

Lecture 2: Bits and Switches

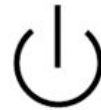
CS442: Great Insights in Computer Science
Michael L. Littman, Spring 2006

Preface

- Hillis mentions “function abstraction”, which is close enough to what I called “reduction”.
- He also mentions “universality”, which is an application of this idea: Any model of computing can simulate any other. There is only one kind of computing.
- Computers empower our minds: *Imagination Machines*.
- Catchy, I think.

The Lowly Bit

- Chemistry has its molecules.
- Physics has its strings.
- Computer science has its bits.
 - They are the smallest unit of information.
 - True (1, on)
 - False (0, off)



What's a Bit?

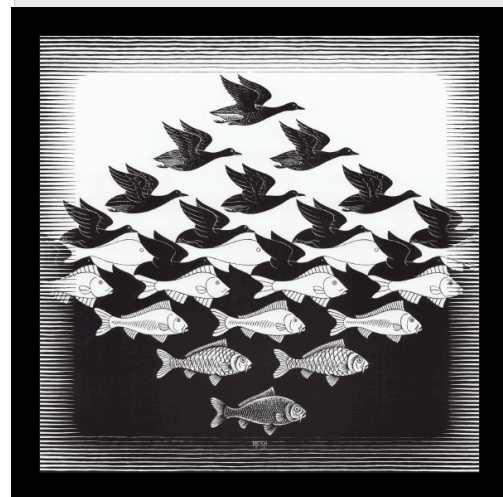
- Like “craptacular”, it is a portmanteau.
- It's a contraction of “binary digit”, used by Claude Shannon, attributed to John Tukey.
- In our decimal (base ten) number system, a digit can be any of the ten values 0,...,9.
- In the binary (base 2) number system, a digit can be any of the two values 0 or 1.

Why a Bit?

- Bits have the property of being...
 - simple: There's no smaller discrete distinction to be made.
 - powerful: sequences of bits can represent seemingly anything!

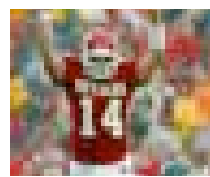
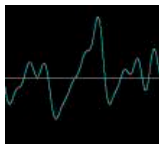
0/1, Black/White

- Escher created spectacular images with just a single distinction.



Bits Abound

- Nearly everything has gone digital.
- Digital sound: groups of bits represent the acoustic intensity at discrete time points.
- Digital images: groups of bits represent the color at each particular discrete image location: pixel.



But There Are Lots of Them

- iPod: 60Gb. 1Gb = one billion bytes, each of which is 8 bits.
- Format uses 128 Kbps (kilobits or 1000 bits per second of sound).
- So, 62,500 minutes of sound or 15,000 songs at 4 minutes per song.
- Screen: 320x240 pixels, each of which stores 1 byte each of R,G,B intensity (24 bits). That's 76,800 pixels and 1.8M bits.
- At 30 frames per sec., that's 55.3 million bits per second or 144 min. of (quiet) video.



Logical Variable

- Bits are for more than just storage, they can store information in the form of logical or Boolean variables.
- A *logical variable* is something that we can imagine as being either True or False.
 - `todayIsWednesday = False`
 - `itIsDarkOut = False`
 - `IAmWearingSocks = True`

And, Or, Not

- The most important logical operations are **and**, **or**, and **not**.
 - `x and y` is True only if both `x` and `y` are True.
 - `x or y` is True only if either `x` or `y` are True.
 - `not x` is True only if `x` is False.
- A lot like their English meanings, but unambiguous.

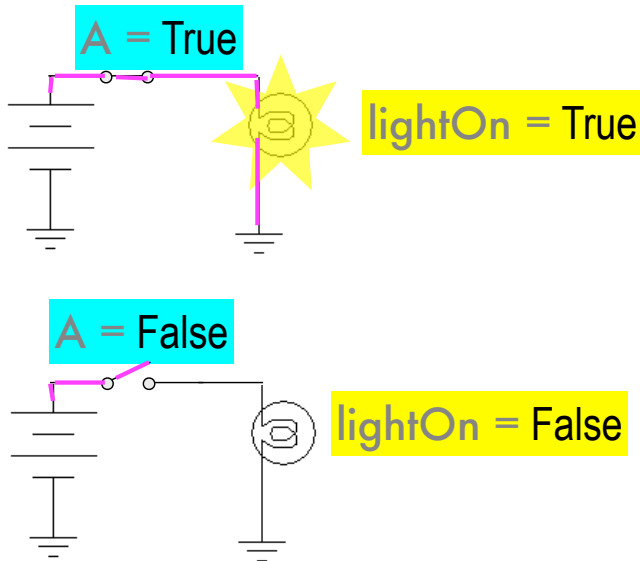
Logical Expressions

- We can combine logical variables and operators into more complex expressions:
 - `todayIsWednesday and IAmWearingSocks = False`
 - `itIsDarkOut or IAmWearingSocks = False`
 - `not itIsDarkOut = True`
 - `(not todayIsWednesday or itIsDarkOut) and (todayIsWednesday or not IAmWearingSocks) = ?`

Implementing Logic

- Clearly our brains can handle these sorts of expressions.
- But, can we automate them?
- Yes, obviously, but let's start with a really simple way to do it before we move on to fancier stuff.

Simple Circuit




Switch A is either on or off making the light either on or off: **lightOn=A**.

Symbols:

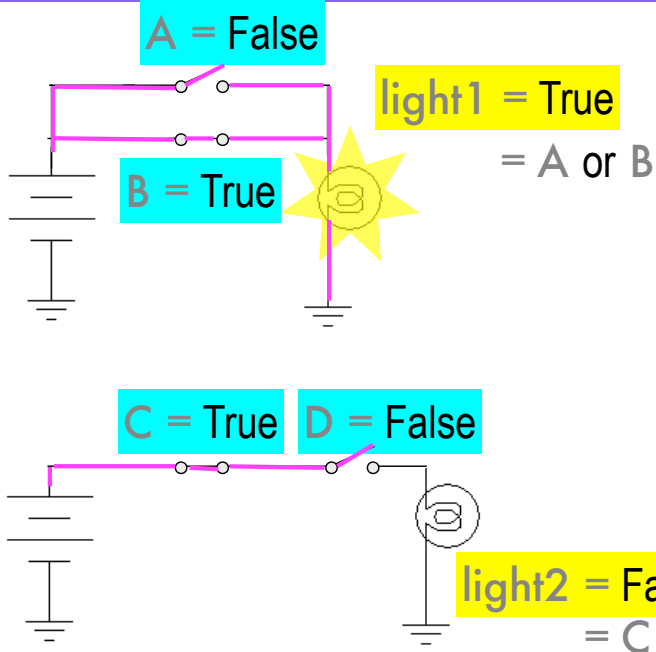
battery 

switch 

bulb 

ground (completes circuit) 

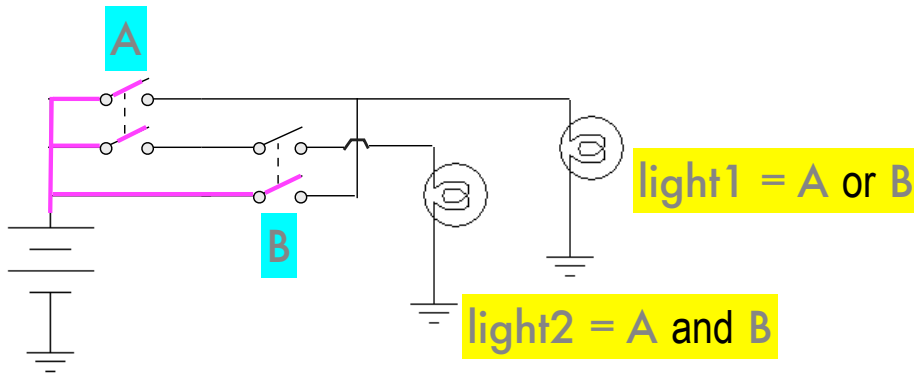
Multiple Switches



Switches A and B are wired in *parallel*: either will light the bulb.

Switches C and D are wired in *series*: both are needed to light the bulb.

Multiple Circuits



Special switches allow a single mechanical switch to control two circuits simultaneously.

miniNim

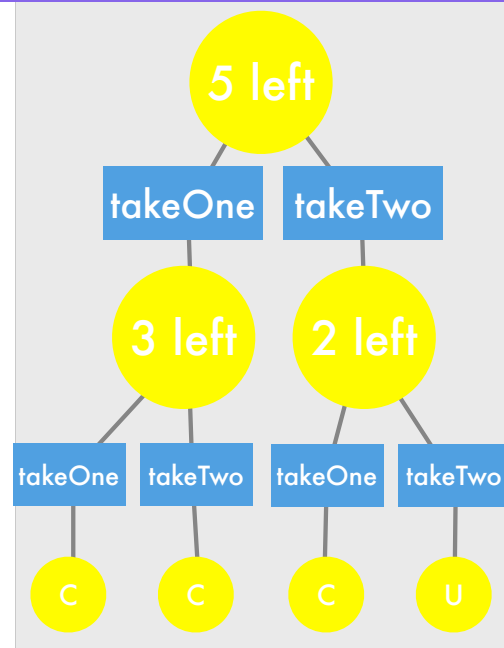
- There's a pile of objects, say 10.
- On her turn, a player can take away either one or two objects.
- Players alternate.
- The player to take the last object wins.



RRRRRRRRRR

Nim5Bot: Game Tree

- Let's start by considering the 5-object version.
- We'll design a strategy for the computer "C" to beat the user "U".



Backwards Induction

- There's a powerful idea here:
 - Many decision making problems can be solved optimally by reasoning backwards from the end of the game.
 - We know how to win from 1 or 2.
 - We use this to win from 4 or 5.
 - We use this to win from 7 or 8.
- Chess: removed pieces don't return.

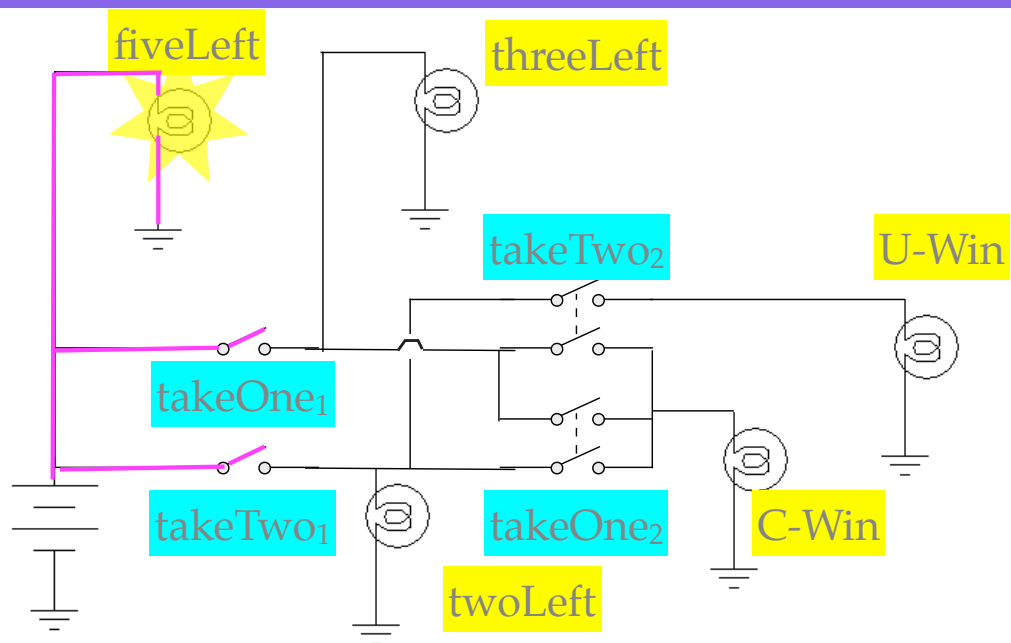
Further Considerations

- To win miniNim: if possible, remove pieces to leave opponent with a multiple of 3.
- Why does it work? We win if opponent has 3; if opponent has a multiple of 3, can leave her with next smaller multiple of 3.
- What if goal is to *not* take the last object?
- What if we can take 1, 2, or 3 objects per round? 2 or 3? 1 or 3? Is there a general rule?

Complete Nim5 Logic

- `fiveLeft = True`
- `threeLeft = takeOne1`
- `twoLeft = takeTwo1`
- `C-Win = (threeLeft and (takeOne2 or takeTwo2)) or (twoLeft and takeOne2)`
- `U-Win = twoLeft and takeTwo2`

Nim5 Circuit



Let's Play!

- <http://www.cs.rutgers.edu/~mlittman/courses/cs442-06/python/>
- nim5.py
- nim10.py

Complete Nim10 Logic

```
tenLeft[1] = True
eightLeft[2] = takeOneSwitch[1]
sixLeft[2] = takeTwoSwitch[1]
sixLeft[3] = eightLeft[2] and takeOneSwitch[2]
fourLeft[3] = eightLeft[2] and takeTwoSwitch[2]
moveTwo = takeOneSwitch[2] or takeTwoSwitch[2]
threeLeft[3] = sixLeft[2] and moveTwo
moveThree = takeOneSwitch[3] or takeTwoSwitch[3]
threeLeft[4] = sixLeft[3] and moveThree
twoLeft[4] = fourLeft[3] and takeOneSwitch[3]
winFourA = threeLeft[3] and moveThree
winFourB = fourLeft[3] and takeTwoSwitch[3]
IWin[4] = winFourA or winFourB
moveFour = takeOneSwitch[4] or takeTwoSwitch[4]
winFiveA = twoLeft[4] and takeOneSwitch[4]
winFiveB = threeLeft[4] and moveFour
IWin[5] = winFiveA or winFiveB
youWin[5] = twoLeft[4] and takeTwoSwitch[4]
```

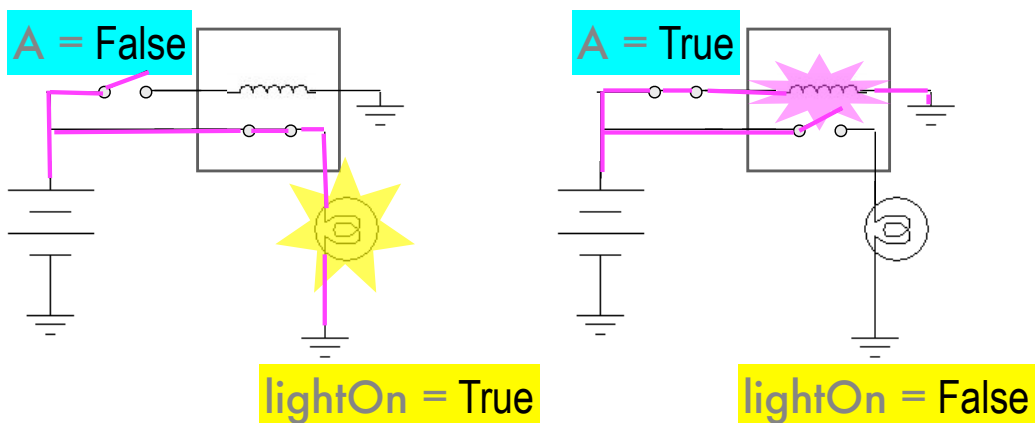
What a Headache!

- I tried to create the circuit diagram for Nim10 and couldn't do it. Why?
 - Since some switches are used in multiple places, needs more than a double-throw switch.
 - Since values are reused, hard to keep track of different circuits.
 - Appears to need a separate circuit for each output: gets too complex too fast.

Need a New Approach

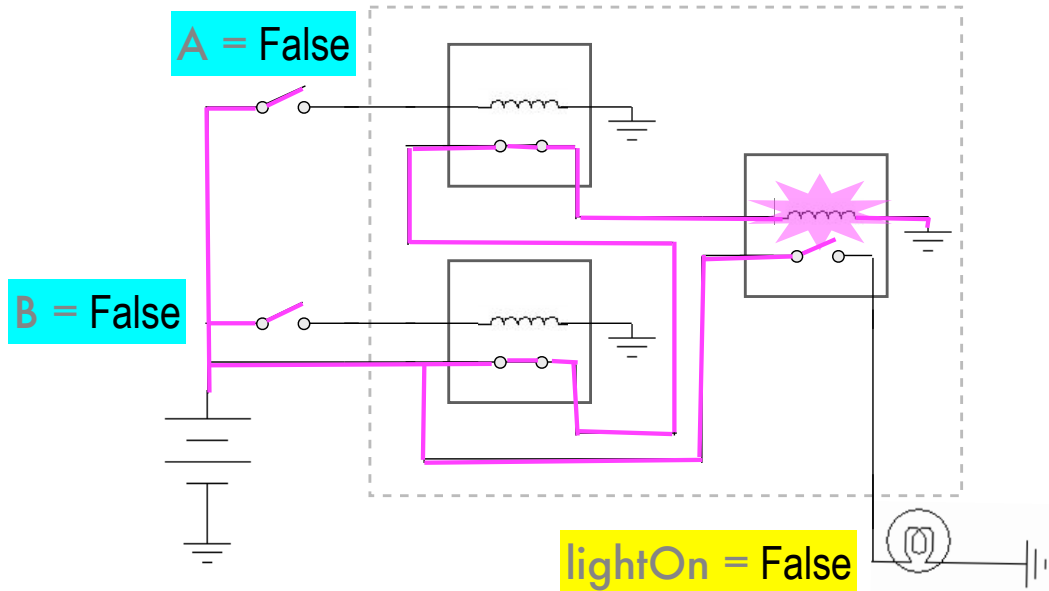
- Let's consider an alternate way of defining "and" and "or".
- Makes simple things *much* more complex.
- Makes complex things *much* simpler!
- That's a tradeoff we can deal with.

Switch Switcher

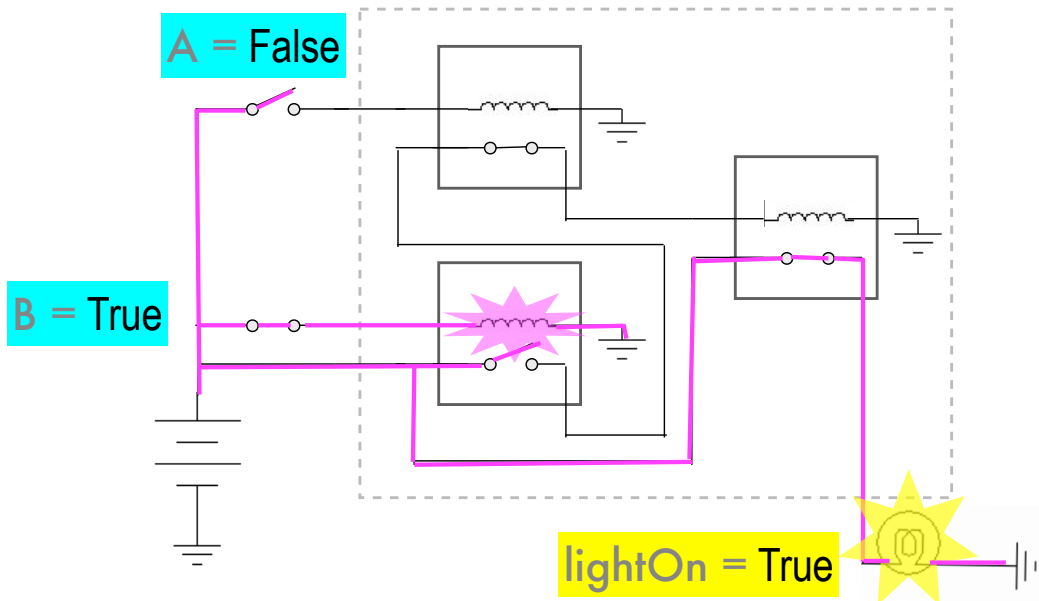


Before, we used switches to control the flow of current; now, we will use current (via electromagnetism) to control switches (which control the flow of current)!

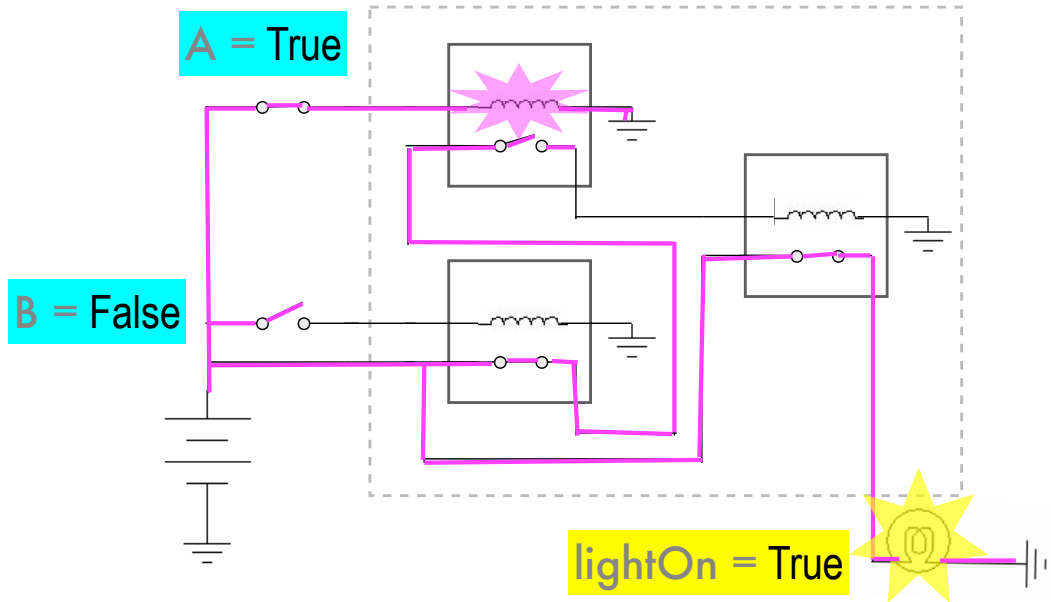
Relay Circuit: A=F, B=F



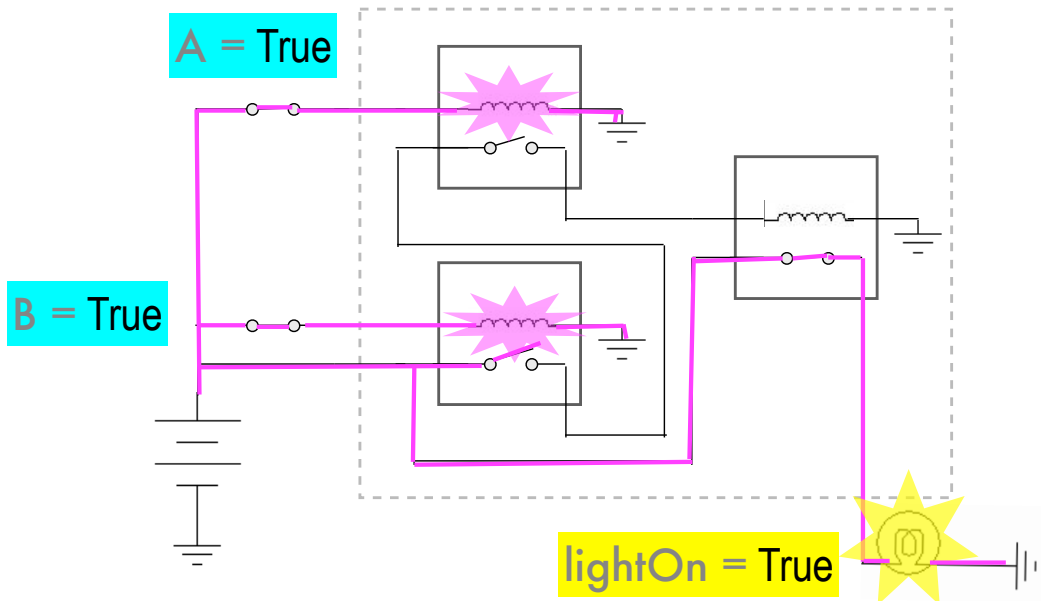
Relay Circuit: A=F, B=T



Relay Circuit: A=T, B=F



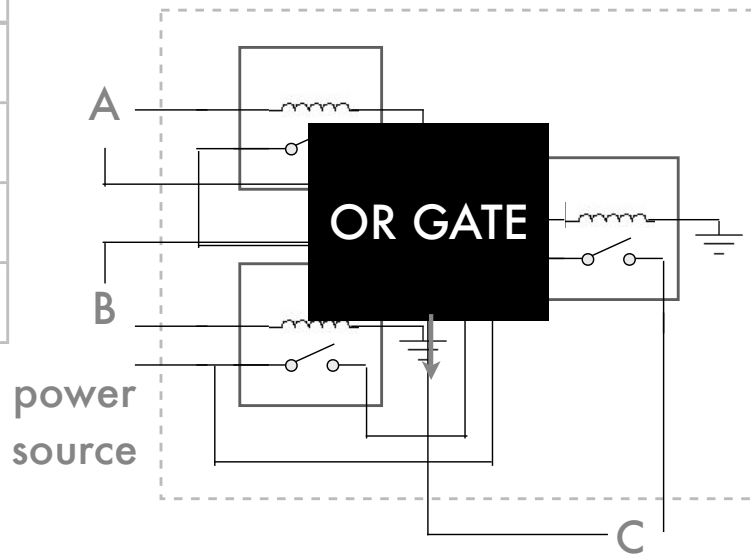
Relay Circuit: A=T, B=T



What Is This Thing?

A	B	C
False	False	False
False	True	True
True	False	True
True	True	True

It's an "or gate":
used to compute
the "or" of two
inputs.



And One For "Not"

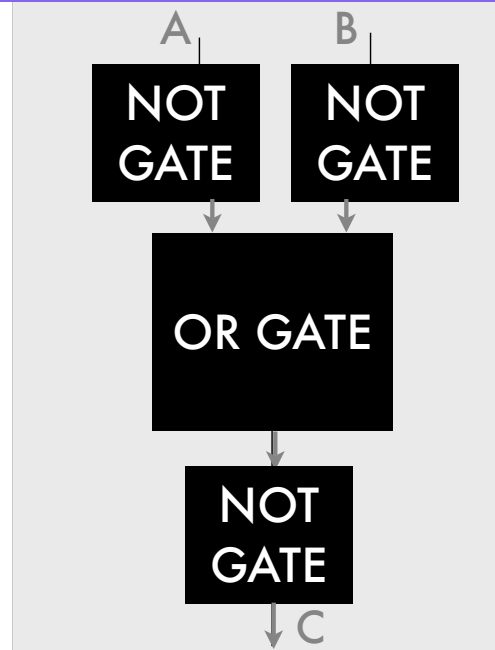
- We saw several slides ago that a single relay "inverts" its input signal, turning a True to a False and vice versa.
- These output summaries are known as "truth tables".

A	B
False	True
True	False



Abstraction: The Black Box

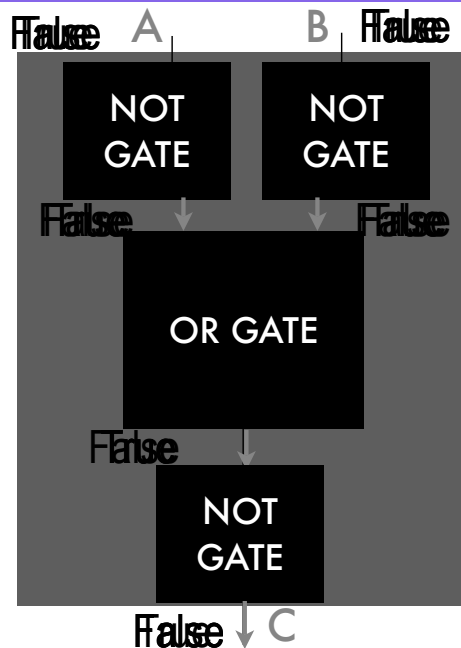
- Our new relayed-based “or” and “not” gates take current, not switches, as input.
- As a result, they are easier to chain together.
- The original switch-based scheme did not support this kind of modularity.



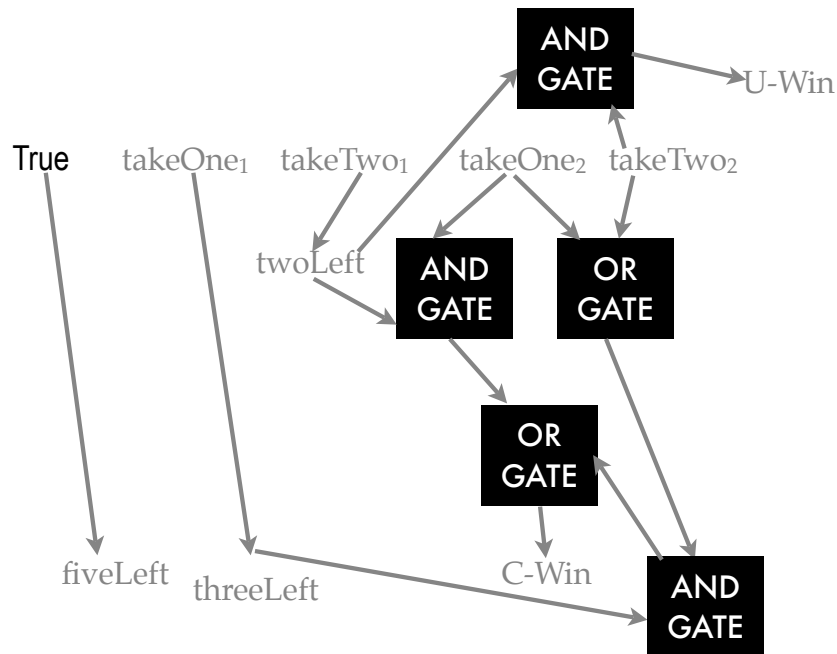
A Third Black Box

A	B	C
False	False	False
False	True	False
True	False	False
True	True	True

It's an “and gate”:
a black box built
out of other black
boxes!



Simplified Nim5 Circuit



Next Time

- We'll talk about some other ways to make "or" and "not" gates.
- We'll use gates to create a bunch of other black boxes, building complexity as we go.
- Read Hillis, Remaining Sections of Chapter 1.