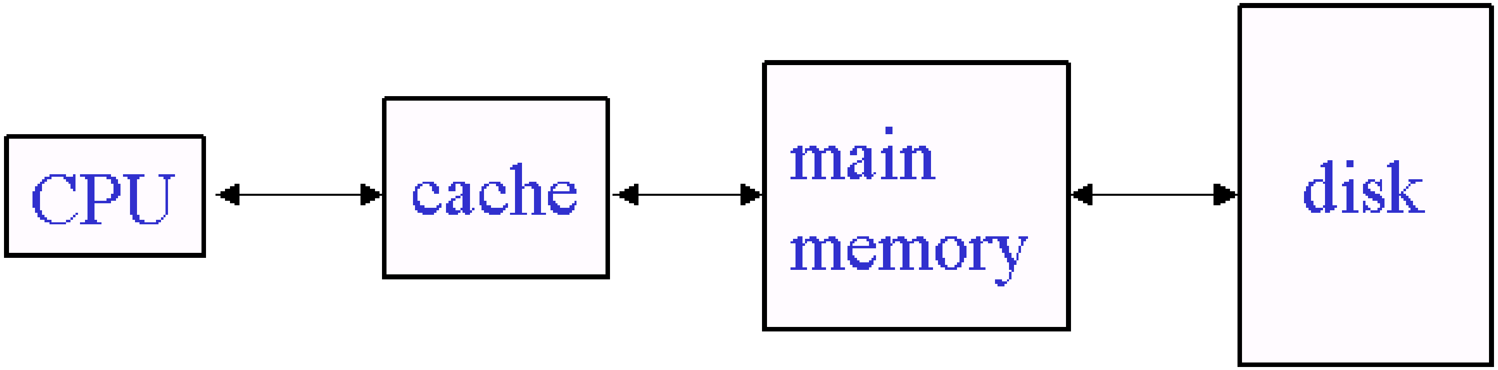
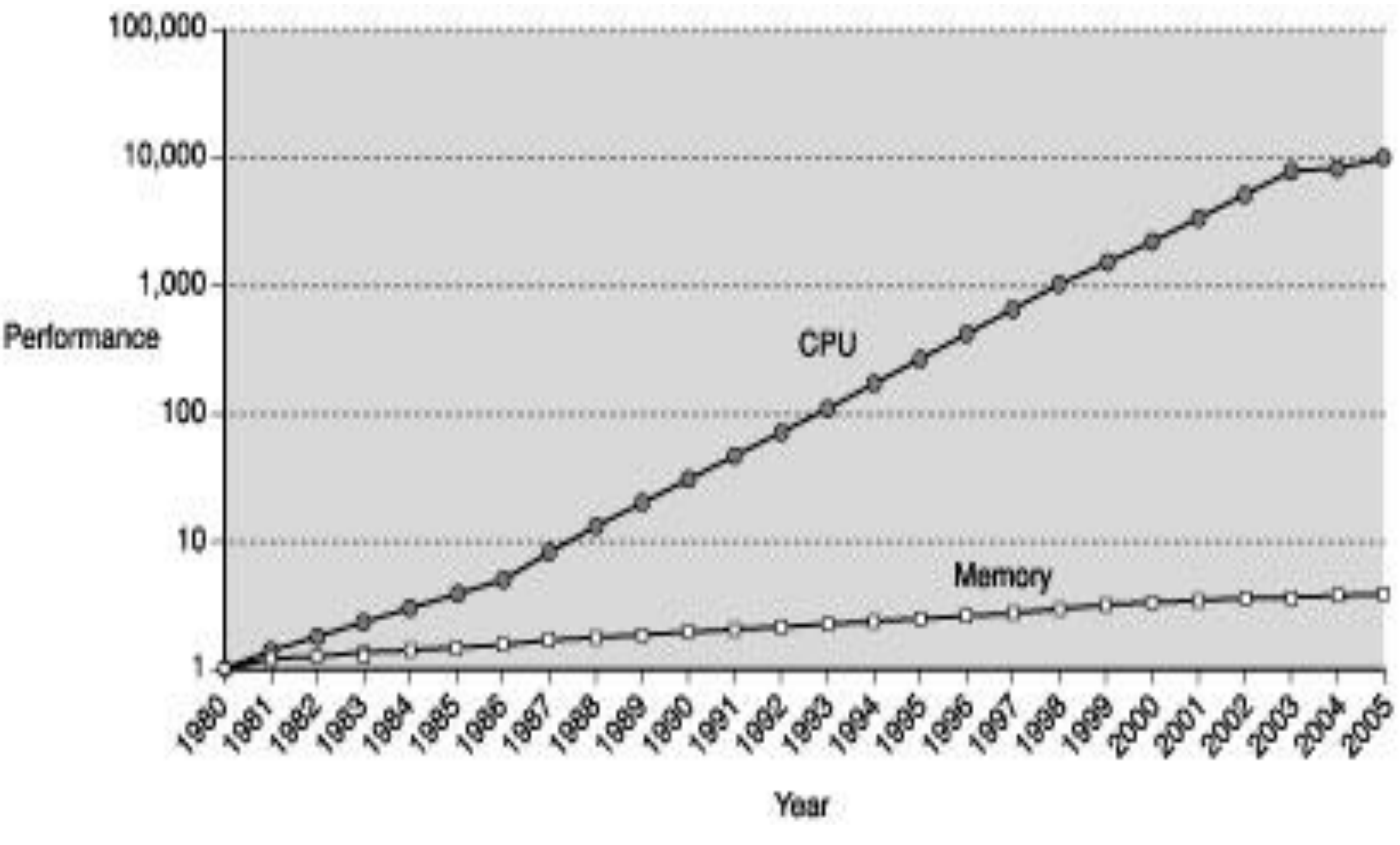
Why memory hierarchy

(3rd Ed: p.468-487, 4th Ed: p.452-470, 5th Ed: p.374-483)

- users want unlimited fast memory
- fast memory expensive, slow memory cheap
- cache: small, fast memory near CPU
- large, slow memory: main memory, disk, …
- connected to faster memory: one level up



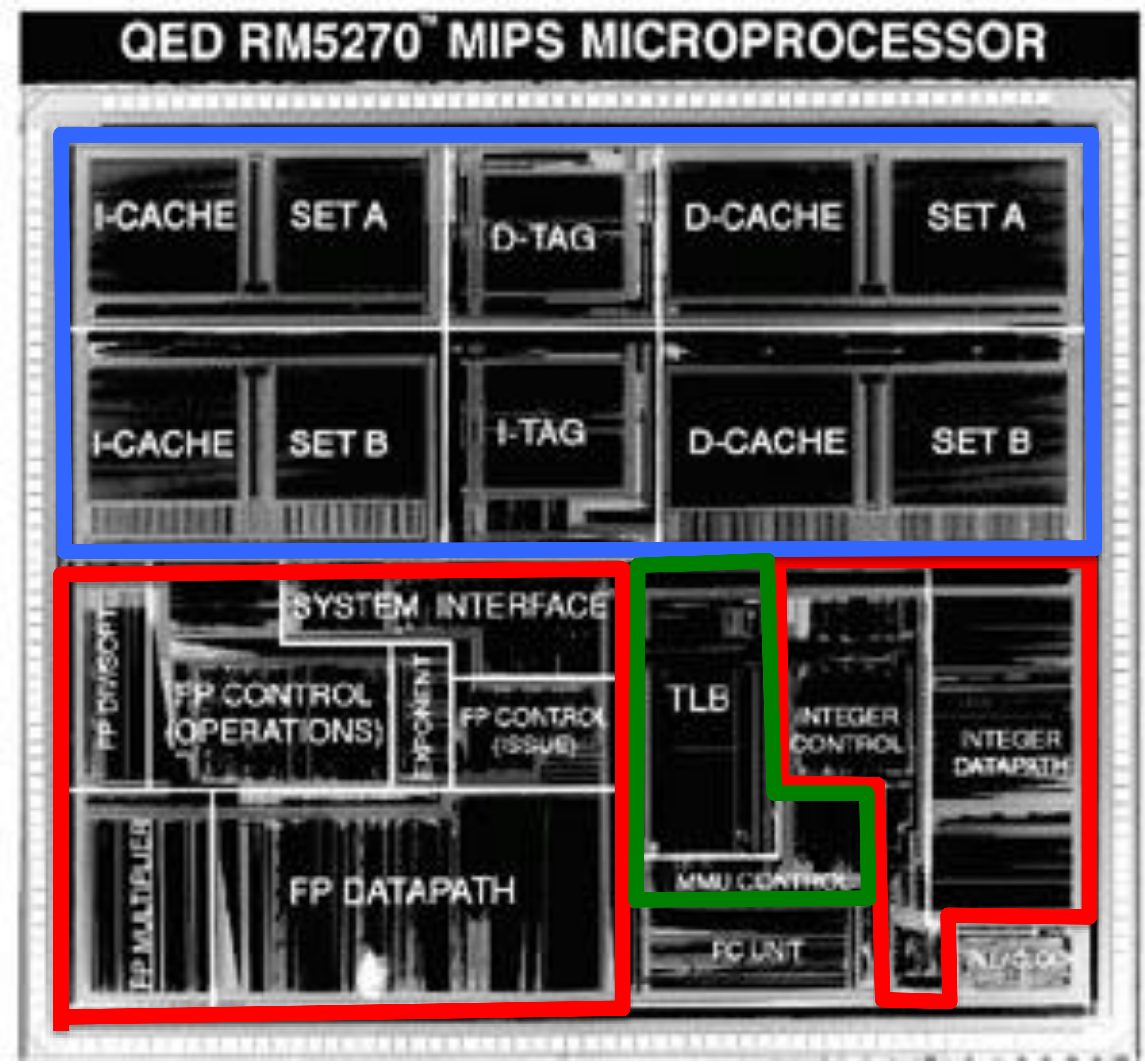
Performance of CPU vs main memory



When was this said?

- Ideally one desired an indefinitely large memory capacity such that any particular word would be immediately available…
- We are forced to recognise the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible…
- *Question: more levels of memory hierarchy?*

Reality Check: Typical MIPS Chip Die Photograph



Protection-oriented Virtual Memory Support

Performance Enhancing On-Chip Memory (iCache + dCache)

Floating Pt Control and Datapath

Integer Control and Datapath

(source: UCB 2011)

Cache operation

- data arranged in blocks
- cache hit: CPU finds required block in cache
- if not, cache miss - get data from main memory
- hit rate: ratio of cache access to total memory access
- hit time: time to access cache
  + time to determine cache hit or miss
- miss penalty: time to replace item in cache
  + time to transfer it to CPU

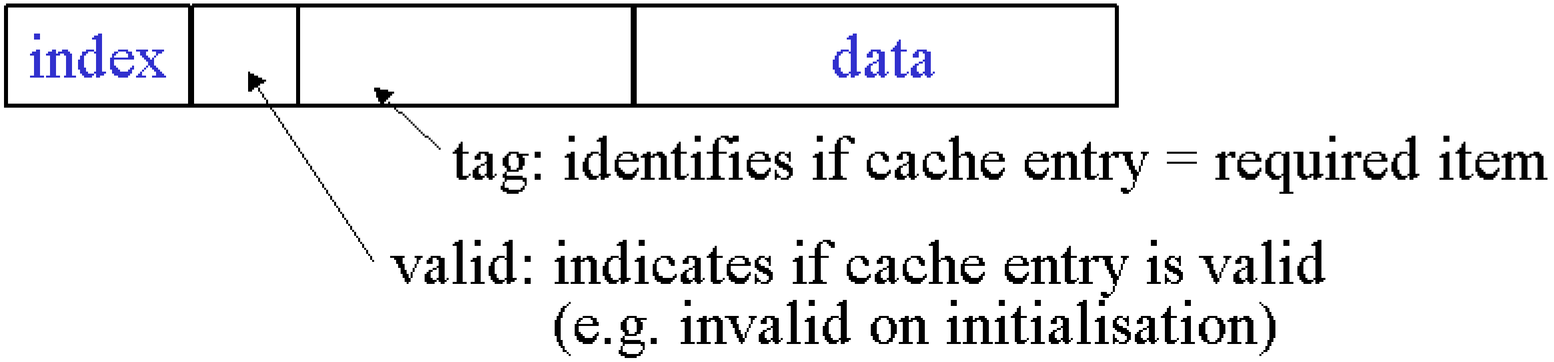
Locality principles

- temporal locality: items recently used by CPU tend to be referenced again soon
  - guides cache replacement policy: what to replace when cache is full
- spatial locality: items with addresses close to recently-used items tend to be referenced
  - fetches multiple data

Direct mapped cache

- each memory location mapped to *one* cache location
  e.g. cache index (cache address) = (memory block address) mod (number of blocks in cache)

- multiple memory location to *one* cache location
  e.g. 8 blocks in cache, cache location 001 may contain items from memory locations 00001, 01001, 10001

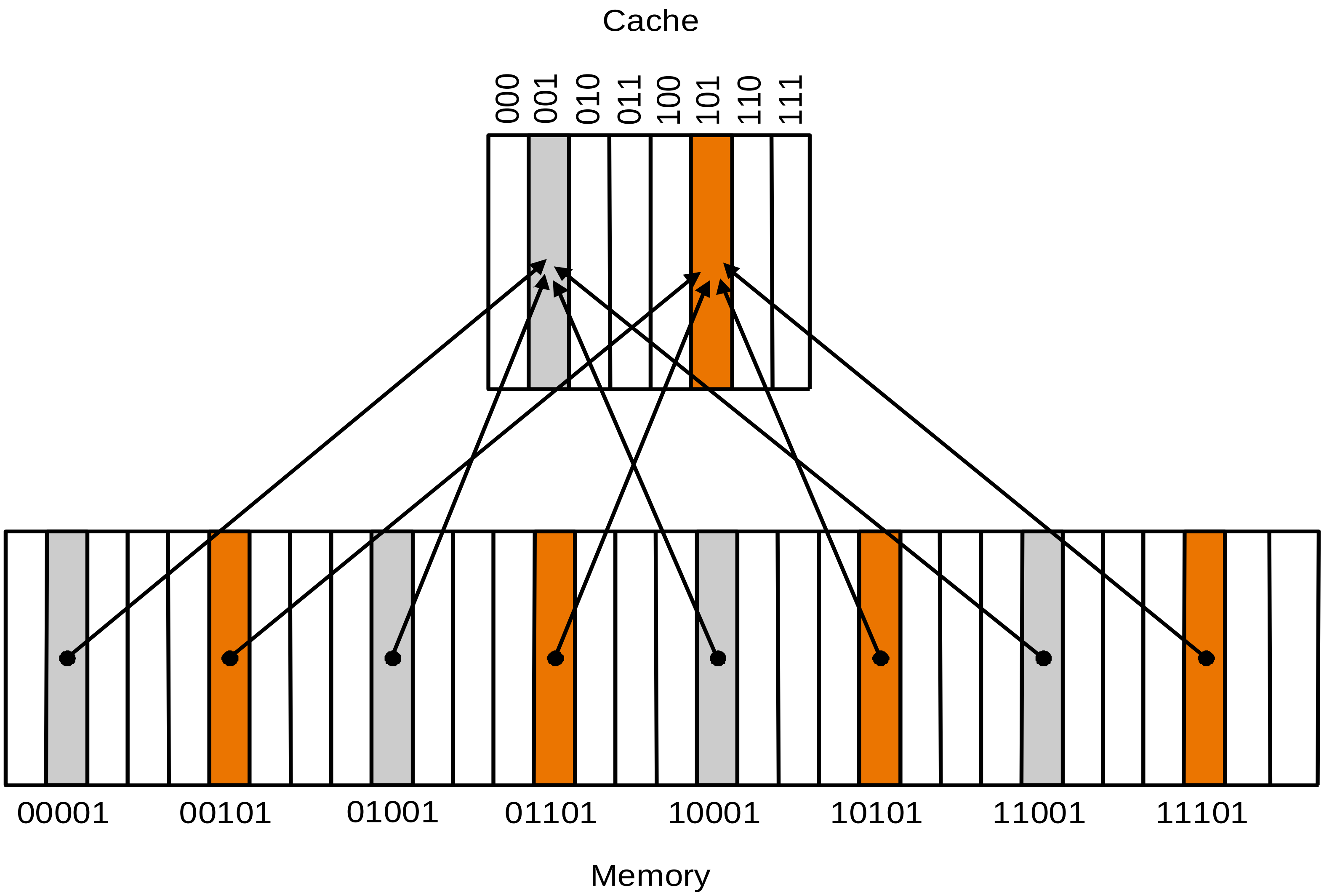
- 

tag: identifies if cache entry = required item

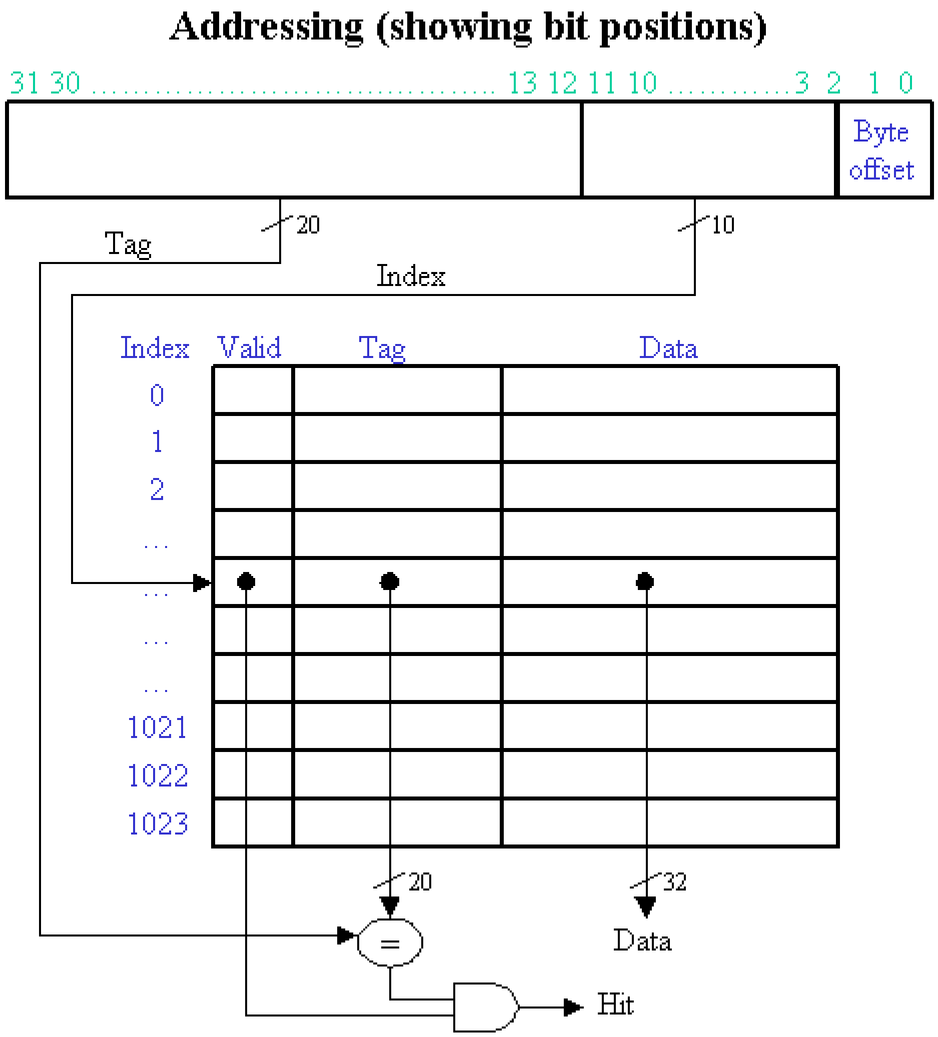
valid: indicates if cache entry is valid
(e.g. invalid on initialisation)

Direct mapped cache mapping

- address is modulo the number of blocks in the cache



Address translation



- lower portion of memory address: becomes cache index
- upper portion: compared with tag in cache to identify cache entry
- Hit if tag OK and valid

Handling misses: by exception

- cache miss on instruction read:
  - restore PC: PC = PC - 4
  - send address to main memory and wait (stall)
  - write returned data in cache
  - refetch instruction: now in cache
- cache miss on data read:
  - similar: stall CPU until data from main memory are available in cache

Cache write policy

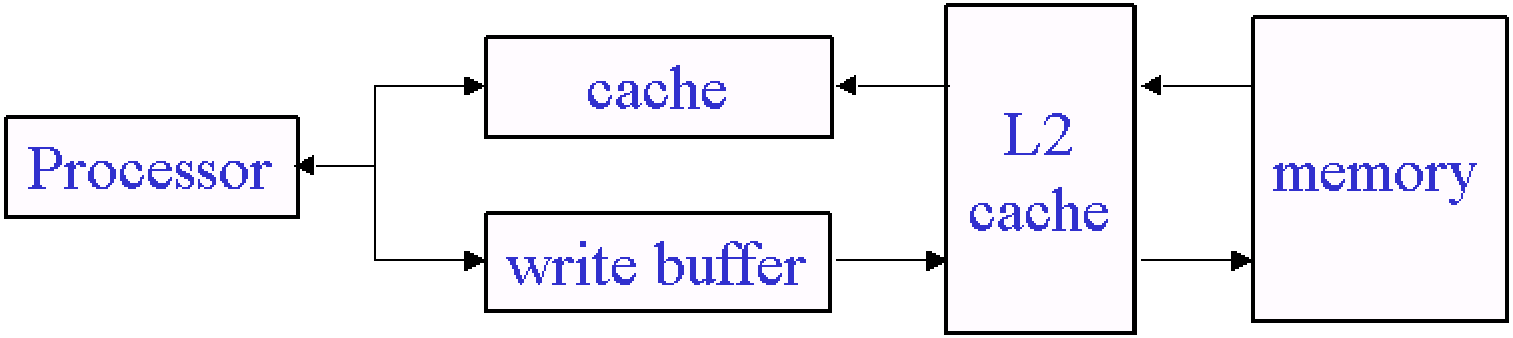
- cache write is tricky:
  - cache consistency problem - how to keep data in cache and in memory consistent
- write back: to cache only, transfer block to memory when the block is replaced on cache miss
  - reduce memory bandwidth
  - complex control, need ‘dirty’ bit
- write through: to cache and to memory concurrently
  - processor can continue execution after storing data in the write buffer

Write buffer for write through

- insert buffer between cache and memory
  - processor: write data into cache and write buffer
  - memory controller: write contents of the write buffer to memory
- write buffer is just a FIFO queue
  - fine if store frequency (w.r.t. time) « $\frac{1}{\text{memory write cycle}}$
  - otherwise have write buffer saturation

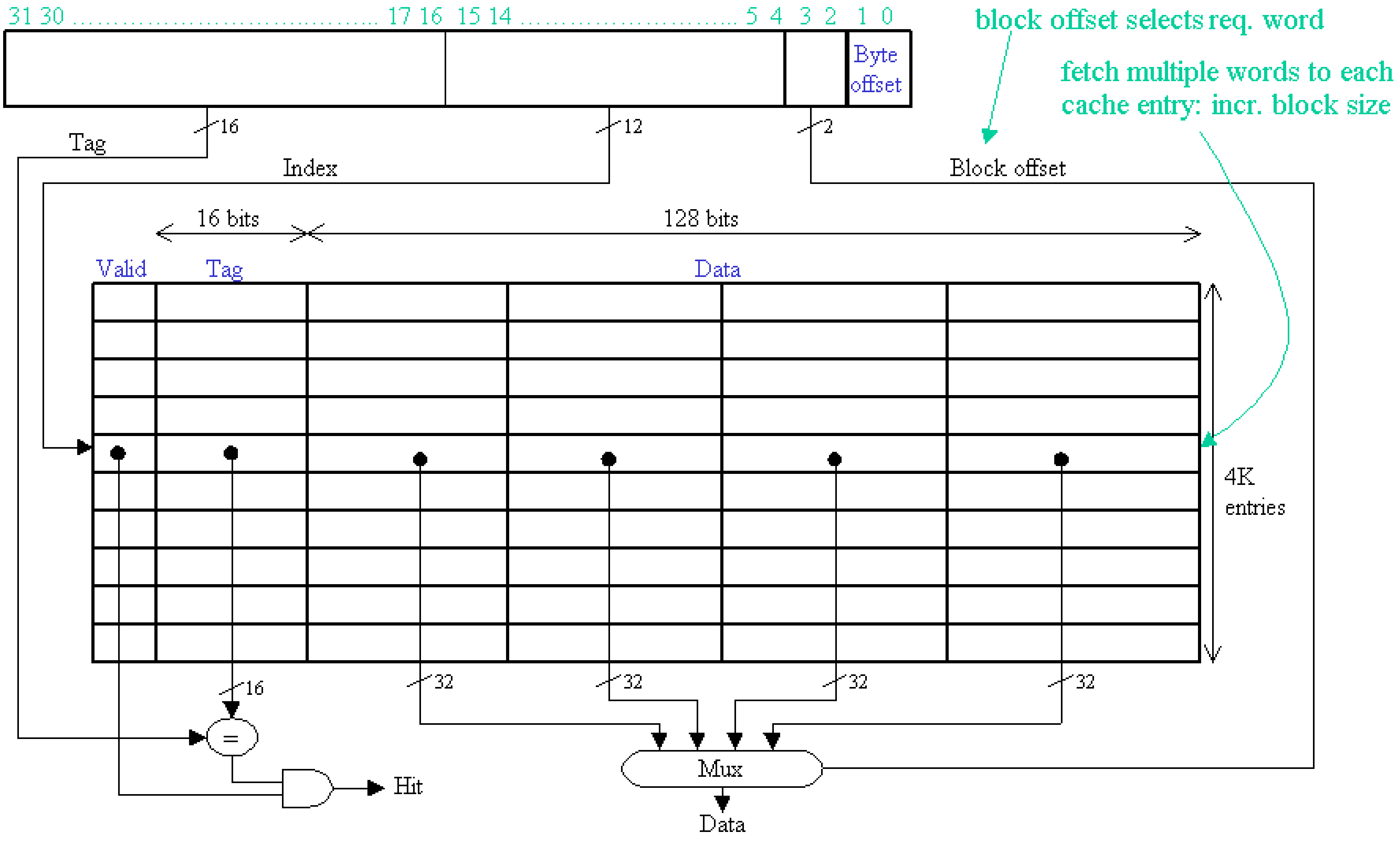
Write buffer saturation

- store buffer overflows when
  - CPU cycle time too fast with respect to memory access time
  - too many store instructions in a row
- solutions to write buffer saturation
  - use a write back cache
  - install a second-level (L2) cache
  - store compression



Exploiting spatial locality

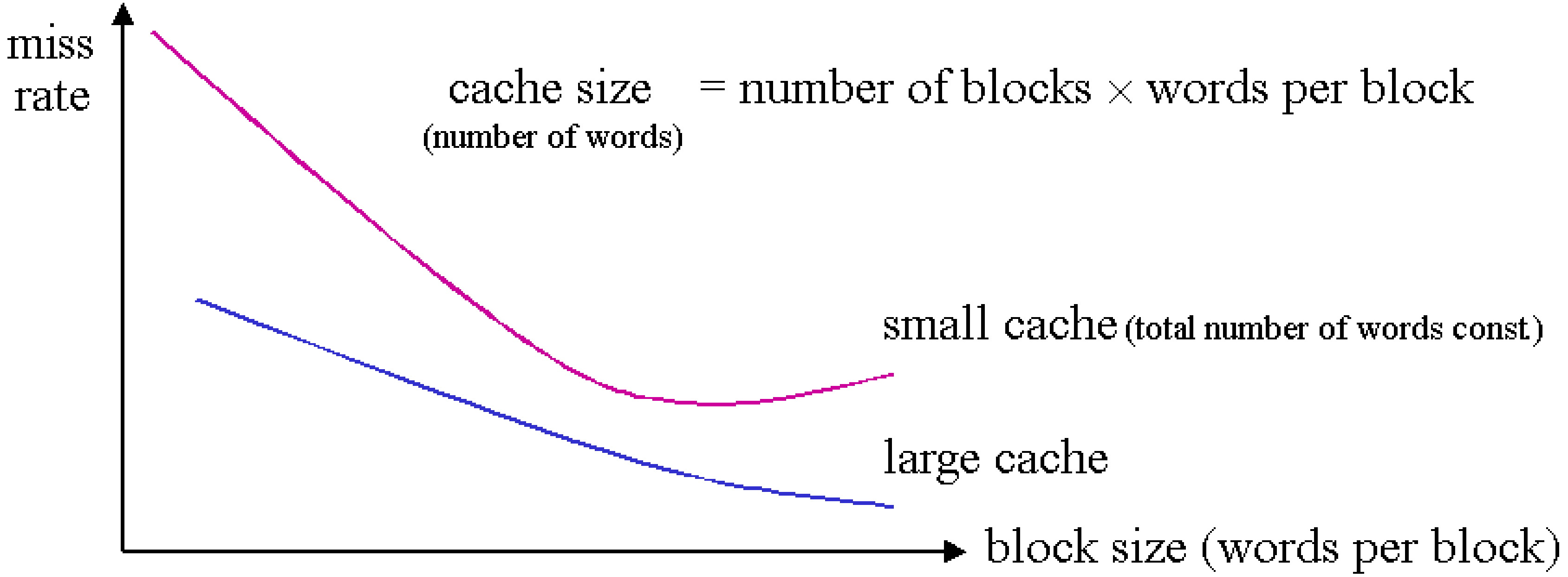
Addressing (showing bit positions)



block offset selects req. word

fetch multiple words to each cache entry: incr. block size

Block size and performance



- ↑ block size, ↓ miss rate generally (especially for instructions)
- large block for small cache: ↑ miss rate - too few blocks
- ↑ block size: ↑ transfer time between cache and main memory

Summary: Part I

- instruction set architecture: MIPS as example RISC
  - performance: execution time equation, CPI
  - arithmetic: ALU, multiplication, Booth’s algorithm, division
  - datapath: hardware compilation, single-cycle design, pipelining
  - memory hierarchy: cache, virtual memory
- next year and beyond
  - custom computing: customise design
  - advanced architecture, computing in space, …
- why exciting?
  - foundation of everything else in computing: theory + practice
  - latest: Microsoft/Amazon adopt FPGAs; Intel bought Altera
  - you can be part of it: reading, summer projects, internship, FYP…