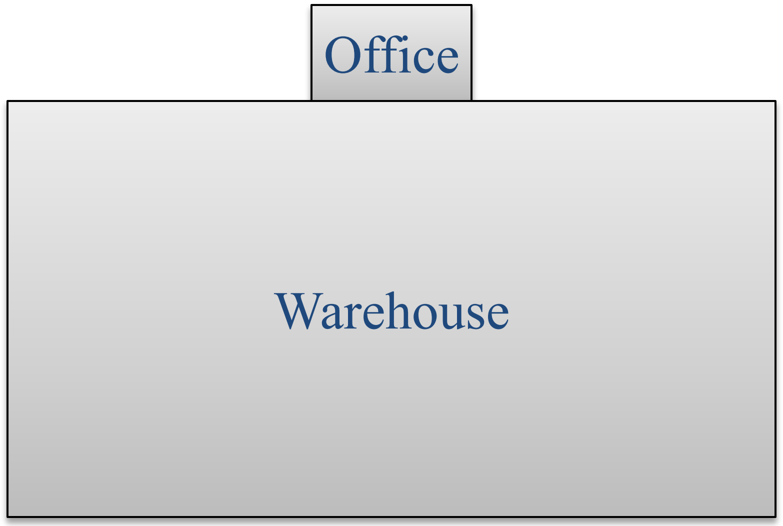
Lesson Exercises

LINKÖPING UNIVERSITY

Question 1

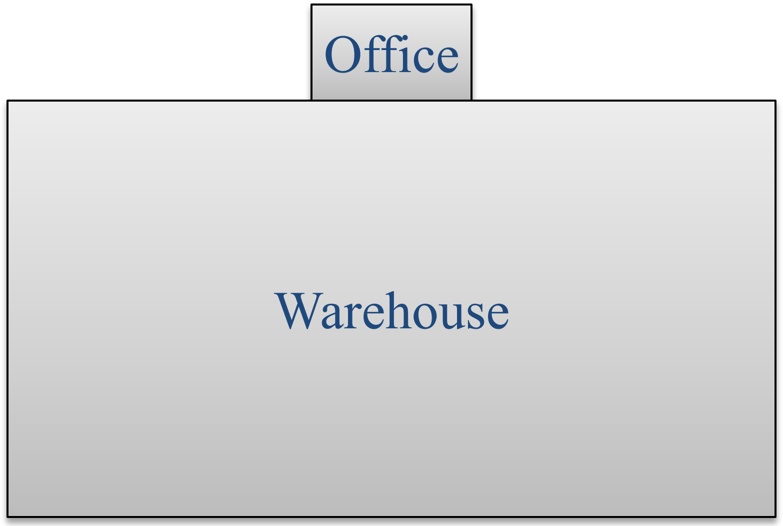
- You have a huge warehouse with EVERY movie ever made (hits, training films, etc.).
- Getting a movie from the warehouse takes 15 minutes.
- You can’t stay in business if every rental takes 15 minutes.
- You have some small shelves in the front office.



Here are some suggested improvements to the store:

1. Whenever someone rents a movie, just keep it in the front office for a while in case someone else wants to rent it.
2. Watch the trends in movie watching and attempt to guess movies that will be rented soon – put those in the front office.
3. Whenever someone rents a movie in a series (Star Wars), grab the other movies in the series and put them in the front office.
4. Buy motorcycles to ride in the warehouse to get the movies faster

Extending the analogy to locality for caches, which pair of changes most closely matches the analogous cache locality?



Question 1

Principle of Locality

Program instructions access a small proportion of their address space at any time

Question 1

Principle of Locality

$\underbrace{\text{Program instructions}}_{\text{Customers}}$ access a $\underbrace{\text{small proportion}}_{\text{Individual movie}}$ of their $\underbrace{\text{address space}}_{\text{Movie collection}}$ at any time

Question 1

Principle of Locality

$\underbrace{\text{Program instructions}}_{\text{Customers}}$ access a $\underbrace{\text{small proportion}}_{\text{Individual movie}}$ of their $\underbrace{\text{address space}}_{\text{Movie collection}}$ at any time

- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon

Question 1

Principle of Locality

$\underbrace{\text{Program instructions}}_{\text{Customers}}$ access a $\underbrace{\text{small proportion}}_{\text{Individual movie}}$ of their $\underbrace{\text{address space}}_{\text{Movie collection}}$ at any time

- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon

- **Spatial locality**
  - Items near those accessed recently
    are likely to be accessed soon

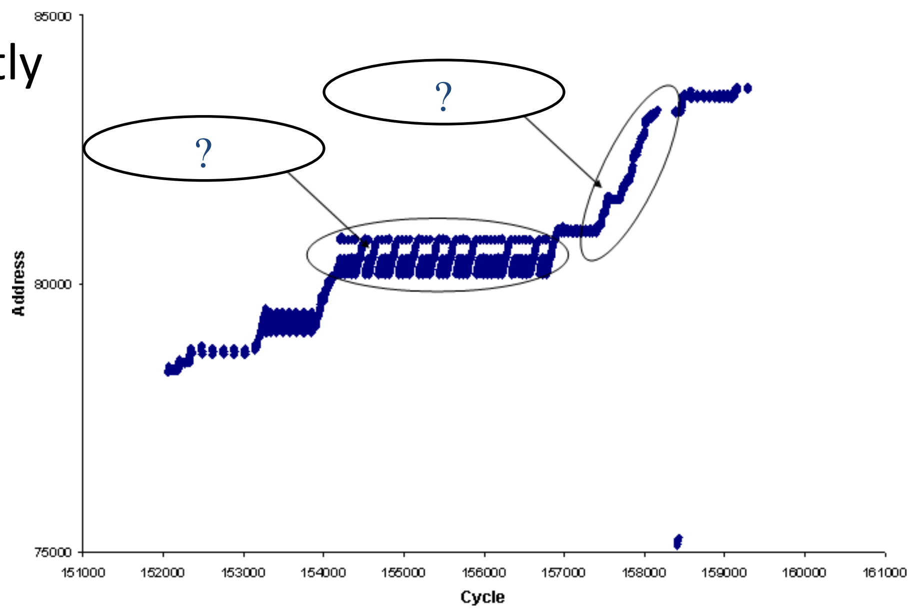
Question 1

Principle of Locality

$\underbrace{\text{Program instructions}}_{\text{Customers}}$ access a $\underbrace{\text{small proportion}}_{\text{Individual movie}}$ of their $\underbrace{\text{address space}}_{\text{Movie collection}}$ at any time

- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon

- **Spatial locality**
  - Items near those accessed recently are likely to be accessed soon



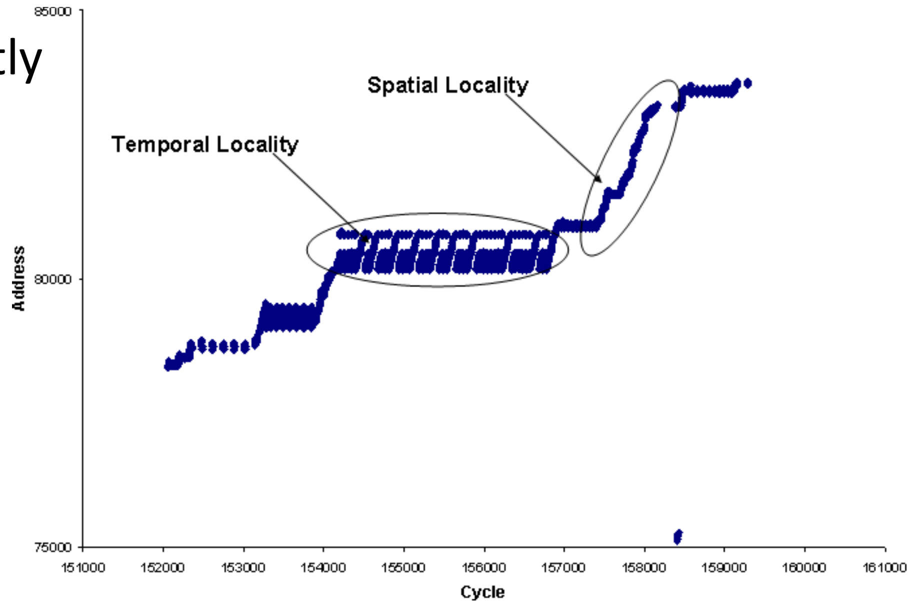
Question 1

Principle of Locality

$\underbrace{\text{Program instructions}}_{\text{Customers}}$ access a $\underbrace{\text{small proportion}}_{\text{Individual movie}}$ of their $\underbrace{\text{address space}}_{\text{Movie collection}}$ at any time

- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon

- **Spatial locality**
  - Items near those accessed recently are likely to be accessed soon



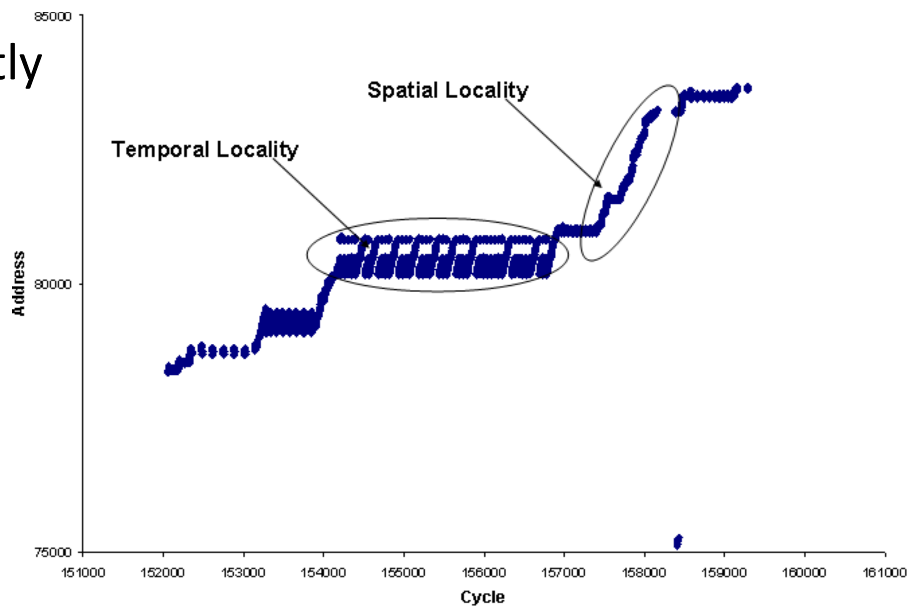
Question 1

Principle of Locality

$\underbrace{\text{Program instructions}}_{\text{Customers}}$ access a $\underbrace{\text{small proportion}}_{\text{Individual movie}}$ of their $\underbrace{\text{address space}}_{\text{Movie collection}}$ at any time

- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon

- **Spatial locality**
  - Items near those accessed recently are likely to be accessed soon



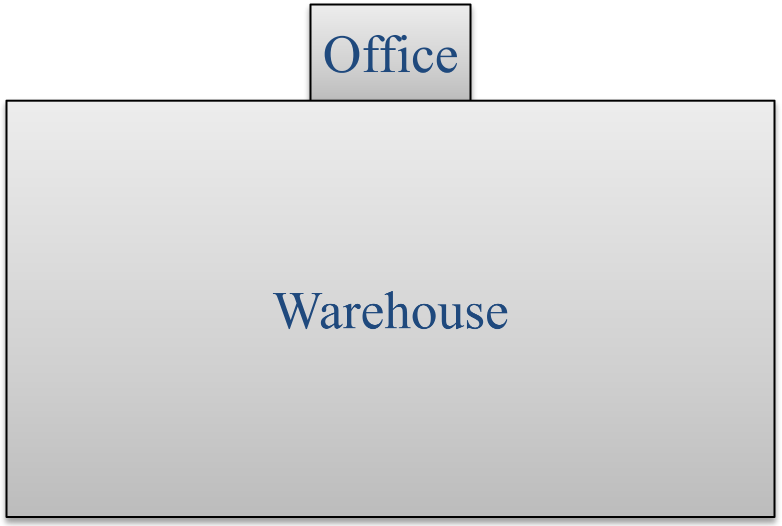
Let's go back to our question

Here are some suggested improvements to the store:

1. Whenever someone rents a movie, just keep it in the front office for a while in case someone else wants to rent it.
2. Watch the trends in movie watching and attempt to guess movies that will be rented soon – put those in the front office.
3. Whenever someone rents a movie in a series (Star Wars), grab the other movies in the series and put them in the front office.
4. Buy motorcycles to ride in the warehouse to get the movies faster

Extending the analogy to locality for caches, which pair of changes most closely matches the analogous cache locality?

Selection	Spatial	Temporal
A	2	1
B	4	2
C	4	3
D	3	1
E	None of the above	

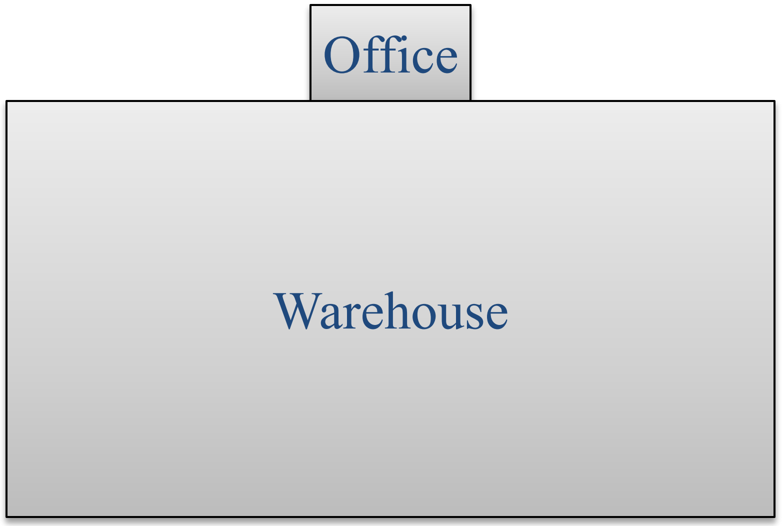


Here are some suggested improvements to the store:

1. Whenever someone rents a movie, just keep it in the front office for a while in case someone else wants to rent it.
2. Watch the trends in movie watching and attempt to guess movies that will be rented soon – put those in the front office.
3. Whenever someone rents a movie in a series (Star Wars), grab the other movies in the series and put them in the front office.
4. Buy motorcycles to ride in the warehouse to get the movies faster

Extending the analogy to locality for caches, which pair of changes most closely matches the analogous cache locality?

Selection	Spatial	Temporal
A	2	1
B	4	2
C	4	3
[✓ D]	3	1
E	None of the above	



Question 2

For the following code, identify the variables that contribute to temporal locality and the variables that contribute to spatial locality. The variables are i, j, and the array A.

```
for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
                A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

Question 2

A is a matrix (two dimensional array)

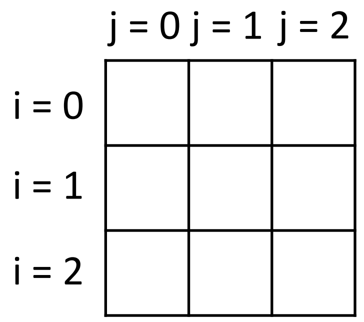
	j = 0	j = 1	j = 2
i = 0			
i = 1			
i = 2			

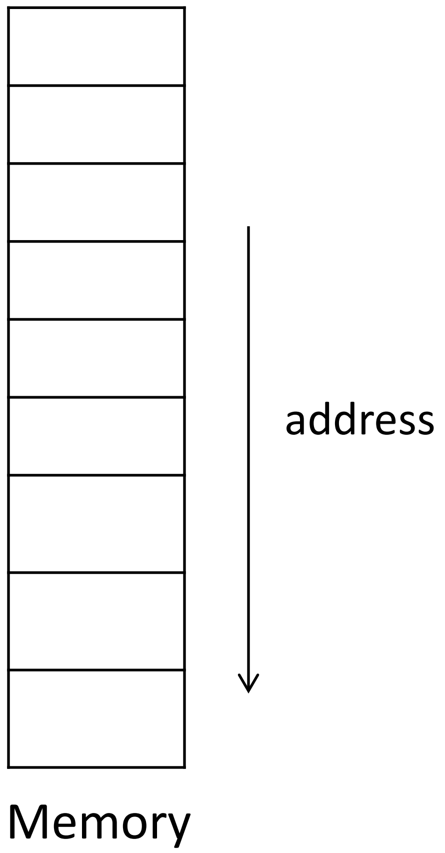
```
for (i = 0; i < 3; i++)
      for (j = 0; j < 3; j++)
            A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

Question 2

A is a matrix (two dimensional array)

How is A stored in the memory?

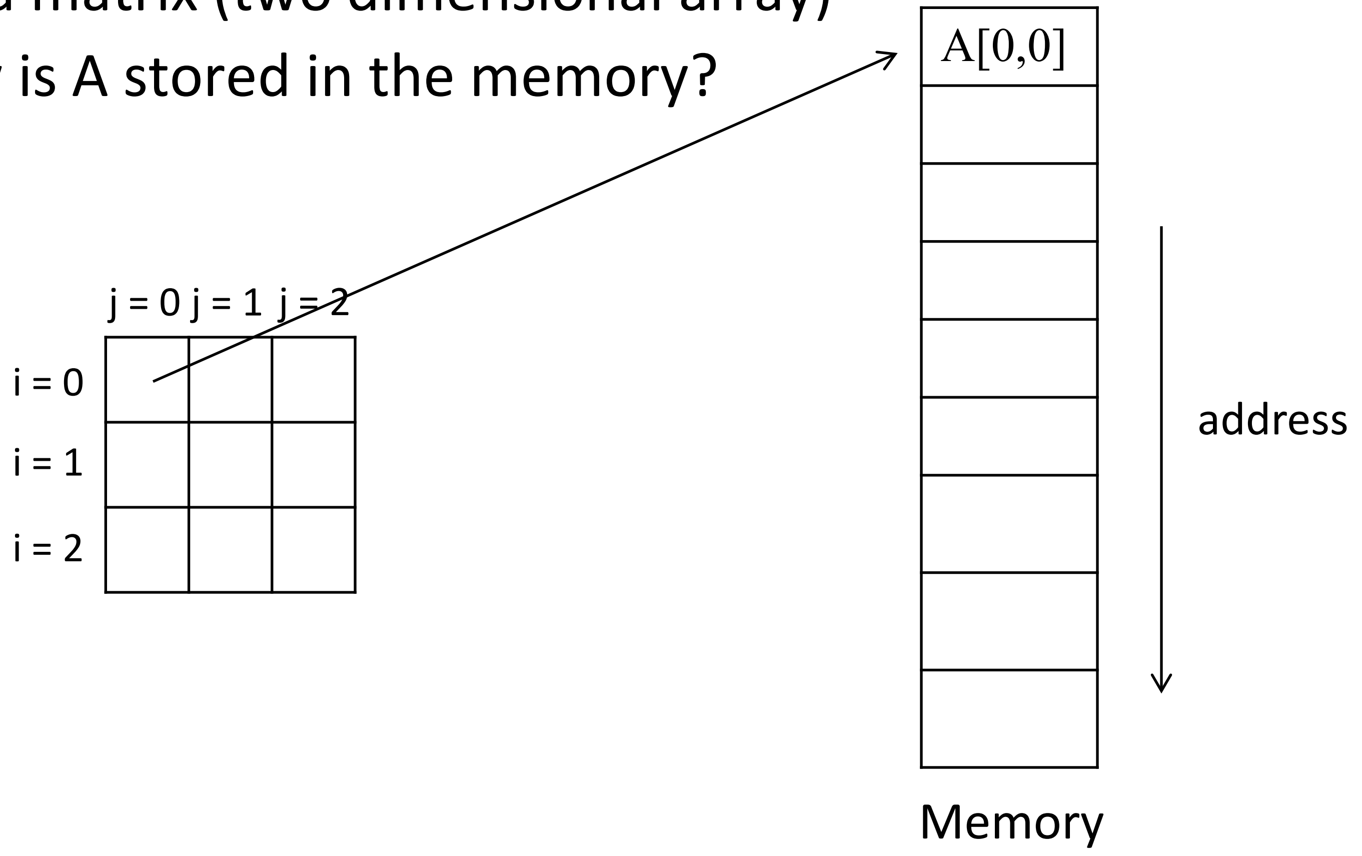




Question 2

A is a matrix (two dimensional array)

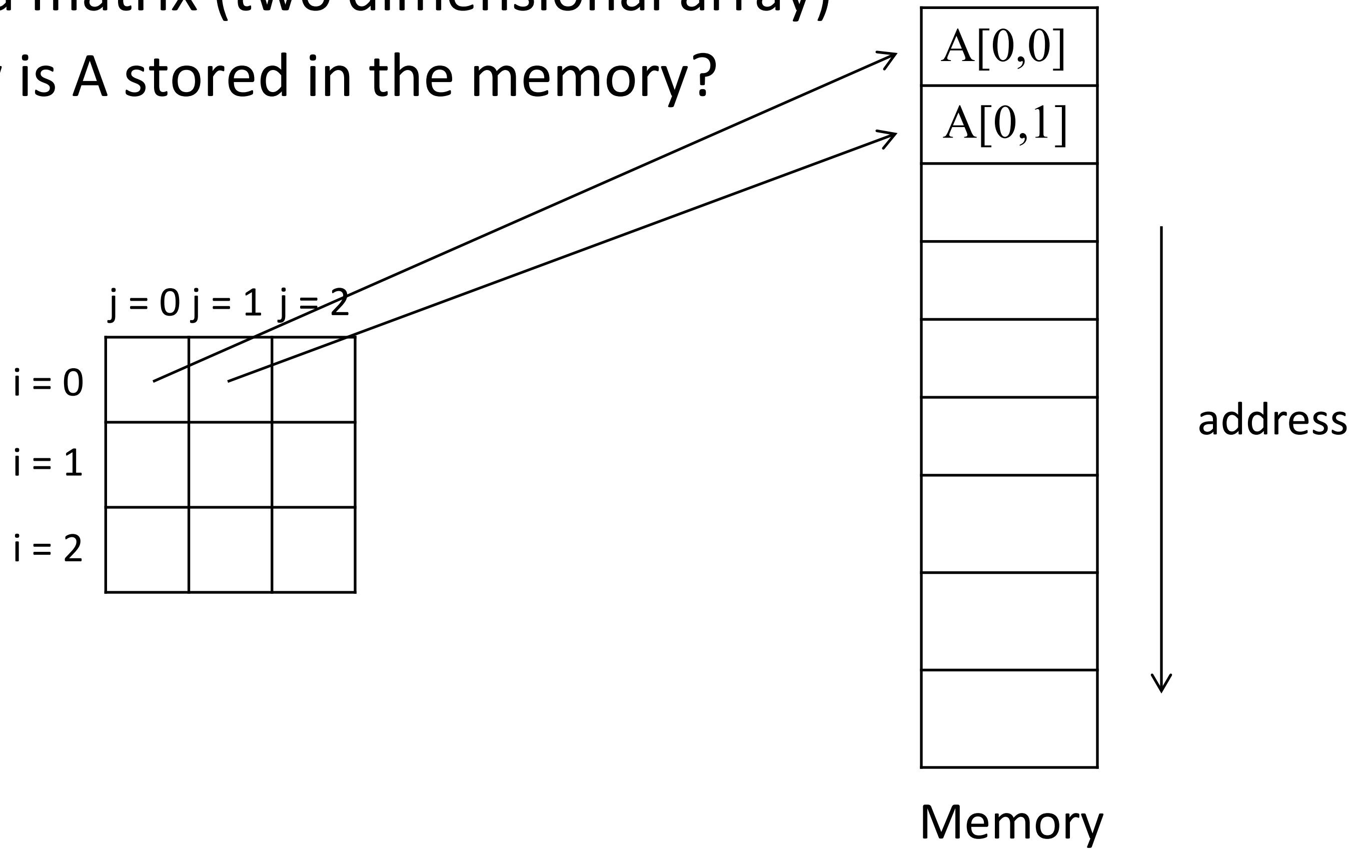
How is A stored in the memory?



Question 2

A is a matrix (two dimensional array)

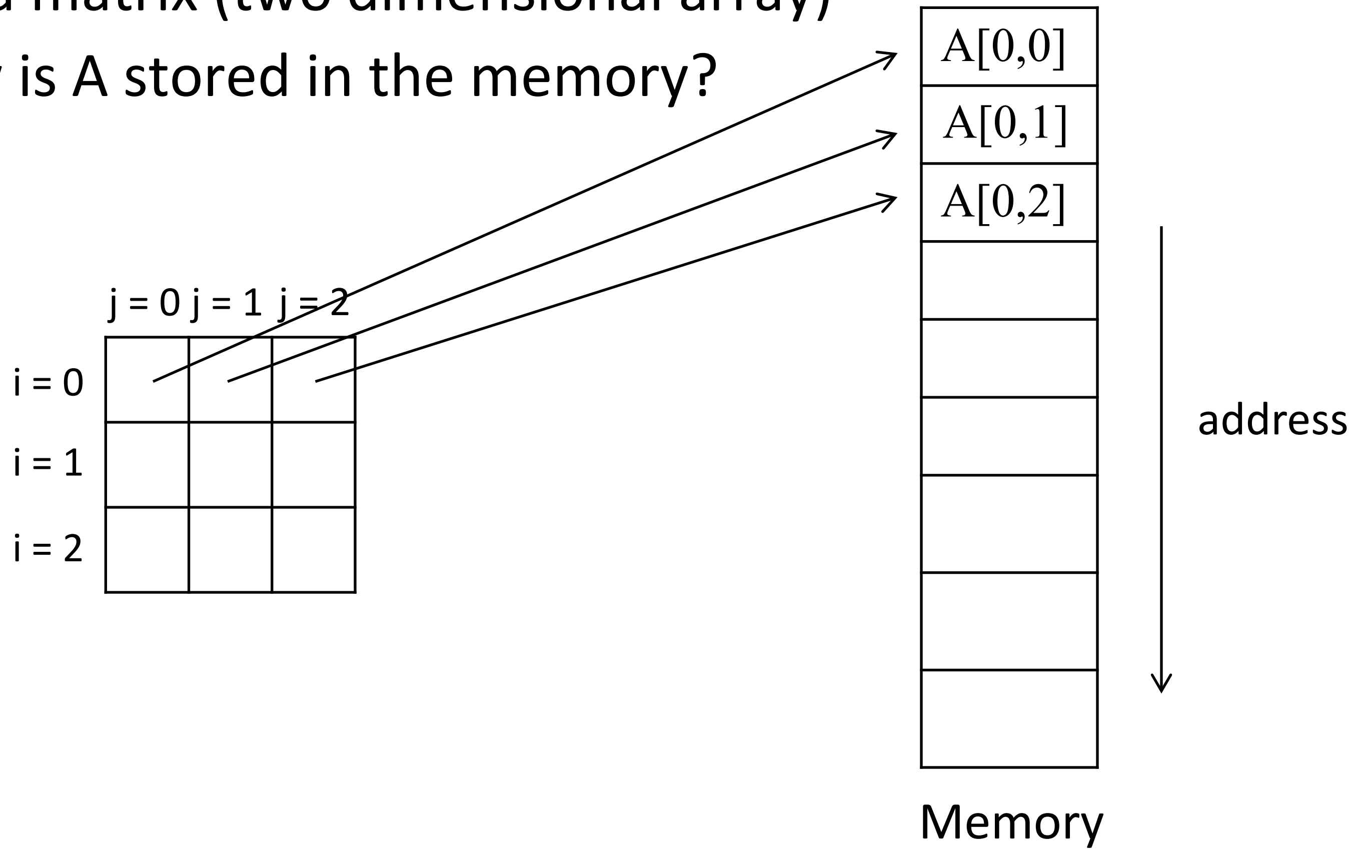
How is A stored in the memory?



Question 2

A is a matrix (two dimensional array)

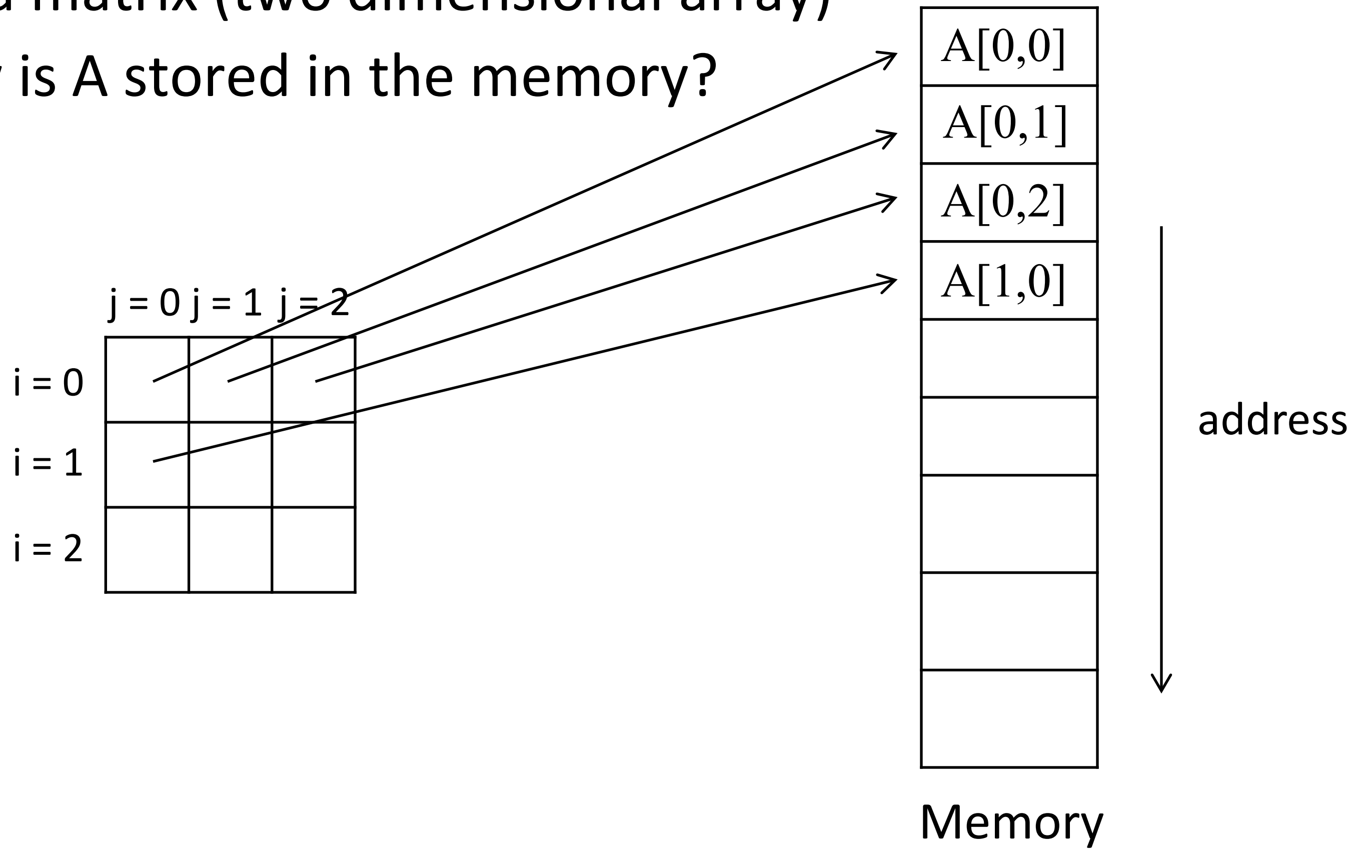
How is A stored in the memory?



Question 2

A is a matrix (two dimensional array)

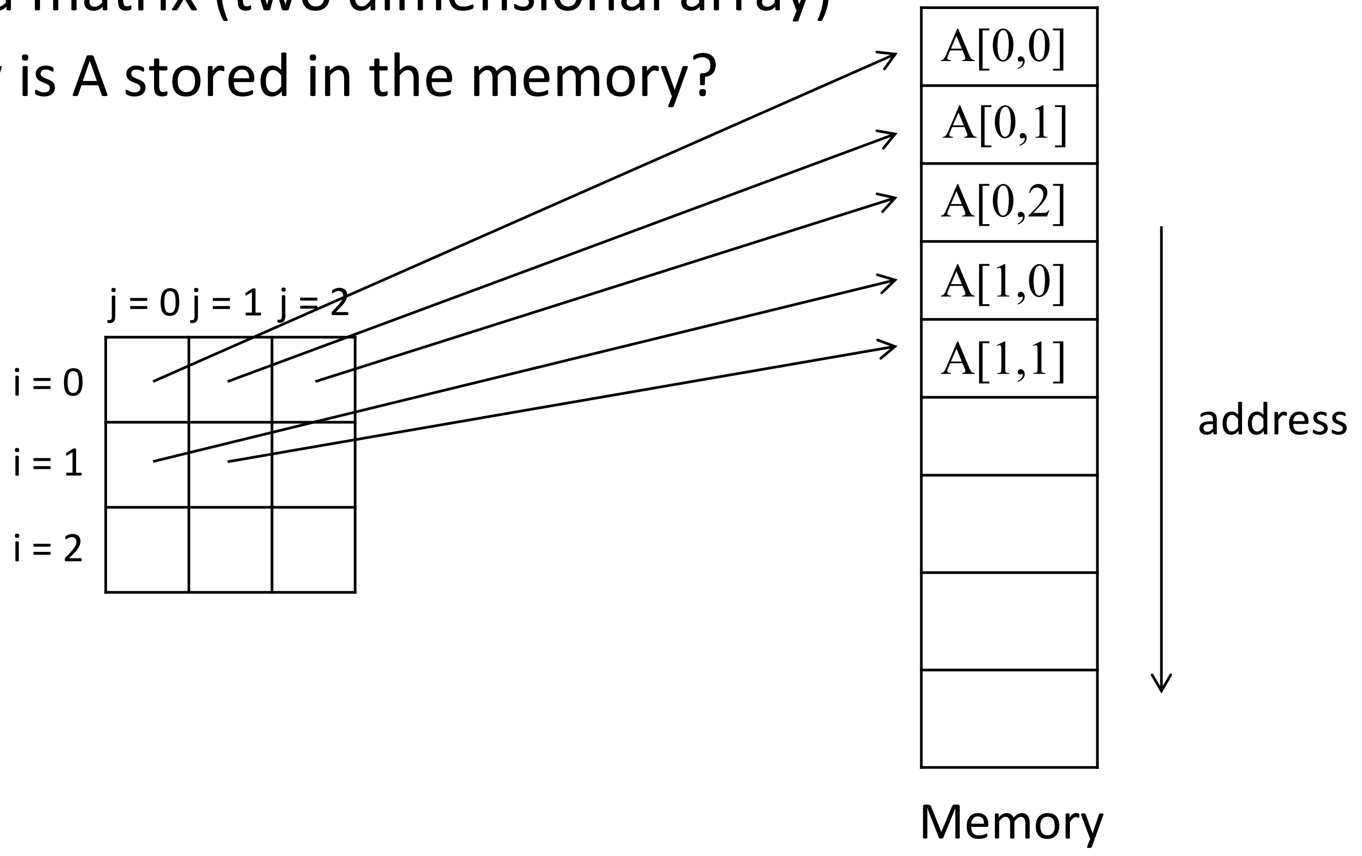
How is A stored in the memory?



Question 2

A is a matrix (two dimensional array)

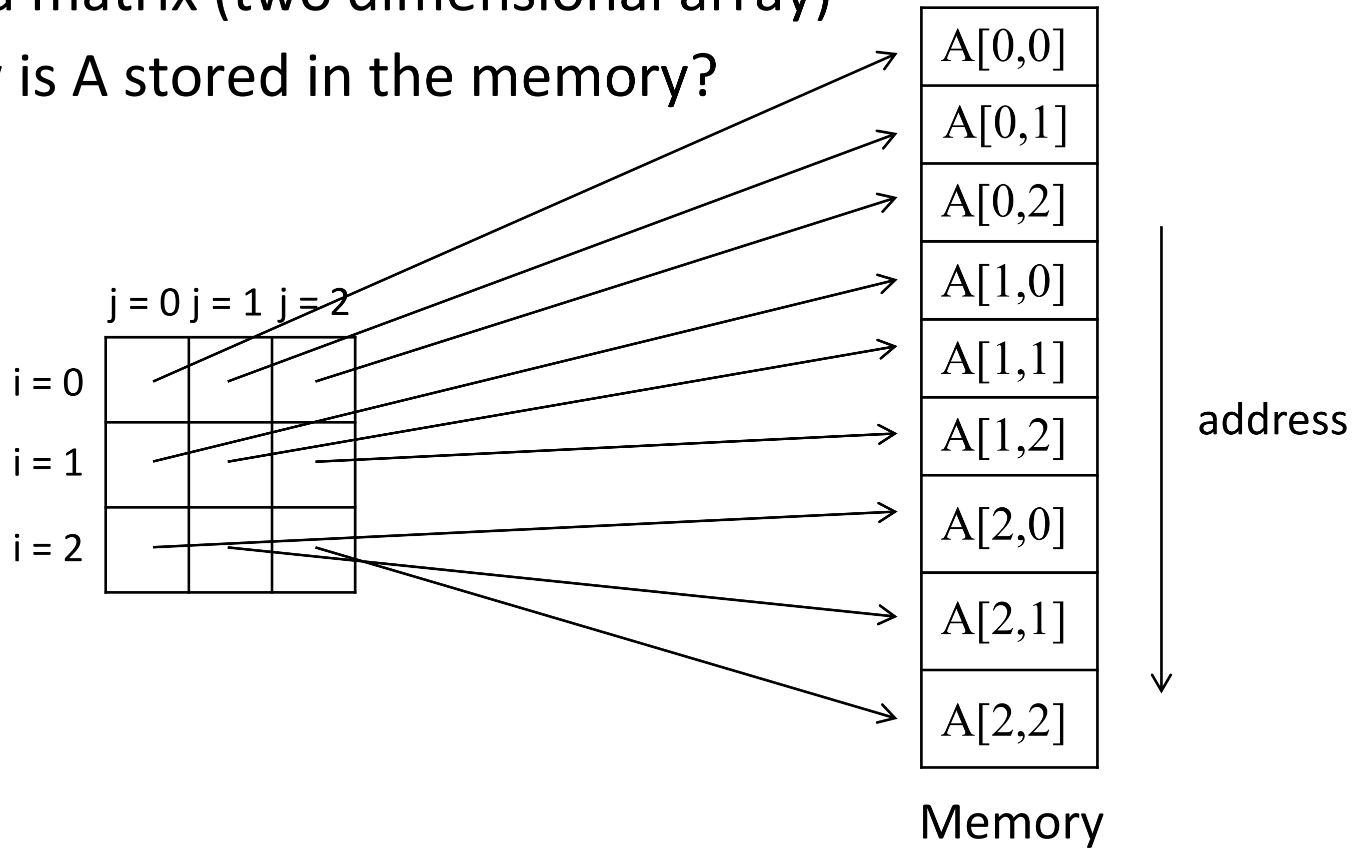
How is A stored in the memory?



Question 2

A is a matrix (two dimensional array)

How is A stored in the memory?



Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

	j = 0	j = 1	j = 2
i = 0			
i = 1			
i = 2			

A[0,0]
A[0,1]
A[0,2]
A[1,0]
A[1,1]
A[1,2]
A[2,0]
A[2,1]
A[2,2]

In each iteration, we access two array slots – A[i][j] and A[j][i]

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

	j = 0	j = 1	j = 2
i = 0			
i = 1			
i = 2			

A[i][j] accesses   A[j][i] accesses

A[0,0]
A[0,1]
A[0,2]
A[1,0]
A[1,1]
A[1,2]
A[2,0]
A[2,1]
A[2,2]

In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 1: i = 0, j = 0

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

	j = 0	j = 1	j = 2
i = 0			
i = 1			
i = 2			

A[i][j] accesses		A[j][i] accesses
*	A[0,0]	
	A[0,1]	
	A[0,2]	
	A[1,0]	
	A[1,1]	
	A[1,2]	
	A[2,0]	
	A[2,1]	
	A[2,2]	

In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 1: i = 0, j = 0

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

	j = 0	j = 1	j = 2
i = 0			
i = 1			
i = 2			

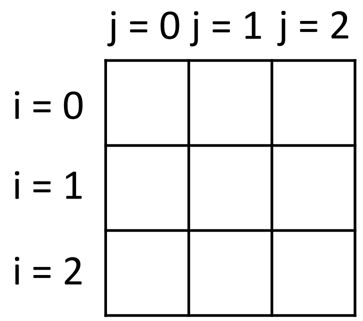
A[i][j] accesses		A[j][i] accesses
*	A[0,0]	*
	A[0,1]	
	A[0,2]	
	A[1,0]	
	A[1,1]	
	A[1,2]	
	A[2,0]	
	A[2,1]	
	A[2,2]	

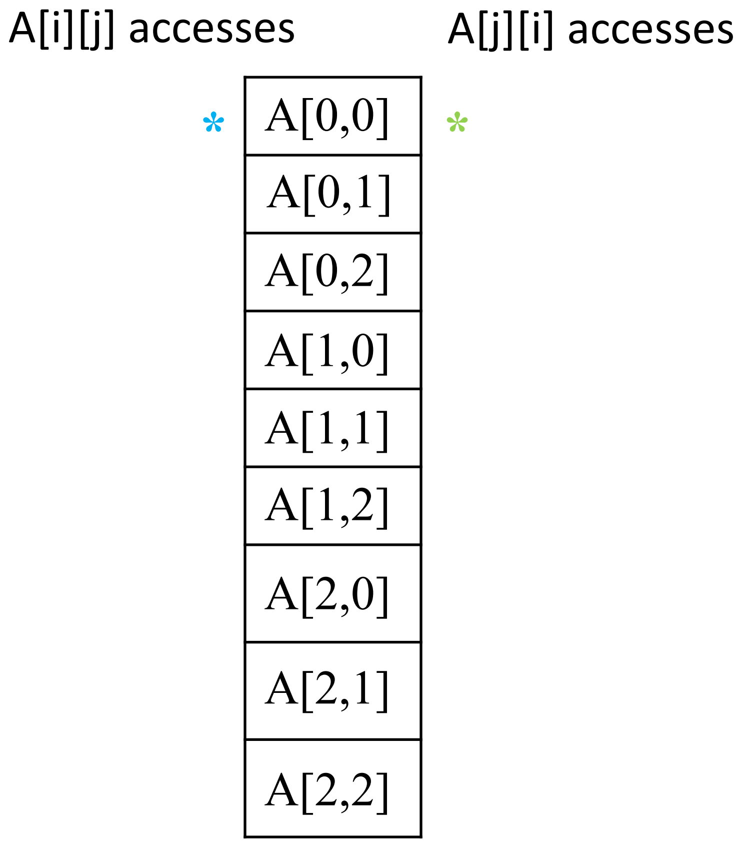
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 1: i = 0, j = 0

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



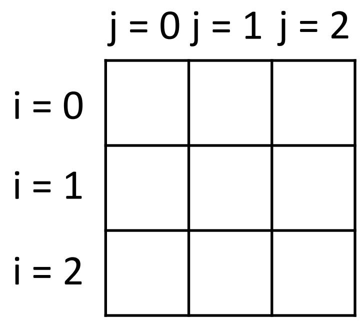


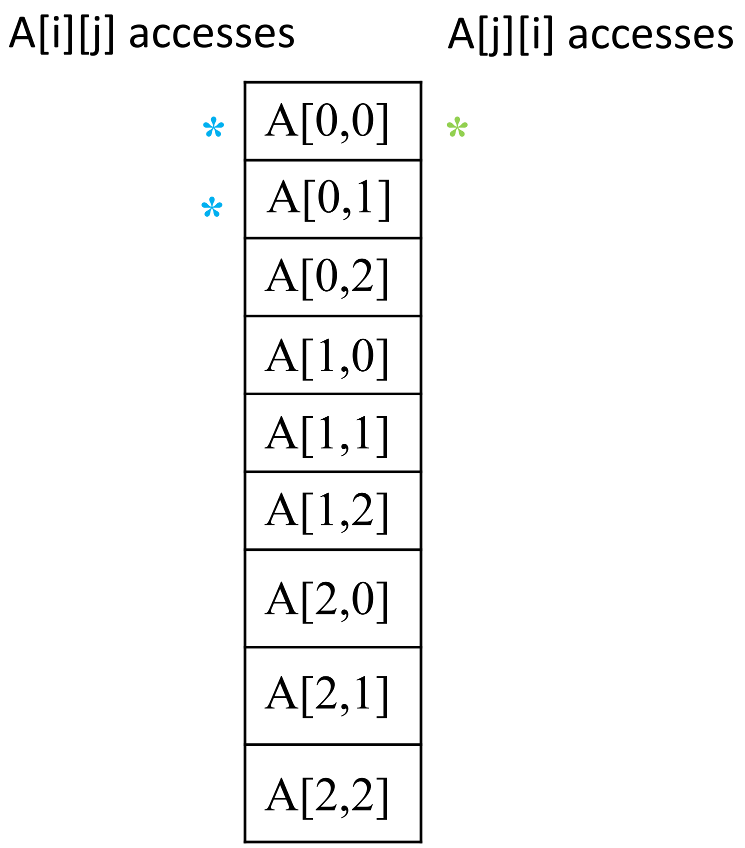
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 2: i = 0, j = 1

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



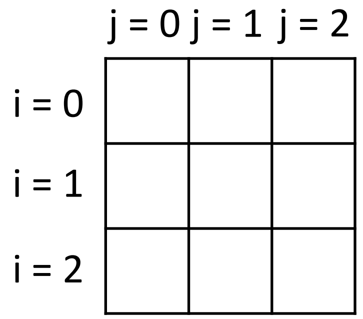


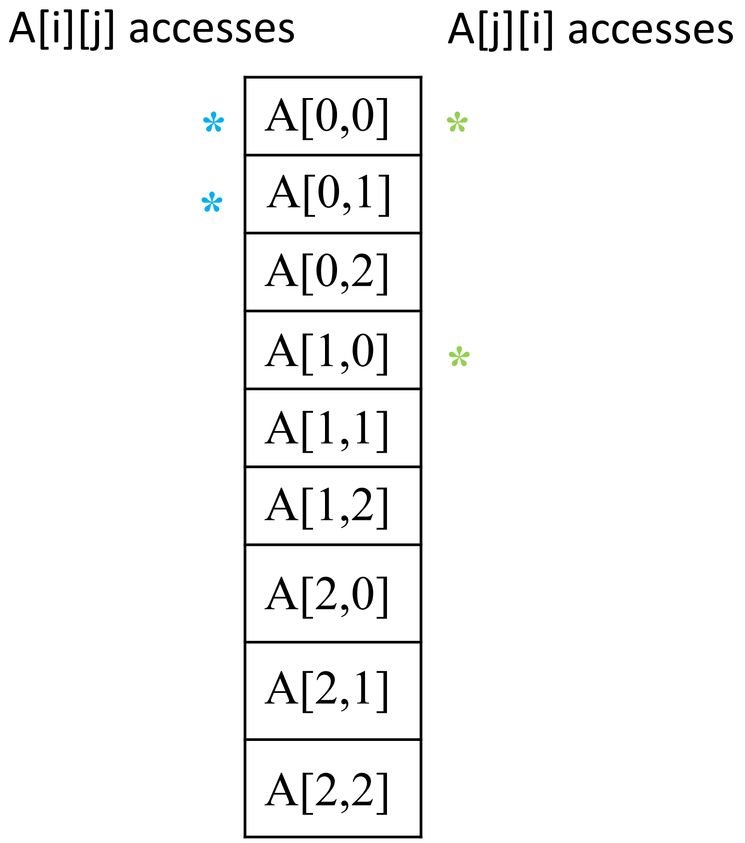
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 2: i = 0, j = 1

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```





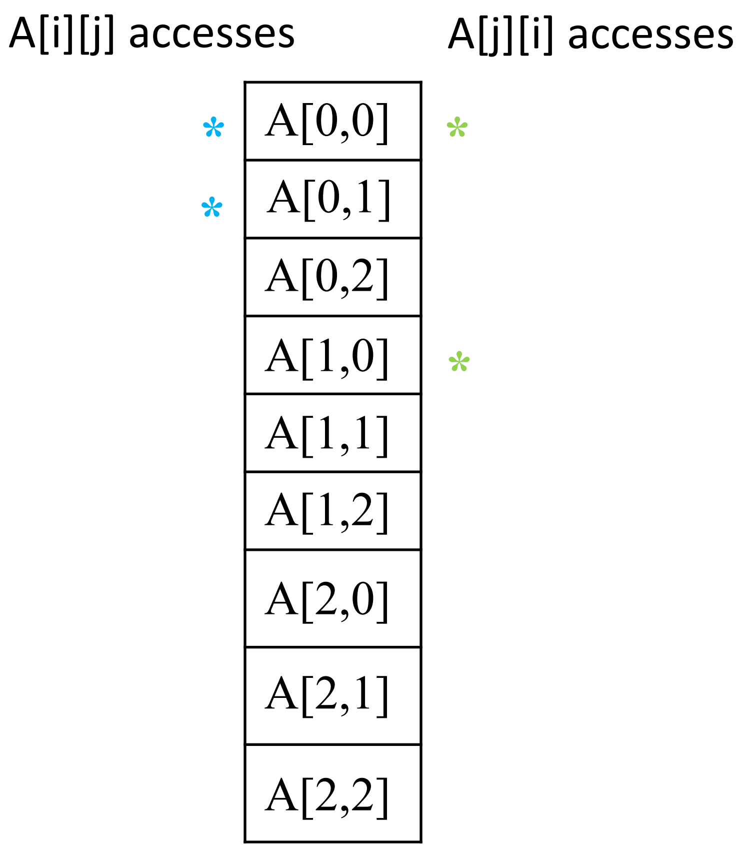
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 2: i = 0, j = 1

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

	j = 0	j = 1	j = 2
i = 0			
i = 1			
i = 2			

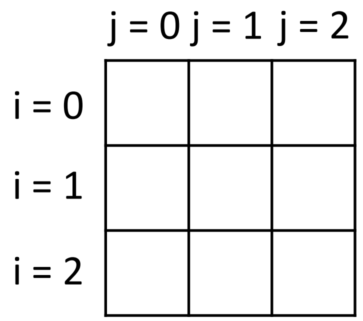


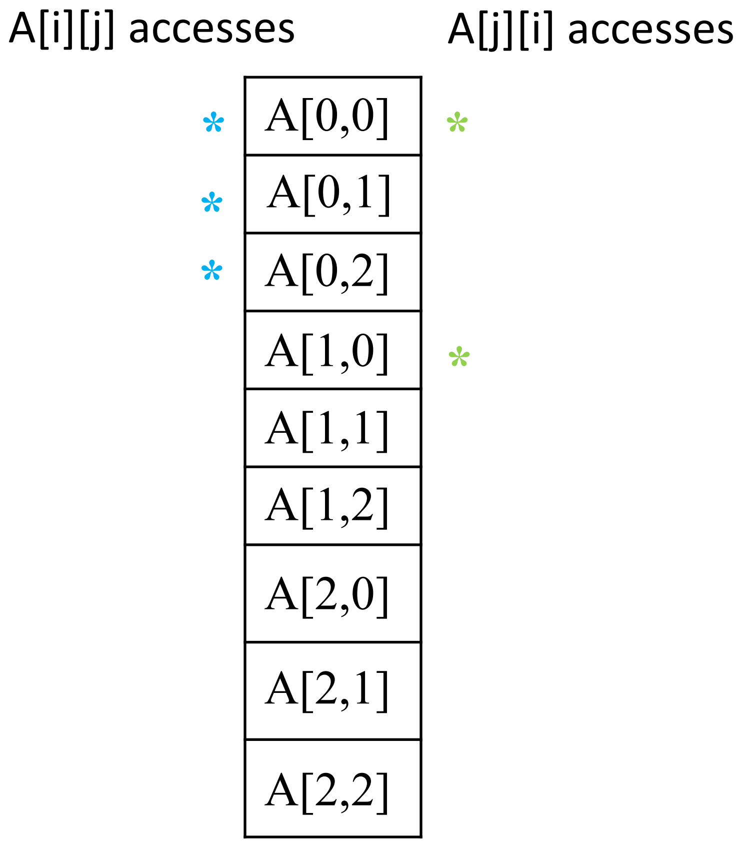
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 3: i = 0, j = 2

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



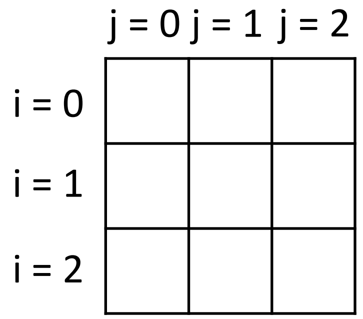


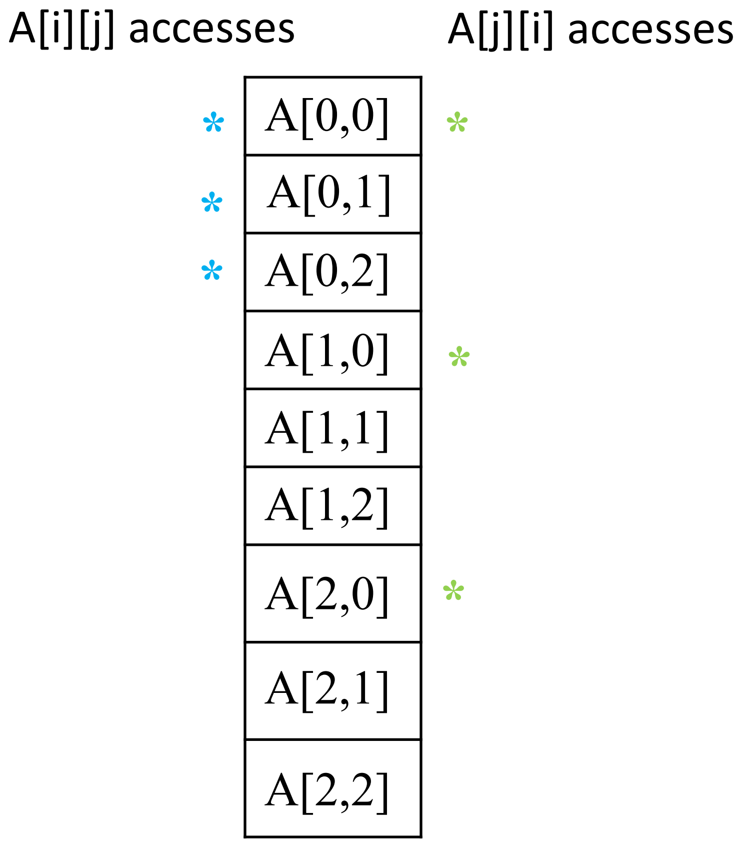
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 3: i = 0, j = 2

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```





In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 3: i = 0, j = 2

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

	j = 0	j = 1	j = 2
i = 0			
i = 1			
i = 2			

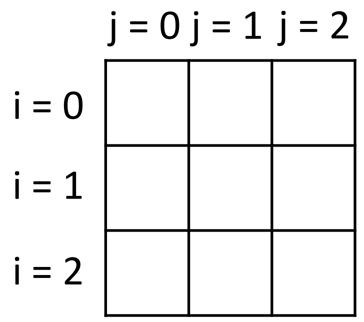
A[i][j] accesses		A[j][i] accesses
*	A[0,0]	*
*	A[0,1]	
*	A[0,2]	
	A[1,0]	*
	A[1,1]	
	A[1,2]	
	A[2,0]	*
	A[2,1]	
	A[2,2]	

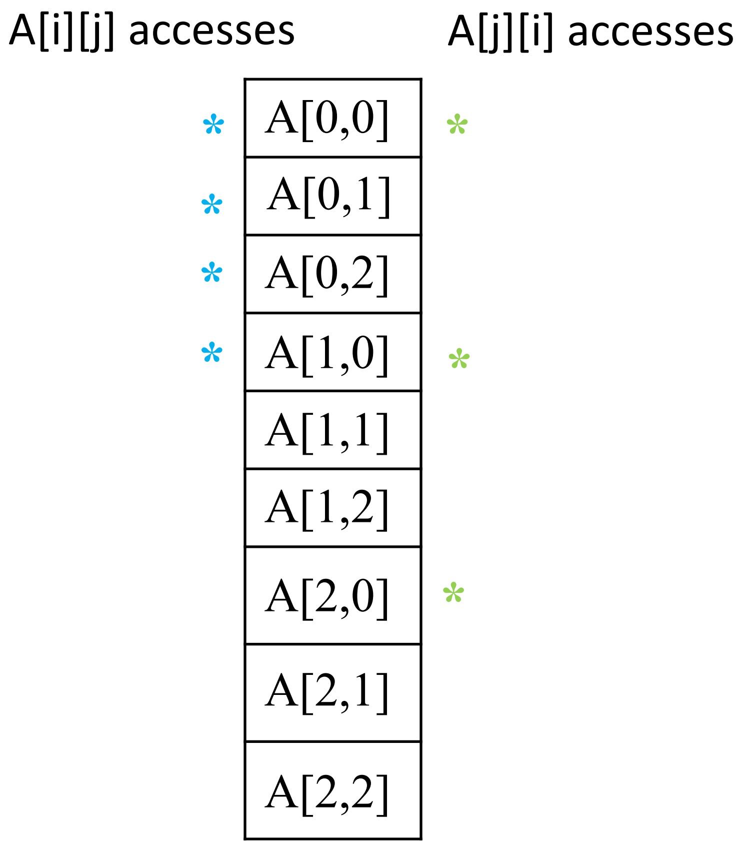
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 4: i = 1, j = 0

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



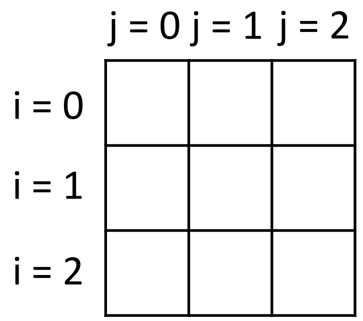


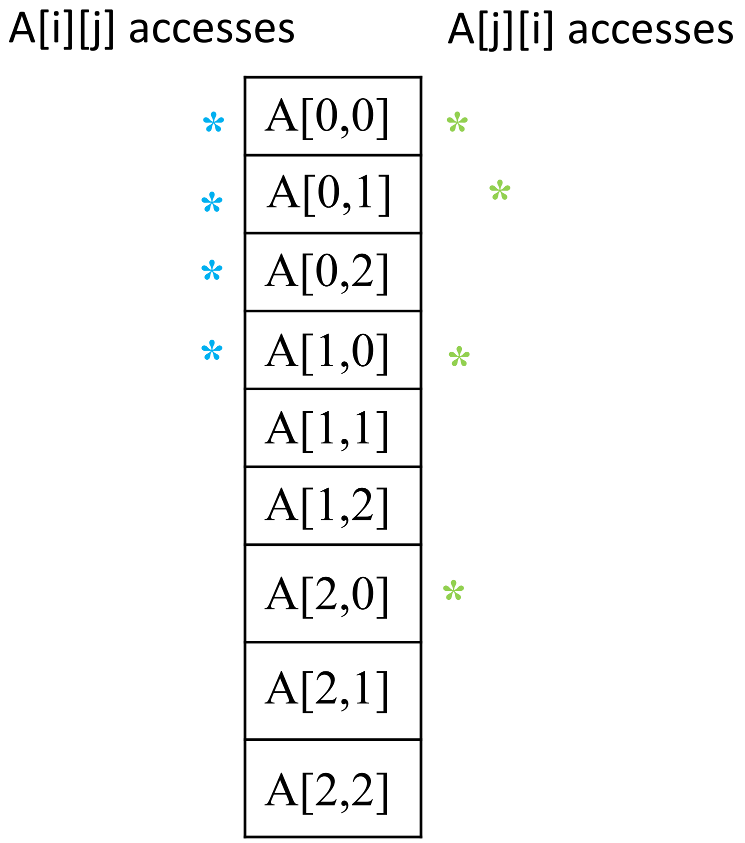
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 4: i = 1, j = 0

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



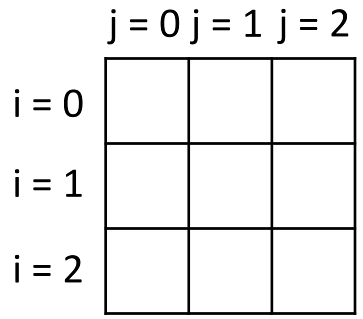


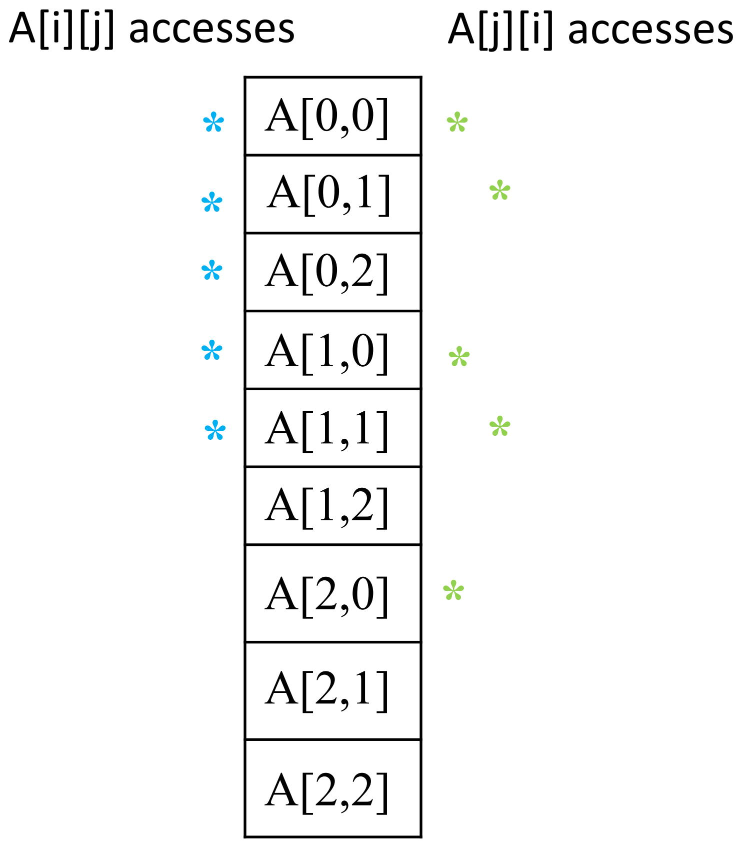
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 4: i = 1, j = 0

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



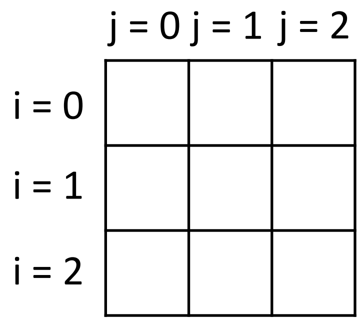


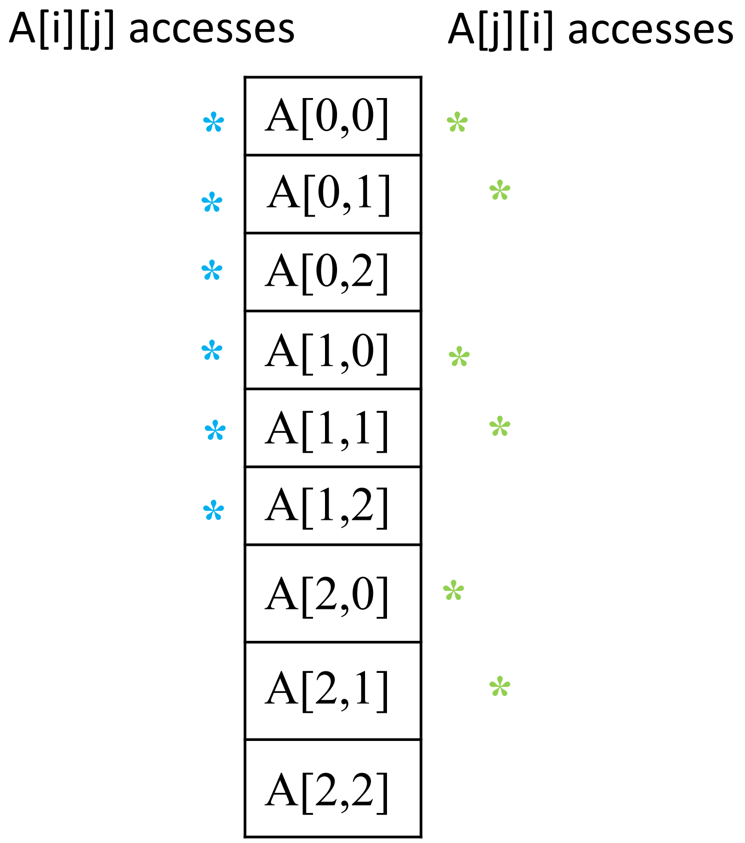
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 5: i = 1, j = 1

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



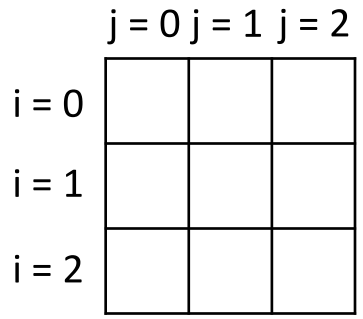


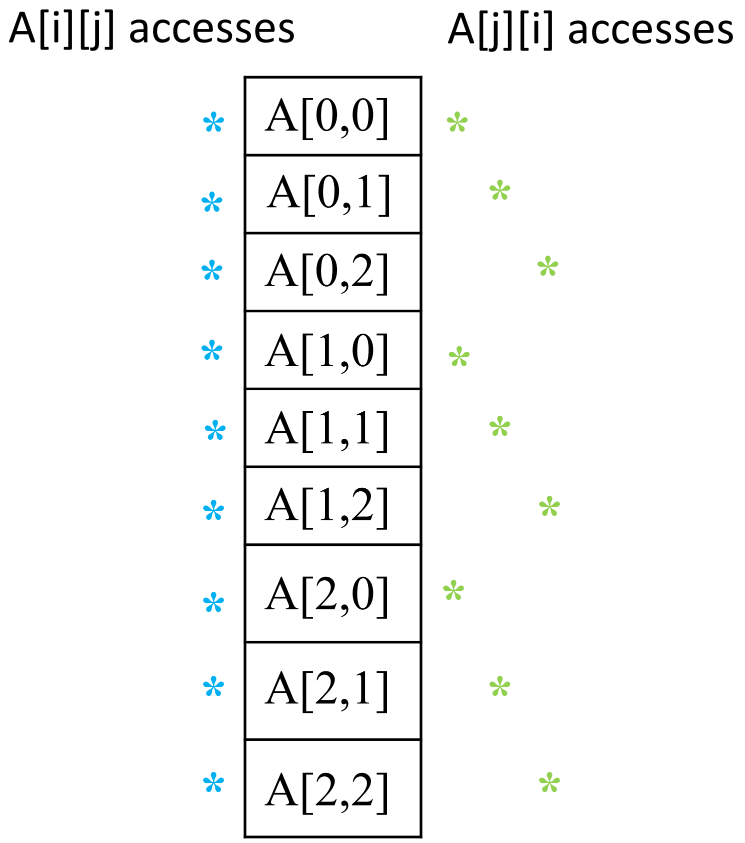
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 6: i = 1, j = 2

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



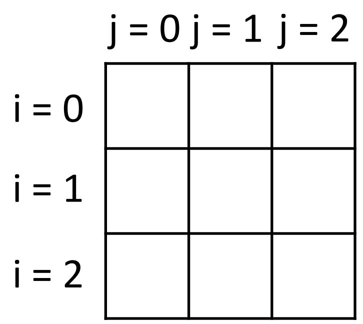


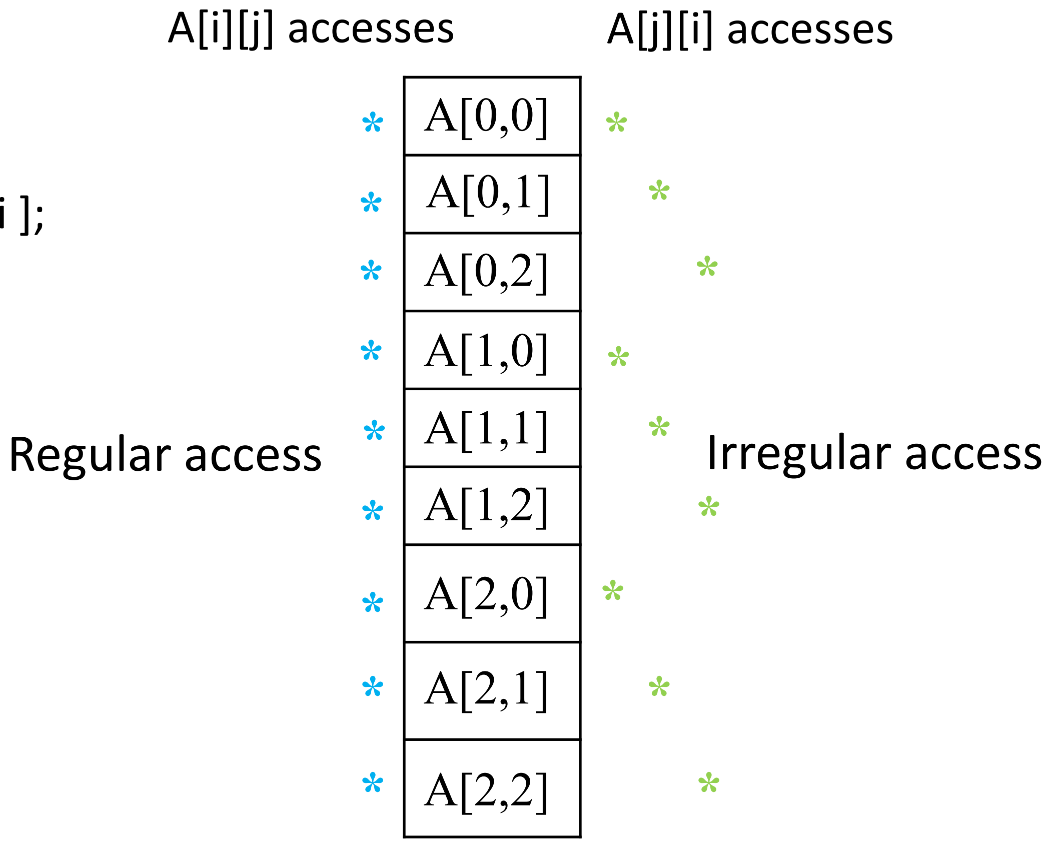
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 9: i = 3, j = 3

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



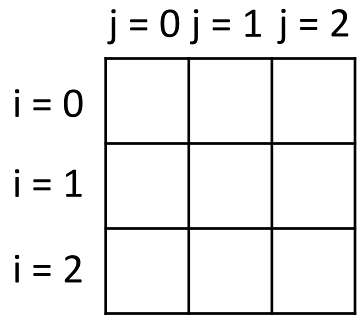


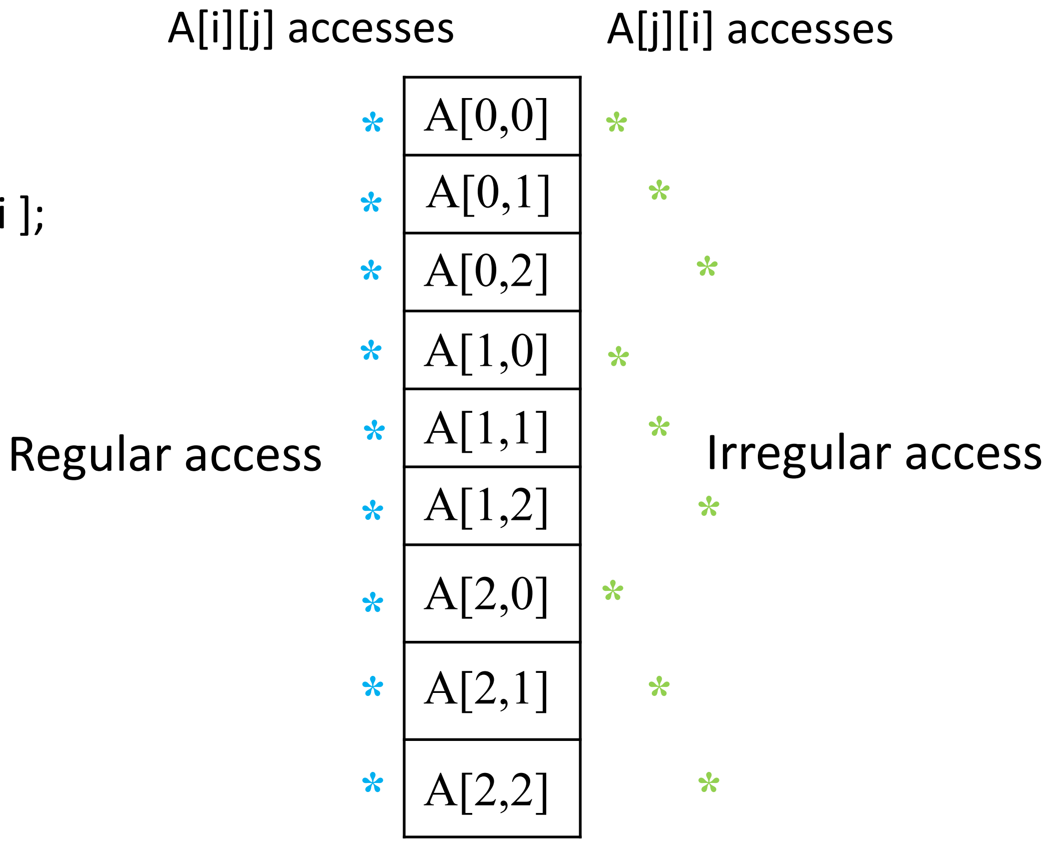
In each iteration, we access two array slots – A[i][j] and A[j][i]

Iteration 9: i = 3, j = 3

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```





In each iteration, we access two array slots – A[i][j] and A[j][i]

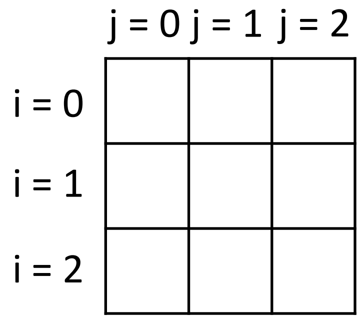
Iteration 9: i = 3, j = 3

Accessing A[i][j] exhibits spatial locality

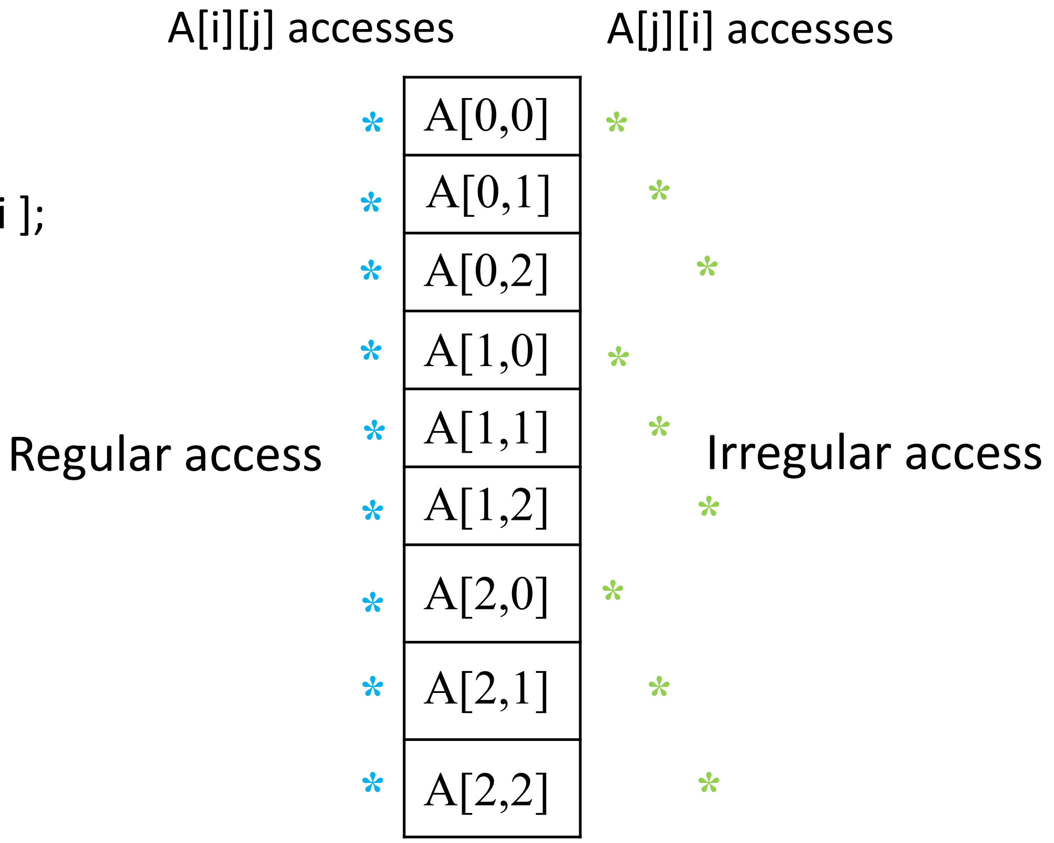
Accessing A[j][i] doesn't exhibit spatial locality

Question 2

```
for (i = 0; i < 3; i++)
      for (j = 0; j < 3; j++)
            A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



What about i and j?

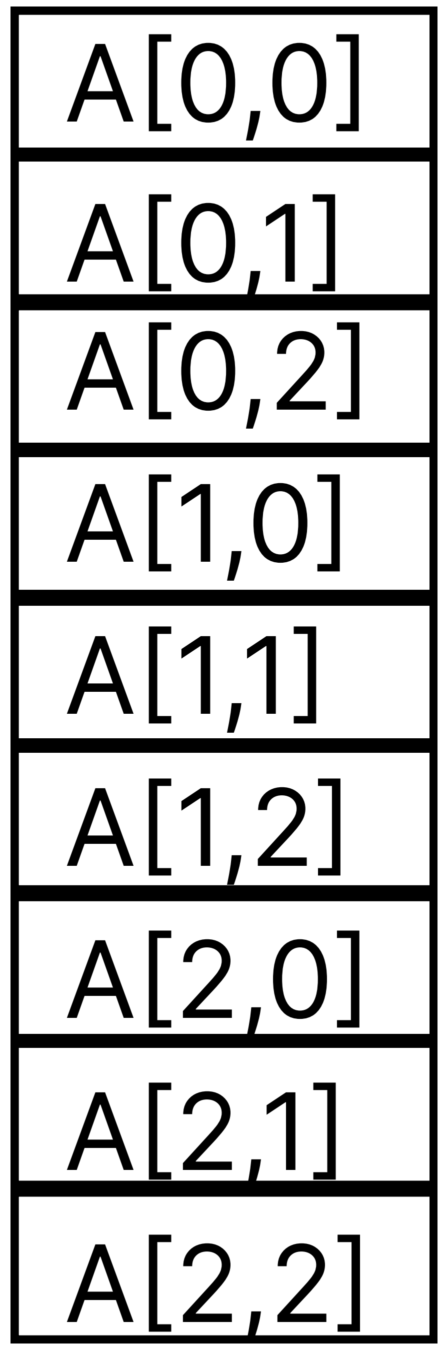


Accessing A[i][j] exhibits spatial locality

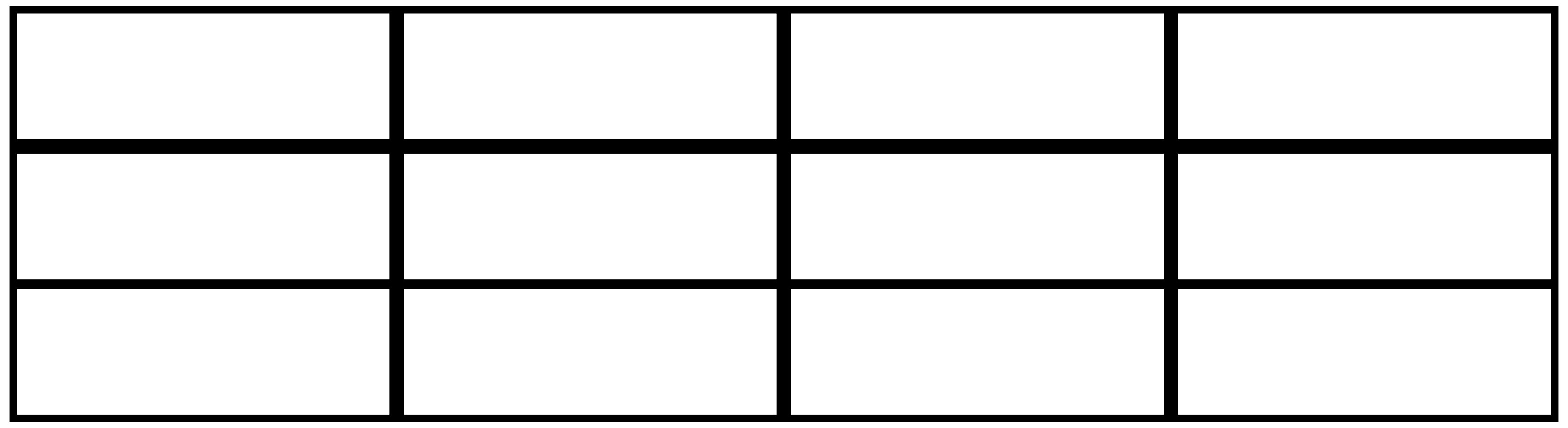
Accessing A[j][i] doesn't exhibit spatial locality

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

Memory

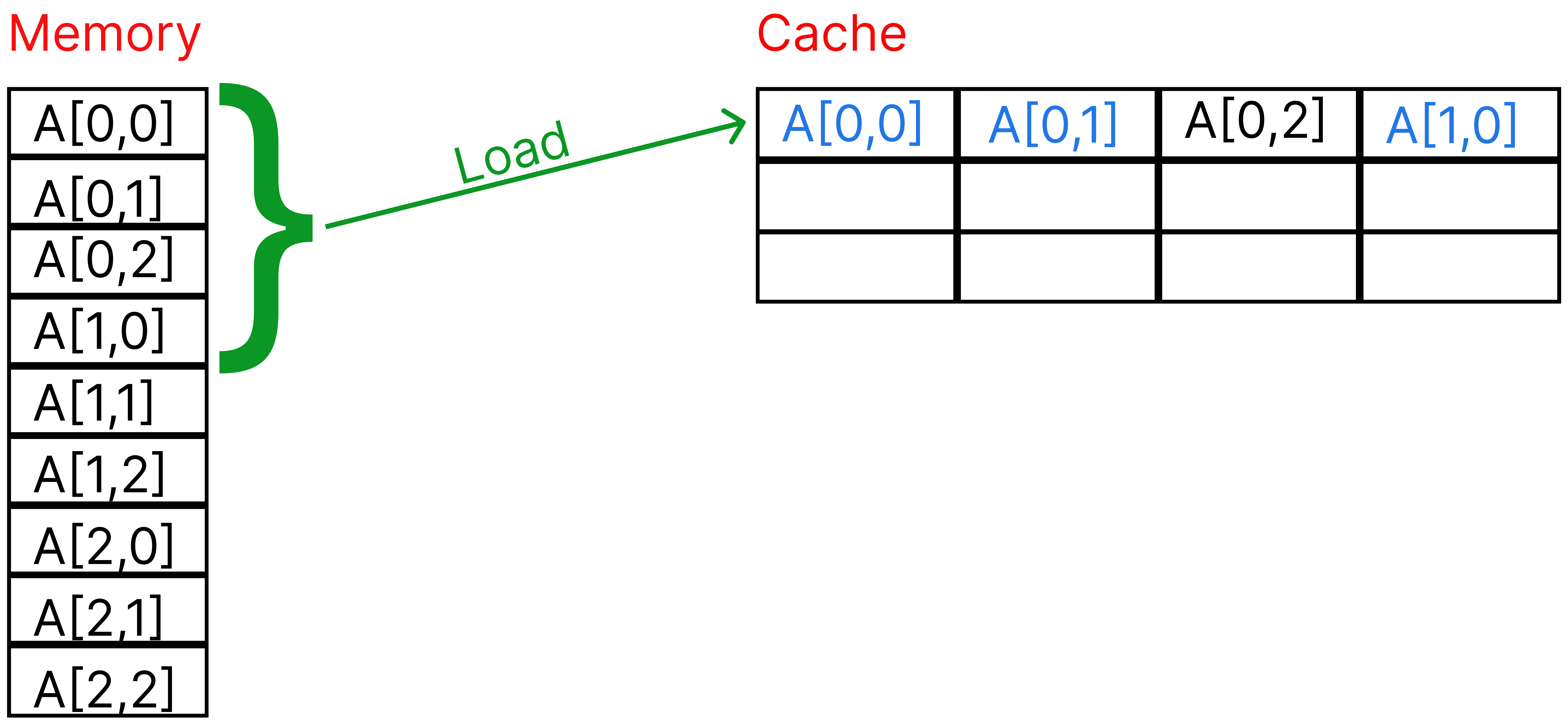


Cache



Iteration 1: i = 0, j = 0. A[0][0] → Miss

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



Iteration 1: i = 0, j = 0. A[0][0] → Hit
Iteration 2: i = 0, j = 1. A[1][0] A[0][1] → Hit

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

Memory

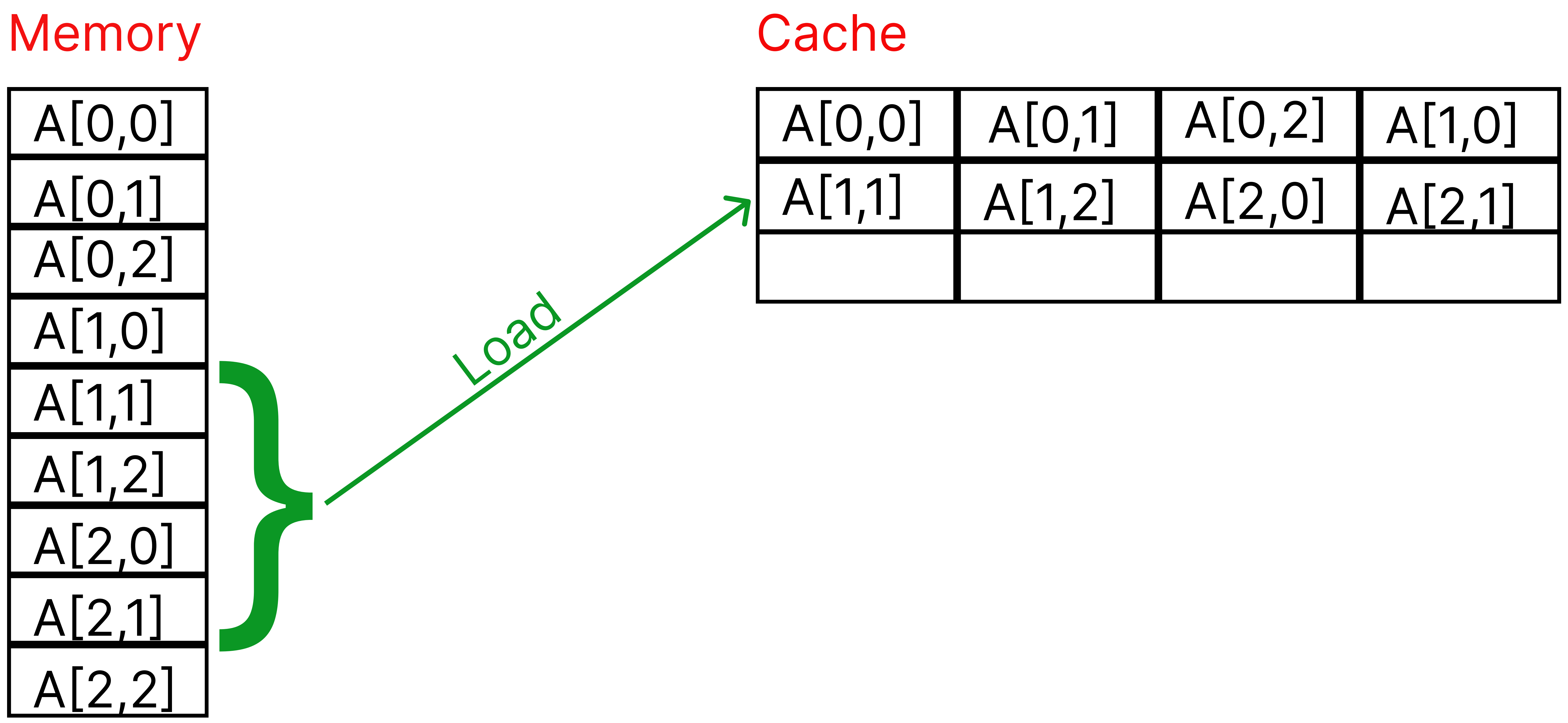
A[0,0]
A[0,1]
A[0,2]
A[1,0]
A[1,1]
A[1,2]
A[2,0]
A[2,1]
A[2,2]

Cache

A[0,0]	A[0,1]	A[0,2]	A[1,0]

Iteration 2: i = 0, j = 1. A[1][0] A[0][1] → Hit

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```



Iteration 3: i = 0, j = 2. A[2][0] → Miss

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```

Memory

A[0,0]
A[0,1]
A[0,2]
A[1,0]
A[1,1]
A[1,2]
A[2,0]
A[2,1]
A[2,2]

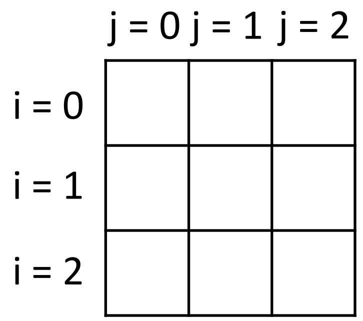
Cache

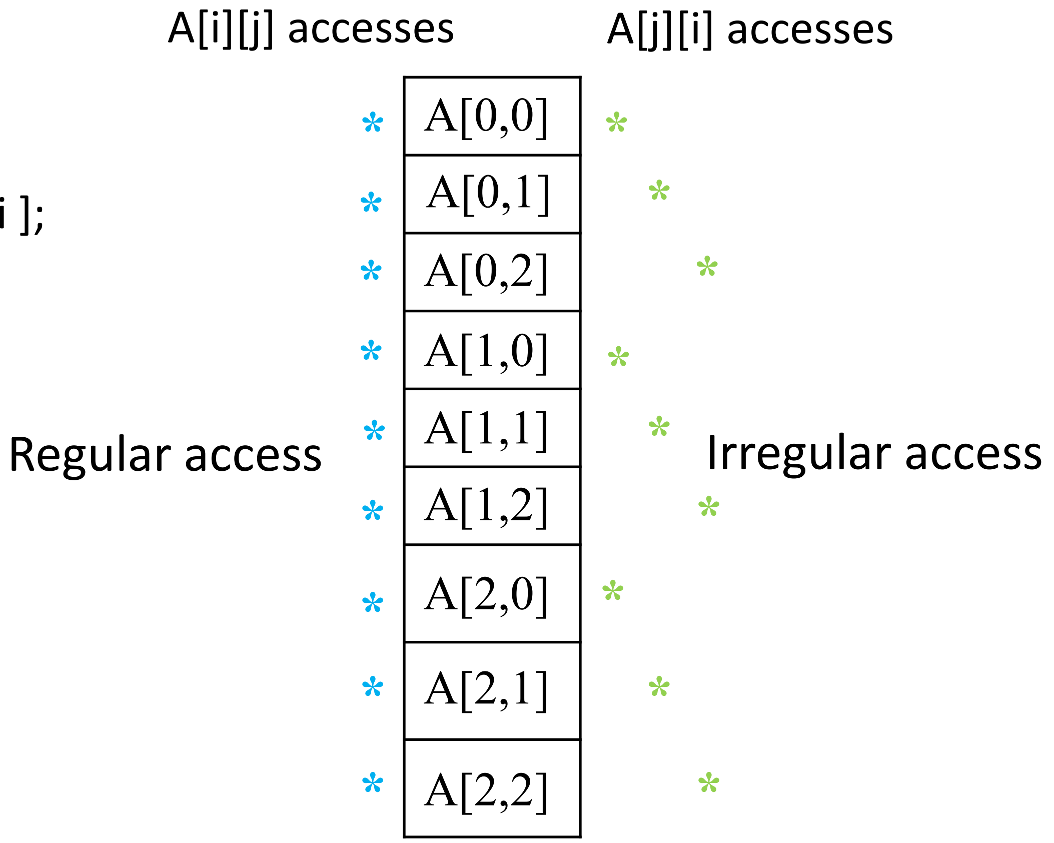
A[0,0]	A[0,1]	A[0,2]	A[1,0]
A[1,1]	A[1,2]	A[2,0]	A[2,1]

Iteration 3: i = 0, j = 2. A[0][2] → Hit

Question 2

```
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        A [ i ] [ j ] = 5 + A [ j ] [ i ];
```





What about i and j?

They exhibit temporal locality because they are accessed in each iteration

Accessing A[i][j] exhibits spatial locality

Accessing A[j][i] doesn't exhibit spatial locality

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- How many sets are there in the cache?

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
?								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- How many sets are there in the cache?

$$\text{Total cache size} = \text{line size} \times associativity \times number\ of\ sets$$
$$16 \times 1024 = 32 \times 4 \times number\ of\ sets \rightarrow number\ of\ sets = 128$$

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

32-bit word memory: 0000 0000 0000 0000 0000 0000 0000 0000

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many byte bits?

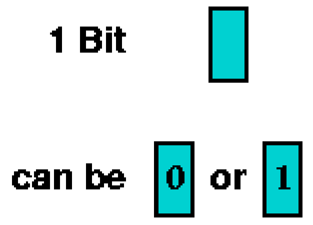
32-bit word memory: 0000 0000 0000 0000 0000 0000 0000 0000

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:



- Which are the tag, index (set) and byte (offset) bits?

How many byte bits?

32-bit word memory: 0000 0000 0000 0000 0000 0000 0000 0000

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

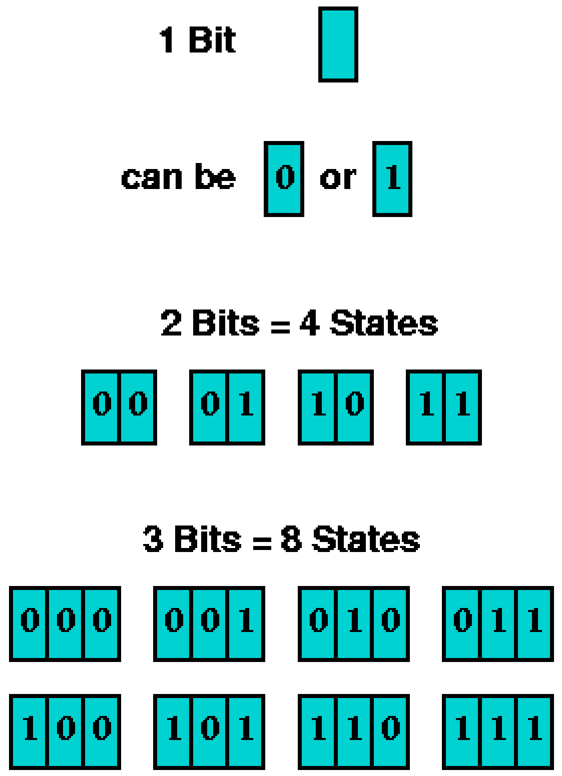
Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many byte bits?

32-bit word memory: 0000 0000 0000 0000 0000 0000 0000 0000



set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

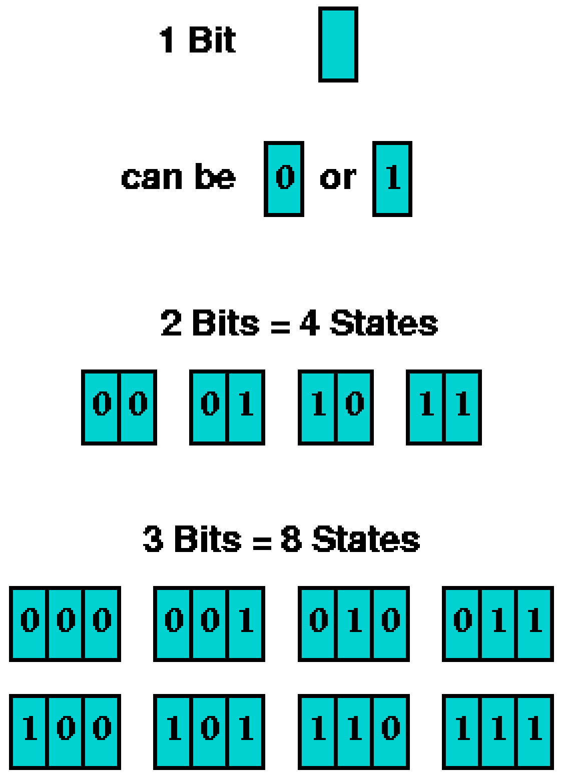
Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many byte bits?

32-bit word memory: 0000 0000 0000 0000 0000 0000 0000 0000



set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

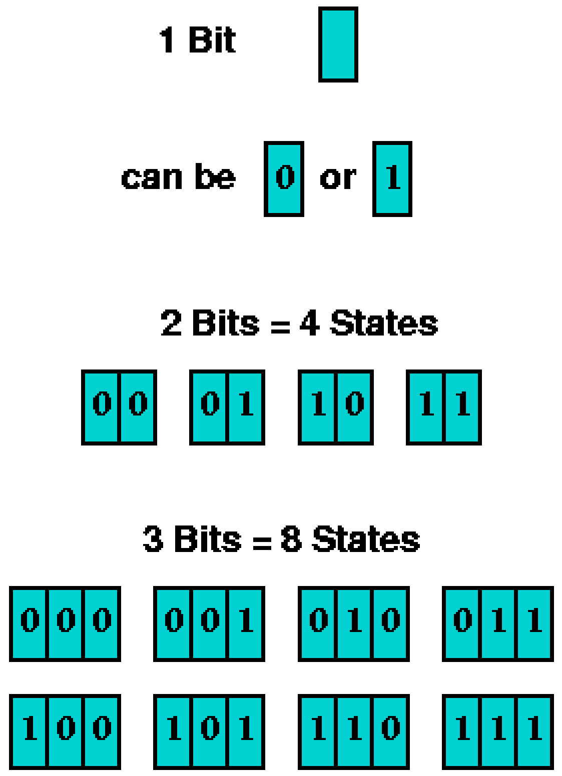
Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many byte bits?

32-bit word memory: 0000 0000 0000 0000 0000 0000 0000 0000



N bits = 2^N States

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many byte bits? Cache line is 32 bytes, so …

N bits = 2^N States

32-bit word memory: 0000 0000 0000 0000 0000 0000 0000 0000

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many byte bits? Cache line is 32 bytes, so … 5 bits

2^5 = 32
N bits = 2^N States

32-bit word memory: 0000 0000 0000 0000 0000 0000 0000 0000

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many byte bits? Cache line is 32 bytes, so … 5 bits

32-bit word memory: $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many set bits?

32-bit word memory: 0000 0000 0000 0000 0000 0000 000$\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many set bits? We have 128 sets, so 7 bits

32-bit word memory: 0000 0000 0000 0000 0000 0000 000$\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many set bits? We have 128 sets, so 7 bits

32-bit word memory: $0000\ 0000\ 0000\ 0000\ 0000\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0		32 bytes						
1								
2								
3								
	…	…	…	…	…	…	…	…
127								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

32-bit word memory: $0000\ 0000\ 0000\ 0000\ 0000\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0000000		32 bytes						
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many tag bits?

32-bit word memory: $0000\ 0000\ 0000\ 0000\ 0000\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0000000		32 bytes						
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many tag bits? The remaining 20 bits.

$$\text{32-bit word memory: } 0000\ 0000\ 0000\ 0000\ 0000\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$$

set	tag	data	tag	data	tag	data	tag	data
0000000		32 bytes						
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many tag bits? The remaining 20 bits.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0000000		32 bytes						
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Which are the tag, index (set) and byte (offset) bits?

How many tag bits? The remaining 20 bits.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

Note: Tag bits are stored in the cache, along with data bytes. But set and offset bits are not stored in the cache; they are used to find the location of the byte in the cache.

set	tag	data	tag	data	tag	data	tag	data
0000000		32 bytes						
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

  A[0]?

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0000000								
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

A[0]?

Address of A[0]$_0$: 0000 0000 0000 0000 0000 0000 0000 0000

set	tag	data	tag	data	tag	data	tag	data
0000000								
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{\text{0000 0000 0000 0000 0000}}_{\text{Tag}}\ \underbrace{\text{0000 000}}_{\text{Set}}\underbrace{\text{0 0000}}_{\text{Offset}}$

A[0]?

Address of $A[0]_0$: 0000 0000 0000 0000 0000 0000 0000 0000

Address of $A[0]_1$: 0000 0000 0000 0000 0000 0000 0000 0001

set	tag	data	tag	data	tag	data	tag	data
0000000								
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

A[0]?

Set bits

Address of A$[0]_0$: 0000 0000 0000 0000 0000 $\boxed{0000\ 000}$0 0000

Address of A$[0]_1$: 0000 0000 0000 0000 0000 $\boxed{0000\ 000}$0 0001

set	tag	data	tag	data	tag	data	tag	data
0000000								
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

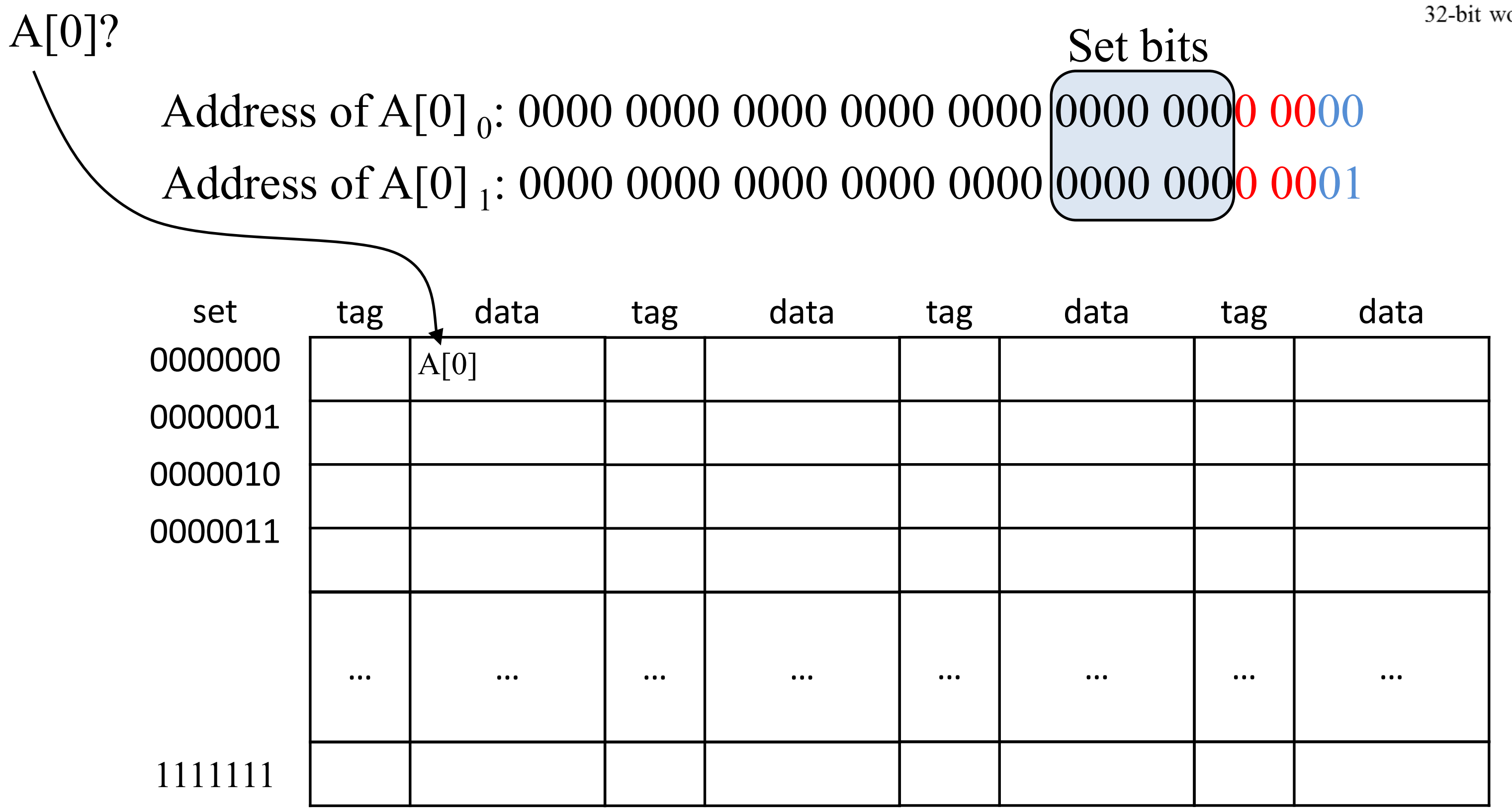
Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{\text{0000 0000 0000 0000 0000}}_{\text{Tag}}\ \underbrace{\text{0000 000}}_{\text{Set}}\underbrace{\text{0 0000}}_{\text{Offset}}$


A[0]?
Set bits
Address of A[0]₀: 0000 0000 0000 0000 0000 0000 0000 0000
Address of A[0]₁: 0000 0000 0000 0000 0000 0000 0000 0001
set
tag
data
tag
data
tag
data
tag
data
0000000
A[0]
0000001
0000010
0000011
...
1111111

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

A[1]?

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0000000		A[0]						
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

A[1]?

Set bits

Address of $A[1]_0$: 0000 0000 0000 0000 0000 $\boxed{0000\ 000}$0 0100

Address of $A[1]_3$: 0000 0000 0000 0000 0000 $\boxed{0000\ 000}$0 0111

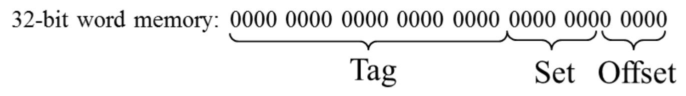
set	tag	data	tag	data	tag	data	tag	data
0000000		A[0]						
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

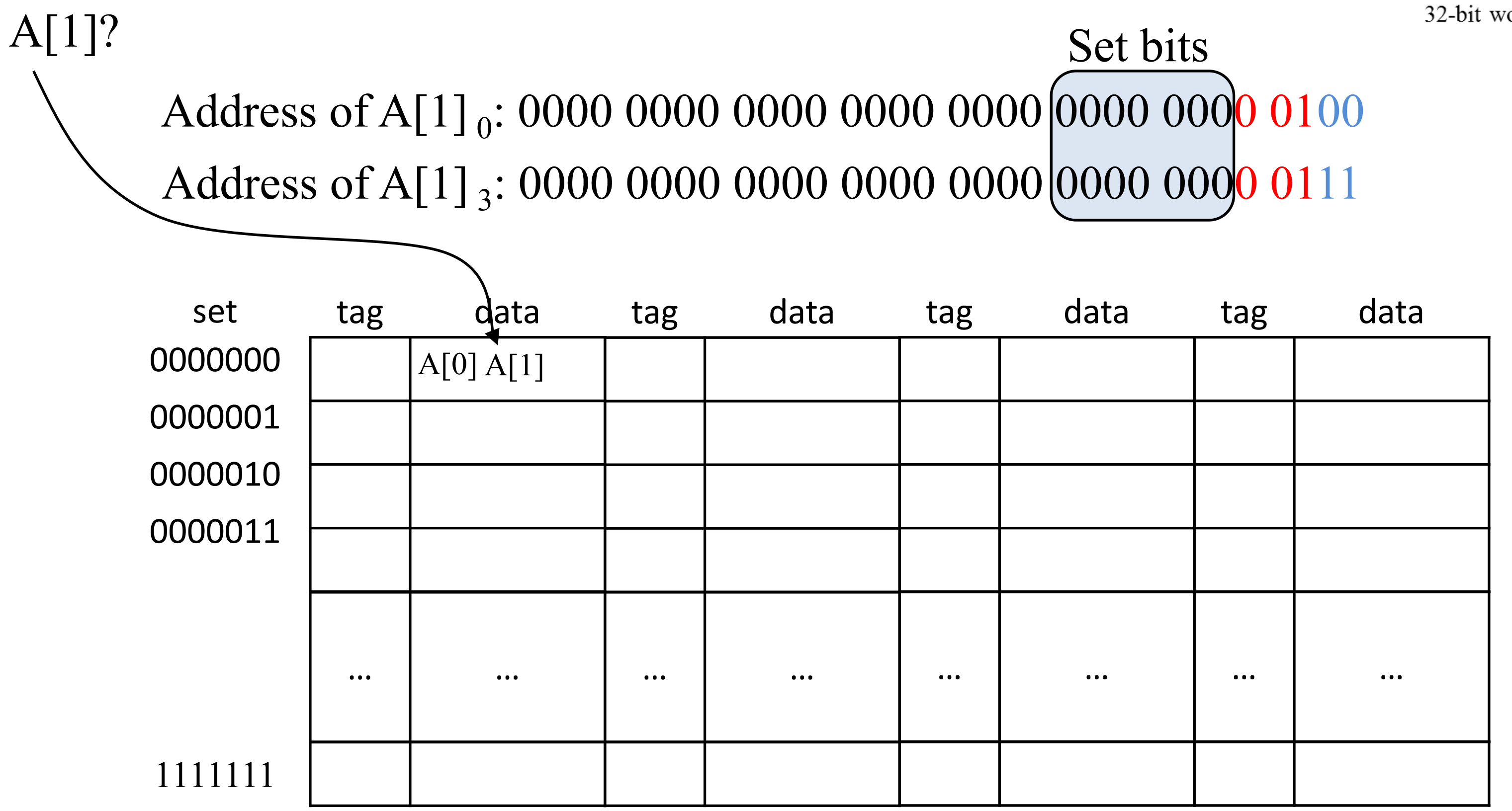
Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.





A[1]?
Set bits
Address of A[1]$_0$: 0000 0000 0000 0000 0000 0000 0000 0100
Address of A[1]$_3$: 0000 0000 0000 0000 0000 0000 0000 0111
set
tag
data
tag
data
tag
data
tag
data
0000000
0000001
0000010
0000011
1111111
A[0] A[1]
…

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

  What is the last element in the same line?

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0000000								
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

What is the last element in the same line?

Set bits

Address of A[?]$_0$: 0000 0000 0000 0000 0000 $\boxed{0000\ 000}$1 1100

Address of A[?]$_3$: 0000 0000 0000 0000 0000 $\boxed{0000\ 000}$1 1111

set	tag	data	tag	data	tag	data	tag	data
0000000								
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

  What is the last element in the same line?

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

Set bits

Address of A[?]$_0$: 0000 0000 0000 0000 0000 0000 0001 1100

Address of A[?]$_3$: 0000 0000 0000 0000 0000 0000 0001 1111

set	tag	data	tag	data	tag	data	tag	data
0000000		A[0-7]						
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

A[8]?

Set bits

Address of A[8] $_0$: 0000 0000 0000 0000 0000 0000 0010 0000

Address of A[8] $_1$: 0000 0000 0000 0000 0000 0000 0010 0001

set	tag	data	tag	data	tag	data	tag	data
0000000		A[0-7]						
0000001								
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

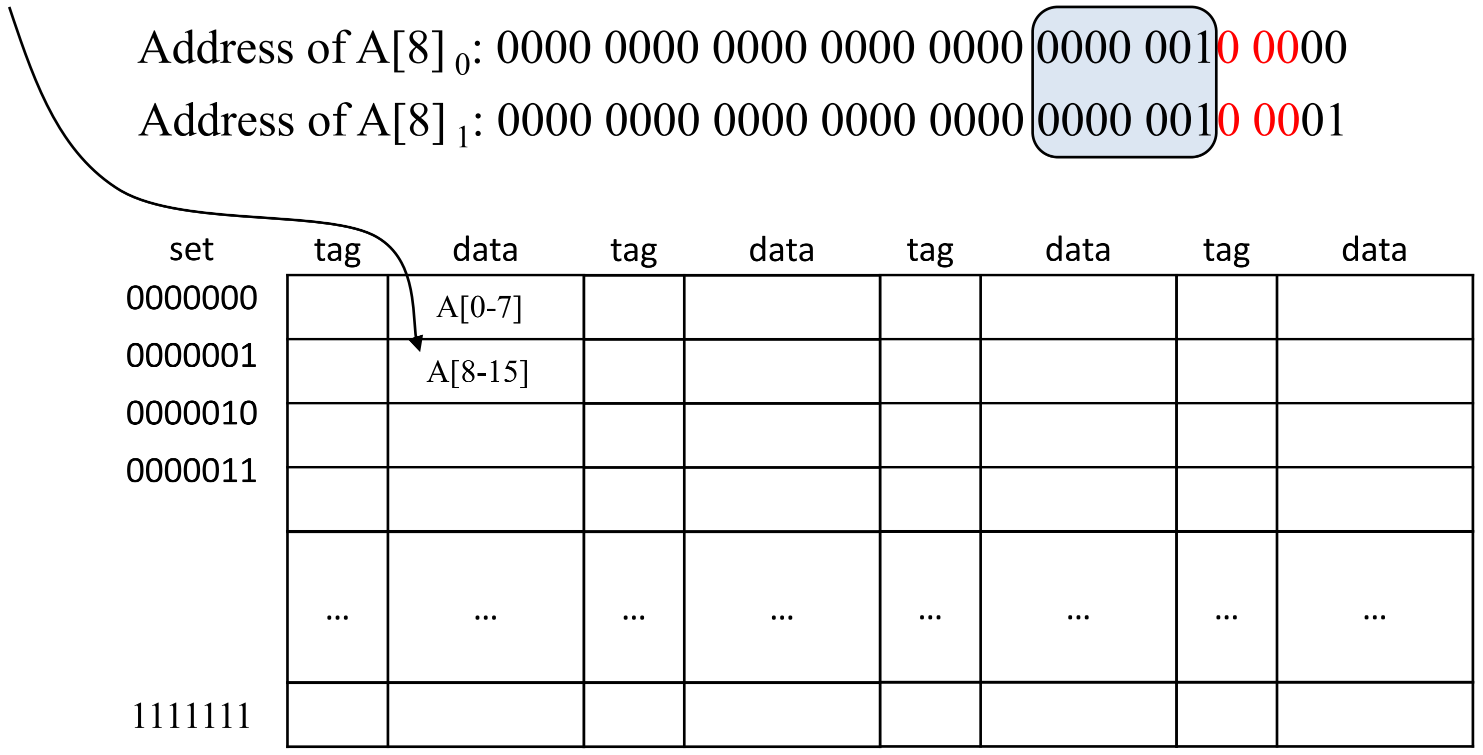
32-bit word memory: $\underbrace{\text{0000 0000 0000 0000 0000 000}}_{\text{Tag}}\underbrace{\text{0 0000 00}}_{\text{Set}}\underbrace{\text{00 000}}_{\text{Offset}}$

A[8]?

Set bits

Address of A[8] $_0$: 0000 0000 0000 0000 0000 $\boxed{\text{0000 001}}$0 0000

Address of A[8] $_1$: 0000 0000 0000 0000 0000 $\boxed{\text{0000 001}}$0 0001



set	tag	data	tag	data	tag	data	tag	data
0000000		A[0-7]						
0000001		A[8-15]						
0000010								
0000011								
	…	…	…	…	…	…	…	…
1111111								

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000\ 000}_{\text{Tag}}\underbrace{0\ 0000\ 00}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

A[8]?

Set bits

Address of $A[8]_0$: $0000\ 0000\ 0000\ 0000\ 0000\ \boxed{0000\ 001}0\ 0000$

Address of $A[8]_1$: $0000\ 0000\ 0000\ 0000\ 0000\ \boxed{0000\ 001}0\ 0001$

set	tag	data	tag	data	tag	data	tag	data
0000000		A[0-7]						
0000001		A[8-15]						
0000010		A[16-23]						
0000011		A[24-31]						
	…	…	…	…	…	…	…	…
1111111		A[1016-1023]						

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

  A[1050]?

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

set	tag	data	tag	data	tag	data	tag	data
0000000	0	A[0-7]						
0000001	0	A[8-15]						
0000010	0	A[16-23]						
0000011	0	A[24-31]						
	…	…	…	…	…	…	…	…
1111111	0	A[1016-1023]						

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000\ 000}_{\text{Tag}}\underbrace{0\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

A[1050]?

Address of A[1050] $_0$:

set	tag	data	tag	data	tag	data	tag	data
0000000	0	A[0-7]						
0000001	0	A[8-15]						
0000010	0	A[16-23]						
0000011	0	A[24-31]						
	…	…	…	…	…	…	…	…
1111111	0	A[1016-1023]						

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

A[1050]?

Address of A[1050]$_0$: byte number $1050 \times 4 = 4200$

set	tag	data	tag	data	tag	data	tag	data
0000000	0	A[0-7]						
0000001	0	A[8-15]						
0000010	0	A[16-23]						
0000011	0	A[24-31]						
	…	…	…	…	…	…	…	…
1111111	0	A[1016-1023]						

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

A[1050]?

Address of $A[1050]_0$: byte number $1050 \times 4 = 4200$

4200 = 0000 0000 0000 0000 0001 0000 0110 1000

set	tag	data	tag	data	tag	data	tag	data
0000000	0	A[0-7]						
0000001	0	A[8-15]						
0000010	0	A[16-23]						
0000011	0	A[24-31]						
	…	…	…	…	…	…	…	…
1111111	0	A[1016-1023]						

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000\ 00}_{\text{Tag}}\underbrace{00\ 00000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

A[1050]?

Address of $A[1050]_0$: byte number $1050 \times 4 = 4200$

$4200 = 0000\ 0000\ 0000\ 0000\ 0001\ \underbrace{\boxed{0000\ 011}}_{\text{Set 3}}0\ 1000$

set	tag	data	tag	data	tag	data	tag	data
0000000	0	A[0-7]						
0000001	0	A[8-15]						
0000010	0	A[16-23]						
0000011	0	A[24-31]						
	…	…	…	…	…	…	…	…
1111111	0	A[1016-1023]						

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

A[1050]?

Address of $A[1050]_0$: byte number $1050 \times 4 = 4200$

$4200 = 0000\ 0000\ 0000\ 0000\ 0001\ \underbrace{\boxed{0000\ 011}}_{\text{Set 3}}0\ 1000$

set	tag	data	tag	data	tag	data	tag	data
0000000	0	A[0-7]						
0000001	0	A[8-15]						
0000010	0	A[16-23]						
0000011	0	A[24-31]		A[1050]				
	…	…	…	…	…	…	…	…
1111111	0	A[1016-1023]						

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

A[1050]?

Or: Set number = (Line number) *mod* (number of sets)

Address of $A[1050]_0$: byte number $1050 \times 4 = 4200$

Line number = $4200 \div 32 = 131$ Set number = $131 \; mod \; 128 = 3$

set	tag	data	tag	data	tag	data	tag	data
0000000	0	A[0-7]						
0000001	0	A[8-15]						
0000010	0	A[16-23]						
0000011	0	A[24-31]		A[1050]				
	…	…	…	…	…	…	…	…
1111111	0	A[1016-1023]						

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

  What are the other elements in the same line with A[1050]?

$$\text{32-bit word memory: } \underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$$

set	tag	data	tag	data	tag	data	tag	data
0000000	0	A[0-7]						
0000001	0	A[8-15]						
0000010	0	A[16-23]						
0000011	0	A[24-31]						
	…	…	…	…	…	…	…	…
1111111	0	A[1016-1023]						

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

  What are the other elements in the same line with A[1050]?

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

Address of $A[1050]_0$ = $0000\ 0000\ 0000\ 0000\ 0001\ 0000\ 011\boxed{0\ 10}00$

set	tag	data	tag	data	tag	data	tag	data
0000000	0	A[0-7]						
0000001	0	A[8-15]						
0000010	0	A[16-23]						
0000011	0	A[24-31]						
	…	…	…	…	…	…	…	…
1111111	0	A[1016-1023]						

Cache structure

Question 3

We have a byte-addressable 32-bit word memory. We also have a 4-way set associative cache, where each cache line is 32 bytes. Total cache size is 16KB. Answer the following questions:

- Assume we have an array of 4K integers, A[4096]. Where is each of the following array elements located in the cache? Suppose A[0] is at address 0 and integer is 4-bytes long.

  What are the other elements in the same line with A[1050]?

32-bit word memory: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000}_{\text{Tag}}\ \underbrace{0000\ 000}_{\text{Set}}\underbrace{0\ 0000}_{\text{Offset}}$

Address of $A[1050]_0$ = $0000\ 0000\ 0000\ 0000\ 0001\ 0000\ 011\boxed{0\ 10}00$

set	tag	data	tag	data	tag	data	tag	data
0000000	0	A[0-7]						
0000001	0	A[8-15]						
0000010	0	A[16-23]						
0000011	0	A[24-31]	1	A[1048-1055]				
	…	…	…	…	…	…	…	…
1111111	0	A[1016-1023]						

Cache structure