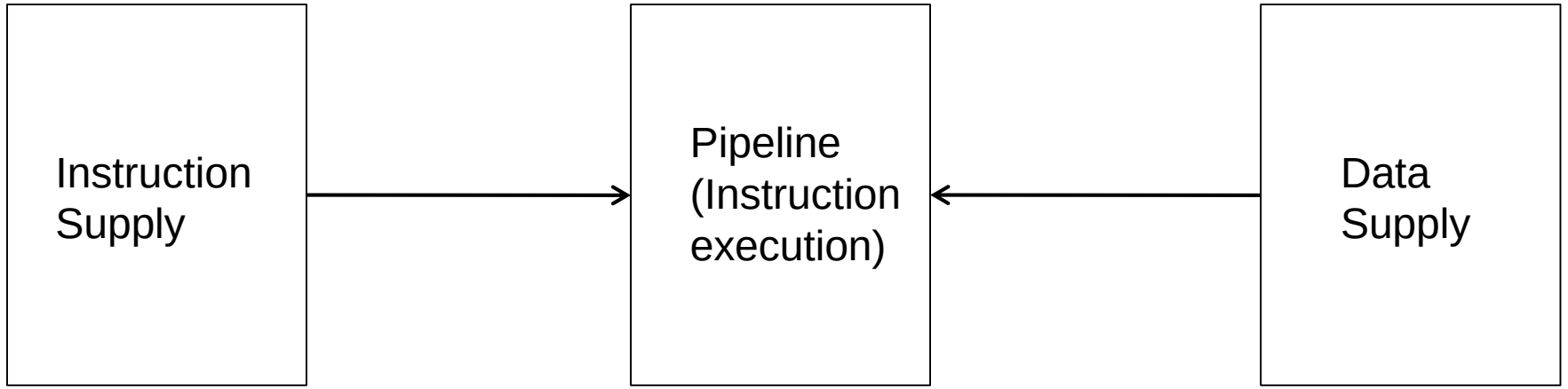
Lecture-14 (Memory Hierarchy)
CS422-Spring 2018

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

World of Memory Hierarchy. But Why?

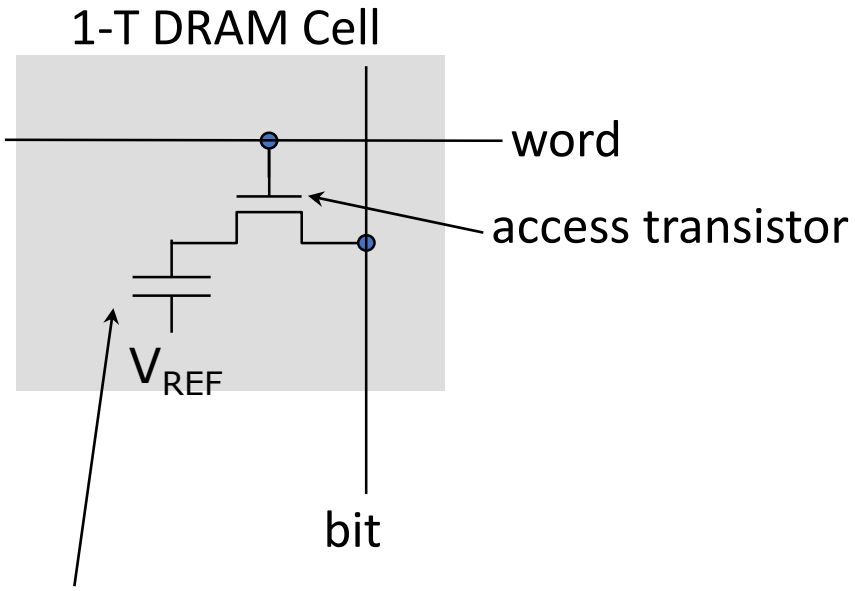
Semiconductor Memory

- 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).

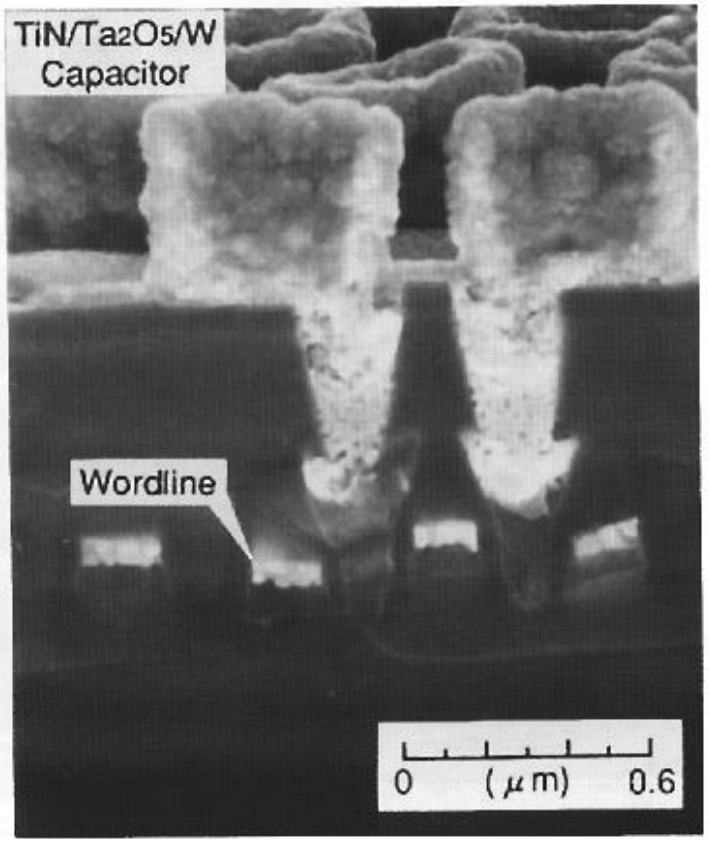
- First commercial Dynamic RAM (DRAM) was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value

Semiconductor memory quickly replaced core in '70s

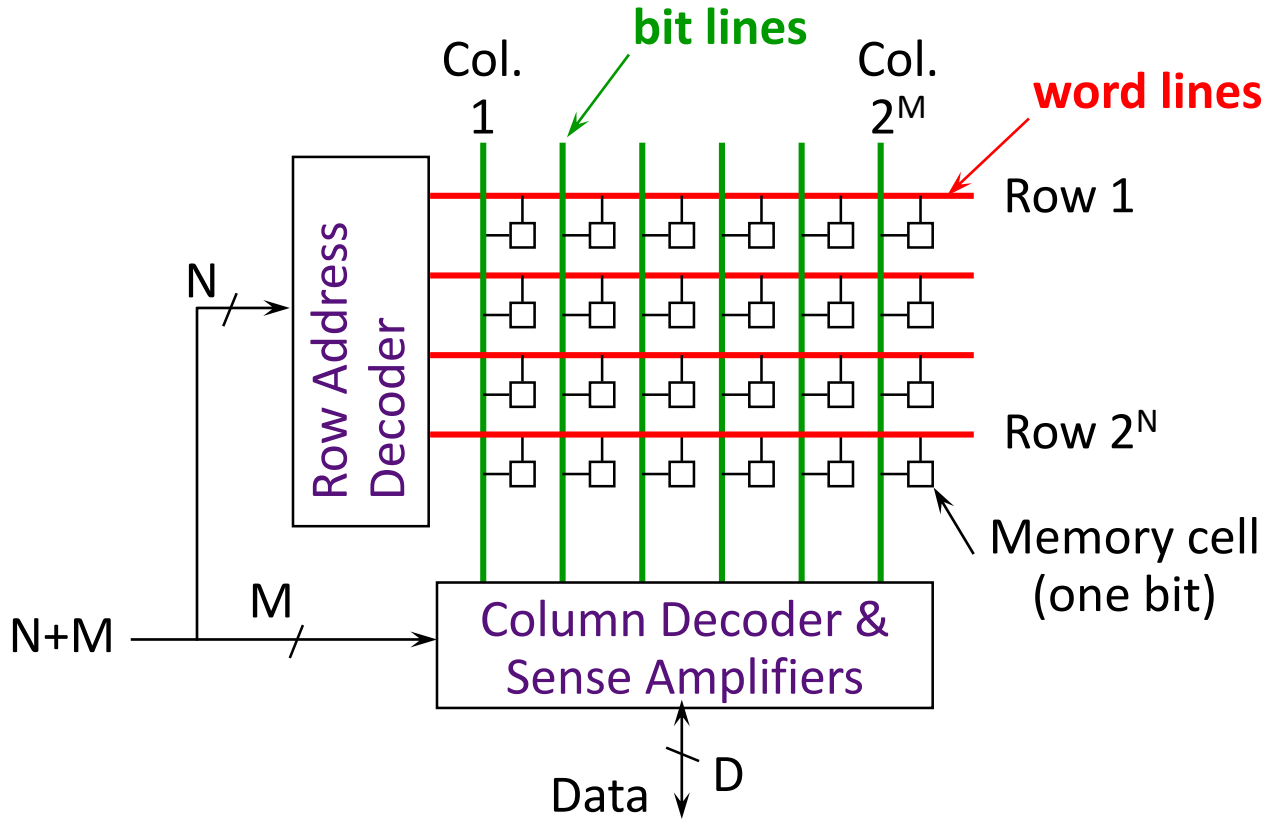
One-transistor DRAM



Storage
capacitor (FET gate, trench, stack)



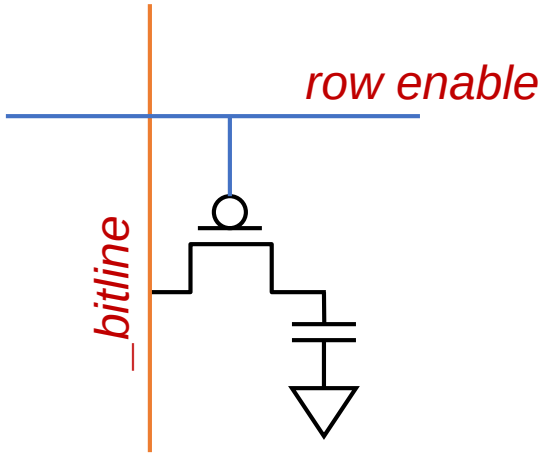
DRAM Architecture



- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4-8 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays

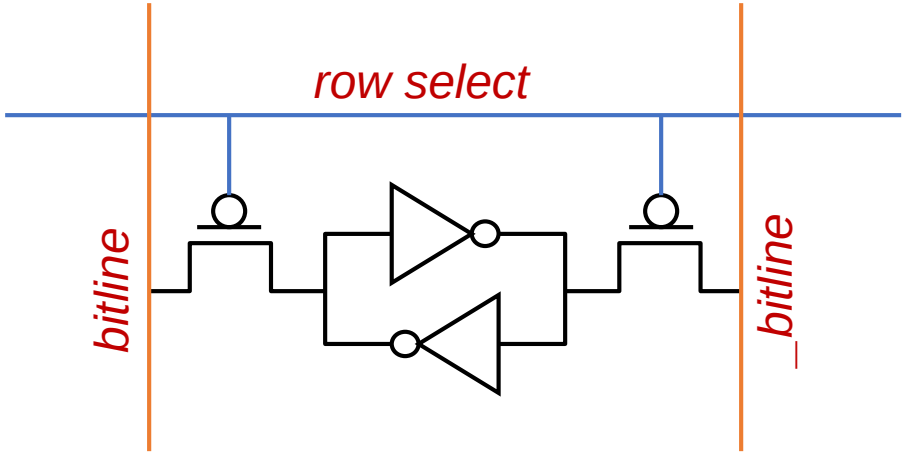
DRAM

- Dynamic random access memory
- Capacitor charge state indicates stored value
  - Whether the capacitor is charged or discharged indicates storage of 1 or 0
  - 1 capacitor
  - 1 access transistor
- Capacitor leaks
  - DRAM cell loses charge over time
  - DRAM cell needs to be refreshed



SRAM

- Static random access memory
- Two cross coupled inverters store a single bit
  - Feedback path enables the stored value to persist in the “cell”
  - 4 transistors for storage
  - 2 transistors for access



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

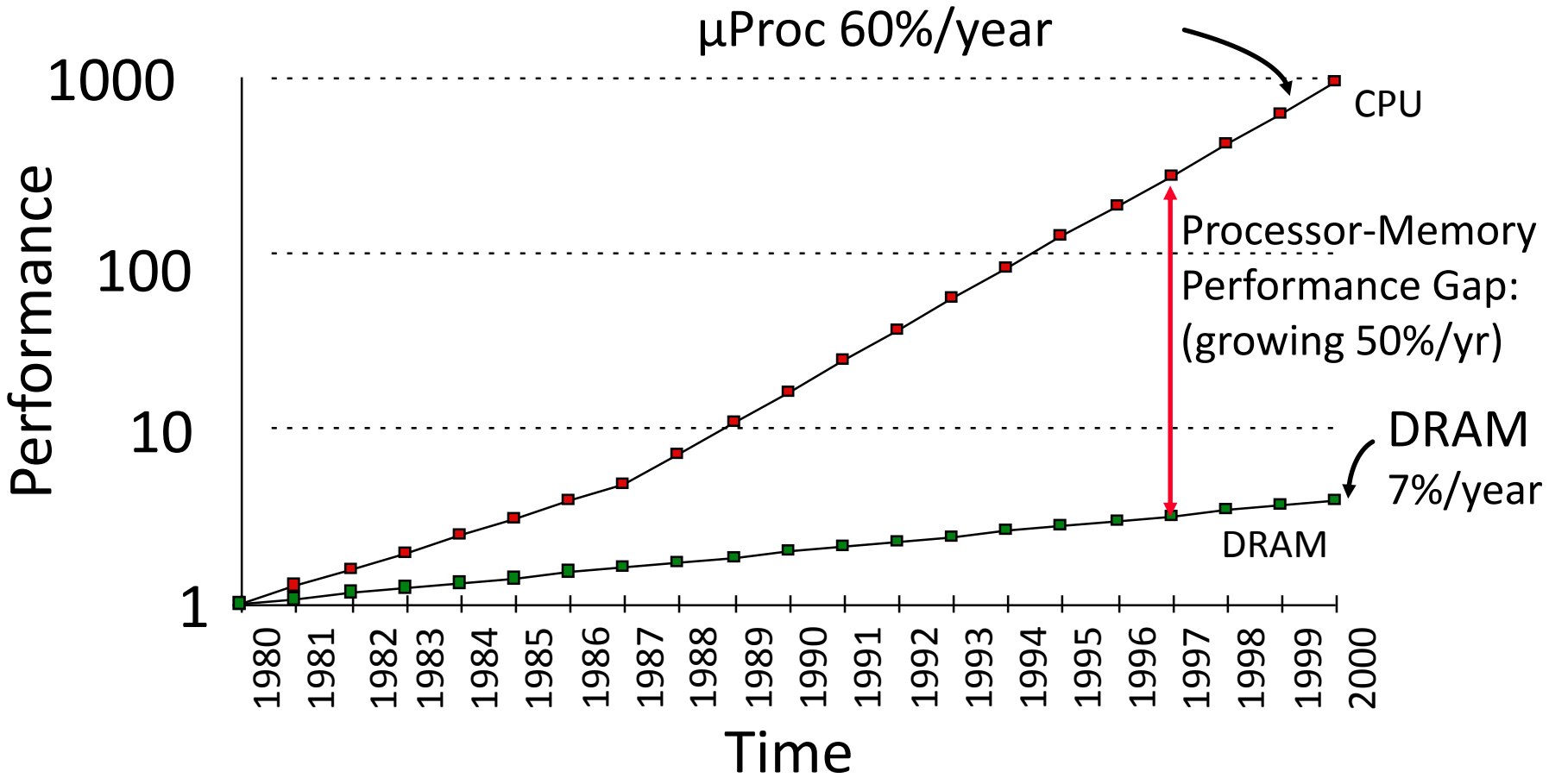
The Problem?

- 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)

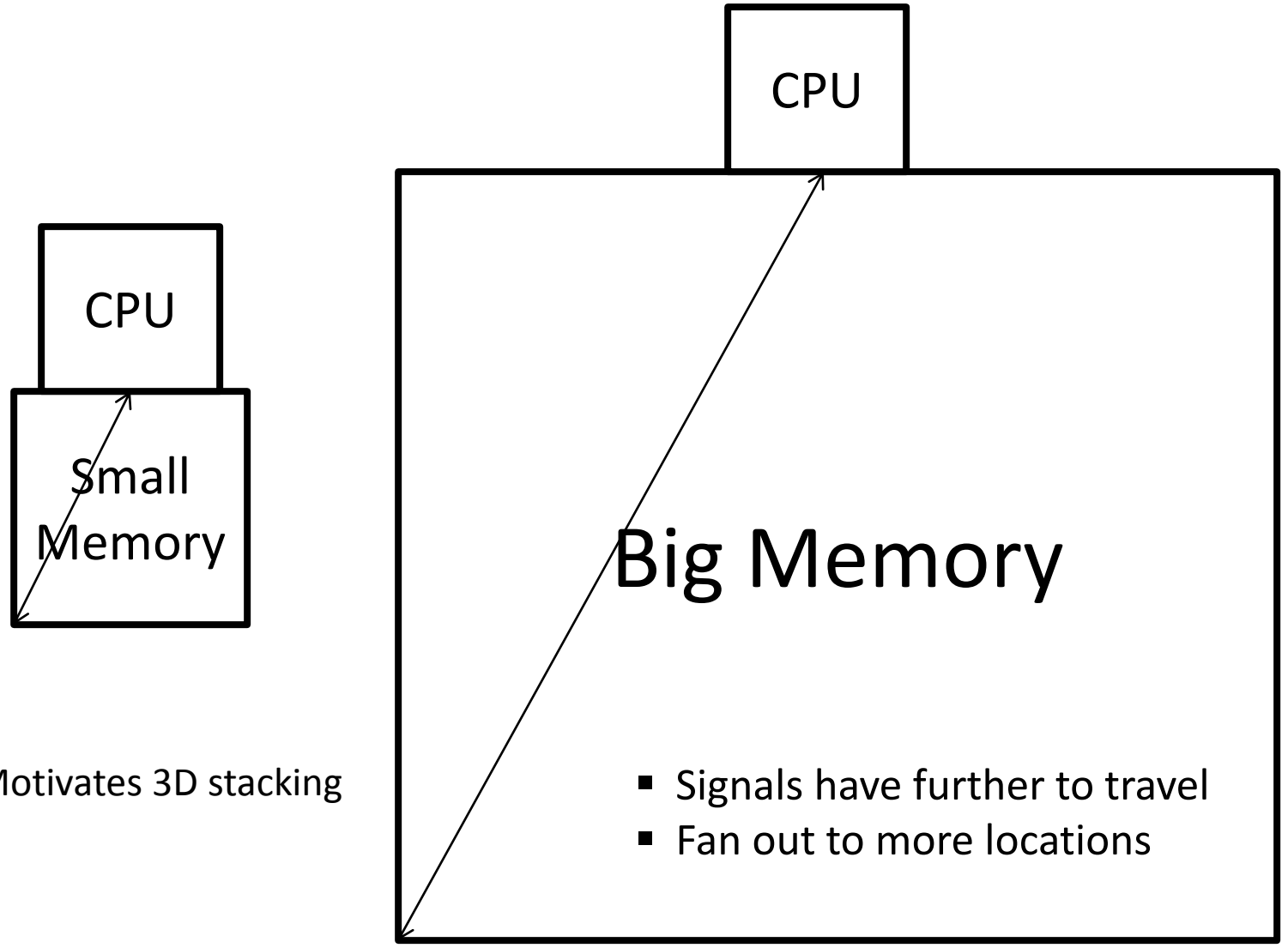
Memory Wall Problem



Four-issue 4GHz superscalar accessing 100ns DRAM could execute _____ instructions during time for one memory access!

1600

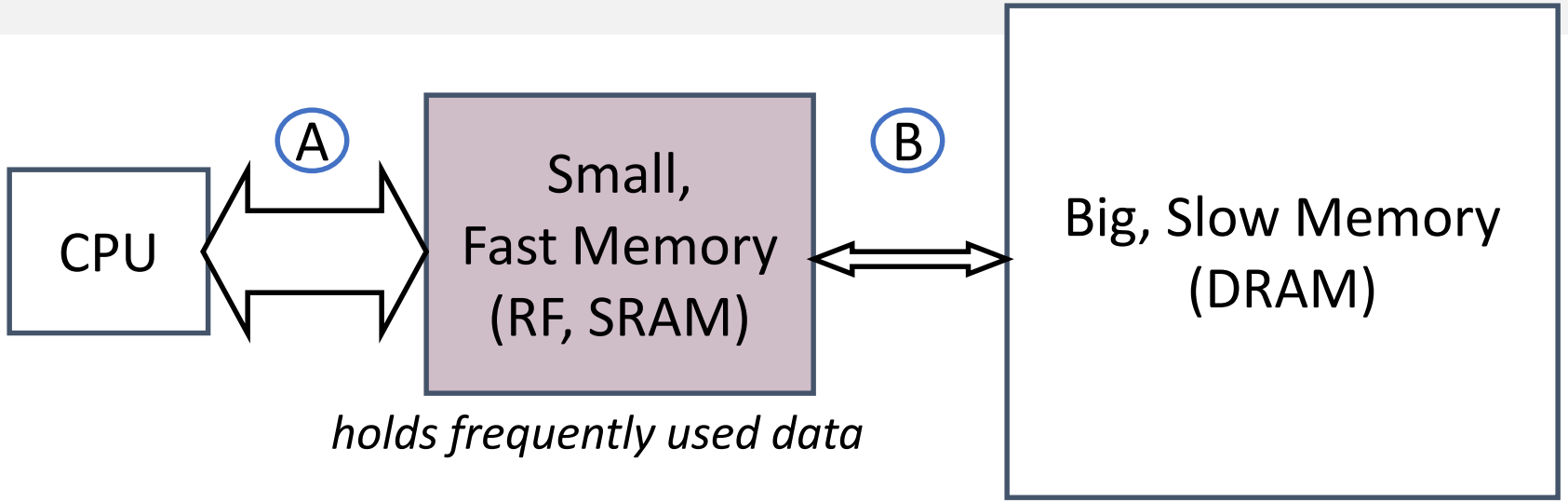
Size Affects Latency



Motivates 3D stacking

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

Memory Hierarchy



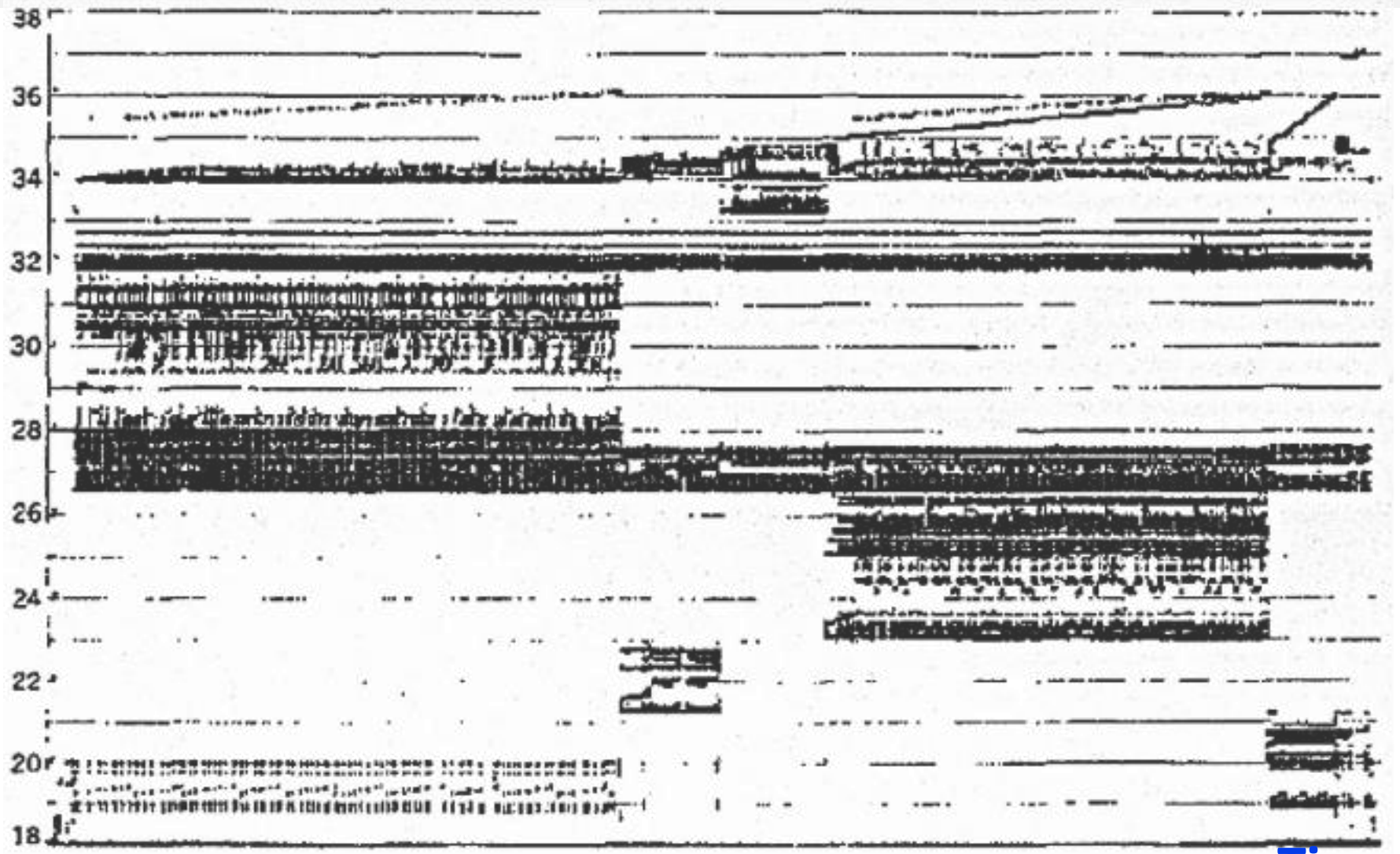
- *capacity*: Register << SRAM << DRAM
- *latency*: Register << SRAM << DRAM
- *bandwidth:* on-chip >> off-chip

On a data access:

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

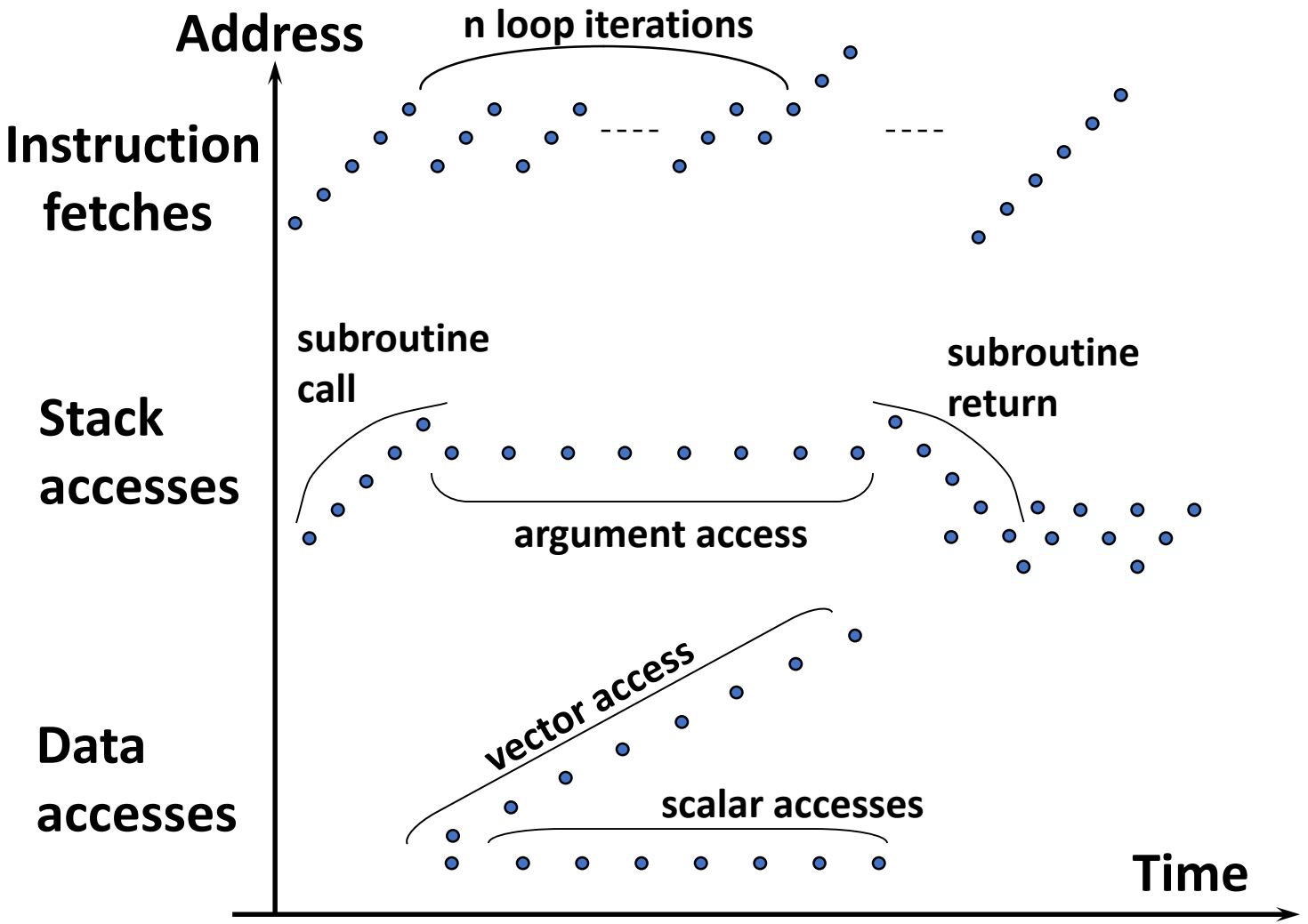
if data $\notin$ fast memory $\Rightarrow$ high latency access *(DRAM)*

Access Patterns



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

Examples

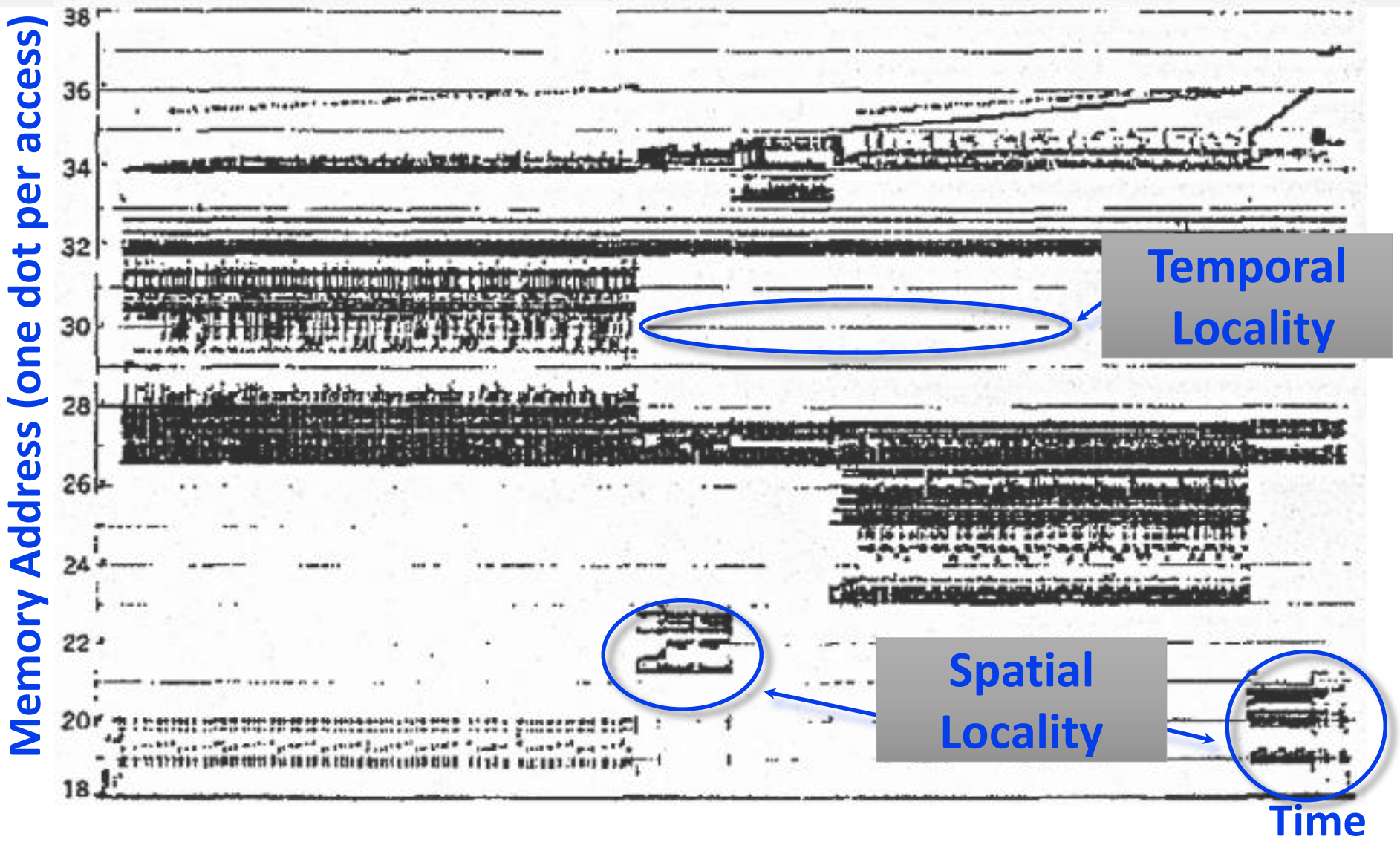

Address
n loop iterations
Instruction fetches
subroutine call
subroutine return
Stack accesses
argument access
vector access
scalar accesses
Data accesses
Time

Locality of Reference

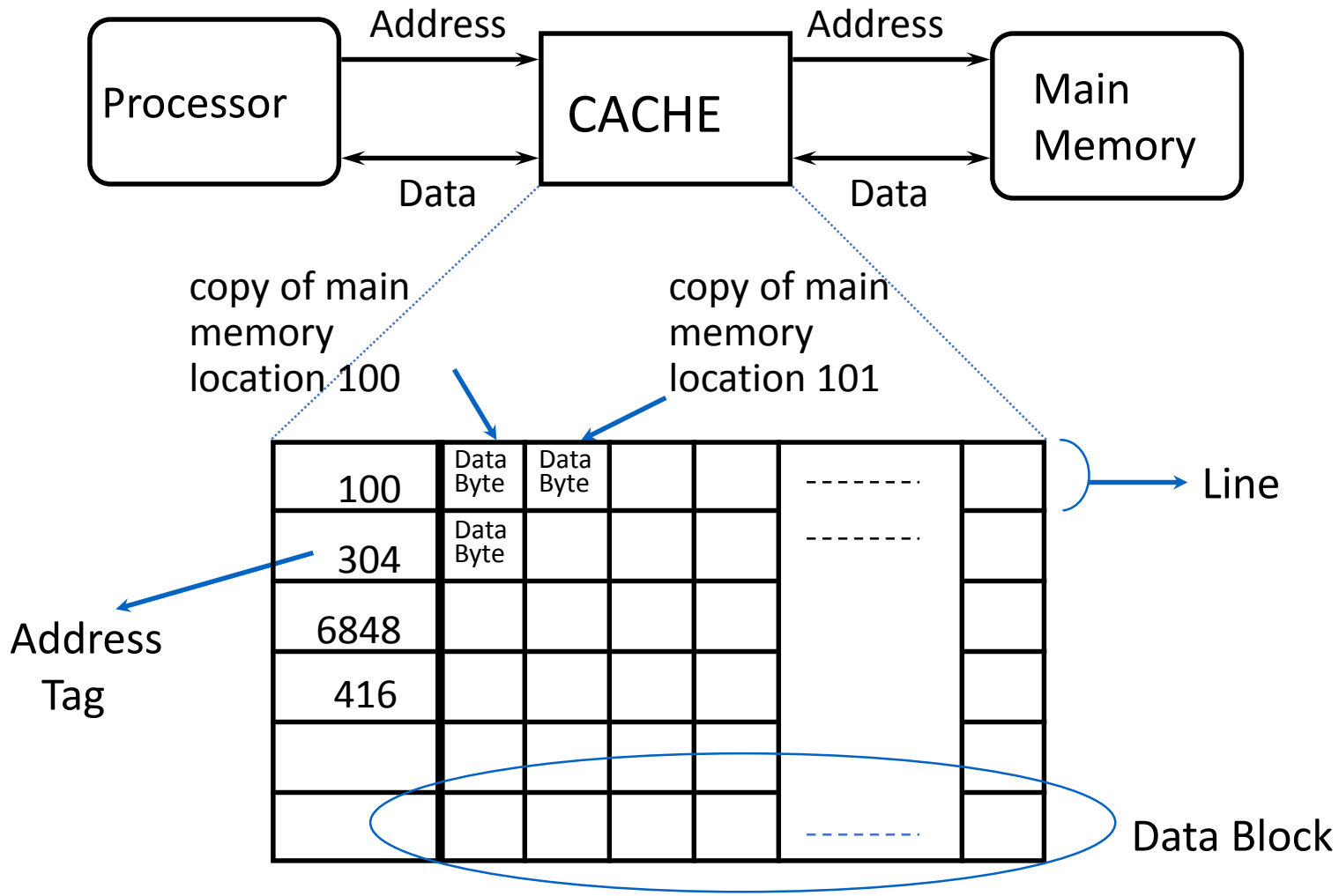
- **Temporal Locality**: If a location is referenced it is likely to be referenced again in the near future.

- **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

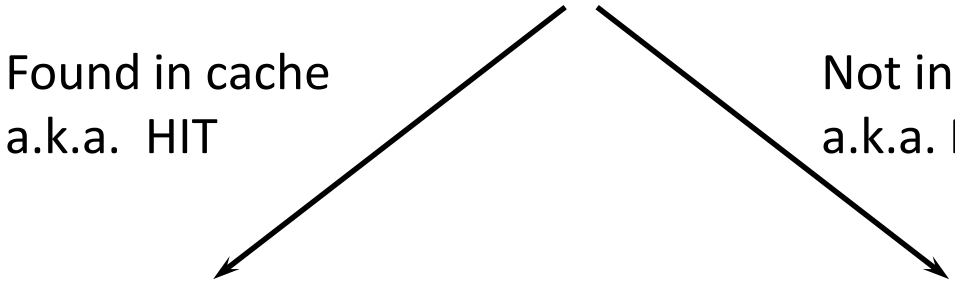


Intel i7

- Private L1 and L2
  - L2 is 256KB each. 10 cycle latency
- 8MB shared L3. ~40 cycles latency

Cache Events

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



Return copy of data from cache

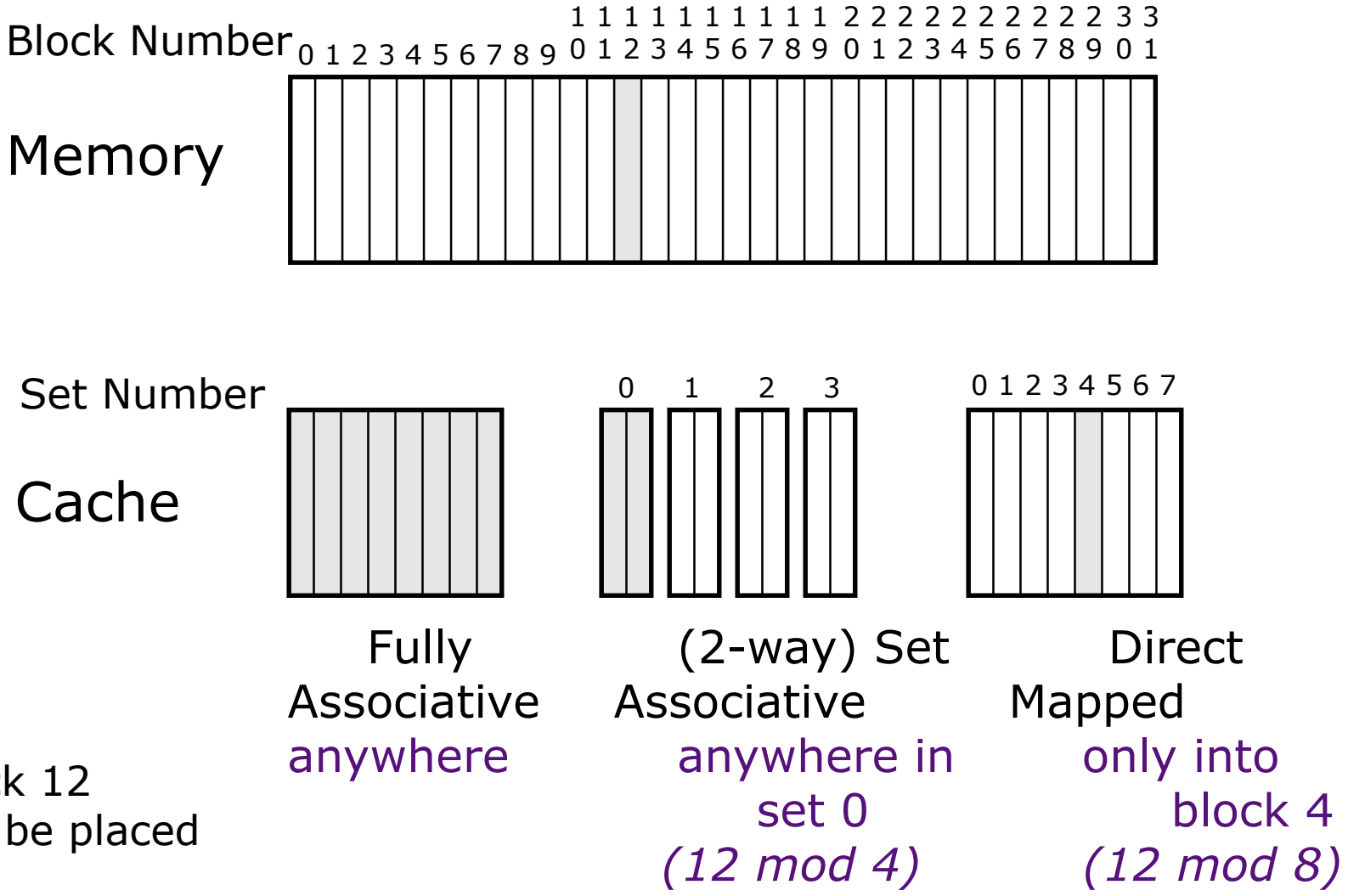
Read block of data from Main Memory

Wait …

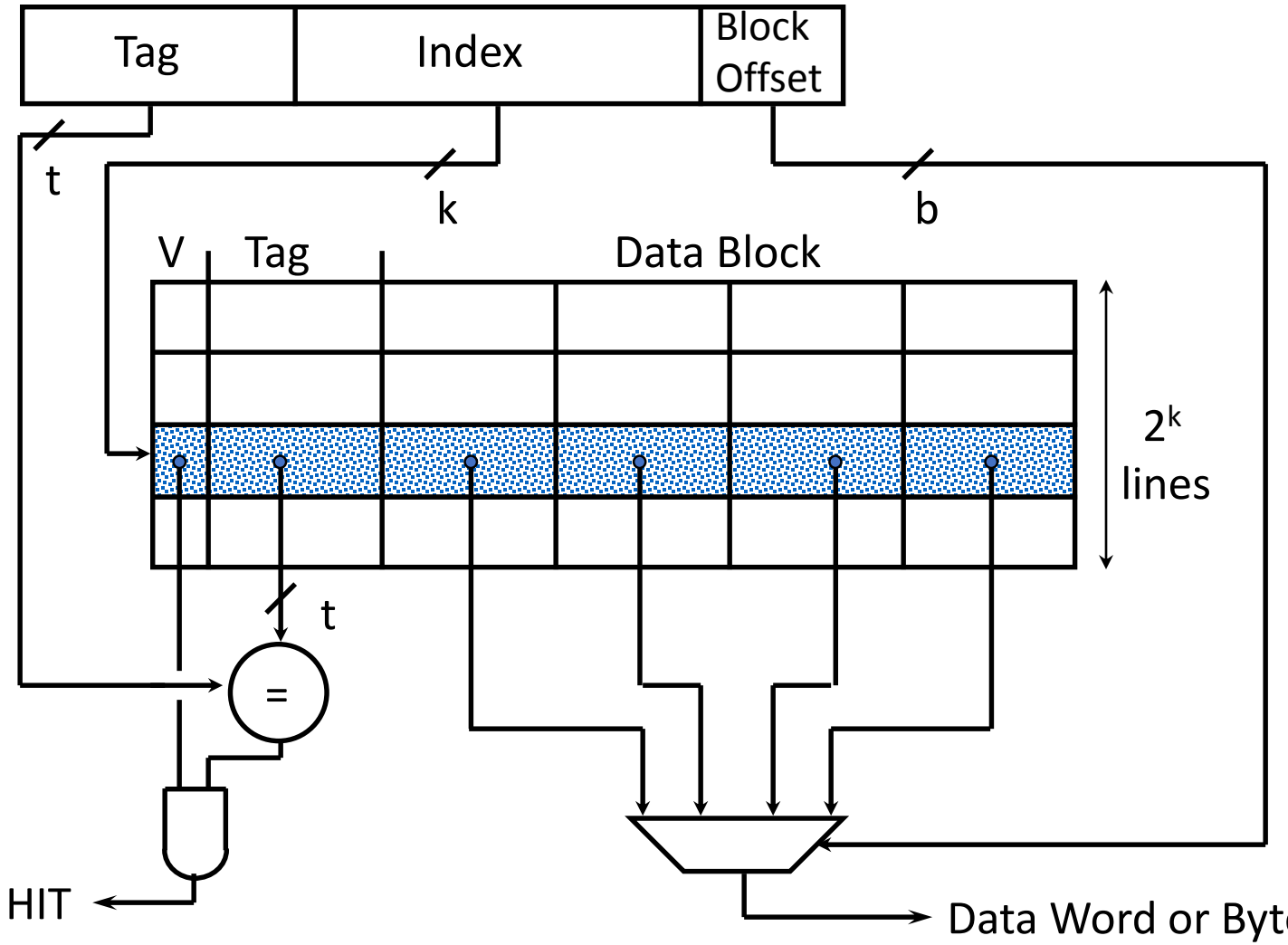
Return data to processor and update cache

Q: Which line do we replace?

Placement Policy

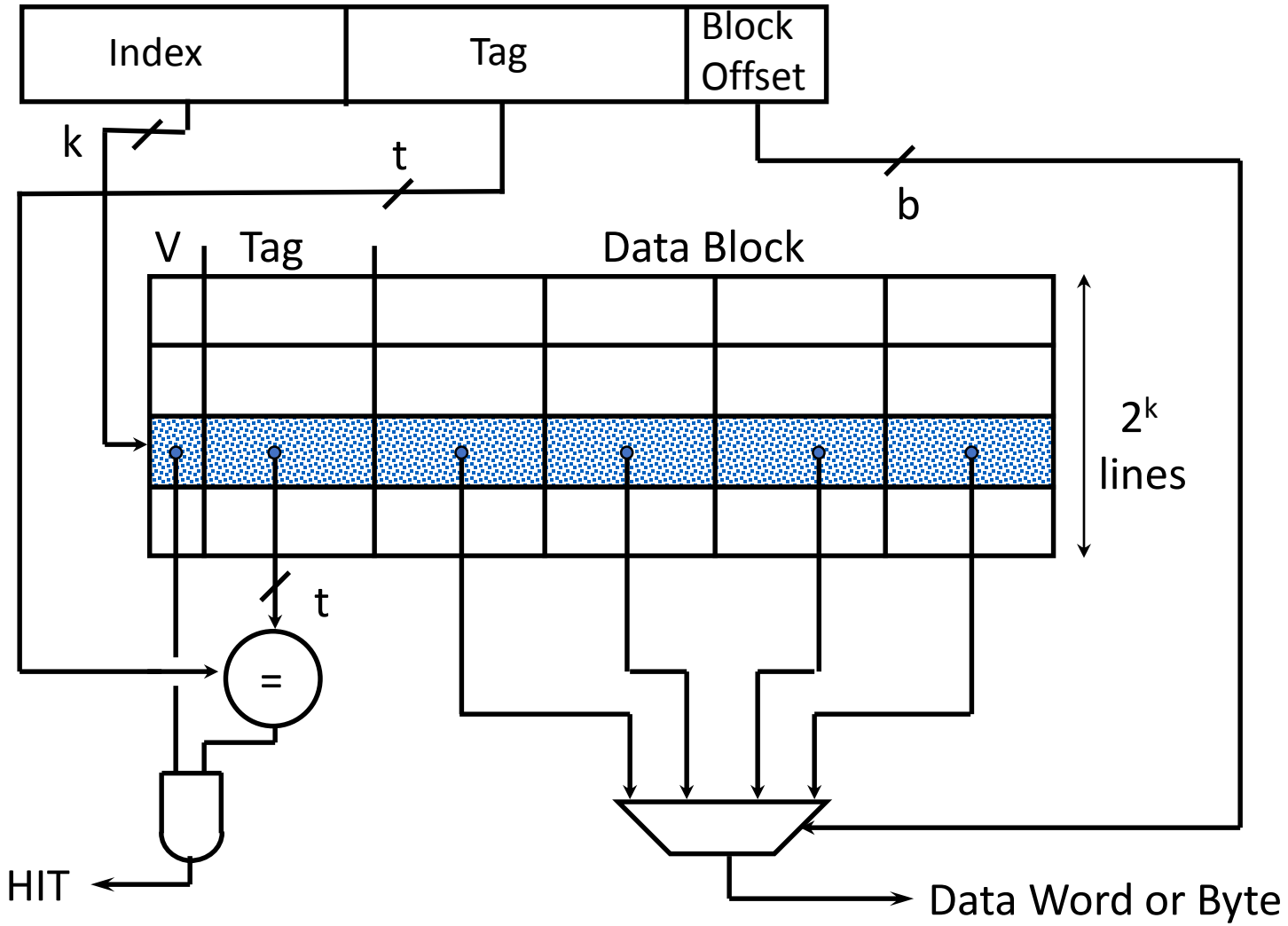


Direct Mapped

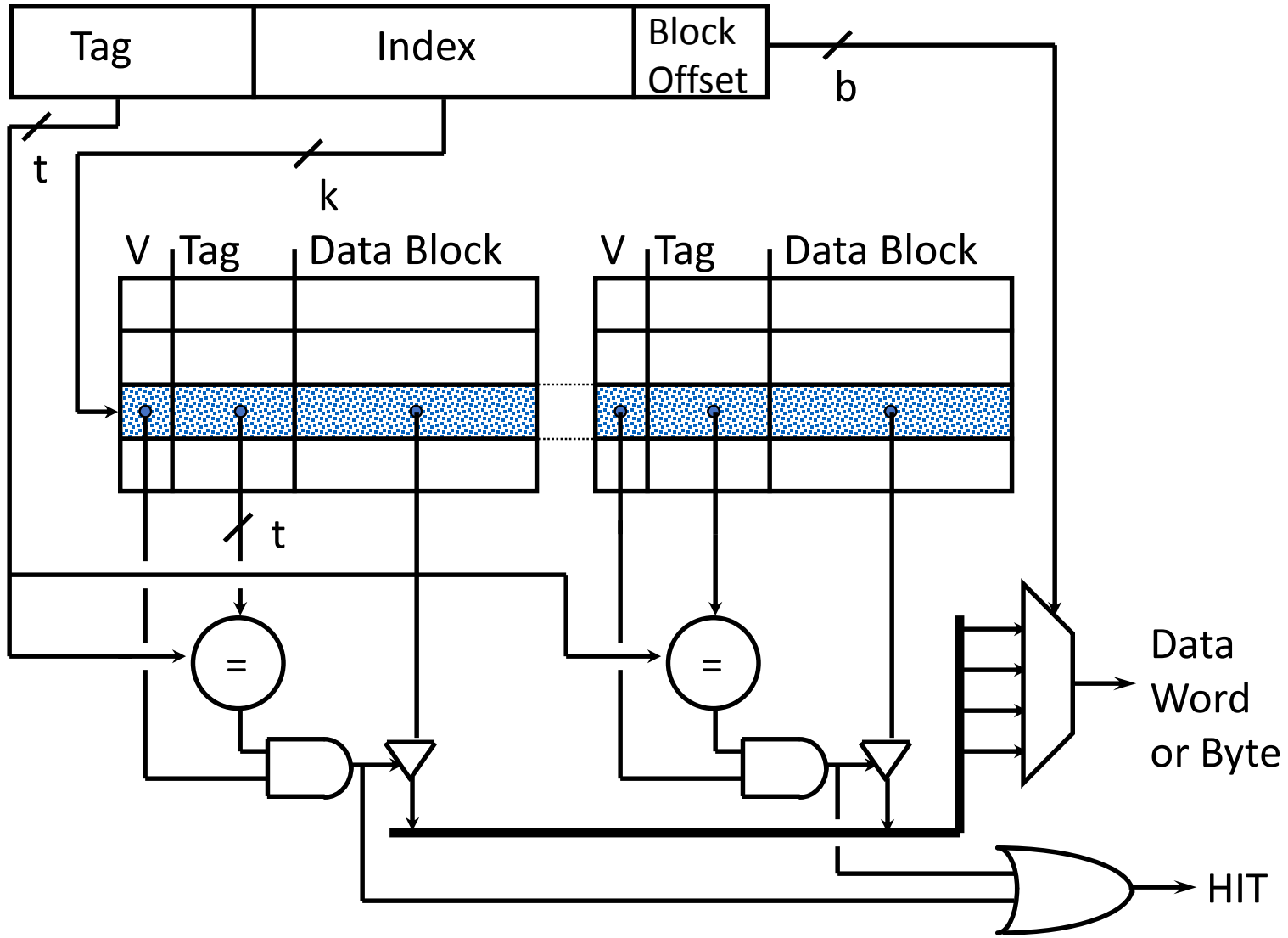


In reality, tag-store is placed separately

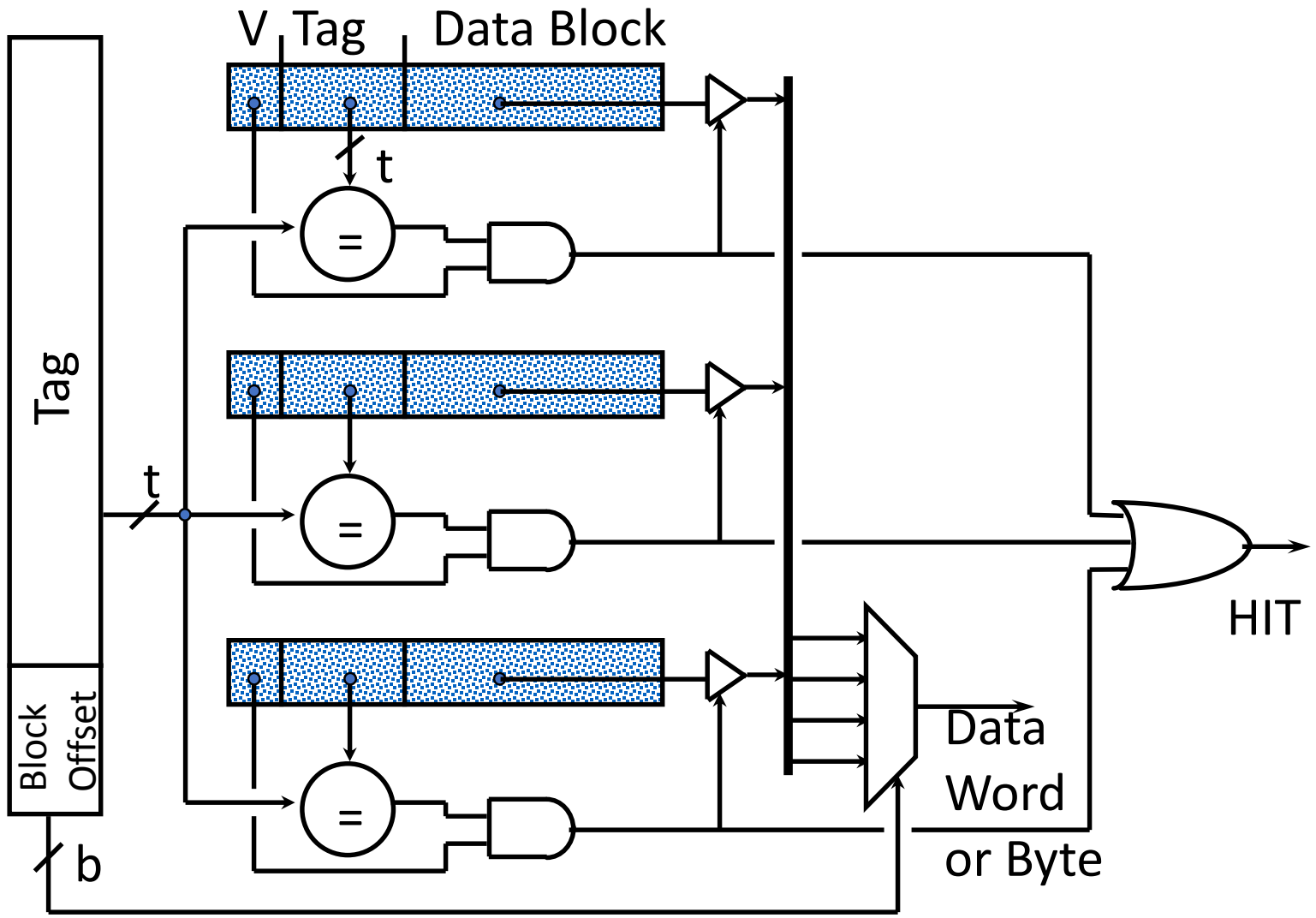
High bits or Low bits



Set-Associative



Fully-associative



What's in Tag Store?

- Valid bit
- Tag
- Replacement policy bits

- Dirty bit?
  - Write back vs. write through caches