Exploiting the Memory Hierarchy

Chapter 7 P&H

Introduction

- Desire is to have unlimited amounts of fast memory
- Too expensive
- Can exploit *locality of reference* to appear to have lots of fast memory
- Two types of locality
  - Spatial Locality
  - Temporal Locality
    - will consider temporal first

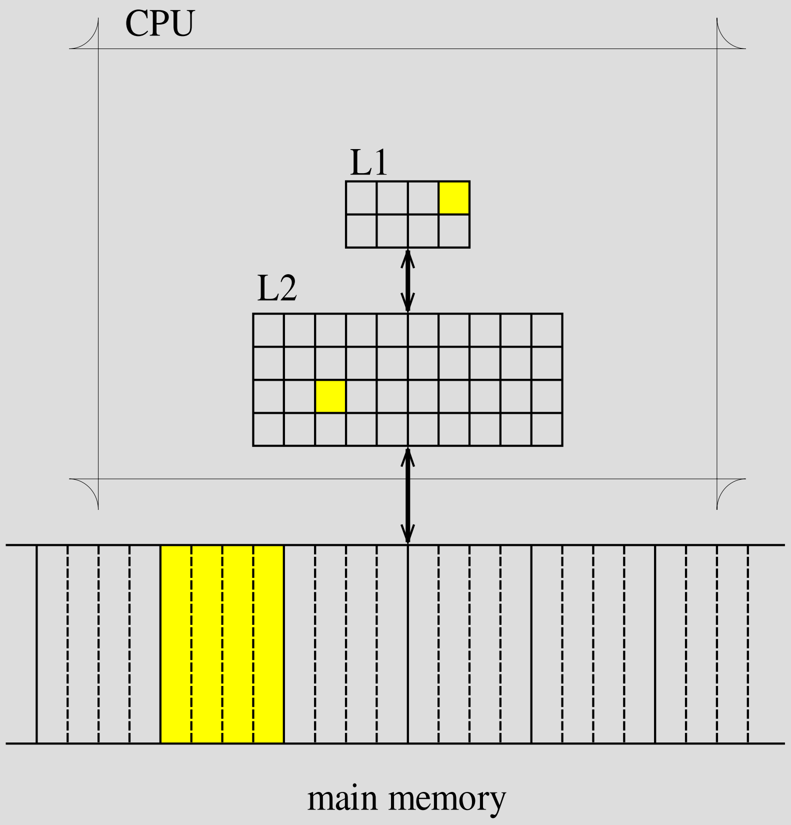
Memory Hierarchy

(very approximate)

Mem Tech	access time	$/Gbyte
SRAM	0.5ns	$30,000
DRAM	15ns	$30
Disk	20 million ns	$0.25

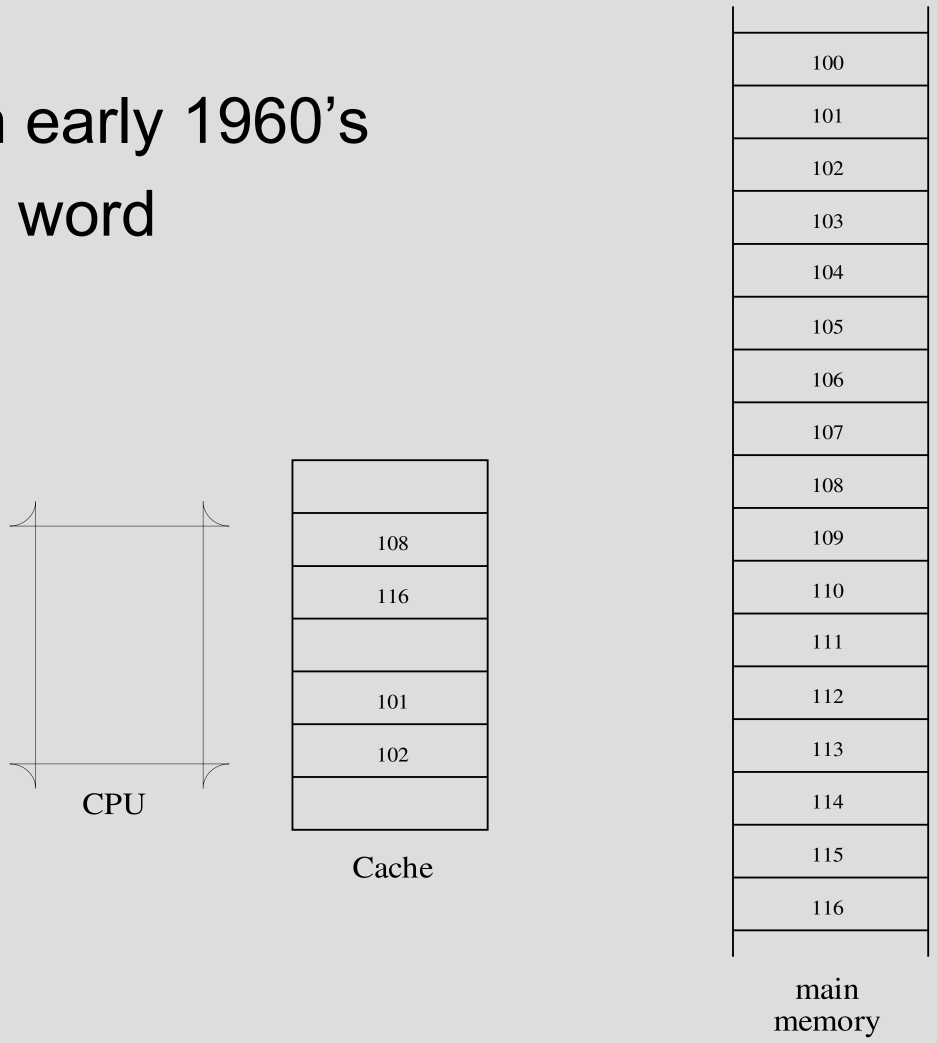
Memory Hierarchy

- minimum unit is called a block
- *hit rate* is the fraction of memory accesses found in upper level
- *miss rate* is (1 – hit rate)
- *hit time* is the time to access upper level
- *miss penalty* is the time to replace block in upper level with corresponding block from lower level



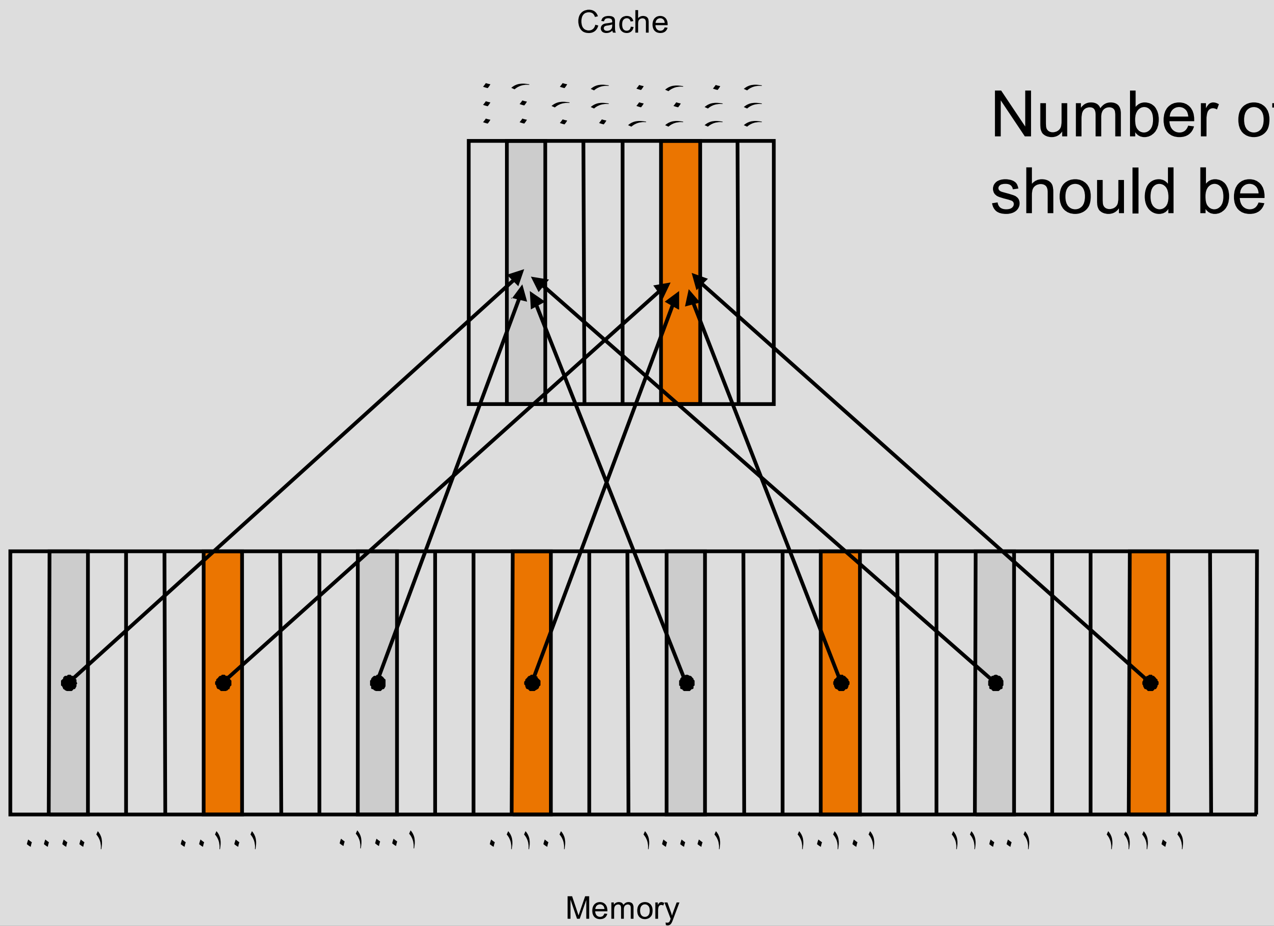
Cache Basics

- Caches first appear in early 1960's
- This cache has single word sized block
- How do we know if the requested item is in the cache?
- If it is where is it?



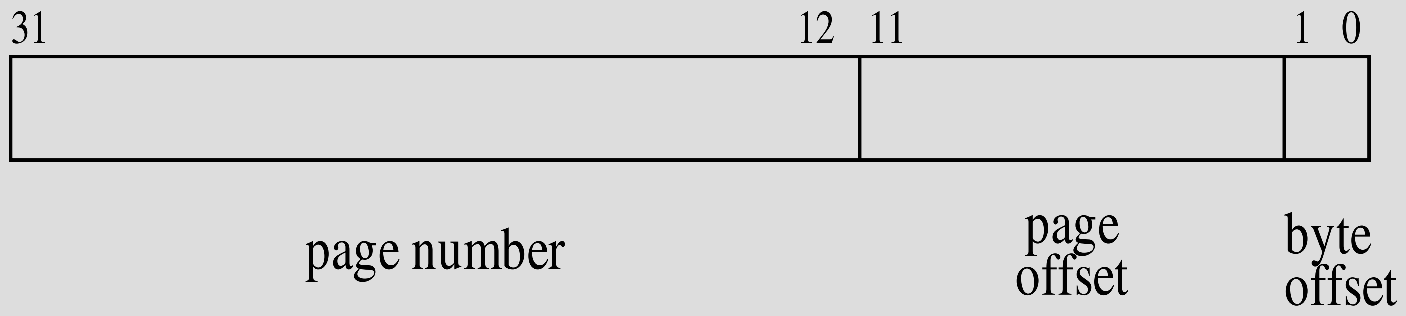
Direct Mapped Cache

- cache address = (block address) mod (No blocks in cache)

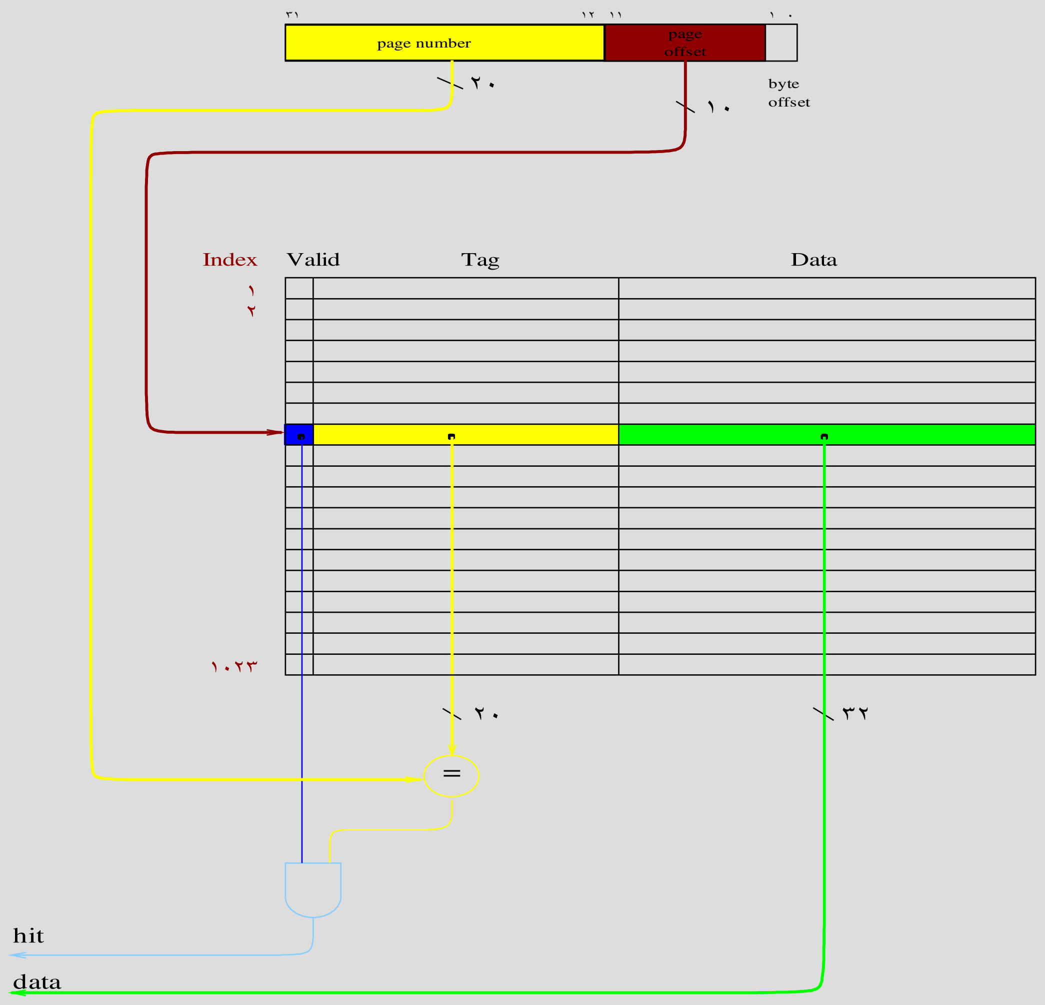


Number of blocks in cache should be a power of two

Word format



Cache With Tags



Cache Size

- The tag structure introduces a memory overhead
- How many bits in total are required for a cache that is:
  - direct mapped cache
  - stores 64 KB of data
  - has one word blocks
  - uses 32 bit address?

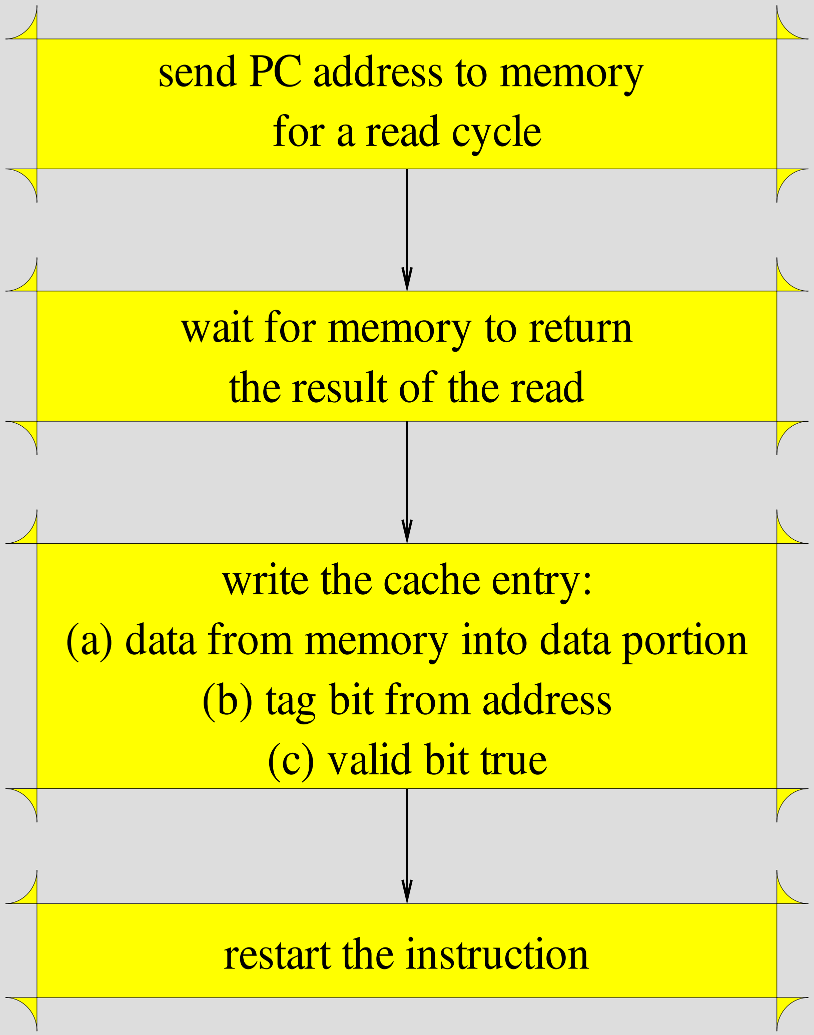
Separate Caches

- Instructions and data have separate localities
- A lot of activity in one set may invalidate the usefulness of the other
  - very bad if we stall the pipeline for cache misses
- Some machines support separate instruction and data caches

Handling Misses

- No modifications required to pipelined data-path to handle hits
- On a miss a basic approach is to stall the *entire* data-path until the value is fetched
- In addition to the CPU micro-controller there is a memory-controller

Read Miss


send PC address to memory
for a read cycle
wait for memory to return
the result of the read
write the cache entry:
(a) data from memory into data portion
(b) tag bit from address
(c) valid bit true
restart the instruction

Write Miss

- The result of a write must (eventually) get to main memory
- Simple caches “write-through” to main memory
  - On a write, write word to the cache and to main memory
  - Eases cache coherence on a multiprocessor with separate caches for each processor

Write-through performance

- Cache write-through requires a lot more time
- E.G. If we have a CPU with:
  - 1.2 CPI when there are no cache misses or writes
  - 13% of memory accesses are writes
  - Write operations take and additional 10 cycles per write
- CPI goes up to:
  - 1.2 + 0.13 * 10 = 2.5

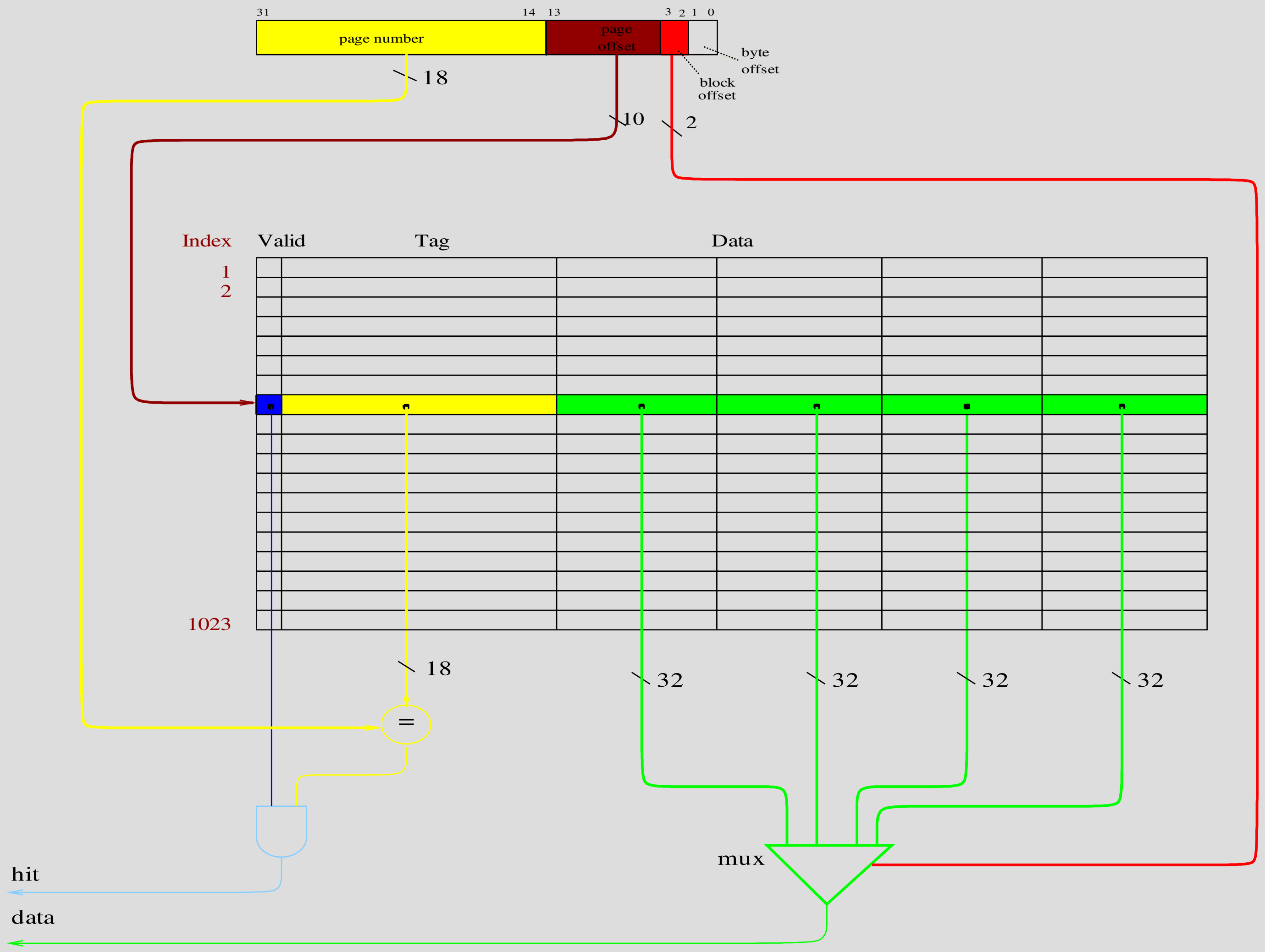
Write Buffer

- One improvement is to use a small write buffer (e.g. 4 words)
- Assumes that writes occur at a slower rate than the values can be written to memory
- Overflows can occur.
- What should happen when an overflow occurs?

Taking Advantage of Spatial Locality

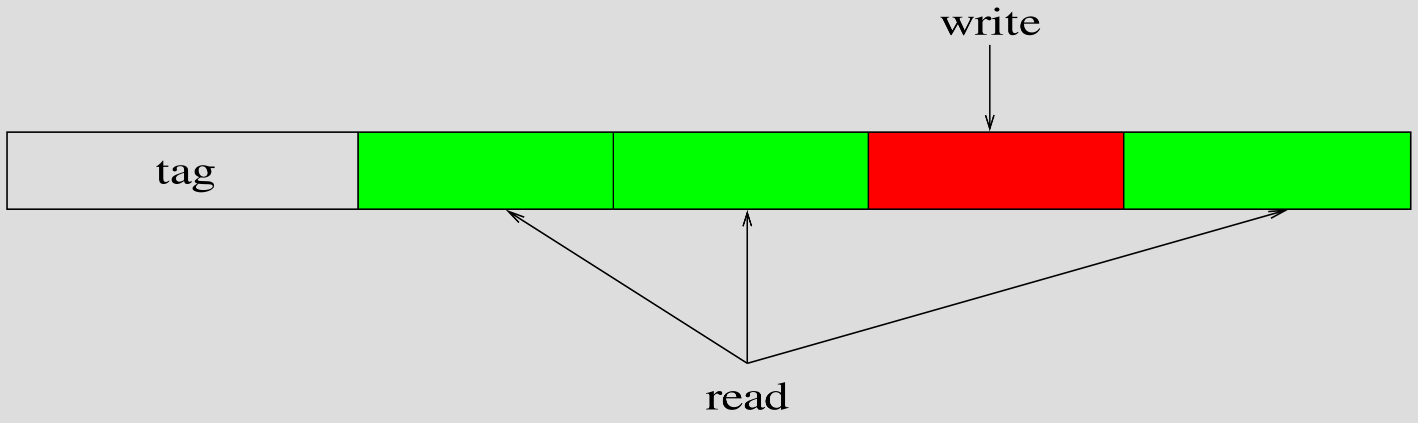
- Need a block size bigger than a word
- When a miss (read or write) occurs then we fetch the whole block

Multi-Word Cache Block

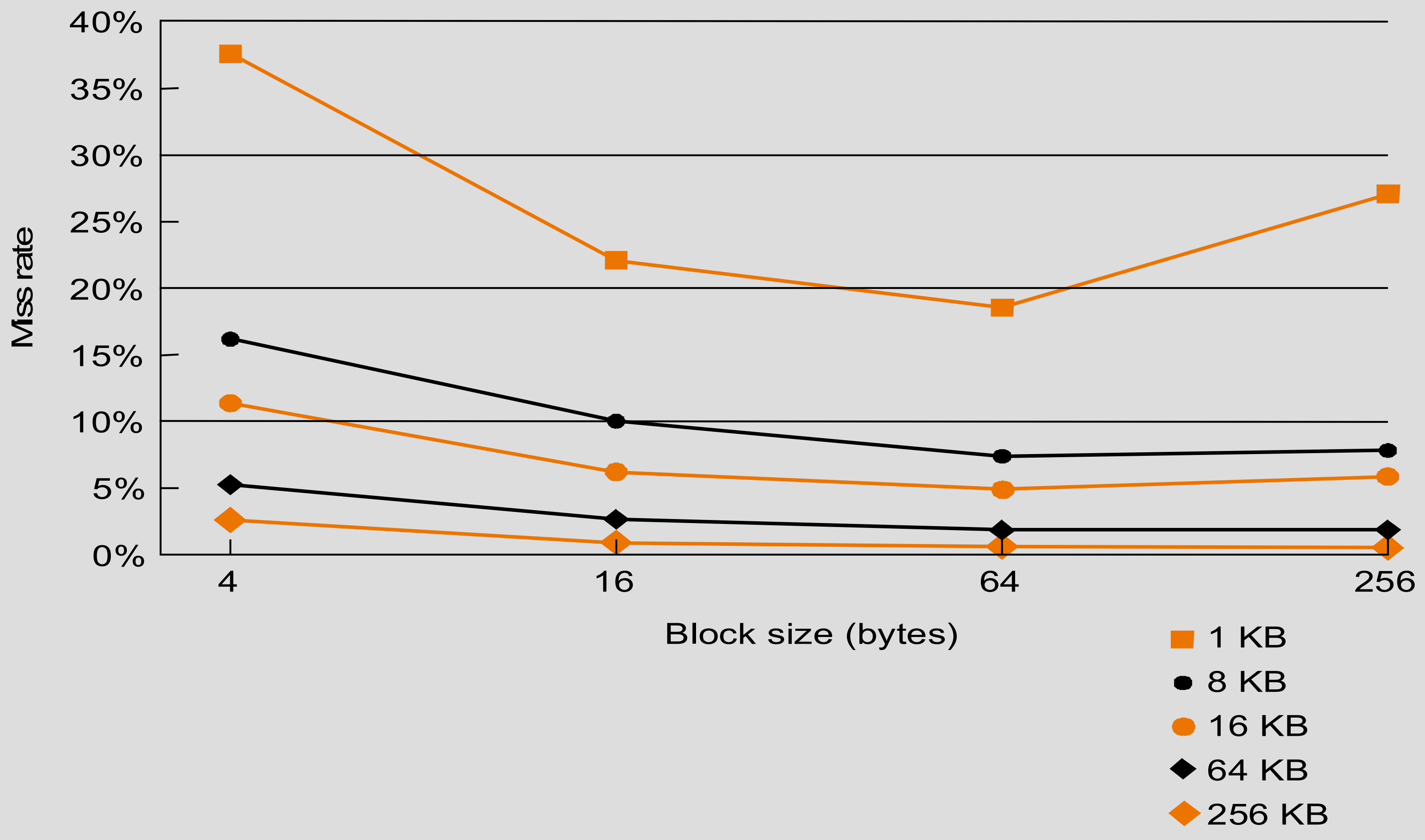


Multiword block size

- Reads are dealt with as before
- Write hits okay
- With write misses it is not possible to just write tag and data
- For a write miss have to fetch replacement block from memory and re-perform write



Miss Rate Verses Block Size



Performance of multi-block cache

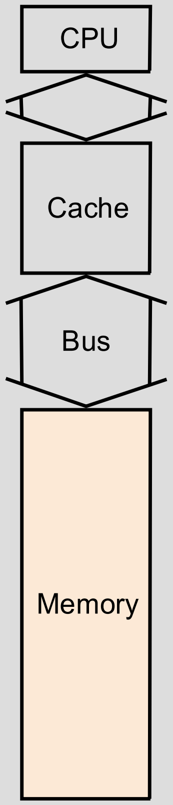
Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

Memory Interface

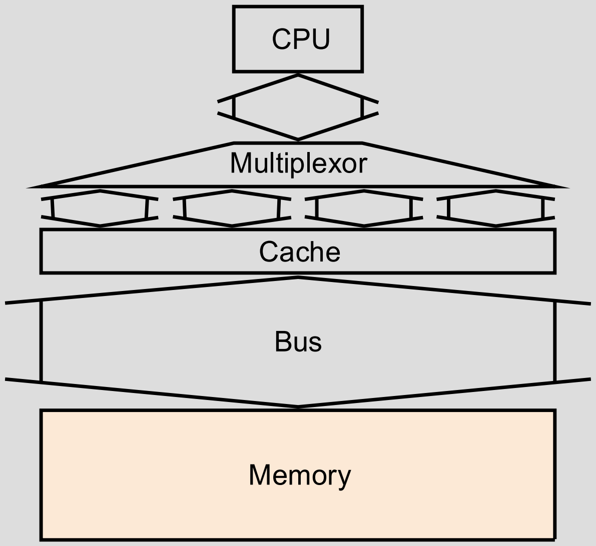
- The memory bus is much slower than the CPUs internal clock rate (maybe 10 times slower)
- DRAM is slower still
  - 1 bus cycle to send address
  - 15 bus cycles for the DRAM to read a word of data
  - 1 bus cycle to send a word of data
- A read miss for a 4 word cache block size costs:

  1 + 4*15 + 4*1 = 65 bus cycles

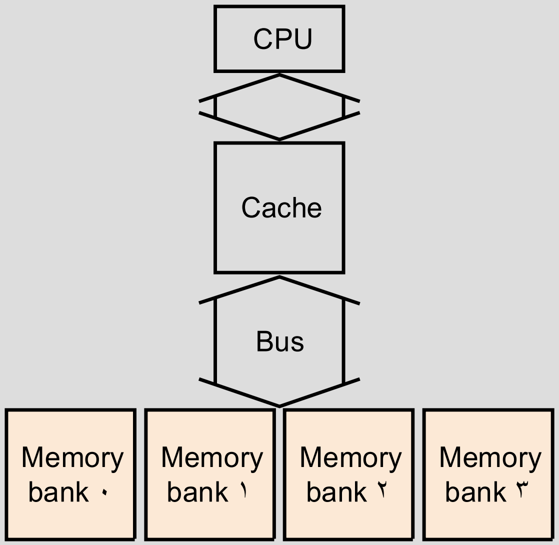
Memory Interface



a. One-word-wide
memory organization



b. Wide memory organization



c. Interleaved memory organization

Wide and Interleaved Memory Interface

- For a four word wide memory and bus, a read miss on a four word block size cache costs:
  - 1 + 1*15 + 1*1 = 17 bus cycles
- For a four word banked memory the miss cost is:
  - 1 + 1*15 + 4*1 = 20 bus cycles

Measuring & Improving Cache Performance

- Measurement and analysis of cache performance
- Look at two different techniques for improving cache performance:
  - Adding associatively to the cache to reduce the miss rate
  - Use of multi-level caches to reduce the miss penalty

Cache miss cost

$$CPU\ Time = (CPU\ execution\ clk\ cycles + Memory\ stall\ cycles) \times CCT$$

$$Memory\ stall\ cycles = read\ stall\ cycles + write\ stall\ cycles$$

$$read\ stall\ cycles = \frac{reads}{program} \times read\ miss\ rate \times read\ miss\ penality$$

$$write\ stall\ cycles = \left(\frac{writes}{program} \times write\ miss\ rate \times write\ miss\ penality \right) + write\ buffer\ stalls$$

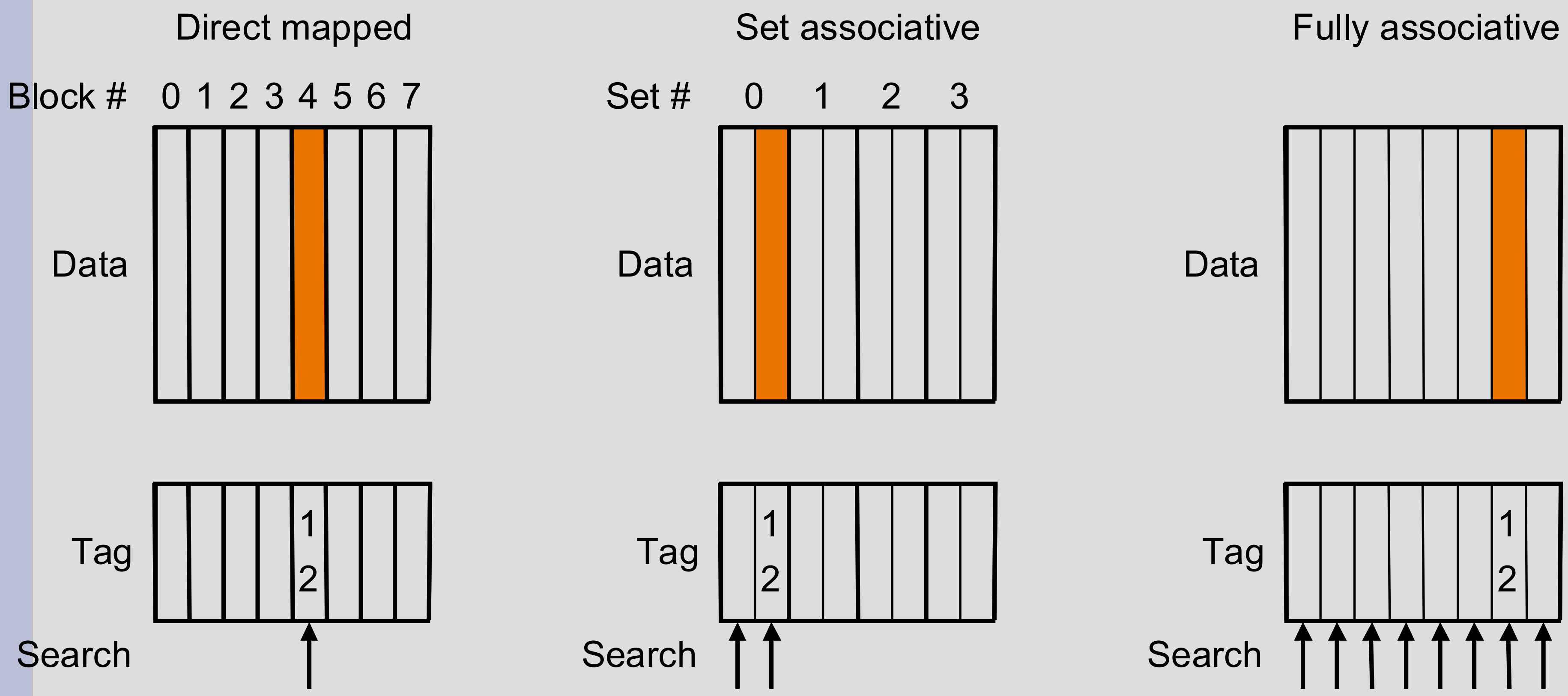
assume write buffer stalls are insignificant and that read and write penalties are the same

$$memory\ stall\ cycles = \frac{mem\ accesses}{program} \times miss\ rate \times miss\ penality$$

Improving Cache Utilisation

- Direct mapping means some blocks are evicted from the cache quickly
- A fully associative caches allow blocks to be placed anywhere in the cache
  - Have to search every tag field for every memory access
- Set associative cache allows blocks to be placed in a fixed number of locations in the cache
  - an *n-way set associative cache* allows a block to be placed in one of *n* locations in the cache

Associative Cache

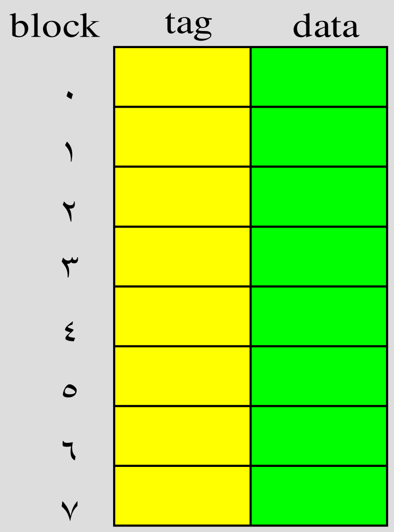


Associative Cache

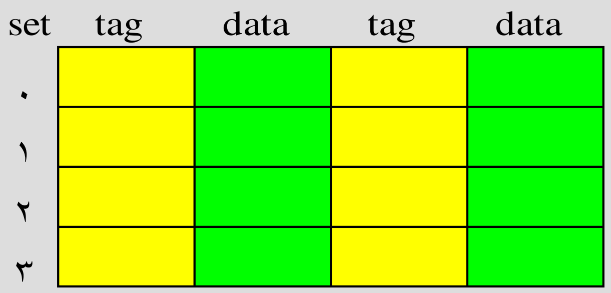
- All caches can be considered as being set associative
- Increasing associatively tends to decrease the miss rate

Associative Caches

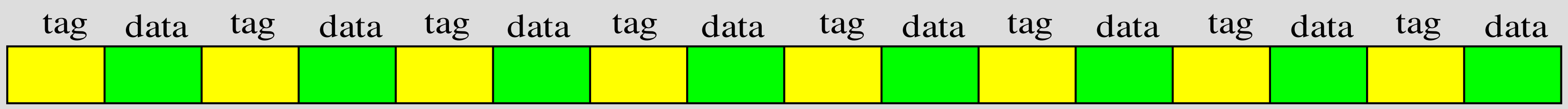
one way set associative
(direct mapped)



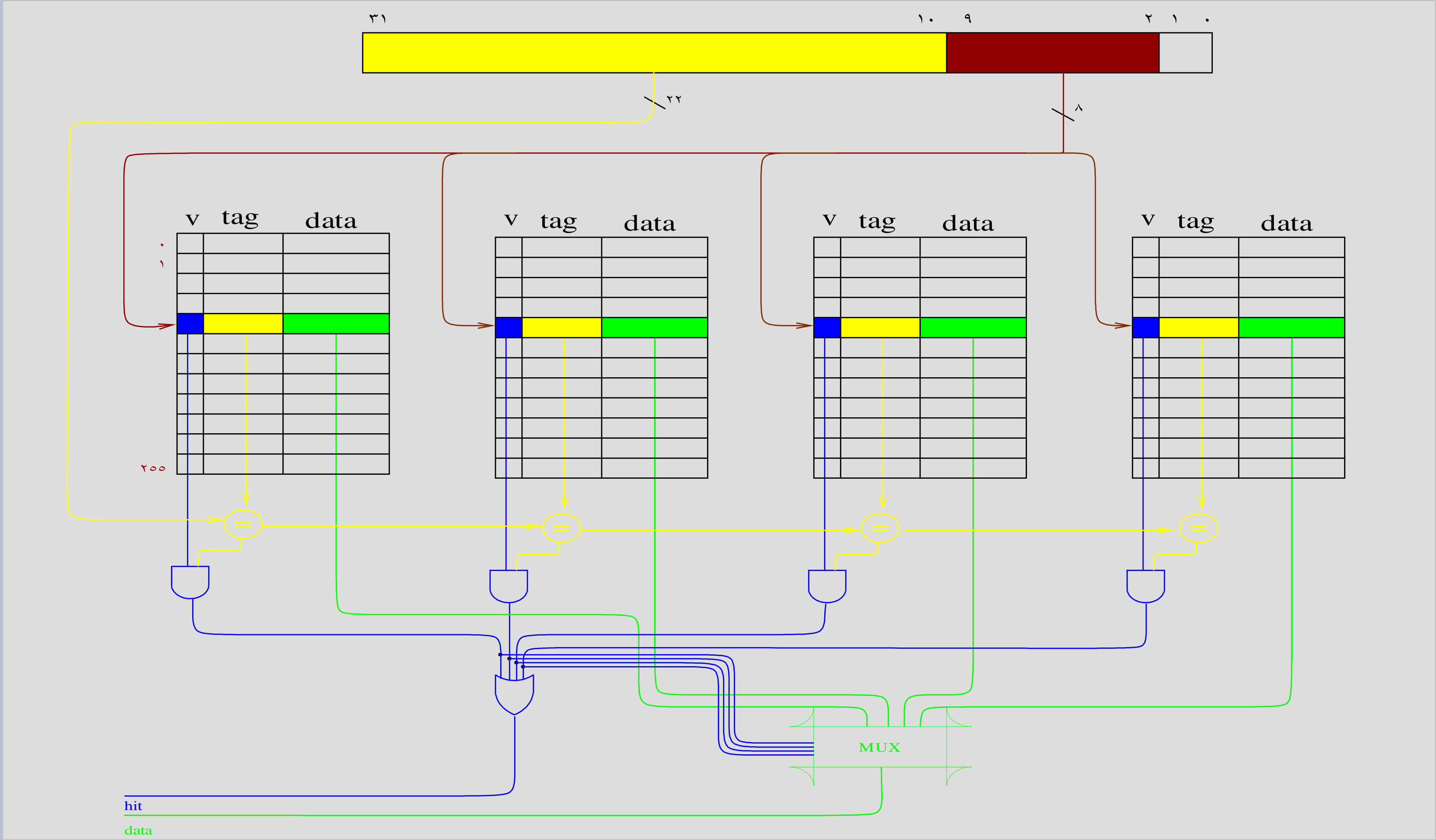
two way set associative



eight way set associative
(fully associative)



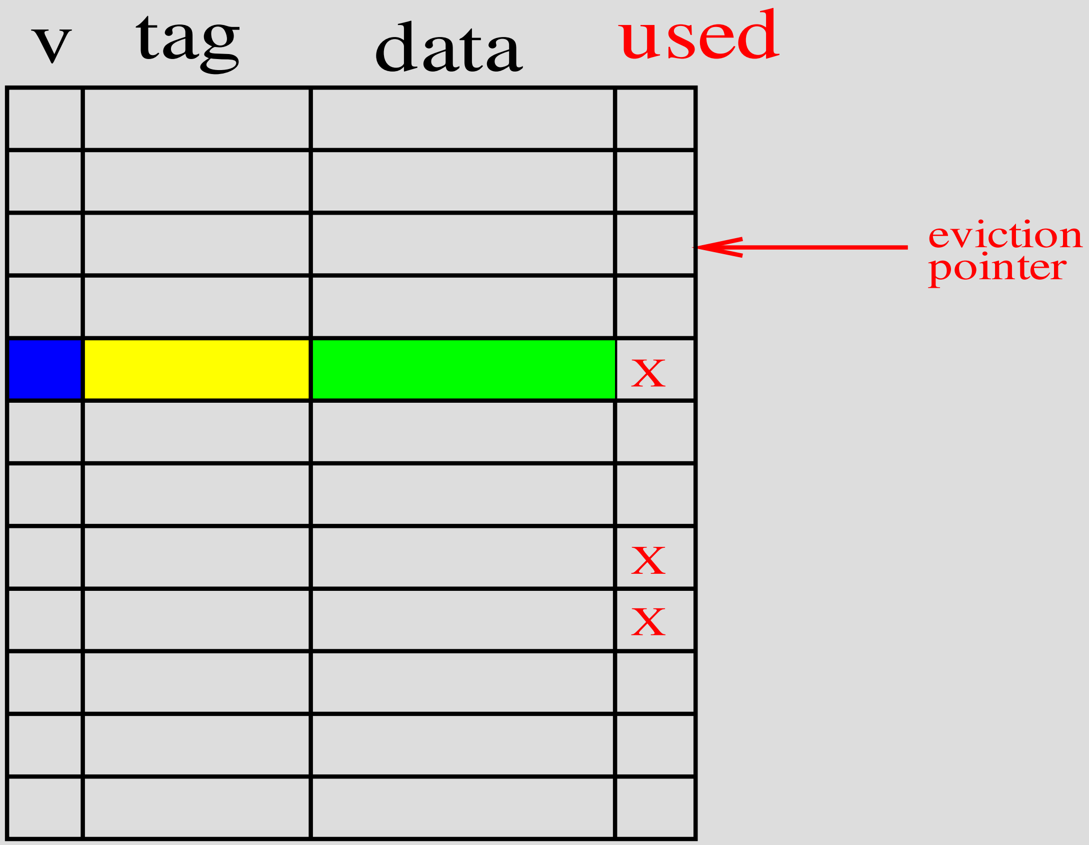
Four-way Set Associative Cache



Replacement Strategies

- When the cache is full one or more entries need to be removed to allow new data to be stored.
- Least Recently Used (LRU) is the most common strategy
- Clock Scan is often used as an approximation to LRU

Clock Scan



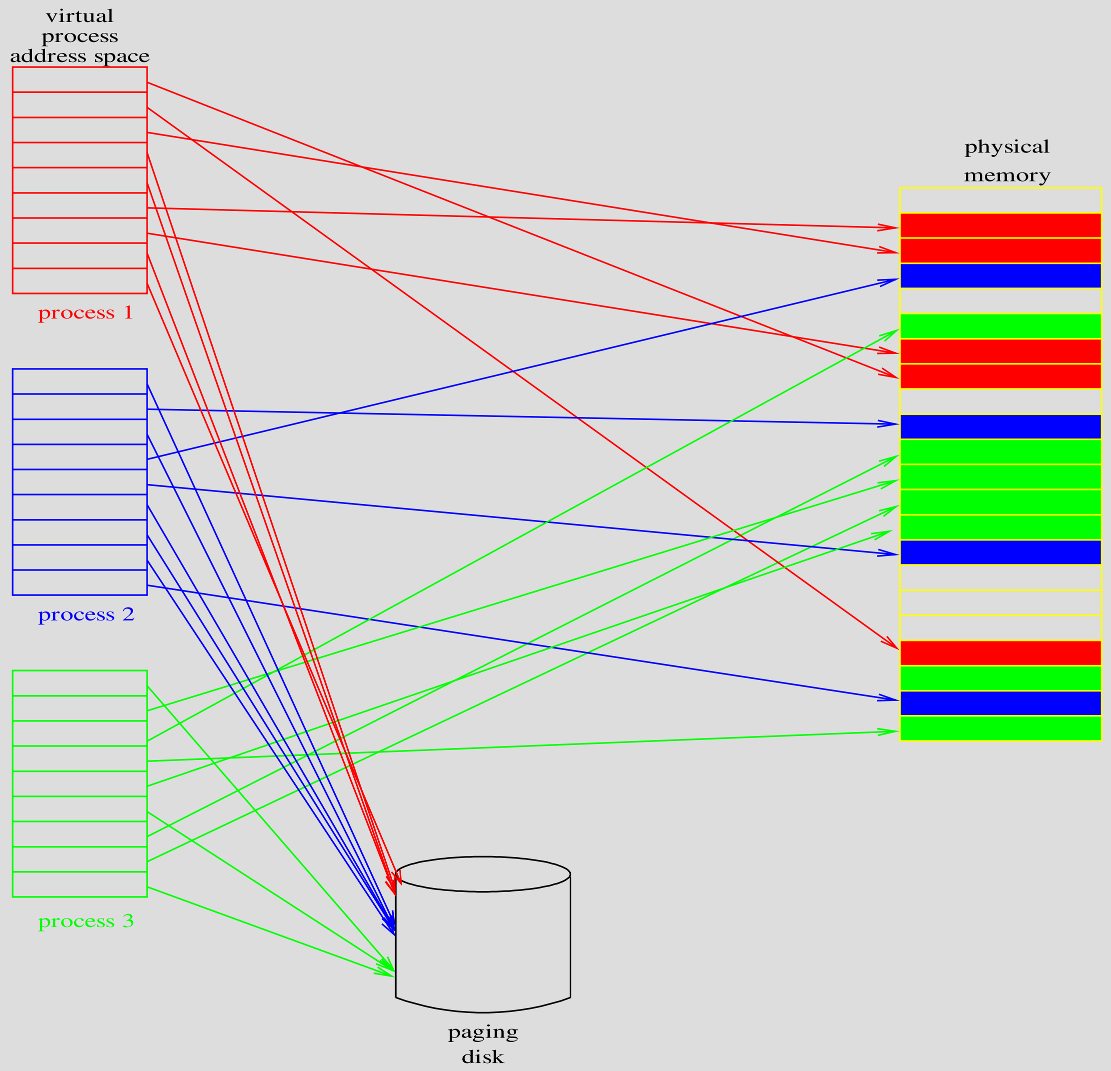
Virtual Memory

- Main memory can act as a cache for secondary storage
- Motivation:
  - Allow programs to use more memory that there is available
    - transparent to programmer
    - c.f. overlays
  - Allow multiple programs concurrently
    - non-active part of programs reside in secondary storage
    - active portions of many programs reside in memory
    - active portion of current program in cache

Virtual Memory

- With multiple programs sharing memory have to deal with:
  - program relocation
  - protection
- Give each process its own (virtual) address space
- When process accesses memory
  - translate virtual address to physical address

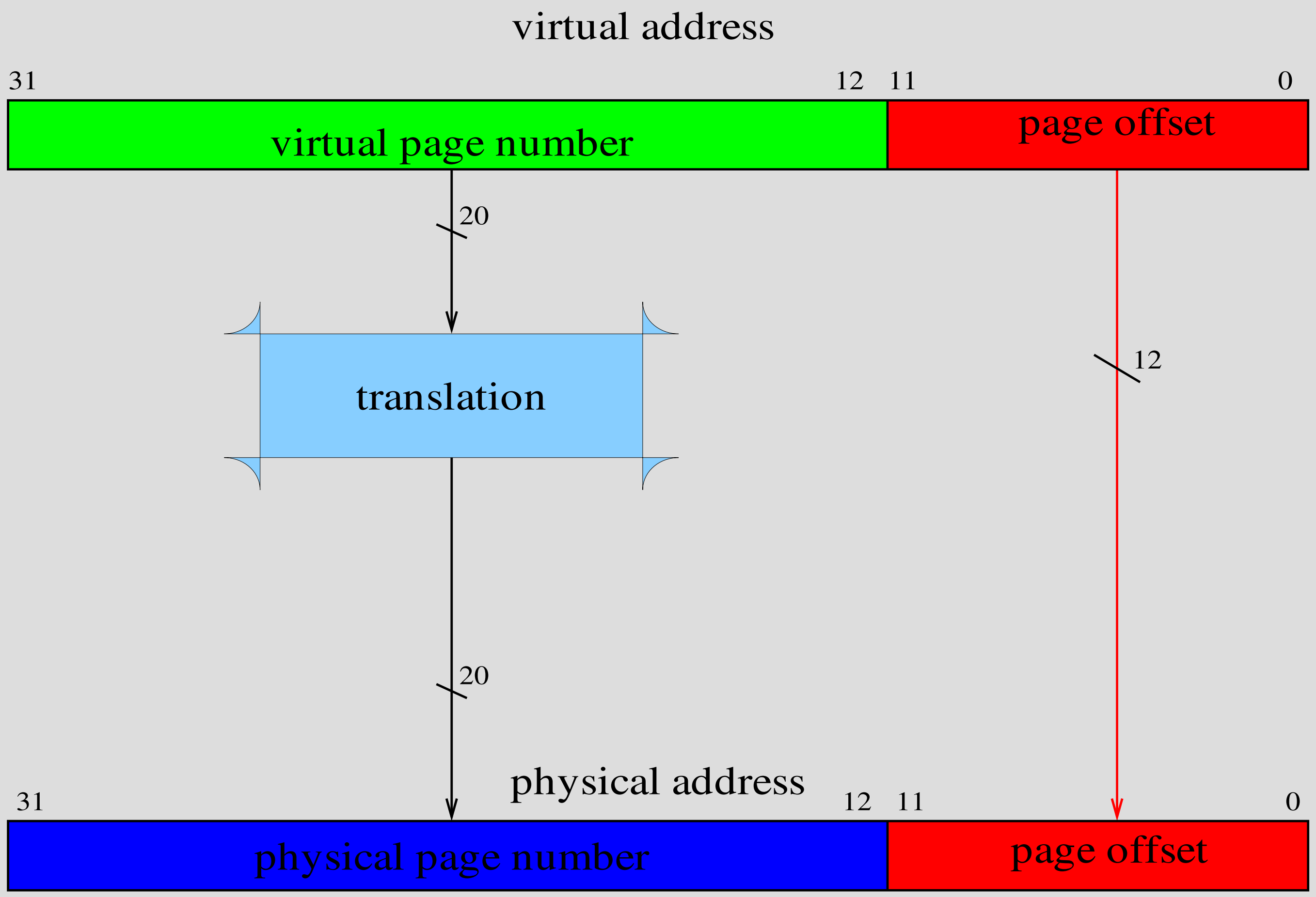
Virtual Memory



Definitions

- main concepts similar to caches however different terminology used
  - virtual memory block => page
  - virtual memory miss => page fault

Address Translation



Design Choices

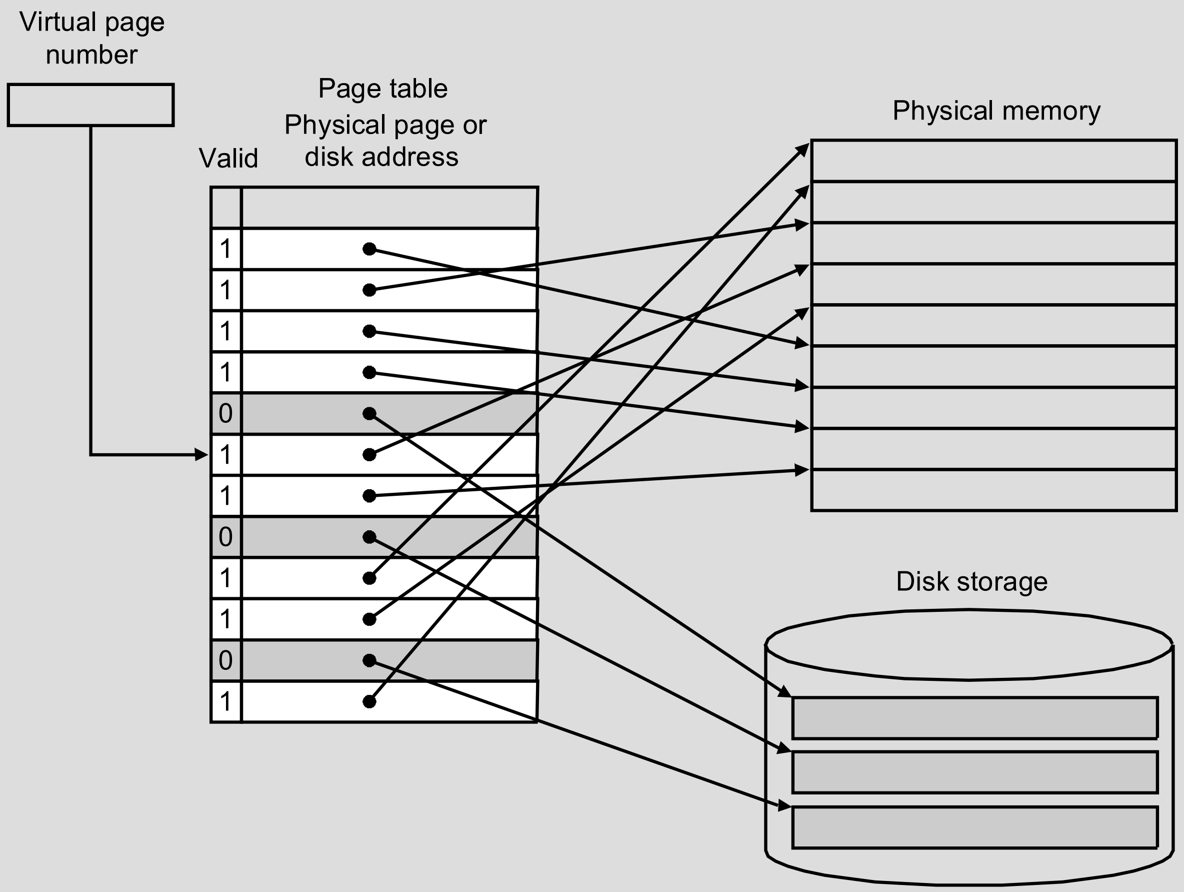
- Page faults : when a page not in memory then have to fetch from disk
  - can take millions of cycles

- Minimise miss penalty:
  - make pages fairly large (4KB to 64KB)
  - reduce page fault rate:
    - High levels of associatively

Placing/Finding a page

- Want Full associatively
  - impractical to search all pages in memory
- Use page tables to map virtual addresses to physical addresses
- Each program has its own page table
- Page tables reside in memory
  - reads/writes to main memory require two accesses:
    - one to get page table entry
    - one to perform data transfer

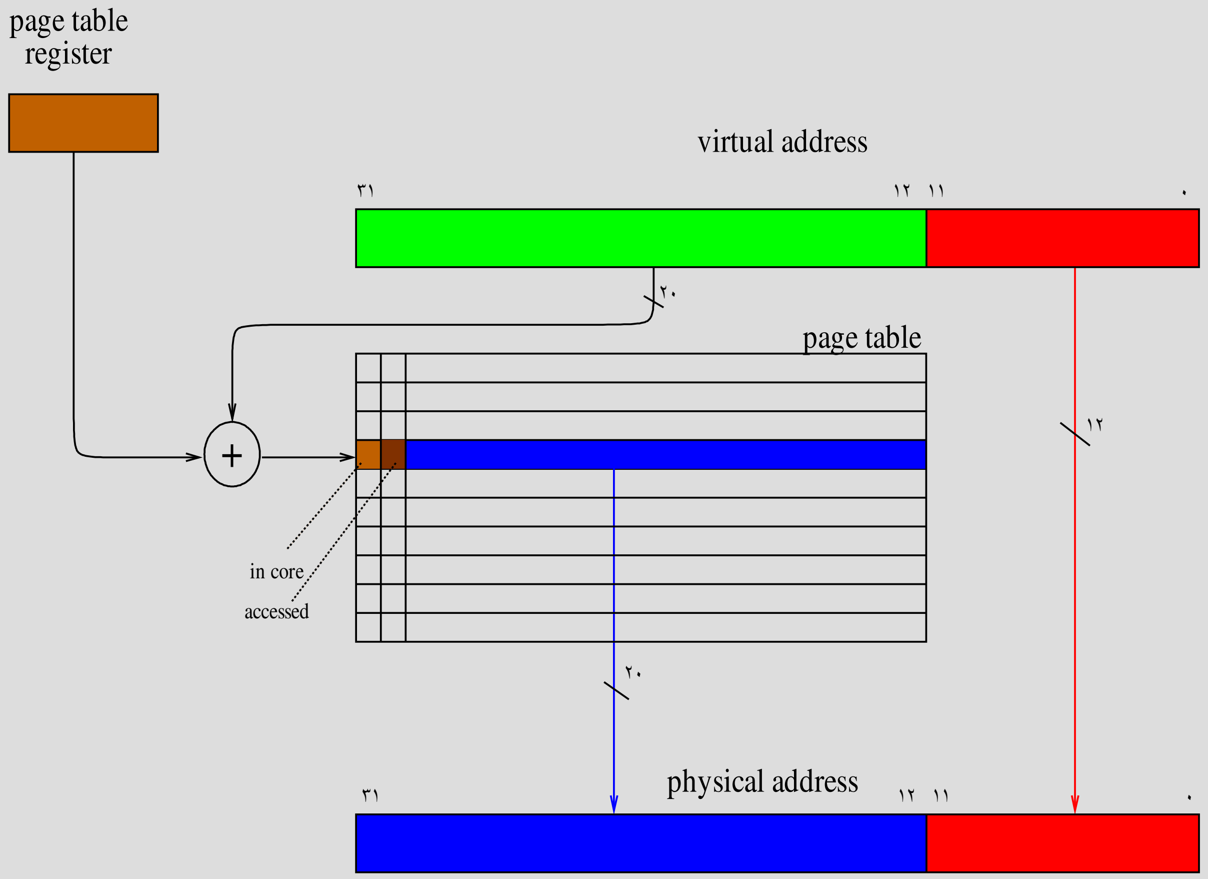
Page Tables



Virtual page number
Page table
Physical page or disk address
Valid
1
1
1
1
0
1
1
0
1
1
0
1
Physical memory
Disk storage

Page Tables

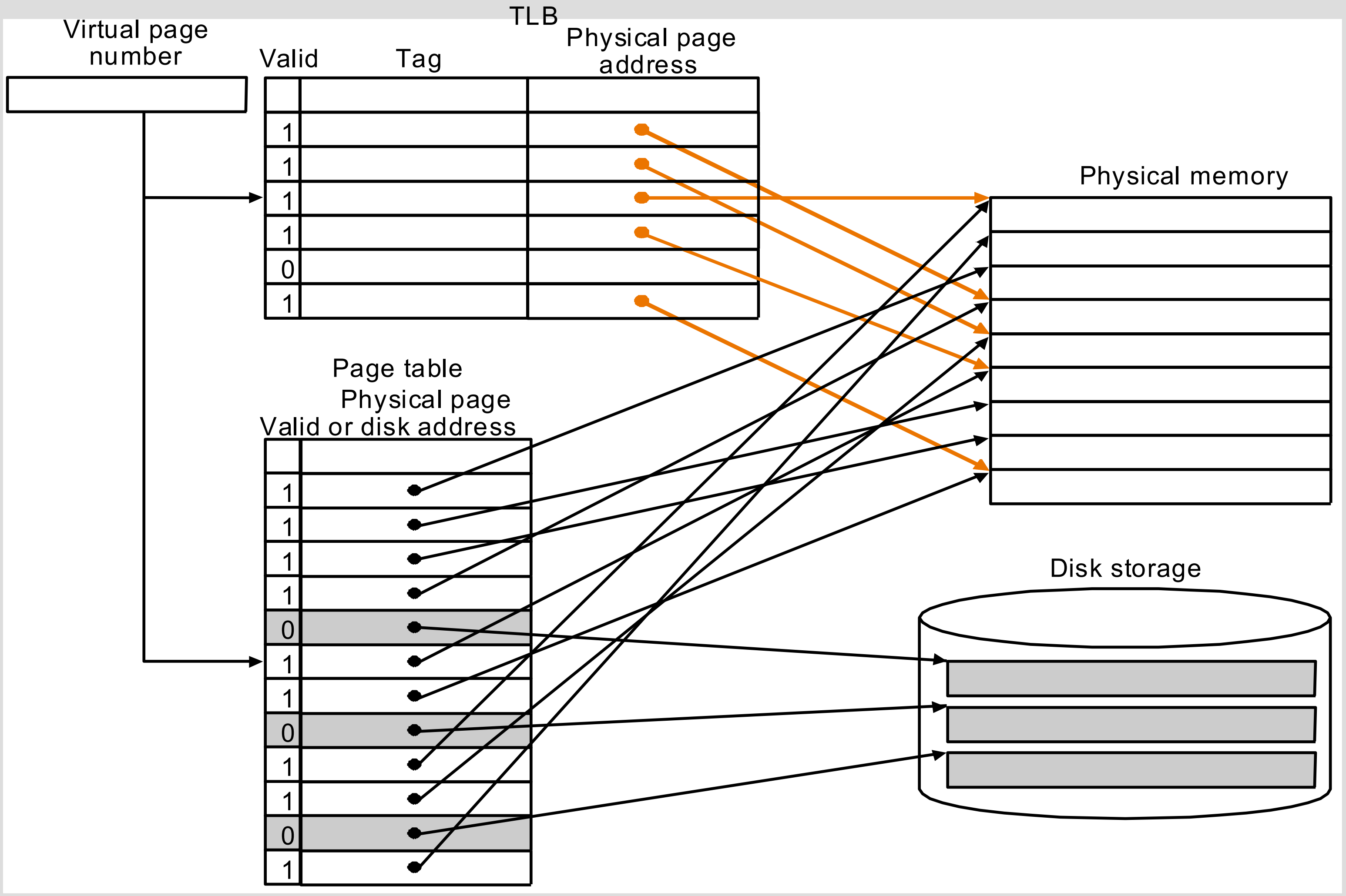
- Base address of current page table held in the *page table register*
- A processes' state is defined by its
  - PC
  - Registers
  - Page table



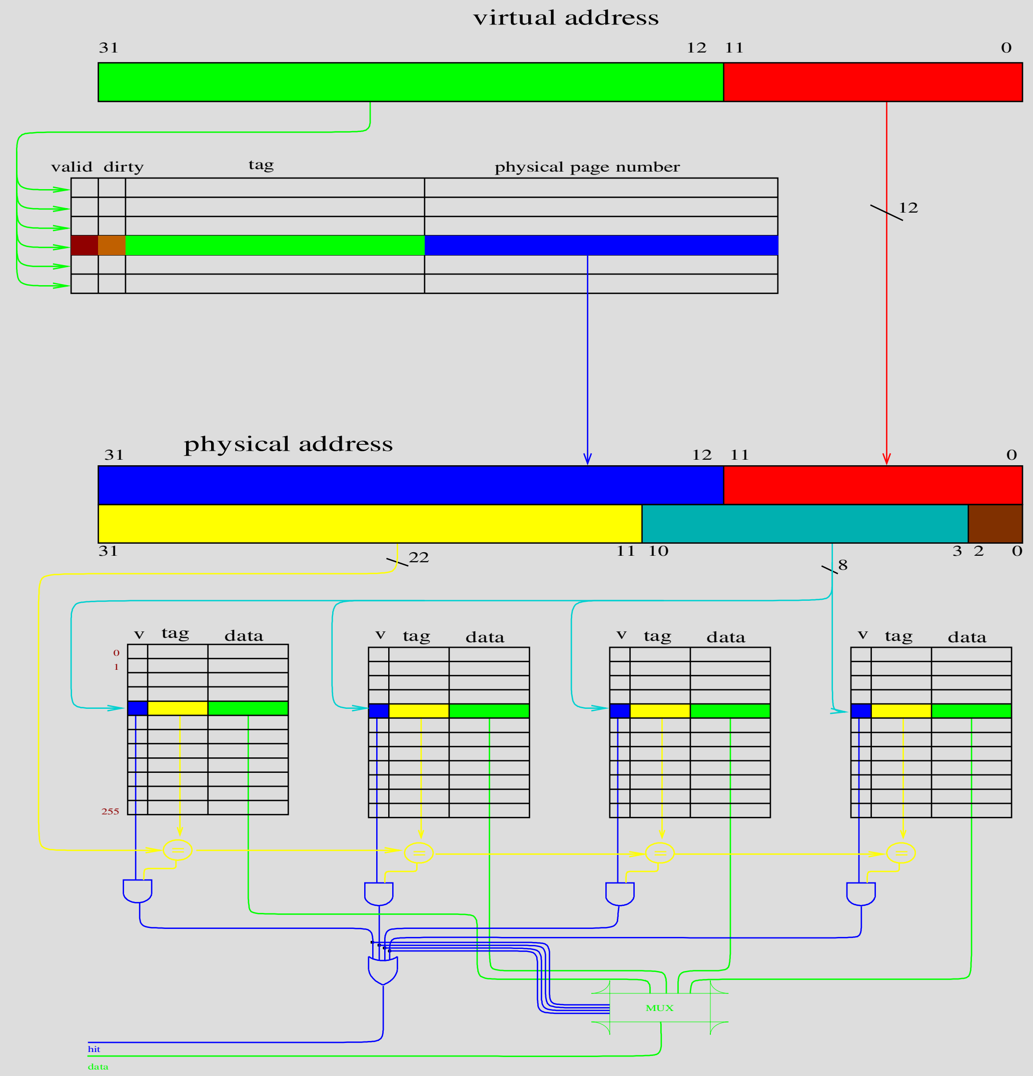
TLBs

- Accessing the page table slows memory accesses
- Use a “Translation Look-aside Buffer”
  - cache of page table mappings
- Typical values for a TLB might be:
  - TLB size 32 – 4096 entries
  - Block Size: 1 – 2 page table entries
  - Hit time: 0.5 – 1 clock cycles
  - Miss penalty: 10 – 30 clock cycles
  - Miss rate: 0.01 – 1%

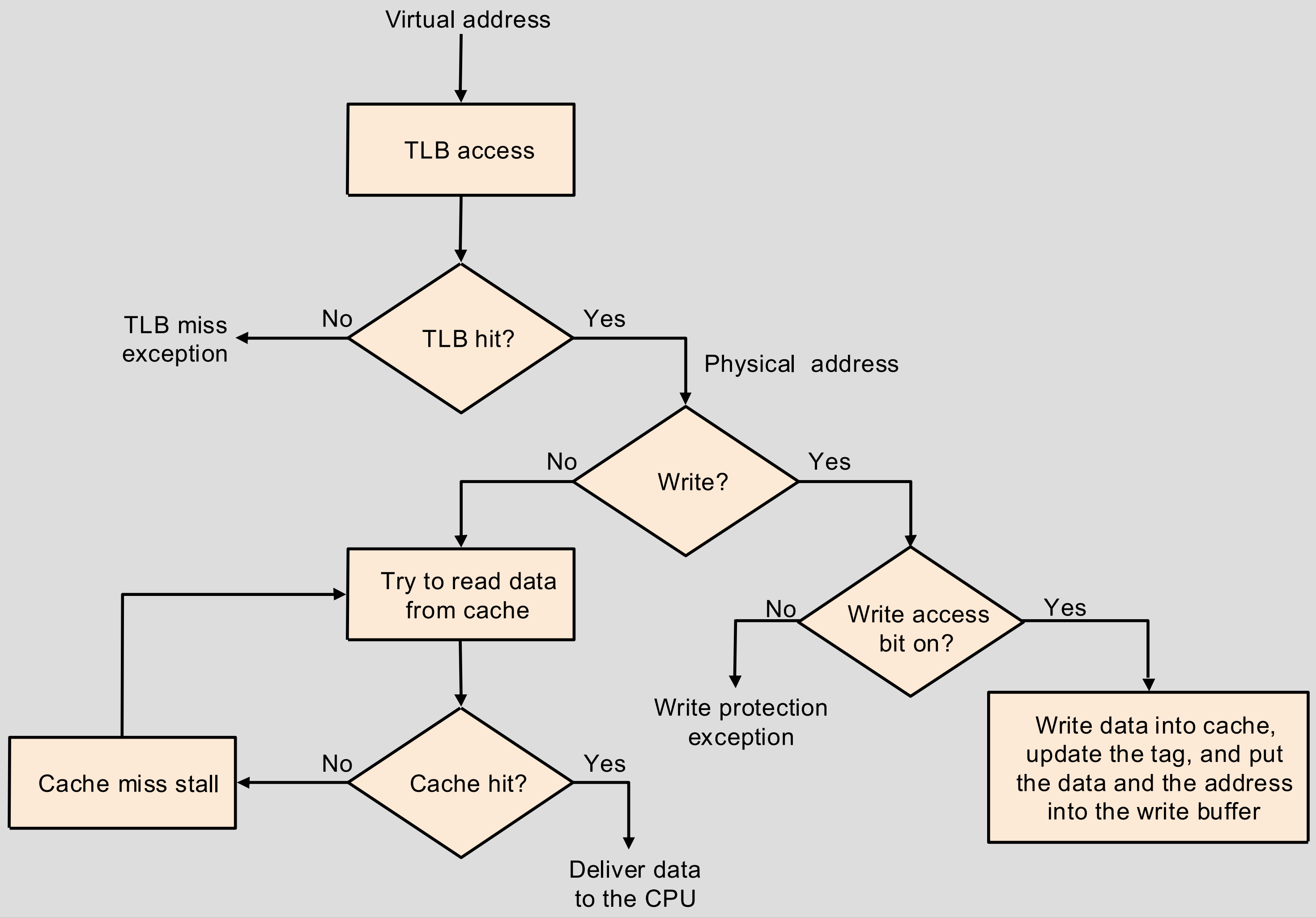
TLB Operation



TLB and Cache



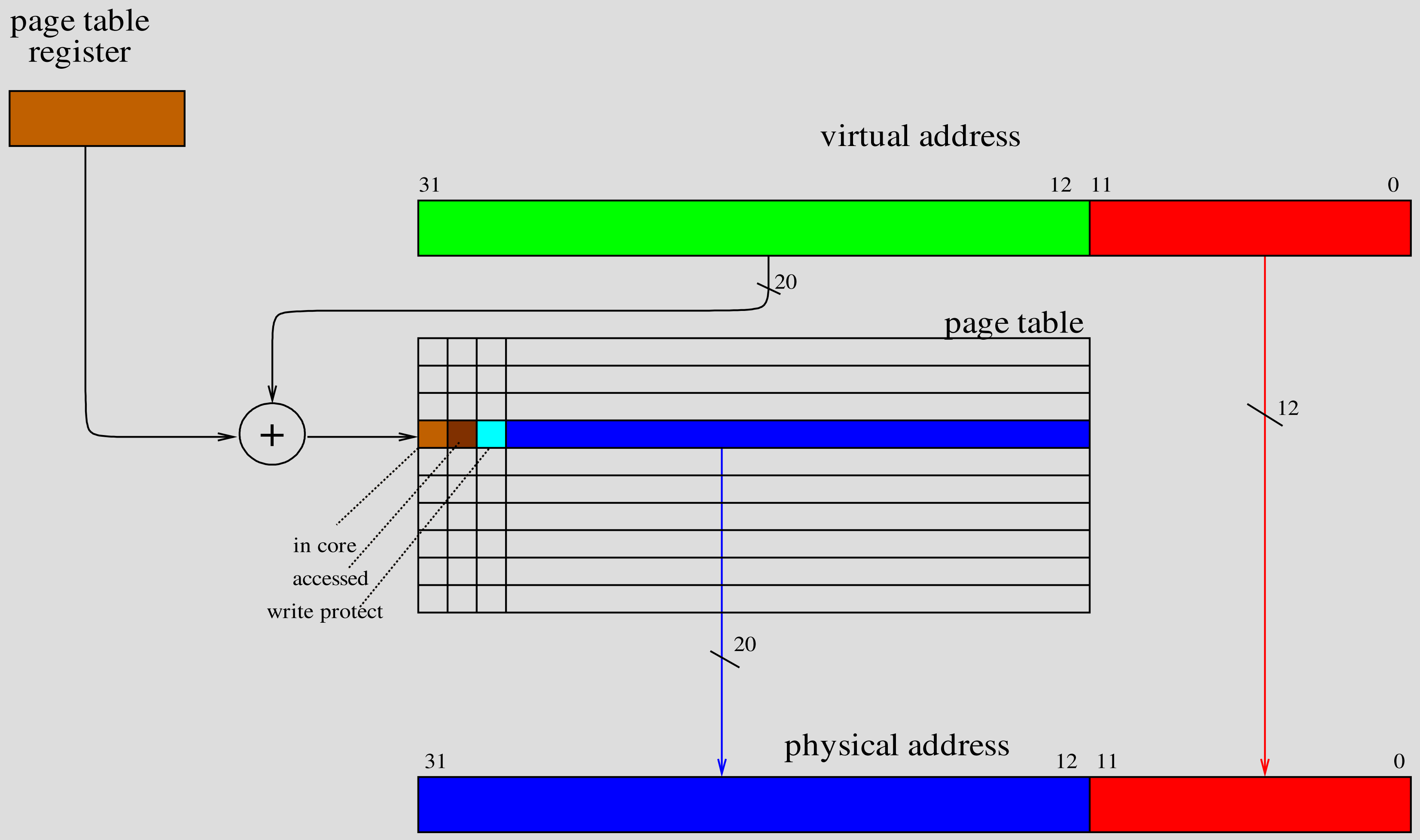
TLB and Cache



Virtual Memory Protection

- Most CPUs have a supervisor (OS) and user mode of operation.
  - Change modes on an interrupt
- Page Table Register only accessable in supervisor mode
- If a physical page is not pointed to from the page table then it can not be accessed
- Some accessable pages may be read only
  - write protect bit in the page table or segment table

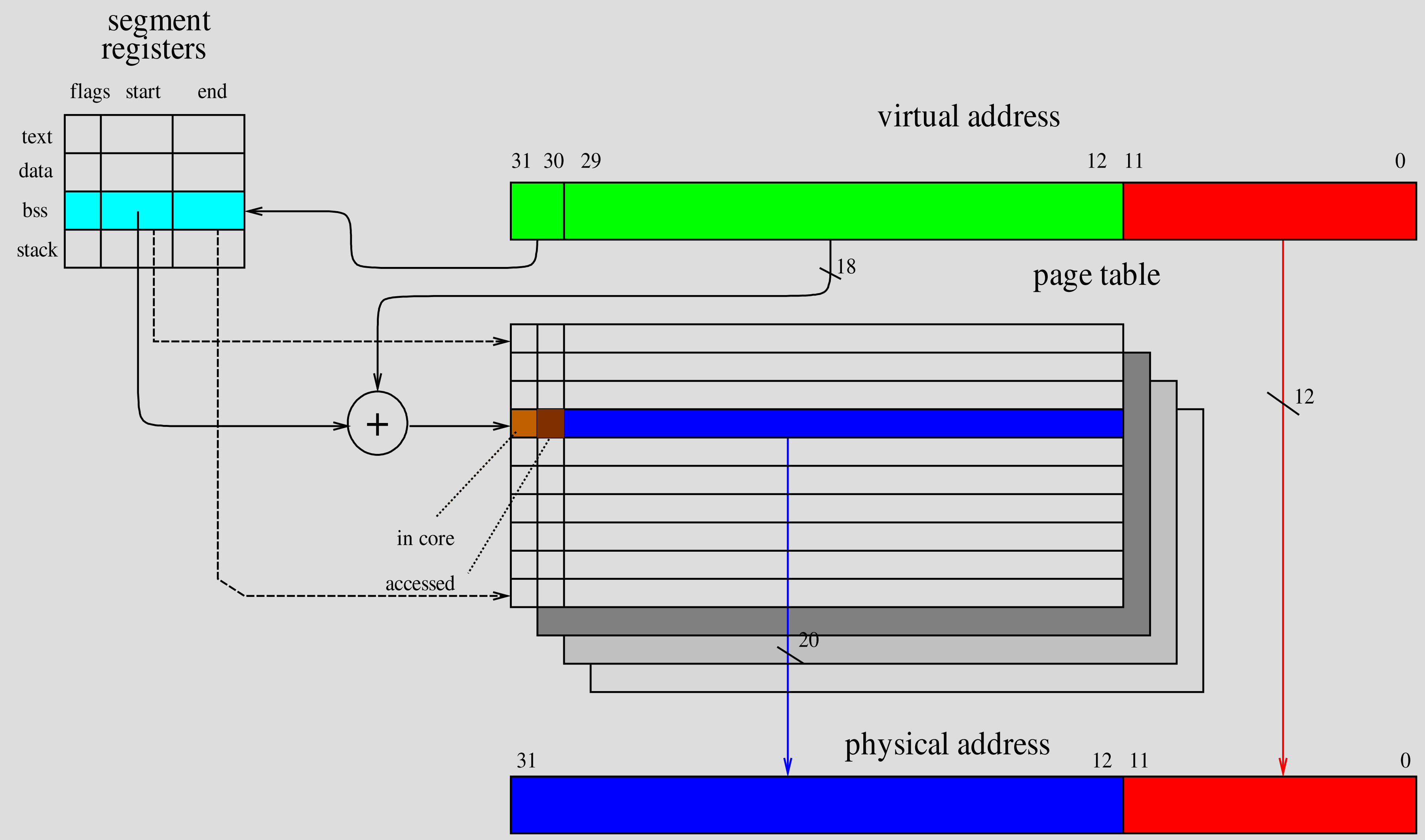
Write Protect Bit



Segments

- Programs have different types of memory
  - Program (`text`)
  - Initilised Data (`data`)
  - Uninitilsed Data (`bss`)
  - Stack (`stack`)
- Have a seperate segment for each
  - Different OS characteristics
    - Bss allocated but not loaded at process start
  - Different protection for each
    - Text read only

Segments



segment registers
flags
start
end
text
data
bss
stack
virtual address
31
30
29
12
11
0
18
page table
12
+
in core
accessed
20
physical address
31
12
11
0

Modern Systems

- Very complicated memory systems:

Characteristic	Intel Pentium Pro	PowerPC 604
Virtual address	32 bits	52 bits
Physical address	32 bits	32 bits
Page size	4 KB, 4 MB	4 KB, selectable, and 256 MB
TLB organization	A TLB for instructions and a TLB for data Both four-way set associative Pseudo-LRU replacement Instruction TLB: 32 entries Data TLB: 64 entries TLB misses handled in hardware	A TLB for instructions and a TLB for data Both two-way set associative LRU replacement Instruction TLB: 128 entries Data TLB: 128 entries TLB misses handled in hardware

Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split intruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through