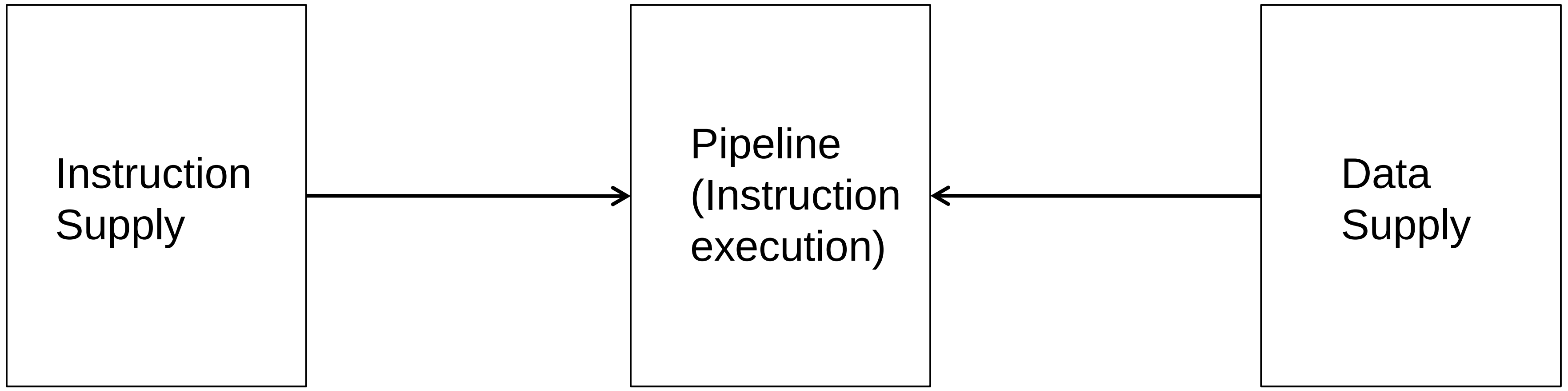
Lecture-13 and 14 (Memory Hierarchy)
CS422-Spring 2020

Biswa@CSE-IITK

The Ideal World



- Zero-cycle latency
- Infinite capacity
- Zero cost
- Perfect control flow

- No pipeline stalls
-Perfect data flow (reg/memory dependencies)
- Zero-cycle interconnect (operand communication)
- Enough functional units
- Zero latency compute

- Zero-cycle latency
- Infinite capacity
- Infinite bandwidth
- Zero cost

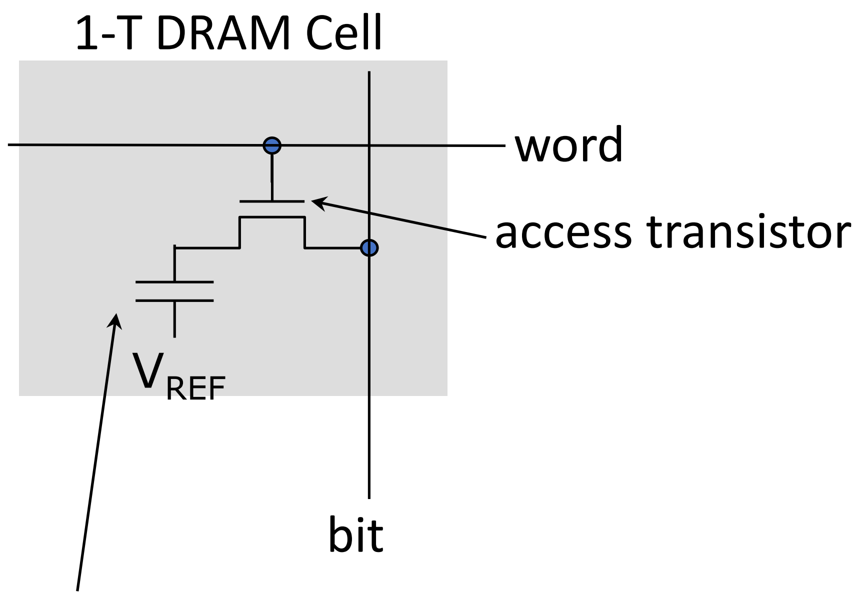
Semiconductor Memory

- Semiconductor memory began to be competitive in early 1970s
  - Intel formed to exploit market for semiconductor memory
  - Early semiconductor memory was Static RAM (SRAM). SRAM cell internals similar to a latch (cross-coupled inverters).

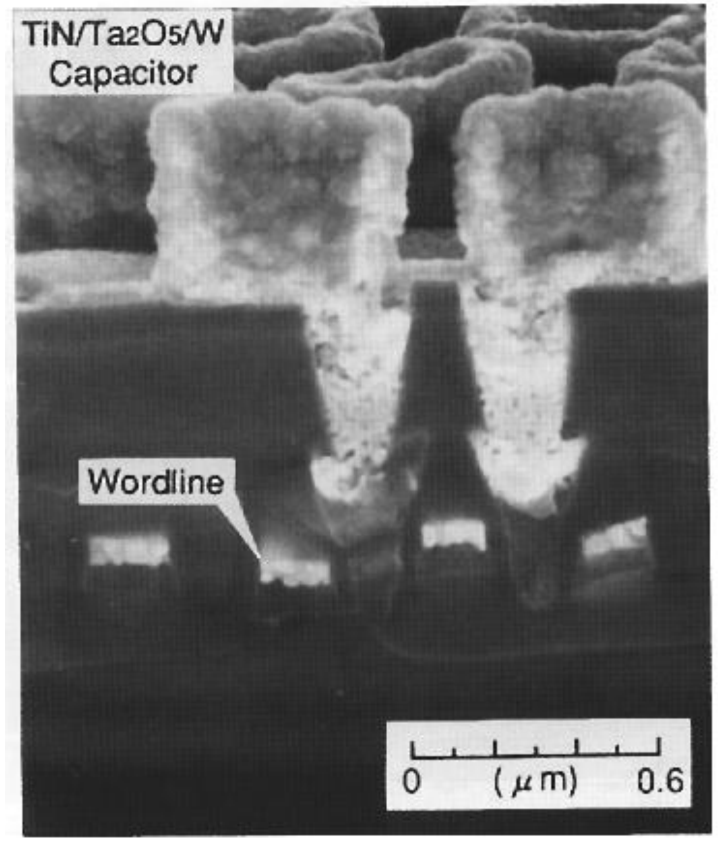
- First commercial Dynamic RAM (DRAM) was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value

Semiconductor memory quickly replaced core in ‘70s

One-transistor DRAM



Storage
capacitor (FET gate, trench, stack)



DRAM vs SRAM

- DRAM
  - Slower access (capacitor)
  - Higher density (1T 1C cell)
  - Lower cost
  - Requires refresh (power, performance, circuitry)

- SRAM
  - Faster access (no capacitor)
  - Lower density (6T cell)
  - Higher cost
  - No need for refresh

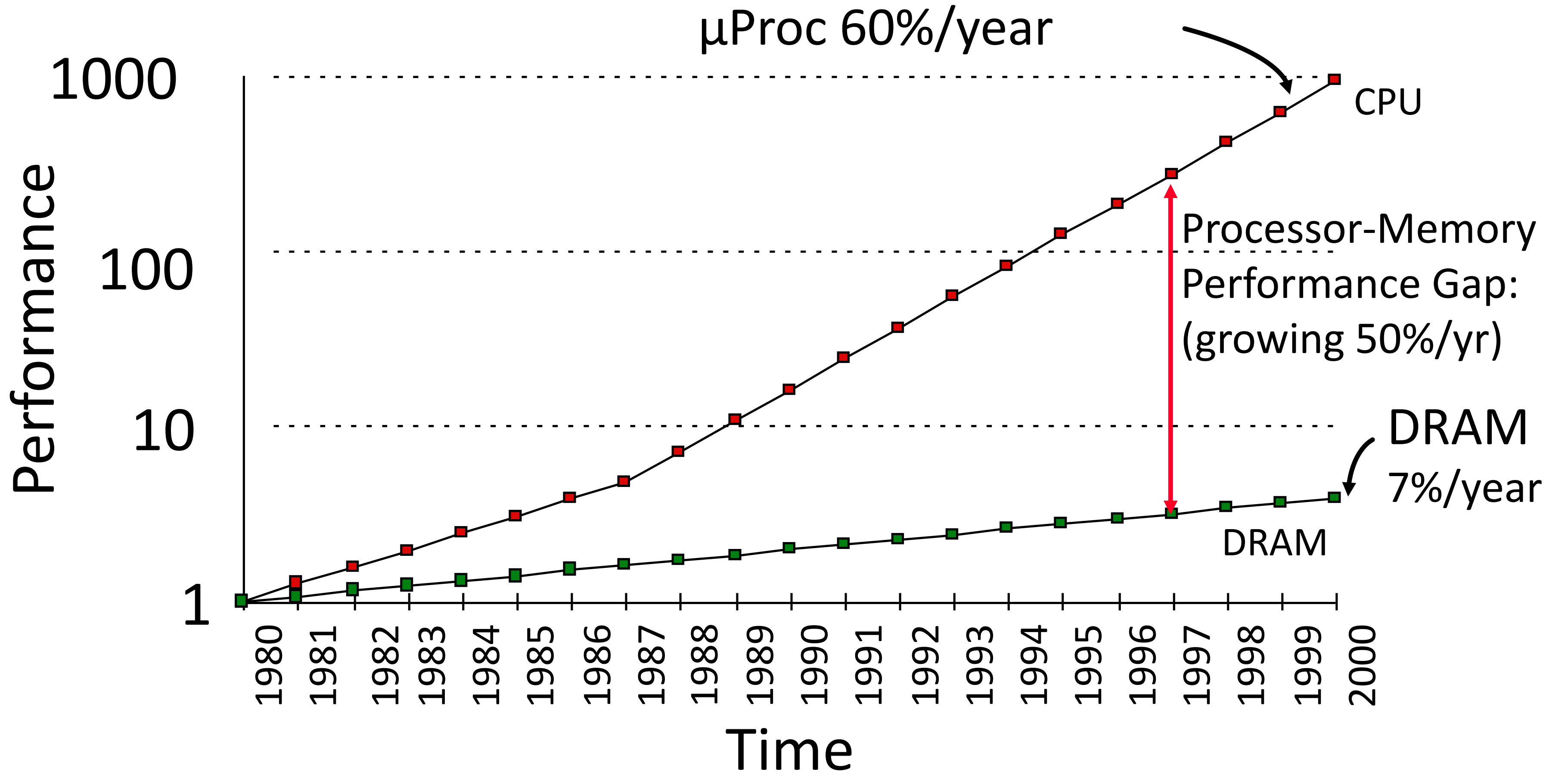
The Problem?

- Bigger is slower
  - SRAM, 512 Bytes, sub-nanosec
  - SRAM, KByte~MByte, ~nanosec
  - DRAM, Gigabyte, ~50 nanosec
  - Hard Disk, Terabyte, ~10 millisec
- Faster is more expensive (dollars and chip area)
  - SRAM, < 10$ per Megabyte
  - DRAM, < 1$ per Megabyte
  - Hard Disk < 1$ per Gigabyte
  - These sample values scale with time
- Other technologies have their place as well
  - Flash memory, PC-RAM, MRAM, RRAM (not mature yet)

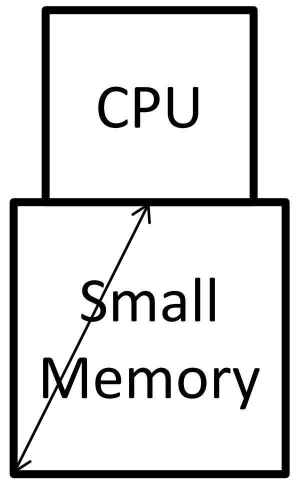
Why Memory Hierarchy?

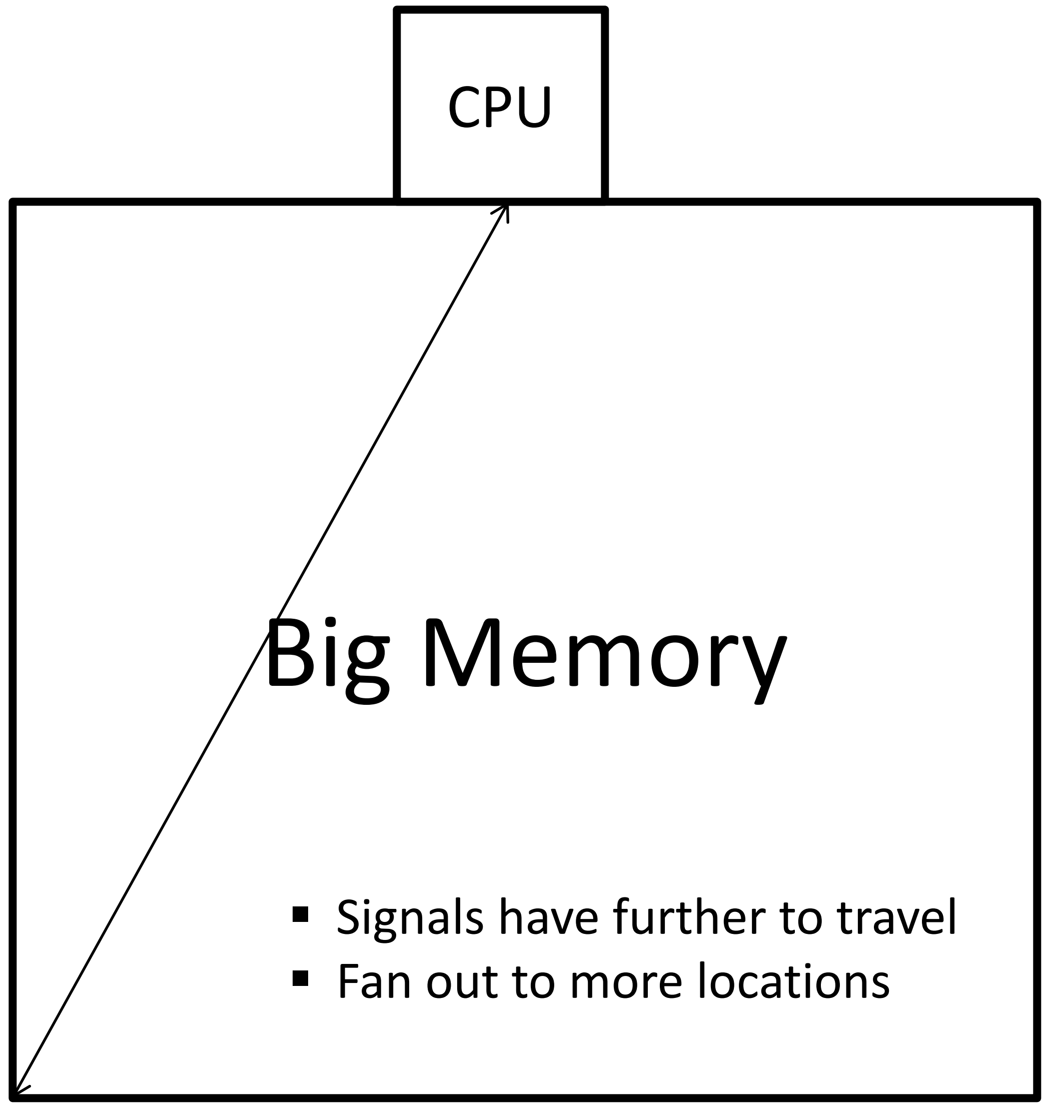
- We want both fast and large
- But we cannot achieve both with a single level of memory
- Idea: Have multiple levels of storage (progressively bigger and slower as the levels are farther from the processor) and ensure most of the data the processor needs is kept in the fast(er) level(s)

Memory Wall Problem



Size Affects Latency

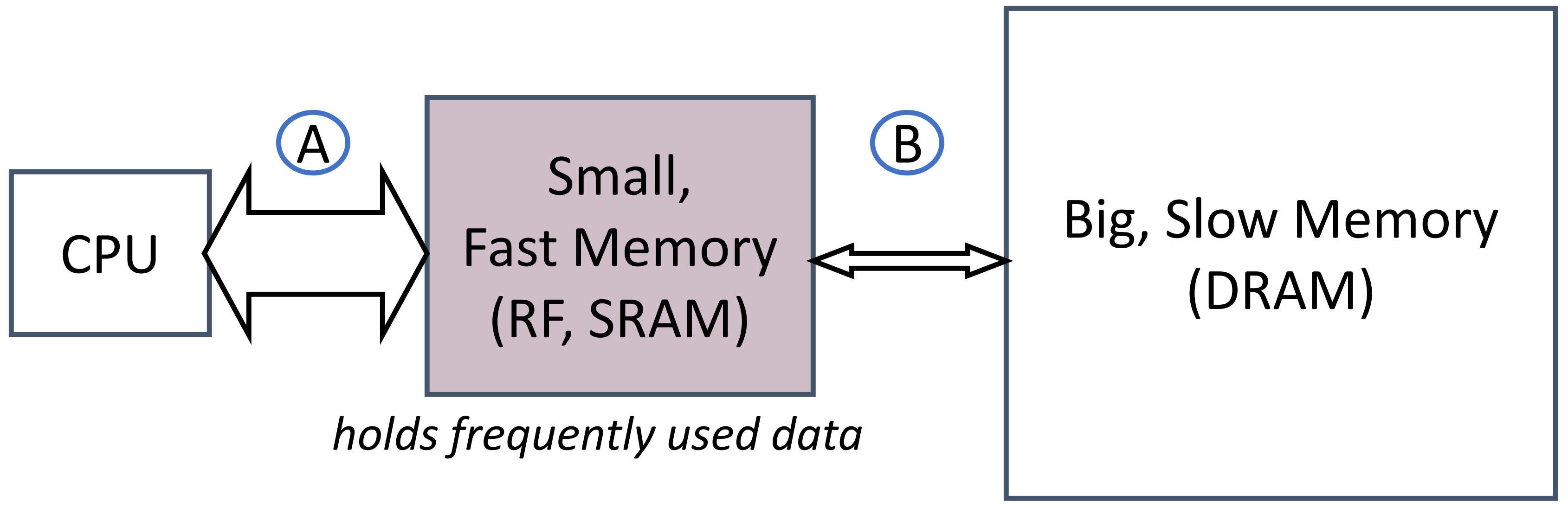




- Signals have further to travel
- Fan out to more locations

Motivates 3D stacking

Memory Hierarchy



holds frequently used data

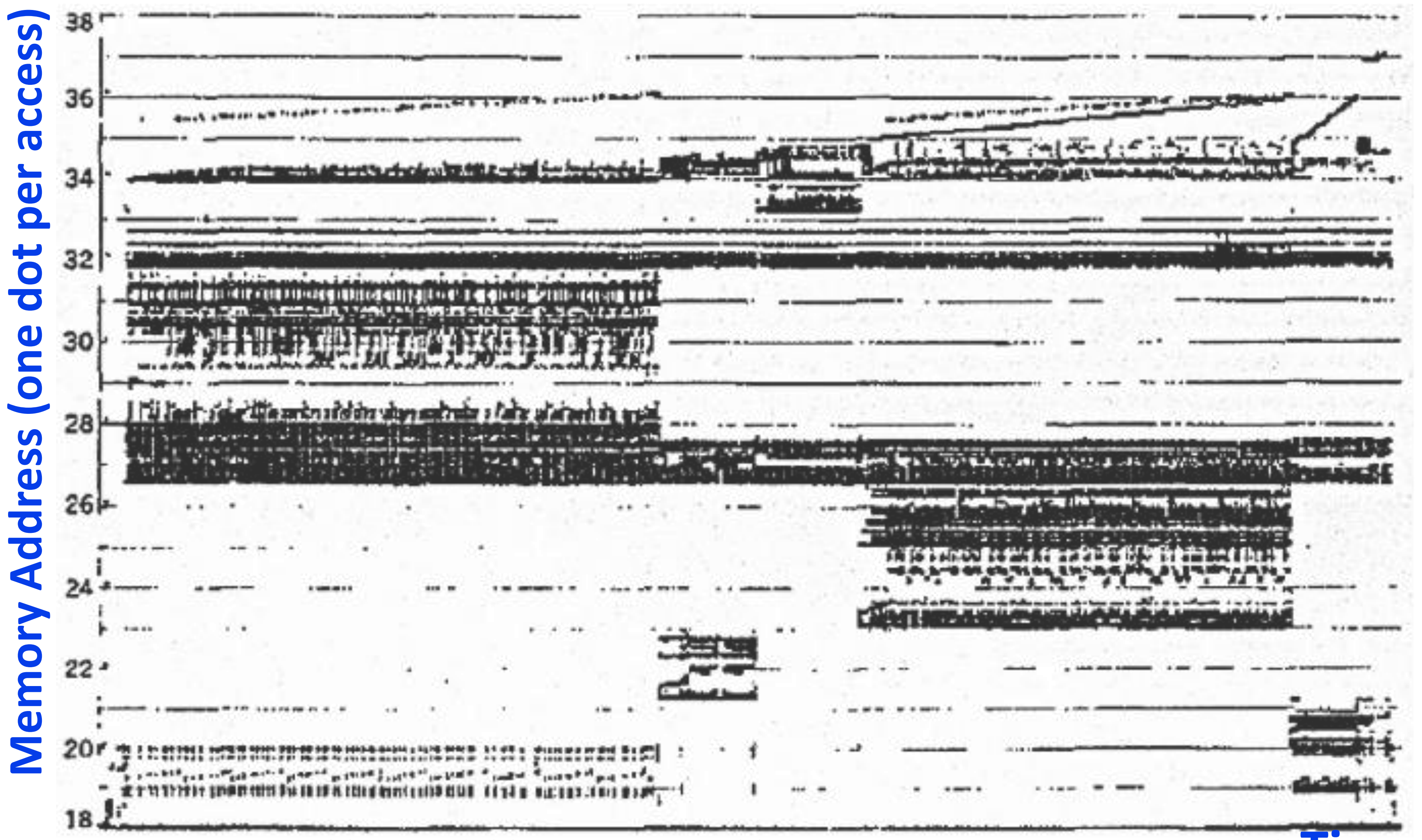
- *capacity*: Register << SRAM << DRAM
- *latency*: Register << SRAM << DRAM
- *bandwidth:* on-chip >> off-chip

On a data access:

if data $\in$ fast memory $\Rightarrow$ low latency access *(SRAM)*

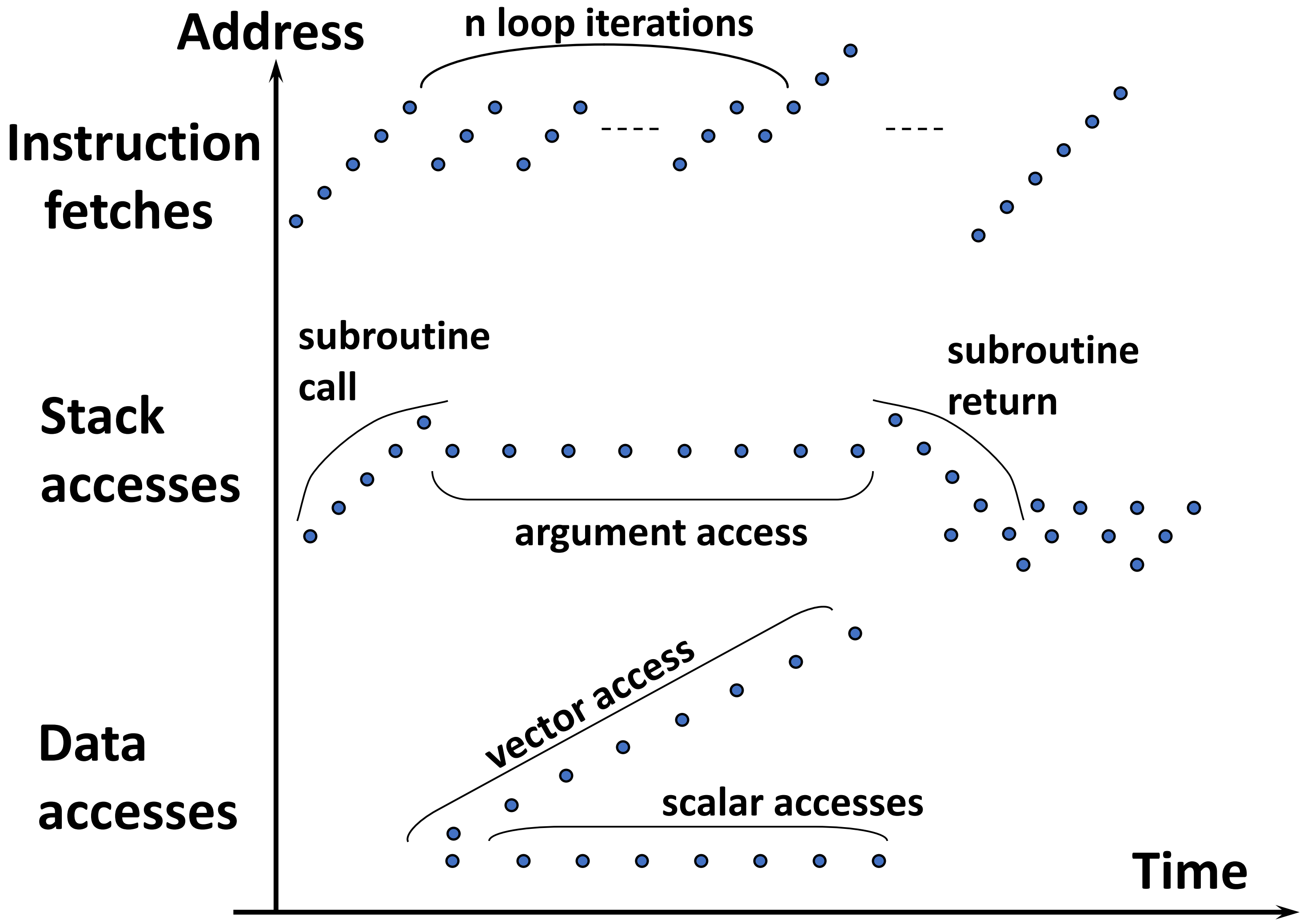
if data $\notin$ fast memory $\Rightarrow$ high latency access *(DRAM)*

Access Patterns



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Examples

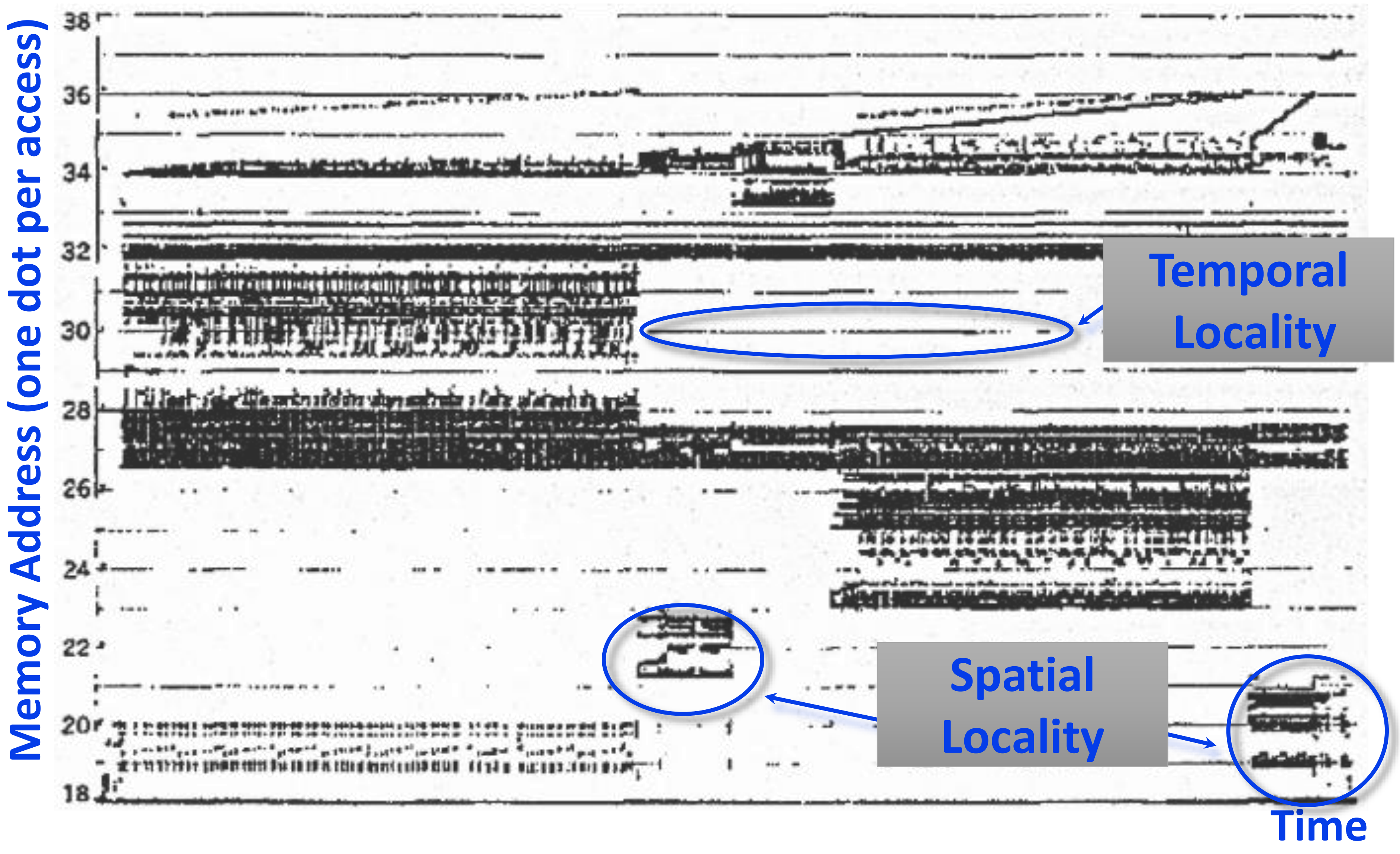

Address
n loop iterations
Instruction fetches
- - - -
- - - -
subroutine call
subroutine return
Stack accesses
argument access
vector access
Data accesses
scalar accesses
Time

Locality of Reference

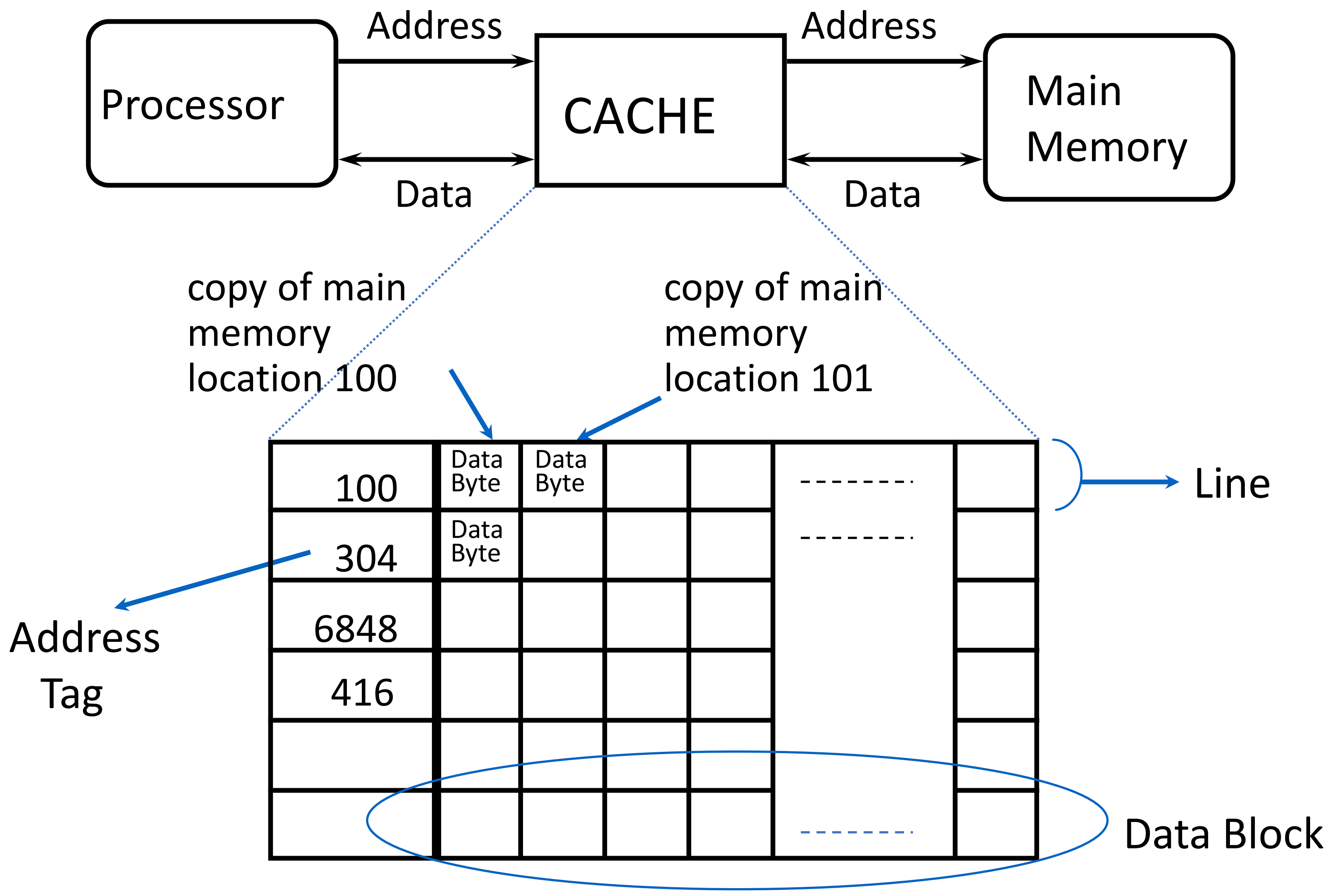
- **Temporal Locality**: If a location is referenced it is likely to be referenced again in the near future.

- **Spatial Locality**: If a location is referenced it is likely that locations near it will be referenced in the near future.

Again


Memory Address (one dot per access)
38
36
34
32
30
28
26
24
22
20
18
Temporal Locality
Spatial Locality
Time

Inside a Cache



Cache Events

Look at Processor Address, search cache tags to find match. Then either

Found in cache
a.k.a. HIT

Return copy of data from cache

Not in cache
a.k.a. MISS

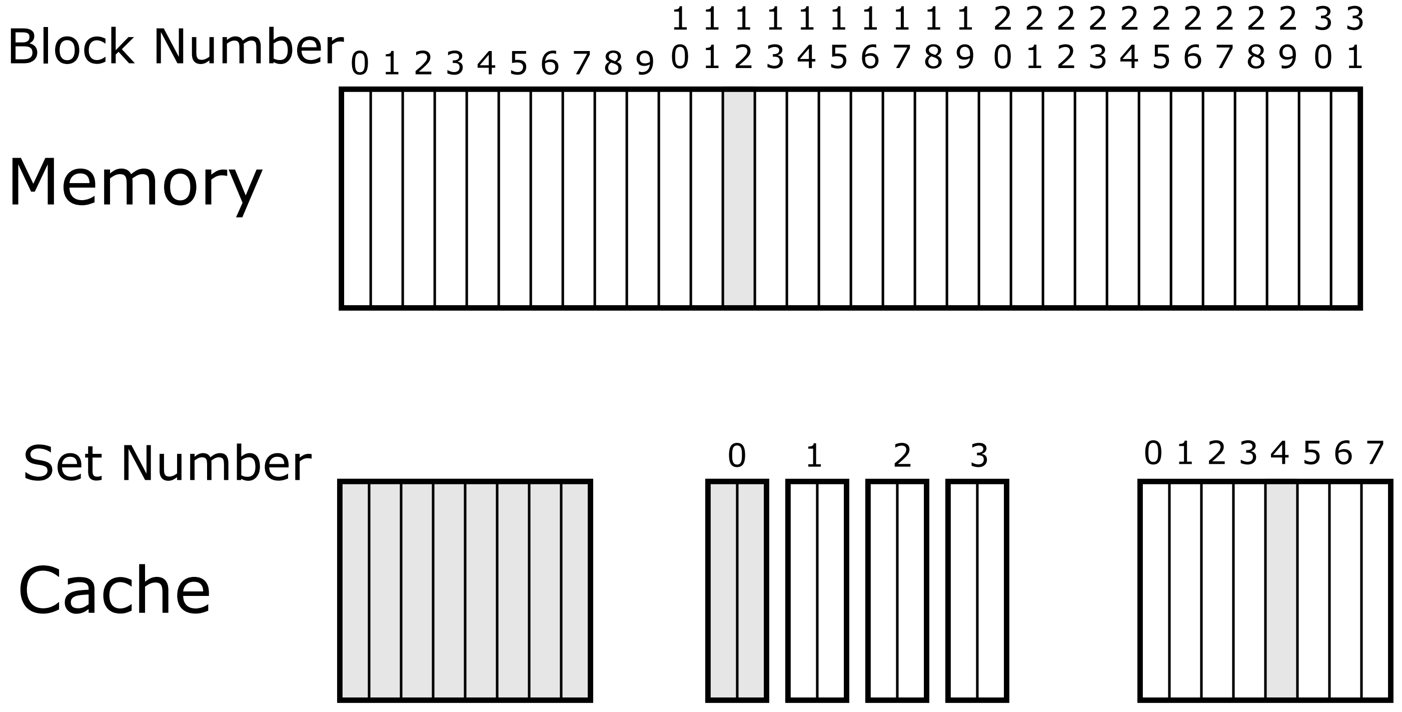
Read block of data from Main Memory

Wait …

Return data to processor and update cache

Q: Which line do we replace?

Placement Policy



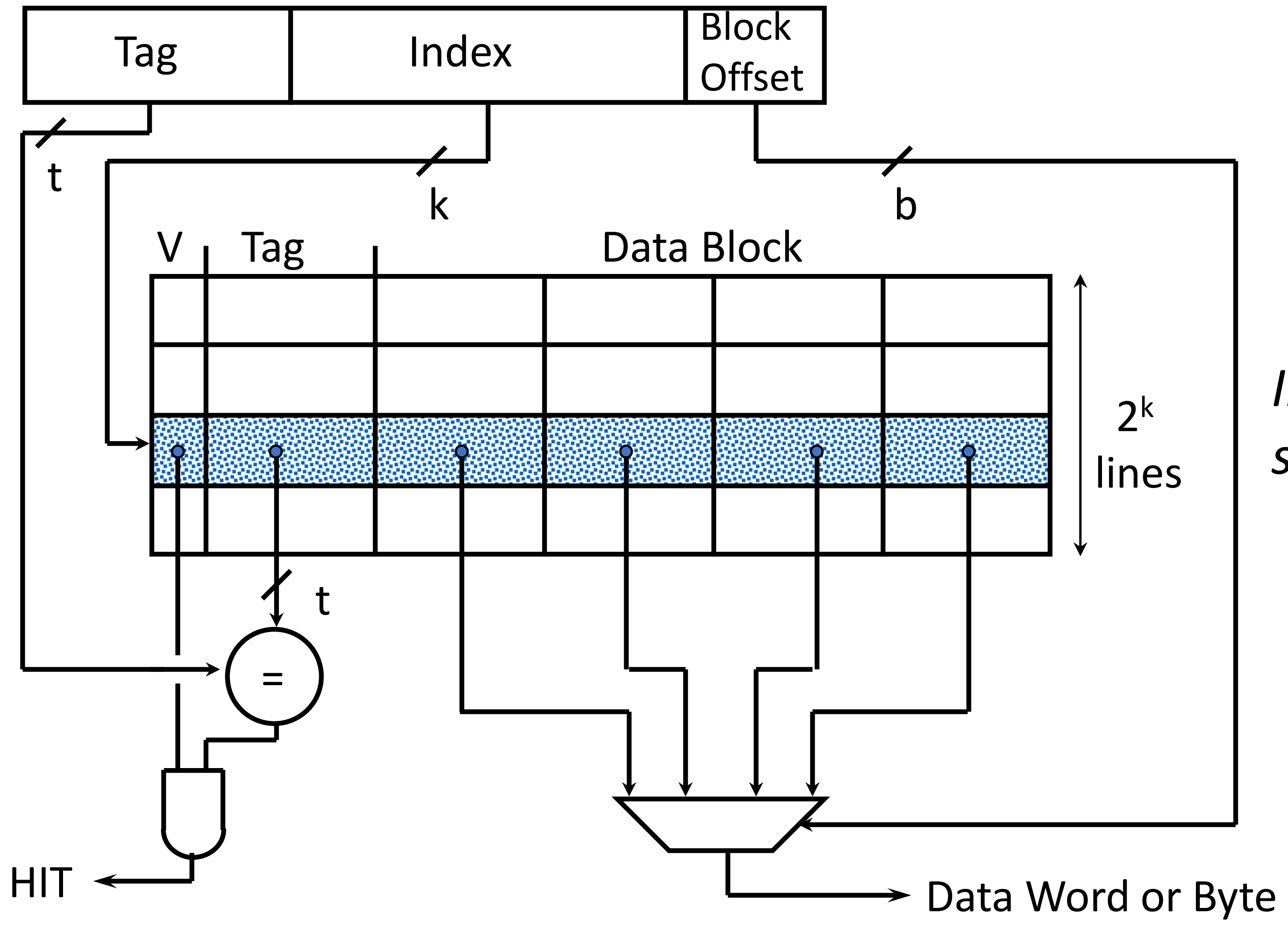
Fully Associative
anywhere

(2-way) Set Associative
anywhere in set 0
(12 mod 4)

Direct Mapped
only into block 4
(12 mod 8)

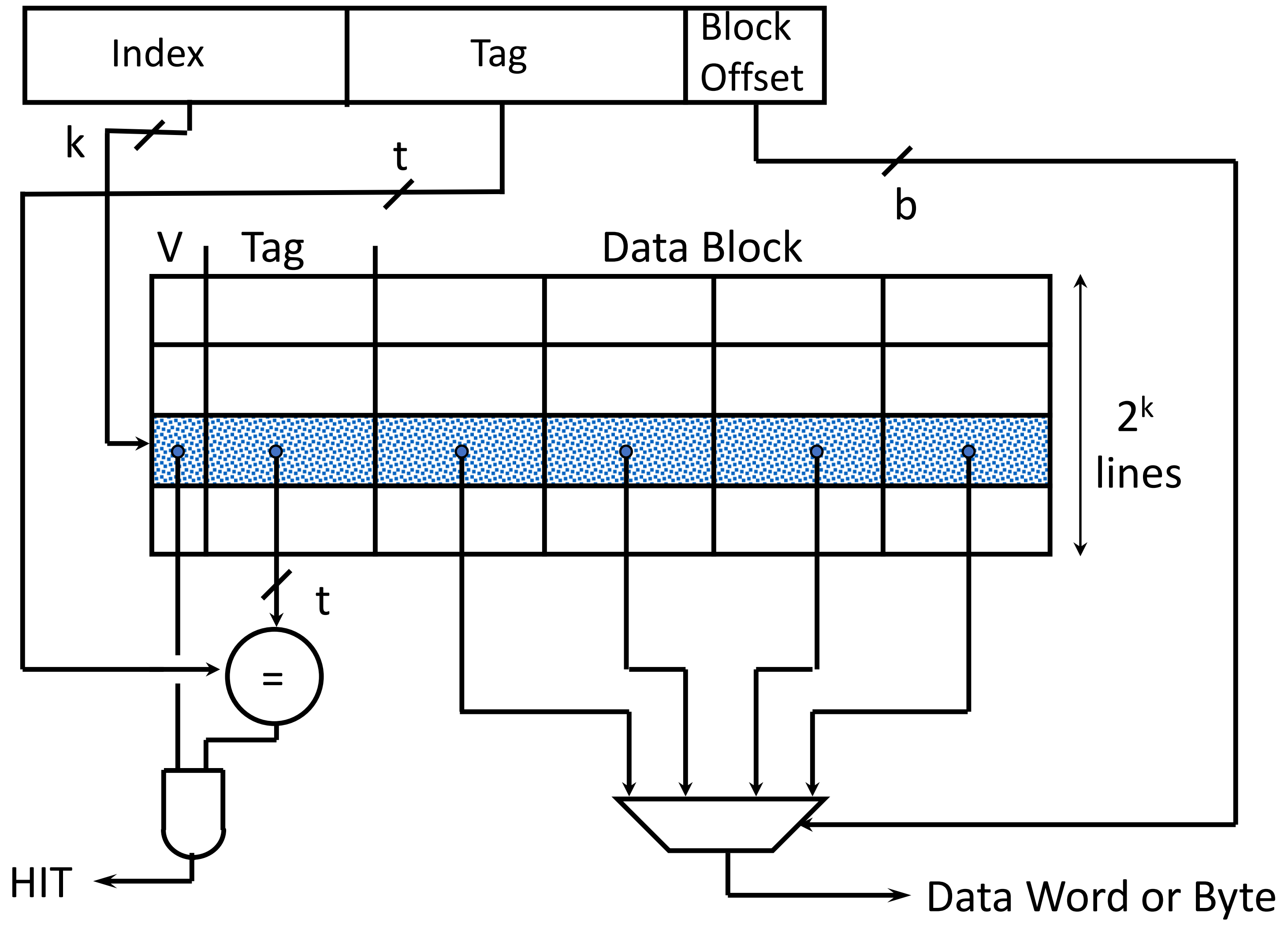
block 12
can be placed

Direct Mapped

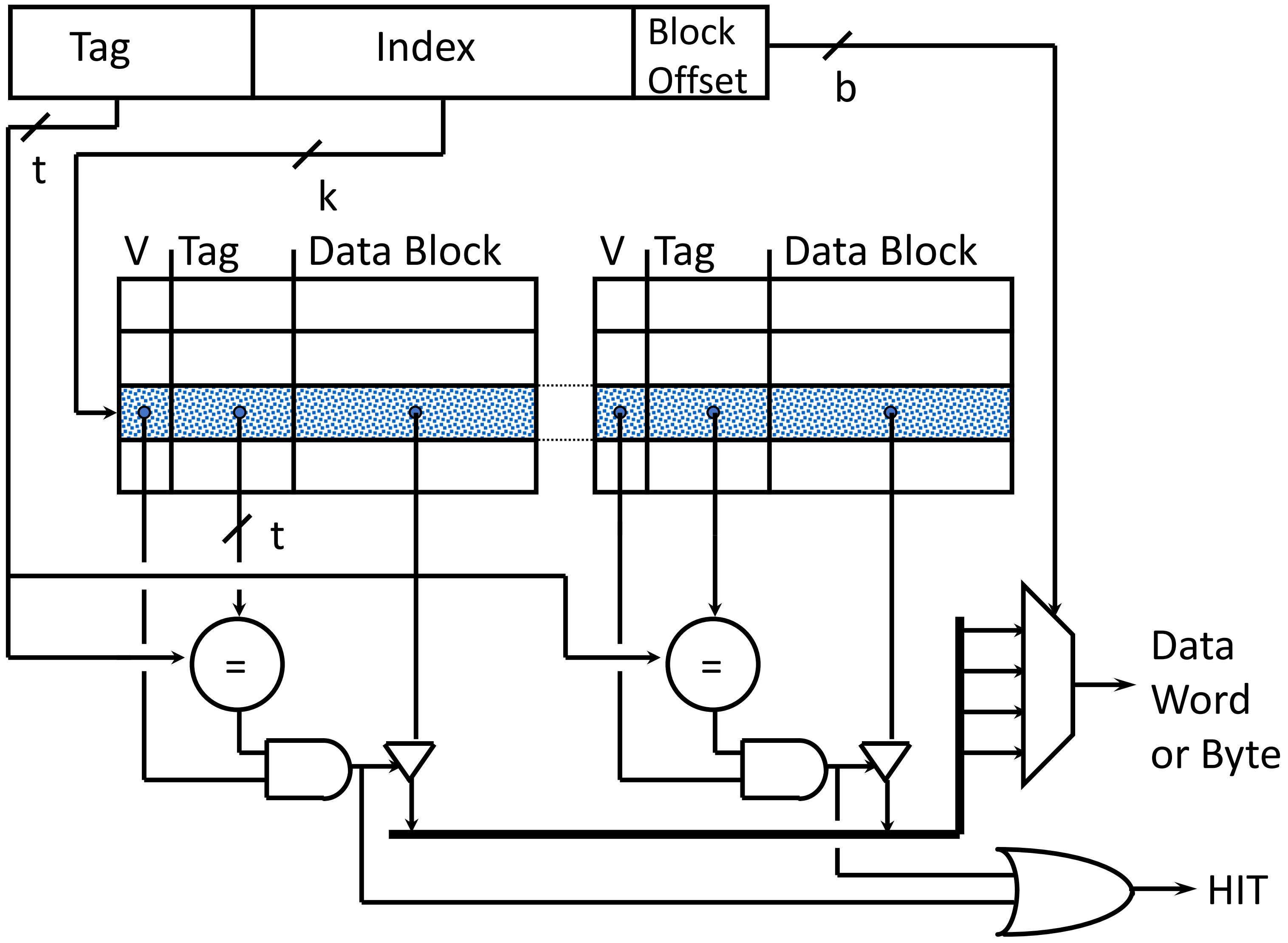


In reality, tag-store is placed separately

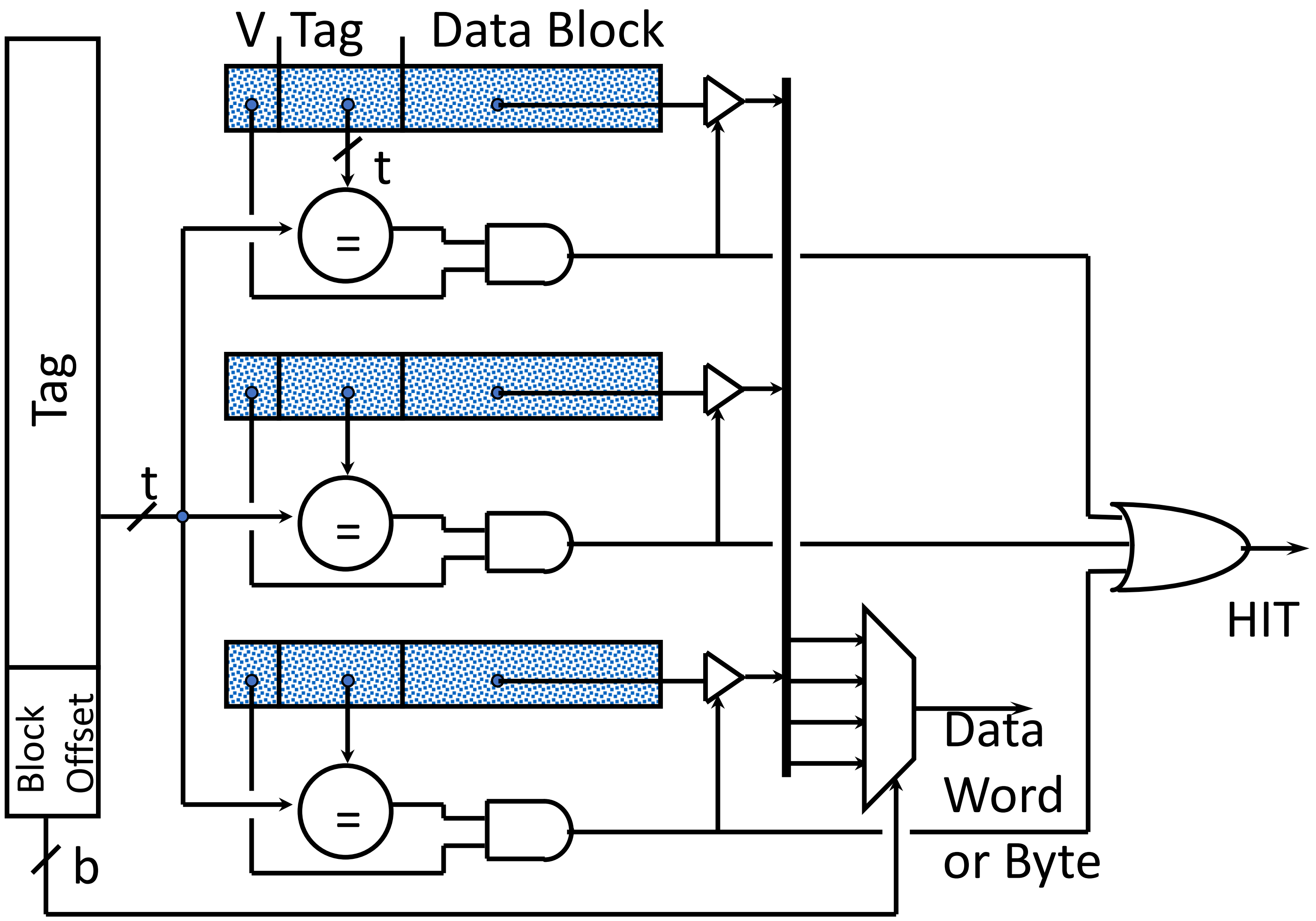
High bits or Low bits



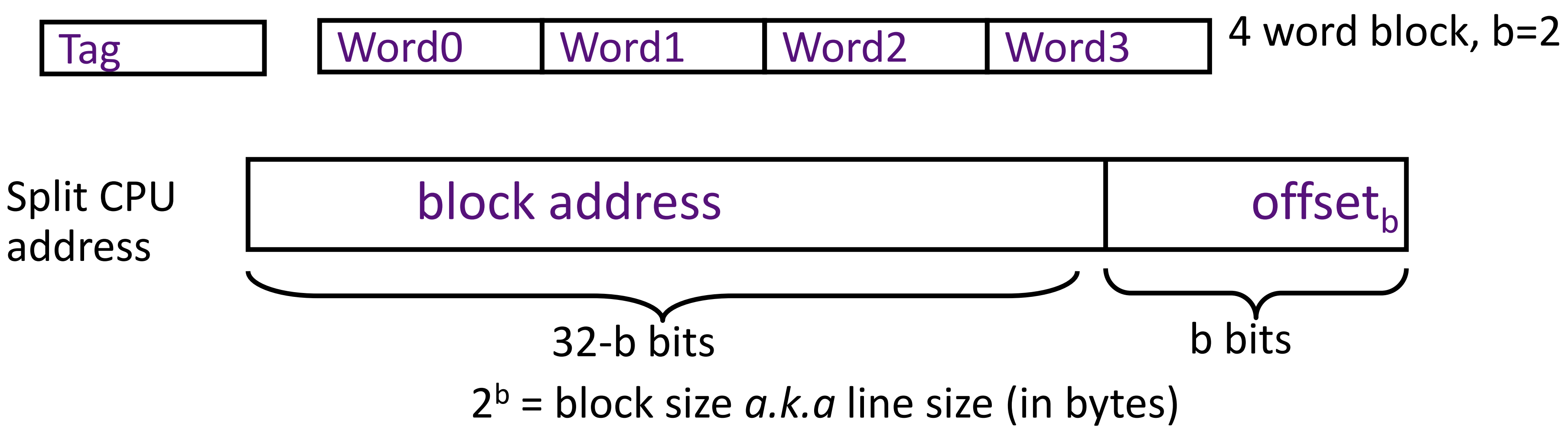
Set-Associative



Fully-associative



Block (line) Size ?



Larger block size has distinct hardware advantages

- less tag overhead
- exploit fast burst transfers from DRAM
- exploit fast burst transfers over wide busses

What are the disadvantages of increasing block size?

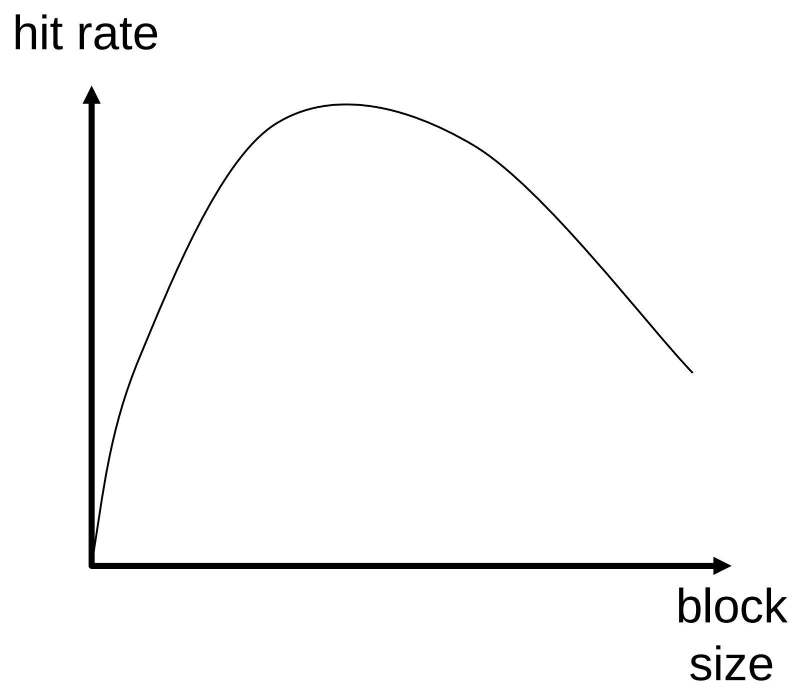
Fewer blocks => more conflicts. Can waste bandwidth.

Block Size?

- Block size is the data that is associated with an address tag
  - not necessarily the unit of transfer between hierarchies
    - Sub-blocking: A block divided into multiple pieces (each with V bit)
      - Can improve "write" performance

- Too small blocks
  - don't exploit spatial locality well
  - have larger tag overhead

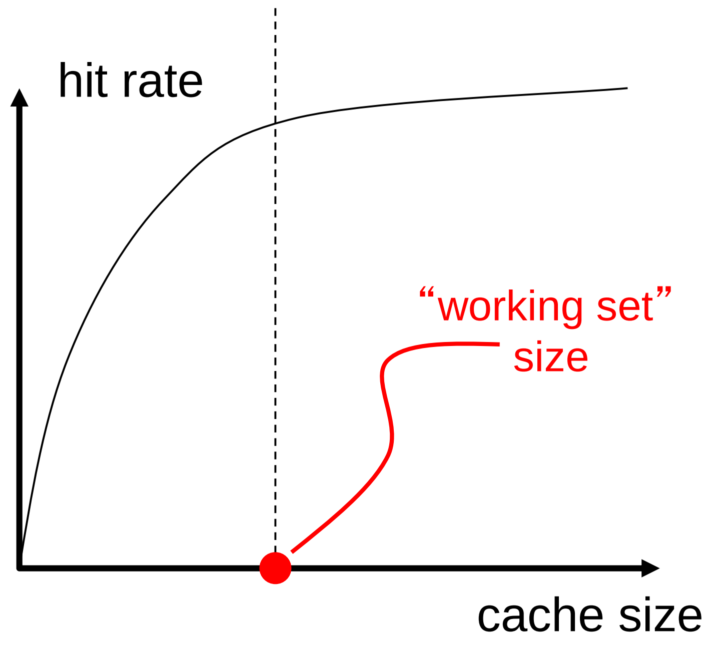
- Too large blocks
  - too few total # of blocks
    - likely-useless data transferred
    - Extra bandwidth/energy consumed



Cache Size

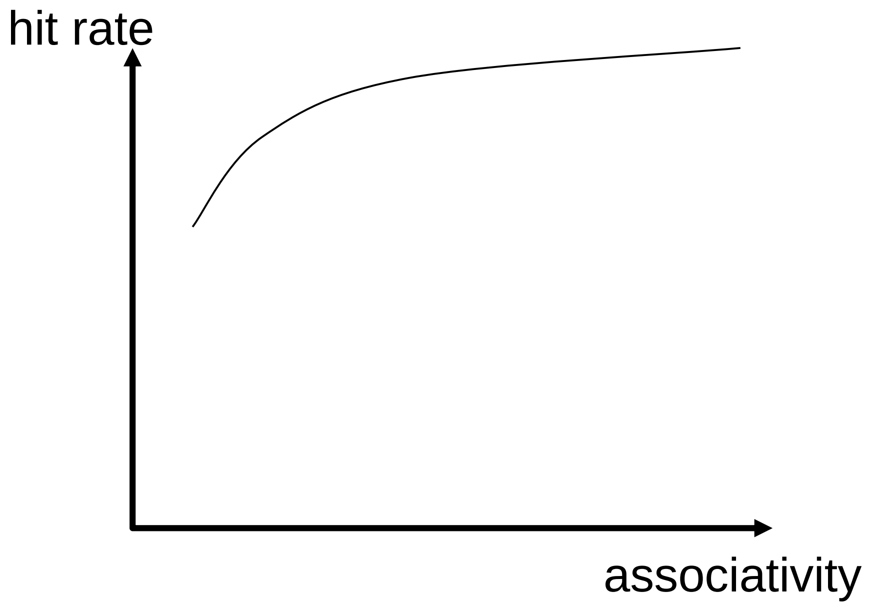
- Cache size: total data (not including tag) capacity
  - bigger can exploit temporal locality better
  - not ALWAYS better
- Too large a cache adversely affects hit and miss latency
  - smaller is faster => bigger is slower
  - access time may degrade critical path
- Too small a cache
  - doesn’t exploit temporal locality well
  - useful data replaced often

- Working set: the whole set of data the executing application references
  - Within a time interval

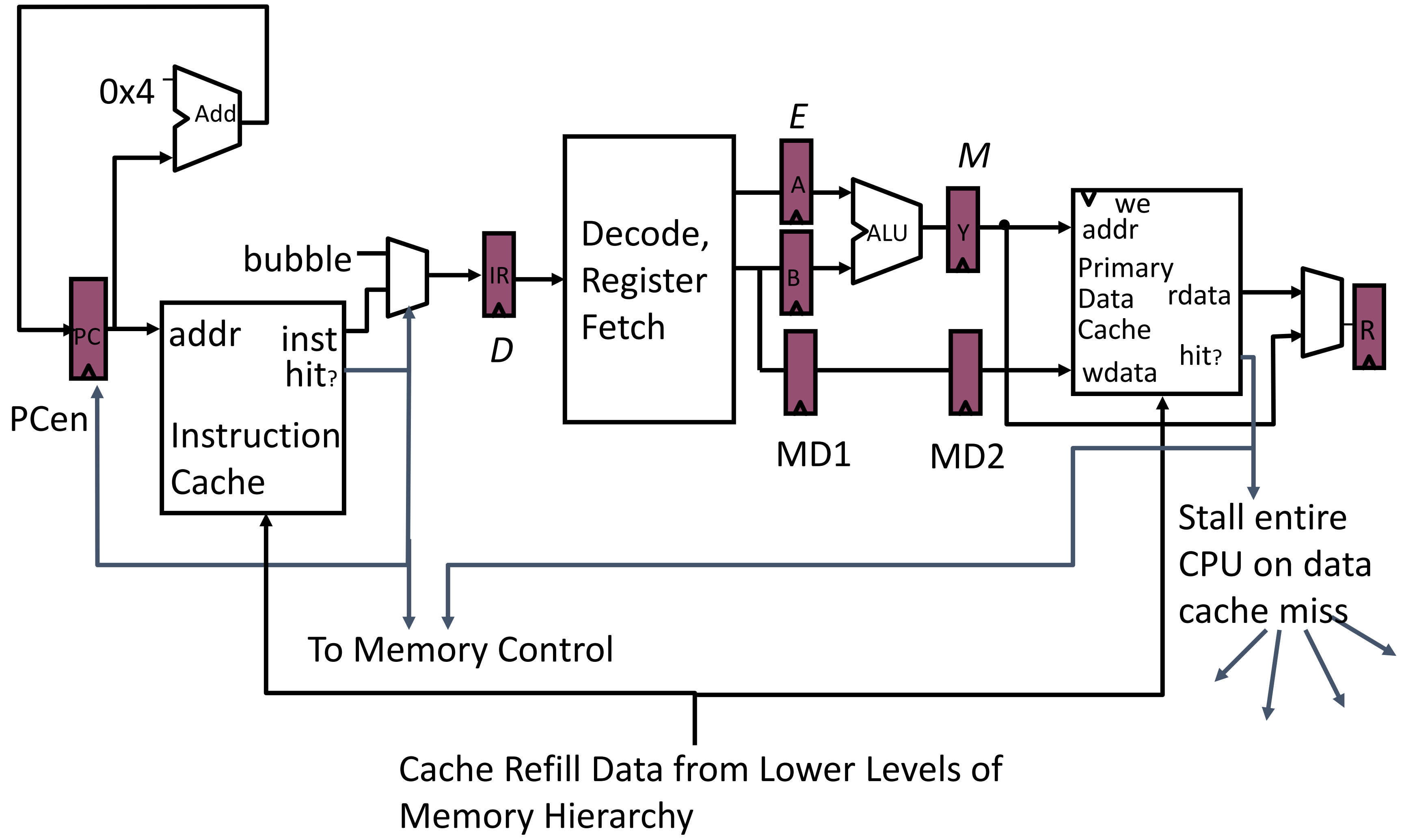


Associativity

- How many blocks can map to the same index (or set)?
- Larger associativity
  - lower miss rate, less variation among programs
  - diminishing returns, higher hit latency
- Smaller associativity
  - lower cost
  - lower hit latency
    - Especially important for L1 caches
- Power of 2 associativity?



CPU – Cache Interaction


0x4
Add
bubble
PC
PCen
addr
inst
hit?
Instruction Cache
IR
D
Decode, Register Fetch
E
A
B
ALU
M
Y
MD1
MD2
we
addr
Primary Data Cache
rdata
hit?
wdata
R
Stall entire CPU on data cache miss
To Memory Control
Cache Refill Data from Lower Levels of Memory Hierarchy