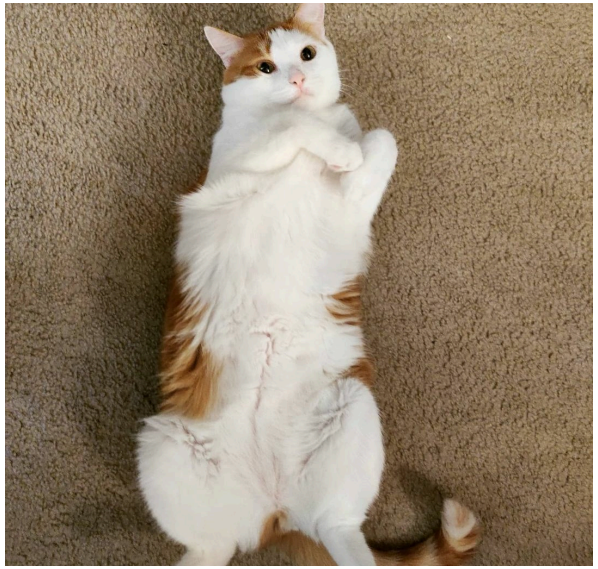
Circuits

CMSC250

Announcements / Reminders / Schedule

- Recording of Sunday's review session is up.
- Homework 4 solutions are also up.
  - Consult them if you have questions about combinatorial proof structure, or other questions!
- Remember:
  - No hw this week
  - No quiz this week
  - But next week there **will** be both! ☺😐☹
- Today & Thursday: circuits!



Announcements / Reminders / Schedule

- Expectation of grading:
  - Homework 4 grading comes first, by Monday.
  - Midterm grading will take extra time (Friday 03-12).
  - You will have a full week (through Friday 03-19, Springbreak Friday) to submit regrade requests on Gradescope.
- Midterm solutions posted as soon as Thursday 08:01 AM
  - Probably a bit later, to avoid last – minute panic of any kind.
- Logic behind solving will be covered in one of the two discussion sessions next week.

Circuits

- We can build circuits for addition, multiplication, division, bit shifting…
- Every logical operation we have learned $(\sim, \wedge, \vee)$ maps straightforwardly to a tiny piece of hardware called a *logical gate*.
- These gates connect to each other to make arbitrarily complicated circuits!

From a truth table to a formula

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

From a truth table to a formula

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- Let us focus entirely on the rows that output 1!

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 1st row…

p	q	r	output
0	0	0	1

- Write a simple formula that is ’1’ only on inputs p =0, q = 0, r = 0.

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 1st row…

p	q	r	output
0	0	0	1

- Write a simple formula that is ’1’ only on inputs p =0, q = 0, r = 0.

$$\sim p \wedge \sim q \wedge \sim r$$

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 4th row…

p	q	r	output
0	1	1	1

- Same deal

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 4th row…

p	q	r	output
0	1	1	1

- Same deal

$$\sim p \wedge q \wedge r$$

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 5th row…

p	**q**	**r**	**output**
1	0	0	1

$$p \wedge \sim q \wedge \sim r$$

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 8th row…

p	q	r	output
1	1	1	1

$$p \wedge q \wedge r$$

How do we combine those simple formulae?

$\sim p \wedge \sim q \wedge \sim r$

$\sim p \wedge q \wedge r$

$p \wedge \sim q \wedge \sim r$

$p \wedge q \wedge r$

How do we combine those simple formulae?

$(\sim p \wedge \sim q \wedge \sim r) \vee$

$(\sim p \wedge q \wedge r) \vee$

$(p \wedge \sim q \wedge \sim r) \vee$

$(p \wedge q \wedge r)$

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- Outputs 1 if and only if the truth table outputs 1!

How do we combine those simple formulae?

$(\sim p \wedge \sim q \wedge \sim r) \vee$

$(\sim p \wedge q \wedge r) \vee$

$(p \wedge \sim q \wedge \sim r) \vee$

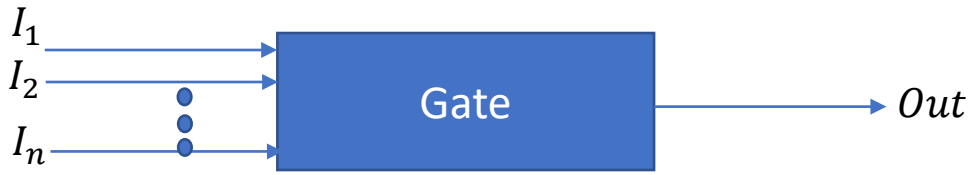
$(p \wedge q \wedge r)$

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- Outputs 1 if and only if the truth table outputs 1!
- **We want to do this in *hardware!***

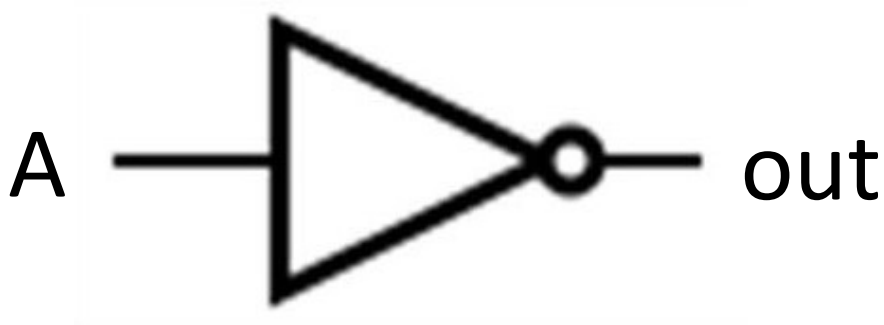
Logical gates

- The smallest pieces of hardware that we will examine are called ***logical gates.***
- Every gate that we encounter will take **bits** as inputs and will emit one **bit** as output.



- Those gates can connect to each other in various different ways in order to create more complex circuits

Our first gate



A	out
0	1
1	0

- This gate is known as the **inverter.**
- It corresponds **exactly** to the negation operation in propositional logic!
  - Where 1, set True.
  - Where 0, set False

Our second gate

p	q	r
0	0	0
0	1	0
1	0	0
1	1	1

- Corresponds to:

Conjunction

Disjunction

Our second gate (AND gate)

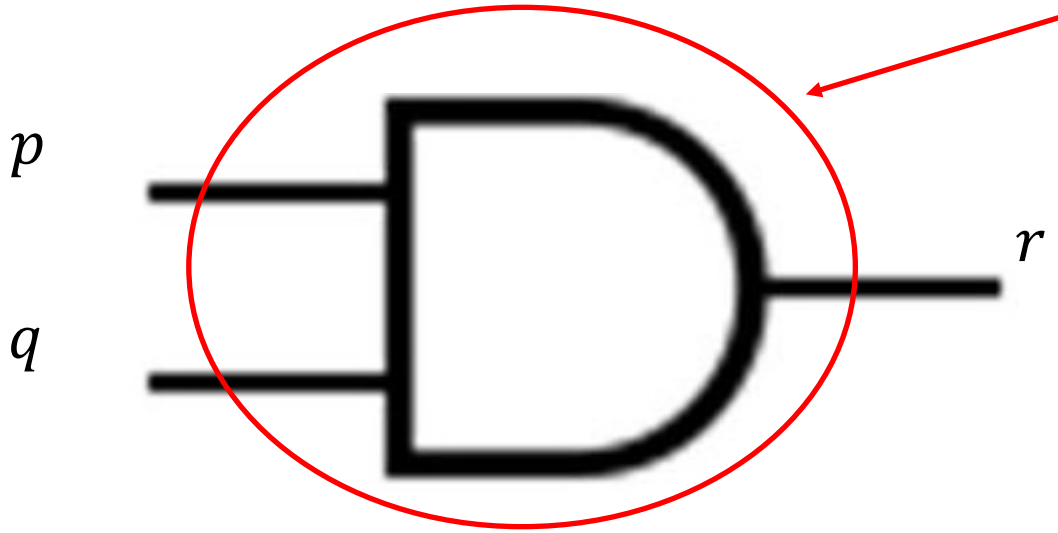
p	q	r
0	0	0
0	1	0
1	0	0
1	1	1

- Corresponds to:

Conjunction

Disjunction

Our second gate (AND gate)



p	q	r
0	0	0
0	1	0
1	0	0
1	1	1

- Corresponds to:

Conjunction

Disjunction

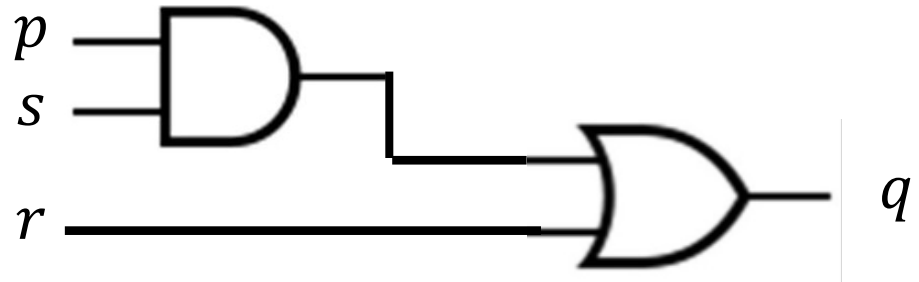
Our third gate (OR gate)

p	q	r
0	0	0
0	1	1
1	0	1
1	1	1

- Corresponds to logical disjunction (OR)

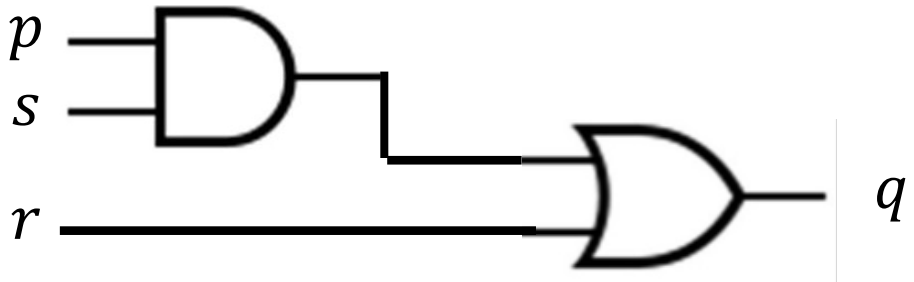
Exercises

- Which boolean function does this circuit correspond to?



Exercises

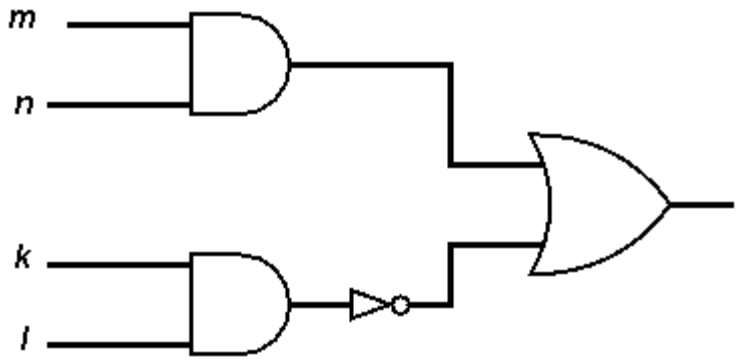
- Which boolean function does this circuit correspond to?



$(p \wedge s) \vee r$

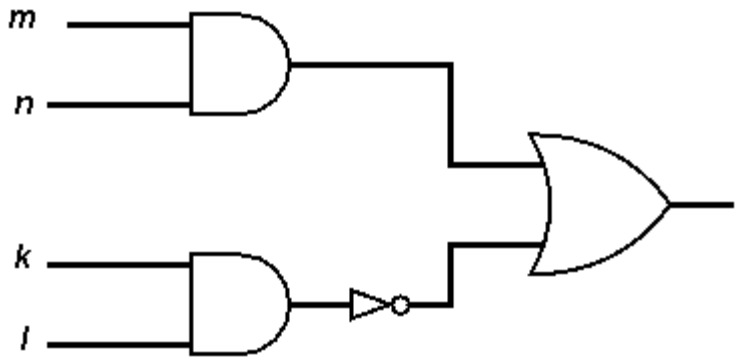
Exercises

- And this?



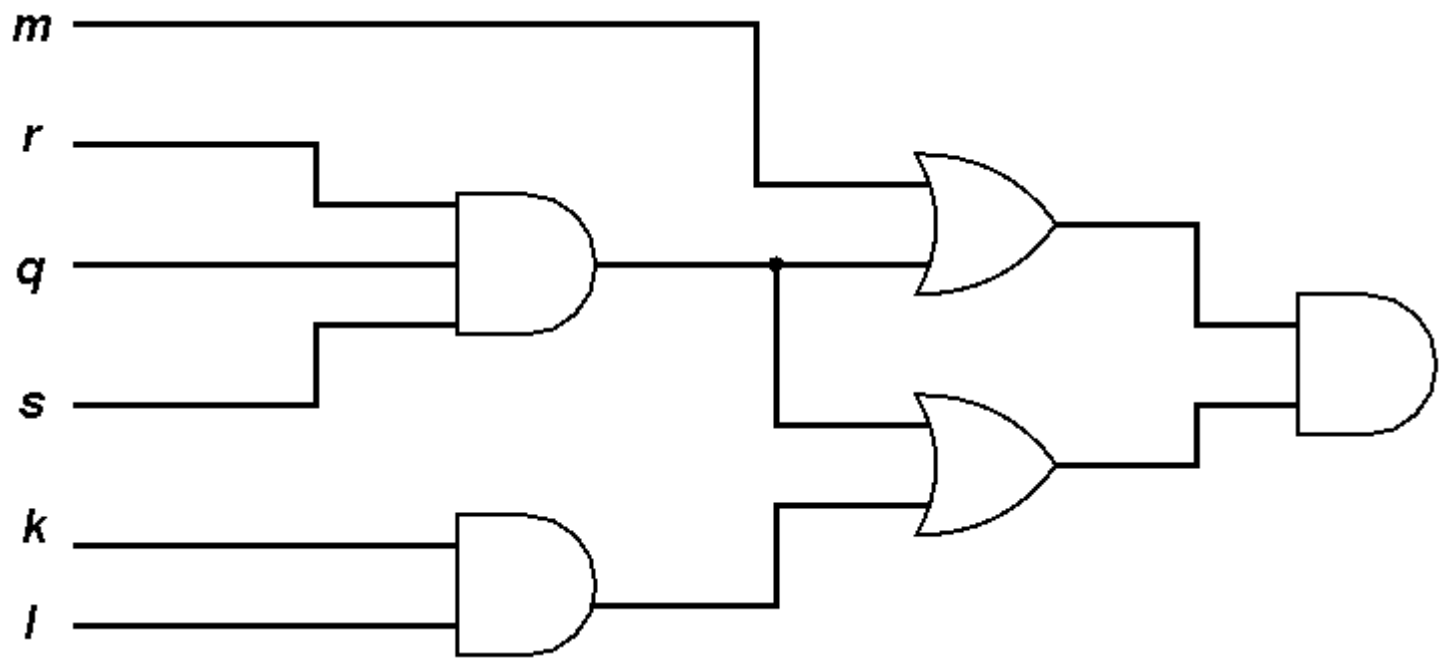
Exercises

- And this?

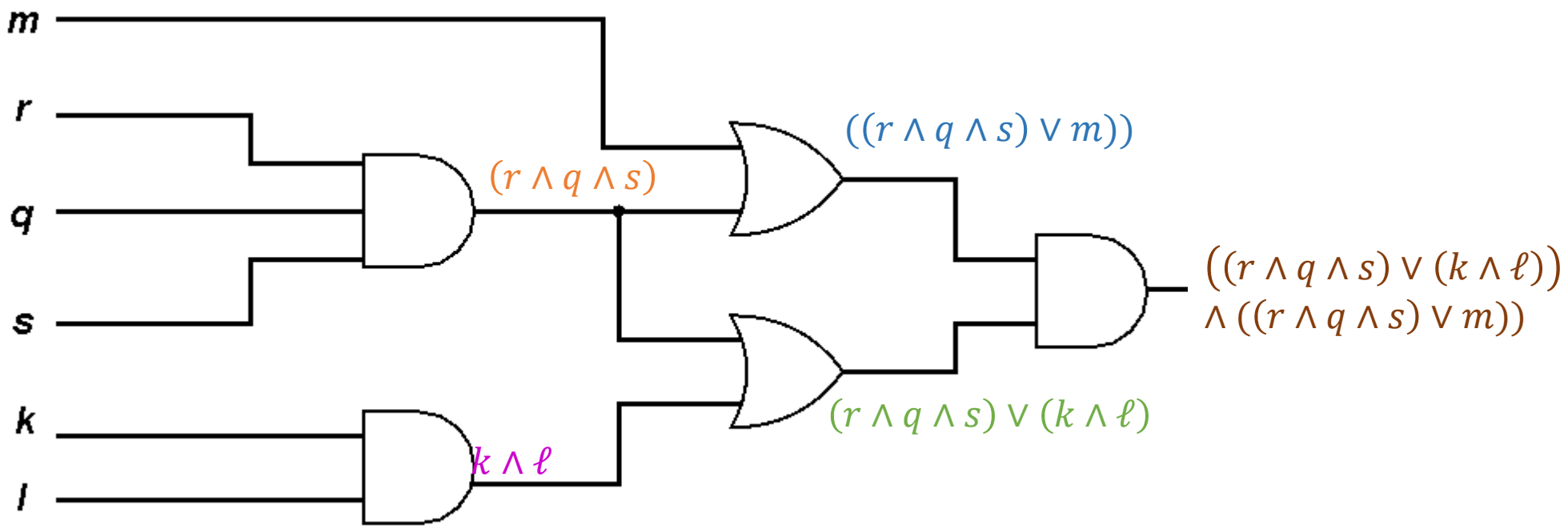


$$(m \wedge n) \vee (\sim (k \wedge l))$$

And this?



And this?



$$\big((r \wedge q \wedge s) \vee (k \wedge \ell)\big) \wedge ((r \wedge q \wedge s) \vee m))$$

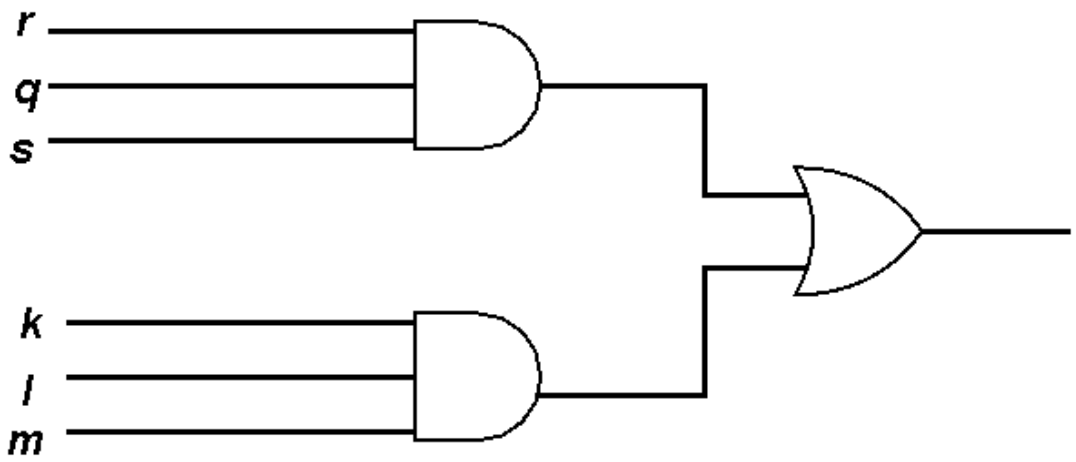
And this?

Can we make this circuit *cheaper*?

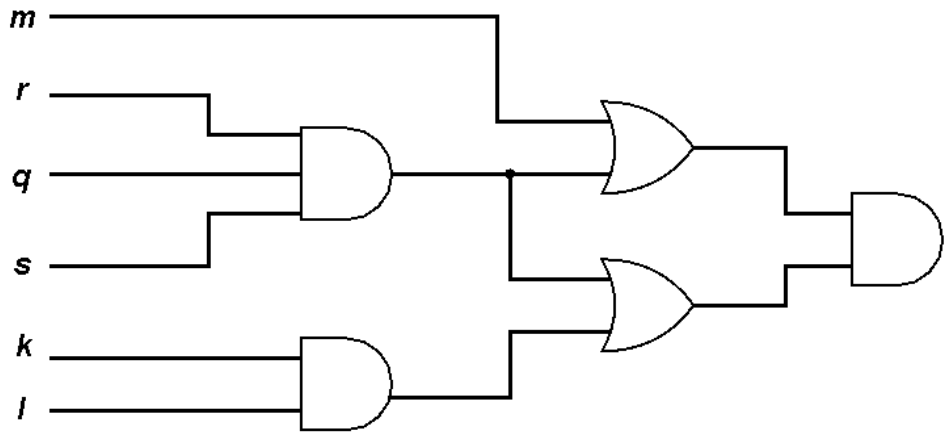
$$\big((r \wedge q \wedge s) \vee (k \wedge \ell)\big) \wedge ((r \wedge q \wedge s) \vee m))$$

Simplifying the circuit…

$$((r \wedge q \wedge s) \vee (k \wedge \ell)) \wedge ((r \wedge q \wedge s) \vee m))$$
$$\equiv (r \wedge q \wedge s) \vee ((k \wedge \ell) \wedge m)$$



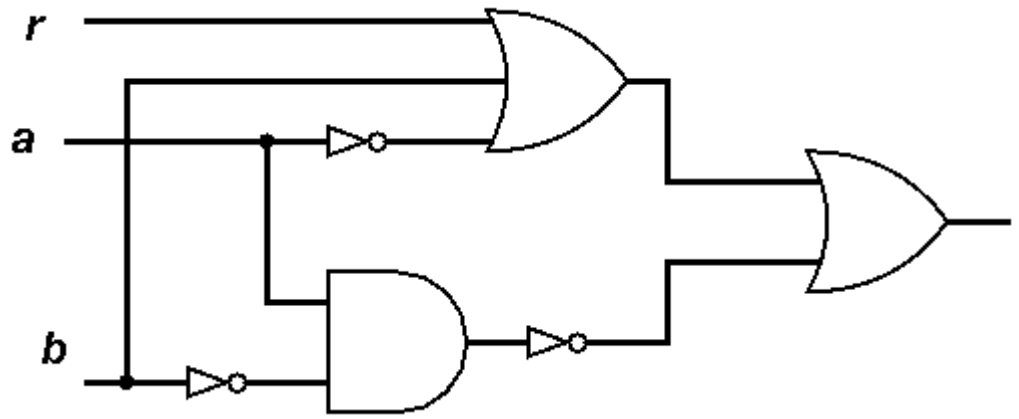
New circuit: Three gates



Old circuit: Five gates

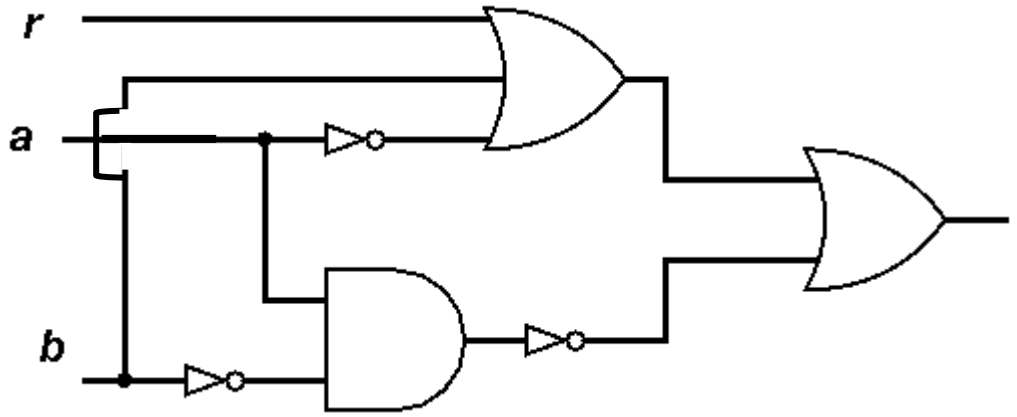
Exercises

1. Which logical expression is computed by the following circuit?



Exercises

1. Which logical expression is computed by the following circuit?
2. ***Simplify*** the circuit as much as possible!



Coming back to our original formula…

$$(\sim p \wedge \sim q \wedge \sim r) \vee (\sim p \wedge q \wedge r) \vee (p \wedge \sim q \wedge \sim r) \vee (p \wedge q \wedge r)$$

Coming back to our original formula…

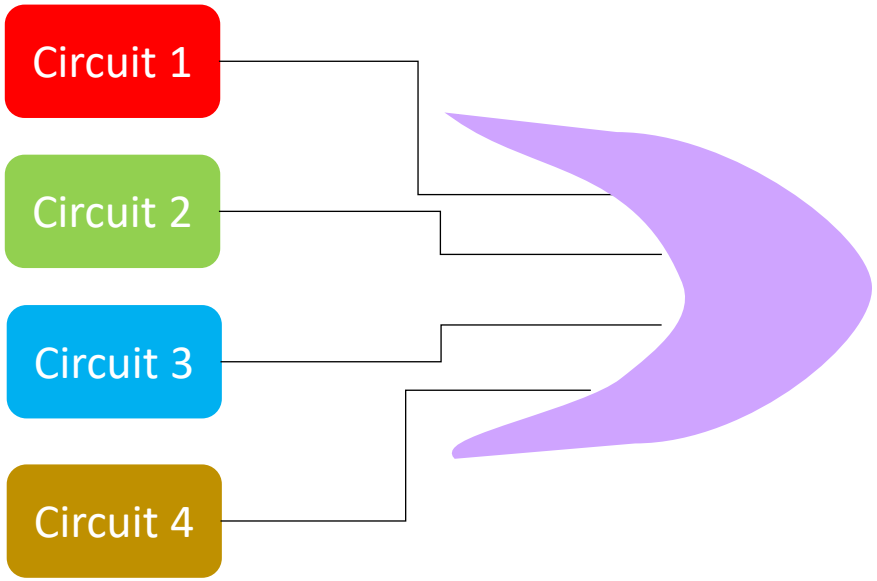
$$(\sim p \wedge \sim q \wedge \sim r) \vee (\sim p \wedge q \wedge r) \vee (p \wedge \sim q \wedge \sim r) \vee (p \wedge q \wedge r)$$

- For each small formula we have a circuit, and we will combine with a 4-input OR gate!

Coming back to our original formula…

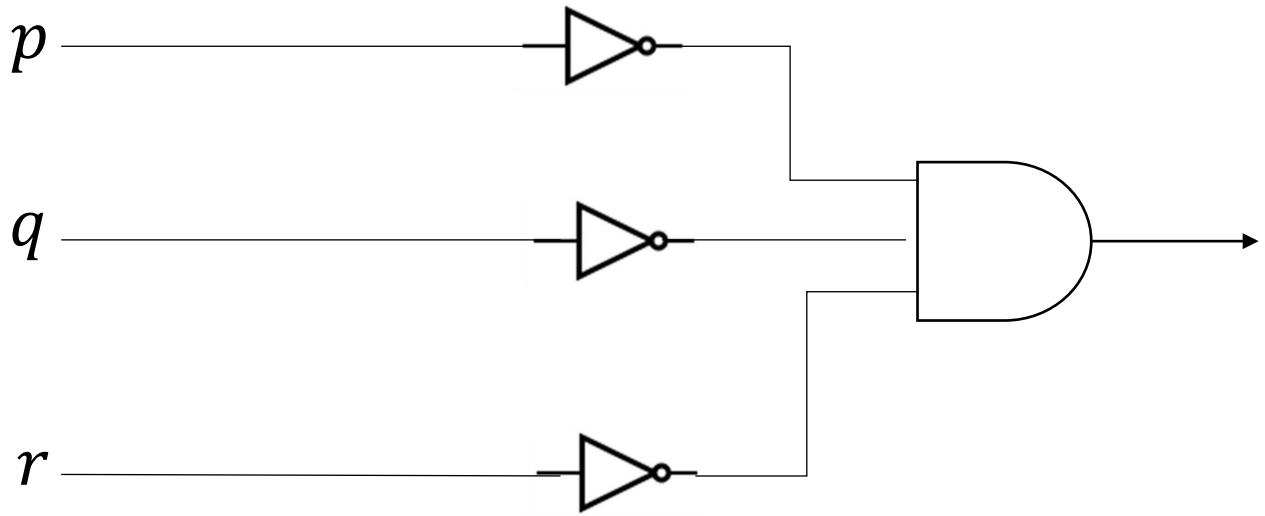
$$(\sim p \wedge \sim q \wedge \sim r) \vee (\sim p \wedge q \wedge r) \vee (p \wedge \sim q \wedge \sim r) \vee (p \wedge q \wedge r)$$

- For each small formula we have a circuit, and we will combine with a 4-input OR gate!



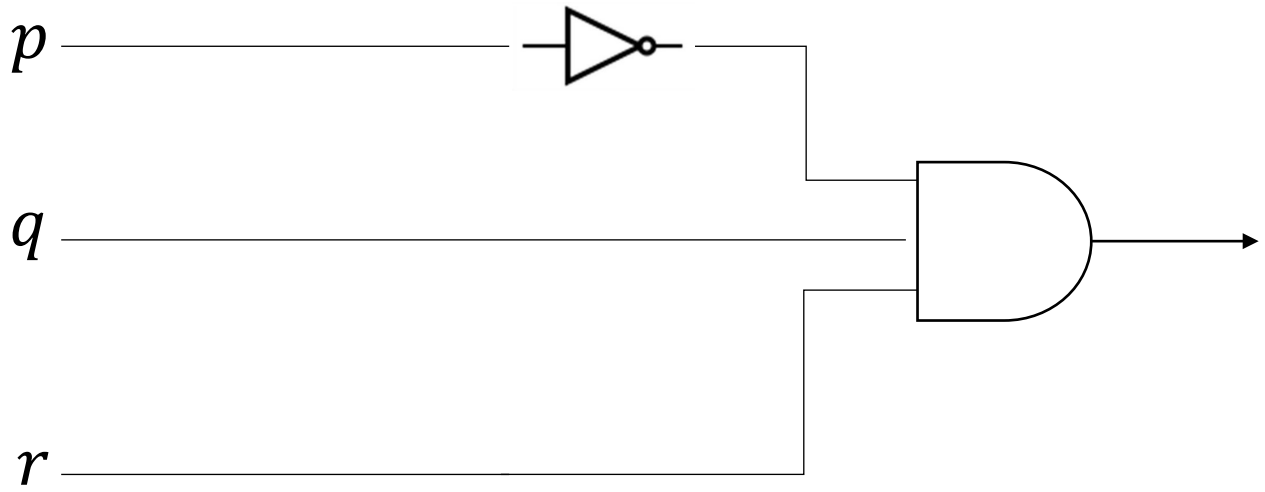
Circuit 1

$(\sim p \wedge \sim q \wedge \sim r)$



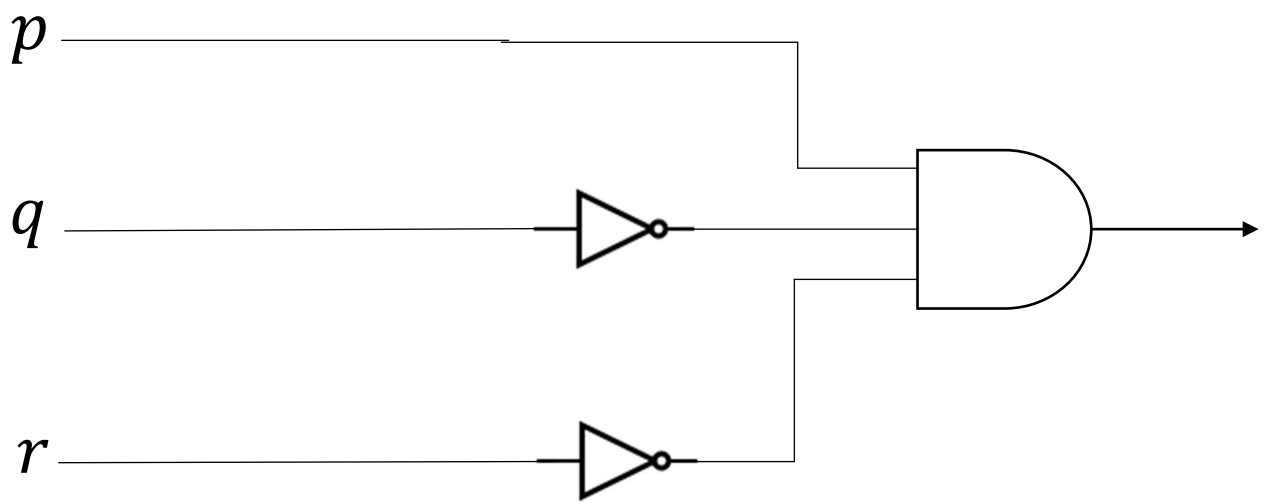
Circuit 2

$(\sim p \wedge q \wedge r)$



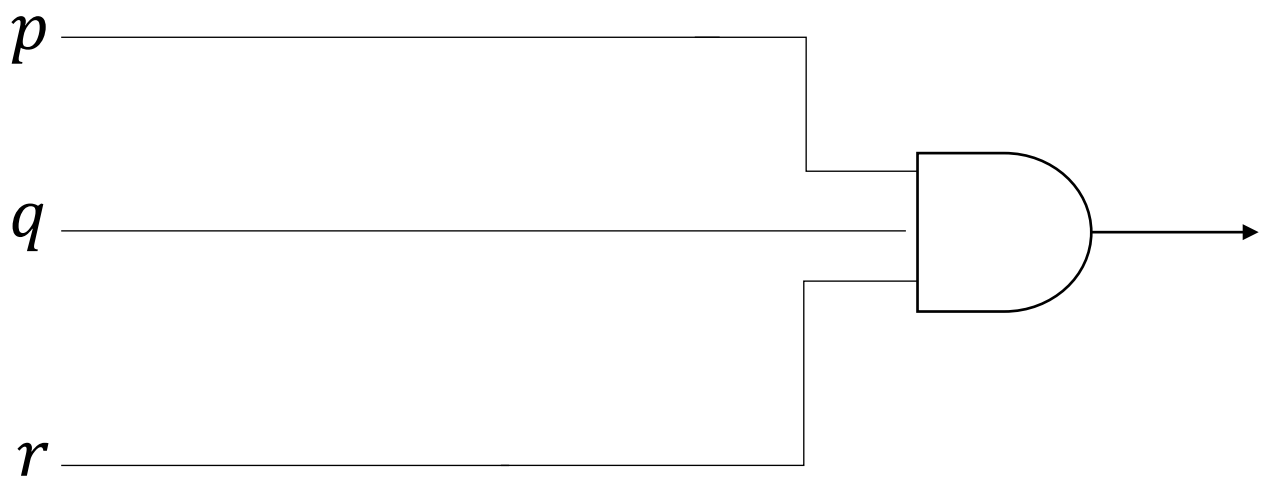
Circuit 3

$$(p \wedge \sim q \wedge \sim r)$$



Circuit 4

$(p \wedge q \wedge r)$



Midterm news / reminders

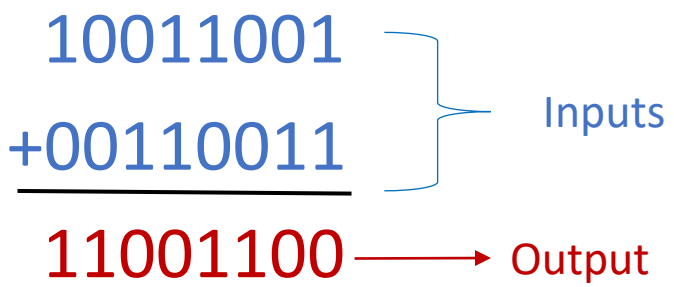
- Panic-free midterm!
- We will take an extra day or two to publish solutions
  - In case somebody has had an emergency / ER / hospital visit / any other emergency.
  - Monday: you will cover the midterm and have questions from today answered.
- Today: Adder circuits!
  - This concludes the logic portion!
- Next week: Set Theory

Midterm news / reminders

- Next week: Set Theory and, possibly, quantifiers (∀, ∃) on Thursday
  - I will probably have you ensure **on your own** what ⊆,∩,∪ and ∈ **mean. Nothing more.**
  - Either textbook, or Wikipedia / Wolfram Alpha are good resources.
- Remember: SpringBreak in two weeks.
  - Next week's homework will be designed for one week, but will have a 2-week deadline through 03-22.
  - There **will** be a short quiz **next week**, which you will still have 1 week and 3 days to submit (Monday 03-22, 11:59pm, like your homework)
    - But come on man, it's a short quiz.

Building Adder Circuits

- We want to build circuits that add arbitrarily large binary numbers.
- E.g



Half-Adder

- A half-adder is a circuit that adds **two bits** together!

$$\begin{array}{r} X \\ +\ Y \\ \hline C\ S \end{array}$$

- (Remember: C is the carry bit.)
- Let's try to build a circuit that computes both S and C!

Truth table

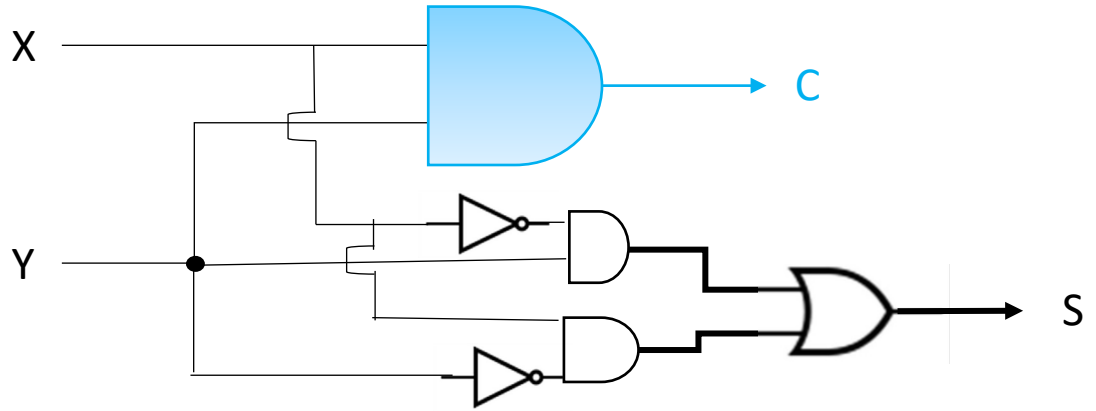
X	Y	S	C
0	0	?	?
0	1	?	?
1	0	?	?
1	1	?	?

Truth table

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

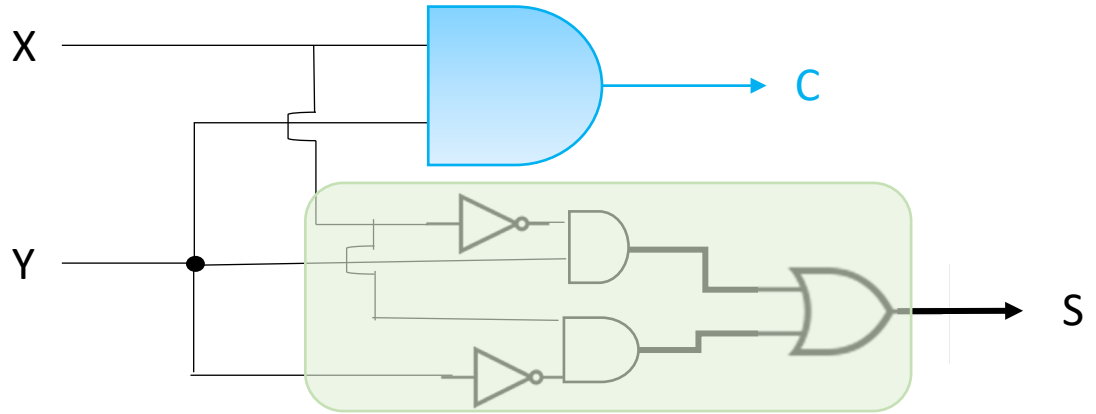
Truth table

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Truth table

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



XOR Gate
("Exclusive OR)

Making XOR cheaper

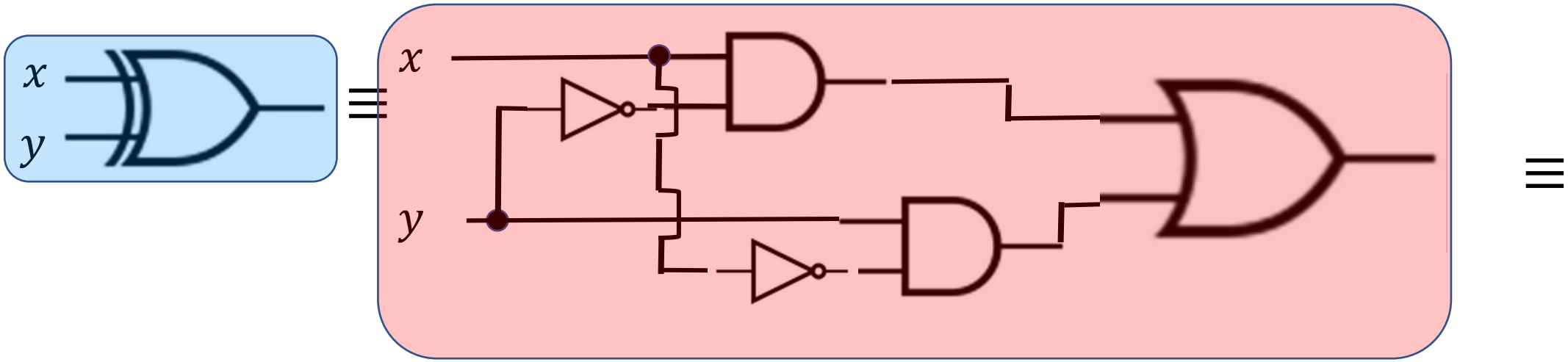
- First, let's convince ourselves that

$$(x \oplus y) \equiv (x \wedge (\sim y)) \vee ((\sim x) \wedge y) \equiv (x \vee y) \wedge (\sim(x \wedge y))$$

Making XOR cheaper

- First, let's convince ourselves that

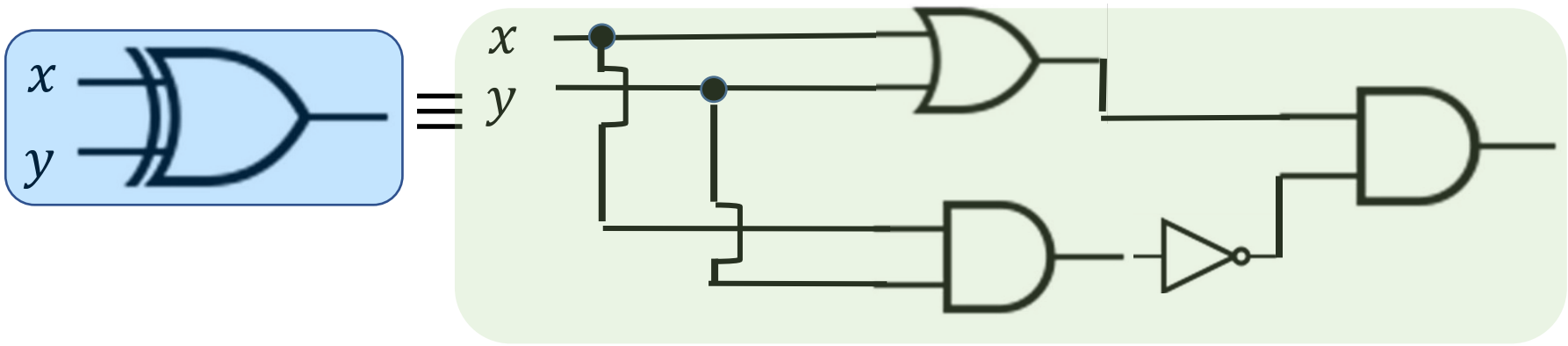
$$(x \oplus y) \equiv (x \wedge (\sim y)) \vee ((\sim x) \wedge y) \equiv (x \vee y) \wedge (\sim(x \wedge y))$$



Making XOR cheaper

- First, let's convince ourselves that

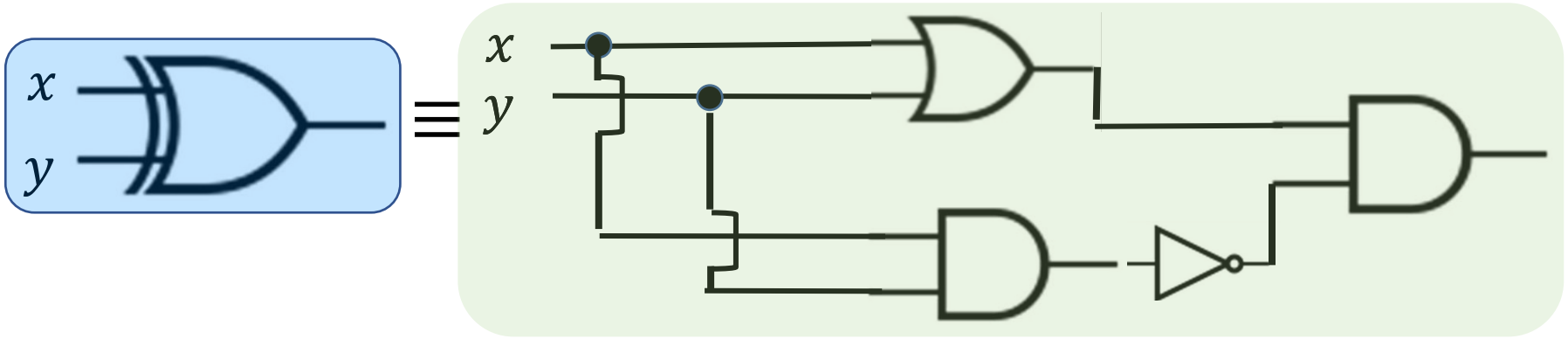
$$(x \oplus y) \equiv (x \wedge (\sim y)) \vee ((\sim x) \wedge y) \equiv (x \vee y) \wedge (\sim(x \wedge y))$$



Making XOR cheaper

- First, let's convince ourselves that

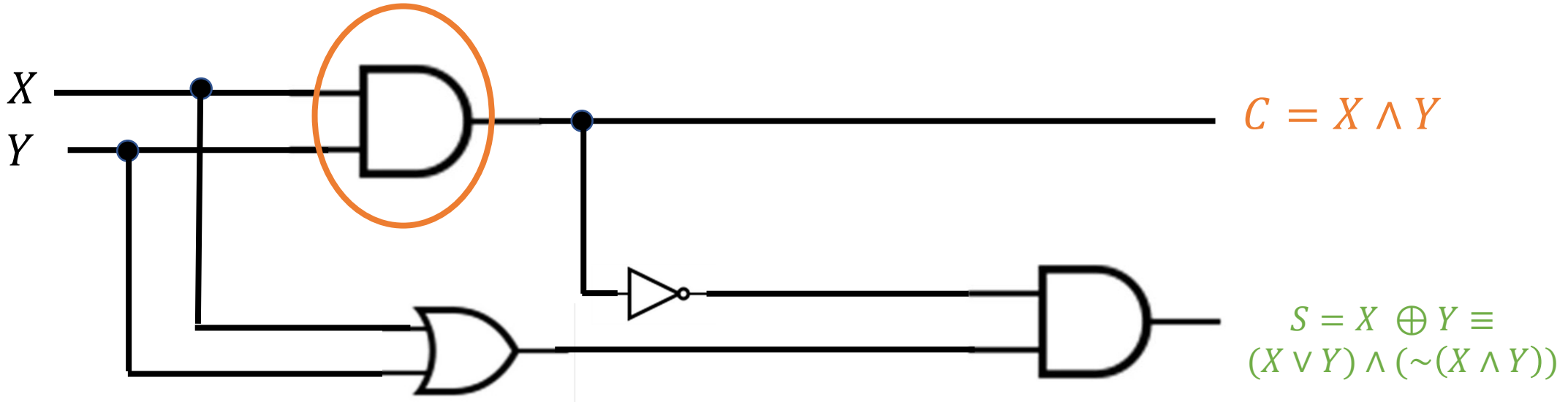
$$(x \oplus y) \equiv (x \wedge (\sim y)) \vee ((\sim x) \wedge y) \equiv (x \vee y) \wedge (\sim(x \wedge y))$$



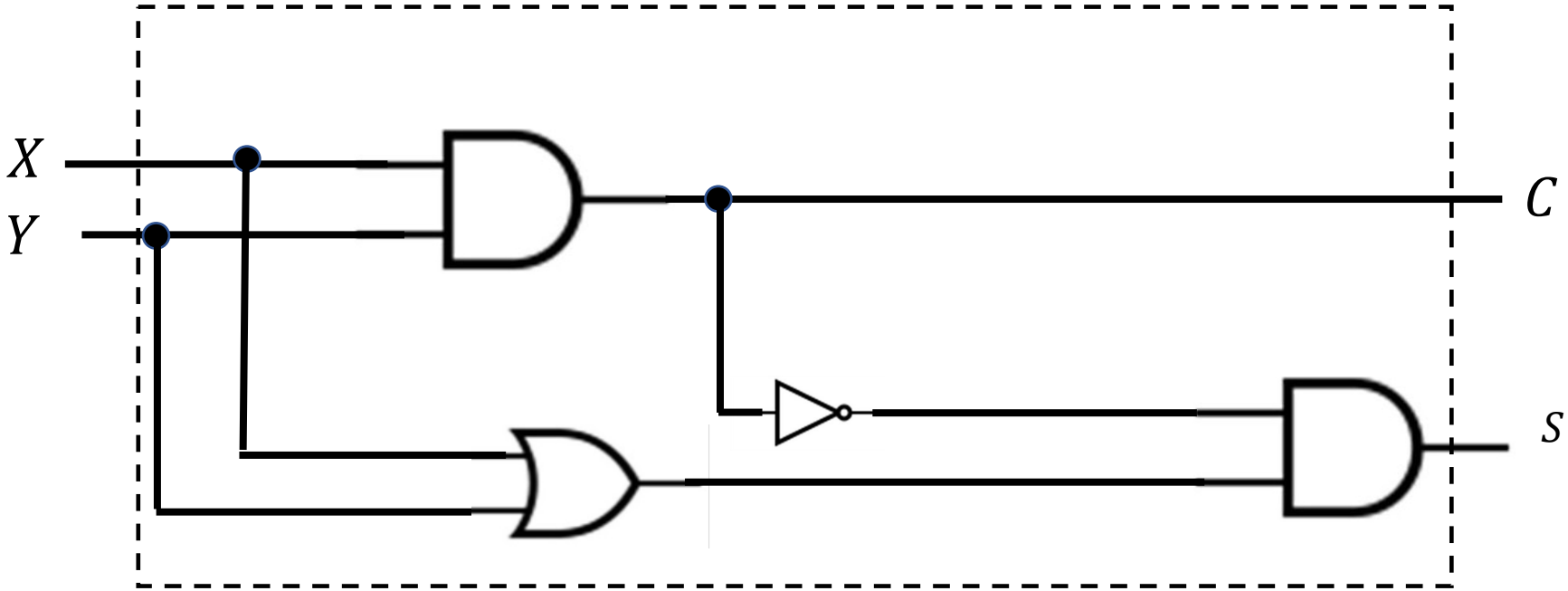
From **five** gates to **four**!

Optimizing Half Adder

- We can now optimize the Half Adder.
- We won't just use simplified XOR, but also leverage simplified XOR to **re-use** the AND gate used to compute the carry bit C !

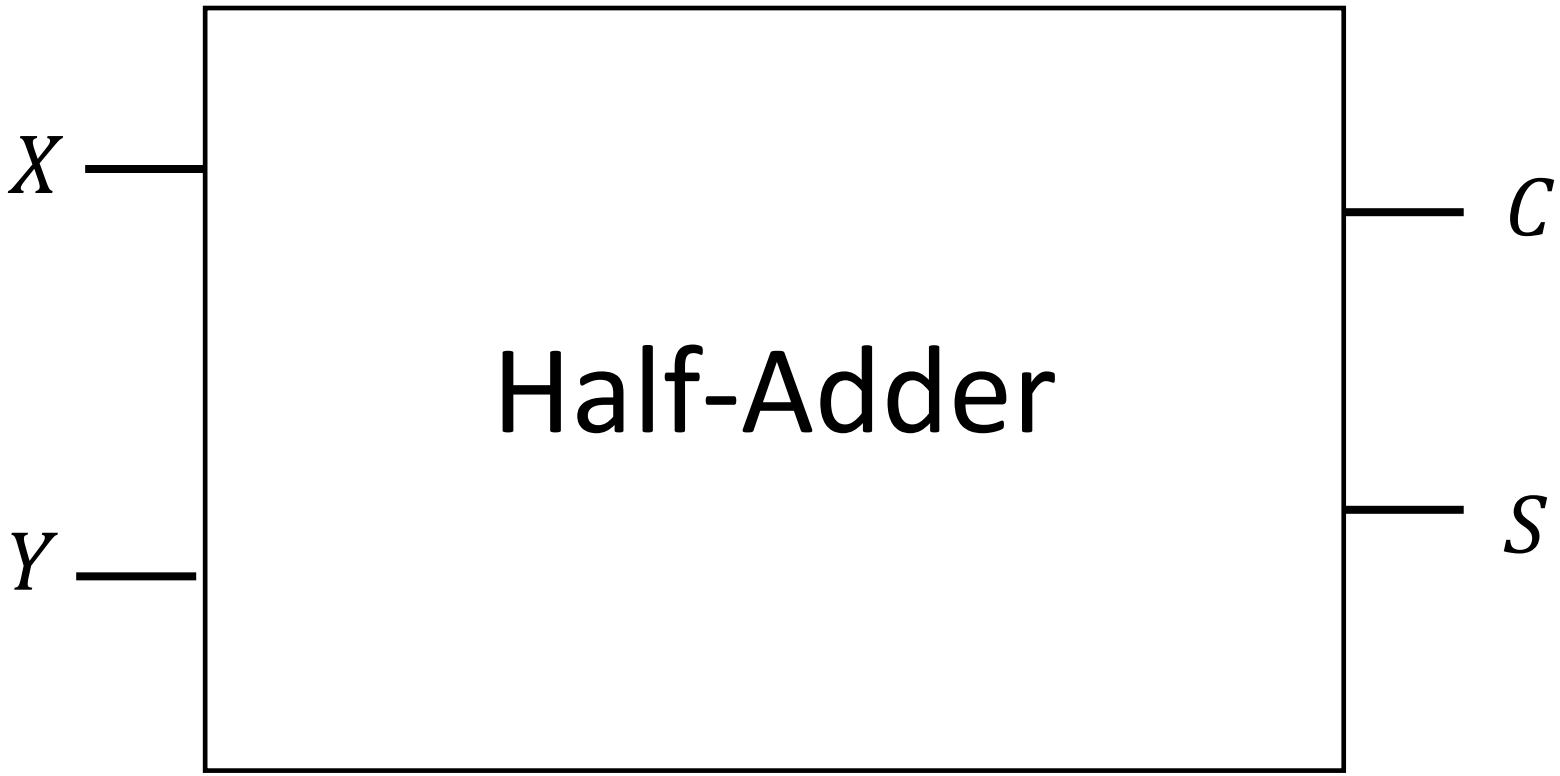


Half Adder Abstraction



4 gates, instead of 6 for the previous one!

Half Adder Abstraction



Full-Adder

- Now, let's consider the complete case, where we want to build a circuit that computes the sum of two 2-digit binary numbers:

$$\begin{array}{r} P\ Q \\ +\ W\ X \\ \hline C\ S_1\ S_2 \end{array}$$

- To do this, we also need the ability to add 3 digits, because:

$$\begin{array}{r} C_1 \\ P\ Q \\ +\ W\ X \\ \hline C\ S_1\ S_2 \end{array}$$

Full-Adder

- Now, let's consider the complete case, where we want to build a circuit that computes the sum of two 2-digit binary numbers:

$$\begin{array}{r} P\ Q \\ +\ W\ X \\ \hline C\ S_1\ S_2 \end{array}$$

- To do this, we also need the ability to add 3 digits, because:

$$\begin{array}{r} C_1 \\ P\ Q \\ +\ W\ X \\ \hline C\ S_1\ S_2 \end{array}$$

We will call a circuit that adds 3 bits a full adder

We could do the truth table….

P Q
+ W X
C S1 S2

P	Q	W	X	C	S_1	S_2
0	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			
0	1	1	1			
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			

We could do the truth table….

P Q
+ W X
C S1 S2

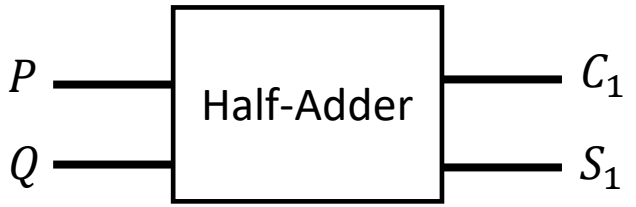
P	Q	W	X	C	S_1	S_2
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			

But it’s time consuming and we are all busy people

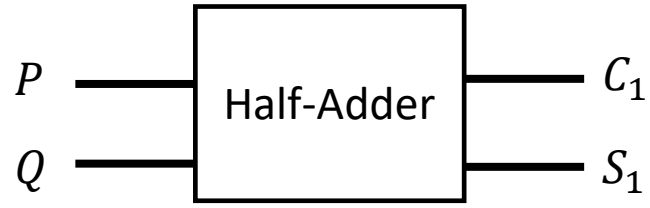
Constructing a Full-Adder in another way

- We need to build a circuit that computes the sum of 3 digits, e.g P + Q + R

- Step 1: Compute $\begin{array}{r} P \\ +\ Q \\ \hline C_1S_1 \end{array}$ with a **half-adder**:



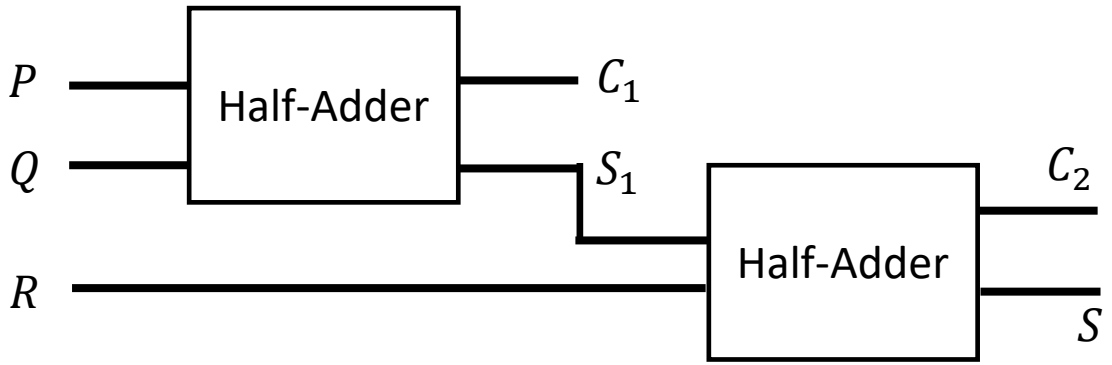
Constructing a Full Adder



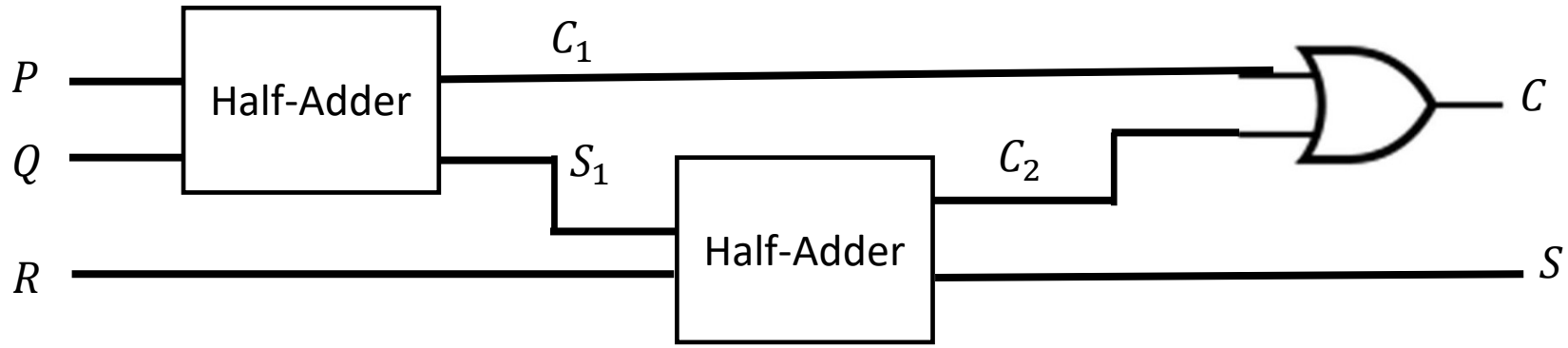
S_1

- Step 2: Compute $+\ R$ with another half-adder:

C_2S

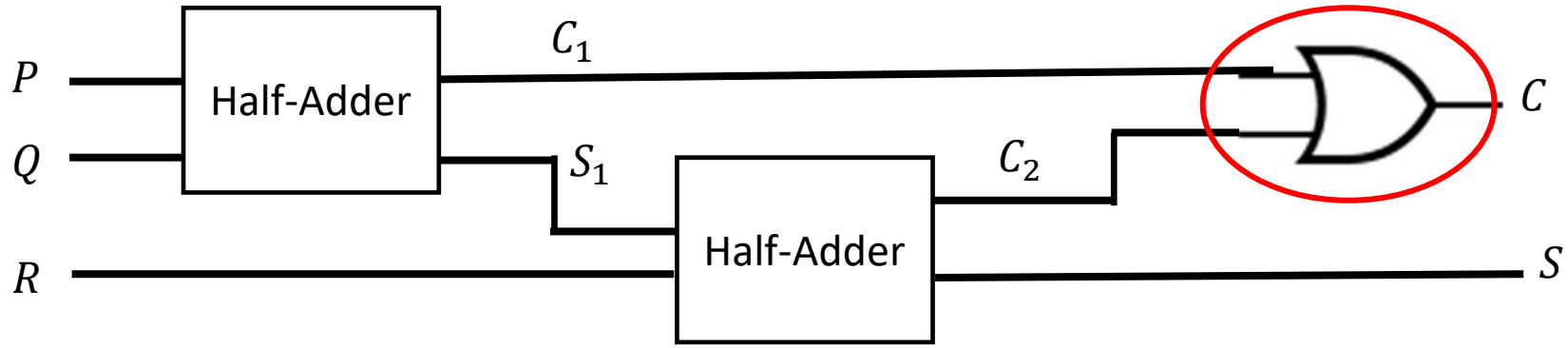


Constructing a full-adder



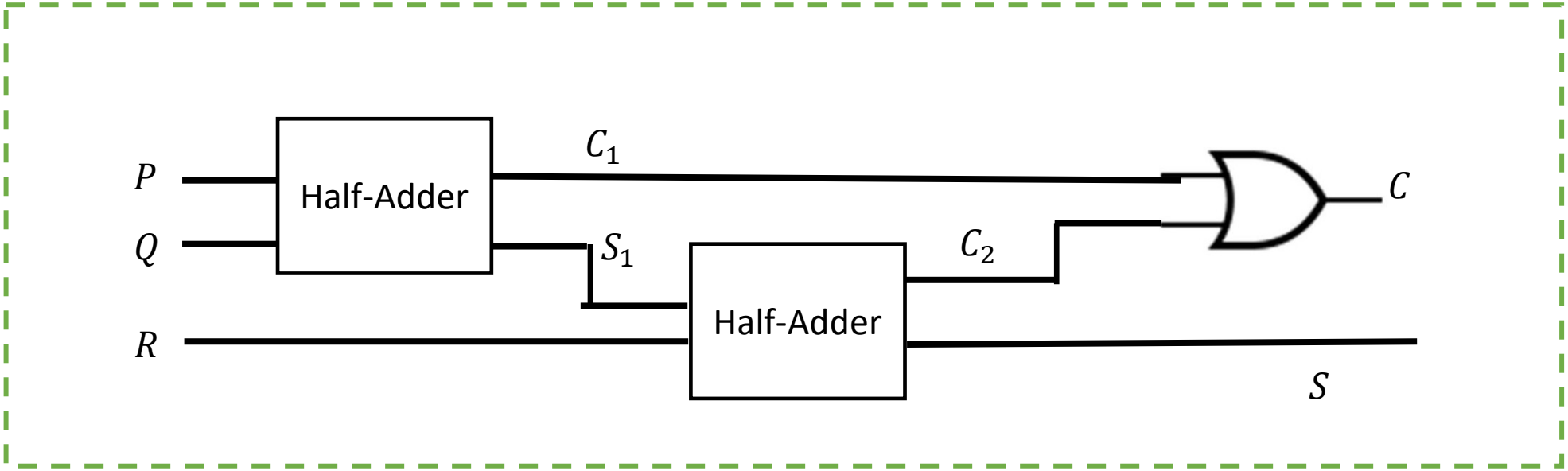
- Step 3: Combine C_1 and C_2 with an OR gate to yield the final carry bit C.

Constructing a full-adder



- Step 3: Combine C_1 and C_2 with an OR gate to yield the final carry bit C.
- Why did we choose an OR gate to combine the "intermediate" carries C_1 and C_2?

Constructing a full-adder



- Step 3: Combine C_1 and C_2 with an OR gate to yield the final carry bit C.

Abstraction time!

Full Adder Black Box

- 3 inputs, 2 outputs

2-bit adder

- However, **we still have not solved our original problem**, which is to construct a circuit that adds 2-bit numbers!

P Q

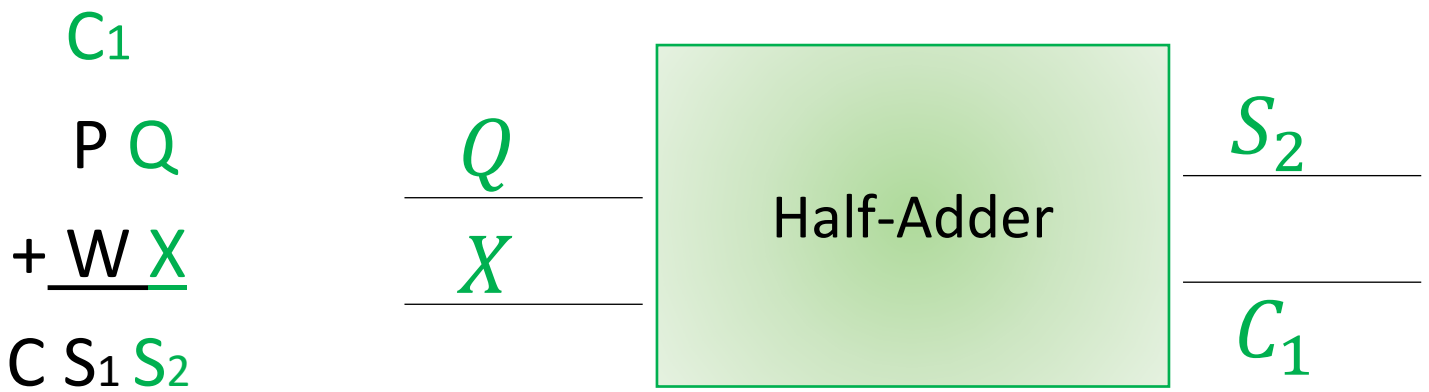
\+ W X

C S_1 S_2

- So, we need a circuit that takes 4 inputs and emits 3 outputs:

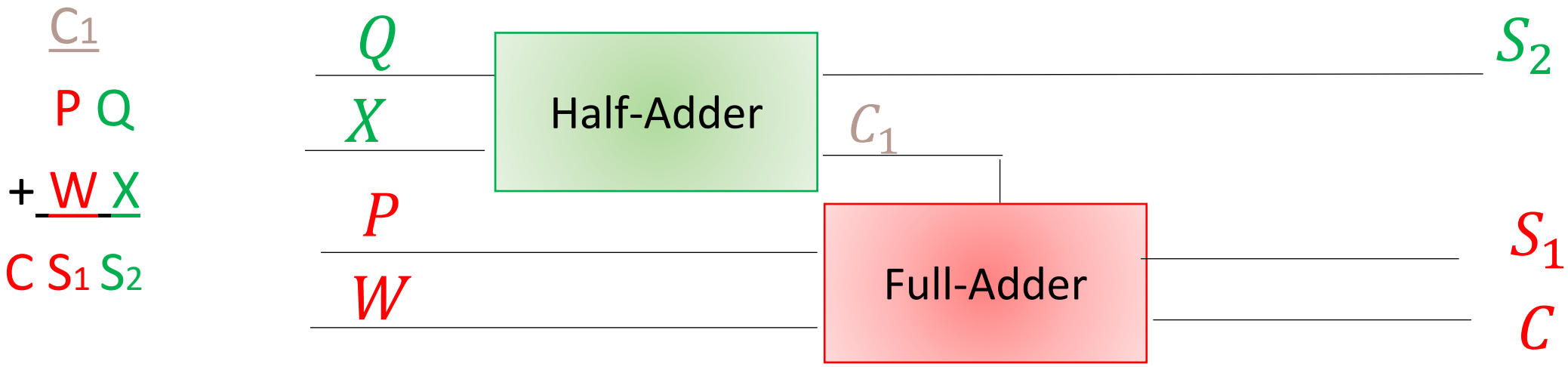
Constructing a 2-bit adder

- Step 1: Take care of the right-most column with a half-adder:



Constructing a 2-bit adder

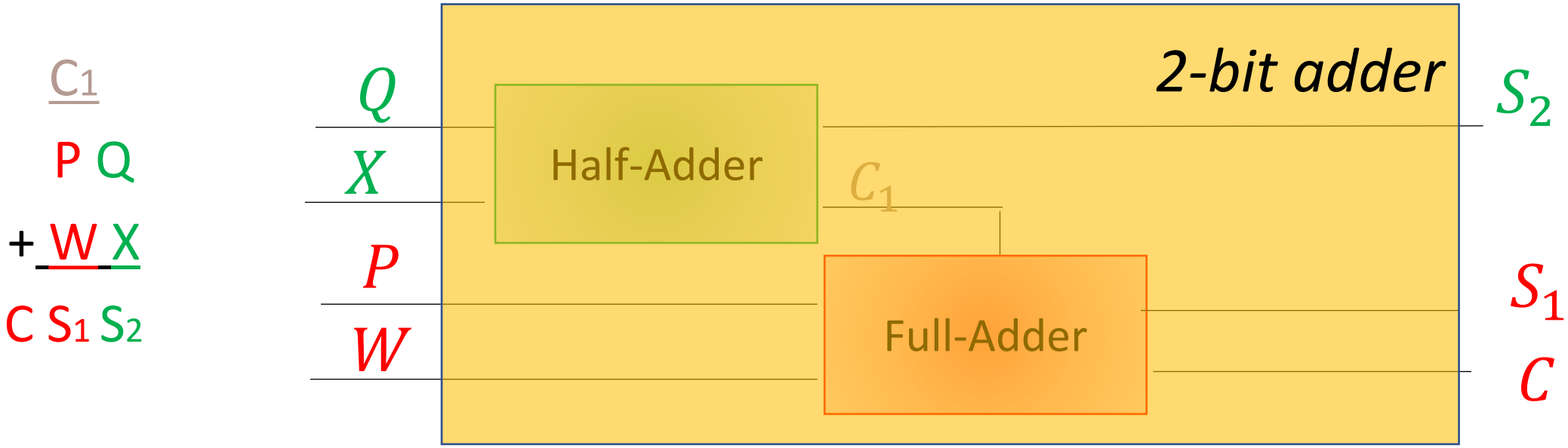
- Step 1: Take care of the right-most column with a half-adder:



- Step 2 (and final): Connect Half-Adder and new inputs to Full-adder appropriately to produce final circuit.

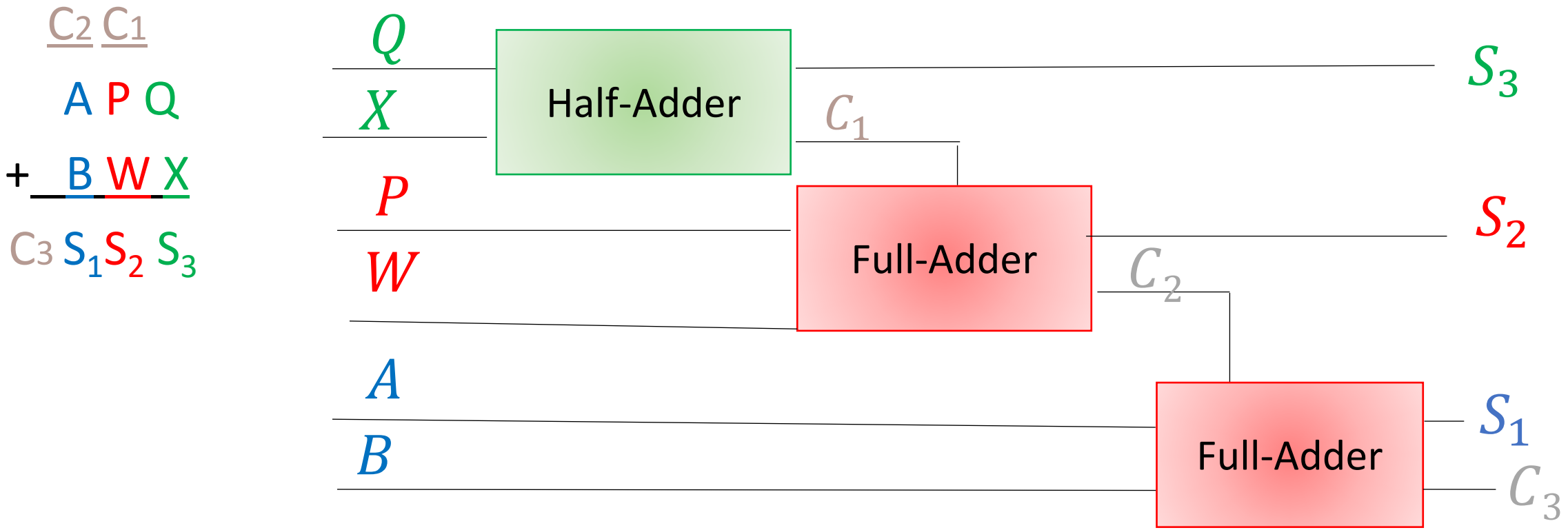
Constructing a 2-bit adder

- Step 1: Take care of the right-most column with a half-adder:

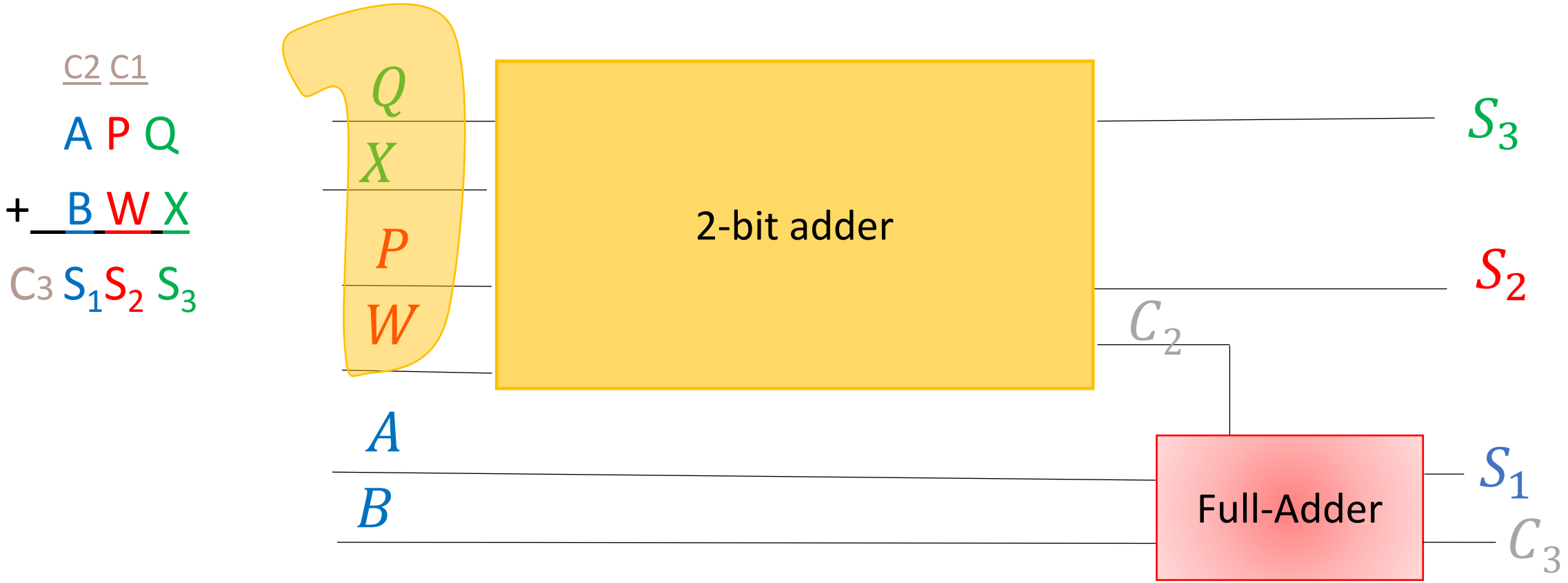


- Step 2 (and final): Connect Half-Adder and new inputs to Full-adder appropriately to produce final circuit.

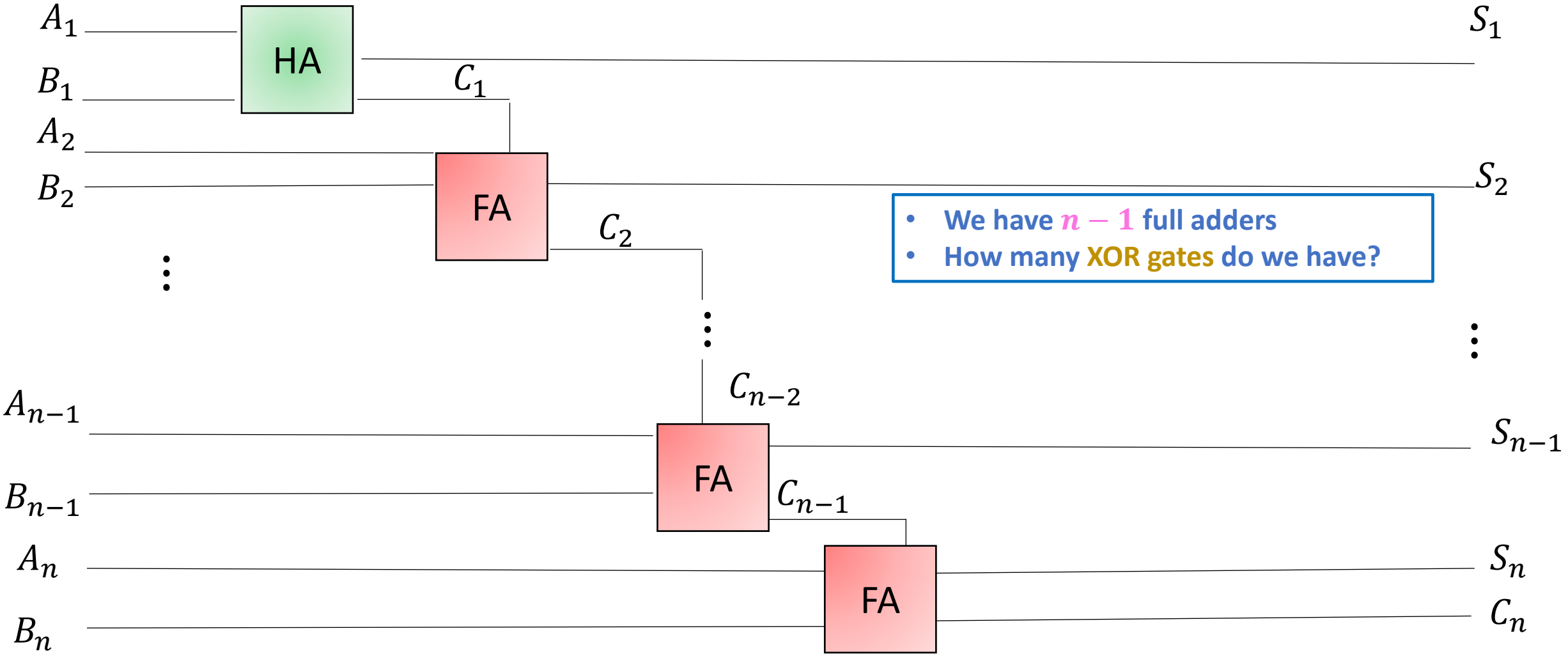
Constructing a 3-bit adder (messy)



Constructing a 3-bit adder (neat)

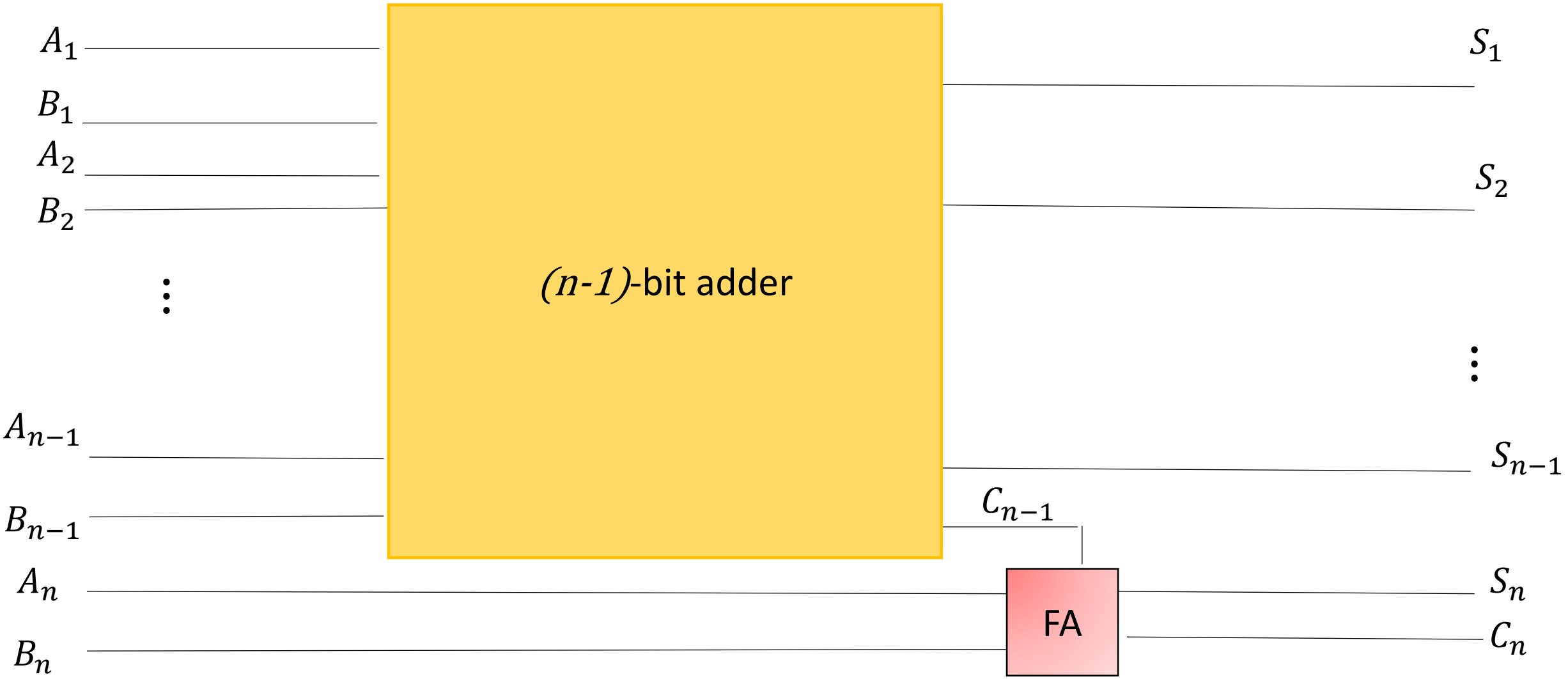


Constructing an n-bit adder (messy)



- We have $n - 1$ full adders
- How many XOR gates do we have?

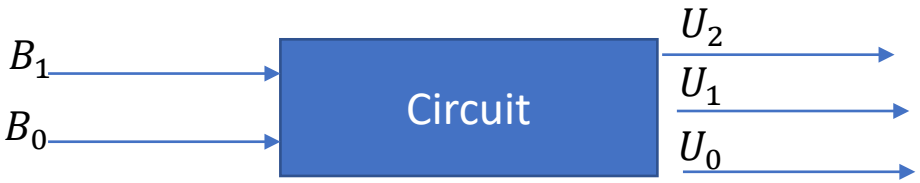
Constructing an n-bit adder (neat)



Fun exercise

- Input: number in **binary** (B_0, B_1)
- Output (in U_0, U_1, U_2): ???

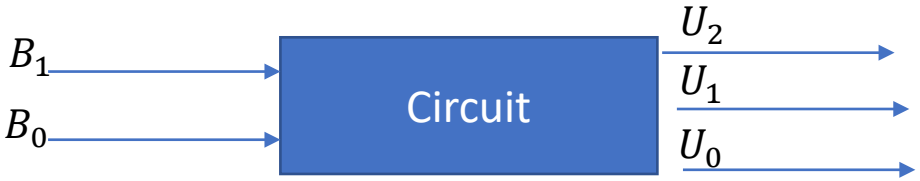
B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1



Fun exercise

- Input: number in **binary** (B_0, B_1)
- Output (in U_0, U_1, U_2): Binary to Unary

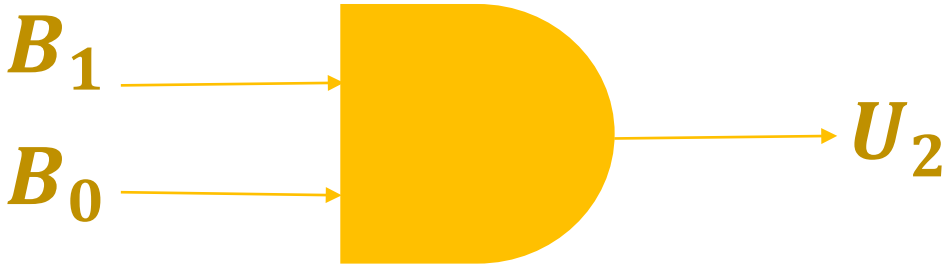
B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1



First micro-circuit

B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

$$U_2 = B_1 \wedge B_0$$



Second micro-circuit

B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

$$U_1 = (B_1 \wedge \sim B_0) \vee (B_1 \wedge B_0)$$

Second micro-circuit

B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

$$U_1 = (B_1 \wedge \sim B_0) \vee (B_1 \wedge B_0) = B_1$$

(from distributive law of conjunction over disjunction!)

$$B_1 \longrightarrow U_1$$

Third micro-circuit

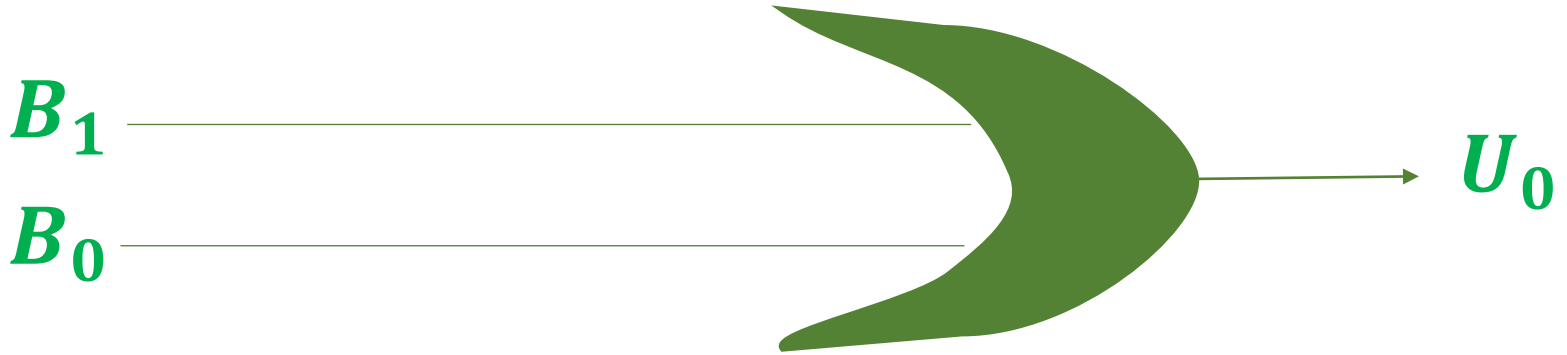
B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

$$U_0 = (\sim B_1 \wedge B_0) \vee (B_1 \wedge \sim B_0) \vee (B_1 \wedge B_0)$$

Third micro-circuit

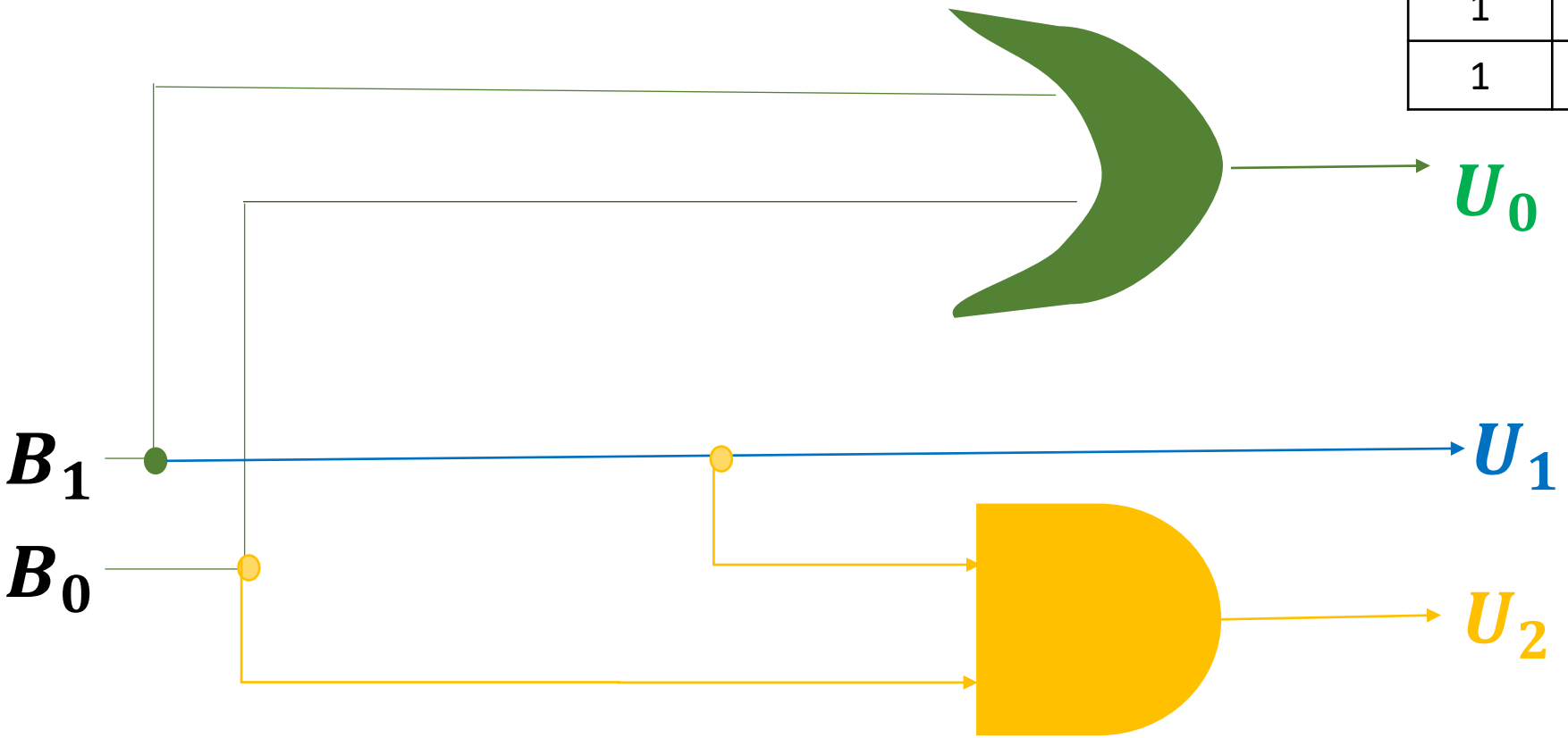
B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

$$U_0 = (\sim B_1 \wedge B_0) \vee (B_1 \wedge \sim B_0) \vee (B_1 \wedge B_0) = (\sim B_1 \wedge B_0) \vee B_1 = (\sim B_1 \vee B_1) \wedge (B_0 \vee B_1) = B_0 \vee B_1$$



Final circuit

B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1



Other numeric functions

- Addition (have done)
- Multiplication
- Division
- Primality test (test whether a number is prime)

- There are circuits for all of these!
  - Computers actually work this way at the base level: they consist of gates.