Computer Science: The Good Parts
We will begin shortly.

A Practical Journey for
Early-Career Developers

Part 1

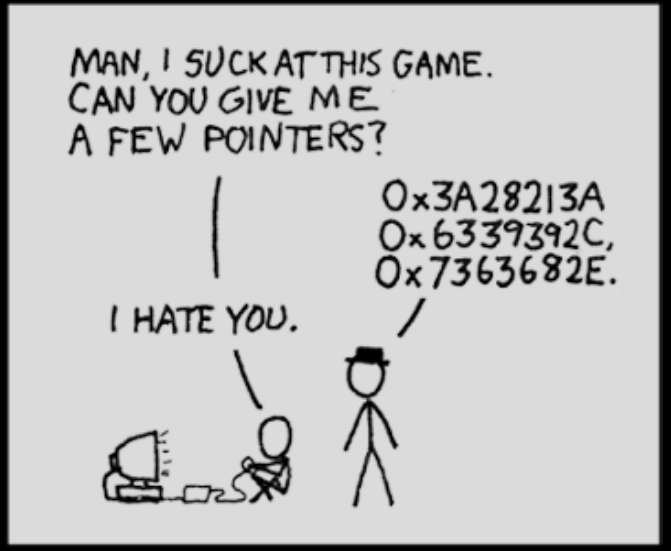


Image credit: https://xkcd.com/138/

Computer Science: The Good Parts

A Practical Journey for
Early-Career Developers

Part 1

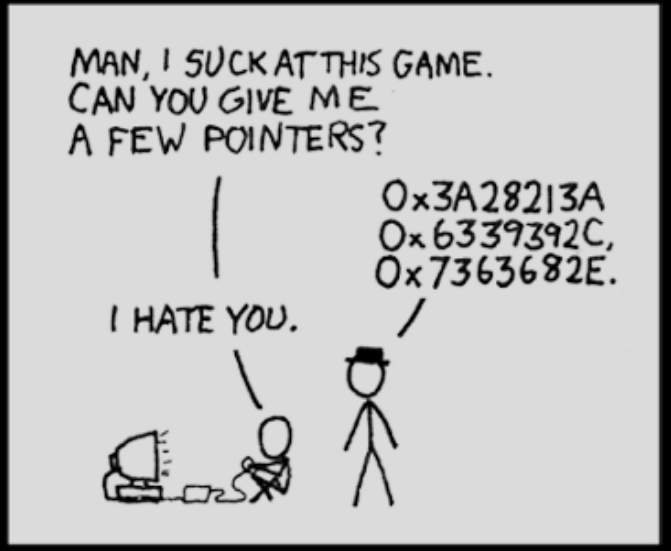


Image credit: https://xkcd.com/138/

Who is this workshop for?

Answer:

Early-career developers without a formal background in CS.

Why "early-career"? Won't this helpful to everyone?

Yes, but the code samples, metaphors, and sequencing is geared for people relatively new to programming, and who haven't yet had the opportunity to encounter these concepts on the job.

Who am I?

Remote Course Protocol

Some of you already know some of this stuff.

Try to wait for others to answer questions.
Feel free to skip that part.

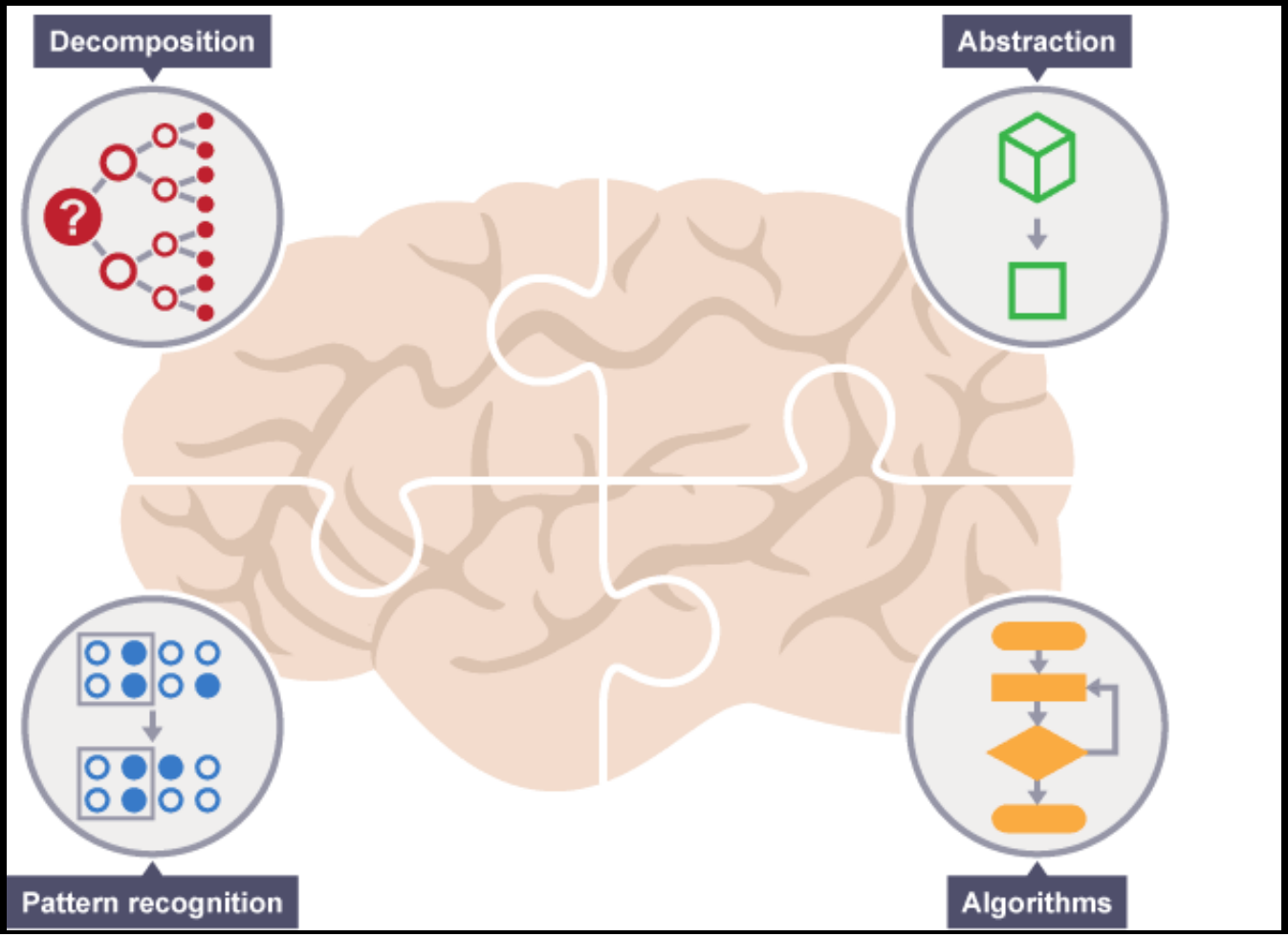
Feel free to turn off your video & mute if you need a break.

Otherwise keep your video on if possible.

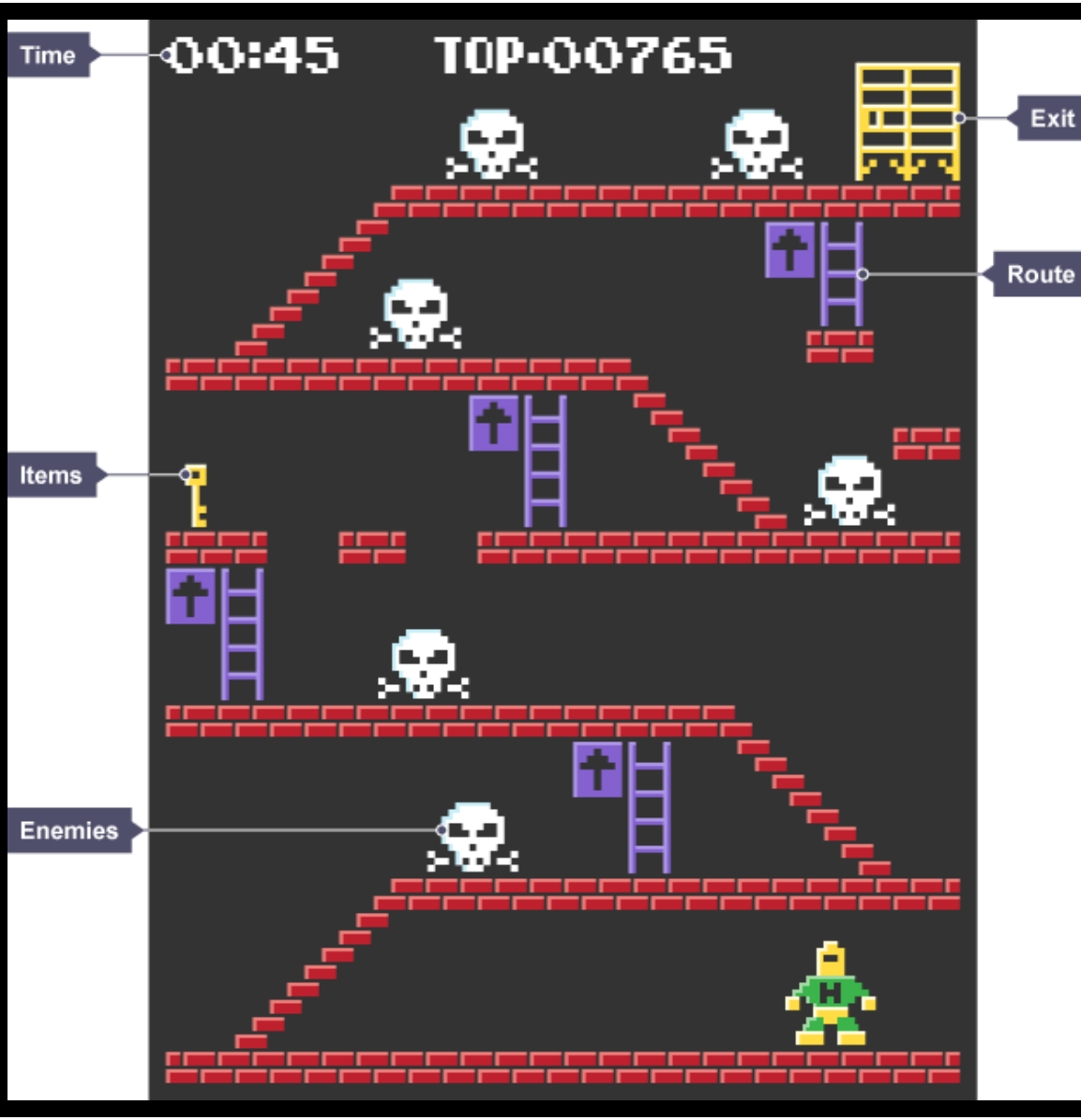
Use the chat for off-topic questions or comments.

I'll review during the breaks.

Computational Thinking



Computational Thinking



Computational Thinking

I think this is the best part of computer science.

It is the primary heuristic for whether or not to include a participular topic in this course.

It is language-agnostic and enhances both coding and non-coding tasks alike.

Number Systems

Binary (Base 2) Number System

	Decimal	Binary
	1	1
	2	10
	4	100
		
		

Binary Addition

N	N + 1	N + 2
0 0 0 0 0 0 0 1		
0 0 0 0 0 0 1 1		
0 0 0 0 0 1 0 0		
0 0 1 1 0 0 0 0		

Binary Numbers

00000001000000010000010000000101

Binary Numbers

00000001000000010000010000001001

8 bits = 1 byte

4 bits = 1 nybble

Binary Numbers

0000000100000010000010000000001001

16 bits =2 bytes = 1 word
on a 16-bit processor

Binary Numbers

```
00000001 00000010 00001000 00001001
```

32 bits =4 bytes = 1 word

on a 32-bit processor

Binary Numbers

```
00000001000000010000010000000001001
```

32 bits =4 bytes = 1 word

on a 32-bit processor

Binary Multiplication

N	N x 2	N x 4
0 0 0 0 0 0 0 1	0 1 1 1 0 1 1 0	0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 1	0 0 0 0 1 1 1 1	0 0 0 0 1 1 0 0
0 0 0 0 0 1 0 0	0 0 0 0 0 0 1 0	
0 0 1 1 0 0 0 0		

Uh Oh

One-half of an apple 0.5000

`0 . 1 0 0 0`

One-third of an apple 0.3333....

`0 . 0 1 0 1 …`

Uh Oh

One-tenth of an apple 0.1

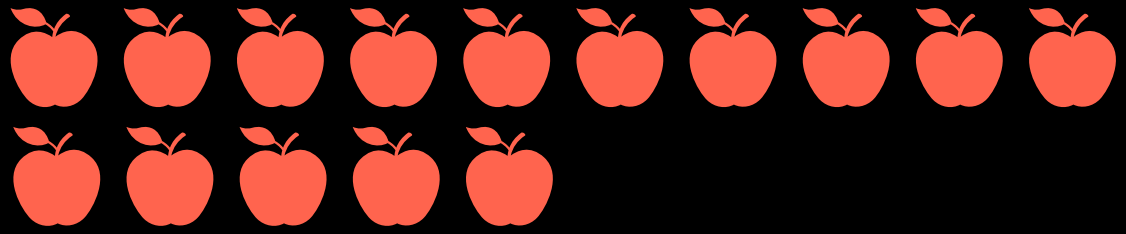
0.0001100…

Uh Oh

```
Python 3.12.2 (v3.12.2:6abddd9f6a, Feb  6 2024, 17:02:
Type "help", "copyright", "credits" or "license" for m
>>> n = 0.1
>>> format(n, ".30f")
'0.100000000000000005551115123126'
>>>
```

(Floating Point Numbers)

Hexadecimal (Base 16) Numbers

	Decimal	Hex
	1	1
	9	9
	10	A
	15	F

Hexadecimal (Base 16) Numbers

#FF00CC

Decimal	Hex
16	10
32	20
256	100 - 1 = FF

Hex Addition

N	N + 1	N + 2
8		
9 9		
0 F		
F E		
E F		

Octal (Base 8) Numbers

Decimal	Octal
8	
15	
64	
63	

Boolean Logic

Binary Operators

```
  0
+ 1
---
  0
```

```
  1
NAND 1
-----
  0
```

```
  0
OR  0
-----
  0
```

```
  1
XOR 1
-----
  0
```

```
  0
NOR 0
-----
  1
```

```
NOT 0
-----
  1
```

```
  1 1
XOR 0
-----
  1
```

Truth Tables

INPUT	AND	NAND	OR	NOR	XOR
0 0	0	1	0	1	0
0 1	0	1	1	0	1
1 0	0	1	1	0	1
1 1	1	0	1	0	0

Logic Gates

https://logic.ly

Here's a puzzle I couldn't solve:

The computer only works with binary digits.

Yet: I can see photos, watch movies, and listen to music on my computer.

How is this possible?

Alan Turing enabled these abilities back in 1936.

The key is simply interpret the bits to *represent* something else.

We need to set things up such that the computer somehow "knows" when a memory value is just a number, and when it should be interpreted as something else.

Representations

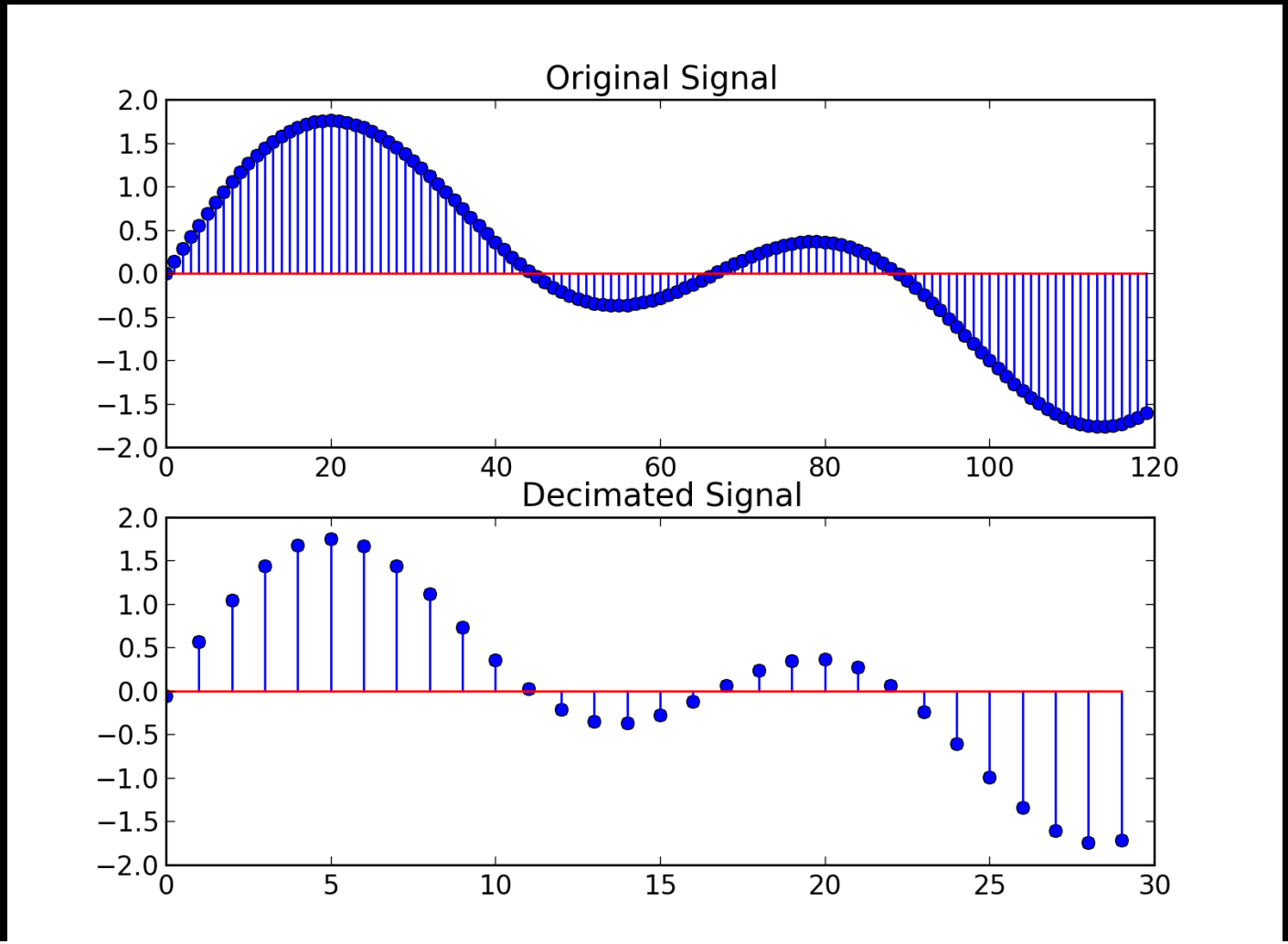
- Alphabets
- Colors
- Images
- Sounds
- … so much more

Representations

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Representations



What was Turing's breakthrough?

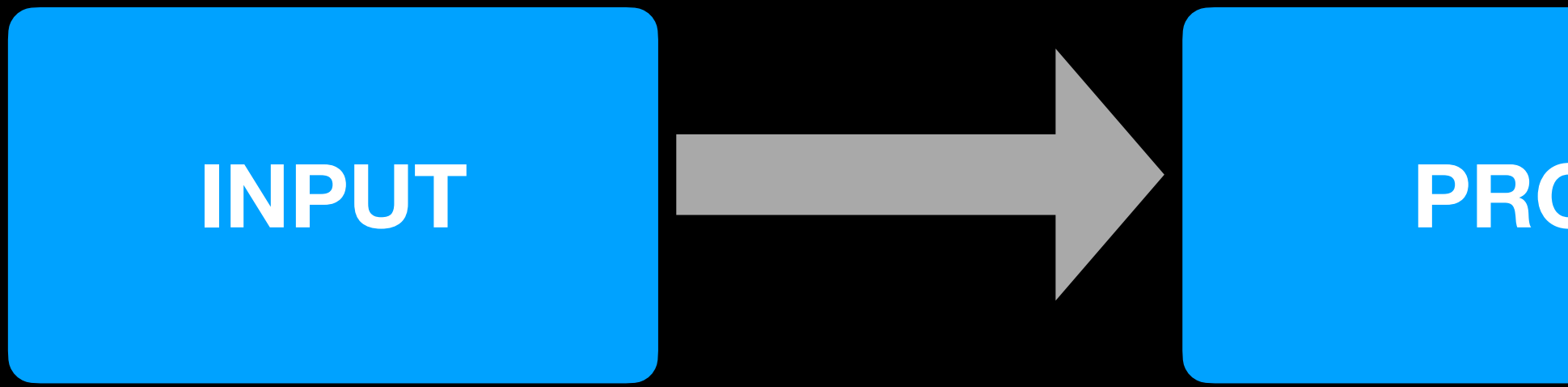
To understand the brilliance of his idea on representation, we must first understand how computing machines worked in the 1930's.

What was Turing's breakthrough?

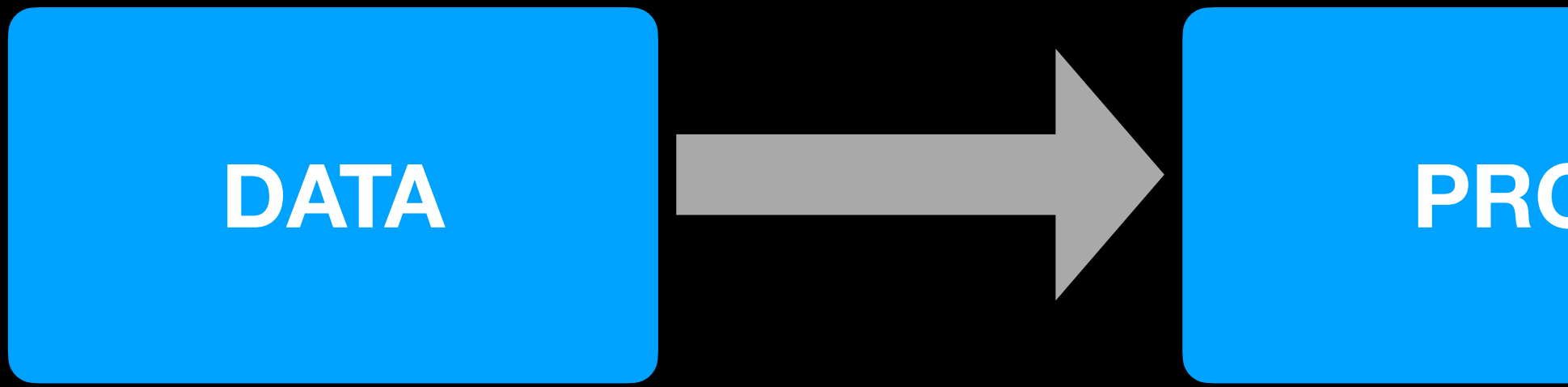
Turing imagined whether there could be a machine that could imitate any other machine.

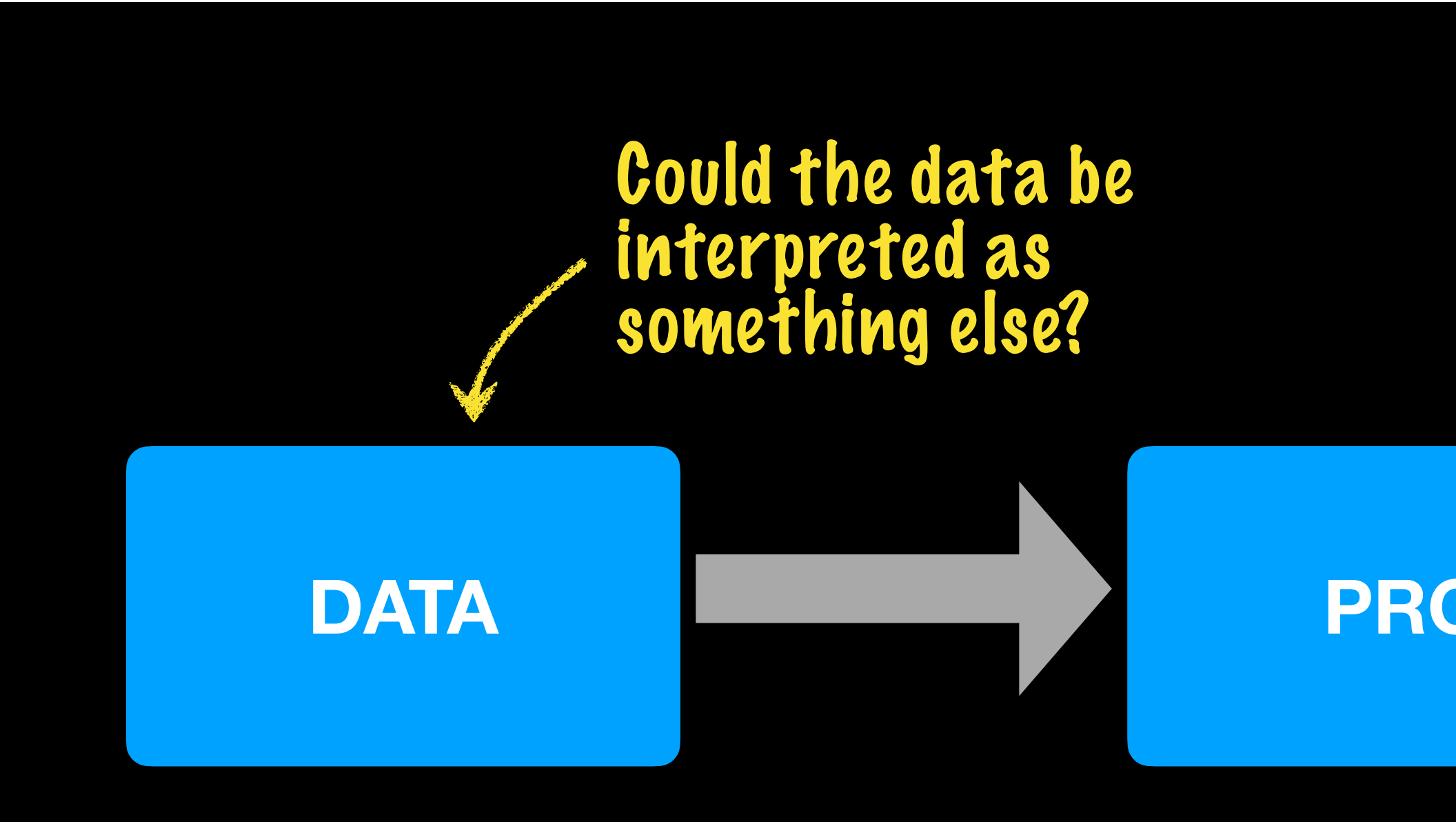
He ended up inventing the famous *Turing Machine*.

What was Turing's breakthrough?

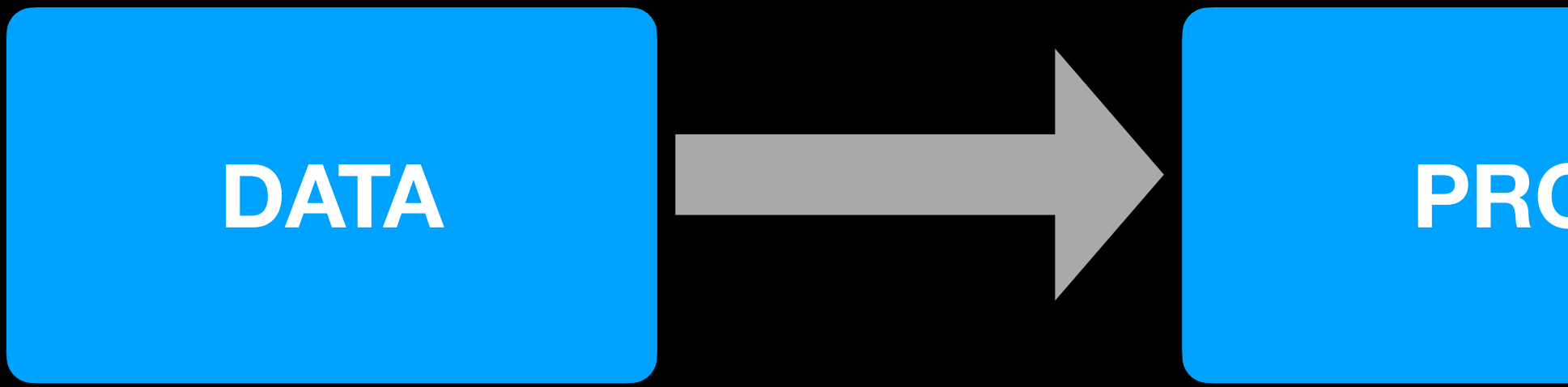


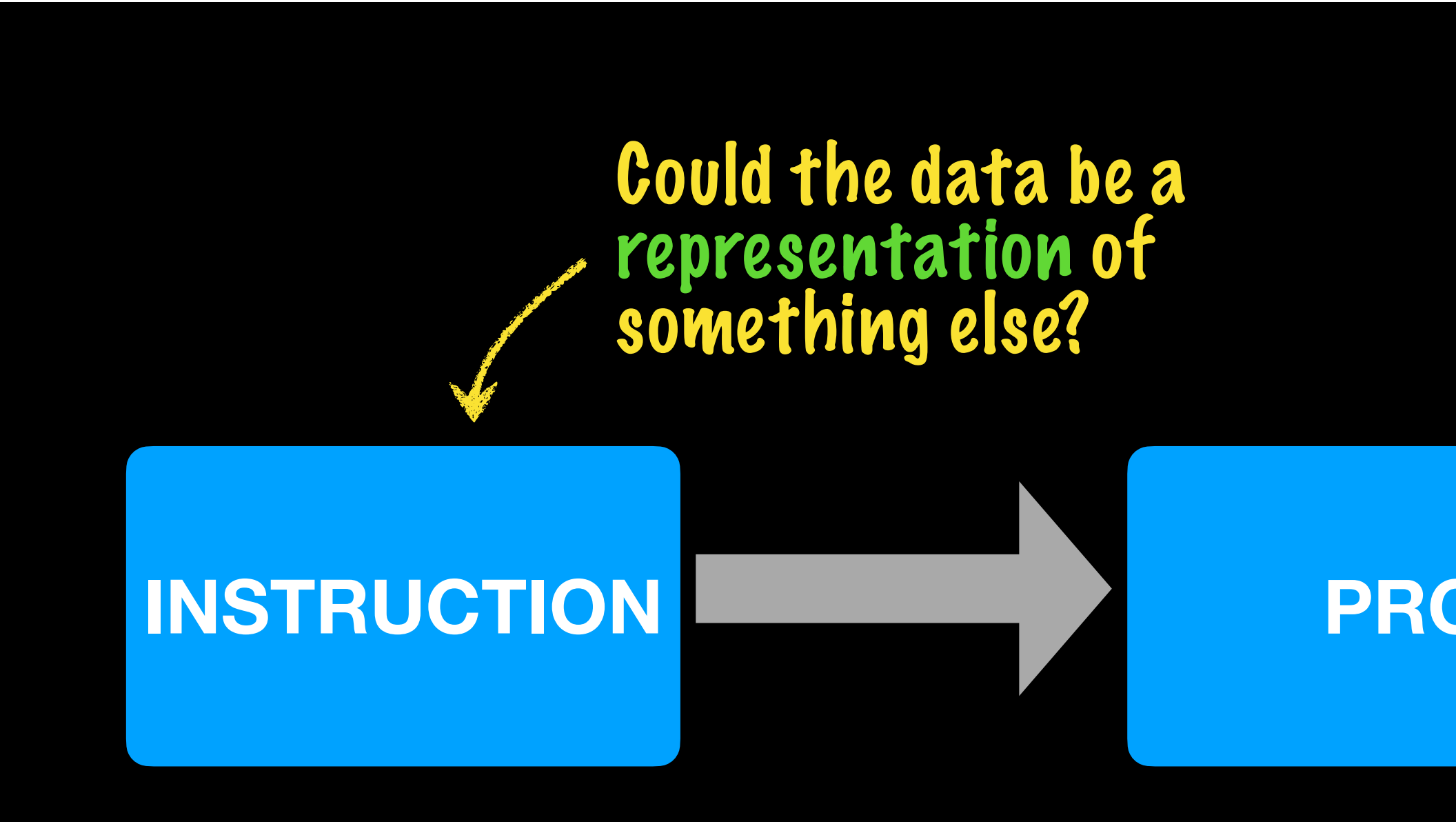
What was Turing's breakthrough?




Could the data be interpreted as something else?
DATA
PRO

Could the data be a representation of something else?




Could the data be a representation of something else?
INSTRUCTION
PRO

An "automatic machine"

instruction table

State	Symbol	New Symbol	New State	Move
A	1	0	A	RIGHT
A	0	1	A	RIGHT

A

current state

infinite strip of tape

0	1	0	0	1	1	0	0	*

State	Symbol	New Symbol	New State	Move
A	1	0	B	RIGHT
A	0	1	B	RIGHT
B	0	0	A	RIGHT
B	1	1	A	RIGHT

B

0	0	0	1	0	0	1	1	*

State	Symbol	New Symbol	New State	Move
A	0	0	A	RIGHT
A	1	1	A	RIGHT
A	*	*	B	LEFT
B	0	1	B	LEFT
B	1	0	C	LEFT
C	1	1	C	RIGHT
C	0	0	C	RIGHT

B

0	0	0	0	1	0	1	1	*

State	Symbol	New Symbol	New State	Move
A	0	0	A	RIGHT
A	1	1	A	RIGHT
A	*	*	B	LEFT
B	0	1	B	LEFT
B	1	0	C	LEFT
C	1	1	C	RIGHT
C	0	0	C	RIGHT

A

What if the instruction table above was just another set of input?

0	0	0	0	1	1	0	0	*