Gates and Logic

Hakim Weatherspoon

CS 3410, Spring 2012

Computer Science

Cornell Universty

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

What came was Von Neumann Architecture?

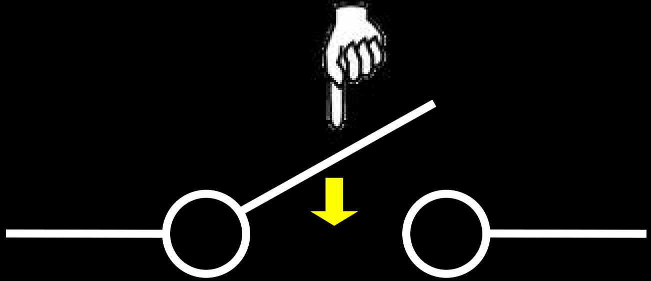
a) The calculator
b) The Bombe
c) The Colossus
d) The ENIAC
e) All of the above

What came was Von Neumann Architecture?

A switch

- Acts as a *conductor* or *insulator*

- Can be used to build amazing things…



Goals for today

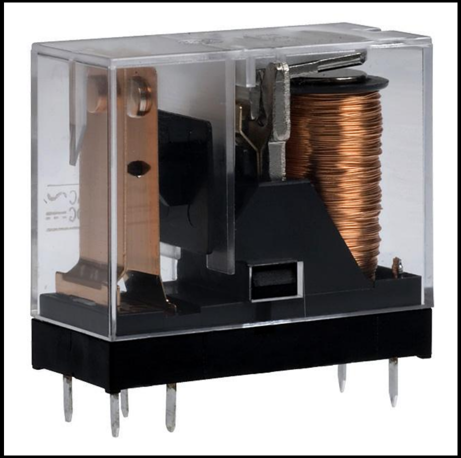
To understand how to program,

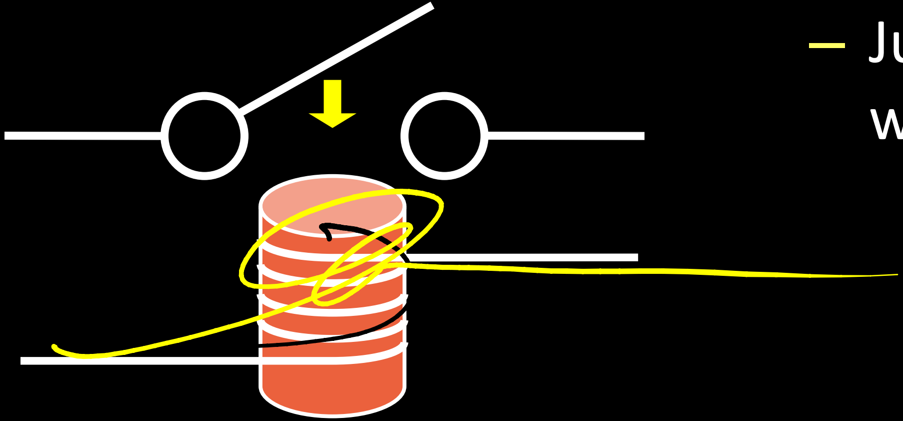
we will build a processor (i.e. a logic circuit)

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

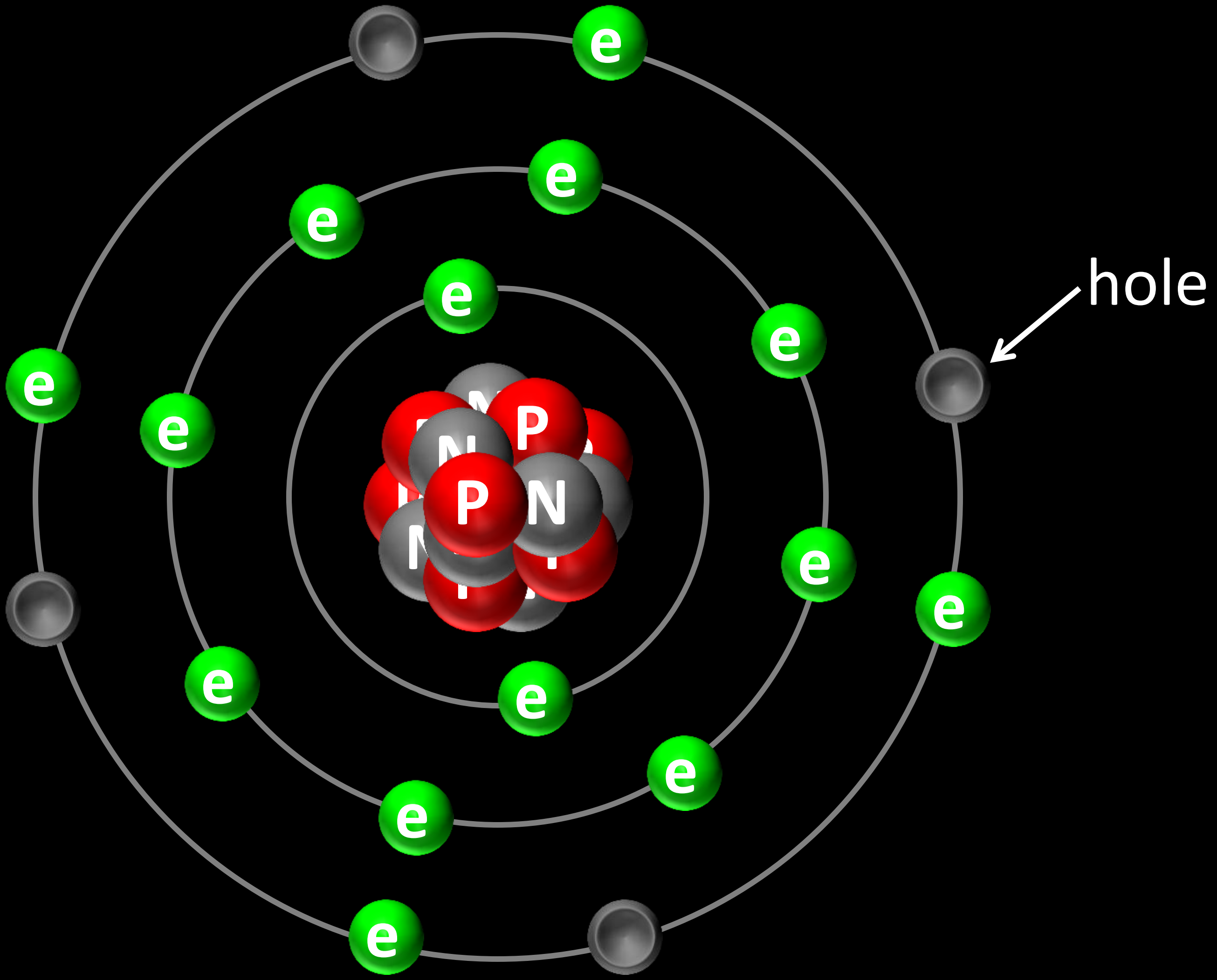




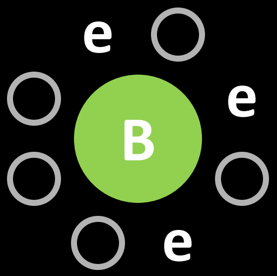
- One current controls another (larger) current

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

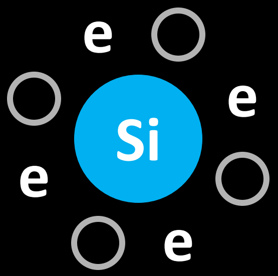
Atoms



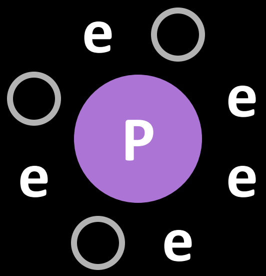
Elements



Boron



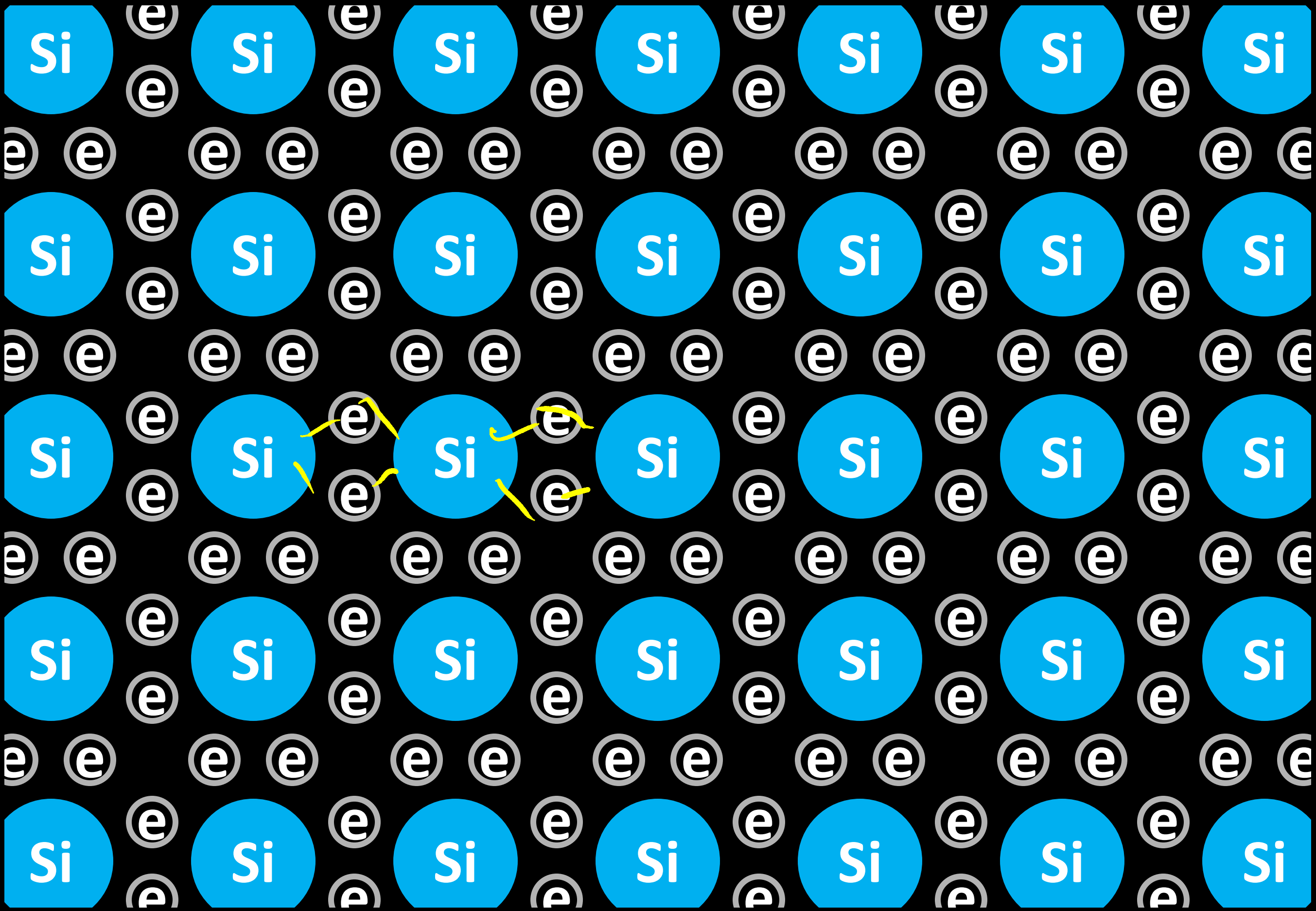
Silicon



Phosphorus

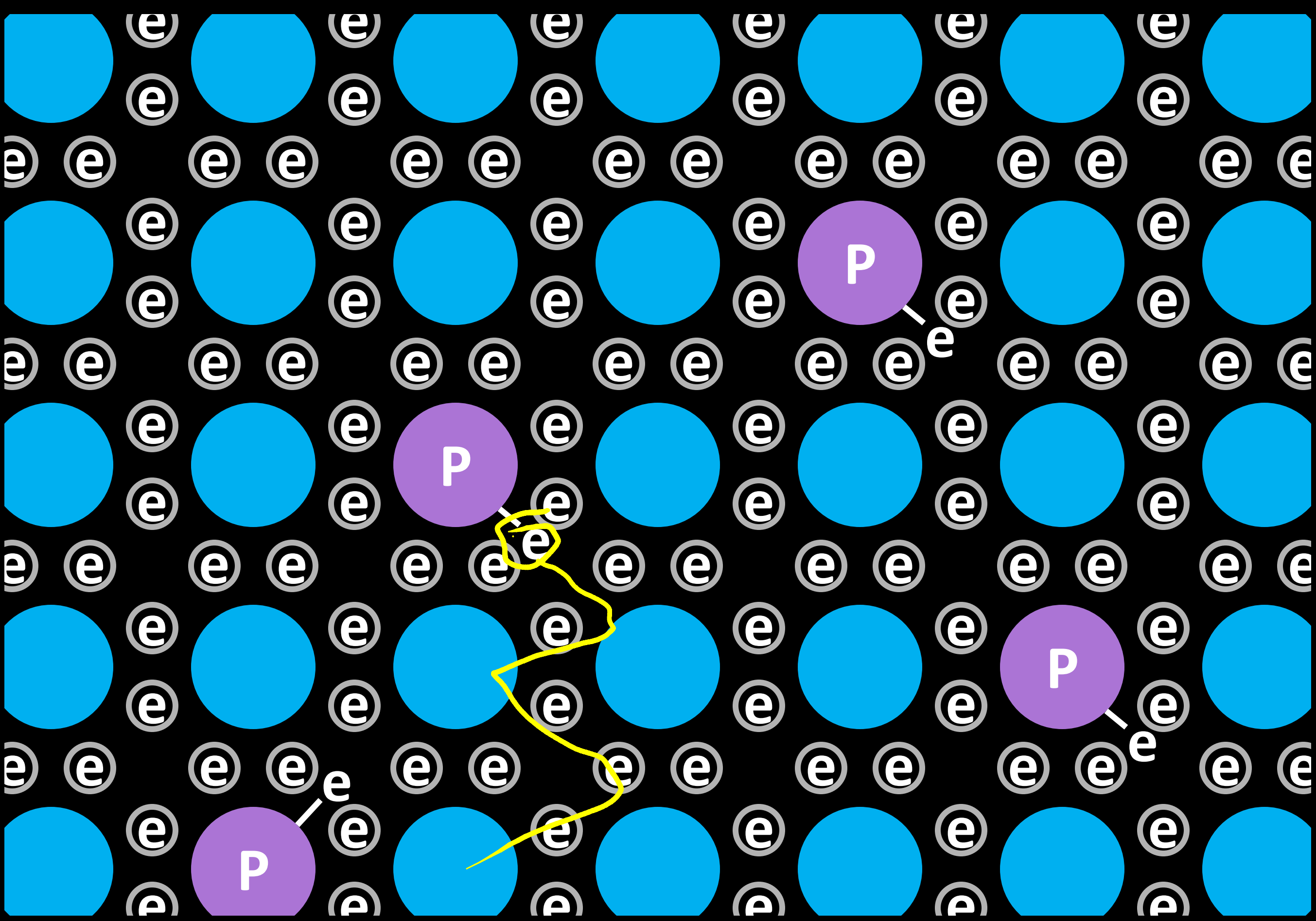
Silicon Crystal

Silicon



Phosphorus Doping

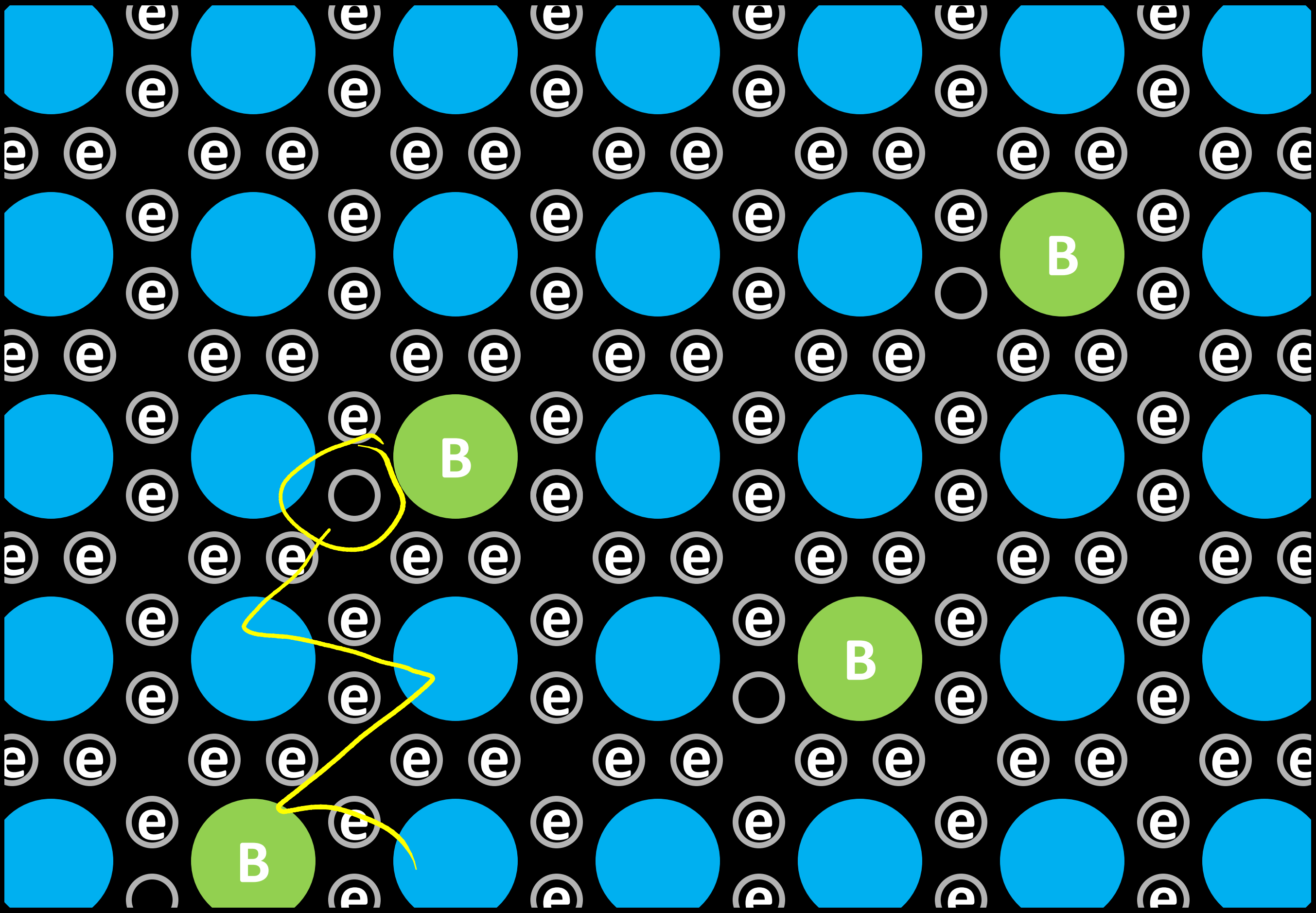
N-Type: Silicon + Phosphorus



mobile electrons
= low V

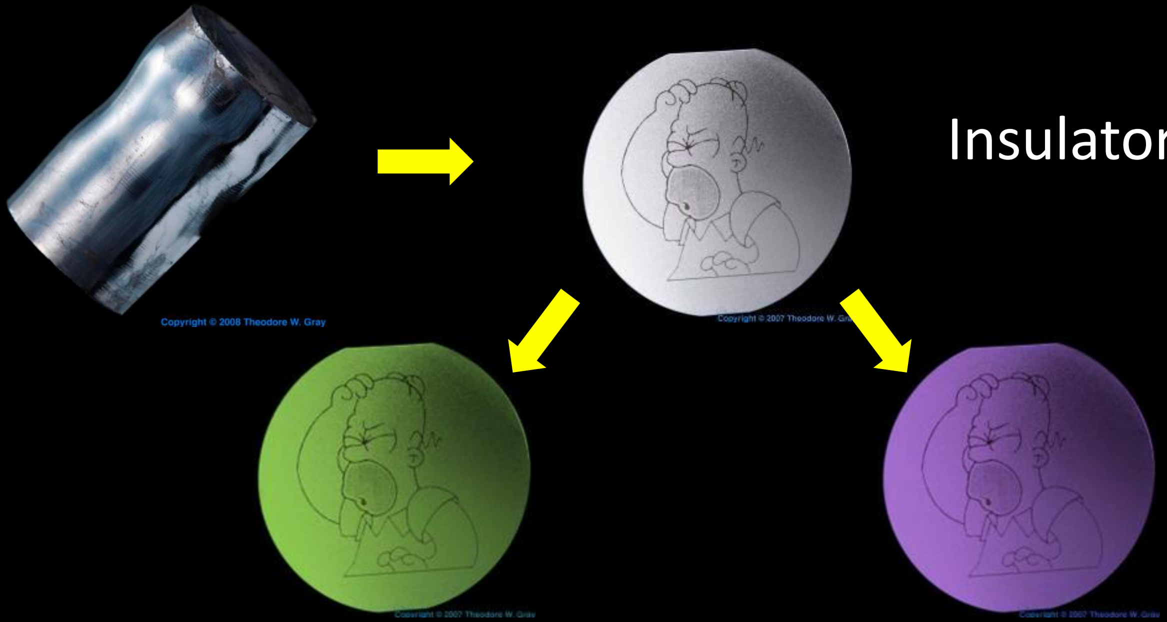
Boron Doping

P-Type: Silicon + Boron



mobile holes =
high v

Semiconductors


Copyright © 2008 Theodore W. Gray
Copyright © 2007 Theodore W. Gray
Insulator

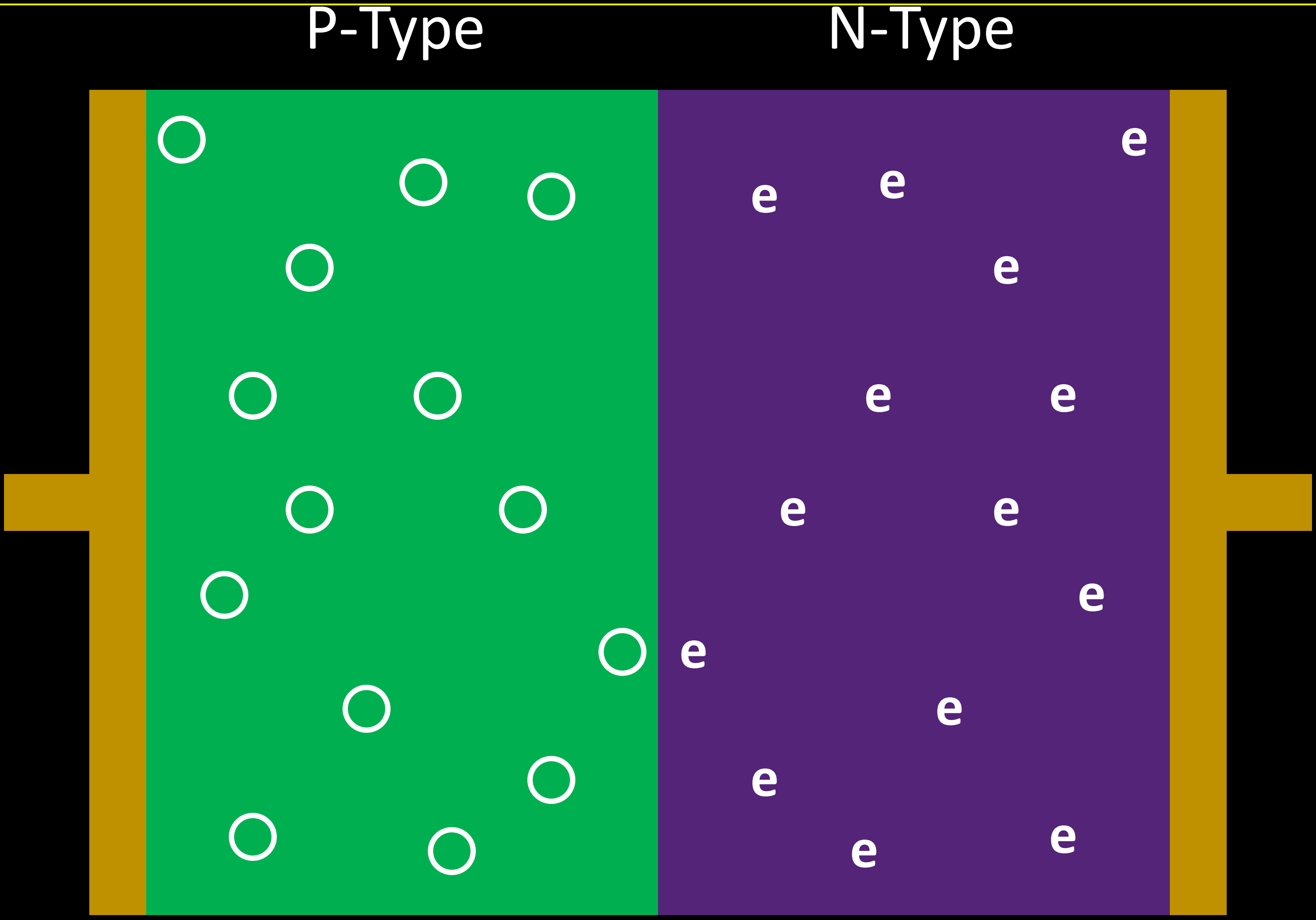
p-type (Si+Boron)
has mobile holes:

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

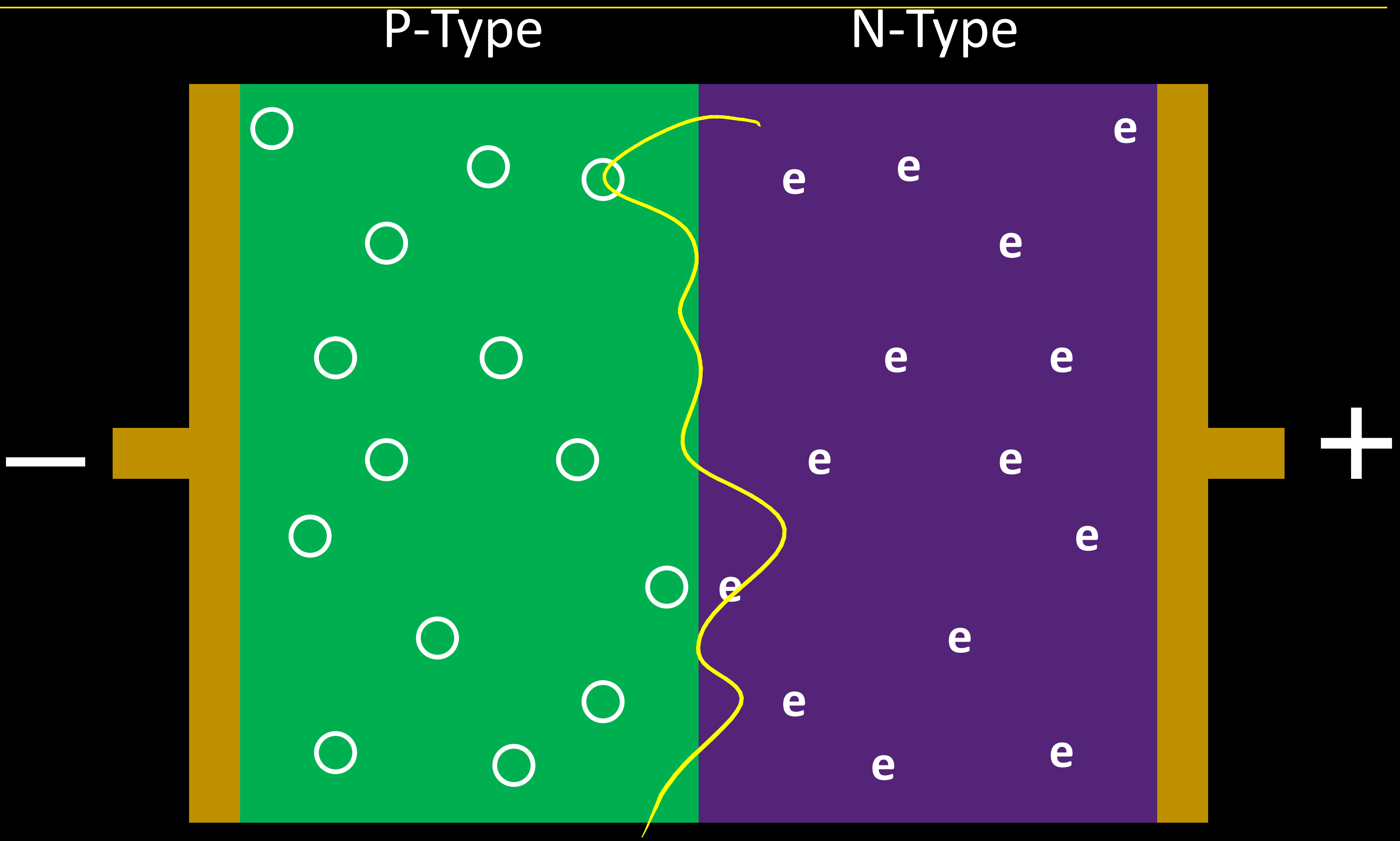
Bipolar Junction



low v → insulator
high v → conductor

low v → conductor
high v → insulator

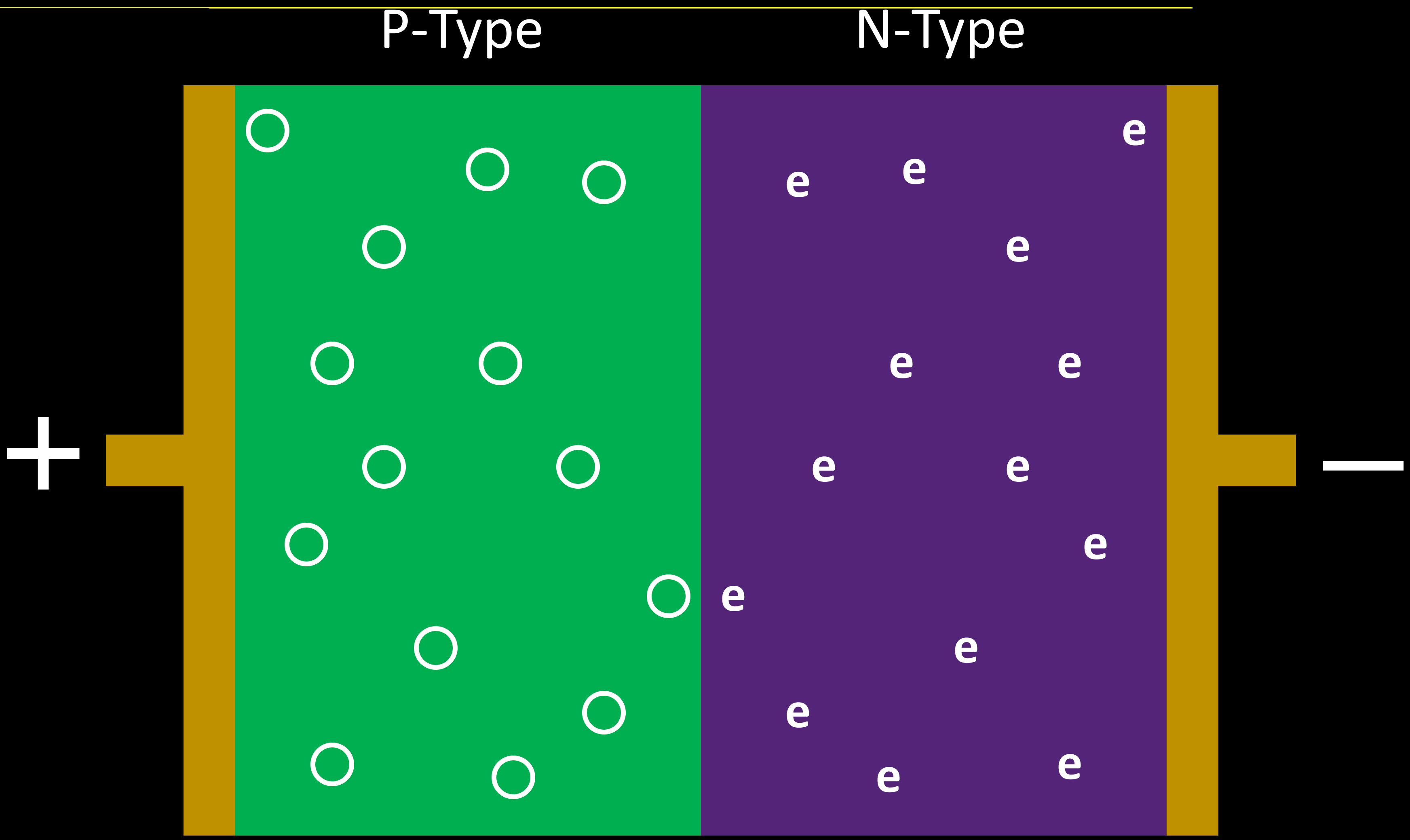
Reverse Bias



low v → insulator
high v → conductor

low v → conductor
high v → insulator

Forward Bias

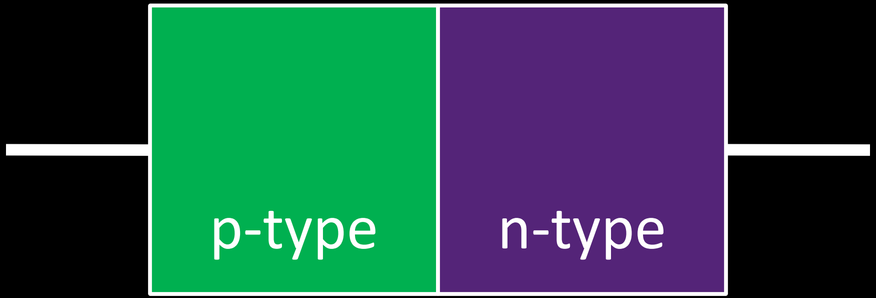


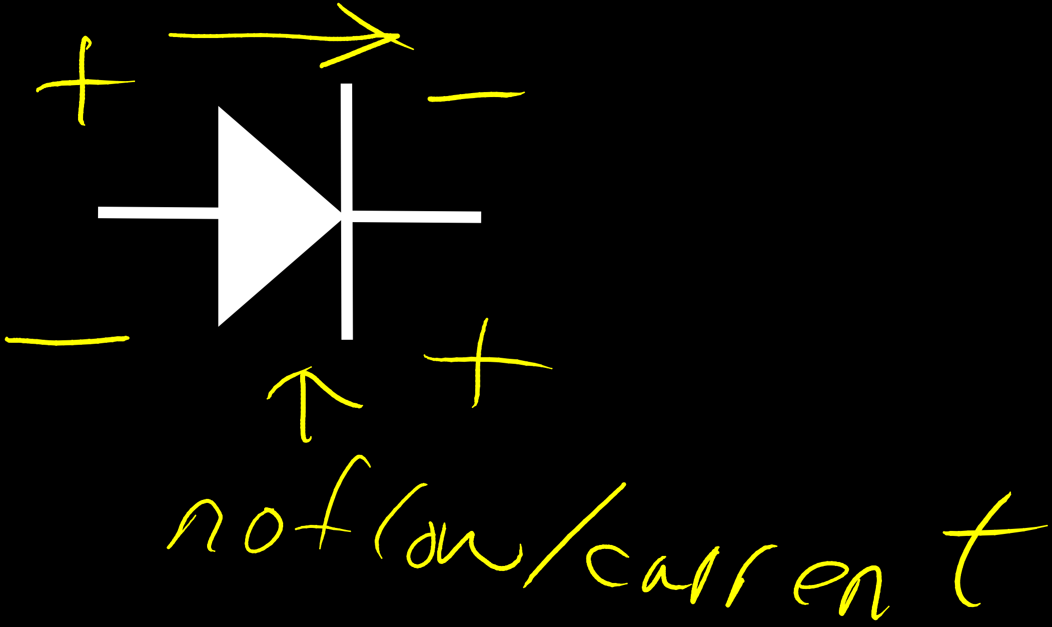
low v → insulator
high v → conductor

low v → conductor
high v → insulator

Diodes

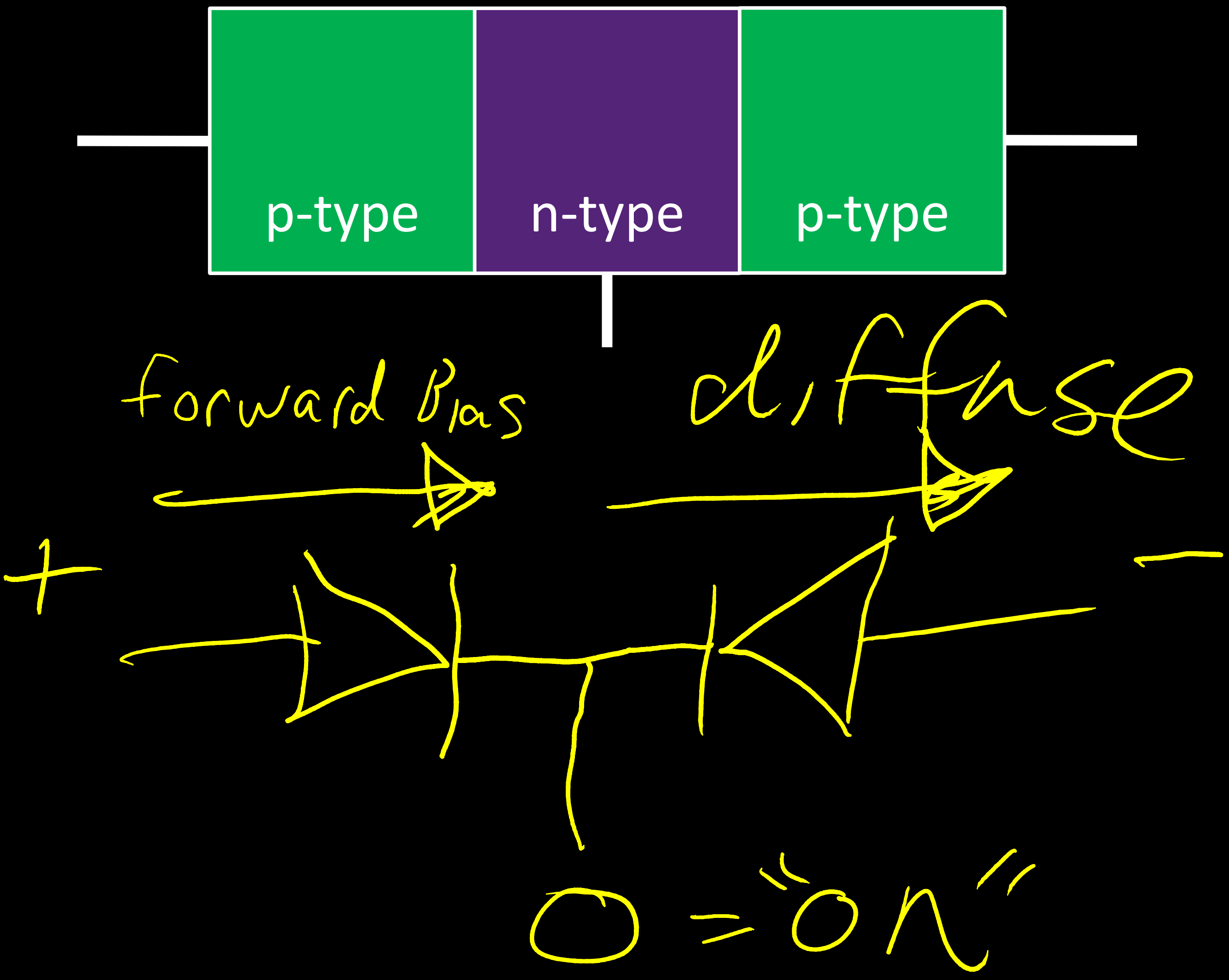
PN Junction “Diode”





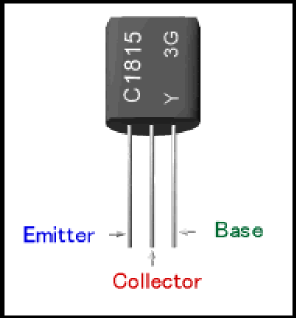
Conventions:
vdd = vcc = +1.2v = +5v = hi
vss = vee = 0v = gnd

PNP Junction



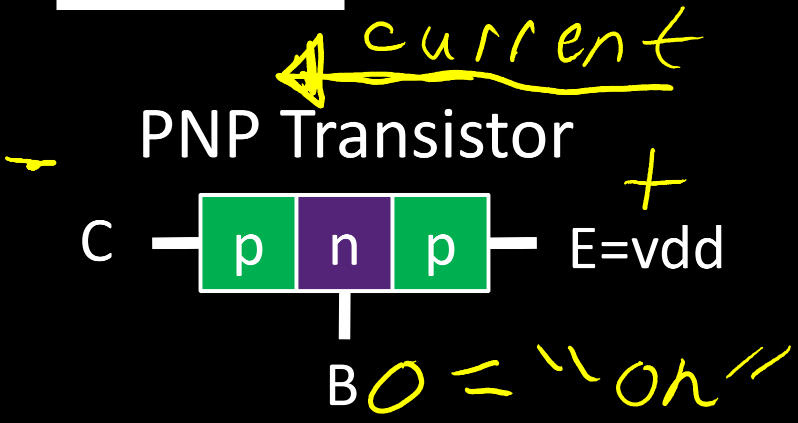
Bipolar Junction Transistors

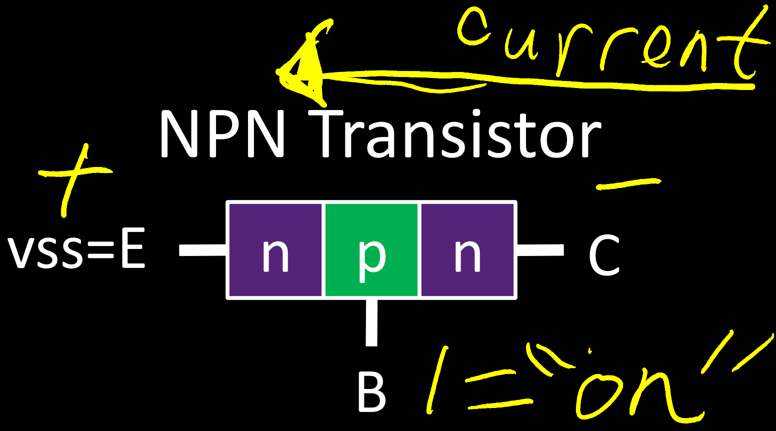
Notice charge and current flipped from last slide

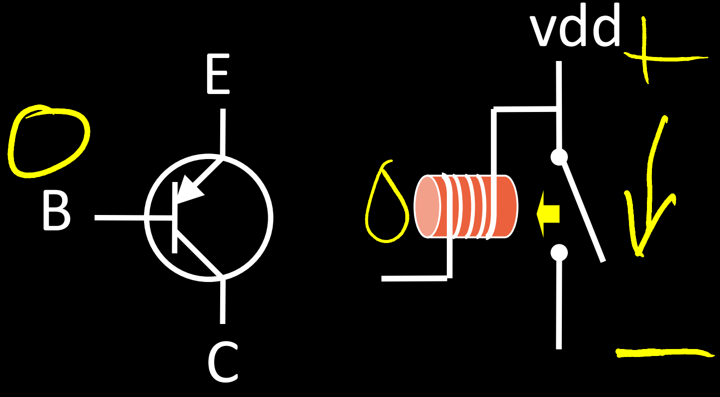


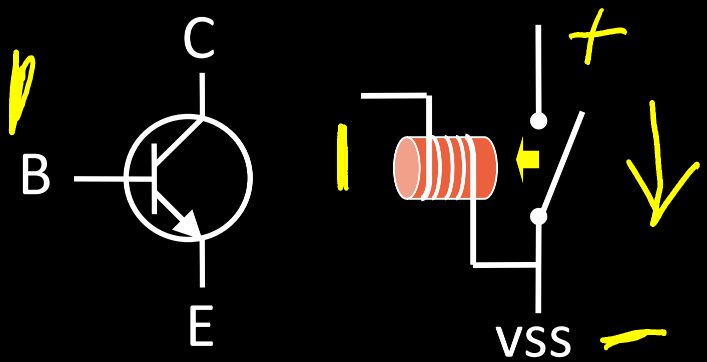
- Solid-state switch: The most amazing invention of the 1900s

Emitter = "input", Base = "switch", Collector = "output"



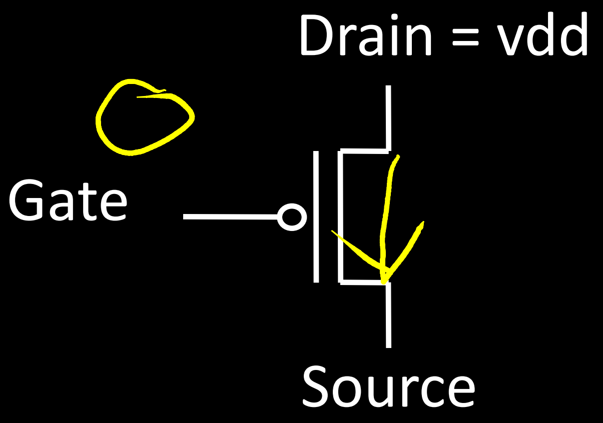






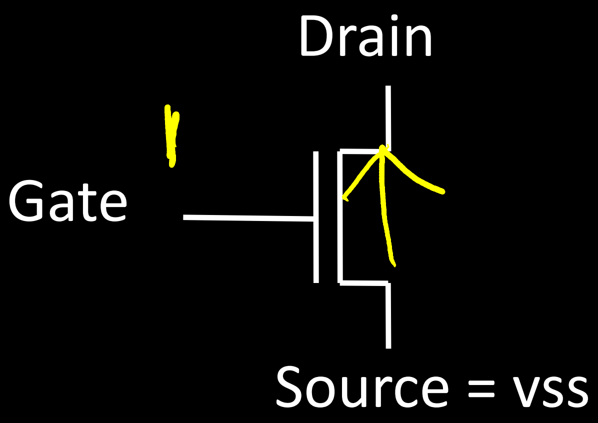
Field Effect Transistors

P-type FET



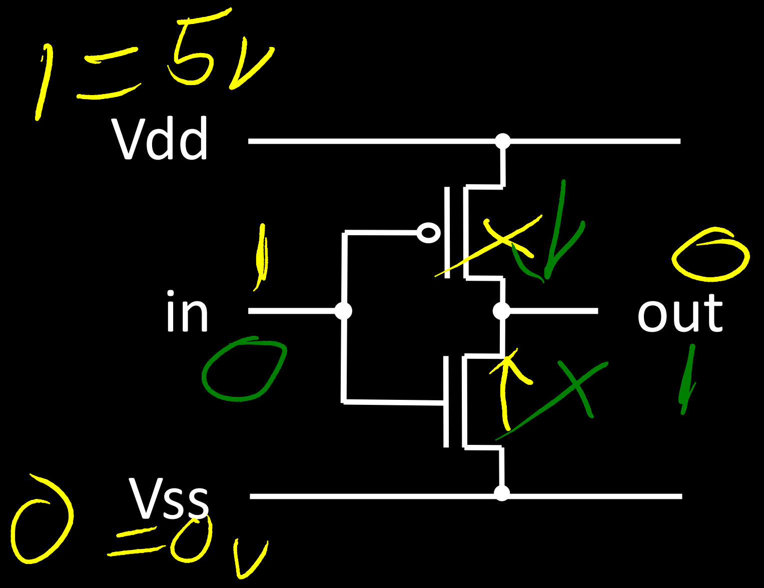
- Connect Source to Drain when Gate = lo
- Drain must be vdd, or connected to source of another P-type transistor

N-type FET



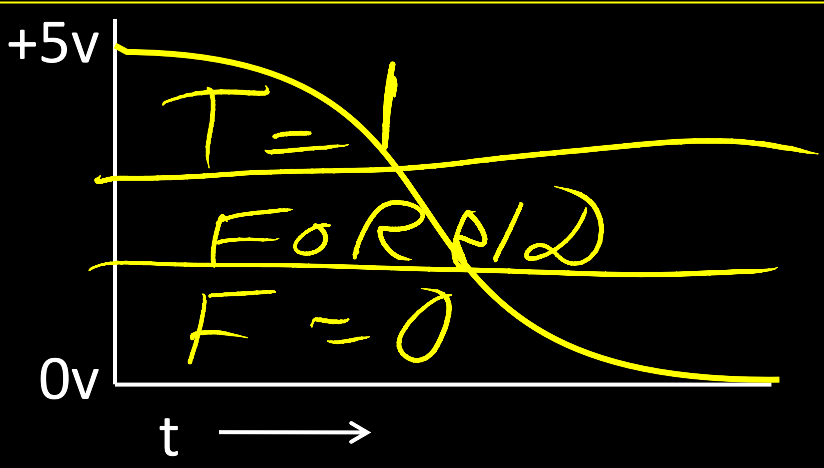
- Connect Source to Drain when Gate = hi
- Source must be vss, or connected to drain of another N-type transistor

Multiple Transistors



In	Out
0v	5v
5v	0v

voltage



Gate delay

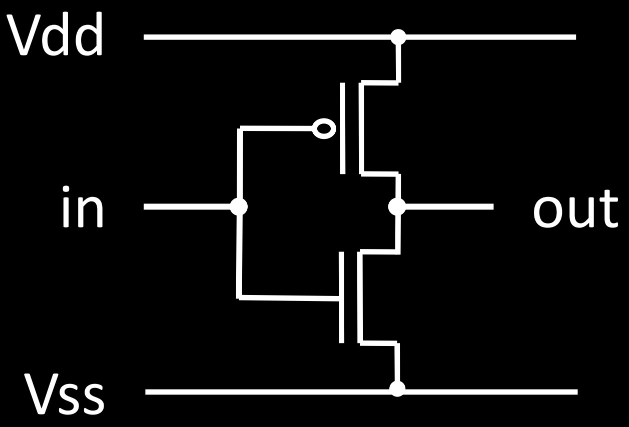
- transistor switching time
- voltage, propagation, fanout, temperature, …

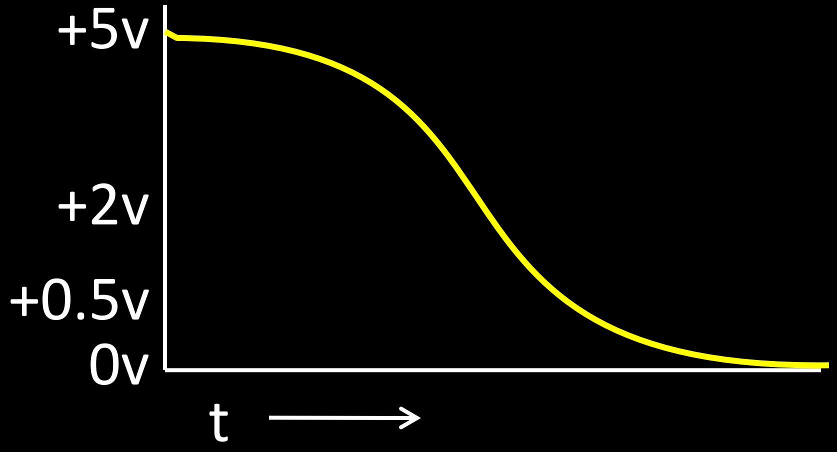
CMOS design

(complementary-symmetry metal–oxide–semiconductor)

- Power consumption = dynamic + leakage

Digital Logic





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

voltage

In	Out

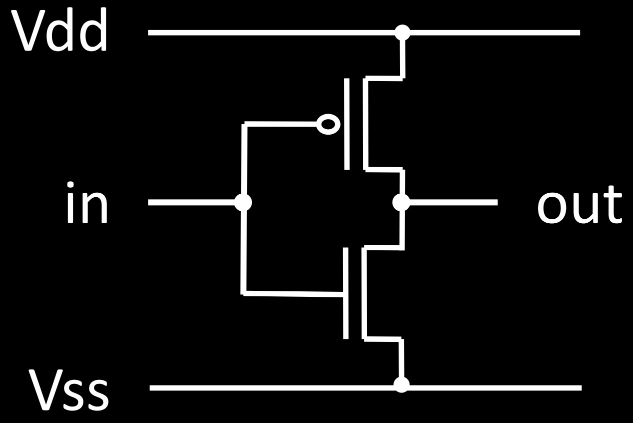
truth table

Conventions:

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

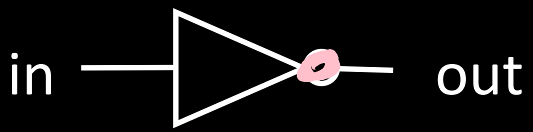
vss = vee = 0v = gnd = false = 0

NOT Gate (Inverter)



Function: NOT

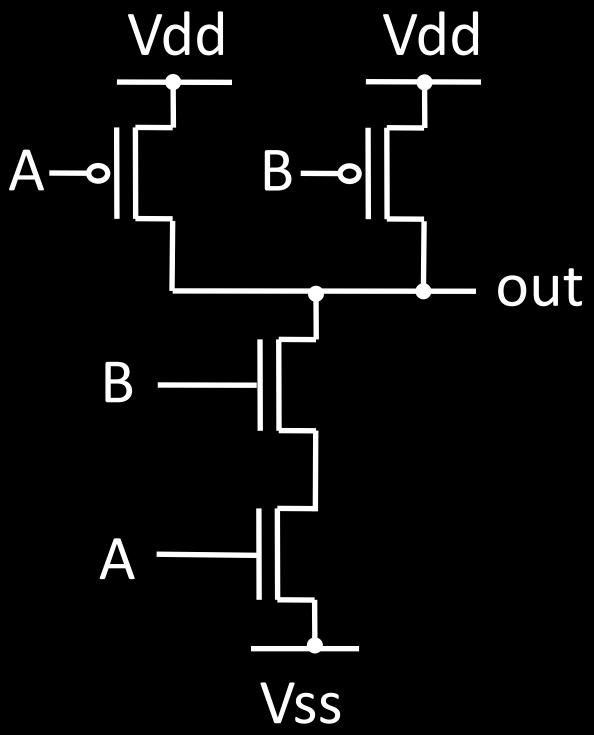
- Symbol:



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

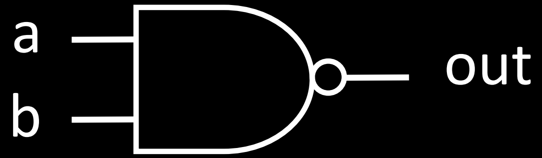
Truth table

NAND Gate



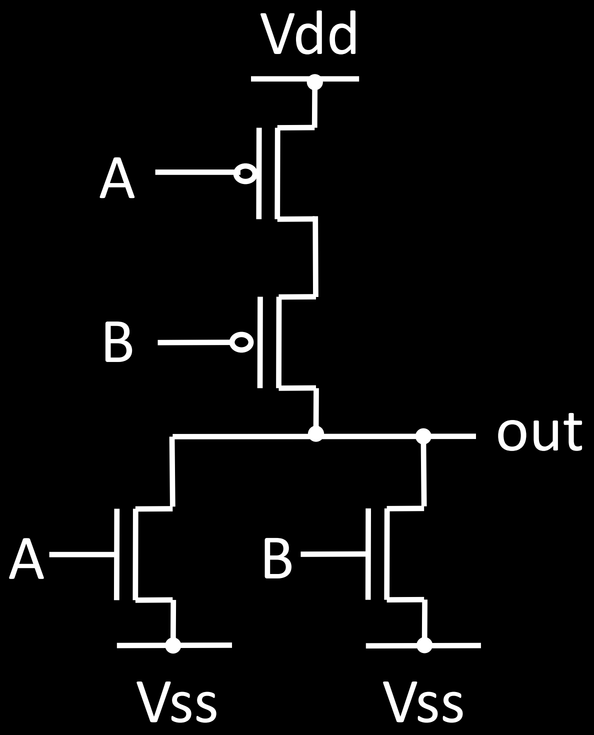
Function: NAND

- Symbol:



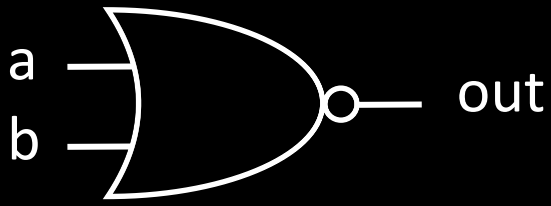
A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate



Function: NOR

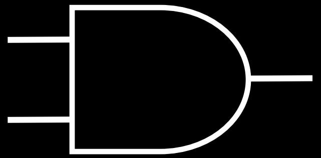
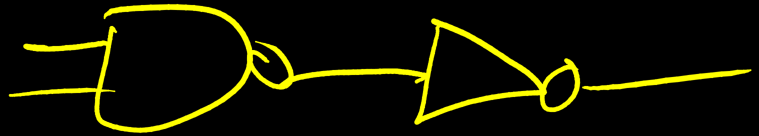
- Symbol:



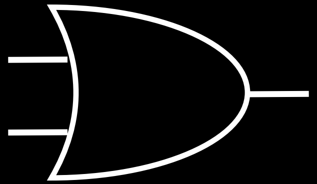
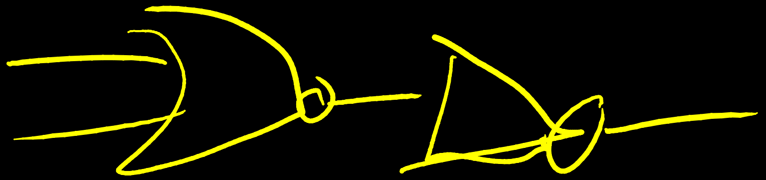
A	B	out
0	0	1
0	1	0
1	0	0
1	1	0

Building Functions

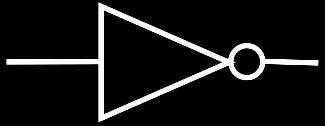
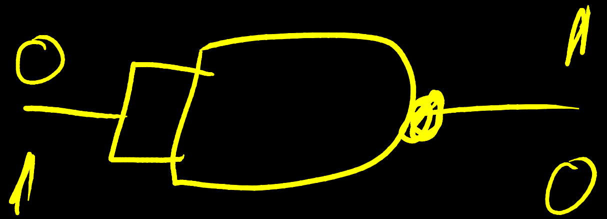
- AND:

- OR:

- NOT:

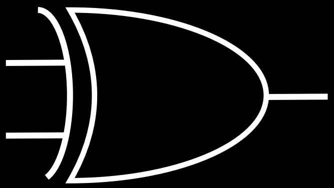



Universal Gates

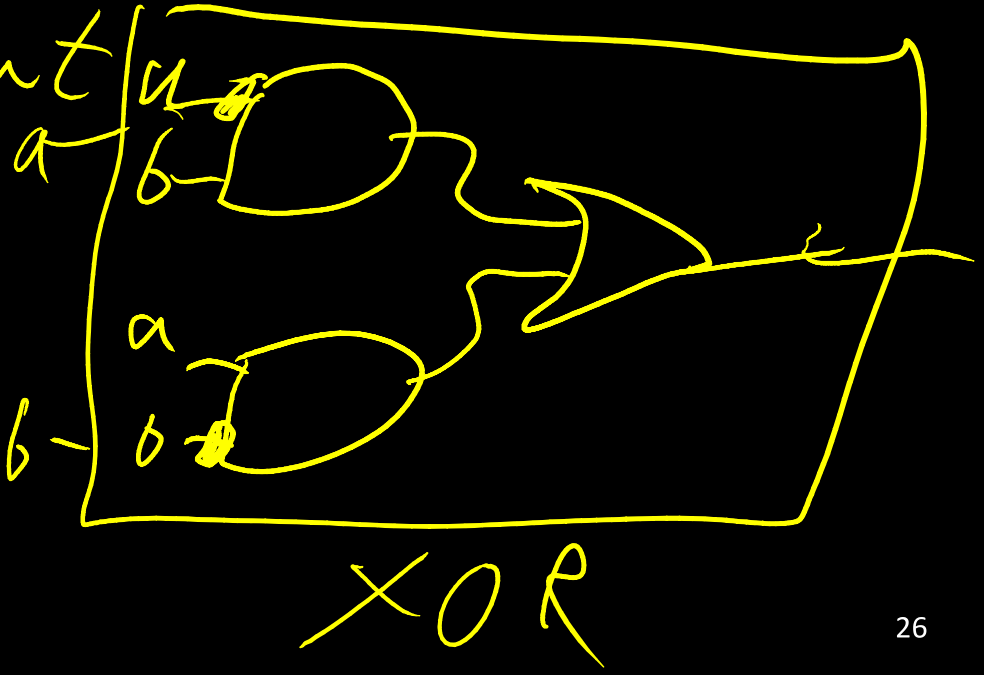
NAND is universal (so is NOR)

- 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")



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



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

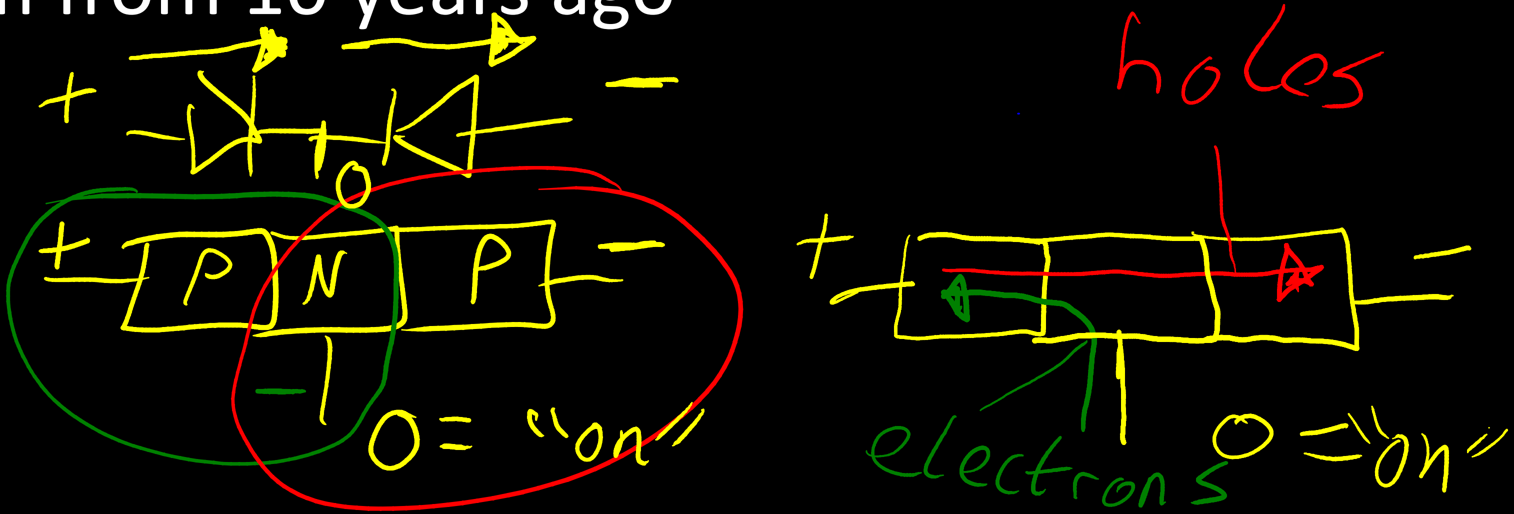
- More information available on website by this weekend

Clickers not required, bring to every lecture

- Participation, not attendance

Example: Big Picture

Computer System Organization and Programming platform from 10 years ago

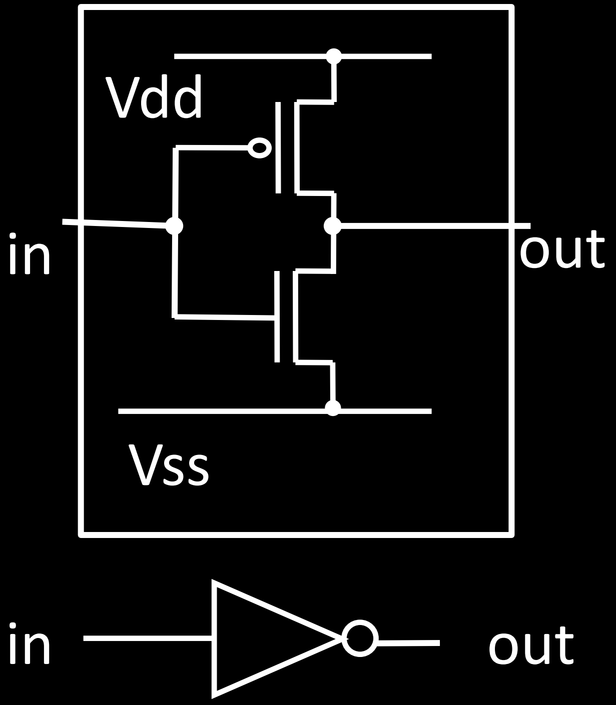


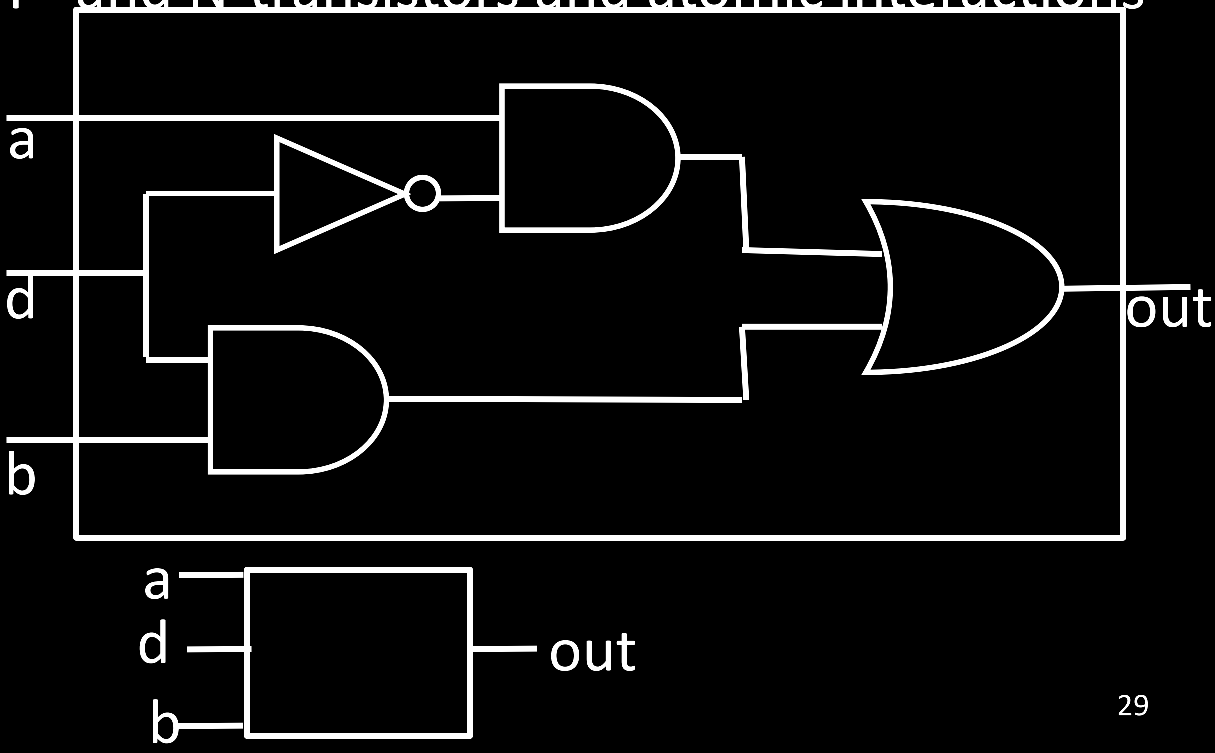
- P- and N-transistors -> NAND and NOR gates -> logic circuit -> processor -> software -> applications (computers, cell phones, TVs, cars, airplanes, buildings, etc).

Big Picture: Abstraction

Hide complexity through simple abstractions

- Simplicity
  - Box diagram represents inputs and outputs
- Complexity
  - Hides underlying P- and N-transistors and atomic interactions





Logic Equations

Some notation:

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

Identities

Identities useful for manipulating logic equations

- For optimization & ease of implementation

$a + 0 = a$ a

$a + 1 = 1$ 1

$a + \bar{a} = 1$ 1

$a\,0 = 0$ 0

$a\,1 = a$ a

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

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

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

$a + a\,b = a$ a

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

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

$\boxed{(a+b)(a+c)}$

$a.a + ac + ba + b.c$

$(a.1 + ac + ba) + bc$

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

$a.1 + bc$

$= \boxed{a + bc}$

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

Predictive – logic minimization

- How to standardize minimizing logic circuits?

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	$a\overline{bc}$
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

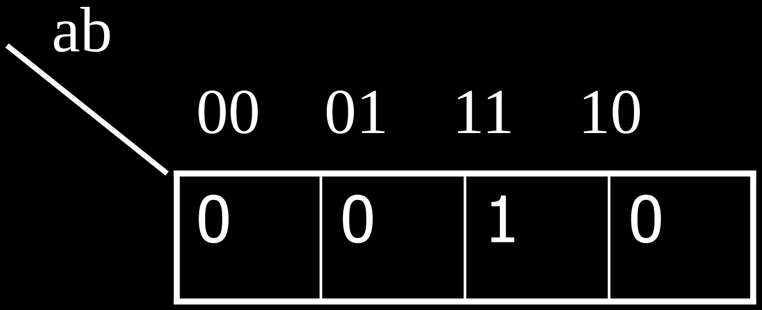
- How does one find the most efficient equation?
  - Manipulate algebraically until satisfied
  - Use Karnaugh maps (or K maps)

Karnaugh maps

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

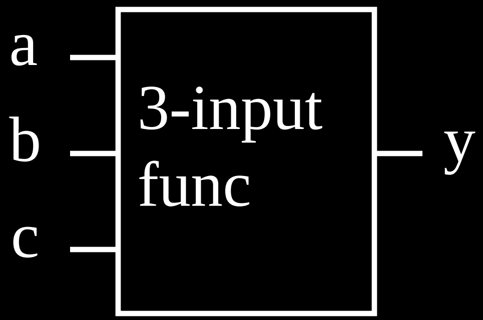
a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

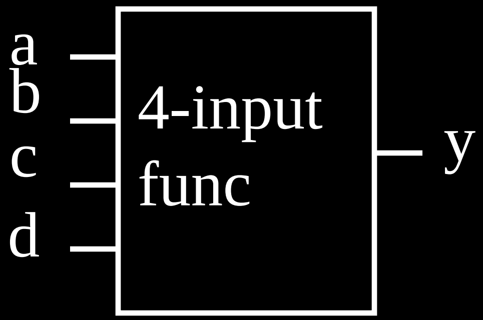
truth table for AND



Corresponding Karnaugh map

Bigger Karnaugh Maps





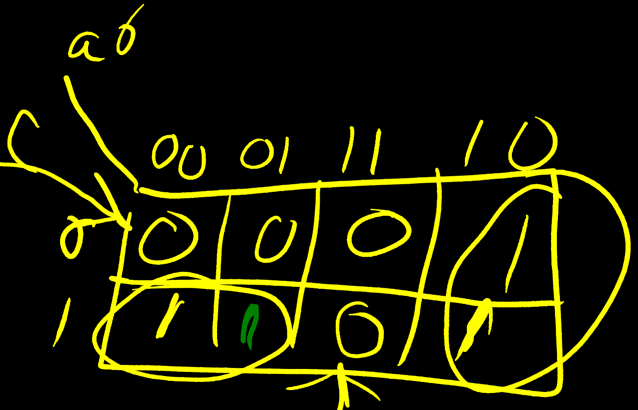
c \ ab	00	01	11	10
0				
1				

cd \ ab	00	01	11	10
00				
01				
11				
10				

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



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

Minimization with Karnaugh maps (2)

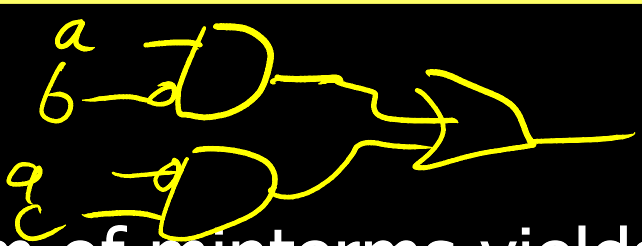
a	b	c	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

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

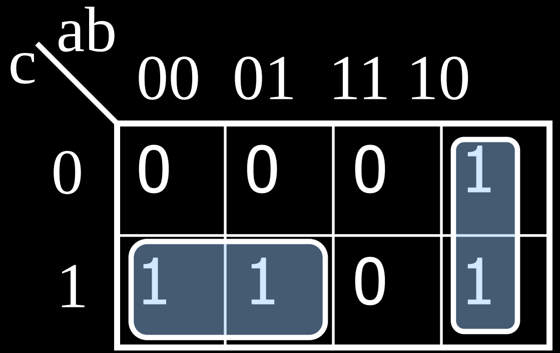
- Sum of minterms yields
  - $\bar{a}\bar{b}c + \bar{a}bc + a\bar{b}\bar{c} + a\bar{b}c$

- Karnaugh maps identify which inputs are (ir)relevant to the output

Minimization with Karnaugh maps (2)



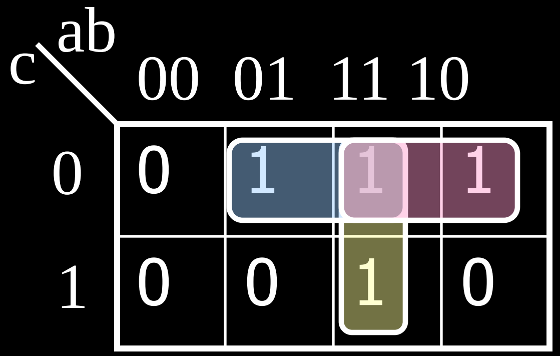
a	b	c	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

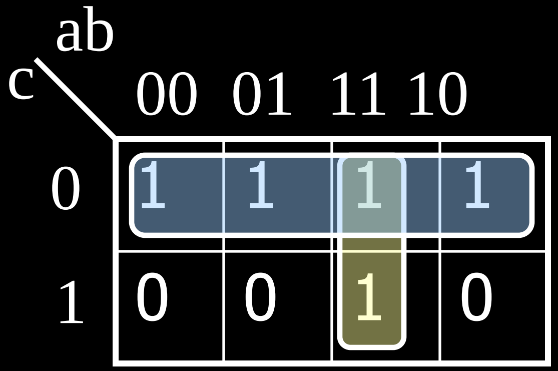


c \ ab	00	01	11	10
0	0	0	0	1
1	1	1	0	1

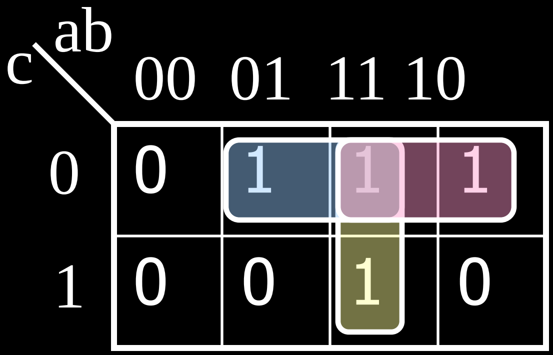
- Sum of minterms yields
  - $\overline{a}\,\overline{b}c + \overline{a}bc + a\overline{b}\,\overline{c} + a\overline{b}c$
- Karnaugh map minimization
  - Cover all 1's
  - Group adjacent blocks of 2^n 1's that yield a rectangular shape
  - Encode the common features of the rectangle
    - out = $a\overline{b} + \overline{a}c$

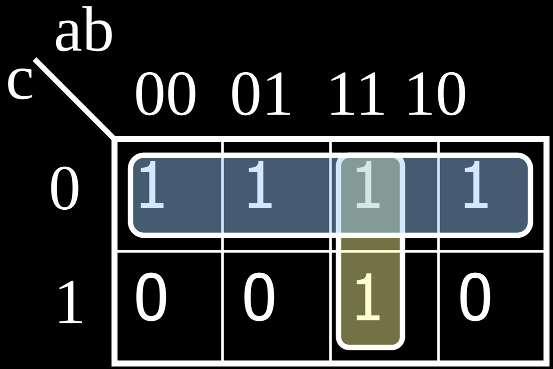
Karnaugh Minimization Tricks (1)





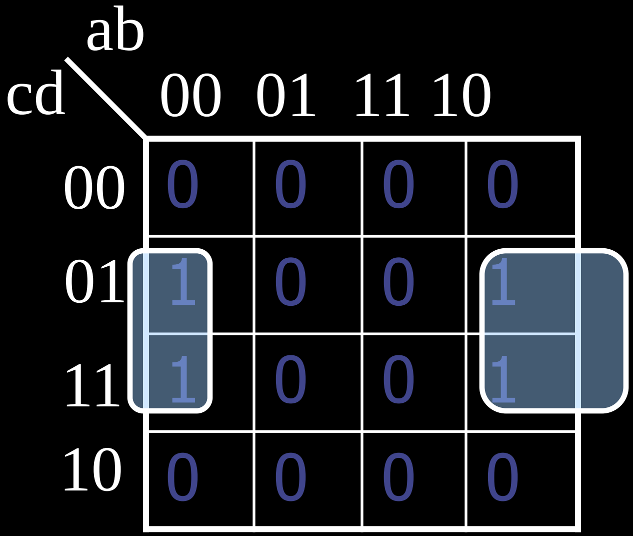
Karnaugh Minimization Tricks (1)



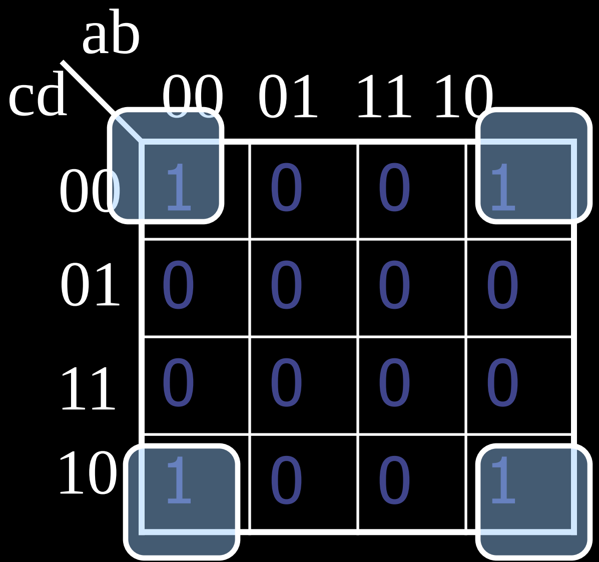


- Minterms can overlap
  - out = $b\bar{c} + a\bar{c} + ab$
- Minterms can span 2, 4, 8 or more cells
  - out = $\bar{c} + ab$

Karnaugh Minimization Tricks (2)

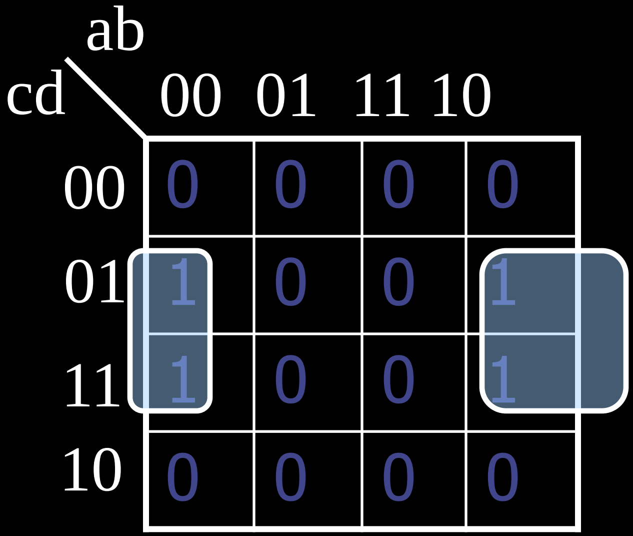


cd \ ab	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

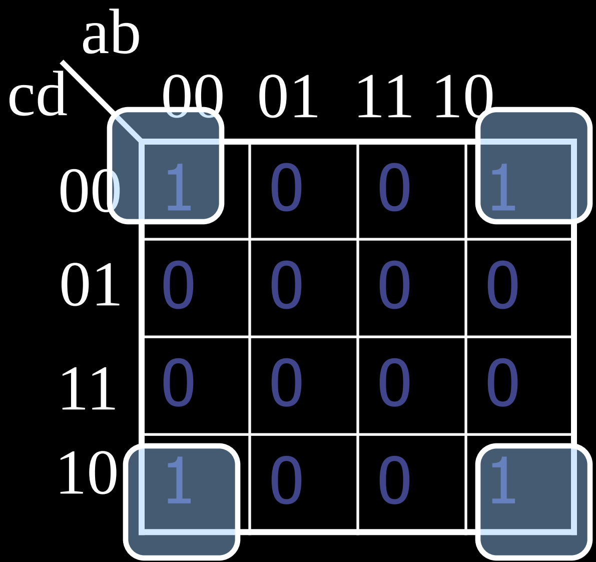


cd \ ab	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

Karnaugh Minimization Tricks (2)

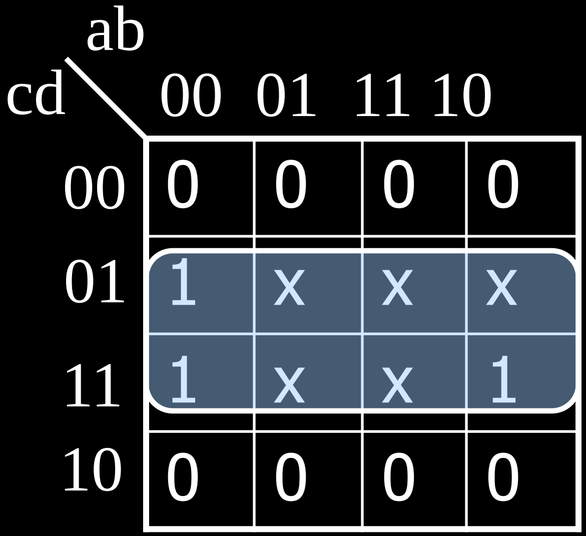


- The map wraps around
  - out = $\overline{b}d$



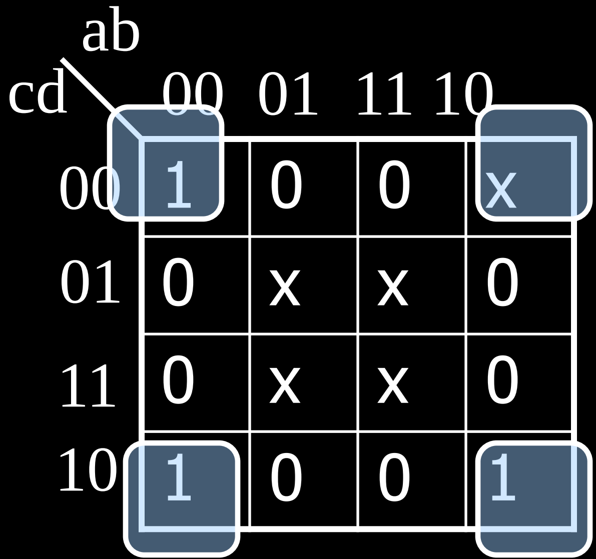
  - out = $\overline{bd}$

Karnaugh Minimization Tricks (3)



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

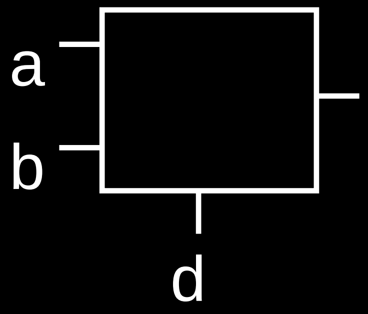
- “Don’t care” values can be interpreted individually in whatever way is convenient
  - assume all x’s = 1
  - out = d



cd \ ab	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 = $\overline{bd}$

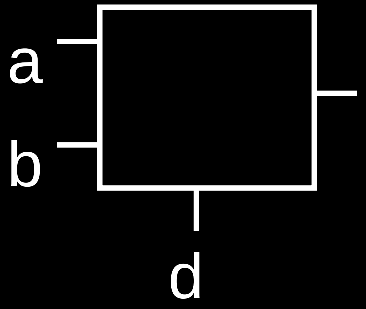
Multiplexer



- 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

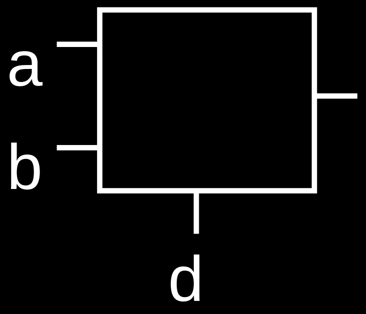


- Build a truth table

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

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

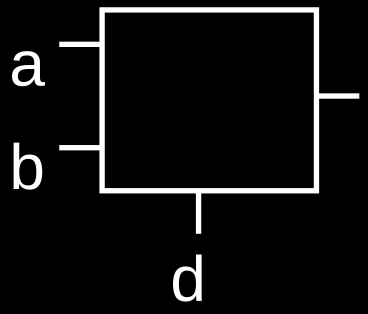


- Build the Karnaugh map

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

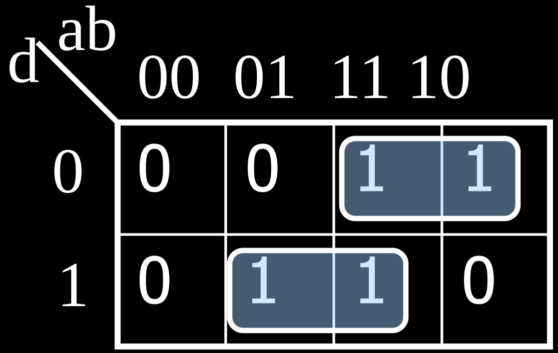
d \ ab	00	01	11	10
0	0	0	1	1
1	0	1	1	0

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

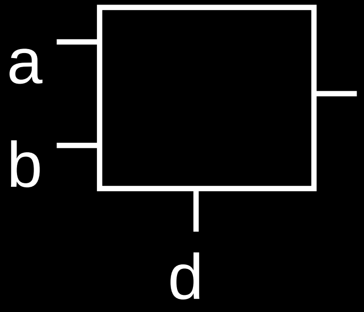
- Derive Minimal Logic Equation



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

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

Multiplexer Implementation

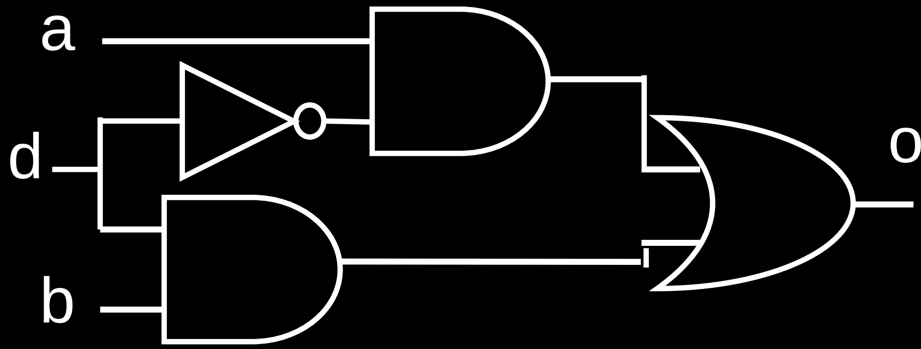


a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Derive Minimal Logic Equation

$$\begin{array}{c|cccc} d \backslash ab & 00 & 01 & 11 & 10 \\ \hline 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{array}$$

- 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