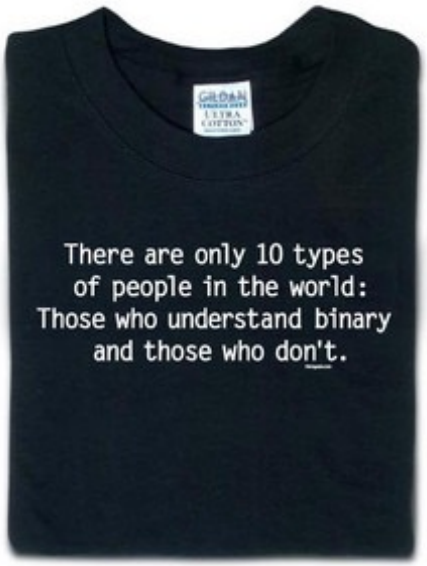
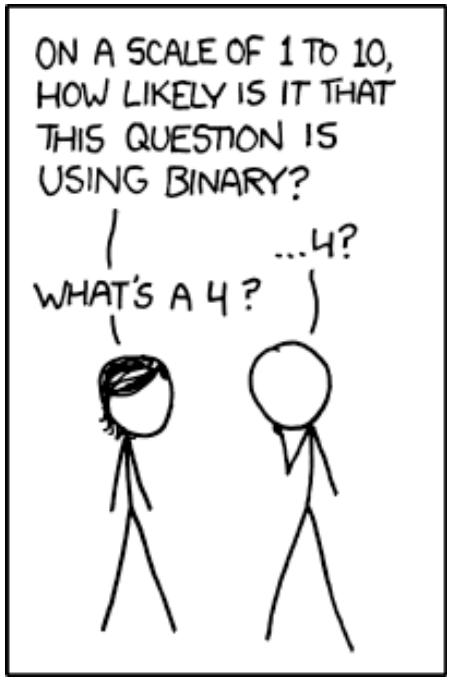
CS 101 Today...

Our top-10 list of binary jokes...





← **Looking Back**

Looking Forward →

Computing as composition

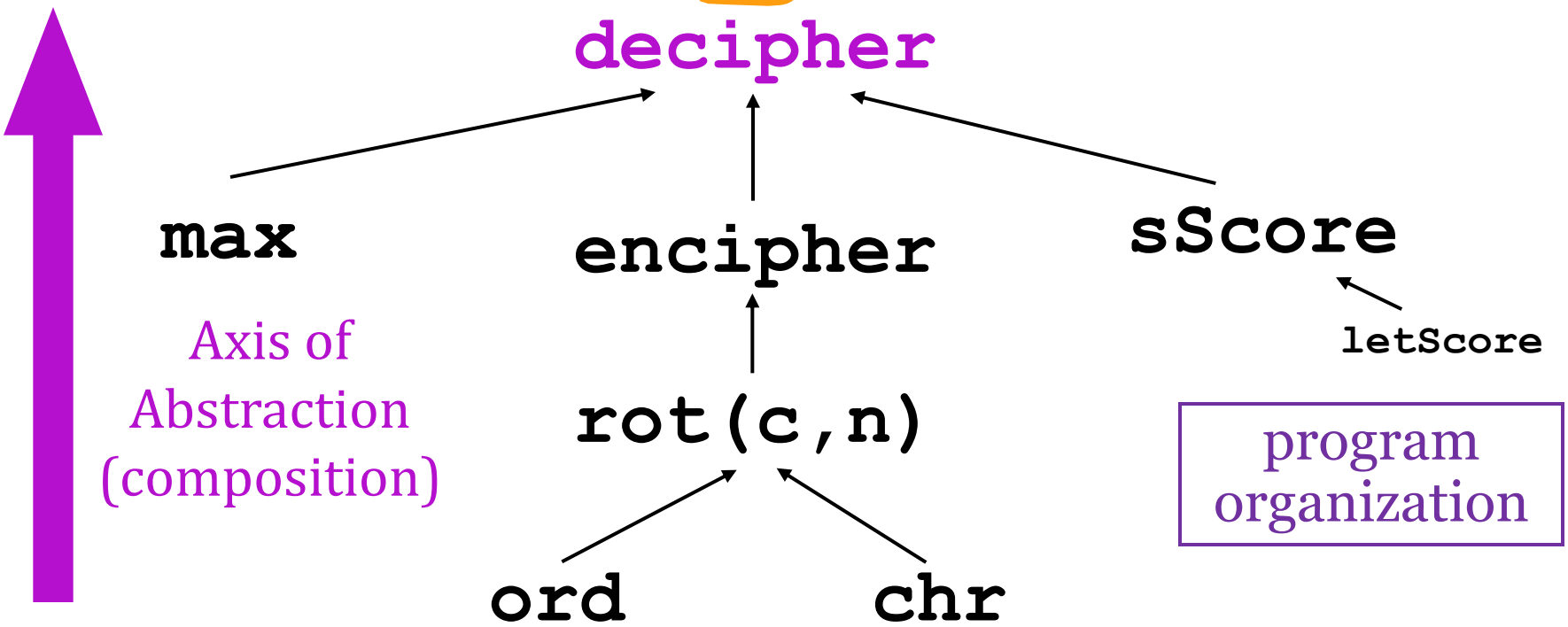
***clay* == functions**

Computing as representation

clay* == data & *bits

Some legs to stand on… ?

This is heady stuff!



Some legs to stand on!

It looks like I'm ahead of this...

creating more and more capable compositions

how do these things happen, physically ?!

decipher

max **encipher** **sScore**

letScore

rot(c,n)

ord **chr**

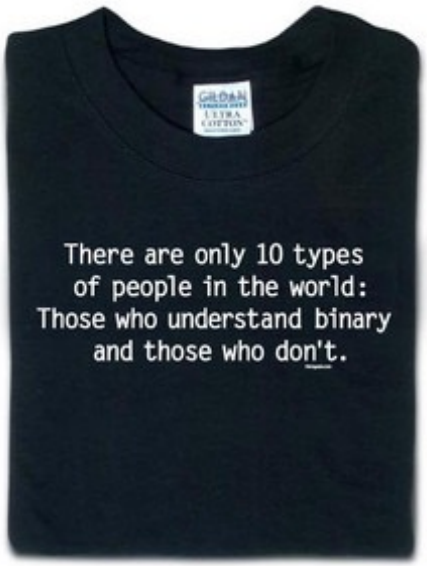
bits & circuits!

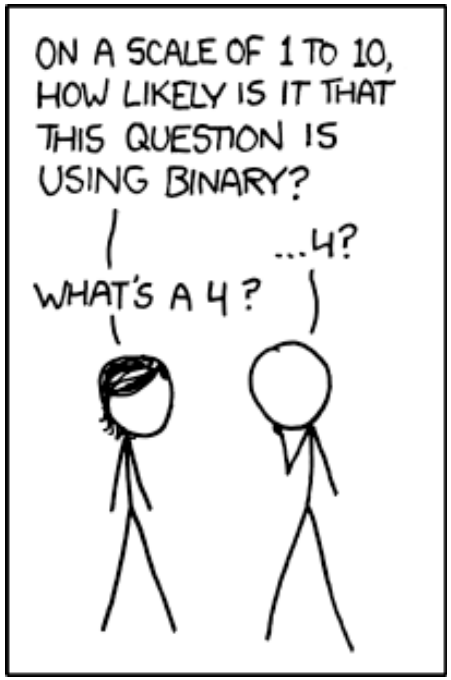
program organization

computer organization

CS 101 Today...

Our top-10 list of binary jokes...





← **Looking Back**

Looking Forward →

Computing as composition

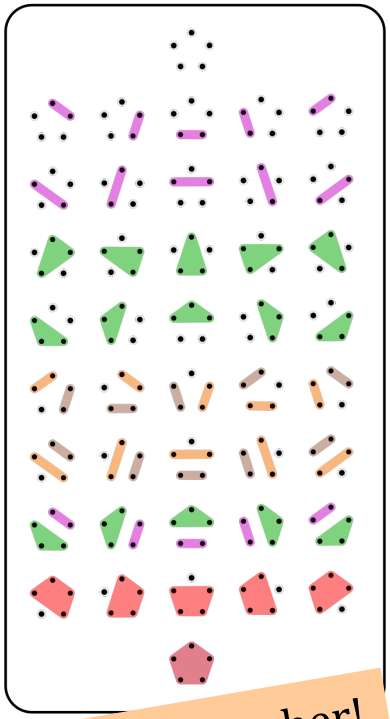
***clay* == functions**

Computing as representation

clay* == data & *bits

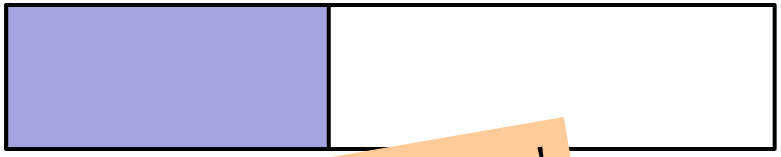
Some 42's! *Which are fundamental?*

```
len("Reveal the answer to the ultimate question")
```



5th Catalan number!

42_{10} 101010_{2}



Percentage!

b2n n2b

What we mean and what we say…

Syntax

42_{10} 101010_{2}

Representation

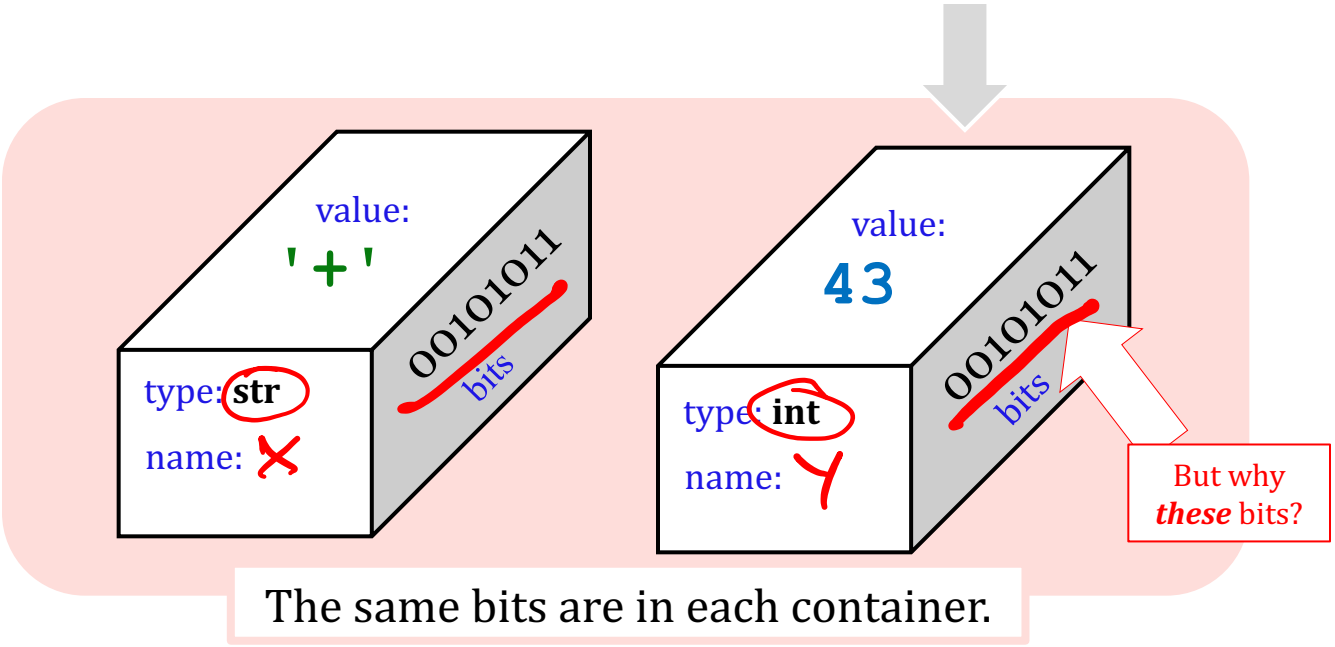
b2n n2b

Base-2 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



chr ord

Which bits?

What *is* 42 ?

42

It's *not* this!

forty-two

Value!

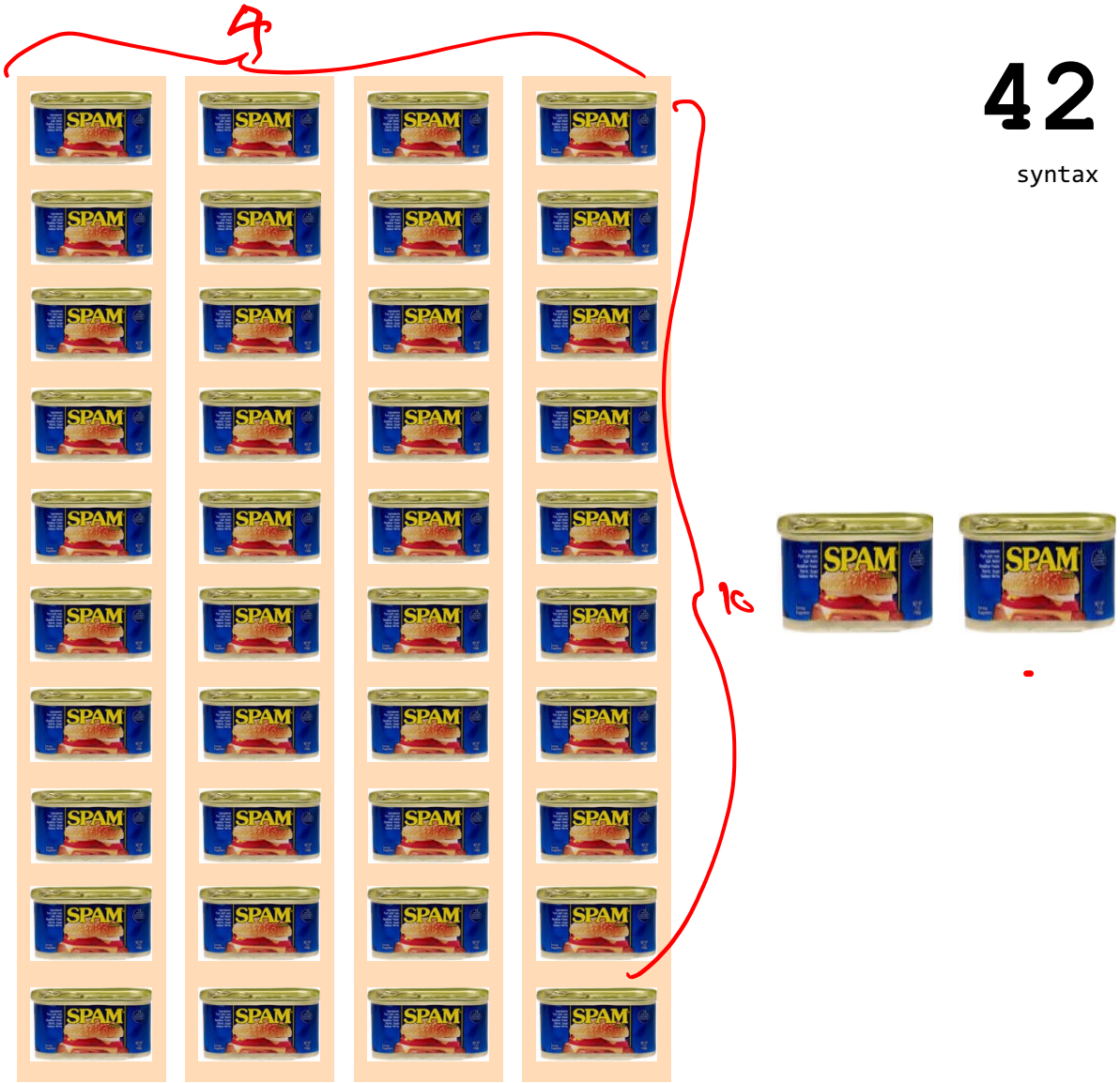
What *is* 42 ?

42

Syntax.

forty-two

value



42

syntax

tens

ones

forty-two

value

42

syntax

100

tens

ones

forty-two

value

Value

(semantics)

stuff we care about
(what things *mean*)

tens

42

syntax

Syntax

stuff we use *to communicate*

ones

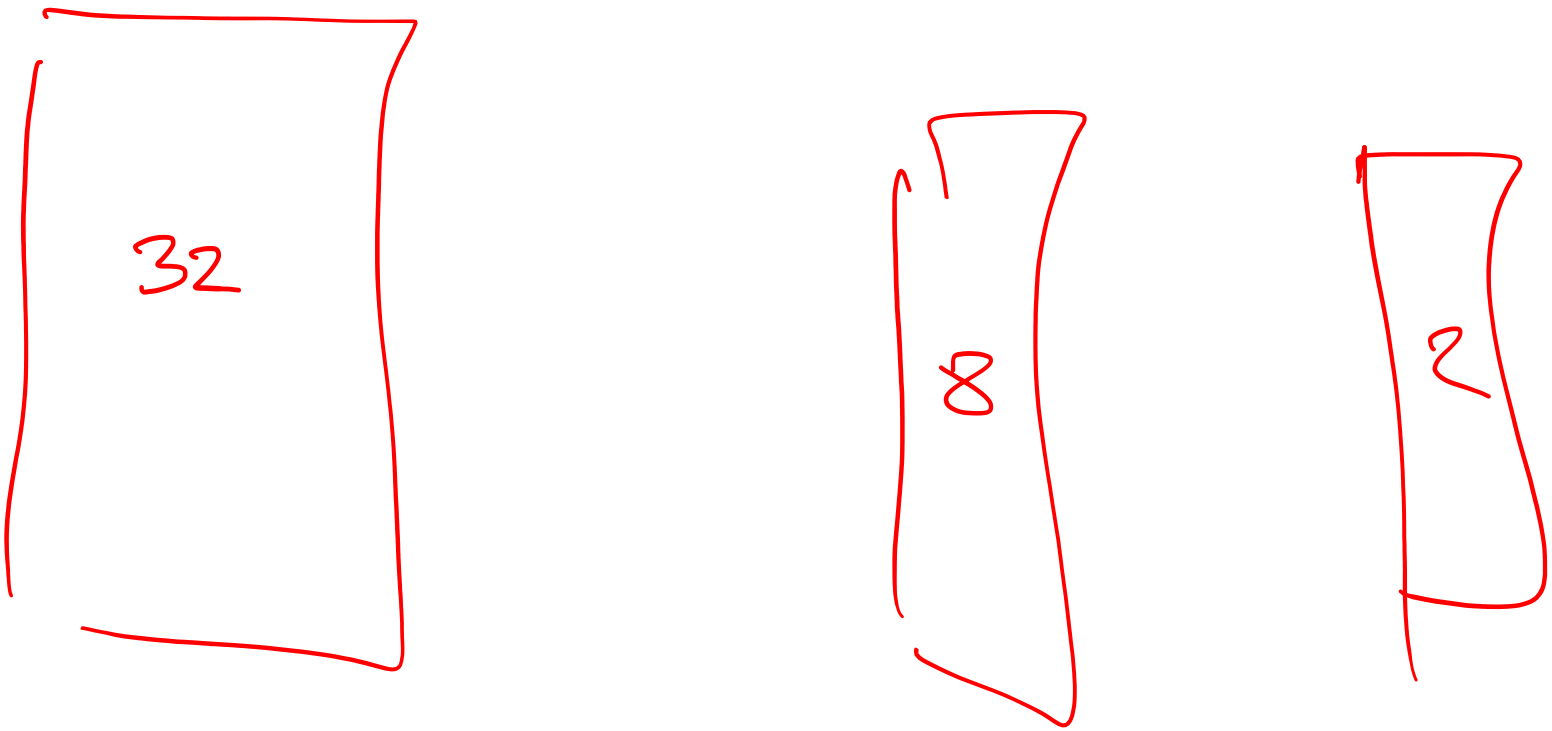
forty-two

value

~~42~~

~~10_2~~

Same Value!



thirty-twos	sixteens	eights	fours	twos	ones

but, a different representation

forty-two

value

Same Value!

thirty-twos	sixteens	eights	fours	twos	ones

but, a different representation

forty-two

value

101010

syntax

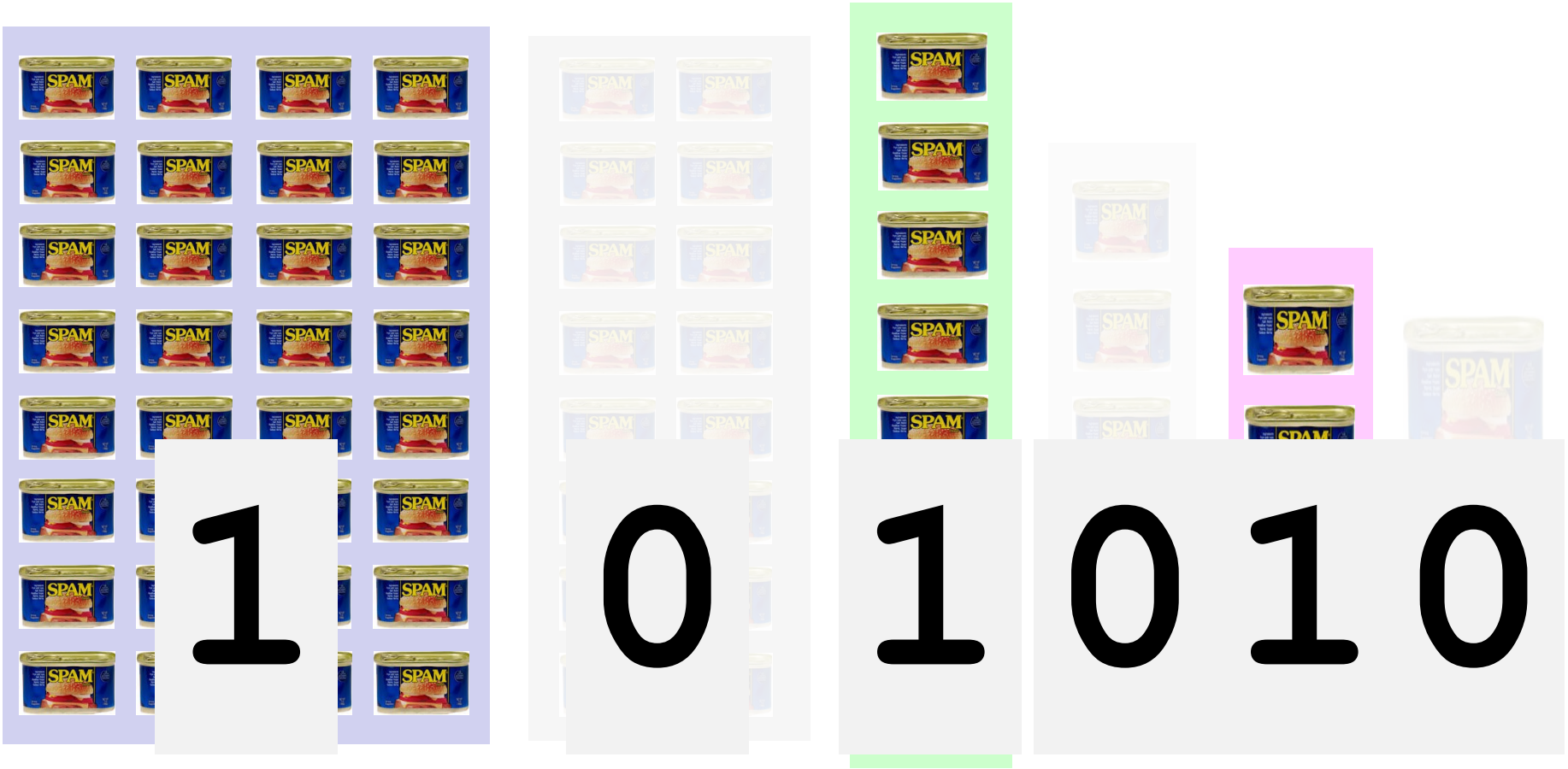
but, a different representation

forty-two

value

101010

syntax



thirty-twos	sixteens	eights	fours	twos	ones

with a *binary* syntax

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

32 + 8 + 2 => 42

TENs column
ONEs column

42_{10}

forty two

value

Value

stuff we care about
(what things *mean)*

Base 2

"binary"

Base 10

"decimal"

Different

THIRTY-
SIXTEENs
EIGHTs column
FOURs column
TWOs column
ONEs column

101010_2

Syntax

the symbols used
(what things look like)

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

Base 10

"decimal"

TENs column ONEs column

42_{10}

4 tens + 2 ones

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

each column represents the base's next power

HUNDREDs column TENs column ONEs column

$\cancel{123_{10}}$

$\cancel{27}$ $\cancel{+3}$ 1

1 hundred + 2 tens + 3 ones

0 1 1 1 1 0 1 1

writing 123 in binary...

Binary math

+	0	1
0	0	1
1	1	10

*	0	1
0	0	0
1	0	1

tables of one-digit 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		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

+	0	1
0	0	1
1	1	10

*	0	1
0	0	0
1	0	1

tables of one-digit 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

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
110011        10001000
```

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

```
32 16 8 4 2 1
28₁₀          101₁₀
```

Add these two binary numbers:

```
   101101
+    1110
--------
```

WITHOUT converting to decimal !

Multiply these binary numbers:

```
   101101
*    1110
--------
+
--------
```

```
  1
  529
+ 742
 1271
```

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

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

```
   1
  529
+ 742
 1271
```

Hint:

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

Add these two binary numbers
WITHOUT converting to decimal !

```
      1 1
    101101     45
+     1110     14
-----------
    111011     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
        101101        45
*         1110        14
-------------------
        101101
       101101
      101101
     101101
-------------------
     111010        630
```

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 !**

Multiply these two binary numbers
WITHOUT converting to decimal !

```
                32  16   8   4   2   1
            101101                      45
*             1110                      14
-------------------------------------
            000000
           1011010
          10110100
        + 101101000
-------------------------------------
         1001110110                     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 !**

Multiply these two binary numbers
WITHOUT converting to decimal !

```
                32  16   8   4   2   1
            101101                      45
*             1110                      14
-------------------------------------
            000000
           1011010
          10110100
        101101000
+
-------------------------------------
      1001110110                        630
      512 256 128  64  32  16   8   4   2   1
```

"partial products"

Multiply these two binary numbers
WITHOUT converting to decimal !

Hint:

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

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

32 16 8 4 2 1

101101 45

*

Perform a "shift right" operation to turn these in!

1001110110 630

512 256 128 64 32 16 8 4 2 1

Goal

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

Beyond Binary

base 1 111111111111111111111111111111111111111111 digits: **1**

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

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

~~42~~ ?

~~15~~ 60

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

Beyond Binary

base 1 11111111111111111111111111111111111111111 digits: **1**

2 A

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

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

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

base-4 base 7 base-9 base-11

Beyond Binary

base 1 1111111111111111111111111111111111111111 digits: **1**

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

base 3 — 27 9 3 1 **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 <u>one</u> of these ***isn't*** 42...?

222_4 60_7 **54** 46_9 39_{11}

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

$54_{7.6} == 42_{10}$

Everything's 42 if fractional bases are allowed!

All 42s!

base 1	1111111111111111111111111111111111111111		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

Off base?

Our Mascot, the Panda

DOZENAL SOCIETY OF AMERICA

Base 12 –

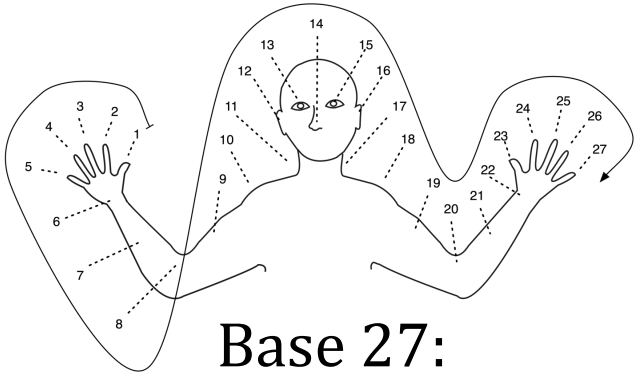
"~~Duodecimal~~ Society"

"***Dozenal*** Society"

Base 20: Americas

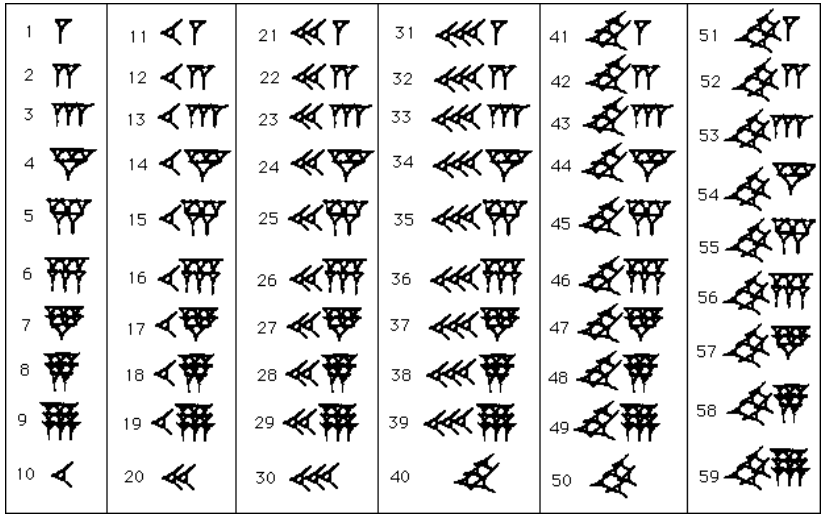
Olmec base-20 numbers
E. Mexico, ~ 300 AD

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

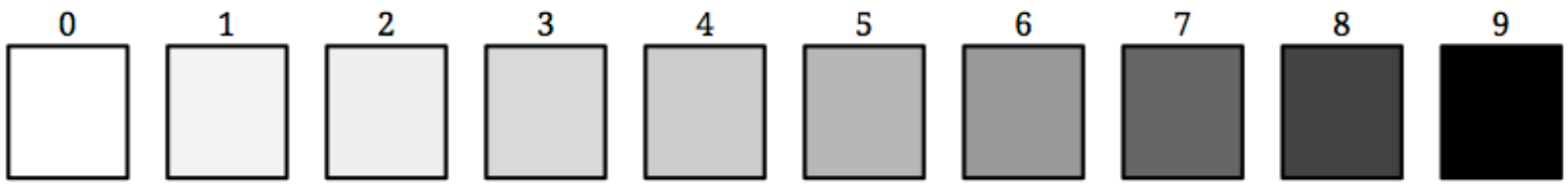


Some of these bases are still echoing around…

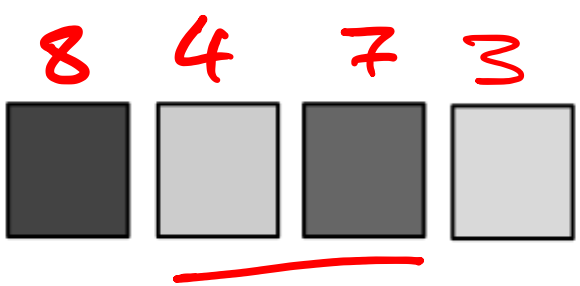
But ***why*** base-2?

Could computer circuits represent decimals?

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



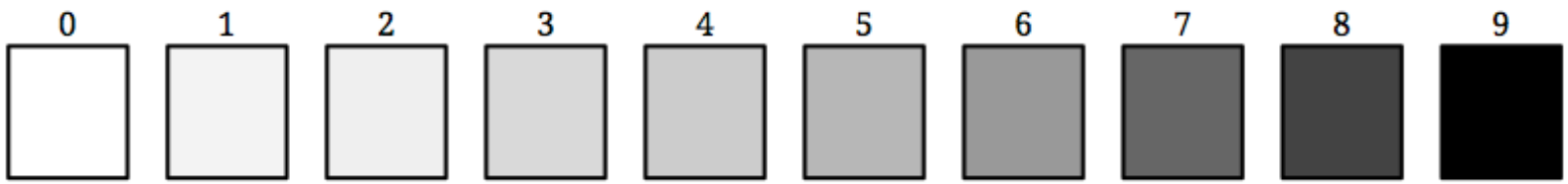
ten symbols ~ ten different voltages



What digits are these?

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

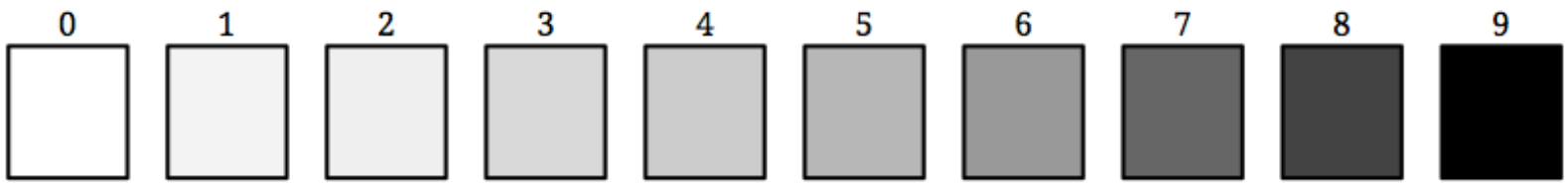
engineering!

What digits are these?

Ouch!

Ten symbols is too many!

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



ten symbols ~ ten different voltages

This is too difficult to replicate billions of times

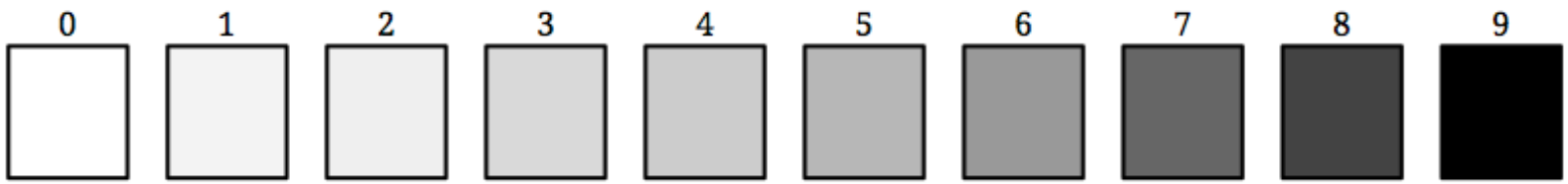
engineering!

What digits are these?

Ouch!

Ten symbols is too many!

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



ten symbols ~ ten different voltages

This is too difficult to replicate billions of times

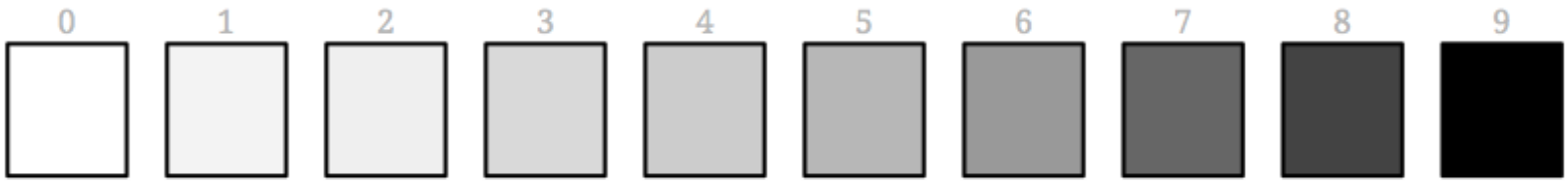
engineering!

What digits are these?

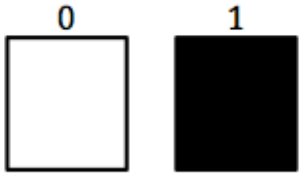
Ouch!

Two symbols is easiest!

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



ten symbols ~ ten different voltages



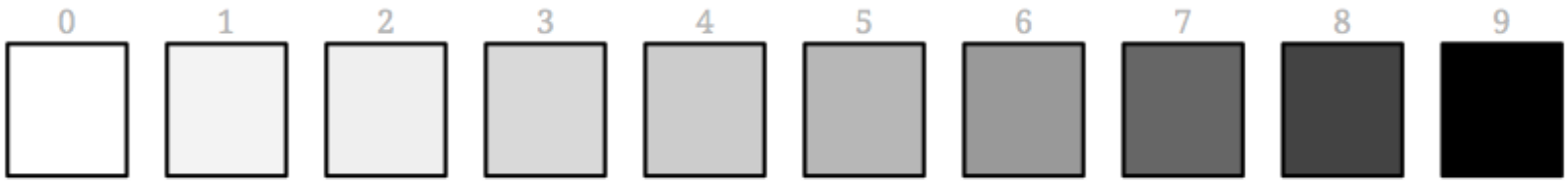
two symbols ~ two different voltages

What digits are these?

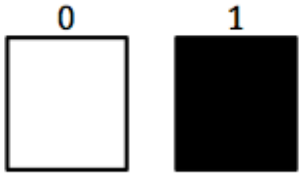
Easy!

Two symbols is easiest!

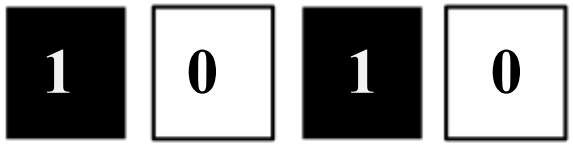
A computer has to differentiate *physically* among all its possibilities.



ten symbols ~ ten different voltages



two symbols ~ two different voltages

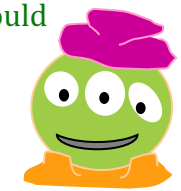


What digits are these?

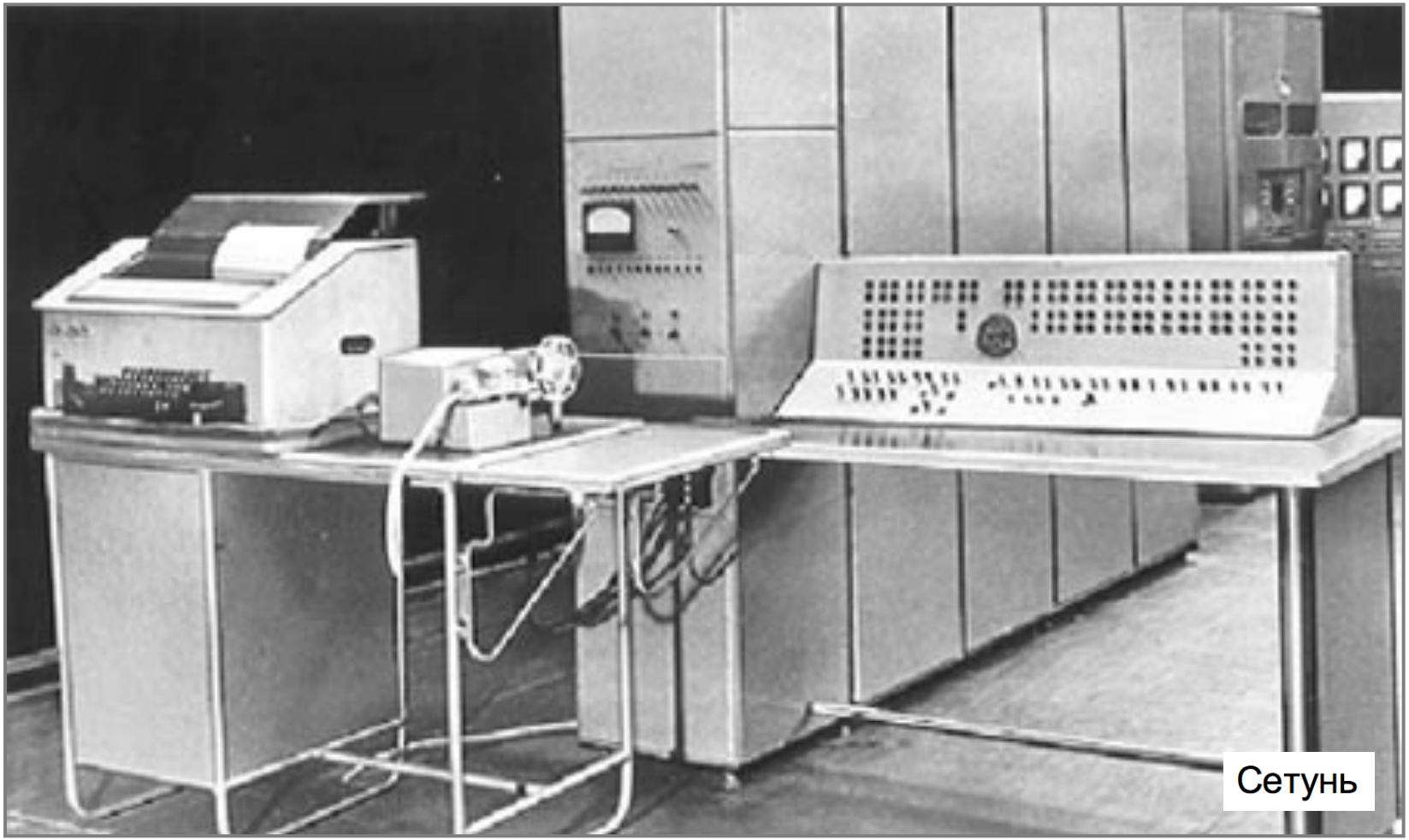
Easy!

Ternary computers?

Everything should be base-3!



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



+1
(+5volts)

0
(0volts)

-1
(-5volts)

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

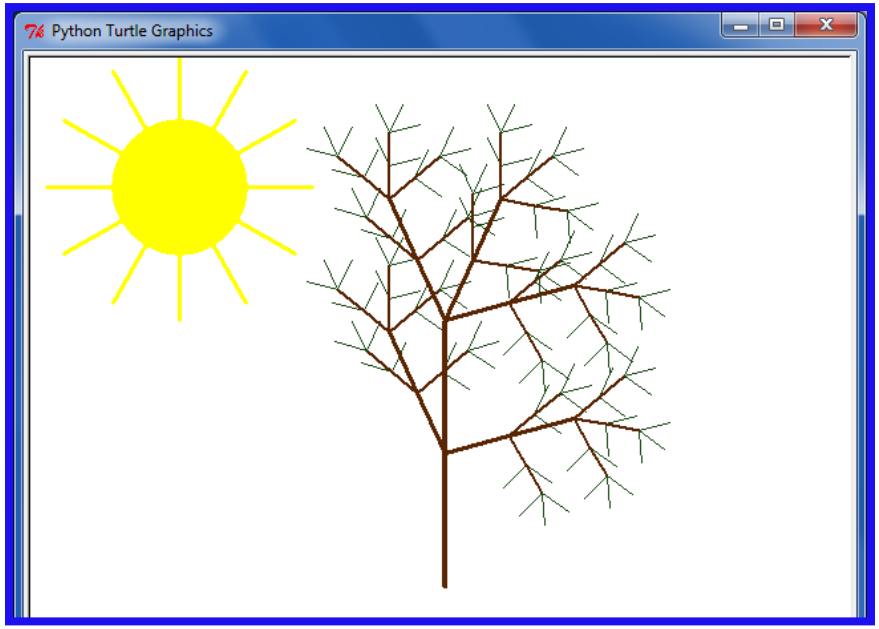
Eye-catching submissions...

```
|\_()_/ 1_________|wave|_________\_()_/ 2|
|_\_()_/ 1________|wave|_________\_()_/ 2|
|\_()_/ 1_________|wave|_________\_()_/ 2|
|_\_()_/ 1________|wave|________\_()_/ 2_|
|\_()_/ 1_________|wave|_________\_()_/ 2|
|_\_()_/ 1________|wave|_________\_()_/ 2|
|\_()_/ 1_________|wave|________\_()_/ 2_|
|_\_()_/ 1________|wave|_______\_()_/ 2__|
|__\_()_/ 1_______|wave|________\_()_/ 2_|
|___\_()_/ 1______|wave|_______\_()_/ 2__|
|__\_()_/ 1_______|wave|______\_()_/ 2___|
|_\_()_/ 1________|wave|_______\_()_/ 2__|
|__\_()_/ 1_______|wave|________\_()_/ 2_|
|_\_()_/ 1________|wave|_________\_()_/ 2|
|\_()_/ 1_________|wave|________\_()_/ 2_|
|_\_()_/ 1________|wave|_______\_()_/ 2__|
|__\_()_/ 1_______|wave|______\_()_/ 2___|
|_\_()_/ 1________|wave|_____\_()_/ 2____|
|\_()_/ 1_________|wave|______\_()_/ 2___|
|_\_()_/ 1________|wave|_______\_()_/ 2__|
|__\_()_/ 1_______|wave|______\_()_/ 2___|
|___\_()_/ 1______|wave|_______\_()_/ 2__|
|__\_()_/ 1_______|wave|______\_()_/ 2___|
|___\_()_/ 1______|wave|_____\_()_/ 2____|
|____\_()_/ 1_____|wave|____\_()_/ 2_____|
|_____\_()_/ 1____|wave|___\_()_/ 2______|
|______\_()_/ 1___|wave|____\_()_/ 2_____|
|_____\_()_/ 1____|wave|___\_()_/ 2______|
|____\_()_/ 1_____|wave|__\_()_/ 2_______|
|_____\_()_/ 1____|wave|__\_()_/ 2_______
```

Leah

ASCII wanderings...

and turtle art



a turtle-drawn portrait from turtle graphics ...

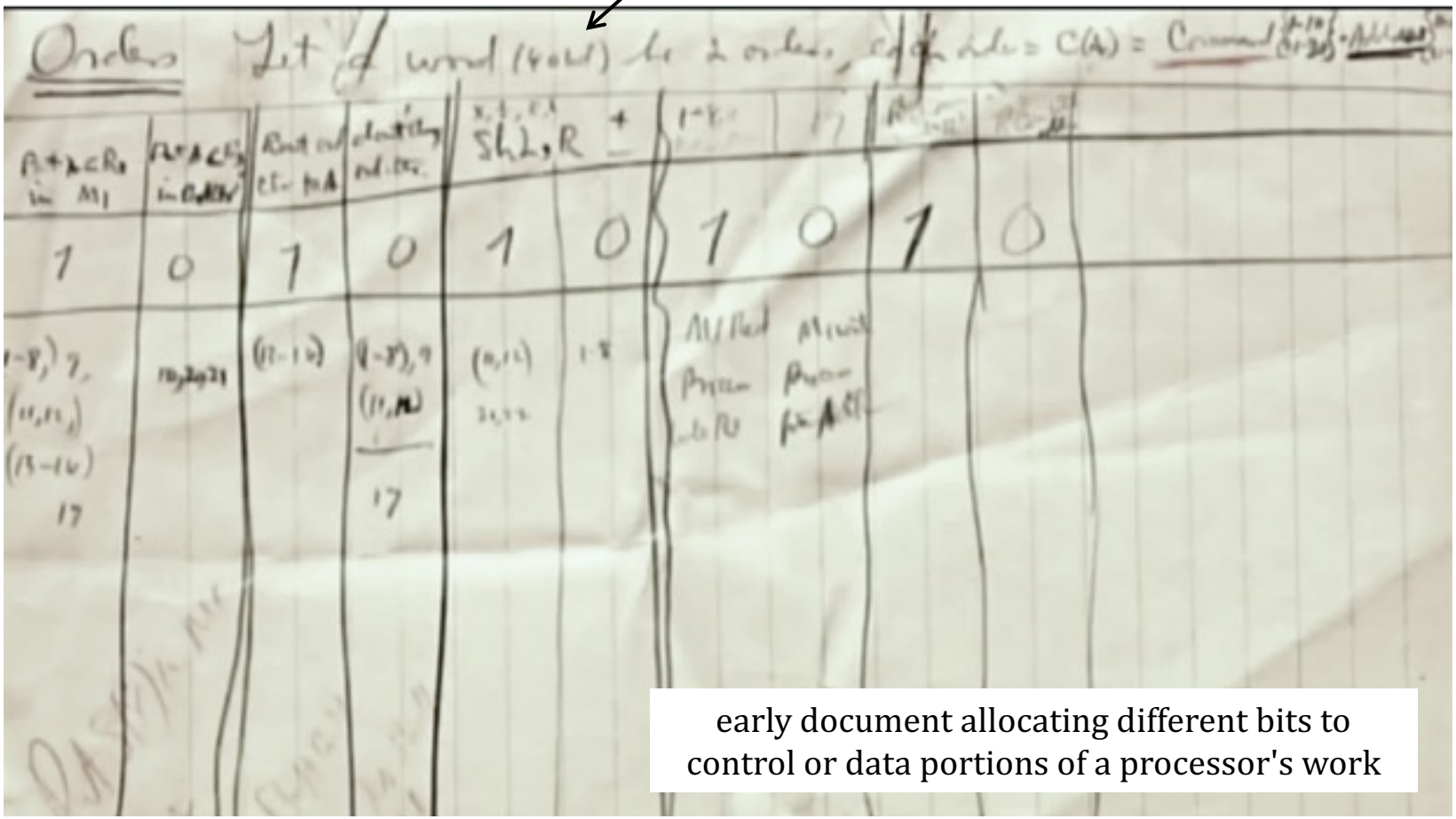
Whoa!
'12

Back to bits…

not the original name...

b.d. ~ binary digit ~ **bit**

"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 **last binary digit** (bit) of **53** *without determining any earlier bits*? The last ***two***? ***three***?

All of them?

Lab 4: *Computing to base-2*

base 10		base 2
100 10 1 **53**	=	… 4 2 1

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

You mean ***aside*** from having to think in binary?

Lab 4: *Computing to base-2*

base 10		base 2
100 10 1 **53**	=	… 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 to base-2*

base 10		base 2
100 10 1 **53**	=	64 32 16 8 4 2 1

What does the fact that 53 is ODD tell us?!

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

53	=	**110101** 128 64 32 16 8 4 2 1

answer

Lab 4: *Computing to base-2*

base 10		base 2
100 10 1 **53**	=	64 32 16 8 4 2 1 **110101**

What does the fact that 53 is ODD tell us?!

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

Why does this work?!

53 = **110101**

128 64 32 16 8 4 2 1

answer

Lab 4: *Computing to base-2*

base 10		base 2
100 10 1 **53**	=	64 32 16 8 4 2 1 **110101**

What does the fact that 53 is ODD tell us?!

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

Why does this work?!

recursion!

53 = **110101**

128 64 32 16 8 4 2 1

answer

53

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

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

top-level reality!

"next"-level reality...

____	____	____	____	____	____

bits!

53

value remaining

Converting to binary ~ starting from the right!

1 3 6 13 26 52 53

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

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

top-level reality!

"next"-level reality...

1 1 0 1 0 1

bits!

53

value remaining

Converting to binary ~ starting from the right!

Lab 4: *Computing in binary*

base 10		base 2
100 10 1 **53**	=	128 64 32 16 8 4 2 1 **'110101'**

Right-to-left works!

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

numToBinary(N)

decimal syntax, N

n2b(53)

binaryToNum(S)

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

b2n('110101')

Lab 4: *Computing in binary*

base 10		base 2
100 10 1 **53**	=	128 64 32 16 8 4 2 1 **'110101'**

```
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!?

hw4pr1 (lab)

hw4pr2 (hw)

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!?

hw4pr1 (lab)

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> #?

<<

left-shift

>>

right-shift

bitwise Python: the left and right "shift operators"

Reasoning, *bit by bit*

<< left-shift

>> right-shift

left-shift by 1

11
110

3 << 1

6

3*2

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

left-shift by 2

11
1100

3 << 2

12

3*4

right-shift by 1

101010
10101

42 >> 1

21

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

42//2

42 >> 4

=> 2

42//4

1010

42 >> 2 ?

10

Intel x86 processor instructions and their speeds

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

Given this, how can we compute these slow statements using ***fast*** operations?

Table C-15. General Purpose Instructions

Instruction	Latency[1]		Throughput	
CPUID	0F_3H	0F_2H	0F_3H	0F_2H
ADC/SBB reg, reg	8	8	3	3
ADC/SBB reg, imm	8	6	2	2
ADD/SUB	1	0.5	0.5	0.5
AND/OR/XOR	1	0.5	0.5	0.5
BSF/BSR	16	8	2	4
BSWAP	1	7	0.5	1
BTC/BTR/BTS	8-9		1	
CLI				26
CMP/TEST	1	0.5	0.5	0.5
DEC/INC	1	1	0.5	0.5
IMUL r32	10	14	1	3
IMUL imm32		14	1	3
IMUL		15-18		5
IDIV + MOD	66-80	56-70	30	23

C `N//4`

D `N*17`

A `N*7`

B `N%16`

8N −N

A `(N<<3) - N`

B `N-((N>>4)<<4)`

C `N >> 2`

D `(N<<4) + N`

16N +N

Intel x86 processor instructions and their speeds

Table C-15. General Purpose Instructions

Instruction	Latency[1]		Throughput	
CPUID	0F_3H	0F_2H	0F_3H	0F_2H
ADC/SBB reg, reg	8	8	3	3
ADC/SBB reg, imm	8	6	2	2
ADD/SUB	1	0.5	0.5	0.5
AND/OR/XOR	1	0.5	0.5	0.5
BSF/BSR	16	8	2	4
BSWAP	1	7	0.5	1
BTC/BTR/BTS	8-9		1	
CLI				26
CMP/TEST	1	0.5	0.5	0.5
DEC/INC	1	1	0.5	0.5
IMUL r32	10	14	1	3
IMUL imm32		14	1	3
IMUL		15-18		5
IDIV + MOD	66-80	56-70	30	23

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

Given this, how can we compute these slow statements using ***fast*** operations?

C `N//4`

D `N*17`

A `N*7`

B `N%16`

A `(N<<3) - N`

B `N-((N>>4)<<4)`

C `N >> 2`

D `(N<<4) + N`

Insight: Ancient Egyptian Multiplication

Ancient Egyptian multiplication

From Wikipedia, the free encyclopedia

Next time?

Insight: Ancient Egyptian Multiplication

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?

21 × 6 == 126

21	6

---

11 × 15 == 165

11	15

Try it here

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
---------
&
```

11 & 5

bitwise or

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

5 | 6

7

bitwise or

```
11: 1011
 5: 0101
---------
|
```

11 | 5