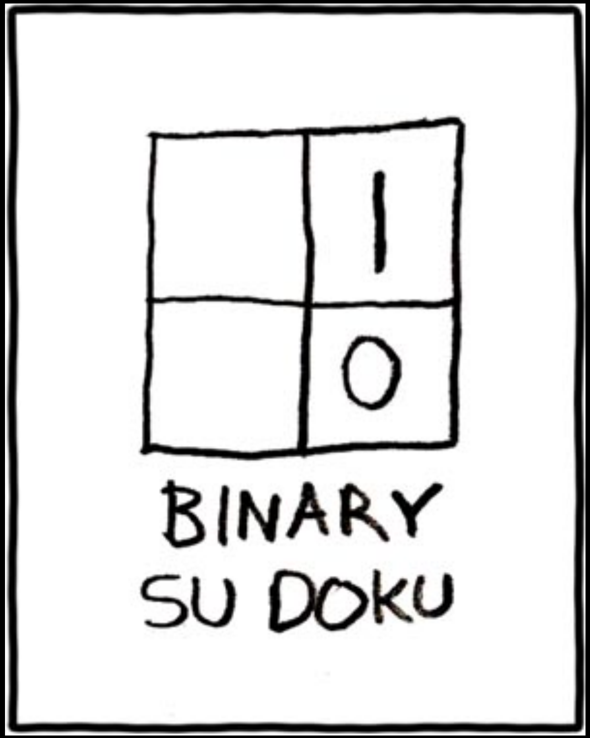
Gates and Logic

Hakim Weatherspoon
CS 3410, Spring 2011
Computer Science
Cornell Universty

See: P&H Appendix C.0, C.1, C.2

Gates and Logic



http://www.xkcd.com/74/

See: P&H Appendix C.0, C.1, C.2

Announcements

Class newsgroup created

- Posted on web-page
- Use it for partner finding

First assignment is to find partners

Sections start next week

- Use this weeks section to find a partner

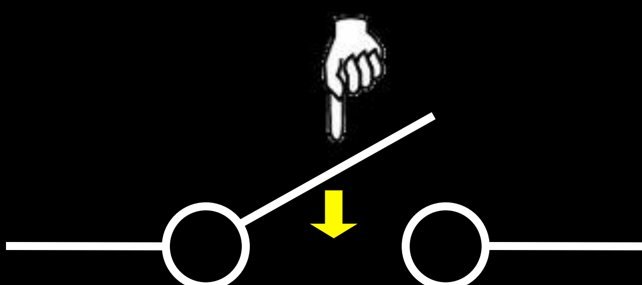
Note about class

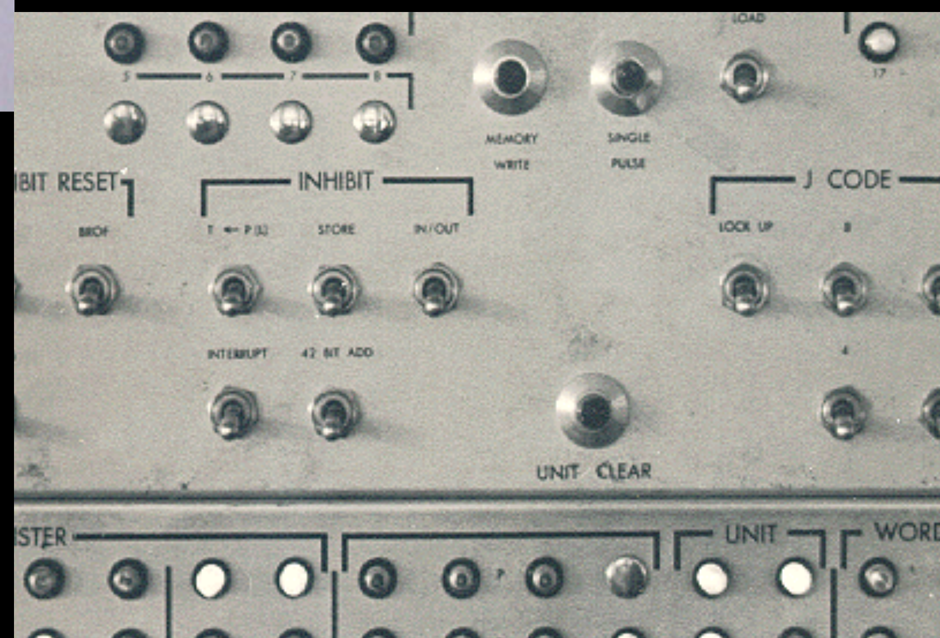
- No Verilog or VHDL
- Clickers not required, but will use them from time-to-time

A switch

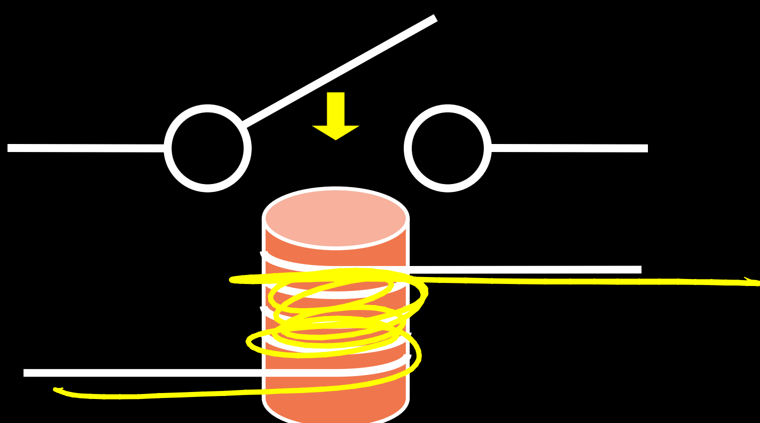
- Acts as a *conductor* or *insulator*

- Can be used to build amazing things…





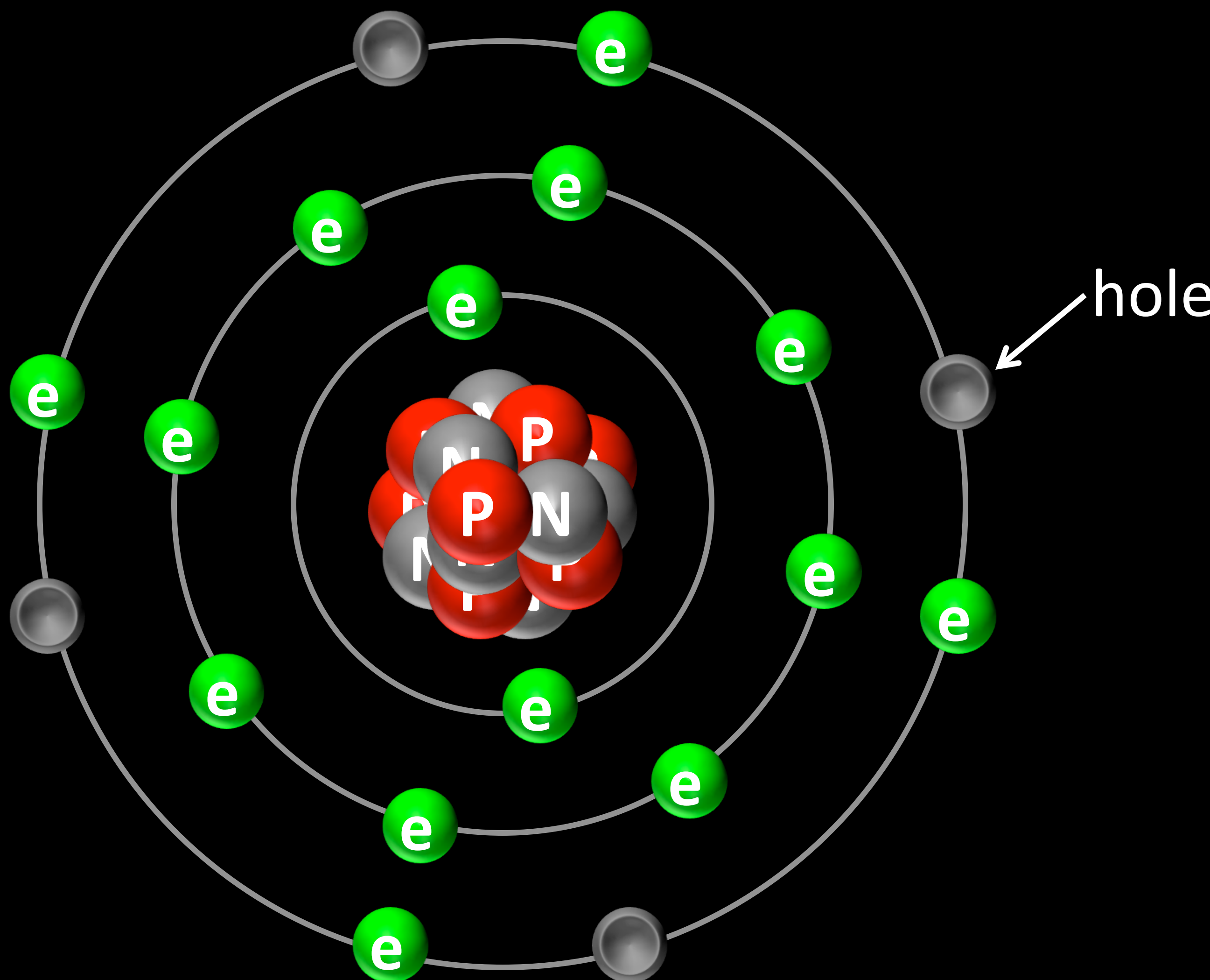
Better Switch



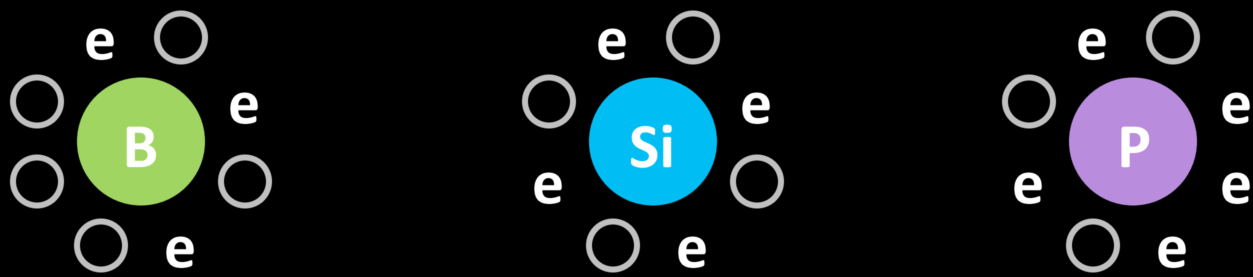
- One current controls another (larger) current

- Static Power:
  - Keeps consuming power when in the *ON* state
- Dynamic Power:
  - Jump in power consumption when switching

Atoms



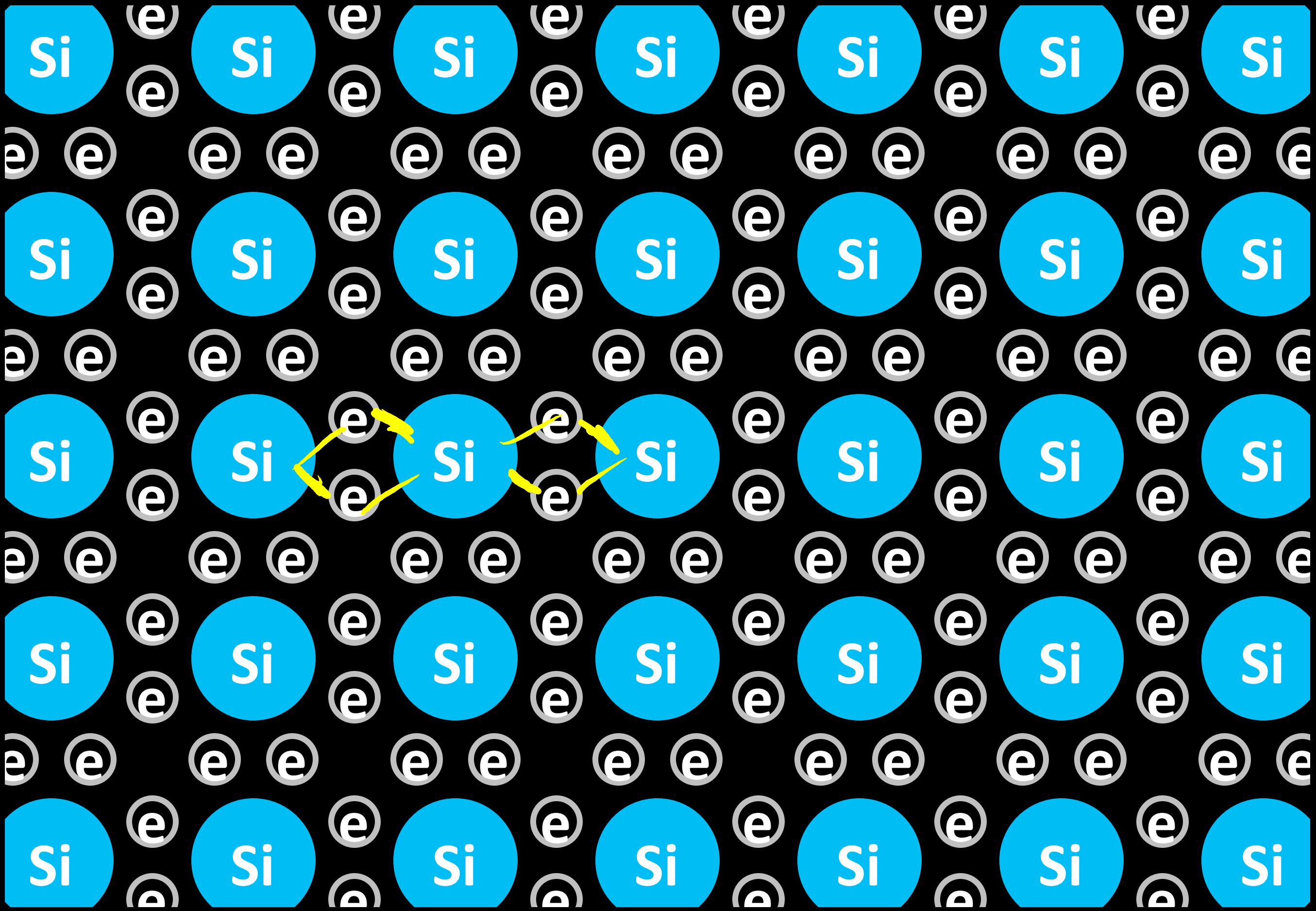
Elements



Boron Silicon Phosphorus

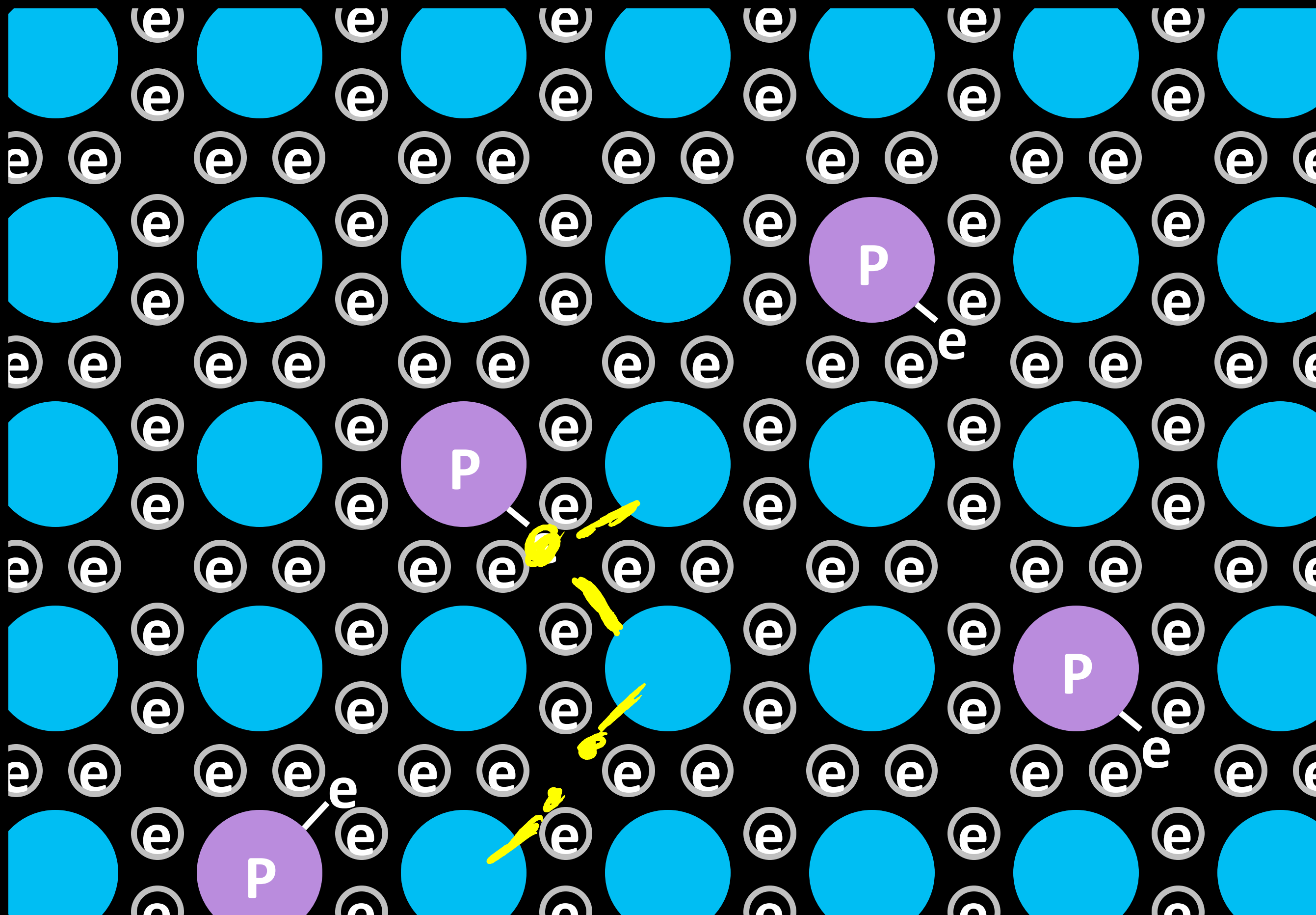
Silicon Crystal

Silicon



Phosphorus Doping

N-Type: Silicon + Phosphorus

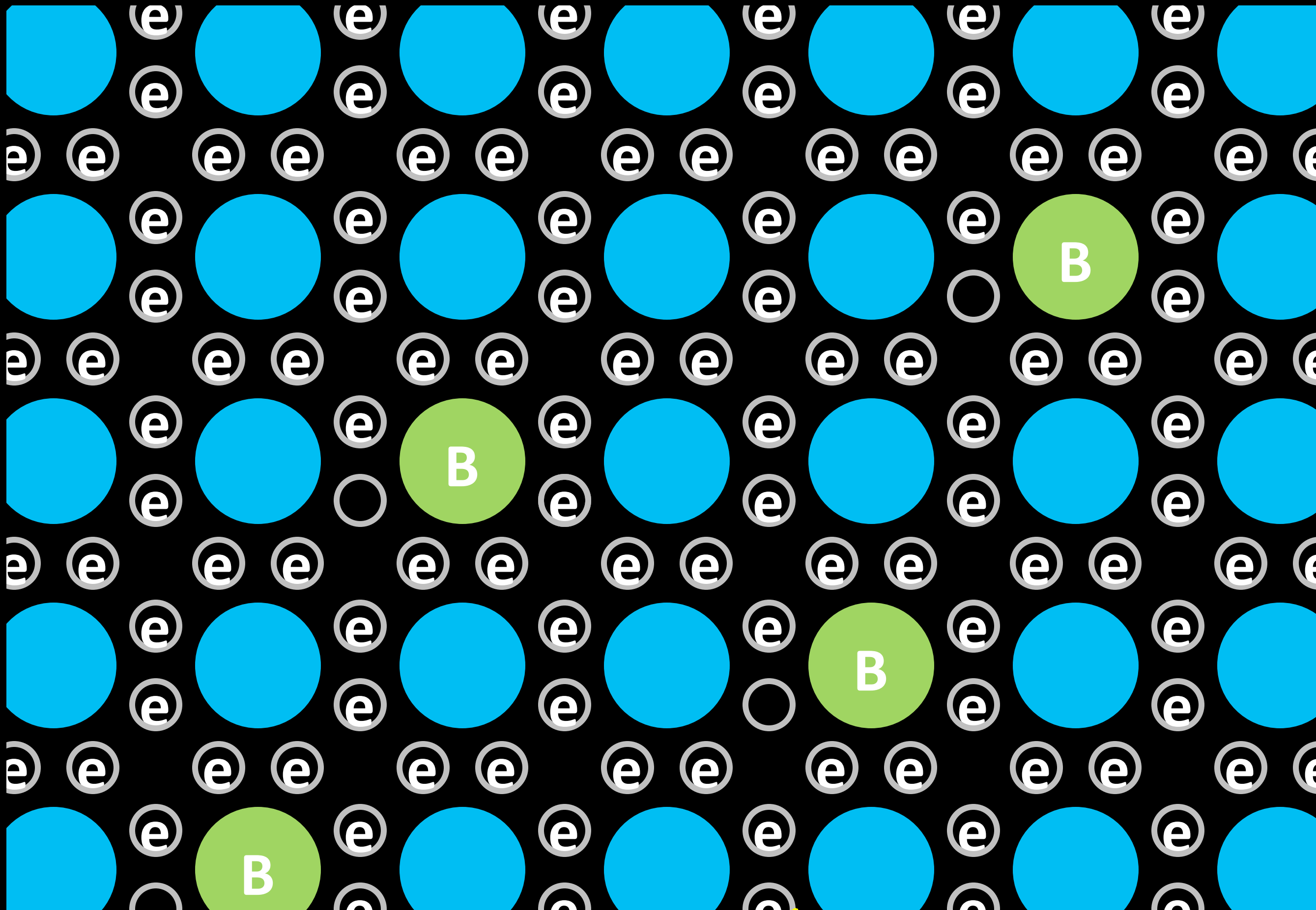


mobile electrons ⇒ conductor

h. + v. ⇒ depleted ⇒ Ins.

Boron Doping

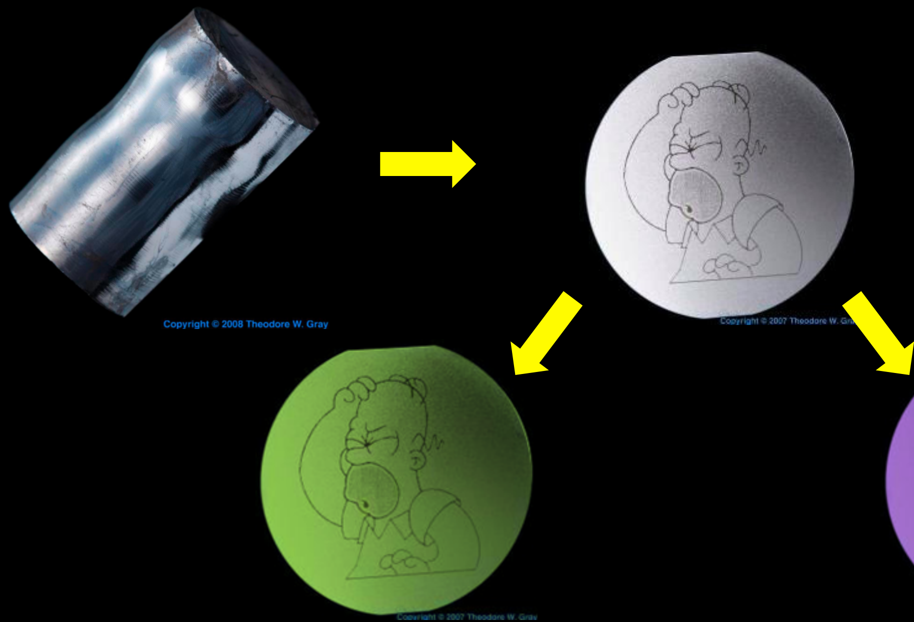
P-Type: Silicon + Boron



mobile holes ⇒ cond.

— ⇒ depleted ⇒ Ins.

Semiconductors



Insulator

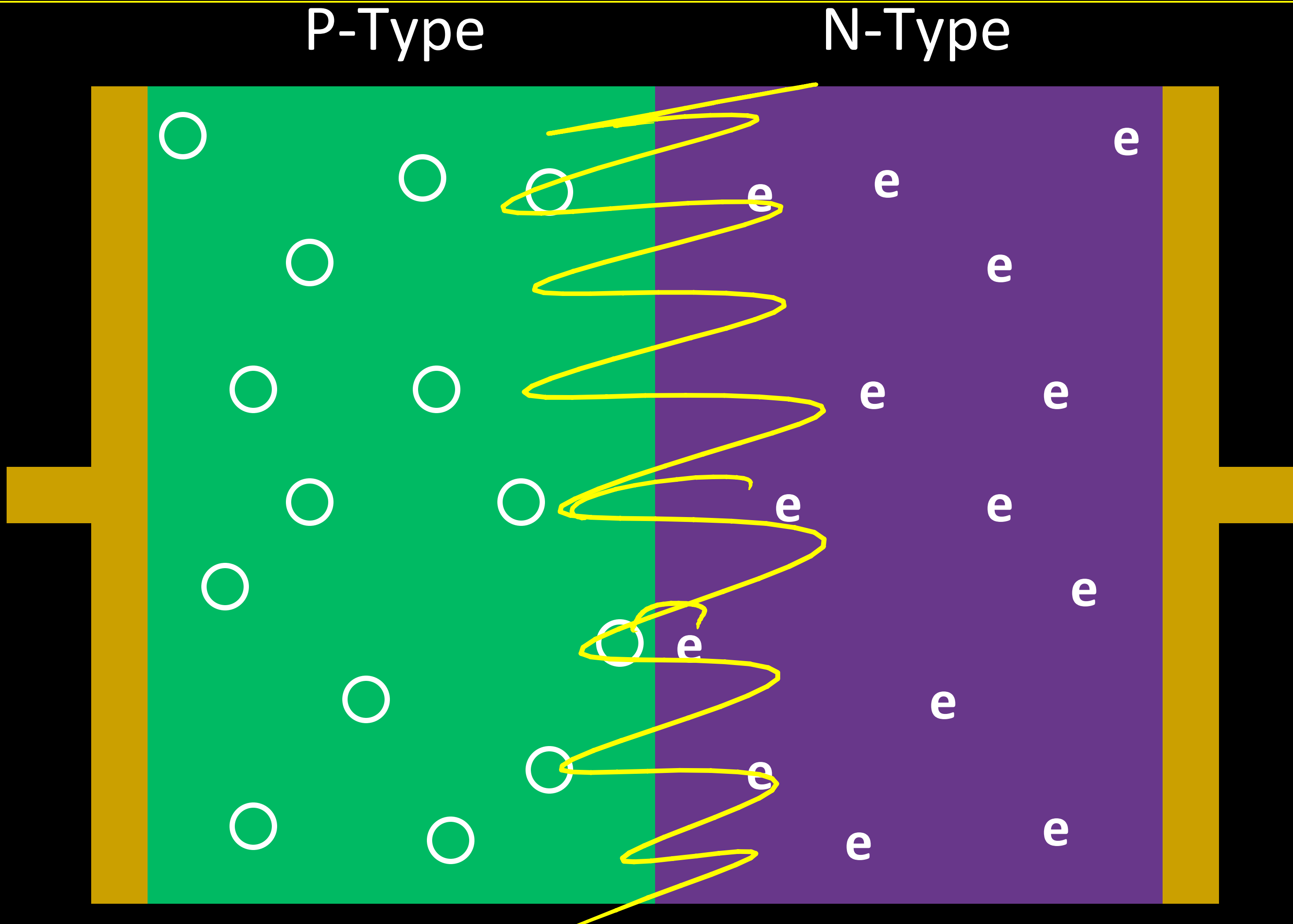
p-type (Si+Boron)
has mobile holes:

low voltage (depleted)
→ insulator
high voltage (mobile holes)
→ conductor

n-type (Si+Phosphorus)
has mobile electrons:

low voltage (mobile electrons)
→ conductor
high voltage (depleted)
→ insulator

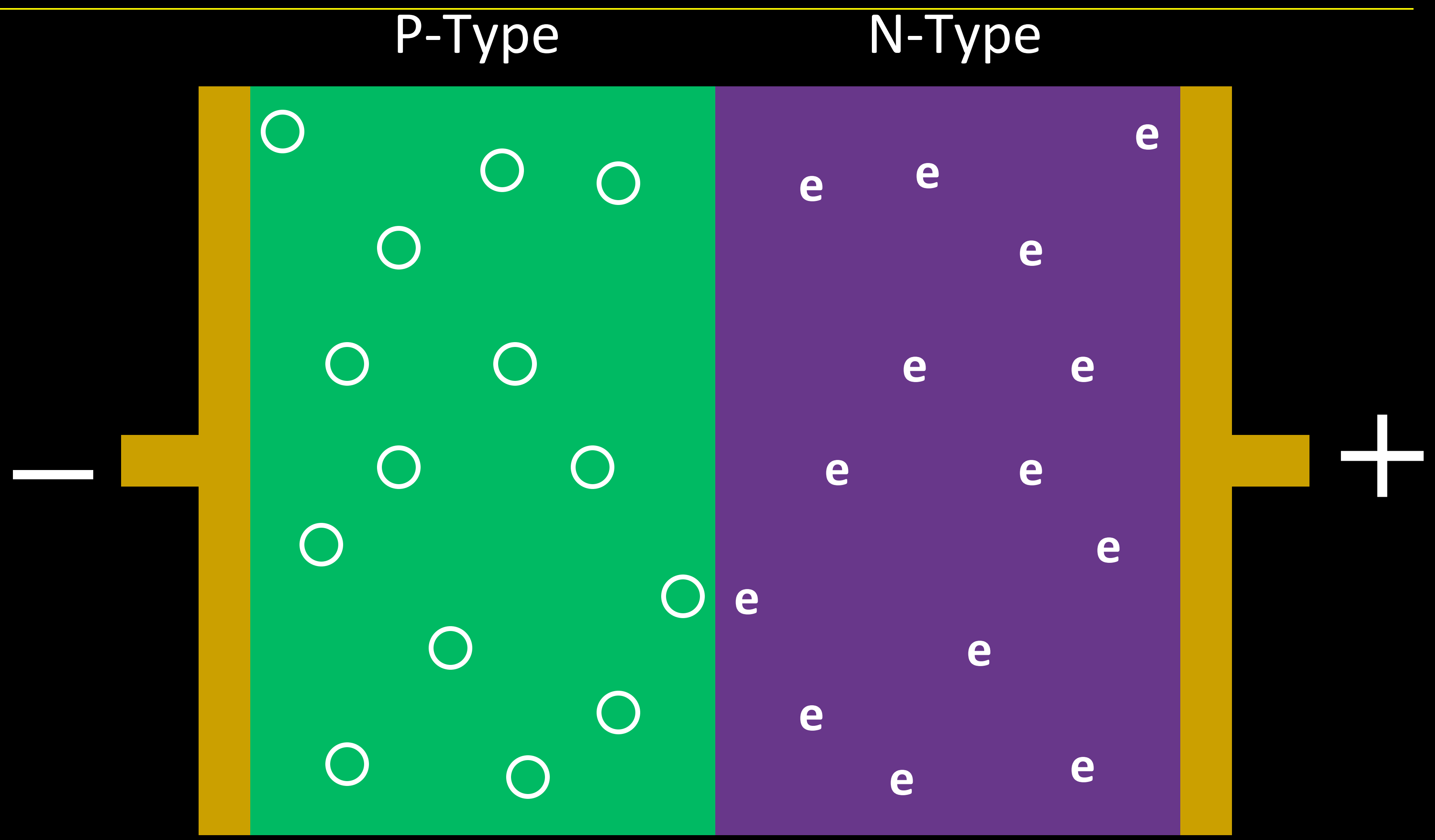
Bipolar Junction



low v → insulator
high v → conductor

low v → conductor
high v → insulator

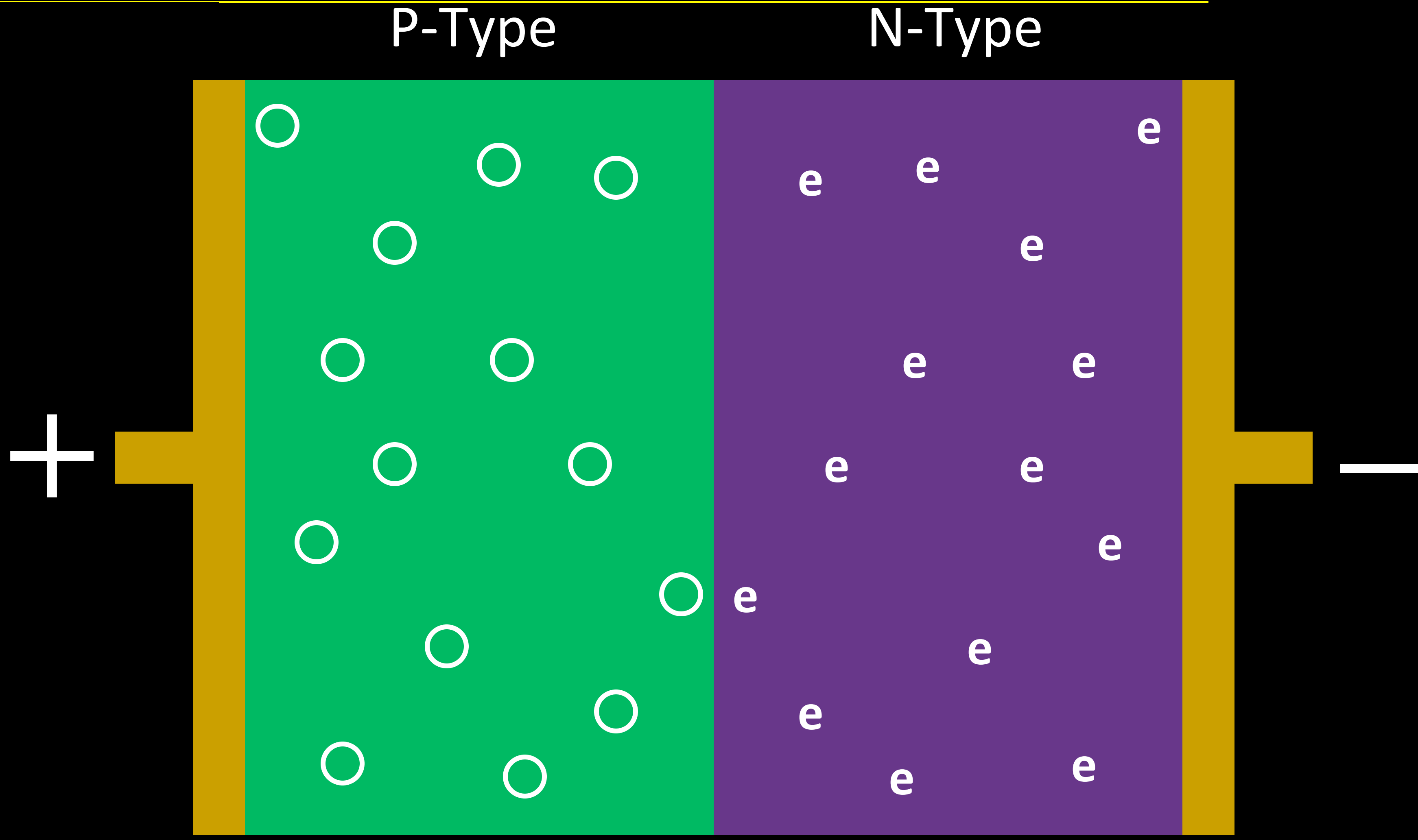
Reverse Bias



low v → insulator
high v → conductor

low v → conductor
high v → insulator

Forward Bias

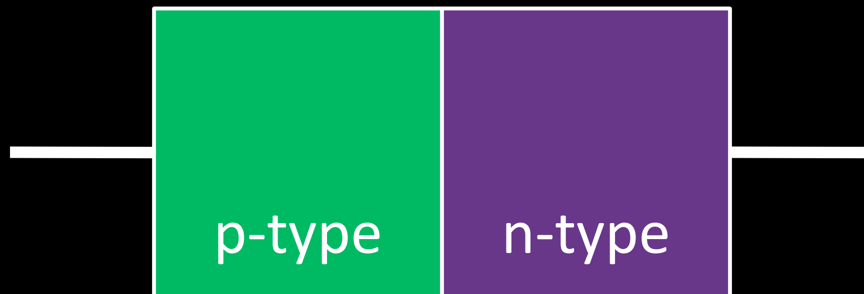


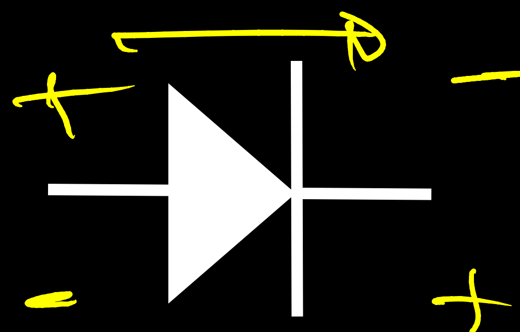
low v → insulator
high v → conductor

low v → conductor
high v → insulator

Diodes

PN Junction "Diode"



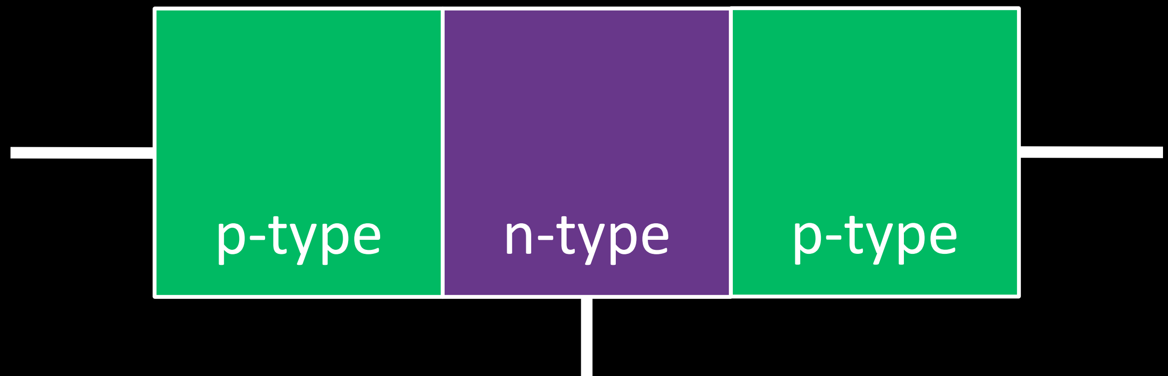


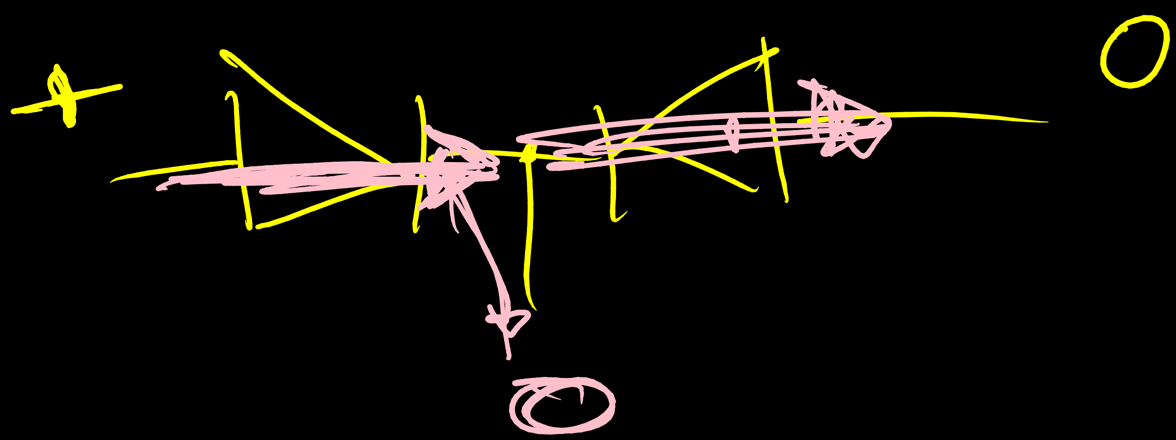
Conventions:

vdd = vcc = +1.2v = +5v = hi

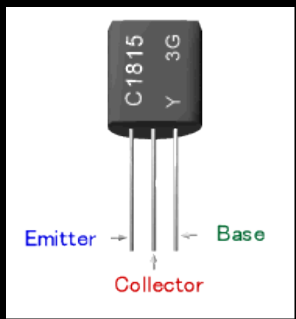
vss = vee = 0v = gnd

PNP Junction





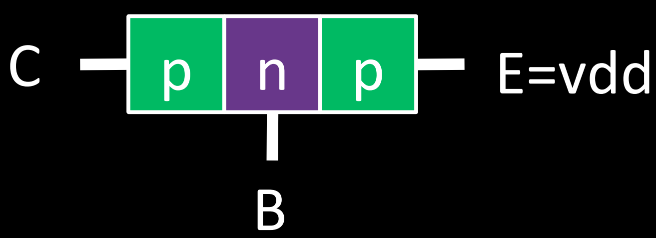
Bipolar Junction Transistors



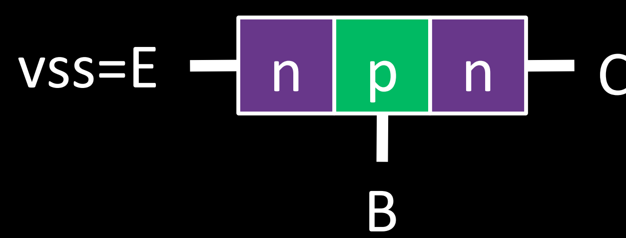
- Solid-state switch: The most amazing invention of the 1900s

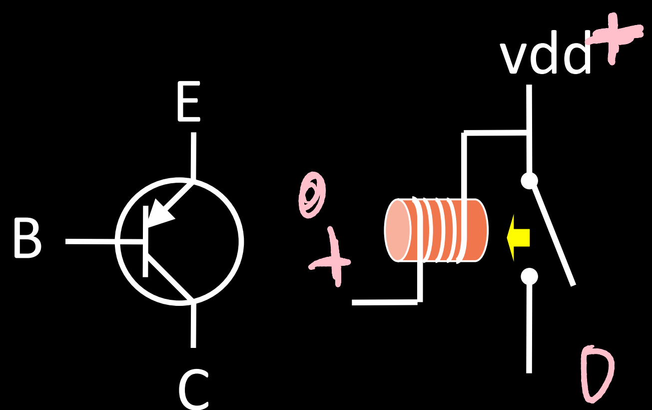
Emitter = “input”, Base = “switch”, Collector = “output”

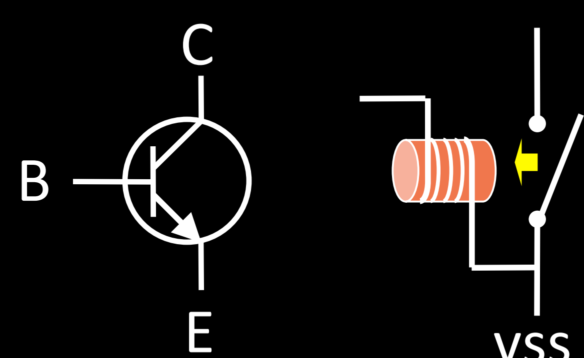
PNP Transistor



NPN Transistor

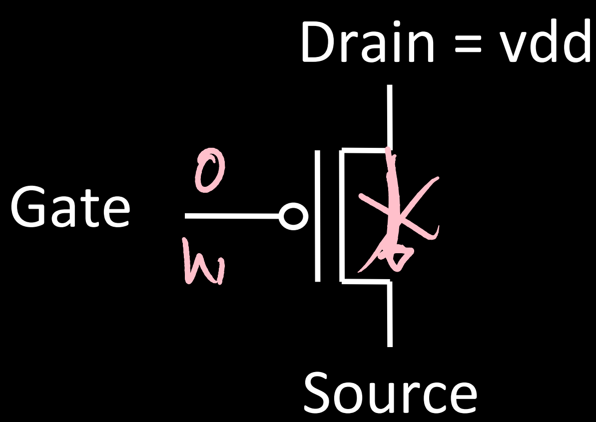




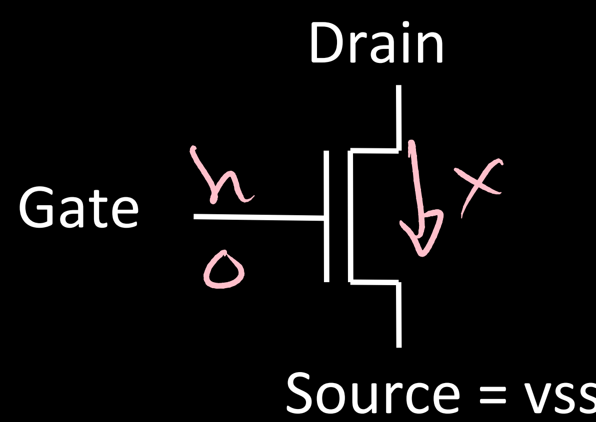


Field Effect Transistors

P-type FET



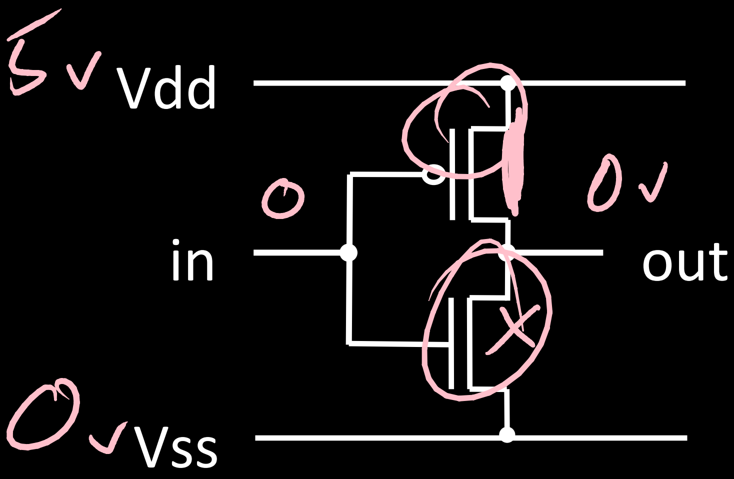
N-type FET

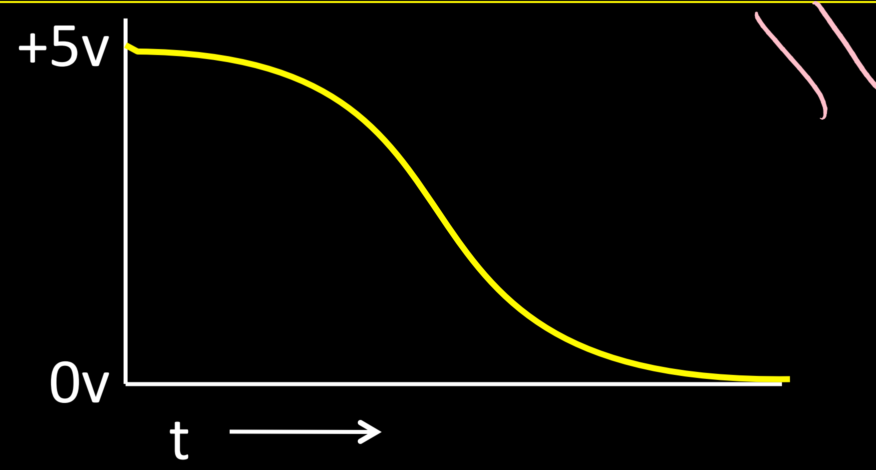


- Connect Source to Drain when Gate = lo
- Drain must be vdd, or connected to source of another P-type transistor

- Connect Source to Drain when Gate = hi
- Source must be vss, or connected to drain of another N-type transistor

Multiple Transistors





In	Out
5v	0v
0v	5v

voltage

Gate delay

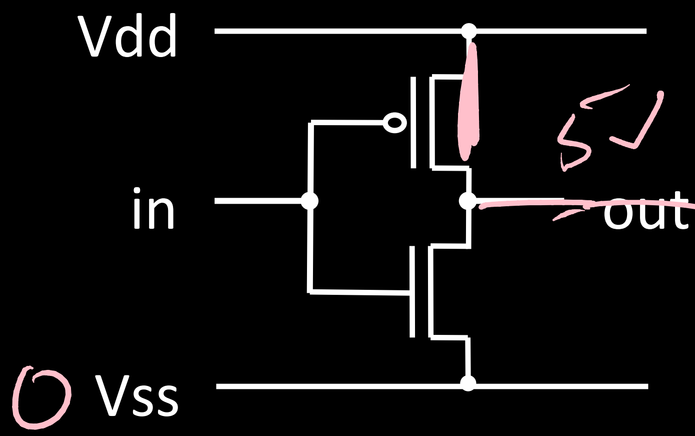
- transistor switching time
- voltage, propagation, fanout, temperature, …

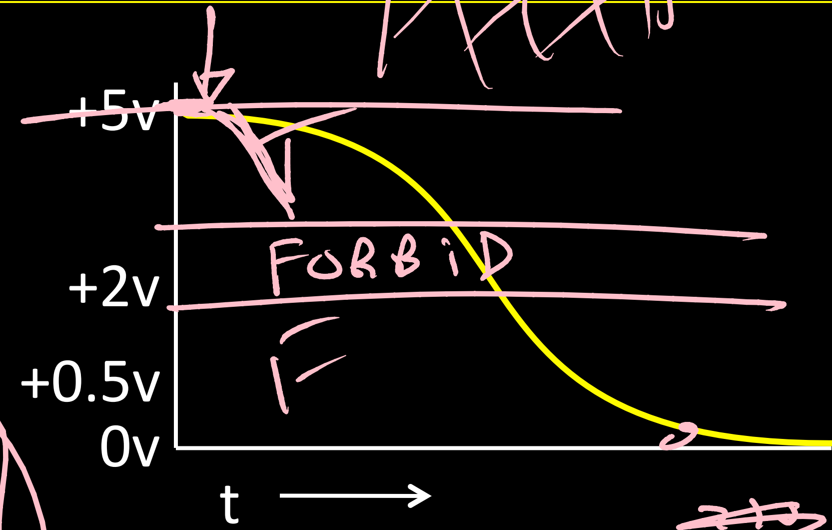
CMOS design

(complementary-symmetry metal–oxide–semiconductor)

- Power consumption = dynamic + leakage

Digital Logic





In	Out
+5v	0v
0v	+5v

voltage

In	Out
T	F
F	T

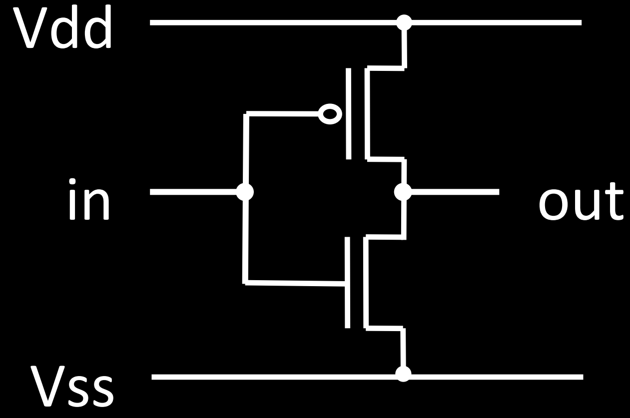
truth table

Conventions:

vdd = vcc = +1.2v = +5v = hi = true = 1

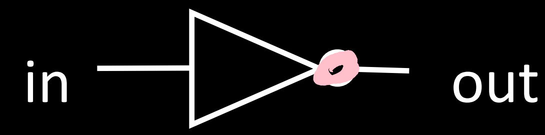
vss = vee = 0v = gnd = false = 0

NOT Gate (Inverter)



Function: NOT

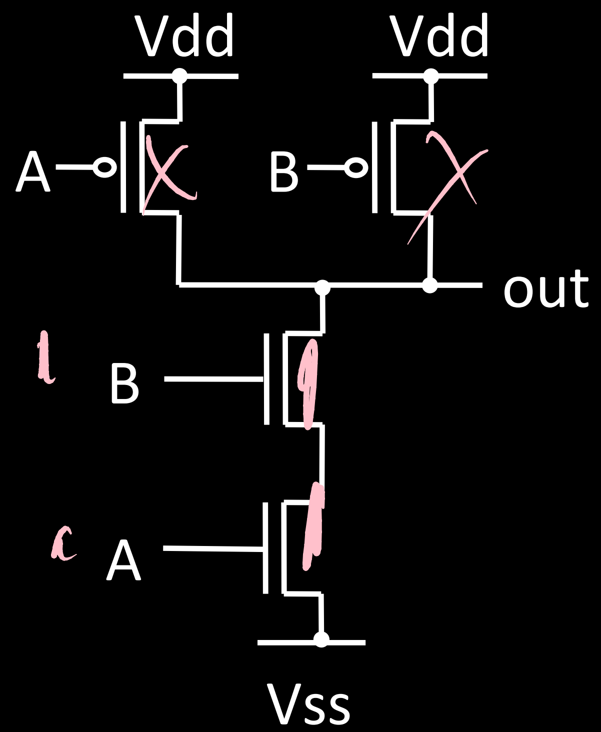
- Symbol:



In	Out
0	**1**
1	**0**

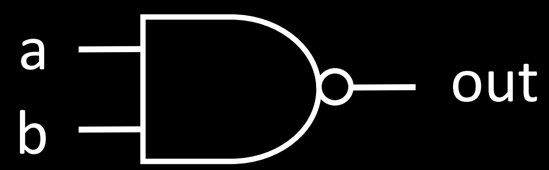
Truth table

NAND Gate



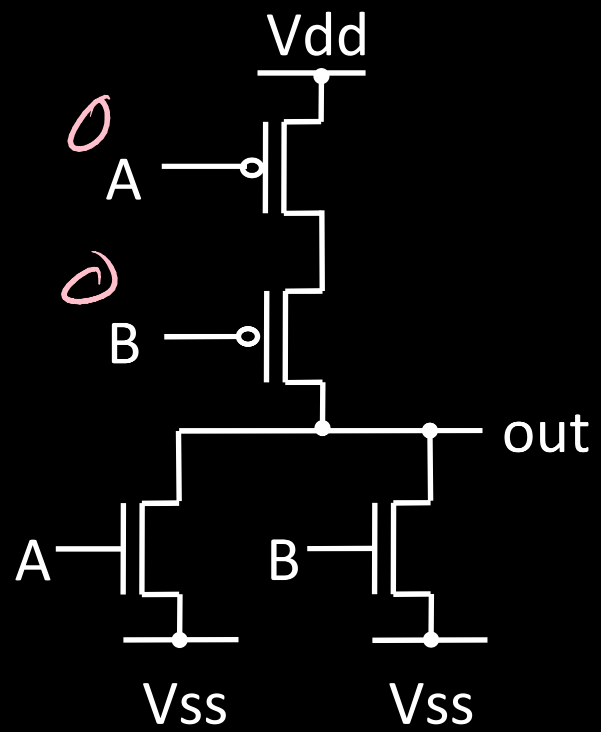
Function: NAND

- Symbol:



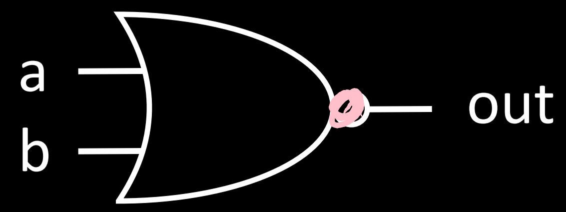
A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate



Function: NOR

- Symbol:



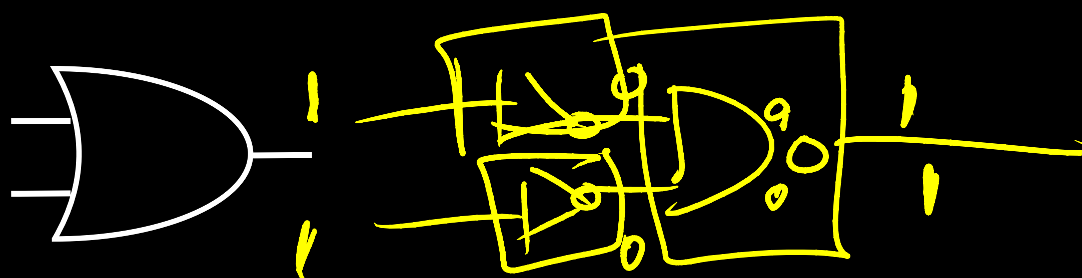
A	B	out
0	0	1
0	1	0
1	0	0
1	1	0

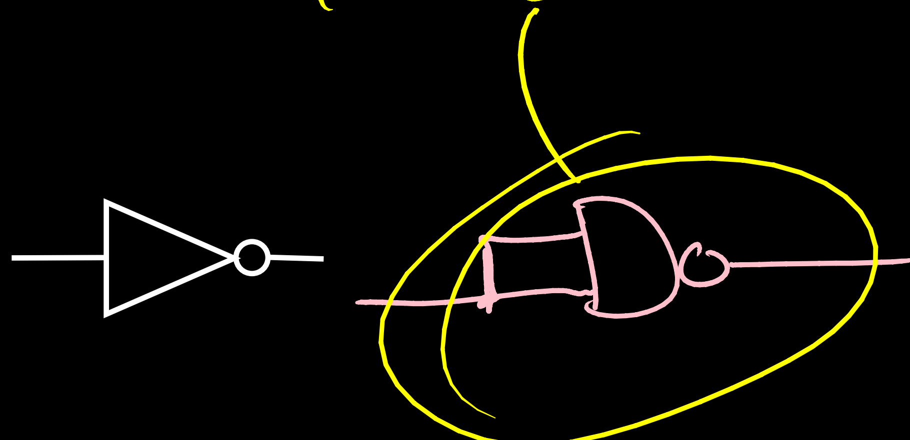
Building Functions

Not (~~A~~ and ~~B~~) = (~~Not~~ ~~A~~) or (~~Not~~ ~~B~~)

Not A Not B Not A Not B

- AND: 

- OR: 

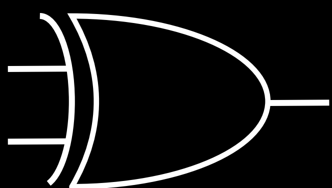
- NOT: 

Universal Gates

NAND is universal (so is NOR)

- Can implement any function with just NAND gates
  - De Morgan's laws are helpful (pushing bubbles)
- useful for manufacturing

E.g.: XOR (A, B) = A or B but not both ("exclusive or")



Proof: ?

Logic Equations

Some notation:

- constants: true = 1, false = 0
- variables: a, b, out, …
- operators:

  1 1
  0 ∅

  - AND(a, b) = a b = a & b = a ∧ b
  - OR(a, b) = a + b = a | b = a ∨ b
  - NOT(a) = ā = !a = ¬a

$0\,0 = 0$

$0 \cdot 1 = 0$

$1 \cdot 0 = 0$

$1 \cdot 1 = 1$

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 1$

Identities

Identities useful for manipulating logic equations

- For optimization & ease of implementation

$a + 0 = a$

$a + 1 = 1$

$a + \bar{a} = 1$

$a\,0 = 0$

$a\,1 = a$

$a\,\bar{a} = 0$

$\overline{(a + b)} = \bar{a}\,\bar{b}$

$\overline{(a\,b)} = \bar{a} + \bar{b}$

$a + a\,b = a$

$a(b+c) = ab + ac$

$\overline{a(b+c)} = \bar{a} + \overline{bc}$

Logic Manipulation

- functions: gates ↔ truth tables ↔ equations
- Example: (a+b)(a+c) = a + bc

a	b	c					
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

Logic Manipulation

- functions: gates ↔ truth tables ↔ equations
- Example: (a+b)(a+c) = a + bc

a	b	c	a+b	a+c	LHS	bc	RHS
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

$$(a+b)(a+c)$$

$$\underbrace{a \cdot a}_{a \cdot 1} + a \cdot c + b \cdot a + b \cdot c$$

$$a(\underbrace{\underbrace{1 + c}_{1} + b}_{1}) + b \cdot c$$

$$\underbrace{a \cdot 1}_{a} + bc$$

$$a + bc$$

Logic Minimization

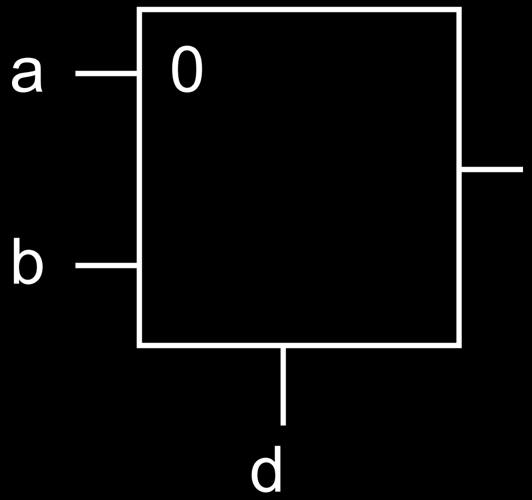
- A common problem is how to implement a desired function most efficiently
- One can derive the equation from the truth table

a	b	c	minterm
0	0	0	$\overline{abc}$
0	0	1	$\overline{ab}c$
0	1	0	$\bar{a}b\bar{c}$
0	1	1	$\bar{a}bc$
1	0	0	$a\overline{bc}$
1	0	1	$a\bar{b}c$
1	1	0	$ab\bar{c}$
1	1	1	abc

for all outputs
that are 1,
take the corresponding
minterm
Obtain the result in
"sum of products" form

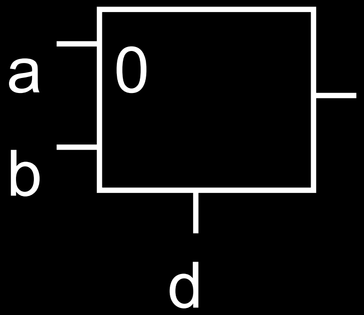
- How does one find the most efficient equation?
  - Manipulate algebraically until satisfied
  - Use Karnaugh maps (or K maps)

Multiplexer



- A multiplexer selects between multiple inputs
  - out = a, if d = 0
  - out = b, if d = 1
- Build truth table
- Minimize diagram
- Derive logic diagram

Multiplexer Implementation



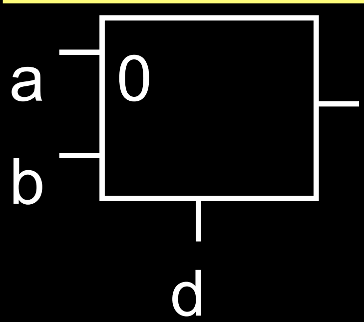
a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Build a truth table

$$= abd + ab\bar{d} + \bar{a}bd + a\bar{b}\bar{d}$$

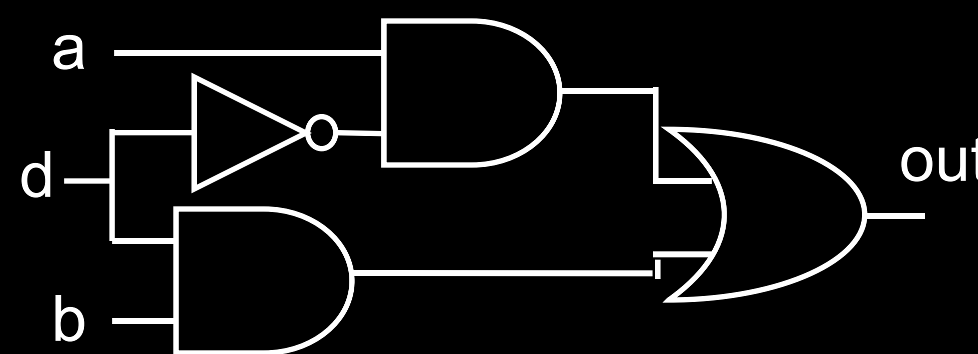
$$= a\bar{d} + bd$$

Multiplexer Implementation

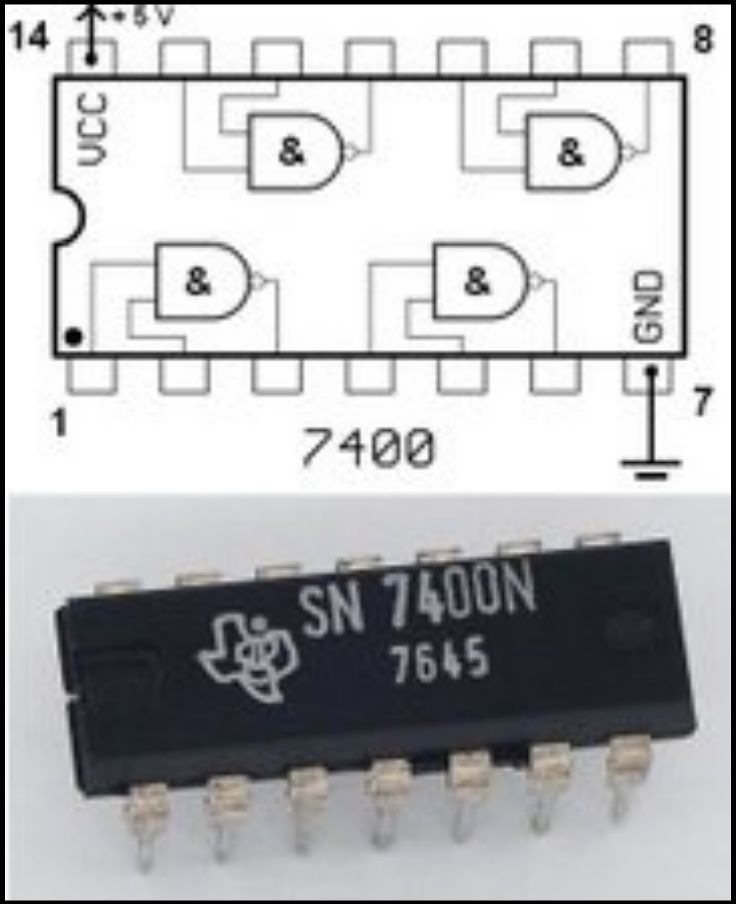


a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Draw the circuit
  - out = $a\overline{d} + bd$

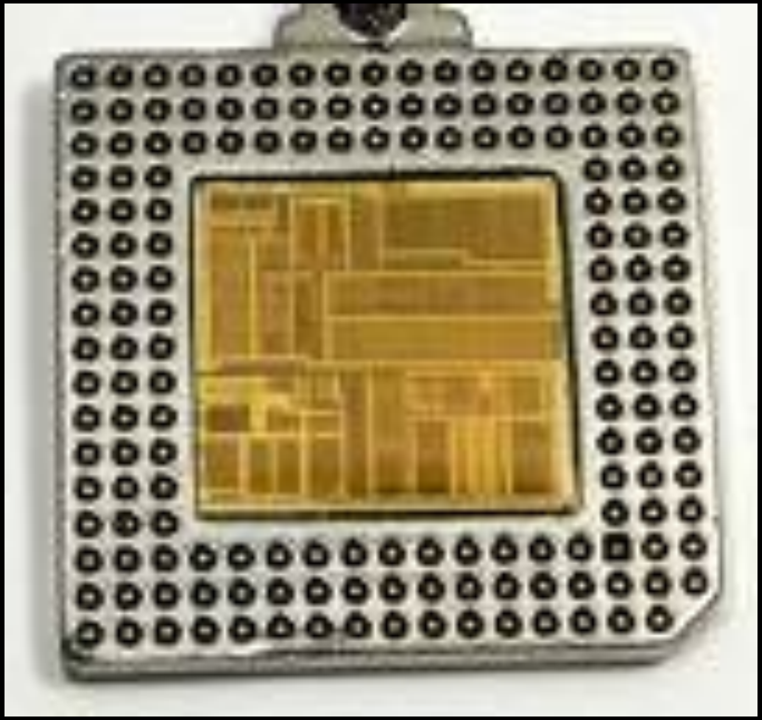


Logic Gates



- One can buy gates separately
  - ex. 74xxx series of integrated circuits
  - cost ~$1 per chip, mostly for packaging and testing
- Cumbersome, but possible to build devices using gates put together manually

Integrated Circuits



- Or one can manufacture a complete design using a custom mask
- Intel Nehalem has approximately 731 million transistors

Voting machine

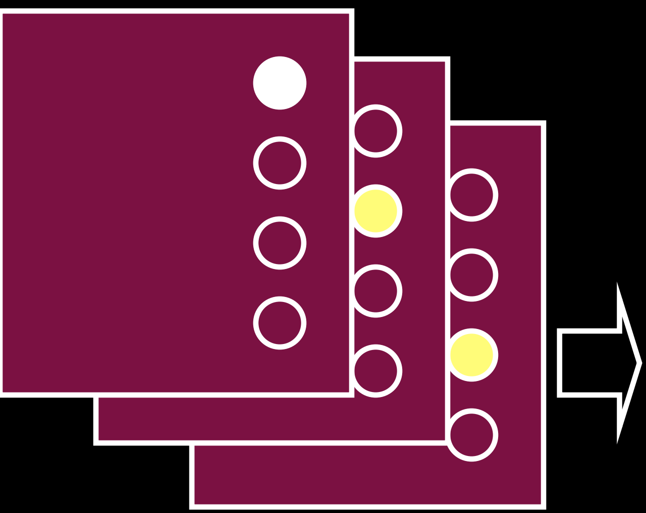
- Build something interesting
- A voting machine
- Assume:
  - A vote is recorded on a piece of paper,
  - by punching out a hole,
  - there are at most 7 choices
  - we will not worry about “hanging chads” or “invalids”

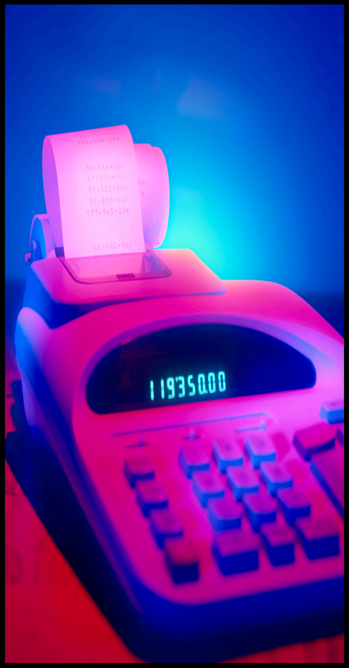
Voting machine

- For now, let's just display the numerical identifier to the ballot supervisor
  - we won't do counting yet, just decoding
  - we can use four photo-sensitive transistors to find out which hole is punched out

- A photo-sensitive transistor detects the presence of light
- Photo-sensitive material triggers the gate

Ballot Reading





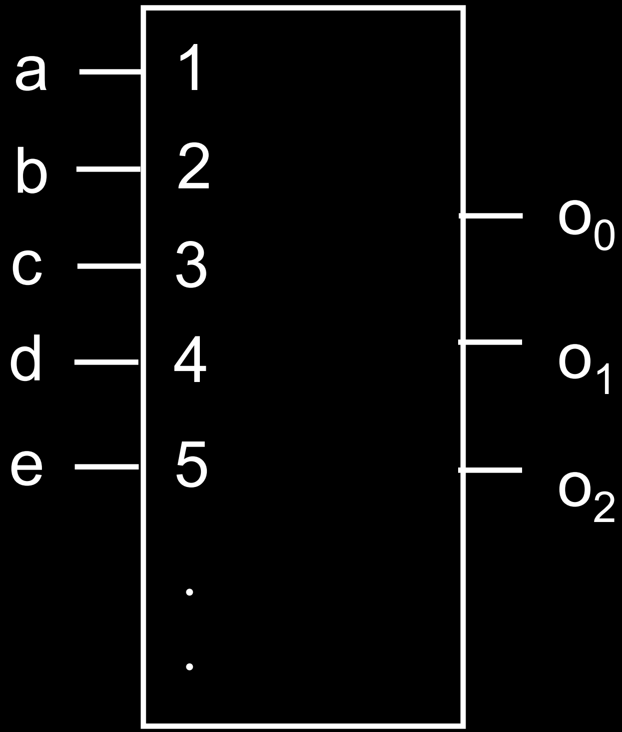
– Input: paper with a hole in it

– Out: number the ballot supervisor can record

Ballots

The 3410 vote recording machine

Encoders



A 3-bit encoder
(7-to-3)
(5 inputs shown)

- N sensors in a row
- Want to distinguish which of the N sensors has fired
- Want to represent the firing sensor number in compact form
  - N might be large
  - Only one wire is on at any time
  - Silly to route N wires everywhere, better to encode in log N wires

Number Representations

$$\underline{3}\;\underline{7}$$

10^1 10^0

- Decimal numbers are written in base 10
  - $3 \times 10^1 + 7 \times 10^0 = 37$
- Just as easily use other bases
  - Base 2 - "Binary"
  - Base 8 - "Octal"
  - Base 16 – "Hexadecimal"

Number Representations

3 7

10^1 10^0

- Base conversion via repetitive division
  - Divide by base,
    write remainder,
    move left with quotient
  - Sanity check with 37 and base 10

Binary Representation

- Check 37 and base 2
- 37 = 32 + 4 + 1

0 1 0 0 1 0 1

2^6 2^5 2^4 2^3 2^2 2^1 2^0

64 32 16 8 4 2 1

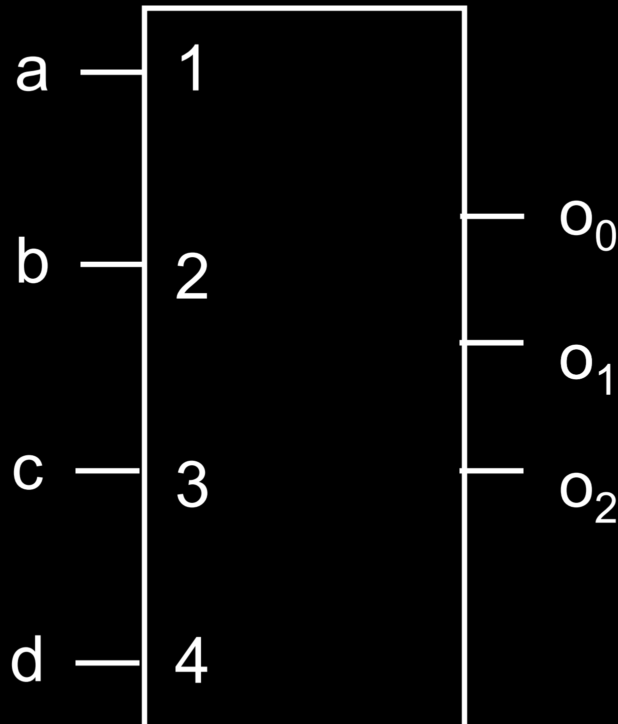
Hexadecimal Representation

$$\underline{2}\;\underline{5}$$

$$16^1 \quad 16^0$$

- 37 decimal = $(25)_{16}$
- Convention
  - Base 16 is written with a leading 0x
  - 37 = 0x25
- Need extra digits!
  - 0, 1, 2, 3, 4, 5, 6, 7,
    8, 9, A, B, C, D, E, F
- Binary to hexadecimal is easy
  - Divide into groups of 4, translate groupwise into hex digits

Encoder Truth Table



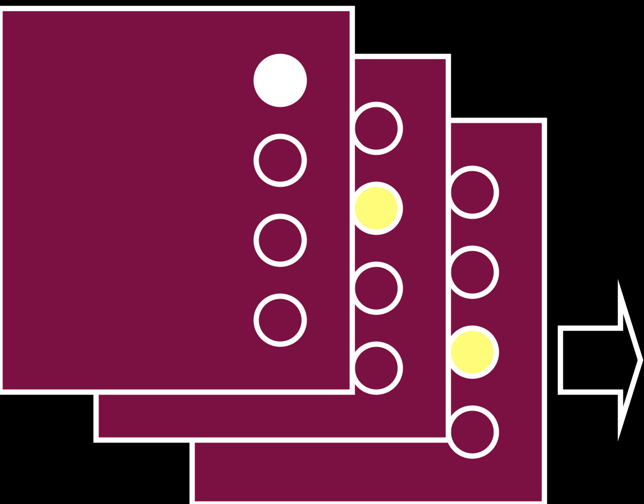
A 3-bit encoder

with 4 inputs for simplicity

a	b	c	d		o2	o1	o0
0	0	0	0		0	0	0
1	0	0	0		0	0	1
0	1	0	0		0	1	0
0	0	1	0		0	1	1
0	0	0	1		1	0	0

- $o2 = \overline{abc}d$
- $o1 = \bar{a}b\bar{c}d + \bar{a}bc\bar{d}$
- $o0 = a\overline{bcd} + \bar{a}bc\bar{d}$

Ballot Reading

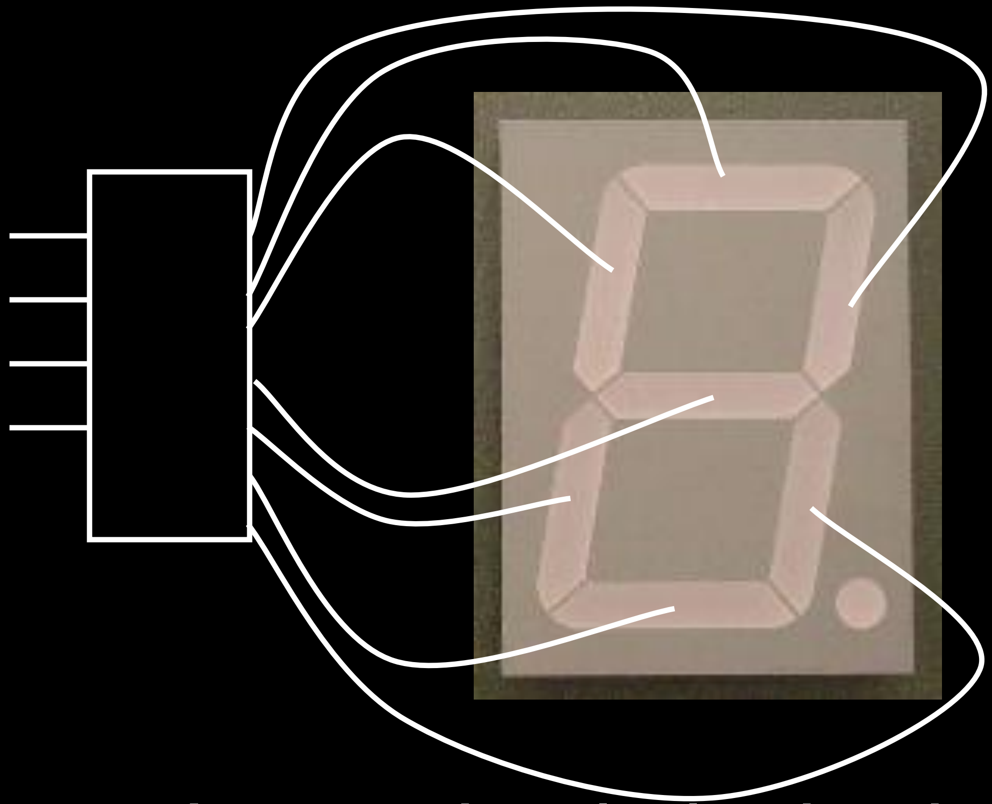


Ballots

The 3410 voting machine

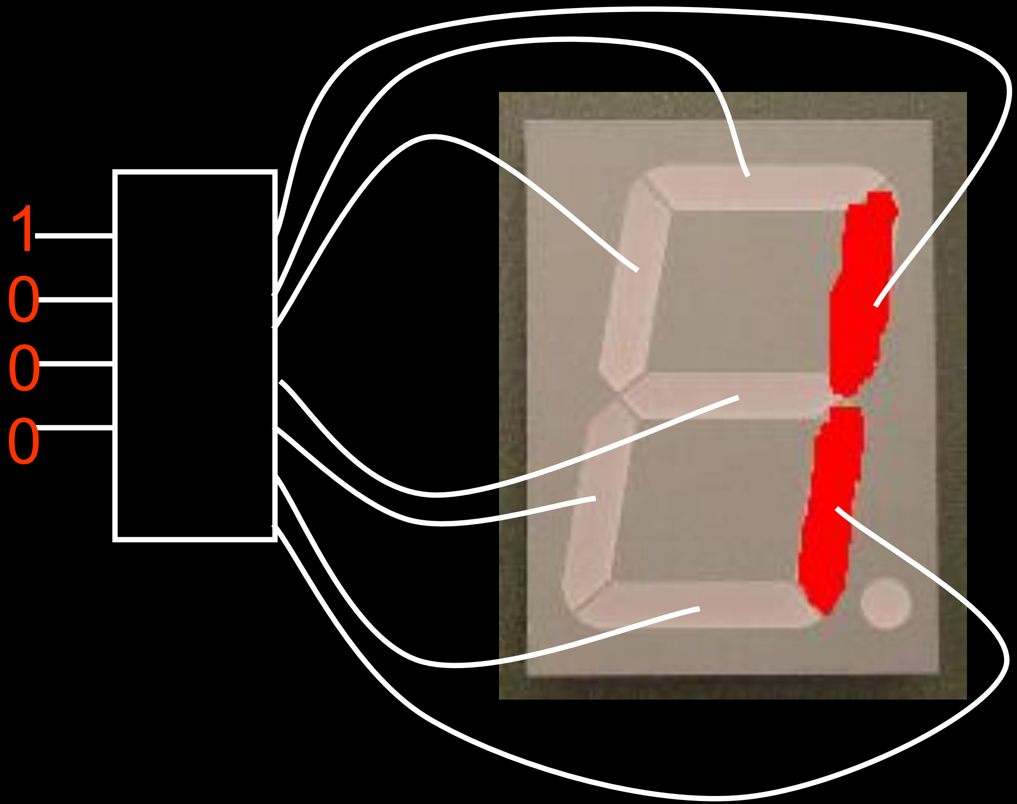
- Ok, we built first half of the machine
- Need to display the result

7-Segment LED Decoder



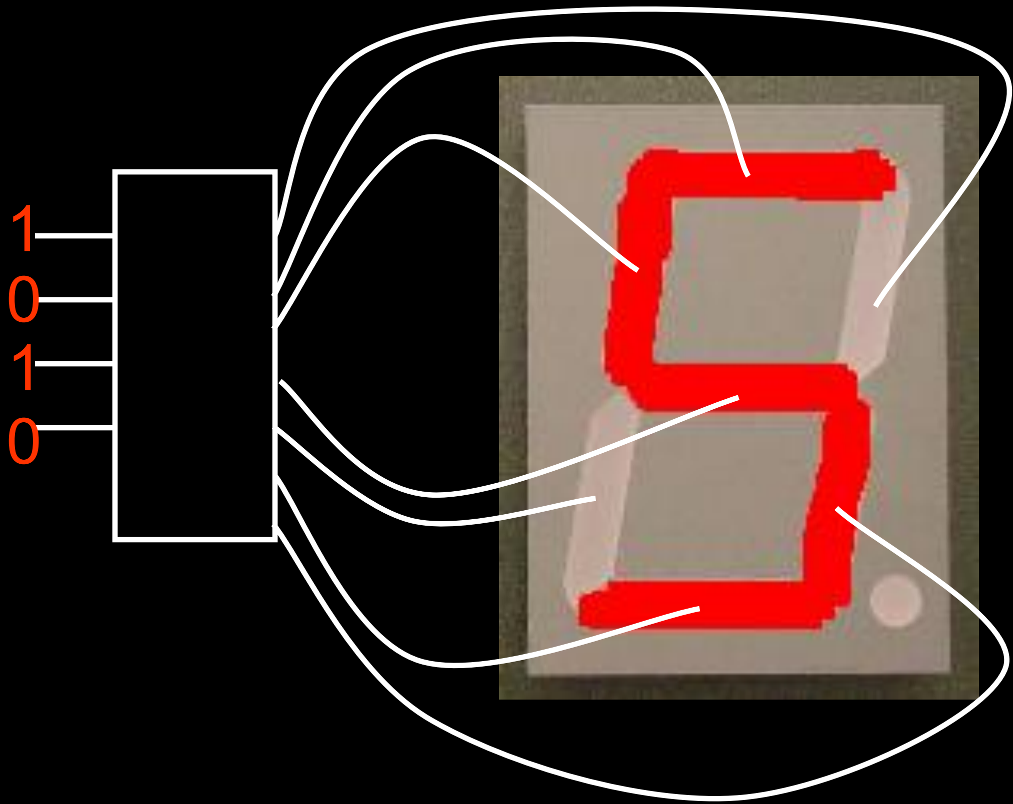
- 4 inputs encoded in binary
- 8 outputs, each driving an independent, rectangular LED
- Can display numbers
- Just a simple logic circuit
- Write the truth table

7-Segment LED Decoder



- 4 inputs encoded in binary
- 8 outputs, each driving an independent, rectangular LED
- Can display numbers

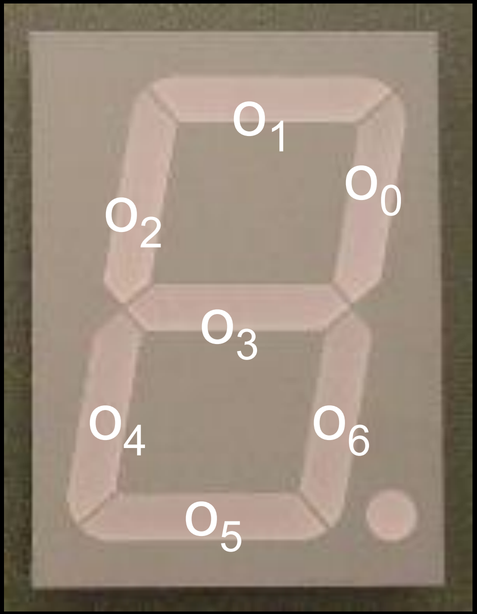
7-Segment LED Decoder



- 4 inputs encoded in binary
- 8 outputs, each driving an independent, rectangular LED
- Can display numbers

7-Segment Decoder Truth Table

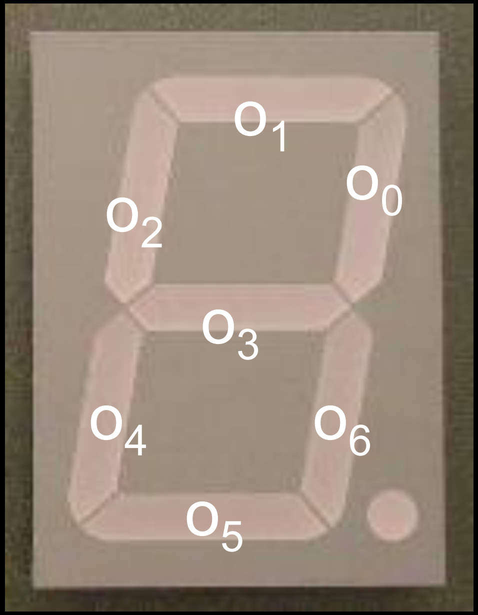
i_3	i_2	i_1	i_0		o_0	o_1	o_2	o_3	o_4	o_5	o_6
0	0	0	0		1	1	1	0	1	1	1
0	0	0	1		1	0	0	0	0	0	1
0	0	1	0		1	1	0	1	1	1	0
0	0	1	1		1	1	0	1	0	1	1
0	1	0	0		1	0	1	1	0	0	1
0	1	0	1		0	1	1	1	0	1	1
0	1	1	0		0	0	1	1	1	1	1
0	1	1	1		1	1	0	0	0	0	0
1	0	0	0		1	1	1	1	1	1	1
1	0	0	1		1	1	1	1	0	1	1



Exercise: find the error(s) in this truth table

7-Segment Decoder Truth Table

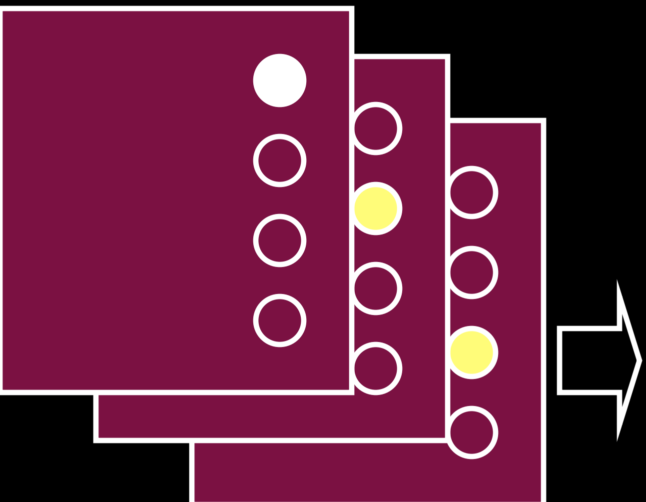
i_3	i_2	i_1	i_0		o_0	o_1	o_2	o_3	o_4	o_5	o_6
0	0	0	0		1	1	1	0	1	1	1
0	0	0	1		1	0	0	0	0	0	1
0	0	1	0		1	1	0	1	1	1	0
0	0	1	1		1	1	0	1	0	1	1
0	1	0	0		1	0	1	1	0	0	1
0	1	0	1		0	1	1	1	0	1	1
0	1	1	0		0	0	1	1	1	1	1
0	1	1	1		1	1	0	0	0	0	0
1	0	0	0		1	1	1	1	1	1	1
1	0	0	1		1	1	1	1	0	1	1



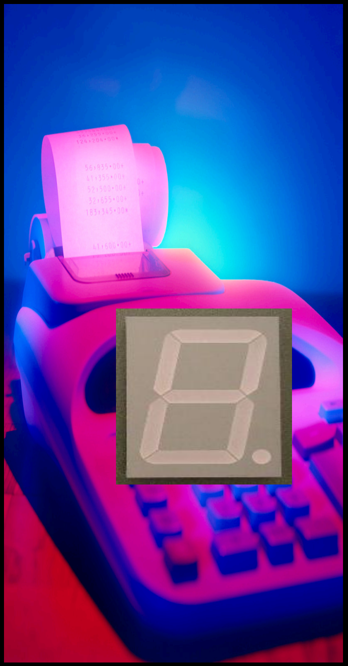
Exercise: find the error(s) in this truth table

Ballot Reading

- Done!



Ballots



The 3410 voting machine

Summary

- We can now implement any logic circuit
  - Can do it efficiently, using Karnaugh maps to find the minimal terms required
  - Can use either NAND or NOR gates to implement the logic circuit
  - Can use P- and N-transistors to implement NAND or NOR gates