Lecture 13 – Virtual Memory Systems

Karl R. Wilcox
Karl@cs.rhul.ac.uk

Objectives

- In this lecture we will cover
  - Virtual Memory Systems
  - The hardware for implementing virtual memory

(Diagrams from Patterson & Hennessy)

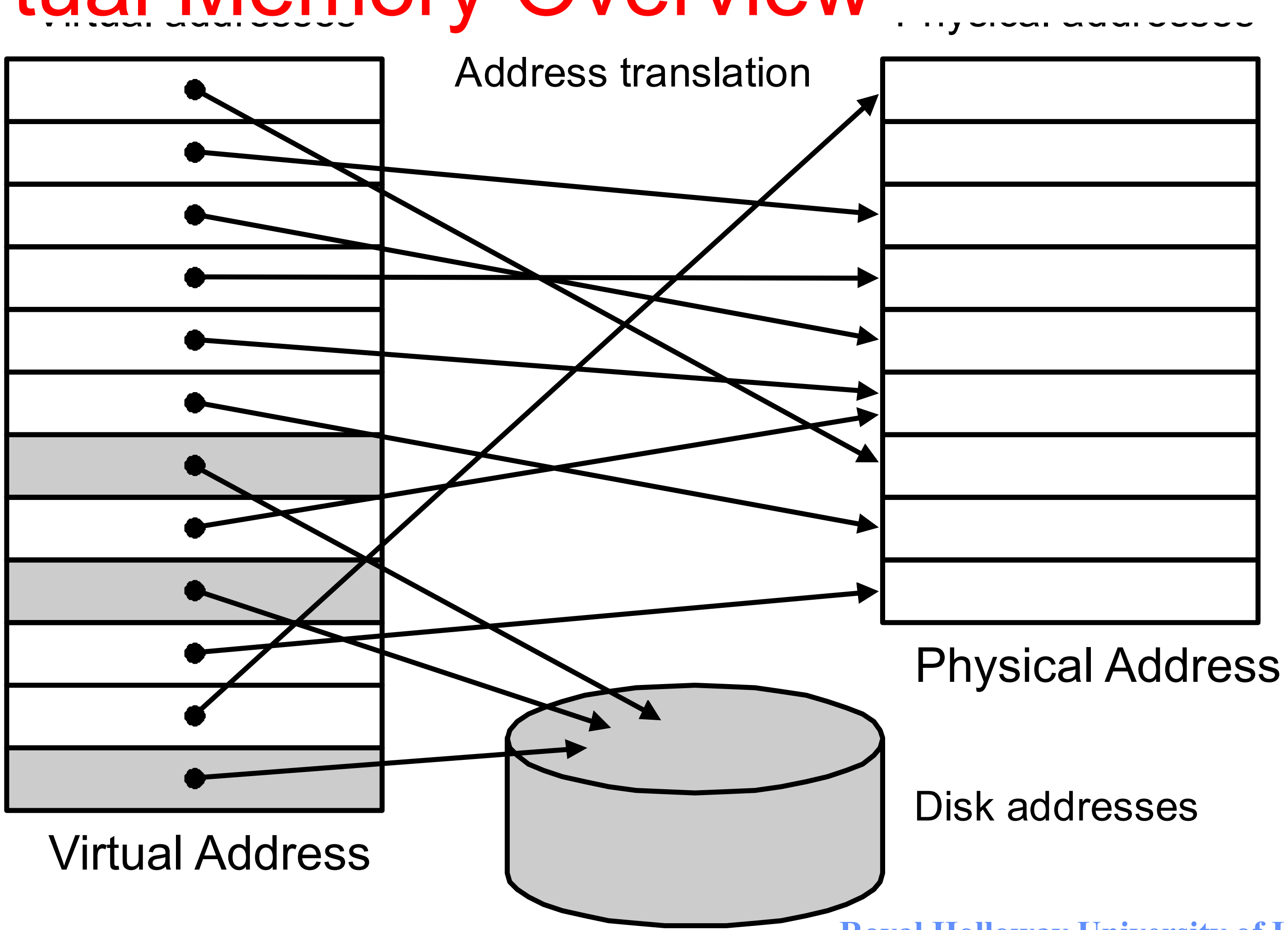
Caching Vs Virtual Memory

- Caching works over the space of a few thousand instructions
  - I.e. “within” the quantum of time that a process gets on the processor
- Virtual Memory operates on entire processes
  - Which pages of a process should be in memory, ready to run
- Caching and virtual memory are independent
  - But both increase processor performance

Points To Note

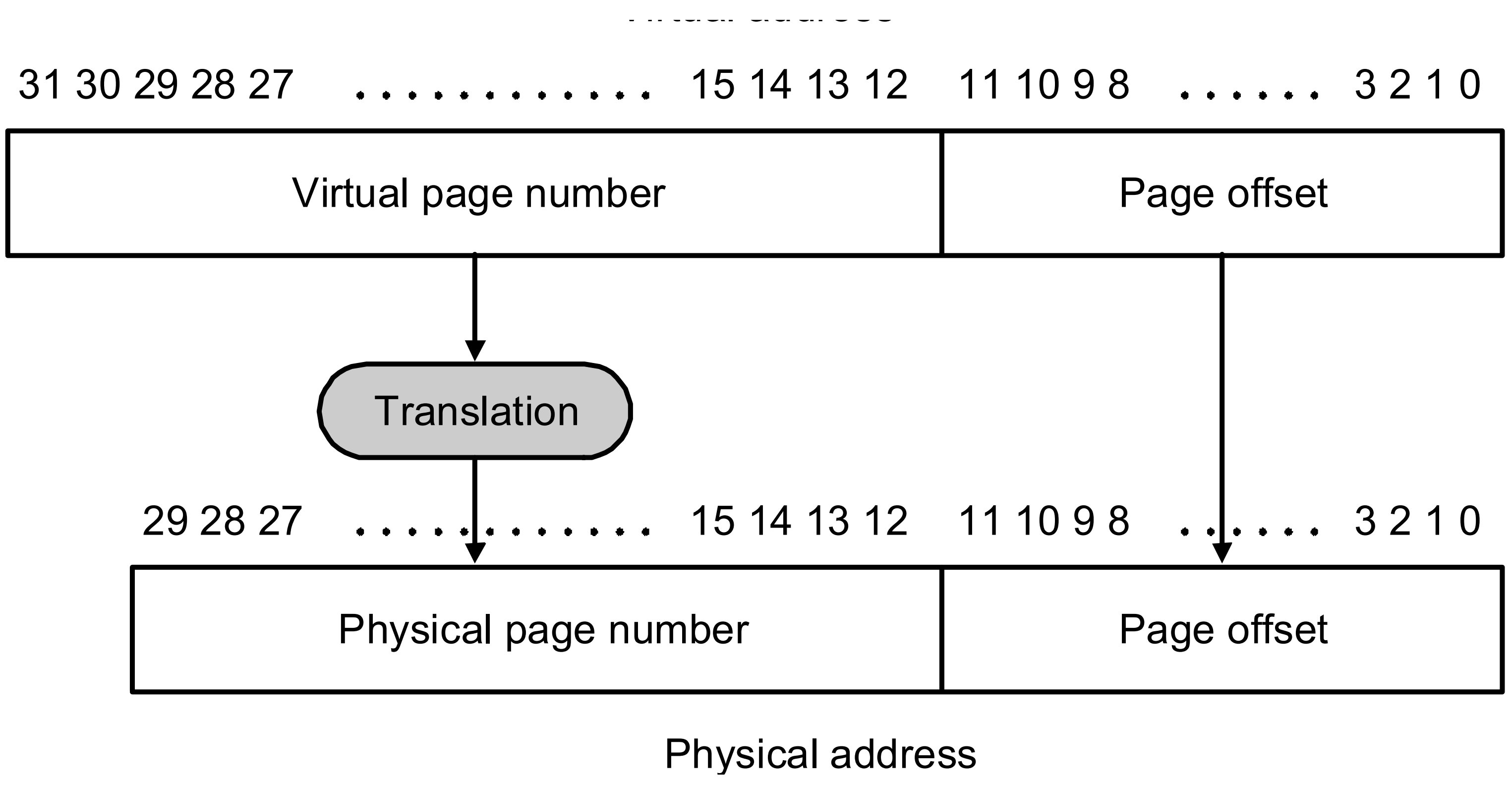
- We use virtual memory to:
  - Allow more than one process to co-exist
  - Provide an address space larger than physical memory
  - (provide protection between processes)
- Different terminology to CPU caches
  - Data is units of a page, that goes into a page frame
  - A page not found is known as “page fault”
- Page faults are very costly (in CPU time)

Virtual Memory Overview

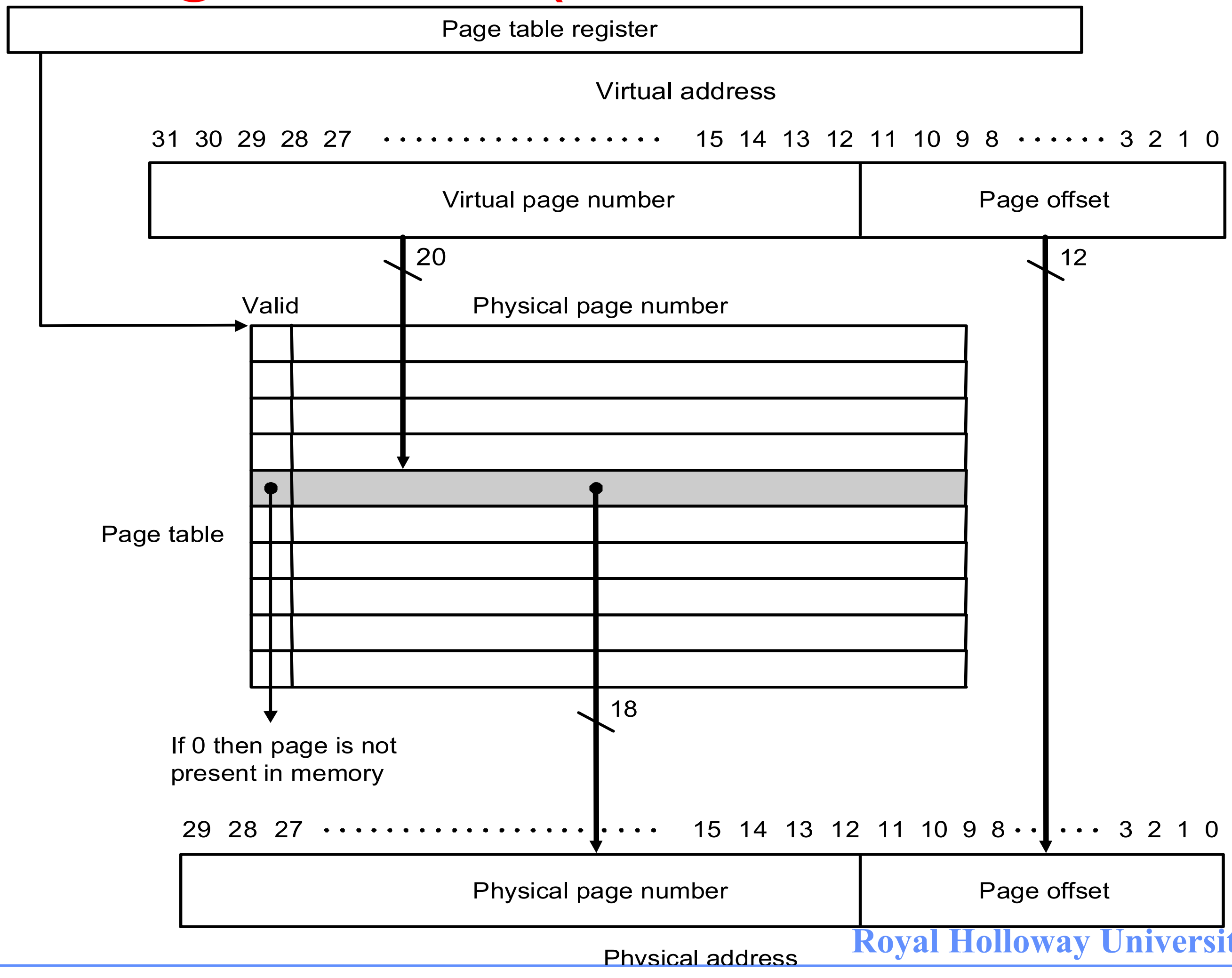


Royal Holloway University of London

Address Translation



The Page Table (One Per Process)



Page Table Problems

- Speed of access (multiple memory accesses per item)
  - Additional page table register
- One per process, can become very large
  - Maximum memory size / page size
- Possible solutions
  - Could restrict number of pages (limit register)
  - Hashing function to page address
  - Multiple levels of page tables
  - In reality, page tables can be paged (!)

Writing Data

- "Write Through" would not work
  - (updating both cached and on-disk versions)
  - Writing is a very slow operation
- Always implemented as "write back"
  - (write page to disk when replaced)
  - Has a "dirty bit" set on page update
  - Can be in hardware or software

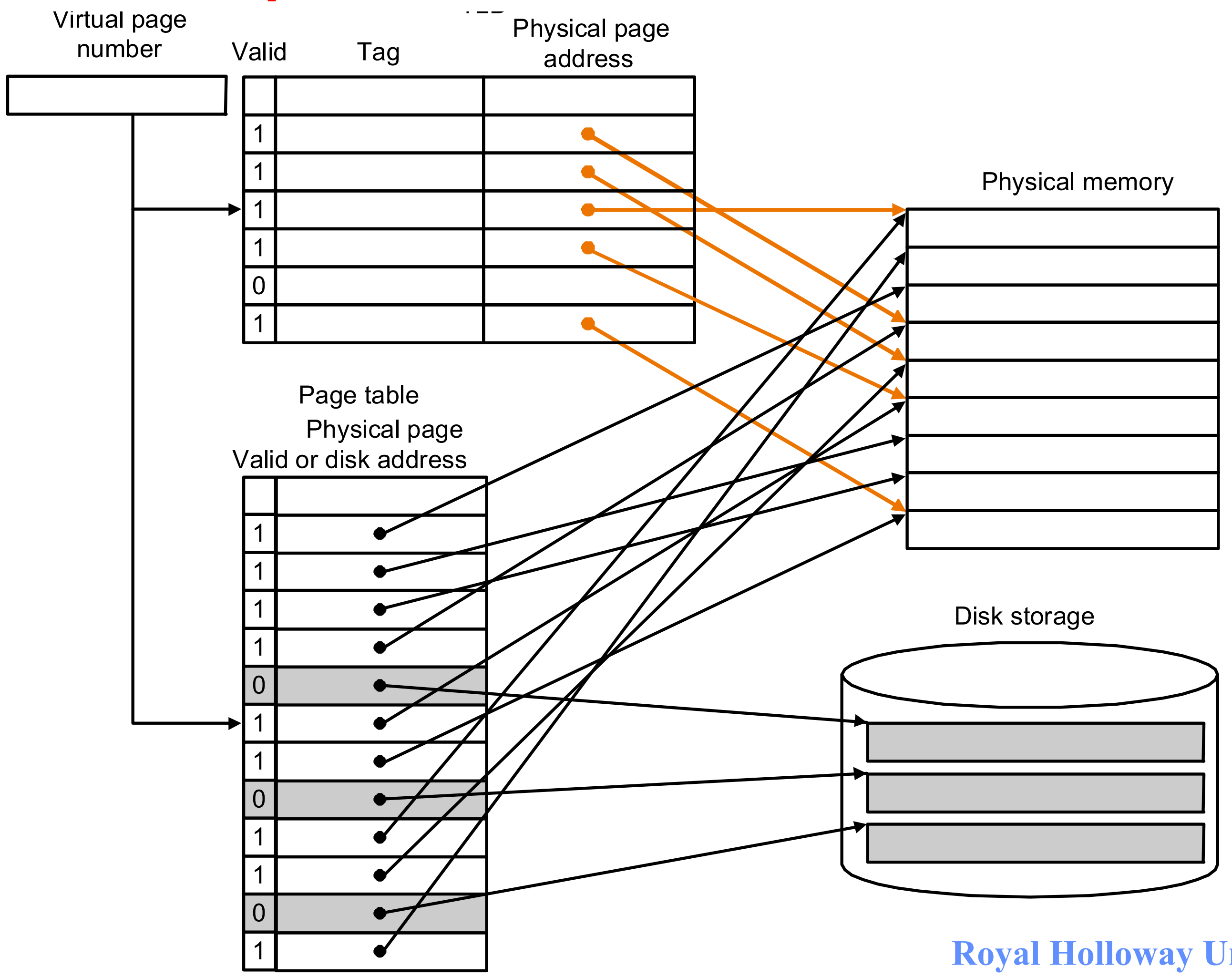
Page Replacement

- Various strategies (See CS263 – Operating Systems)
- Least Recently Used (LRU) is common
- Hardware support for recording page accesses
  - Not true LRU (which is expensive)
  - Has an access bit cleared by a clock tick

The Translation Lookaside Buffer

- Known as a TLB
- Hardware lookup of (a subset of) virtual page to physical address mapping
  - During instruction execution cycle
- Effectively a cache of the page table
  - Only caches pages in memory

TLB Implementation



Memory Protection

- **Pages can be made "read only"**
  - Hardware support for a read only bit
  - Exception generated on violation
- **Page tables must <u>not</u> be writable by the user**
  - Could read / write any pages (of any user)
  - Page tables are in operating system memory space
- **However, pages can be shared between processes**
  - For shared data
  - Or shared code (run time shared libraries / DLLs)
  - Page appears in > 1 page table

Handling Page Faults

- Page fault handling quite complicated
- Hardware generates an exception
- “offending” address stored in EPC
- Current process suspended & exception handler is invoked
- If instruction miss, need page with EPC address
- If data miss, must read (& decode) instruction at EPC to find required address
- Start disk request for reading of page
  - Hand control to scheduler
  - Disk read completes, original process now runnable

Summary

- We have looked at virtual memory schemes
- And the hardware to implement them
  - Page table register
  - Page information (dirty bit / LRU bits / read only)
  - Translation Lookaside Buffer
  - Exception generation
- Close interaction between OS & the hardware

Next Lecture

- Procedure calls
  - Stack frames
  - Hardware support for procedure calls
  - Compiler requirements