CS425
Computer Systems Architecture

Fall 2019

Caches: Improving Performance

Classification of Cache Optimizations

- Goal: Reduce the Average Memory Access Time
  - AMAT = Hit Time + Miss Rate * Miss Penalty
- Approaches
  - Reduce Miss Rate
  - Reduce or Hide Miss Penalty
  - Reduce Hit Time
- Caveats
  - These may be conflicting goals
  - Keep track of clock cycle time, area, and power consumption

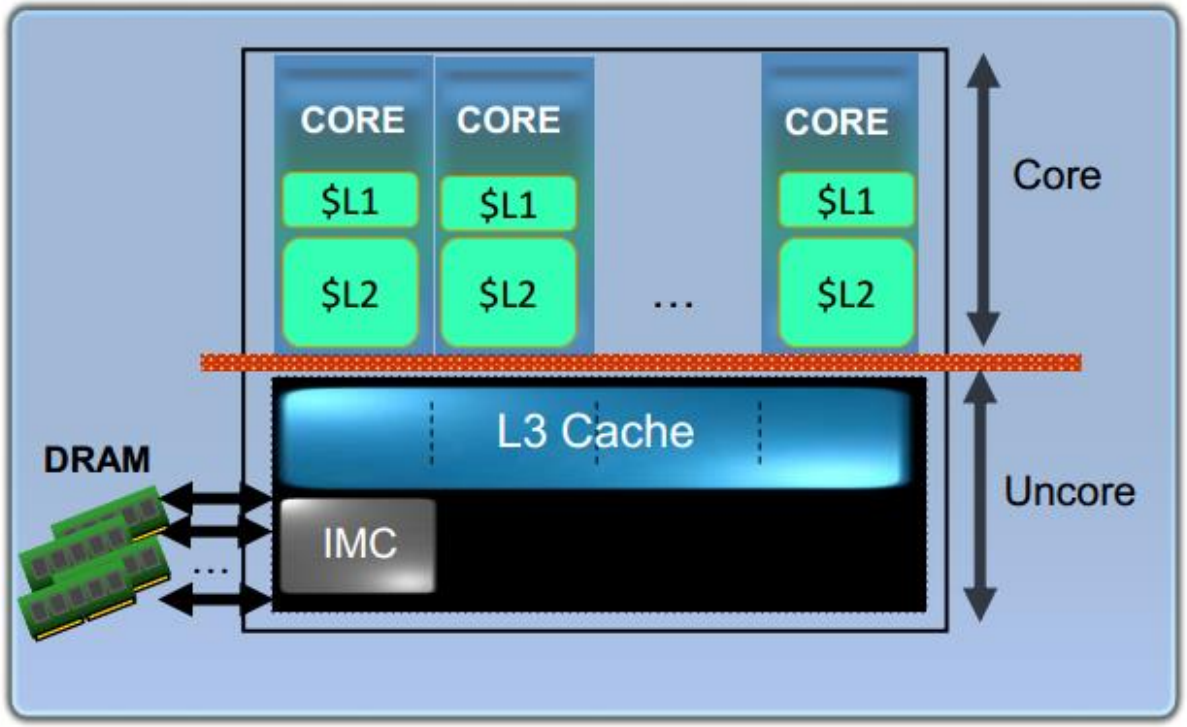
Common Advanced Cache Optimizations

- Multi-level caches and inclusion
- Victim caches
- Pseudo-associative caches
- Skew-associative caches
- Critical word first
- Non-blocking caches
- Prefetching
- Multi-ported caches

Classification of Cache Optimizations

- Reduce (or Hide) Miss Penalty
- Reduce Miss Rate
- Reduce Hit Time

Multi-core Processors



- Multi-level caches
  - Private L1 instruction and data caches and (optionally) L2 caches
  - Shared last level cache (L3 cache in this diagram)
  - Connected through wide links

1. Multi-level Caches

- Motivation
  - Optimize each cache for different constraints
  - Exploit cost/capacity trade-offs at different levels
- Private L1 caches
  - Optimized for fast access time by one core
  - 8KB-64KB, direct-mapped to 4-way associative, 1-3 CPU cycles
- Private L2 caches
  - Extend the capacity of L1, shield from latency of shared LLC cache
  - 256KB-512KB, 4 to 8-way associate
- L3 caches
  - Shared for best utilization (how?)
  - Optimized for low miss-rate: Multi-MB, highly associative (why?)

Multi-level Caches

Motivation

- Bigger caches bridge gap between CPU and DRAM
- Smaller caches keep pace with CPU speed
- Mutli-level caches a compromise between the two
  - L2 cache captures misses from L1 cache
  - L2 cache provides additional on-chip caching space

Performance analysis

$$AMAT = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times \text{Miss penalty}_{L1}$$

$$\text{miss penalty}_{L1} = \text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2}$$

$$AMAT = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

Multi-level cache miss rates

Local vs. global miss rate

- Local miss rate: Number of misses in the cache divided by the number of access to this cache
- Global miss rate: Number of misses in the cache divide by the number of accesses issued from the CPU

Performance analysis

$$\begin{aligned}\text{Average memory stalls per instruction} = {} & \text{Miss per instruction}_{L1} \times \text{Hit time}_{L2} \\ & + \text{Misses per instruction}_{L2} \times \text{Miss penalty}_{L2}\end{aligned}$$

AMAT in multi-level caches

Example

- Write-back first-level cache
- 40 L1 misses per 1000 memory references
- 20 L2 misses per 1000 memory references
- L2 cache miss penalty = 100 cycles
- L1 hit time = 1 cycle
- L2 cache hit time = 10 cycles
- 1.5 memory references per instructions

$$\text{Miss rate}_{L1} = \frac{40}{1000} = 0.04$$

$$\text{Miss rate}_{L2,local} = \frac{\text{misses in L2}}{\text{misses in L1}} = \frac{20}{40} = 0.50$$

$$AMAT = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

$$AMAT = 1.0 + 0.04 \times (10 + 0.50 \times 100) = 3.4\text{cycles}$$

Stalls in multi-level caches

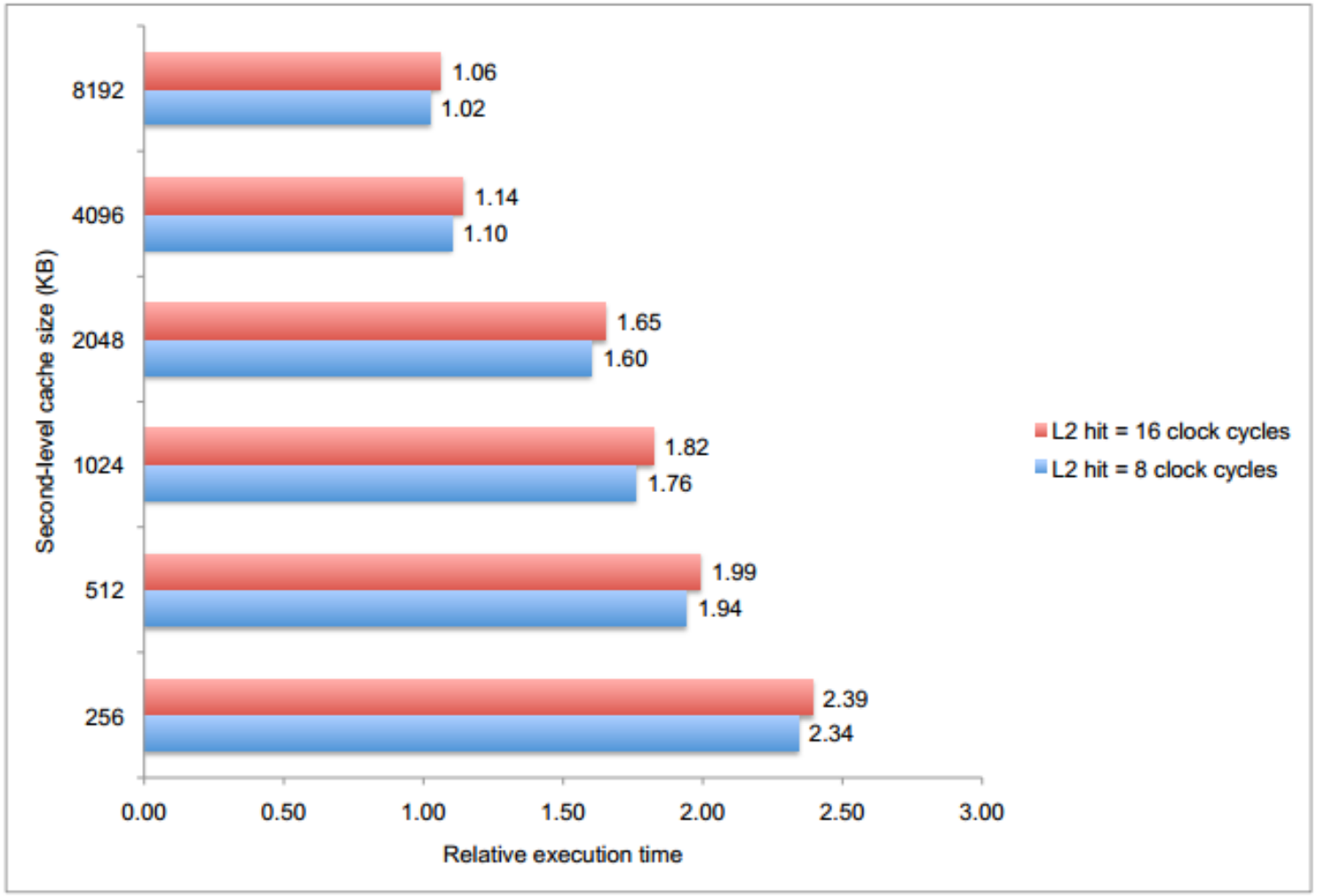
Example

- Write-back first-level cache
- 40 L1 misses per 1000 memory references
- 20 L2 misses per 1000 memory references
- L2 cache miss penalty = 100 cycles
- L1 hit time = 1 cycle
- L2 cache hit time = 10 cycles
- 1.5 memory references per instructions

$$\text{Average memory stalls per instruction} = \text{Misses per instruction}_{L1} \times \text{Hit time}_{L2} + \text{Misses per instruction}_{L2} \times \text{Miss penalty}_{L2}$$

$$\frac{40}{1000} \times 1.5 \times 10 + \frac{20}{1000} \times 1.5 \times 100 = 3.6 \text{ clock cycles}$$

L2 cache performance implications



Normalized to 8K KB, 1 clock cycle hit L2 cache

Inclusion Property

L1 cache contents always in L2 cache?

- Mutil-level inclusion guarantees that L1 data is always present in L2, L2 in L3, . . .
  - Simplifies cache consistency check in multiprocessors, or between I/O devices and processor
  - Complicates the use of different block sizes for L1 and L2 – L1 refill may require storing more than one blocks

    Second-level cache miss must invalidate all first-level blocks
- Mutil-level exclusion guarantees that L1 data is never present in L2, L2 data never present in in L3, . . .
  - Reasonable choice for systems with small L2 cache relative to the L1 cache
  - Effective expansion of total caching space with a slower cache

AMD Athlon supports exclusive caches

Pentium 4 has no constraints (non-inclusive/non-exclusive)

2. Critical word first and early restart

Critical word first

- Cache block size tends to increase to exploit spatial locality
- Any given reference needs only one word from a multi-word block
- CWF fetches requested word first and sends it to processor
- Processor continues execution while rest of the block is fetched

Early restart

- Fetches words in the order stored in the block
- As soon as critical word arrives, sends to processor and processor restarts

3. Giving priority to read misses over writes

Write-through caches

- Write buffer holds written data to mask memory latency
- Write buffer may hold values needed by a later read miss

```
SW R3, 512(R0)      ;M[512] = R3   (cache index 0)

LW R1, 1024(R0)     ;R1 = M[1024]  (cache index 0)
LW R2, 512(R0)      ;R2 = M[512]   (cache index 0)
```

miss in cache / hit in WB

- Store to 512[R0] with block from cache index 0 waits in write buffer
- Load to 1024[R0] misses and brings new block in cache index 0
- Second load attempts to bring block from 512[R0] (held in write buffer)
- Memory RAW hazard

Giving priority to read misses over writes

Write-through caches

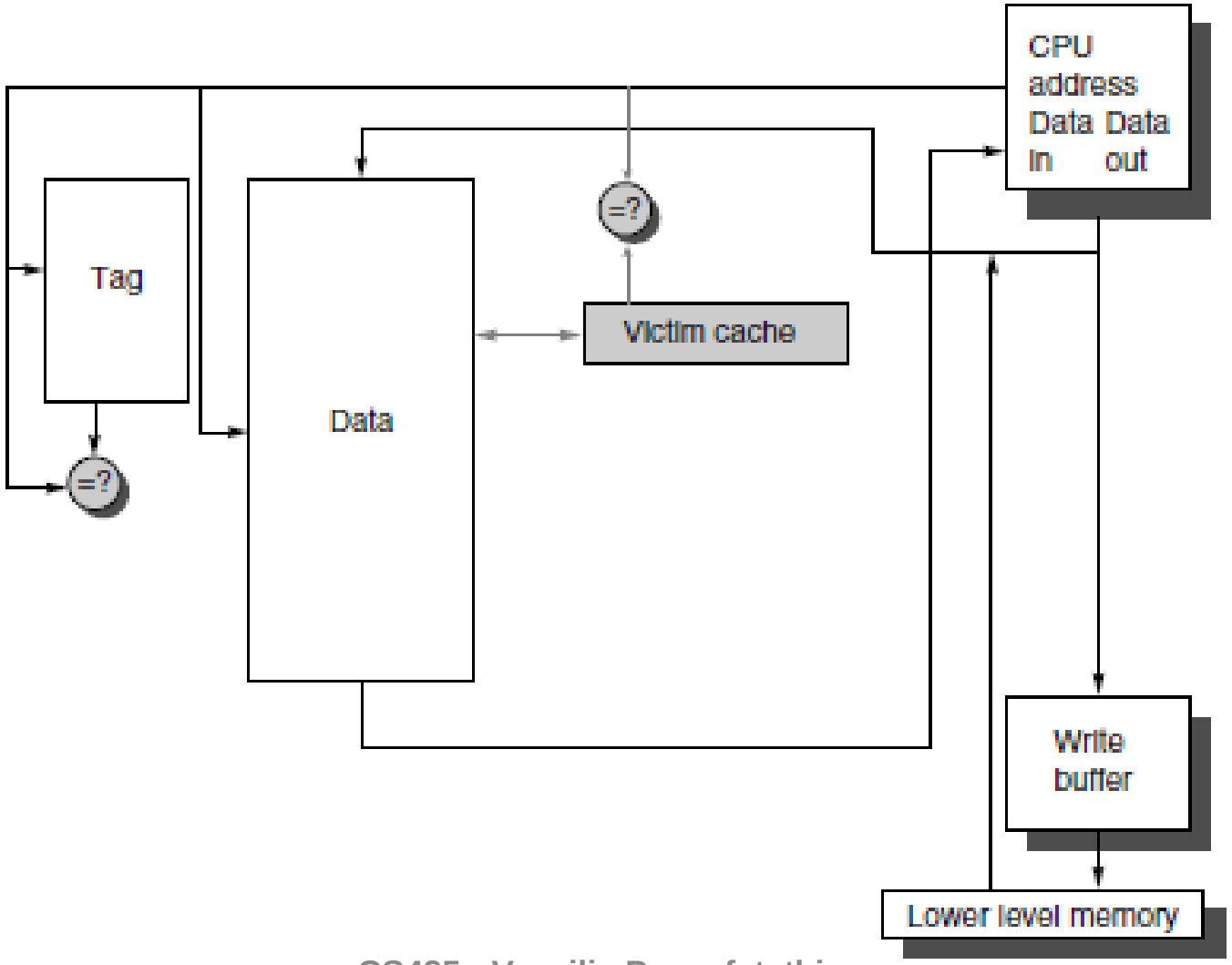
- Check contents of write buffer on read miss
- If no conflict then let missing read bypass pending write

  all desktop and server processors give reads priority over writes.

Write-back caches

- Slow path: write dirty block to memory, then fetch new block from memory to cache
- Faster path:write dirty block to write buffer, then fetch new block from memory to cache, then write back dirty block **(a.k.a. write-back buffer)**

4. Write Buffer & Victim Cache



Merging Write Buffer

Write buffer organization

- Processor blocks on write if write buffer full
- Processor checks write address with address in write buffer
- Processor merges writes to same address if address is present in write buffer
- Assume write buffer with 4 entries, with 4 64-bit words each
- Writes to same cache block in different cycles, no write merging

Write buffer, no write merging

Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

Merging Write Buffer

Write buffer organization

- Processor blocks on write if write buffer full
- Processor checks write address with address in write buffer
- Processor merges writes to same address if address is present in write buffer
- Assume write buffer with 4 entries, with 4 64-bit words each
- Writes to same cache block in different cycles, write merging

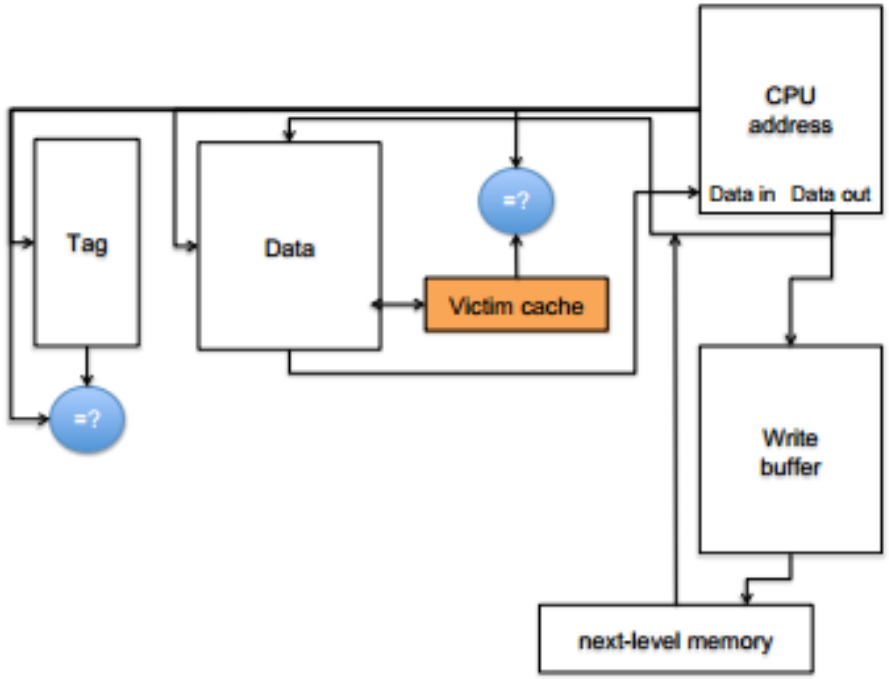
Write buffer,write merging

Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

Victim Cache

Tiny cache holds evicted cache blocks

- Small (e.g. 4-entry) fully associative buffer for evicted blocks
- Proposed to reduce impact of conflicts on direct-mapped caches
  - Victim cache + Direct-mapped cache ≈ associative cache



a four-entry victim cache might remove one quarter of the misses in a 4-KB direct-mapped data cache.

5. Non-blocking or Lookup Free Caches

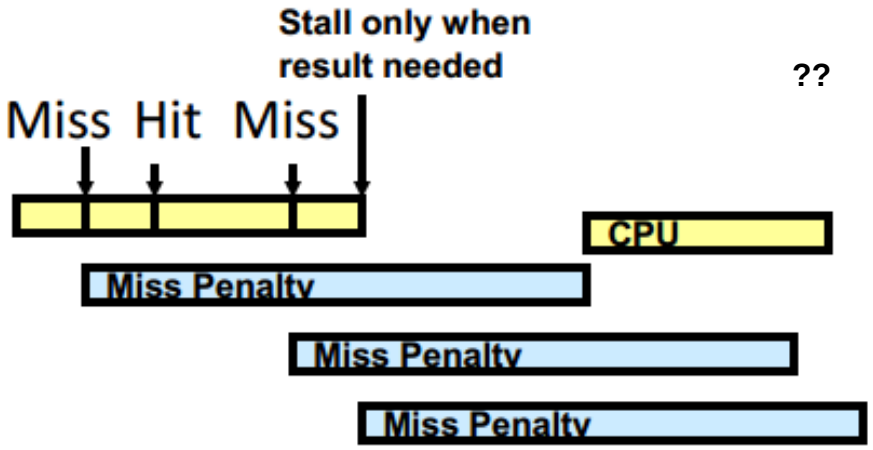
- Basic idea
  - Allow for hits while serving a miss (hit-under-miss)
  - Allow for more than one outstanding miss (miss-under-miss)
- When does it make sense (for L1, L2, …)
  - When the processor can handle >1 pending load/store
    - This is the case with superscalar processors
  - When the cache serves >1 processor or other cache
  - When the lower level allows for multiple pending accesses
    - More on this later
- What is difficult about non-blocking caches
  - Handling multiple misses at the same time
  - Handling loads to pending misses
  - Handling stores to pending misses

Out-of-order pipelines already have this functionality built in… (load queues, etc).

Potential of Non-blocking Caches

Stall CPU on miss

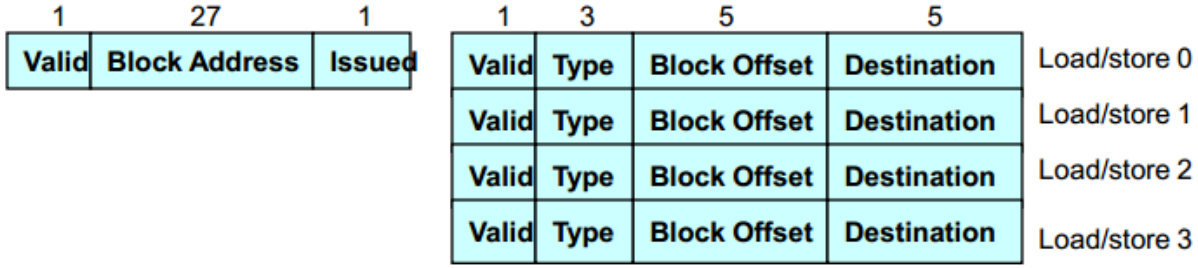
Hit under miss



Multiple out-standing misses

Miss Status Handling Registers

- Keeps track of
  - Outstanding cache misses
  - Pending load & stores that refer to that cache block
- Fields of an MSHR
  - Valid bit
  - Cache block address
    - Must support associative search
  - Issued bit (1 if already request issued to memory)
  - For each pending load or store
    - Valid bit
    - Type (load/store) and format (byte/halfword/…)
    - Block offset
    - Destination register for load OR store buffer entry for stores



Non-blocking Caches: Operation

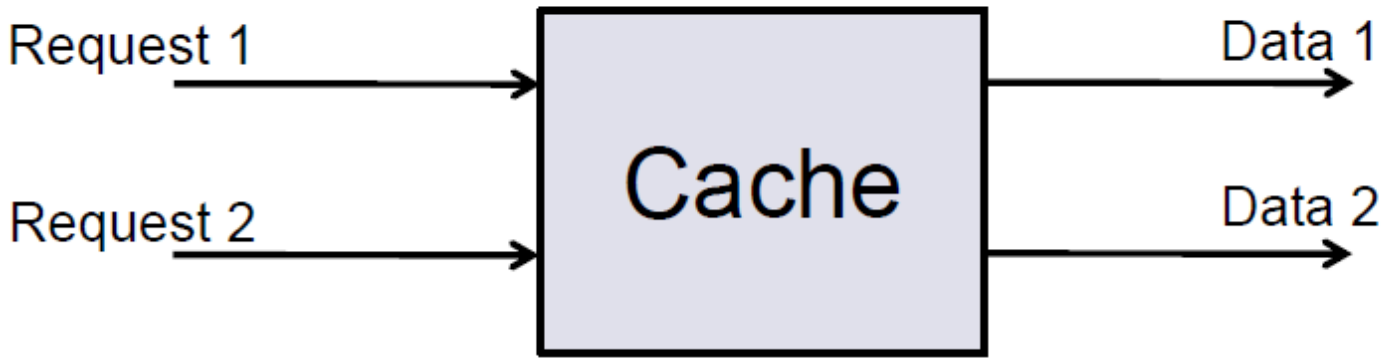
- On a cache miss:
  - Search MSHRs for pending access to same cache block
    - If yes, just allocate new load/store entry
  - (if no) Allocate free MSHR
    - Update block address and first load/store entry
  - If no MSHR or load/store entry free, stall
- When one word/sub-block for cache line become available
  - Check which load/stores are waiting for it
    - Forward data to LSU
    - Mark loads/store as invalid
  - Write word in the cache
- When last word for cache line is available
  - Mark MSHR as invalid

6. Multi-ported Caches

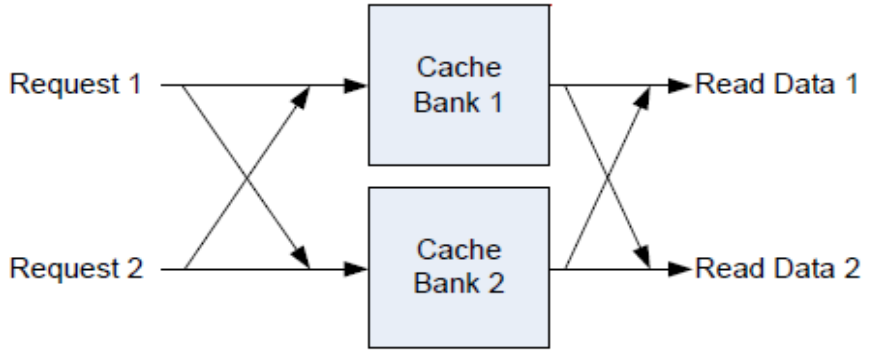
- Idea: allow for multiple accesses in parallel
  - Processor with many LSUs, I+D access in L2, …
- Can be implemented in multiple ways
  - True multi-porting
  - Multiple banks
- What is difficult about multi-porting
  - Interaction between parallel accesses (especially for stores)

True Multi-porting

- True multiporting
  - Use 2-ported tag/data storage
  - Problem: large area increase
  - Problem: hit time increase

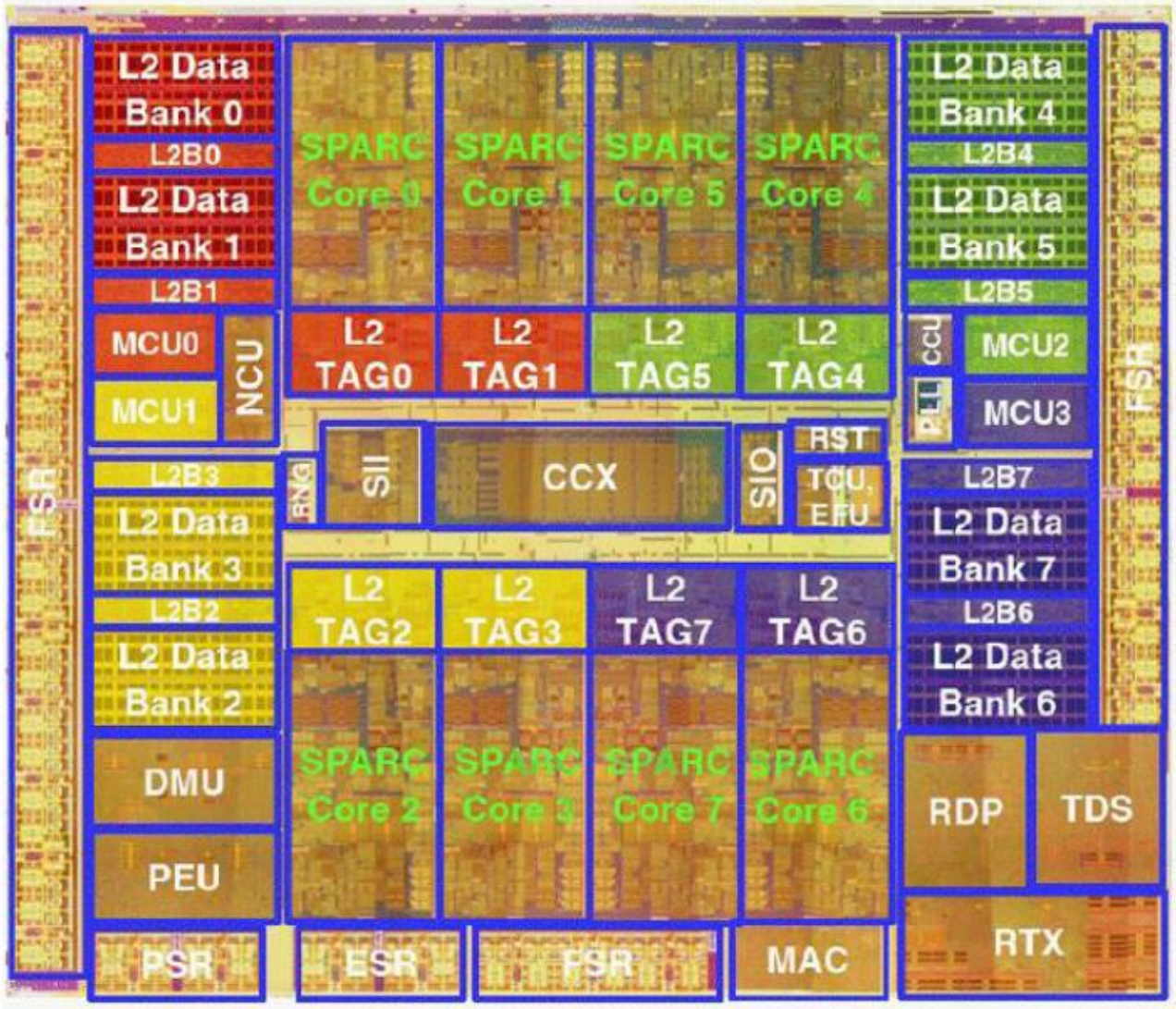


Multi-banked Caches



- Partition address space into multiple banks
  - Bank0 caches addresses from partition 0, bank1 from partition 1…
  - Can use least or most significant address bits for partitioning (block address)
    - What are the advantages of each approach?
- Benefits: accesses can go in parallel if no conflicts
- Challenges: conflicts, distribution network, bank utilization

Sun UltraSPARC T2: 8-bank L2 cache



Classification of Cache Optimizations

- Reduce Miss Penalty
- Reduce Miss Rate
- Reduce Hit Time

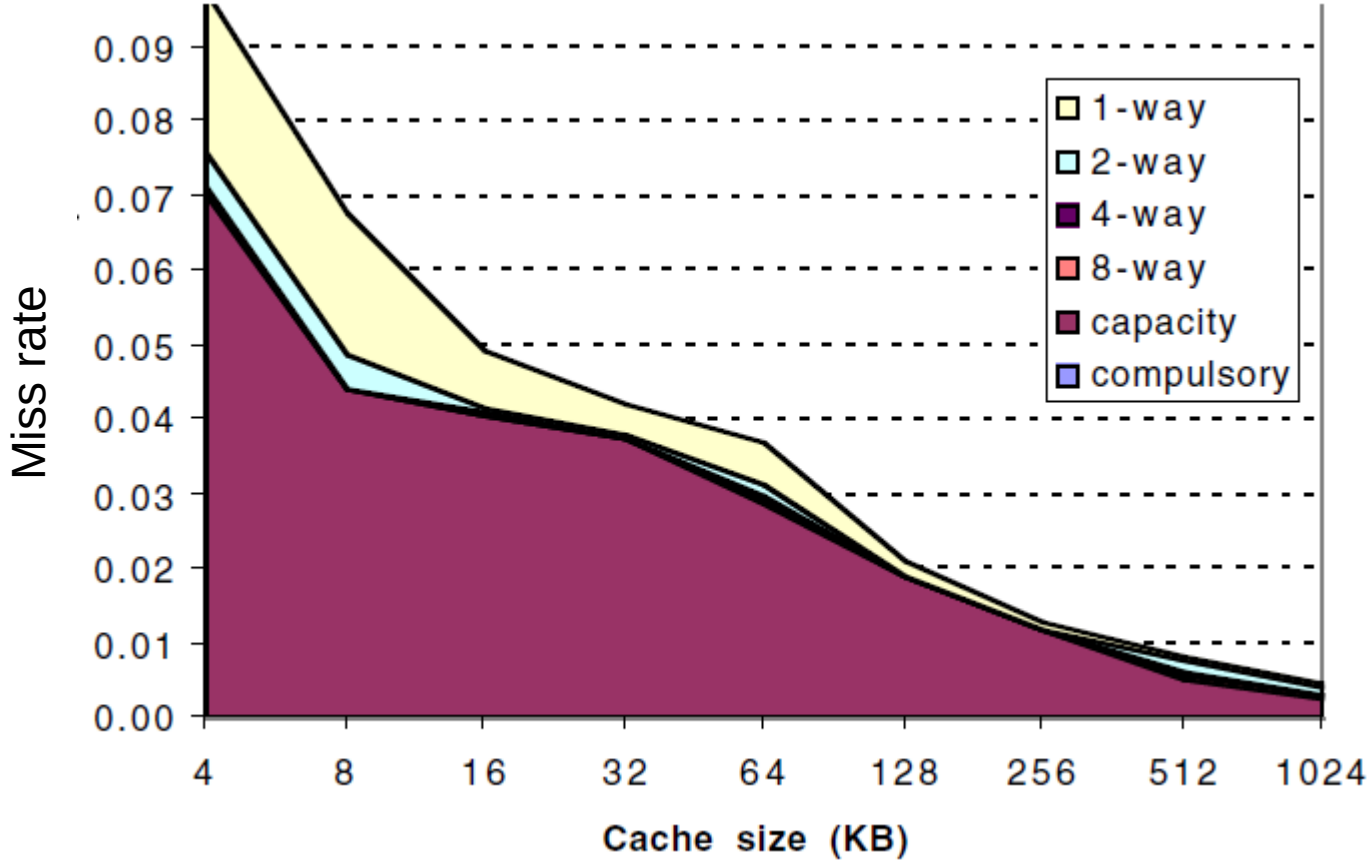
3 C's model

Characterization of cache misses

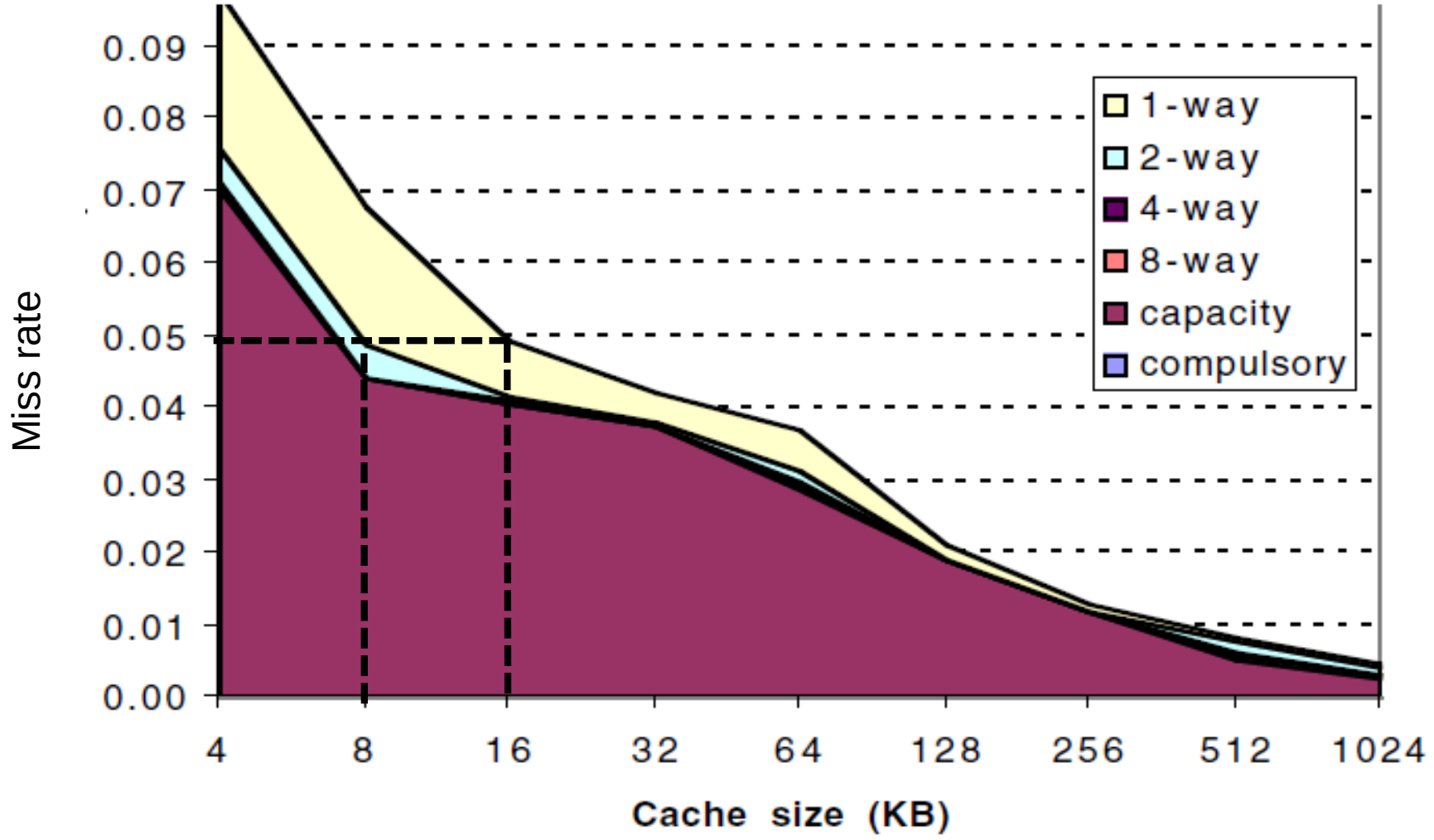
- Compulsory miss: Miss that happens due to the first access to a block since program began execution. Also called cold-start miss.
- Capacity miss: Miss that happens because a block that has been fetched in the cache needed to be replaced due to limited capacity (all blocks valid in the cache, cache needed to select victim block). Block had been fetched, replaced, and re-fetched to count as capacity miss.
- Conflict miss: MIss that happens because address of block maps to same location in the cache with other block(s) in memory. Block had been fetched, replaced, re-fetched, and cache has invalid locations that could hold the block if a different address mapping scheme were used, to count as conflict miss (as opposed to compulsory miss with first-time fetch).

Associativity and conflict misses

- Compulsory misses are those that occur in an infinite cache
- Capacity misses are those that occur in a fully associative cache
- Conflict misses are those that occur going from fully associative to 8-way associative, 4-way associative, and so on

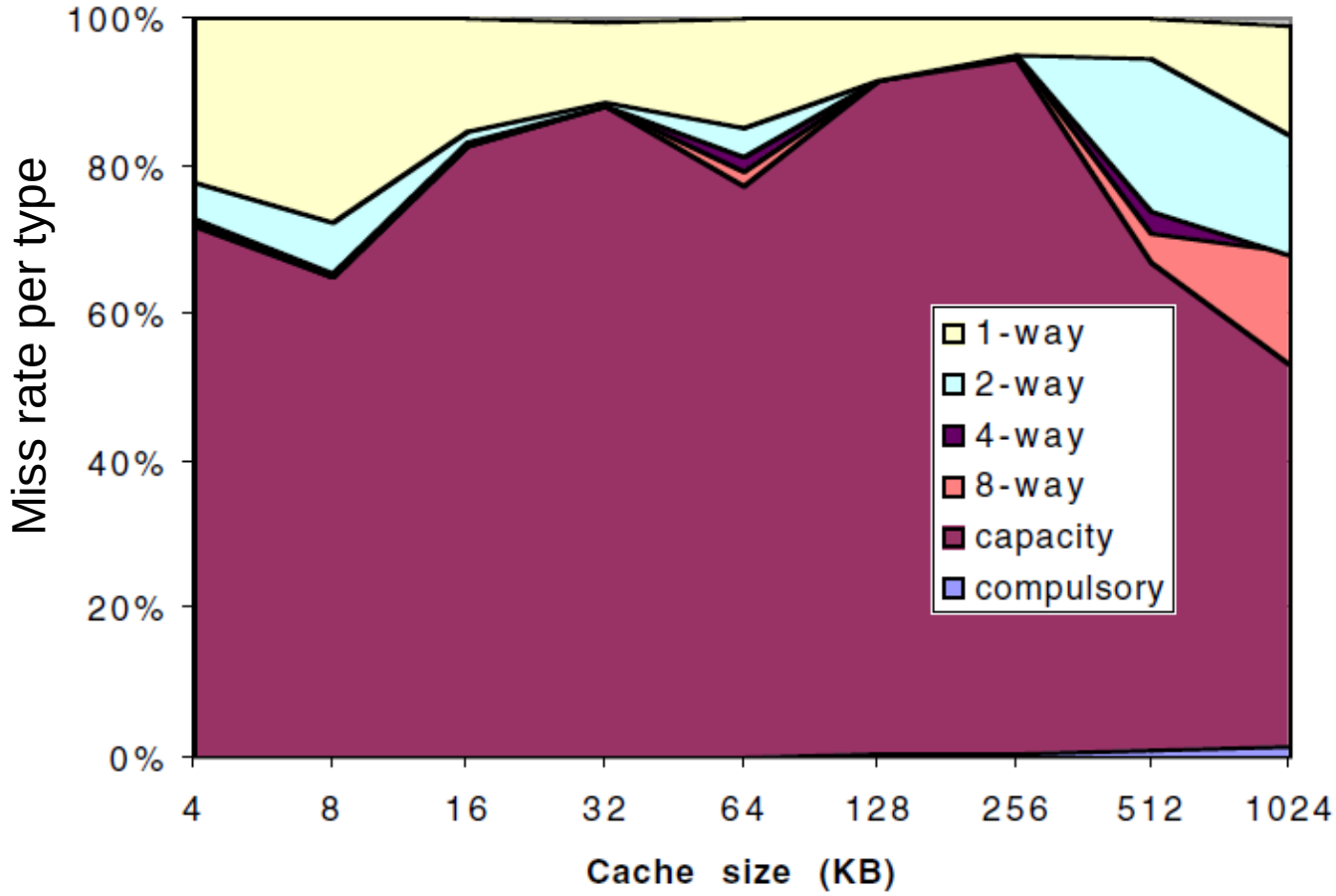


2 to 1 cache rule



miss rate 1-way associative cache of size X = miss rate 2-way associative cache of size X/2

Miss rate distribution



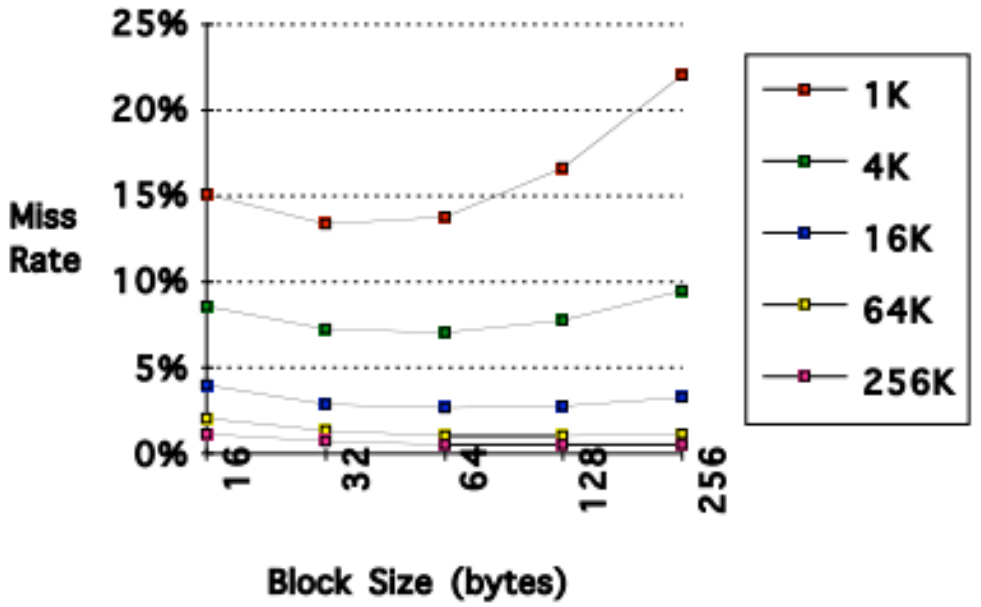
- Associativity tends to increase in modern caches (for example 8-way L1 and 16-way L3)
- Increased associativity may result in complex design and slow clock

7. Increasing the block size

Spatial locality

- Larger block size usually reduces compulsory misses
- Larger block size increases miss penalty, since processor needs to fetch more data
- Increasing block size may increase conflict misses, if spatial locality is poor (most words in fetched block not used)
- Increasing block size may increase capacity misses, if spatial locality is poor (most words in fetched block not used)

Block size impact



Miss rate versus block size

SPEC92 benchmarks

	Cache size			
Block size	**4K**	**16K**	**64K**	**256K**
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

AMAT versus block size

SPEC92 benchmarks

- Example assumes 80-cycle memory latency, 16 bytes per 2 cycles pipelined memory throughput

		Cache size			
Block size	**Miss penalty**	**4K**	**16K**	**64K**	**256K**
16	82	8.027	4.231	2.673	1.894
32	84	7.082	3.411	2.134	1.588
64	88	7.160	3.323	1.933	1.449
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

Hit time = 1, Miss rate from previous slide
AMAT = Hit Time + Miss Penalty x Miss Rate

8. Larger Caches

Implications of higher cache capacity

- Reduction of capacity misses
- Longer hit time
- Increased area, power, and cost
  - Very large multi-MB caches often placed off-chip

AMAT = Hit Time + Miss Rate * Miss Penalty

9. Increasing Associativity

Example

- Higher associativity increases hit time
- Increased hit time for L1 cache means increased cycle time
- Assume

$$\text{Hit time} = 1.0 \text{ cycle}$$

$$\text{Miss penalty}_{direct-mapped} = 25 \text{ cycles to perfect L2 cache}$$

$$\text{Clock cycle time}_{2-way} = 1.36 \times \text{Clock cycle time}_{1-way}$$

$$\text{Clock cycle time}_{4-way} = 1.44 \times \text{Clock cycle time}_{1-way}$$

$$\text{Clock cycle time}_{8-way} = 1.52 \times \text{Clock cycle time}_{1-way}$$

$$AMAT_{8-way} < AMAT_{4-way}?$$

$$AMAT_{4-way} < AMAT_{2-way}?$$

$$AMAT_{2-way} < AMAT_{1-way}?$$

Increasing Associativity

Example

- Assume

$$\text{Hit time} = 1.0 \text{ cycle}$$

$$\text{Miss penalty}_{direct-mapped} = 25 \text{ cycles to perfect L2 cache}$$

$$\text{Clock cycle time}_{2-way} = 1.36 \times \text{Clock cycle time}_{1-way}$$

$$\text{Clock cycle time}_{4-way} = 1.44 \times \text{Clock cycle time}_{1-way}$$

$$\text{Clock cycle time}_{8-way} = 1.52 \times \text{Clock cycle time}_{1-way}$$

$$AMAT_{8-way} = 1.52 + \text{Miss rate}_{8-way} \times 25.0$$

$$AMAT_{4-way} = 1.44 + \text{Miss rate}_{4-way} \times 25.0$$

$$AMAT_{2-way} = 1.36 + \text{Miss rate}_{2-way} \times 25.0$$

$$AMAT_{1-way} = 1.00 + \text{Miss rate}_{1-way} \times 25.0$$

AMAT versus Associativity

SPEC92 benchmarks

	Associativity			
Cache size (KB)	One-way	Two-way	Four-way	Eight-way
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.23	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.14	2.18	2.25
128	1.32	1.66	1.75	1.82
256	1.20	1.55	1.59	1.66

Miss rates from Computer Architecture book

Makes more sense to have high associativity in L2, L3 caches where local miss rate is high

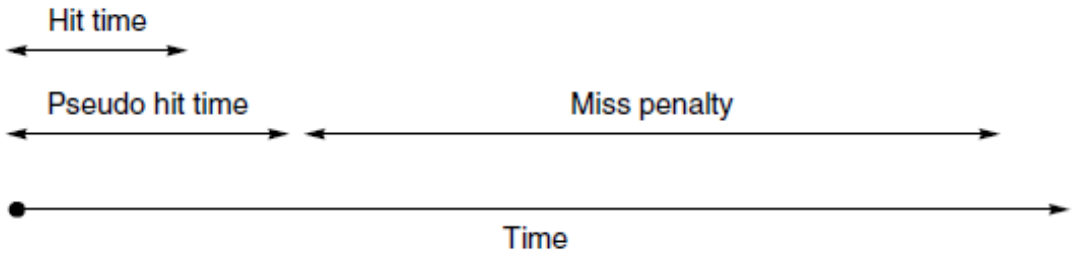
10. Way Prediction

Way prediction

- Predict way (cache bank) of next cache access
  - Block predictor bit per way
  - 85% accuracy on Alpha 21264
- Multiplexor set early to select predicted block
- If tag match succeeds (hit) predicted block returned in one cycle
- If tag match fails (miss) rest of the blocks checked in second cycle
  - Two hit times: fast hit (way predicted), slow hit (way mispredicted)
- Effective technique for power reduction
  - Supply power only to tag arrays of selected ways

Pseudoassociativity

- Cache organized as direct-mapped cache with pseudosets
  - Pseudosets contain blocks in different lines of the tag/data arrays
- Hit processed as in direct-mapped cache
- On miss, check other block in pseudo-set
- Two hit times: fast hit (hit, direct mapped), slow hit (hit pseudoset)
- Increases hit time of direct-mapped cache, especially if slow hits are many
- May increase miss penalty, overhead to select pseudo-way



A pseudo-associative cache tests each possible way one at a time.

11. Prefetching

- Idea: fetch data into the cache before processors request them
  - Can address cold misses
  - Can be done by the programmer, compiler, or hardware

- Characteristics of ideal prefetching
  - You only prefetch data that are truly needed
    - Avoid bandwidth waste
  - You issue prefetch requests early enough
    - To hide the memory latency
  - You don’t issue prefetch requests too early
    - To avoid cache pollution

Software Prefetching

```
for (i=0; i<N; i++) {
   __prefetch(a[i+8]);
   __prefetch(b[i+8]);
   sum += a[i]*b[i];
}
```

Doesn't have to be correct!

```
   __prefetch(-1);
```

- Issues software prefetching
  - Takes up issue slots
    - Not big issue with superscalar
  - Takes up system bandwidth
  - Must have non-blocking caches
  - Prefetch distance depends on specific system implementation
    - Non-portable code
  - Not easy to use for pointer based structures
  - Requires ninja programmer/compiler!

Hardware Prefetching

- Same goal with software prefetching but initiated by hardware
  - Can tune to specific system implementation
  - Does not waste instruction issue bandwidth
  - More portable code
- Major design questions
  - Where to place a prefetch engine?
    - L1, L2, …
  - What to prefetch?
    - Next sequential cache line(s), strided patterns, pointers, …
  - When to prefetch?
    - On a load, on a miss, when other prefetched data used, …
  - Where to place prefetched data
    - In the cache or in a special prefetch buffer
  - How to handle VM exceptions?
    - Don’t prefetch beyond a page?

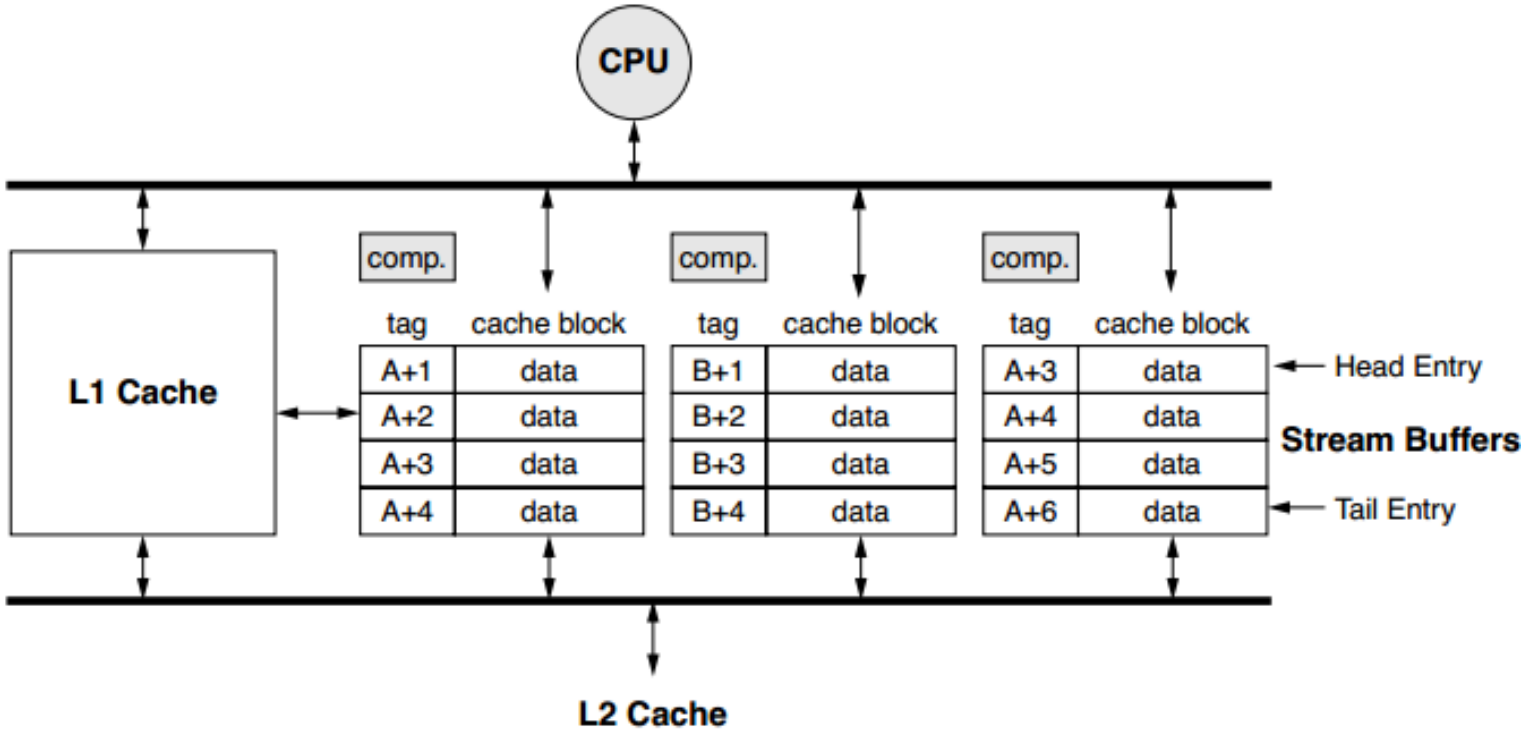
Simple Sequential Prefetching

- On a cache miss, fetch two sequential memory blocks
  - Exploits spatial locality in both instructions & data
  - Exploits high bandwidth for sequential accesses
- Called “Adjacent Cache Line Prefetch” or “Spatial Prefetch” by Intel
- Extend to fetching N sequential memory blocks
  - Pick N large enough to hide the memory latency

Stream Prefetching

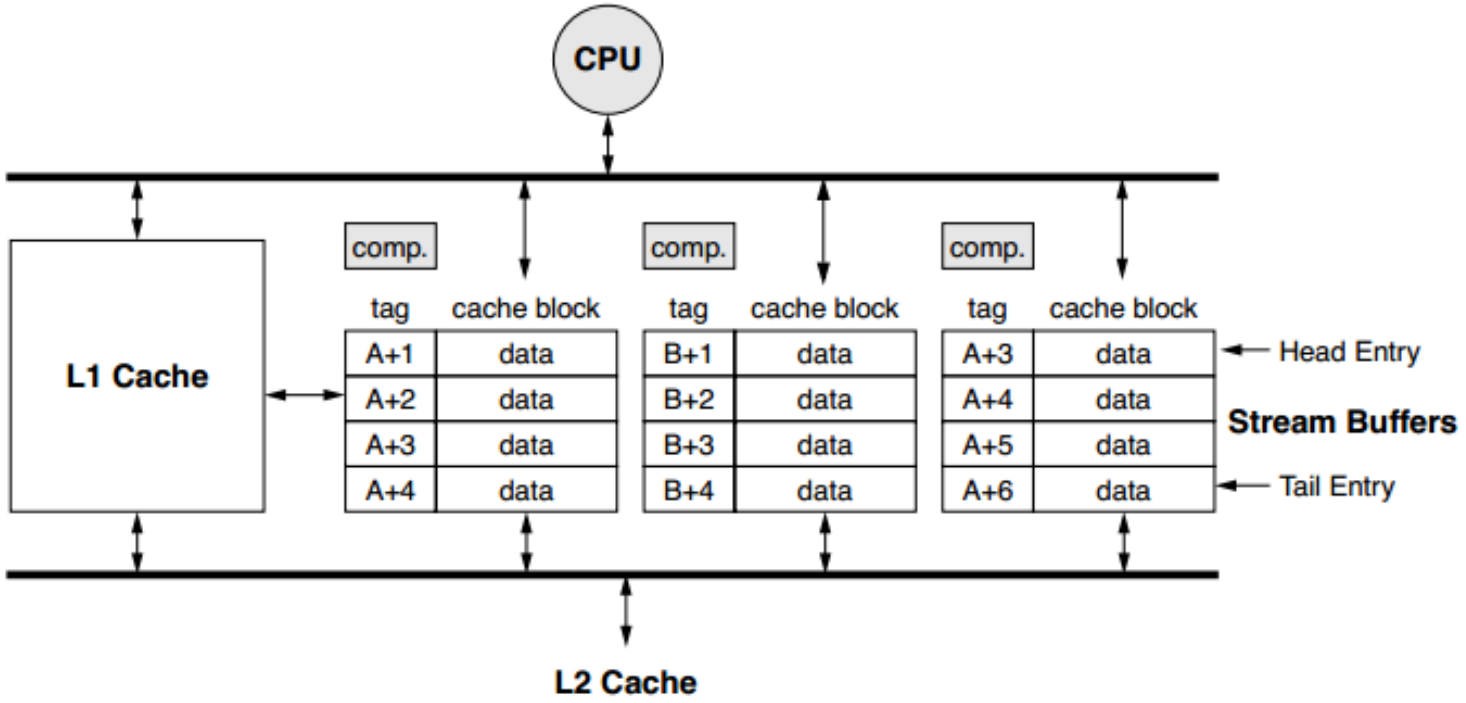
- Sequential prefetching problem
  - Performance slows down once every N cache lines
- Stream prefetching is a continuous version of prefetching
  - Stream buffer can fit N cache lines
  - On a miss, start fetching N sequential cache lines
  - On a stream buffer hit:
    - Move cache line to cache, start fetching line (N+1)
- In other words, stream buffer tries to stay N cache lines ahead
- Design issues
  - When is a stream buffer allocated
  - When is a stream buffer released
  - Can use multiple stream buffers to capture multiple streams
    - E.g. a program operating on 2 arrays

Stream Buffer Design



- Each buffer fetches data from one contiguous stream
- Cache and head entries of stream buffers checked upon access
- Cache miss may be served by head of stream buffer

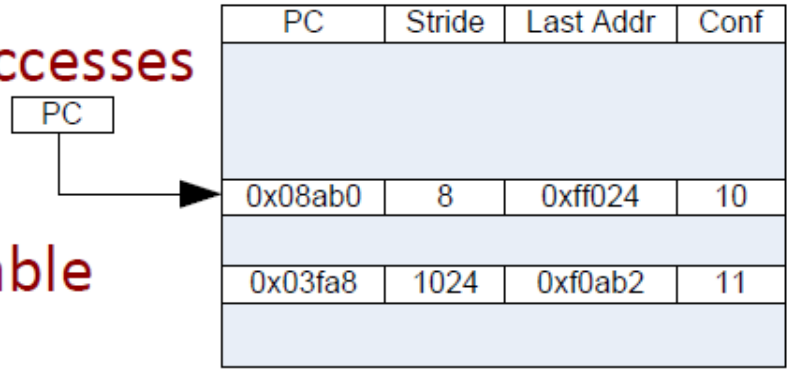
Stream Buffer Design



- If cache miss hits on stream buffer, head pointer moves down and prefetching is triggered
- Available bit per entry indicates if prefetching is in flight
- Buffer allocated when a stream of misses (e.g. address A, A+1,...) is detected

Strided Prefething

- Idea: detect and prefetch strided accesses
  - for (i=0; i<N; i++) A[i*1024]++;
- Stride detected using a PC-based table
  - For each PC, remember the stride
  - Stride detection
    - Remember the last address used for this PC
    - Compare to currently used address for this PC
  - Track confidence using a two bit saturating counter
    - Increment when stride correct, decrement when incorrect
- How to use the PC-based table
  - Similar to stream prefetching except using stride instead of +1



Sandybridge Prefetching (Intel Core i7-2600K)

- "Intel 64 and IA-32 Architectures Optimization Reference Manual, Jan 2011", pg 2-24

Two hardware prefetchers load data to the L1 DCache:

- **Data cache unit (DCU) prefetcher**. This prefetcher, also known as the streaming prefetcher, is triggered by an ascending access to very recently loaded data. The processor assumes that this access is part of a streaming algorithm and automatically fetches the next line.
- **Instruction pointer (IP)-based stride prefetcher**. This prefetcher keeps track of individual load instructions. If a load instruction is detected to have a regular stride, then a prefetch is sent to the next address which is the sum of the current address and the stride. This prefetcher can prefetch forward or backward and can detect strides of up to 2K bytes.

http://www.intel.com/Assets/PDF/manual/248966.pdf

Other Ideas in Prefetching

- Prefetch for pointer-based data structures
  - Predict if fetched data contain a pointer & follow it
  - Works for linked-lists, graphs, etc
  - Must be very careful:
    - What is a pointer?
    - How far to prefetch?

- Different correlation techniques
  - Markov prefetchers
  - Delta correlation prefetchers

12. Compiler Optimizations

Cache-aware optimizations in software

- Code transformations to improve:
  - Spatial locality, through higher utilization of fetched cache blocks
  - Temporal locality, through reduction of the reuse distance of cache blocks
  - Examples: loop interchange, loop blocking, loop fusion, loop fision
- Data layout and data structure transformations to improve:
  - Spatial locality, through higher utilization of fetched cache blocks
  - Examples: array merging, structure/object class member reordering in memory, block array layouts

Array merging

Data structure reorganization for spatial locality

```
/* Before */
int val[SIZE];
int key[SIZE];
```

- Assume code accessing val[i], key[i], for every i
- Accesses to val and key may conflict in direct-mapped caches
- Solution, merge arrays, accesses to val[i], key[i] do not conflict in the cache, spatial locality exploited

```
/* After */
struct merge {
  int val;
  int key;
}
struct merge merged_array[SIZE];
```

Loop Interchange

```
/* Before */
for (j = 0; j < 100; j = j+1)
        for (i = 0; i < 5000; i = i+1)
                x[i][j] = 2 * x[i][j];

/* After */
for (i = 0; i < 5000; i = i+1)
        for (j = 0; j < 100; j = j+1)
                x[i][j] = 2 * x[i][j];
```

- Row-Major order: rows stored one after the other (i is row index, j is column index)
- Exchange the nesting of loops taking advantage of spatial locality. Maximize use of a cache block before it is replaced.

Data blocking

Code transformations for temporal locality

- Reduce reuse distance for same data
- Organize code so that data is accessed in blocks
- Best performance if block accessed many times and few accesses to data outside block

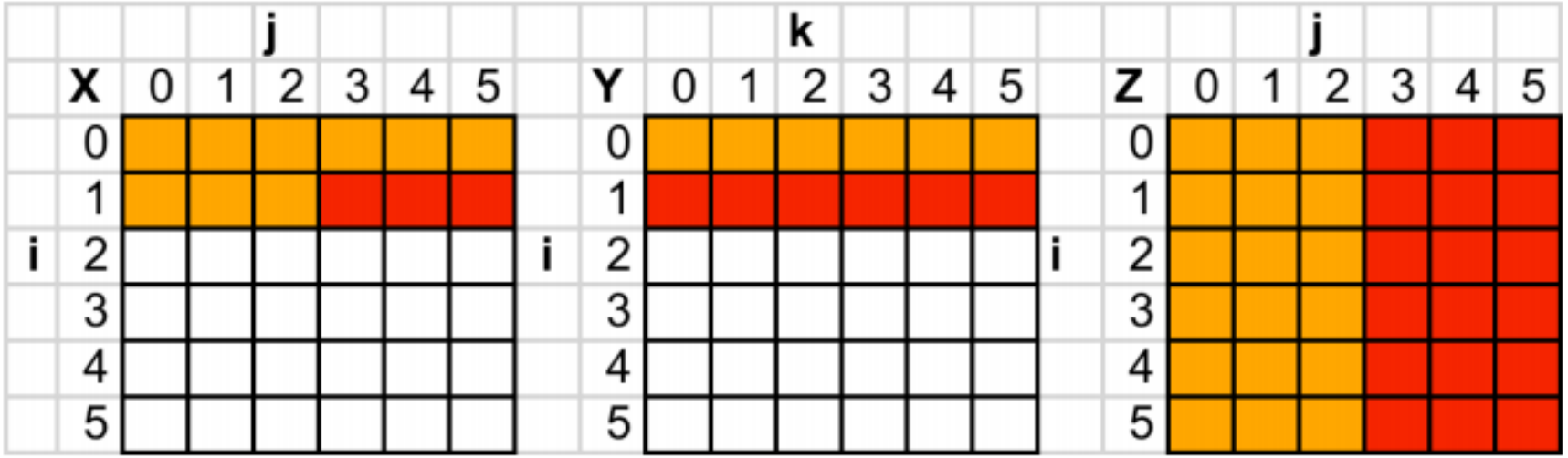
Example: Matrix multiplication

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    {r = 0;
      for (k = 0; k < N; k  = k + 1)
        r = r + y[i][k]*z[k][j];
       x[i][j] = r;
};
```

Data blocking

Array accesses without blocking

- Snapshot with i=1
- Assume cache line holds one array element
- Two innermost loops access N^2 elements of z, N elements of y, N elements of x
- $N \times (N^2 + 2N) = 2N^2 + N^3$ memory accesses
- Need cache space at least $N^2 + N$ to exploit temporal locality



Total required cache space to exploit locality = N^2(for Z) + N(for Y) → Total misses = $2N^2 + N^3$

Data blocking

Example: Blocked matrix multiplication

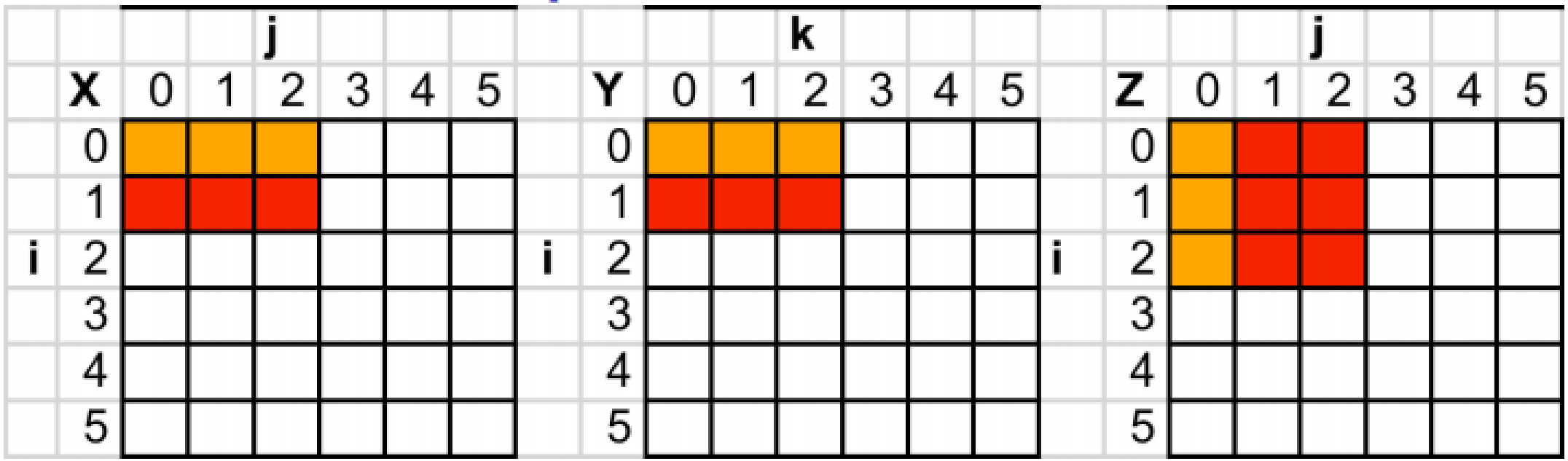
```
/* After */
for (jj = 0; jj < N; jj = jj+B)
  for (kk = 0; kk < N; kk = kk+B)
    for (i = 0; i < N; i = i+1)
      for (j = jj; j < min(jj+B,N); j = j+1)
        {r = 0;
          for (k = kk; k < min(kk+B,N); k = k + 1)
            r = r + y[i][k]*z[k][j];
          x[i][j] = x[i][j] + r;
        };
```

- Load a block of z of size $B \times B$
- Compute partial sum for B elements of x
- Load next block
- $\frac{N}{B} \times \frac{N}{B} \times (N \times 2B + B^2) = \frac{2N^3}{B} + N^2$ memory accesses

Total required cache space to exploit locality = B^2(for Z) + B(for Y)

Data blocking

Blocked matrix multiplication



Classification of Cache Optimizations

- Reduce Miss Penalty
- Reduce Miss Rate
- Reduce Hit Time?
  - Small and Simple Caches
  - Virtually Addressed Caches
  - Pipelined Caches
  - Trace Caches (storing traces of instructions)