RISC Design
Memory System

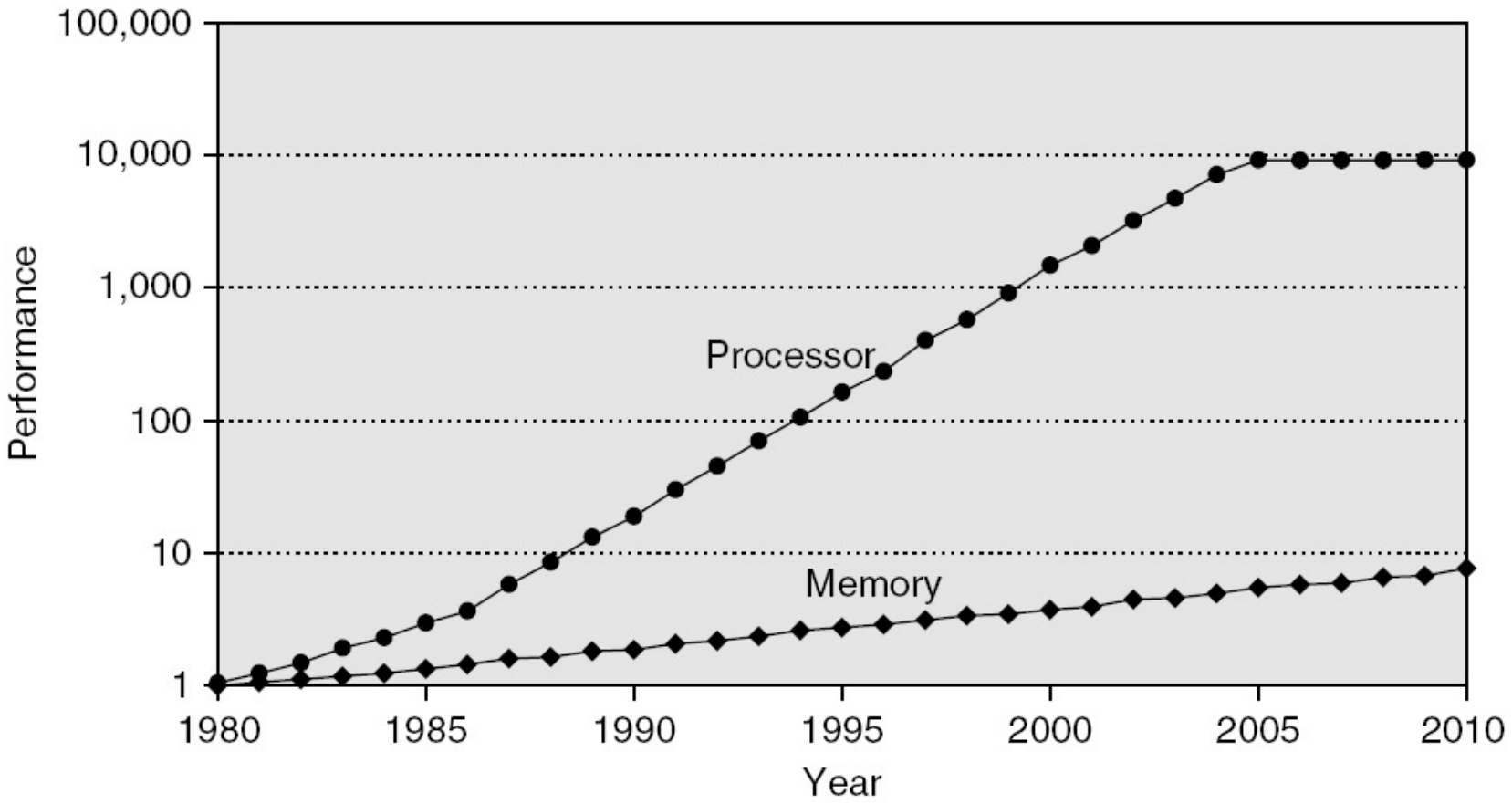
Virendra Singh
Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

EE-739: Processor Design

Lecture 26 (30 March 2021)

CADSL

Memory Performance Gap



Why Memory Hierarchy?

- Need lots of bandwidth

$$BW = \frac{1.0 inst}{cycle} \times \left[\frac{1 Ifetch}{inst} \times \frac{4B}{Ifetch} + \frac{0.2 Dref}{inst} \times \frac{4B}{Dref} \right] \times \frac{1 Gcycles}{\text{sec}}$$
$$= \frac{4.8 GB}{\text{sec}}$$

- Need lots of storage
  - 64MB (minimum) to multiple TB
- Must be cheap per bit
  - (TB x anything) is a lot of money!
- These requirements seem incompatible

Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/second +
      - 12.8 billion 128-bit instruction references
      - = 409.6 GB/s!
    - DRAM bandwidth is only 6% of this (25 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

Why Locality?

- Analogy:
  - Library (Disk)
  - Bookshelf (Main memory)
  - Stack of books on desk (off-chip cache)
  - Opened book on desk (on-chip cache)
- Likelihood of:
  - Referring to same book or chapter again?
    - Probability decays over time
    - Book moves to bottom of stack, then bookshelf, then library
  - Referring to chapter n+1 if looking at chapter n?

Why Does a Hierarchy Work?

- Locality of reference
  - Temporal locality
    - Reference same memory location repeatedly
  - Spatial locality
    - Reference near neighbors around the same time
- Empirically observed
  - Significant!
  - Even small local storage (8KB) often satisfies >90% of references to multi-MB data set

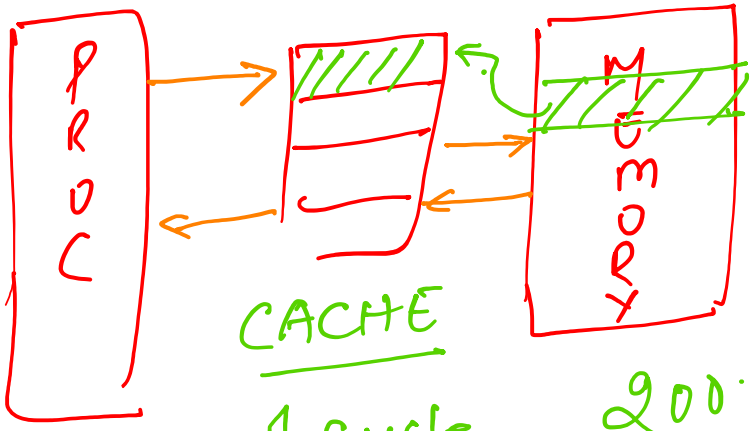
Four Burning Questions

- These are:
  - Placement
    - Where can a block of memory go?
  - Identification
    - How do I find a block of memory?
  - Replacement
    - How do I make space for new blocks?
  - Write Policy
    - How do I propagate changes?
- Consider these for caches
  - Usually SRAM
- Will consider main memory, disks later

Placement

Memory Type	Placement	Comments
Registers	Anywhere; Int, FP, SPR	Compiler/programmer manages
Cache (SRAM)	Fixed in H/W	*Direct-mapped, set-associative, fully-associative*
DRAM	Anywhere	O/S manages
Disk	Anywhere	O/S manages

HUH?



1 KB

1. Where to place?
2. How to identify
3. Whom to evict
4. When to write in memory

Reading – same image in RAM (Memory)?

CACHE

Write –

1 KB → 10 bits to address $\frac{1024}{16}$

Chunk – 16 bytes → Block

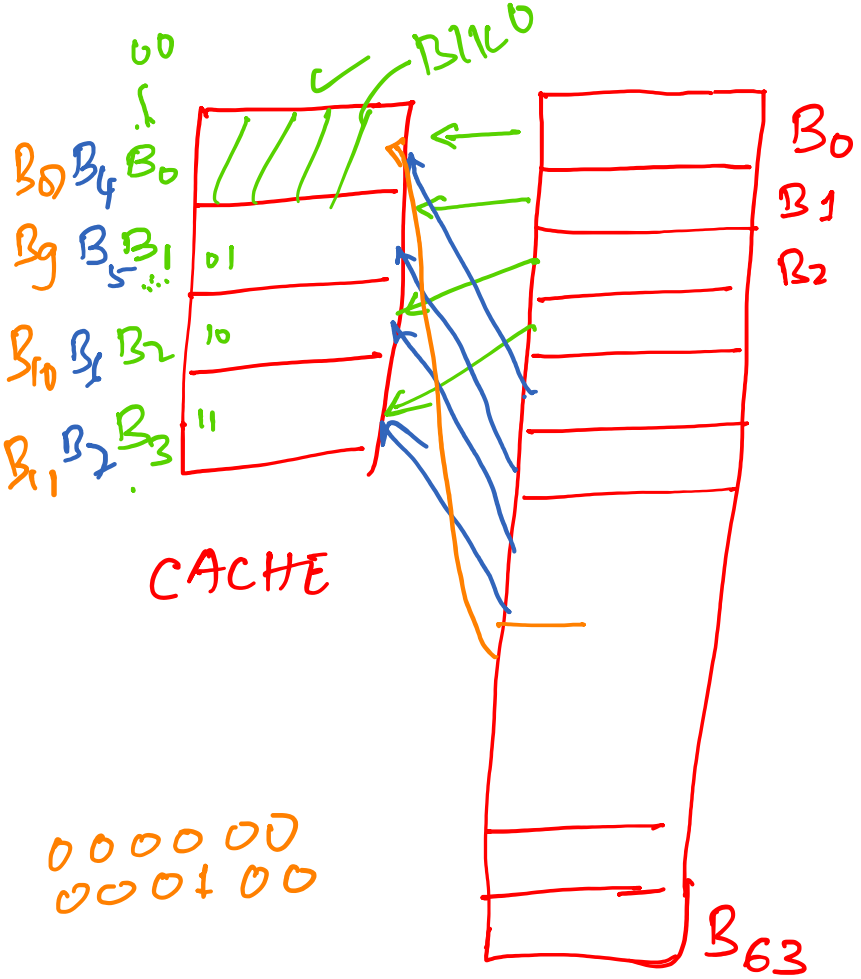
4 bit

Block no

offset within block

Blocks = 64) $B_{9} - B_{0}$

PLACEMENT POLICY



Map Sequentially

$B_0 - B_{63}$ to $B_0 - B_4$.

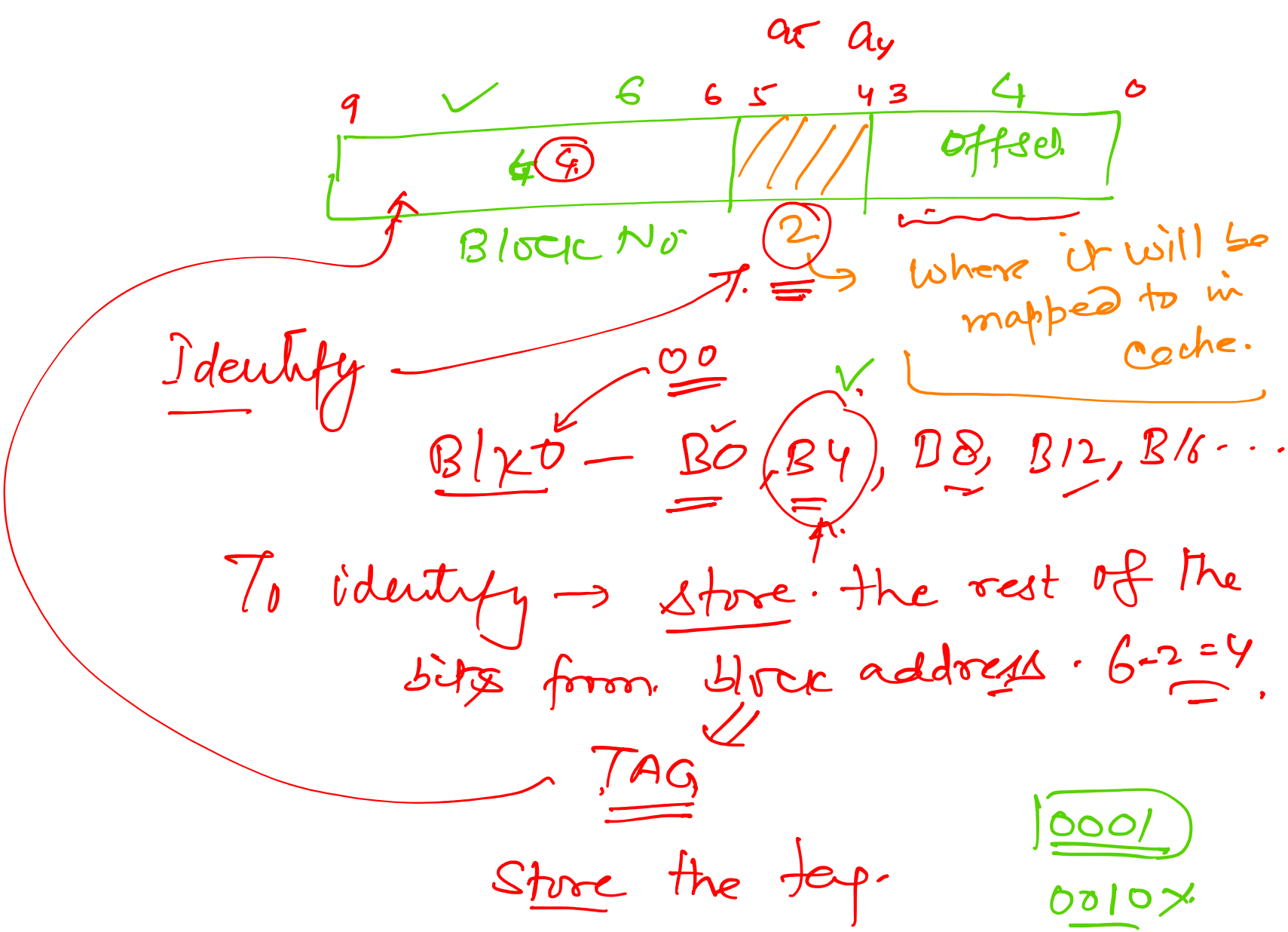
Blk 0 → B0, B4, B8, B12, B16, B20, B24...

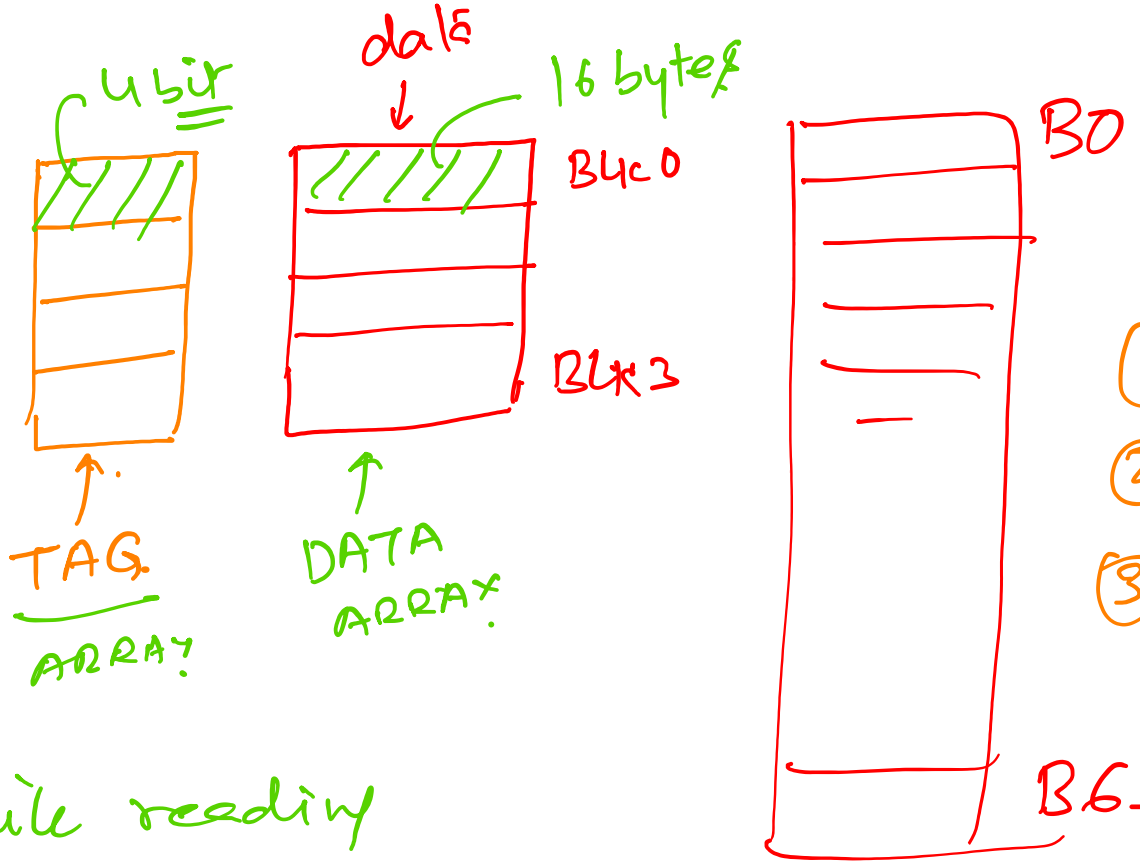
↓

every fourth block gets mapped to Blk 0,

Block No % 4.

Blk No = Block No. % 4.


a5 a4
9
6
6 5
4 3
4
0
Offset
Block No
2
where it will be mapped to in cache.
Identify
00
Blk 0 — B0 B4, D8, B12, B16 ...
To identify → store the rest of the bits from block address. 6-2 = 4.
TAG
Store the tag.
0001
0010



delay

(1) decoder delay

(2) Tag match delay

(3) Data read delay

While reading

(1) Identify cache block (bits of Block No).

(2) match the tag (bits of Block no.)

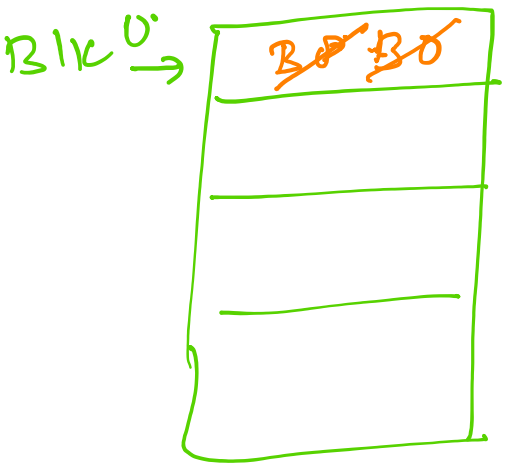
(3) If match then read the from data array

Blk 0 — B0, B4, B8, B12 - - -

```
for (i=0; i<100; i++)
  if (i%2)
    {
      g = g+1.
    }
  else
    f = f-1
}
```

B8

B0

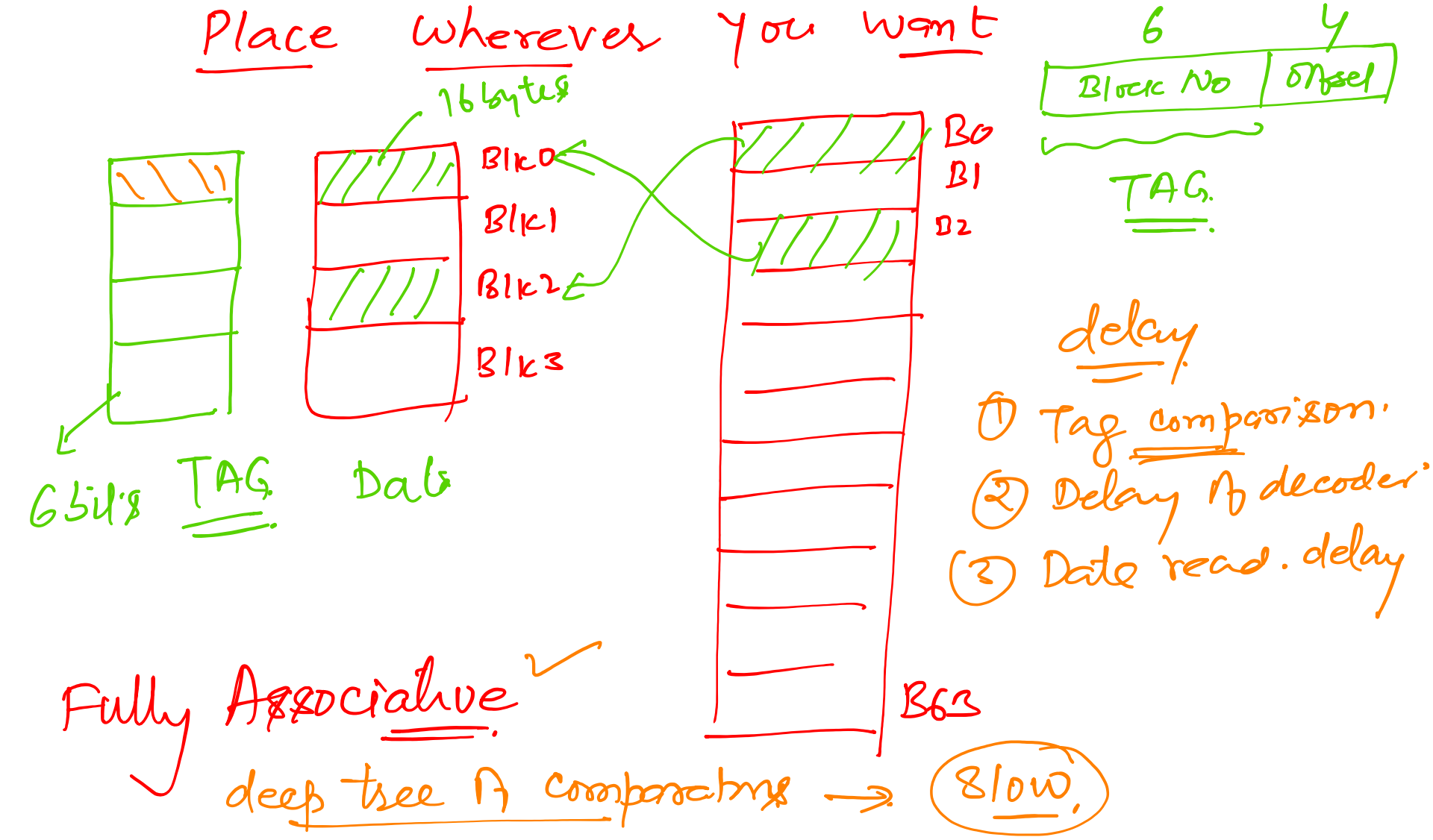


Strict mapping.

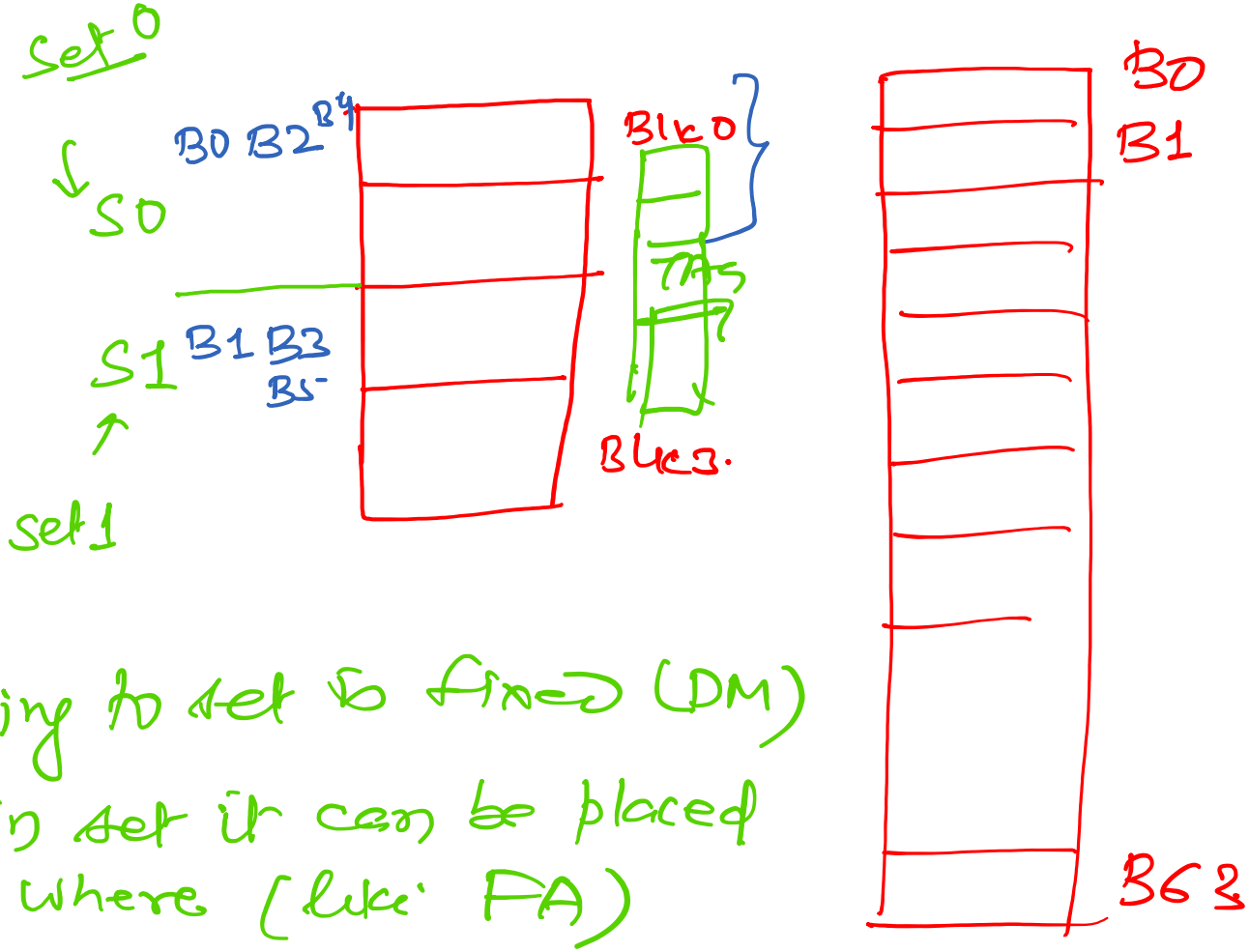
DIRECT MAPPINGS.

Simple.

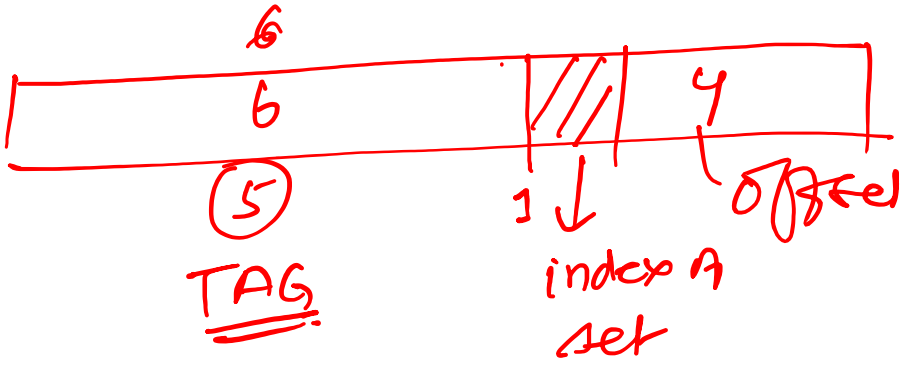
- not very efficient


Place wherever you want
16 bytes
Blk0
Blk1
Blk2
Blk3
B0
B1
B2
B63
6
4
Block No
Offset
TAG.
delay
① Tag comparison.
② Delay of decoder
③ Data read. delay
64 blks
TAG
Data
Fully Associative
deep tree & comparators → slow.

Set Associative Mapping

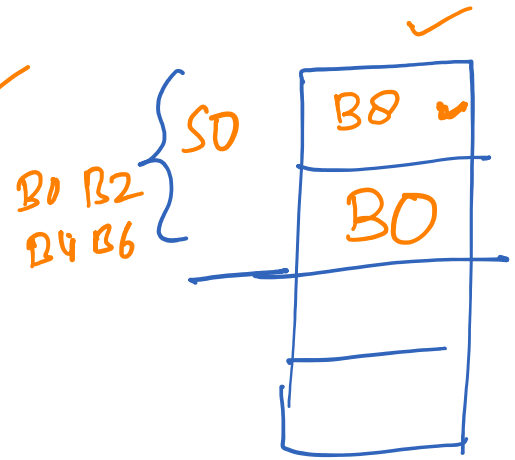


- Mapping to set is fixed (DM)
- within set it can be placed any where (like FA)




```
for (i=0; i<100; i++)
  if (i%2)
  then
    {
      j = j+1   // B8
    }
  else
    {
      j = j-1   // B0
    }
```

Only 2 tags (in this case) has to be matched in parallel.



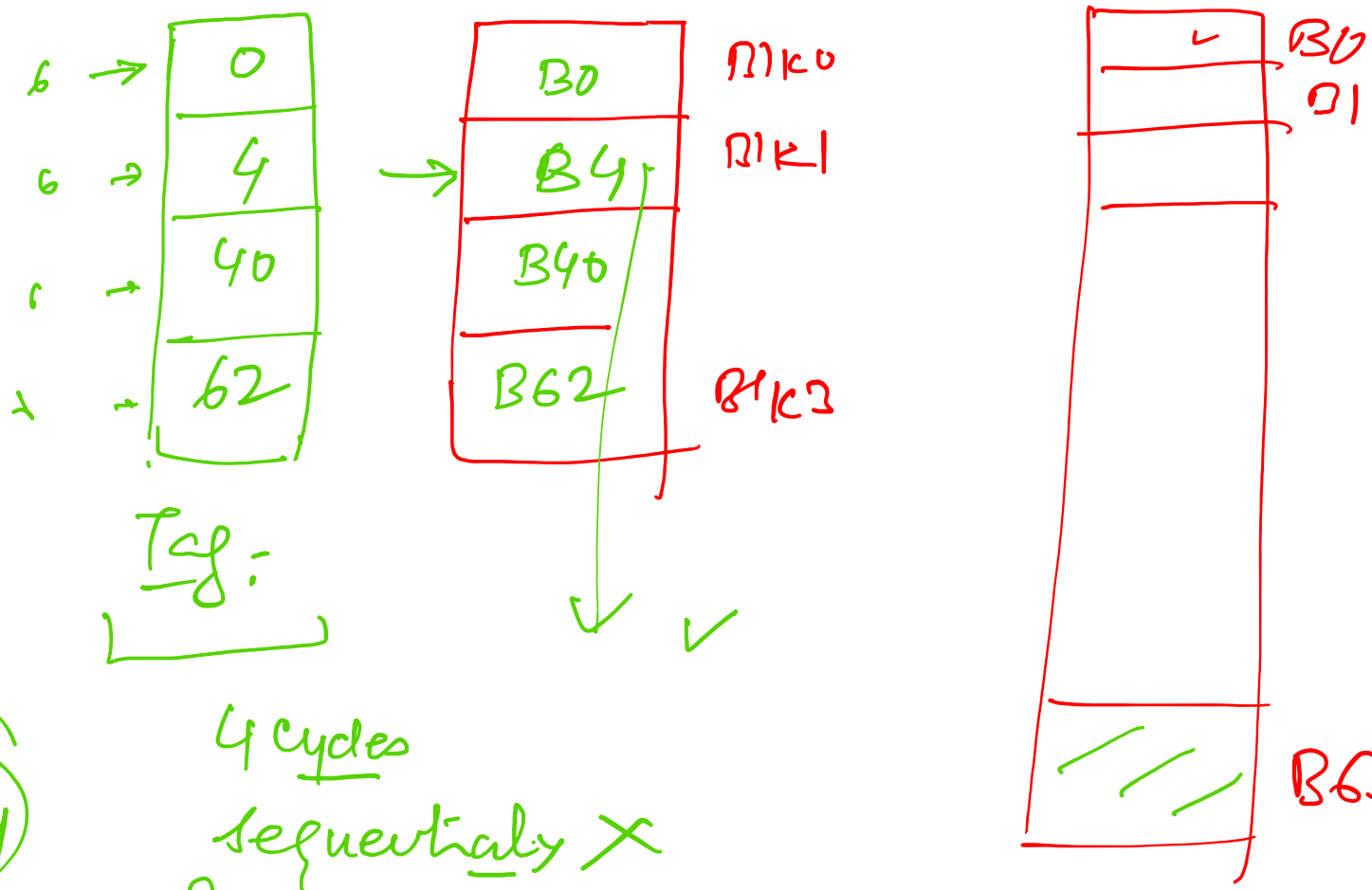
How to place & identify

(1) Direct mapped — fast / inflexible

(2) Fully Associative — slow / efficient.

(3) Set Associative ✓

FULLY ASSOCIATIVE



B4

4 cycles sequentialy ✗

Parallel ✓

Aim → Access in single cycle.

Thank You