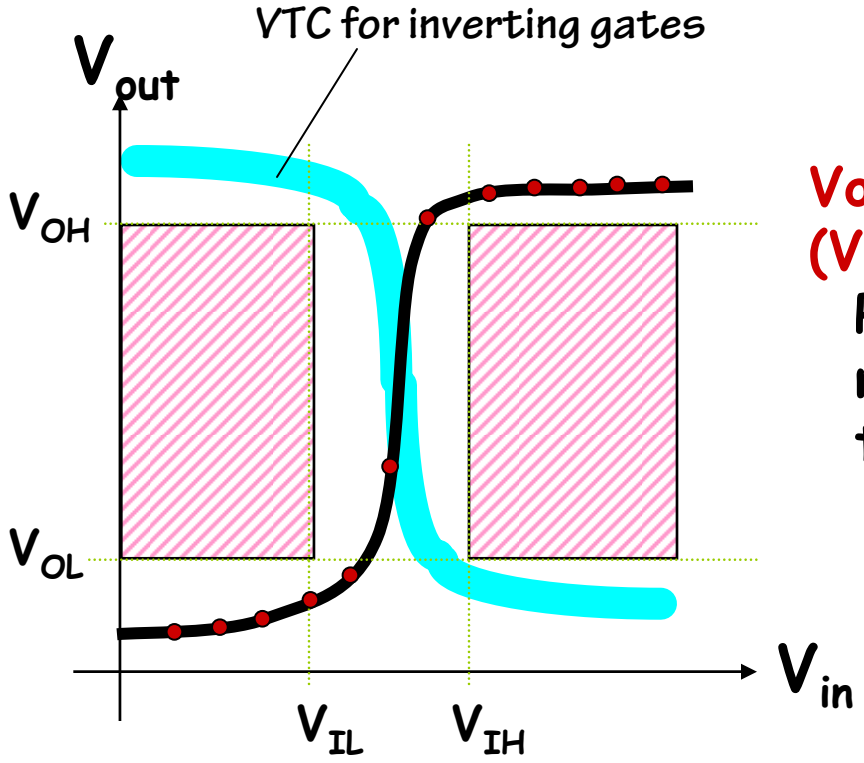
Example device: A Buffer



Voltage Transfer Characteristic (VTC):

Plot of V_{out} vs. V_{in} where each measurement is taken after any transients have died out.

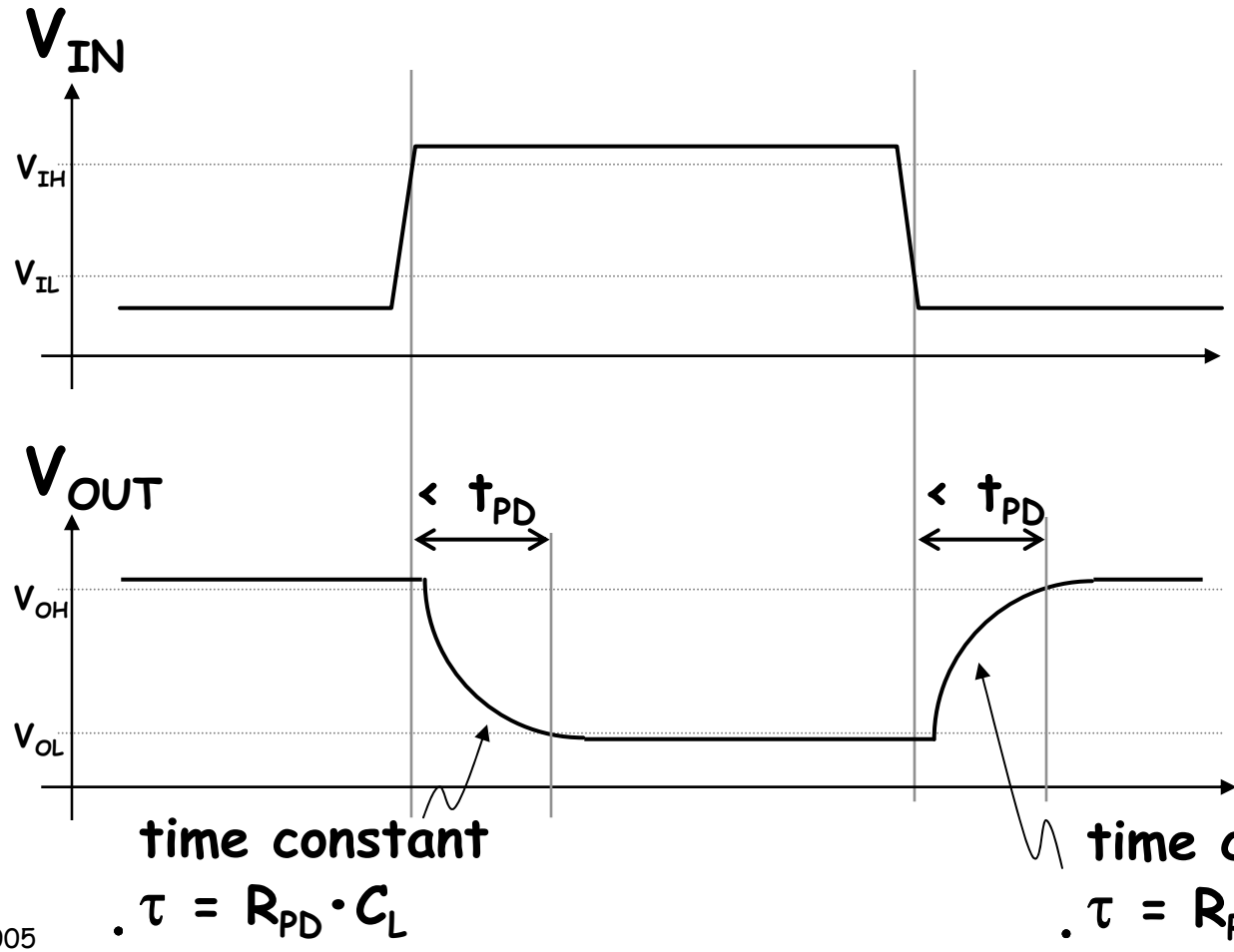
Note: VTC does not tell you anything about how fast a device is—it measures static behavior not dynamic behavior

Static Discipline requires that we avoid the shaded regions aka "forbidden zones"), which correspond to *valid* inputs but *invalid* outputs. Net result: combinational devices must have GAIN > 1 and be NONLINEAR.

Due to unavoidable delays…

Propagation delay (t_{PD}):

An UPPER BOUND on the delay from valid inputs to valid outputs.



GOAL:

minimize propagation delay!

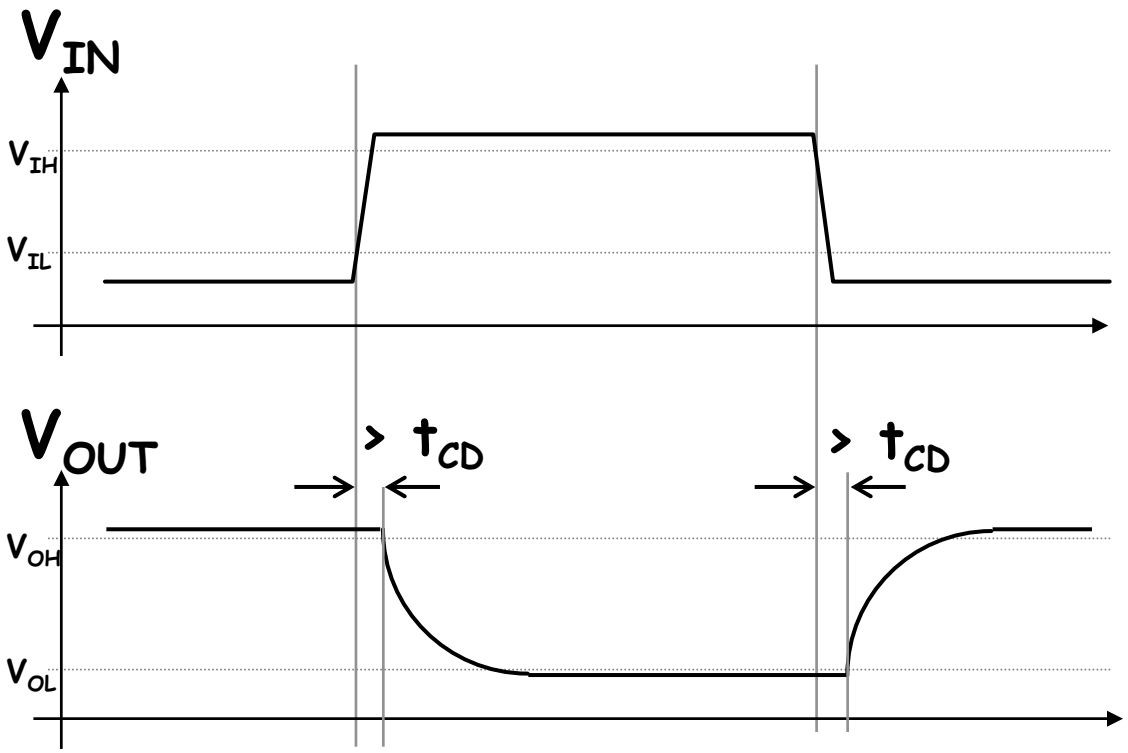
ISSUE:

keep Capacitances low and transistors fast

Contamination Delay

an optional, additional timing spec

INVALID inputs take time to propagate, too...



Do we really need t_{CD}?

Usually not… it'll be important when we design circuits with registers (coming soon!)

If t_{CD} is not specified, safe to assume it's 0.

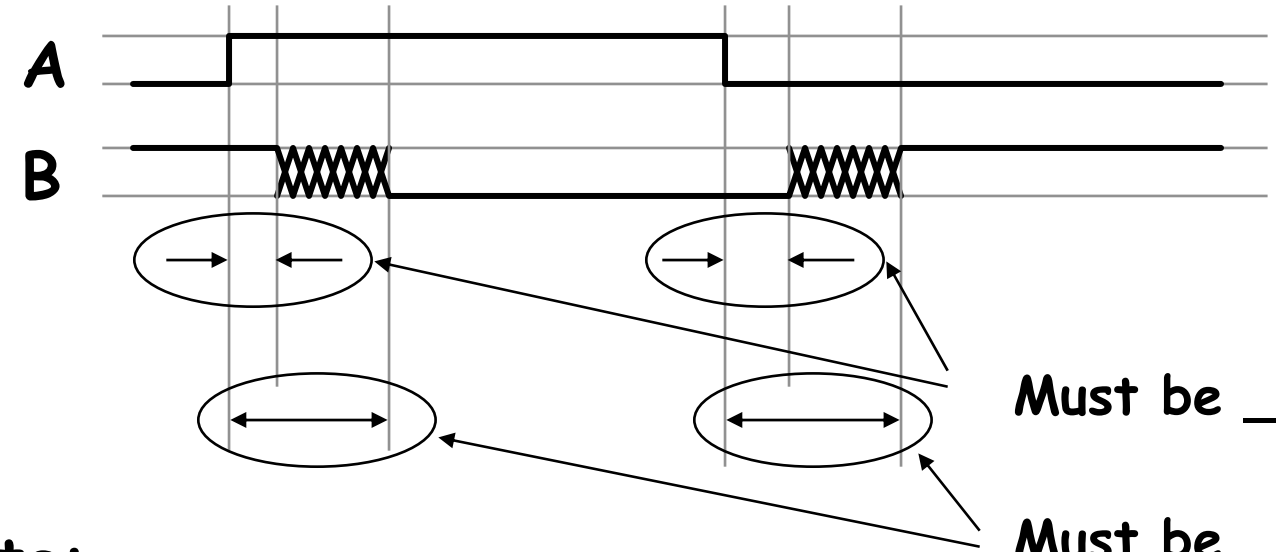
CONTAMINATION DELAY, t_{CD}

A LOWER BOUND on the delay from any invalid input to an invalid output

The Combinational Contract

A	B
0	1
1	0

t_{PD} propagation delay
t_{CD} contamination delay



Must be $> t_{CD}$

Must be $< t_{PD}$

Note:

1. *No Promises* during XXXXX
2. Default (conservative) spec: $t_{CD} = 0$

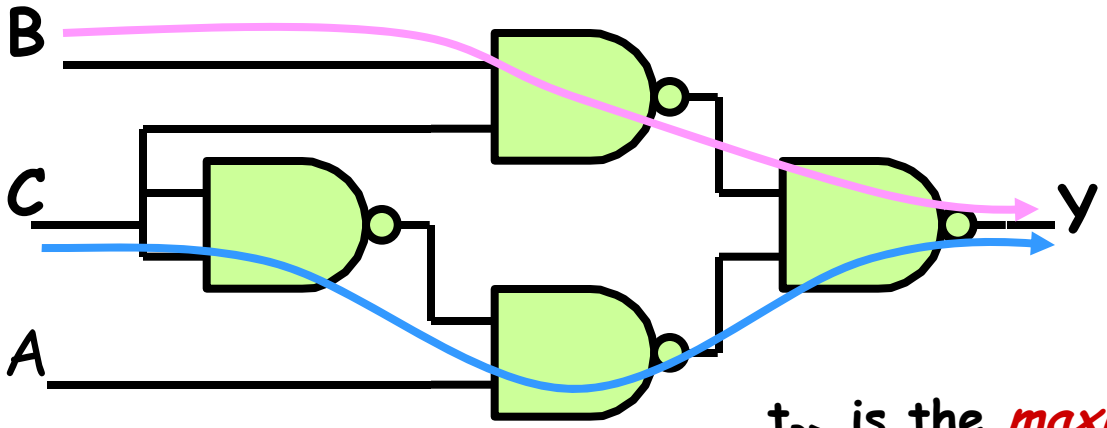
Example: Timing Analysis

If NAND gates have a t_{PD} = 4nS and t_{CD} = 1nS

t_{CD} is the *minimum* cumulative contamination delay over all paths from inputs to outputs

t_{PD} = 12 nS

t_{CD} = 2 nS

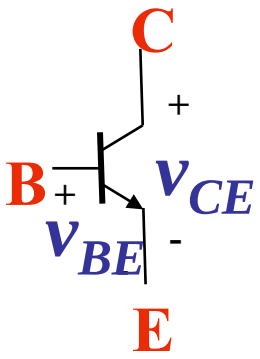


t_{PD} is the *maximum* cumulative propagation delay over all paths from inputs to outputs

The "perfect" logic family

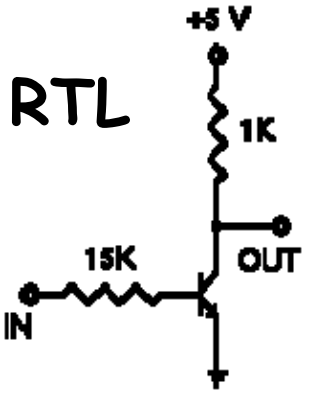
- Good noise margins (want a "step" VTC)
- Implement useful selection of (binary) logic
  - INVERTER, NAND, NOR with modest fan-in (4? Inputs)
  - More complex logic in a single step? (minimize delay)
- Small physical size
  - Shorter signal transmission distances (faster)
  - Cost proportional to size (cheaper)
- Inexpensive to manufacture
  - "print" technology (lithographic masks, deposition, etching)
  - Large-scale integration
- Minimal power consumption
  - Portable
  - Massive processing without meltdown

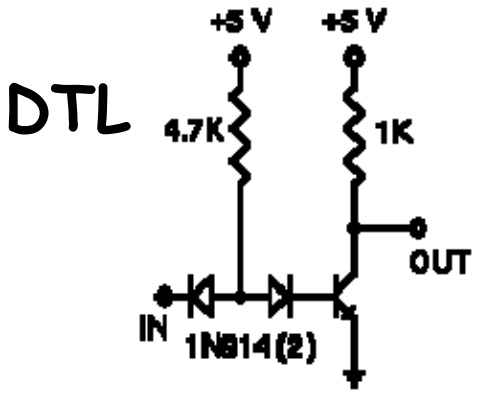
Transitor-transitor Logic (TTL)

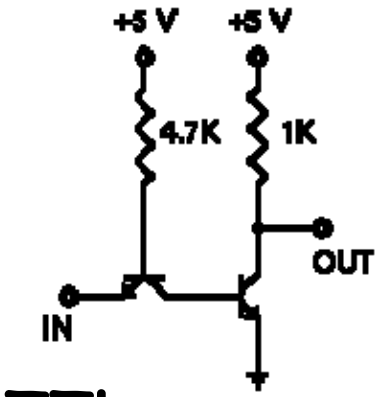


NPN BJT

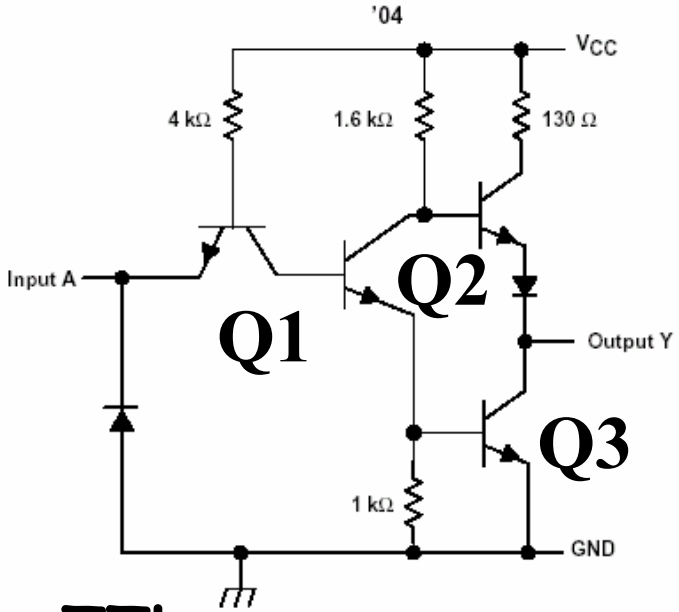
$I_{CE} = \beta I_{BE}$



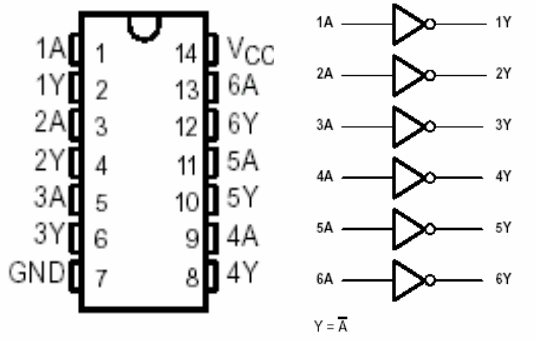




TTL



TTL w/ totem pole outputs
("on" threshold = 2 diode drops)



74LS04
(courtesy TI)

TTL Signaling

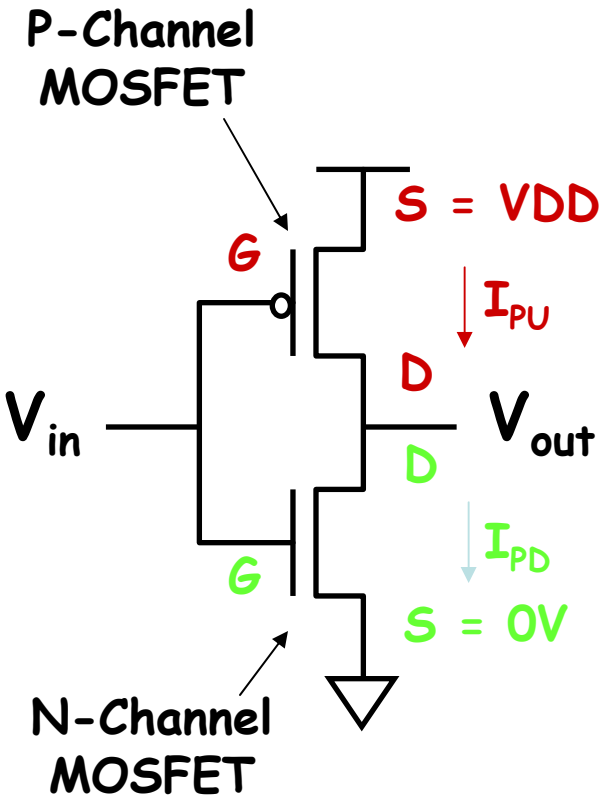
- Typical TTL signaling spec
  - I_{OL} = 16mA, I_{OH} = -0.4mA (V_{OL}=0.4V, V_{OH}=2.7V, V_{CC}=5V)
  - I_{IL} = -1.6mA, I_{IH} = 0.04mA (V_{IL}=0.8V, V_{IH}=2.0V)
  - Switching threshold = 1.3V

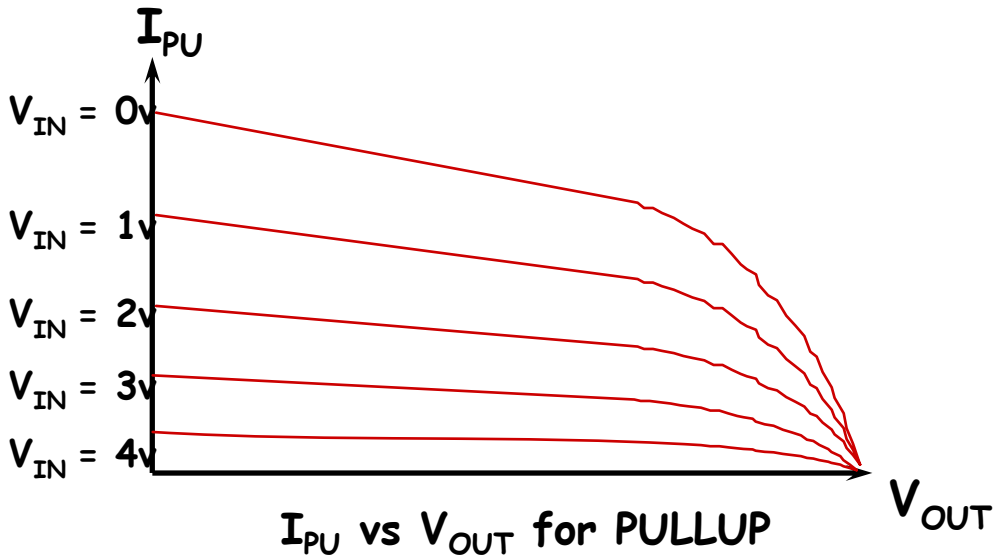
- Each input requires current flow (I_{IL},I_{IH}) and each output can only source/sink a certain amount of current (I_{OL},I_{OH}), so

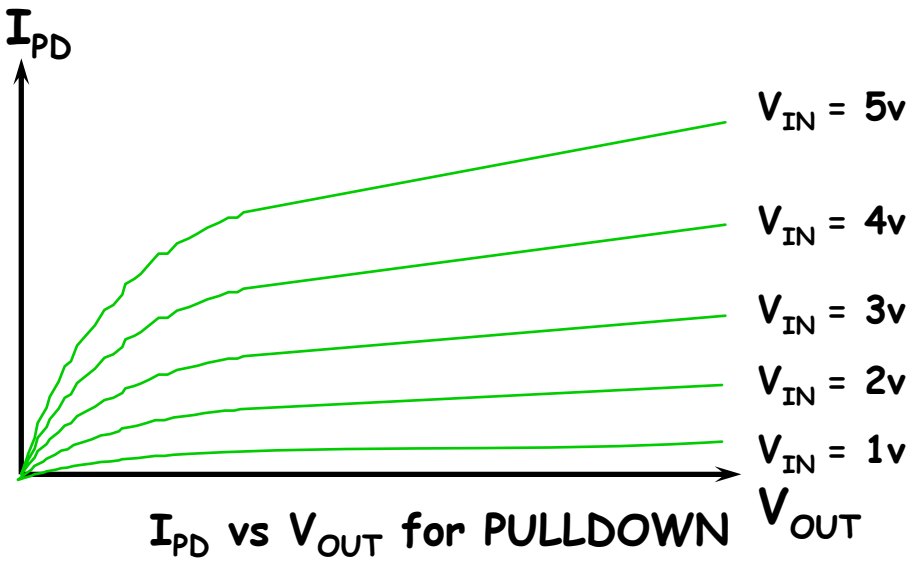
  Max number of inputs that can be driven by a single output is $\min(-I_{IL}/I_{OL}, -I_{IH}/I_{OH}) \approx 10$.

- Current-based logic $\rightarrow$ power dissipation even in steady state, limitations on fanout

Complementary MOS Logic



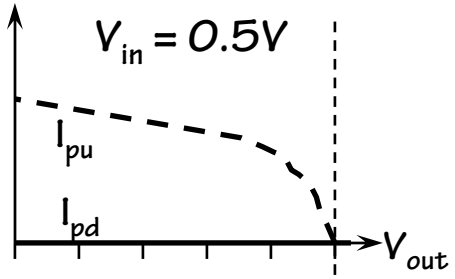


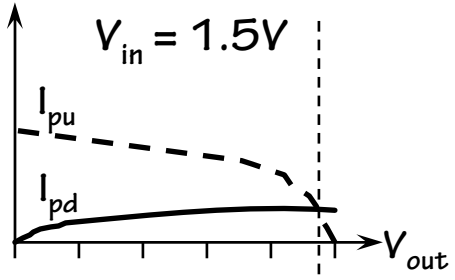


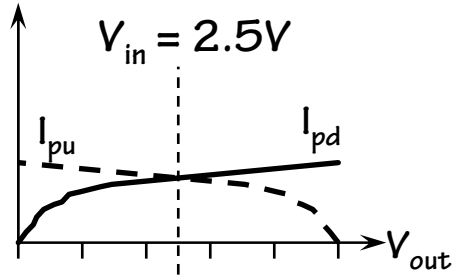
MOSFET:

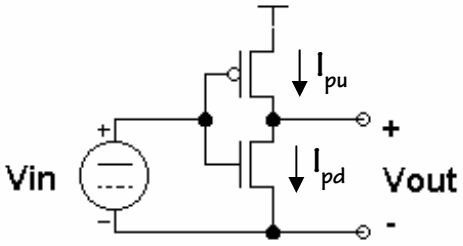
$I_{DS} = f(V_{GS}, V_{DS})$

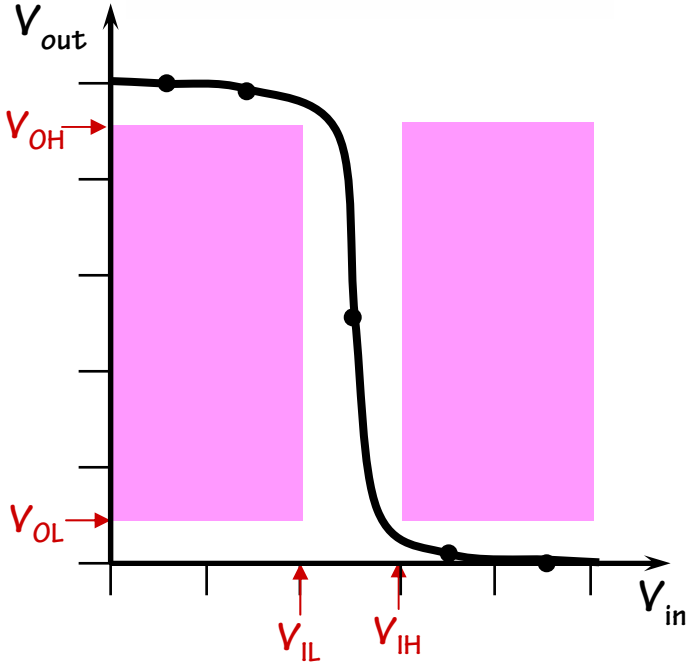
CMOS Inverter VTC



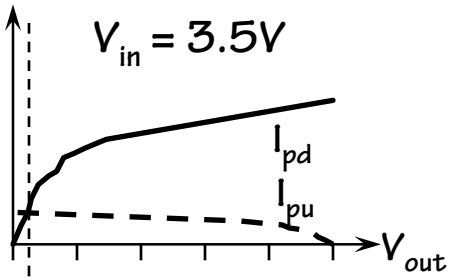


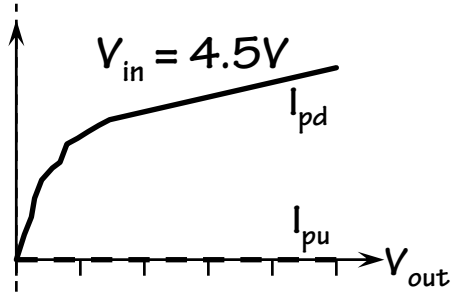






Steady state reached when V_{out} reaches value where $I_{pu} = I_{pd}$.





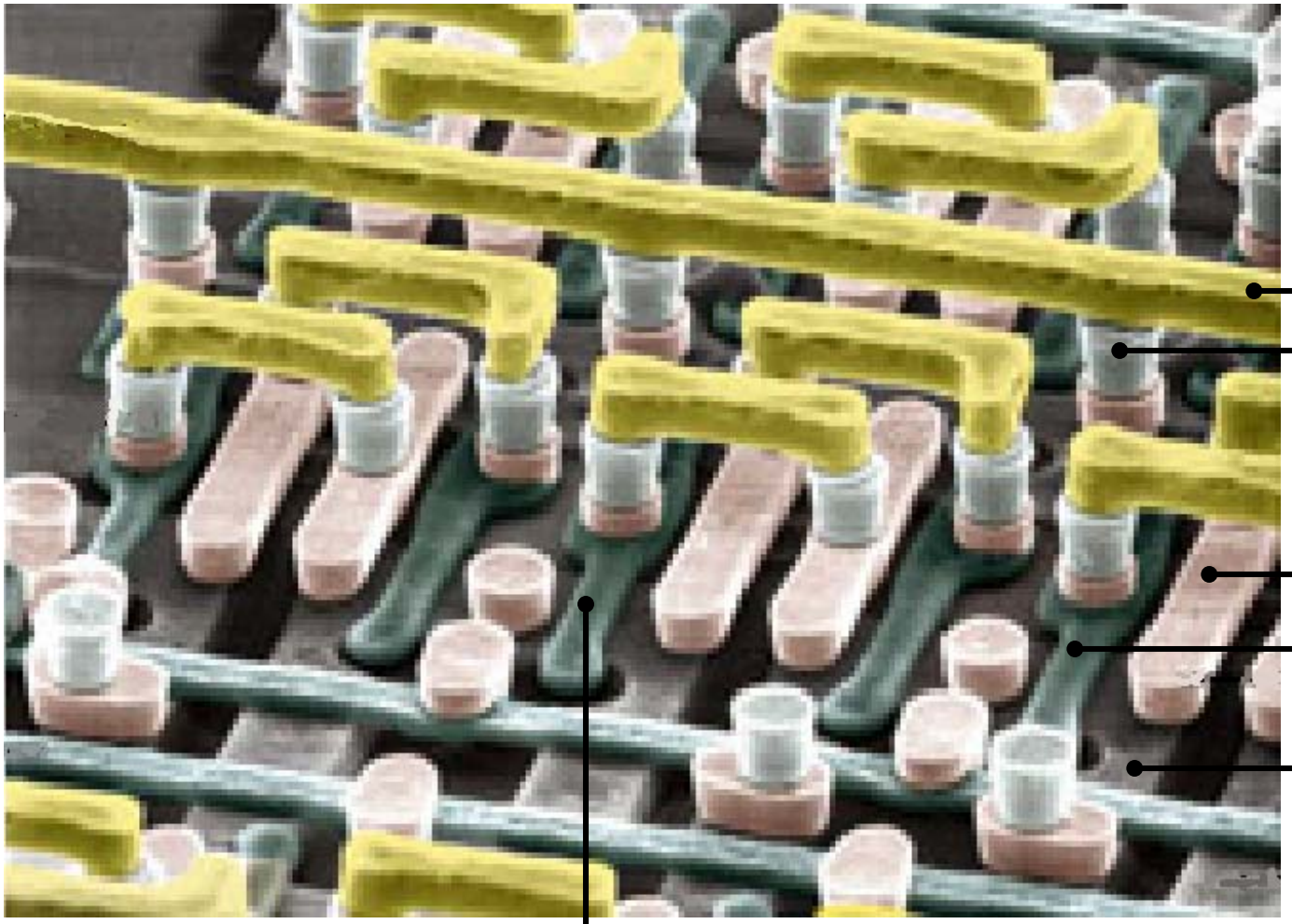
When both fets are saturated, small changes in V_{in} produce large changes in V_{out}

CMOS Signaling

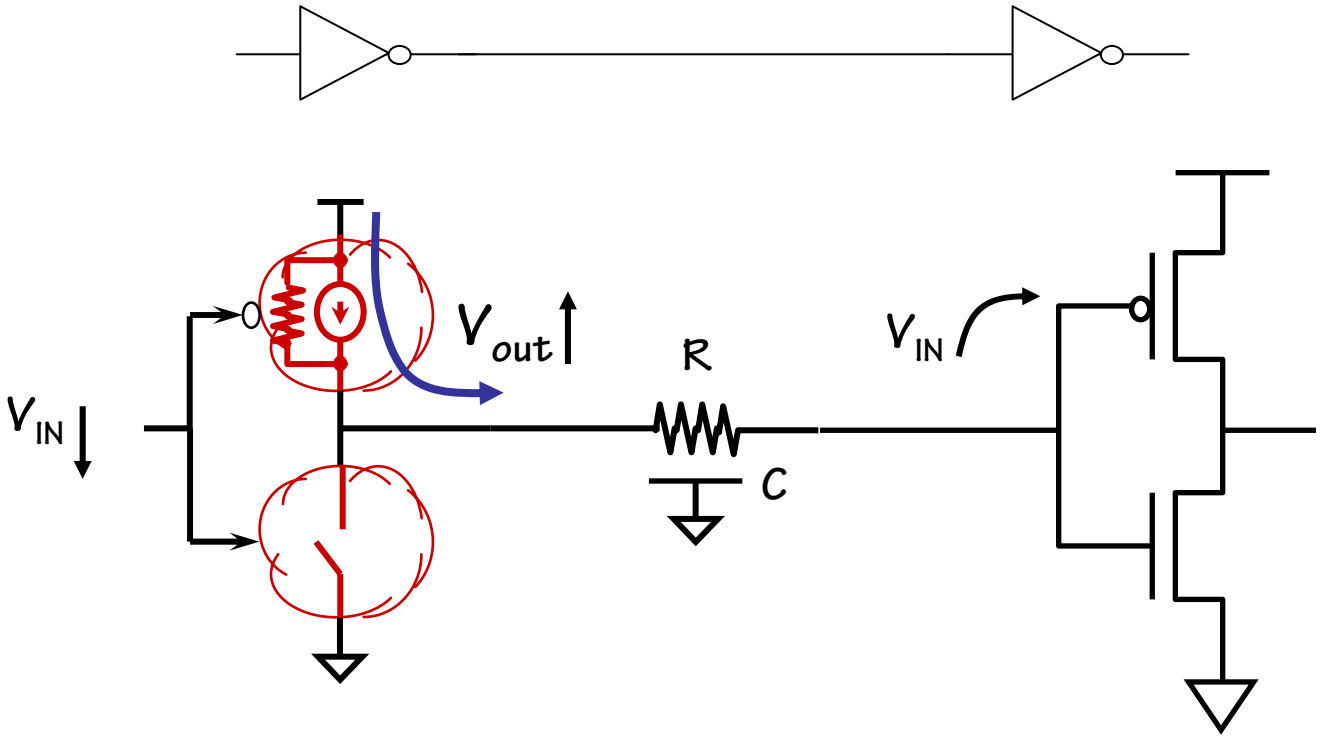
- Typical CMOS signaling specifications:
  - $V_{OL} \approx 0, V_{OH} \approx V_{DD}$ (V_{DD} is the power supply voltage)
  - V_{IL} ≈ just under $V_{DD}/2, V_{IH}$ ≈ just over $V_{DD}/2$
  - Great noise margins! ~$V_{DD}/2$
- Inputs electrically isolated from outputs:
  - An output can drive many, many inputs without violating signaling spec (but transitions will get slower)
- In the steady state, signals are either "0" or "1"
  - When V_{OUT} = 0V, I_{PD} = 0 (and I_{PU} = 0 since pullup is off)
  - When $V_{OUT} = V_{DD}$, I_{PU} = 0 (and I_{PD} = 0 since pulldown is off)
  - No power dissipated in steady state!
  - Power dissipated only when signals change (ie, power proportional to operating frequency).

Multiple interconnect layers

IBM photomicrograph (SiO_2 has been removed!)

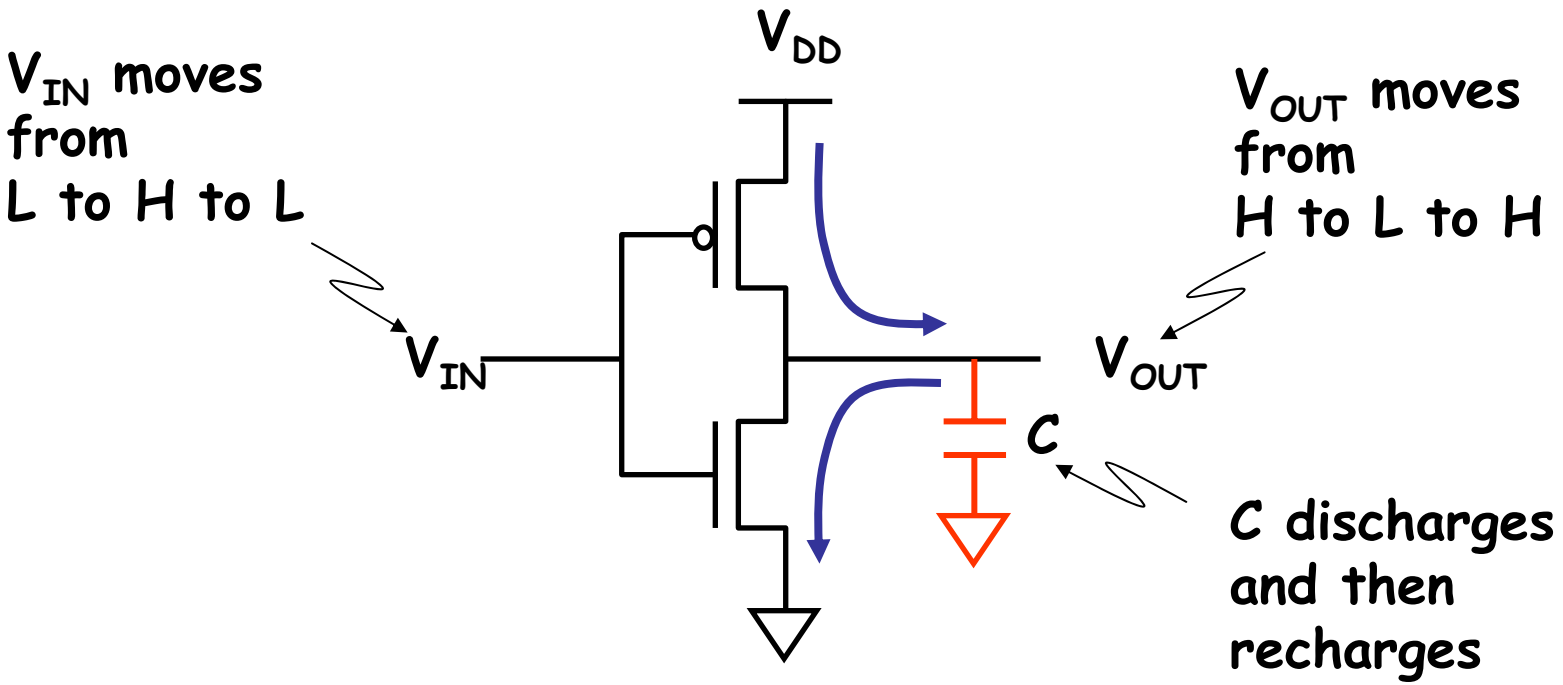


Big Issue 1: Wires



- **Today (i.e., 100nm):**
  - $\tau_{RC} \approx 50$ps/mm
  - Implies > 1 ns to traverse a 20mm x 20mm chip
  - This is a long time in a 2GHz processor

Big Issue 2: Power



- Energy dissipated = $C\ V_{DD}^2$ per gate
  Power consumed = $f\ n\ C\ V_{DD}^2$ per chip

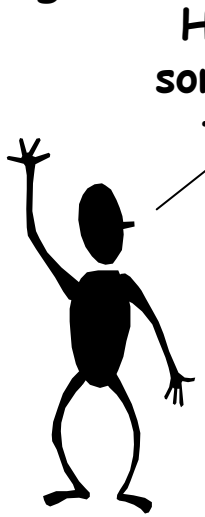
where f = frequency of charge/discharge
n = number of gates /chip

Unfortunately…

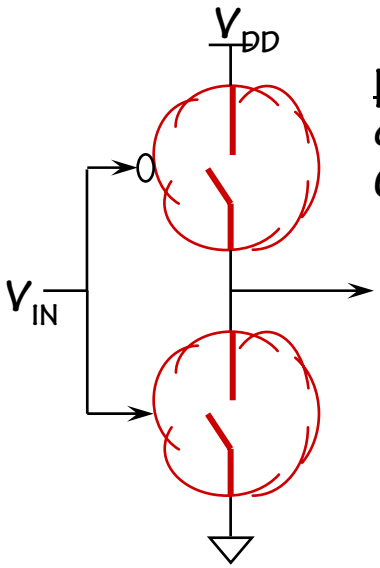
32 Amps (@220v)

- Modern chips (UltraSparc III, Power4, Itanium 2) dissipate from 80W to 150W with a Vdd ≈ 1.2V
  (Power supply current is ≈ 100 Amps)
- Cooling challenge is like making the filament of a 100W incandescent lamp cool to the touch!
  - Worse yet…
    - Little room left to reduce Vdd
    - nC and f continue to grow

MIT Computation Center and Pizzeria



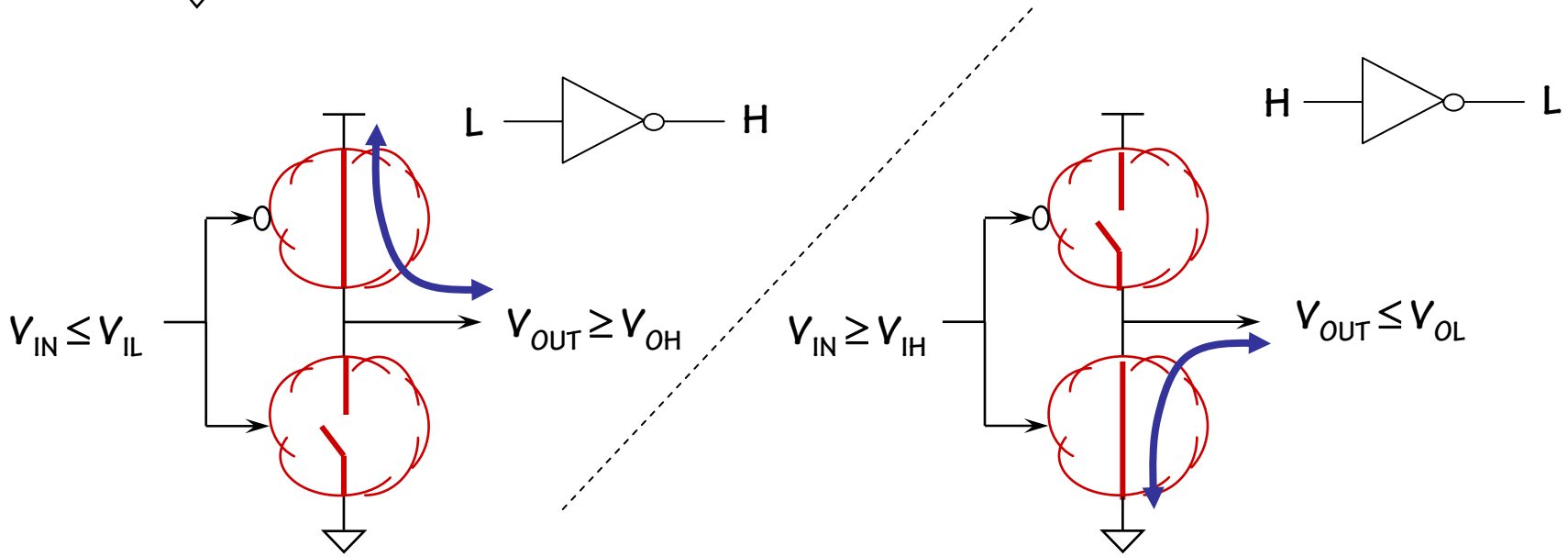
CMOS Gate Recipe: Think Switches



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

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

One power supply →
Two voltages (V_{DD}, GND) →
Binary signaling



Beyond Inverters: Complementary pullups and pulldowns

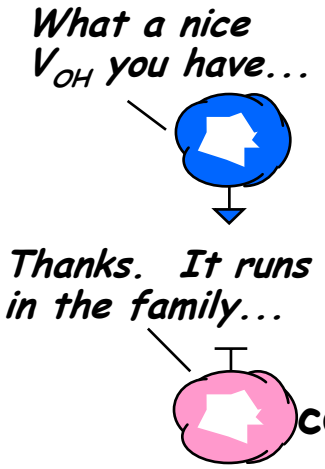
Now you know what the "C" in CMOS stands for!

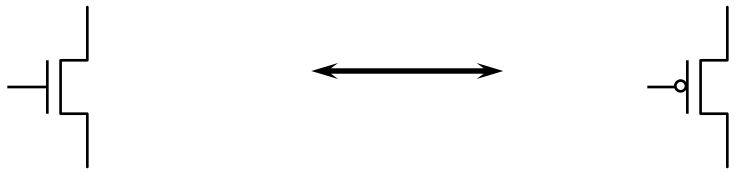
We want *complementary* pullup and pulldown logic, i.e., the pulldown should be "on" when the pullup is "off" and vice versa.

pullup	pulldown	F(A_1,…,An)
on	off	driven "1"
off	on	driven "0"
on	on	driven "X"
off	off	no connection

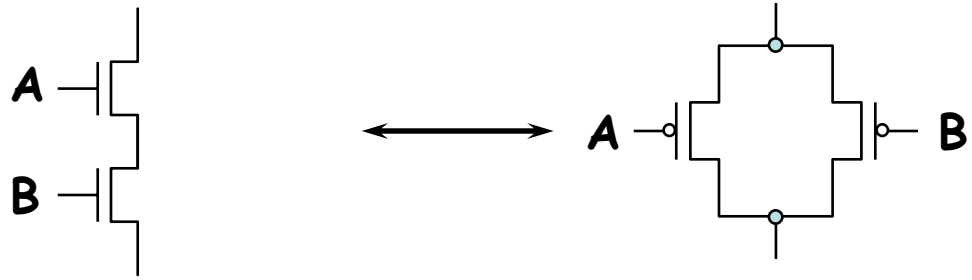
Since there's plenty of capacitance on the output node, when the output becomes disconnected it "remembers" its previous voltage – at least for a while. The "memory" is the load capacitor's charge. Leakage currents will cause eventual decay of the charge (that's why DRAMs need to be refreshed!).

CMOS complements



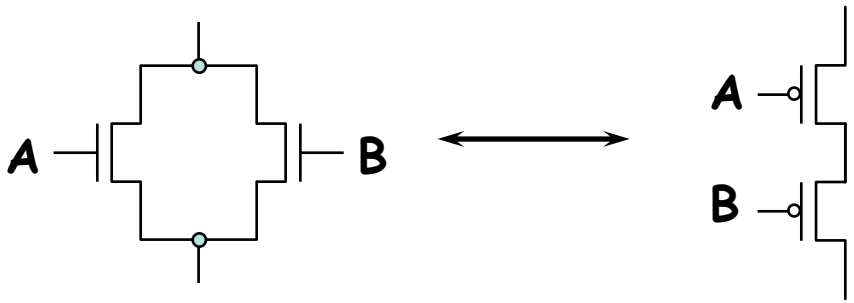


conducts when V_{GS} is high conducts when V_{GS} is low



conducts when A is high and B is high: $A \cdot B$

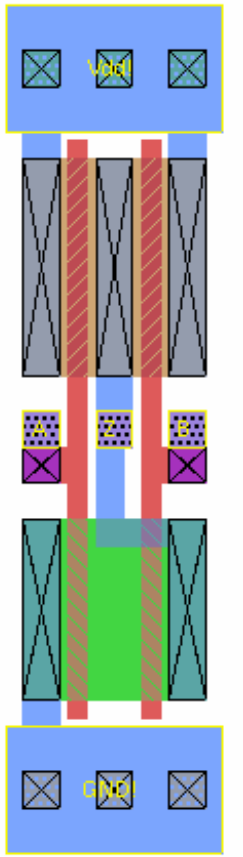
conducts when A is low or B is low: $\overline{A}+\overline{B} = \overline{A \cdot B}$

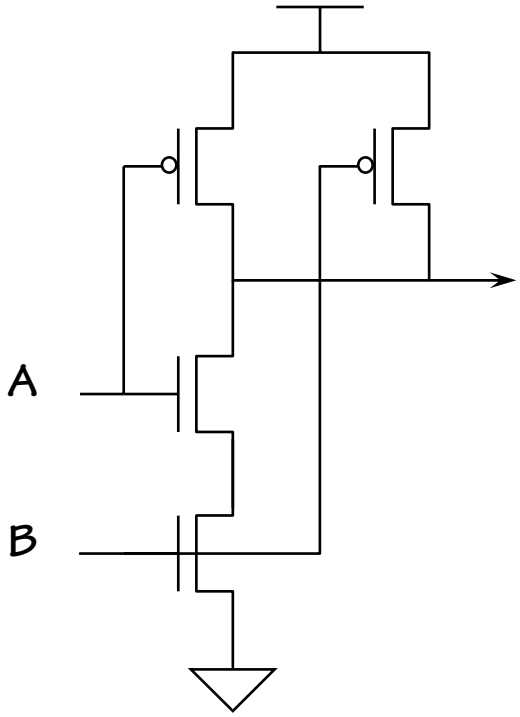


conducts when A is high or B is high: $A+B$

conducts when A is low and B is low: $\overline{A} \cdot \overline{B} = \overline{A+B}$

A pop quiz!

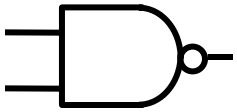




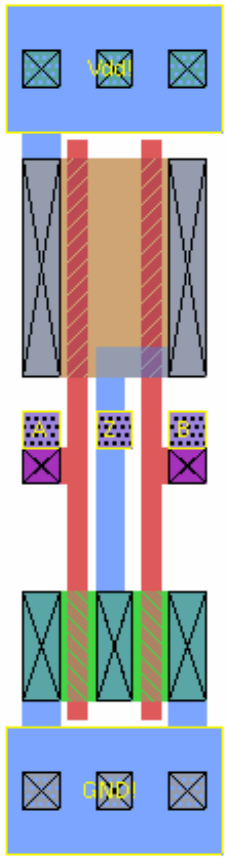
What function does this gate compute?

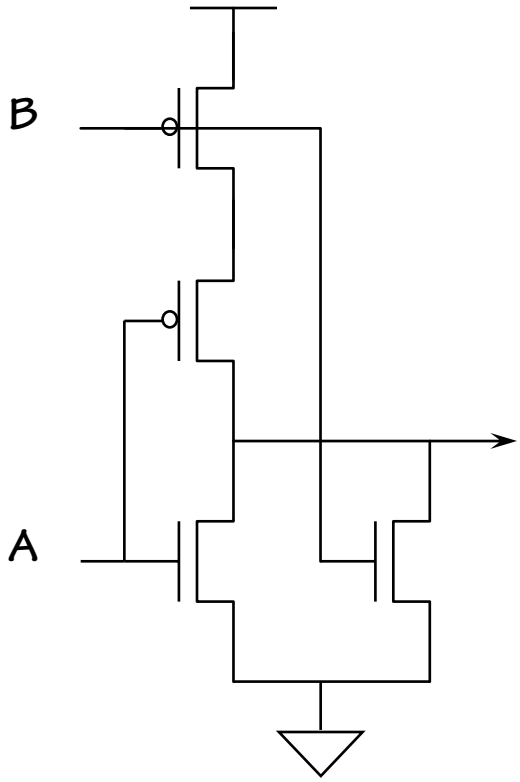
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

NAND



Here's another…

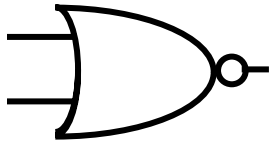




What function does this gate compute?

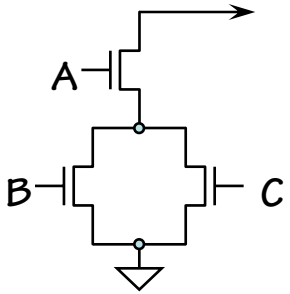
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

NOR

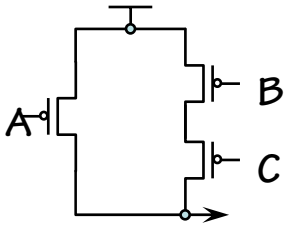


General CMOS gate recipe

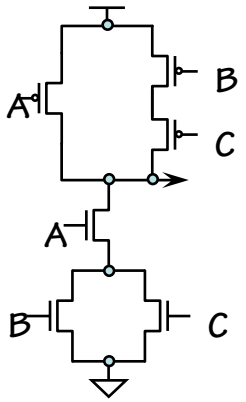
Step 1. Figure out pulldown network that does what you want, *e.g.*, F = A*(B+C)
(What combination of inputs generates a low output)



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.



So, whats the big deal?

Basic Gate Repertoire

Are we sure we have all the gates we need?

Just how many two-input gates are there?

AND	
AB	Y
00	0
01	0
10	0
11	1

OR	
AB	Y
00	0
01	1
10	1
11	1

NAND	
AB	Y
00	1
01	1
10	1
11	0

NOR	
AB	Y
00	1
01	0
10	0
11	0

Hmmmm… all of these have 2-inputs (no surprise)

… each with 4 combinations, giving 2^2 output cases

How many ways are there of assigning 4 outputs? $2^{2^2} = 2^4 = 16$

There are only so many gates

There are only 16 possible 2-input gates

… some we know already, others are just silly

INPUT AB	ZERO	AND	A > B	A	B > A	B	XOR	OR	NOR	XNOR	NOT 'B'	A <= B	NOT 'A'	B <= A	NAND	ONE
00	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
10	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
11	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

How many of these gates can be implemented using a single CMOS gate?

CMOS gates are inverting; we can always respond positively to positive transitions by cascaded gates. But suppose our logic yielded cheap *positive* functions, while inverters were expensive…

Fortunately, we can get by with a few basic gates…

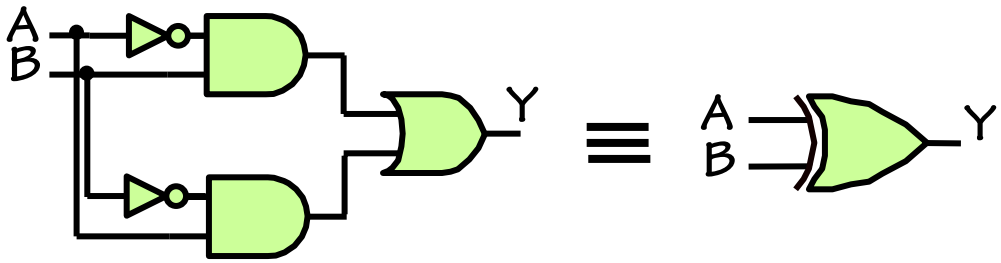
AND, OR, and NOT are sufficient… (cf Boolean Expressions):

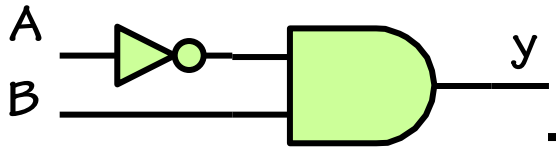
B>A

AB	Y
00	0
01	1
10	0
11	0

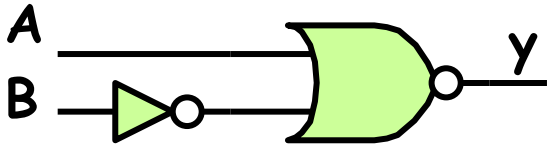
XOR

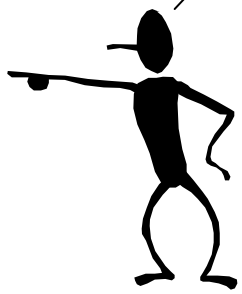
AB	Y
00	0
01	1
10	0
11	0





$\overline{A}B=\overline{A+\overline{B}}$





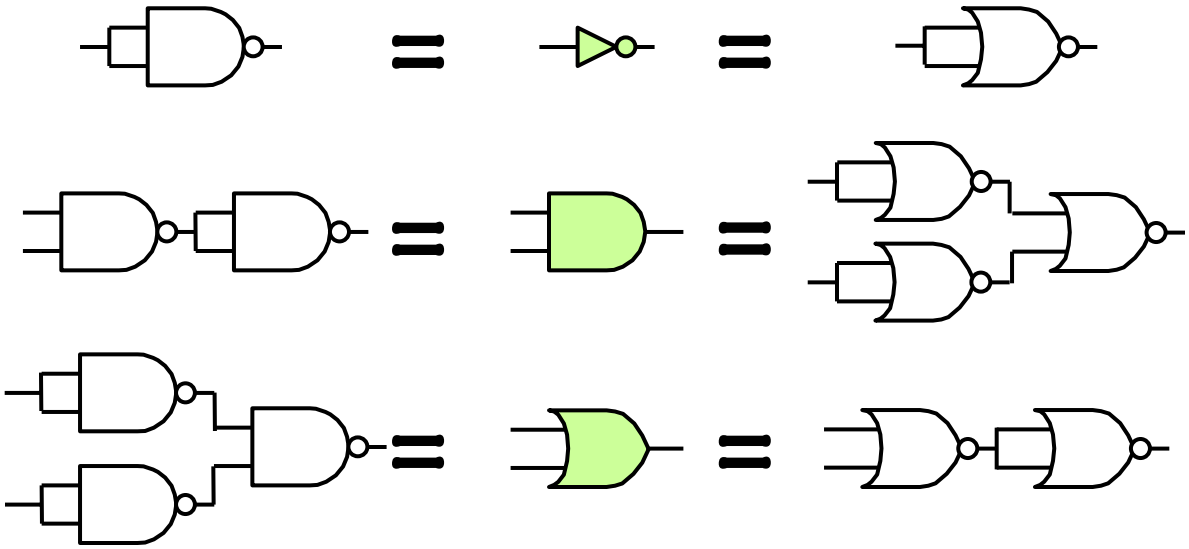
$\overline{\overline{A}\,\overline{B}}=\overline{A+B}$

$\overline{A}+\overline{B}=\overline{AB}$

How many different gates do we *really* need?

One will do!

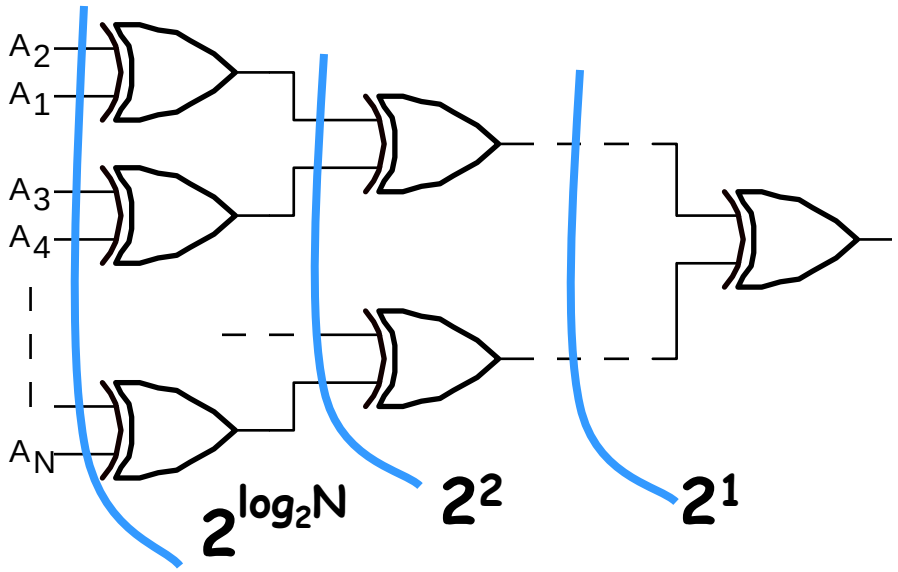
NANDs and NORs are universal:



Ah!, but what if we want more than 2 inputs?

I think that I shall never see

a circuit lovely as...



N-input TREE has O(log N) levels...

Signal propagation takes O(log N) gate delays.

Question: Can EVERY N-Input Boolean function be implemented as a tree of 2-input gates?

Here's a Design Approach

Truth Table

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

-it's systematic!
-it works!
-it's easy!
-are we done yet???

1) Write out our functional spec as a truth table
2) Write down a Boolean expression for every '1' in the output

$$Y = \overline{C}\overline{B}A + \overline{C}B\overline{A} + C\overline{B}\overline{A} + CB\overline{A}$$

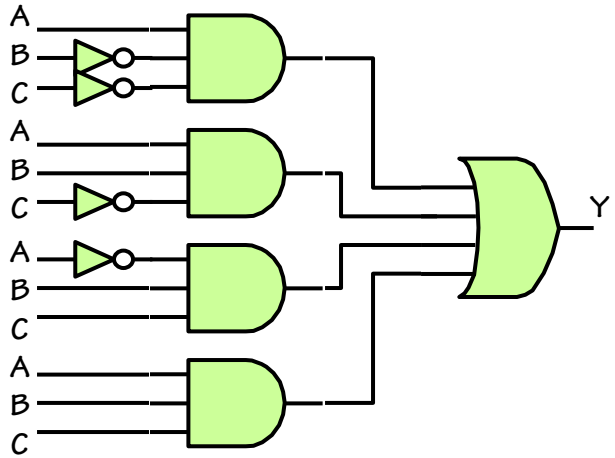
3) Wire up the gates, call it a day, and declare success!

This approach will always give us Boolean expressions in a particular form:

SUM-OF-PRODUCTS

Straightforward Synthesis

We can implement
SUM-OF-PRODUCTS
with just three levels of logic.



INVERTERS/AND/OR

Propagation delay --
No more than "3" gate delays
(well, it's actually O(log N) gate delays)