Computer Architecture

R. Poss
Computer Systems Architecture group (UvA)
e-mail: r.c.poss@uva.nl

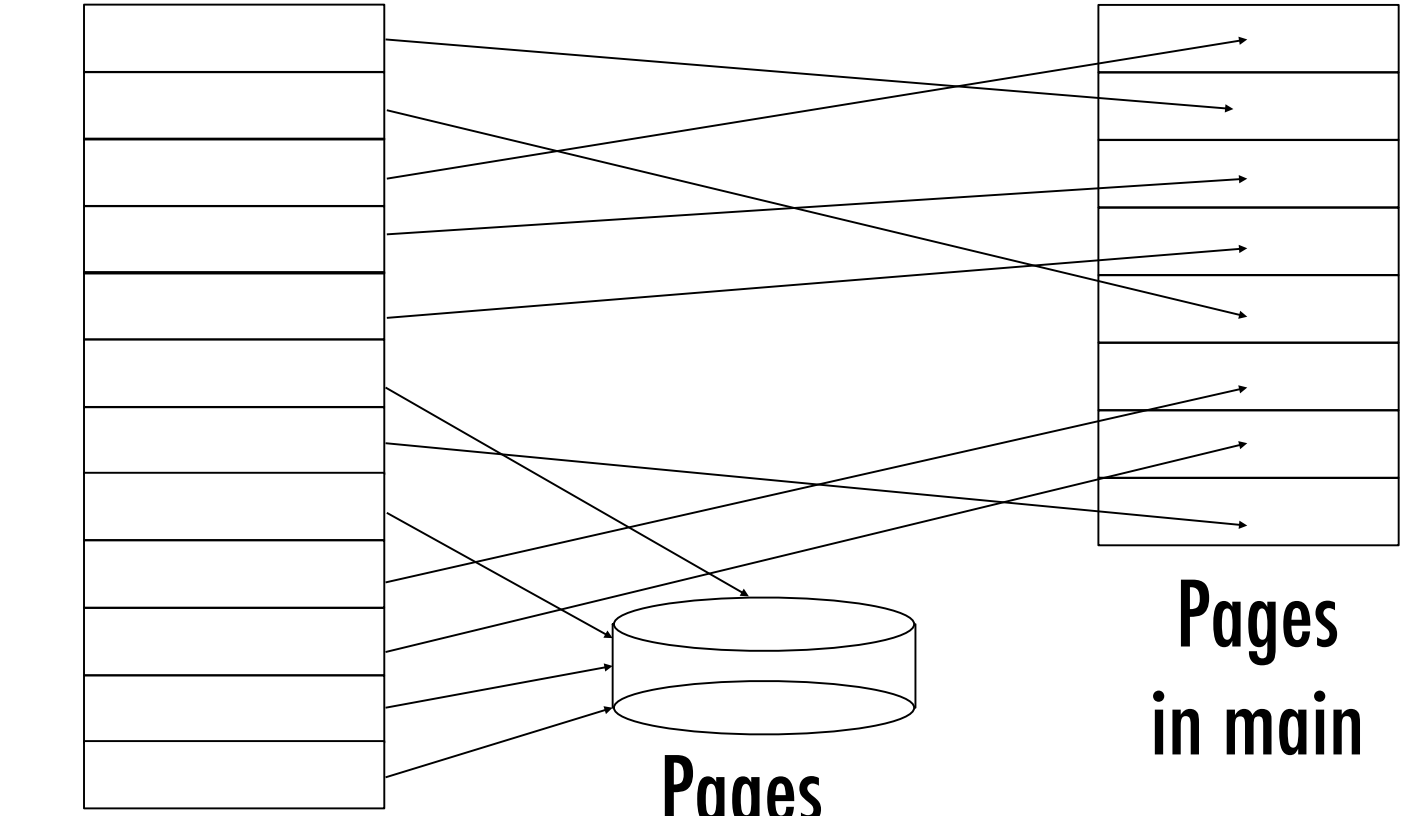
Virtual Memory

cf. Henessy & Patterson, App. C4

Virtual Memory

- It is easier for the programmer to have a large Virtual memory than to program explicit I/O due to memory limitations
- Also in multi-programming memory is shared between many programs, some suspended or inactive for a while
  - **only a small fraction of virtual memory is used at any one time in a multi-programming environment**
- Virtual memory uses main memory to store only part of the larger virtual memory space and the remainder is held on external storage, eg discs
  - The unit exchanged between memory and disc is called a **page** which may be divided into segments

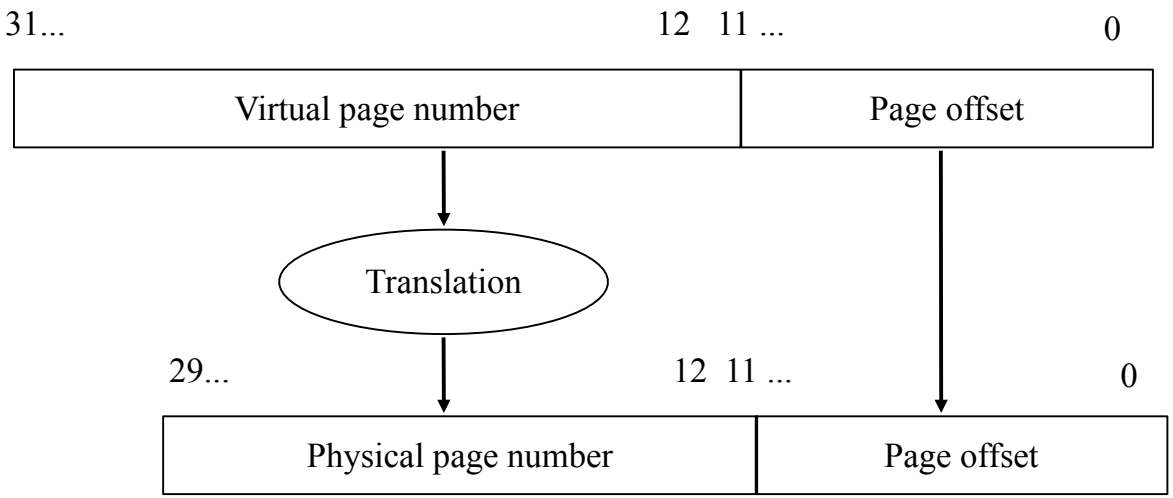
Virtual memory mapping



VM Terminology

- The replacement unit in virtual memory is called a **page**
- The address produced by the processor is called a **virtual address**
- This get translated by a **MMU** via a **page table**
  into a **physical address** (PT hit) or **page fault** (PT miss)
- The page table is in main memory but has a special cache called a **TLB**
  (translation look-aside buffer)
  - Page faults usually managed by a software trap to an operating system
  - This mapping process is called **address translation**

VM Address translation



- This shows address mapping from a 4 GiB virtual address space onto in a 1 GiB physical address space using 4KiB memory pages
- The translation is performed using a 1M entries (3MiB) table in memory, addressed by the virtual page number

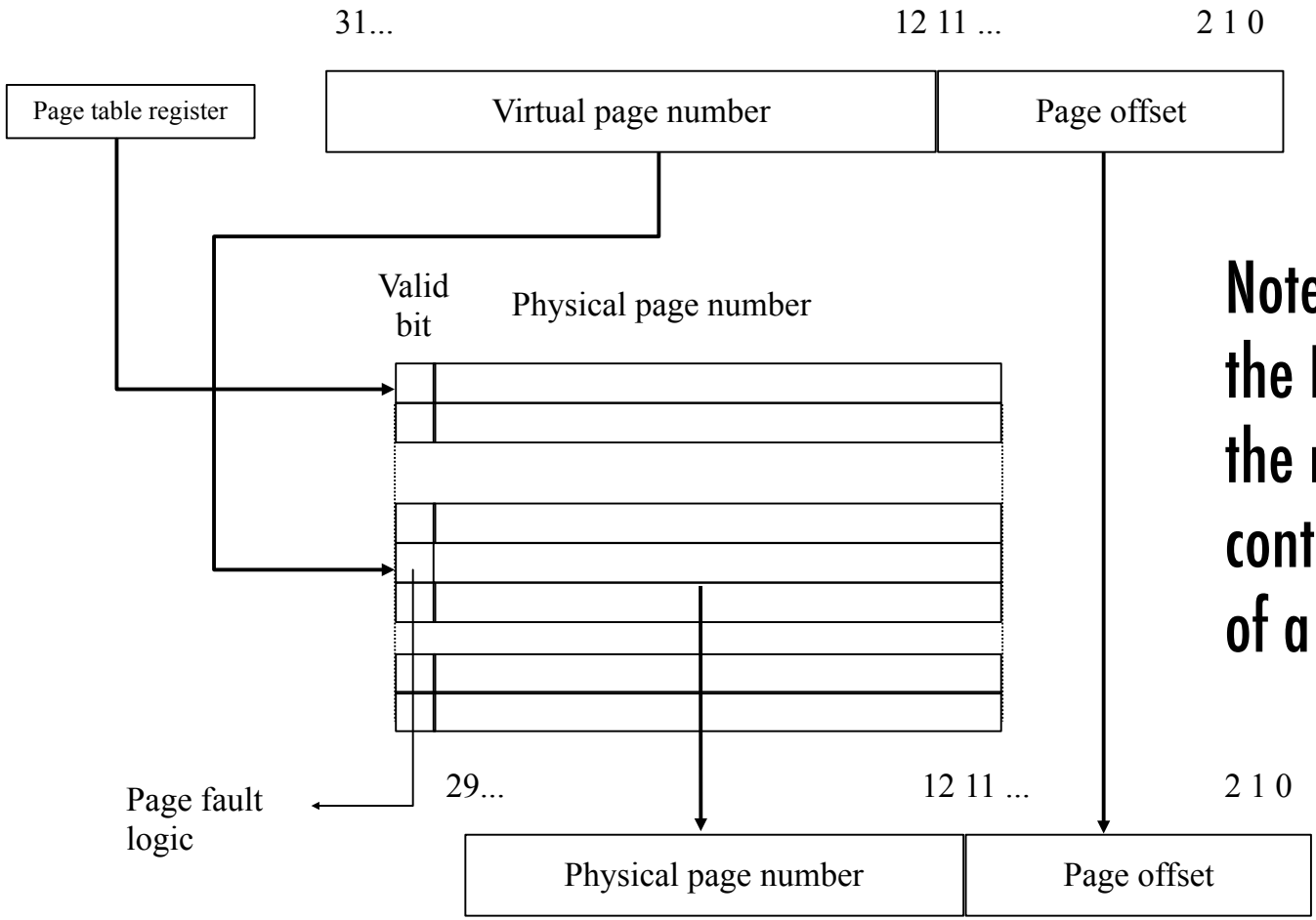
Virtual memory issues

- Need flexibility in page placement to avoid costly page misses
- Unlike cache mapping, VM mapping is implemented as a table in main memory - allows arbitrary mapping
  - indexed by virtual address
  - that yields the physical address
- Page misses are handled by software and incur a large penalty
- Pages must be *sufficiently large to amortise this large overhead* and *to minimise the mapping table size*
  - 4 to 64KByte is a typical page size
    with *variable size pages* can be as large as 1MByte

Replacement, processes and protection

- Sophisticated algorithms for placement can be coded in software
  - pages known to be often required can be locked down
- Each **process** has its own virtual address space and page table
  - this means programs can not interfere (read/write) the memory of any other
- To achieve **protection**, user code must be prevented from altering the page tables
  - This is normally achieved by having different *modes* of operation (eg. user mode vs. kernel mode)
  - alternatively, using security *capabilities* on the page table data

Page table



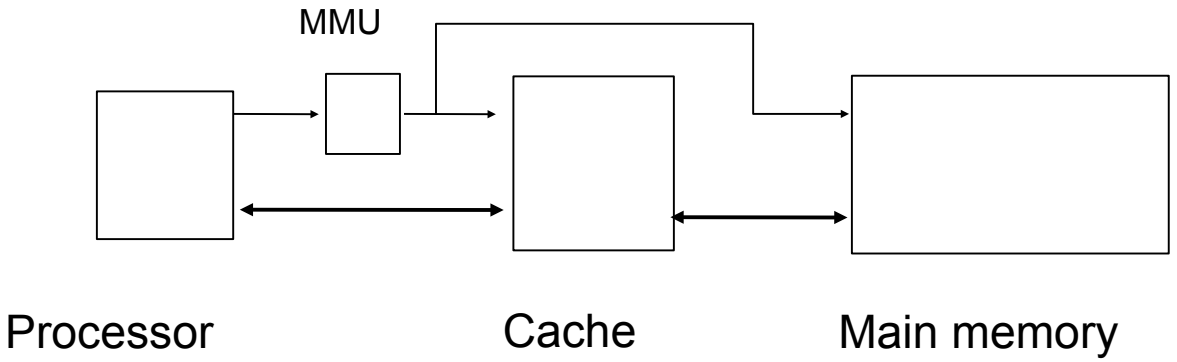
Note: the page table, the PC and the state of the registers all contribute to the state of a program

Translation Look-aside buffers

- Translation Look-aside buffers (TLB) cache the page table in small fast memory
- NB: The page table is too large to be held entirely in fast memory
- Without the TLB, access to memory would be twice as slow
  - One access to the page table for address translation
  - One to the data itself
- Address translation and L1 cache access can be performed in one or two processor cycles (so long as we get a cache hit)
- **Big question: which memory space do we cache: Virtual or Physical?**

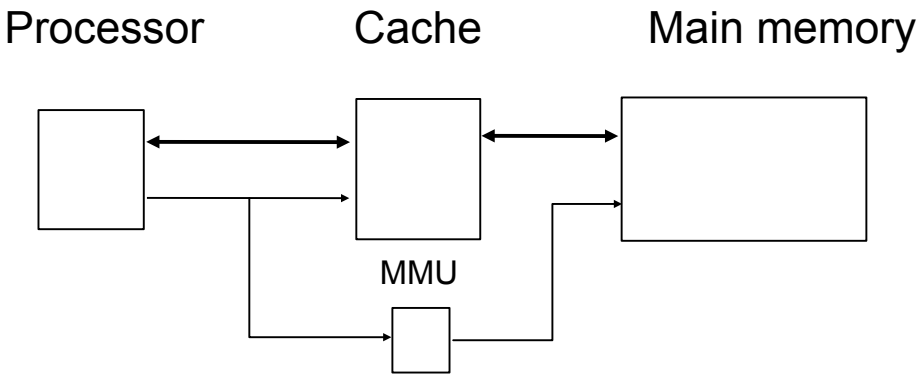
Physically addressed caches

- Addresses translated by memory management unit (MMU) before cache lookup
- Sequential - even with a TLB and cache hit,
  access can be slow as it requires sequential memory accesses



Virtually addressed caches

- Addresses translated by MMU in parallel with cache lookup
- **Aliasing** – is where the same virtual address in different processes maps to the same location in cache
- Context switching therefore requires a full cache invalidation (time expensive) or a process identifier in the tag (space expensive)
- Aliasing is averted if all processes share the same virtual address space



Page table size

- The example earlier was for 32-bit addresses and yielded a 1MiB table
- For a 64-bit architecture and say a 48-bit virtual address and 4KiB pages we get:
  - table size = $2^{48}/2^{12} = 2^{36}$ entries = 2^{39} bytes = 512GiB!!
  - and this is replicated for each process (!!)
- Solution is to **grow page table as required**
  - keep limit and check limit on each access
  - increase (e.g. double size) on each overflow

Page table size

- Address usage may be **sparse**
- Another solution is to use a **multi-level page table** as this takes advantage of sparseness
  - e.g. use very large pages and keep a table of these
  - within a large page keep a table of smaller pages (e.g. 4KiB)

Multi-level page table