Inf2C - Computer Systems
Lecture 17-18
Virtual Memory

Vijay Nagarajan

School of Informatics

University of Edinburgh

Previous lecture: Memory hierarchy

- Main idea: exploit locality in memory references to create the illusion of a fast & large memory
  - Two types of locality: temporal and spatial
- Memory hierarchy levels: registers, cache (≥1 levels), main memory, disk/SSD
- Cache: hardware-managed storage
  - Exploits temporal & spatial locality
  - Various degrees of associativity (complexity-performance trade-off)
  - Replacement policy

Lecture 17-18: Virtual memory

- Motivation
- Overview
- Address translation basics
- Making address translation fast with a TLB

Motivation

Virtual memory addresses two main problems:

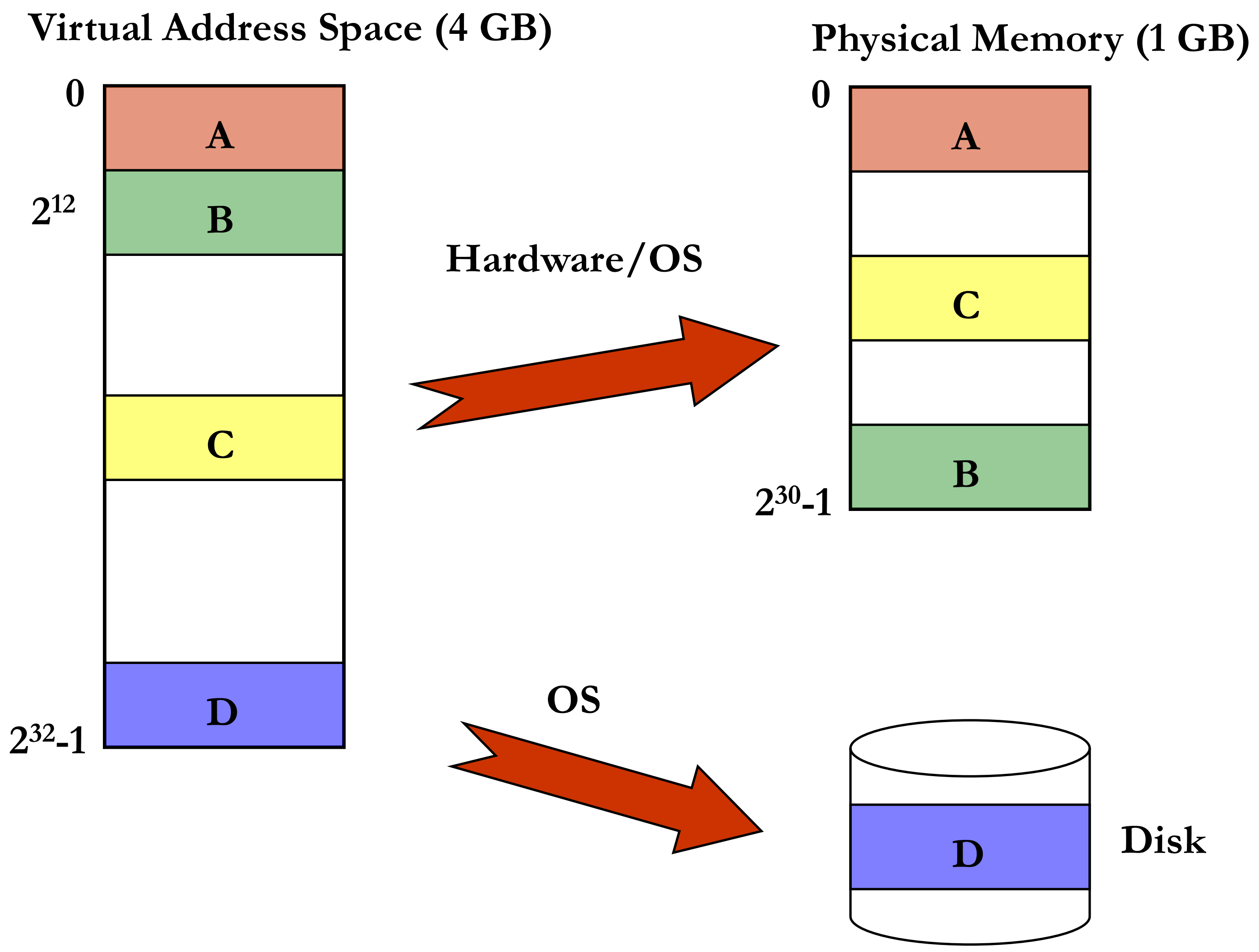
1) Capacity: how do we relieve the programmers/users from dealing with limited main memory?
   - Want to allow for the physical memory to be smaller than the program's address space (e.g., 32 bits → 4GB)
   - Want to allow multiple programs to share the limited physical memory with no human intervention

2) Safety: how do we allow for safe and efficient sharing of memory among multiple programs?
   - Want to prevent user programs from accessing the memory used by the OS
   - Want strict control of access by each user program to memory of other user programs

Virtual Memory

- Basic idea: each program thinks it owns the entire memory → the **virtual address space**
  - PC and load/store addresses are **virtual addresses**
- Actual main memory: **physical address space**
  - Virtual addresses are **translated** on-the-fly to physical addresses
  - Parts of virtual address space not recently used are stored on disk
- Address translation is done jointly by the OS and hardware

Address translation for a program



Physical memory as a cache for VM

- Virtual memory space can be larger than physical memory
  - Programmer always sees the full address space (MIPS: 2^{32} bytes)
- Physical memory is a cache for the virtual memory
  - Physical memory holds the currently used portions of a program's code and data (exploits locality!)
- Secondary storage (disk or SSD) "backs" the physical memory
  - OS reserves a portion of the disk for **swap space**
  - OS swaps portions of each process' code and data areas in & out of physical memory on demand (this is called **paging**)
  - Paging is transparent to the application and the programmer

Paging

- A “cache line” or “block” of VM is called a page
  - Simply “page” or “virtual page” for virtual memory
  - “Page frame” or “physical page” for physical memory
- Typical page sizes are 4-16 KB (can be higher in servers)
  - Large enough for efficient disk use and to keep translation tables (called page tables) small
- Mapping is done through a per-program **page table**
  - Allows control of which pages each program can access
  - Different programs can use same virtual addresses

Typical Virtual Memory Parameters

parameter	Cache	Physical Memory
size	4KB – 40MB	4MB – 4TB
block/page size	16 - 128 bytes	4KB (up to 4GB)
hit time	2-40 cycles	100 - 400 cycles
miss penalty	8 - 400 cycles	1M - 10M cycles
miss rate	0.1 - 10%	0.00001 - 0.001%

Modified from H&P 5/e Fig. 5.35

Page fault occurs when a valid virtual address is not in memory

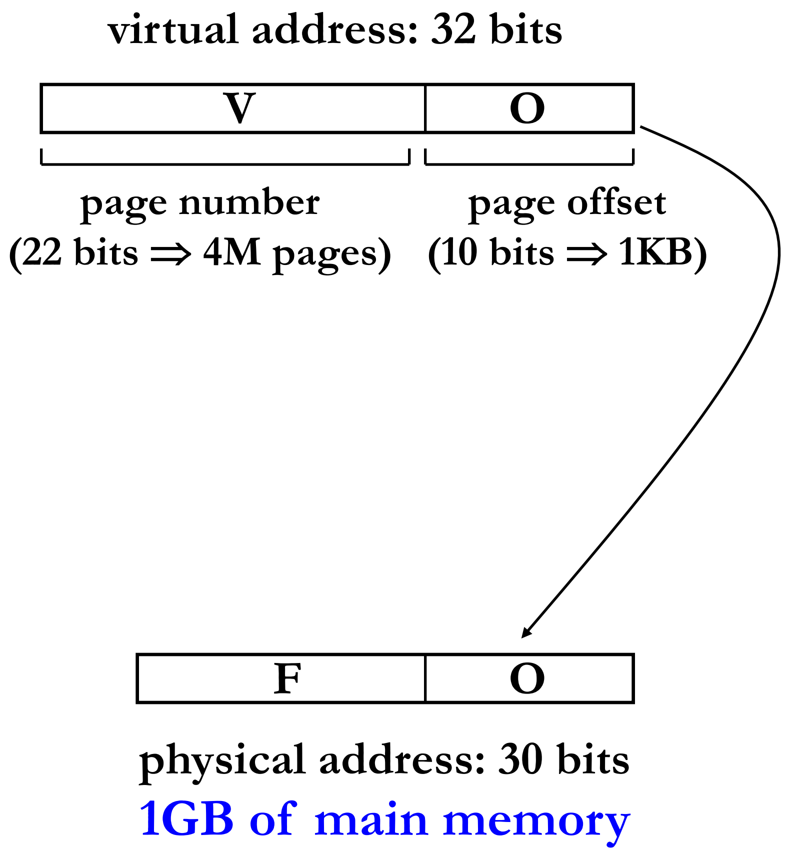
Address Translation

Example: 1KB pages

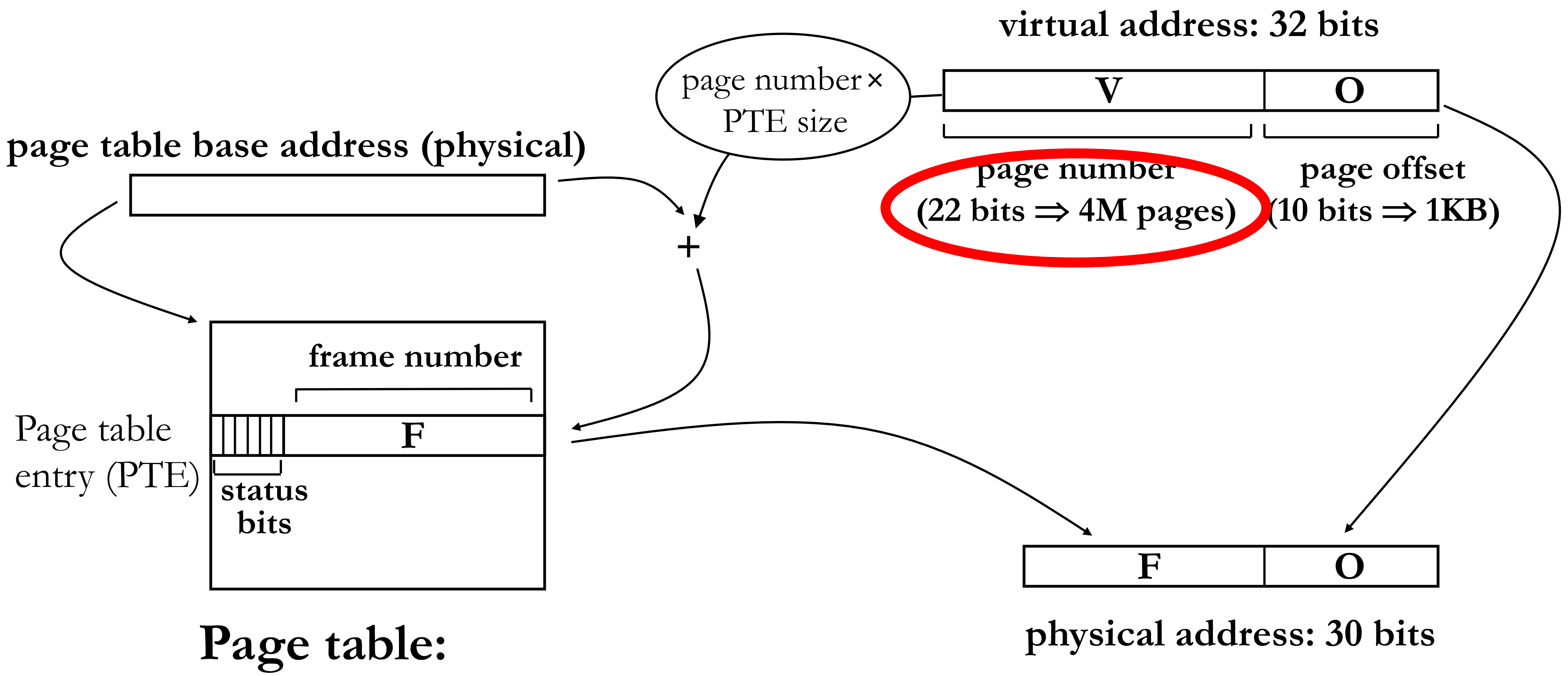
Need:

- A mapping from virtual (V) to physical (F) page numbers
- Page offset not translated
- Must be efficient (in time and space)

Solution: **Page Table**



Address Translation



Page table:

- per program
- one entry per page (e.g. 4M entries)
- located in the system portion of main memory

Practice problem

What is the size of the page table given a 32-bit virtual address space, 4 KB physical pages, and 1 GB of main memory?

Moving pages to/from memory

- Pages are allocated on demand
  - E.g., program launch (results in pages allocated for code, data, and stack); malloc (heap space)
- Pages are replaced and swapped to disk when system runs out of free page frames
  - Aim to replace pages not recently used (principle of locality). **A**(ccess) bit for a page is set whenever page is accessed and is reset periodically
  - If any data in page has been modified, the page must be written back to disk: **M**(odified) bit in status bits is set
- Access to a swapped-out page causes a **page fault** which invokes the OS through the interrupt mechanism
  - **R**(esidence) bit in page table to indicate if page is in memory (R=1)

Providing Protection

- Each page table entry can have permission bits that control whether
  - the process is allowed to access a page
  - read & write (W), read-only (R) or execute-only (E) access is allowed
- This enables per-process memory protection
  - Can set up private and shared areas
  - Only pages with code can be executed (for security)
- Important that only OS can change page tables

Fast address translation

- Problem: page table accesses add latency to each memory access
  - Two memory accesses per load or store (1 to get the page table entry + 1 for the actual load/store)
- Fast address translation: **Translation Lookaside Buffer (TLB)**
  - A cache of page table entries
    - Each TLB entry holds translation information, not program data
    - Tag: virtual page number. Entry: physical page number
  - Small and fast table in hardware, located close to processor
  - Can capture most translations due to principle of locality
  - When page not in TLB: access the page table and save the translation entry in TLB

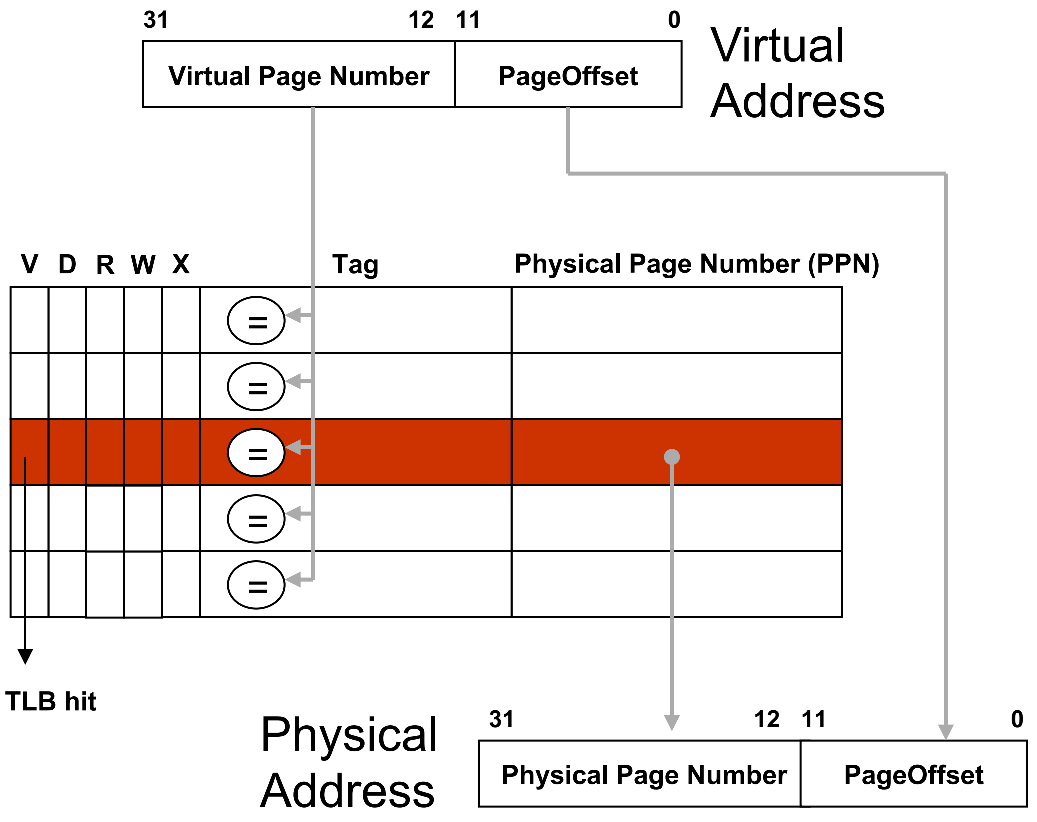
Translation Look-aside Buffer (TLB)

TLB: a small, fully-associative cache of page table entries

- Accessed with Virtual Page Number (VPN)
- Each entry stores the translation (PPN) for a given VPN
- Physical address formed from PPN and Page Offset
- Page table accessed on a TLB miss

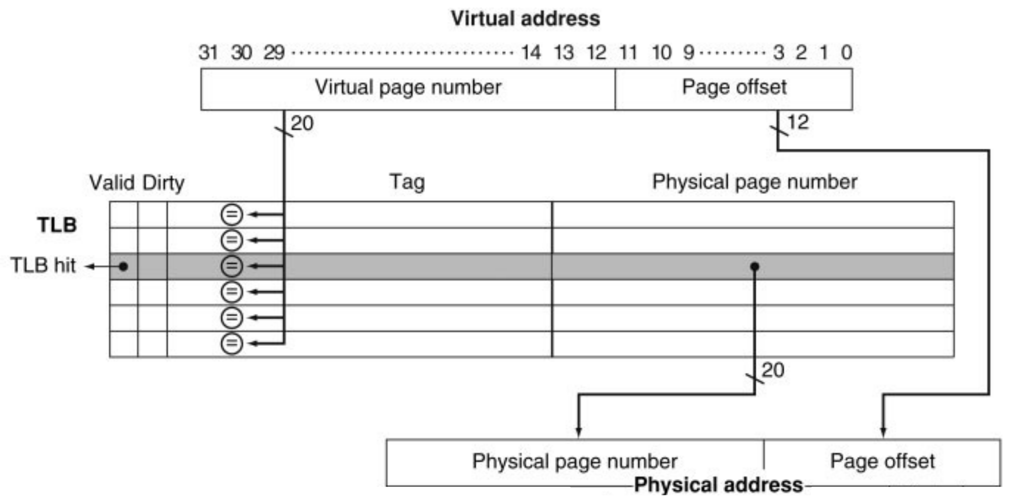
TLB status bits:

- V (valid) bit indicates a valid entry
- D (dirty) bit indicates whether page has been modified
- R, W, X permission bits. Checked on every memory access

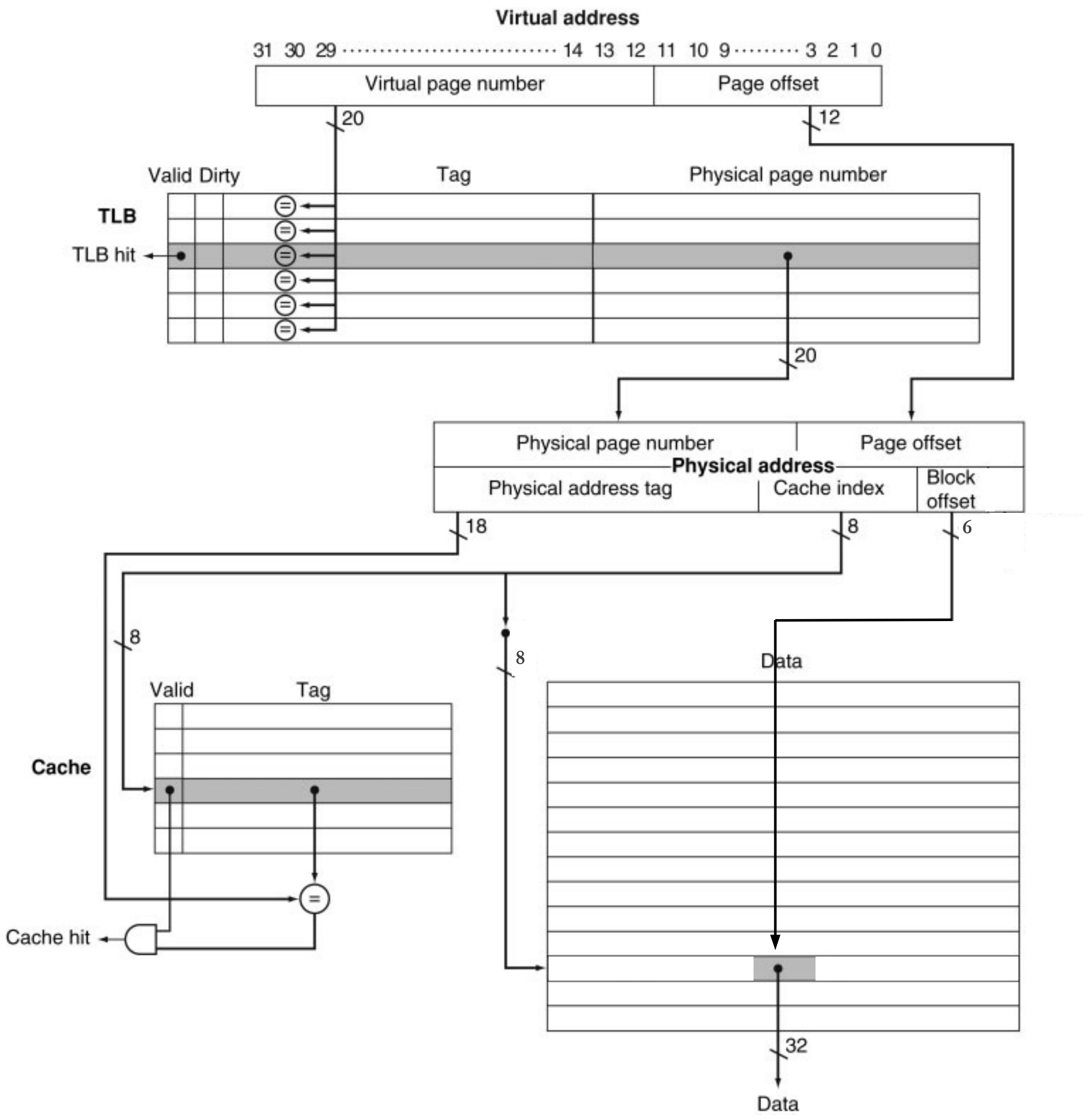


Note: physical address is always 32 bits, regardless of actual physical memory size

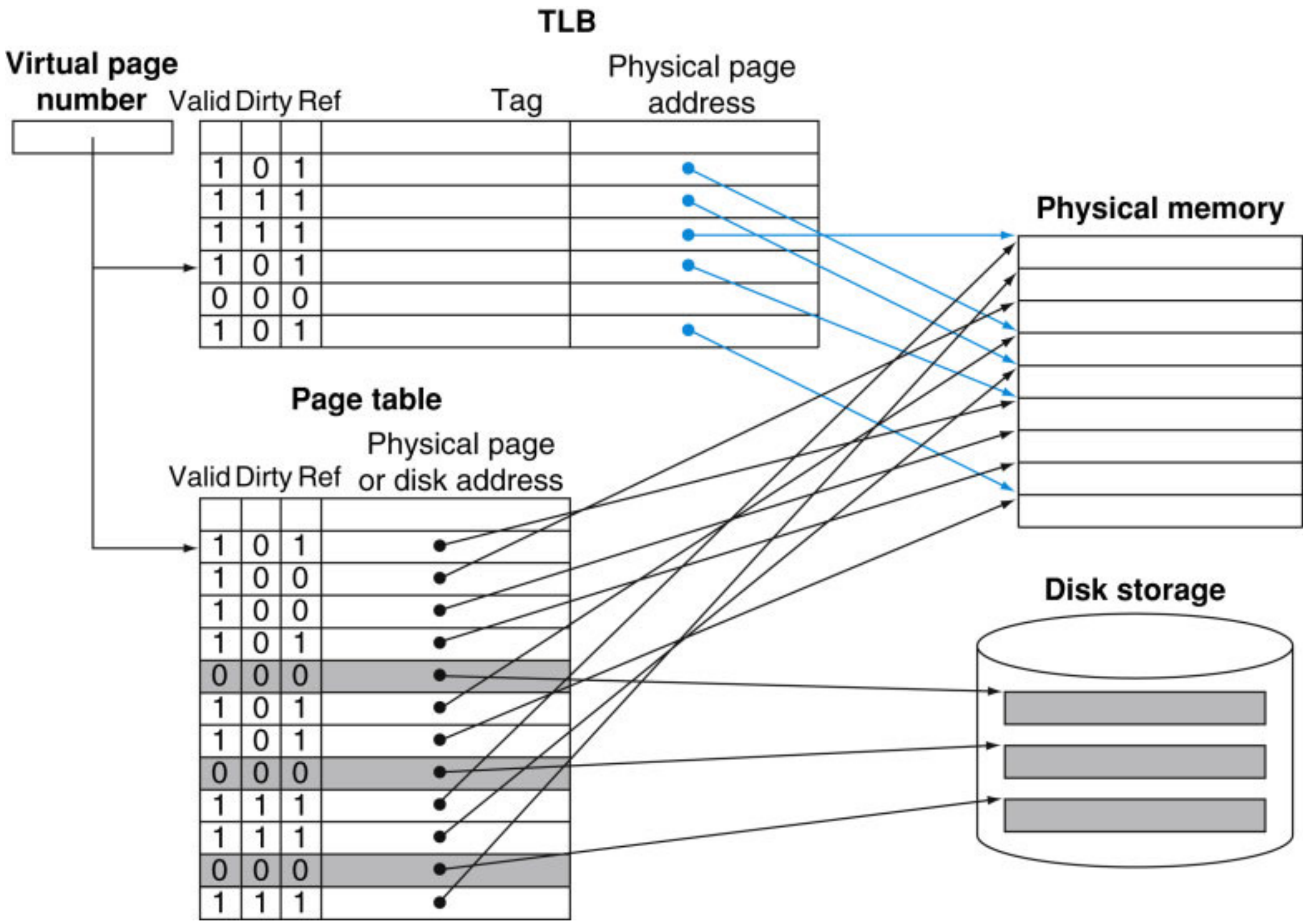
Integrating a TLB with the Cache



Integrating a TLB with the Cache



Virtual Memory: full picture


TLB
Virtual page number
Valid Dirty Ref
Tag
Physical page address
1 0 1
1 1 1
1 1 1
1 0 1
0 0 0
1 0 1
Physical memory
Page table
Valid Dirty Ref
Physical page or disk address
1 0 1
1 0 0
1 0 0
1 0 1
0 0 0
1 0 1
1 0 1
0 0 0
1 1 1
1 1 1
0 0 0
1 1 1
Disk storage