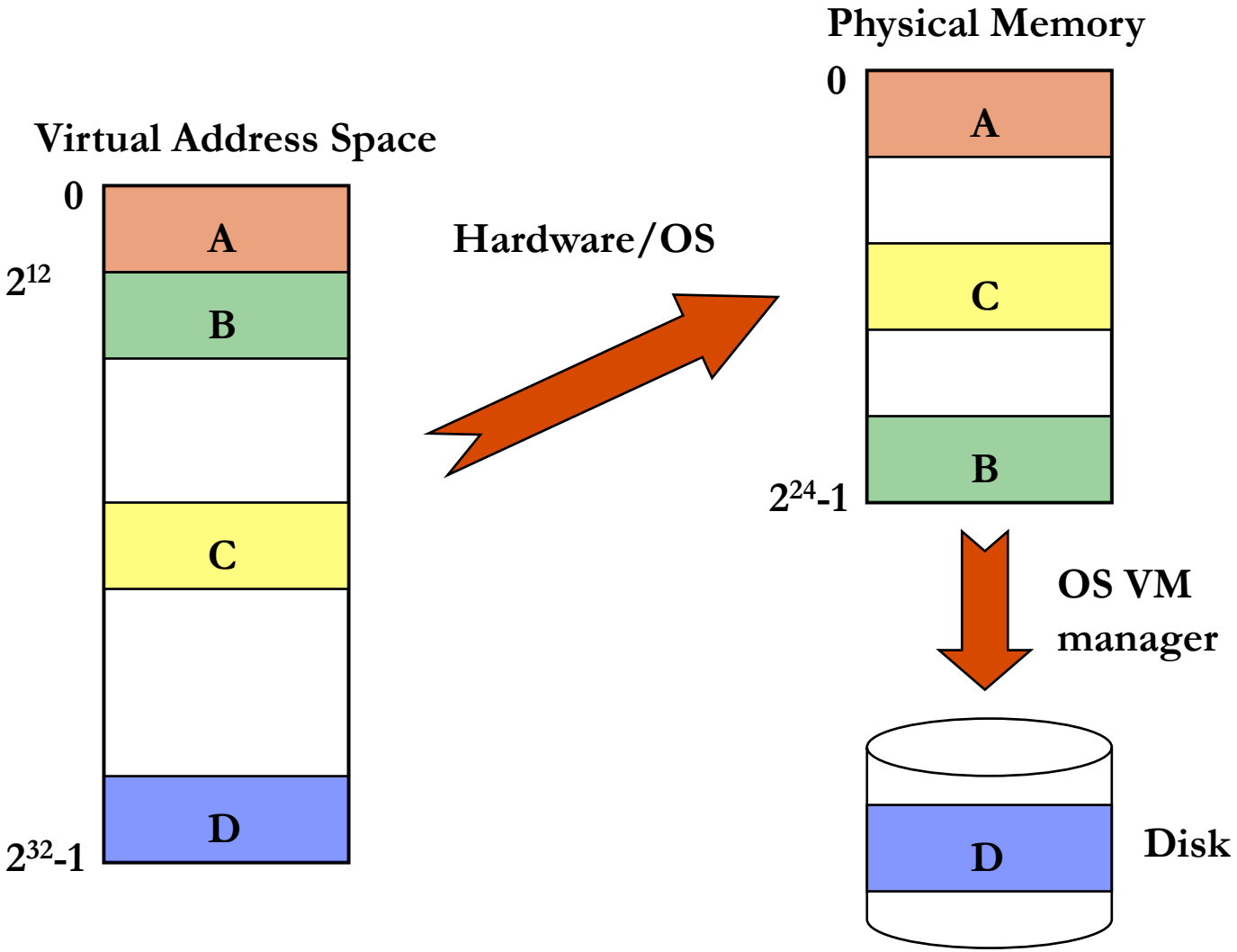
Lecture 13: Virtual memory

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

Motivation

- Problems:
  - Where in (physical) memory will the program be located?
    - What happens to the addresses in the program?
  - How to protect each process' memory space from the others?
  - What if the memory is not large enough?
  - What happens if the memory is upgraded?
- Solution: Virtual Memory
  - Decouple the memory visible to the processes from the real physical memory in the machine
  - Transparent translation between the two views of memory
  - Secondary storage (disk) used as part of the memory hierarchy

Virtual Address vs. Physical Address



Virtual Memory

- Processor uses virtual address space
  - PC and other regs 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 the memory management unit (MMU), a hardware unit
  - Before or after accessing the cache

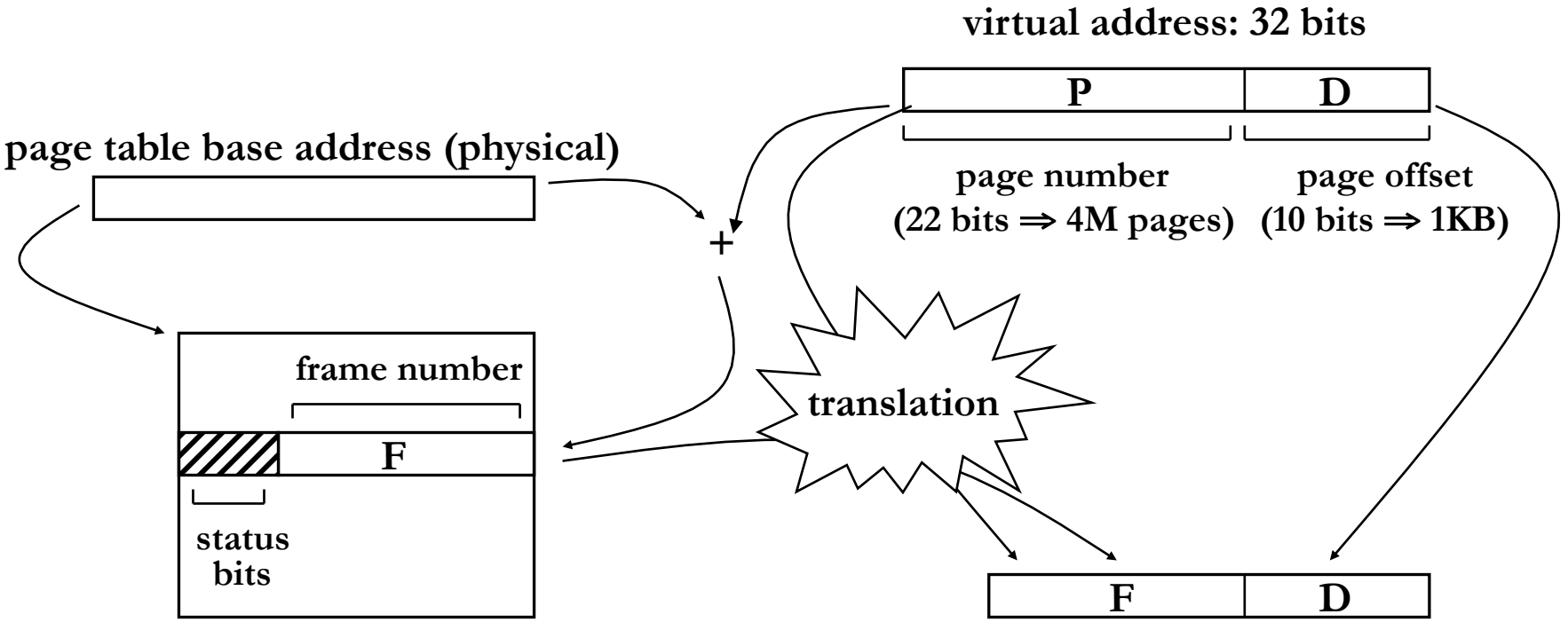
Main memory as cache for VM

- Virtual memory space can be larger than physical memory
- Secondary storage is used as another level in the memory hierarchy
- Main memory used as a cache for the virtual memory
  - Only keeps the currently used portions of the process' code and data areas; the rest stays on disk
  - OS swaps portions of process' code and data areas in and out of memory on demand
  - This is transparent to the programmer

Paging

- The "cache line" of VM is called a page
  - Plain "page" for virtual memory
  - Page frame for physical memory
- Typical sizes are 1KB to 16KB
  - Large enough for efficient disk use and to keep translation tables small
- Mapping is done through a per-process page table

Dynamic Address Translation



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

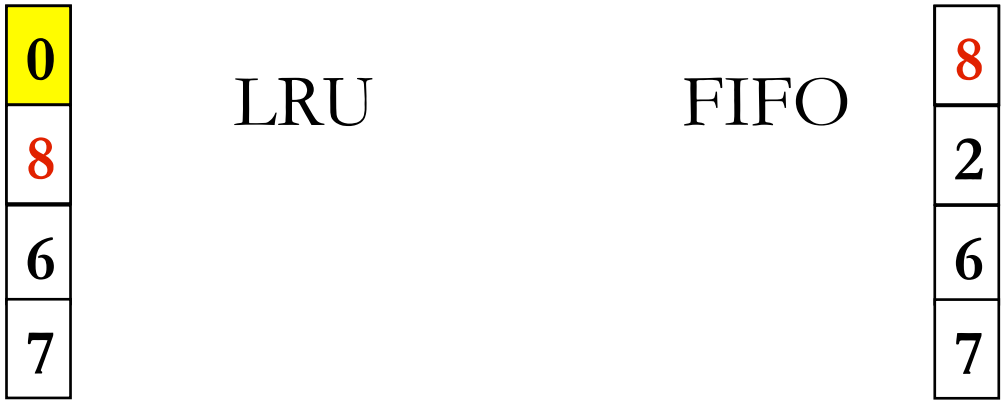
physical address: 26 bits
(64MB of memory)

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
  - Priority given to pages not recently used (principle of locality): **A**(ccess) bit in status bits zero (**A** is set whenever process access the page and 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 one

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



Implementing address translation

- Page table inefficiencies:
  - Two memory accesses per load and store (1 to get the page table entry + 1 to get the data)
  - Page table too large:

Virtual address: 32 bits
Physical address: 26 bits
Pages: 1KB → 10 bits

4M pages → 4M entries
16 bits per entry

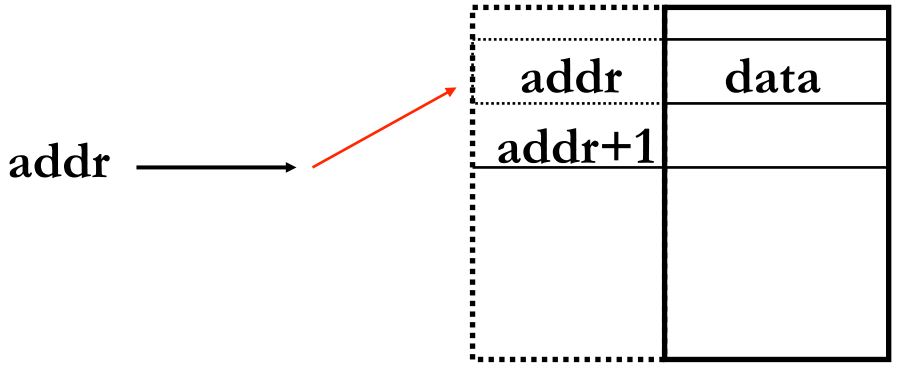
8MB of page table per process!

Translation Lookaside Buffer

- Fast address translation: Translation Lookaside Buffer (TLB) contained in the MMU
  - Small and fast table in hardware, located close to processor
  - 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

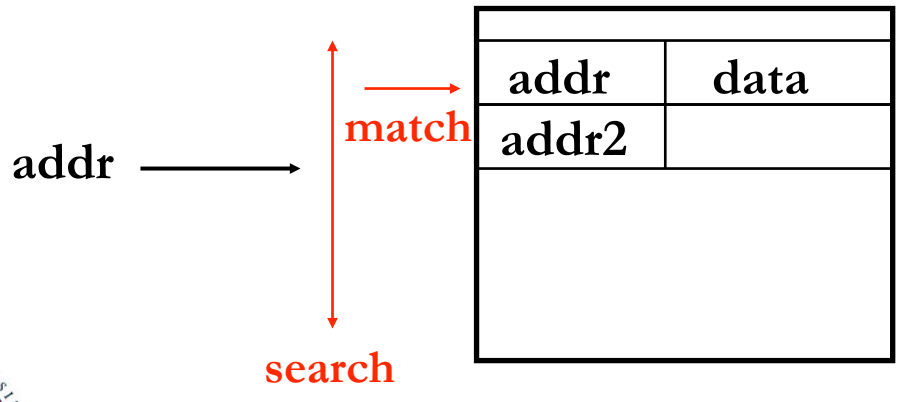
Associative Memory

- Ordinary memory



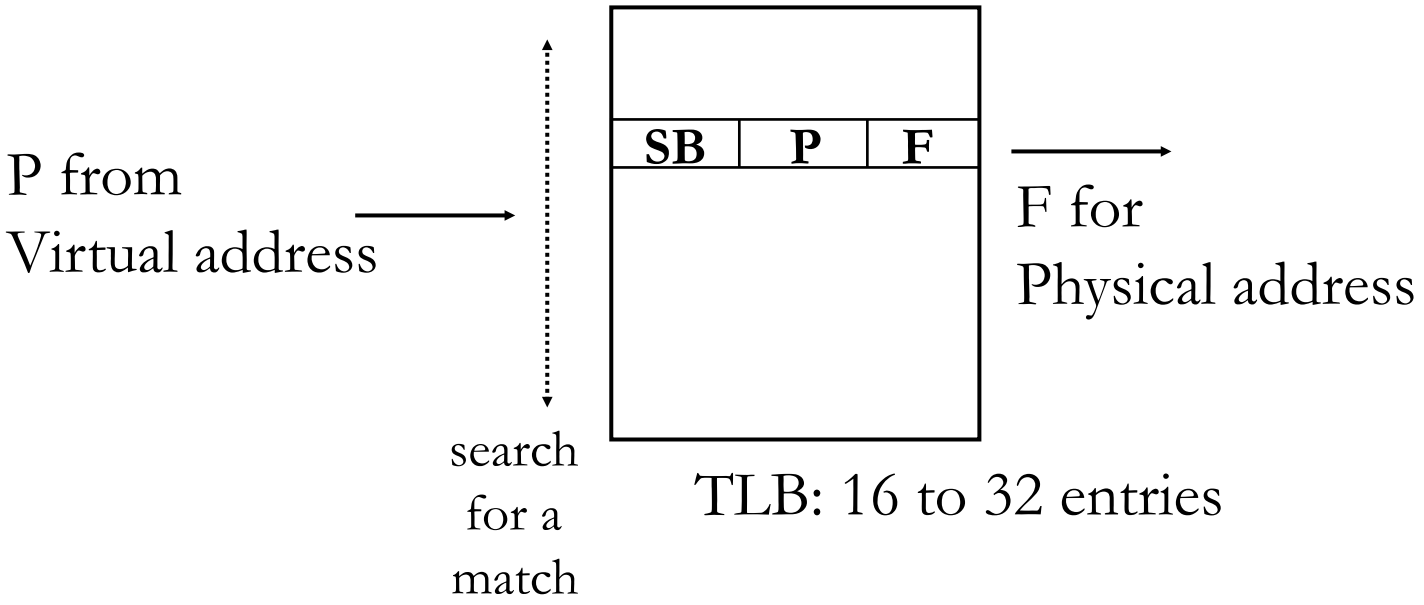
• addr has a single possible location in memory

- Associative memory



• addr can be in any of several locations in memory
• "search" is usually done by a parallel match
• addr is normally called tag

Translation Lookaside Buffer



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