High-Performance and Embedded Computer Architecture

Course 1: Fundamentals of Computer Architecture

Module 1.5: Memory Hierarchy Review

Lesson 1.5.2: Basic Cache Optimization Techniques

Prof. Dr. Ben Juurlink

Embedded Systems Architecture

Institute of Computer Engineering and Micro-Electronics

TU Berlin

EIT ICT Labs

AES Embedded Systems Architecture

Cache Performance Metrics

- HitRate = #CacheHits / #CacheAccesses
- MissRate = #CacheMisses / #CacheAccesses = 1 – MissRate

- HitTime = time for a hit
- MissPenalty = cost of a miss

- AMAT = HitTime + MissRate x MissPenalty

Cache Miss Categories – 3 Cs Model

- Compulsory — First access to a block is always a miss.
  - Also called cold start misses
  - Misses in infinite size cache
- Capacity — Cache cannot contain all blocks needed, capacity misses occur due to blocks being discarded and later retrieved
  - Misses in Fully Associative Cache with optimal replacement
- Conflict — Misses occurring because several blocks map to same set.
  - Also called collision misses.
  - All other misses

Improving Cache Performance

- AMAT = HitTime + MissRate x MissPenalty
- Reduce HitTime
  - Small and simple cache
- Reduce MissRate
  - Larger block size
  - Higher associativity
  - Larger cache
- Reduce MissPenalty
  - Multilevel caches
  - Give priority to read misses

Larger Block Size

- Blocks of 1 word do not exploit spatial locality
- More efficient to transfer n words than n single words
- Typical block size level-1 (L1) cache: 32B, 64B
- Larger block size reduces miss rate but increases miss penalty

Larger Block Size

- Assumptions
  - HitTime = 1 cc
  - MissPenalty = 10 + (WordsPerBlock) cc
  - MissRate:

Block size	4B	8B	16B	32B	64B	128B
Miss Rate	10%	6%	4%	3%	2.5%	2.3%

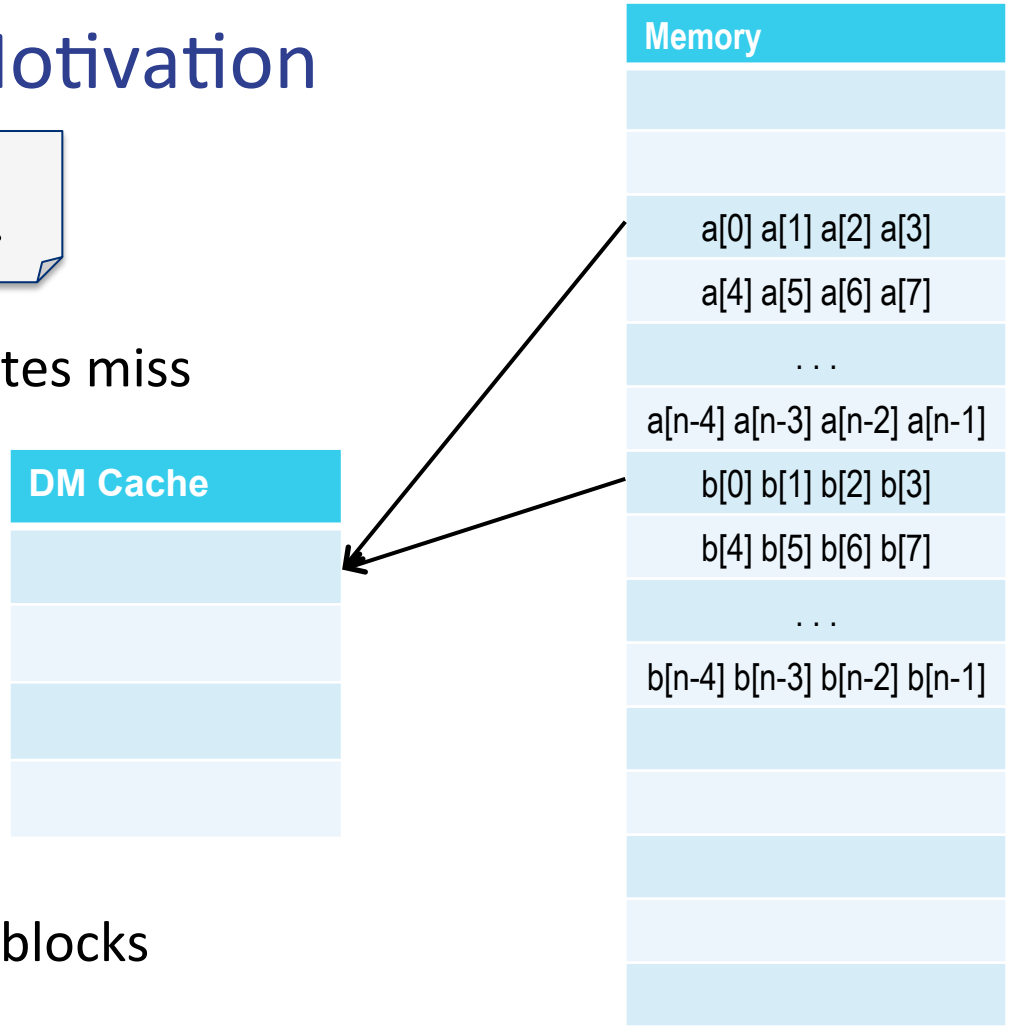
- Which block size has smallest AMAT?

Block size	4B	8B	16B	32B	64B	128B
Miss penalty	11	12	14	18	26	42
AMAT	2.1	1.72	1.56	**1.54**	1.65	1.97

Associative Caches - Motivation

```
for (i=0; i<n; i++)
   dotprod += a[i]*b[i];
```

- Every access to a and b generates miss



- Solution: flexible placement of blocks
  - ➢ associative caches

Possible Organizations of Cache w/ 4 Blocks

1-way set associative (= direct mapped)		
Set	Tag	Data
0		
1		
2		
3		

2-way set associative				
Set	Tag	Data	Tag	Data
0				
1				

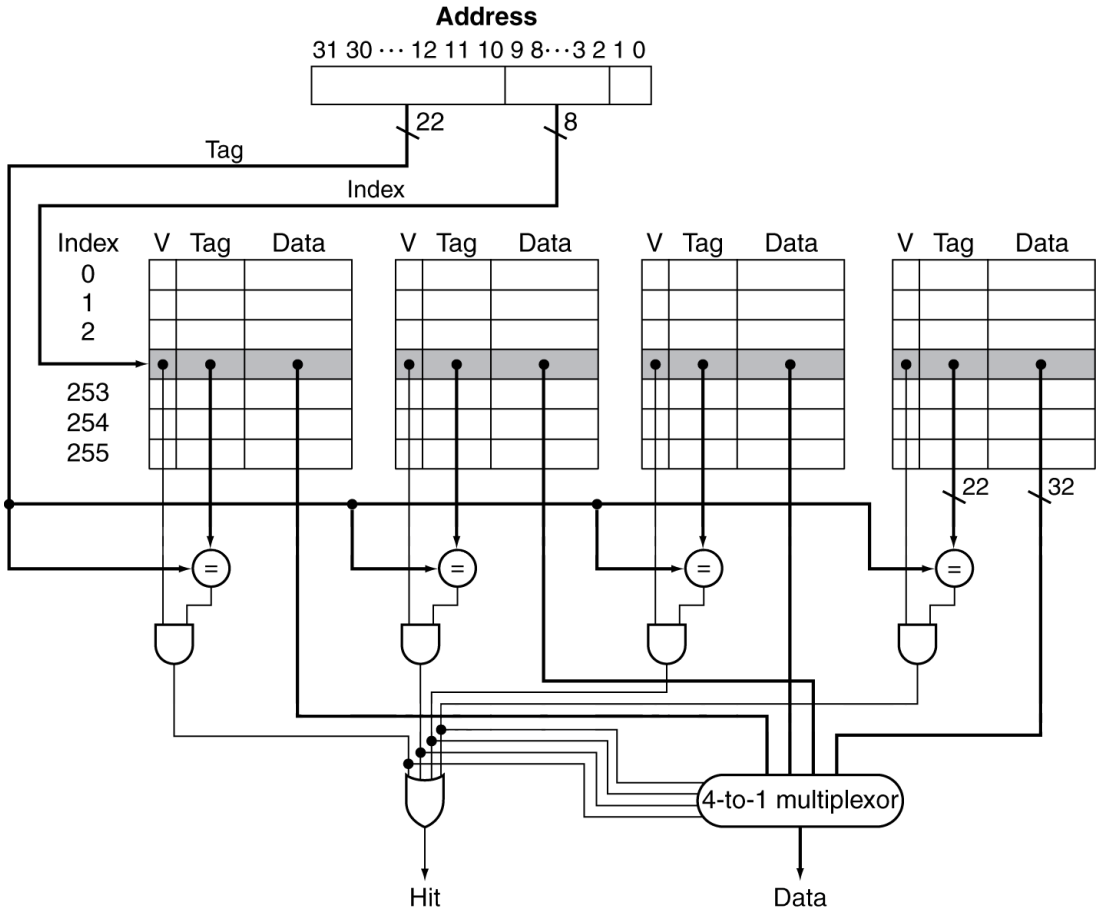
4-way set associative (fully associative)				
Set	Tag	Data	Tag	Data
0				

Cache Equations for Set-Associative Cache

- BlockAddress = ByteAddress / BytesPerBlock
- CacheIndex = BlockAddress % #Sets
  - DM cache → #Sets = #Blocks
  - Fully associative → #Sets = 1 → CacheIndex == 0
- Tag = BlockAddress / #Sets
- CacheSize = #Sets x #Ways x BytesPerBlock

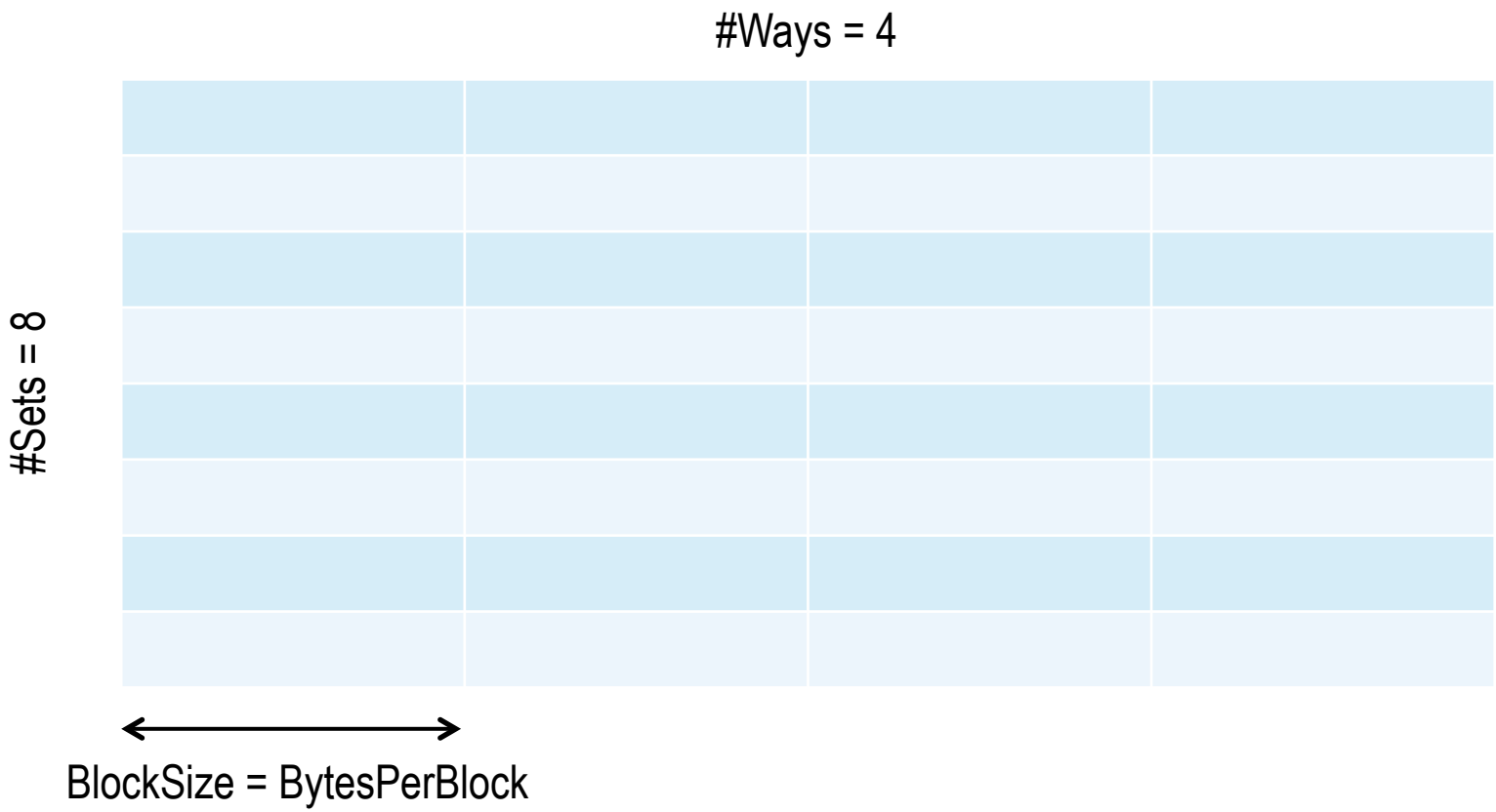
2-way set associative				
Set	Tag	Data	Tag	Data
0				
1				

4-Way Set-Associative Cache



- What is the block size?
- What is the size of this cache?

Cache Size = Cache Capacity



CacheSize = #Sets x #Ways x BytesPerBlock

Intel Core i7 Instruction Cache (I$)

- Size = 32 KB
- Block size = 64 bytes
- Associativity = 4-way
- Address length = 48-bit

- How long is the index (in bits)?
- How long is the tag (in bits)?

- CacheSize = #Sets x #Ways x BytesPerBlock
- 32 KB = #Sets x 4 x 64
- #Sets = 32 x 1024 / (4 x 64) = 128 = 2^7

➢ Index length = 7 bits

- TagLength = AddressLength – IndexLength – BlockOffsetLength
- $2^{\text{BlockOffsetLength}}$ = BlockSize = 64
- BlockOffsetLength = 6

➢ TagLength = 48 – 7 – 6 = 35 bits

Replacement Algorithm

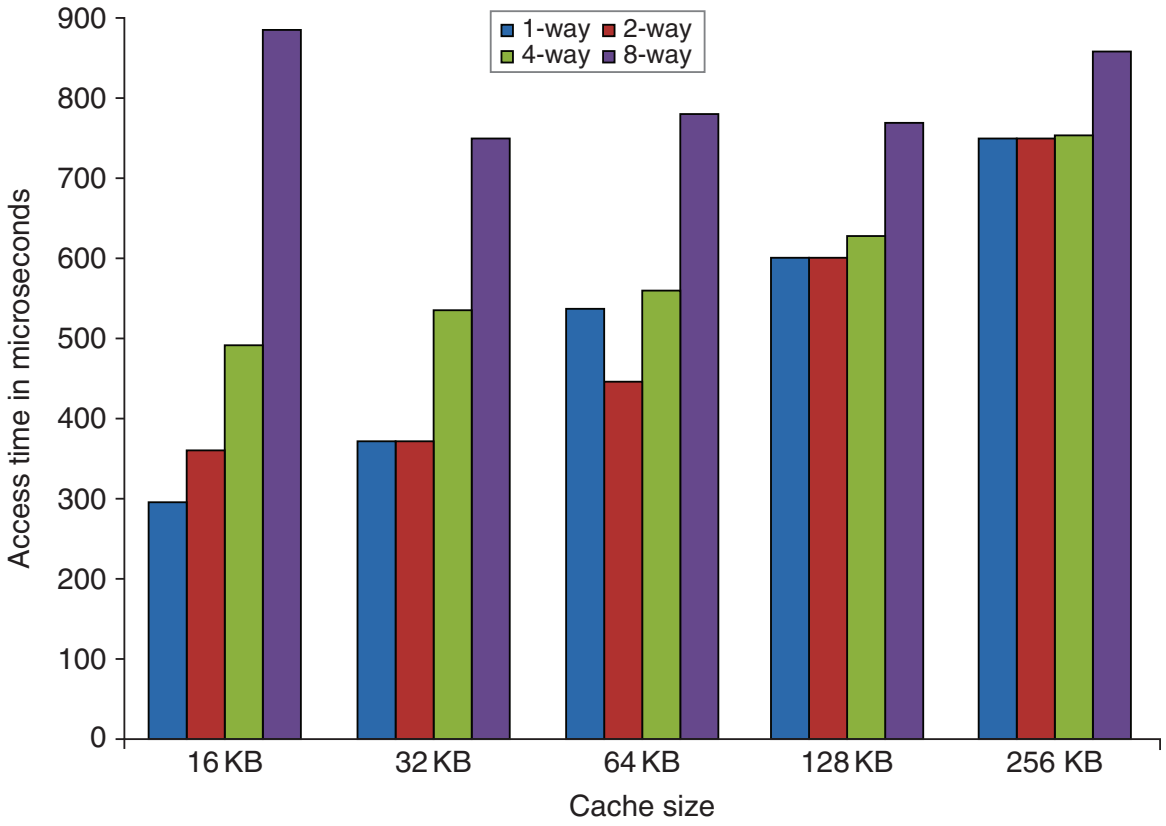
- Which block will be replaced if all blocks in set are occupied?
- No choice for direct-mapped cache

- Random
  - ➢replace random block within set
- First-In-First-Out (FIFO)
  - ➢replace block that entered cache first
- Least Recently Used (LRU)
  - ➢replace block that has not been accessed for the longest time
- Optimal algorithm (OPT or MIN)
  - ➢replace block that will not be used for the longest time

Improving Cache Performance

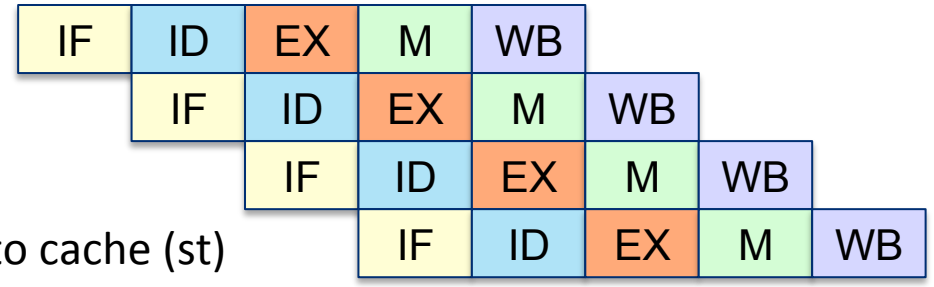
- AMAT = HitTime + MissRate x MissPenalty
- Reduce HitTime
  - Small and simple cache
- Reduce MissRate
  - Larger block size
  - Higher associativity
  - Larger cache
- Reduce MissPenalty
  - Multilevel caches
  - Give priority to read misses

Small and Simple Caches

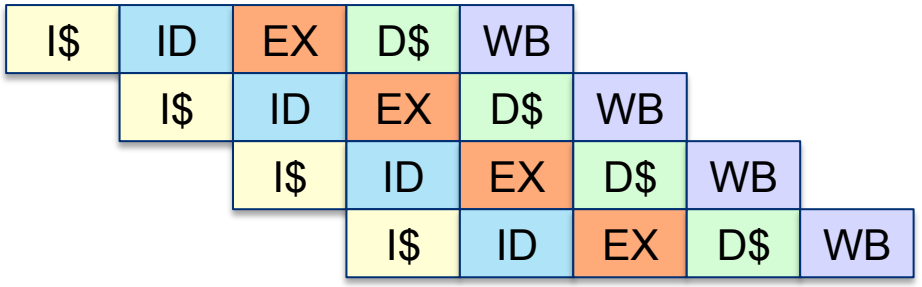


- FIXME
- Microseconds???

Split Cache

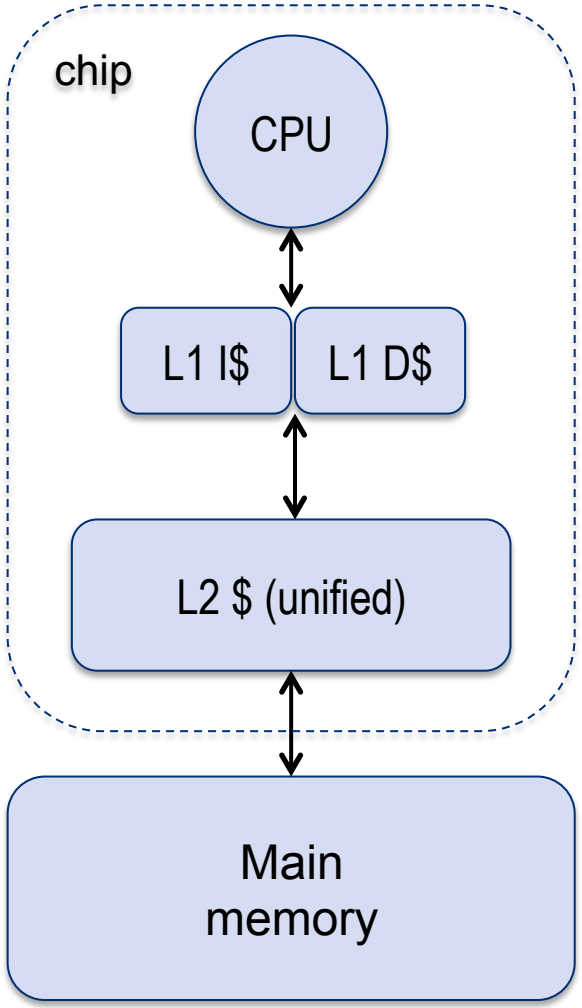


- In 1 cycle of canonical pipeline
  - fetch instruction
  - read data from cache (ld) or store data to cache (st)
- If only 1 (unified) cache, cache must have
  - 2 read ports
  - 1 write port
- Multiported cache are slower, larger, and more power consuming
- Better to use split cache
  - 1 cache for instructions (I-cache or I$)
  - 1 cache for data (D-cache or D$)



Multilevel Cache

- Reduce miss penalty by employing multiple cache levels
- L1 cache small and fast
  - ➢ Hit time 1-3 cc
- L2 cache larger and slower, but still much faster than main memory
  - ➢ L2 hit time: 10-20 cc
  - ➢ Main memory latency: 100-500 cc
- High-end processors even have L3 cache



Performance of Multilevel Cache

FIXME

- Assumptions
  - Base CPI = 1.0 (all accesses hit L1 cache)
  - Frequency = 4 GHz (cycle time = 0.25ns)
  - Main memory latency = 100ns (400 cc)
  - MissRate_{L1} = 2% (Both instructions and data)
- Only L1 cache
  - CPI = Base CPI + MissRate_{L1} x MissPenalty = 1.0 + 0.02x400 = 9.0
- Add L2 cache
  - hit time = 5ns = 20 cc
  - MissRate_{L2} = 25% (#L2misses / #L2 accesses)

Performance of Multilevel Cache

FIXME

- Only L1 cache
  - CPI = Base CPI + MissRate_{L1} x MissPenalty_{L1} = 1.0 + 0.02x400 = 9.0
- Add L2 cache
  - hit time = 5ns = 20 cc
  - MissRate_{L2} = 25% (#L2misses / #L2 accesses)
- CPI = Base CPI + MissRate_{L1} x MissPenalty_{L1}
- MissPenalty_{L1} = HitTime_{L2} + MissRate_{L2} x MissPenalty_{L2}
- CPI = 1.0 + 0.02 x (20 + 0.25 x 400) = 3.4

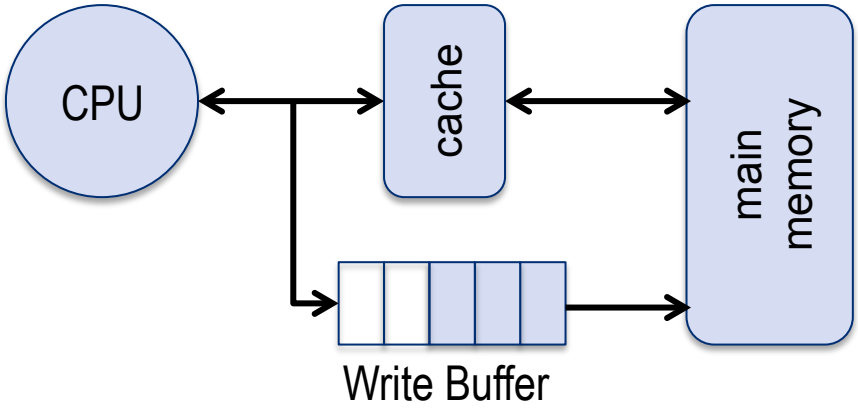
- Processor w/ L2 cache is 9.0/2.4 = 2.6x faster

What Happens on a Write Hit?

- Do we write new data only to cache or also to memory?
- Write-through: both to cache and to memory
  - Advantage: Cache and memory consistent
  - Disadvantage: Many writes to memory
- Write-back: only to cache; to memory when block is replaced
  - Advantage: fewer writes to memory; several writes to same cache block before it is replaced
  - Disadvantage: Cache and memory inconsistent

Write-through Cache – Write Buffer

- Write buffer stores data until they are written to memory
  - ➢CPU continues and only stalls when write buffer full

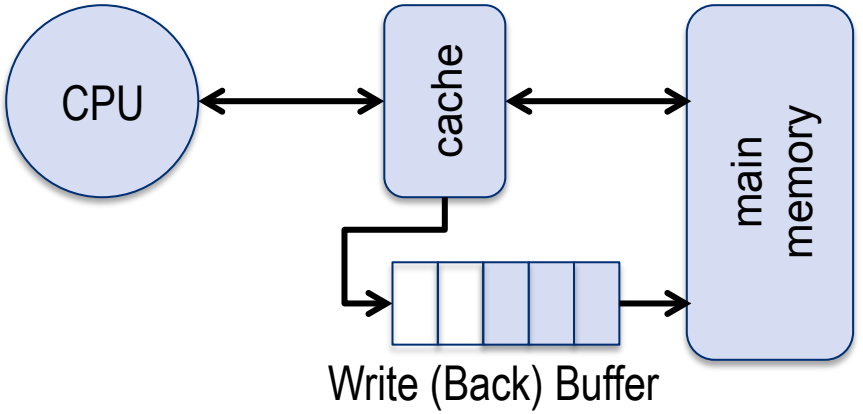


Write-back Cache – Dirty Bit & Write Buffer

- Only need to write back cache block if it has been written
  - Indicated by dirty bit

D	V	Tag	Data

- Write-back caches have Write Buffer for replaced dirty blocks
  - Also called Write-Back Buffer



Write-through Versus Write Back

- Write through cache can be written during tag comparison
- Write back cache not

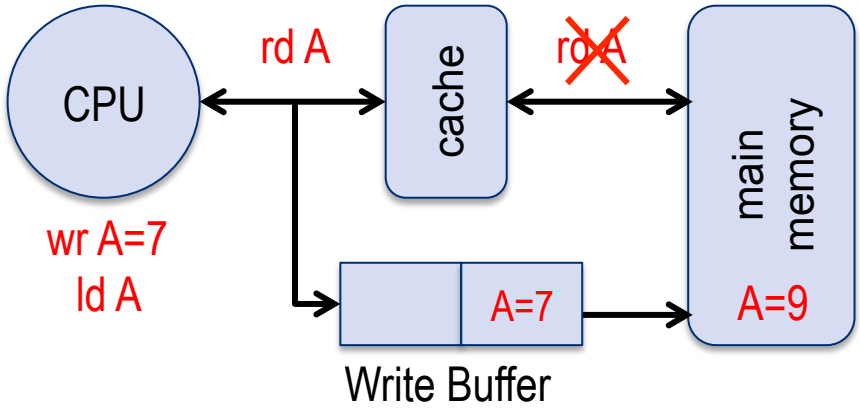
What Happens on a Write Miss?

- Write allocate (or fetch on write): block is loaded on write miss, followed by same actions as for write hit
- No-write allocate (or write around): block is modified in memory and not loaded into cache

- Write policy ∈ {write through, write back} x {write allocate, no-write-allocate}
  - ➢ write back caches generally use write allocate
  - ➢ write through often use no-write allocate

Give Read Misses Priority Over Writes

- Read misses must be handled asap
  - Execution units stall waiting for data
- Writes can happen "in the background"
- Must maintain memory order
  - Load should return value written by most recent store to same address
  - On read miss, must drain or check write buffer first



Thank You For Your Attention

Prof. Dr. Ben Juurlink

Embedded Systems Architecture

Institute for Computer Engineering and Micro-Electronics

School of Electrical Engineering and Computer Science

TU Berlin

Back-up Slides

Processor-memory performance gap