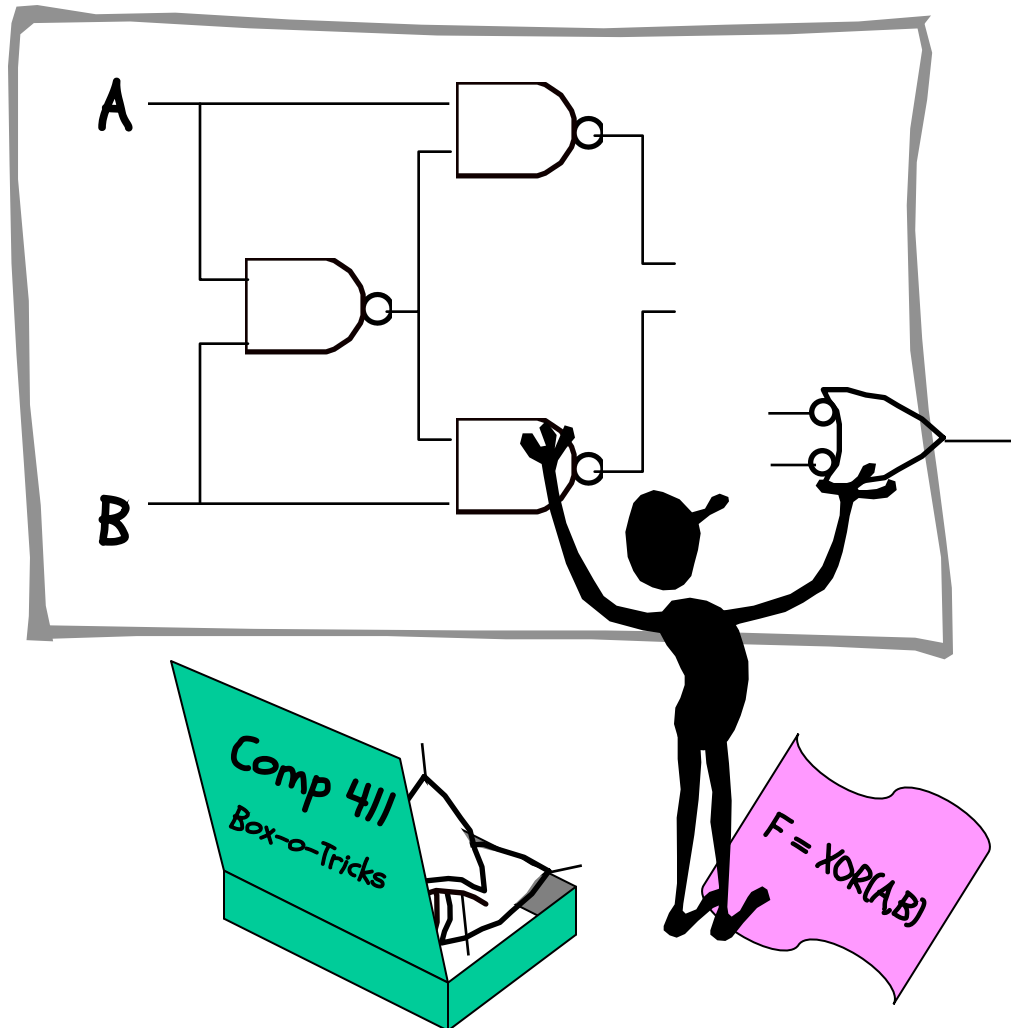


PHYSICAL BITS: TRANSISTORS AND LOGIC



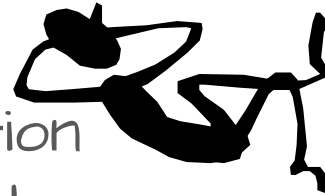
- Encoding bits with voltages
- The "Digital" contract
- Digital processing elements
- Gates
- Transistors
- Building gates with transistors

WHERE ARE WE?



Things we know so far -

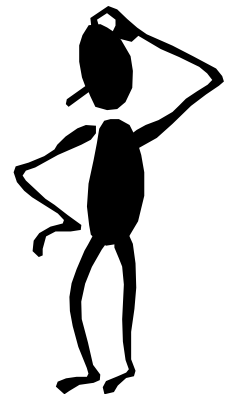
- 1) Computers process information
- 2) Information is measured in bits
- 3) Data can be represented as groups of bits
- 4) Computer instructions are encoded as bits
- 5) Computer instructions are just data
- 6) But, we don't want to deal with details of bits...



so we use ASSEMBLY Language

- 7) Even that is too low-level...

So we use COMPILERS to generate assembly code, and assemblers to generate the final bits ...



But, how are bits PROCESSED?

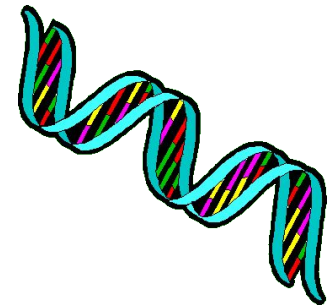
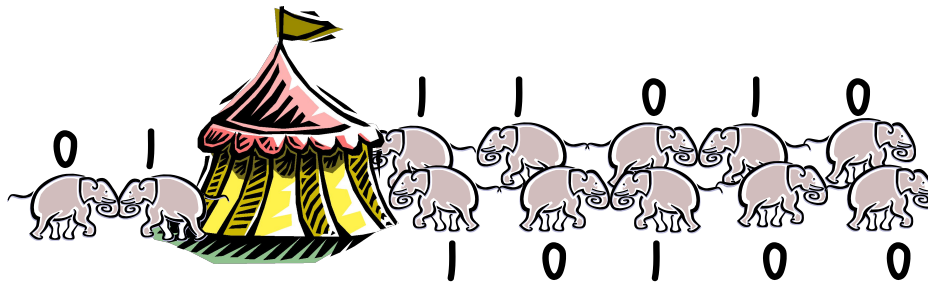
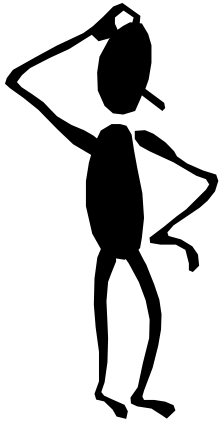
A SUBSTRATE FOR COMPUTATION



We can build devices for processing and representing bits using almost any physical phenomenon

Wait! Some of those
might have potential...

- neutrino flux
- trained elephants
- engraved stone tablets
- orbits of planets
- DNA sequences
- polarization of a photon



USING ELECTROMAGNETIC PHENOMENA



Some EM things we could encode bits with:

voltages phase
currents frequency

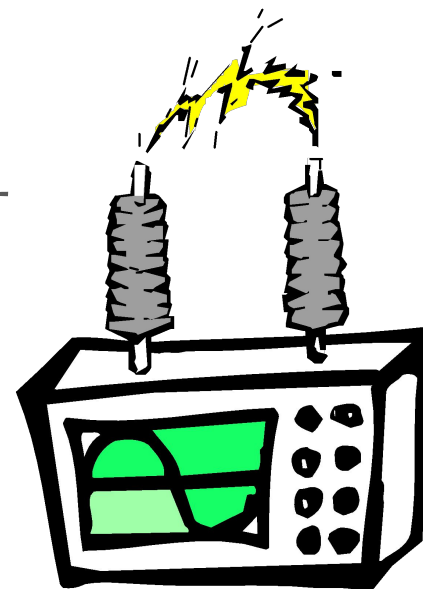
With today's technologies **voltages** are most often used.

Voltage pros:

easy generation, detection
voltage changes can be very fast
lots of engineering knowledge

Voltage cons:

easily affected by environment
DC connectivity required?
R & C effects slow things down



REPRESENTING INFORMATION WITH VOLTAGES



Representation of each point (x, y) in a B&W Picture:

0 volts: BLACK
1 volt: WHITE
0.37 volts: 37% Gray
etc.



Representation of a picture:

Scan points in some prescribed
raster order... generate voltage
waveform

How much information
at each point?

INFORMATION PROCESSING = COMPUTATION



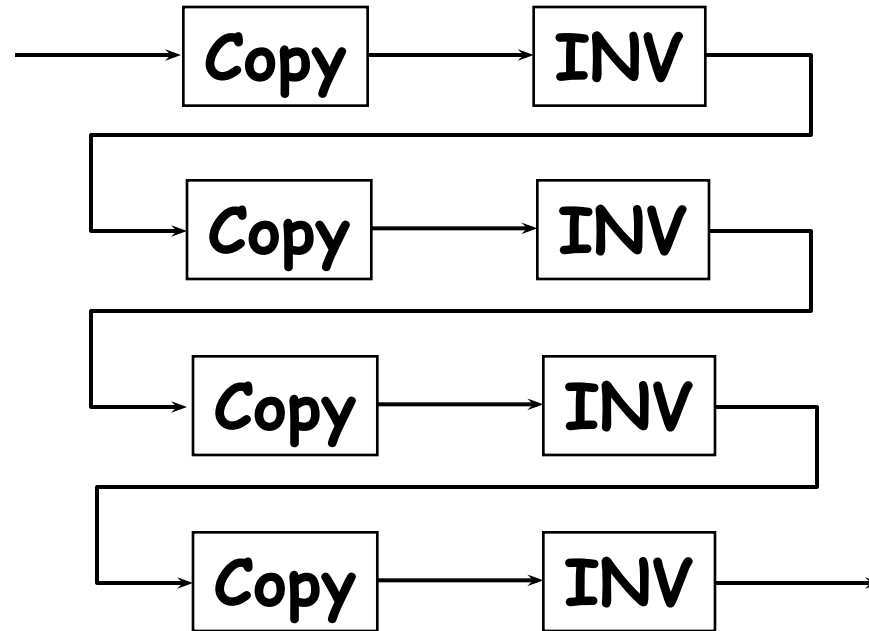
First, let's consider some processing blocks:



LET'S BUILD A SYSTEM!



input



(Reality)



output

WHY DID OUR SYSTEM FAIL?



Why doesn't reality match theory?

1. COPY Operator doesn't work right
2. INVERSION operator doesn't work right
3. Theory is imperfect
4. Reality is imperfect
5. Our system architecture stinks



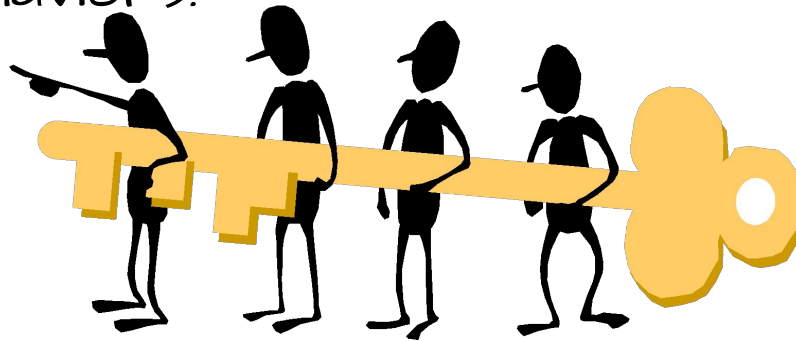
ANSWER: all of the above!

Noise and inaccuracy are inevitable; we can't reliably reproduce infinite information-- we must **design our system to tolerate some amount of error** if it is to process information reliably.

THE KEY TO SYSTEM DESIGN



A *SYSTEM* is a structure that is "guaranteed" to exhibit a specified behavior, assuming all of its components obey their specified behaviors.



How is this achieved? **Through Contracts**

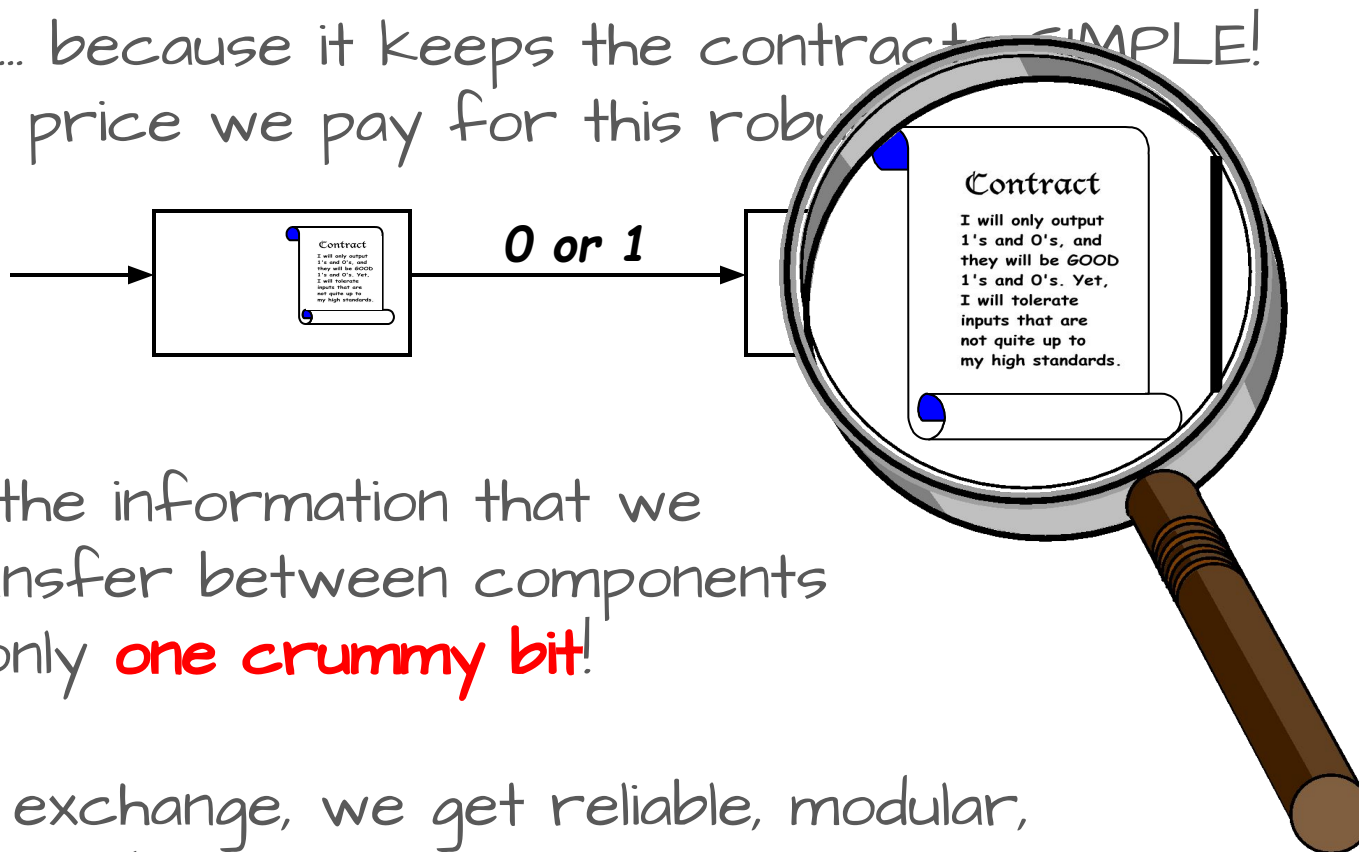
Every system component will have clear obligations and responsibilities. If these are maintained we have every right to expect the system to behave as planned. If contracts are violated all bets are off.

DIGITAL CONTRACTS



Why DIGITAL?

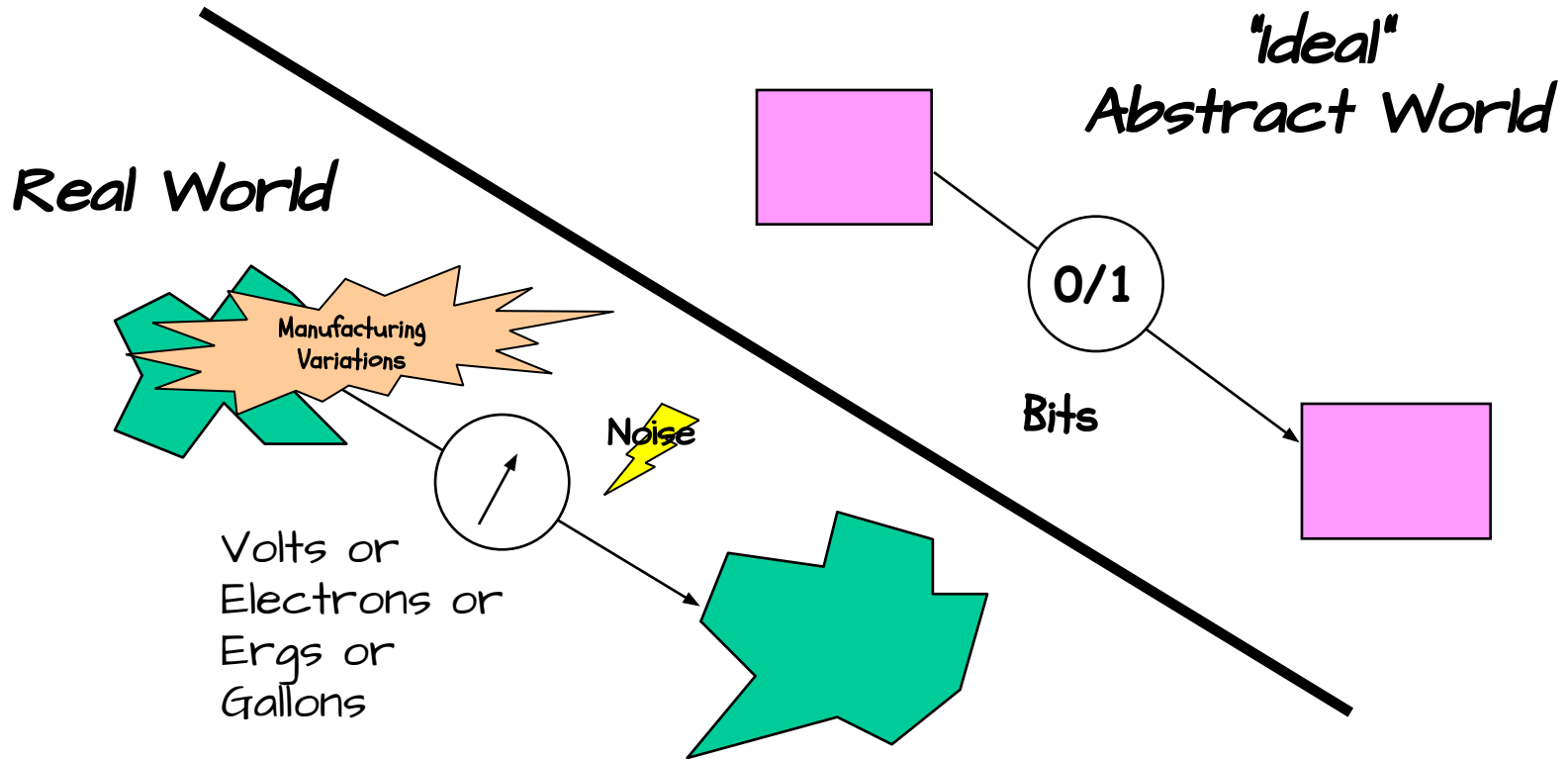
... because it keeps the contract SIMPLE!
It's the price we pay for this robustness.



All the information that we transfer between components is only **one crummy bit!**

But, in exchange, we get reliable, modular, and reproducible systems.

THE DIGITAL ABSTRACTION



Keep in mind, **the world is not digital, we engineer it to behave that way.** We coerce real physical phenomena to implement digital designs!

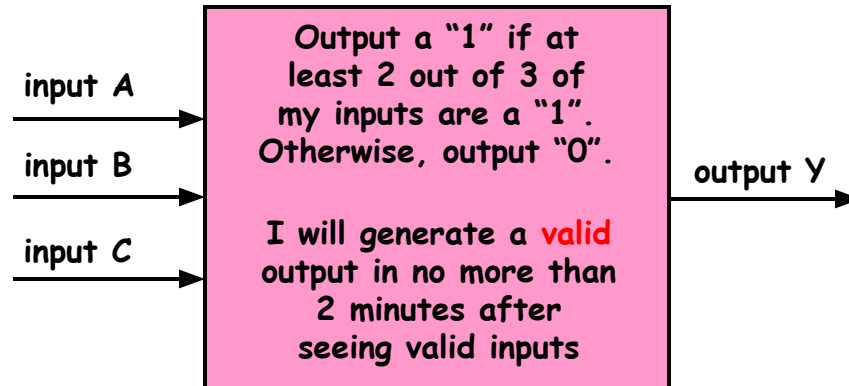
A DIGITAL PROCESSING ELEMENT



- A **combinational device** is a digital element that has

Static
Discipline

- one or more digital inputs
- one or more digital outputs
- a functional specification that details the value of each output for **every possible combination of valid input values**
- a timing specification consisting (at a minimum) an upper bound propagation delay, t_{pd} , on the required time for the device to compute the specified **valid** output values from an arbitrary set of stable, **valid** input values



A COMBINATIONAL DIGITAL SYSTEM



A system of interconnected elements is combinational if

- each circuit element is combinational
- every input is connected to exactly one output or directly to some source of 0's or 1's
- the circuit contains no directed cycles

No feedback (yet!)



But, in order to realize digital processing elements we have one more requirement!

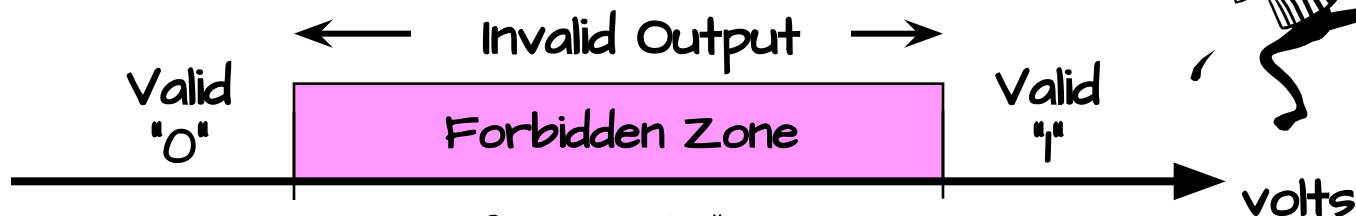
A definition for a VALID input
and a VALID output!



VALID = NOISE MARGINS

- Key idea:
Don't allow "0" to be mistaken for a "1" or vice versa
- Use the same "uniform bit-representation convention", for every component in our digital system
- To implement devices with high reliability, we outlaw "close calls" via a representation convention which forbids a range of voltages between "0" and "1".
- Ensure the valid input range is more tolerant (larger) than the valid output range

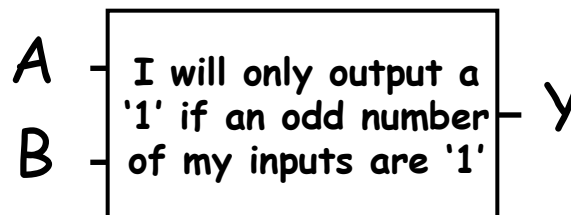
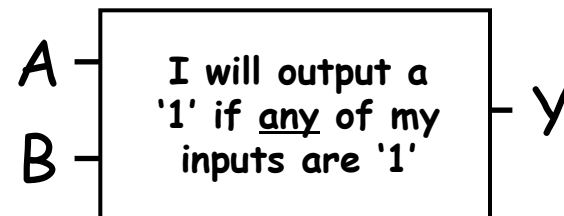
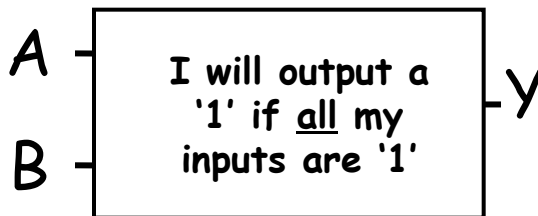
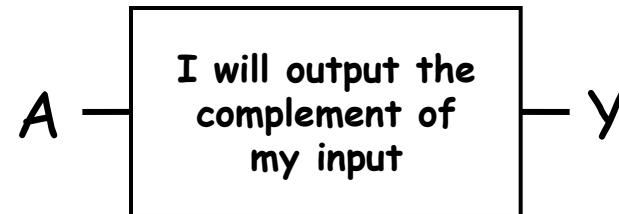
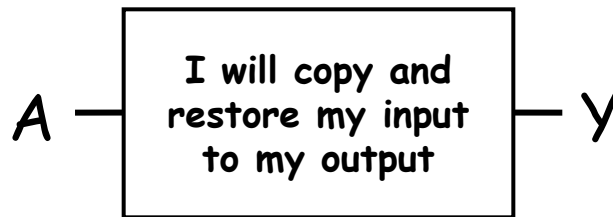
Our definition of valid does not preclude inputs and outputs from passing through invalid values. In fact, they must, but only during transitions. Our specifications allow for this (i.e. outputs are specified sometime (T_{pd}) after inputs become valid).



DIGITAL PROCESSING ELEMENTS



Some digital processing elements occur so frequently that we give them special names and symbols



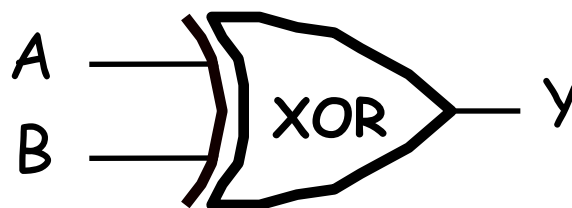
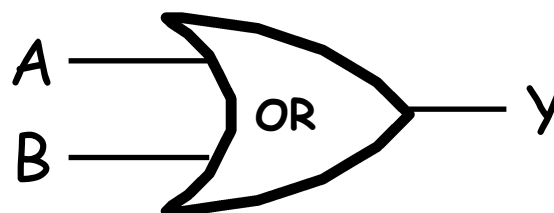
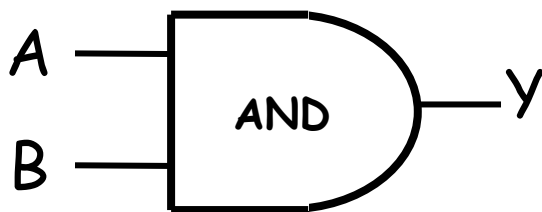
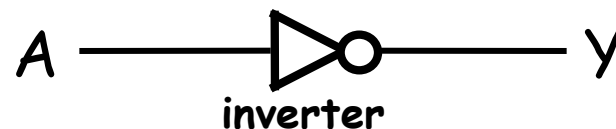
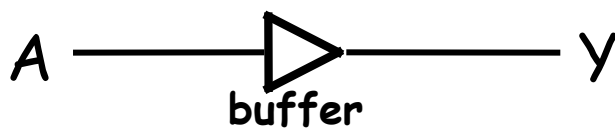
Q: What is the point of a buffer?
Doesn't a wire do the same thing?
A: A buffer restores marginal digital signals, because the output is as good or "better" than the input (i.e. it solves that bad image problem from slide 7).



DIGITAL PROCESSING ELEMENTS



Some digital processing elements occur so frequently that we give them special names and symbols



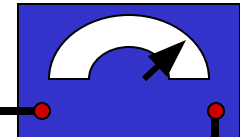
In honor of the richest man in the world we will henceforth refer to digital processing elements as "GATES"



HOW DO WE MAKE GATES?

- A controllable switch is the common link of all computing technologies
- How do you control voltages with a switch?
- By creating and opening paths between higher and lower potentials

This symbol indicates a "high" potential, or the voltage of the power supply



This symbol indicates a "low" or ground potential

N-CHANNEL FIELD-EFFECT TRANSISTORS (NFETs)



Operating regions:

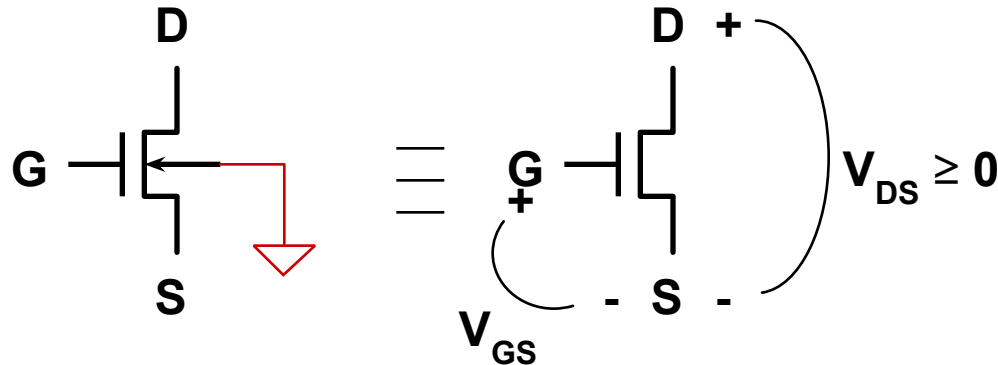
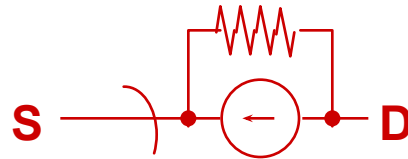
cut-off:
 $V_{GS} < V_{TH}$ 0.8V



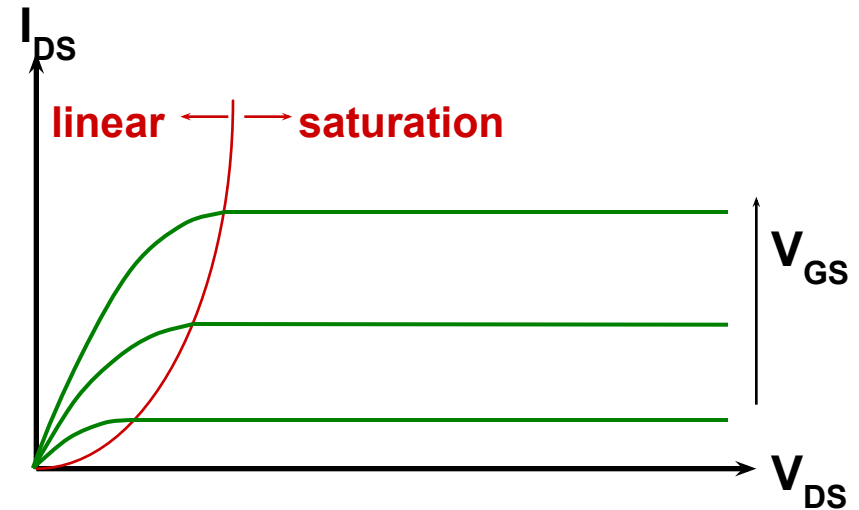
linear:
 $V_{GS} \geq V_{TH}$
 $V_{DS} < V_{Dsat}$



saturation:
 $V_{GS} \geq V_{TH}$
 $V_{DS} \geq V_{Dsat}$



When the gate voltage is "high", the switch closes. Good at pulling things "low".



P-CHANNEL FIELD-EFFECT TRANSISTORS (PFETs)



Operating regions:

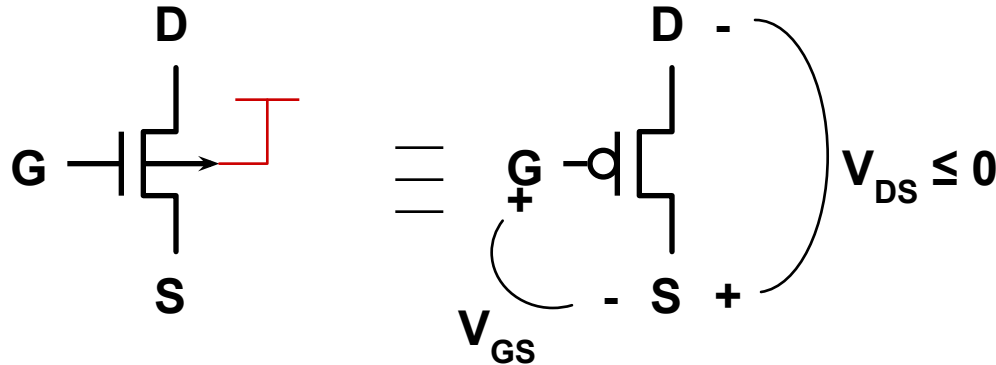
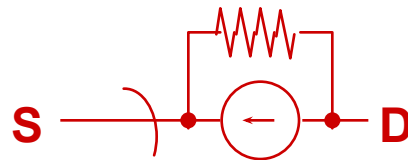
cut-off:
 $V_{GS} > V_{TH} - 0.8V$



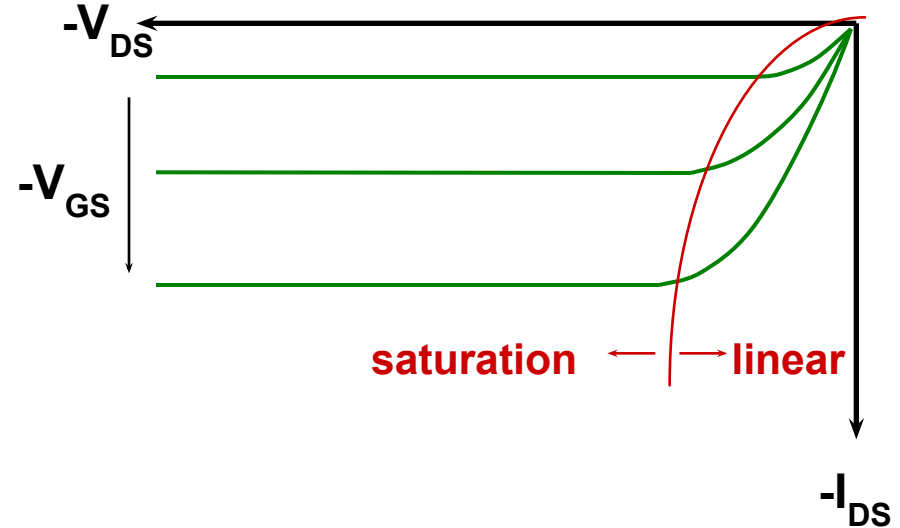
linear:
 $V_{GS} \leq V_{TH}$
 $V_{DS} > V_{Dsat}$



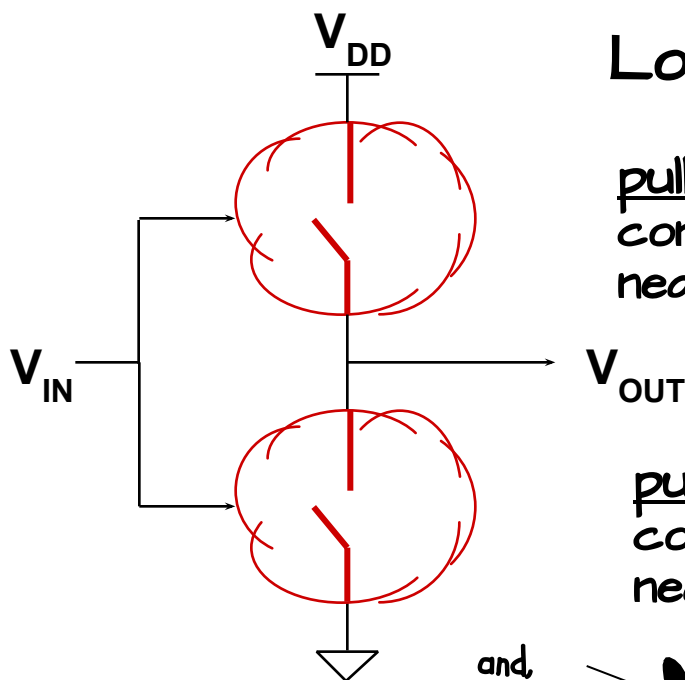
saturation:
 $V_{GS} \leq V_{TH}$
 $V_{DS} \leq V_{Dsat}$



When the gate voltage is "low", the switch closes. Good at pulling things "high".



USING TRANSISTORS TO BUILD LOGIC GATES!



Logic Gate recipe:

pullup: make this connection when V_{IN} is near 0 so that $V_{OUT} = V_{DD}$

We use PFETs here

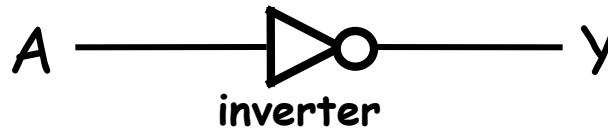
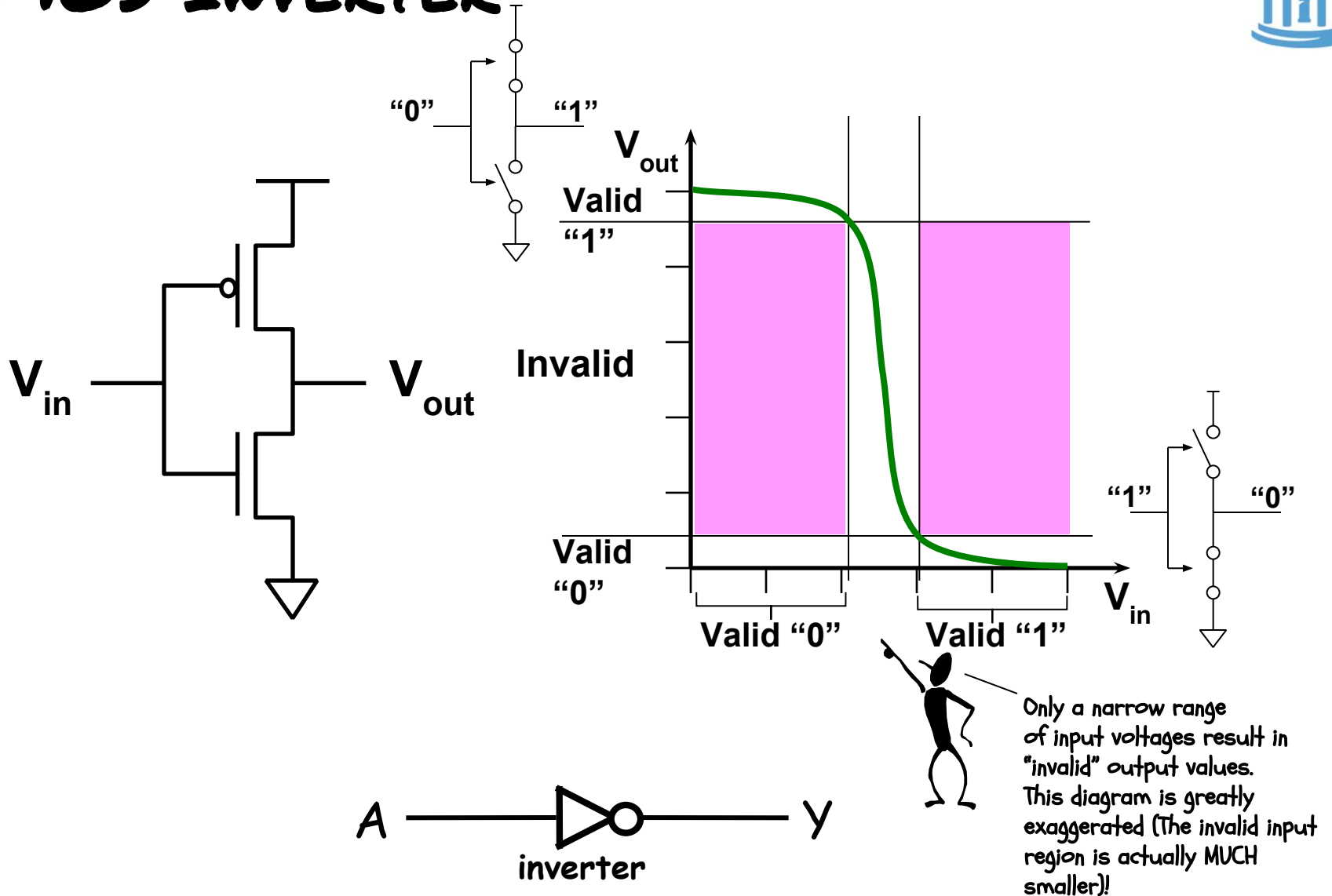


pulldown: make this connection when V_{IN} is near V_{DD} so that $V_{OUT} = 0$

and, NFETs here



CMOS INVERTER

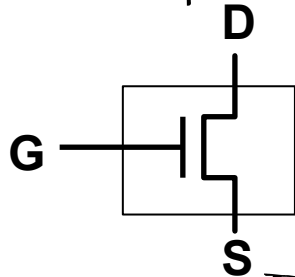


"DIGITAL" TRANSISTOR ABSTRACTION

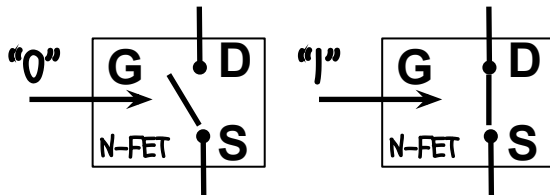


- Transistors are extremely flexible, but fickle analog devices.
- If we limit how we use them, (i.e. adopt the following conventions), they can act as robust digital devices.
- Which we can treat as a simple switch abstraction.

N-channel FET,
a 3-input device

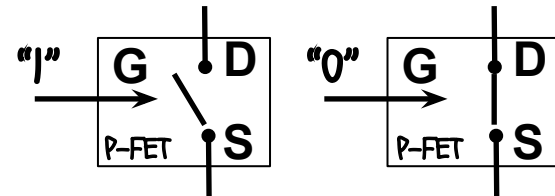
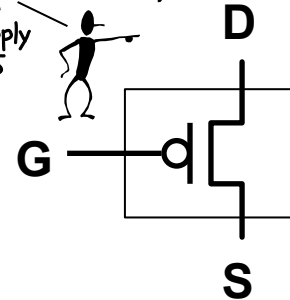


Convention: The S terminal of an N-FET
will be connected to either ground or
the D terminal of another N-FET



Convention: The D terminal of a P-FET
will be connected to either the supply
(the voltage representing "1") or the S
terminal of another P-FET

P-channel FET,
a 3-input device



COMPLEMENTARY PULLUPS AND PULLDOWNS



This is what the "C"
in CMOS stands for!

We design components with **complementary** pullup and pulldown logic (i.e., the pulldown should be "on" when the pullup is "off" and vice versa).

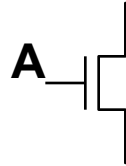
pullup	pulldown	$F(I_1, \dots, I_n)$
on	off	driven "1"
off	on	driven "0"
on	on	driven "X"
off	off	no connection

Convention: In general, let's avoid these last two cases.

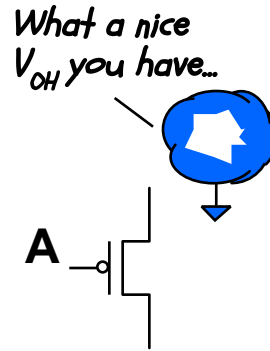
When they are used, the resulting device is not STRICTLY following our STATIC DISCIPLINE (eg. Pass gates and storage devices).

Such devices are only QUASI-DIGITAL!

CMOS COMPLEMENTS

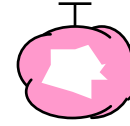


On when A is "1"



On when A is "0"

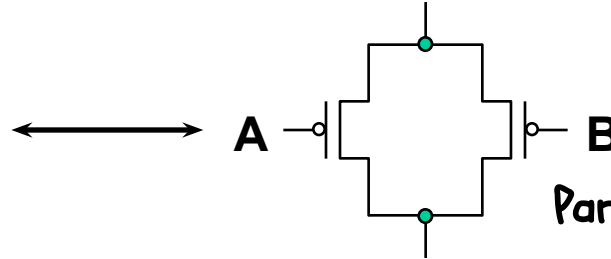
What a nice V_{OH} you have...



Thanks. It runs in the family...

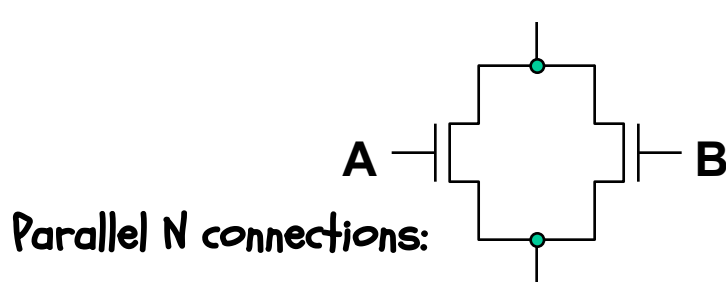


On when A is "1" and B is "1": $A \cdot B$

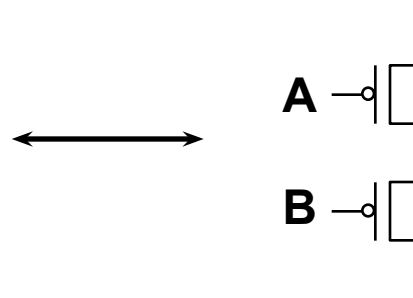


Parallel P connections:

On when A is "0" or B is "0": $\overline{A} + \overline{B}$



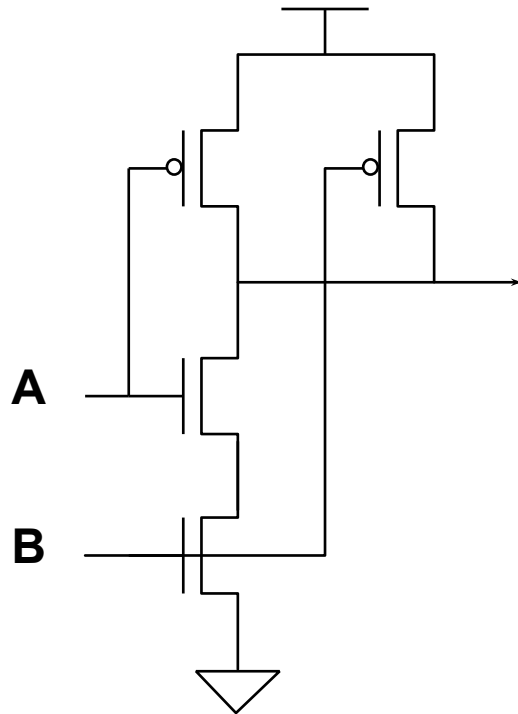
On when A is "1" or B is "1": $A + B$



Series P connections:

On when A is "0" and B is "0": $\overline{A} \cdot \overline{B}$

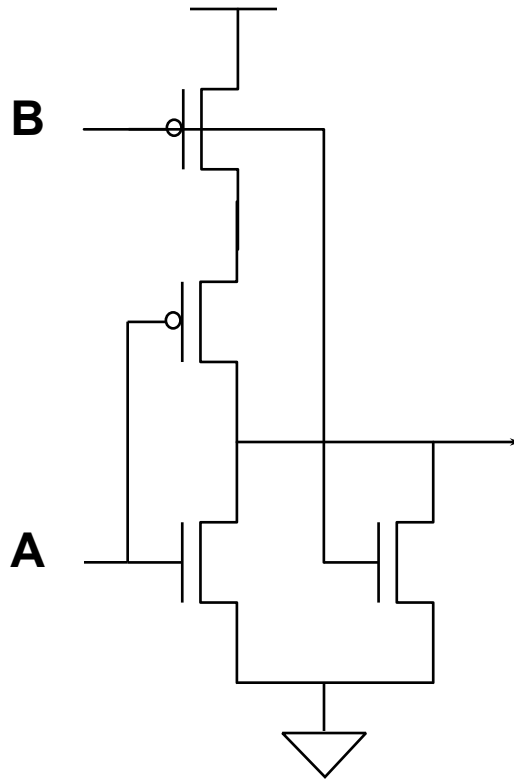
A TWO-INPUT LOGIC GATE



What function does this gate compute?

A	B	C
0	0	
0	1	
1	0	
1	1	

HERE'S ANOTHER...



What function does this gate compute?

A	B	C
0	0	
0	1	
1	0	
1	1	

GENERAL CMOS GATE RECIPE

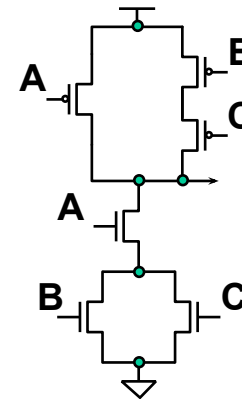
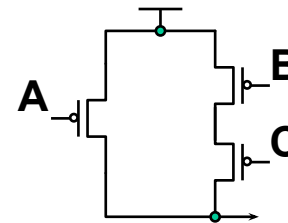
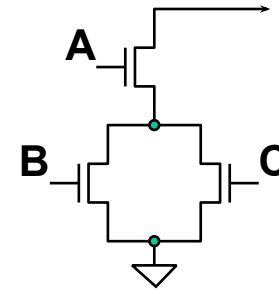


Step 1. Figure out pulldown network that does what you want (i.e the set of conditions where the output is '0')

$$\text{e.g., } F = \overline{A \cdot (B + C)}$$

Step 2. Walk the hierarchy replacing nfets with pfets, series subnets with parallel subnets, and parallel subnets with series subnets

Step 3. Combine pfet pullup network from Step 2 with nfet pulldown network from Step 1 to form fully-complementary CMOS gate.



But isn't it hard to wire it all up?



ONE LAST EXERCISE



Let's construct a gate to compute:

$$F = \overline{A+BC} = \text{NOT}(\text{OR}(A, \text{AND}(B, C)))$$

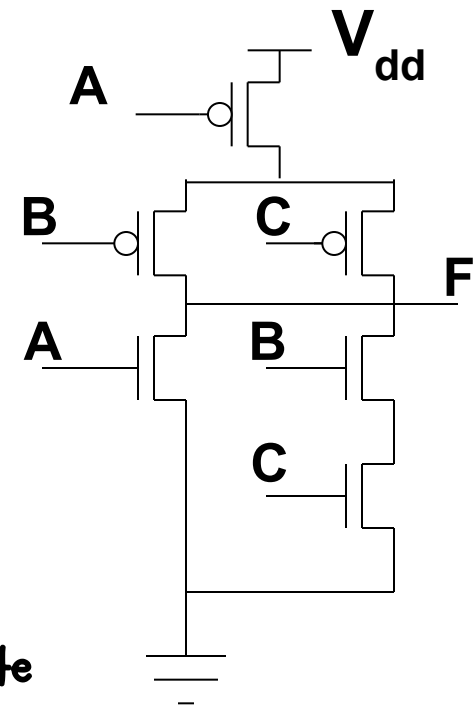
A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Step 1: The pull-down network

Step 2: The complementary pull-up network



OBSERVATION: CMOS gates tend to be inverting! Precisely, one or more "0" inputs are necessary to generate a "1" output, and one or more "1" inputs are necessary to generate a "0" output. Why?



NEXT TIME



Now that we can see what goes on inside of a single gate, we'll next use several them to compose larger systems that compute other logic functions.

