Computer Structure

X86 Virtual Memory and TLB

Franck Sala

Updated by tomer gurevich

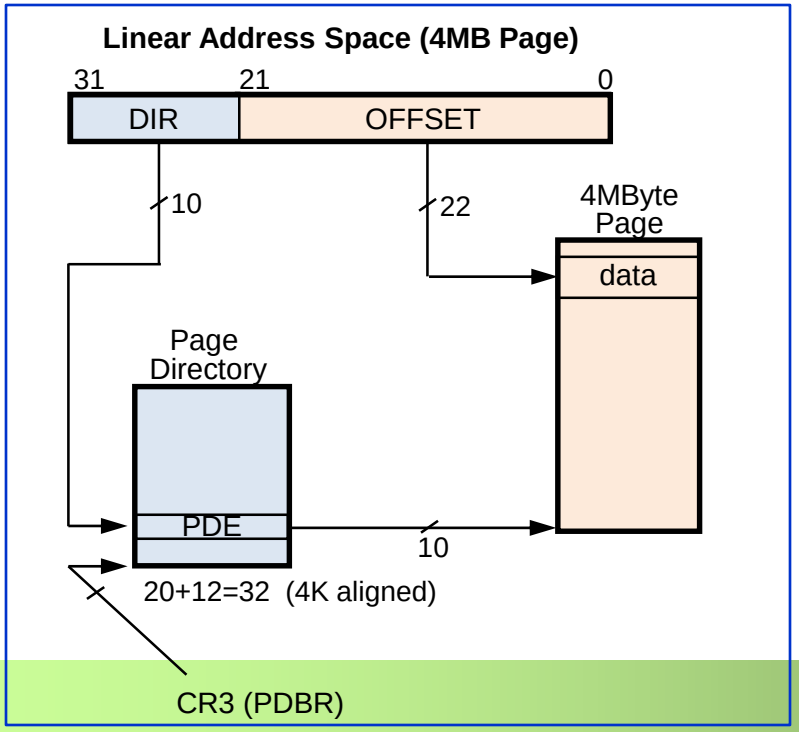
Slides from Lihu and Adi's Lecture

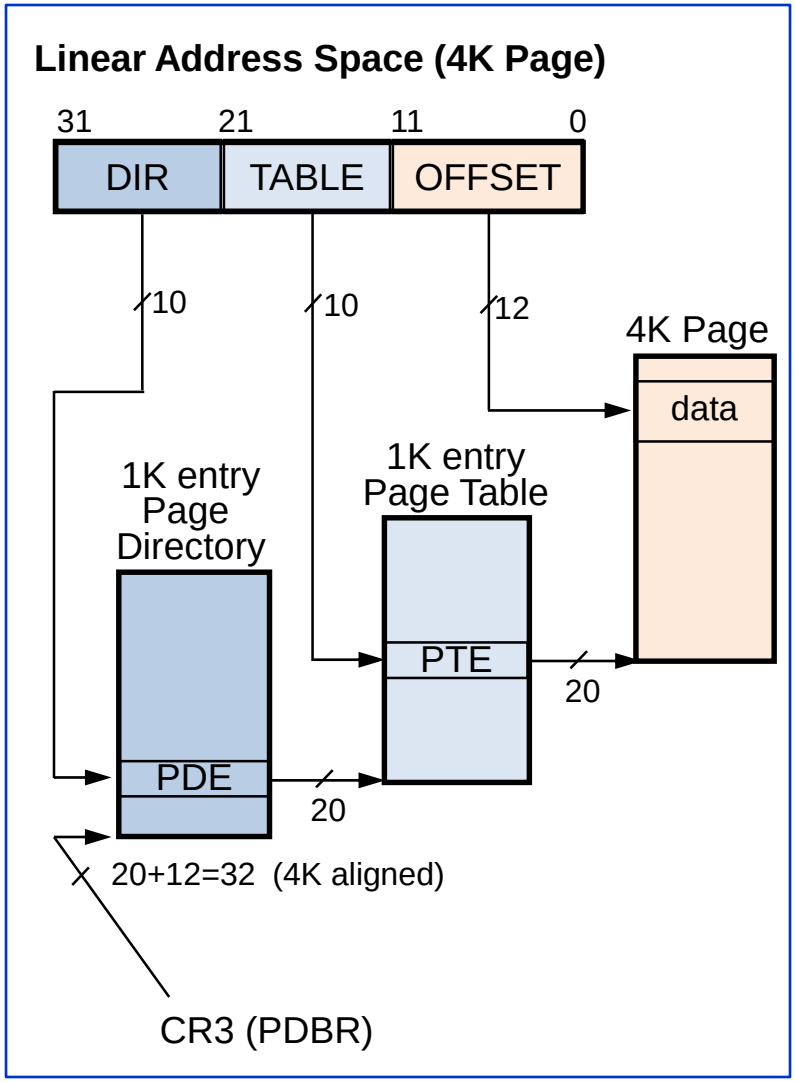
X86 paging

- עבור זיכרון עם כתובת בגודל של 32 ביט וגודל הדף הוא 4kb , גודל טבלת הדפים הדרושה הוא 4Mb .
- רוב התהליכים במערכת משתמשים רק במעט זיכרון. התקורה של 4Mb עבור כל תהליך היא לרוב יקרה ומיותרת. עבור רוב התהליכים טבלת תרגום כזאת תהיה רוב הזיכרון שהתהליך צורך.
- ב-X86 ישנן מספר רמות של טבלאות תרגום , המסודרות במבנה של עץ. אנו מקצים טבלאות תרגום באופן דינאמי , רק כאשר יש צורך ממשי בטבלה.

32bit Mode: 4KB / 4MB Page Mapping

- **2-level hierarchical mapping: Page Directories and Page tables**
  - 4KB aligned
- **PDE**
  - Present (0 = page fault)
  - Page size (4KB or 4 MB)
    - CR4.PSE=1 ⇒ both 4MB & 4KB pages supported
    - Separate TLBs

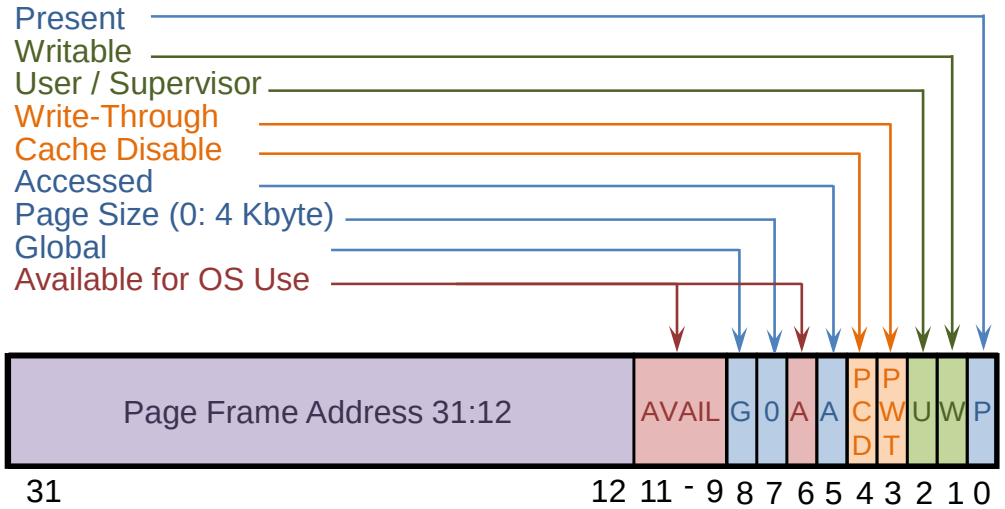




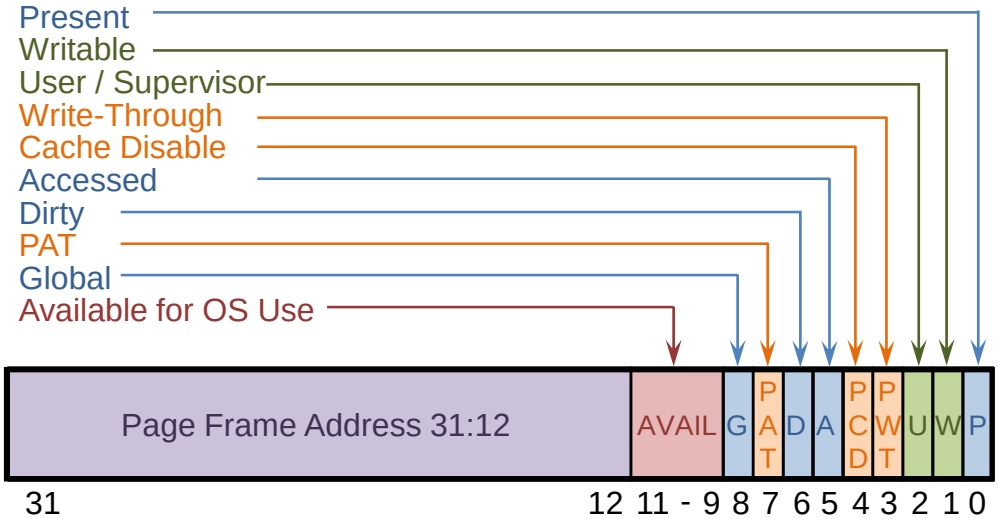
32bit Mode: PDE and PTE Format

- **20 bit pointer to a 4K Aligned address**
- **Virtual memory**
  - Present
  - Accessed
  - Dirty (in PTE only)
  - Page size (in PDE only)
  - Global
- **Protection**
  - Writable (R#/W)
  - User / Supervisor #
    - 2 levels/type only
- **Caching**
  - Page WT
  - Page Cache Disabled
  - PAT – PT Attribute Index
- **3 bits available for OS usage**

Page Directory Entry (4KB page table)

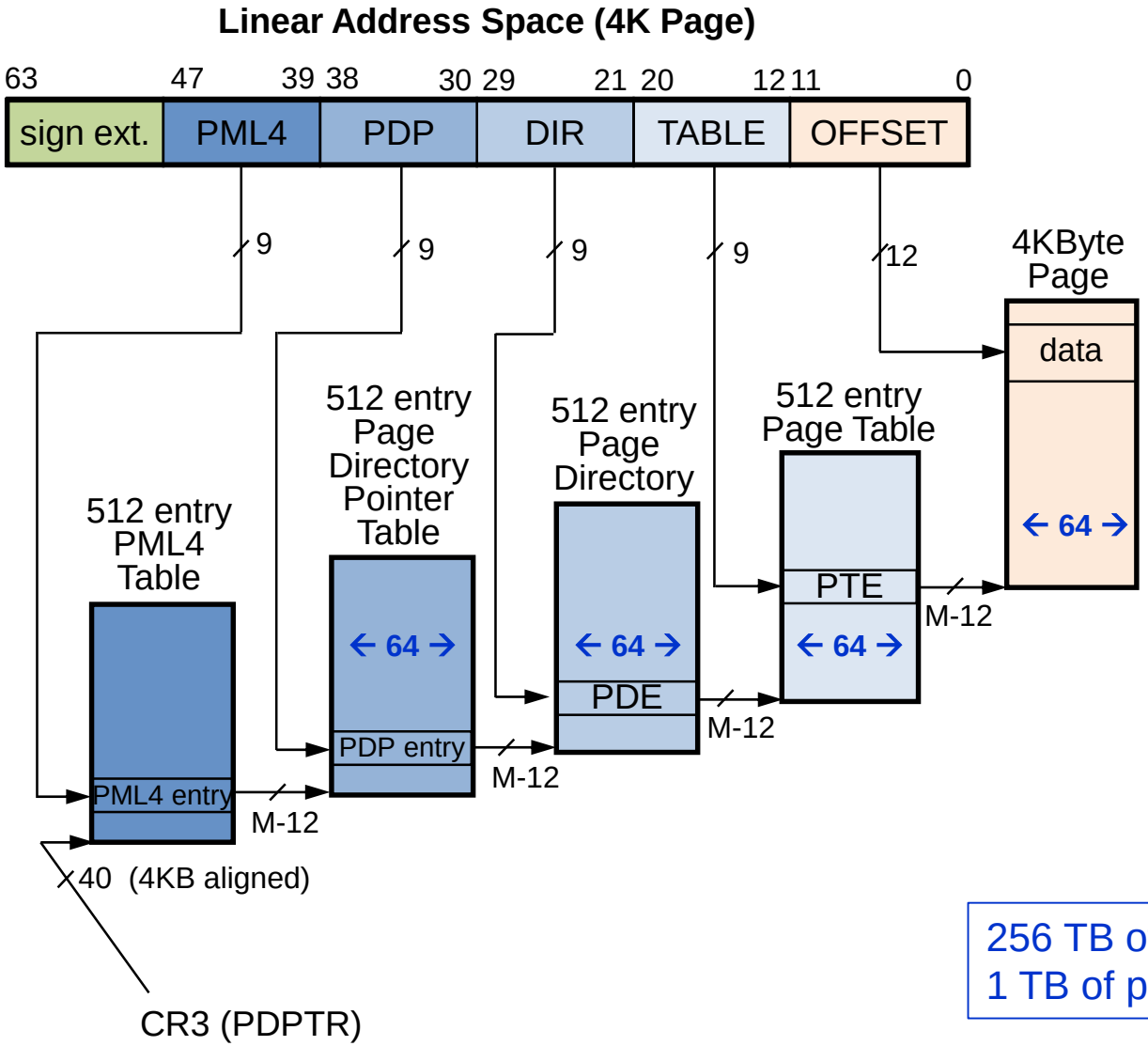


Page Table Entry



4KB Page Mapping in 64 bit Mode

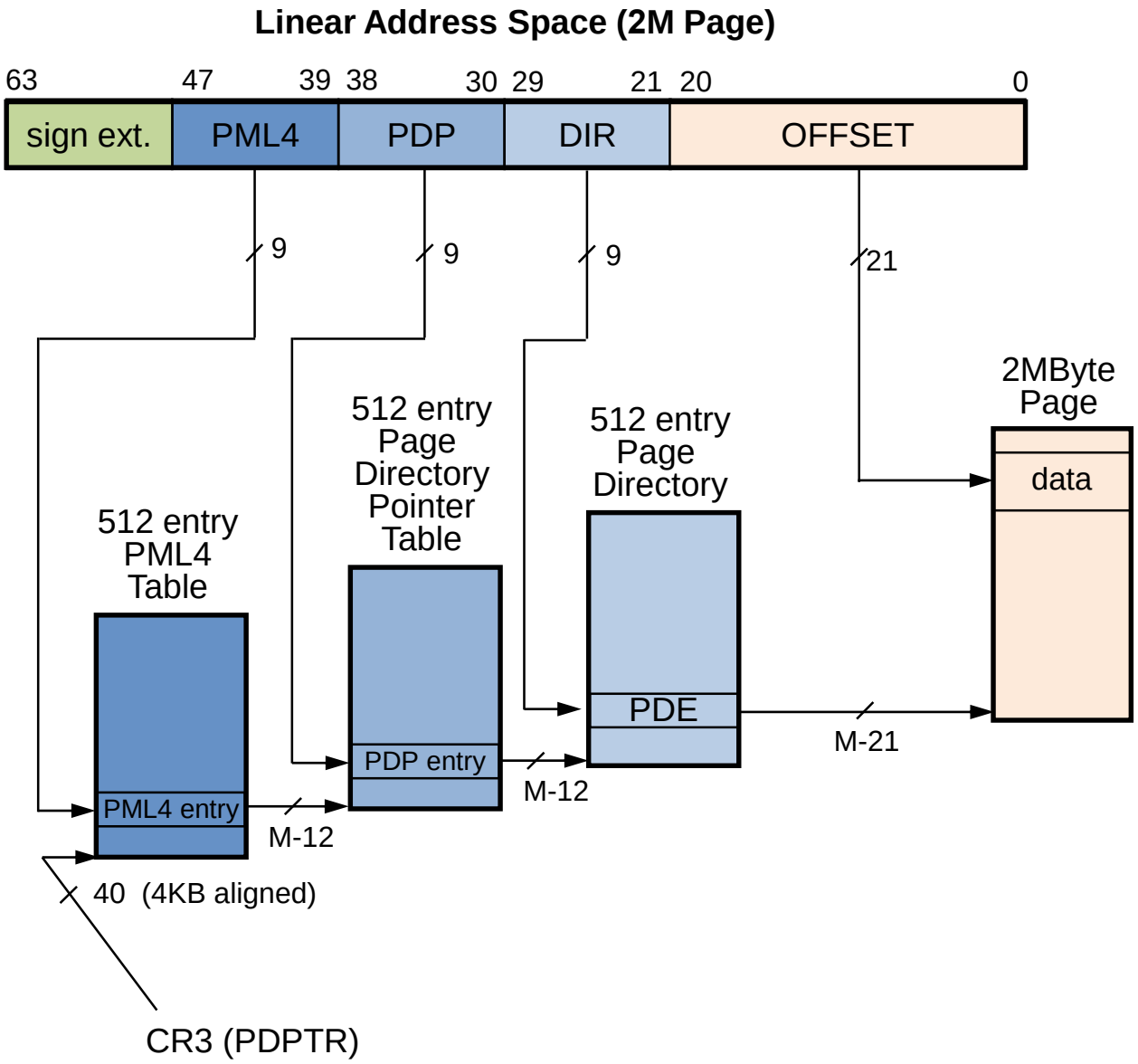
2003: AMD Opteron…



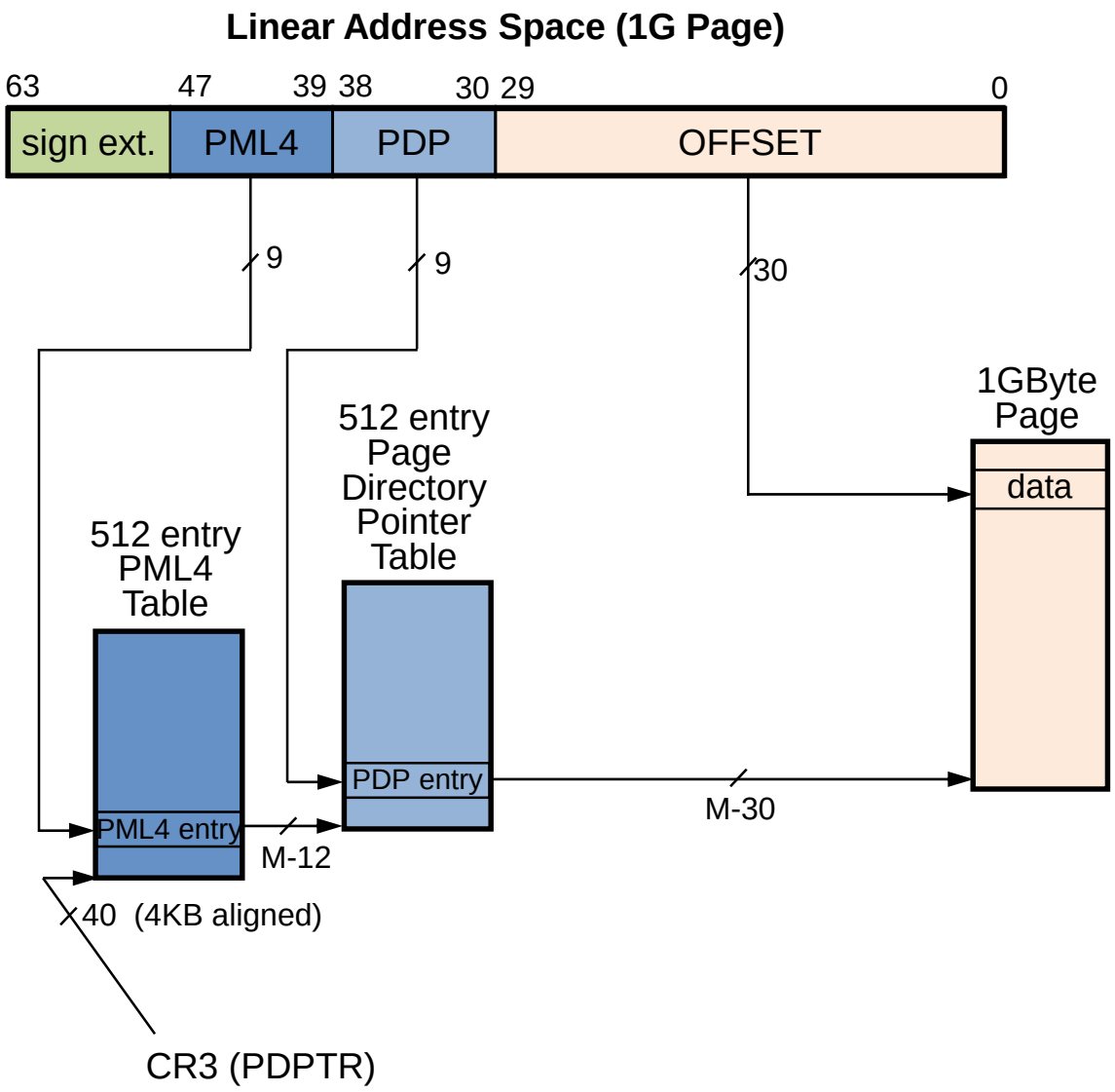
PML4: Page Map Level 4
PDP: Page Directory Pointer

256 TB of virtual memory (2^{48})
1 TB of physical memory (2^{40})

2MB Page Mapping in 64 bit Mode

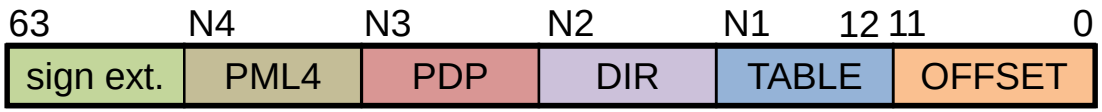


1GB Page Mapping in 64 bit Mode



Question 1

- **We have a core similar to X86**
  - 64 bit mode
  - Support Small Pages (PTE) and Large Pages (DIR)
  - Page table size in each hierarchy is the size of a small page
  - Entry size in the Page Table is 16 byte, in all the hierarchies



- What is the size of a small page ?

12 bits in the offset field

→ 2^{12} B = 4KB

- How many entries are in each Page Table?

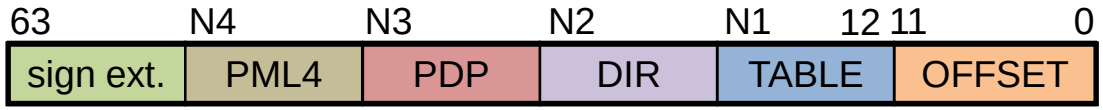
Page Table size = Page Size = 4KB

PTE = 16B

→ 4KB / 16B = 2^{12} / 2^{4} = 2^{8} = 256 entries in each Page Table

- 64 bit (large & small)
- PT size = Page size = 4KB
- PTE = 16B
- Page Table: 256 entries

Question 1



- What are the values of N1, N2, N3 and N4 ?

Since we have 256 entries in each table, we need 8 bits to address them

- Table [19:12] N1 = 19
- DIR [27:20] N2 = 27
- PDP [35:28] N3 = 35
- PML4 [43:36] N4 = 43

- What is the size of a large page ?

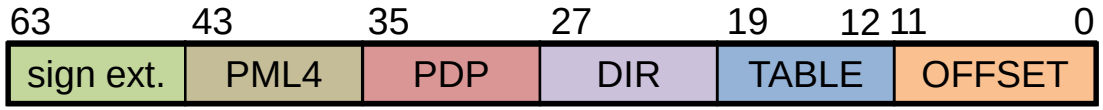
Large pages are pointed by DIR

So the large pages offset is 20 bits [19:0] → large pages size: 2^{20} = 1MB

We can also say: DIR can point to 256 pages of 4KB = 1MB

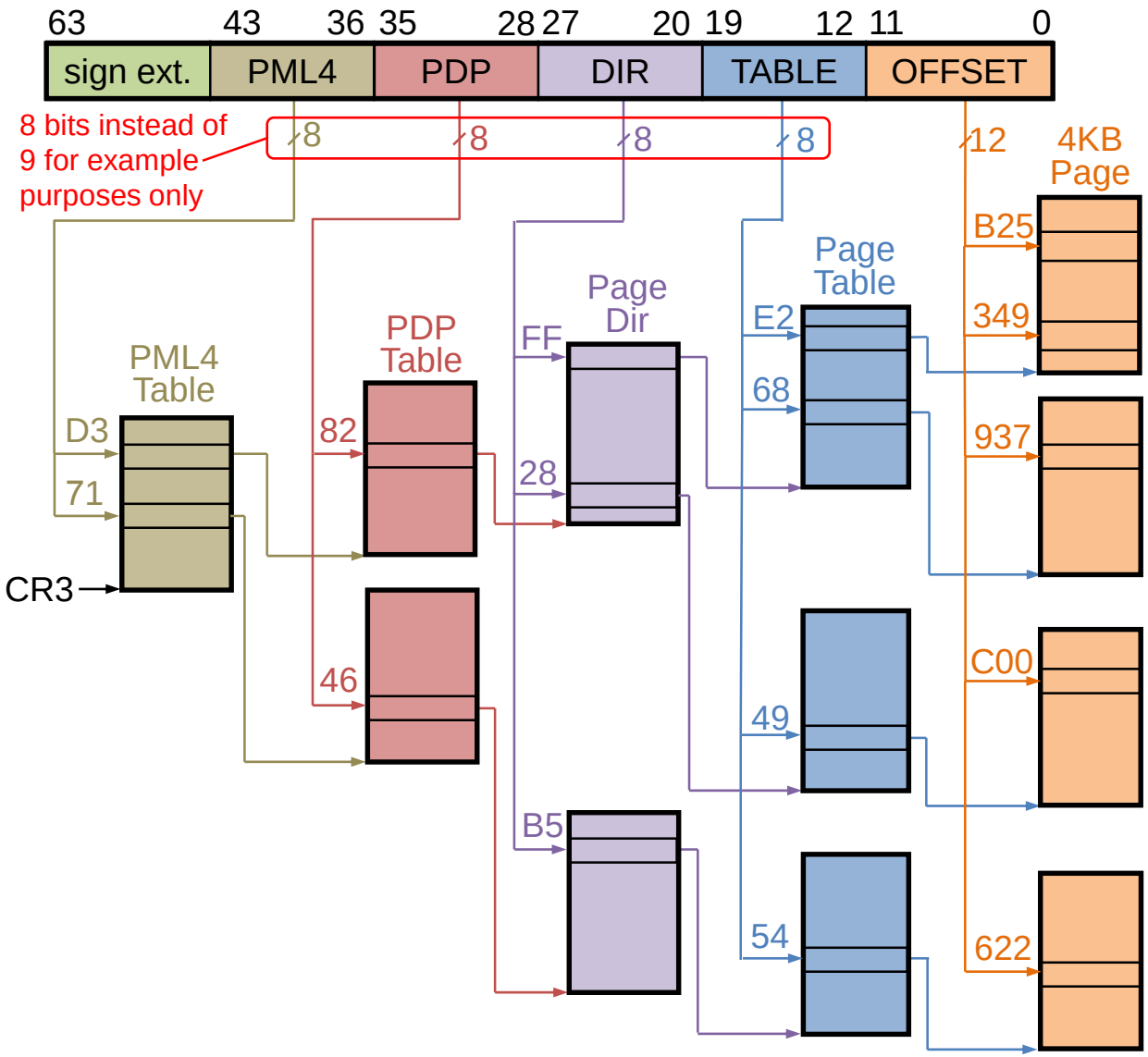
- 64 bit (large & small)
- PT size = Page size = 4KB
- Large page size = 1 MB
- PTE = 16B
- Page Table: 256 entries

Question 1



- We access a sequence of virtual addresses
  For each address, what is the minimal number of tables that were added in all the hierarchies ?
- *See next foil in presentation mode…*

Question 1: sequence of allocations



PML4 PDP DIR PTE offset

00000 D3 82 FF E2 B25
3 new tables + 1 page

00000 D3 82 FF E2 349
0 new table

00000 D3 82 FF 68 937
0 new table + 1 page

00000 D3 82 28 49 C00
1 new tables + 1 page

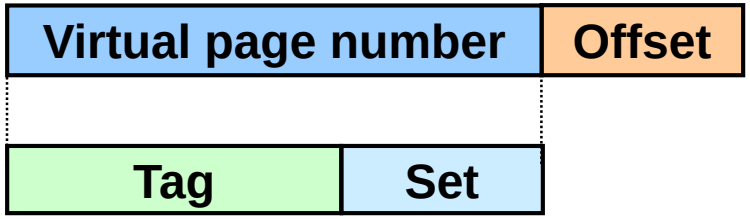
00000 71 46 B5 54 622
3 new tables + 1 page

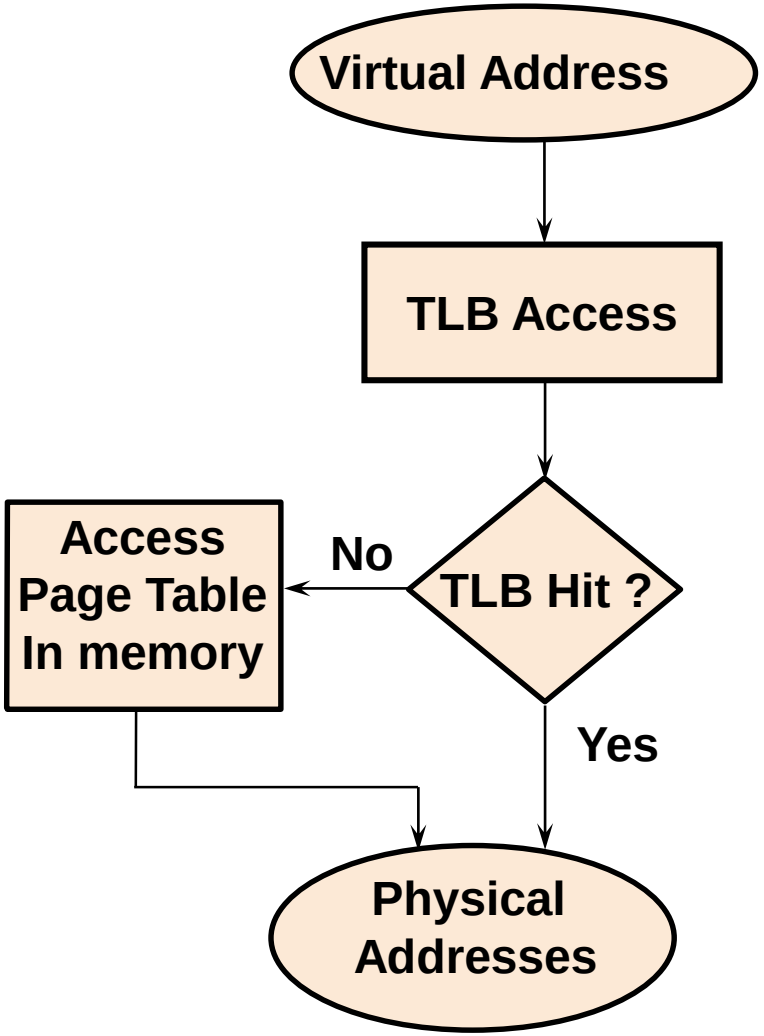
Translation Look aside Buffer (TLB)

- **Page table resides in memory**
  ⇒ each translation requires an extra memory access

- **TLB caches recently used PTEs**
  - speed up translation
  - typically 128 to 256 entries, 4 to 8 way associative

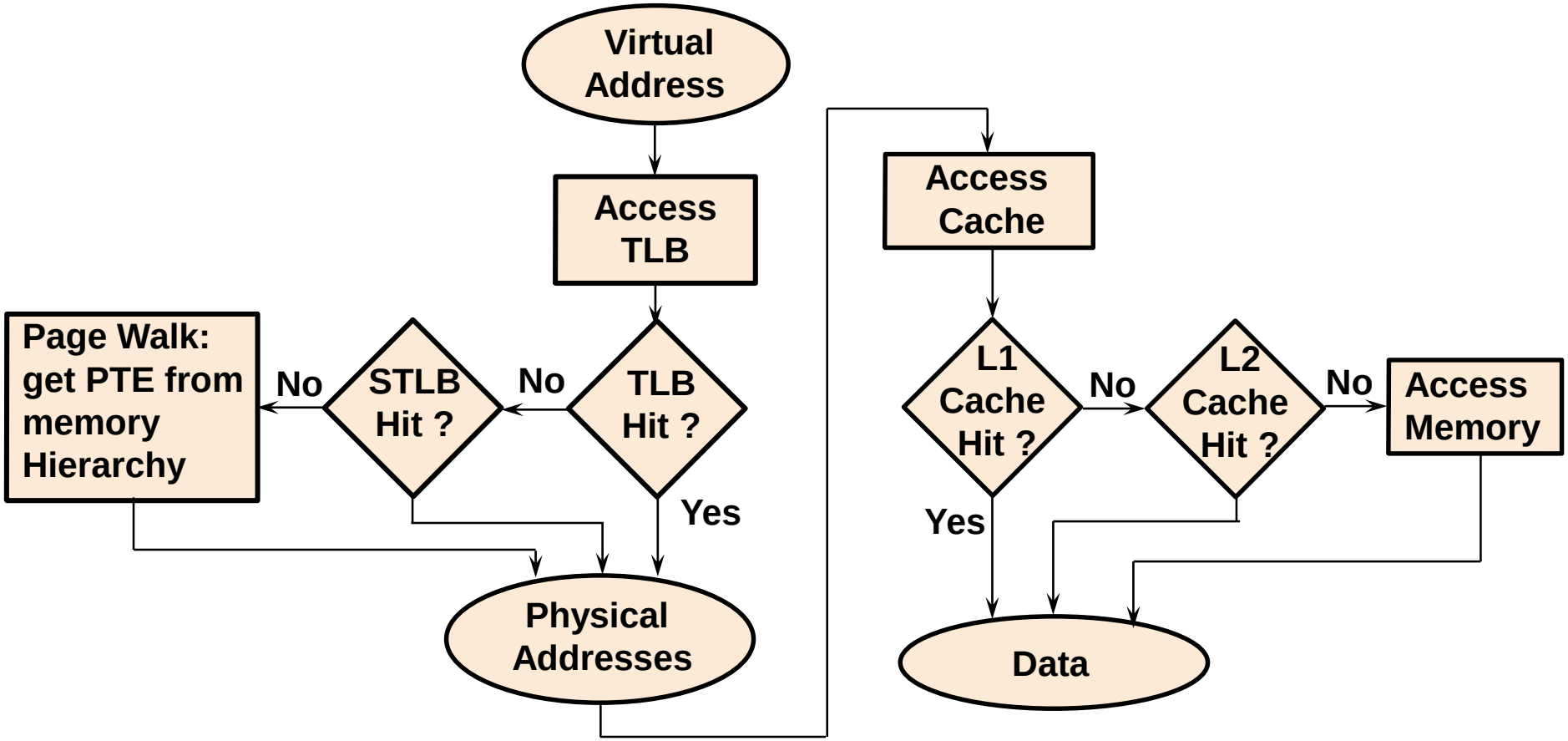
- **TLB Indexing**





- **On A TLB miss**
  - Page Miss Handler (HW PMH) gets PTE from memory

Virtual Memory And Cache



- TLB access is serial with cache access
- Page table entries are cached in L1 D$, L2$ and L3 $ as data

TLBs

- **The processor saves most recently used PDEs and PTEs in TLBs**
  - Separate TLB for data and instruction caches
  - Separate TLBs for 4KB and 2/4MB page sizes

PMH Page miss handler

- כאשר ישנה גישה לזיכרון מתרחש התהליך הבא:
- ראשית , ניגש ל- TLB המתאים עם ה- VPN המלא.
- ישנם dTLB ו – iTLB אשר מכילים תרגום של מידע וזיכרון בהתאמה.
- ה- TLBs מחולקים גם לדפים קטנים וגדולים בהתאמה.
- אם ישנו TLB HIT , נשתמש בתרגום שמצאנו .
- במקרה של TLB MISS נפנה ל- PMH

STLB –secondary TLB

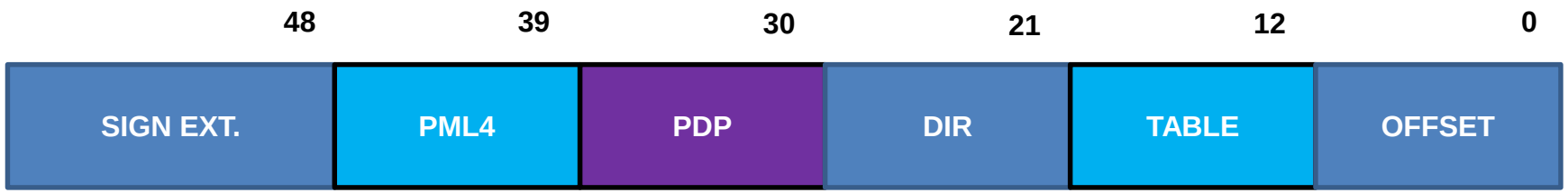
- ה – PMH מכיל את ה- STLB .
- לאחר ה- TLB MISS ניגש ל- STLB . ה-STLB מהווה עוד "רמה " של TLB . הוא מכיל יותר PTE גם הוא.
- נפנה אליו לאחר ה- TLB MISS ..
- ה- STLB מכיל גם זיכרון של פקודות וגם של מידע. כמו כן , הוא מכיל גם תרגומים של דפים גדולים.
- כמו ב- TLB אנו משתמשים ב- VPN כדי לחפש ב-STLB .
- עבור STLB HIT , נשתמש בתרגום שמצאנו
- עבור STLB MISS, ה- PMH יבצע Page walk .

Page Walk

- בשלב הזה ,ה-PMH חייב לבצע Page Walk : הוא מטייל על הירארכיית טבלאות הדפים החל מהשורש (PML4) .
- כדי לקצר את התהליך ה- PMH שומר cache עבור הרמות הגבוהות של התרגום: PML4 cache,PDP cache,DIR cache .
- מדוע אין צורך לשמור Table cache ?

מכיוון שה-TLB שומר תרגום מלא של כתובת, אין צורך ב- Table Cache אלא רק עבור הרמות הגבוהות יותר, אשר מהוות תרגום חלקי.

Page Walk



cache	Accessed with virtual address bits	If hits, returns
DIR cache	[47:21]	PDE
PDP cache	[47:30]	PDP entry
PML4 cache	[47:39]	PML4 entry

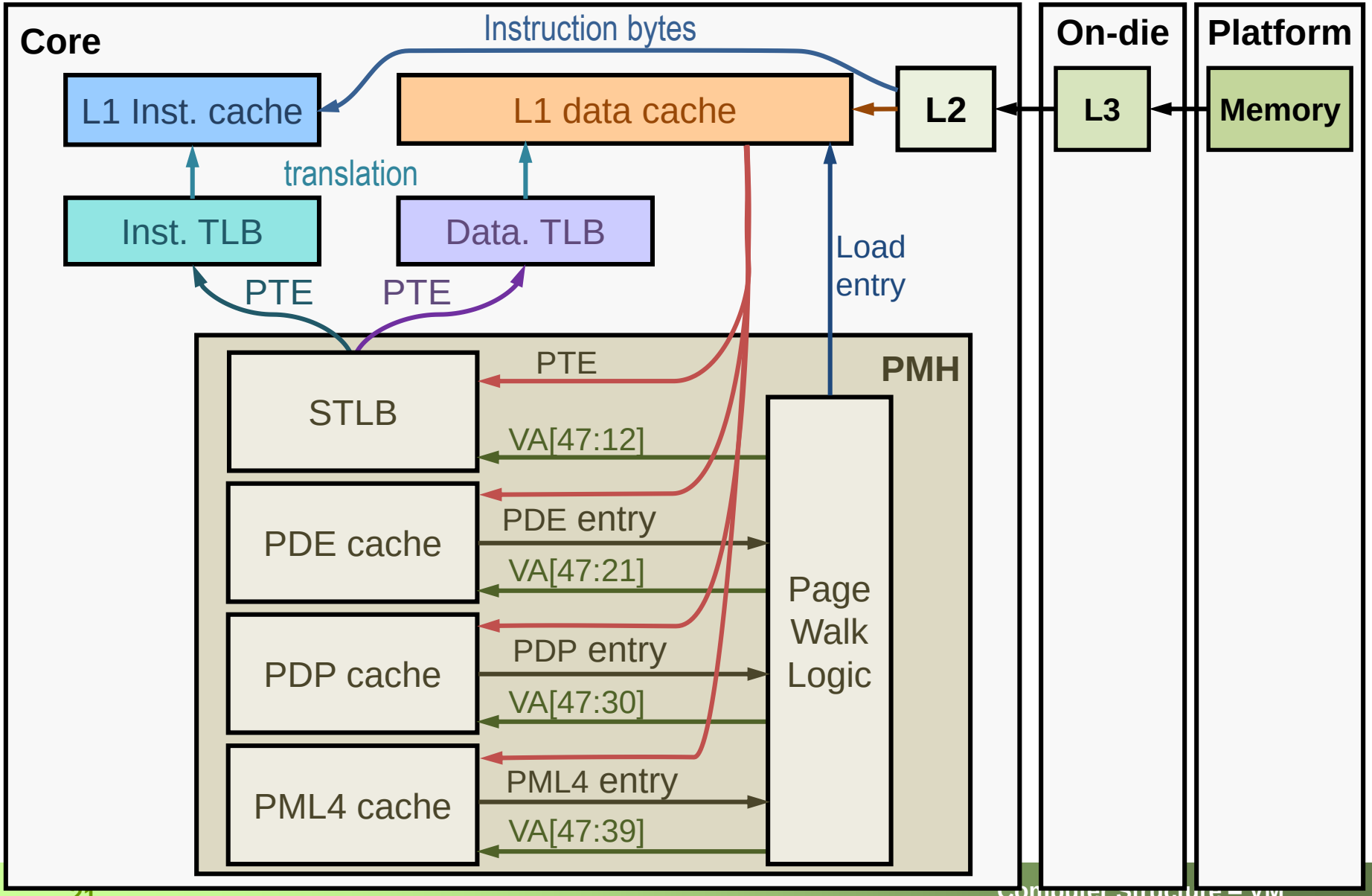
Page Walk

- אל כל cache אנו פונים עם הסיביות ששימשו אותנו גם לפנייה לרמה גבוהה יותר .
- אם ברמה כלשהיא של ה- cache התרחשה פגיעה. אזי הצלחנו לחסוך (לפחות) את כל הגישות לרמה הנוכחית ולרמות הגבוהות יותר.
- לדוגמא , עבור PDP cache hit נמצא PDPE מתאים וכך נחסוך פנייה ל- PML4 ו – PDP . עדיין נצטרך לגשת לזיכרון עבור הרמות הנמוכות יותר.

Page Walk

- ה-PMH ניגש לכל המטמונים במקביל ובוחר ברמה הנמוכה ביותר עבורה היה HIT .
- את שארית התרגום נבצע כרגיל , באמצעות גישה לטבלאות אשר שמורות בזיכרון .
- נשים לב לכל הפחות נהיה חייבים לגשת ל-PAGE TABLE .
- את טבלת הדפים המתאימה נחפש בהיראכיית הזיכרון כרגיל: ניגש קודם ל-L1 cache , L2 cache , L3 cache ורק לבסוף ניגש לזיכרון .

Caches and Translation Structures



Question 2

- **Processor similar to X86 – 64 bits**
- **Pages of 4KB**
- **The processor has a TLB**
  - TLB Hit: we get the translation with no need to access the translation tables
  - TLB Miss: the processor has to do a Page Walk
- **The hardware that does the Page Walk (PMH) contains a cache for each of the translation tables**
- **All Caches and TLB are empty on Reset**

63	43	35	27	19	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

- **For the sequence of memory access below, how many accesses are needed for the translations?**

Address	memory access	Explanations
0000022334455666H		
0000022334455777H		
0000022884455777H		

Question 2

- **Processor similar to X86 – 64 bits**
- **Pages of 4KB**
- **The processor has a TLB**
  - TLB Hit: we get the translation with no need to access the translation tables
  - TLB Miss: the processor has to do a Page Walk
- **The hardware that does the Page Walk (PMH) contains a cache for each of the translation tables**
- **All Caches and TLB are empty on Reset**

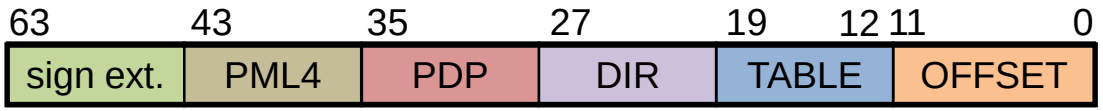
63	43	35	27	19	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

- **For the sequence of memory access below, how many accesses are needed for the translations?**

Address	memory access	Explanations
00000223344556 66H		
00000223344557 77H		
00000228844557 77H		

Question 2

- **Processor similar to X86 – 64 bits**
- **Pages of 4KB**
- **The processor has a TLB**
  - TLB Hit: we get the translation with no need to access the translation tables
  - TLB Miss: the processor has to do a Page Walk
- **The hardware that does the Page Walk (PMH) contains a cache for each of the translation tables**
- **All Caches and TLB are empty on Reset**



- **For the sequence of memory access below, how many accesses are needed for the translations?**

Address	memory access	Explanations
00000223344556 66H	4	We need to access the memory for each of the 4 translation tables
00000223344557 77H		
00000228844557 77H		

Question 2

- **Processor similar to X86 – 64 bits**
- **Pages of 4KB**
- **The processor has a TLB**
  - TLB Hit: we get the translation with no need to access the translation tables
  - TLB Miss: the processor has to do a Page Walk
- **The hardware that does the Page Walk (PMH) contains a cache for each of the translation tables**
- **All Caches and TLB are empty on Reset**

63	43	35	27	19	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

- **For the sequence of memory access below, how many accesses are needed for the translations?**

Address	memory access	Explanations
00000223344556 66H	4	We need to access the memory for each of the 4 translation tables
00000223344557 77H	0	Same pages as above → TLB hit → No memory access
00000228844557 77H		

Question 2

- **Processor similar to X86 – 64 bits**
- **Pages of 4KB**
- **The processor has a TLB**
  - TLB Hit: we get the translation with no need to access the translation tables
  - TLB Miss: the processor has to do a Page Walk

63	43	35	27	19	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

- **The hardware that does the Page Walk (PMH) contains a cache for each of the translation tables**
- **All Caches and TLB are empty on Reset**
- **For the sequence of memory access below, how many accesses are needed for the translations?**

Address	memory access	Explanations
00000223344556 66H	4	We need to access the memory for each of the 4 translation tables
00000223344557 77H	0	Same pages as above → TLB hit → No memory access
00000228844557 77H	3	We hit in PML4 cache. Then we miss in the PDP and so we need to access the memory 3 times: PDP, DIR, PTE

Question 2

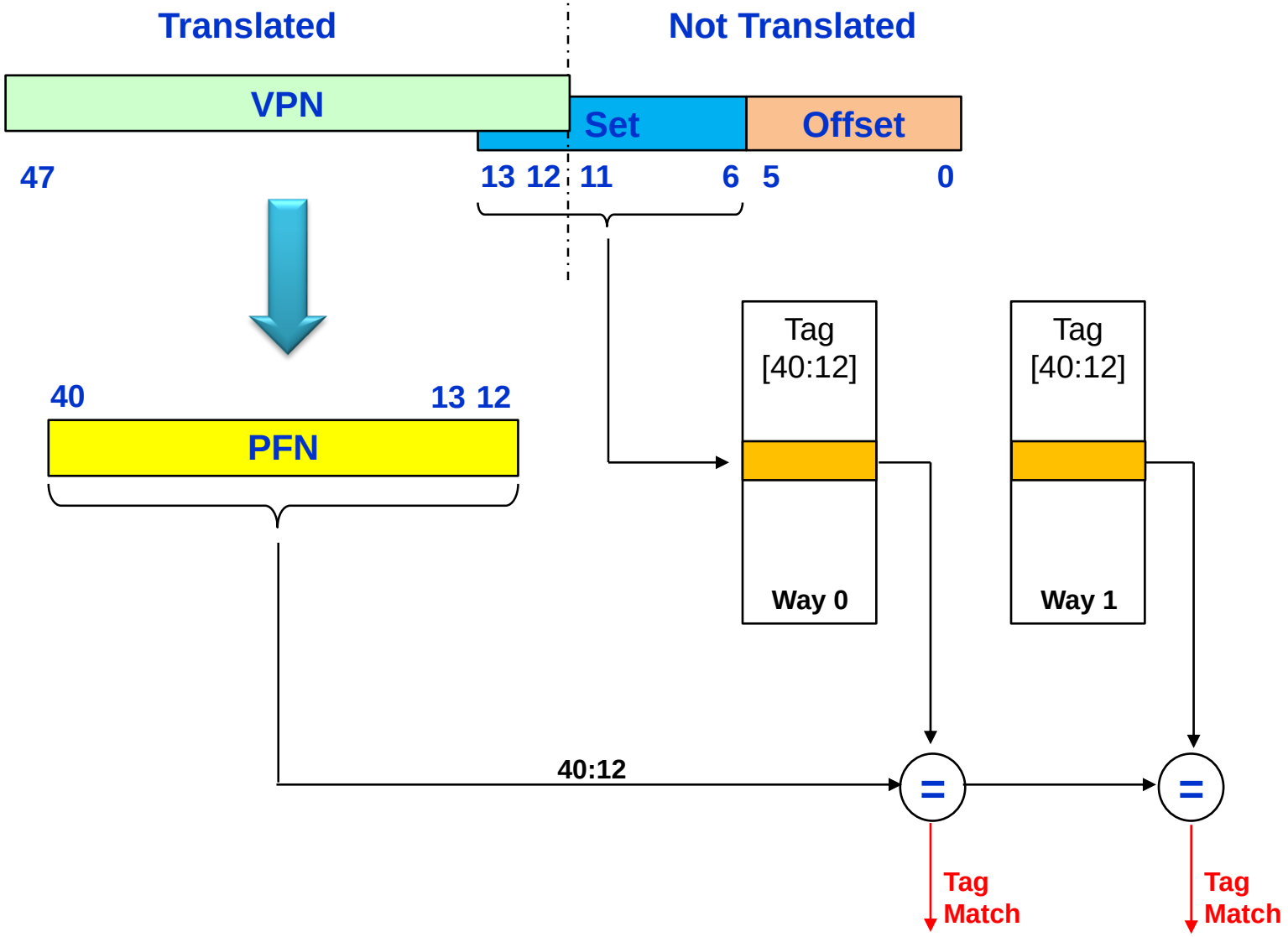
- **L1 data Cache: 32KB – 2 ways of 64B each**
- **How can we access this cache before we get the physical address?**

64B → 6 bits offset bits [5:0]
32KB = 2^{15} / (2 ways * 2^6 bytes) = 2^8 = 256 sets [13:6]

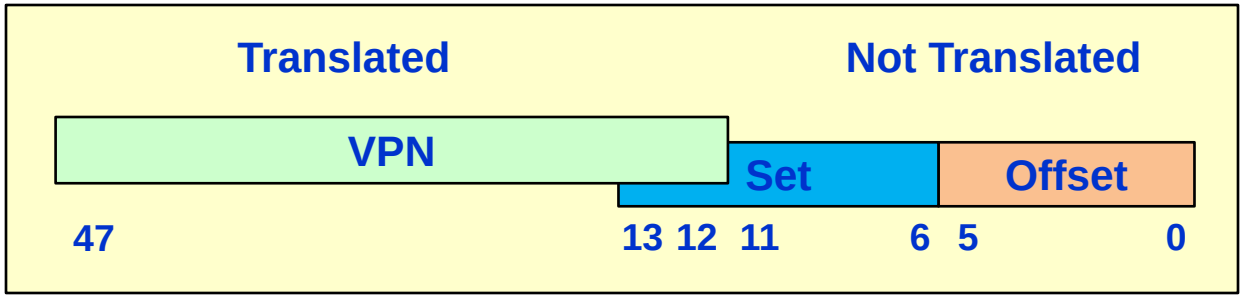
→ 12 bits are not translated: [11:0]
→ we lack 2 bits [13:12] to get the set address
→ So we do a lookup using 2 un-translated bits for the set address

→ Those bits can be different from the PFN obtained after translation, therefore we need to compare the whole PFN to the tag stored in the Cache Tag array

Question 2: Read Acces



Question 2: example



Write Virtual Address A

	VPN	Offset
VA:	0000 … 0000	0000 0000 0000

Set: [13:6] = 0

	PFN	Offset
PA:	0 … 0000	0000 0000 0000

Tag: 0

Read Virtual Address B

VB:	0011 … 0000	0000 0000 0000

Set: [13:6] = 0

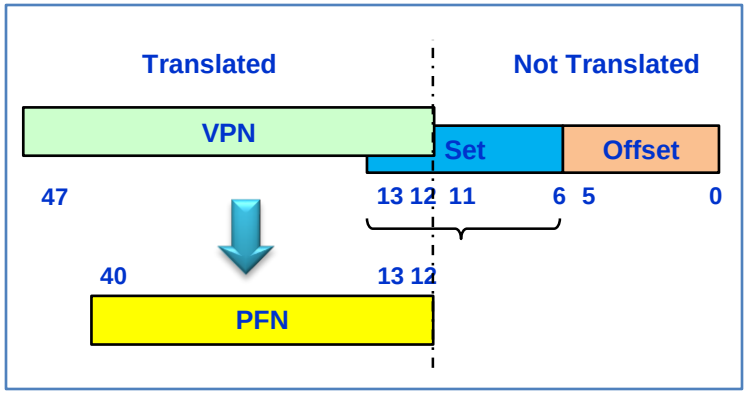
PB:	0 … 0011	0000 0000 0000

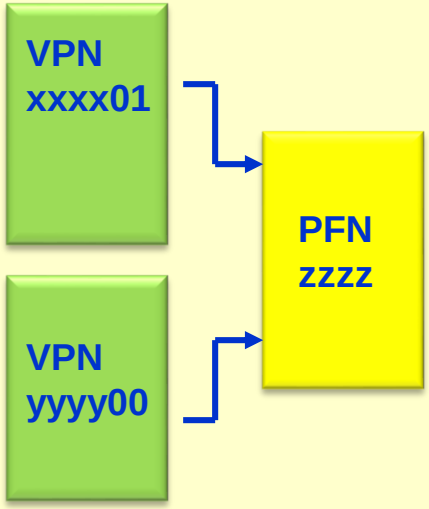
Tag: 3

(0 if we don't take [13:12])

Question 2: Virtual Alias

- **L1 data Cache: 32KB**
- **2 ways of 64B each**
- **What will happen when we access with a given offset the virtual page A and after this, there is an access with the same offset in the virtual page B, which is mapped by the OS to the same physical page as A?**

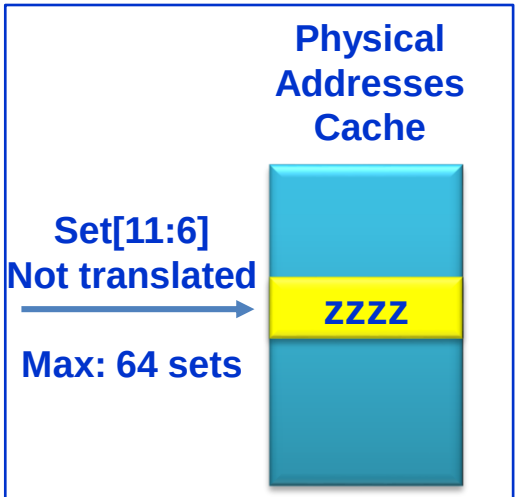




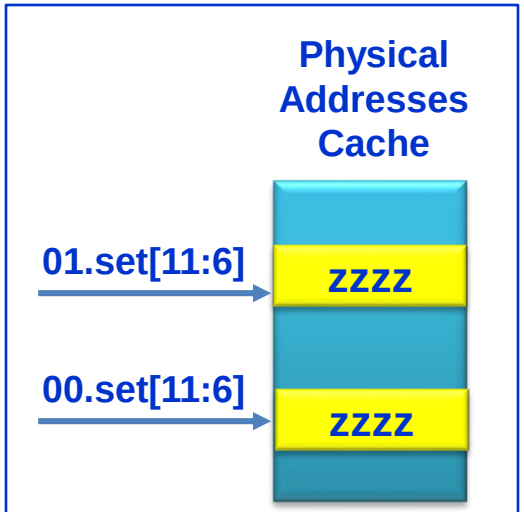
- 2 virtual pages map to the same frame

xxxx01.set.ofset and **yyyy00.set.ofset**

01 and 00 are bits 13:12



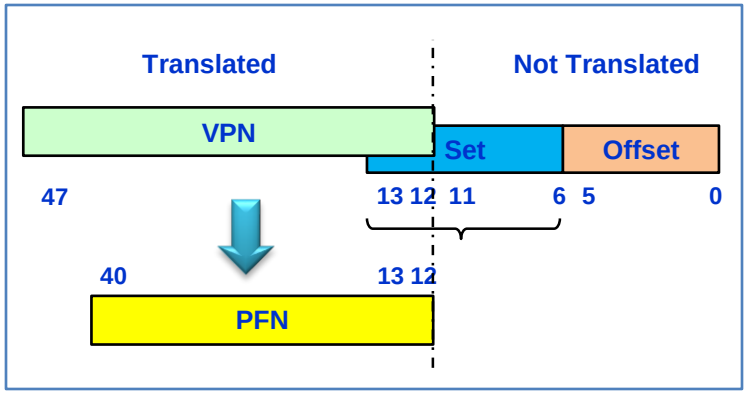
Phys. addressed cache:
The data exist only once



Virtual addressed cache:
The data may exist twice
AVOID THIS !!!

Question 2: Virtual Alias

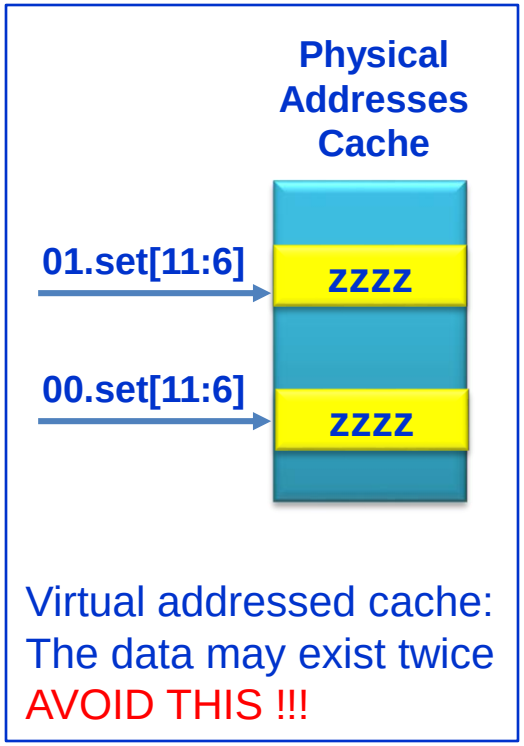
- **L1 data Cache: 32KB**
- **2 ways of 64B each**
- **What will happen when we access with a given offset the virtual page A and after this, there is an access with the same offset in the virtual page B, which is mapped by the OS to the same physical page as A?**



- Avoid having the same data twice in the cache

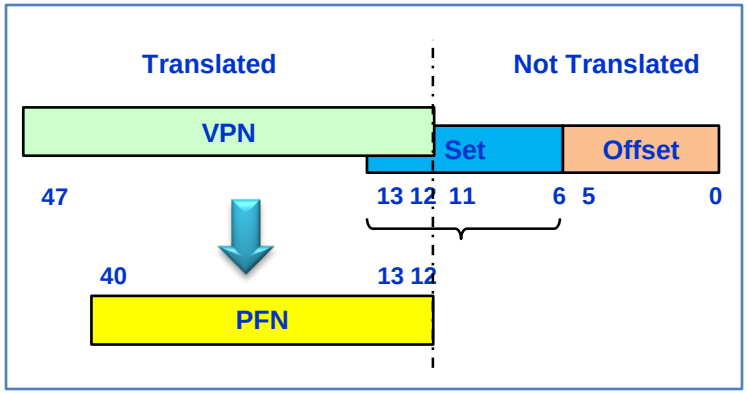
xxxx01.set.ofset and **yyyy00.set.ofset**

- Check 4 sets when we allocate a new entry and see if the same tag appears
- If yes, evict the second occurrence of the data (the alias)



Question 2: Snoop

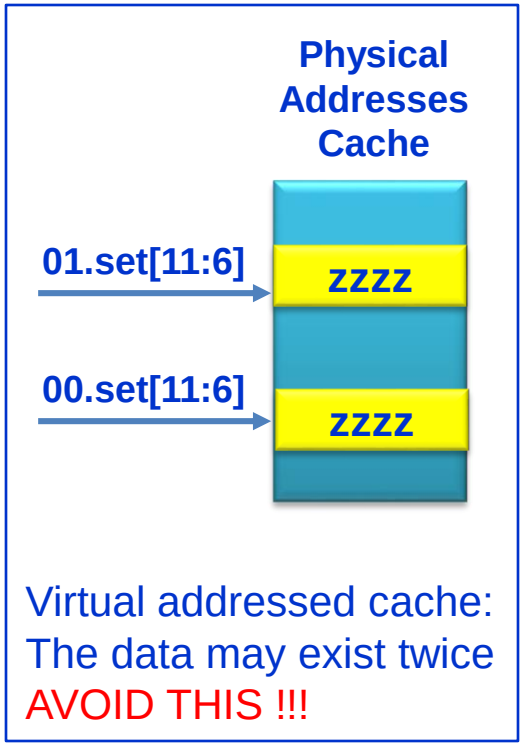
- **L1 data Cache: 32KB**
- **2 ways of 64B each**
- **What happens in case of snoop in the cache?**



The cache is snooped with a physical address [40:0]

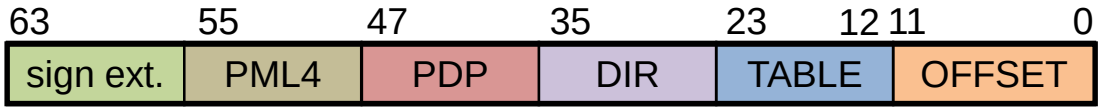
Since the 2 MSB bits of the set address are virtual, a given physical address can map to 4 different sets in the cache (depending on the virtual page that is mapped to it)

So we must snoop 4 sets * 2 ways in the cache



Question 3

- **Core similar to X86 in 64 bit mode**



- **Supports small pages (pointed by PTE) and large pages (pointed by DIR).**
- **Size of an entry in all the different page tables is 8 Bytes**
- **PMH Caches at all the levels**
  - 4 entries direct mapped
  - Access time on hit: 2 cycles
  - Miss known after 1 cycle
- **PMH caches are accessed at all the levels in parallel**
- **In each level, when there is a HIT, the PMH cache provides the relevant entry in the page table in the relevant level**
- **In each level, when there is a miss: the core accesses the relevant page table in the main memory.**

- **Access time to the main memory is 100 cycles, not including the time needed to get the PMH cache miss.**

Question 3

63	55	47	35	23	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

- **What is the size of the large pages?**

The large page is pointed by DIR, therefore, all the bits under are offset inside the large page: 2^{24} = 16 MB

- **How many entries in each Page Table ?**

- **PTE:** 2^{12}
- **DIR:** 2^{12}
- **PDP:** 2^{12}
- **PML4:** 2^{8}

63	55	47	35	23	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

TLB 4entries
Direct mapped
TLB hit: 2 cycles
TLB miss: 1 cycle
Memory access: 100 cycle

Virtual Addr.	Cycles	Comment
FF81 2345 6789 ABCD		
FF81 2340 6789 ABCD		
FF80 2340 6789 ABCD		
FF81 2340 6709 ABCD		
FF81 2340 6709 A0CD		

63	55	47	35	23	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

TLB 4entries
Direct mapped
TLB hit: 2 cycles
TLB miss: 1 cycle
Memory access: 100 cycle

Virtual Addr.	Cycles	Comment
FF81 2345 6789 ABCD	401	Miss and memory access at each level: 1+ 4 *100 = 401 cycles
FF81 2340 6789 ABCD		
FF80 2340 6789 ABCD		
FF81 2340 6709 ABCD		
FF81 2340 6709 A0CD		

63	55	47	35	23	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

TLB 4entries
Direct mapped
TLB hit: 2 cycles
TLB miss: 1 cycle
Memory access: 100 cycle

Virtual Addr.	Cycles	Comment
FF81 2345 6789 ABCD	401	Miss and memory access at each level: 1+ 4 *100 = 401 cycles
FF81 2340 6789 ABCD	202	(PML4, PDP, DIR, sTLB) = (H,H,M,M) 1cyc PDP TLB read 1 more cycle DIR TLB: Miss 100 cycles PTE TLB: Miss 100 cycles
FF80 2340 6789 ABCD		
FF81 2340 6709 ABCD		
FF81 2340 6709 A0CD		

63	55	47	35	23	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

TLB 4entries
Direct mapped
TLB hit: 2 cycles
TLB miss: 1 cycle
Memory access: 100 cycle

Virtual Addr.	Cycles	Comment
FF81 2345 6789 ABCD	401	Miss and memory access at each level: 1+ 4 *100 = 401 cycles
FF81 2340 6789 ABCD	202	(PML4, PDP, DIR, sTLB) = (H,H,M,M) 1cyc PDP TLB read 1 more cycle DIR TLB: Miss 100 cycles PTE TLB: Miss 100 cycles
FF80 2340 6789 ABCD	401	PMH cache miss in all the levels: 1 + 4*100 = 401 cycles
FF81 2340 6709 ABCD		
FF81 2340 6709 A0CD		

63	55	47	35	23	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

TLB 4entries
Direct mapped
TLB hit: 2 cycles
TLB miss: 1 cycle
Memory access: 100 cycle

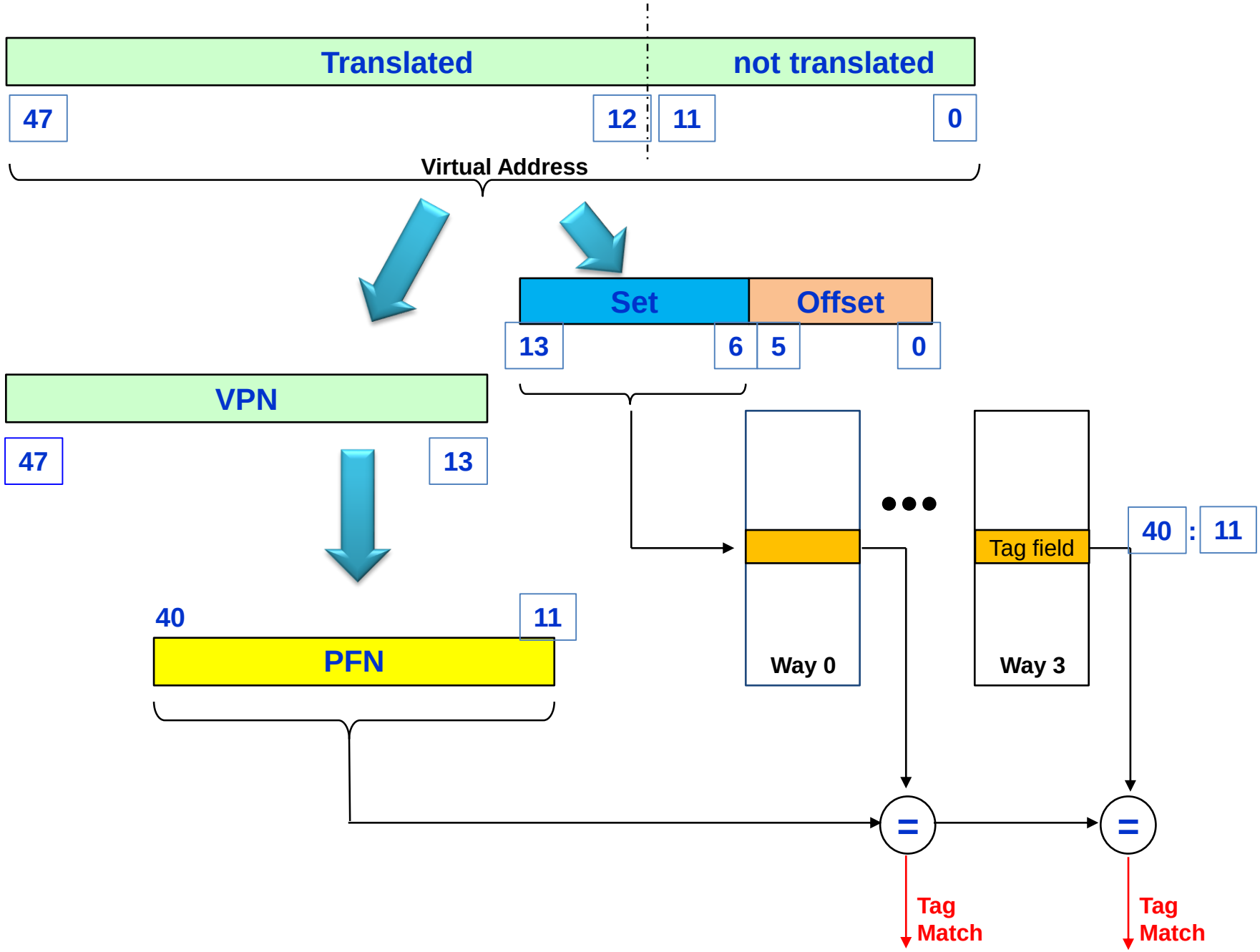
Virtual Addr.	Cycles	Comment
FF81 2345 6789 ABCD	401	Miss and memory access at each level: 1+ 4 *100 = 401 cycles
FF81 2340 6789 ABCD	202	(PML4, PDP, DIR, sTLB) = (H,H,M,M) 1cyc PDP TLB read 1 more cycle DIR TLB: Miss 100 cycles PTE TLB: Miss 100 cycles
FF80 2340 6789 ABCD	401	PMH cache miss in all the levels: 1 + 4*100 = 401 cycles
FF81 2340 6709 ABCD	302	(PML4, PDP, DIR, sTLB) = (H,M,M,M) 1 cyc PDP: the entry 234 that was filled for the second access was replaced by the entry that was filled in access 3, as it is in the same set → miss 2 + (3 × 100) = 302
FF81 2340 6709 A0CD		

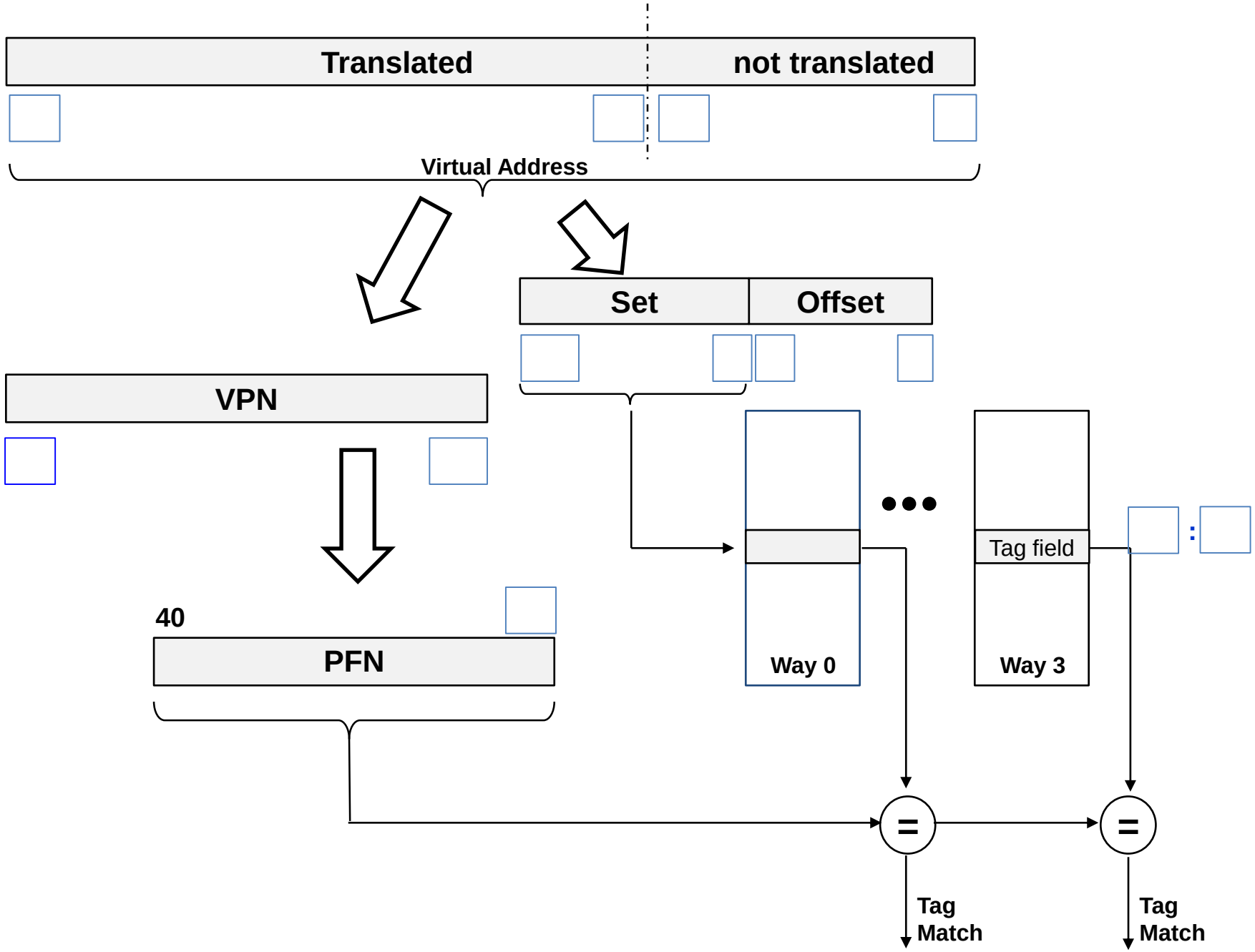
63	55	47	35	23	12 11	0
sign ext.	PML4	PDP	DIR	TABLE	OFFSET	

TLB 4entries
Direct mapped
TLB hit: 2 cycles
TLB miss: 1 cycle
Memory access: 100 cycle

Virtual Addr.	Cycles	Comment
FF81 2345 6789 ABCD	401	Miss and memory access at each level: 1+ 4 *100 = 401 cycles
FF81 2340 6789 ABCD	202	(PML4, PDP, DIR, sTLB) = (H,H,M,M) 1cyc PDP TLB read 1 more cycle DIR TLB: Miss 100 cycles PTE TLB: Miss 100 cycles
FF80 2340 6789 ABCD	401	PMH cache miss in all the levels: 1 + 4*100 = 401 cycles
FF81 2340 6709 ABCD	302	(PML4, PDP, DIR, sTLB) = (H,M,M,M) 1 cyc PDP: the entry 234 that was filled for the second access was replaced by the entry that was filled in access 3, as it is in the same set → miss 2 + (3 × 100) = 302
FF81 2340 6709 A0CD	2	Hit in TLB – no need to go to PMH: 2 cycles

Backup Slides


Translated
not translated
47
12
11
0
Virtual Address
Set
Offset
13
6
5
0
VPN
47
13
40
11
PFN
Way 0
Way 3
Tag field
40
:
11
=
=
Tag
Match
Tag
Match


Translated
not translated
Virtual Address
Set
Offset
VPN
40
PFN
Way 0
Way 3
Tag field
Tag
Match
Tag
Match