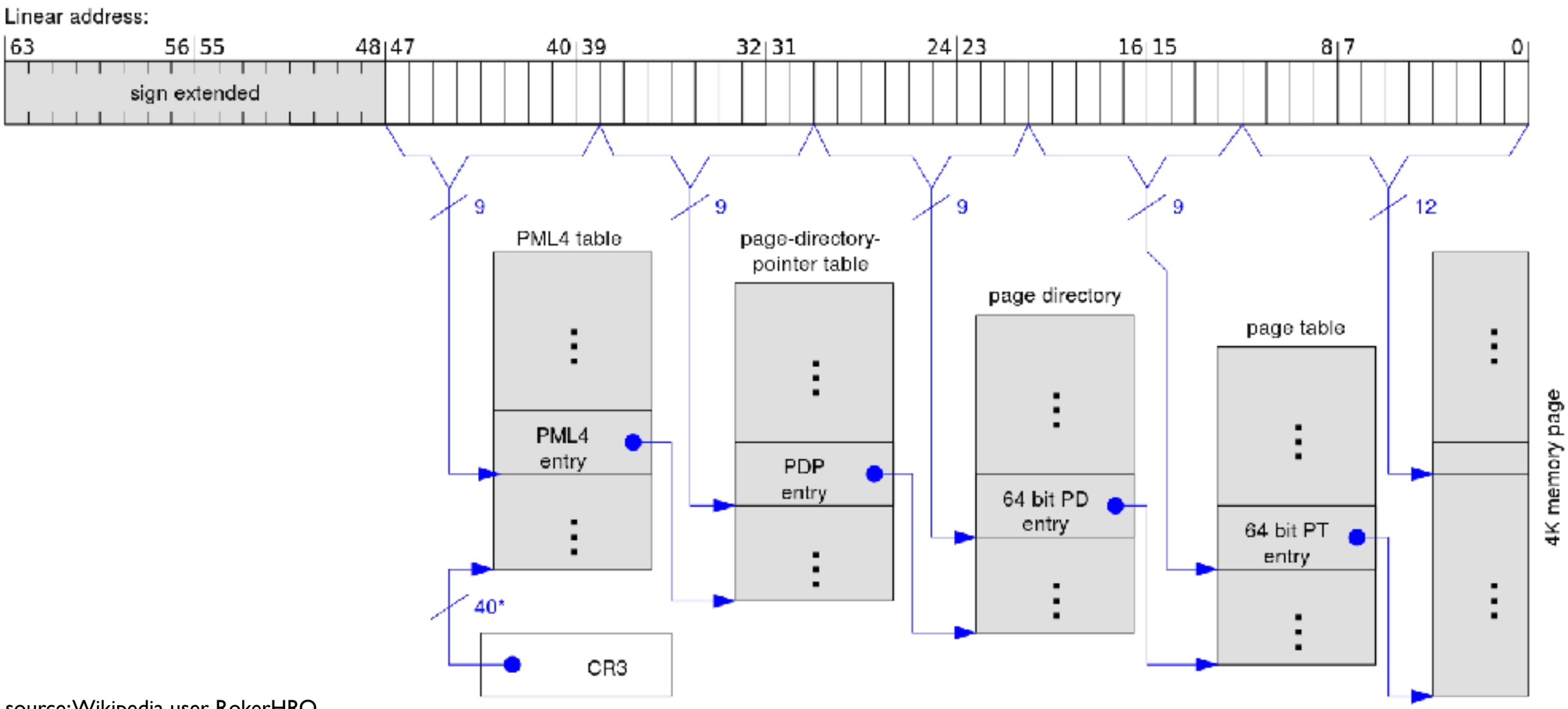
Address Translation

Protection and Performance

Hammurabi Mendes

Fall 18

Superpages

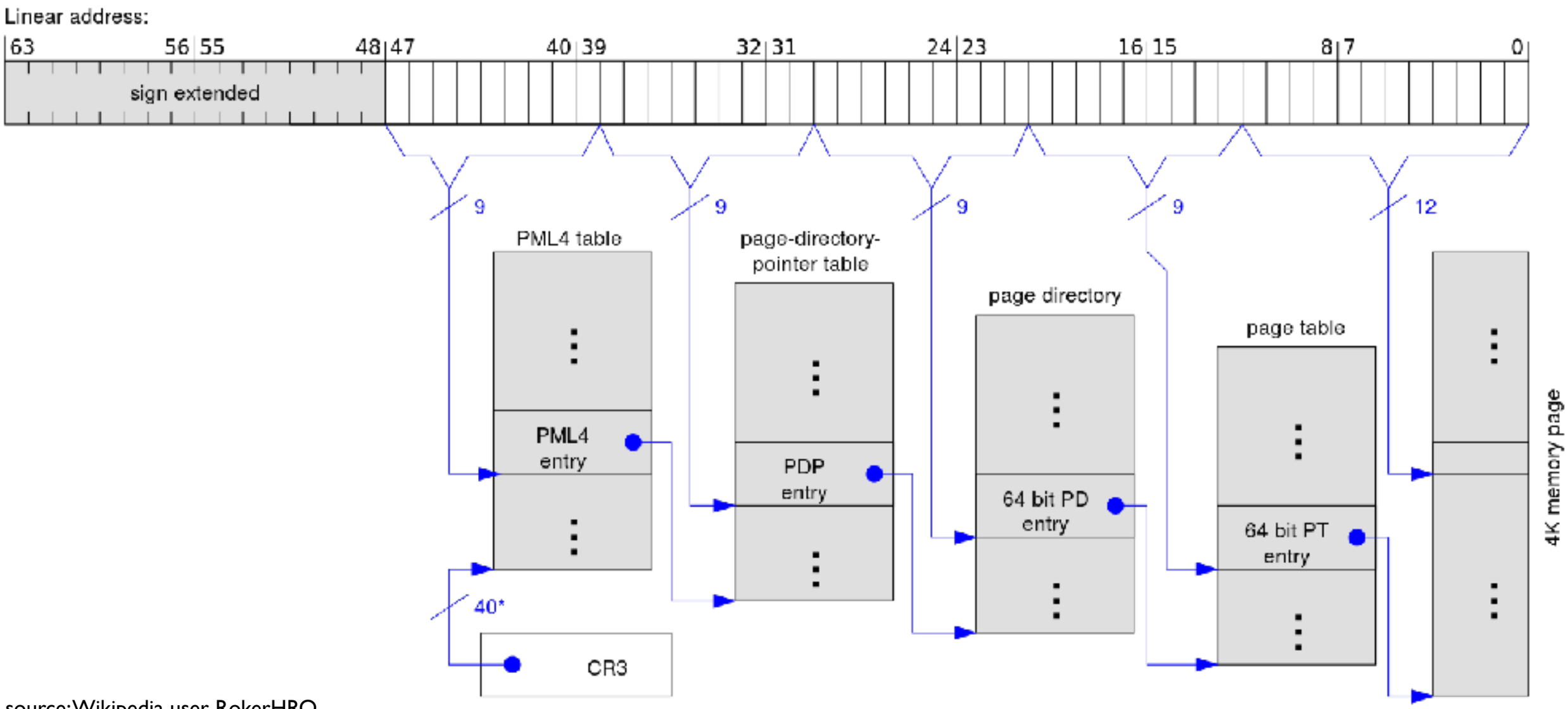


* source: Wikipedia user RokerHRO

Superpages

Use one entry to map a single 1GB chunk

without allocating the $2^9 \times 2^9$ entries (2^9 tables) underneath

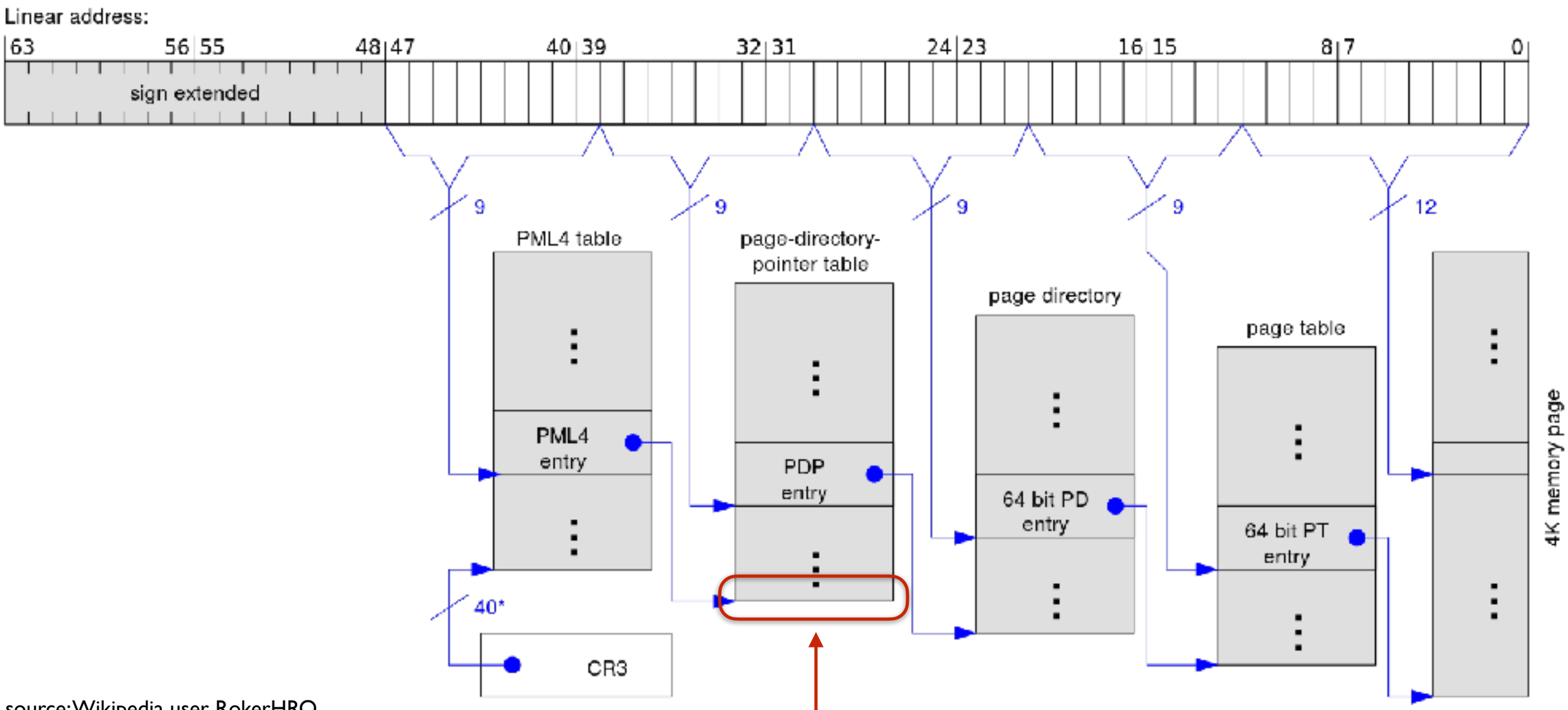


* source: Wikipedia user RokerHRO

Superpages

Use one entry to map a single 1GB chunk

without allocating the $2^9 \times 2^9$ entries (2^9 tables) underneath



* source: Wikipedia user RokerHRO

Entries have flags - set the flag for "superpage"

Need for Speed

Need for Speed

- CPU capabilities:

Need for Speed

- CPU capabilities:
  - 2.5GHz = 2500000000 cycles/s

Need for Speed

- CPU capabilities:
  - 2.5GHz = 2500000000 cycles/s
  - Instructions per cycle: ~2

Need for Speed

- CPU capabilities:
  - 2.5GHz = 2500000000 cycles/s
  - Instructions per cycle: ~2
- Perspective on memory access cost*:

* *Rough* numbers on Intel Sandy Bridge line

Need for Speed

- CPU capabilities:
  - 2.5GHz = 2500000000 cycles/s
  - Instructions per cycle: ~2
- Perspective on memory access cost*:
  - CPU Internal Registers: 1 cycle

* *Rough* numbers on Intel Sandy Bridge line

Need for Speed

- CPU capabilities:
  - 2.5GHz = 2500000000 cycles/s
  - Instructions per cycle: ~2
- Perspective on memory access cost*:
  - CPU Internal Registers: 1 cycle
  - Main memory: 65ns (~150 cycles)

* *Rough* numbers on Intel Sandy Bridge line

Need for Speed

- CPU capabilities:
  - 2.5GHz = 2500000000 cycles/s
  - Instructions per cycle: ~2
- Perspective on memory access cost*:
  - CPU Internal Registers: 1 cycle
  - Main memory: 65ns (~150 cycles)
- Address translation

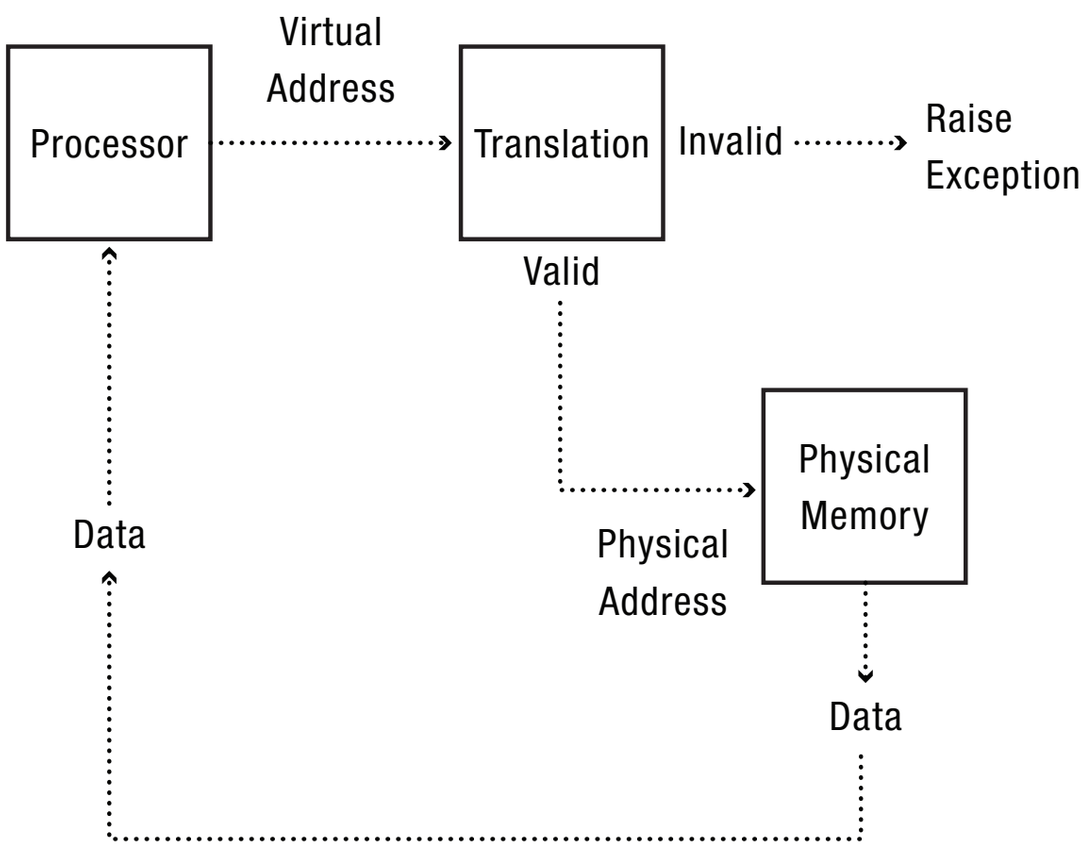
* *Rough* numbers on Intel Sandy Bridge line

Need for Speed

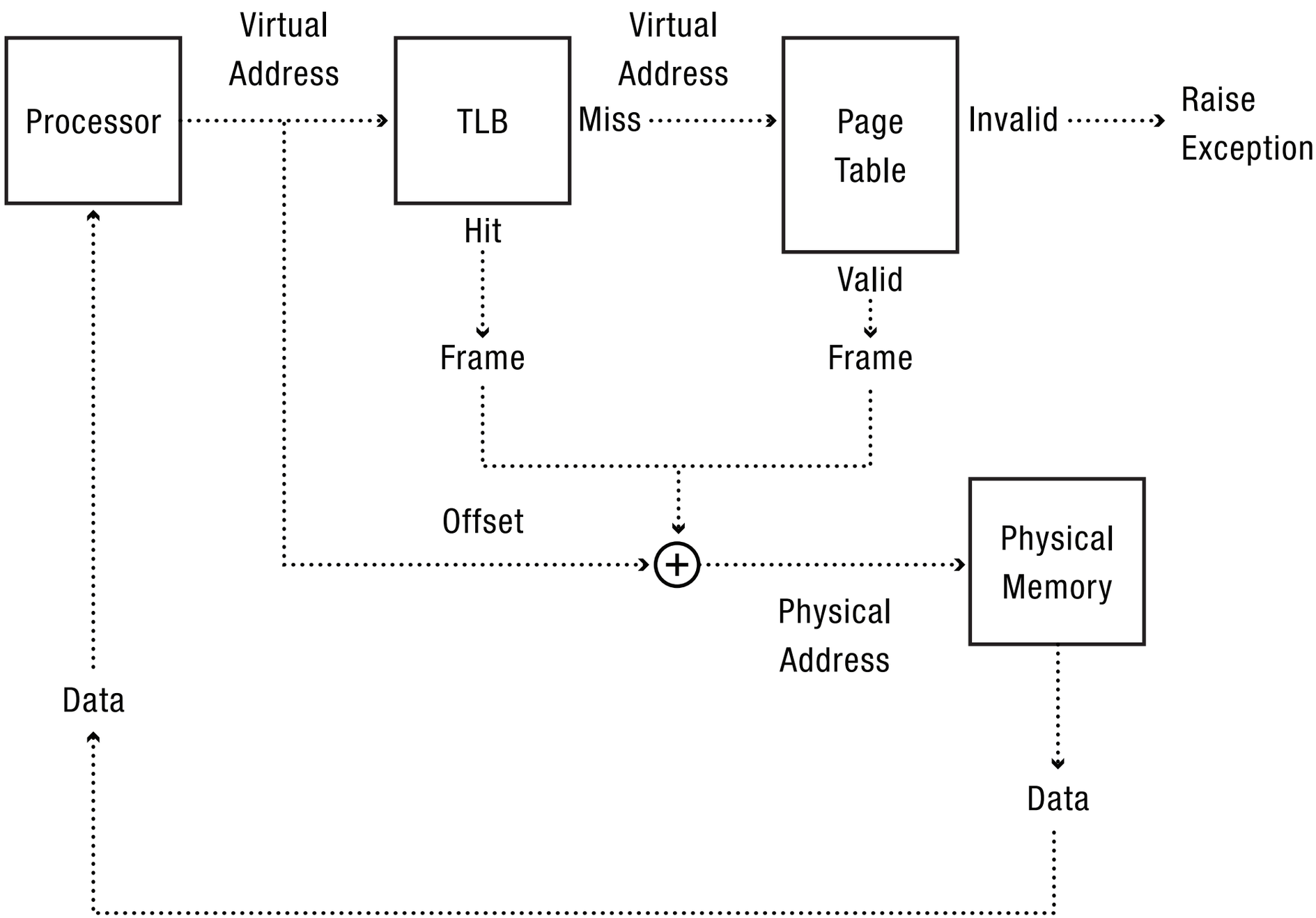
- CPU capabilities:
  - 2.5GHz = 2500000000 cycles/s
  - Instructions per cycle: ~2
- Perspective on memory access cost*:
  - CPU Internal Registers: 1 cycle
  - Main memory: 65ns (~150 cycles)
- Address translation
  - Consulting page tables makes memory references cost *3* or *4* times as much (*600 cycles!*)

* *Rough* numbers on Intel Sandy Bridge line

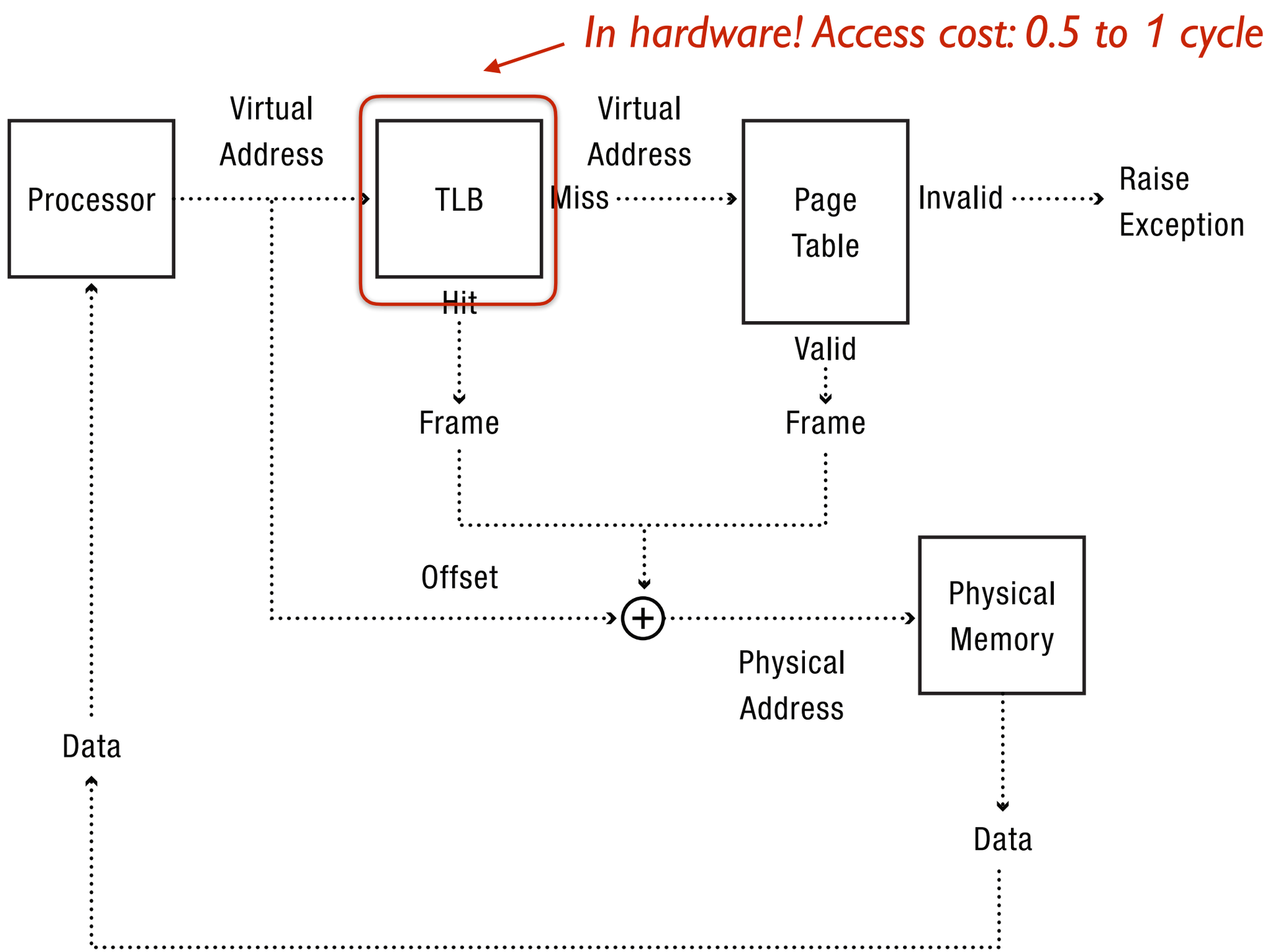
So Far…


Processor
Virtual
Address
Translation
Invalid
Raise
Exception
Valid
Physical
Memory
Physical
Address
Data
Data

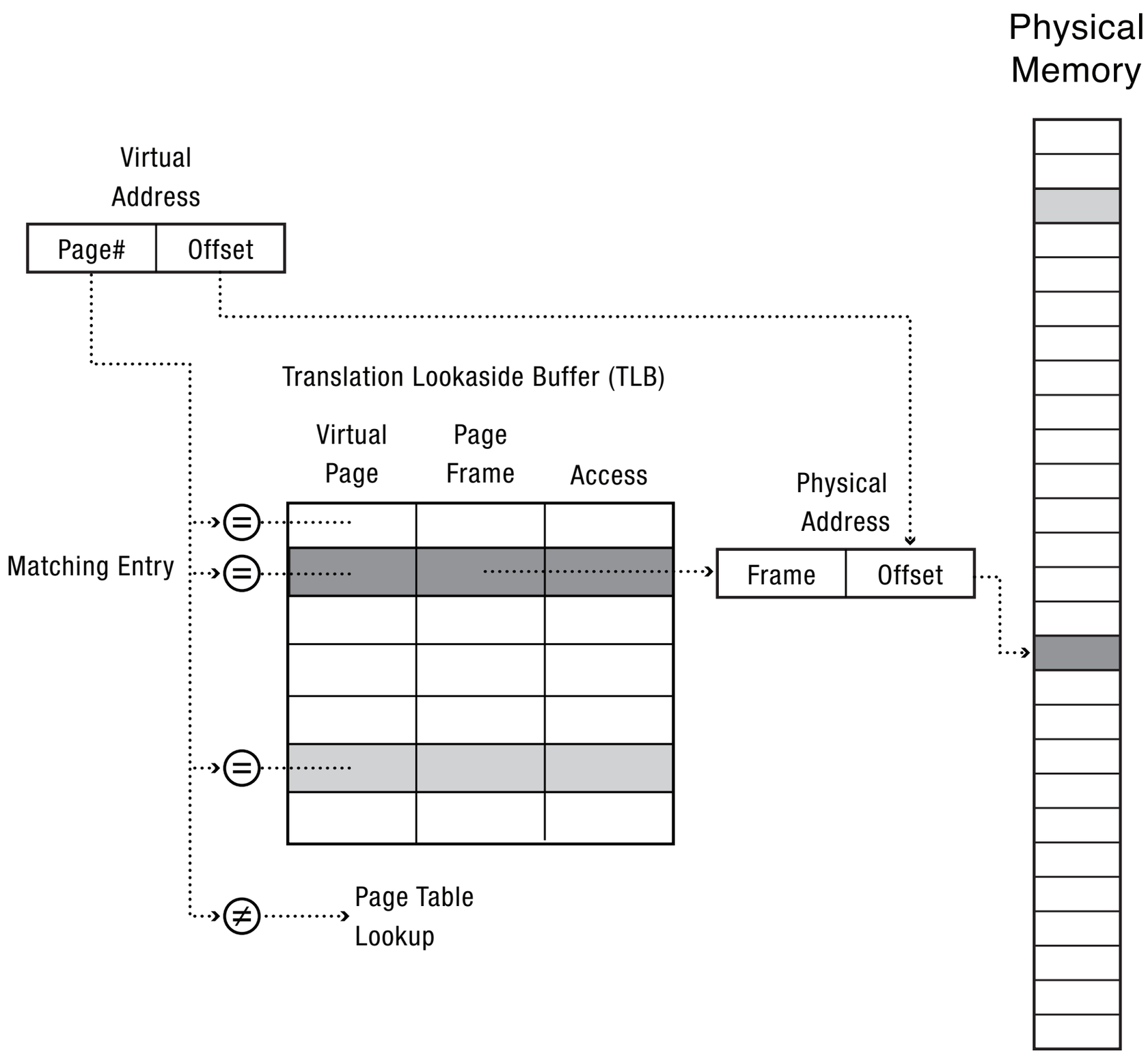
Caching: TLB


Processor
Virtual Address
TLB
Virtual Address
Miss
Page Table
Invalid
Raise Exception
Hit
Frame
Valid
Frame
Offset
Physical Memory
Physical Address
Data
Data

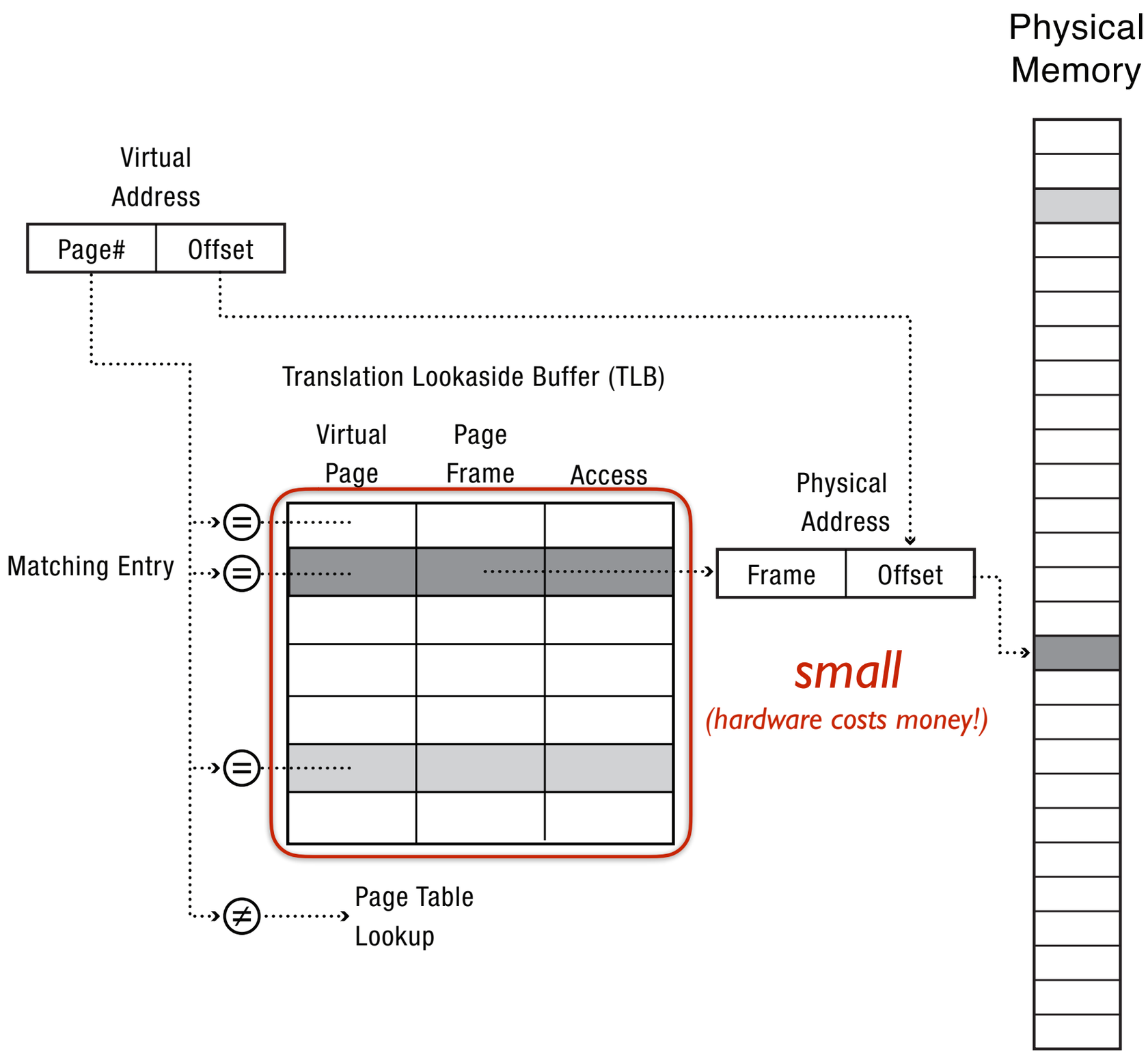
Caching: TLB


In hardware! Access cost: 0.5 to 1 cycle
Processor
Virtual Address
TLB
Hit
Virtual Address
Miss
Page Table
Invalid
Raise Exception
Valid
Frame
Frame
Offset
Physical Memory
Physical Address
Data
Data

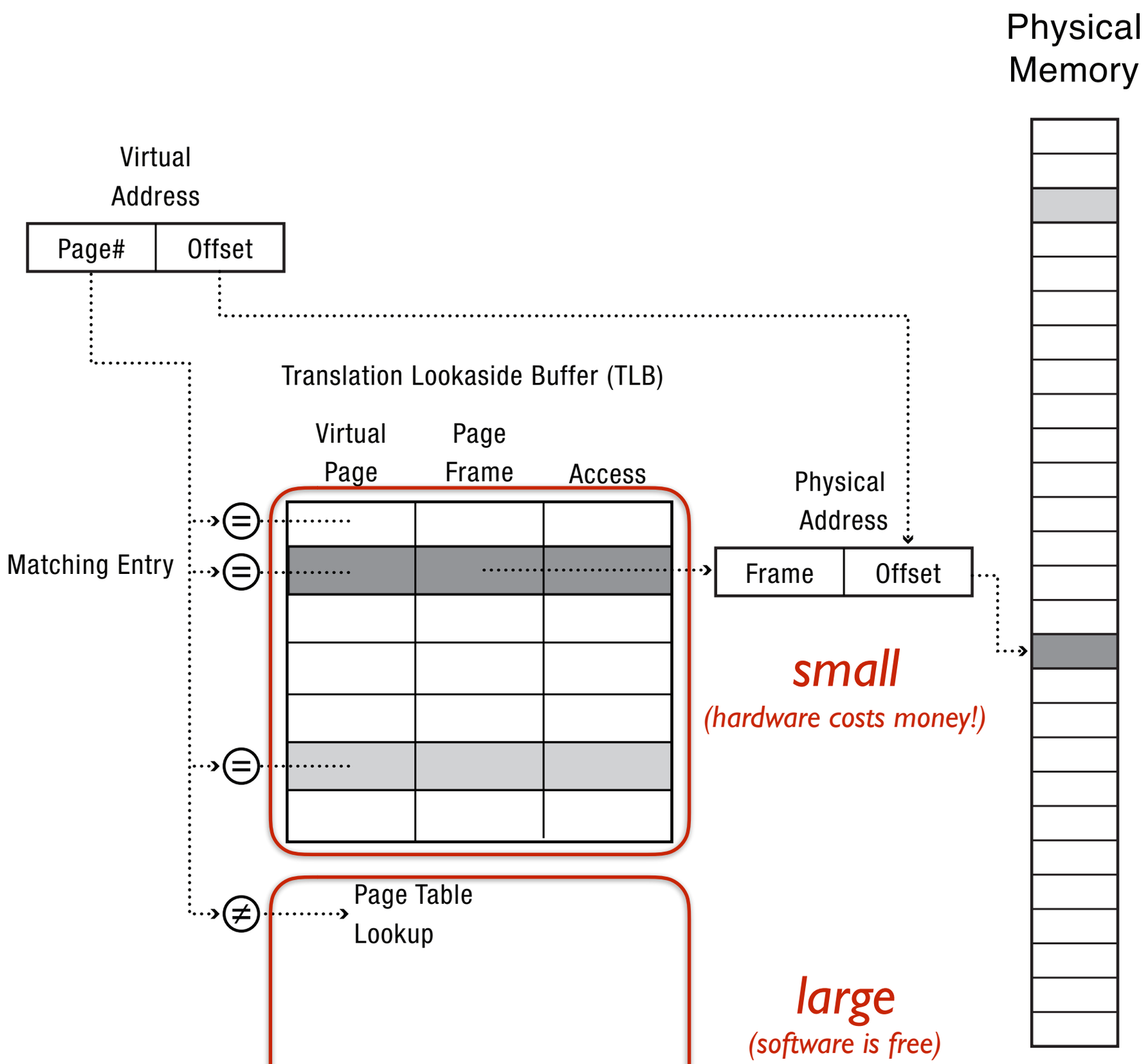
TLB Operation



TLB Operation



TLB Operation



More Details

More Details

- If the page is not in the TLB

More Details

- If the page is not in the TLB
  - x86 and ARM will look into the page tables and load it!

More Details

- If the page is not in the TLB
  - x86 and ARM will look into the page tables and load it!
    - *Hardware-loading* of the TLB

More Details

- If the page is not in the TLB
  - x86 and ARM will look into the page tables and load it!
    - *Hardware-loading* of the TLB (x86: "thank you, CR3")

More Details

- If the page is not in the TLB
  - x86 and ARM will look into the page tables and load it!
    - *Hardware-loading* of the TLB (x86: "thank you, CR3")
  - MIPS will generate an OS trap

More Details

- If the page is not in the TLB
  - x86 and ARM will look into the page tables and load it!
    - *Hardware-loading* of the TLB (x86: "thank you, CR3")
  - MIPS will generate an OS trap
    - OS handler responsible for *software-loading* the TLB

More Details

- If the page is not in the TLB
  - x86 and ARM will look into the page tables and load it!
    - *Hardware-loading* of the TLB (x86: "thank you, CR3")
  - MIPS will generate an OS trap
    - OS handler responsible for *software-loading* the TLB
- **Q:** Which TLB entry is replaced when it is full?

More Details

- If the page is not in the TLB
  - x86 and ARM will look into the page tables and load it!
    - *Hardware-loading* of the TLB (x86: "thank you, CR3")
  - MIPS will generate an OS trap
    - OS handler responsible for *software-loading* the TLB
- **Q:** Which TLB entry is replaced when it is full?
  - *Replacement policy*

More Details

- If the page is not in the TLB
  - x86 and ARM will look into the page tables and load it!
    - *Hardware-loading* of the TLB (x86: "thank you, CR3")
  - MIPS will generate an OS trap
    - OS handler responsible for *software-loading* the TLB
- **Q:** Which TLB entry is replaced when it is full?
  - *Replacement policy*
  - LRU — least recently used (note the practical implementation considerations)

More Details

More Details

- Typical size on desktops:
  - Low hundreds of entries

More Details

- Typical size on desktops:
  - Low hundreds of entries
- **Tricky Q**: How does hardware enforces LRU?
  - It doesn't. Usually hardware implements NRU
  - *Not recently used:* set a timestamp for each entry and retire old entries

More Details

More Details

- **Q:** What happens when context-switch?

More Details

- **Q:** What happens when context-switch?
  - Bunch of irrelevant entries in the TLB

More Details

- **Q:** What happens when context-switch?
  - Bunch of irrelevant entries in the TLB
- **Q:** How do we fix it?

More Details

- **Q:** What happens when context-switch?
  - Bunch of irrelevant entries in the TLB
- **Q:** How do we fix it?
  - Flush the TLB

More Details

- **Q:** What happens when context-switch?
  - Bunch of irrelevant entries in the TLB
- **Q:** How do we fix it?
  - Flush the TLB
- **Q:** What if a page has permissions changed?

More Details

- **Q:** What happens when context-switch?
  - Bunch of irrelevant entries in the TLB
- **Q:** How do we fix it?
  - Flush the TLB
- **Q:** What if a page has permissions changed?
  - Make sure that the entry is not in the TLB

More Details

- **Q:** What happens when context-switch?
  - Bunch of irrelevant entries in the TLB
- **Q:** How do we fix it?
  - Flush the TLB
- **Q:** What if a page has permissions changed?
  - Make sure that the entry is not in the TLB
  - Or fix it

More Details

More Details

- In a multiprocessor, each CPU has its own TLB, and we want to keep them with same contents

More Details

- In a multiprocessor, each CPU has its own TLB, and we want to keep them with same contents
  - Q: Why?

More Details

- In a multiprocessor, each CPU has its own TLB, and we want to keep them with same contents
  - Q: Why?
    - Process migration

More Details

- In a multiprocessor, each CPU has its own TLB, and we want to keep them with same contents
  - Q: Why?
    - Process migration
  - Q: How?

More Details

- In a multiprocessor, each CPU has its own TLB, and we want to keep them with same contents
  - **Q:** Why?
    - Process migration
  - **Q:** How?
    - Hardware protocol — *TLB shootdown*:

More Details

- In a multiprocessor, each CPU has its own TLB, and we want to keep them with same contents
  - **Q:** Why?
    - Process migration
  - **Q:** How?
    - Hardware protocol — *TLB shootdown*:
      - Stops the CPU, requests that other CPUs change or invalidate their entries, restarts the CPU

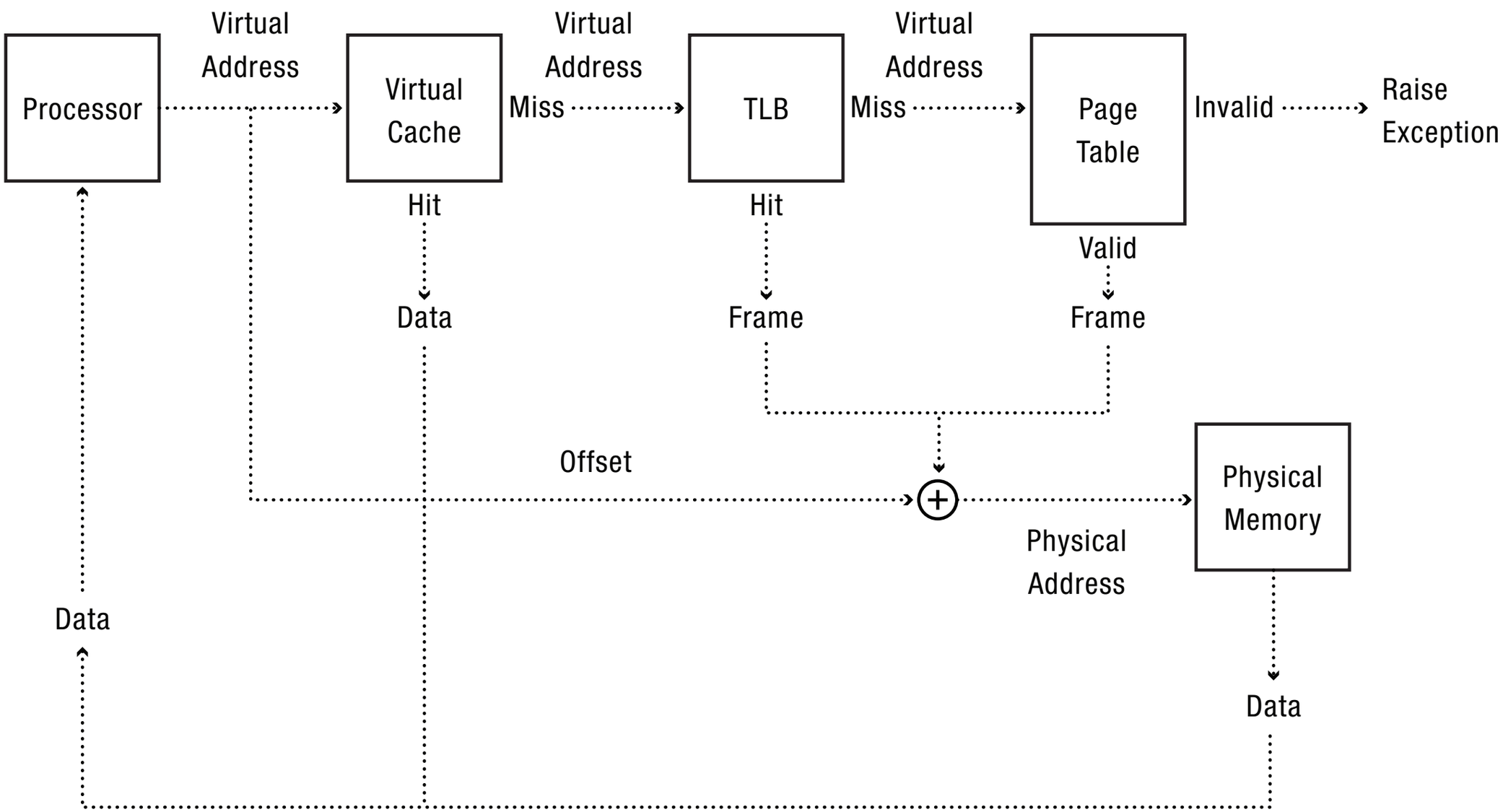
More Details

- In a multiprocessor, each CPU has its own TLB, and we want to keep them with same contents
  - **Q:** Why?
    - Process migration
  - **Q:** How?
    - Hardware protocol — *TLB shootdown*:
      - Stops the CPU, requests that other CPUs change or invalidate their entries, restarts the CPU
    - Injury: it's slow

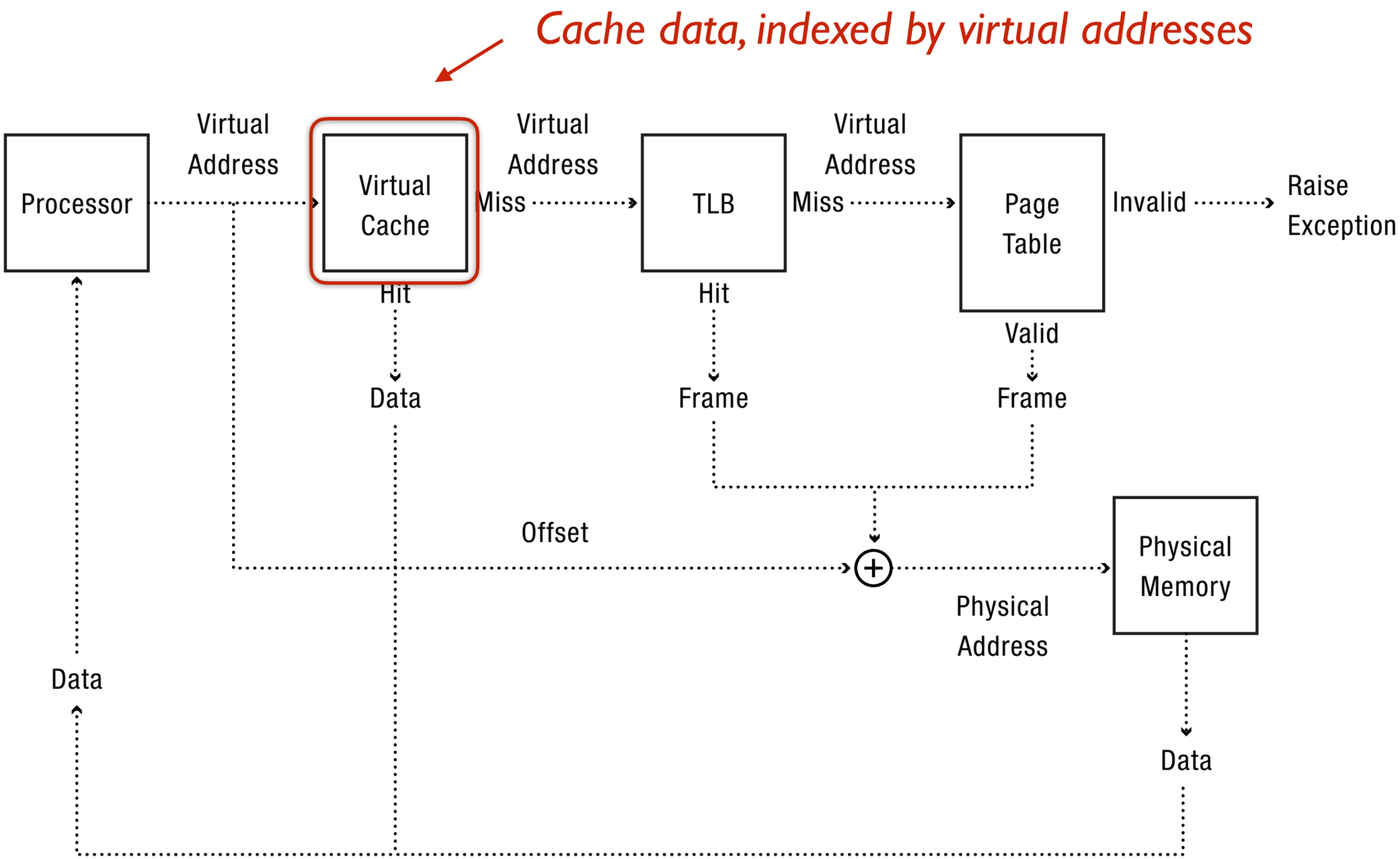
More Details

- In a multiprocessor, each CPU has its own TLB, and we want to keep them with same contents
  - **Q:** Why?
    - Process migration
  - **Q:** How?
    - Hardware protocol — *TLB shootdown*:
      - Stops the CPU, requests that other CPUs change or invalidate their entries, restarts the CPU
    - Injury: it's slow
    - Insult: the more the #CPUs, the worse the problem

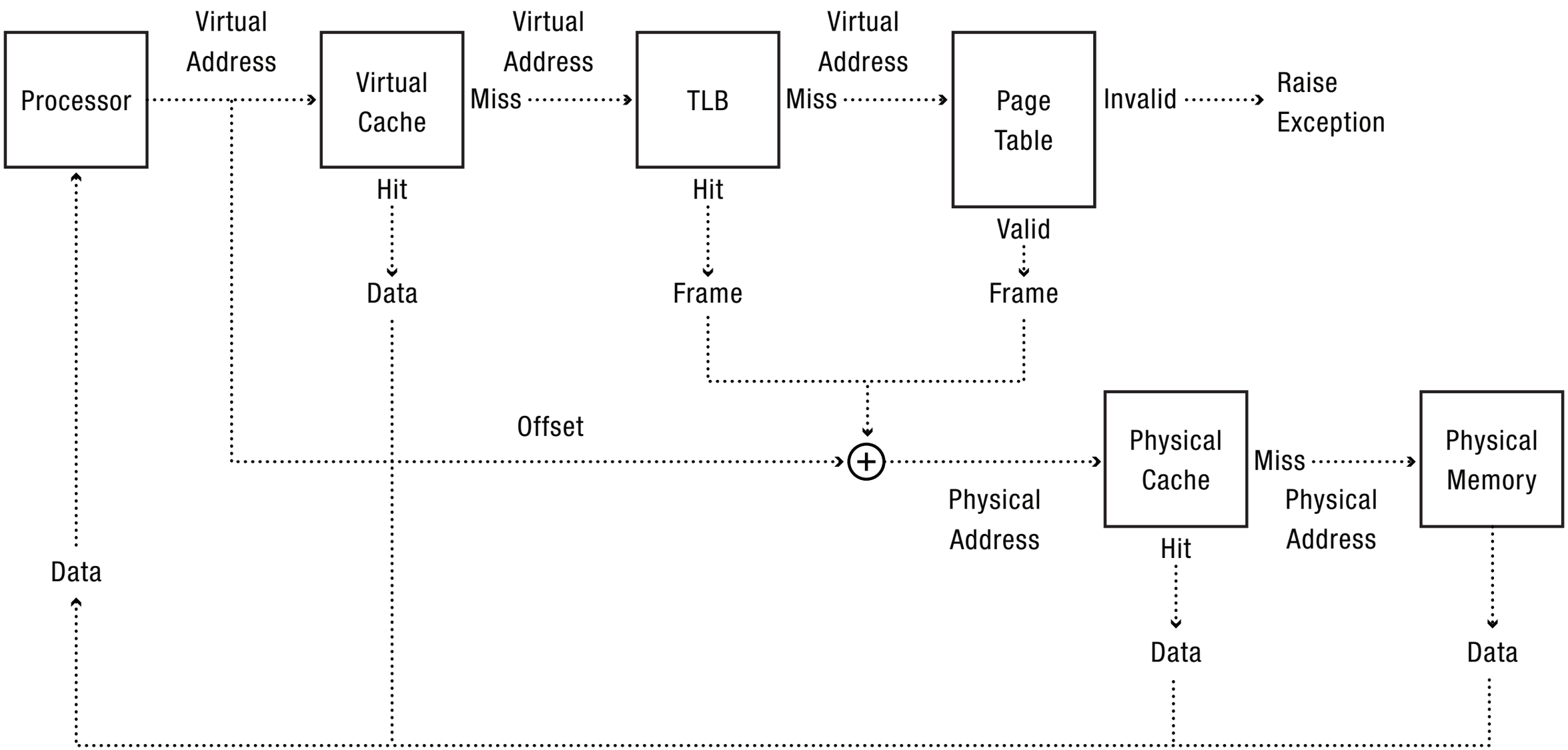
Virtually-Addressed Caches



Virtually-Addressed Caches

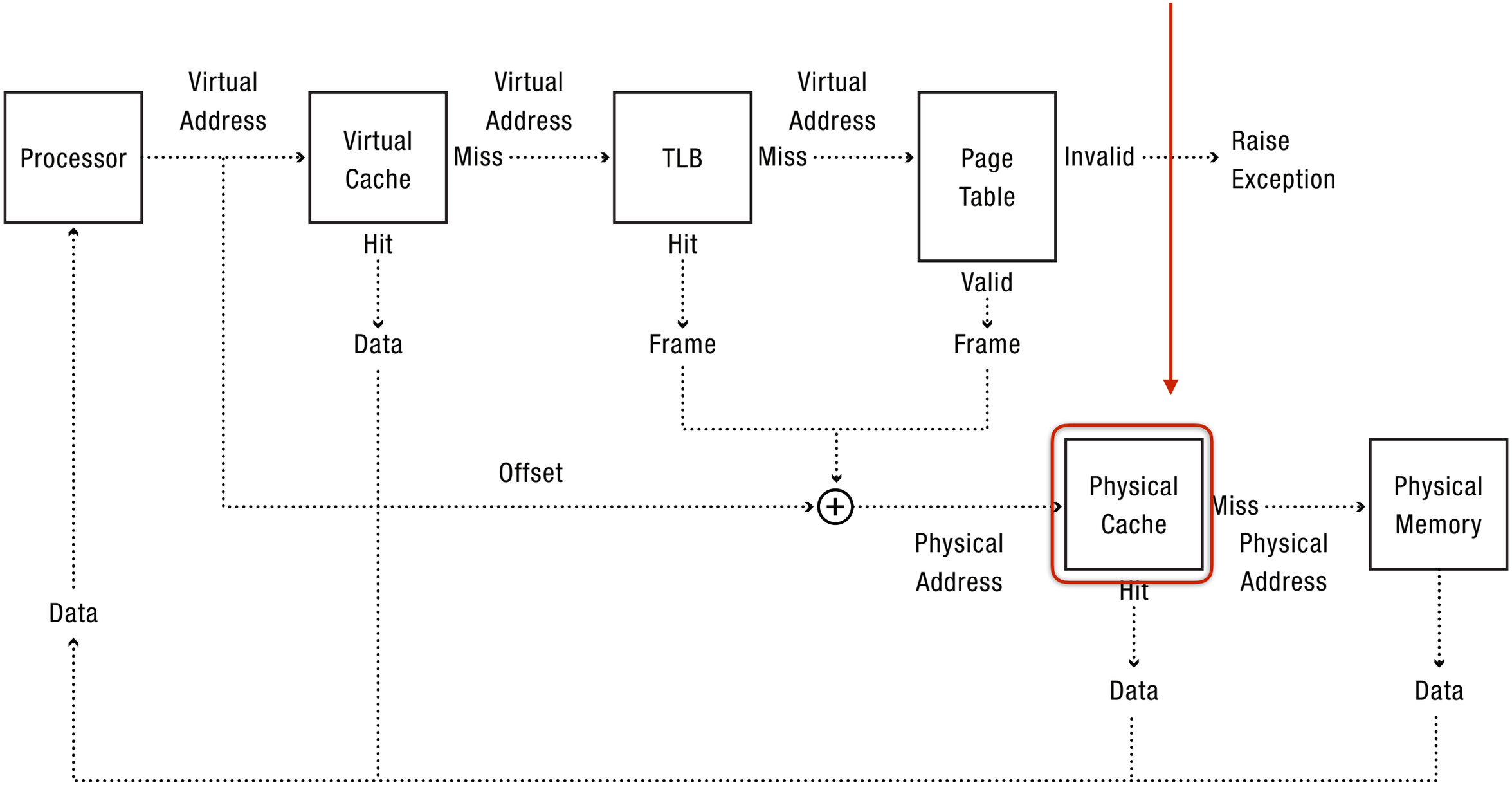


Physically-Addressed Caches



Physically-Addressed Caches

Trick: pin the most used page tables here!
Translation done on-chip even with TLB misses!



Big Question:
How efficient are caches?

Big Question:
How efficient are caches?

They are *incredibly* efficient

Effective Access Time

Effective Access Time

- Consider a *65ns* memory access time

Effective Access Time

- Consider a *65ns* memory access time
- Consider a *1ns* TLB lookup time

Effective Access Time

- Consider a *65ns* memory access time
- Consider a *1ns* TLB lookup time

Effective Access Time

- Consider a *65ns* memory access time
- Consider a *1ns* TLB lookup time

- With a TLB hit ratio of 80% (very low):
  - $(65+1)$ x $0.8 + (4*65 + 1)$ x $0.2 = 105\text{ns}$

Effective Access Time

- Consider a *65ns* memory access time
- Consider a *1ns* TLB lookup time

- With a TLB hit ratio of 80% (very low):
  - $(65+1)$ x $0.8 + (4*65 + 1)$ x $0.2 = 105\text{ns}$

- With a TLB hit ratio of 98% (typical):
  - $(65+1)$ x $0.98 + (4*65 + 1)$ x $0.02 = 69.9\text{ns}$

Fim

*all images belong to their copyright owners, including
the Computer History Museum, Wikipedia/Youtube users, or Recursive Books