Operating Systems

Paging - Smaller Tables

Hongliang Liang, BUPT

Sep. 2023

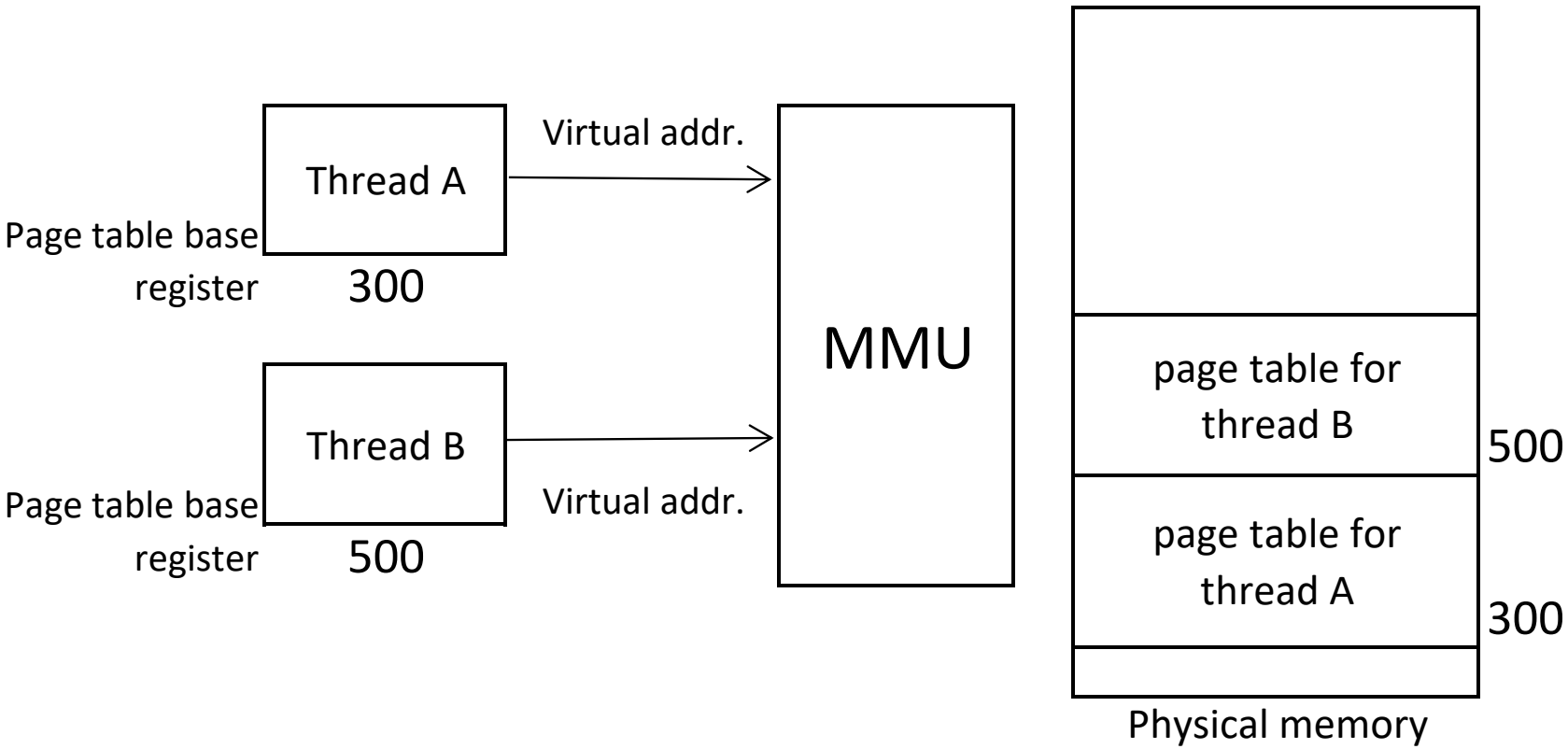
Storing the Page Table in Memory

Store the page table in **contiguous** physical memory

The **physical address** of the base of the page table is stored in a reserved processor register — the **page table base register (PTBR)** in BLITZ

- This register is only writable in kernel mode
- Each address space has its **own PTBR**
- Its value is typically stored in the Process Control Block so that it can be reloaded with user registers

Threads in their own address spaces



Paging: Two major challenges

1. The mapping from virtual to physical address must be **fast**
2. If the virtual address space is large, the linear page table will be large!

- If the page size is 2^{12} bytes, virtual addresses are 64 bits wide, physical addresses are 32 bits wide (4GB memory), how large is the linear page table?

Paging: Two major challenges

1. The mapping from virtual to physical address must be **fast**
2. If the virtual address space is large, the linear page table will be large!

- If the page size is 2^{12} bytes, virtual addresses are 64 bits wide, physical addresses are 32 bits wide (4GB memory), how large is the linear page table?

- Answer: 2^{52} x (20 bits + status bits) = at least 12 PB (preferably 16 PB)
- Making matters worse: each address space has its own page table!

Designing Page Tables

Page table size depends on

- Size of a page
- Size of the virtual address space

Designing Page Tables

Page table size depends on

- Size of a page
- Size of the virtual address space

Memory used for page tables is **overhead**

- How can we conserve/save space, and still find entries **quickly**?

Linear Page Tables

Why linear page tables can potentially be large?

- Requires one page table entry per virtual page, even if it may not be mapped to a physical frame

Linear Page Tables

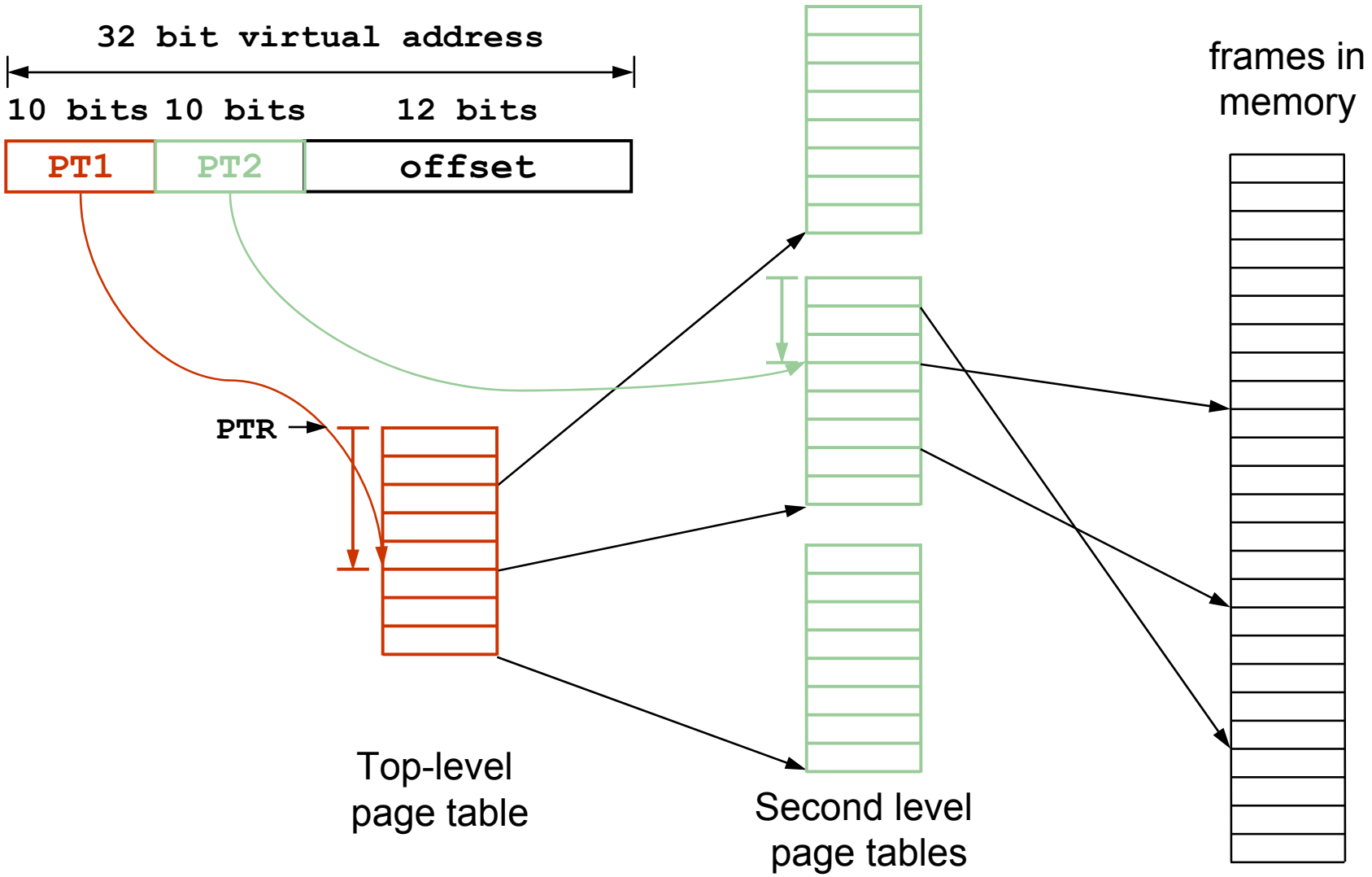
Why linear page tables can potentially be large?

- Requires one page table entry per virtual page, even if it may not be mapped to a physical frame

Possible ideas to solve this problem

- **Introduce a hierarchy**: multi-level page table
- An **inverted** page table

Multi-level Page Tables



The size of a page table is a page. 4 bytes/entry x 1024 entries

An address translation involves three memory reference.

Benefits of Multi-level Page Tables

Is address translation faster with a single-level page table or a multi-level page table?

Benefits of Multi-level Page Tables

Is address translation faster with a single-level page table or a multi-level page table?

- Single-level page table is faster

How does a multi-level page table conserve space compared to a single-level page table?

Benefits of Multi-level Page Tables

Is address translation faster with a single-level page table or a multi-level page table?

- Single-level page table is faster

How does a multi-level page table conserve space compared to a single-level page table?

- **Not all** pages within an address space are **allocated**
- e.g., consider the region between heap and stack
- This region does not need any physical frames
- So there is no need to maintain mapping information

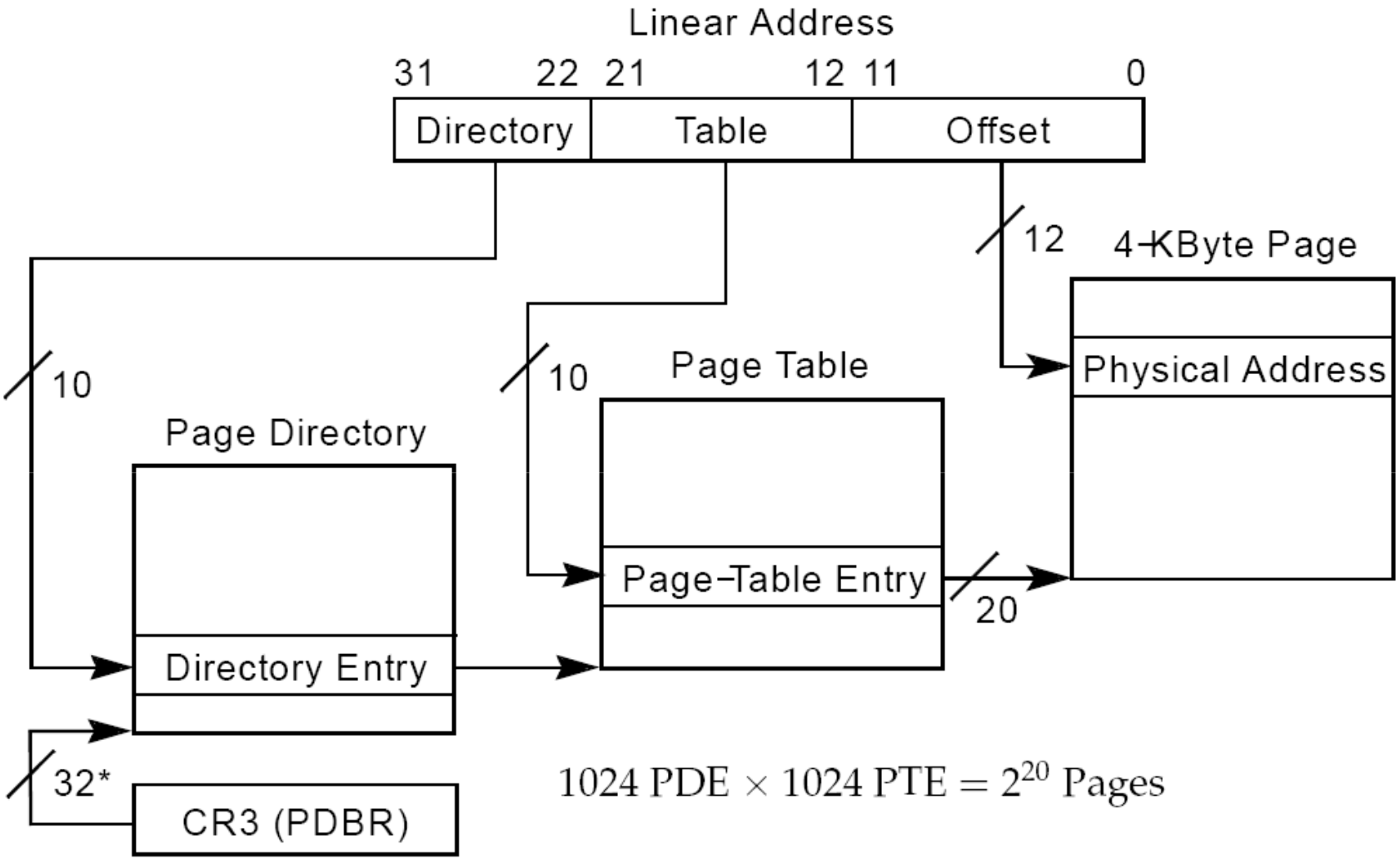
Some 2nd level page tables can be empty and do not need to be allocated!

Intel x86 Architecture

Intel x86 architecture uses two-level page tables

- 10 bits (first level) + 10 bits (second level) + 12 bits (offset)
- 4KB long **page directory** — each entry has 32 bits
- 4KB long **page tables** — each entry has 32 bits
- 4KB long pages

Intel x86 Paging



Intel x86 Paging: Physical Address Extension

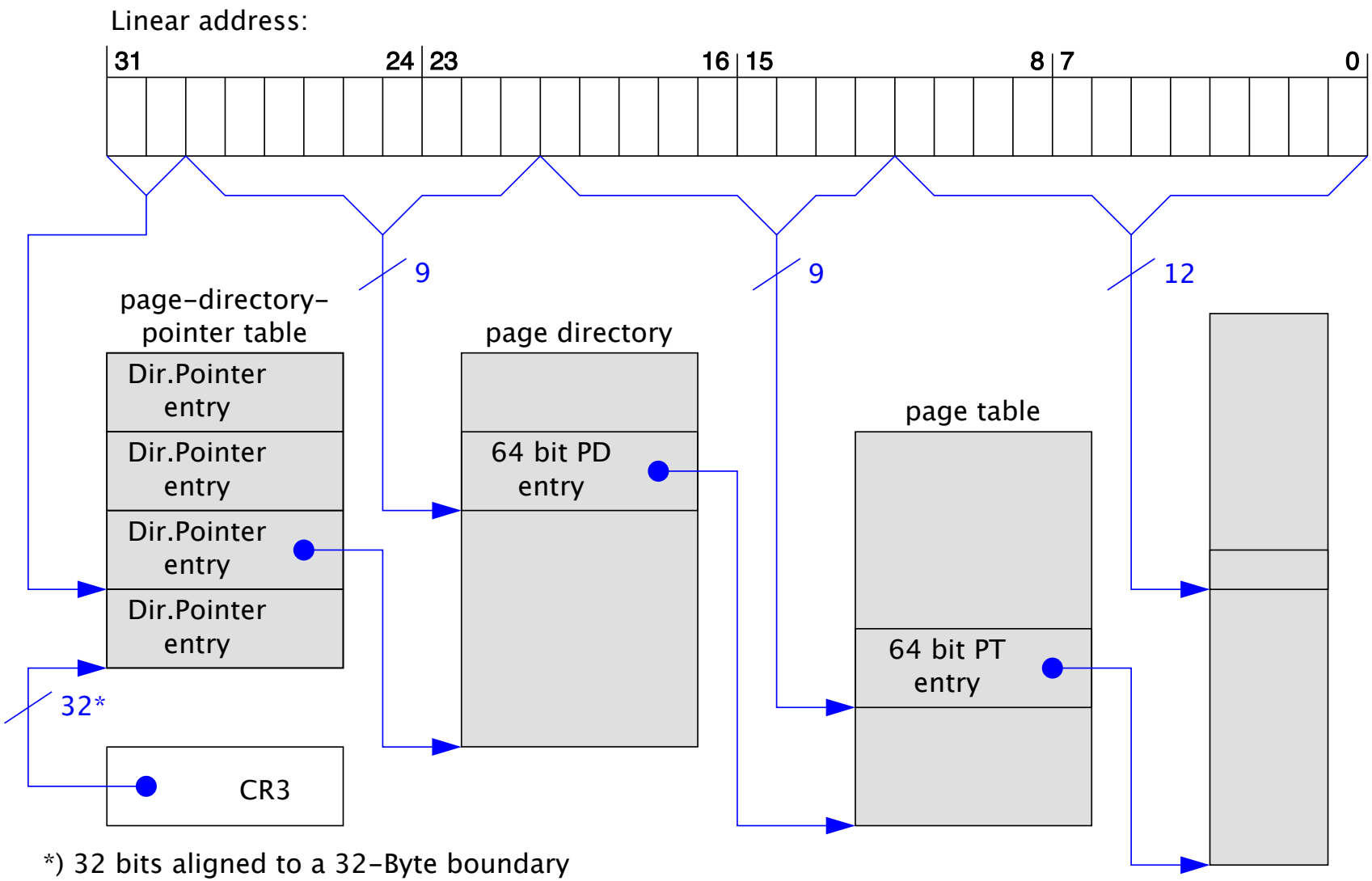
What if we need to address more than 4GB of physical memory?

Intel x86 Paging: Physical Address Extension

What if we need to address more than 4GB of physical memory?

- Physical Address Extension (PAE): supported by x86 (since Pentium Pro) and x86-64
- **Three-level** hierarchy used in page tables at 4KB pages
- 2 bits + 9 bits (first level) + 9 bits (second level) + 12 bits (offset) — each page directory/page table entry has **64** bits

Intel x86 Paging: Physical Address Extension



64-bit virtual addresses in x86-64: long mode

Page size: still 4KB (12 bits for the offset), or 21 bits / 30 bits for the offset

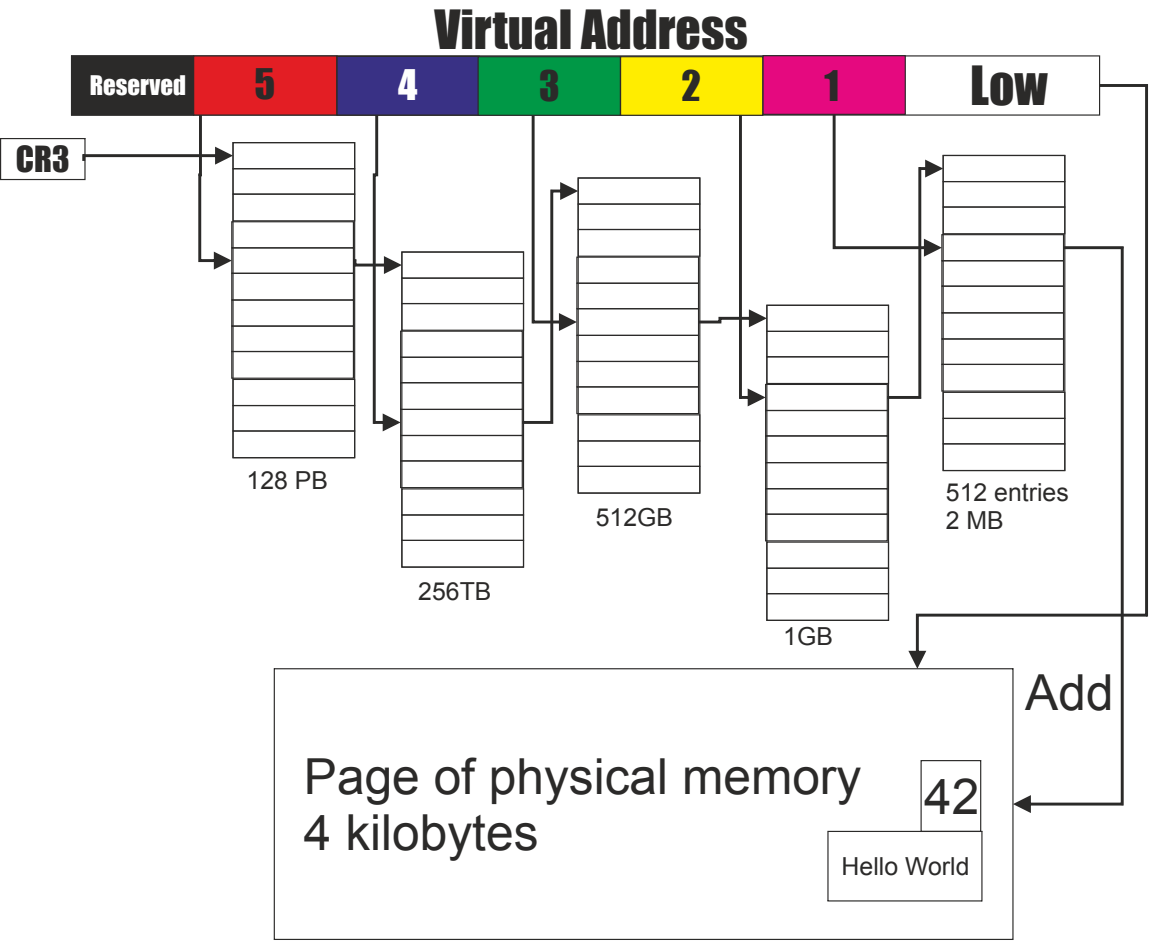
64-bit virtual addresses in x86-64: long mode

Page size: still 4KB (12 bits for the offset), or 21 bits / 30 bits for the offset

Four levels of page tables —

- PAE's Page-Directory Pointer Table is extended from four entries to 512
- an additional Page-Map Level 4 (PML4) Table is added, containing 512 entries in 48-bit implementations
- 256 TB of virtual address space
- Requires 64-bit OS, 64-bit Unified Extensible Firmware Interface (UEFI) firmware (which replaces BIOS firmware)

Intel 5-Level Paging



Used for current x86-64 line of Intel processors

Extends the size of virtual addresses from 48 bits to 57 bits

Addressable virtual memory from 256 TB to 128 PB

First implemented in Ice Lake processors

Linux kernel support since 4.14

Inverted Page Tables

Both single and multi-level page tables allocate a page table entry per page of memory

Page table overhead increases with virtual address space size

Inverted Page Tables

What is the maximum number of mappings needed at any time?

Inverted Page Tables

What is the maximum number of mappings needed at any time?

Need mappings only for physical memory that **exists**

- Consider a computer with 64 bit virtual addresses, but only 256 MB physical memory
- 256 MB (2^{28}) physical memory can only hold 2^{16} 4 KB pages
- Need a total of 2^{16} entries $= 2^{19}$ bytes $=$ 512 KB!

Used in PowerPC, UltraSPARC, and IA-64 architectures

Inverted Page Tables

An inverted page table stores one entry for every **frame** of physical memory

Records which page is in that frame

- Indexed by **frame number**, not page number

Inverted Page Tables

An inverted page table stores one entry for every **frame** of physical memory

Records which page is in that frame

- Indexed by **frame number**, not page number

Problem

- Address translation requires page -> frame mapping
- So how should an inverted page table be searched?

Options for solution

Inverted Page Tables

An inverted page table stores one entry for every **frame** of physical memory

Records which page is in that frame

- Indexed by **frame number**, not page number

Problem

- Address translation requires page -> frame mapping
- So how should an inverted page table be searched?

Options for solution

- Search all entries to find a matching page number. Exhaustive search is too slow

Inverted Page Tables

An inverted page table stores one entry for every **frame** of physical memory

Records which page is in that frame

- Indexed by **frame number**, not page number

Problem

- Address translation requires page -> frame mapping
- So how should an inverted page table be searched?

Options for solution

- Search all entries to find a matching page number. Exhaustive search is too slow
- Use a hash table with a good hash of page numbers. O(1) search time!