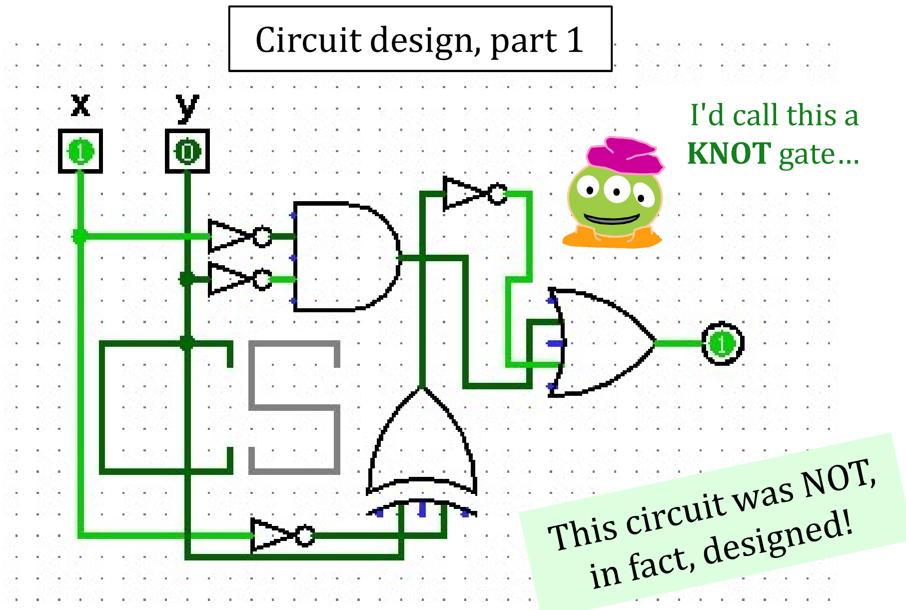
More ***bits*** of CS

Too many bits? <u>Compress!</u>



Below binary: ***physical circuits***

Hw #5 due Mon. **2/19**

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

pr1 *(lab)* binary ~ decimal

pr2 conversion + compression

extra image processing...

Lots of tutoring hrs - join in... !

P/F vs T/F

Bits' big idea

Aha! This can be implemented ***just with wiring***!

purely mechanical

Take-home Concept	Python	Bitwise reason
left-shifting by 1 ***doubles*** a value	`42 << 1` `84`	`'101010'` 42 in binary, columns double in value leftward `'1010100'` 84
right-shifting by 1 ***halves*** a value	`42 >> 1` `21`	`'101010'` 42 in binary, columns halve in value rightward `'10101'` 21

Do I halve to remember this?

I hope I don't have to remember L vs R!

No - *it falls out!*

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

binary inputs **A** and **B**		output, **A+B**
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
A	**B**	

This week: you'll build this in Python.

bitwise addition function

`addB`

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

Adding *strings*?

is **circuit** *addition!*

is **syntactic** *addition!*

```
       S       T
     '31'    '11'

def add10(S,T):
    """ adds the *strings* S and T
        as decimal numbers
    """
    if len(S) == 0: return T
    if len(T) == 0: return S
    eS = S[-1]
    eT = T[-1]
   if eS == '0' and eT == '1': return  add10(S[:-1],T[:-1]) + '1'
   if eS == '1' and eT == '1': return  add10(S[:-1],T[:-1]) + '2'
   if eS == '2' and eT == '1': return  add10(S[:-1],T[:-1]) + '3'
   if eS == '3' and eT == '1': return  add10(S[:-1],T[:-1]) + '4'
   # Lots more rules - how many in all?
```

eS ~ the "end of S"

eT ~ the "end of T"

```
S  '31'
T  '11'
   ______
```

Notice that this code doesn't "understand" addition at all!

*Carry*ing on...

hw5: **addB**

S **T**

'23' '19'

```
def add10(S,T):
    """ adds the *strings* S and T
        as decimal numbers
    """
    if len(S) == 0: return T
    if len(T) == 0: return S
    eS = S[-1]
    eT = T[-1]
    if eS == '0' and eT == '1': return  add10(S[:-1],T[:-1]) + '1'
    if eS == '1' and eT == '1': return  add10(S[:-1],T[:-1]) + '2'
    if eS == '2' and eT == '1': return  add10(S[:-1],T[:-1]) + '3'
    if eS == '3' and eT == '1': return  add10(S[:-1],T[:-1]) + '4'
    # what if we have to carry to the next column?
    if eS == '3' and eT == '9':
        return
```

eS ~ the "end of S"

eT ~ the "end of T"

```
S   '23'
T   '19'
   ______
```

Notice that this code doesn't "understand" addition at all!

Lab Debriefing &
`hw5pr2.py`

Lab Debriefing & **hw5pr1.py**

in

42

```
ntb( 42 )
ntb( 21 ) + '0'
ntb( 10 ) + '1'
ntb( 5 ) + '0'
ntb( 2 ) + '1'
ntb( 1 ) + '0'
ntb( 0 ) + '1'
''
```

'101010'

out

```
def numToBin( N ):
    """ converts a decimal int to a binary string
    """
    if N==0:         return ''
    elif N%2==0:     return numToBin( N//2 ) +  '0'
    elif N%2==1:     return numToBin( N//2 ) +  '1'
```

these are awfully similar…

Lab Debriefing & **hw5pr1.py**

in

42

```
ntb( 42 )
ntb( 21 ) + '0'
ntb( 10 ) + '1'
ntb( 5 ) + '0'
ntb( 2 ) + '1'
ntb( 1 ) + '0'
ntb( 0 ) + '1'
''
```

'101010'

out

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)
```

make sure your notes have **TWO** forward slashes!

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

Lab Debriefing & **hw5pr1.py**

in

'101010'

```
btn('101010')
2*btn('10101') + 0
2*btn('1010') + 1
2*btn('101') + 0
2*btn('10') + 1
2*btn('1') + 0
2*btn('') + 1
0
```

2*1

2*2*2*1

2*2*2*2*2*1

42

out

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

again, awfully similar…

Lab Debriefing & **hw5pr1.py**

in

'101010'

```
btn('101010')
2*btn('10101') + 0
2*btn('1010') + 1
2*btn('101') + 0
2*btn('10') + 1
2*btn('1') + 0
2*btn('') + 1
0
```

2*1

2*2*2*1

2*2*2*2*2*1

42

out

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 the need for another **`if`**

Ariane 5

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

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

16 bits ← 64 bits

version 4 version 5

How *far* 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		
	31 bits		

How far *back* can we remember...?

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 far *back* can we remember...?

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...

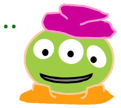
Top videos

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]

Another overflow error!

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

Ariane 5

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

Is everything bits?

`IndexError`   `TypeError`   `HumanError`

16 bits ← 64 bits

version 4   version 5

Insight: Ancient Egyptian multiplication

From Wikipedia, the free encyclopedia

Is everything bits?

Not sure - but surprisingly much ***is*** ...

Insight Ancient Egyptian Multiplication

halver		dbler	
21	×	**6**	(ans. should be 126)

halver	dbler
21	**6**

Example

AEM/RPM algorithm

Write the factors in two columns.

Repeatedly **halve** the LEFT and **double** the RIGHT. (toss remainders…)

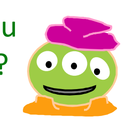
Pull out the RIGHT values where the LEFT values are ***odd***.

***Sum** those values for the answer!*

Why does this work?

a.k.a. *RPM*

Buddy, can you spare an eye?



Name(s) ______________________

Quiz

Ancient Egyptian Multiplication!

halver		dbler	(ans. should be 126)
21	×	6	
21		6	6
10		12	
5		24	24
2		48	
1		96	+ 96
			126

Example

AEM algorithm

Write the factors in two columns.

Repeatedly **halve** the LEFT and **double** the RIGHT. (toss remainders...)

Pull out the RIGHT values where the LEFT values are ***odd***.

***Sum** those values for the answer!*

halver dbler

11 × 15 (ans. ~ 165)

Try it!

halver dbler

12 × 20 (ans. ~ 240)

Extra: *Why does this always work?* **Hint**: it's binary!

Name(s) ______________________

Quiz

Ancient Egyptian Multiplication!

halver × dbler

21 × 6 (ans. should be 126)

halver	dbler	
21	6	6
10	12	
5	24	24
2	48	
1	96	+ 96
		126

Example

AEM algorithm

Write the factors in two columns.

Repeatedly **halve** the LEFT and **double** the RIGHT. (toss remainders...)

Pull out the RIGHT values where the LEFT values are ***odd***.

***Sum** those values for the answer!*

Try it!

halver × dbler

11 × 15 (ans. ~ 165)

halver × dbler

12 × 20 (ans. ~ 240)

Extra: *Why does this always work?* **Hint**: it's binary!

Insight AEM algorithm

Decimal	
21	6
21	6
10	12
5	24
2	48
1	96

Binary

$$
\begin{array}{rrr}
 & 110_{6} & \\
\times & 10101_{21} & \\
\hline
 & 110_{6} & 6 \\
 & 0000_{12} & \\
 & 11000_{24} & 24 \\
 & 000000_{48} & \\
+ & 1100000_{96} & +\ 96 \\
\hline
 & 1111110_{126} & 126
\end{array}
$$

Although in ancient Egypt the concept of base 2 did not exist, the algorithm is essentially the same algorithm as long multiplication after the multiplier and multiplicand are converted to binary. The method as interpreted by conversion to binary is therefore still in wide use today as implemented by binary multiplier circuits in modern computer processors.

Insight Egyptian + Russian Multiplication

Decimal

11	15
11	15
5	30
2	60
1	120

Binary

$$\begin{array}{rr} & 1111_{15} \\ \text{x} & 1011_{11} \\ \hline & 1111_{15} \\ & 11110_{30} \\ & 000000_{60} \\ + & 1111000_{120} \\ \hline & 10100101_{165} \end{array}$$

$$\begin{array}{rr} & 15 \\ & 30 \\ & \\ + & 120 \\ \hline & 165 \end{array}$$

Insight Egyptian + Russian Multiplication

Decimal

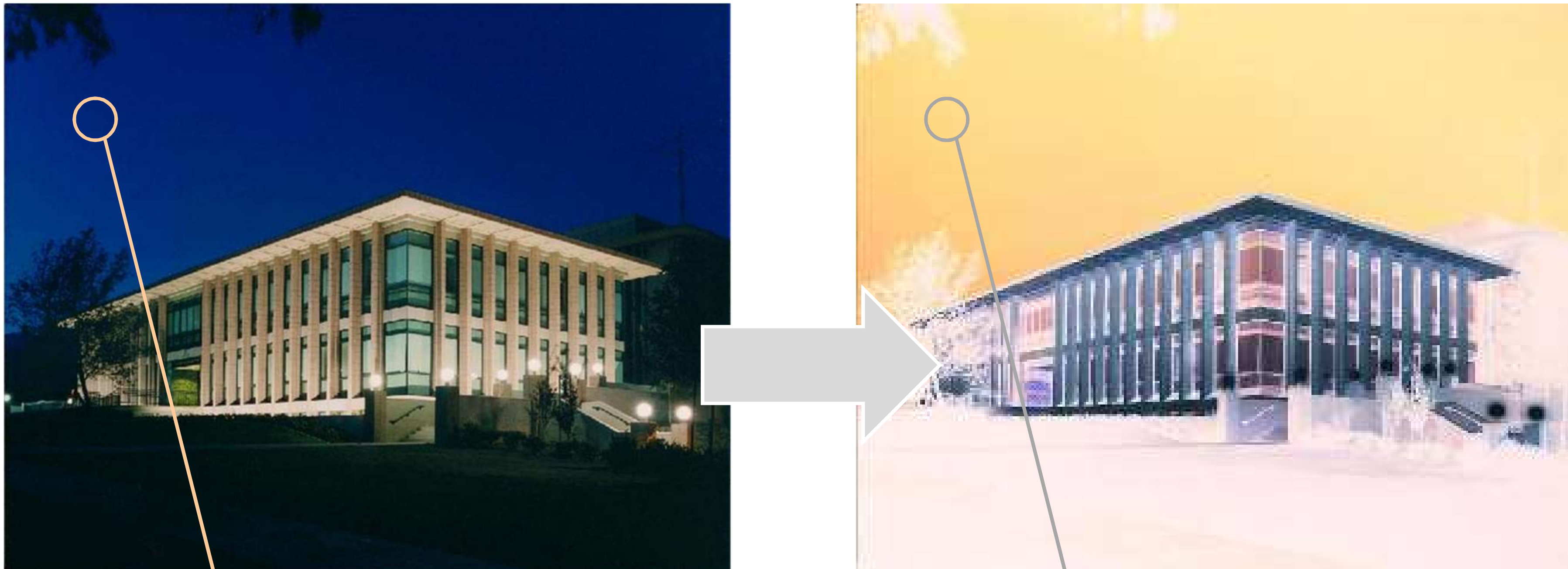
$\boxed{12}$	$\boxed{20}$
12	20
6	40
3	80
1	160

Binary

$$
\begin{array}{rl}
 10100 & _{20} \\
\times \quad 1100 & _{12} \\
\hline
 00000 & _{20} \\
 000000 & _{40} \\
 1010000 & _{80} \\
+\,10100000 & _{160} \\
\hline
 11110000 & _{240}
\end{array}
\qquad
\begin{array}{r}
80 \\
+\,160 \\
\hline
240
\end{array}
$$

Hw5: *images are just bits, too*!

hw5pr3 ***(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

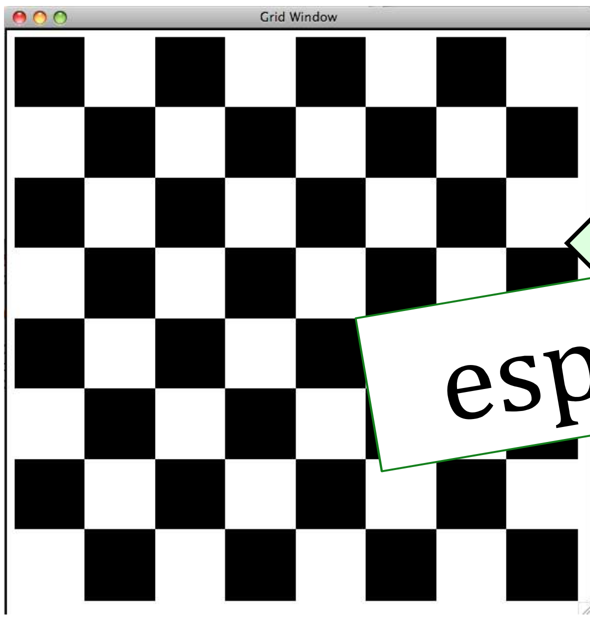
red = 1 (out of 255)
green = 36 (out of 255)
blue = 117 (out of 255)

<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?

Hw5: *images are just bits, too*!



Binary Image

especially ***binary*** images

```
10101010
01010101
01010101
10101010
01010101
```

Encoding as raw bits

one big string of 64 characters

```
"1010101001010101101010100101010110101010010101011010101001010101"
```

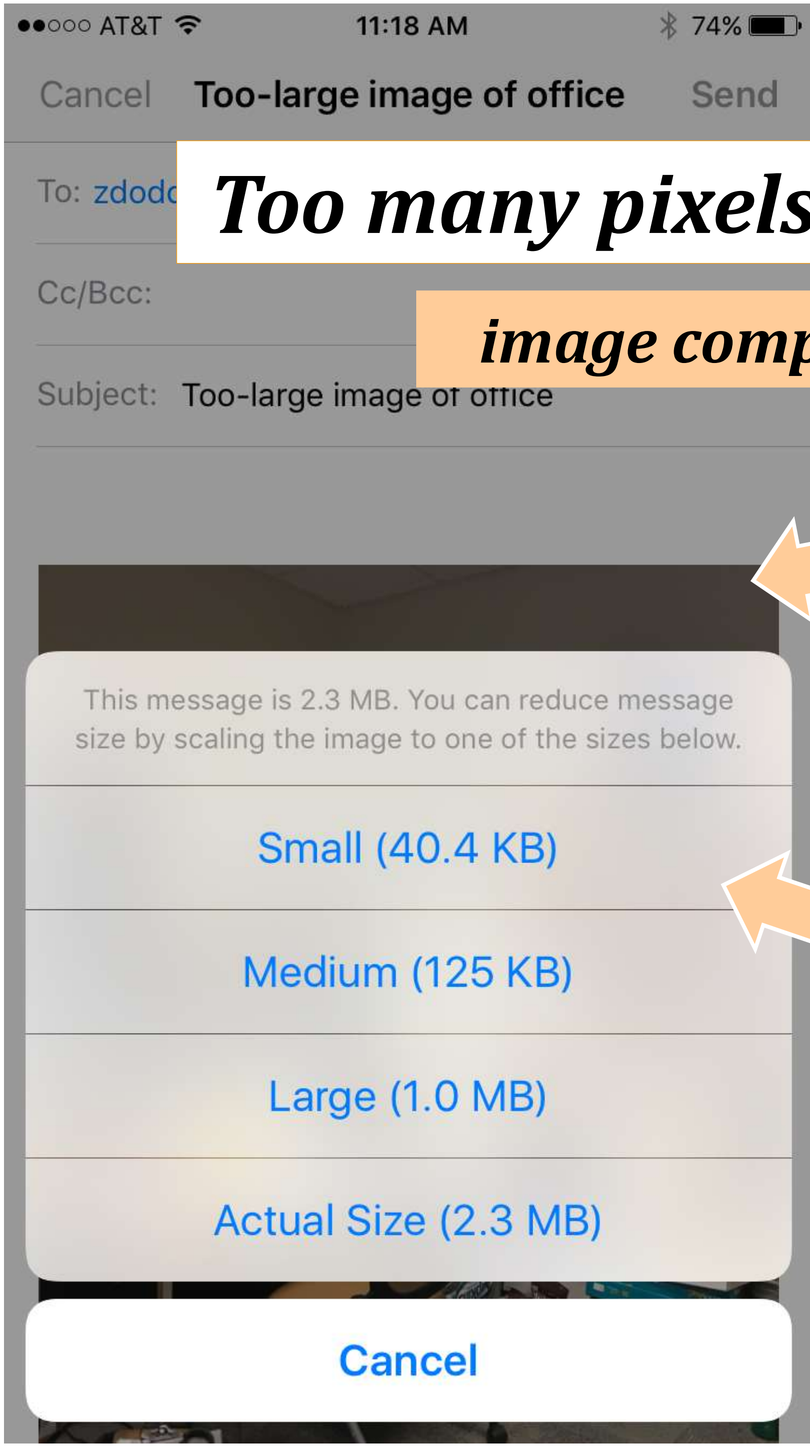
likelier binary image...

and a reasonable candidate for ***compression***

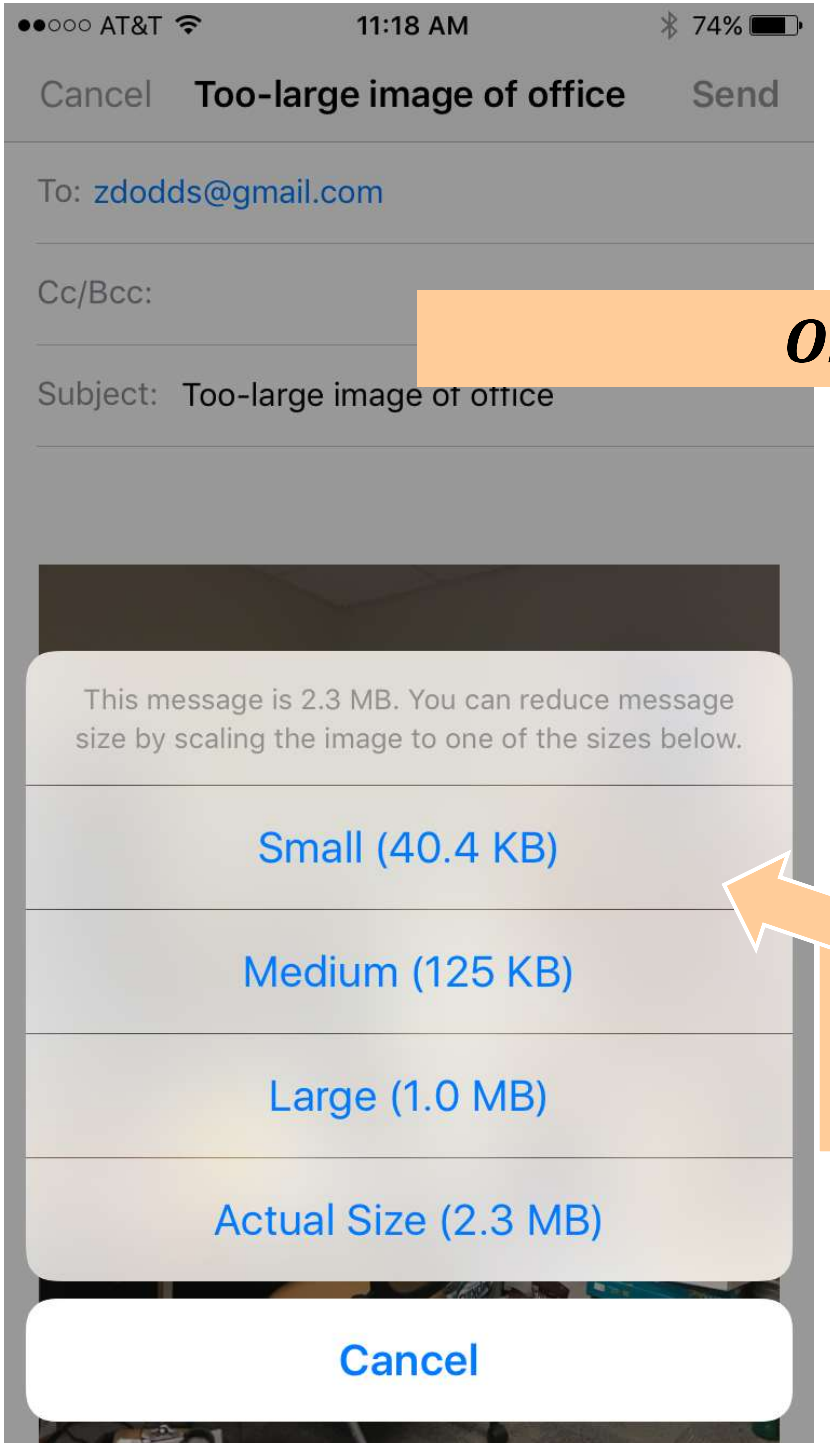
home!

Too many pixels… too little time + space!

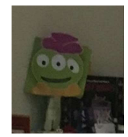
image compression is everywhere!



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



One solution!

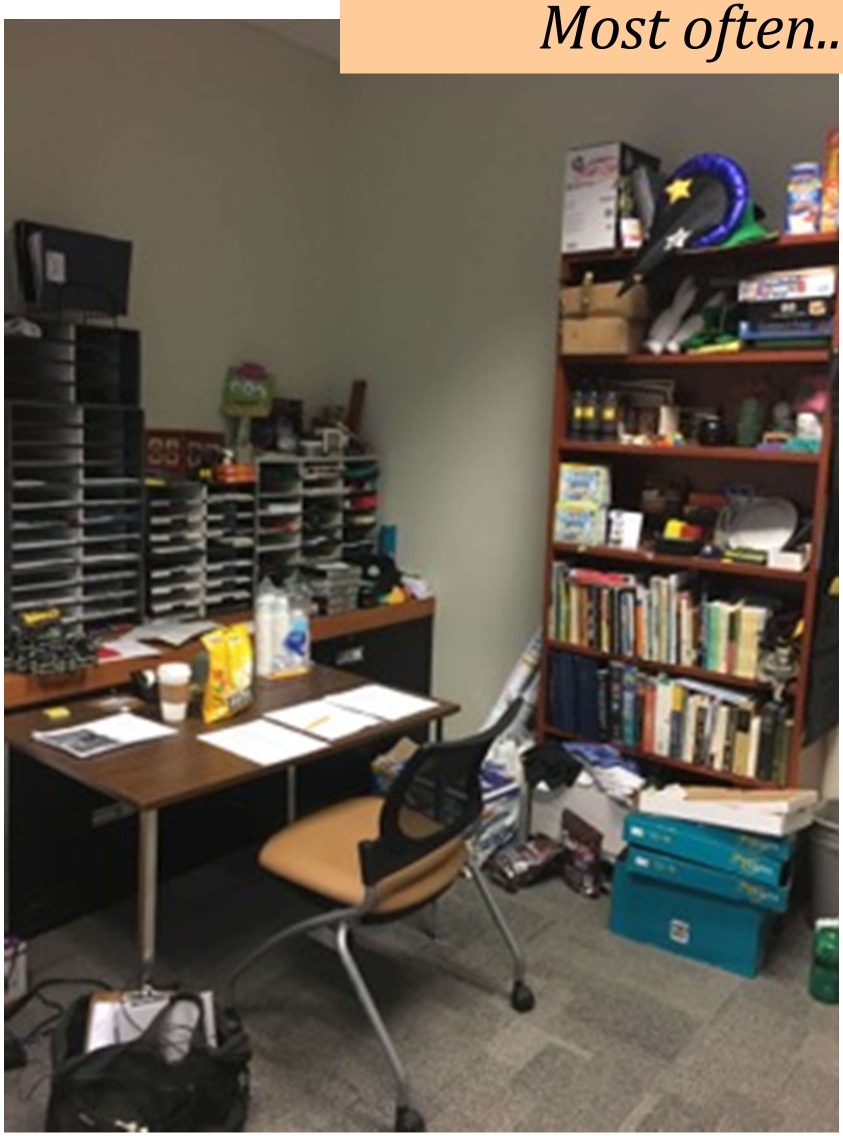


We throw away 98% of the image ***area***!

Looks like the right 2% to keep!

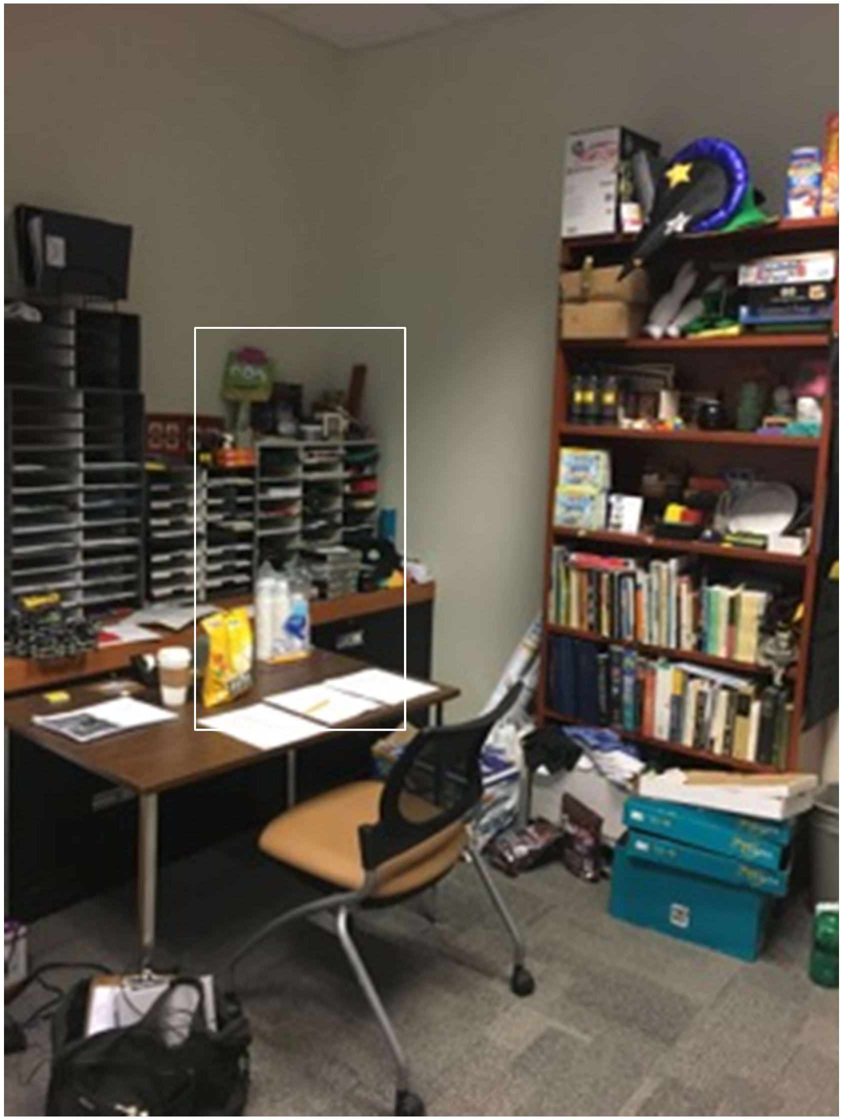
How is it possible to throw away **98%** of the image data!?

Most often... what's done?

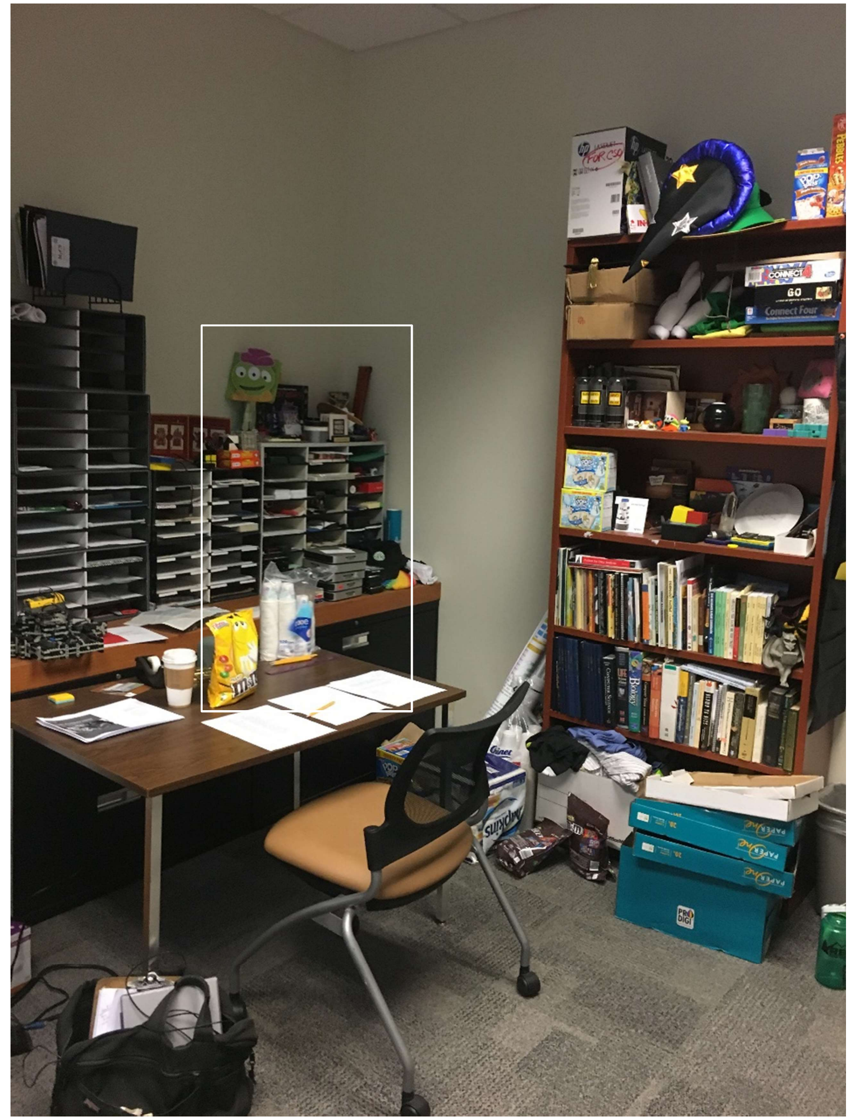


***compressed** to 40kb*

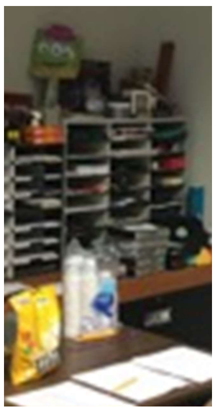
***original:** 2.3mb*

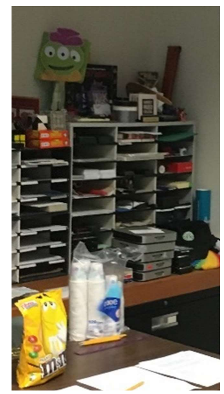


***compressed** to 40kb*



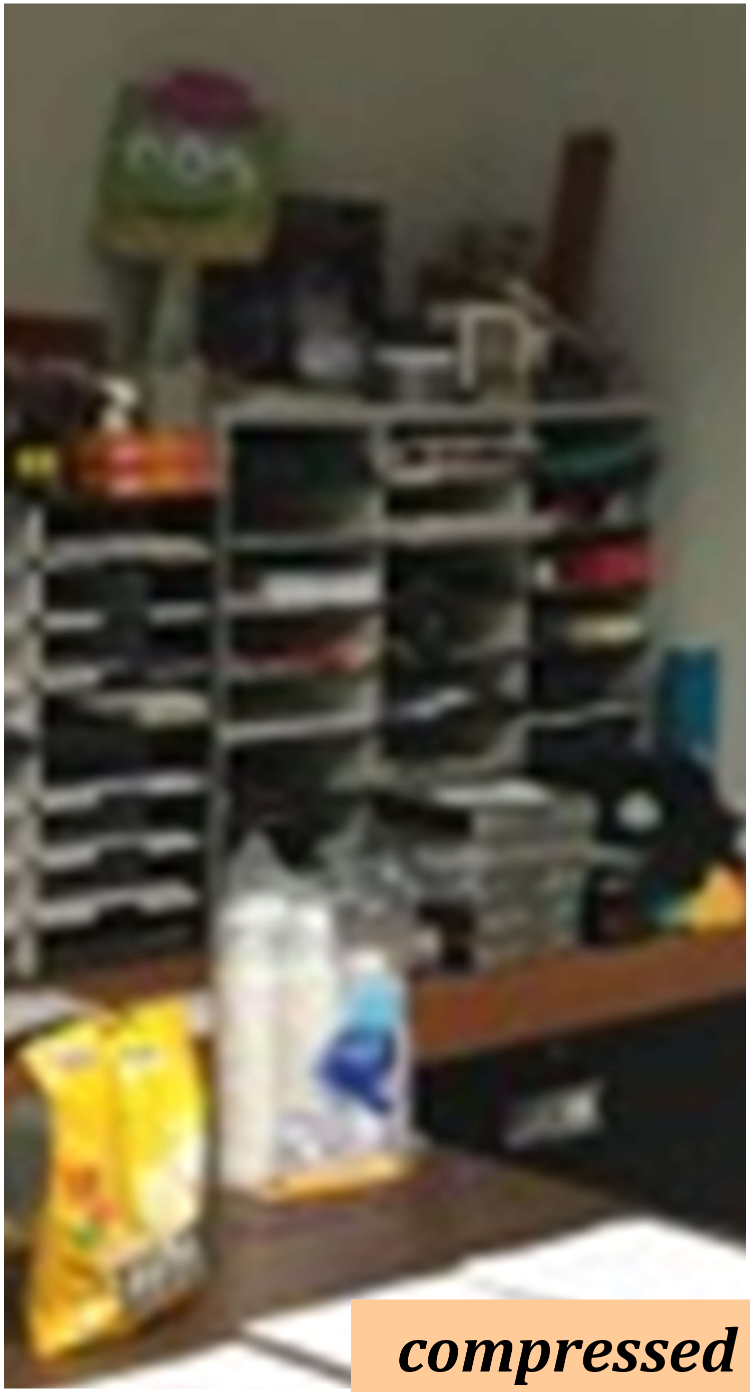
***original:** 2.3mb*



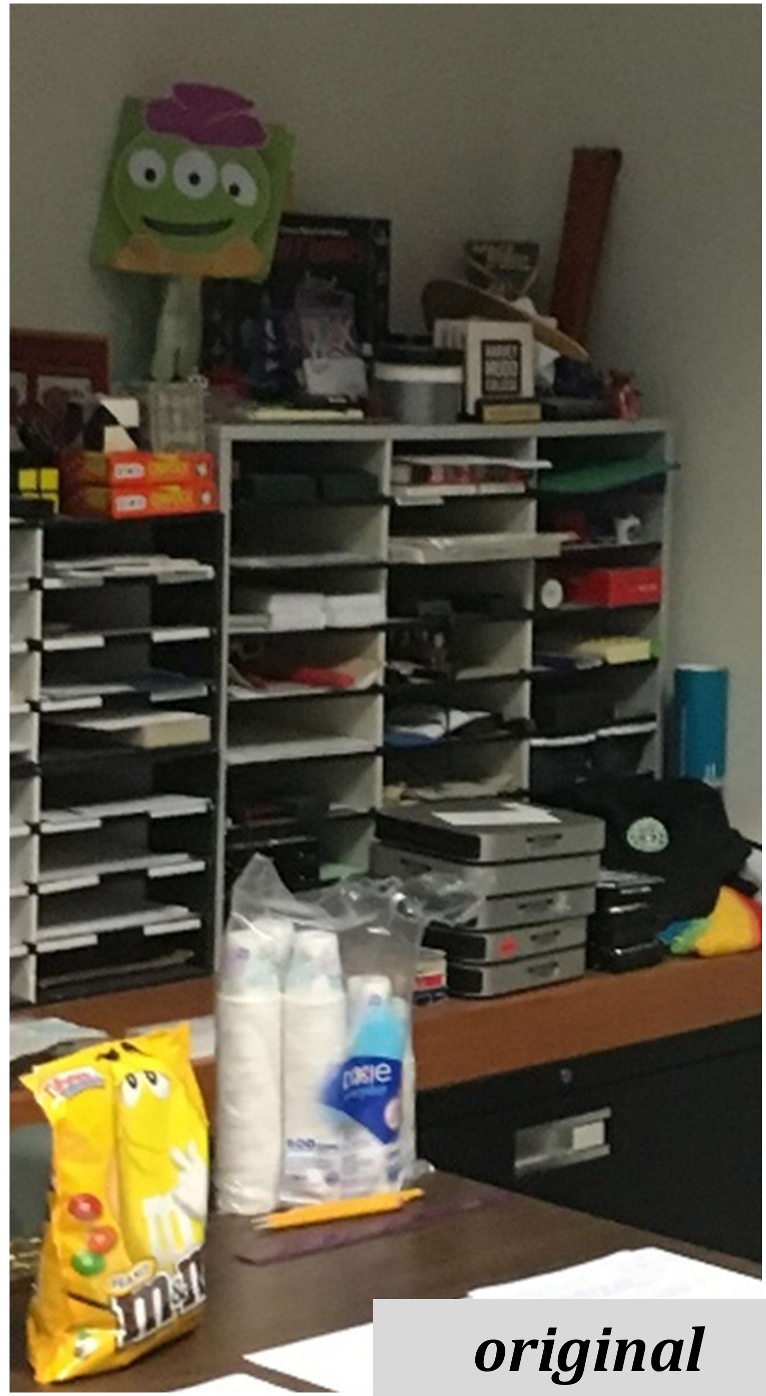


compressed

original

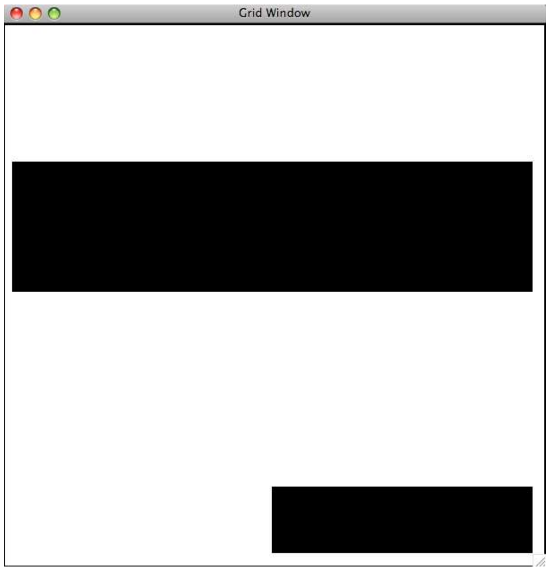


compressed



original

Hw5: ***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 <u>long streaks of unchanging data</u>, how might we represent it more efficiently, ***but still in binary***?

compress
uncompress

"0000000000000000111111111111111100000000000000000000000000001111"

Hw5: *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

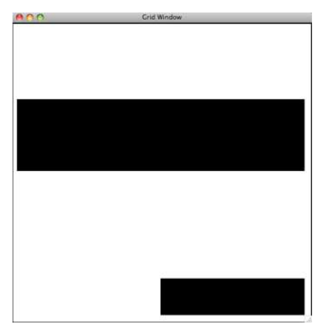
1 is the final digit

There are 4

01000001100000011100110 0

```
010000011000000111001100
```

```
00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111
```



One possible algorithm:

```
bit #repeats
```

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

Hw5: *lossless* image compression

0 is the
first digit

and there are 1,098,188 of them.

$0\underbrace{1000001100000111001100}$

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

`0100000110000011100 1100`

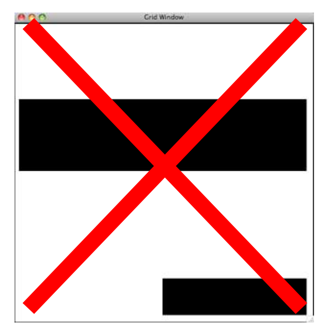
our algorithm:

`bit #repeats`

could be misinterpreted!

00000000
00000000
00000000
00000000
00000000
00000000
00000000
000000...

1,098,188 zeros!



fixed-width compression

0 is the first digit

There are 16 of them.

1 is the next digit

Again, there are 16 of them.

28 zeros

and so on...

4 ones

0 0010000 1 0010000 0 0011100 1 0000100

7 bits: # of repeats

7 bits: # of repeats

8-bit data block

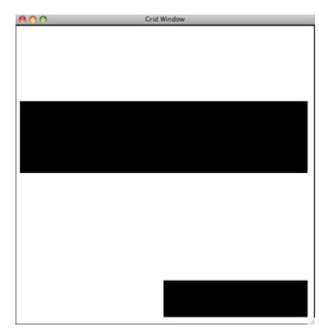
8-bit data block

8-bit data block

8-bit data block

00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111

00010000100100000001110010000100



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 """


def uncompress( CI ):
    """ returns the binary image I
        from the run-length-encoded,
        "compressed" input, CI """
```

hw5 pr2

a binary image

"0000000000000000111111111111111100000000000000000000000000001111"

16 zeros | 16 ones | 28 zeros | 4 ones

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

the "compressed" image:

"00010000100100000001110010000100"

16 | 16 | 28 | 4

```
def uncompress( CI ):
    """ returns the binary image I
        from the run-length-encoded,
        "compressed" input, CI """
```

back to the original binary image

"0000000000000000111111111111111100000000000000000000000000001111"

16 zeros | 16 ones | 28 zeros | 4 ones

hw5 pr2

a binary image

"0000000000000000111111111111111100000000000000000000000000001111"

16 zeros | 16 ones | 28 zeros | 4 ones

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

what helper function might be useful here?

the "compressed" image:

"00010000100100000001110010000100"

16 | 16 | 28 | 4

```
def uncompress( CI ):
    """ returns the binary image I
        from the run-length-encoded,
        "compressed" input, CI """
```

back to the original binary image

"0000000000000000111111111111111100000000000000000000000000001111"

16 zeros | 16 ones | 28 zeros | 4 ones

Try it!

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

Try writing the recursive function, **`frontNum(S)`**

Examples…

```
>>>   frontNum('1111010')
4
>>>   frontNum('00110010')
2
```

```
def frontNum(S):

    if  len(S)<=1:
        return

    elif  S[0] == ______ :
        return

    else:
        return
```

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

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

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

shortest — BEST; longest — WORST

EXTRA! How can you change our algorithm so that compressed images are always smaller than the originals?

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

?

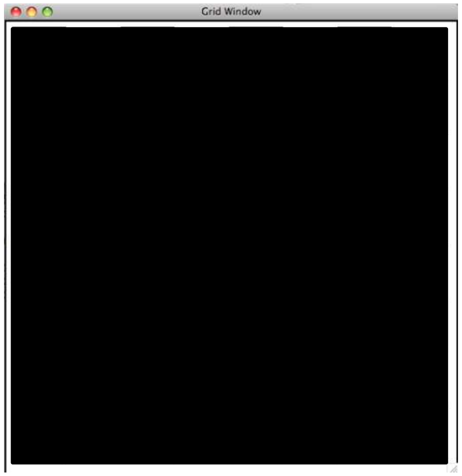
How could we improve this compression algorithm so that ***all images*** compress to smaller than the originals? That is, how can we make compression always work ?

shortest compressed
representation

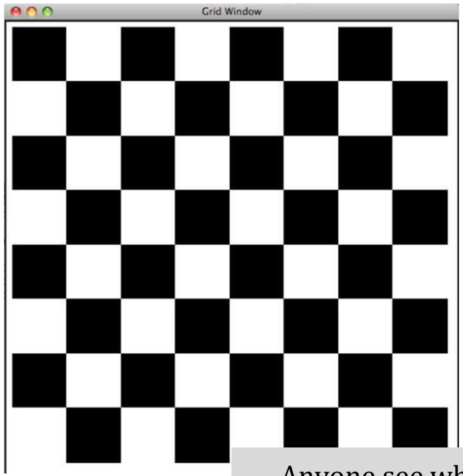
longest compressed
representation

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

only 8 bits total!



aargh! 512 bits!



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

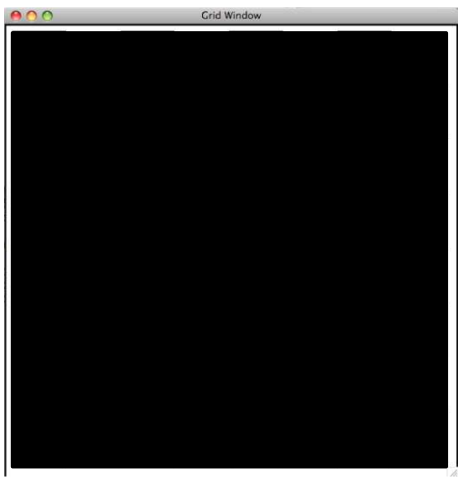
?

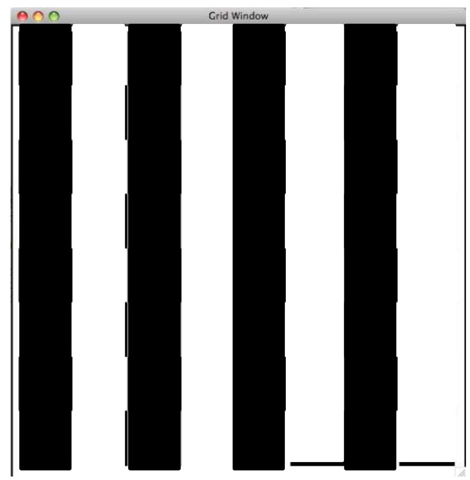
How could we improve this compression algorithm so that ***all images*** compress to smaller than the originals? That is, how can we make compression always <u>work</u> ?

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)?





This is provably IMpossible!

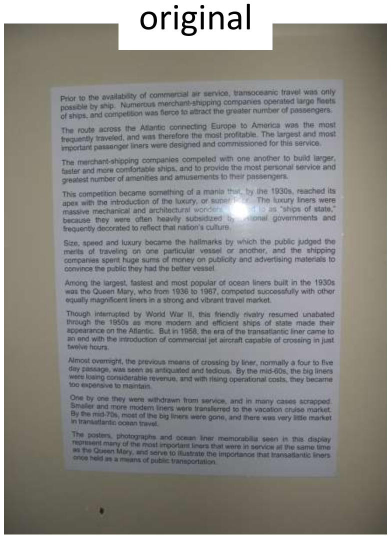
!

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 ?

Binary images in practice...

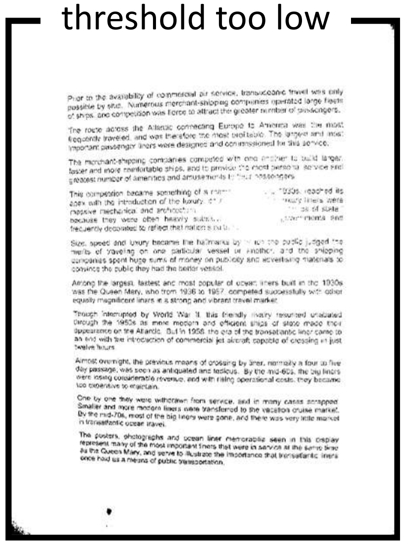
Laserfiche®

original



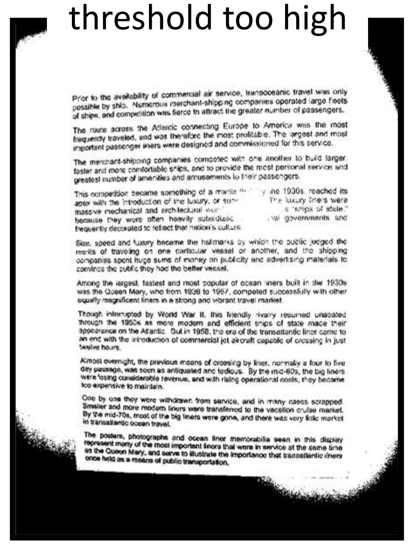
Original Image

threshold too low



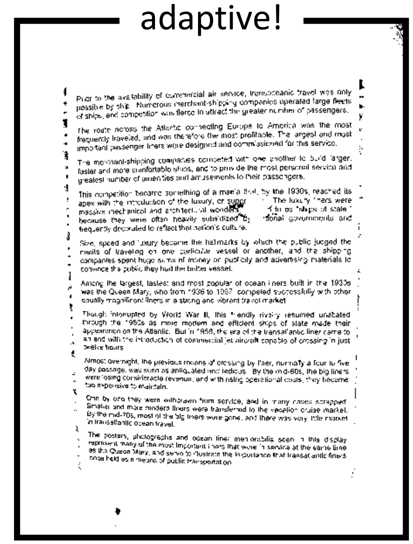
T = 78/255

threshold too high

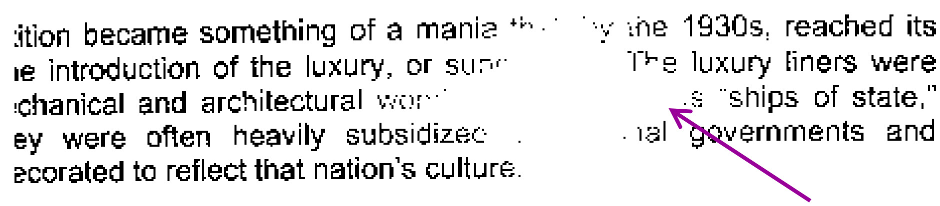


T = 120/255

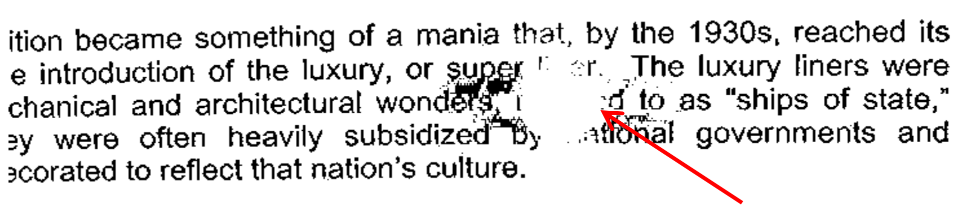
adaptive!



Adaptive T



T = 120/255



Adaptive Threshold

No magic!
(unfortunately!)

Portrait vs. landscape?

ENGINEERING

Transforming Technology: Engineering for a Changing World

- Wednesday, January 28, 7:00 P.M.
  "ARMing the Wireless World"
  Timothy J. O'Donnell '74, President, Nordam, Inc.
- Wednesday, February 11, 7:00 P.M.
  "The Engineering Challenges of Satellite Television"
  David A. Baylor, Trustee, Harvey Mudd College, and Executive Vice President, DIRECTV, Inc.
- Wednesday, March 3, 7:00 P.M.
  "Power Electronics"
  Alex Lidow, CEO, International Rectifier Corp.
- Wednesday, March 10, 7:00 P.M.
  "Megaprojects: Major Challenges in Urban Infrastructure Design/Build Around the World"
  Jude P. Laspa '65, Trustee, Harvey Mudd College, and Executive Vice President and Deputy COO, Bechtel Corporation
- Wednesday, March 24, 7:00 P.M.
  "Software Engineering"
  Ivan E. Sutherland, Sun Fellow and Vice President, Sun Microsystems, Inc.
- Wednesday, April 7, 7:00 P.M.
  "Electroactive Polymers as Artificial Muscles: Realities and Challenges"
  Yoseph Bar-Cohen, Senior Research Scientist, Jet Propulsion Laboratory
- Wednesday, April 21, 7:00 P.M.
  "Biomedical Device Design and Development"
  Jerome J. Jackson '76, Trustee, Harvey Mudd College, and Vice President of Research and Development, Stellar Technology, Inc.

Dr. Bruce J. Nelson '74 Distinguished Speaker Series ■ Spring 2004 ■ Harvey Mudd College

Harvey Mudd College is a coeducational liberal arts college of engineering, science and mathematics that also places strong emphasis on humanities and the social sciences. The college's aim is to graduate engineers, scientists and mathematicians sensitive to the impact of their work on society.

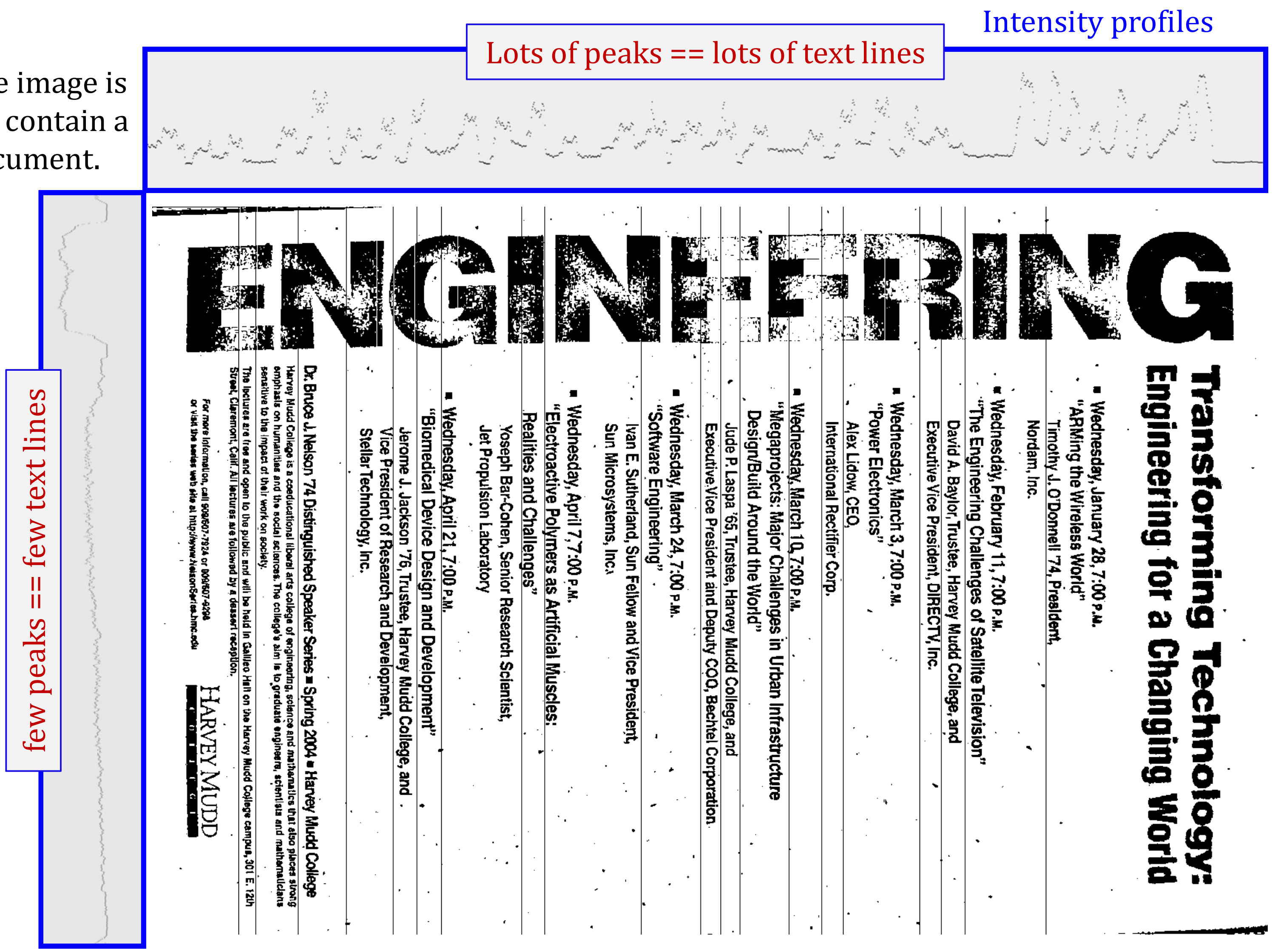
The lectures are free and open to the public and will be held in Galileo Hall on the Harvey Mudd College campus, 301 E. 12th Street, Claremont, Calif. All lectures are followed by a dessert reception.

For more information, call 909/607-7924 or 909/607-4298 or visit the series web site at http://www.NelsonSeries.hmc.edu

HARVEY MUDD COLLEGE

Portrait vs. landscape?

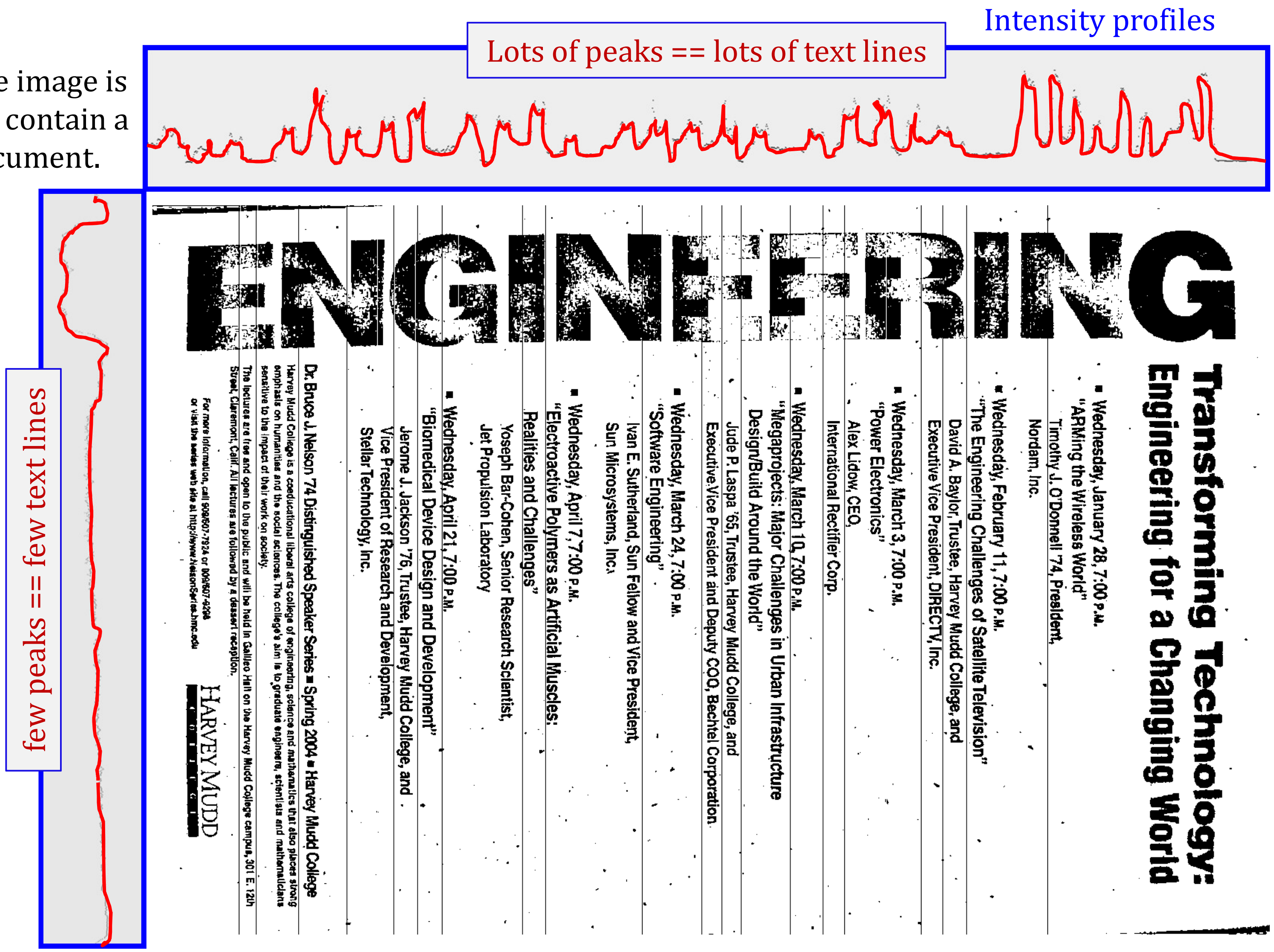
This landscape image is determined to contain a ***portrait*** document.



Right-side up?

Portrait vs. landscape?

This landscape image is determined to contain a ***portrait*** document.



Right-side up?

It's all bits!

images, text, sounds, data, ...

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

'forty*two'

0110011001101111011100100111010001111001<u>00101010</u>011101000111011101101111

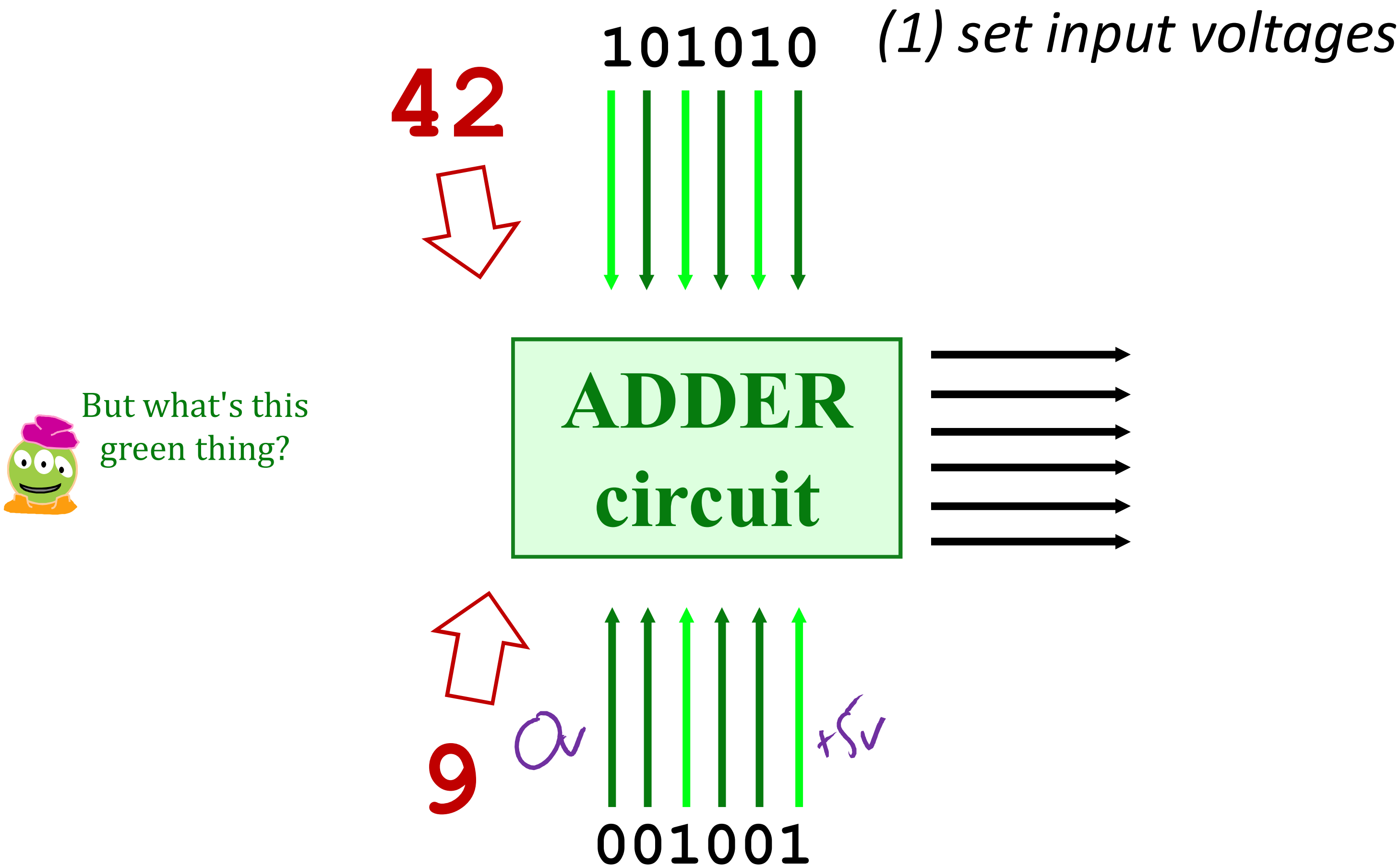
9 ASCII characters
8 bits each

9*8 == 72 bits total

All computation boils down to manipulating bits!

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!

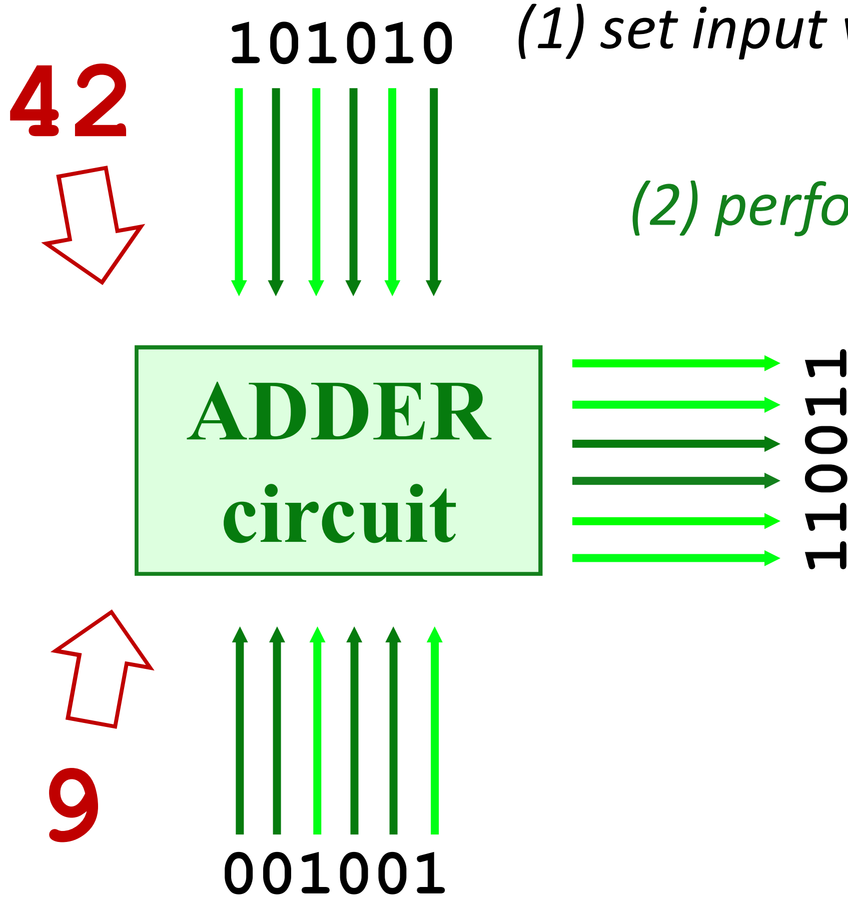


In a computer, each bit is represented as a <u>voltage</u> (**1** is +5v and **0** is 0v)

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

(1) *set input voltages*

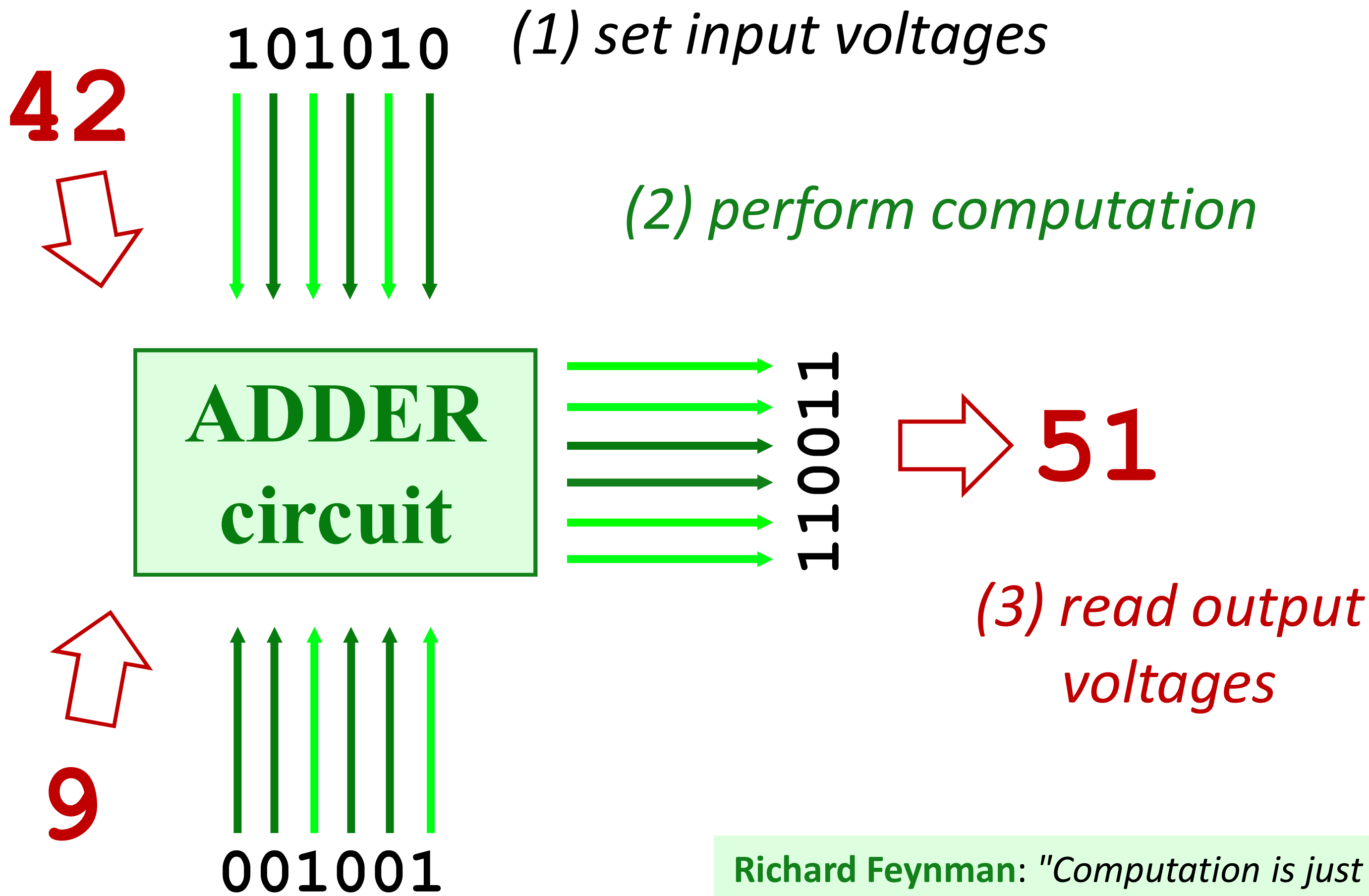
(2) perform computation



But what's this green thing?

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!



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

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**…*

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

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

AND

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

OR

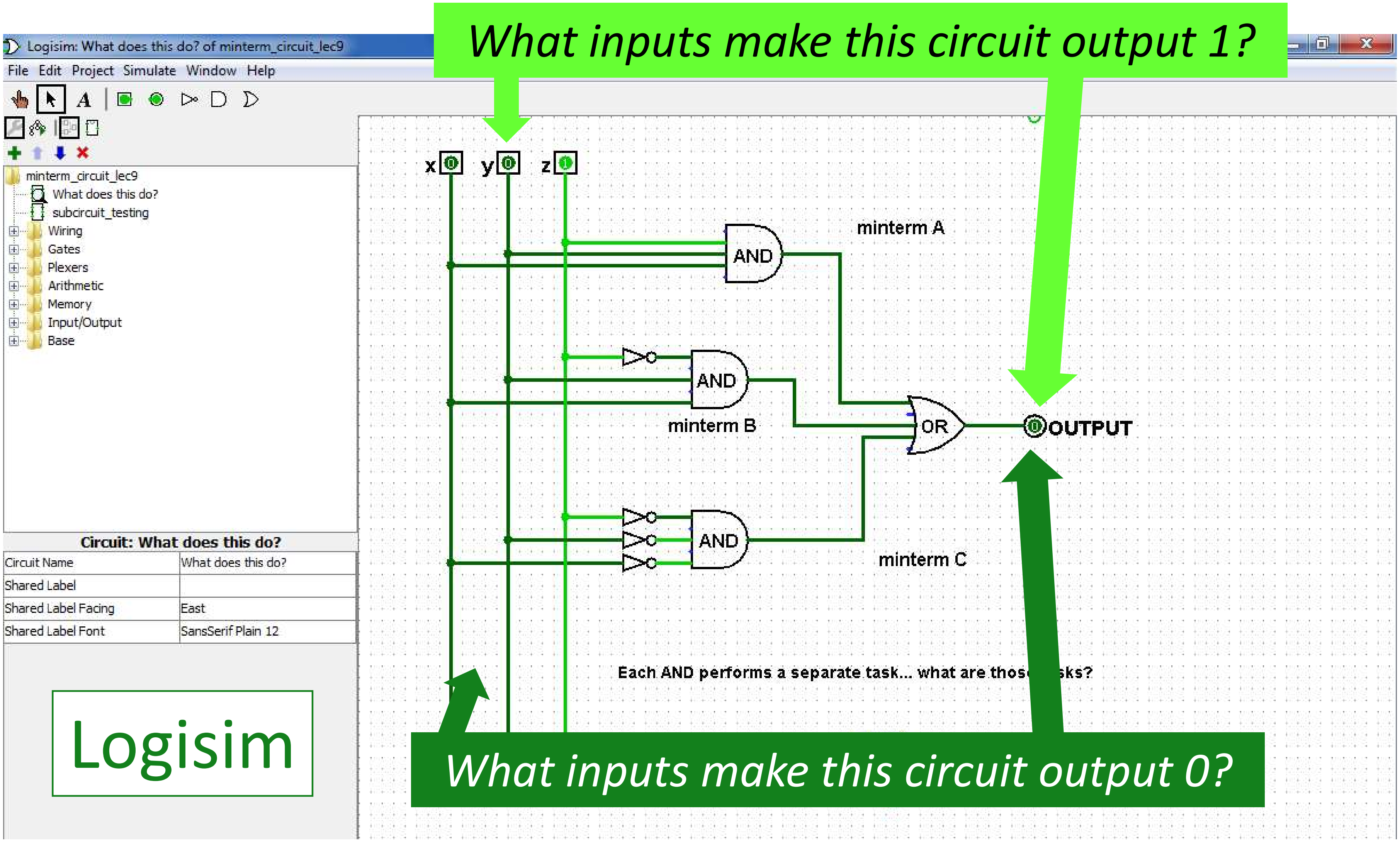
NOT reverses its input

NOT

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

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

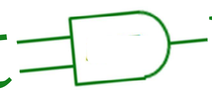
From gates to ***circuits***...



from circuit design…

next 2 weeks

…to a full computer!

Have an outsting and ftuitous week(end)!

Why  ?!