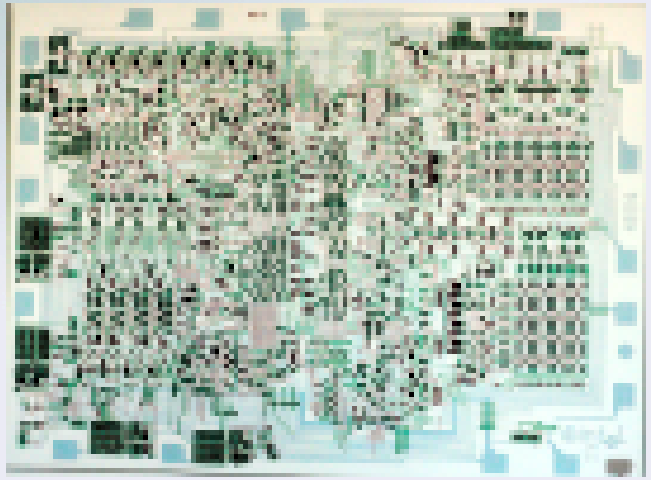


Lecture 2b: Transistors, Gates, and Boolean Logic

CSCI11: **Computer Architecture and Organization**

Chapter 3.1-3.3.3 in Patt and Patel: Introduction to Computing Systems...

All of AspenCore Boolean Algebra

Everything is a zero or a one

- 0 or 1
- pristine or hole, in a punch card
- gnd or VCC
- 0 or 5V
- ground or power
- open or closed

How do we do this electronically?

- Transistors
- Lots of them...
- 100's of Billions in latest processors

Figure 3. Two Switches in Parallel

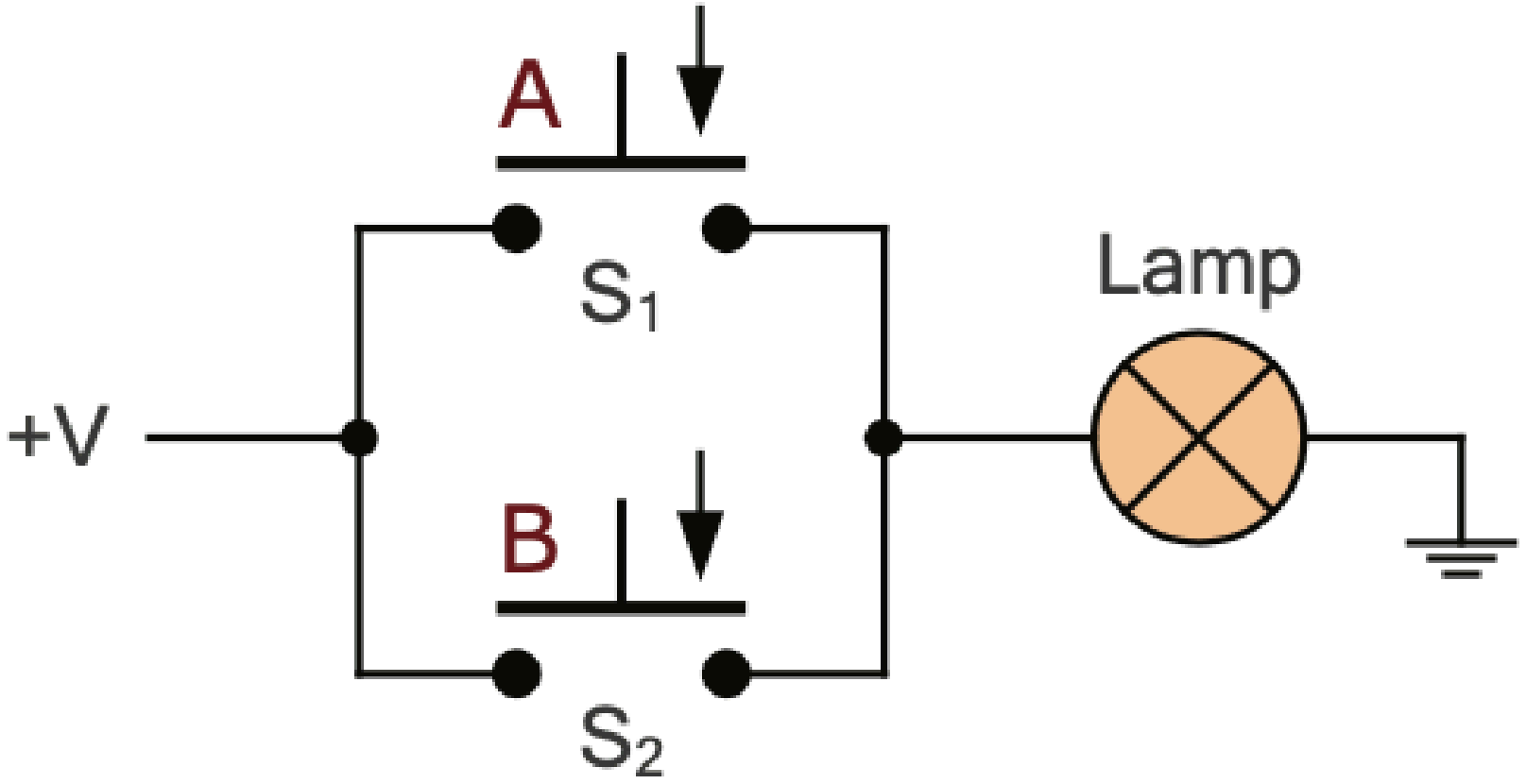
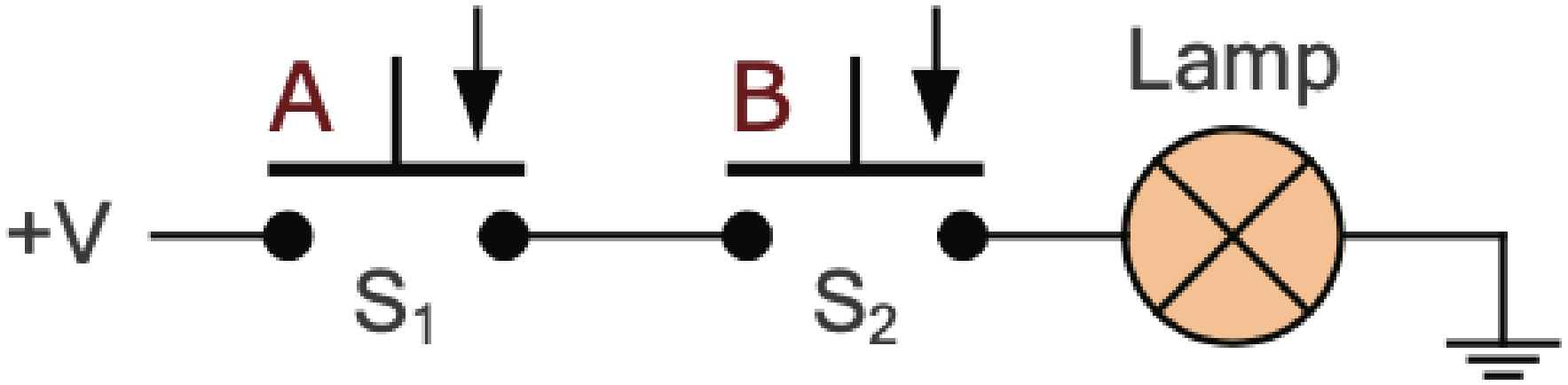


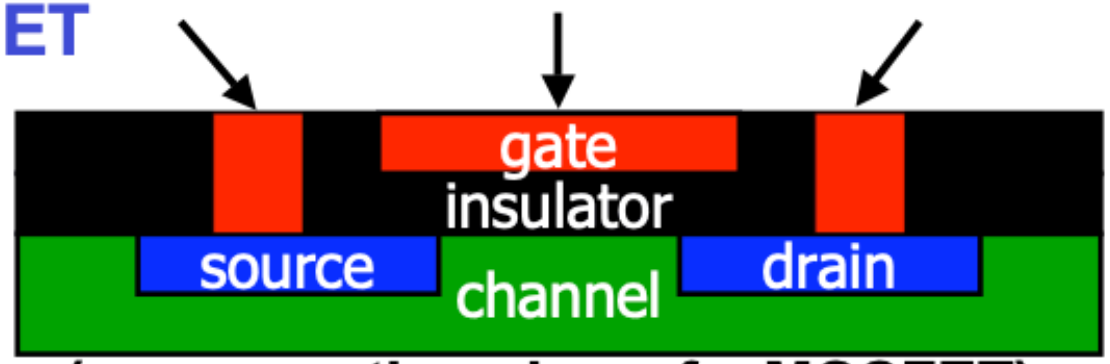
Figure 2. Two Switches in Series



What if we had switches which were:

- really fast!
- really small!
- cheap!
- low power consumption!
- easy to manufacture and test (*relatively*)

MOS + FET



(cross-section view of a MOSFET)

MOS: three materials needed to make a transistor

- **Metal (Al, W, Cu): conductor**
- **Oxide (SiO_2): insulator**
- **Semiconductor (doped Si): conducts under certain conditions**

FET: field effect (the mechanism) transistor

- **Voltage on gate: current flows source to drain (transistor on)**
- **No voltage on gate: no current (transistor off)**

Recall, two types of MOSFET: n and p

Why electrons flow, or why current flows:

- applied gate voltage creates an **electric field** that attracts charge carriers (electrons or holes), or what we know as *current*
- forming a conductive path (channel) between *source* and *drain*
- The type of charge carriers depends on whether it's an **n-type** or **p-type** MOSFET
- **The gate voltage effectively acts like an electronic switch, controlling current flow through the channel**

MOSFETs are ideal for digital electronics as they can be:

- very efficient switches
- consume almost no power in their off state
- provide good conductivity when on
- can be made extremely small* (*measured in 10's nm*)
- can be produced and tested in high volumes

***25,400,000 nanometers (nm) in 1"**

***645,160,000,000,000 square nanometers (nm^2) in 1 square inch**

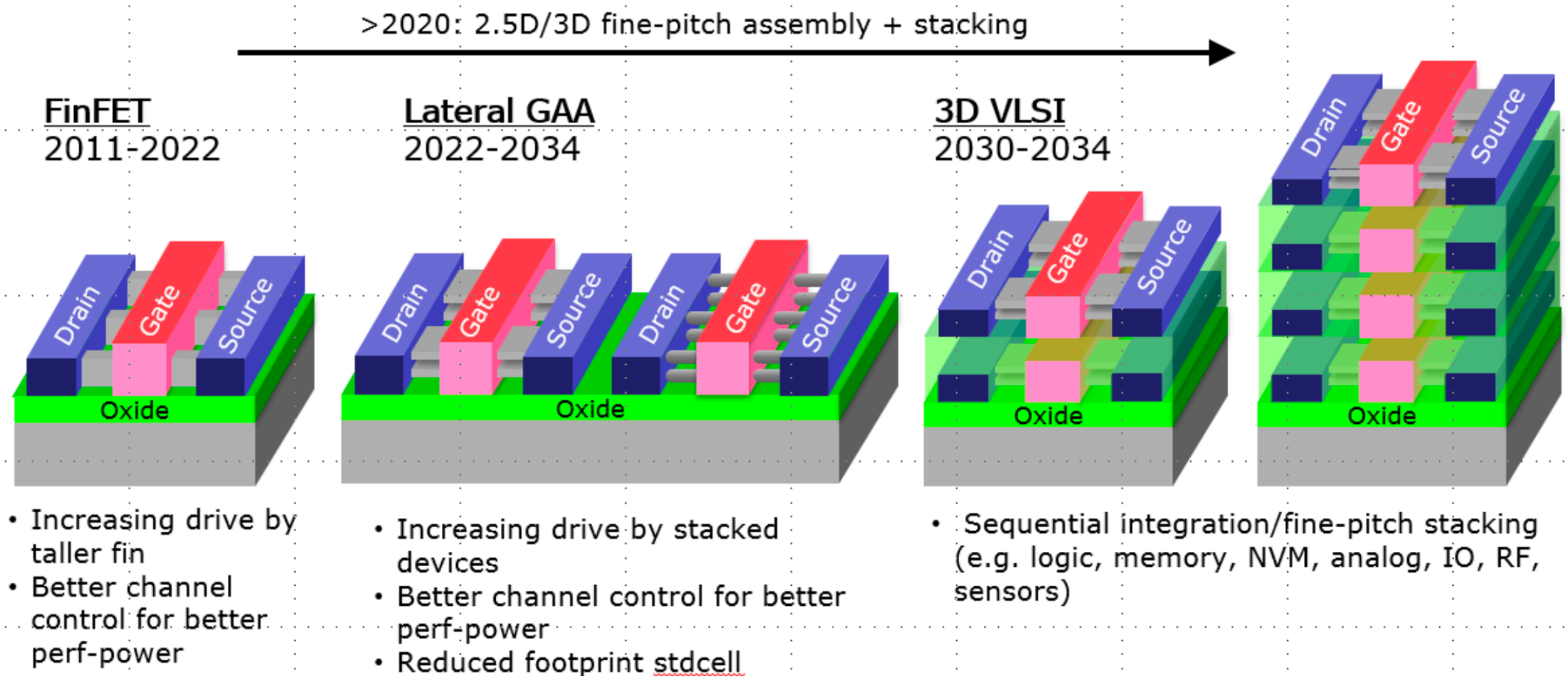
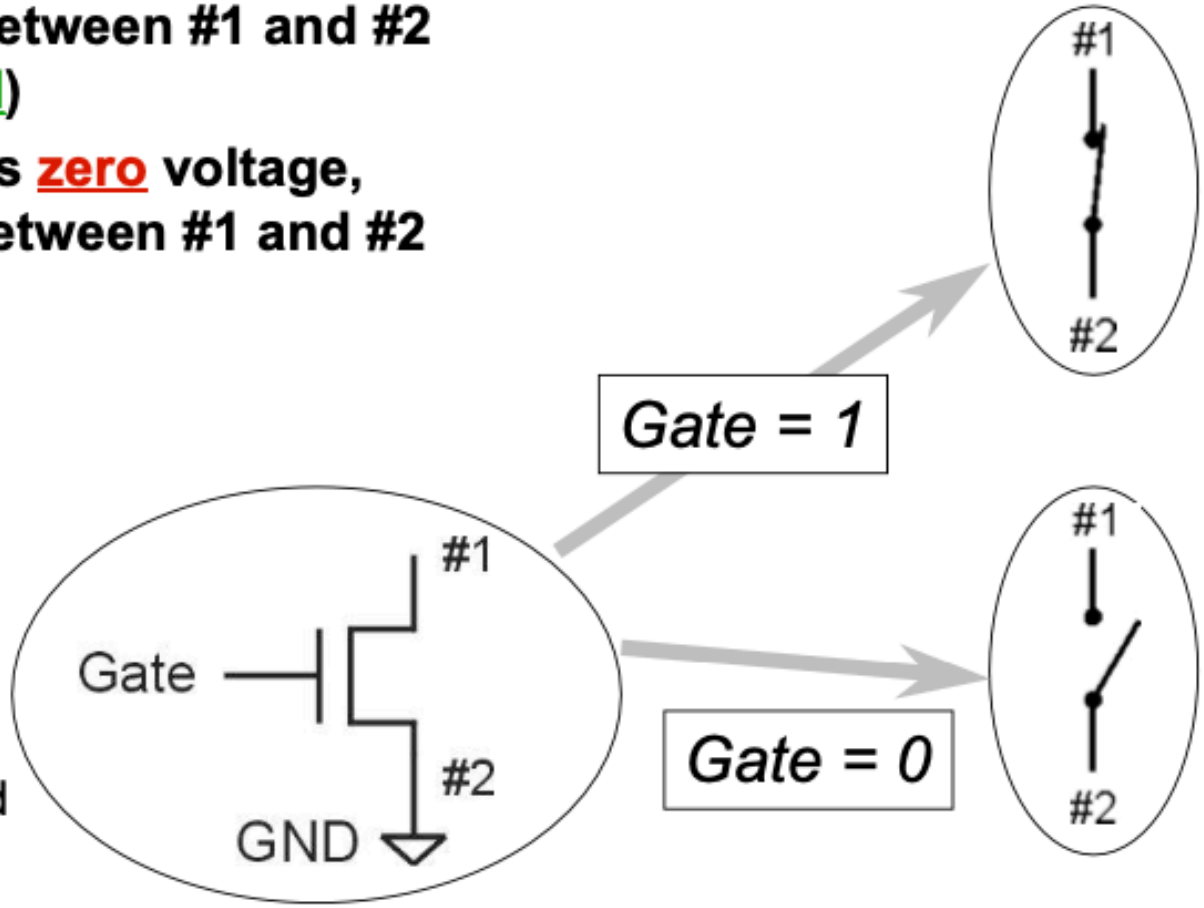


Figure MM-5 Evolution of device architectures in the IRDS More Moore roadmap

N-type MOS Transistor

- When Gate has positive voltage, short circuit between #1 and #2 (switch closed)
- When Gate has zero voltage, open circuit between #1 and #2 (switch open)

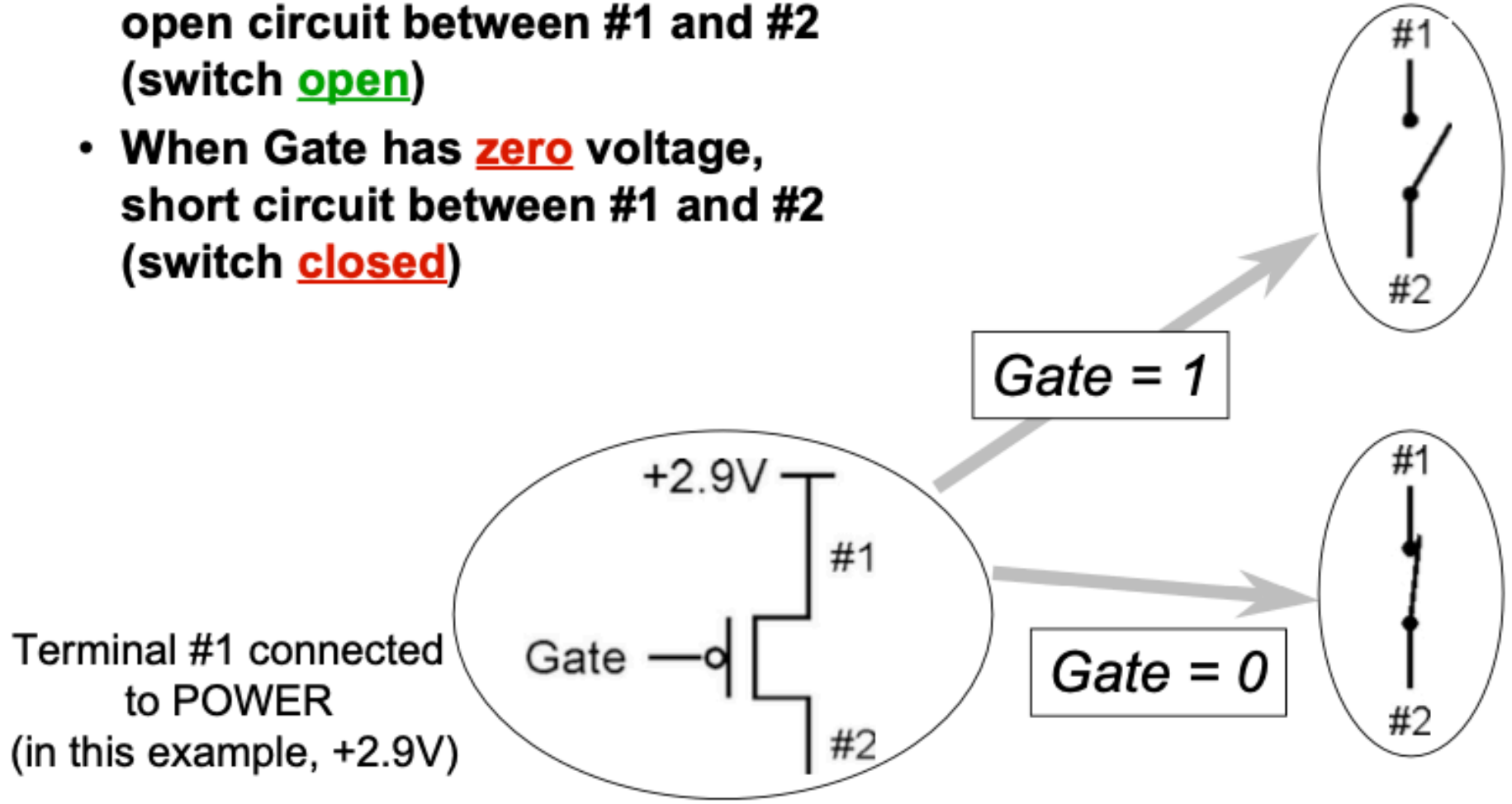


Terminal #2 connected to GROUND (0V).

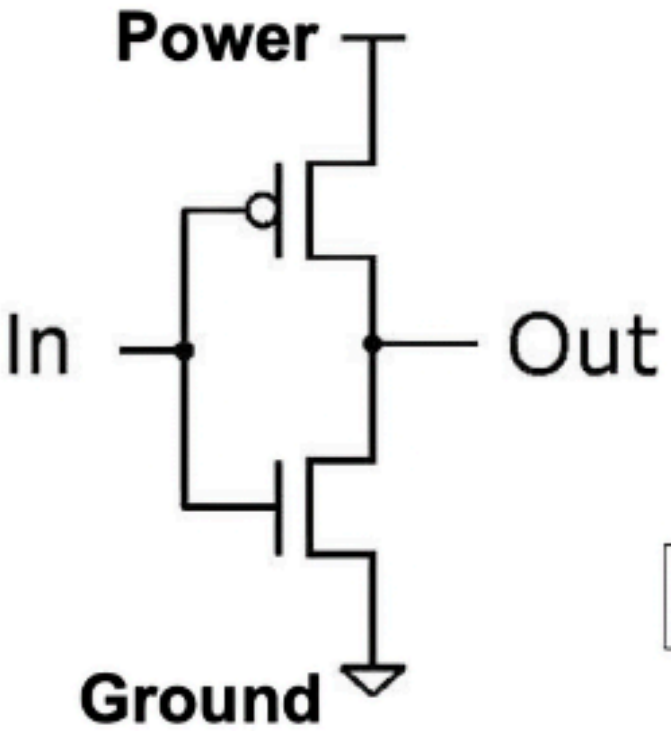
P-type MOS Transistor

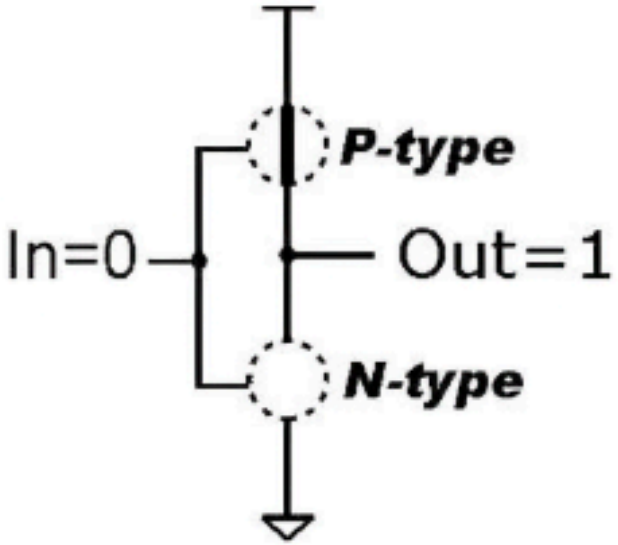
P-type is *complementary* to n-type

- When Gate has positive voltage, open circuit between #1 and #2 (switch open)
- When Gate has zero voltage, short circuit between #1 and #2 (switch closed)



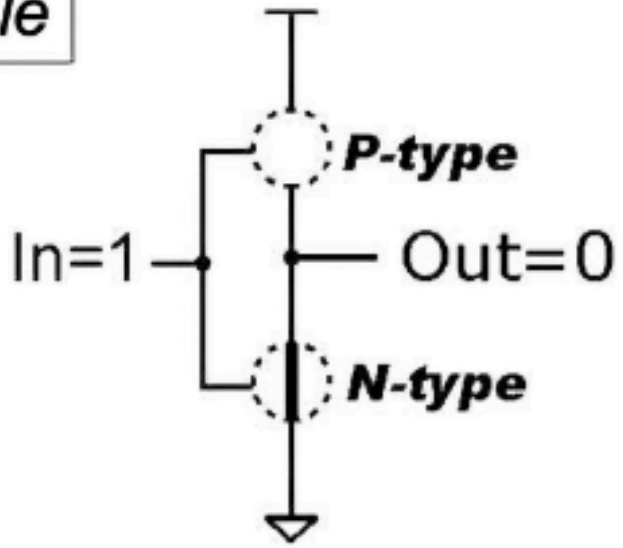
Inverter (NOT Gate)

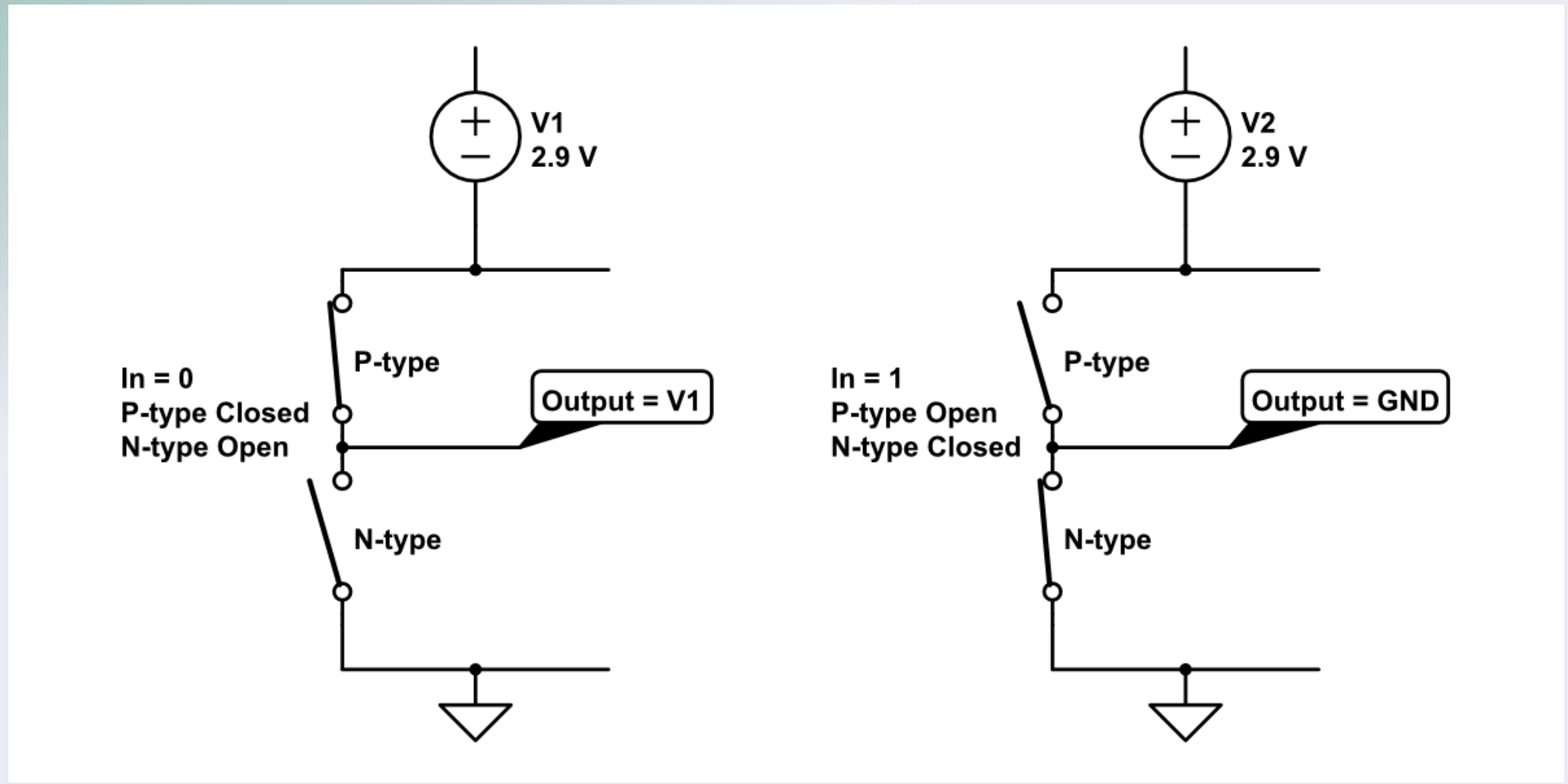




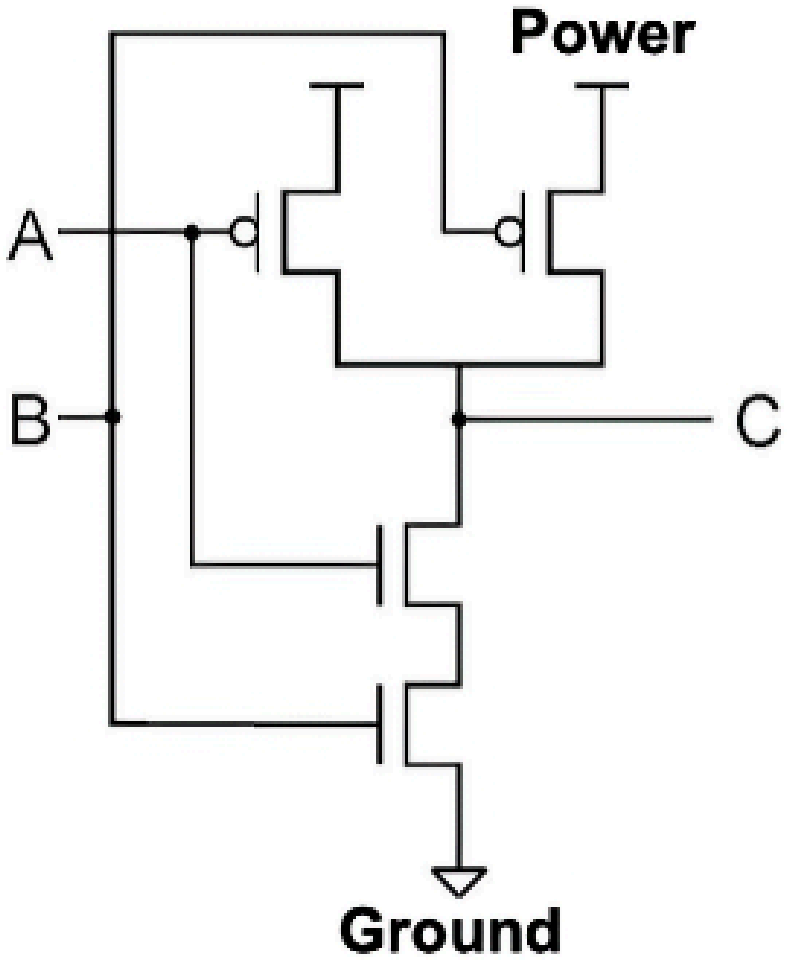
Truth table

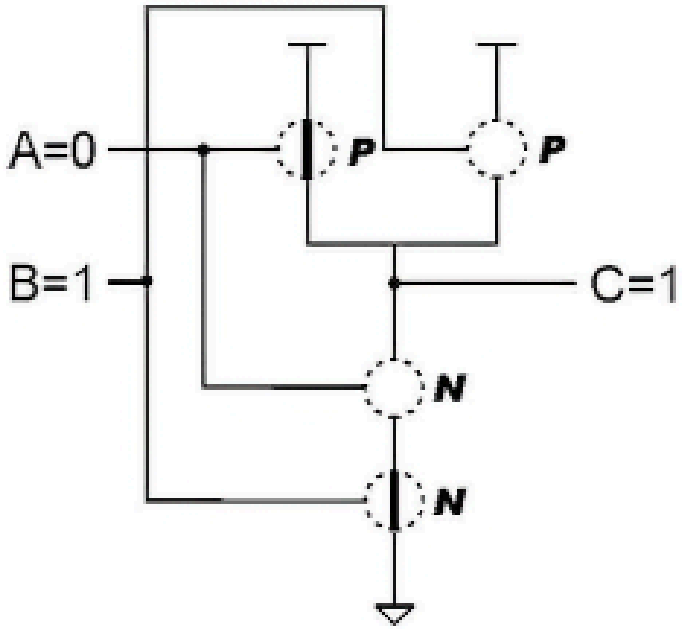
In	Out
0	1
1	0




V1
2.9 V
In = 0
P-type Closed
N-type Open
P-type
Output = V1
N-type
V2
2.9 V
In = 1
P-type Open
N-type Closed
P-type
Output = GND
N-type

NAND Gate (NOT-AND)

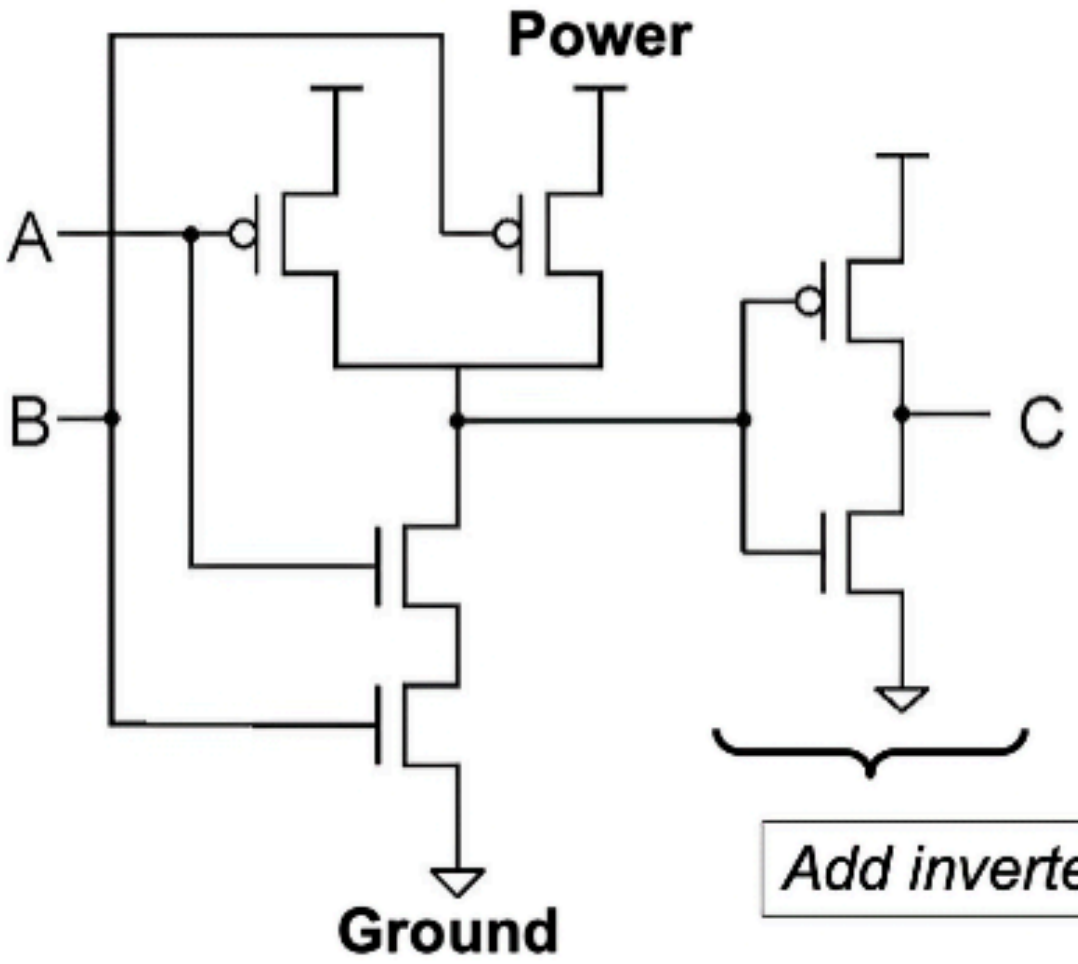




A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Note: Parallel structure on top, serial on bottom.

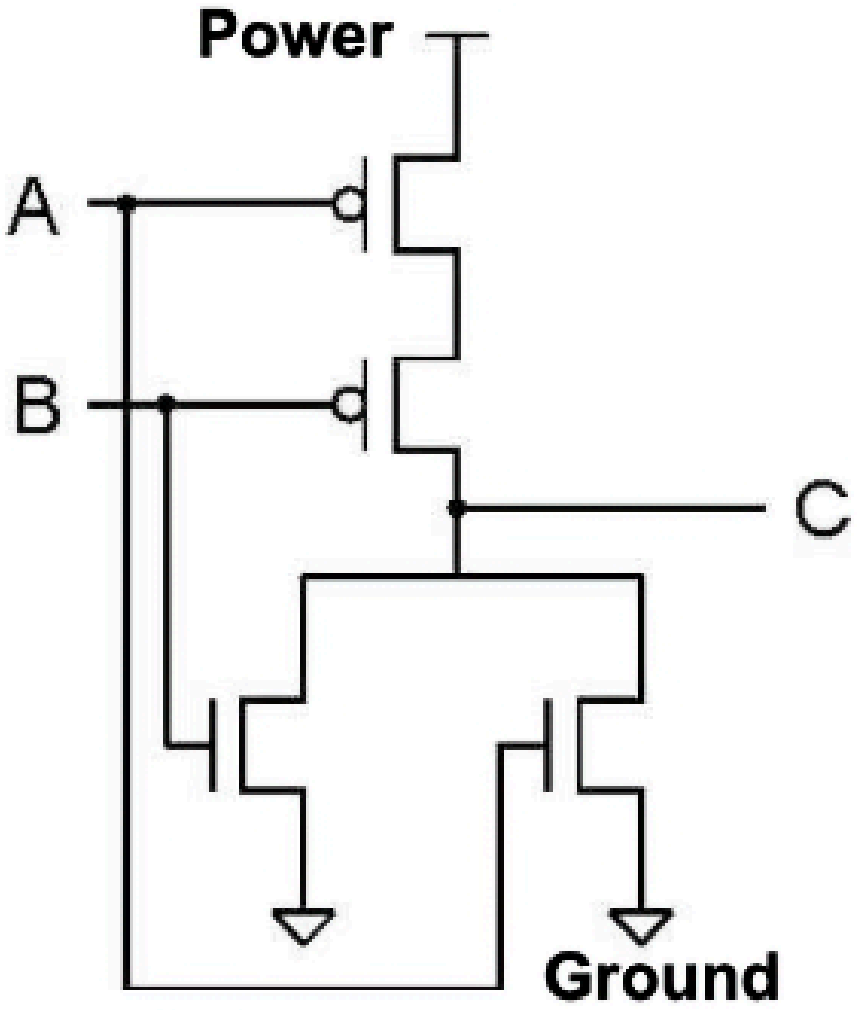
AND Gate

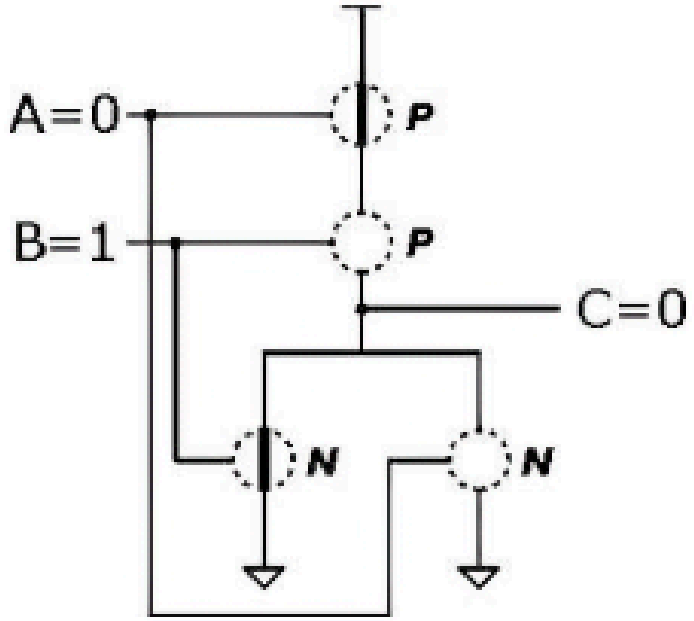


A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Add inverter to NAND.

NOR Gate (NOT-OR)

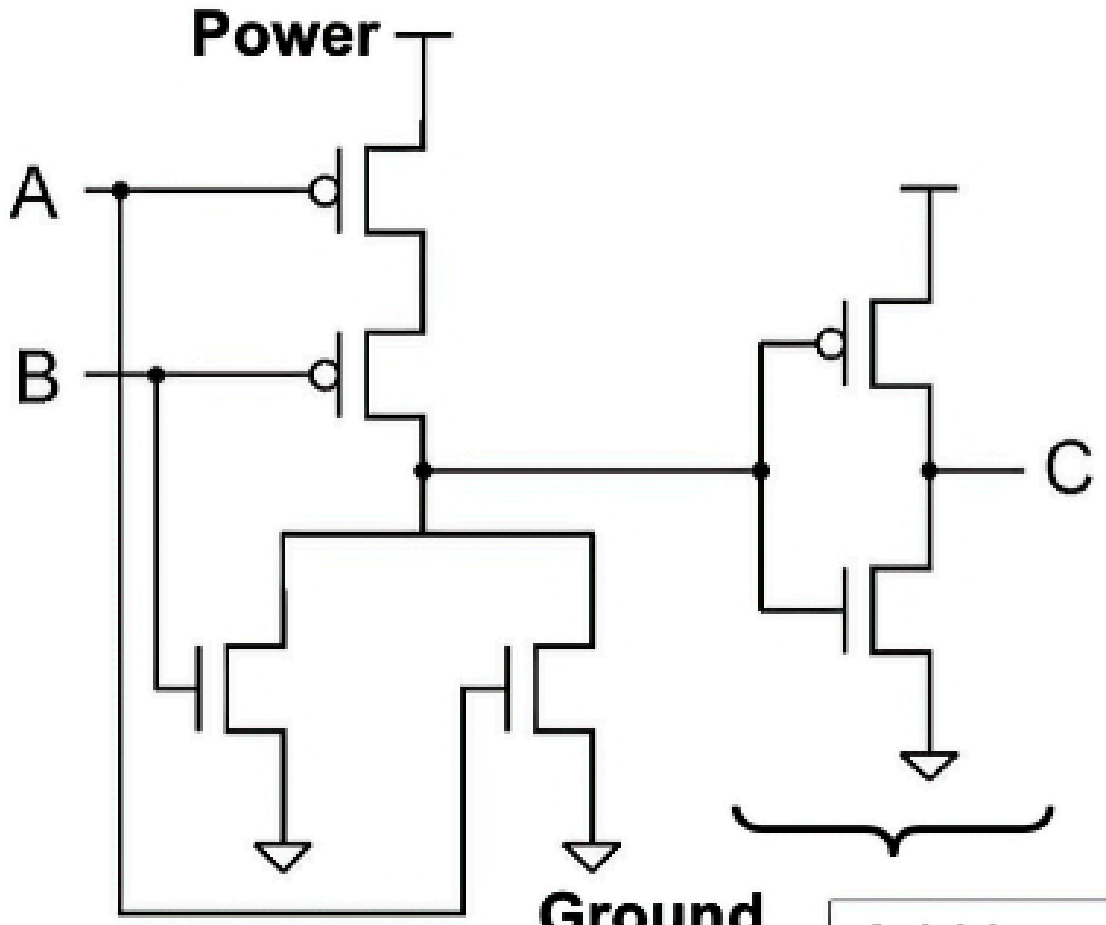




A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Note: Serial structure on top, parallel on bottom.

OR Gate



A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Add inverter to NOR.

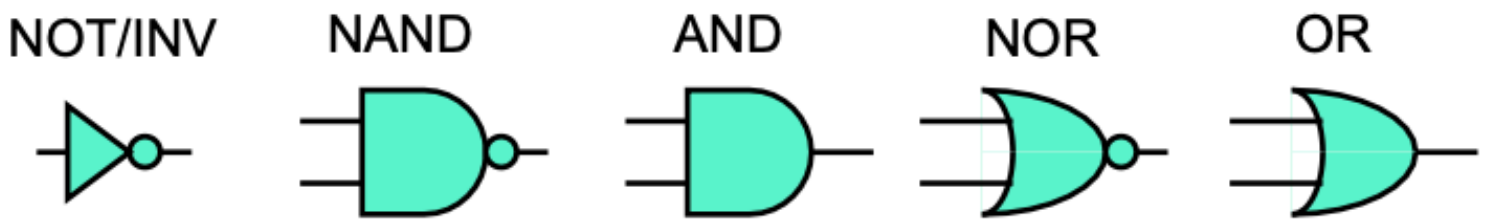
Basic Gates

From Now On… Gates

- Covered transistors mostly so that you know they exist
- Note: “Logic Gate” not related to “Gate” of transistors

Will study implementation in terms of gates

- Circuits that implement Boolean functions



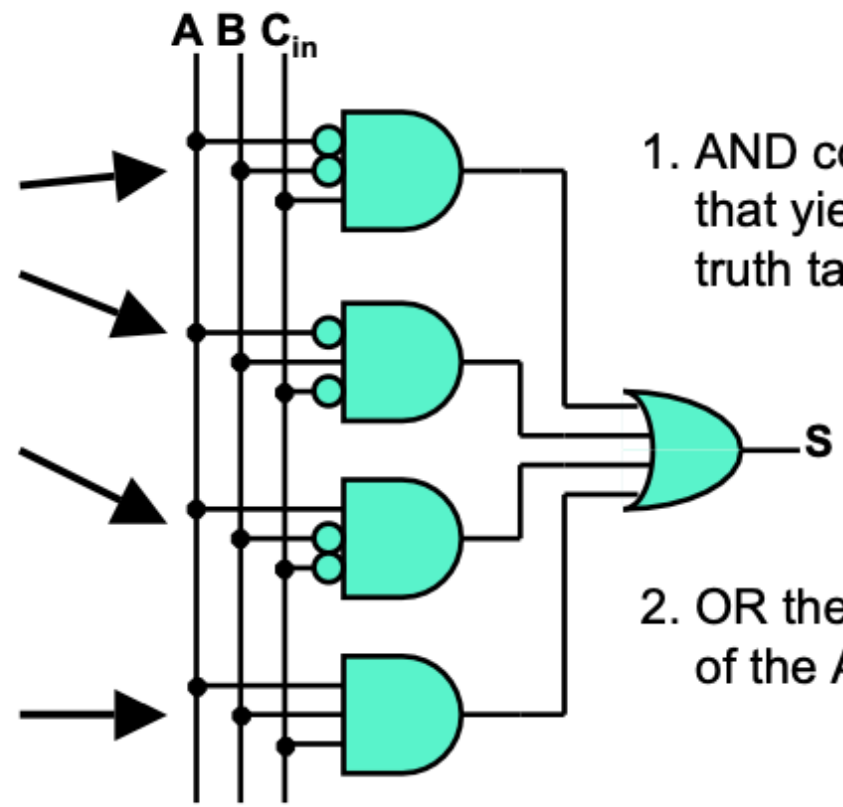
More complicated gates from transistors possible

- XOR, Multiple-input AND-OR-Invert (AOI) gates

Logical Completeness

AND, OR, NOT can implement ANY truth table

A	B	C_{in}	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



1. AND combinations that yield a "1" in the truth table

2. OR the results of the AND gates

Now What?

Before you go make a microprocessor, let's add the ability to simplify

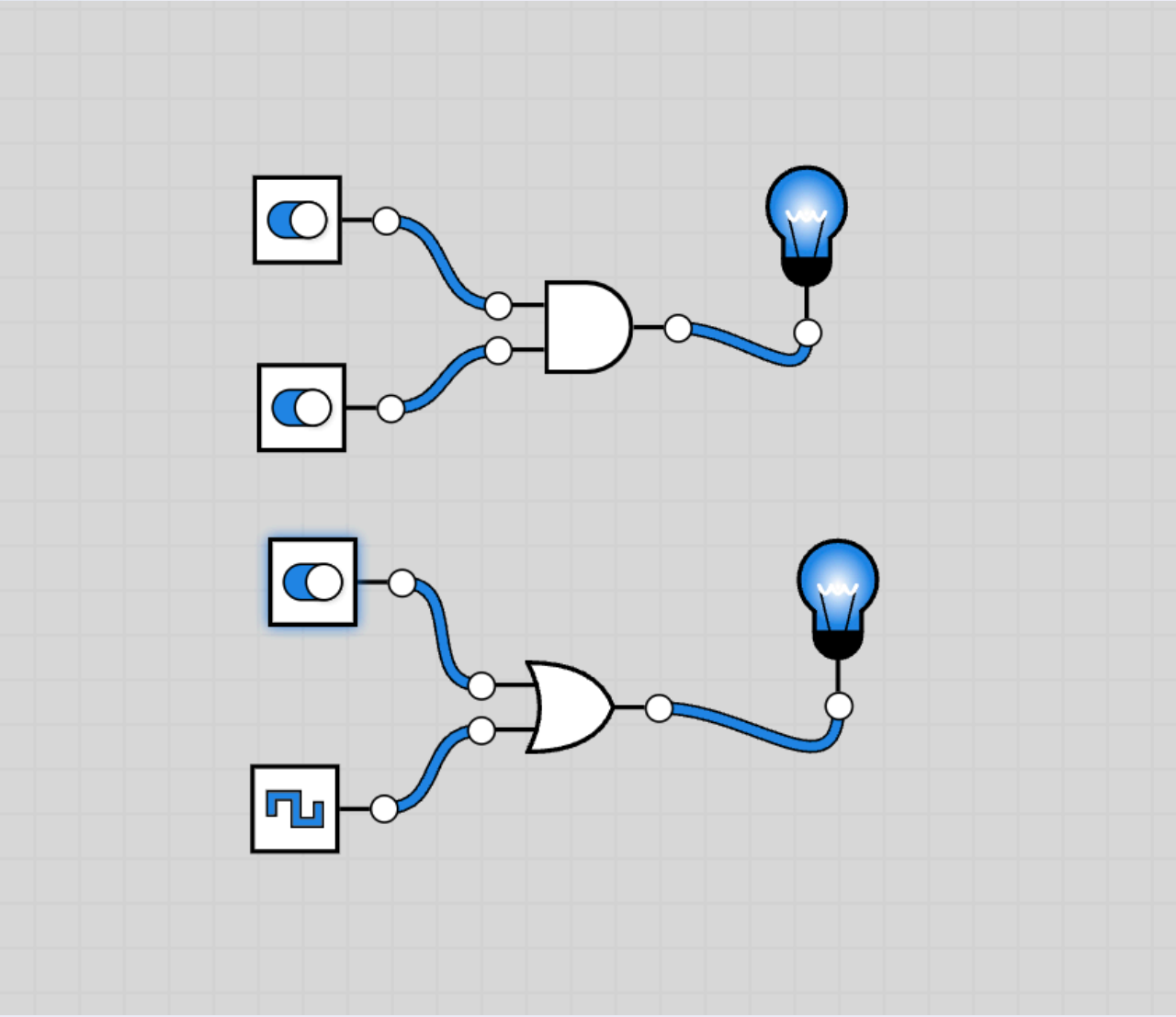
Boolean Algebra

A series of laws which help you to understand how to *simplify* and *solve* boolean expressions.

Digital Logic Simulator

Digital Logic Simulator*

*(*otherwise, known as don't take my word for it.*)



Process

1. I'll describe the boolean algebraic law
2. You implement in the simulator
3. Once you believe, you have a successful implementation, screenshot and save to a homework folder

Idempotent

A fundamental law in binary logic where an operation repeated on itself yields the same result. For example:

- A AND A = A
- A OR A = A
  This means performing the same logical operation multiple times doesn't change the outcome.

Annulment

Also known as the dominance law, this principle states that:

- A AND 0 = 0
- A OR 1 = 1

  When combined with these special values, the result is predetermined regardless of the other operand.

Identity

The identity law states that certain operations with specific values leave the operand unchanged:

- A AND 1 = A
- A OR 0 = A
  These values (1 for AND, 0 for OR) act as neutral elements in the operation.

Complement Law of Switches

This law deals with complementary operations:

- A AND (NOT A) = 0
- A OR (NOT A) = 1
  When a value is combined with its complement using AND or OR, the result is always definite.

Double Complement Law

States that negating a value twice returns the original value:

- NOT(NOT A) = A
  This is similar to how two negatives make a positive in regular mathematics.

Commutative Law

The order of operands doesn't affect the result:

- A AND B = B AND A
- A OR B = B OR A
  You can swap the position of values and get the same outcome.

Distributive Law

Similar to algebra, one operation distributes over another:

- A AND (B OR C) = (A AND B) OR (A AND C)
- A OR (B AND C) = (A OR B) AND (A OR C)
  This allows for restructuring complex expressions.

Absorptive Law

Describes how certain combinations of operations can be simplified:

- A AND (A OR B) = A
- A OR (A AND B) = A
  The additional term gets "absorbed" into the expression.

Associative Law

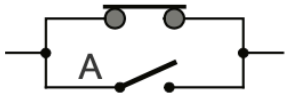
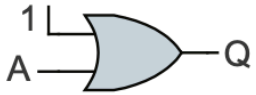
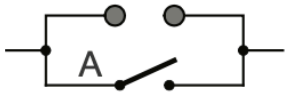
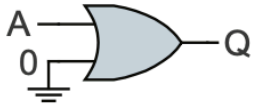
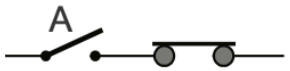
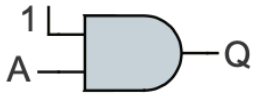
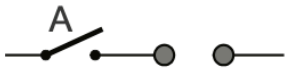
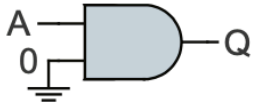
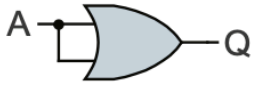
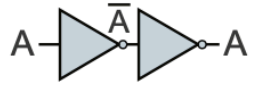
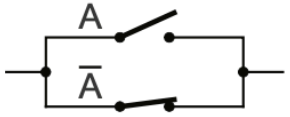
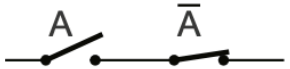
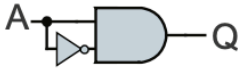
The grouping of operations doesn't affect the result:

- (A AND B) AND C = A AND (B AND C)
- (A OR B) OR C = A OR (B OR C)
  You can regroup terms without changing the outcome.

De Morgan's Theorem

A fundamental principle for simplifying logical expressions:

- NOT(A AND B) = (NOT A) OR (NOT B)
- NOT(A OR B) = (NOT A) AND (NOT B)
  This law shows how negation distributes across AND/OR operations.

Boolean Expression	Description	Equivalent Switching Circuit	Equivalent Logic gate	Boolean Algebra Law or Rule
A+1 = 1	A in parallel with closed = CLOSED			Annulment
A+0 = A	A in parallel with open = A			Identity
A.1 = A	A in series with closed = A			Identity
A.0 = 0	A in series with open = OPEN			Annulment
A+A = A	A in parallel with A = A			Idempotent
A.A = A	A in series with A = A			Idempotent
NOT(not-A) = A	NOT NOT A = A (double negative)			Double Negation
A+$\overline{A}$ = 1	A in parallel with NOT-A = CLOSED			Complement
A.$\overline{A}$ = 0	A in series with NOT-A = OPEN			Complement

Credits:

Patt and Patel: "*Introduction to Computing Systems*"

AspenCore: "*Boolean Algebra*"

UPenn CSE240 Fall 2006 Fassell: "*Chapter 3 Lecture*"

IEEE 2021: "*More Moore*"