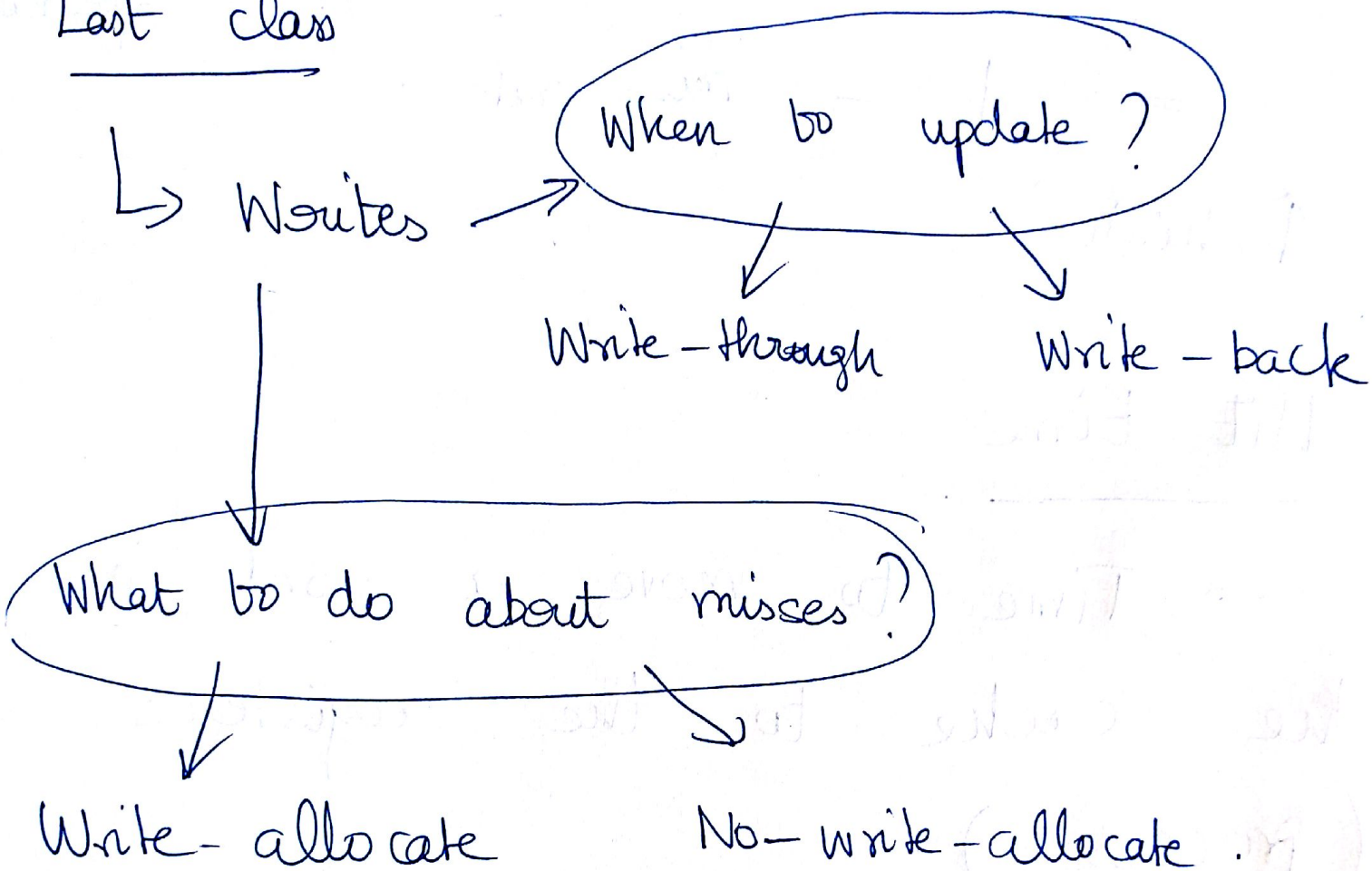
4/1

Last class



Cache Metrics

- **Miss Rate** = $\frac{\#\text{ of Misses}}{\#\text{ of References}}$.

↓ low

Hit rate

$$\frac{\#\text{ of hits}}{\#\text{ of references}}$$

= 1 − miss rate.

↑ high.

Hit time

→ Time to move a word in the cache to the registers (processor).

(includes → time for set, line and word within block. (selections))

↓ low as possible

Miss Penalty.

Additional time because of a miss.

If the miss is on L1, then miss penalty is the time it takes to read the word from L2.

How do different cache parameters affect cache performance?

↓

Block size,
cache size,
Line numbers (associativity)

Impact of Cache Size

increase ↑ Better hit rates!

Drawbacks.

↑ hit times.

Impact of Block Size

increase ↑ increase the hit rate. if programs have good spatial locality.

Drawbacks.

① For a given cache capacity, this means fewer lines.

Programs with temporal might not locality do so well.

② Transfer time

Impact of Associativity.

Increase E ↑

Fewer conflict misses due to thrashing.

↑ hit rate.

Drawbacks

① - Increased hit times.

② Increased miss penalty. (time to choose a victim block)

Faster memory → Smaller associativity

↓ low hit rate.

↓ Fast hit times.

↑ miss penalty for the next memory is not that high.

Slower memories → high associativity.

Impact of write strategies

Write - through vs Write back

simple. vs o not simple
need to store an additional bit //

transfer times ! vs. fewer transfers.

traffic is going to be high.

Cache Friendly Code.

Cache Organization IV

April 1, 2016 . Ganesh Kumar

Consider this program

```
int sumarray(int arr[8]) {
    int sum = 0;

    for (int i = 0; i < 8; i++)
        sum += arr[i];

    return sum;
}
```

Good temporal locality?

Good spatial locality?

Code

```
int sumarray(int arr[8]) {
    int sum = 0;

    for (int i = 0; i < 8; i++)
        sum += arr[i];

    return sum;
}
```

Good temporal locality?

YES! Variables **i** and **sum** are accessed repeatedly!

Good spatial locality?

Clearly! Variable **arr** is being accessed sequentially in a stride-1-reference pattern.

Assume the following,

- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

arr[i]	0	1	2	3	4	5	6	7
Access order	1[m]							

[h] - hit

[m] - miss

Assume the following,

- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

arr[i]	0	1	2	3	4	5	6	7
Access order	1[m]							

[h] - hit

[m] - miss

- **Now load block with arr[0] onto the cache.**
- **Since block size is 16 Bytes, the first four elements of the array get loaded into the cache.**

Assume the following,

- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

arr[i]	0	1	2	3	4	5	6	7
Access order	1[m]	2[h]	3[h]	4[h]				

[h] - hit

[m] - miss

- Now load block with arr[0] onto the cache.
- Since block size is 16 Bytes, the first four elements of the array get loaded into the cache.
- **Accessing arr[1], arr[2] and arr[3] will now be hits!**

Assume the following,

- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

arr[i]	0	1	2	3	4	5	6	7
Access order	1[m]	2[h]	3[h]	4[h]	5[m]			

[h] - hit

[m] - miss

- Now the loop will access arr[4].
- Miss!
- **Load the block containing arr[4] onto the cache.**

Assume the following,

- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

arr[i]	0	1	2	3	4	5	6	7
Access order	1[m]	2[h]	3[h]	4[h]	5[m]	6[h]	7[h]	8[h]

[h] - hit

[m] - miss

- Now the loop will access arr[4].
- Miss!
- Load the block containing arr[4] onto the cache.
- **Since block size is 16 bytes, arr[5], arr[6] and arr[7] will also get loaded.**
- **And when the loop accesses them, it will be cache hits!**

Assume the following,

- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

arr[i]	0	1	2	3	4	5	6	7
Access order	1[m]	2[h]	3[h]	4[h]	5[m]	6[h]	7[h]	8[h]

[h] - hit

[m] - miss

Miss Rate = # of Misses / # of References

Miss Rate = 2/8 = 0.25

In general, if a cache has a block size of B bytes, then stride-k-reference pattern will produce an average of

min (1, (wordsize * k) / B)

misses for each iteration of the loop

where k is expressed in words.

For our example,
Average misses = min (1, (4 * 1) / 16) => min (1, 0.25) => 0.25

We can expect an average of 0.25 misses for every iteration.