Logic Gates and Adders

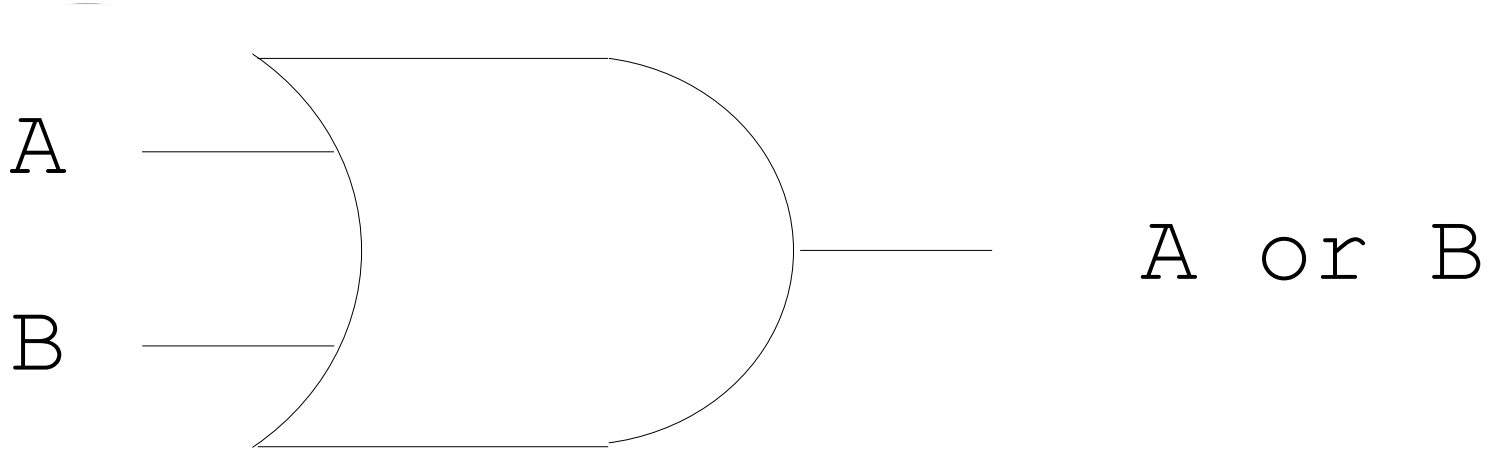
- Logic Gates
- Sum of Products
- Half Adder
- Full Adder
- Ripple-Carry Adder

Logic Gates

- How are boolean values represented in hardware?
  - Electrical signals on wires
  - Low voltage = 0 = false
  - High voltage = 1 = true
- How is logic implemented in hardware?
  - Logic gates
  - Input wires, output wires
- **All logic is running, all the time!**

Logic Gates

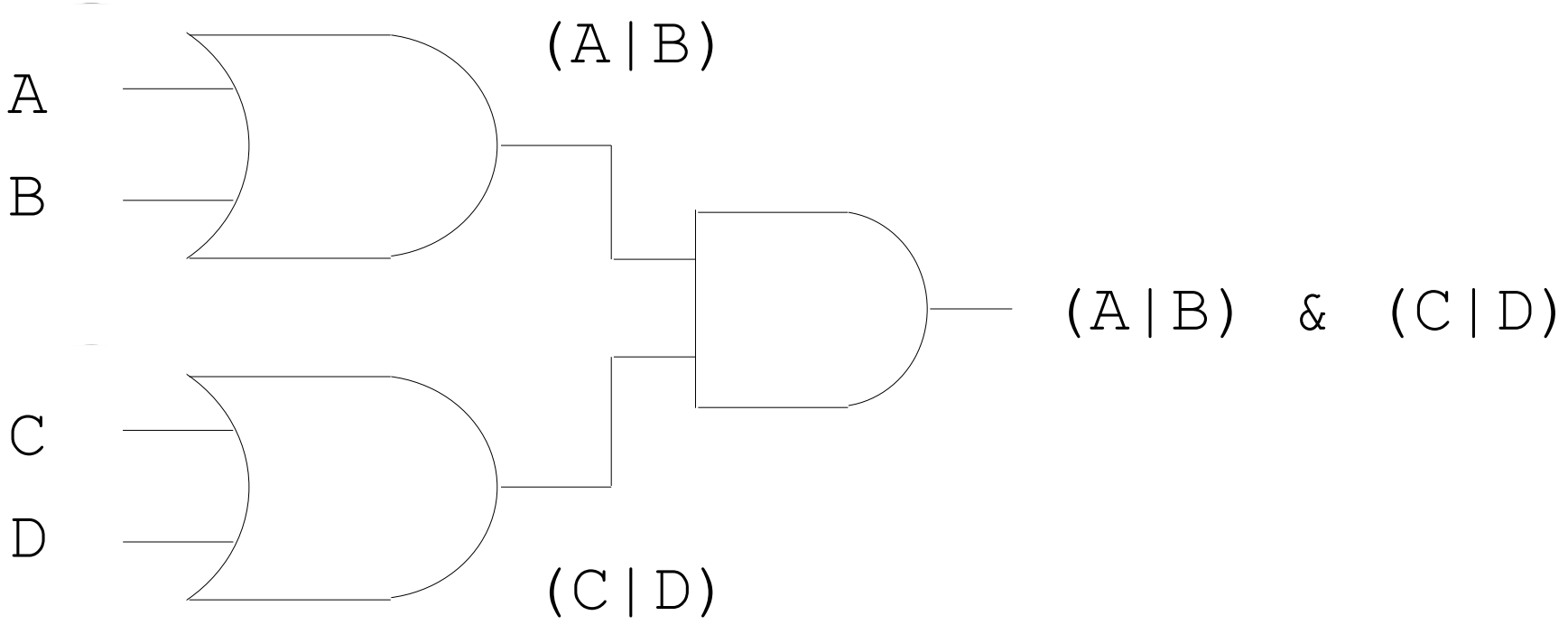
- This is an OR gate. Two inputs, one output.



- The value of the output is **always** equal to the OR of the two inputs
  - Never turns off!
  - Detail: propagation delay

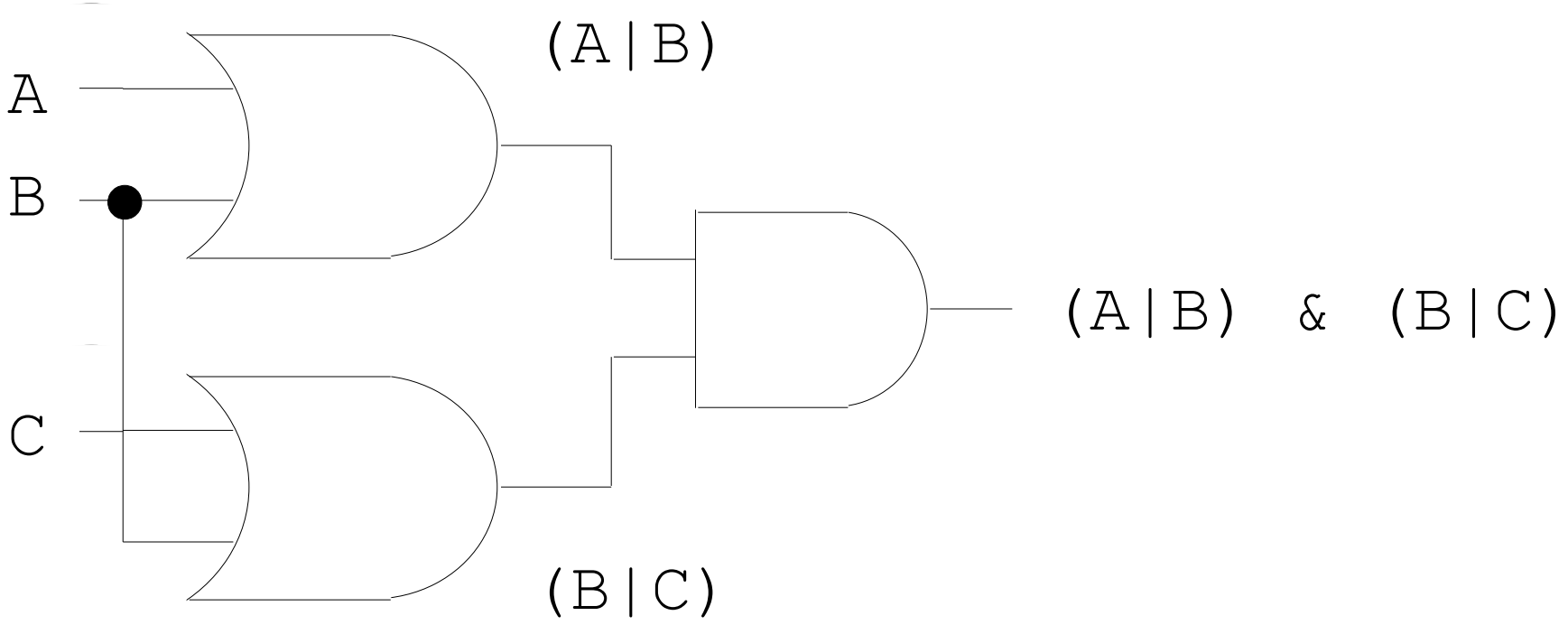
Logic Gates

- We can build more complex expressions by connecting inputs to outputs:



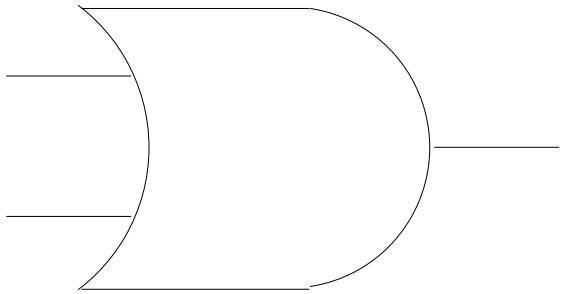
Logic Gates

- Solid dots represent connected wires (sometimes omitted)

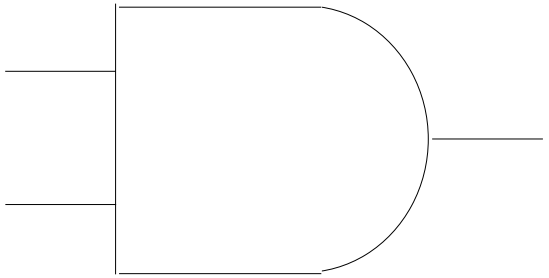


The Basic Elements

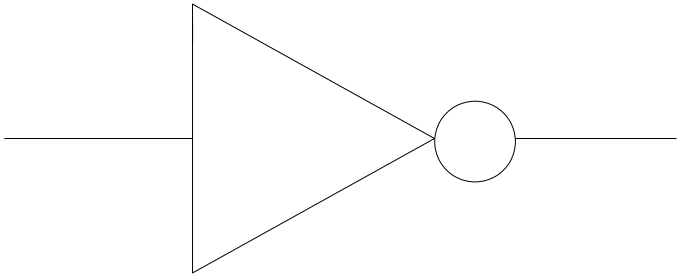
OR



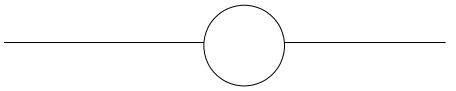
AND



NOT



NOT
(alternate)

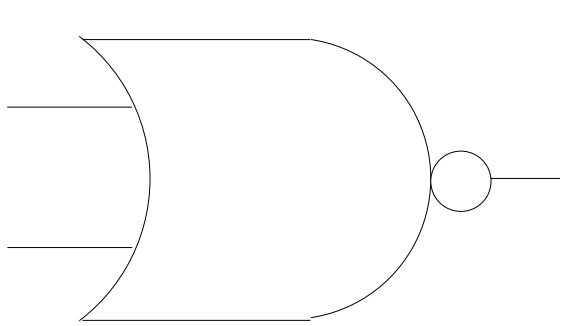


It's OK to label the gates with text if you need.

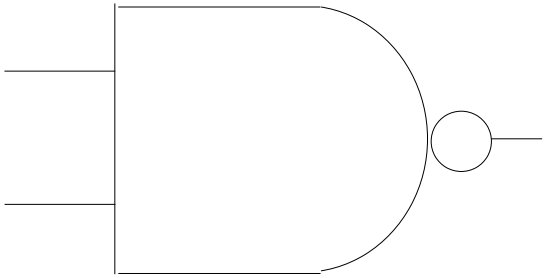
Shorthand for NOT

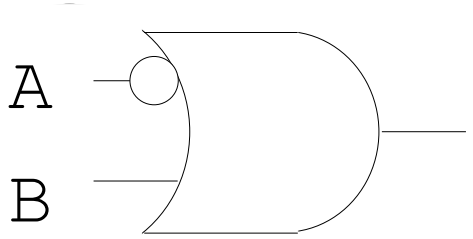
- The dot (NOT) can be applied on inputs or outputs

NOR



NAND

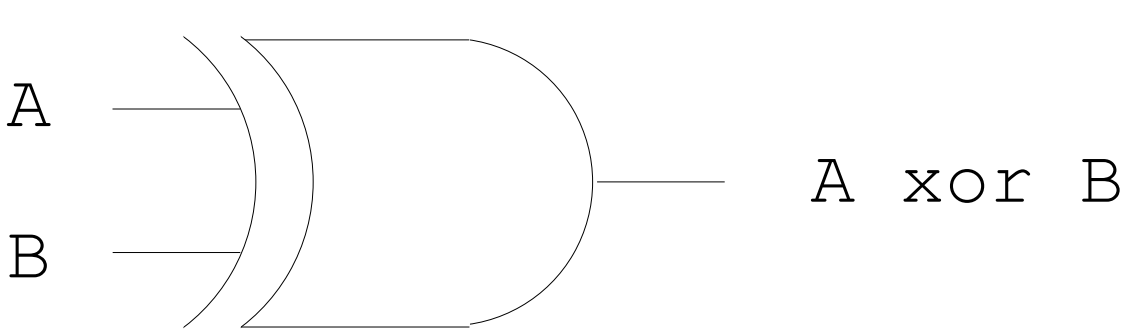




`(~A|B)`

XOR

- XOR is occasionally useful

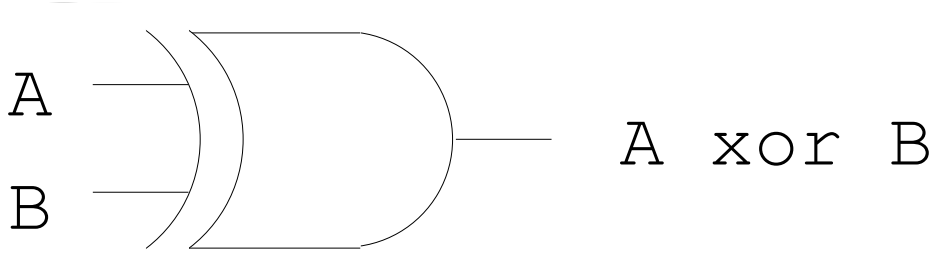


Group Exercise:

Devise a network of AND, OR, NOT gates which implements XOR.

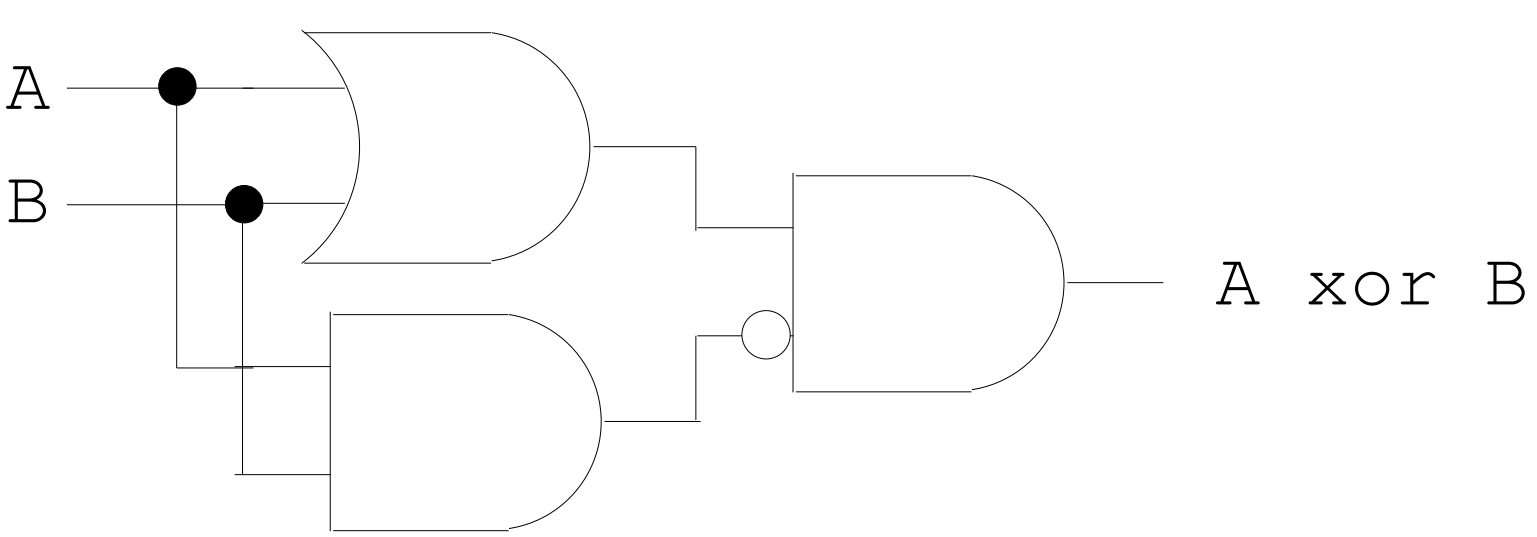
Two inputs: A, B

XOR

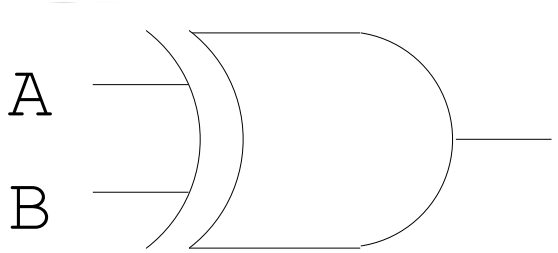


Solution 1 of 2:

```
A xor B = (A|B) & ~(A&B)
```



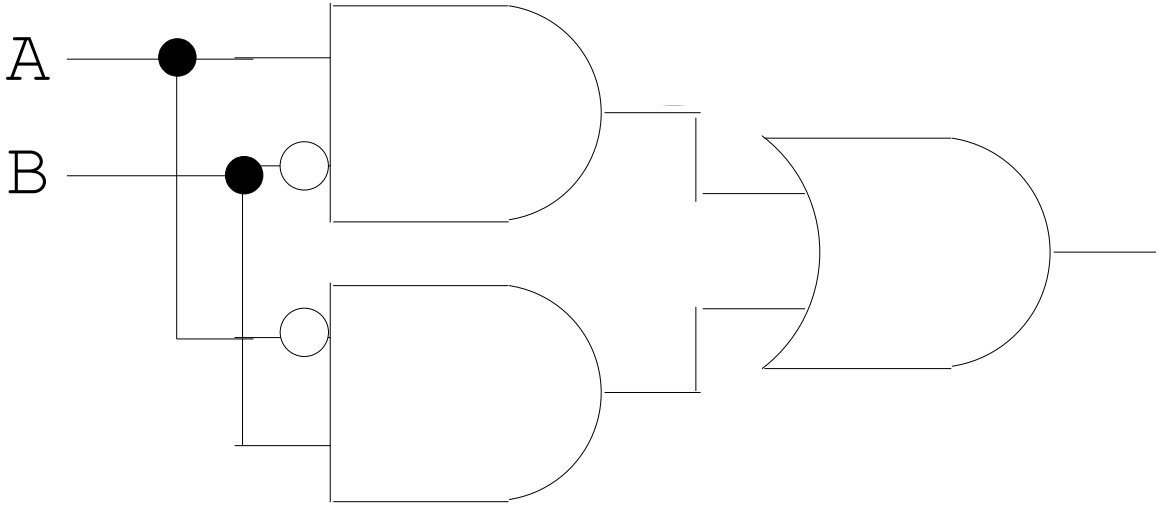
XOR



A xor B

Solution 2 of 2:

```
A xor B = (A & ~B) | (~A & B)
```



A xor B

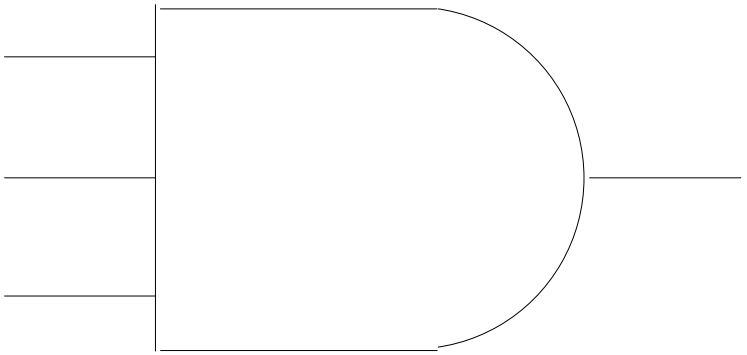
Principles of Logic Design

- Composition
  - Implement large components from smaller pieces
  - Like programming!

- Logic = Boolean Algebra
  - One-to-one map between gates, operators

More Inputs

- How to implement gates with multiple inputs?

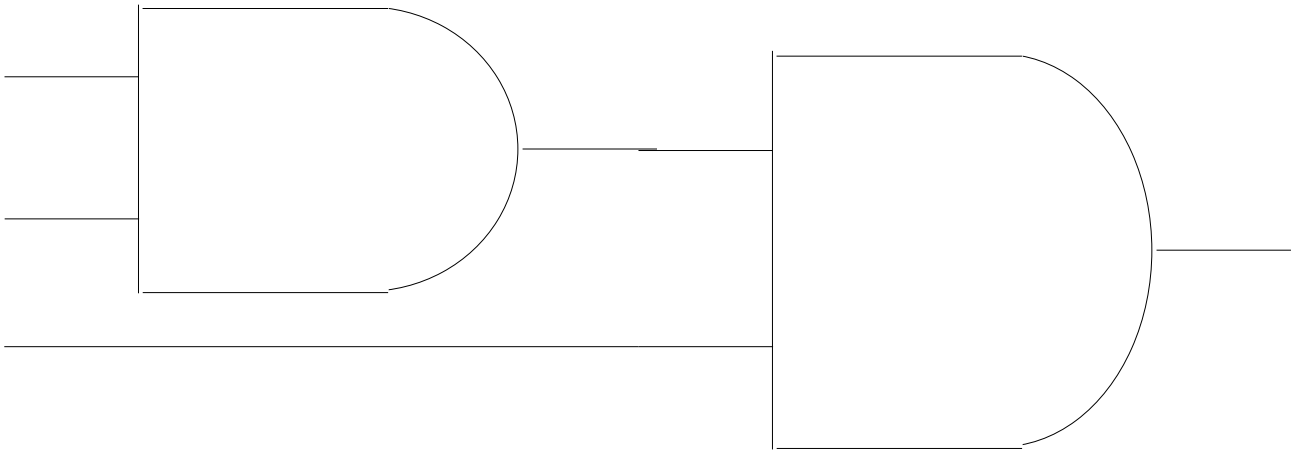


Group Exercise:

Devise a way to build a 3-input AND gate from 2-input gates.

Then devise **2 different** ways to build an 8-input AND !

More Inputs

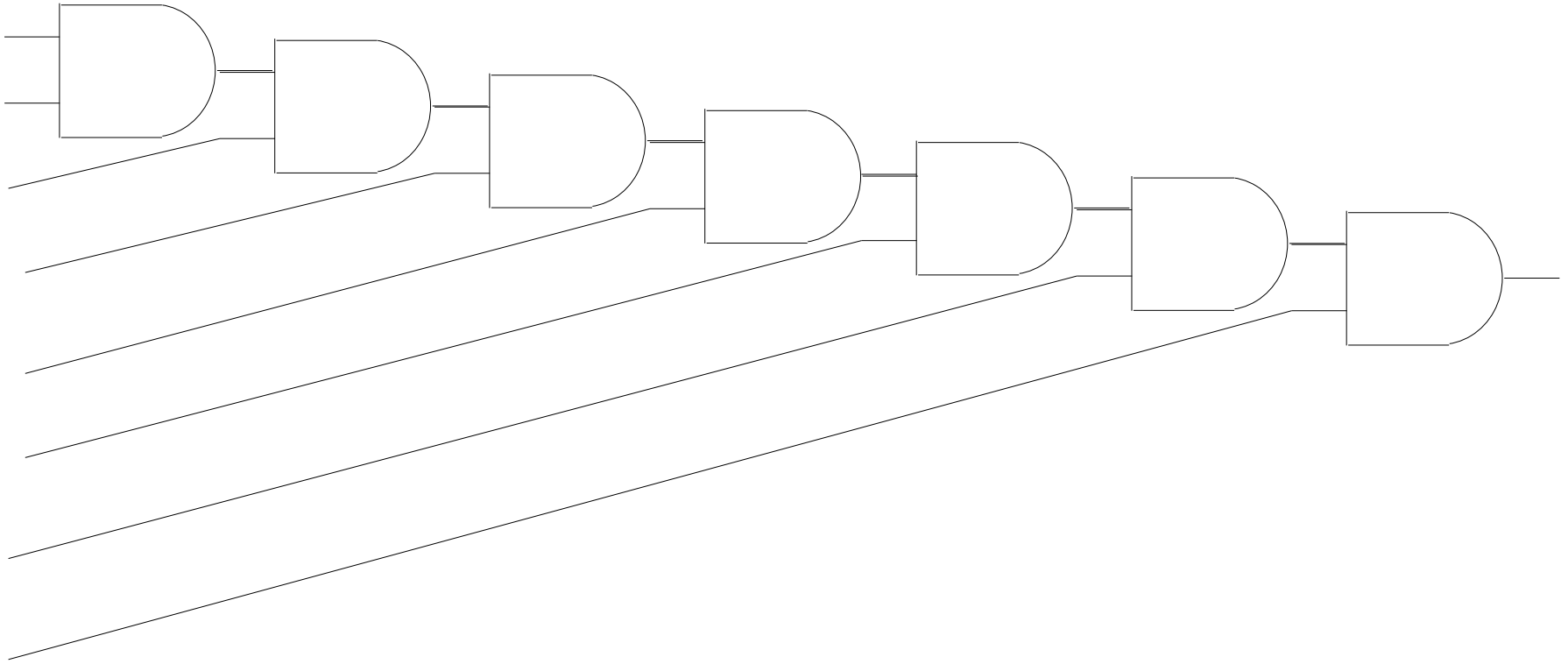


Questions:

How many 2-input gates were required?
(This affects the **cost** of the chip)

What is the longest path from an input to the output?
(This affects the **speed** of the chip)

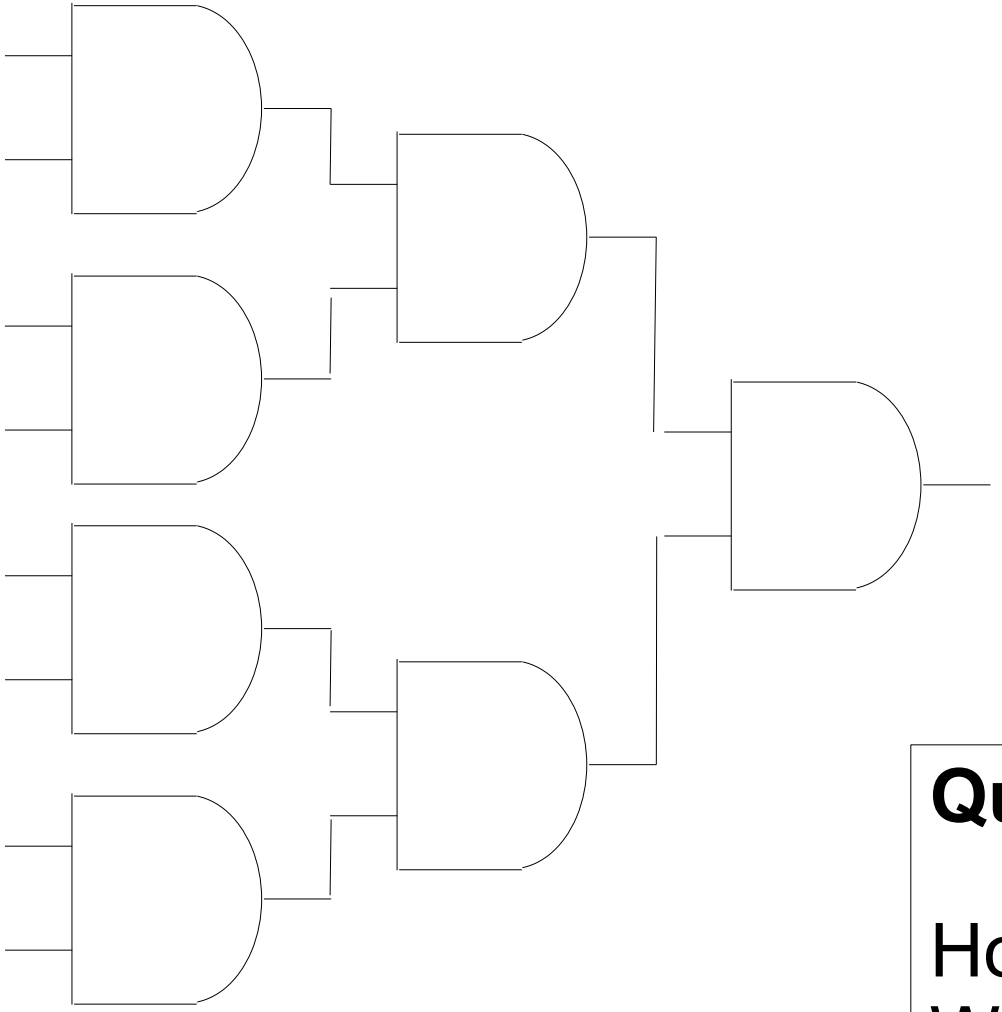
More Inputs



Questions:

How many gates?
What is the path length?

More Inputs



Questions:

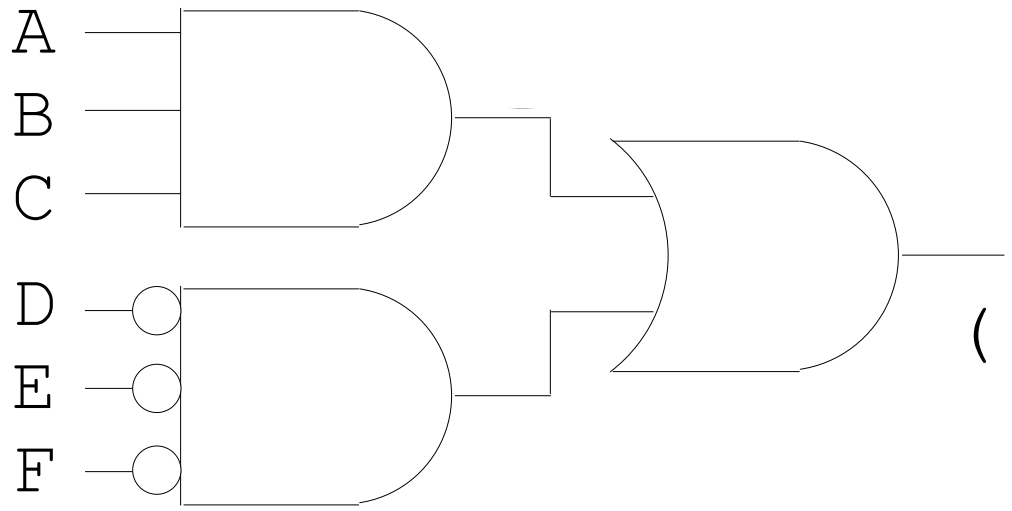
How many gates?
What is the path length?

Sum of Products

- **Sum of Products** is shorthand for a specific type of logical expression, representing a special gate arrangement

$$ABC + \bar{D}\,\bar{E}\,\bar{F}$$

Question:
What would change if you linked the over-bar in DEF into one big bar?




```
(A and B and C)
      or
(!D and !E and !F)
```

Truth Tables

- Any logical expression (no matter how complex) can be modeled as a truth table.

Group Exercise:

Write a truth table with three inputs (A,B,C), and three outputs:

X is true if **all** of the inputs are true
Y is true if **any** of the inputs are true
Z is true if **exactly two** of the inputs are true

X is true if **all** of the inputs are true
Y is true if **any** of the inputs are true
Z is true if **exactly two** of the inputs are true

A	*B*	*C*	*X*	*Y*	*Z*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	0

Sum of Products

- Any truth table can be modeled as a “sum of products”
  - Each product represents one **true** row.

A	*B*	*A xor B*
0	0	0
0	1	1
1	0	1
1	1	0

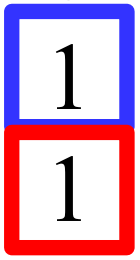
Questions:

Which rows are true?
How to encode each as a product?

Sum of Products

- Any truth table can be modeled as a “sum of products”
  - Each product represents one **true** row.

A	*B*	*A xor B*
0	0	0
0	1	1
1	0	1
1	1	0



$$A \text{ xor } B$$

$$(\overline{A}B) + (A\overline{B})$$

Sum of Products

Group Exercise:

Write a truth table with three inputs (A,B,C), and three outputs:

X is true if all of the inputs are true
Y is true if any of the inputs are true
Z is true if exactly two of the inputs are true

Convert X,Y,Z to sum-of-products.

Convert each sum-of-products to a gate network.

A	B	C	X	Y	Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	0

X = ABC

A	B	C	X	Y	Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	0

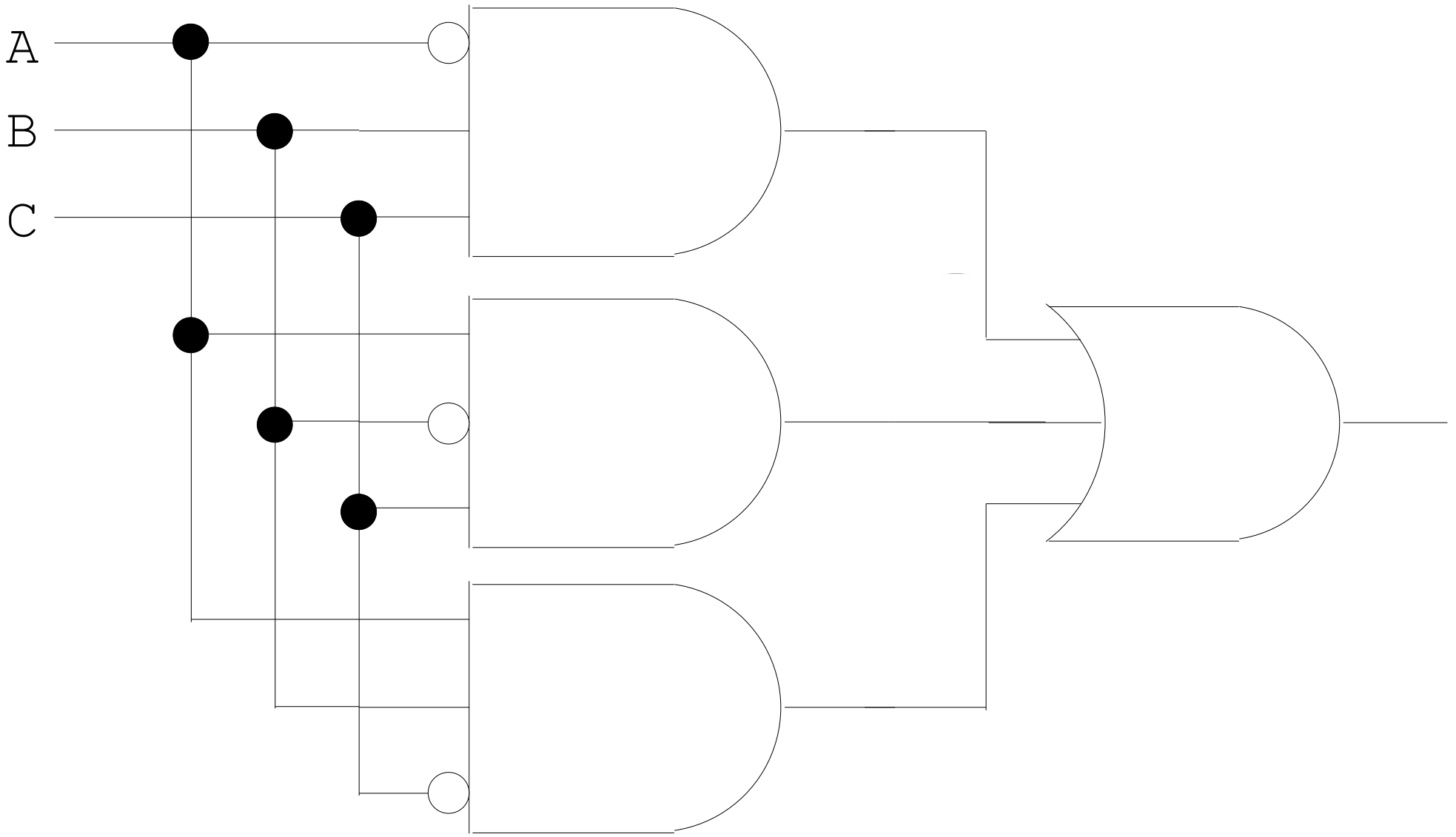
Optimization is possible.

But it's outside the scope of this class!

$$Y = (\overline{\overline{A}BC}) + (\overline{\overline{A}B}\,\overline{C}) + (\overline{\overline{A}}BC) + (A\overline{\overline{BC}}) + (A\overline{\overline{B}}C) + (AB\overline{C}) + (ABC)$$

A	B	C	X	Y	Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	0

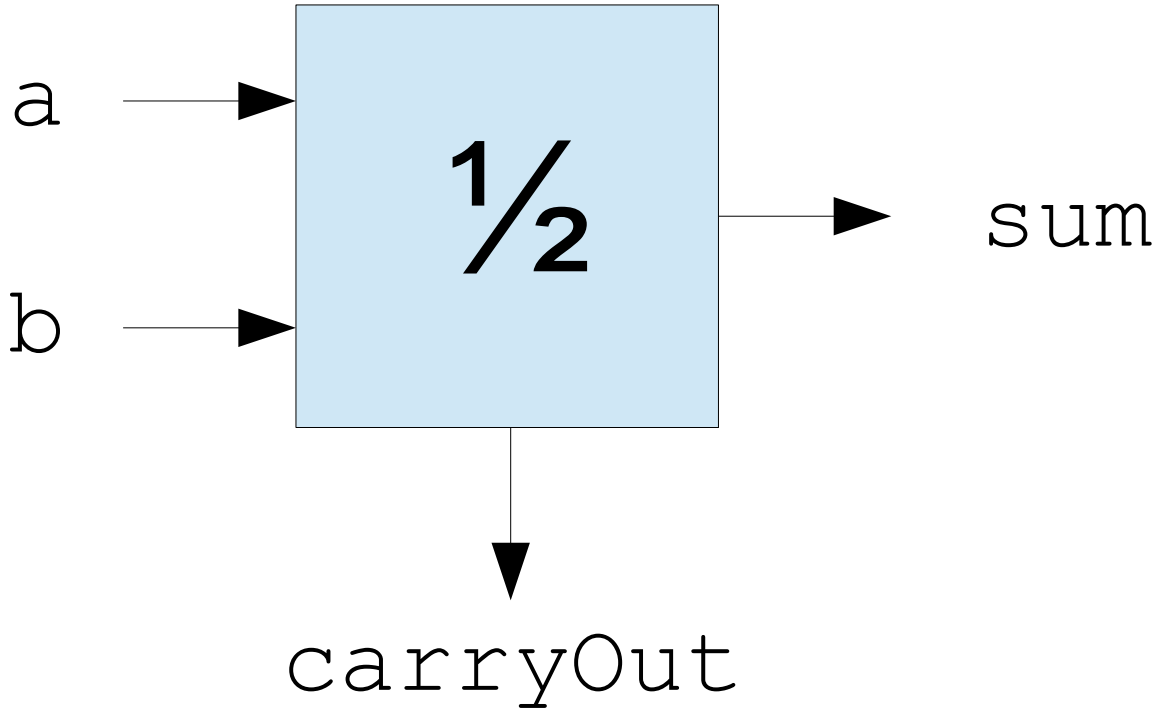
$$Z = (\overline{A}BC) + (A\overline{B}C) + (AB\overline{C})$$



$$Z = (\overline{A}BC) + (A\overline{B}C) + (AB\overline{C})$$

Half Adder

- A **half adder** is a logic component which does addition – but which has no `carryIn`



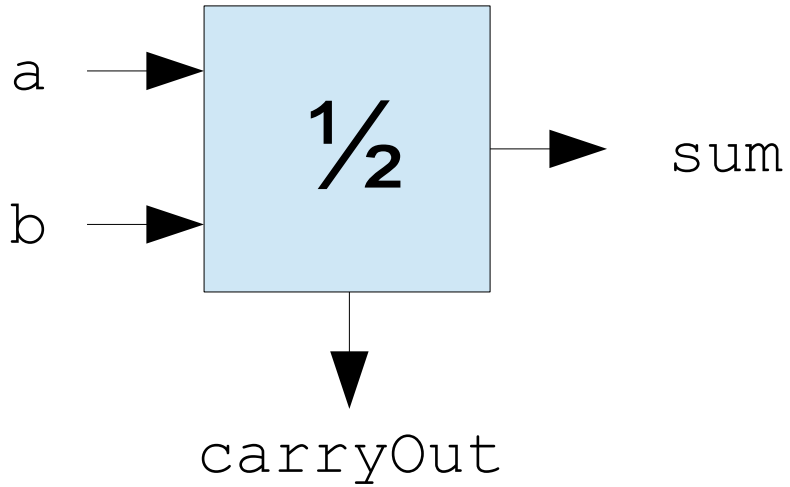
Half Adder

Group Exercise:

Write the truth table for a half adder.

Convert to sum of products, then to a logic network.

<u>Optional:</u>
Optimize the 'sum' output to be a single gate. It will be something more complex than AND/OR/NOT.



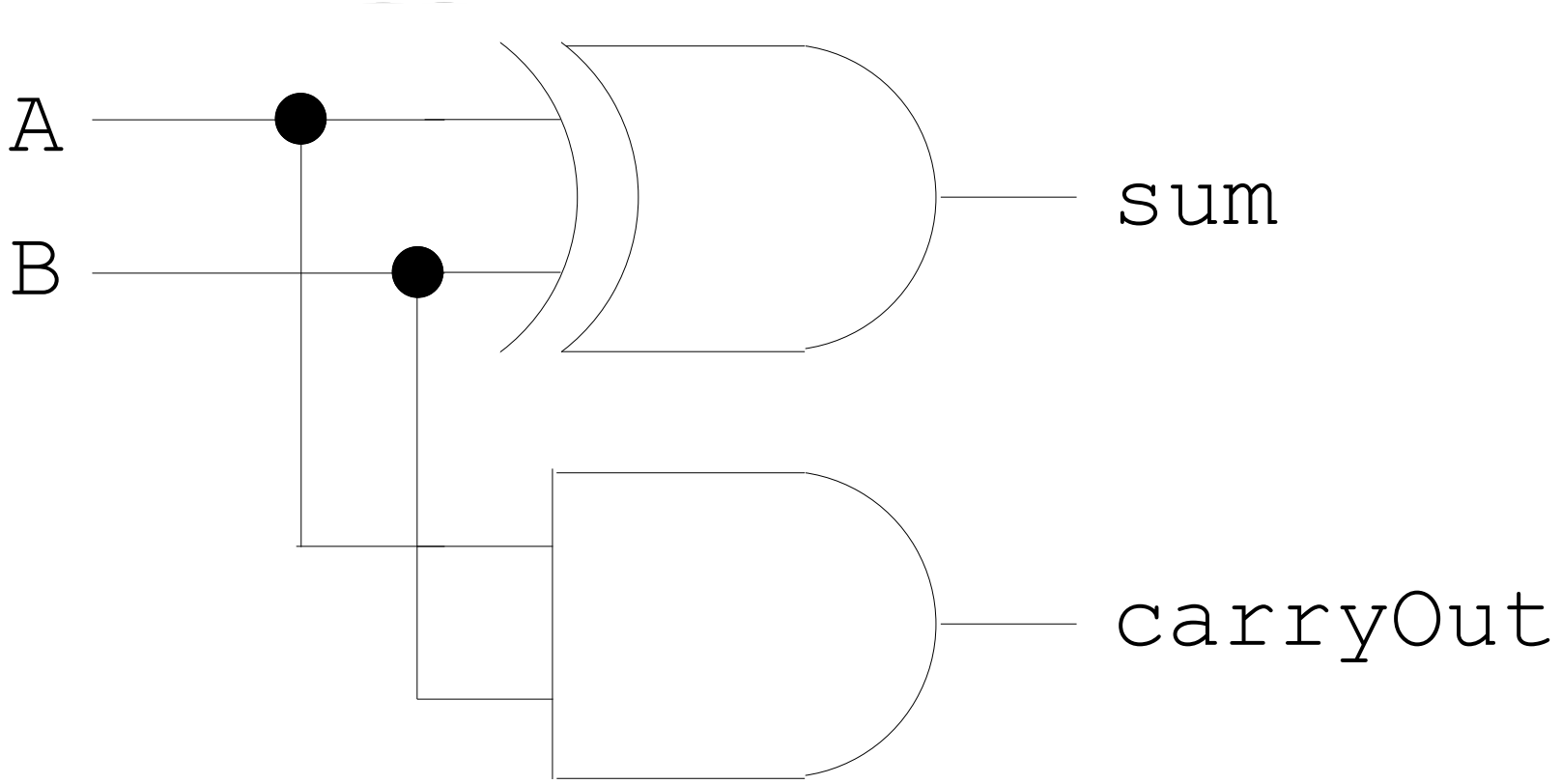
Half Adder

A	B	sum	carryOut
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$\texttt{sum} = (\overline{A}B)+(A\overline{B}) = \texttt{A xor B}$

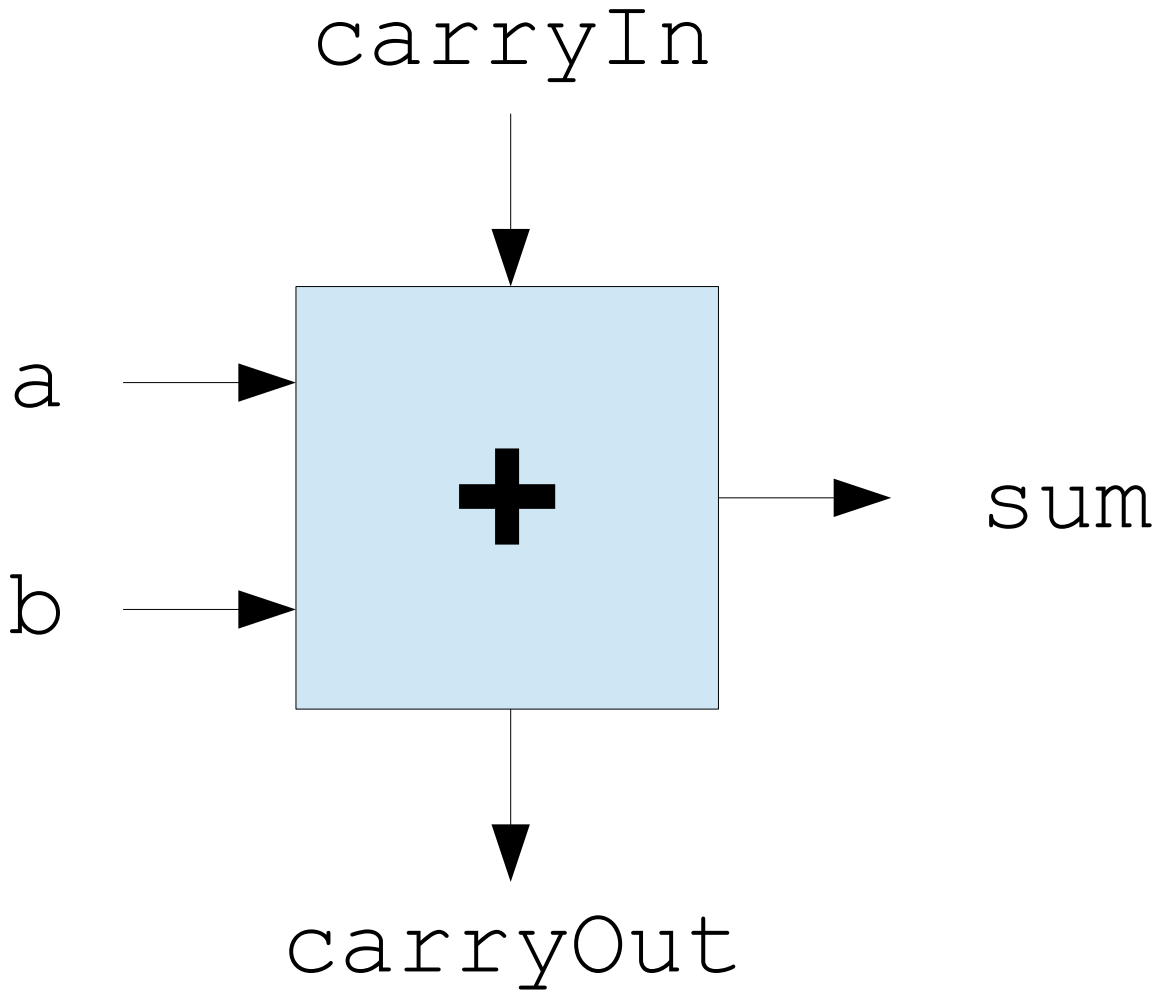
$\texttt{CarryOut} = AB$

Half Adder



Full Adder

- This is the symbol for a full adder



Full Adder

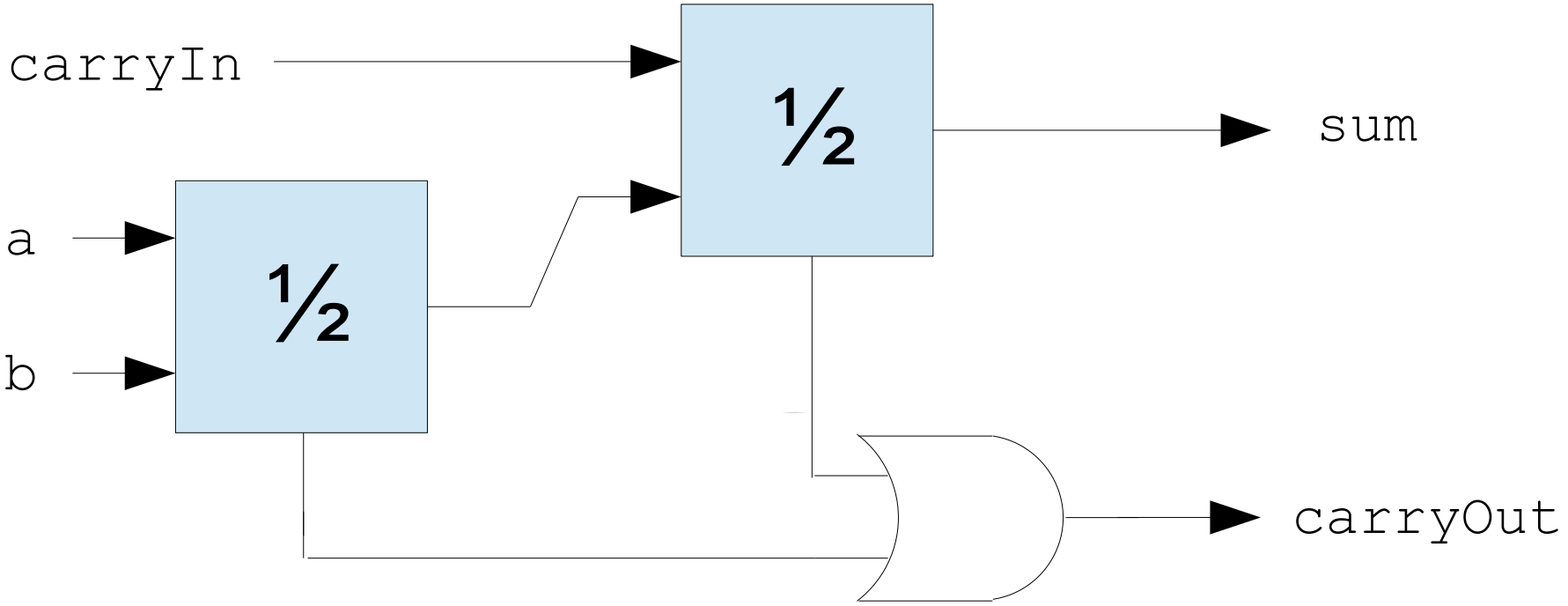
- To build a **full adder,** add twice:

  `(a+b)+carryin`

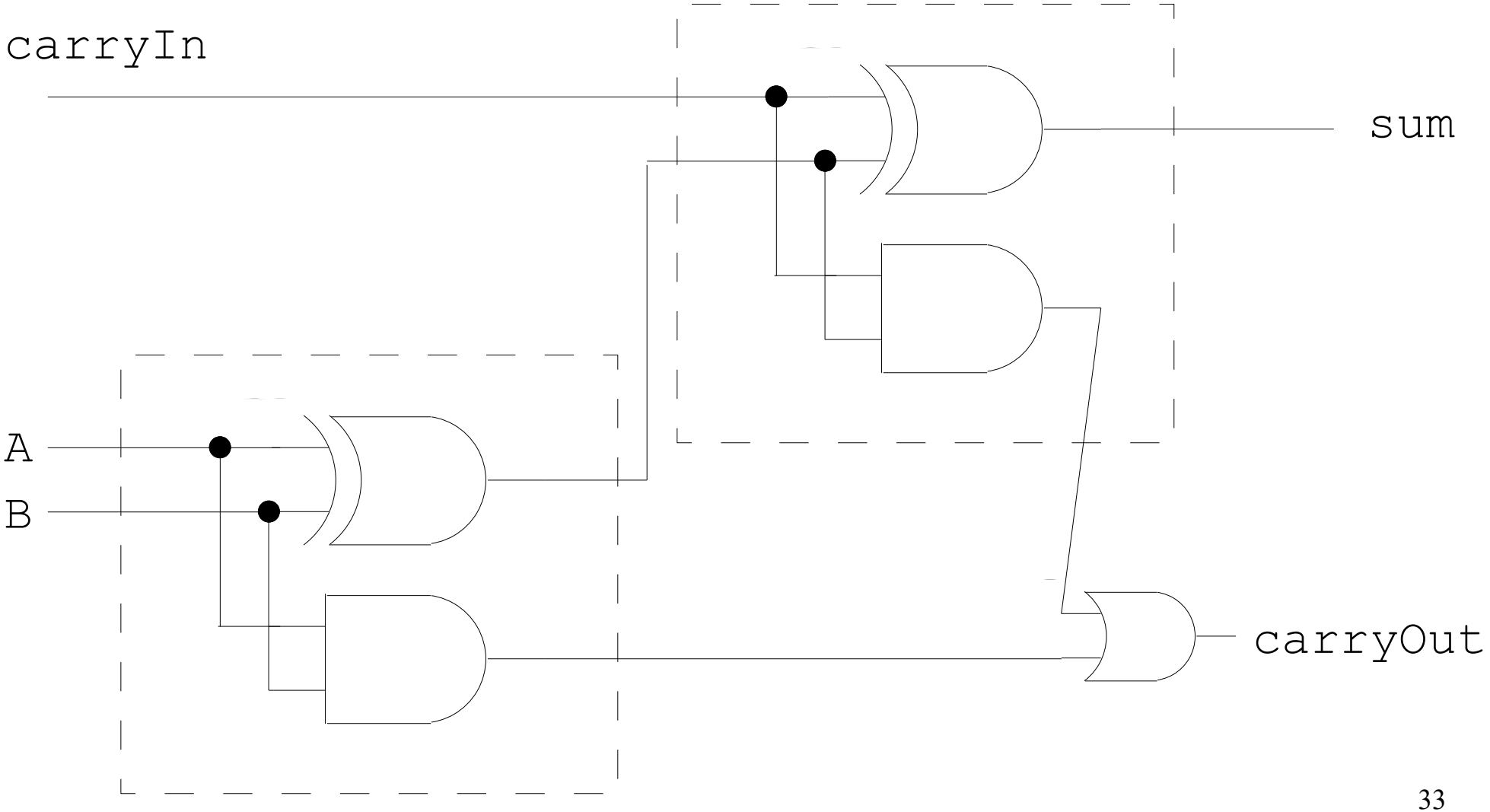
  - (This is why we call the simpler piece a half adder.)

- We have a carry if there was a carry in **either** of the two add operations!

Full Adder



Full Adder, a 2nd View

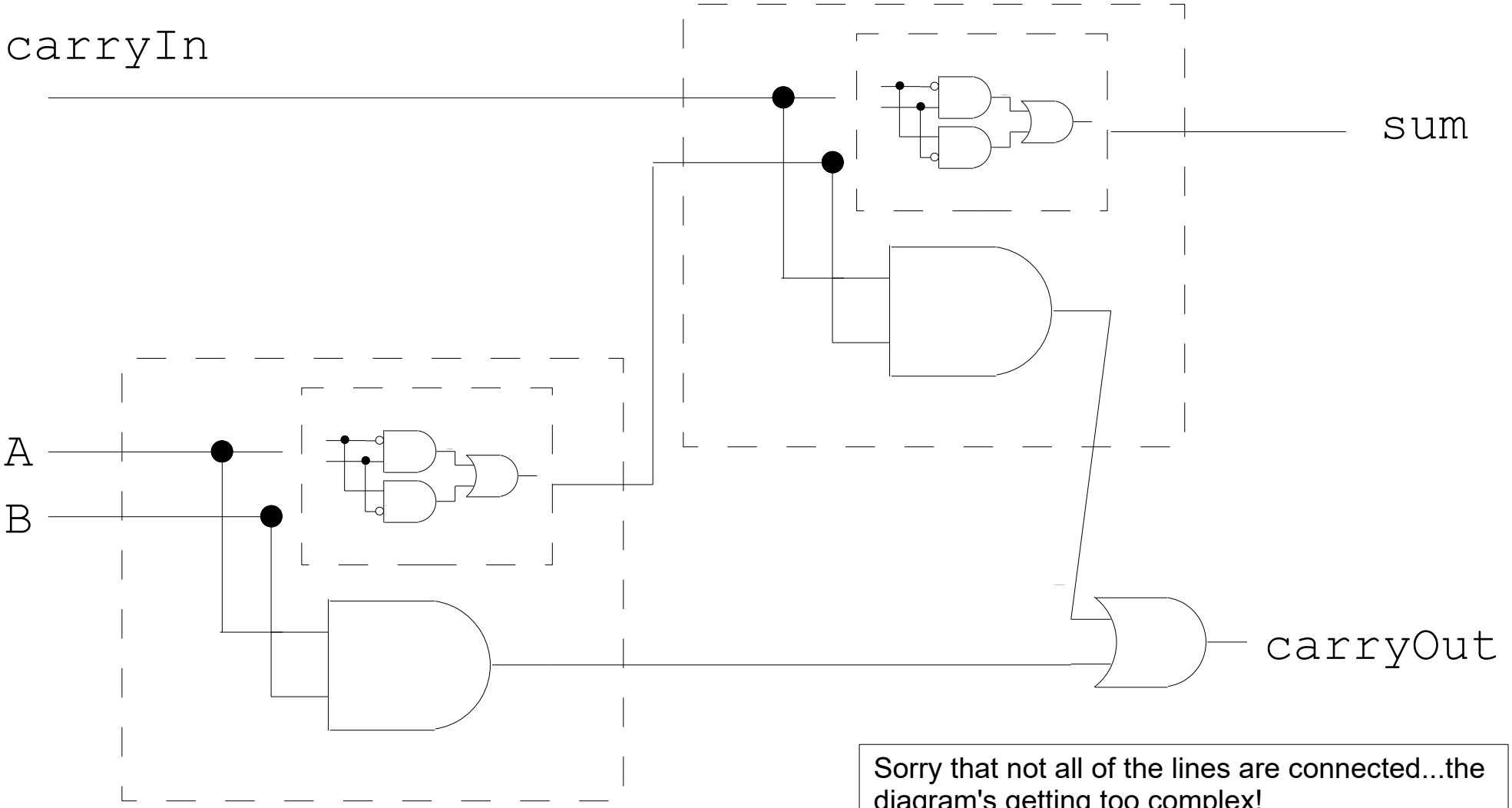


Full Adder

- We just gave 2 different versions of the Full Adder
  - They are equivalent
  - Two different levels of abstraction
    - 2 half adders
    - Lots of gates

- Wait … XOR isn't a low-level gate, either!

Full Adder, a 3rd View



Sorry that not all of the lines are connected...the diagram's getting too complex!

(But that's the point. **Abstraction is good.**)

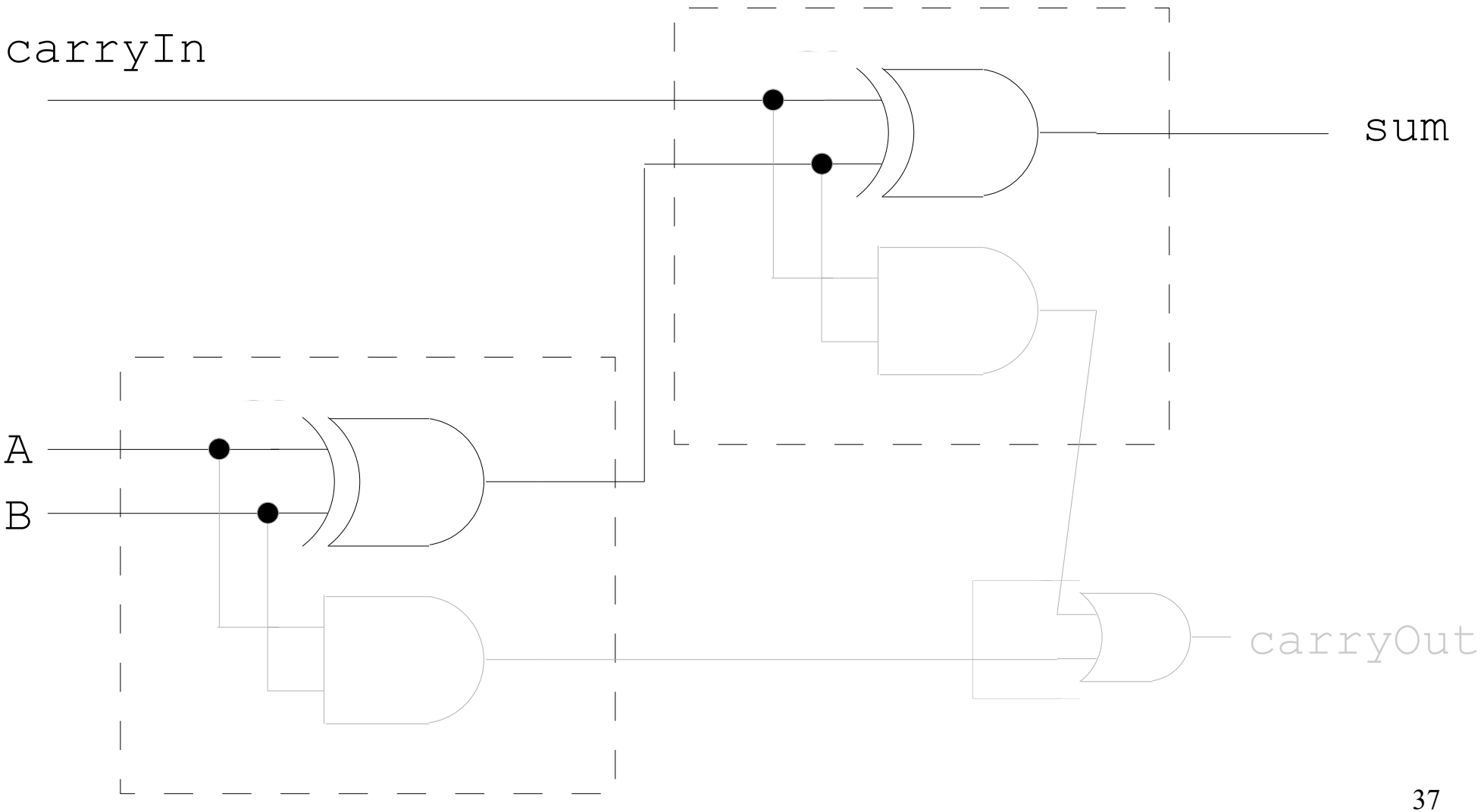
Two Gate Networks, in parallel

Group Exercise:

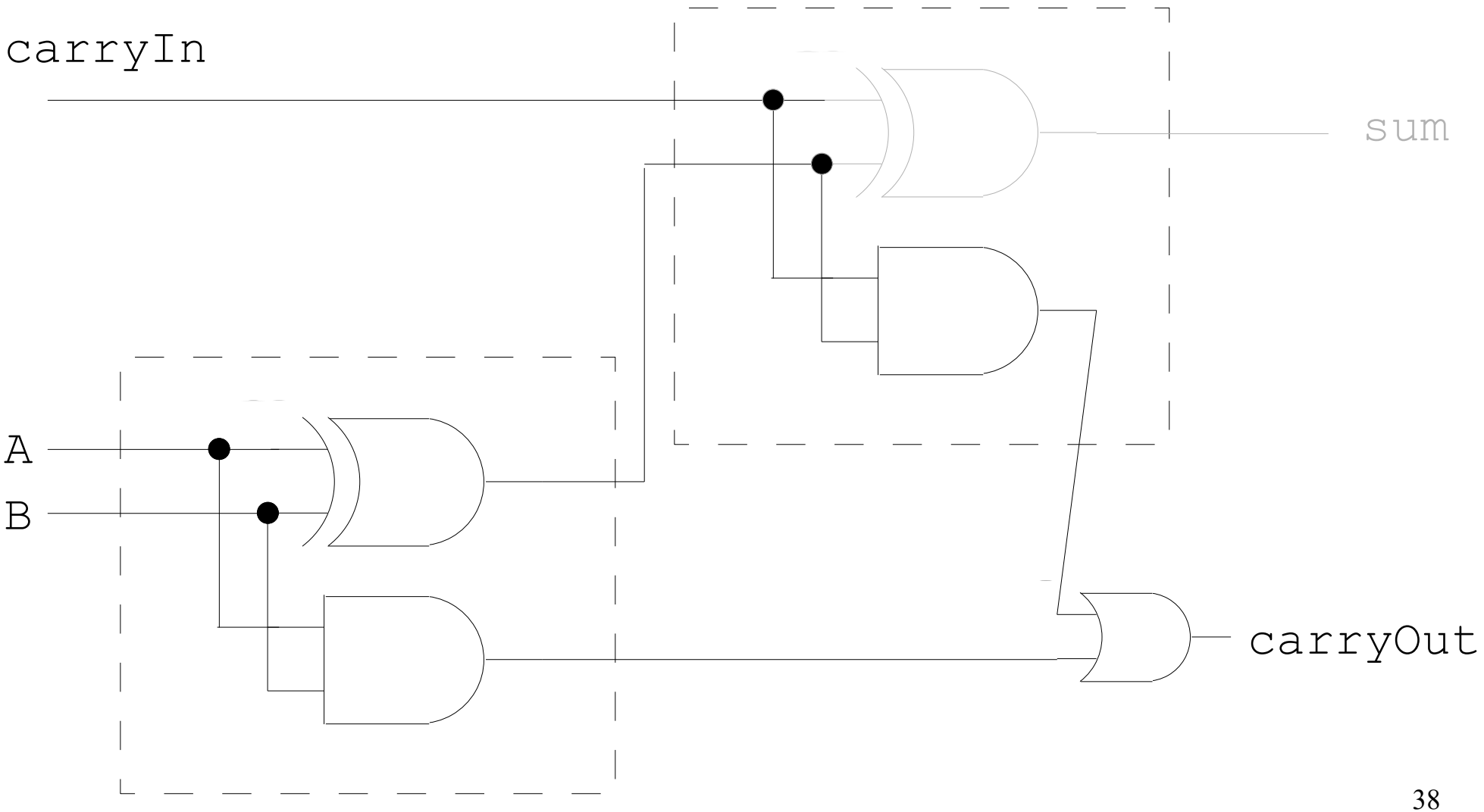
Redraw the circuit as **two gate networks.** In each network, only include the gates necessary for that output.

(I'll go back to the XOR version of the Full Adder so that you can see what it was.)

Full Adder: sum



Full Adder: `carryOut`



Group Exercise:

Work backwards from a gate network to a truth table!

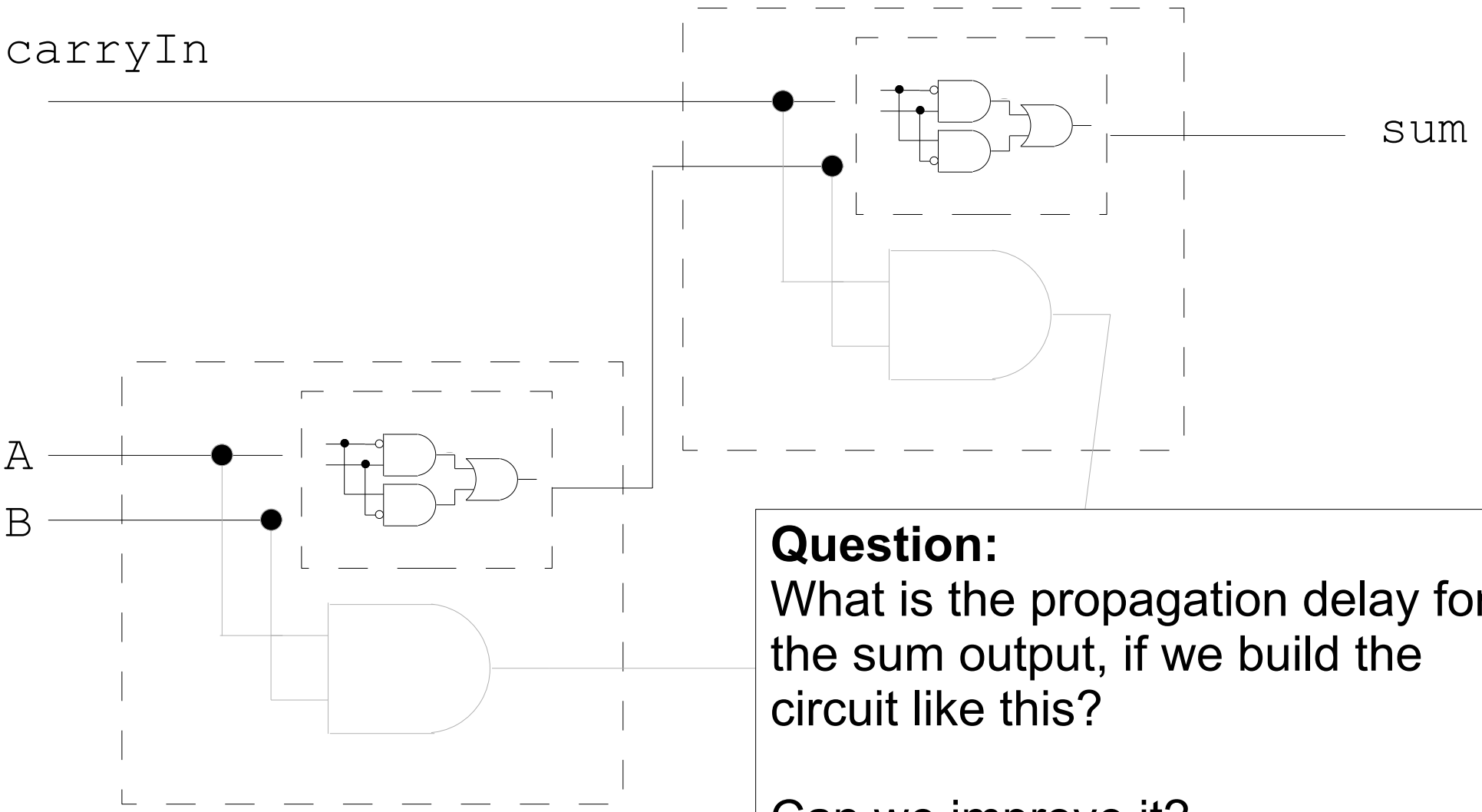
Using the diagrams from the previous slides (I'll go back in a moment), write a logical expression for the outputs `sum`, `carryOut`.

HINT: They will **not** be a sum-of-products.

Then build a truth table, and see what those expressions resolve to in each case.

Finally, **check** to see if the outputs are correct, based on what you know about how addition should work.

Full Adder, a 3rd View



Question:
What is the propagation delay for the sum output, if we build the circuit like this?

Can we improve it?

Group Exercise:

Using your truth table for `sum`,`carryOut`, build sum-of-products expressions for both outputs.

Insight:
Every circuit has multiple implementations. Abstraction is good for designers; but often, the real circuit will need to be optimized.

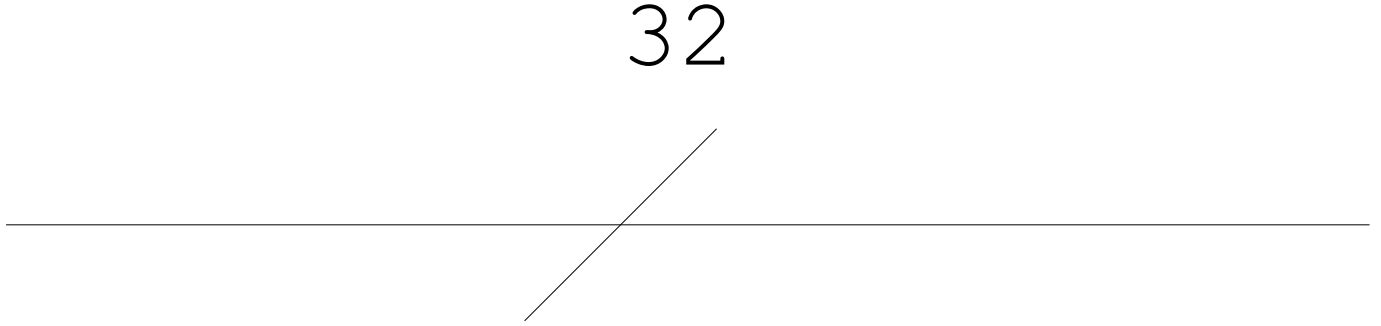
In this class, we see that optimization is possible, but don't study it in depth.

A	*B*	*carryIn*	sum	carryOut
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\begin{aligned} \texttt{sum} &= (\overline{\texttt{A}}\overline{\texttt{B}}\texttt{C}) + (\overline{\texttt{A}}\texttt{B}\overline{\texttt{C}}) + (\texttt{A}\overline{\texttt{B}}\overline{\texttt{C}}) + (\texttt{ABC}) \\ \texttt{cOut} &= (\overline{\texttt{A}}\texttt{BC}) + (\texttt{A}\overline{\texttt{B}}\texttt{C}) + (\texttt{AB}\overline{\texttt{C}}) + (\texttt{ABC}) \\ &= (\texttt{BC}) + (\texttt{AC}) + (\texttt{AB}) \end{aligned}$$

Buses

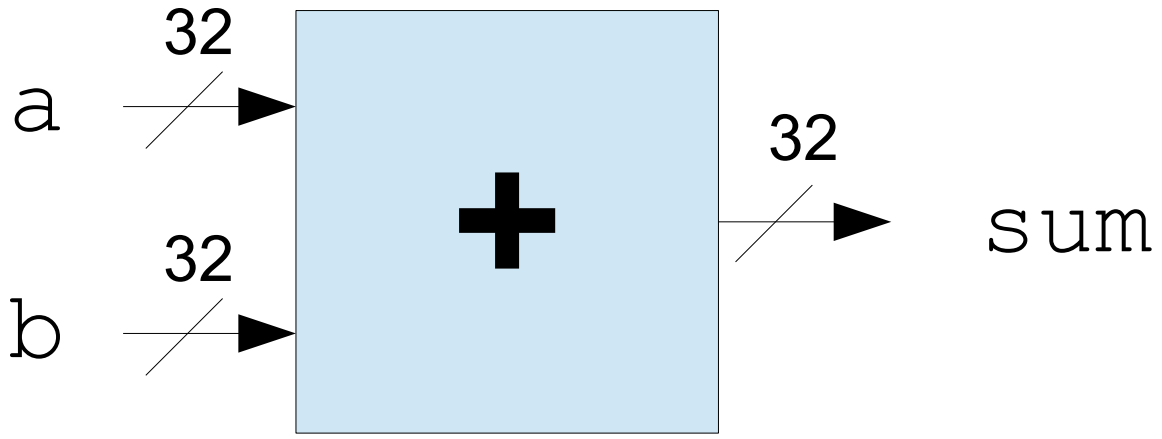
- We use the following symbol to represent a **multi-bit bus:**



- A **bus** is a bunch of wires going together to the same destination.

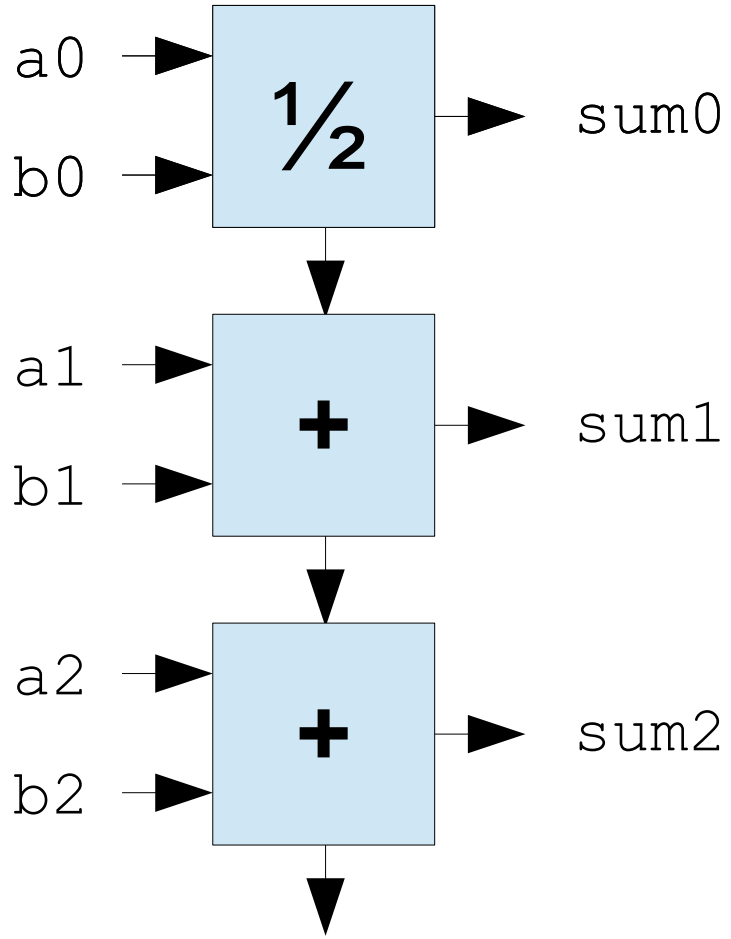
Adder

- We want to build an adder that adds **multiple bits:**



Multi-Bit Adder

- We can chain the adders together to build a multi-bit adder.
- The LSB only needs a half adder (for now).



Multi-Bit Adder

- This is called a **ripple carry adder.**
  - Each bit cannot calculate its `sum` or `carryOut` until `carryIn` is ready.
- Ripple carry adders are easy to understand but **very slow.**
  - (Better version coming later)