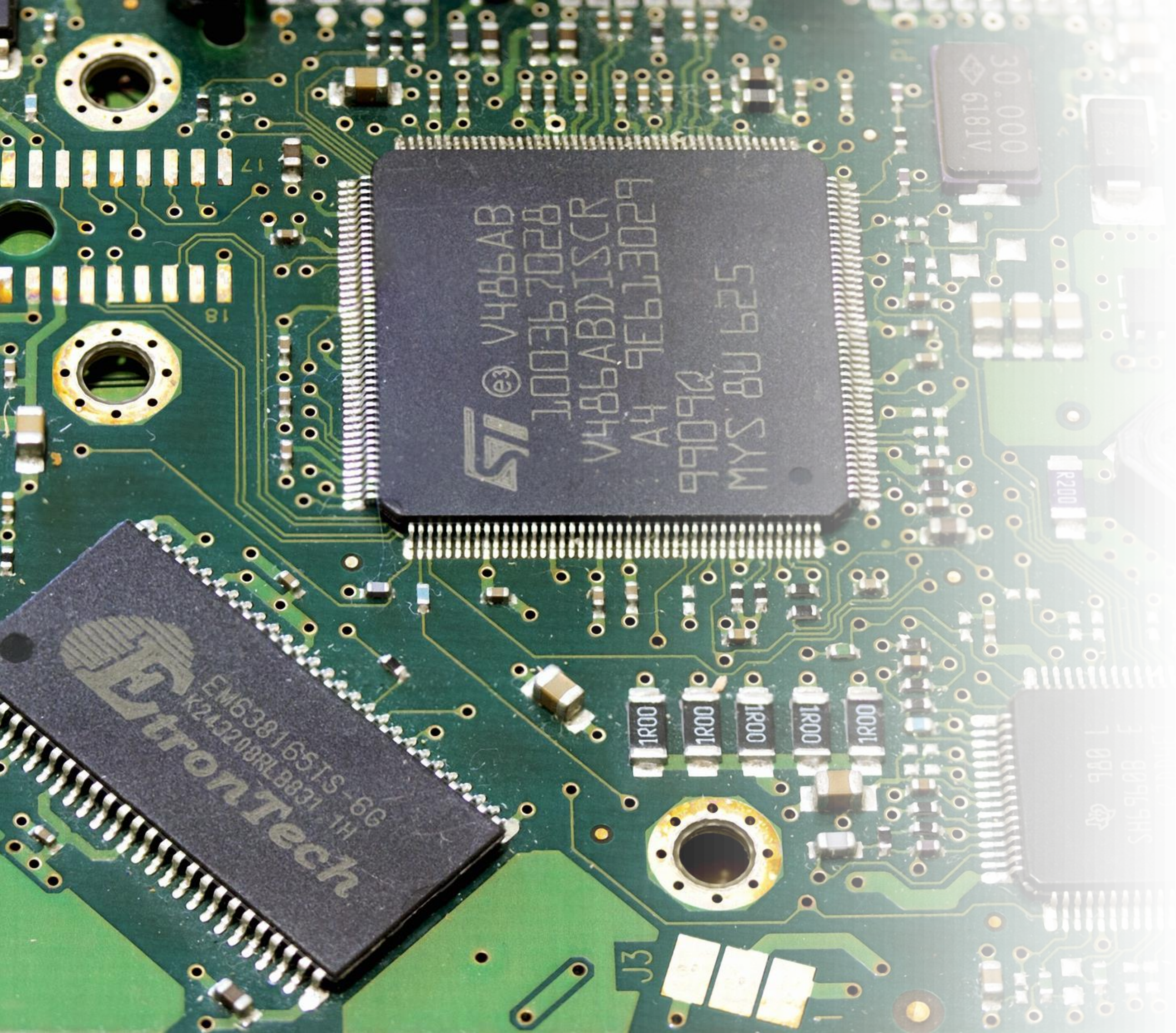


Digital Logic and System Design

8. Registers, Counters, and Memory

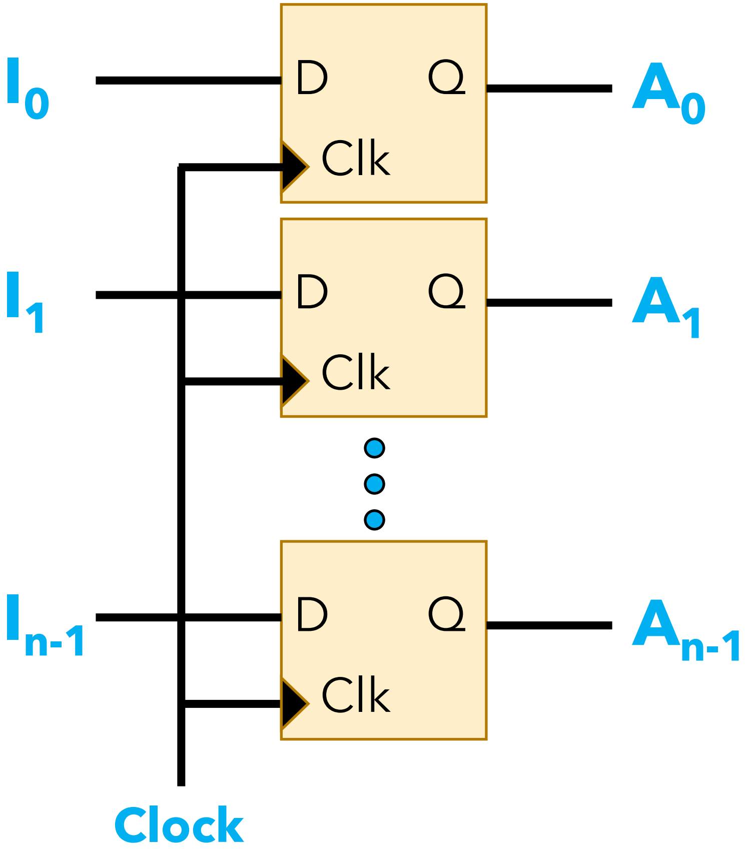
COL215, I Semester 2022-2023
Venue: LHC 111
'E' Slot: Tue, Wed, Fri 10:00-11:00

Instructor: Preeti Ranjan Panda
panda@cse.iitd.ac.in
www.cse.iitd.ac.in/~panda/
Dept. of Computer Science & Engg., IIT Delhi

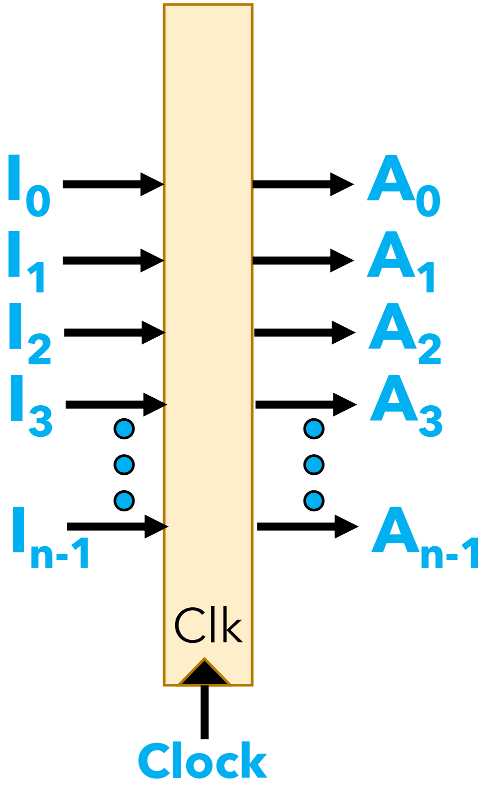
Registers

- Group of Flip-flops
  - common **clock**
  - stores **1 bit** per flip-flop
- Recall: **State Register** of FSM
- n-bit value transferred from Ds to Qs on clock edge
  - storing **n-bit data**

n-bit Register

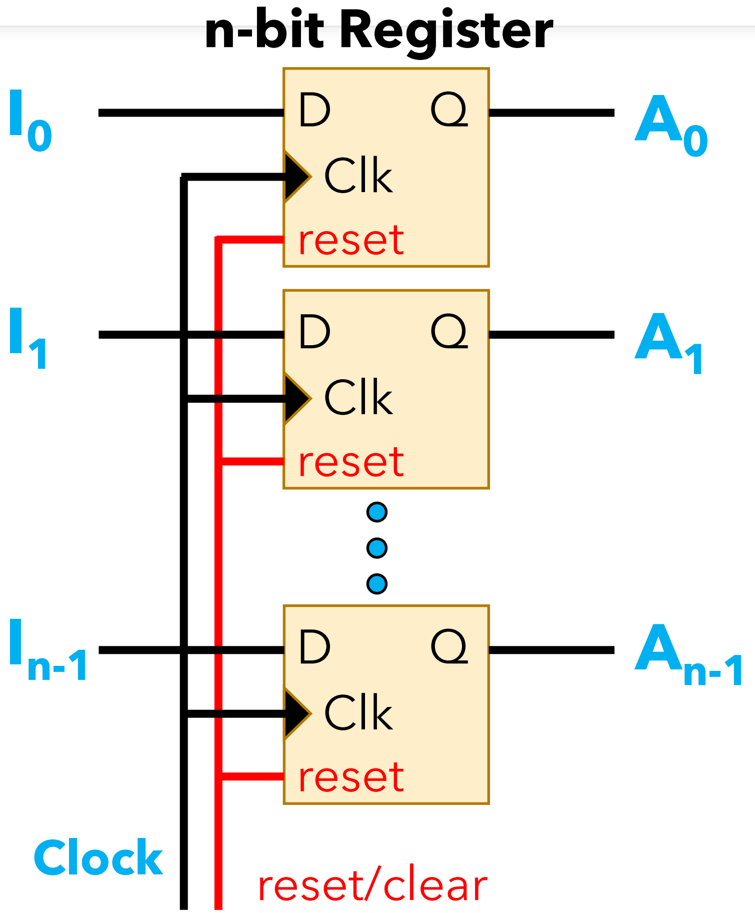


n-bit Register Symbol



Registers with Reset

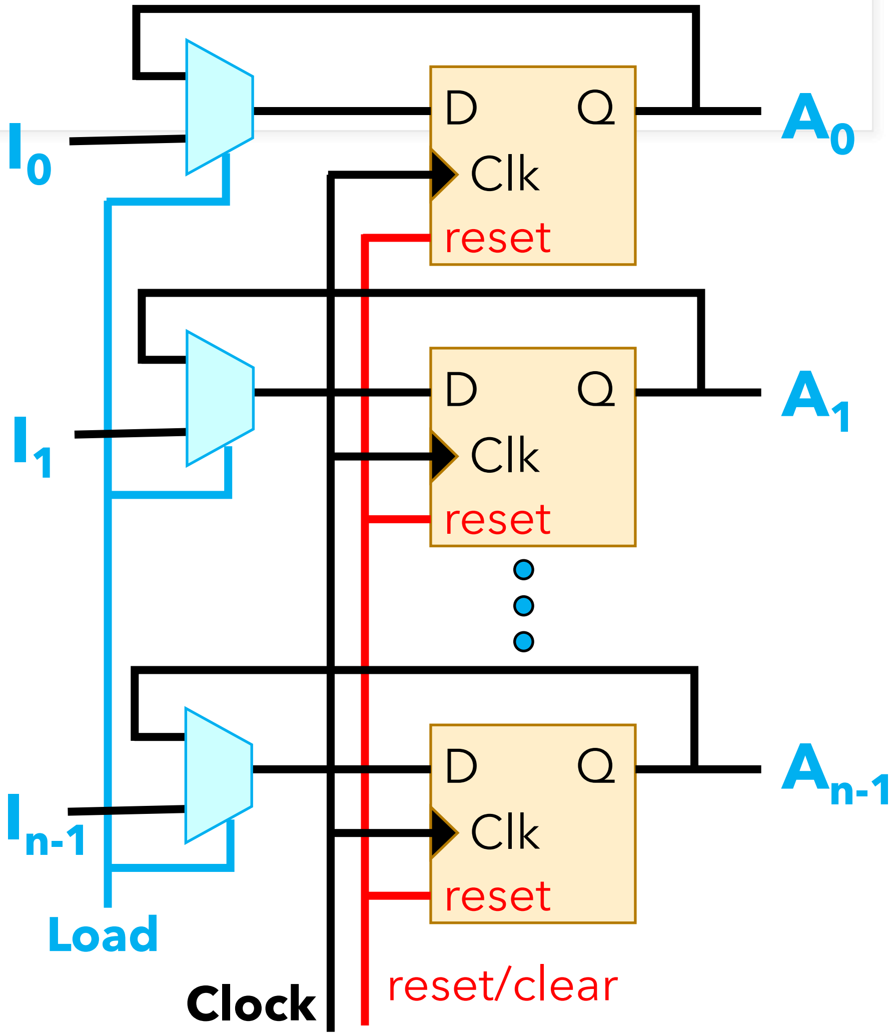
- Reset/Clear signal **asynchronously** clears **A** to **0**
  - independent of Clk
  - using DFF with asynchronous Reset



Registers with **Load** signal

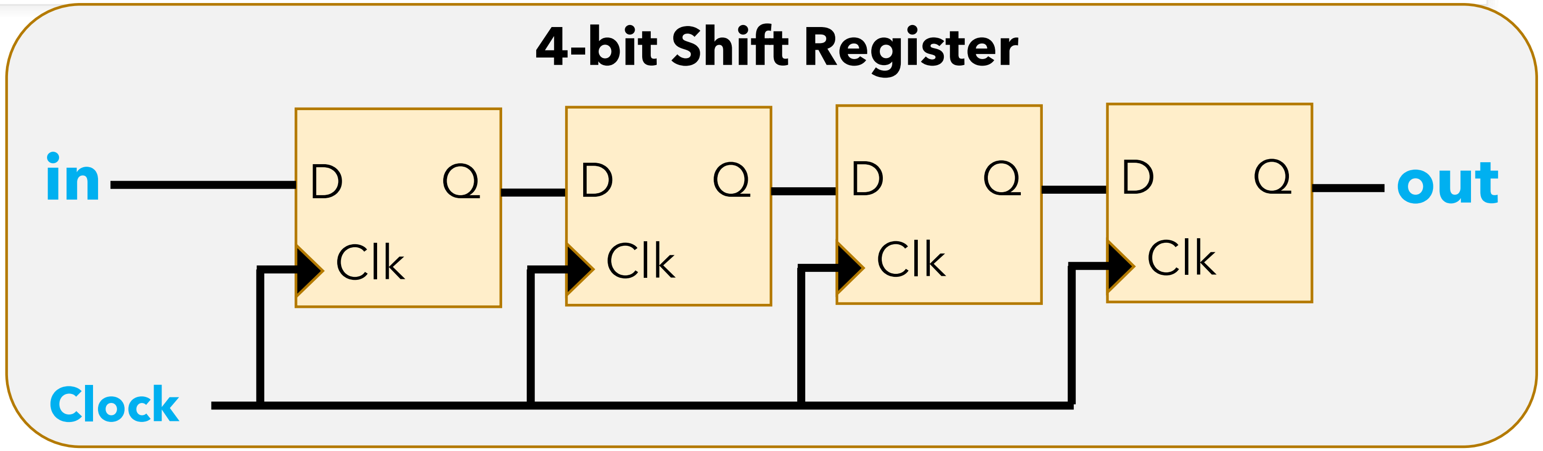
- Register function:
  - Transfers I to A on every clock
  - Called **Loading** new value to register (or **updating** the register)
- Desired function:
  - Load new value **only when required**
  - New control signal **Load**
- **Modify D input**
  - **Not clock** (don't disturb clock, causes uneven propagation delays)

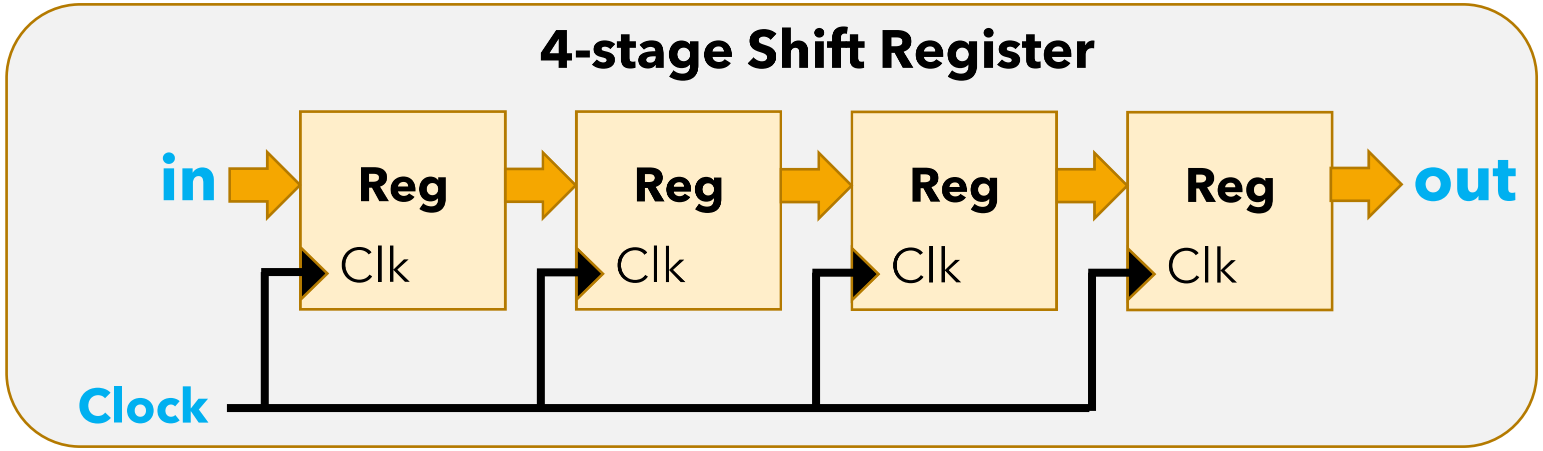
n-bit Register



Shift Registers

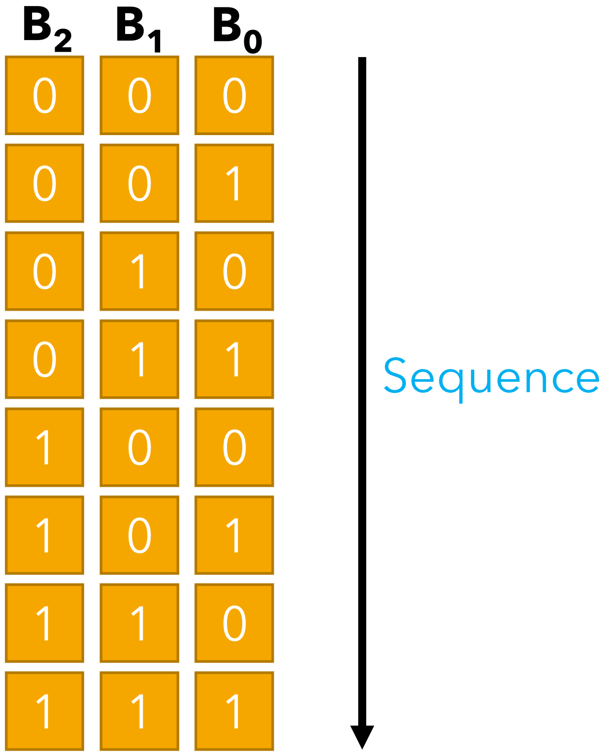
- **Cascade/Chain:** Q of one stage connected to D of next stage
- Common **Clock**
- **Shifts** bit to next DFF on clock edge
- General: Chained **data registers** (n-bits wide)





Counters

- Register going through a given **sequence of states on input pulse**
- Already studied: **mod 3 counter**
  - Sequence: 0,1,2,0,1,2,0,1,2
- Counting on common **clock pulse**: synchronous counter (e.g., mod 3 counter)
- Pulse could be internal signal: ripple counter
- Counter value on **Q of DFFs**



B_2	B_1	B_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Counters

- How do we generate sequence for DFF **B_0**?
  - Alternating Sequence
- How do we generate sequence for DFF **B_1**?
  - Alternating Sequence
  - Triggered by?
- How do we generate sequence for DFF **B_2**?
  - Alternating Sequence
  - Triggered by?

DFF:	B_2	B_1	B_0
	0	0	0
	0	0	1
	0	1	0
	0	1	1
	1	0	0
	1	0	1
	1	1	0
	1	1	1

Sequence ↓

Counters

- How do we generate sequence for DFF $\mathbf{B_0}$?
  - Alternating Sequence
  - Triggered by **External Count Signal**
- How do we generate sequence for DFF $\mathbf{B_1}$?
  - Alternating Sequence
  - Triggered by **Negative edge of $\mathbf{B_0}$**
- How do we generate sequence for DFF $\mathbf{B_2}$?
  - Alternating Sequence
  - Triggered by **Negative edge of $\mathbf{B_1}$**

DFF:	$\mathbf{B_2}$	$\mathbf{B_1}$	$\mathbf{B_0}$
	0	0	0
	0	0	1
	0	1	0
	0	1	1
	1	0	0
	1	0	1
	1	1	0
	1	1	1

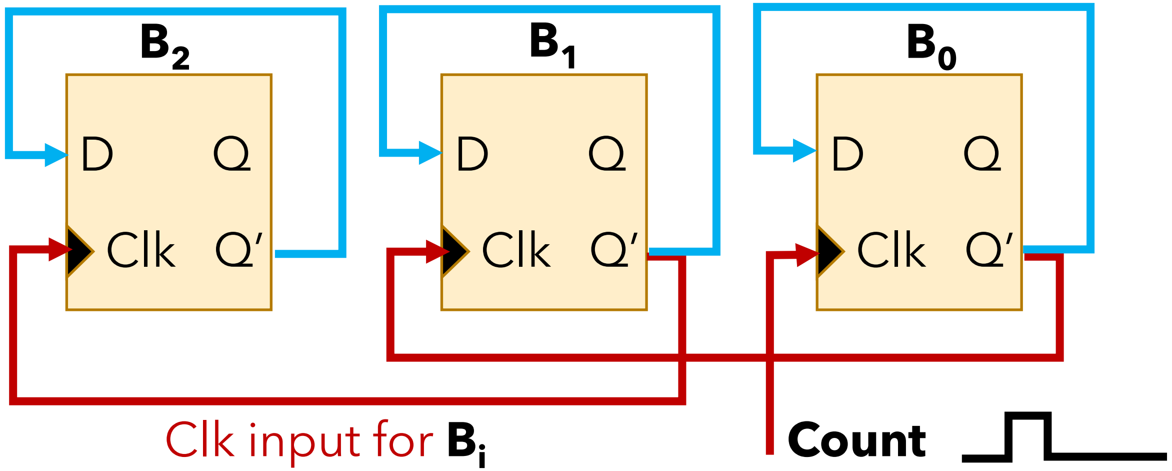
Sequence

Ripple Counters

DFF:	B_2	B_1	B_0
	0	0	0
	0	0	1
	0	1	0
	0	1	1
	1	0	0
	1	0	1
	1	1	0
	1	1	1

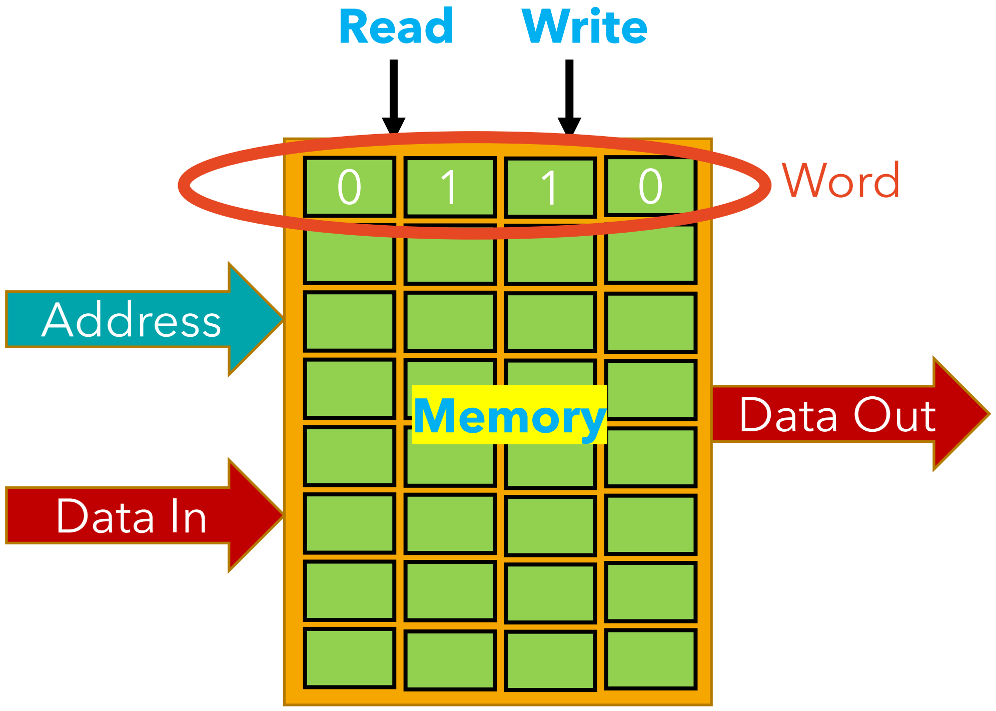
Sequence

Toggles DFF output

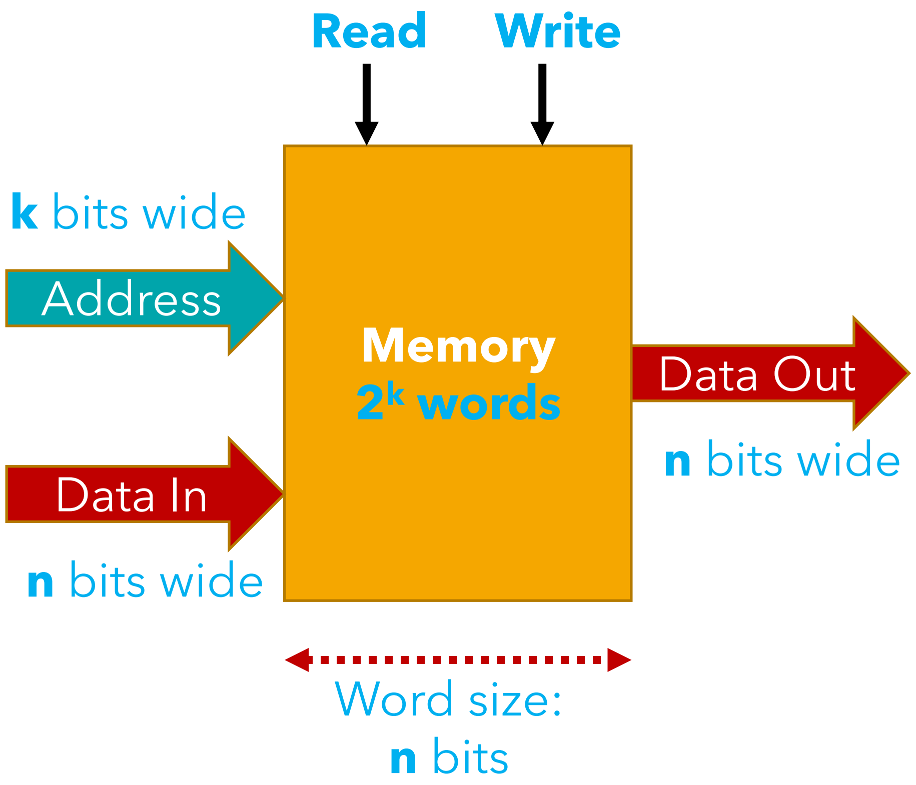


Clk input for $\mathbf{B_i}$
from **Q'** of $\mathbf{B_{i-1}}$
(negative edge)

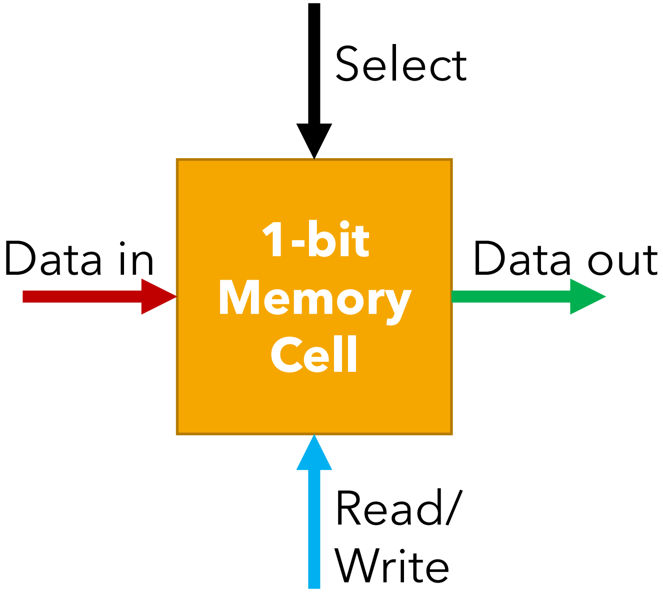
Recall: Memory Interface and Function

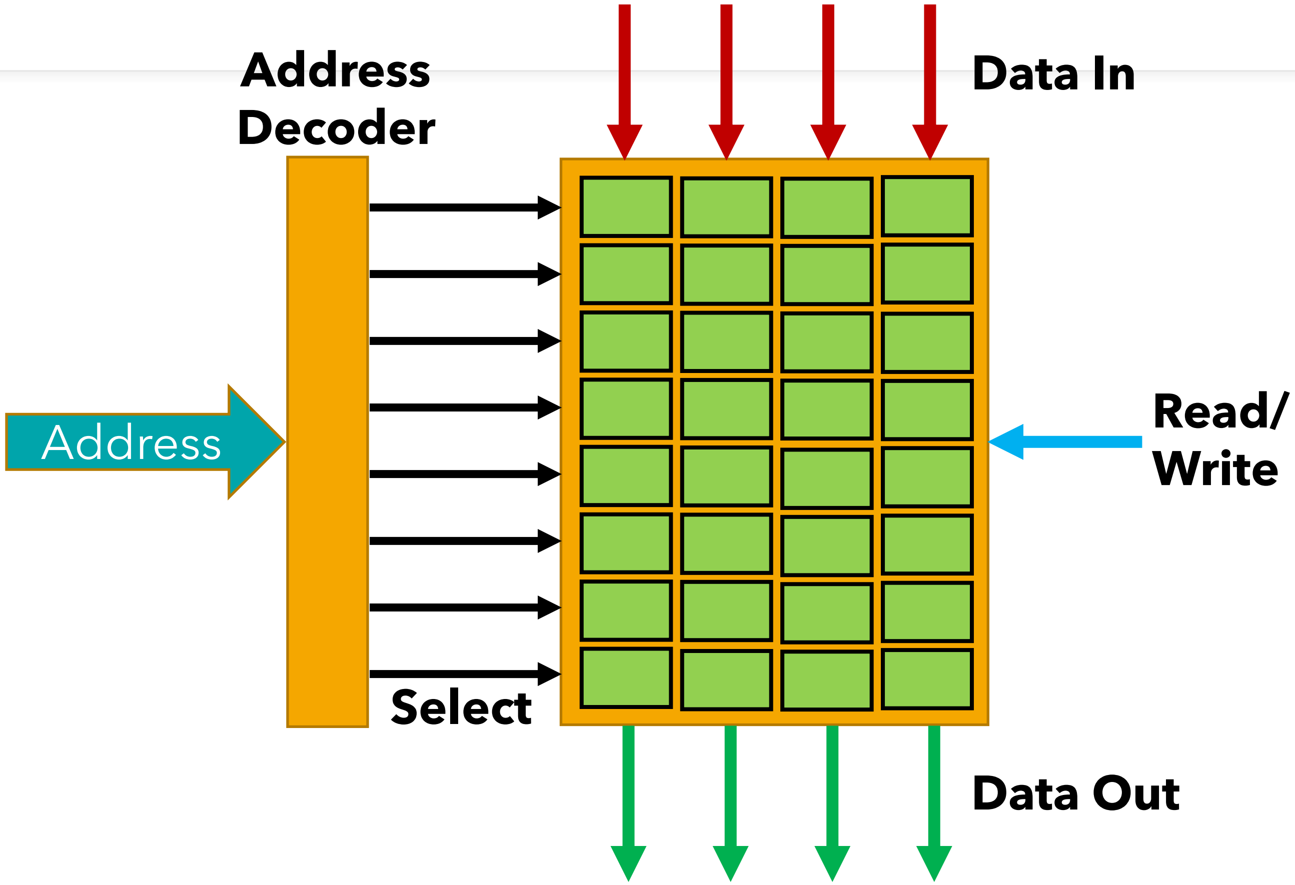


Word Size: 4 bits
Memory Size: 8 words

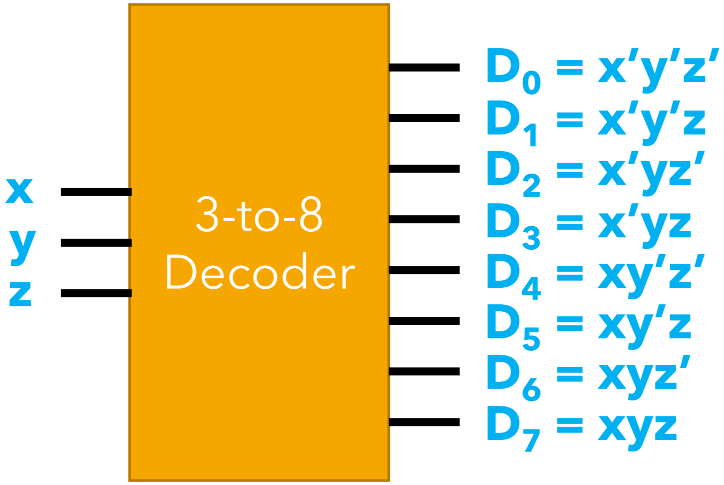


Memory Cell


Select
Data in
1-bit Memory Cell
Data out
Read/ Write


Address Decoder
Data In
Address
Read/ Write
Select
Data Out

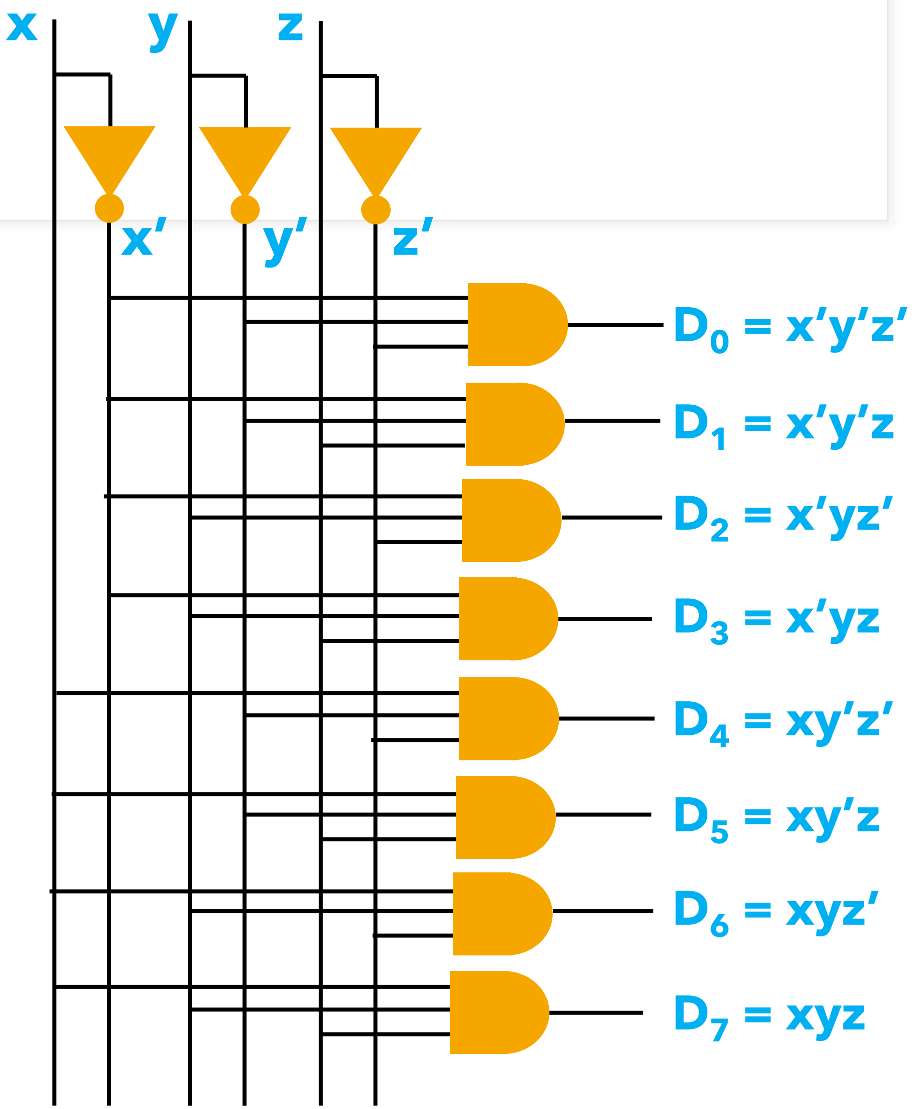
Recall: Decoder Implementation



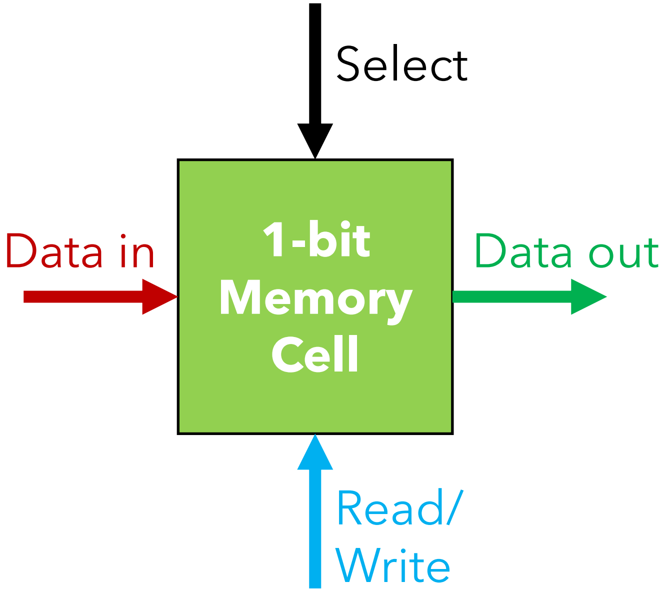
Each output is a **minterm**

Truth Table?

a	b	c	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

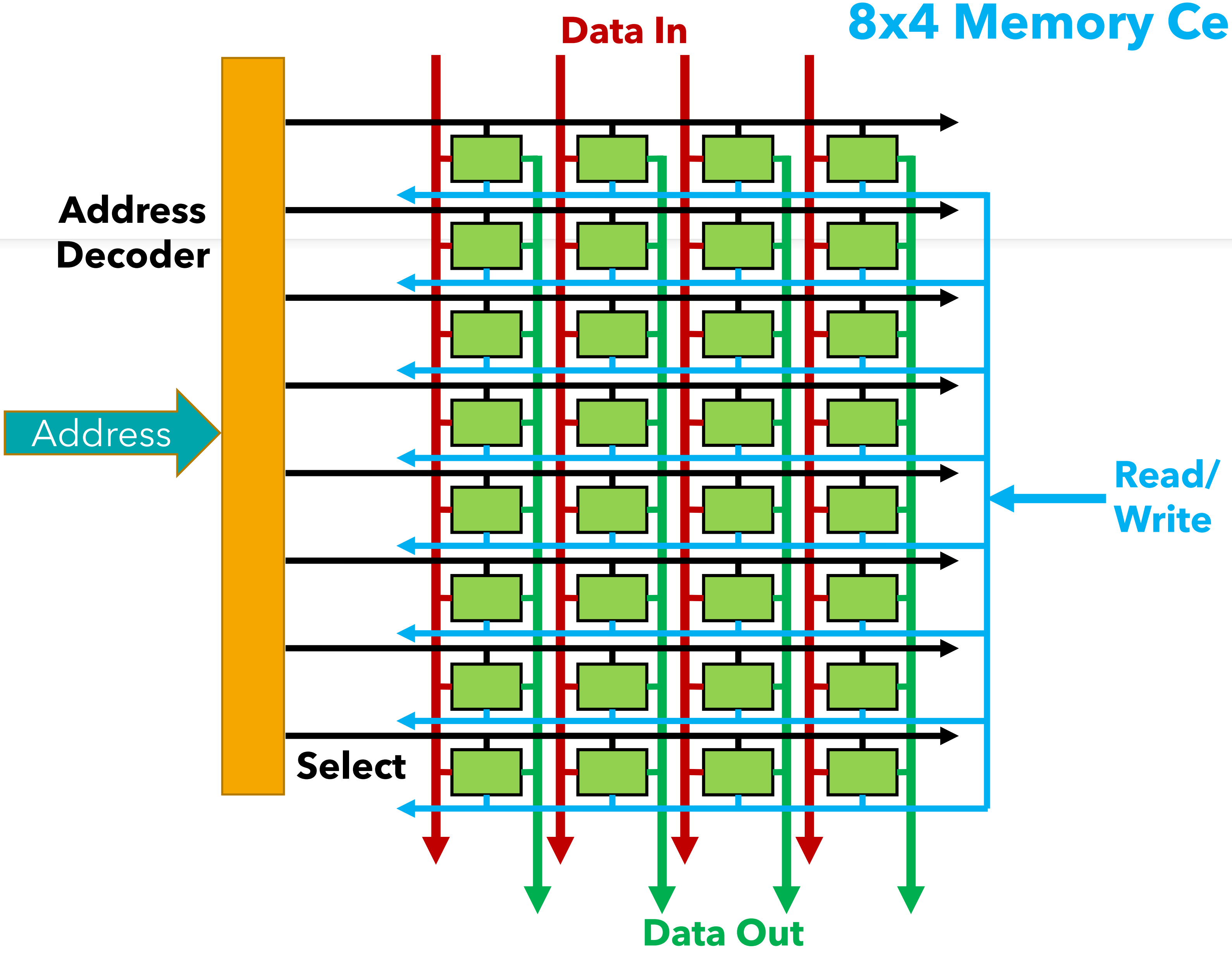


Memory Cell and Array



Memory Cell

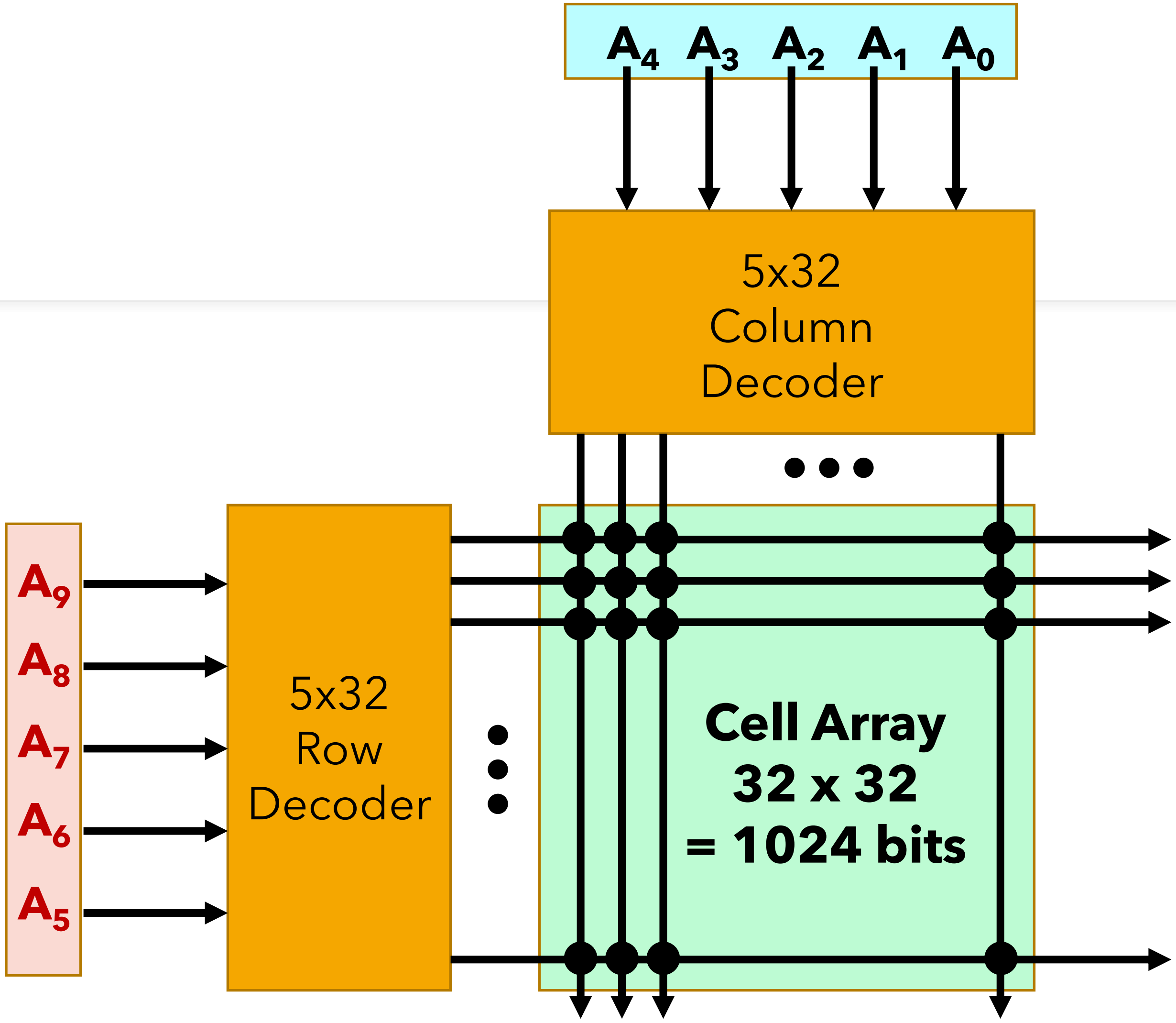
Memory Array
8x4 Memory Cells



2D Memory Layout

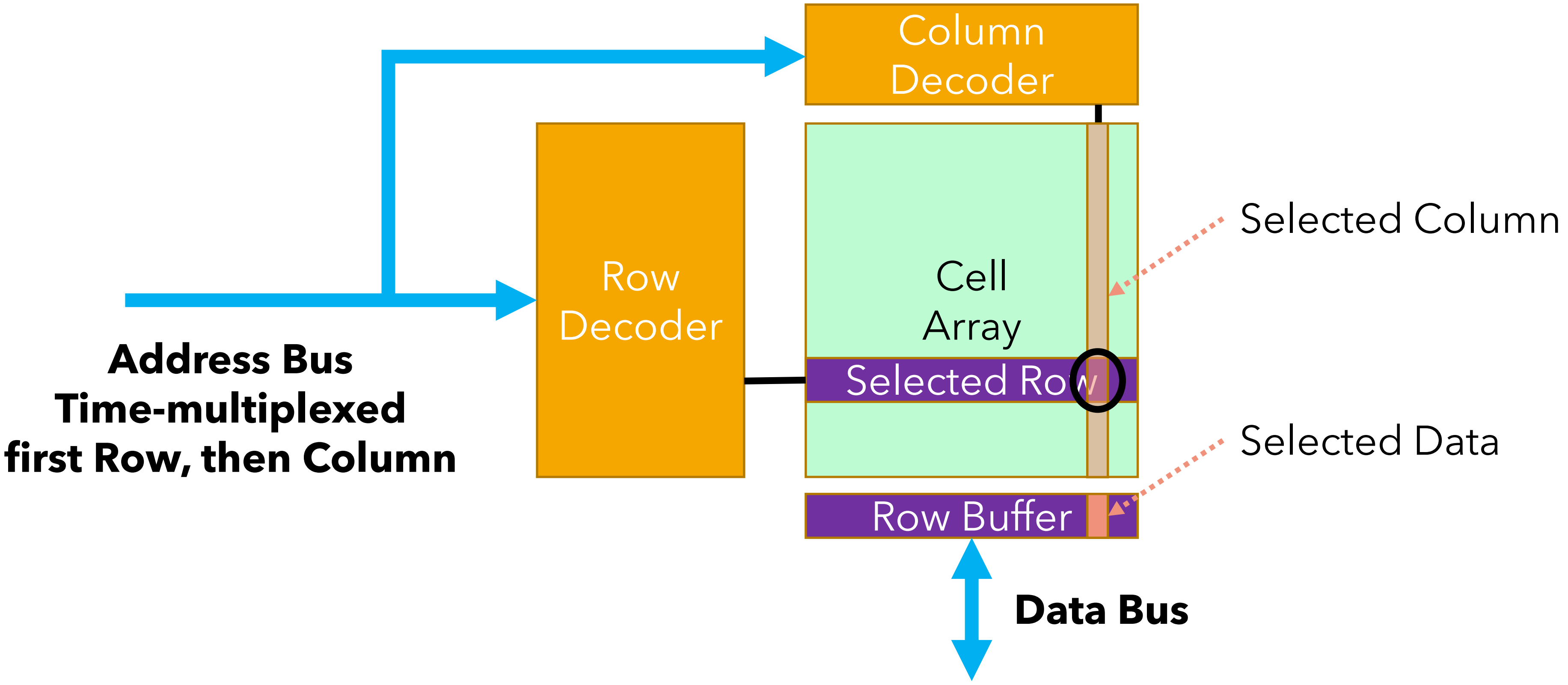
Address
A_9-A_0

A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0



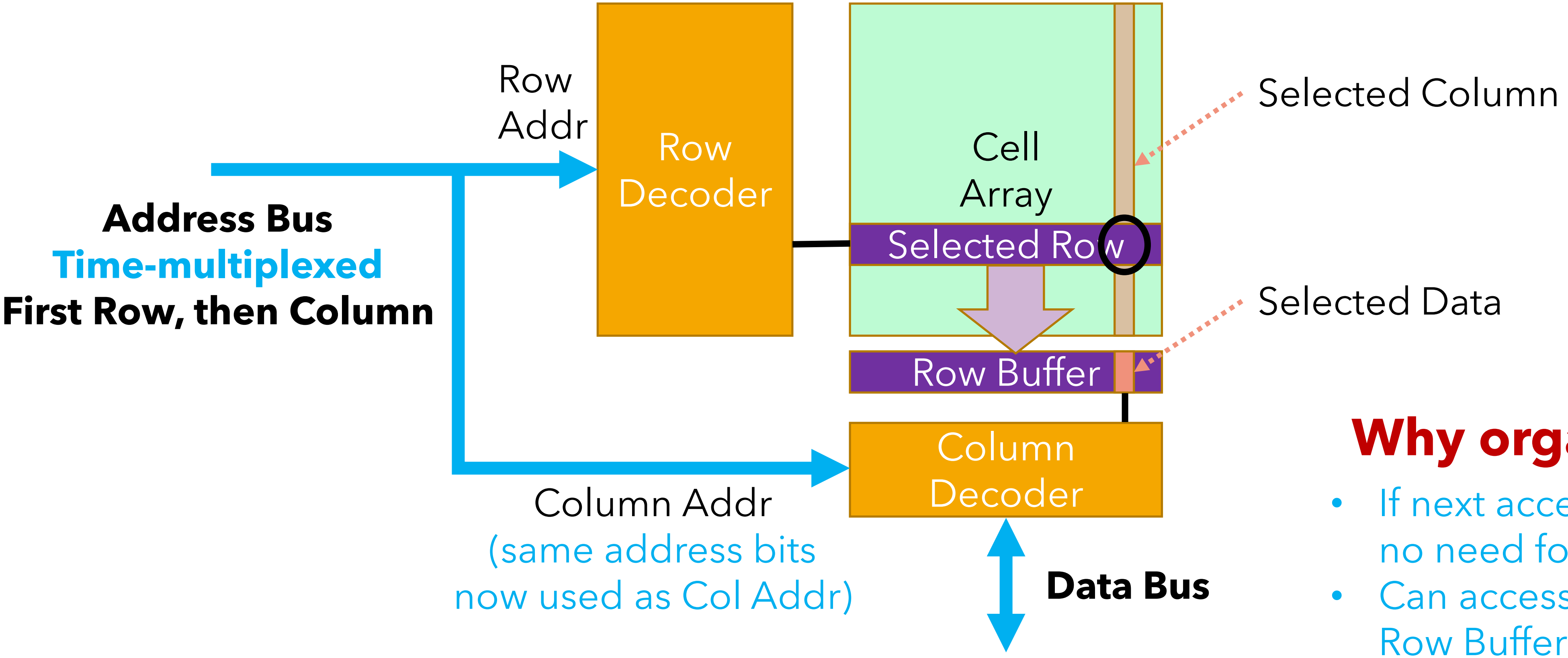
Data Arranged in Square/Rectangle

DRAM Addressing



Address Bus
Time-multiplexed
first Row, then Column

DRAM Addressing



Address Bus
Time-multiplexed
First Row, then Column

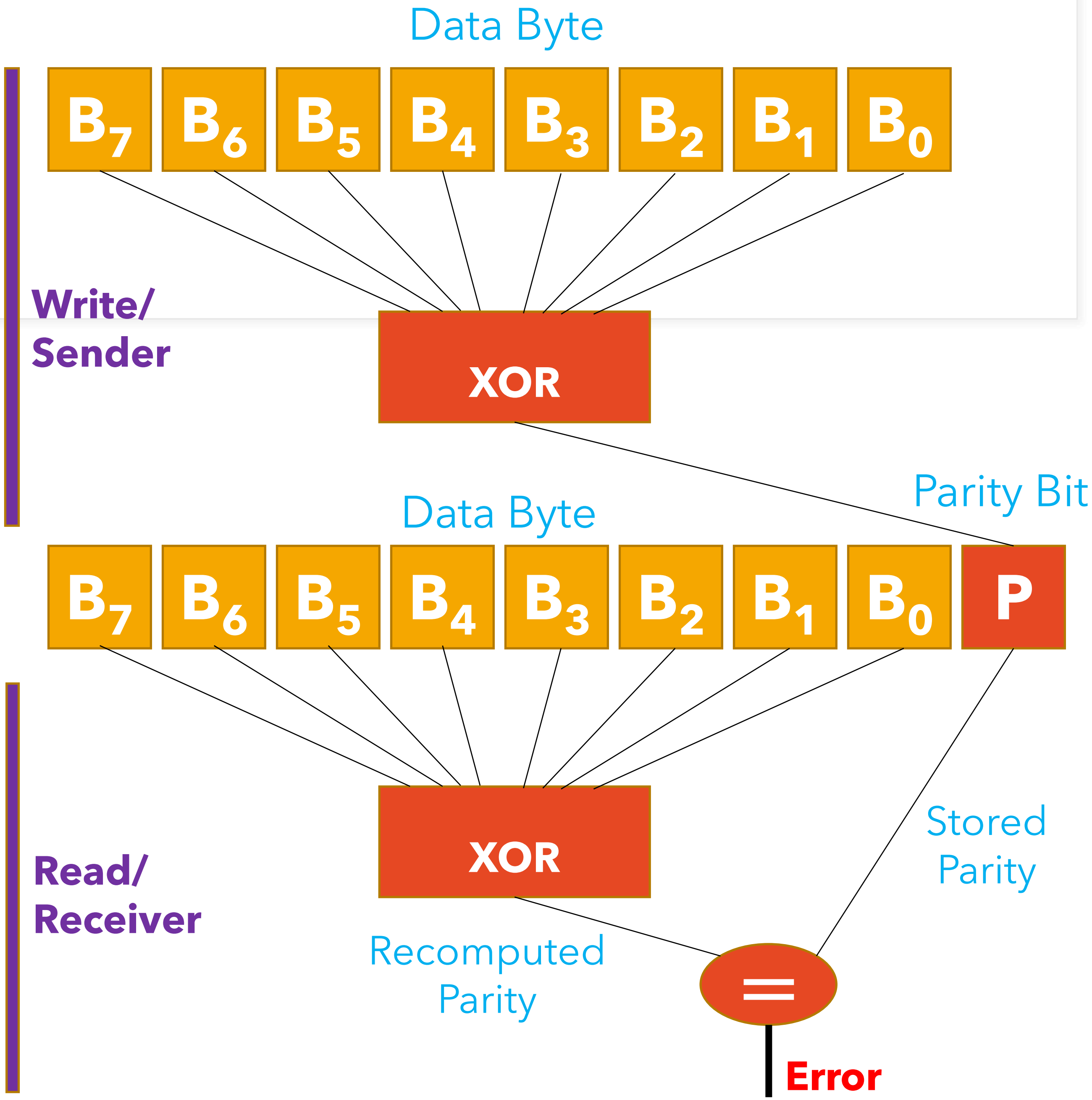
(same address bits now used as Col Addr)

Why organize this way?

- If next access is to same row, no need for Row Decoding
- Can access data directly from Row Buffer. Only Col Decoding

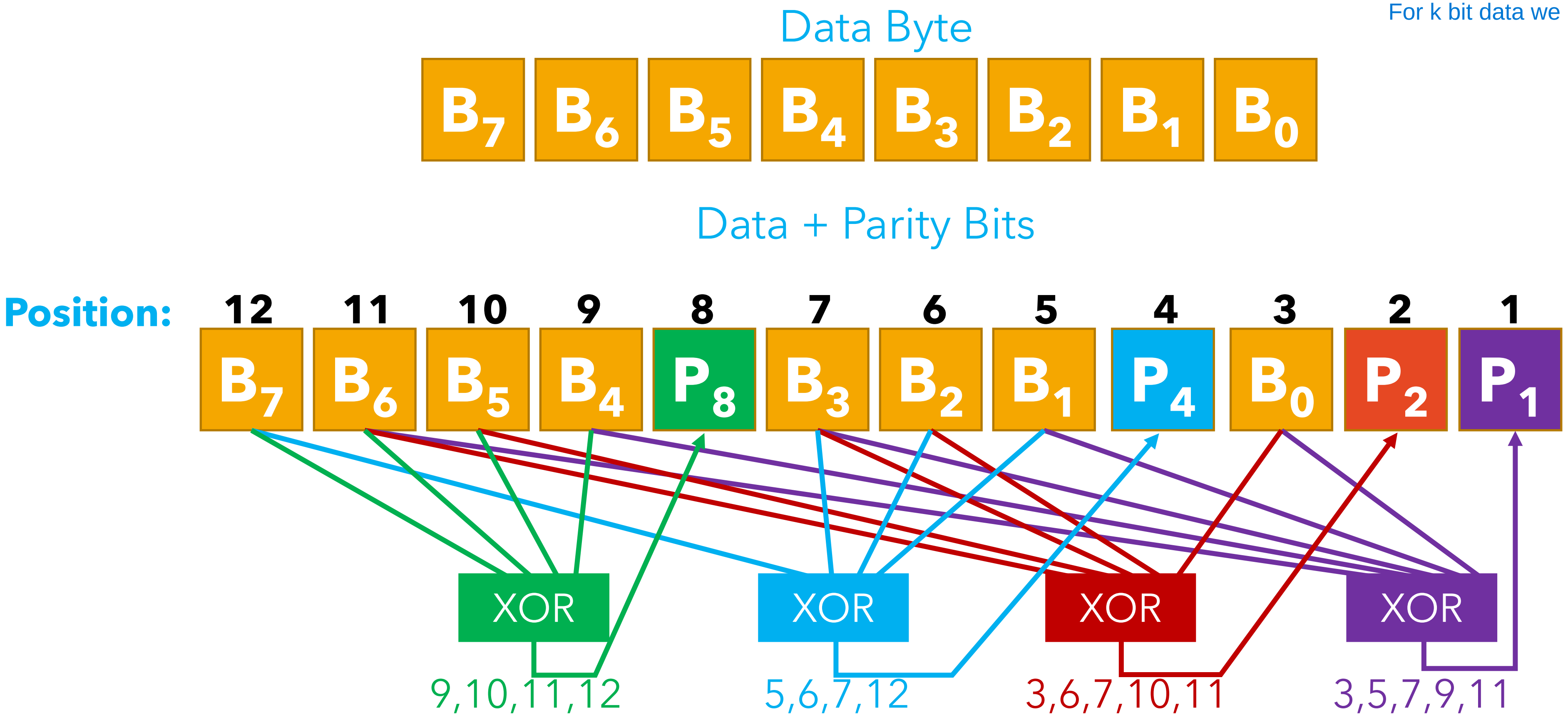
Error Detection

- Errors might occur in storage and transmission
- Error Detection: **Parity Bit**
- Parity: Additional bit stored along with data
- Parity **re-computed** by receiver, **compared** with stored parity
- If different, **error**



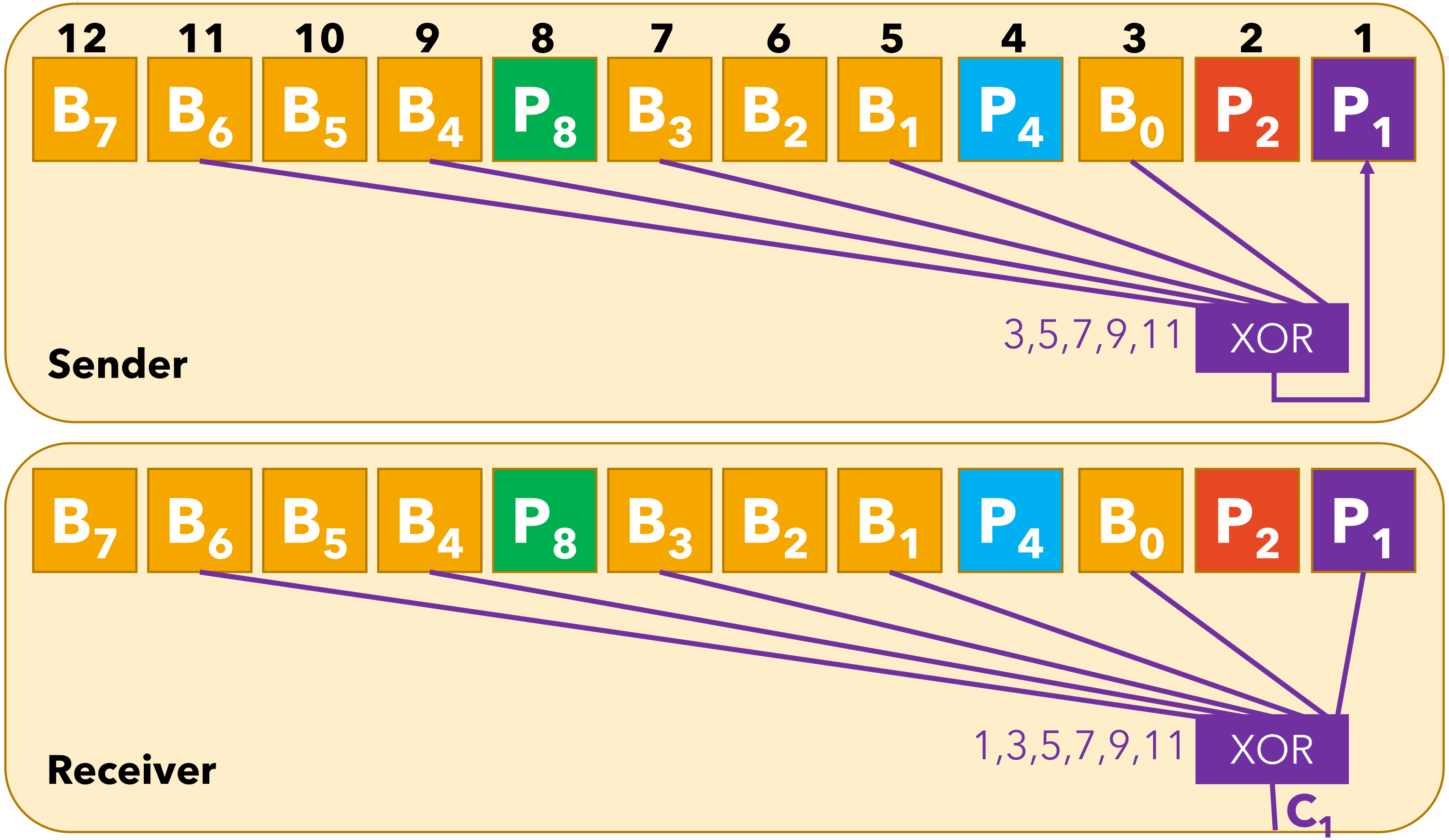
Error Correction with Hamming Codes

For k bit data we need log_2 k parity bits



Data Byte
B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0
Data + Parity Bits
Position:
12 11 10 9 8 7 6 5 4 3 2 1
B_7 B_6 B_5 B_4 P_8 B_3 B_2 B_1 P_4 B_0 P_2 P_1
XOR
XOR
XOR
XOR
9,10,11,12
5,6,7,12
3,6,7,10,11
3,5,7,9,11

Error Correction with Hamming Codes


12
11
10
9
8
7
6
5
4
3
2
1
B7 B6 B5 B4 P8 B3 B2 B1 P4 B0 P2 P1
3,5,7,9,11
XOR
Sender
B7 B6 B5 B4 P8 B3 B2 B1 P4 B0 P2 P1
1,3,5,7,9,11
XOR
C1
Receiver

$C_1 = ?$

0 if bits 1,3,5,7,9,11 **OK**

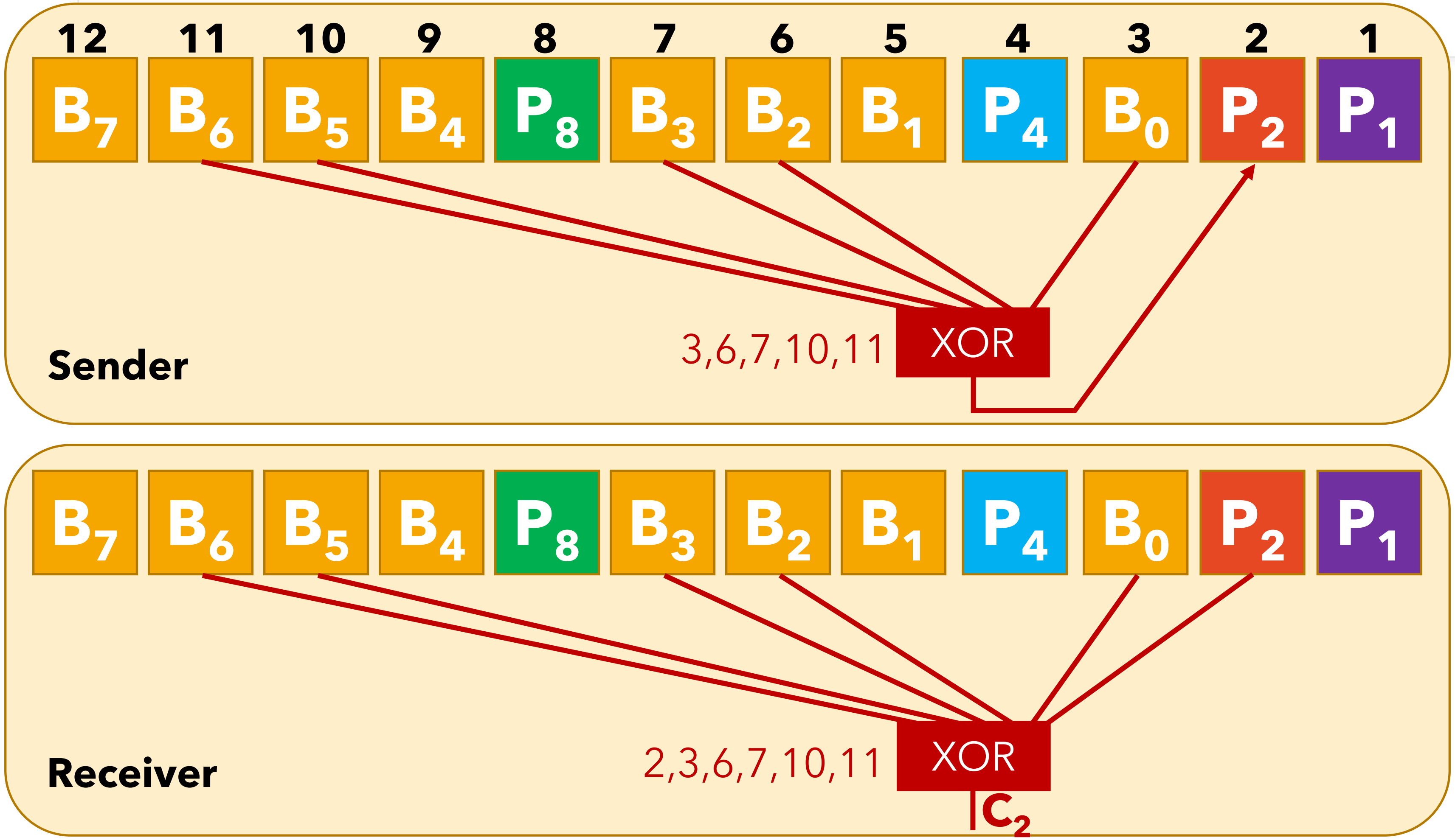
1 if **error** on any of bits 1,3,5,7,9,11

Including Parity Bit

Error type: single bit flip

(C) P. R. Panda, IIT Delhi, 2022

Error Correction with Hamming Codes

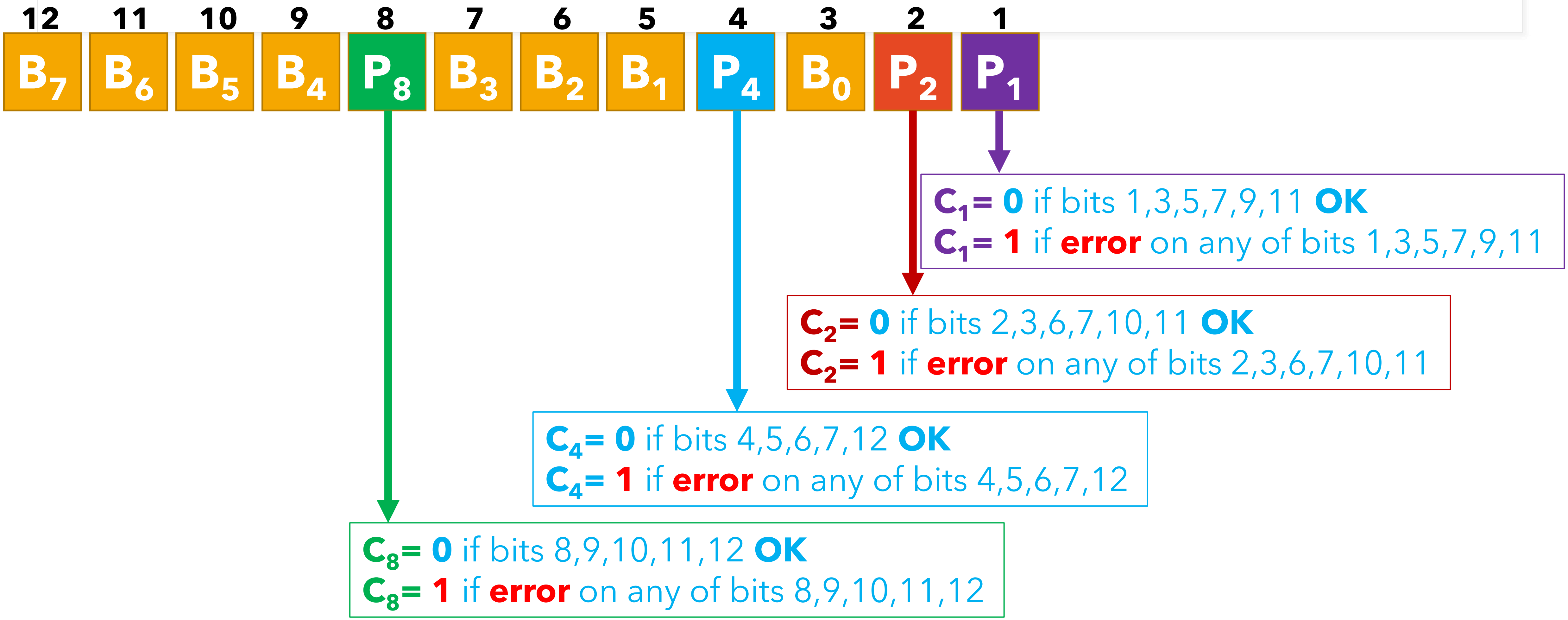


C_2= ?

0 if bits 2,3,6,7,10,11 **OK**

1 if **error** on any of bits 2,3,6,7,10,11

Error Correction with Hamming Codes



C_1= **0** if bits 1,3,5,7,9,11 **OK**

C_1= **1** if **error** on any of bits 1,3,5,7,9,11

C_2= **0** if bits 2,3,6,7,10,11 **OK**

C_2= **1** if **error** on any of bits 2,3,6,7,10,11

C_4= **0** if bits 4,5,6,7,12 **OK**

C_4= **1** if **error** on any of bits 4,5,6,7,12

C_8= **0** if bits 8,9,10,11,12 **OK**

C_8= **1** if **error** on any of bits 8,9,10,11,12

Error Correction with Hamming Codes

				C_1
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

C_1= **0** if bits 1,3,5,7,9,11 **OK**
C_1= **1** if **error** on any of bits 1,3,5,7,9,11

C_1 gives Bit 1 of the erroneous bit position

C_2= **0** if bits 2,3,6,7,10,11 **OK**
C_2= **1** if **error** on any of bits 2,3,6,7,10,11

C_2 gives Bit 2 of the erroneous bit position

			C_2	
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

Error Correction with Hamming Codes

		C_4		
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

C_4= **0** if bits 4,5,6,7,12 **OK**
C_4= **1** if **error** on any of bits 4,5,6,7,12

C_4 gives Bit 3 of the erroneous bit position

C_8= **0** if bits 8,9,10,11,12 **OK**
C_8= **1** if **error** on any of bits 8,9,10,11,12

C_8 gives Bit 4 of the erroneous bit position

$C_8C_4C_2C_1$ together give the position of the erroneous bit!
$C_8C_4C_2C_1$ = 0 means no error

Now we can perform Error Correction
(flip the erroneous bit)

	C_8			
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

Research Areas

- Effect of 3D stacking
- Effect of newer device technologies (e.g., non-volatile memory)
- Trade-offs: Power/Temperature vs. Performance