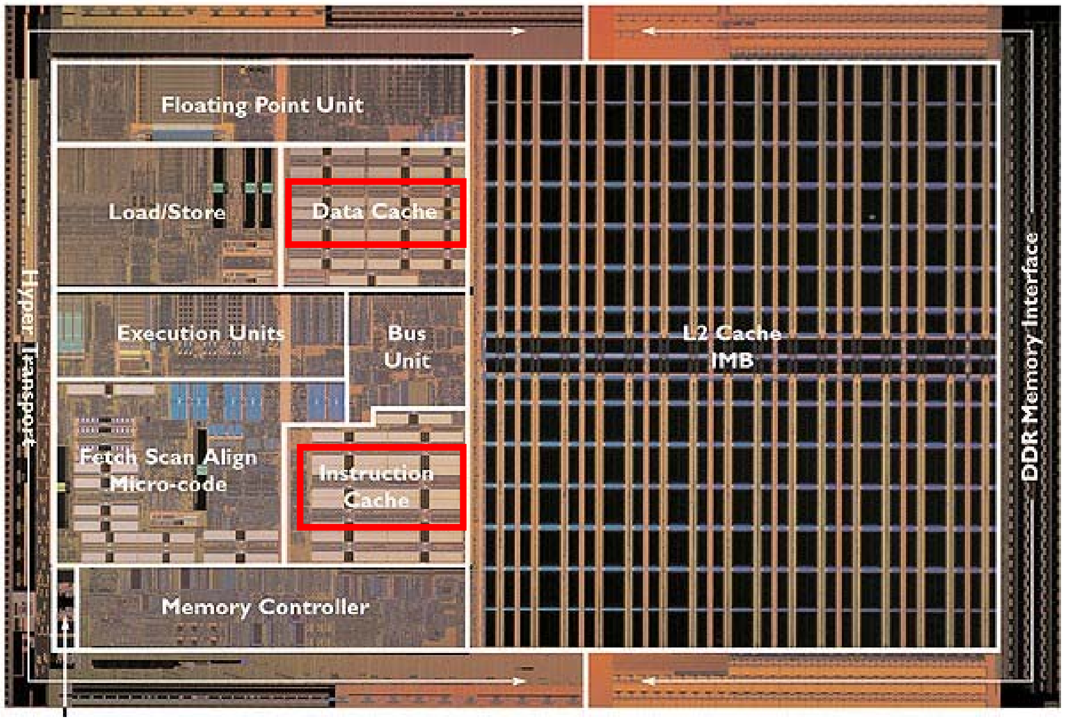
Cache performance and memory access patterns

Today's lecture

- Trade-offs between cache-size, block-size, associativity and performance (miss rate)
- Predicting Cache Performance from Code
  - Memory access patterns
  - Cache hit analysis for a given address stream

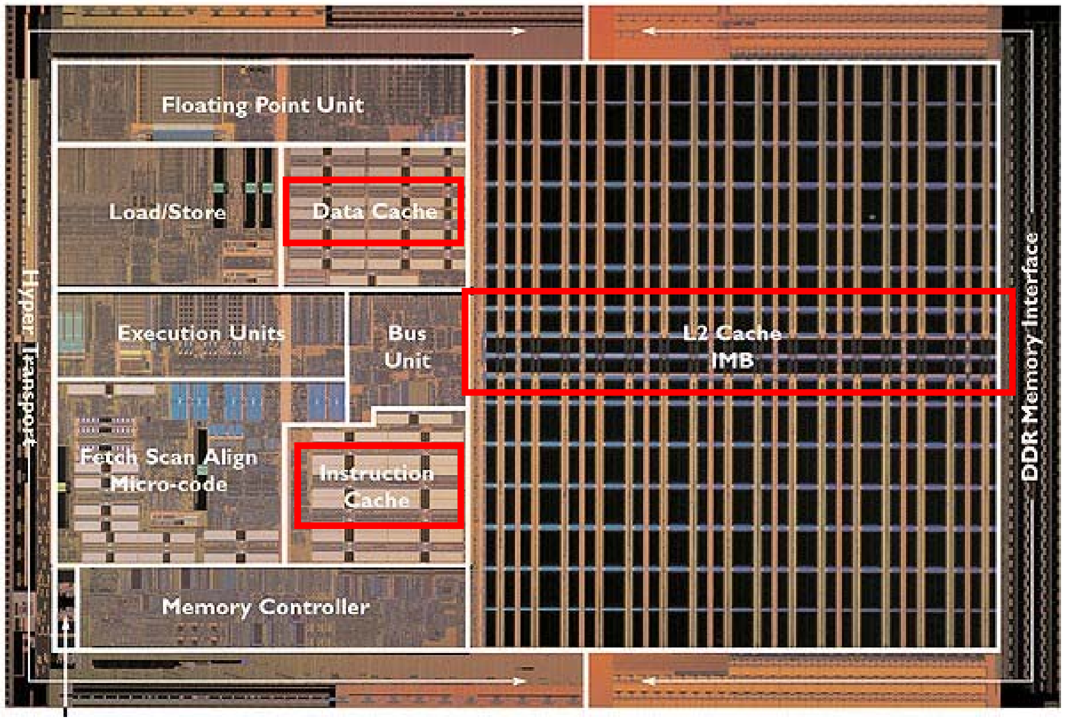
Most real architectures split instruction and data caches

- Pro: No structural hazard between IF & MEM
  - A single-ported, unified cache stalls fetch during load or store
- Con: Static partitioning of cache between instructions & data
  - Bad if working sets unequal: *e.g.,* code/**DATA** or **CODE**/data



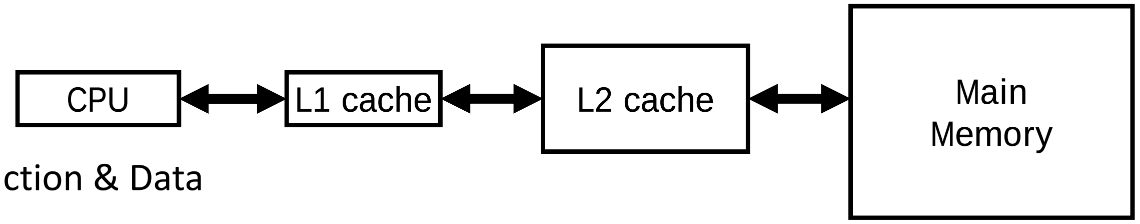
Most real architectures use multiple levels of caches

- Trade-off between access time & hit rate
  - L1 cache focuses on fast access time (with okay hit rate)
  - L2 cache focuses on good hit rate (with okay access time)



$$\begin{tikzcd}
\boxed{\text{CPU}} \arrow[r, leftrightarrow] & \boxed{\text{L1 cache}} \arrow[r, leftrightarrow] & \boxed{\text{L2 cache}} \arrow[r, leftrightarrow] & \boxed{\text{Main Memory}}
\end{tikzcd}$$

Example cache statistics from Opteron



- L1 Caches: Instruction & Data
  - 64 kB
  - 64 byte blocks
  - 2-way set associative
  - 2 cycle access time

- L2 Cache:
  - 1 MB
  - 64 byte blocks
  - 4-way set associative
  - 16 cycle access time (total, not just miss penalty)

- Memory
  - 200+ cycle access time

Tradeoffs between cache parameters are evaluated experimentally

- General goal is to minimize miss rate
- Will look at examples on the following slides
- We will do some cache simulations on the MP’s.

Remember: miss rate is the number of misses per memory access

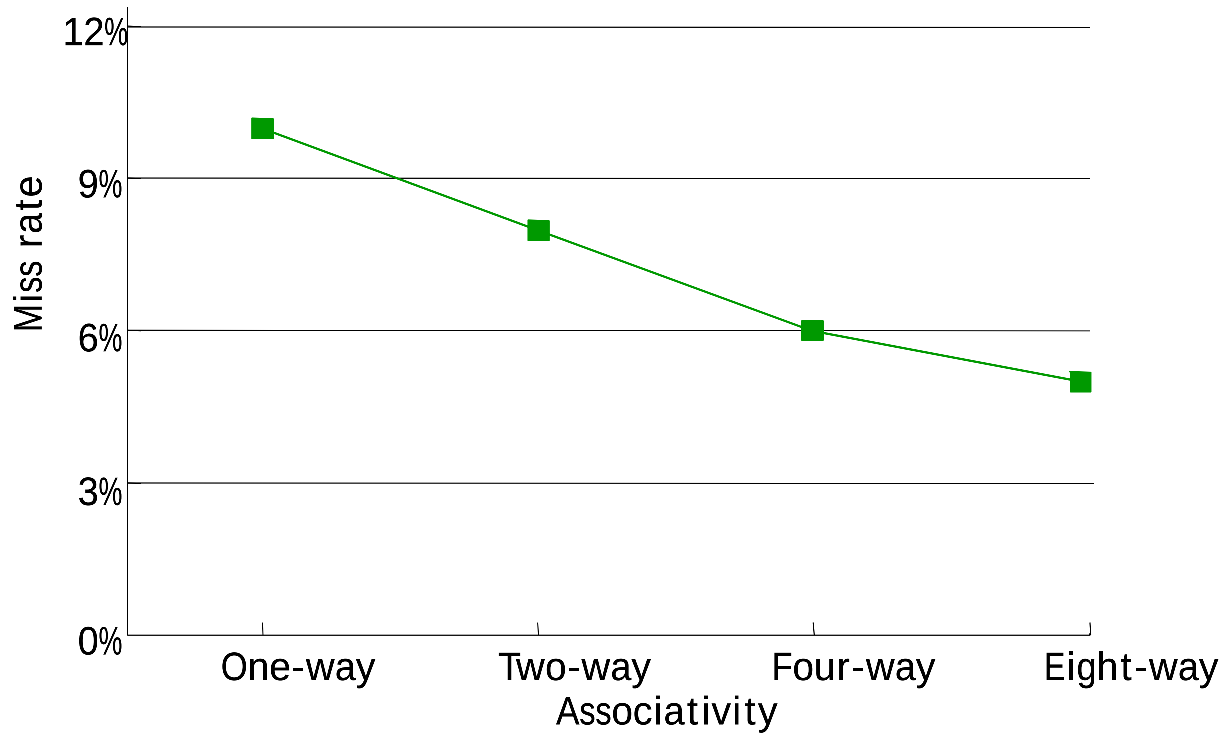
Predict associativity vs. miss rate

Increasing associativity will

a) Decrease the miss rate
b) Increase the miss rate
c) Have mixed results

Increasing associativity generally improves miss rates at the cost of hardware complexity

- Each set has more blocks, so there's less chance of a conflict between two addresses which both belong in the same set.



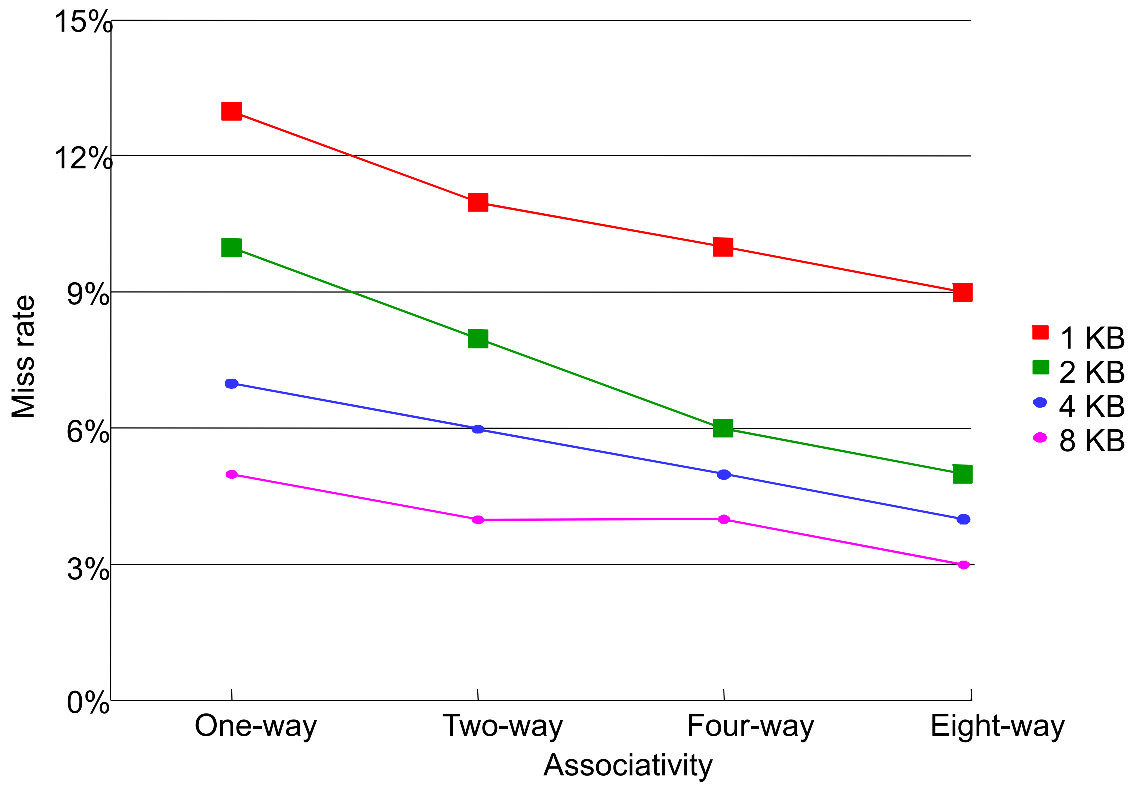
Predict cache size vs. miss rate

Increasing cache size will

a) Decrease the miss rate
b) Increase the miss rate
c) Have mixed results

Increasing cache size also generally improves miss rates

- The larger a cache is, the lower chance of conflict.



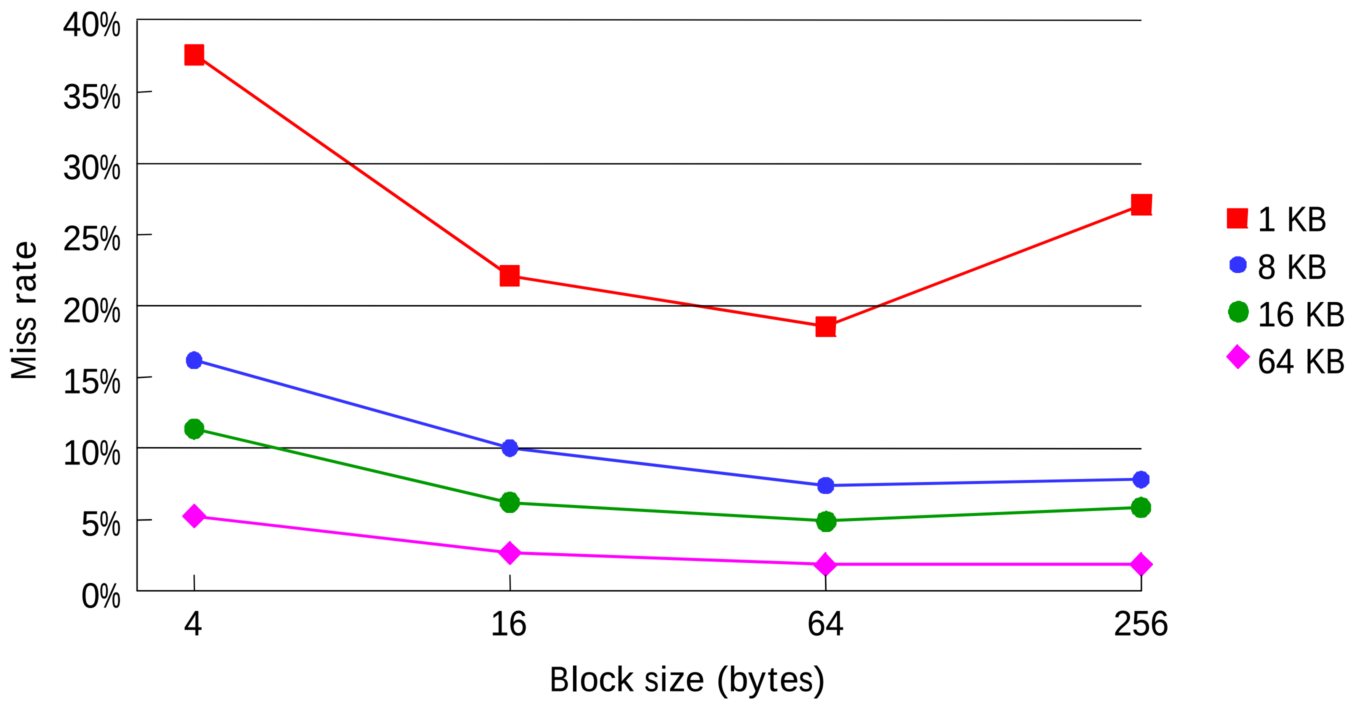
Predict block size vs. miss rate

Increasing block size will

a) Decrease the miss rate
b) Increase the miss rate
c) Have mixed results

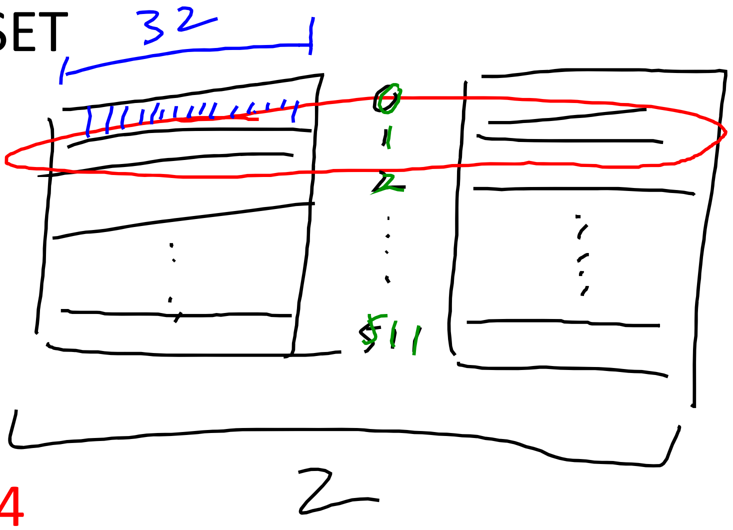
Small blocks do not take advantage of spatial locality

- But if cache blocks get too big relative to the cache size, they increase conflicts



Cache Size Relationships

- NUM_BLOCKS = NUM_SETS * NUM_BLOCKS/SET
- CACHE_SIZE = $\underline{\text{NUM\_BLOCKS}}$ * BLOCK_SIZE



- Example:
  - 2-way set associative, 512 sets, 32B blocks
  - NUM_BLOCKS = a) 32 b) 64 c) 512 d) 1024
  - CACHE_SIZE =
    a) 32B
    b) 512B
    c) 1KB
    d) 16KB
    e) 32KB

K =1024

Code -> Address stream (Example)

```
int A[SIZE], total = 0;
for (int i = 0 ; i < SIZE ; i ++) {
      total += A[i];                lw  A[i]
}
```

total = total + A[i]

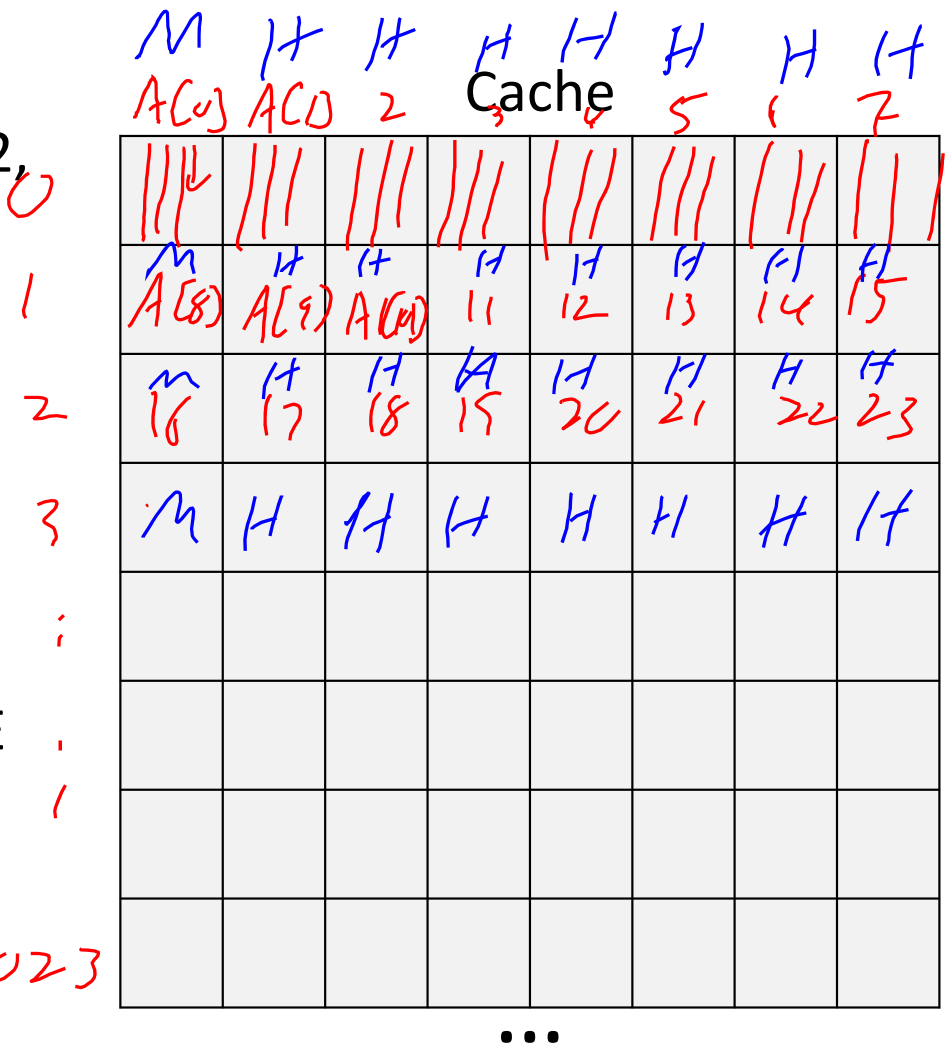
- How many loads/stores are there in this piece of code?
- What is the stream of addresses generated?

assume A = & A[0]

A, A+4, A+8, A+12 ...

We can estimate the miss rate based on the address stream

- Address stream: A, A+4, A+8, A+12, A+16, A+20, A+24, A+28, A+32, A+36, A+40, A+44, A+48, A+52, A+56, A+60, A+64, A+68, …
- Direct-mapped cache, 1024 x 32B blocks (total size: 32,768B)
- What is the miss rate if the stream is smaller than the cache (i.e., SIZE is smaller than 1032 x 32)?



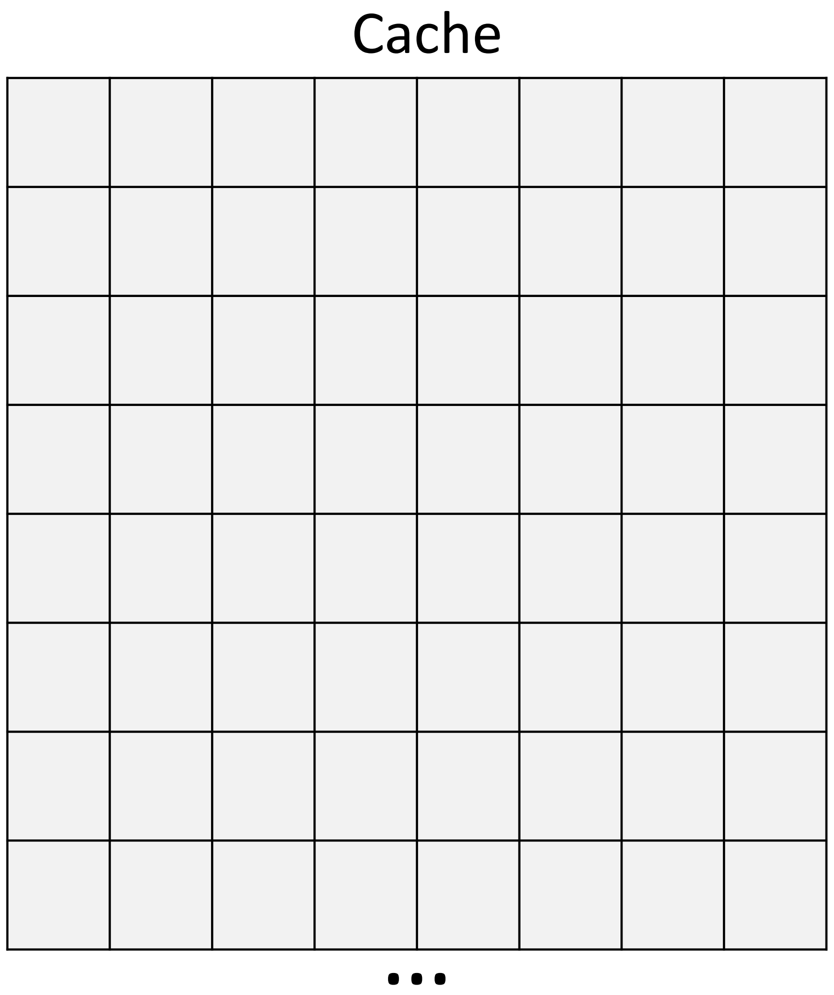
1 miss / 8 accesses

1/8 miss rate

i>clicker question

- Address stream: A, A+4, A+8, A+12, A+16, A+20, A+24, A+28, A+32, A+36, A+40, A+44, A+48, A+52, A+56, A+60, A+64, A+68, …
- Direct-mapped cache, 1024 x 32B blocks (total size: 32,768B)
- What is the miss rate if the stream is greater than the cache?

  a) >1/8 b) 1/8 c) <1/8



Adjacent memory blocks go to adjacent cache sets

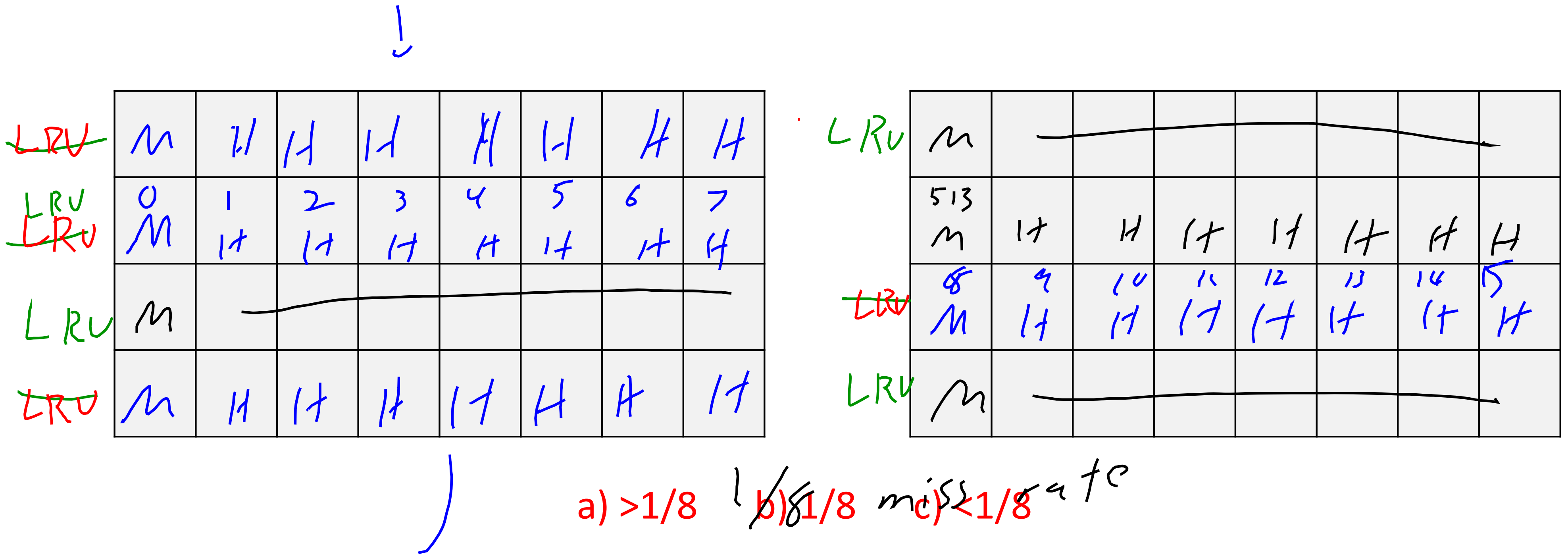
- Assume that A = 0x00010038

```
   A = 0x00010038 = 0001 0000 0000 0011 1000
 A+4 = 0x0001003c = 0001 0000 0000 0011 1100
 A+8 = 0x00010040 = 0001 0000 0000 0100 0000
A+12 = 0x00010044 = 0001 0000 0000 0100 0100
                ...
A+40 = 0x0001005c = 0001 0000 0000 0101 1100
A+44 = 0x00010060 = 0001 0000 0000 0110 0000
```

1024 x 32B Blocks -> 5 bit block offset, 10-bit set index

Estimate the miss rate for a two-way set-associative cache with the same number of blocks

- Address stream: A, A+4, A+8, A+12, A+16, A+20, A+24, A+28, A+32, A+36, A+40, A+44, A+48, A+52, A+56, A+60, A+64, A+68, …
- Cache: Two-way set associative, 512 sets, 32B blocks (same size)



a) >1/8 b) 1/8 c) <1/8

1/8 miss rate

Stride length is the amount we increment over an array

```
int A[SIZE], total = 0;
for (int i = 0 ; i < SIZE ; i += STRIDE) {
                                       // STRIDE != 1
     total += A[i];
}
```

- If STRIDE = 2, what is the stream of addresses generated?

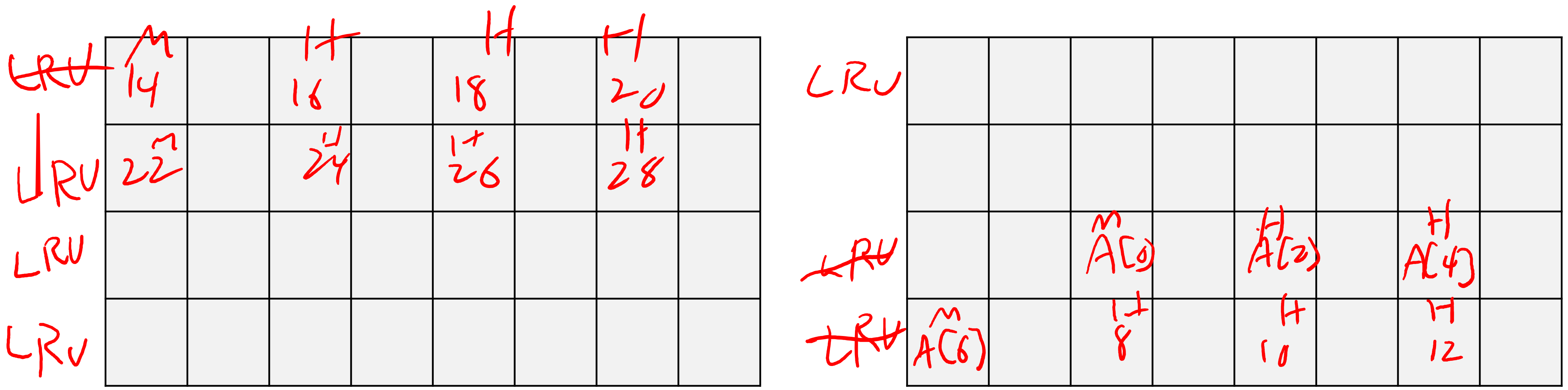
A, A+8, A+16, A+24

What is the miss rate for a stride length of 2?

```
int A[SIZE], total = 0;                     2
for (int i = 0 ; i < SIZE ; i += ~~STRIDE~~) { // STRIDE ~~= 1~~
     total += A[i];                                    = 2
}
```

Cache: 2-way SA, 512 sets, 32B blocks

a) 1/32 b) 1/16 c) 1/8 d) 1/4 e) Not enough info

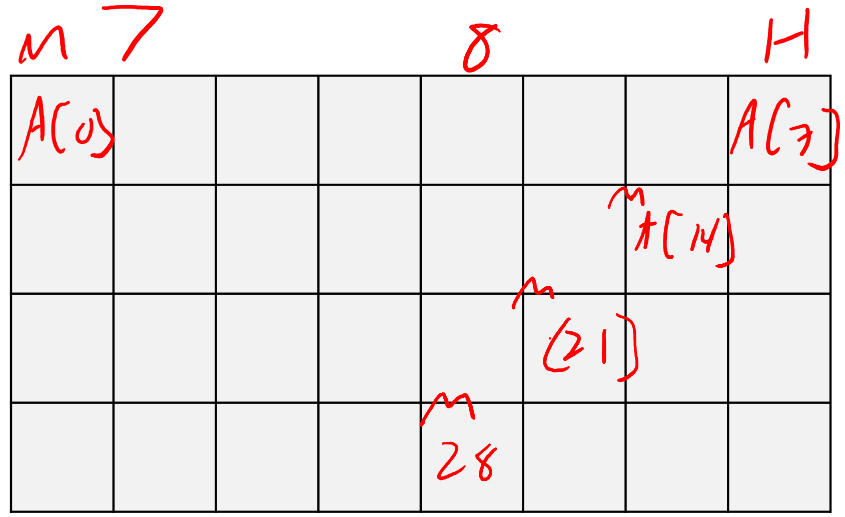


Strided access generally increases the miss rate because it reduces spatial locality

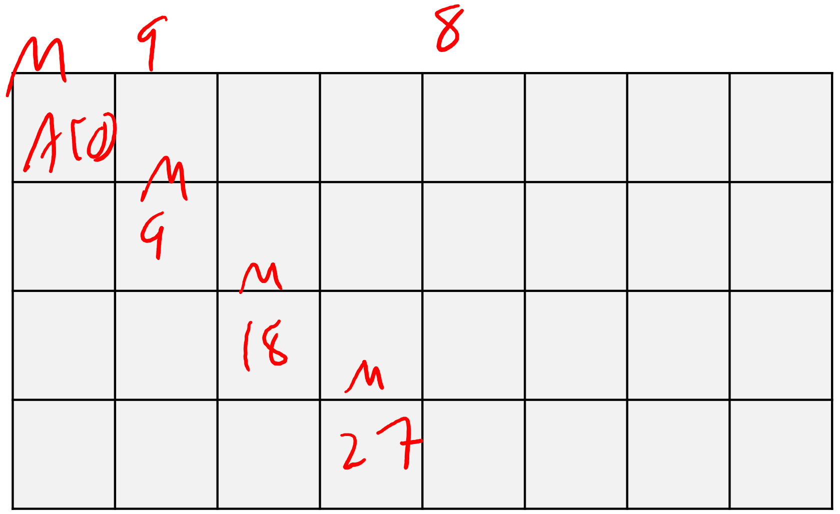
```
int A[SIZE], total = 0;
for (int i = 0 ; i < SIZE ; i += STRIDE) { // STRIDE != 1
      total += A[i];
}
```

In general, two scenarios:

STRIDE < CACHE BLOCK SIZE



STRIDE >= CACHE BLOCK SIZE



The miss rate of strided access can be expressed mathematically

- Miss Rate = MIN(1, DATA_SIZE*STRIDE / CACHE_BLOCK_SIZE)

Nested loops create temporal locality in data access

```
int A[SIZE], total = 0;
for (int j = 0 ; j < N ; j ++) {
    for (int i = 0 ; i < SIZE ; i ++) {
        total += A[i];
    }
}
```

What does stream of addresses look like?

A, A+4, A+8, A+12 . . . . . .

A, A+4, A+8

N times

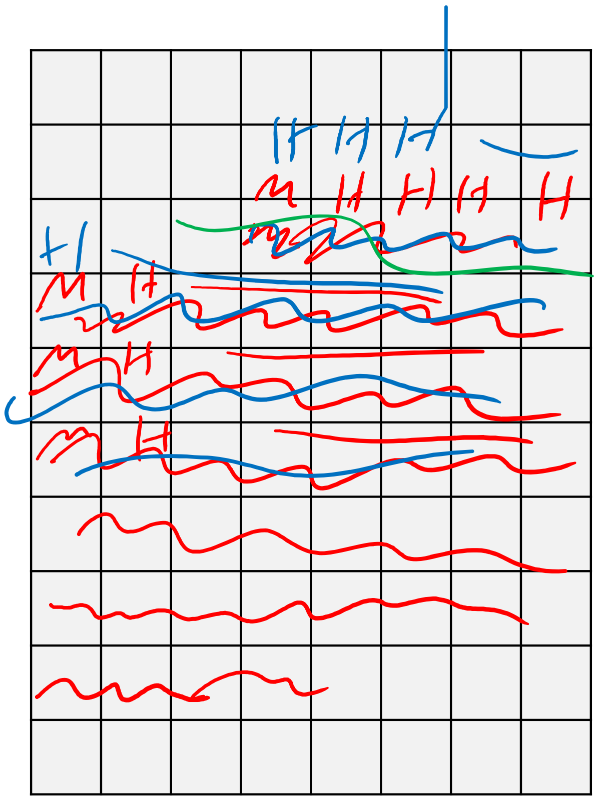
If sizeof(int)*SIZE < CACHE_SIZE and N is large, what is the miss rate?

a) 1/32 b) 1/2 c) 1 d) 0 e) not enough info

(pick best answer)

iter 1: $\frac{1}{8}$ miss rate

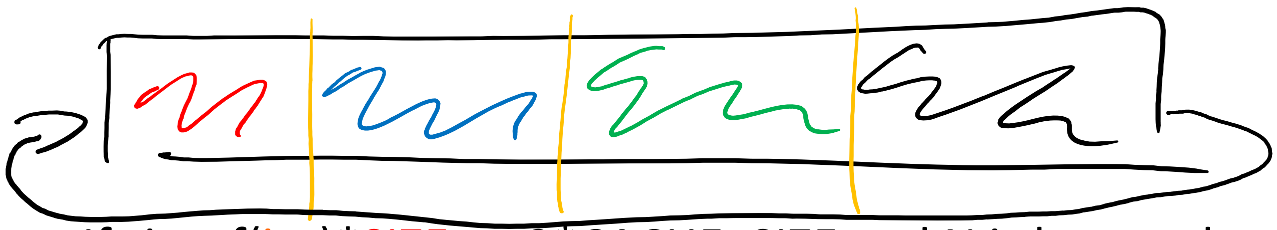
iter 2: 0 miss rate

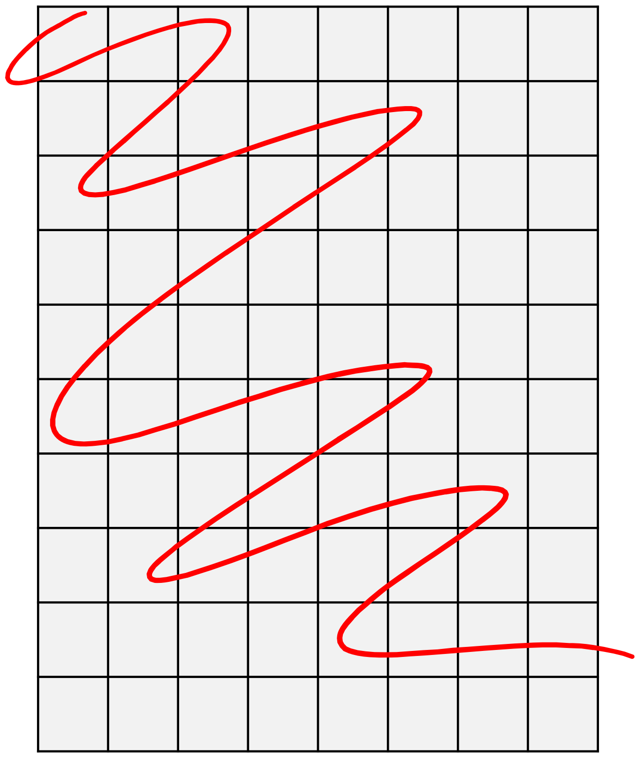


If the data structure is too big, we lose temporal locality

```
int A[SIZE], total = 0;
for (int j = 0 ; j < N ; j ++) {
      for (int i = 0 ; i < SIZE ; i ++) {
            total += A[i];
      }
}
```

What does stream of addresses look like?





If sizeof(int)*SIZE >= 2*CACHE_SIZE and N is large, what is the miss rate?

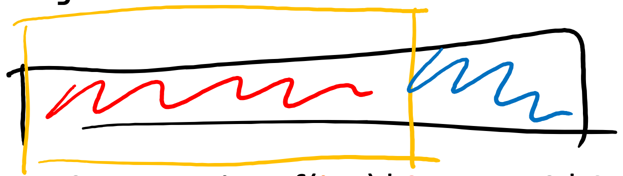
iter 1: $\frac{1}{8}$ miss rate

iter 22: $\frac{1}{8}$ miss rate

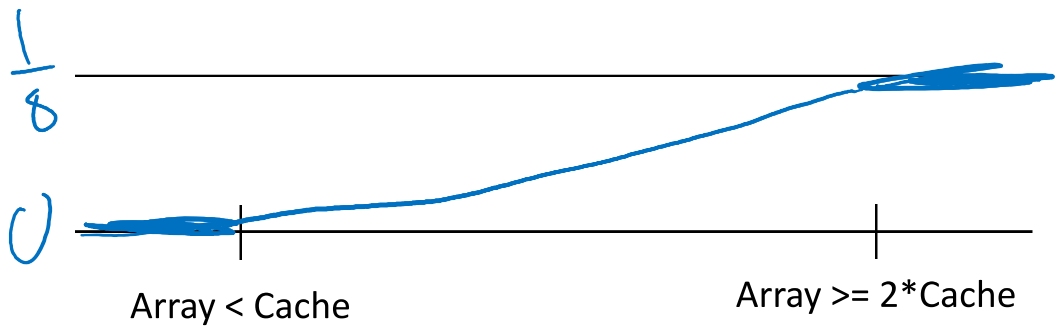
$\sim \frac{1}{8}$ miss rate

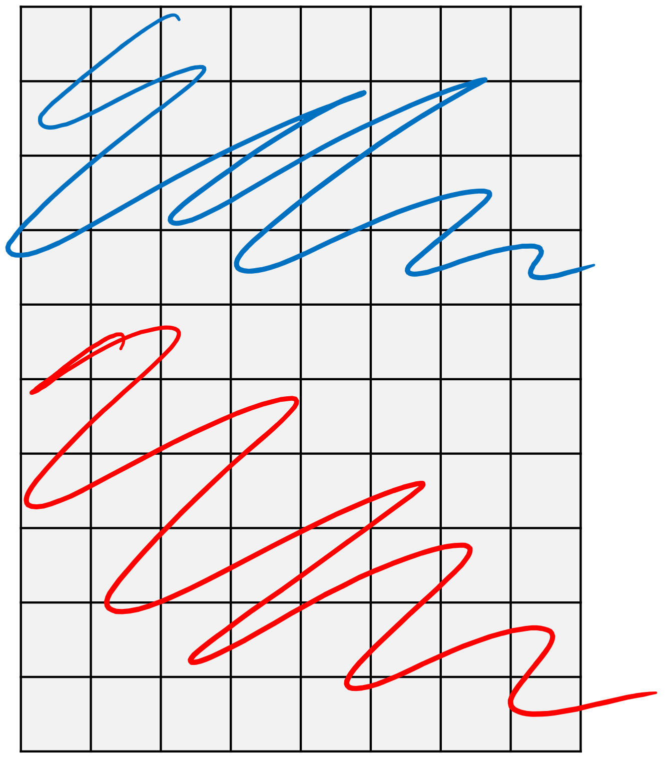
Performance depends on both the cache and data structure properties

```
int A[SIZE], total = 0;
for (int j = 0 ; j < N ; j ++) {
     for (int i = 0 ; i < SIZE ; i ++) {
          total += A[i];
     }
}
```



CACHE_SIZE <= sizeof(int)*SIZE <= 2*CACHE_SIZE





Non-linear access of arrays sacrifices temporal and spatial locality

```
int A[SIZE], total = 0;
for (int j = 0 ; j < 4 ; j ++) {
     for (int i = 0 ; i < SIZE ; i += 4) {
          total += A[i+j];
     }
}
```

What does stream of addresses look like?

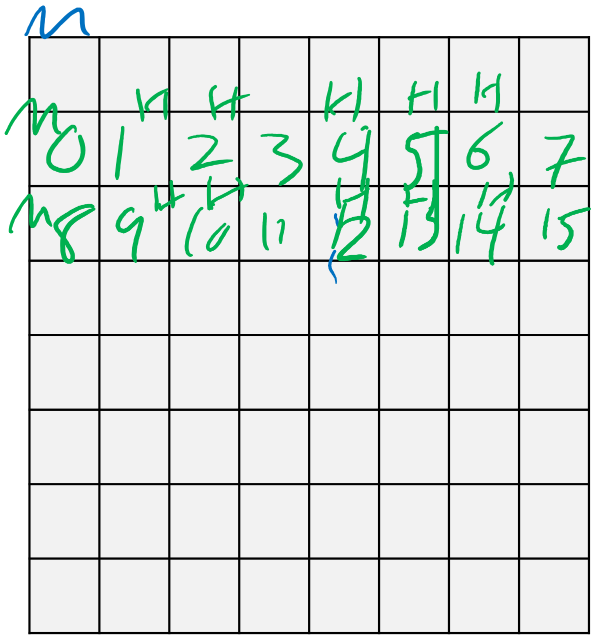
A[0], A[4], A[8], A[12], ...

A[1], A[5], A[9], A[13], ...

What is the miss rate?

$\frac{1}{2}$

$\frac{1}{8}$



General Reuse Questions

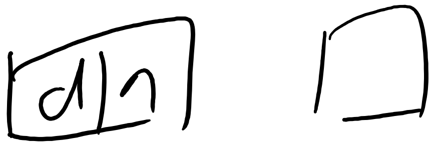
- Is the data still in the cache when reuse occurs?

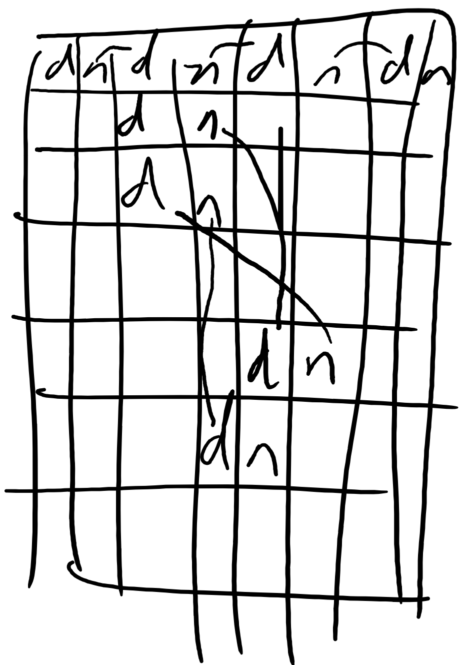
Temporal Locality

- Special case: Does all of the data fit in the cache?

i>clicker, how do linked lists perform on a cache?

```
struct list_t { int data, struct list_t *next };
for (struct list_t *l = list_head ; l != NULL ; l = l->next) {
    total += l->data;
}
```



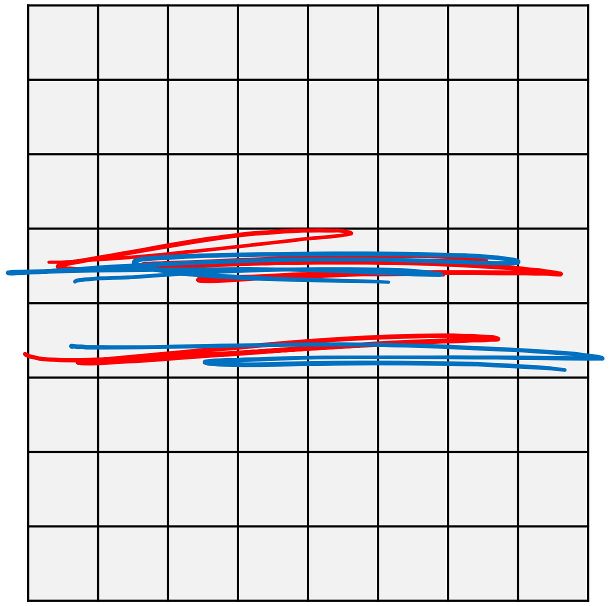


Assume a 2-way SA cache with 512 sets, 32B blocks.

If the list is 1000 elements long, how many misses per iteration of the loop?

a) 1 b) ¼ c) 1/8 d) 0 e) not enough info

i>clicker Multiple Data Structures



```
int A[SIZE], B[SIZE];
for (int i = 0 ; i < SIZE ; i ++) {
    B[i] = A[i];
}
```

What is the stream of addresses generated?

Assume a direct-mapped cache with 1024 32B blocks.

Assume sizeof(int)*SIZE >> CACHE_SIZE.

How many cache misses per iteration of the loop?

a) 2 b) 1 c) 1/4 d) 1/8 e) not enough info

What addresses map to the same set?

- Multiples of the set size.

- SET_SIZE = NUM_SETS * BLOCK_SIZE
  - Also SET_SIZE = CACHE_SIZE / NUM_BLOCKS/SET

- Examples:
  - Direct-mapped, 1024 32B blocks

  - 2-way SA, 512 sets, 32B blocks

Cache Analysis Questions

- How big is each data item? (how many fit in a cache block?)
- Is there data reuse or use of data in the same cache block?
- Is the data/block still in cache when reuse occurs?
  - **How big is the data relative to the cache?**
  - **Is there aliasing (cache conflicts) that is problematic?**

- Some useful relationships:

  Strides: miss rate = MIN(1, sizeof(data) * stride/BLOCK_SIZE)

  Adjacent blocks in memory -> adjacent sets

  Addresses A, A+BLOCK_SIZE*NUM_SETS are in the same set