More ***bits*** of CS

Too many bits? Compress!

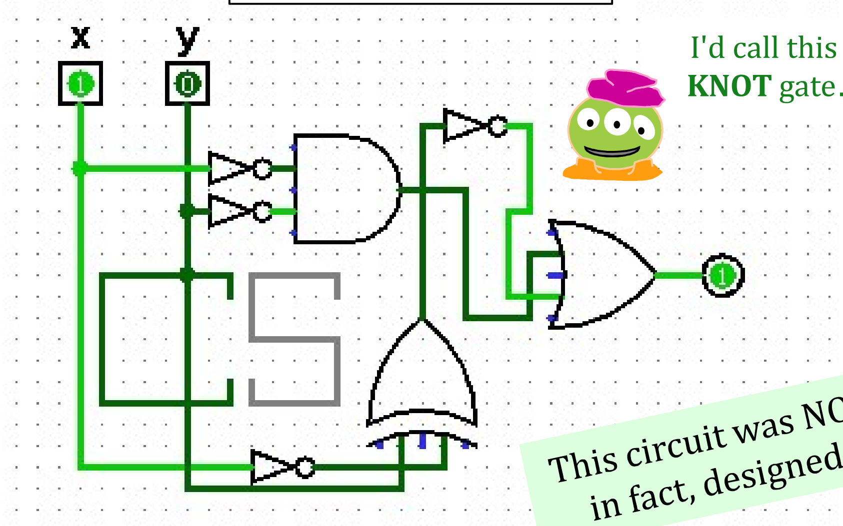
Below binary: ***physical circuits***

Jotto Corner

Olivia		Deanna	
AM guess	my guess	HS guess	my guess
camel: 4	diner: 1	**human: 1**	diner: 2
cable: 3	savvy: 0	**slaps: 2**	savvy: 1
?????: ?	**flock:** ?	?????: ? diner 1	**flock:** ? !
?	?	?	?

words remaining

Circuit design, part 1



I'd call this a **KNOT** gate...

This circuit was NOT, in fact, designed!

P/F vs T/F

Hw #4 due Mon. **10/7**

pr0 *(reading)* A bug and a crash!

pr1 *(lab)* binary ~ decimal

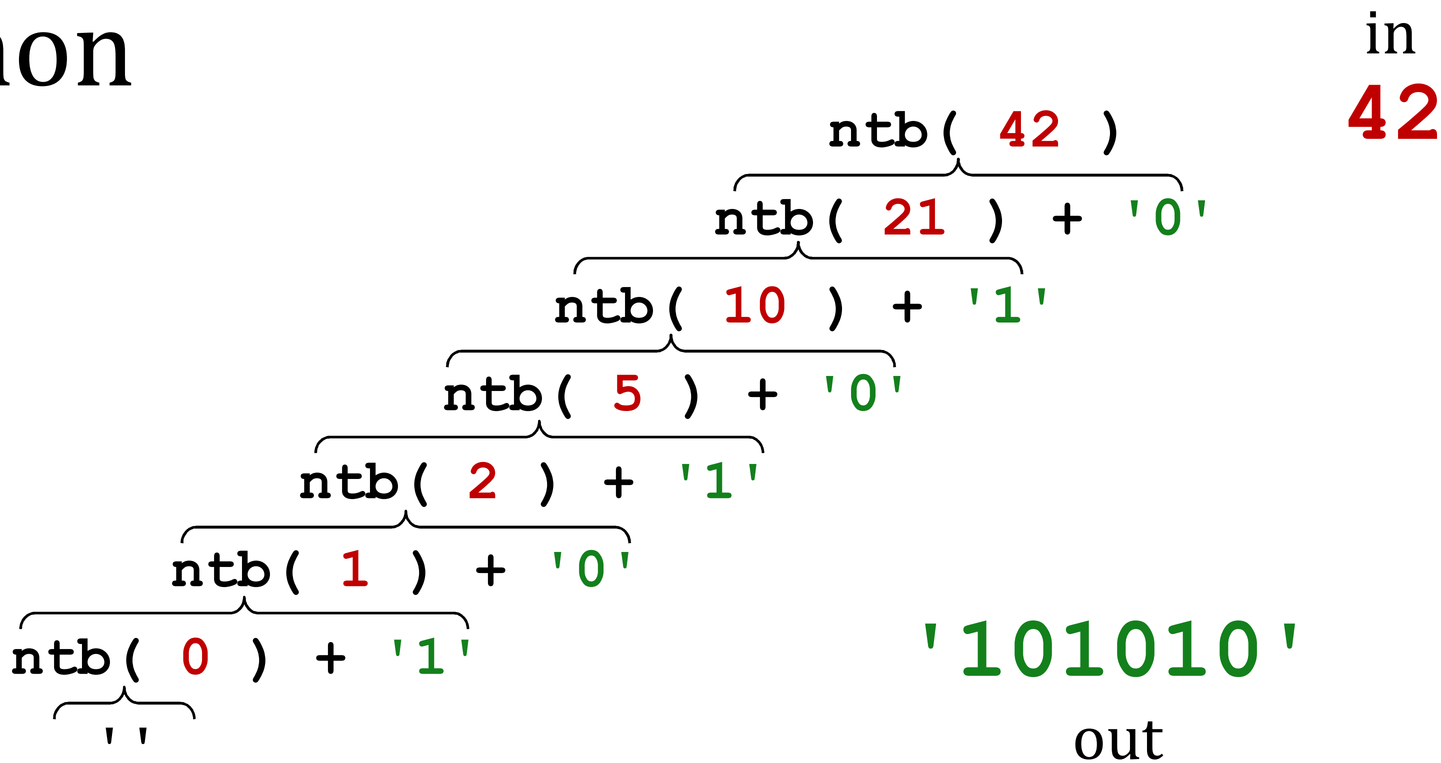
pr2 conversion + compression

extra image processing...

Lots of tutoring hrs - join in... !

Office hours == Fri. aft.

Python

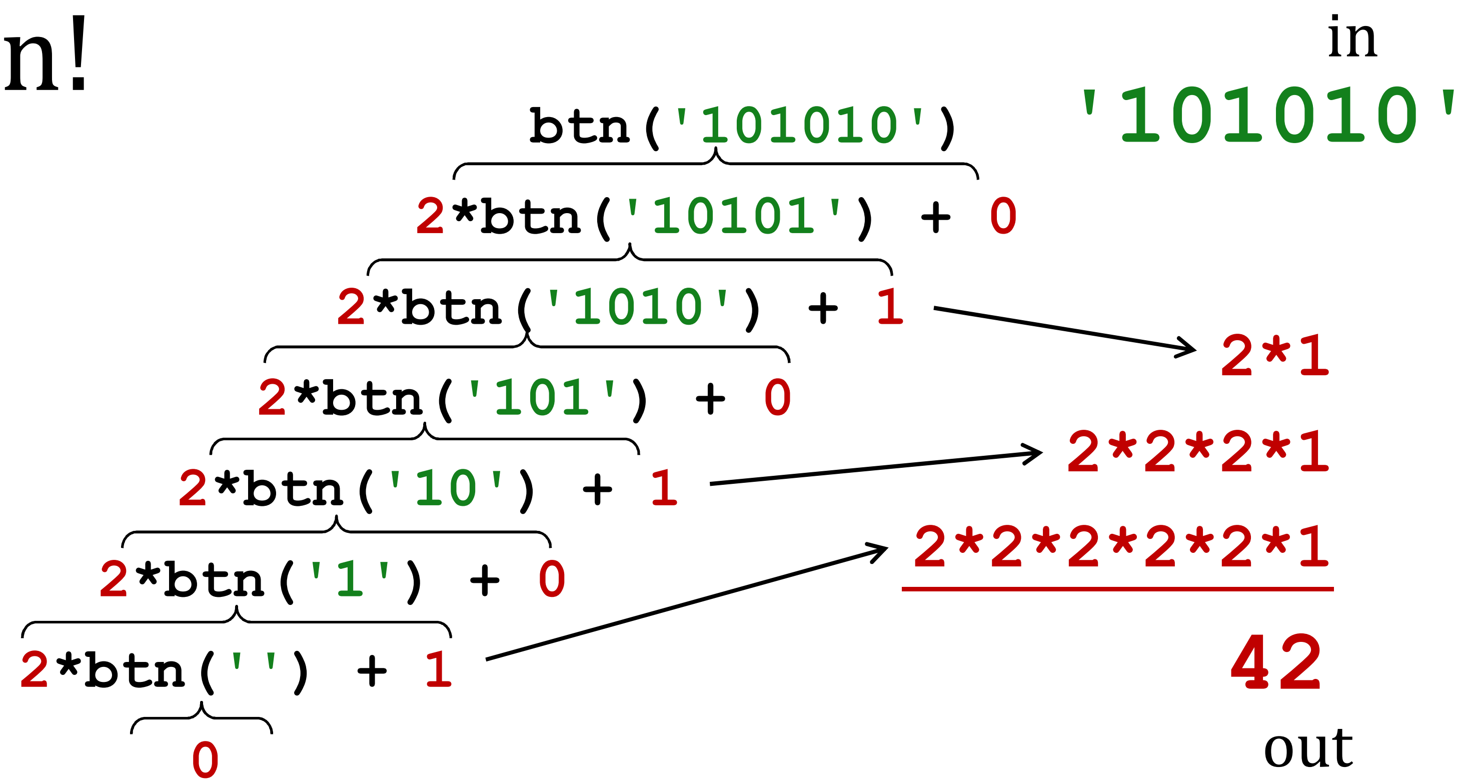


fleek-ified!

```
def numToBin( N      ):
    """ converts a decimal int to a binary string
    """
    if N==0:      return ''
    else:         return numToBin( N//2 ) + str(N%2)
```

What if you wanted base-3 output?! ***base-B output?***

Python!



fleek-ified!

```
def binToNum( S     ):
    """ converts a binary string to a decimal int
    """
    if S=='': return 0
    else:     return 2*binToNum(S[:-1]) + int( S[-1] )
```

What if you wanted base-3 input?! ***base-B input?***

saves the need for another **`if`**

Bits & Binary

value	representation
	64 32 16 8 4 2 1
0	0
1	1
2	10
5	101
10	1010
15	1111
16	10000
21	10101
42	101010
127	1111111
	64 32 16 8 4 2 1

shifting bits left << 1

What's **101000** ?

shifting bits right >> 1

What's **1000** ?

maximum value of 4 bits?

maximum value of 7 bits?

maximum value of <u>N</u> bits?!

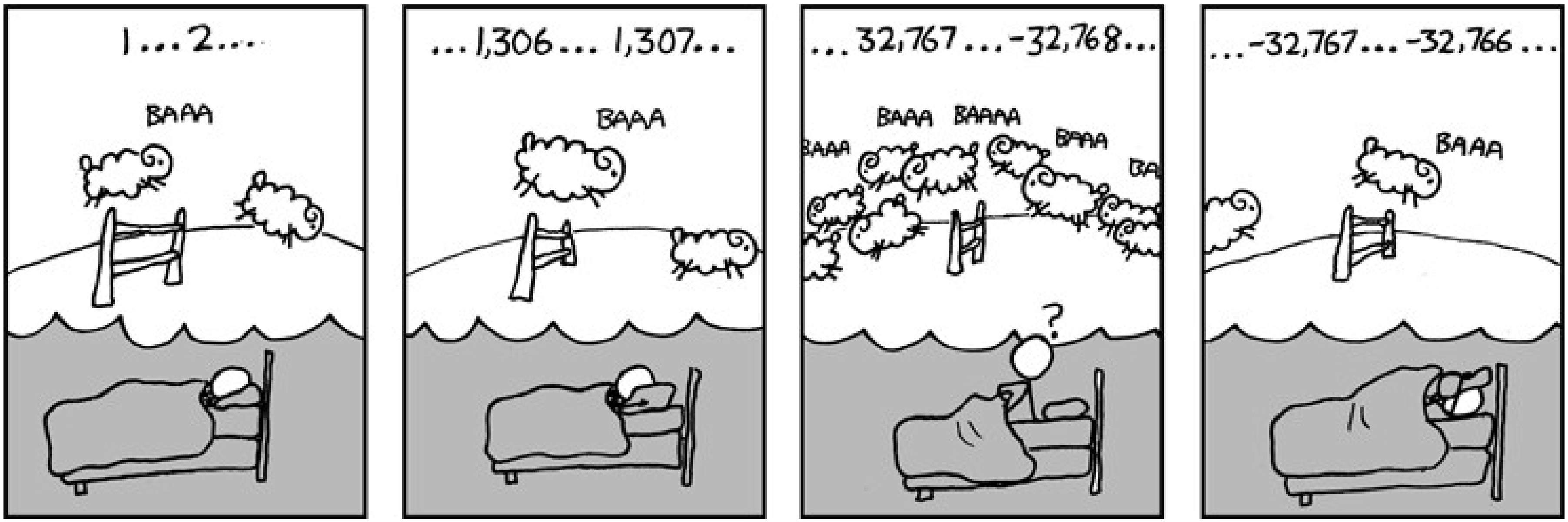
How high can we count...?

I can see some patterns here –
even *with one eye closed!*

with	**1 bit**	1	1
	2 bits	11	3
	3 bits	111	7
	4 bits	1111	15
	7 bits	1111111	127
	8 bits	11111111	255
	N bits		

15 bits ***31 bits***

Counting sheep, xkcd *style...*



How many bits?

Ariane 5

This week's reading: ***bits can be vital***

`IndexError` **`TypeError`** **`HumanError`**

16 bits ← 64 bits

version 4 version 5

How high can we count… *in 2015?*

List of most viewed YouTube videos

From Wikipedia, the free encyclopedia

Top videos

indicates a video that is not a music video.

Rank	Video name[A]	Uploader / artist	Views (as of September 29, 2015)	Upload date	Notes
1.					
2.	"Baby"[4]	Justin Bieber featuring Ludacris	1,216,729,955	February 19, 2010	[C]
3.	"Blank Space"[5]	Taylor Swift	1,173,509,710	November 10, 2014	[D]

How high can we count... *in 2015!*

List of most viewed YouTube videos

From Wikipedia, the free encyclopedia

This **list of most viewed YouTube videos** consists of the 30 most viewed videos of all time as derived from YouTube charts.[1] Videos that YouTube suspects have had their view counts manipulated[2] are not included in this list. View counts are based on the YouTube website; many of the videos are music videos that play through YouTube's partner site, Vevo, and YouTube view counts will lag those of Vevo by a few days.[1]

As of September 2015, nine music videos have received over 1 billion views, with the top video, "Gangnam Style", exceeding 2 billion views.

Psy's "Gangnam Style" is the most watched video on YouTube as of September 2015, with over 2.4 billion views.

only briefly, of course...

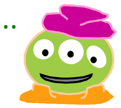
Top videos

indicates a video that is not a music video.

Rank	Video name[A]	Uploader / artist	Views (as of September 29, 2015)	Upload date	Notes
1.	"Gangnam Style"[3]	Psy	2,421,271,749	July 15, 2012	[B]
2.	"Baby"[4]	Justin Bieber featuring Ludacris	1,216,729,955	February 19, 2010	[C]
3.	"Blank Space"[5]	Taylor Swift	1,173,509,710	November 10, 2014	[D]

Other overflow errors...

Less worrisome, perhaps...



THE WALL STREET JOURNAL. ARTS & ENTERTAINMENT

9:19 am ET
Dec 3, 2014 MUSIC

Psy's 'Gangnam Style' Has Forced YouTube to 'Upgrade' Systems

Gangnam Style Video Overflows YouTube Counter

By Rick Regan (Published December 3rd, 2014)

On Monday, Psy's Gangnam Style video exceeded the limit of YouTube's view counter; this is what Google had to say (hat tip: Digg):

> *"We never thought a video would be watched in numbers greater than a 32-bit integer (=2,147,483,647 views)..."*

The "sign bit" has flipped to one. Thus, the number has become *negative*... !

PSY - GANGNAM STYLE (강남스타일) M/V

officialpsy

Subscribe

-2142584554

Add to Share ••• More

Hw4: *images are just bits, too*!

hw4pr3 ***(extra)***

<u>old</u> pixel at 42,42 has

red = 1 (out of 255)
green = 36 (out of 255)
blue = 117 (out of 255)

<u>new</u> pixel at 42,42 has

guesses as to what this transformation was?

how many <u>bits</u> represent each color channel?

Hw4: *images are just bits, too*!

hw4pr3 ***(extra)***

<u>old</u> pixel at 42,42 has

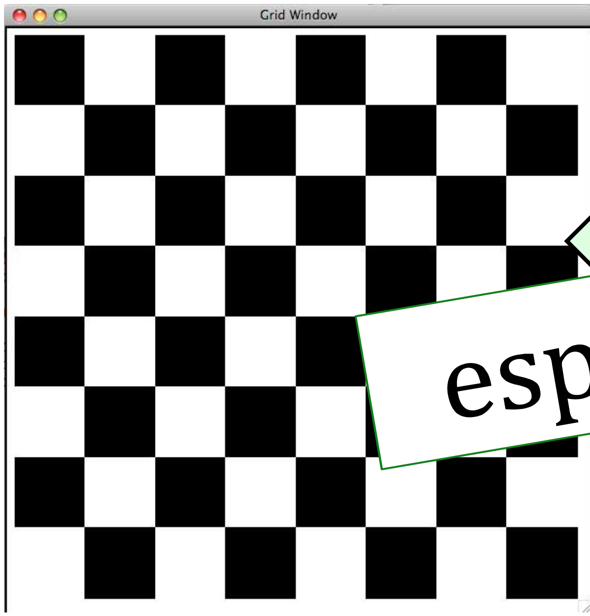
red = 1 (out of 255)
green = 36 (out of 255)
blue = 117 (out of 255)

<u>new</u> pixel at 42,42 has

red = 254 (out of 255)
green = 219 (out of 255)
blue = 138 (out of 255)

how many <u>bits</u> represent each color channel?

Hw4: *images are just bits, too*!



Binary Image

especially ***binary*** images

```
10101010
01010101
1010
01010
01010101
10101010
01010101
```

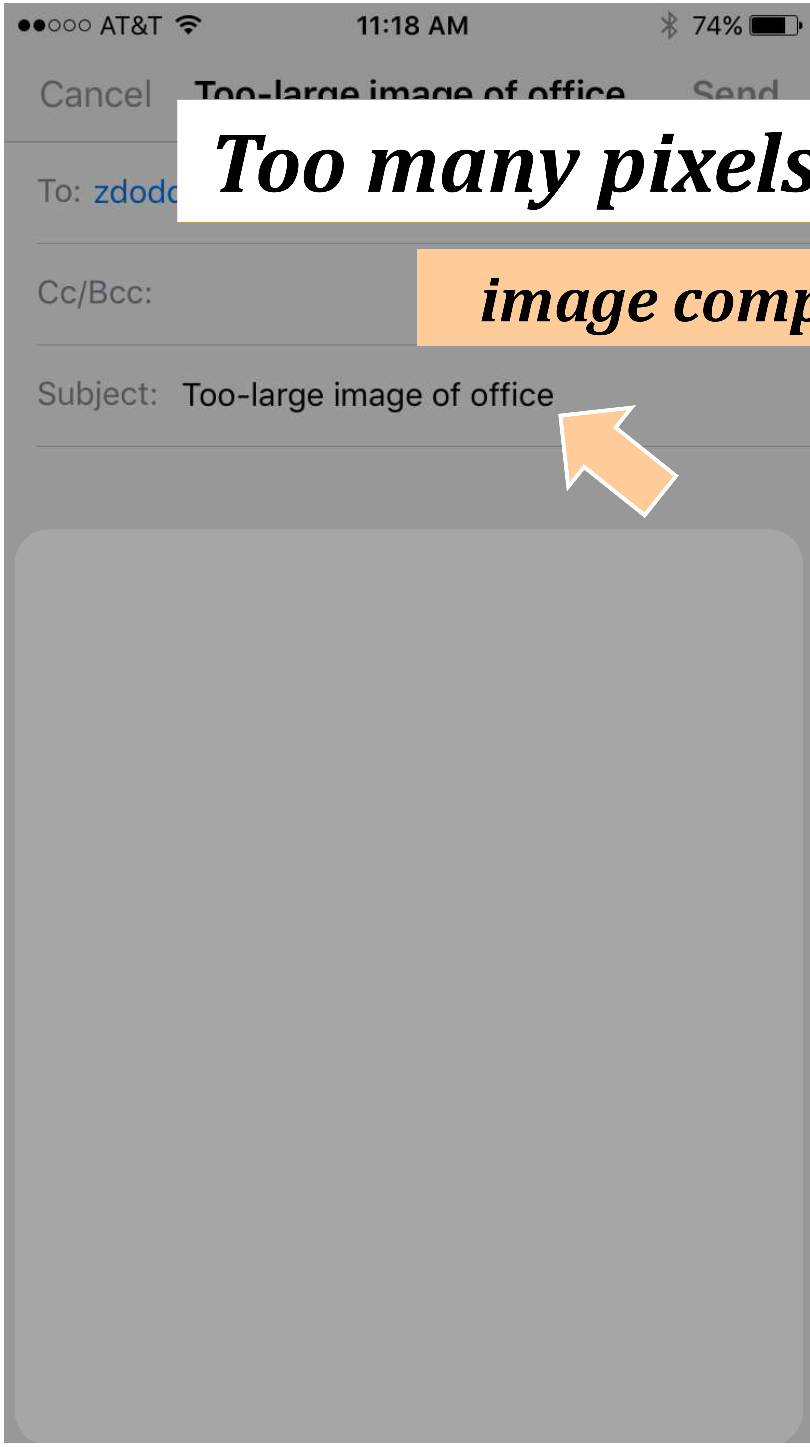
Encoding as raw bits

one big string of 64 characters

"1010101001010101101010100101010110101010010101011010101001010101"

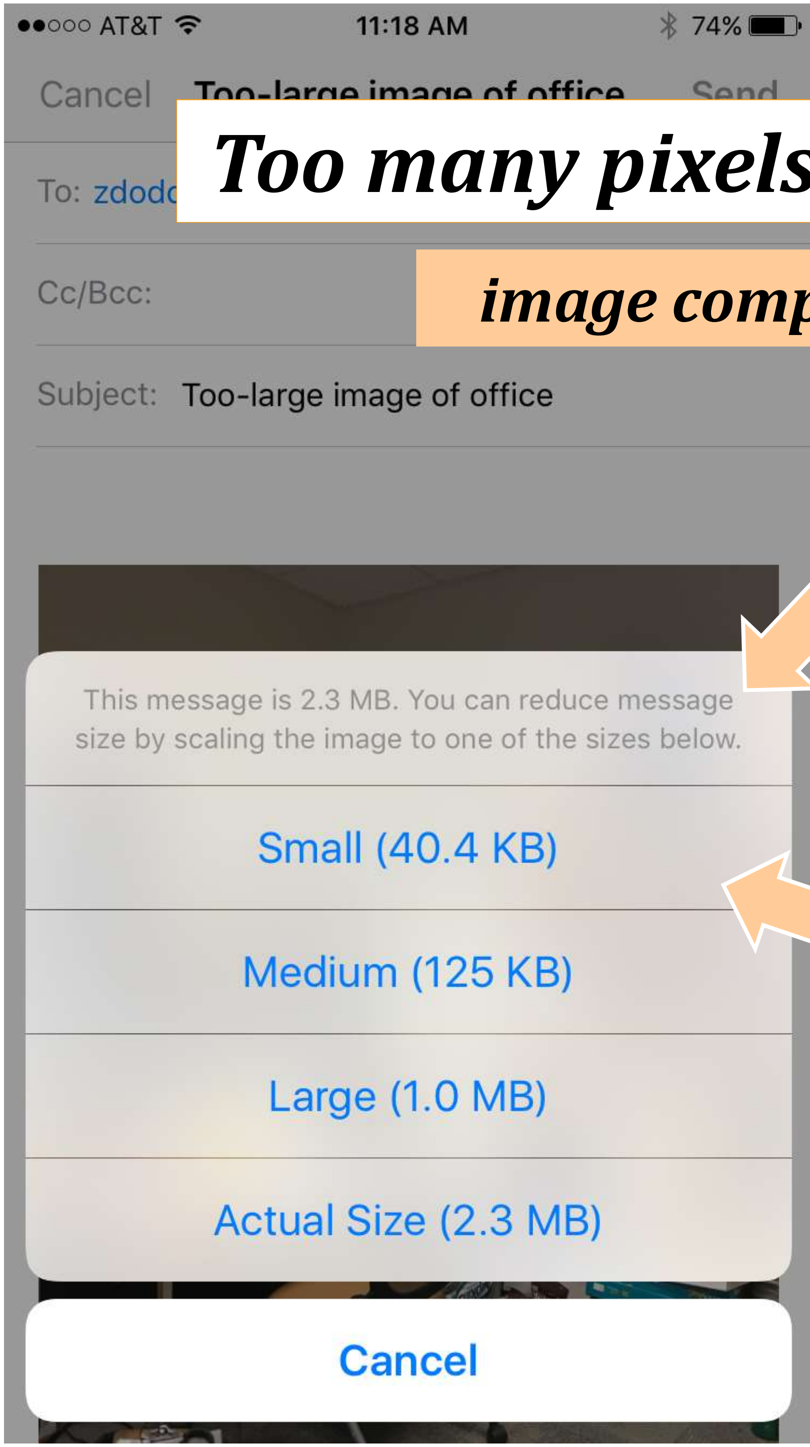
Too many pixels... too little time + space!

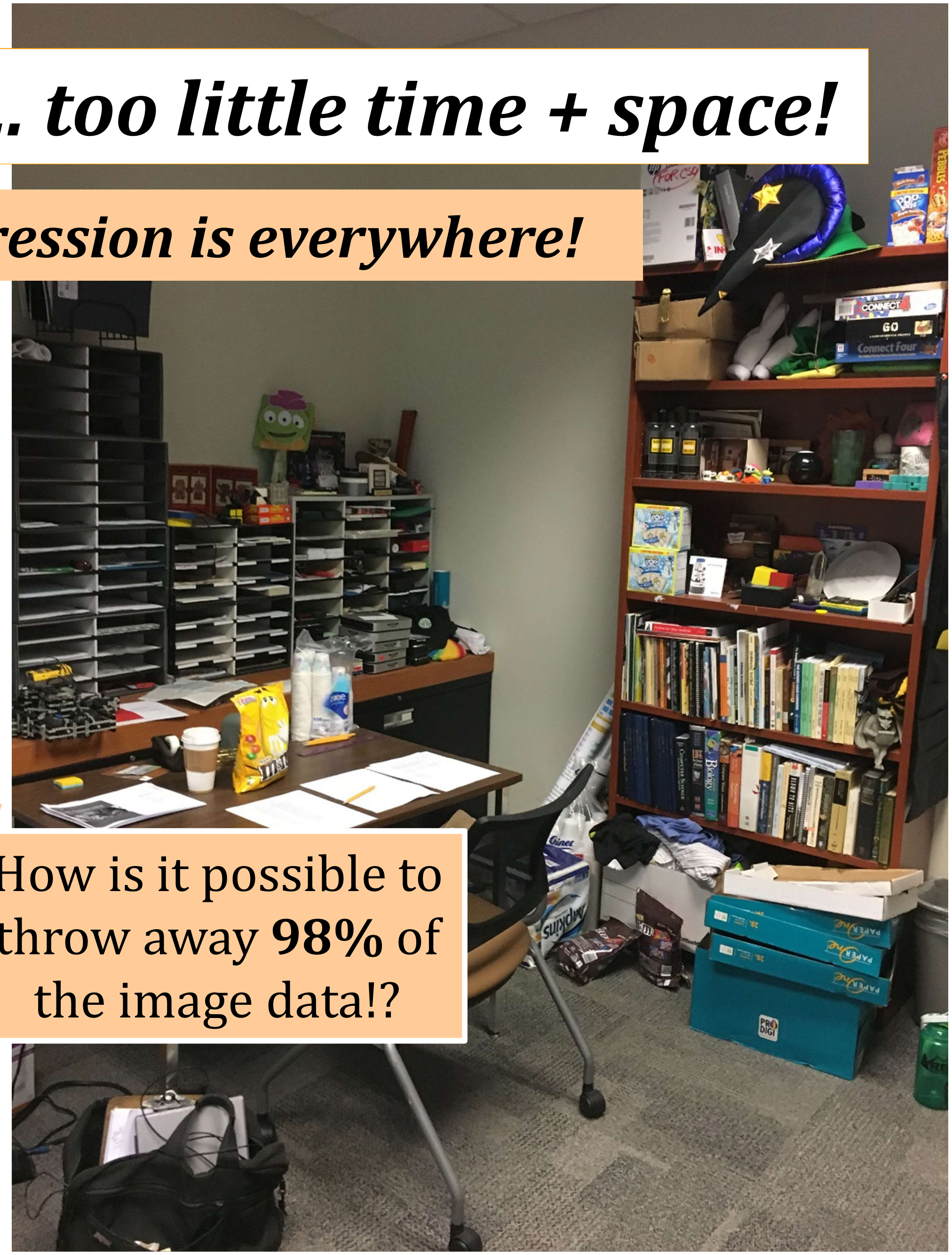
image compression is everywhere!



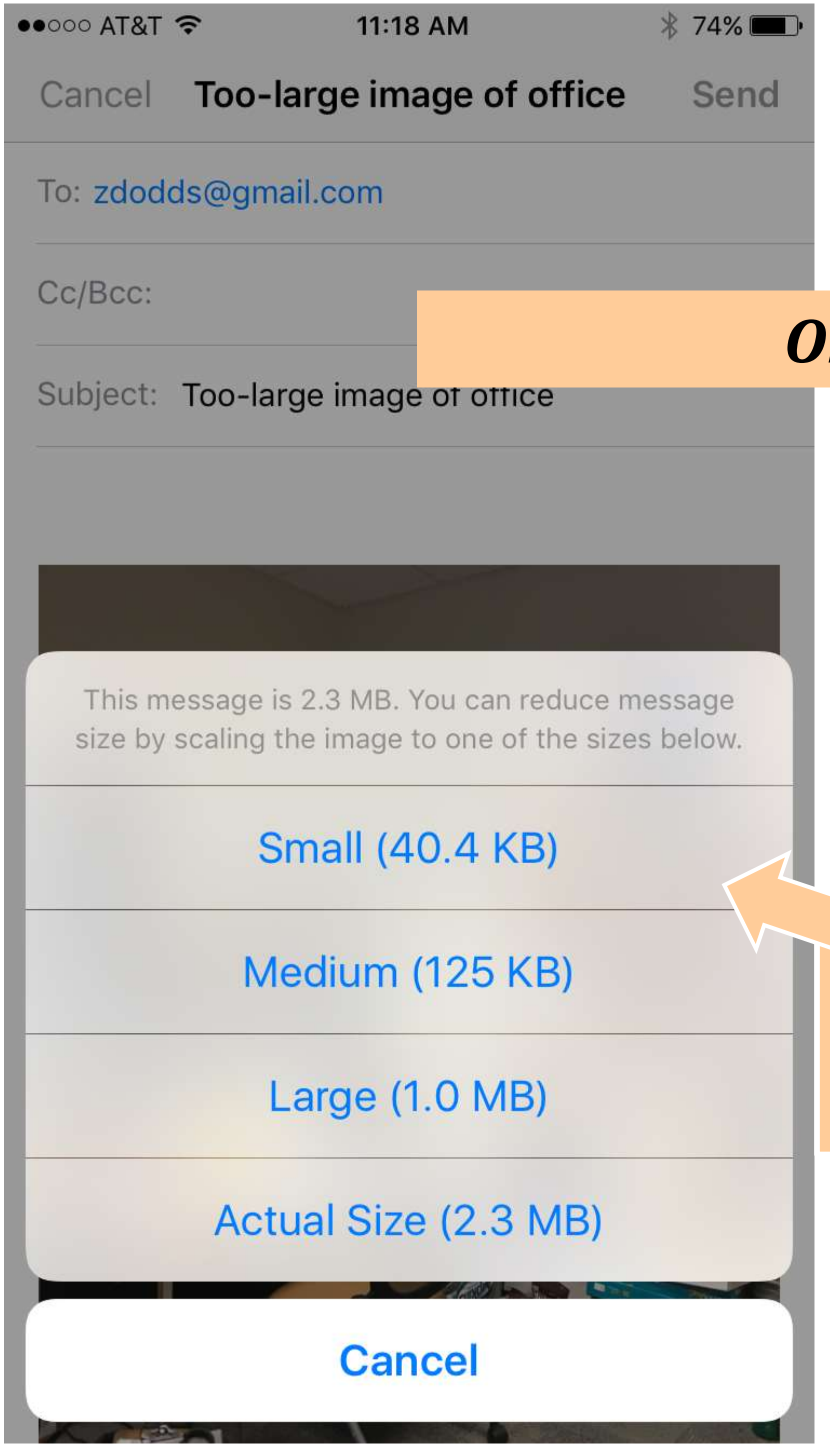
Too many pixels... too little time + space!

image compression is everywhere!

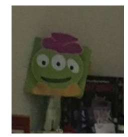




How is it possible to throw away **98%** of the image data!?



One solution!

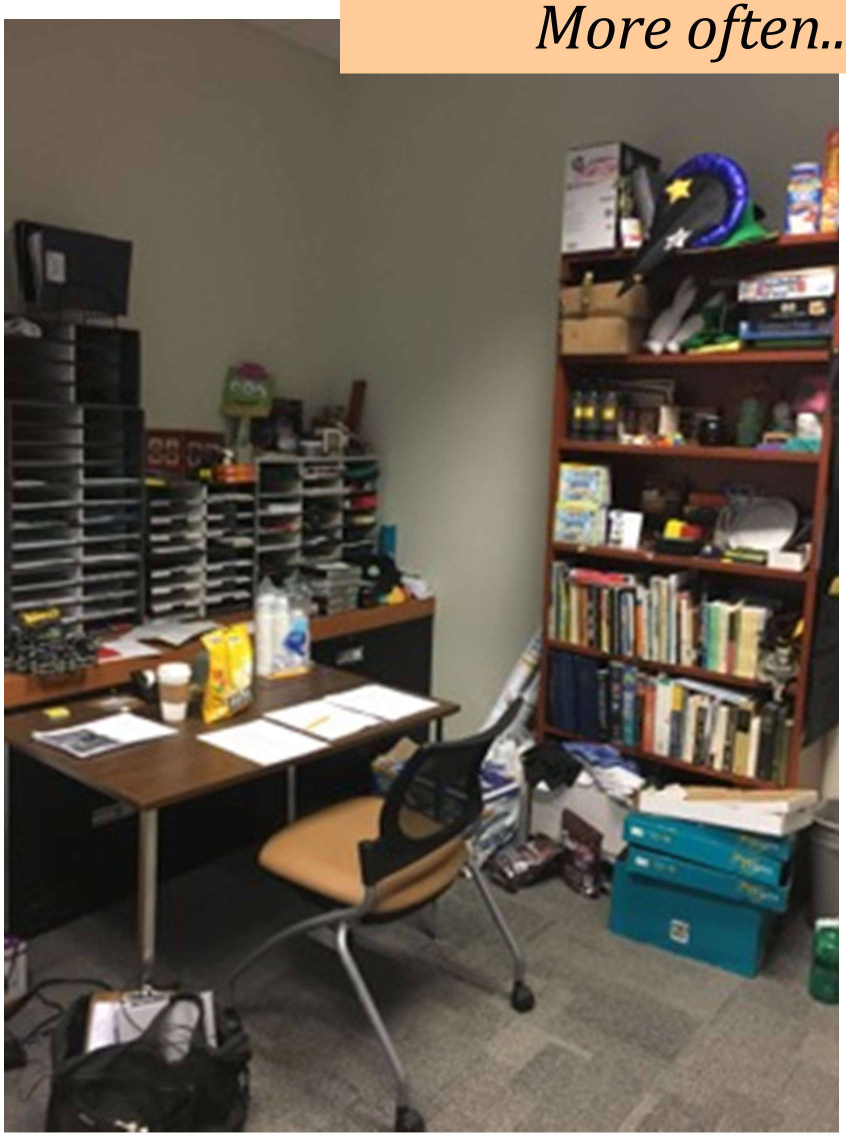


We throw away 98% of the image ***area***!

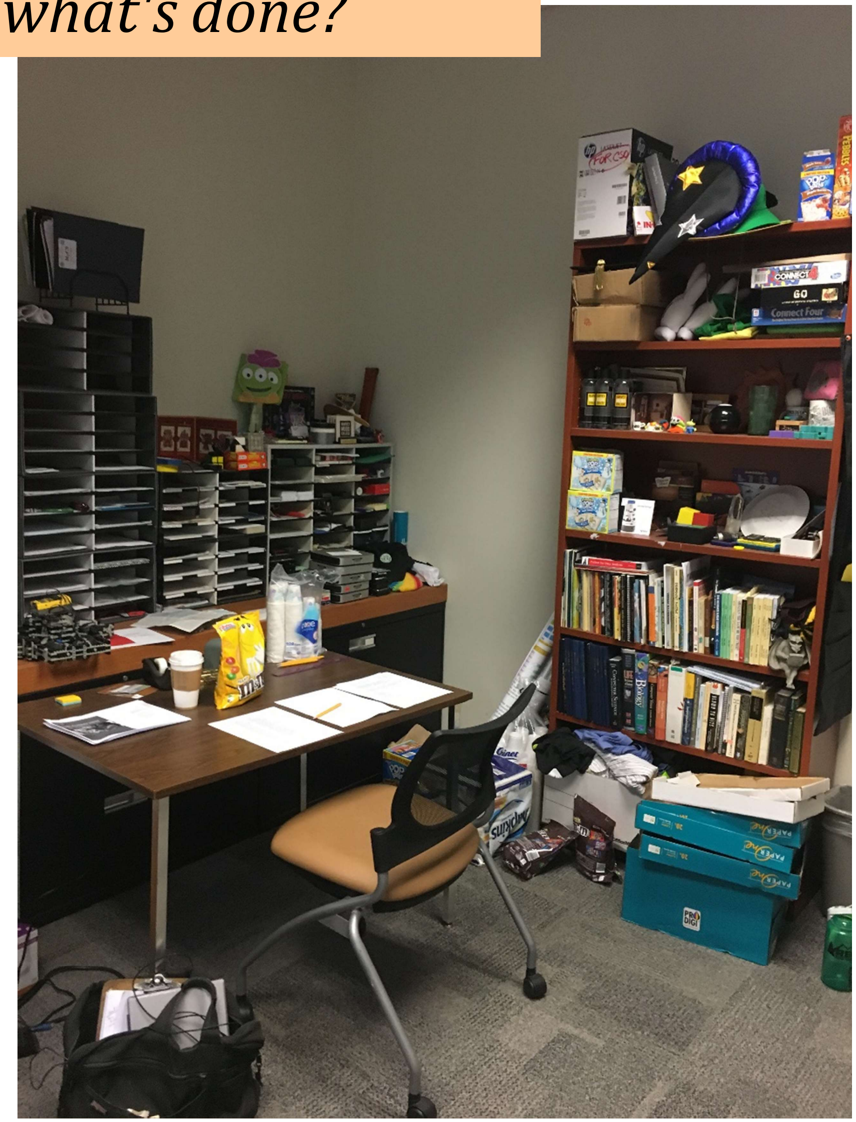
Looks like the right 2% to keep!

How is it possible to throw away **98%** of the image data!?

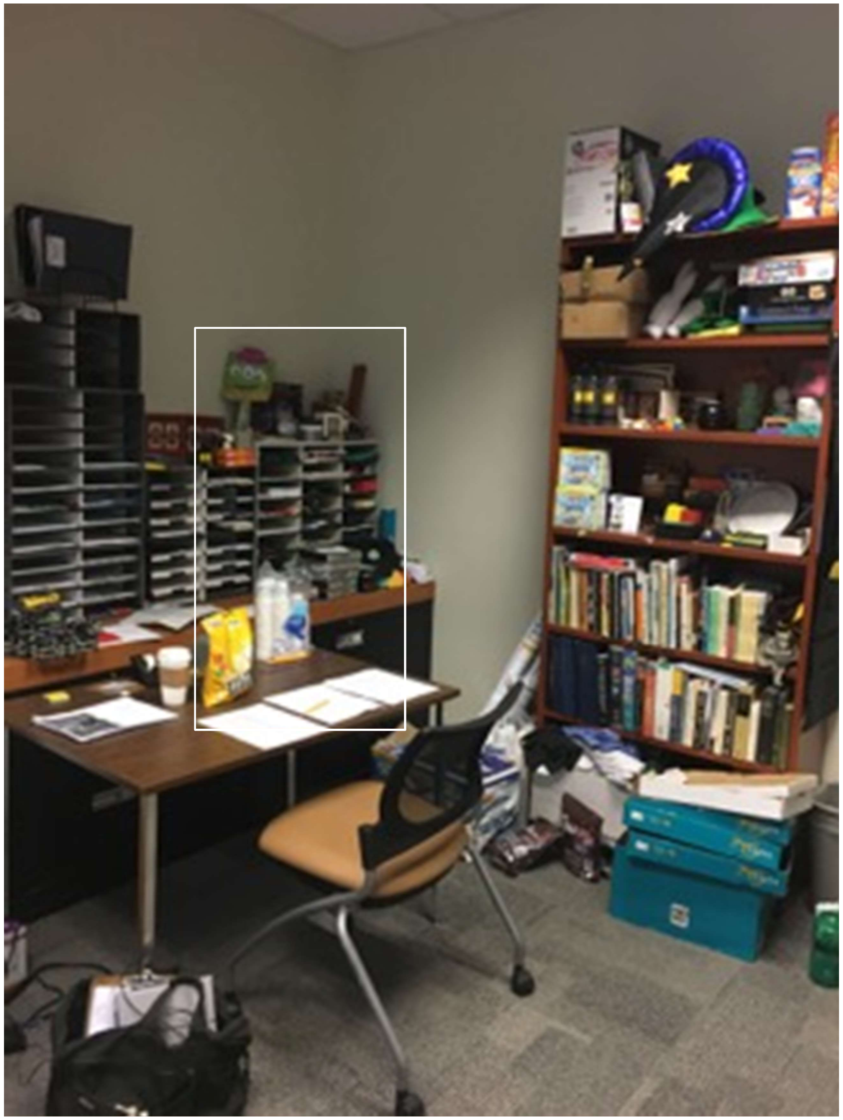
More often... what's done?



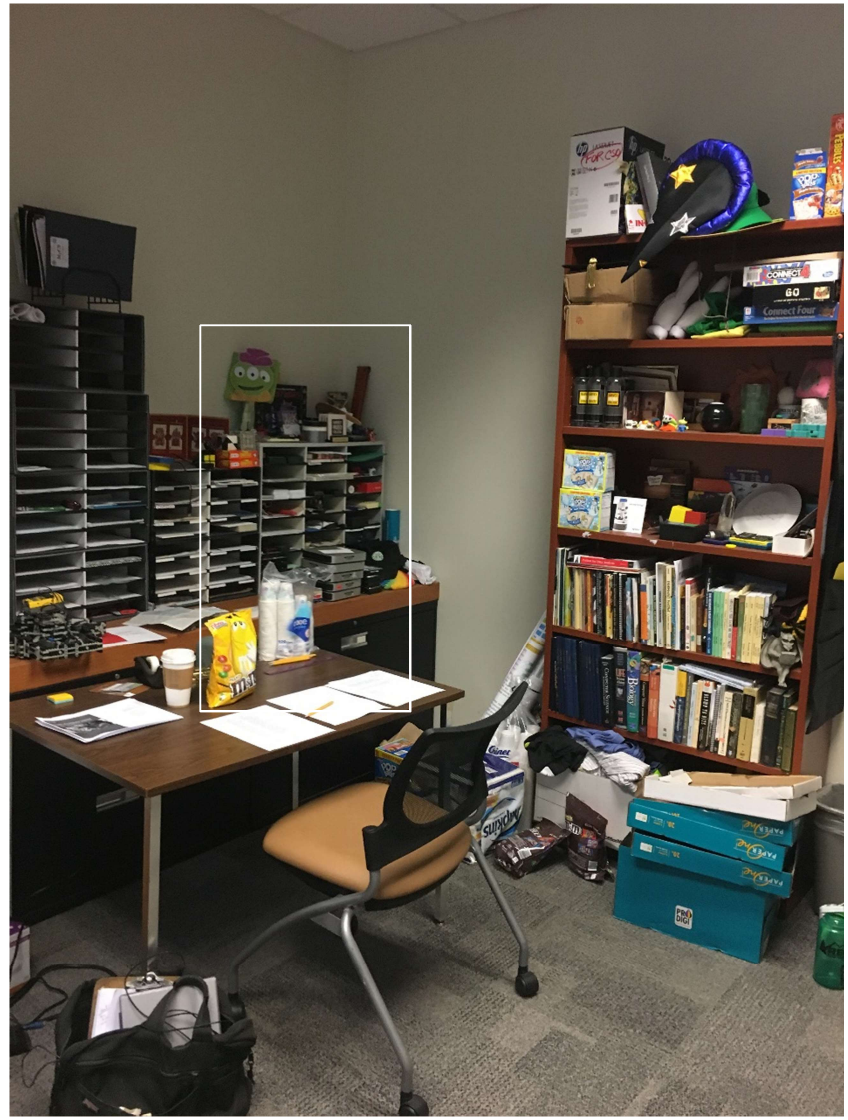
compressed *to 40kb*



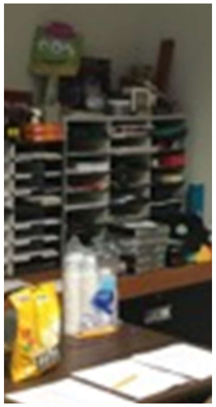
original: *2.3mb*

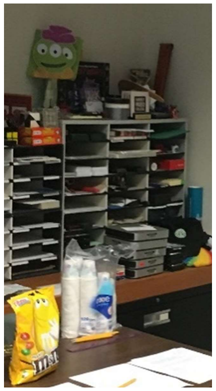


***compressed** to 40kb*



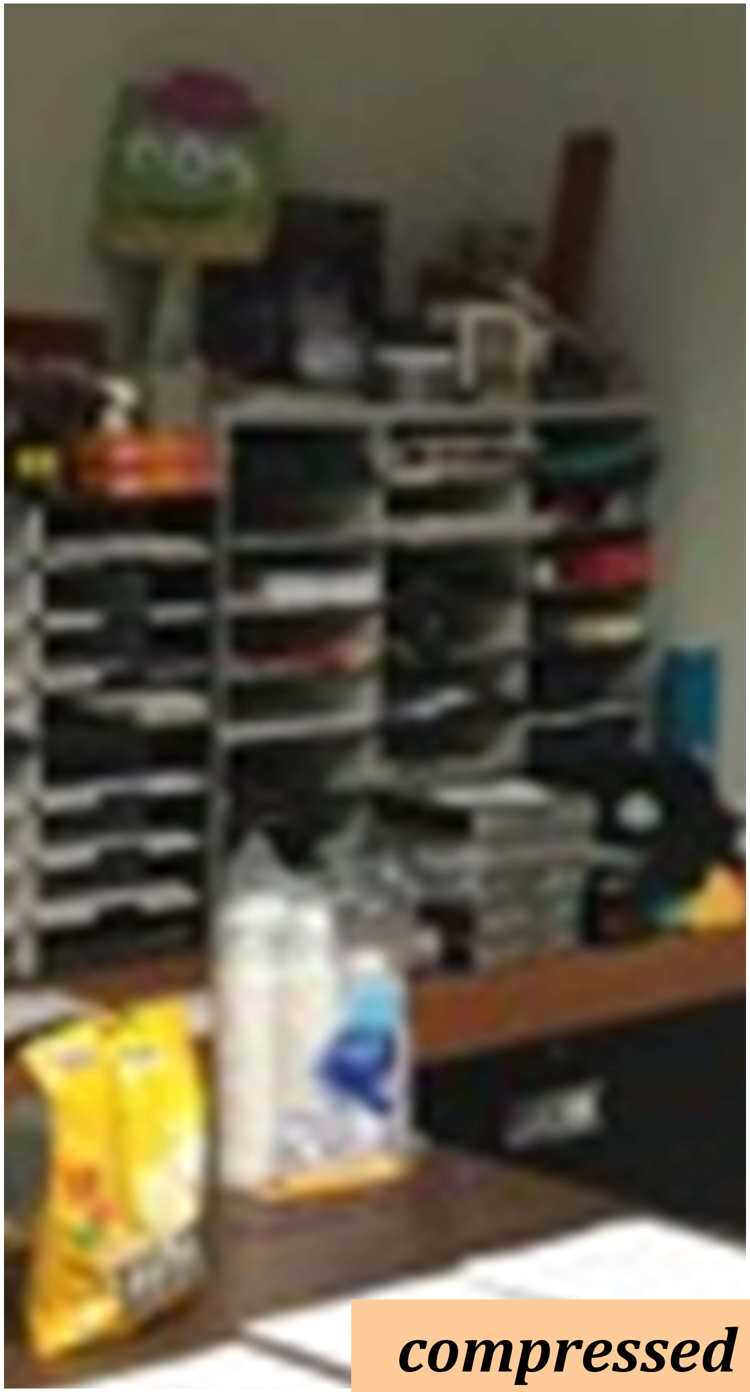
***original:** 2.3mb*



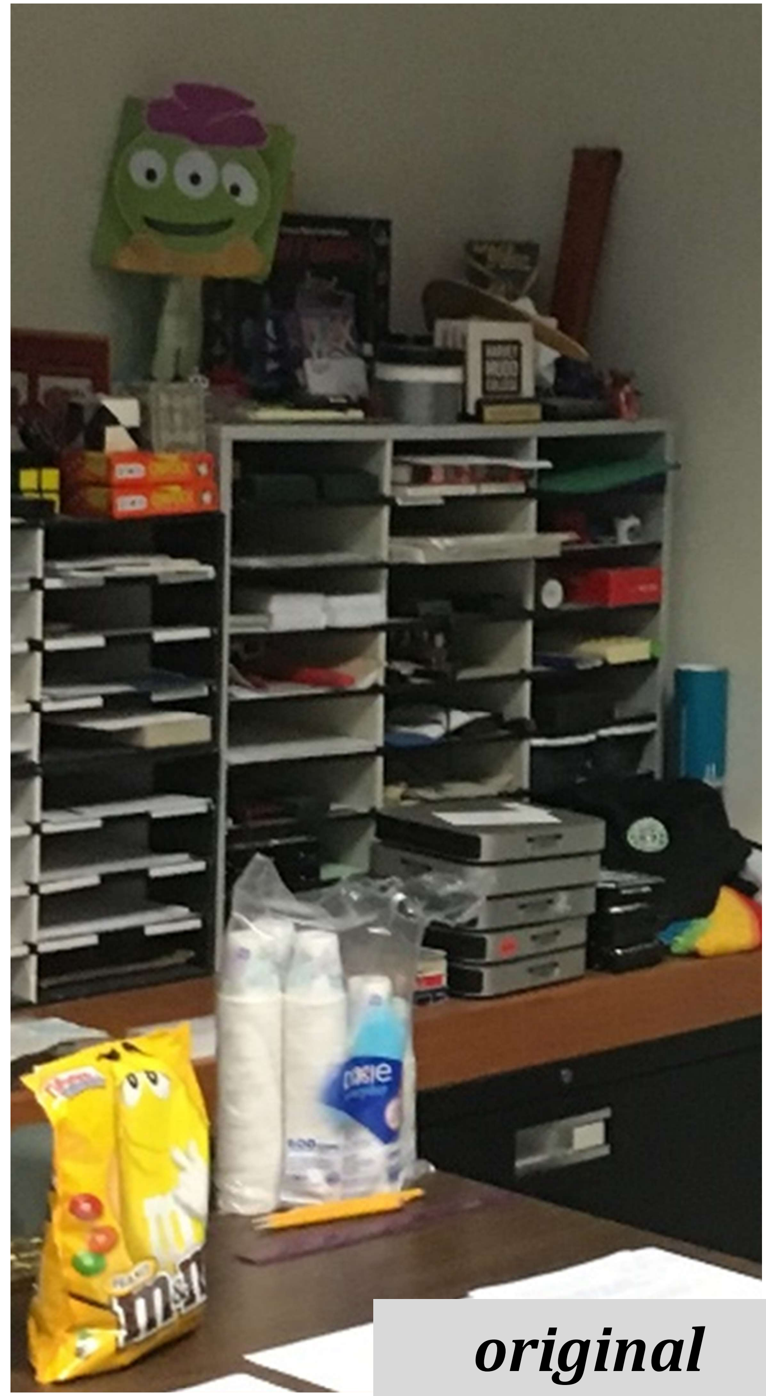


compressed

original

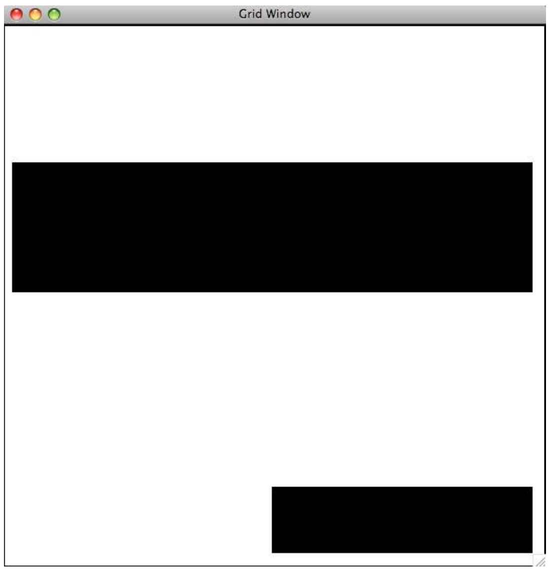


compressed



original

Hw4: ***lossless*** binary image compression



Binary Image

00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111

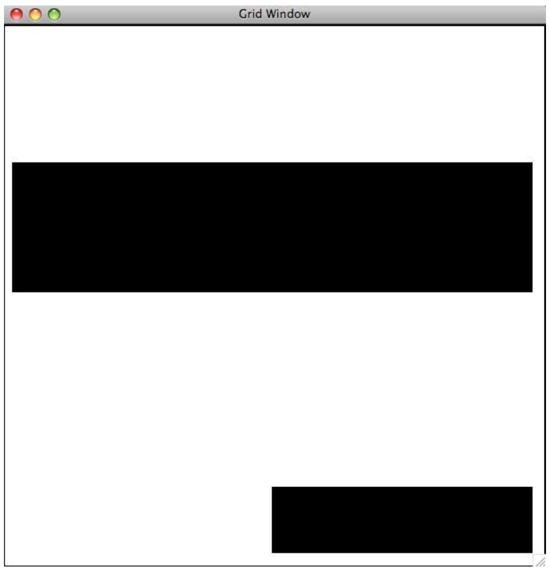
same-data streaks

Encoding as raw bits

one big string of 64 characters

a very compressible image...

Hw4: *lossless* binary image compression



Binary Image

00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111

same-data streaks

Encoding as raw bits

one big string of 64 characters

If our images tend to have long streaks of unchanging data, how might we represent it more efficiently, ***but still in binary***?

`compress`
`uncompress`

"0000000000000000111111111111111100000000000000000000000000001111"

Hw4: lossless image compression

0 is the first digit

There are 16 of them.

1 is the next digit

Again, there are 16 of them.

0 is the next digit

There are 28

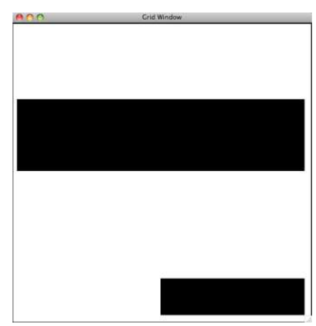
1 is the final digit

There are 4

01000001100000011100110 0

01000001100000011100 1100

00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111



One possible algorithm:

bit #repeats

Any <u>problems</u> with this?

Hw4: lossless image compression

0 is the
first digit

and there are 1,098,188 of them.

01000001100000111001100

It's ambiguous! this could just be a ***huge number*** of 0 pixels!

01000001100000111001100

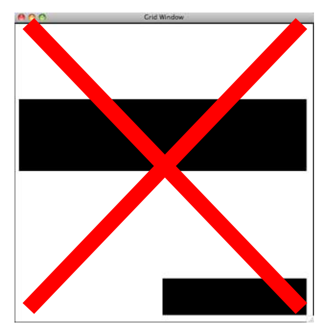
00000000
00000000
00000000
00000000
00000000
00000000
00000000
000000...

1,098,188 zeros!

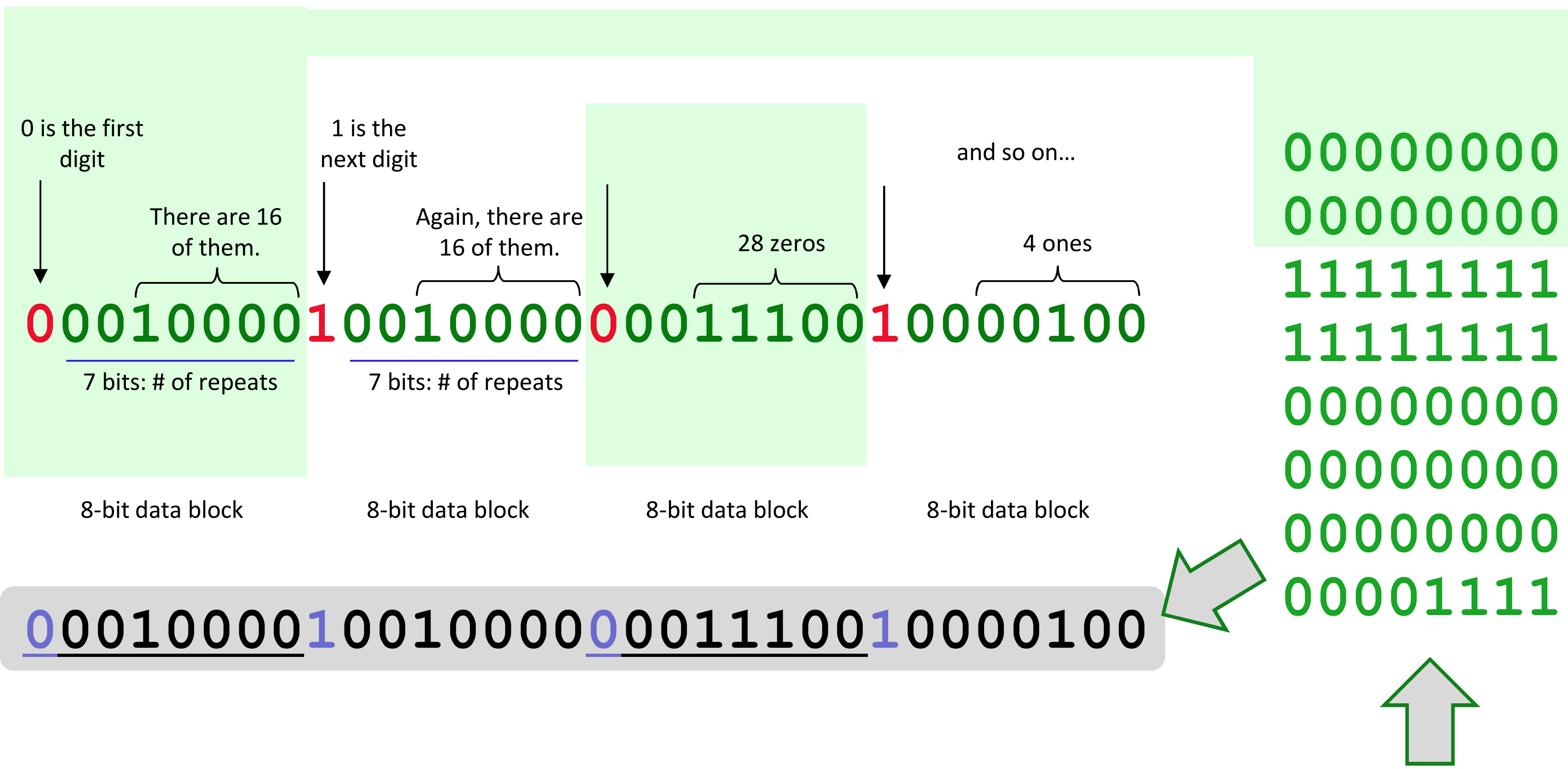
our algorithm:

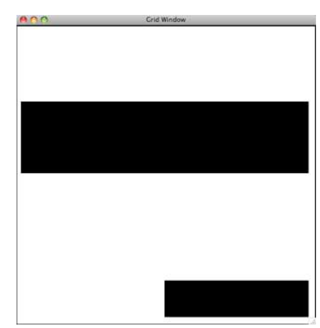
`bit #repeats`

could be misinterpreted!



fixed-width compression





We need ***fixed-width*** blocks:

bit	#repeats
1 bit fill	7 bits for the # of repeats

← **8-bits total** →

If you use ***7 bits*** to hold the # of consecutive repeats, what is the largest number of bits that *one block can represent*?

0 **0010000**

1 bit: the initial pixel

7 bits: # of repeats

8-bit total data block

7 bits?

B bits?

*What if you need a **larger** # of repeats?*

hw4 pr2

```
def compress( I ):
    """ returns the RLE of the
        input binary image, I """
```

a binary image, **I**

"0000000000000000000000000000000000000000001111111111111111111111111111111"

42 zeros

31 ones

"00101010100011111"

42, in binary

31, in binary

the "compressed" output returned by **compress(I)**

```python
def compress( I ):
    """ returns the RLE of the
        input binary image, I """
```

Name(s): ______________________________

Try it!

a 64-bit binary image, **IQuiz**

"0000000000001111111111111111111100000000000000000000011111111111"

12 zeros | 20 ones | 21 zeros | 11 ones

compress(I)

____ ______________ ____ ______________ ____ ______________ ____ ______________

the "compressed" output returned by **compress(IQuiz)**

Then, discuss …

What helper function would be useful for **compress**?

What's an image **I** whose compressed output ***gets larger, not smaller***? (Aargh!)

- What are the BEST-compressible / WORST-compressible 64-bit images?
- How could you ***improve*** the algorithm so that it always compresses?!!

hw4 pr2

```
def compress( I ):
    """ returns the RLE of the
        input binary image, I """
```

Helper function?

a binary image, **IQuiz**

"000000000000111111111111111111110000000000000000000000011111111111"

12 zeros | 20 ones | 21 zeros | 11 ones

"00001100100101000001010110001011"

12 | 20 | 21 | 11

the "compressed" image returned from **compress(IQuiz)**

Use this!

`frontNum(S)` returns the # of times the first element of the input **S** appears consecutively ***at the start*** of S:

```
frontNum('1111010')
4

frontNum('00110010')
2
```

```
def frontNum(S):

    if len(S) <= 1:
        return

    elif                              :
        return

    else:
        return
```

1 base case: `len(S) <= 1:`

or 2 base cases: `len(S)== 0:` `len(S)== 1:`

If S == '' or S == '1' or S == '0'

If the first two bits **DO** match....

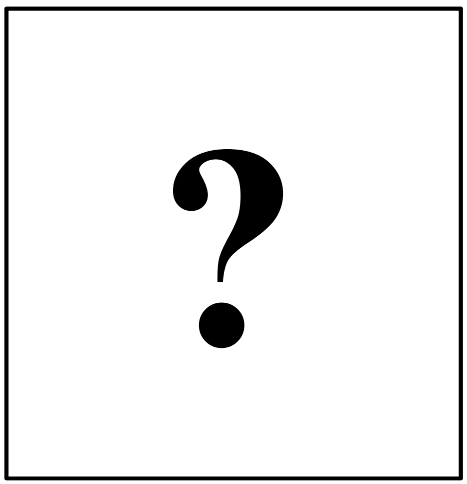
If the first two bits ***DON'T*** match....

BEST / **WORST** images?

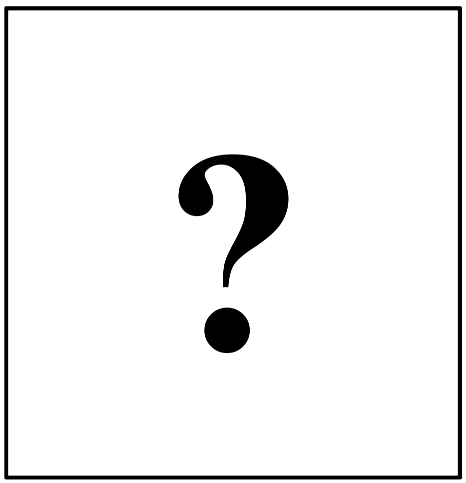
shortest compressed representation

longest compressed representation

What are the **BEST** and the **WORST** compression results you can get for an 8x8 image input (64 bits)?



BEST



WORST

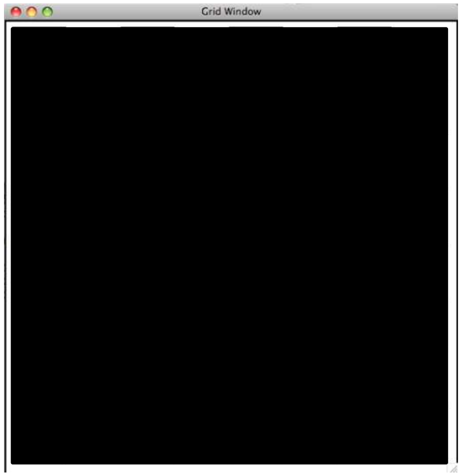
How could we improve this compression algorithm so that ***all images*** compress to smaller than the originals? That is, how can we make compression always <u>work</u> ?

?

shortest compressed representation

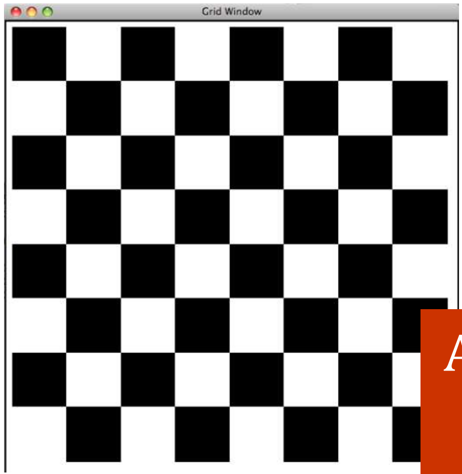
longest compressed representation

What are the **BEST** and the **WORST** compression results you can get for an 8x8 image input (64 bits)?



BEST

only 8 bits total!



WORST

aargh! 512 bits!

Anyone see why this is NOT QUITE the worst-compressable image?

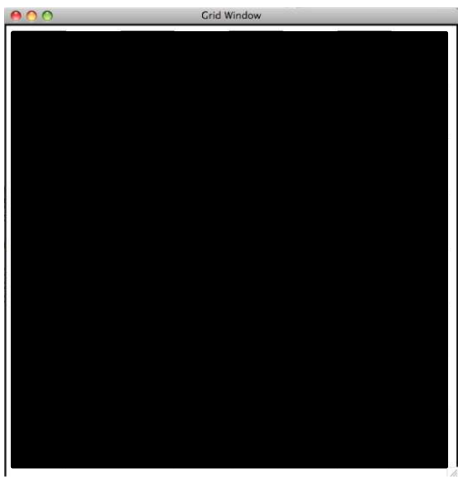
How could we improve this compression algorithm so that ***all images*** compress to smaller than the originals? That is, how can we make compression always work ?

?

shortest compressed representation

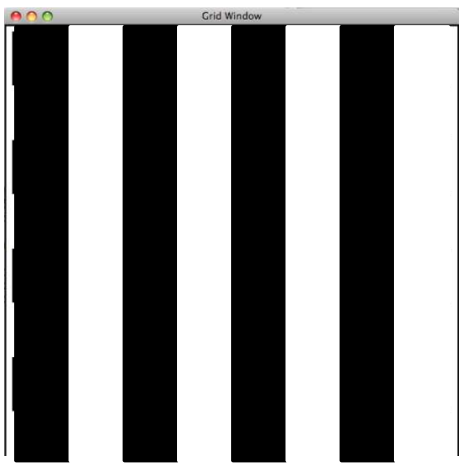
longest compressed representation

What are the BEST and the WORST compression results you can get for an 8x8 image input (64 bits)?



BEST

only 8 bits total!



WORST

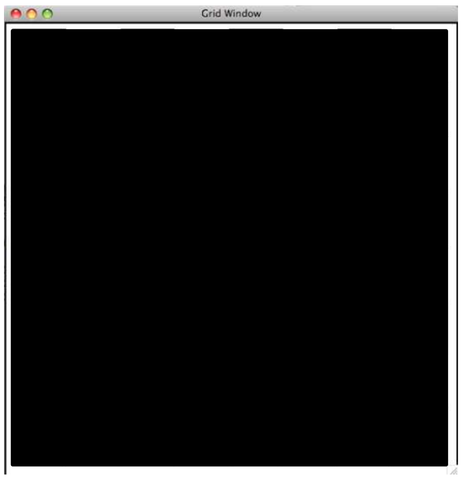
aargh! 512 bits!

How could we improve this compression algorithm so that ***all images*** compress to smaller than the originals? That is, how can we make compression always <u>work</u> ?

shortest compressed representation

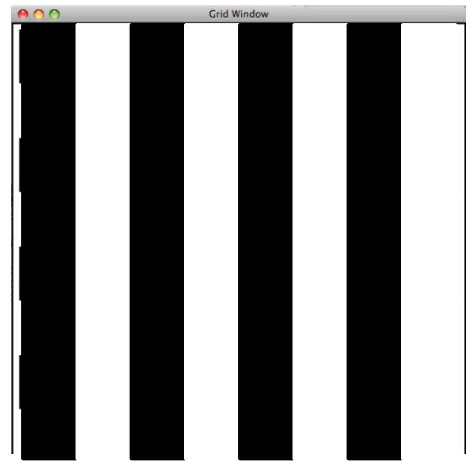
longest compressed representation

What are the BEST and the WORST compression results you can get for an 8x8 image input (64 bits)?



BEST

only 8 bits total!



WORST

aargh! 512 bits!

Impossible! *Provably!*

How could we improve this compression algorithm so that ***all images*** compress to smaller than the originals? That is, how can we make compression always <u>work</u> ?

!

It's <u>all</u> bits!

images, text, sounds, data, ...

even the string `'forty*two'` is represented
as a sequence of bits...

`'forty*two'`

`011001100110111101110010011101000111100100101010011101000111011101101111`

9 ASCII characters
8 bits each

9*8 == 72 bits total

All computation boils down to manipulating bits!

All computation is simply ***functions of bits***

binary inputs **S** and **T** → output, **S+T**

S	T	S+T
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

bitwise addition function

This week: you'll build this in Python.

`addB`

Next week: you'll design this with *wires.*

```
if S[-1] == '0' and T[-1] == '0':
    return ______________________ + '0'
```

binary inputs **S** and **T** → output, **S+T**

S	T	S+T
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

This week: you'll build this in Python.

bitwise addition function

addB

```
if S[-1] == '1' and T[-1] == '1' :
    return ________________________ + '0'
```

binary inputs **S** and **T** → output, **S+T**

S	T	S+T
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

This week: you'll build this in Python.

bitwise addition function

addB

Adding strings?

is **circuit** *addition!*

is **syntactic** *addition!*

syntactic ~ meaning-free

Multiplying by machine:

is **circuit** *multiplying!*

is **syntactic** *multiplying!*

Doing anything by machine…

is **circuit** *interaction!*

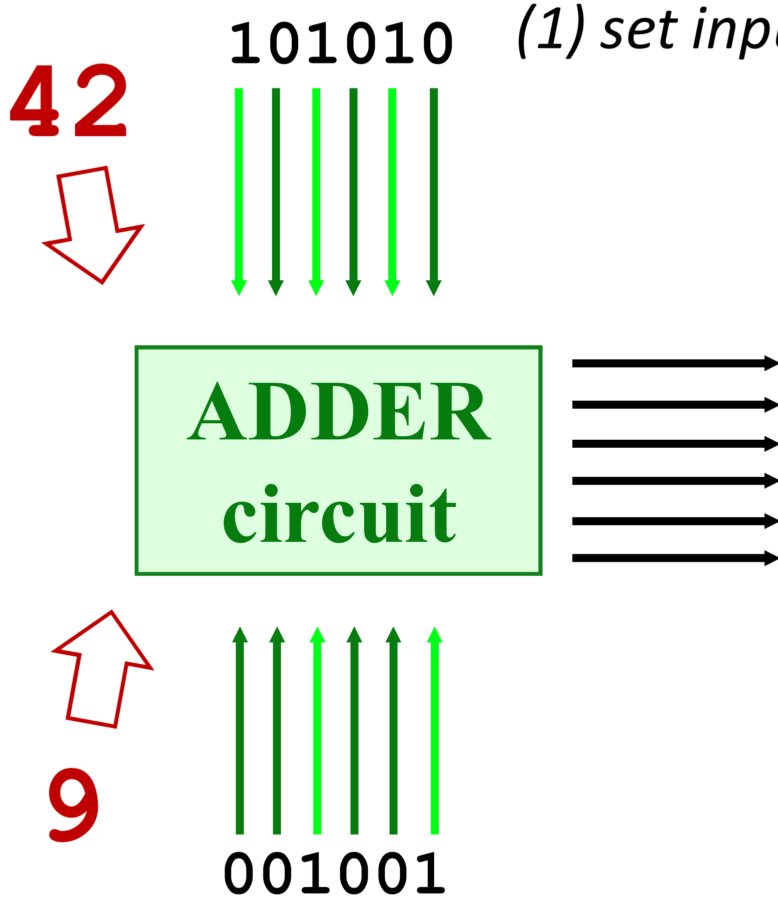
is **syntactic** *interaction!*

*means it can be done purely via **surface syntax**, which means it can be done **without thinking**…*

In a computer, each bit is represented as a <u>voltage</u>
(**1** is +5v and **0** is 0v)

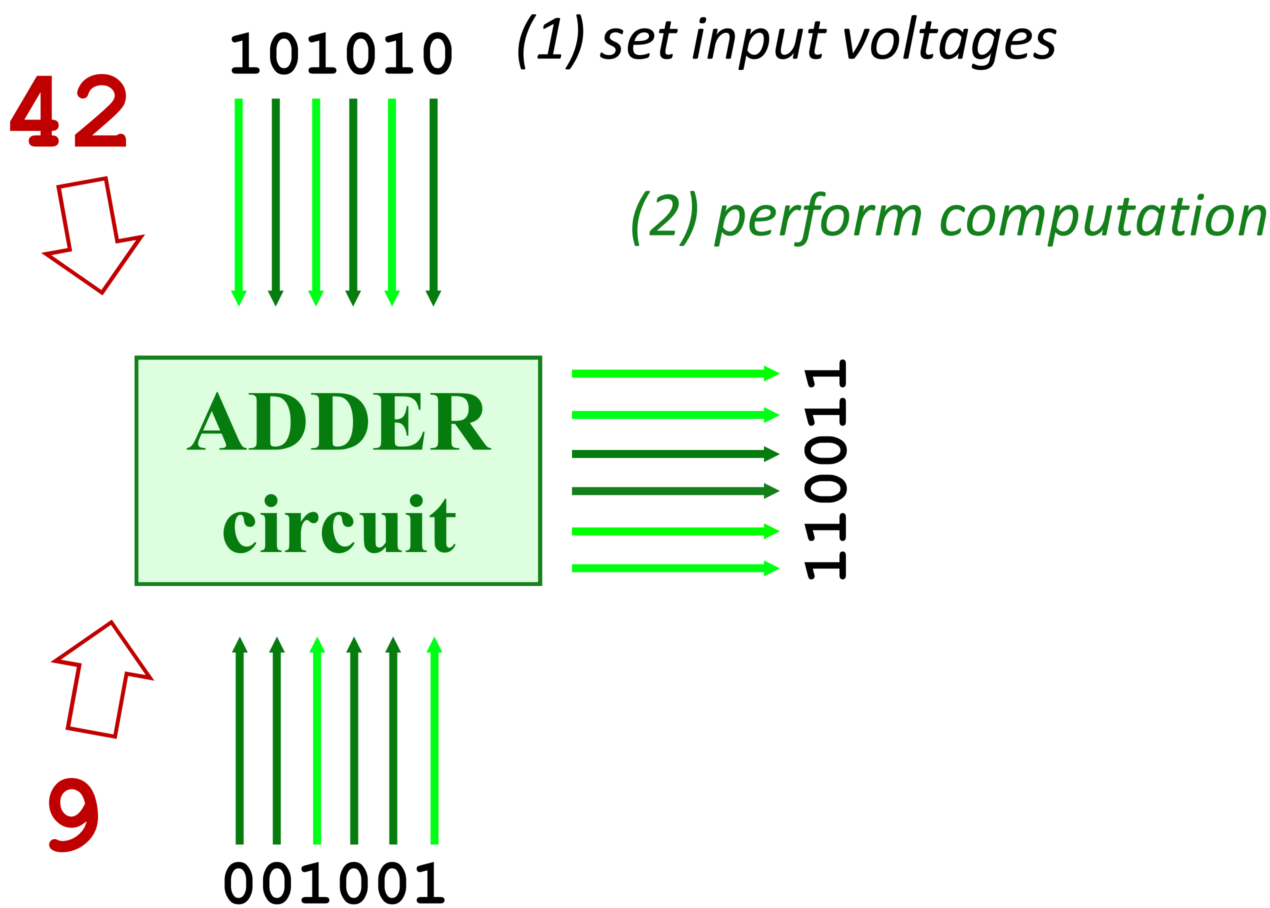
Computation is simply the **deliberate combination** of those voltages!

(1) set input voltages



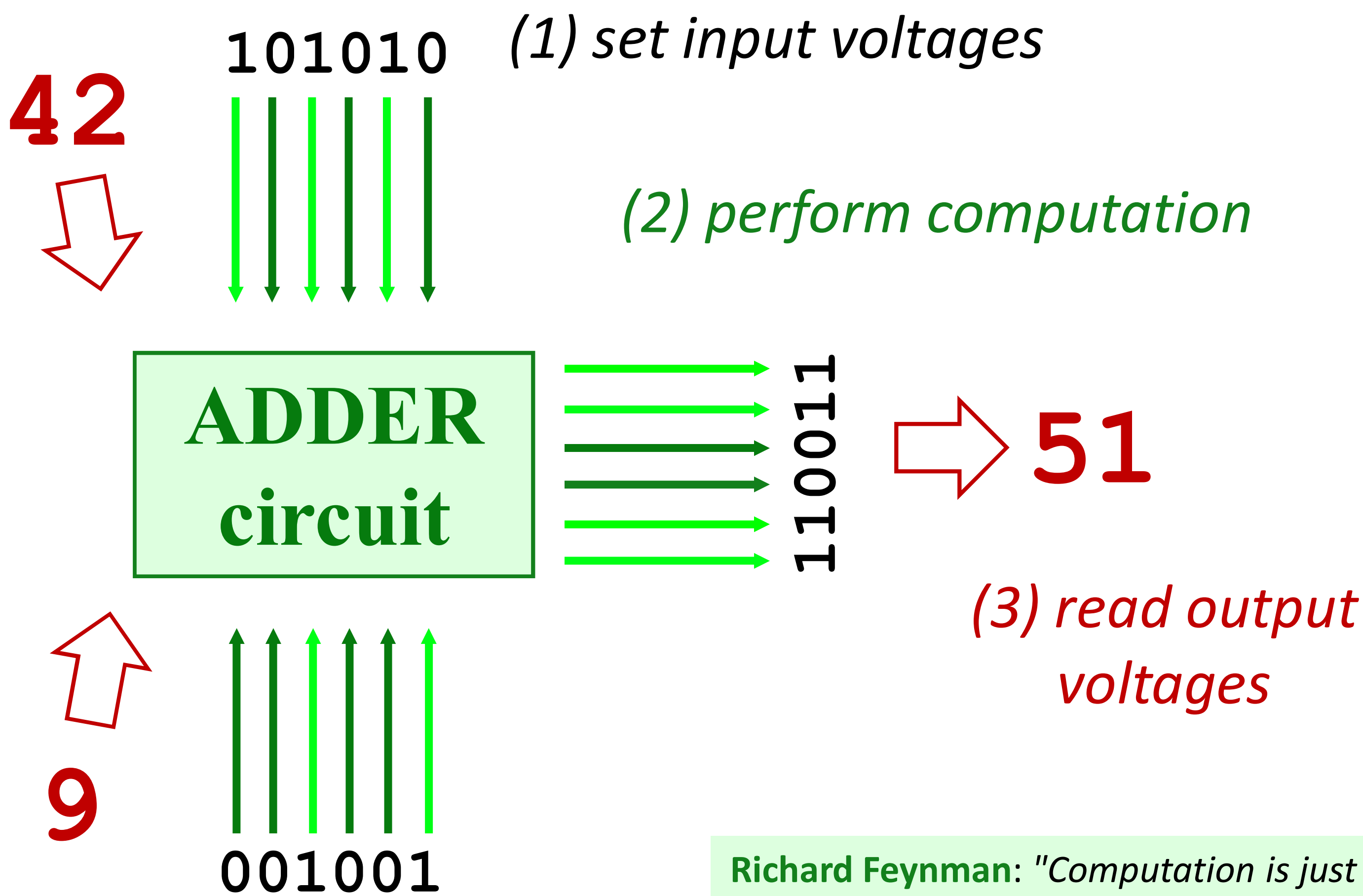
In a computer, each bit is represented as a <u>voltage</u> (**1** is +5v and **0** is 0v)

Computation is simply the **deliberate combination** of those voltages!



In a computer, each bit is represented as a voltage
(**1** is +5v and **0** is 0v)

Computation is simply the **deliberate combination** of those voltages!



But what's this green thing?

Richard Feynman: *"Computation is just a physics experiment that always works!"*

Our building blocks: ***logic gates***

AND outputs 1 only if **ALL** inputs are 1

AND

OR outputs 1 if **ANY** input is 1

OR

NOT reverses its input

NOT

These circuits are ***physical*** functions of bits…

… and *all* mathematical functions can be built from them!

Our building blocks: ***logic gates***

AND outputs 1 only if **ALL** inputs are 1

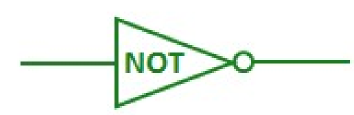
OR outputs 1 if **ANY** input is 1

NOT reverses its input

AND

OR

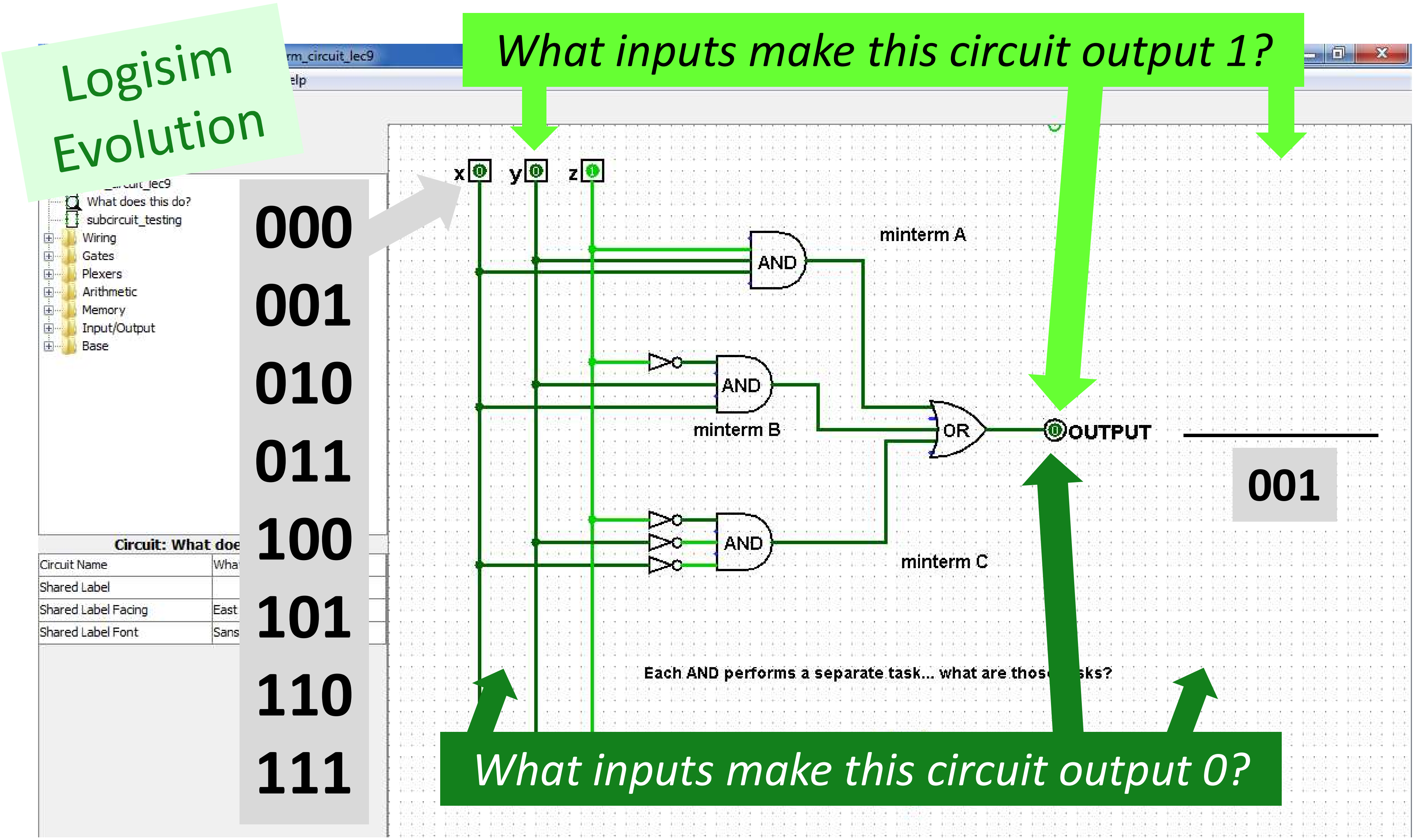
NOT



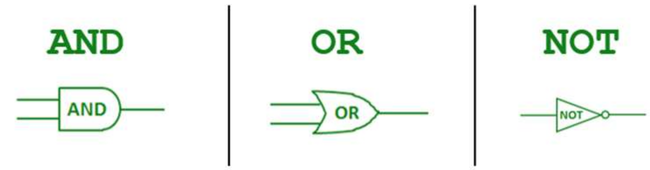
ALL circuits are **ANY** tions of bits…

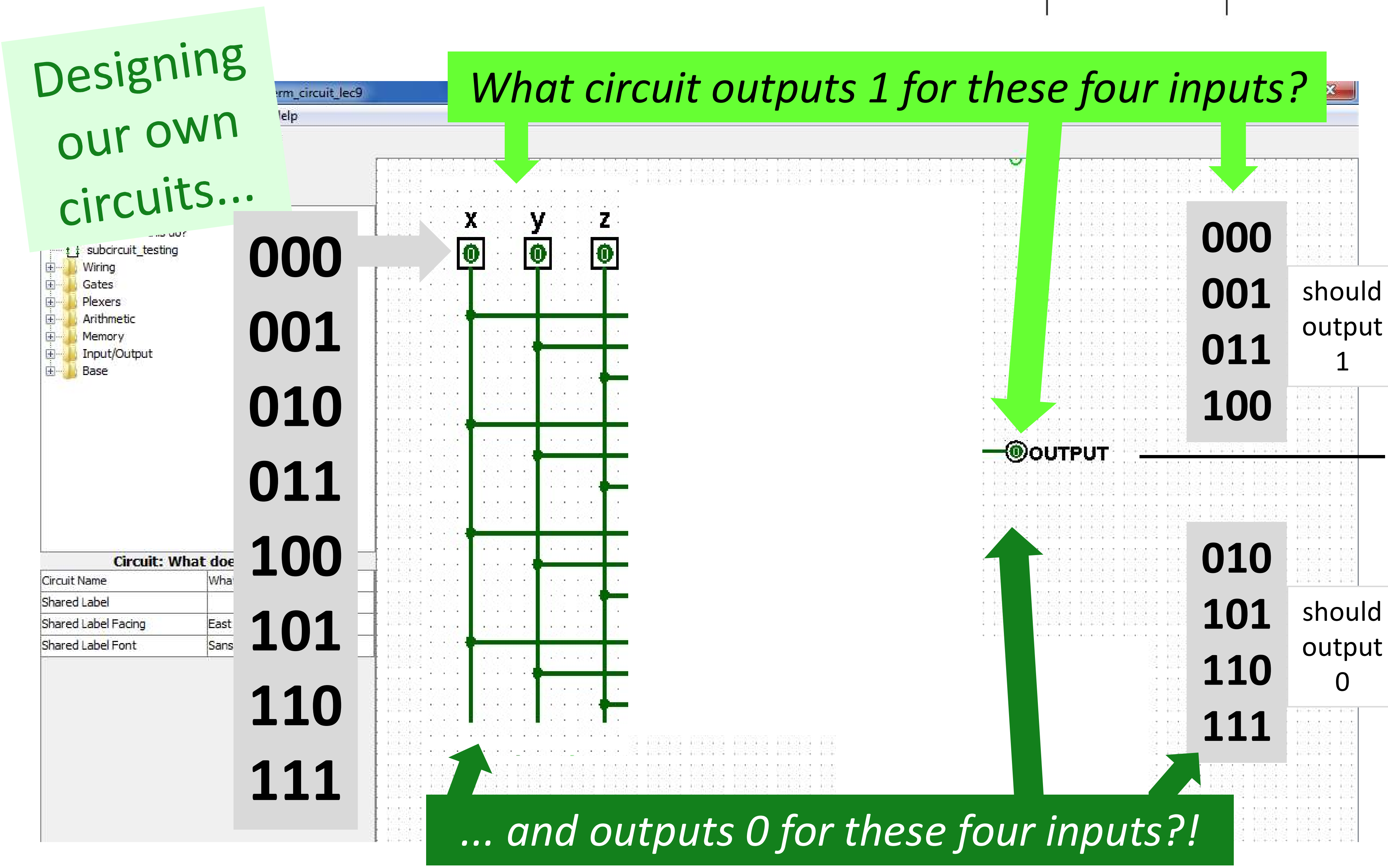
… and *all* mathematical functions can be built from them!

From gates to ***circuits***...



From gates to ***circuits***...





from circuit design…

next 2 weeks

…to a full computer!

Have an outst-ing and f-tunate weekend!

Why ?!