
Cache Control

Philipp Koehn

4 April 2018



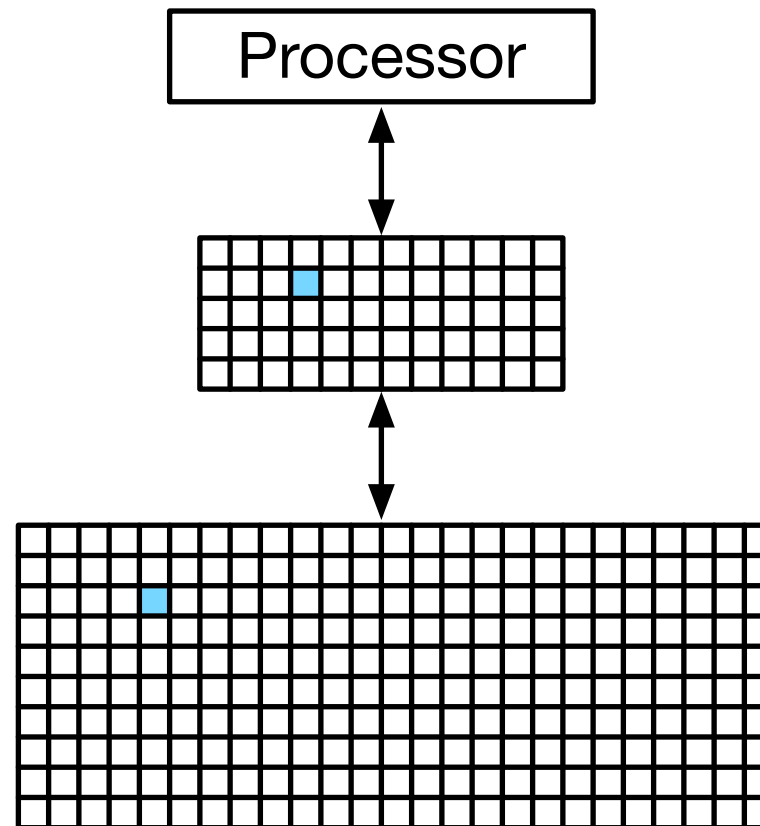
Memory Tradeoff



1

- Fastest memory is on same chip as CPU
... but it is not very big (say, 32 KB in L1 cache)
- Slowest memory is DRAM on different chips
... but can be very large (say, 256GB in compute server)
- Goal: illusion that large memory is fast
- Idea: use small memory as cache for large memory

Simplified View



Smaller memory mirrors some of the large memory content

Direct Mapping



- Idea: keep mapping from cache to main memory simple

⇒ Use part of the address as index to cache

- Address broken up into 3 parts
 - memory position in block (offset)
 - index
 - tag to identify position in main memory
- If blocks with same index are used, older one is overwritten

Direct Mapping: Example



- Main memory address (32 bit)

0010 0011 1101 1100 0001 0011 1010 1111

- Block size: 256 bytes (8 bits)
- Cache size: 1MB (20 bits)

0010 0011 1101	1100 0001 0011	1010 1111
Tag	Index	Offset

Cache Organization

- Mapping of the address

0010 0011 1101	1100 0001 0011	1010 1111
Tag	Index	Offset

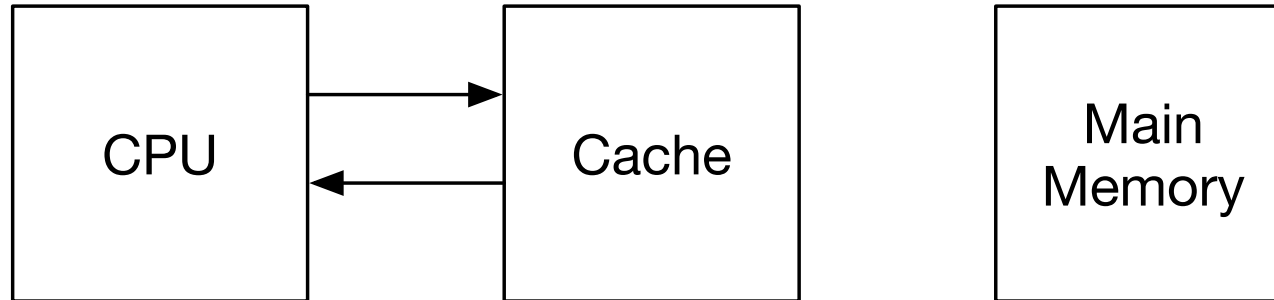
- Cache data structure

Index 4096 slots	Tag (12 bits)	Valid (1 bit)	Data 256 bytes
---------------------	------------------	------------------	-------------------

000			
001			
002			
...
fff			

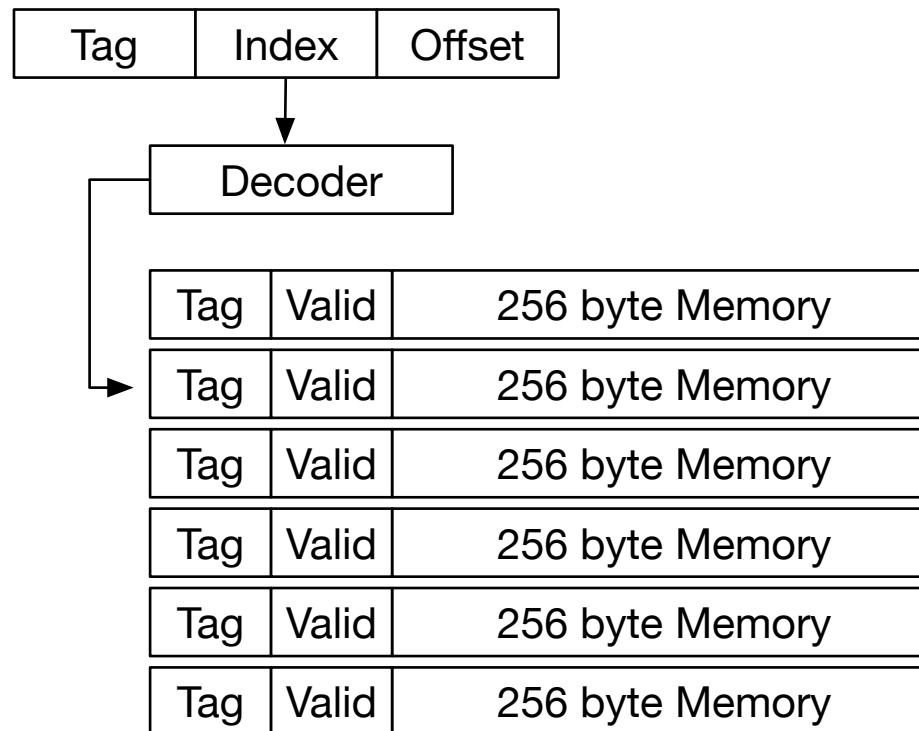
cache read

Cache Hit



- Memory request from CPU
- Data found in cache
- Send data to CPU

Cache Circuit

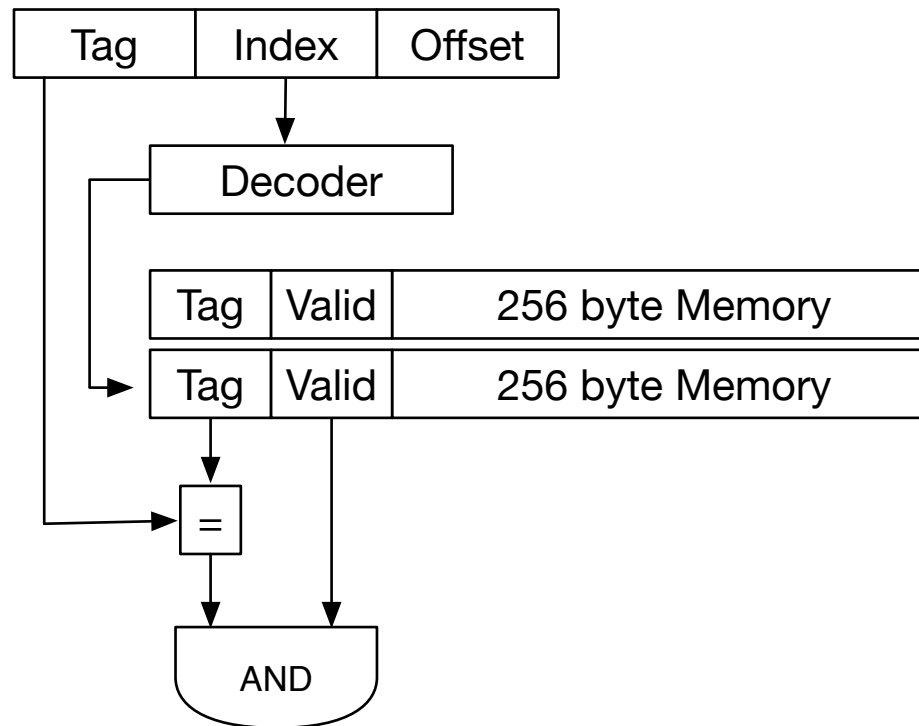


- Address split up into tag, index, and offset
- Index contains address of block in cache
- Decoded to select correct row

Cache Circuit

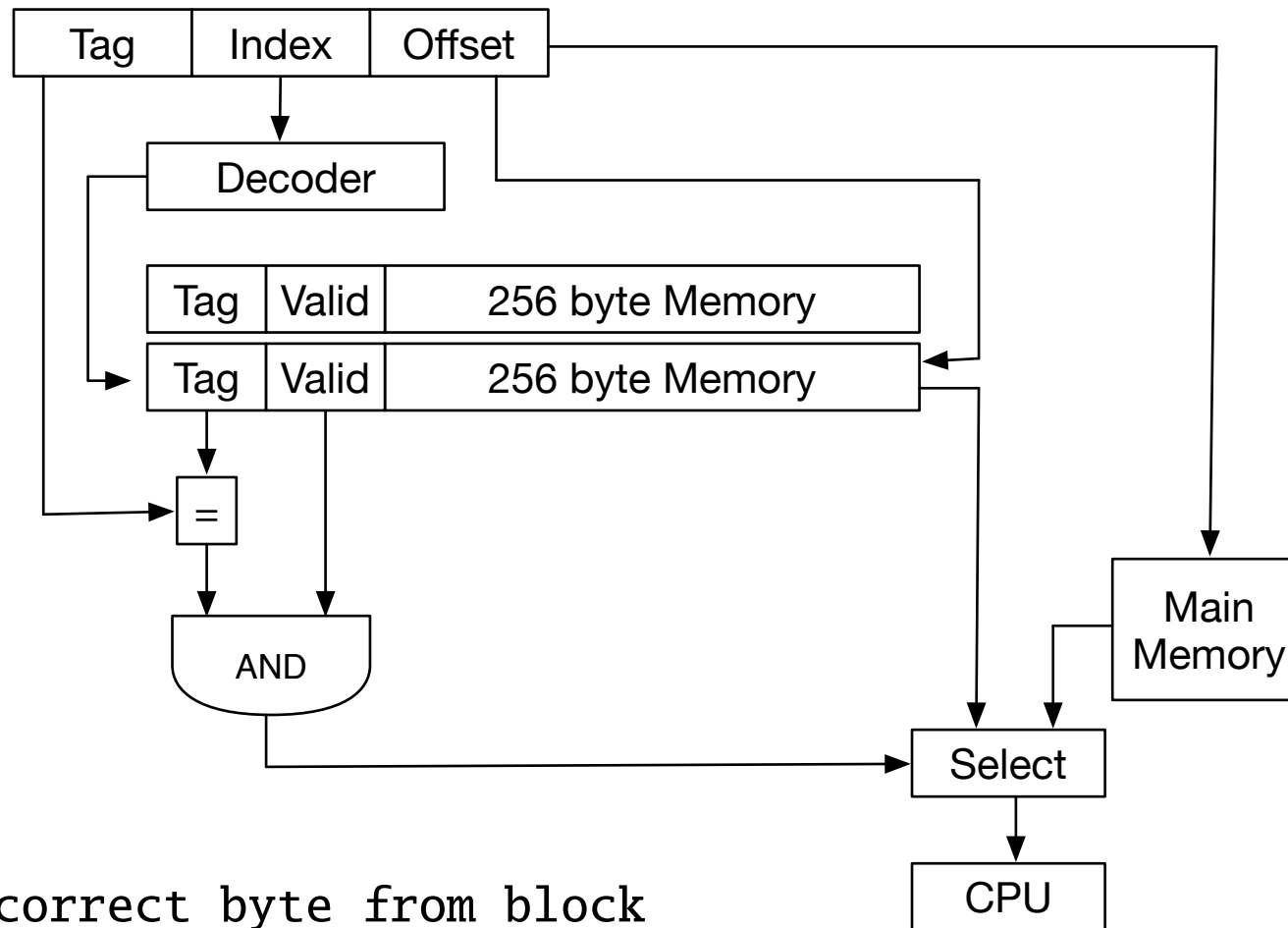


9



- Check tag for equality
- Check if valid bit is set

Cache Circuit

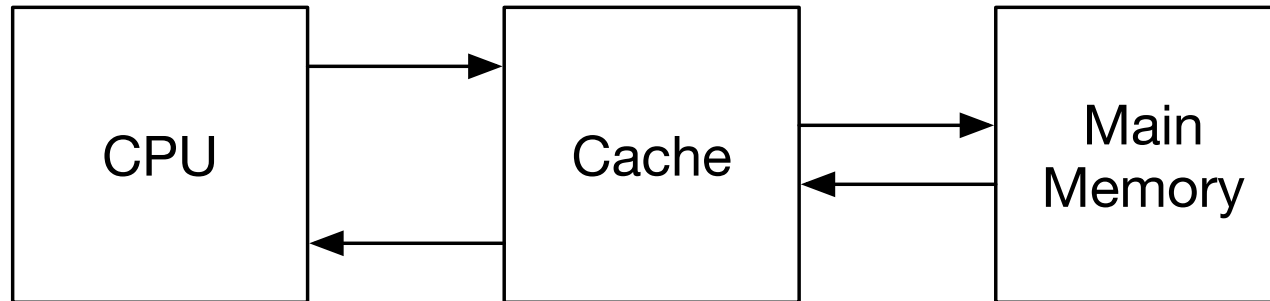


- Retrieve correct byte from block (identified by offset)
- Use cache only if valid and correct tag



cache miss

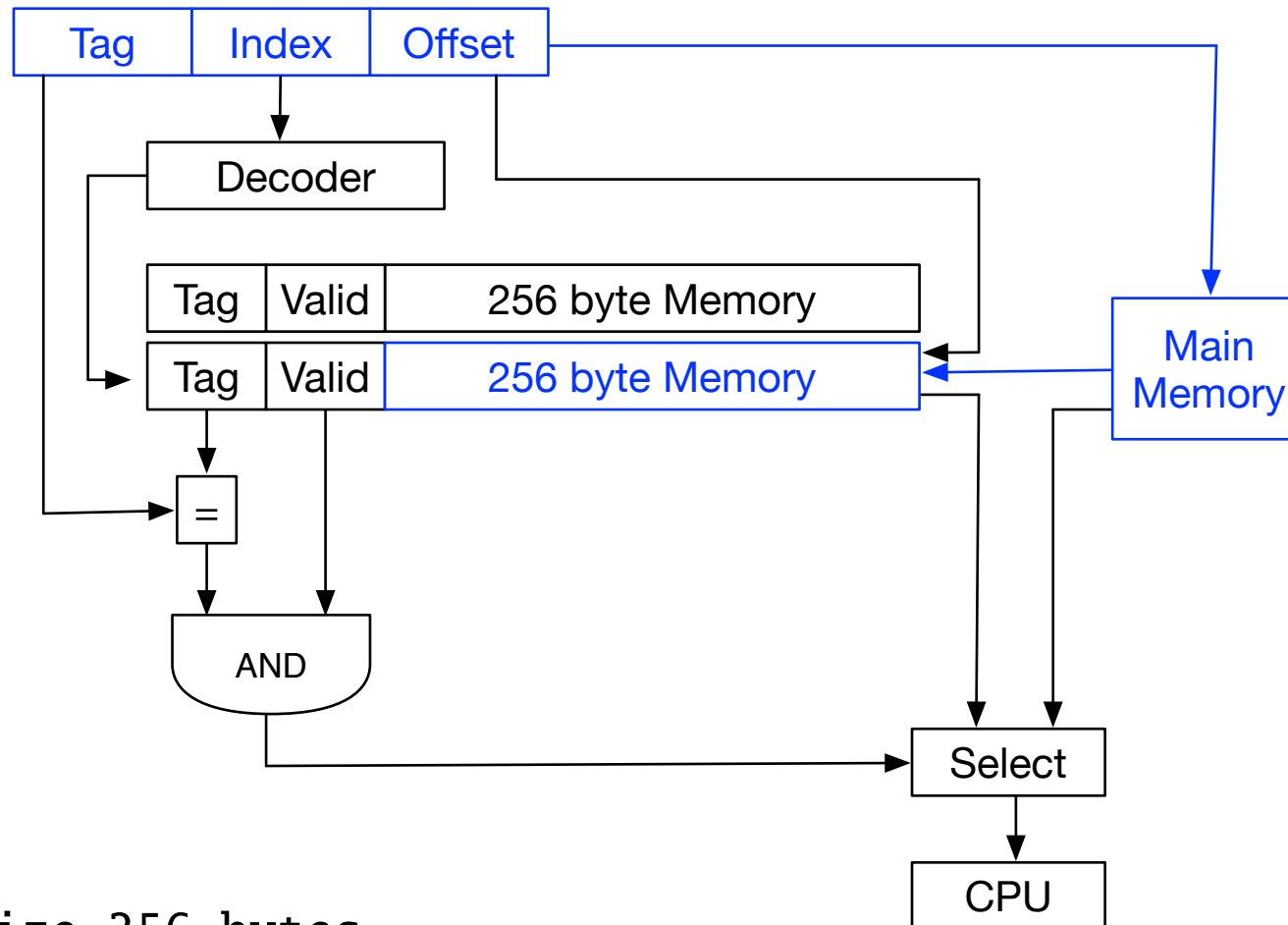
Cache Miss



- Memory request from CPU
- Data **not** found in cache
- Memory request from cache to main memory
- Send data from memory to cache
- Store data in cache
- Send data to CPU

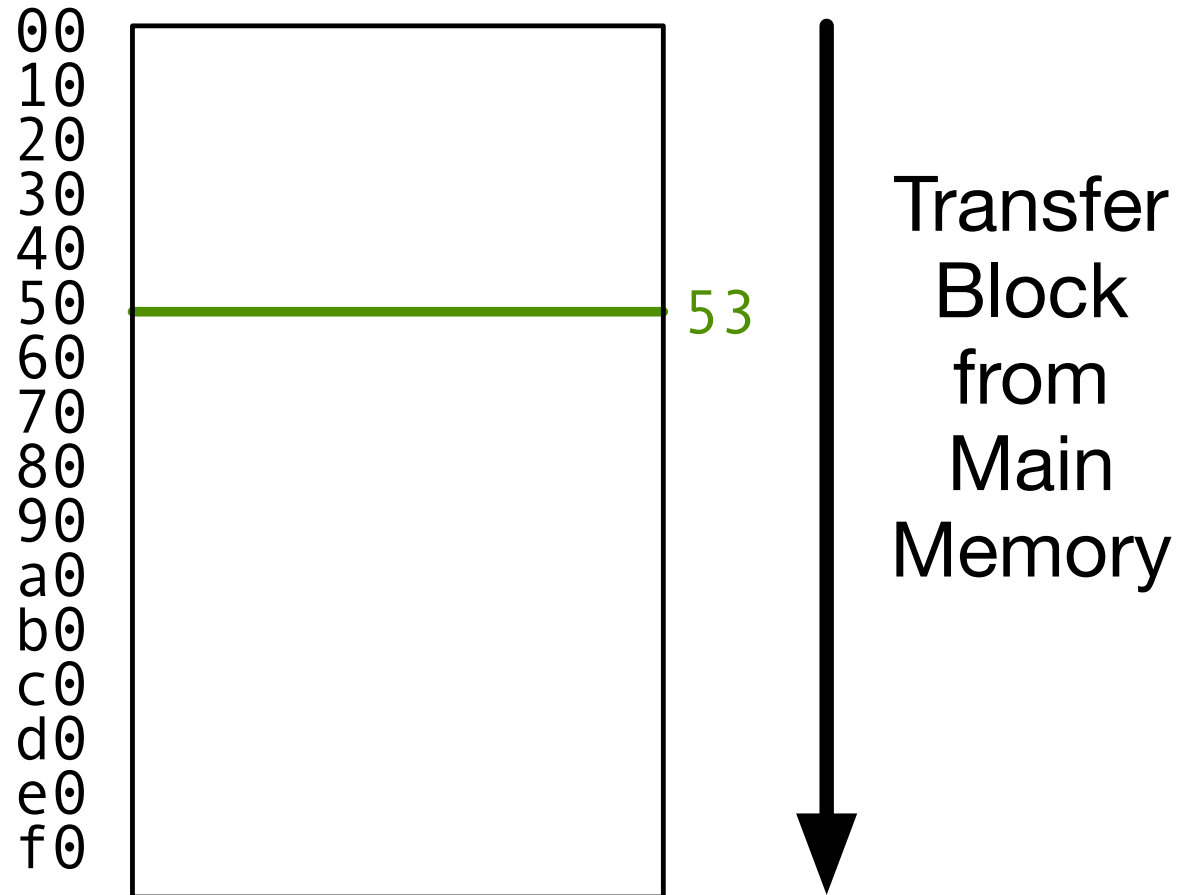
- Requires load of block from main memory
 - Blocks execution of instructions
 - Recall discussion of memory access speeds
 - CPU clock cycle: 3 GHz \rightarrow 0.33ns per instruction
 - DRAM speeds: 50ns
- \Rightarrow Significant delay (150 instruction cycles stalled)

Block Loading



- Example
 - block size 256 bytes
 - request to read memory address `$00d3ff53`
- Cache miss triggers read of block `$00d3ff00-$00d3ffff`

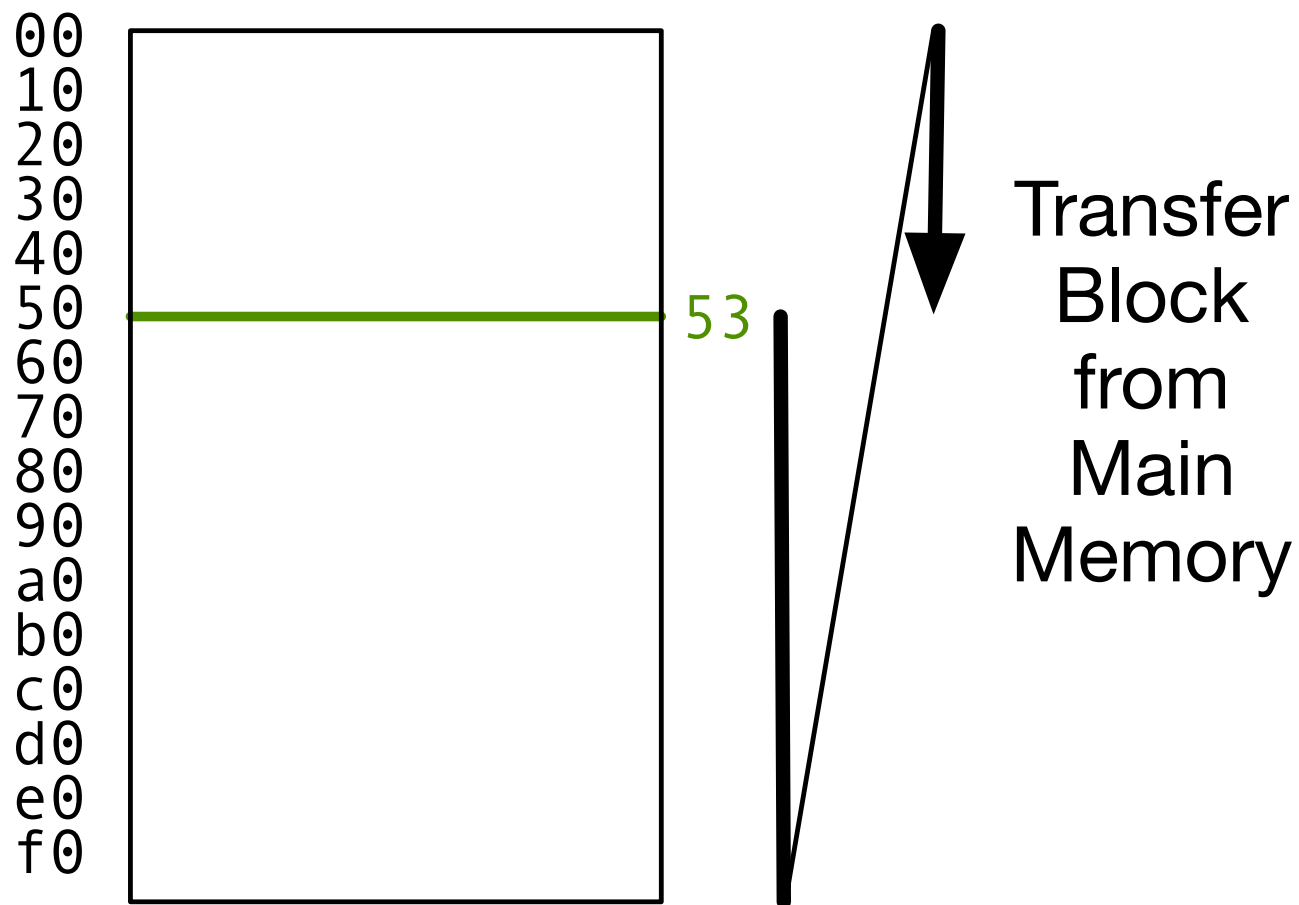
Read \$00d3ff53



- But: this requires 53 read cycles before relevant byte is loaded

Better

16

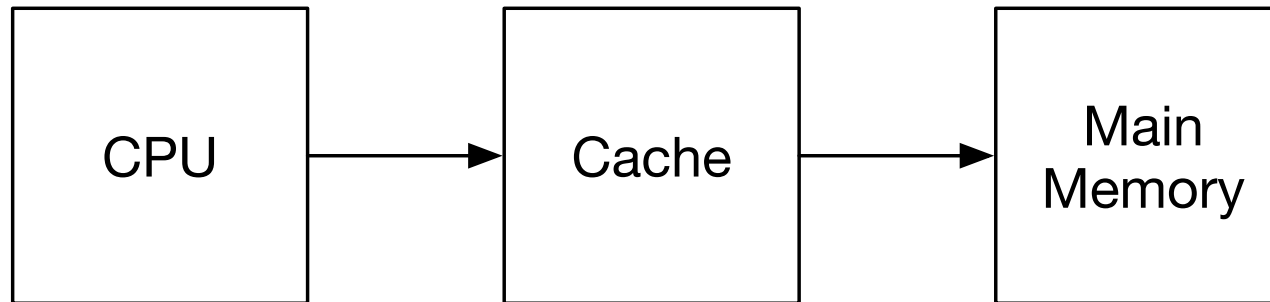


- Read requested byte first



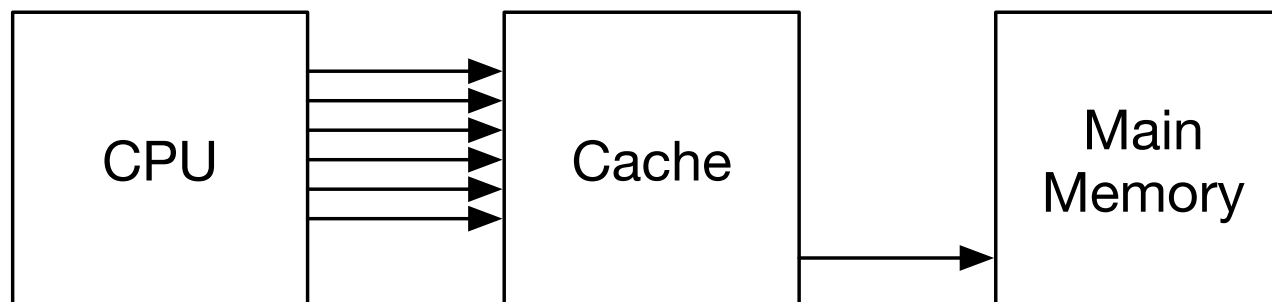
cache write

Write Through



- Writes change value in cache
- Write through: immediately store changed value in memory
- Drawback: slows down **every** write

Write Back



- Only change value in cache
- Record that cache block is changed with "dirty bit"
- Write back to RAM only when block is pre-empted

Write Buffer

- CPU does not need to wait for write to finish
- Write buffer
 - store value in write buffer
 - transfer values from write buffer to main memory in background
 - free write buffer
- This works fine, unless process overloads write buffer

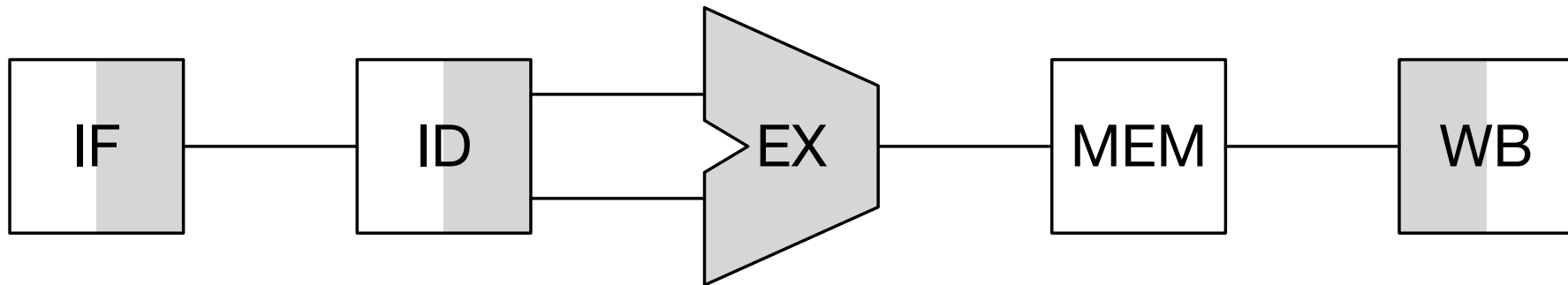
Write Miss

- Problem: CPU writes to address X, but X is not cached
- Need to load block into cache first
- Write allocate
 - allocate cache slot
 - write in value for X
 - load remaining values from main memory
 - set dirty bit



split cache

MIPS Pipeline

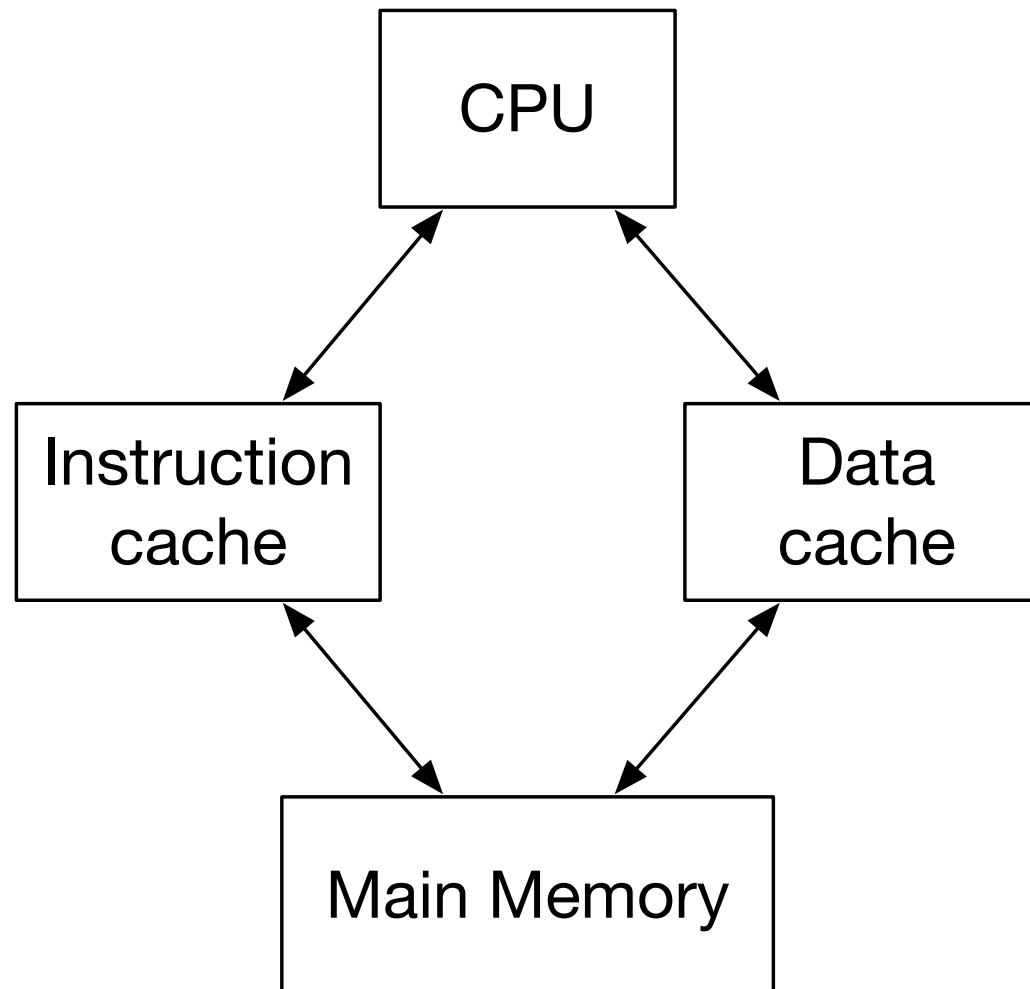


- 2 stages access memory
 - IF: instruction fetch loads current instruction
 - MEM: memory stage reads and writes data

⇒ 2 memory caches in processor

- instruction memory
- data memory

Architecture



- IF and MEM operations can be executed simultaneously
- Possible drawback: same memory block in both caches
... but very unlikely: code and data usually separated
- Cache misses possible in both caches
→ contention for memory lookup, blocking
- Instruction cache simpler: no writes