

Gates and Logic

Hakim Weatherspoon

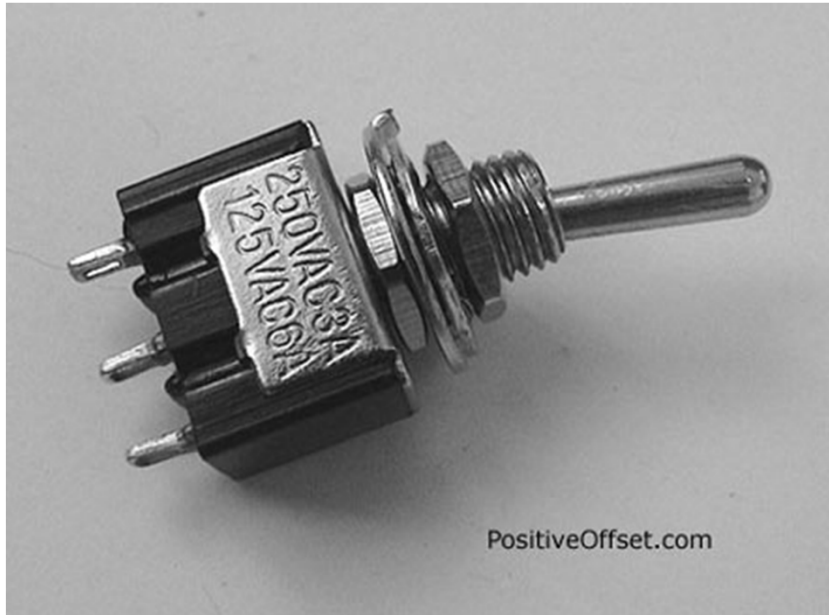
CS 3410, Spring 2012

Computer Science

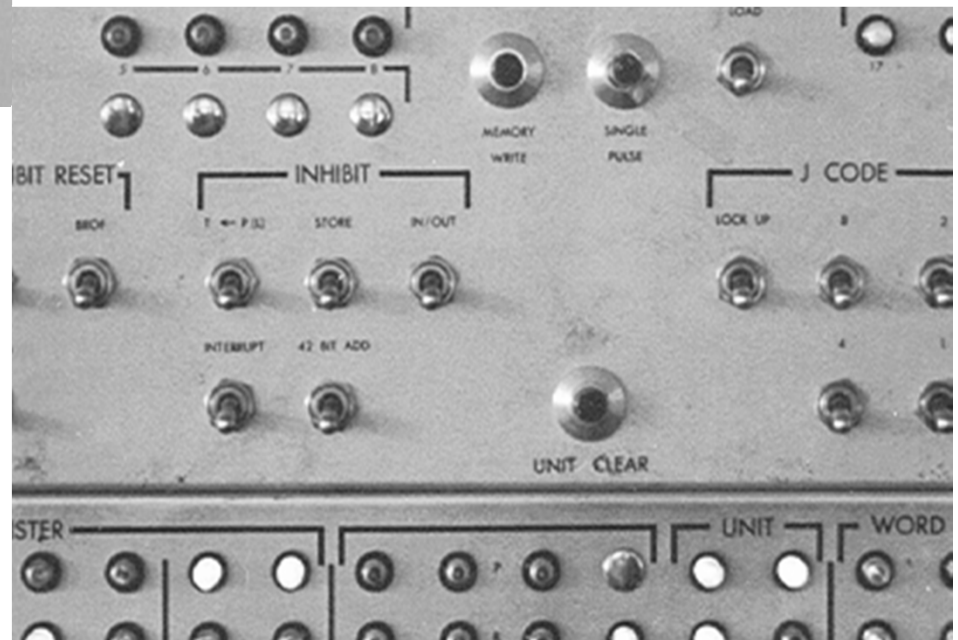
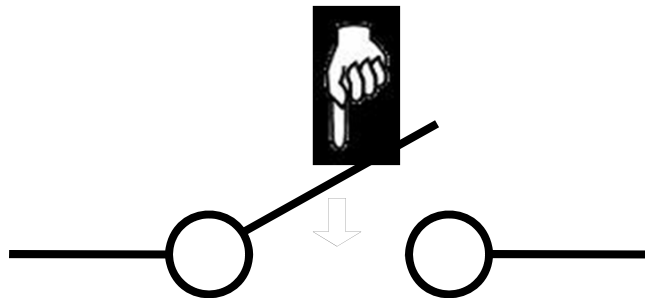
Cornell University

See: P&H Appendix C.2 and C.3 (Also, see C.0 and C.1)

A switch



- Acts as a *conductor* or *insulator*
- Can be used to build amazing things...



Goals for today

To understand how to program,
we will build a processor (i.e. a logic circuit)

Logic circuits

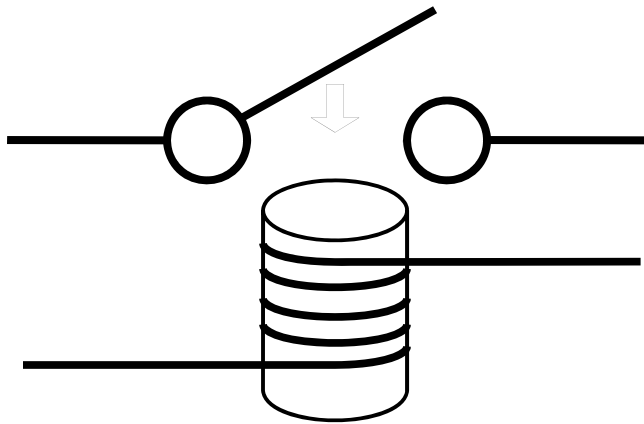
- Use P- and N-transistors to implement NAND or NOR gates
- Use NAND or NOR gates to implement the logic circuits
- Build efficient logic circuits

Better Switch

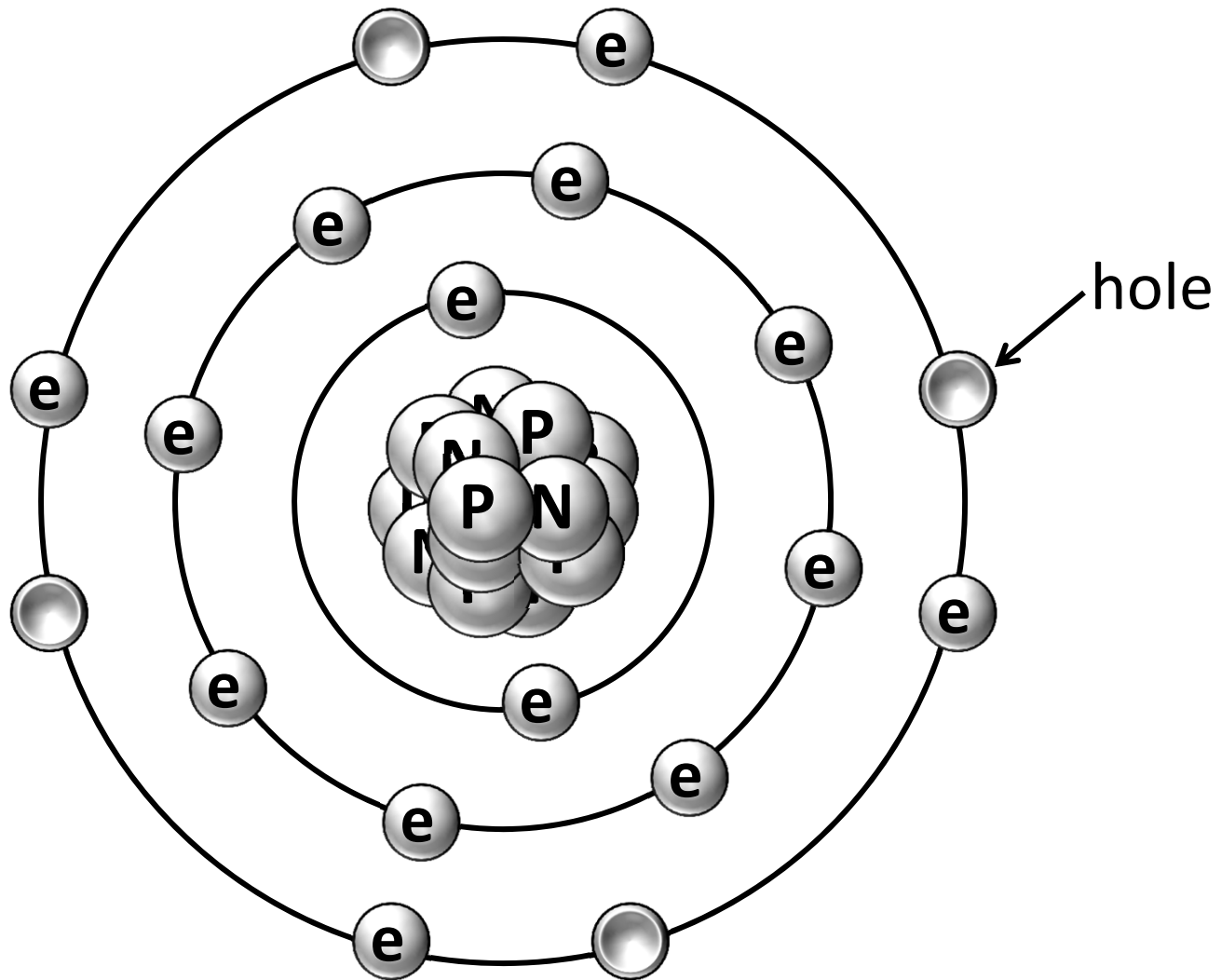


- One current controls another (larger) current

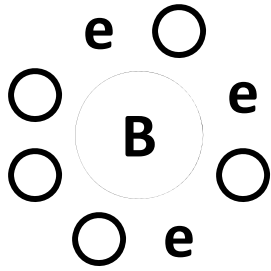
- Static Power:
 - Keeps consuming power when in the *ON* state
- Dynamic Power:
 - Jump in power consumption when switching



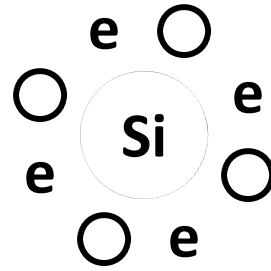
Atoms



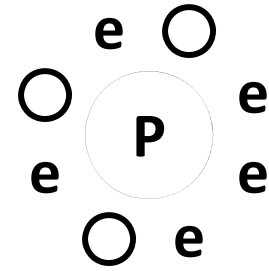
Elements



Boron

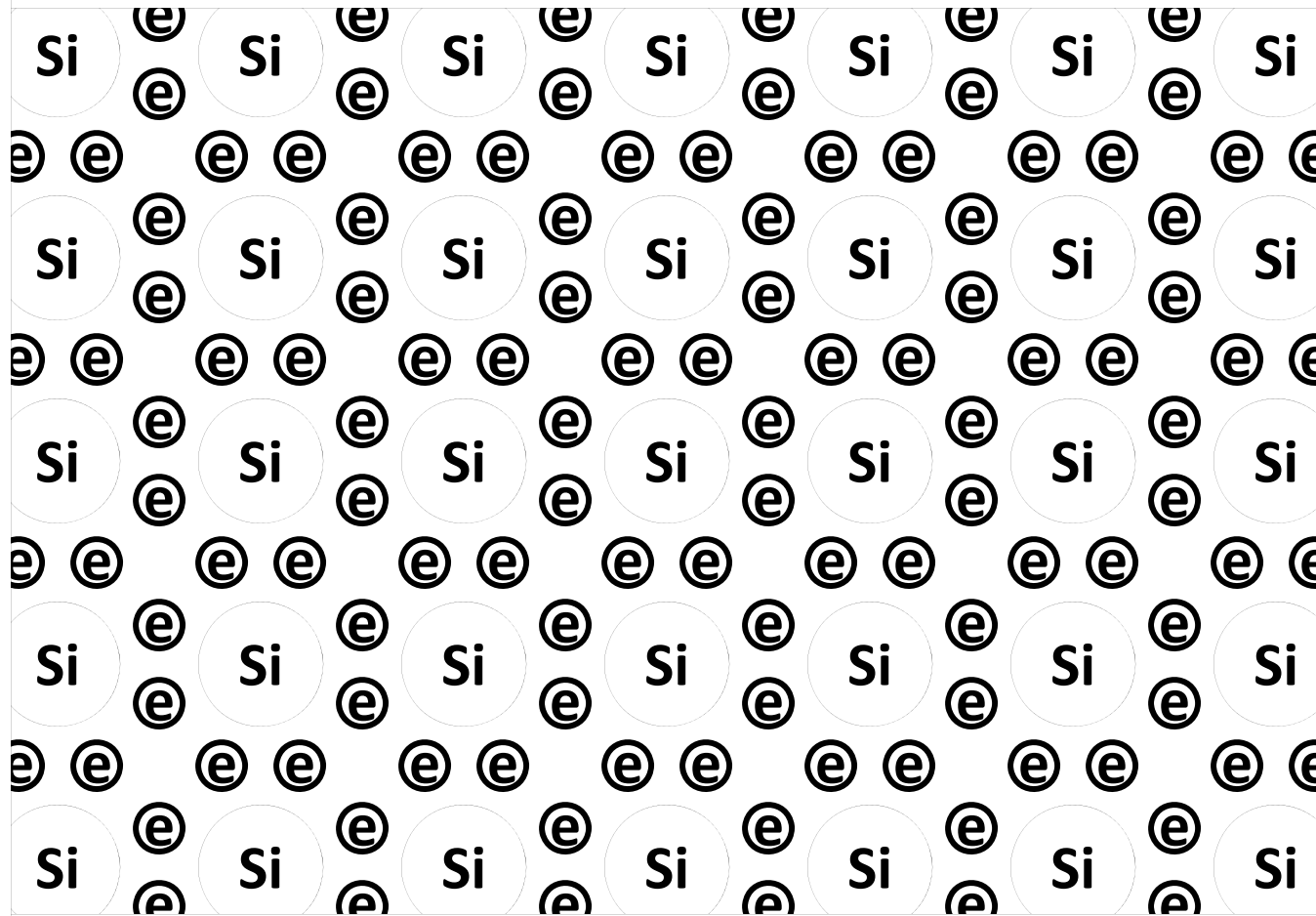


Silicon



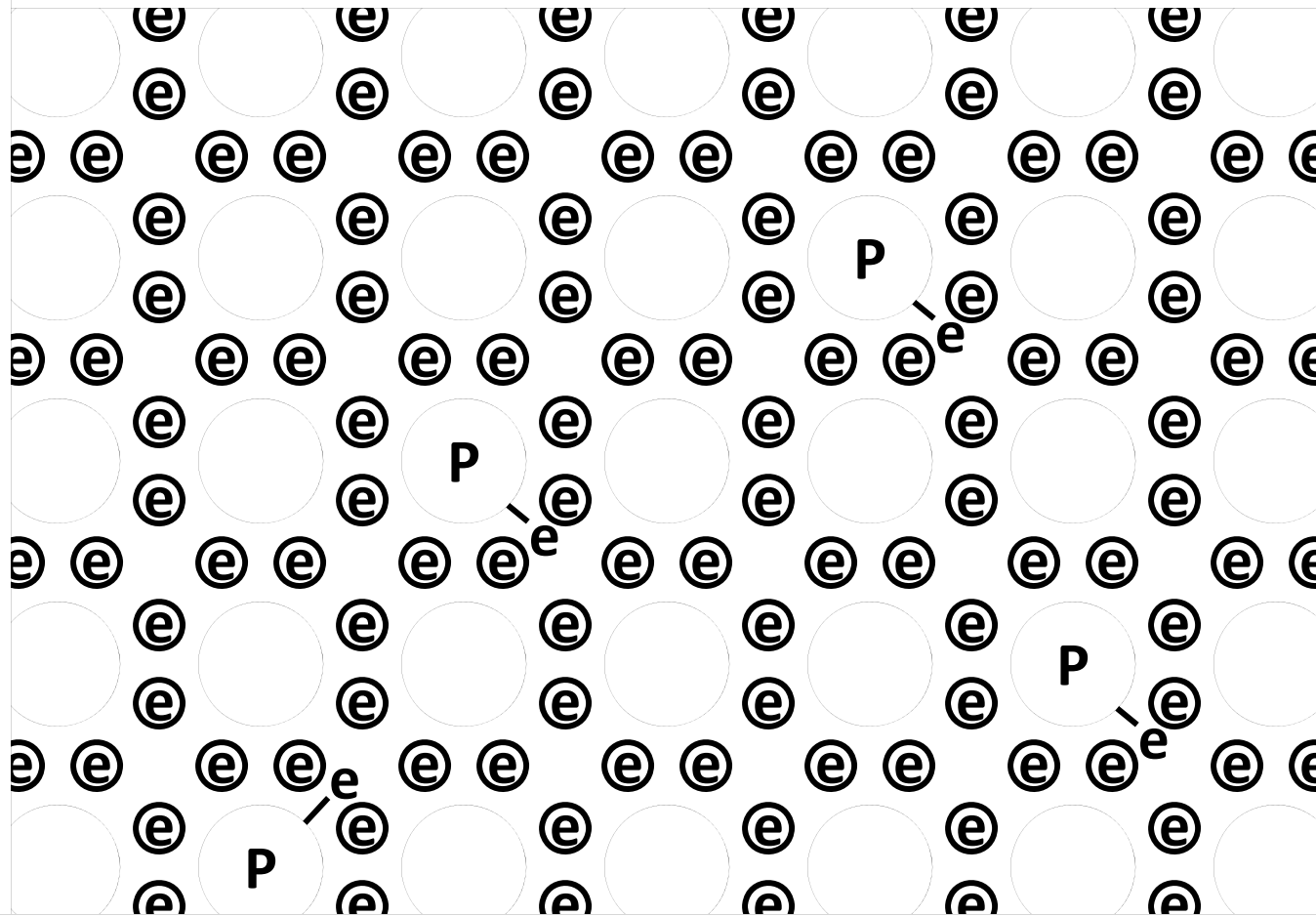
Phosphorus

Silicon Crystal Silicon



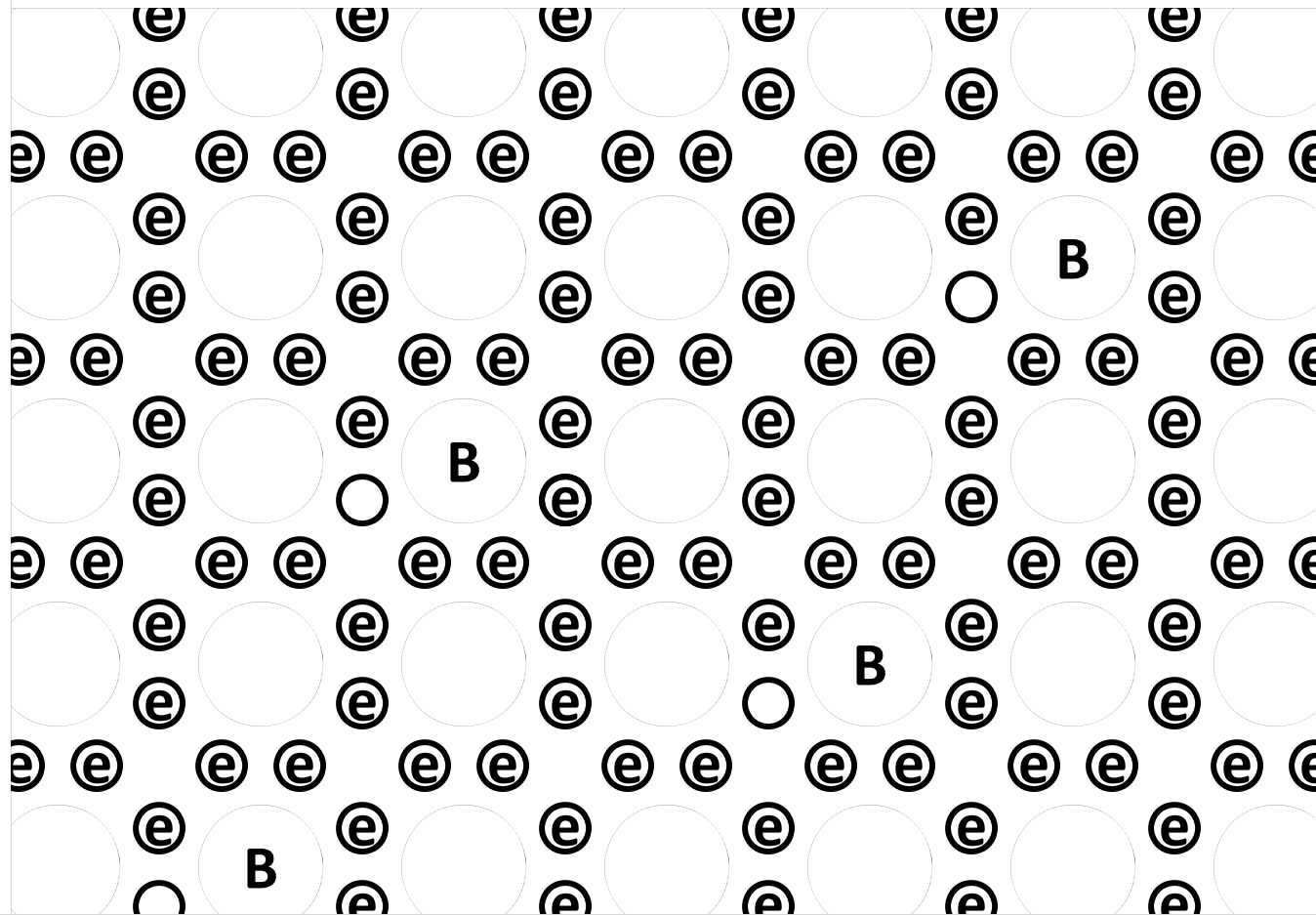
Phosphorus Doping

N-Type: Silicon + Phosphorus

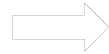
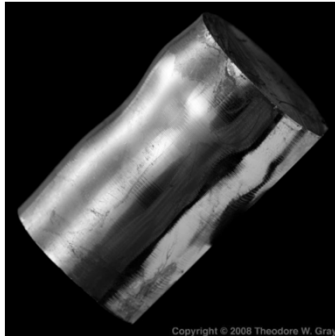


Boron Doping

P-Type: Silicon + Boron



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

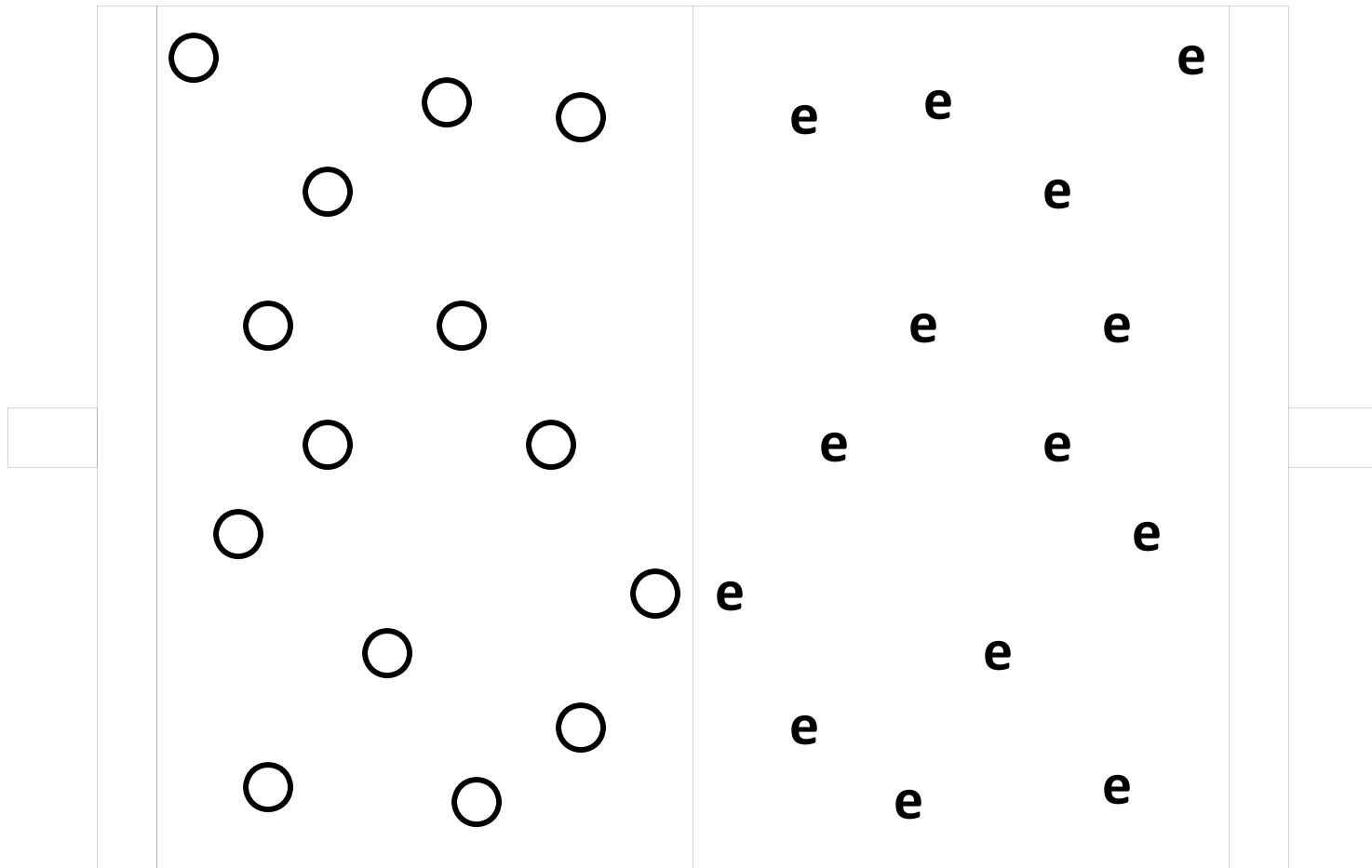
high voltage (depleted)

→ insulator

Bipolar Junction

P-Type

N-Type



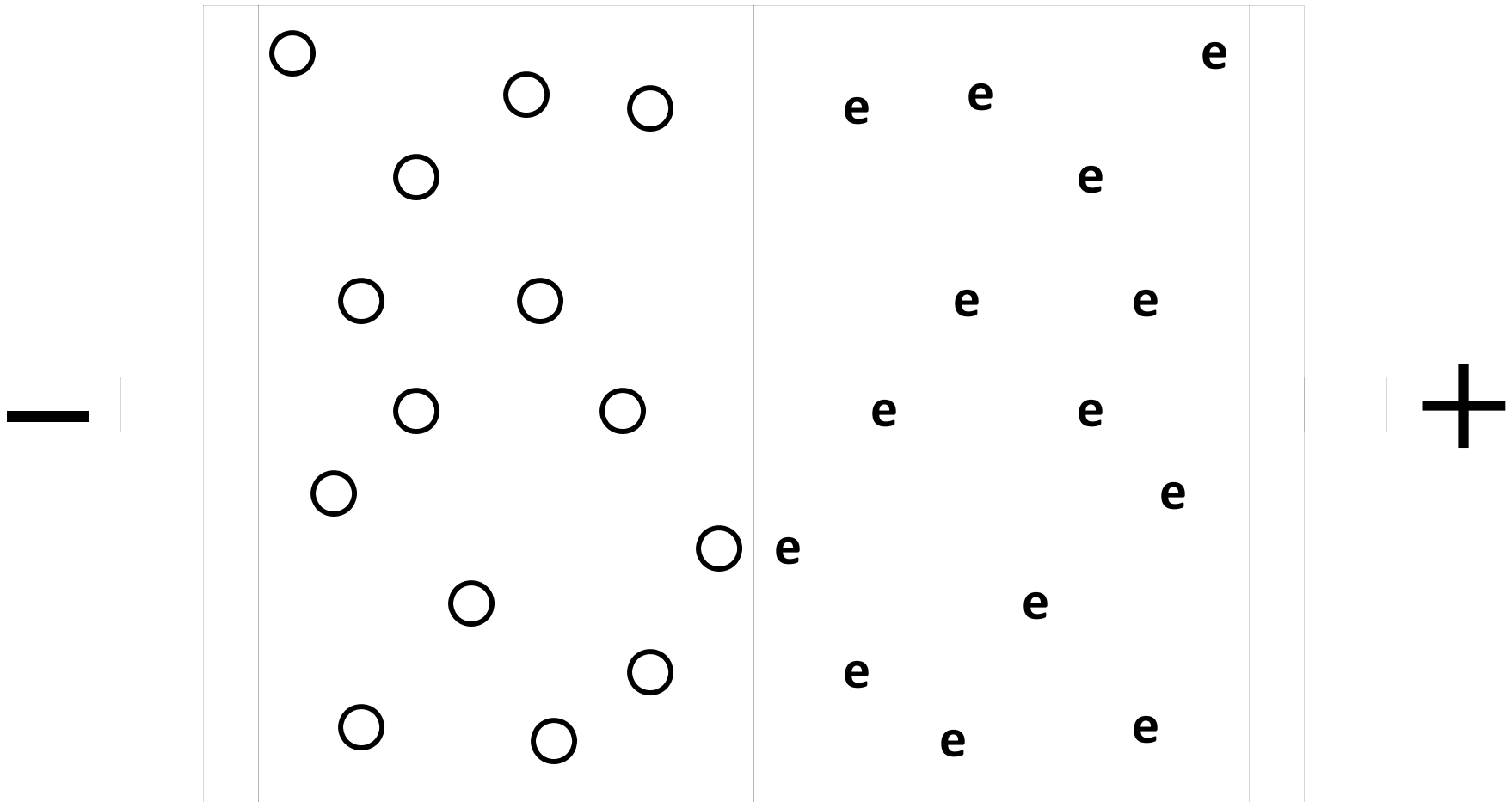
low $v \rightarrow$ insulator
high $v \rightarrow$ conductor

low $v \rightarrow$ conductor
high $v \rightarrow$ insulator

Reverse Bias

P-Type

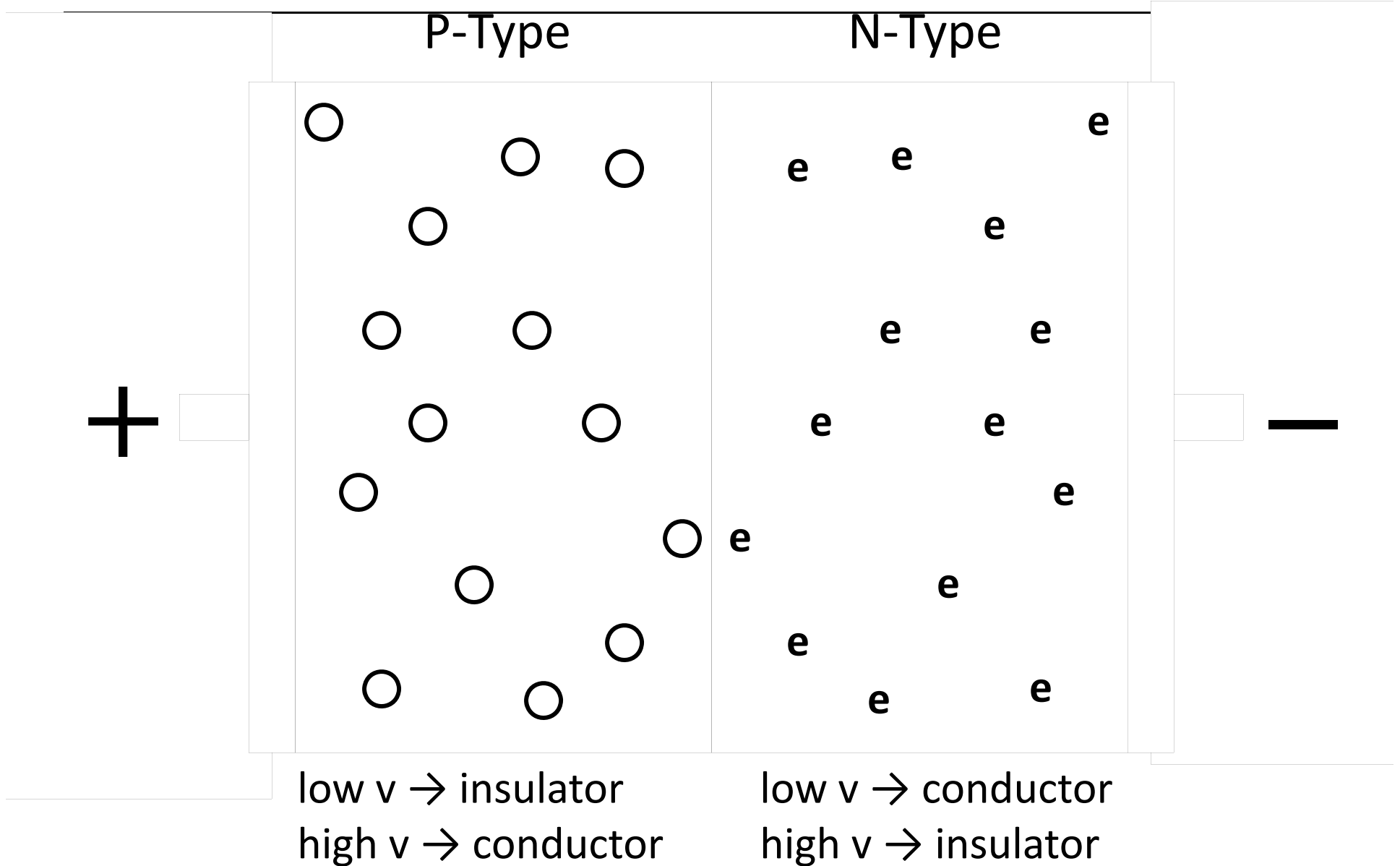
N-Type



low $v \rightarrow$ insulator
high $v \rightarrow$ conductor

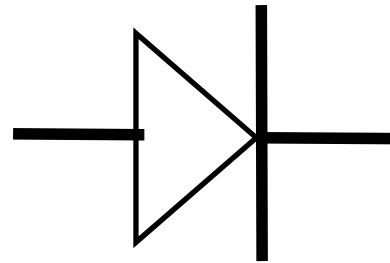
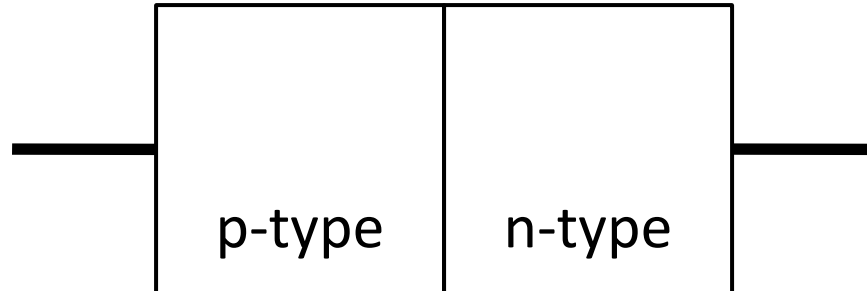
low $v \rightarrow$ conductor
high $v \rightarrow$ insulator

Forward Bias



Diodes

PN Junction "Diode"

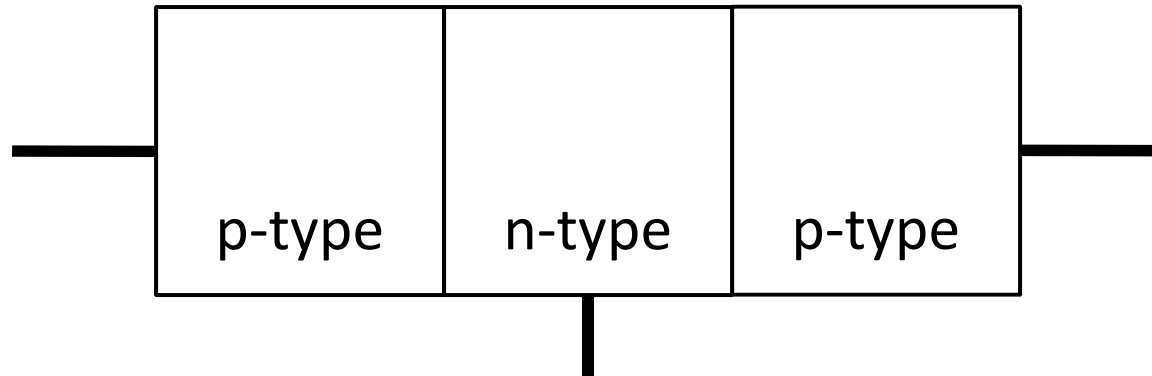


Conventions:

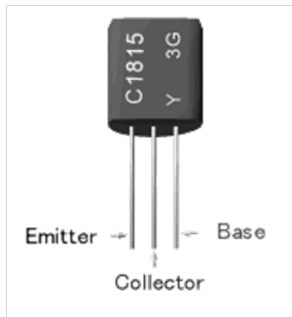
$v_{dd} = v_{cc} = +1.2v = +5v = hi$

$v_{ss} = v_{ee} = 0v = gnd$

PNP Junction



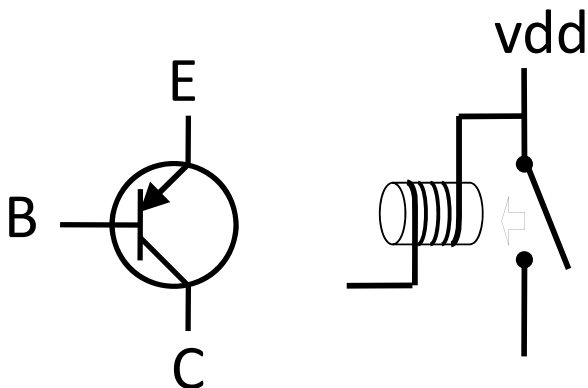
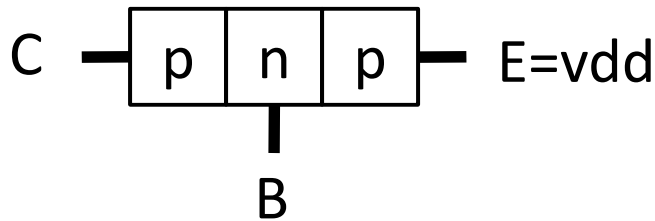
Bipolar Junction Transistors



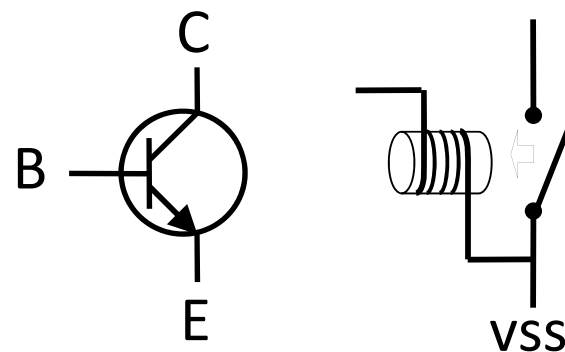
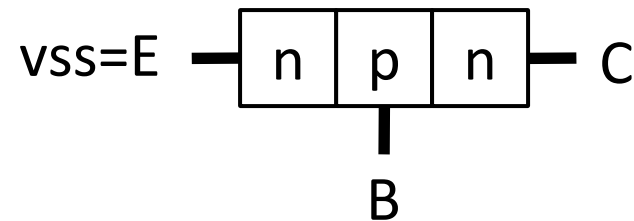
- Solid-state switch: The most amazing invention of the 1900s

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

PNP Transistor

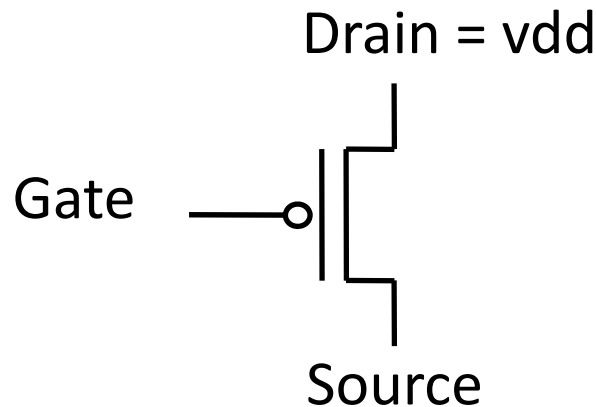


NPN Transistor



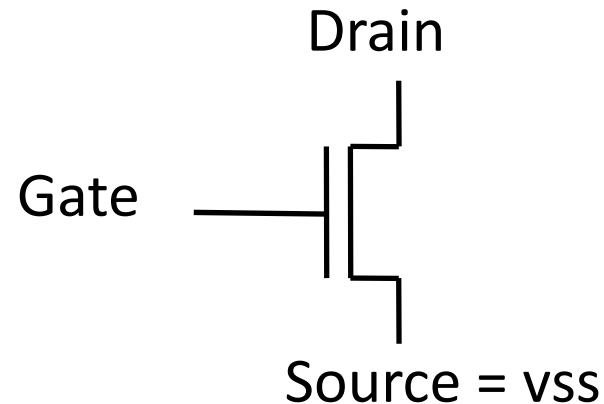
Field Effect Transistors

P-type FET



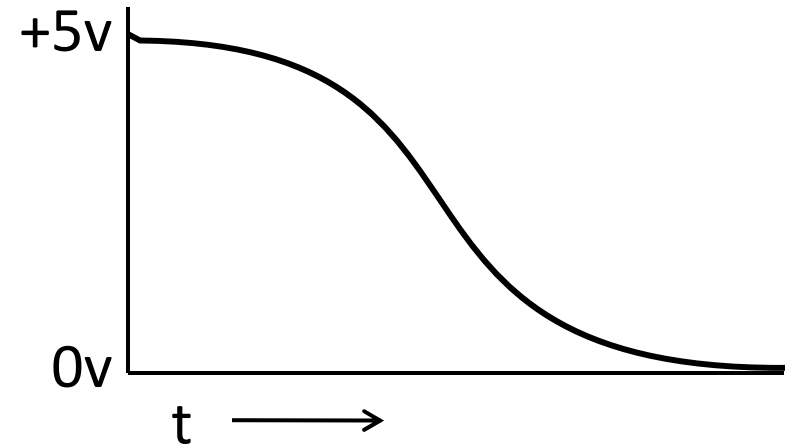
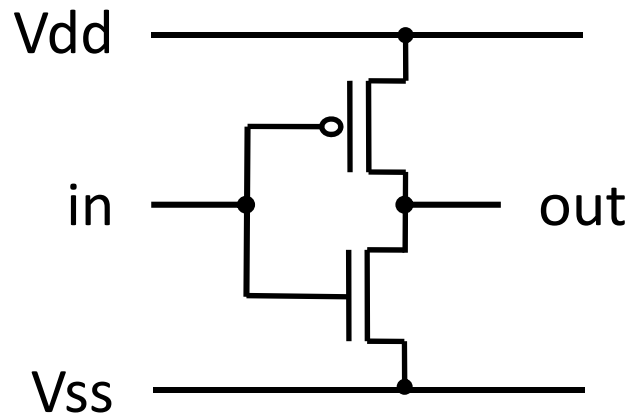
- Connect Source to Drain when Gate = lo
- Drain must be vdd, or connected to source of another P-type transistor

N-type FET



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

Multiple Transistors



In	Out

voltage

Gate delay

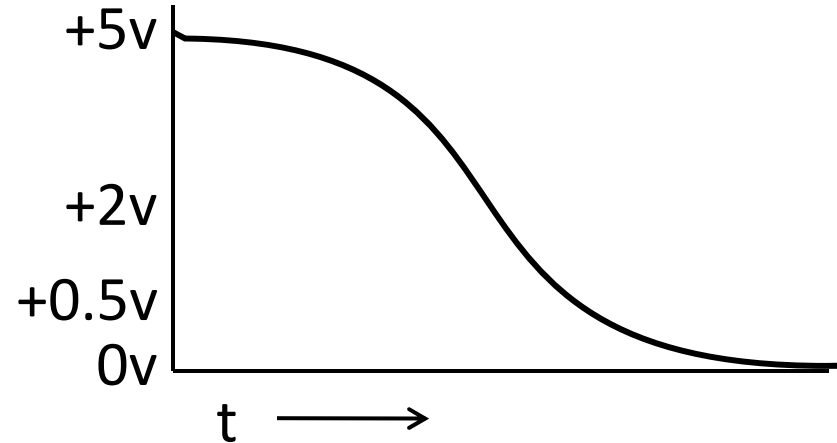
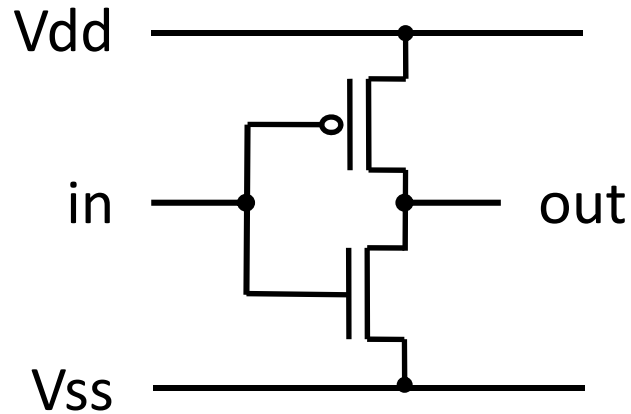
- transistor switching time
- voltage, propagation, fanout, temperature, ...

CMOS design

(complementary-symmetry metal-oxide-semiconductor)

- Power consumption = dynamic + leakage

Digital Logic



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

voltage

In	Out

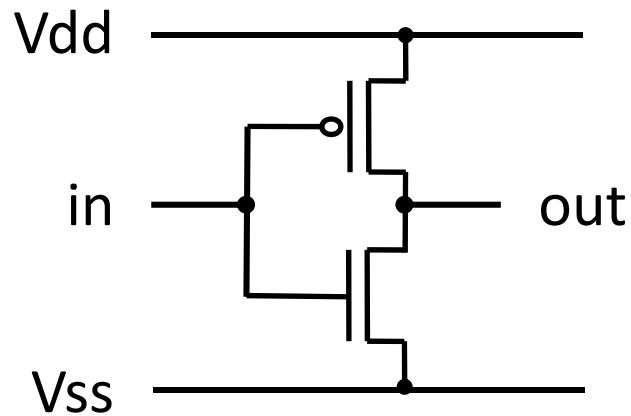
truth table

Conventions:

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

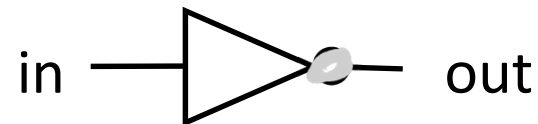
vss = vee = 0v = gnd = false = 0

NOT Gate (Inverter)



Function: NOT

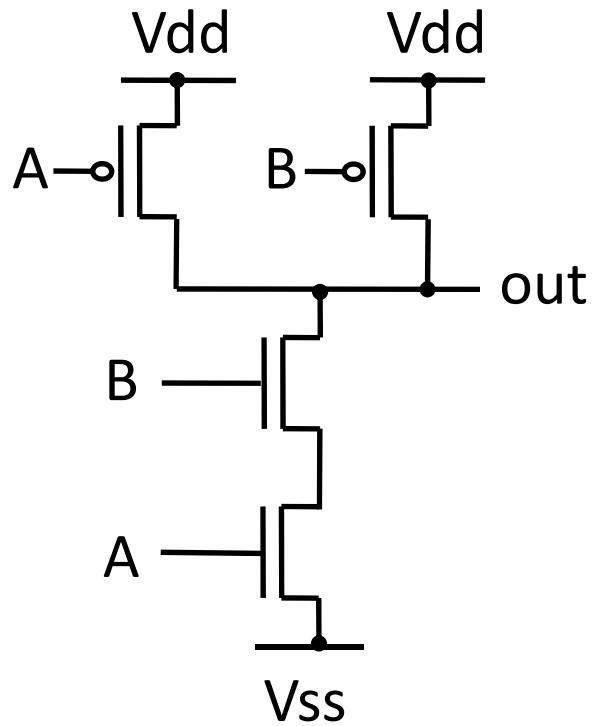
- Symbol:



In	Out
0	1
1	0

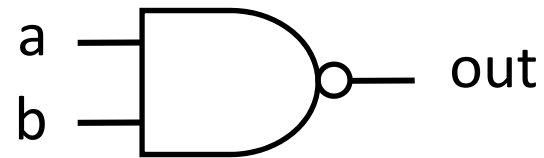
Truth table

NAND Gate



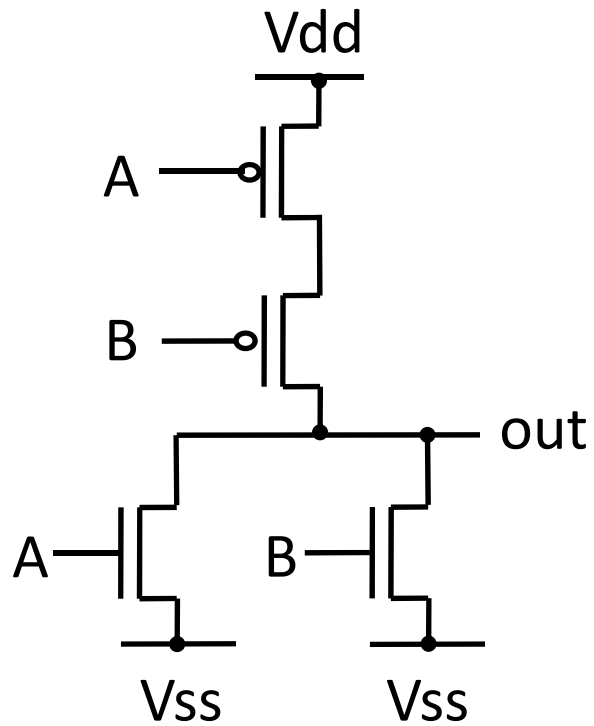
Function: NAND

- Symbol:



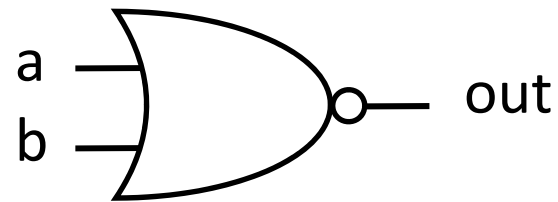
A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate



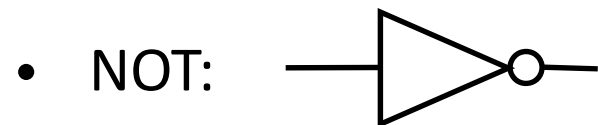
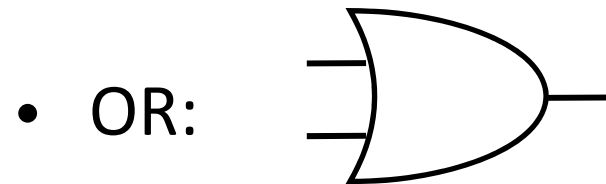
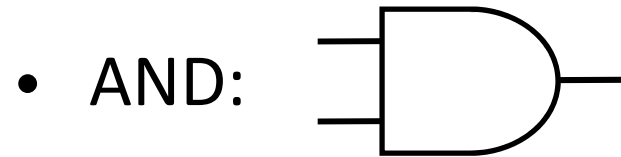
Function: NOR

- Symbol:



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

Building Functions

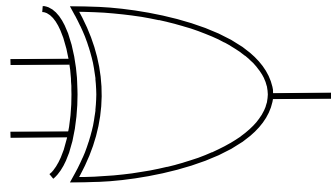


Universal Gates

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: ?

Administrivia

Make sure you have access to CMS and Piazza.com

Lab Sections started *this* week

- Lab0 turned in during Section
- Bring laptop to section, if possible (not required)
- Lab1 available Monday next week (due following Monday)
- Group projects start in week 4 (partner in same section)

Homework1 available Monday

- Due following Monday

Office hours start next week

- More information available on website by this weekend

Clickers not required, bring to every lecture

- Participation, not attendance

Logic Equations

Some notation:

- constants: true = 1, false = 0
- variables: a, b, out, ...
- operators:
 - $\text{AND}(a, b) = a b = a \& b = a \wedge b$
 - $\text{OR}(a, b) = a + b = a | b = a \vee b$
 - $\text{NOT}(a) = \bar{a} = !a = \neg a$

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 \leftrightarrow truth tables \leftrightarrow equations
- Example:

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

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

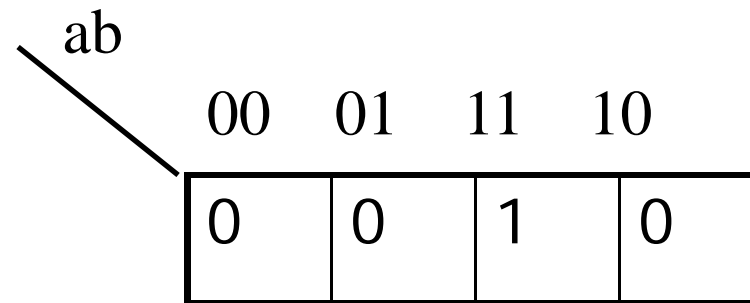
- How does one find the most efficient equation?
 - Manipulate algebraically until satisfied
 - Use Karnaugh maps (or K maps)

Karnaugh maps

- Encoding of the truth table where adjacent cells differ in only one bit

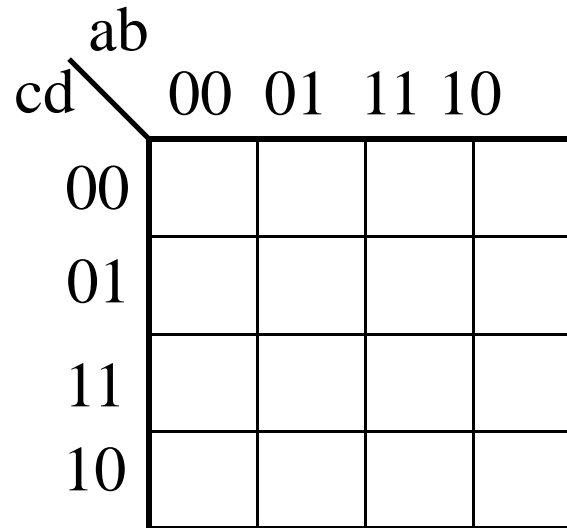
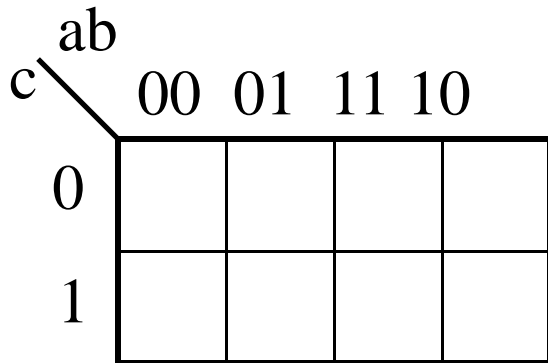
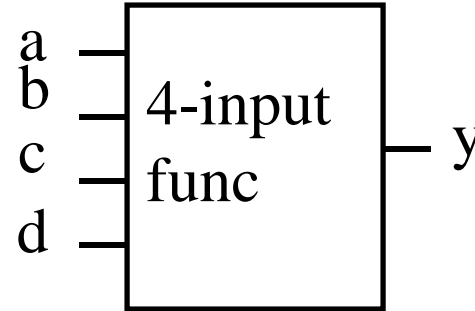
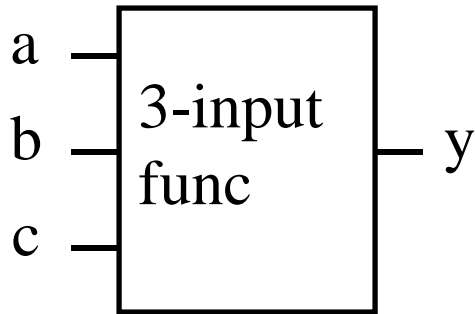
a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

truth table
for AND



Corresponding
Karnaugh map

Bigger Karnaugh Maps



Minimization with Karnaugh maps (1)

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

- ◆ Sum of minterms yields
- $\bar{a}bc + a\bar{b}c + a\bar{b}\bar{c} + a\bar{b}c$

Minimization with Karnaugh maps (2)

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

		ab			
		00	01	11	10
c	0	0	0	0	1
	1	1	1	0	1

- ◆ Sum of minterms yields
 - $\bar{a}bc + a\bar{b}c + a\bar{b}\bar{c} + a\bar{b}c$
- ◆ Karnaugh maps identify which inputs are (ir)relevant to the output

Minimization with Karnaugh maps (2)

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

		ab			
		00	01	11	10
c	0	0	0	0	1
	1	1	1	0	1

◆ Sum of minterms yields

- $\bar{a}bc + \bar{a}bc + a\bar{b}c + a\bar{b}c$

◆ Karnaugh map minimization

- Cover all 1's
- Group adjacent blocks of 2^n 1's that yield a rectangular shape
- Encode the common features of the rectangle
 - ◆ $out = a\bar{b} + \bar{a}c$

Karnaugh Minimization Tricks (1)

		ab			
c		00	01	11	10
0	0	1	1	1	
1	0	0	1	0	

◆ Minterms can overlap

- $out = b\bar{c} + a\bar{c} + ab$

		ab			
c		00	01	11	10
0	1	1	1	1	1
1	0	0	1	0	

◆ Minterms can span 2, 4, 8 or more cells

- $out = \bar{c} + ab$

Karnaugh Minimization Tricks (2)

	ab			
cd	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

- The map wraps around
 – out = $\bar{b}d$

	ab			
cd	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

- out = $\bar{b}d$

Karnaugh Minimization Tricks (3)

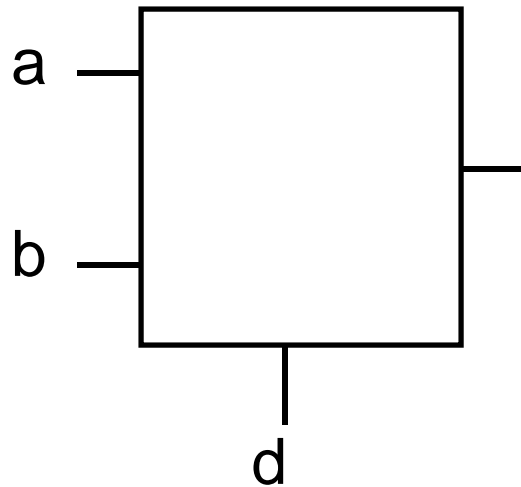
	ab			
cd	00	01	11	10
00	0	0	0	0
01	1	x	x	x
11	1	x	x	1
10	0	0	0	0

- “Don’t care” values can be interpreted individually in whatever way is convenient
 - assume all x’s = 1
 - out = d

	ab			
cd	00	01	11	10
00	1	0	0	x
01	0	x	x	0
11	0	x	x	0
10	1	0	0	1

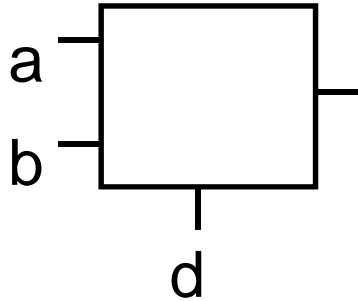
- assume middle x’s = 0
- assume 4th column x = 1
- out = bd

Multiplexer



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

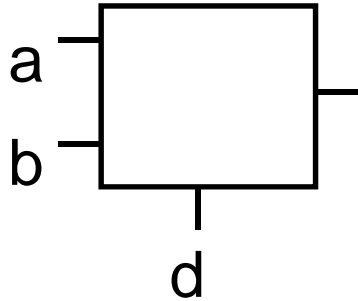
Multiplexer Implementation



- Build a truth table
= $abd + ab\bar{d} + \bar{a}bd + a\bar{b}\bar{d}$

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

Multiplexer Implementation

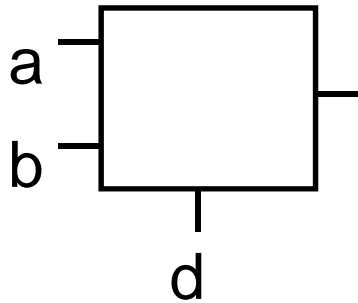


- Build the Karnaugh map

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

d \ ab	00	01	11	10
0	0	0	1	1
1	0	1	1	0

Multiplexer Implementation



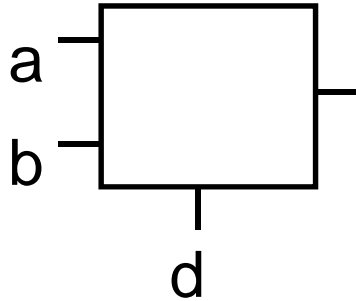
a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Derive Minimal Logic Equation

d \ ab	00	01	11	10
0	0	0	1	1
1	0	1	1	0

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

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

- Derive Minimal Logic Equation

d \ ab	00	01	11	10
0	0	0	1	1
1	0	1	1	0

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

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