18-600 Foundations of Computer Systems

Lecture 10:
"The Memory Hierarchy"

John P. Shen & Gregory Kesden
October 2, 2017

- Required Reading Assignment:
  - **Chapter 6 of CS:APP (3rd edition) by Randy Bryant & Dave O'Hallaron**
- Recommended Reference:
  - Sec. 1 & Sec. 3: Bruce Jacob, "The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It," Synthesis Lectures on Computer Architecture 2009.

Electrical & Computer ENGINEERING

18-600 Foundations of Computer Systems

Lecture 10:
“The Memory Hierarchy”

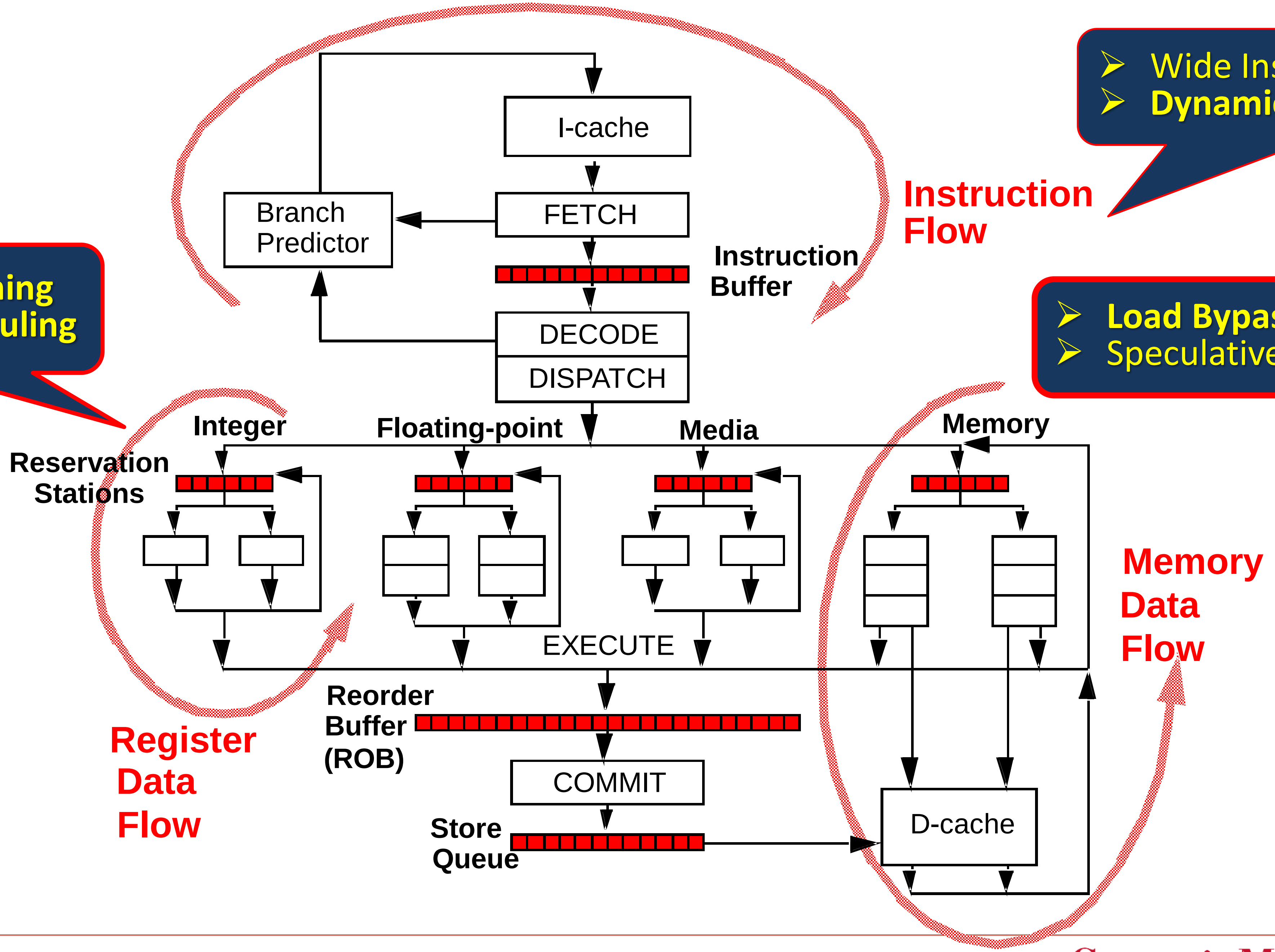
A. Memory Technologies
B. Main Memory Implementation
   a. DRAM Organization
   b. DRAM Operation
   c. Memory Controller
C. Disk Storage Technologies

Three Flow Paths of Superscalar Processors

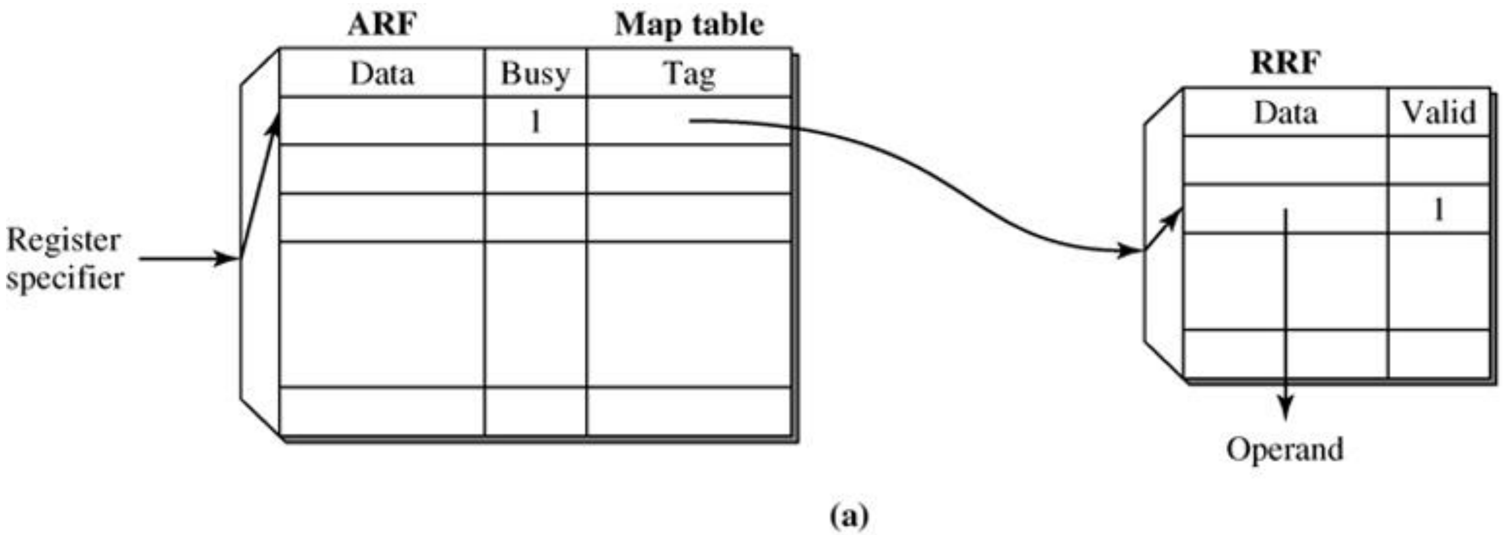
- Wide Instruction Fetching
- **Dynamic Branch Prediction**

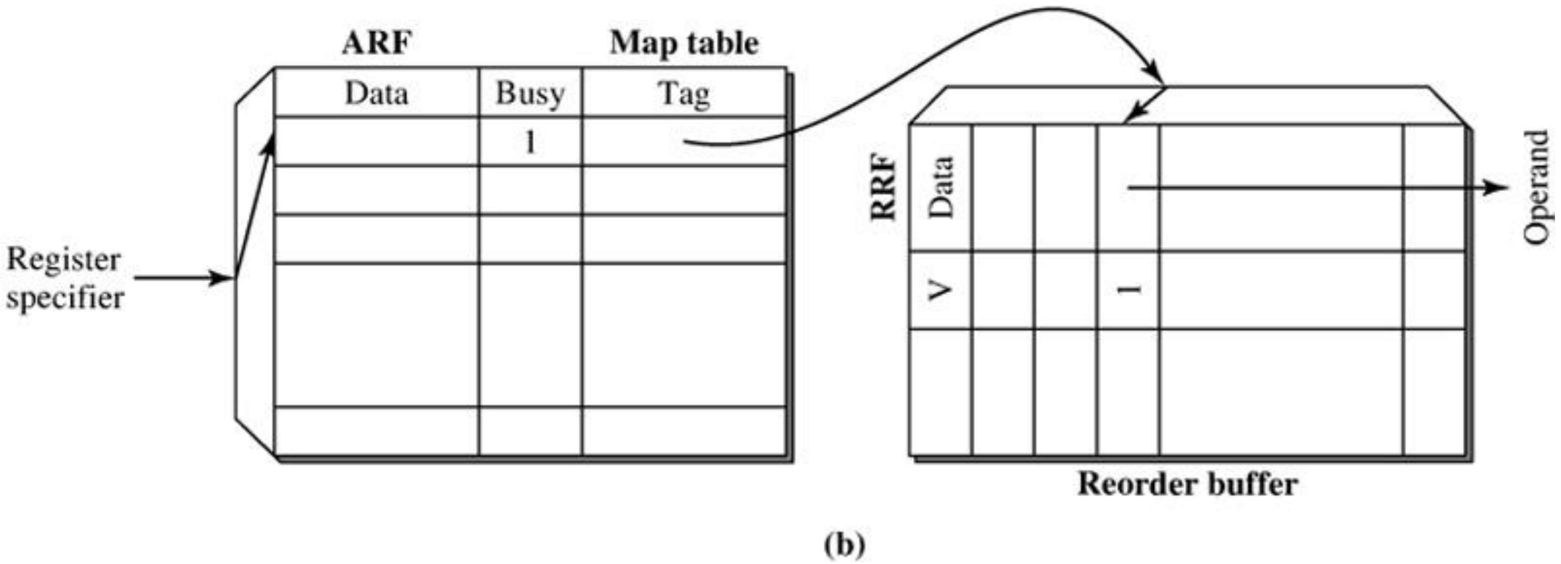
- **Register Renaming**
- **Dynamic Scheduling**

- **Load Bypassing & Forwarding**
- Speculative Memory Disamb.

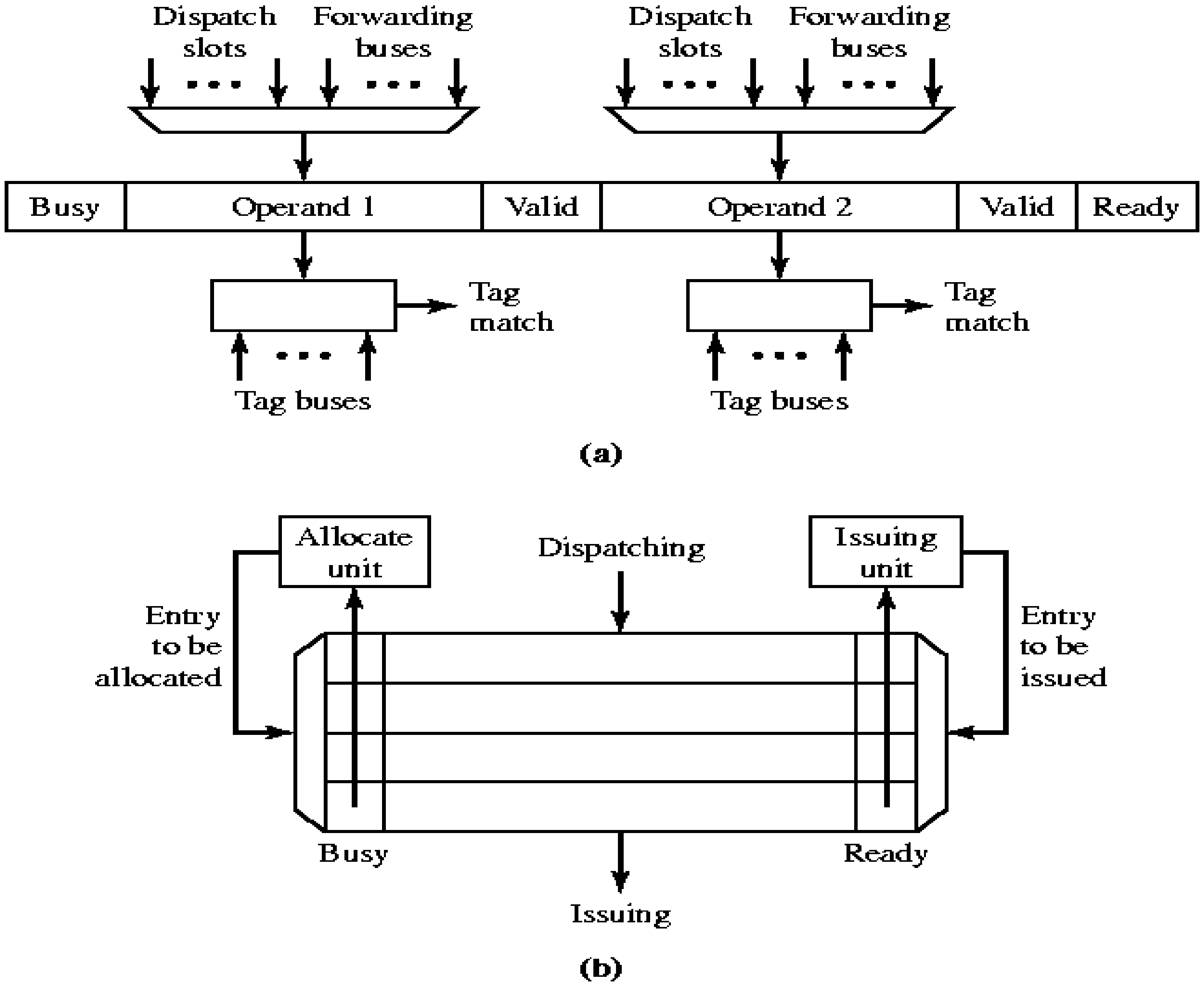


Integrating Map Table with the ARF





Reservation Station Implementation



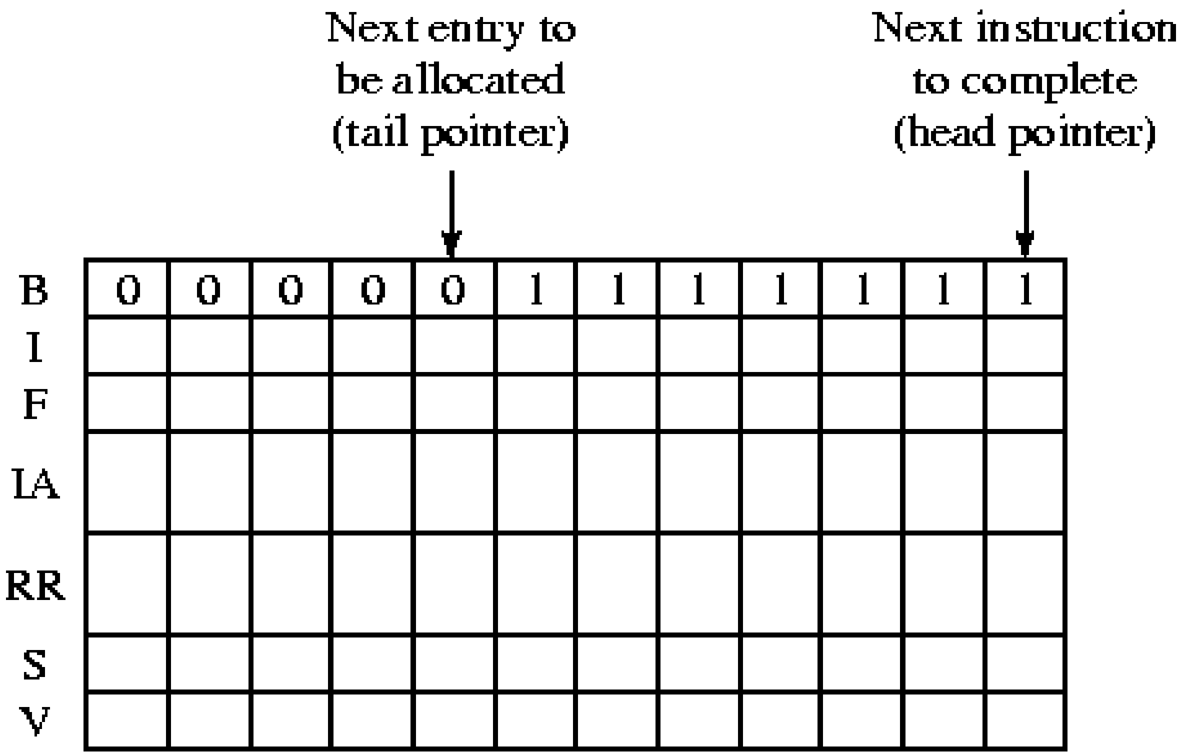
+ info for executing instruction (opcode, ROB entry, RRF entry…)

- Reservation Stations: distributed vs. centralized
  - Wakeup: benefit to partition across data types
  - Select: much easier with partitioned scheme
    - Select 1 of n/4 vs. 4 of n

Reorder Buffer Implementation

Busy	Issued	Finished	Instruction address	Rename register	Speculative	Valid

(a)


Next entry to be allocated (tail pointer)
Next instruction to complete (head pointer)

B	0	0	0	0	0	1	1	1	1	1	1	1
I												
F												
IA												
RR												
S												
V												

Reorder buffer

(b)

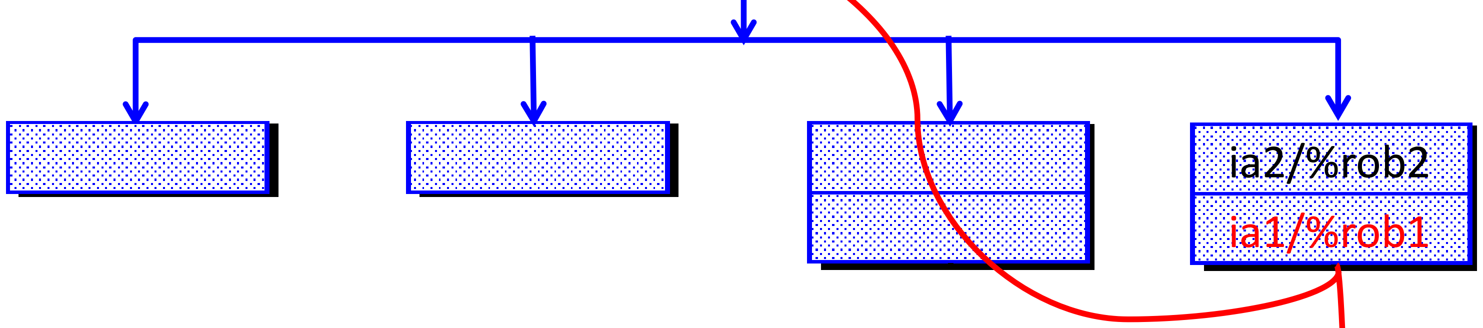
- Reorder Buffer
  - "Bookkeeping"
  - Can be instruction-grained, or block-grained (4-5 ops)

Cycle: 001

```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
     Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```

ia1	1	%rax	1	-- /%rob1	1	1
ia2	1	%rip	1	-- /%rob2	1	1
ia3	1	%rob1	0	%rob2/%rob3	0	0
ia4	1	%rax	1	-- /%rob4	1	0

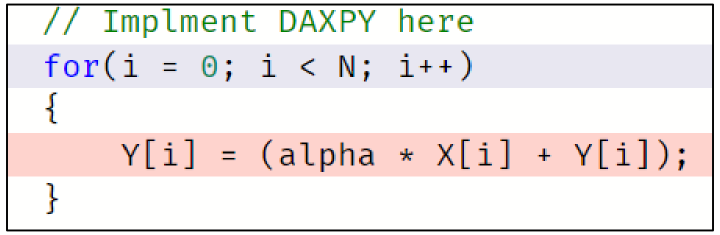


TP HP

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
I												0	0	1	1	
F												0	0	0	0	
IA												ia4	ia3	ia2	ia1	
RR												rob4	rob3	rob2	rob1	
S																
V												0	0	0	0	

Four instructions (ia1, ia2, ia3, ia4) have been dispatched to RS with four corresponding entries allocated in ROB. ia1 and ia2 have been issued into LS FU. Next cycle, ia1 will finish and broadcast its result using tag "%rob1" to ia3.

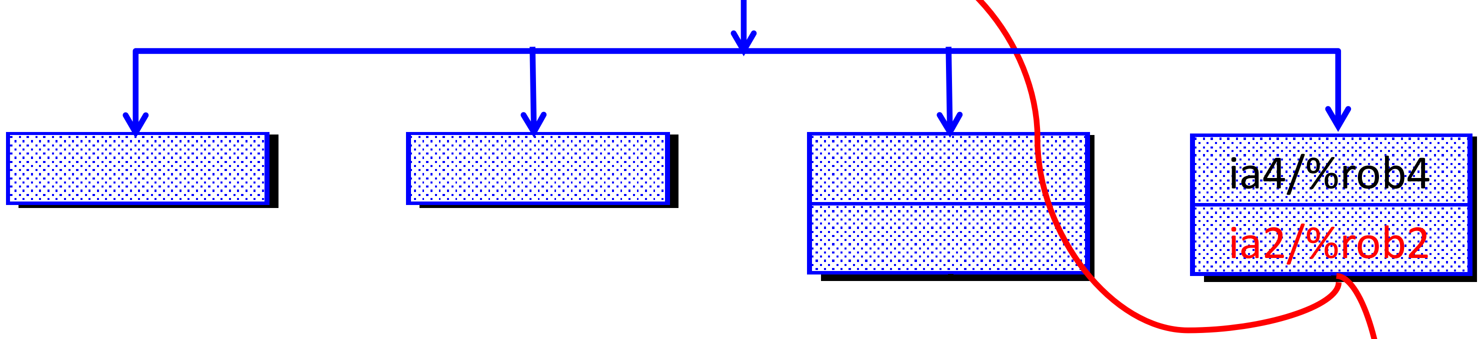
Cycle: 002



```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
            movl $0, -4(%rbp)
            jmp .L2
.L3:
ia1         movsd X(,%rax,8), %xmm1
ia2         movsd alpha(%rip), %xmm0
ia3         mulsd %xmm1, %xmm0
ia4         movsd Y(,%rax,8), %xmm1
ia5         addsd %xmm1, %xmm0
ia6         movsd %xmm0, Y(,%rax,8)
ia7         addl $1, -4(%rbp)
.L2:
ia8         cmpl $2055, -4(%rbp)
ia9         jle .L3
```

ia5	1	%rob3	0	%rob4/**%rob5**	1	1
ia2	1	%rip	1	-- /**%rob2**	1	1
ia3	1	%rob1	1	%rob2/**%rob3**	0	0
ia4	1	%rax	1	-- /**%rob4**	1	1



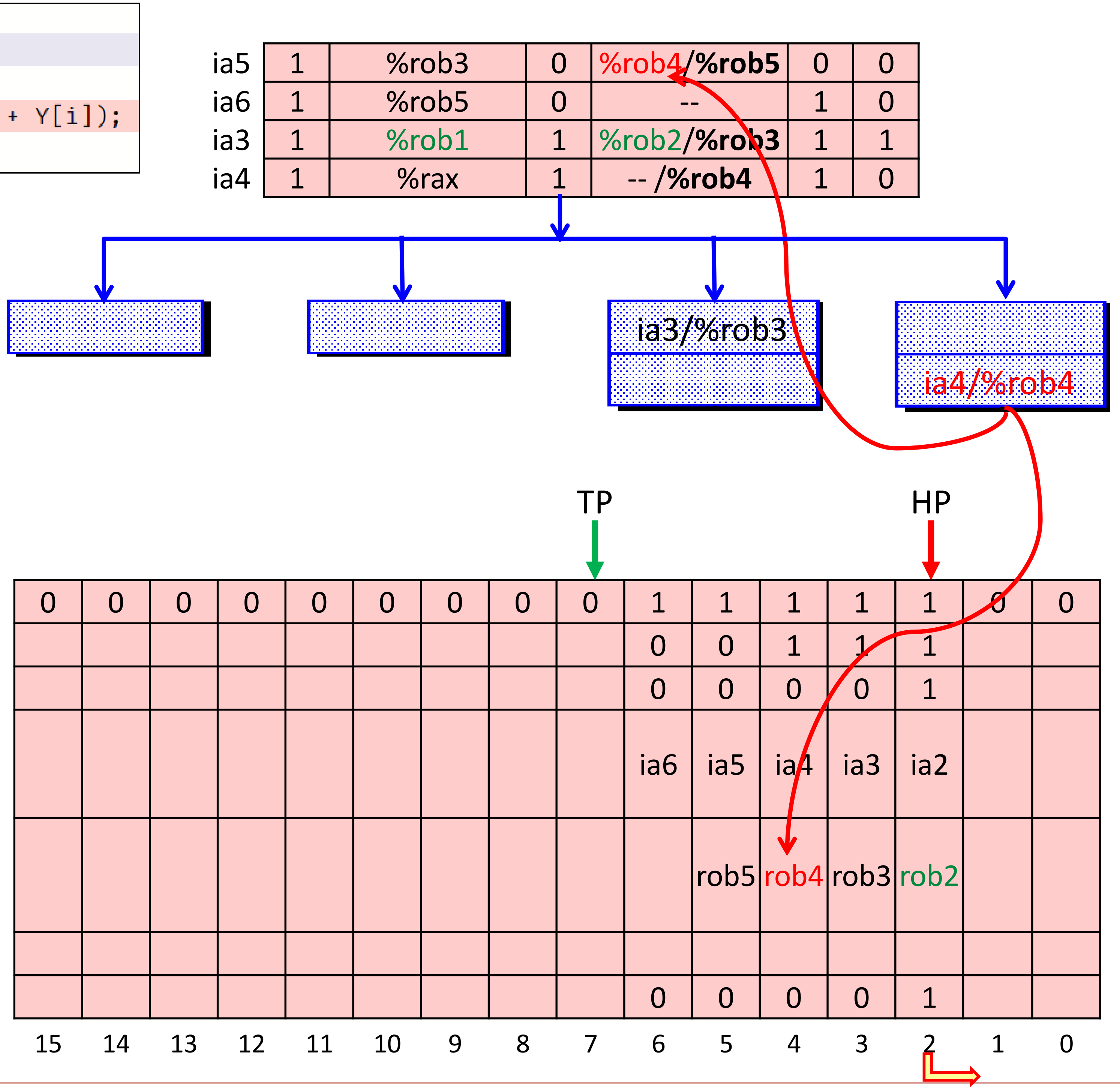
											TP					HP	
B	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0
I											0	1	0	1	1	
F											0	0	0	0	1	
IA											ia5	ia4	ia3	ia2	ia1	
RR											Rob5	rob4	rob3	rob2	rob1	
S																
V											0	0	0	0	1	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ia1 completes and is next to leave ROB.
Its RS entry has been reallocated to ia5.
ia4 is issued into LS FU.
Next cycle, ia2 will broadcast its results using tag "%rob2" to ia3.

Cycle: 003

```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```



ia5	1	%rob3	0	%rob4/%rob5	0	0
ia6	1	%rob5	0	--	1	0
ia3	1	%rob1	1	%rob2/%rob3	1	1
ia4	1	%rax	1	-- /%rob4	1	0

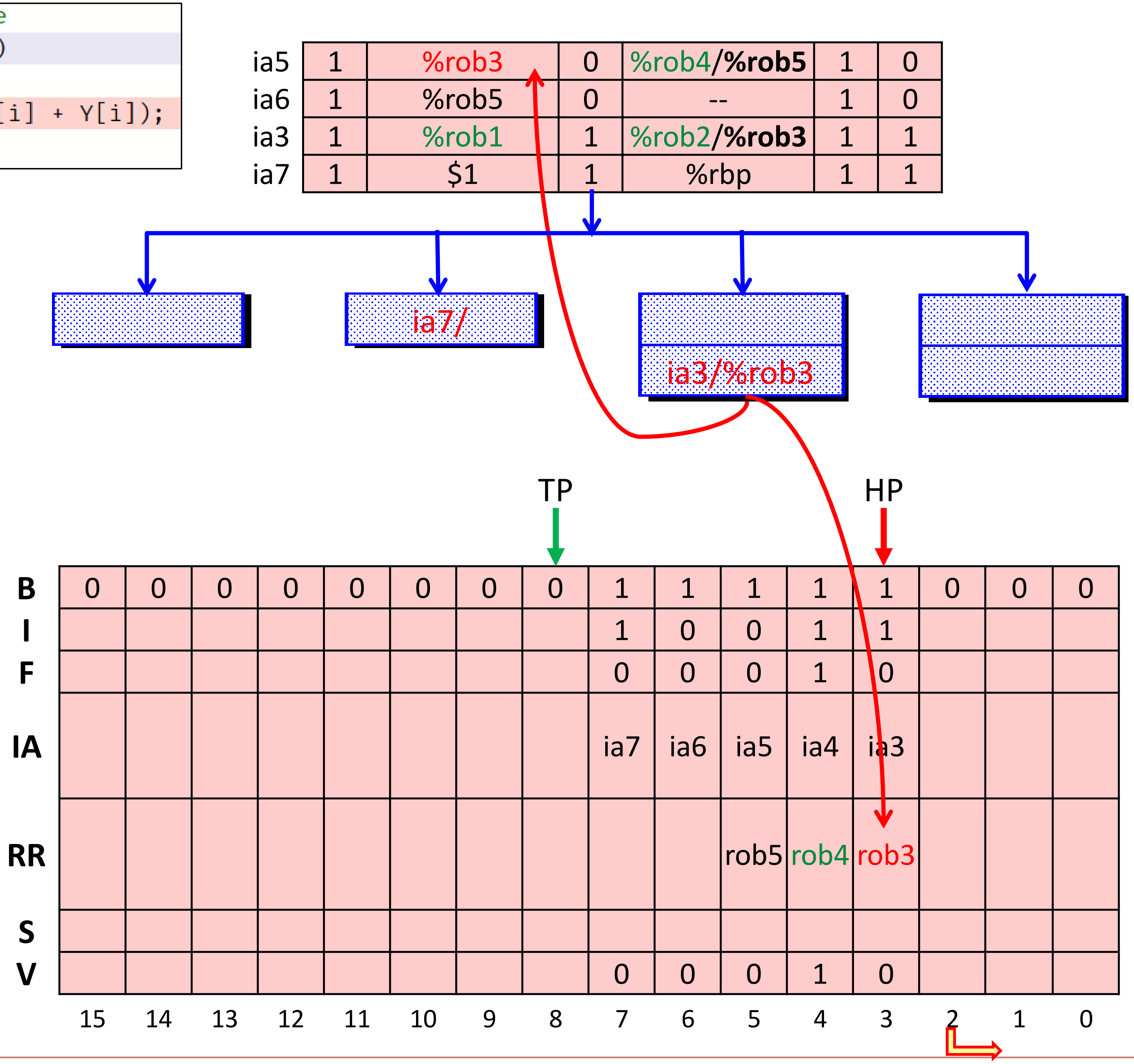
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
I										0	0	1	1	1		
F										0	0	0	0	1		
IA										ia6	ia5	ia4	ia3	ia2		
RR											rob5	rob4	rob3	rob2		
S																
V										0	0	0	0	1		

ia2 completes and is ready to leave ROB.
ia3 is now ready and is issued to MULT FU.
Next cycle, ia4 will forward result to ia5 in RS using tag "%rob4".

Cycle: 004

```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```



ia5	1	%rob3	0	%rob4/%rob5	1	0
ia6	1	%rob5	0	--	1	0
ia3	1	%rob1	1	%rob2/%rob3	1	1
ia7	1	$1	1	%rbp	1	1

B	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0
I									1	0	0	1	1			
F									0	0	0	1	0			
IA									ia7	ia6	ia5	ia4	ia3			
RR											rob5	rob4	rob3			
S																
V									0	0	0	1	0			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

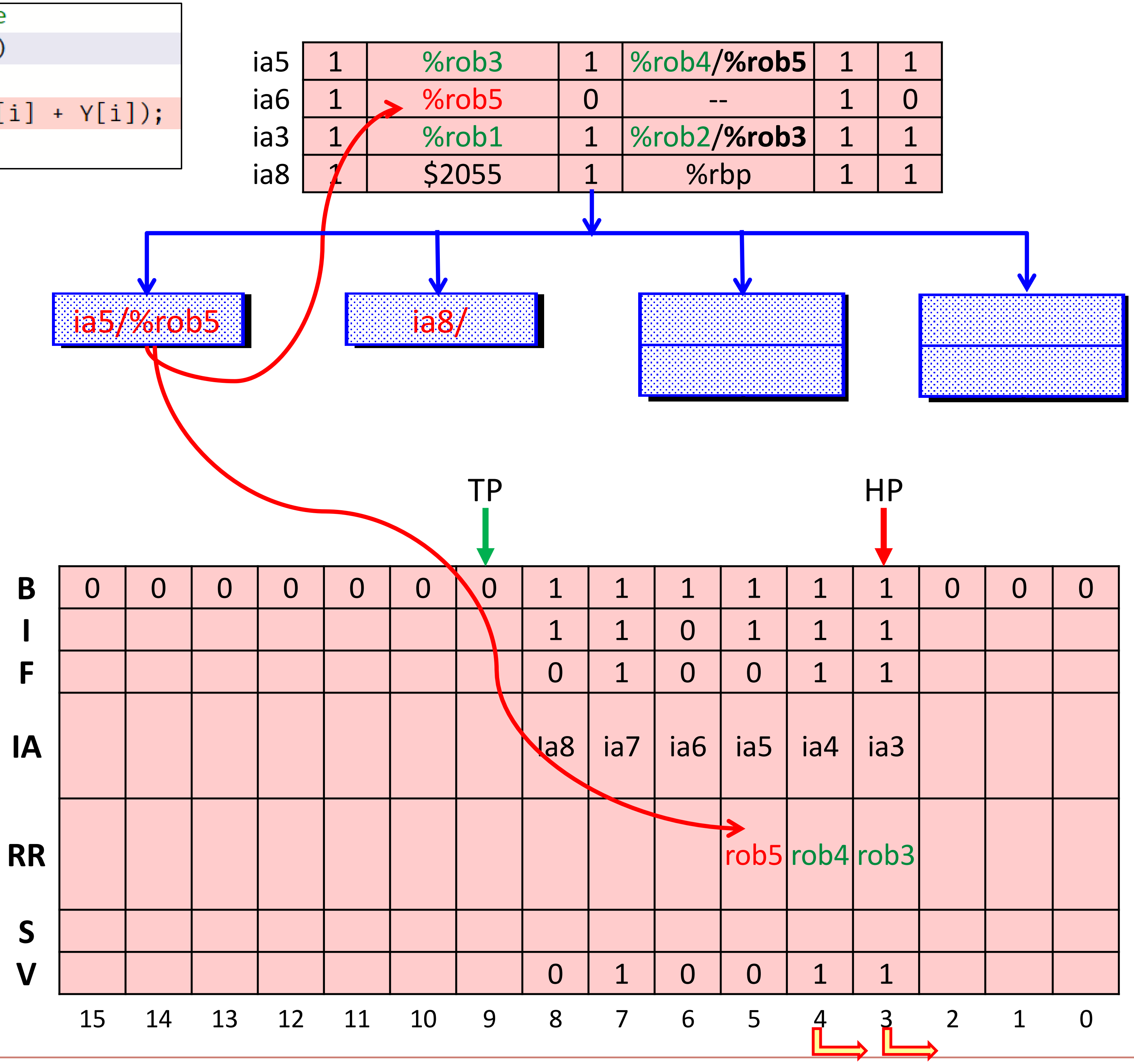
ia4 finishes but must wait for ia3 before it can leave ROB.
ia7 has been dispatched to RS and allocated entry in ROB, and is issued.
Next cycle, ia3 will forward result to ia5 using the tag "%rob3".

Cycle: 005

```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```

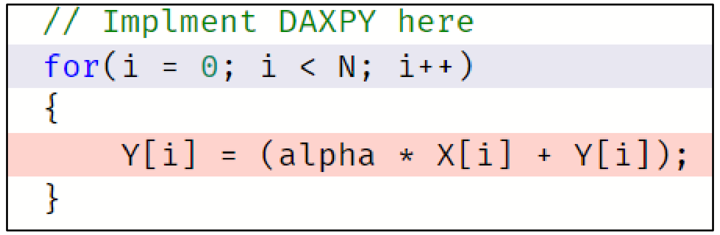
ia5 is now ready and is issued to FU. Next cycle, ia5 will forward result to ia6 using the tag "%rob5".
ia7 finishes, its RS deallocated, but must wait in ROB. ia3 and ia4 ready to retire.
ia8 is dispatched to RS and issued to FU.



ia5	1	%rob3	1	%rob4/**%rob5**	1	1
ia6	1	%rob5	0	--	1	0
ia3	1	%rob1	1	%rob2/**%rob3**	1	1
ia8	1	$2055	1	%rbp	1	1

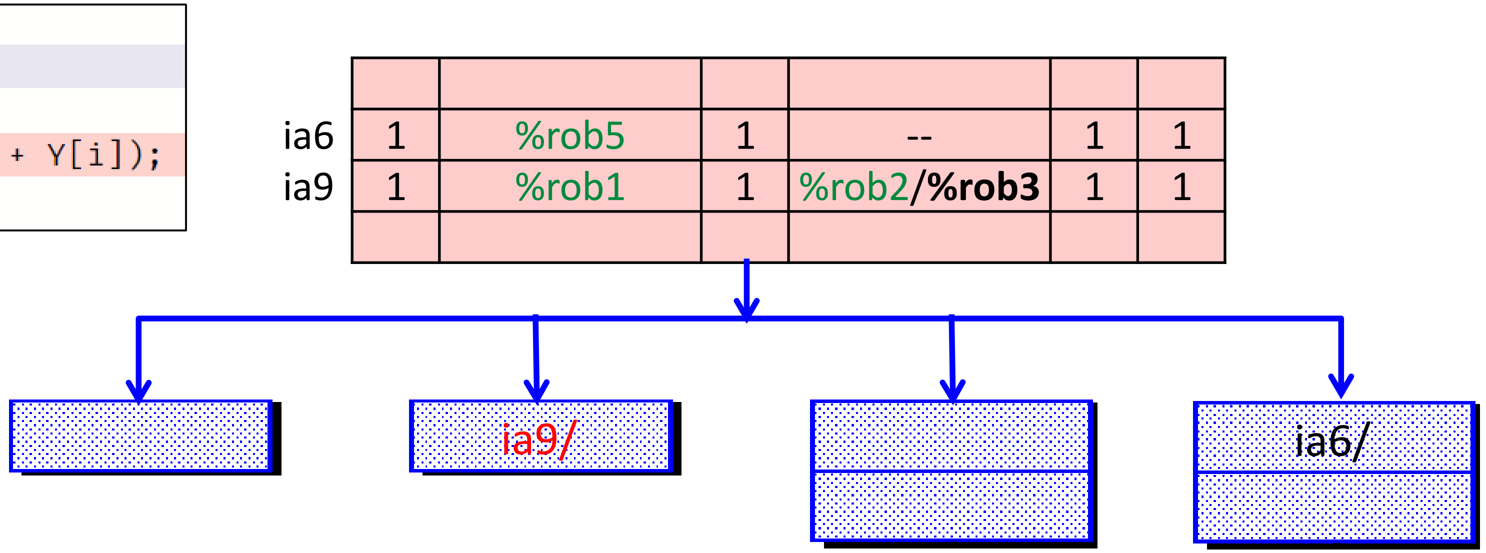
B	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0
I								1	1	0	1	1	1			
F								0	1	0	0	1	1			
IA								Ia8	ia7	ia6	ia5	ia4	ia3			
RR											rob5	rob4	rob3			
S																
V								0	1	0	0	1	1			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Cycle: 006

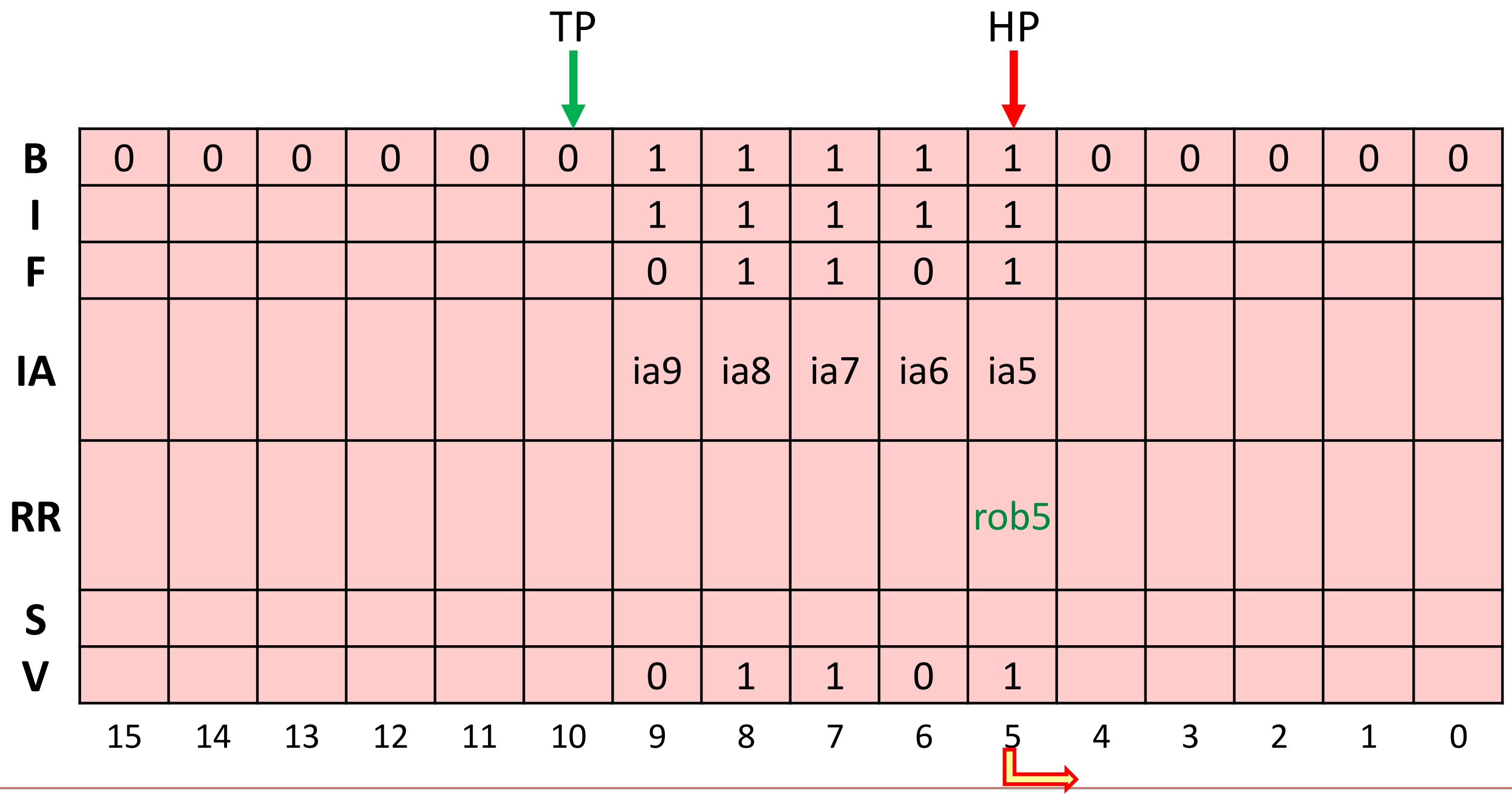


```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```



ia6	1	%rob5	1	--	1	1
ia9	1	%rob1	1	%rob2/%rob3	1	1



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
I							1	1	1	1	1					
F							0	1	1	0	1					
IA							ia9	ia8	ia7	ia6	ia5					
RR											rob5					
S																
V							0	1	1	0	1					

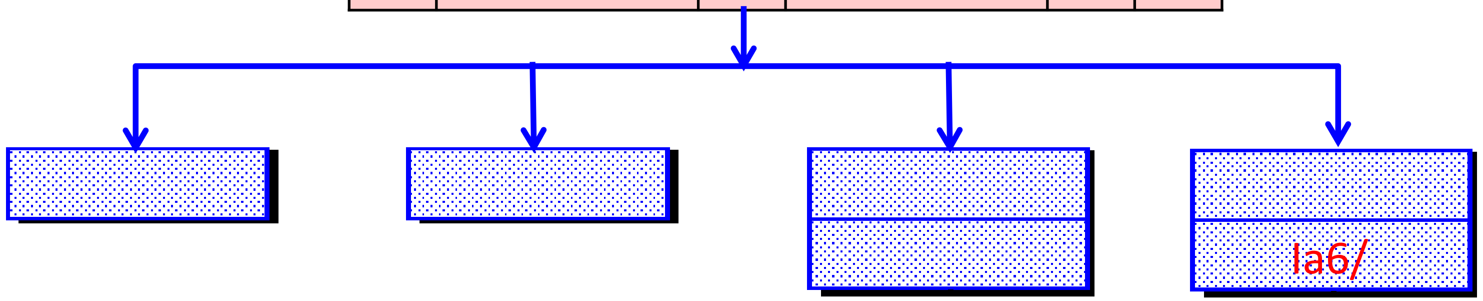
is6 is now ready and is issued to LS FU.
ia9 is dispatched to RS and issued into FU.
ia5 finishes, its RS deallocated, and is ready to leave ROB.
ia8 finishes, its RS deallocated, but must wait in ROB.

Cycle: 007

```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```

	0					
ia6	1	%rob5	1	--	1	1
	0					
	0					



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						TP				HP						
B	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
I							1	1	1	1						
F							1	1	1	0						
IA							ia9	ia8	ia7	ia6						
RR																
S																
V							1	1	1	0						

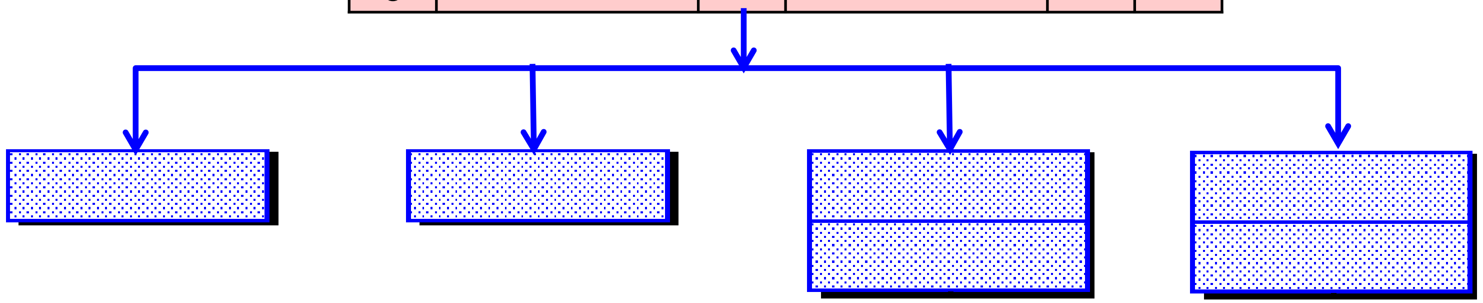
ia9 finishes, its RS deallocated, but must wait in ROB.
ia6 is finishing and will be complete next cycle and be ready to leave ROB.

Cycle: 008

```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
     Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```

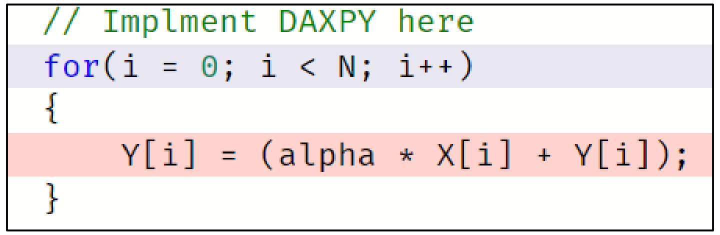
	0					
ia6	1	%rob5	1	--	1	1
	0					
	0					



						TP				HP						
B	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
I							1	1	1	1						
F							1	1	1	1						
IA							ia9	ia8	ia7	ia6						
RR																
S																
V							1	1	1	1						
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

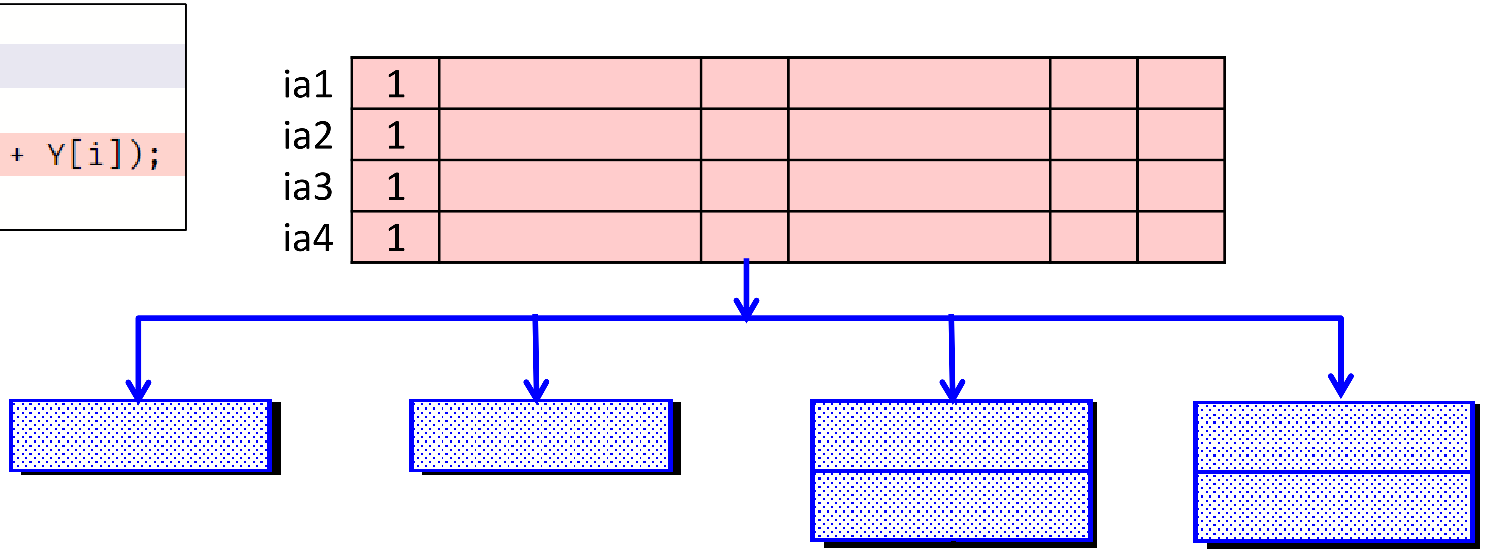
i6 completes and all four remaining instructions will leave ROB next cycle.

Cycle: 009

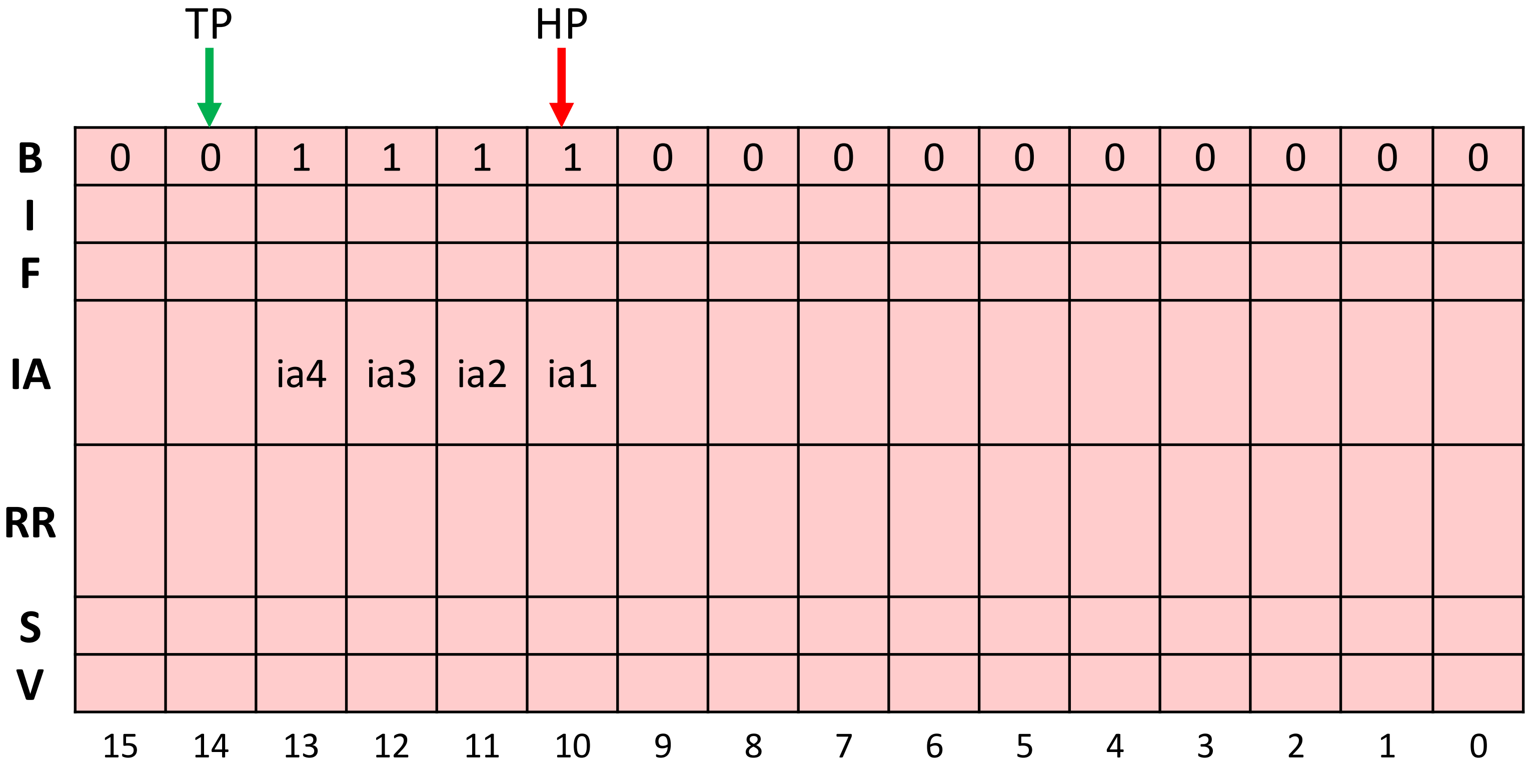


```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```



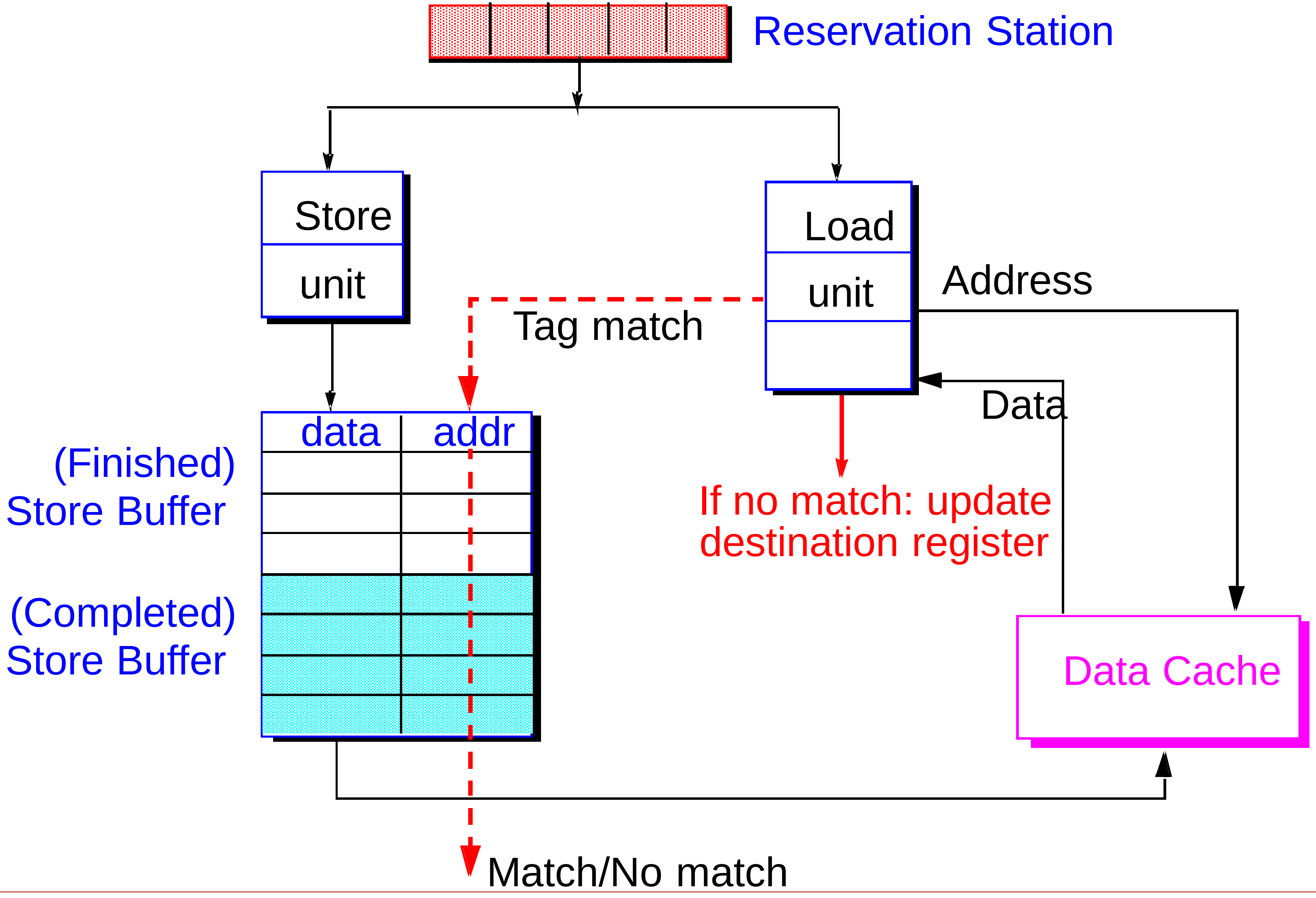
ia1	1					
ia2	1					
ia3	1					
ia4	1					



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
I																
F																
IA			ia4	ia3	ia2	ia1										
RR																
S																
V																

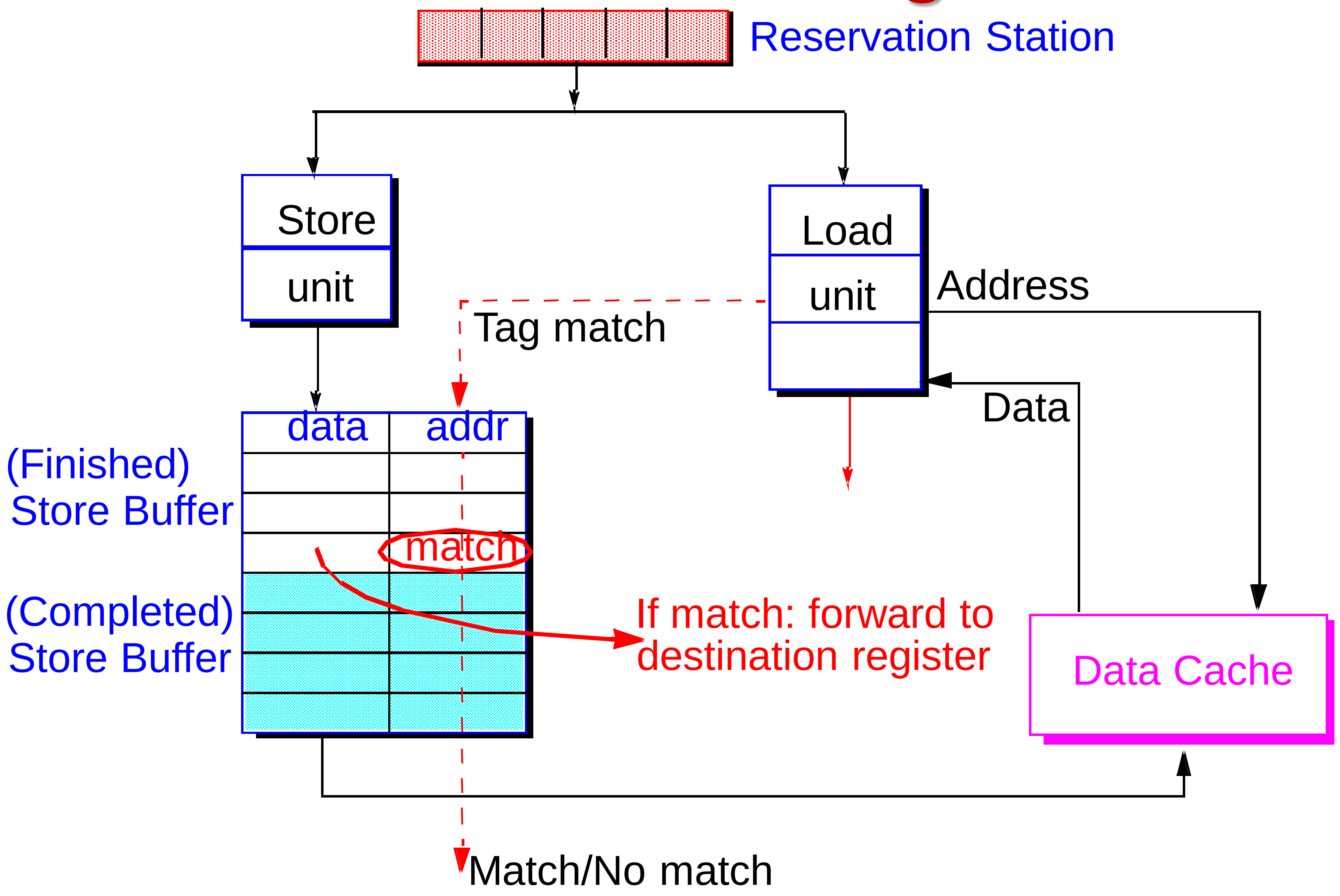
ia9 is executed and branches to top of the loop and starts next iteration by dispatching ia1, ia2, ia3, and ia4 to RS and ROB. The same process is repeated.

Illustration of Load Bypassing

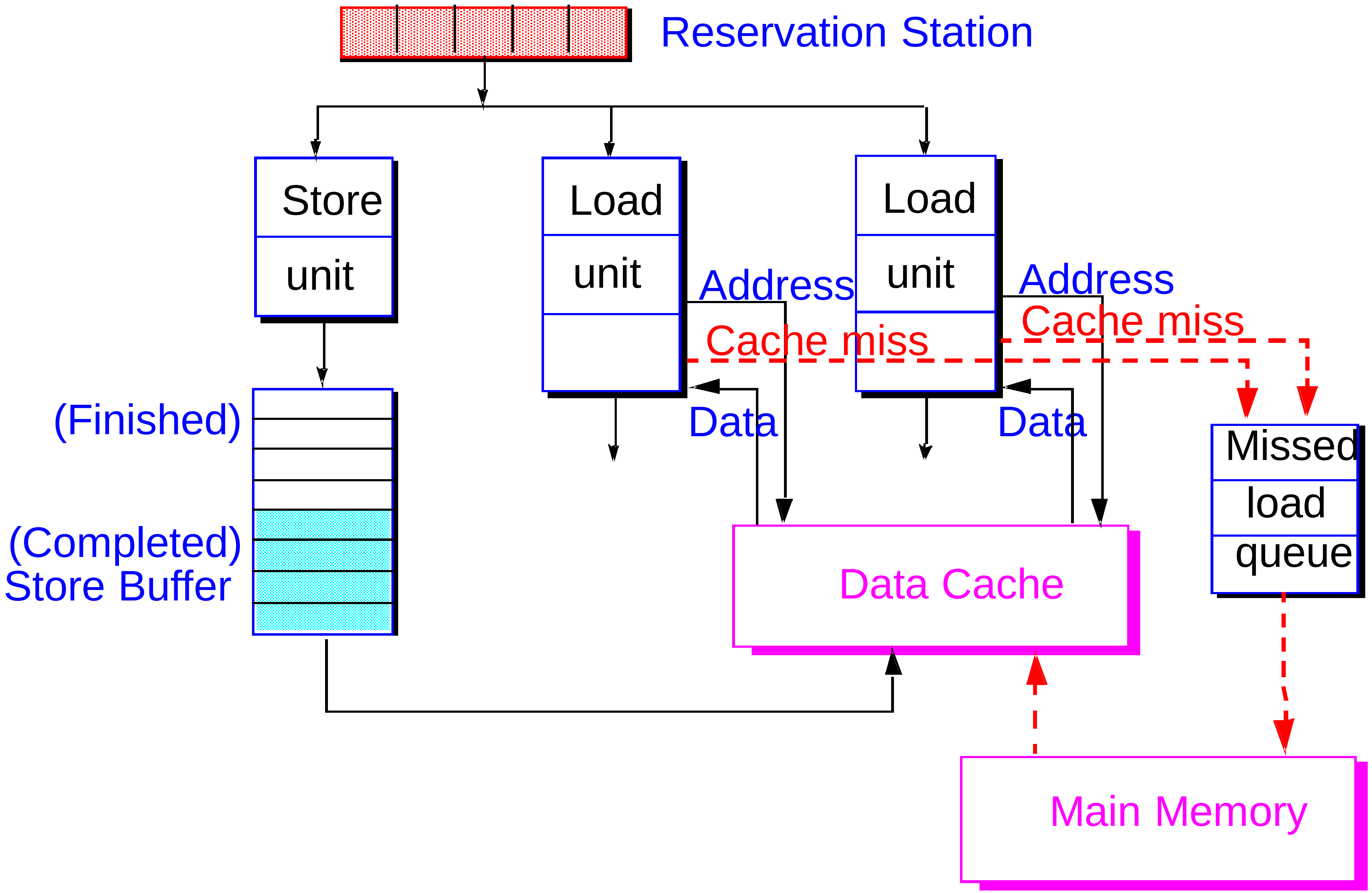


From Lec #9 …

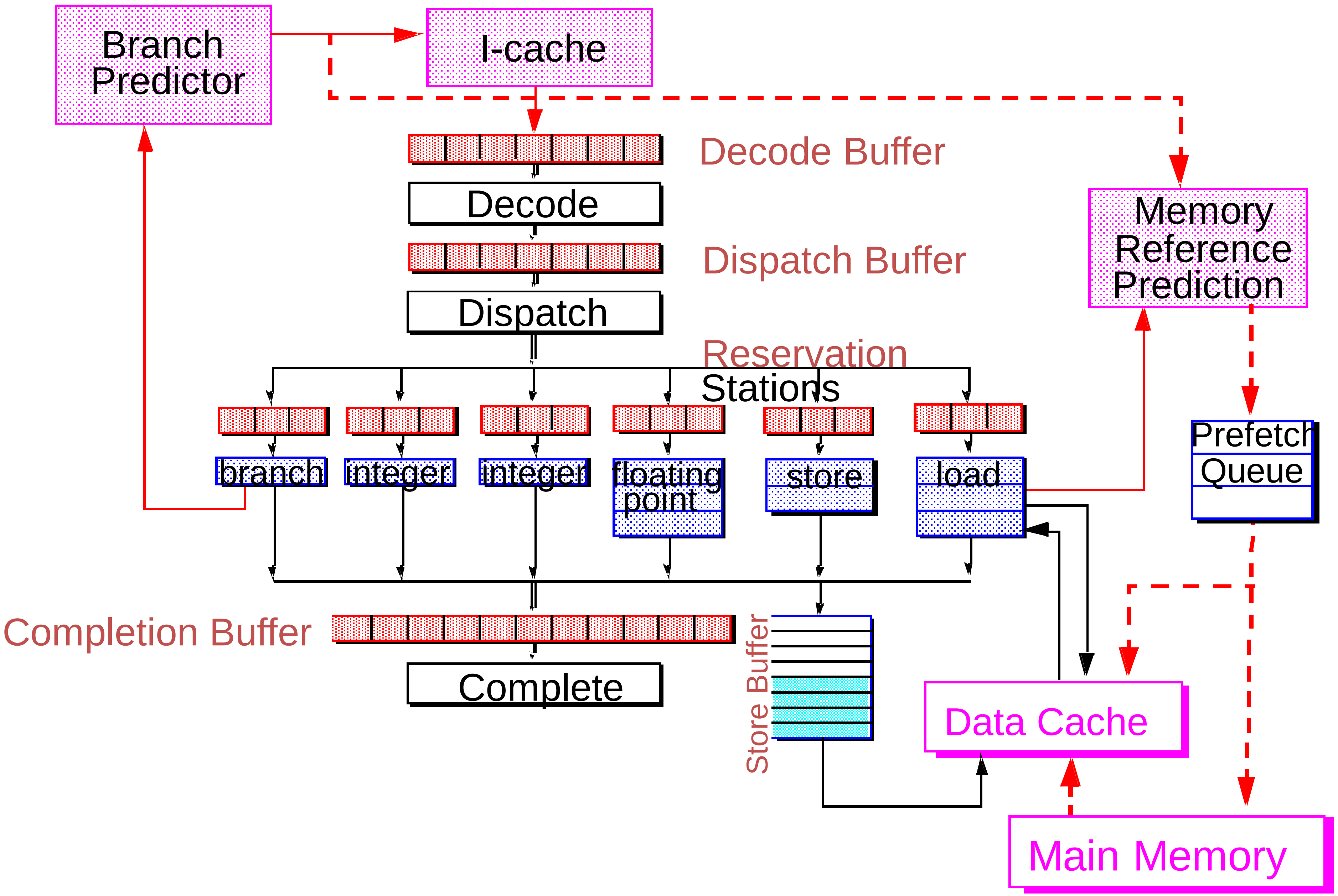
Illustration of Load Forwarding



Dual-Ported Non-Blocking Cache

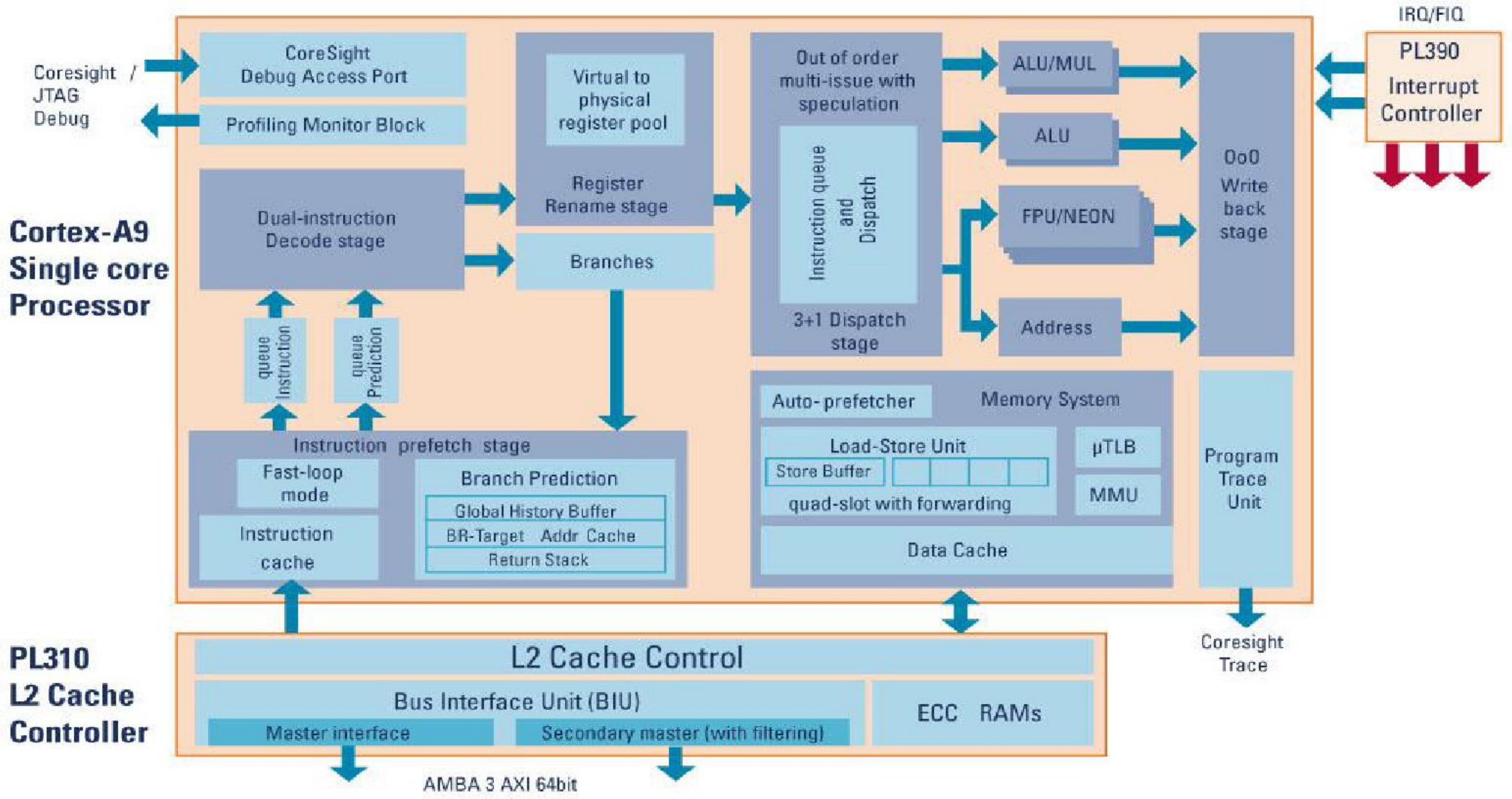


Prefetching Data Cache



From Lec #9 ...

Cortex-A9 Single Core Microarchitecture



Cortex-A9 Microarchitecture Structure and the Single Core Interfaces

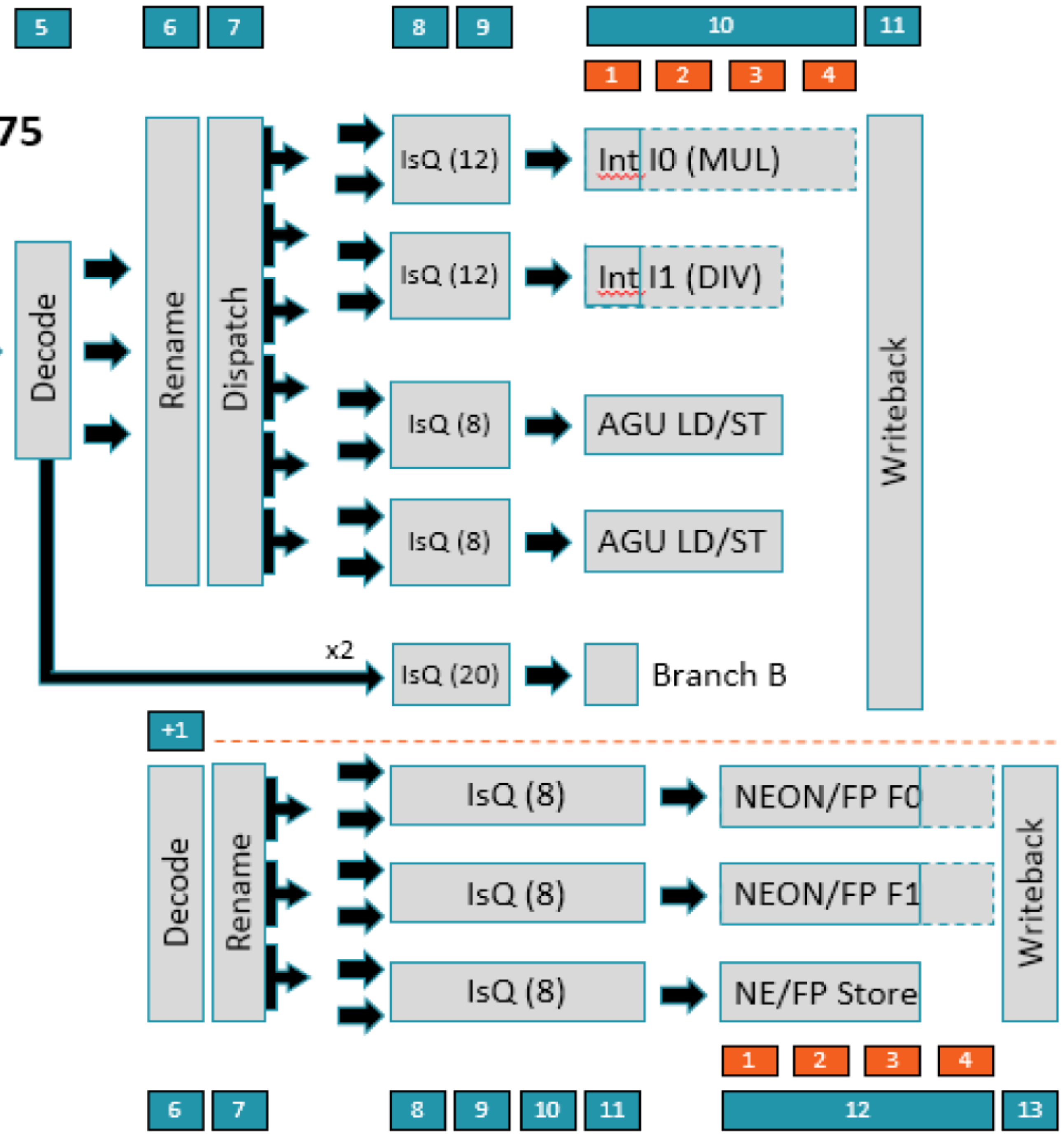
ARM Cortex-A75

1 2 3 4

ARM Cortex-A75

ANANDTECH

Instruction Fetch



5 6 7 8 9 10 11

1 2 3 4

Decode

Rename

Dispatch

IsQ (12) → Int I0 (MUL)

IsQ (12) → Int I1 (DIV)

IsQ (8) → AGU LD/ST

IsQ (8) → AGU LD/ST

x2

IsQ (20) → Branch B

Writeback

+1

Decode

Rename

IsQ (8) → NEON/FP F0

IsQ (8) → NEON/FP F1

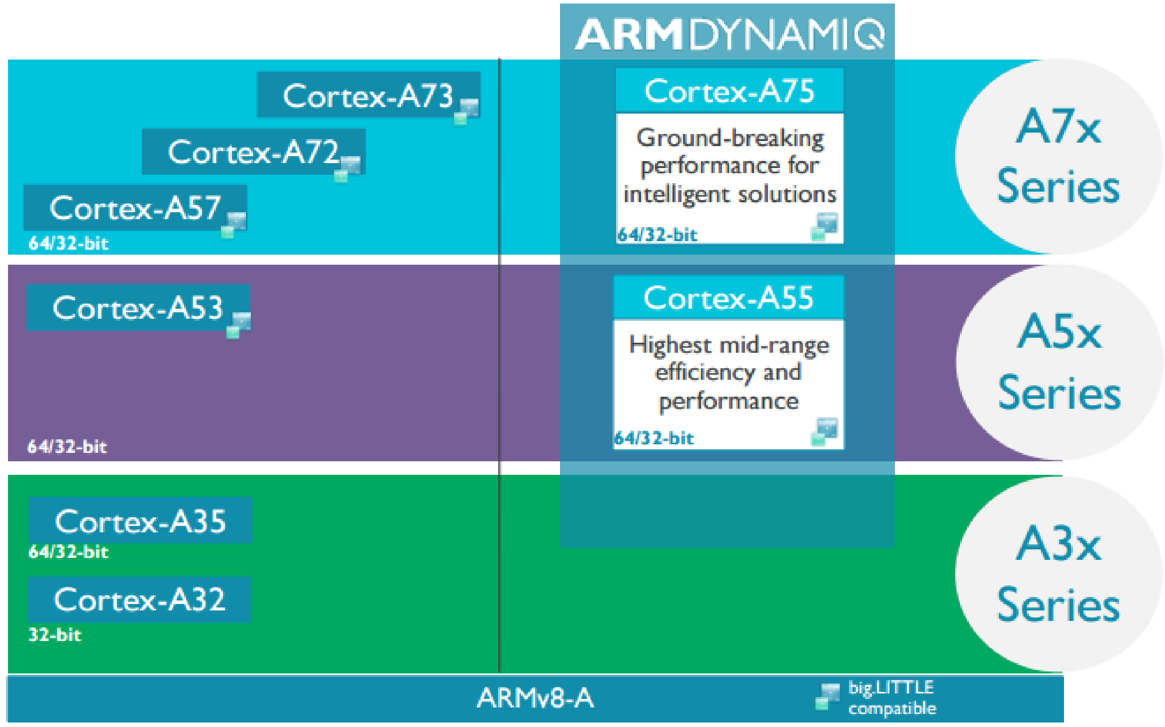
IsQ (8) → NE/FP Store

Writeback

1 2 3 4

6 7 8 9 10 11 12 13

ARM Cortex-A ARMv8 portfolio



ARMDYNAMIQ

Cortex-A73

Cortex-A72

Cortex-A57

64/32-bit

Cortex-A75

Ground-breaking performance for intelligent solutions

64/32-bit

A7x Series

Cortex-A53

64/32-bit

Cortex-A55

Highest mid-range efficiency and performance

64/32-bit

A5x Series

Cortex-A35

64/32-bit

Cortex-A32

32-bit

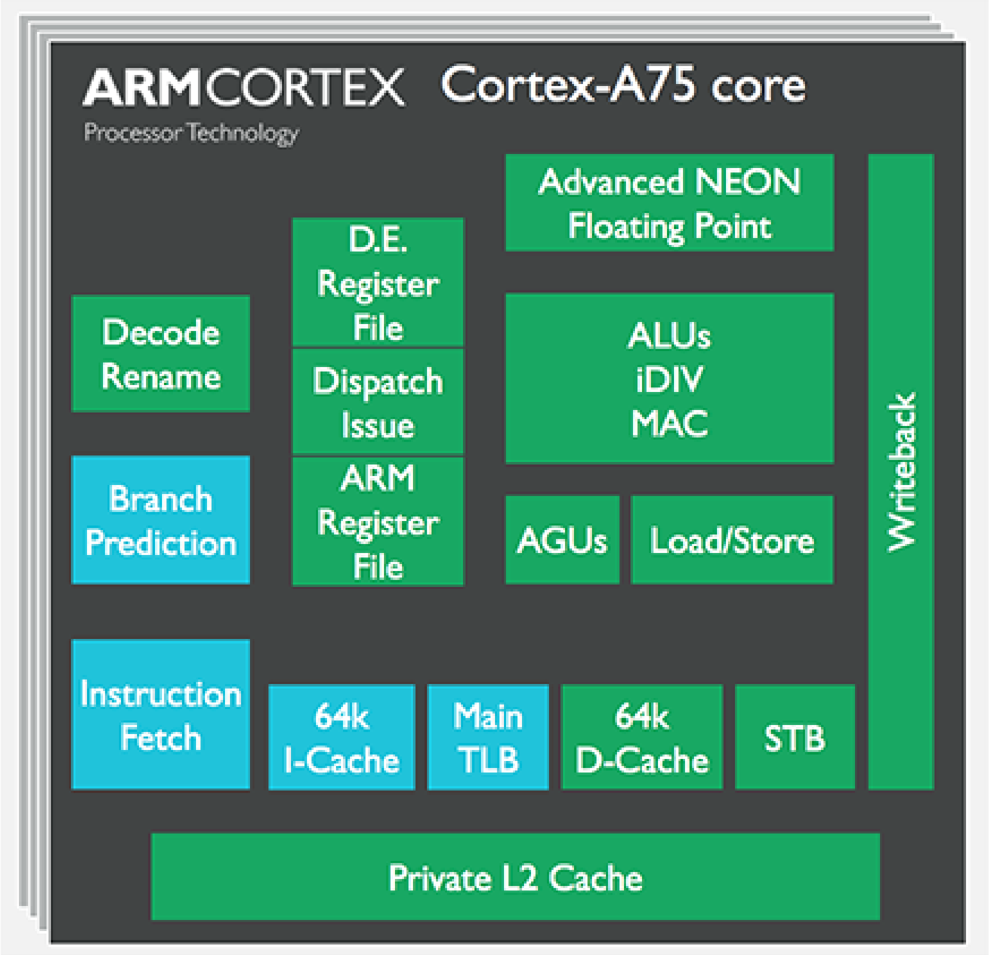
A3x Series

ARMv8-A

big.LITTLE compatible

State-of-the-art branch prediction

- Fine tuned 0-cycle branch prediction for better IPC
  - Further optimized from Cortex-A73
  - Sustains required instruction bandwidth to the core even on tight loops
- Resources for micro-BTACs, return stack and static branch predictors unchanged
  - Sustains additional performance required by Cortex-A75

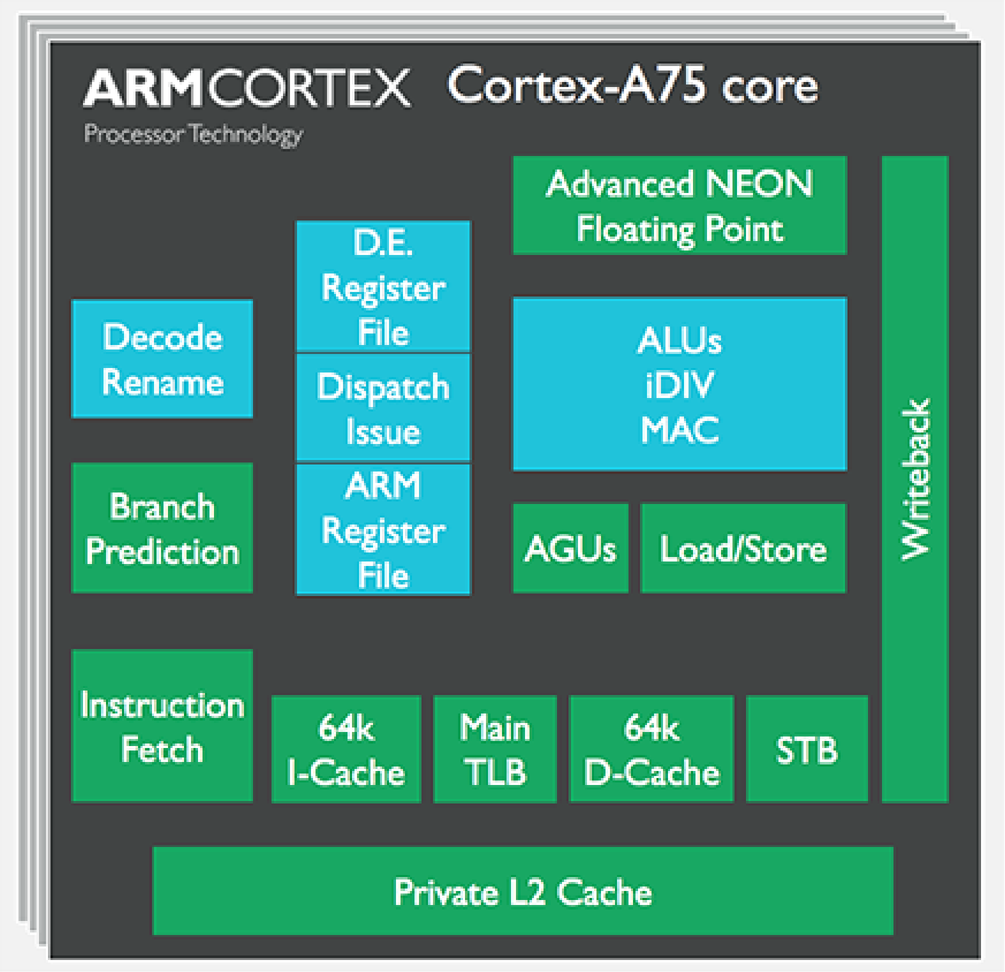


©ARM 2017

ARM

High-performance processor core

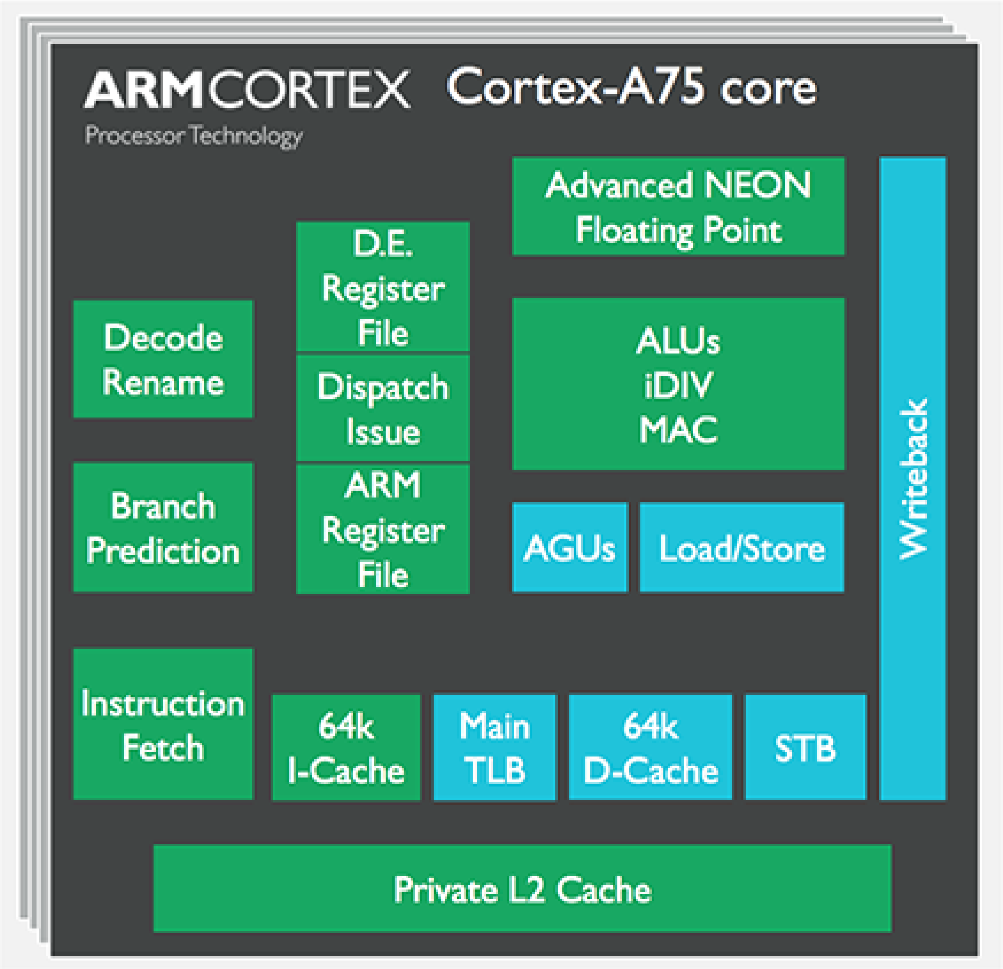
- 3-way superscalar high-performance pipeline
  - Single cycle decode with instruction fusing and micro-ops
- 7 independent high-performance issue queues
  - 2x Load/Store, 2x NEON/FPU, 1x Branch and 2x Integer core
- Increased capacity to sustain operation under L1 miss / L2 hit
  - 12 entries for integer core to maximise on in-flight instructions and out-of-order capabilities
  - 8 entries for Load/Store and NEON/FPU



ARM

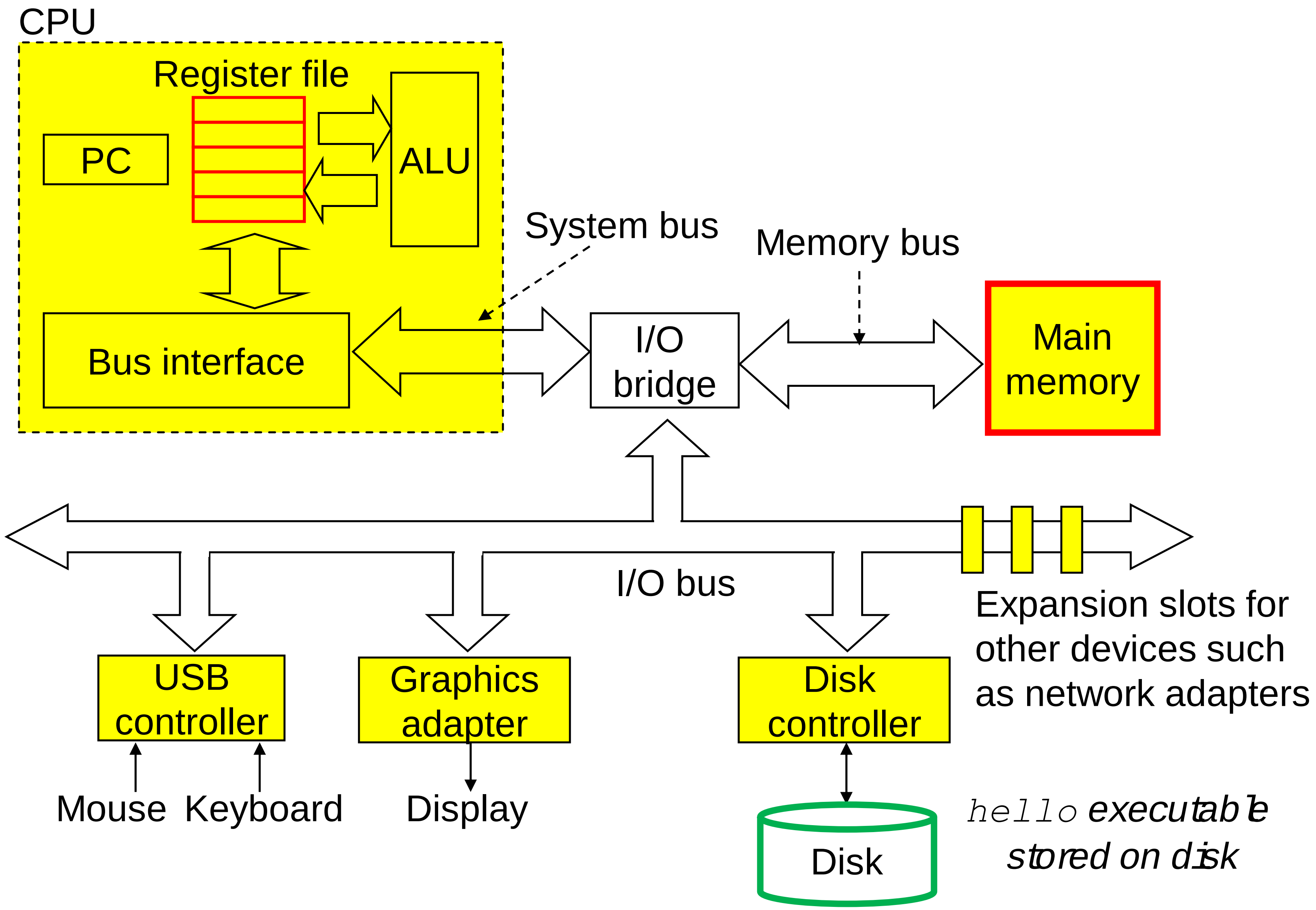
High-throughput data path

- L1 D-Cache, 64KB, 4-way set associative
  - VIPT with PIPT programmer's view
  - Load Store Unit extended to 16 slots
  - Increased core ↔ L1 cache bandwidth
- Aggressive Out-of-Order support
  - Support Read-after-Write OoO with filtering

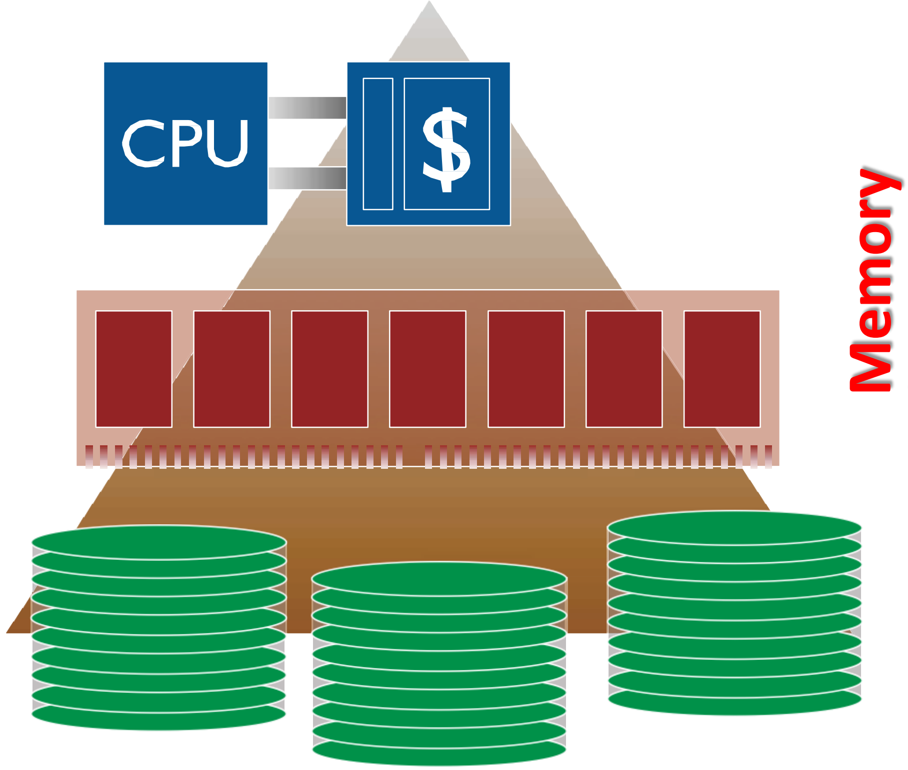


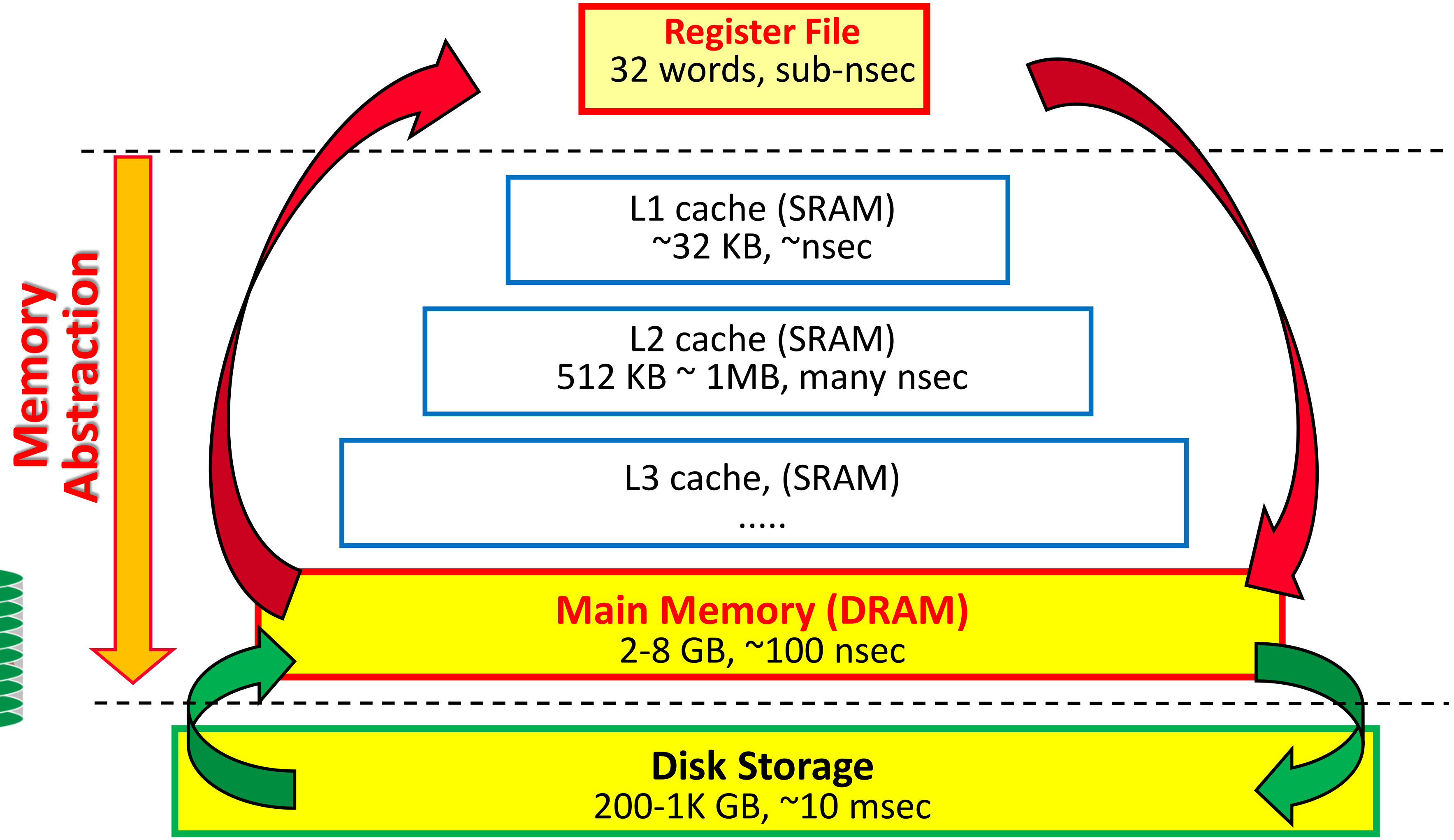
From Lec #1…

Typical Computer Organization



Memory Hierarchy (where do all the bits live?)





Memory Abstraction

Register File
32 words, sub-nsec

L1 cache (SRAM)
~32 KB, ~nsec

L2 cache (SRAM)
512 KB ~ 1MB, many nsec

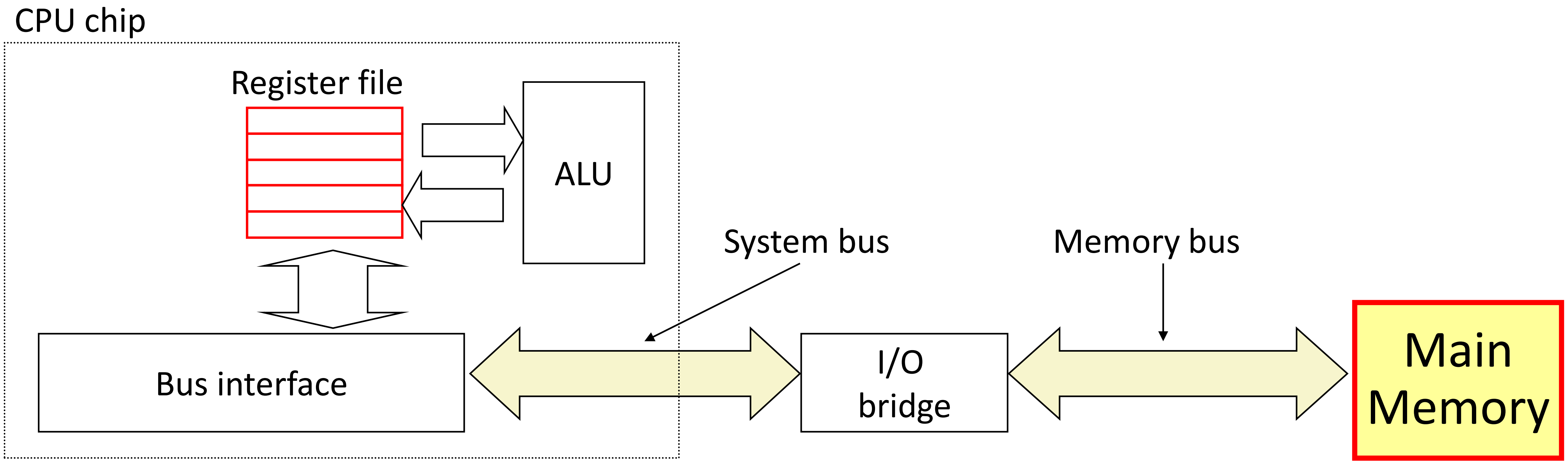
L3 cache, (SRAM)
.....

Main Memory (DRAM)
2-8 GB, ~100 nsec

Disk Storage
200-1K GB, ~10 msec

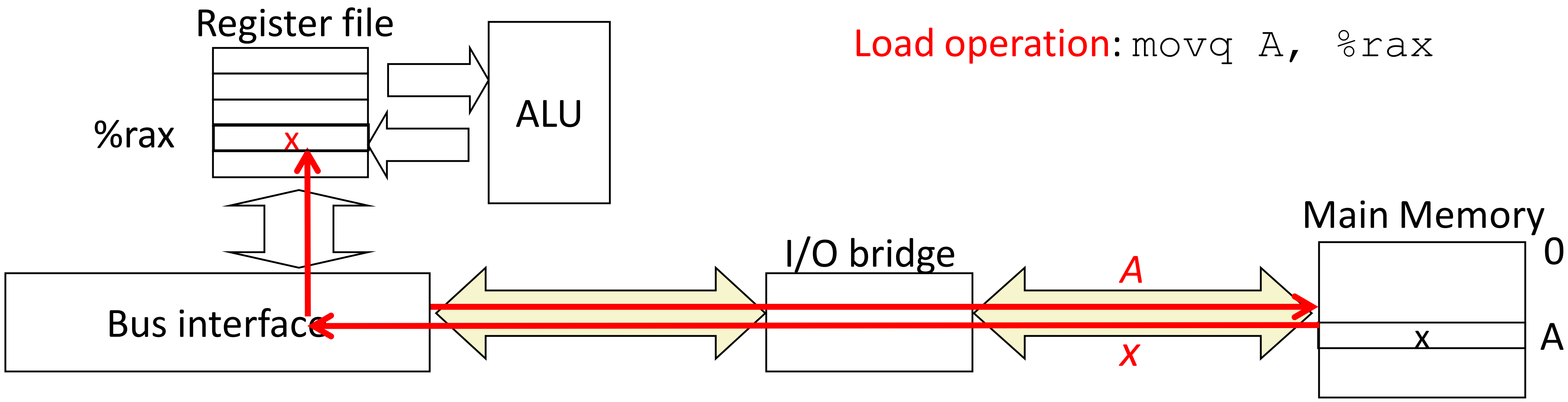
Bus Structure Connecting CPU and Memory

- A bus is a collection of parallel wires that carry address, data, and control signals.
- Buses are typically shared by multiple devices.

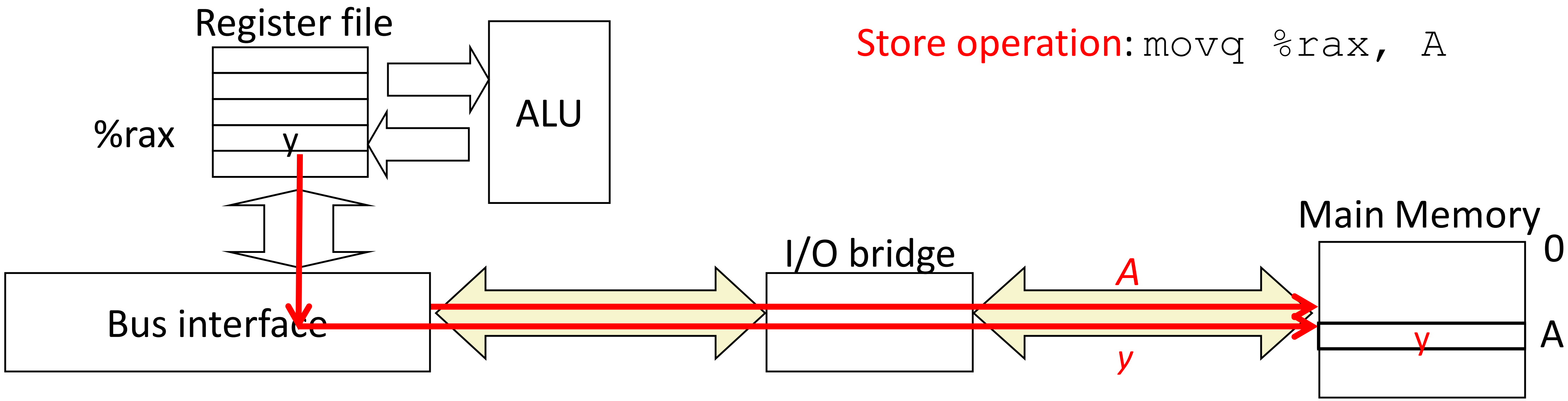


Memory Read & Write Transactions

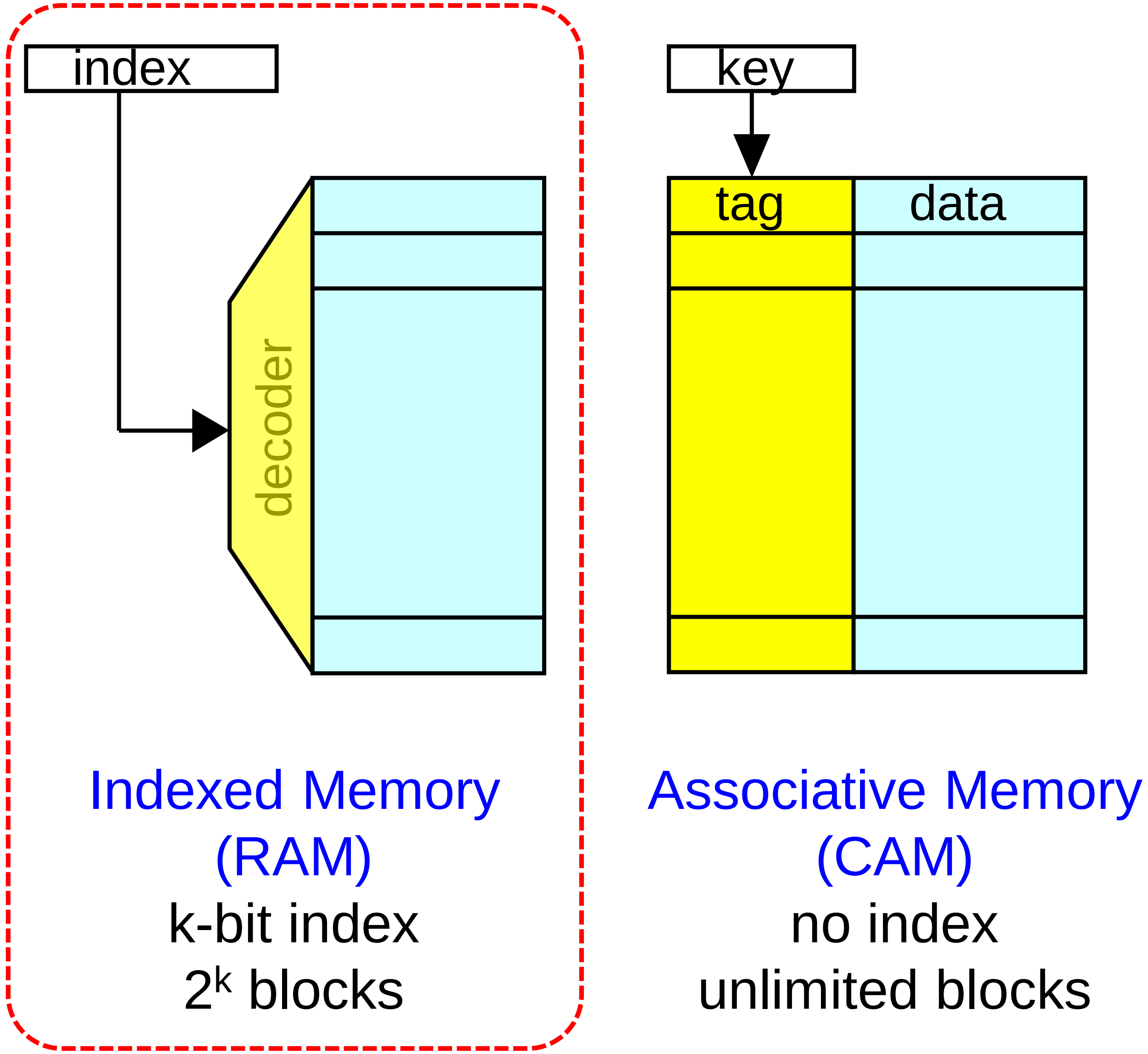
Load operation: `movq A, %rax`



Store operation: `movq %rax, A`



"Random Access" Memories (RAM)



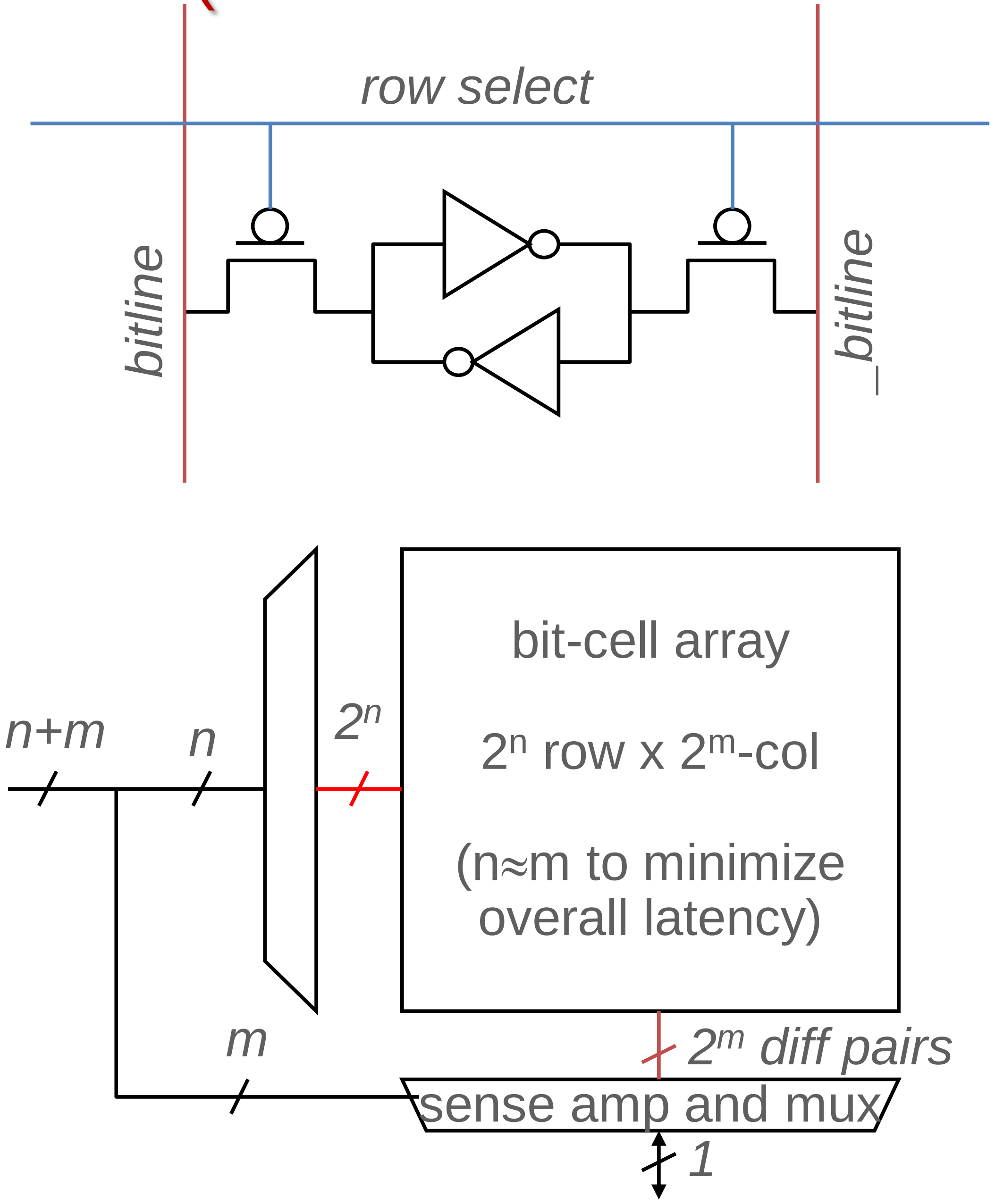
Indexed Memory (RAM)
k-bit index
2^k blocks

Associative Memory (CAM)
no index
unlimited blocks

- Key features
  - RAM is traditionally packaged as a chip.
  - Basic storage unit is normally a cell (one bit per cell).
  - Multiple RAM chips form a memory.

- RAM comes in two varieties:
  - SRAM (Static RAM)
  - DRAM (Dynamic RAM)

SRAM (Static Random Access Memory)



Read Sequence

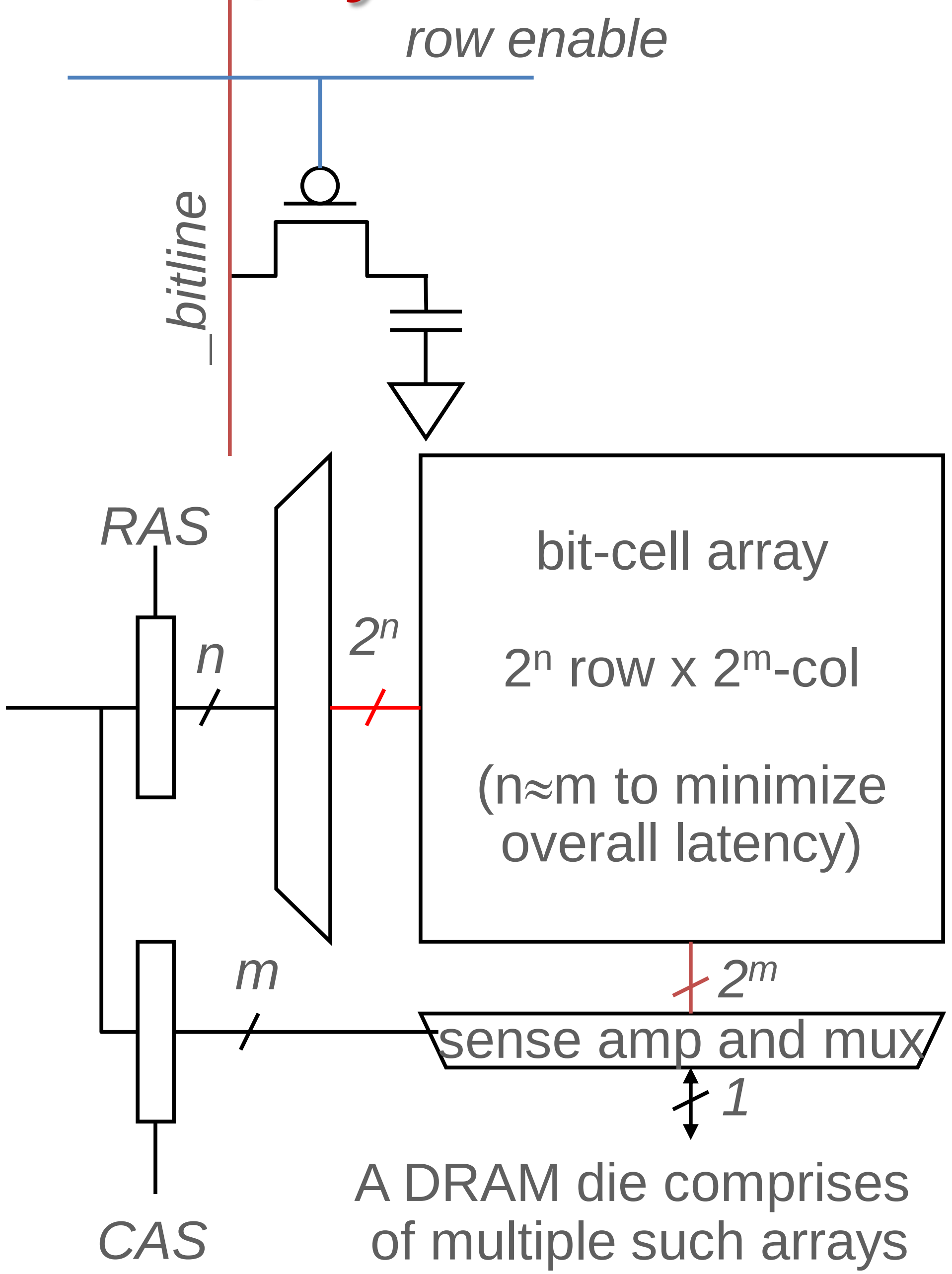
1. address decode
2. drive row select
3. selected bit-cells drive bitlines
   (entire row is read together)
4. diff. sensing and col. select
   (data is ready)
5. precharge all bitlines
   (for next read or write)

Access latency dominated by steps 2 and 3

Cycling time dominated by steps 2, 3 and 5

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

DRAM (Dynamic Random Access Memory)



A DRAM die comprises of multiple such arrays

Bits stored as charges on node capacitance (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

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

DRAM vs. SRAM

- DRAM (used in main memories)
  - Slower access (capacitor) ← 10x access time
  - Higher density (1T 1C cell)
  - Lower cost ← 1x cost
  - Requires refresh and READs are destructive
  - Manufacturing requires putting capacitor and logic together
- SRAM (used in cache memories)
  - Faster access (no capacitor) ← 1x access time
  - Lower density (6T cell)
  - Higher cost ← 100x cost
  - No need for refresh and non-destructive READs
  - Manufacturing compatible with logic process (no capacitor)

18-600 Foundations of Computer Systems

Lecture 10:
"The Memory Hierarchy"

A. Memory Technologies

B. Main Memory Implementation

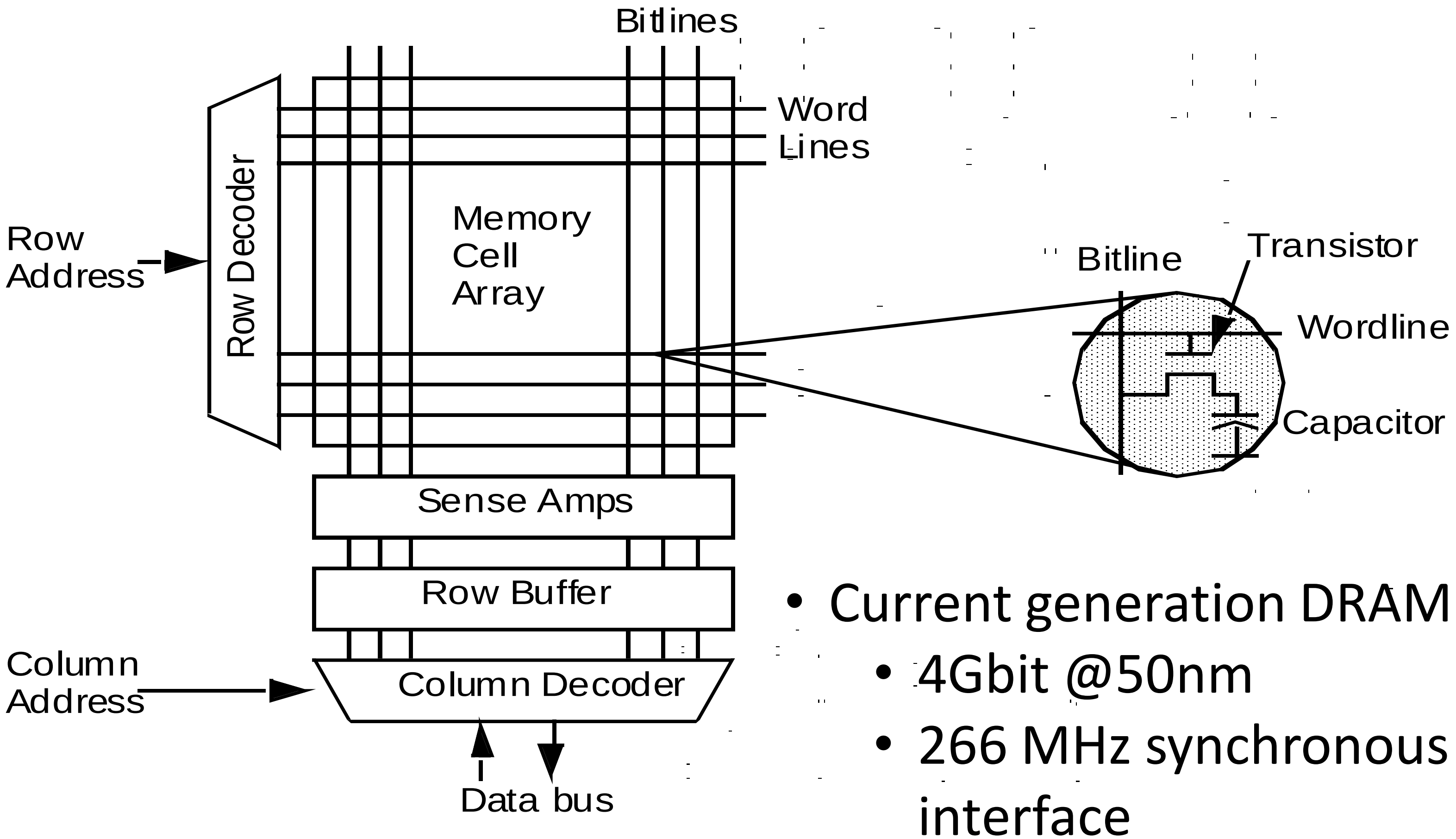
  a. DRAM Organization

  b. DRAM Operation

  c. Memory Controller

C. Disk Storage Technologies

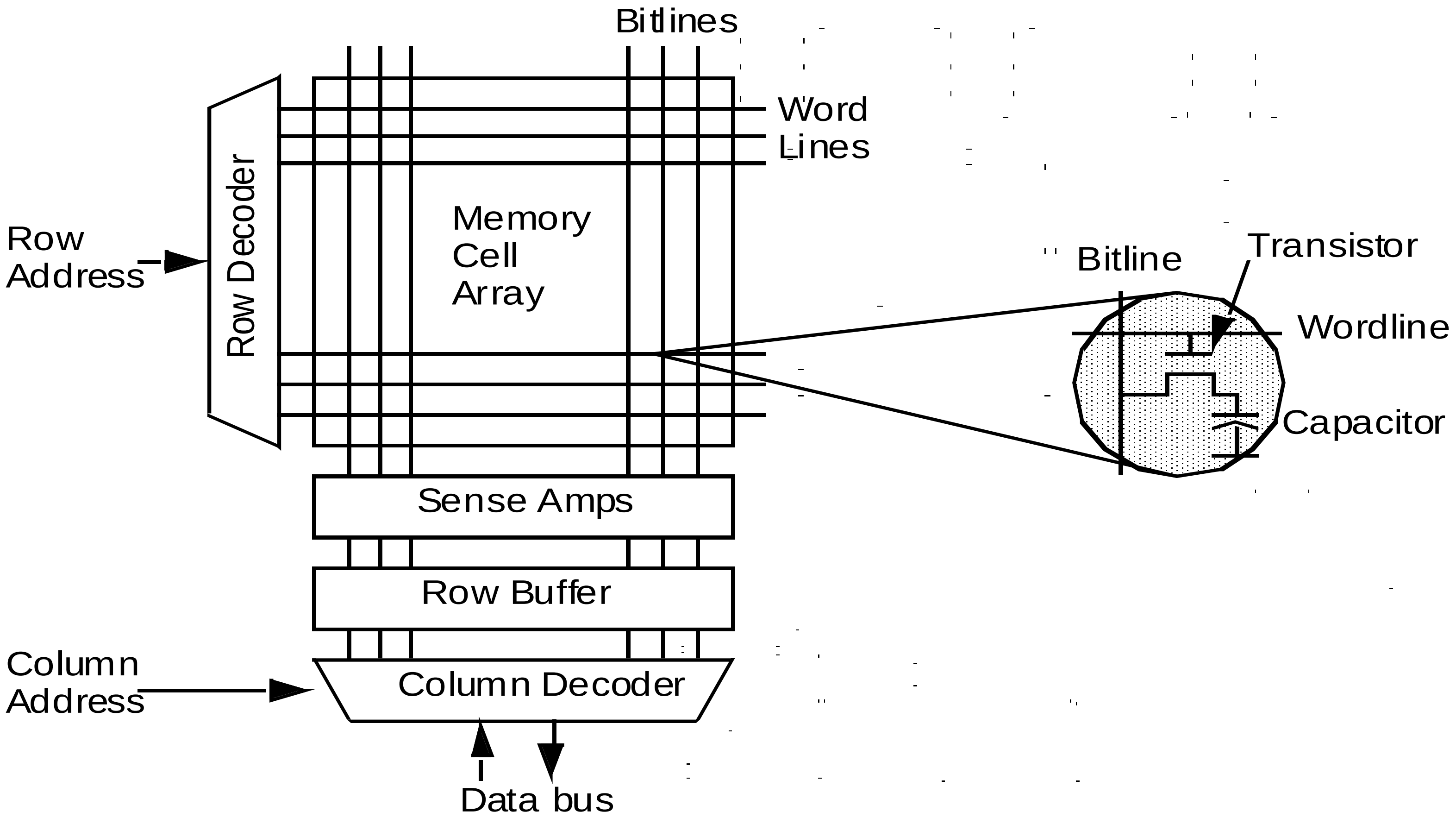
DRAM Chip Organization



- Current generation DRAM
  - 4Gbit @50nm
  - 266 MHz synchronous interface
  - Peak "double data rate" 2x4x memory clock

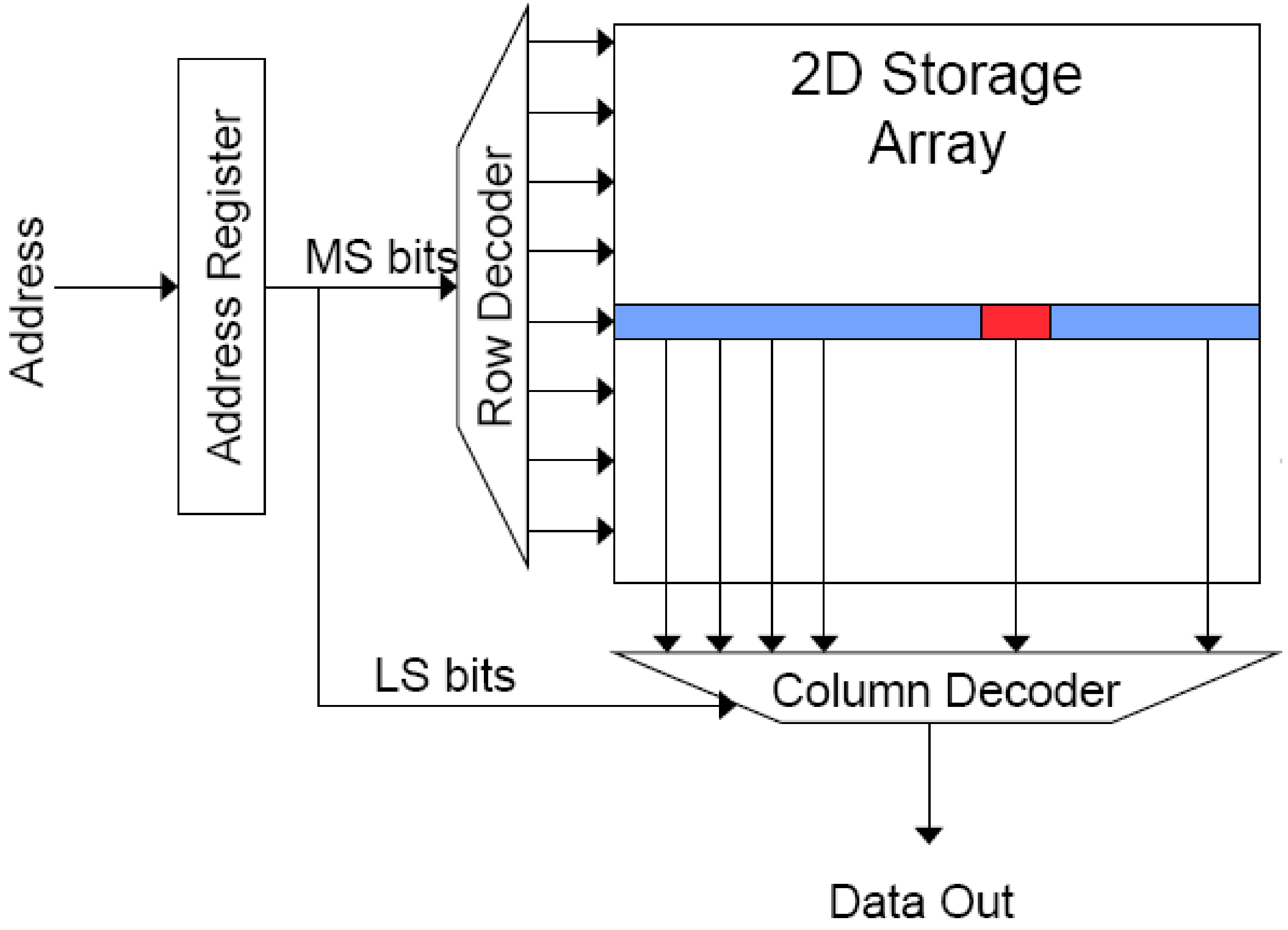
- Optimized for density, not speed
- Data stored as charge in capacitor
- Discharge on reads => destructive reads
- Charge leaks over time
  - refresh every 64ms
- Cycle time roughly twice access time
- Need to precharge bitlines before access

DRAM Chip Organization



- Address pins are time-multiplexed
  - Row address strobe (RAS)
  - Column address strobe (CAS)
- New RAS results in:
  - Bitline precharge
  - Row decode, sense
  - Row buffer write (up to 8K)
- New CAS
  - Read from row buffer
  - Much faster (3x)
- Streaming row accesses desirable

DRAM Bank Organization

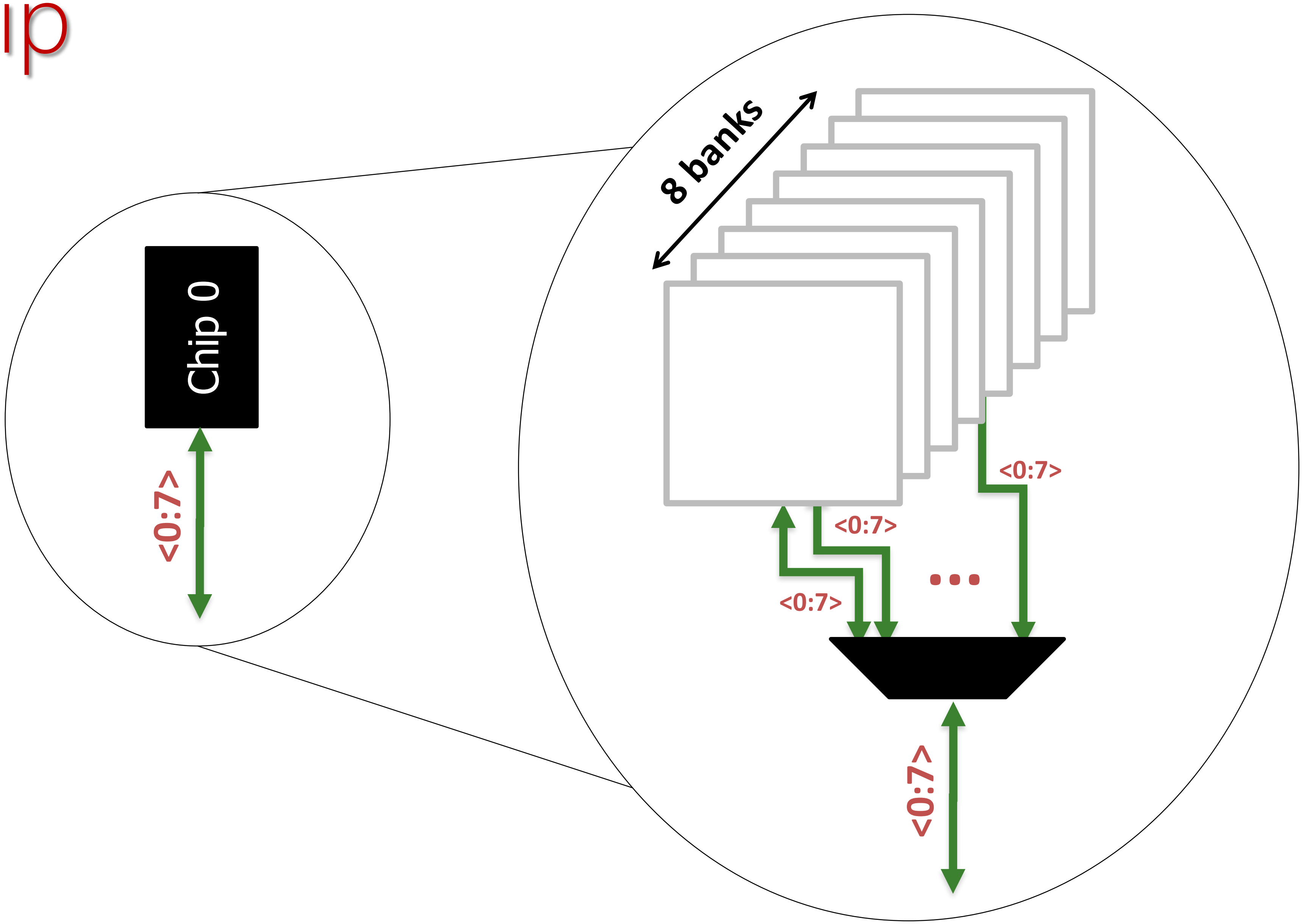


- Read access sequence:

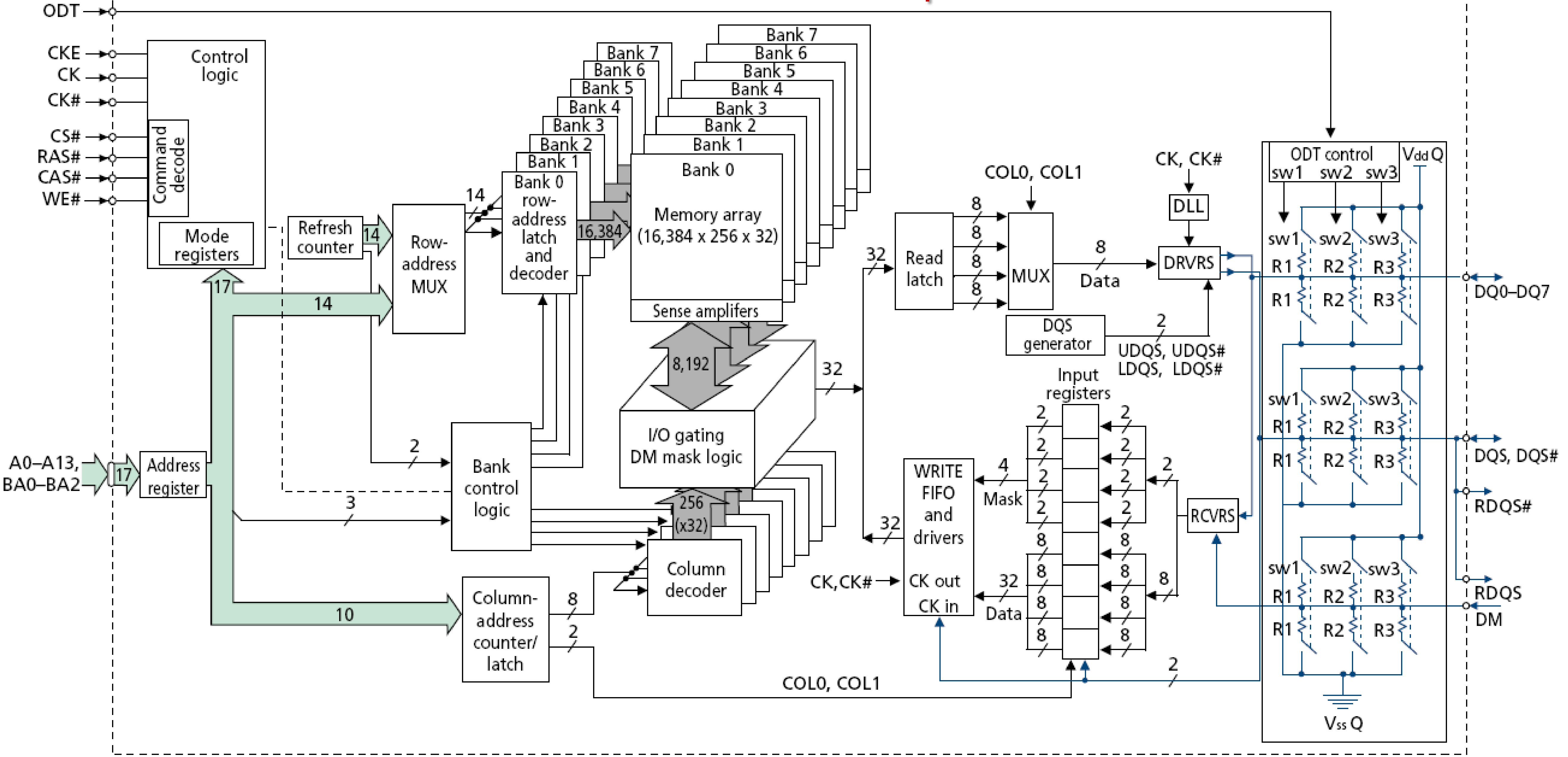
  1. Decode row address & drive word-line
  2. Selected bits drive bit-lines
     - Entire row read
  3. Amplify row data
  4. Decode column address & select subset of row
     - Send to output
  5. Precharge bit-lines
     - For next access

The DRAM Chip

- Consists of multiple banks (2-16 in Synchronous DRAM)
- Banks share command/address/ data buses
- The chip itself has a narrow interface (4-16 bits per read)



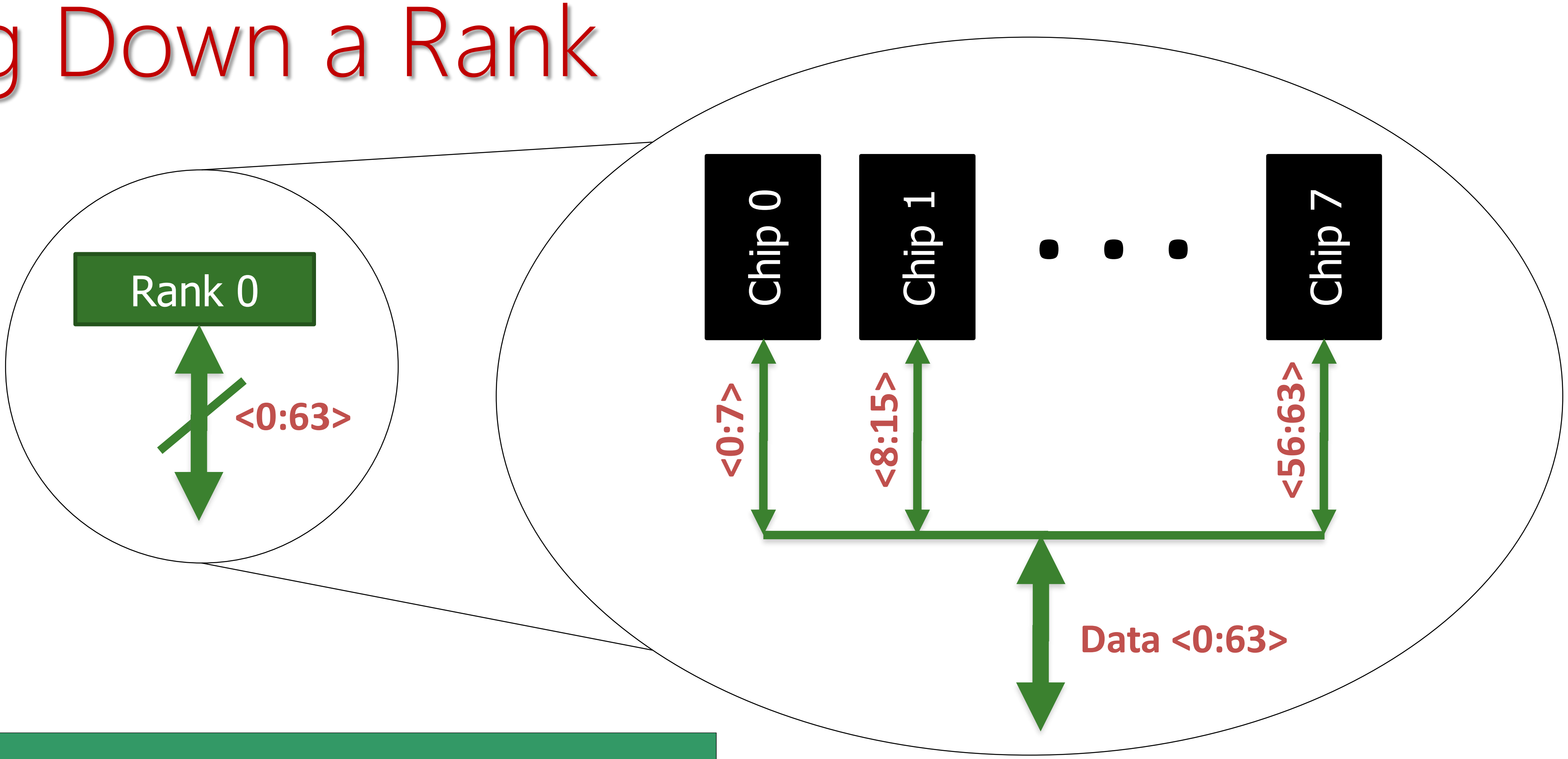
128M x 8-bit DRAM Chip

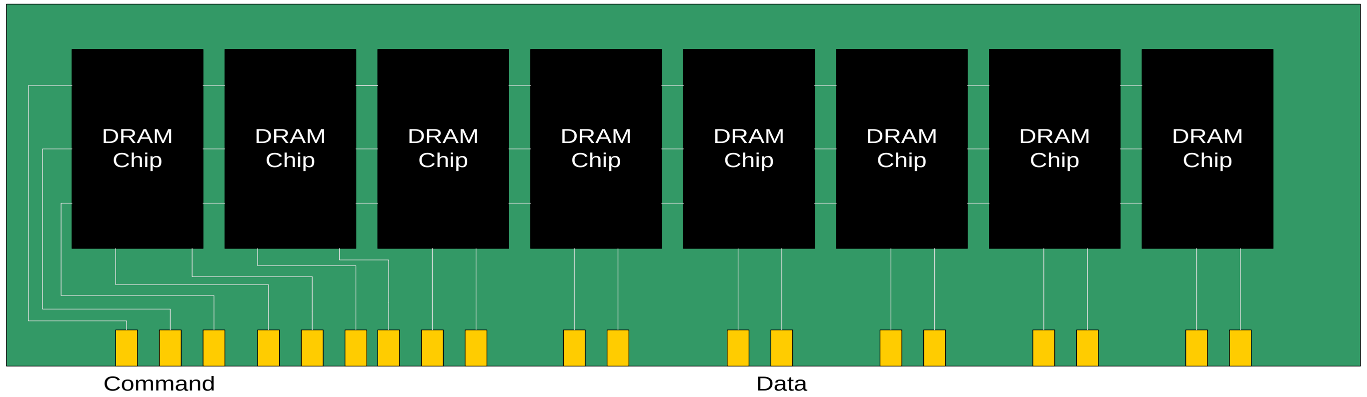


DRAM Rank and Module

- Rank: Multiple chips operated together to form a wide interface
- All chips comprising a rank are controlled at the same time
  - Respond to a single command
  - Share address and command buses, but provide different data
- A DRAM module consists of one or more ranks
  - E.g., DIMM (dual inline memory module)
  - This is what you plug into your motherboard
- If we have chips with 8-bit interface, to read 8 bytes in a single access, use 8 chips in a DIMM

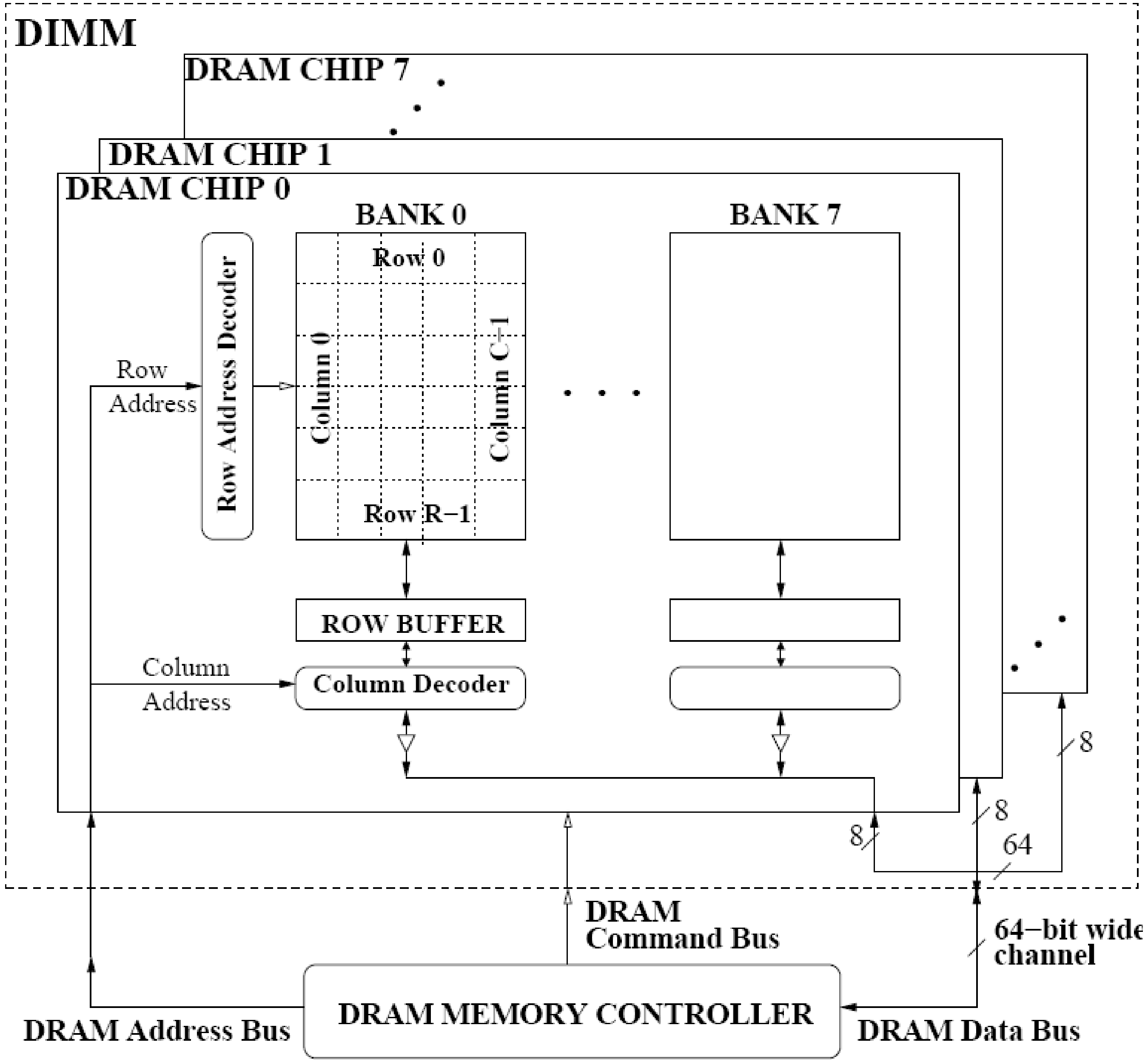
Breaking Down a Rank





A 64-bit Wide DIMM (One Rank)

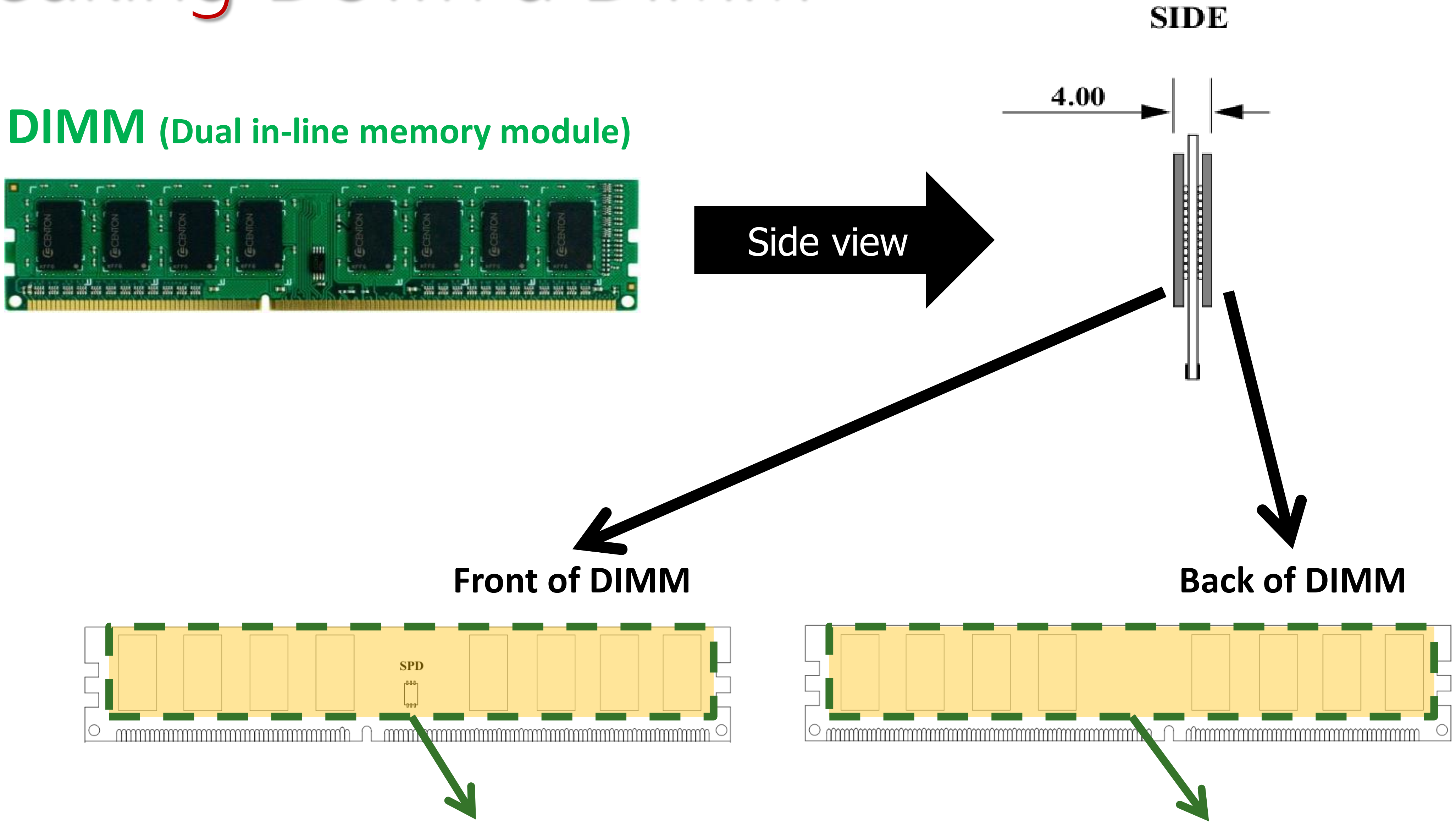
A 64-bit Wide DIMM (One Rank)



- Advantages:
  - Acts like a high-capacity DRAM chip with a wide interface
  - Flexibility: memory controller does not need to deal with individual chips
- Disadvantages:
  - Granularity: Accesses cannot be smaller than the interface width

Breaking Down a DIMM

DIMM **(Dual in-line memory module)**



Side view

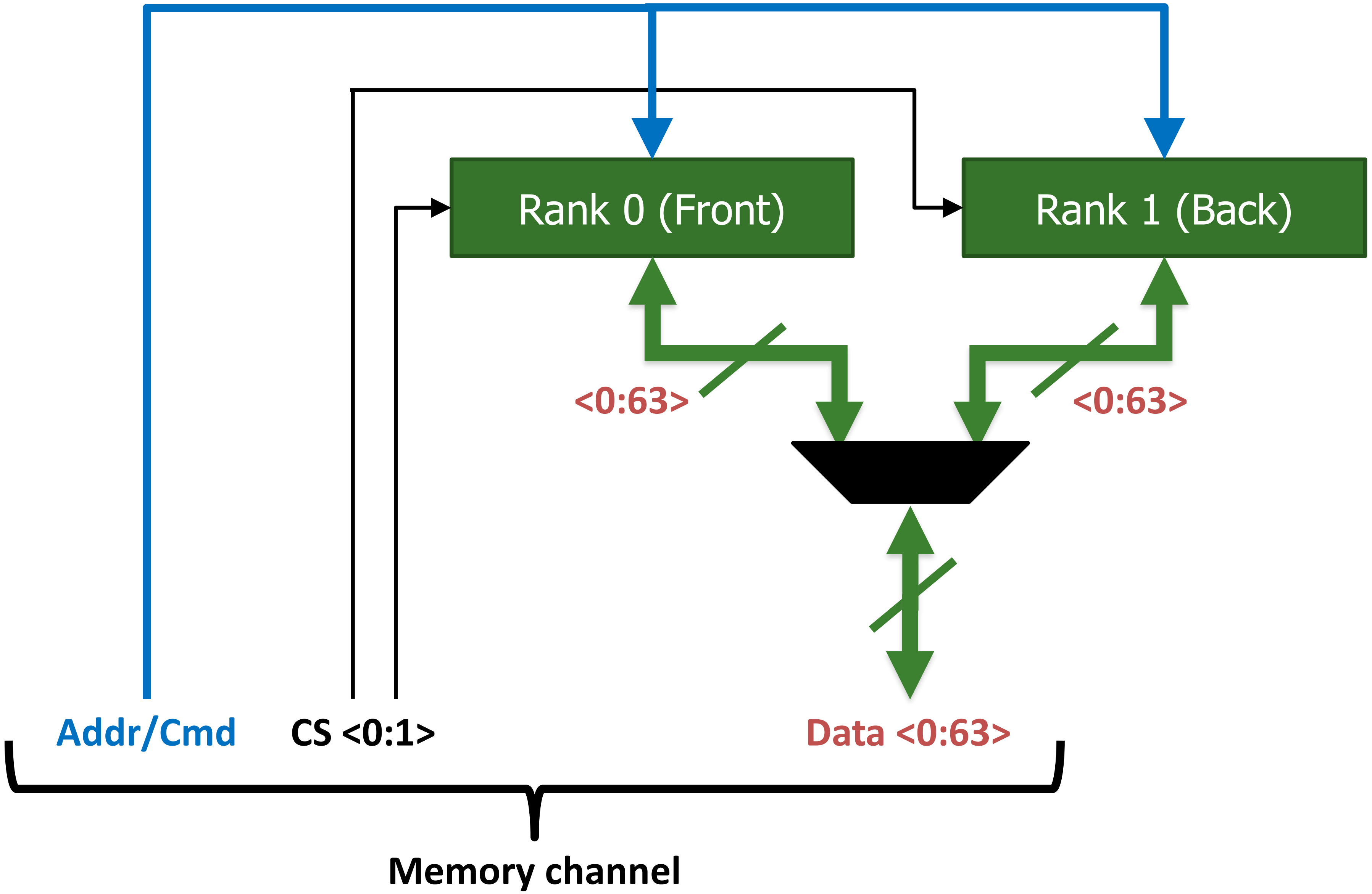
Front of DIMM

Back of DIMM

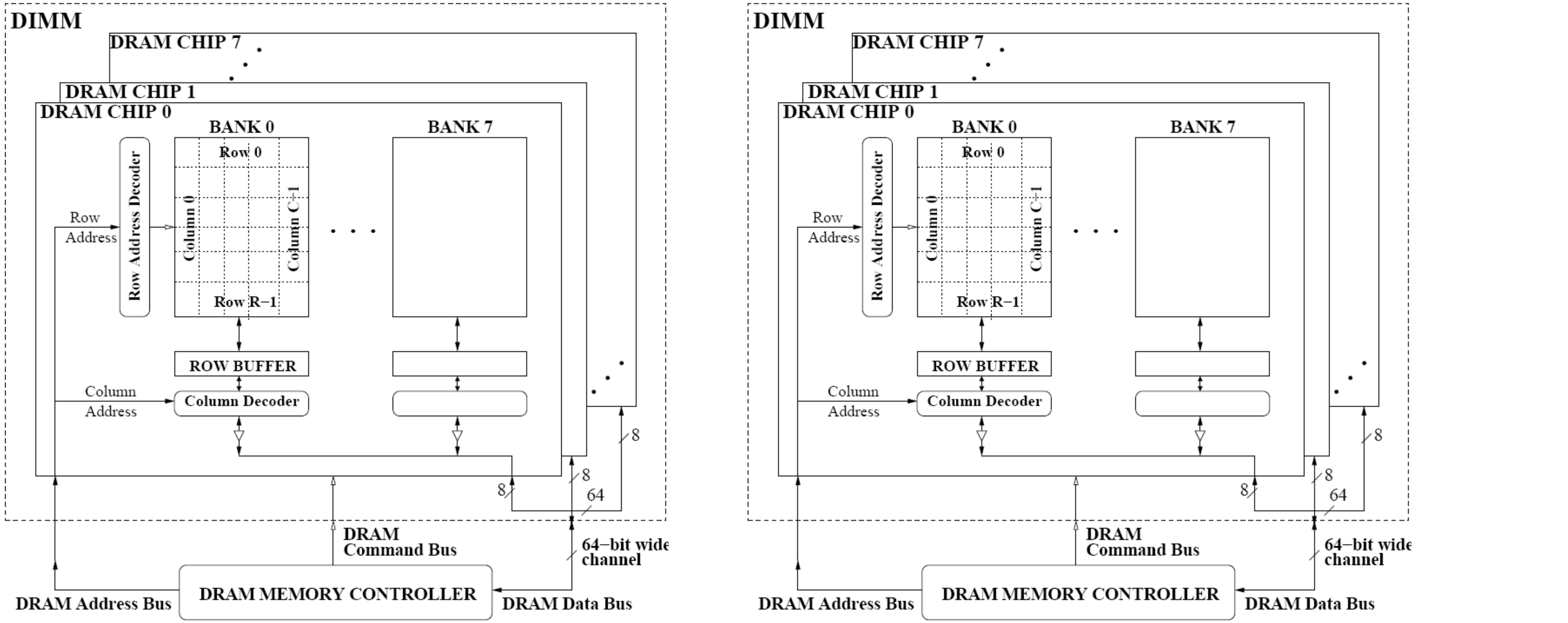
Rank 0: collection of 8 chips

Rank 1

Rank and Channel



DRAM Channels



- 2 Independent Channels: 2 Memory Controllers (Above)
- 2 Dependent/Lockstep Channels: 1 Memory Controller with wide interface (Not Shown above)

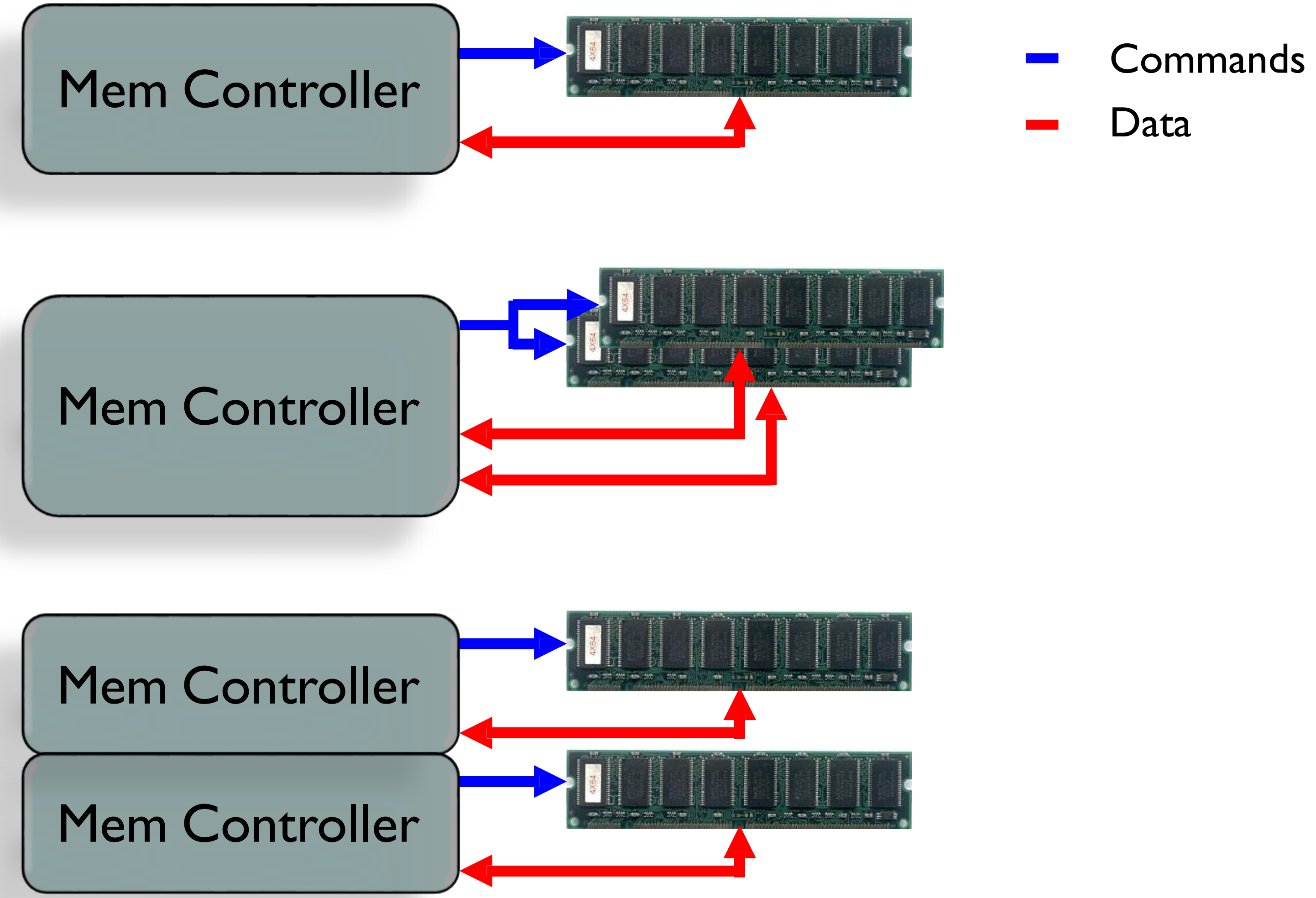
[Nima Honarmand, Stony Brook University, Spring 2015 (CSE 502)]

Channel and Controller

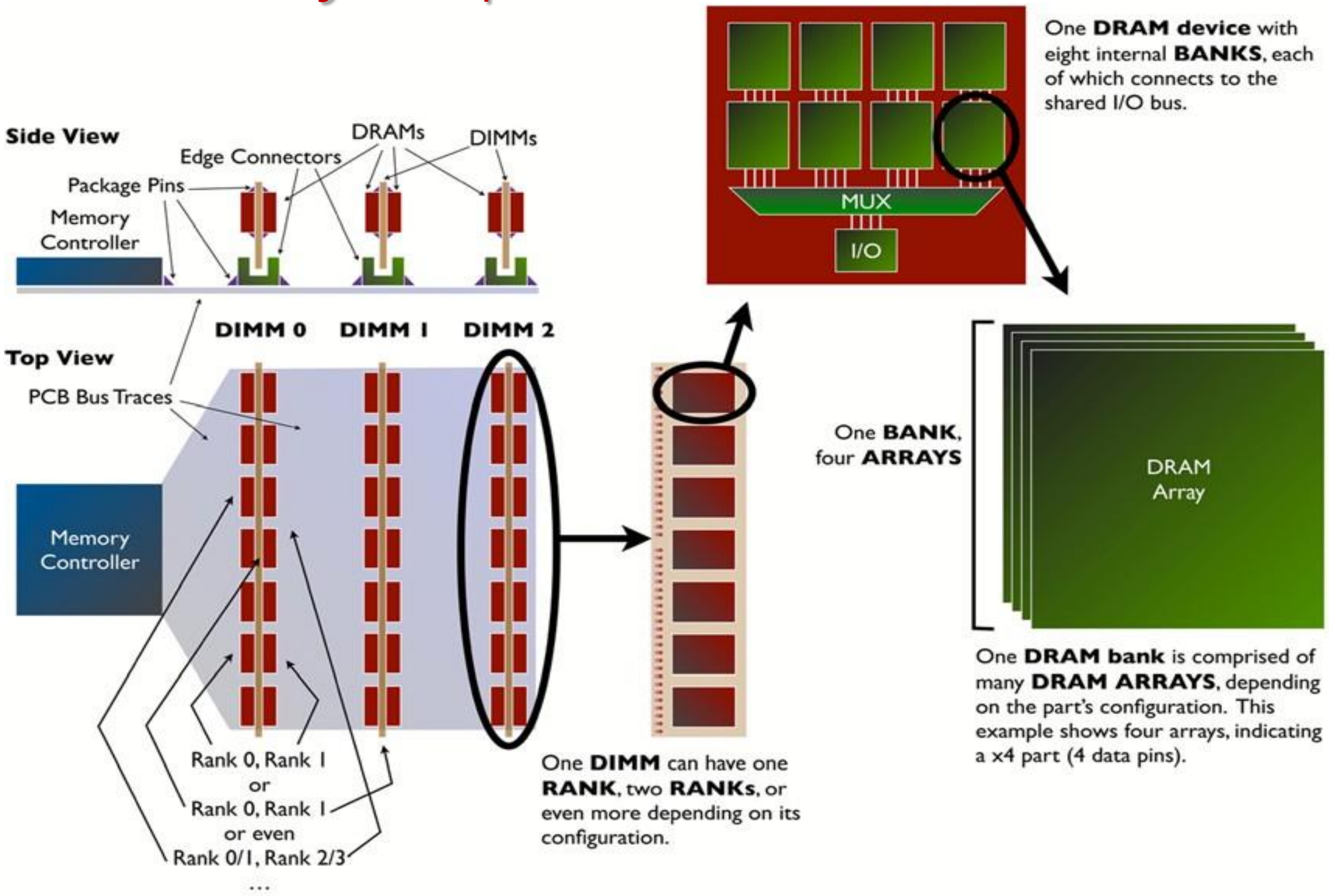
One controller
One 64-bit channel

One controller
Two 64-bit channels

Two controllers
Two 64-bit channels



Main Memory Implementation



One **DRAM device** with eight internal **BANKS**, each of which connects to the shared I/O bus.

One **BANK**, four **ARRAYS**

One **DIMM** can have one **RANK**, two **RANKs**, or even more depending on its configuration.

One **DRAM bank** is comprised of many **DRAM ARRAYS**, depending on the part's configuration. This example shows four arrays, indicating a x4 part (4 data pins).

18-600 Foundations of Computer Systems

Lecture 10:
"The Memory Hierarchy"

A. Memory Technologies
B. Main Memory Implementation
    a. DRAM Organization
    b. DRAM Operation
    c. Memory Controller
C. Disk Storage Technologies

Electrical & Computer ENGINEERING

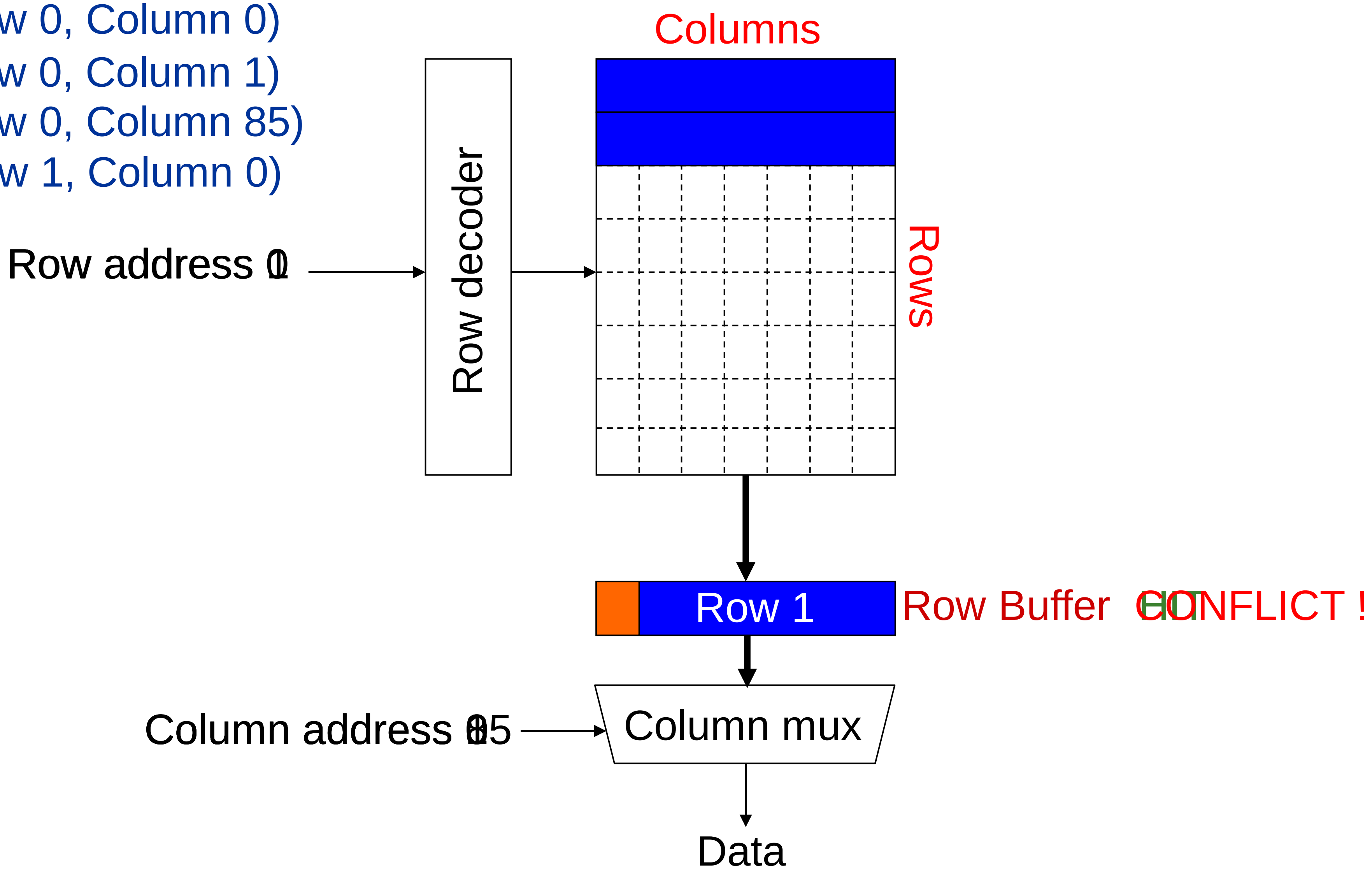
Page Mode DRAM

- A DRAM bank is a 2D array of cells: rows x columns
- A “DRAM row” is also called a “DRAM page”
- “Sense amplifiers” also called “row buffer”

- Each address is a <row,column> pair
- Access to a “closed row”
  - Activate command opens row (placed into row buffer)
  - Read/write command reads/writes column in the row buffer
  - Precharge command closes the row and prepares the bank for next access
- Access to an “open row”
  - No need for activate command

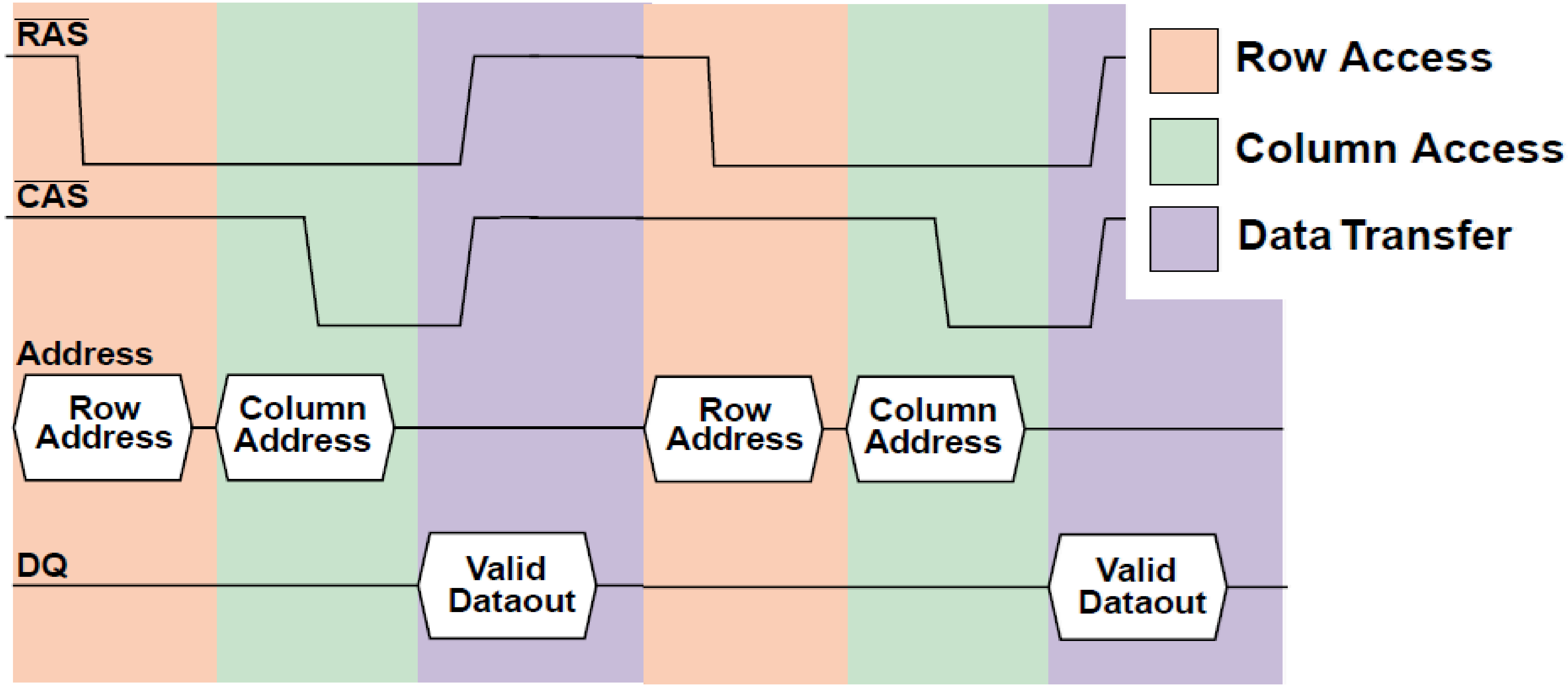
DRAM (Bank) Operation

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



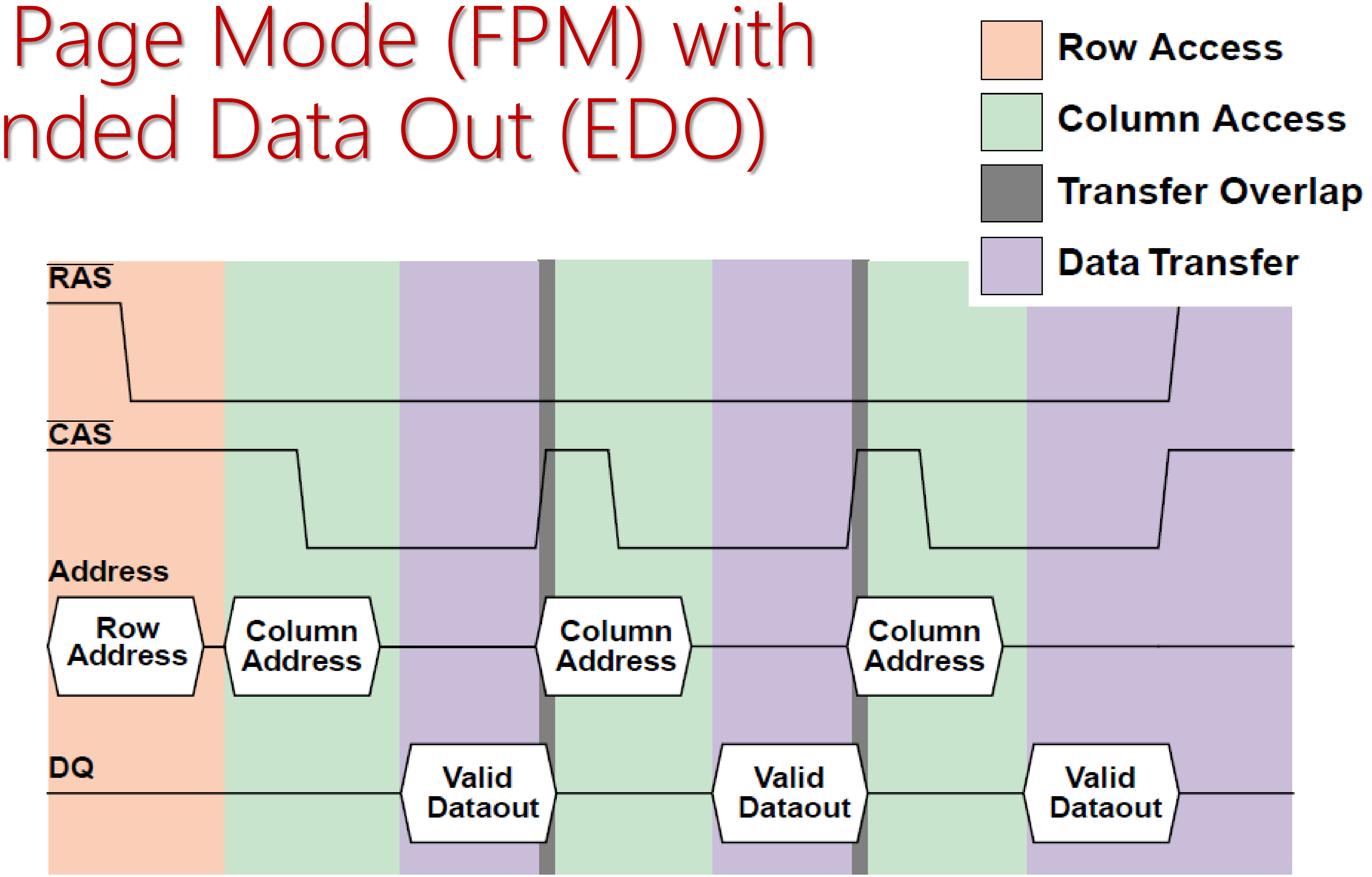
[Nima Honarmand, Stony Brook University, Spring 2015 (CSE 502)]

Original (Old) DRAM Read Timing

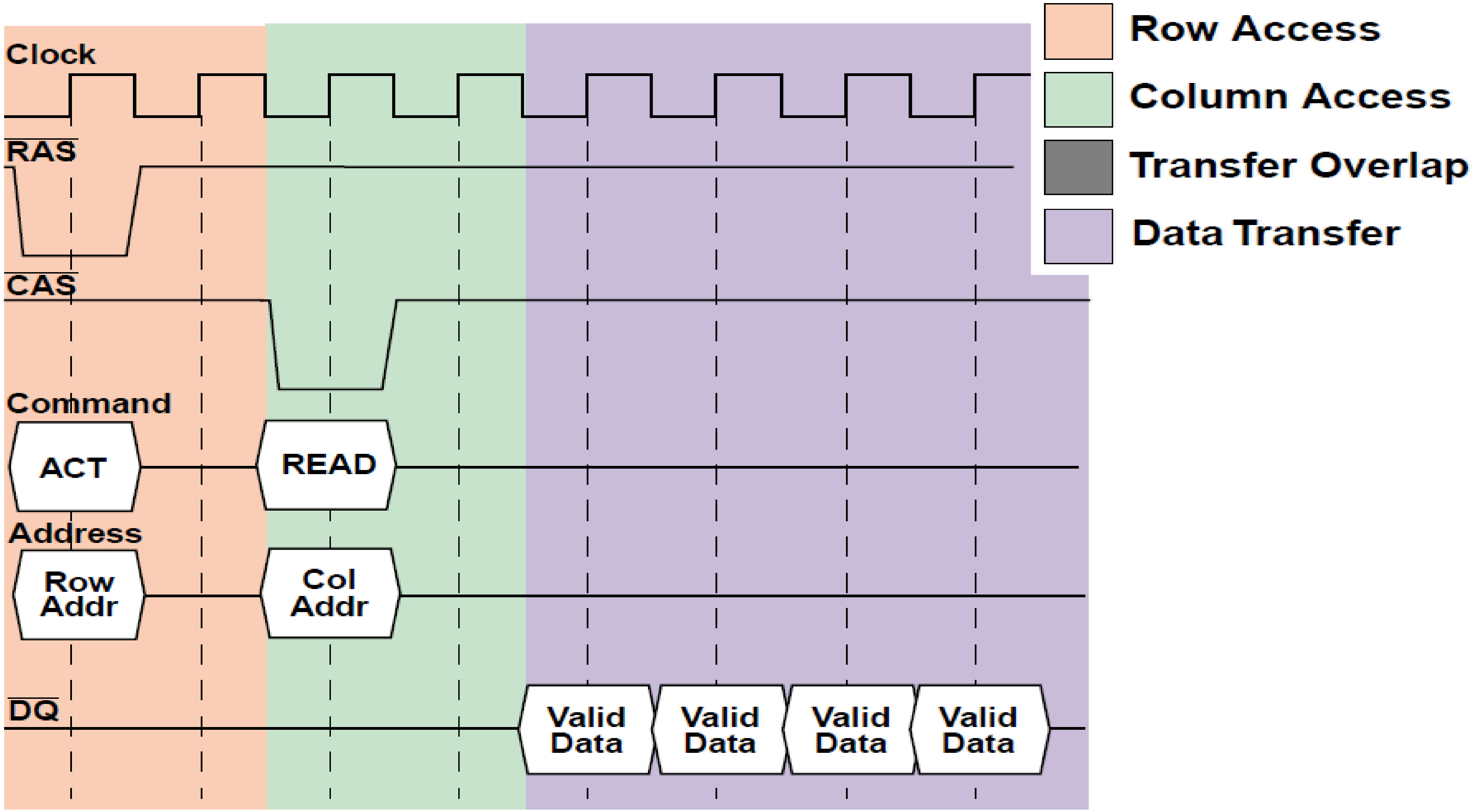


[Nima Honarmand, Stony Brook University, Spring 2015 (CSE 502)]

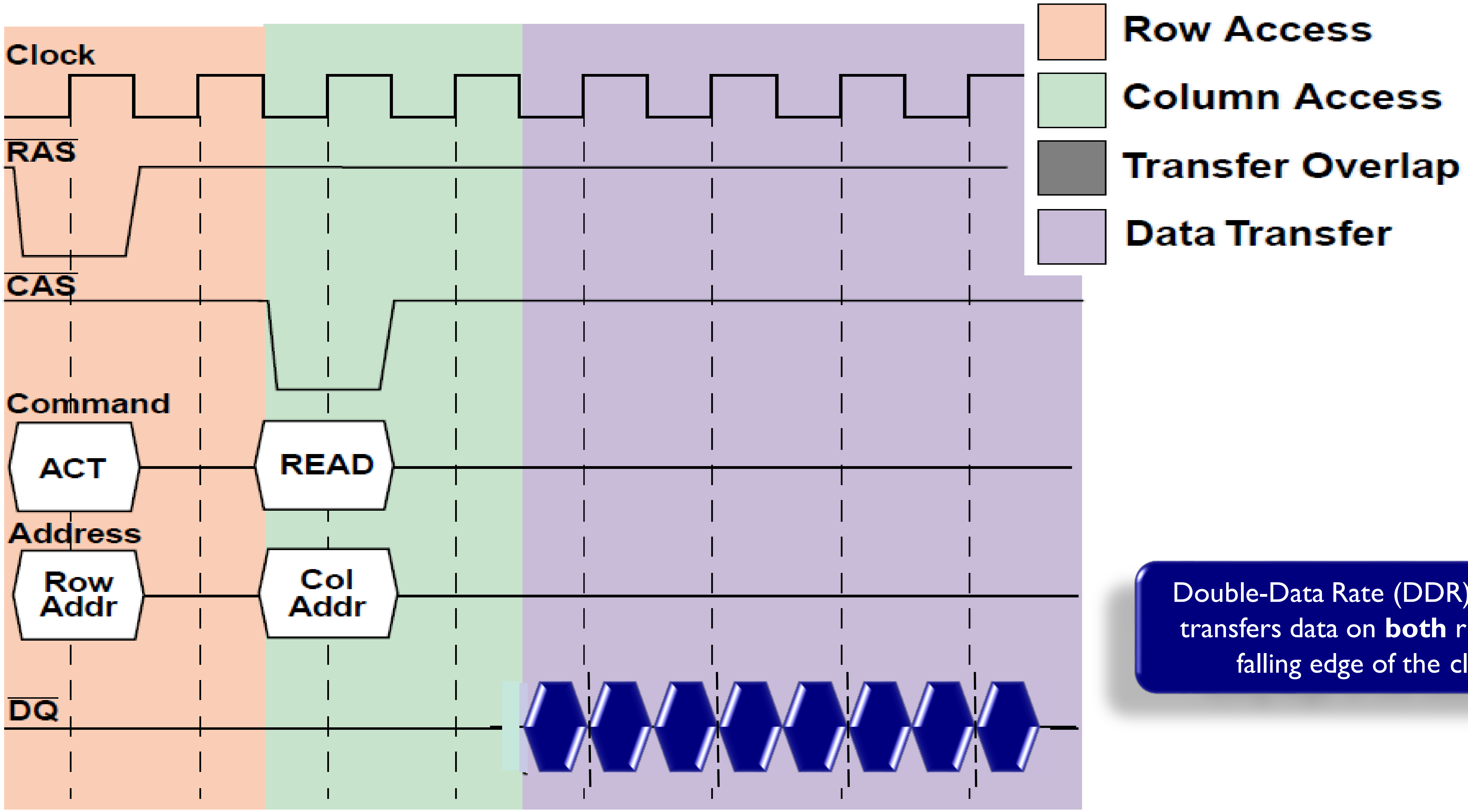
Fast Page Mode (FPM) with Extended Data Out (EDO)



Synchronous DRAM (SDRAM)



DDR SDRAM Timing

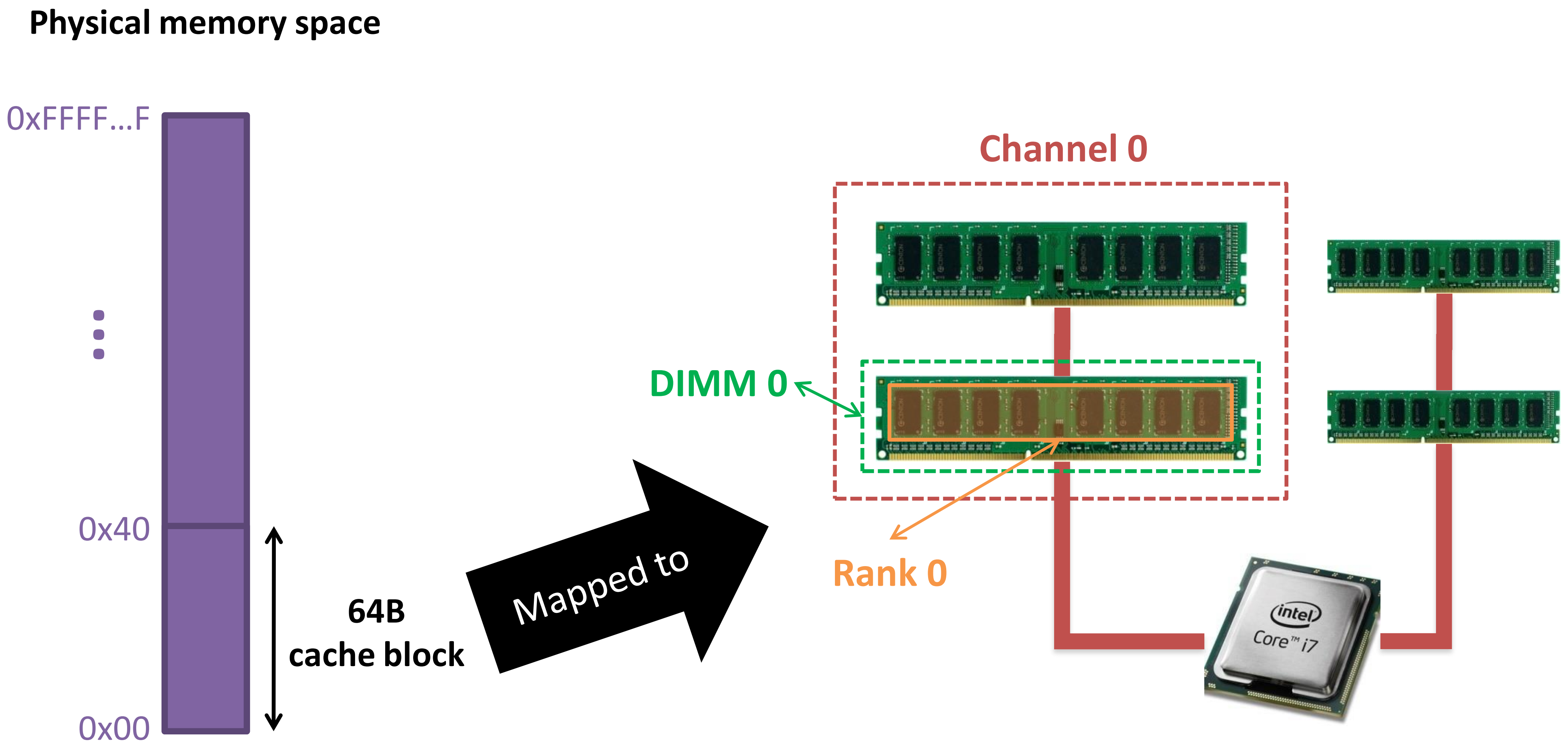


Double-Data Rate (DDR) SDRAM transfers data on **both** rising and falling edge of the clock

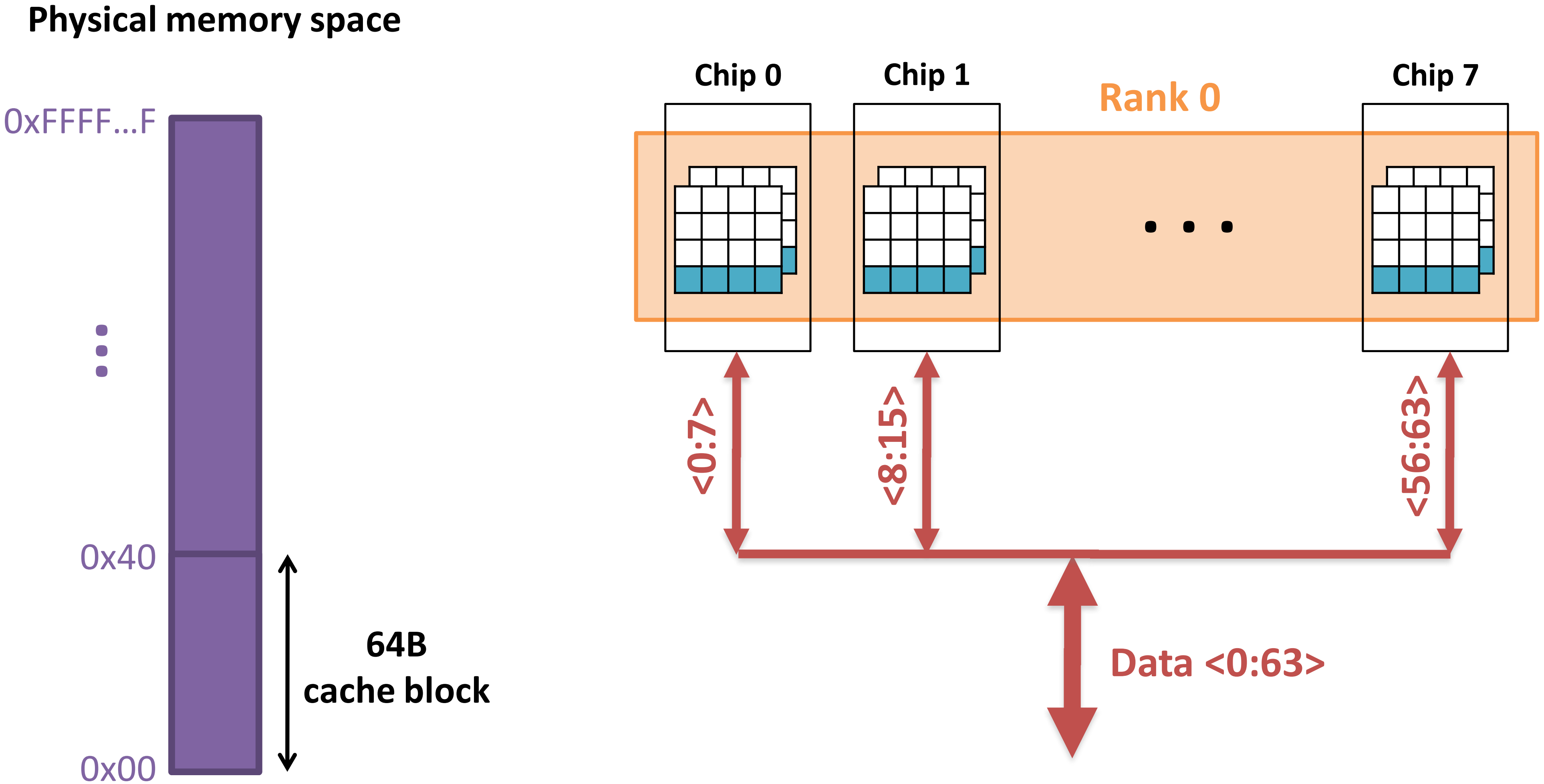
Enhanced DRAMs

- Basic DRAM cell has not changed since its invention in 1966.
  - Commercialized by Intel in 1970.
- DRAM cores with better interface logic and faster I/O :
  - Synchronous DRAM (SDRAM)
    - Uses a conventional clock signal instead of asynchronous control
    - Allows reuse of the row addresses (e.g., RAS, CAS, CAS, CAS)
  - Double data-rate synchronous DRAM (DDR SDRAM)
    - Double edge clocking sends two bits per cycle per pin
    - Different types distinguished by size of small prefetch buffer:
      - DDR (2 bits), DDR2 (4 bits), DDR3 (8 bits)
    - By 2010, standard for most server and desktop systems
    - Intel Core i7 supports only DDR3 SDRAM

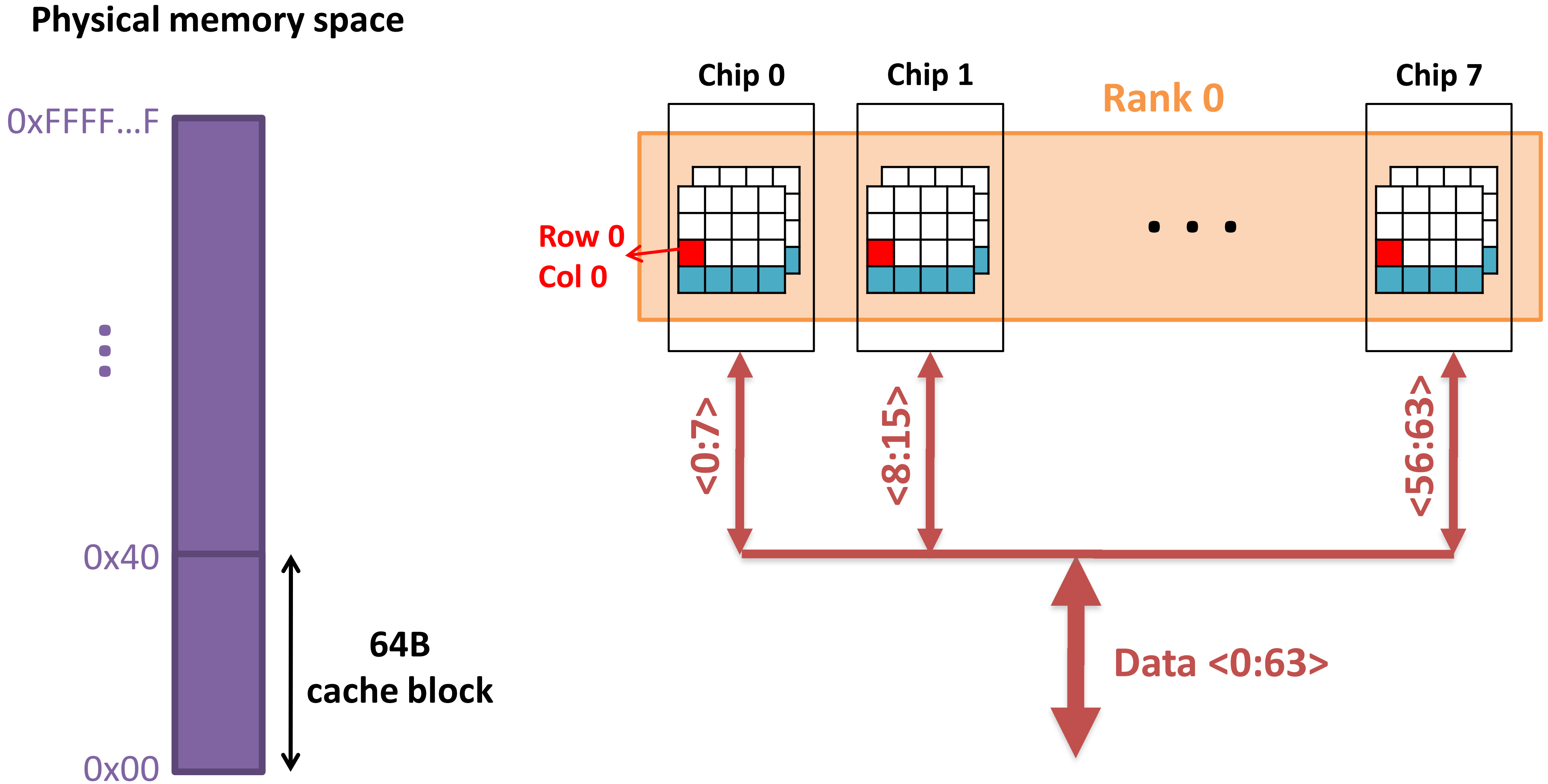
Example: Transferring a Cache Block



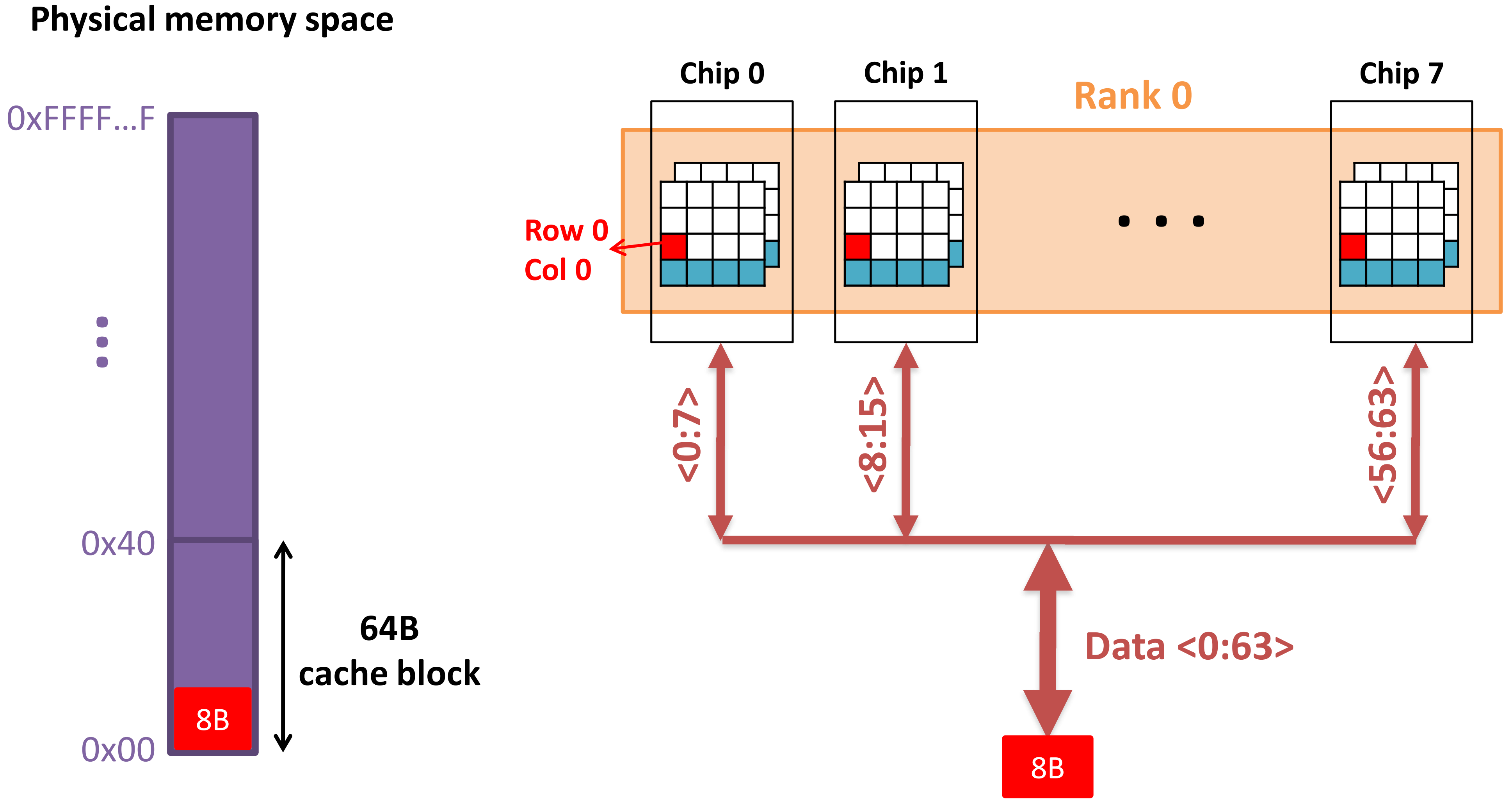
Example: Transferring a Cache Block



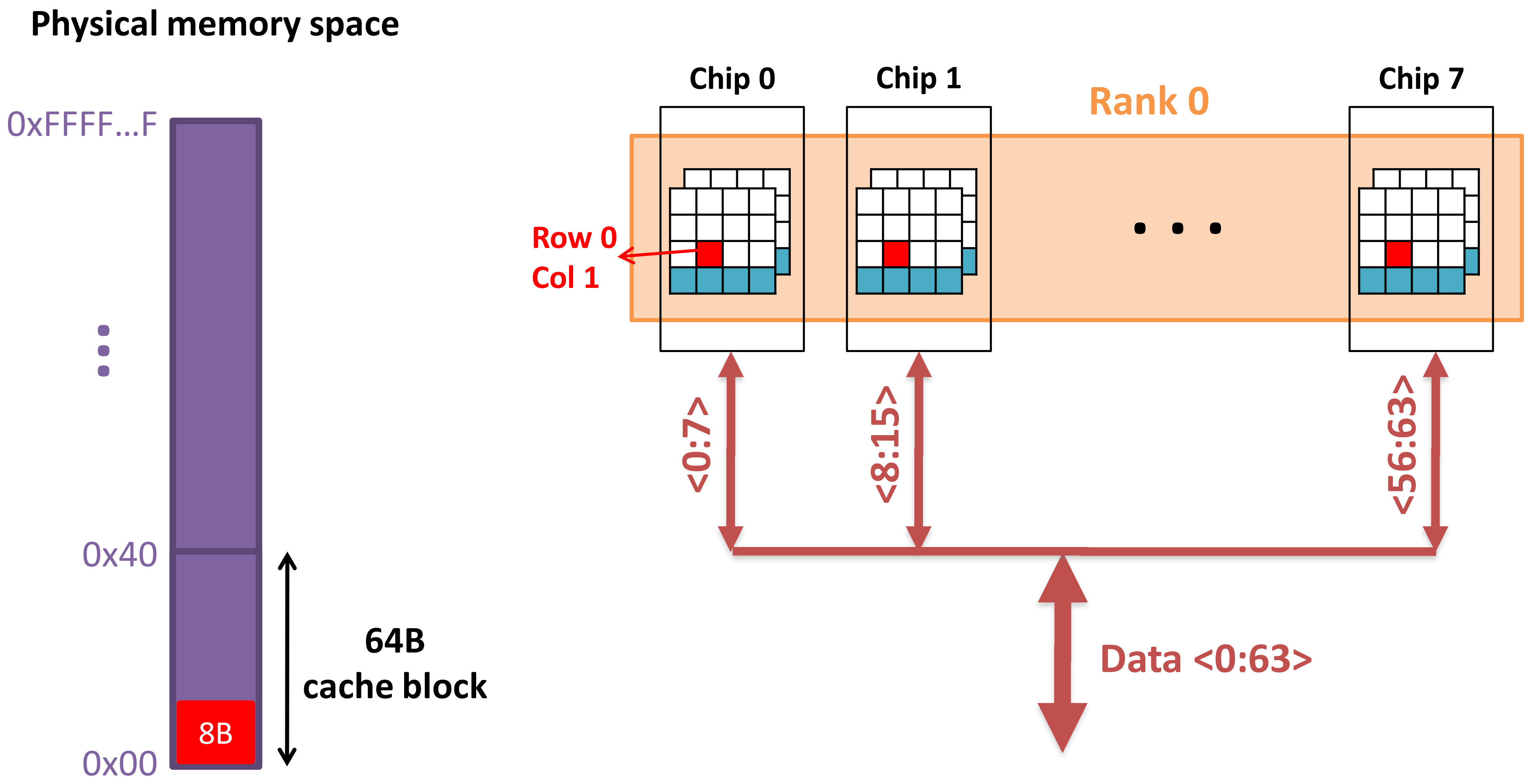
Example: Transferring a Cache Block



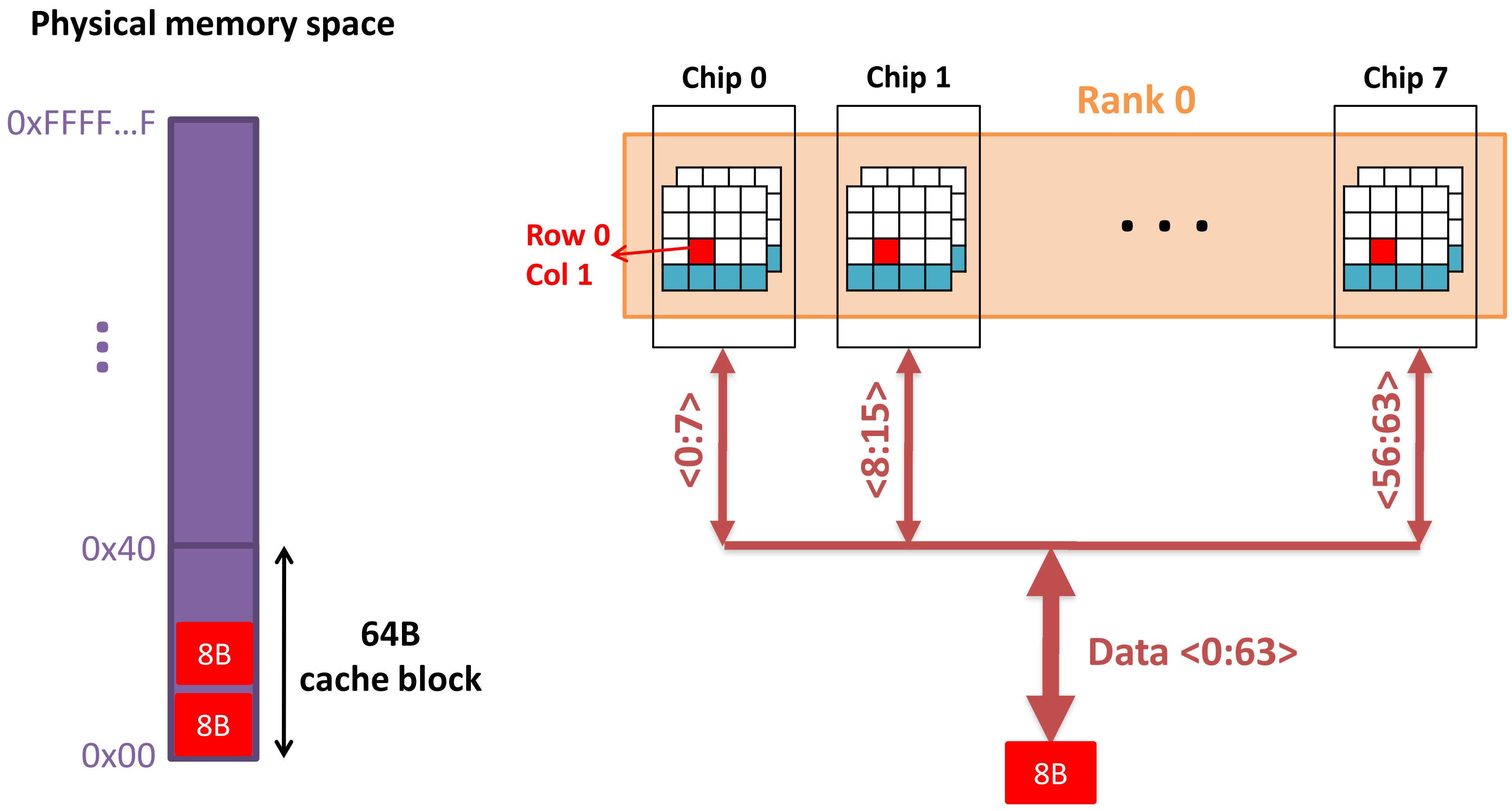
Example: Transferring a Cache Block



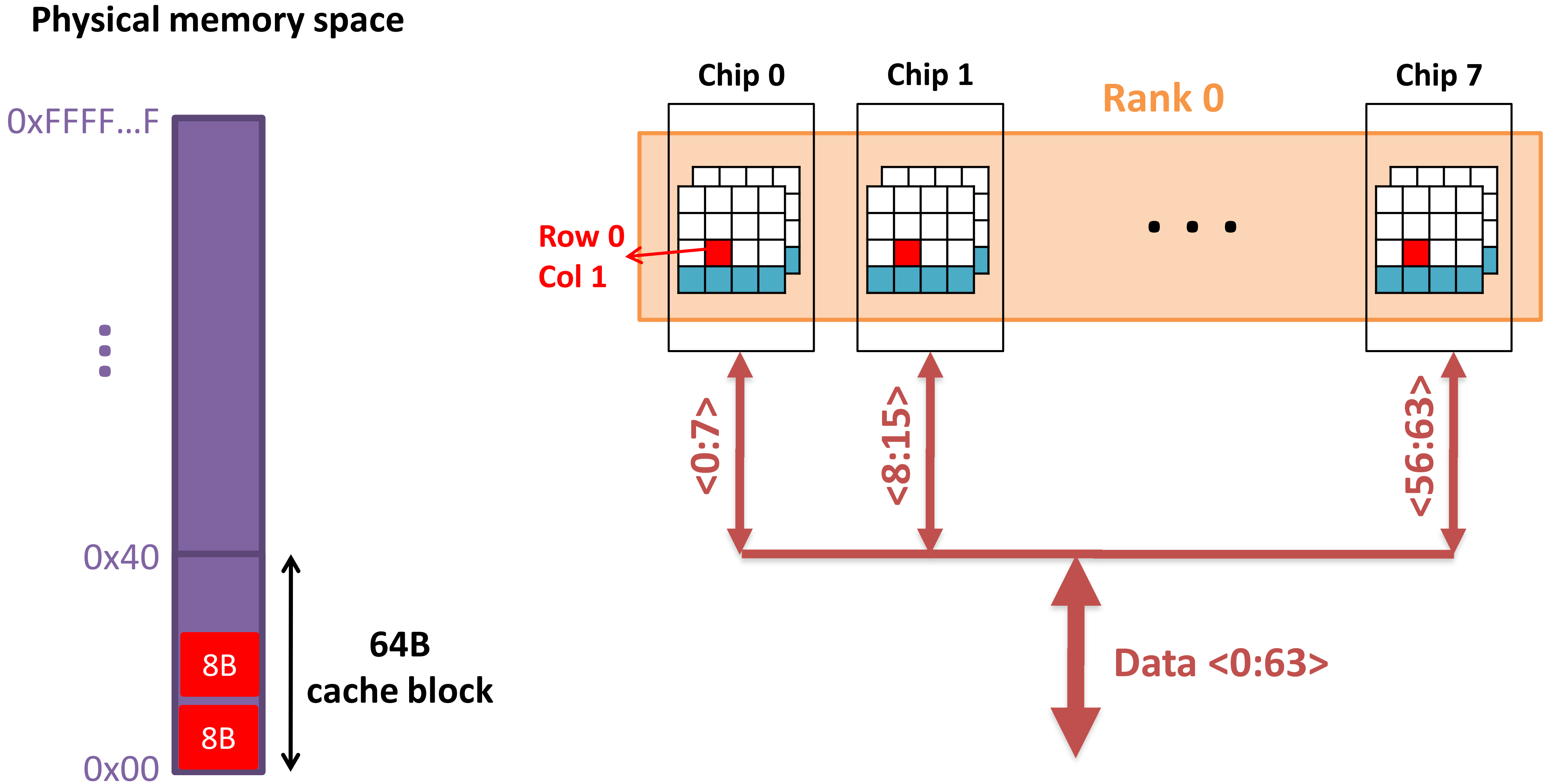
Example: Transferring a Cache Block



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.

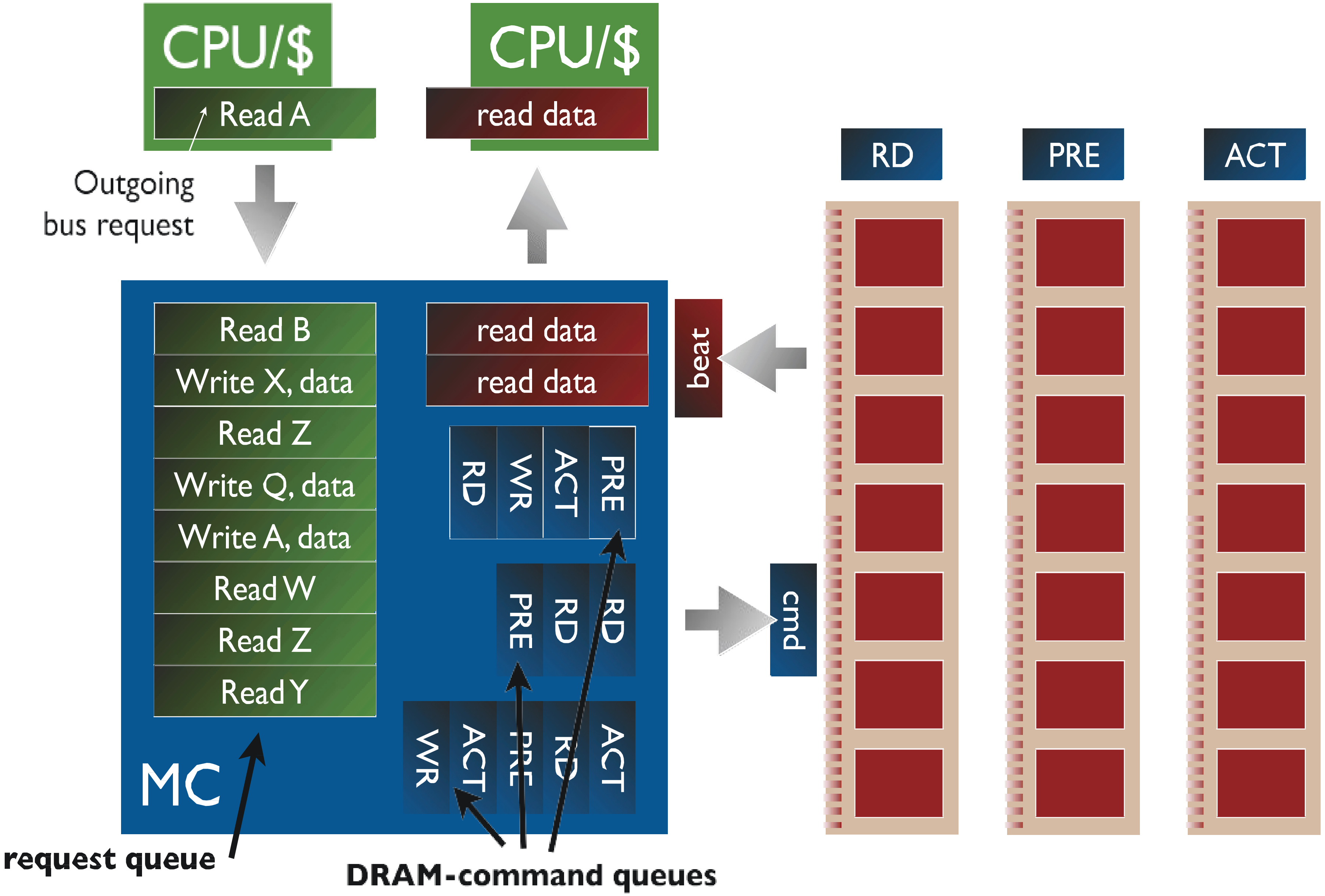
18-600 Foundations of Computer Systems

Lecture 10:
"The Memory Hierarchy"

A. Memory Technologies
B. Main Memory Implementation
   a. DRAM Organization
   b. DRAM Operation
   c. Memory Controller
C. Disk Storage Technologies

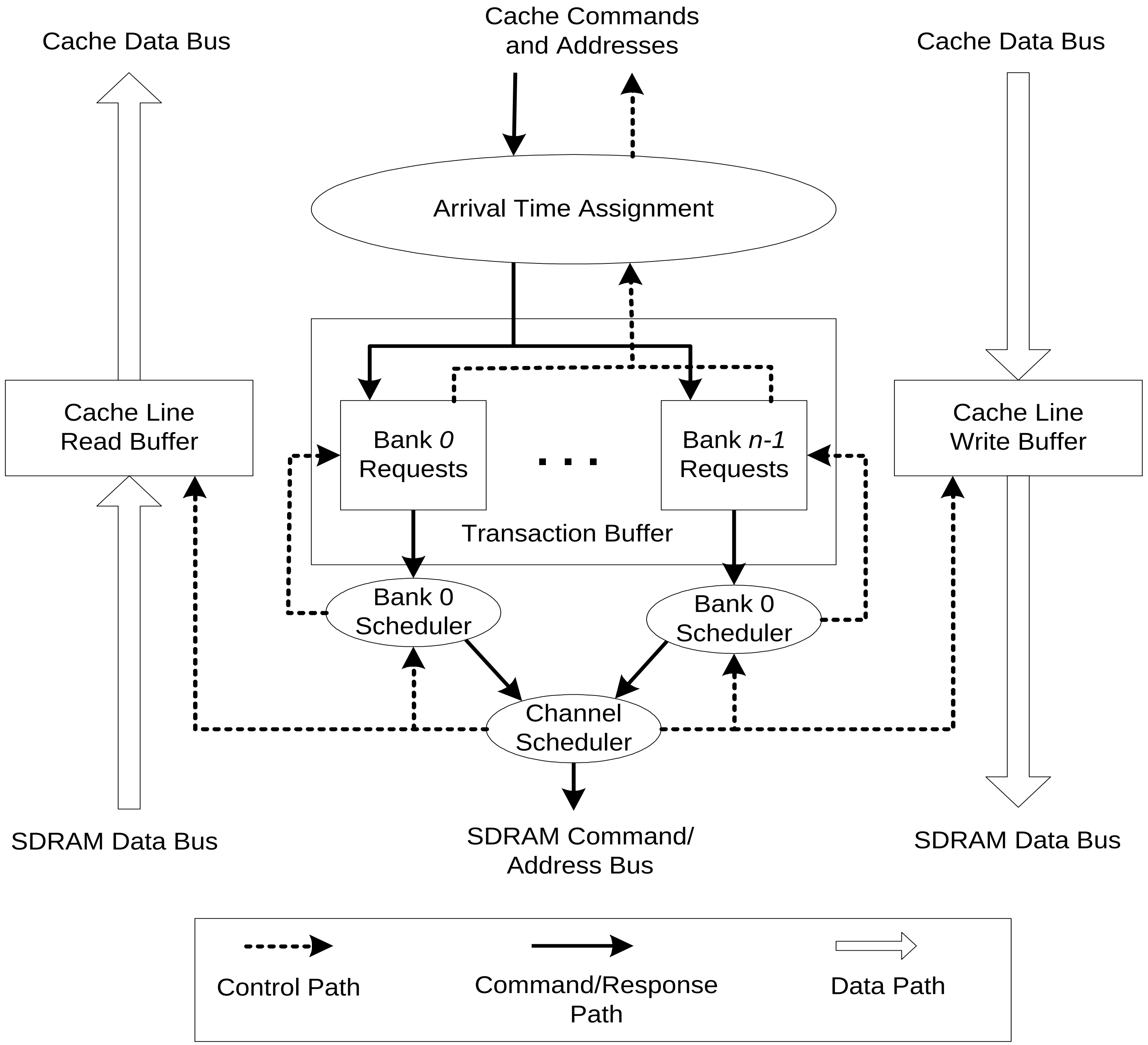
Electrical & Computer ENGINEERING

Memory Controller



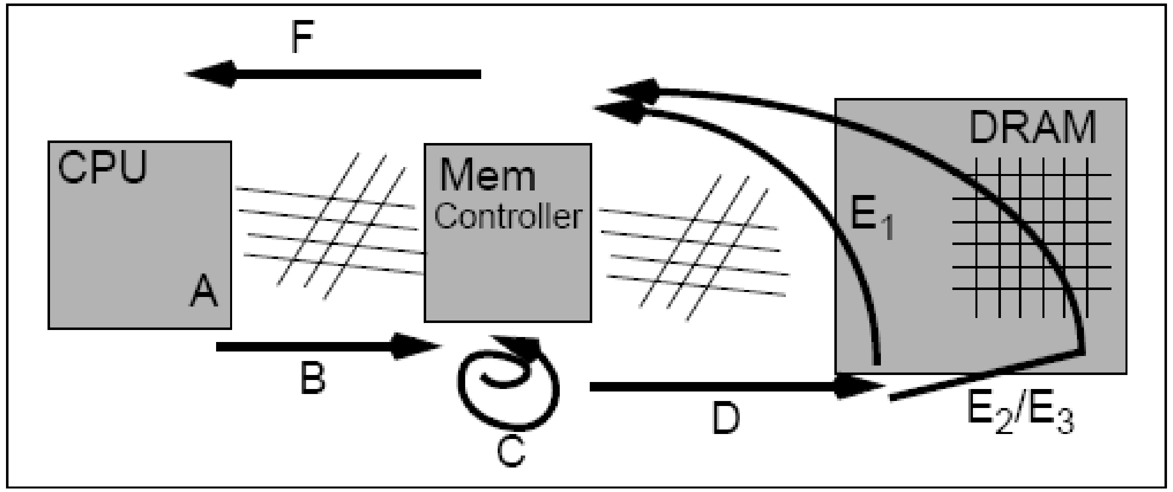
Memory Controllers

- Contains buffering
  - **In both directions**
- Schedulers manage resources
  - **Channel and banks**



Latency Components: Basic DRAM Operation

- CPU → controller transfer time
- Controller latency
  - Queuing & scheduling delay at the controller
  - Access converted to basic commands
- Controller → DRAM transfer time
- DRAM bank latency
  - Simple CAS if row is "open" OR
  - RAS + CAS if array precharged OR
  - PRE + RAS + CAS (worst case)
- DRAM → CPU transfer time (through controller)



A: Transaction request may be delayed in Queue
B: Transaction request sent to Memory Controller
C: Transaction converted to Command Sequences (may be queued)
D: Command/s Sent to DRAM
E_1: Requires only a **CAS** or
E_2: Requires **RAS + CAS** or
E_3: Requires **PRE + RAS + CAS**
F: Transaction sent back to CPU

"DRAM Latency" = A + B + C + D + E + F

18-600 Foundations of Computer Systems

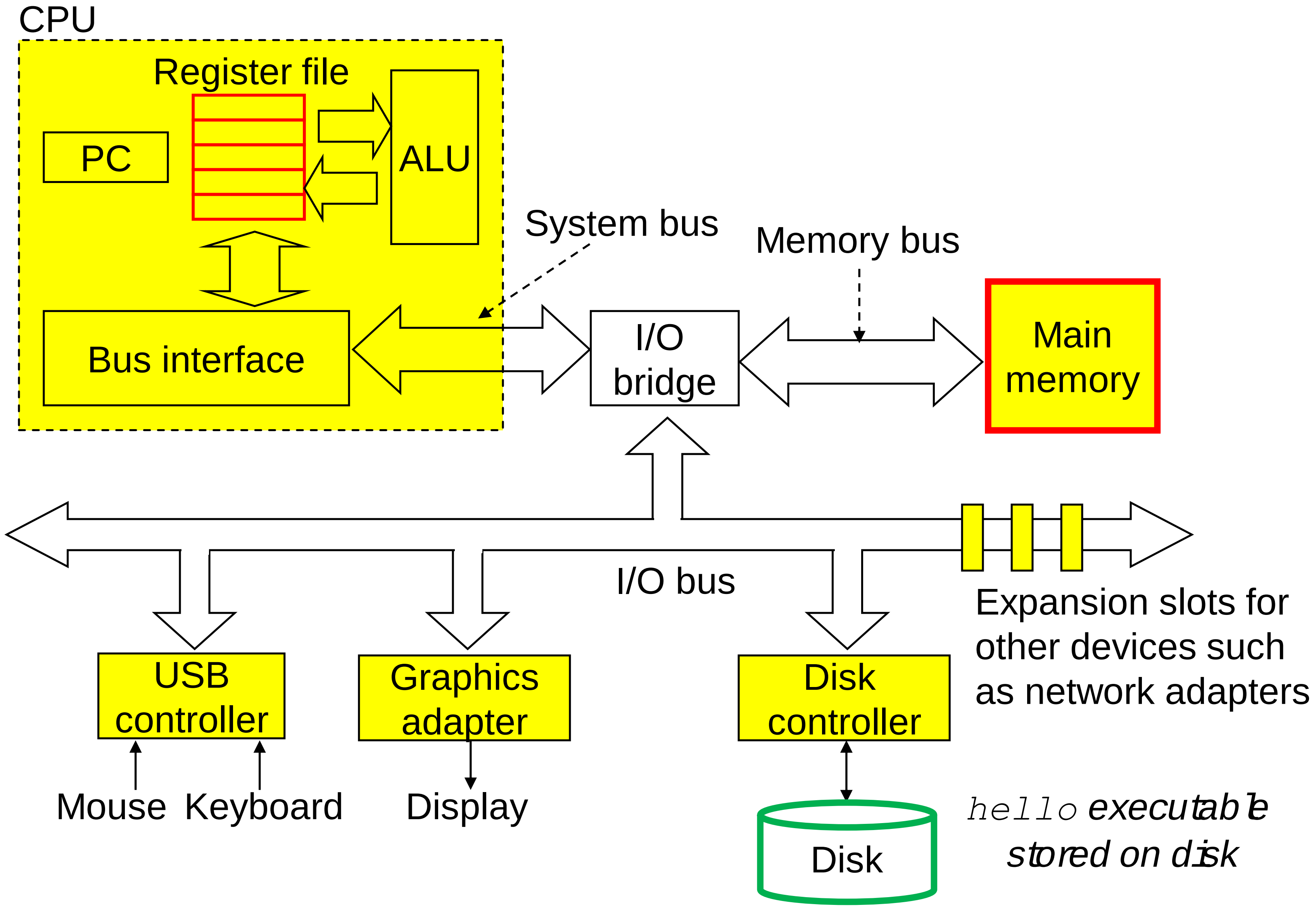
Lecture 10:
"The Memory Hierarchy"

A. Memory Technologies
B. Main Memory Implementation
   a. DRAM Organization
   b. DRAM Operation
   c. Memory Controller
C. Disk Storage Technologies

Electrical & Computer ENGINEERING

From Lec #1…

Typical Computer Organization



What's Inside A Disk Drive?

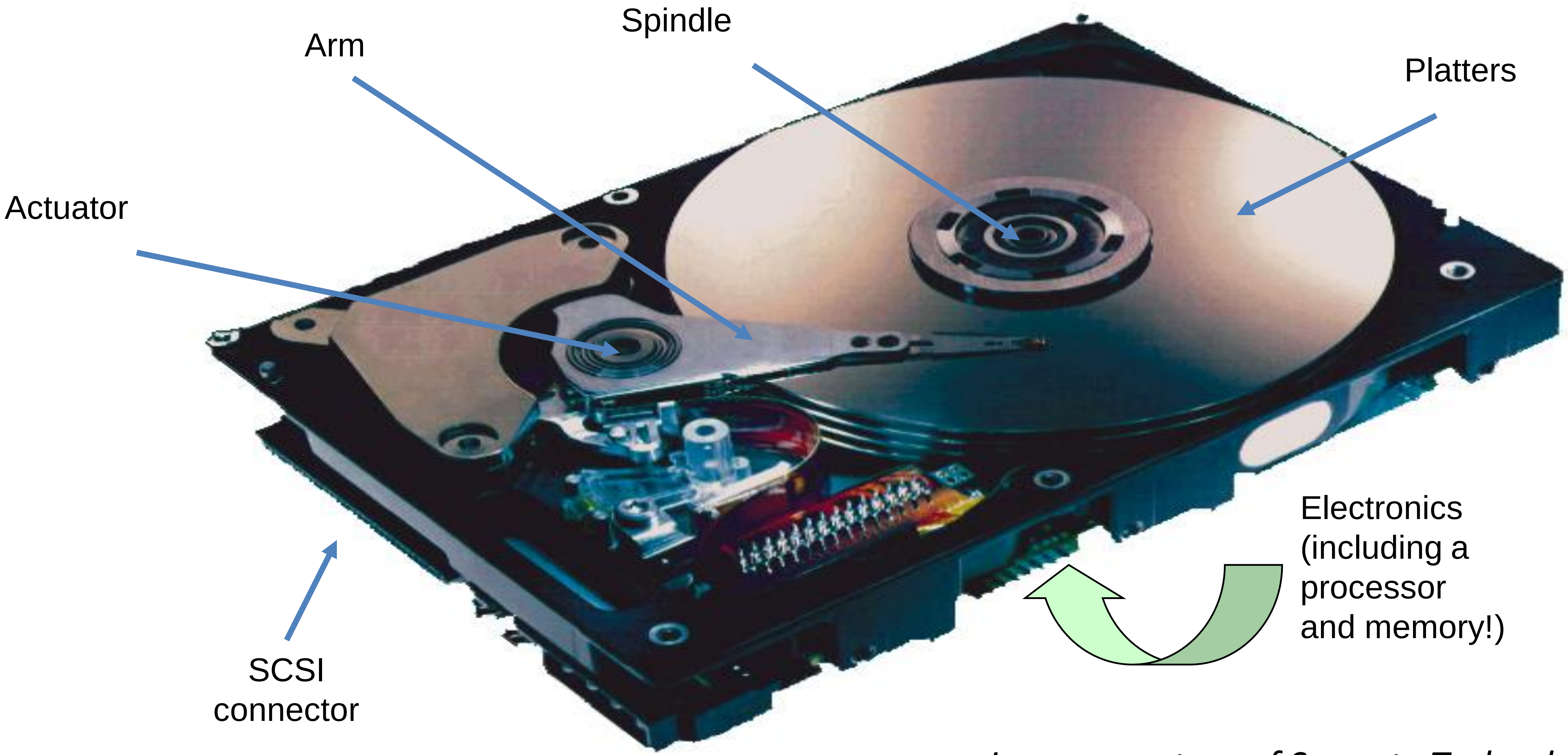
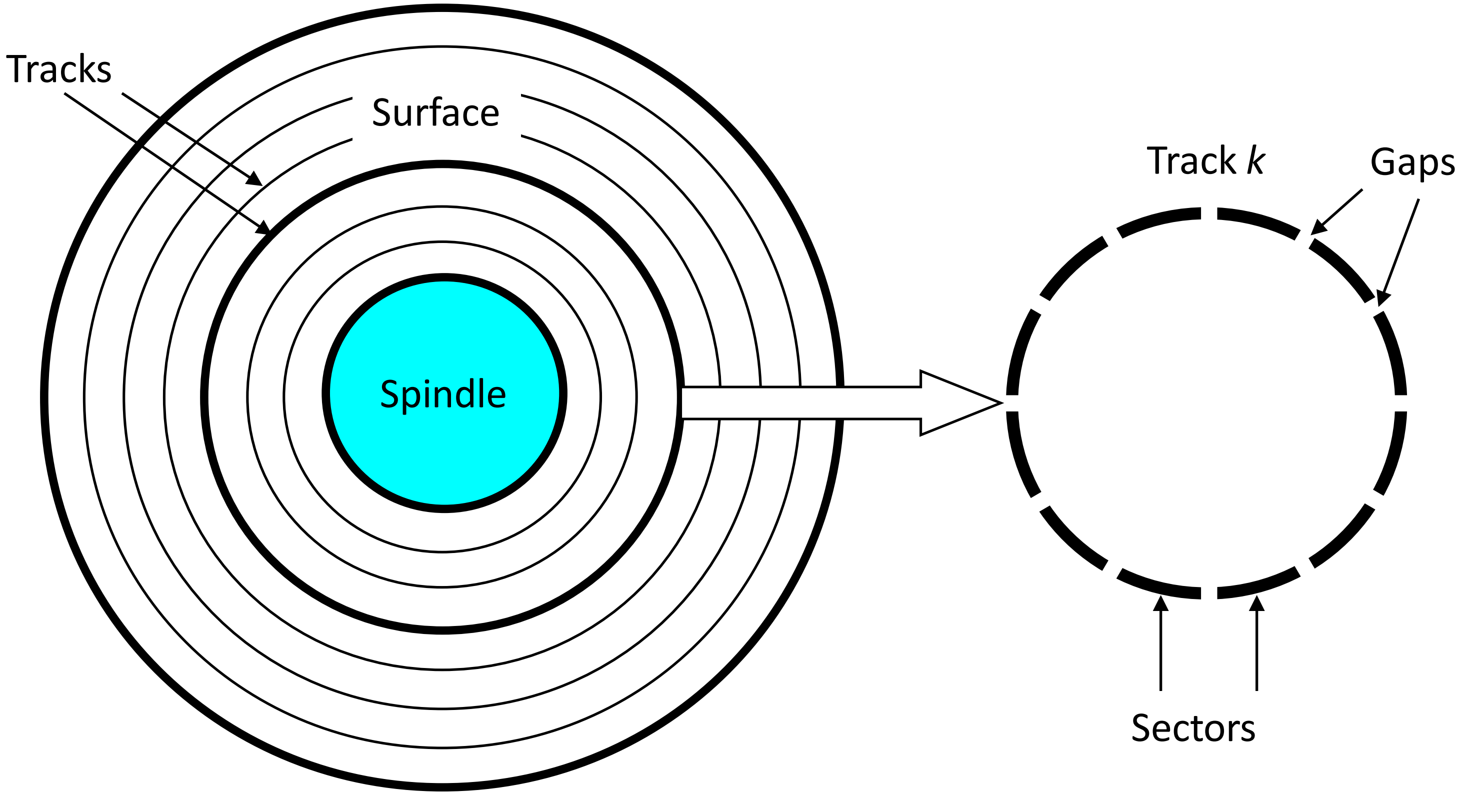
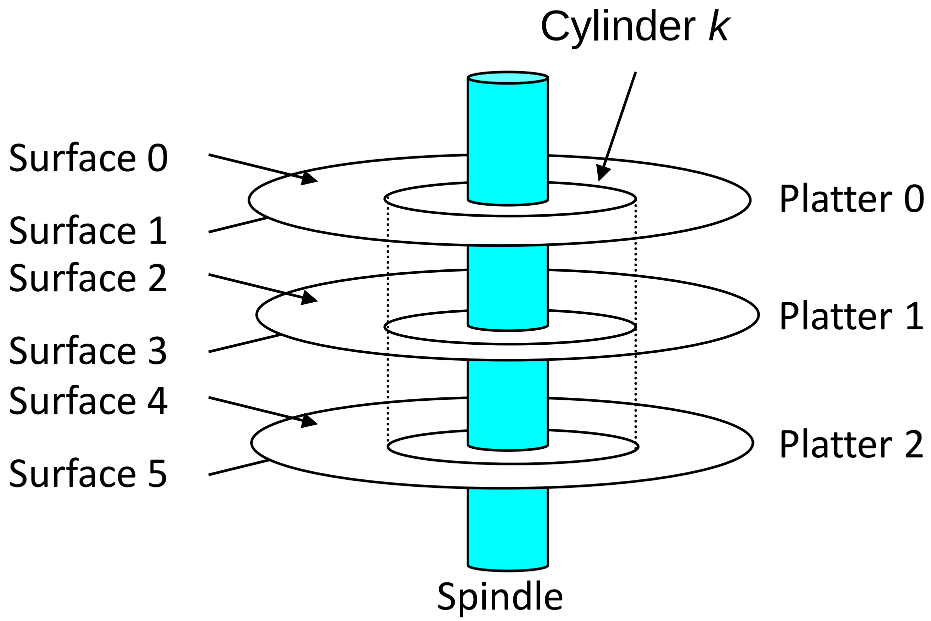


Image courtesy of Seagate Technology

Disk Geometry

- Disks consist of platters, each with two surfaces.
- Each surface consists of concentric rings called tracks.
- Each track consists of sectors separated by gaps.
- Aligned tracks form a cylinder.





Disk Capacity

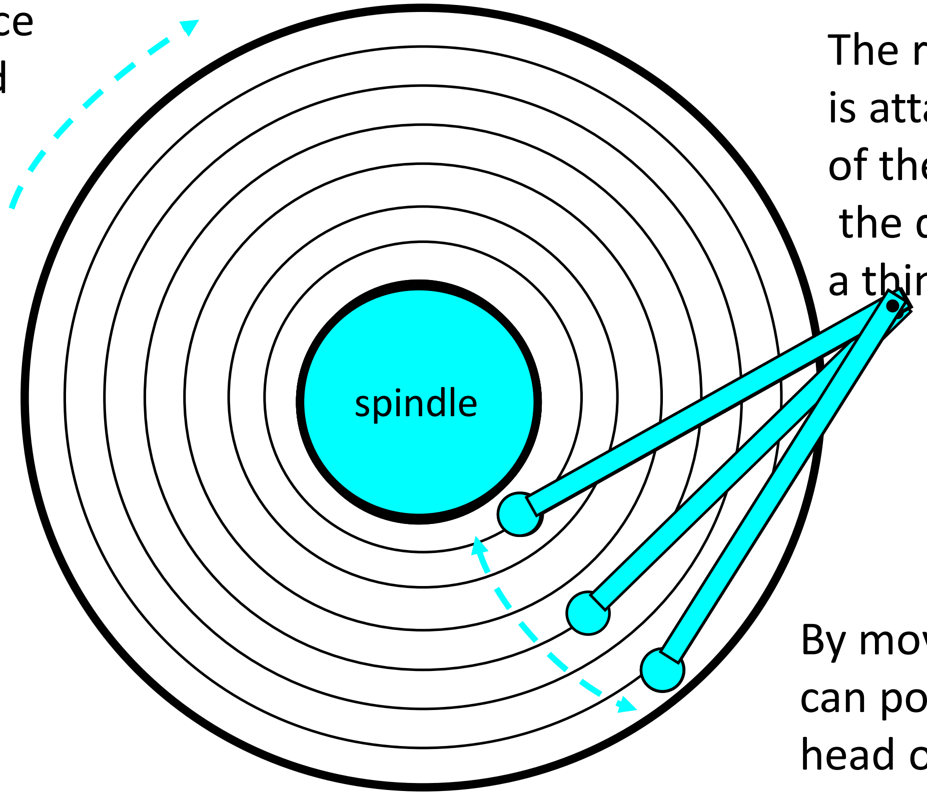
Capacity = (# bytes/sector) x (avg. # sectors/track) x
(# tracks/surface) x (# surfaces/platter) x
(# platters/disk)

Example:

- 512 bytes/sector
- 300 sectors/track (on average)
- 20,000 tracks/surface
- 2 surfaces/platter
- 5 platters/disk

Capacity = 512 x 300 x 20000 x 2 x 5
= 30,720,000,000
= 30.72 GB

Disk Operation

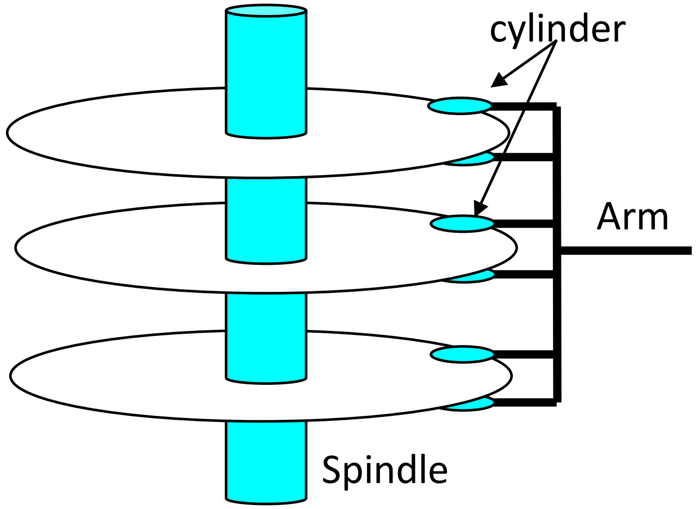


The disk surface spins at a fixed rotational rate

The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

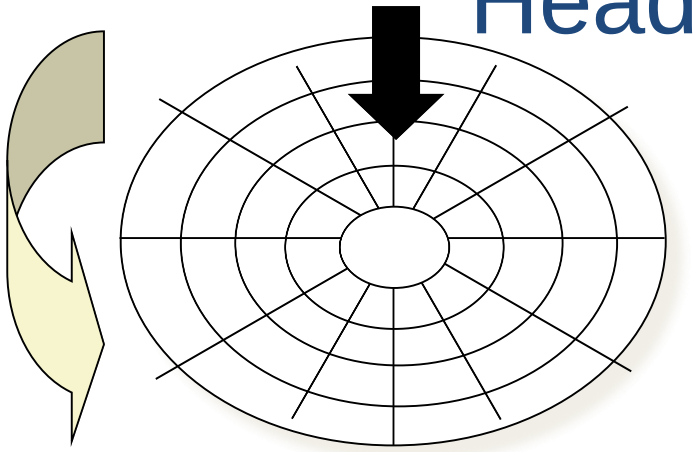
Single-Platter View



Read/write heads move in unison from cylinder to cylinder

Multi-Platter View

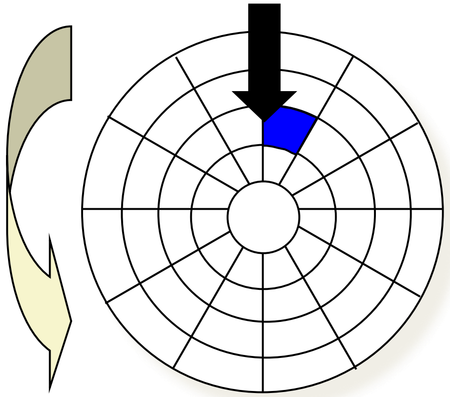
Disk Access



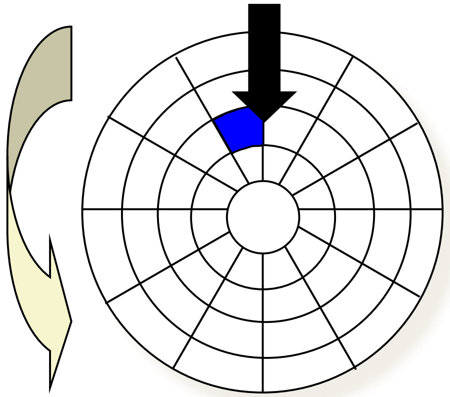
Head in position above a track

Rotation is counter-clockwise

Disk Access – Read

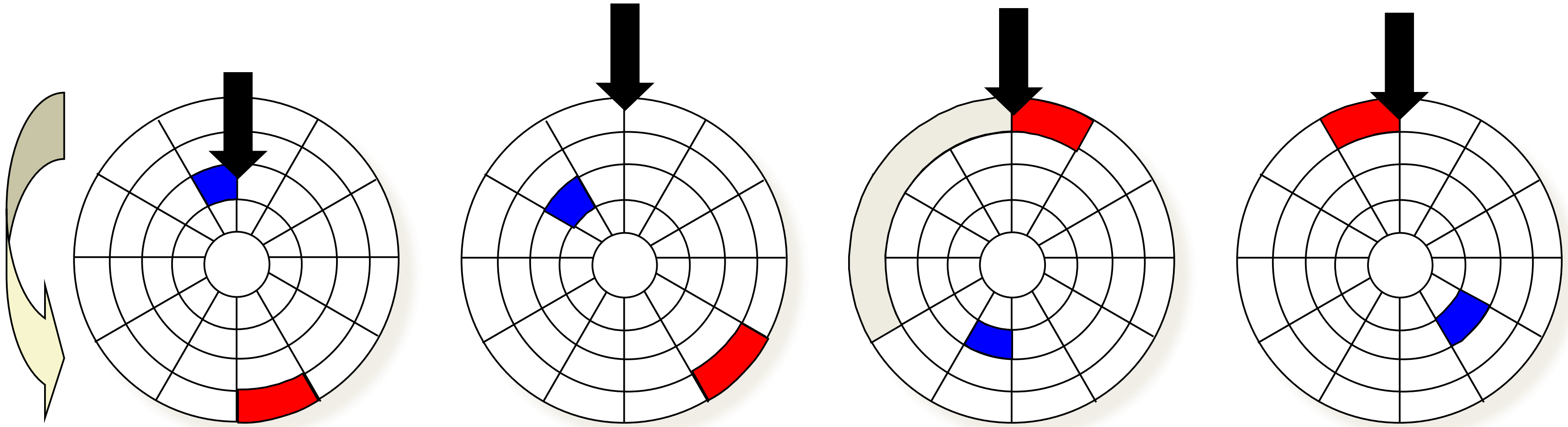


About to read blue sector



After BLUE read

Disk Access of RED



After BLUE read

Seek for RED

Rotational latency

After RED read

Red request scheduled next

Wait for red sector to rotate around

Data transfer

Seek

Rotational latency

Data transfer

Disk Access Time

- Average time to access some target sector approximated by :
  - Taccess = Tavg seek + Tavg rotation + Tavg transfer
- Seek time (Tavg seek)
  - Time to position heads over cylinder containing target sector.
  - Typical Tavg seek is 3—9 ms
- Rotational latency (Tavg rotation)
  - Time waiting for first bit of target sector to pass under r/w head.
  - Tavg rotation = 1/2 x 1/RPMs x 60 sec/1 min
  - Typical Tavg rotation = 7200 RPMs
- Transfer time (Tavg transfer)
  - Time to read the bits in the target sector.
  - Tavg transfer = 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min.

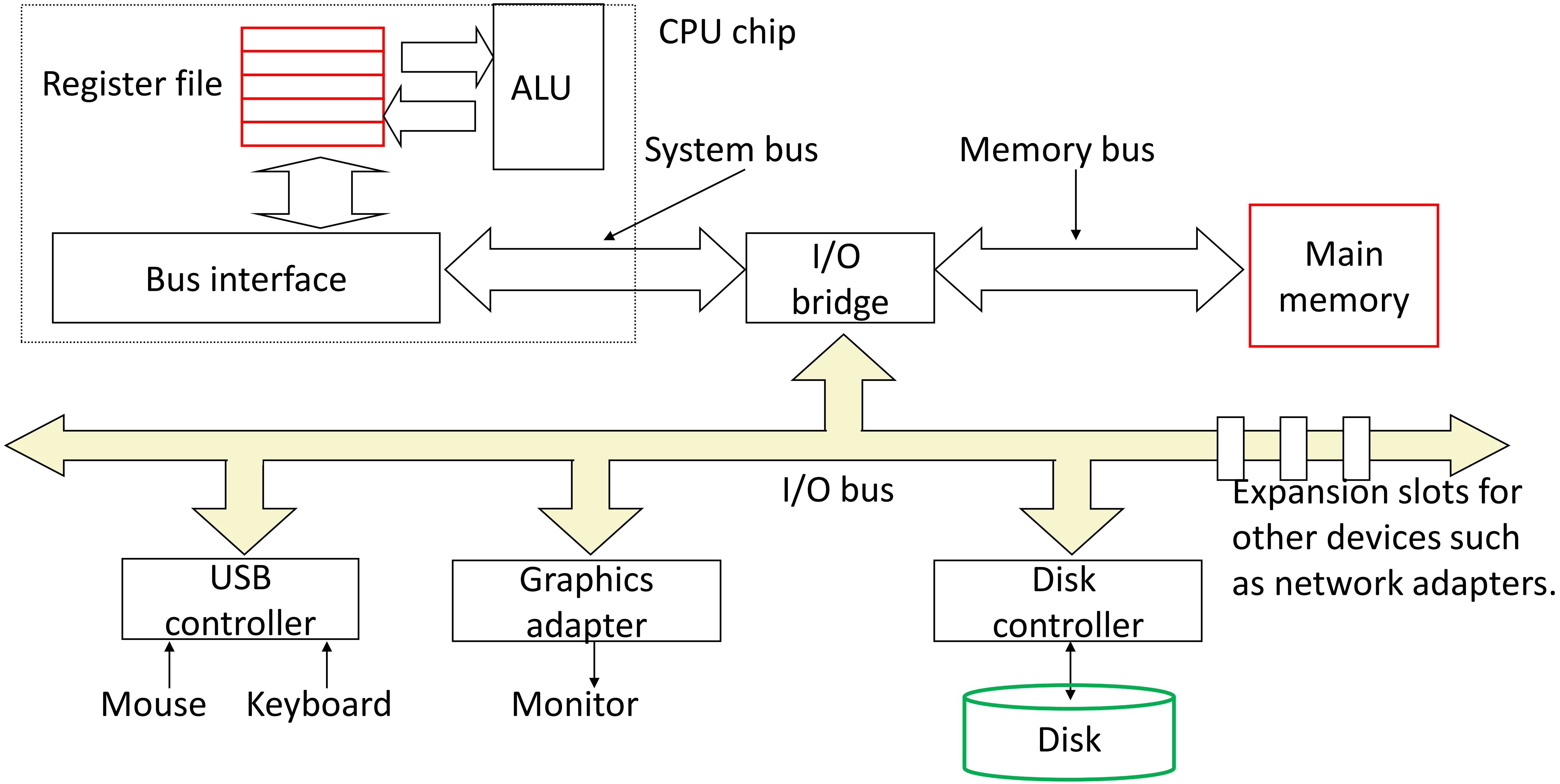
Disk Access Time Example

- Given:
  - Rotational rate = 7,200 RPM
  - Average seek time = 9 ms.
  - Avg # sectors/track = 400.
- Derived:
  - Tavg rotation = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms.
  - Tavg transfer = 60/7200 RPM x 1/400 secs/track x 1000 ms/sec = 0.02 ms
  - Taccess = 9 ms + 4 ms + 0.02 ms
- Important points:
  - Access time dominated by seek time and rotational latency.
  - First bit in a sector is the most expensive, the rest are free.
  - SRAM access time is about 4 ns/doubleword, DRAM about 60 ns
    - Disk is about 40,000 times slower than SRAM,
    - 2,500 times slower then DRAM.

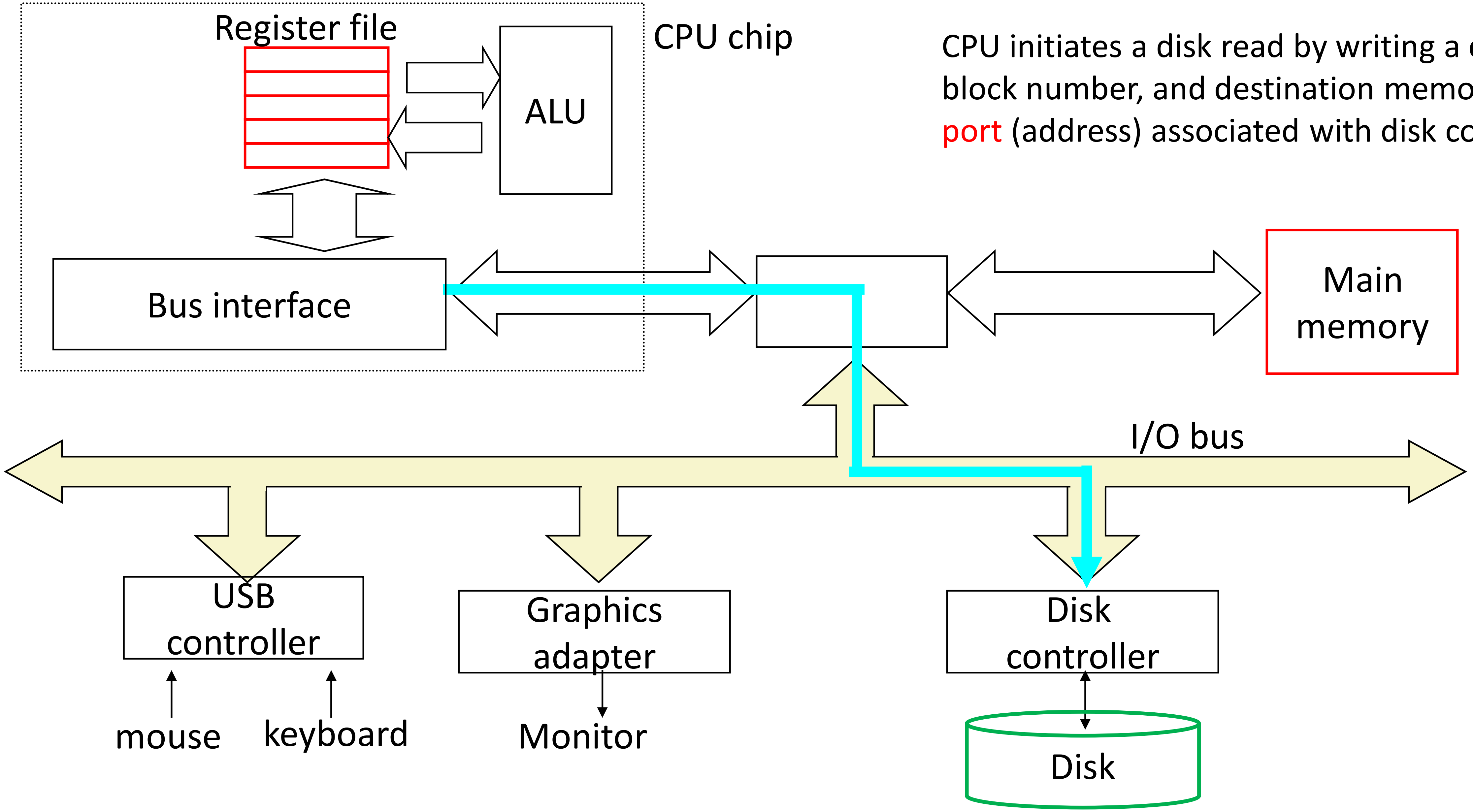
Logical Disk Blocks

- Modern disks present a simpler abstract view of the complex sector geometry:
  - The set of available sectors is modeled as a sequence of b-sized logical blocks (0, 1, 2, ...)
- Mapping between logical blocks and actual (physical) sectors
  - Maintained by hardware/firmware device called disk controller.
  - Converts requests for logical blocks into (surface,track,sector) triples.
- Allows controller to set aside spare cylinders for each zone.
  - Accounts for the difference in "formatted capacity" and "maximum capacity".

I/O Bus


CPU chip
Register file
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory
I/O bus
Expansion slots for other devices such as network adapters.
USB controller
Graphics adapter
Disk controller
Mouse
Keyboard
Monitor
Disk

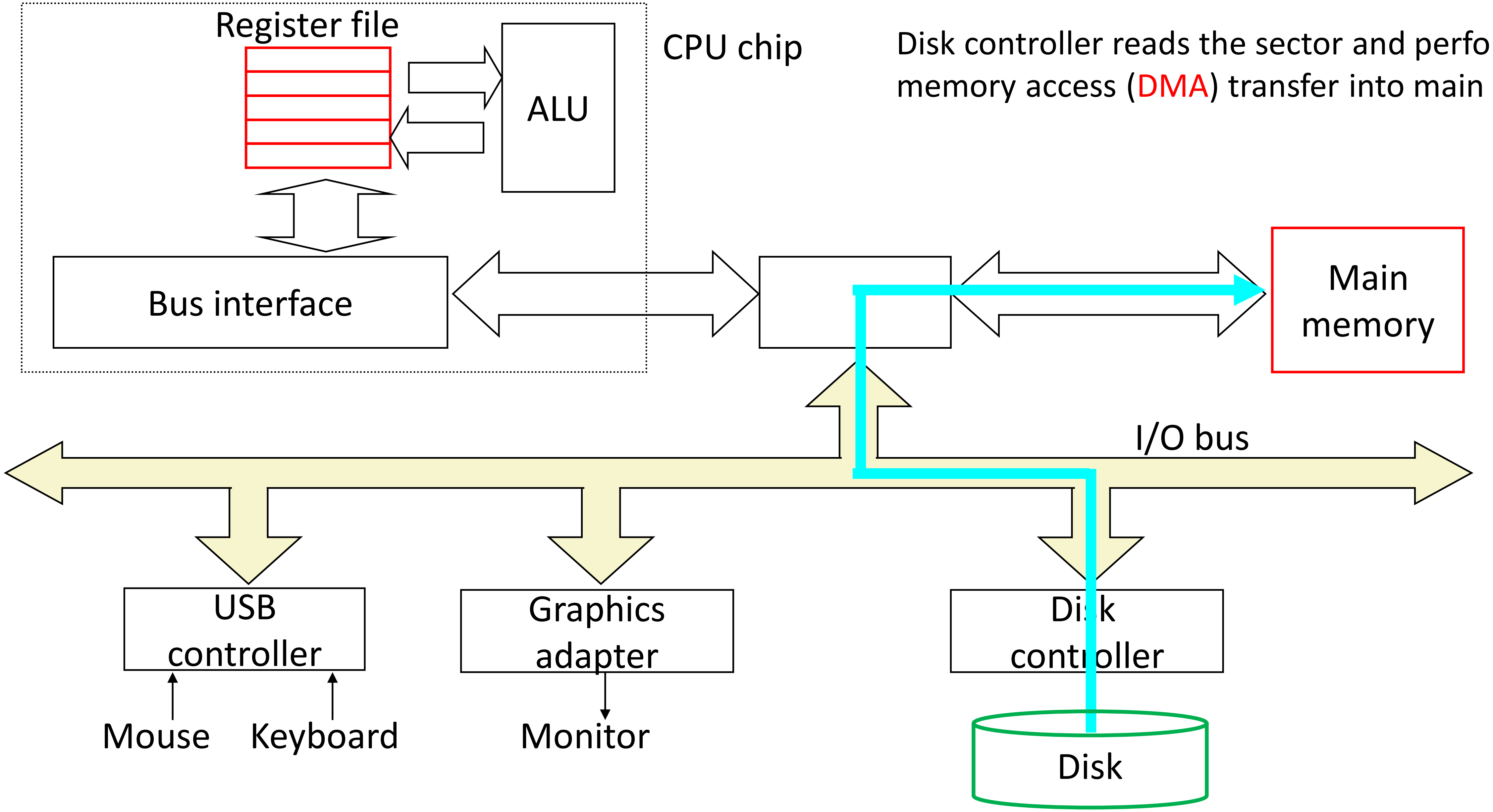
Reading a Disk Sector (1)



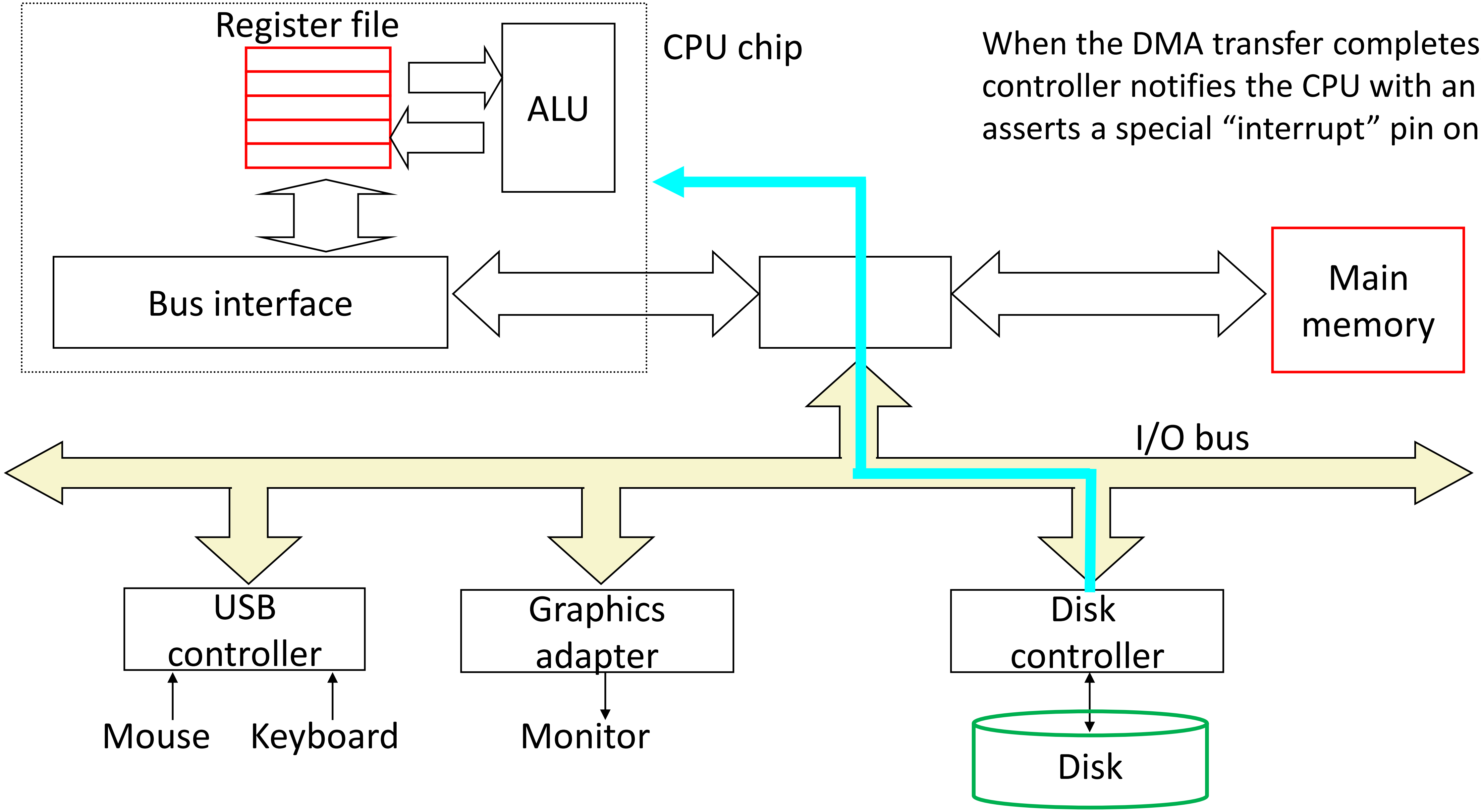
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

Reading a Disk Sector (2)

Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.



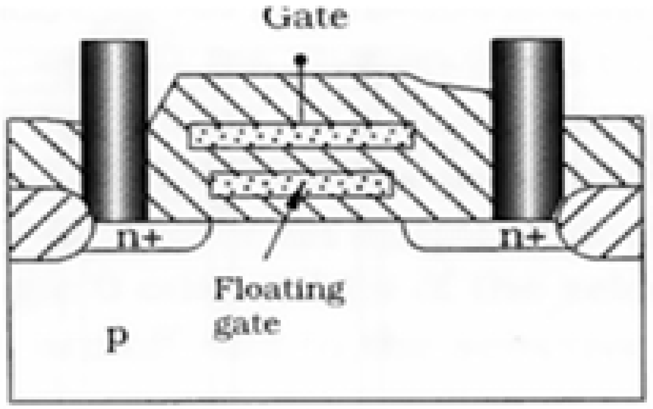
Reading a Disk Sector (3)



When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)

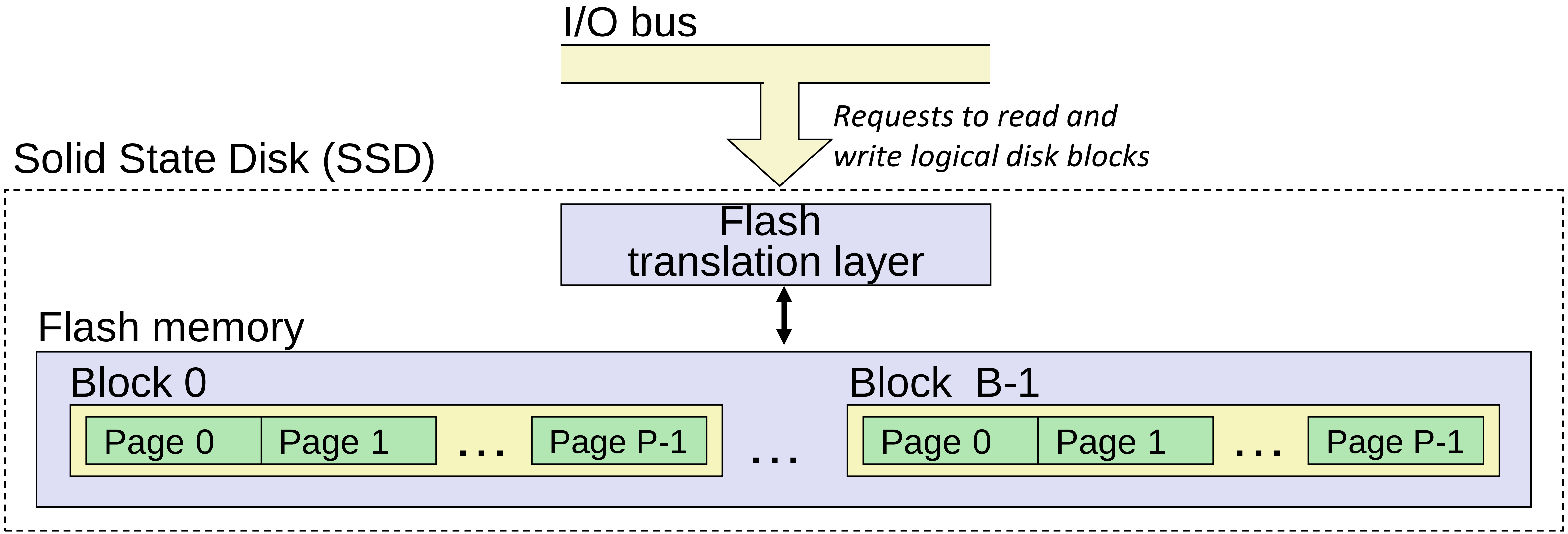
Nonvolatile Memories

- DRAM and SRAM are volatile memories
  - Lose information if powered off.
- Nonvolatile memories retain value even if powered off
  - Read-only memory (ROM): programmed during production
  - Programmable ROM (PROM): can be programmed once
  - Erasable PROM (EPROM): can be bulk erased (UV, X-Ray)
  - Electrically erasable PROM (EEPROM): electronic erase capability
  - Flash memory: EEPROMs. with partial (block-level) erase capability
    - Wears out after about 100,000 erasing cycles
- Uses for Nonvolatile Memories
  - Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,…)
  - Solid state disks (replace rotating disks in thumb drives, smart phones, mp3 players, tablets, laptops,…)
  - Disk caches



EPROM device structure

Solid State Disks (SSDs)

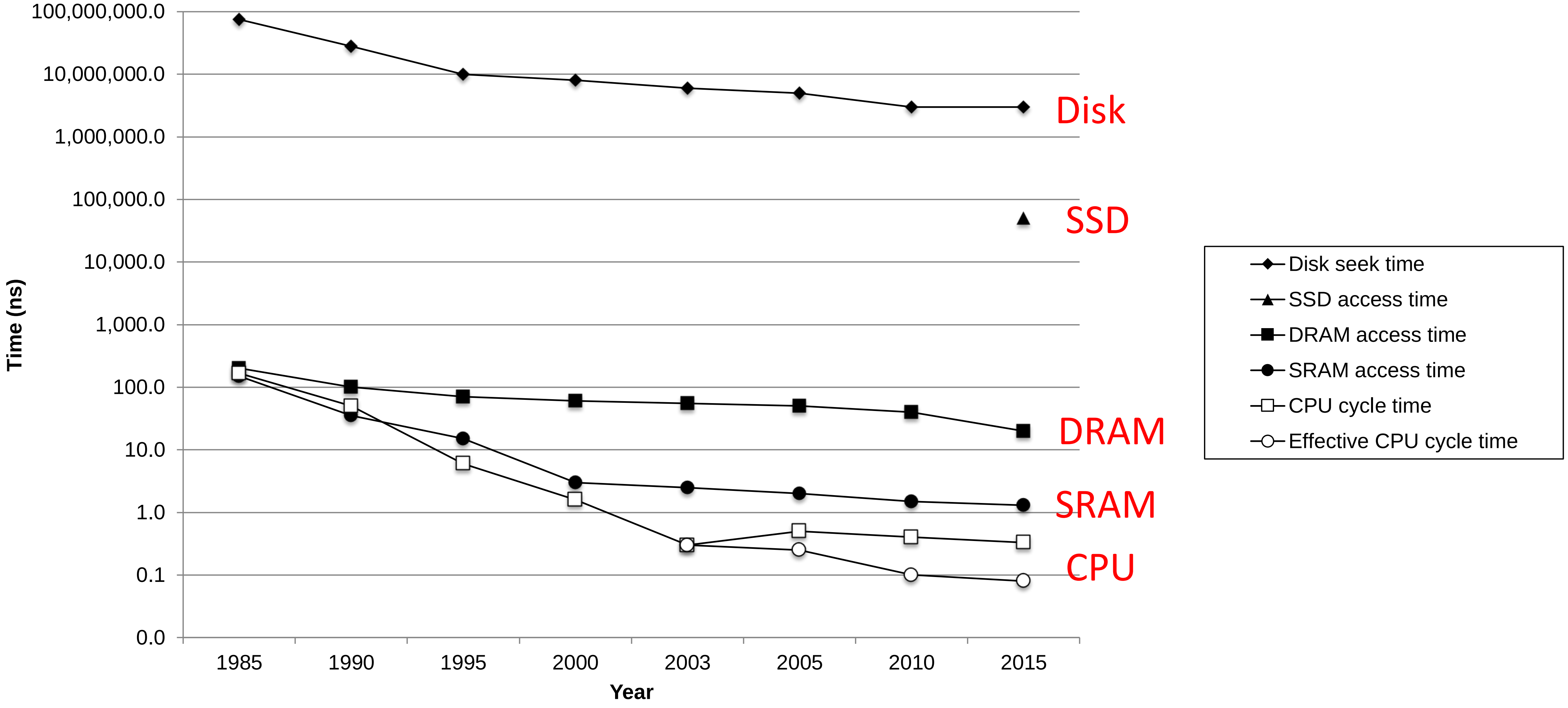


- Pages: 512B to 4KB, Blocks: 32 to 128 pages
- Data read/written in units of pages.
- Page can be written only after its block has been erased
- A block wears out after about 100,000 repeated writes.

SSD Tradeoffs vs Rotating Disks

- Advantages
  - No moving parts → faster, less power, more rugged
- Disadvantages
  - Have the potential to wear out
    - Mitigated by "wear leveling logic" in flash translation layer
    - E.g. Intel SSD 730 guarantees 128 petabyte (128 x 10^{15} bytes) of writes before they wear out
  - In 2015, about 30 times more expensive per byte
- Applications
  - MP3 players, smart phones, laptops
  - Beginning to appear in desktops and servers (as disk cache)

The CPU-Memory-Storage Gaps



18-600 Foundations of Computer Systems

Lecture 11:
"Cache Memories & Multicore Processors"

John P. Shen & Gregory Kesden
October 4, 2017

Next Time …

- Required Reading Assignment:
  - **Chapter 6 of CS:APP (3rd edition) by Randy Bryant & Dave O'Hallaron.**

Electrical & Computer ENGINEERING