CSCI-564 Advanced Computer Architecture

Lecture 5: Advanced Cache

Bo Wu

Colorado School of Mines

Cache misses are our enemy…

Know the Enemy

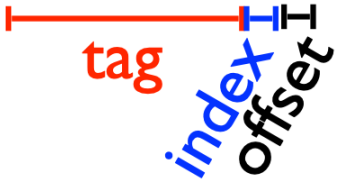
- Misses happen for different reasons
- The three C's (types of cache misses)
  - Compulsory: The program has never requested this data before. A miss is mostly unavoidable.
  - Conflict: The program has seen this data, but it was evicted by another piece of data that mapped to the same "set"
  - Capacity: The program is actively using more data than the cache can hold.

A Simple Example

- Consider a direct mapped cache with 16 blocks, a block size of 16 bytes, and the application repeat the following memory access sequence:
  - 0x80000000, 0x80000008, 0x80000010, 0x80000018, 0x30000010

A Simple Example

- a direct mapped cache with 16 blocks, a block size of 16 bytes
  - 16 = 2^4 : 4 bits are used for the index
  - 16 = 2^4 : 4 bits are used for the byte offset
  - The tag is 32 - (4 + 4) = 24 bits
  - For example: 0x80000010



A Simple Example

	valid	tag	data
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000

0x80000008

0x80000010

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000
0x80000008
0x80000010
0x80000018
0x30000010
0x80000000
0x80000008
0x80000010
0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008

0x80000010

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1	1	800000	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1	1	800000	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 miss: compulsory

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1	1	800000	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 miss: compulsory

0x80000018 hit!

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1	1	300000	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 miss: compulsory

0x80000018 hit!

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1	1	300000	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 miss: compulsory

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1	1	300000	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 miss: compulsory

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 hit!

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1	1	300000	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 miss: compulsory

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 hit!

0x80000008 hit!

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1	1	800000	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 miss: compulsory

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 hit!

0x80000008 hit!

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1	1	800000	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 miss: compulsory

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 hit!

0x80000008 hit!

0x80000010 miss: conflict

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1	1	800000	
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 miss: compulsory

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 hit!

0x80000008 hit!

0x80000010 miss: conflict

0x80000018 hit!

A Simple Example: Increased Cache line Size

- Consider a direct mapped cache with 8 blocks, a block size of 32 bytes, and the application repeat the following memory access sequence:
  - 0x80000000, 0x80000008, 0x80000010, 0x80000018, 0x30000010

A Simple Example

- a direct mapped cache with 8 blocks, a block size of 32 bytes
  - 8 = 2^3 : 3 bits are used for the index
  - 32 = 2^5 : 5 bits are used for the byte offset
  - The tag is 32 - (3 + 5) = 24 bits
  - For example: 0x80000010 =
    - 01110000000000000000000000010000

A Simple Example

	valid	tag	data
0			
1			
2			
3			
4			
5			
6			
7			

0x80000000
0x80000008
0x80000010
0x80000018
0x30000010
0x80000000
0x80000008
0x80000010
0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			

0x80000000
0x80000008
0x80000010
0x80000018
0x30000010
0x80000000
0x80000008
0x80000010
0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			

0x80000000 miss: compulsory

0x80000008

0x80000010

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	300000	
1			
2			
3			
4			
5			
6			
7			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	300000	
1			
2			
3			
4			
5			
6			
7			

0x80000000	miss: compulsory
0x80000008	hit!
0x80000010	hit!
0x80000018	hit!
0x30000010	miss: compulsory
0x80000000	
0x80000008	
0x80000010	
0x80000018	

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 miss: conflict

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			

0x80000000	miss: compulsory
0x80000008	hit!
0x80000010	hit!
0x80000018	hit!
0x30000010	miss: compulsory
0x80000000	miss: conflict
0x80000008	hit!
0x80000010	
0x80000018	

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 miss: conflict

0x80000008 hit!

0x80000010 hit!

0x80000018

A Simple Example

	valid	tag	data
0	1	800000	
1			
2			
3			
4			
5			
6			
7			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 miss: conflict

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

A Simple Example: Increased Associativity

- Consider a 2-way set associative cache with 8 blocks, a block size of 32 bytes, and the application repeat the following memory access sequence:
  - 0x80000000, 0x80000008, 0x80000010, 0x80000018, 0x30000010

A Simple Example

- a 2-way set-associative cache with 8 blocks, a block size of 32 bytes
  - The cache has 8/2 = 4 sets: 2 bits are used for the index
  - 32 = 2^5 : 5 bits are used for the byte offset
  - The tag is 32 - (2+ 5) = 25 bits
  - For example: 0x80000010 =
    - 01110000000000000000000000010000

A Simple Example

	valid	tag	data
0			
1			
2			
3			

0x80000000
0x80000008
0x80000010
0x80000018
0x30000010
0x80000000
0x80000008
0x80000010
0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
1			
2			
3			

0x80000000
0x80000008
0x80000010
0x80000018
0x30000010
0x80000000
0x80000008
0x80000010
0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
1			
2			
3			

0x80000000 miss: compulsory

0x80000008

0x80000010

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
1			
2			
3			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
1			
2			
3			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
1			
2			
3			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
	1	600000	
1			
2			
3			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
	1	600000	
1			
2			
3			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
	1	600000	
1			
2			
3			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 hit!

0x80000008

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
	1	600000	
1			
2			
3			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 hit!

0x80000008 hit!

0x80000010

0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
	1	600000	
1			
2			
3			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 hit!

0x80000008 hit!

0x80000010 hit!

0x80000018

A Simple Example

	valid	tag	data
0	1	1000000	
	1	600000	
1			
2			
3			

0x80000000 miss: compulsory

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

0x30000010 miss: compulsory

0x80000000 hit!

0x80000008 hit!

0x80000010 hit!

0x80000018 hit!

Reducing Compulsory Misses

- Increase cache line size so the processor requests bigger chunks of memory
- This only works if there is good spatial locality, otherwise you are bringing in data you don't need
  - If you are reading a few bytes here and a few bytes there (i.e., no spatial locality) this will hurt performance
  - But it will help in cases like this

```
for(i = 0;i < 1000000; i++) {
   sum += data[i];
}
```

Prefetching

- Speculate on future instruction and data accesses and fetch them into cache
  - Instruction accesses easier to predict than data accesses
- Varieties of prefetching
  - Hardware prefetching
  - Software prefetching
  - Mixed schemes

Reducing Compulsory Misses

- Hardware Prefetching

```
for(i = 0;i < 1000000; i++) {
   sum += data[i];
}
```

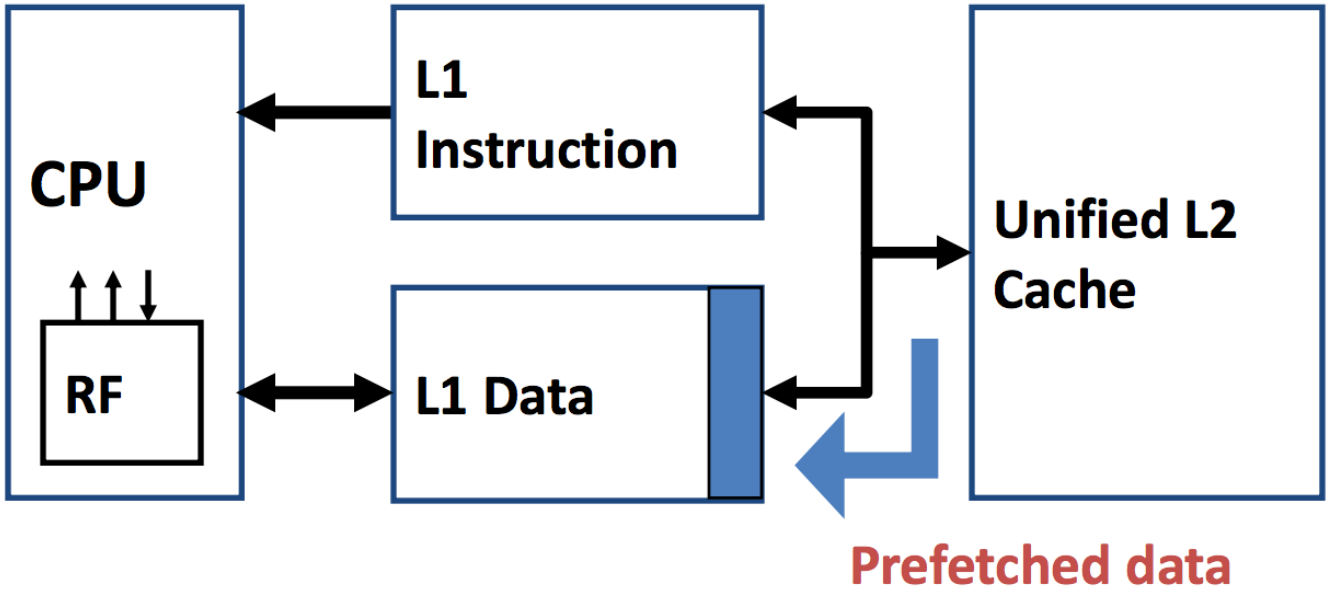
  - In this case, the processor could identify the pattern and proactively prefetch data program will ask for
  - Pattern: nextAddr = curAddr + 4

Hardware Prefetching

- Prefetch-on-miss
  - prefetch b+1 upon miss on b
- One block lookahead scheme
  - initiate prefetch for block b+1 when block b is accessed
  - Can extend to N-block lookahead
- Strided prefetch
  - If observe sequence of accesses to block b, b+N, b+2N, then prefetch b+3N etc.

Issues in Prefetching

- Usefulness – should produce hits
- Timeliness – not late and not too early
- Cache and bandwidth pollution



Software Prefetching

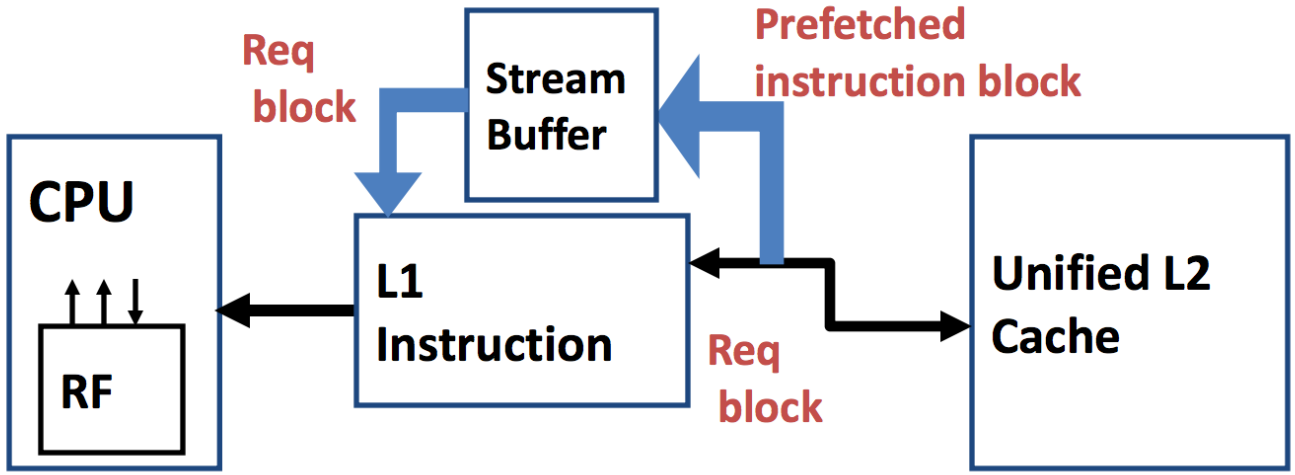
```
for(i=0; i < N; i++) {

    SUM = SUM + a[i] * b[i];
}
```

- Timing is the biggest issue, not predictability
  - If you prefetch very close to when the data is requested, you may be too late
  - Prefetch too early, cause pollution
  - Estimate how long it will take for the data to come into l1 cache, so we can set P appropriately
  - Why is this hard to do?

Hardware Instruction Prefetching

- Fetch two blocks on a miss; the requested block (i) and the next consecutive block (i+1)
- Requested block placed in cache, and the next block in instruction stream buffer
- If miss in cache but hit in stream buffer, move stream buffer block into cache and prefetch next block (i+2)



Restructuring

```
struct Atom {
    double3 v,
    double3 f,
    double3 p
};

Atom atoms[N];
```

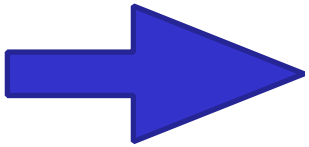
```
for (i=0; i<N; ++i)
    …  = atoms[i].f + …

for (i=0; i<N; ++i)
    …  = atoms[i].v - …

for (i=0; i<N; ++i)
    …  = atoms[i].p + …
```

```
structure Atom {
    double3 v,
    double3 f,
    double3 p
};

Atom atoms[N];
```



```
double3 vs[N];
Double3 fs[N];
double3 ps[N];
```

Conflict Misses

- Conflict misses occur when the data we need was in the cache previously but got evicted
- Evictions occur because:
  - Direct mapped: Another request mapped to the same cache line
  - Associative: Too many other requests mapped to the same set

```
while(1) {
  for(i = 0;i < 1024*1024; i
  +=4096) {
     sum += data[i];
  } // Assume a 4 KB Cache
}
```

Colliding threads and data

- The stack and the heap tend to be aligned to large chunks of memory (maybe 128MB).

Capacity Misses

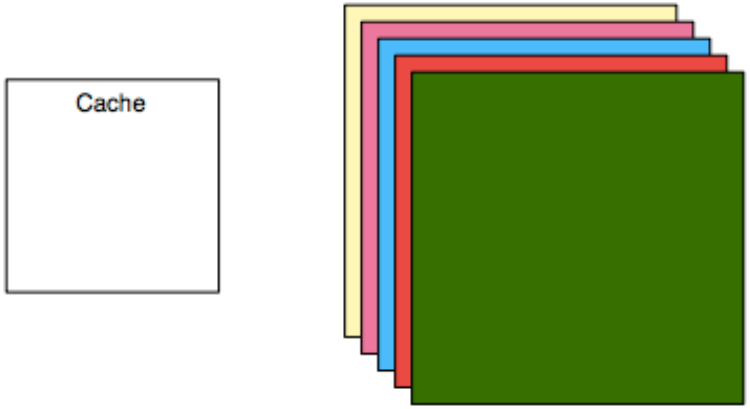
- Capacity misses occur because the processor is trying to access too much data
  - Working set: The data that is currently important to the program
  - If the working set is bigger than the cache, you are going to miss frequently
- Capacity misses are a bit hard to measure
  - Easiest definition: non-compulsory miss rate in an equivalently-sized fully-associative cache
  - Intuition: Take away the compulsory misses and the conflict misses, and what you have left are the capacity misses

Reducing Capacity Misses

- Increase capacity
- More associativity or more associative “sets”
  - Costs area and makes the cache slower
- Cache hierarchy do this implicitly already
  - if the working set “falls out” of the L1, you start using L2
- In practice, you make the L1 as big as you can within your cycle time and the L2 and L3 as big as you can while upping it on chip

Reducing capacity misses: the compiler

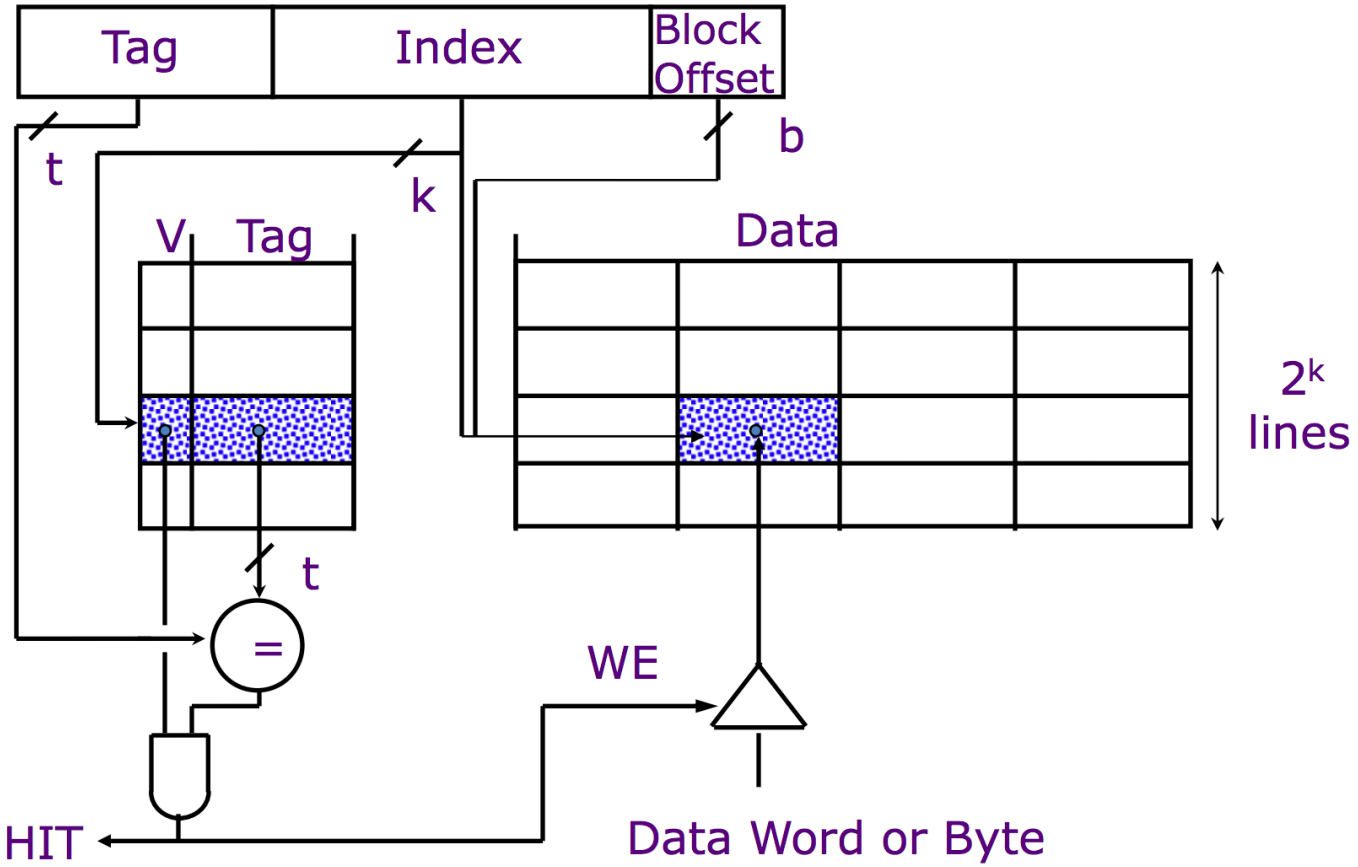
- Tiling
  - We need to makes several passes over a large array
  - Doing each pass in turn will "blow out" our cache
  - "blocking" or "tiling" the loops will prevent the blow out
  - Whether this is possible depends on the structure of the loop
- You can tile hierarchically, to fit into each level of the memory hierarchy.



More discussion about prefetching

- Affect capacity misses
- Affect conflict misses

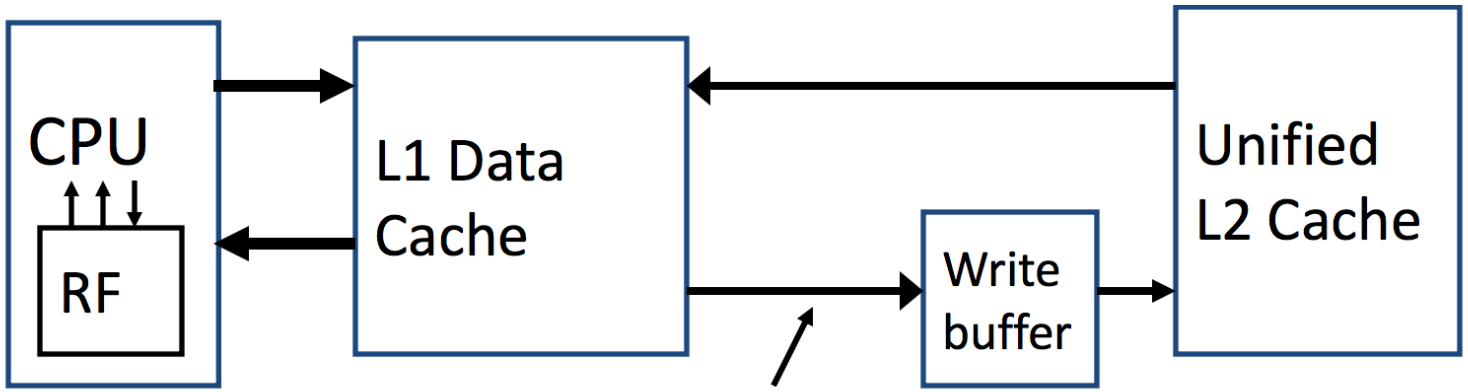
Write Performance



Reducing Write Time

- Problem: Writes take two cycles; One for tag check and the other for writing data
- Solution1
  - Step 1: Tag check, buffering old data and write data
  - Step 2: if tag check fails, write old data back
- Solution 2
  - Pipelining the writes

Reducing Miss Penalty

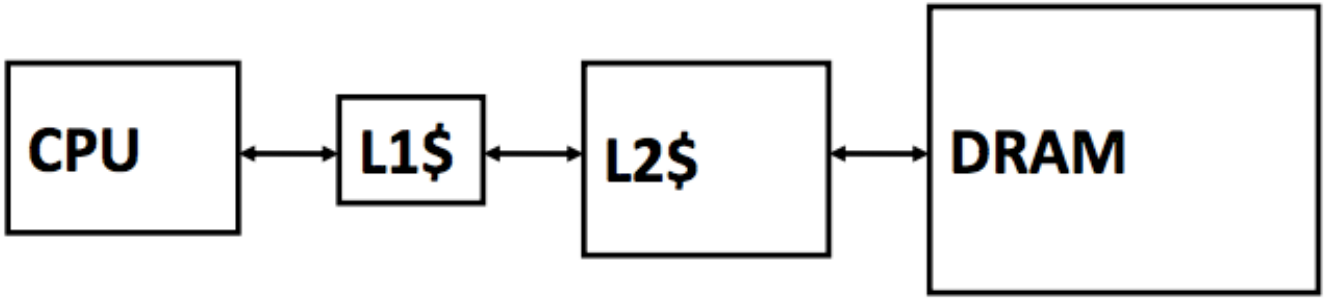


Evicted dirty lines for writeback cache
OR
All writes in writethrough cache

Multi-level Caches

Problem: A memory cannot be large and fast

Solution: Increasing sizes of cache at each level



Local miss rate = misses in cache / accesses to cache

Global miss rate = misses in cache / CPU memory accesses

Misses per instruction = misses in cache / number of instructions

Presence of L2 Influences L1 Design

- Use smaller L1 if there is also L2
  - Trade increased L1 miss rate for reduced L1 hit time and reduce L1 miss penalty
  - Reduce average access energy
- Use simpler write-through L1 with on-chip L2
  - Write-back L2 absorbs write traffic, doesn’t go off-chip

Inclusion Policy

- Inclusive multilevel cache
  - e.g., L2 cache holds copies of data in L1 cache
- Exclusive multilevel cache
  - e.g., L2 cache may hold data not in L1 cache

Why choose one type or the other?

Victim Cache

- Say a set has W ways, but W+2 lines are mapped to each set