Lecture 13: Virtual memory

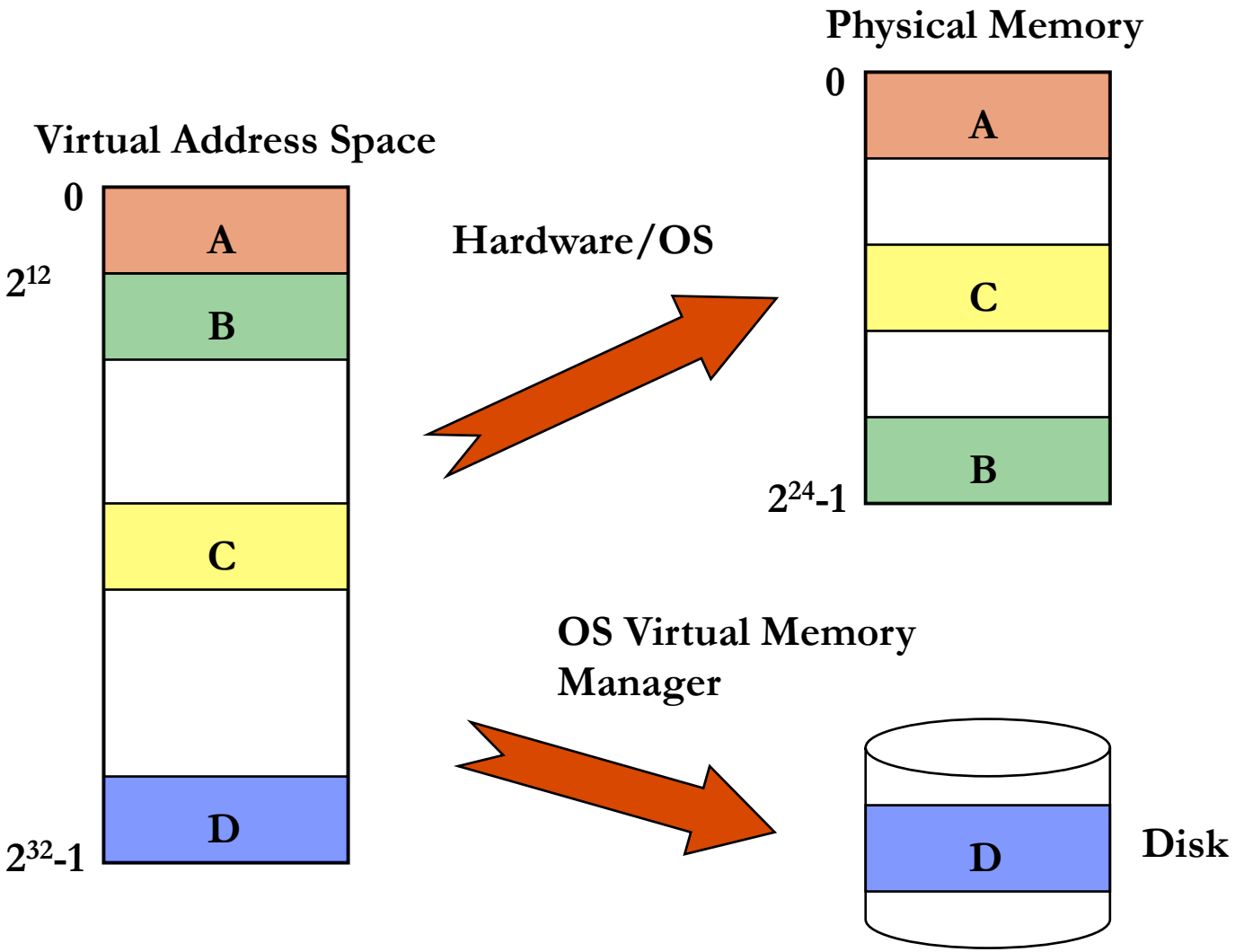
- Motivation
- Overview
- Address translation
- Page replacement
- Fast translation – TLB

Motivation

Virtual memory addresses two main problems:

- How do we remove burden of programmers dealing with limited main memory?
  - Want to allow for the total memory required for all programs being larger than the physical memory
  - Want to avoid changing addresses in programs to make them fit simultaneously in memory
- How do we allow efficient and safe sharing of memory among multiple programs?
  - Want user programs to not have access to data and program memory used by kernel
  - Want strict control of access by each user program to memory of other user programs

Address translation for 1 process



Virtual Memory

- Processor uses virtual address space
  - PC and other CPU registers all hold virtual addresses
- Actual physical memory: physical address space
- Virtual addresses are translated on-the-fly to physical addresses
- Dynamic address translation is done by combination of hardware (the memory management unit or MMU) and the OS

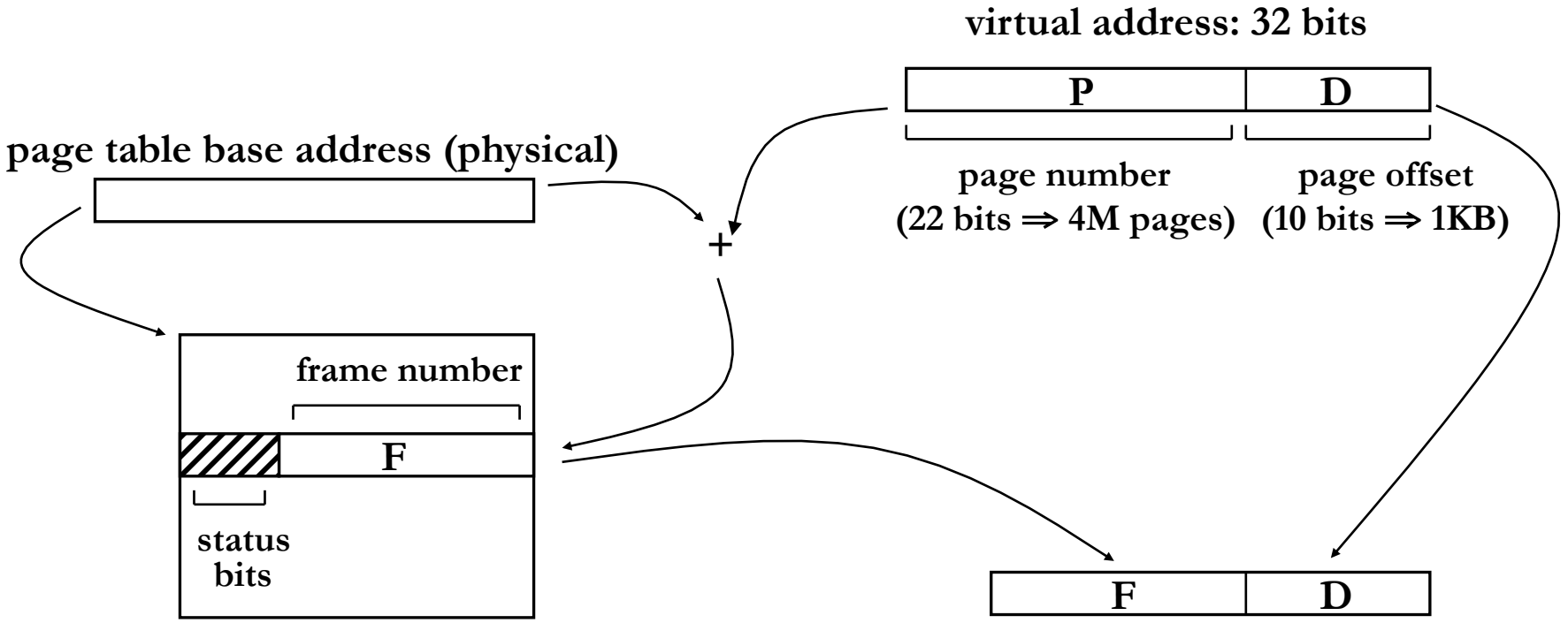
Physical memory as cache for VM

- Virtual memory space can be larger than physical memory
- Secondary storage (usually magnetic disks) is used as another level in the memory hierarchy
- Physical memory used as a cache for the virtual memory
  - Physical memory holds the currently used portions of a process's code and data areas
  - Full memory space used by process kept in swap-space area on disk
  - OS swaps portions of each process's code and data areas in and out of physical memory on demand
  - Swapping is transparent to the programmer

Paging

- A "cache line" or "block" of VM is called a page
  - Plain "page" or virtual page for virtual memory
  - "Page frame" or "physical page" for physical memory
- Typical sizes are 1 – 64 KB
  - Large enough for efficient disk use and to keep translation tables small
- Mapping is done through a per-process page table
  - Different processes can use same virtual addresses
  - Allows control of which pages each process can access

Dynamic Address Translation



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

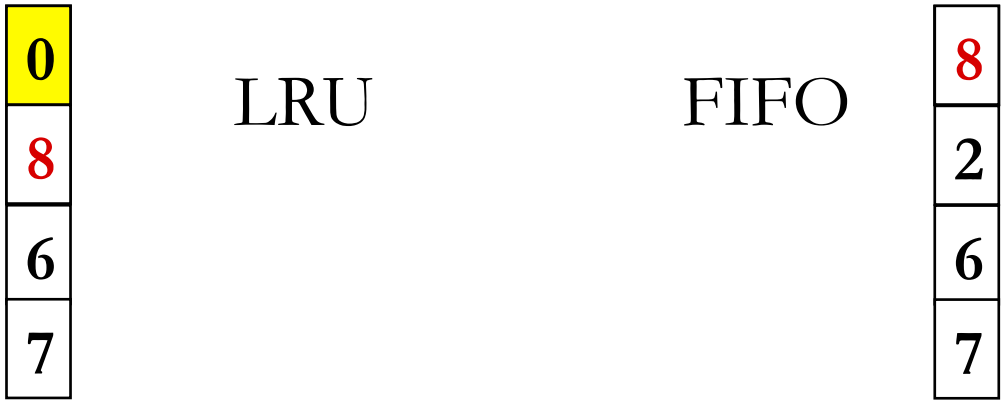
physical address: 26 bits
(64MB of memory)

Moving pages to/from memory

- Access to a non-allocated page causes a page-fault which invokes the OS through the interrupt mechanism
  - **R**(esidence) bit in page table status bits is zero
- Pages are allocated on demand
- 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

Page replacement

- Least Recently Used – outlined previously
  - Use past behaviour to predict future
- FIFO – replace in same order as filled
  - Simpler to implement
- Example: page references: 0 2 6 0 7 8
  - Physical memory 4 frames



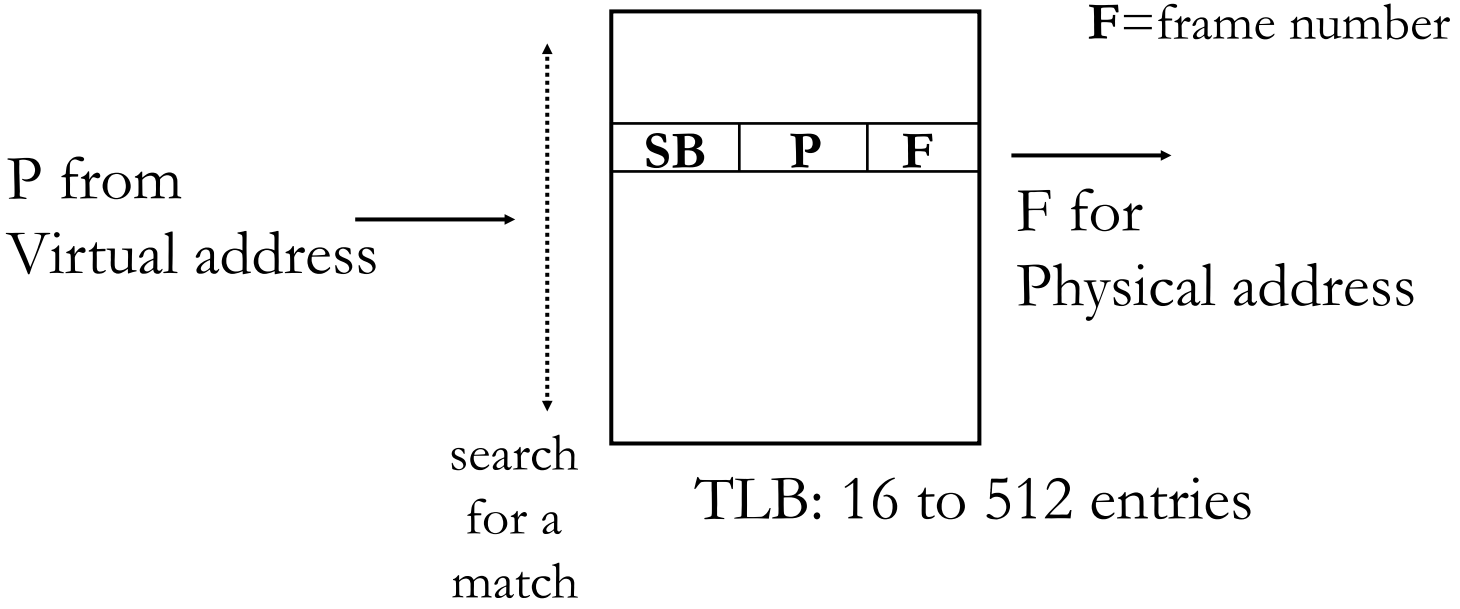
Translation Lookaside Buffer

- Page table is costly
  - Two memory accesses per load and store (1 to get the page table entry + 1 to get the data)
- Fast address translation: Translation Lookaside Buffer (TLB) contained in the MMU
  - Small and fast table in hardware, located close to processor
  - Is a cache for page table: holds frame numbers, not data
  - Can capture most translations due to principle of locality
  - One for all processes → must be invalidated on context switches
  - When page not in TLB, check page table, and save new entry in TLB

Translation Lookaside Buffer

Fully associative memory:

SB=status bits
P=page number
F=frame number



Partially (set) associative memory also used