

计算机组成与系统结构

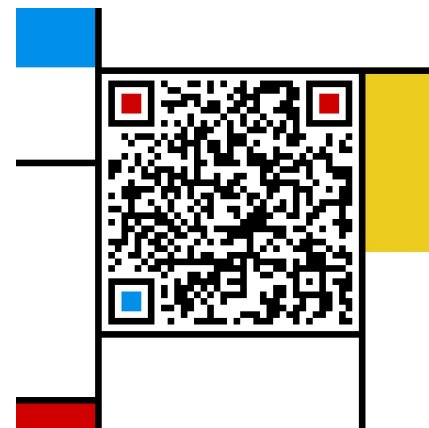
Computer Organization & System Architecture

Huang Kejie (黄科杰) 百人计划研究员

Office: 玉泉校区老生仪楼 304

Email address: huangkejie@zju.edu.cn

HP: 17706443800



New-School Machine Structures

- **Parallel Requests**
Assigned to computer
e.g., Search “Katz”
- **Parallel Threads**
Assigned to core
e.g., Lookup, Ads
- **Parallel Instructions**
>1 instruction @ one time
e.g., 5 pipelined instructions
- **Parallel Data**
>1 data item @ one time
e.g., Add of 4 pairs of words
- **Hardware descriptions**
All gates @ one time
- **Programming Languages**

Software

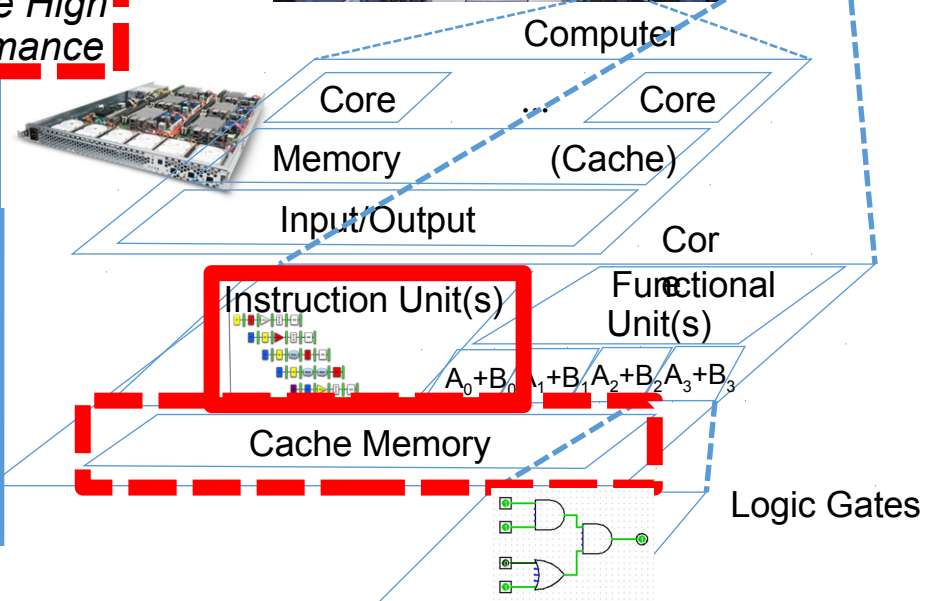
Hardware

*Harness
Parallelism &
Achieve High
Performance*

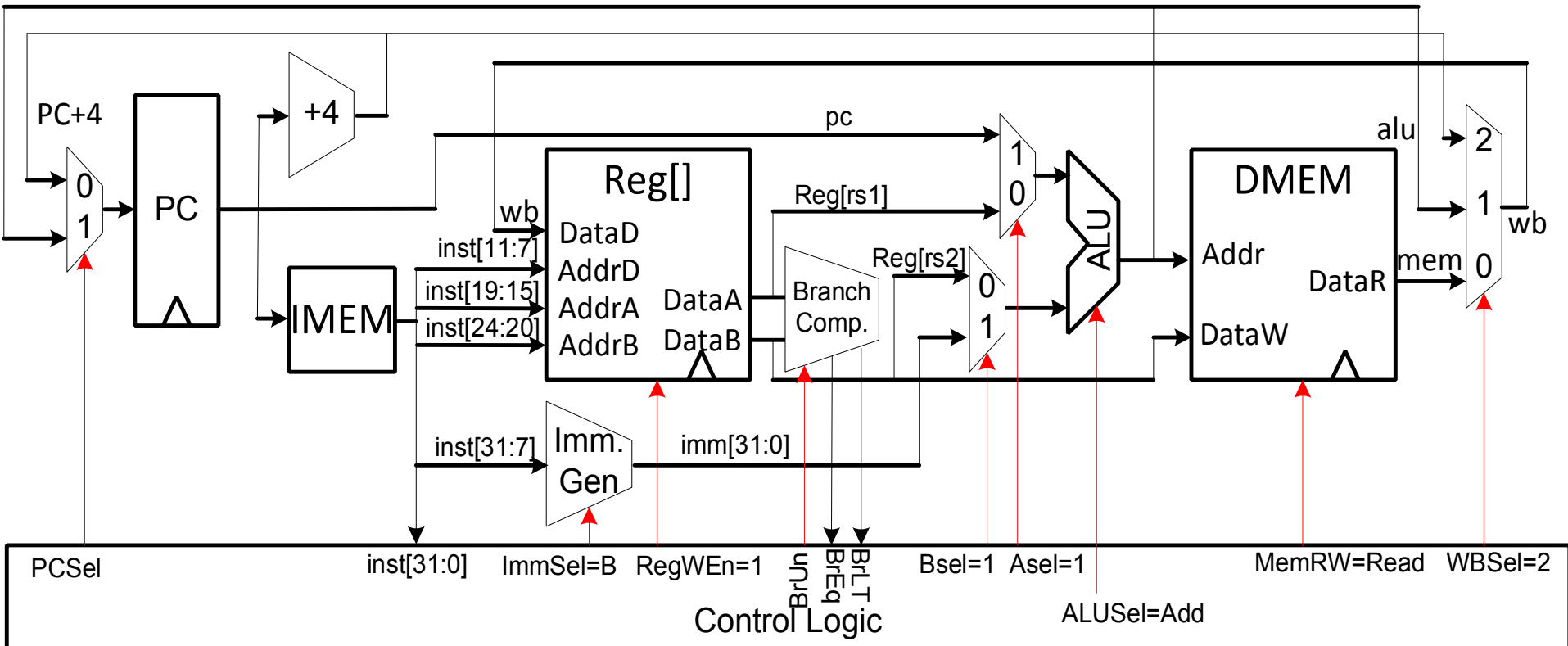
Warehouse
e Scale
Computer



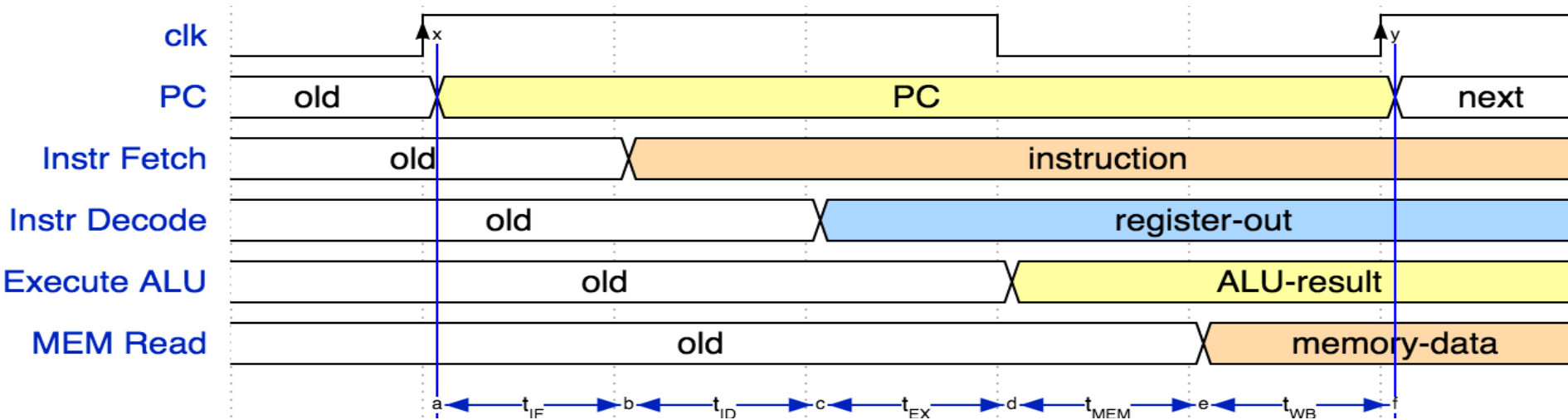
Smart
Phone



Review: Single-Cycle RISC-V RV32I Datapath

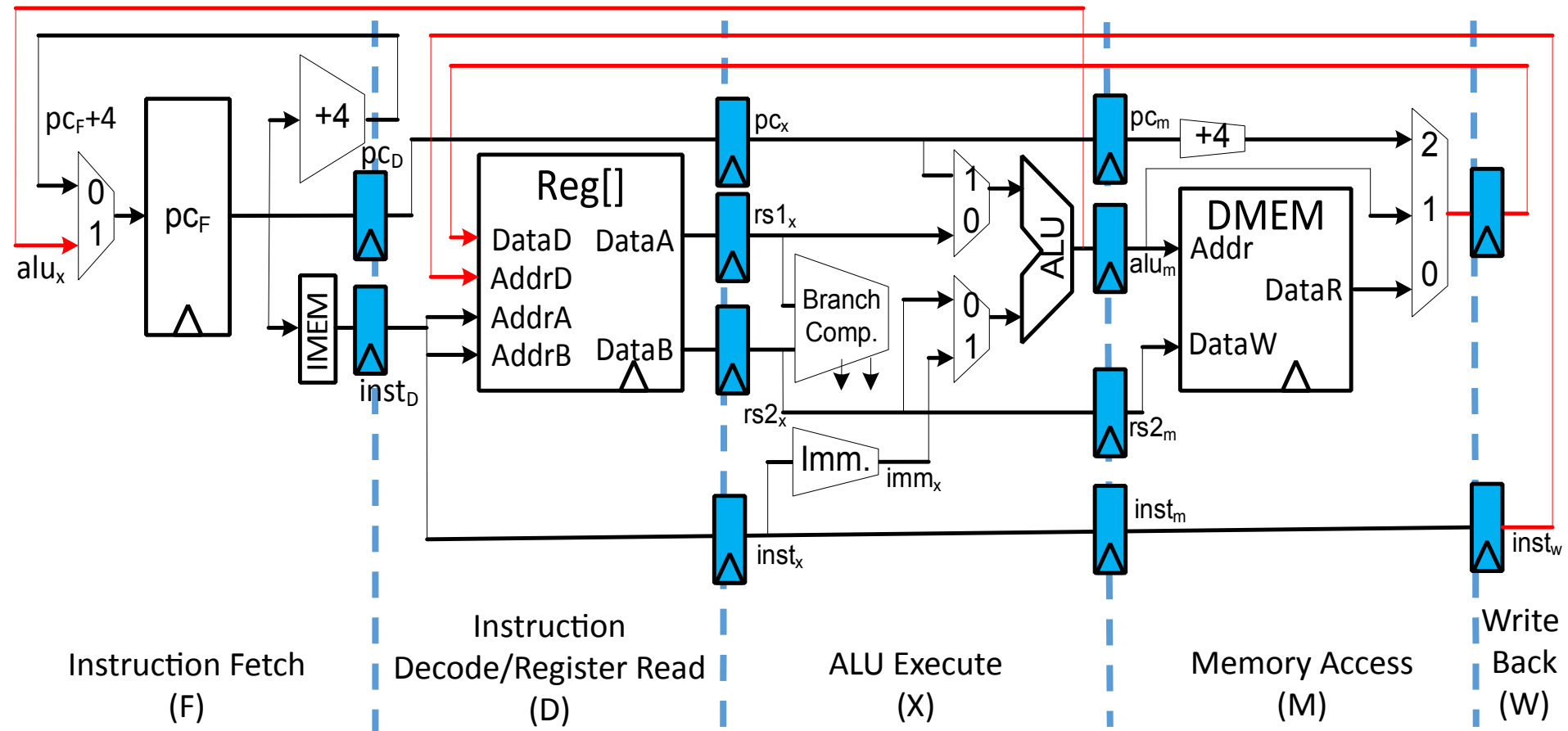


Review: Single Cycle Instruction Timing

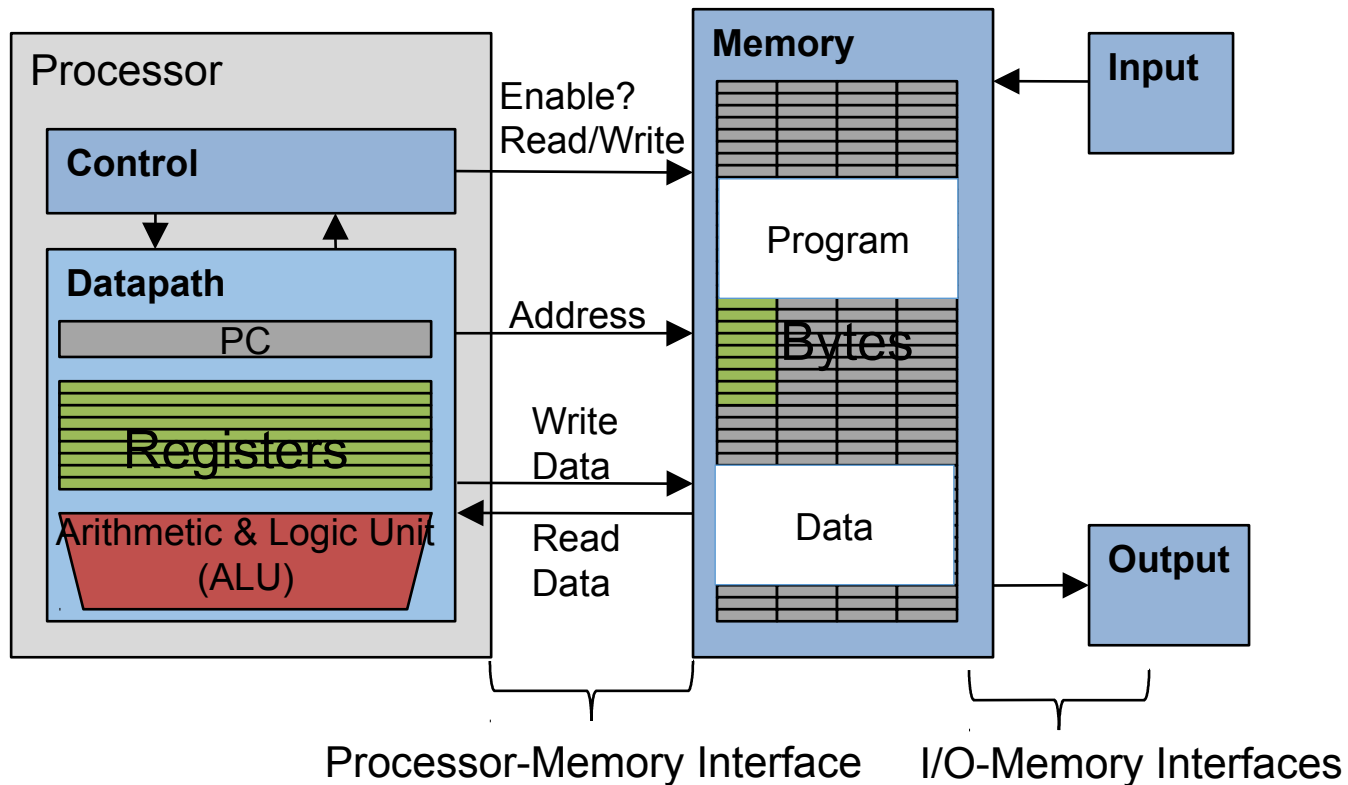


IF	ID	EX	MEM	WB	Total
I-MEM	Reg Read	ALU	D-MEM	Reg W	
200 ps	100 ps	200 ps	200 ps	100 ps	800 ps

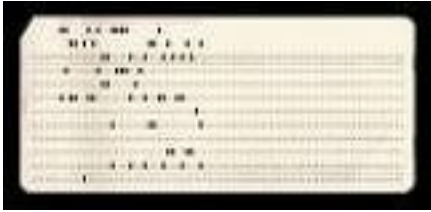
Review: Pipelining RISC-V RV32I Datapath



Components of a Computer



Early Read-Only Memory Technologies

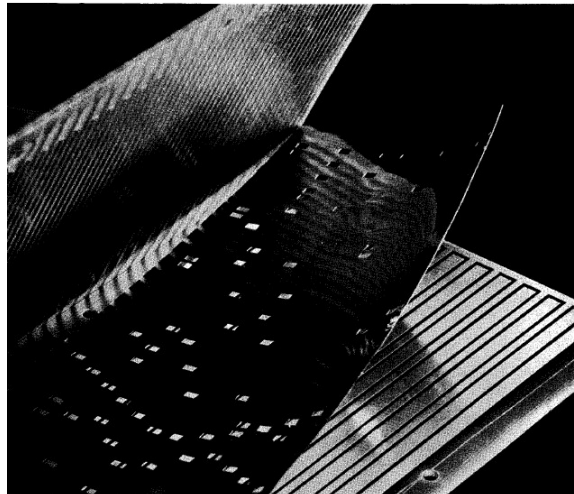
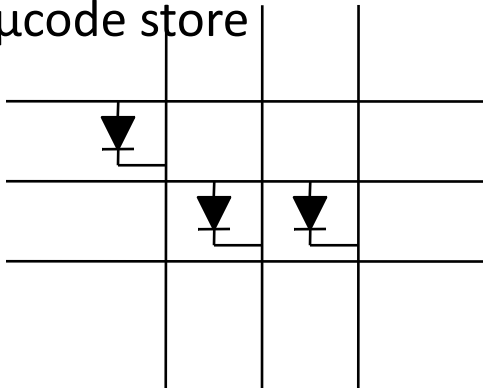


Punched cards, From early 1700s through Jacquard Loom, Babbage, and then IBM

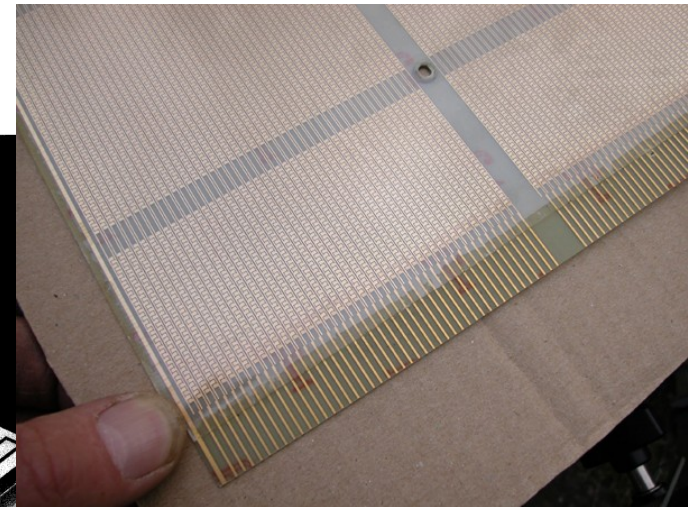


Punched paper tape, instruction stream in Harvard Mk 1

Diode Matrix, EDSAC-2 μ code store



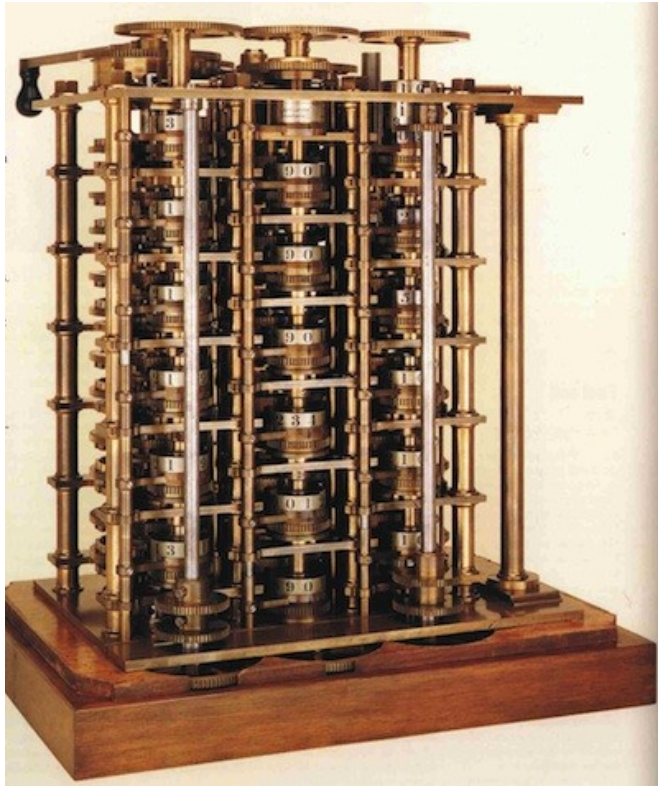
IBM Card Capacitor ROS



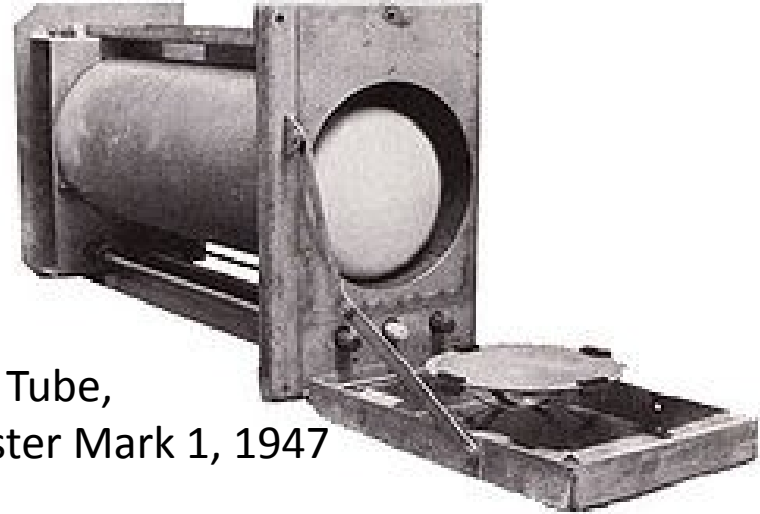
IBM Balanced Capacitor ROS

Early Read/Write Main Memory Technologies

Babbage, 1800s: Digits stored on mechanical wheels



Also, regenerative capacitor memory on Atanasoff-Berry computer, and rotating magnetic drum memory on IBM 650

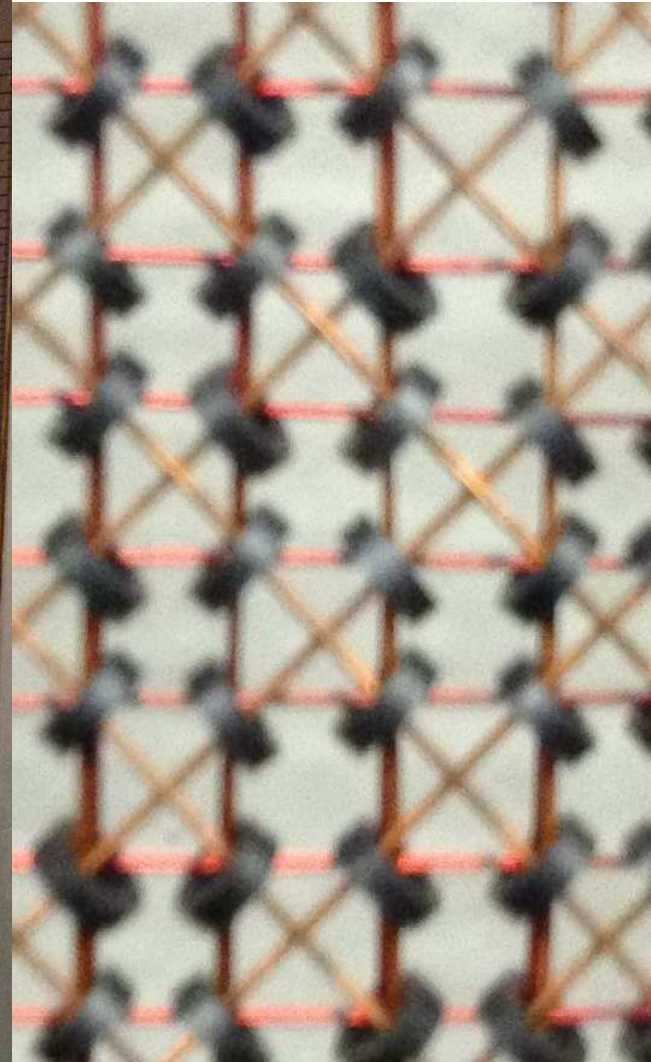


Williams Tube,
Manchester Mark 1, 1947

Mercury Delay Line, Univac 1, 1951



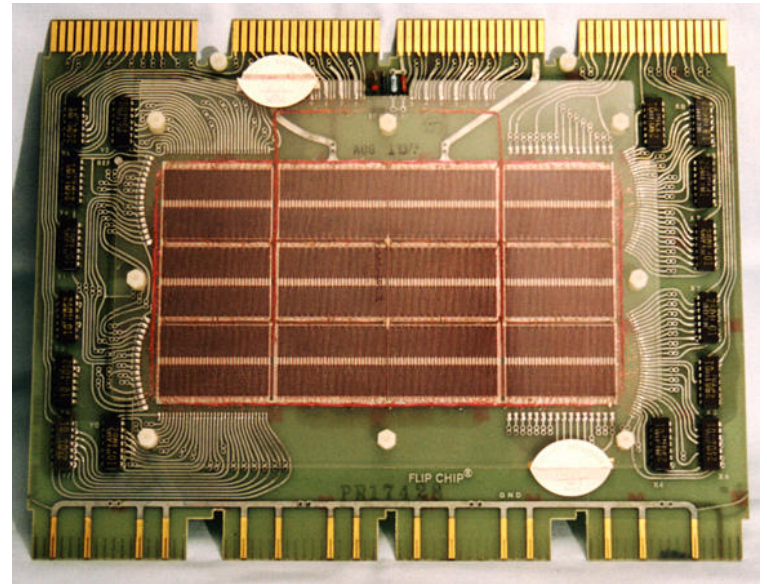
MIT Whirlwind Core Memory



Core Memory

- Core memory was first large scale reliable main memory
 - invented by Forrester in late 40s/early 50s at MIT for Whirlwind project
- Bits stored as magnetization polarity on small ferrite cores threaded onto two-dimensional grid of wires
- Coincident current pulses on X and Y wires would write cell and also sense original state (destructive reads)
- Robust, non-volatile storage
- Used on space shuttle computers
- Cores threaded onto wires by hand (25 billion a year at peak production)
- Core access time $\sim 1\mu\text{s}$

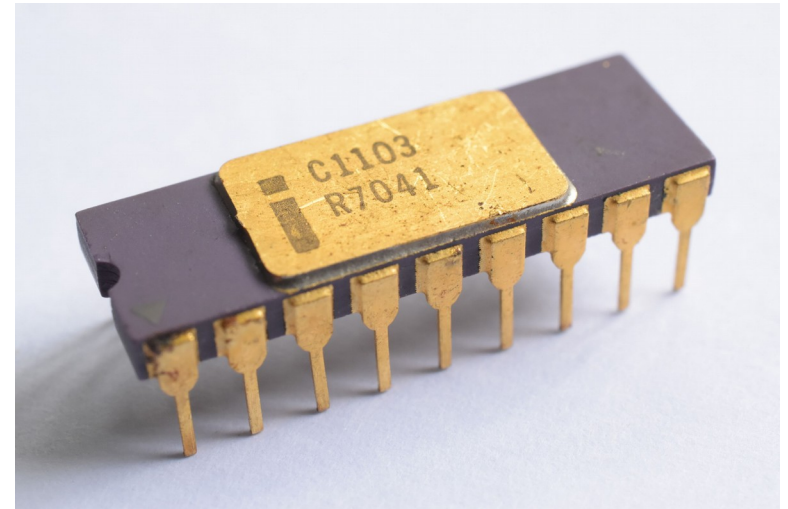
DEC PDP-8/E Board,
4K words x 12 bits, (1968)



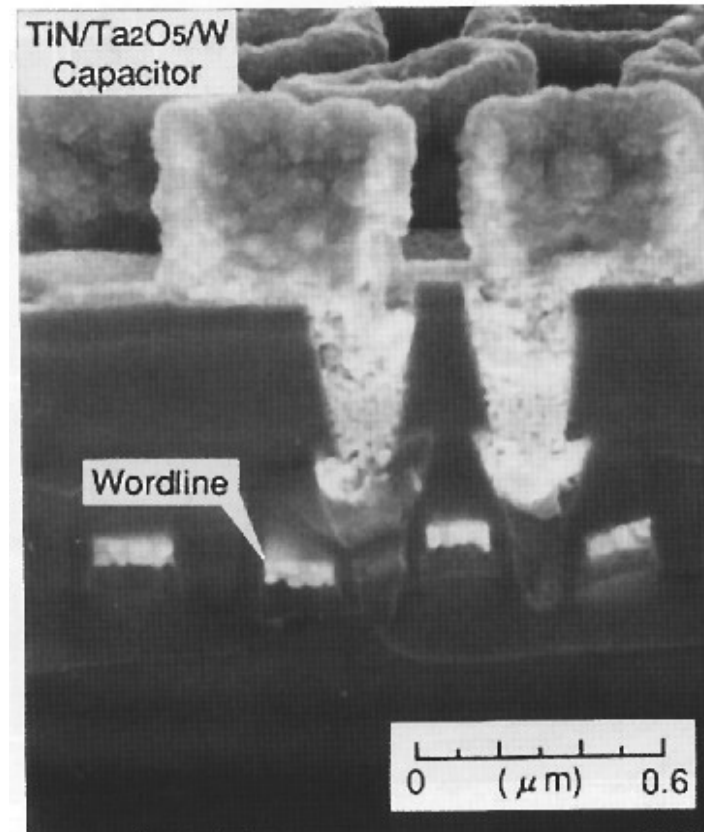
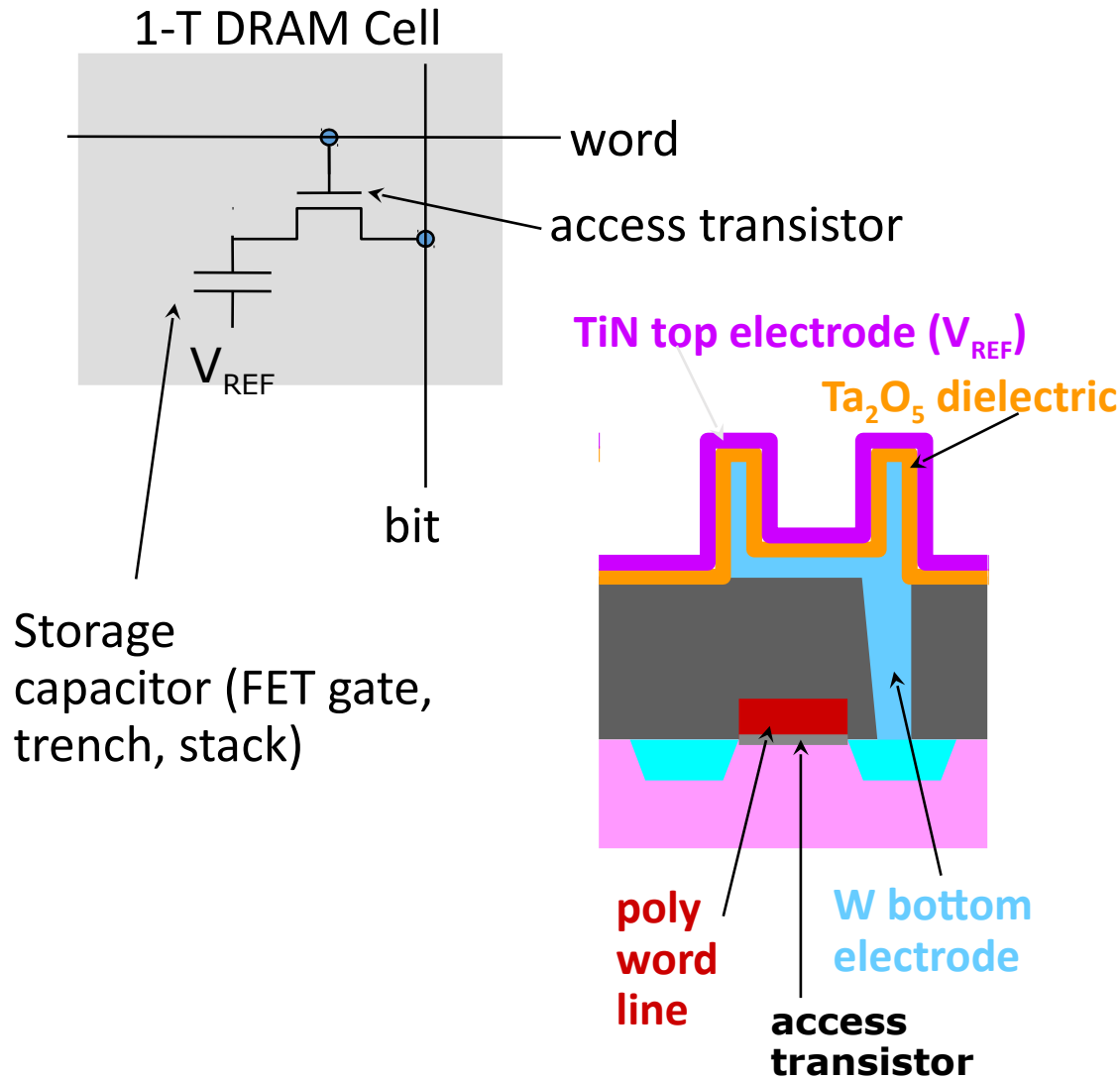
Core Memory

- 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

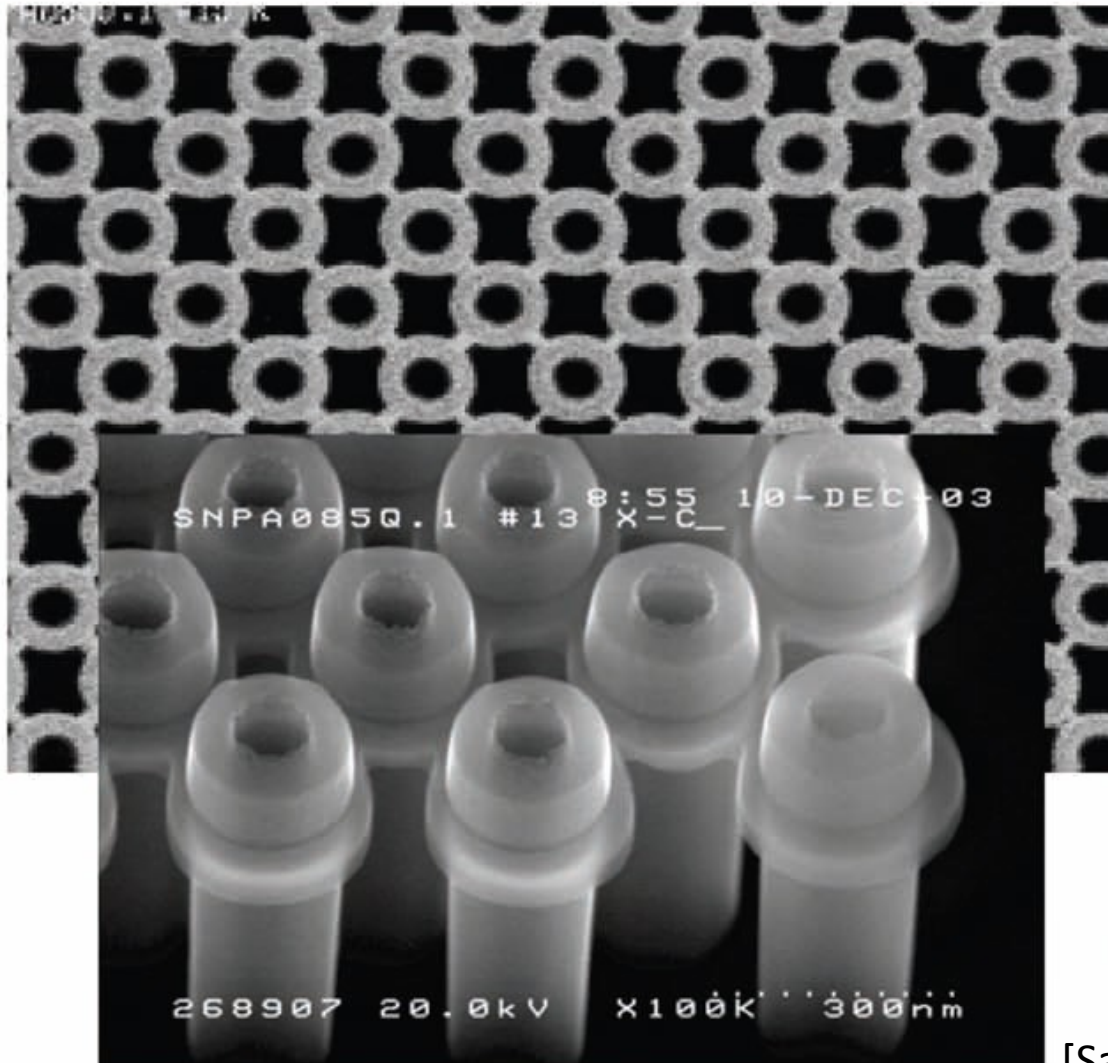
Semiconductor memory quickly replaced core in '70s



One-Transistor Dynamic RAM [Dennard, IBM]

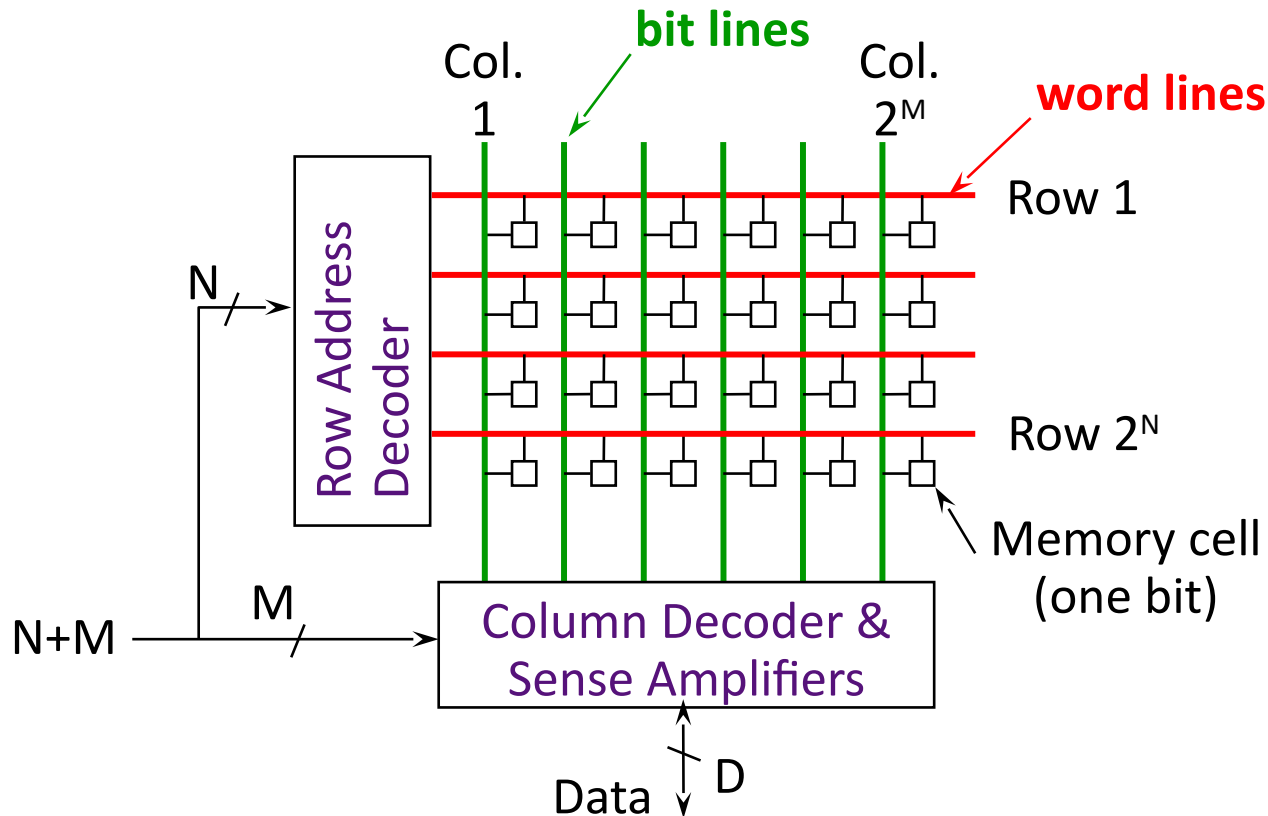


Modern DRAM Structure



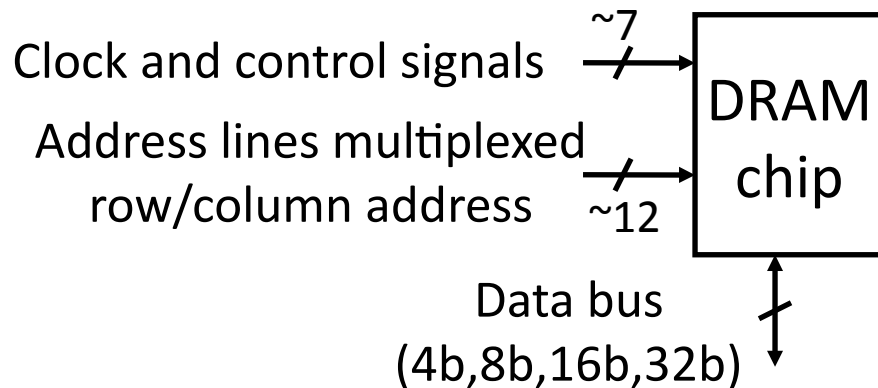
[Samsung, sub-70nm DRAM, 2004]

DRAM Architecture

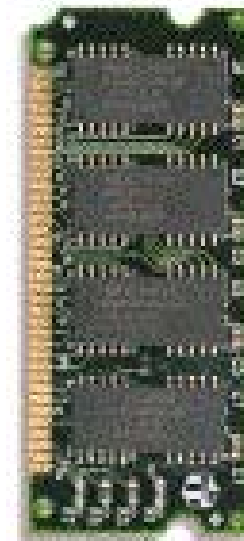


- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4-8 logical banks on each chip
 - each logical bank physically implemented as many smaller arrays

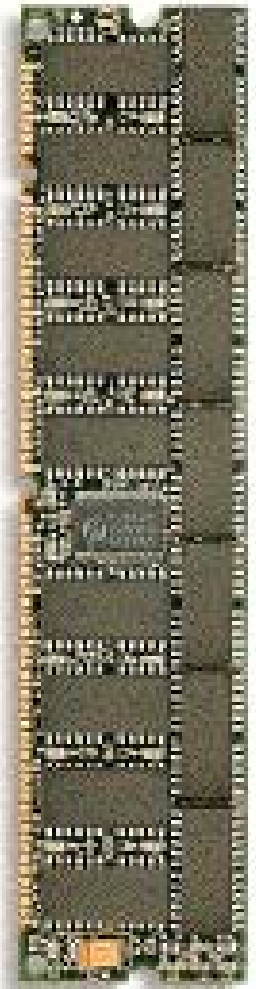
DRAM Architecture



- DIMM (Dual Inline Memory Module) contains multiple chips with clock/control/address signals connected in parallel (sometimes need buffers to drive signals to all chips)
- Data pins work together to return wide word (e.g., 64-bit data bus using 16x4-bit parts)



72-pin SO DIMM

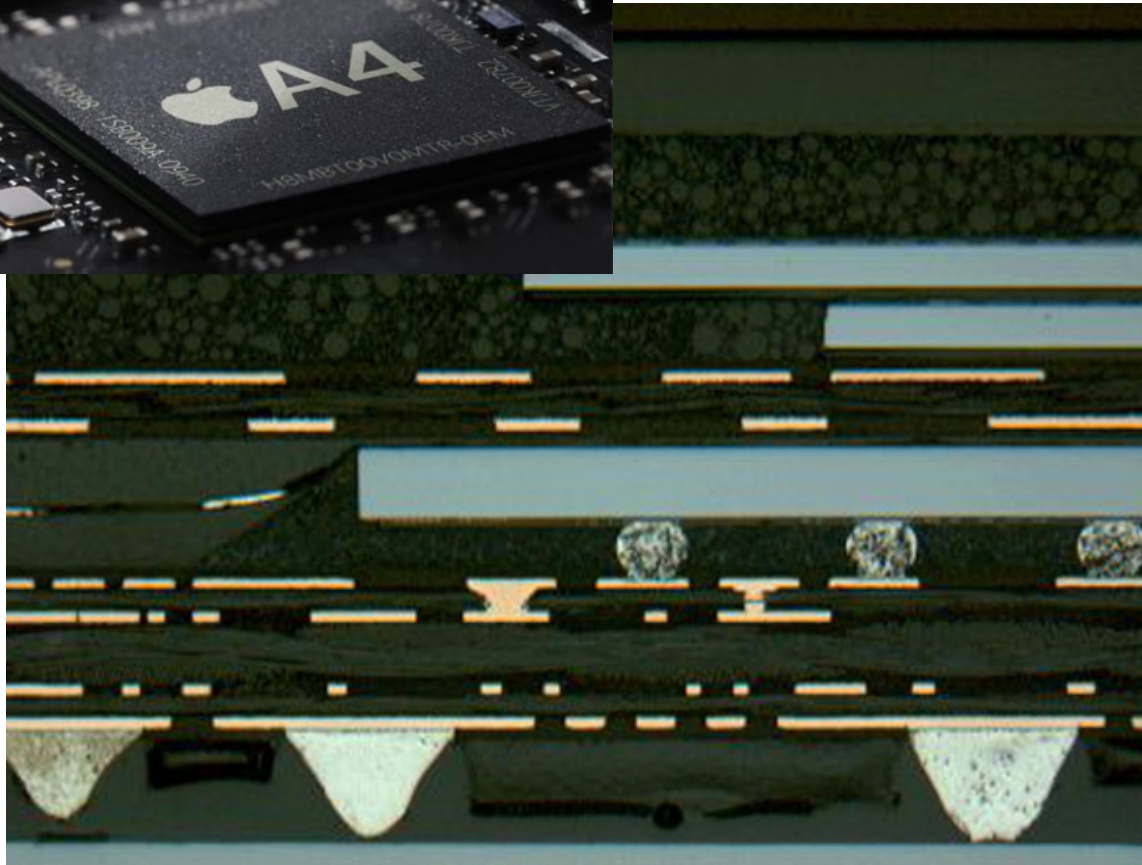


168-pin DIMM

DRAM Packaging, Mobile Devices



[Apple A4 package on circuit board]

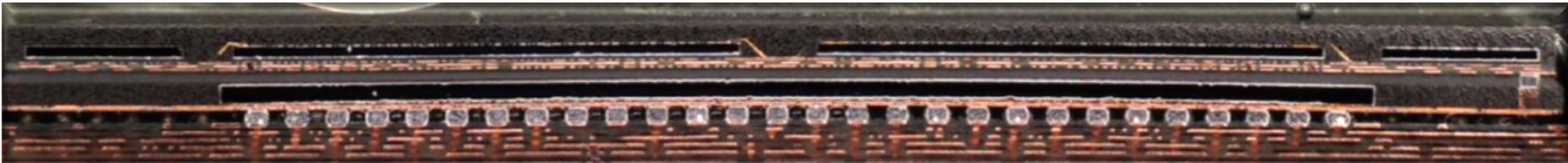
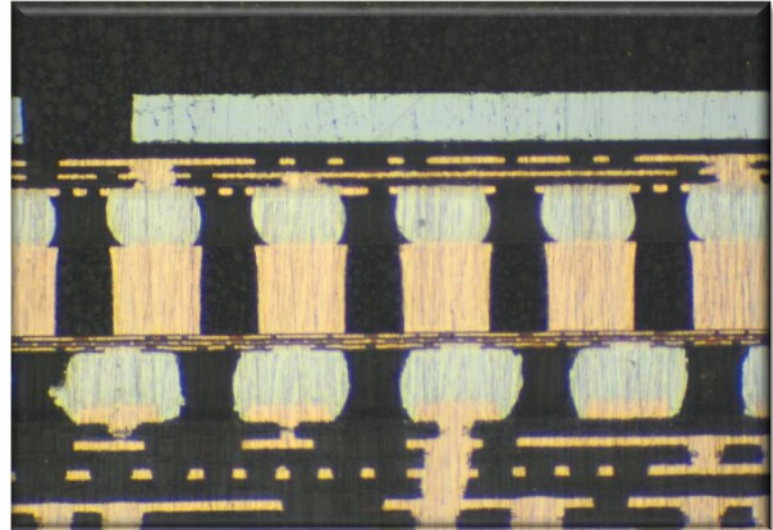
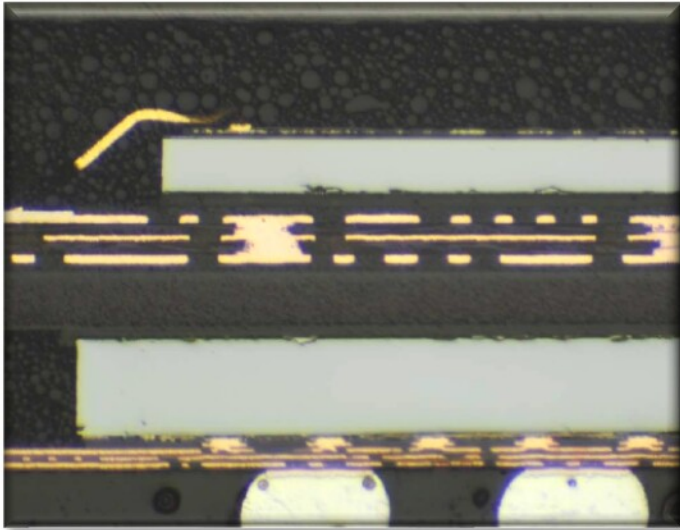


← Two stacked
DRAM die
← Processor
plus logic die

[Apple A4 package cross-section, iFixit 2010]

DRAM Packaging Apple A10

Apple A10 Fan Out Package – TSMC InFO



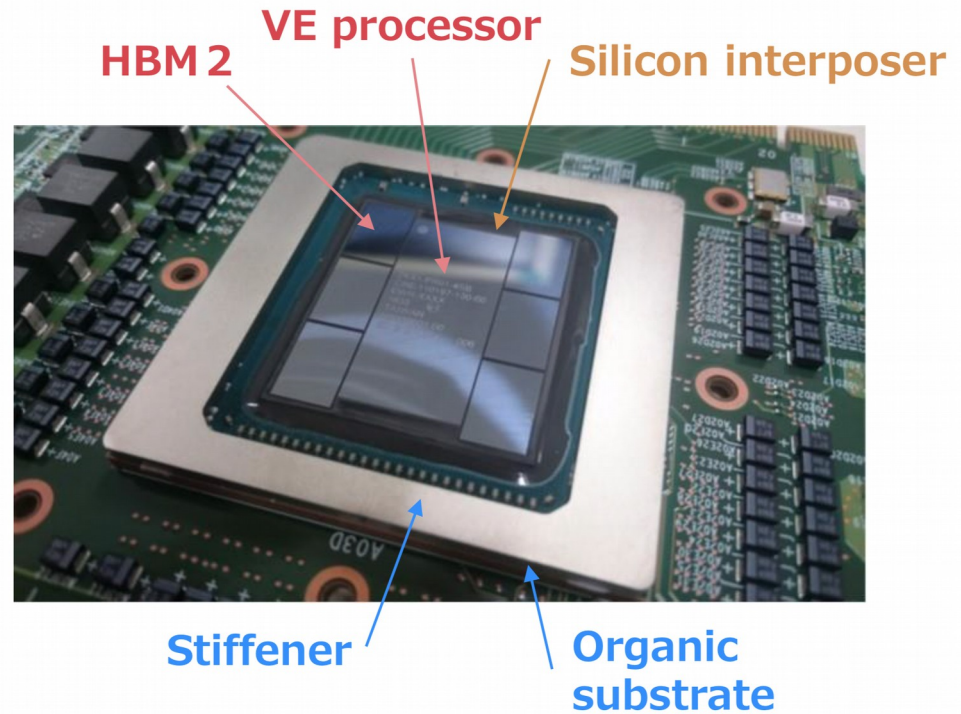
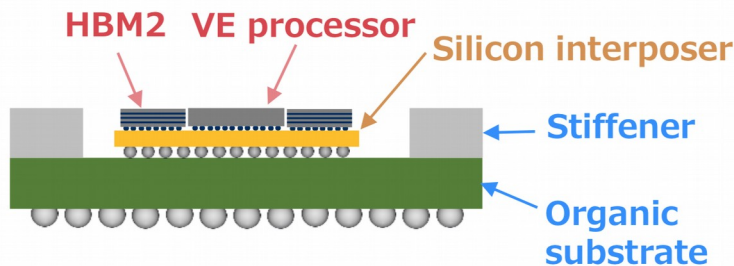
High-Bandwidth Memory in SX-Aurora

Vector Engine Processor Module

SX-Aurora TSUBASA

2.5D implementation

- A VE processor and six 8Hi or 4Hi HBM2 modules on a silicon interposer
- Lidless package to minimize thermal resistance
- Package size: 60mm x 60mm
- Interposer size: 32.5mm x 38mm
- VE processor size: 15mm x 33mm



World's first implementation of a processor with 6 HBM2s

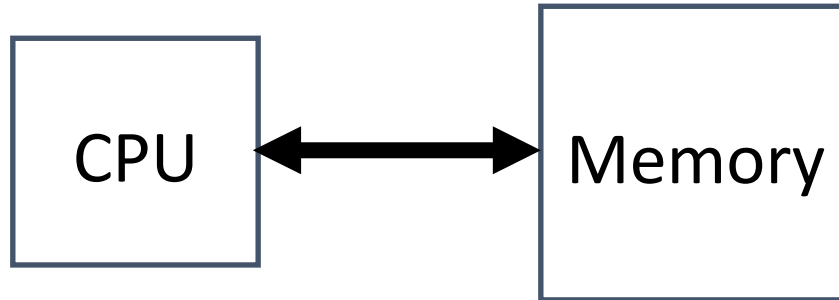
DRAM Operation

- Three steps in read/write access to a given bank
- Row access (RAS)
 - decode row address, enable addressed row (often multiple Kb in row)
 - bitlines share charge with storage cell
 - small change in voltage detected by SAs which latch whole row of bits
 - sense amplifiers drive bitlines full rail to recharge storage cells
- Column access (CAS)
 - decode column address to select small number of sense amplifier latches (4, 8, 16, or 32 bits depending on DRAM package)
 - on read, send latched bits out to chip pins
 - on write, change SA latches which then charge storage cells to required value
 - can perform multiple column accesses on same row without another row access (burst mode)
- Precharge
 - charges bit lines to known value, required before next row access
- Each step has a latency of around 15-20ns in modern DRAMs
- Various DRAM standards (DDR, RDRAM) have different ways of encoding the signals for transmission to the DRAM, but all share same core architecture

200MHz Clock



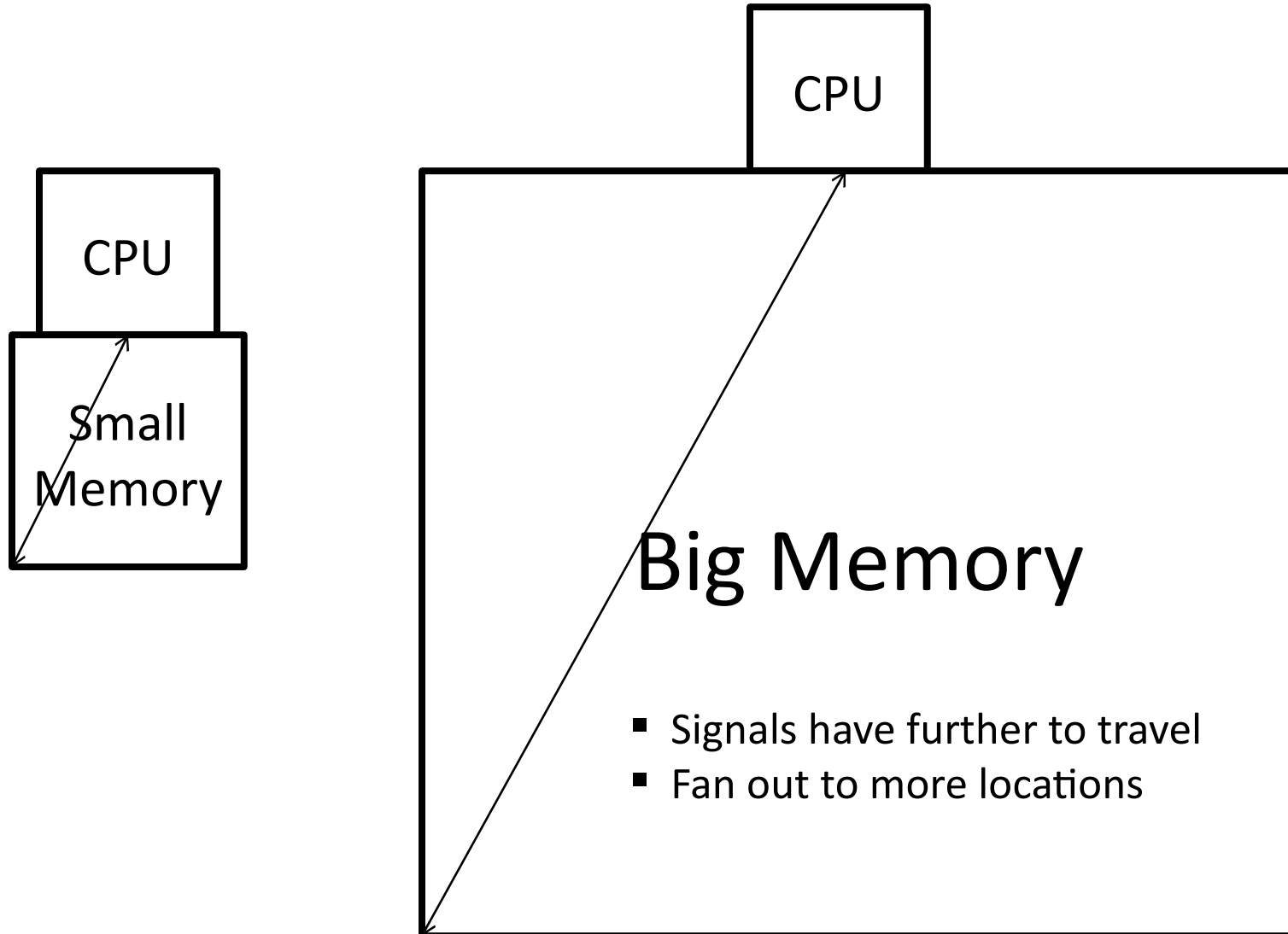
CPU-Memory Bottleneck



Performance of high-speed computers is usually limited by memory bandwidth & latency

- Latency (time for a single access)
 - Memory access time \gg Processor cycle time
- Bandwidth (number of accesses per unit time)
 - if fraction m of instructions access memory
 - $1+m$ memory references / instruction
 - CPI = 1 requires $1+m$ memory refs / cycle (assuming RISC-V ISA)

Physical Size Affects Latency

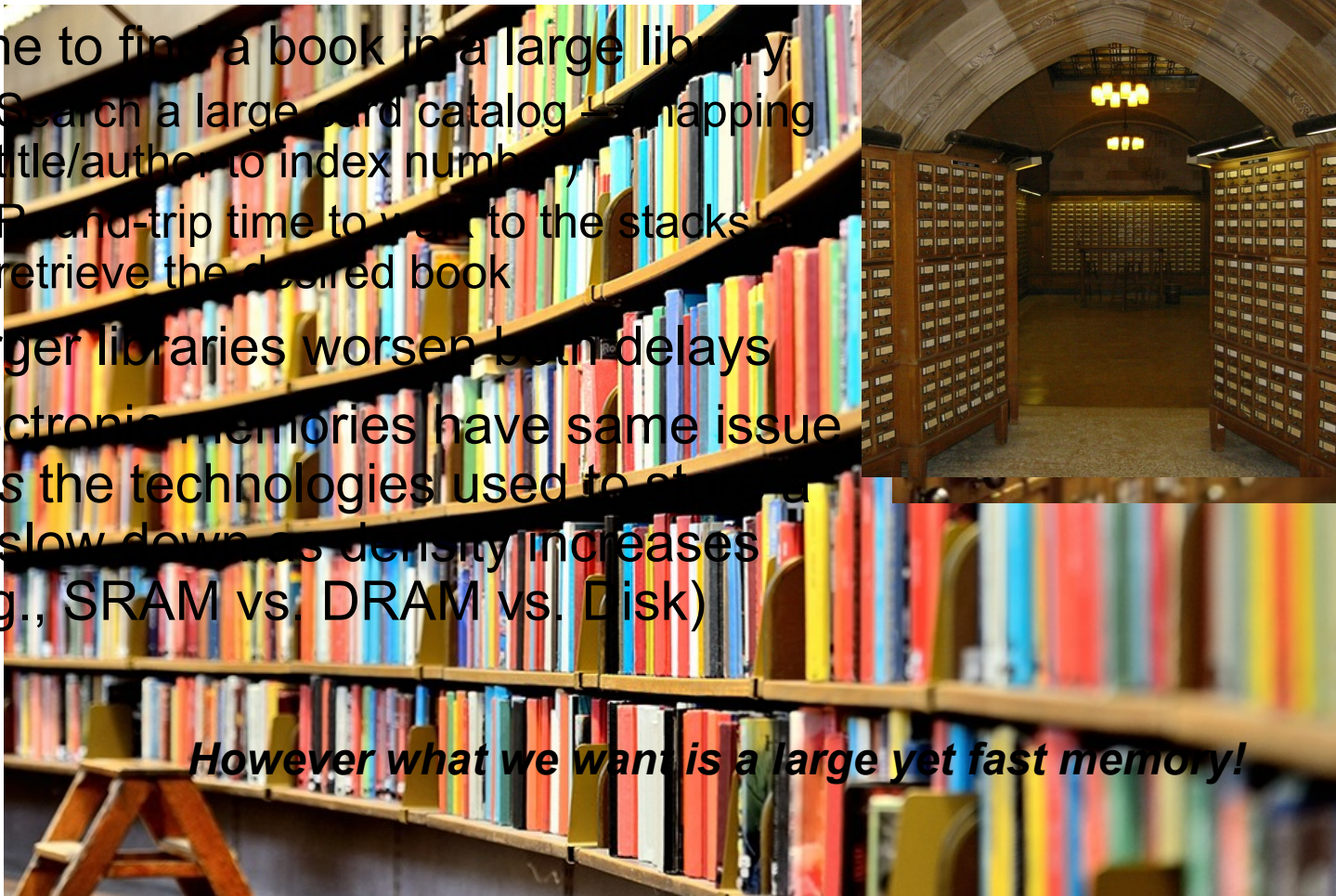


Why are Large Memories Slow?

Library Analogy

- Time to find a book in a large library:
 - Search a large card catalog + mapping title/author to index number
 - Round-trip time to walk to the stacks + retrieve the desired book
- Larger libraries worsen both delays
- Electronic memories have same issue *plus* the technologies used to store a bit slow down as density increases (e.g., SRAM vs. DRAM vs. Disk)

However what we want is a large yet fast memory!

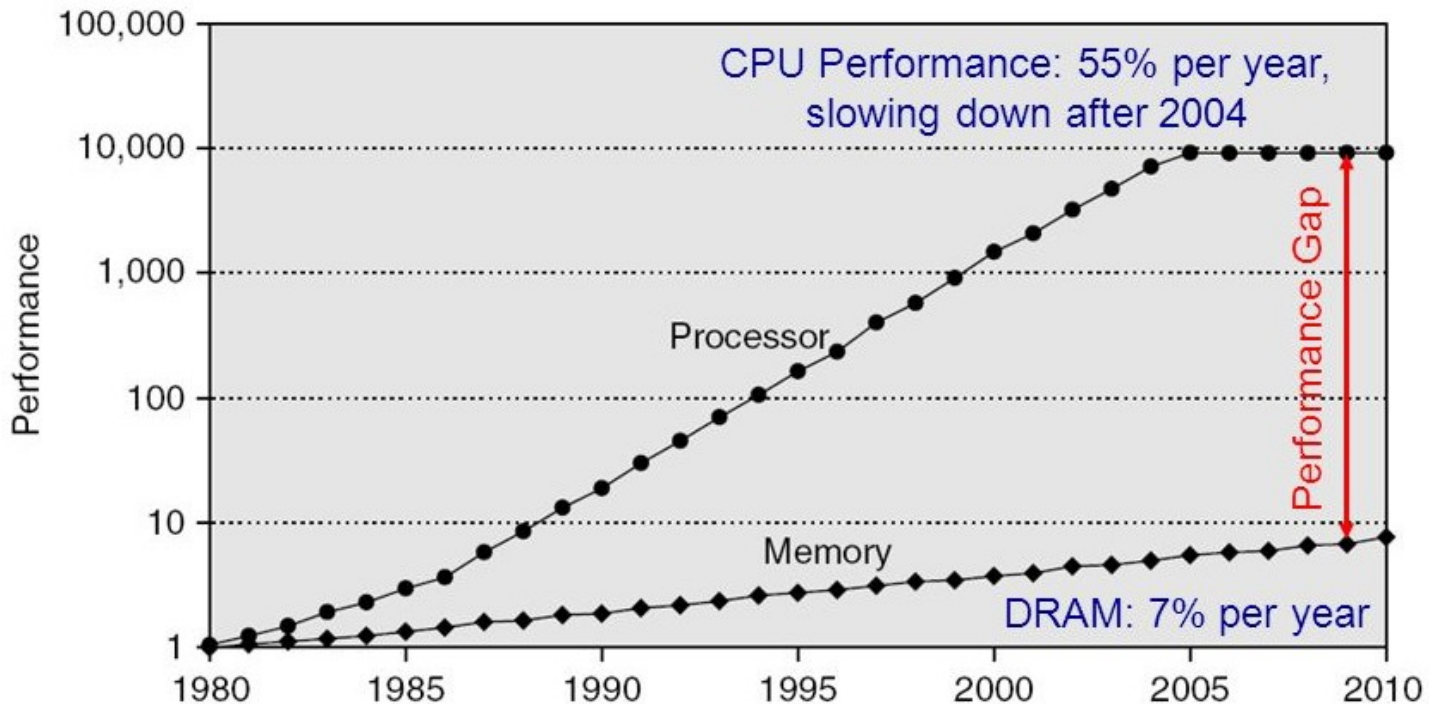


What To Do: Library Analogy

- Write a report using library books
 - E.g., works of J.D. Salinger
- Go to library, look up relevant books, fetch from stacks, and place on desk in library
- If need more, check them out and keep on desk
 - But don't return earlier books since might need them
- You hope this collection of ~10 books on desk enough to write report, despite 10 being only 0.00001% of books in ZJU libraries



Processor-DRAM Gap (Latency)

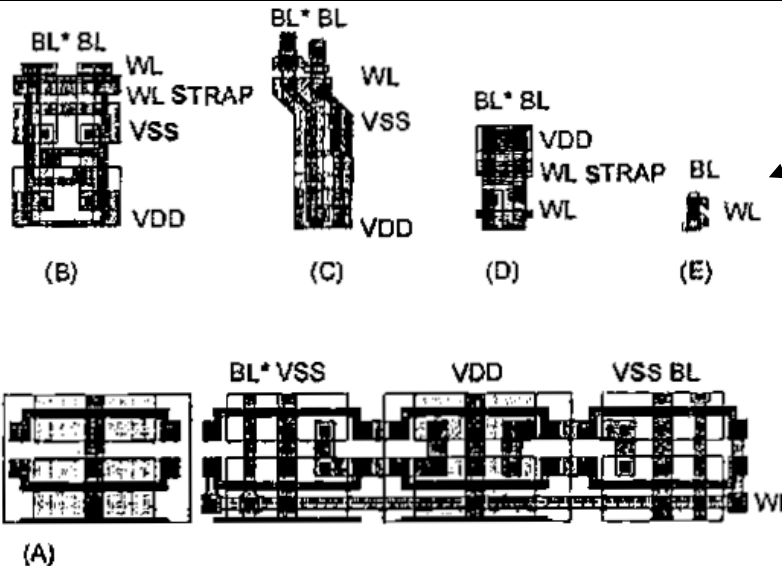


1980 microprocessor executes **~one instruction** in same time as DRAM access
2017 microprocessor executes **~1000 instructions** in same time as DRAM access

Slow DRAM access has disastrous impact on CPU performance!

Relative Memory Cell Sizes

On-Chip
SRAM in
logic chip



DRAM on
memory chip

1 Memory cell in 0.5μm processes

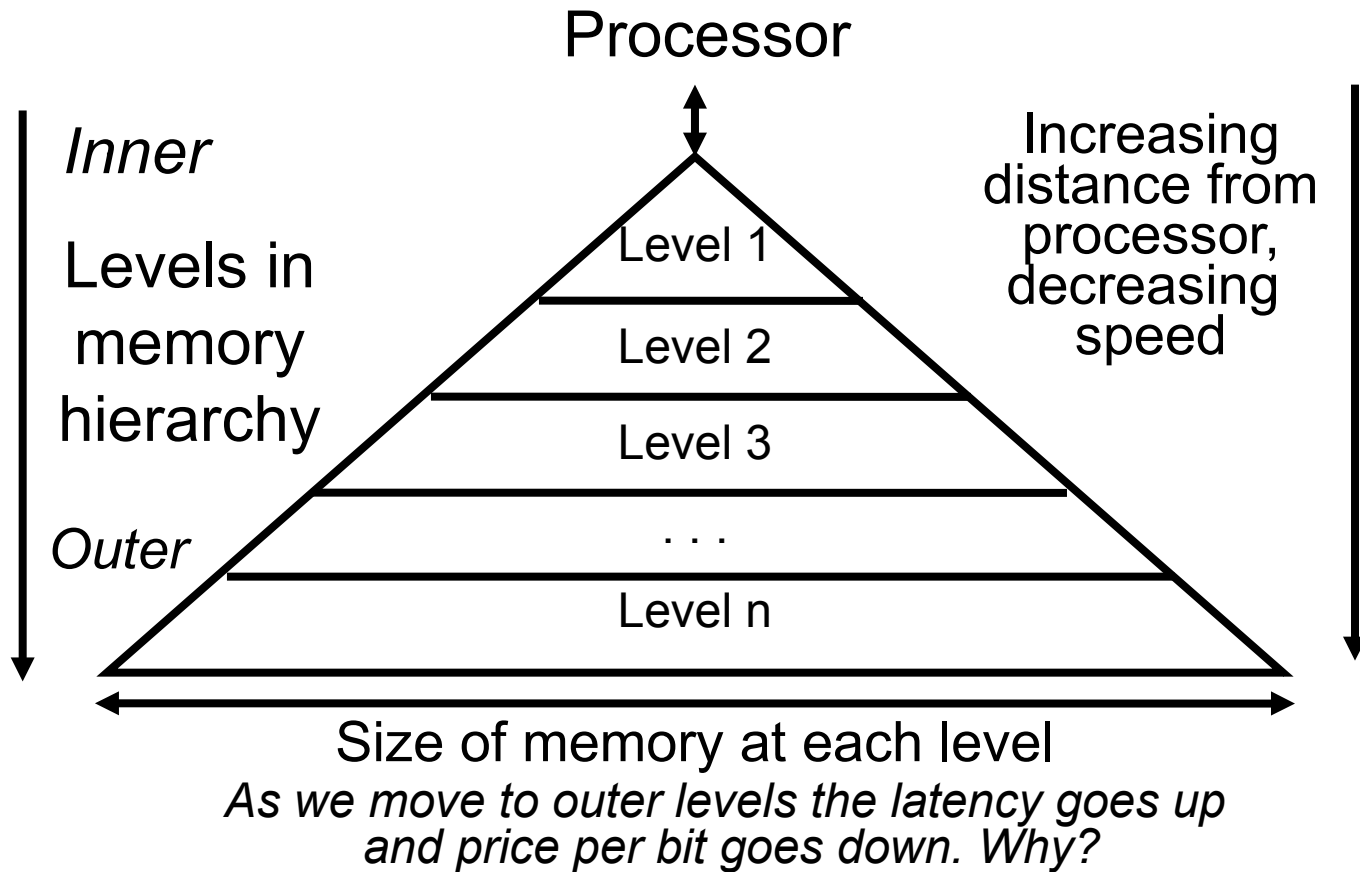
- a) Gate Array SRAM
- b) Embedded SRAM
- c) Standard SRAM (6T cell with local interconnect)
- d) ASIC DRAM
- e) Standard DRAM (stacked cell)

[Foss, "Implementing
Application-Specific
Memory", ISSCC 1996]

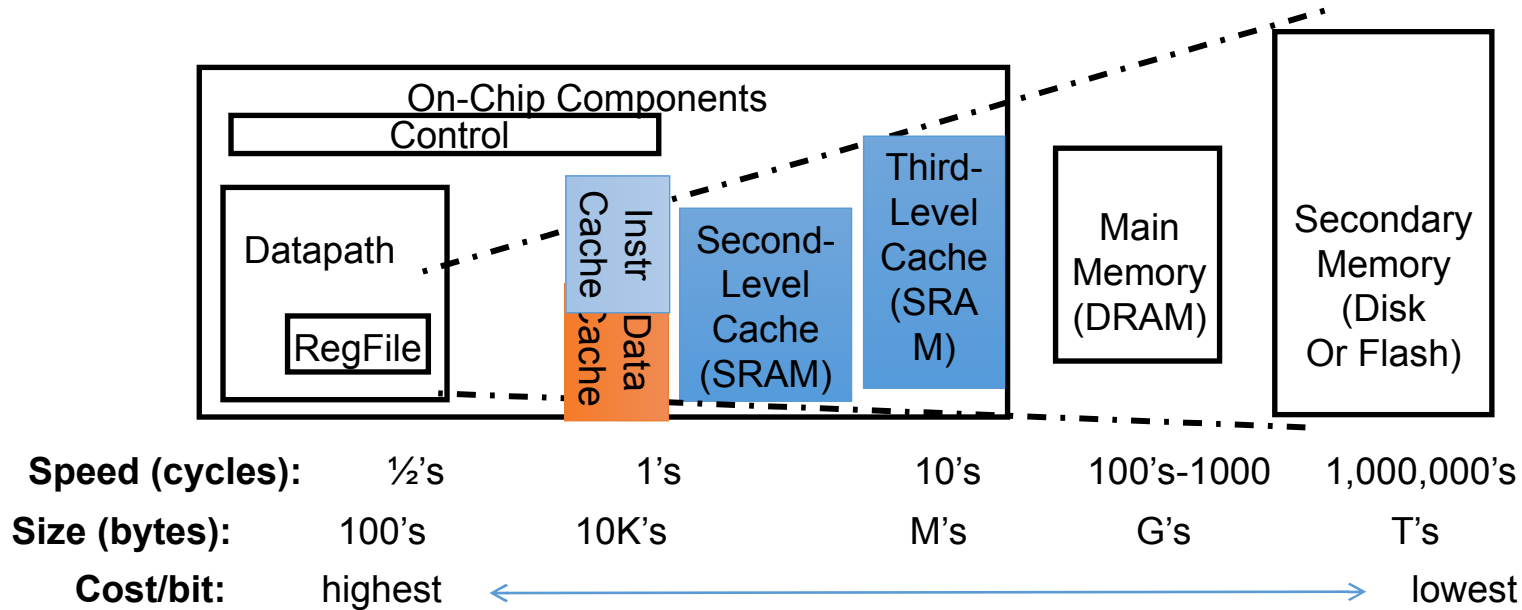
Memory	Process	Cell size (μm ²)	Cell efficiency	Bits in 100mm ² (10 ³)	Gate size (μm ²)	Gate utilization	Gates in 100mm ² (10 ³)
Gate array SRAM	3-metal ASIC	370	80%	216	185	70%	378
Embedded SRAM	3-metal ASIC	67	70%	1045	185	70%	378
Standard SRAM	2-metal 6T local int.	43	65%	1512	245	40%	163
Embedded ASIC-DRAM	3-metal ASIC	23	60%	2609	185	70%	378
Standard DRAM	2-metal stacked cell	3.2	50%	15625	411	40%	97

Table 1: Memory and logic density for a variety of 0.5μm implementations.

Big Idea: Memory Hierarchy

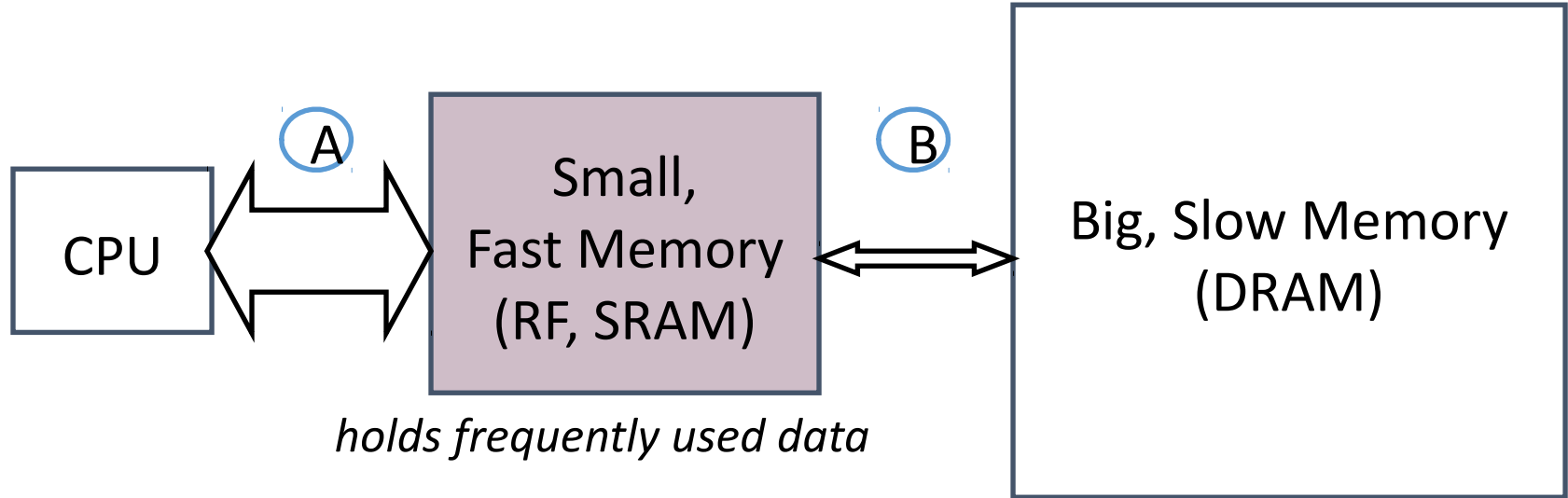


Typical Memory Hierarchy



- **Principle of locality + memory hierarchy** presents programmer with \approx as much memory as is available in the *cheapest* technology at the \approx speed offered by the *fastest* technology

Memory Hierarchy



- *capacity*: Register \ll SRAM \ll DRAM
- *latency*: Register \ll SRAM \ll DRAM
- *bandwidth*: on-chip \gg 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)

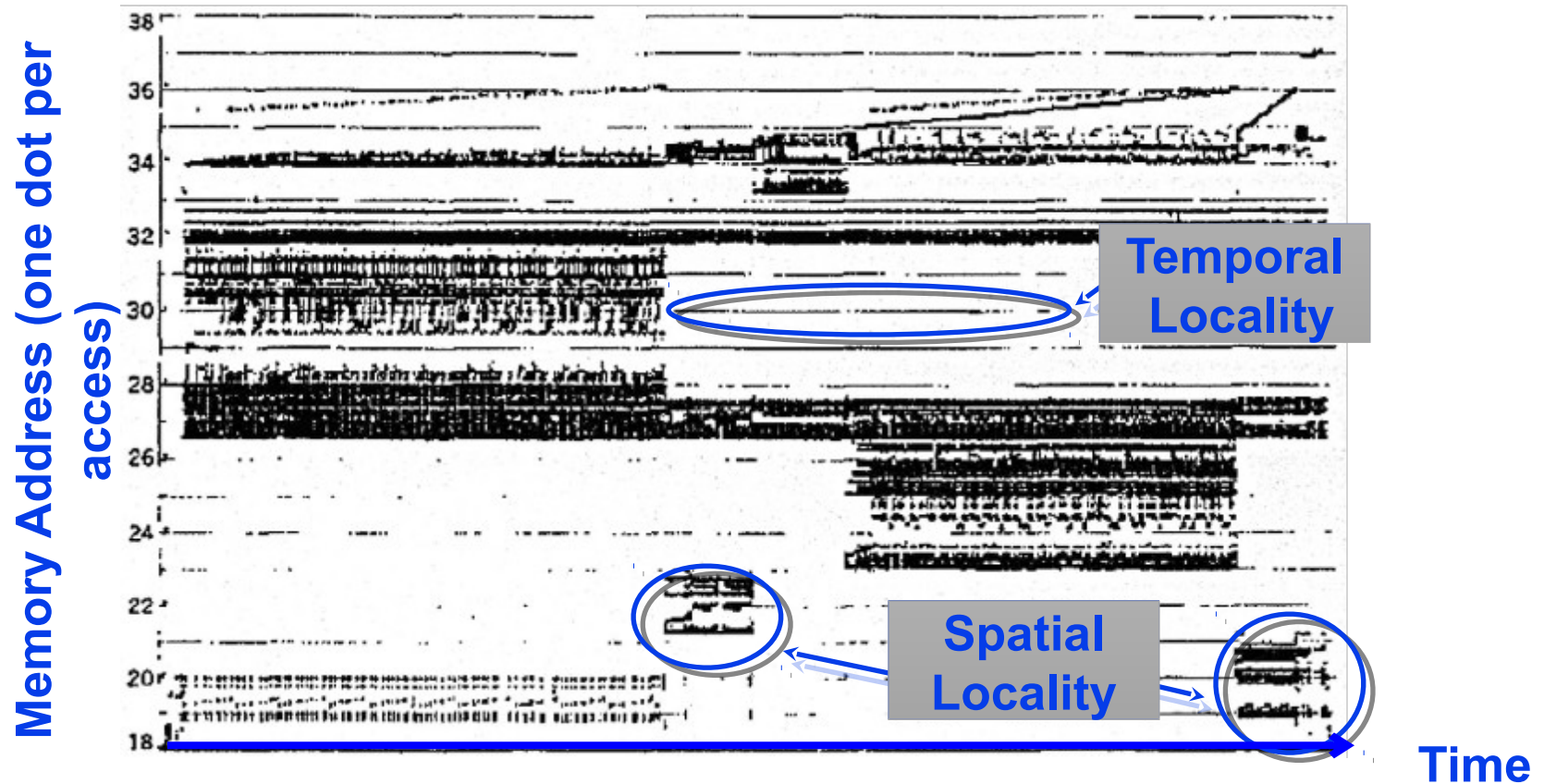
Management of Memory Hierarchy

- Small/fast storage, e.g., registers
 - Address usually specified in instruction
 - Generally implemented directly as a register file
 - *but hardware might do things behind software's back, e.g., stack management, register renaming*
- Larger/slower storage, e.g., main memory
 - Address usually computed from values in register
 - Generally implemented as a hardware-managed cache hierarchy (hardware decides what is kept in fast memory)
 - *but software may provide "hints", e.g., don't cache or prefetch*

Big Idea: Locality

- *Temporal Locality* (locality in time)
 - Go back to same book on desk multiple times
 - If a memory location is referenced, then it will tend to be referenced again soon
- *Spatial Locality* (locality in space)
 - When go to book shelf, pick up multiple books on J.D. Salinger since library stores related books together
 - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

Memory Reference Patterns

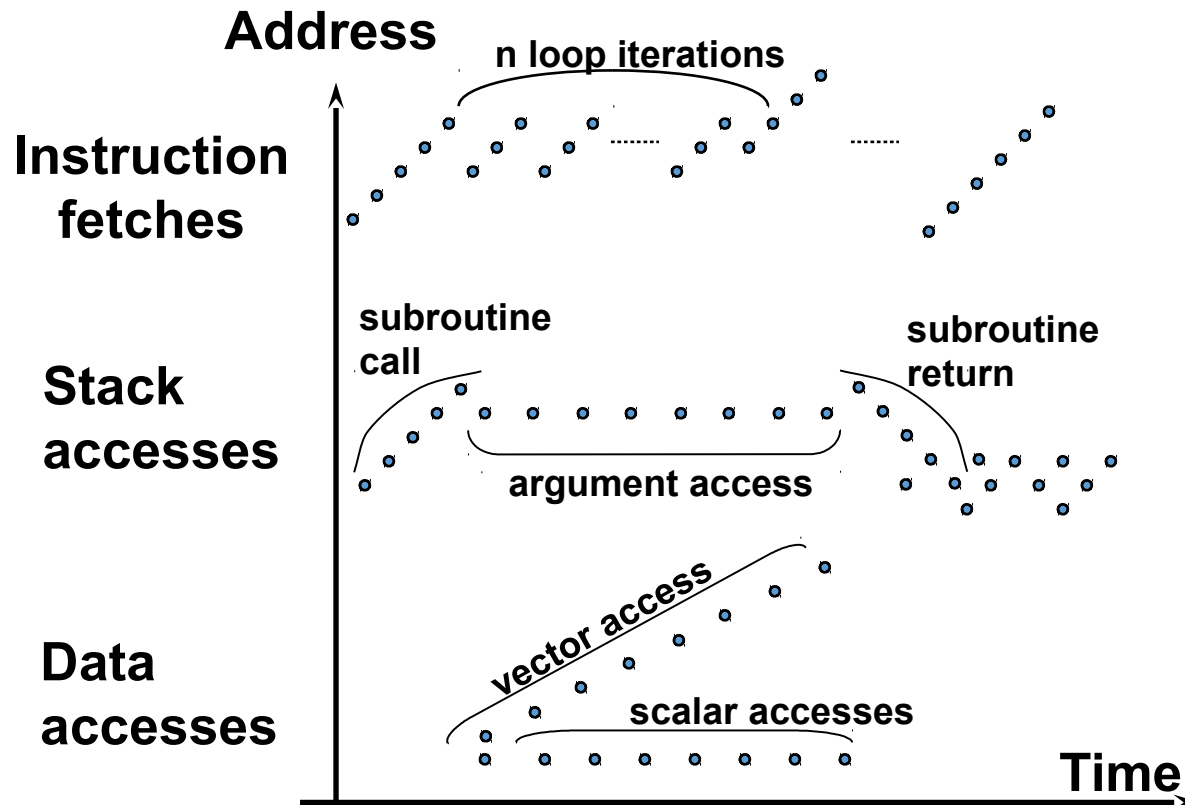


Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory.
IBM Systems Journal 10(3): 168-192 (1971)

Principle of Locality

- *Principle of Locality*: Programs access small portion of address space at any instant of time (spatial locality) and repeatedly access that portion (temporal locality)
- What program structures lead to *temporal* and *spatial locality* in instruction accesses?
- In data accesses?
- What structures defeat temporal and spatial locality in instruction and data accesses?

Memory Reference Patterns




Caches exploit both types of predictability:

- Exploit temporal locality by remembering the contents of recently accessed locations.
 - Keep recently referenced items at higher levels, future references satisfied quickly
- Exploit spatial locality by fetching blocks of data around recently accessed locations.
 - Bring neighbors of recently referenced to higher levels, future references satisfied quickly

Cache

cache

/kæʃ/ 

noun

noun: **cache**; plural noun: **caches**

1. a collection of items of the same type stored in a hidden or inaccessible place.

"an arms cache"

synonyms: **hoard**, **store**, **stockpile**, **stock**, **supply**, **reserve**; [More](#)

- a hidden or inaccessible storage place for valuables, provisions, or ammunition.

synonyms: **hoard**, **store**, **stockpile**, **stock**, **supply**, **reserve**; [More](#)

- **COMPUTING**

an auxiliary memory from which high-speed retrieval is possible.

noun: **cache memory**; plural noun: **cache memories**

verb

verb: **cache**; 3rd person present: **caches**; past tense: **cached**; past participle: **cached**; gerund or present participle: **caching**

1. store away in hiding or for future use.

- **COMPUTING**

store (data) in a cache memory.

- **COMPUTING**

provide (hardware) with a cache memory.

Origin

FRENCH

acher → cache

to hide late 18th century

late 18th century: from French, from *acher* 'to hide.'

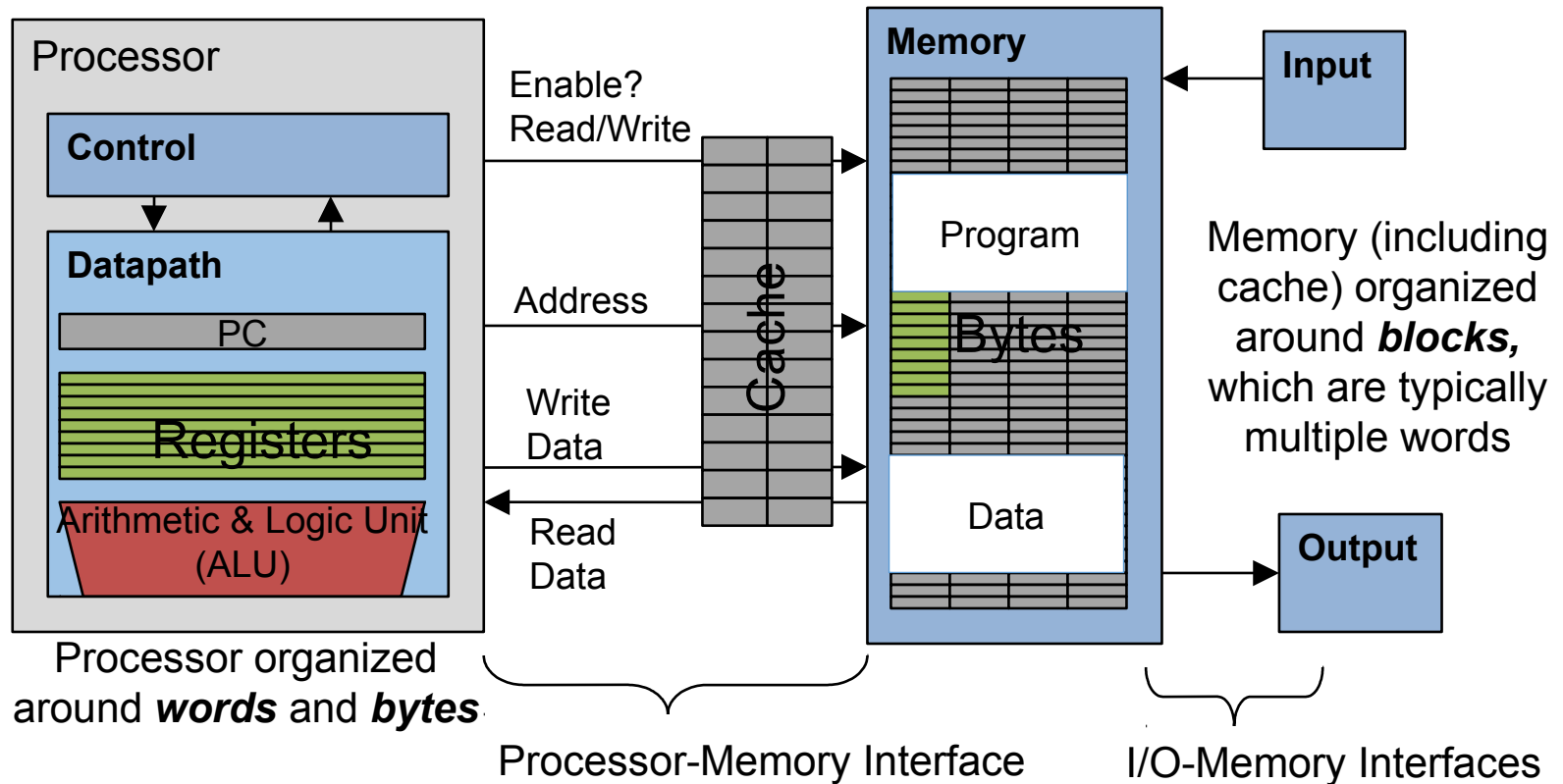
Cache Philosophy

- Programmer-invisible hardware mechanism gives illusion of speed of fastest memory with size of largest memory
 - Works even if you have no idea what a cache is
 - Performance-oriented programmers sometimes “reverse engineer” cache organization to design data structures and access patterns optimized for a specific cache design

Memory Access without Cache

- Load word instruction: `lw t0, 0(t1)`
 - `t1` contains 1022_{ten} , `Memory[1022] = 99`
1. Processor issues address 1022_{ten} to Memory
 2. Memory reads word at address 1022_{ten} (99)
 3. Memory sends 99 to Processor
 4. Processor loads 99 into register `t0`

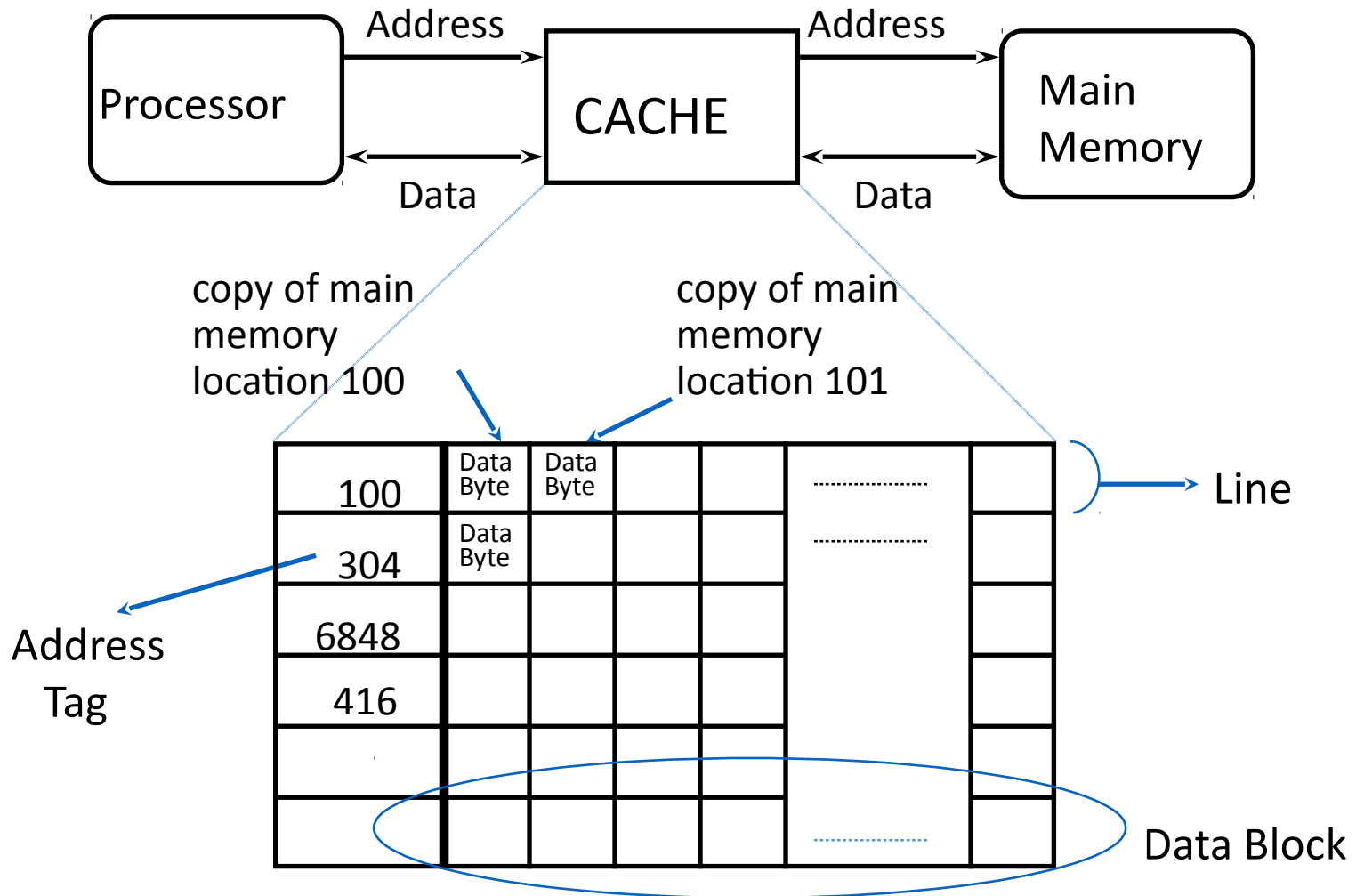
Adding Cache to Computer



Memory Access with Cache

- Load word instruction: `lw t0, 0(t1)`
- `t1` contains 1022_{ten} , `Memory[1022] = 99`
- With cache: Processor issues address 1022_{ten} to Cache
 1. Cache checks to see if has copy of data at address 1022_{ten}
 - 1) If finds a match (**Hit**): cache reads 99, sends to processor
 - 2) No match (**Miss**): cache sends address 1022 to Memory
 - I. Memory reads 99 at address 1022_{ten}
 - II. Memory sends 99 to Cache
 - III. Cache replaces word with new 99
 - IV. Cache sends 99 to processor
 2. Processor loads 99 into register `t0`

Inside a Cache



Cache Algorithm (Read)

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?

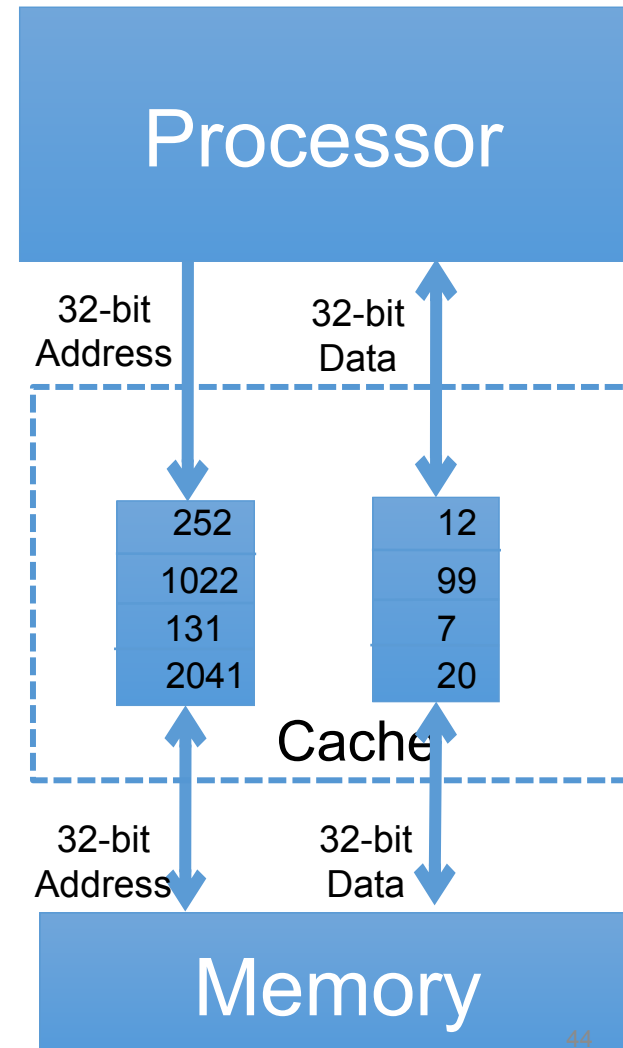
Cache “Tags”

- Need way to tell if have copy of location in memory so that can decide on hit or miss
- On cache miss, put memory address of block in “tag address” of cache block
 - 1022 placed in tag next to data from memory (99)

Tag	Data	
252	12	
1022	99	← From earlier loads or stores
131	7	
2041	20	

Anatomy of a 16 Byte Cache, with 4 Byte Blocks

- Operations:
 - Cache Hit
 - Cache Miss
 - Refill cache from memory
- Cache needs Address Tags to decide if Processor Address is a Cache Hit or Cache Miss
 - Compares all four tags



Cache Replacement

- Suppose processor now requests location 511, which contains 11?
- Doesn't match any cache block, so must "evict" a resident block to make room
 - Which block to evict?
- Replace "victim" with new memory block at address 511

Tag	Data
252	12
1022	99
131	7
2041	20

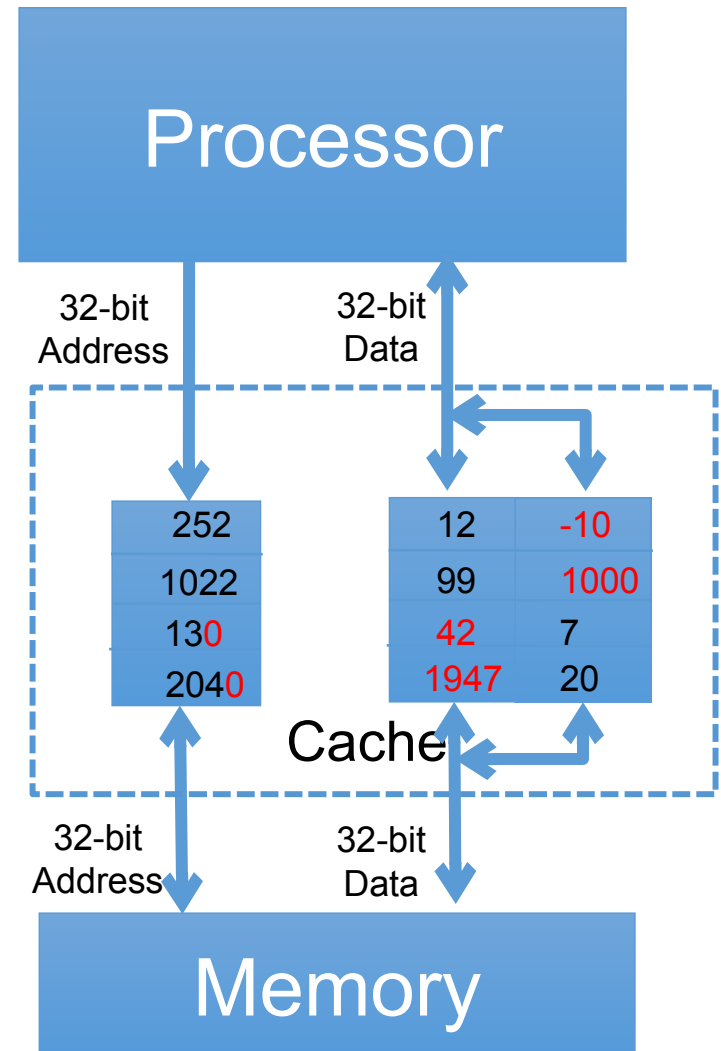
least_recently used(LRU)

Block Must be Aligned in Memory

- Word blocks are aligned, so binary address of all words in cache always ends in 00_{two}
- How to take advantage of this to save hardware and energy?
- Don't need to compare last 2 bits of 32-bit byte address (comparator can be narrower)
 - Don't need to store last 2 bits of 32-bit byte address in Cache Tag (Tag can be narrower)

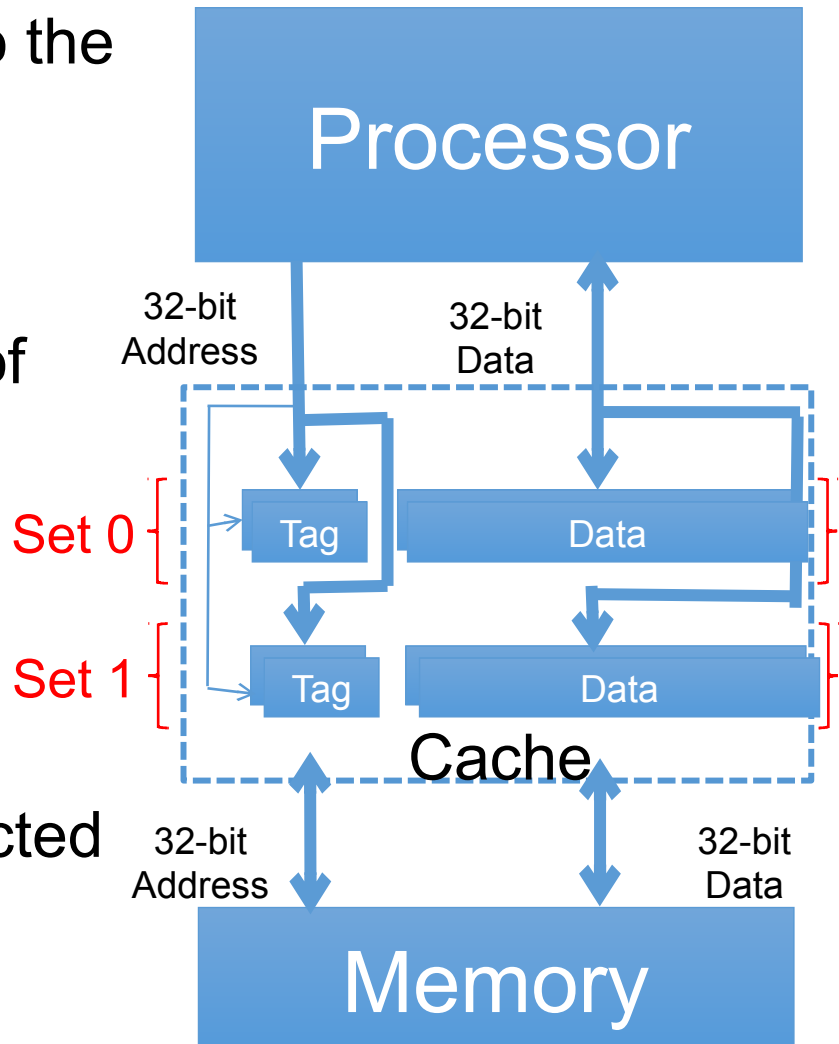
Anatomy of a 32B Cache, 8B Blocks

- Blocks must be aligned in pairs, otherwise could get same word twice in cache
 - Tags only have even-numbered words
 - Last 3 bits of address always 000_{two}
 - Tags, comparators can be narrower
- Can get hit for either word in block



Hardware Cost of Cache

- Need to compare every tag to the Processor address
- Comparators are expensive
- Optimization: use two “sets” of data with a total of only 2 comparators
- Use one Address bit to select which set
- Compare only tags from selected set
- Generalize to more sets



Hardware Cost of Cache

