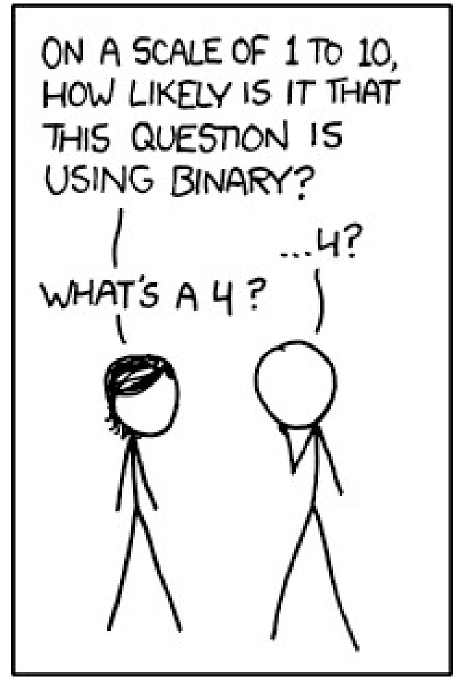
CS 101 Today...

Our top-10 list of binary jokes:

Jotto Corner

5C guess	ZD guess	HS guess	ZD guess
camel: 4	diner: 2	human: 1	diner: 2
?????: ?	savvy: ?	??????: ?	savvy: ? 1
		slaps 2	

Olivia Deanna



Looking Back ←

Looking Forward →

Computing as composition

clay == **functions**

Computing as representation

clay == **data & *bits***

Language Quiz: Are You on Fleek?

By WILSON ANDREWS and JOSH KATZ FEB. 22, 2015

Yolo. Rekt. Bae. Xans. Lordt. Every era has its own version of emerging language, and the new words and phrases of our time tend to spring from the Internet – from emails, texts, tweets and other rapid-fire, written communication.

They're often acronyms or abbreviations. Some become enduring parts of communication – as O.K., P.S. and R.S.V.P. did, from earlier times – while others flare briefly and then fade.

tfw

"To flirt with" "That feel when" "Twerk for what"

6/12

tfw you take a quiz and you just get rekt

Speaking of ***language!?***

Language Quiz: Are You on Fleek?

By WILSON ANDREWS and JOSH KATZ   FEB. 22, 2015

Yolo. Rekt. Bae. Xans. Lordt. Every era has its own version of emerging language, and the new words and phrases of our time tend to spring from the Internet – from emails, texts, tweets and other rapid-fire, written communication.

They're often acronyms or abbreviations. Some become enduring parts of communication – as O.K., P.S. and R.S.V.P. did, from earlier times – while others flare briefly and then fade.

boolin

Relaxing | Driving | Lying

An alteration of *coolin'*.

Justine Skye
@JustineSkye

back in brooklyn boolin, back to business

6:51 PM - 10 Feb 2015

33 RETWEETS 96 FAVORITES

Language Quiz: Are You on Fleek?

By WILSON ANDREWS and JOSH KATZ   FEB. 22, 2015

Yolo. Rekt. Bae. Xans. Lordt. Every era has its own version of emerging language, and the new words and phrases of our time tend to spring from the Internet – from emails, texts, tweets and other rapid-fire, written communication.

They're often acronyms or abbreviations. Some become enduring parts of communication – as O.K., P.S. and R.S.V.P. did, from earlier times – while others flare briefly and then fade.

boolin

Relaxing | Driving | Lying

```
>>> answer == 42
True
>>> not answer == 42
False
>>> type( answer == 42 )
<type 'bool'>
```

back to Python, boolin'...

Some legs to stand on... ?

This is heady stuff!

decipher

max

encipher

sScore

letScore

rot(c,n)

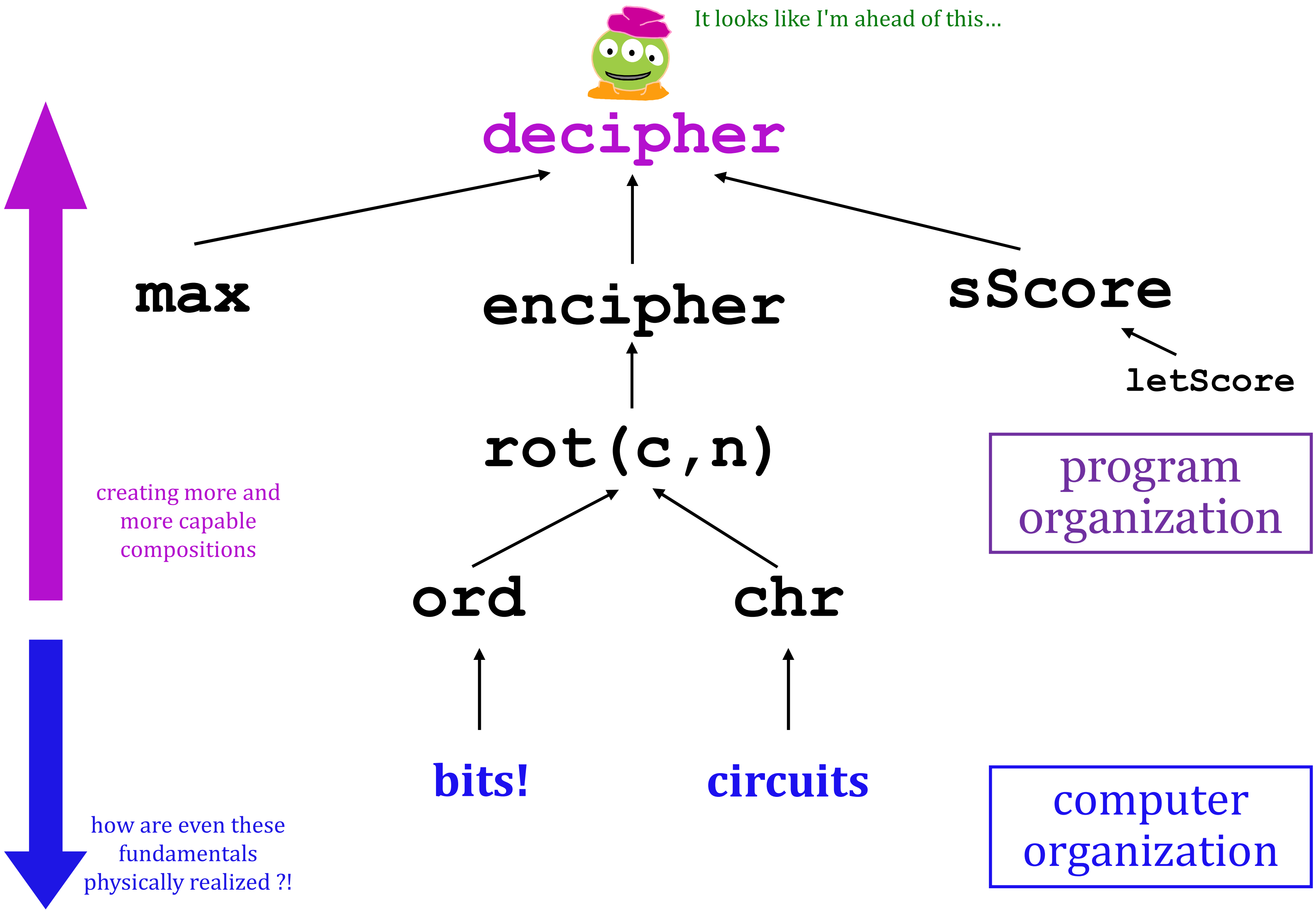
ord

chr

creating more and
more capable
compositions

program
organization

Some legs to stand on!

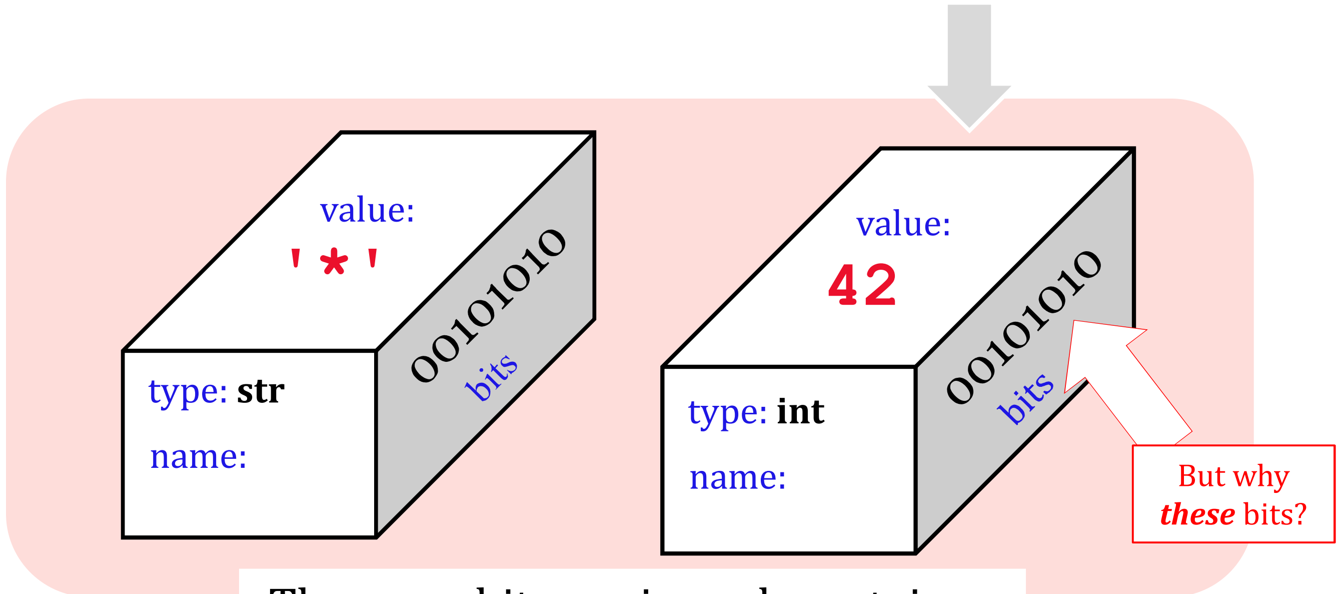


Binary Storage & Representation

Binary	Dec	Hex	Glyph
0010 0000	32	20	(blank) (SP)
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+

The SAME bits can represent different pieces of data, depending on **type**

8 bits = 1 byte = 1 box



The same bits are in each container.

What *is* 42 ?

`42`

It's ***not*** this!

forty two

Value!

`42`

What *is* 42 ?

Syntax.

forty two

value

42

syntax

tens

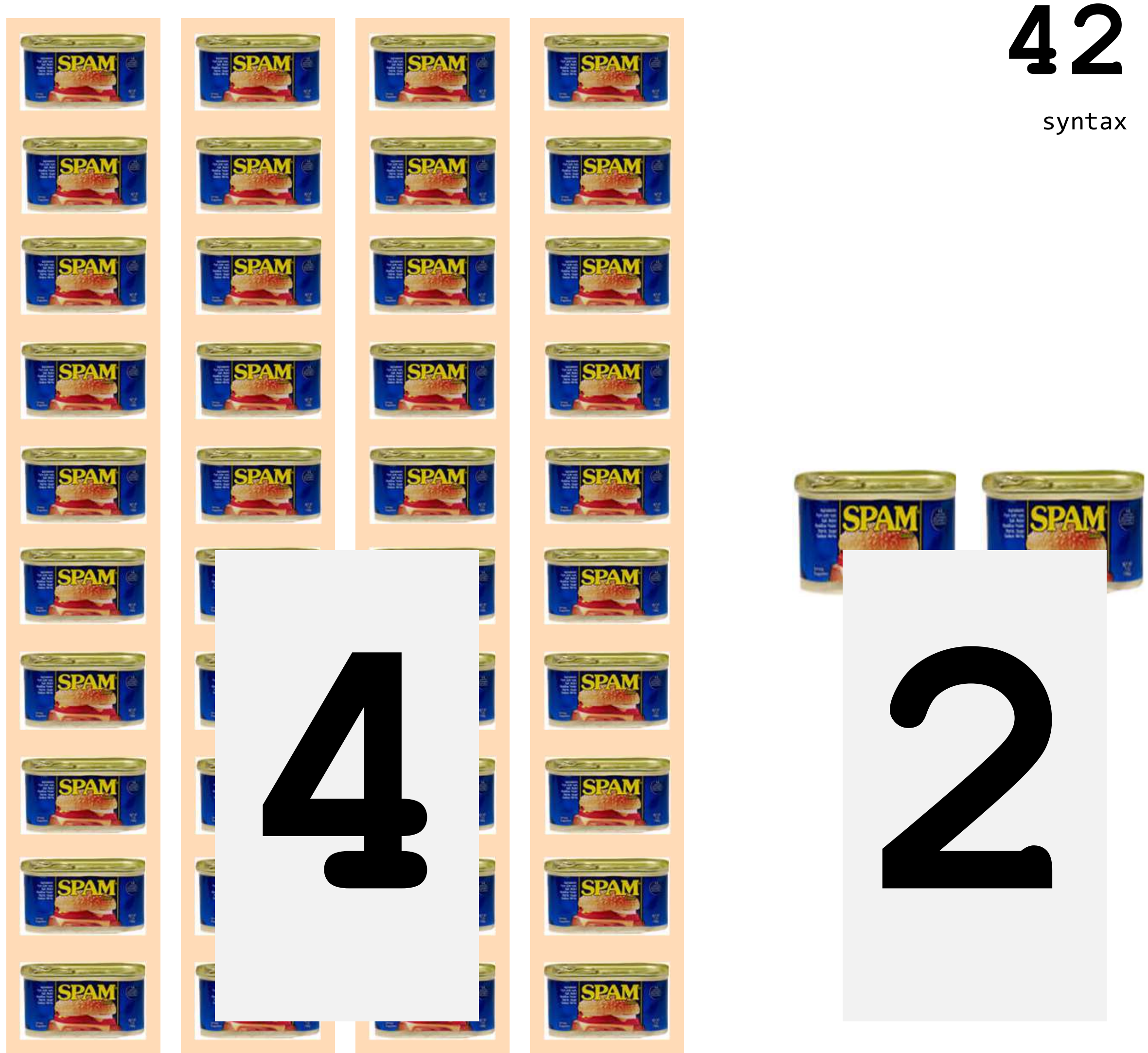
ones

forty two

value

42

syntax



tens

ones

forty two

value

Value

(semantics)

stuff we care about
(what things *mean*)

tens

ones

42

syntax

Syntax

stuff we use *to communicate*

forty two

value

Same Value!

forty two

value

Same Value!

thirty-twos	sixteens	eights	fours	twos	ones

but, <u>different</u> syntax...

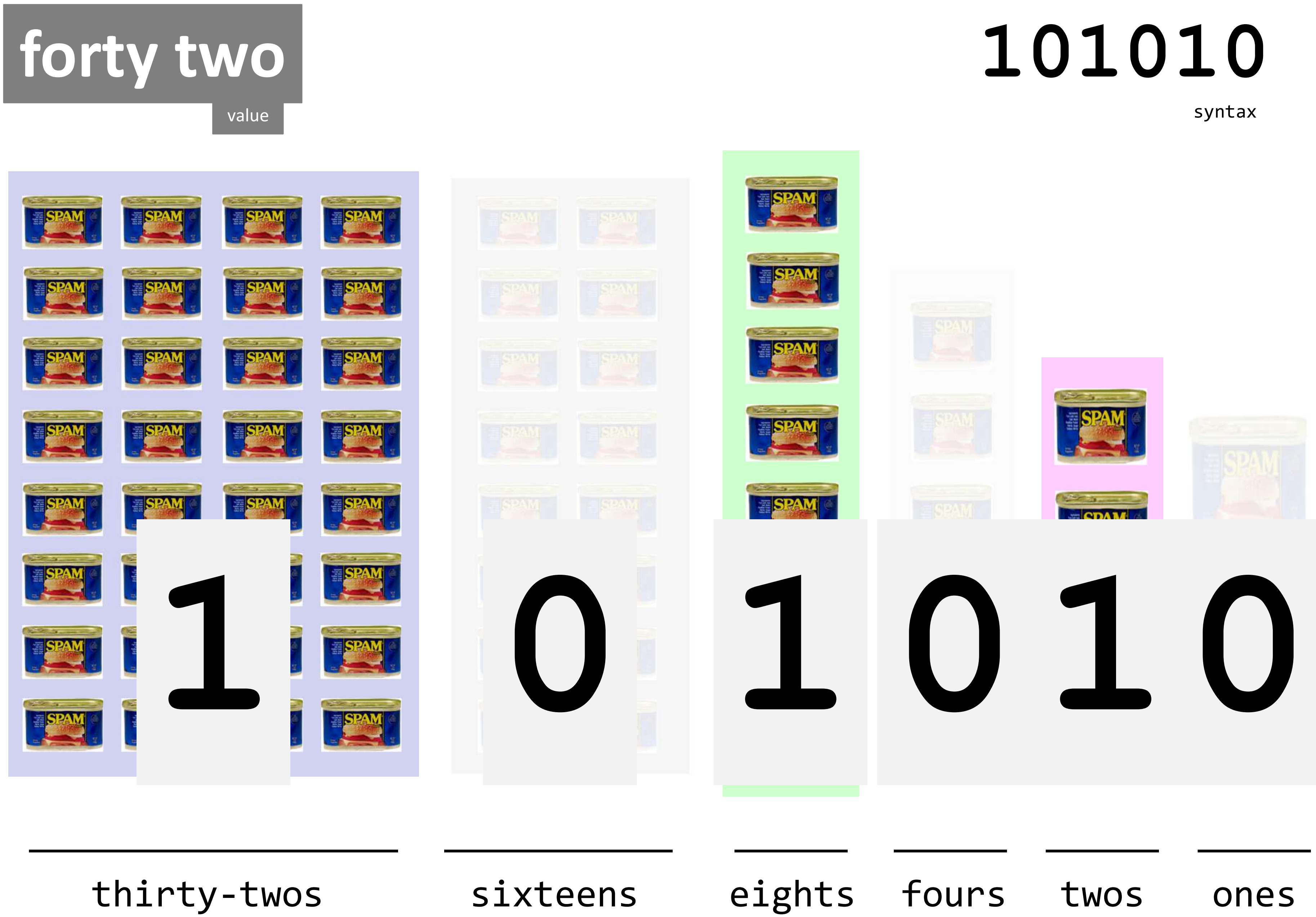
forty two

value

101010

syntax

thirty-twos | sixteens | eights | fours | twos | ones


forty two
value
101010
syntax
1
0
1
0
1
0
thirty-twos
sixteens
eights
fours
twos
ones

Base 2

"binary"

Base 10

"decimal"

Syntax

the symbols used
(what things look like)

THIRTYTWOs col.
SIXTEENs col.
EIGHTs column
FOURs column
TWOs column
ONEs column

101010_2

TENs column
ONEs column

42_{10}

forty two

value

Value

stuff we care about
(what things *mean*)

Base 2
"binary"

Base 10
"decimal"

Different

Syntax
the symbols used
(what things look like)

THIRT
SIXTEEN
EIGHTs co
FOURs column
TWOs column
ONEs column

101010_2

TENs column
ONEs column

42_{10}

Same!

forty two
value

Value
stuff we care about
(what things *mean*)

Base 2
"binary"

THIRTYTWOs col. SIXTEENs col. EIGHTs column FOURs column TWOs column ONEs column

101010_2

128's column SIXTYFOURs col THIRTYTWOs col. SIXTEENs col. EIGHTs column FOURs column TWOs column ONEs column

_ _ _ _ _ _ _ _

writing 123 in binary...

Base 10
"decimal"

TENs column ONEs column

42_{10}

4 tens + 2 ones

each column represents the base's next power

HUNDREDs column TENs column ONEs column

123_{10}

1 hundred + 2 tens + 3 ones

Binary math

Decimal math

tables of
one-digit
facts

+

Addition

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

*

Multiplication

×	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2		4	6	8	10	12	14	16	18	20
3			9	12	15	18	21	24	27	30
4				16	20	24	28	32	36	40
5					25	30	35	40	45	50
6						36	42	48	54	60
7							49	56	63	70
8								64	72	80
9									81	90
10										100

www.youtube.com/watch?v=Nh7xapVB-Wk

Name(s): ______________________________

Quiz

In binary, I'm an 11-eyed alien!

Convert these two binary numbers ***to decimal***:

```
32 16 8 4 2 1
 1  1 0 0 1 1        10001000
```

Convert these two decimal numbers ***to binary***:

```
32 16 8 4 2 1
```

28_{10} 101_{10}

Add these two binary numbers:

```
  101101
+   1110
```

Multiply these binary numbers:

```
  101101
*   1110
```

+

WITHOUT converting to decimal !

Hint: Remember these algorithms? They're the same in binary!

```
    1
  529
+ 742
 1271
```

```
   529
*   42
  1058
+ 2116
 22218
```

Extra! Can you figure out the last binary digit (bit) of 53 ***without determining any other bits?*** The last two? 3?

Convert these two binary numbers ***to decimal***:

32	16	8	4	2	1
1	1	0	0	1	1

32 + 16 + 2 + 1

51

128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

128 + 8

136

values in blue

Convert these two decimal numbers ***to binary***:

28

32	16	8	4	2	1
0	1	1	1	0	0

101_{10}

128	64	32	16	8	4	2	1
0	1	1	0	0	1	0	1

syntax in orange

Extra! Can you figure out the last binary digit (bit) of 53 ***without determining any other bits?*** The last _two_? _3_?

We'll return to this ***in a bit***...

Add these two binary numbers
WITHOUT converting to decimal !

32	16	8	4	2	1	
1	0	1	1	0	1	45
+		1	1	1	0	14
						59
32	16	8	4	2	1	

Hint:

$$\begin{array}{r} {}^{1} \\ 529 \\ +\ 742 \\ \hline 1271 \end{array}$$

Do you remember this algorithm? It's the same!

Add these two binary numbers
WITHOUT converting to decimal !

$$\begin{array}{rcccccc|l} & 32 & 16 & 8 & 4 & 2 & 1 & \\ & 1 & 0 & 1 & 1 & 0 & 1 & 45 \\ + & & & 1 & 1 & 1 & 0 & 14 \\ \hline \end{array}$$

Hint:

$$\begin{array}{r} {}^{1} \\ 529 \\ +\ 742 \\ \hline 1271 \end{array}$$

Do you remember this algorithm? It's the same!

Add these two binary numbers
WITHOUT converting to decimal !

```
        1 1
  32 16  8  4  2  1
   1  0  1  1  0  1     45
+        1  1  1  0     14
---------------------
   1  0  1  1  0  1     59
```

Hint:

```
  1
 529
+742
----
1271
```

Do you remember this algorithm? It's the same!

Multiply these two binary numbers
WITHOUT converting to decimal !

```
   32  16   8   4   2   1
    1   0   1   1   0   1     45
*           1   1   1   0     14
-------------------------
```

Hint:

```
   529
*   42
------
  1058
+2116
------
 22218
```

Do you remember this algorithm? It's the same!

630

Goal

A machine could -
and probably ***should***
- **be doing this !**

Multiply these two binary numbers
WITHOUT converting to decimal !

```
     32  16   8   4   2   1
      1   0   1   1   0   1        45
*             1   1   1   0        14
-----------------------------
      0   0   0   0   0   0
  1   0   1   1   0   1   0
1 0   1   1   0   1   0   0
+ 1 0 1 1 0 1 0 0 0
-----------------------------
1 0 0 1 1 1 0 1 1 0                630
512 256 128 64 32 16 8 4 2 1
```

"partial products"

Goal

Hint:

```
   529
*   42
------
  1058
+2116
------
 22218
```

Do you remember this algorithm? It's the same!

A machine could -
*and probably **should***
- **be doing this !**

Beyond Binary

base 1 111111111111111111111111111111111111111111 digits: **1**

	32	16	8	4	2	1	
base 2	1	0	1	0	1	0	digits: **0, 1**
			27	9	3	1	
base 3							digits: **0, 1, 2**

42 ?

There are 10 kinds of "people" in the universe:

those who know ternary,
those who don't, and
those who think this is a binary joke!

base 1 11111111111111111111111111111111111111111111 digits: 1

Beyond Binary

32 16 8 4 2 1

base 2 —— 101010 digits: 0, 1

27 9 3 1

base 3 —— 1120 digits: 0, 1, 2

16 4 1

base 4

base 5

base 6

base 7

base 8

base 9

base 10 42 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

base 11

base 12

. . .

base 16

Which of these ***isn't*** 42...?

222 60 54 46 39

and what are the ***bases*** of the rest?

Beyond Binary

base 1 111111111111111111111111111111111111111111 digits: **1**

base 2 —— 101010 (place values: 32 16 8 4 2 1) digits: **0, 1**

base 3 —— 1120 (place values: 27 9 3 1) digits: **0, 1, 2**

base 4 (place values: 16 4 1)

base 5

base 6

base 7

base 8

base 9

base 10 42 digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

base 11

base 12

. . .

base 16

Which of these ***isn't*** 42…?

222 **60** **54** **46** **39**

and what are the ***bases*** of the rest?

16s 4s 1s 16 4s 1s

7s 1s 7

9s 1s 9

11s 1s 11

42

base 1		111111111111111111111111111111111111111111	digits: 1
base 2	128 64 32 16 8 4 2 1	101010	digits: 0, 1
base 3	81 27 9 3 1	1120	digits: 0, 1, 2
base 4	64 16 4 1	222	digits: 0, 1, 2, 3
base 5	125 25 5 1	132	digits: 0, 1, 2, 3, 4
base 6	216 36 6 1	110	digits: 0, 1, 2, 3, 4, 5
base 7	49 7 1	60	digits: 0, 1, 2, 3, 4, 5, 6
base 8	64 8 1	52	digits: 0, 1, 2, 3, 4, 5, 6, 7
base 9	81 9 1	46	digits: 0, 1, 2, 3, 4, 5, 6, 7, 8
base 10	100 10 1	42	digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
base 11	121 11 1	39	digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A
base 16	256 16 1	2A	Hexadecimal digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

All 42s!

base 1 111111111111111111111111111111111111111111 digits: **1**

128 64 32 16 8 4 2 1

base 2 —— **101010** digits: **0, 1**

81 27 9 3 1

base 3 —— **1120** digits: **0, 1, 2**

All 42s!

Hexadecimal

256 16 1

base 16 —— **2A** digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**

base 1 111111111111111111111111111111111111111111 digits: 1

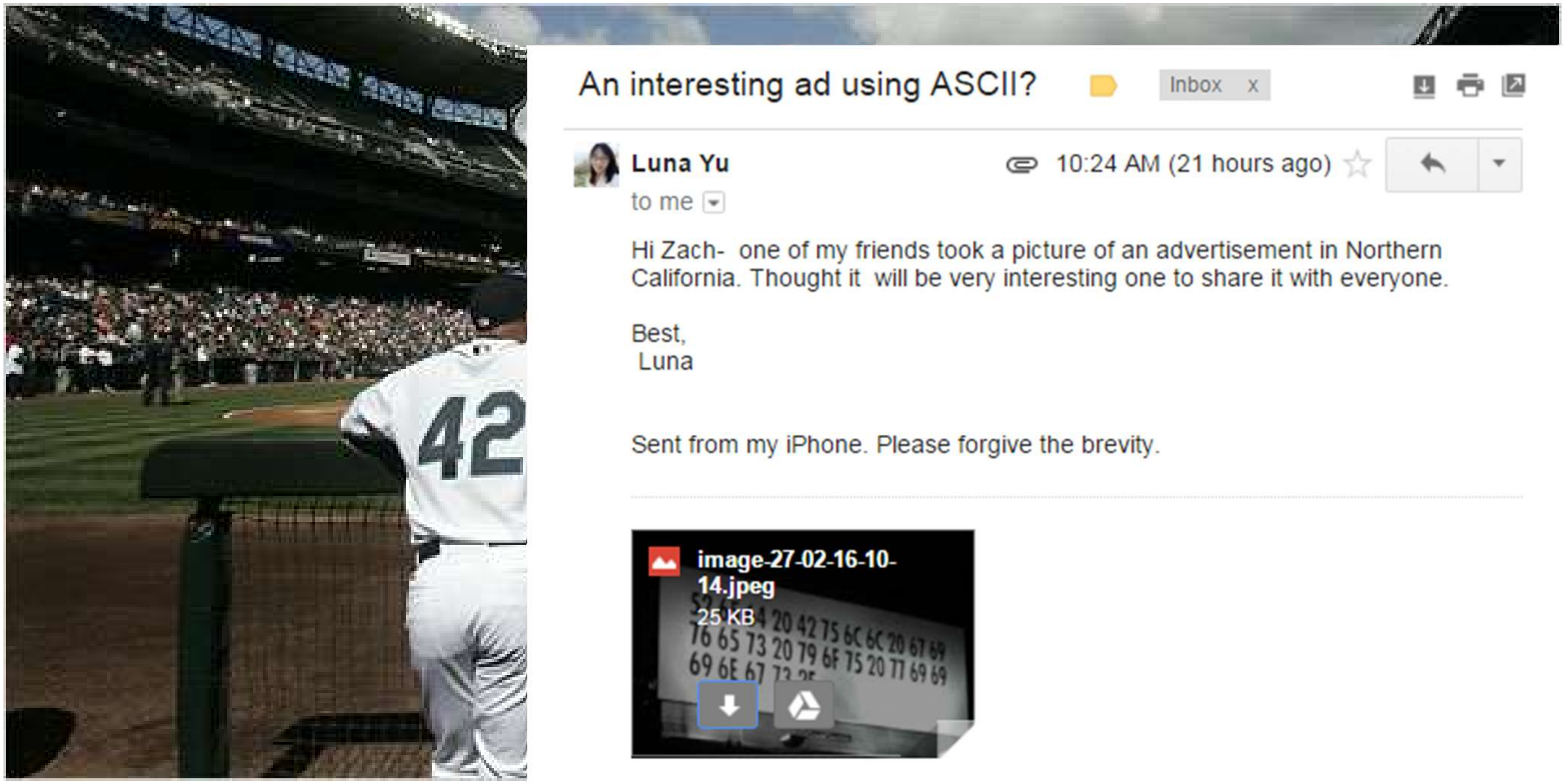
128 64 32 16 8 4 2 1

base 2 —— 101010 digits: 0, 1

81 27 9 3 1

base 3 —— 1120 digits: 0, 1, 2

All 42s!



Hexadecimal

256 16 1

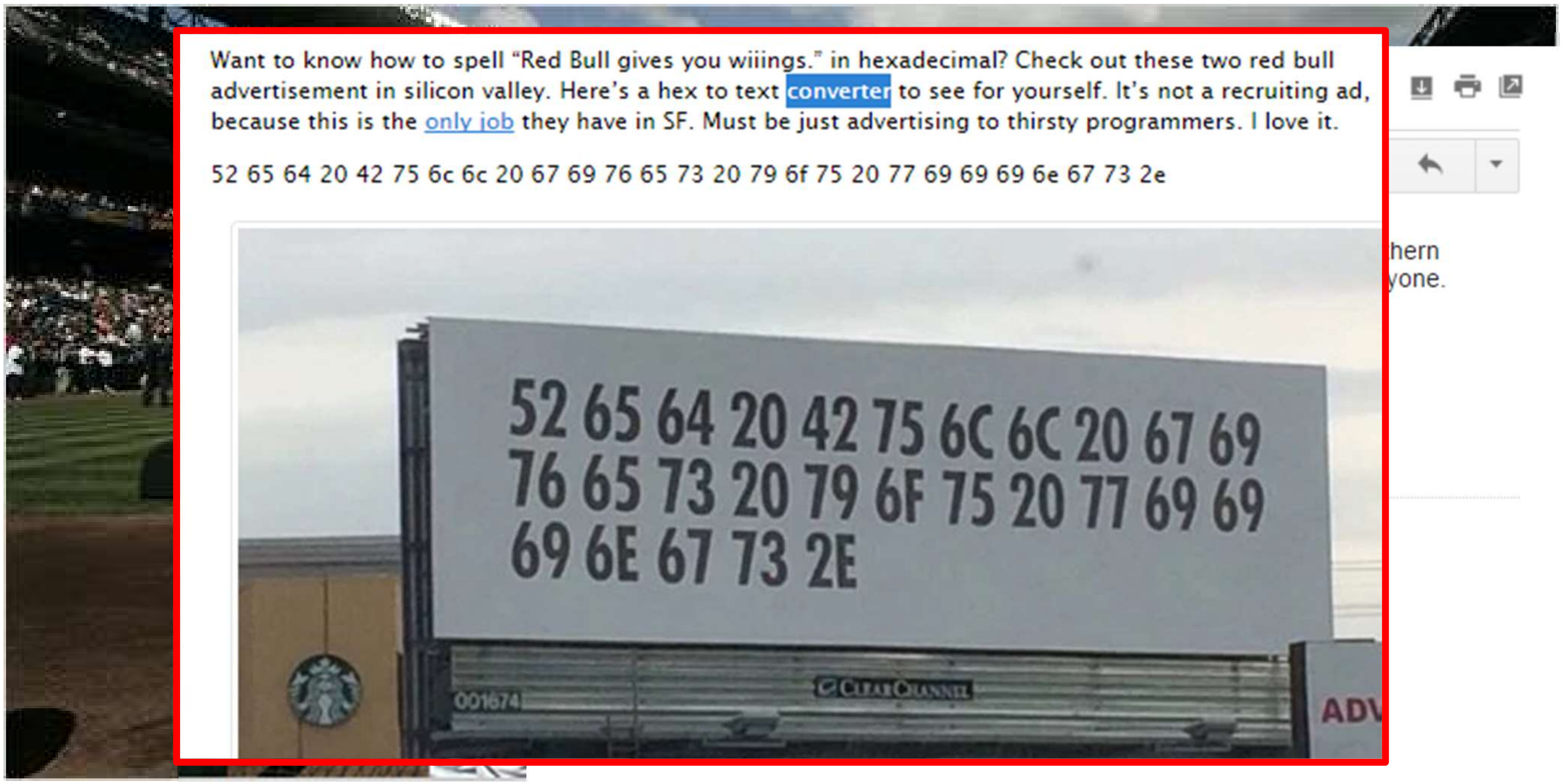
base 16 —— 2A digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

base 1 111111111111111111111111111111111111111111 digits: **1**

base 2 — 128 64 32 16 8 4 2 1 **101010** digits: **0, 1**

base 3 — 81 27 9 3 1 **1120** digits: **0, 1, 2**

All 42s!



Hexadecimal

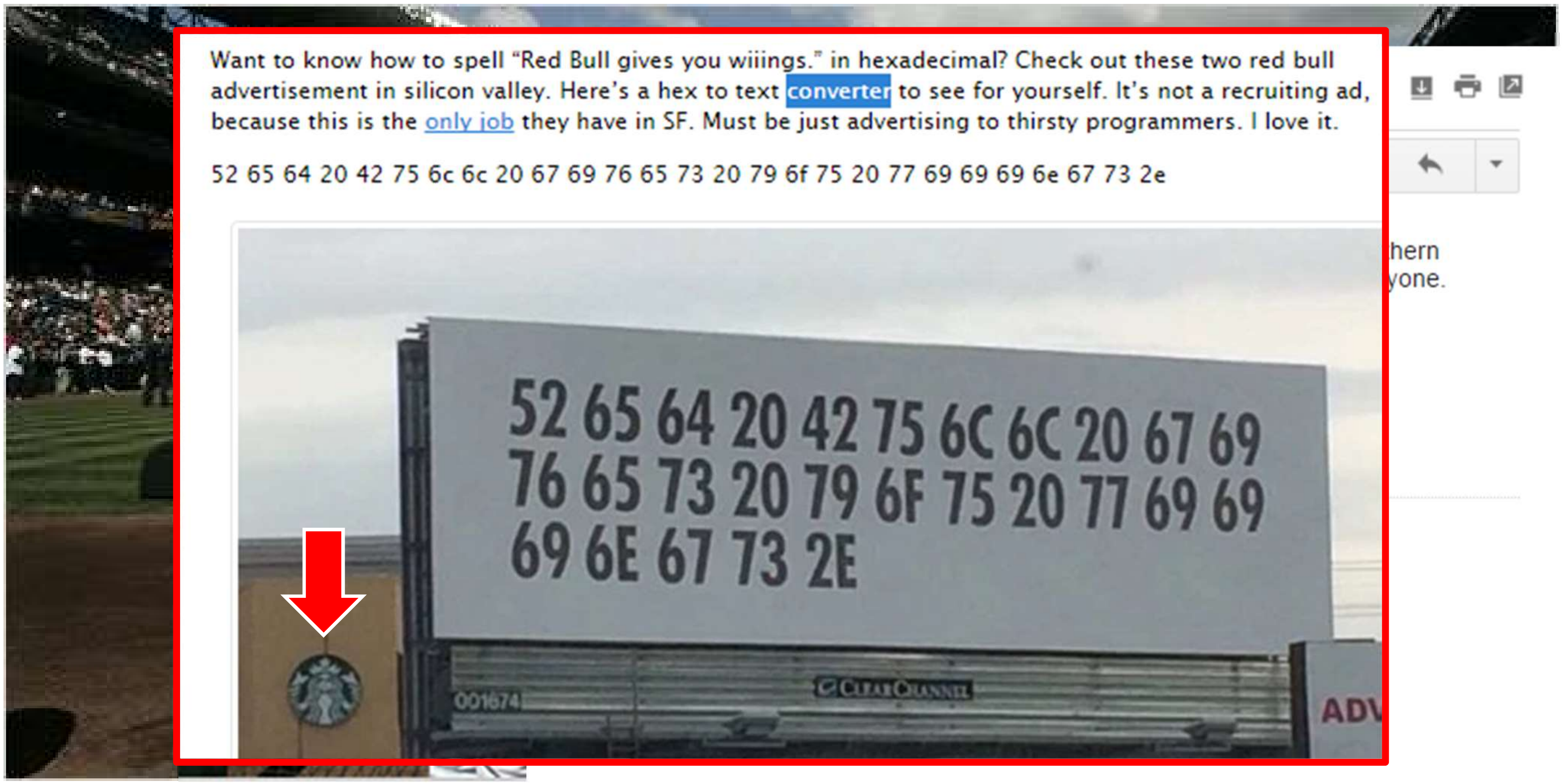
base 16 — 256 16 1 **2A** digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**

base 1 111111111111111111111111111111111111111111 digits: **1**

base 2 128 64 32 16 8 4 2 1
101010 digits: **0, 1**

base 3 81 27 9 3 1
1120 digits: **0, 1, 2**

All 42s!



base 16 256 16 1
2A

Hexadecimal

digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**

Off base?

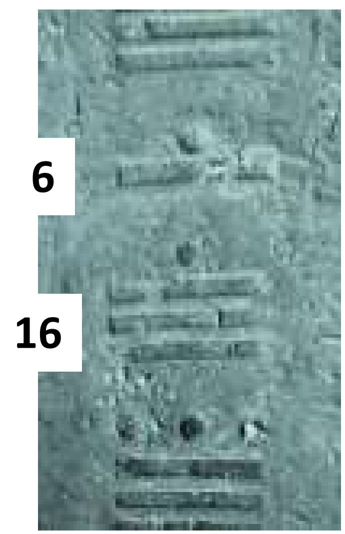
Our Mascot, the Panda

Base 12 –

"~~Duodecimal~~ Society"

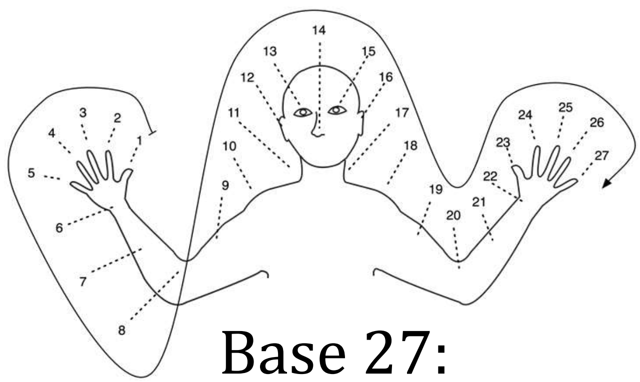
"***Dozenal*** Society"

Base 20: Americas



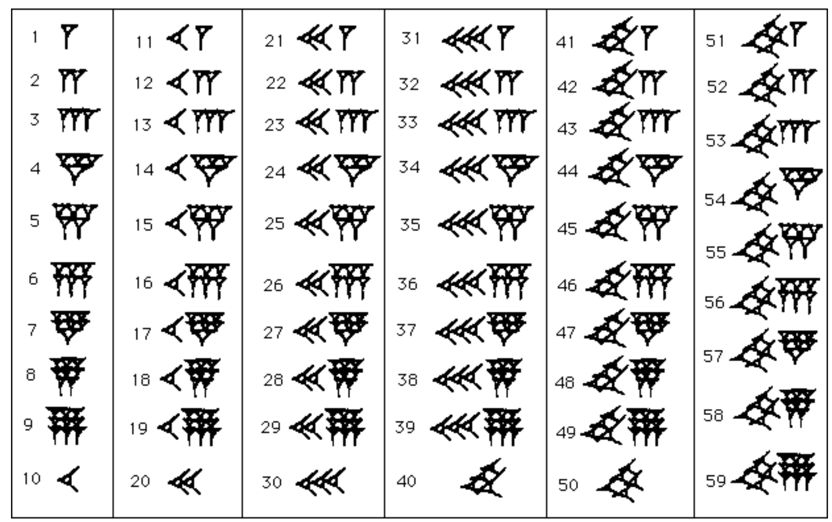
Olmec base-20 numbers
E. Mexico, ~ 300 AD

Telefol is a language spoken by the Telefol people in Papua New Guinea, notable for possessing a base-27 numeral system.



Base 27: New Guinea

Base 60 – Ancient Sumeria

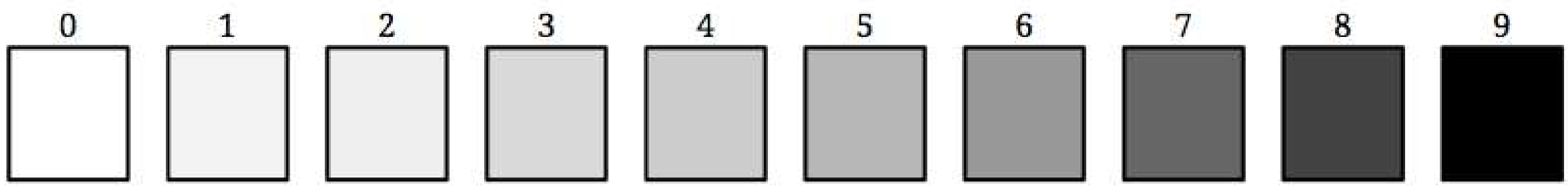


Some of these bases are still echoing around...

But ***why*** binary?

Ten symbols is too many!

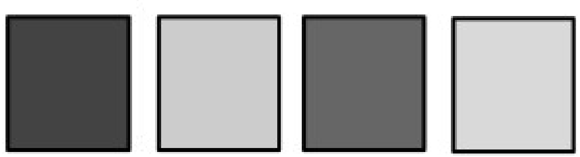
A computer has to differentiate *physically* among all its possibilities.



ten symbols ~ ten different voltages

This is too difficult to replicate billions of times

engineering!

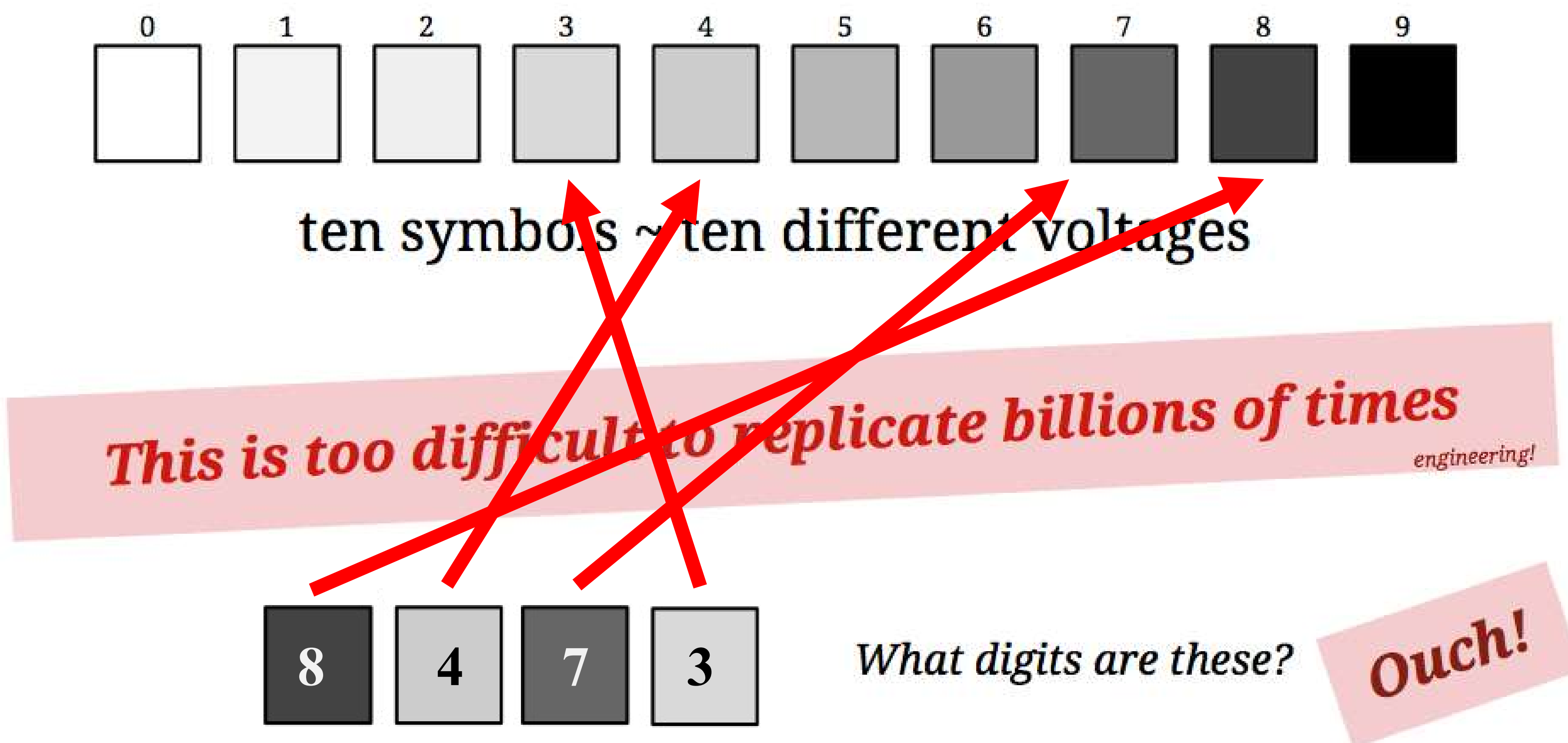


What digits are these?

Ouch!

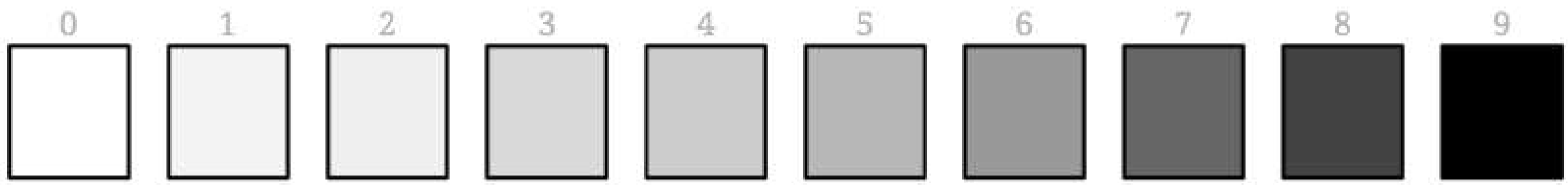
Ten symbols is too many!

A computer has to differentiate *physically* among all its possibilities.

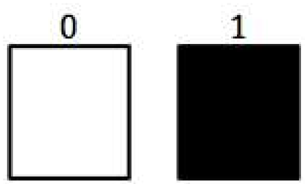


Two symbols is easiest!

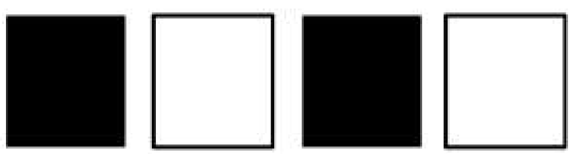
A computer has to differentiate *physically* among all its possibilities.



ten symbols ~ ten different voltages



two symbols ~ two different voltages

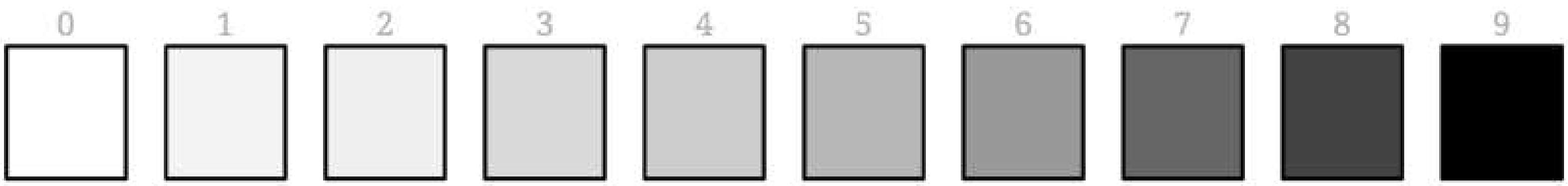


What digits are these?

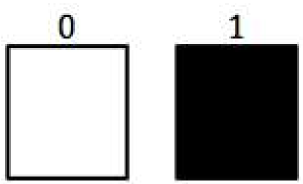
Easy!

Two symbols is easiest!

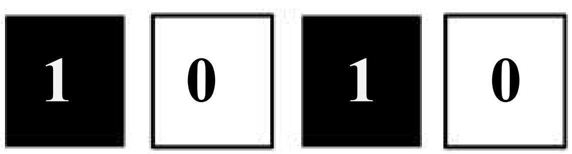
A computer has to differentiate *physically* among all its possibilities.



ten symbols ~ ten different voltages



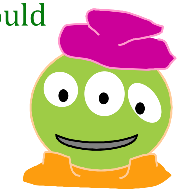
two symbols ~ two different voltages



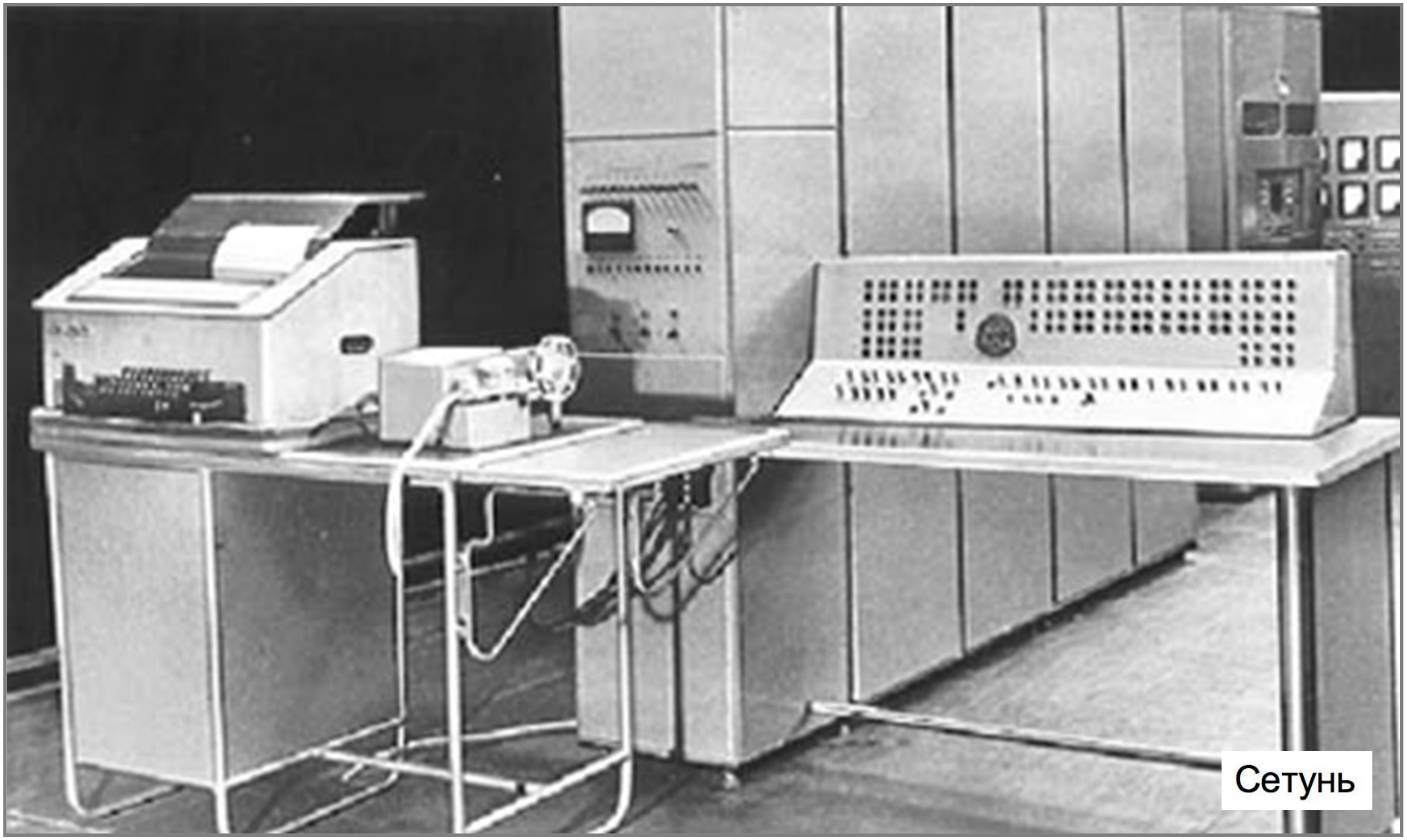
What digits are these?

Easy!

Ternary computers?

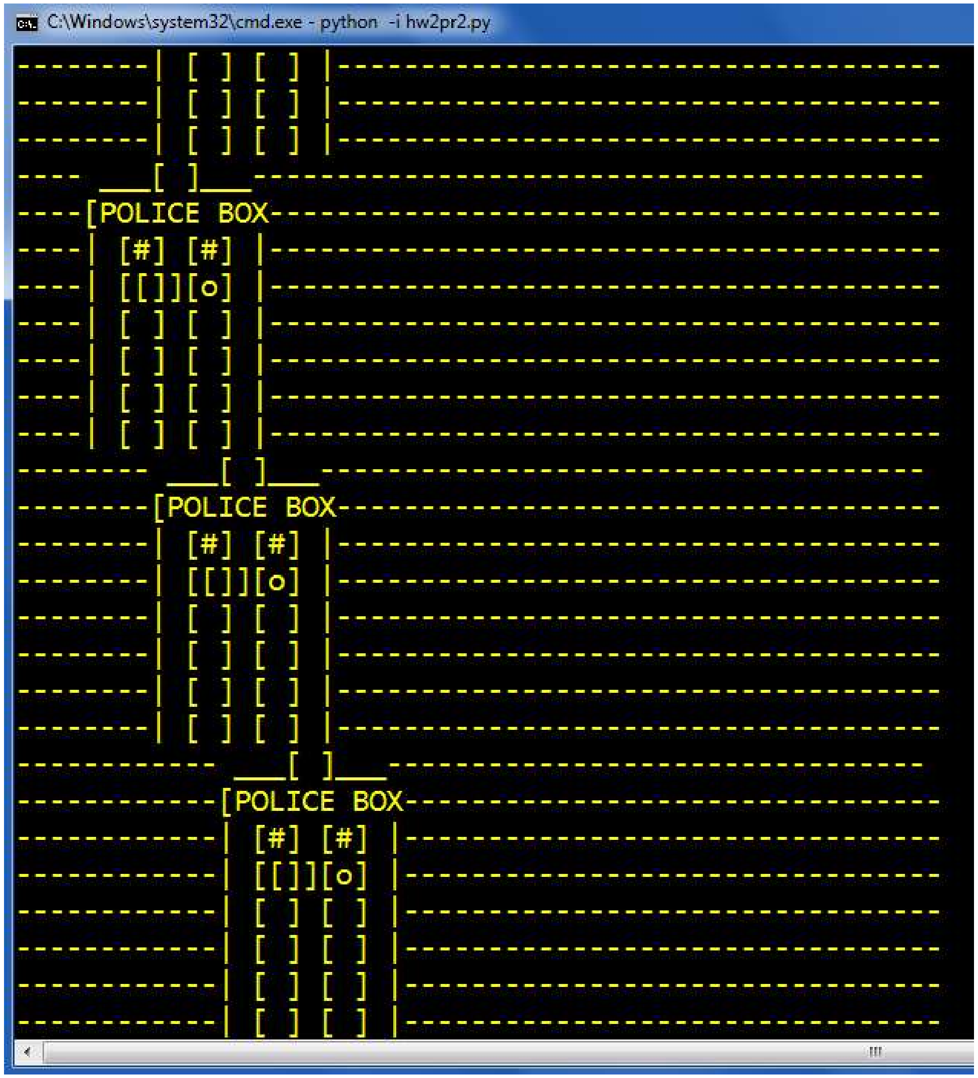


50 of these ***Setun*** ternary machines were made at Moscow U. ~ **1958**



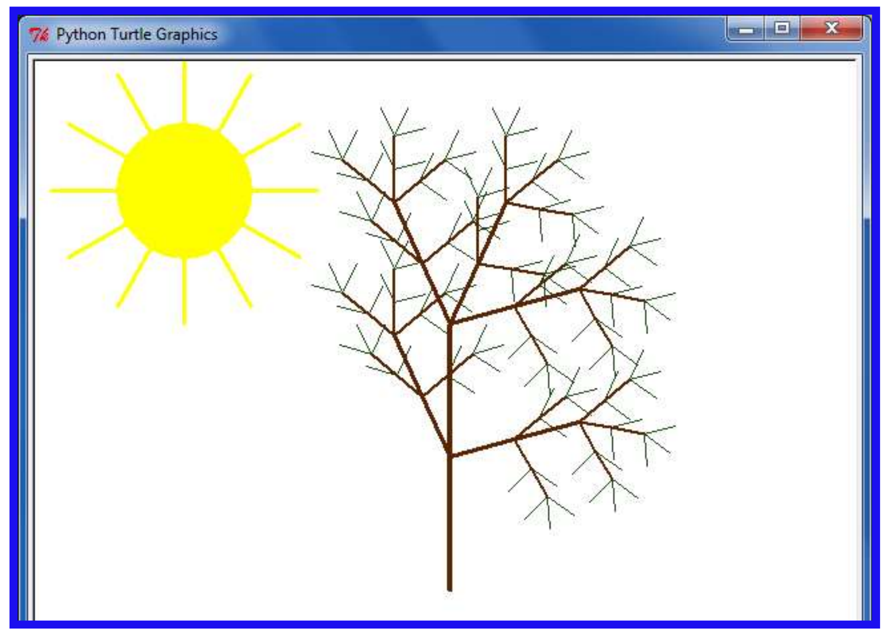
This project was discontinued in 1970… ***though not because of the ternary design!***

Eye-catching submissions...



ASCII wanderings...

and turtle art




Python Turtle Graphics
a turtle-drawn portrait from turtle graphics …

Whoa!
'12

Back to bits...

Reasoning ~ *Value* vs. *Syntax*

53

530

What does ***left-shifting*** do to the **value** of a <u>decimal</u> #?

537

53

What does ***right-shifting*** do to the **value** of a <u>decimal</u> #?

bitwise Python operators

<<

left-shift

>>

right-shift

Reasoning, *bit by bit*

<< left-shift

>> right-shift

left-shift by 1

11	3 << 1
110	6

What does ***left-shifting*** do to the **value** of a binary #?

left-shift by 2

11	3 << 2
1100	12

right-shift by 1

101010	42 >> 1
10101	21
1010	42 >> 2 ?

What does ***right-shifting*** do to the **value** of a binary #?

Reasoning, *bit by bit*

`<<`	`>>`	`&`	`|`
left-shift	right-shift	and	or

What do **and** and **or** do!?

and (both) `&` `|` or (either)

bitwise and

```
5:   101
6:   110
&    100
```

5 & 6

4

bitwise and

```
9: 1001
5: 0101
&
```

9 & 5

bitwise or

```
5:   101
6:   110
|    111
```

5 | 6

7

bitwise or

```
9: 1001
5: 0101
|
```

9 | 5

Intel x86 processor instructions
and their speeds (2016)

In processors **shift**, **and**, **or**, **add**, and **subtract** are ***much faster*** than **multiply**, **divide**, and **mod**, which are ***relatively* slow.**

Table C-16. General Purpose Instruction

Instruction	Latency[1] first time in a row	Throughput rest of times (in a row)
CPUID	0F_3H	0F_3H
ADC/SBB reg, reg	8	3
ADC/SBB reg, imm	8	2
ADD/SUB **and SHIFT**	1	0.5
AND/OR/XOR	1	0.5
BSF/BSR	16	2
BSWAP	1	0.5
BTC/BTR/BTS	8-9	1
CLI		
CMP/TEST	1	0.5
DEC/INC	1	0.5
IMUL r32	10	1
IDIV **MOD is the same**	66-80	30

Intel® 64 and IA-32 Architectures

Old Microsoft ***systems-interview*** question, #42:

42. Give a fast way to multiply a number by 7.

Being bit-*wise*

You **don't** need to convert to binary for these three...

`7 << 1`

left-shift

`5 << 4`

`170 >> 2`

right-shift

Try these for a bit...

You ***do*** need to use binary for these two!

```
14:   1110
 9:   1001
```

`14 | 9`

or

`14 & 9`

and

In today's processors shifts, and, or, add, and subtract are all ***very fast***, whereas multiplying, dividing, and mod are relatively **slow**.

Instruction	Latency
CPUID	**0F_3H**
ADC/SBB reg, reg	8
ADC/SBB reg, imm	8
ADD/SUB	1
AND/OR/XOR	1
BSF/BSR	16
BSWAP	1
BTC/BTR/BTS	8-9
CLI	
CMP/TEST	1
DEC/INC	1
IMUL r32	10
IDIV	66-80

With this in mind, how could we compute these expressions using *only* ***fast*** operations, maybe in combination?

`N//4`

`N*7`

`N*17`

`N%16`

extra fleek!

Being bit-*wise*

You **don't** need to convert to binary for these three...

`7 << 1` → `14`

left-shift

`5 << 4` → `80`

`170 >> 2` → `42`

right-shift

Try these for a bit...

You ***do*** need to use binary for these two!

```
14:   1110
 9:   1001
```

`14 | 9` → `15`   1111

or

`14 & 9` → `8`   1000

and

Instruction	Latency
CPUID	0F_3H
ADC/SBB reg, reg	8
ADC/SBB reg, imm	8
ADD/SUB	1
AND/OR/XOR	1
BSF/BSR	16
BSWAP	1
BTC/BTR/BTS	8-9
CLI	
CMP/TEST	1
DEC/INC	1
IMUL r32	10
IDIV	66-80

In today's processors shifts, and, or, add, and subtract are all ***very fast***, whereas multiplying, dividing, and mod are relatively **slow**.

With this in mind, how could we compute these expressions using *only* ***fast*** operations, maybe in combination?

`N//4`	`N >> 2`
`N*7`	`(N<<3) - N`
`N*17`	`(N<<4) + N`
`N%16`	`N-((N>>4)<<4)`

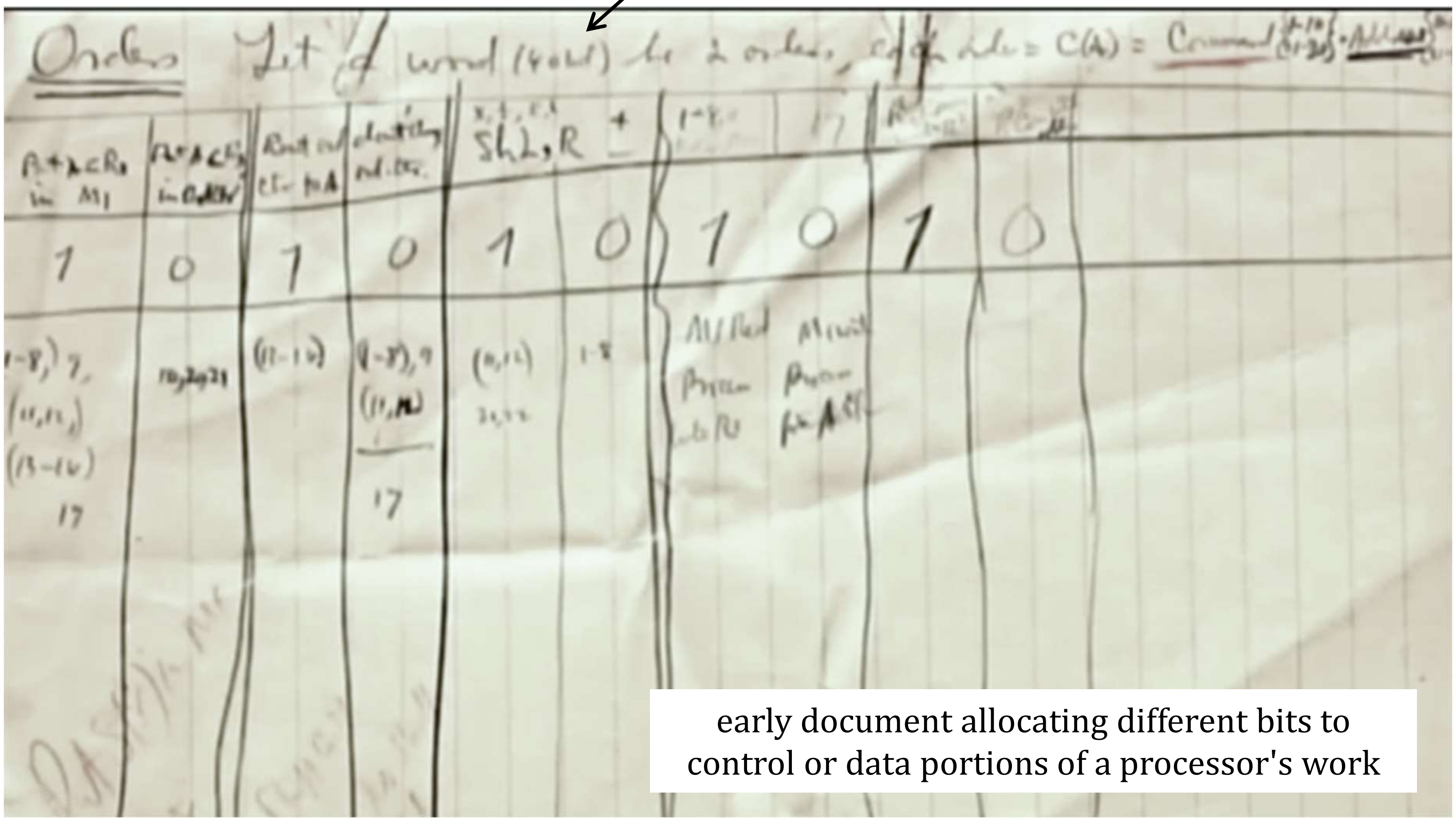
extra fleek!

Back to bits...

not the original name...

b.d. ~ binary digit ~ <u>**bit**</u>

"bit" first appeared in print in 1948 (Claude Shannon)



early document allocating different bits to control or data portions of a processor's work

Extra! Can you figure out the **<u>last binary digit</u>** (bit) of **53** *without determining any earlier bits*? The last ***<u>two</u>***? ***<u>three</u>***?

All of them?

Lab 4: *Computing in binary*

base 10		base 2
100 10 1 **141**	=	… 4 2 1

This first step of **left-to-right** conversion into binary is tricky to program... ***Why?***

Lab 4: *Computing in binary*

base 10		base 2
100 10 1 **141**	=	… 4 2 1

This first step of **left-to-right** conversion into binary is tricky to program... ***Why?***

It's tricky to find the largest power needed...

Lab 4: *Computing in binary*

base 10		base 2
100 10 1 **141**	=	64 32 16 8 4 2 1

What does the fact that 141 is ODD tell us?!

Let's run **right-to-left**!

141 = **10001101**

128 64 32 16 8 4 2 1

answer

Lab 4: *Computing in binary*

base 10		base 2
100 10 1		128 64 32 16 8 4 2 1
~~141~~	=	1 0 0 0 1 1 0 1

~~140~~
~~70~~
~~35~~
34
~~17~~
16
~~8~~
4
~~2~~
1

What does the fact that 141 is ODD tell us?!

Let's run **right-to-left**!

141	=	**10001101**
		128 64 32 16 8 4 2 1

answer

Lab 4: *Computing in binary*

base 10		base 2
100 10 1		128 64 32 16 8 4 2 1
~~141~~	=	1 0 0 0 1 1 0 1

~~140~~
~~70~~
~~35~~
34

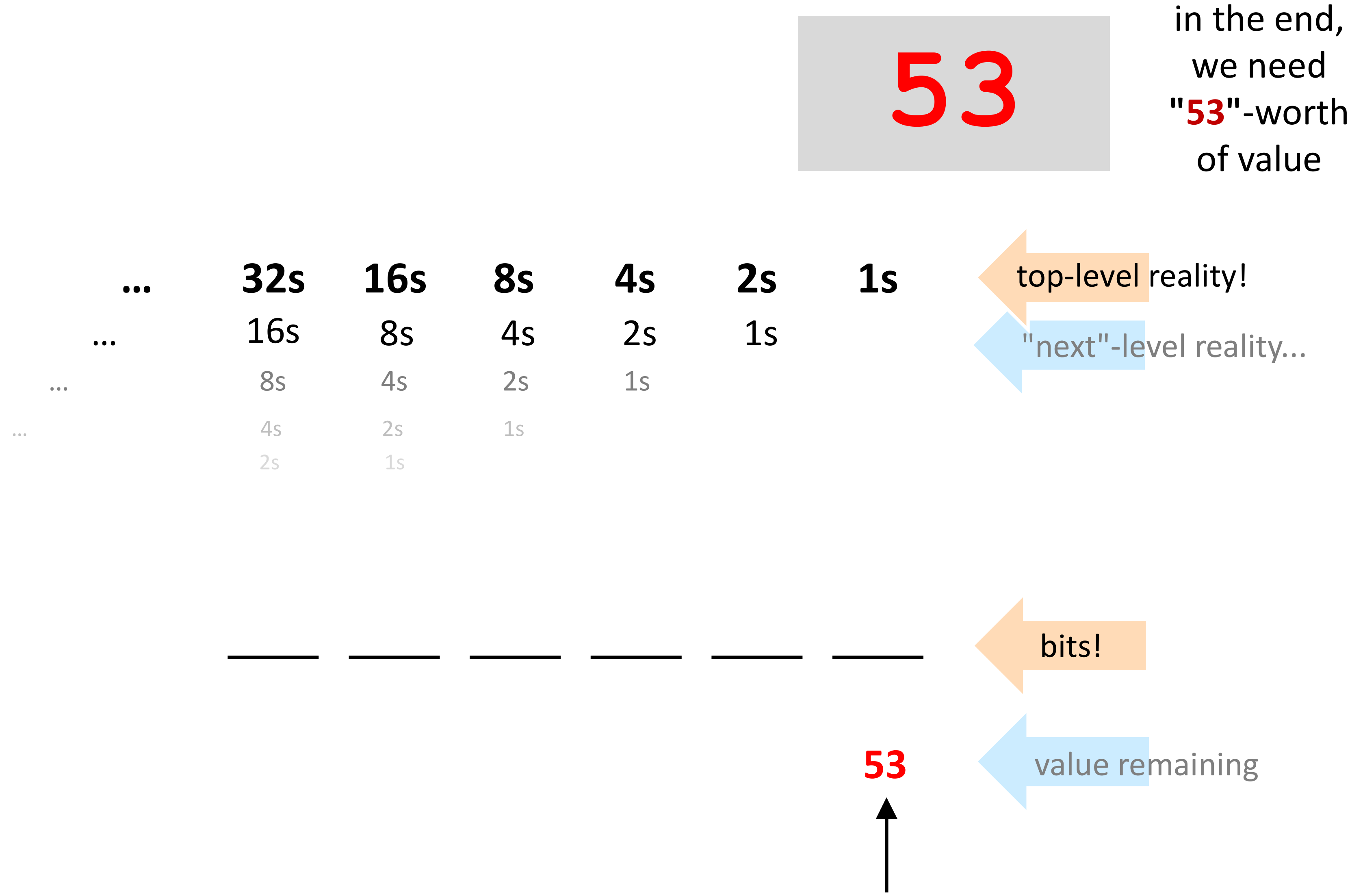
What does the fact that 141 is ODD tell us?!

Let's run **right-to-left**!

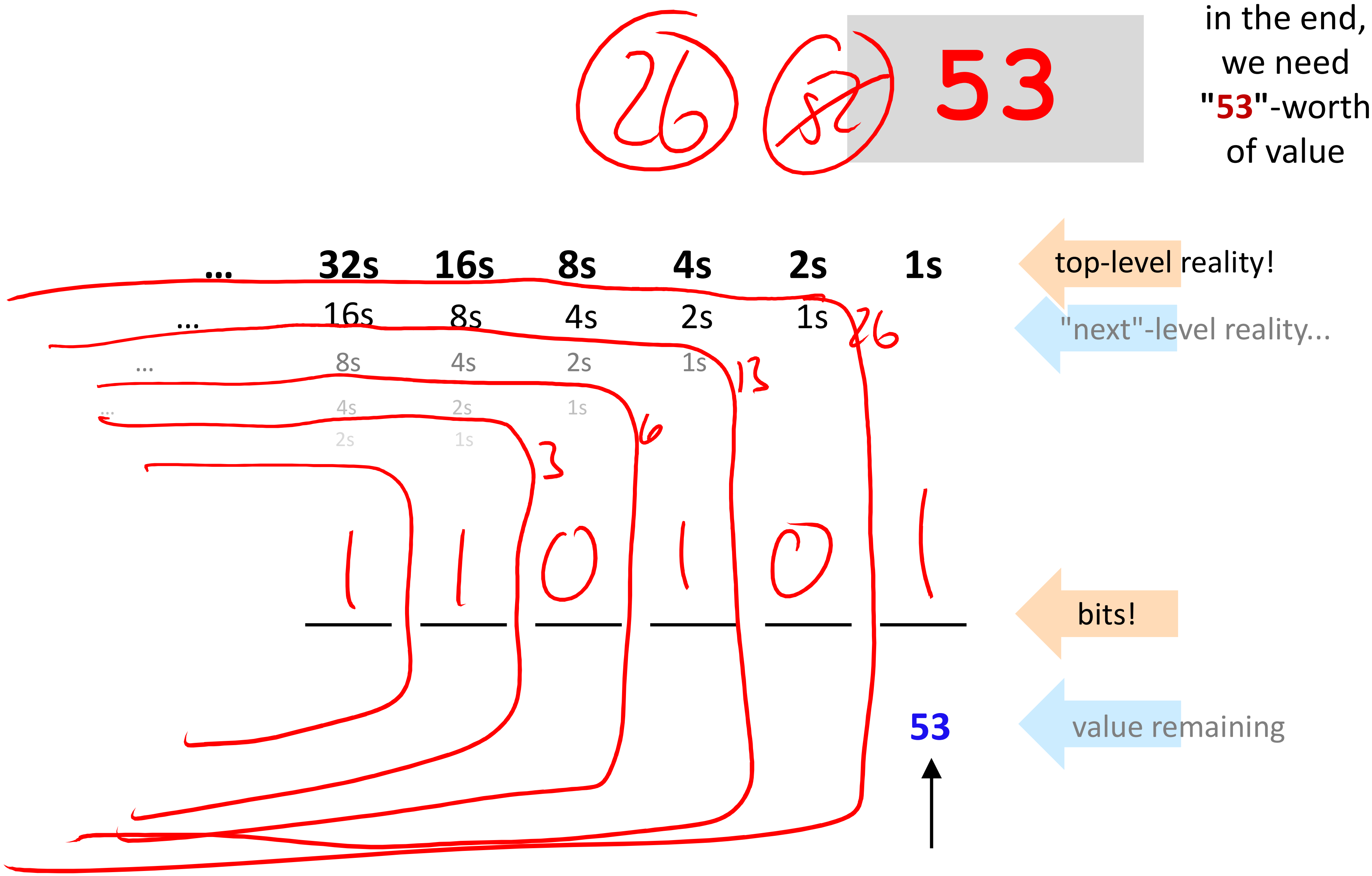
Why does this work?!

141	=	**10001101**
		128 64 32 16 8 4 2 1

answer



Converting to binary ~ starting from the right!



Converting to binary ~ starting from the right!

53

in the end,
we need
"**53**"-worth
of value

Extra! Can you figure out the **last binary digit** (bit) of **53** *without determining any earlier bits?* The last ***two***? ***three***?

All of them?

Lab 4: *Computing in binary*

base 10		base 2
100 10 1		128 64 32 16 8 4 2 1
~~141~~	=	1 0 0 0 1 1 0 1

~~140~~
~~70~~
~~35~~
~~34~~
~~17~~
~~16~~
~~8~~
~~4~~
~~2~~
1

What does the fact that 141 is ODD tell us?!

Let's run **right-to-left**!

141	=	**10001101**
		128 64 32 16 8 4 2 1

answer

Lab 4: *Computing in binary*

base 10		**base 2**
100 10 1		128 64 32 16 8 4 2 1
141	=	'10001101'

Right-to-left works!

You'll write these right! (-to-left)

numToBinary(N) **binaryToNum(S)**

decimal syntax, N

n2b(141)

we need to *represent* binary numbers with **strings**

b2n('10001101')

Lab 4: *Computing in binary*

base 10	base 2
100 10 1	128 64 32 16 8 4 2 1
141 =	'10001101'

```
def numToBinary( N ):
    if N == 0:
        return ''
    elif N%2 == 0:
        return  numToBinary(      ) +
    else:
        return  numToBinary(      ) +
```

empty string means 0

?

If N is even, what is the final bit?

If N is odd, what is the final bit?

How much VALUE is left to convert!?

Lab 4: *Fleek* binary conversion !

```
def numToBinary( N ):
    if N == 0: return ''
    else: return numToBinary(N//2) +
```

```
def numToBinary( N ):
    if N == 0:
        return ''
    elif N%2 == 0:
        return  numToBinary( N//2 )+ '0'
    else:
        return  numToBinary( N//2 )+ '1'
```

empty string means 0

If N is even, what is the final bit?

If N is odd, what is the final bit?

How much VALUE is left to convert!?

When traveling, always insist on ***bitwise*** accommodations ... !

See you at lab
– *in just a* ***bit*****!**

This room is a 10!

Insight: Ancient Egyptian Multiplication

Ancient Egyptian multiplication

From Wikipedia, the free encyclopedia

Next time?

Insight: Ancient Egyptian Multiplication

21 × 6 == 126

21 6

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?

---

11 × 15 == 165

11 15

Try it here

or *RPM…*

Buddy, can you spare an eye?

Base 2

Base 10

each column represents the base's next power

THIRTYTWOs col. SIXTEENs col. EIGHTs column FOURs column TWOs column ONEs column

101010_2

0
2
0
8
0
32

42

TENs column ONEs column

42_{10}

4 tens + 2 ones

123

128's column SIXTYFOURs col THIRTYTWOs col. SIXTEENs col. EIGHTs column FOURs column TWOs column ONEs column

1111011

_ _ _ _ _ _ _ _

Write 123 in binary...

HUNDREDs column TENs column ONEs column

123_{10}

1 hundred + 2 tens + 3 ones

123
-64
59
-32
27
-16
11

11
-8
3

Binary math

+	0	1
0	0	1
1	1	10

*	0	1
0	0	0
1	0	1

tables of basic facts

+

Addition

*****

Multiplication

Decimal math

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

×	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	[redacted]	4	6	8	10	12	14	16	18	20
3	[redacted]	[redacted]	9	12	15	18	21	24	27	30
4	[redacted]	[redacted]	[redacted]	16	20	24	28	32	36	40
5	[redacted]	[redacted]	[redacted]	[redacted]	25	30	35	40	45	50
6	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	36	42	48	54	60
7	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	49	56	63	70
8	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	64	72	80
9	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	81	90
10	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	100

www.youtube.com/watch?v=Nh7xapVB-Wk

Binary math

+	0	1
0	0	1
1	1	10 (two)

tables of basic facts

+

Addition

Decimal math

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

*	0	1
0	0	0
1	0	1

*****

Multiplication

×	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2		4	6	8	10	12	14	16	18	20
3			9	12	15	18	21	24	27	30
4				16	20	24	28	32	36	40
5					25	30	35	40	45	50
6						36	42	48	54	60
7							49	56	63	70
8								64	72	80
9									81	90
10										100

www.youtube.com/watch?v=Nh7xapVB-Wk

Binary math

Decimal math

tables of basic facts

+	0	1
0	0	1
1	1	10

two ones

+

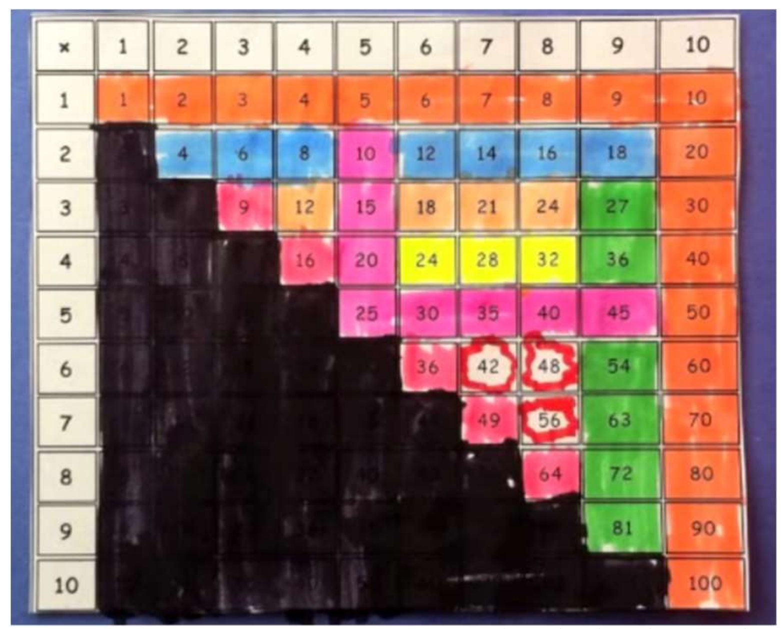
Addition

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

*	0	1
0	0	0
1	0	1

*****

Multiplication



×	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2		4	6	8	10	12	14	16	18	20
3			9	12	15	18	21	24	27	30
4				16	20	24	28	32	36	40
5					25	30	35	40	45	50
6						36	42	48	54	60
7							49	56	63	70
8								64	72	80
9									81	90
10										100

www.youtube.com/watch?v=Nh7xapVB-Wk

Beyond Binary

base 1 1111111111111111111111111111111111111111111 digits: **1**

16s 4s 1s 7s 1s

base 2 — 101010 digits: **0, 1**

(place values: 32 16 8 4 2 1)

base 3 — 1120 digits: **0, 1, 2**

(place values: 27 9 3 1)

base 4 (16 4 1)

base 5

base 6

base 7

base 8

base 9

base 10 42 digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

base 11

base 12

. . .

base 16

9s 1s 11s 1s

Which of these ***isn't*** 42…?

222 **60** **~~54~~** **46** **39**

and what are the ***bases*** of the rest?

Reasoning, *bit by bit*

<<	>>	&	|
left-shift	right-shift	and	or

and (both) **&** **|** or (either)

bitwise and

```
5:  101
6:  110
&   100
```

5 & 6

4

bitwise and

```
11: 1011
 5: 0101
&   0001
```

11 & 5

1

bitwise or

```
5:  101
6:  110
|   111
```

5 | 6

7

bitwise or

```
11: 1011
 5: 0101
|   1111
```

11 | 5

15

Reasoning, *bit by bit*

<<	>>	&	|
left-shift	right-shift	and	or

4

40

left-shift by 1

11

110

`3 << 1`

6

What does ***left-shifting*** do to the **value** of a #?

400

left-shift by 2

11

1100

`3 << 2`

12

What does ***right-shifting*** do to the **value** of a #?

right-shift by 1

101010

10101

`42 >> 1`

21

1010

`42 >> 2` 10

Being bit-*wise*

You **don't** need to convert to binary for these three...

`7 << 1`   14

left-shift

`5 << 4`

`170 >> 2`

right-shift

Try these for a bit...

14:	1110
9:	1001

You ***do*** need to use binary for these two!

`14 | 9`

or

`14 & 9`

and

In processors shifts, ands, ors, adds, and subtractions are ***very fast***, whereas multiplying, dividing, and mod, which are relatively **slow**.

Given this, what is a way to compute these expressions using *only* ***fast*** operations, maybe in combination?

`N//4`

`N*7`

`N*17`

`N%16`

extra fleek!

Being bit-*wise*

You **don't** need to convert to binary for these three...

`7 << 1` ⇨ `14`

left-shift

`5 << 4` ⇨ `80`

`170 >> 2` ⇨ `42`

right-shift

Try these for a bit...

```
14:   1110
 9:   1001
```

You ***do*** need to use binary for these two!

`14 | 9` ⇨ `15` 1111

or

`14 & 9` ⇨ `8` 1000

and

In processors shifts, ands, ors, adds, and subtractions are ***very fast***, whereas multiplying, dividing, and mod, which are relatively **slow**.

Given this, what is a way to compute these expressions using *only* ***fast*** operations, maybe in combination?

```
N//4
N*7
N*17
N%16
```

extra fleek!

Let's first look at ***why*** you'd bother ... !?

Intel x86 processor instructions
and their speeds (2016)

In processors shift, and, or, add, and subtract are ***much faster*** than multiply, divide, and mod, which are ***relatively* slow.**

Table C-16. General Purpose Instruction

Instruction	Latency[1] first time in a row	Throughput rest of times (in a row)
CPUID	0F_3H	0F_3H
ADC/SBB reg, reg	8	3
ADC/SBB reg, imm	8	2
ADD/SUB	1	0.5
AND/OR/XOR	1	0.5
BSF/BSR	16	2
BSWAP	1	0.5
BTC/BTR/BTS	8-9	1
CLI		
CMP/TEST	1	0.5
DEC/INC	1	0.5
IMUL r32	10	1
IDIV **MOD is the same**	66-80	30

Intel® 64 and IA-32 Architectures

Old Microsoft ***systems-interview*** question, #42:

42. Give a fast way to multiply a number by 7.

Intel x86 processor instructions
and their speeds (2014)

Table C-16. General Purpose Instructions

Instruction	Latency[1]	Throughput
CPUID	0F_3H	0F_3H
ADC/SBB reg, reg	8	3
ADC/SBB reg, imm	8	2
ADD/SUB	1	0.5
AND/OR/XOR	1	0.5
BSF/BSR	16	2
BSWAP	1	0.5
BTC/BTR/BTS	8-9	1
CLI		
CMP/TEST	1	0.5
DEC/INC	1	0.5
IMUL r32	10	1
IDIV **MOD is the same**	66-80	30

In processors shift, and, or, add, and subtract are ***much faster*** than multiply, divide, and mod, which are ***relatively* slow.**

Given this, what is a way to compute these statements using combinations from only the ***fast*** operations above?

N//4 ⇨ N >> 2

N*7 ⇨ N<<3 − N

N*17 ⇨ N<<4 + N

N%16 ⇨

Intel x86 processor instructions
and their speeds (2014)

In processors shift, and, or, add, and subtract are ***much faster*** than multiply, divide, and mod, which are ***relatively*** **slow.**

Table C-16. General Purpose Instruction

Instruction	Latency[1]	Throughput
CPUID	0F_3H	0F_3H
ADC/SBB reg, reg	8	3
ADC/SBB reg, imm	8	2
ADD/SUB	1	0.5
AND/OR/XOR	1	0.5
BSF/BSR	16	2
BSWAP	1	0.5
BTC/BTR/BTS	8-9	1
CLI		
CMP/TEST	1	0.5
DEC/INC	1	0.5
IMUL r32	10	1
IDIV **MOD is the same**	66-80	30

Given this, what is a way to compute these statements using combinations from <u>only</u> the ***fast*** operations above?

```
N//4   ⇨  N >> 2
N*7    ⇨  (N<<3) - N
N*17   ⇨  (N<<4) + N
N%16   ⇨  N-((N>>4)<<4)
```

Lab 4: Converting to binary...

base 10		base 2
100 10 1 141	=	128 64 32 16 8 4 2 1 1 0 0 0 1 1 0 1

141 // 2
70 // 2
35 // 2
17 // 2
8 // 2
4 // 2
2 // 2
1

What does the fact that 141 is ODD tell us?!

Try **right-to-left**!

141 = 10001101
128 64 32 16 8 4 2 1

answer

Insight: Ancient Egyptian Multiplication

Ancient Egyptian multiplication

From Wikipedia, the free encyclopedia

Next time?

Insight: Ancient Egyptian Multiplication

```
21  ×  6    ==    126
21      6
```

```
11  ×  15    ==    165
11      15
```

Try it here

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!

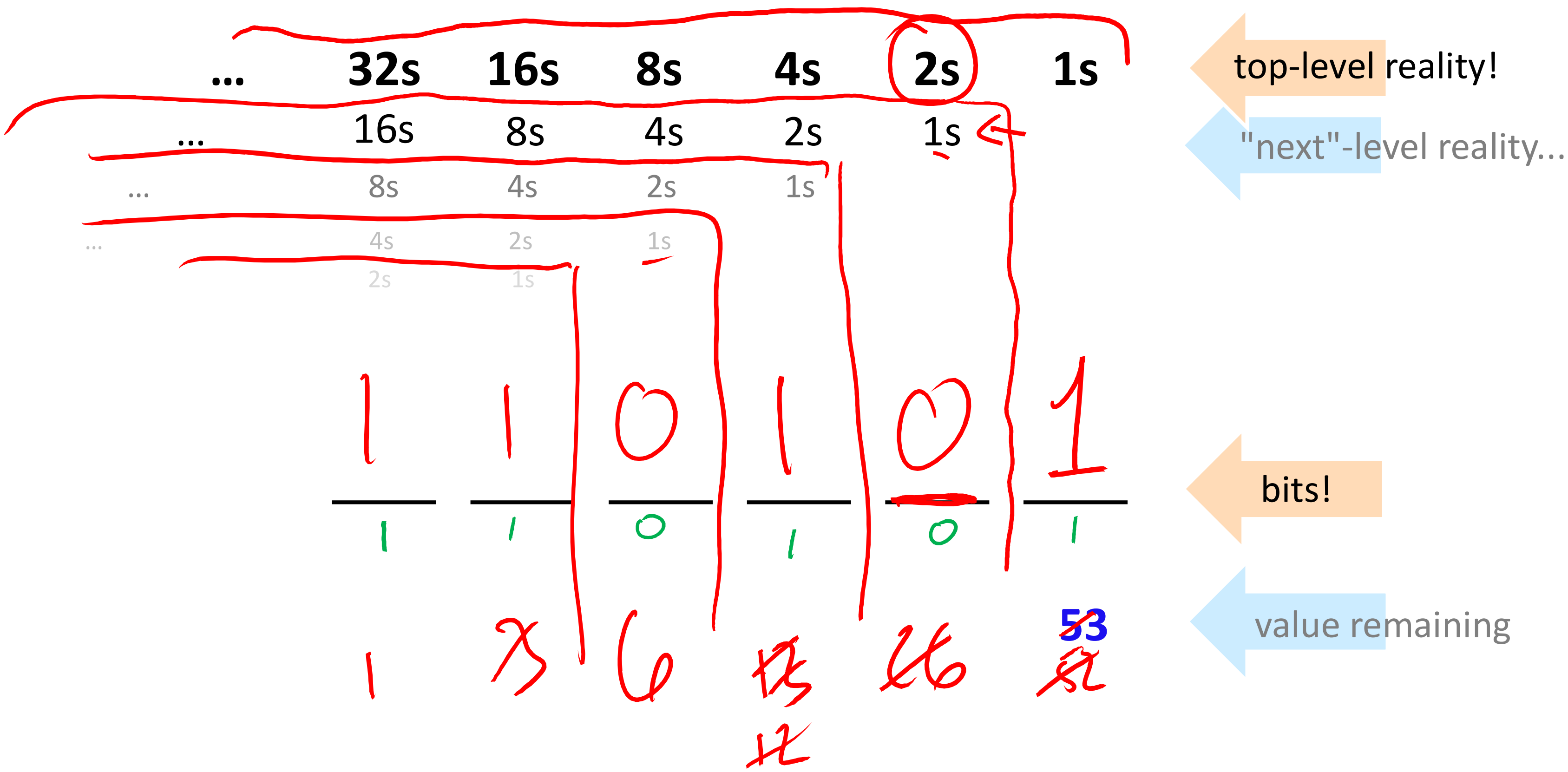
Why does this work?

or *RPM...*

Buddy, can you spare an eye?

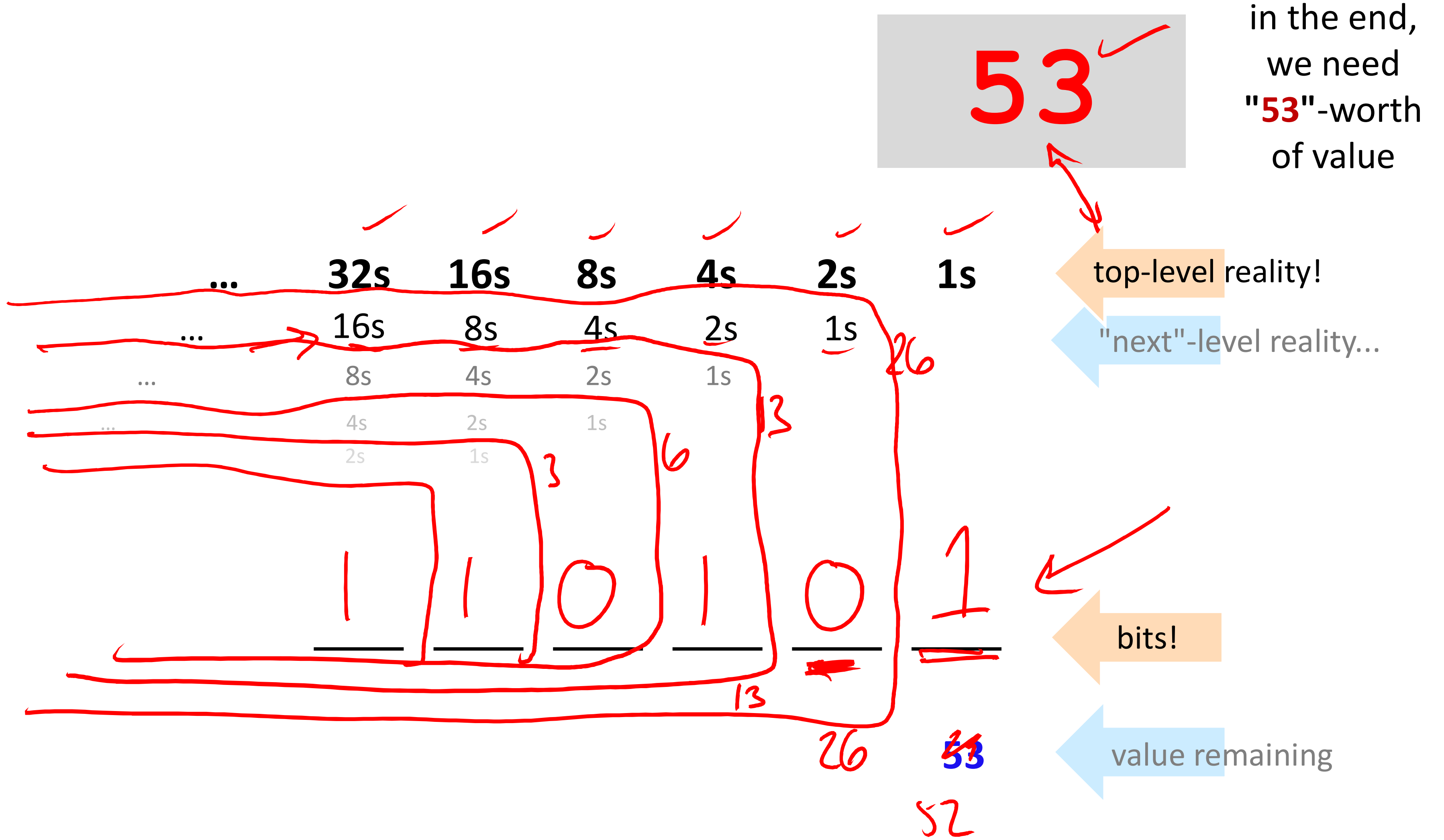
53

in the end,
we need
"**53**"-worth
of value



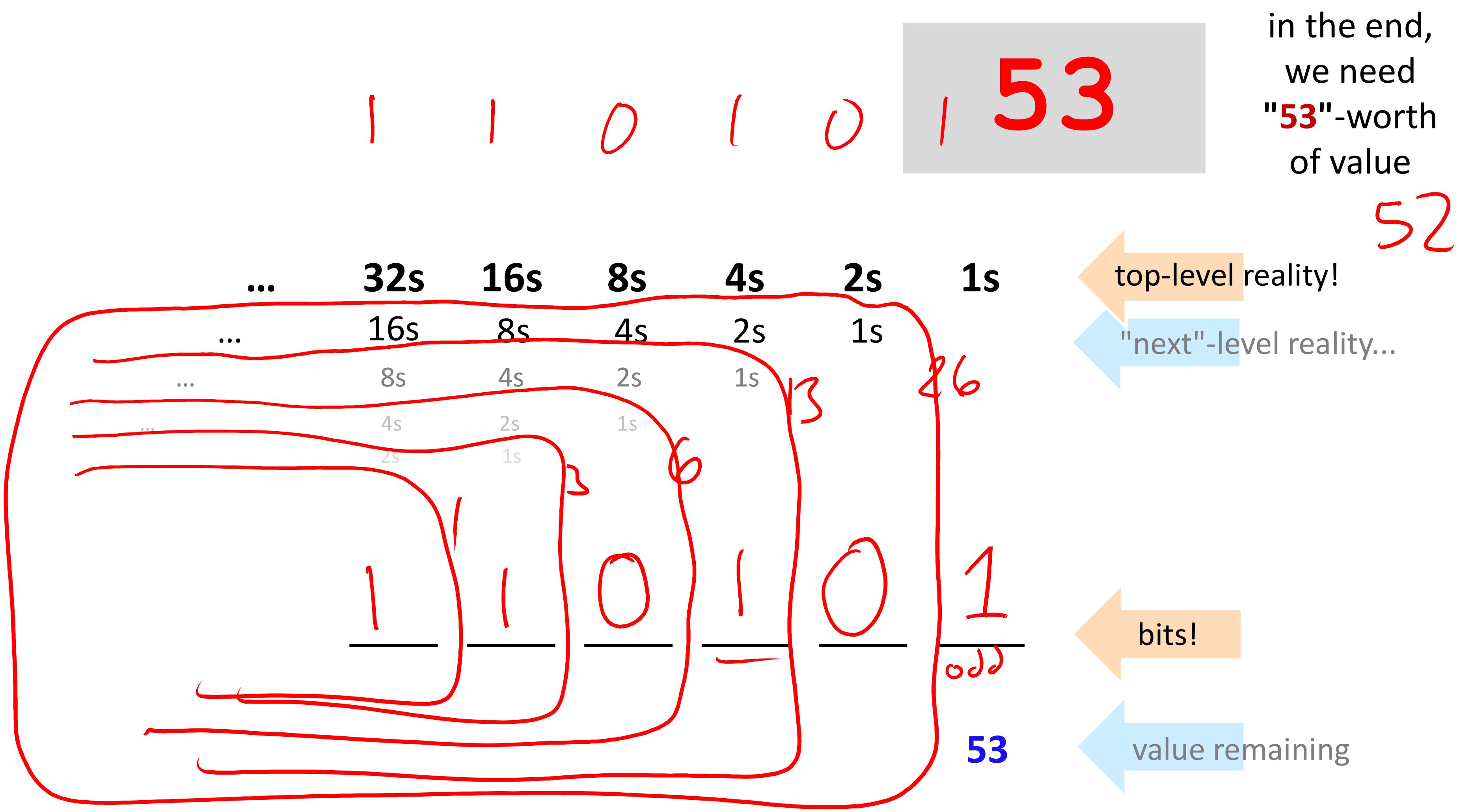
Extra! Can you figure out the **last binary digit** (bit) of **53** *without determining any earlier bits?* The last ***two***? ***three***?

All *of them?*



Extra! Can you figure out the **last binary digit** (bit) of **53** *without determining any earlier bits?* The last ***two***? ***three***?

All of them?



Extra! Can you figure out the **last binary digit** (bit) of **53** *without determining any earlier bits?* The last ***two***? ***three***?

All of them?

53

in the end, we need "**53**"-worth of value

dbl

1 1 0 1 0 1

...	**32s**	**16s**	**8s**	**4s**	**2s**	**1s**
...	16s	8s	4s	2s	1s	
...	8s	4s	2s	1s		
	4s	2s	1s			
	2s	1s				
	1	1	0	1	0	1

110101

top-level reality!

"next"-level reality...

bits!

value remaining

53

Extra! Can you figure out the **<u>last binary digit</u>** (bit) of **53** *without determining any earlier bits?* The last ***<u>two</u>***? ***<u>three</u>***?

All of them?

Extra! Can you figure out the **last binary digit** (bit) of **53** *without determining any other bits?* The last ***two***? ***3***? All?

...	32s	16s	8s	4s	2s	1s
...	16s	8s	4s	2s	1s	
...	8s	4s	2s	1s		
..	4s	2s	1s			
	2s	1s				
	1	1	0	1	0	1

top-level reality!

52 ✓

3

6

13

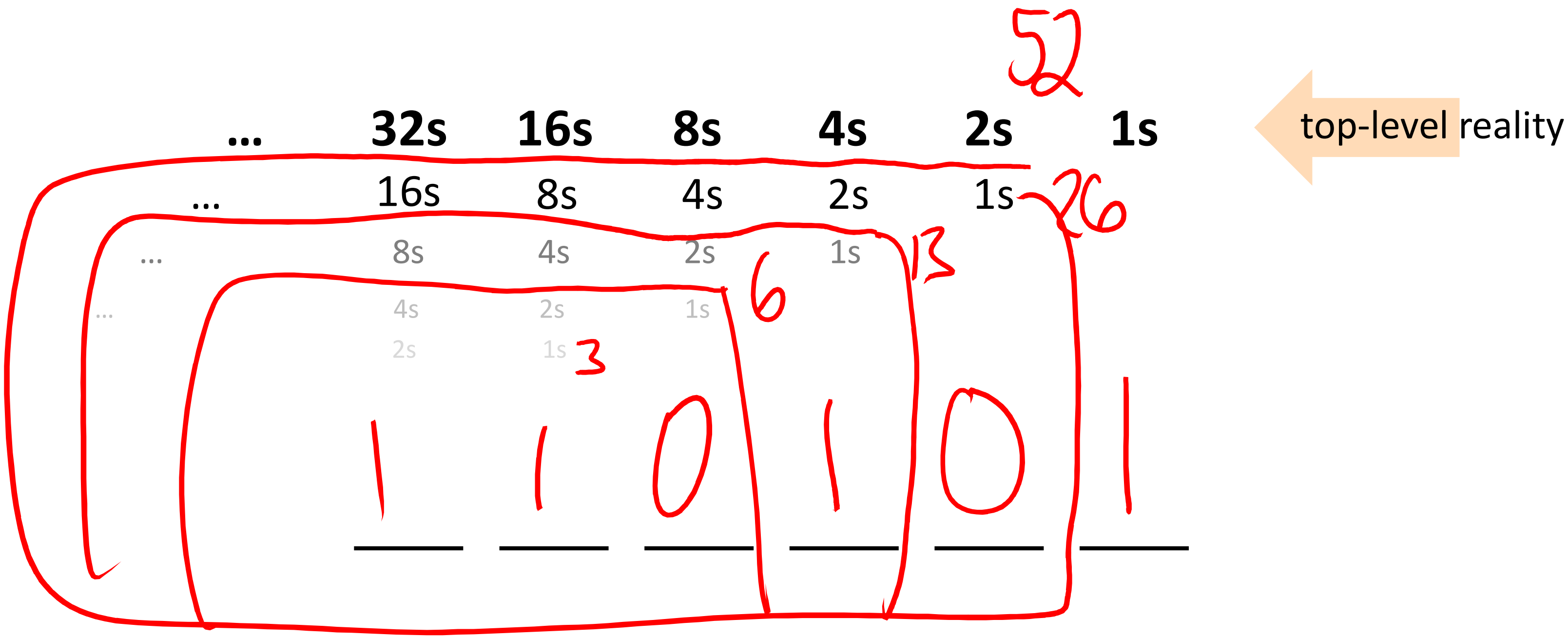
26

53

53

in the end, we need "**53**"-worth of value

Extra! Can you figure out the **last binary digit** (bit) of **53** *without determining any other bits?* The last ***two***? ***3***? All?



53

in the end,
we need
"**53**"-worth
of value

Extra! Can you figure out the **last binary digit** (bit) of **53** *without determining any other bits*? The last two? three?

All?

26 53

13

6

3

... ?s ?s ?s ?s

1 1 0 1 0 1

52

32 16 8 4 2 1

110101

42

$52/2 = 26$

53

in the end, we need "**53**"-worth of value

Extra! Can you figure out the **last binary digit** (bit) of **53** *without determining any other bits*? The last two? three?

All?

?s ?s ?s ?s

1 1 0 1 0 1

53

53 value

32 16 4 1

1 1 0 1 0 1

52 value remaining

53

in the end, we need "**53**"-worth of value