RUTGERS

THE STATE UNIVERSITY
OF NEW JERSEY

Recitation 10

Jae Woo Joo

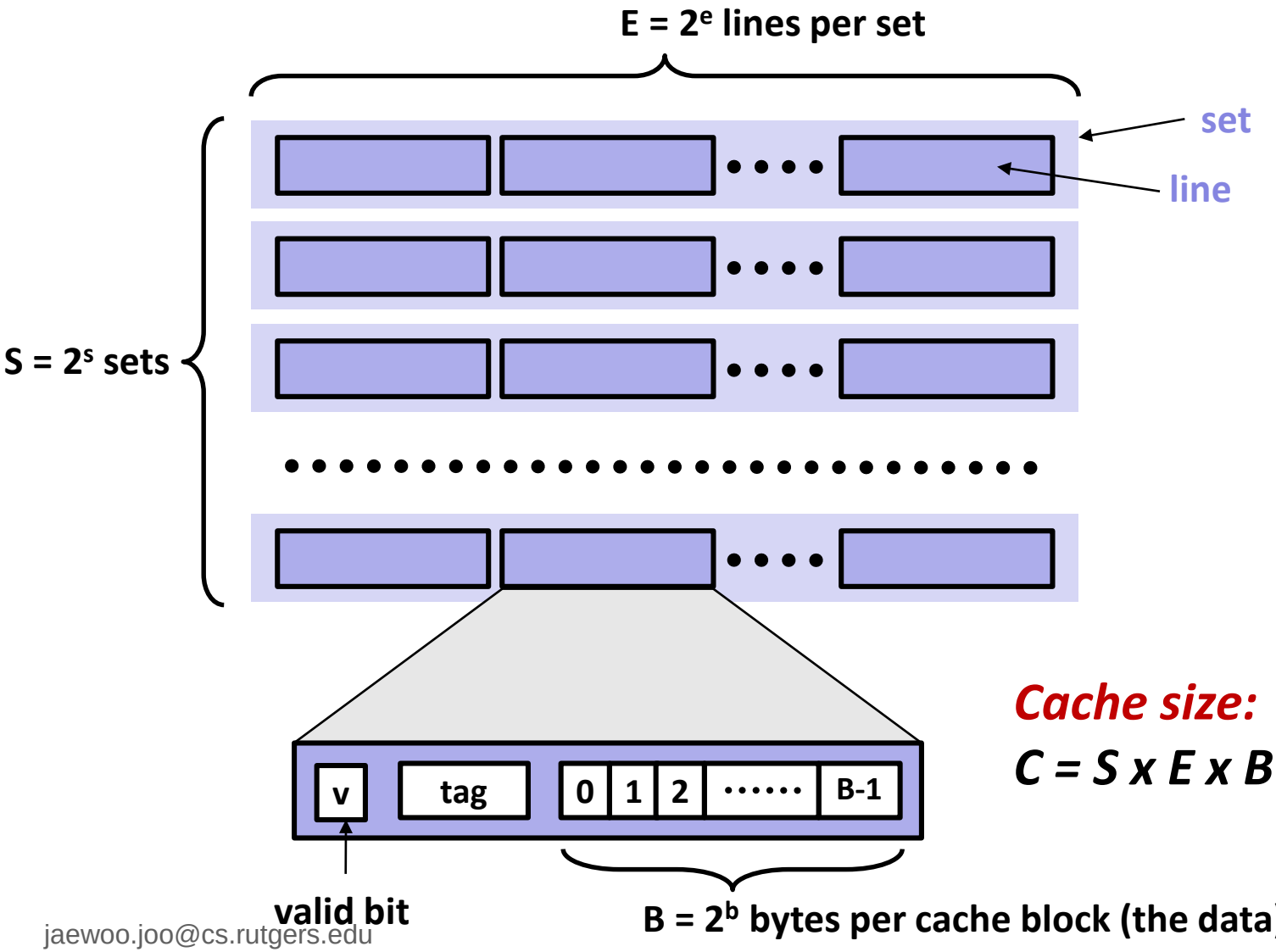
General Cache Organization

- Block
  - A fixed size chunk of data that moves back and forth between the cache and main memory
- Line (Associativity)
  - Meta information about one block consisting of tag bits, selector bits, and valid bit
- Set
  - A collection of one or more lines

General Cache Organization

- Cache capacity
  - The total number of bytes that a cache can hold
  - C = S * E * B
  - S : Number of sets
  - E : Number of lines per set (associativity)
  - B : Block size

General Cache Organization

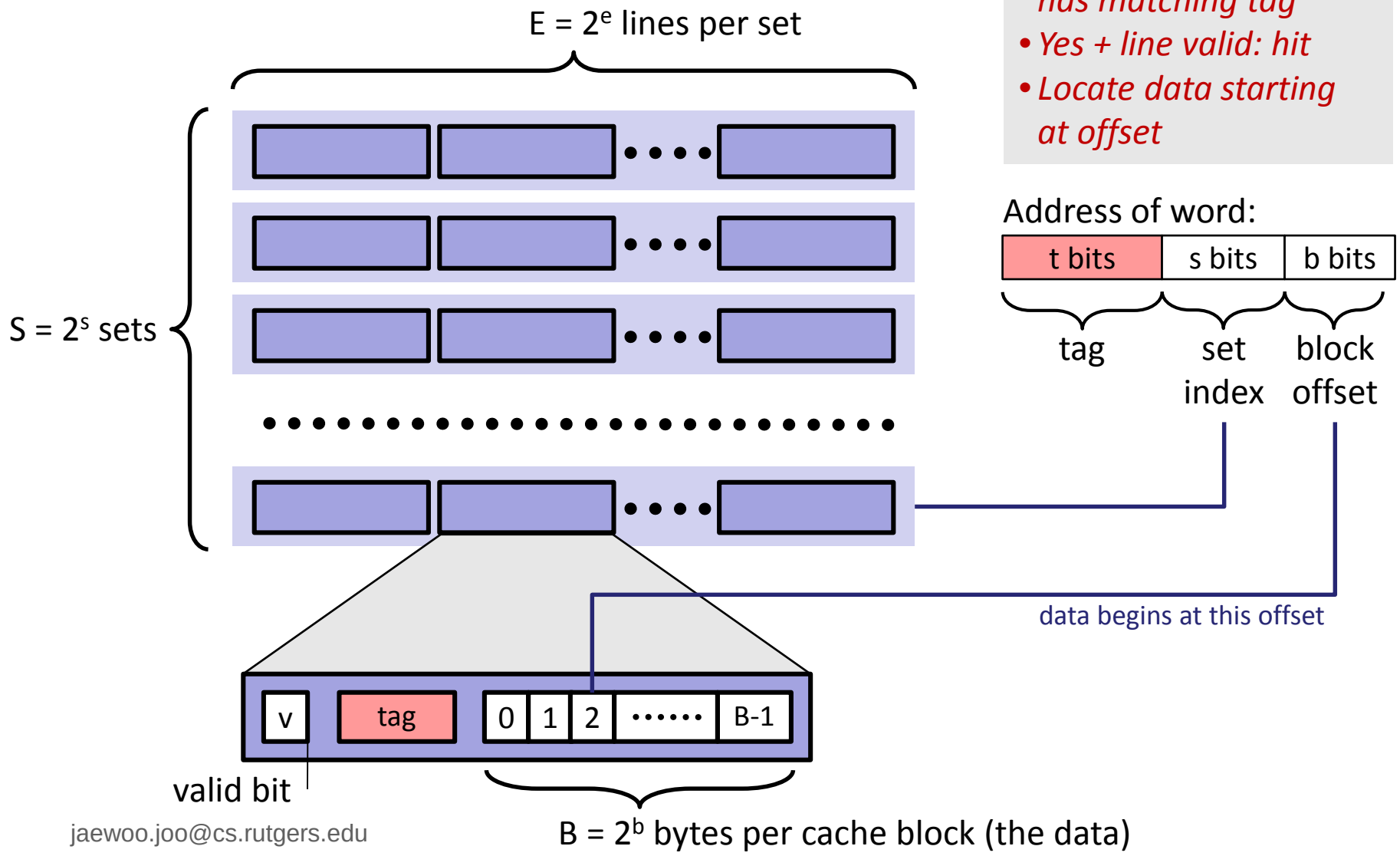


Cache size:

C = S x E x B data bytes

Cache Read

- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*
- *Locate data starting at offset*

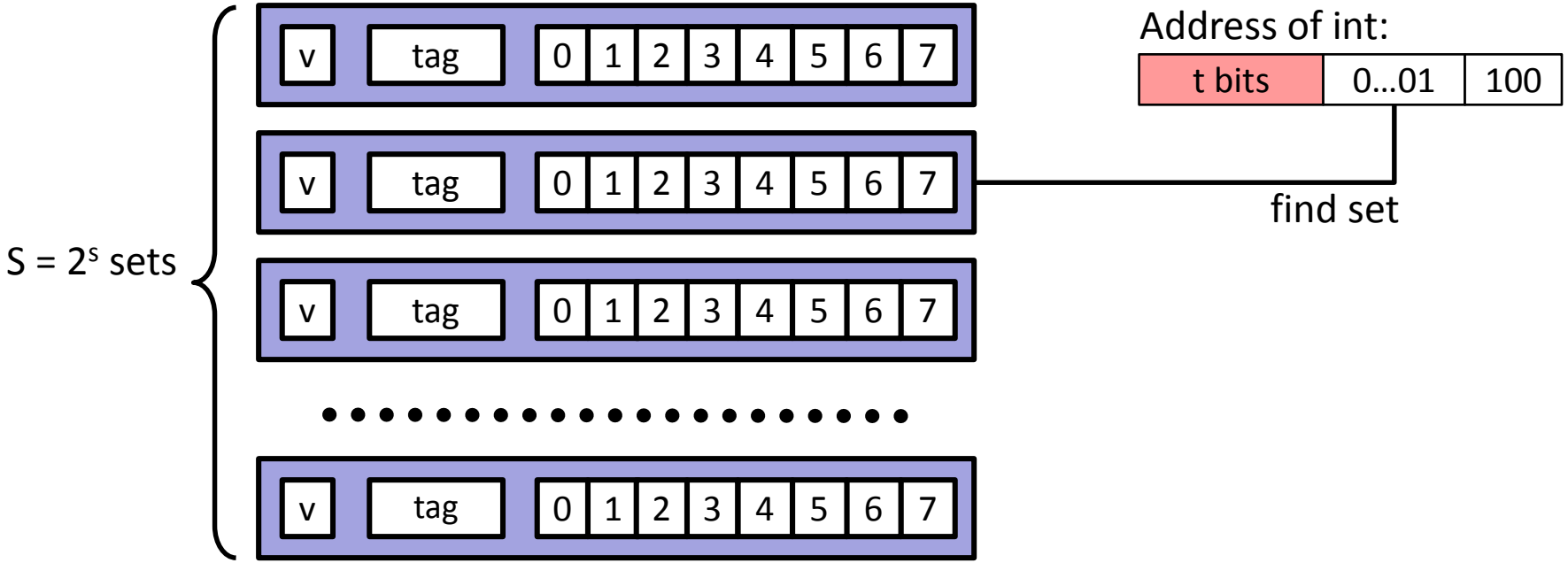


Memory address

- Block offset: b bits
- Set index: s bits
- Tag Bits: (Address size – b – s)

Direct Mapped Cache (E = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes



Direct-Mapped Cache Simulation

t=1	s=2	b=1
x	xx	x

M=16 bytes (4-bit addresses), B=2 bytes/block,
S=4 sets, E=1 Blocks/set

Address trace (reads, one byte per read):

0	[$0\underline{00}0_2$],	miss
1	[$0\underline{00}1_2$],	hit
7	[$0\underline{11}1_2$],	miss
8	[$1\underline{00}0_2$],	miss
0	[$0\underline{00}0_2$]	miss

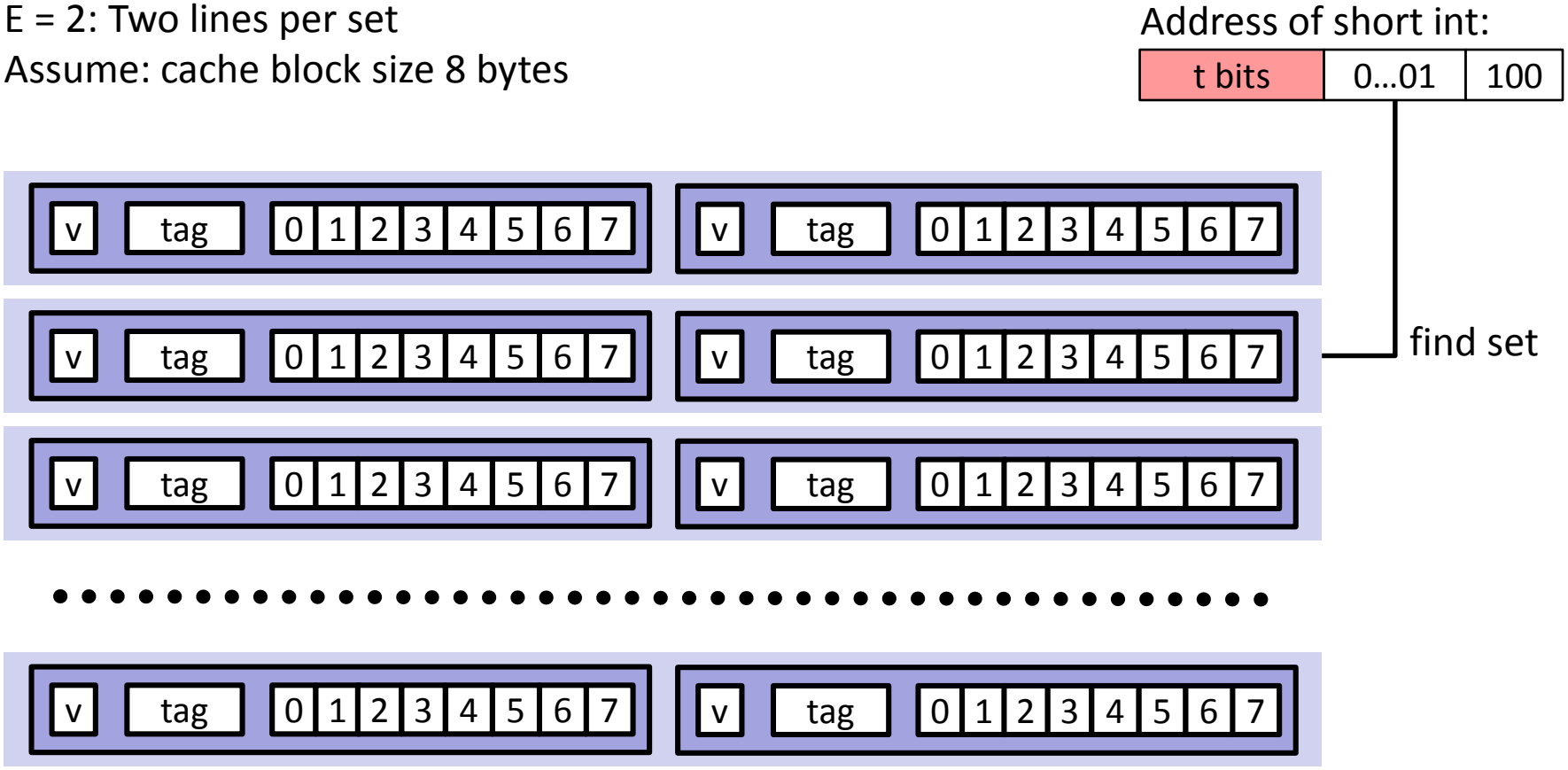
	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

jaewoo.joo@cs.rutgers.edu

E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set

Assume: cache block size 8 bytes



2-Way Set Associative Cache Simulation

t=2	s=1	b=1
xx	x	x

M=16 byte addresses, B=2 bytes/block,
S=2 sets, E=2 blocks/set

Address trace (reads, one byte per read):

0	[00$\underline{0}$0$_2$],	miss
1	[00$\underline{0}$1$_2$],	hit
7	[01$\underline{1}$1$_2$],	miss
8	[10$\underline{0}$0$_2$],	miss
0	[00$\underline{0}$0$_2$]	hit

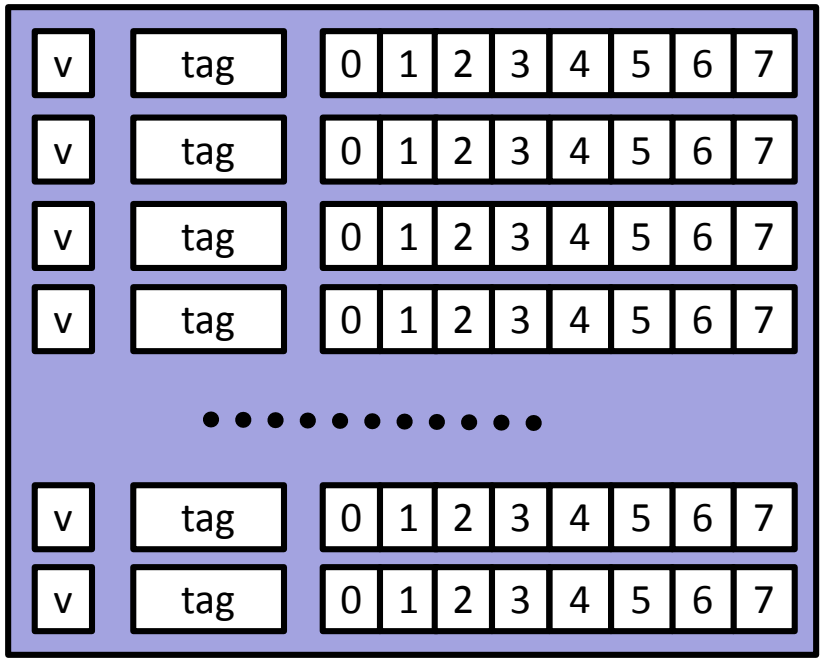
	v	Tag	Block
Set 0	1	00	M[0-1]
	1	10	M[8-9]
Set 1	1	01	M[6-7]
	0		

jaewoo.joo@cs.rutgers.edu

Fully Associative Caches

- Set selection is trivial

Assume: cache block size 8 bytes



Cache Design

- Factors affecting cache design
  - Cache size
  - Block size
  - Replacement algorithm for blocks
  - Write policy

Cache Design

- Cache size
  - Determined by economics
  - The larger, the better but there is decreasing marginal benefit after some point
- Block size
  - As block size increases, probability of hit goes up due to principle of locality
  - If block size gets too large, the probability of hit goes down for newly fetched data

Cache Design

- Block replacement policy

- First In First Out (FIFO)
  - When a block needs to be replaced, the cache evicts the block that was accessed first

- Least Recently Used (LRU)
  - Hardware keeps track of the access history
  - Replace the entry that has not been used for the longest time

Cache Design

- Write policy
- Write-through : All data writes sent immediately to data file
  - Pros
    - Simple to implement
    - Data in cache and file are always consistent
  - Cons
    - Slow for lots of writes

Cache Design

- Write-back
  - Writes to cache are sent to data file when cache block is evicted
  - Blocks with modified data is considered as “Dirty blocks”
  - Dirty blocks are written back to file when evicted
  - Pros
    - It is fast
  - Cons
    - Complex to implement
    - There exists data inconsistency issue if program crashes before writing dirty blocks

Programming Assignment 4

- Implement a cache simulator!
- Input parameters: **cache size, associativity, block size**
  - Cache size: power of 2
  - Block size: power of 2
- Replacement policy: **FIFO**
- Write policy: **Write through**

Programming Assignment 4

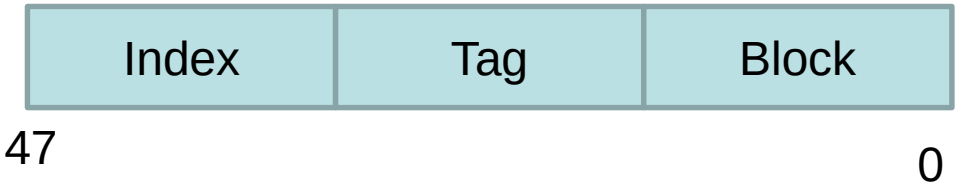
- Your program should follow
  - $ ./first <cache size> <associativity> <block size> <trace file>

- Cache size: total size of the cache (power of 2)
- Associativity
  - **direct** – simulate a direct mapped cache
  - **assoc** – simulate a fully associative cache
  - **assoc:n** – simulate an n-way associative cache (n should be power of 2)
- Block size: size of cache block (power of 2)
- Trace file: name of the trace file

Programming Assignment 4

- You should implement two types of caches
- **Type A**: uses the indexing schemes discussed in class

- **Type B**: the order of index and tag bits are reversed



Programming Assignment 4

- Your program should print out
  - Number of memory reads
  - Number of memory writes
  - Cache hits
  - Cache misses

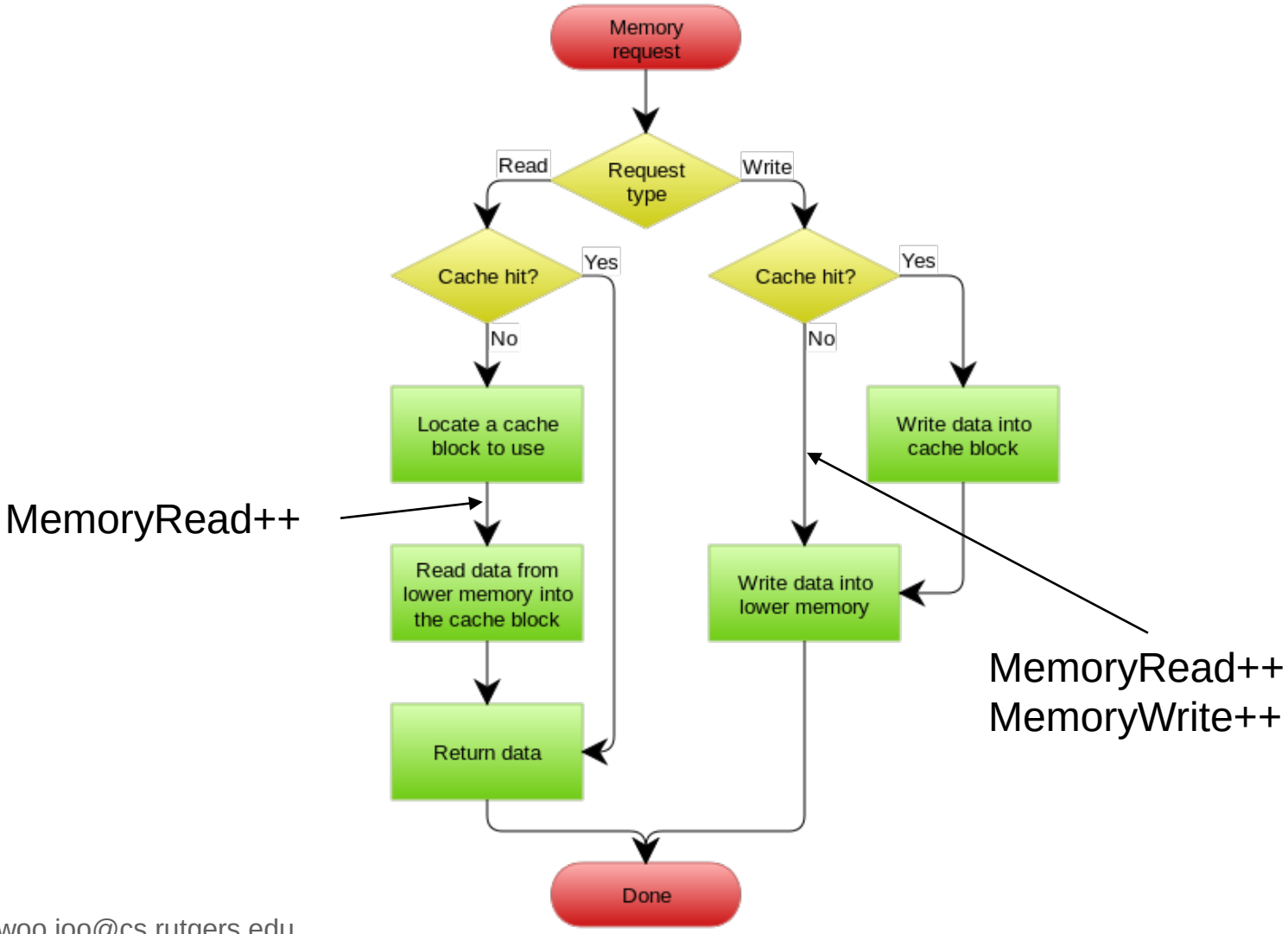
```
$./first 32 assoc:2  4 trace1.txt
cache A
Memory reads: 173
Memory writes: 334
Cache hits: 827
Cache misses: 173
cache B
Memory reads: 667
Memory writes: 334
Cache hits: 333
Cache misses: 667
```

C = S * E * B

32 = S * 2 * 4
S = 4 Bytes

Set bits : 2
Block bits : 2
Tag bits : 48 - 2 – 2 = 44

Programming Assignment 4



Q & A

- Any questions?