Example device: A Buffer

VTC for inverting gates

V_{out}

V_{OH}

V_{OL}

V_{in}

V_{IL} V_{IH}

0 ▷ 0   1 ▷ 1

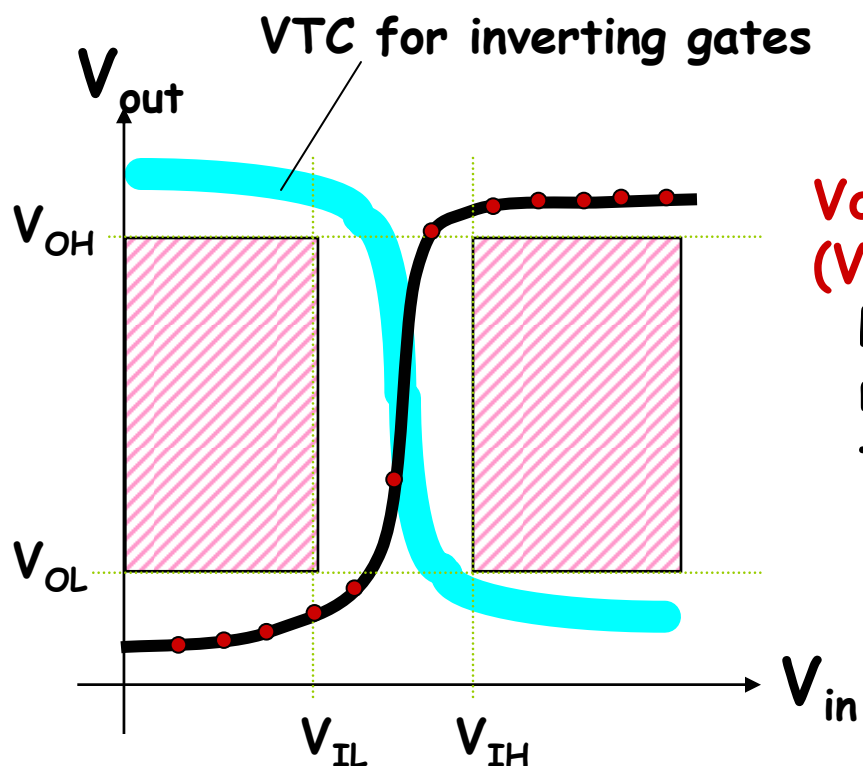**Voltage Transfer Characteristic (VTC):**

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**.
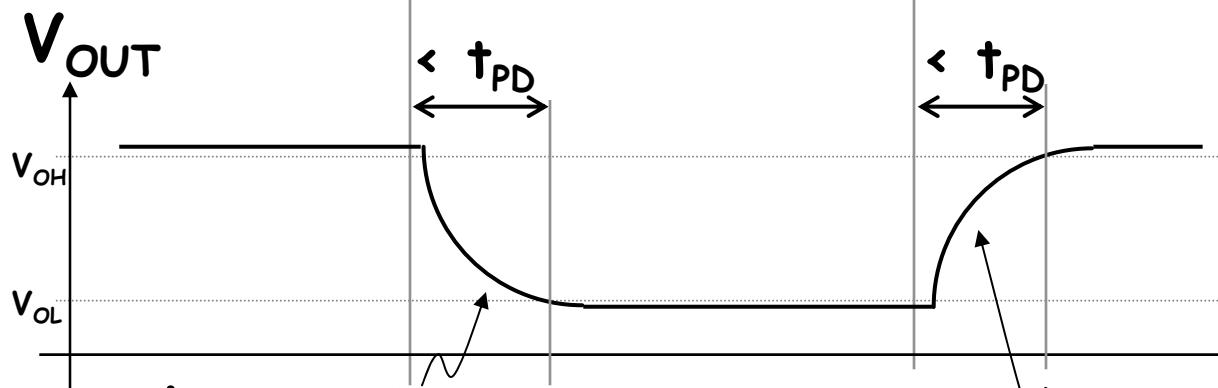
Due to unavoidable delays...

Propagation delay (t_{PD}):
   An UPPER BOUND on the delay from valid inputs to valid outputs.

V_{IN}

V_{IH}

V_{IL}
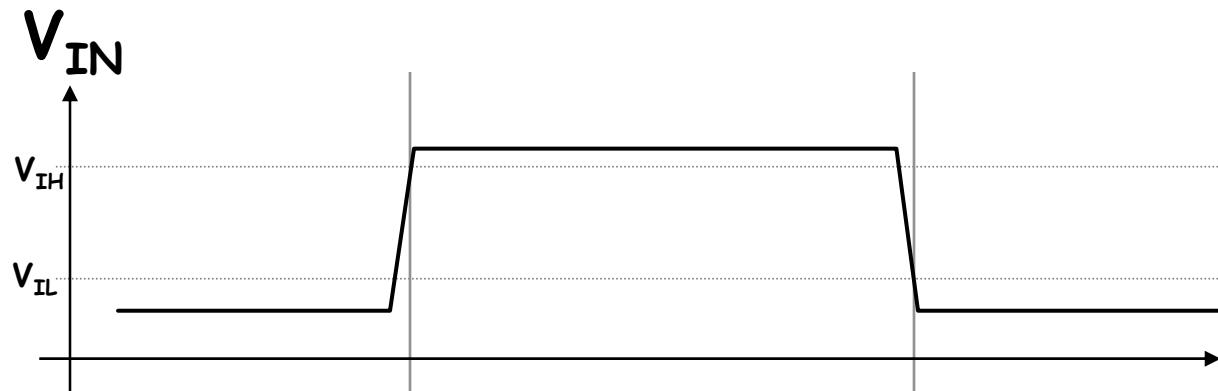
GOAL:
   minimize propagation delay!

V_{OUT}

$< t_{PD}$ $< t_{PD}$

V_{OH}

V_{OL}

ISSUE:
   keep Capacitances low and transistors fast

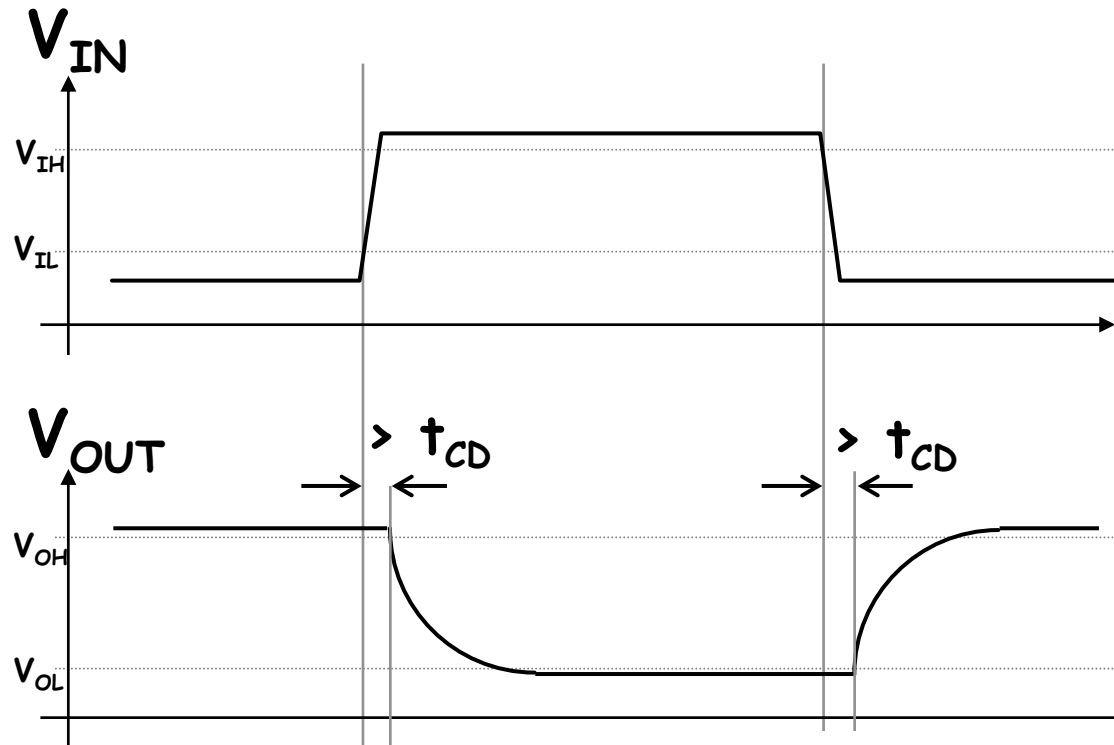time constant
.$\tau = R_{PD} \cdot C_L$

time constant
.$\tau = R_{PU} \cdot C_L$

Contamination Delay
an optional, additional timing spec

INVALID inputs take time to propagate, too...
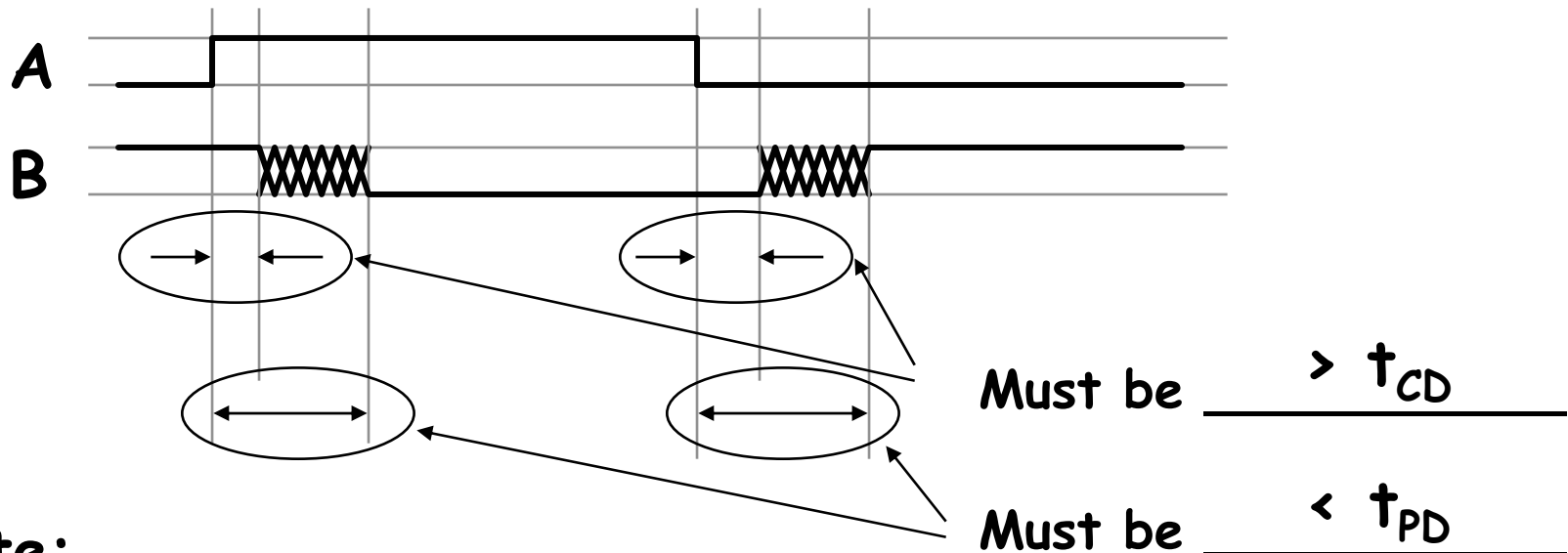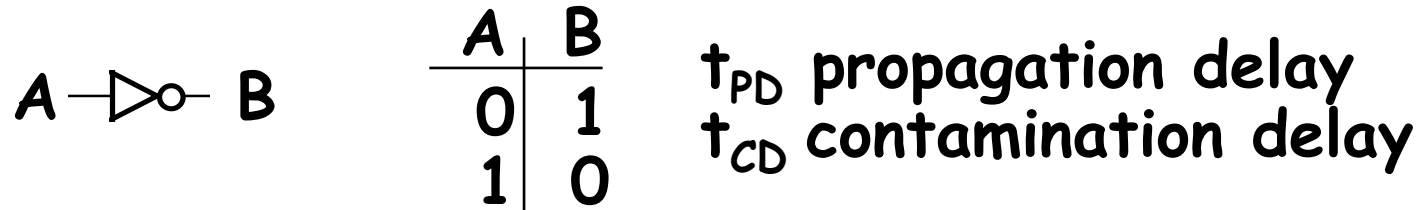

Do we really need t_{CD}?

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 $\rightarrow\!\!\!\triangleright\!\!\circ\!\!\rightarrow$ B

A	B
0	1
1	0

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

A

B

Must be _____ $> t_{CD}$

Must be _____ $< t_{PD}$

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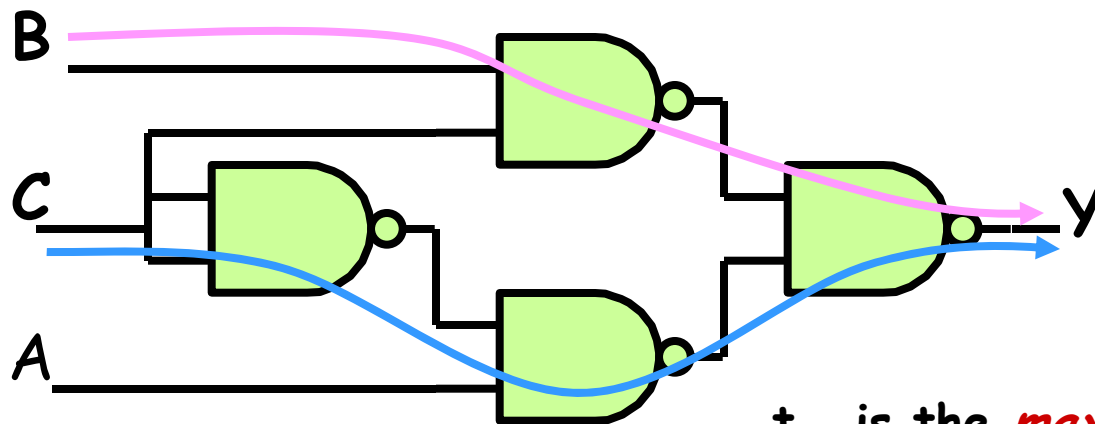1. *No Promises* during ⬭⬭⬭⬭⬭
2. Default (conservative) spec: $t_{CD} = 0$

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

$$t_{PD} = \underline{\quad 12 \quad} \text{ nS}$$
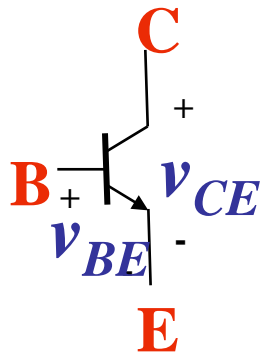
$$t_{CD} = \underline{\quad 2 \quad} \text{ nS}$$

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

The "perfect" logic family

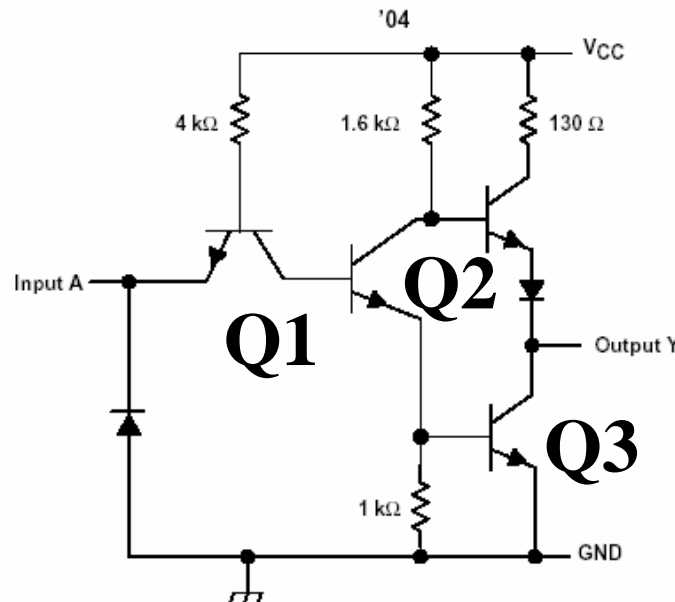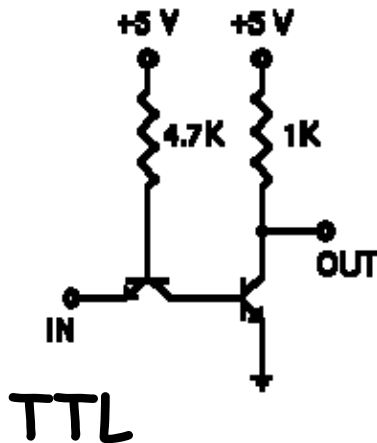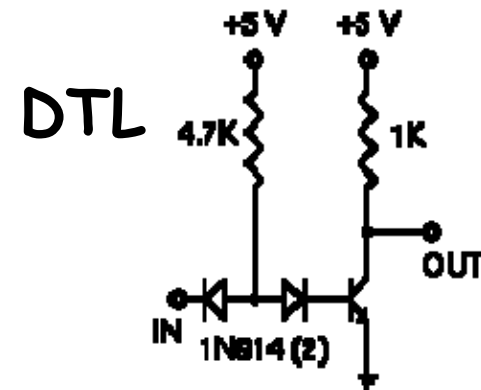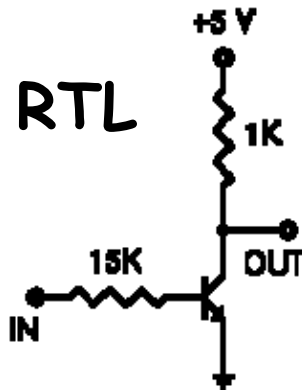- Good noise margins (want a "step" VTC)
- Implement useful selection of (binary) logic
  - INVERTER, NAND, NOR with modest fan-in (4? Inputs)
  - More complex logic in a single step? (minimize delay)
- Small physical size
  - Shorter signal transmission distances (faster)
  - Cost proportional to size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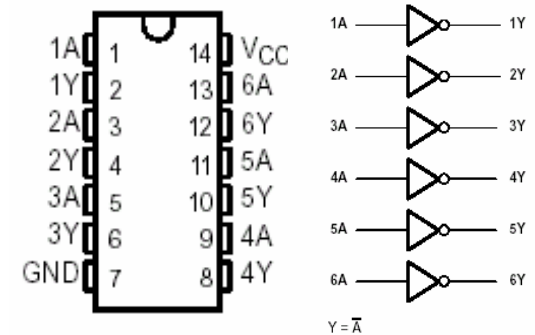 - Massive processing without meltdown

Transitor-transitor Logic (TTL)

C

$+$

B $+$ v_{CE}

v_{BE} -

E

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

RTL

+5 V

1K

15K   OUT

IN

DTL

+5V   +5V

4.7K   1K

OUT

IN   1N914 (2)

TTL

+5 V   +5 V

4.7K   1K

OUT

IN

'04

Vcc

4 kΩ   1.6 kΩ   130 Ω

Input A

Q1   **Q2**

Output Y

Q3

1 kΩ

GND

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

1A	1	14	Vcc
1Y	2	13	6A
2A	3	12	6Y
2Y	4	11	5A
3A	5	10	5Y
3Y	6	9	4A
GND	7	8	4Y

1A ——▷o—— 1Y
2A ——▷o—— 2Y
3A ——▷o—— 3Y
4A ——▷o—— 4Y
5A ——▷o—— 5Y
6A ——▷o—— 6Y

$Y = \overline{A}$

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($-I_{IL}/I_{OL}$,$-I_{IH}/I_{OH}$) ≈ 10.

- Current-based logic → power dissipation even in steady state, limitations on fanout

Complementary MOS Logic
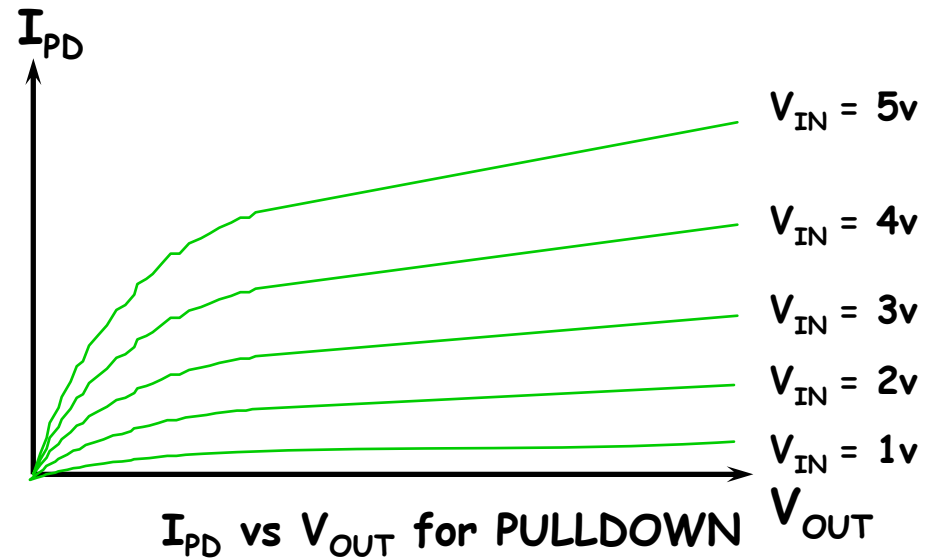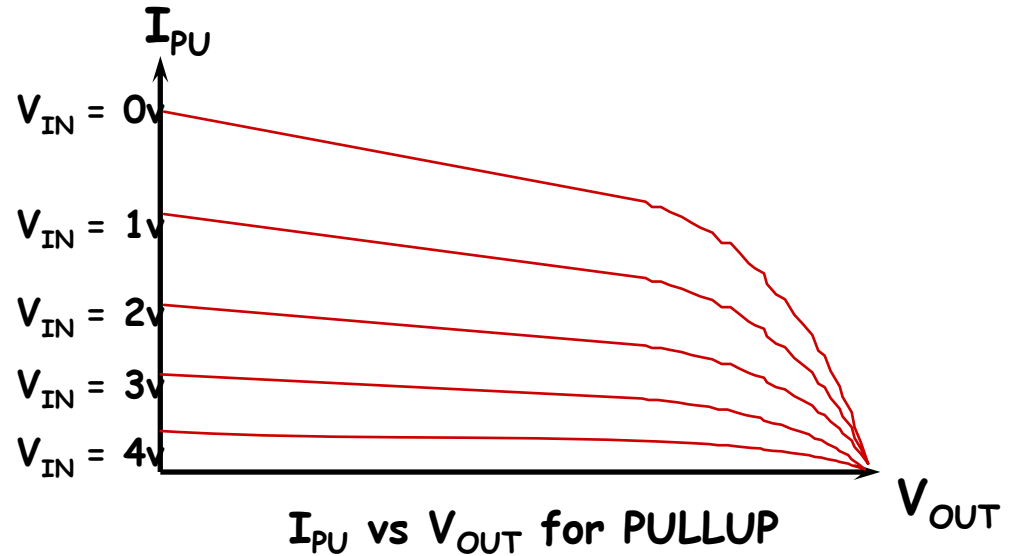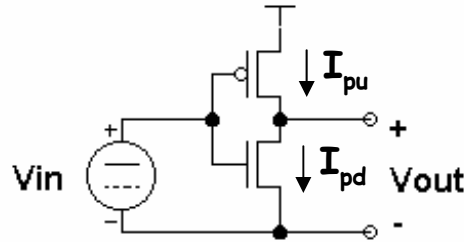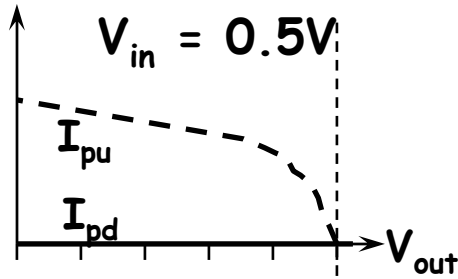
P-Channel
MOSFET

G

S = VDD

I_{PU}

D

V_{out}

D

I_{PD}

G

S = 0V

V_{in}

N-Channel
MOSFET

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

I_{PU}

V_{IN} = 0v

V_{IN} = 1v

V_{IN} = 2v

V_{IN} = 3v

V_{IN} = 4v

V_{OUT}

I_{PU} vs V_{OUT} for PULLUP

I_{PD}

V_{IN} = 5v

V_{IN} = 4v

V_{IN} = 3v

V_{IN} = 2v

V_{IN} = 1v

V_{OUT}

I_{PD} vs V_{OUT} for PULLDOWN

CMOS Inverter VTC

$V_{in} = 0.5V$

I_{pu}

I_{pd}

V_{out}

$V_{in} = 1.5V$

I_{pu}

I_{pd}

V_{out}

$V_{in} = 2.5V$

I_{pu} I_{pd}

V_{out}

$\downarrow I_{pu}$

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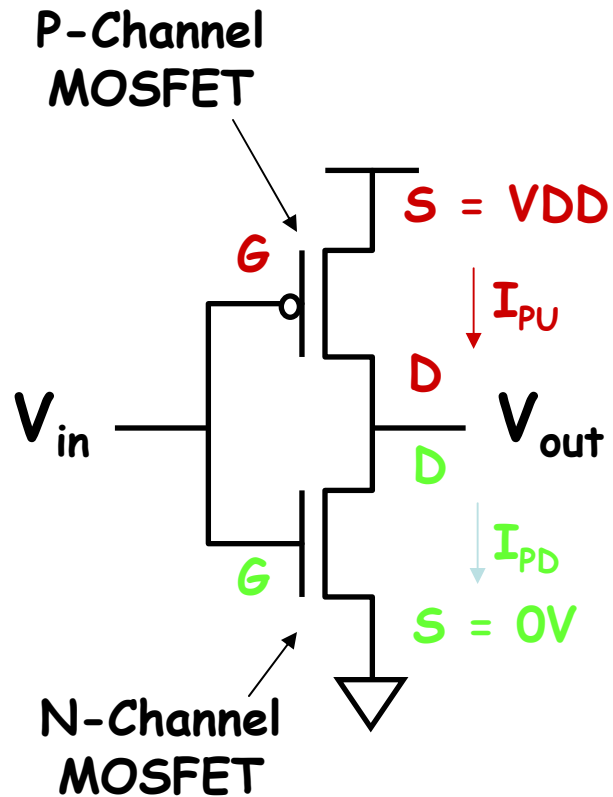
Steady state reached when V_{out} reaches value where $I_{pu} = I_{pd}$.

V_{out}

V_{OH}

V_{OL}

V_{IL} V_{IH}

V_{in}

$V_{in} = 3.5V$

I_{pd}

I_{pu}

V_{out}

$V_{in} = 4.5V$

I_{pd}

I_{pu}
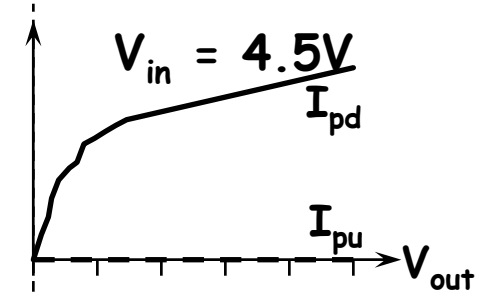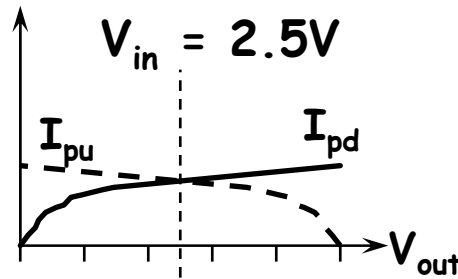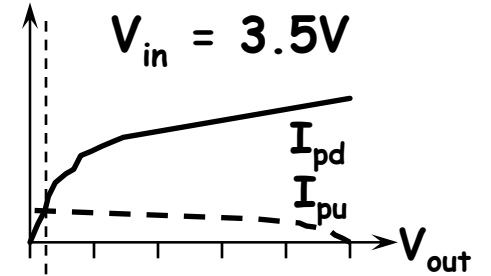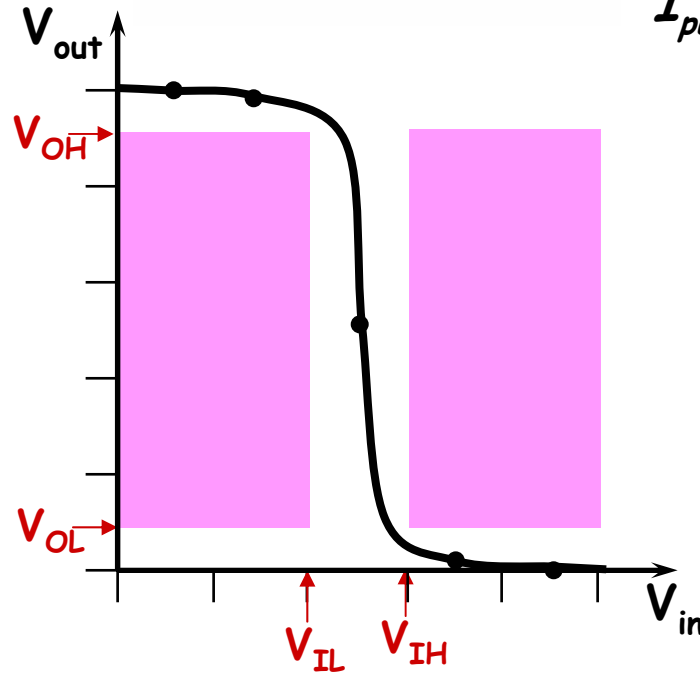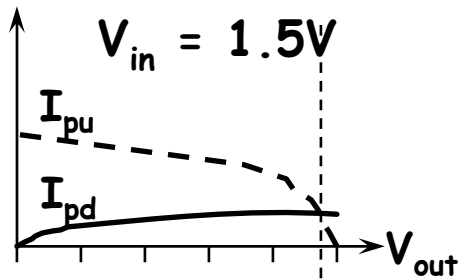
V_{out}

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)
  - $V_{IL} \approx$ just under $V_{DD}/2$, $V_{IH} \approx$ 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!
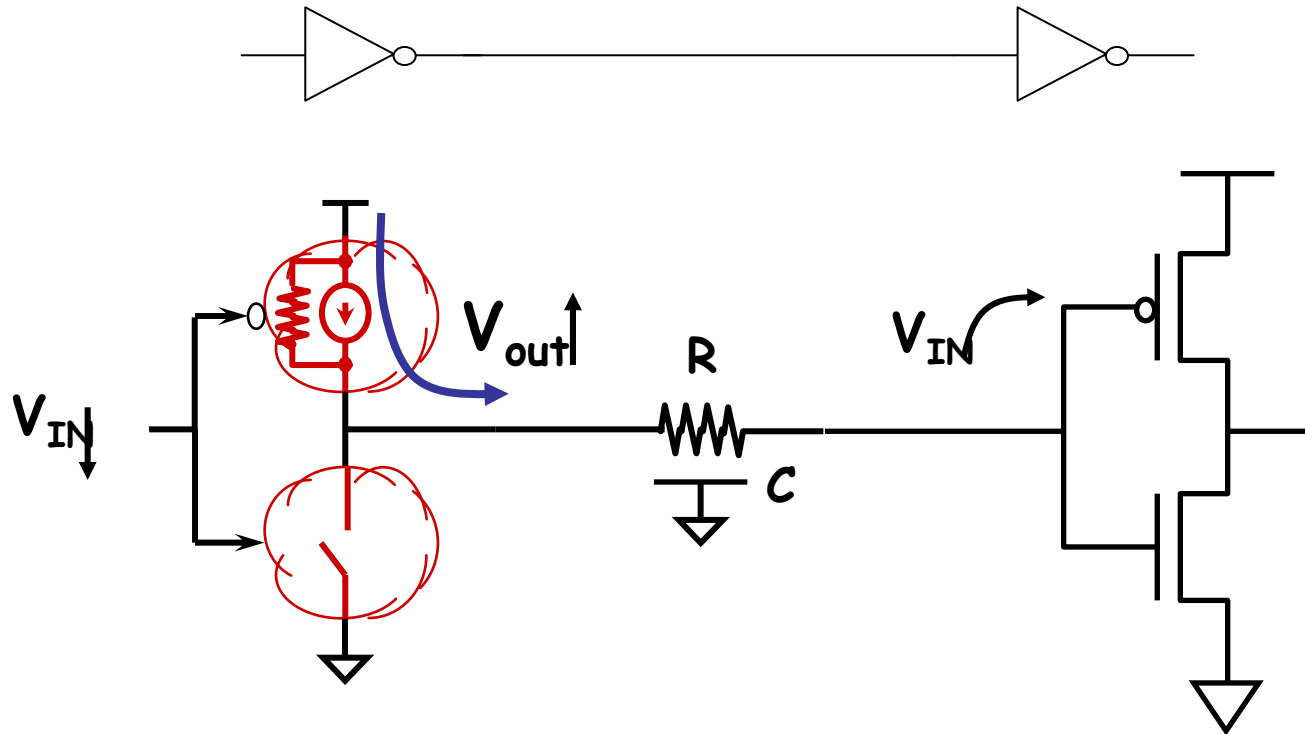 - Power dissipated only when signals change (ie, power proportional to operating frequency).

Multiple interconnect layers
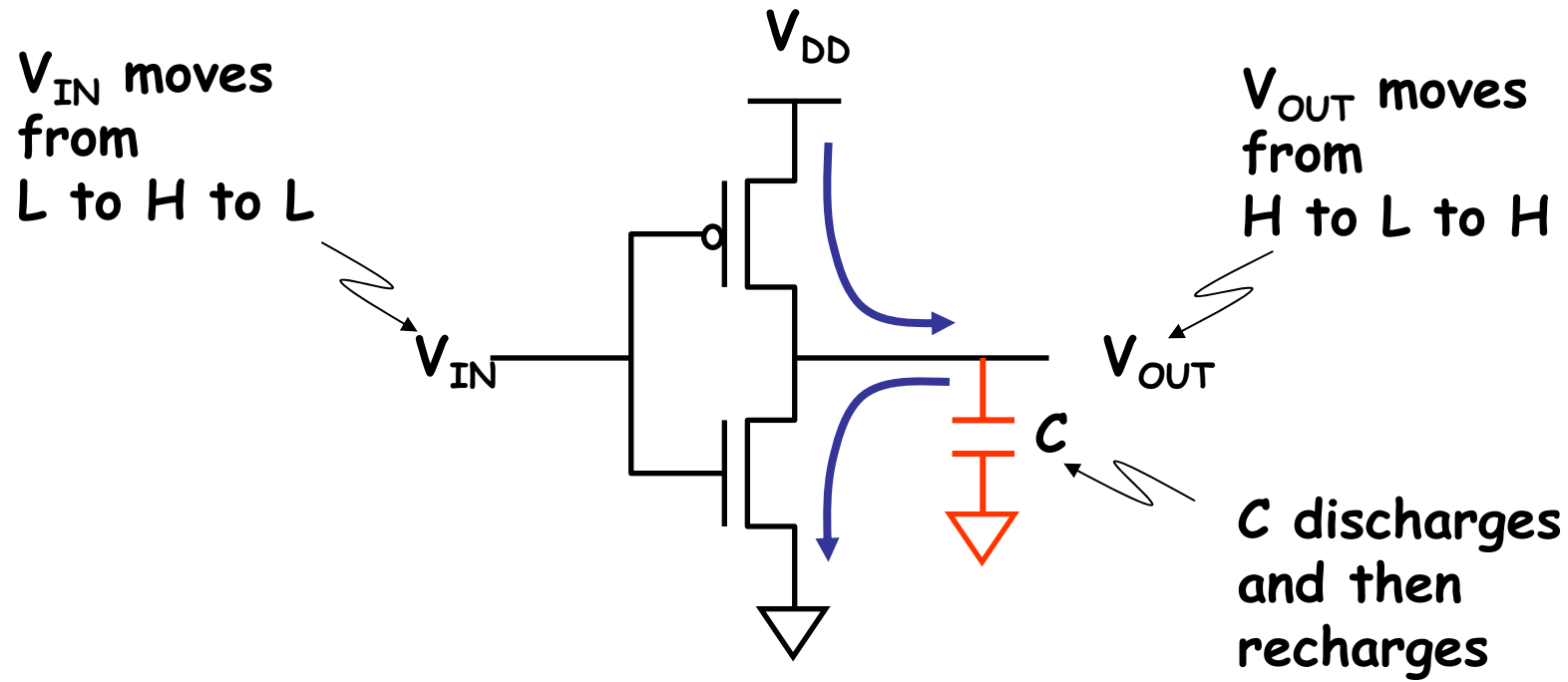
IBM photomicrograph (SiO_2 has been removed!)

Metal 2

M1/M2 via

Metal 1

Polysilicon

Diffusion
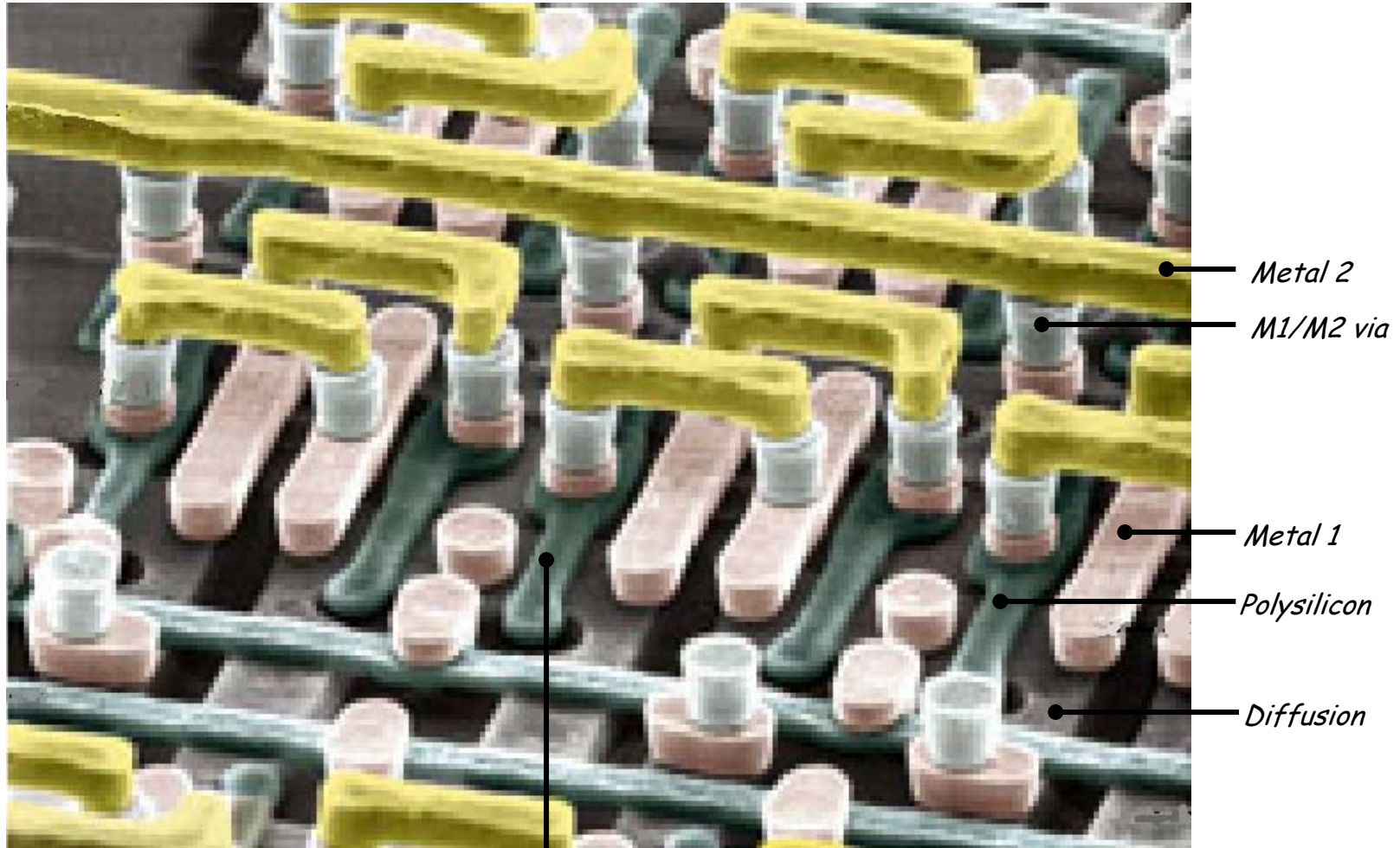
Mosfet (under polysilicon gate)

Big Issue 1: Wires

- ## Today (i.e., 100nm):

  $\tau_{RC} \approx$ 50ps/mm

  Implies > 1 ns to traverse a 20mm x 20mm chip

  This is a long time in a 2GHz processor

Big Issue 2: Power

V_{IN} moves
from
L to H to L

V_{DD}

V_{IN}

V_{OUT}

C

V_{OUT} moves
from
H to L to H

C discharges
and then
recharges

- **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)

MIT Computation Center and Pizzeria

I've got the solution!

- 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
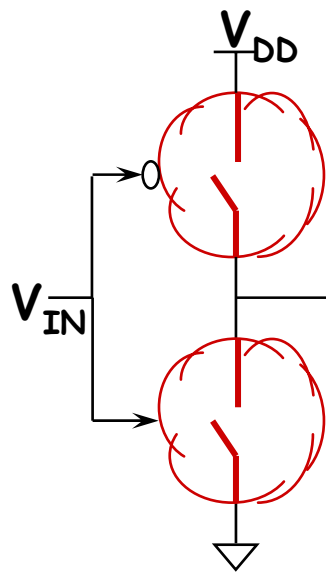   - nC and f continue to grow

Hey: could we somehow recycle the charge?

CMOS Gate Recipe: Think Switches

V_{DD}

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

V_{IN}

V_{OUT}

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

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

L —▷o— H

$V_{IN} \leq V_{IL}$

$V_{OUT} \geq V_{OH}$

H —▷o— L

$V_{IN} \geq V_{IH}$

$V_{OUT} \leq V_{OL}$
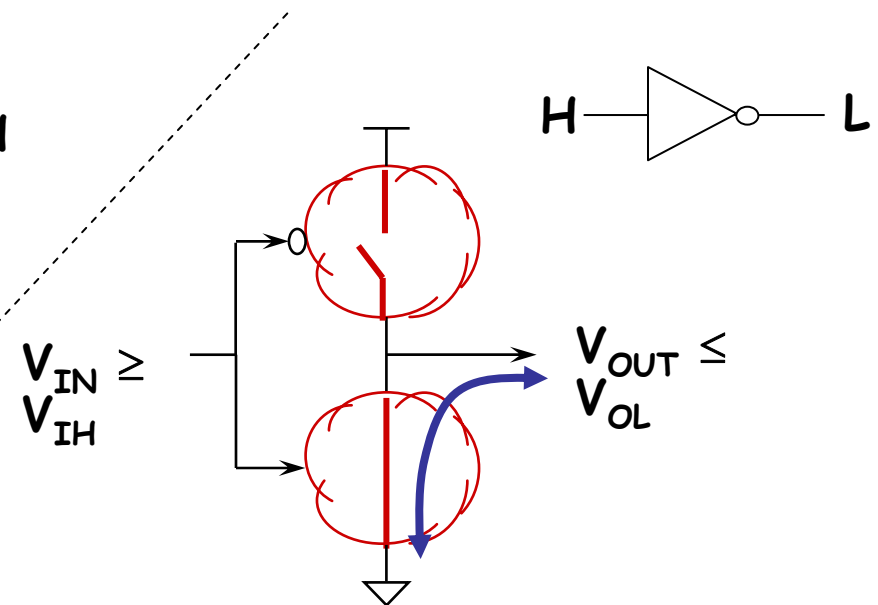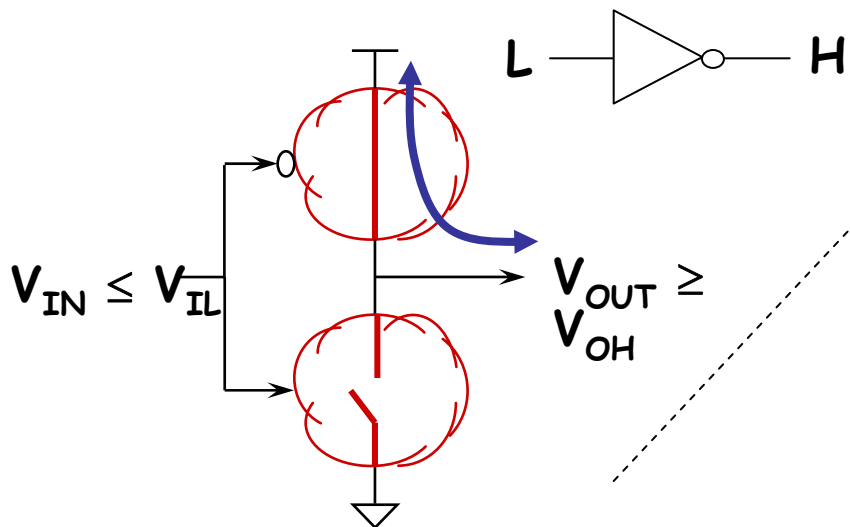
Beyond Inverters:
Complementary pullups and pulldowns

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
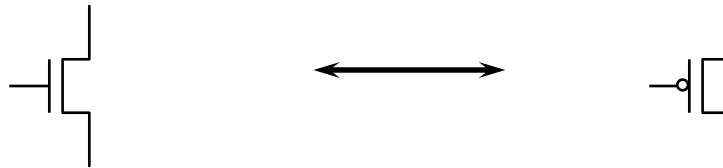(that's why DRAMs need to be refreshed!).

CMOS complements

What a nice V_{OH} you have...
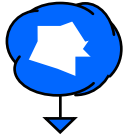
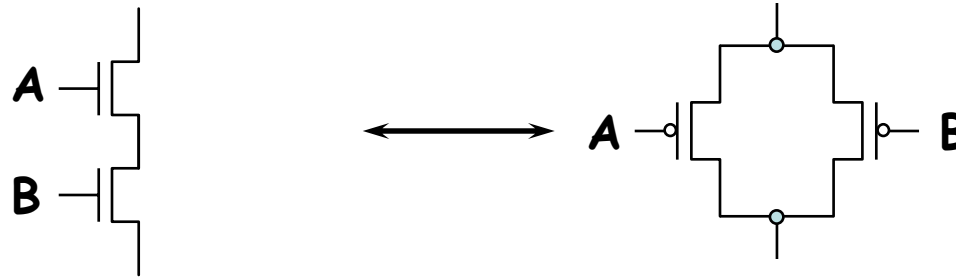*Thanks. It runs in the family...*
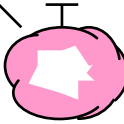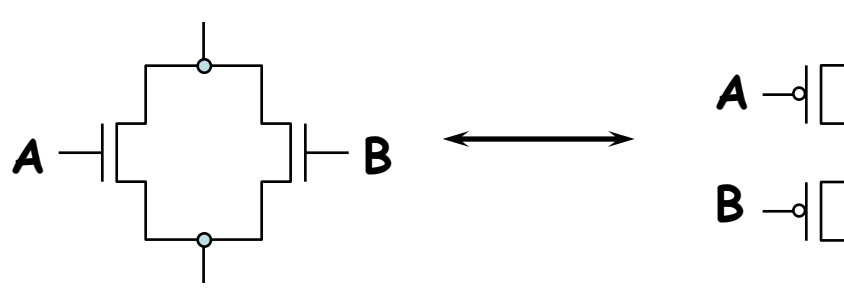
conducts when V_{GS} is high   conducts when V_{GS} is low

A
B

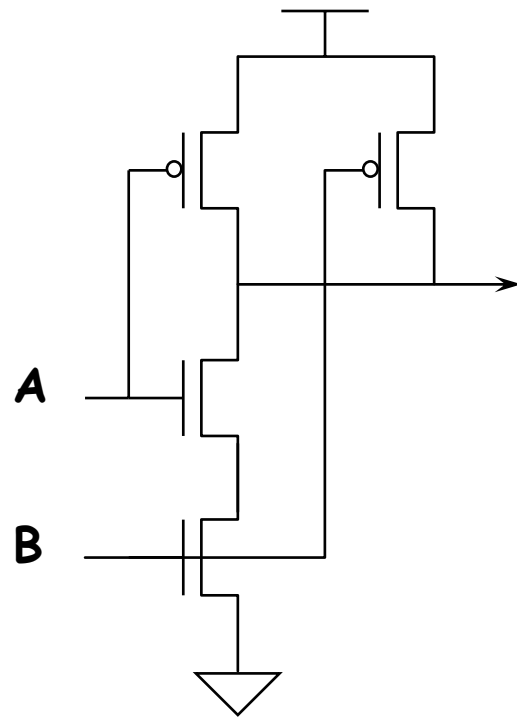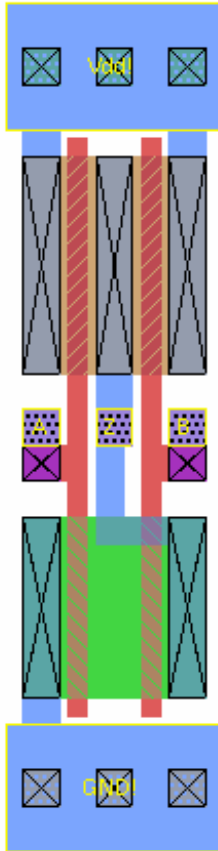conducts when A is high <u>and</u> B is high: $A \cdot B$

A   B

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

A   B

conducts when A is high <u>or</u> B is high: $A + B$

A
B

conducts when A is <u>low</u> <u>and</u> 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
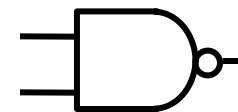
Here's another...
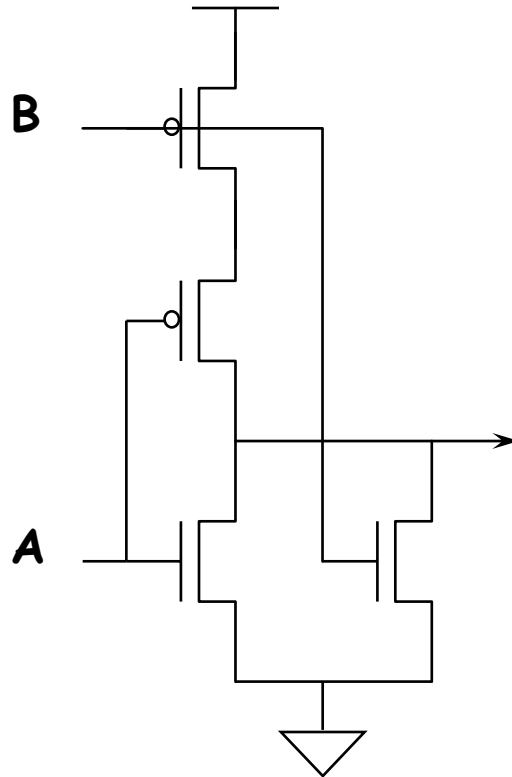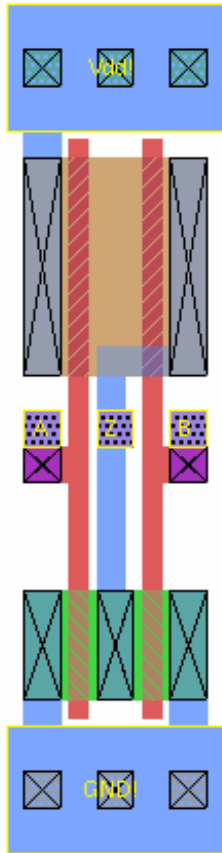
What function does
this gate compute?

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

NOR
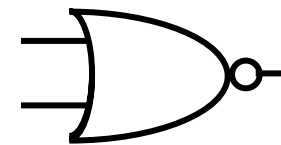
General CMOS gate recipe

Step 1. Figure out pulldown network that <u>does</u> what you want, e.g., $F = \overline{A*(B+C)}$ (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

Looks pretty easy to do!

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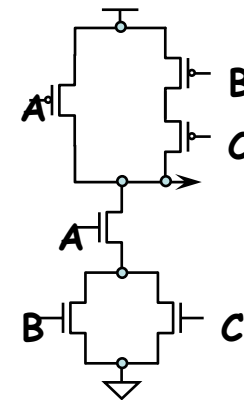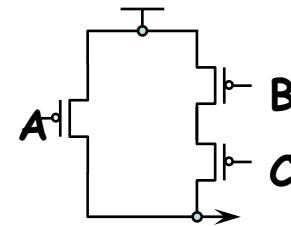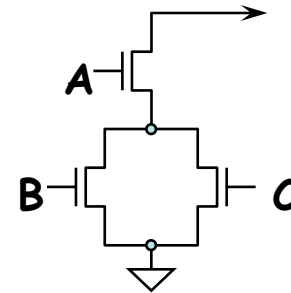
Basic Gate Repertoire

Are we sure we have all the gates we need?
Just how many two-input gates are there?

AND		OR		NAND		NOR	
AB	Y	AB	Y	AB	Y	AB	Y
00	0	00	0	00	1	00	1
01	0	01	1	01	1	01	0
10	0	10	1	10	1	10	0
11	1	11	1	11	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? $\underline{2^{2^2} = 2^4 = 16}$

There are only so many gates

There are only 16 possible 2-input gates
   ... some we know already, others are just silly

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

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

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...

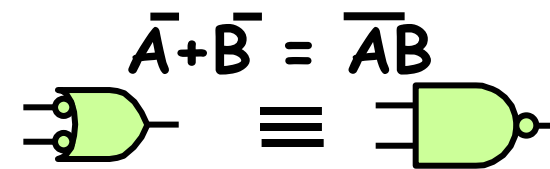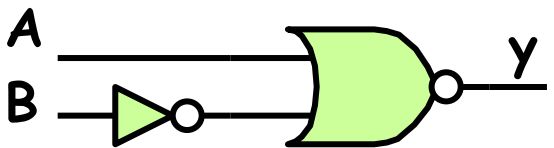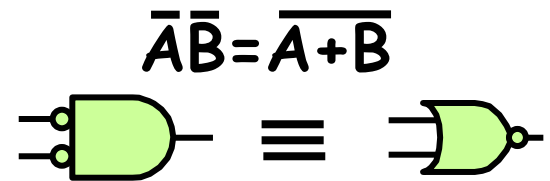# Fortunately, we can get by with a few basic gates...

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	1
11	0

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

That is just DeMorgan's Theorem!
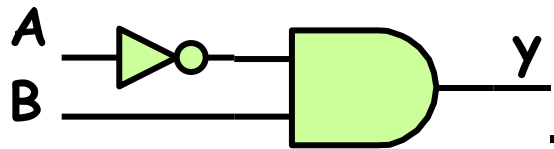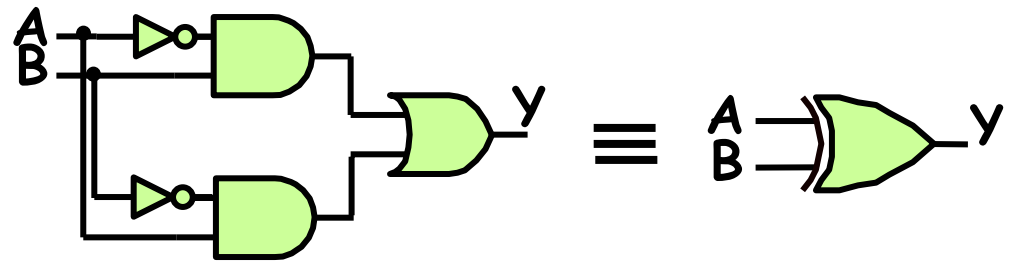
$$\bar{A}\bar{B} = \overline{A + B}$$

$$\bar{A} + \bar{B} = \overline{AB}$$
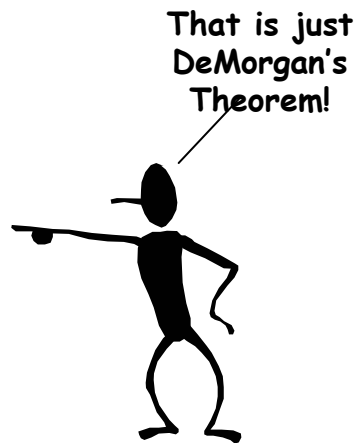
How many different gates do we *really* need?

One will do!

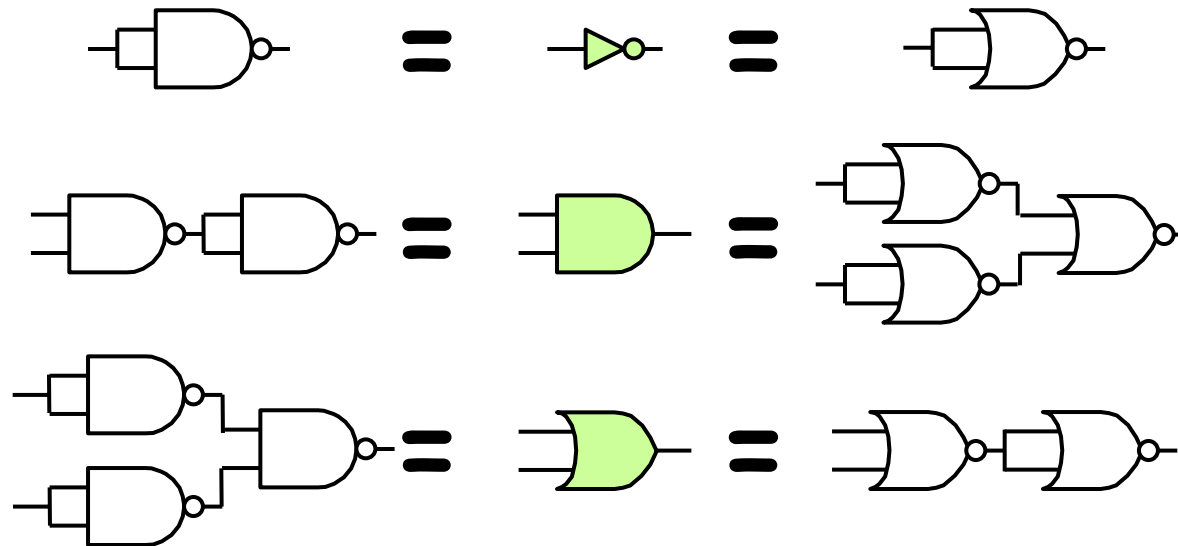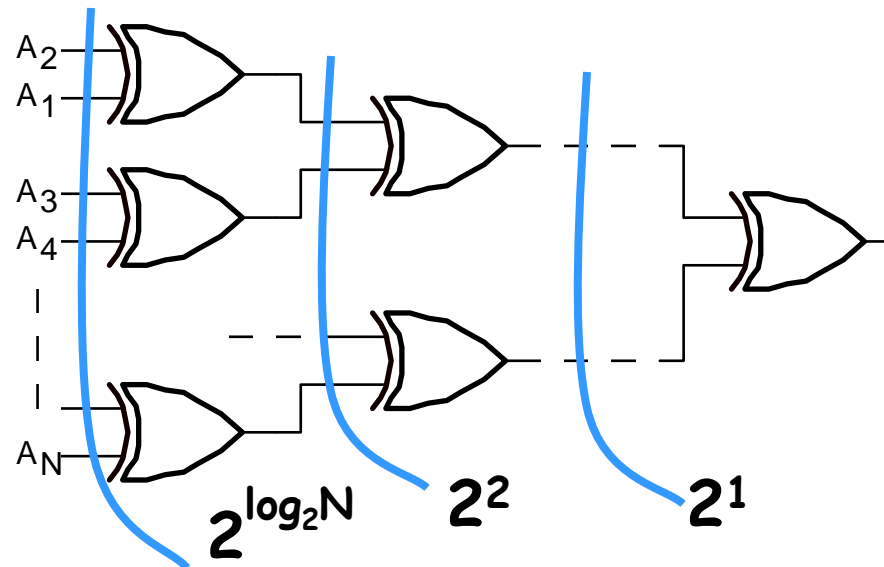NANDs and NORs are <u>underline</u>:

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

I think that I shall never see
a circuit lovely as...

A_2
A_1
A_3
A_4
A_N

$2^{\log_2 N}$ 2^2 2^1

N-input TREE has O(<u>log N</u>) levels...

Signal propagation takes O(<u>log N</u>) 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

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

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)