Combinational Logic

Jia Chen
jiac@ucr.edu

Outline

- Two types of logics
- The **theory** behind combinational logics
- The building blocks of combinational logics

Types of circuits

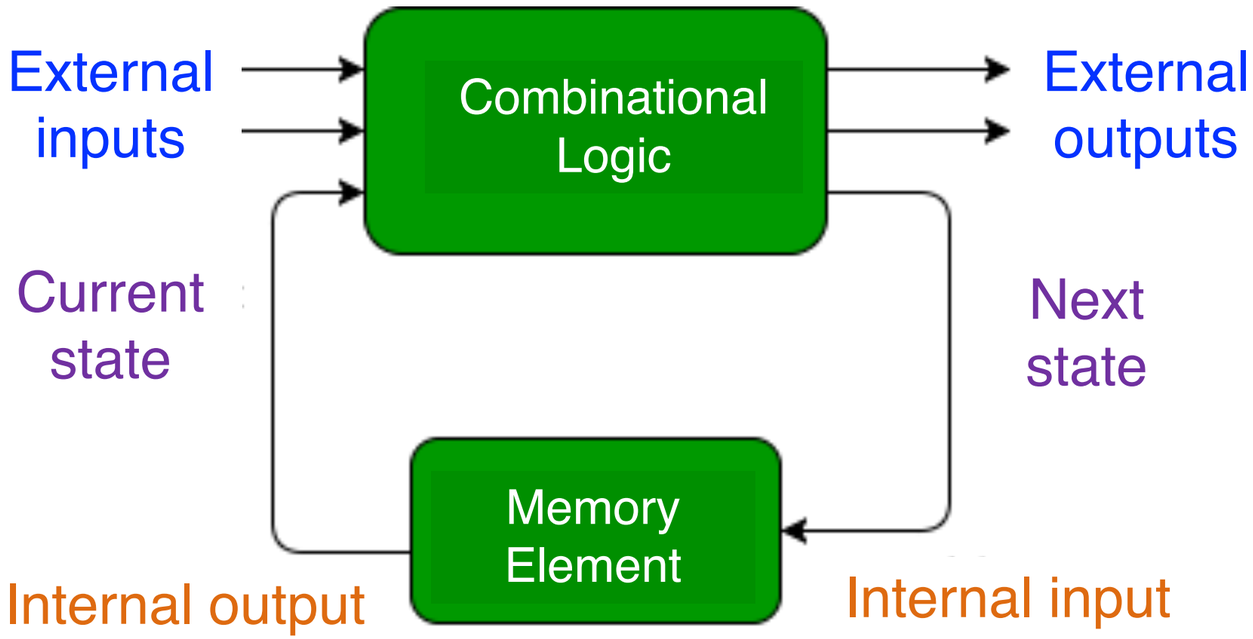
Combinational v.s. sequential logic

- Combinational logic
  - The output is a pure function of its current inputs
  - The output doesn't change regardless how many times the logic is triggered
- Sequential logic
  - The output depends on current and the previous sequence of inputs

Combinational v.s. sequential logic

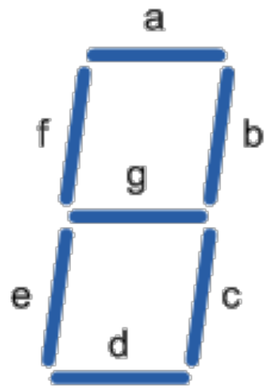
- Combinational logic

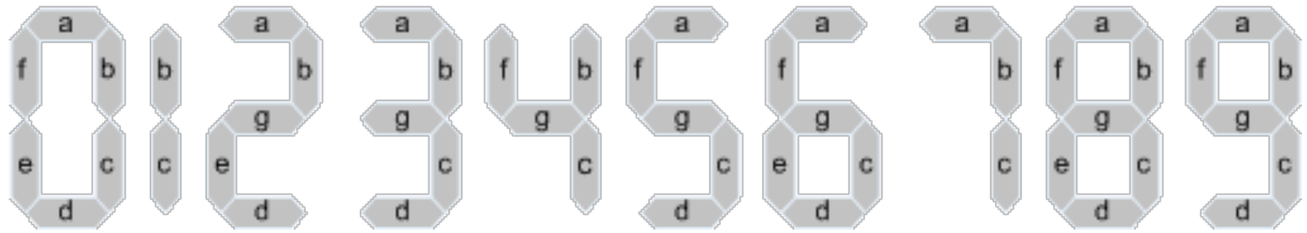
- Sequential logic



When to use combinational logic?

- How many of the following can we simply use combinational logics to accomplish?
  - ① Counters
  - ② Adders
  - ③ Memory cells
  - ④ Decimal to 7-segment LED-decoders
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4





When to use combinational logic?

- How many of the following can we simply use combinational logics to accomplish?
  - ① Counters — **You need the previous input**
  - ② Adders
  - ③ Memory cells — **You need to keep the current state**
  - ④ Decimal to 7-segment LED-decoders

A. 0

B. 1

C. 2

D. 3

E. 4

Slide is adapted from Hung-Wei Tseng

Theory behind each

- A **Combinational logic** is the implementation of a **Boolean Algebra** function with only Boolean Variables as their inputs

- A **Sequential logic** is the implementation of a **Finite-State Machine**

Boolean Algebra

Boolean algebra

- Boolean algebra — George Boole, 1815—1864
  - Introduced binary variables
  - Introduced the three fundamental logic operations: AND, OR, and NOT
  - Extended to abstract algebra with set operations: intersect, union, complement

Basic Boolean Algebra Concepts

- {0, 1}: The only two possible values in inputs/outputs
- Basic operators
  - AND (•) — a • b or ab
    - returns 1 only if both a **and** b are 1s
    - otherwise returns 0
  - OR (+) — a + b
    - returns 1 if a **or** b is 1
    - returns 0 if none of them are 1s
  - NOT (') — a' or !a or $\overline{a}$
    - returns 0 if a is 1
    - returns 1 if a is 0

Generalization of AND & OR to > 2 variables

- Given n variables a_1, a_2, …, a_n where n > 2

  - AND (•) — $a_1 \bullet a_2 \ldots \bullet a_n$ or $a_1 a_2 \ldots a_n$
    - returns 1 if all variables are 1s
    - otherwise returns 0
    - In other words, it returns 1 *if and only if* $a_1 = a_2 = \ldots = a_n = 1$

  - OR (+) — $a_1 + a_2 \ldots + a_n$
    - returns 1 if at least one variable is 1
    - returns 0 if none of the variables are 1s
    - In other words, it returns 0 *if and only if* $a_1 = a_2 = \ldots = a_n = 0$

Truth tables

- A table sets out the functional values of logical expressions on each of their functional arguments, that is, for each combination of values taken by their logical variables

AND

Input		Output
A	B	
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

OR

Input		Output
A	B	
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

NOT

Input	Output
A	
0	1
1	0

Let’s practice!

- X, Y are two Boolean variables. Consider the following function:
  X • Y + X
  How many of the following the input values of X and Y can lead to an output of 1
  ① X = 0, Y = 0
  ② X = 0, Y = 1
  ③ X = 1, Y = 0
  ④ X = 1, Y = 1
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Let's practice!

- X, Y are two Boolean variables. Consider the following function:
  X • Y' + X
  How many of the following the input values of X and Y can lead to an output of 1

  ① X = 0, Y = 0
  ② X = 0, Y = 1
  ③ X = 1, Y = 0
  ④ X = 1, Y = 1

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Input		Y'	XY'	XY' + X	Output
X	Y				
0	0	1	0	0	**0**
0	1	0	0	0	**0**
1	0	1	1	1	**1**
1	1	0	0	1	**1**

Derived Boolean operators

- NAND — (a • b)’
- NOR — (a + b)’
- XOR — (a + b) • (a’ + b’) or ab’ + a’b or a⊕b, a.k.a. odd operator
- XNOR — (a + b’) • (a’ + b) or ab + a’b’ or a⊙b, a.k.a. concurrence operator

NAND

Input		Output
A	B	
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

NOR

Input		Output
A	B	
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

XOR

Input		Output
A	B	
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

XNOR

Input		Output
A	B	
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

Generalization of XOR & XNOR to > 2 variables

- XOR: Exclusive-OR (Ex-OR for abbreviation)
- XNOR: Exclusive-NOR (Ex-NOR for abbreviation)

- Given n variables $a_1, a_2, \ldots, a_n$ where $n > 2$

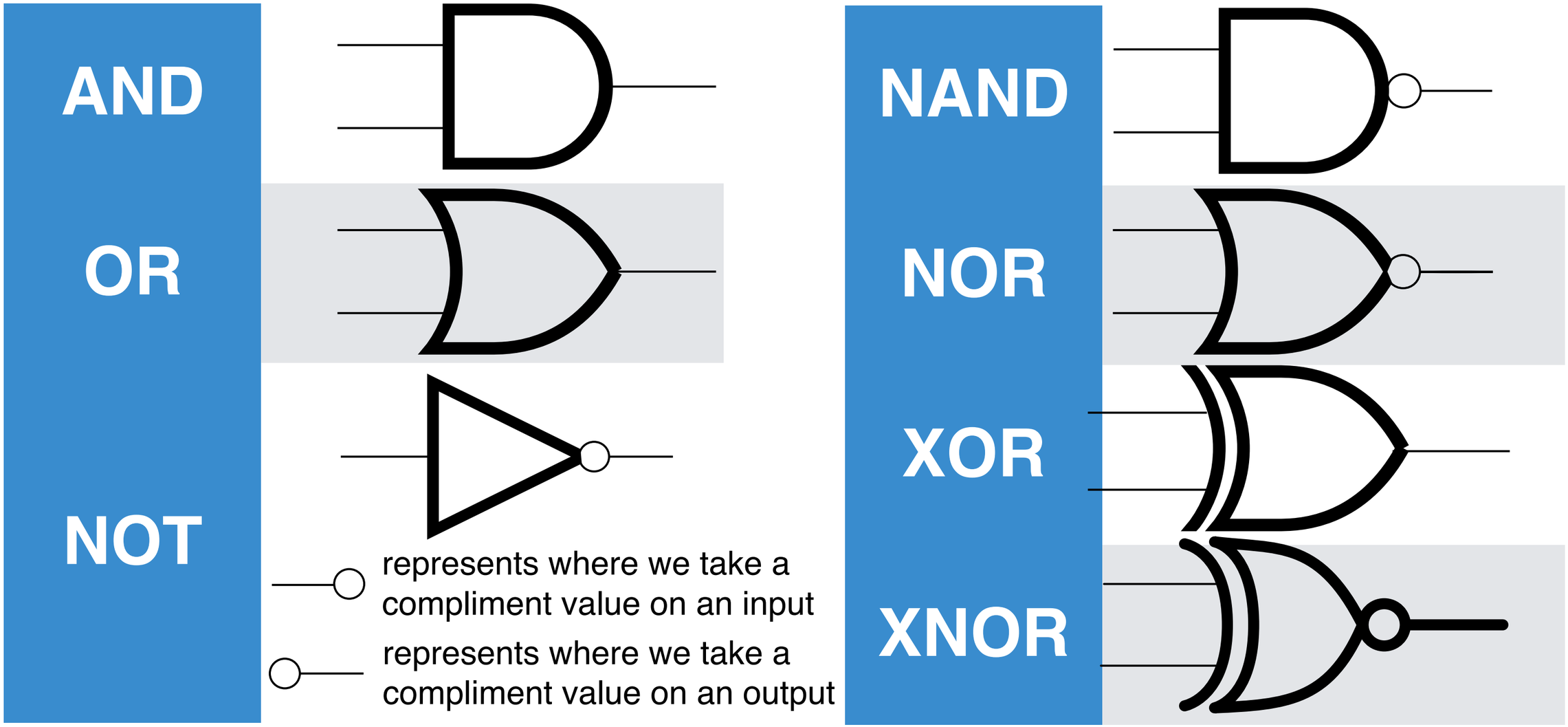
  - XOR ($\oplus$) — $a_1 \oplus a_2 \ldots \oplus a_n$ or $\overline{a_1 \odot a_2 \ldots \odot a_n}$
    - returns 1 if there is an odd number of variables that are 1s
    - otherwise returns 0
    - odd operator

  - XNOR ($\odot$) — $a_1 \odot a_2 \ldots \odot a_n$ or $\overline{a_1 \oplus a_2 \ldots \oplus a_n}$
    - returns 1 if there is an even number of variables that are 1s
    - otherwise returns 0
    - concurrence operator

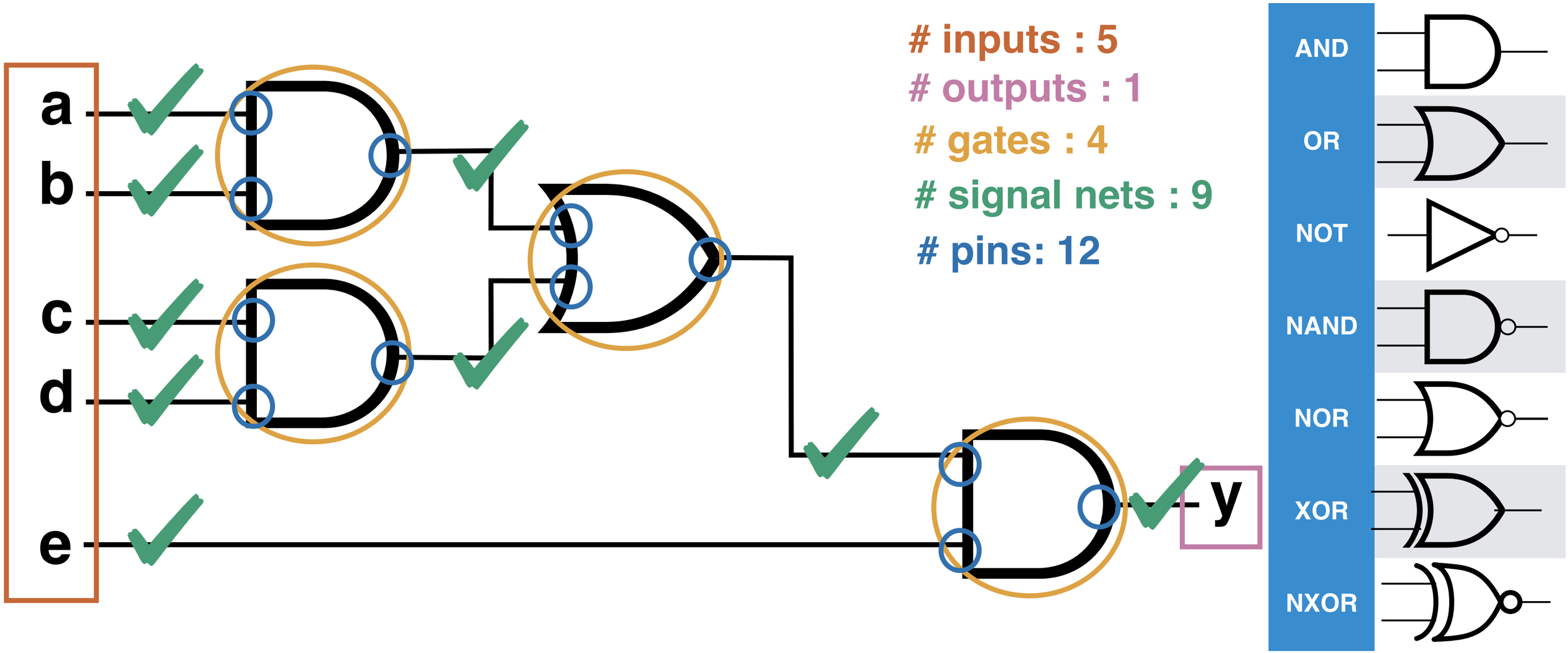
Express Boolean Operators/Functions in Circuit "Gates"

Part of slides are adapted from Hung-Wei Tseng

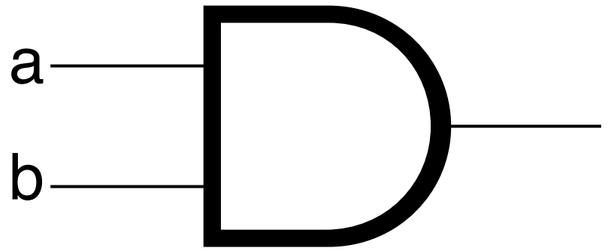
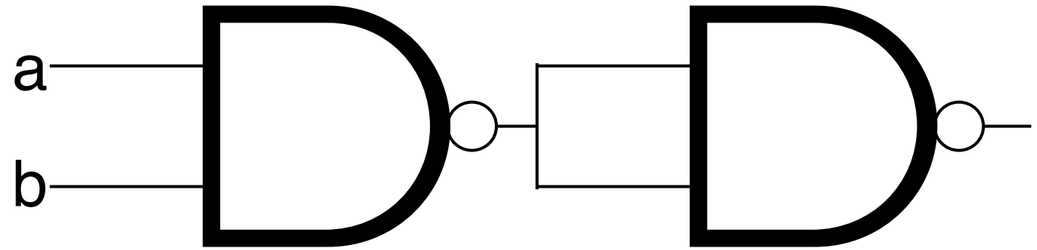
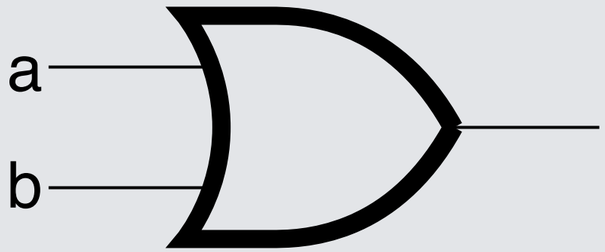
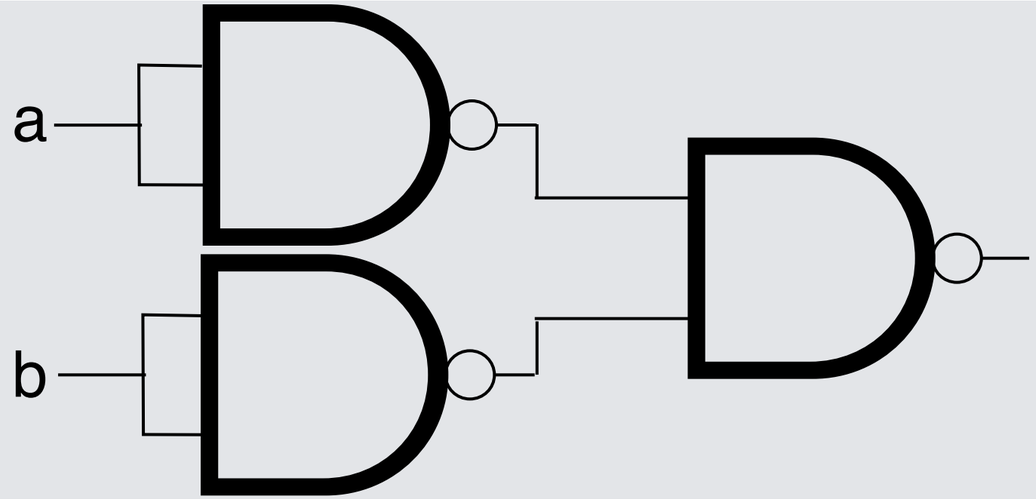
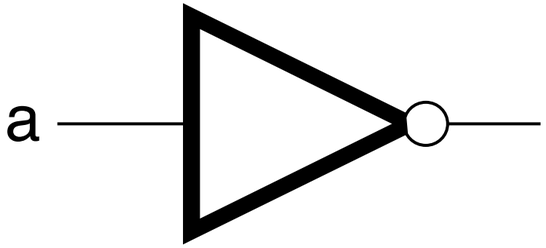
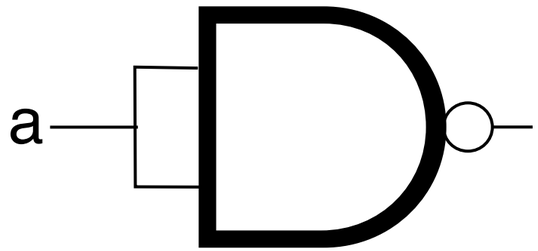
Boolean operators their circuit “gate” symbols



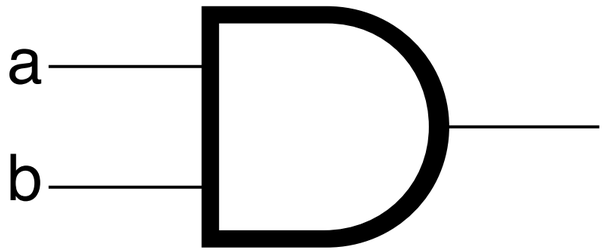
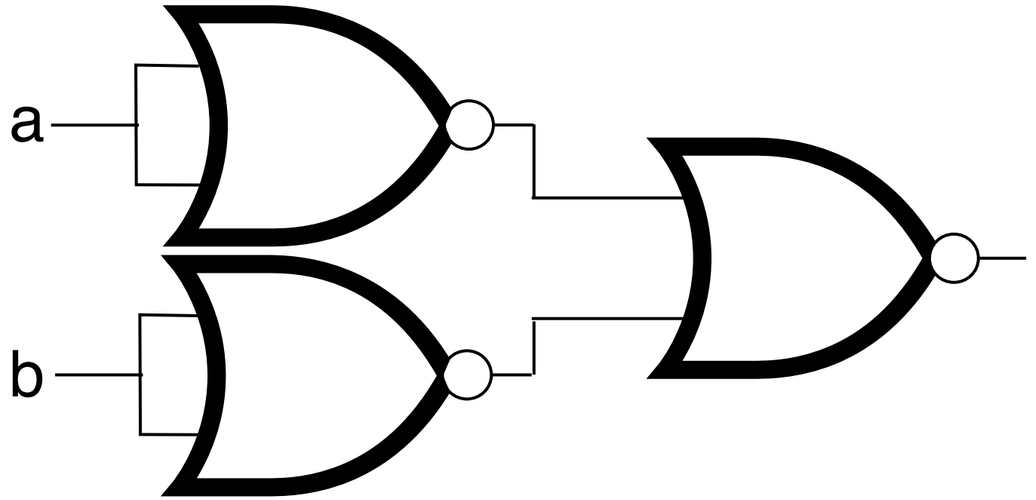
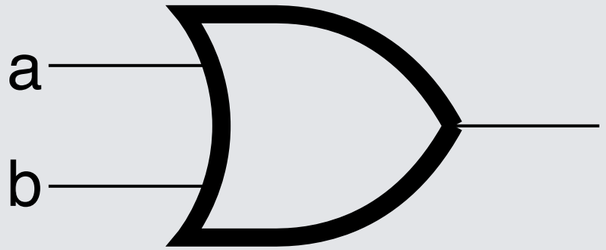
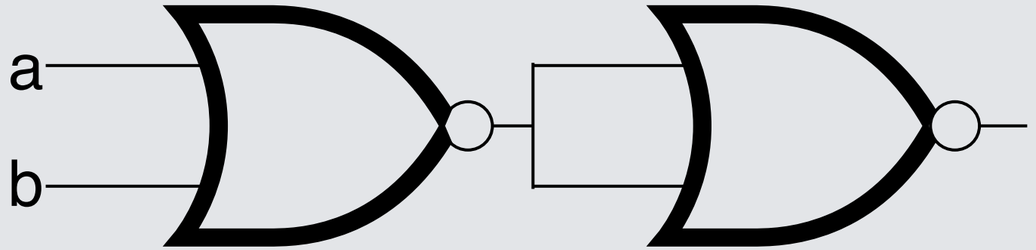
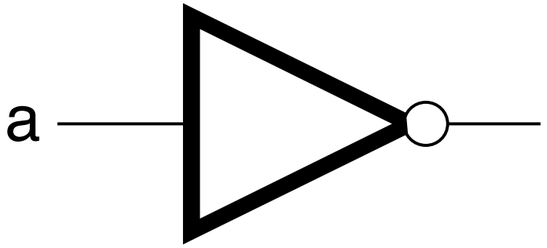
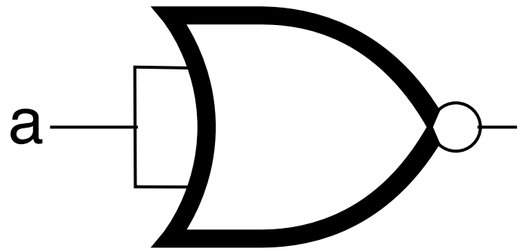
How to express y = e(ab+cd)



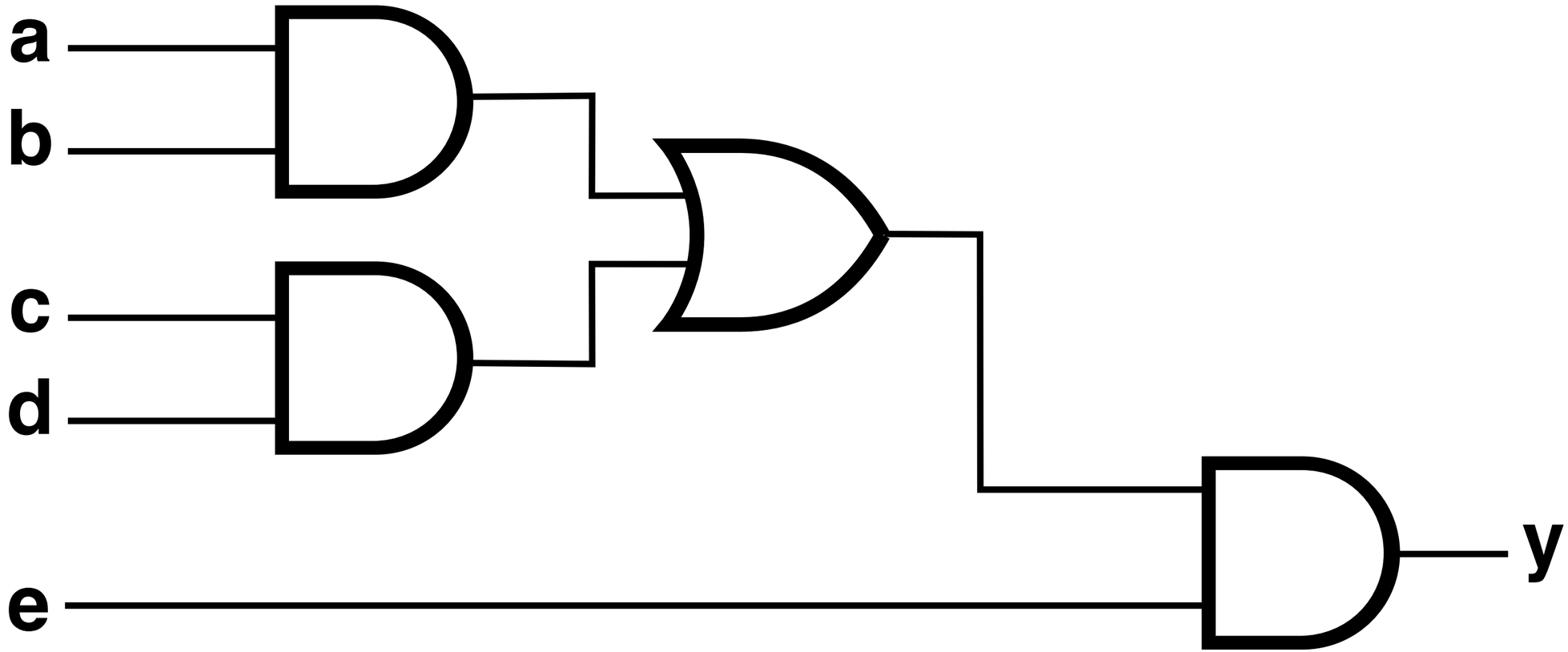
We can make everything NAND!

	Original	NAND
AND		
OR		
NOT		

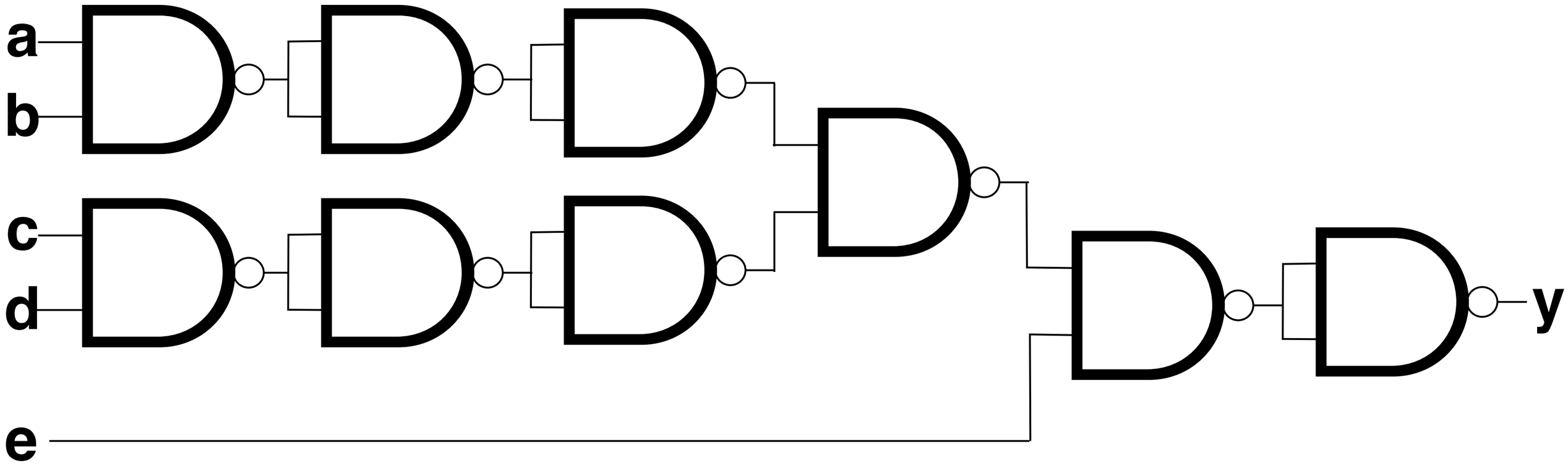
We can also make everything NOR!

	Original	NOR
AND		
OR		
NOT		

How to express y = e(ab+cd)



How to express y = e(ab+cd)

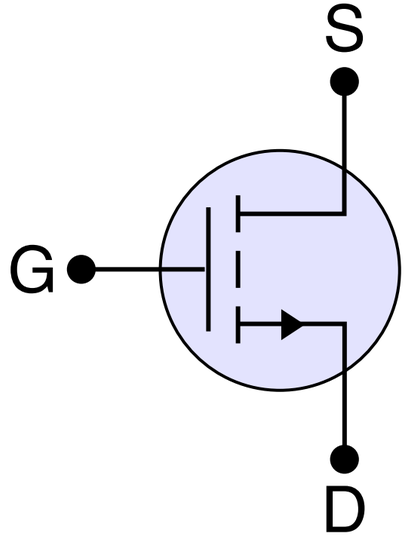


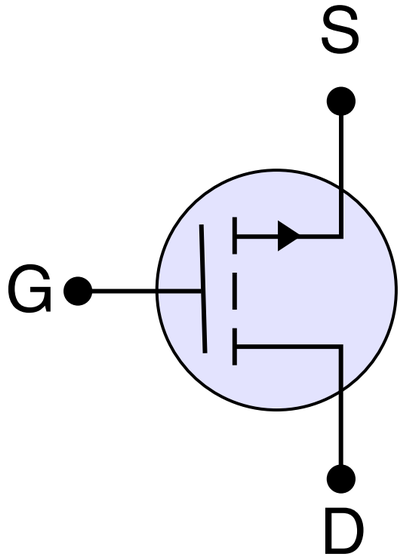
How gates are implemented?

Two type of CMOSs

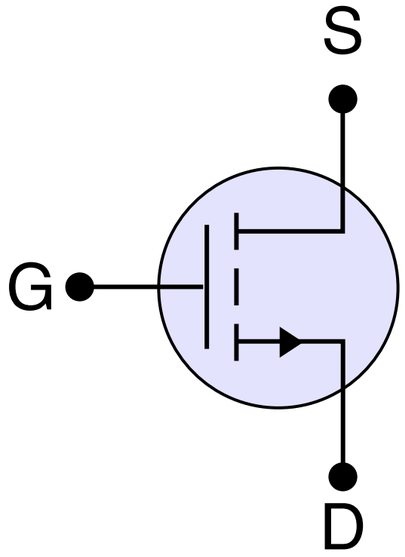
CMOS: complementary metal-oxide-semiconductor

- nMOS
  - Turns on when G = 1
  - When it's on, passes 0s, but not 1s
  - Connect S to ground (0)
  - Pulldown network
- pMOS
  - Turns on when G = 0
  - When it's on, passes 1s, but not 0s
  - Connect S to Vdd (Voltage Drain Drain, positive supply voltage to a circuit, 1)
  - Pullup network

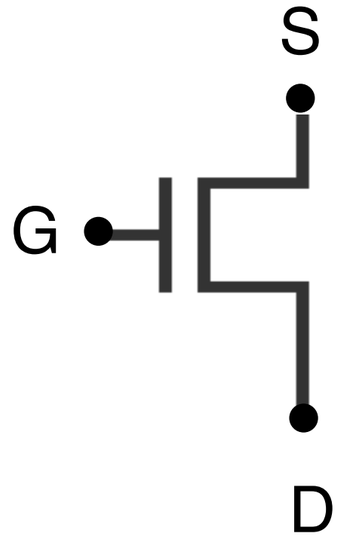


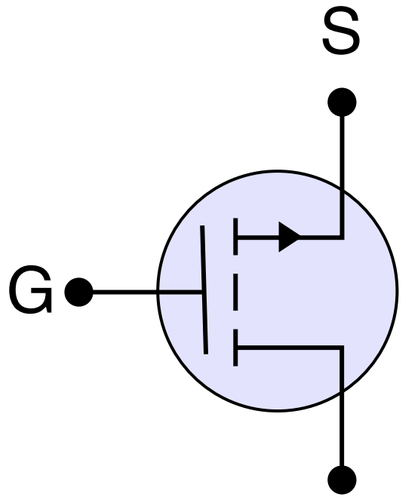


Other symbols for CMOSs

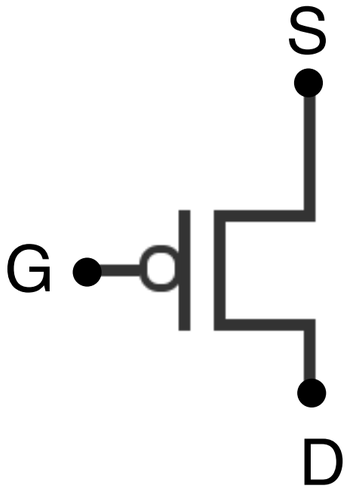


nMOS



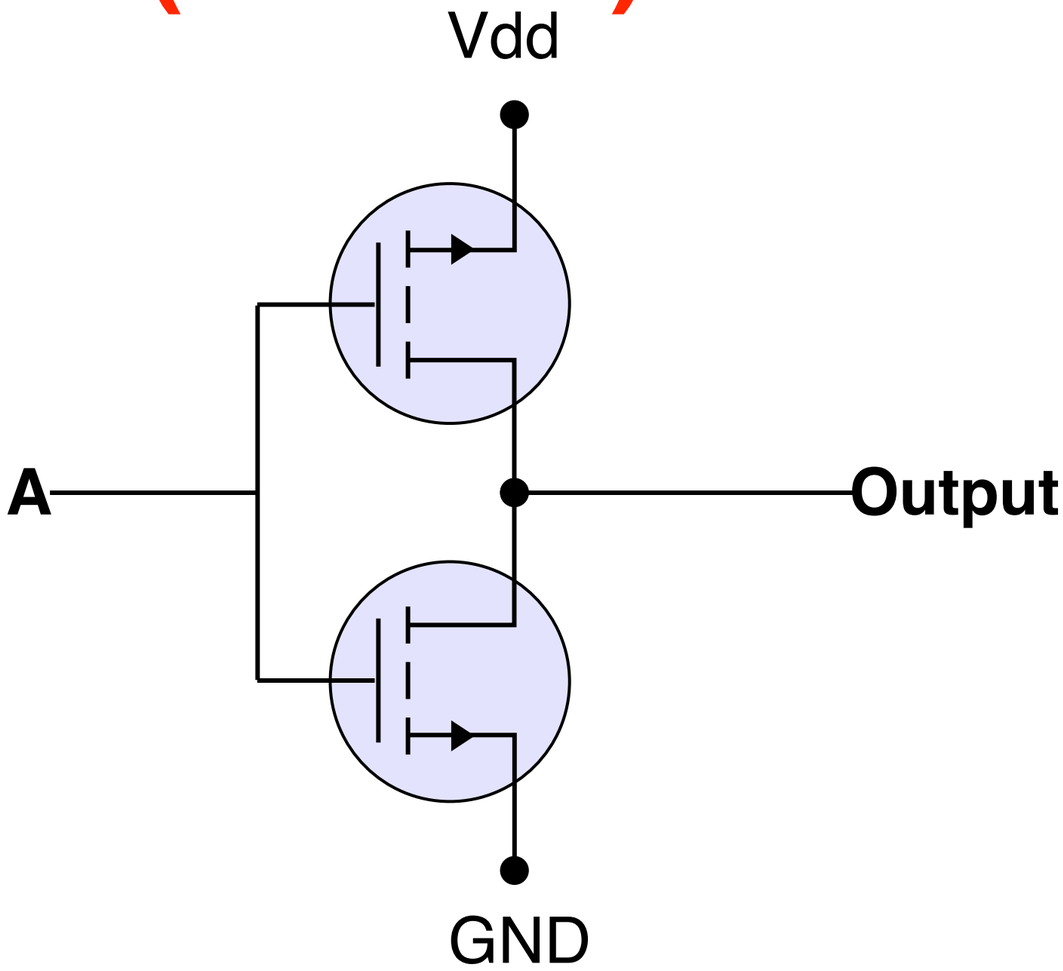


pMOS

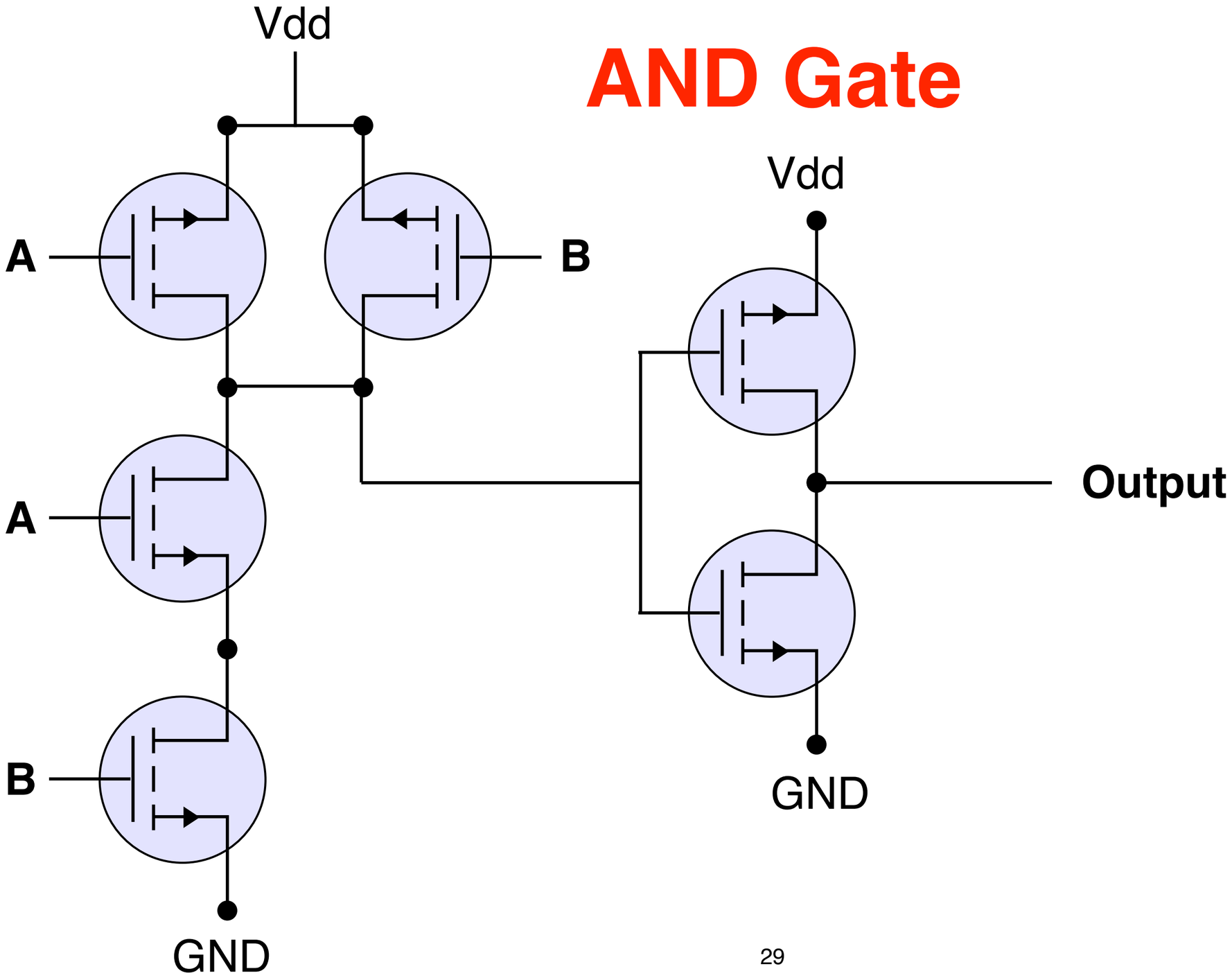


NOT Gate (Inverter)

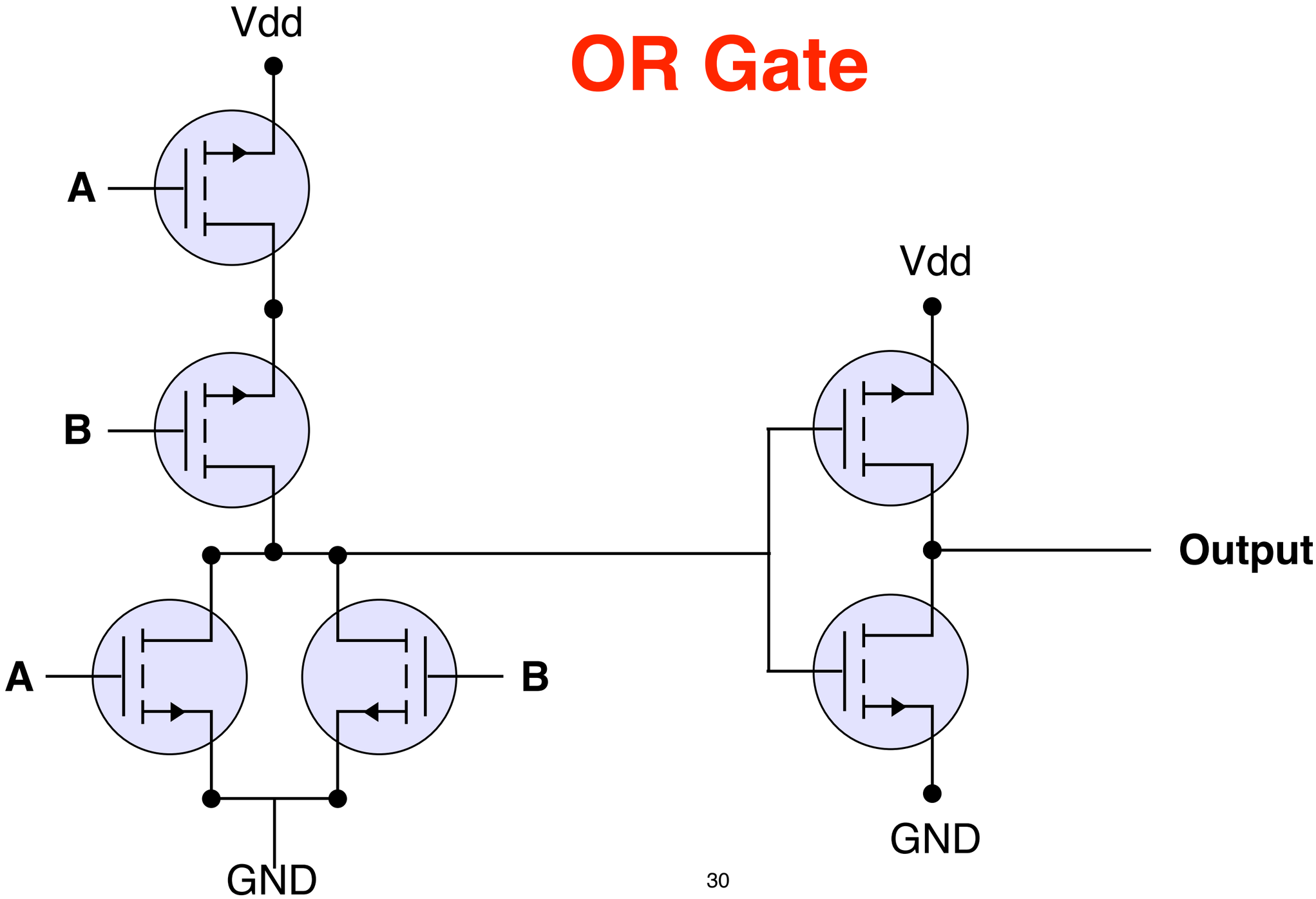
Input A	NMOS (passes 0 when on G=1)	PMOS (passes 1 when on G=0)	Output
0	OFF	ON	1
1	ON	OFF	0



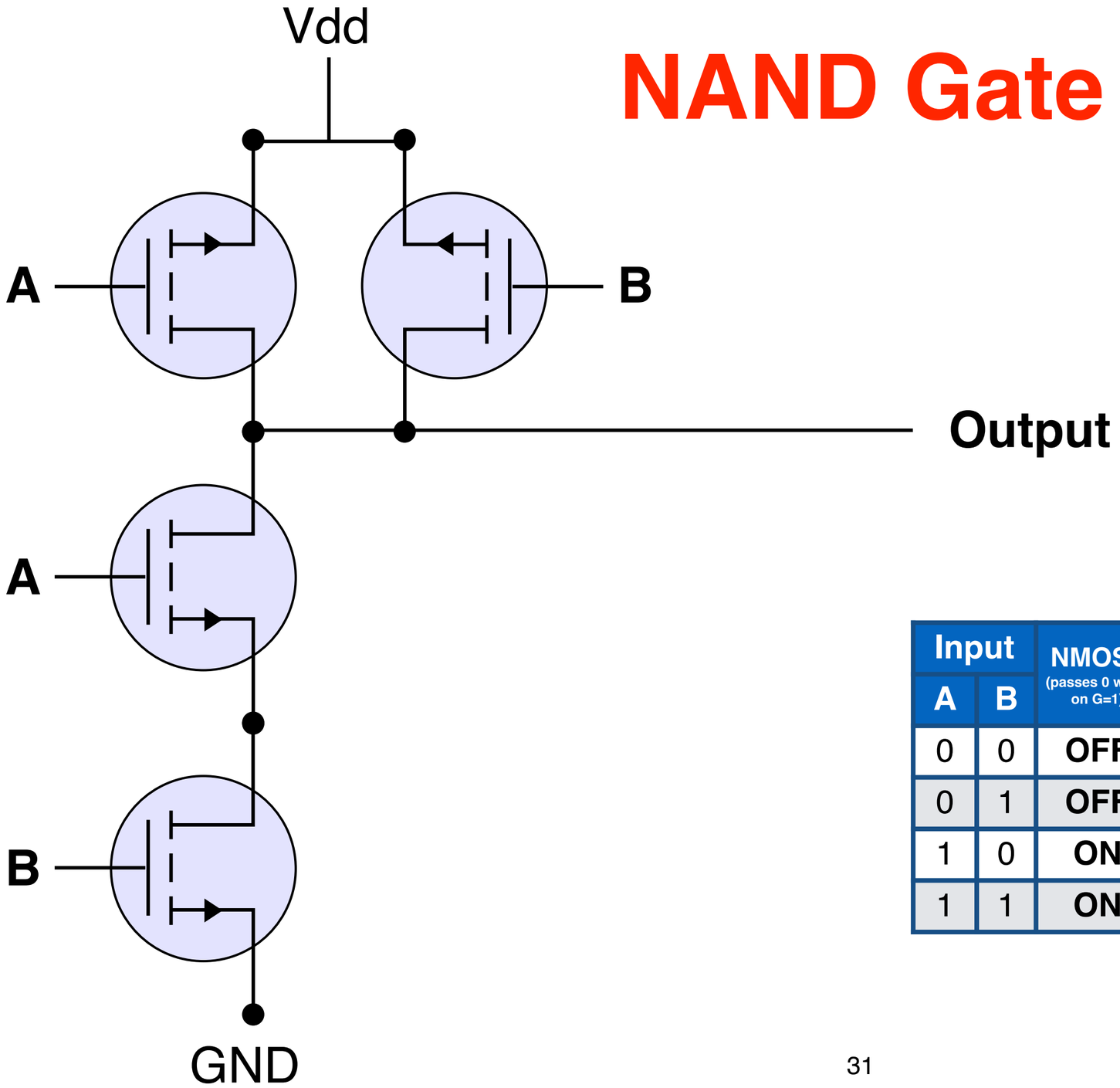
AND Gate



OR Gate



NAND Gate

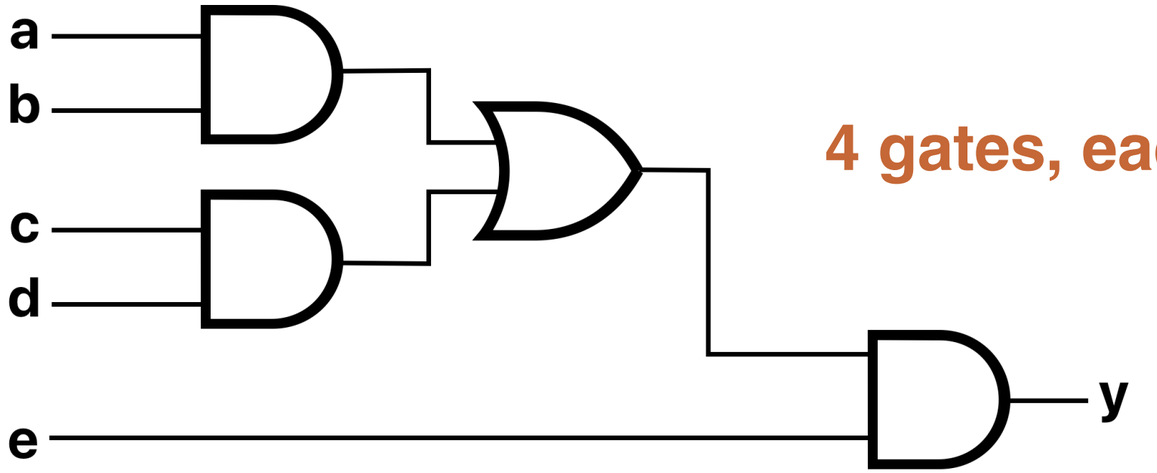


Input		NMOS1 (passes 0 when on G=1)	PMOS1 (passes 1 when on G=0)	NMOS2 (passes 0 when on G=1)	PMOS2 (passes 1 when on G=0)	Output
A	B					
0	0	OFF	ON	OFF	ON	1
0	1	OFF	ON	ON	OFF	1
1	0	ON	OFF	OFF	ON	1
1	1	ON	OFF	ON	OFF	0

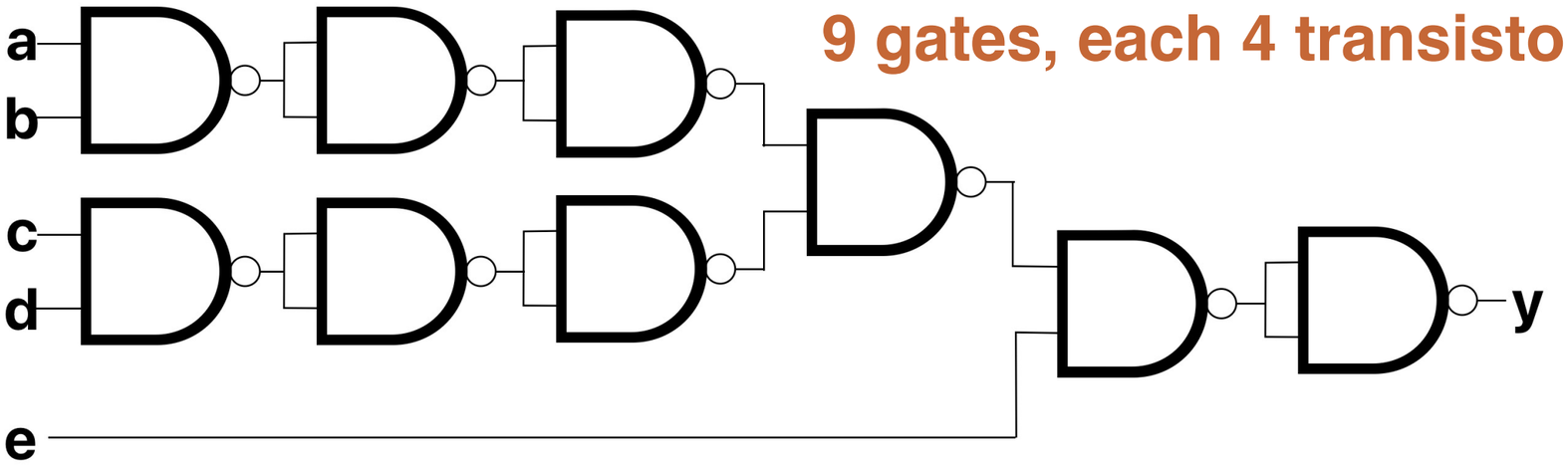
Why use NAND?

- NAND and NOR are “universal gates” — you can build any circuit with everything NAND or NOR
- Simplifies the design as you only need one type of gate
- NAND only needs 4 transistors — gate delay is smaller than OR/AND that needs 6 transistors
- NAND is slightly faster than NOR due to the physics nature

How about total number of transistors?



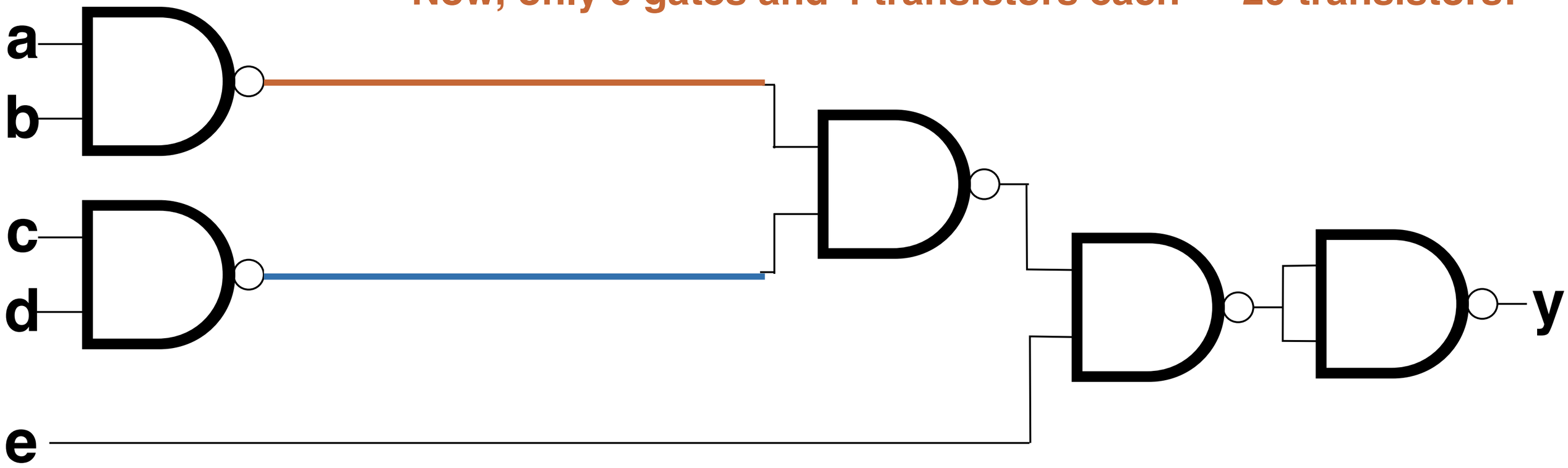
4 gates, each 6 transistors : total 24 transistors



9 gates, each 4 transistors : total 36 transistors

However …

Now, only 5 gates and 4 transistors each — 20 transistors!



How big is the truth table of y = e(ab+cd)

- How many rows do we need to express the circuit represented by y = e(ab+cd)?
  A. 5
  B. 9
  C. 25
  D. 32
  E. 64

How big is the truth table of y = e(ab+cd)

- How many rows do we need to express the circuit represented by y = e(ab+cd)?

A. 5

B. 9

C. 25

D. 32

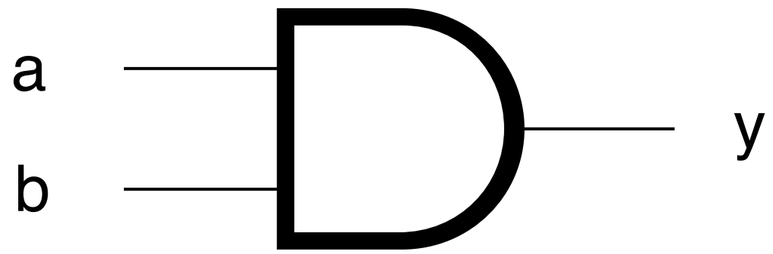
E. 64

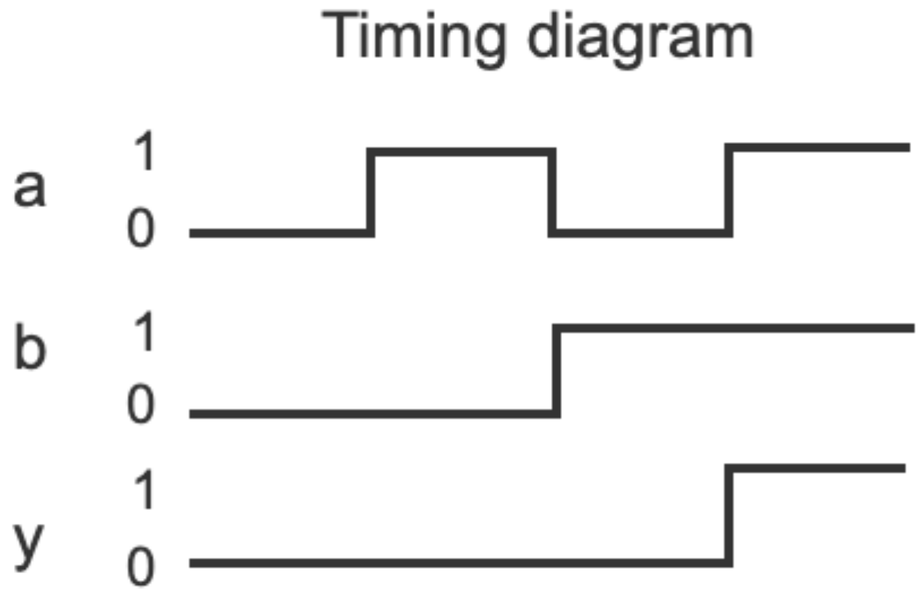
$$2 \times 2 \times 2 \times 2 \times 2 = 2^5 = 32$$

Boolean expression is a lot more compact than a truth table!

Timing diagram

- A timing diagram graphically shows a circuit's output values for given input values that change over time. Each signal (input or output) name is listed on the left. Time proceeds to the right. Each signal is drawn as a high line (1) or a low line (0).





Can We Get the Boolean Equation from a Truth Table?

Definitions of Boolean Function Expressions

- Complement: variable with a bar over it or a ’ — A’, B’, C’
- Literal: variable or its complement — A, A’, B, B’, C, C’
- Implicant (Product term): product of literals — ABC, AC, BC
- Implicate (Sum terms): sum of literals — (A+B+C), (A+C), (B+C)
- Minterm: AND that includes all input variables — ABC, A’BC, AB’C
- Maxterm: OR that includes all input variables — (A+B+C), (A’+B+C), (A’+B’+C)

Minterms and Maxterms

Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \overline{x}_1\overline{x}_2 x_3$	$M_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$m_2 = \overline{x}_1 x_2\overline{x}_3$	$M_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$m_3 = \overline{x}_1 x_2 x_3$	$M_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$m_4 = x_1\overline{x}_2\overline{x}_3$	$M_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$m_5 = x_1\overline{x}_2 x_3$	$M_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$m_6 = x_1 x_2\overline{x}_3$	$M_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$m_7 = x_1 x_2 x_3$	$M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

Canonical form — Sum of “Minterms”

Input		Output
X	Y	
0	0	**0**
0	1	**0**
1	0	**1**
1	1	**1**

A minterm

$$f(X,Y) = XY' + XY$$

Sum (OR) of “product” terms

XNOR

Input		Output
A	B	
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

$$f(A,B) = A'B' + AB$$

Canonical form — Product of “Maxterms”

A maxterm

Input		Output
X	Y	
0	0	**0**
0	1	**0**
1	0	**1**
1	1	**1**

$f(X,Y) = (X+Y)\,(X + Y')$ ← Product of maxterms

XNOR

Input		Output
A	B	
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

$f(A,B) = (A+B')\,(A'+B)$

Sum-of-Products Form

- Represent a function f by a sum of minterms
- Each minterm is ANDed with the value of f

Row number	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$f = m_1 + m_4 + m_5 + m_6$$

$$= \sum(m_1, m_4, m_5, m_6)$$

$$= \sum m(1, 4, 5, 6)$$

$$= \sum_{x_1, x_2, x_3}(1, 4, 5, 6)$$

$$= x_1' x_2' x_3 + x_1 x_2' x_3' + x_1 x_2' x_3 + x_1 x_2 x_3'$$

- A logic expression in the sum-of-products (SoP) form:
  - Consisting of product (AND) terms that are summed (ORed)
- Canonical SoP: each product term is a minterm

Product-of-Sums Form

Row number	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$f = \mathrm{M}_0\,\mathrm{M}_2\,\mathrm{M}_3\,\mathrm{M}_7$$

$$=\Pi\,(\mathrm{M}_0,\ \mathrm{M}_2,\ \mathrm{M}_3,\ \mathrm{M}_7)$$

$$=\Pi\,\mathrm{M}(0, 2, 3, 7)$$

$$=\Pi_{x_1, x_2, x_3}(0, 2, 3, 7)$$

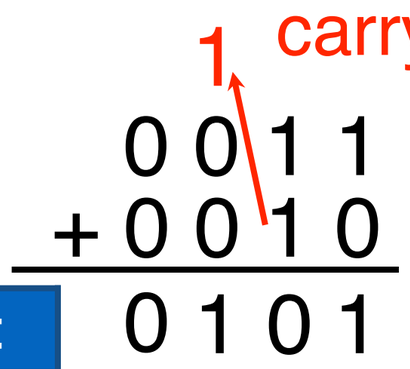
$$=(x_1 + x_2 + x_3)(x_1 + x_2' + x_3)(x_1 + x_2' + x_3')(x_1' + x_2' + x_3')$$

- A logic expression in the product-of-sums (PoS) form:
  - Consisting of sum (OR) terms that are the factors of a logical product
- Canonical PoS: each sum term is a maxterm

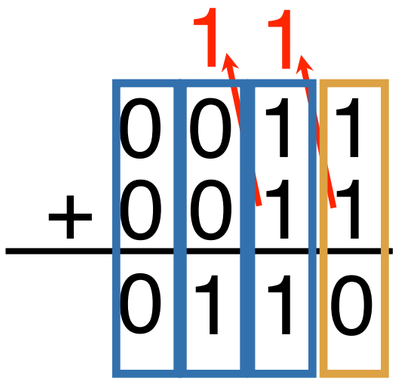
Let’s design a circuit!

Binary addition

3 + 2 = 5



3 + 3 = 6



full adder — adder with a carry as an input

Input			Output	
A	B	Cin	Out	Cout
0	0	0	**0**	**0**
0	1	0	**1**	**0**
1	0	0	**1**	**0**
1	1	0	**0**	**1**
0	0	1	**1**	**0**
0	1	1	**0**	**1**
1	0	1	**0**	**1**
1	1	1	**1**	**1**

half adder — adder without a carry as an input

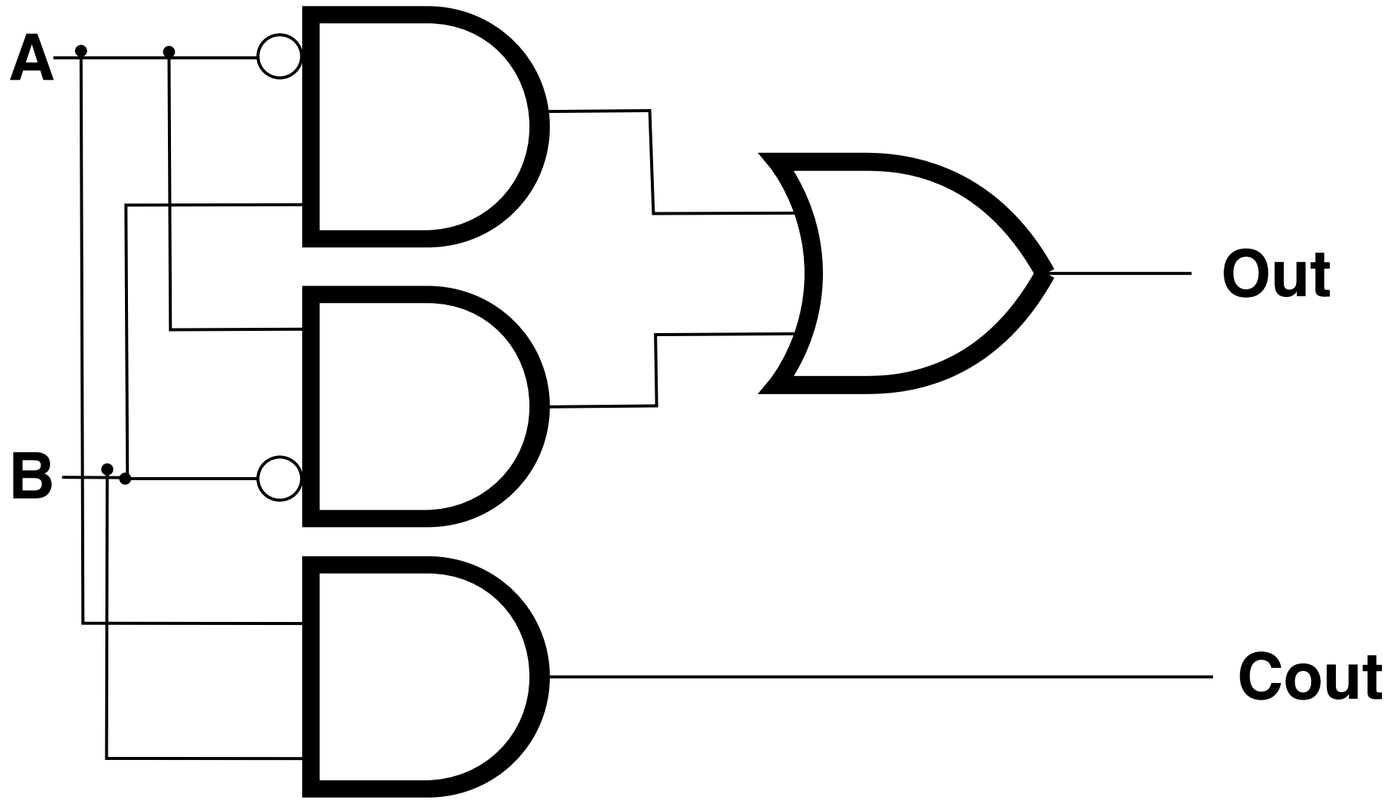
Input		Output	
A	B	Out	Cout
0	0	**0**	**0**
0	1	**1**	**0**
1	0	**1**	**0**
1	1	**0**	**1**

Half adder

Input		Output	
A	B	Out	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Out = A'B + AB'

Cout = AB



The sum-of-product form of the full adder

- How many of the following minterms are part of the sum-of-product form of the full adder in generating the output bit?
  - ① A’B’Cin’
  - ② A’BCin’
  - ③ AB’Cin’
  - ④ ABCin’
  - ⑤ A’B’Cin
  - ⑥ A’BCin
  - ⑦ AB’Cin
  - ⑧ ABCin
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Input			Output	
A	B	Cin	Out	Cout
0	0	0	**0**	**0**
0	1	0	**1**	**0**
1	0	0	**1**	**0**
1	1	0	**0**	**1**
0	0	1	**1**	**0**
0	1	1	**0**	**1**
1	0	1	**0**	**1**
1	1	1	**1**	**1**

The sum-of-product form of the full adder

- How many of the following minterms are part of the sum-of-product form of the full adder in generating the output bit?

① A’B’Cin’
② A’BCin’
③ AB’Cin’
④ ABCin’
⑤ A’B’Cin
⑥ A’BCin
⑦ AB’Cin
⑧ ABCin

A. 0
B. 1
C. 2
D. 3
E. 4

Input			Output	
A	B	Cin	Out	Cout
0	0	0	**0**	**0**
0	1	0	**1**	**0**
1	0	0	**1**	**0**
1	1	0	**0**	**1**
0	0	1	**1**	**0**
0	1	1	**0**	**1**
1	0	1	**0**	**1**
1	1	1	**1**	**1**

Out = A’BCin’+ AB’Cin’+ A’B’Cin+ ABCin
Cout = ABCin’ + A’BCin + AB’Cin + ABCin

sum-of-products/product-of-sums

- They can be used interchangeably
- Depends on if the truth table has more 0s or 1s in the result
- Neither forms give you the “optimized” equation. By optimized, we mean — minimize the number of operations

Can we simplify these functions?

Sum-of-minterms

- Different equations may represent the same function. Ex: y = a + b, and y = a + a'b, represent the same function. The sameness is not obvious, so a standard equation form is desirable.
- A **canonical form** of a Boolean equation is a standard equation form for a function.
- **Sum-of-minterms** form is a canonical form of a Boolean equation where the right-side expression is a **sum-of-products** with each product a unique minterm.
- A **minterm** is a product term having **exactly one literal** for **every** function **variable**.
- A **literal** is a variable appearance, in true or complemented form, in an expression, such as b, or b'.

Laws in Boolean Algebra

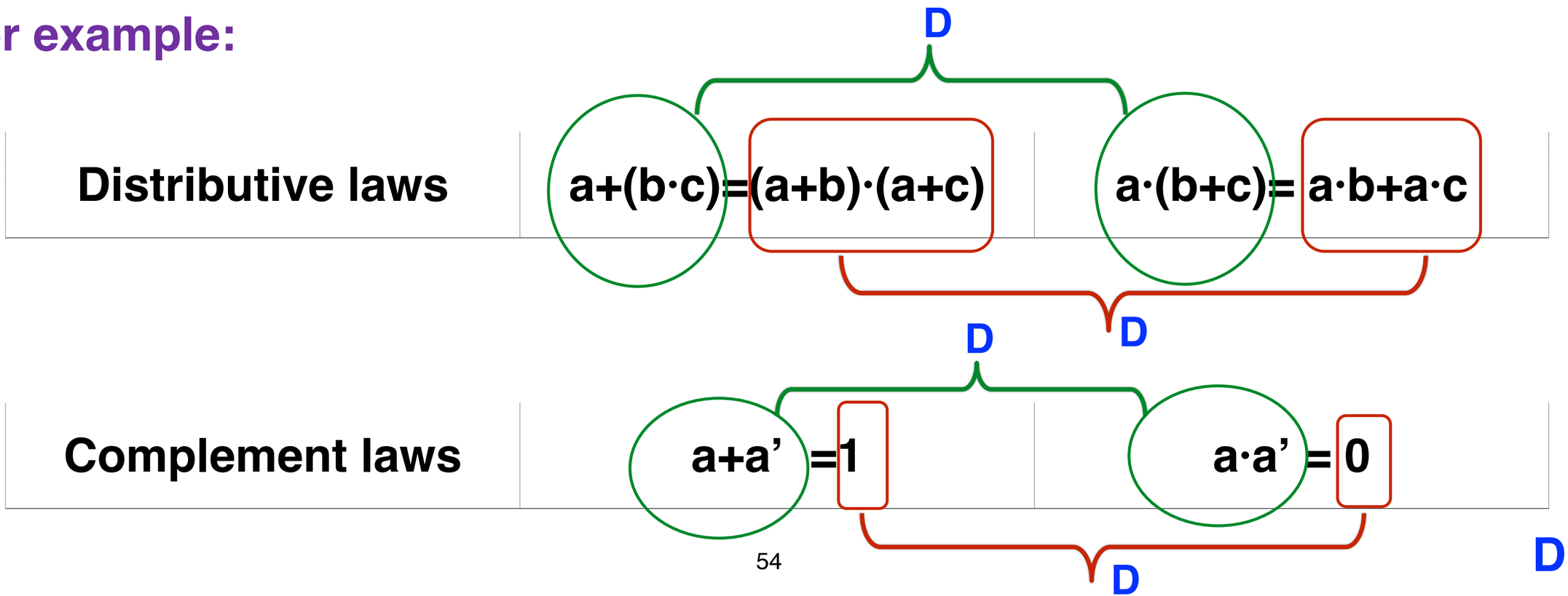
	OR	AND
Associative laws	(a+b)+c=a+(b+c)	(a·b) ·c=a·(b·c)
Commutative laws	a+b=b+a	a·b=b·a
Distributive laws	a+(b·c)=(a+b)·(a+c)	a·(b+c)=a·b+a·c
Identity laws	a+0=a	a·1=a
Complement laws	a+a’=1	a·a’=0

Duality

Duality: We swap all operators between (**+** , **·**)
and interchange all elements between (**0** , **1**).

For a theorem if the statement can be proven with the laws of Boolean algebra, then the duality of the statement is also true.

For example:



D: dual

Some more tools

	OR	AND
DeMorgan's Theorem	(a + b)' = a'b'	(ab)' = a' + b'
Covering Theorem	a(a+b) = a+ab = a	ab + ab' = (a+b)(a+b') = a
Consensus Theorem	ab+ac+b'c = ab+b'c	(a+b)(a+c)(b'+c) = (a+b)(b'+c)
Uniting Theorem	a (b + b') = a	(a+b)·(a+b')=a
Shannon's Expansion	f(a,b,c) = a f(1, b, c) + a' f(0,b,c)	

Another useful tool

$$a + a'b = a + b$$

$$a(a' + b) = ab$$

Proof:

$$\begin{aligned} a + a'b &= a\cdot 1 + a'b \\ &= a(1+b) + a'b \\ &= a + ab + a'b \\ &= a + (ab + a'b) \\ &= a + b(a + a') \\ &= a + b\cdot 1 \\ &= a + b \end{aligned}$$

Based on duality, one can prove that $a(a' + b) = ab$.

Applying Theorems

- Which of the following represents CB+BA+C'A?
  A. AB+AC'
  B. BC+AC'
  C. AB+BC
  D. AB+AC
  E. None of the other choices

	OR	AND
Associative laws	(a+b)+c=a+(b+c)	(a·b) ·c=a·(b·c)
Commutative laws	a+b=b+a	a·b=b·a
Distributive laws	a+(b·c)=(a+b)·(a+c)	a·(b+c)=a·b+a·c
Identity laws	a+0=a	a·1=a
Complement laws	a+a'=1	a·a'=0
DeMorgan's Theorem	(a + b)' = a'b'	(ab)' = a' + b'
Covering Theorem	a(a+b) = a+ab = a	ab + ab' = (a+b)(a+b') = a
Consensus Theorem	ab+ac+b'c = ab+b'c	(a+b)(a+c)(b'+c) = (a+b)(b'+c)
Uniting Theorem	a (b + b') = a	(a+b)·(a+b')=a
Shannon's Expansion	f(a,b,c) = a f(1, b, c) + a' f(0,b,c)	

Applying Theorems

- Which of the following represents BC+BA+C'A?
  - A. AB+AC'
  - B. BC+AC'
  - C. AB+BC
  - D. AB+AC
  - E. None of the other choices

Consensus Theorem	ab+ac+b'c = ab+b'c

CB + BA1 + C'A
= BC + BA (C'+C) + C'A
= BC + BAC' + BAC + C'A
= (1+A)BC + (B+1)AC'
= BC + AC'

SoP simplification

- For the truth table shown on the right, please simplify the SoP as much as possible.

Input			Output
A	B	C	
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	0	0	**1**
1	1	1	**0**

SoP simplification

- For the truth table shown on the right, please simplify the SoP as much as possible.

F(A, B, C) =

A’B’C’+ A’B’C+ A’BC’+ A’BC+ AB’C’+ ABC’

= A’B’(C’+C)+ A’B(C’+C)+ AC’(B’+B)

= A’B’+ A’B + AC’

= A’ + AC’ = A’(1+C’)+AC’ Distributive Laws

= A’ + A’C’ + AC’

= A’ + (A’+A)C’

= A’ + C’

Input			Output
A	B	C	
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**