The Memory Hierarchy

Memory Technologies

- Technologies have vastly different tradeoffs between capacity, latency, bandwidth, energy, and cost
  - … and thus different applications

	Capacity	Latency	Cost/GB
Register	100s of bits	20 ps	$$$$
SRAM	~10 KB-10 MB	1-10 ns	~$1000
DRAM	~10 GB	80 ns	~$10
Flash*	~100 GB	100 us	~$1
Hard disk*	~1 TB	10 ms	~$0.1

Processor Datapath

Memory Hierarchy

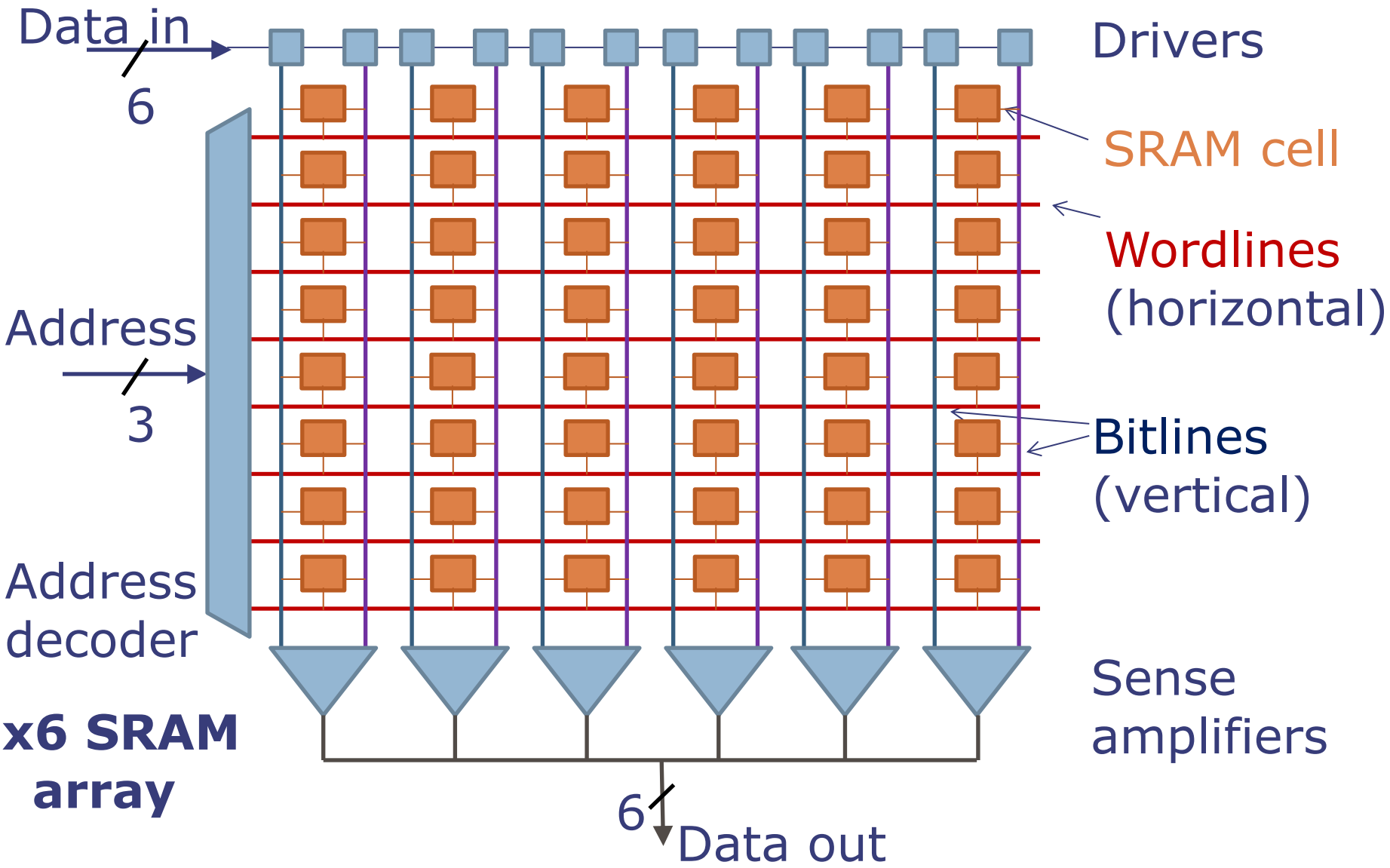
I/O subsystem

* non-volatile (retains contents when powered off)

Memory Technologies: SRAM, DRAM, Flash, Hard Disk

NOTE: Demystification, will not be on the quiz

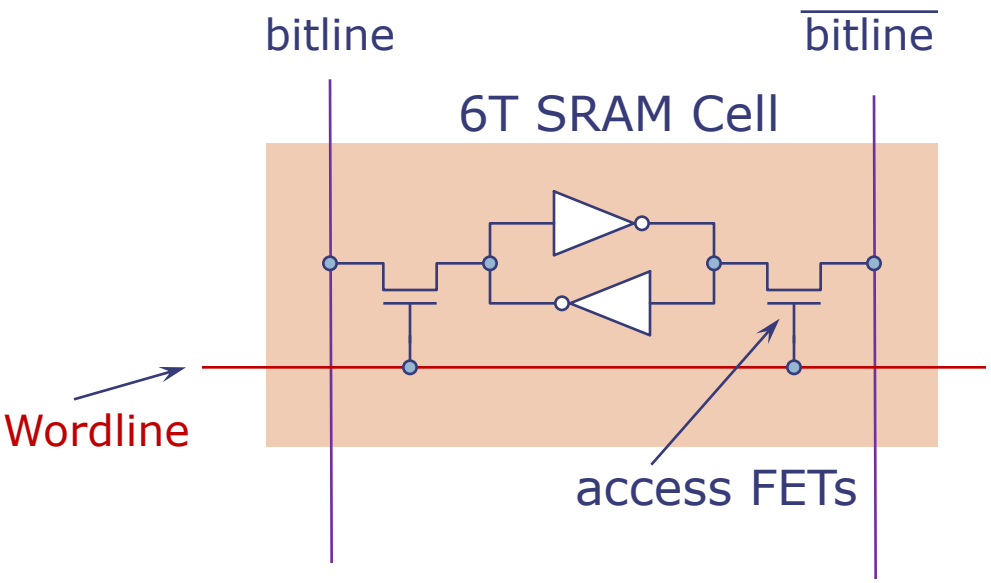
Static RAM (SRAM)



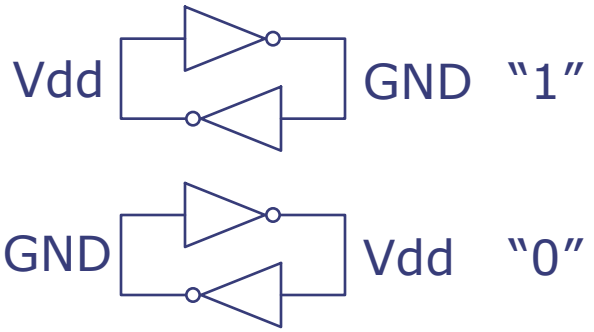
SRAM Cell

6-transistor (6T) cell:

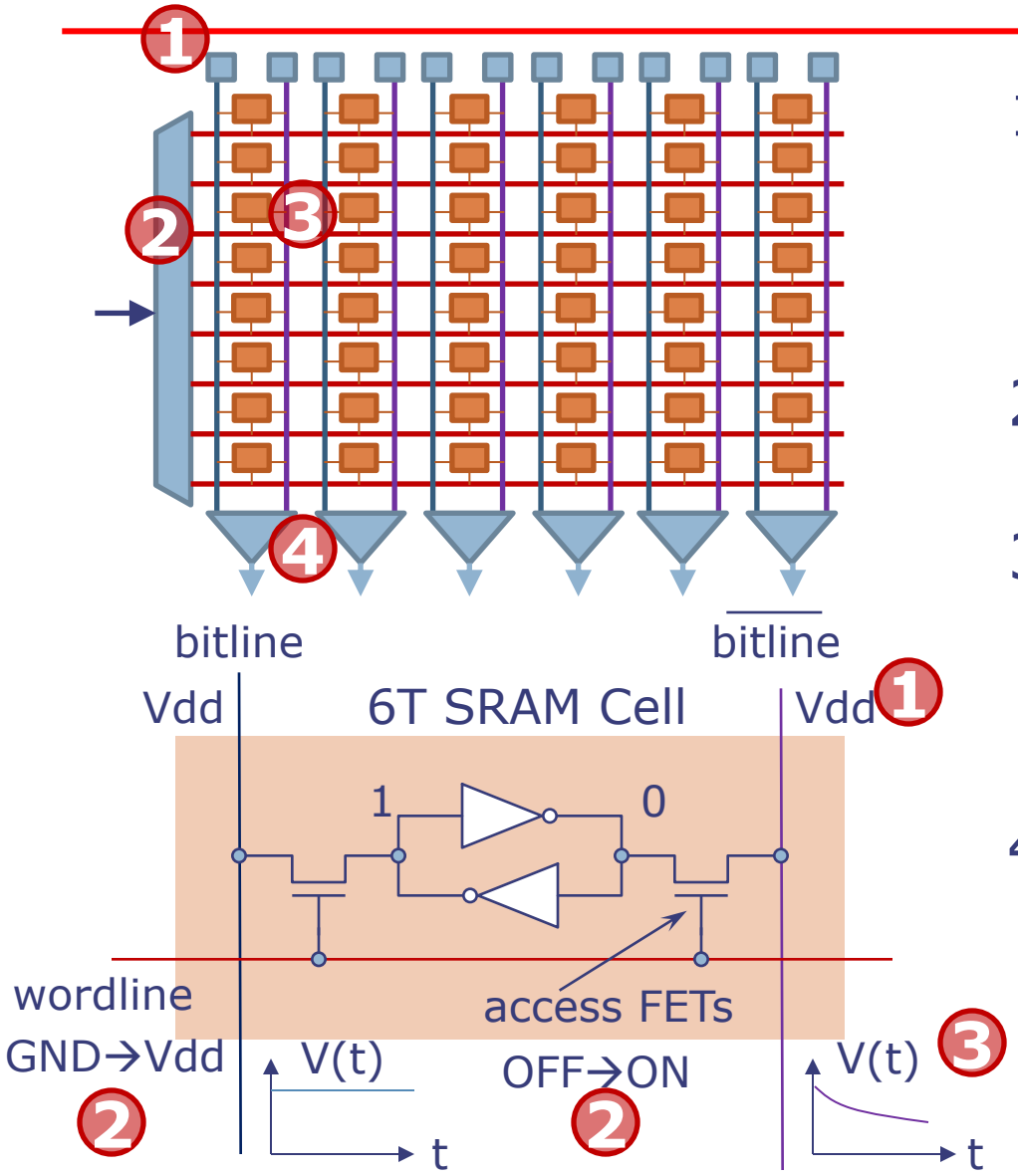
- Two CMOS inverters (4 FETs) forming a bistable element
- Two access transistors



Bistable element
(two stable states)
stores a single bit

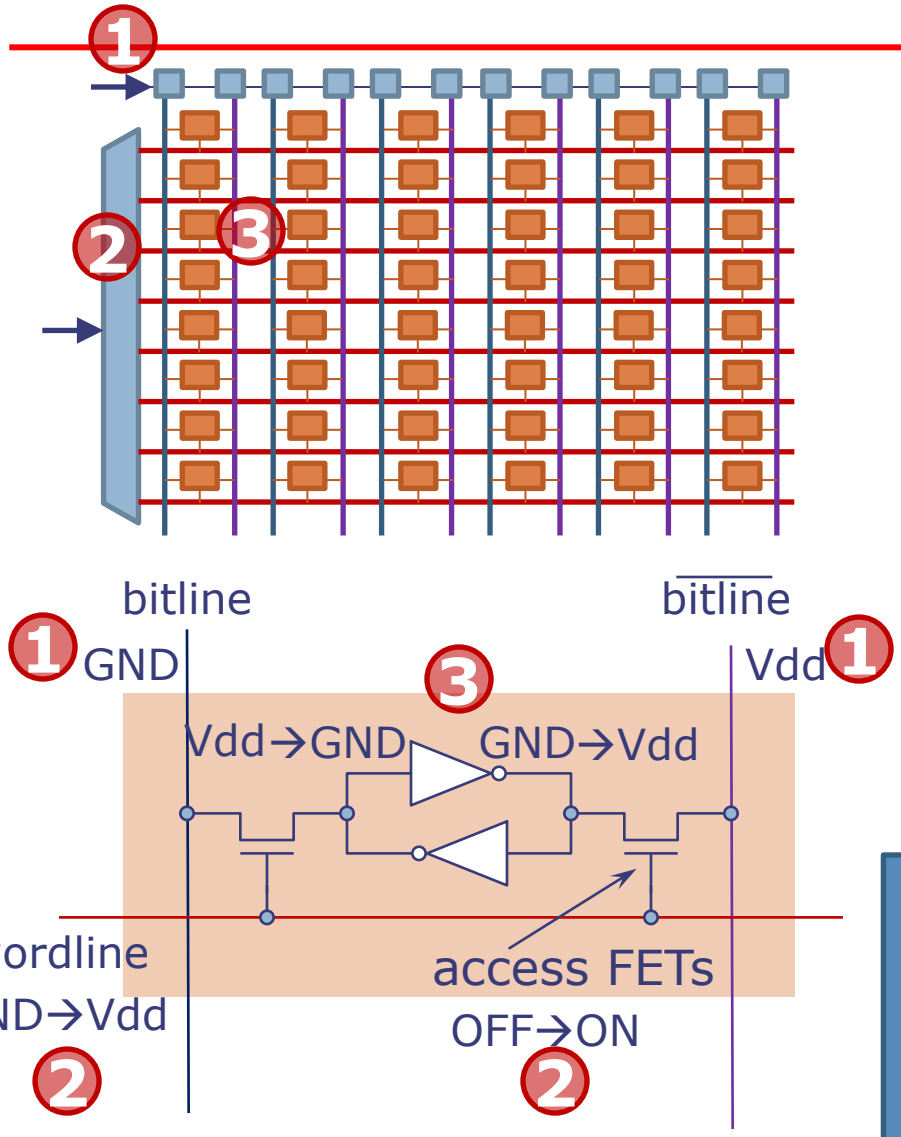


SRAM Read



1. Drivers precharge all bitlines to Vdd (1), and leave them floating
2. Address decoder activates one wordline
3. Each cell in the activated word slowly pulls down one of the bitlines to GND (0)
4. Sense amplifiers sense change in bitline voltages, produce output data

SRAM Write

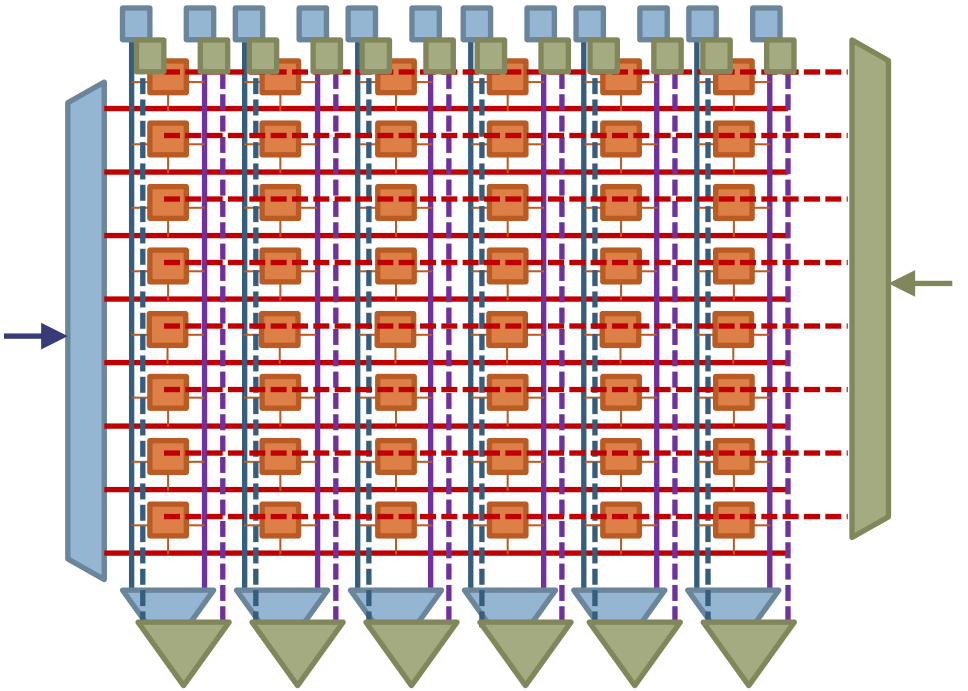


1. Drivers set and hold bitlines to desired values (Vdd and GND for 1, GND and Vdd for 0)
2. Address decoder activates one wordline
3. Each cell in word is overpowered by the drivers, stores value

Cell transistors are carefully sized so that bitline GND overpowers cell Vdd, but bitline Vdd does not overpower cell GND

Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port



- *Cost/bit for N ports?*
  - *Wordlines?* N
  - *Bitlines?* 2*N
  - *Access FETs?* 2*N

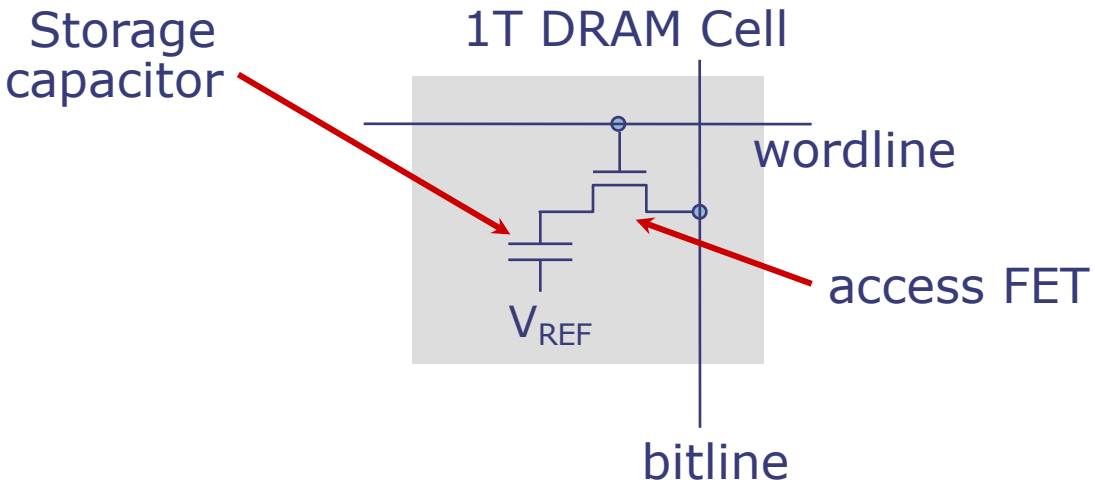
- Wires dominate area → $O(N^2)$ area!

Summary: SRAMs

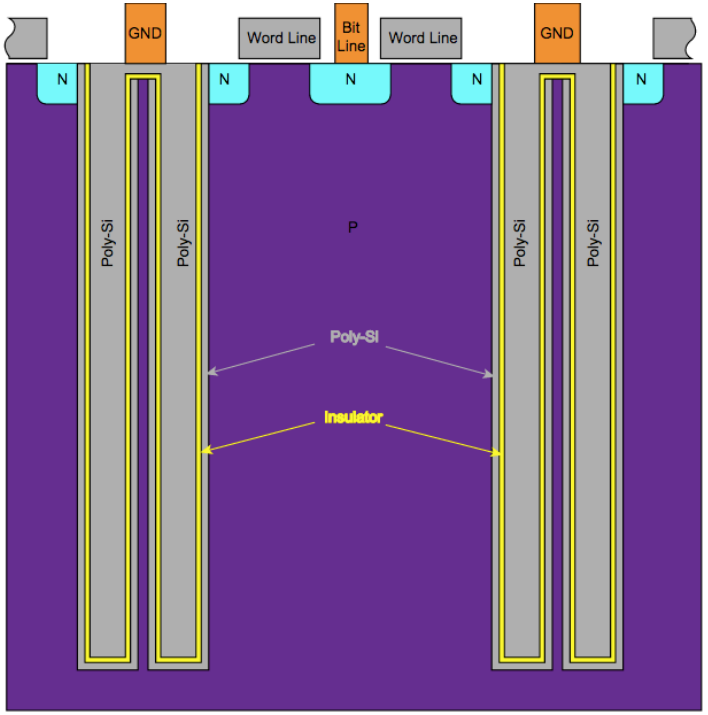
- Array of k*b cells (k words, b cells per word)
- Cell is a bistable element + access transistors
  - Analog circuit with carefully sized transistors
- Read: Precharge bitlines, activate wordline, sense
- Write: Drive bitlines, activate wordline, overpower cells

- 6 FETs/cell… can we do better?

1T Dynamic RAM (DRAM) Cell



Cyferz (CC BY 2.5)



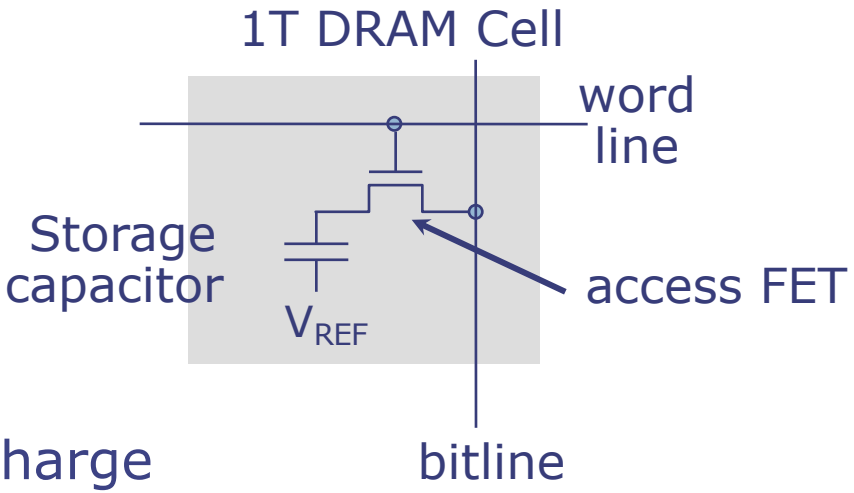
Trench capacitors take little area

✓~20x smaller area than SRAM cell → Denser and cheaper!

✗ Problem: Capacitor leaks charge, must be refreshed periodically (~milliseconds)

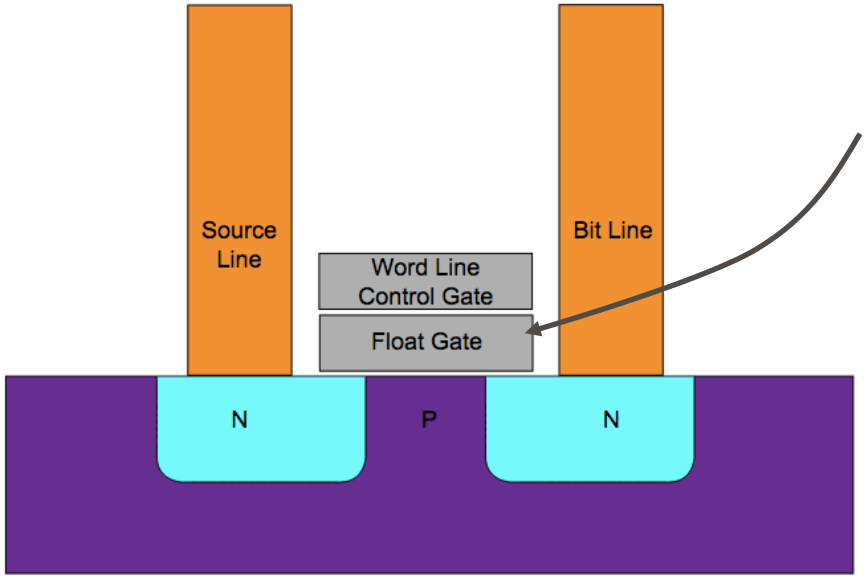
DRAM Writes and Reads

- Writes: Drive bitline to Vdd or GND, activate wordline, charge or discharge capacitor



- Reads:
  1. Precharge bitline to Vdd/2
  2. Activate wordline
  3. Capacitor and bitline share charge
     - If capacitor was discharged, bitline voltage decreases slightly
     - If capacitor was charged, bitline voltage increases slightly
  4. Sense bitline to determine if 0 or 1
- Issue: Reads are destructive! (charge is gone!)
  - Data must be rewritten to cells at end of read

Non-Volatile Storage: Flash



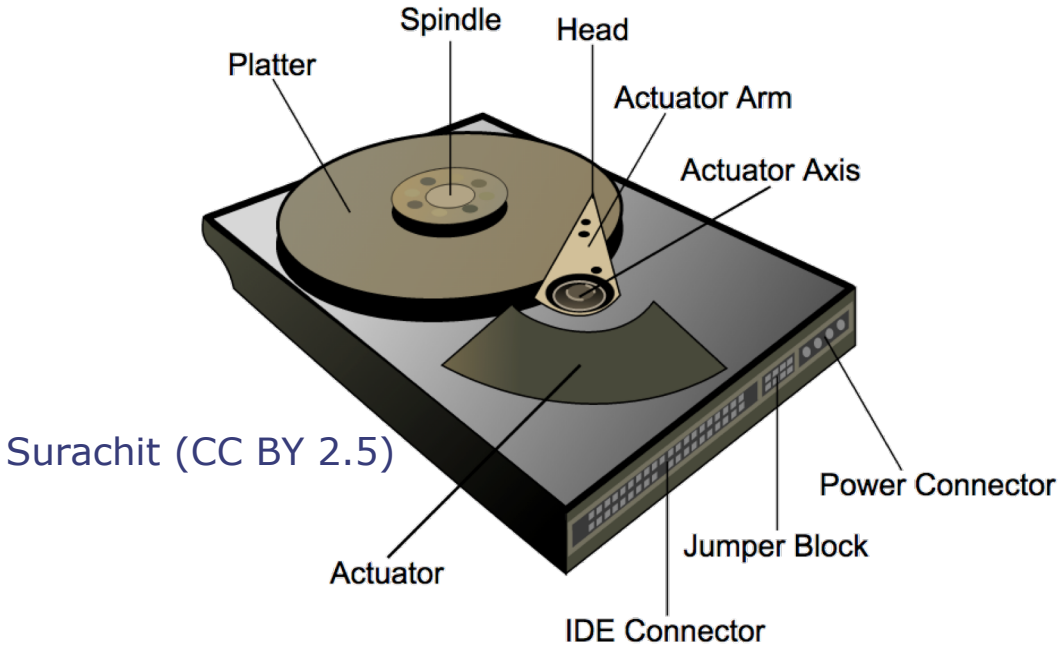
Electrons here diminish strength of field from control gate ⇒ no inversion ⇒ NFET stays off even when word line is high.

Cyferz (CC BY 2.5)

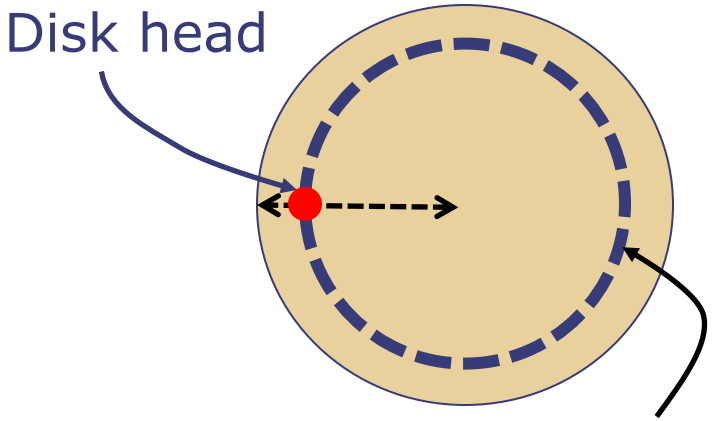
Flash Memory: Use "floating gate" transistors to store charge (floating gate is a well insulated conductor)

- Very dense: Multiple bits/transistor, read and written in blocks
- Slow (especially on writes), 10-100 us
- Limited number of writes: charging/discharging the floating gate (writes) requires large voltages that damage transistor

Non-Volatile Storage: Hard Disk



Surachit (CC BY 2.5)



Circular track divided into sectors

Hard Disk: Rotating magnetic platters + read/write head

- Extremely slow (~10ms): Mechanically move head to position, wait for data to pass underneath head
- ~100MB/s for sequential read/writes
- ~100KB/s for random read/writes
- Cheap

The Memory Hierarchy

Summary: Memory Technologies

	Capacity	Latency	Cost/GB
Register	100s of bits	20 ps	$$$$
SRAM	~10 KB-10 MB	1-10 ns	~$1000
DRAM	~10 GB	80 ns	~$10
Flash	~100 GB	100 us	~$1
Hard disk	~1 TB	10 ms	~$0.1

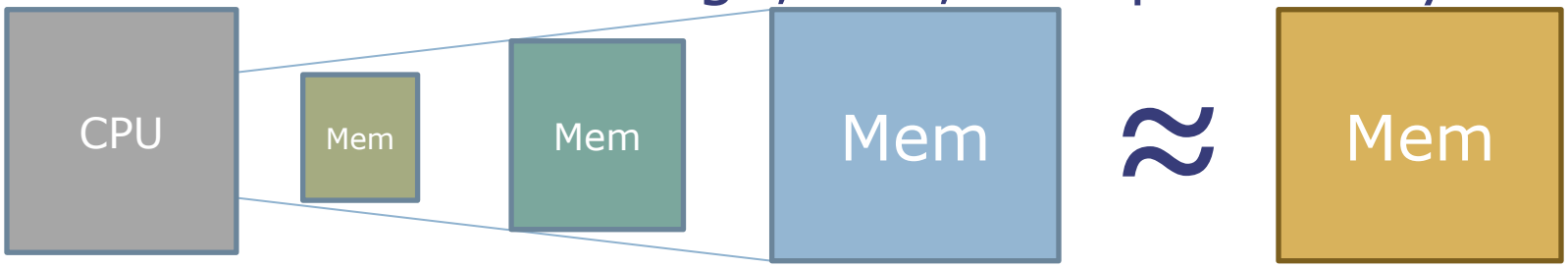
- Different technologies have vastly different tradeoffs
- Size is a fundamental limit, even setting cost aside:
  - Small + low latency, high bandwidth, low energy, or
  - Large + high-latency, low bandwidth, high energy
- Can we get best of both worlds? (large, fast, cheap)

The Memory Hierarchy

Want large, fast, and cheap memory, but…

- Large memories are slow (even if built with fast components)
- Fast memories are expensive

Solution: Use a hierarchy of memories with different tradeoffs to fake a large, fast, cheap memory

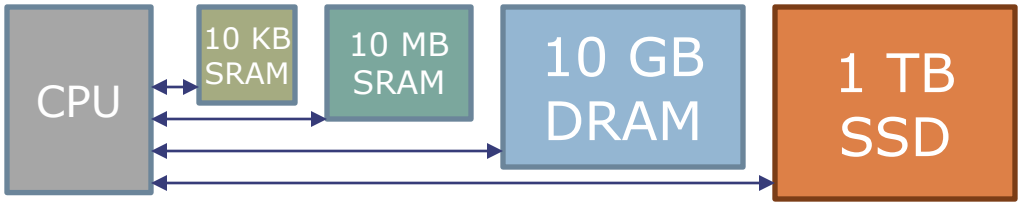


Speed:	Fastest	Slowest	Fast
Capacity:	Smallest	Largest	Large
Cost:	Highest	Lowest	Cheap

Memory Hierarchy Interface

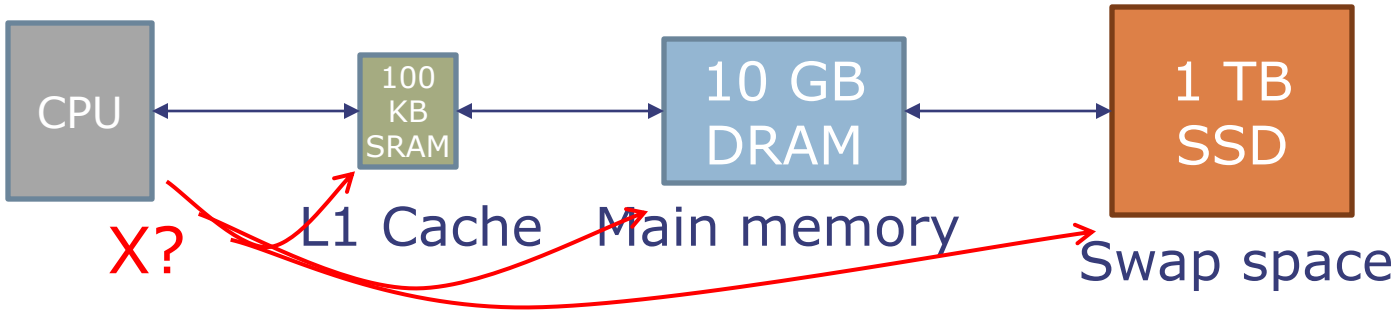
Approach 1: Expose Hierarchy

- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives
- Tell programmers: "Use them cleverly"

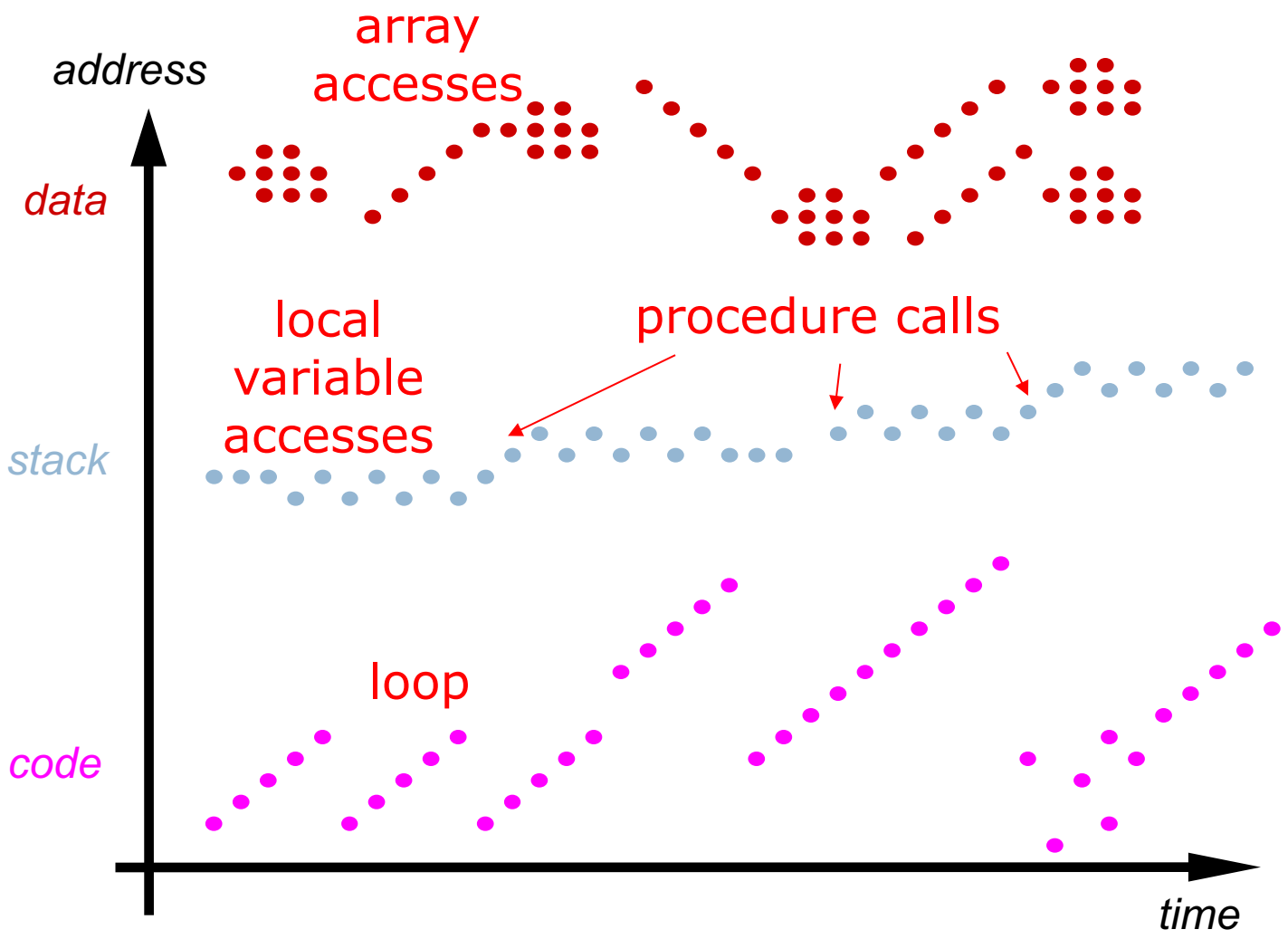


Approach 2: Hide Hierarchy

- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns



Typical Memory Access Patterns

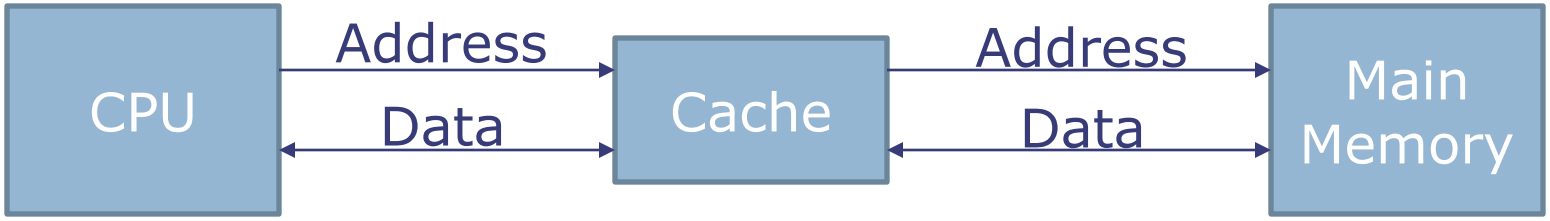


Common Predictable Patterns

- Two predictable properties of memory accesses:
  - Temporal locality: If a location has been accessed recently, it is likely to be accessed (reused) soon
  - Spatial locality: If a location has been accessed recently, it is likely that nearby locations will be accessed soon

Caches

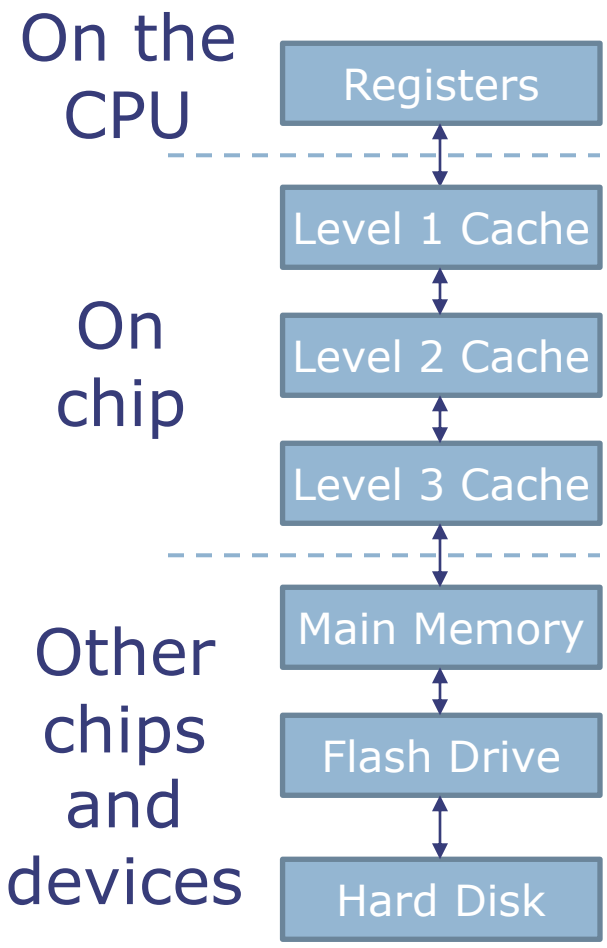
- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations



- Processor sends accesses to cache. Two options:
  - Cache hit: Data for this address in cache, returned quickly
  - Cache miss: Data not in cache
    - Fetch data from memory, send it back to processor
    - Retain this data in the cache (replacing some other data)
  - Processor must deal with variable memory access time

A Typical Memory Hierarchy

Computers use many levels of caches:



Access time	Capacity	Managed By
1 cycle	1 KB	Software/Compiler
2-4 cycles	32 KB	Hardware
10 cycles	256 KB	Hardware
40 cycles	10 MB	Hardware
200 cycles	10 GB	Software/OS
10-100us	100 GB	Software/OS
10ms	1 TB	Software/OS

Cache Metrics

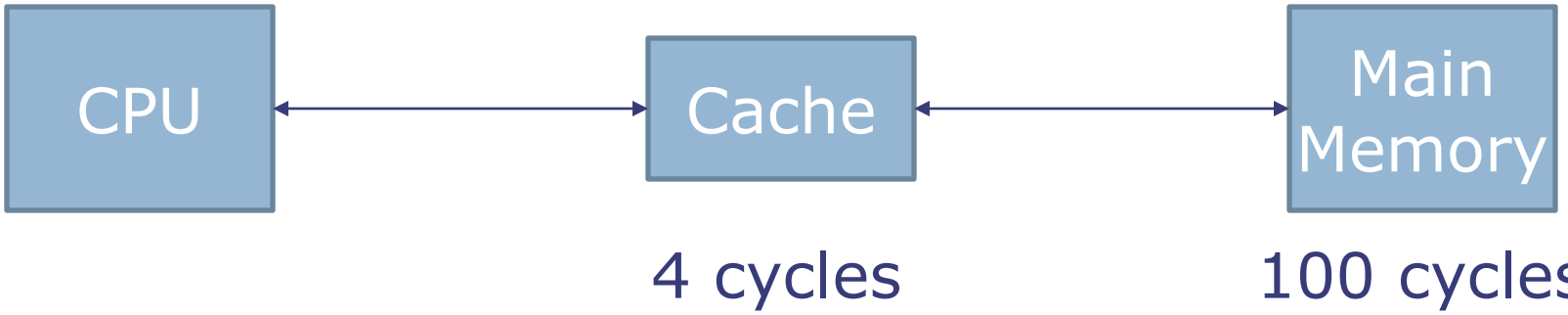
- Hit Ratio: $HR = \frac{hits}{hits + misses} = 1 - MR$
- Miss Ratio: $MR = \frac{misses}{hits + misses} = 1 - HR$
- Average Memory Access Time (AMAT):

  AMAT = HitTime + MissRatio × MissPenalty

  - Goal of caching is to improve AMAT
  - Formula can be applied recursively in multi-level hierarchies:

$$AMAT = HitTime_{L1} + MissRatio_{L1} \times AMAT_{L2} =$$
$$AMAT = HitTime_{L1} + MissRatio_{L1} \times (HitTime_{L2} + MissRatio_{L2} \times AMAT_{L3}) = \ldots$$

Example: How High of a Hit Ratio?



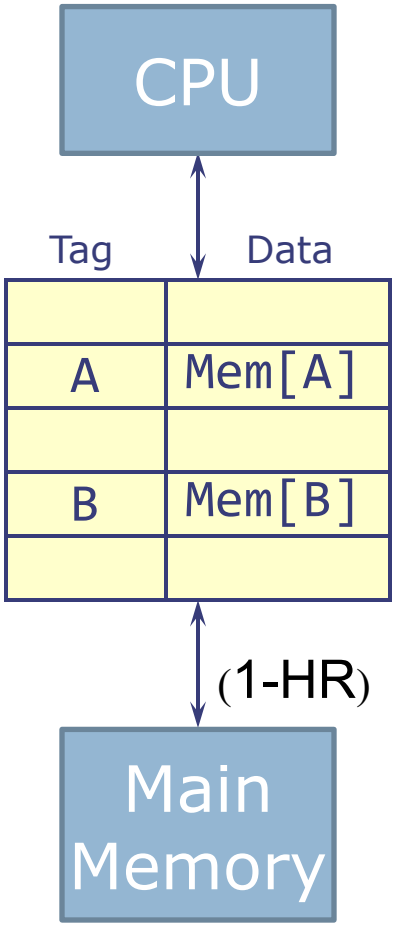
What hit ratio do we need to break even?
(Main memory only: AMAT = 100)

$$100 = 4 + (1 - HR) \times 100 \Rightarrow HR = 4\%$$

What hit ratio do we need to achieve AMAT = 5 cycles?

$$5 = 4 + (1 - HR) \times 100 \Rightarrow HR = 99\%$$

Basic Cache Algorithm (Reads)



On reference to Mem[X], look for X among cache tags

HIT: X = Tag(i) for some cache line i

Return Data(i)

MISS: X not found in Tag of any cache line

Read Mem[X]
Return Mem[X]
Select a line k to hold Mem[X]
Write Tag(k)=X, Data(k) = Mem[X]

Q: How do we "search" the cache?

Thank you!

Next lecture: Cache Tradeoffs