MANCHEStER
1824

The University
of Manchester

Memory hierarchy

- Outline:
  - memory hierarchy basics
  - on-chip RAM and caches
  - memory management
  - operating systems

☞ hands-on: C and assembly code interworking

MANCHEstER 1824

The University of Manchester

Memory hierarchy

❑ Outline:

➜ **memory hierarchy basics**

❍ on-chip RAM and caches

❍ memory management

❍ operating systems

☞ hands-on: C and assembly code interworking

MANCHEstER
1824

The University
of Manchester

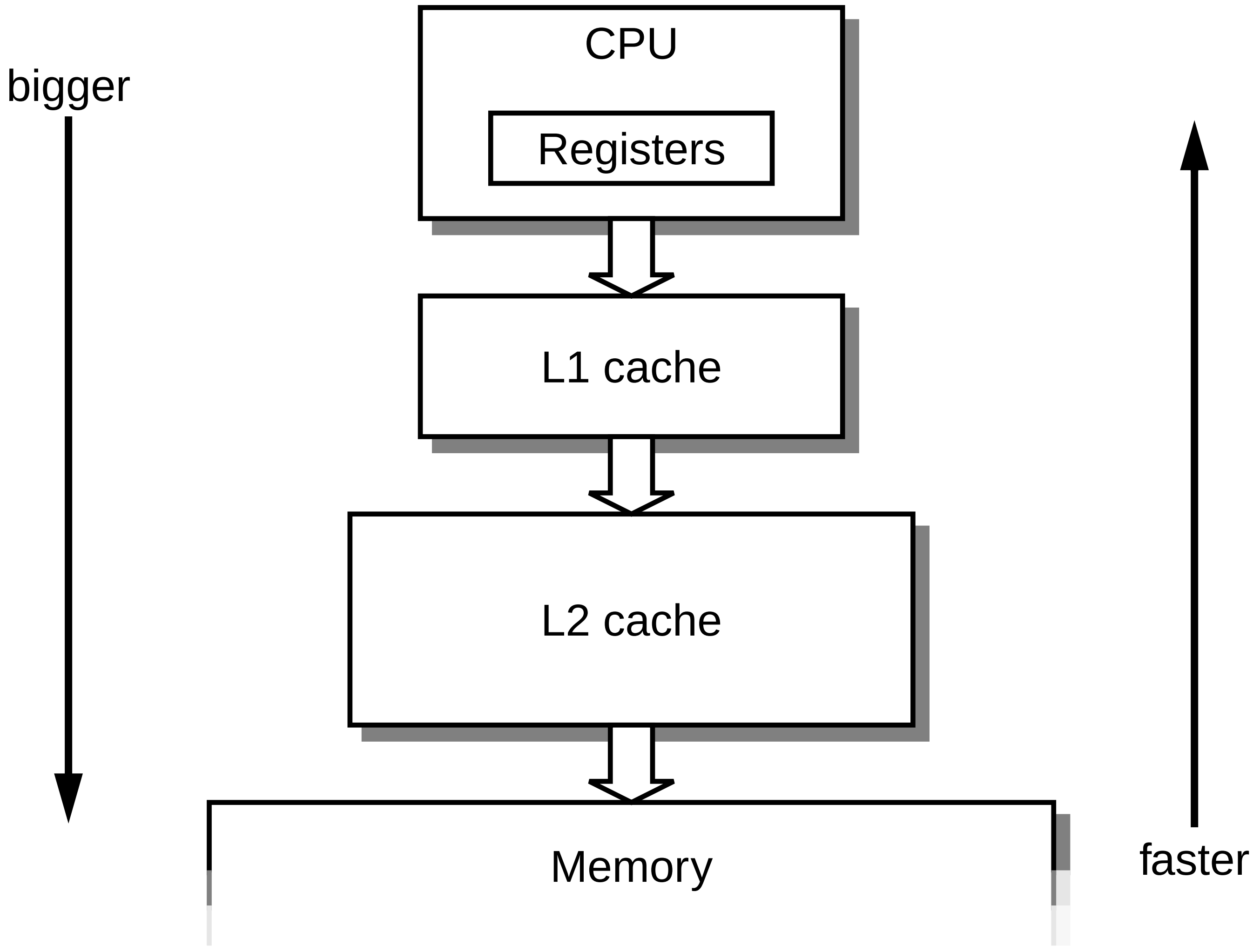
Memory hierarchy

- A typical system has several different memory subsystems:
  - processor registers: ~100 bytes, 1 ns
    - access is a small part of a clock cycle
  - on-chip cache or RAM: ~10 Kbytes, 5 ns
    - accessed at the processor clock rate
  - off-chip ROM and RAM: ~ Mbytes, 50 ns
    - access costs several processor cycles
  - backup store: ~ Gbytes, 5 ms

MANCHEStER
1824

The University
of Manchester

Memory hierarchy



- There may be more or fewer (or no) levels of cache

Memory hierarchy

- Efficient operation depends on:
  - the right things
    - the code and data
  - being in the right place
    - the on-chip memory or processor registers
  - at the right time
    - when they are in use

MANCHEStER 1824

The University of Manchester

Memory hierarchy

- Processor registers
  - are managed directly by the compiler
- Cache
  - is managed automatically by the hardware
- On-chip RAM
  - is managed by the programmer
- Off-chip RAM
  - is managed by the operating system

Memory hierarchy

- The objective is to approach:
  - the performance of the fastest memory …
  - … at the cost/bit of the slowest memory
- Feasible because programs display:
  - *temporal locality*
    - accesses to a location are clustered in time
  - *spatial locality*
    - accesses are clustered in the address space

Memory hierarchy

- Outline:
  - memory hierarchy basics
  - ➜ **on-chip RAM and caches**
  - memory management
  - operating systems

  ☞ hands-on: C and assembly code interworking

On-chip RAM

- System benefits of on-chip memory:
  - increased performance – no wait states
  - reduced power consumption
  - improved EMC
- On-chip RAM (“Tightly Coupled Memory”) is used in preference to a cache in some embedded systems:
  - it is simpler, cheaper and uses less power
  - its behaviour is more deterministic
  - however it requires explicit management

Caches

- A cache is a small on-chip memory which *automatically*:
  - keeps copies of recently used memory values
  - supplies these to the processor when it asks for them again
    - thereby avoiding an off-chip memory access
  - decides which values to over-write when it is full

MANCHEStER
1824

The University
of Manchester

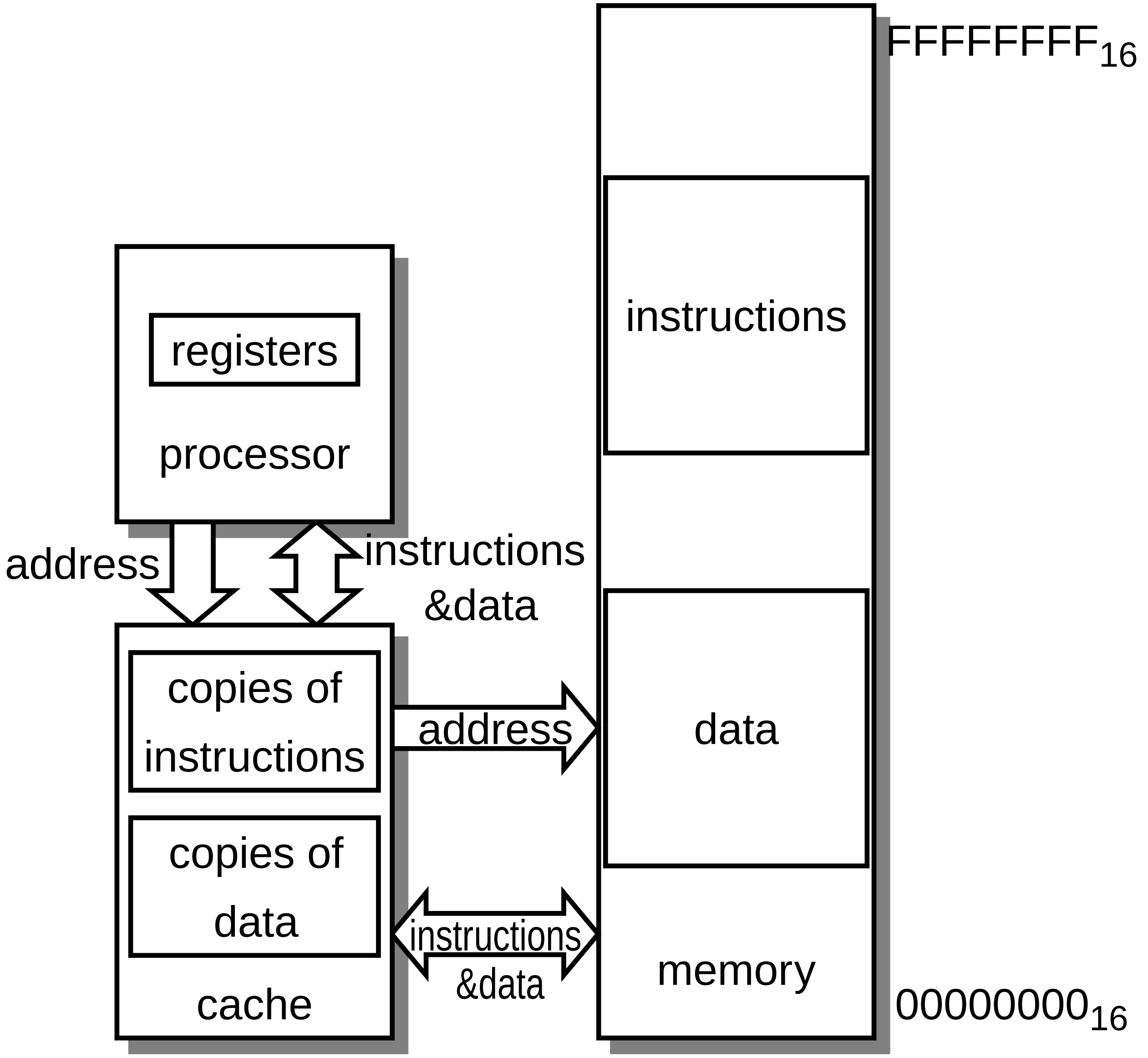
Cache organization

- There are many ways to arrange a cache:
  - separate or mixed instructions and data?
  - how much memory should be loaded on a cache miss?
  - how flexible should the allocation of cache space be?
  - how should writes be handled?

MANCHEstER 1824

The University of Manchester

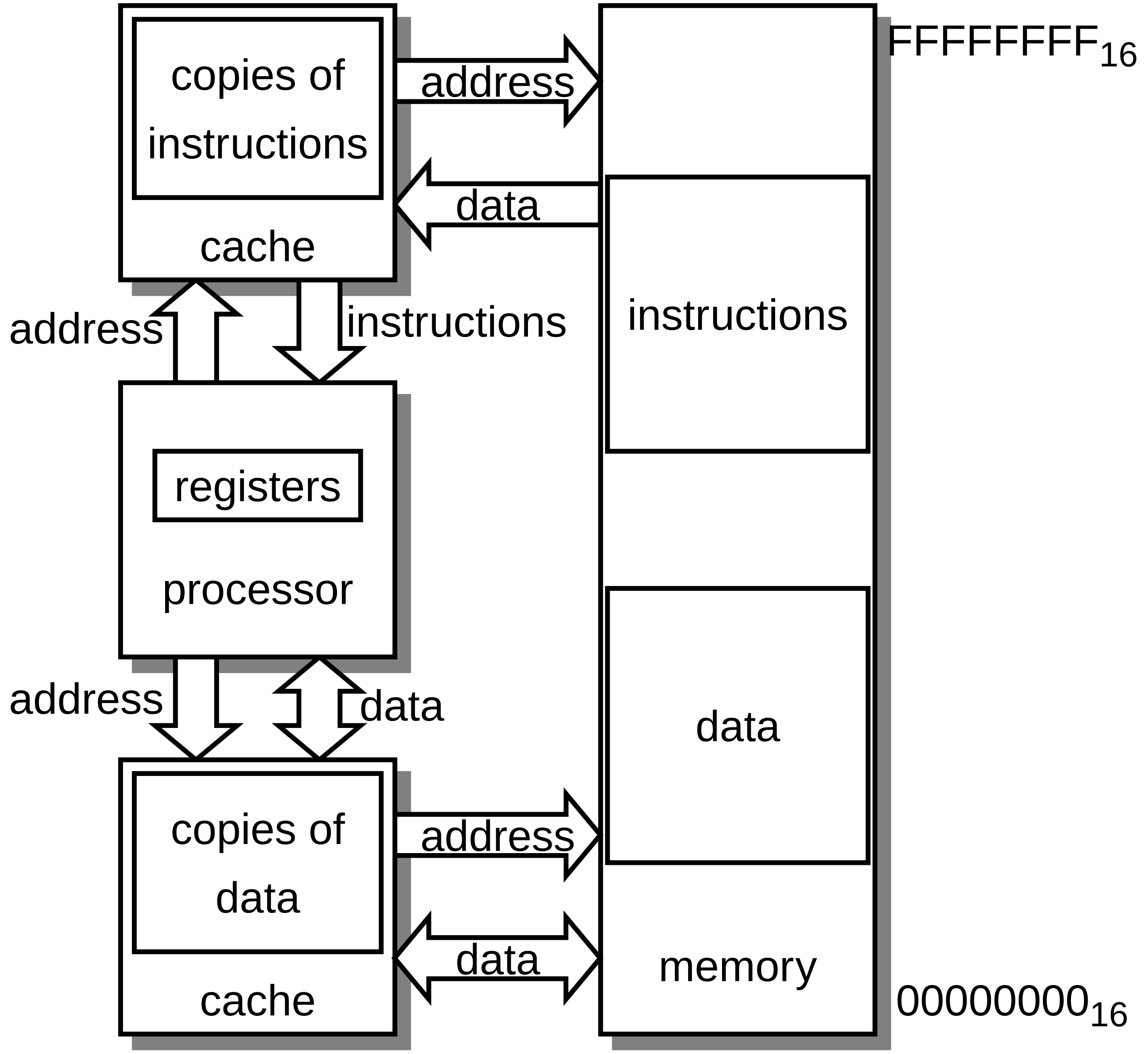
Unified instruction and data cache



MANCHEstER
1824

The University
of Manchester

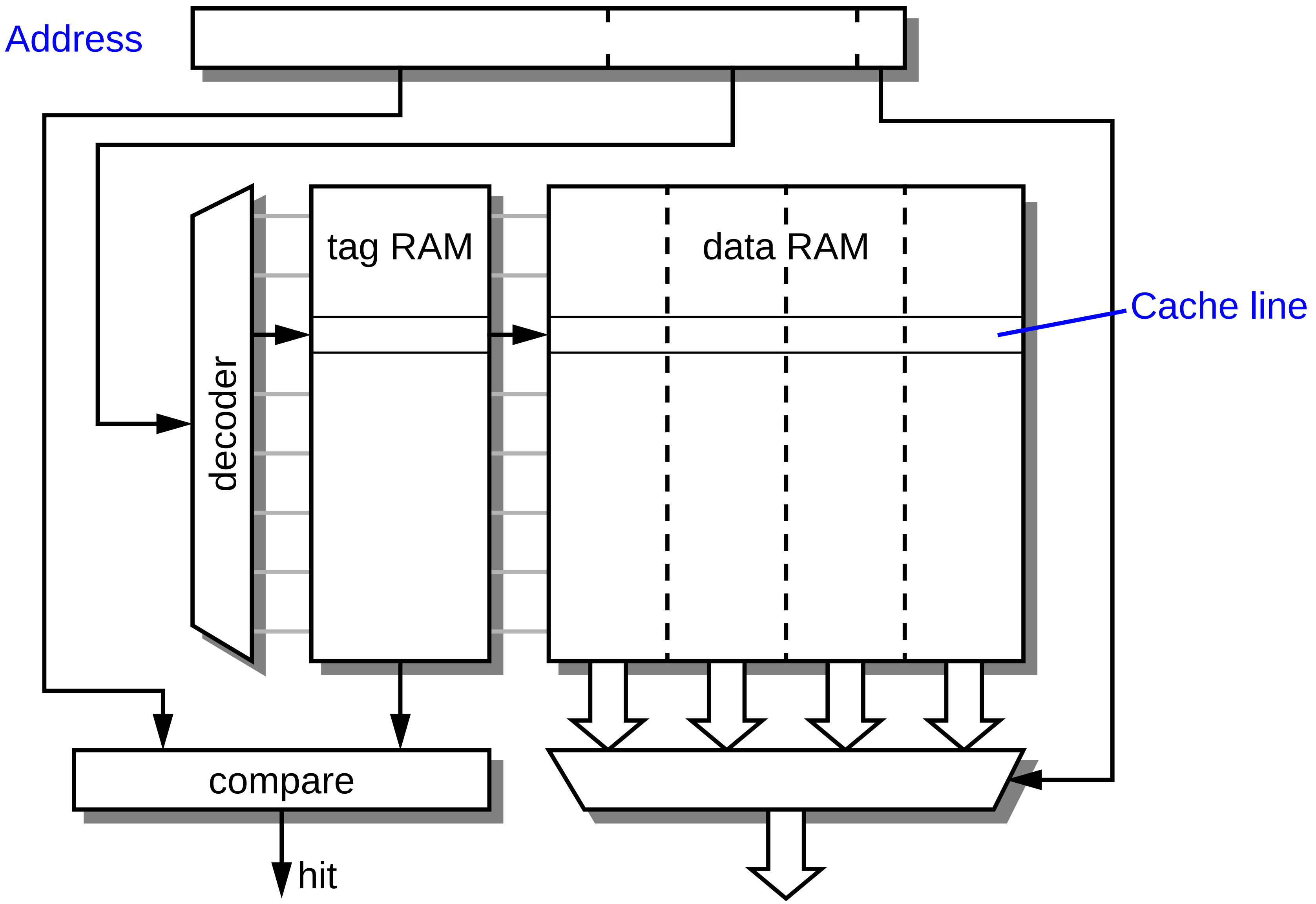
Separate data and instruction caches



MANCHEstER
1824

The University
of Manchester

Direct-mapped cache

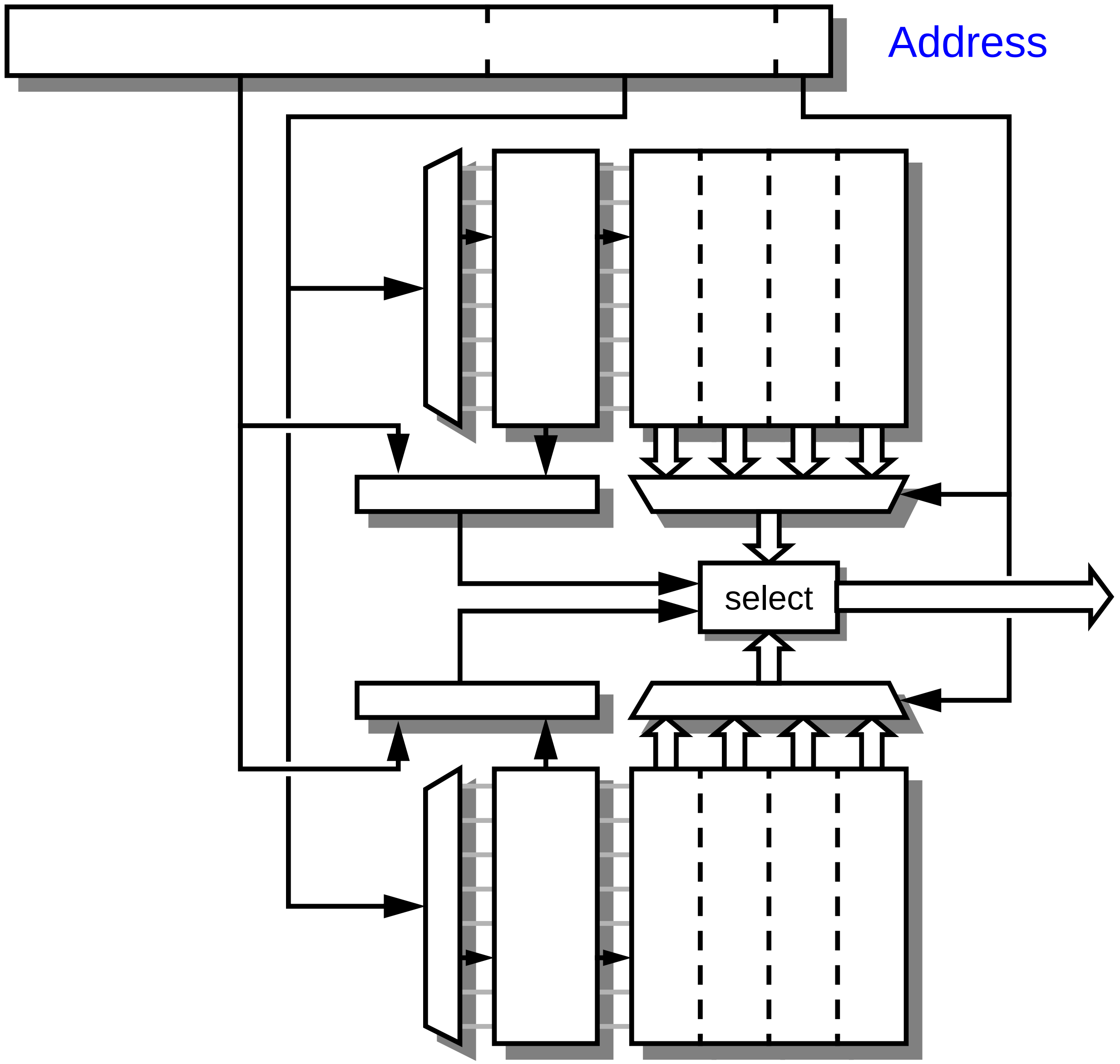


MANCHEStER 1824

The University of Manchester

2-way set associative cache

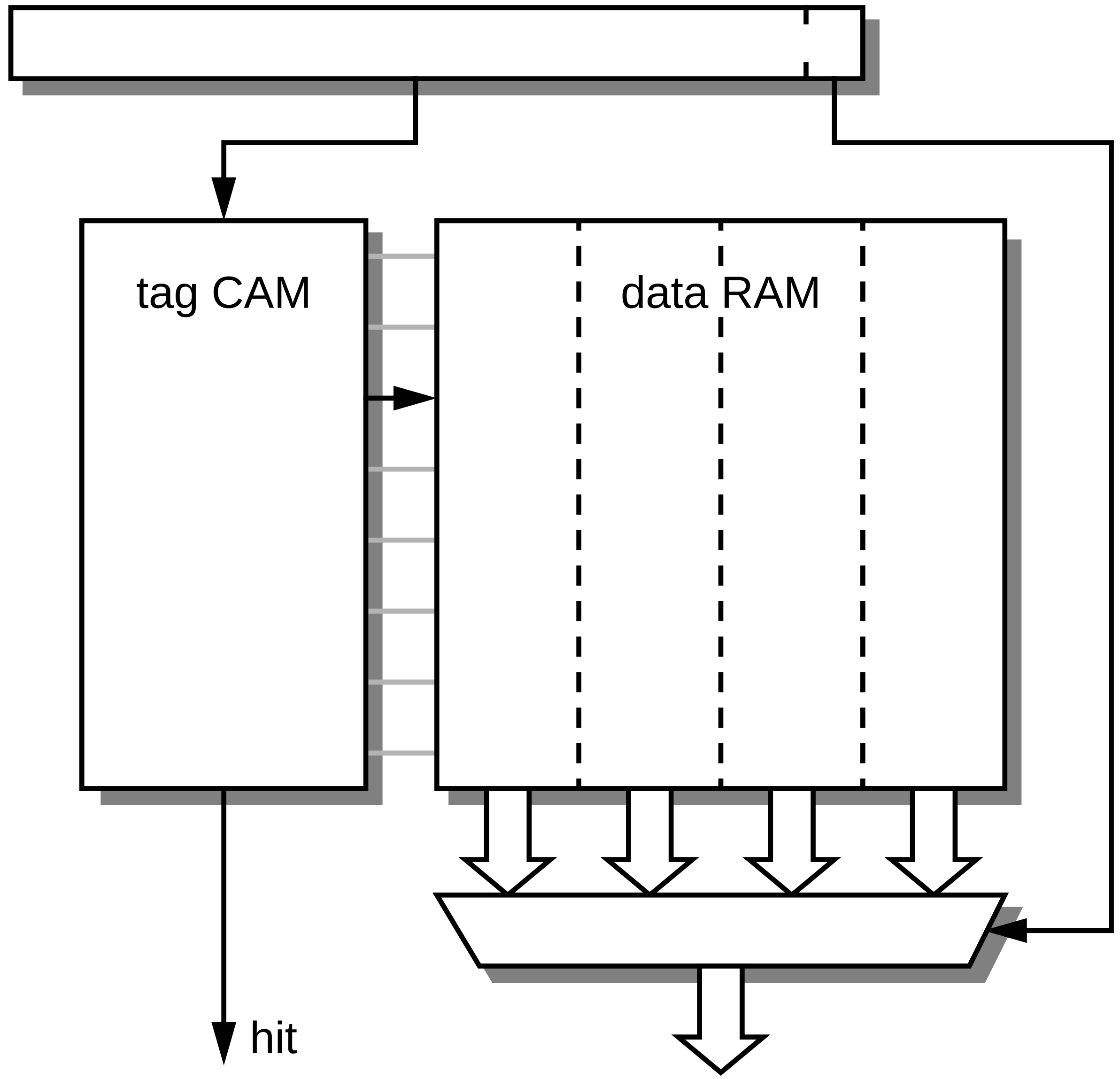
- two (smaller) cache blocks
- two chances to store any line
- better hit rate
- more expensive
- can extend to 4-way, etc.



Fully associative cache

- more places to store given line
- even better hit rate
- even more expensive
- (potentially) slower

- requires CAM (Content Addressable Memory)



Cache architecture comparison

- Direct mapped
  - simple, cheap, fast
  - subject to ‘thrashing’
  - choice for large caches
- Set associative
  - compromise
  - may be 2-, 4-, 8-, etc. way
  - often preferred
- Fully associative
  - best hit rate
  - slow, expensive
  - choice for small caches

Cache write strategies

- Write-through
  - all data is written to memory; matching cache locations are updated
- Write-through with write buffer
  - all data is written to memory, but the write is performed through a buffer
- Copy-back
  - the processor writes to the cache - main memory is only updated on flushes.

Cache power-efficiency

- What is the influence of organization on power-efficiency?
  - a high hit rate minimizes off-chip activity
    - hit rate increases with associativity (up to 4)
  - set-associative caches burn more power
    - due to the increased number of active sense amplifiers
  - CAM (in fully associative caches) is also power-hungry

MANCHEStER
1824

The University
of Manchester

Cache power-efficiency

- How can cache power-efficiency be improved?
  - use serial tag and data accesses in a set associative cache
    - enable only the relevant data RAM
  - segment the CAM in a fully associative cache
  - exploit sequential address sequences

MANCHEstER 1824

The University of Manchester

Memory hierarchy

- Outline:
  - memory hierarchy basics
  - on-chip RAM and caches
  - ➜ **memory management**
  - operating systems

☞ hands-on: C and assembly code interworking

Memory management

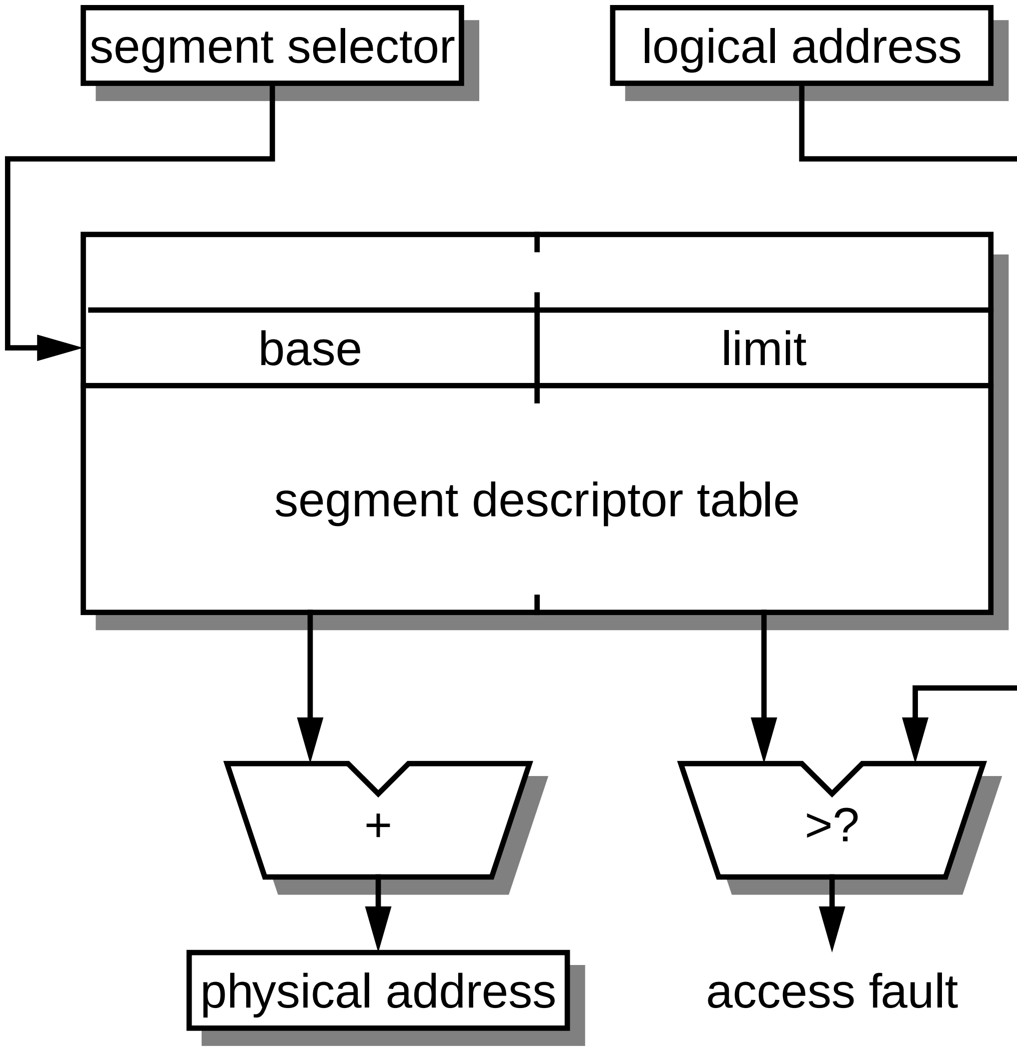
- Allows each program to run in its own address space
  - using *address translation*
    - greatly simplifying programming
- protects programs from other programs
  - using *memory protection*
    - improving system reliability
- supports the memory hierarchy
  - between the off-chip RAM and the disc

Address translation

- Translates
  - the processor's *logical* (or '*virtual*') address …
  - … into the *physical* memory address
- There are two main schemes:
  - *segmented* memory management
    - variable size (usually large) segments
  - *paged* memory management
    - fixed size pages, usually around 4 Kbytes

The University of Manchester

Segmented memory management

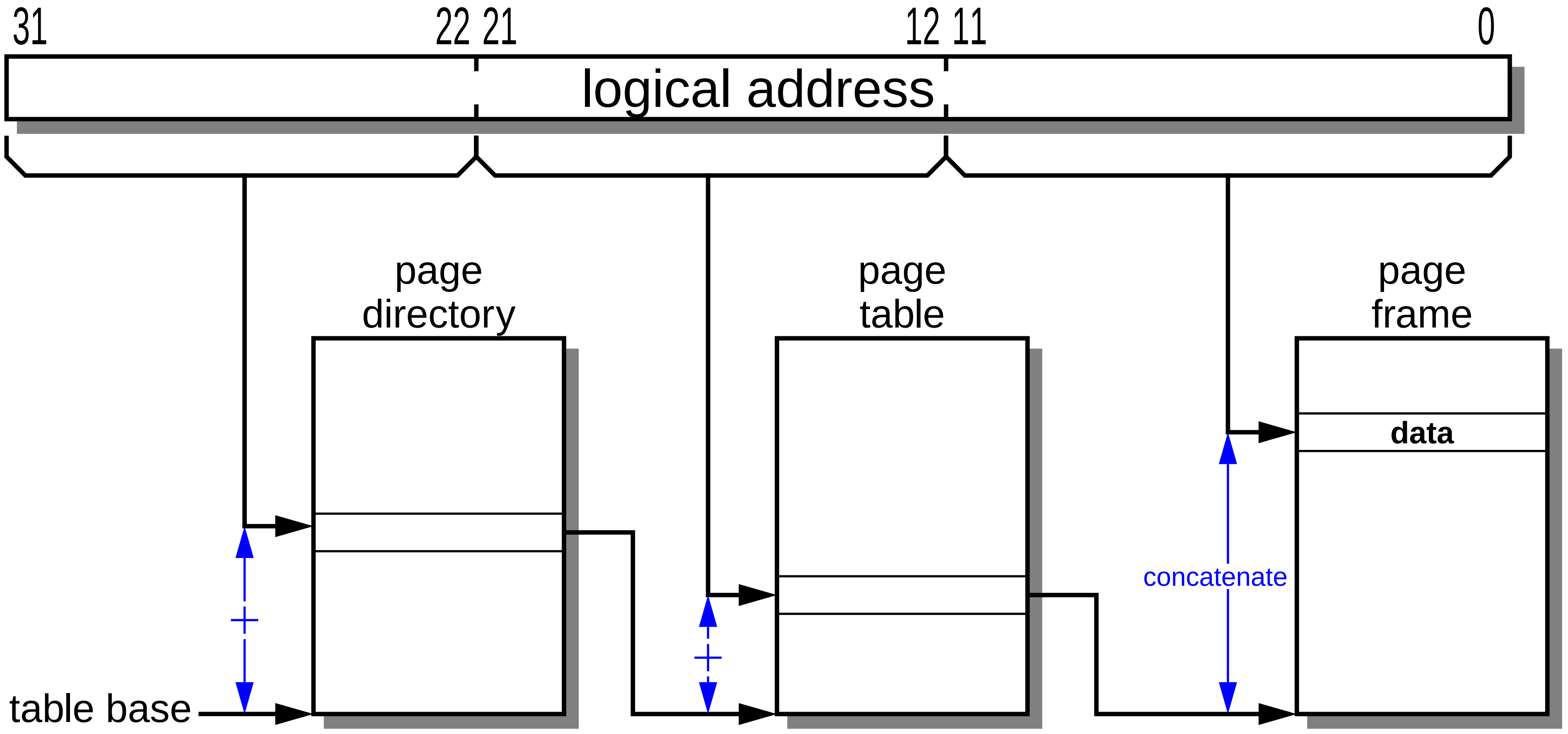


MANCHEstER 1824

The University of Manchester

Paged memory management

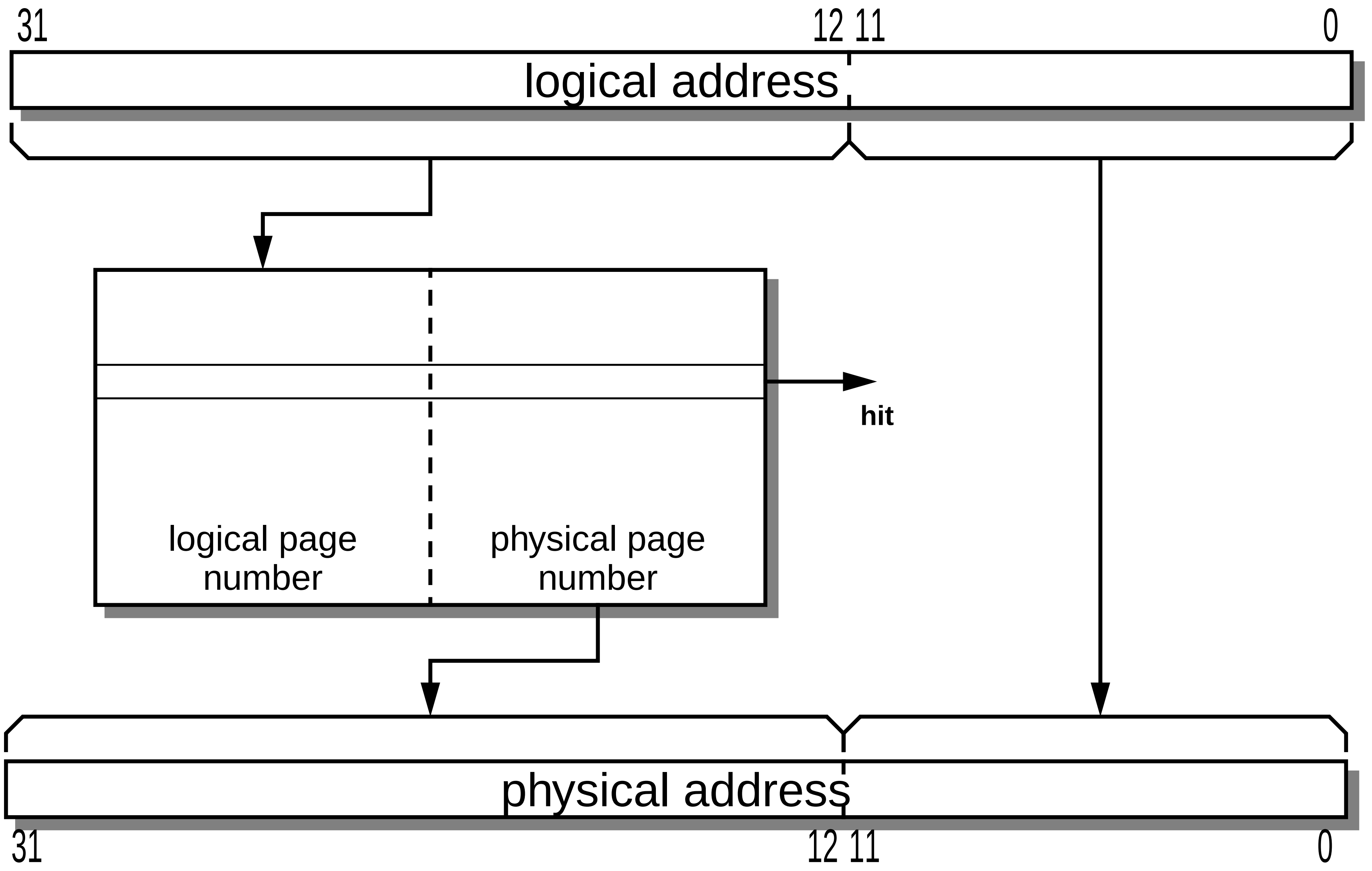
- Three stage look-up



Address translation

- Segmentation schemes generally keep all relevant registers on chip
- Paging schemes have too much translation data to keep on chip
  - three memory accesses per translation would be too slow!
  - therefore a cache of recently-used translations is kept on-chip
    - a *translation look-aside buffer* (TLB)

Translation look-aside buffer



Virtual and physical caches

- A cache may use *virtual* (pre-MMU) or *physical* addresses
  - physical caches
    - have fewer coherency problems
    - require a translation on every access
  - virtual caches
    - must be flushed whenever the translation tables change (e.g. on a process switch)
    - do not support synonyms
    - when two virtual addresses map to the same physical address

MANCHEStER
1824

The University
of Manchester

Memory hierarchy

- Outline:
  - memory hierarchy basics
  - on-chip RAM and caches
  - memory management
  - ➜ **operating systems**

☞ hands-on: C and assembly code interworking

MANCHEStER 1824

The University of Manchester

Operating systems

- Scheduling
  - an operating system allows a machine to run multiple programs concurrently
    - sometimes owned by different users
- Protection
  - each program is protected from errors in others (to a greater or lesser extent)
- Resource allocation
  - limited resources, conflicting demands

MANCHEstER
1824

The University
of Manchester

Operating systems

- Hardware support is needed:
  - memory management hardware
  - a protected operating system mode
    - to prevent unauthorized access to the MMU
- Single-user systems
  - do not need protection from malicious programmers!
    - except, possibly, virus and network attacks
  - use protection to improve reliability

MANCHEstER
1824

The University of Manchester

Operating systems

- Embedded systems
  - usually run a fixed set of programs
    - these can run in a single memory space
    - MMU hardware is often not needed
  - sometimes use a small *Real Time Operating System* (RTOS) for:
    - scheduling
    - hardware resource management
  - sometimes run a single program
    - no MMU; simple monitor operating system

MANCHEstER
1824

The University
of Manchester

Hands-on: C and assembly code interworking

❑ Closer look at the APCS

❍ see how assembly programs may be called from a C program

❍ look at various ways the stack may be used

☞ Follow the 'Hands-on' instructions