Virtual Memory

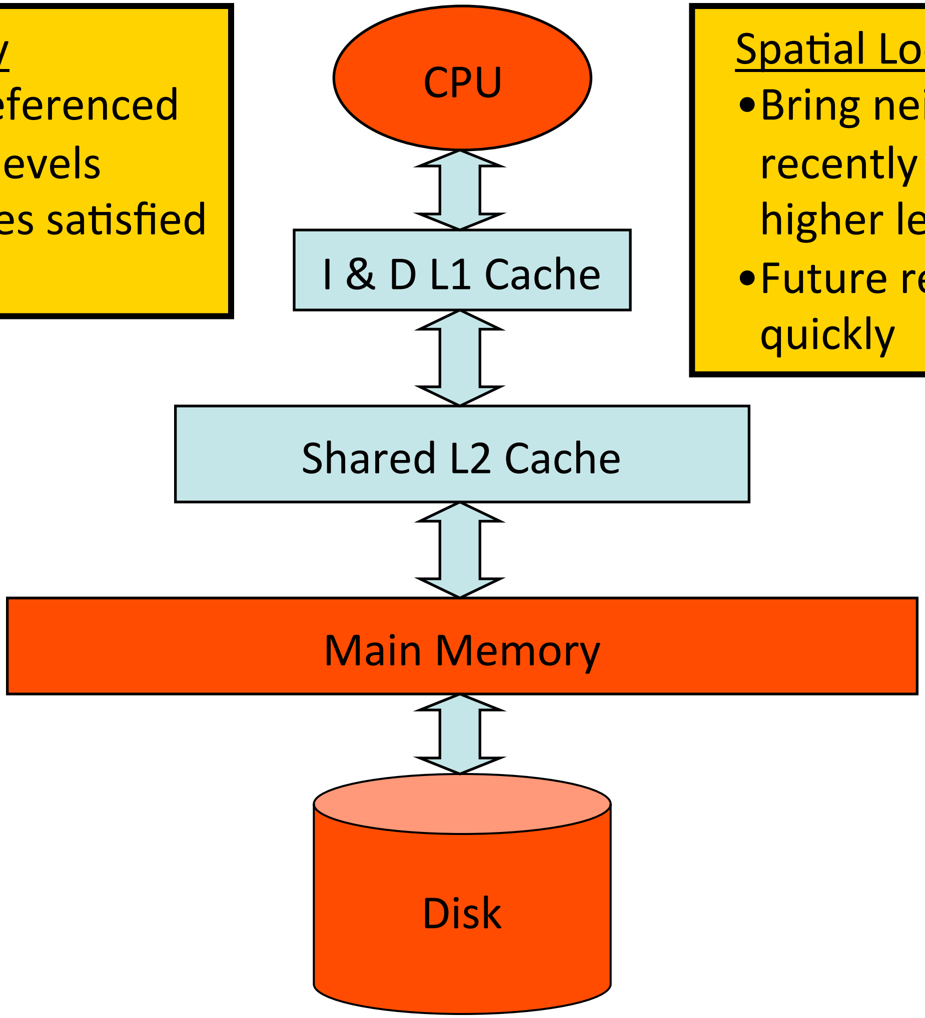
Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

CP-226: Computer Architecture

Memory Hierarchy

<u>Temporal Locality</u>

- Keep recently referenced items at higher levels
- Future references satisfied quickly



CPU

I & D L1 Cache

Shared L2 Cache

Main Memory

Disk

<u>Spatial Locality</u>

- Bring neighbors of recently referenced to higher levels
- Future references satisfied quickly

Virtual Memory Implementation

- Caches have fixed policies, hardware FSM for control, pipeline stall
- VM has very different miss penalties
  - Remember disks are 10+ ms!
- Hence engineered differently

Page Faults

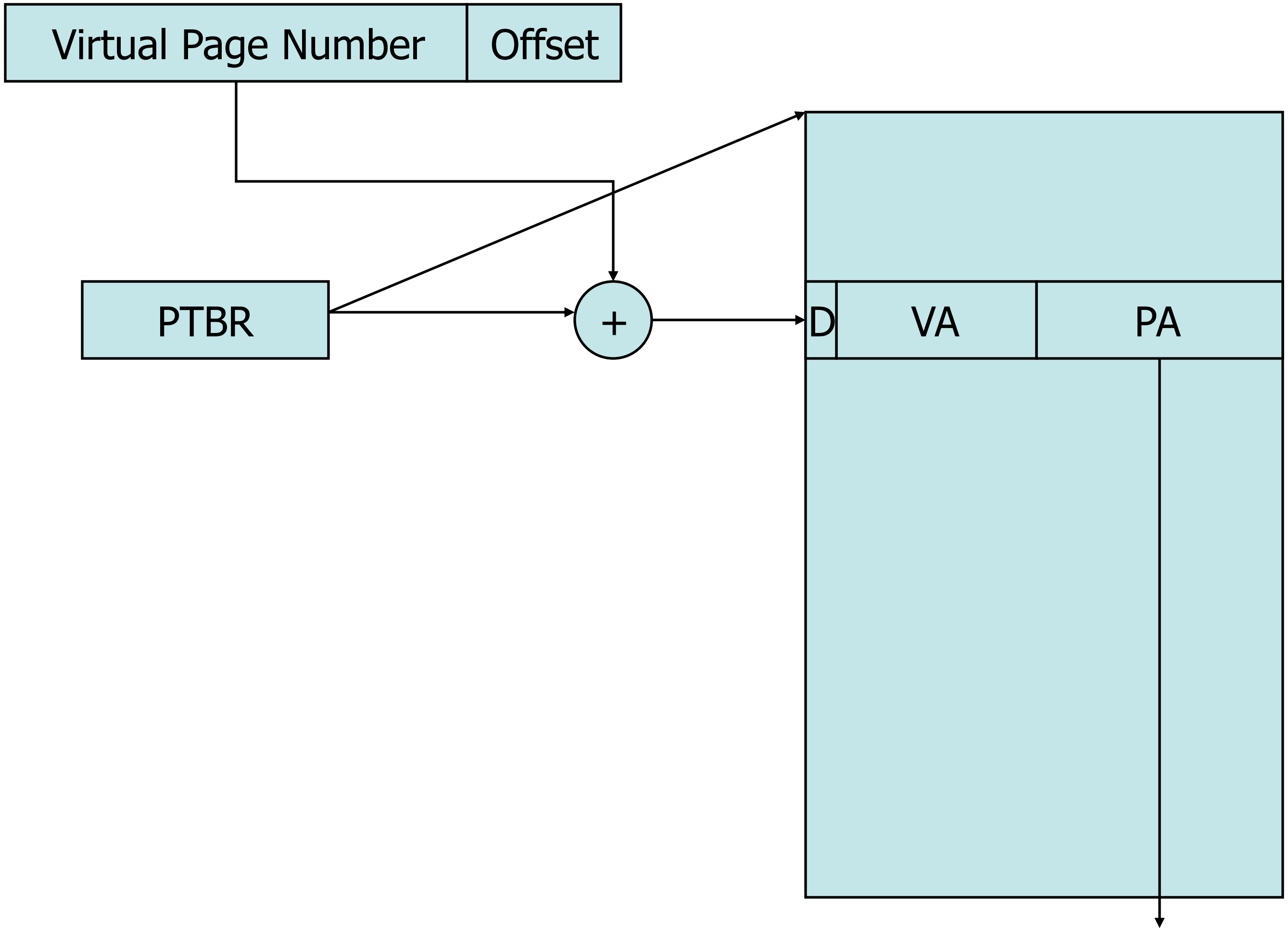
- A virtual memory miss is a page fault
  - Physical memory location does not exist
  - Exception is raised, save PC
  - Invoke OS page fault handler
    - Find a physical page (possibly evict)
    - Initiate fetch from disk
  - Switch to other task that is ready to run
  - Interrupt when disk access complete
  - Restart original instruction

Address Translation

VA	PA	Dirty	Ref	Protection
0x20004000	0x2000	Y/N	Y/N	Read/Write/ Execute

- O/S and hardware communicate via PTE
- How do we find a PTE?
  - &PTE = PTBR + page number * sizeof(PTE)
  - PTBR is private for each program
    - Context switch replaces PTBR contents

Address Translation

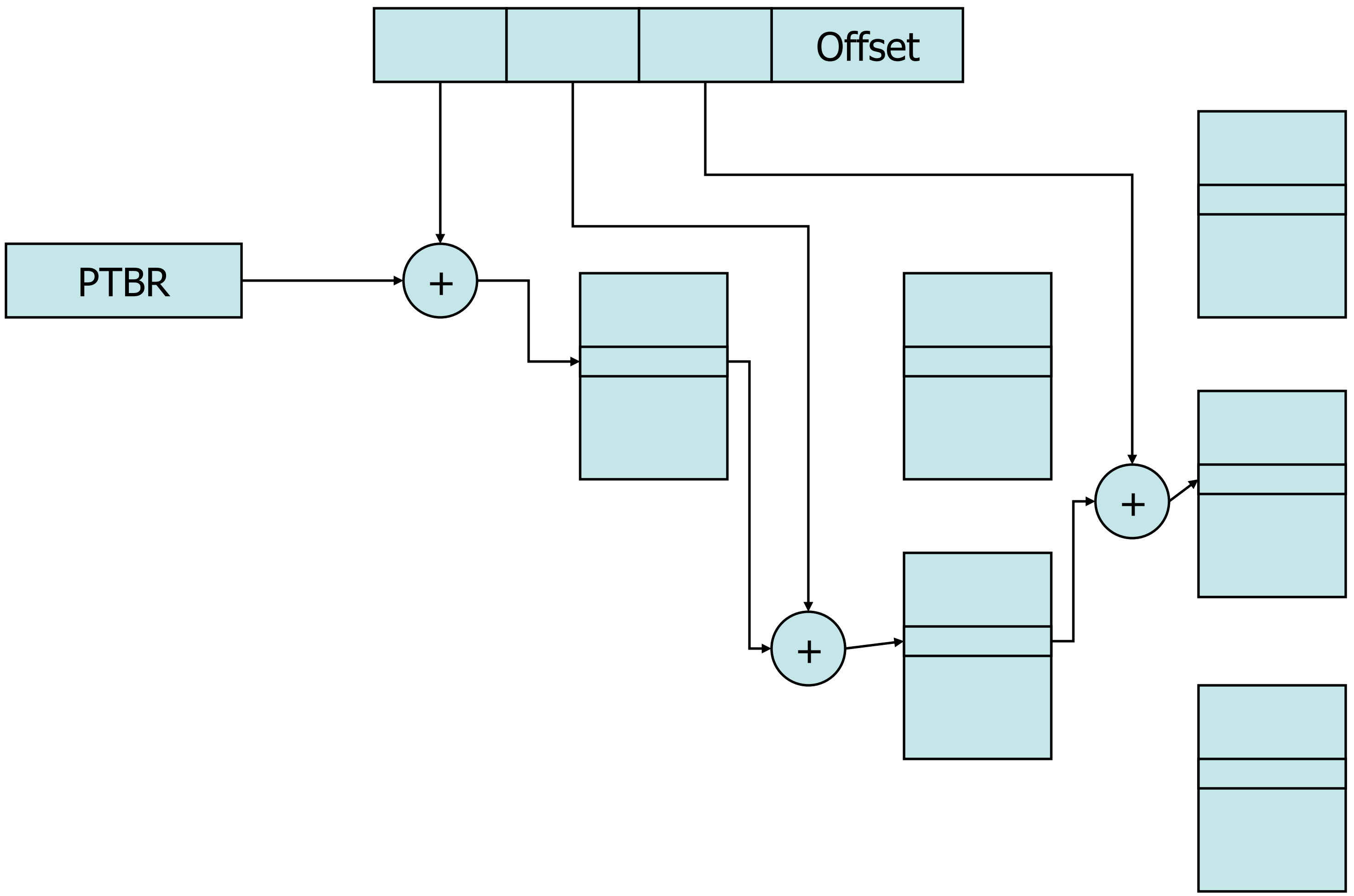


Page Table Size

- How big is page table?
  - 2^{32} / 4K * 4B = 4M per program (!)
  - Much worse for 64-bit machines

- To make it smaller
  - Use limit register(s)
    - If VA exceeds limit, invoke O/S to grow region
  - Use a multi-level page table
  - Make the page table pageable (use VM)

Multilevel Page Table

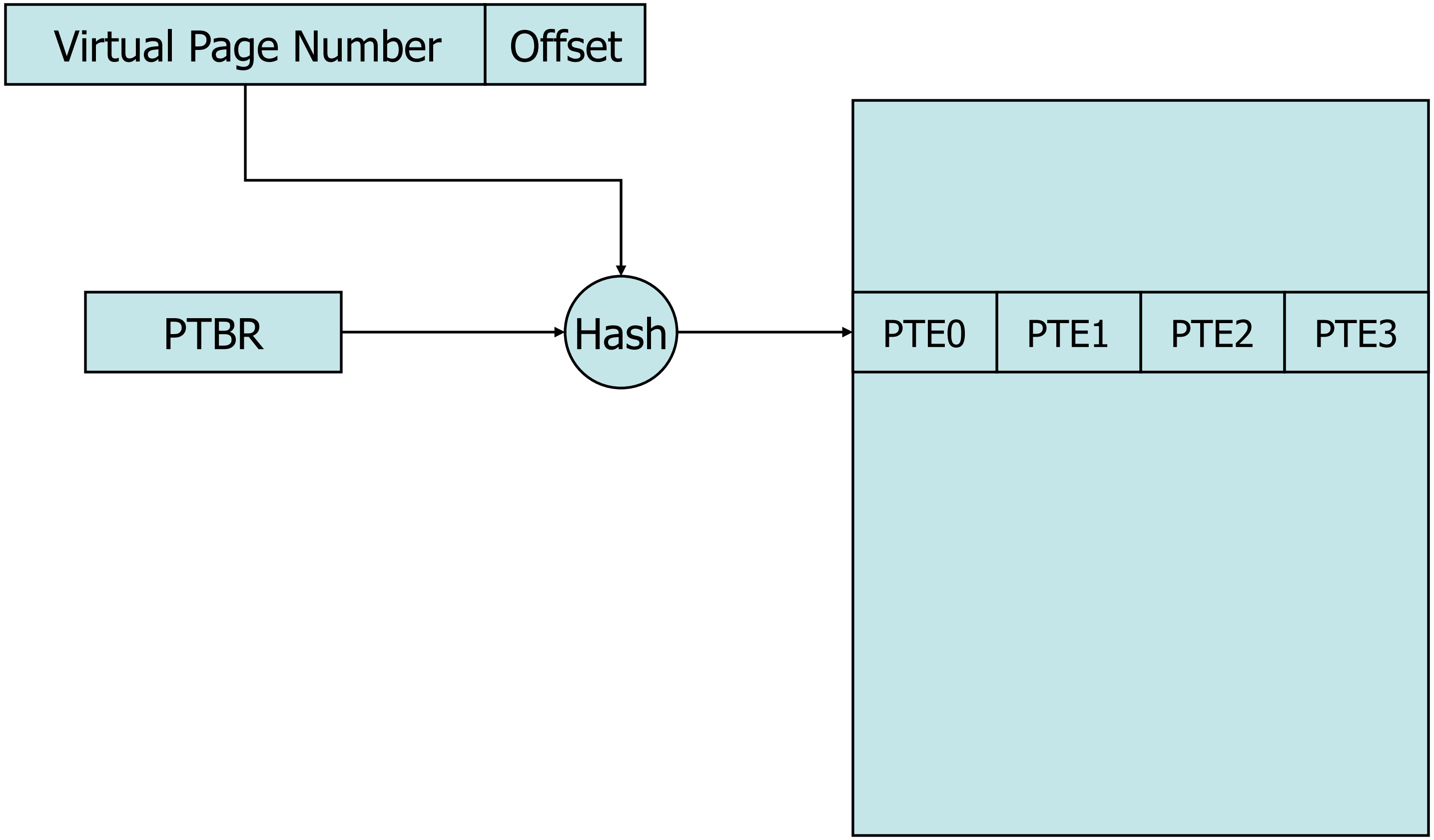


Hashed Page Table

- Use a hash table or inverted page table
  - PT contains an entry for each real address
    - Instead of entry for every virtual address
  - Entry is found by hashing VA
  - Oversize PT to reduce collisions:

    #PTE = 4 x (#phys. pages)

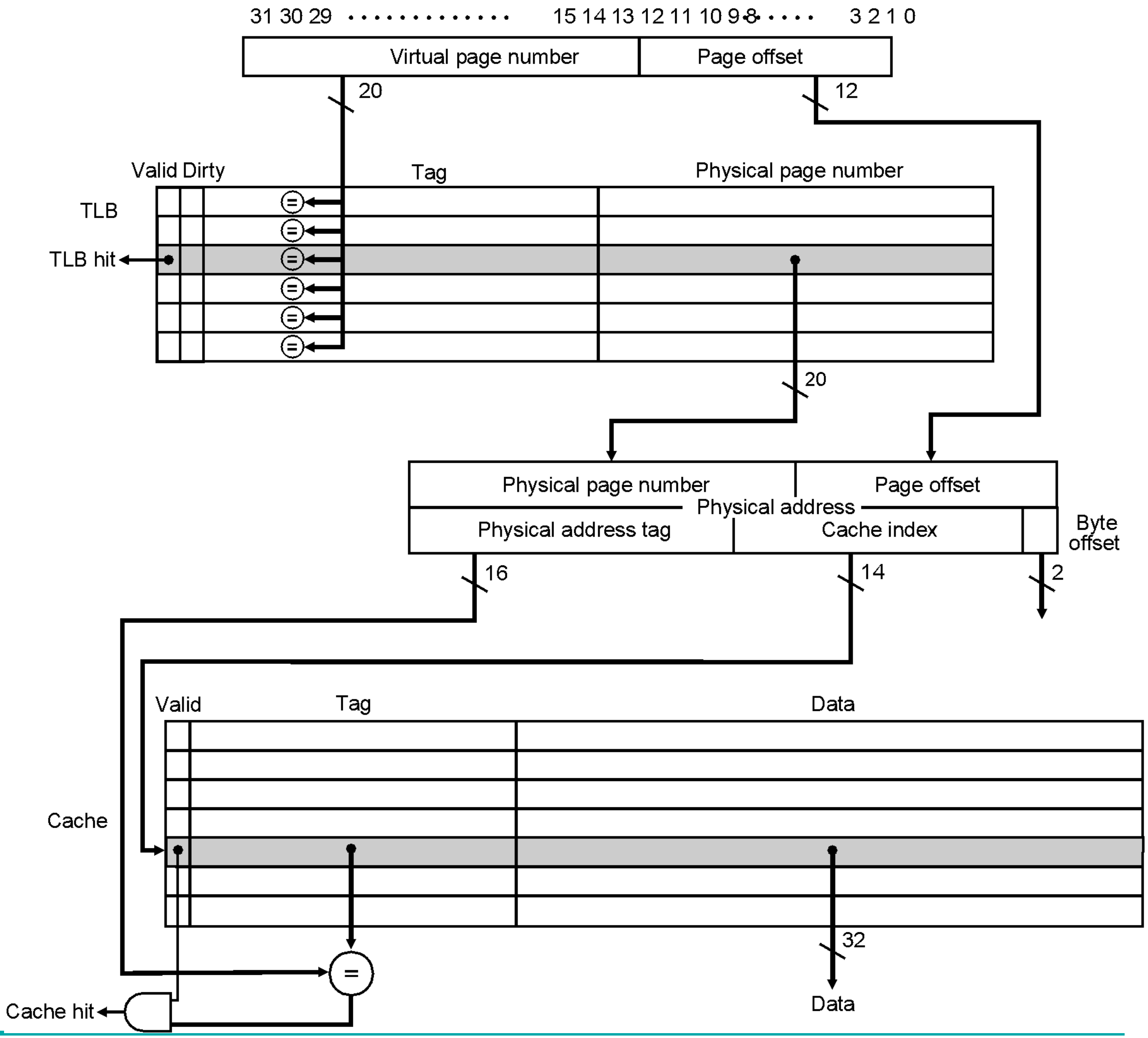
Hashed Page Table



High-Performance VM

- VA translation
  - Additional memory reference to PTE
  - Each instruction fetch/load/store now 2 memory references
    - Or more, with multilevel table or hash collisions
  - Even if PTE are cached, still slow
- Hence, use special-purpose cache for PTEs
  - Called TLB (translation lookaside buffer)
  - Caches PTE entries
  - Exploits temporal and spatial locality (just a cache)

TLB


31 30 29 . . . . 15 14 13 12 11 10 9 8 . . . . 3 2 1 0
Virtual page number
Page offset
20
12
Valid
Dirty
Tag
Physical page number
TLB
TLB hit
=
20
Physical page number
Page offset
Physical address
Physical address tag
Cache index
Byte offset
16
14
2
Valid
Tag
Data
Cache
=
32
Cache hit
Data

Virtual Memory Protection

- Each process/program has private virtual address space
  - Automatically protected from rogue programs
- Sharing is possible, necessary, desirable
  - Avoid copying, staleness issues, etc.
- Sharing in a controlled manner
  - Grant specific permissions
    - Read
    - Write
    - Execute
    - Any combination
  - Store permissions in PTE and TLB

Summary

- Memory hierarchy: Register file
  - Under compiler/programmer control
  - Complex register allocation algorithms to optimize utilization
- Memory hierarchy: Virtual Memory
  - Placement: fully flexible
  - Identification: through page table
  - Replacement: approximate LRU or LFU
  - Write policy: write-through

Summary

- Page tables
  - Forward page table
    - &PTE = PTBR + VPN * sizeof(PTE)
  - Multilevel page table
    - Tree structure enables more compact storage for sparsely populated address space
  - Inverted or hashed page table
    - Stores PTE for each real page instead of each virtual page
    - HPT size scales up with physical memory
  - Also used for protection, sharing at page level

Summary

- TLB
  - Special-purpose cache for PTEs
  - Often accessed in parallel with L1 cache
- Main memory design
  - Commodity DRAM chips
  - Wide design space for
    - Minimizing cost, latency
    - Maximizing bandwidth, storage

Thank You