Inf2C - Computer Systems
Lecture 13
Memory Hierarchy and Caches

Boris Grot

School of Informatics

University of Edinburgh

Previous lecture: multi-cycle processor

- Execution time:

  `(inst count) x (cycles/inst) x (cycle time)`

- Multi-cycle processor: reduce the cycle time by breaking up an instruction's execution into multiple simple operations (1 or 2 per cycle)
  - Control FSM sequences the cycles
  - Reuse components (memory, ALU) to reduce "cost"

Memory requirements

- Programmers wish for memory to be
  - Large
  - Fast
  - Random access
- Wish not achievable with 1 kind of memory
  - Issues of cost and technical feasibility
- Idea of a **memory hierarchy**: approximate the "ideal" large+fast memory through a combination of different kinds of memories

Memory examples

Technology	Typical access time	$ per GB
SRAM	1-10 ns	£1000
DRAM	~100 ns	£10
Flash SSD	~100 µs	£1
Magnetic disk	~10 ms	£0.1

Which of these is "main memory"? **DRAM**

Memory hierarchy overview

- Use a combination of memory kinds
  - Smaller amounts of expensive but fast memory closer to the processor
  - Larger amounts of cheaper but slower memory farther from the processor
- Idea is not new:

"Ideally one would desire an indefinitely large memory capacity such that any particular … word would be immediately available… we are … forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible."

A. W. Burks, H. H. Goldstine, and J. von Neumann - 1946

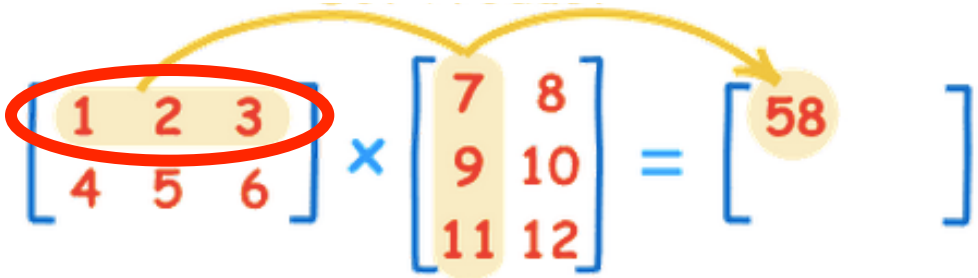
Why is a memory hierarchy effective?

- Temporal Locality:
  - A recently accessed memory location (instruction or data) is likely to be accessed again in the near future
- Spatial Locality:
  - Memory locations (instructions or data) close to a recently accessed location are likely to be accessed in the near future
- Why does locality exist in programs?
  - Instruction reuse: loops, functions
  - Data working sets: arrays, temporary variables, objects

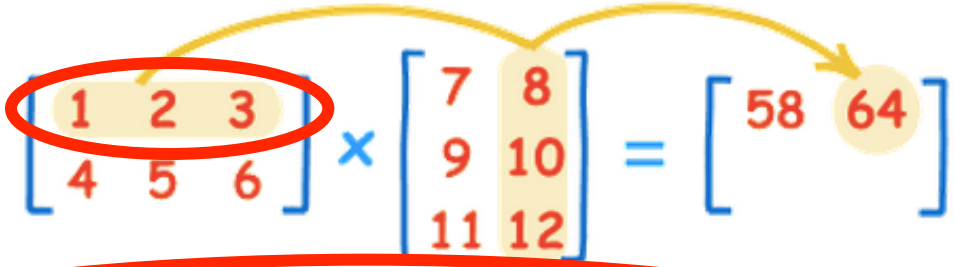
Example of Temporal & Spatial Locality

Matrix – matrix multiplication:

Spatial locality



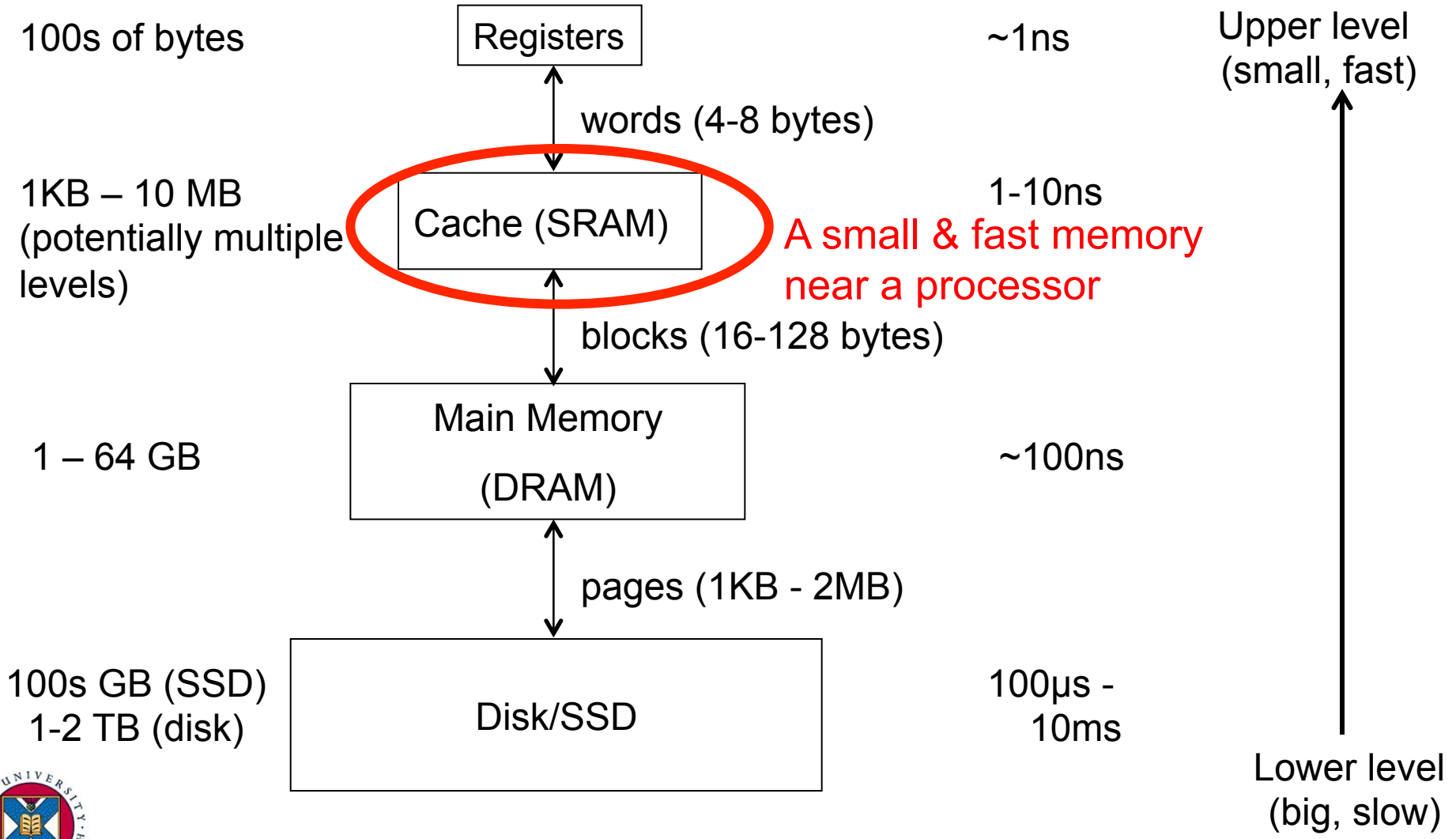
Temporal locality




```
for i = 1 to M
  for j = 1 to N
    for k = 1 to P
      c[i,j] = c[i,j] + a[i,k] * b[k,j]
```

Temporal & spatial locality in the code itself

Levels of the memory hierarchy



Memory hierarchy in a modern processor

- Small, fast **cache** next to a processor backed up by larger & slower cache(s) and main memory give the impression of a single, large, fast memory
- Take advantage of temporal locality
  - If access data from slower memory, move it to faster memory
  - If data in faster memory unused recently, move it to slower memory
- Take advantage of spatial locality
  - If need to move a word from slower to faster memory, move adjacent words at same time
  - Gives rise to **block** & **pages**: units of storage within the memory hierarchy composed of multiple contiguous words

Control of data transfers in hierarchy

- Q. Should the SW or HW be responsible for moving data between levels of the memory hierarchy?

- A. It depends: there is a trade-off between ease of programming, complexity, and performance.
  - *SW (compiler):* between registers and main memory or cache
  - *HW:* between caches and main memory (SW is usually unaware of caches)
  - *SW (Operating System):* between main memory and disk

Control of data transfers in hierarchy

- Q. Should the programmer explicitly copy data between levels of memory hierarchy?
- A. It depends: there is a trade-off between ease of programming and performance.
  - *Yes:* between registers and caches/main memory
  - *No:* between caches and main memory
  - *Sometimes:* between main memory and disk
    - *No:* when use disk area as virtual memory
    - *Yes:* when read and write files

HW-managed transfers between levels

- Occurs between cache memory and main memory levels
- Programmer & processor both oblivious to where data resides
  - Just issue loads & stores to "memory"
- Cache Hardware manages transfers between levels
  - Data moved or copied between levels automatically in response to the program's memory accesses
  - Memory always has a copy of cached data, but data in the cache may be more recent
    - This creates interesting problems.
      Discussed in Computer Architecture and Parallel Architectures ☺

Memory hierarchy terminology

- Block (or line): the unit of data stored in the cache
  - Typically in the range of 32-128 bytes
- Hit: data is found (this is what we want to happen)
  - Memory access completes quickly
- Miss: data not found
  - Must continue the search at the next level of the memory hierarchy (could be another cache or main memory)
  - After data is eventually located, it is copied to the memory level where the miss happened

More memory hierarchy terminology

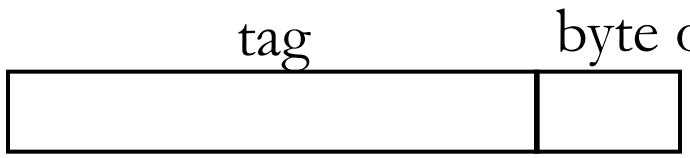
- Hit rate (hit ratio): fraction of accesses that are hits at a given level of the hierarchy
- Hit time: Time required to access a level of the hierarchy, including time to determine whether access is a hit or miss
- Miss rate (miss ratio): fraction of accesses that are misses at a given level (= 1 – hit rate)
- Miss penalty: Extra time required to fetch a block into some level from the next level down

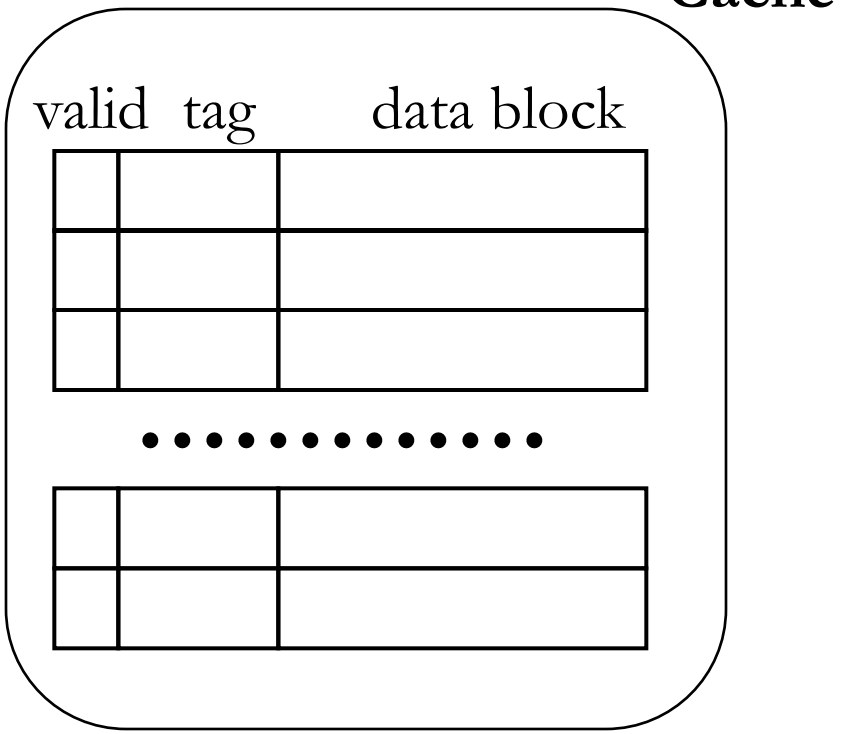
Cache basics

- Data are identified in (main) memory by their full 32-bit address
- Problem: how to map a 32-bit address to a much smaller memory, such as a cache?
- Answer: associate with each data block in cache:
  - a **tag** word, indicating the address of the main memory block it holds
  - a **valid bit**, indicating the block is in use

Fully-associative cache

requested address:





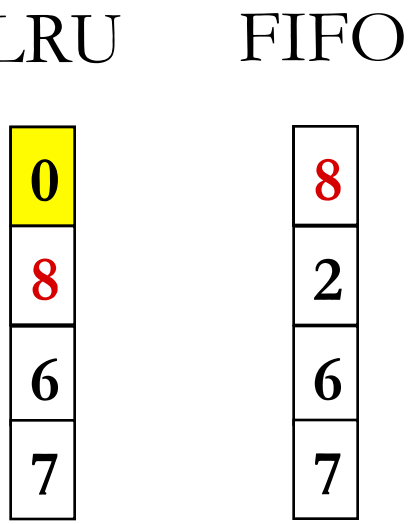
Correct cache block identified by matching tags

Byte offset selects word/byte within block

Address tag can potentially match tag of *any* cache block

Cache Replacement

- Least Recently Used (LRU)
  - Evict the cache block that hasn't been accessed longest
  - Relies on past behaviour as a predictor of the future
- FIFO – replace in same order as filled
  - Simpler to implement
- Example:
  - address references: 0 2 6 0 7 8
  - Cache with 4 blocks

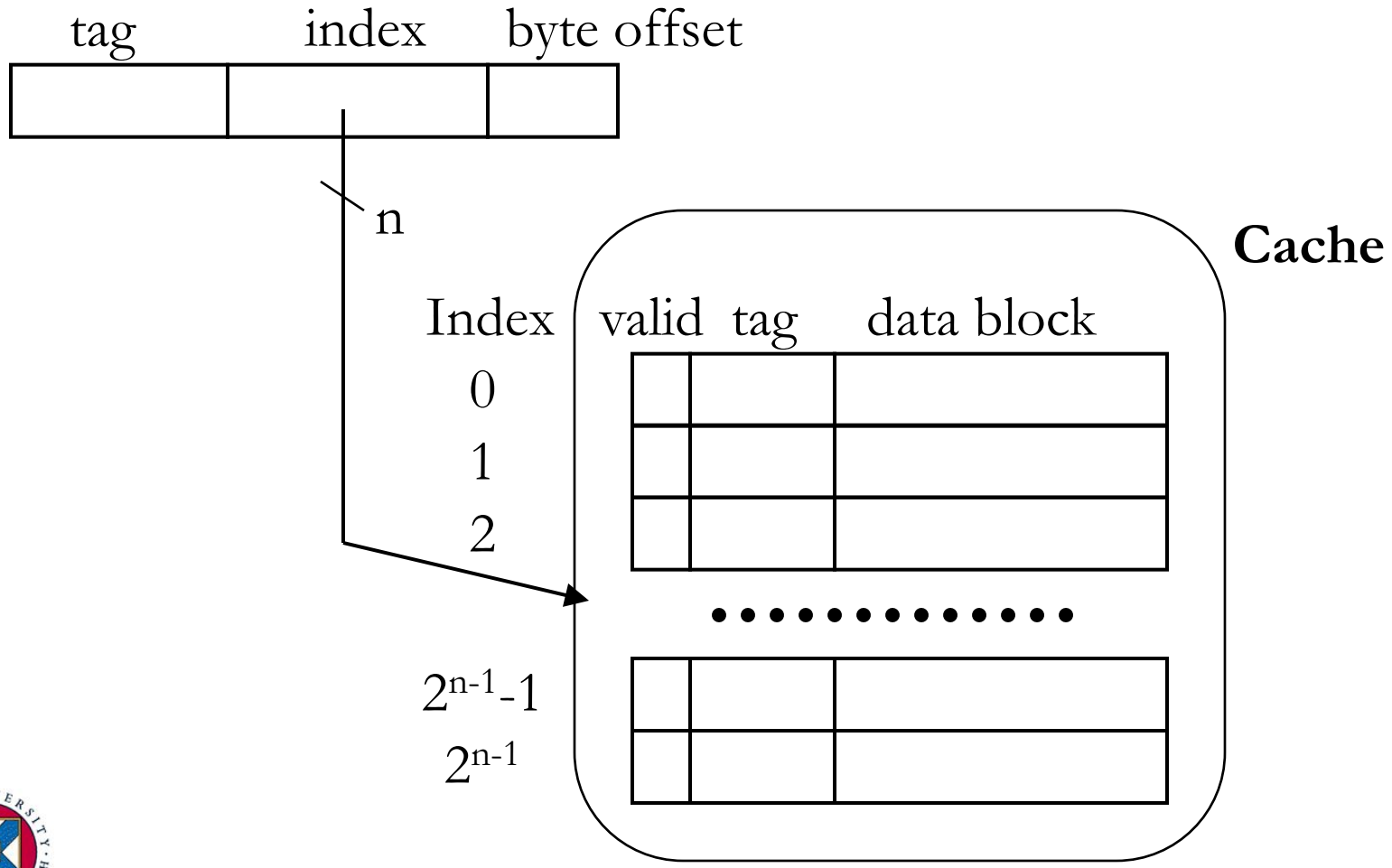


Direct-mapped cache

- In a fully-associative cache, search for matching tags is either very slow, or requires a very expensive memory type called Content Addressable Memory (CAM)
- By restricting the cache location where a data item can be stored, we can simplify the cache
- In a **direct-mapped** cache, a data item can be stored in one location only, determined by its address
  - Use some of the address bits as index to the cache array

Address mapping for direct-mapped cache

requested address:



Example problem

Given a 4 KB direct-mapped cache with 4-byte blocks and 32-bit addresses.

Question: How many tag, index, and offset bits does the address decompose into?

Example problem

Given a 4 KB direct-mapped cache with 4-byte blocks and 32-bit addresses.

Question: How many tag, index, and offset bits does the address decompose into?

Answer:

- 4 KB / 4 bytes per block = 1K blocks
  - Requires a 10-bit index
- 4-byte block: requires a 2-bit offset
- Tag: 32 – 10 – 2 = 20 bits

Direct-mapped cache in detail