Boolean Logic

Digital Systems 2022/23

Dr James Stovold

Boolean Algebra

(George Boole, 1815–64)

- Foundation of logical operations within computers.
- Variables are either True (T) or False (F).
  - Mutually-exclusive characters map nicely to binary numbers!
- Fundamental operations are conjunction (and), disjunction (or), negation (not).

Conjunction
(AND)

- Conjunction is represented by ∧ · or <nothing> (& or && in C)
  - Similar to product in elementary algebra

$$Q = AB = A \cdot B$$

$$Q = A \wedge B$$

Truth table

A	B	Q
F	F	F
F	T	F
T	F	F
T	T	T

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Also a truth table

Disjunction
(OR)

- Disjunction is represented by ∨ or + (| or || in C)
  - Similar to addition in elementary algebra

$$Q = A + B$$
$$Q = A \vee B$$

A	B	Q
F	F	F
F	T	T
T	F	T
T	T	T

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Negation
(NOT)

- Negation is represented by ¬ ¯ or ' (~ or ! in C)
  - $\bar{X}$ is pronounced "X bar", X' is "X prime"

$$Q = \neg A$$
$$Q = A' = \bar{A}$$

A	Q
F	T
T	F

A	Q
0	1
1	0

Negated Operators

(NAND, NOR)

- Represented by a combination:
  - NAND (NOT AND), e.g. NOT(X AND Y), $\overline{XY}$
  - NOR (NOR OR), e.g. NOT(X OR Y), $\overline{X+Y}$

$$Q = \neg(A \wedge B)$$

A	B	AB	Q
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

$$Q = \neg(A \vee B)$$

A	B	A + B	Q
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Exclusive-OR/-NOR
(XOR, XNOR)

- Exclusive-OR is represented by $\oplus$ (^ in C)
- Exclusive-NOR is represented by $\odot$

$$Q = A \oplus B$$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

$$Q = A \odot B$$

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

Quick aside: Truth Tables

- The truth table is a useful technique for mapping out an entire function.
- We list all possible permutations of the input variables and calculate the output variables.
- Particularly useful for working through a complicated equation.

Truth Table Examples

$$Q = AB + C$$

A	B	C	AB	Q
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Truth Table Examples

$$Q = AB + C$$

A	B	C	AB	Q
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Truth Table Examples

$$Q = AB + C$$

A	B	C	AB	Q
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	
1	1	1	1	

Truth Table Examples

$$Q = AB + C$$

A	B	C	AB	Q
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	
1	1	1	1	

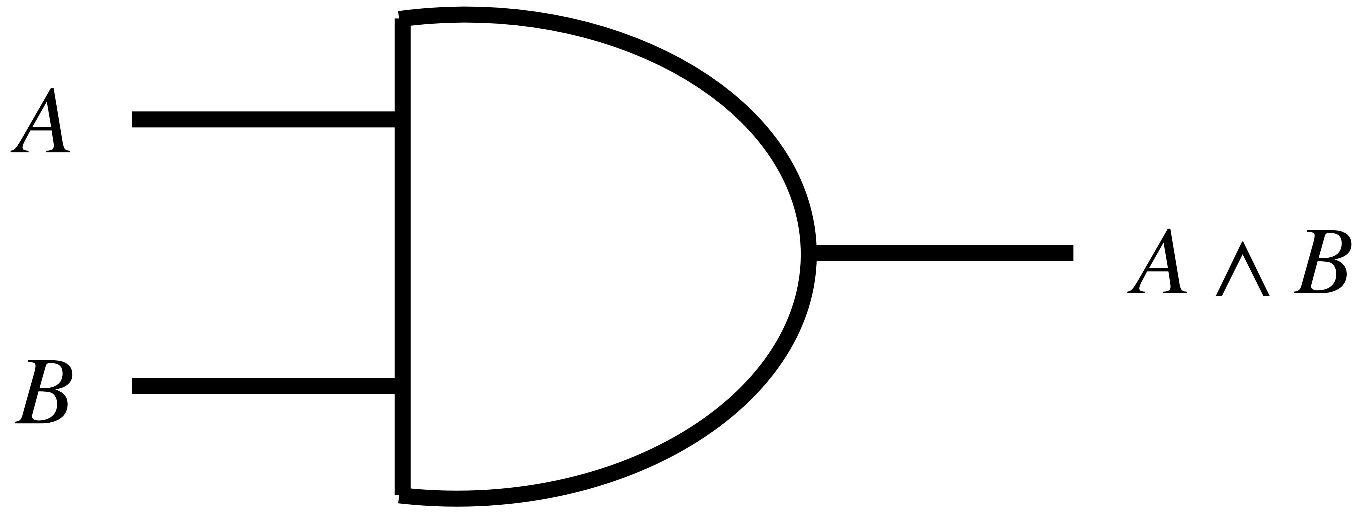
Truth Table Examples

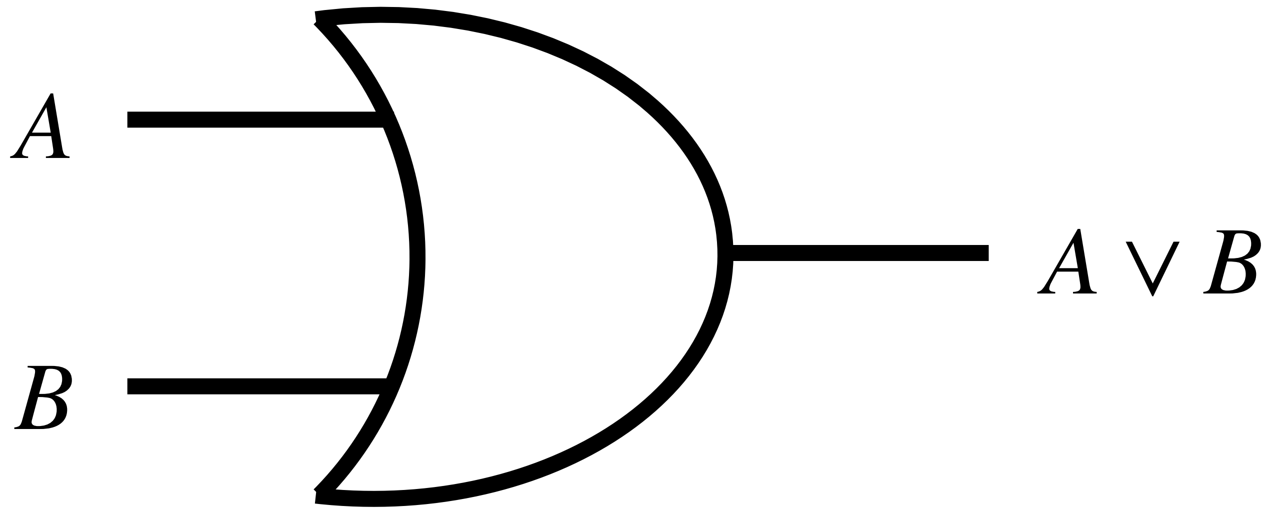
$$Q = AB + C$$

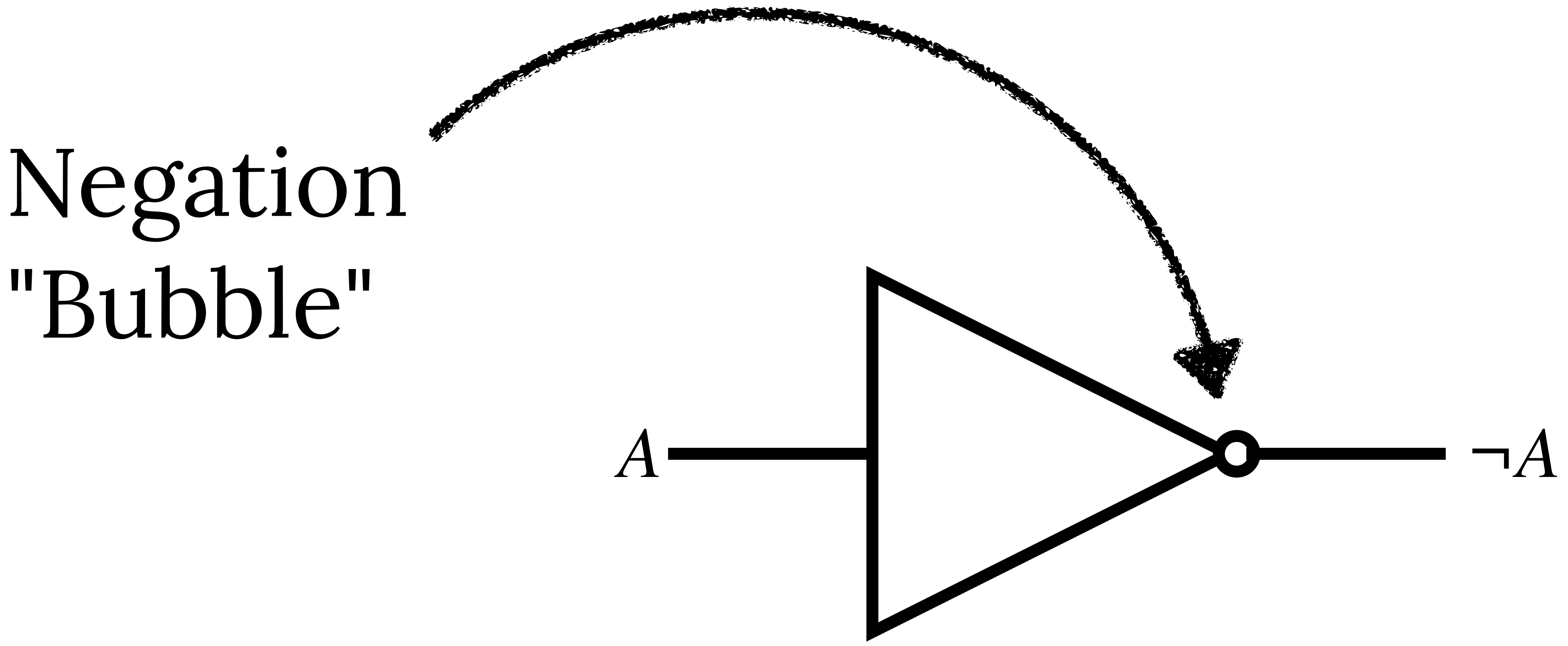
A	B	C	AB	Q
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

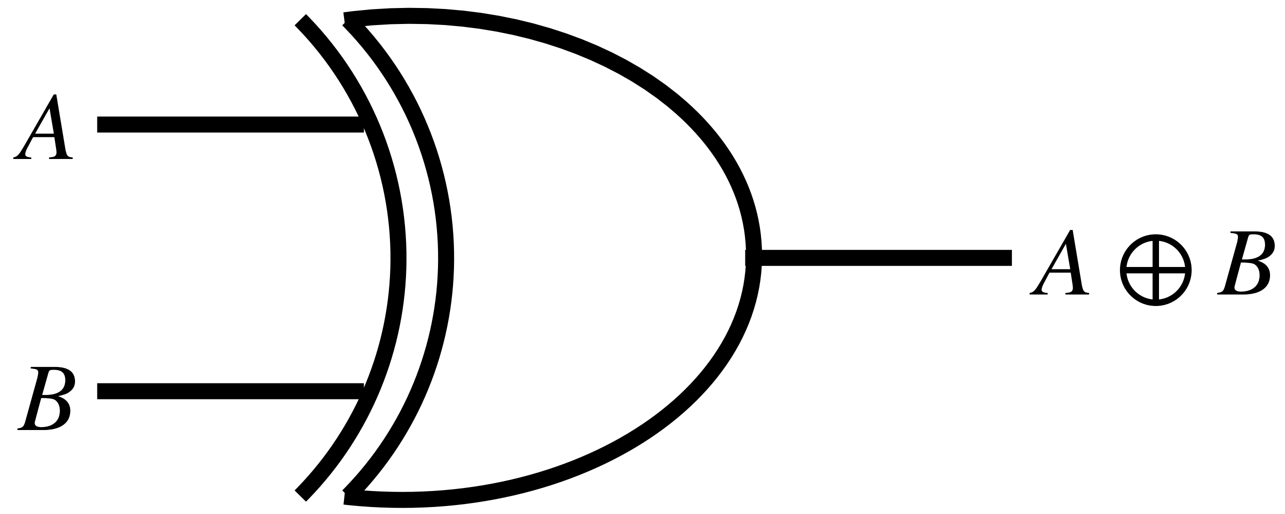
Logic Gates

- Each of our operators has a conceptual electronic component/symbol:



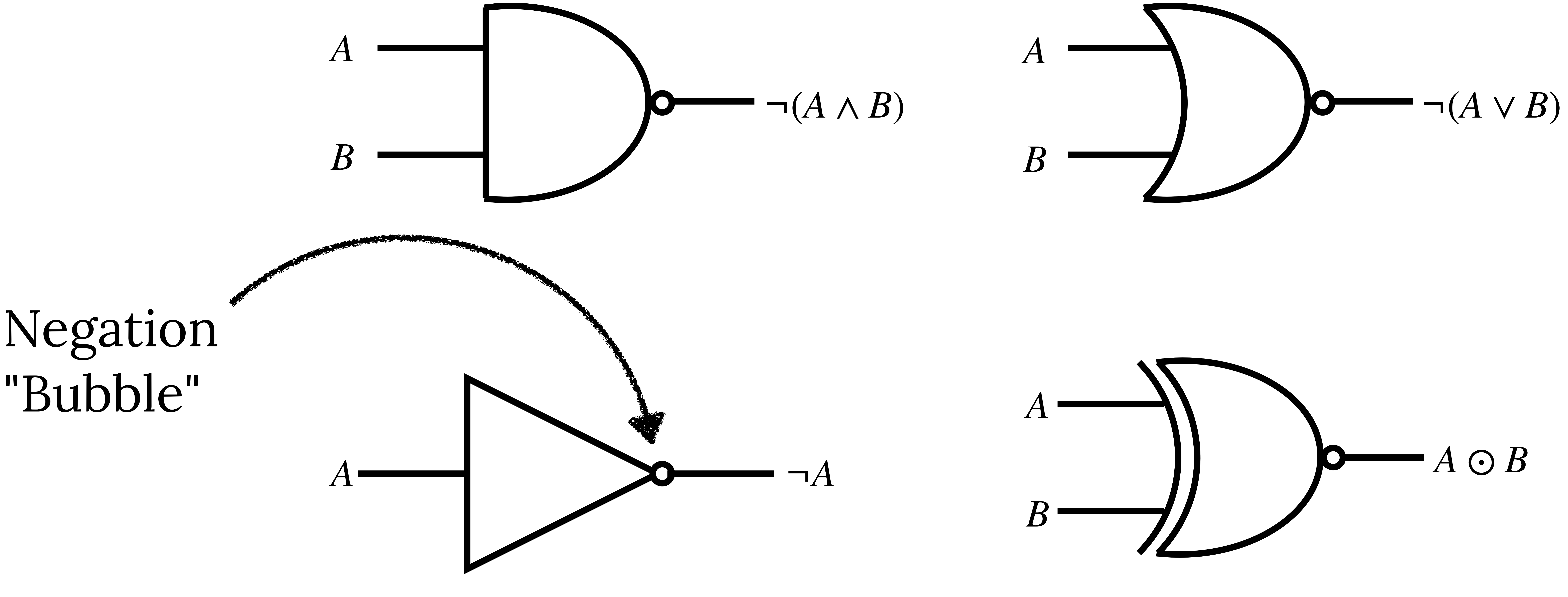






Logic Gates

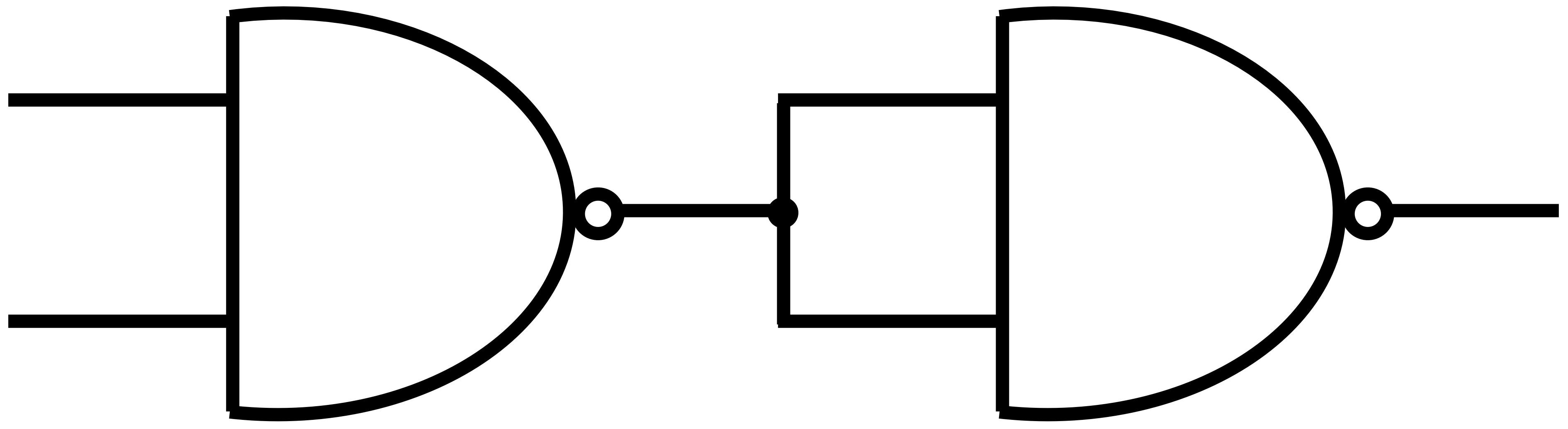
- Each of our operators has a conceptual electronic component/symbol:

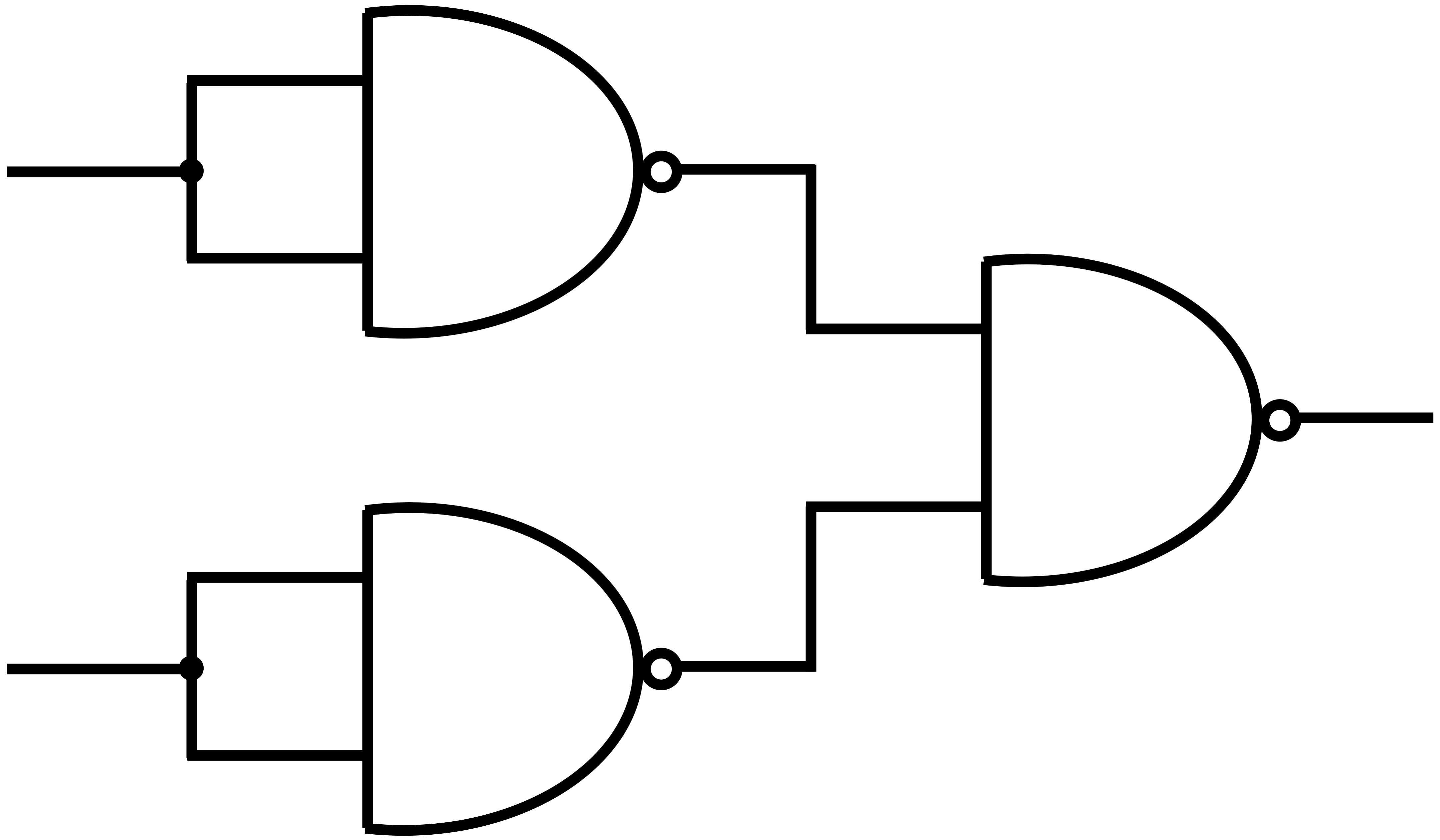


Universal Gates

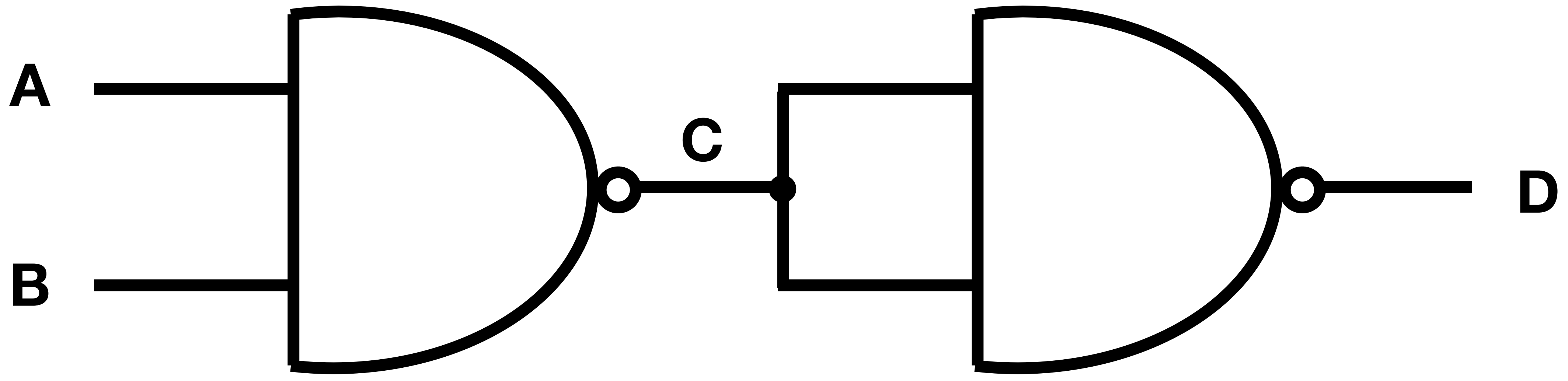
- NAND and NOR gates are considered "universal" gates
- Any binary logic can be built just from NAND gates, or just from NOR gates.
    - Remember: simple is cheap, simple is good!

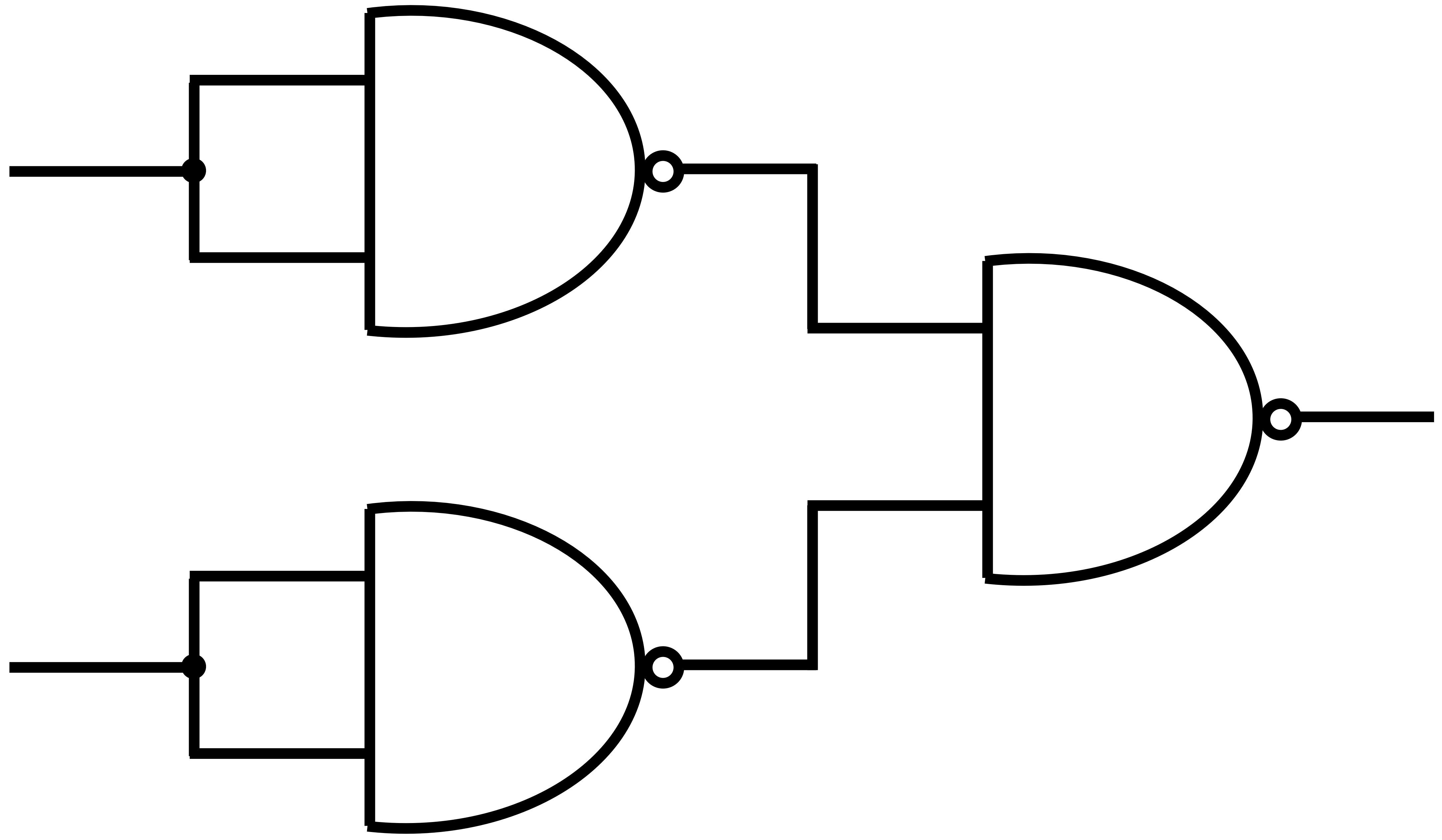
Using NANDs



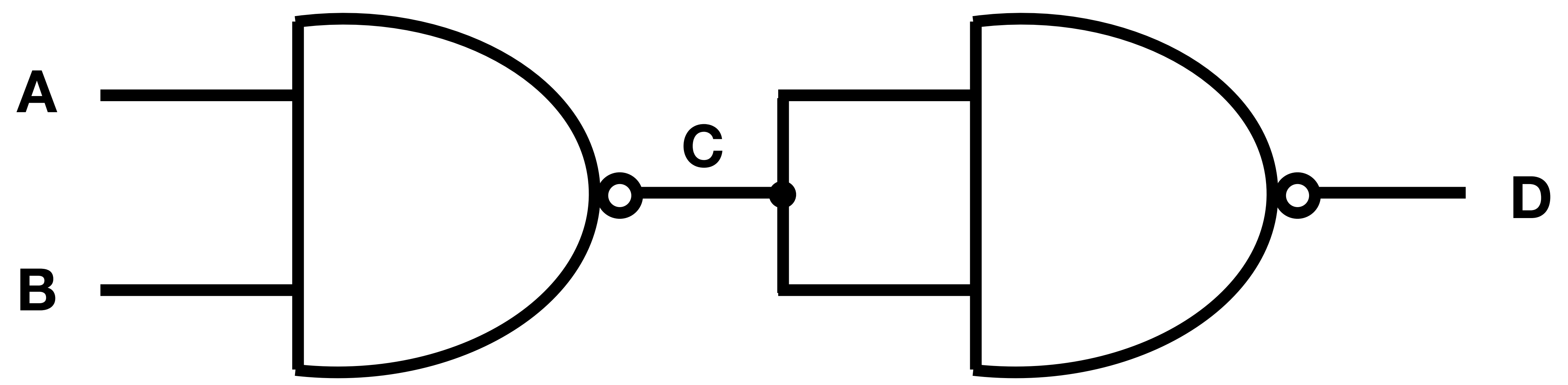


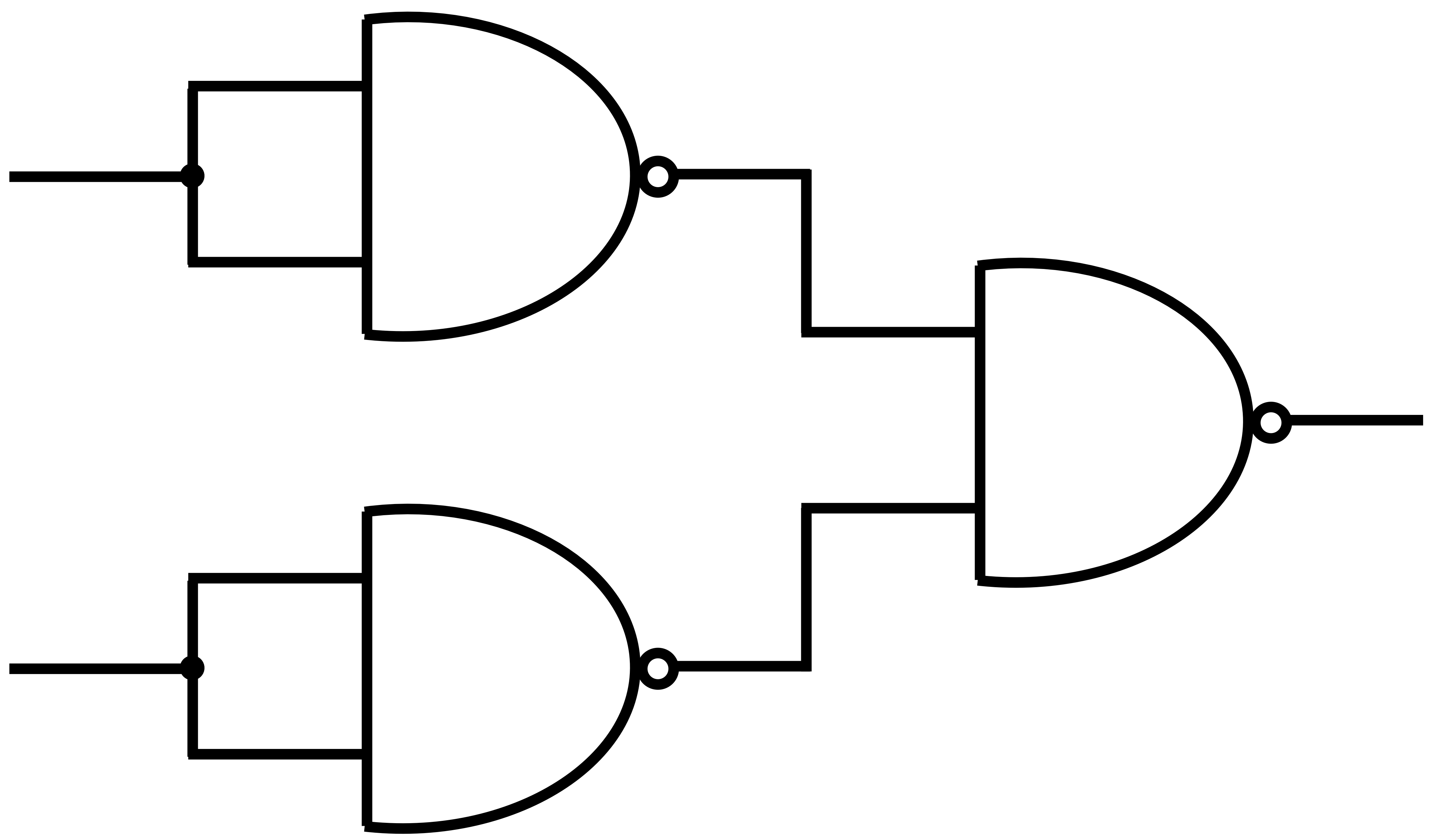
Using NANDs





Using NANDs





A	B
0	0
0	1
1	0
1	1

Boolean Algebra

Law	AND form	OR form
Identity 1	$A = \overline{\overline{A}}$	$A = \overline{\overline{A}}$
Identity 2	$1A = A$	$0 + A = A$
Null	$0A = 0$	$1 + A = 1$
Idempotence	$AA = A$	$A + A = A$
Complementarity	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutativity	$AB = BA$	$A + B = B + A$
Associativity	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributivity	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption	$A(A + B) = A$	$A + AB = A$
de Morgan's Law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$

Proof by Perfect Induction

(AND Forms)

$A = \overline{\overline{A}}$		
A	$\overline{A}$	$\overline{\overline{A}}$
0	1	0
1	0	1

$1A = A$		
1	A	$1A$
1	0	0
1	1	1

$0A = 0$		
0	A	$0A$
0	0	0
0	1	0

$AA = A$		
A	A	AA
0	0	0
1	1	1

$A\overline{A} = 0$		
A	$\overline{A}$	$A\overline{A}$
0	1	0
1	0	0

$A(A + B) = A$			
A	B	$A + B$	$A(A + B)$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

de Morgan's Law

- $\overline{(AB)} = \overline{A} + \overline{B}$ and $\overline{(A + B)} = \overline{A} \cdot \overline{B}$
- Useful for restating expressions
- If all sub-expressions are ANDed together, we can switch to ORing them together:
  1. negate overall expression
  2. negate all sub-expressions
  3. switch between ANDs and ORs

Designing Logic Circuits

- Basic approach:
  1. Write out a truth table for the desired logical function
  2. Derive a Boolean expression by ORing together all the rows whose output column is 1
  3. Translate to logic gates
     - Can simplify using digital circuit analysis techniques (see later lecture).

Logic Circuit Design

Example

- Consider a lighting system for a house.
- You have a light over the stairs which is controlled by two switches:
  - one at the bottom, in the hallway (H)
  - one at the top of the stairs (S)
- We want to be able to switch the light on at the bottom and off again at the top (and vice-versa).

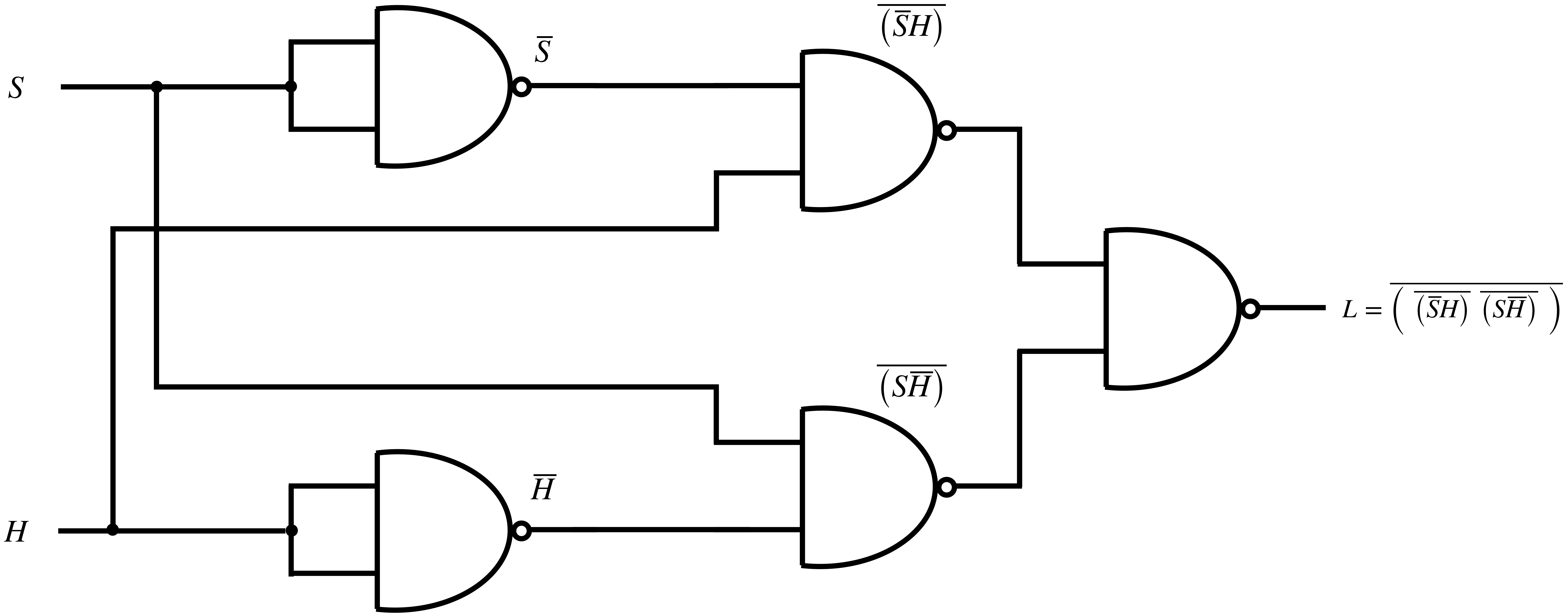
Stairway Light Logic

- We want the light to be **on** if S is **up** and H is **down**.
- Alternatively, the light should be **on** if S is **down** and H is **up**.

	S	H	Light
	0	0	0
$\overline{S}H$	0	1	1
$S\overline{H}$	1	0	1
	1	1	0

Stairway Light Logic

- So, $L = \overline{S}H + S\overline{H}$ (in sum-of-products form)
- We can implement this using NAND gates.
- Apply de Morgan's law:
  - Let $X = \overline{S}H$ and $Y = S\overline{H}$, so we have $X + Y$
  - By de Morgan's: $X + Y = \overline{(\overline{X}\overline{Y})}$
  - Expand: $L = \overline{\left(\overline{(\overline{S}H)}\ \overline{(S\overline{H})} \right)}$
  - This is now in the required 'Inverted AND' form...



S
H
$\overline{S}$
$\overline{H}$
$\overline{(\overline{S}H)}$
$\overline{(S\overline{H})}$
$L = \overline{\left(\overline{(\overline{S}H)}\ \overline{(S\overline{H})} \right)}$