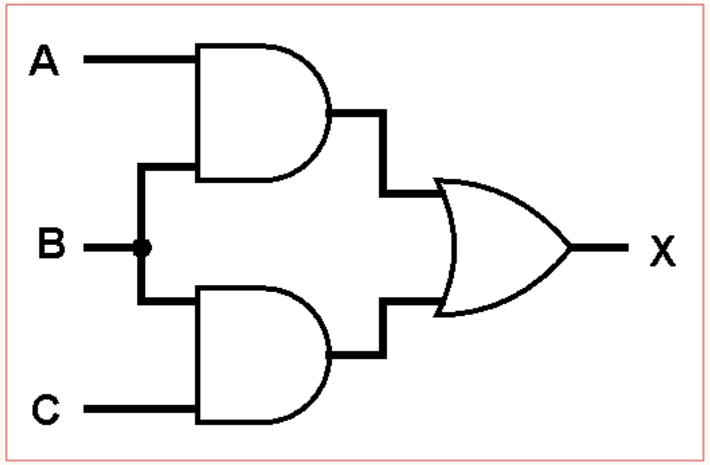
NAND

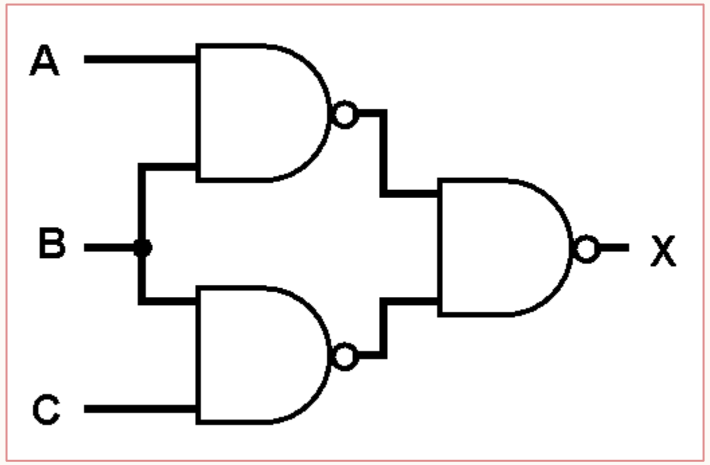
Kira Clements, University of Bristol

FUNCTIONALLY COMPLETE

NAND is **functionally complete**, meaning any boolean expression can be expressed just by a combination of NAND operators. This makes it an excellent building block and makes physical production more efficient as only one type of gate needs to be manufactured.



Any circuit can be re-implemented using NAND gates only



NOR is also functionally complete but NAND is the industry standard as it is structurally faster than NOR.

DOUBLE NEGATION

Although double negation is generally a method of simplifying expressions, by cancelling out the negations, this rule also allows us to add a double negation to an expression without changing its meaning i.e. the new expression will be logically equivalent.

DOUBLE NEGATION	
$\neg\neg A \equiv A$	Two negations cancel each other out

Combined with De Morgan's Laws, this creates a powerful tool for translating any boolean function into one expressed only using NANDs.

For example, these show that an OR gate is equivalent to a NAND gate with inverted inputs:

$$A \lor B \xrightarrow{\text{Double negation}} \neg\neg(A \lor B) \xrightarrow{\text{De Morgan's}} \neg(\neg A \land \neg B)$$

NOT IS NAND?

Double negation also shows that an AND gate is equivalent to a NAND gate with an inverted output:

$$A \land B \equiv \neg\neg(A \land B)$$

You might notice that both our NAND implementations so far include at least one negation other than those that are obviously part of a NAND. We are satisfied with this as negation is logically equivalent to a NAND with a copied input:

$$\neg A \equiv \neg(A \land A)$$

To show this more explicitly, our NAND implementation of AND could be expressed as:

$$\neg(\neg(A \land B) \land \neg(A \land B))$$

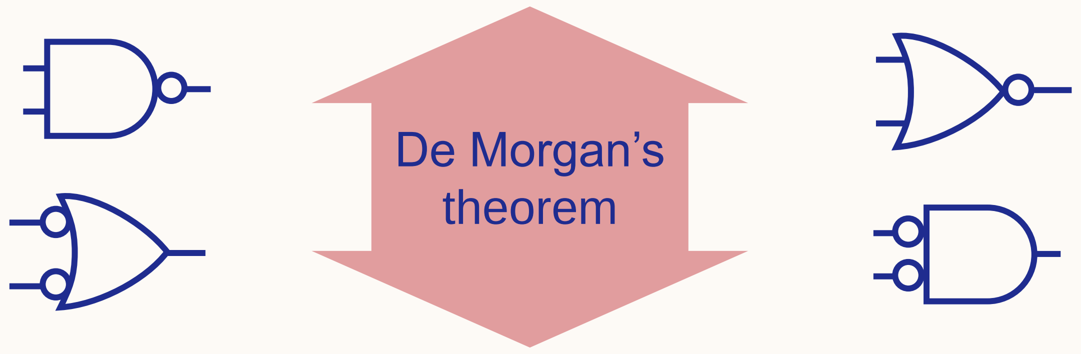
This means that once our logical expression is in a suitable form, we can count the number of NAND gates needed to implement it by **counting the number of negations**!

BUBBLE PUSHING

Looking at NAND implementations visually, we can consider each negation to be a *bubble* such as appears on the tips of NOT, NAND and NOR gates. **Bubble pushing** is then a technique to apply De Morgan's theorem directly to the logic diagram.

NAND

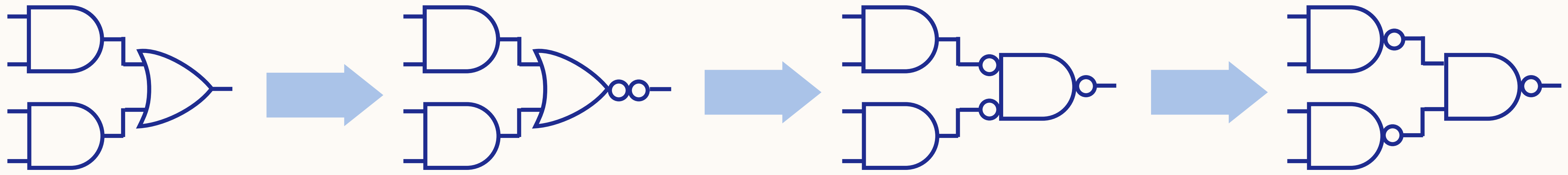
$\neg(A \land B) \equiv \neg A \lor \neg B$



NOR

$\neg(A \lor B) \equiv \neg A \land \neg B$

If we have a simplified circuit that only consists of AND, OR, and NOT gates, we then begin at the output and simply need to push bubbles or add double bubbles where appropriate to synthesise an equivalent NAND implementation.

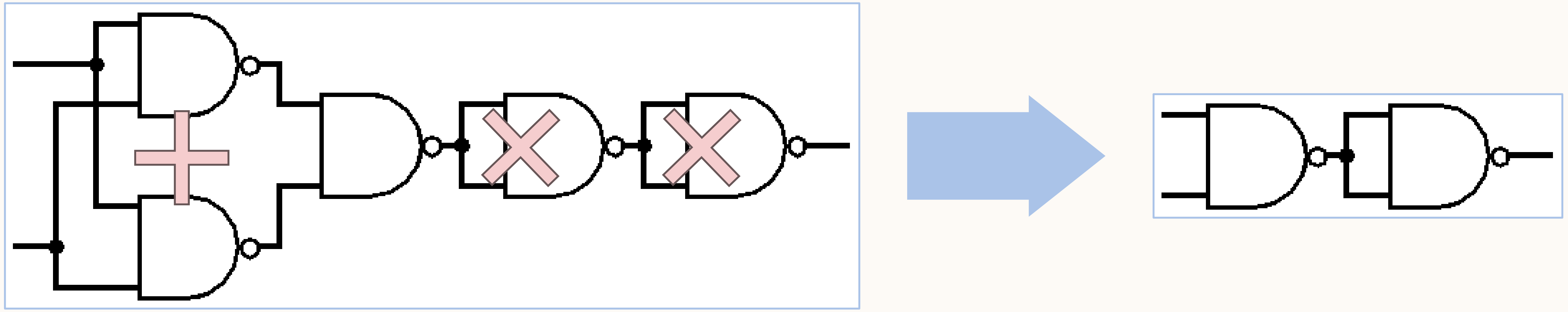


NAND CIRCUITS

NAND implementation can often be **simplified** in a variety of ways and it’s best to attempt simplification of the formula. However, there are some methods for simplifying the circuit after it’s been created:

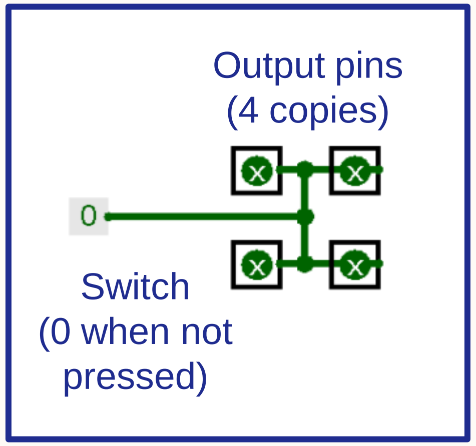
Remove 2 successive NANDs that act as a double negation.

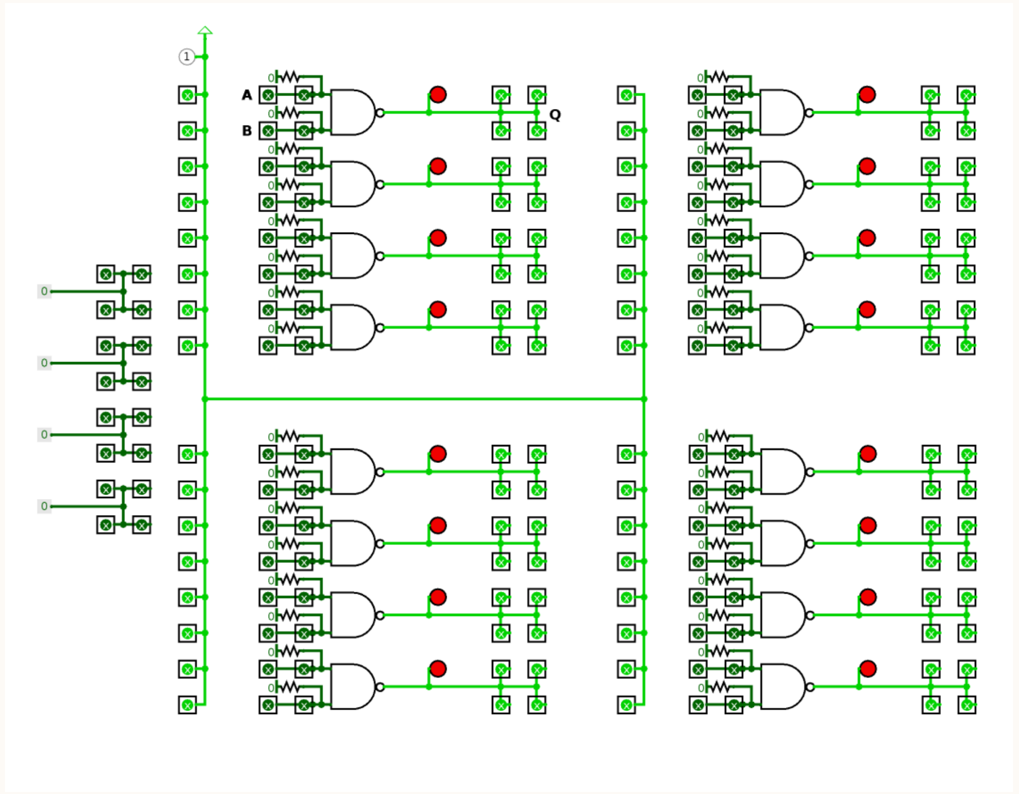
Combine NANDs that serve the same purpose by branching the output instead.



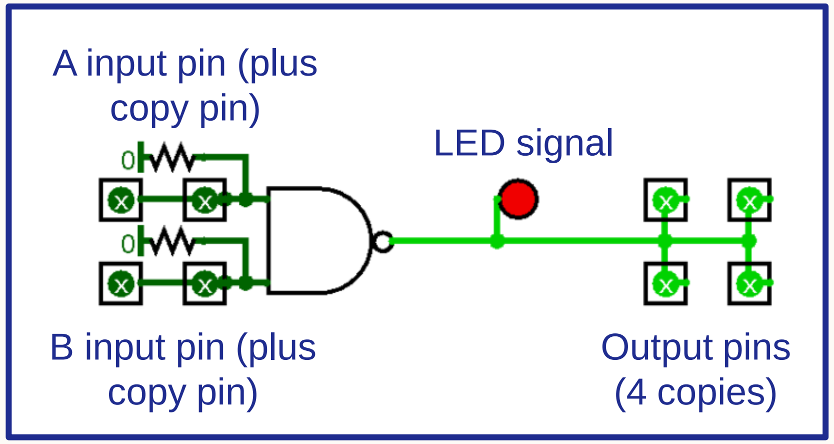
NAND BOARDS

4 x input switches





16 x NANDs

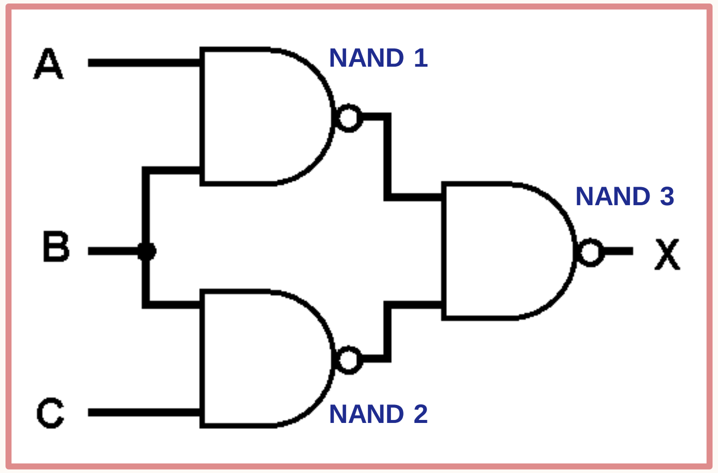


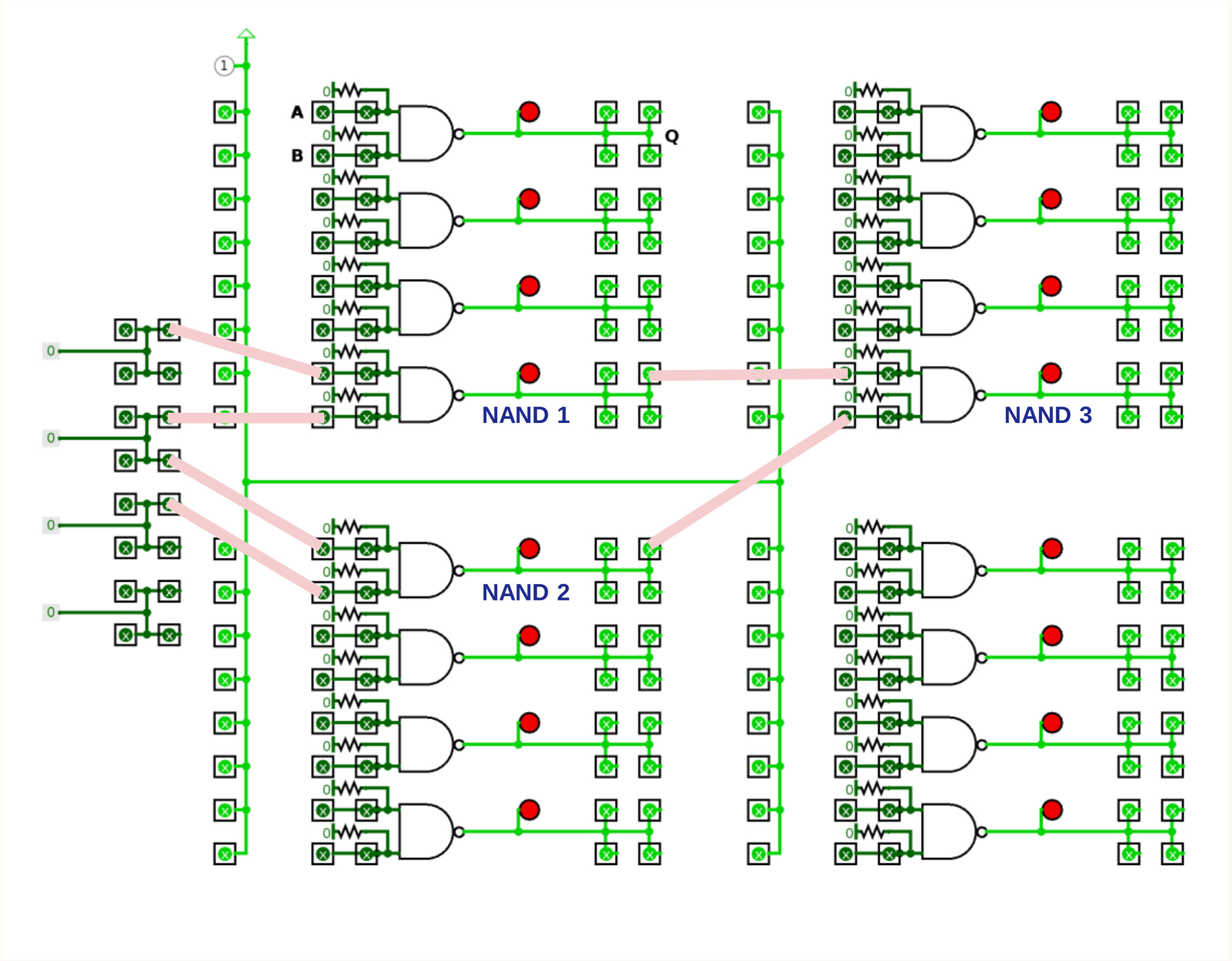
16 x

Constant 1
pin

NAND BOARDS

It's important to plan out how to transfer a circuit design onto a NAND board, as it can quickly become a confusing mess of wires.





NAND BOARDS

Please watch the NAND board introduction video before attending the first lab.

Well done on completing week 1 lectures!