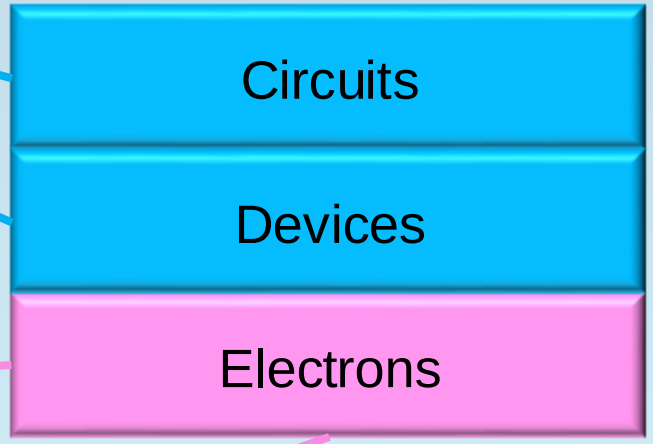
University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Logic Gates

Today: How Can We Build Gates?

3. Functions on bits (Boolean operators, gates)
4. Implementation?
2. Representations based on bits
1. Two voltage levels → 1 bit



How can we build gates?

But First: Check Out My New Invention!

Last night I had a great idea.

I call it a **"torch."**

At night, you can
point it at things.

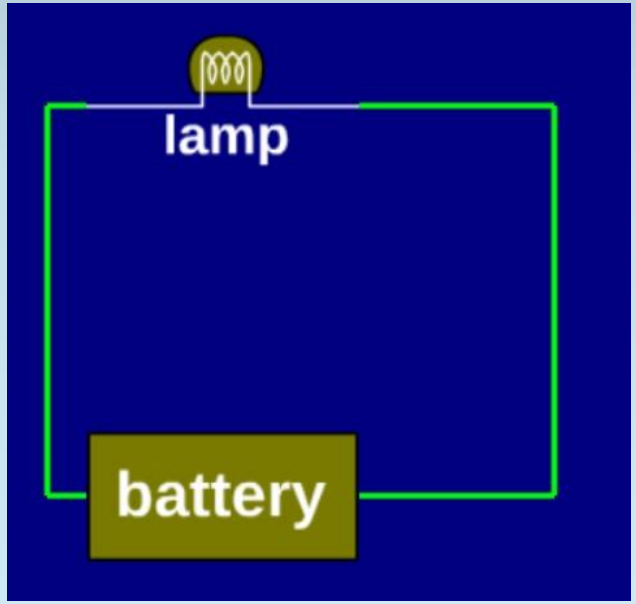
And **they will be lit up**.

Anything!

Your car or bike.

Your door lock.

A friend.



What do you think?

You Think I Should Do What?

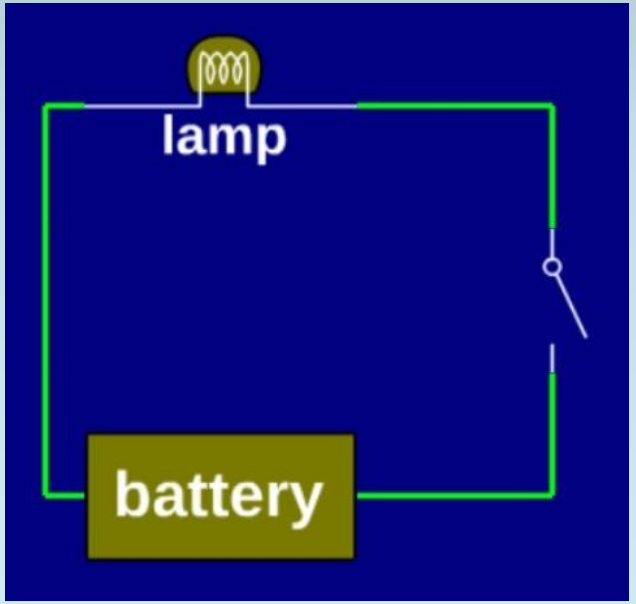
Like this?

I think **people already make those**.

The switch is **controlled by your thumb**.

They call it a **flashlight**.

I won't be able to patent it.



Don't Worry: Here's Another Idea

So, you like switches?

Let's put a bunch of switches together.

Each controlled by ~~our~~ your thumbs.

When we want **to change a bit**,

we will just **flip a switch**!

We'll call it a **hand-operated computer**!

We'll need about **2,000,000,000** switches.

What do you think?

Still Don't Like It? One Last Try…

What if we develop a **voltage-controlled switch**?

Then **one switch**
- can **control another switch,**
- which can **control a third switch,**
- **and so on**!

Instead of using **your thumbs**, we can **build circuits with 2,000,000,000 switches**!

Now THAT's a really cool idea!

Digital Electronics is Based on MOSFETs

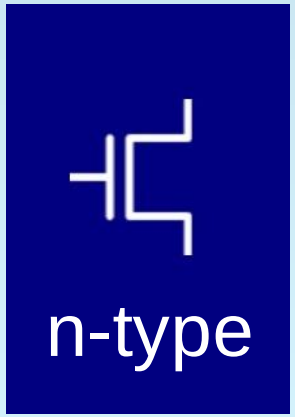
Digital electronics today uses MOSFETs.

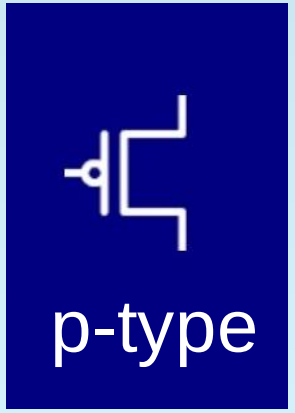
- the material: Metal-Oxide Semiconductors
- the mechanism: Field-Effect Transistors (electric field/voltage-controlled)

There are two kinds, named after the charge carrier,

- **n**(egative)**-type**, and
- **p**(ositive)**-type**,

drawn as shown here.



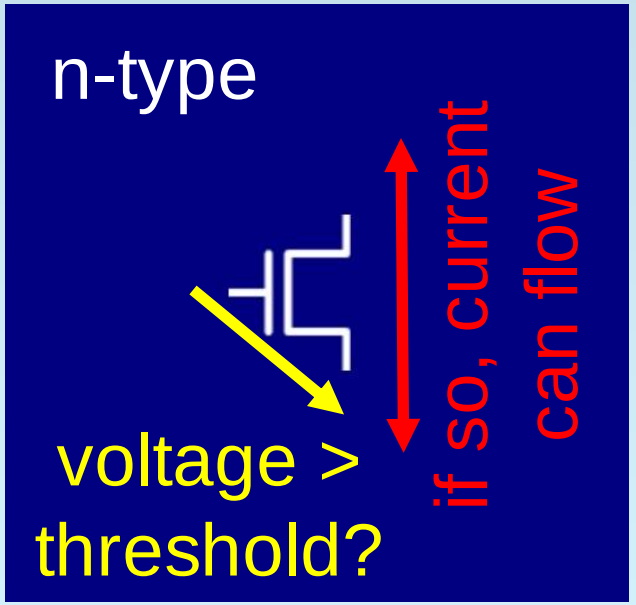


n-type is On With Positive Gate to Source/Drain Voltage

An **n-type MOSFET**

- turns **on** (switch is **closed**, allowing current to flow)
- if the **voltage from gate** (left terminal) **to other terminals exceeds a threshold**

If the voltage is smaller, the transistor is **off** (the switch is **open**).



Our Voltages Will Be Binary

We need two voltages:

- **0V**, a ground
  (this is the binary 0 value)

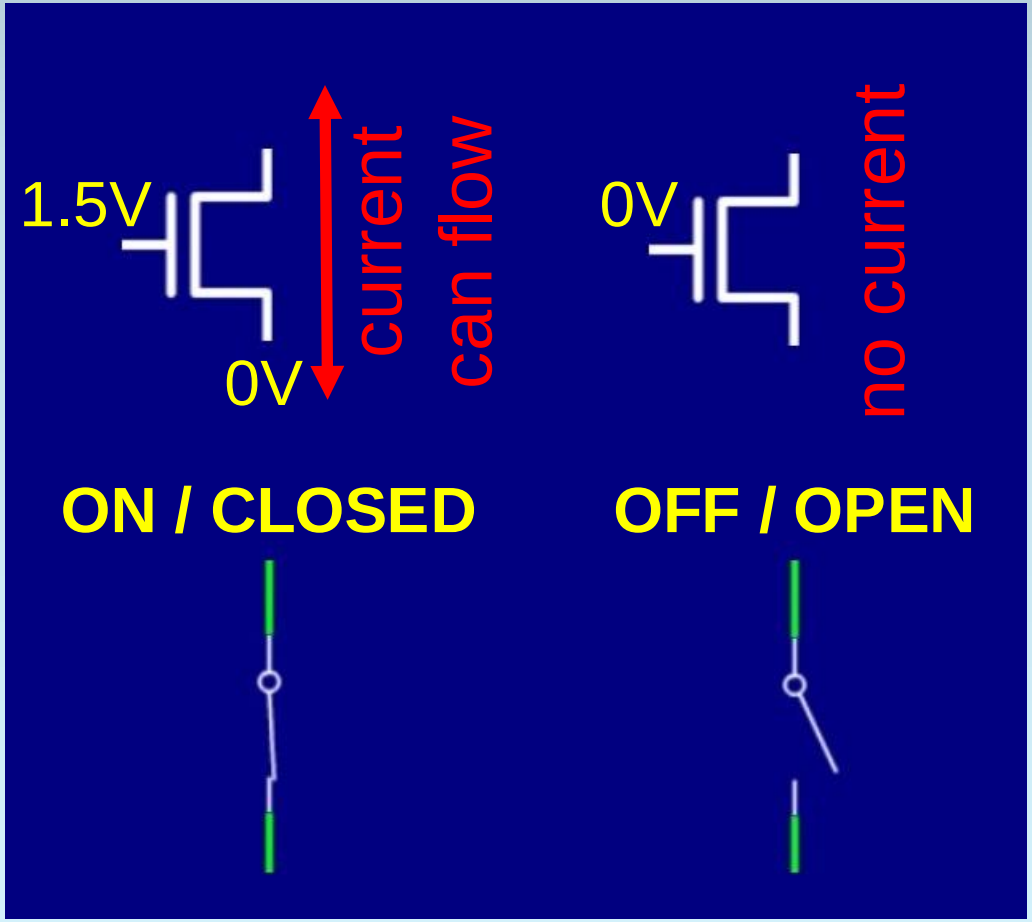
- **V_{dd}**, around **1.5V**, high voltage*
  (this is the binary 1 value)

*Used to be 5V, but has been decreasing for decades.
The rate of decrease is now slowing down.

Use Binary Voltages to Control n-Type MOSFETs

n-type only **turns on when gate voltage is high** (V_{dd}).

An **n-type** can pull one terminal down to **0V** with the other terminal.

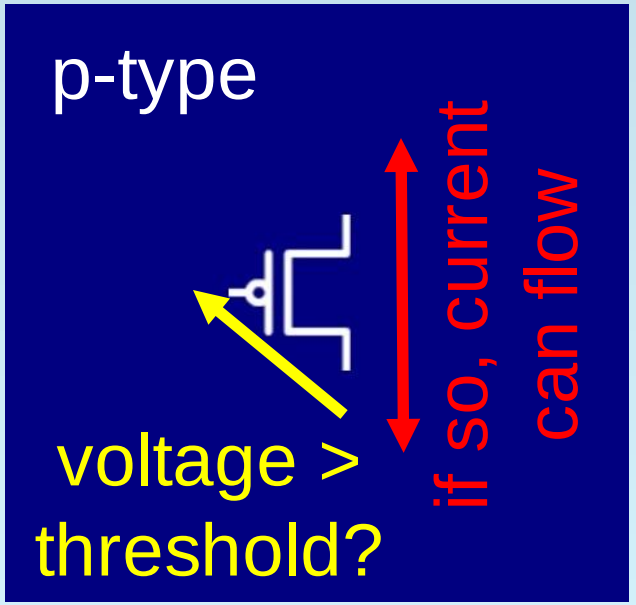


p-type is On With Negative Gate to Source/Drain Voltage

A **p-type MOSFET**

- turns **on** (switch is **closed**, allowing current to flow)
- if the **voltage from other terminals to the gate** (left terminal) **exceeds a threshold**

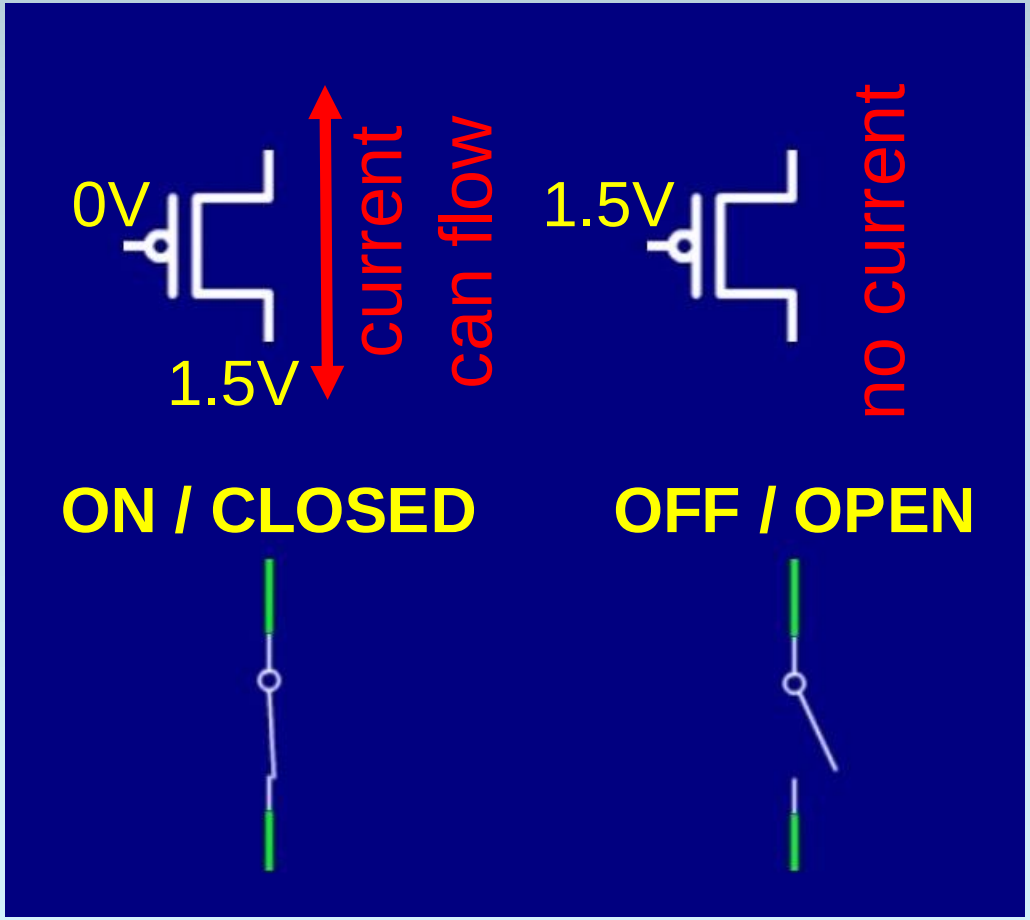
If the voltage is smaller, the transistor is **off** (the switch is **open**).



Use Binary Voltages to Control p-Type MOSFETs

p-type only **turns on when gate voltage is low** (**0V**).

A **p-type** can pull one terminal up to $\mathbf{V_{dd}}$ with the other terminal.



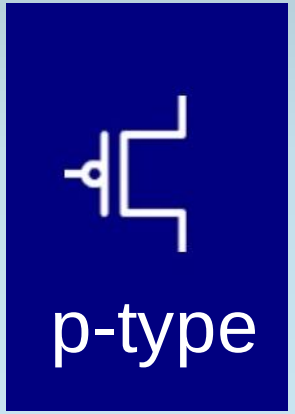
The Drawings Help You Remember How They Work

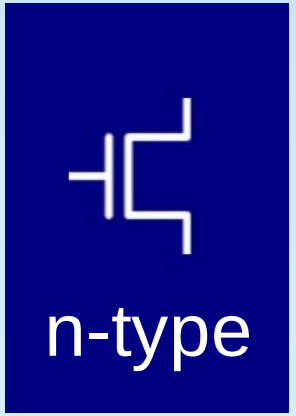
Notice the use of the inverter bubble on the **p-type**.

Use it to help you remember:

- **p-type turns on with low voltage** (**0V**, or binary **0**).
- **n-type turns on with high voltage** (**Vdd**, or binary **1**).

The names may not be so helpful (again, they refer to charge carriers).





Gates are Based on Complementary MOS (CMOS)

So how do we build gates?

Gates use complementary structures of **p-type** and **n-type MOSFETs**.

Each gate uses an equal number of each type.

For that reason, we say that

- most **digital systems are based on CMOS**,
- or Complementary MOS.

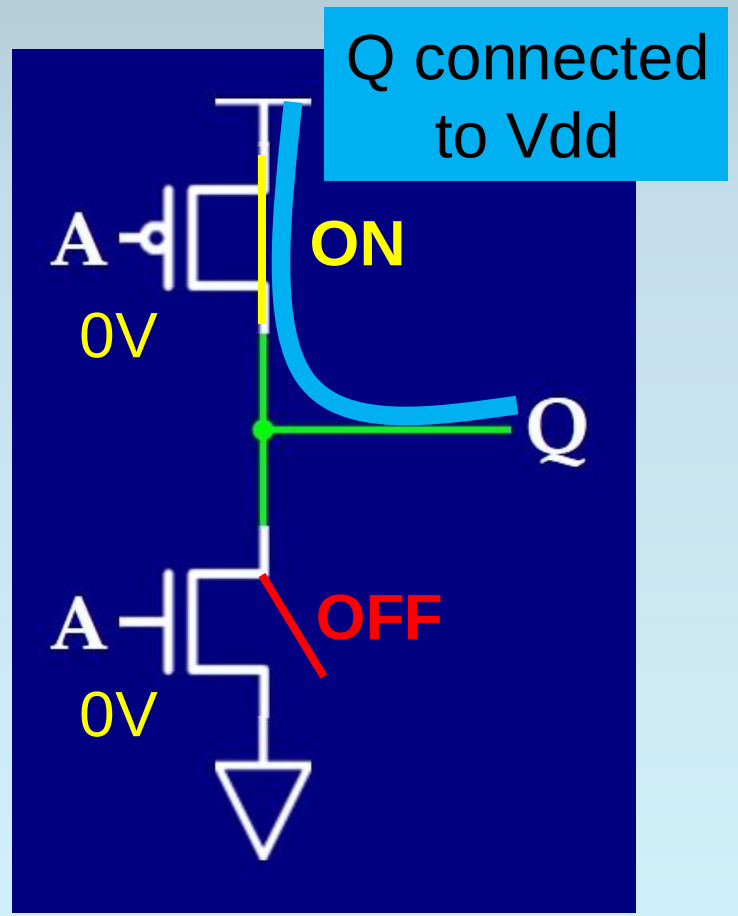
What Does This Gate Do? (when A=0V)

Here is the simplest gate.

What does it do?

Let's write a truth table!

A	Q
0V	1.5V
1.5V	



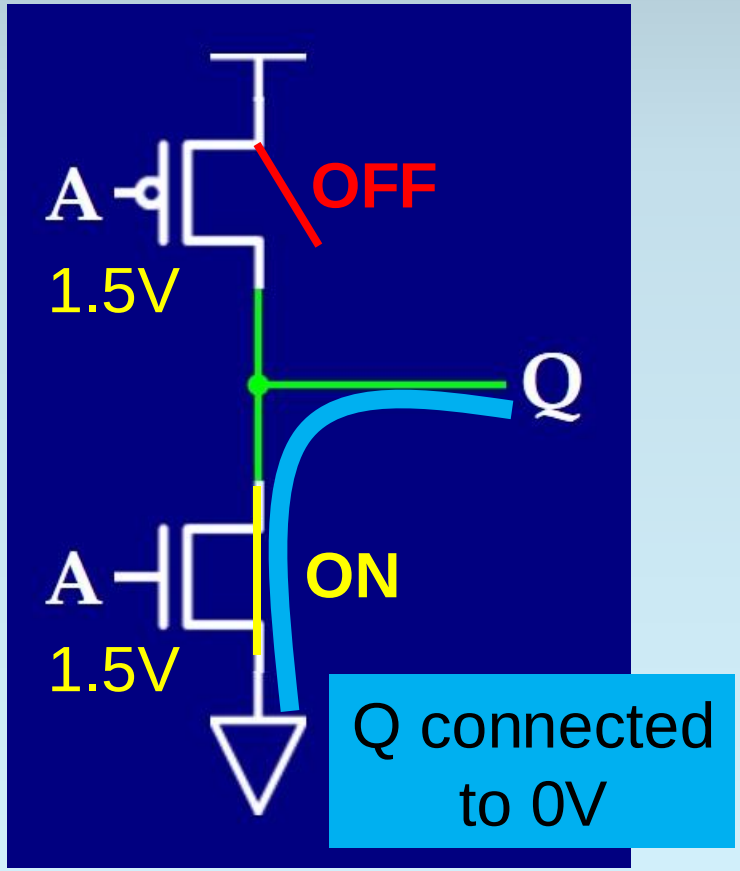
What Does This Gate Do? (when A=1.5V)

Here is the simplest gate.

What does it do?

Let's write a truth table!

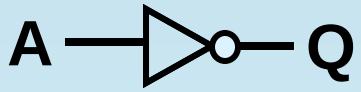
A	Q
0V	1.5V
1.5V	0V



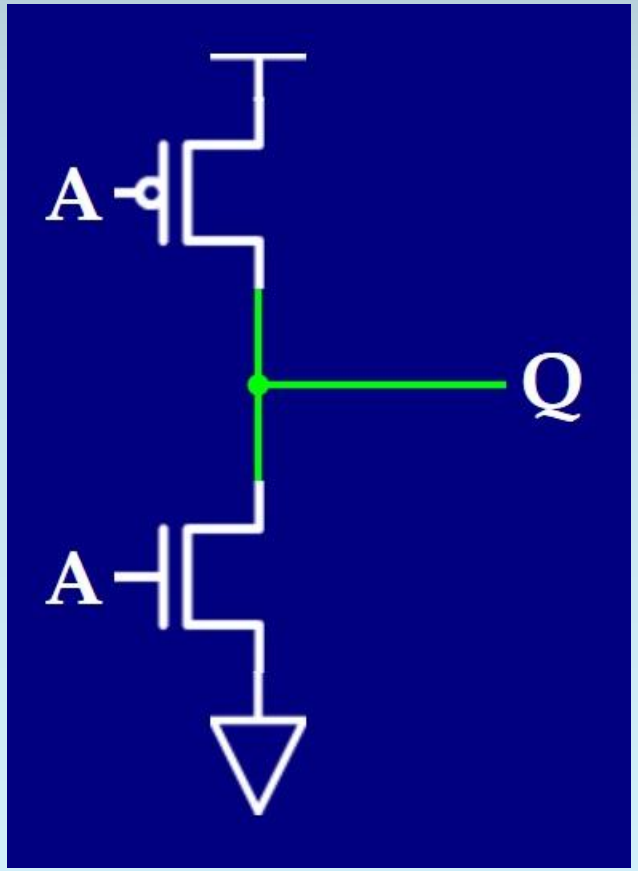
It's a NOT Gate!

Now convert the truth table from voltages to binary.

It's a NOT gate!



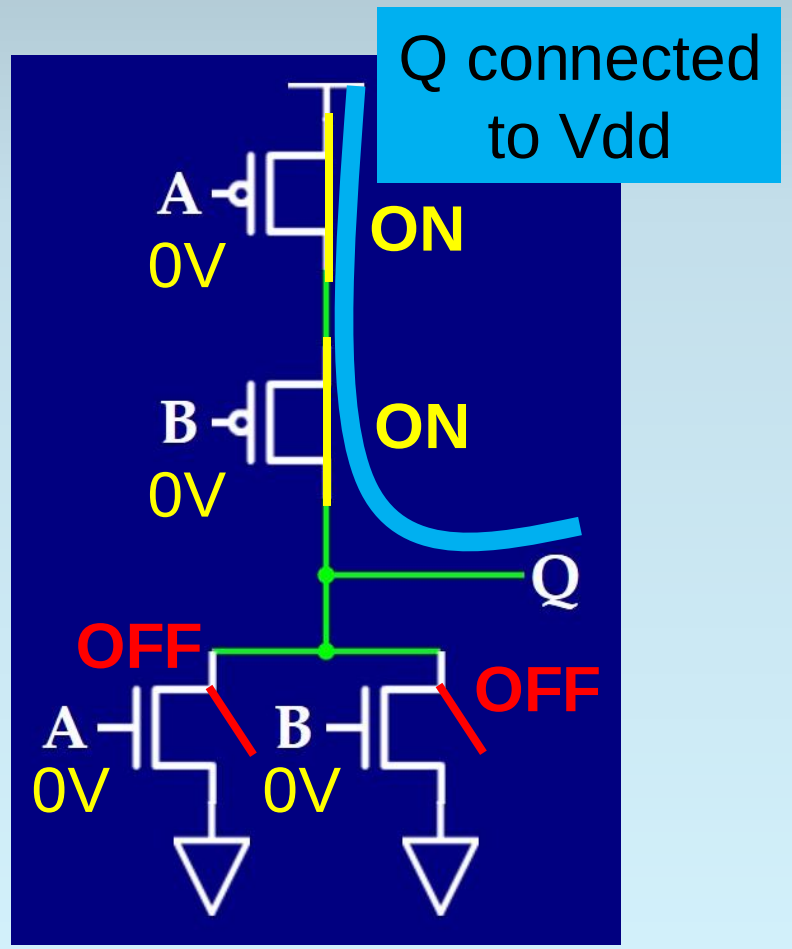
	A	Q	
(0)	0V	1.5V	(1)
(1)	1.5V	0V	(0)



Let's Analyze Another Structure

What about this structure?

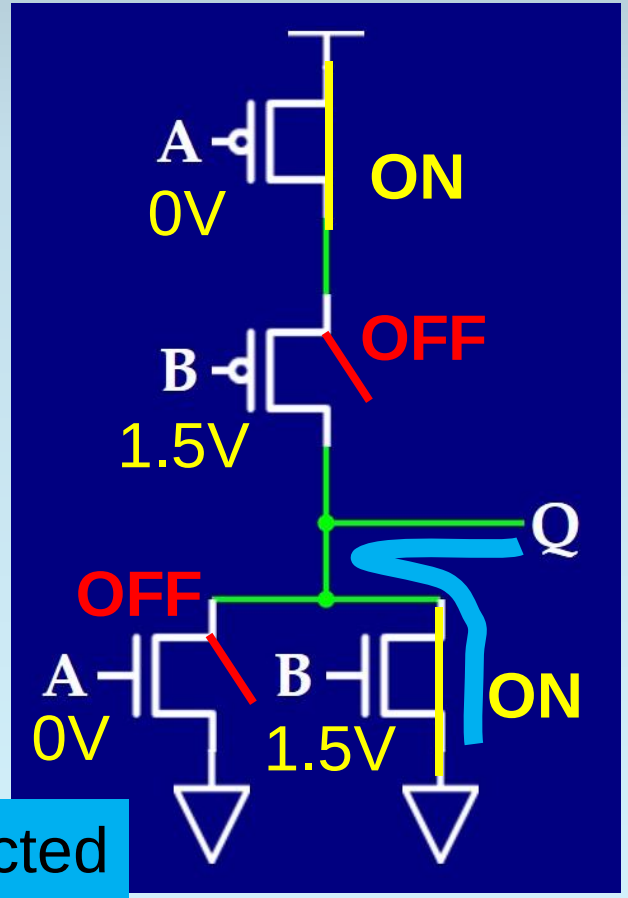
A	B	Q
0	0	1
0	1	
1	0	
1	1	



Next, Assume A = 0 and B = 1

The A value is the same, so we leave the markings.

A	B	Q
0	0	1
0	1	0
1	0	
1	1	

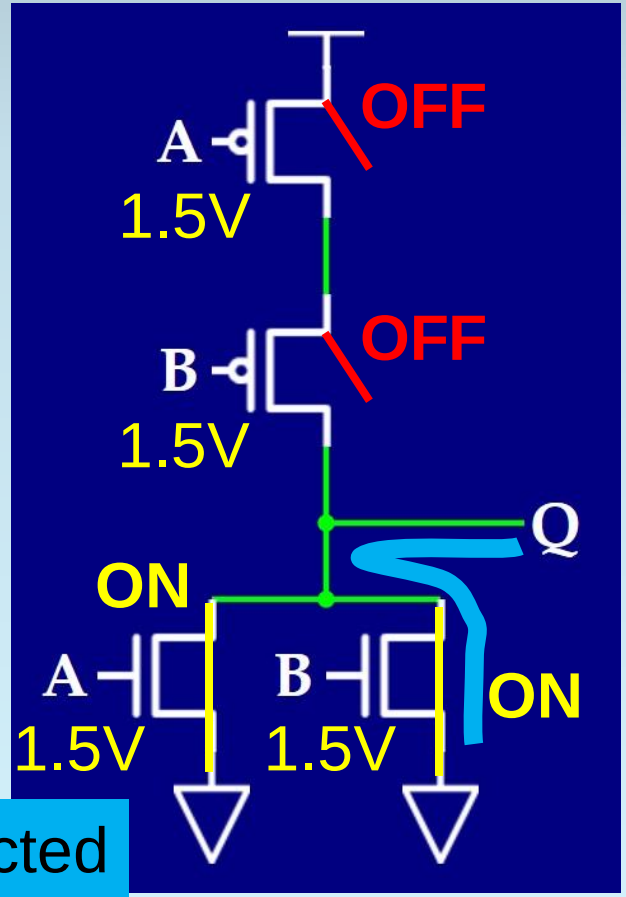


Q connected to 0V

Next, Assume A = 1 and B = 1 (BOTTOM LINE!)

The B value is the same, so we leave the markings.

A	B	Q
0	0	1
0	1	0
1	0	
1	1	0

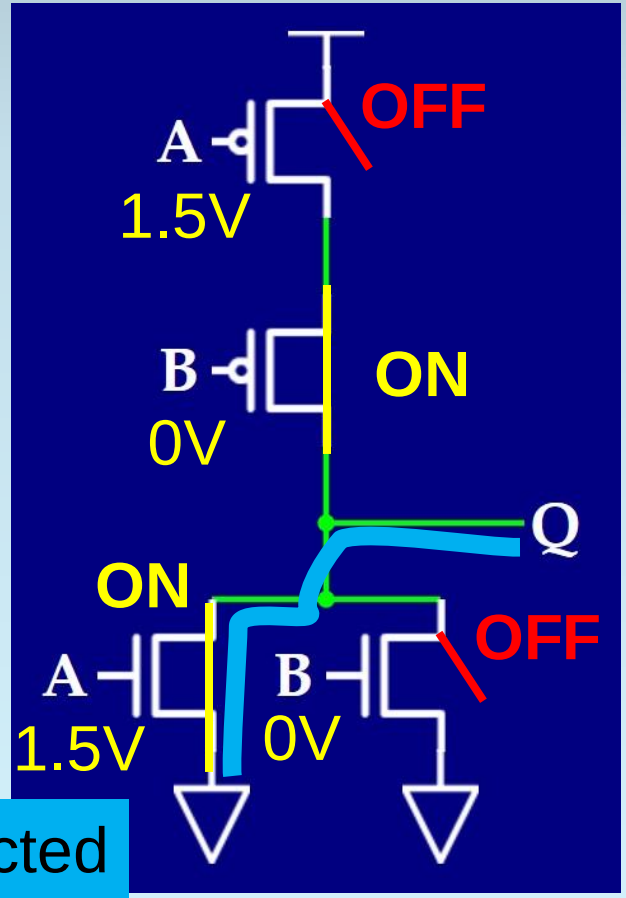


Q connected to 0V

Finally, Assume A = 1 and B = 0

The A value is the same, so we leave the markings.

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

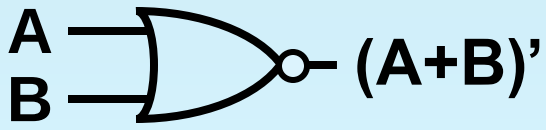


Q connected to 0V

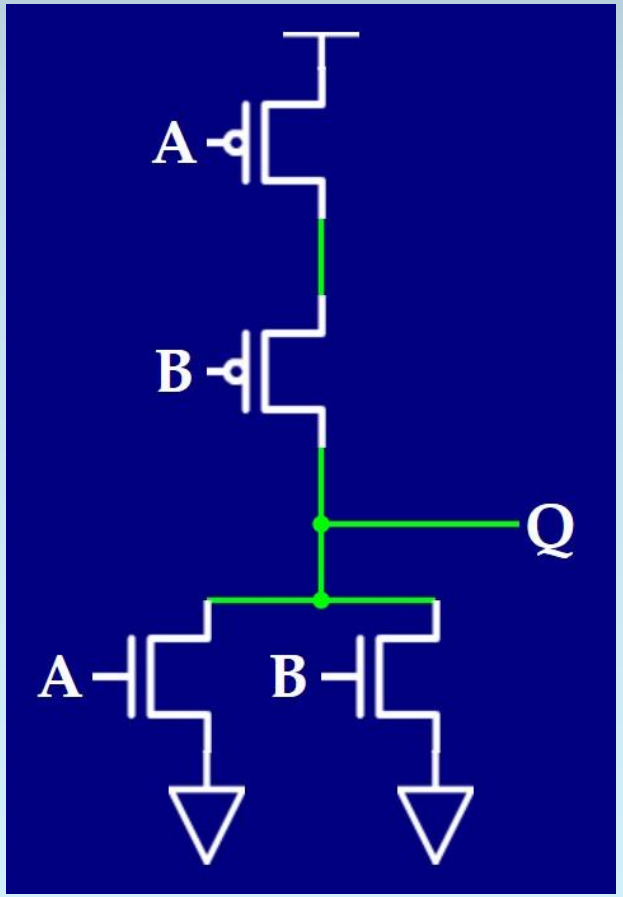
It's a NOR Gate!

We see that **Q = (A+B)'**.

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0



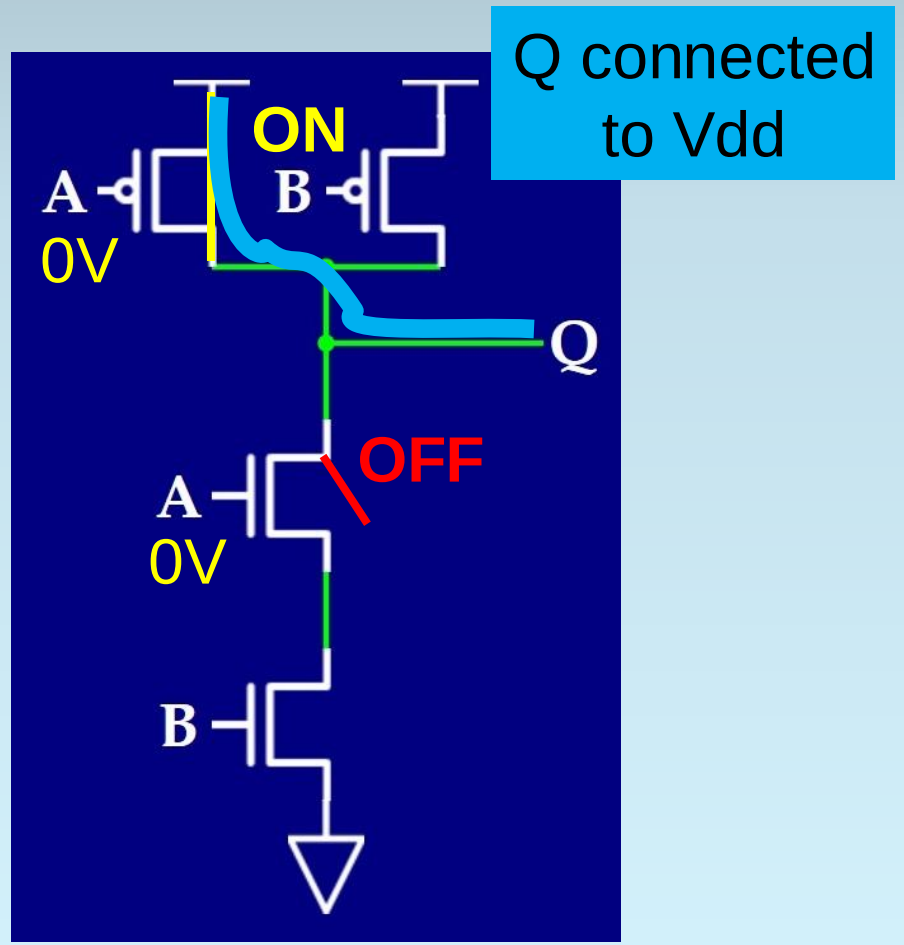
NOR gate



And Just One More to Analyze…

What if A=0?

A	B	Q
0	0	1
0	1	1
1	0	
1	1	

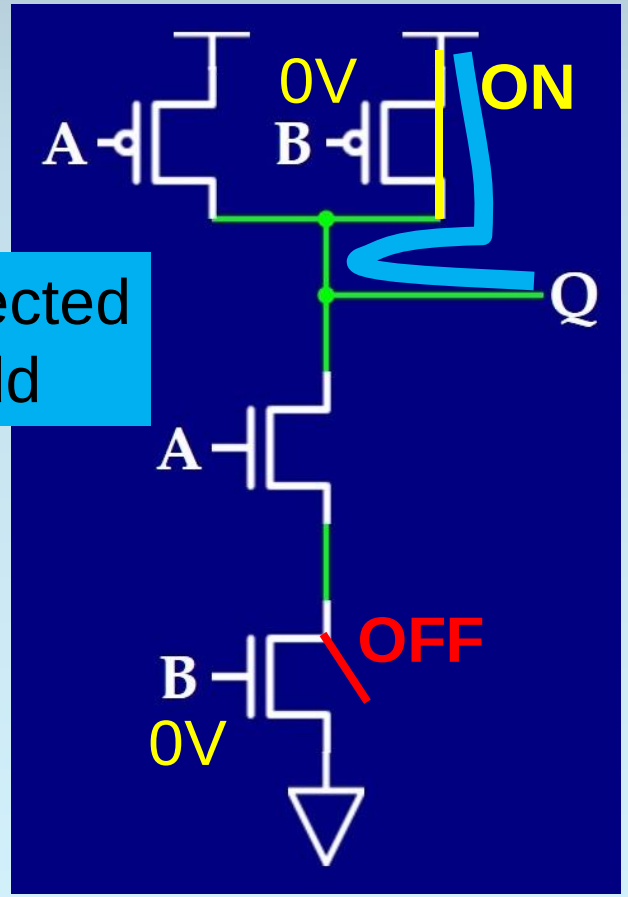


Notice that the Circuit is Symmetric in A and B

What if B=0?

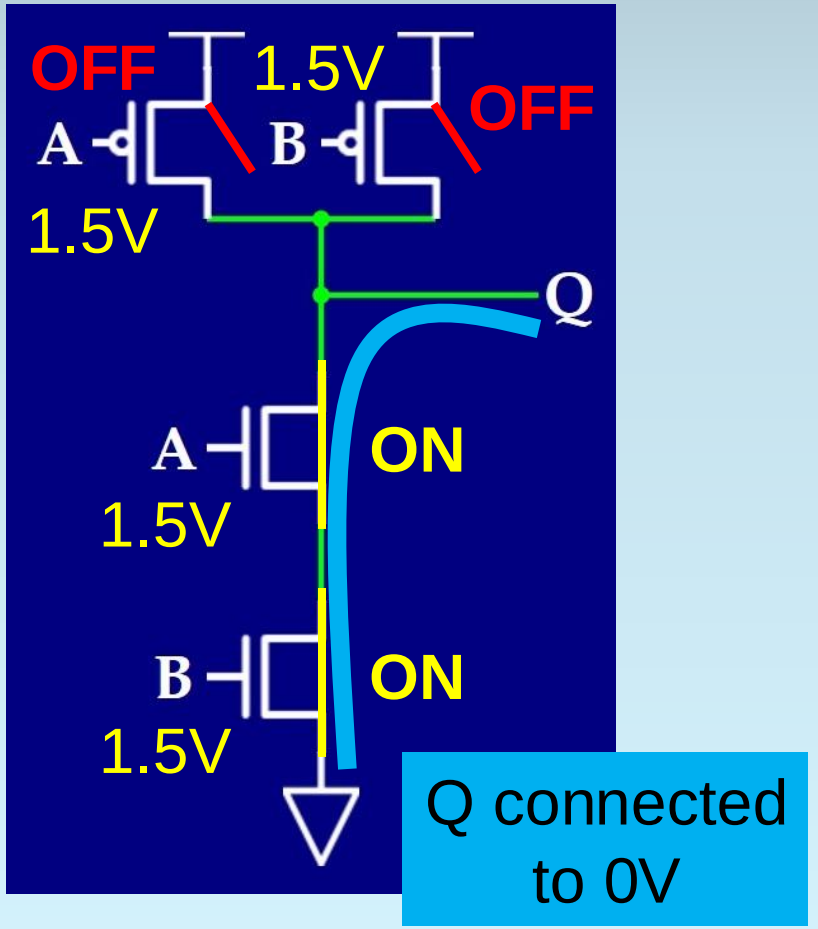
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	

Q connected to Vdd



And if Both A = 1 and B = 1?

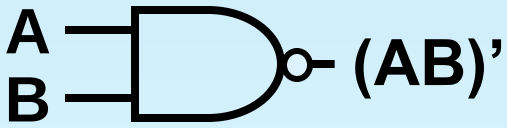
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



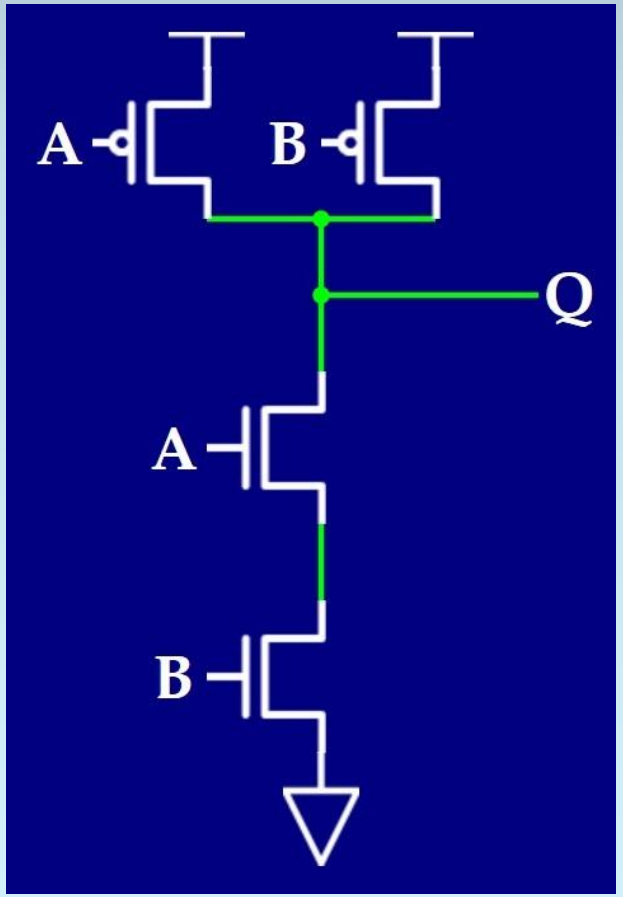
It's a NAND Gate!

We see that **Q = (AB)'**.

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



NAND gate



Generalizing to More Inputs

Notice the common features

- p-type always connected to Vdd.
- n-type always connected to 0V.
- One side is parallel, the other is serial (they are duals* of one another).

Can you generalize NAND/NOR to more inputs?

Let's try it in the online tool…

*See Notes Section 2.2.1.

A Couple of Practical Limits

Gates scale to about 4 inputs before using more gates is a better approach.

One can easily

- design an AND or an OR gate with CMOS
- by swapping n-type with p-type,
- but MOSFETs don't work properly in those designs.
- Try it in the online tool to see what happens.
- (NAND followed by NOT is, of course, AND.)

University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Optimizing Logic Expressions

We Can Use Logical Completeness to Express Functions

Let the truth table to the right define the **function F**.

Recall that we can use the logical completeness construction to write **F** as a Boolean expression:

- This row is… **AB'C**
- And this is… **ABC'**
- And this is … **ABC**

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

What's the Best Way to Write Function F?

So **F = AB'C + ABC' + ABC**

But we can also write
F = AB + AC.

What about **F = A (B + C)**?

Which one is best?

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

Your Answer Is Wrong! Choose a Metric First

The answer depends on our **choice of metric!**

How do we measure good?

Common answers for circuit design:

- area / size / cost, OR
- performance / speed, OR
- power / energy consumption, OR
- complexity / reliability.

We Use Heuristics for These Metrics

In practice, **measuring exactly is expensive** (~$50-100M for a full design, and ~$2-5M just for trying something.)

Instead, we use **heuristics**, which are ways of **estimating a metric**.

A good heuristic is

- **reasonably accurate**, and
- **monotonic** relative to a real measurement
- (so that bigger estimates mean bigger measurements).

An Area Heuristic for ECE120

Here's a **heuristic for area**:

- **Count literals** (A, A', B, B', C, C'), then
- **Add the number of operations** (not including complements for literals).

Why does it work? Remember gate structures?

- each input (literal) → two transistors
- operators into operators → two transistors

So it gives an approximate **transistor count**.

(But wires also take space!)

A Delay (Speed) Heuristic for ECE120

Here's a **heuristic for delay / speed**:

- **Find the maximum number of gates between any input and any output.**
- Do not include complements for literals.

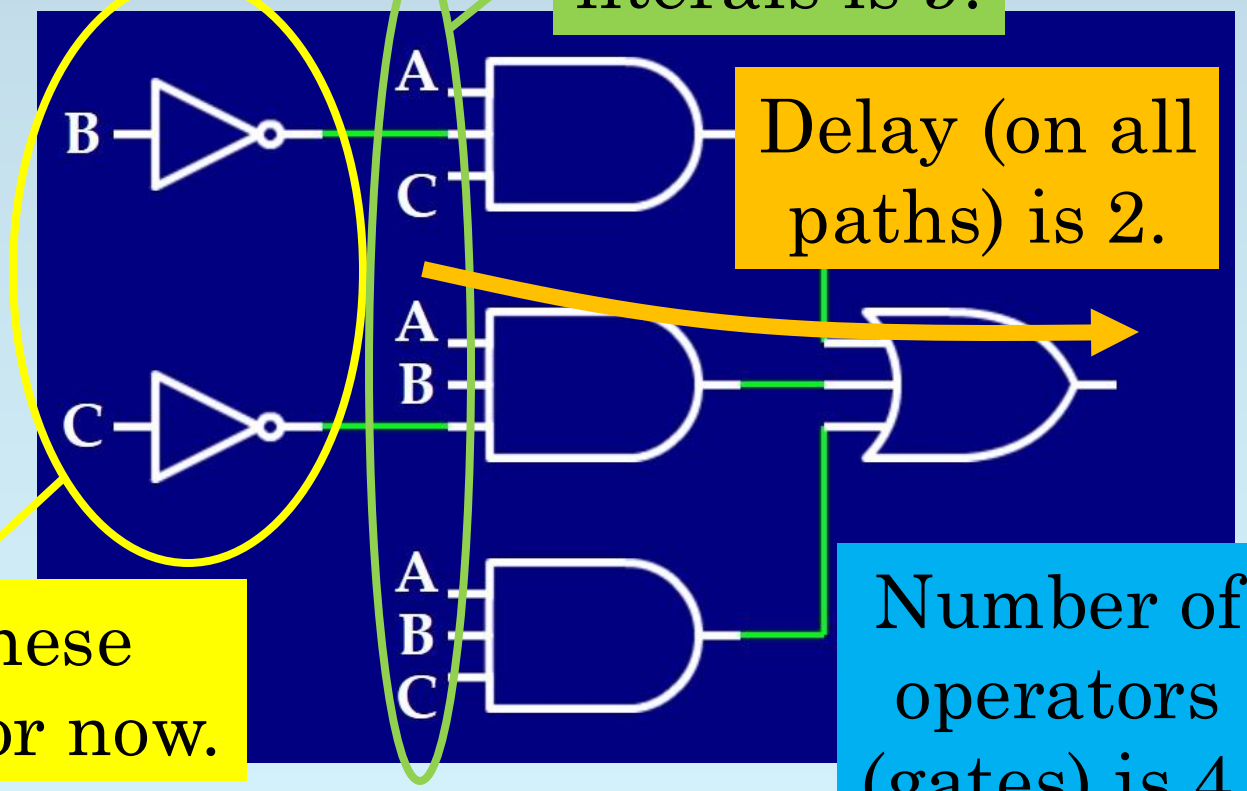
Why does it work?

- Each gate takes time switch its output on/off.
- We call this time a **gate delay**.

So it gives an approximate **delay** between inputs changing and outputs changing**.**

A Graphical View May Make Counting Easier

F = AB'C + ABC' + ABC



Number of literals is 9.

Delay (on all paths) is 2.

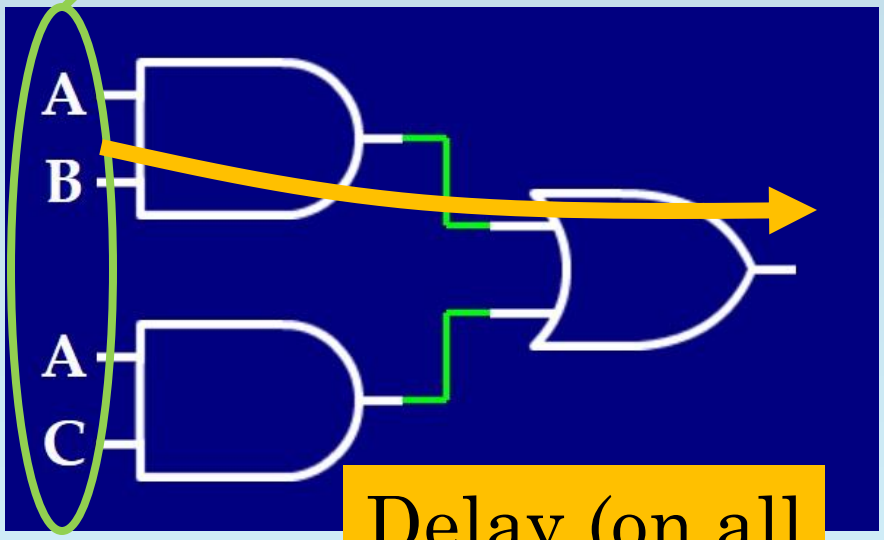
Ignore these inverters for now.

Number of operators (gates) is 4.

Let's Analyze the Second Form of F

$F = AB + AC$

Number of literals is 4.



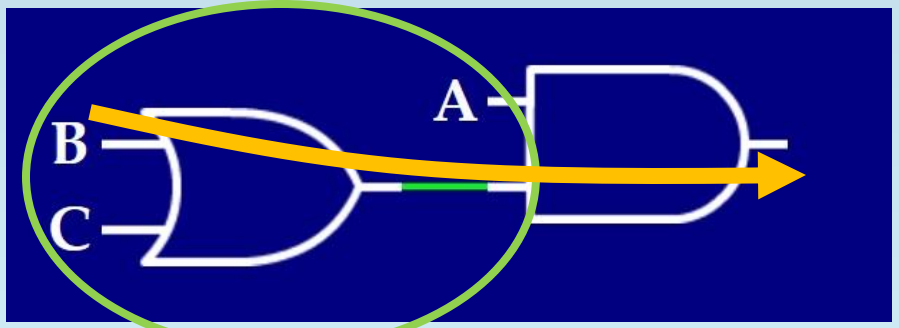
Delay (on all paths) is 2.

Number of operators (gates) is 3.

Let's Analyze the Third Form of F

$F = A\,(B + C)$

Number of literals is 3.



Number of operators (gates) is 2.

Delay (on longest paths) is 2.

The Area Heuristic Favors F = A (B + C)

Let's calculate the area heuristic for our three forms of F.

So **F = A (B + C) is the smallest design.**

Form of F	Lits	Ops	Area
AB'C + ABC' + ABC	9	4	13
AB + AC	4	3	7
A (B + C)	3	2	5

All Forms Are Equivalent in Delay

All designs are the same for delay.

Form of F	Lits	Ops	Area	Delay
AB'C + ABC' + ABC	9	4	13	2
AB + AC	4	3	7	2
A (B + C)	3	2	5	2

We Have a Winner: F = A (B + C)

F = A (B + C) is best by both metrics.

But the answers are not always so simple.

Sometimes no solution is best by both metrics.

- See Section 2.1.1 for a simple example.
- Later in our class, we will explore more space/time tradeoffs in design.
- In practice, tradeoffs are commonplace.
- Take a look at Section 2.1.6* for more.

What About Power and Complexity?

These two metrics are beyond our class' scope.
You'll see power in ECE385.

One heuristic for power

- uses the fact that **current flows when a transistor switches on/off**
- and uses simulation to **estimate the number of times** that happens.

Complexity is hard to measure, and
is usually based on experience.