

# **ECE 550D**

## **Fundamentals of Computer Systems and Engineering**

### **Fall 2023**

## Memory Hierarchy

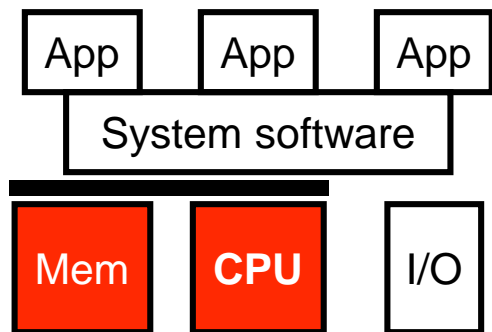
Xin Li & Dawei Liu  
Duke Kunshan University

Slides are derived from work by  
Andrew Hilton, Tyler Bletsch and Rabih Younes (Duke)

# Last Time

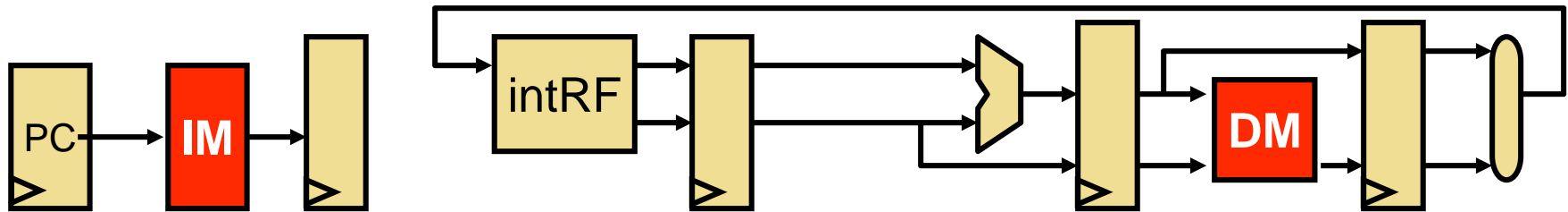
- What did we do last time?
- Pipelining
  - Pipeline control

# Memory Hierarchy



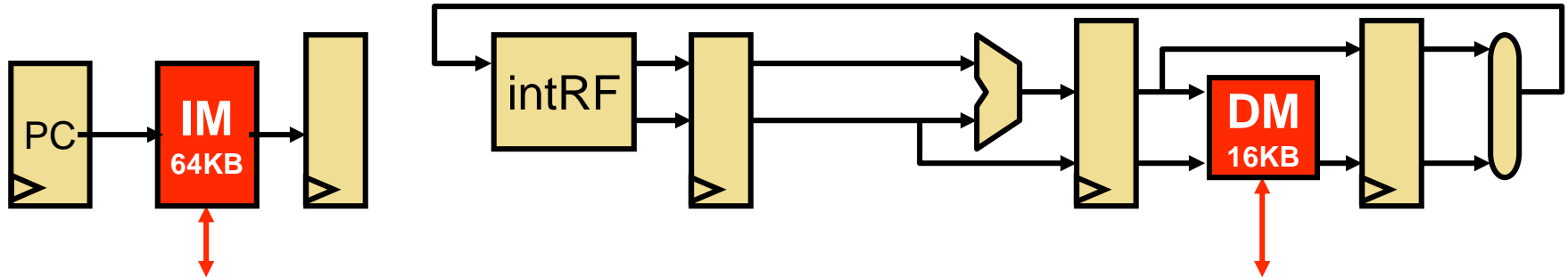
- Basic concepts
- Technology background
- Organizing a single memory component
  - ABC
  - Write issues
  - Miss classification and optimization
- Organizing an entire memory hierarchy
- Virtual memory
  - Highly integrated into real hierarchies, but...
  - ...won't talk about until later

# How Do We Build Insn/Data Memory?



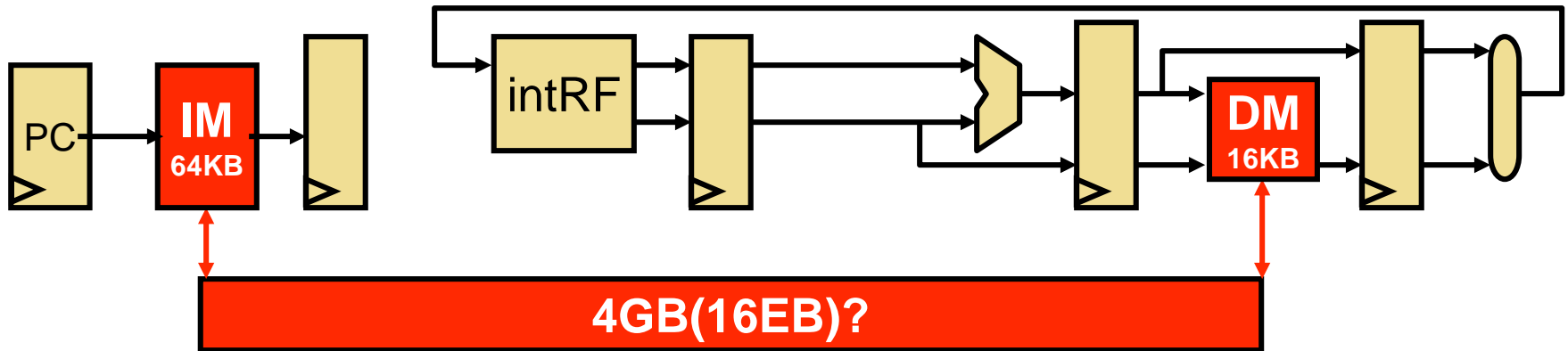
- Register file? Just a multi-ported SRAM
  - 32 32-bit registers = 1Kb = 128B
  - Multiple ports make it bigger and slower but still OK
- Insn/data memory? Just a single-ported SRAM?
  - Uh, umm... **it's  $2^{32}\text{B} = 4\text{GB}!!!!$** 
    - It would be huge, expensive, and pointlessly slow
    - And we can't build something that big on-chip anyway
  - Most ISAs now 64-bit, so memory is  $2^{64}\text{B} = 16\text{EB}$

# So What Do We Do? Actually...



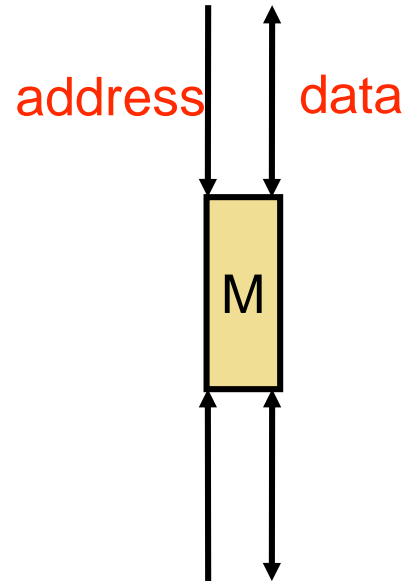
- “Primary” insn/data memory are single-ported SRAMs...
  - “primary” = “in the datapath”
  - Key 1: they contain only a dynamic subset of “memory”
    - Subset is small enough to fit in a reasonable SRAM
  - Key 2: missing chunks fetched on demand (transparent to program)
    - From somewhere else... (next slide)
- Program has illusion that all 4GB (16EB) of memory is physically there
- Just like it has the illusion that all insns execute atomically

# But...



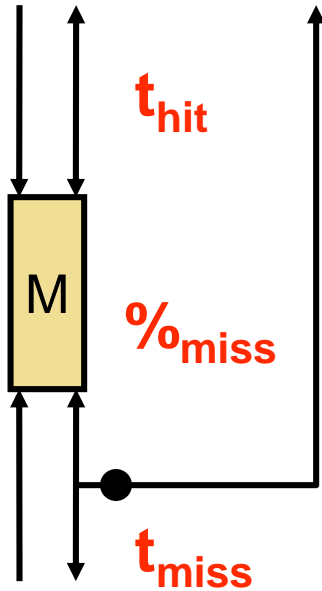
- If requested insn/data not found in primary memory
  - Doesn't the place it comes from have to be a 4GB (16EB) SRAM?
  - And won't it be huge, expensive, and slow? And can we build it?
  - We will answer these questions step by step

# Memory Overview



- Functionality
  - “Like a big array...”
  - N-bit **address** bus (on N-bit machine)
  - **Data** bus: typically read/write on same bus
  - Can have multiple **ports**: address/data bus pairs
- Access time:
  - Access latency  $\sim \text{\#bits} * \text{\#ports}^2$

# Memory Performance Equation



- For memory component M
    - **Access**: read or write to M
    - **Hit**: desired data found in M
    - **Miss**: desired data not found in M
      - Must get from another component
      - No notion of “miss” in register file
    - **Fill**: action of placing data in M
  - $\%_{miss}$  (miss-rate): #misses / #accesses
  - $t_{hit}$ : time to read data from (write data to) M
  - $t_{miss}$ : time to read data into M
- 
- Performance metric: average access time

$$t_{avg} = t_{hit} + \%_{miss} * t_{miss}$$



# Memory Hierarchy

$$t_{avg} = t_{hit} + \%_{miss} * t_{miss}$$

- Problem: hard to get low  $t_{hit}$  and  $\%_{miss}$  in one structure
  - Large structures have low  $\%_{miss}$  but high  $t_{hit}$
  - Small structures have low  $t_{hit}$  but high  $\%_{miss}$
- Solution: use a hierarchy of memory structures
  - Known from the very beginning

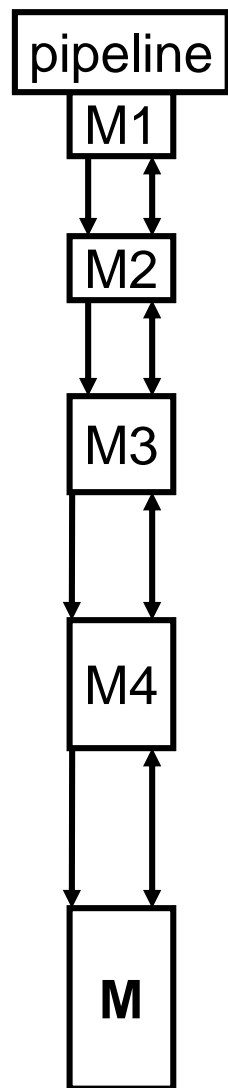
“Ideally, one would desire an infinitely large memory capacity such that any particular word would be immediately available ... We are forced to recognize the possibility of constructing a hierarchy of memories, each of which has a greater capacity than the preceding but which is less quickly accessible.”

Burks, Goldstine, VonNeumann

“Preliminary discussion of the logical design of an electronic computing instrument”

IAS memo 1946

# Abstract Memory Hierarchy



- Hierarchy of memory components
  - Upper levels: small  $\rightarrow$  low  $t_{hit}$ , high  $\%_{miss}$
  - Going down: larger  $\rightarrow$  higher  $t_{hit}$ , lower  $\%_{miss}$
- Connected by buses
  - Ignore for the moment
- Make average access time close to M1's
  - How?
  - Most frequently accessed data in M1
  - M1 + next most frequently accessed in M2, etc.
  - **Automatically** move data up/down hierarchy

# Why Memory Hierarchy Works

- **10/90 rule (of thumb)**

- 10% of static insns/data account for 90% of accessed insns/data
  - Insns: inner loops
  - Data: frequently used globals, inner loop stack variables

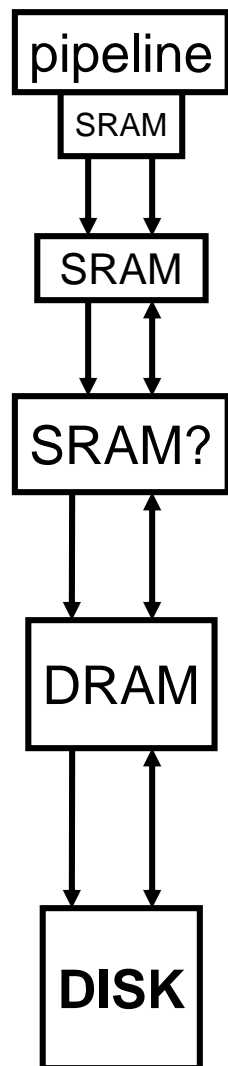
- **Temporal locality**

- Recently accessed insns/data likely to be accessed again soon
  - Insns: inner loops (next iteration)
  - Data: inner loop local variables, globals
- Hierarchy can be **"reactive"**: move things up when accessed

- **Spatial locality**

- Insns/data near recently accessed insns/data likely accessed soon
  - Insns: sequential execution
  - Data: elements in array, fields in struct, variables in stack frame
- Hierarchy is **"proactive"**: move things up speculatively

# Exploiting Heterogeneous Technologies



- Apparent problem
  - Lower level components must be huge
  - Huge SRAMs are difficult to build and expensive
- Solution: don't use SRAM for lower levels
  - **Cheaper, denser storage technologies**
  - Will be slower than SRAM, but that's OK
    - Won't be accessed very frequently
    - We have no choice anyway
  - Upper levels: SRAM → expensive (/B), fast
  - Going down: DRAM, Disk → cheaper (/B), slow

# Memory Technology Overview

- **Latency**

- SRAM: <1 to 5ns (on chip)
- DRAM: ~100ns — 100x or more slower than SRAM
- Disk: 10,000,000ns or 10ms — 100,000x slower than DRAM
- Flash: ~200ns — 2x slower than DRAM (read, much slower writes)

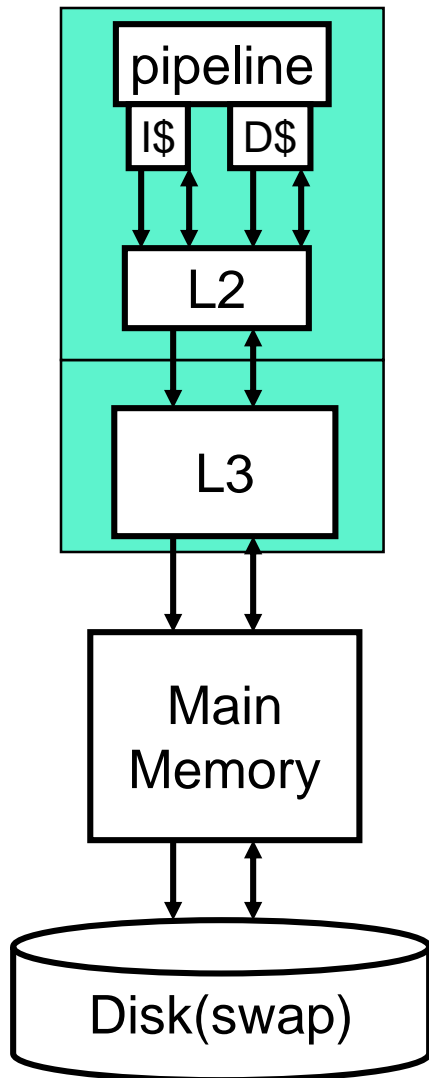
- **Bandwidth**

- SRAM: 10-100GB/sec
- DRAM: ~1GB/sec — 10x less than SRAM
- Disk: 100MB/sec (0.1 GB/sec) — sequential access only
- Flash: about same as DRAM for read (much less for writes)

- **Cost:** what can \$300 buy ~~today~~ a few years ago?

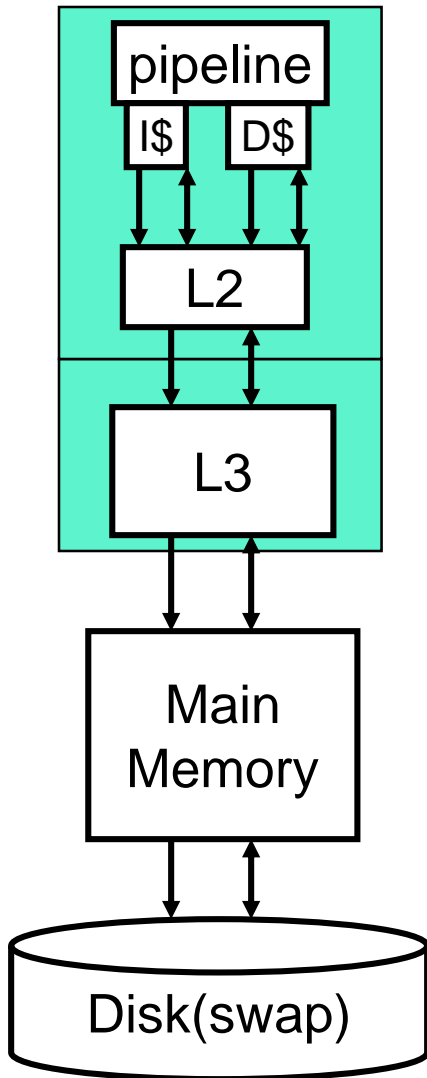
- SRAM: 4MB
- DRAM: 1,000MB (1GB) — 250x cheaper than SRAM
- Disk: 400,000MB (400GB) — 400x cheaper than DRAM
- Flash: 4,000 MB (4GB) — 4x cheaper than DRAM

# (Traditional) Concrete Memory Hierarchy



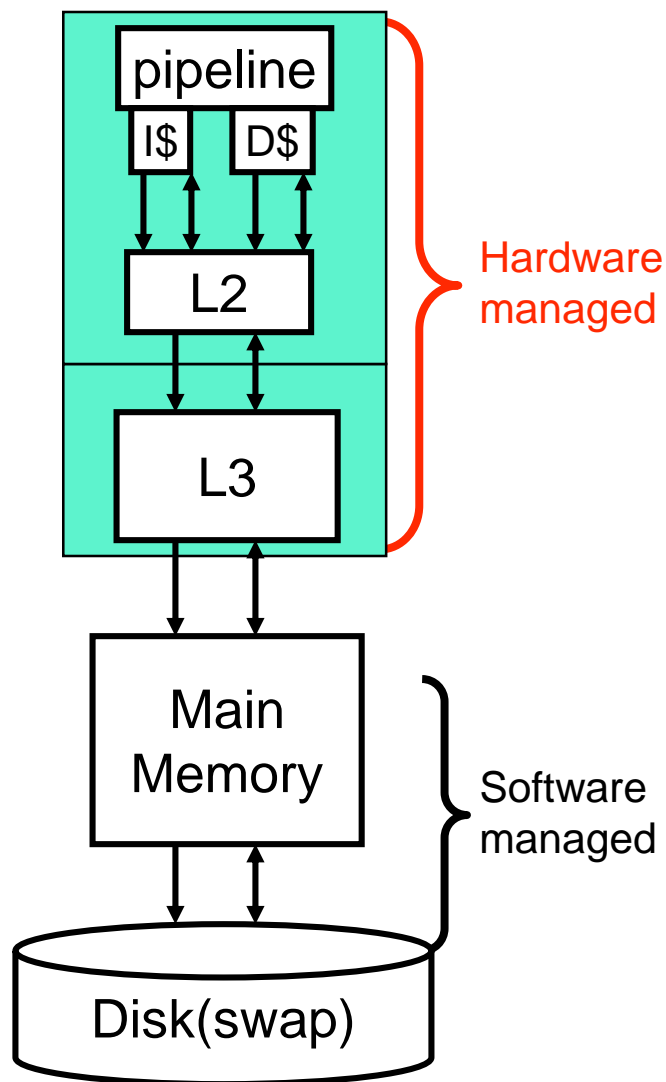
- 1st level: **I\$, D\$** (insn/data caches)
- 2nd level: **L2** (cache)
  - On-chip
  - Made of SRAM (or embedded DRAM)
- 3rd level: **L3** (cache)
  - Same as L2, on-chip
- N-1 level: **main memory**
  - Off-chip
  - Made of DRAM
- N level: **disk (swap space)**
  - Electrical-mechanical

# Virtual Memory Teaser



- For 32-bit ISA
  - 4GB disk is easy
  - Even 4GB main memory is common
- For 64-bit ISA
  - 16EB main memory is right out
  - Even 16EB disk is extremely difficult
- Virtual memory
  - Never referenced addresses don't have to physically exist anywhere!
  - Future lecture...

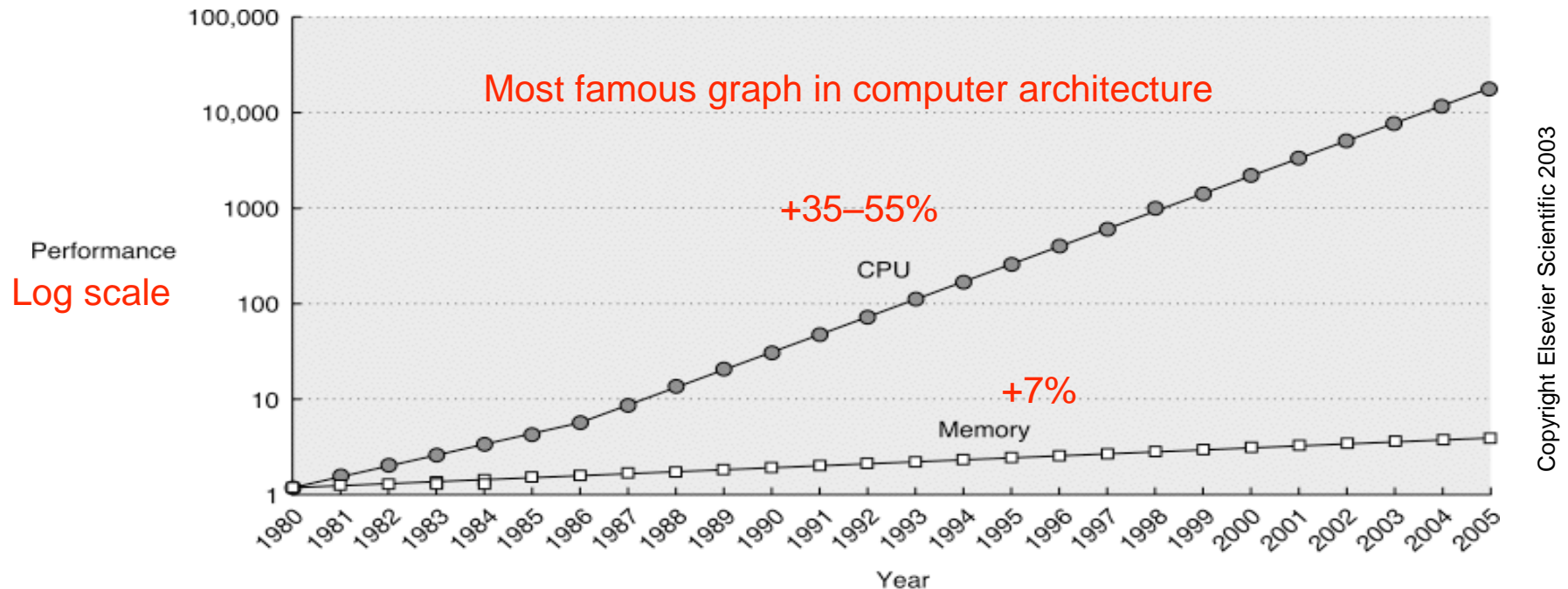
# Start With “Caches”



- “Cache”: hardware managed
  - Missing chunks retrieved by hardware
- SRAM technology
  - Technology basis of latency
- Cache organization
  - ABC
  - Miss classification & optimization
  - What about writes?
- Cache hierarchy organization
- Some example calculations

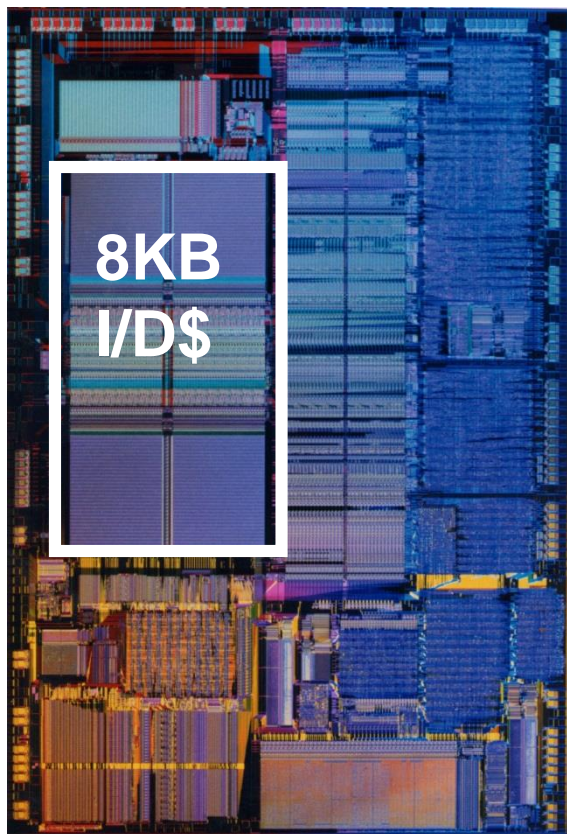


# Why Are There 2-3 Levels of Cache?

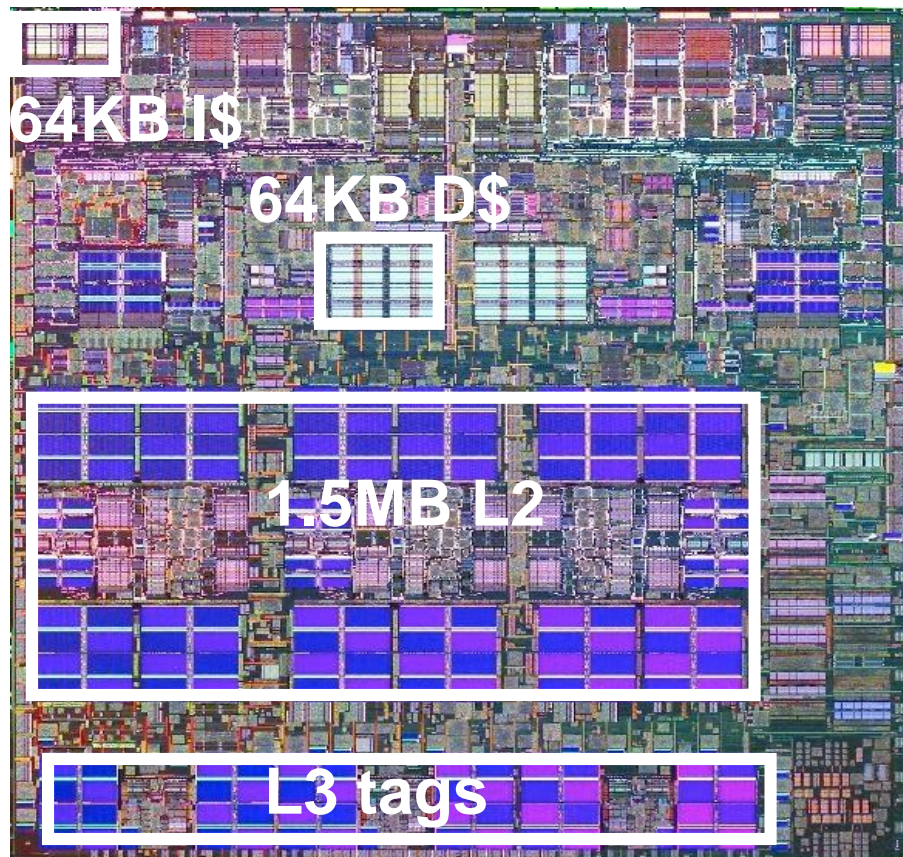


- **"Memory Wall":** DRAM 100X slower than primary caches
  - Multiple levels of cache needed to bridge the difference
- **"Disk Wall?":** disk is 100,000X slower than memory
  - Why aren't there 56 levels of main memory to bridge that difference?
  - Doesn't matter: program can't keep itself busy for 10M cycles
  - So slow, may as well swap out and run another program

# Evolution of Cache Hierarchies



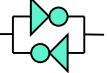
Intel 486



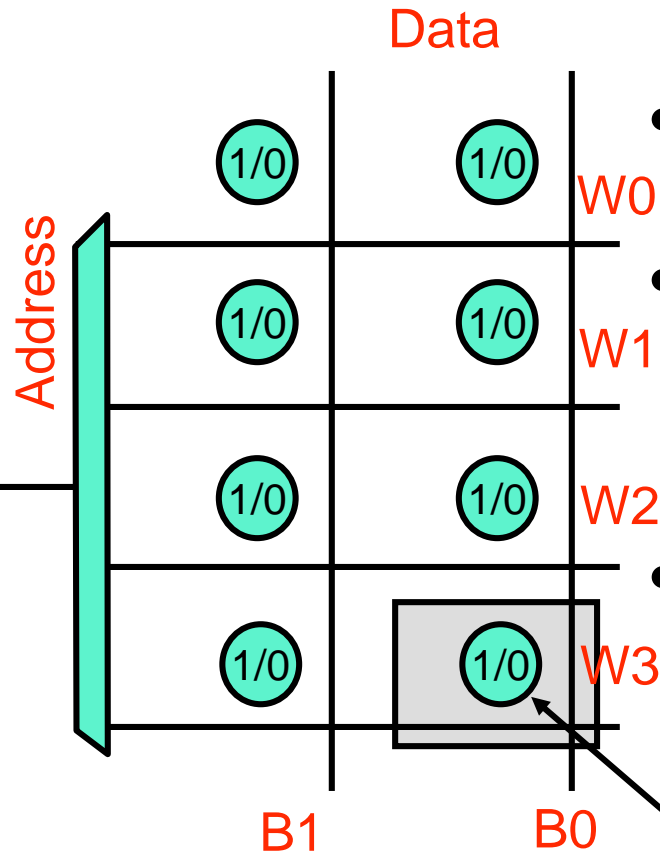
IBM Power5 (dual core)

- Chips today are 30–70% cache by area

# RAM and SRAM

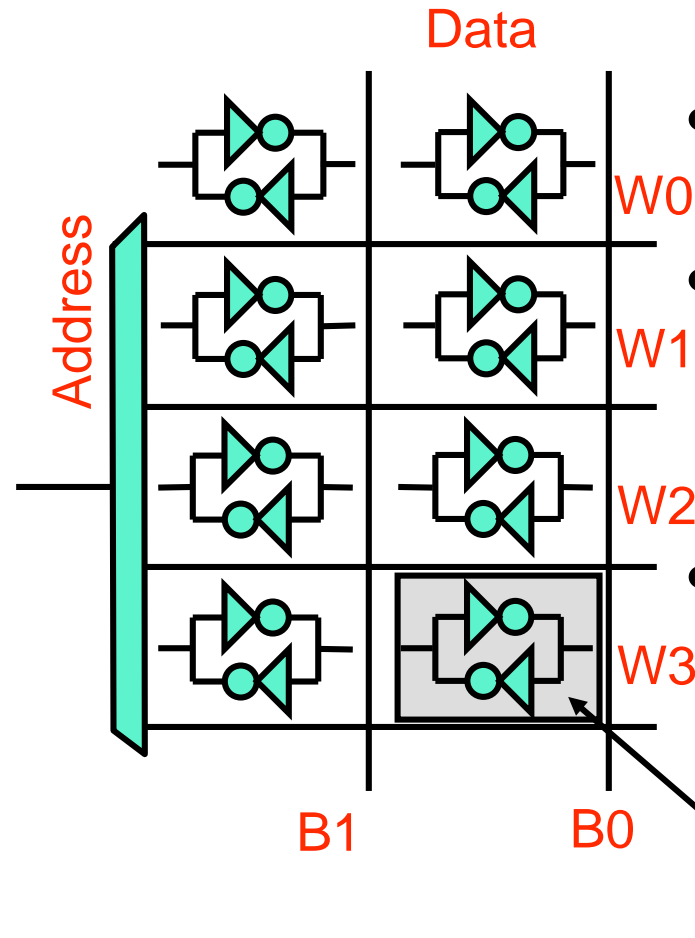
- Reality: large storage arrays implemented in “analog” way
  - Not as flip-flops (FFs) + giant muxes
- **RAM (random access memory)**
  - Ports implemented as shared buses called wordlines/bitlines
- **SRAM: static RAM**
  - Static = bit maintains its value indefinitely, as long as power is on
  - Bits implemented as **cross-coupled inverters (CCIs)** 
    - + 2 gates, 4 transistors per bit
  - All processor storage arrays: regfile, caches, branch predictor, etc.
- Other forms of RAM: Dynamic RAM (DRAM), Flash

# Basic RAM



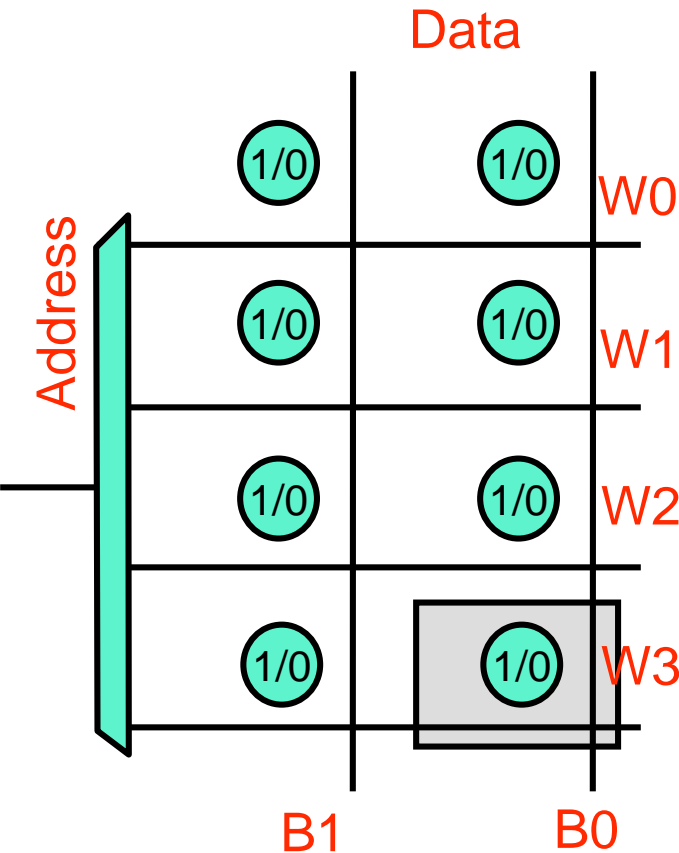
- Storage array
  - M words of N bits each (e.g., 4w, 2b each)
- RAM storage array
  - M by N array of "bits" (e.g., 4 by 2)
- RAM port
  - Grid of wires that overlays bit array
  - **M wordlines**: carry 1H decoded address
  - **N bitlines**: carry data
- RAM port operation
  - Send address → 1 wordline goes high
  - "bits" on this line read/write bitline data
  - Operation depends on bit/W/B connection
    - "Magic" analog stuff

# Basic SRAM



- Storage array
  - M words of N bits each (e.g., 4w, 2b each)
- SRAM storage array
  - M by N array of CCI's (e.g., 4 by 2)
- SRAM port
  - Grid of wires that overlays CCI array
    - **M wordlines**: carry 1H decoded address
    - **N bitlines**: carry data
- SRAM port operation
  - Send address → 1 wordline goes high
  - CCIs on this line read/write bitline data
  - Operation depends on CCI/W/B connection
    - “Magic” analog stuff

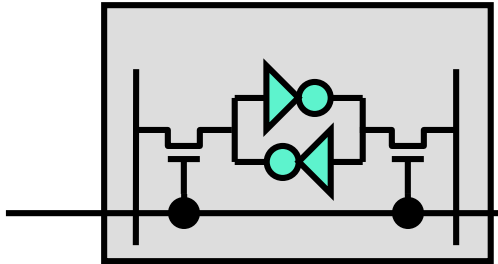
# ROMS:



- ROMs = Read Only memory
- Similar layout (wordlines, bitlines) to RAMs
  - Except not writeable: fixed connections to Power/Gnd instead of CCI
- Also EPROMs
  - Programmable once electronically
- And EEPROMs
  - Erasable and re-programable (very slow)

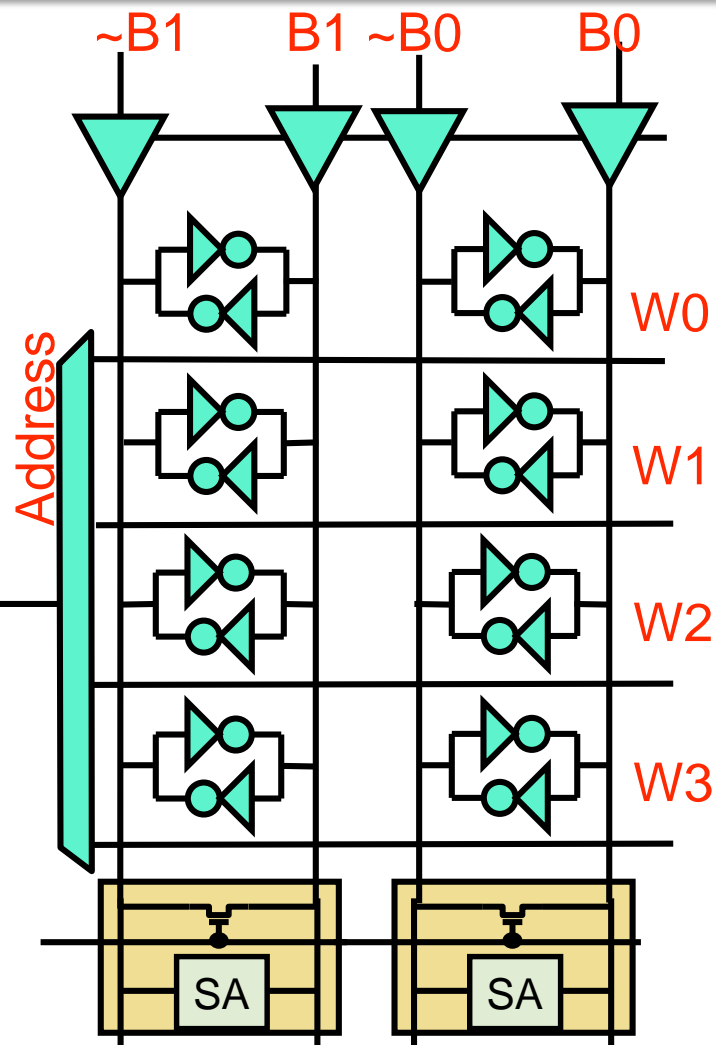
# SRAM Read/Write Port

- Cache: read/write on same port
  - Not at the same time
  - Trick: two bitlines
    - “Double-ended” or “differential” bitlines
  - Smaller → faster than separate ports





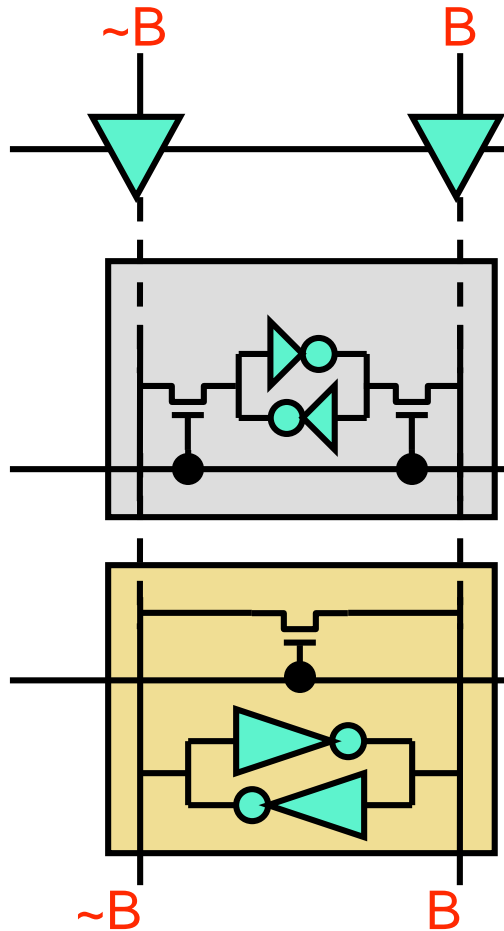
# SRAM Read/Write



- Some extra logic on the edges
  - To write: tristates "at the top"
    - Drive write data when appropriate
  - To read: 2 things at the bottom
    - Ability to equalize bit lines
    - Sense amps

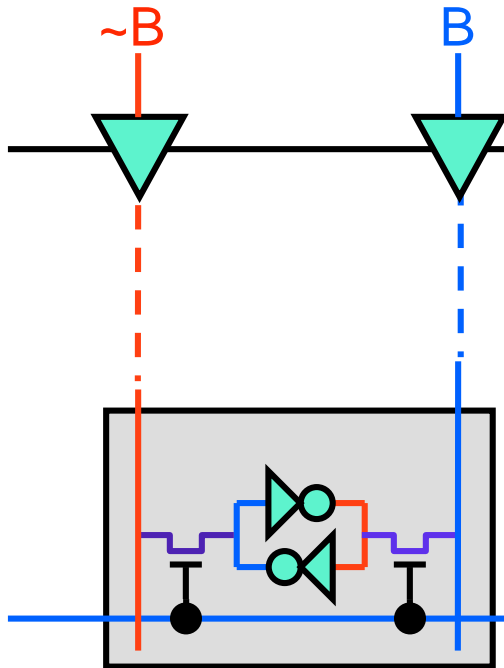


# SRAM Read/Write Port



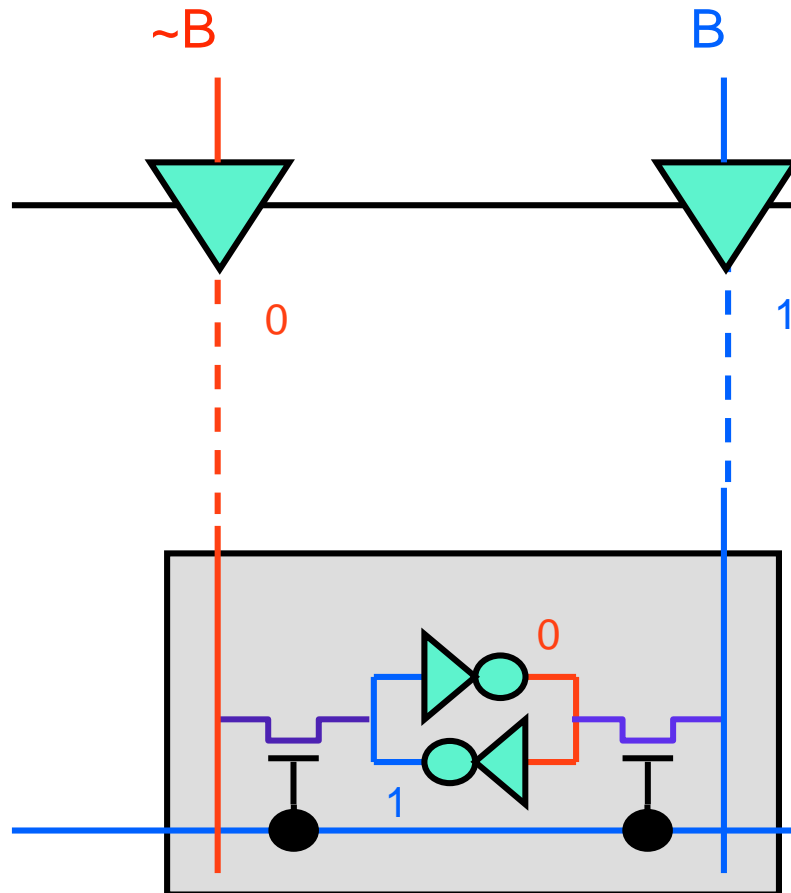
- Write process
  - Drive  $B$  and  $\sim B$  onto bit lines
  - Open transistors to access CCI
  - Done for "row" by one-hot word line

# SRAM Read/Write Port

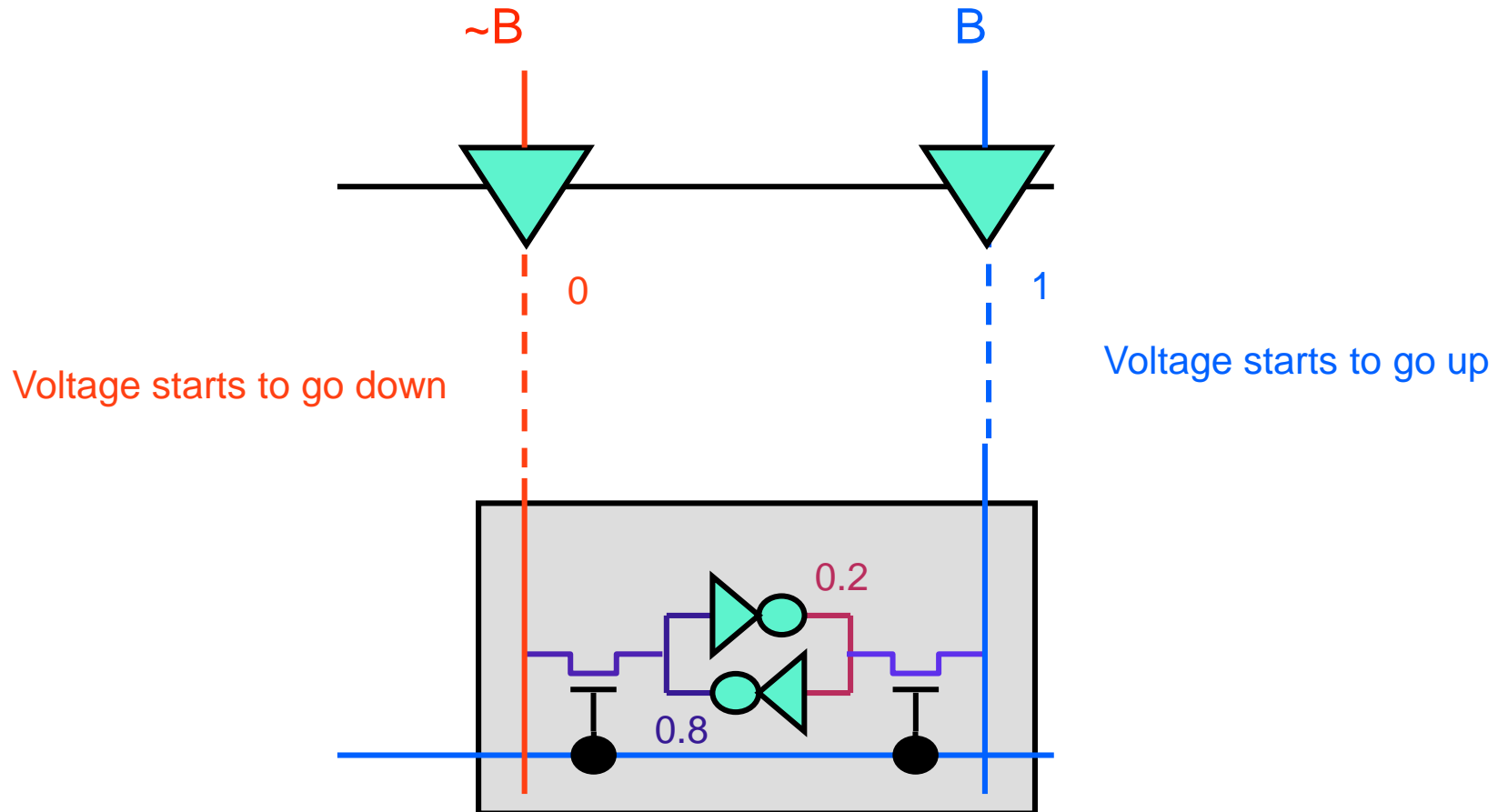


- Write process
  - Drive  $B$  and  $\sim B$  onto bit lines
  - Open transistors to access CCI
  - Done for "row" by one-hot word line
- Example on left:
  - Storing a 0 (" $B$ " side CCI has 0)
  - Writing a 1 (" $B$ " side bit-line is 1)
- Short Circuit??
  - CCI is driving a 0
  - Tri-state is driving a 1
  - (Opposite on  $\sim B$  side)
- Yes, briefly, but its ok...

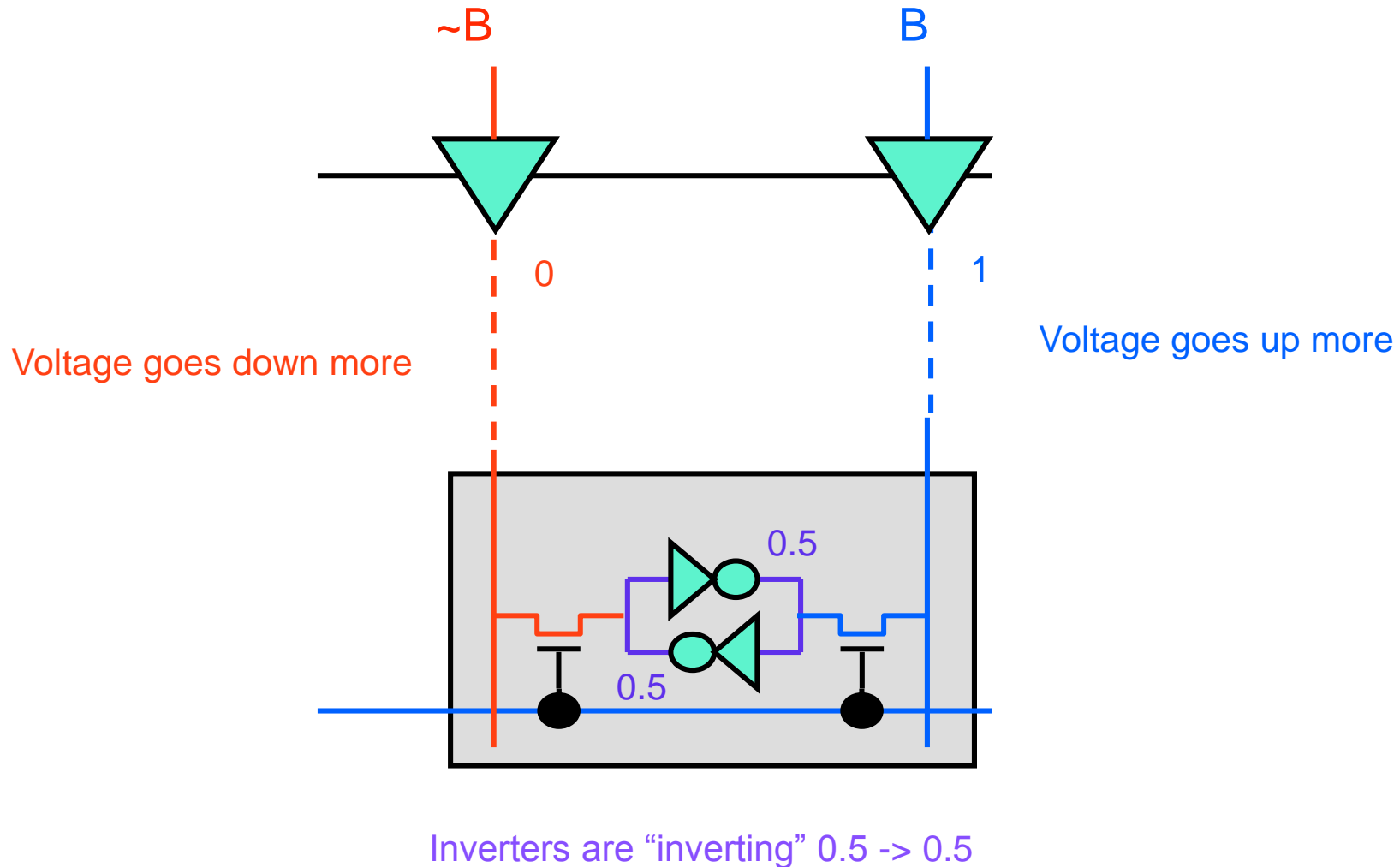
# SRAM Read/Write Port



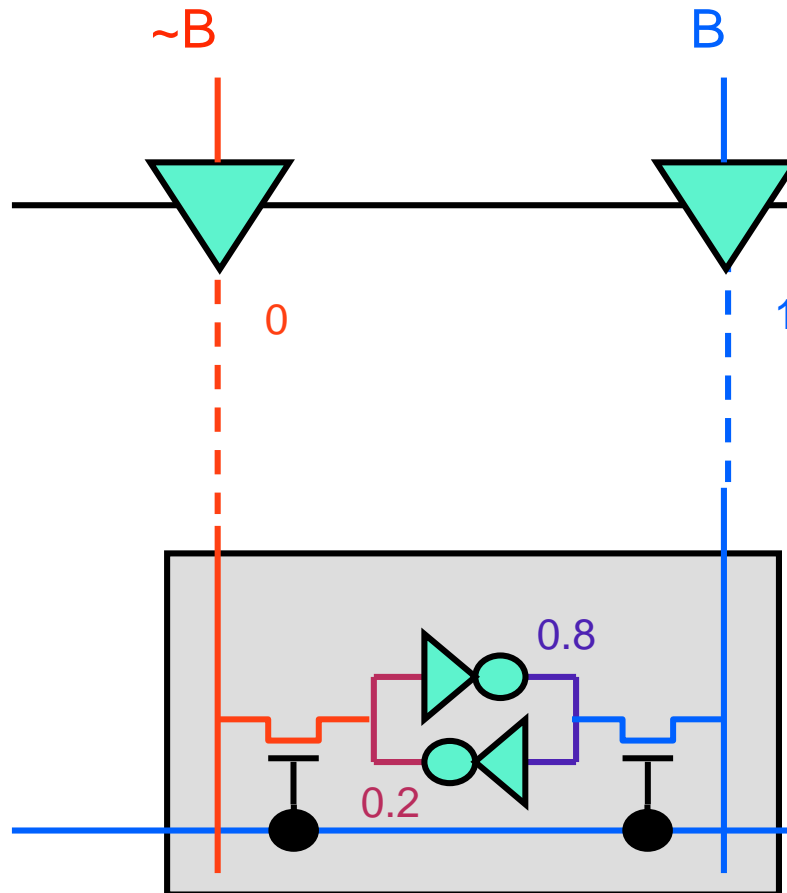
# SRAM Read/Write Port



# SRAM Read/Write Port

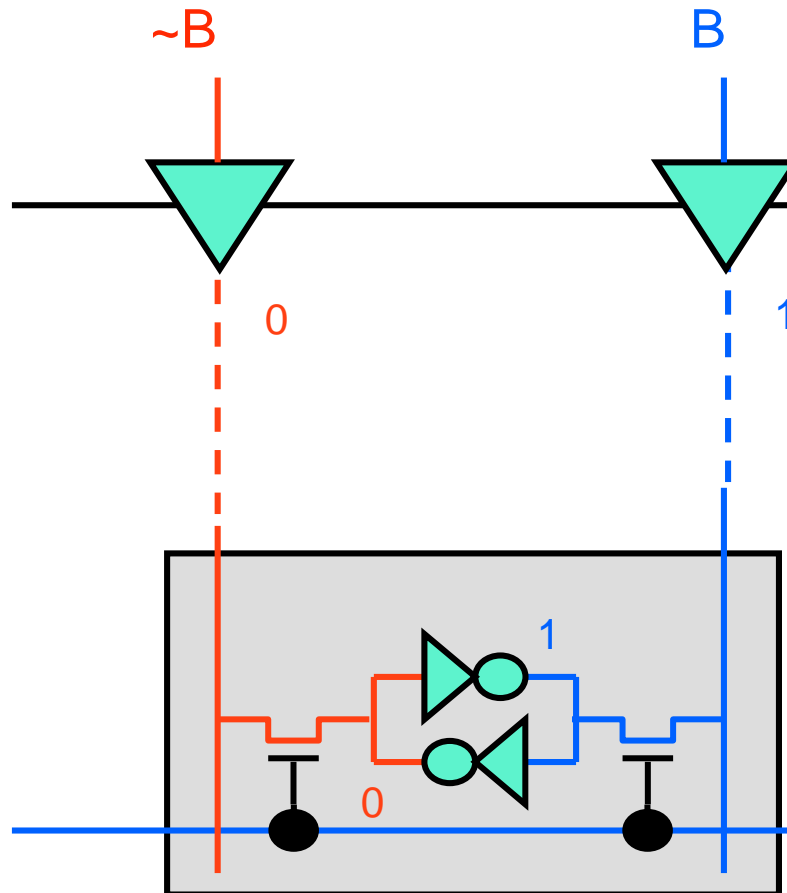


# SRAM Read/Write Port



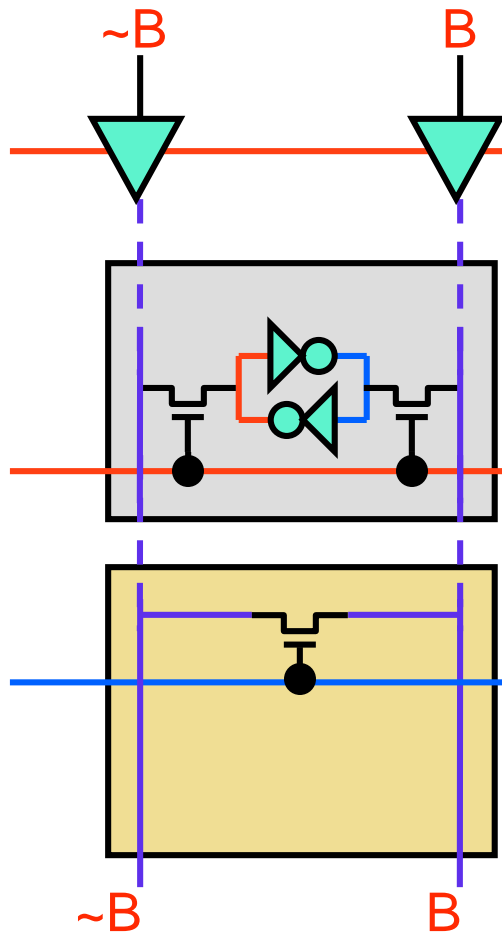
Past half-way point, inverters are “helping”

# SRAM Read/Write Port



Eventually (hundreds of pico-sec) reach 0/1 state

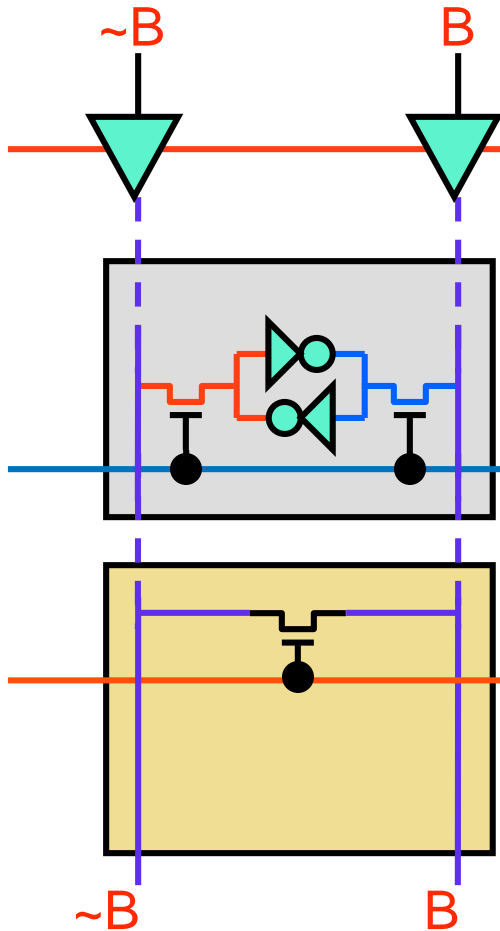
# SRAM Read/Write Port



- Now, reading: bit we will read holds 1
- Two steps to read: half clock each
  - First-half clock: equalize bit lines to **0.5**

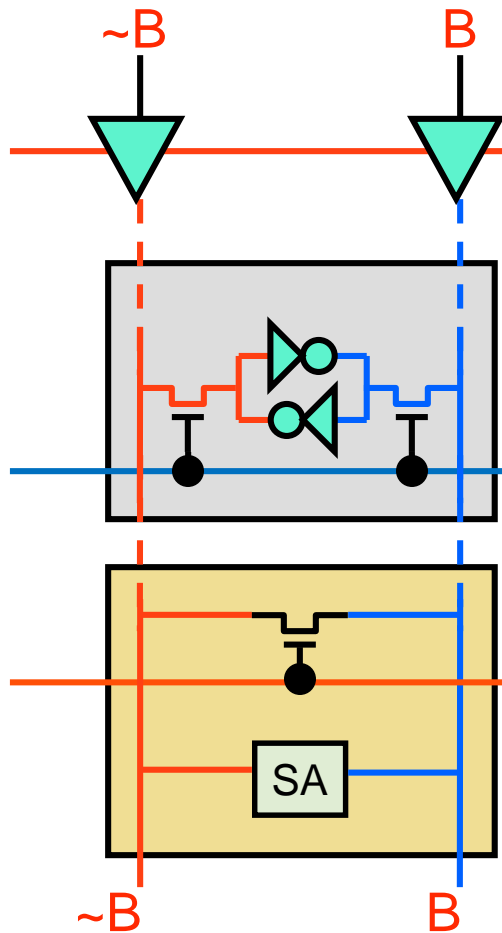


# SRAM Read/Write Port



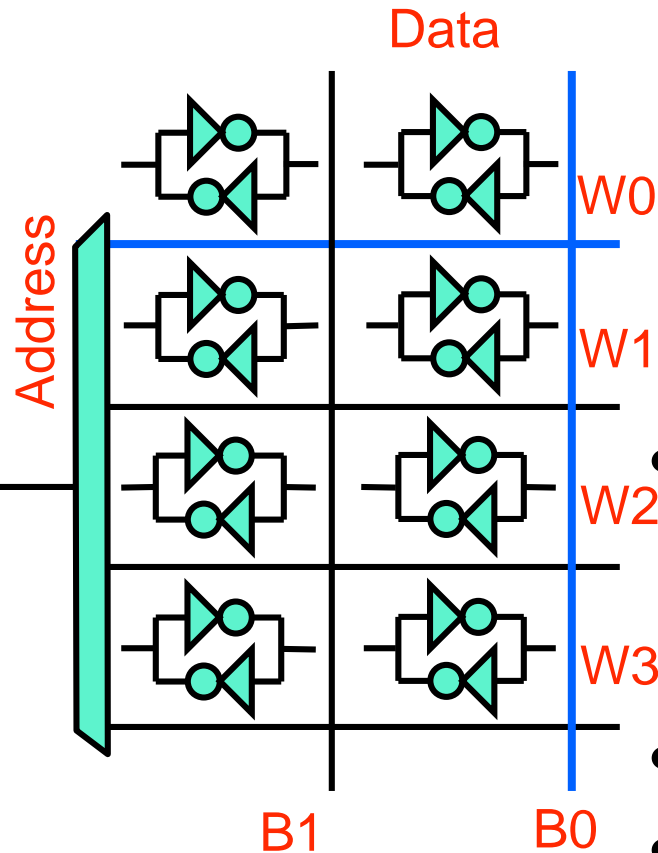
- Now, reading: bit we will read holds 1
- Last activity wrote or read 0
- Two steps to read: half clock each
  - First-half clock: equalize bit lines to **0.5**
  - Second-half clock:
    - Close equalizing transistor
    - Open r/w transistor
      - CCI is now driving bit-line
      - Bit-line slowly gets to right value

# SRAM Read/Write Port



- Now, reading: bit we will read holds 1
- Last activity wrote or read 0
- Two steps to read: half clock each
  - Low clock: equalize bit lines to **0.5**
  - High clock:
    - Close equalizing transistor
    - Open r/w transistor
      - CCI is now driving bit-line
      - Bit-line slowly gets to right value
  - Sense-amps speedup up process
    - Amplify voltage swing

# SRAM Latency



- Physics 102:
  - Delay  $\sim$  Resistance \* Capacitance
  - Wires have capacitance
    - Proportional to their length
  - Wires have resistance
    - Proportional to their length
  - Delay  $\sim$  (Length)<sup>2</sup>
- SRAM latency:
  - Word line latency + bit line latency
  - Word line length: "width"
  - Bit line length: "height"
- Word line length: columns \* ports
- Bit line length: rows \* ports
- Each port has its own WL and BL

# SRAM Latency

- Physics 102:
  - Delay  $\sim R * C$
  - Wires have capacitance
    - Proportional to their length
  - Wires have resistance
    - Proportional to their length
  - Delay  $\sim (\text{Length})^2$
- SRAM latency:
  - Word line latency + bit line latency
  - Word line length: "width"
  - Bit line length: "height"
- Word line length: columns \* ports
- Bit line length: rows \* ports

$$(c * p)^2 + (r * p)^2$$

$$c^2 p^2 + r^2 p^2$$

$$p^2 (c^2 + r^2)$$

Optimize RAM with  $c = r$

$$p^2 (2c^2)$$

$c^2$  is "number of bits"  
(square of side  $c$ )

$$\mathbf{ports^2 * numBits}$$