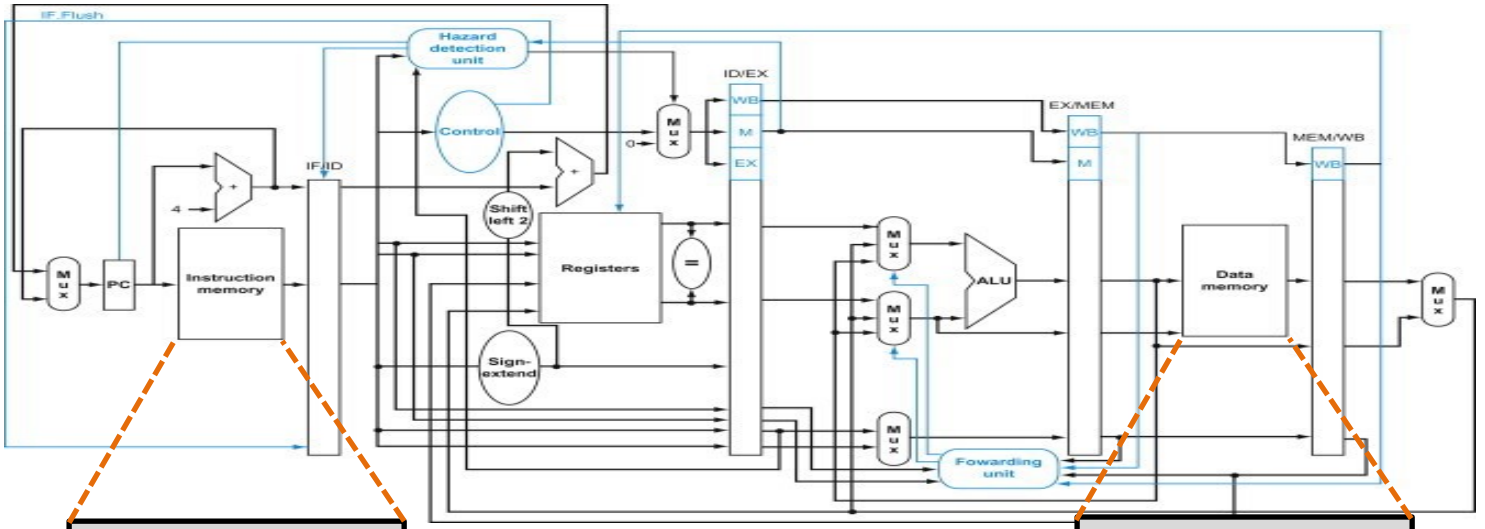
CS/ECE 552: Virtual Memory

Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mikko Lipasti, Mark Hill, David Wood, Guri Sohi, John Shen, Josh San Miguel, and Jim Smith

Memory Hierarchy



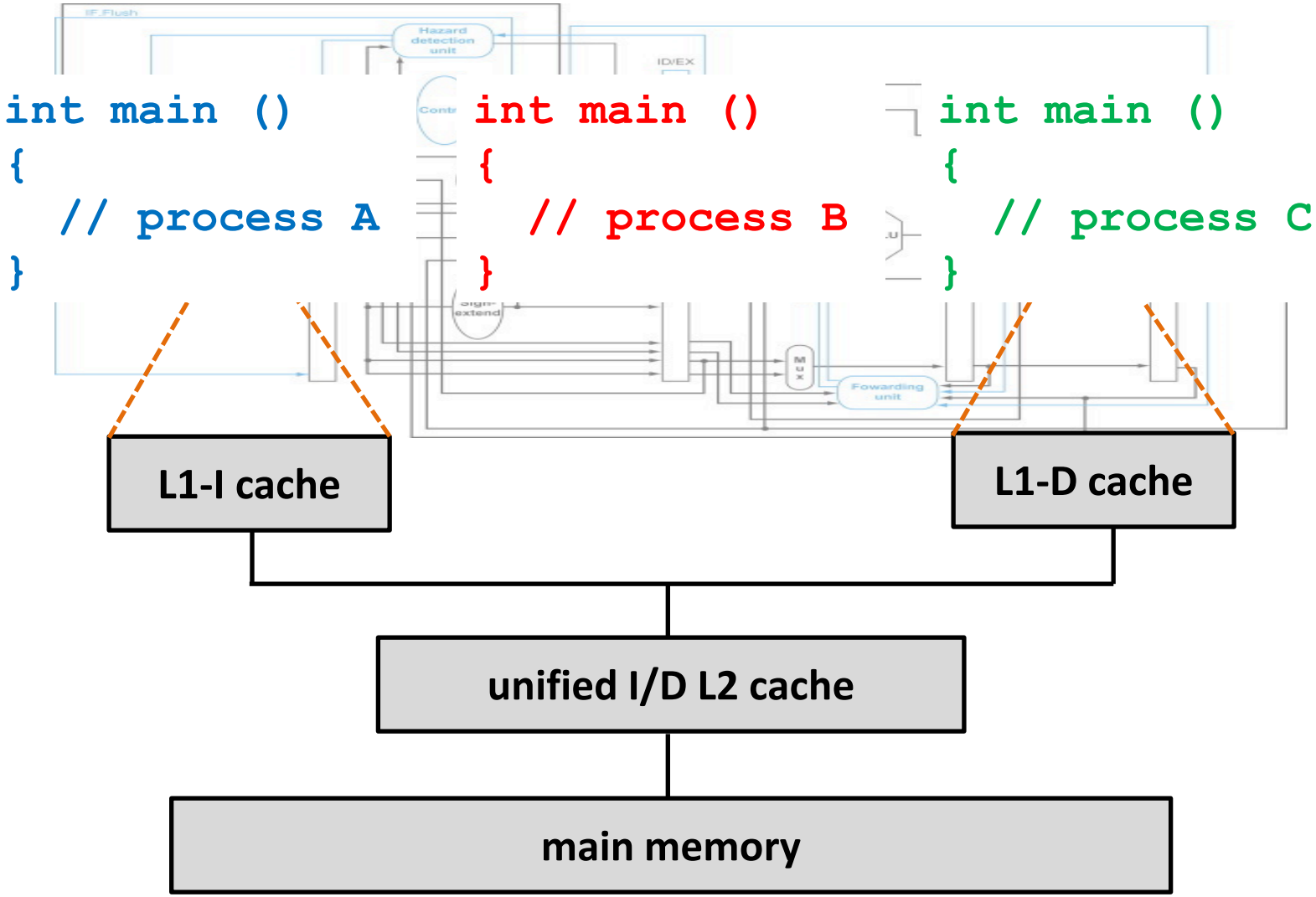
L1-I cache

L1-D cache

unified I/D L2 cache

main memory

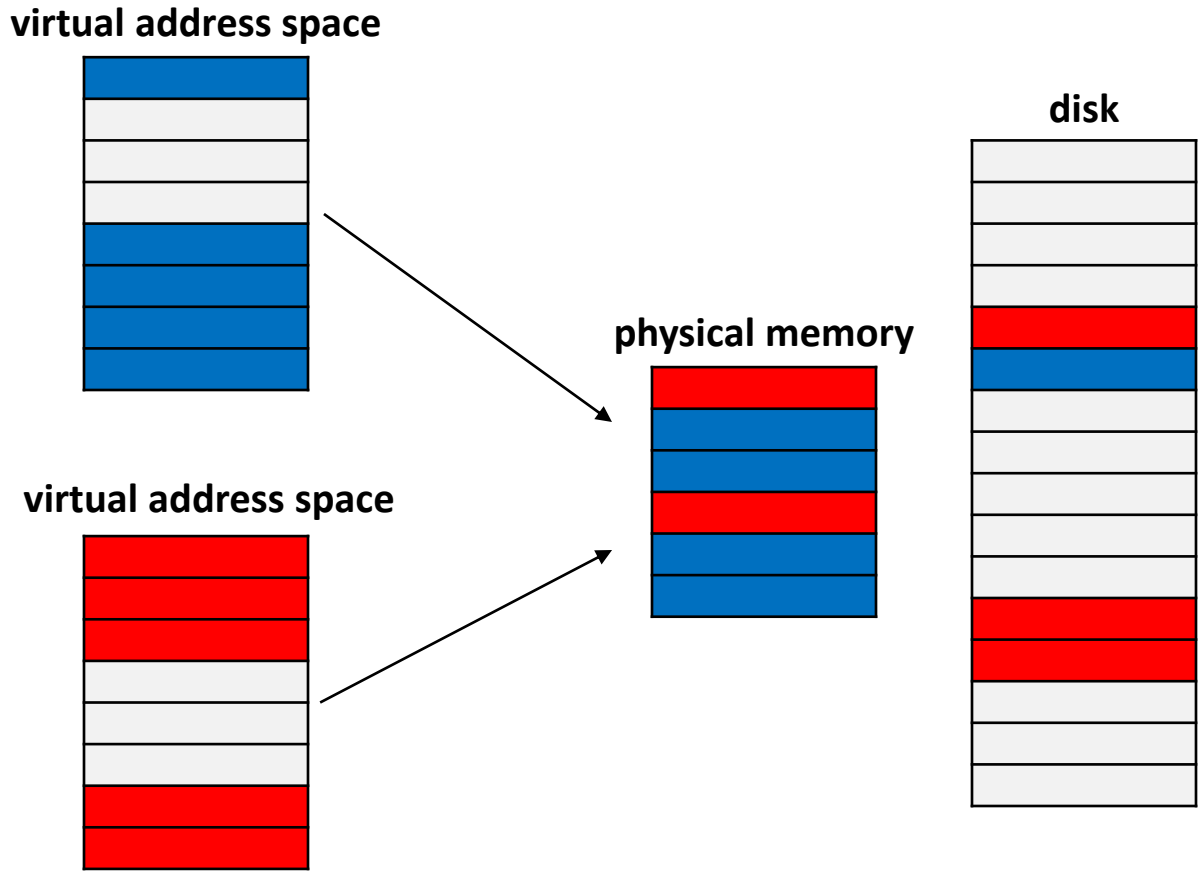
Memory Hierarchy



Virtual Memory

```
int main ()
{
   // process A
}
```

```
int main ()
{
   // process B
}
```

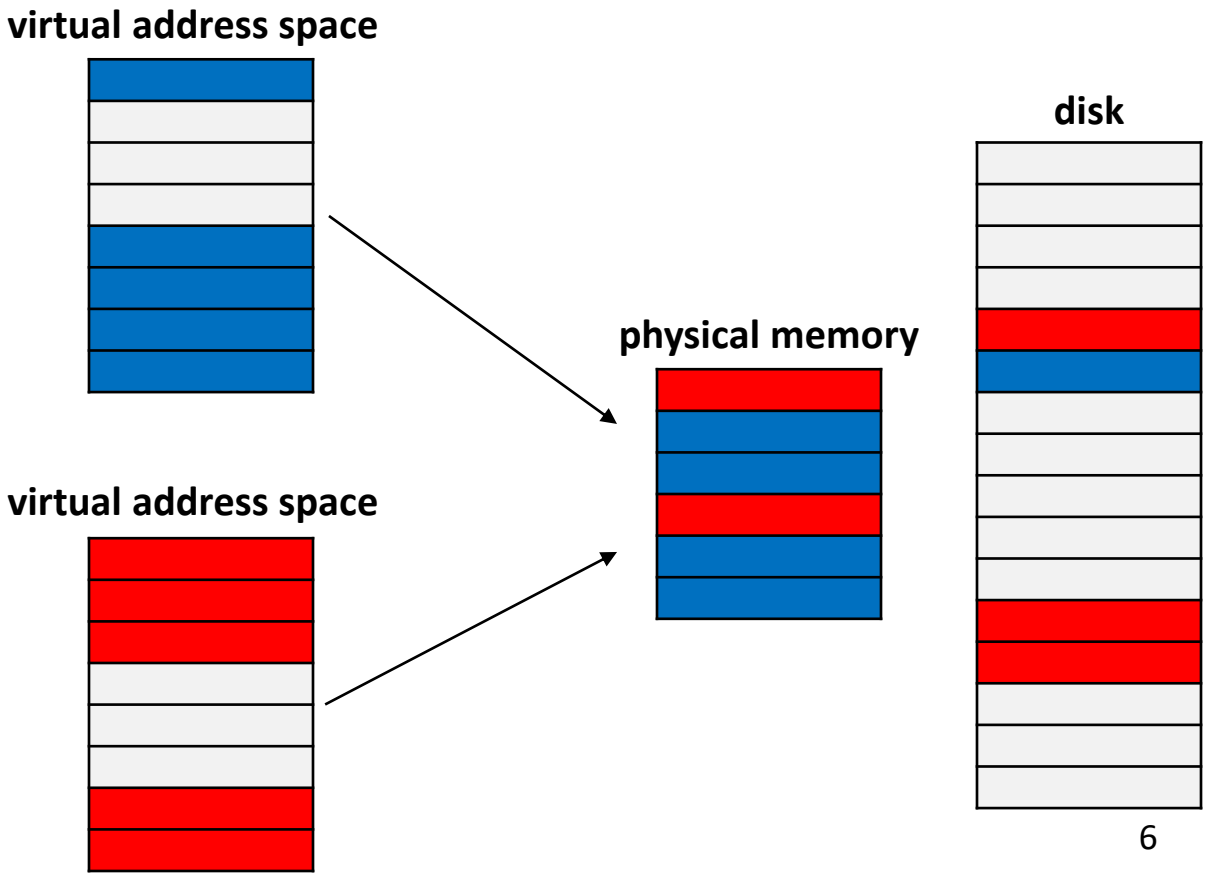


Virtual Memory – Benefits

1) Program can use more memory than the system has

```
int main ()
{
  // process A
}
```

```
int main ()
{
  // process B
}
```



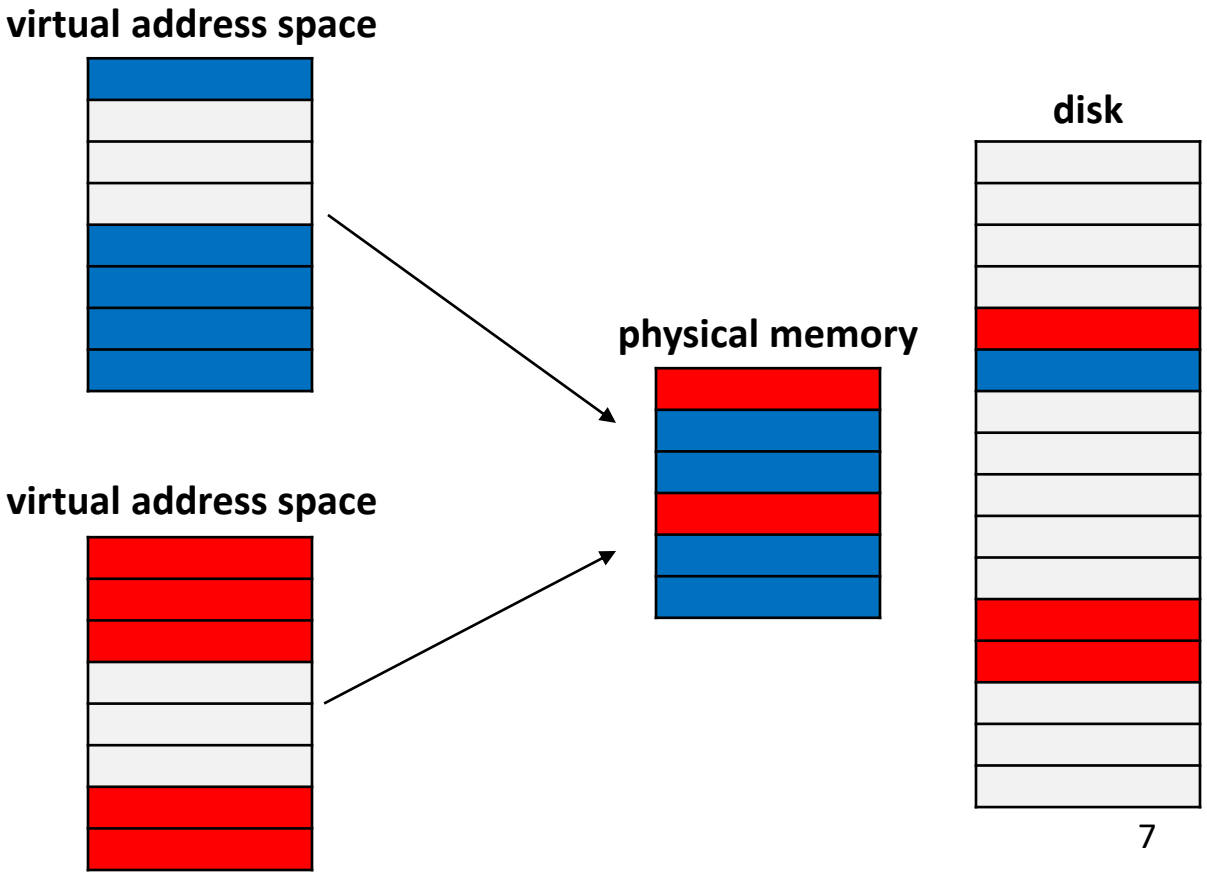
Virtual Memory – Benefits

2) Program can think that it is the only one running

- Virtual addresses are defined independent of environment

```
int main ()
{
  // process A
}
```

```
int main ()
{
  // process B
}
```



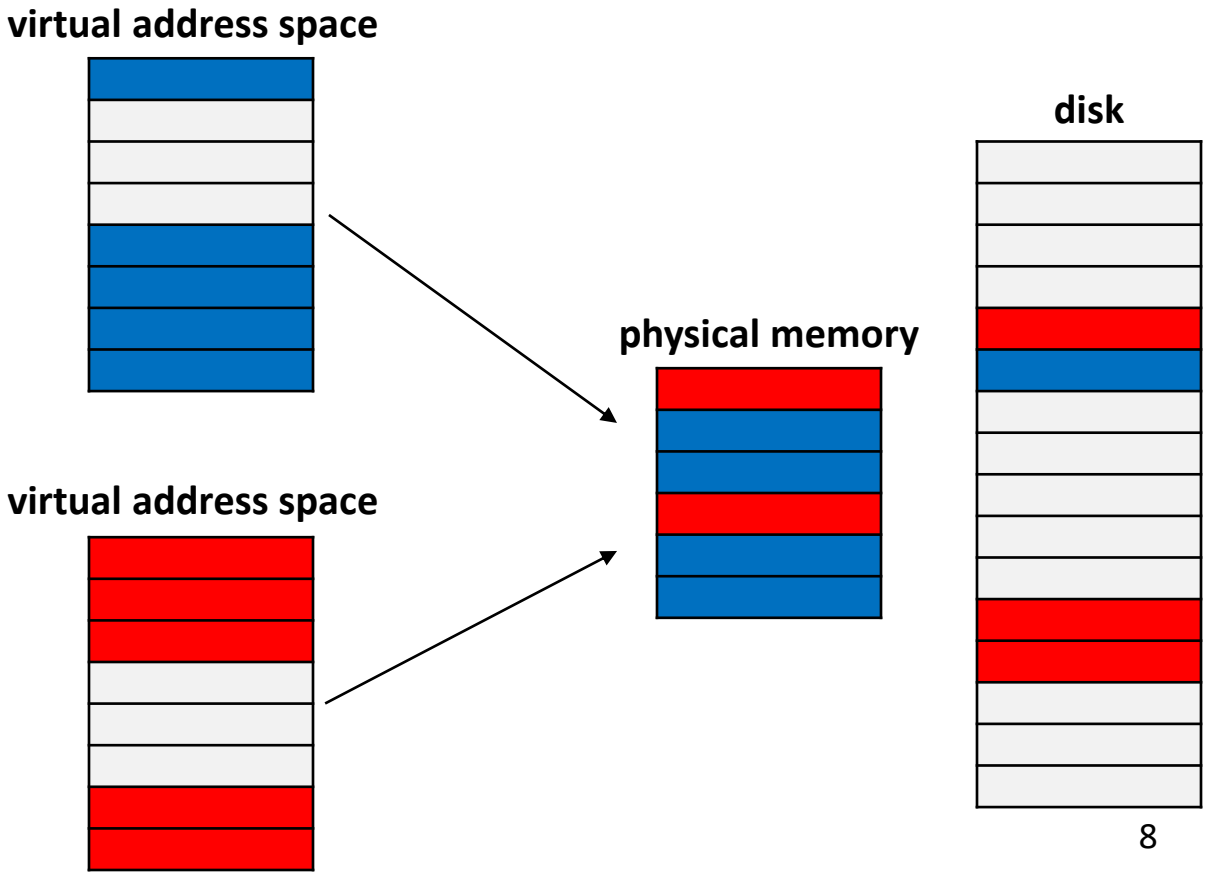
Virtual Memory – Benefits

3) Program data is protected via isolation

- Processes cannot clobber other processes' virtual memory

```
int main ()
{
  // process A
}
```

```
int main ()
{
  // process B
}
```

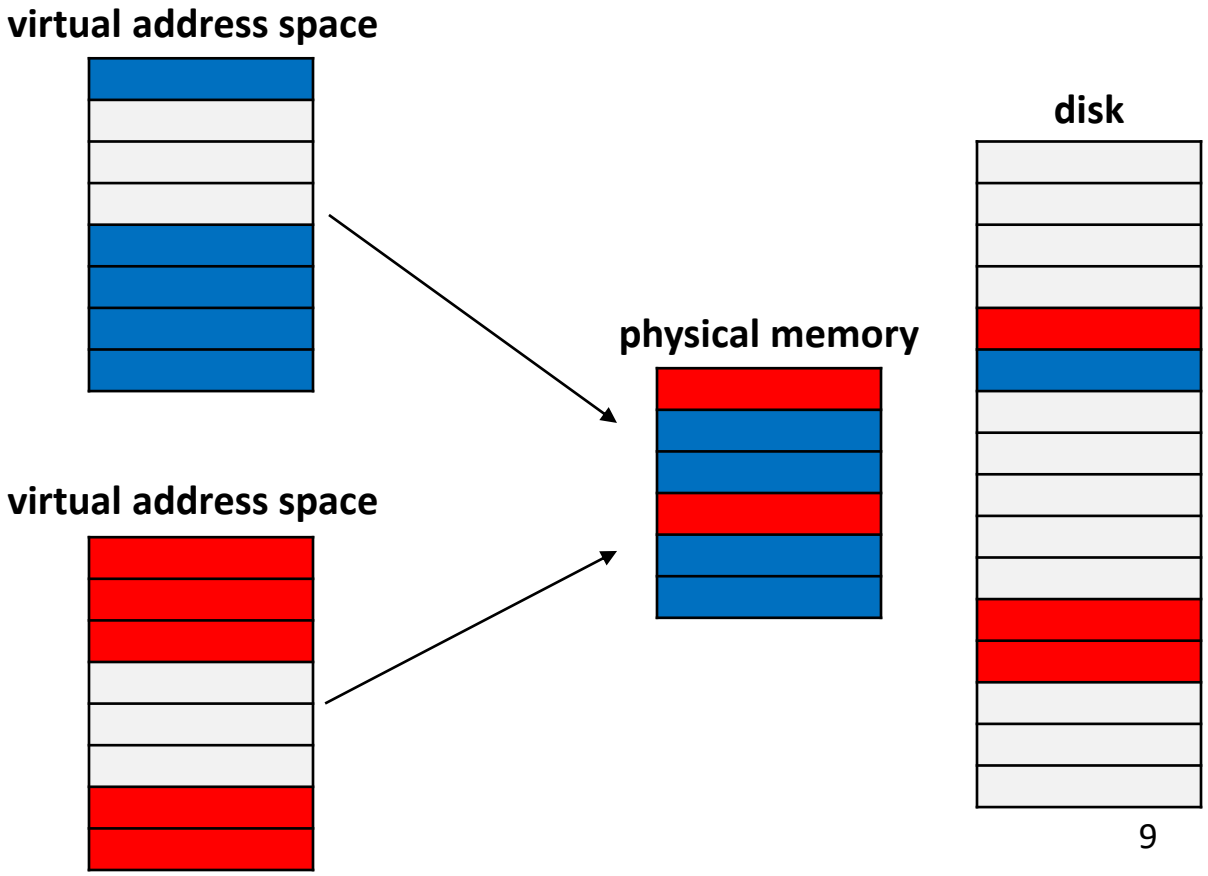


Virtual Memory – Benefits

4) Programs can start running before being fully loaded into memory

```
int main ()
{
  // process A
}
```

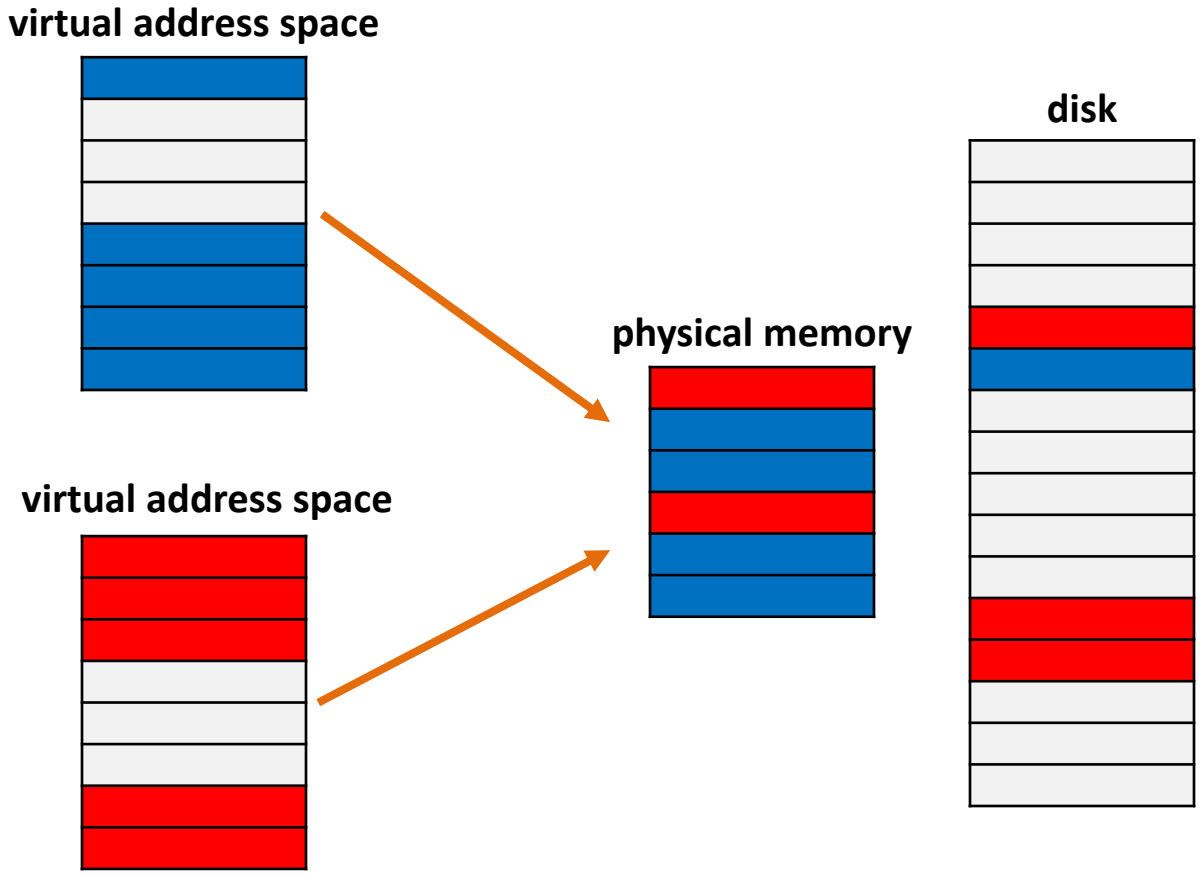
```
int main ()
{
  // process B
}
```



Virtual Memory – Translation

```
int main ()
{
   // process A
}
```

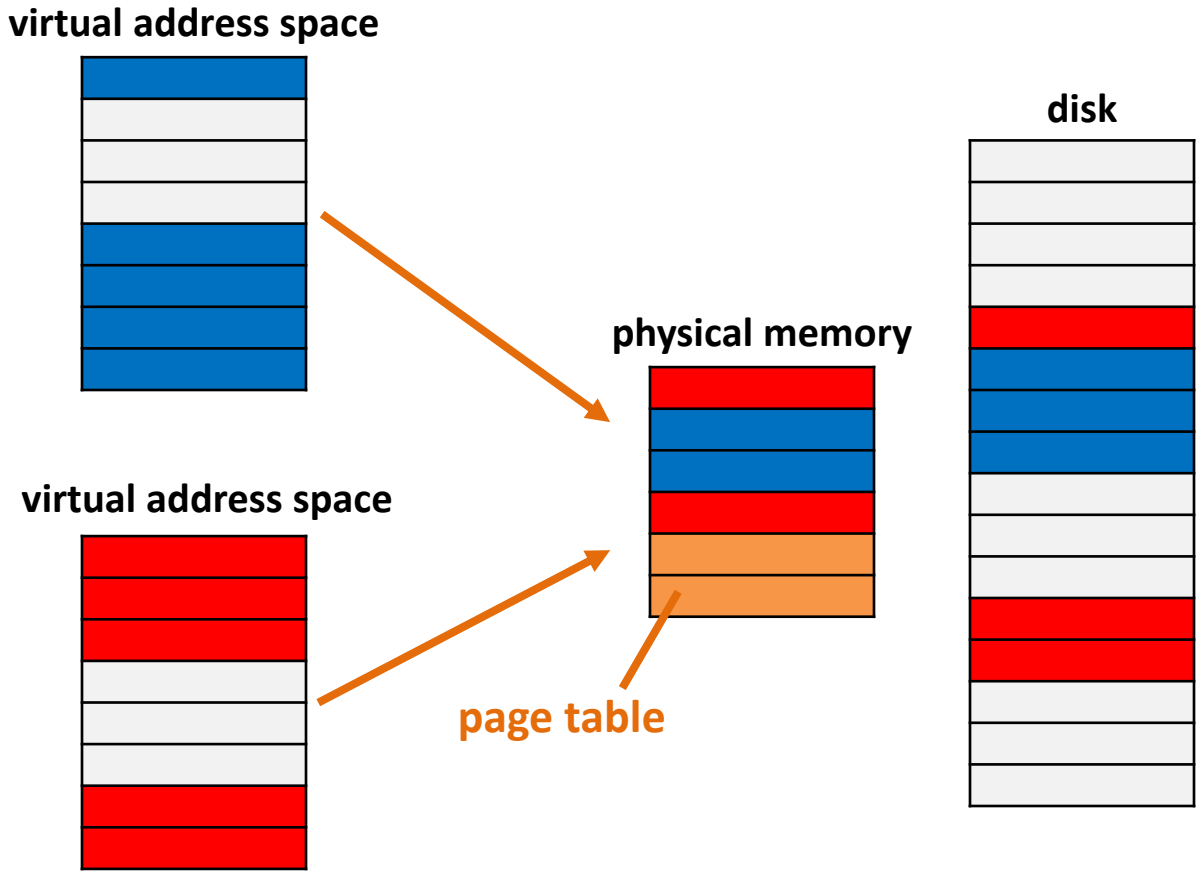
```
int main ()
{
   // process B
}
```



Virtual Memory – Translation

```
int main ()
{
   // process A
}
```

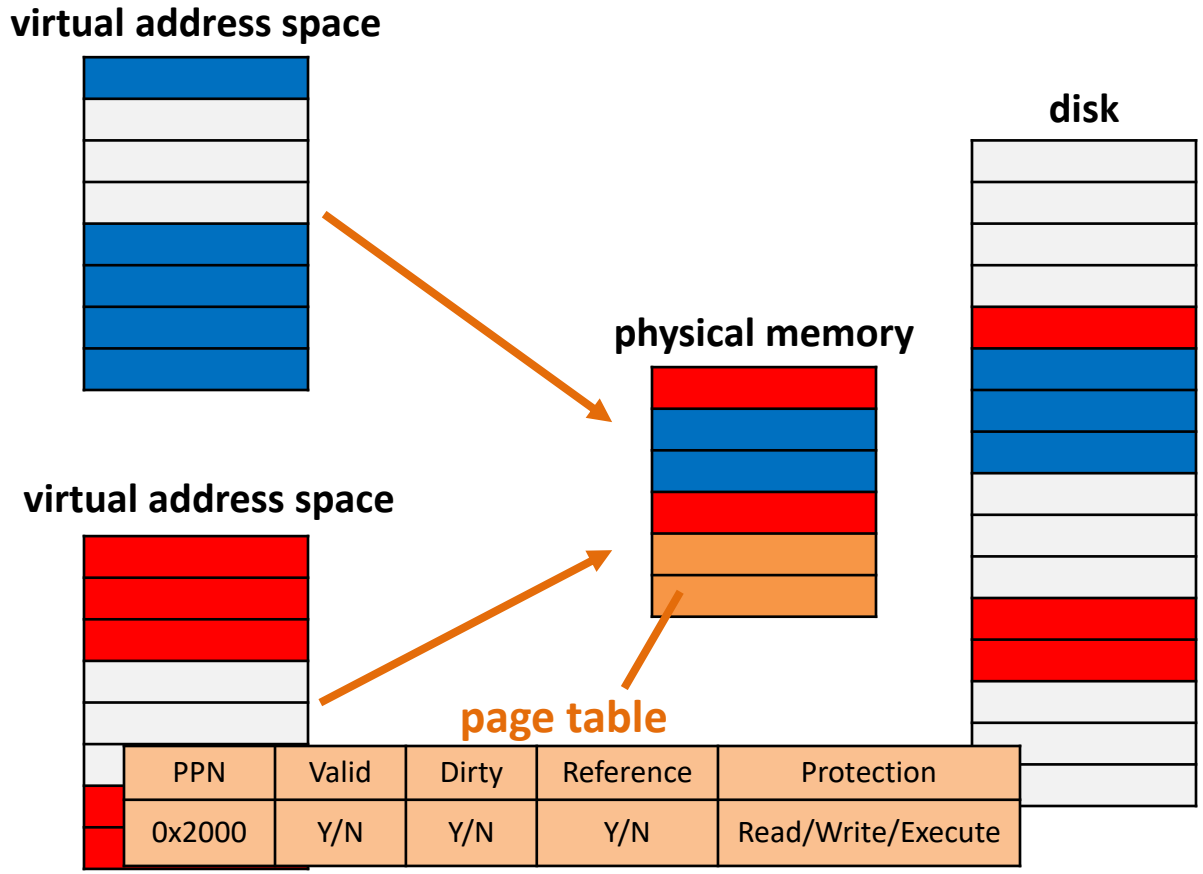
```
int main ()
{
   // process B
}
```



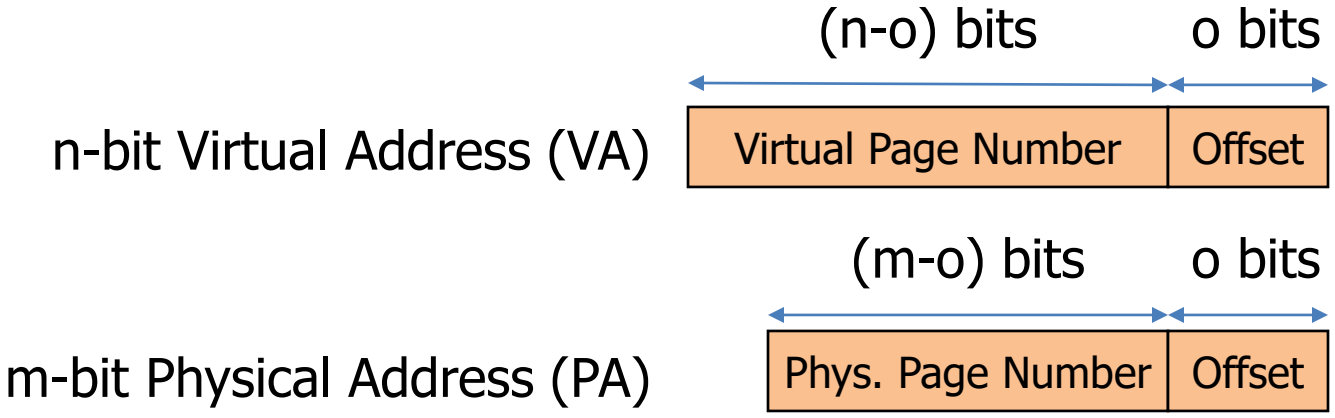
Virtual Memory – Translation

```
int main ()
{
  // process A
}
```

```
int main ()
{
  // process B
}
```



Virtual Memory – Translation



- Offset bits are the same for both VA and PA
  - Why is this ok?
- To translate VA → PA, need VPN → PPN translation
  - Translations usually stored in the page table
  - TLBs cache common page table entries (PTEs) locally
  - Page fault: TLB and page table don't have entry

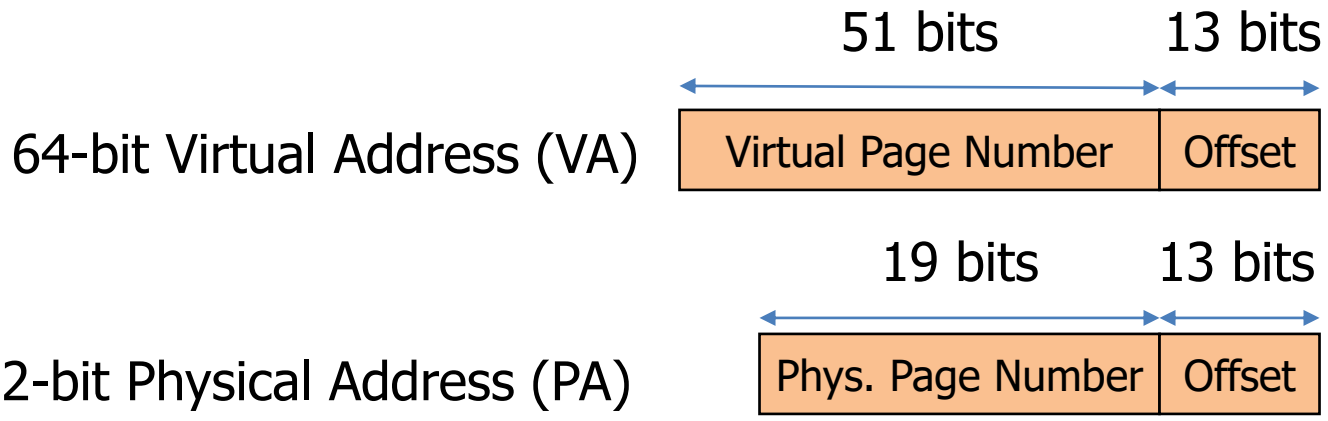
Example

- *Given 64-bit virtual addresses, 32-bit physical addresses, and 8 KB pages, byte addressable system, what are the VPN, PPN, and page offset sizes?*

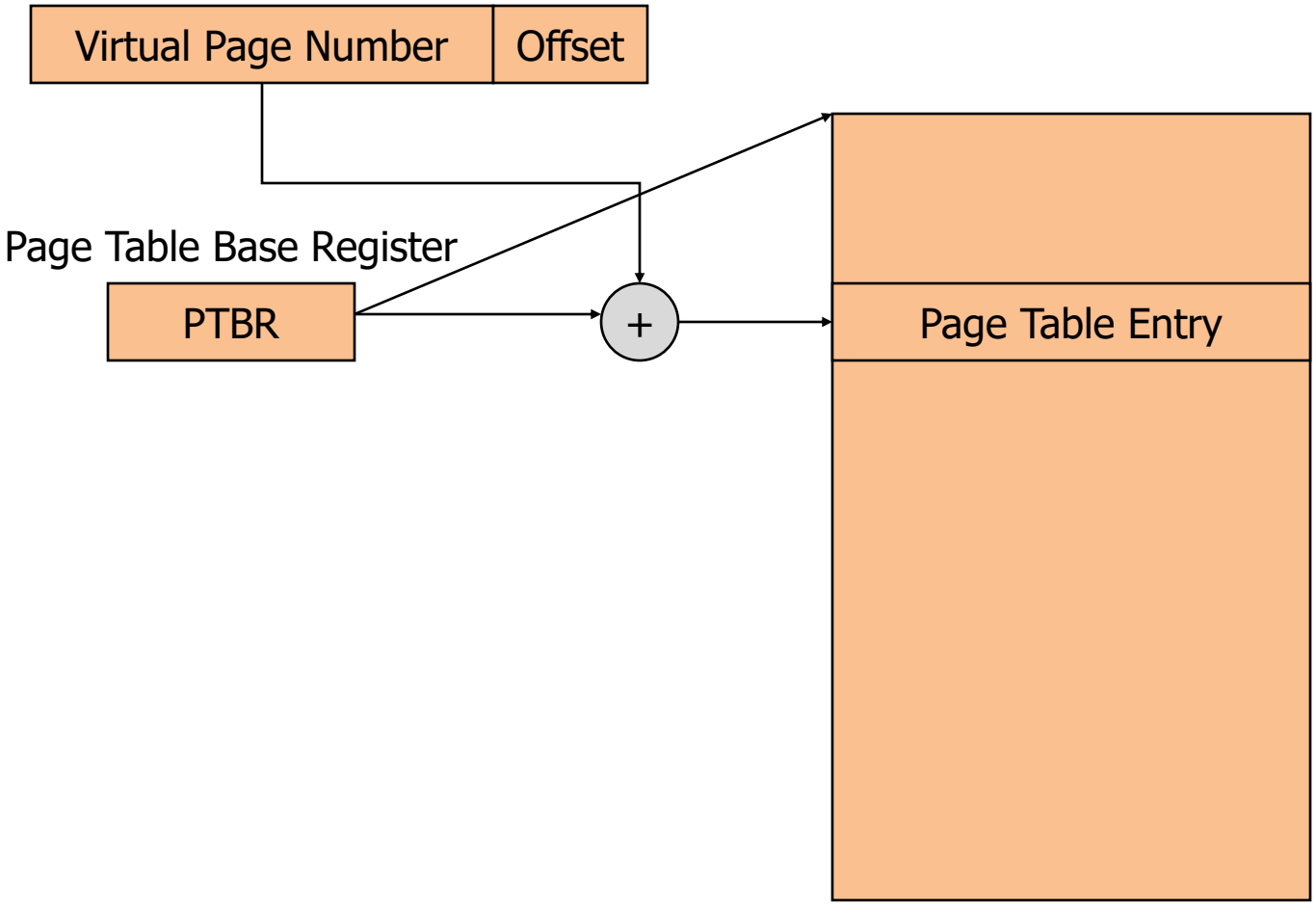
$\log_2$(8 KB) = 13 bits for page offset

64 bits – 13 bits = 51 bits for VPN

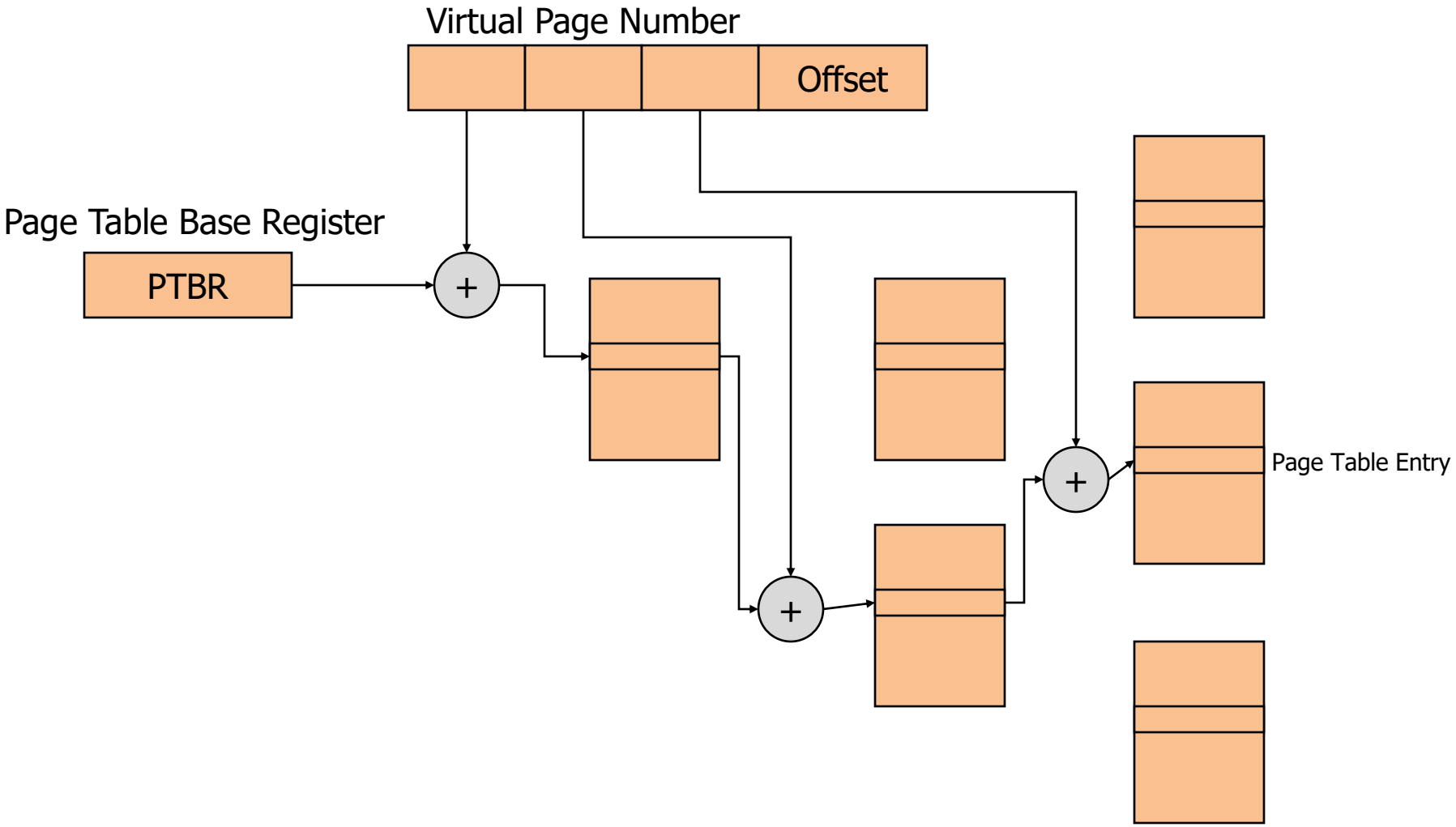
32 bits – 13 bits = 19 bits for PPN



Page Table

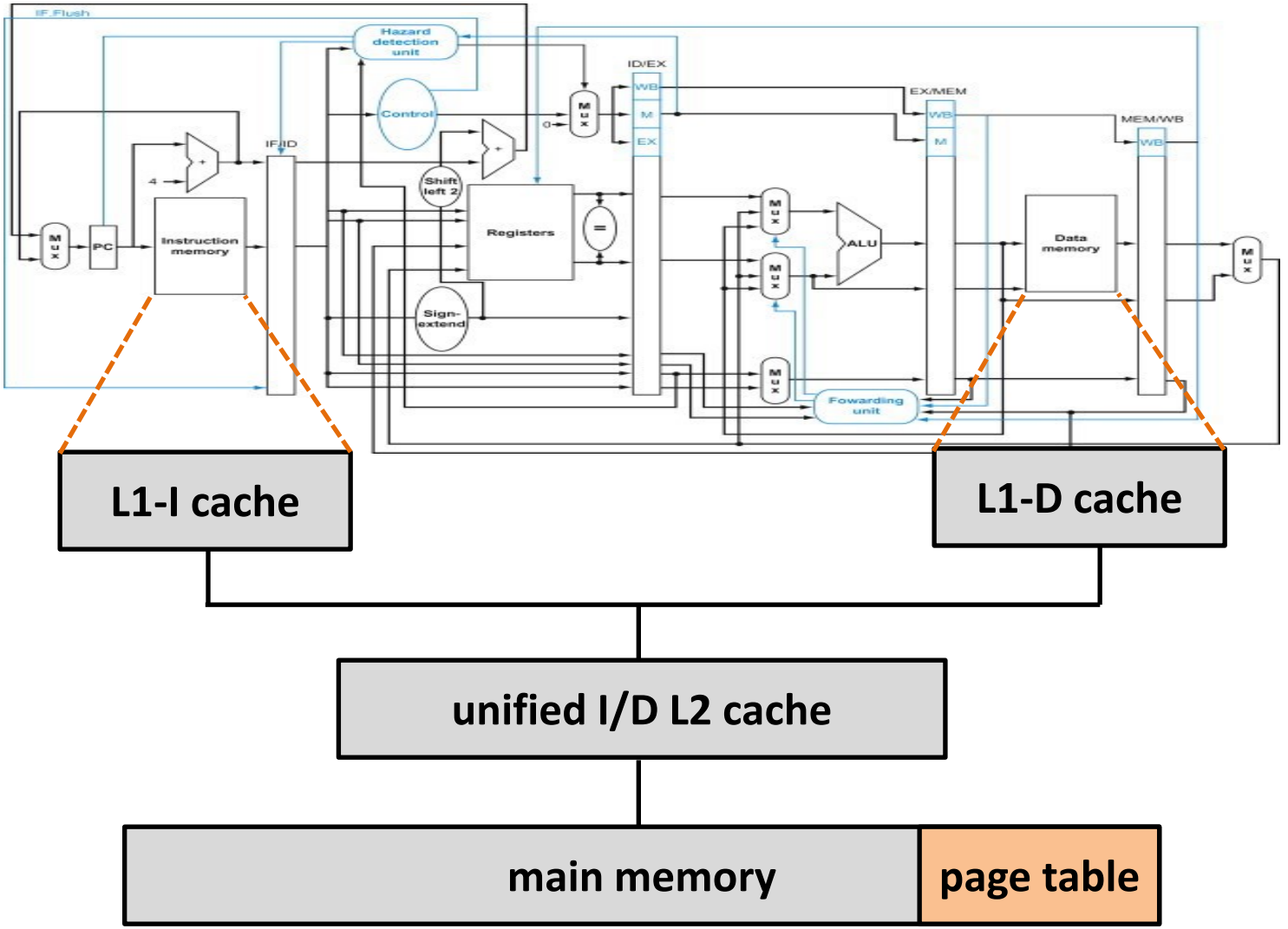

Virtual Page Number
Offset
Page Table Base Register
PTBR
+
Page Table Entry

Multi-Level Page Table

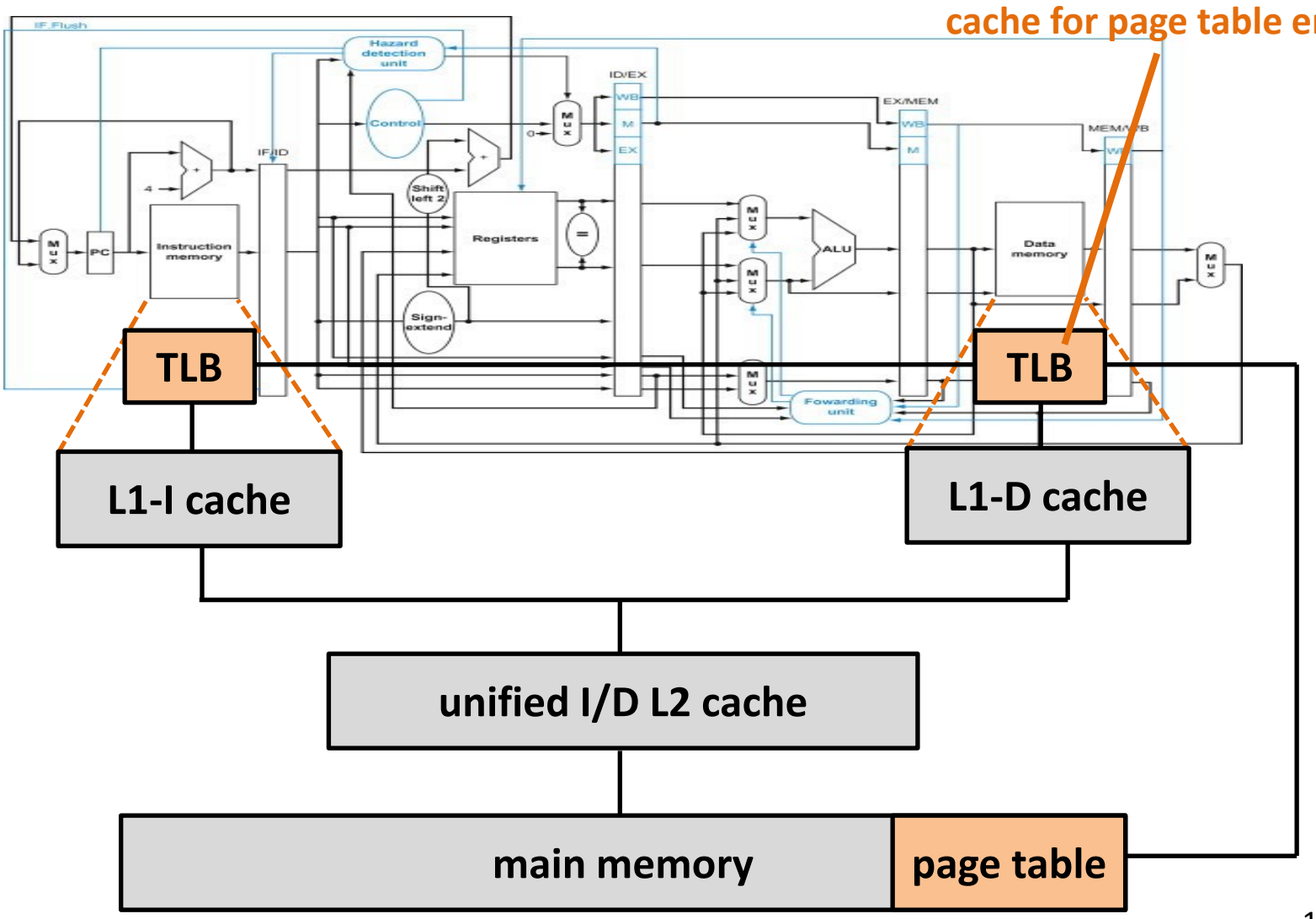


AMD and Intel CPUs usually have 4+ level page tables!

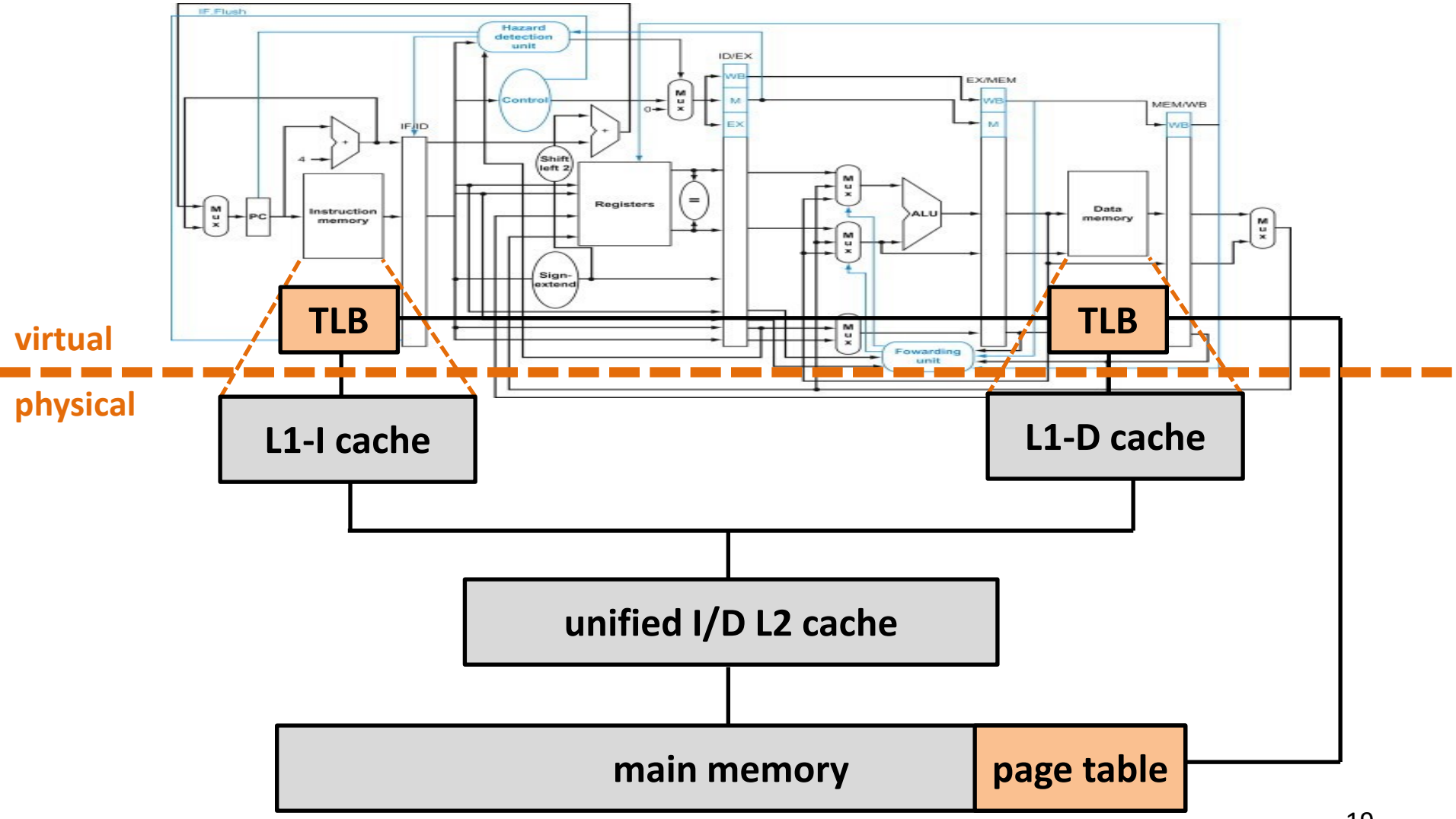
Virtual Memory – Translation



Virtual Memory – Translation



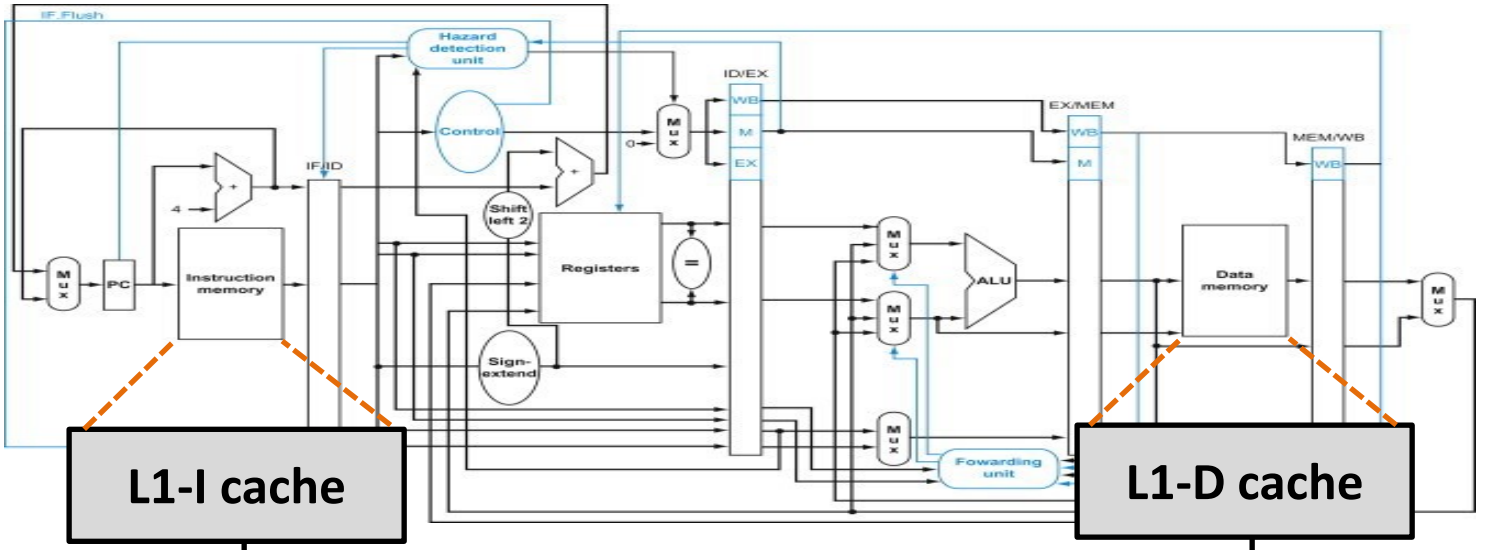
Virtual Memory – Translation



PIPT Translation Summary

- PIPT (physically indexed, physically tagged)
  - Pros:
    - All caches/memory use same address/tag
    - Logically simple
  - Cons:
    - Extra latency on **all** accesses
- Some real systems use, but not popular

Virtual Memory – Translation



L1-I cache

L1-D cache

TLB

TLB

virtual

physical

unified I/D L2 cache

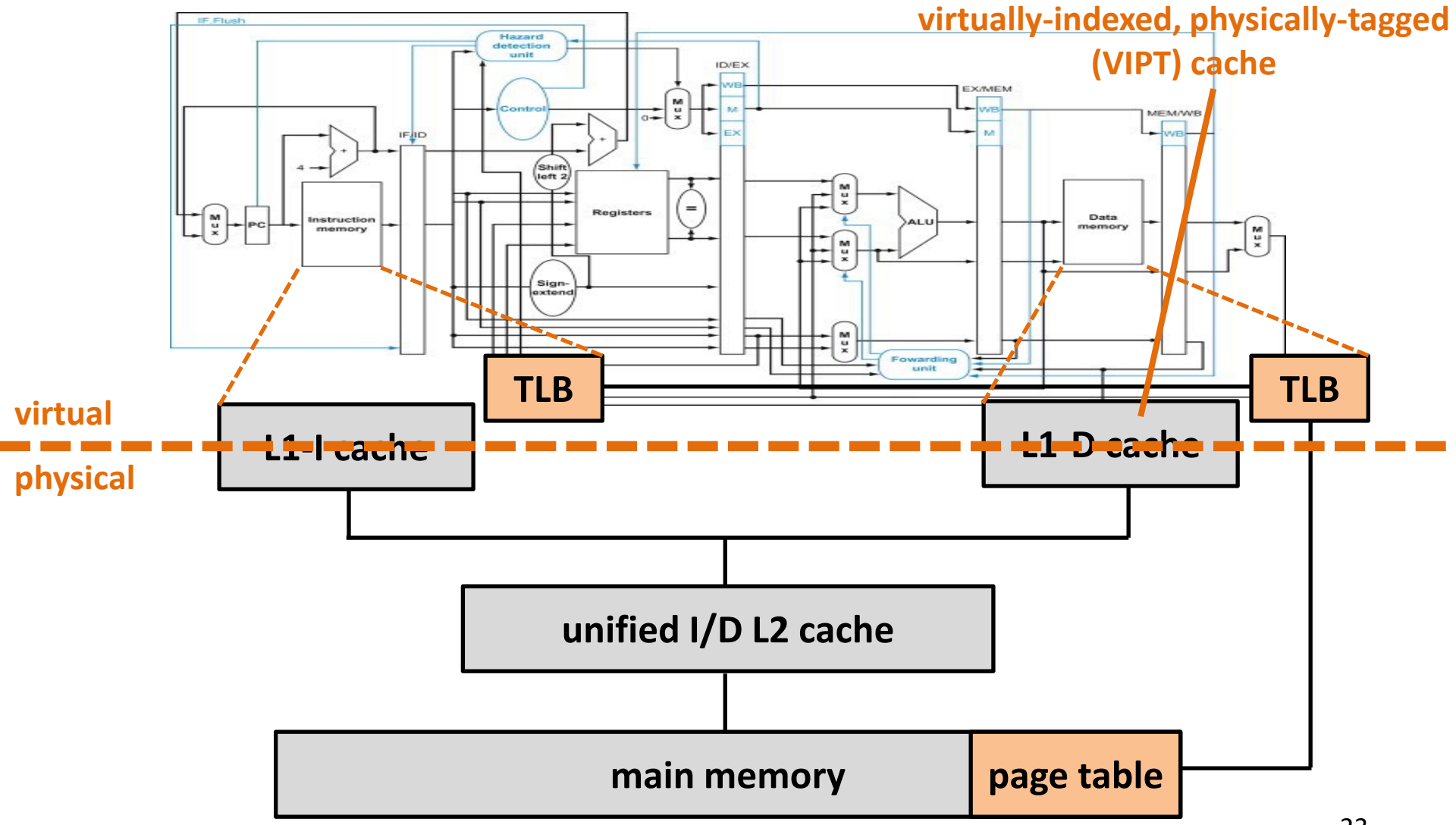
main memory

page table

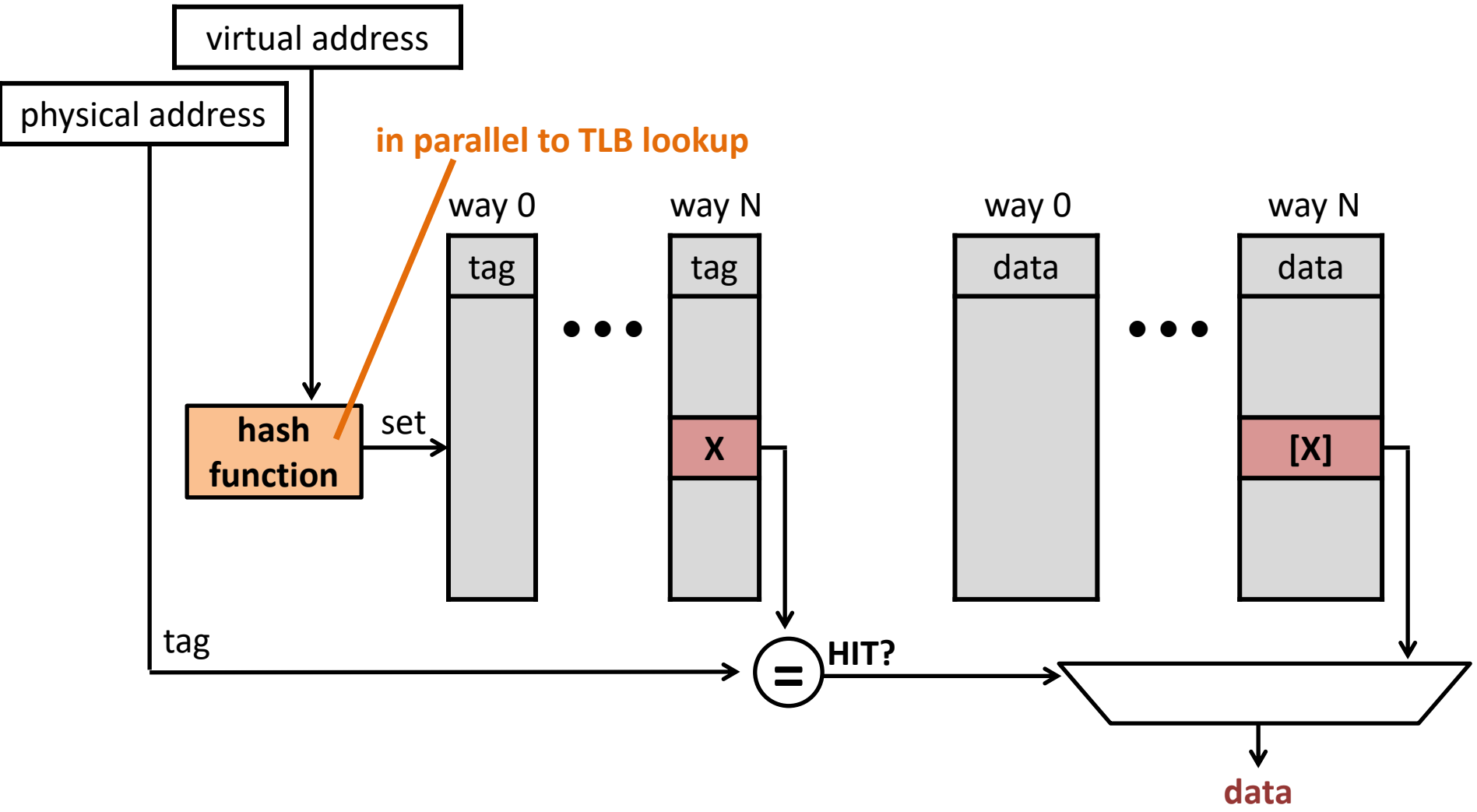
VIVT Translation Summary

- VIVT (virtually indexed, virtually tagged)
  - Pros:
    - No translation cost on L1 hits (if L1 is VIVT)
    - Common case is fast
  - Cons:
    - Cache coherence (Units 13/14) more complex
    - Extra latency on misses
- Some real systems use: e.g., NVIDIA GPUs

Virtual Memory – Translation



VIPT Cache


virtual address
physical address
in parallel to TLB lookup
way 0
way N
way 0
way N
tag
tag
data
data
hash function
set
X
[X]
tag
=
HIT?
data

VIPT Translation Summary

- VIPT (virtually indexed, physically tagged)
  - Pros:
    - Translation done in parallel – no added overhead!
    - Best of both worlds (VIVT, PIPT)
      - No extra latency on misses
      - No extra latency before all accesses
  - Cons:
    - Some limitations on way sizes …
    - … or synonyms and homonyms possible (next unit)
- Basically all modern CPUs do this

Last Option: PIVT

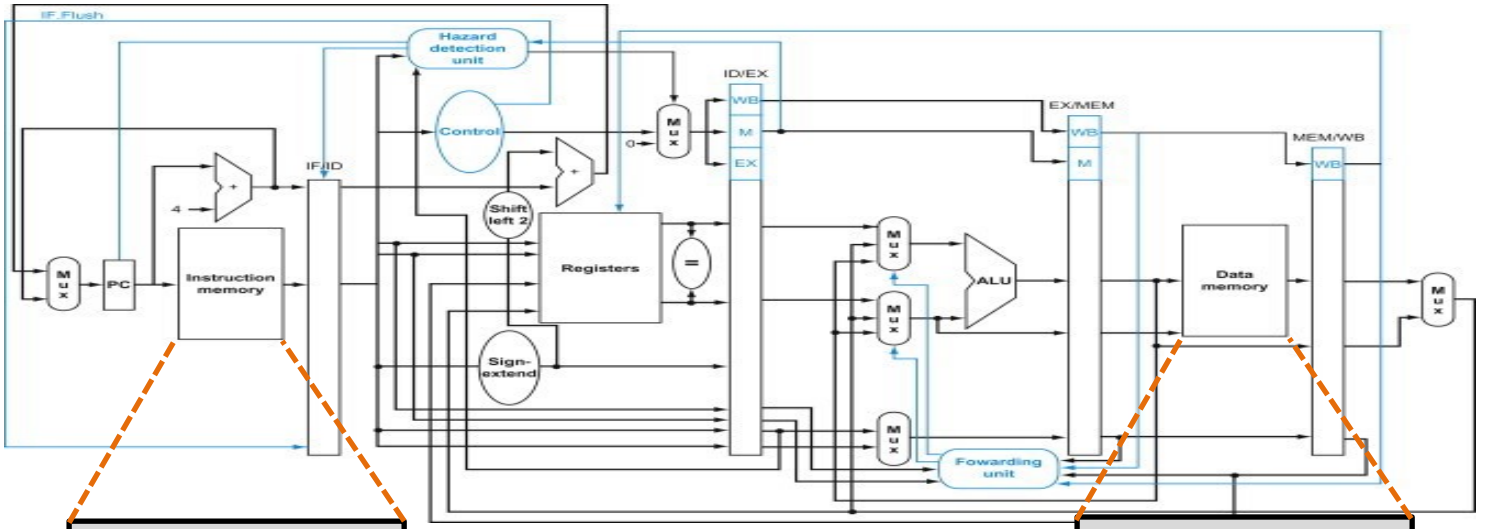
- PI = Physically indexed
- VT = Virtually tagged
- Realistically never used
  - Cache is physically indexed
  - Virtual address used for tag
  - Why?

CS/ECE 552: ECC and Virtual Memory

Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mikko Lipasti, Mark Hill, David Wood, Guri Sohi, John Shen, Joshua San Miguel, and Jim Smith

Memory Hierarchy



L1-I cache

L1-D cache

unified I/D L2 cache

main memory

Error Detection and Correction

- Main memory stores a huge number of bits
  - Probability of bit flip becomes non-trivial
  - Bit-flips (called soft errors) caused by
    - Slight manufacturing defects
    - Gamma rays and alpha particles
    - Electrical interference
    - etc.
  - Getting worse with smaller feature sizes
- Reliable systems must be protected from soft errors via ECC (error correction codes)
  - Even PCs support ECC these days

No ECC

A single bit of data is stored as one bit in memory

- **Hamming Distance:** how many bits must flip to convert a valid codeword to a different valid codeword?
- Hamming Distance = 1

Data Word (1 bit)	Codeword in Memory (1 bit)
0	0
1	1

Single Error Detection (SED)

Store one extra bit of redundancy in memory

- Hamming Distance = 2

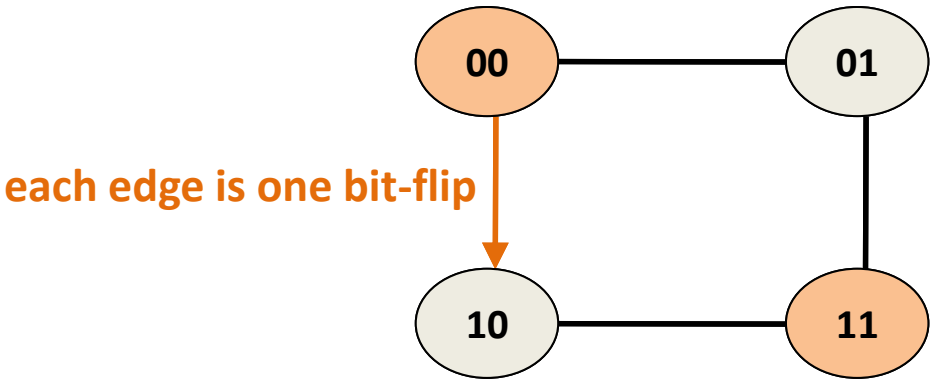
Data Word (1 bit)	Codeword in Memory (2 bits)
0	00
1	11

Single Error Detection (SED)

Store one extra bit of redundancy in memory

- Hamming Distance = 2

Data Word (1 bit)	Codeword in Memory (2 bits)
0	00
1	11

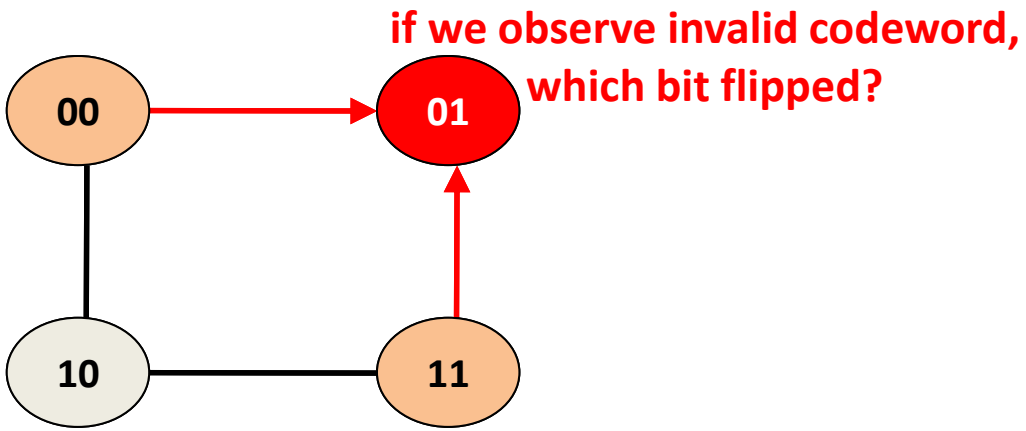


Single Error Detection (SED)

Store one extra bit of redundancy in memory

- Hamming Distance = 2

Data Word (1 bit)	Codeword in Memory (2 bits)
0	00
1	11



Single Error Correction (SECSED)

Store two extra bits of redundancy in memory

- Hamming Distance = 3

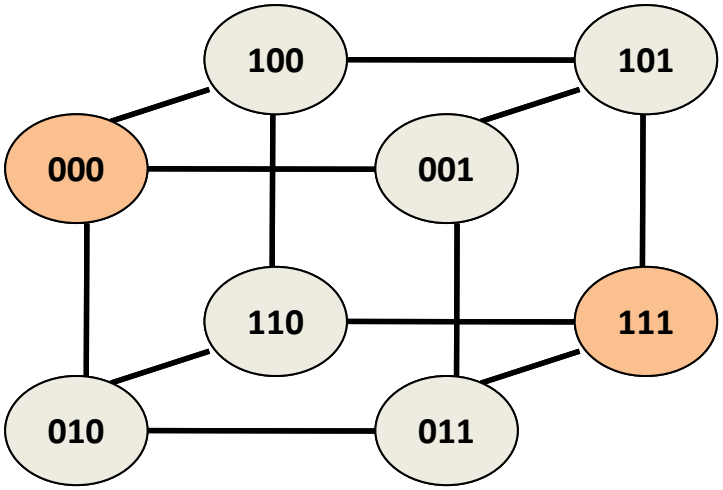
Data Word (1 bit)	Codeword in Memory (3 bits)
0	000
1	111

Single Error Correction (SECSED)

Store two extra bits of redundancy in memory

- Hamming Distance = 3

Data Word (1 bit)	Codeword in Memory (3 bits)
0	000
1	111

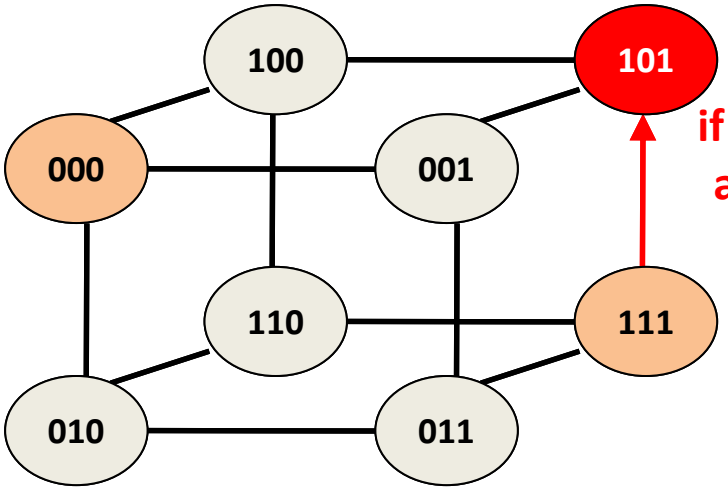


Single Error Correction (SECSED)

Store two extra bits of redundancy in memory

- Hamming Distance = 3

Data Word (1 bit)	Codeword in Memory (3 bits)
0	000
1	111



if we observe invalid codeword, assuming only one bit flipped, we know exactly which valid codeword we had previously

SED – Parity

For arbitrary n-bit data word ($b_n \ldots b_1$), add one parity bit P to codeword

- $P = b_1 \oplus \ldots \oplus b_n$
- When reading codeword, check *syndrome* = $P \oplus b_1 \oplus \ldots \oplus b_n == 0$?
- Note: following discussions assume one-based indexing

SED – Parity

For arbitrary n-bit data word ($b_n \ldots b_1$), add one parity bit P to codeword

- $P = b_1 \oplus \ldots \oplus b_n$
- When reading codeword, check *syndrome* = $P \oplus b_1 \oplus \ldots \oplus b_n == 0$?
- Note: following discussions assume one-based indexing

Data Word (4 bits)	**Codeword in Memory (**4 bits + **1 parity bit)**
0000	0000**0**
0001	0001**1**
0010	0010**1**
0011	0011**0**
0100	0100**1**
0101	0101**0**

SECSED – Hamming Code

For arbitrary n-bit data word ($b_n \ldots b_1$), add k check bits ($C_k \ldots C_1$) to codeword, where $2^k \geq m+k+1$

- e.g., Hamming(7,4): n=4, k=3

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	**Bit flipped?**
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	Bit flipped?
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

Syndrome	
S_1	
S_2	
S_3	

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	Bit flipped?
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

Syndrome	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	
S_3	

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	Bit flipped?
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

Syndrome	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	Bit flipped?
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

Syndrome	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	Bit flipped?
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

Syndrome	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$

Codeword	
W_1	
W_2	
W_3	
W_4	
W_5	
W_6	
W_7	

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	Bit flipped?
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

Syndrome	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$

use these as check bits (i.e., parity)

Codeword	
W_1	$C_1 = W_3 \oplus W_5 \oplus W_7$
W_2	$C_2 = W_3 \oplus W_6 \oplus W_7$
W_3	
W_4	$C_3 = W_5 \oplus W_6 \oplus W_7$
W_5	
W_6	
W_7	

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	Bit flipped?
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

e.g., S_1 is 1 if any one of W_1, W_3, W_5 or W_7 flipped

Syndrome	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$

use these as check bits (i.e., parity)

Codeword	
W_1	$C_1 = W_3 \oplus W_5 \oplus W_7$
W_2	$C_2 = W_3 \oplus W_6 \oplus W_7$
W_3	
W_4	$C_3 = W_5 \oplus W_6 \oplus W_7$
W_5	
W_6	
W_7	

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	Bit flipped?
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

Syndrome	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$

Codeword	
W_1	$C_1 = W_3 \oplus W_5 \oplus W_7$
W_2	$C_2 = W_3 \oplus W_6 \oplus W_7$
W_3	b_1
W_4	$C_3 = W_5 \oplus W_6 \oplus W_7$
W_5	b_2
W_6	b_3
W_7	b_4

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	Bit flipped?
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

Syndrome	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4

SECSED – Hamming(7,4)

Store a 4-bit data word ($b_4b_3b_2b_1$) as a 7-bit codeword ($W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 3-bit *syndrome* ($S_3S_2S_1$) that specifies if any (and which) bit has flipped:

Syndrome	Bit flipped?
000	none
001	W_1
010	W_2
011	W_3
100	W_4
101	W_5
110	W_6
111	W_7

Syndrome	
S_1	$W_1 \oplus W_3 \oplus \mathbf{W_5} \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus \mathbf{W_5} \oplus W_6 \oplus W_7$

e.g., if both S_1 and S_3 are 1, then W_5 must have flipped

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4

SECDED – Hamming(7,4) + Parity

Store a 4-bit data word ($b_4b_3b_2b_1$) as an 8-bit codeword ($W_8W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 4-bit *syndrome* that includes a 1-bit *parity* ($S_3S_2S_1$, S_P):

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

SECDED – Hamming(7,4) + Parity

Store a 4-bit data word ($b_4b_3b_2b_1$) as an 8-bit codeword ($W_8W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 4-bit *syndrome* that includes a 1-bit *parity* ($S_3S_2S_1$, S_P):

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$
S_P	

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	

SECDED – Hamming(7,4) + Parity

Store a 4-bit data word ($b_4b_3b_2b_1$) as an 8-bit codeword ($W_8W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 4-bit *syndrome* that includes a 1-bit *parity* ($S_3S_2S_1$, S_P):

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$
S_P	$W_8 \oplus \ldots \oplus W_1$

use this as parity bit of entire codeword

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

Store a 4-bit data word ($b_4b_3b_2b_1$) as an 8-bit codeword ($W_8W_7W_6W_5W_4W_3W_2W_1$) in memory. Whenever we read the codeword, generate a 4-bit *syndrome* that includes a 1-bit *parity* ($S_3S_2S_1$, S_P):

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$
S_P	$W_8 \oplus \ldots \oplus W_1$

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED Example (from Quiz)

- 8-bit SECDED codeword:
  - 1101 0110 ($C_1C_2b_1C_3b_2b_3b_4P$)
  - Single bit error has occurred
  - Which bit did the single-bit error occur in?
  - What is the original data ($b_4b_3b_2b_1$)?

SECDED Example (from Quiz) Cont.

1. Calculate syndrome values
   - $S_1 = W_1 \oplus W_3 \oplus W_5 \oplus W_7 = C_1 \oplus b_1 \oplus b_2 \oplus b_4$
     $S_1 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$
   - $S_2 = W_2 \oplus W_3 \oplus W_6 \oplus W_7 = C_2 \oplus b_1 \oplus b_3 \oplus b_4$
     $S_2 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$
   - $S_3 = W_4 \oplus W_5 \oplus W_6 \oplus W_7 = C_3 \oplus b_2 \oplus b_3 \oplus b_4$
     $S_3 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$
   - $S_P = W_1 \oplus W_2 \oplus W_3 \oplus W_4 \oplus W_5 \oplus W_6 \oplus W_7 \oplus W_8$
     $S_P = C_1 \oplus C_2 \oplus b_1 \oplus C_3 \oplus b_2 \oplus b_3 \oplus b_4 \oplus P$
     $S_P = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

SECDED Example (from Quiz) Cont.

2. Look up syndrome in table:
   - $S_3S_2S_1,S_P$ = 110,1

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

   - Thus W_6 must be the culprit → bit b_3
   - Original data: b4b3b2b1 = 1000

SECDED Example: Alt. Approach

1. Calculate check values
   - $C_1' = b_1 \oplus b_2 \oplus b_4 = 0 \oplus 0 \oplus 1 = 1$
   - $C_2' = b_1 \oplus b_3 \oplus b_4 = 0 \oplus 1 \oplus 1 = 0$
     - Sidenote: C_2' doesn't match codeword C_2
   - $C_3' = b_2 \oplus b_3 \oplus b_4 = 0 \oplus 1 \oplus 1 = 0$
     - Sidenote: C_3' doesn't match codeword C_3
   - $P' = W_1 \oplus W_2 \oplus W_3 \oplus W_4 \oplus W_5 \oplus W_6 \oplus W_7$

     $P' = C_1 \oplus C_2 \oplus b_1 \oplus C_3 \oplus b_2 \oplus b_3 \oplus b_4$

     $P' = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 1$
     - Sidenote: P' doesn't match codeword P

SECDED Example: Alt. Approach

2. Calculate syndromes (from handout):
   - $S_1 = e_1 = C_1' \oplus C_1 = 1 \oplus 1 = 0$
   - $S_2 = e_2 = C_2' \oplus C_2 = 0 \oplus 1 = 1$
   - $S_3 = e_3 = C_3' \oplus C_3 = 0 \oplus 1 = 1$
   - Same syndrome as other approach

SECDED – Hamming(7,4) + Parity

e.g., Encode $b_1b_2b_3b_4$ = 1010 into an 8-bit codeword:

$W_1W_2W_3W_4W_5W_6W_7W_8$ = _____

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$
S_P	$W_8 \oplus \ldots \oplus W_1$

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

e.g., Encode $b_1b_2b_3b_4$ = 1010 into an 8-bit codeword:

$W_1W_2W_3W_4W_5W_6W_7W_8$ = 10110100

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$
S_P	$W_8 \oplus \ldots \oplus W_1$

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

e.g., A single bit-flip has occurred. Recover original data:

$W_1W_2W_3W_4W_5W_6W_7W_8$ = **10010100**

$b_1b_2b_3b_4$ = _____

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	$W_1 \oplus W_3 \oplus W_5 \oplus W_7$
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$
S_P	$W_8 \oplus \ldots \oplus W_1$

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

e.g., A single bit-flip has occurred. Recover original data:

$W_1W_2W_3W_4W_5W_6W_7W_8$ = **10010100**

$b_1b_2b_3b_4$ = _____

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	1
S_2	$W_2 \oplus W_3 \oplus W_6 \oplus W_7$
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$
S_P	$W_8 \oplus \ldots \oplus W_1$

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

e.g., A single bit-flip has occurred. Recover original data:

$W_1W_2W_3W_4W_5W_6W_7W_8$ = **10010100**

$b_1b_2b_3b_4$ = _____

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	1
S_2	1
S_3	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$
S_P	$W_8 \oplus \ldots \oplus W_1$

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

e.g., A single bit-flip has occurred. Recover original data:

$W_1W_2W_3W_4W_5W_6W_7W_8$ = **10010100**

$b_1b_2b_3b_4$ = _____

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	1
S_2	1
S_3	0
S_P	$W_8 \oplus \ldots \oplus W_1$

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

e.g., A single bit-flip has occurred. Recover original data:

$W_1W_2W_3W_4W_5W_6W_7W_8$ = **10010100**

$b_1b_2b_3b_4$ = _____

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	1
S_2	1
S_3	0
S_P	1

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

e.g., A single bit-flip has occurred. Recover original data:

$W_1W_2W_3W_4W_5W_6W_7W_8$ = **10010100**

$b_1b_2b_3b_4$ = _____

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	1
S_2	1
S_3	0
S_P	1

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

e.g., A single bit-flip has occurred. Recover original data:

$W_1W_2W_3W_4W_5W_6W_7W_8$ = **10010100**

$b_1b_2b_3b_4$ = _____

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	1
S_2	1
S_3	0
S_P	1

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

e.g., A single bit-flip has occurred. Recover original data:

$W_1W_2W_3W_4W_5W_6W_7W_8$ = **10010100**

$b_1b_2b_3b_4$ = 1010

$S_3S_2S_1$, S_P	Bit flipped?
000, 0	none
001, 1	W_1
010, 1	W_2
011, 1	W_3
100, 1	W_4
101, 1	W_5
110, 1	W_6
111, 1	W_7
000, 1	W_8
non-zero, 0	double bit-flip

Syndrome, Parity	
S_1	1
S_2	1
S_3	0
S_P	1

Codeword	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

SECDED – Hamming(7,4) + Parity

Compare this to simply storing the bits redundantly:

Hamming(7,4) + Parity (SECDED Codeword)	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

vs.

1-Bit Redundancy (_____ Codeword)	
W_1	b_1
W_2	b_2
W_3	b_3
W_4	b_4
W_5	b_1
W_6	b_2
W_7	b_3
W_8	b_4

SECDED – Hamming(7,4) + Parity

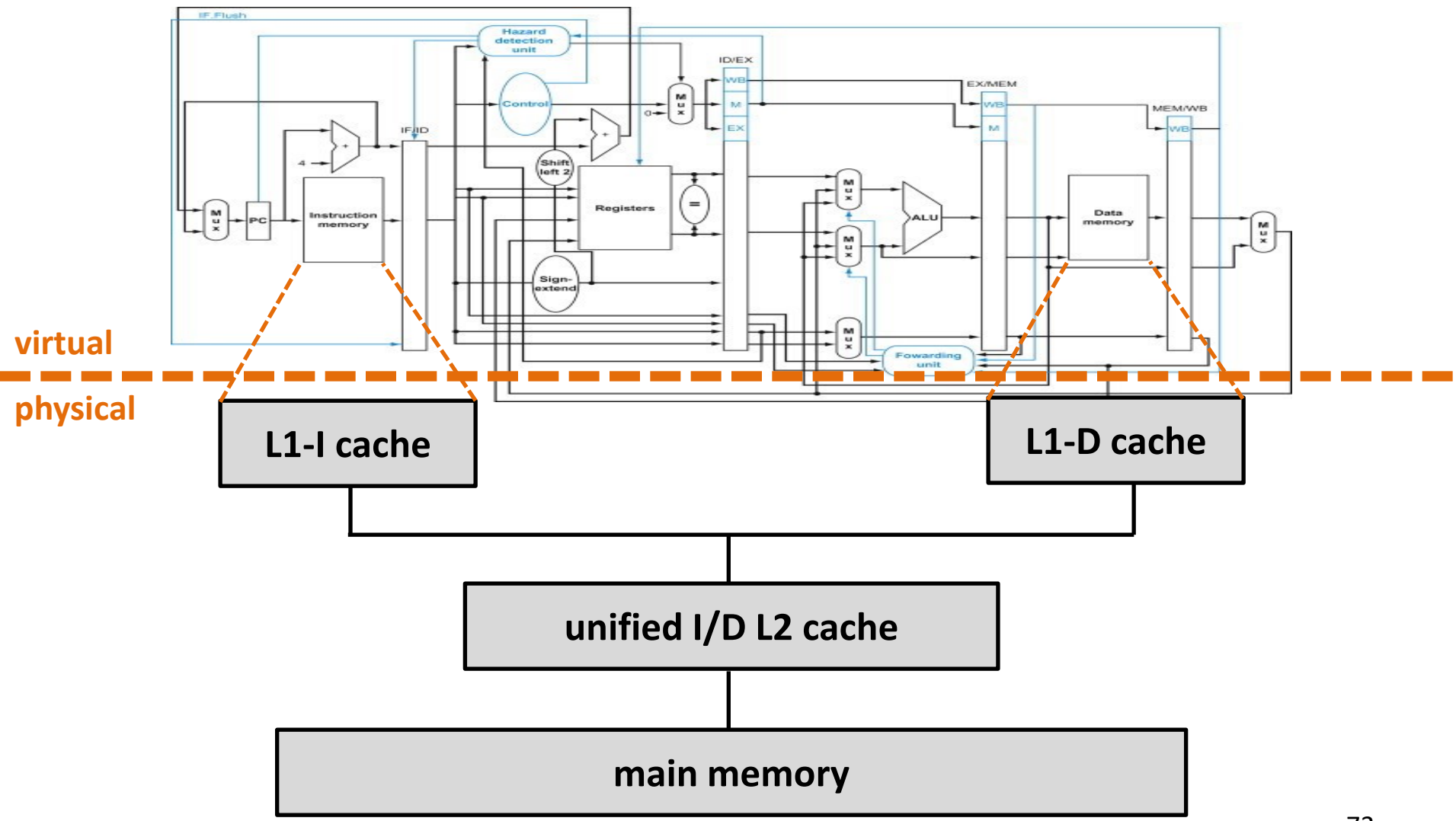
Compare this to simply storing the bits redundantly:

Hamming(7,4) + Parity (SECDED Codeword)	
W_1	$C_1 = b_1 \oplus b_2 \oplus b_4$
W_2	$C_2 = b_1 \oplus b_3 \oplus b_4$
W_3	b_1
W_4	$C_3 = b_2 \oplus b_3 \oplus b_4$
W_5	b_2
W_6	b_3
W_7	b_4
W_8	$P = W_1 \oplus \ldots \oplus W_7$

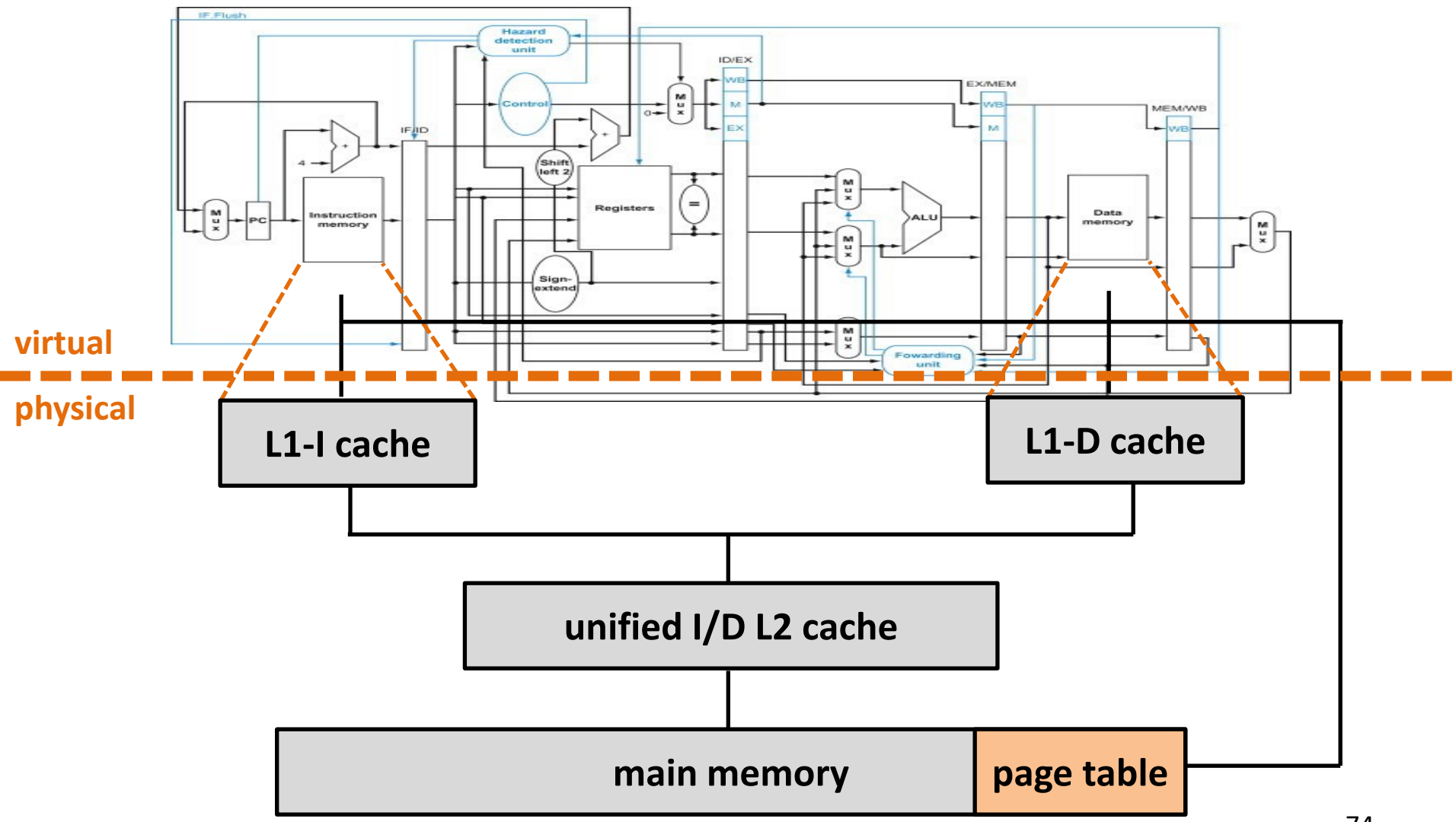
vs.

1-Bit Redundancy (SED Codeword)	
W_1	b_1
W_2	b_2
W_3	b_3
W_4	b_4
W_5	b_1
W_6	b_2
W_7	b_3
W_8	b_4

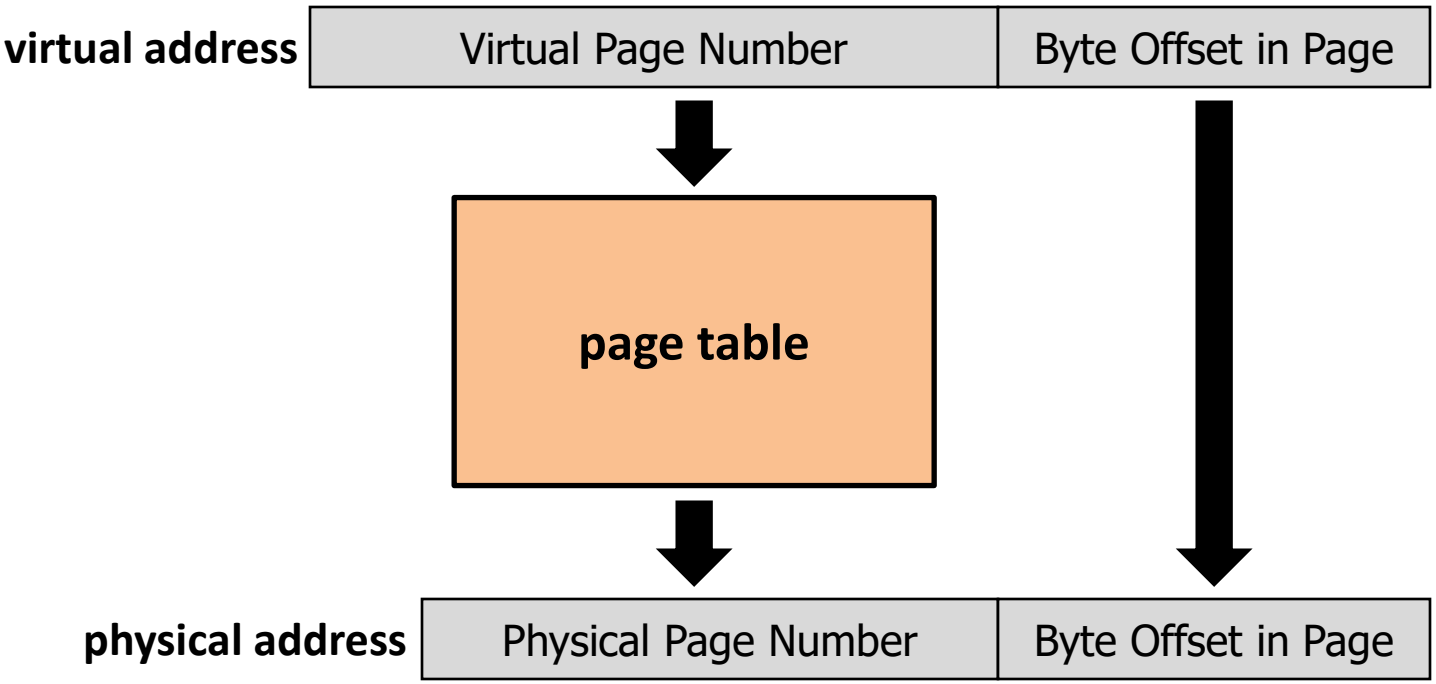
Virtual Memory



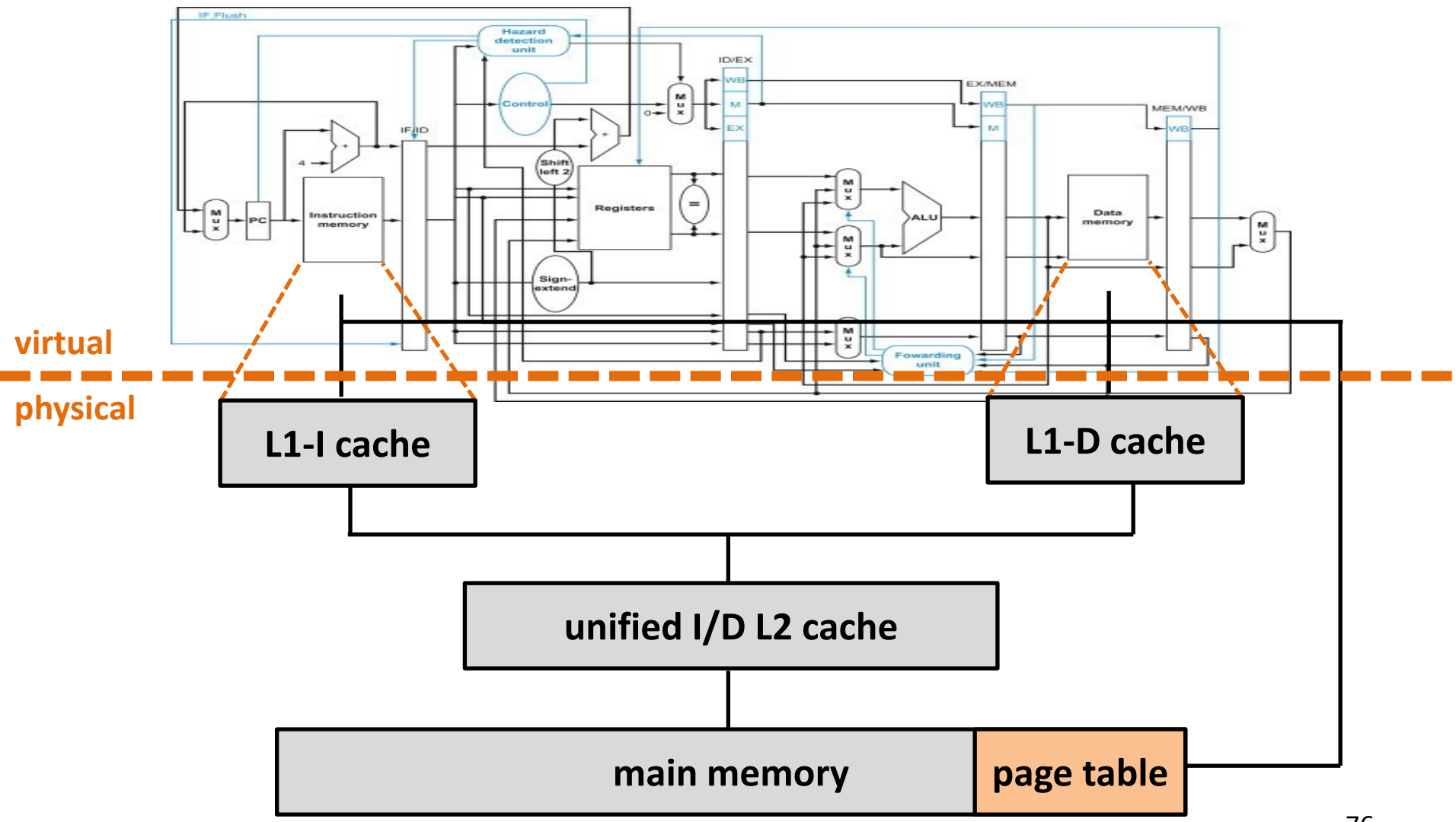
Virtual Memory



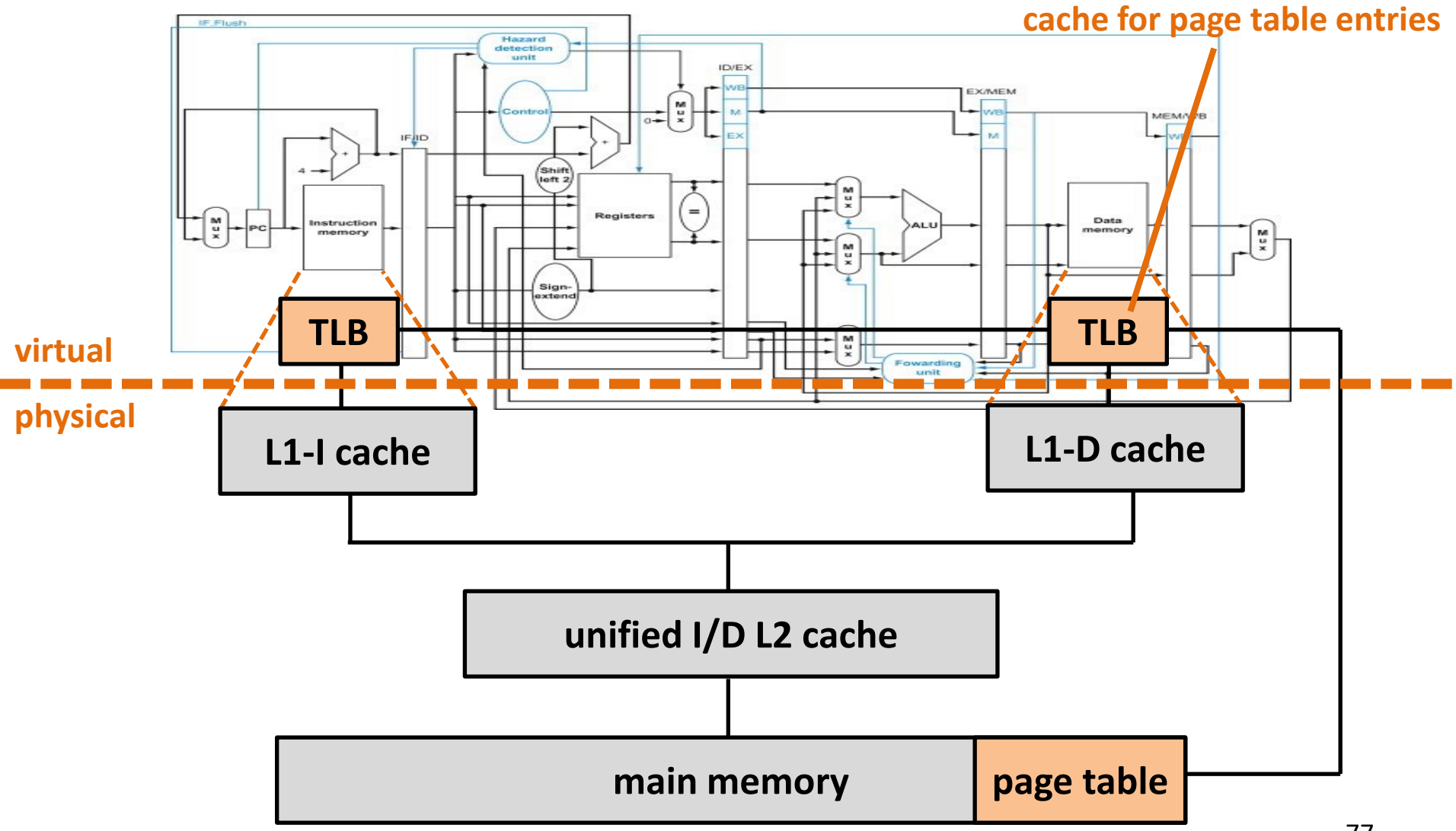
Virtual Memory – Page Table



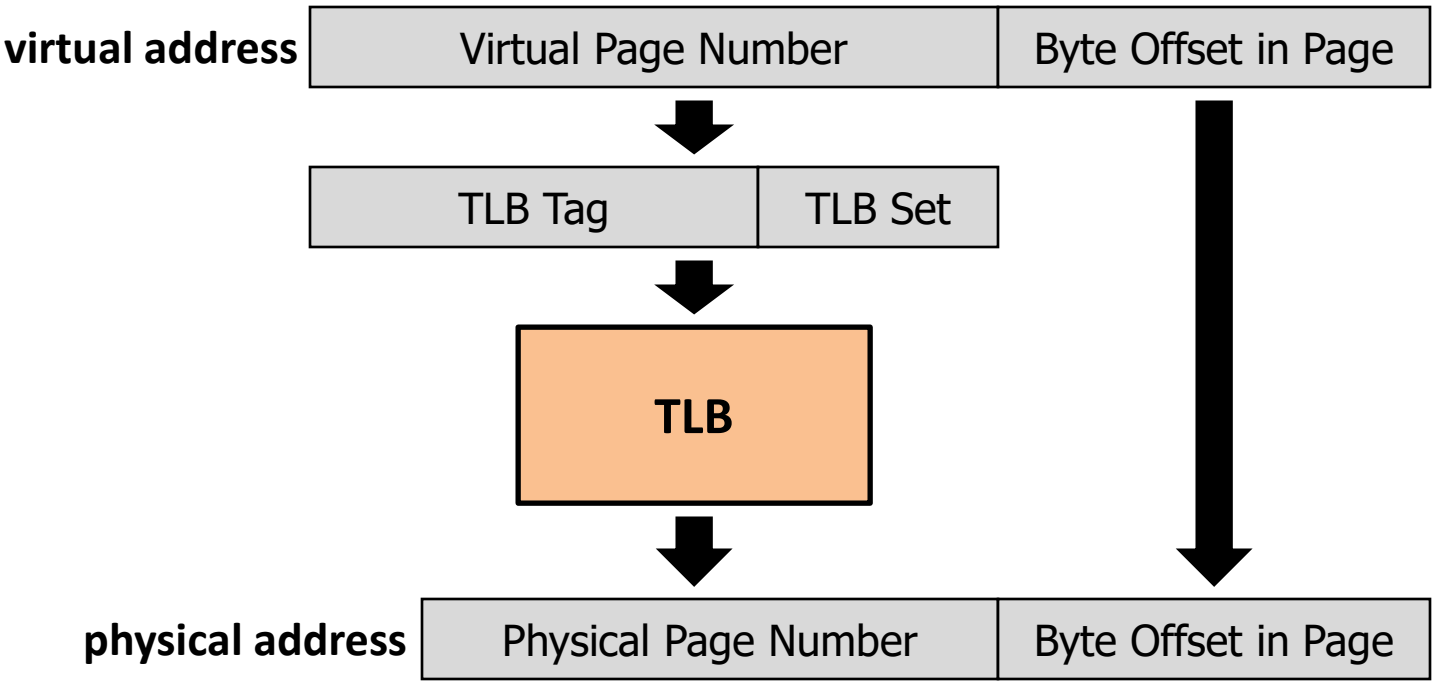
Virtual Memory



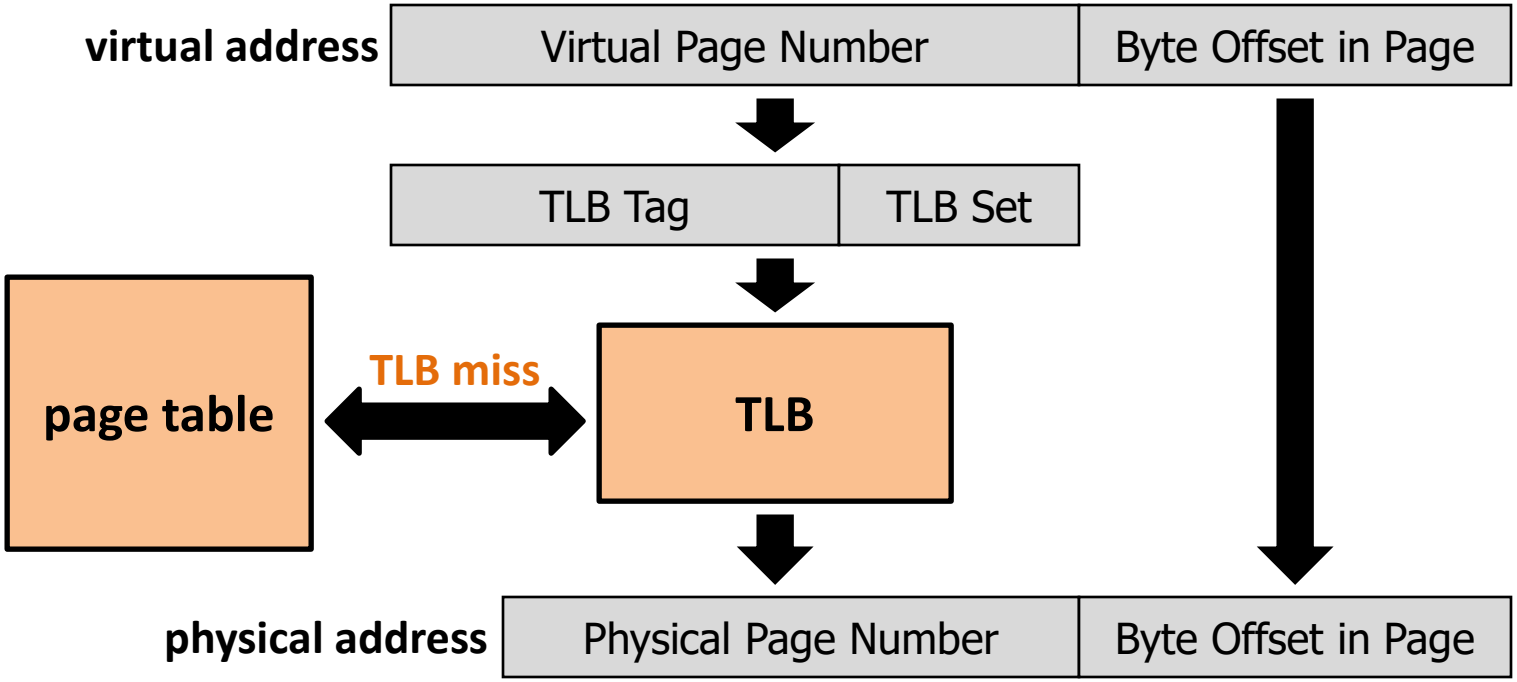
Virtual Memory



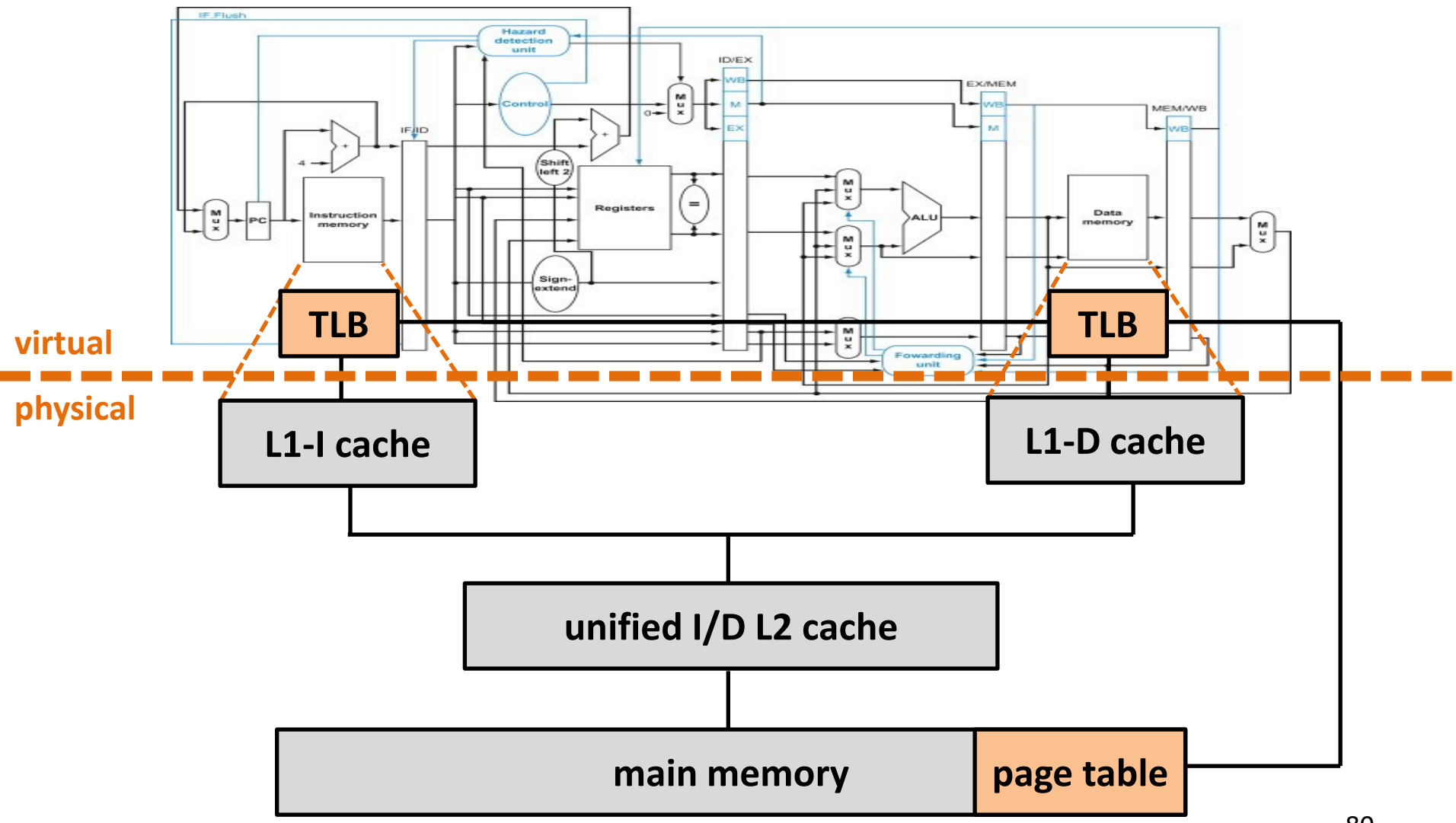
Virtual Memory – TLB



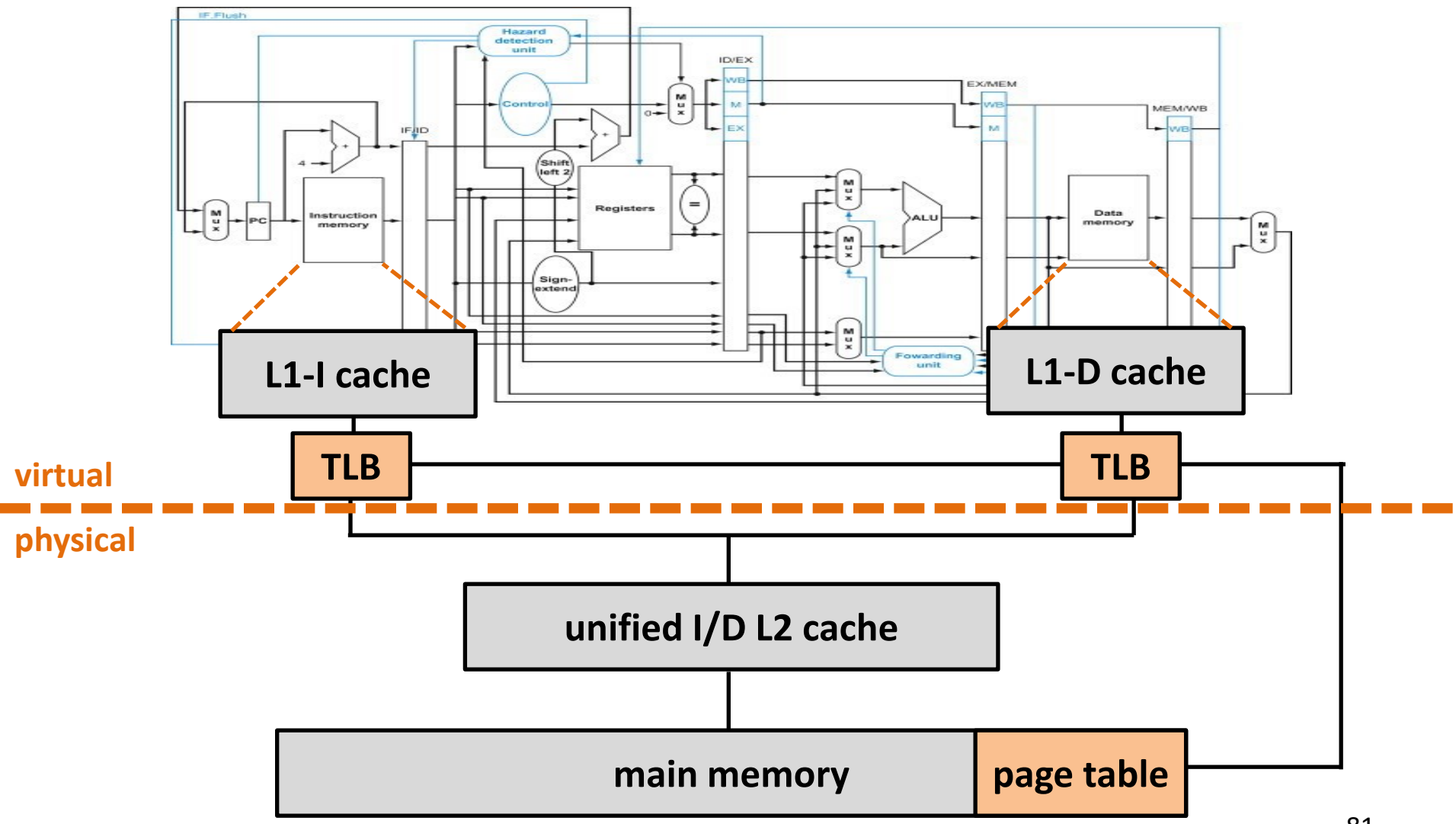
Virtual Memory – TLB



Virtual Memory



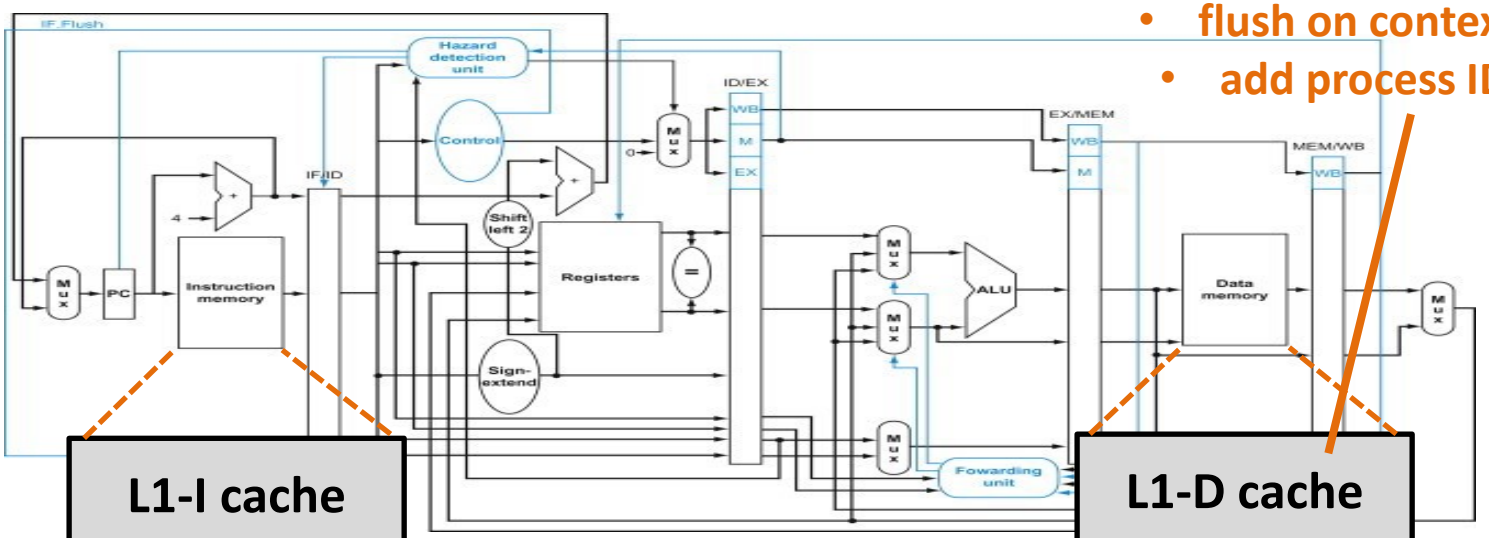
Virtual Memory



Virtual Memory

Homonyms: same VA, diff PA?

- flush on context switch
- add process ID to tag



L1-I cache

L1-D cache

TLB

TLB

virtual

physical

unified I/D L2 cache

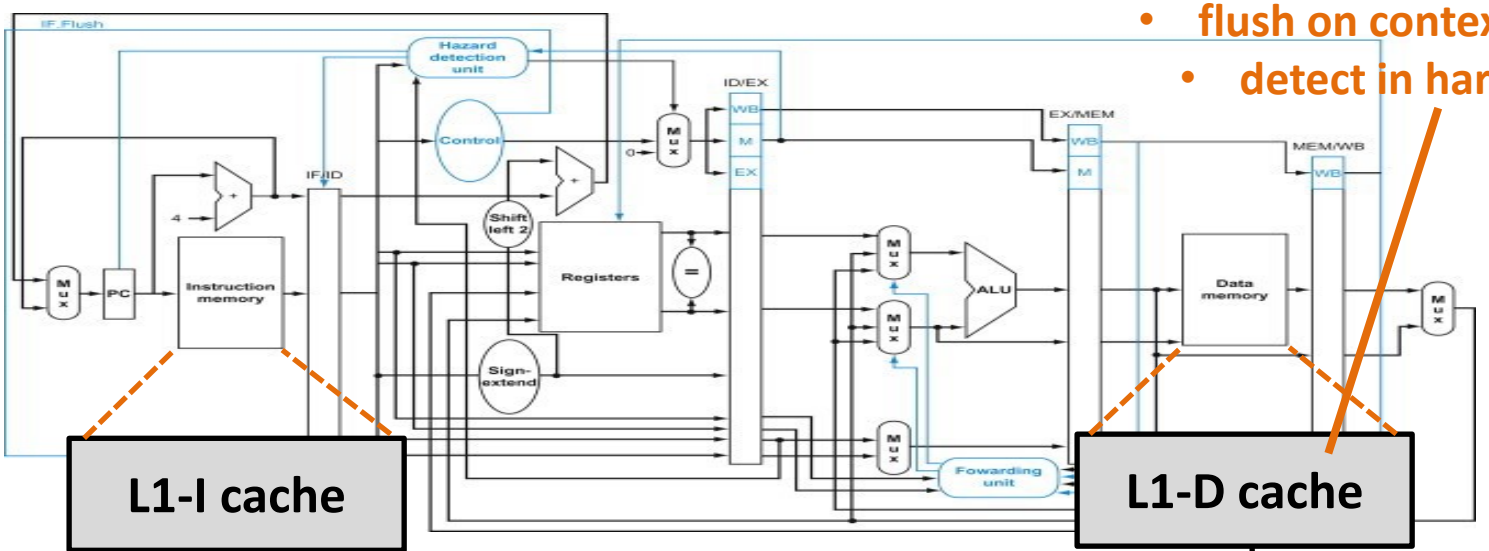
main memory

page table

Virtual Memory

Synonyms: diff VA, same PA?

- flush on context switch
- detect in hardware



L1-I cache

L1-D cache

TLB

TLB

virtual

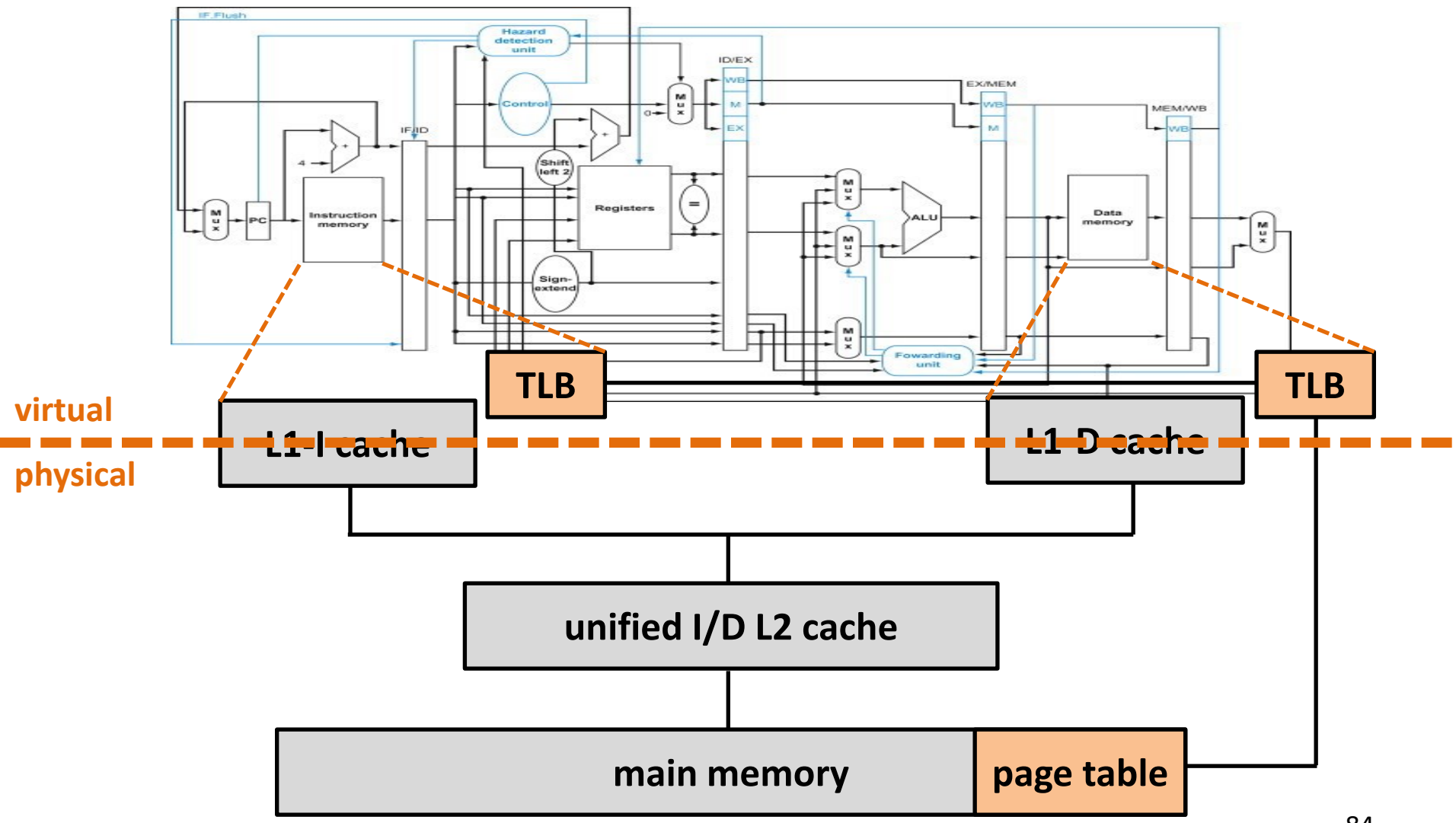
physical

unified I/D L2 cache

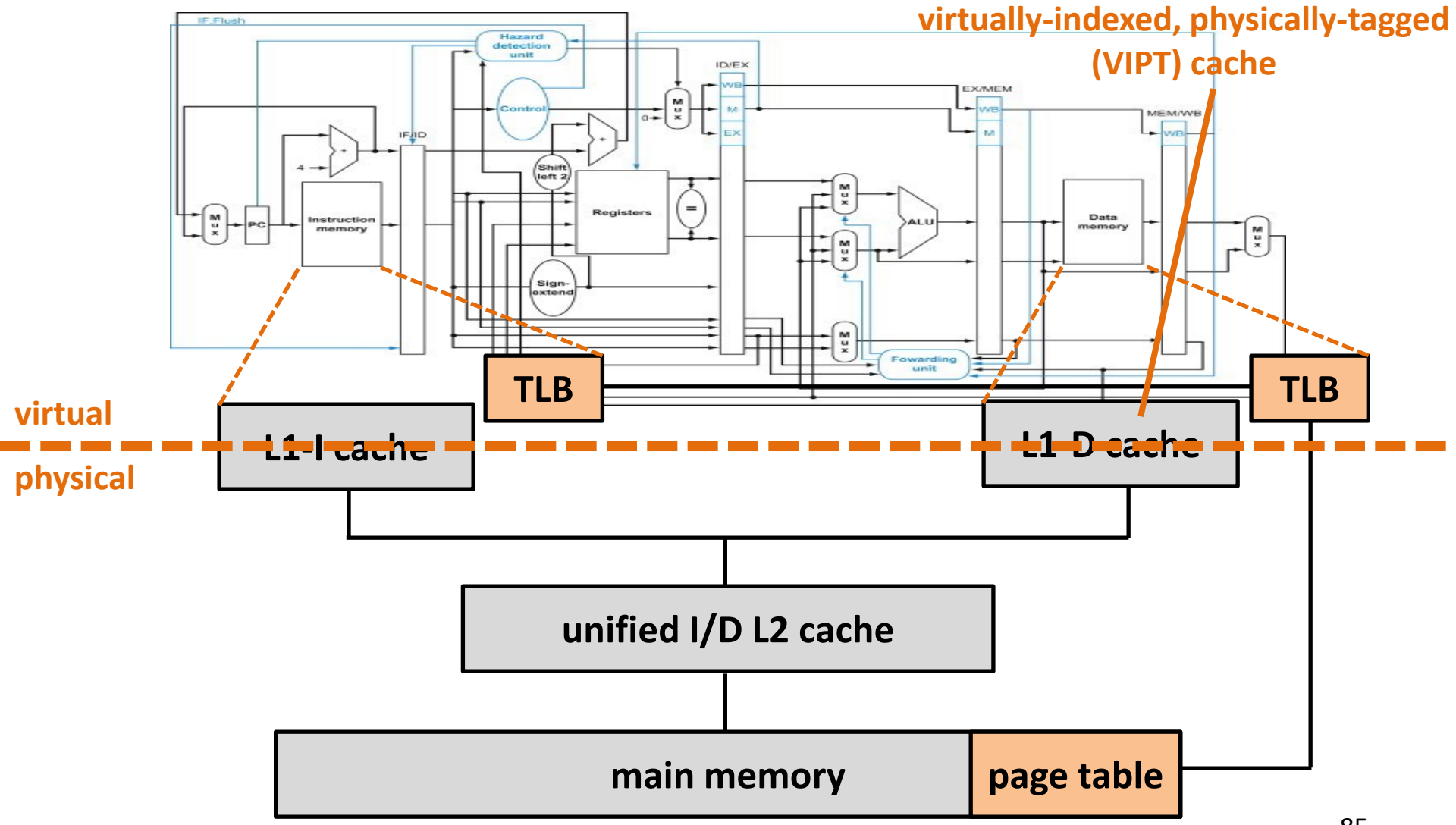
main memory

page table

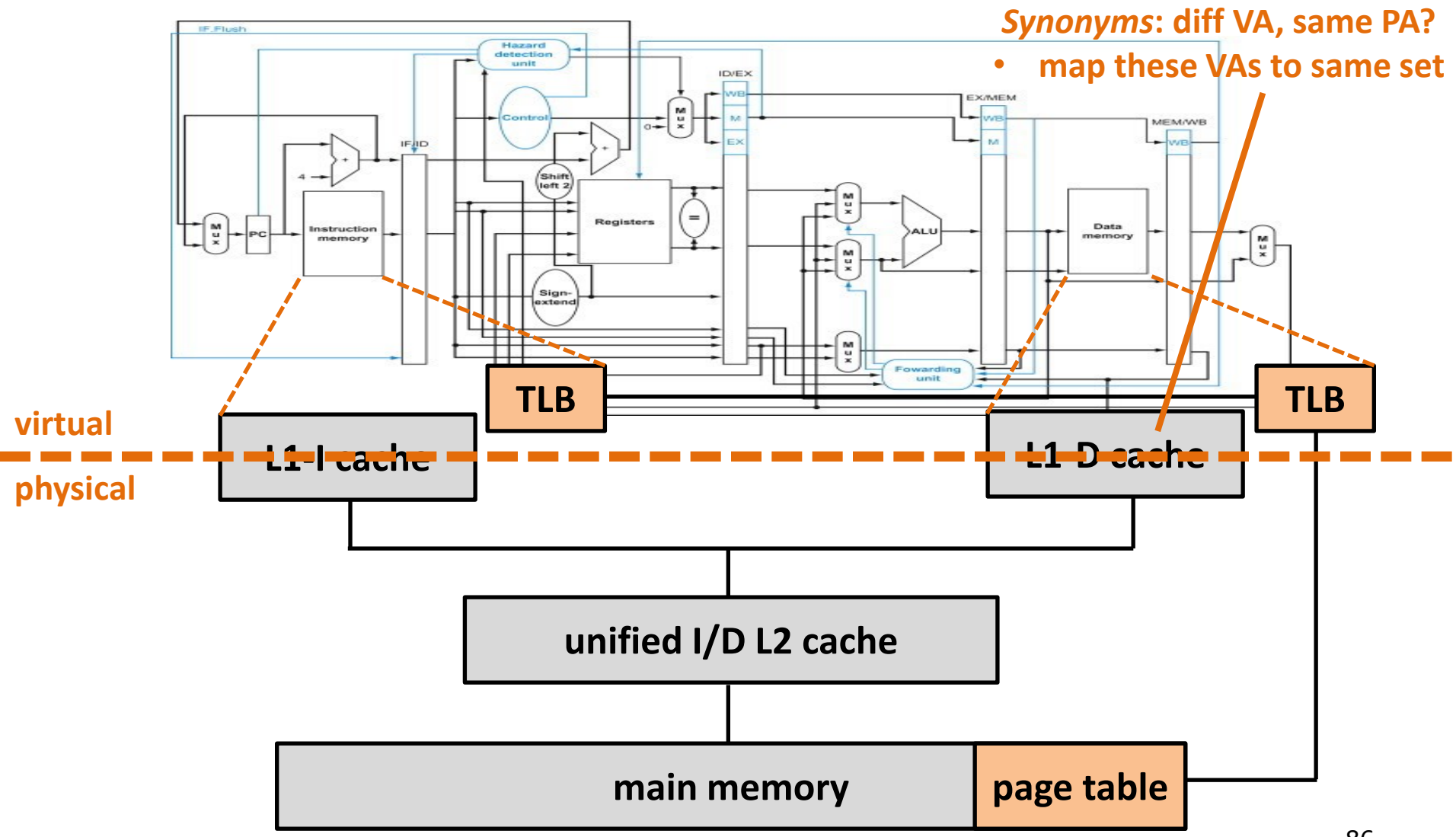
Virtual Memory



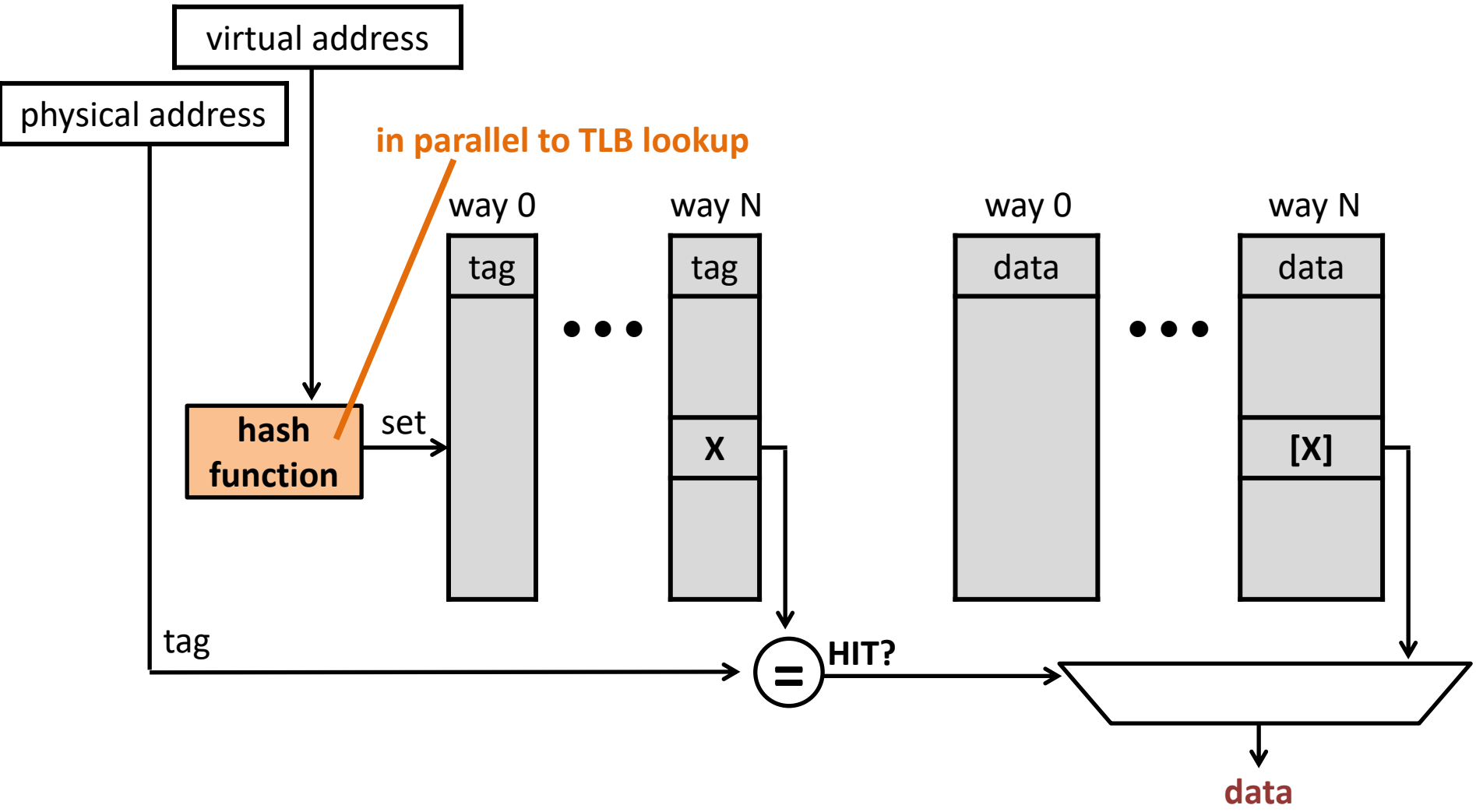
Virtual Memory



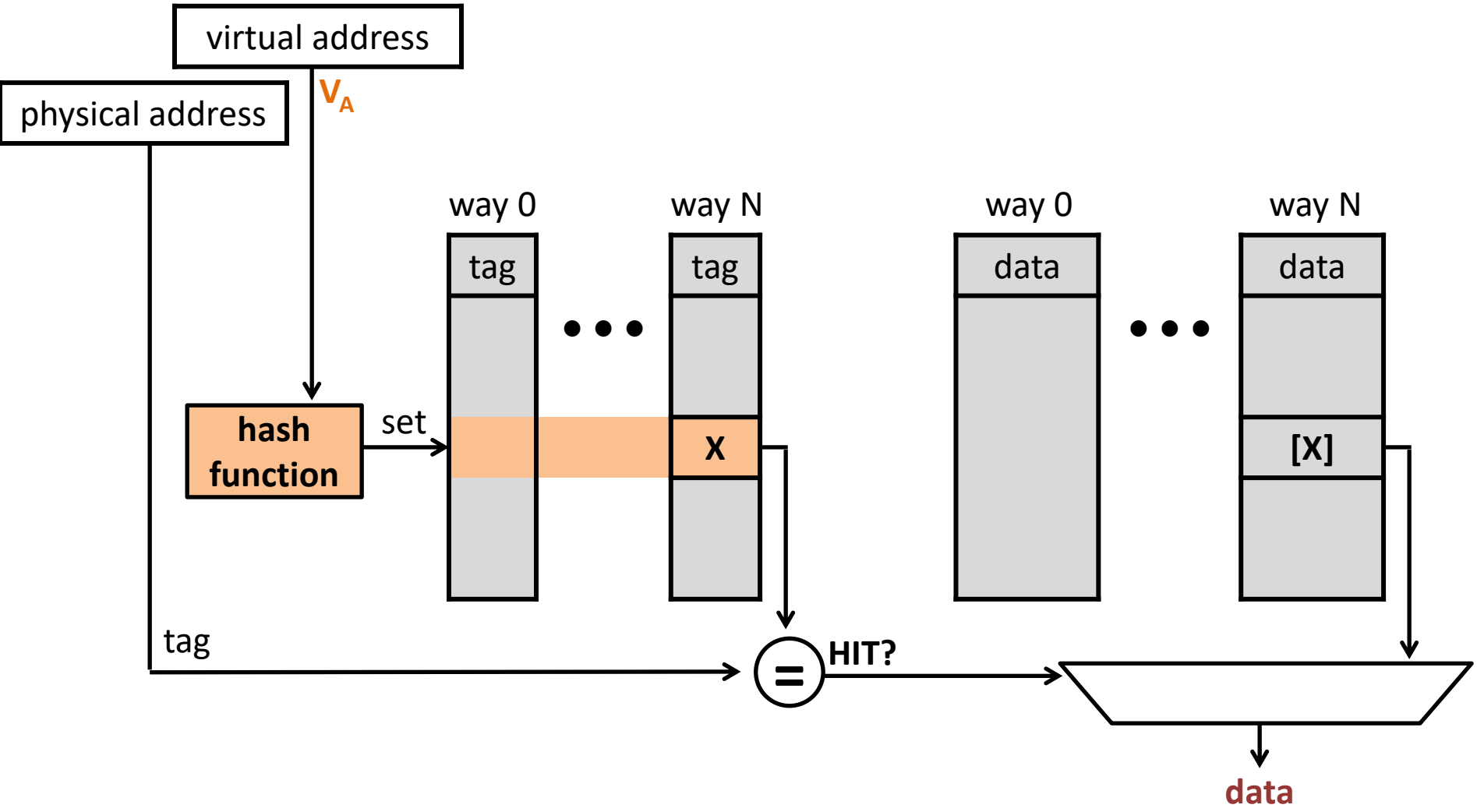
Virtual Memory



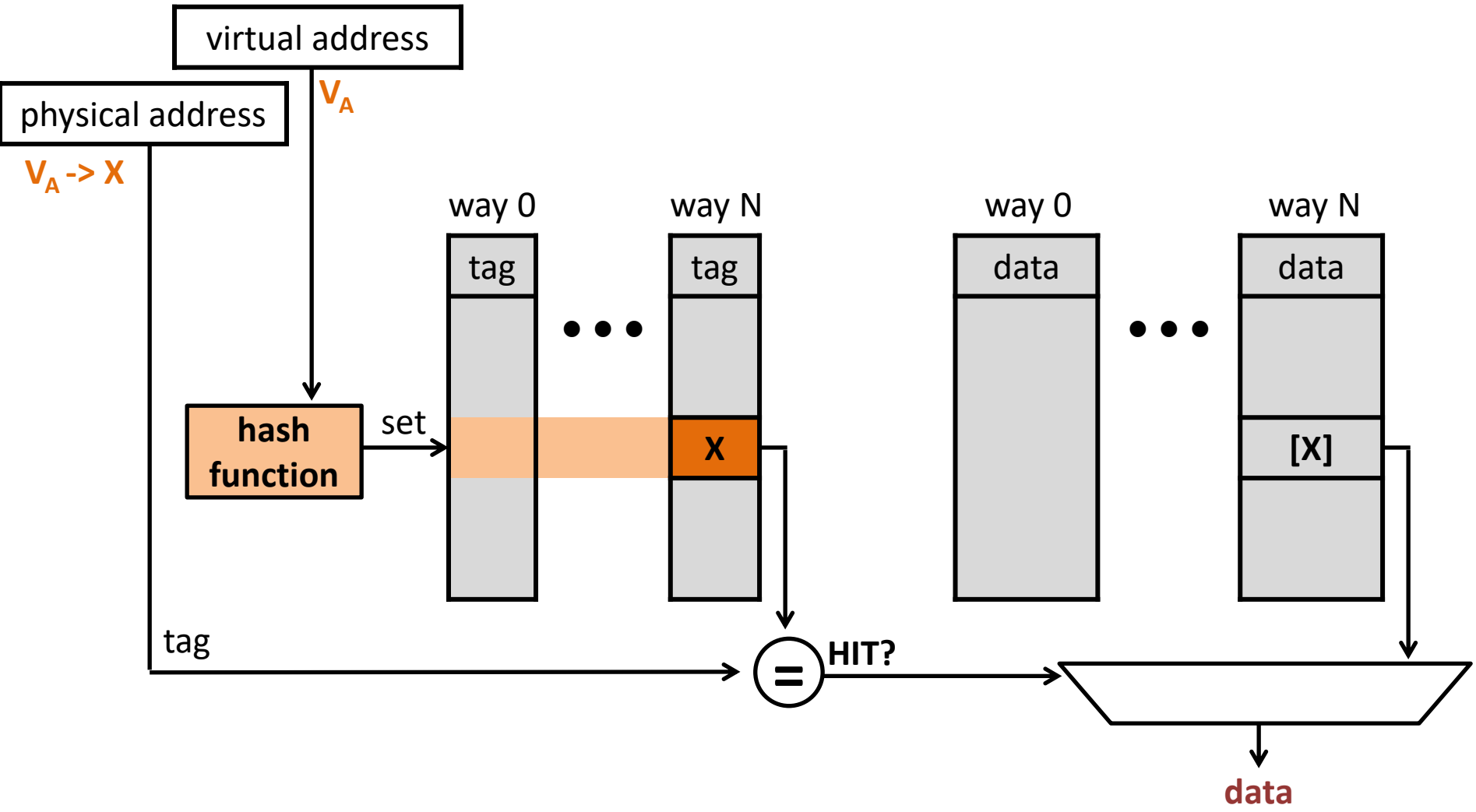
VIPT Cache


virtual address
physical address
in parallel to TLB lookup
way 0
way N
way 0
way N
tag
tag
data
data
hash function
set
X
[X]
tag
=
HIT?
data

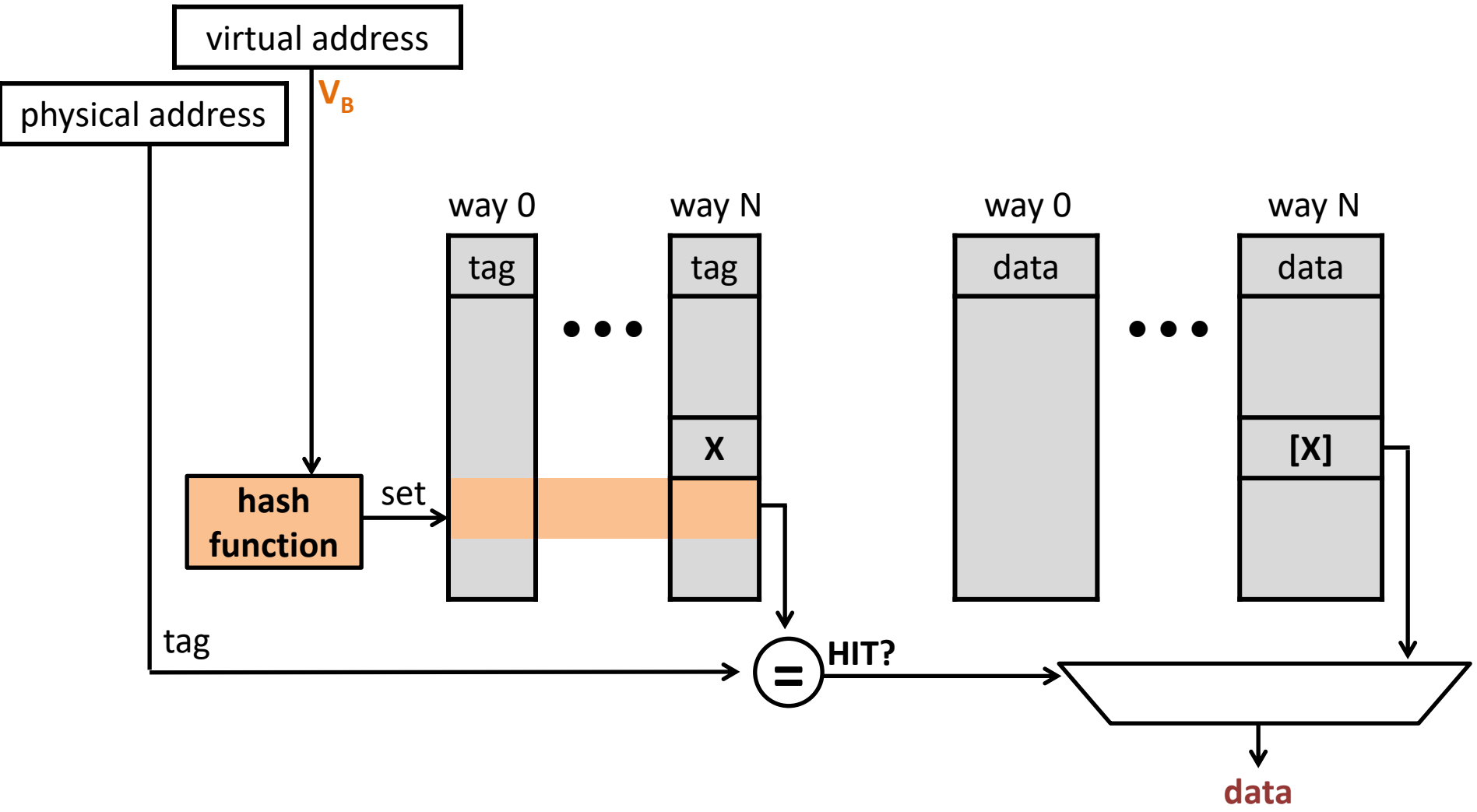
VIPT Cache – Synonyms



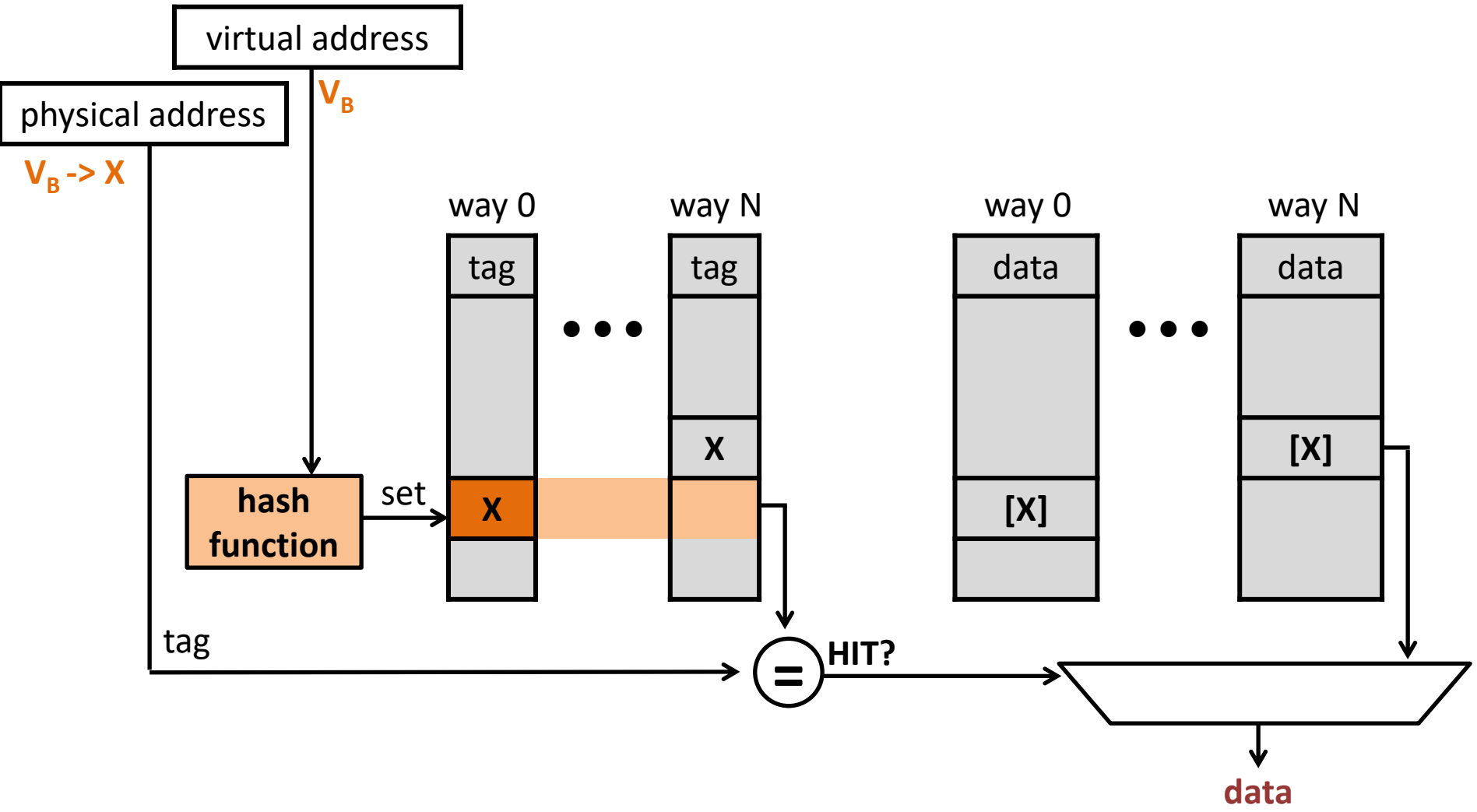
VIPT Cache – Synonyms



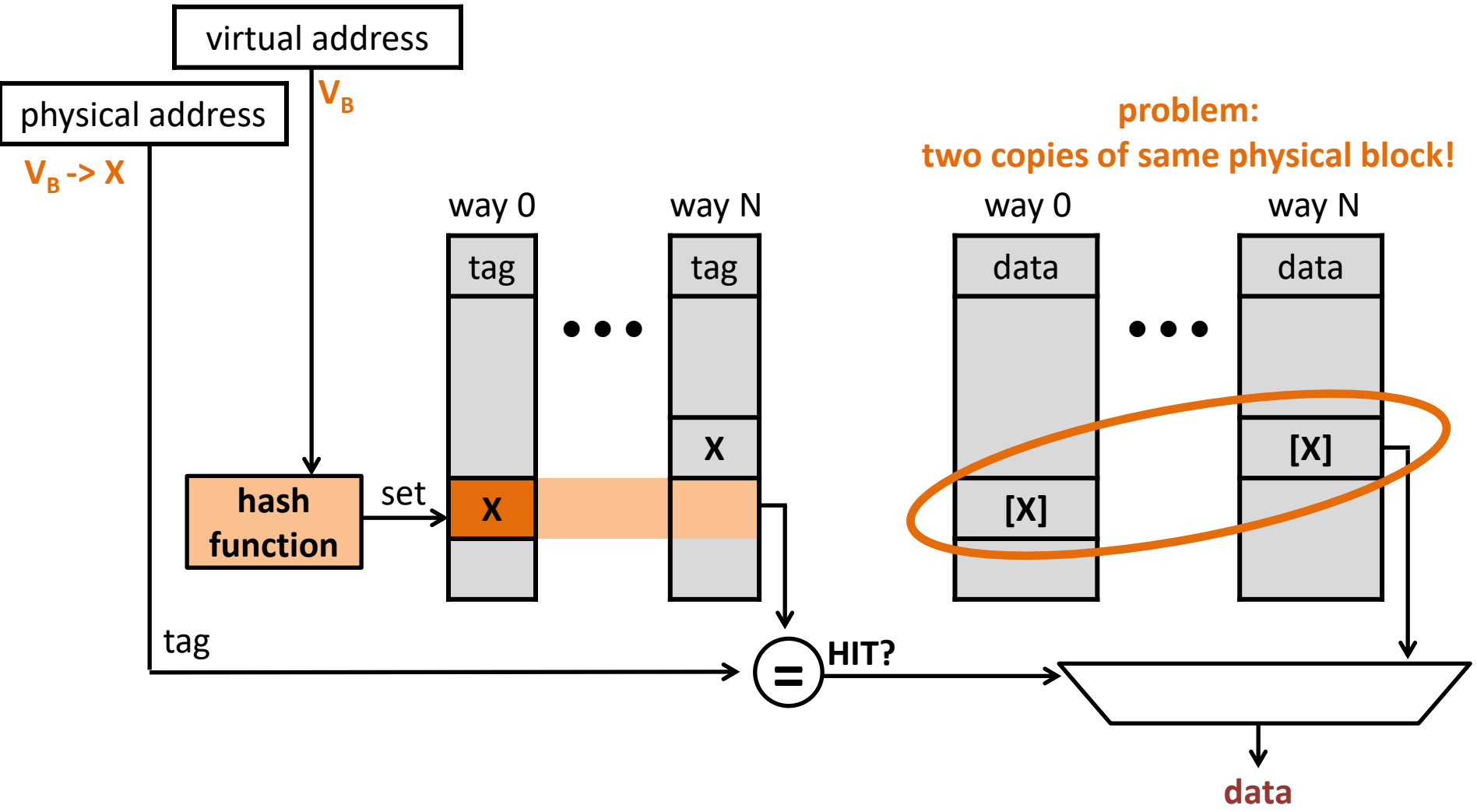
VIPT Cache – Synonyms



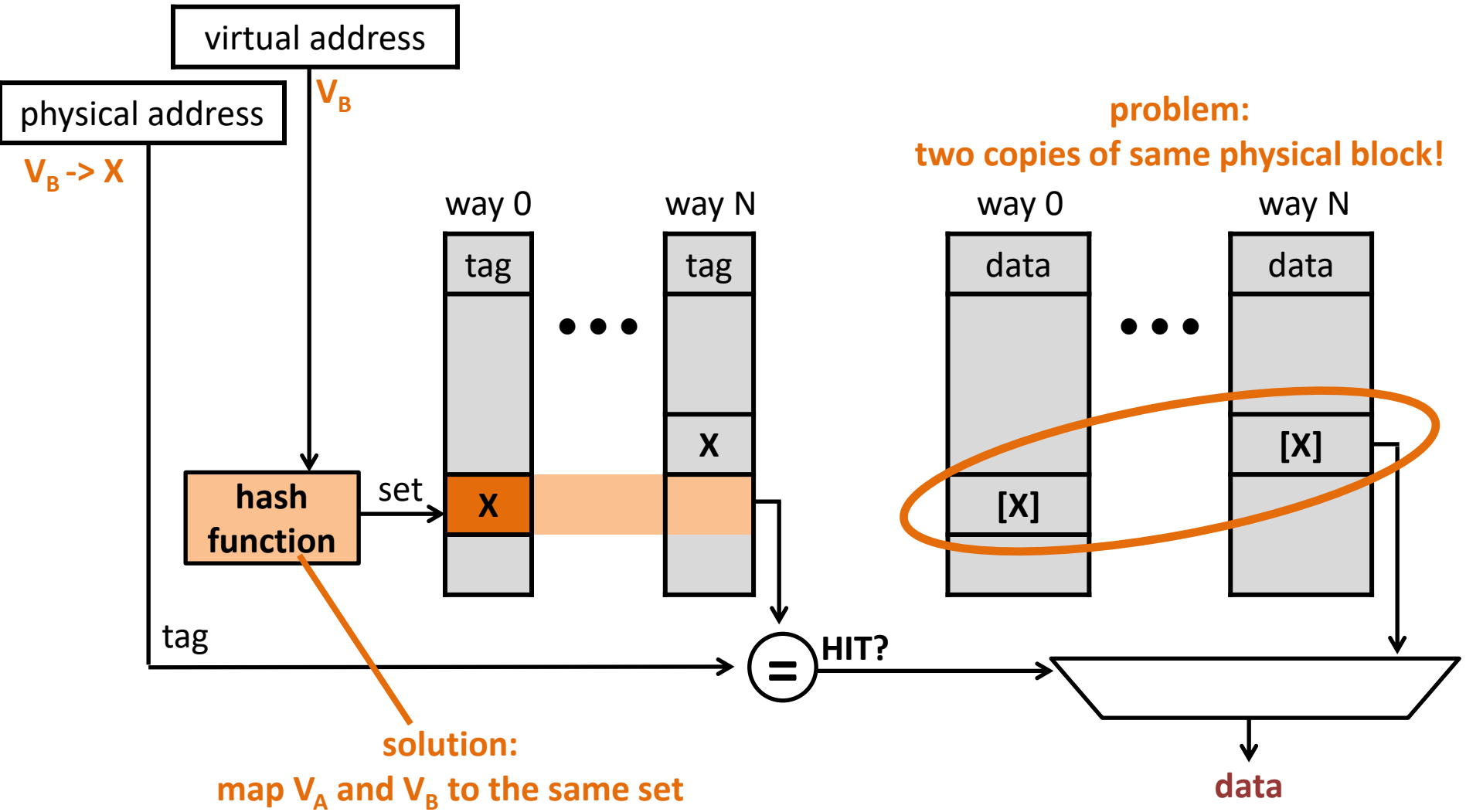
VIPT Cache – Synonyms



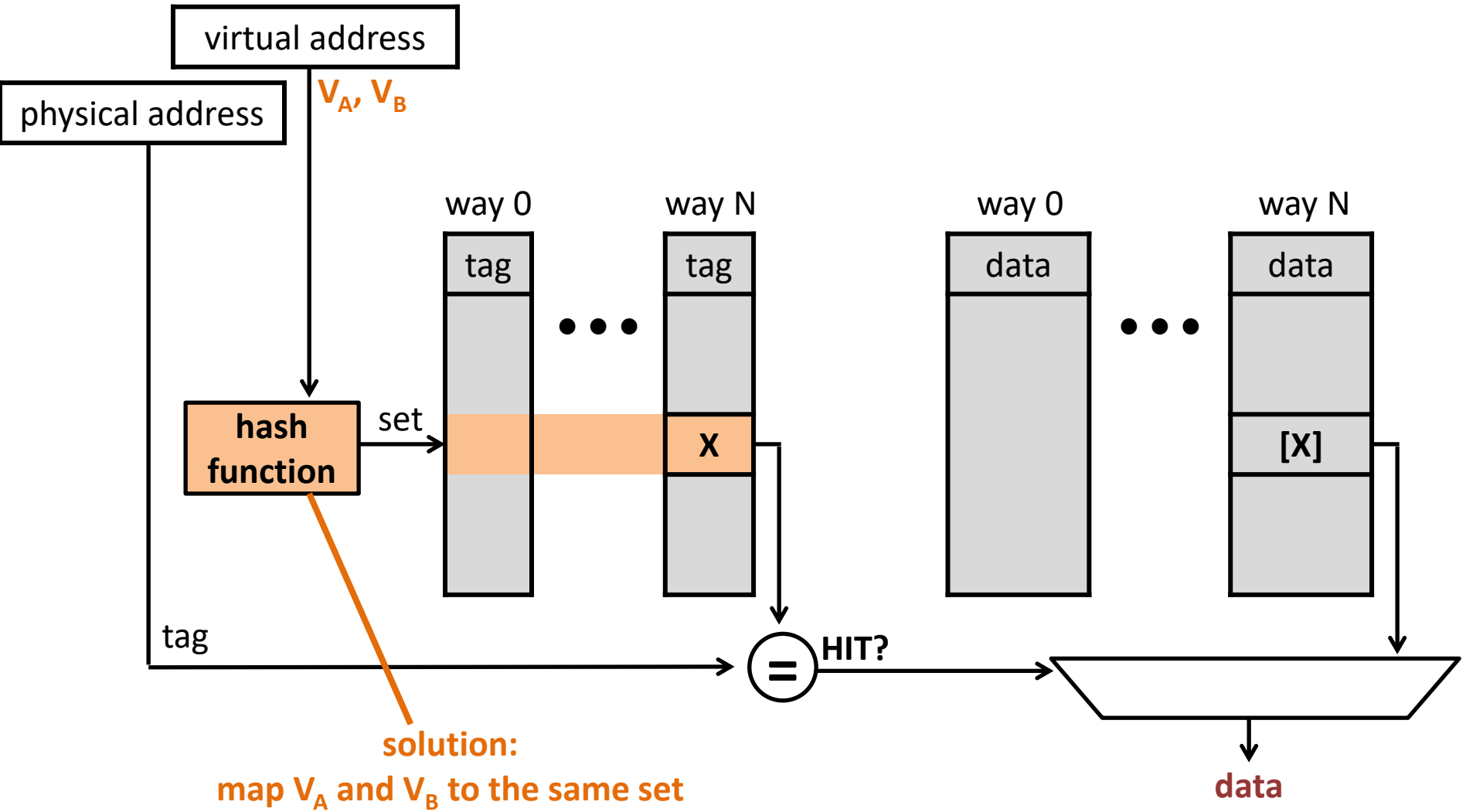
VIPT Cache – Synonyms



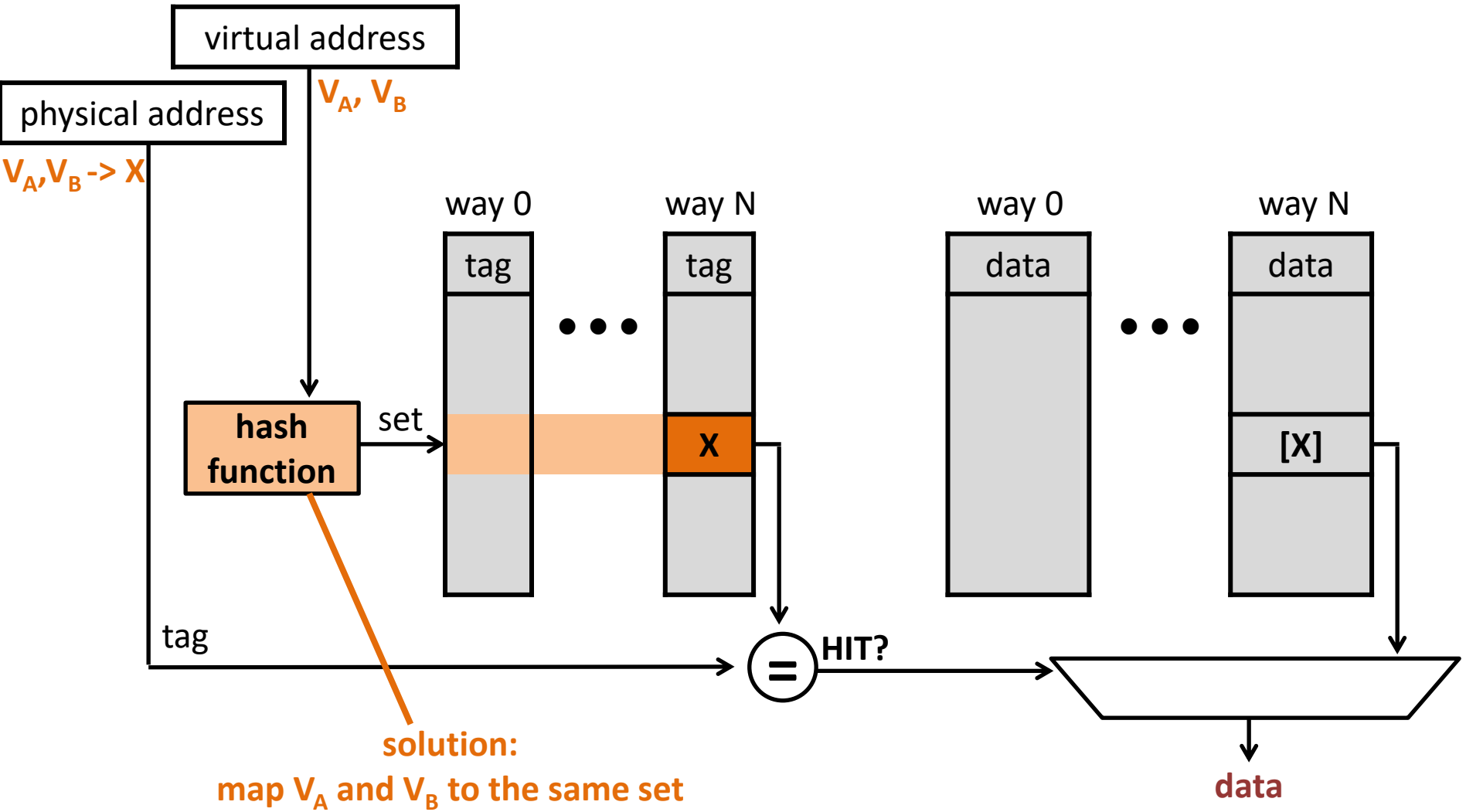
VIPT Cache – Synonyms



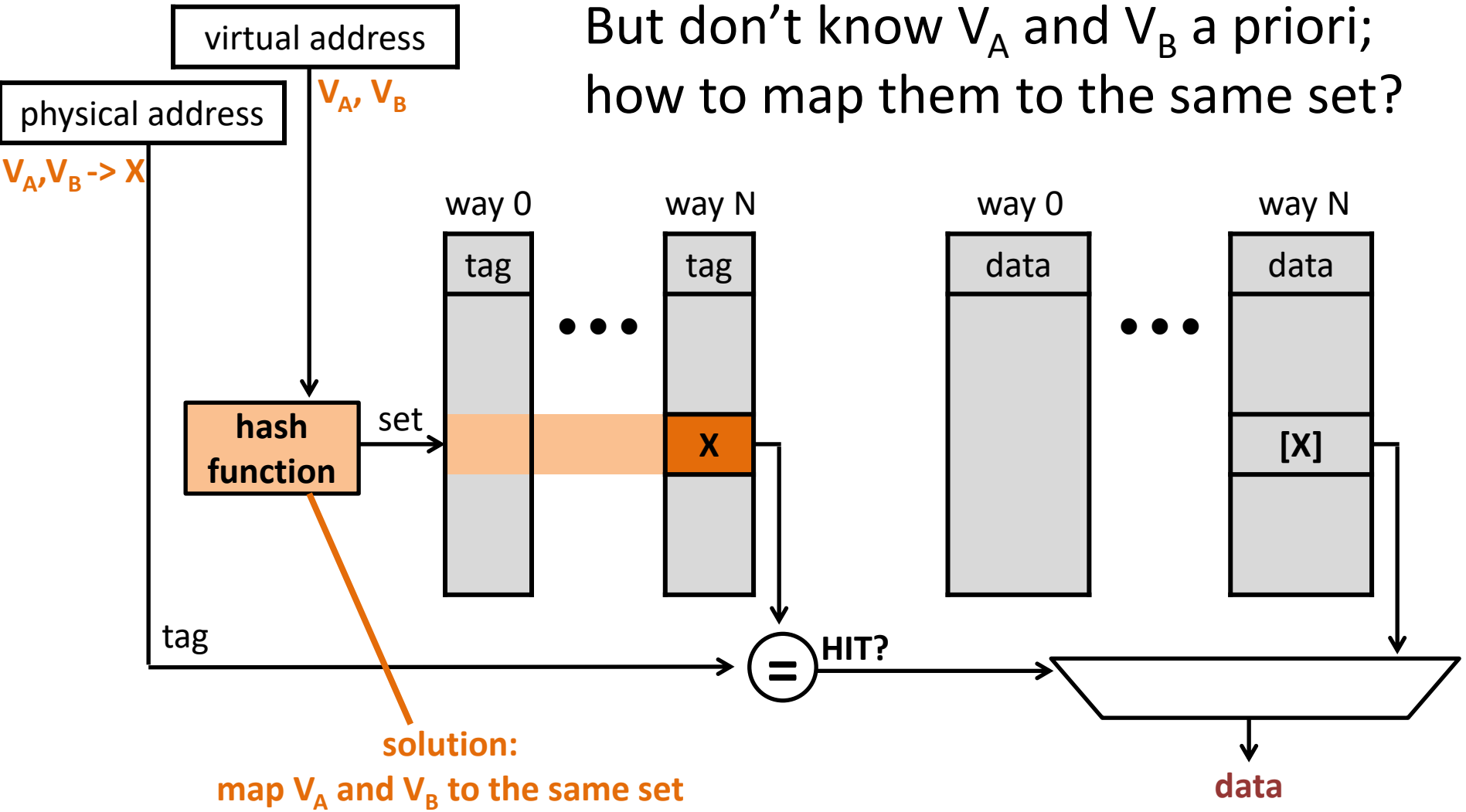
VIPT Cache – Synonyms



VIPT Cache – Synonyms



VIPT Cache – Synonyms



VIPT Cache – Synonyms

e.g., Assume 4KB pages. If the physical address of our data is **0x1A318ED4**, and virtual addresses V_A and V_B both map to it, what do we know about V_A and V_B?

VIPT Cache – Synonyms

e.g., Assume 4KB pages. If the physical address of our data is **0x1A318ED4**, and virtual addresses V_A and V_B both map to it, what do we know about V_A and V_B?

V_A = 0x_____ED4

V_B = 0x_____ED4

VIPT Cache – Synonyms

e.g., Assume 4KB pages. If the physical address of our data is **0x1A318ED4**, and virtual addresses V_A and V_B both map to it, what do we know about V_A and V_B?

V_A = 0x_____ED4

V_B = 0x_____ED4

So then without knowing what V_A and V_B exactly are, how can we make sure they always map to the same set in our VIPT cache?

Handing Synonyms

- Avoid synonyms if:
  - cache index and offset to come from page offset
  - Why? These bits are the **same** for VA and PA
- Design decision:
  - Limits number of entries in a way in L1 caches
  - Led CPU L1 caches to increase associativity
    - Increases L1 cache size without causing synonyms
- Key question:
  - (cache block offset + index) bits ≤ page offset bits?