Inf2C - Computer Systems
Lecture 14
Virtual Memory

Boris Grot

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
  - Temporal vs spatial locality
- Memory hierarchy levels: registers, cache (≥1 levels), main memory, disk
- Cache: hardware-managed storage
  - Exploits temporal & spatial locality
  - Fully-associative vs direct mapped

Lecture 14: Virtual memory

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

Motivation

Virtual memory addresses two main problems:

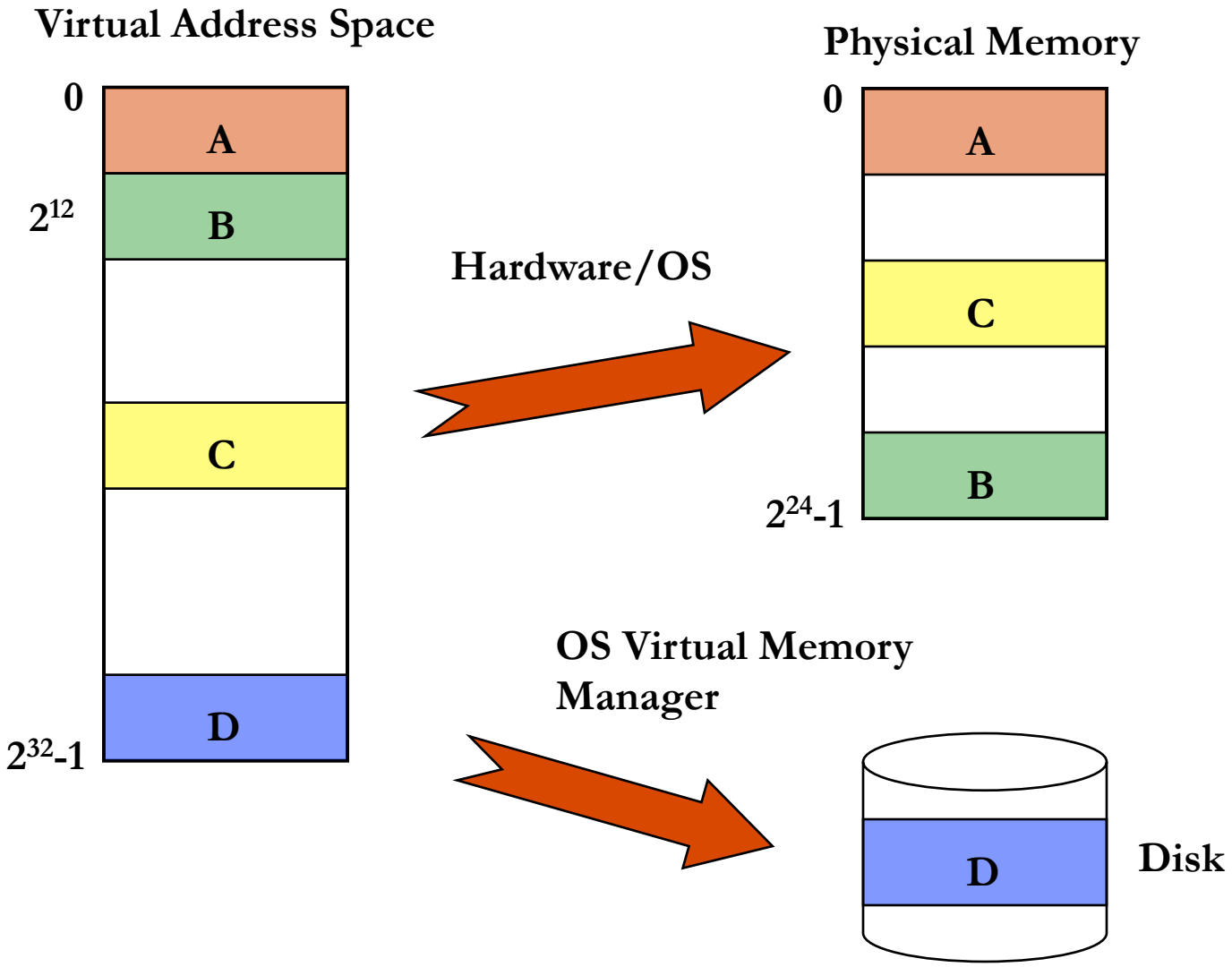
1) Capacity: how do we remove burden of programmers dealing with limited main memory?
   - Want to allow for the physical memory to be smaller than the program's address space (e.g., 32 bits → 4GB)
   - Want to allow multiple programs to share the limited physical memory with no human intervention

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
- Dynamic address translation is done by combination of hardware and the OS

Address translation for 1 process



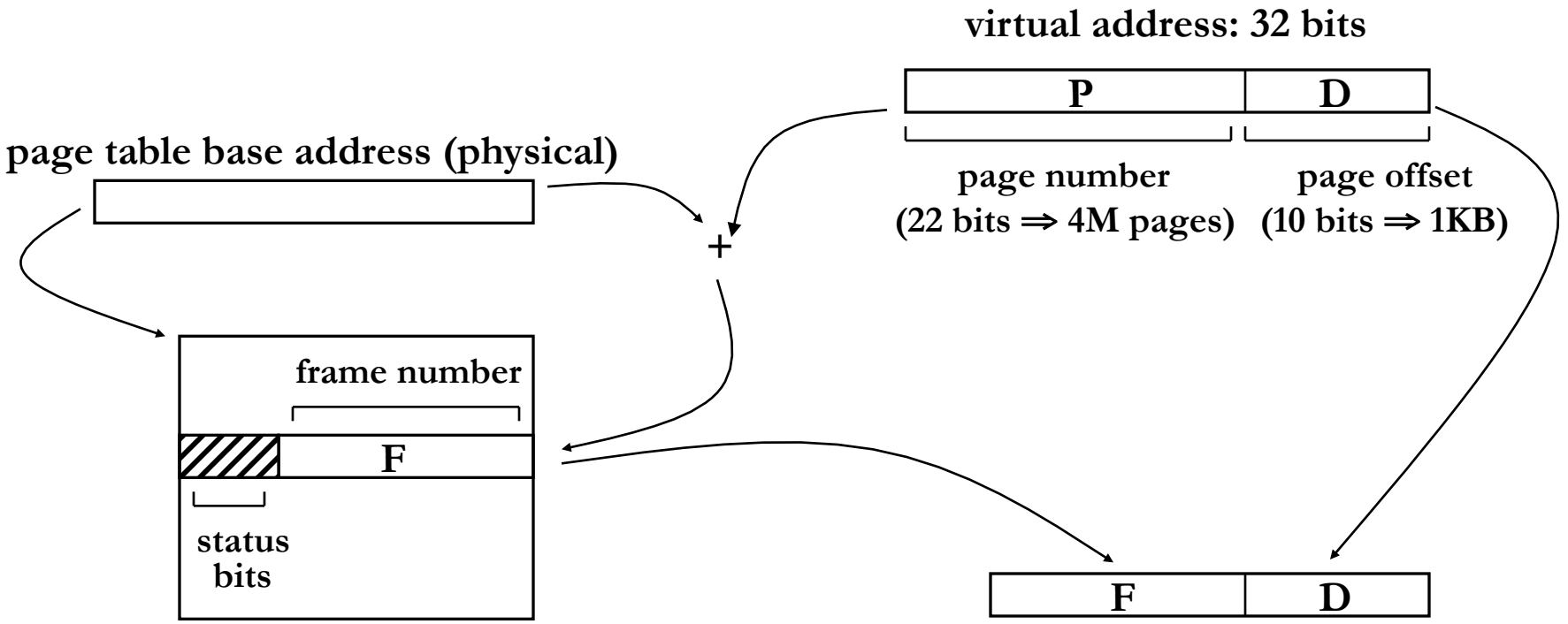
Physical memory as cache for VM

- Virtual memory space can be larger than physical memory
  - Programmer always sees the full address space (MIPS: 2^{32} bytes)
- Physical memory used as a cache for the virtual memory
  - Physical memory holds the currently used portions of a process' code and data areas (exploits locality!)
- Secondary storage (usually disk) "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 and out of physical memory on demand (process called paging)
  - 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 4-8 KB (MB or GB in servers)
  - Large enough for efficient disk use and to keep translation tables small
- Mapping is done through a per-process page table
  - Allows control of which pages each process can access
  - Different processes can use same virtual addresses

Dynamic Address Translation



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

physical address: 30 bits
(1GB of main 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

Providing Protection

- Each page table entry can have permission bits controlling
  - access allowed or not
  - read & write, read only or execute only access
- This enables per-process memory protection
  - E.g. can set up private and shared areas
- Important that only OS can change page tables
  - One of motivations for having kernel & user modes

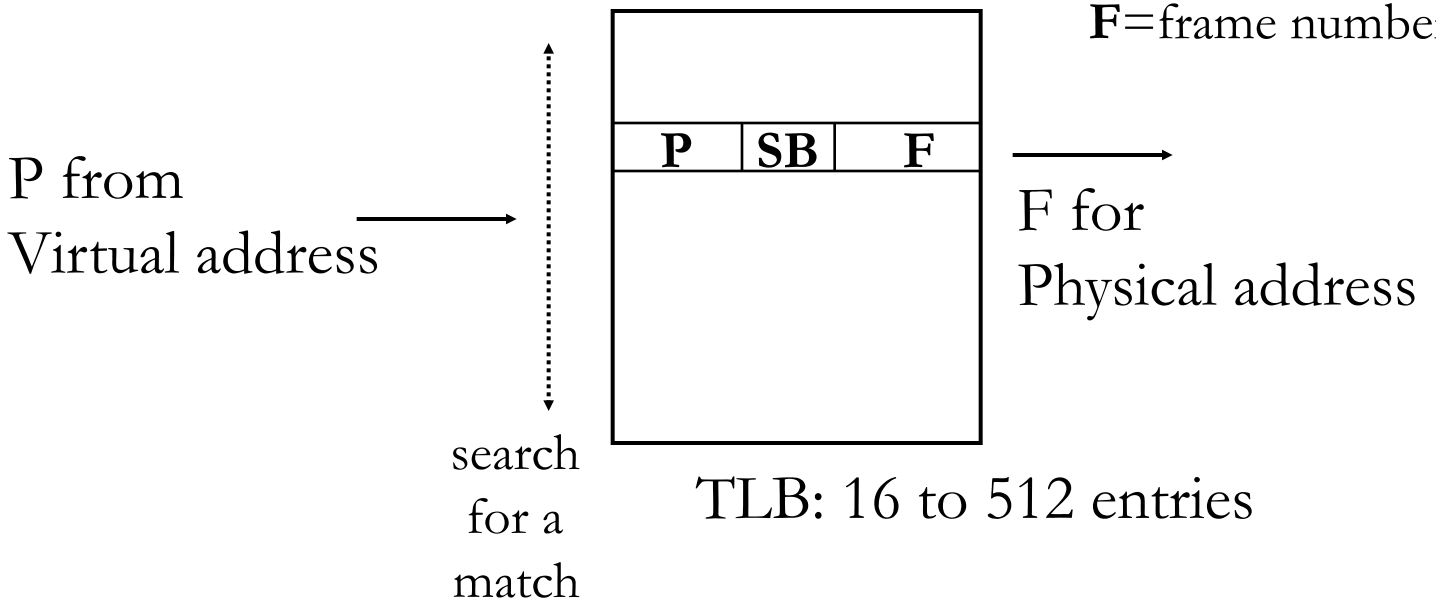
Translation Lookaside Buffer

- Accessing the page table is costly in terms of latency
  - 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 addr., not program data
    - Tag: virtual address. Entry: physical frame address
  - 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 Lookaside Buffer

Fully associative memory:

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



Virtual Memory: full picture