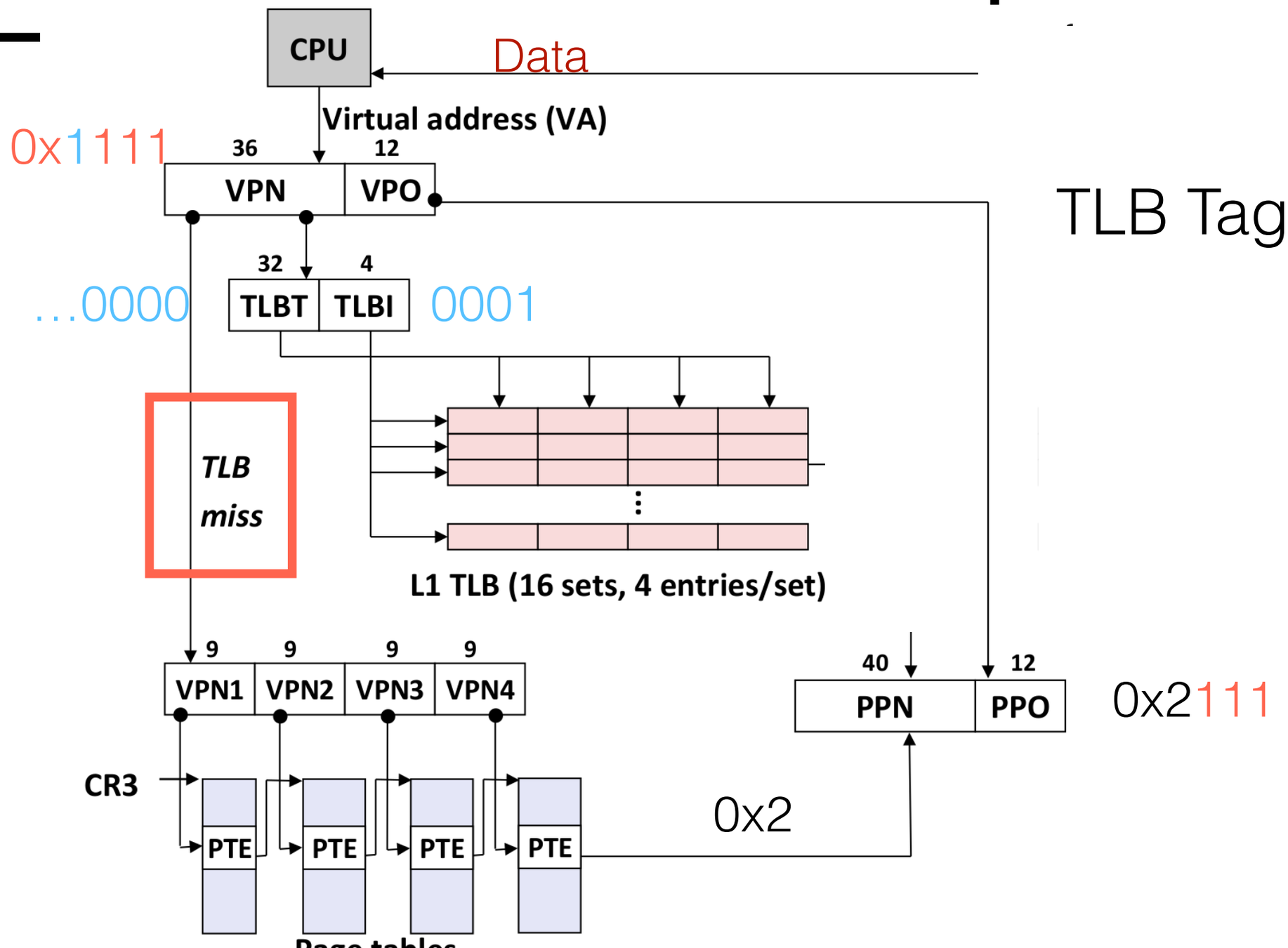
Consider a machine with:

4) PTE are 8 bytes the first 40bits of which represent Physical Page Number

5) 4 way TLB with 64 entries

0x1111 ⟶ 0x2111 ⟶ 0xBAC

VA PA DATA



TLB Tag = 36 -4 =32

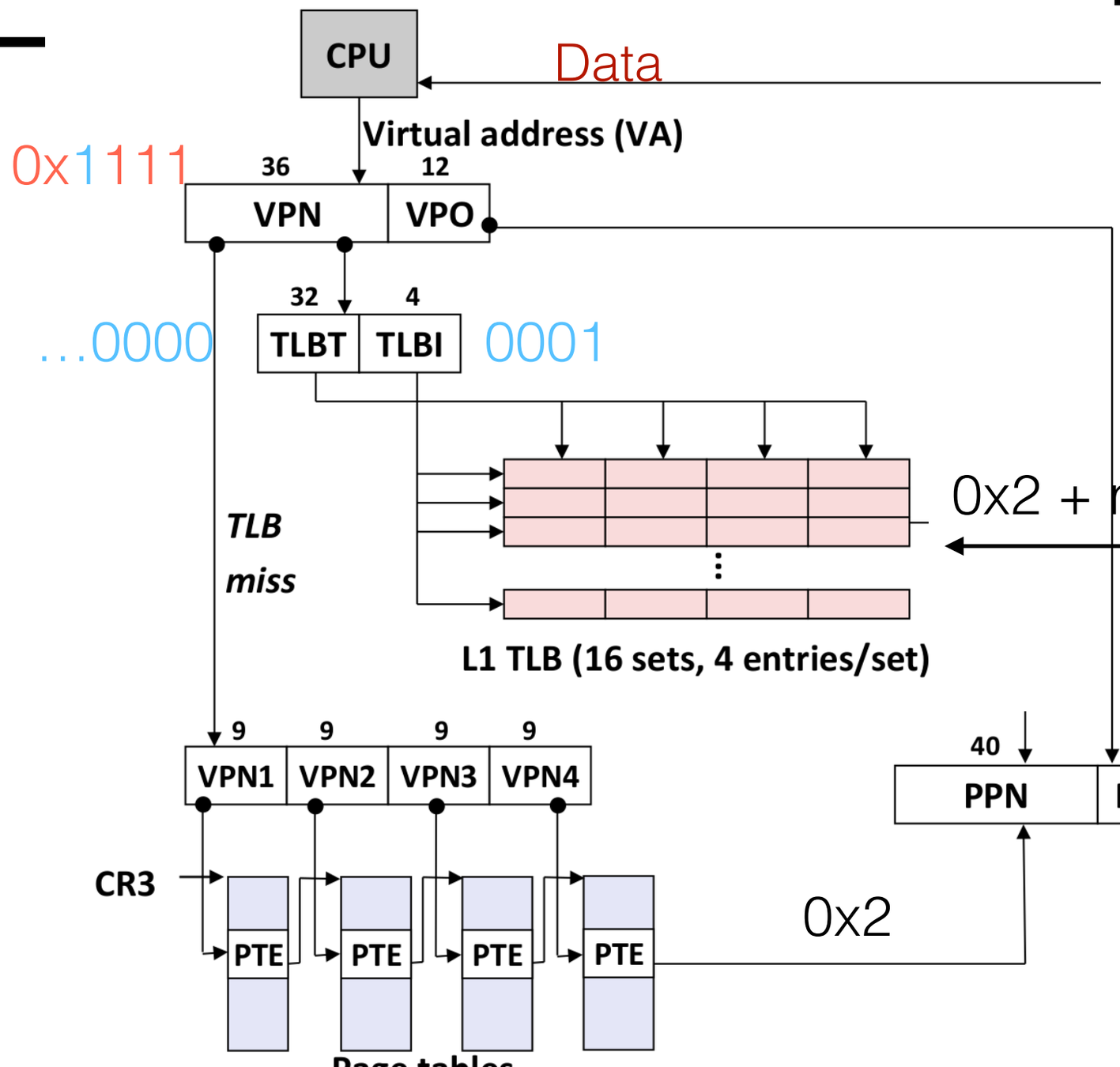
Consider a machine with:

4) PTE are 8 bytes the first 40bits of which represent Physical Page Number

5) 4 way TLB with 64 entries

0x1111 ⟶ 0x2111 ⟶ 0xBAC

VA PA DATA



TLB Tag = 36 -4 =32

TLB now gets updated PTE

Consider a machine with:

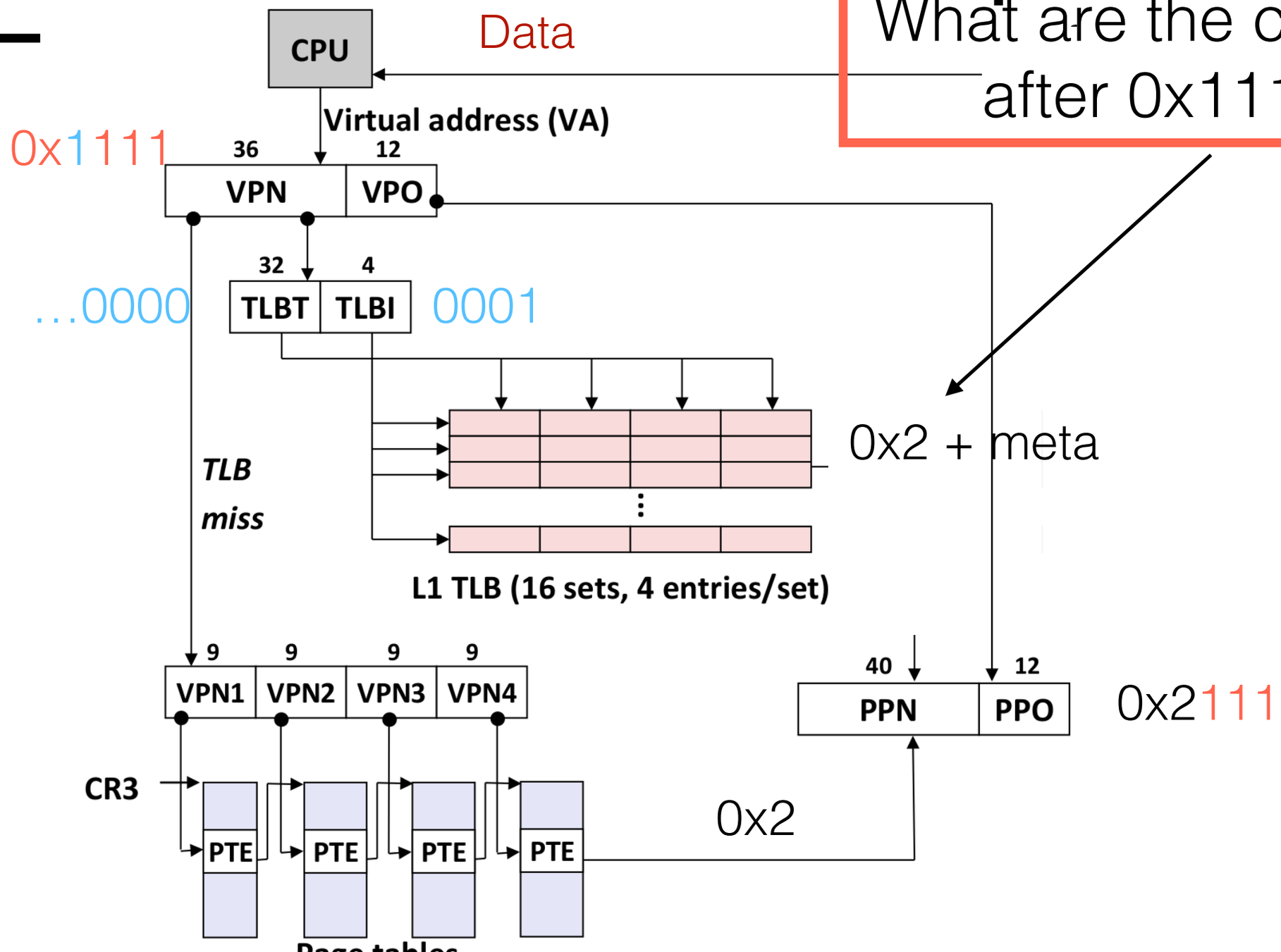
4) PTE are 8 bytes the first 40bits of which represent Physical Page Number

5) 4 way TLB with 64 entries

0x1111 → 0x2111 → 0xBAC

VA PA DATA

What are the contents of TLB after 0x1111 access?



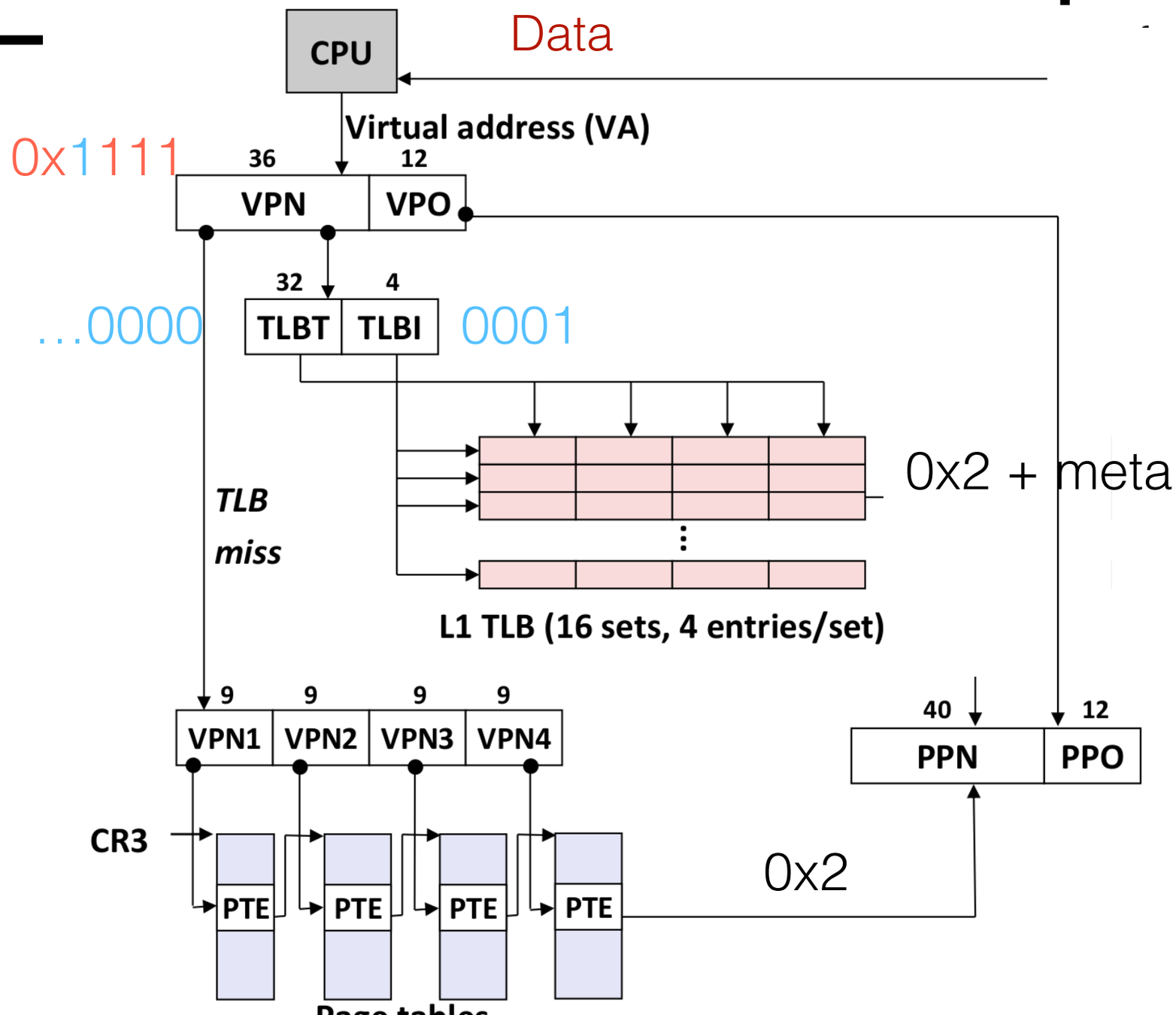
Consider a machine with:

4) PTE are 8 bytes the first 40bits of which represent Physical Page Number

5) 4 way TLB with 64 entries

0x1111 → 0x2111 → 0xBAC

VA PA DATA



How many bits are in the physical address?

0x1111 → 0x2111 → 0xBAC

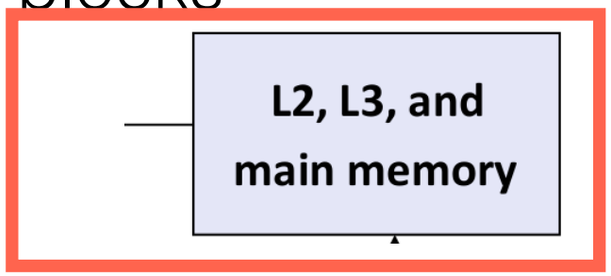
VA PA DATA

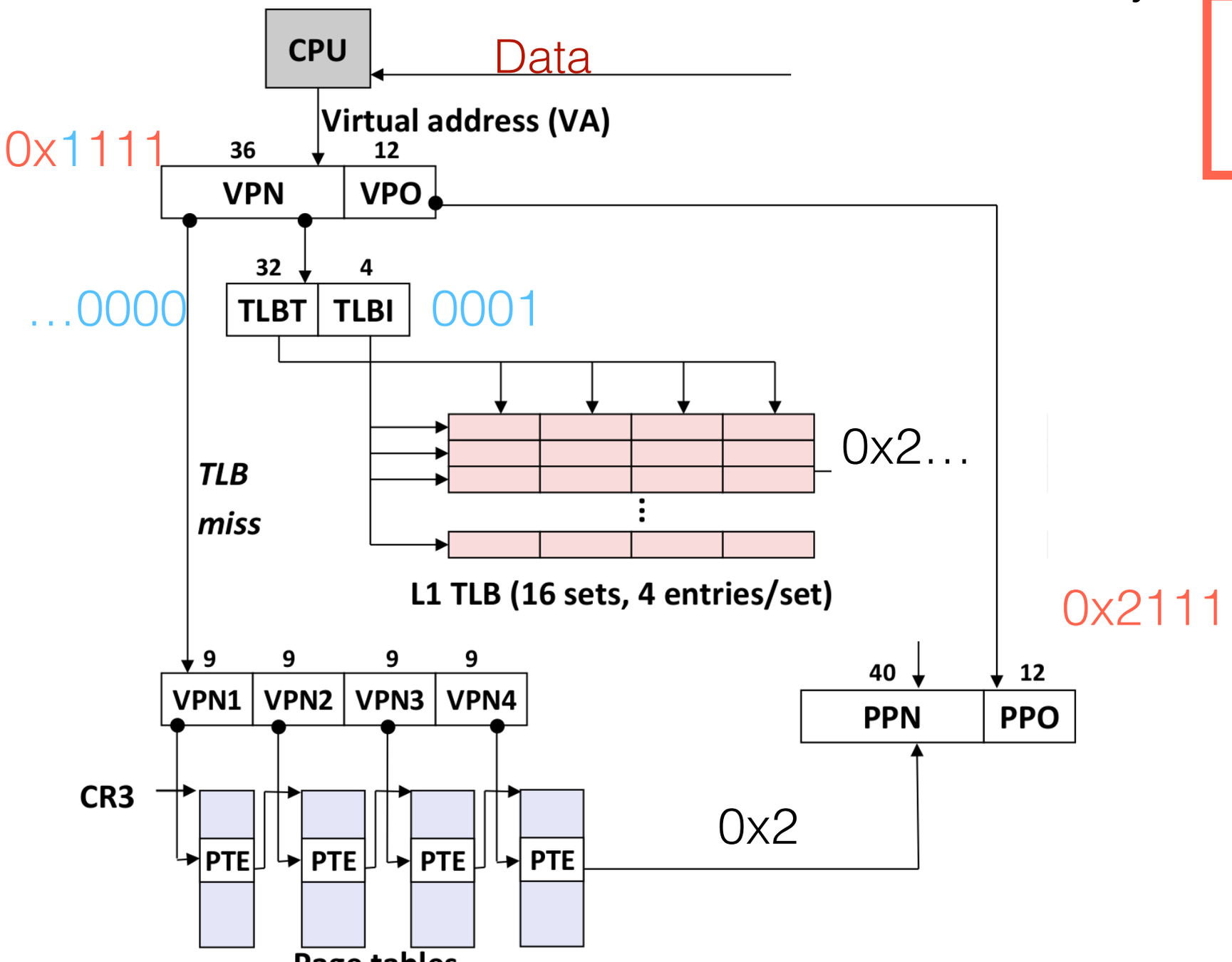
Description of Physical Memory

1)1TB Disk

2)16GB DRAM

3) 8-way associative L1 cache with 64 sets and 64 Byte blocks

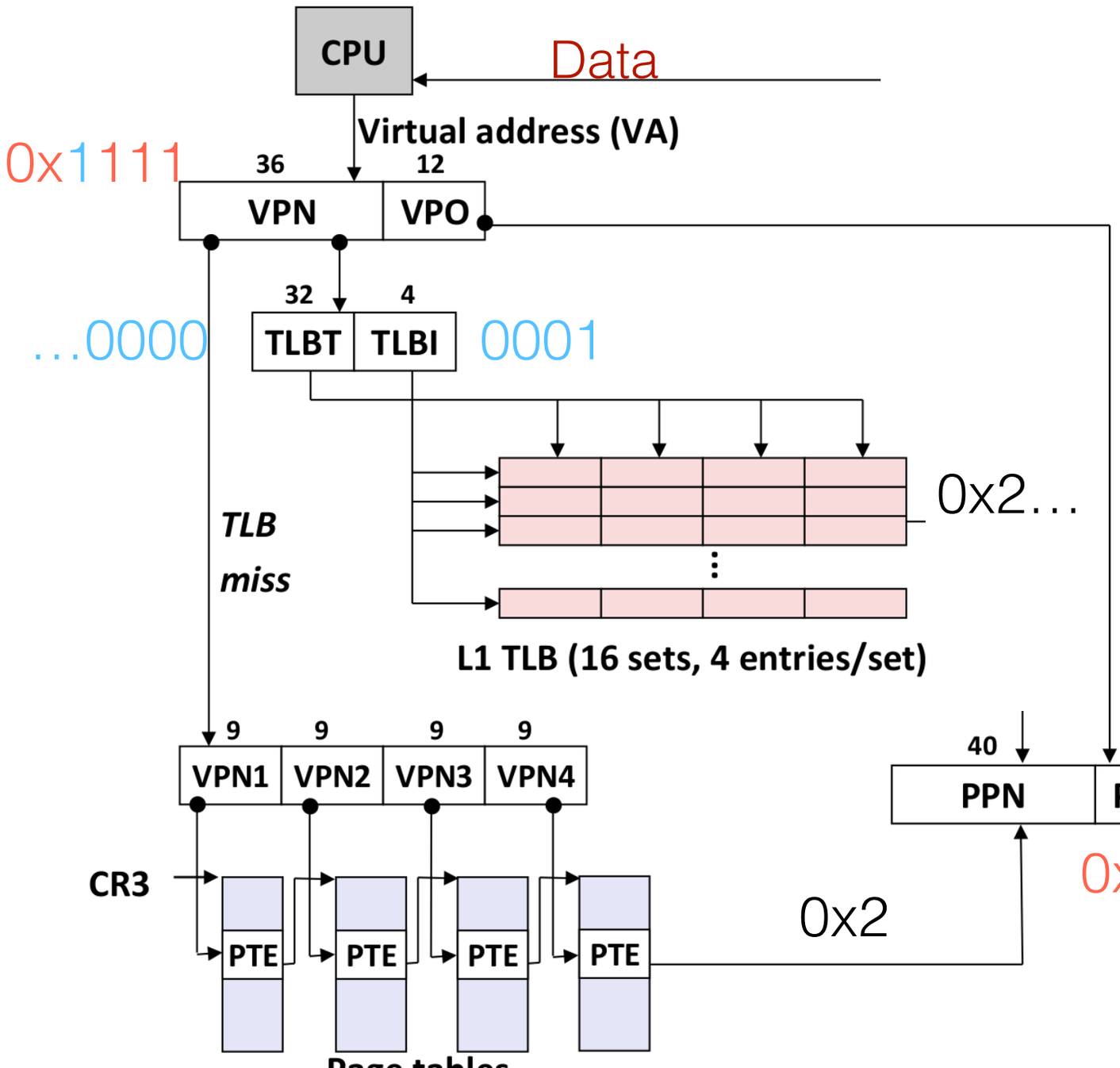




0x1111 → 0x2111 → 0xBAC

VA PA DATA

Description of Physical Memory
3) 8-way associative L1 cache with 64 sets and 64 Byte blocks



Cache Offset

Log2(64) = 6

Cache Index

Log2(64) = 6

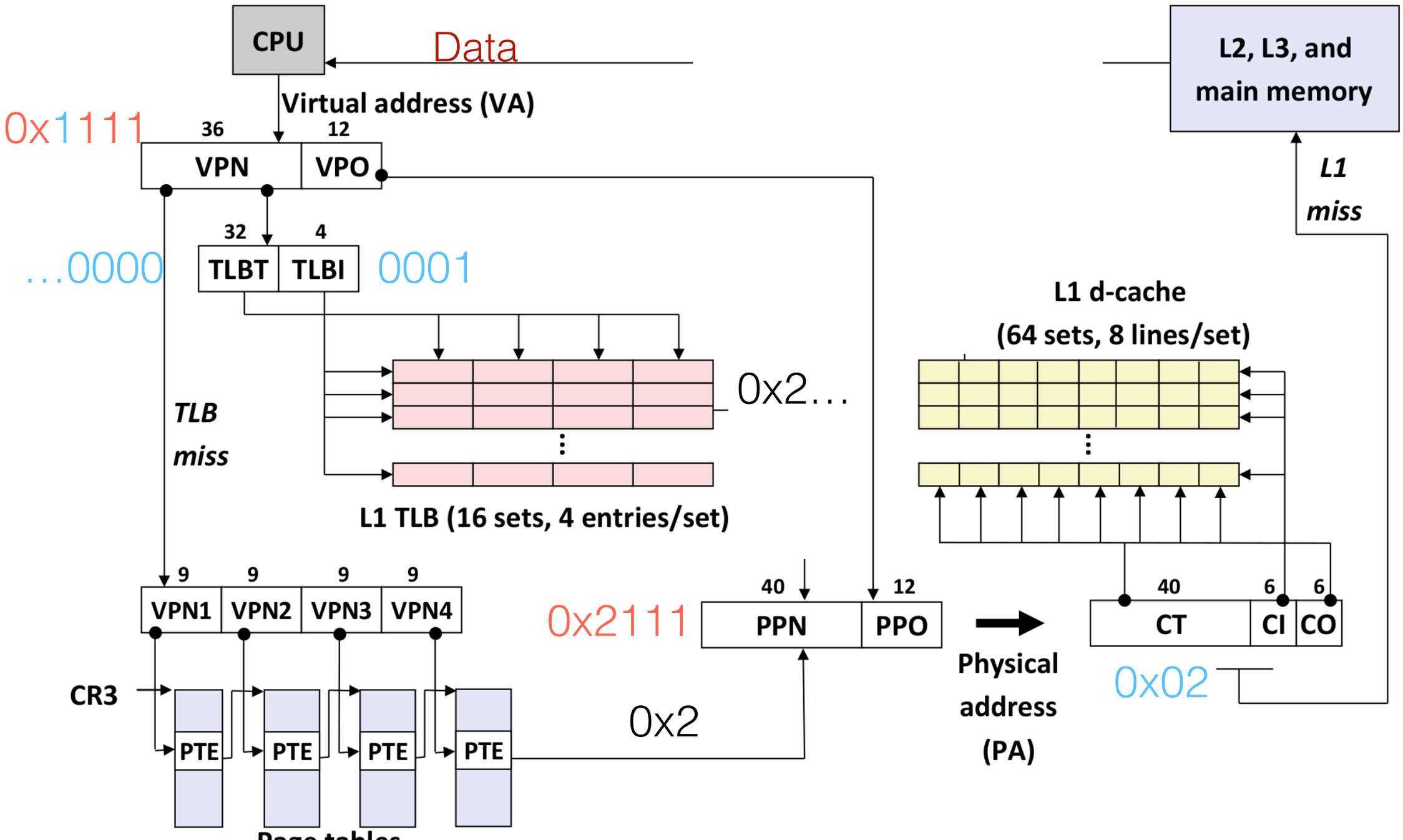
Cache Tag

52 -12 = 40

0x1111 → 0x2111 → 0xBAC

VA PA DATA

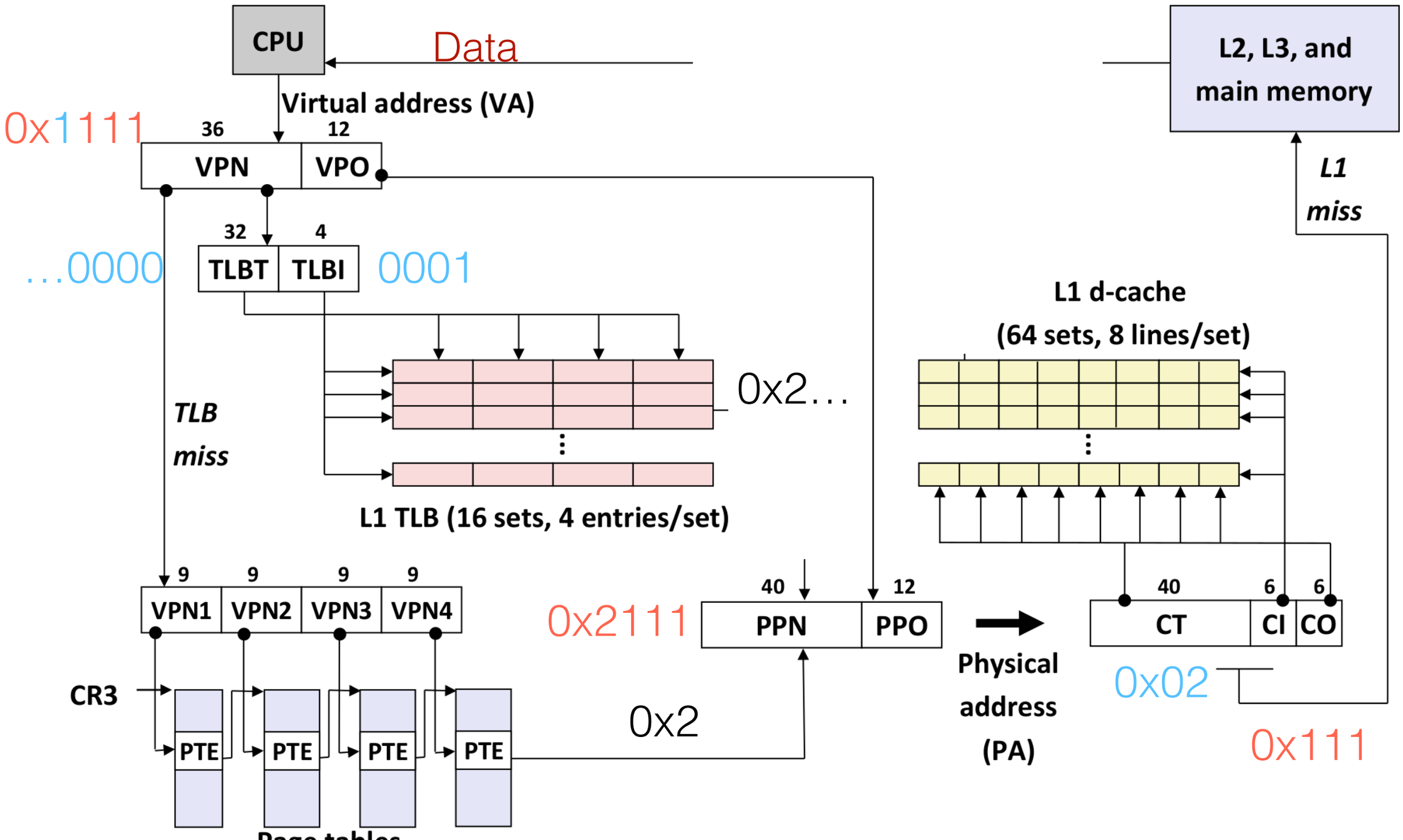
Description of Physical Memory
3) 8-way associative L1 cache with 64 sets and 64 Byte blocks



0x1111 → 0x2111 → 0xBAC

VA PA DATA

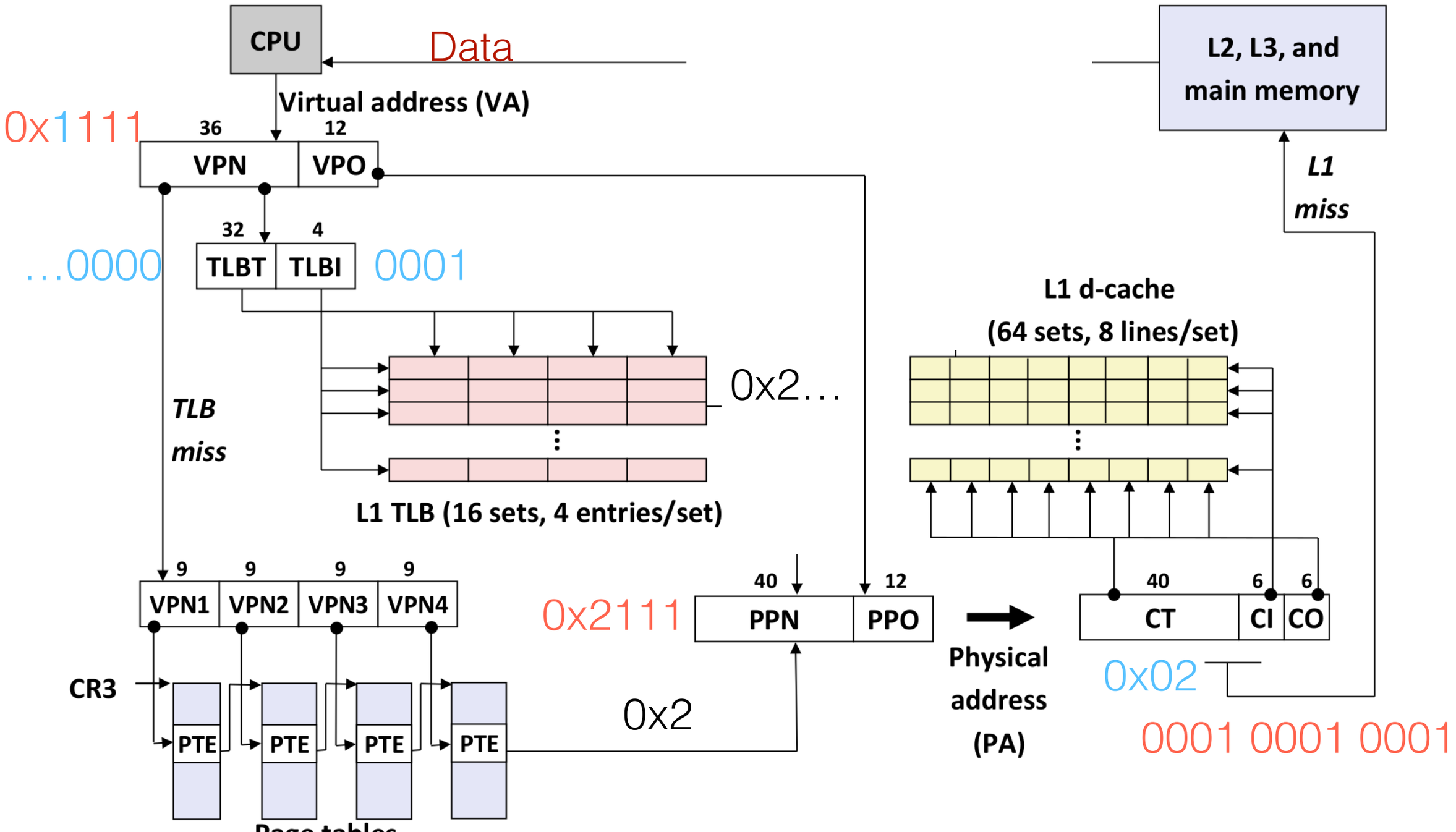
Description of Physical Memory
3) 8-way associative L1 cache with 64 sets and 64 Byte blocks



0x1111 → 0x2111 → 0xBAC

VA PA DATA

Description of Physical Memory
3) 8-way associative L1 cache with 64 sets and 64 Byte blocks

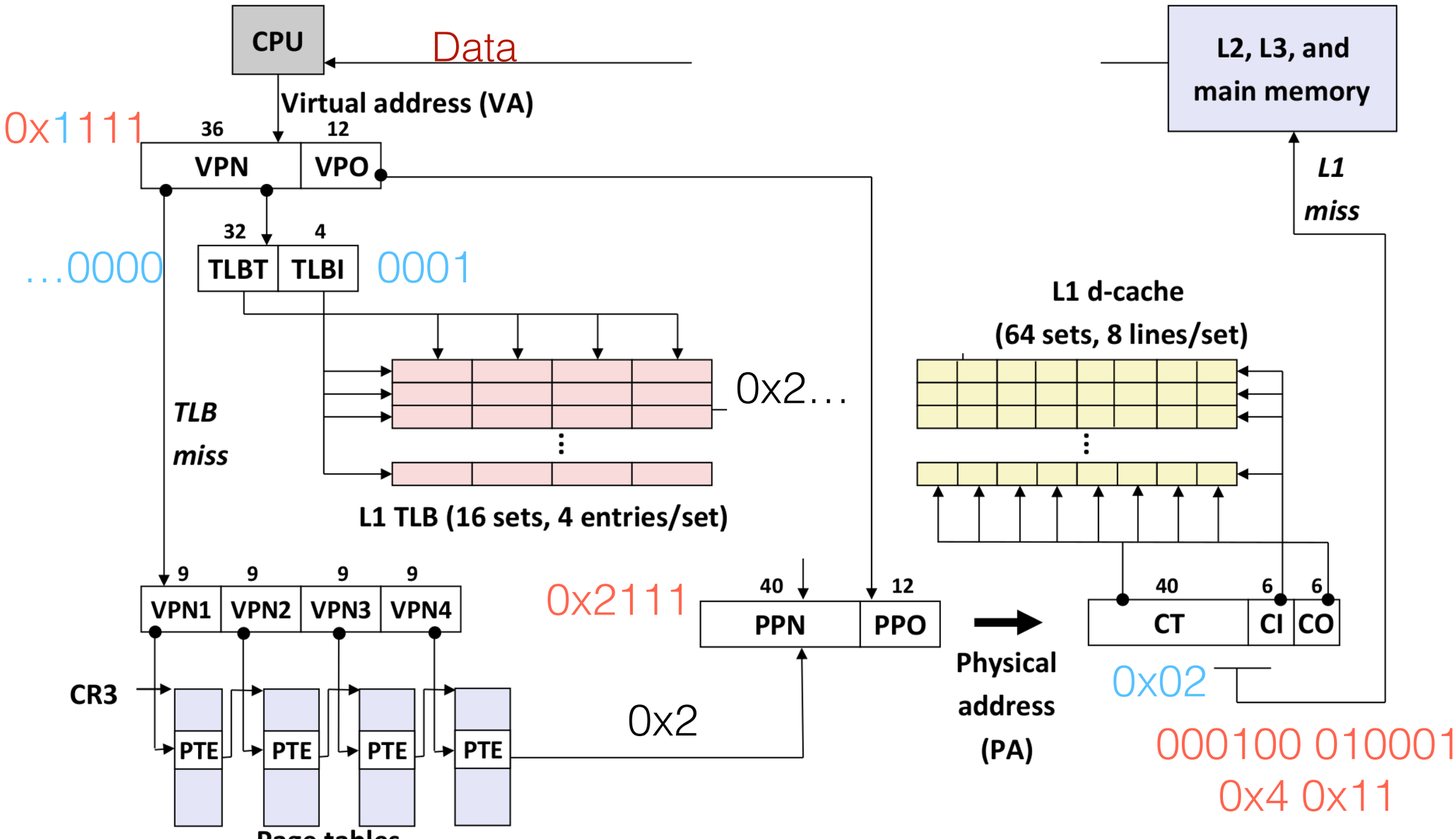


0x1111 → 0x2111 → 0xBAC

VA PA DATA

Description of Physical Memory

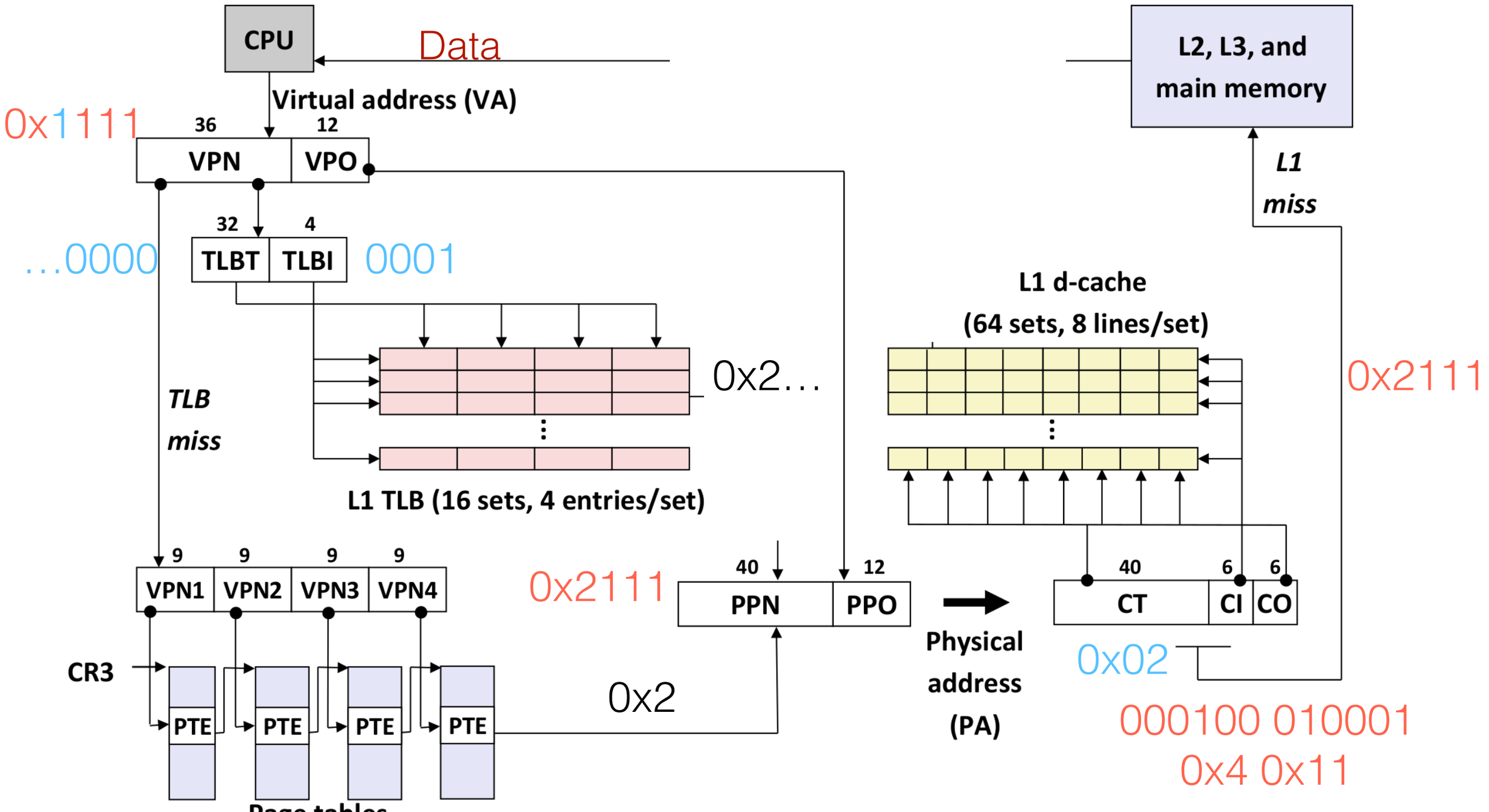
3) 8-way associative L1 cache with 64 sets and 64 Byte blocks



0x1111 → 0x2111 → 0xBAC

VA PA DATA

Description of Physical Memory
3) 8-way associative L1 cache with 64 sets and 64 Byte blocks

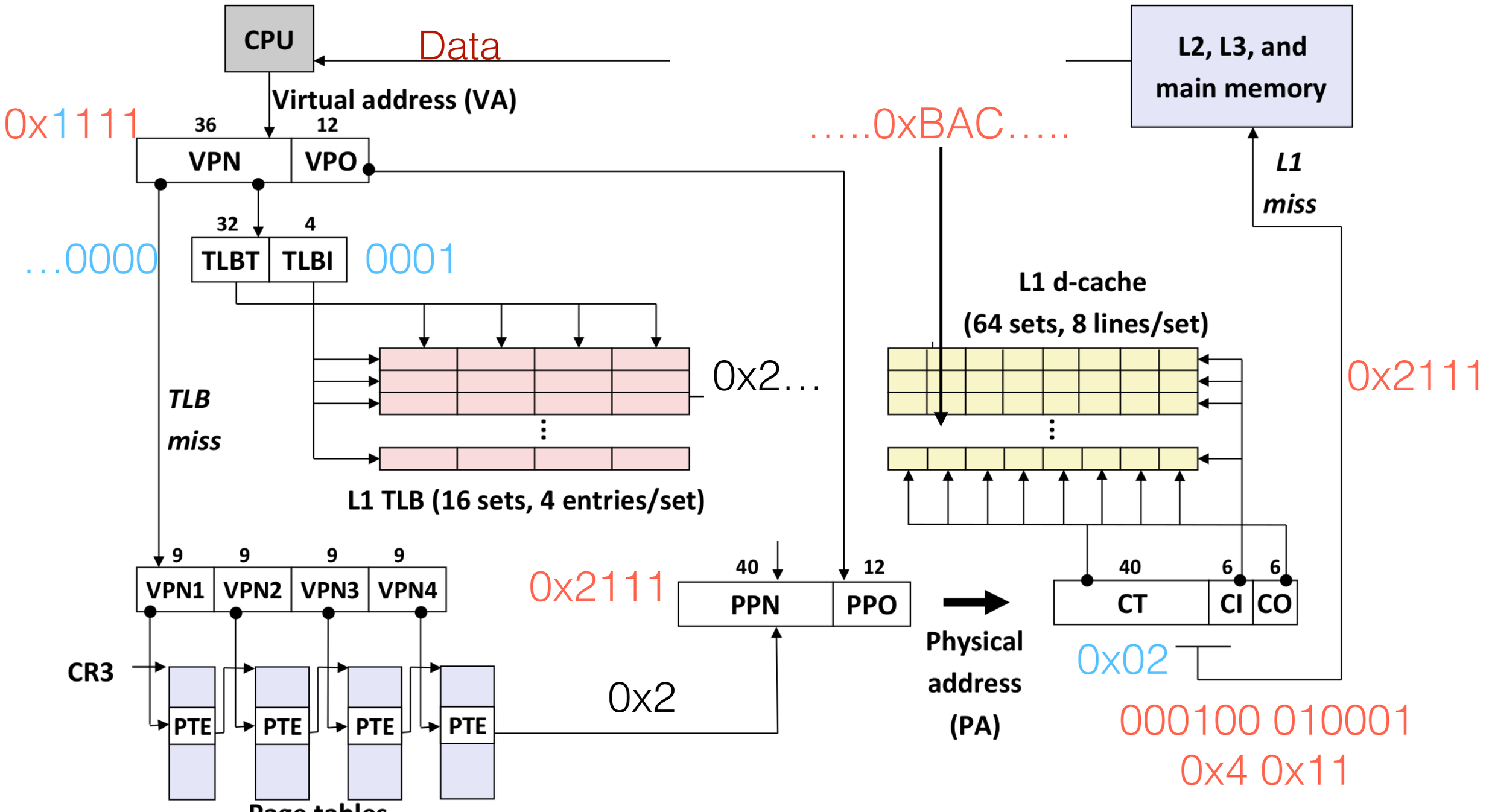


0x1111 → 0x2111 → 0xBAC

VA PA DATA

Description of Physical Memory

3) 8-way associative L1 cache with 64 sets and 64 Byte blocks

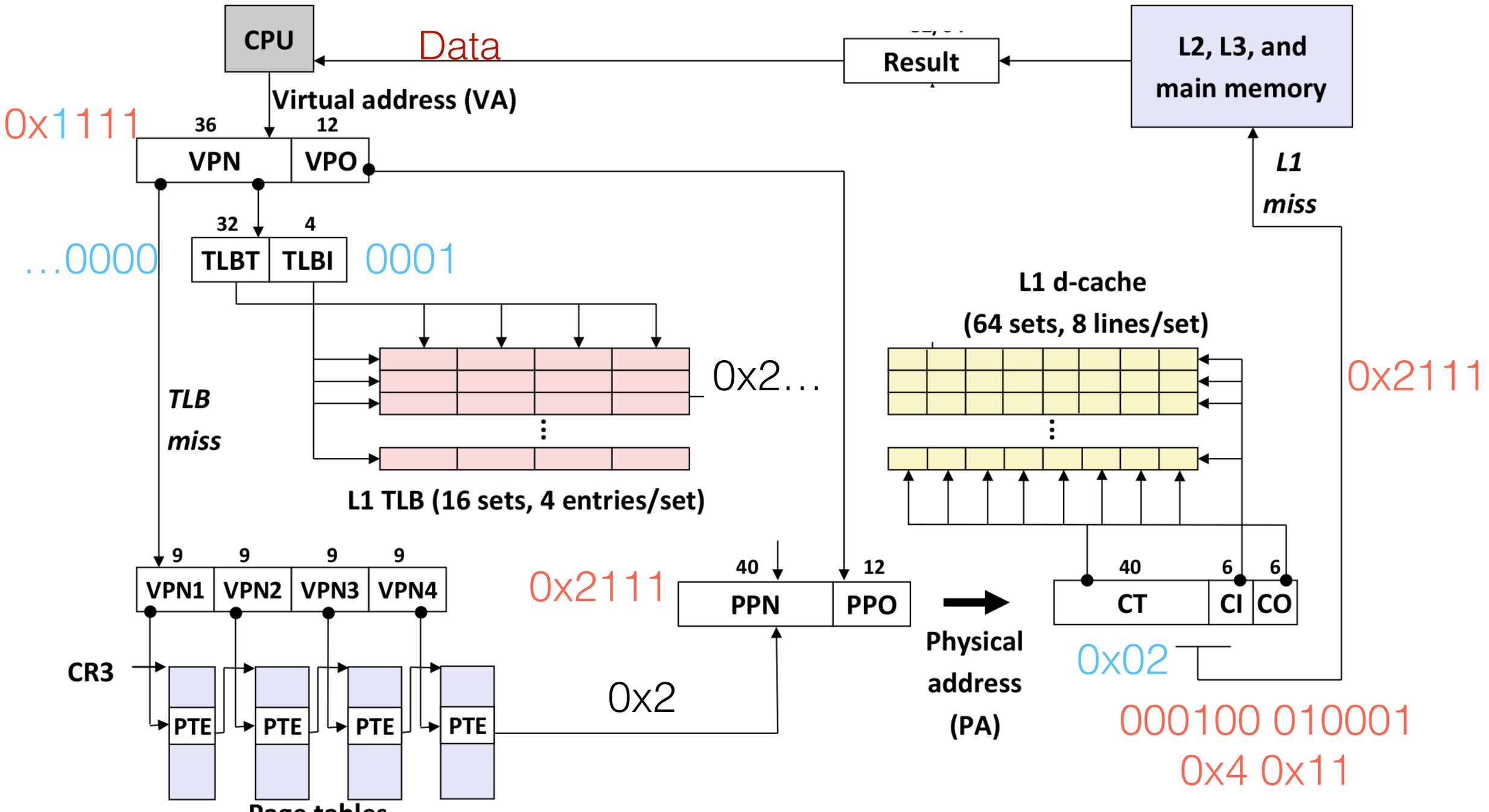


0x1111 → 0x2111 → 0xBAC

VA PA DATA

Description of Physical Memory

3) 8-way associative L1 cache with 64 sets and 64 Byte blocks



What happens if you access virtual address 0x1111 again?