# CE/CZ 3001: Advanced Computer Architecture

## Module 5: Memory Systems

**- cache design**

Asst Prof Liu Weichen
School of Computer Science and Engineering
Nanyang Technological University, Singapore

# Summary of pre-recorded videos

- Memory hierarchy
- Cache design principles
- General organization of cache
- Cache placement policy
- Direct-mapped cache
- Fully associative cache
- Set associative cache

# Review: Principle of Locality

- Temporal Locality
  - a resource that is referenced at one point in time will be referenced again sometime in the near future
- Spatial Locality
  - the likelihood of referencing a resource is higher if a resource near it was just referenced
- 90/10 Locality Rule of Thumb
  - a program spends 90% of its execution time in only 10% of its code
    - a consequence of how we program and store data in the memory
    - hence, it is possible to predict with reasonable accuracy what instructions and data a program will use in the near future based on its accesses in the recent past

# Cache Concepts

- The term "Cache"
  - the first level of the memory hierarchy (from the CPU)
  - often used to refer to any buffering technique exploiting the principle of locality

- Directly exploits temporal locality providing faster access to a smaller subset of the main memory which contains copy of data recently used

- When a cache miss occurs, a fixed-size block of contiguous memory cells is retrieved from the main memory based on the principle of spatial locality

# Example

```
for ( i= 0 ; i< 20 ; i++)
        for( j=0; j<10; j++)
            a[i]=a[i]*j;
```

- Temporal locality: for the inner loop, it accesses memory a[i] repeatedly in each iteration.

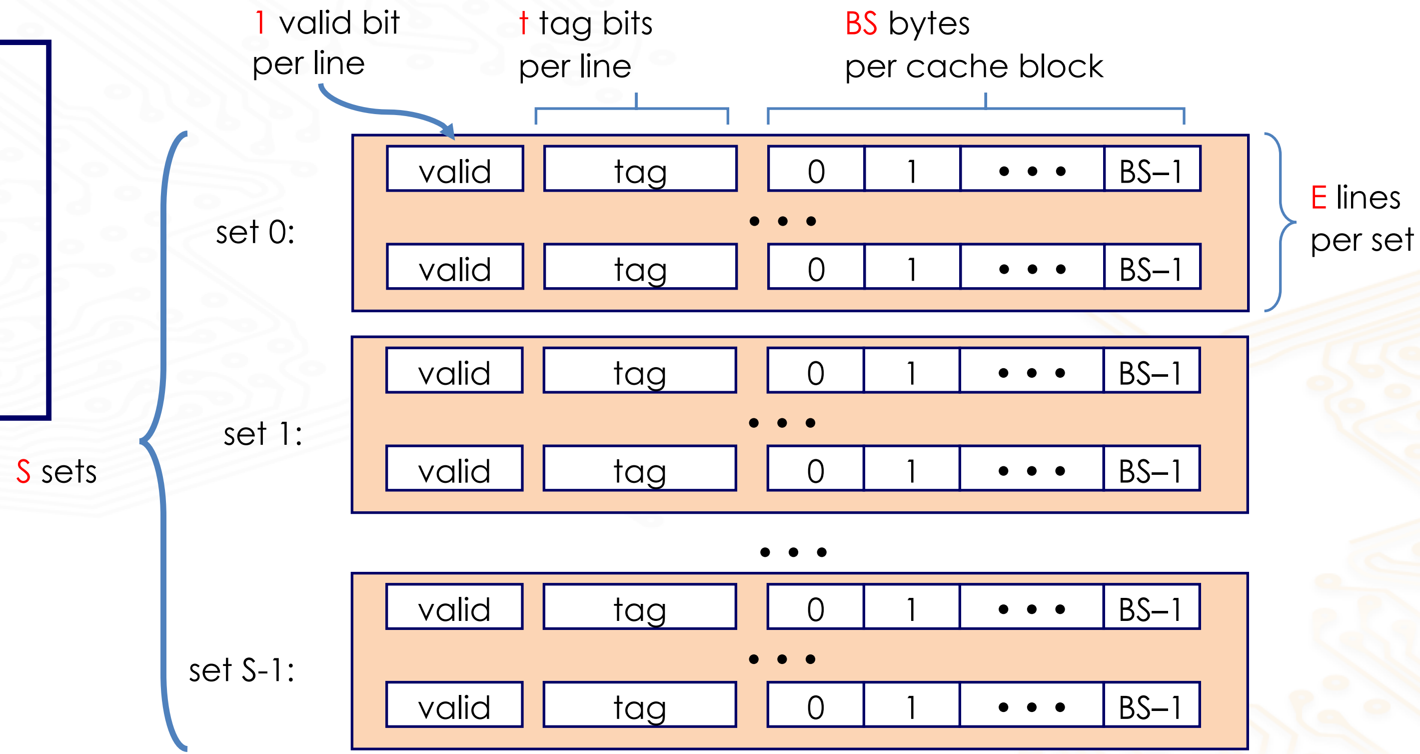
- Spatial locality: for the outer loop, it accesses memory a[0], a[1], …,a[20] sequentially.

# General Organization of a Cache (Part 1/3)

Cache is an array of sets

Each set contains one or more lines

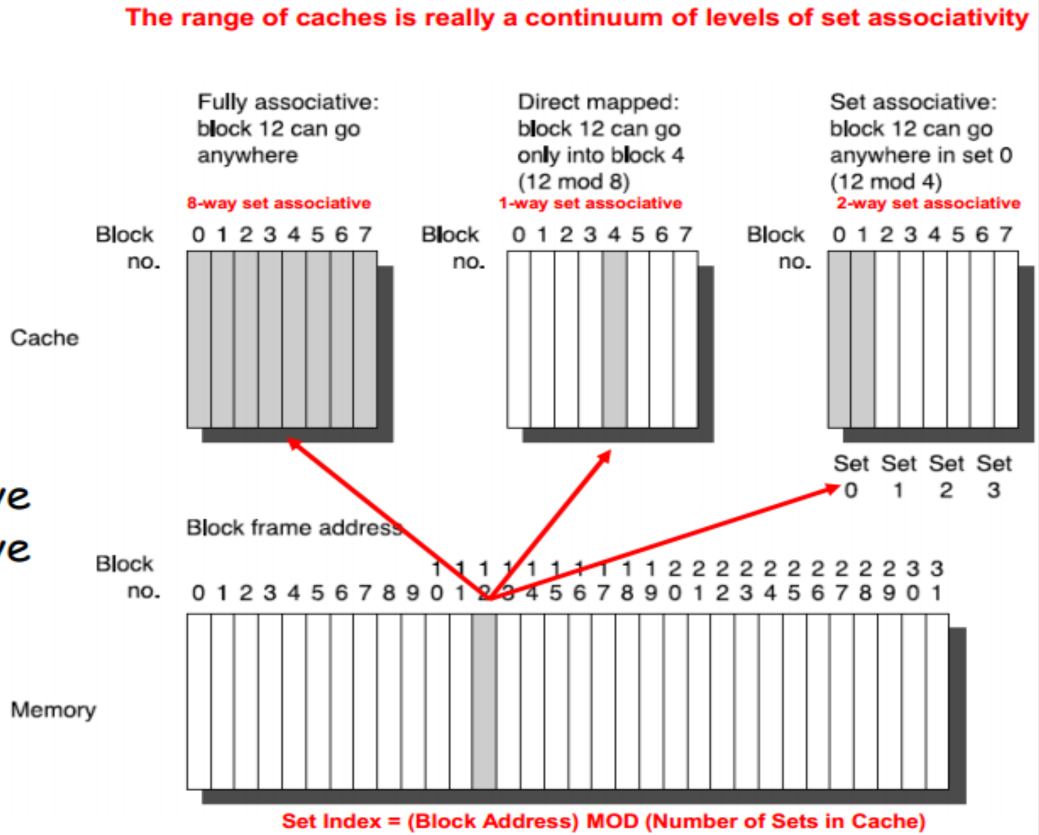
Each line holds a block of data




**Cache size: C = (1 bit + t bits + BS bytes) x E x S**

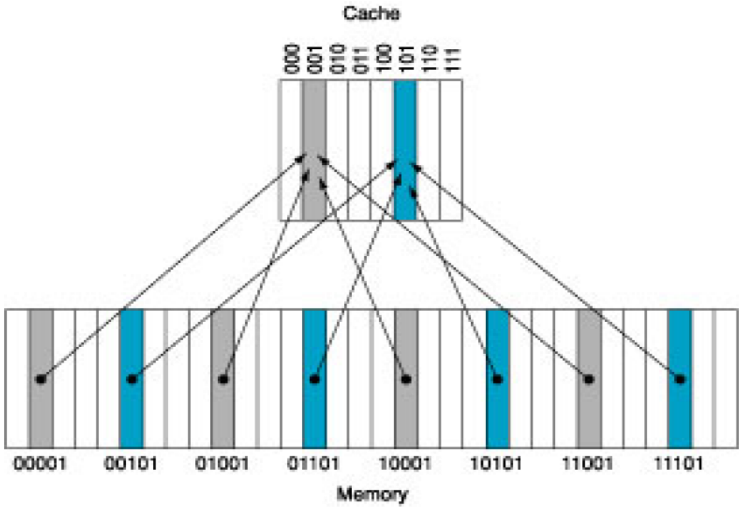
# Cache Organization Schemes: Placing a Memory Block into Cache

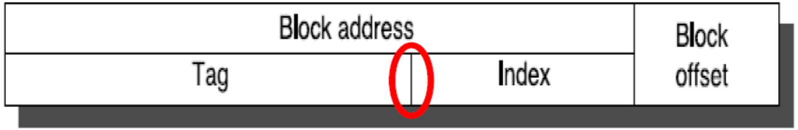
- Block
  - unit of memory transferred across hierarchy levels
- Set
  - a group of blocks
- Modern processors
  - direct map
  - 2-way set associative
  - 4-way set associative
- Modern memories
  - millions of blocks
- Modern caches
  - thousands of block frames



# Example: Direct Mapped Cache with 8 Blocks

- Each memory block is mapped to one cache entry
  - cache index = (block address) mod (# of cache blocks)
  - e.g., with 8 blocks, 3 low-order address bits are sufficient
    - Log2 (8) = 3
- Is a block present in cache?
  - must check cache *block tag*
    - upper bit of block address
- Block offset
  - addresses bytes in a block
    - block==word ⇒ offset =2 bits
- How do we know if data in a block is valid?
  - add valid bit to each entry




**The tag index boundary moves to the right as we increase associativity (no index field in fully associative caches)**

# Example: Measuring Cache Size

- How many total bits are required for a direct-mapped cache with 16KB of data and 4-word block frames assuming a 32-bit address?
- 16KB of data = 4K words = $2^{12}$ words
- Block Size of 4 $(=2^{2})$ words $\Rightarrow$ $2^{10}$ blocks

| TAG | INDEX | OFFSET | |
|---|---|---|---|
| 18 | 10 | 2 | 2 |

- # Bits in a Tag = 32 - (10 + 2 + 2) = 18
- # Bits in a block = # Tag Bits + # Data Bits + Valid bit
- # Bits in a block = 18 + (4 * 32) + 1 = 147
- Cache Size= # Blocks x #Bits in a block= $2^{10}$ x 147=147Kbits
- Cache Overhead = 147Kbits / 16KB = 147 / 128 = 1.15

So, 15% overhead.