KING'S College LONDON

Hardware Design

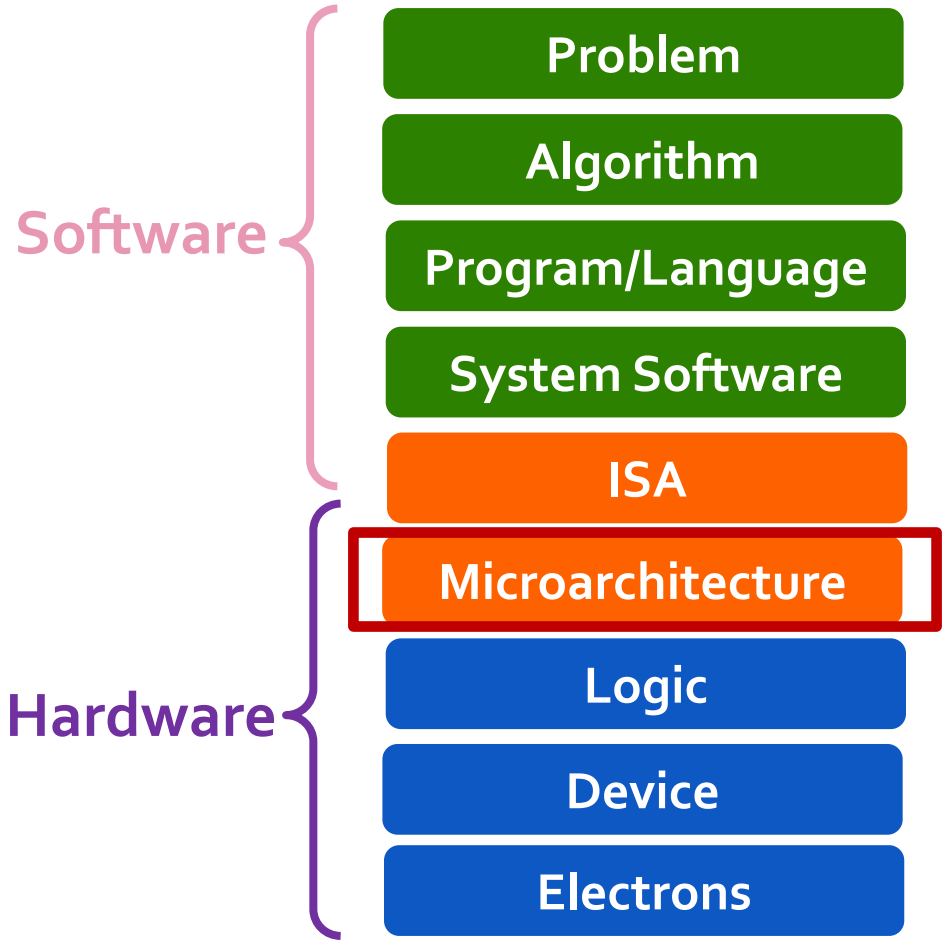
Lecture 7: Memory

Dr. Haiyu Mao

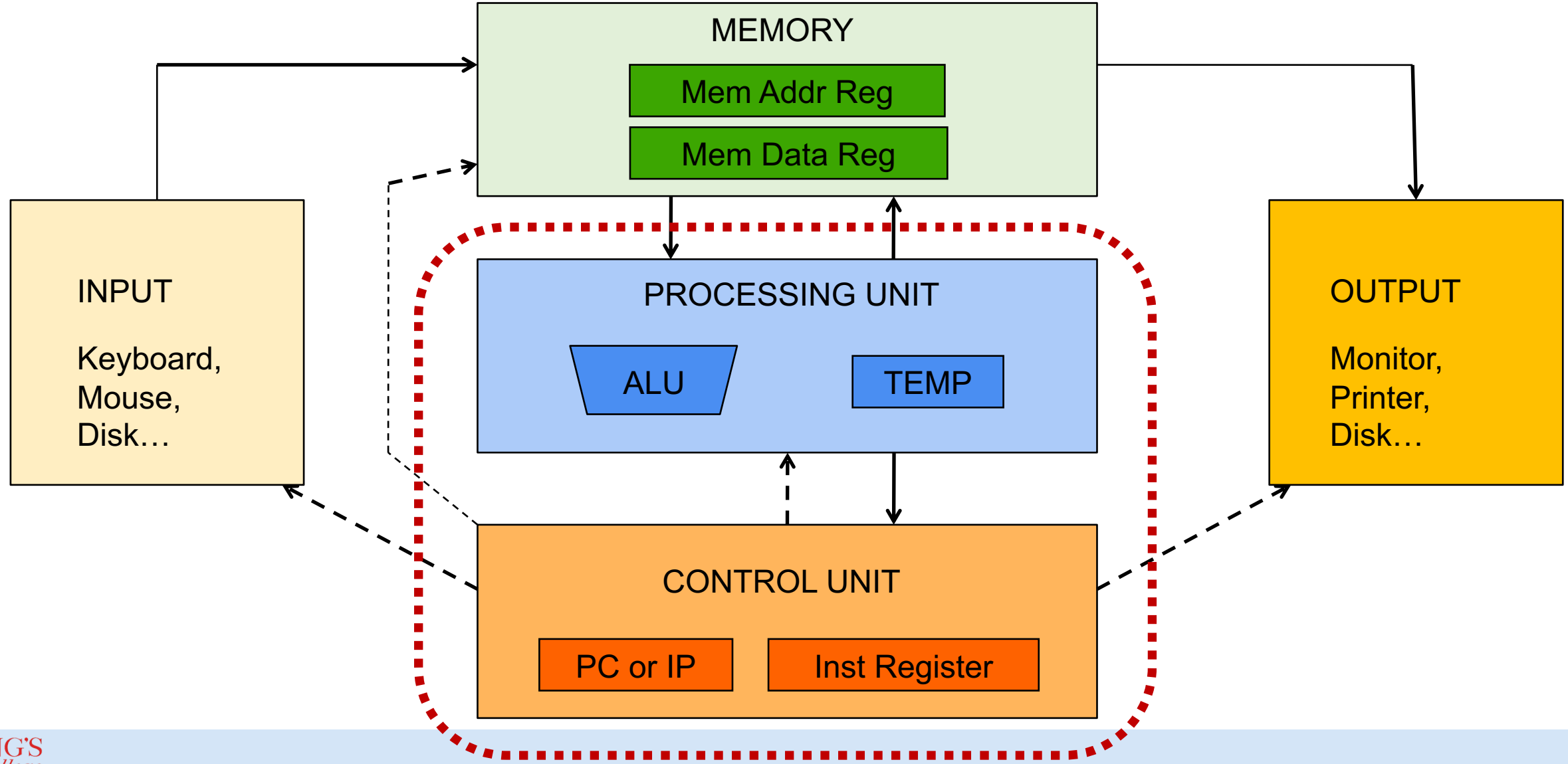
12.03.2026

What We Learned & Will Learn

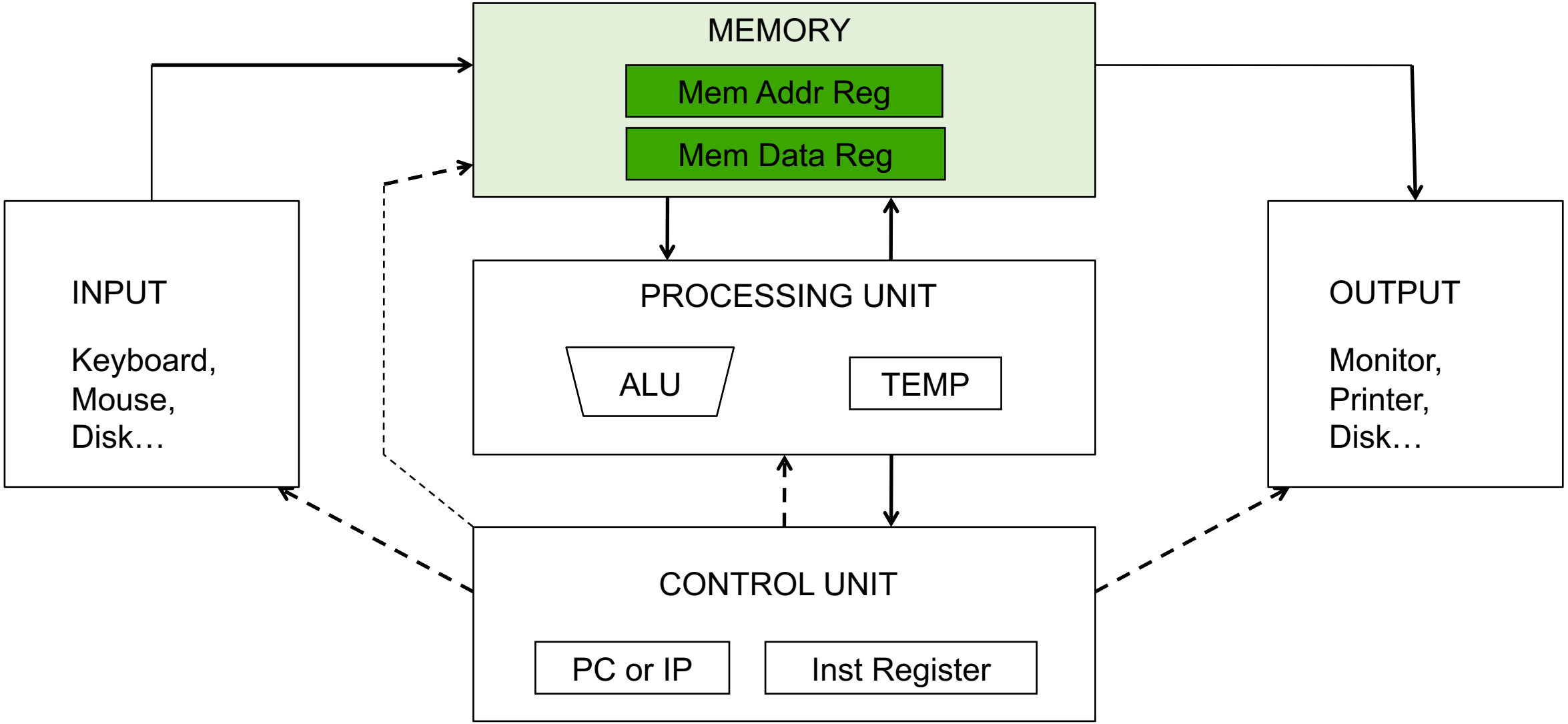
- The von Neumann model
- Instruction Set Architectures (ISA)
- Assembly programming: LC-3 and MIPS
- Microarchitecture: basics
- Microarchitecture: Single-cycle
- Microarchitecture: Multi-cycle
- Pipelining
- **Cache and Memory**



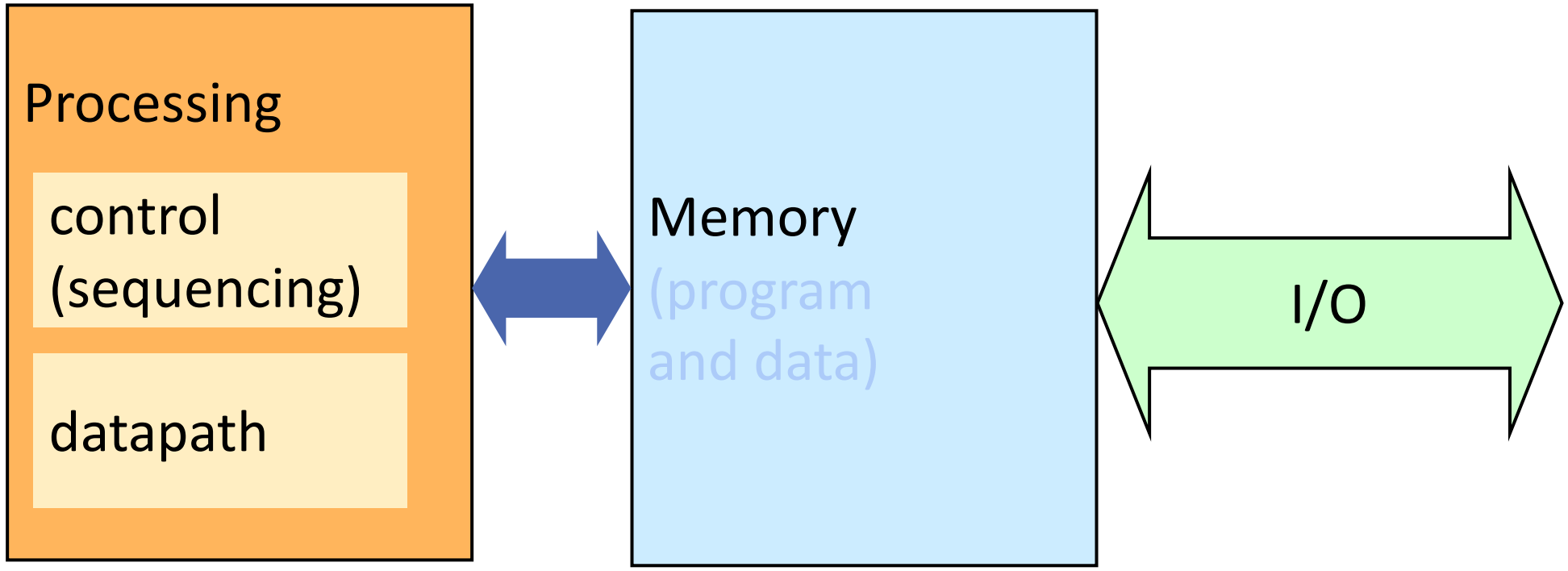
The von Neumann Model



The von Neumann Model



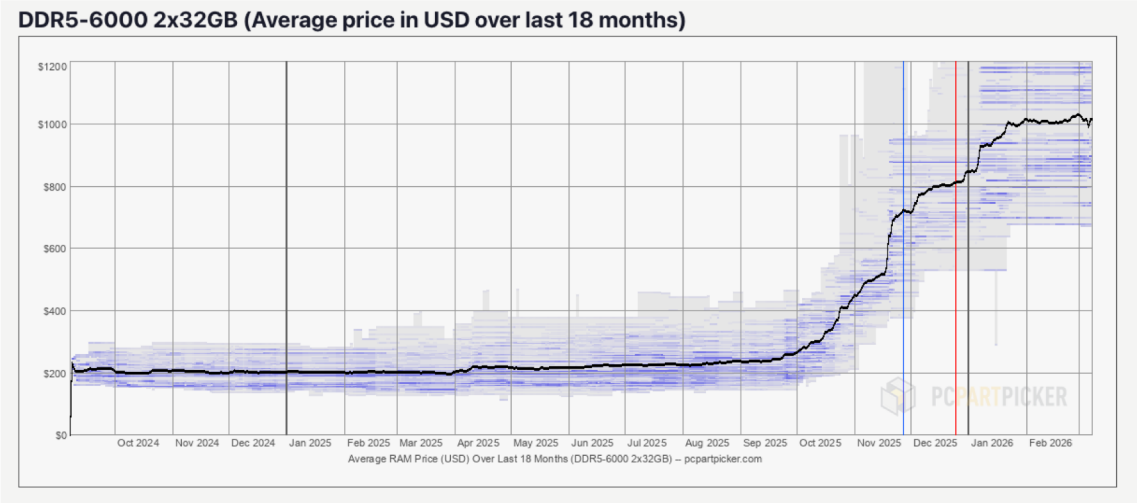
What is A Computer?



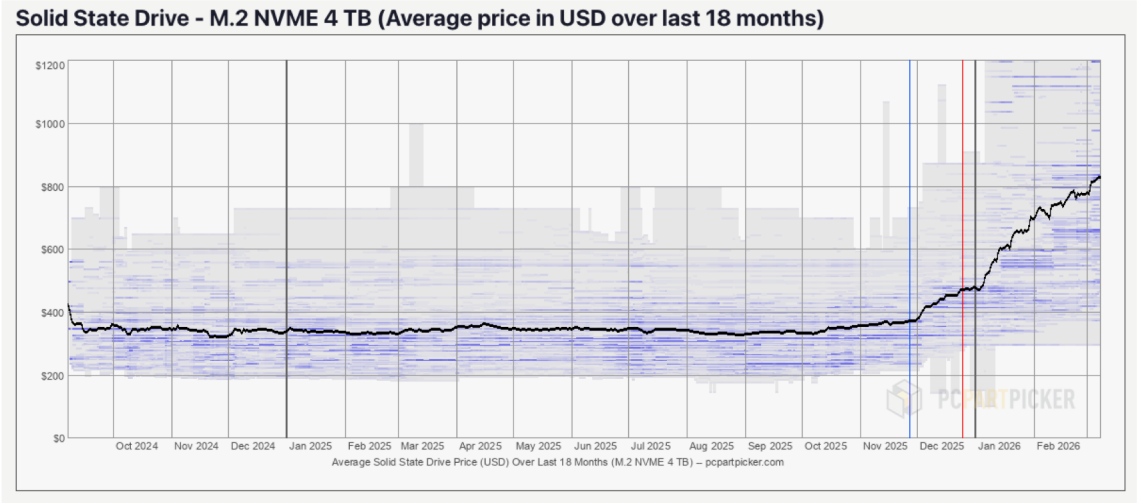
Memory Is Critically Important

An Eye-opening Memory Fact

- Memory prices are experiencing an unprecedented surge in **early 2026**, with DRAM contract prices forecasted to **rise 90–95%** quarter-over-quarter and NAND flash by **55–60%**.



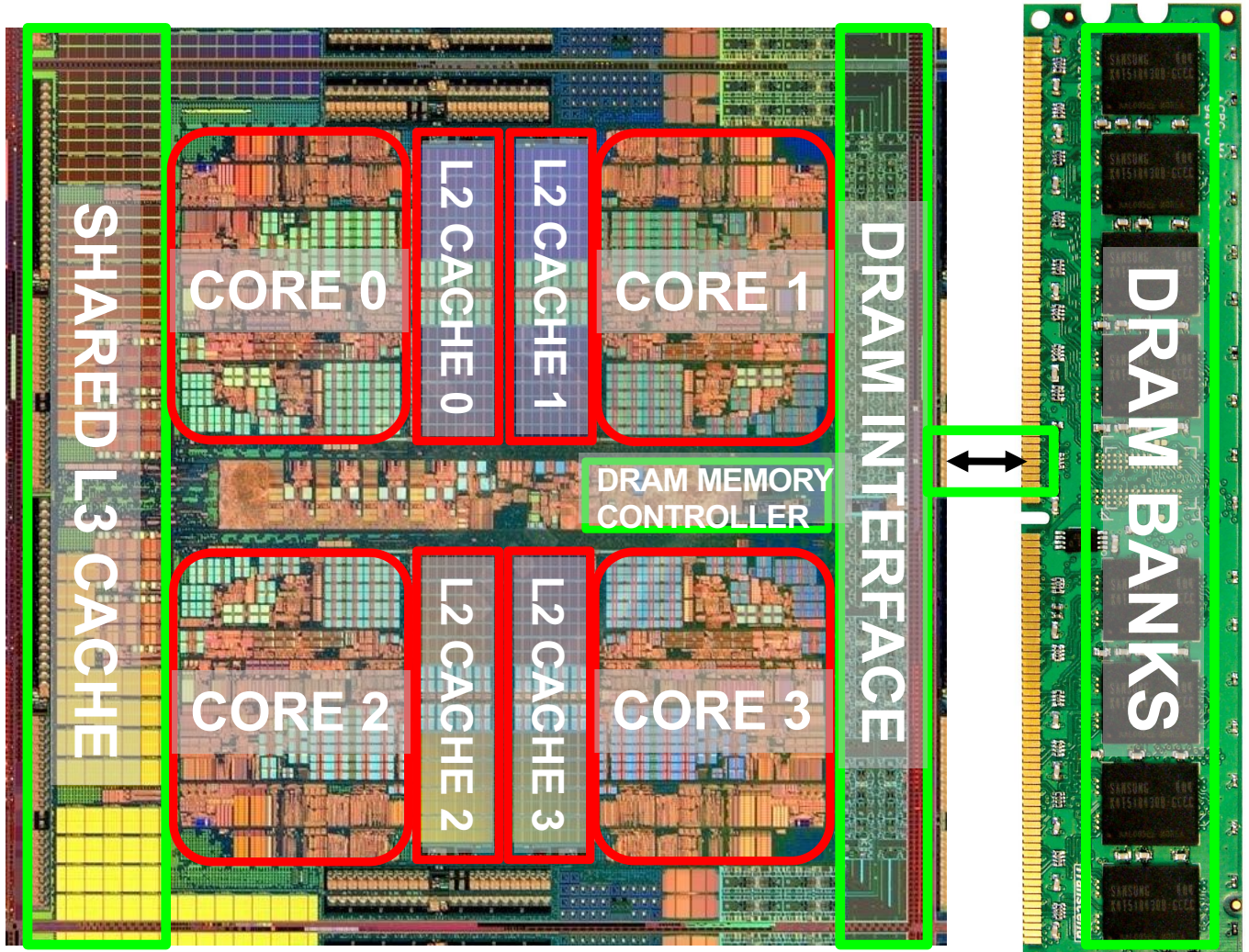
https://pcpartpicker.com/trends/price/memory/



https://pcpartpicker.com/trends/price/internal-hard-drive/

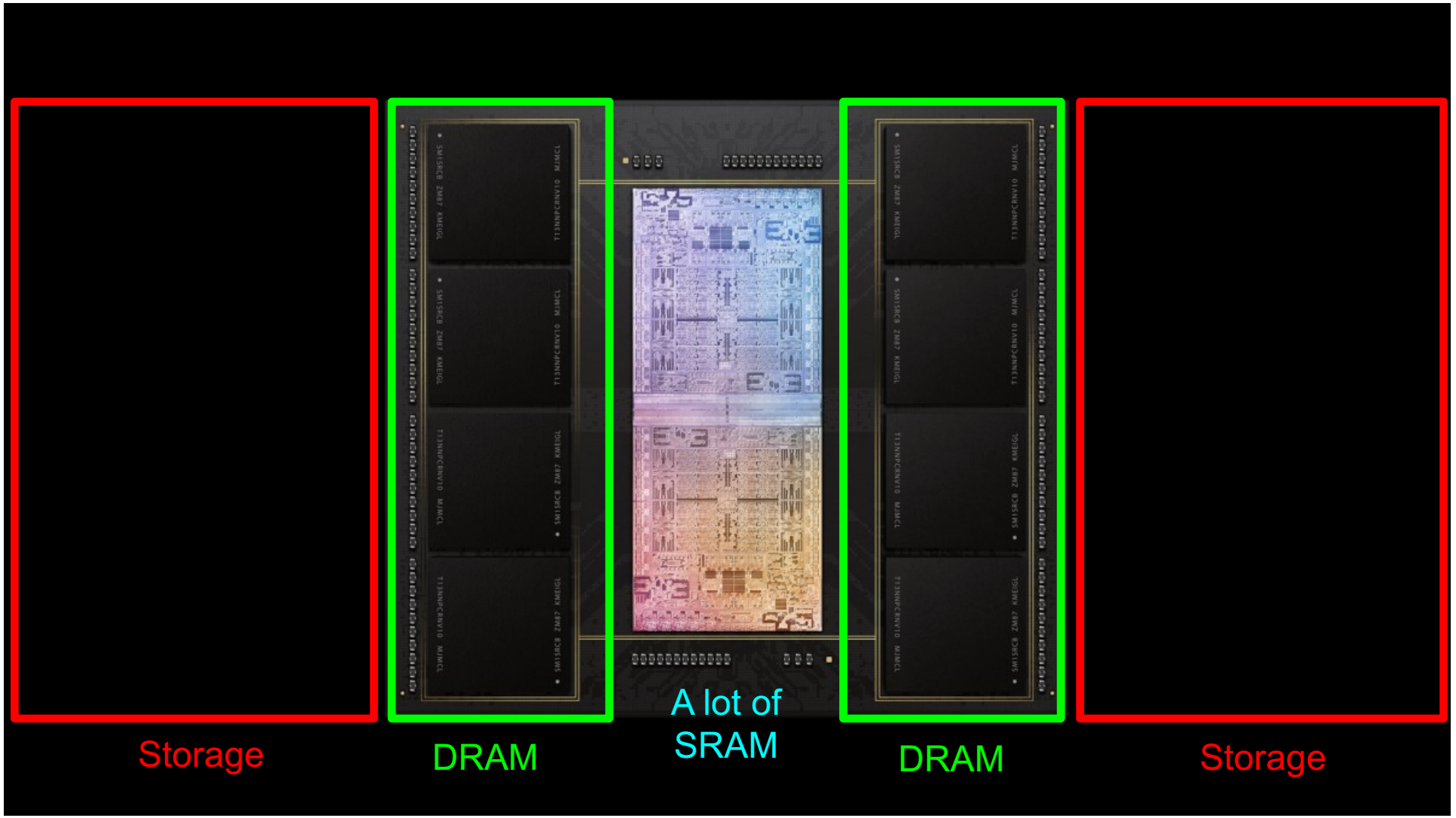
- **Key Driver: AI Demand Surge -** The need for high-end, memory-intensive AI models has caused massive demand for HBM (High Bandwidth Memory) and DDR5.

Memory Examples in Modern Systems (I)



AMD Barcelona, 2006

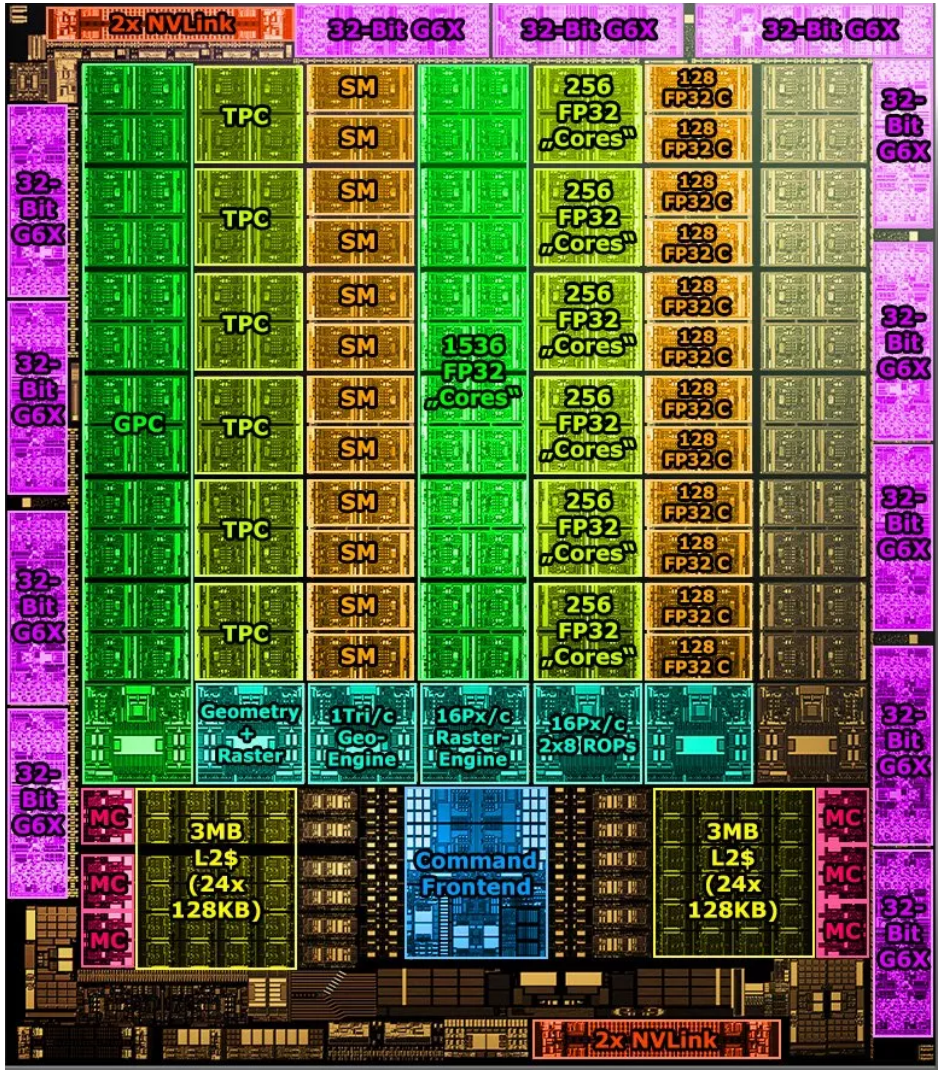
Memory Examples in Modern Systems (II)



Apple M1 Ultra System (2022)

https://www.gsmarena.com/apple_announces_m1_ultra_with_20core_cpu_and_64core_gpu-news-53481.php

Memory Examples in Modern Systems (III)



Cores:

128 Streaming Multiprocessors

L1 Cache or Scratchpad:

192KB per SM

Can be used as L1 Cache and/or Scratchpad

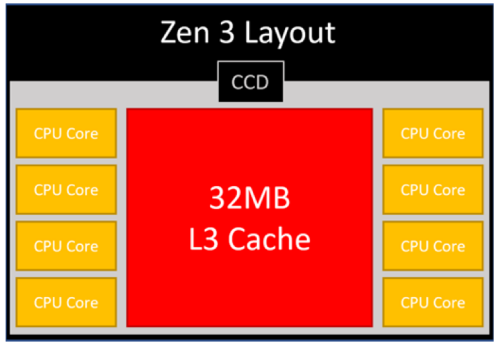
L2 Cache:

40 MB shared

Nvidia Ampere, 2020

https://www.tomshardware.com/news/infrared-photographer-photos-nvidia-ga102-ampere-silicon

Memory Examples in Modern Systems (IV)

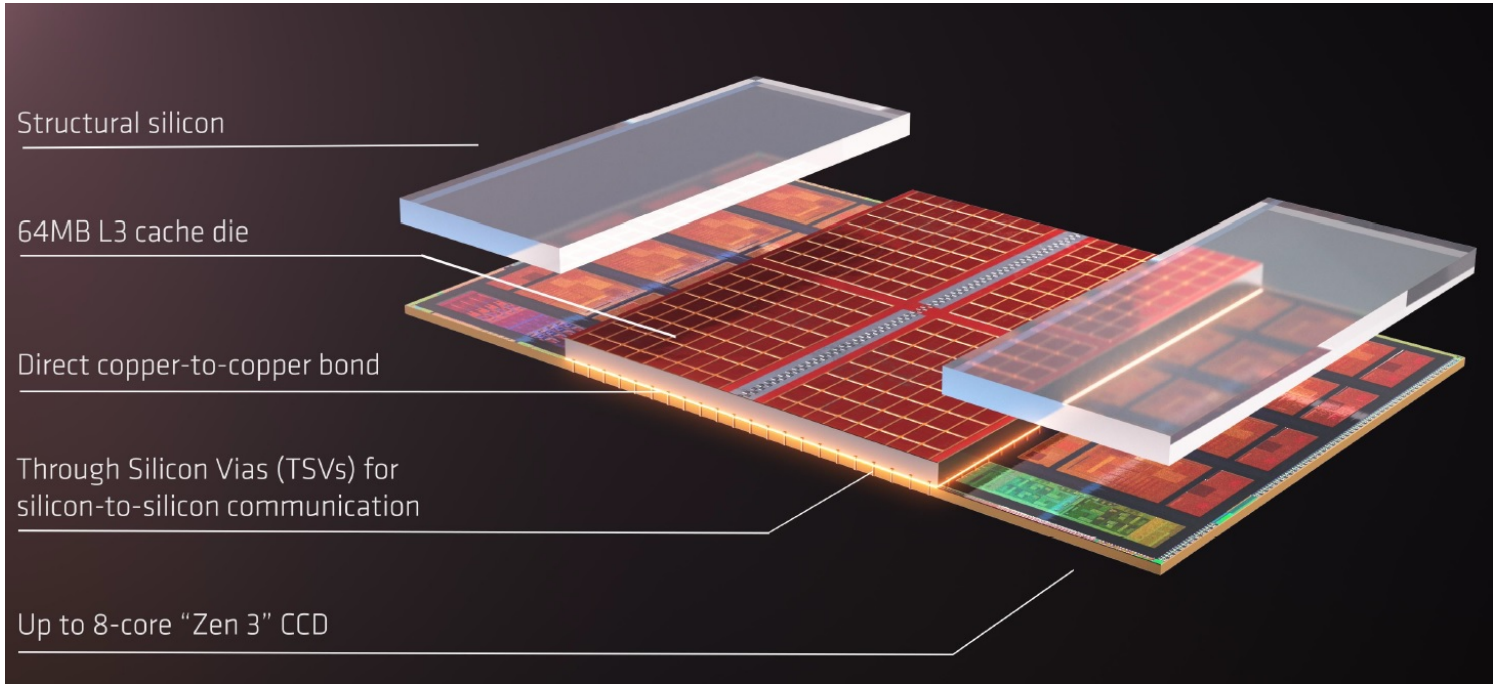


https://community.microcenter.com/discussion/5134/comparing-zen-3-to-zen-2

AMD increases the L3 size of their 8-core Zen 3 processors from 32 MB to 96 MB

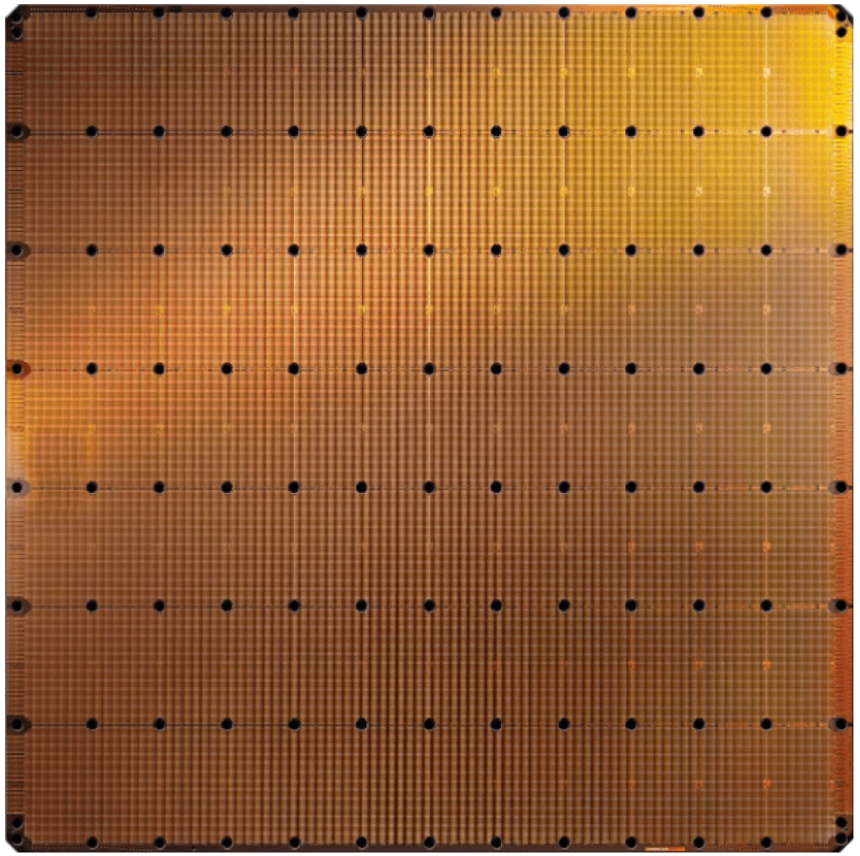
Additional 64 MB L3 cache die stacked on top of the processor die

- Connected using Through Silicon Vias (TSVs)
- Total of 96 MB L3 cache



https://youtu.be/gqAYMx34euU
https://www.tech-critter.com/amd-keynote-computex-2021/

Cerebras's Wafer Scale Engine-2 (V)



Cerebras WSE-2
2.6 Trillion transistors
46,225 mm^2

- The largest ML accelerator chip
- 850,000 cores
- **40 GB of on-chip memory**
- **20 PB/s memory bandwidth**

Largest GPU
54.2 Billion transistors
826 mm^2

NVIDIA Ampere GA100

https://cerebras.net/product/#overview

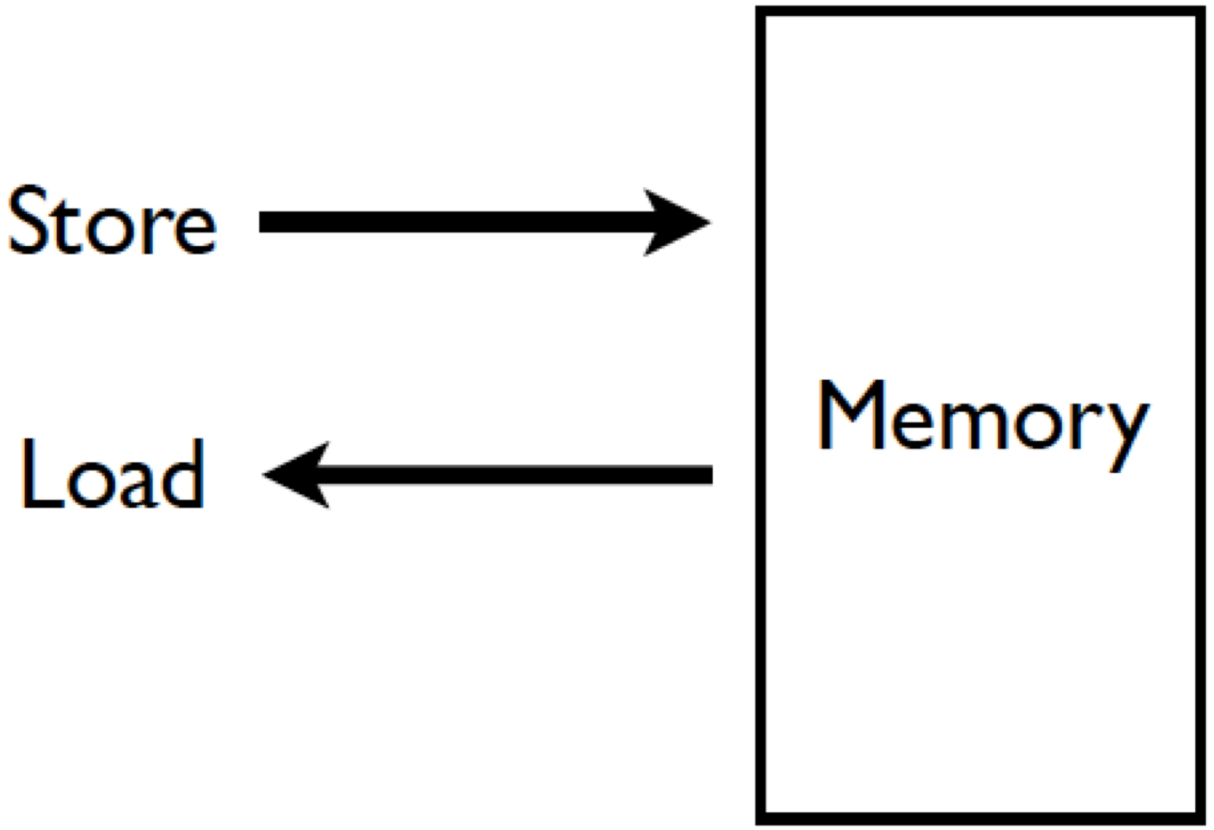
Hardware Design

Memory Fundamentals

Hardware Design

Memory Organization & Technology

Memory (Programmer's View)



Abstraction: Virtual vs. Physical Memory

- Programmer sees virtual memory
  - Can assume the memory is "infinite"
- Reality: Physical memory size is much smaller than what the programmer assumes
- The system (system software + hardware, cooperatively) maps virtual memory addresses to physical memory
  - The system automatically manages the physical memory space transparently to the programmer

+ Programmer does not need to know the physical size of memory nor manage it → A small physical memory can appear as a huge one to the programmer → Life is easier for the programmer

-- More complex system software and architecture

A classic example of the programmer/(micro)architect tradeoff

(Physical) Memory System

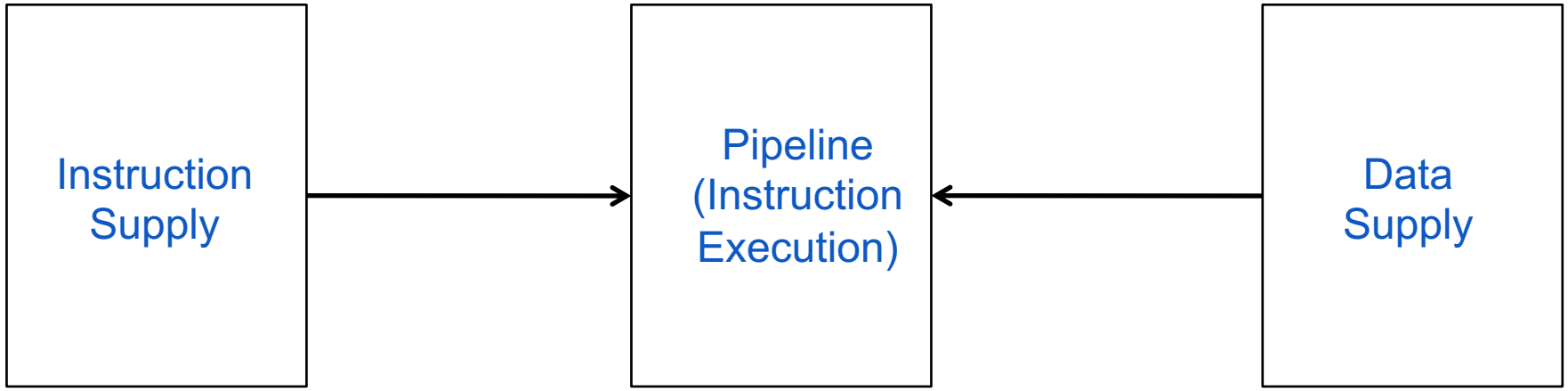
- You need a larger level of storage to manage a small amount of physical memory automatically
  → Physical memory has a backing store: disk

- We will first start with the physical memory system

- For now, ignore the virtual→physical indirection

- We will get back to it later, if time permits…

Idealism



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

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

- Zero latency access
- Infinite capacity
- Infinite bandwidth
- Zero cost

Ideal Memory

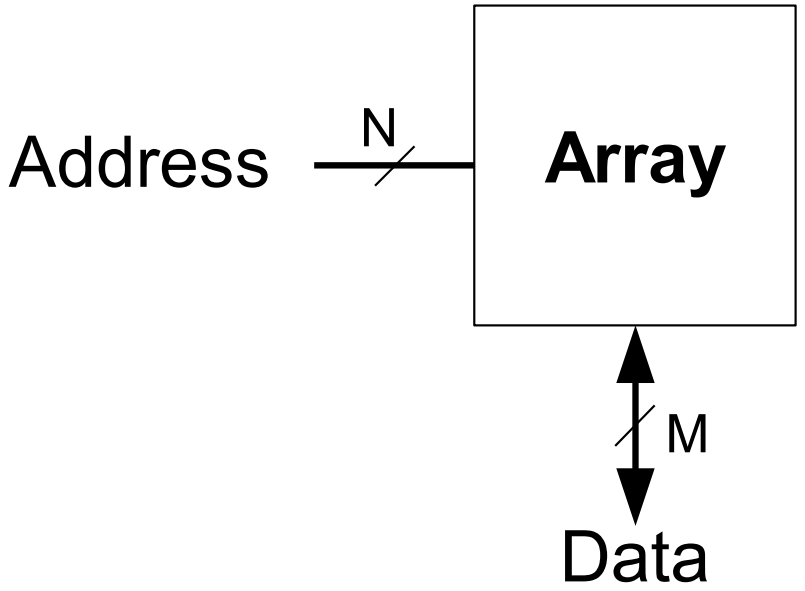
- Zero access time (latency)
- Infinite capacity
- Zero cost
- Infinite bandwidth (to support multiple accesses in parallel)
- Zero energy

Quick Overview of Memory Arrays

- Flip-Flops (or Latches)
  - Very fast
  - Very expensive (one bit costs tens of transistors)
- Static RAM (we will describe it soon)
  - Relatively fast
  - Expensive (one bit costs 6+ transistors)
- Dynamic RAM (we will describe it soon)
  - Slower, reading and charge leakage destroy content (refresh), needs a special process for manufacturing (due to the capacitor)
  - Cheap (one bit costs only one transistor plus one capacitor)
- Other storage technology (flash memory, hard disk, tape)
  - Much slower, special manufacturing needed, non-volatile
  - Very cheap (one transistor stores many bits or no transistors involved)

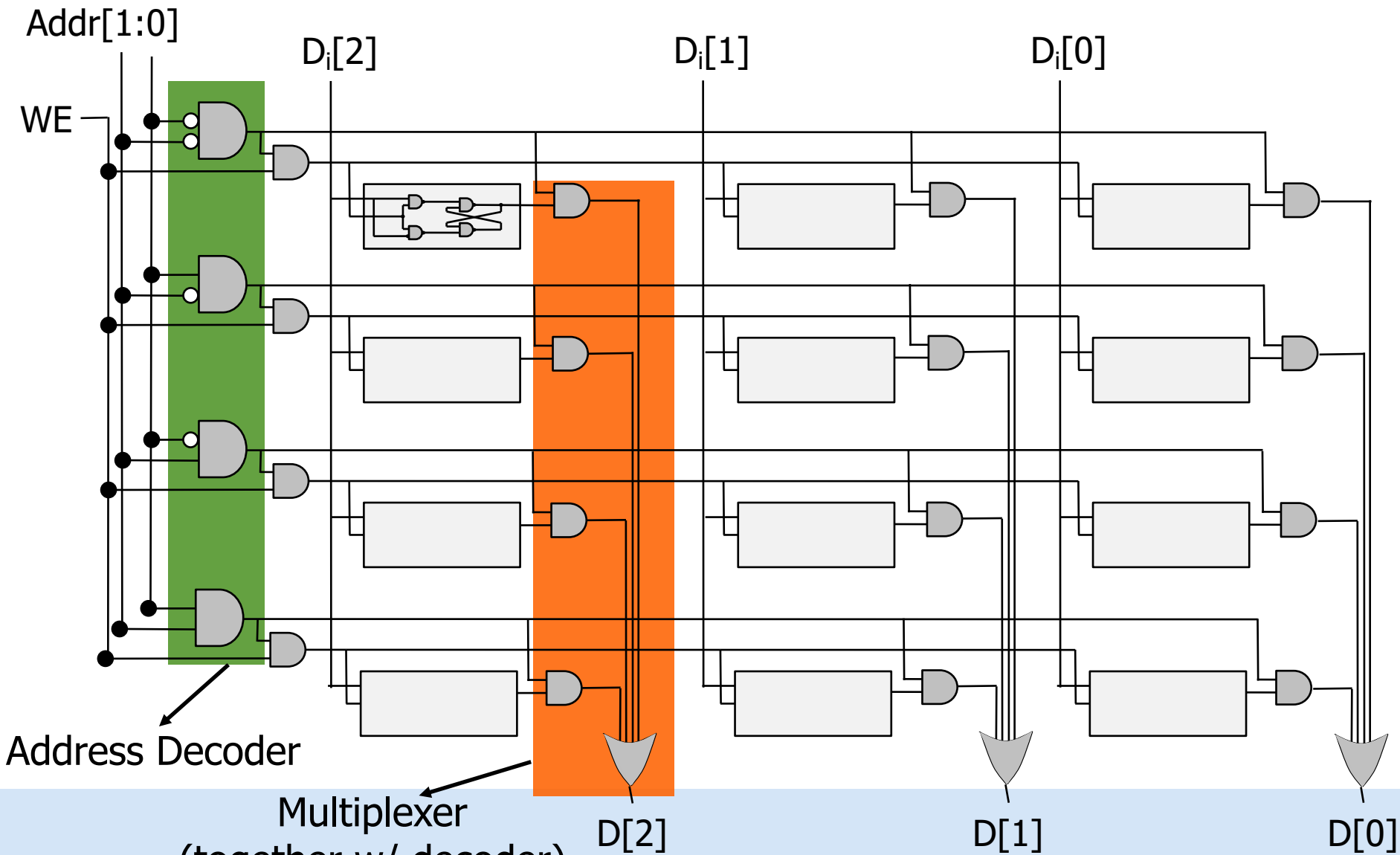
Array Organization of Memories

- Goal: Efficiently store large amounts of data
  - A memory array (stores data)
  - Address selection logic (selects one row of the array)
  - Readout circuitry (reads data out)



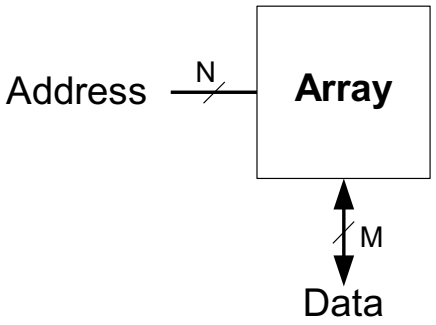
- An M-bit value can be read or written at each unique N-bit address
  - All values can be accessed, but only M-bits at a time
  - Access restriction allows more compact organization

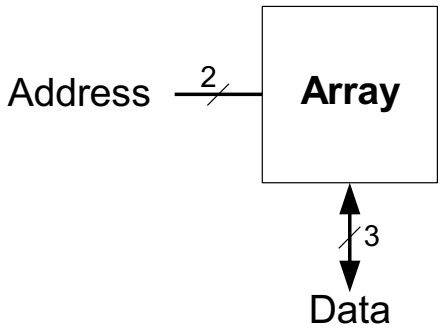
Recall: A Memory Array (4 locations x 3 bits)

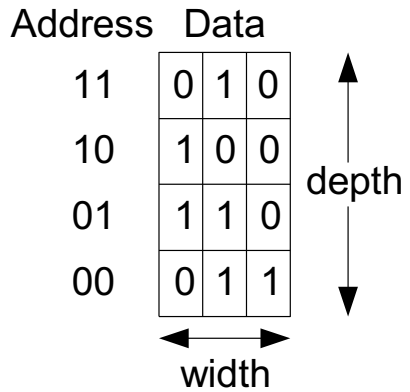


Memory Arrays

- Two-dimensional array of bit cells
  - Each bit cell stores one bit
- An array with N address bits and M data bits:
  - 2^N rows and M columns
  - Depth: number of rows (can be number of "words")
  - Width: number of columns (can be the "word" size)
  - Array size: depth × width (rows x columns) = $2^N \times M$

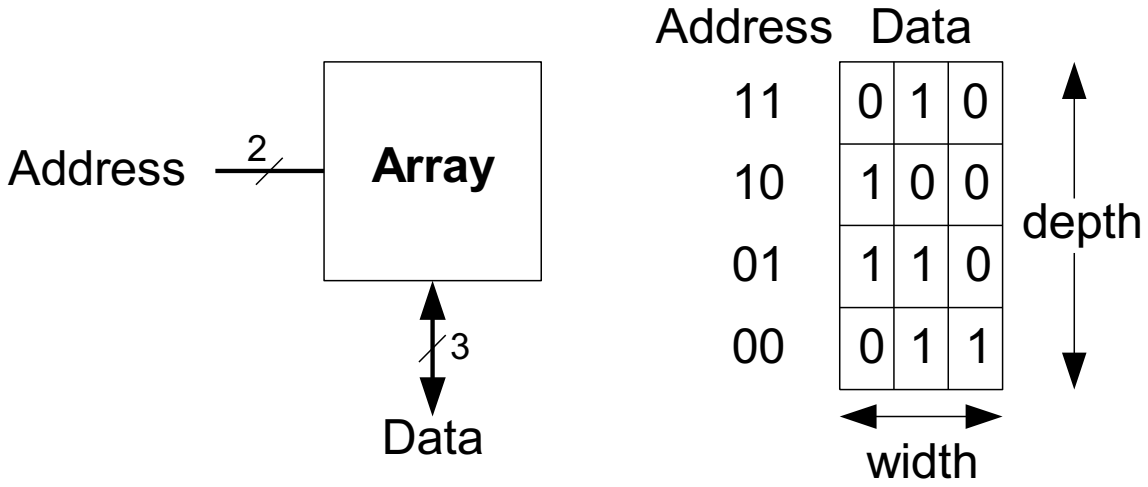




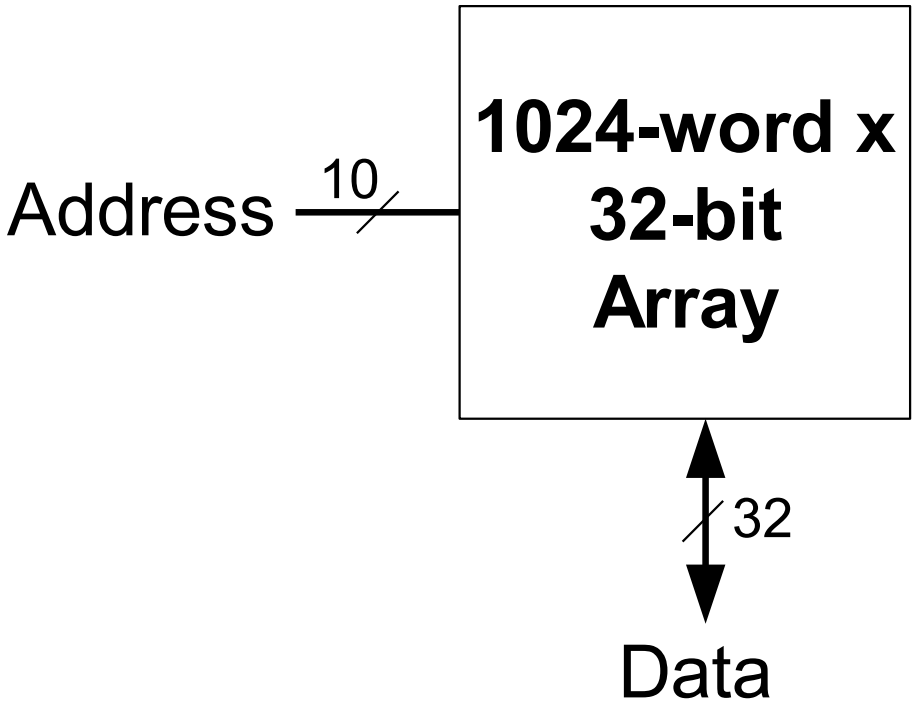


Memory Array Example

- $2^2 \times 3$-bit array
- Number of rows: 4
- Row size: 3 bits
- For example, the 3-bit data stored at row 10 is 100

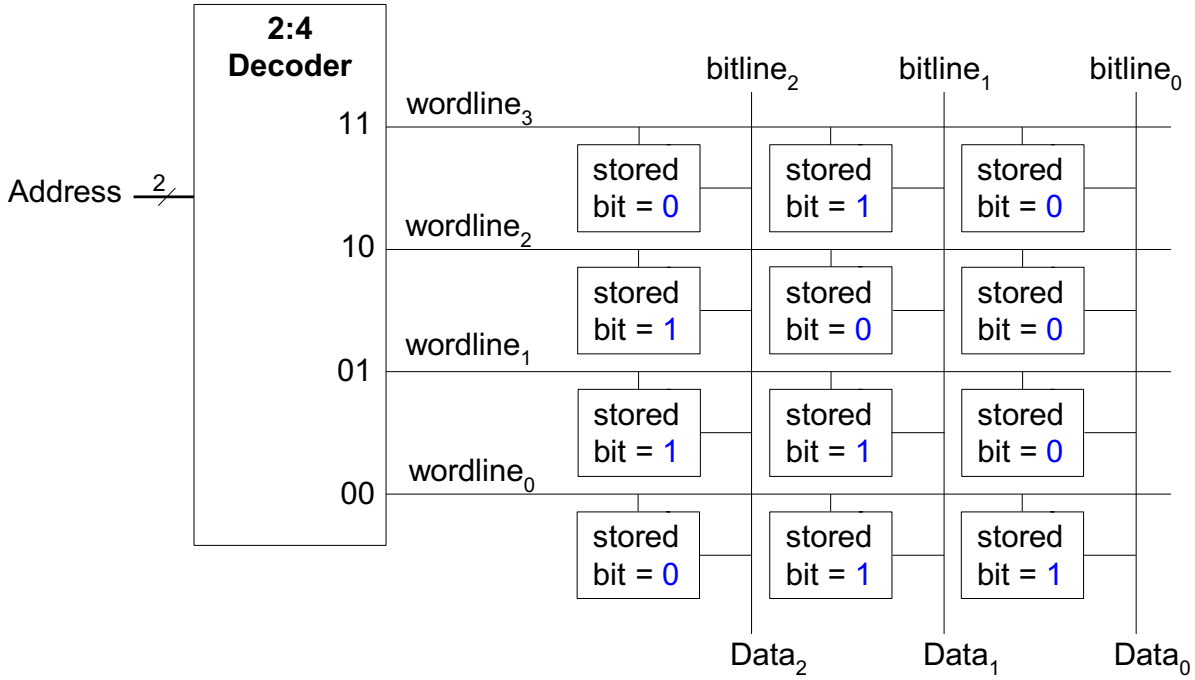


Larger and Wider Memory Array Example



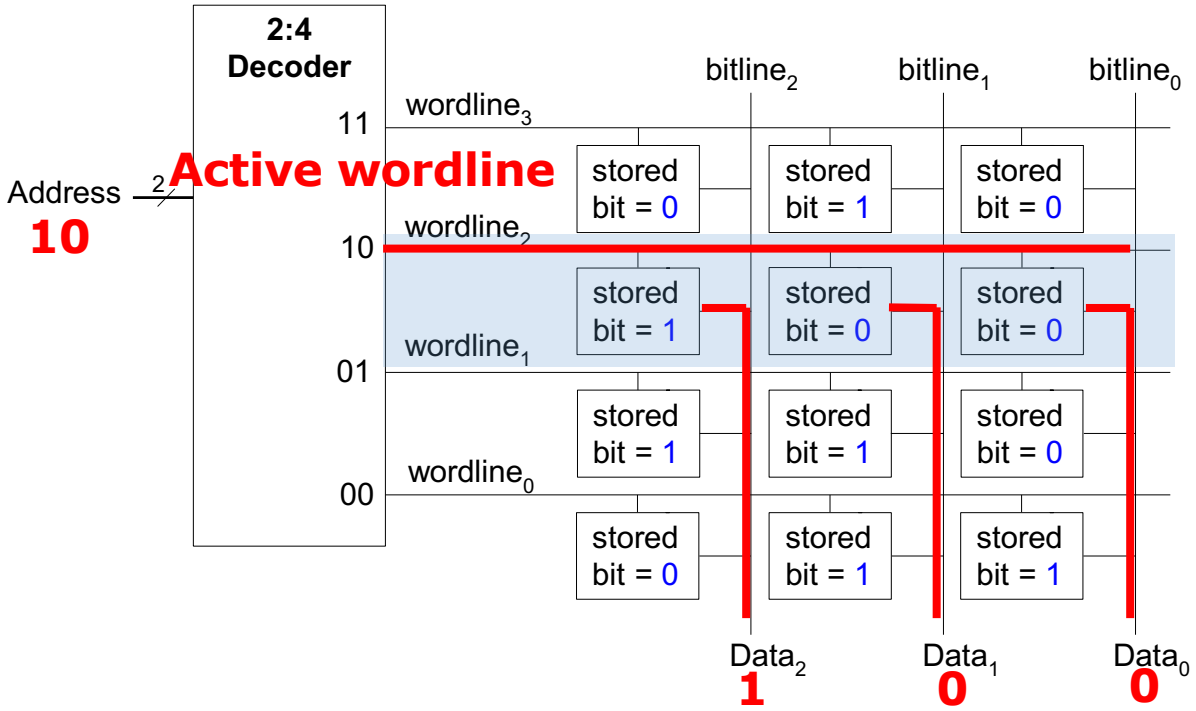
Memory Array Organization (I)

- Storage nodes in one column connected to one bitline
- Address decoder activates only ONE wordline
- Content of one line of storage available at output



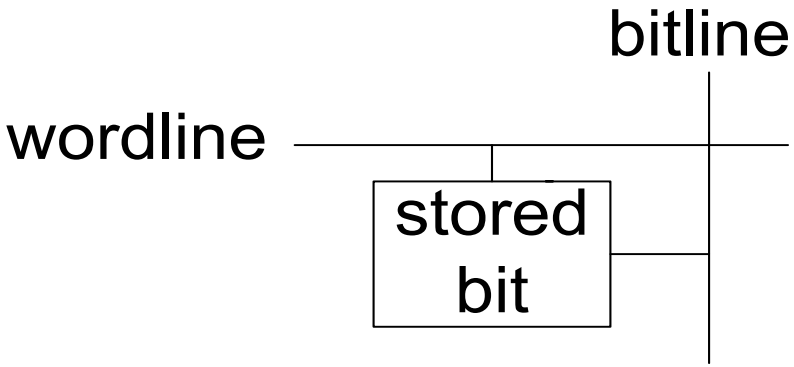
Memory Array Organization (II)

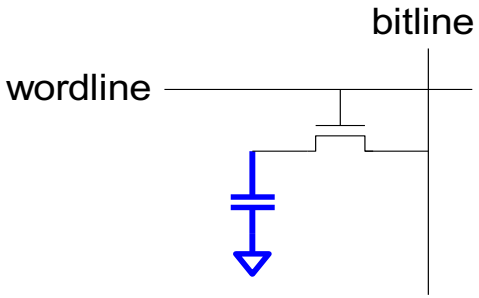
- Storage nodes in one column connected to one bitline
- Address decoder activates only ONE wordline
- Content of one line of storage available at output



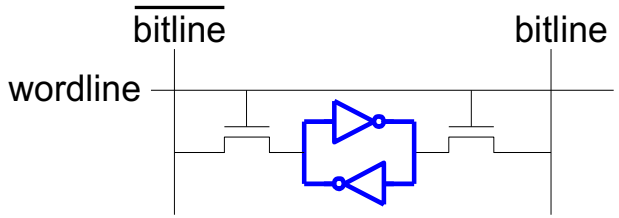
How is Access Controlled?

- Access transistors (that are configured as switches) connect the bit storage to the bitline
- Access is controlled by the wordline





DRAM



SRAM

Building Larger Memories

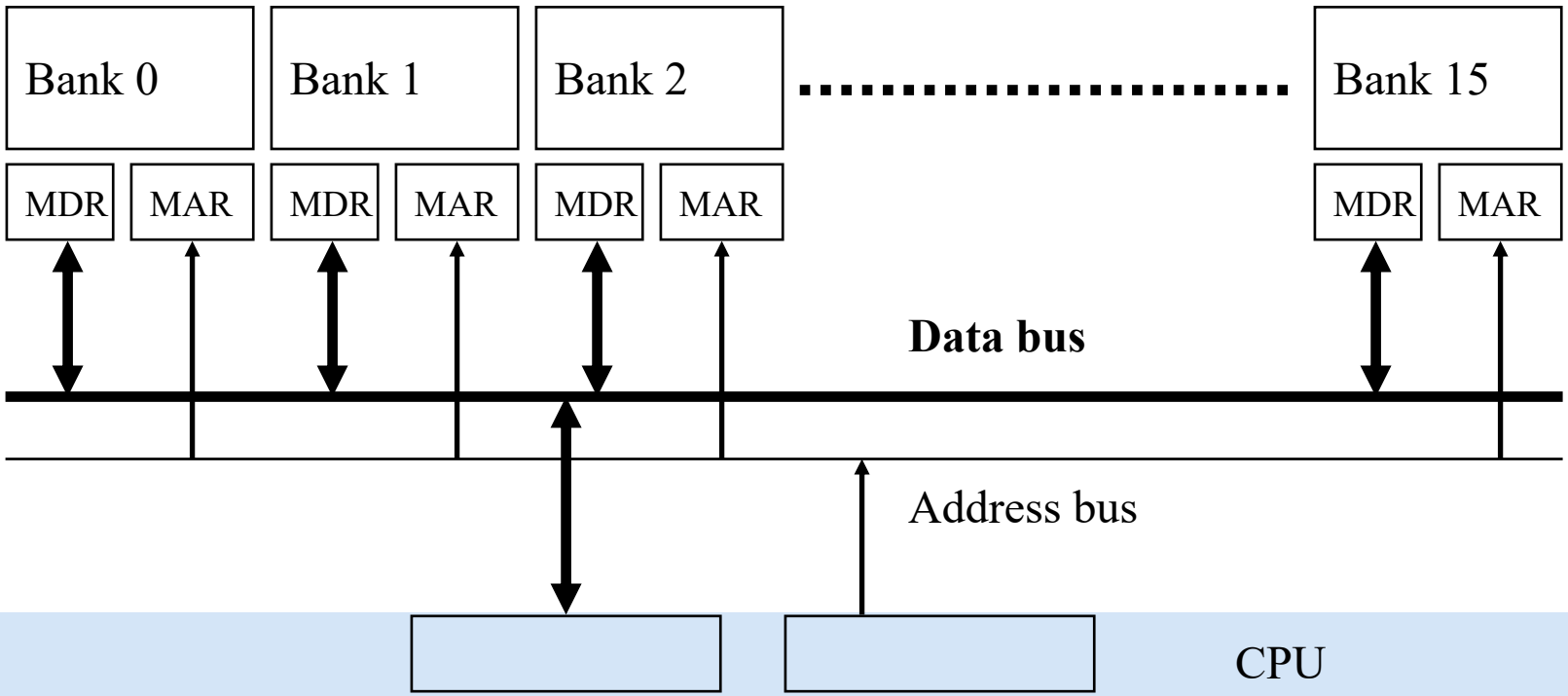
- Requires larger memory arrays
- Large → slow
- How do we make the memory large without making it too slow?
- Idea: Divide the memory into smaller arrays and interconnect the arrays to input/output buses
  - Large memories are hierarchical array structures
  - DRAM: Channel → Rank → Bank → Subarrays → Mats

General Principle: Interleaving (Banking)

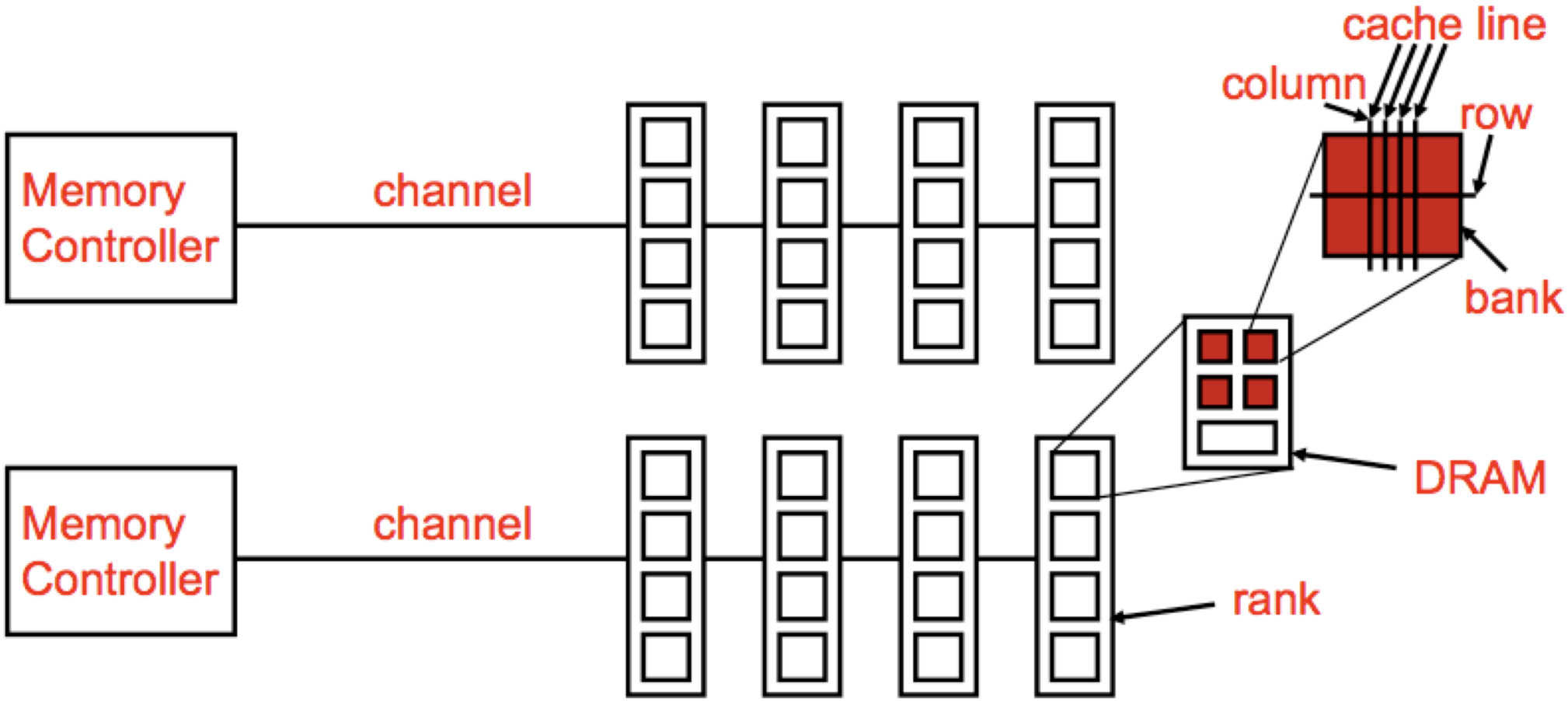
- Interleaving (banking)
  - Problem: a single monolithic large memory array takes long to access and does not enable multiple accesses in parallel
  - Goal: Reduce the latency of memory array access and enable multiple accesses in parallel
  - Idea: Divide a monolithic large array into multiple banks that can be accessed independently (in the same cycle or in consecutive cycles)
    - Each bank is smaller than the entire memory storage
    - Accesses to different banks can be overlapped
  - A Key Issue: How do you map data to different banks? (i.e., how do you interleave data across banks?)

Memory Banking

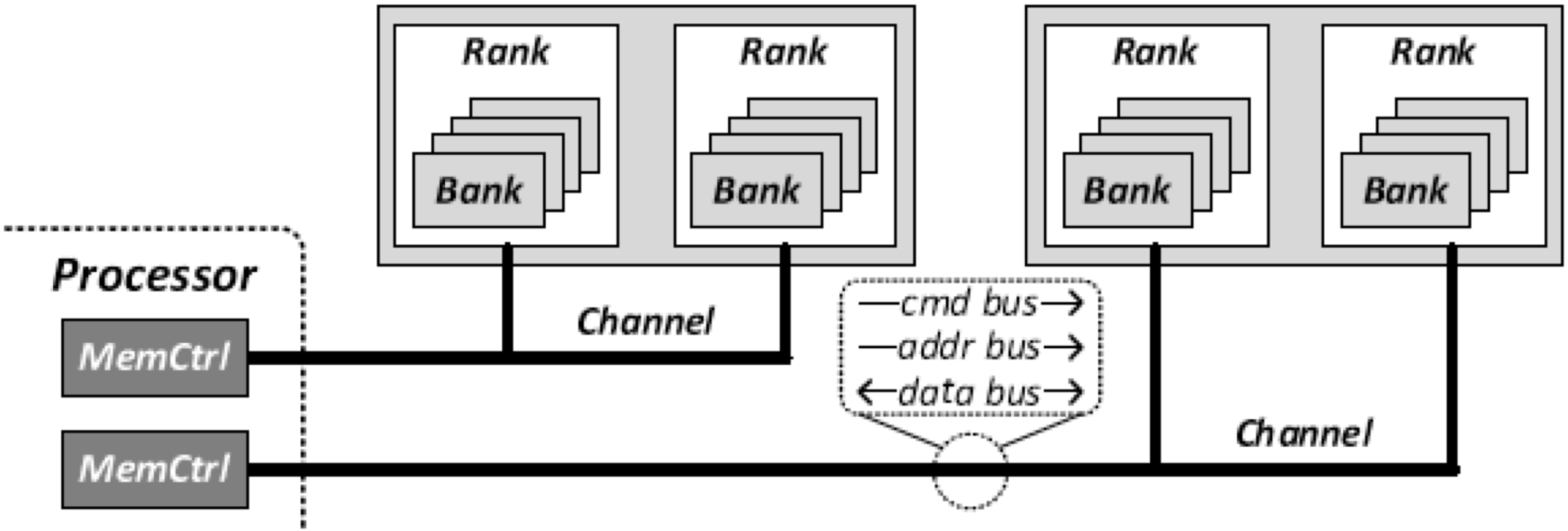
- Memory is divided into banks that can be accessed independently; banks share address and data buses (to reduce memory chip pins)
- Can start and complete one bank access per cycle
- Can sustain N concurrent accesses if all N go to different banks



Generalized Memory Structure



Generalized Memory Structure

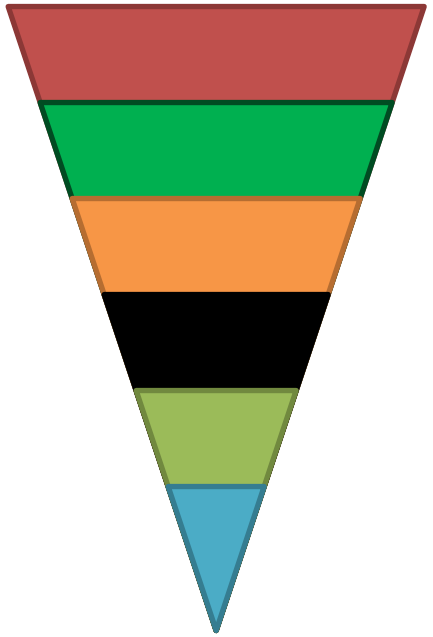


Hardware Design

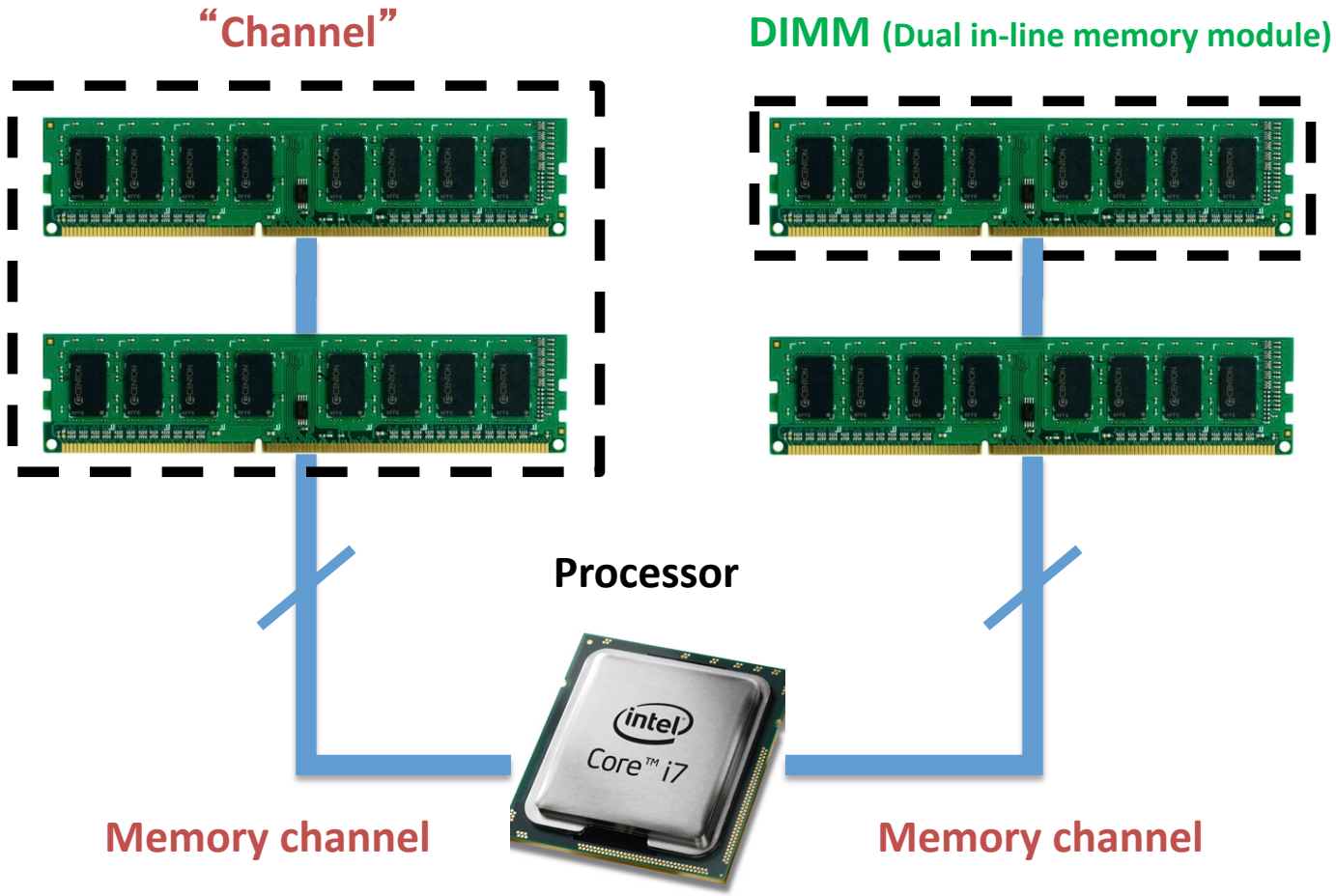
The DRAM Subsystem
A Top-Down View

DRAM Subsystem Organization

- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column



The DRAM Subsystem



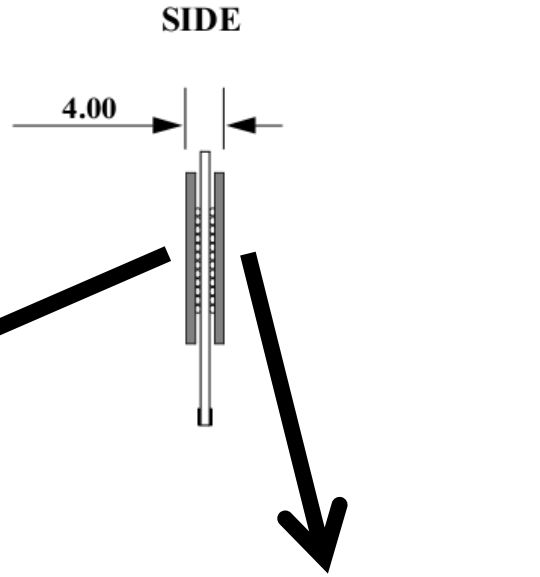
Breaking down a DIMM (module)

DIMM (Dual in-line memory module)

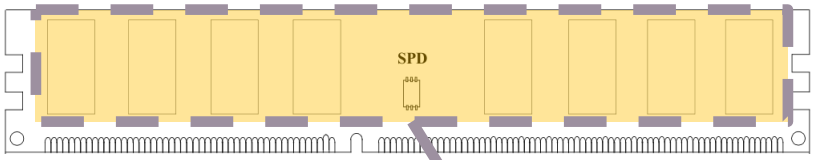
Breaking down a DIMM (module)

DIMM (Dual in-line memory module)

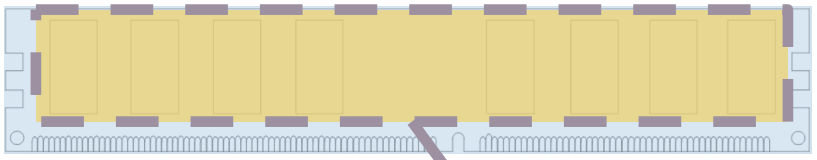
Side view



Front of DIMM



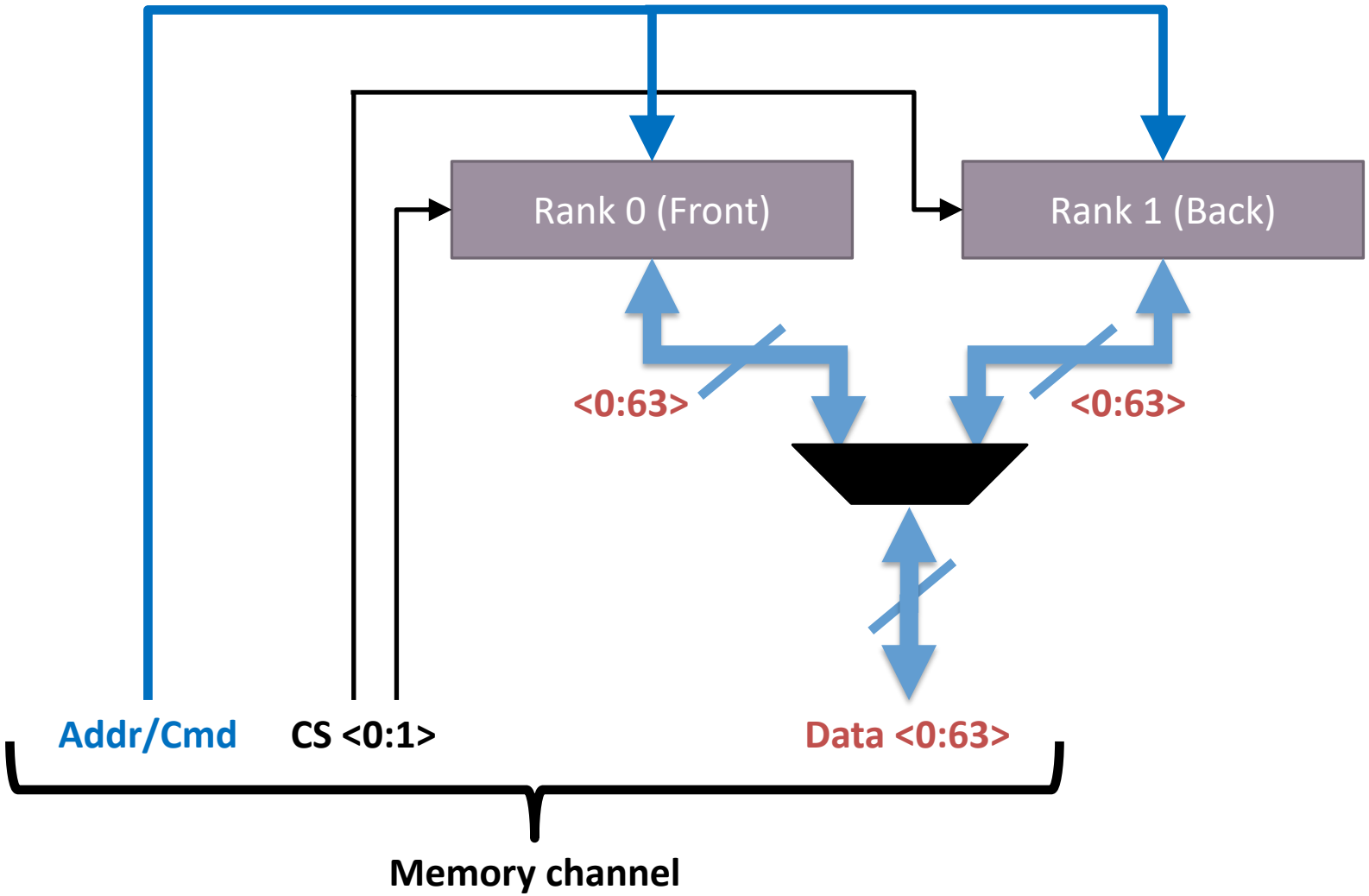
Back of DIMM



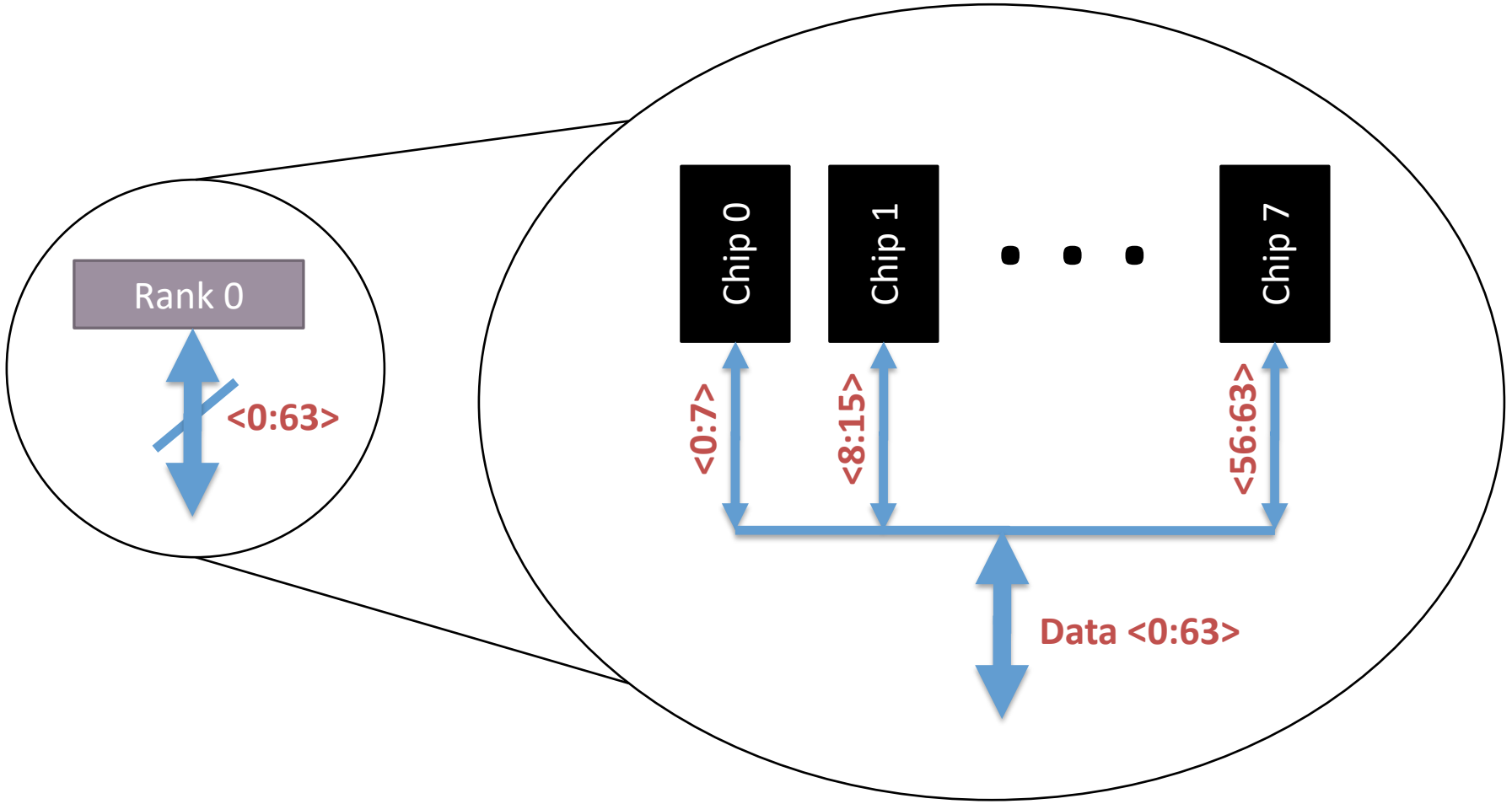
Rank 0: collection of 8 chips

Rank 1

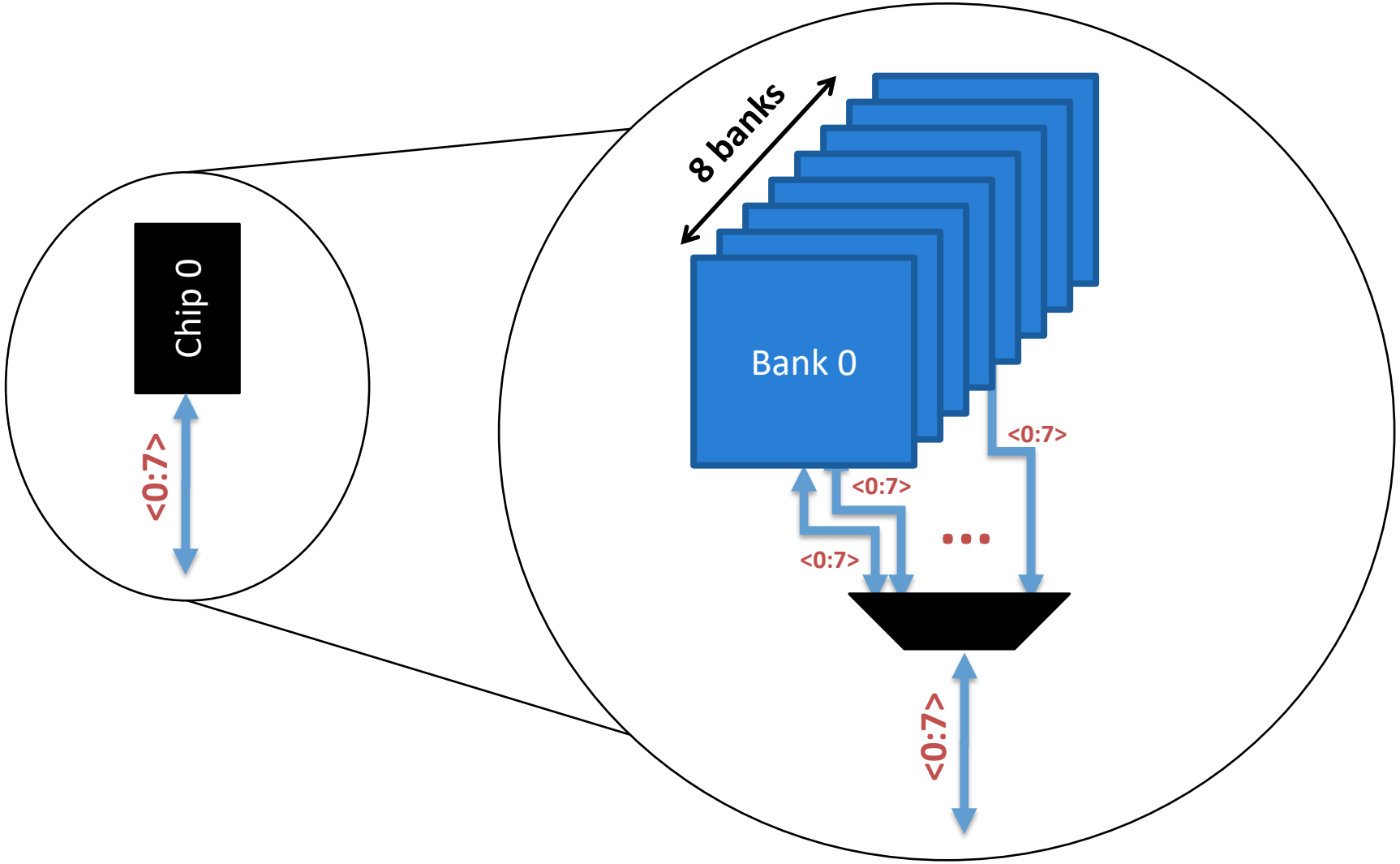
Rank


Rank 0 (Front)
Rank 1 (Back)
<0:63>
<0:63>
Addr/Cmd
CS <0:1>
Data <0:63>
Memory channel

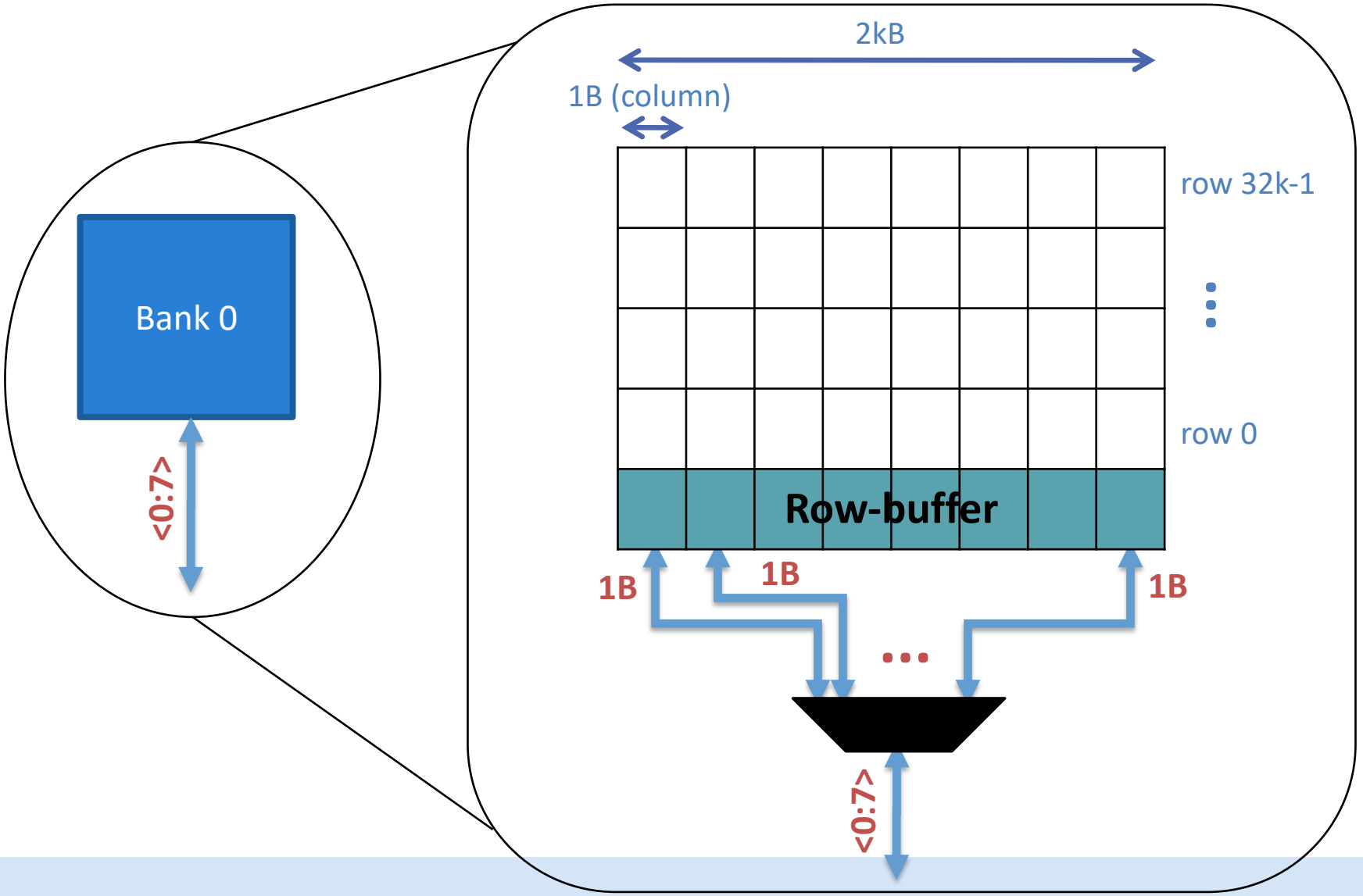
Breaking down a Rank



Breaking down a Chip

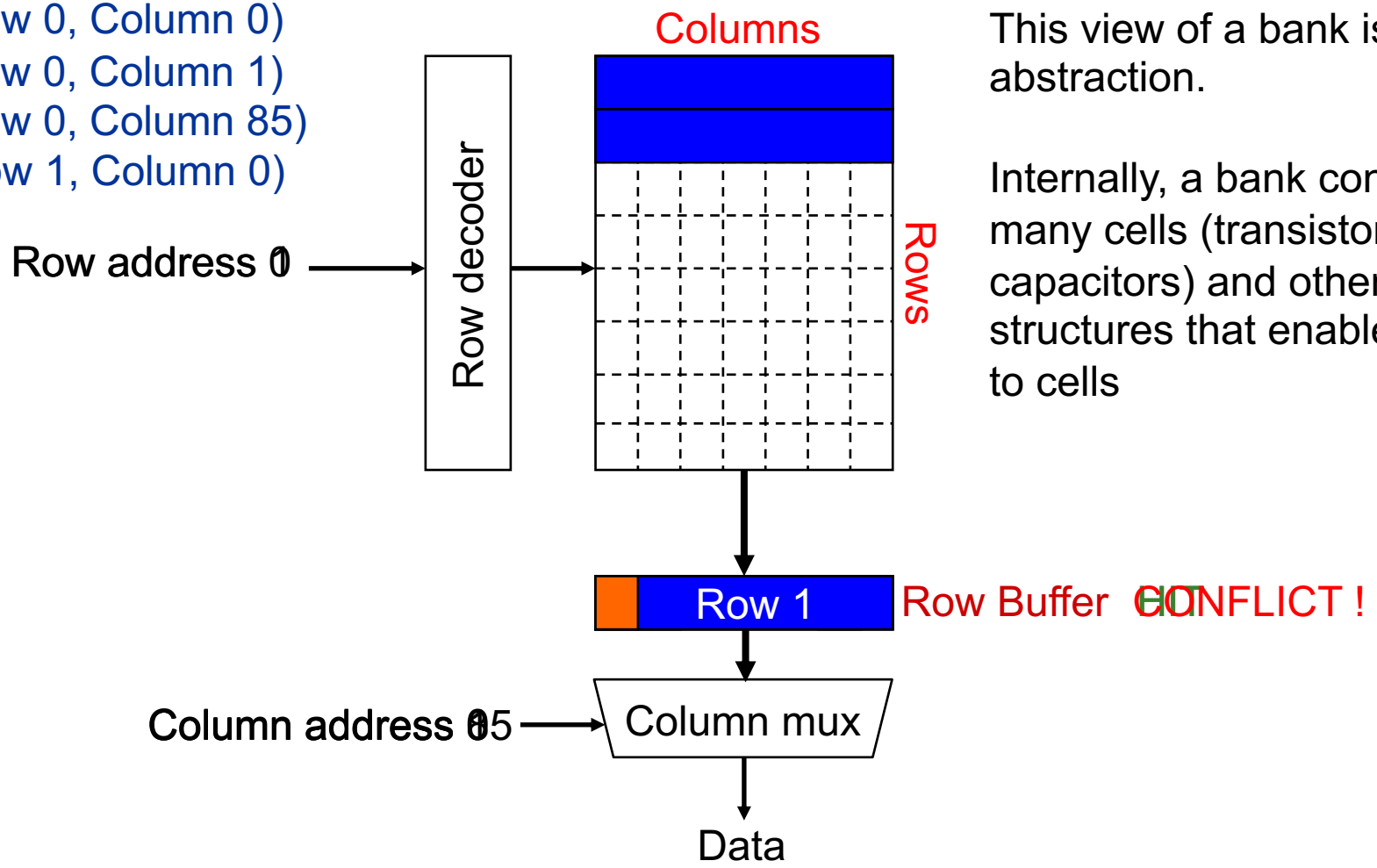


Breaking down a Bank



Digging Deeper: DRAM Bank Operation

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)



This view of a bank is an abstraction.

Internally, a bank consists of many cells (transistors & capacitors) and other structures that enable access to cells

A DRAM Bank Internally Has Sub-Banks

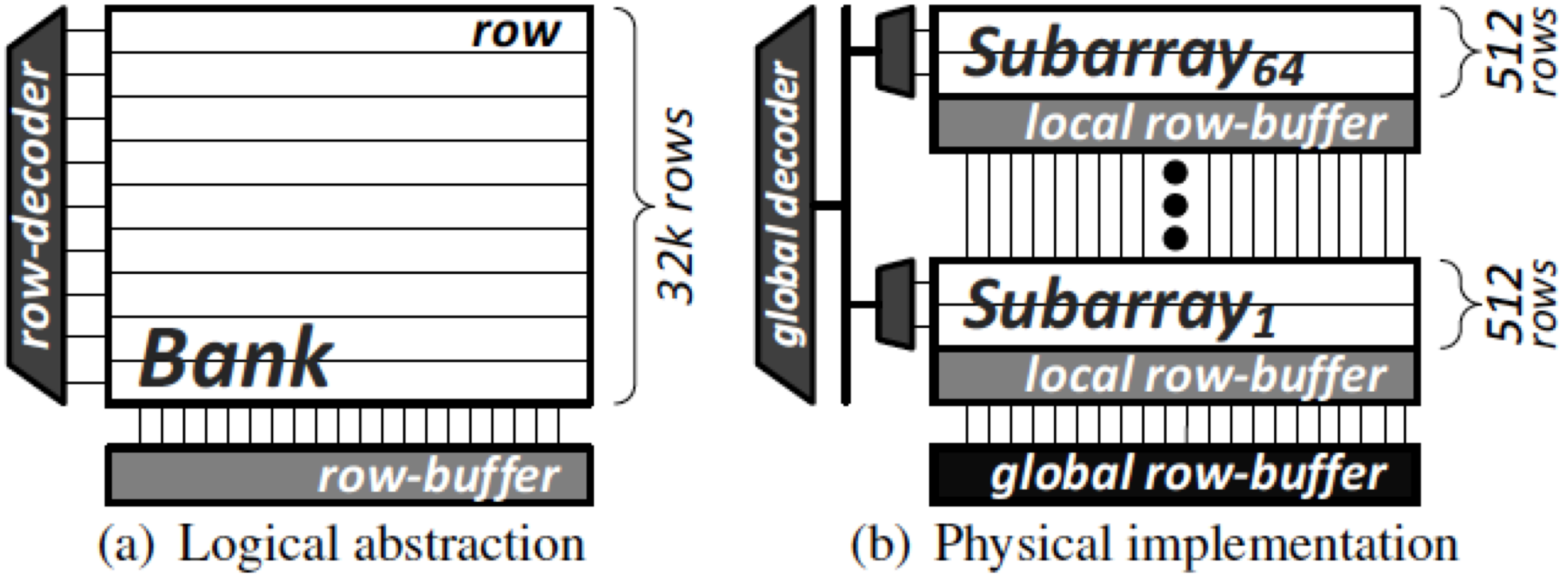
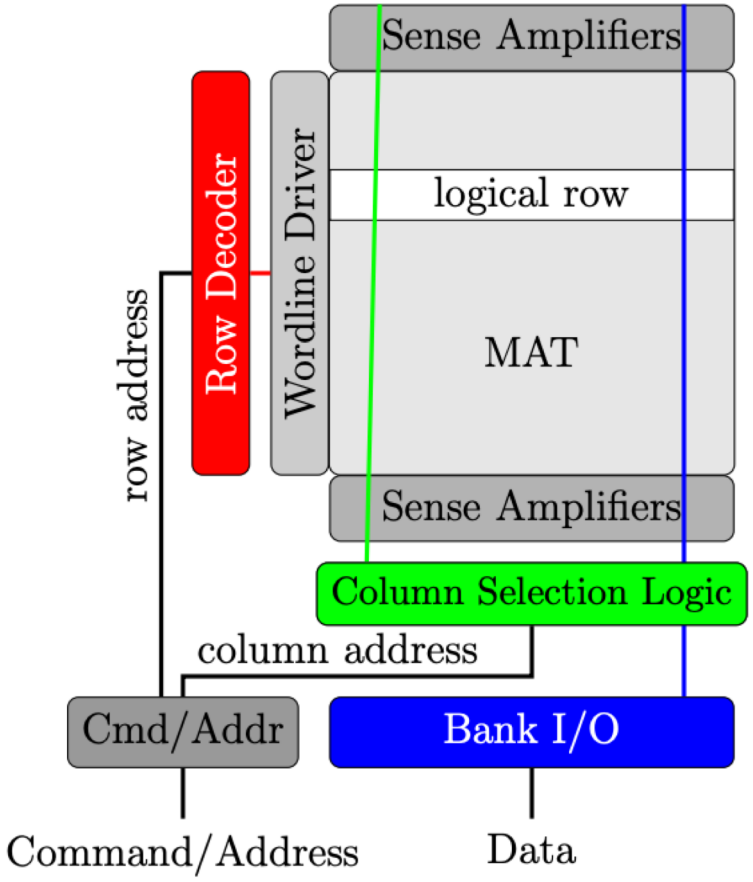
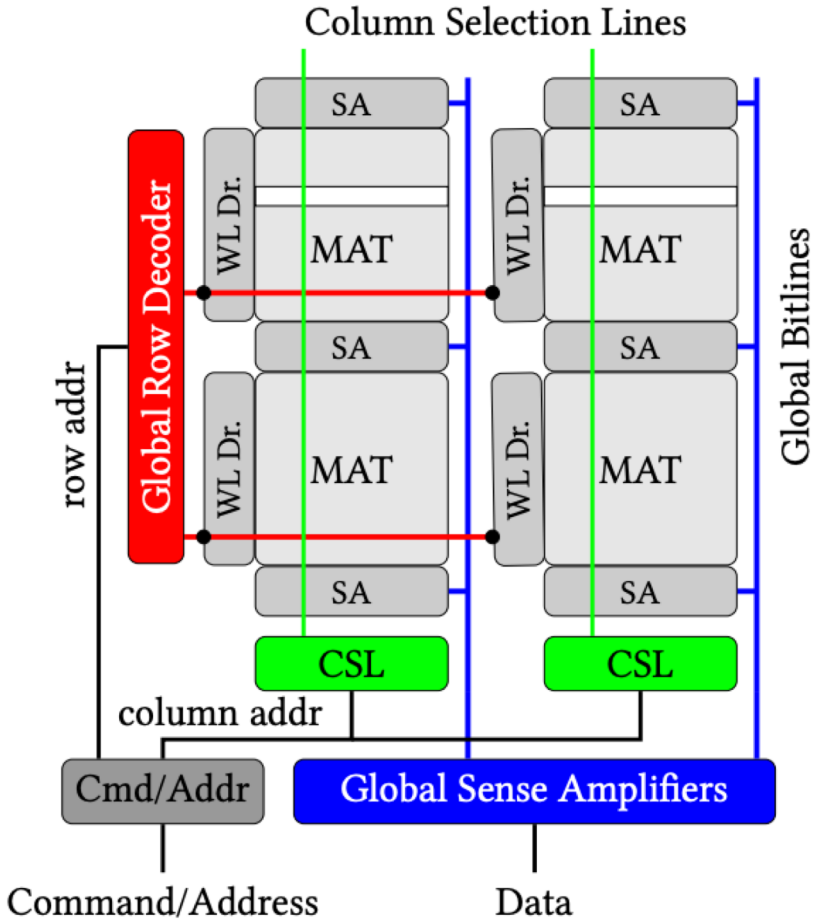


Figure 1. DRAM bank organization

Another View of a DRAM Bank



Logical Abstraction

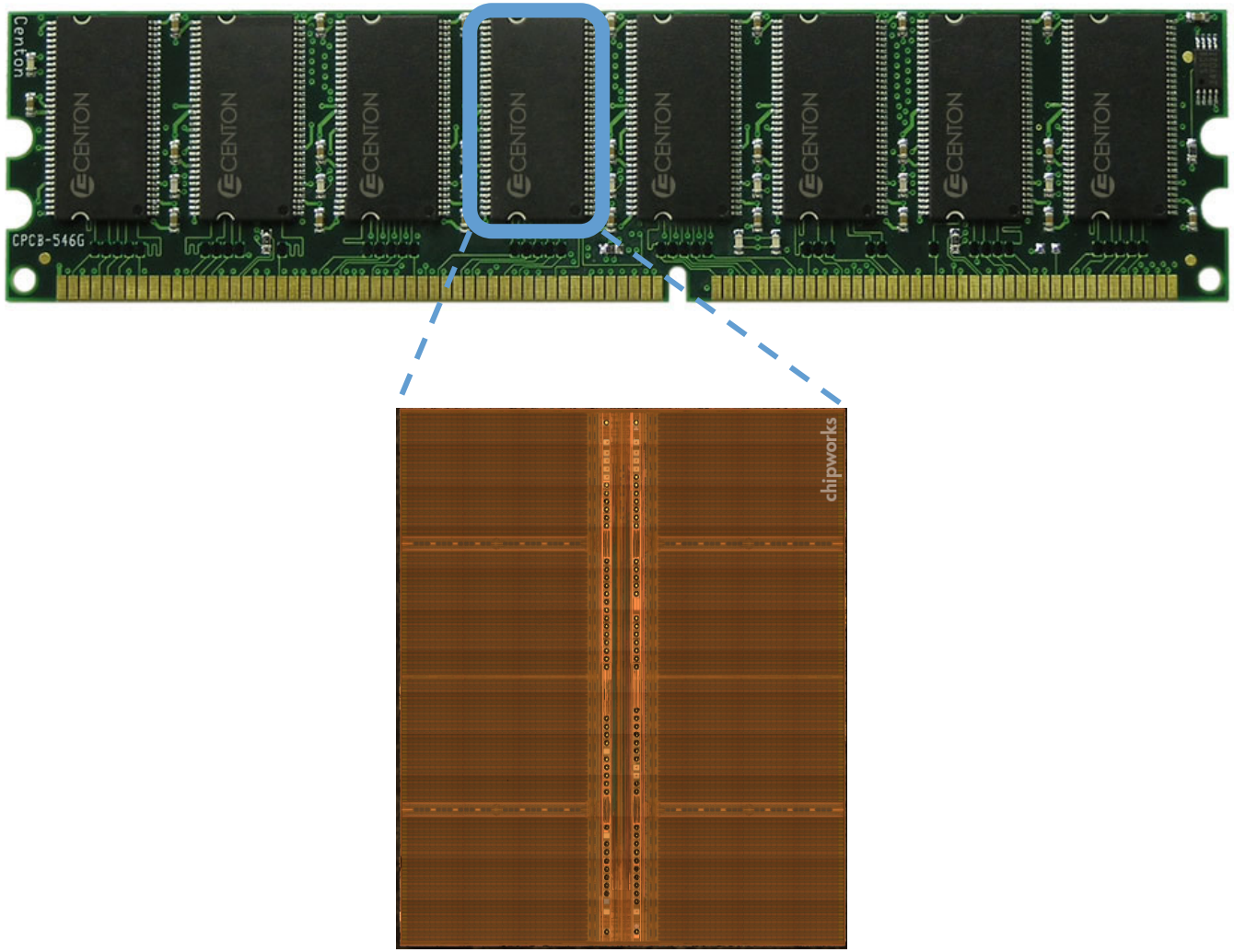


Physical View

Hardware Design

Quick View
Inside A DRAM Chip

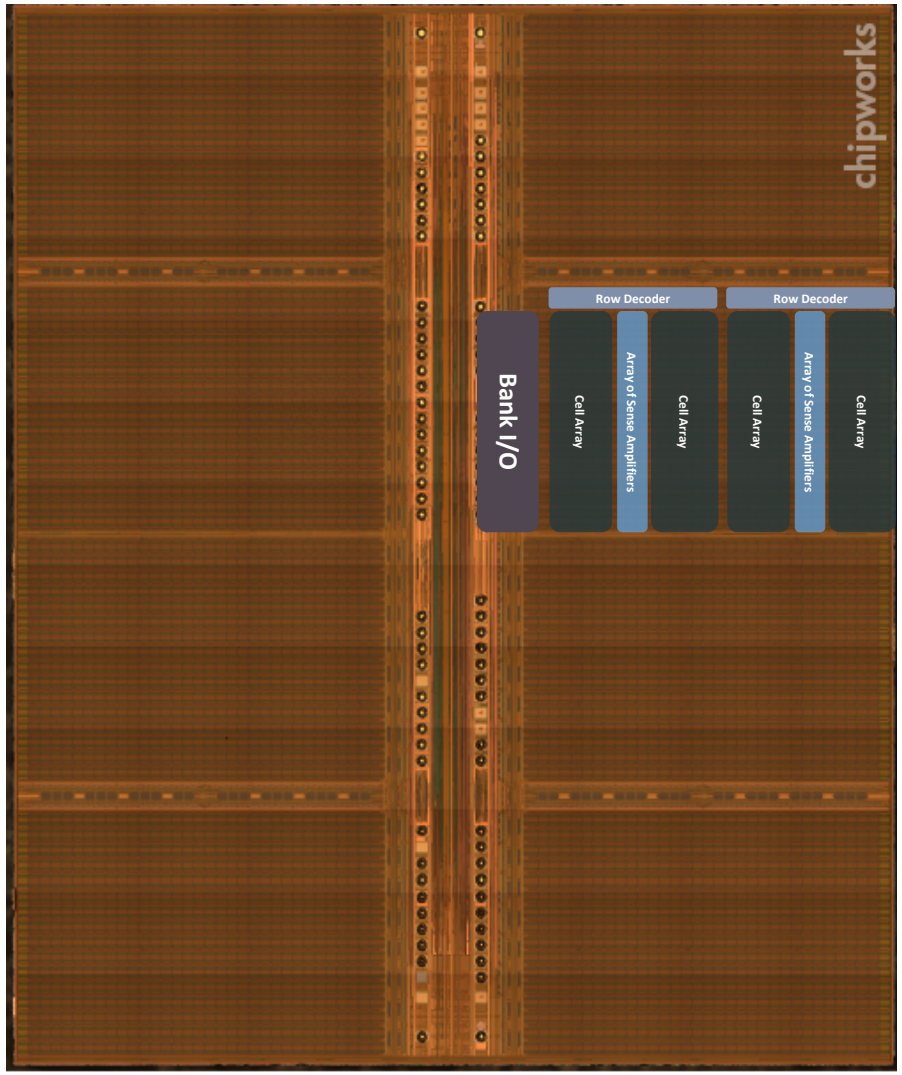
DRAM Module and Chip



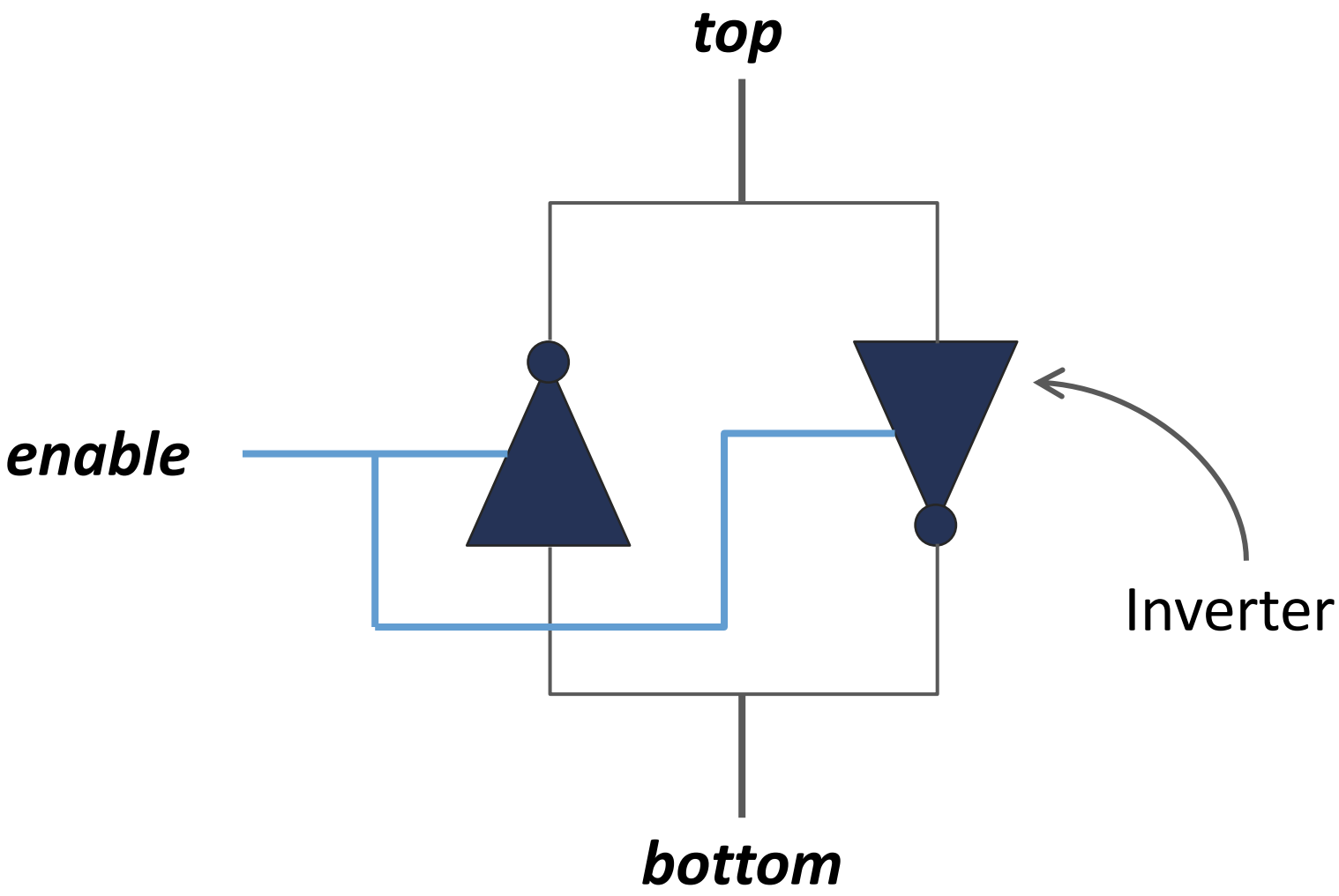
Goals in DRAM Design

- Cost
- Latency
- Bandwidth
- Parallelism
- Power
- Energy
- Reliability
- Security
- …

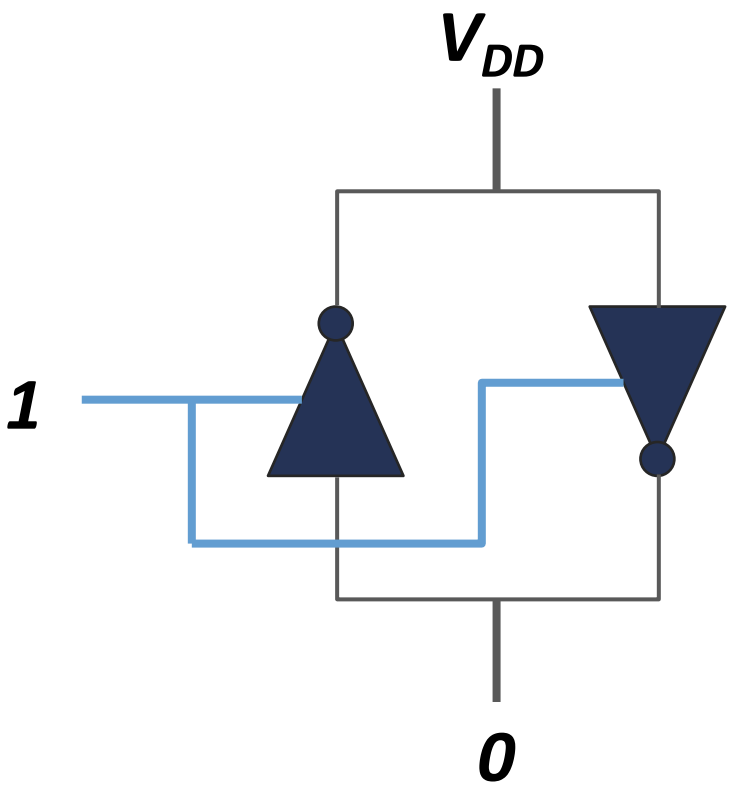
DRAM Chip


chipworks
Row Decoder
Row Decoder
Bank I/O
Cell Array
Array of Sense Amplifiers
Cell Array
Cell Array
Array of Sense Amplifiers
Cell Array

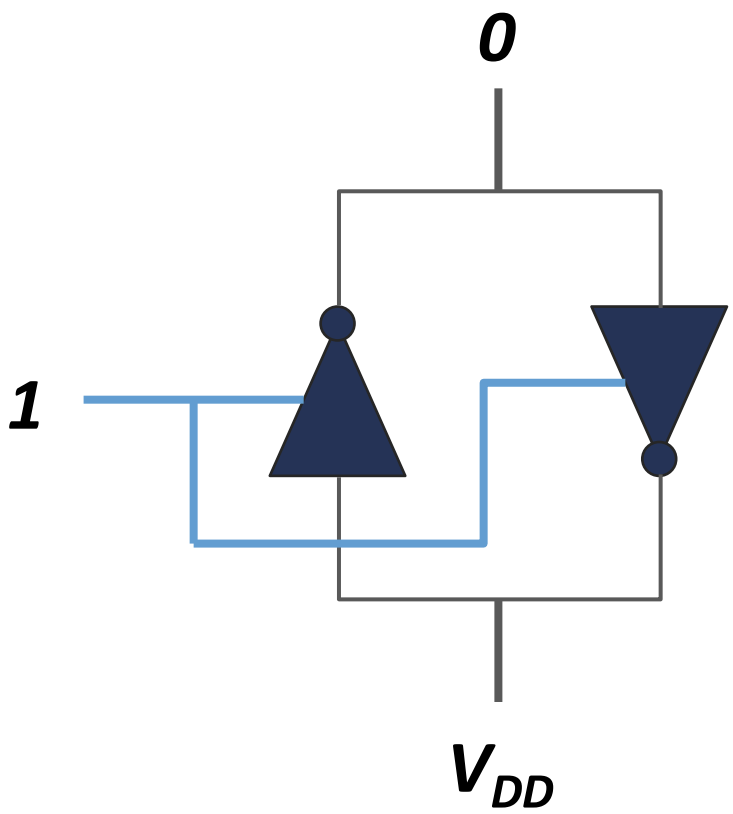
Sense Amplifier



Sense Amplifier – Two Stable States

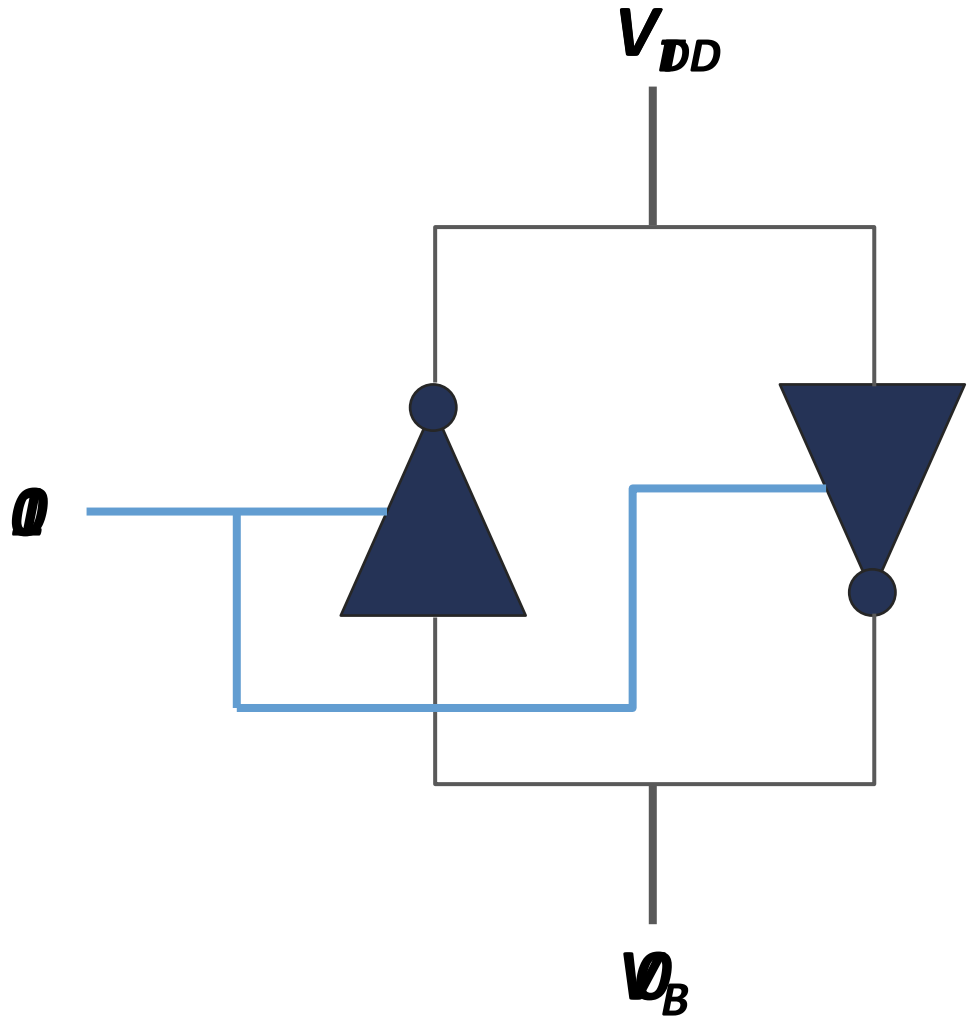


Logical “1”



Logical “0”

Sense Amplifier Operation

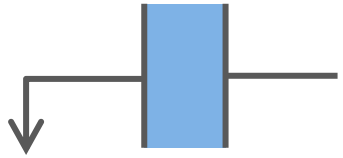


$V_T > V_B$

DRAM Cell – Capacitor

Empty State

Logical “0”

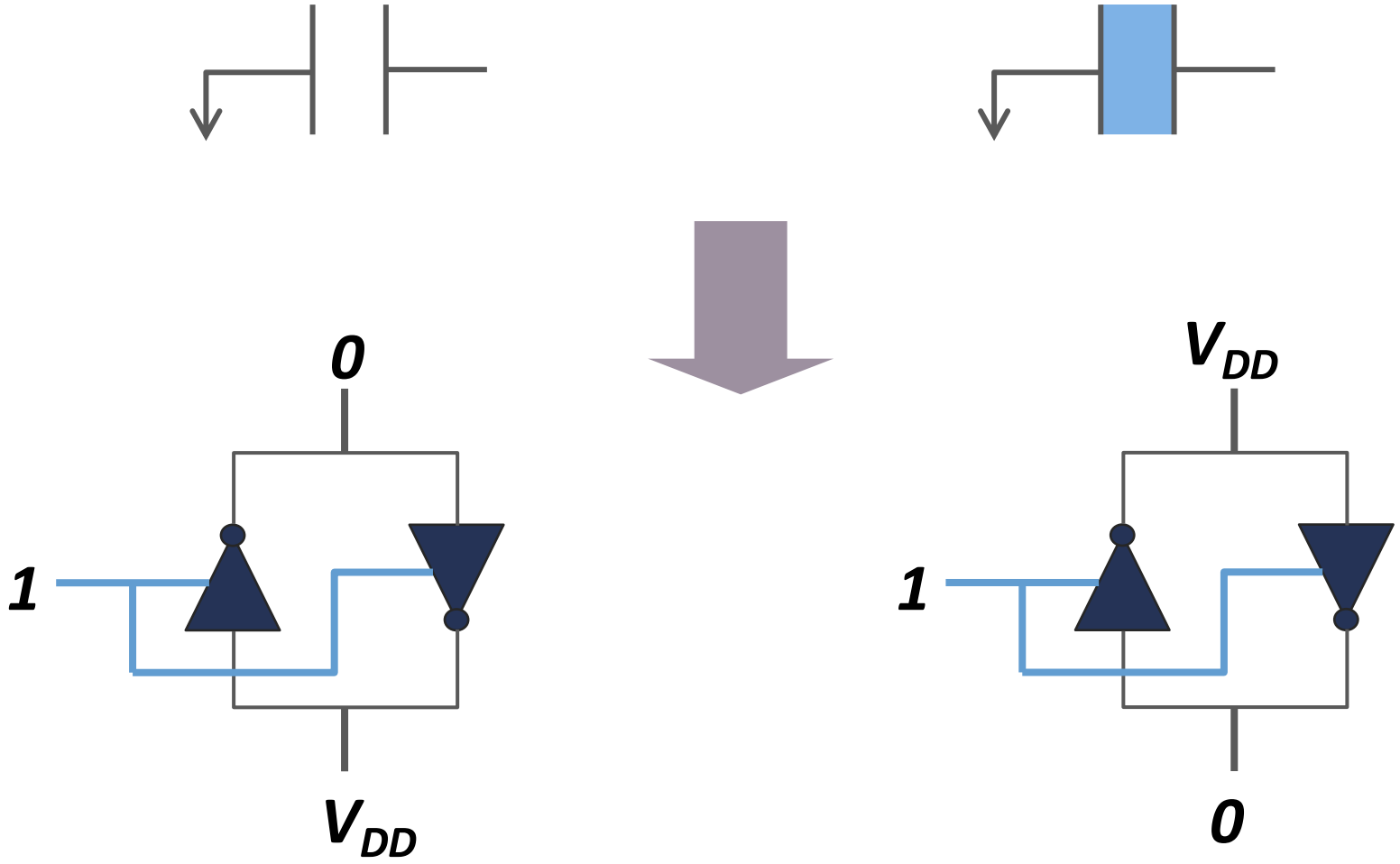


Fully Charged State

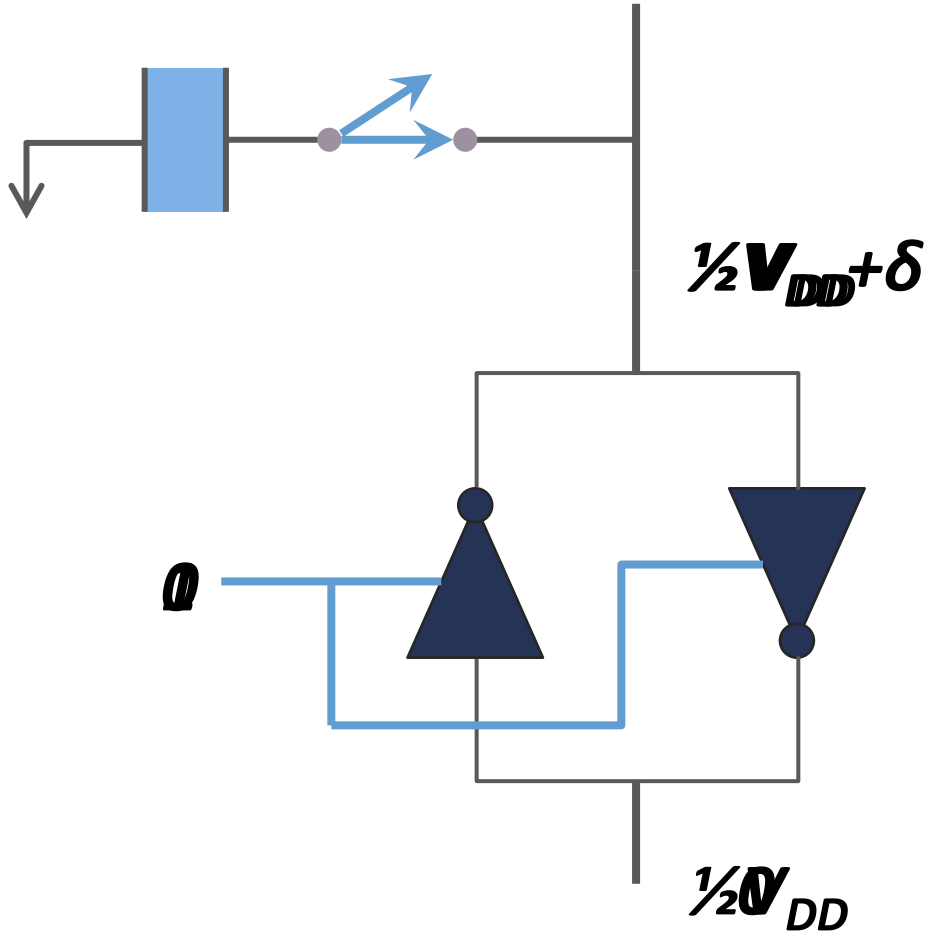
Logical “1”

1. Small – Cannot drive circuits
2. Reading destroys the state

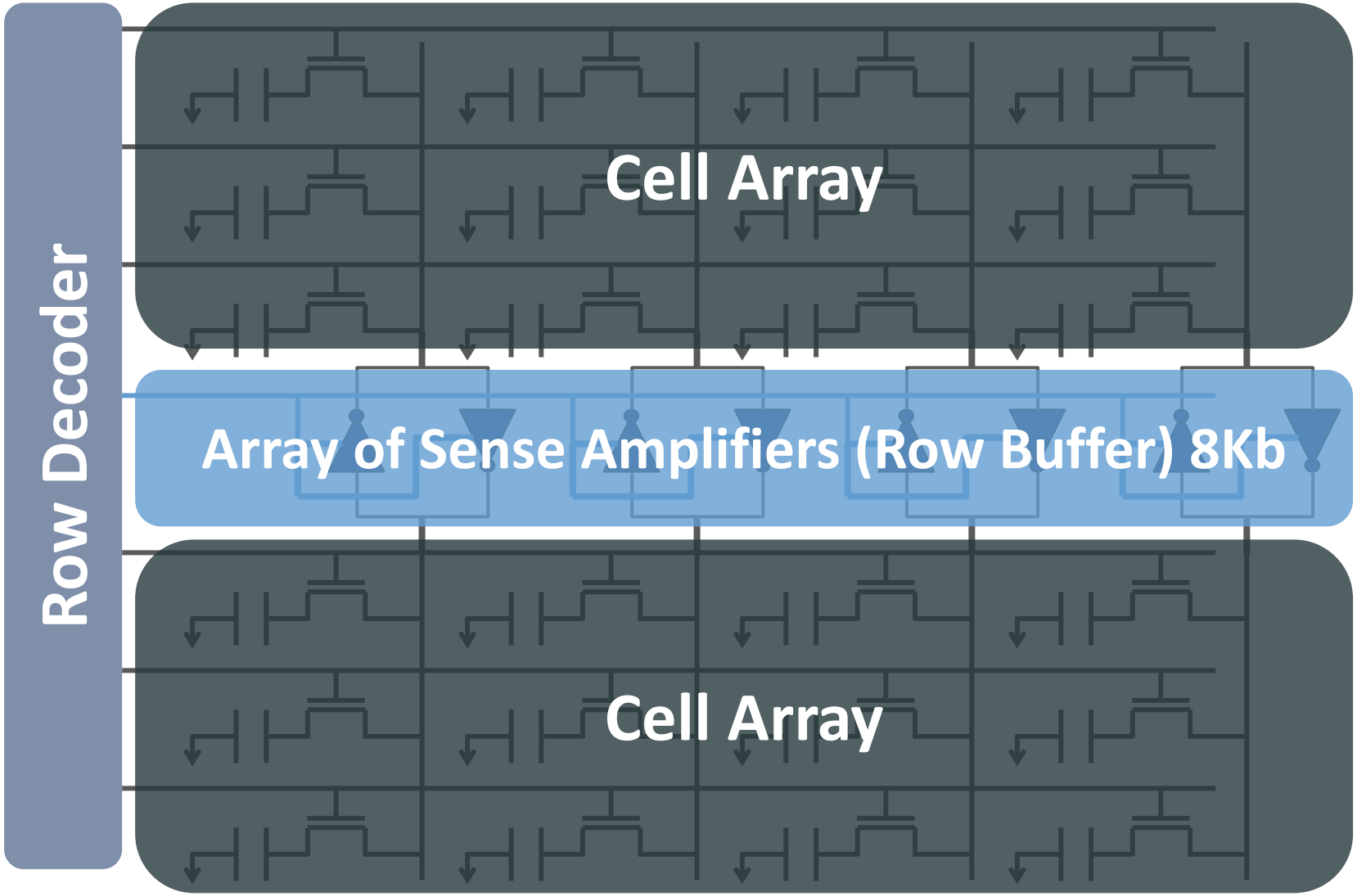
Capacitor to Sense Amplifier



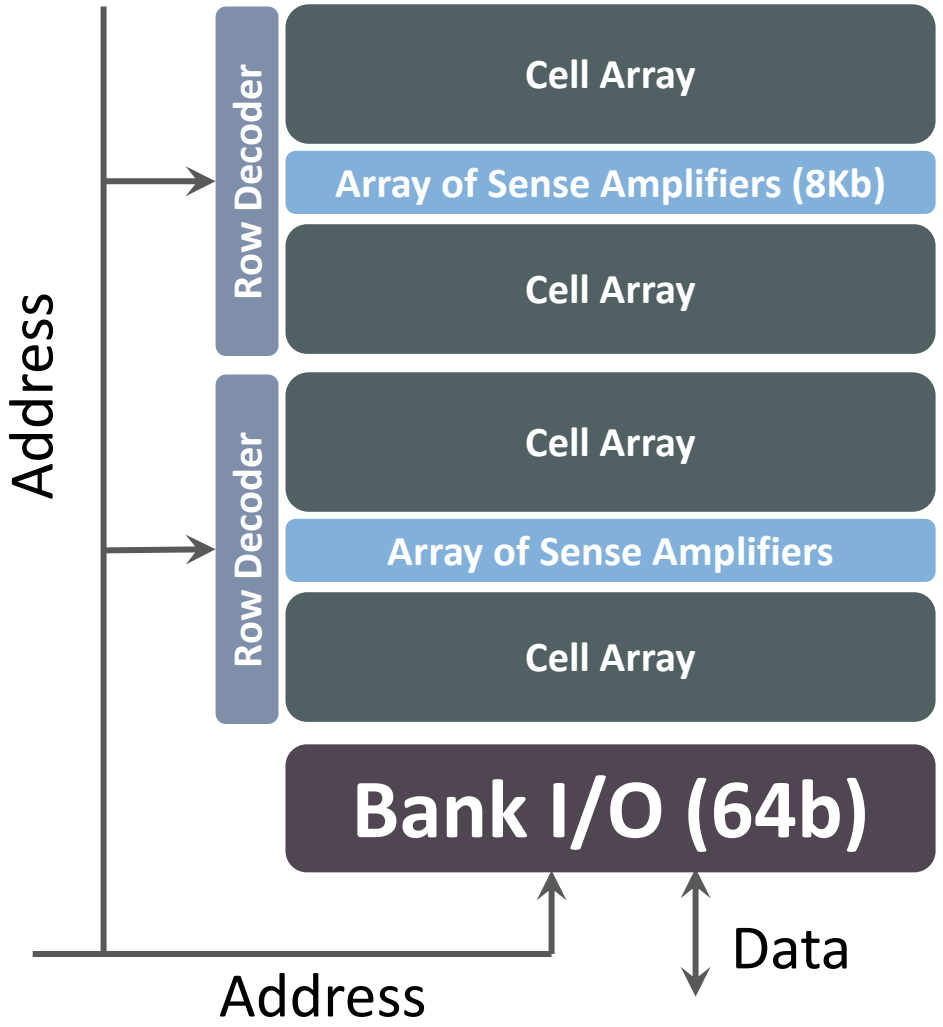
DRAM Cell Operation



DRAM Subarray – Building Block for DRAM Chip

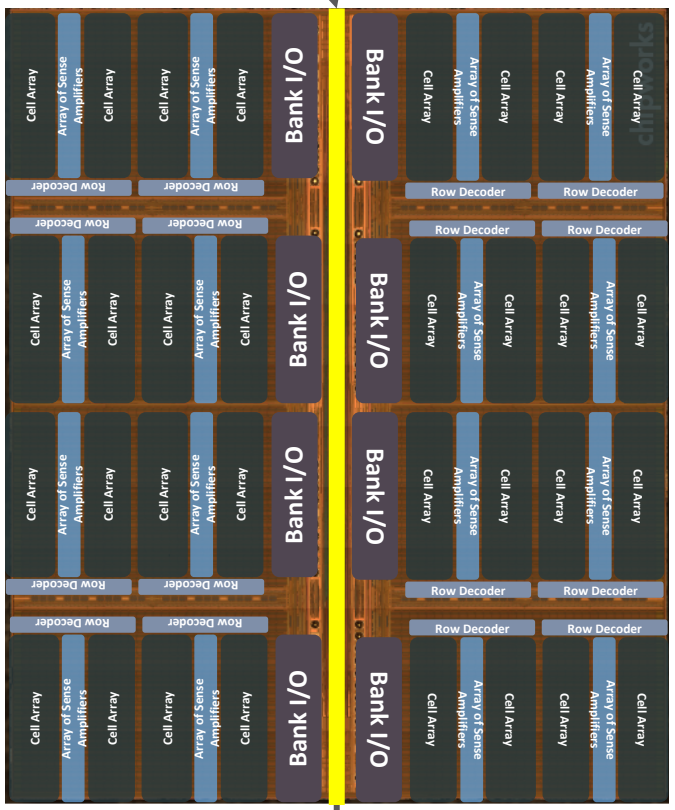


DRAM Bank


Address
Row Decoder
Cell Array
Array of Sense Amplifiers (8Kb)
Cell Array
Row Decoder
Cell Array
Array of Sense Amplifiers
Cell Array
Bank I/O (64b)
Address
Data

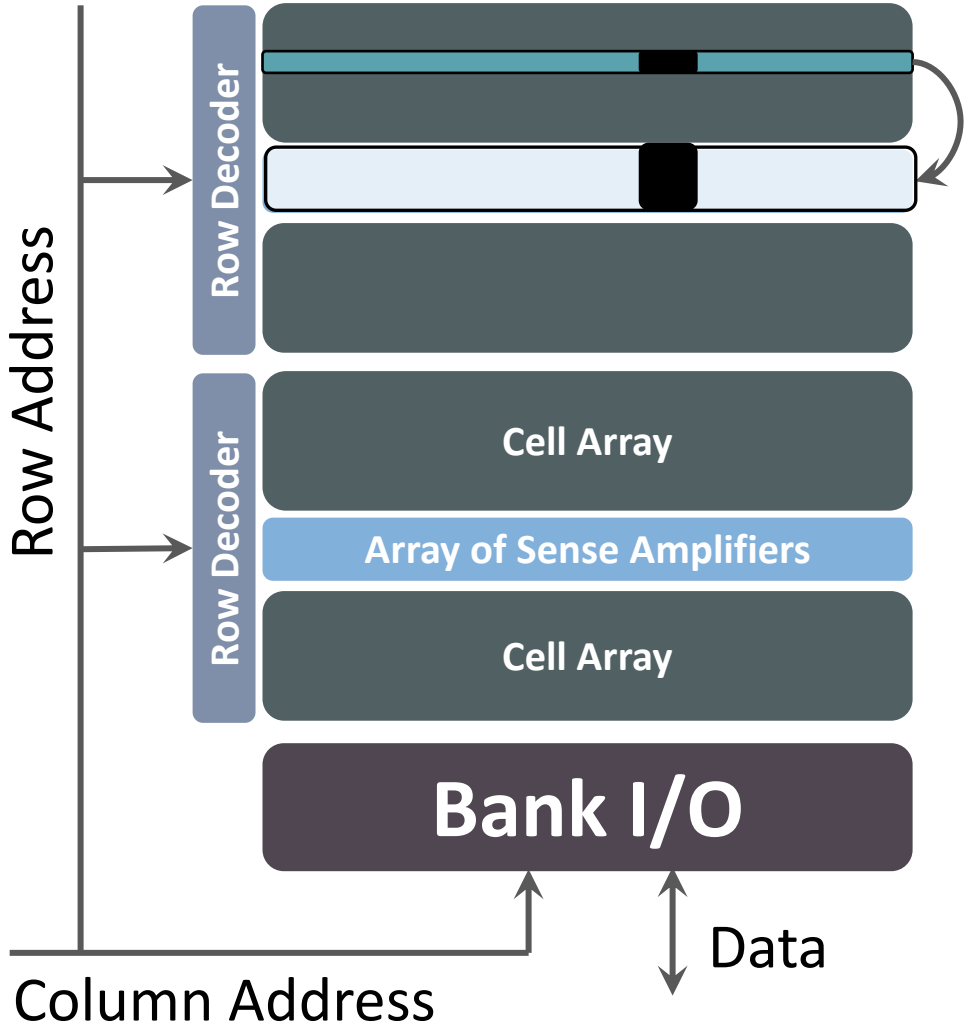
DRAM Chip

Shared internal bus



Memory channel - 8bits

DRAM Operation

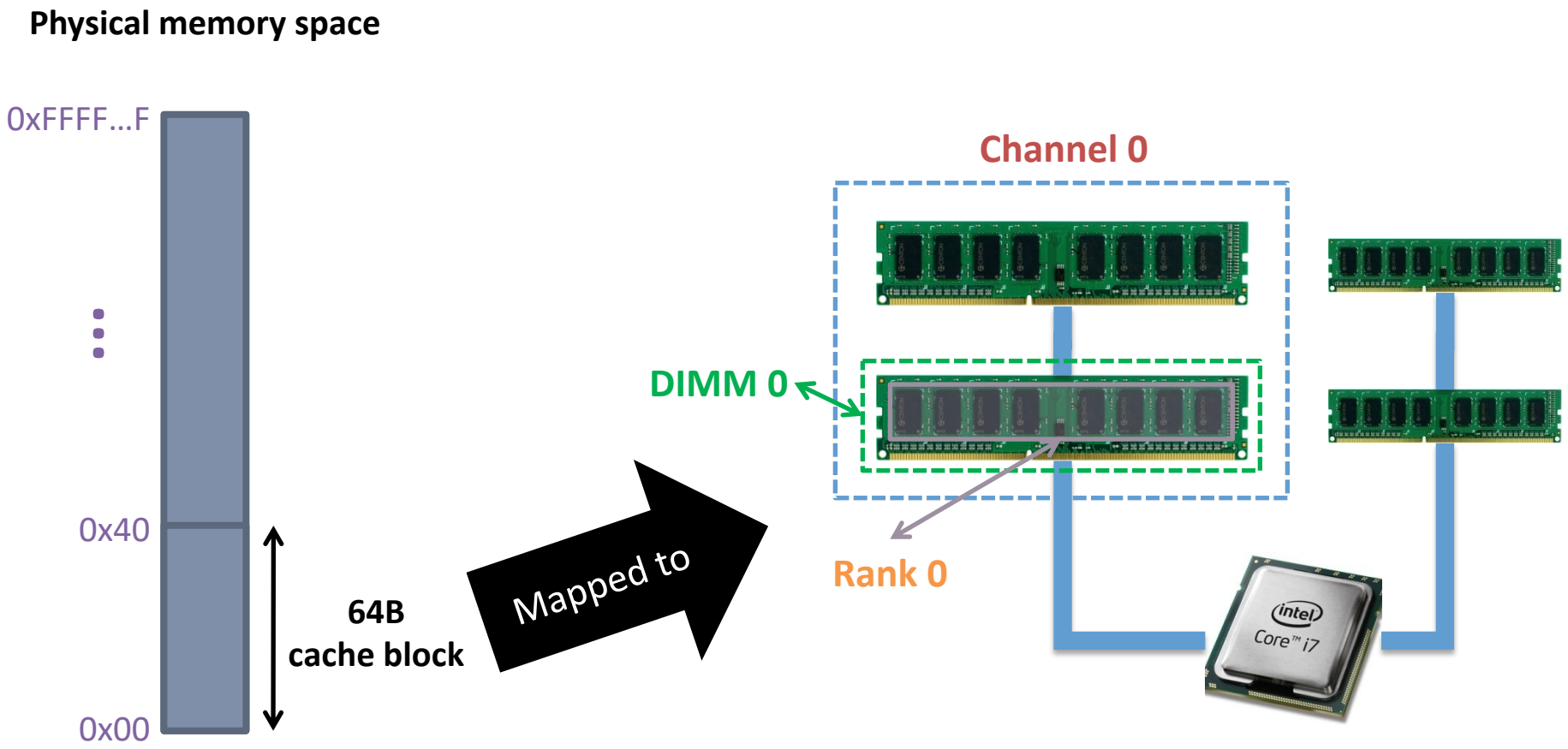


1 ACTIVATE Row

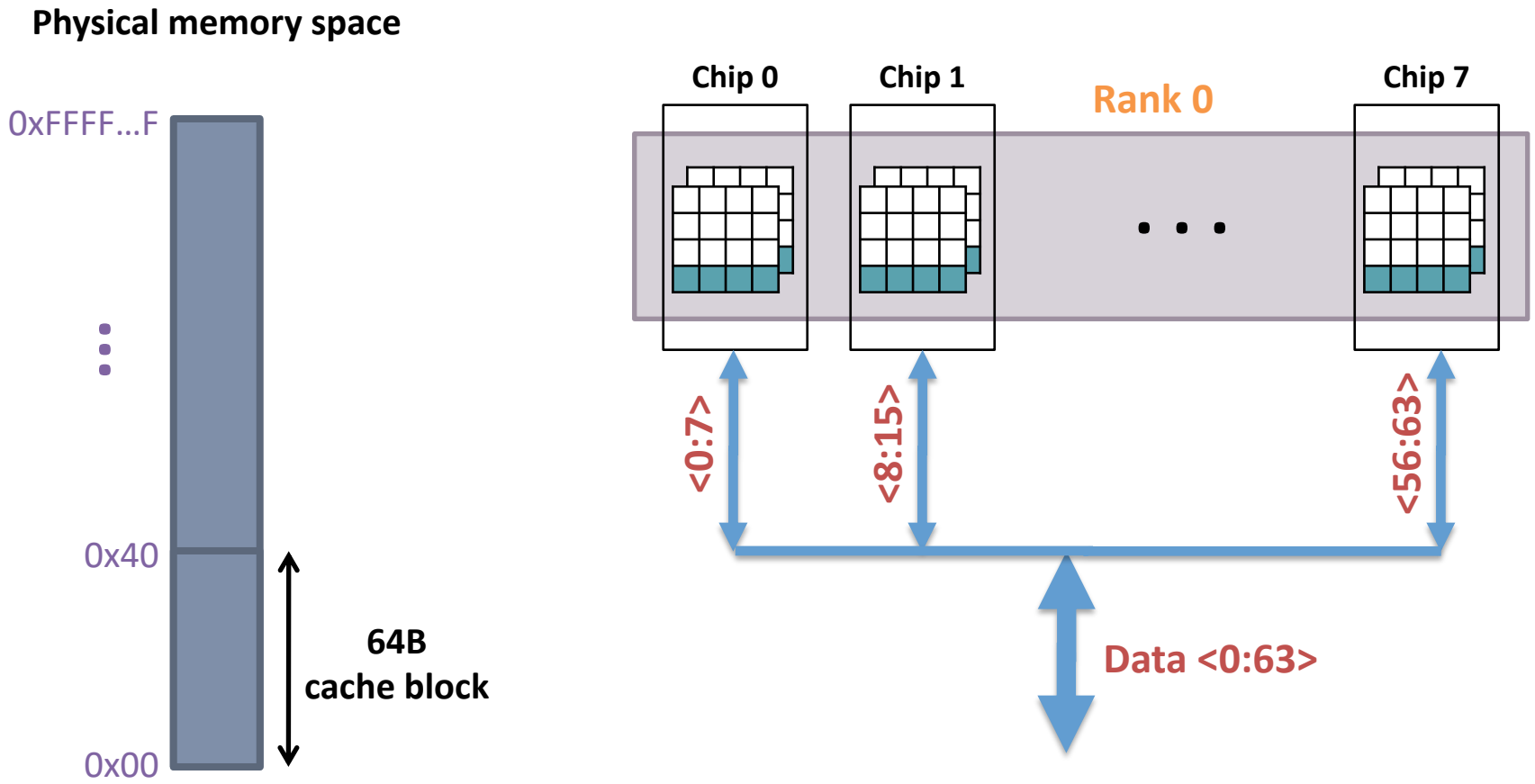
2 READ/WRITE Column

3 PRECHARGE

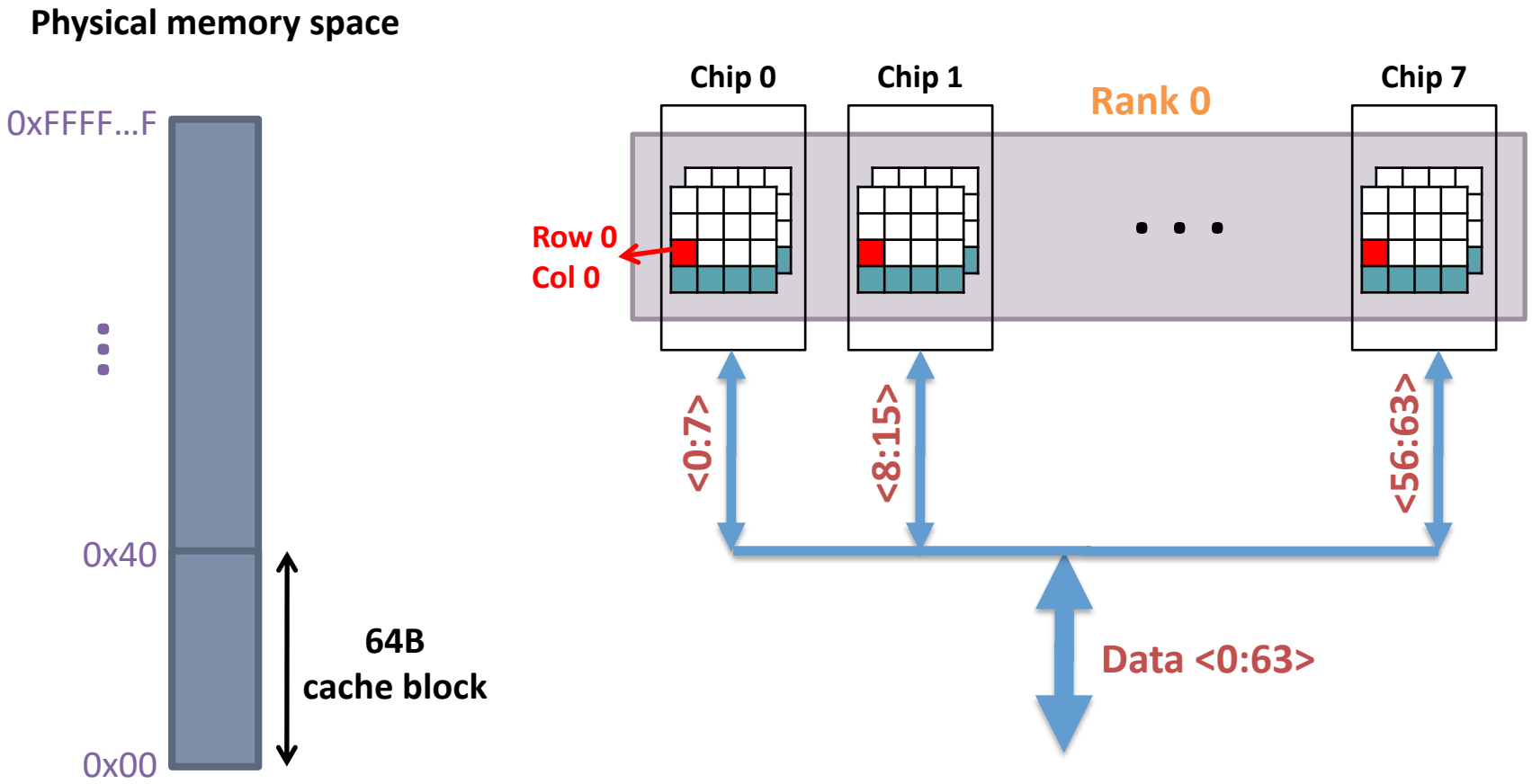
Example: Transferring a cache block



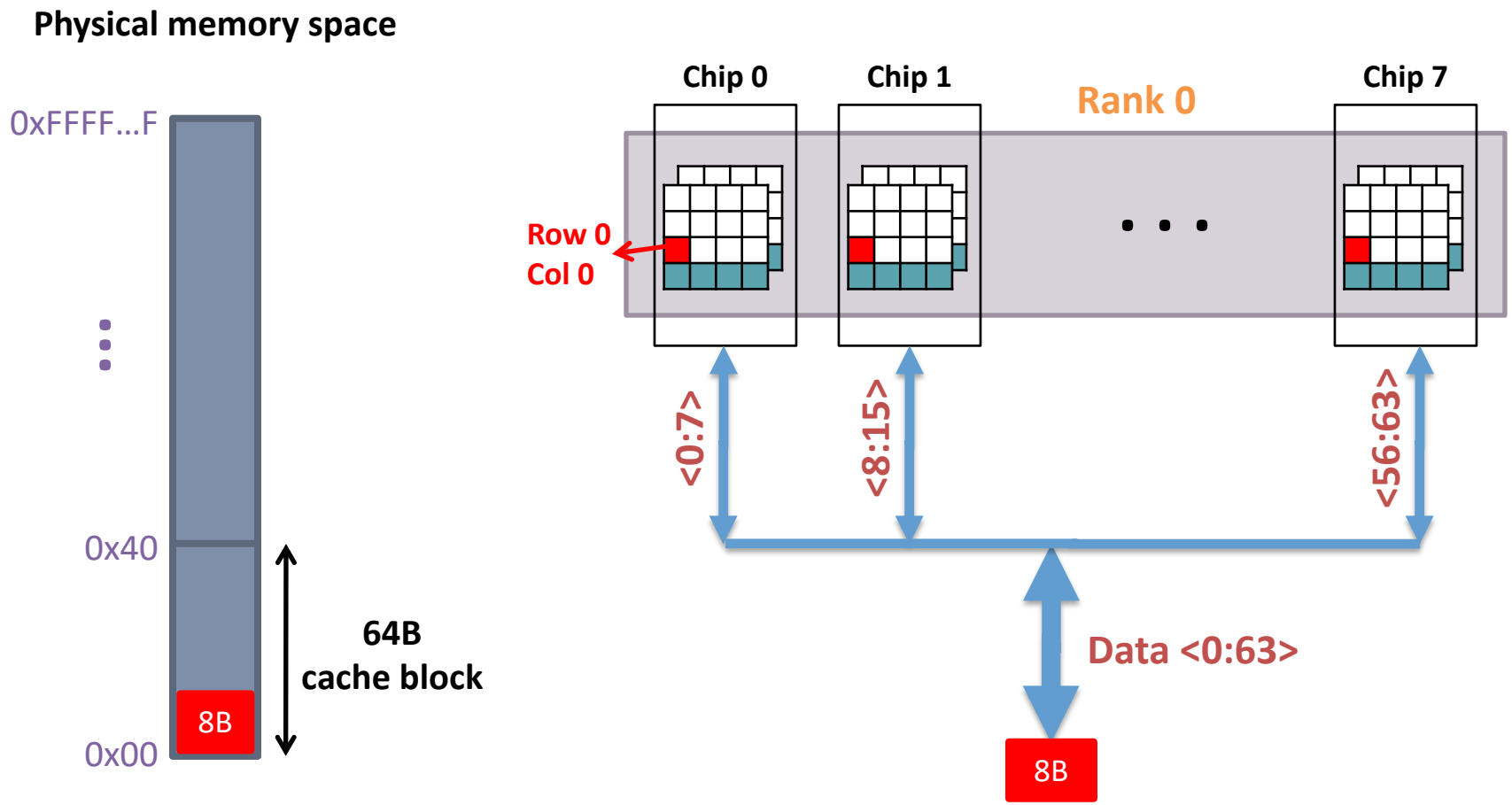
Example: Transferring a cache block



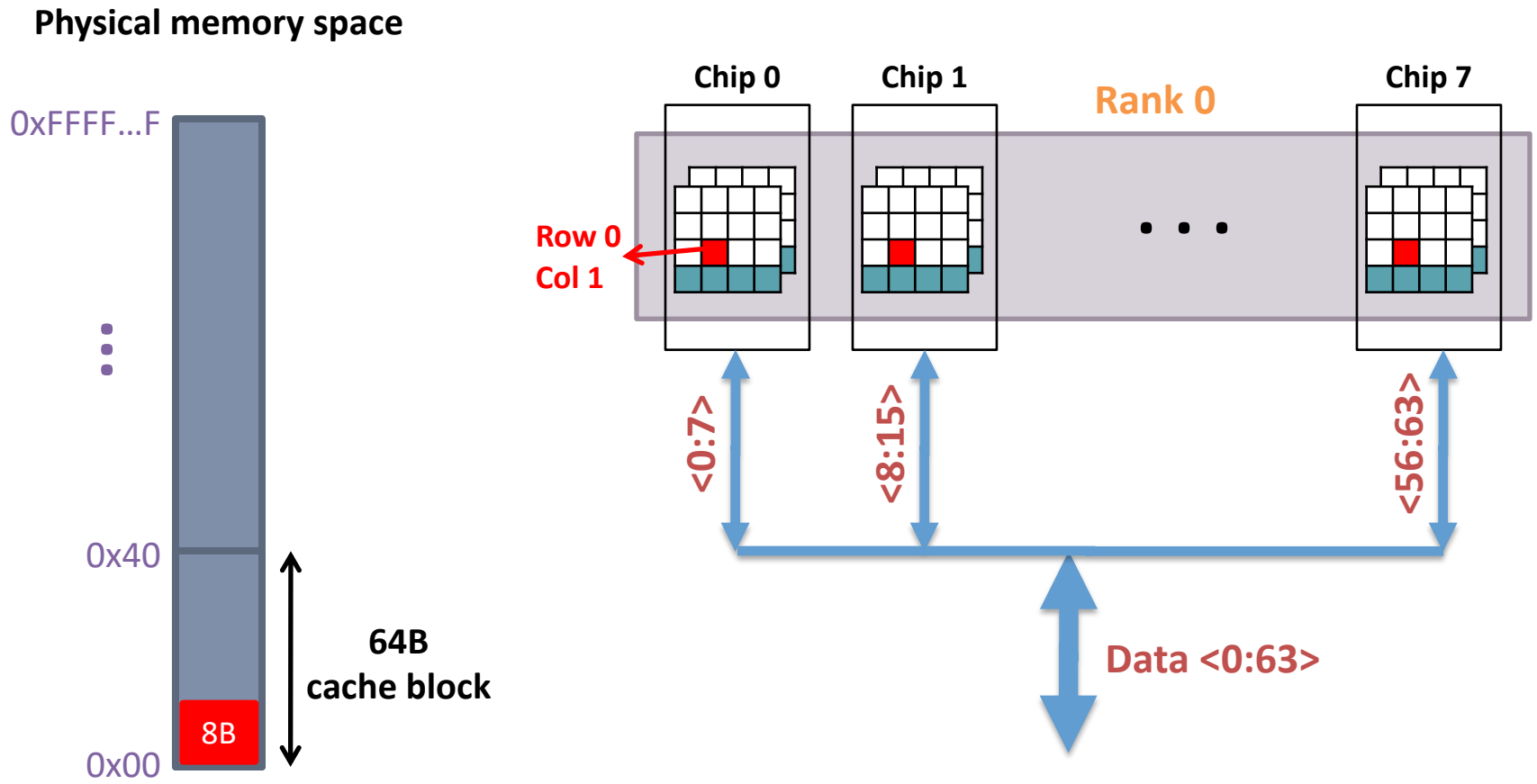
Example: Transferring a cache block



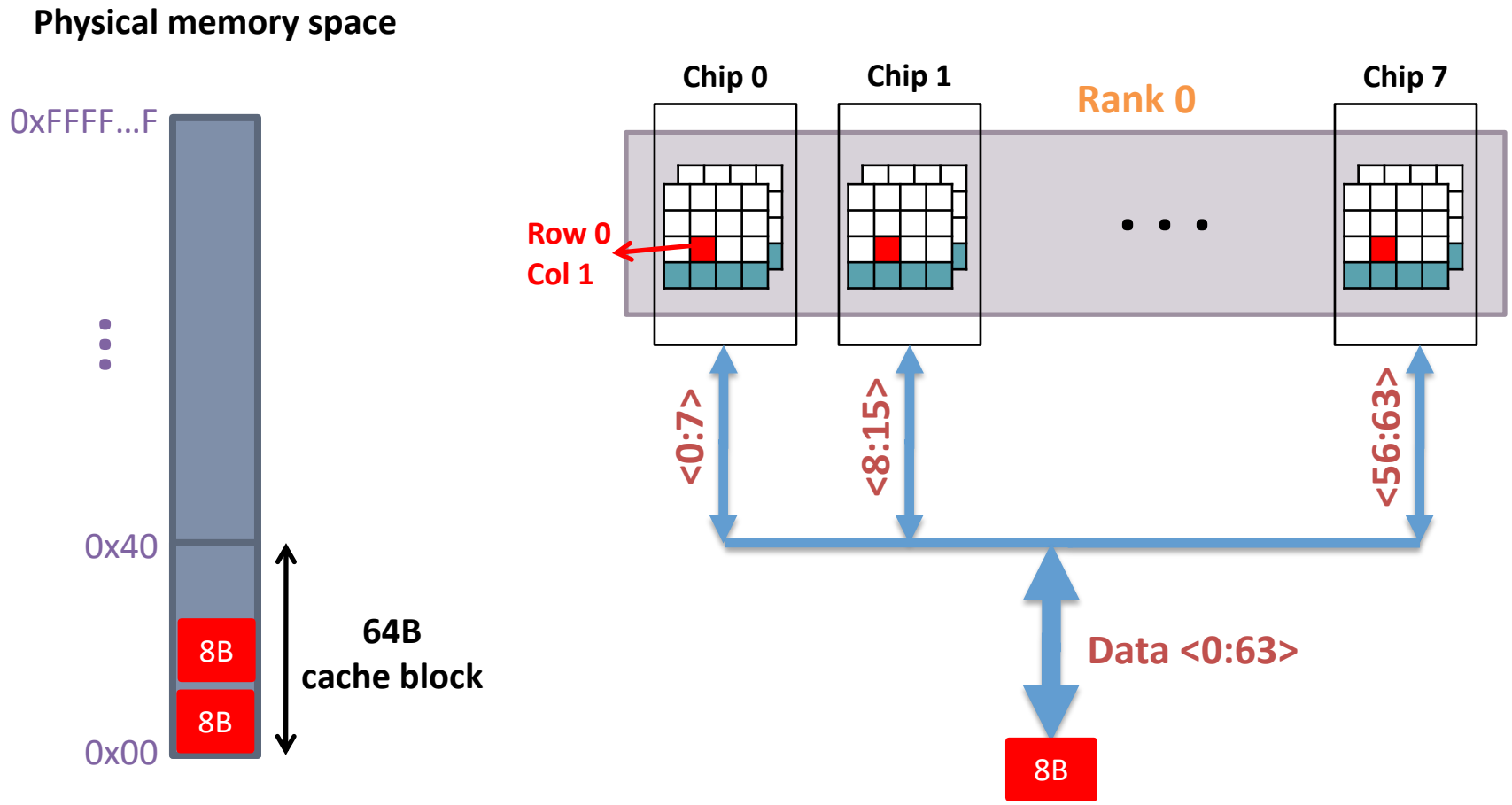
Example: Transferring a cache block



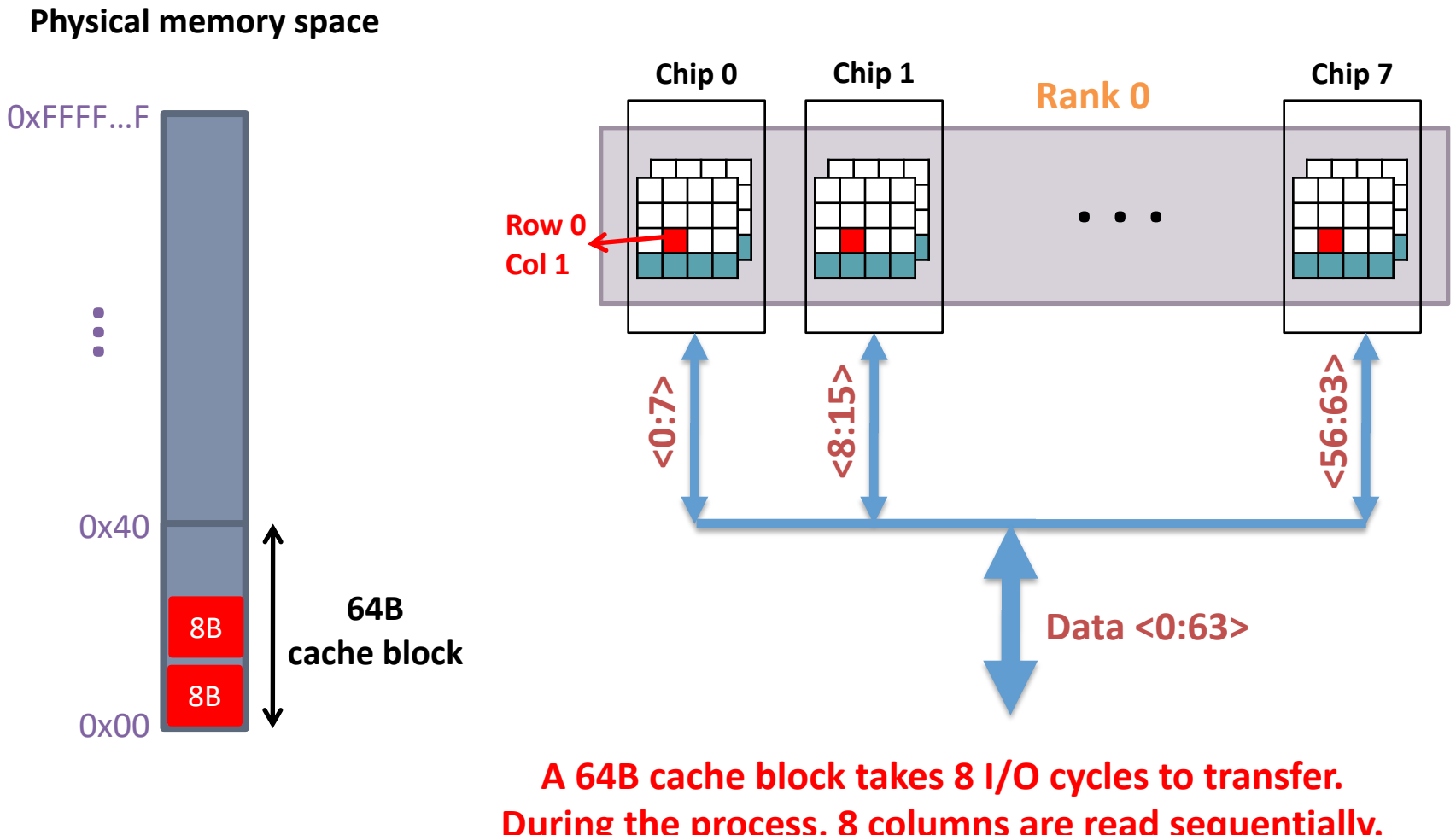
Example: Transferring a cache block



Example: Transferring a cache block



Example: Transferring a cache block



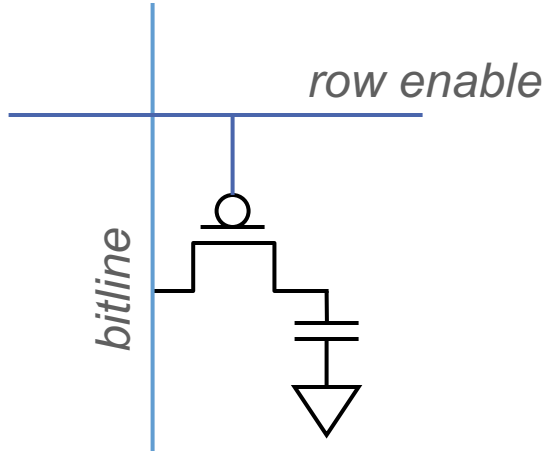
A 64B cache block takes 8 I/O cycles to transfer.
During the process, 8 columns are read sequentially.

Hardware Design

Memory Technology: DRAM and SRAM

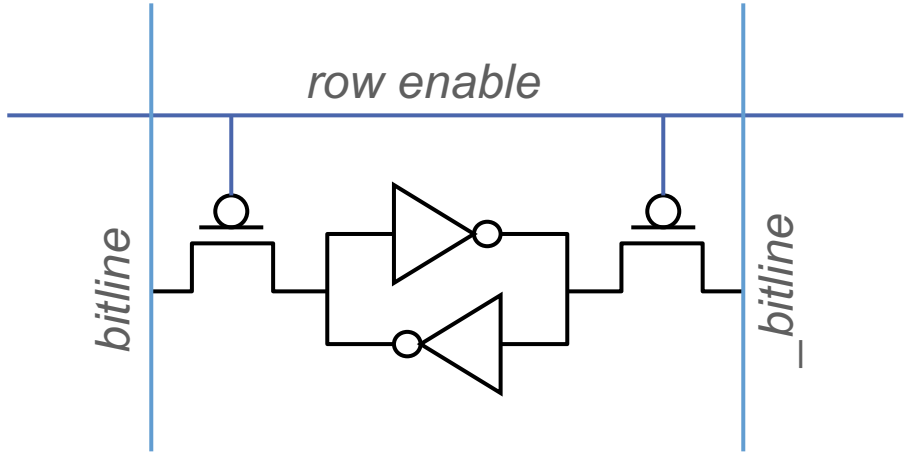
Memory Technology: DRAM

- 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 through the RC path
  - DRAM cell loses charge over time
  - DRAM cell needs to be refreshed

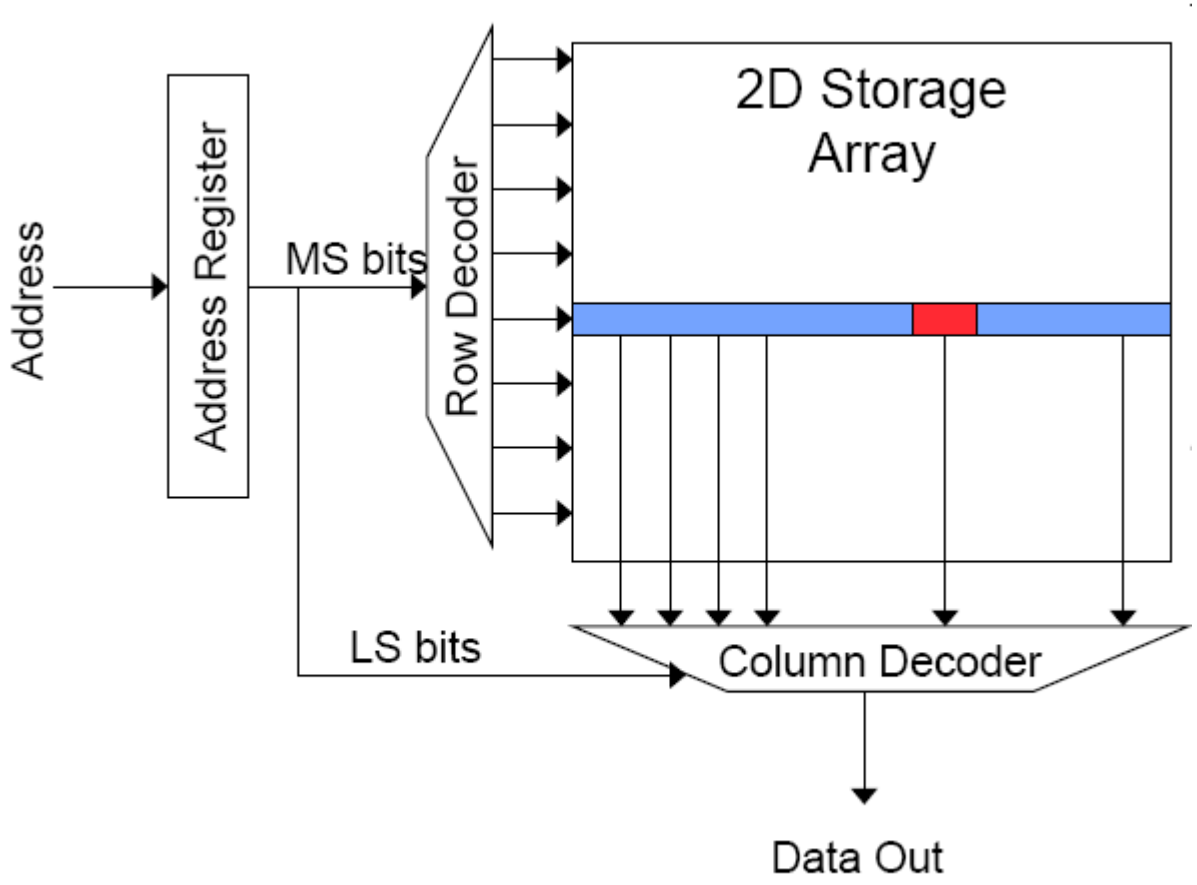


Memory Technology: SRAM

- Static random access memory
- Two cross coupled inverters store a single bit
  - Feedback path enables the stored value to stay in the "cell" (as long as powered on)
  - 4 transistors for storage
  - 2 transistors for access



Memory Bank Organization and Operation



- Read access sequence:

1. Decode row address & drive word-lines

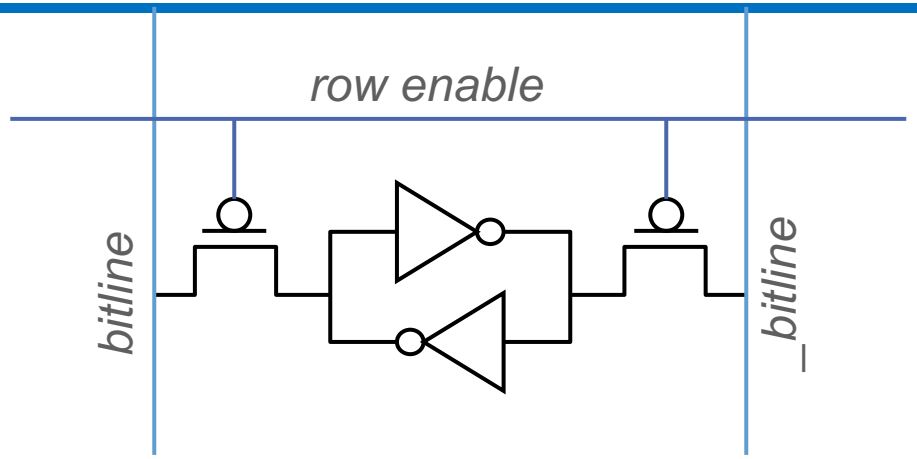
2. Selected bits drive bit-lines
   - Entire row read

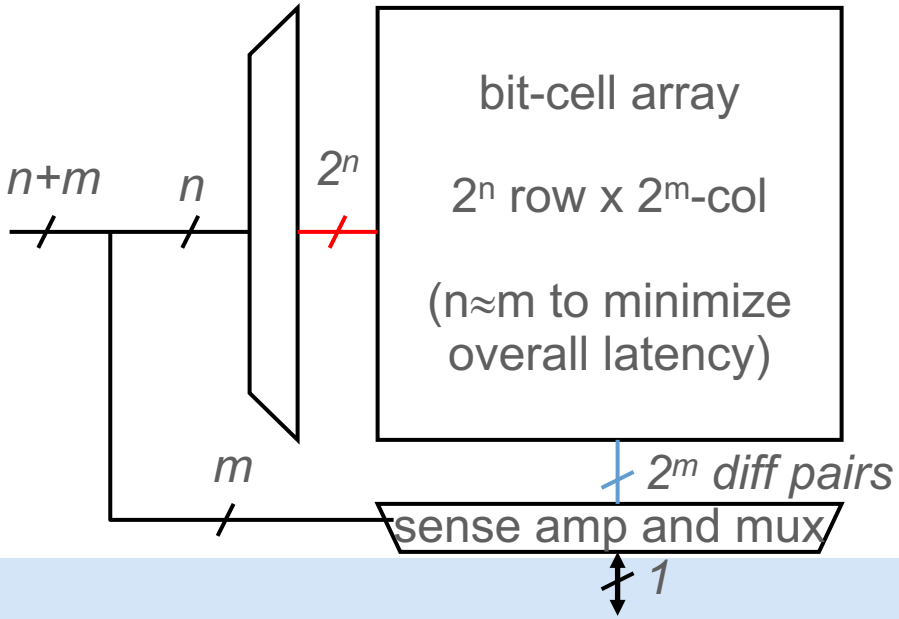
3. Amplify row data

4. Decode column address & select subset of row
   - Send to output

5. Precharge bit-lines
   - For next access

SRAM (Static Random Access Memory)





Read Sequence

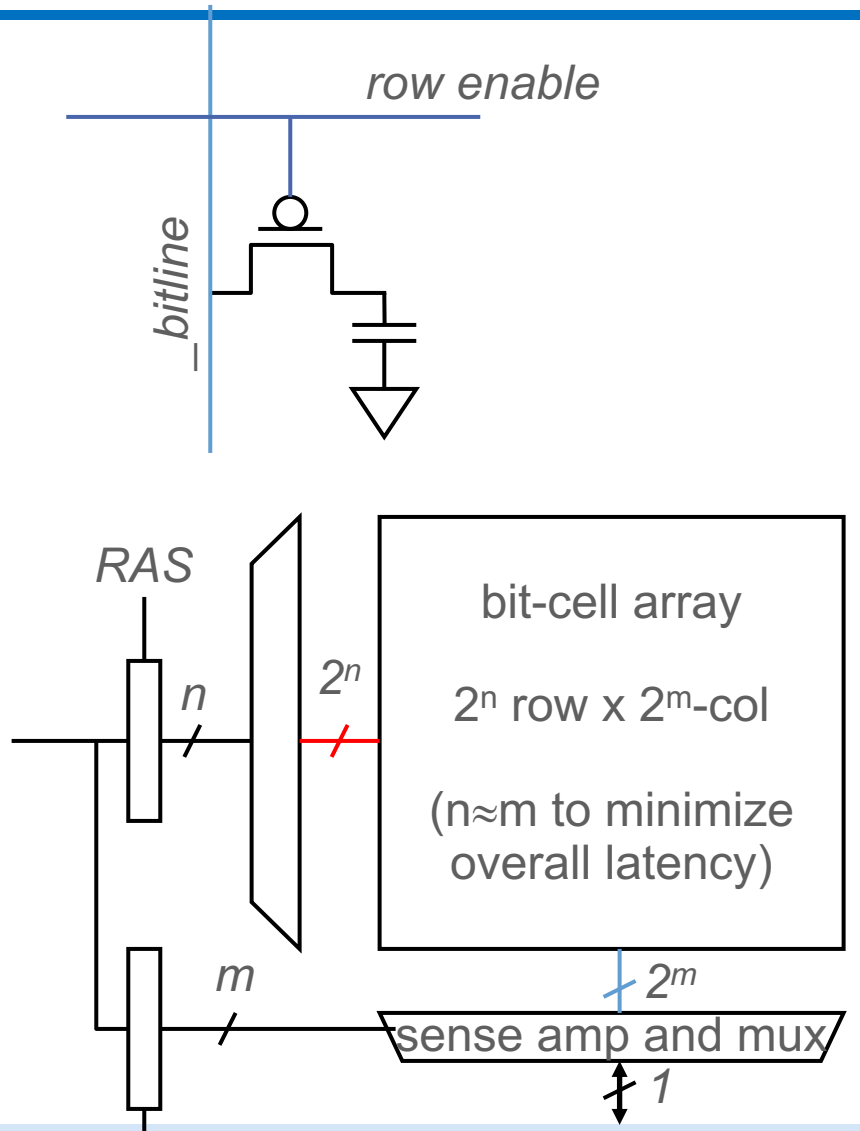
1. address decode
2. drive row enable
3. selected bit-cells drive bitlines
   (entire row is read together)
4. differential sensing and column select
   (data is ready)
5. precharge all bitlines
   (for next read or write)

Access latency dominated by steps 2 and 3

Cycling time dominated by steps 2, 3 and 5

- step 2 proportional to 2^m
- step 3 and 5 proportional to 2^n

DRAM (Dynamic Random Access Memory)



A DRAM die comprises of multiple such arrays

Bit stored as charge on node capacitor (non-restorative)

- bit cell loses charge when read
- bit cell loses charge over time

Read Sequence

1~3 same as SRAM

4. a "flip-flopping" sense amp amplifies and regenerates the bitline, data bit is mux'ed out
5. precharge all bitlines

Destructive reads

Charge loss over time

Refresh: A DRAM controller must periodically read each row within the allowed refresh time (10s of ms) such that charge is restored

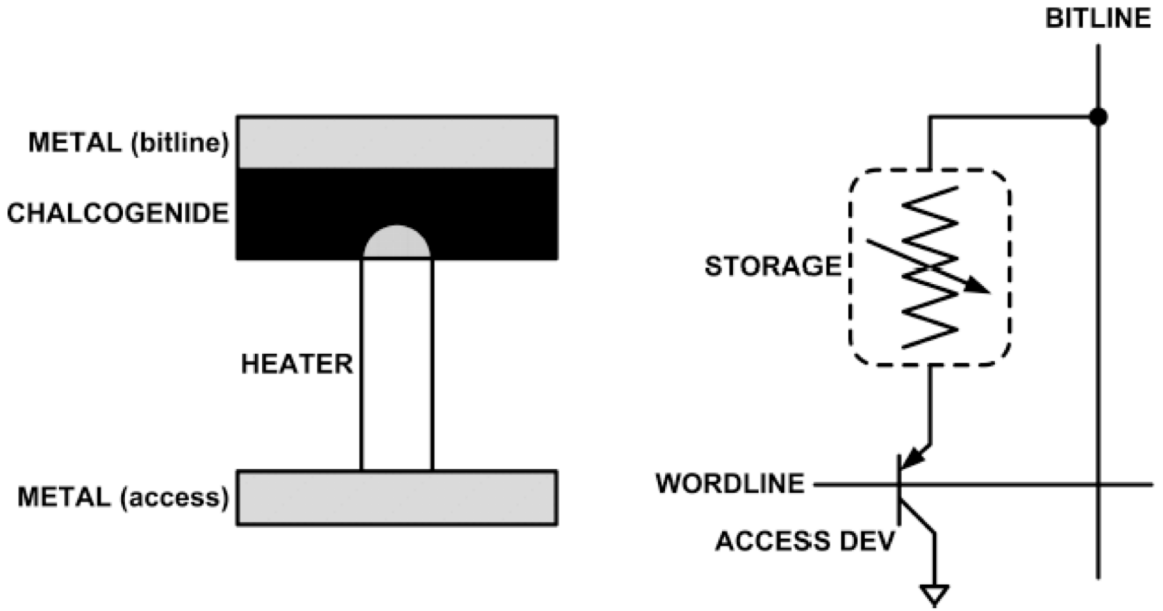
DRAM vs. SRAM

- DRAM
  - Slower access (capacitor)
  - Higher density (1T 1C cell)
  - Lower cost
  - Requires refresh (power, performance, circuitry)
  - Manufacturing requires putting the capacitor and logic together

- SRAM
  - Faster access (no capacitor)
  - Lower density (6T cell)
  - Higher cost
  - No need for refresh
  - Manufacturing compatible with logic process (no capacitor)

An Aside: Phase Change Memory

- Phase change material (chalcogenide glass) exists in two states:
  - Amorphous: Low optical reflexivity and high electrical resistivity
  - Crystalline: High optical reflexivity and low electrical resistivity



PCM is resistive memory: High resistance (0), Low resistance (1)

Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.

DRAM vs. PCM

- DRAM
  - Faster access (capacitor)
  - Lower density (capacitor less scalable) → higher cost
  - Requires refresh (power, performance, circuitry)
  - Manufacturing requires putting capacitor and logic together
  - Volatile (loses data at loss of power)
  - No endurance problems
  - Lower access energy

- PCM
  - Slower access (heating and cooling based "phase change" operation)
  - Higher density (phase change material more scalable) → lower cost
  - No need for refresh
  - Manufacturing requires less conventional processes – less mature
  - Non-volatile (does **not** lose data at loss of power)
  - Endurance problems (a cell cannot be used after N writes to it)
  - Higher access energy

Charge vs. Resistive Memories

- Charge Memory (e.g., DRAM, Flash)
  - Write data by capturing charge Q
  - Read data by detecting voltage V

- Resistive Memory (e.g., PCM, STT-MRAM, memristors)
  - Write data by pulsing current dQ/dt
  - Read data by detecting resistance R

Promising Resistive Memory Technologies

- PCM
  - Inject current to change material phase
  - Resistance determined by the phase
- STT-MRAM
  - Inject current to change magnet polarity
  - Resistance determined by polarity
- Memristors/RRAM/ReRAM
  - Inject current to change atomic structure
  - Resistance determined by the atom distance

Hardware Design

The Memory Hierarchy

The Problem

- Ideal memory's requirements oppose each other

- Bigger is slower
  - Bigger → Takes longer to determine the location

- Faster is more expensive
  - Memory technology: SRAM vs. DRAM vs. SSD vs. Disk vs. Tape

- Higher bandwidth is more expensive
  - Need more banks, more ports, more channels, higher frequency or faster technology

The Problem

- Bigger is slower
  - SRAM, < 1KByte, sub-nanosec
  - SRAM, KByte~MByte, ~nanosec
  - DRAM, Gigabyte, ~50 nanosec
  - PCM-DIMM (Intel Optane DC DIMM), Gigabyte, ~300 nanosec
  - PCM-SSD (Intel Optane SSD), Gigabyte ~Terabyte, ~6-10 µs
  - Flash memory, Gigabyte~Terabyte, ~50-100 µs
  - Hard Disk, Terabyte, ~10 millisec
- Faster is more expensive (monetary cost and chip area)
  - SRAM, < 0.3$ per Megabyte
  - DRAM, < 0.006$ per Megabyte
  - PCM-DIMM (Intel Optane DC DIMM), < 0.004$ per Megabyte
  - PCM-SSD, < 0.002$ per Megabyte
  - Flash memory, < 0.00008$ per Megabyte
  - Hard Disk, < 0.00003$ per Megabyte
  - These sample values (circa ~2023) scale with time
- Other technologies have their place as well
  - FeRAM, MRAM, RRAM, STT-MRAM, memristors, … (not mature yet)

The Problem (Table View)

Memory Device	Capacity	Latency	Cost per Megabyte
SRAM	< 1 KByte	sub-nanosec	
SRAM	KByte~MByte	~nanosec	< 0.3$
DRAM	Gigabyte	~50 nanosec	< 0.006$
PCM-DIMM (Intel Optane DC DIMM)	Gigabyte	~300 nanosec	< 0.004$
PCM-SSD (Intel Optane SSD)	Gigabyte ~Terabyte	~6-10 µs	< 0.002$
Flash memory	Gigabyte ~Terabyte	~50-100 µs	< 0.00008$
Hard Disk	Terabyte	~10 millisec	< 0.00003$

Bigger is slower Faster is more expensive ($$$ and chip area)

These sample values (circa ~2023) scale with time

The Problem (Table View): Energy

Bigger is slower

Faster is more energy-efficient

Memory Device	Capacity	Latency	Cost per Megabyte	Energy per access	Energy per byte access
SRAM	< 1 KByte	sub-nanosec		~5 pJ	~1.25 pJ
SRAM	KByte~MByte	~nanosec	< 0.3$		
DRAM	Gigabyte	~50 nanosec	< 0.006$	~40-140 pJ	~10-35 pJ
PCM-DIMM (Intel Optane DC DIMM)	Gigabyte	~300 nanosec	< 0.004$	~80-540 pJ	~20-135 pJ
PCM-SSD (Intel Optane SSD)	Gigabyte ~Terabyte	~6-10 µs	< 0.002$	~120 µJ	~30 nJ
Flash memory	Gigabyte ~Terabyte	~50-100 µs	< 0.00008$	~250 µJ	~61 nJ
Hard Disk	Terabyte	~10 millisec	< 0.00003$	~60 mJ	~15 µJ

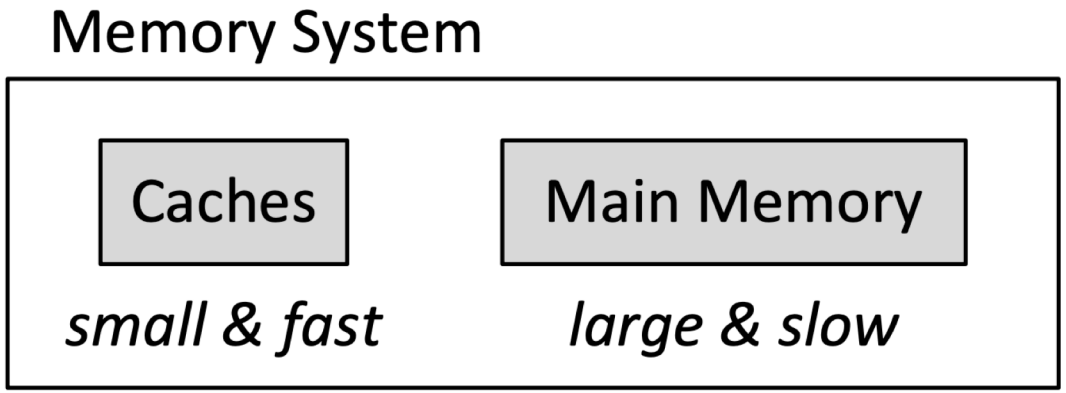
Faster is more expensive
($$$ and chip area)

These sample values (circa ~2023) scale with time

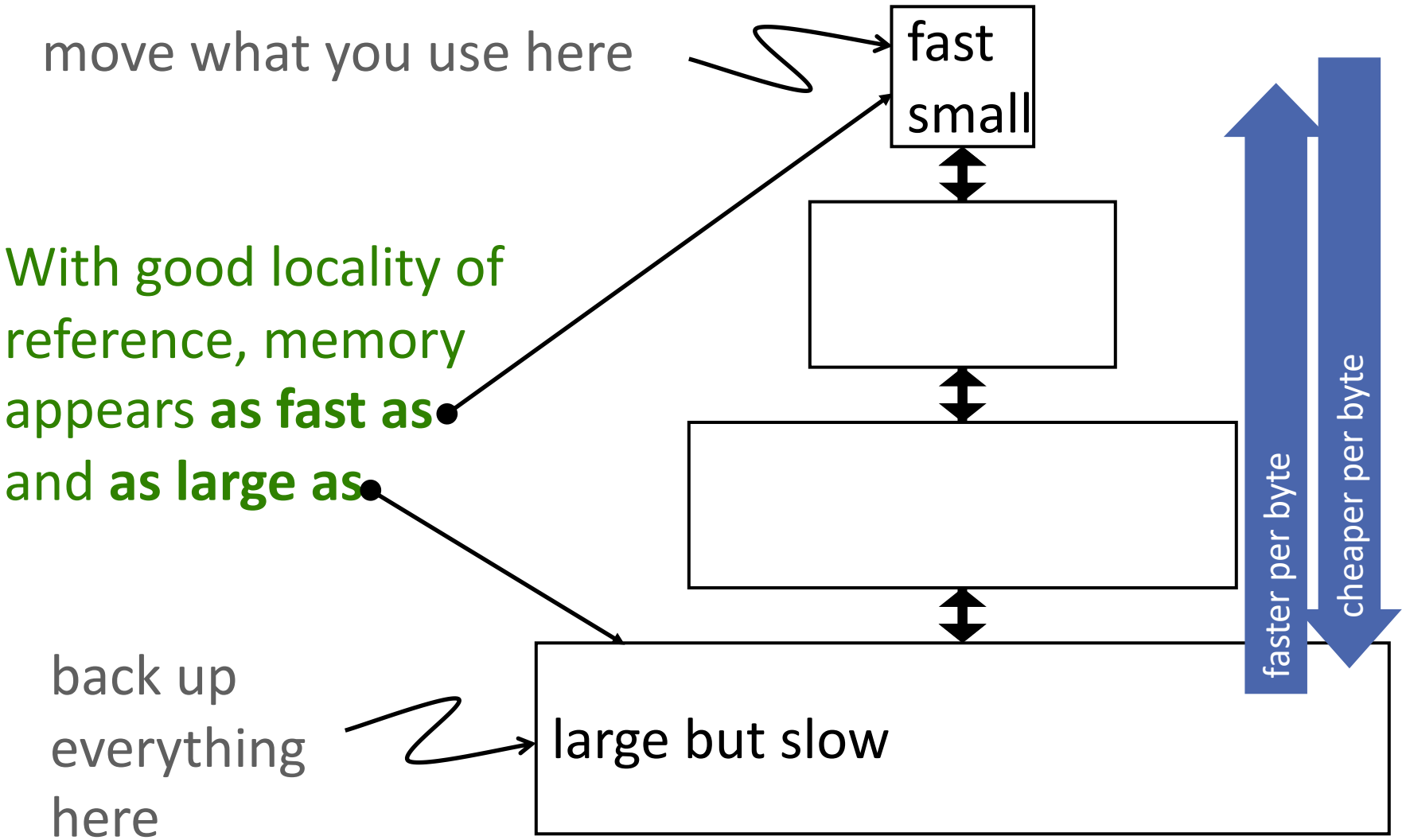
Disclaimer: Take the energy values with a grain of salt as there are different assumptions

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)

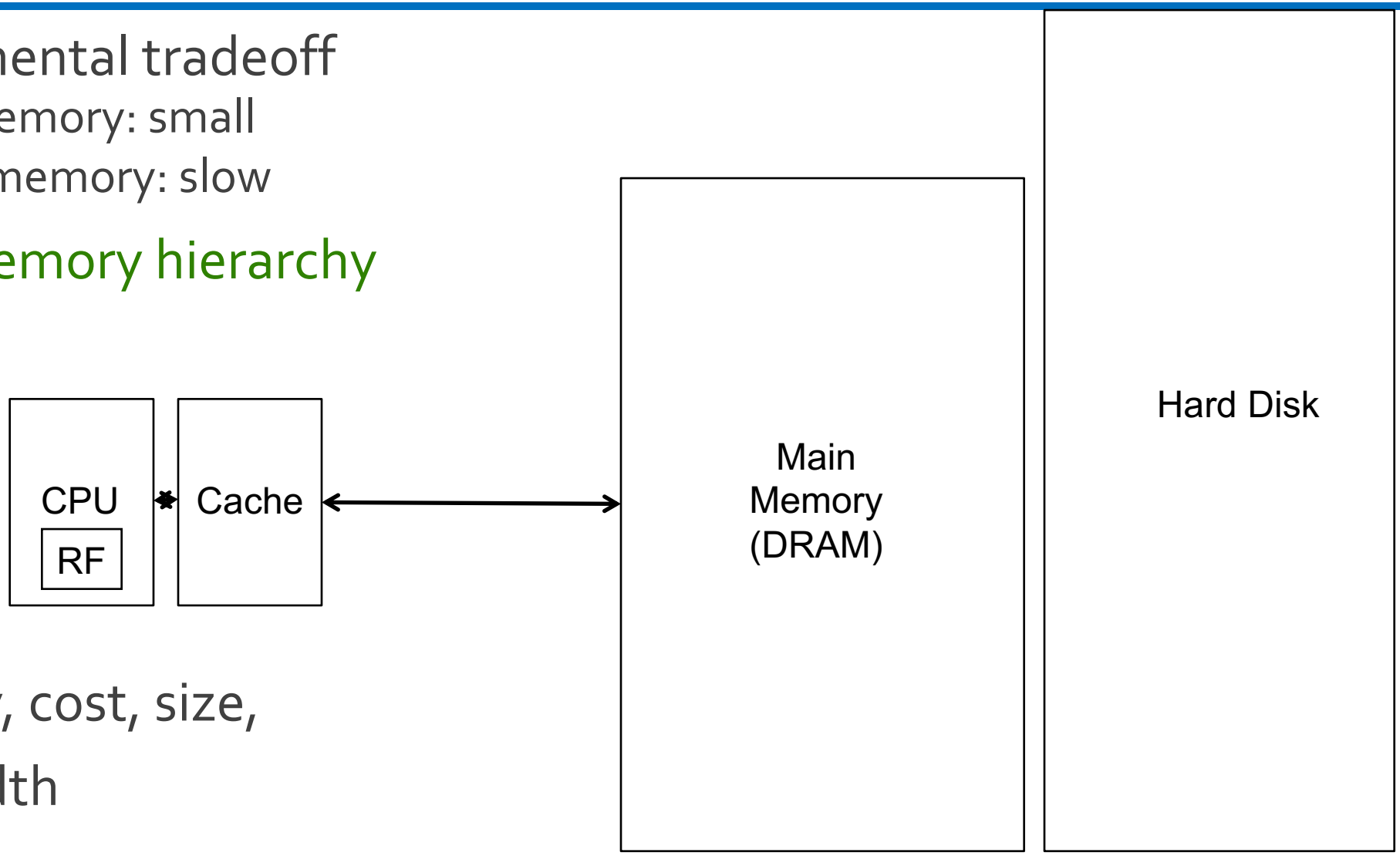


The Memory Hierarchy



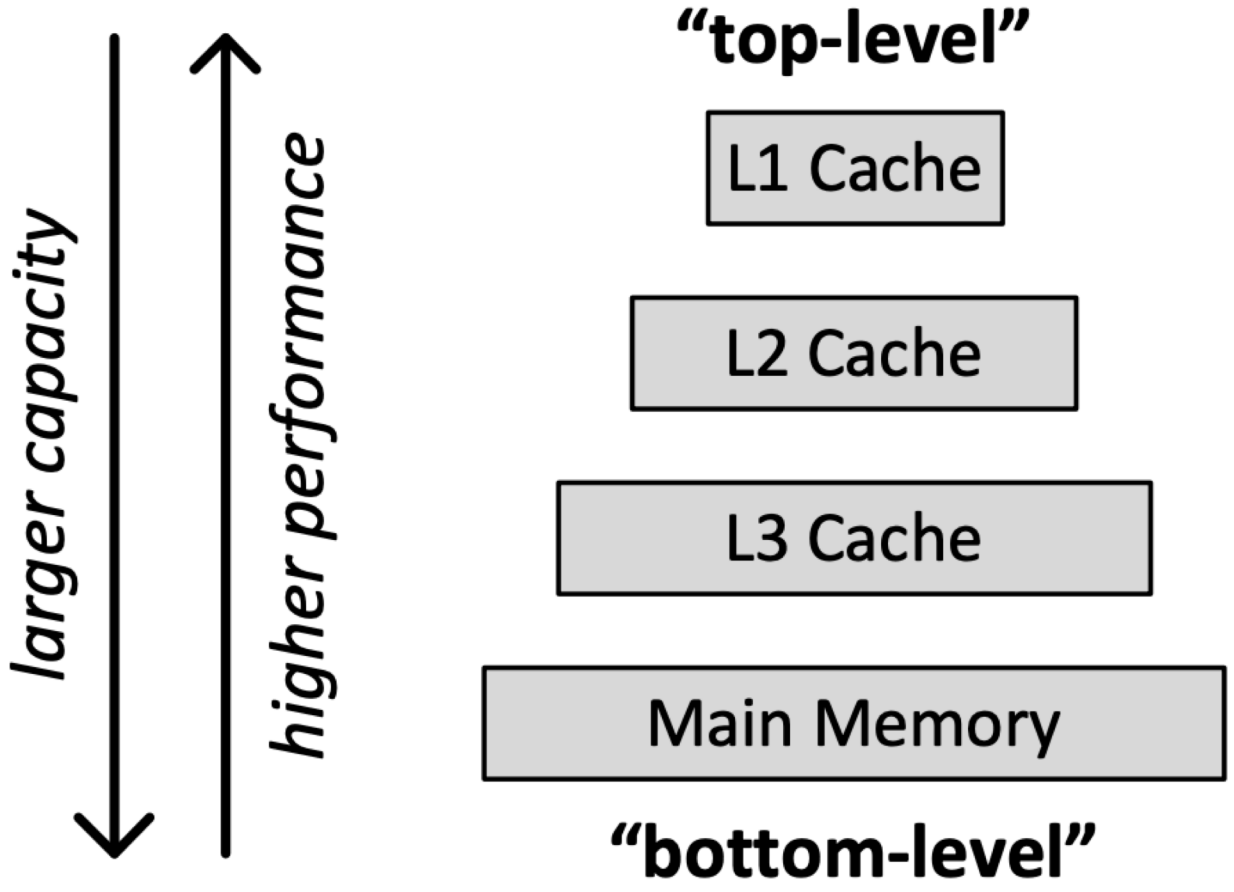
Memory Hierarchy

- Fundamental tradeoff
  - Fast memory: small
  - Large memory: slow
- Idea: Memory hierarchy



- Latency, cost, size, bandwidth

Memory Hierarchy Example



capacity	latency
10's of KB	≈ 1ns
100's of KB	< 5ns
several MB	≈ 10ns
several GB	≈ 100ns

Locality

- One's recent past is a very good predictor of their near future

- Temporal Locality: If you just did something, it is very likely that you will do the same thing again soon
  - since you are here today, there is a good chance you will be here again and again regularly

- Spatial Locality: If you did something, it is very likely you will do something similar/related (in space)
  - every time I find you in this room, you are probably sitting close to the same people AND/OR in closeby seats

Memory Locality

- A "typical" program has a lot of locality in memory references
  - typical programs are composed of "loops"

- Temporal: A program tends to reference the same memory location many times and all within a small window of time

- Spatial: A program tends to reference nearby memory locations within a window of time
  - most notable examples:
    1. instruction memory references → mostly sequential/streaming
    2. references to arrays/vectors → often streaming/strided

Caching Basics: Exploit Temporal Locality

- Idea: Store recently accessed data in automatically-managed fast memory (called cache)
- Anticipation: same mem. location will be accessed again soon

- Temporal locality principle
  - Recently accessed data will be again accessed in the near future
  - This is what Maurice Wilkes had in mind:
    - Wilkes, "Slave Memories and Dynamic Storage Allocation," IEEE Trans. On Electronic Computers, 1965.
    - "The use is discussed of a fast core memory of, say 32000 words as a slave to a slower core memory of, say, one million words in such a way that in practical cases the effective access time is nearer that of the fast memory than that of the slow memory."

Caching Basics: Exploit Spatial Locality

- Idea: Store data in addresses adjacent to the recently accessed one in automatically-managed fast memory
  - Logically divide memory into equal-size blocks
  - Fetch to cache the accessed block in its entirety
- Anticipation: nearby memory locations will be accessed soon

- Spatial locality principle
  - Nearby data in memory will be accessed in the near future
    - E.g., sequential instruction access, array traversal
  - This is what IBM 360/85 implemented
    - 16 Kbyte cache with 64 byte blocks
    - Liptay, "Structural aspects of the System/360 Model 85 II: the cache," IBM Systems Journal, 1968.

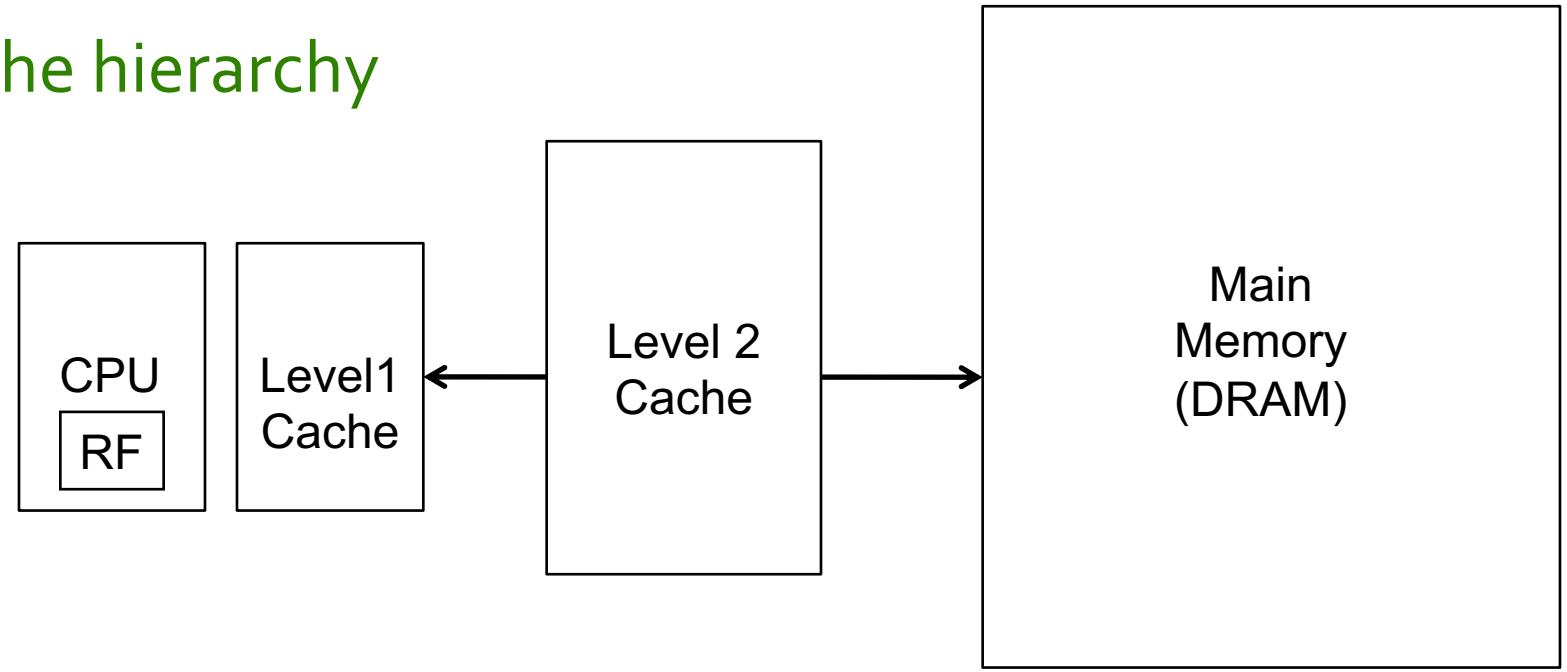
The Bookshelf Analogy

- Book in your hand
- Desk
- Bookshelf
- Boxes at home
- Boxes in storage

- Recently-used books tend to stay on desk
  - Comp Arch books, books for classes you are currently taking
  - Until the desk gets full
- Adjacent books in the shelf needed around the same time
  - If I have organized/categorized my books well in the shelf

Caching in a Pipelined Design

- The cache needs to be tightly integrated into the pipeline
  - Ideally, access in 1-cycle so that load-dependent operations do not stall
- High frequency pipeline → Cannot make the cache large
  - But, we want a large cache AND a pipelined design
- Idea: Cache hierarchy



A Note on Manual vs. Automatic Management

- Manual: Programmer manages data movement across levels

  -- too painful for programmers on substantial programs

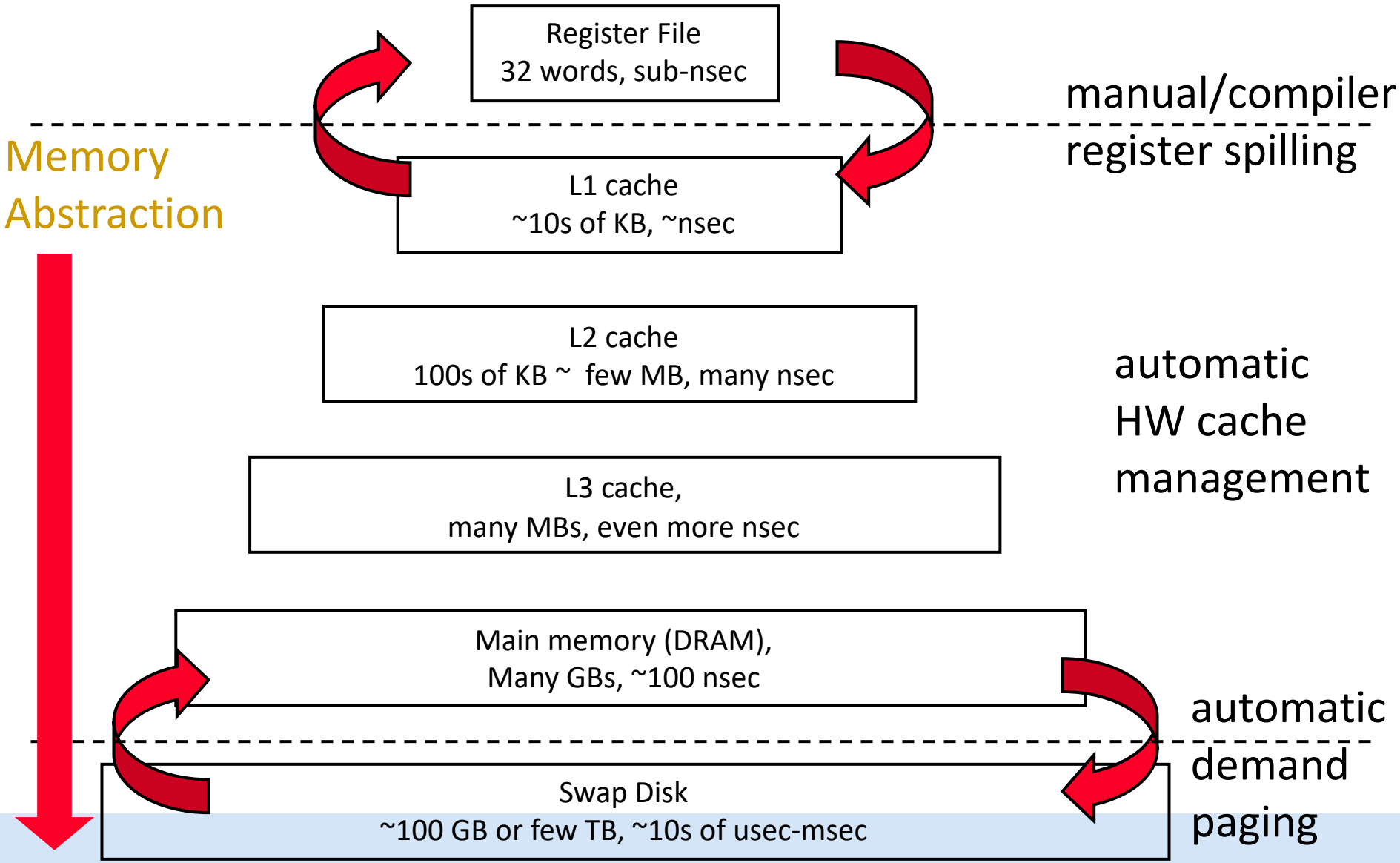
  - "core" vs "drum" memory in the 1950s
  - done in embedded processors (on-chip scratchpad SRAM in lieu of a cache), GPUs (called "shared memory"), ML accelerators, …

- Automatic: Hardware manages data movement across levels, transparently to the programmer

  ++ programmer's life is easier

  - the average programmer doesn't need to know about caches
    - You don't need to know how big the cache is and how it works to write a "correct" program! (What if you want a "fast" program?)

A Modern Memory Hierarchy



Hierarchical Latency Analysis

- A given memory hierarchy level i has **intrinsic access time** of t_i
- It also has **perceived access time** T_i that is longer than t_i
- Except for the outer-most hierarchy level, when looking for a given address there is
  - a chance (hit-rate h_i) you "hit" and access time is t_i
  - a chance (miss-rate m_i) you "miss" and access time $t_i + T_{i+1}$
  - $h_i + m_i = 1$
- Thus

$$T_i = h_i \cdot t_i + m_i \cdot (t_i + T_{i+1})$$

$$T_i = t_i + m_i \cdot T_{i+1}$$

h_i and m_i are defined to be the hit-rate and miss-rate of only the references that missed at L_{i-1}

Hierarchy Design Considerations

- Recursive latency equation

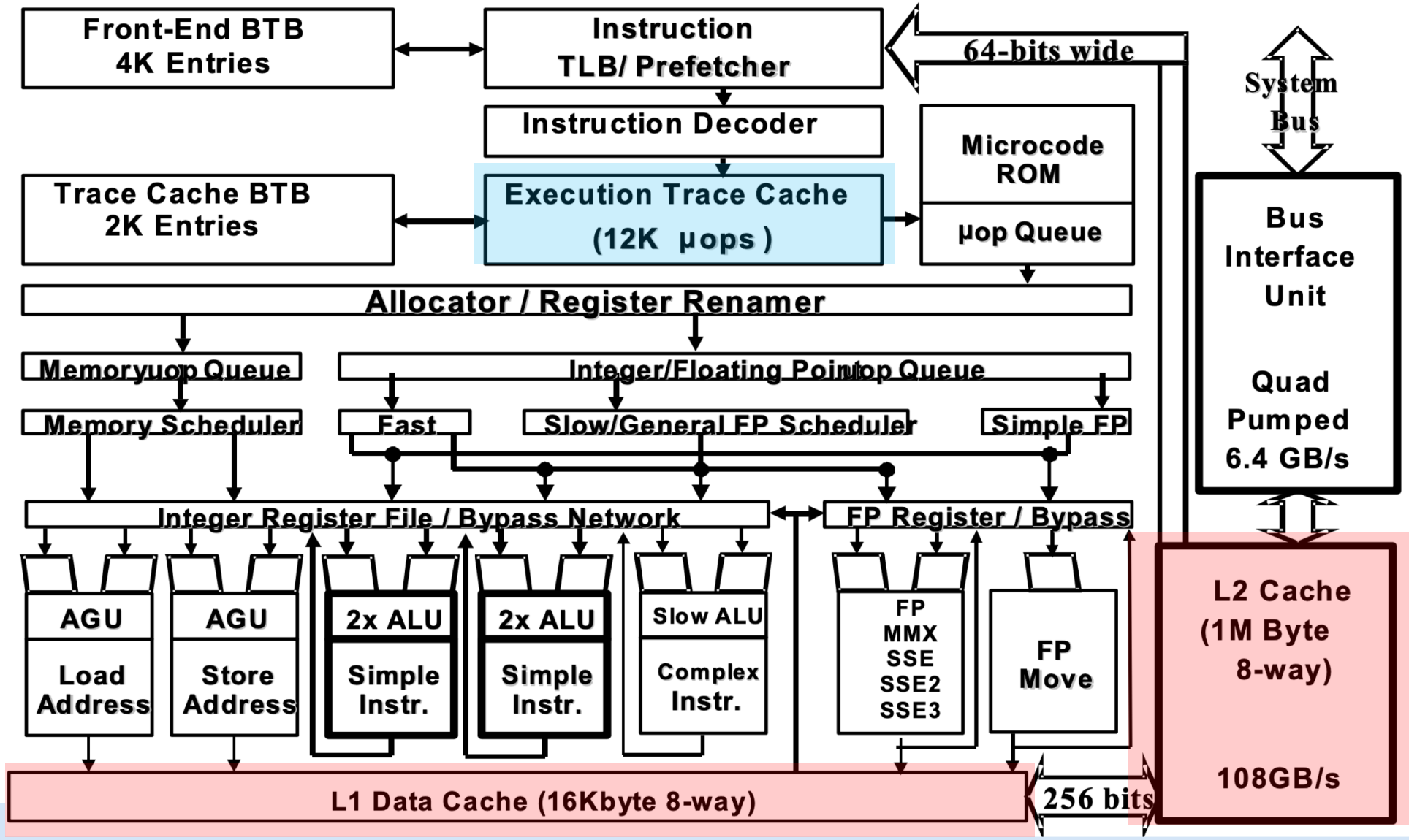
$$T_i = t_i + m_i \cdot T_{i+1}$$

- The goal: achieve the desired T_1 within the allowed cost
- $T_i \approx t_i$ is desirable

- Keep m_i low
  - increasing capacity C_i lowers m_i, but beware of increasing t_i
  - lower m_i by smarter cache management (replacement::anticipate what you don't need, prefetching::anticipate what you will need)

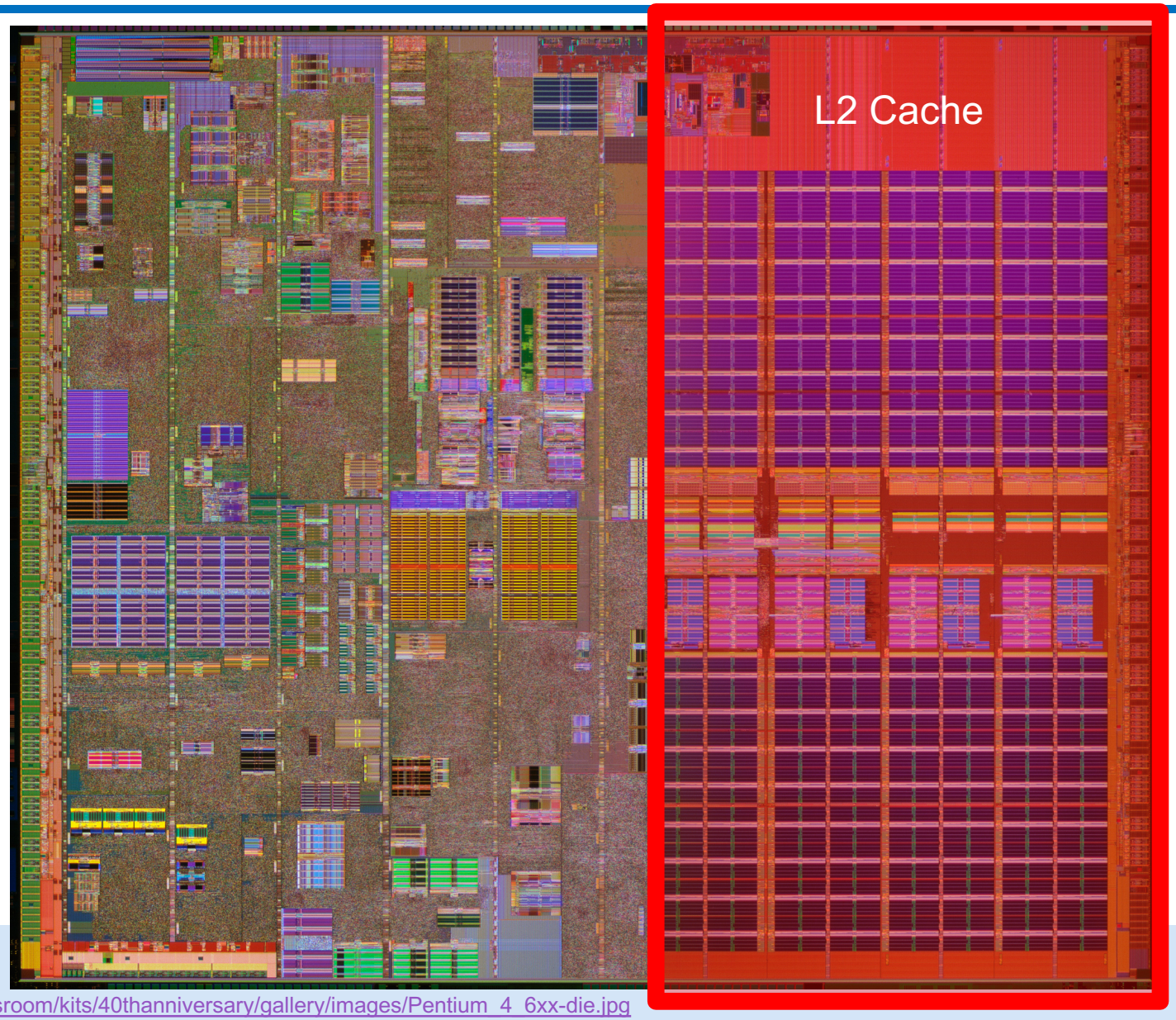
- Keep T_{i+1} low
  - faster outer hierarchy levels can help, but beware of increasing cost
  - introduce intermediate hierarchy levels as a compromise

Intel Pentium 4 Example



Boggs et al., "The Microarchitecture of the Pentium 4 Processor," Intel Technology Journal, 2004.

Intel Pentium 4 Example



https://download.intel.com/newsroom/kits/40thanniversary/gallery/images/Pentium_4_6xx-die.jpg

Intel Pentium 4 Example

- 90nm P4, 3.6 GHz
- L1 D-cache
  - C_1 = 16 kB
  - t_1 = 4 cyc int / 9 cycle fp
- L2 D-cache
  - C_2 = 1024 kB
  - t_2 = 18 cyc int / 18 cyc fp
- Main memory
  - t_3 = ~ 50ns or 180 cyc
- Notice
  - best case latency is not 1
  - worst case access latencies are into 500+ cycles

$$T_i = t_i + m_i \cdot T_{i+1}$$

if m_1=0.1, m_2=0.1
T_1=7.6, T_2=36

if m_1=0.01, m_2=0.01
T_1=4.2, T_2=19.8

if m_1=0.05, m_2=0.01
T_1=5.00, T_2=19.8

if m_1=0.01, m_2=0.50
T_1=5.08, T_2=108

KING'S College LONDON

Hardware Design

Lecture 7: Memory

Dr. Haiyu Mao

12.03.2026