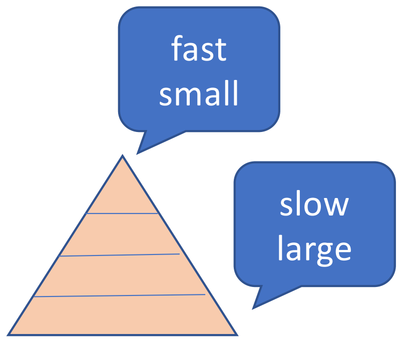
Virtual memory

Jinyang Li

Some slides are from Patterson and Hennessy

What we've learnt last time

- CPU design
  - Single cycle
  - 5 stage pipeline
- Memory hierarchy
- Caching
  - Why it works: principles of locality
  - Direct map vs. fully associative vs. n-way associative



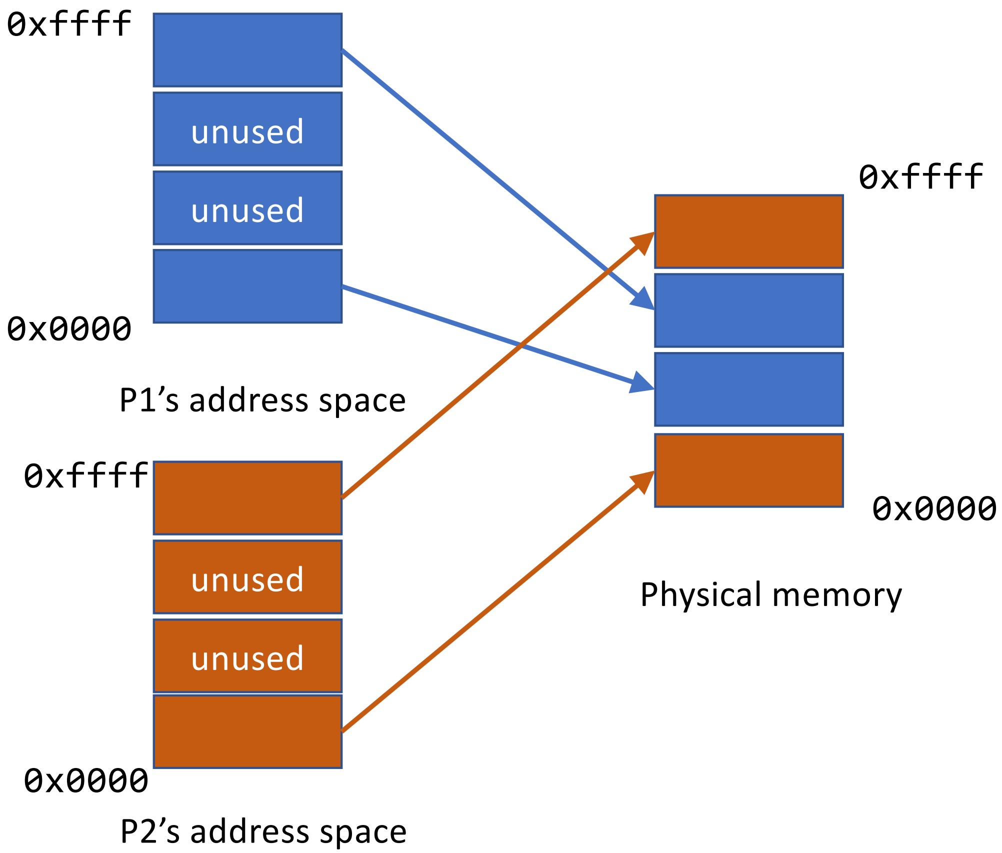
Today's lesson plan

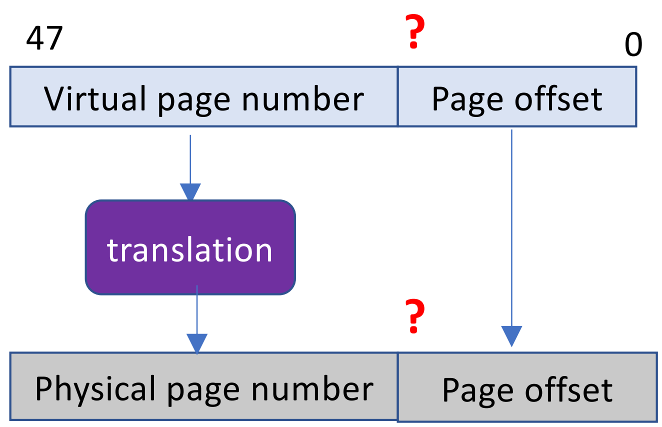
- Virtual memory
  - Why?
  - How? Address translation
  - Make it fast

Virtual Memory: goals

- Goal #1: Use main memory as a “cache” for disk storage
- Goal #2: Allow programs to share main memory safely
  - Each process gets a private virtual address space, isolated from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

Sharing memory safely using VM





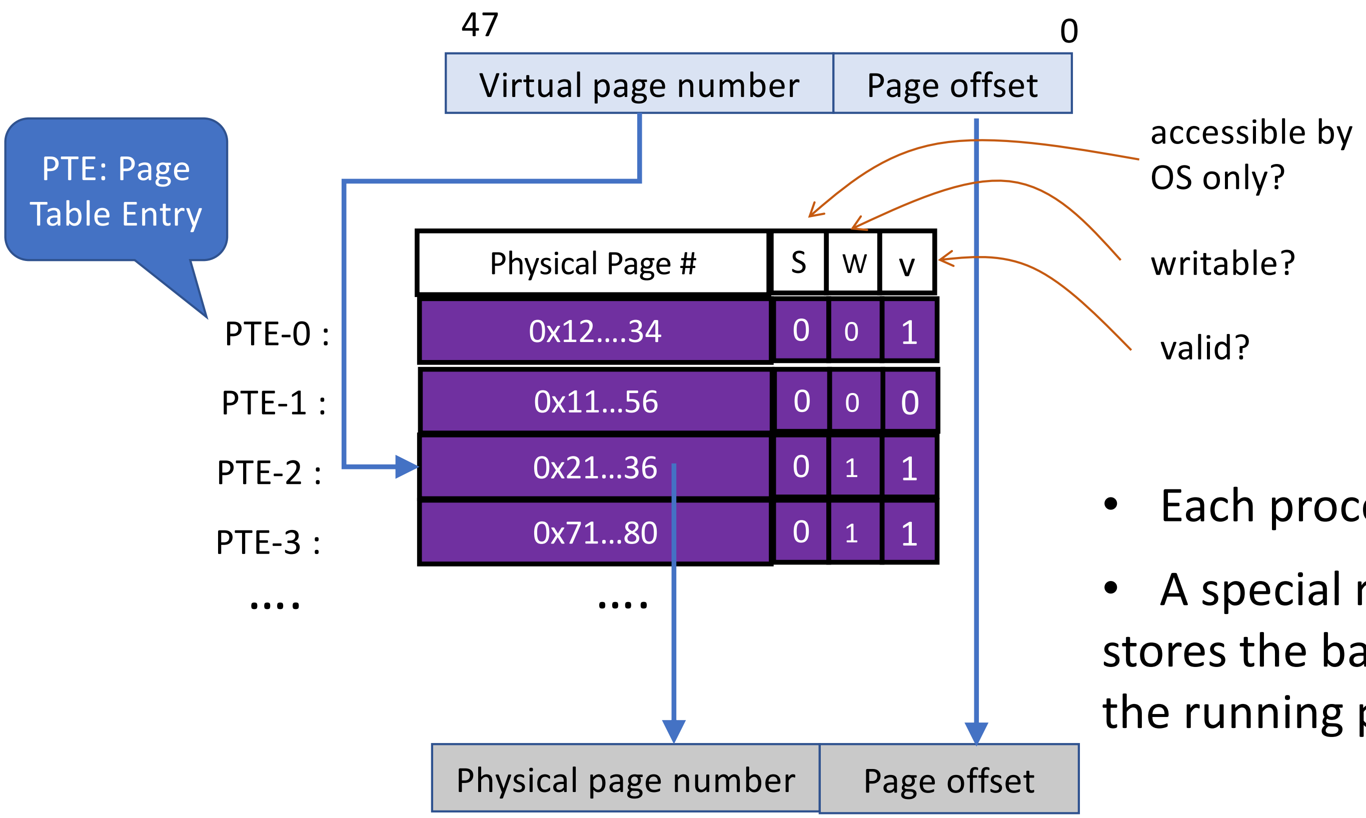
Suppose page size is 4KB, how many bits in page offset?

2^{12}= 4KB, so 12 bits

Mapping from Virtual to Physical pages

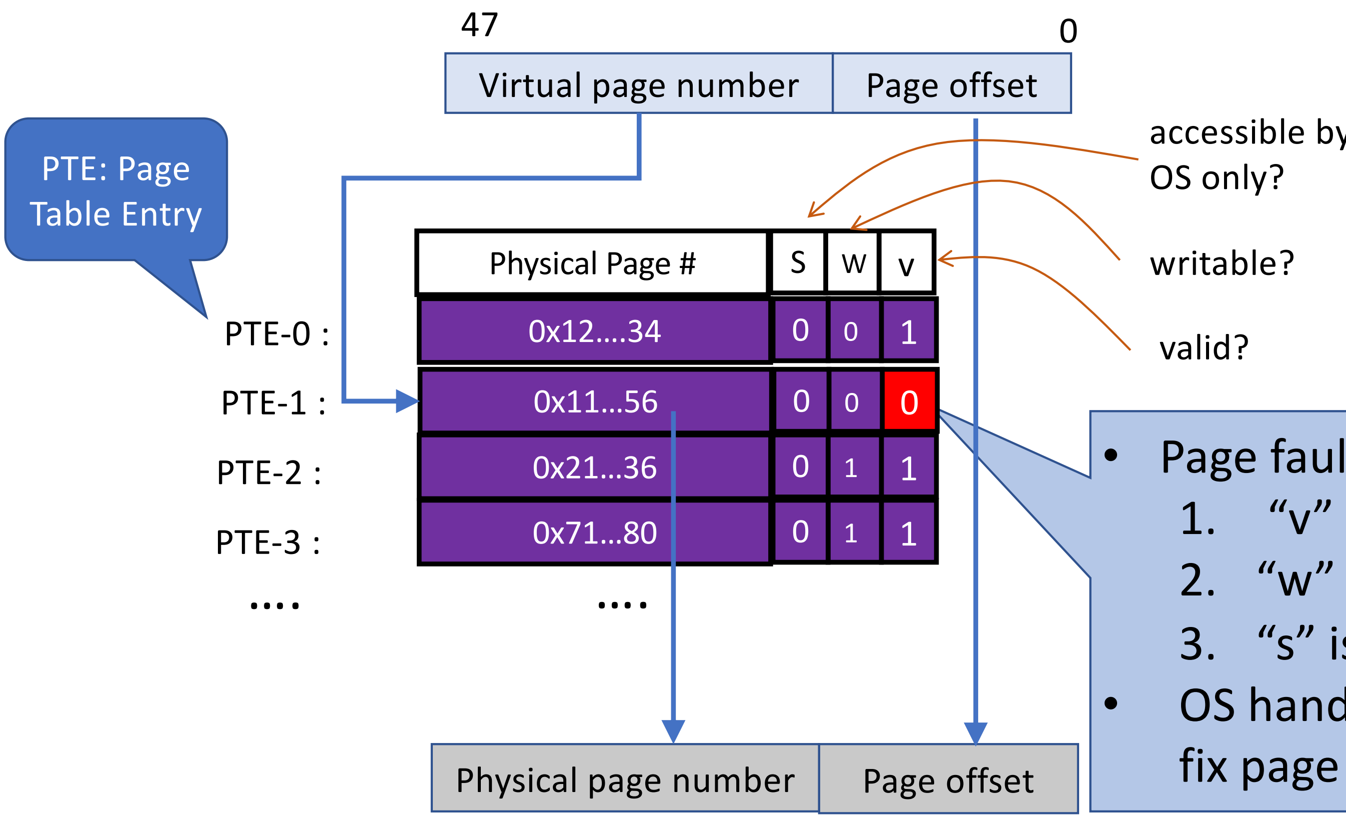
- “Fully associative”
  - A virtual page can map to any physical page
- Use an index called “page table” to store the mapping
  - No need to check all physical pages for a match
- OS populates the page table for each process
  - Page table is stored in the main memory
- CPU (MMU) performs translation using the page table

Translation using the page table



- Each process has its own page table
- A special register, "page table register" stores the base (physical) address of the running process' page table

Page Fault



- Page fault occurs when
  1. "v" is invalid
  2. "w" is invalid for a write access
  3. "s" is invalid for a non-OS access
- OS handles page fault by either fix page table or terminate process

One level page table is impractical

- How large is the page table for 48-bit virtual address space?
  - Page size 4KB

$2^{48}/2^{12} = 2^{36}$ pages

Page Table size = 2^{36} pages * 8 byte-PTE/page = 2^{39} bytes = 0.5TB!!!

Multi-level page tables

Problem

- how to reduce # of page table entries required?

Solution

- Multi-level page table
  - A tree of “page tables”

2-level Paging Example

- 6-bit virtual and physical address, 4-byte page

2-level Paging Example

- 6-bit virtual and physical address, 4-byte page
- 1-level page table

Suppose a process has only two 2 mapped virtual pages

2^4 PTEs

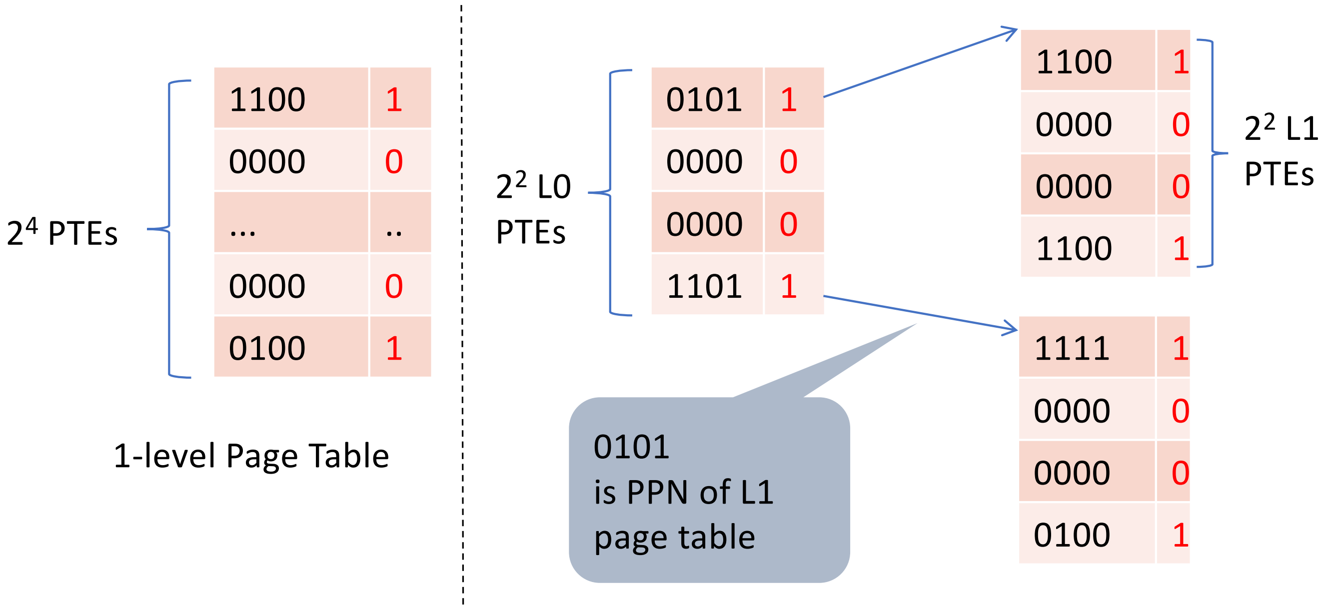
PPN	v
1100	1
0000	0
…	..
0000	0
0100	1

1-level Page Table

2-level Paging Example

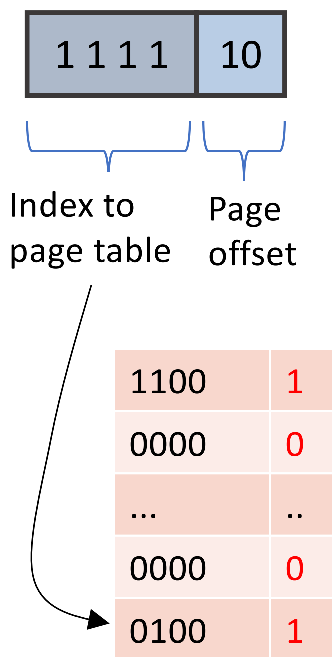
- 6-bit virtual and physical address, 4-byte page
- 2-level page table

Suppose a process has only two 2 mapped virtual pages



2-level Paging Example

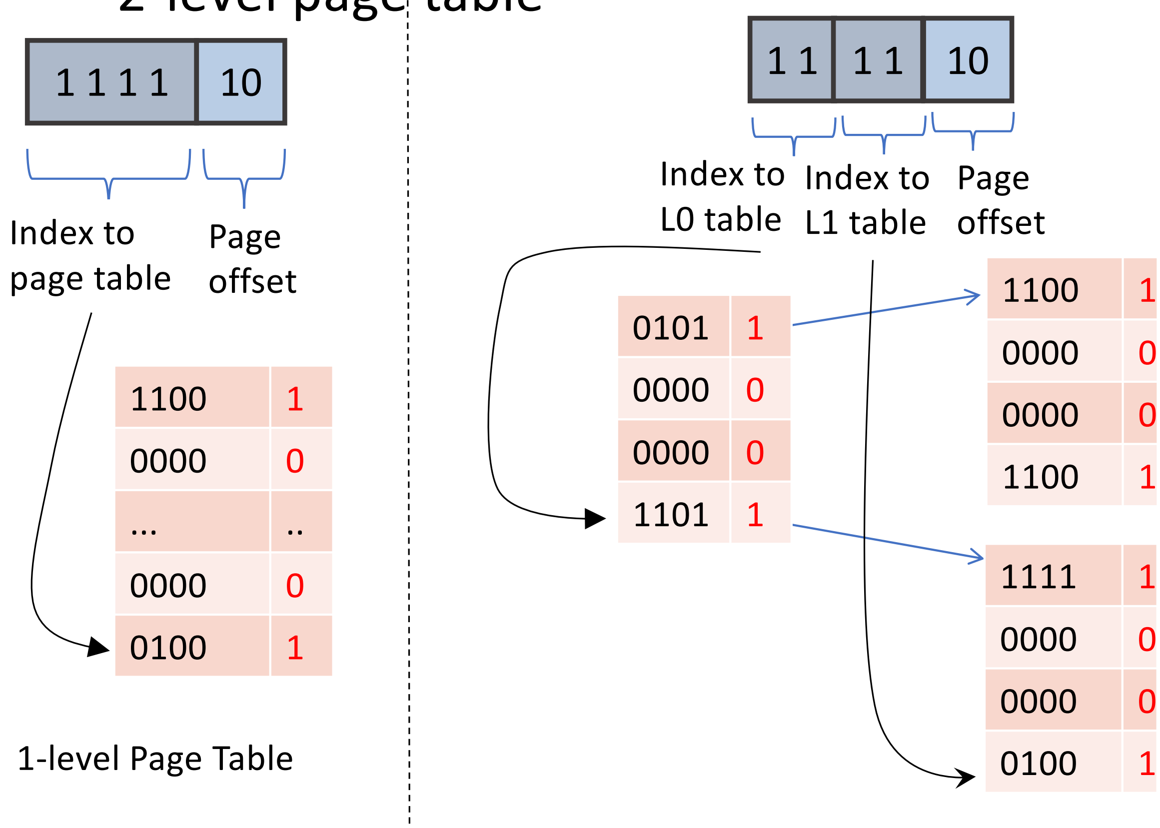
- how to perform address translation?
  - 1-level page table



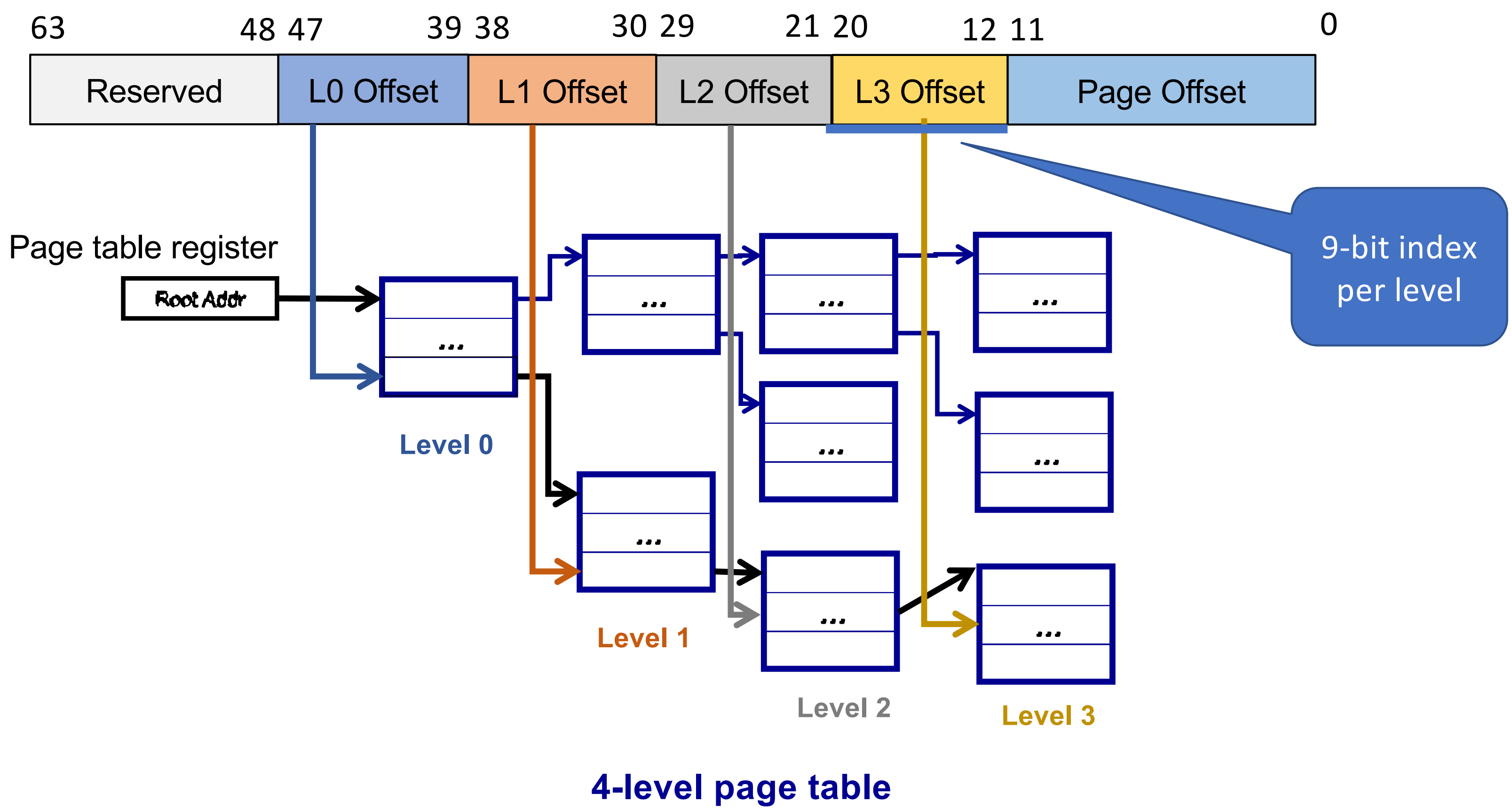
1-level Page Table

2-level Paging Example

- how to perform address translation?
  - 2-level page table

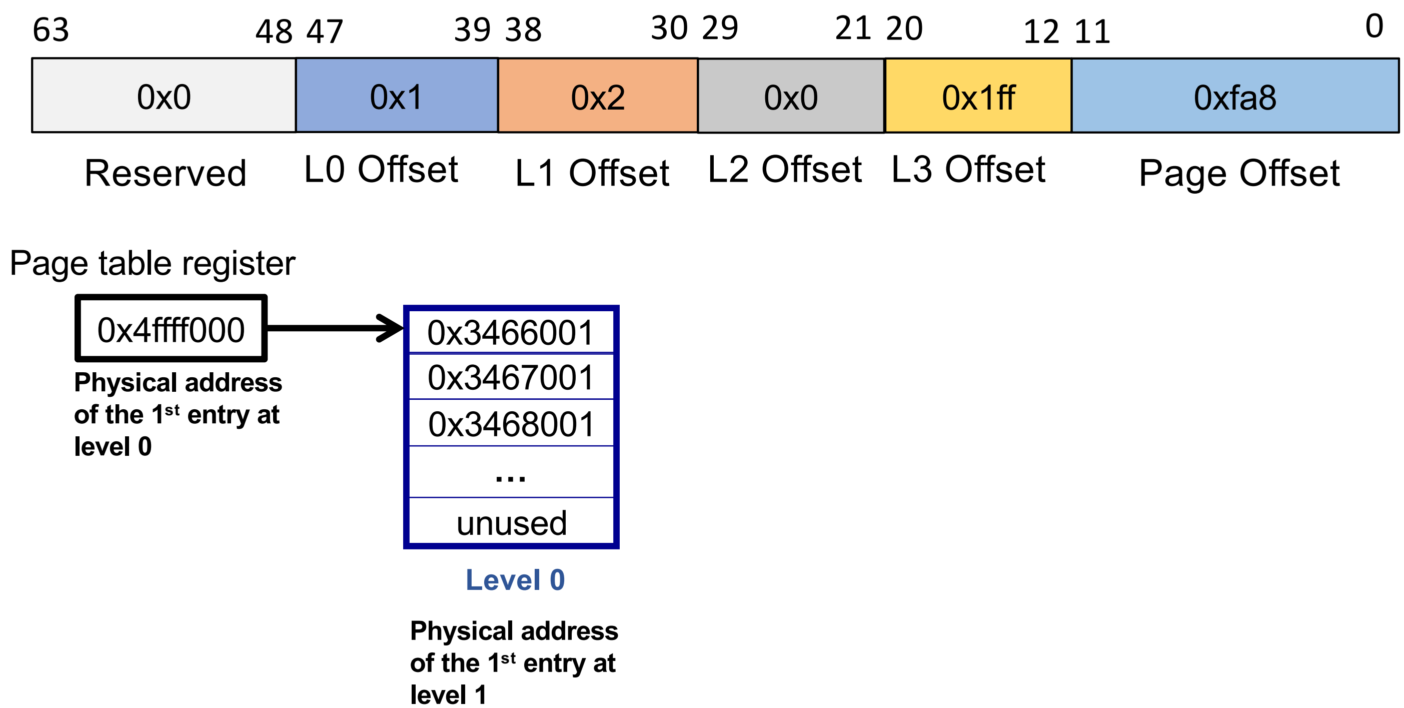


X86_64 and RISC-V use 4-level page table



Example translation using a 4-level table

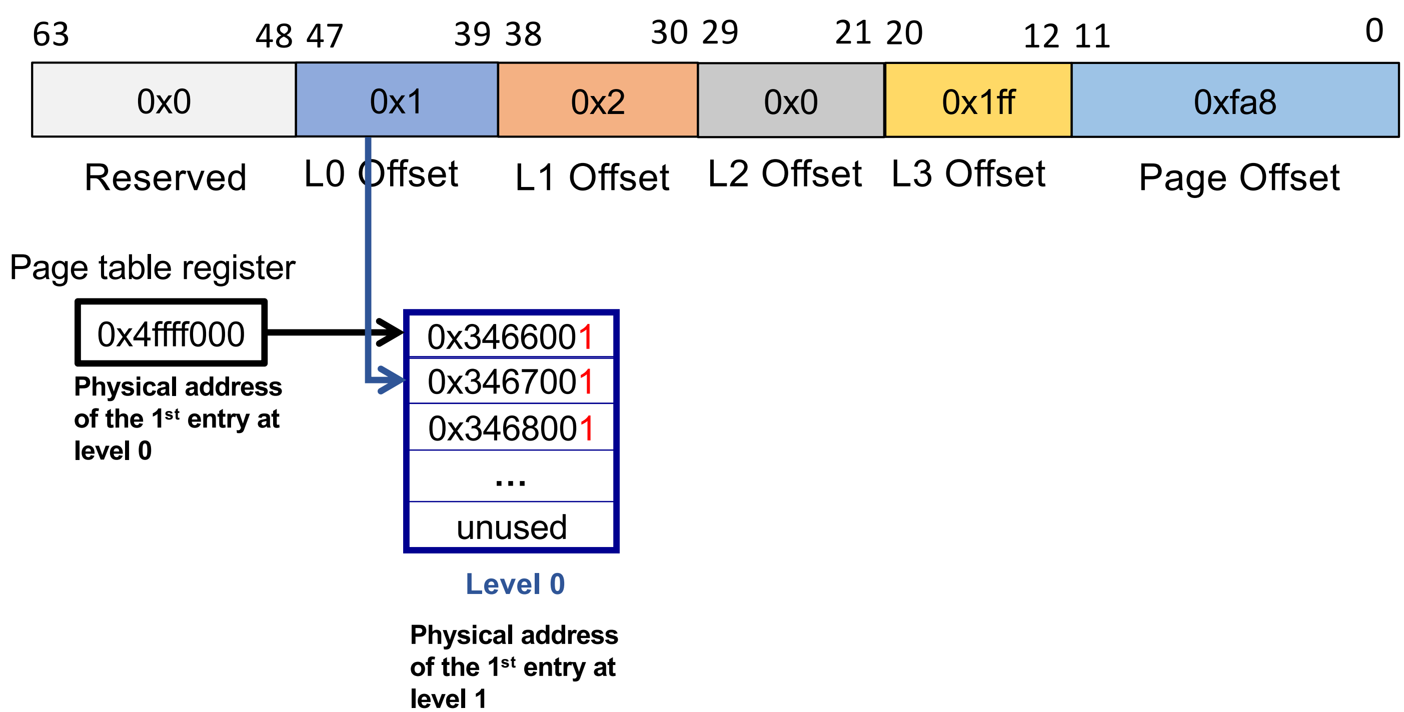
Virtual Address: *0x80801fffa8*



4-level page table

Example translation using a 4-level table

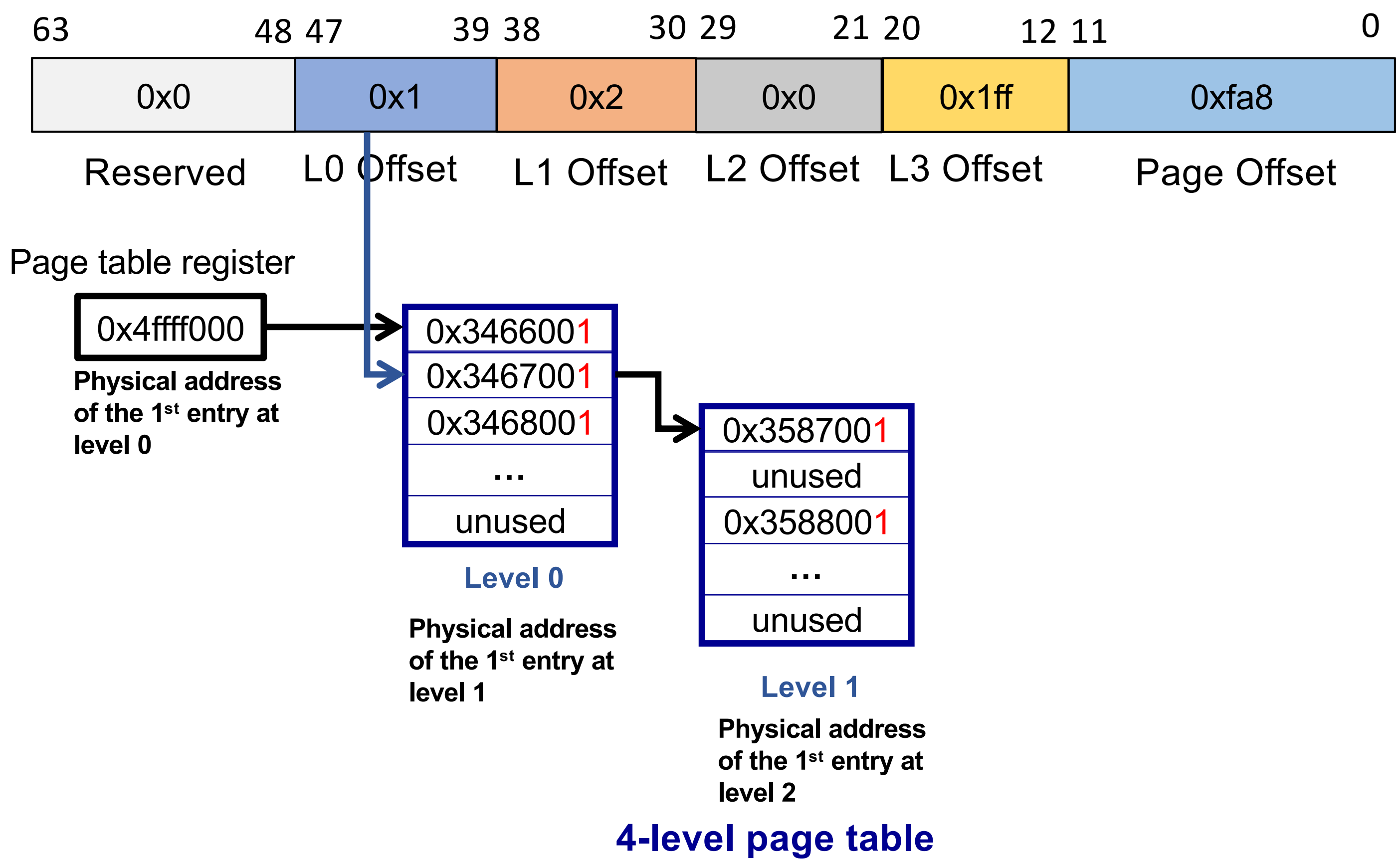
Virtual Address: *0x80801fffa8*



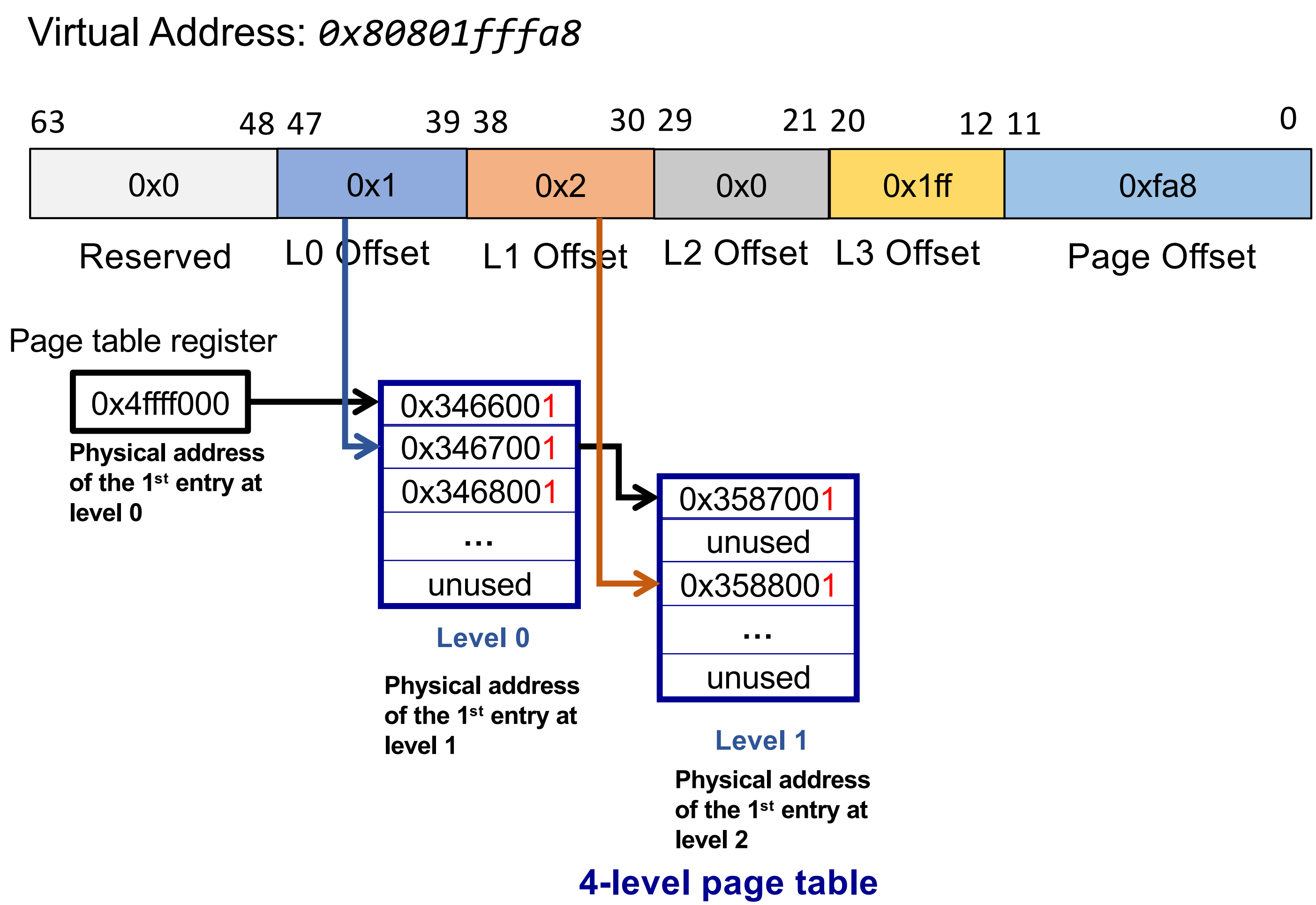
4-level page table

Example translation using a 4-level table

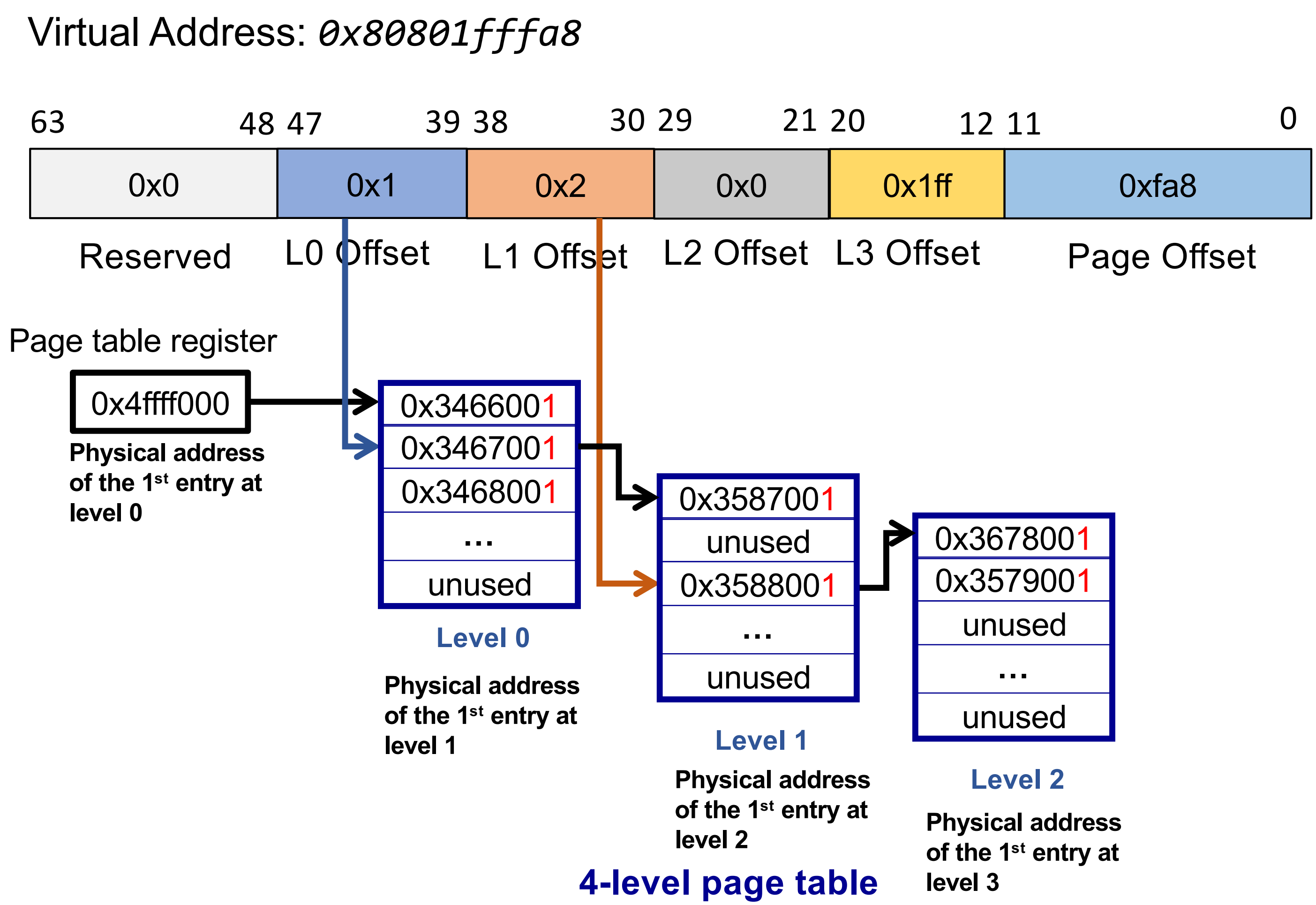
Virtual Address: *0x80801fffa8*



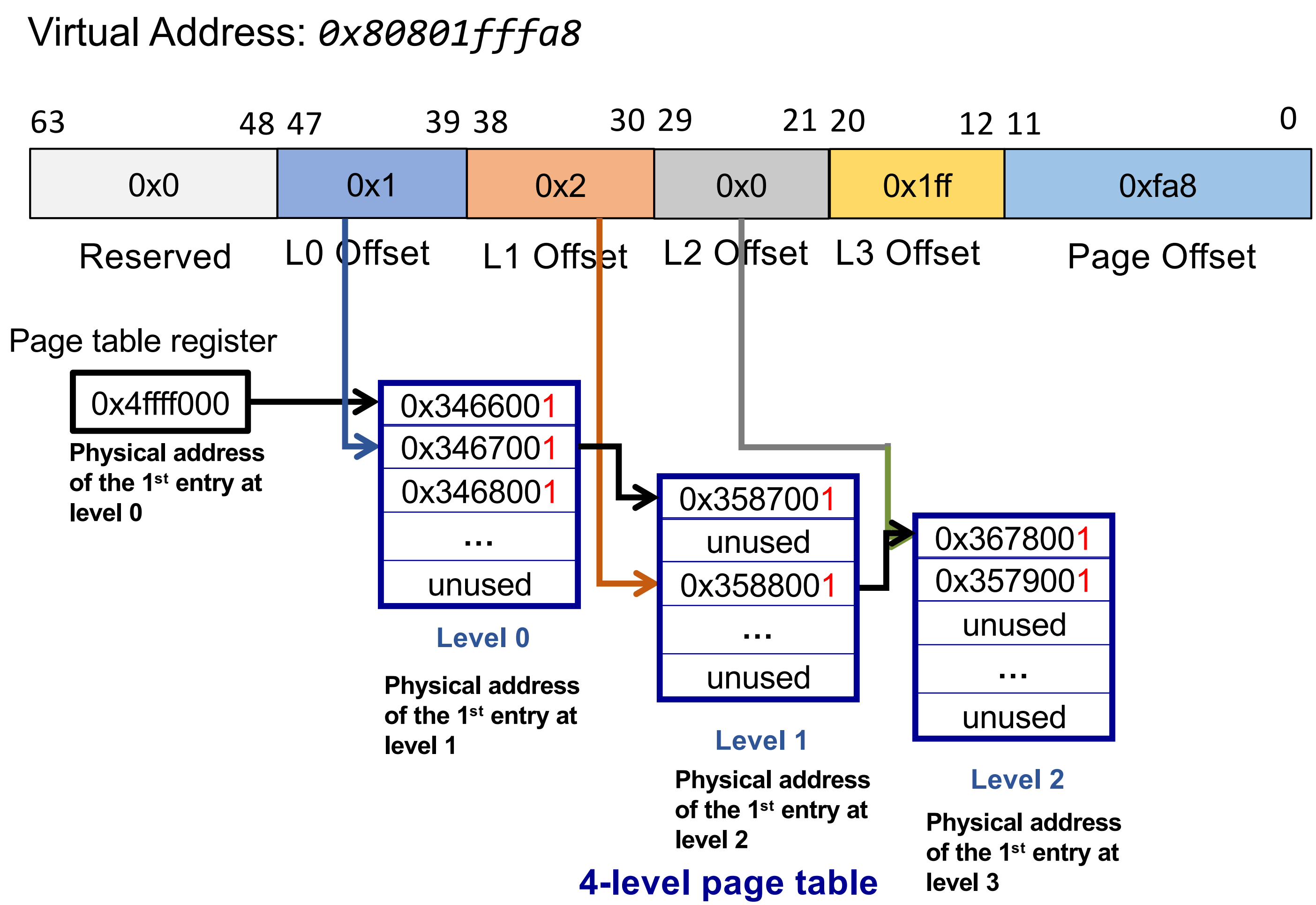
Example translation using a 4-level table



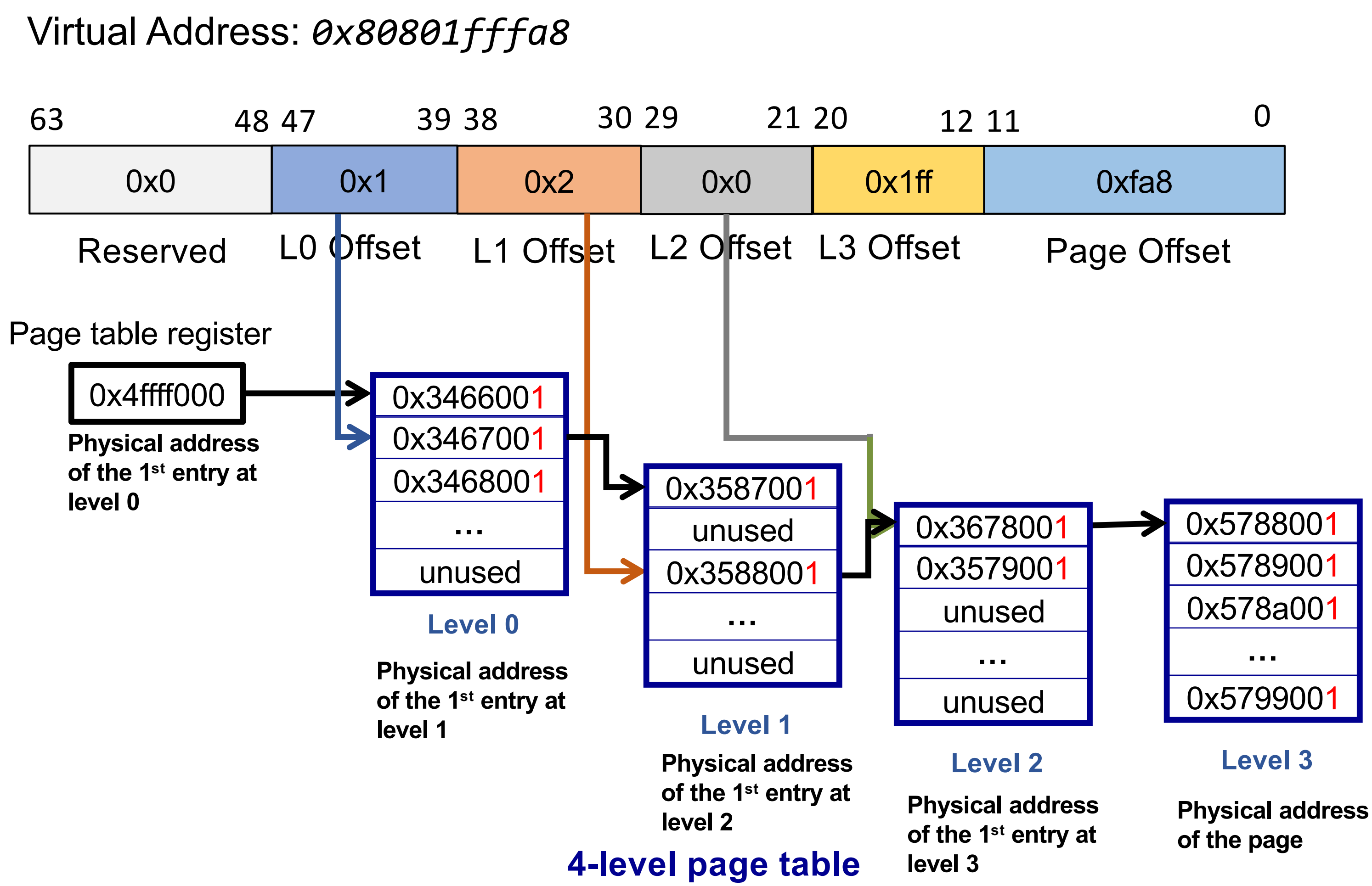
Example translation using a 4-level table



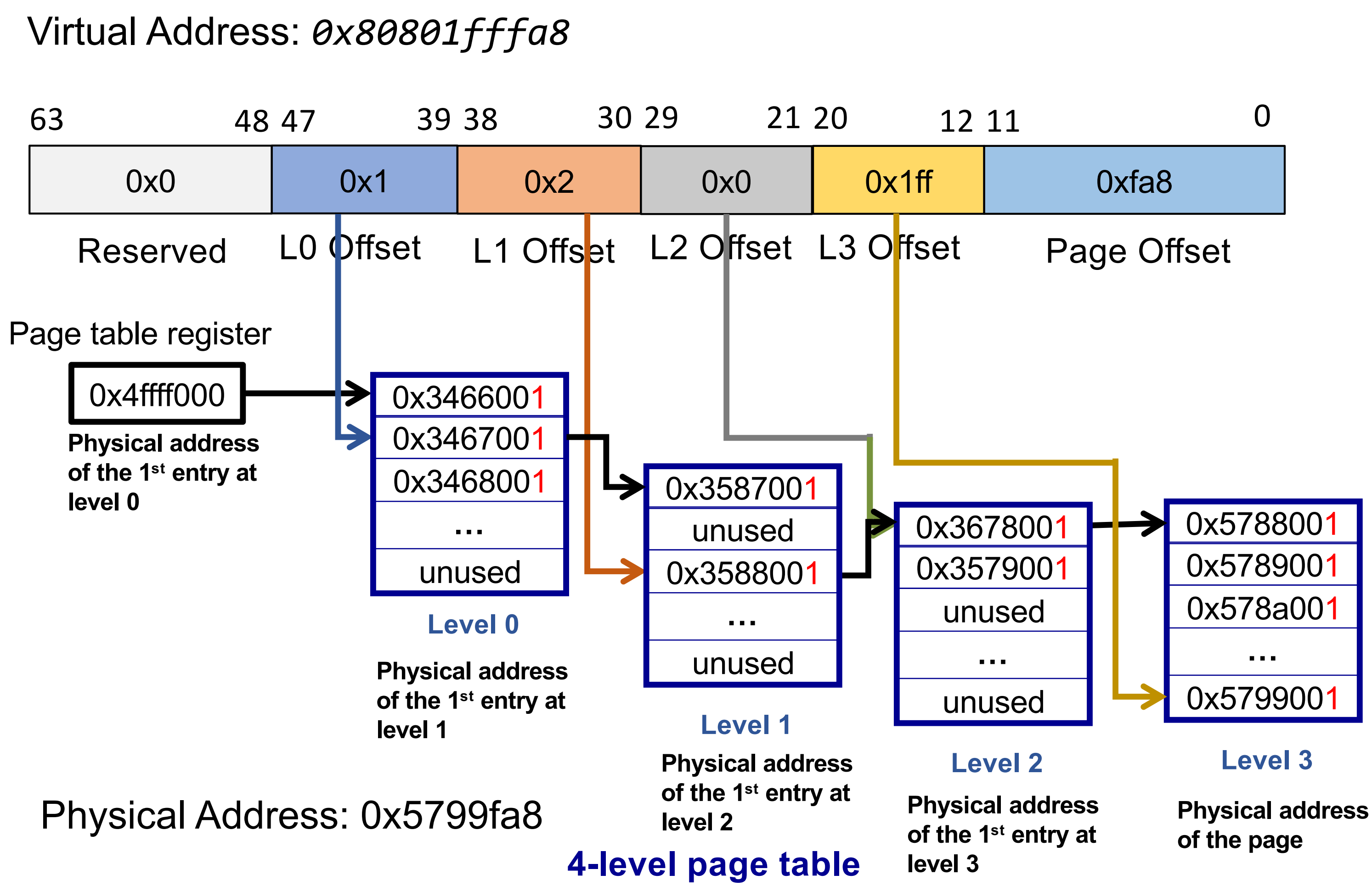
Example translation using a 4-level table



Example translation using a 4-level table



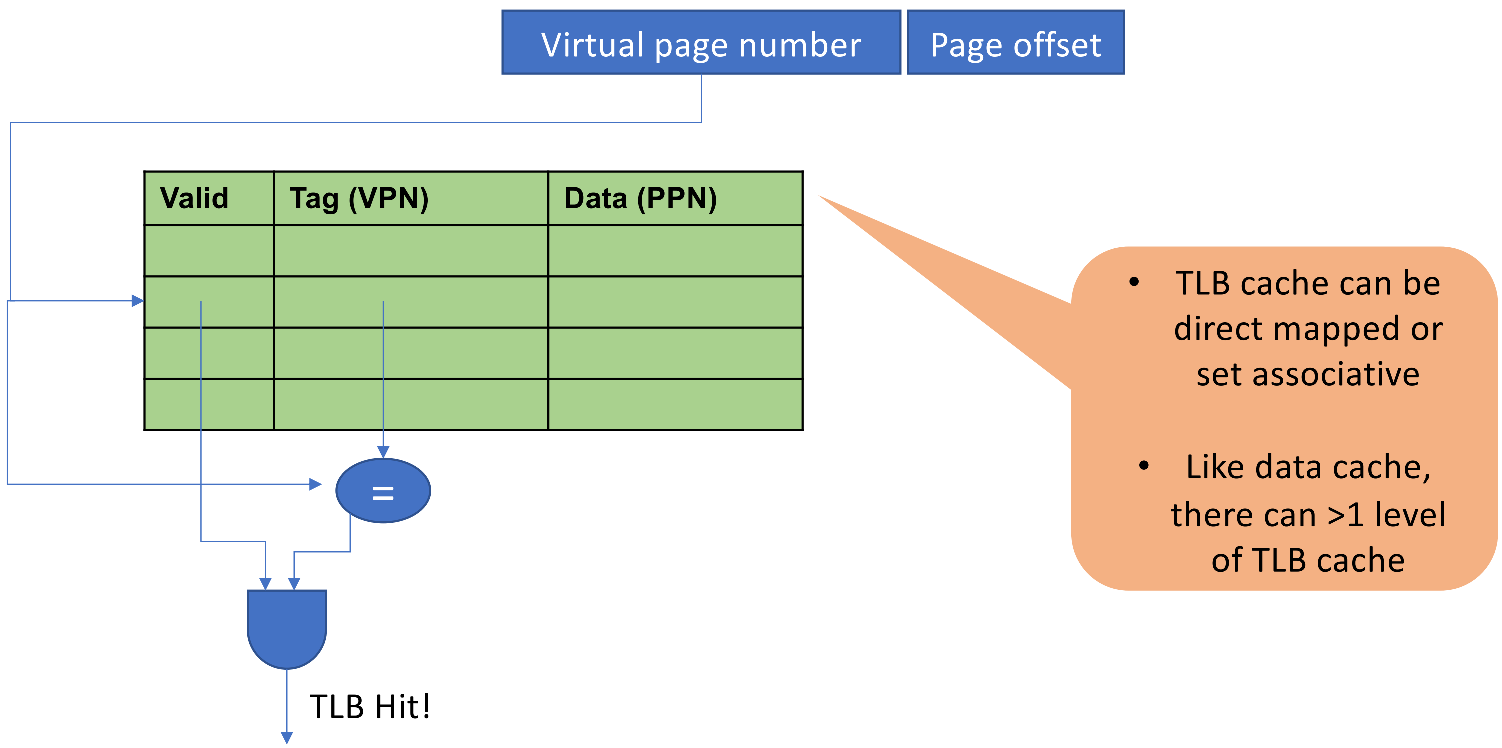
Example translation using a 4-level table



Fast translation using a TLB

- Address translation is costly
  - How many memory accesses per actual data access? (4-level page table)
  - 4 (page table access) + 1 (data access)
- Solution?
  - Cache virtual→physical page mappings in a fast small cache
  - This cache is called Translation Lookaside Buffer (TLB)

Address translation with TLB



- If TLB miss, translate using 4-level page table, populate TLB with resulting VPN->PPN

Summary

- Virtual memory allows multiple processes to share memory safely
- Address translation uses multi-level page table
  - Speed up translation using TLB
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for performance