DAVIDSON

Address Translation
Introduction

Hammurabi Mendes
Fall 18

Paging

Paging

- Both *virtual* and *physical* address spaces are divided in "buckets"

Paging

- Both *virtual* and *physical* address spaces are divided in "buckets"
  - In the virtual space, they're called *pages*

Paging

- Both *virtual* and *physical* address spaces are divided in "buckets"
  - In the virtual space, they're called *pages*
  - In the physical space, they're called *frames*

Paging

- Both *virtual* and *physical* address spaces are divided in "buckets"
  - In the virtual space, they're called *pages*
  - In the physical space, they're called *frames*
- Pages and frames have the *same*, *fixed* size

Paging

- Both *virtual* and *physical* address spaces are divided in "buckets"
  - In the virtual space, they're called *pages*
  - In the physical space, they're called *frames*

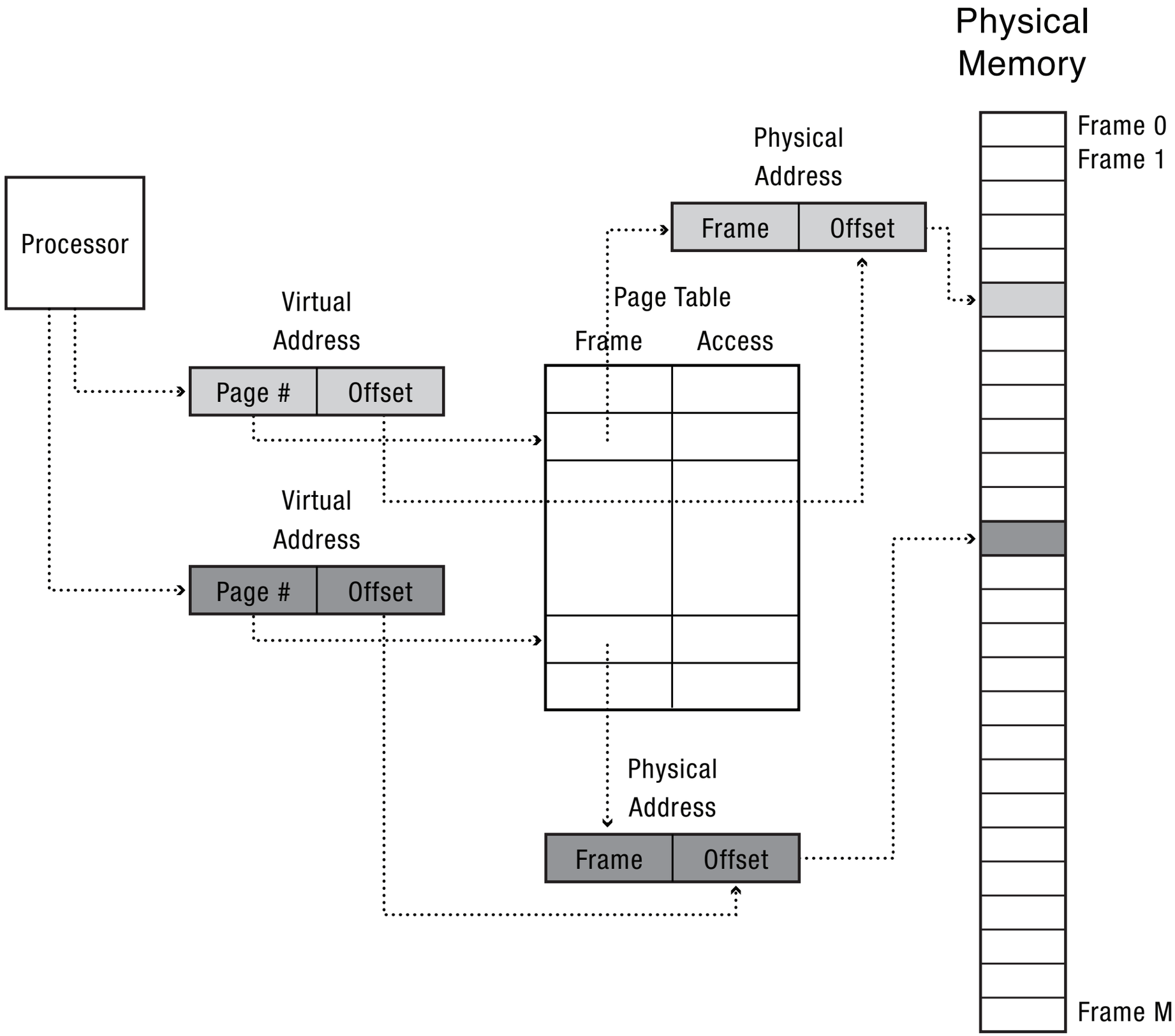
- Pages and frames have the *same*, *fixed* size
  - A power of 2: say 2^{12} = 4096 B

Paging

- Both *virtual* and *physical* address spaces are divided in "buckets"
  - In the virtual space, they're called *pages*
  - In the physical space, they're called *frames*

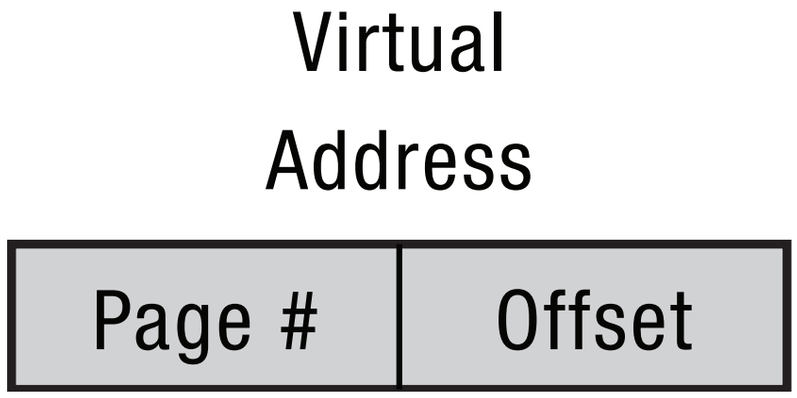
- Pages and frames have the *same*, *fixed* size
  - A power of 2: say 2^{12} = 4096 B
  - This prevents external fragmentation!

Paging


Physical
Memory
Frame 0
Frame 1
Frame M
Processor
Virtual
Address
Page #
Offset
Page Table
Frame
Access
Physical
Address
Frame
Offset
Virtual
Address
Page #
Offset
Physical
Address
Frame
Offset

Paging

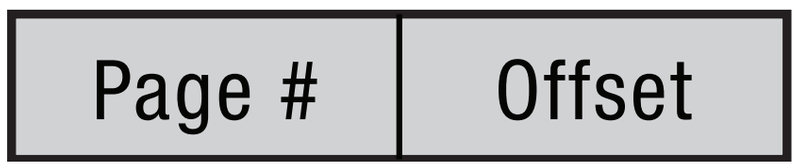
x86-64 gives you a 48-bit virtual address space:



Paging

x86-64 gives you a 48-bit virtual address space:

Virtual Address

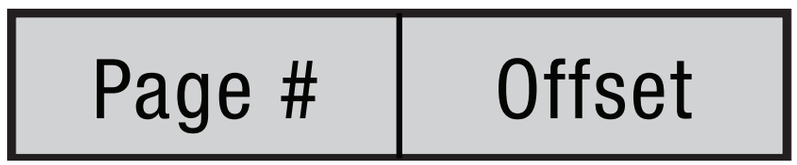


12 bits

Paging

x86-64 gives you a 48-bit virtual address space:

Virtual Address



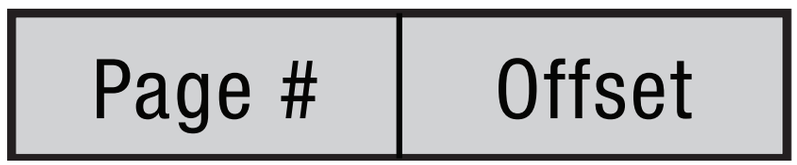
12 bits

Page size:
2^{12} = 4096 B

Paging

x86-64 gives you a 48-bit virtual address space:

Virtual Address

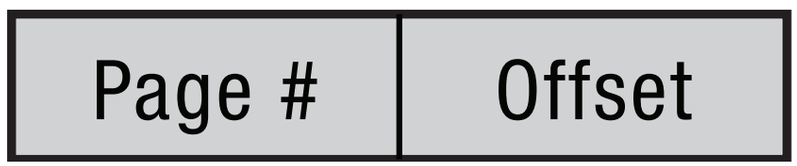


Page size:
2^{12} = 4096 B

Paging

x86-64 gives you a 48-bit virtual address space:

Virtual Address



Page size:
2^{12} = 4096 B

Maximum process image:
2^{48} = 281474976710655 B = 256 TB

Paging

Paging

- Advantages over Segmentation

Paging

- Advantages over Segmentation
  - No external fragmentation
    - Only *internal fragmentation*: last page is not fully used

Paging

- Advantages over Segmentation
  - No external fragmentation
    - Only *internal fragmentation*: last page is not fully used
  - Can load code/data on demand as the process executes

Paging

- Advantages over Segmentation
  - No external fragmentation
    - Only *internal fragmentation*: last page is not fully used
  - Can load code/data on demand as the process executes
    - When the program references a non-mapped virtual address, the hardware generates a *page fault (interrupt)*

Paging

- Advantages over Segmentation
  - No external fragmentation
    - Only *internal fragmentation*: last page is not fully used
  - Can load code/data on demand as the process executes
    - When the program references a non-mapped virtual address, the hardware generates a *page fault (interrupt)*
    - The OS loads the page and resumes the program

Paging

- Advantages over Segmentation
  - No external fragmentation
    - Only *internal fragmentation*: last page is not fully used
  - Can load code/data on demand as the process executes
    - When the program references a non-mapped virtual address, the hardware generates a *page fault (interrupt)*
    - The OS loads the page and resumes the program
  - Out of memory? Store unused pages in the disk!

Paging

Paging

- Drawbacks:

Paging

- Drawbacks:
  - Requires more space to translate address

Paging

- Drawbacks:
  - Requires more space to translate address
    - *Naively*, we need an entry for every page!

Paging

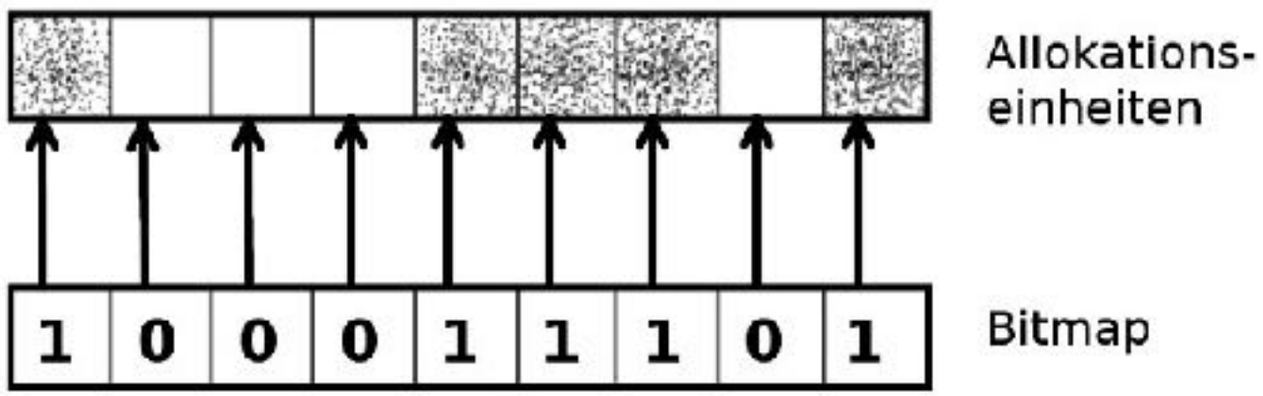
- Drawbacks:
  - Requires more space to translate address
    - *Naively*, we need an entry for every page!
    - That's 262144 entries for a 1GB space

Paging

- Drawbacks:
  - Requires more space to translate address
    - *Naively*, we need an entry for every page!
    - That's 262144 entries for a 1GB space
  - Keeping track of free/used physical frames

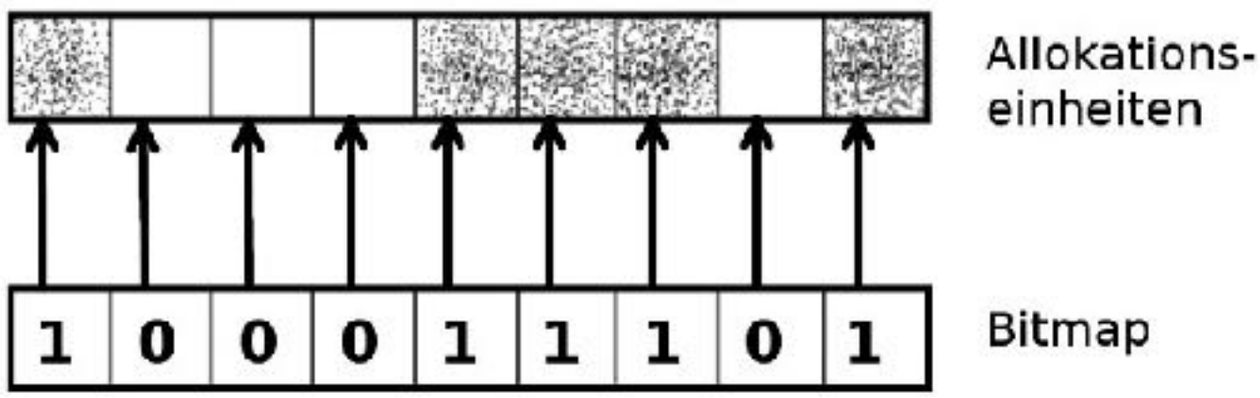
Paging

- Drawbacks:
  - Requires more space to translate address
    - *Naively*, we need an entry for every page!
    - That's 262144 entries for a 1GB space
- Keeping track of free/used physical frames
  - *Simplistically*, a bitmap solves the problem



Paging

- Drawbacks:
  - Requires more space to translate address
    - *Naively*, we need an entry for every page!
    - That's 262144 entries for a 1GB space
- Keeping track of free/used physical frames
  - *Simplistically*, a bitmap solves the problem

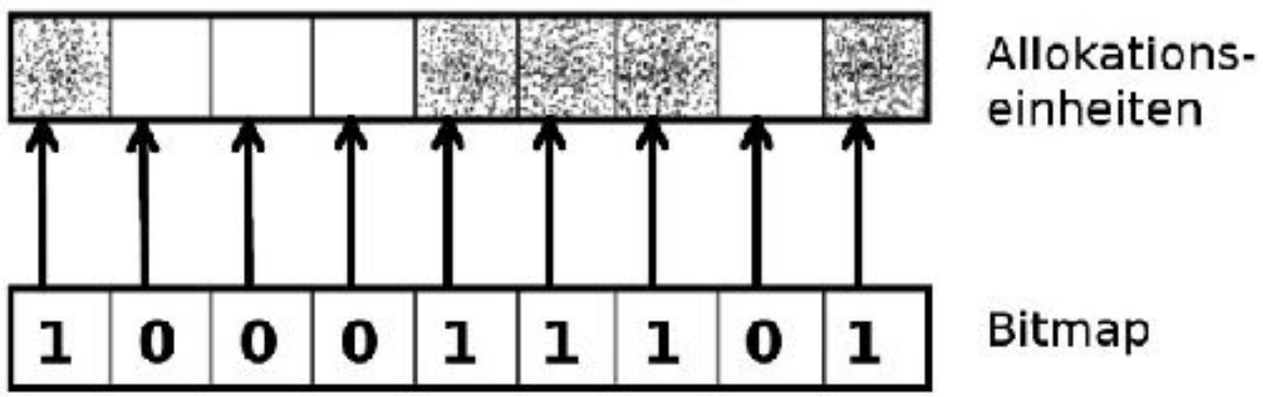


1) Thanks Wikipedia user "Didia"

Paging

- Drawbacks:
  - Requires more space to translate address
    - *Naively*, we need an entry for every page!
    - That's 262144 entries for a 1GB space

- Keeping track of free/used physical frames
  - *Simplistically*, a bitmap solves the problem

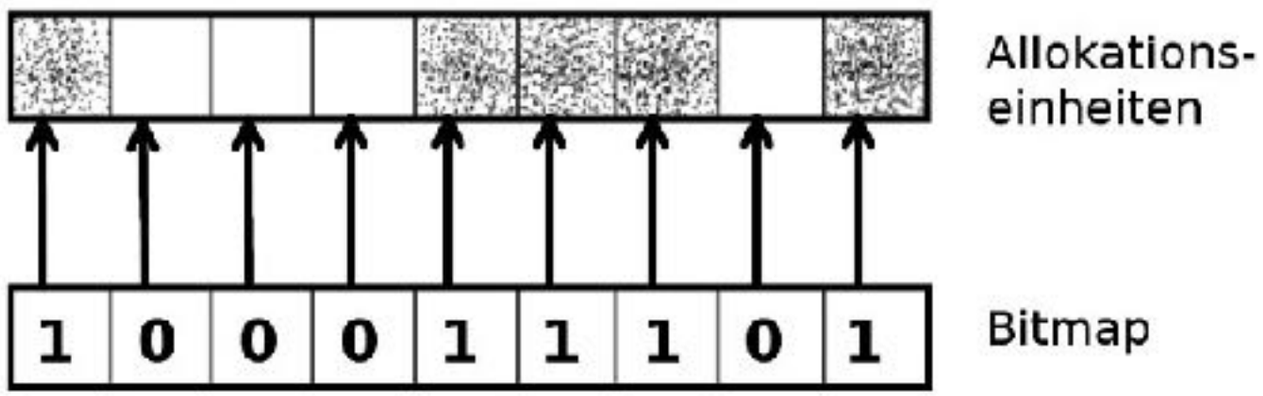


1) Thanks Wikipedia user "Didia"
2) This means "allocation units"

Paging

- Drawbacks:
  - Requires more space to translate address
    - *Naively*, we need an entry for every page!
    - That's 262144 entries for a 1GB space

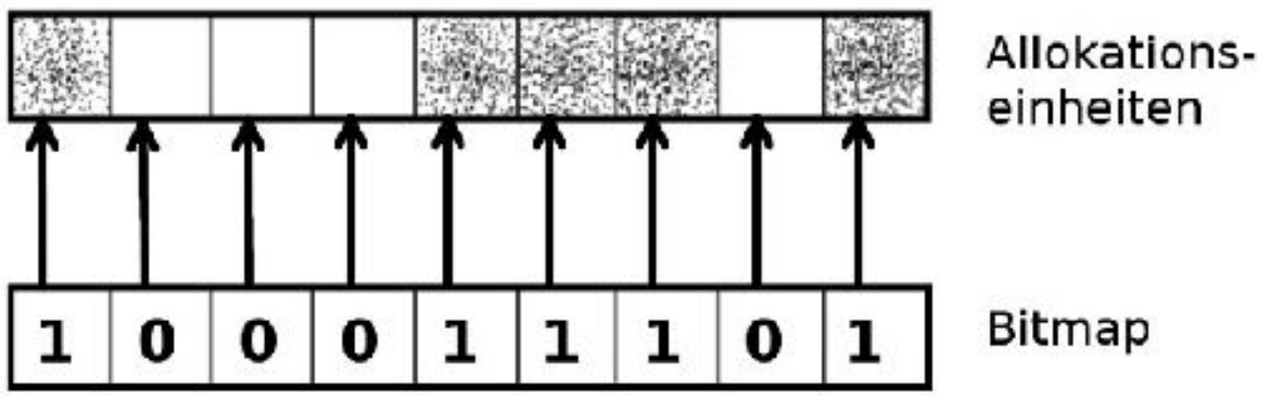
- Keeping track of free/used physical frames
  - *Simplistically*, a bitmap solves the problem



1) Thanks Wikipedia user "Didia"
2) This means "allocation units"
3) Thanks Google Translate

Paging

- Drawbacks:
  - Requires more space to translate address
    - *Naively*, we need an entry for every page!
    - That's 262144 entries for a 1GB space
- Keeping track of free/used physical frames
  - *Simplistically*, a bitmap solves the problem



Of course

we will use better data structures
to tackle both problems

Multilevel Paging

Multilevel Paging

- Never heard of a program using 256TB of memory

Multilevel Paging

- Never heard of a program using 256TB of memory
  - Page table is a sparse data structure

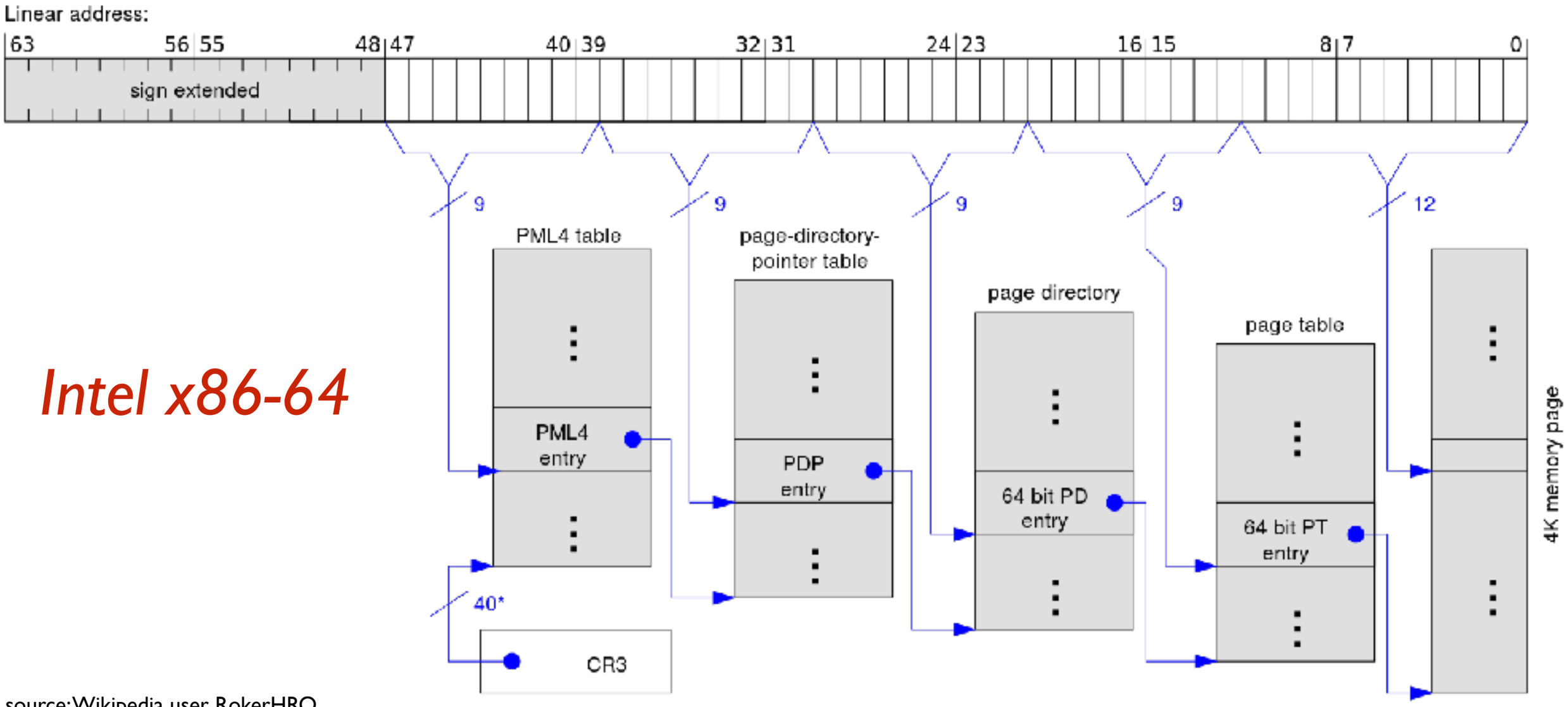
Multilevel Paging

- Never heard of a program using 256TB of memory
  - Page table is a sparse data structure
  - Should be divided, with pieces allocated on demand!

Multilevel Paging

- Never heard of a program using 256TB of memory
  - Page table is a sparse data structure
  - Should be divided, with pieces allocated on demand!

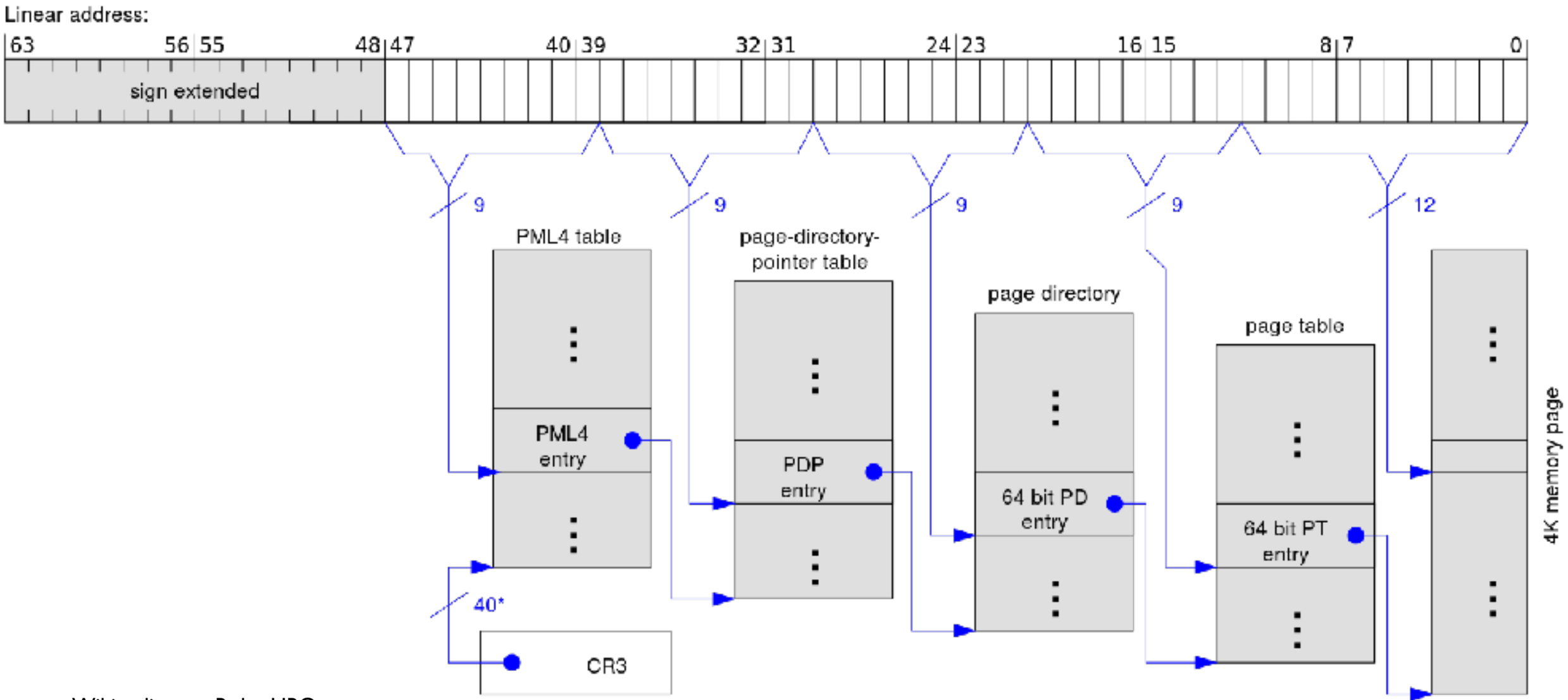
Intel x86-64



* source: Wikipedia user RokerHRO

Multilevel Paging

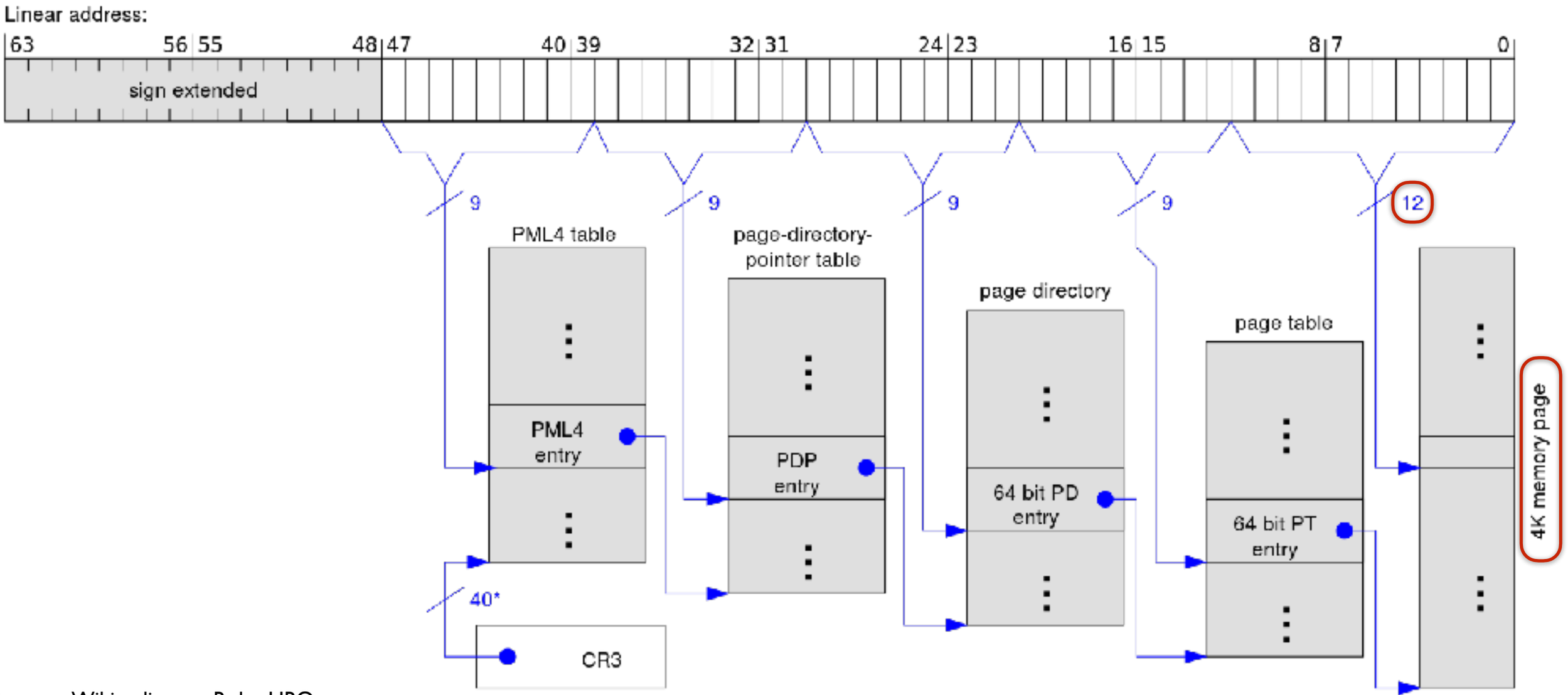
Intel x86-64



* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64

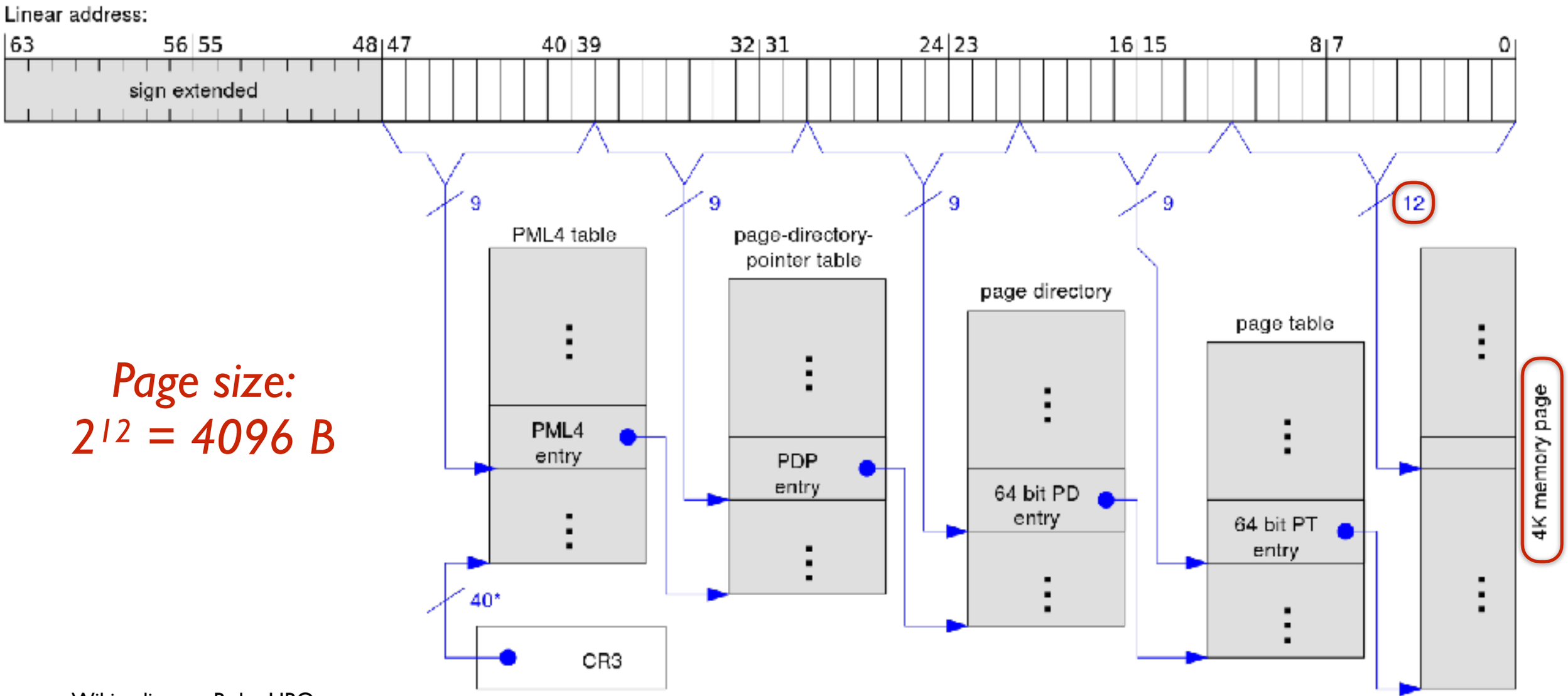


* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64

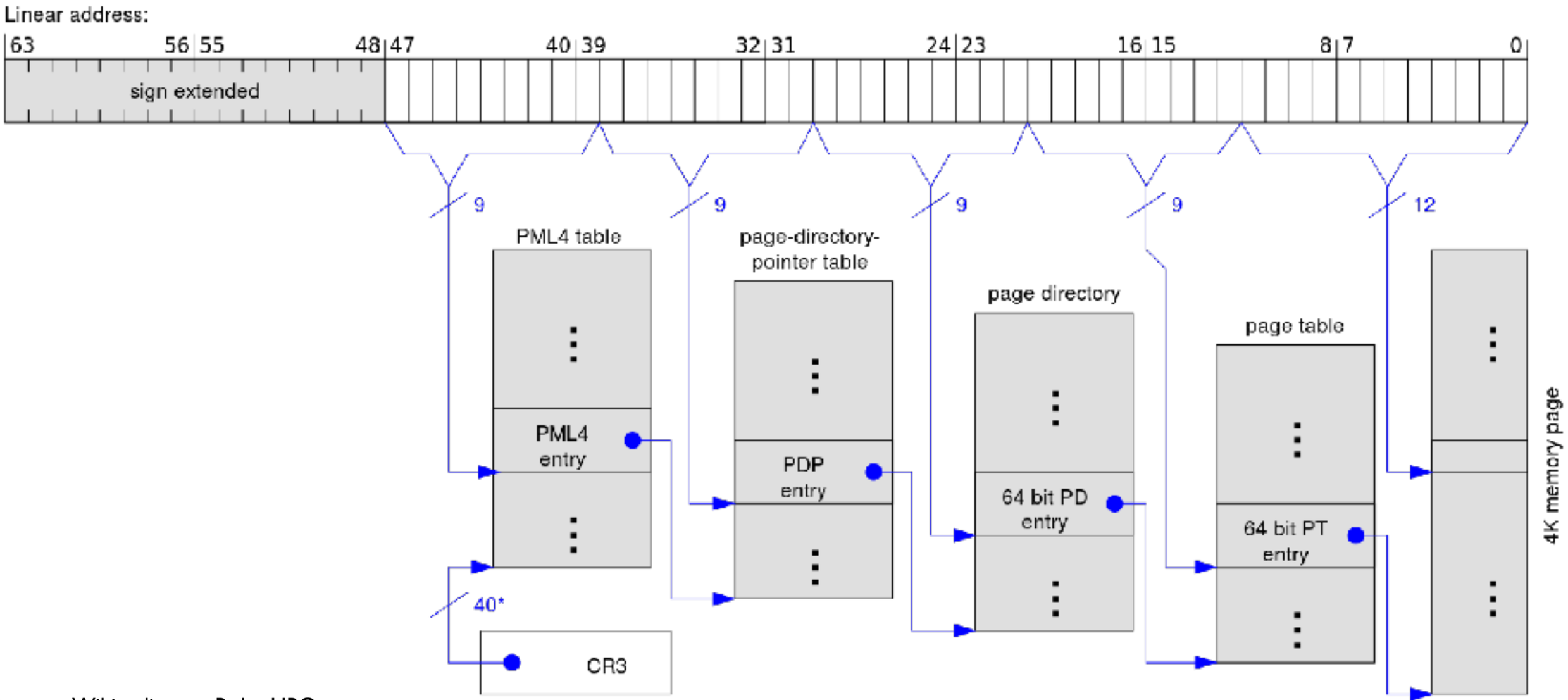
Page size:
2^{12} = 4096 B



* source: Wikipedia user RokerHRO

Multilevel Paging

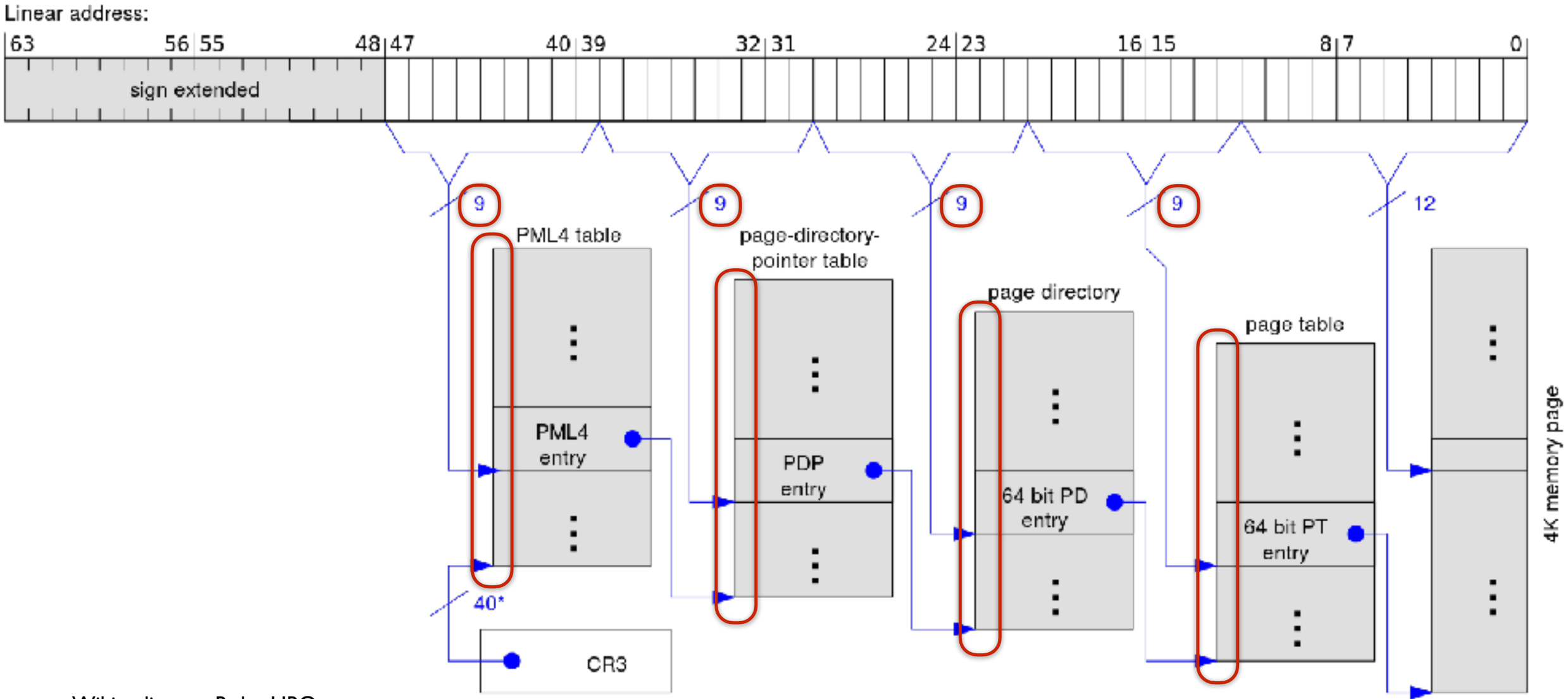
Intel x86-64



* source: Wikipedia user RokerHRO

Multilevel Paging

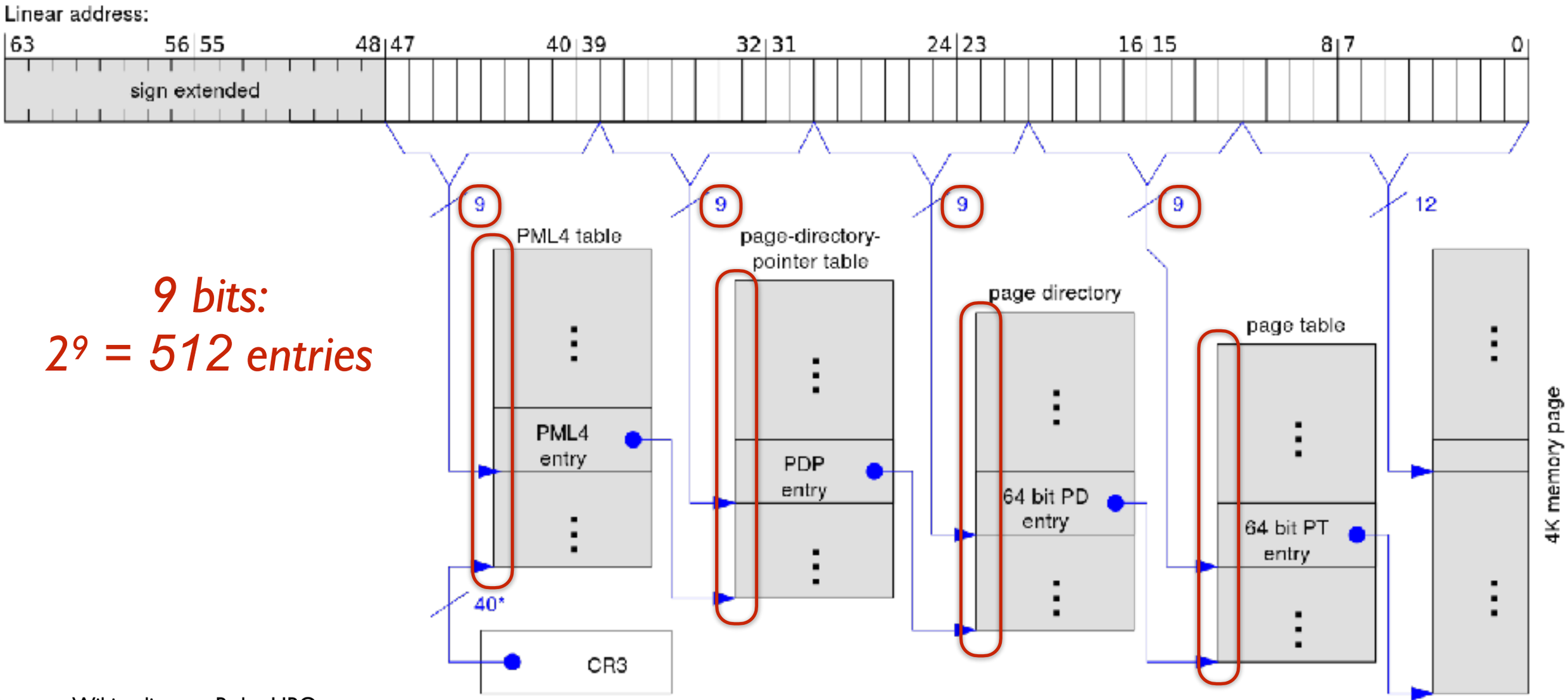
Intel x86-64



* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64

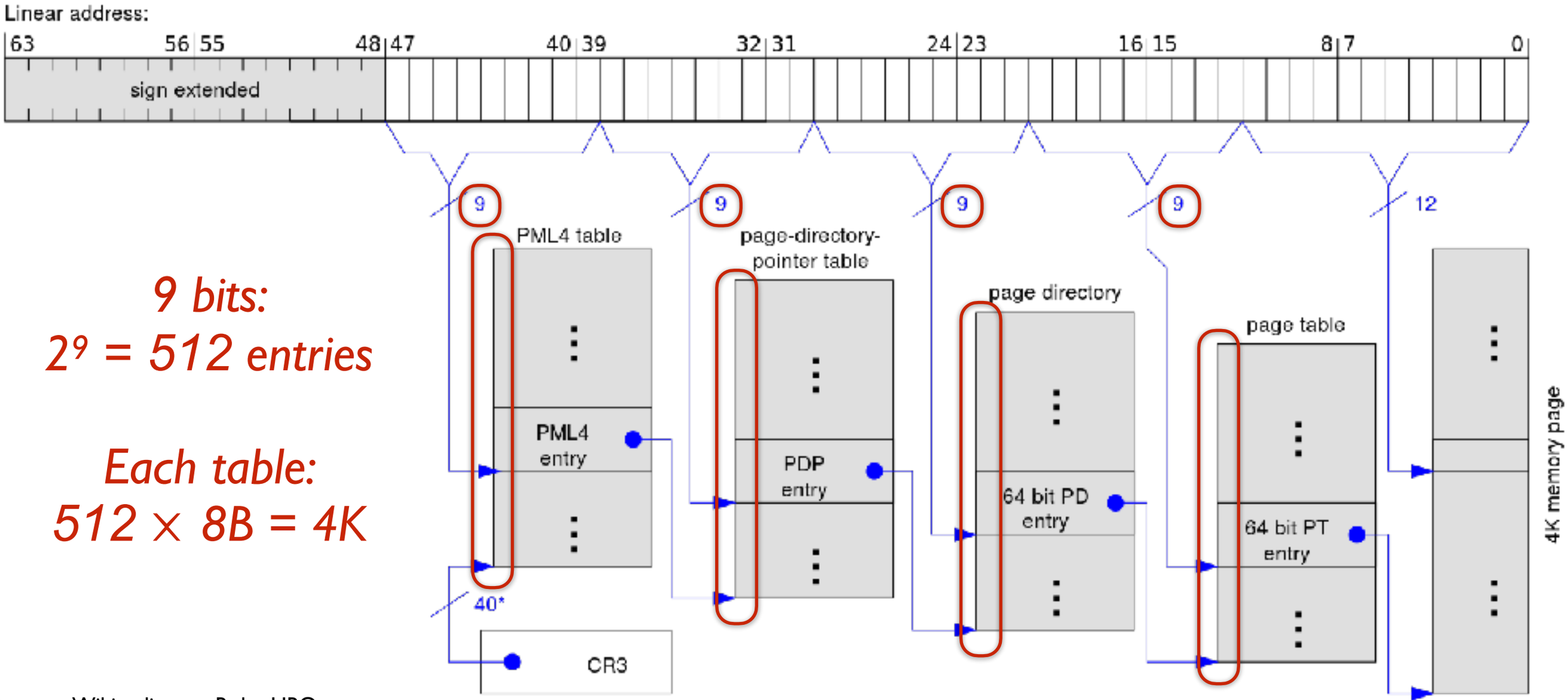


9 bits:
2^9 = 512 entries

* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64



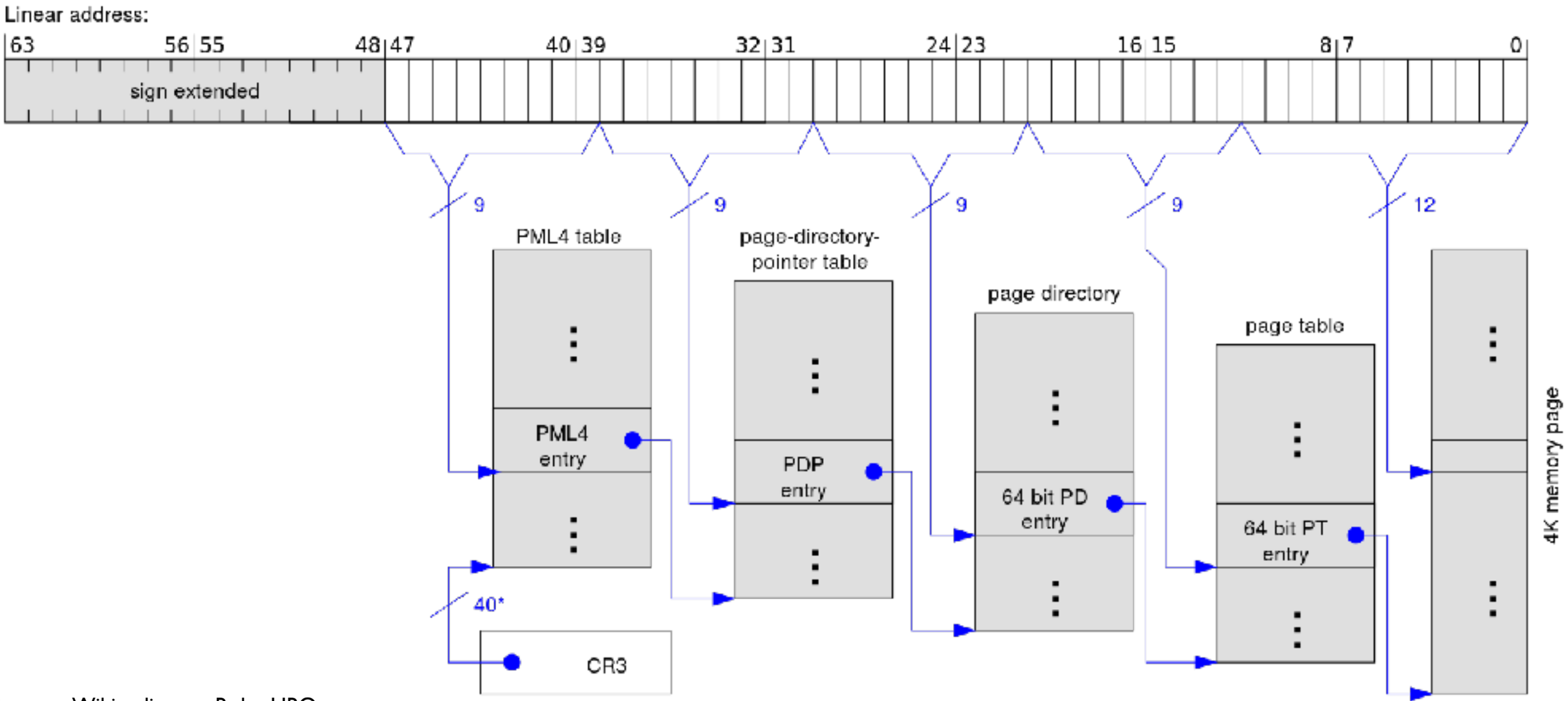
9 bits:
2^9 = 512 entries

Each table:
512 × 8B = 4K

* source: Wikipedia user RokerHRO

Multilevel Paging

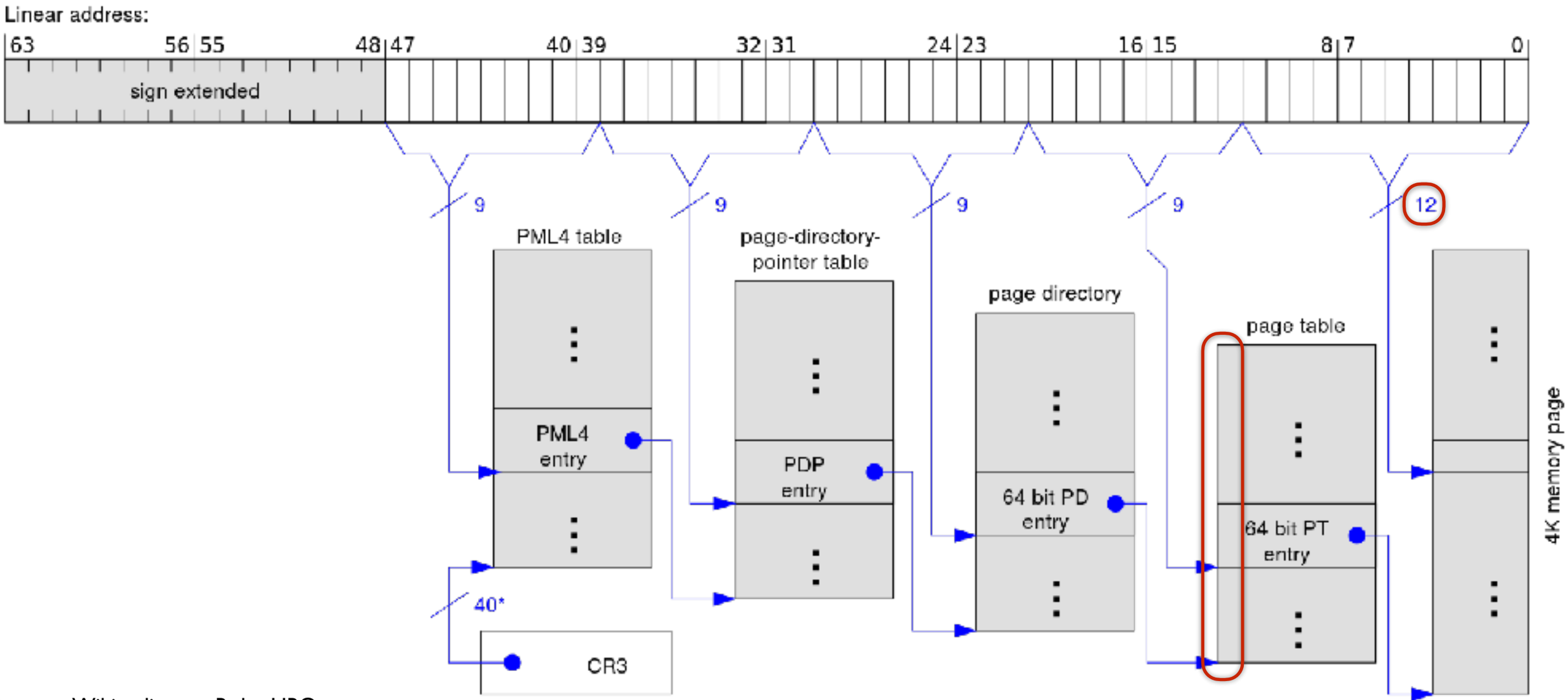
Intel x86-64



* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64

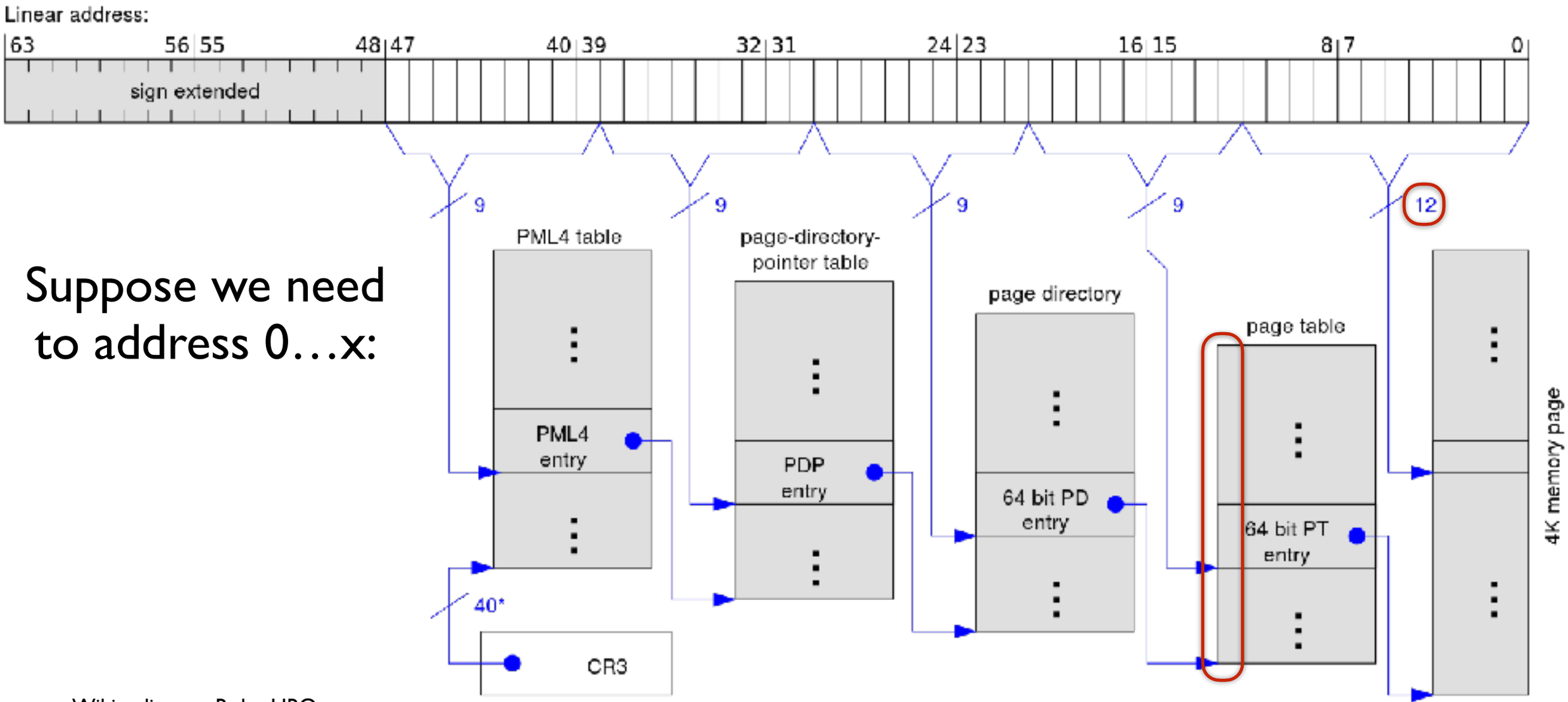


* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64

Suppose we need to address 0…x:



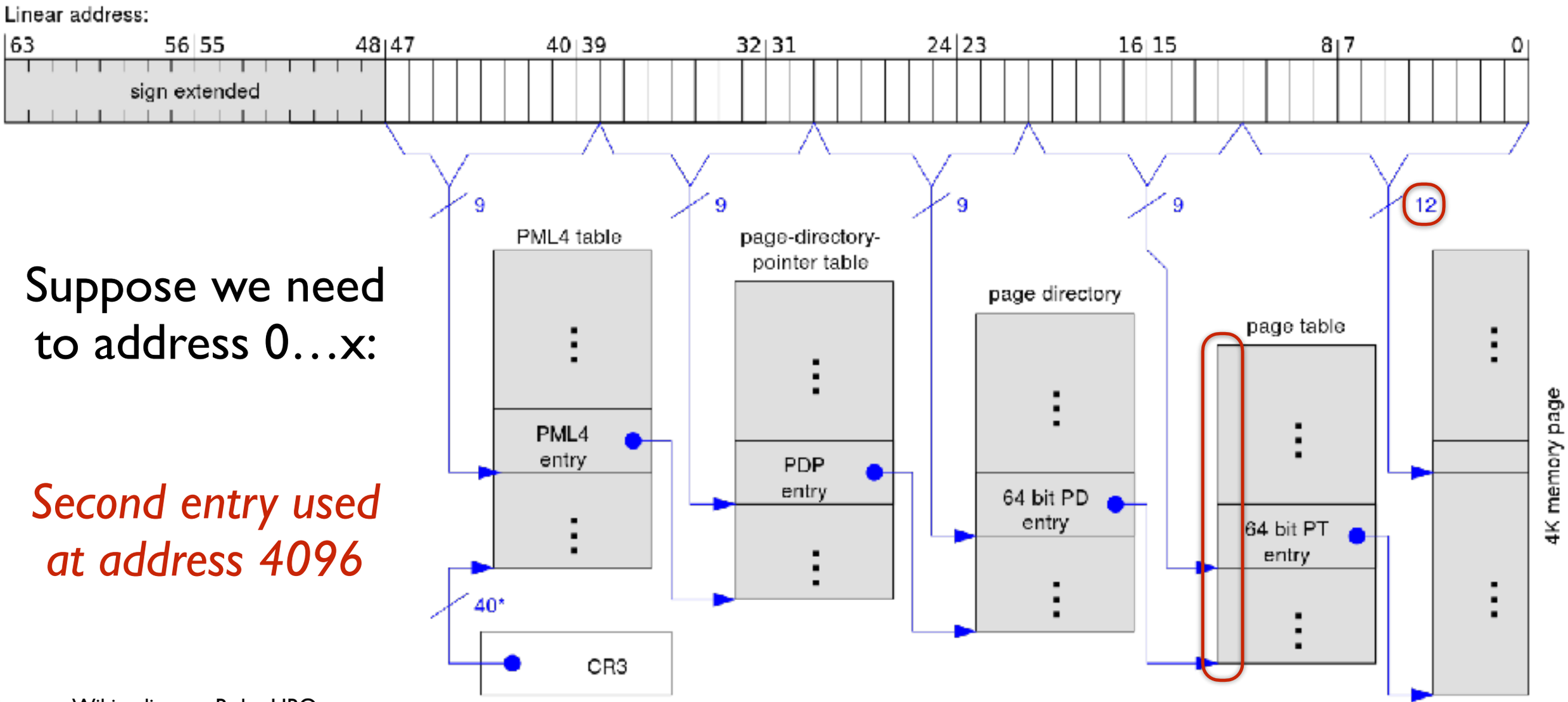
* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64

Suppose we need to address 0…x:

Second entry used at address 4096

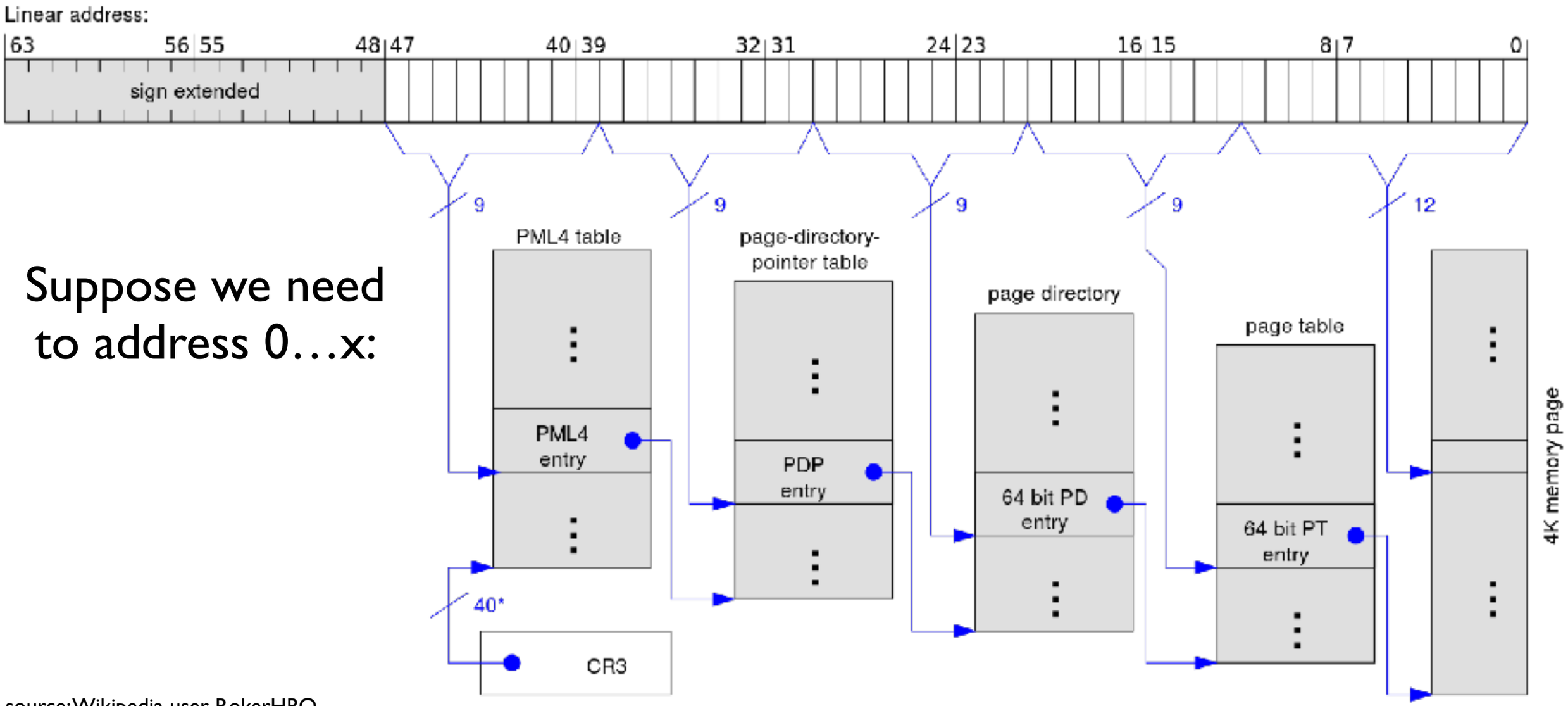


* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64

Suppose we need to address 0…x:

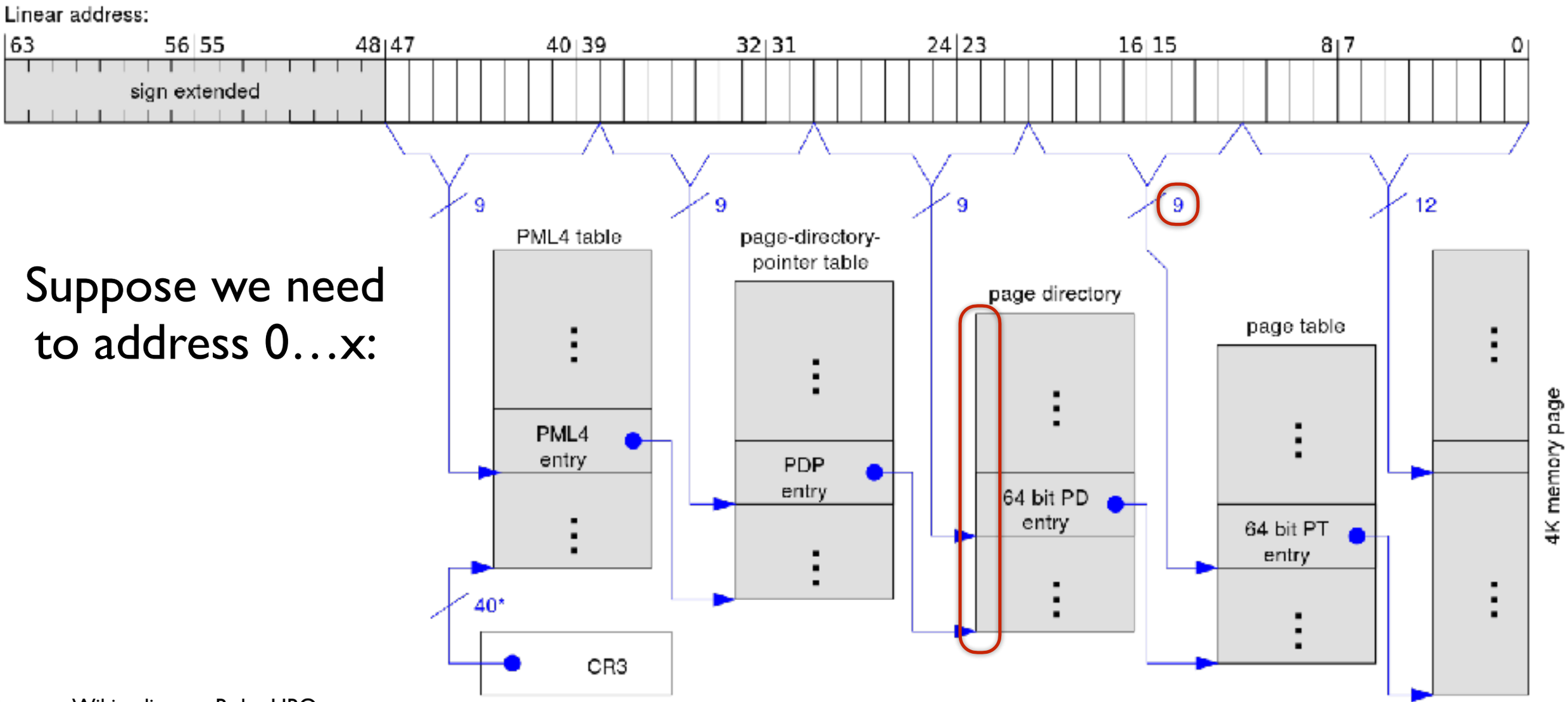


* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64

Suppose we need to address 0…x:



* source: Wikipedia user RokerHRO

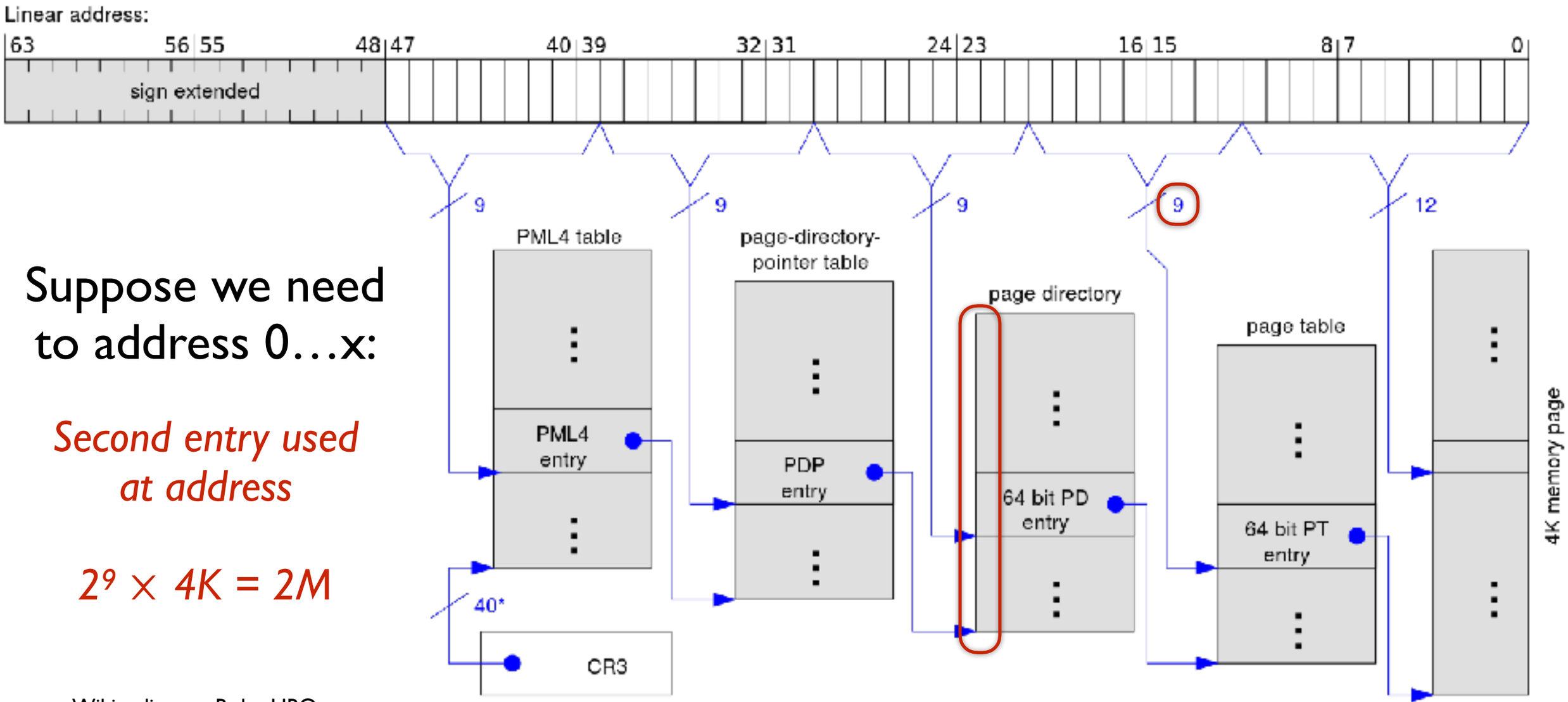
Multilevel Paging

Intel x86-64

Suppose we need to address 0…x:

Second entry used at address

$2^9 \times 4K = 2M$

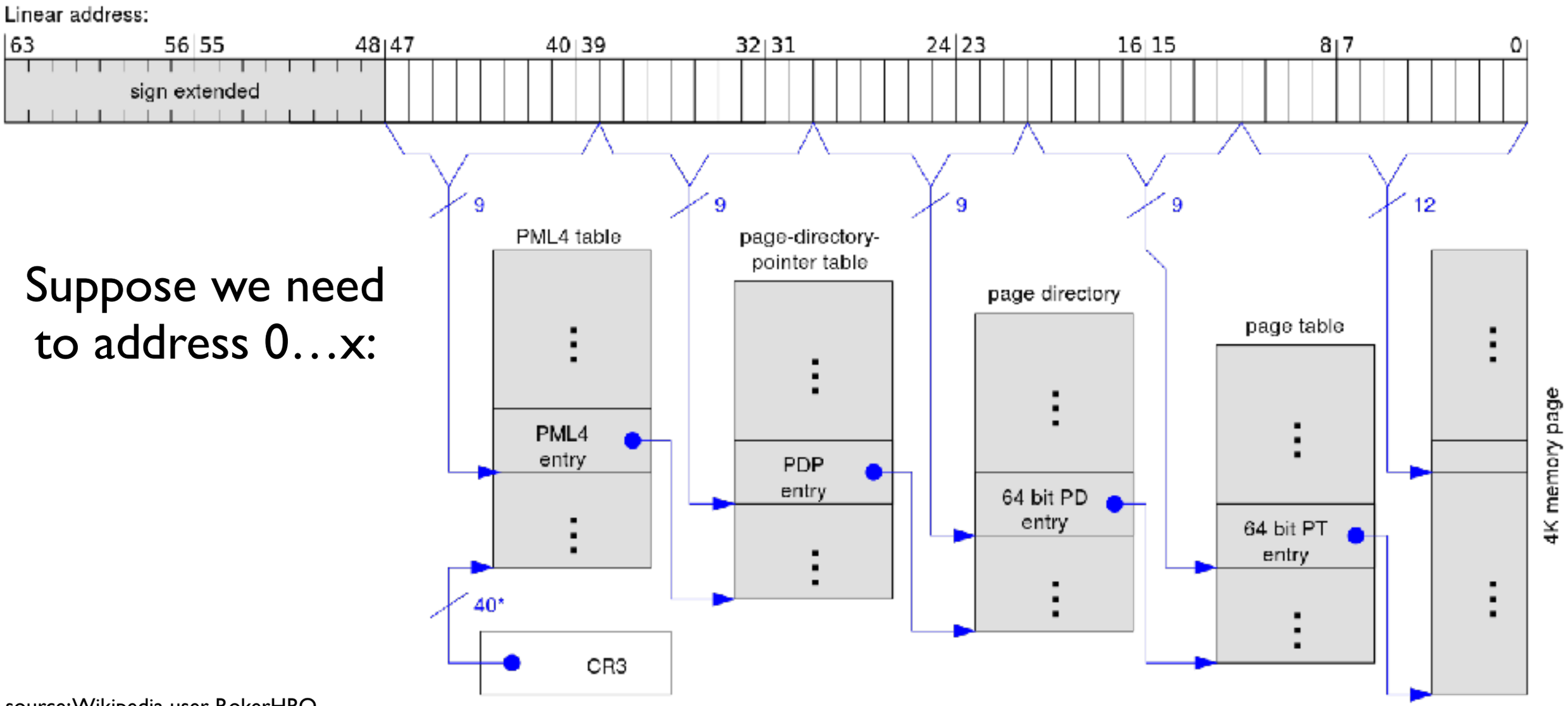


* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64

Suppose we need to address 0…x:

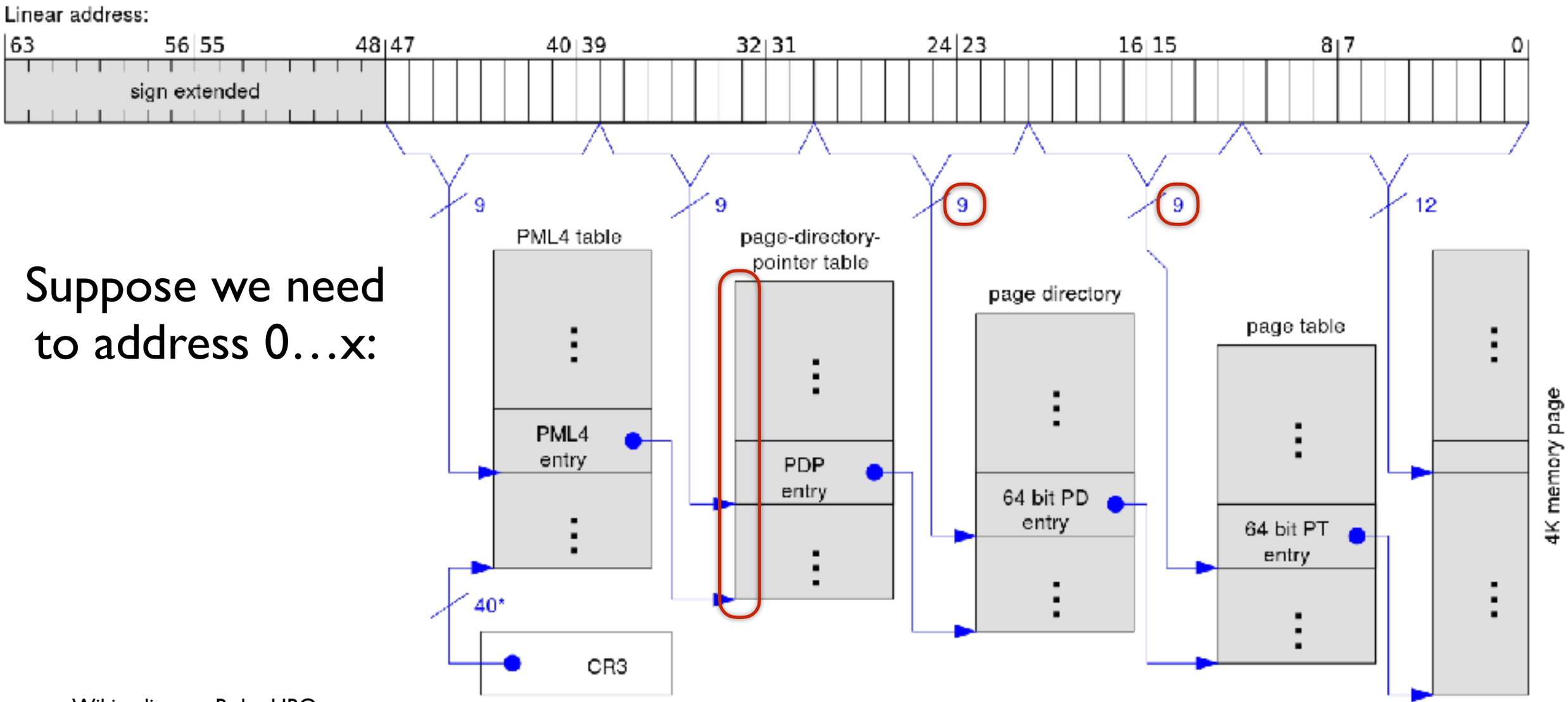


* source: Wikipedia user RokerHRO

Multilevel Paging

Intel x86-64

Suppose we need to address 0…x:



* source: Wikipedia user RokerHRO

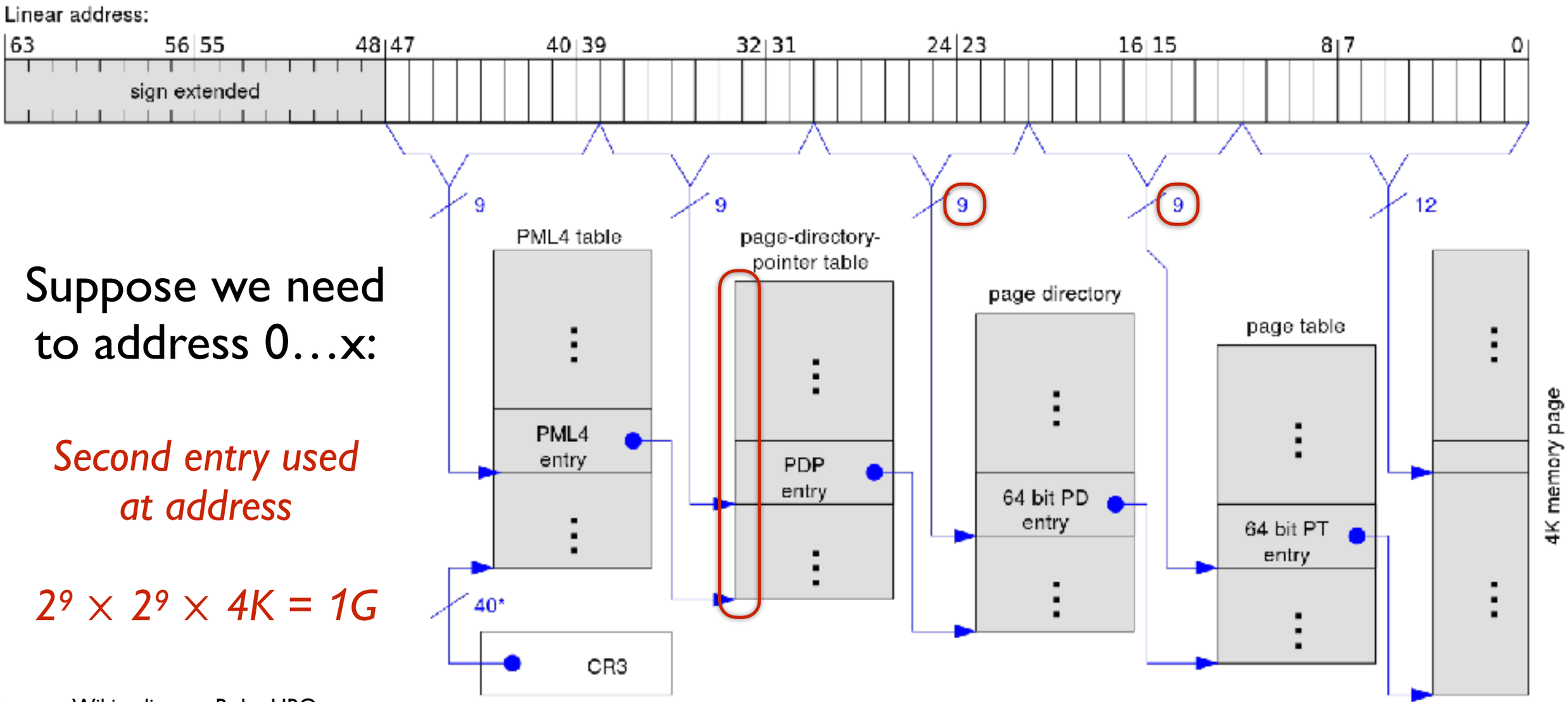
Multilevel Paging

Intel x86-64

Suppose we need to address 0…x:

Second entry used at address

$2^9 \times 2^9 \times 4K = 1G$



* source: Wikipedia user RokerHRO

Paging

Paging

- Each process has its own page-table structure

Paging

- Each process has its own page-table structure
- It's a *tree*, not a chain

Paging

- Each process has its own page-table structure
- It's a *tree*, not a chain
  - May have looked like a chain in the previous slides…

Paging

- Each process has its own page-table structure
- It's a *tree*, not a chain
  - May have looked like a chain in the previous slides…
  - But it's not! As your memory footprint increases, the more the structure looks like a tree

Paging

- Each process has its own page-table structure
- It's a *tree*, not a chain
  - May have looked like a chain in the previous slides…
  - But it's not! As your memory footprint increases, the more the structure looks like a tree
- On context-switch, change it completely

Paging

- Each process has its own page-table structure
- It's a *tree*, not a chain
  - May have looked like a chain in the previous slides…
  - But it's not! As your memory footprint increases, the more the structure looks like a tree
- On context-switch, change it completely
  - On x86-64, that means reassigning the CR3 register

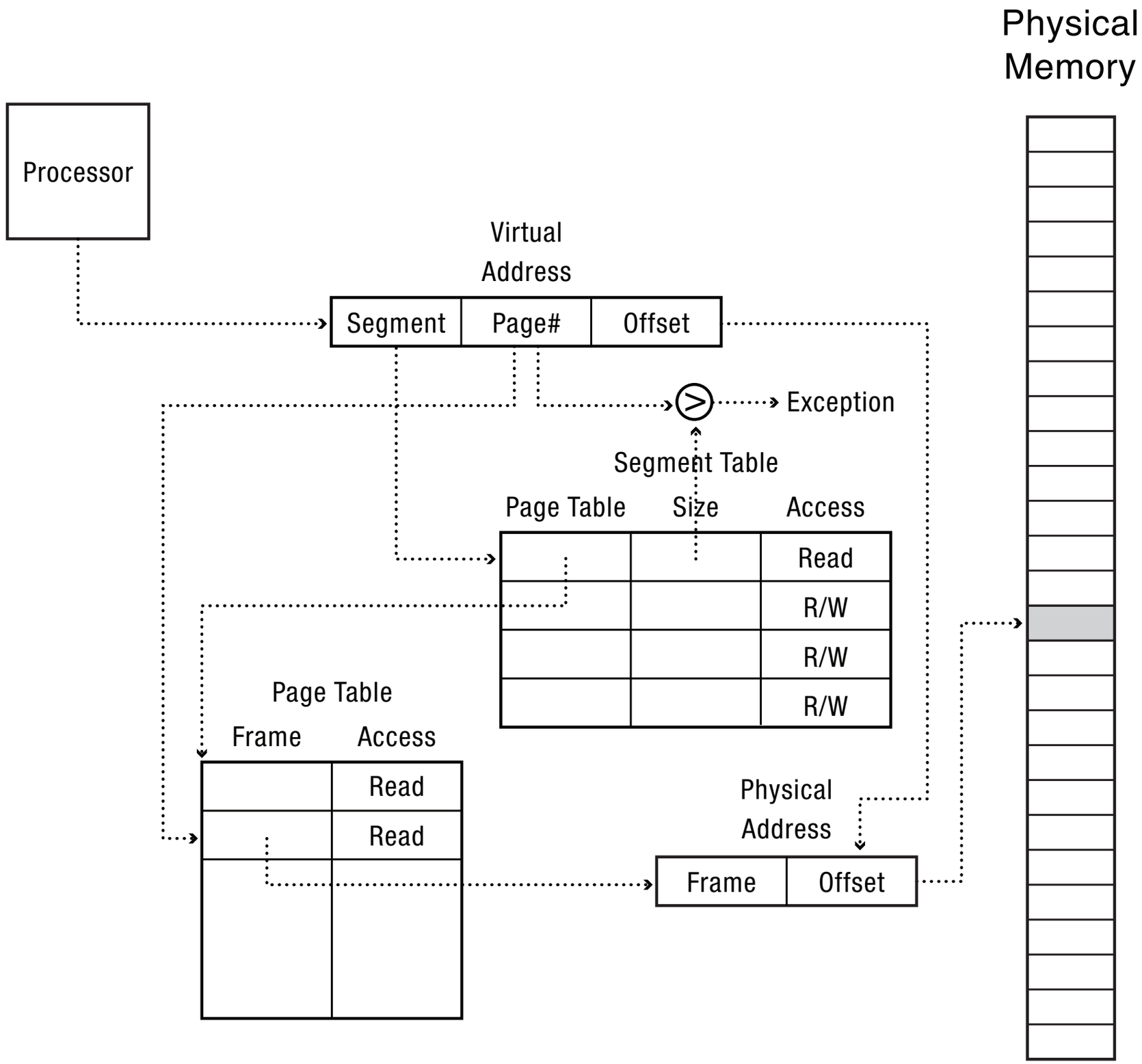
Paging

- Each process has its own page-table structure
- It's a *tree*, not a chain
  - May have looked like a chain in the previous slides…
  - But it's not! As your memory footprint increases, the more the structure looks like a tree
- On context-switch, change it completely
  - On x86-64, that means reassigning the CR3 register
  - One-shot change

Alternative

(less important)

Paged Segmentation



Fim

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