

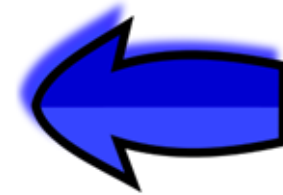


Chapter 7: A Primer On Digital Logic ✓

Basic Computer Architecture

Outline

*Transistors and Gates



*Combinational Logic

* Sequential Logic

* SRAM/ DRAM Cells

Disclaimer: This chapter is only to



get a high level overview ...

- * We assume some **background** in logic gates, transistors, combinational and sequential logic
- * The **aim** of this chapter is to only provide a [high level overview](#)
- * For a **deeper** understanding consult any of the classic textbooks on digital logic

Atoms and Molecules of Circuits

The Transistor

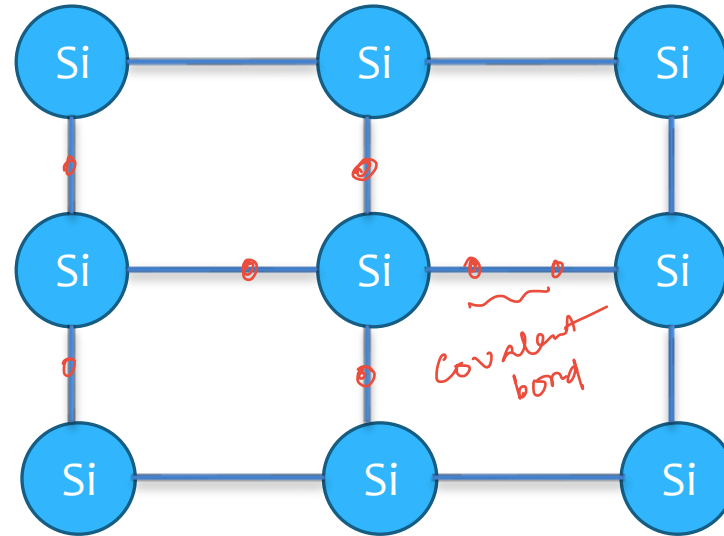


- It is just a **switch**.
- It is either switched **on** (current can flow), or switched **off** (no current flow)

How is a Transistor Made?

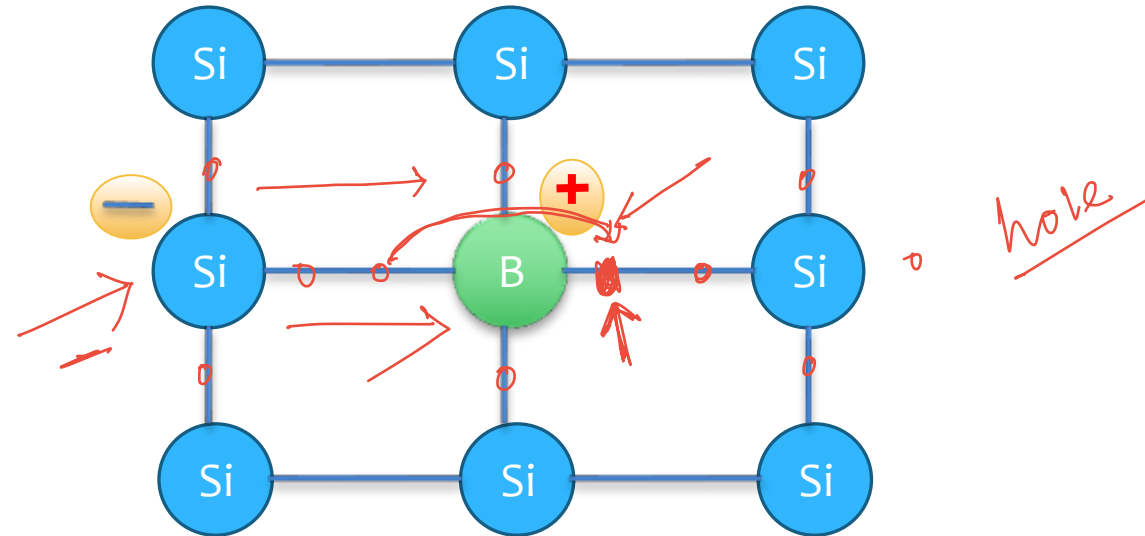
- * It is made of **Silicon**
 - * Silicon is a semi-conductor (neither conductor or insulator).
 - * We can change its properties:
 - * Add a little bit of impurities → **doping**
- * Dope it with Group III **elements (3 valence electrons)**
 - * Boron, Aluminum and Gallium
 - * It is called a **p-type** semiconductor
- * Or, dope it with Group V **elements (5 valence electrons)**
 - * Phosphorus or Arsenic
 - * It is called a **n-type** semiconductor

Silicon Lattice



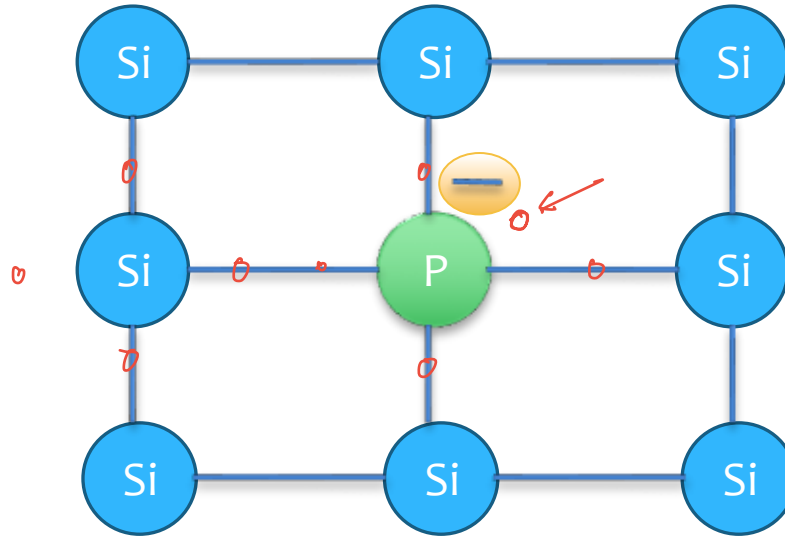
- * This is a typical silicon lattice. Each Si atom is connected to 4 other atoms via covalent bond

P-Type Doping



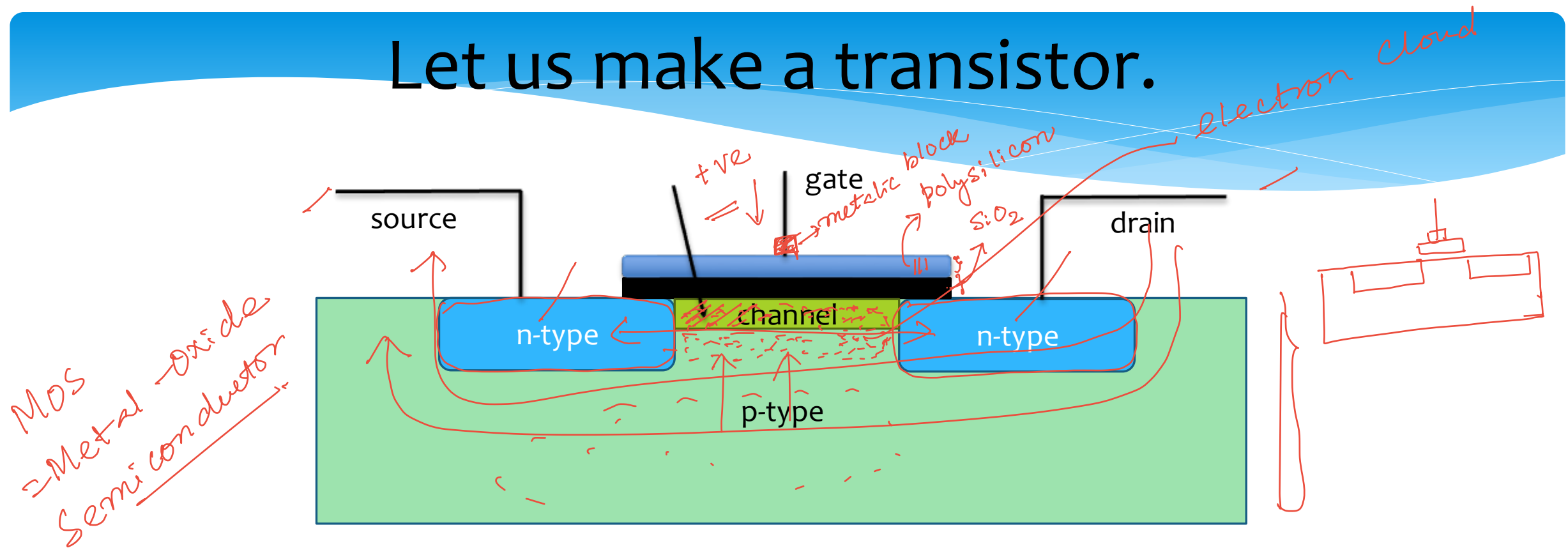
- * If there is a Boron atom in the **lattice**. It will create bonds with the rest of the atoms.
- * There will be one less electron (**hole**)
- * Holes can **flow**. They are associated with +ve charge. (see the animation)

N-Type Doping



- * If there is a Phosphorus atom in the **lattice**. It will create bonds with the rest of the atoms.
- * There will be one more electron (**electron**)
- * Electrons can **flow**. They are associated with -ve charge. (see the animation)

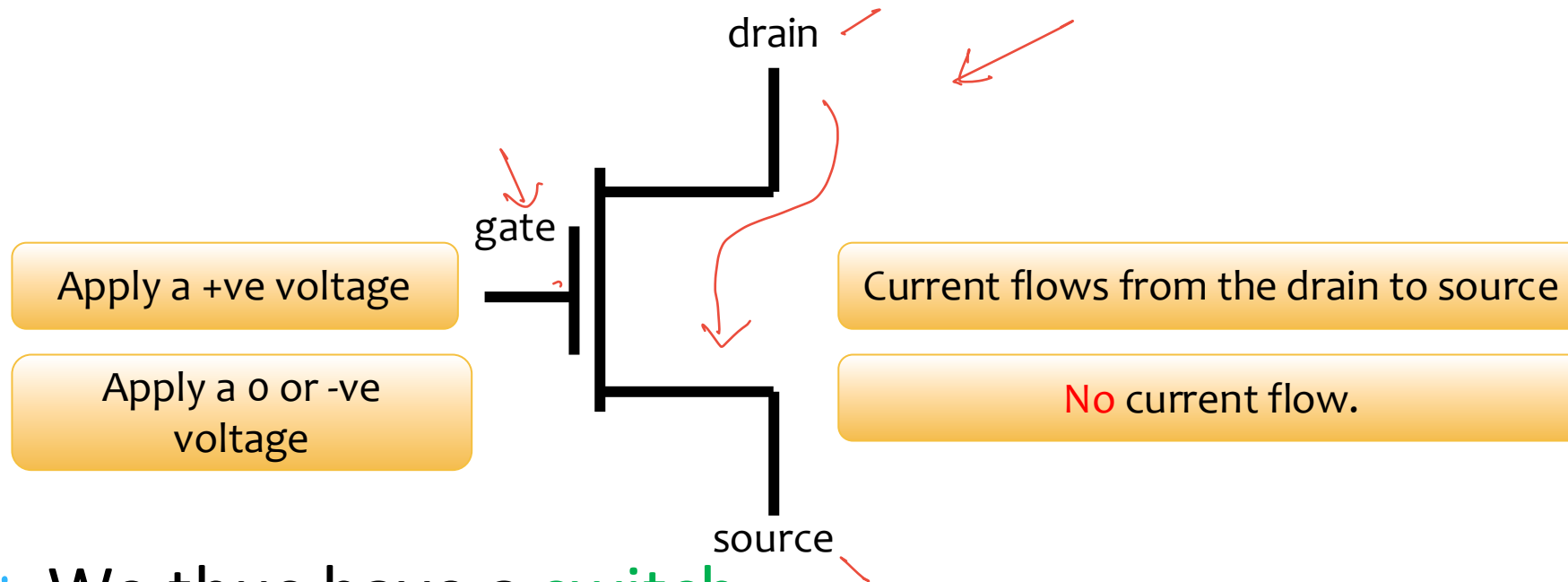
Let us make a transistor.



- * NMOS transistor → Put two **n-type** wells in a **p-type** substrate
- * Now, assume that we apply **+ve** charge to the gate.
- * Channel → This forms because electrons move towards the positive charge. Forms a conductive layer that can conduct **current**.

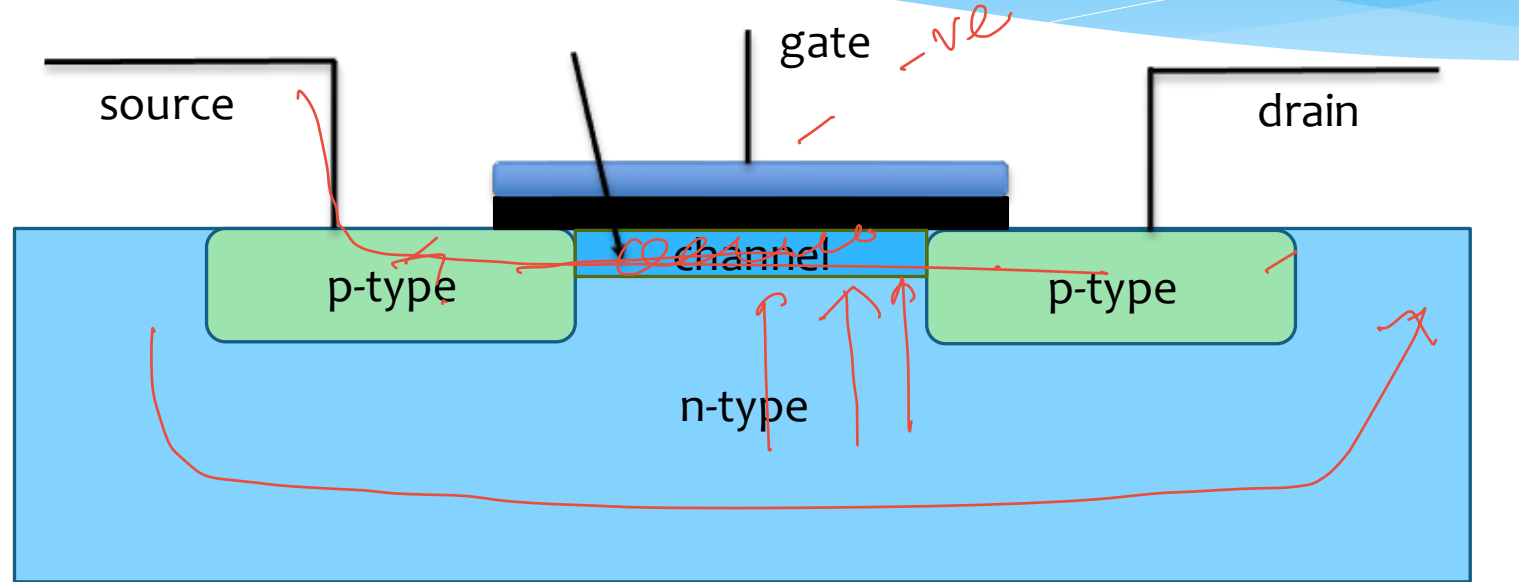
NMOS Transistor

Metal-Oxide-Semiconductor



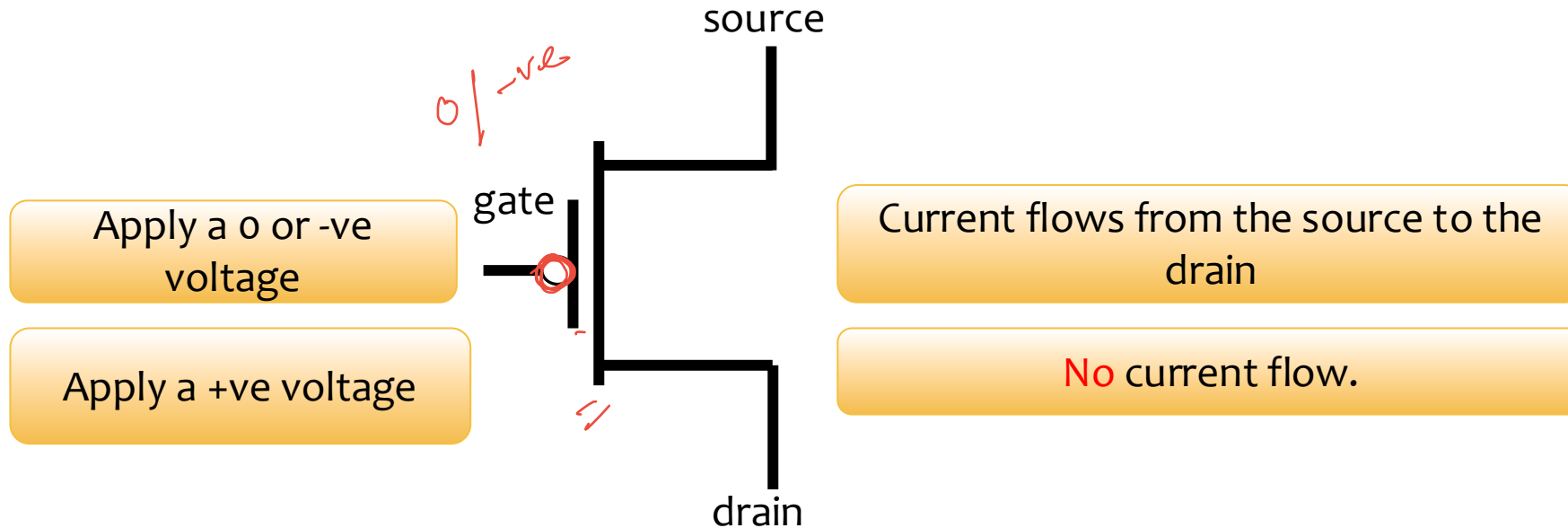
- * We thus have a **switch**
 - * Apply a **+ve** voltage at the gate → make the transistor conduct
 - * Apply a **-ve** or 0 voltage at the gate → transistor is off (no current flow across it)

PMOS Transistor



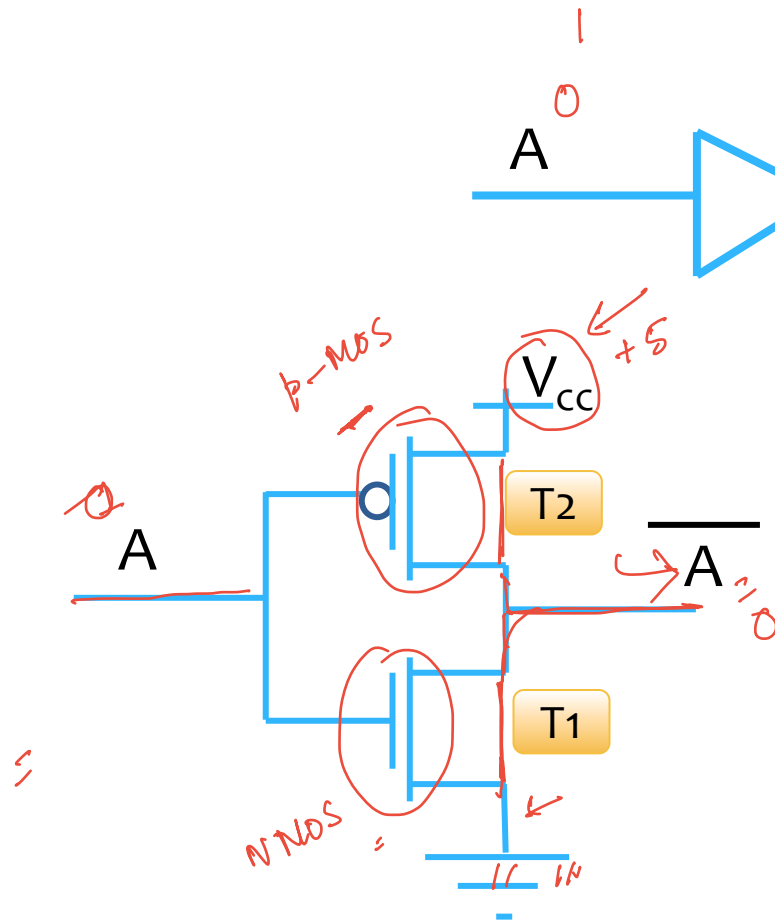
- * PMOS transistor → Put two **p-type** wells in an **n-type** substrate
- * Now, assume we apply **0 or -ve** charge to the gate.
- * Channel → This forms because holes move towards the gate. This channel can conduct **current**.

PMOS Transistor



- * We thus have a **switch**
 - * Apply a **+ve** voltage at the gate \rightarrow transistor is off (no current flow across it)
 - * Apply a **-ve** or 0 voltage at the gate \rightarrow transistor conducts

Let us make an inverter

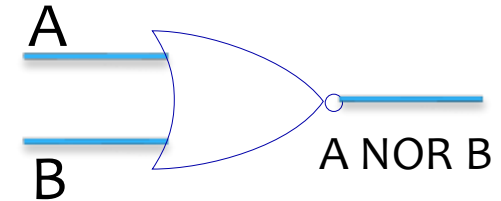
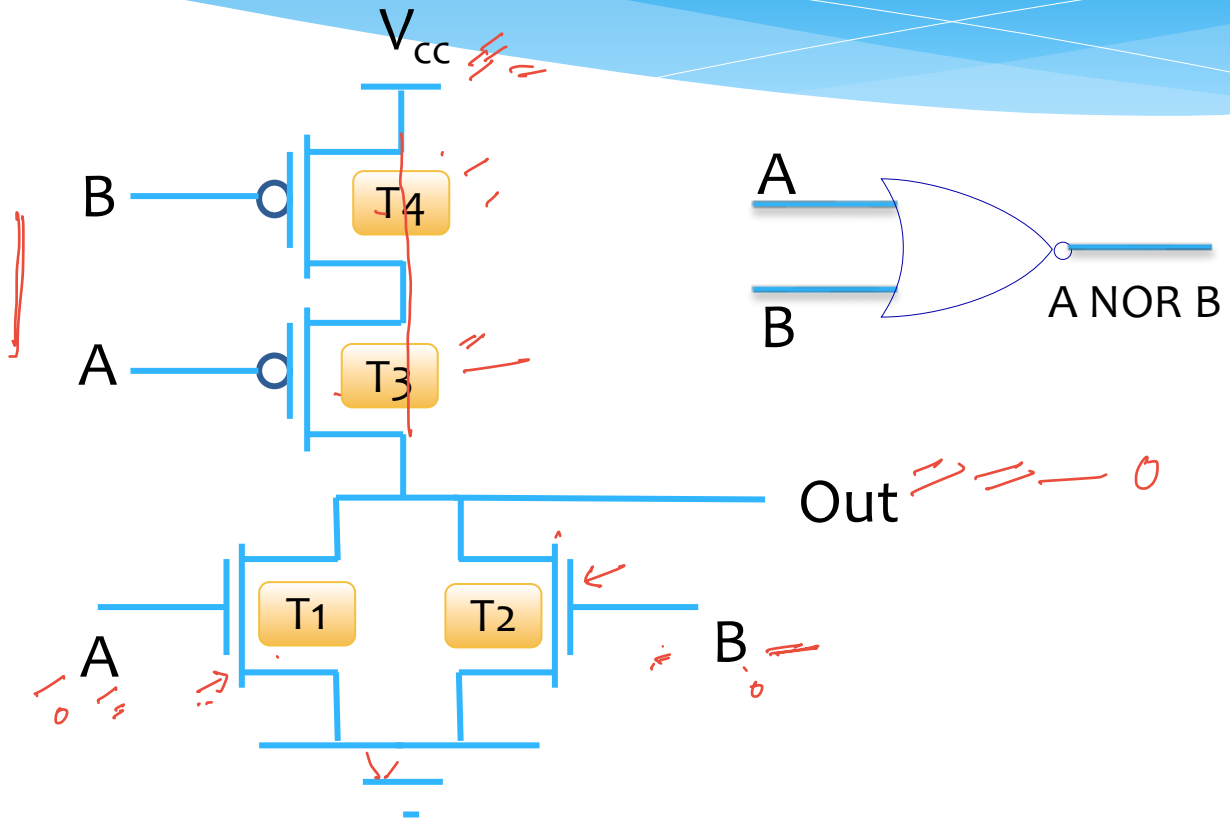


If $A = 1$ (+ve voltage)
T1 is on, T2 is off
Thus, the output will be connected to the ground, and it will be a logical 0

If $A = 0$ (0 voltage)
T1 is off, T2 is on
Thus, the output will be connected to the supply (V_{cc}), and it will be a logical 1

Complementary Metal-Oxide-Semiconductor

NOR Gate



A	B	T1	T2	T3	T4	Out
0	0	off	off	on	on	1
1	0	on	off	off	on	0
0	1	off	on	on	off	0
1	1	on	on	off	off	0

NAND

O/P

A B

0 0

0 1

1 0

1 1

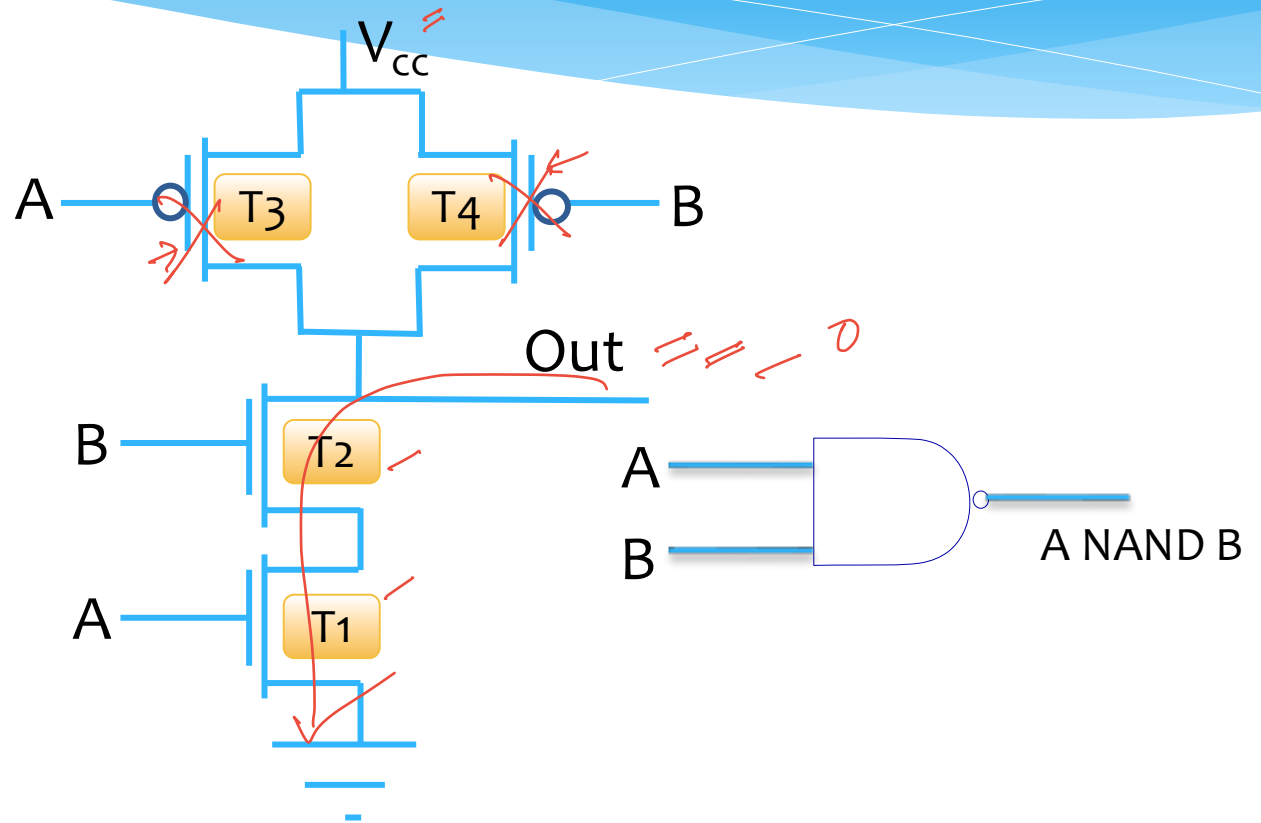
1

1

0

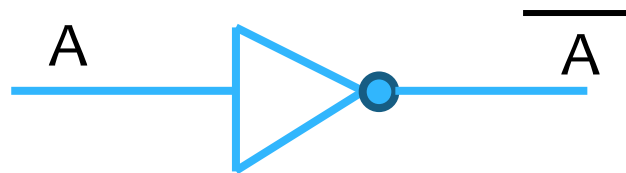
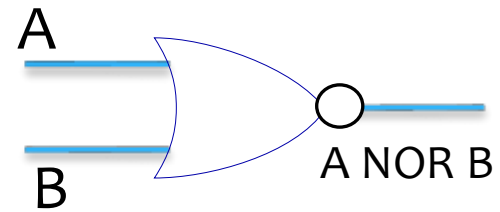
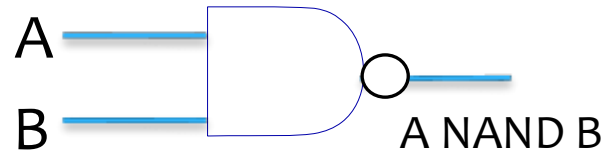
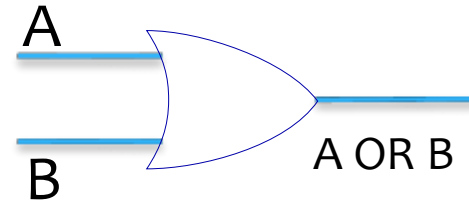
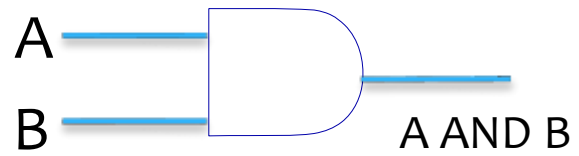
0

NAND Gate



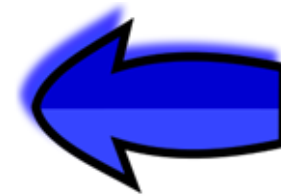
A	B	T1	T2	T3	T4	Out
0	0	off	off	on	on	1
1	0	on	off	off	on	1
0	1	off	on	on	off	1
1	1	on	on	off	off	0

Summary of Logic Gates



Outline

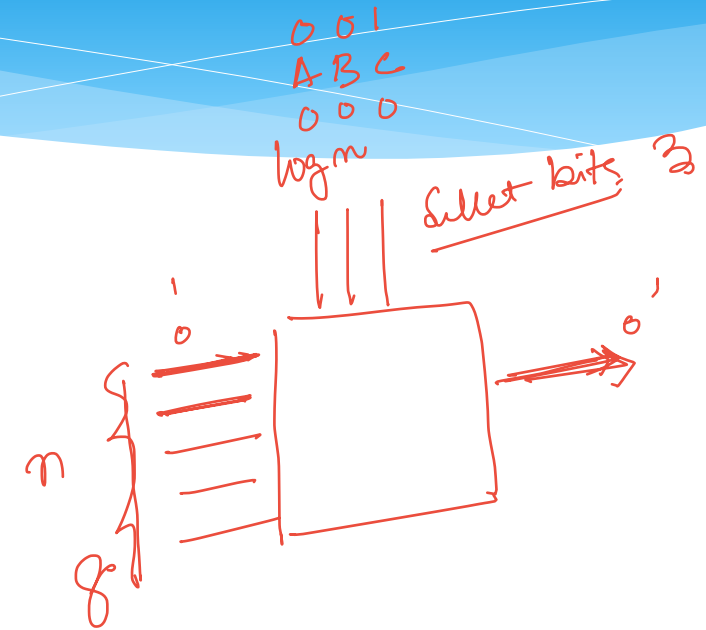
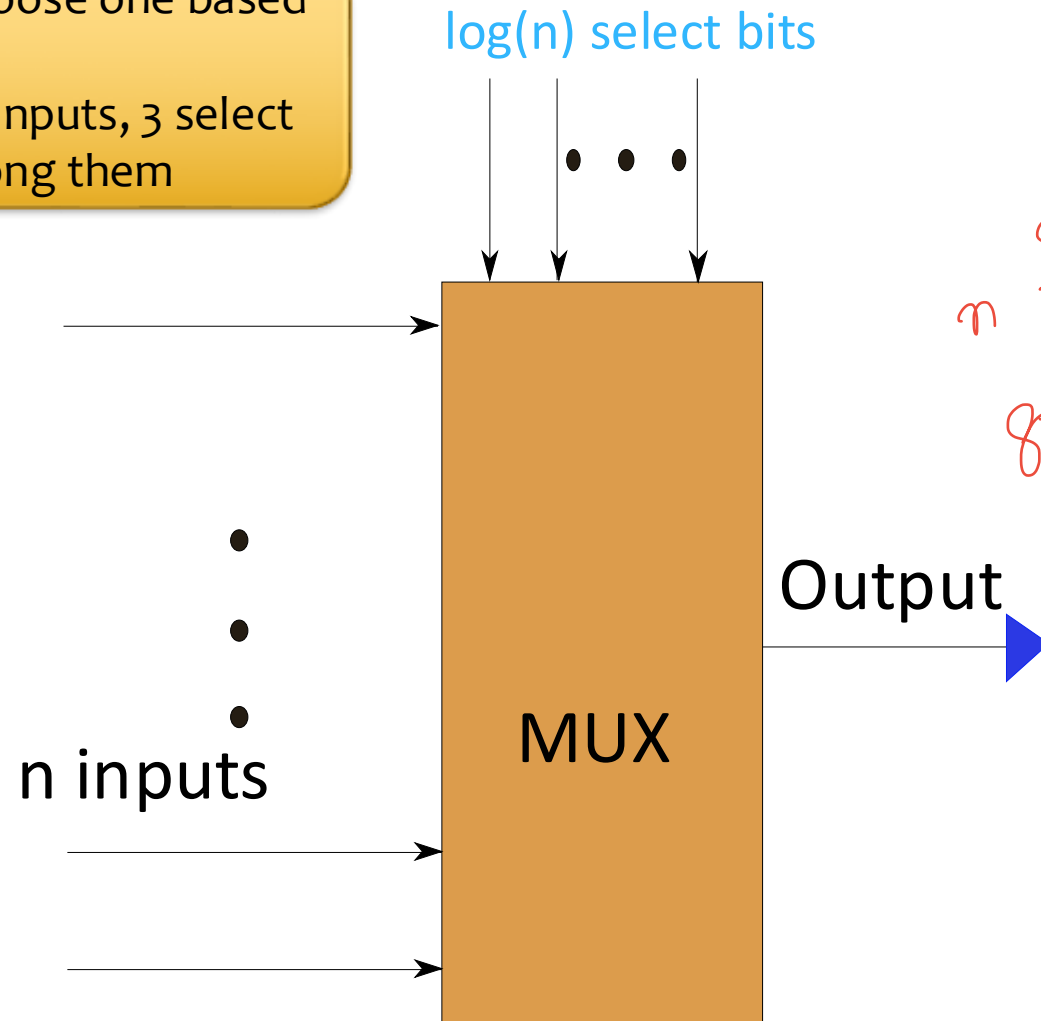
- * Transistors and Gates
- * Combinational Logic
- * Sequential Logic
- * SRAM/ DRAM Cells



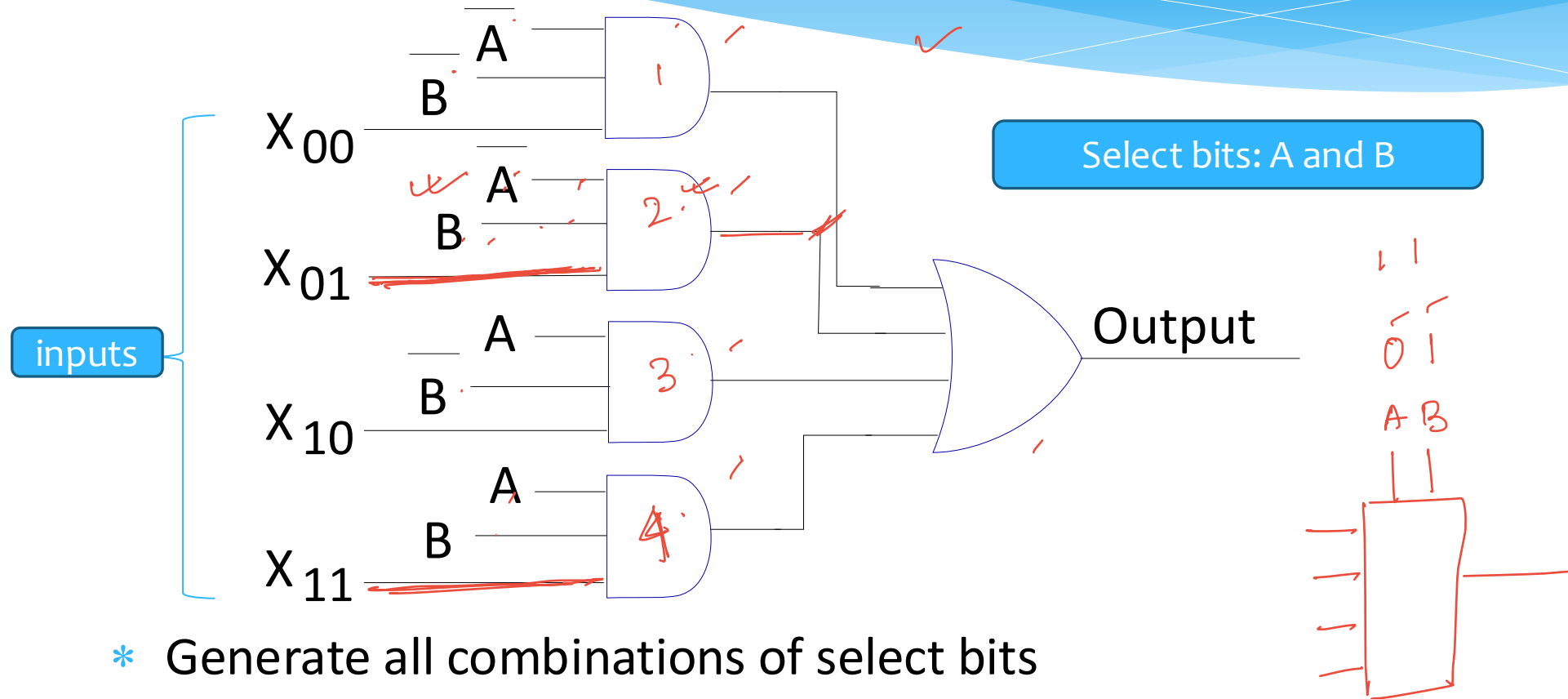
Multiplexer

Given n **inputs**, choose one based on the **select bits**.

Example: Given 8 inputs, 3 select bits, choose 1 among them



Design of a Multiplexer



* Generate all combinations of select bits

* $\bar{A} \cdot \bar{B}, A \cdot \bar{B}, \bar{A} \cdot B, A \cdot B$

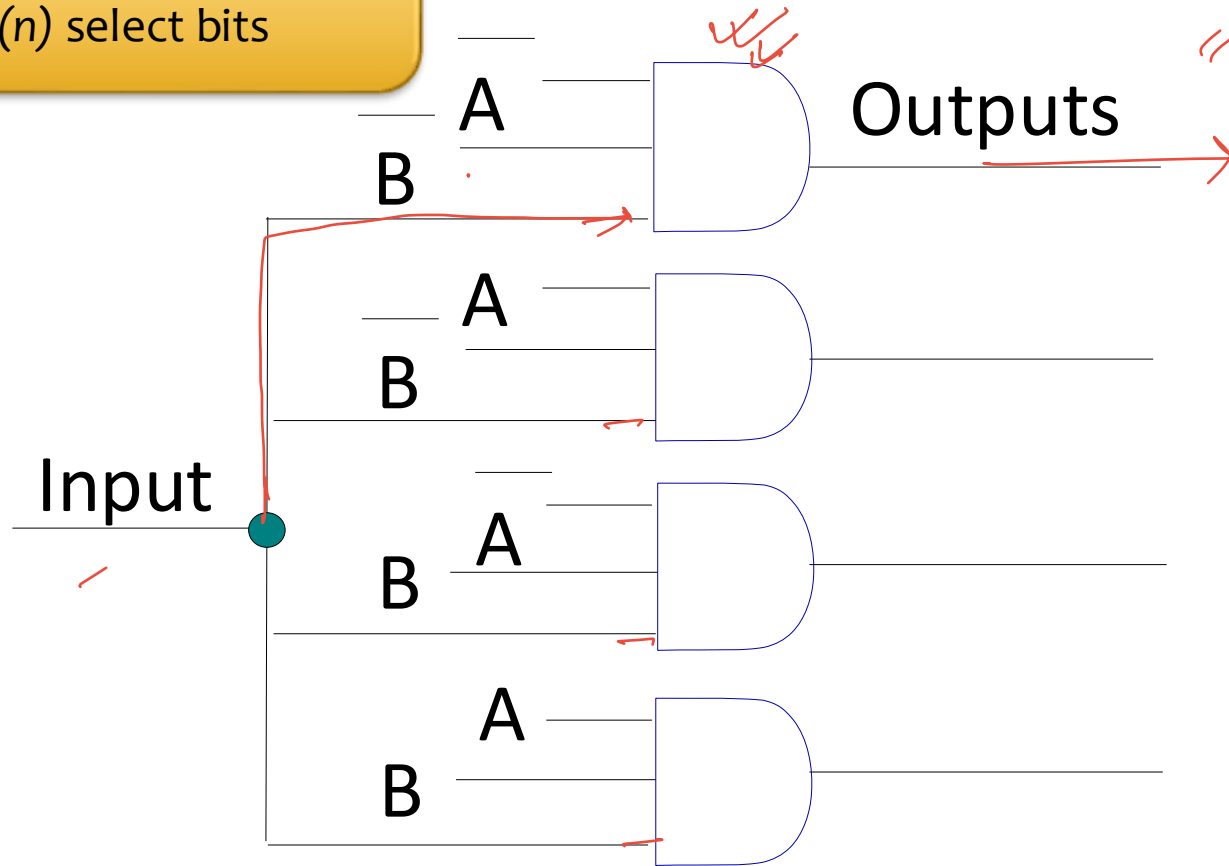
* Only one of the combinations is true.

* For this combination the output of the AND gate is equal to the input (X_{AB}), rest of the outputs of the AND gates are 0

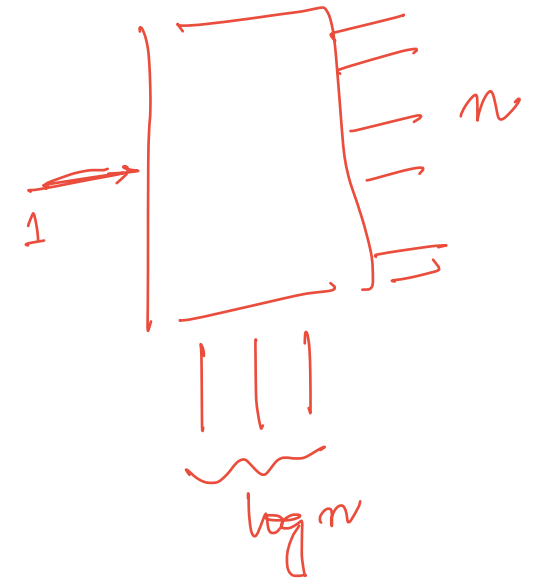
* The output is equal to $(X_{AB} \text{ OR } 0 \text{ OR } 0 \text{ OR } 0) = X_{AB}$

Demultiplexer

Given 1 **input**, make it appear on one of n outputs. The output is decided by $\log(n)$ select bits



A B
0 0

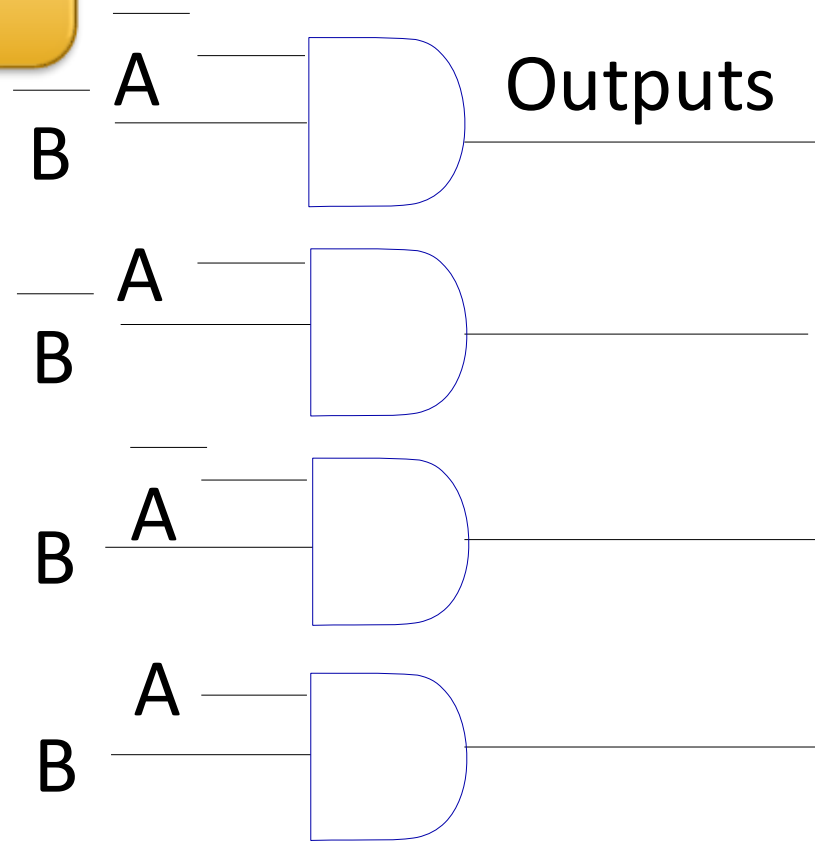


Working of a Demultiplexer

- * Same logic
 - * Generate all combinations of A, B, \bar{A} , and \bar{B}
 - * Only one combination is TRUE
 - * The output of that AND gate is equal to: $X \text{ AND } 1 = X$
Here, X is the input
 - * The outputs of the rest of the AND gates are 0

Decoder

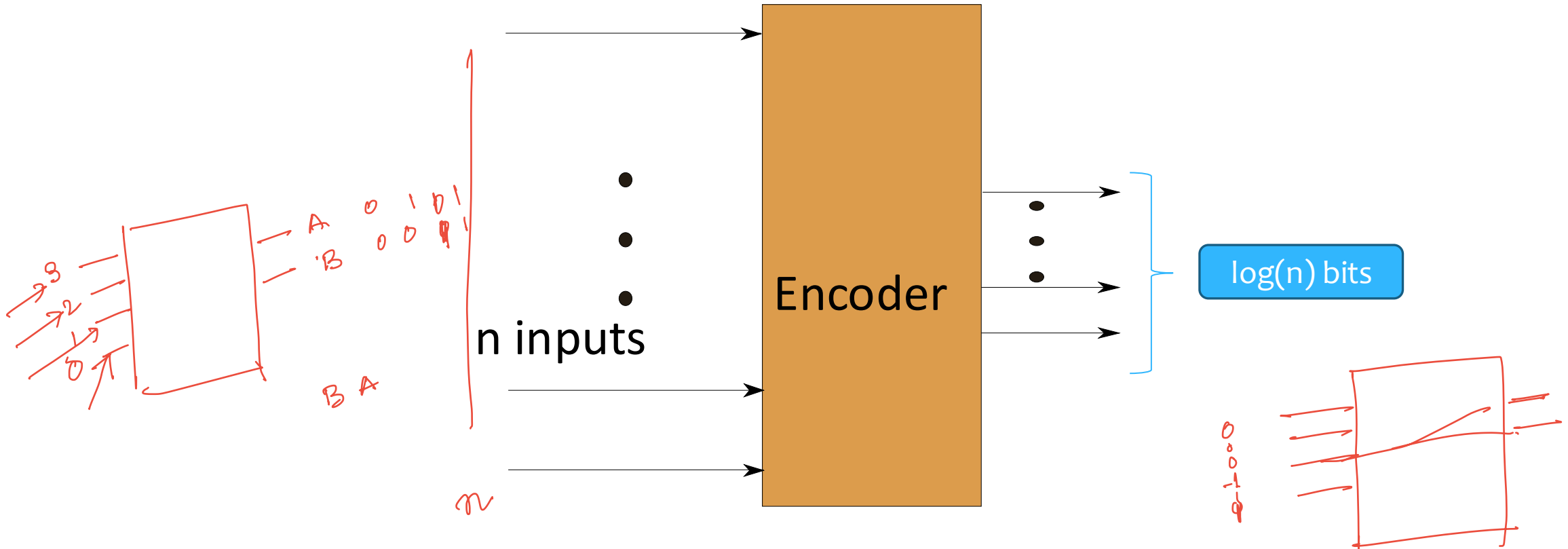
Set one of the n outputs to 1. The output is decided by $\log(n)$ select bits



Operation is similar to a demultiplexer

Encoder

Given n inputs, assume only one of them is 1. Find its id.
For example: if the 6th input out of 8 inputs is 1. The output should be 110



Example of a 3-bit Encoder

Output bits

Input bit	Bit2	Bit1	Bit0
b0	0	0	0
b1	0	0	1
b2	0	1	0
b3	0	1	1
b4	1	0	0
b5	1	0	1
b6	1	1	0
b7	1	1	1

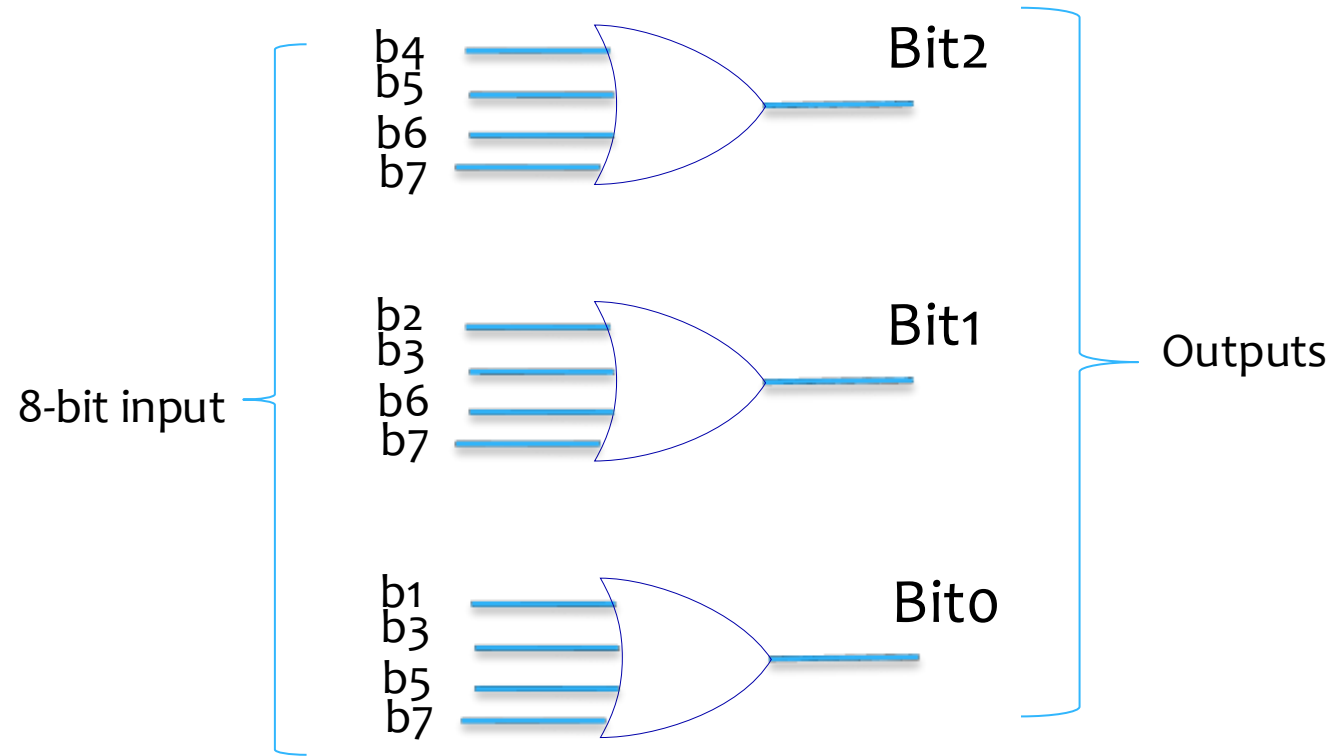
Here, the + symbol stands for OR

$$\text{Bit2} = b4 + b5 + b6 + b7$$

$$\text{Bit1} = b2 + b3 + b6 + b7$$

$$\text{Bit0} = b1 + b3 + b5 + b7$$

Example of a 3-bit Encoder - II



Outline

- * Transistors and Gates
- * Combinational Logic
- * Sequential Logic
- * SRAM/ DRAM Cells



S-R latch

S: Set
R: Reset

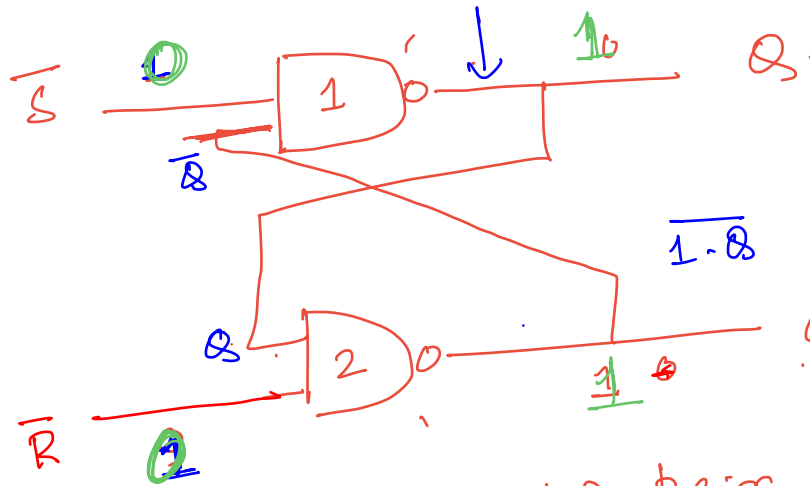
$S=1 \quad R=0$ | $Q=1 \quad \bar{Q}=0$ - Set
 $S=0 \quad R=1$ | $Q=0 \quad \bar{Q}=1$ - Reset
 $S=0 \quad R=0$ | maintain old values.
 $S=1 \quad R=1$ | X not designed state

$$\overline{1 \cdot \bar{Q}} = \bar{1} + \bar{\bar{Q}} = 0 + Q = Q$$

NAND

0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

At least one i/p = 0
 \hookrightarrow o/p = 1



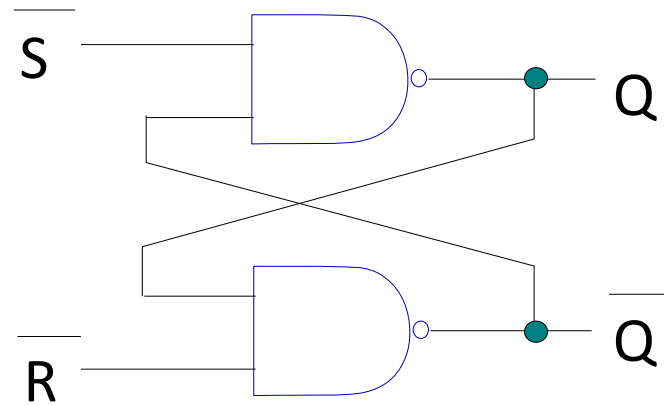
$$\overline{1 \cdot Q} = \bar{1} + \bar{Q} = 0 + \bar{Q} = \bar{Q}$$

cross-coupled pair (latch)

SR Latch



Use a cross-coupled pair of NAND gates to store a stable value.



S → Set
R → Reset

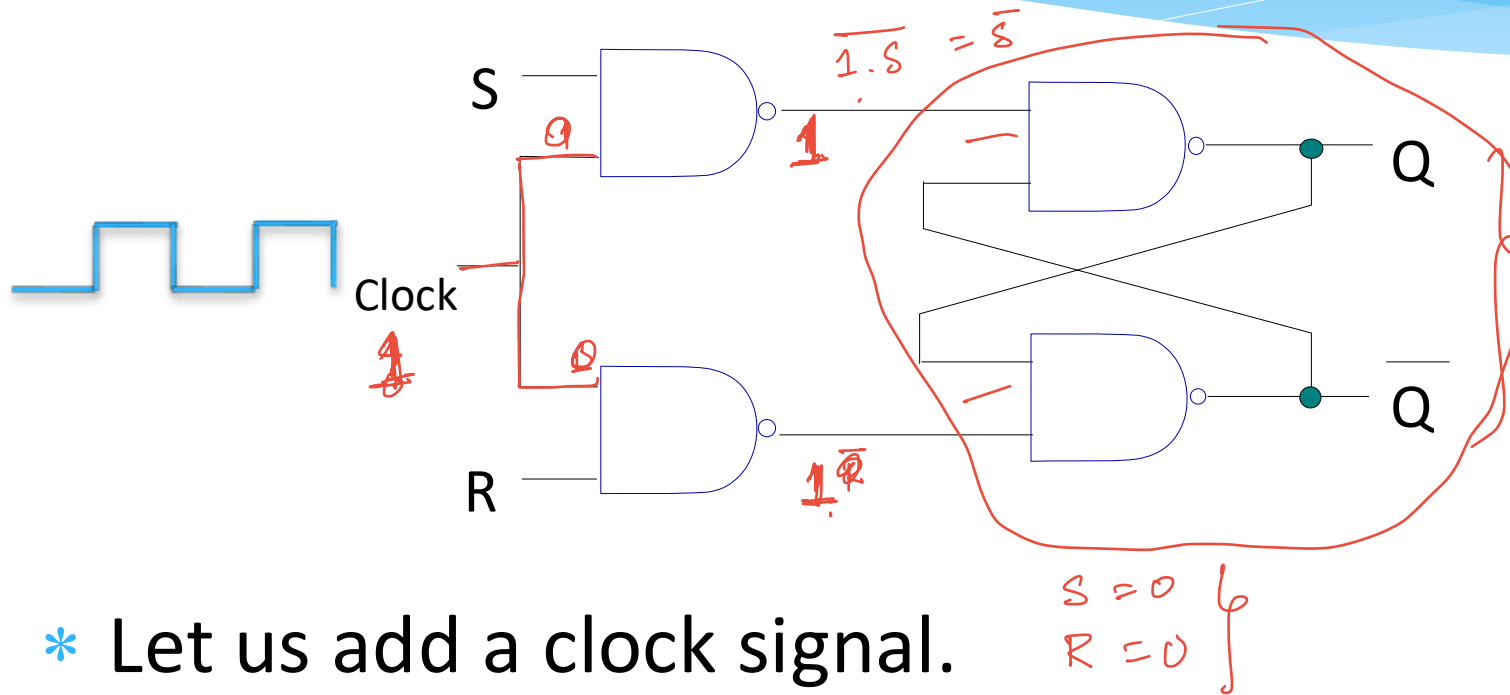
- * $S = 1, R = 0, Q = 1$
- * $S = 0, R = 1, Q = 0$
- * $S = 0, R = 0, \langle \text{maintain old values} \rangle$

Set the values



Understand these points

Clocked SR Latch



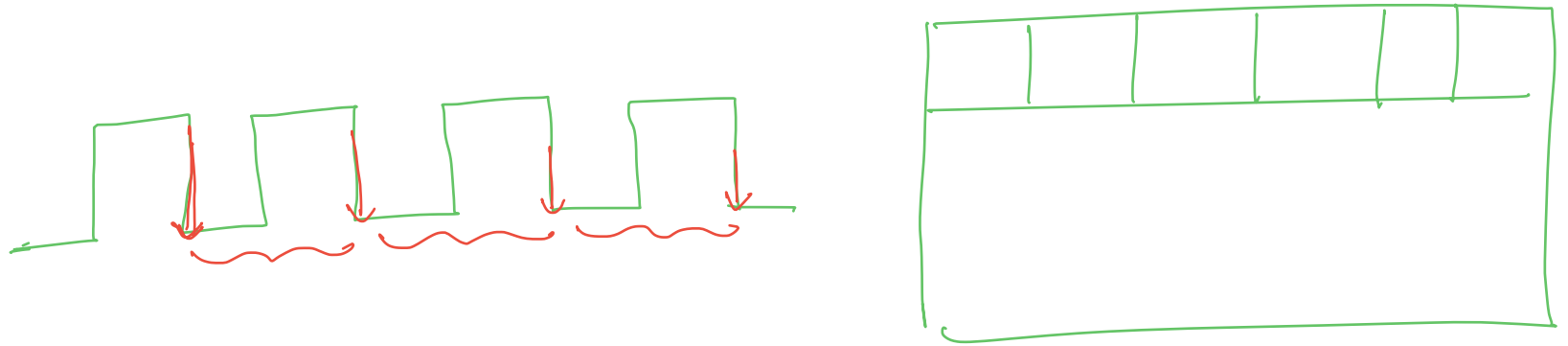
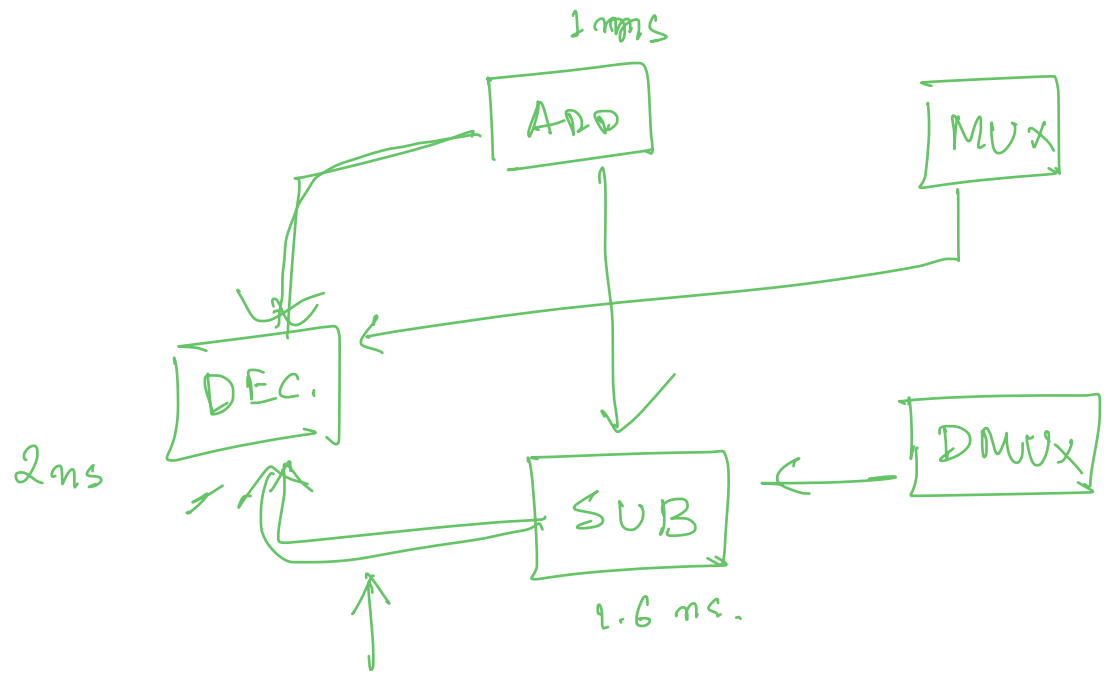
$$freq = \frac{1}{\text{clock period}}$$

200 kHz

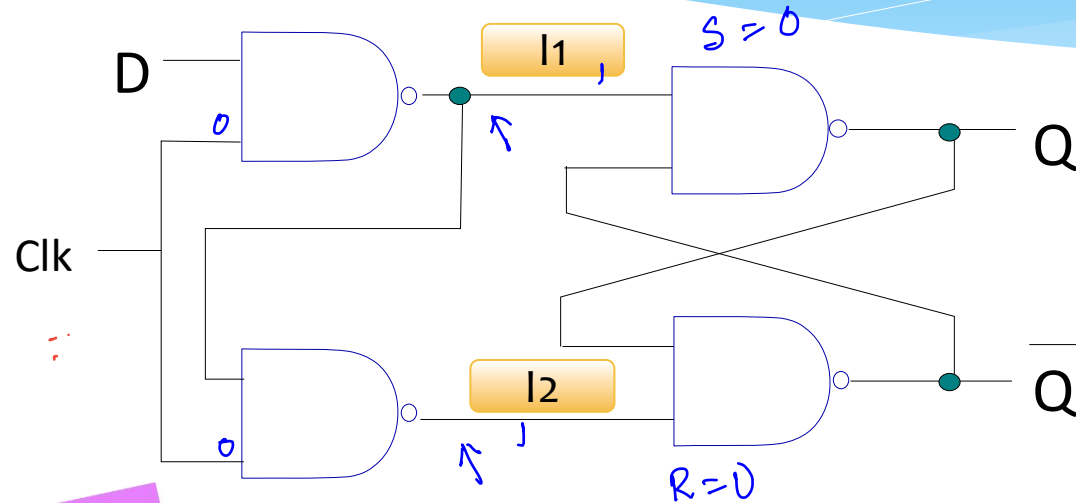
$S=1$ $R=0$

* Let us add a clock signal.

- * When the clock is 1, outputs of the NAND gates are \bar{S} and \bar{R} respectively (same as the classic SR latch)
- * Clock is 0 \rightarrow they are 1 and 1 respectively (maintain old values)



D Flip Flop



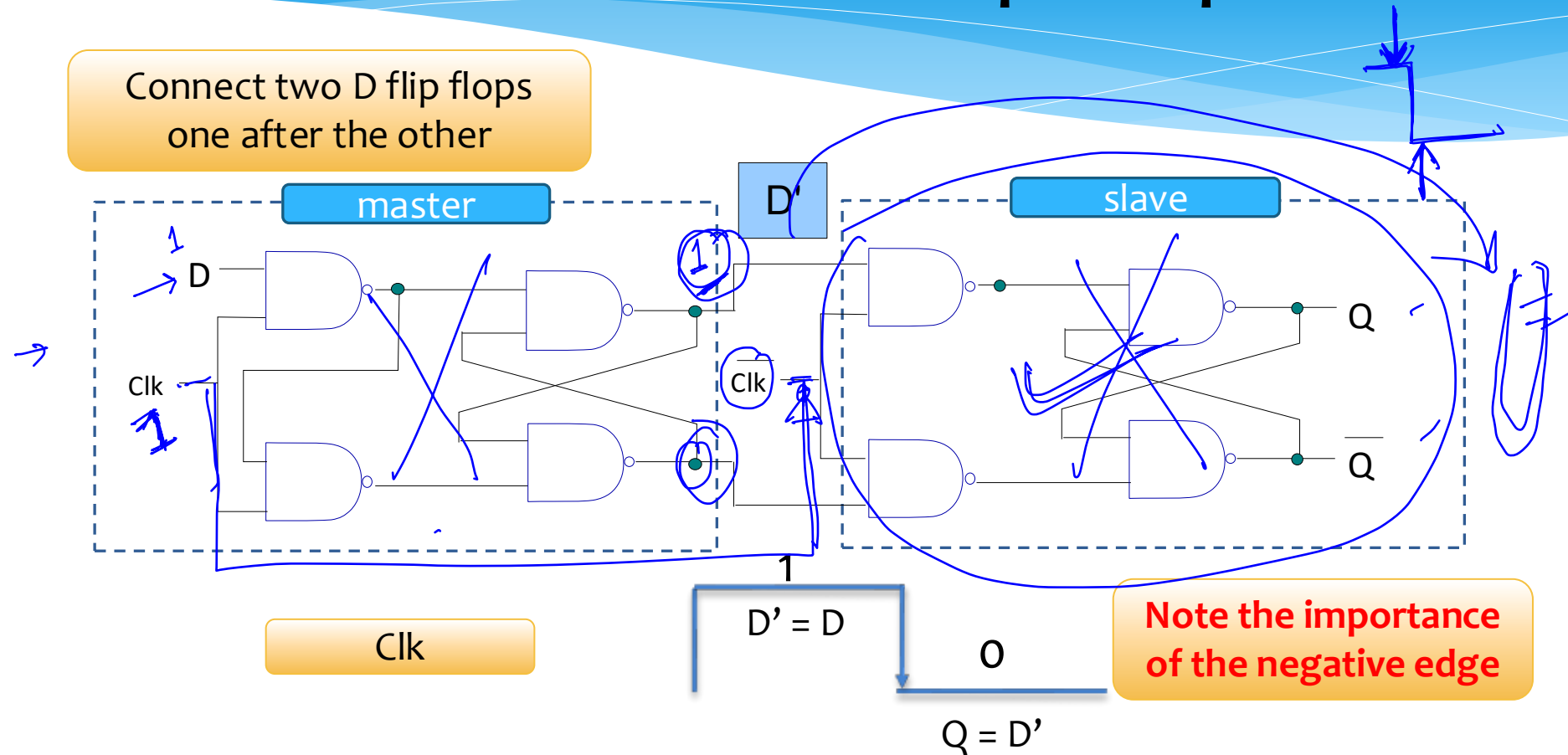
Important

$$D = 1, Q = 1$$

$$D = 0, Q = 0$$

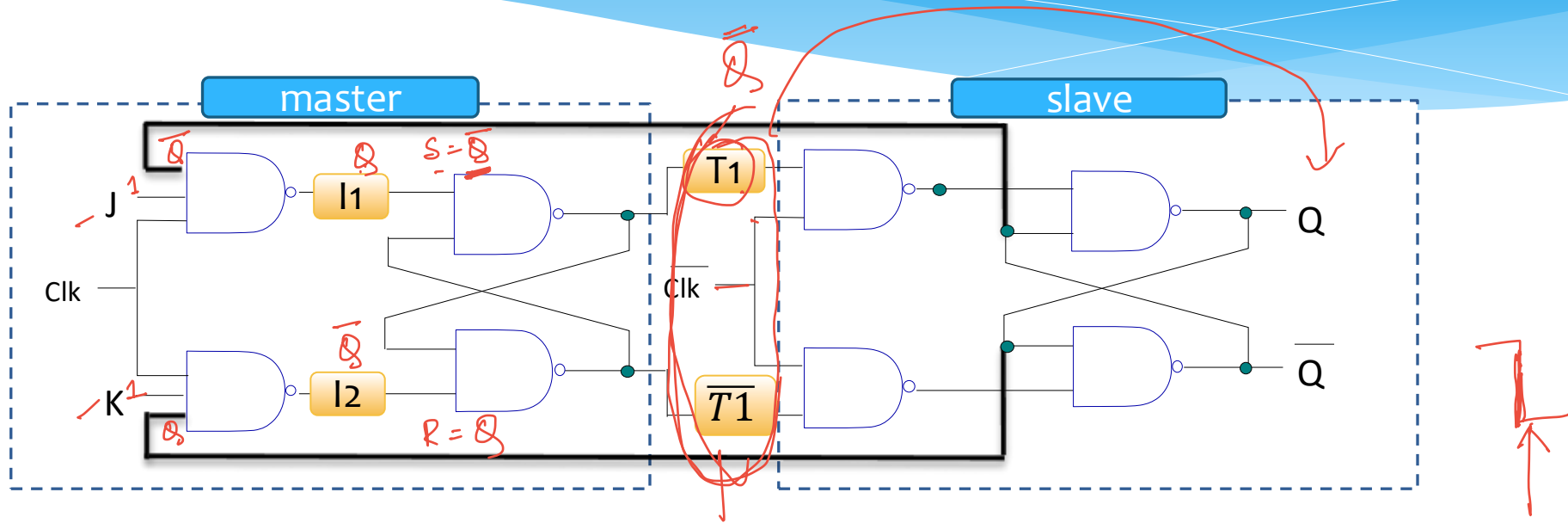
- Let us not have two inputs: S and R
- Let us consider a single input, D, and modify the clocked SR latched appropriately
- If $\text{Clk} = 1$, $I_1 = \bar{D}$, $I_2 = D \rightarrow$ Essentially, this sets Q to D ✓
- If $\text{Clk} = 0$, $I_1 = I_2 = 1$ (maintain old values)

Master Slave D Flip Flop



- When the Clk = 1, the value of D gets transferred to the slave (D')
- Then when the Clk transitions to 0 ($1 \rightarrow 0$)
- The value of D' gets transferred to the output (Q)

Master Slave J-K Flip Flop



master is active

J	K	I1	I2	T1	Clk
0	0	1	1	Q	1
1	0	Q	1	1	1
0	1	1	\bar{Q}	0	1
1	1	Q	\bar{Q}	\bar{Q}	1

Maintain, Set, Reset, or Toggle

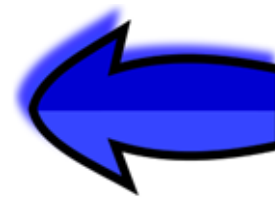
slave is active

J	K	T1	Clk	Q
0	0	old Q	0	old Q
1	0	1	0	1
0	1	0	0	0
1	1	old \bar{Q}	0	old \bar{Q}

Transfer the input to the output

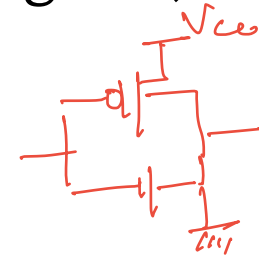
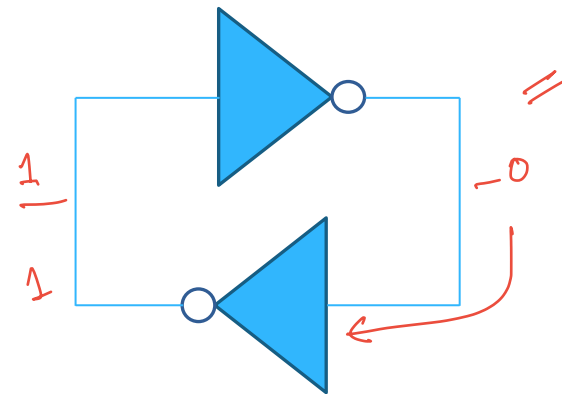
Outline

- * Transistors and Gates
- * Combinational Logic
- * Sequential Logic
- * SRAM/ DRAM Cells



SRAM Cell

- We want to **reduce** the number of transistors required to store a single bit
- Instead of using a **cross-coupled** pair of NAND gates, use a cross coupled pair of **inverters**

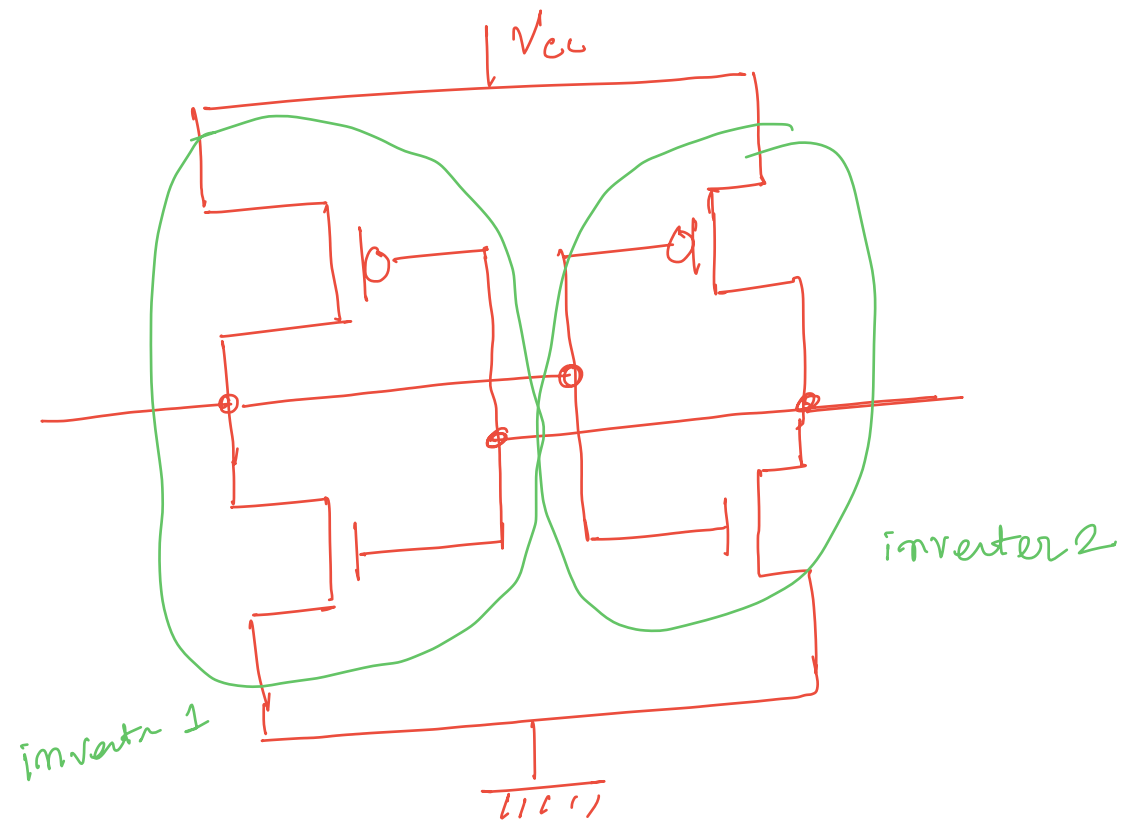


inverter

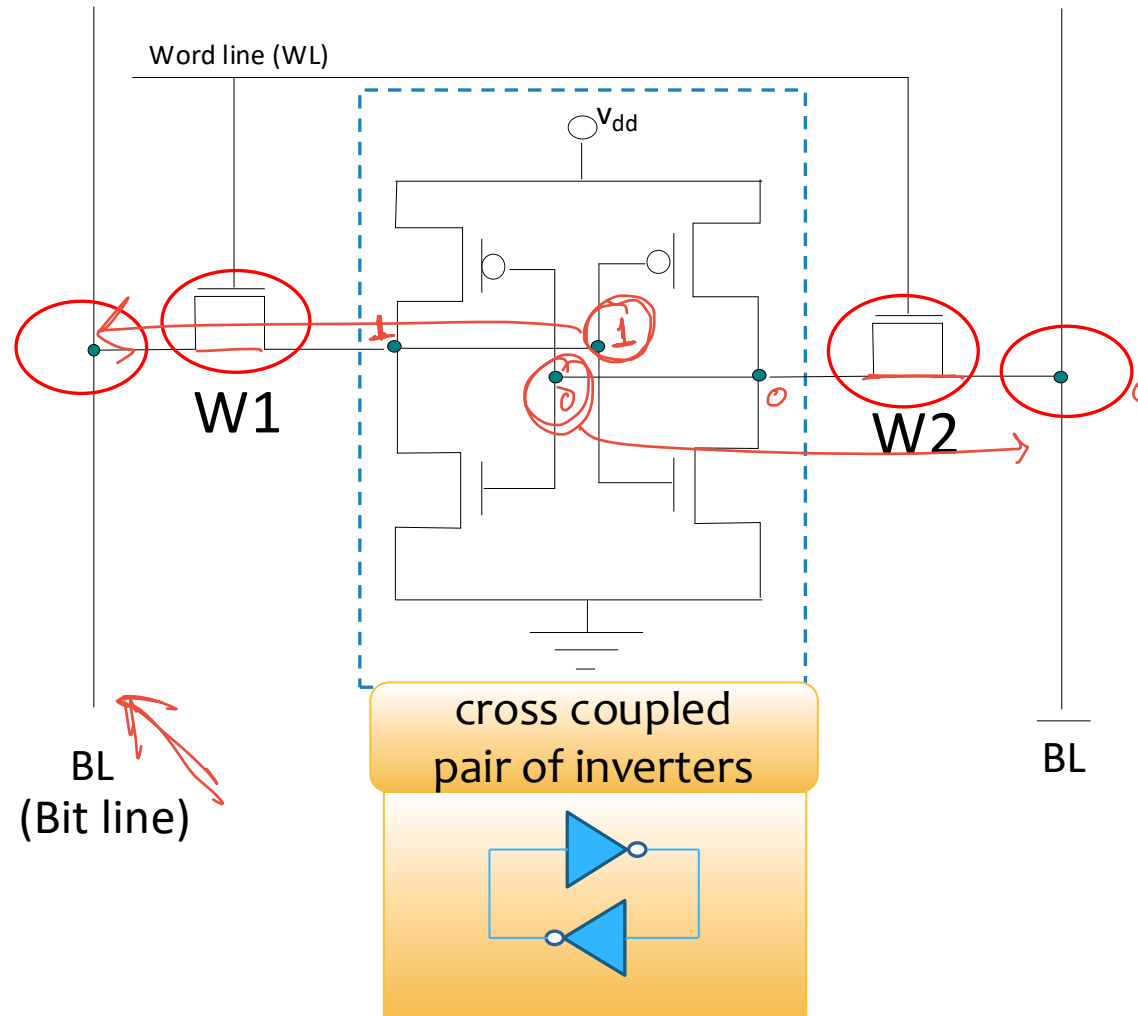
A pair of inverters stores 1 bit



- **Problems:** Need extra circuitry to read and write values

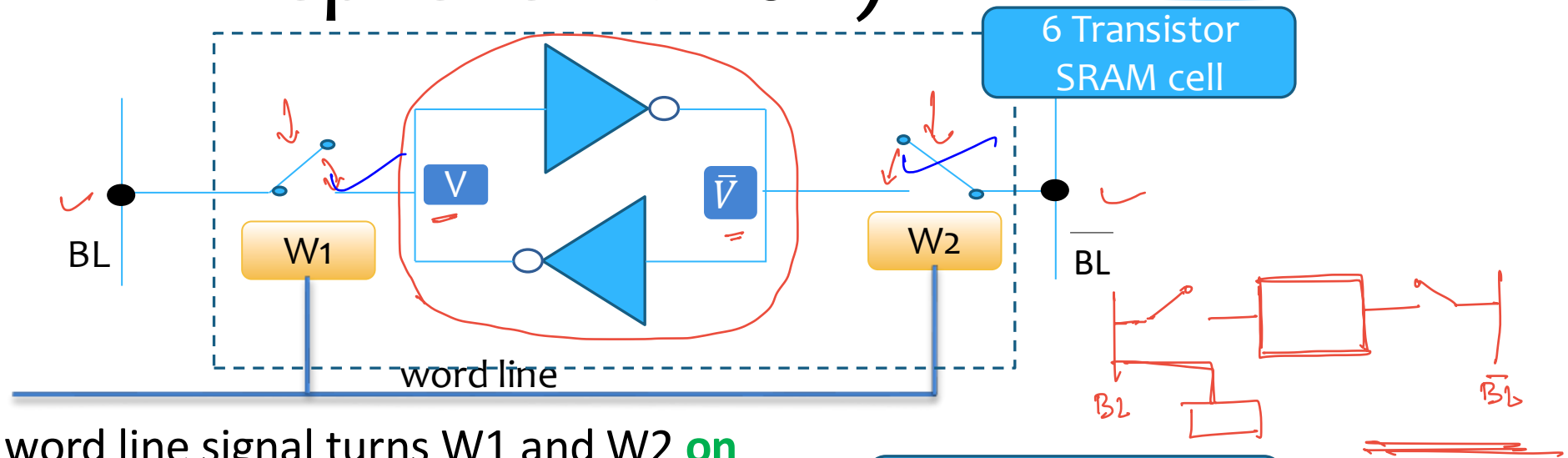


SRAM Cell



Bitlines will float

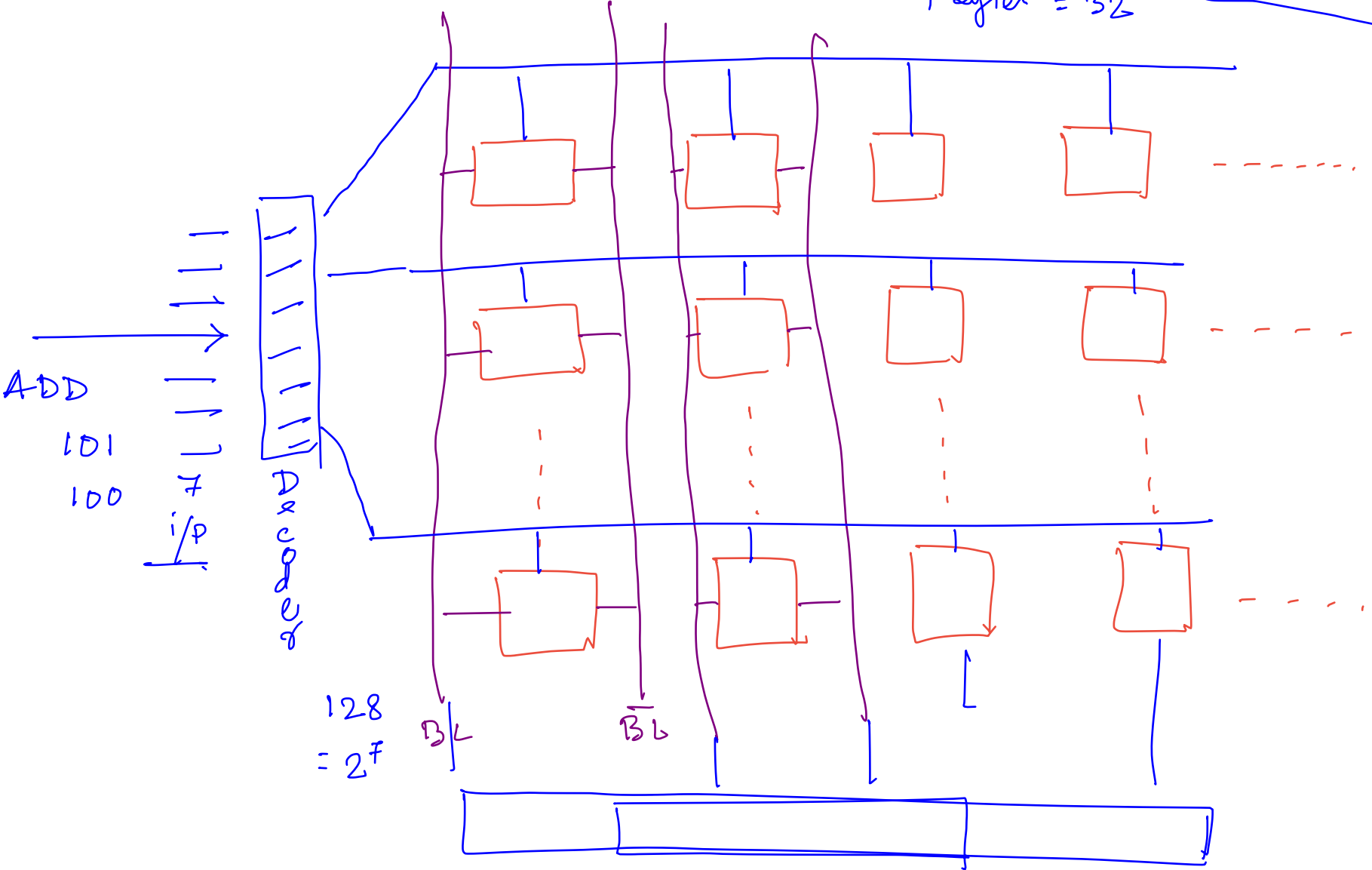
SRAM Cell (Alternative Representation)



- * The word line signal turns W1 and W2 **on**
 - * The inverter pair gets **connected** to the **bit lines**
- * **Read** mode:
 - * The values of BL and \overline{BL} are set to V and \overline{V} respectively. The bit lines are kept floating.
- * **Write** mode:
 - * The values of the terminals, V and \overline{V} , are set to the values of BL and \overline{BL} respectively. The bit lines are charged by strong driver circuits.

$256 \text{ KB} = 2^8 \times 2^{10} = 2^{18} \text{ B}$
 $= 2^{21} \text{ B}$

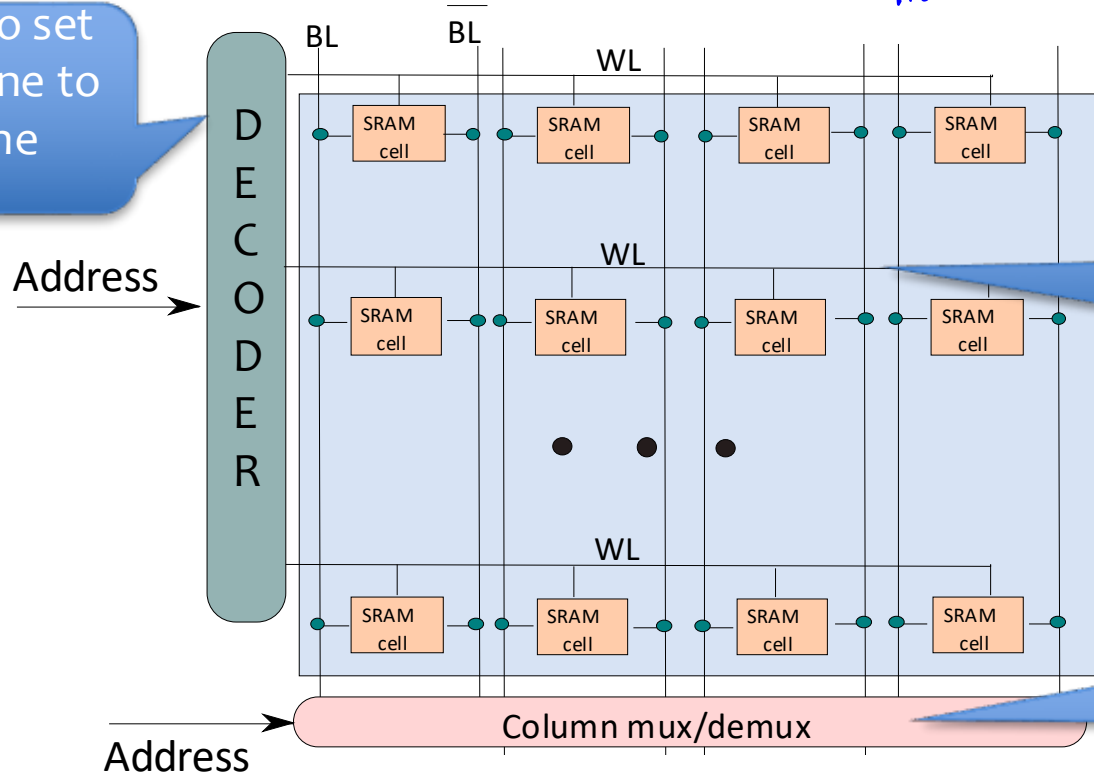
4 bytes = 32



Array of SRAM Cells

To save space, we arrange SRAM cells into rows and columns

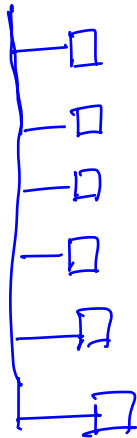
Use a decoder to set only one word line to 1 (based on the address)



Setting a word line enables all the cells in a row

! Only one row can be enabled at a time

Choose a subset of columns to read or write



driving them to logical 0 or 1 will take time.

Operating an Array of SRAM Cells

- * **Write** mode:

- * Use a **driver circuit** to set the values of BL and \overline{BL}
- * Simultaneously **enable** the word line
- * The values in BL and \overline{BL} will get **transferred** to the SRAM cell (thus a write)

- * **Read**:

- * **Disconnect** the bit lines from supply and ground using transistors (floating)
- * **Enable** the word line
- * The **values** of (BL and \overline{BL}) will get set to logical 1 and 0 respectively if the value stored in the SRAM cell is 1, or vice-versa

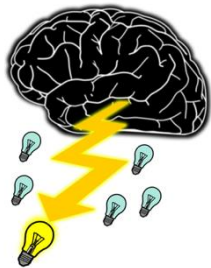
Pre-charging

* Issues with the simple design:

- * The bit lines are very long, and are connected to a lot of SRAM cells
- * They thus have a lot of resistance and capacitance
- * Driving them to a logical 0 or 1 (typically 1V) will take a long time
- * This will make the SRAM **slow**
- * Can we do something **better**

* Observe:

- * Let us pre-charge both the bitlines to 0.5 V (assuming logical 1 is 1V) using strong pre-charge driver circuits
- * Let us then **enable** the word line
- * One of the bit lines will move **towards** 0V and the other **towards** 1V
- * **IDEA:** Monitor the difference in voltages between the bit lines



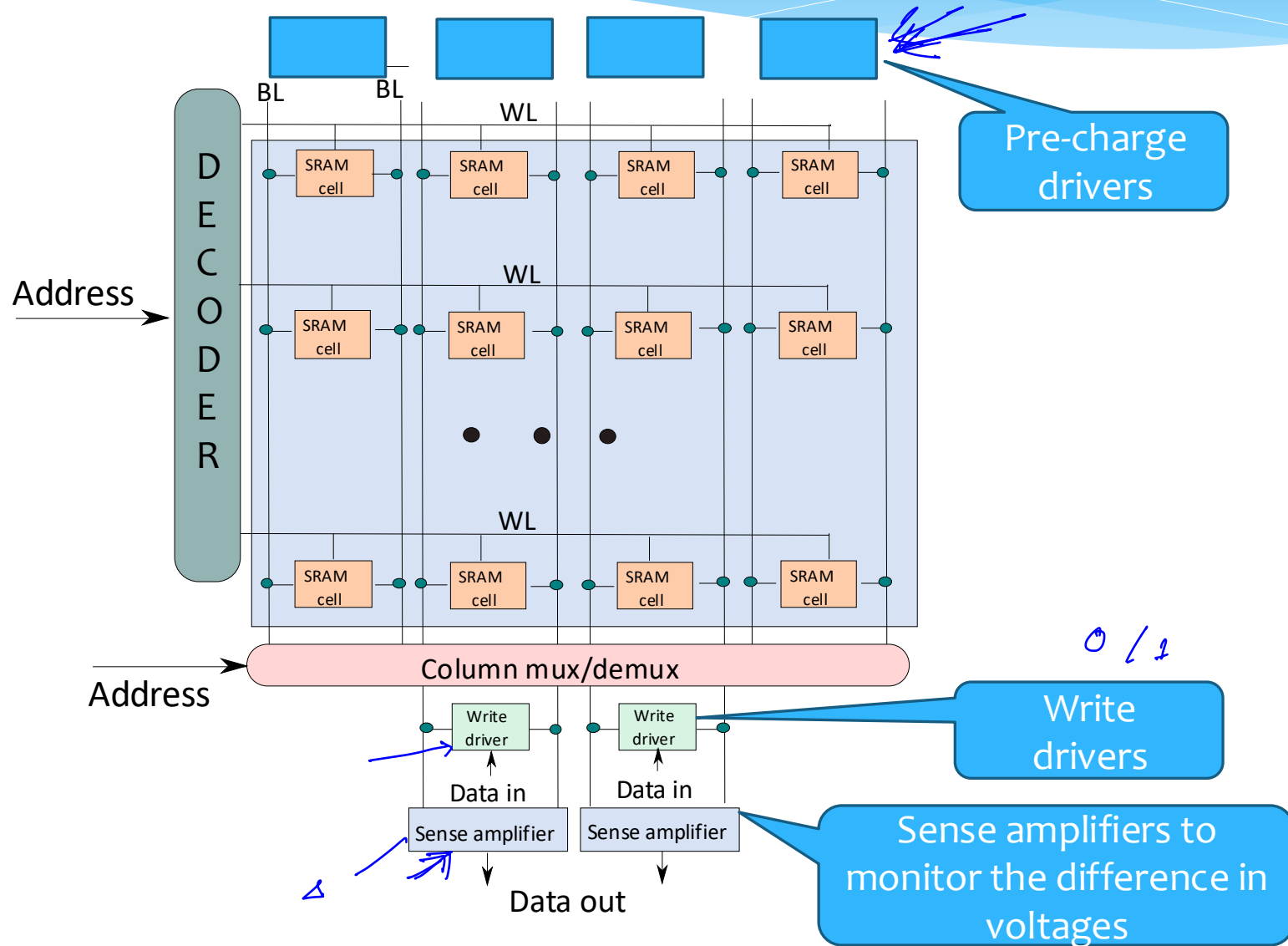
Pre-charging - II

- * Should we **wait** for one of the **bit lines** to reach 0V and the **other** to reach 1V
 - * Answer: **NO**
 - * If we know the outcome: **why wait**
- * **Monitor** the difference: $\Delta = \text{Voltage}(BL) - \text{Voltage}(BL)$
 - * There can be some amount of electrical noise that might cause a little bit of fluctuation in the voltages of the bit line. Define a noise threshold, T .
 - * The moment: $|\Delta| > T$, declare the **result**
 - * If Δ is +ve, infer a logical 1
 - * Else, infer a logical 0



A very fast method of reading

Array of SRAM Cells



Array of DRAM Cells

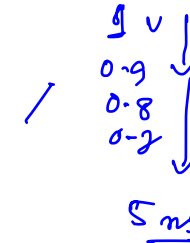
* Features

- * There is a single bit line ✓
- * ✓ For writing → Enable the word line and charge the bit line with a driver circuit
- * For reading → pre charge the bit line to 0.5 V, enable the word line, and monitor the difference of the voltage w.r.t 0.5 V
 - * If the difference in voltages exceeds a threshold
 - * Infer a logical 0 or 1 depending on the sign of the difference

DRAM Refresh

* Problems

- * After every read, we lose some charge from the capacitor
- * Periodically, charge from the capacitor leaks out
- * Hence, it is necessary to:
 - * Periodically read each and every DRAM cell
 - * And write the same data back
- * **Example:** The charge across the capacitor reduces from 1 V to 0.7 V over time.
 - * Let's say, this is enough to infer a logical 1.
 - * **Read** the value (logical 1 in this case)
 - * And **write** the same value again. We thus restore the charge on the capacitor to 1 V.

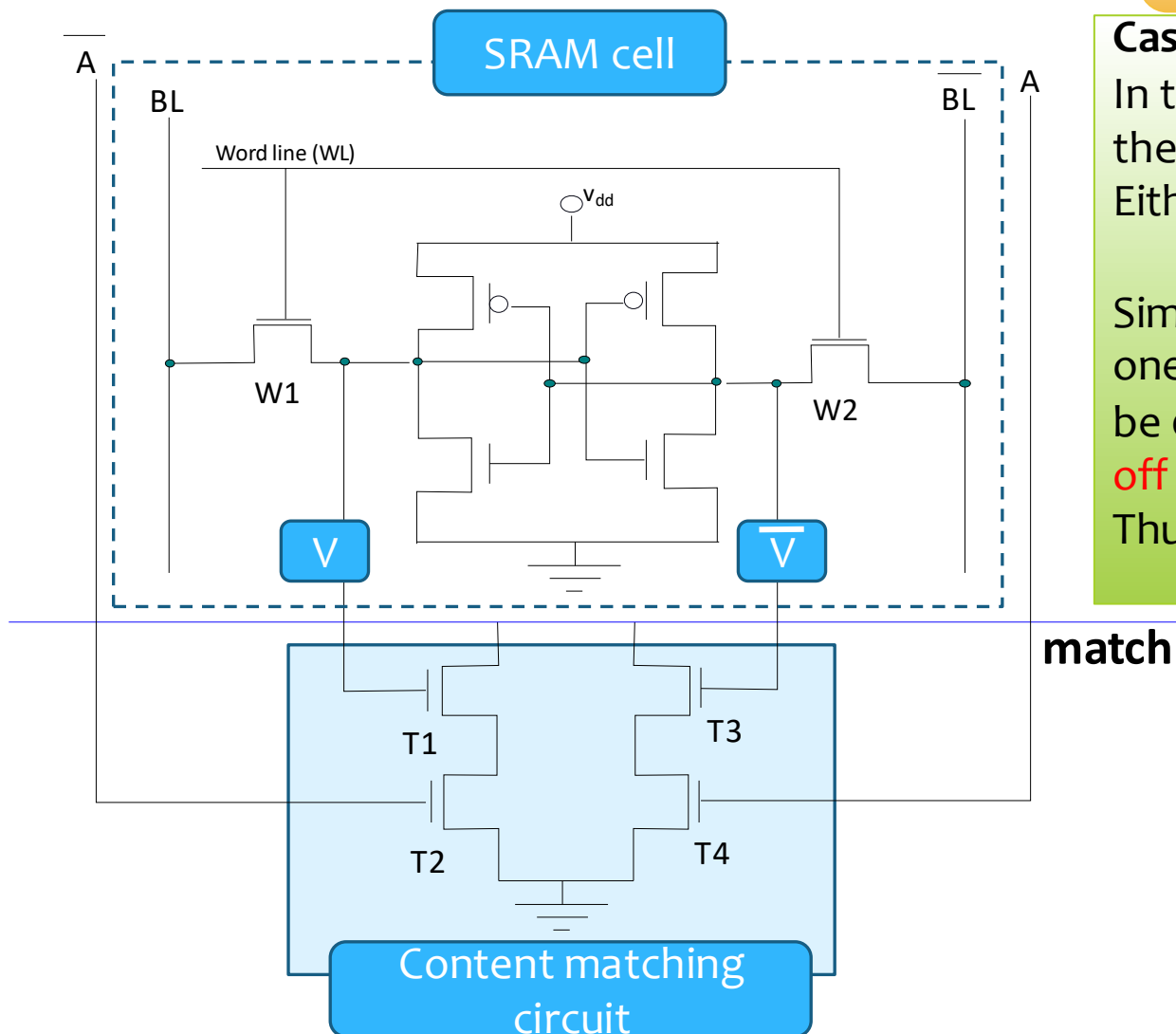


This is known as DRAM refresh

CAM Cell

Instead of enabling cells by their address, can we find them by their contents?

Content Addressable Memory \rightarrow CAM



Case: $A == V$

In the pair (A, \overline{V}) , one of the values has to be 0 \rightarrow Either T3 or T4 is **off**

Similarly, in the pair (\overline{A}, V) , one of the values has to be 0 \rightarrow Either T1 or T2 is **off**

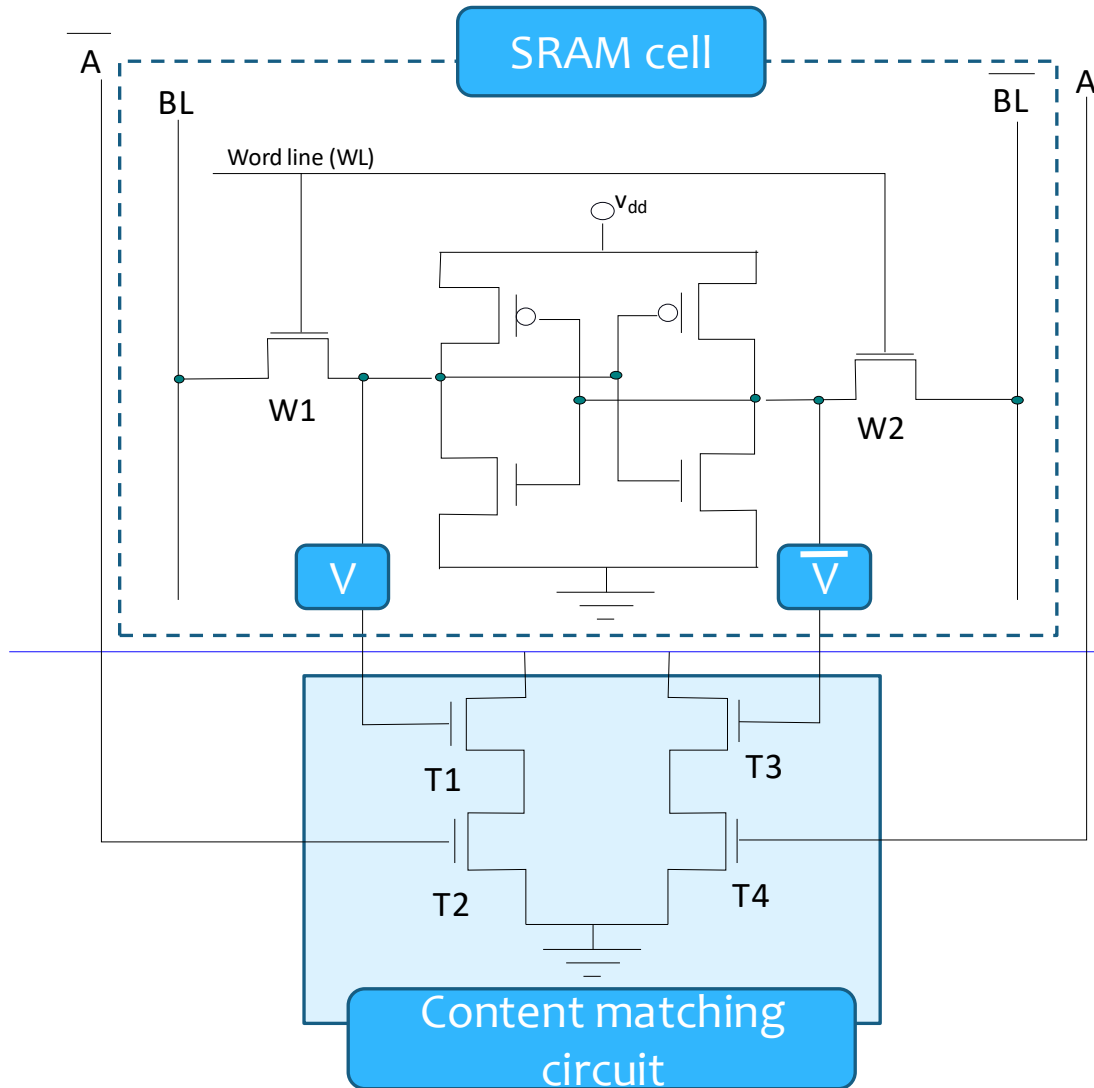
Thus, match is **floating**

match

CAM Cell

Instead of enabling cells by their address, can we find them by their contents?

Content Addressable Memory \rightarrow CAM

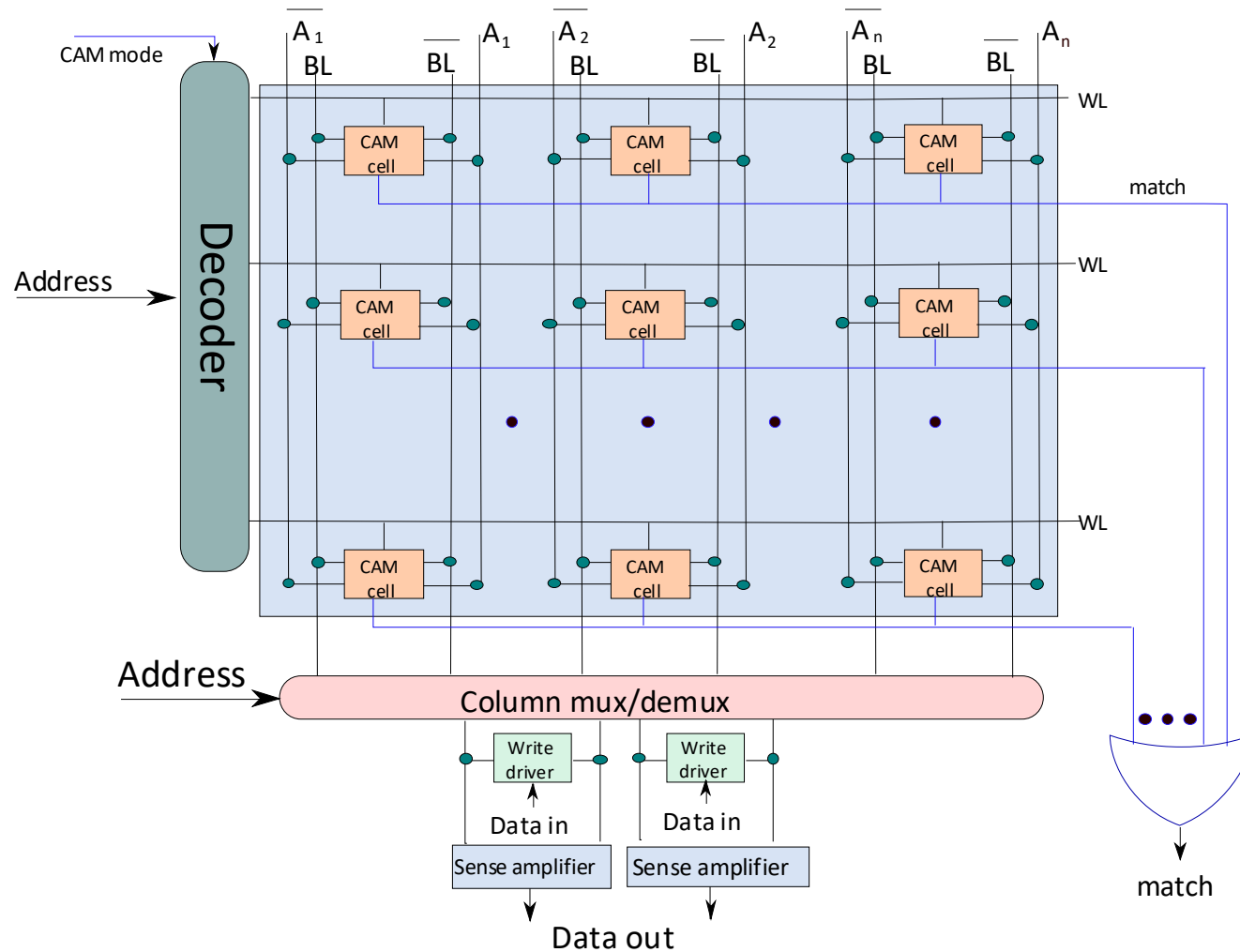


Case: $A \neq V$
Either (A, \bar{V}) , or (\bar{A}, V) is equal to $(1,1)$.

This connects the match line to the ground (sets its voltage to 0 volts)

➔ If there is a **mismatch**, the match line is driven to 0 V

Array of CAM Cells

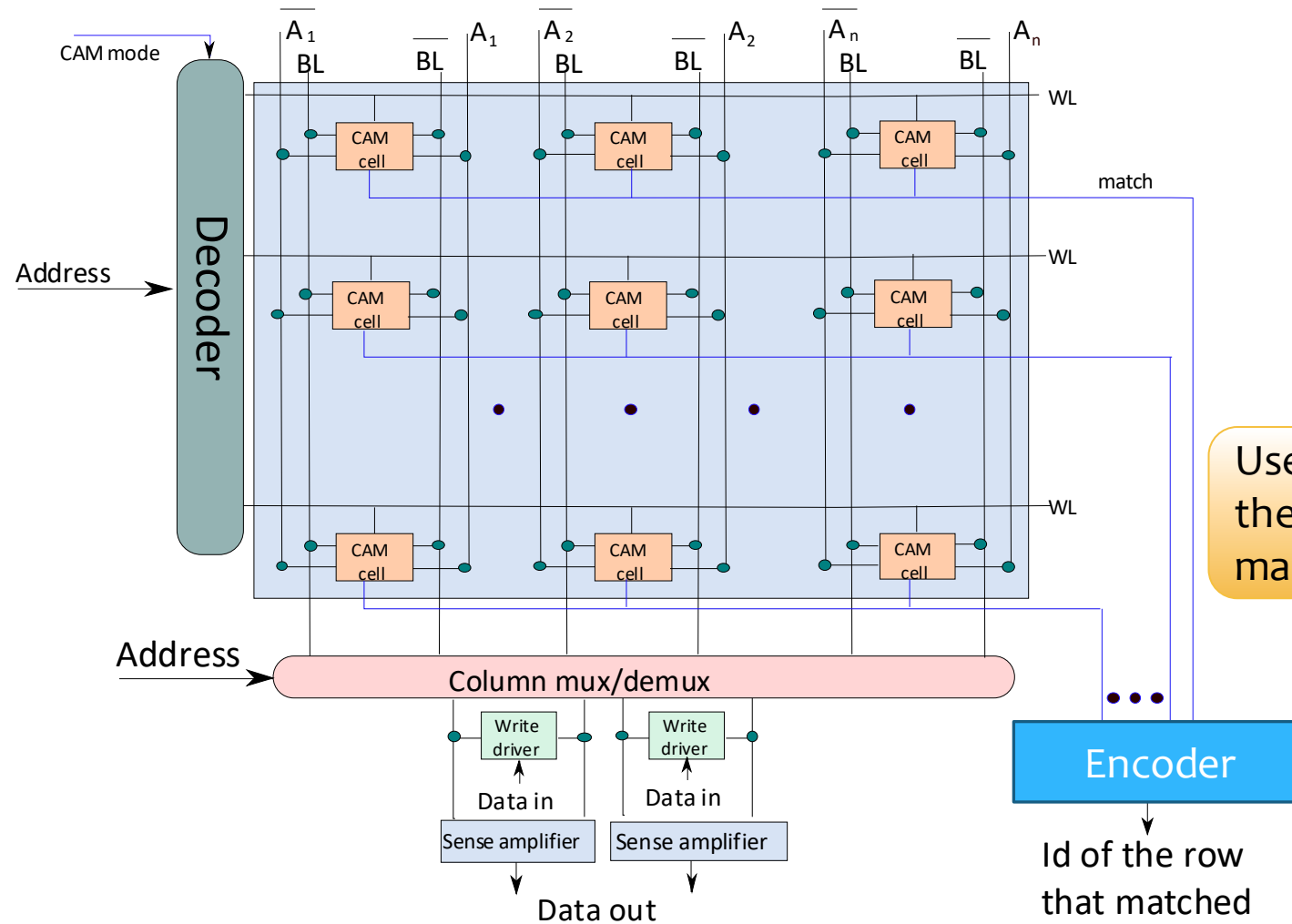


Pre-charge all the match lines to 1

If there is a content match, one of the match lines will be 1

The OR gate will thus compute 1 (else 0 if none of the rows match)

Array of CAM Cells - II



Use an encoder to find the id of the row that matched.



THE END

