

Houzair Koussa

Software Engineer

github.com/houzyk

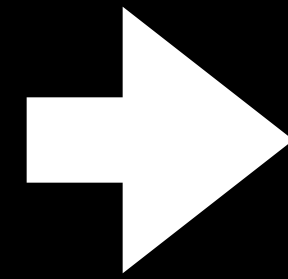
linkedin.com/in/houzairk (Houzair Koussa)

Philosophy Meets Code

Exploring the intersection of philosophy & computation

Disclaimer

**Scratching the
surface**



Left some gaps

Agenda

1. Context
2. Framework
 - Foundation
 - Two methods - argumentation + inquisitive allegory
3. Method 1 in action - unsolvable problems
4. Method 2 in action - Grelling's paradox
5. Q & A

Context

Philosophy & mathematics - maths is like a Ferrari...

The mathematician drives it, we look under the hood

Goal - conceptually break down maths itself

What is a number ?

Why is $1 + 1 = 2$? ...

Context

Philosophy & computation - code is like a Ferrari...

The engineer drives it, we look under the hood

Goal - *conceptually break down computation/code itself*

... and ask cool questions

Framework

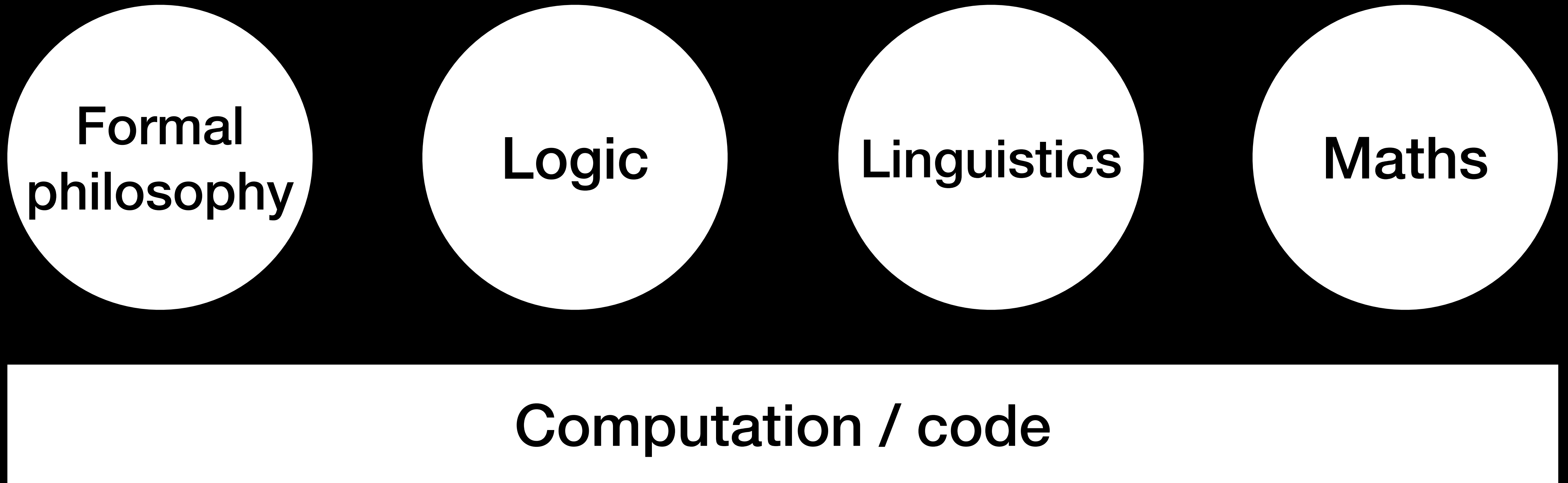
Philosophy is not about our opinions

It's meticulous, rigorous & systematic

Just like with web development, we require a framework

Theoretical framework = foundation + methods

Foundation



Bit like snapchat lenses

Method 1 - argumentation

Core idea - we ask a philosophical question. We answer it through a *semi-formal* argument

P1. }
P2. } Justify these premises *with other arguments*

...

C. The conclusion *logically follows* from the premises

The grandfather paradox

An example of argumentation

Question: Can we travel back in time?

P1. Assume that backward time travel is possible

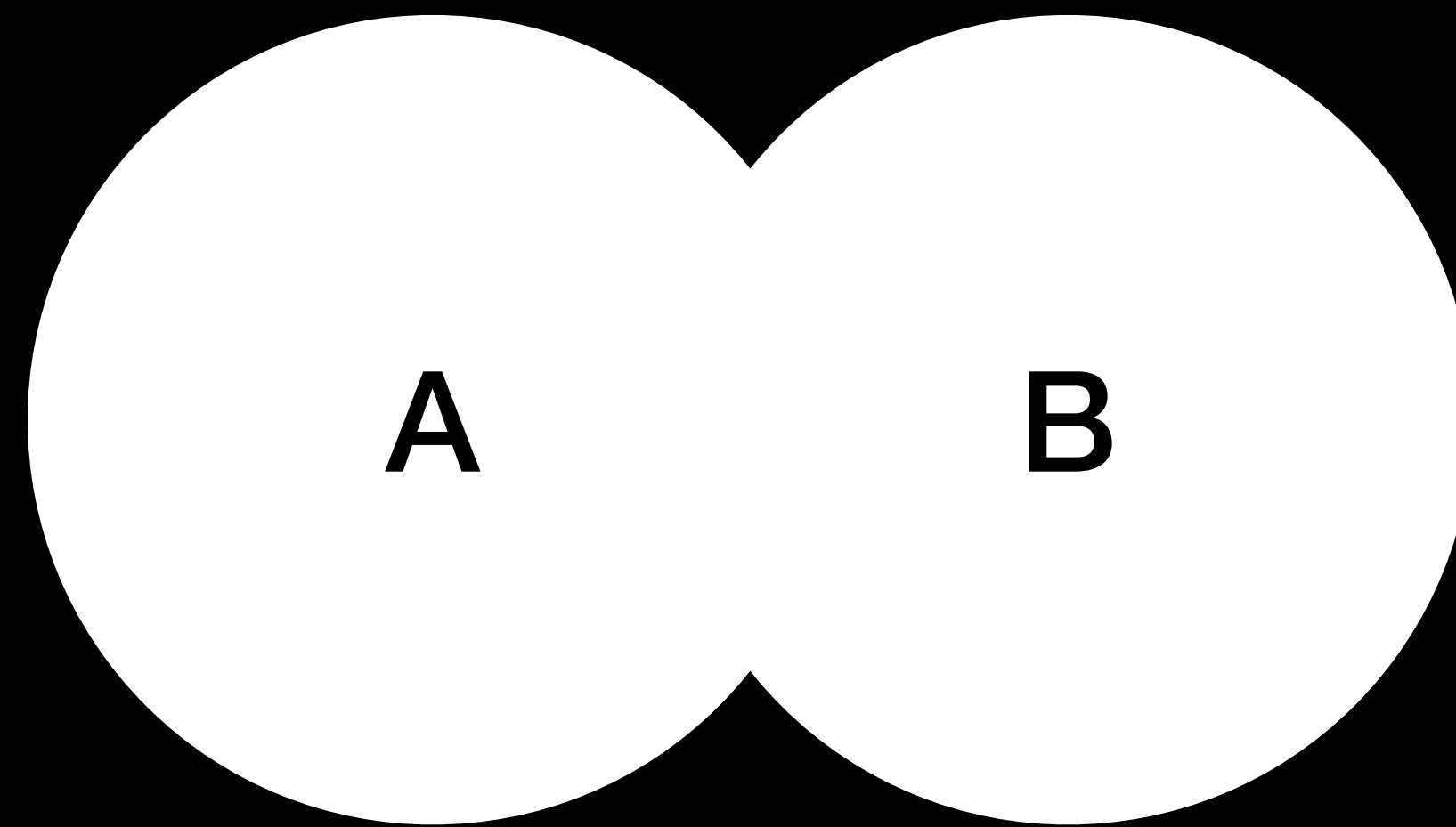
P2. If time travel is possible, we can kill our grandpa as a baby

P3. If our baby grandpa is killed, we can't kill him

C. We can & can't kill our grandpa. So, backward time travel is impossible

Method 2 - inquisitive allegory

Core idea - we draw parallels between two concepts



We contrast, compare & analyse. And ask questions

Trivially, just like life and lemons...

Our framework so far

1. Argumentation

2. Inquisitive allegory

Foundation - formal philosophy, logic, linguistics, maths

Conceptually break down computation/code itself

Unsolvable problems

Argumentation in action

Foundation - ideas from maths + logic

Question: Are there *problems* that a computer can't solve?

There is the halting problem but we wanna generalise

Arguing for unsolvable problems

P1. Amount of programs \leq amount of natural numbers

P2. Amount of problems = amount of real numbers

P3. Amount of natural numbers $<$ amount of real numbers

C. There are problems that a computer can't solve

P1. Amount of programs \leq amount of natural numbers

We gotta count all computational programs

a. JS is Turing complete

Theoretically, anything computable can be written in JS

So, counting all JS programs is like counting all programs

b. Assign a *unique* natural number to each character in JS

c \rightarrow 1

o \rightarrow 2

n \rightarrow 3

s \rightarrow 4

t \rightarrow 5

....

const \rightarrow 12345

A JS program \approx a unique natural number. Hence,

Amount of programs \leq amount of natural numbers

P2. Amount of problems = amount of real numbers

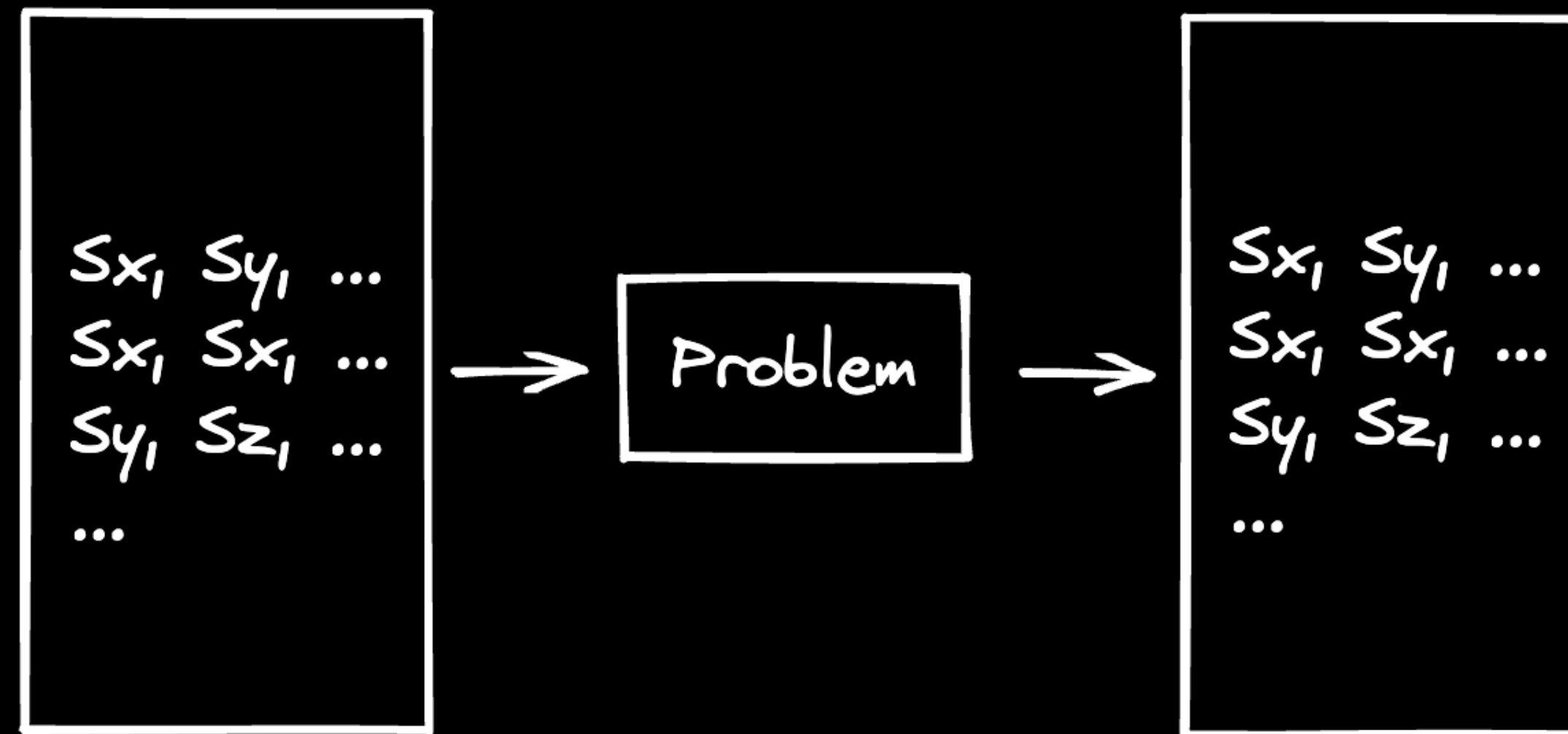
We gotta conceptualise & count computational problems

a. Theoretically, all problems are search problems

A problem is like a black-box of inputs & outputs. *To solve is to search for a program matching inputs & outputs*



b. Inputs & outputs are arbitrary & potentially infinite



All problems \approx all mappings from an infinite set to another

Lemma: amount of such mappings = amount of real numbers. Hence,

Amount of problems = amount of real numbers


P3. Amount of natural numbers < amount of real numbers

We gotta visualise Cantor's proof

a. List all natural numbers in a table

Natural Numbers	
0	
1	
2	
3	
...	

To infinity
and beyond!



b. Attempt to list a unique real number for each natural number

Natural Numbers	Real Numbers
0	0 . 0 0 0 0 0 0 0 0 1 ...
1	0 . 0 0 0 0 0 0 0 1 1 ...
2	0 . 0 0 0 0 0 0 1 1 1 ...
3	0 . 0 0 0 0 0 1 1 1 1 ...
...	...

To infinity
and beyond!
↓

c. Draw a diagonal through each real number - one unique digit place per real number

Natural Numbers	Real Numbers
0	0 . 0 0 0 0 0 0 0 0 1 ...
1	0 . 0 0 0 0 0 0 0 1 1 ...
2	0 . 0 0 0 0 0 0 1 1 1 ...
3	0 . 0 0 0 0 1 1 1 1 ...
...	...

To infinity
and beyond!

d. Rule - for each digit, if it's 9, subtract 1 else add 1

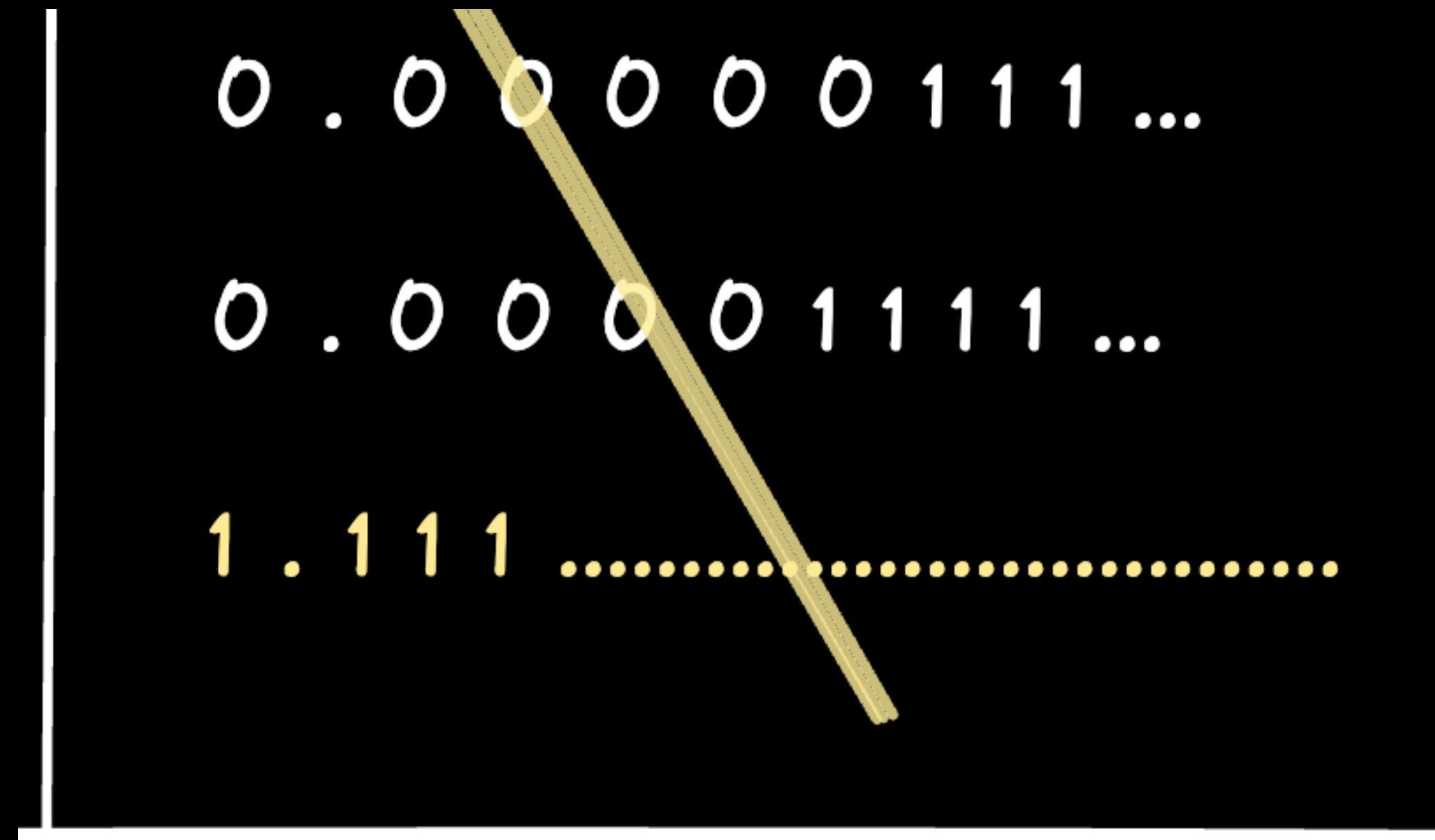
Natural Numbers		Real Numbers	
0		0	. 0 0 0 0 0 0 0 0 1 ...
1		0	. 0 0 0 0 0 0 0 1 1 ...
2		0	. 0 0 0 0 0 0 1 1 1 ...
3		0	. 0 0 0 0 1 1 1 1 ...
...		...	

To infinity
and beyond!

1 . 1 1 1

We get a “new” real number which is nowhere else on our *infinite* list

Importantly, we can apply our rule recursively. We'll keep getting “new” real numbers nowhere else on our *infinite* list



1 . 1 1 1x

Hence, *amount of natural numbers* < *amount of real numbers*

Back to our argument

P1. Amount of programs \leq amount of natural numbers

P2. Amount of problems = amount of real numbers

P3. Amount of natural numbers $<$ amount of real numbers

So, amount of programs $<$ amount of problems

Since *a program solves a problem*, there are more problems than programs available to solve them

Hence, there are problems that a computer can't solve

In retrospect

We broke down computation with ideas from maths & logic

Particularly, we answered a question by an argument

We justified each premises & its conclusion followed

We proved that *there is an infinity bigger than infinity* & reasoned about problems

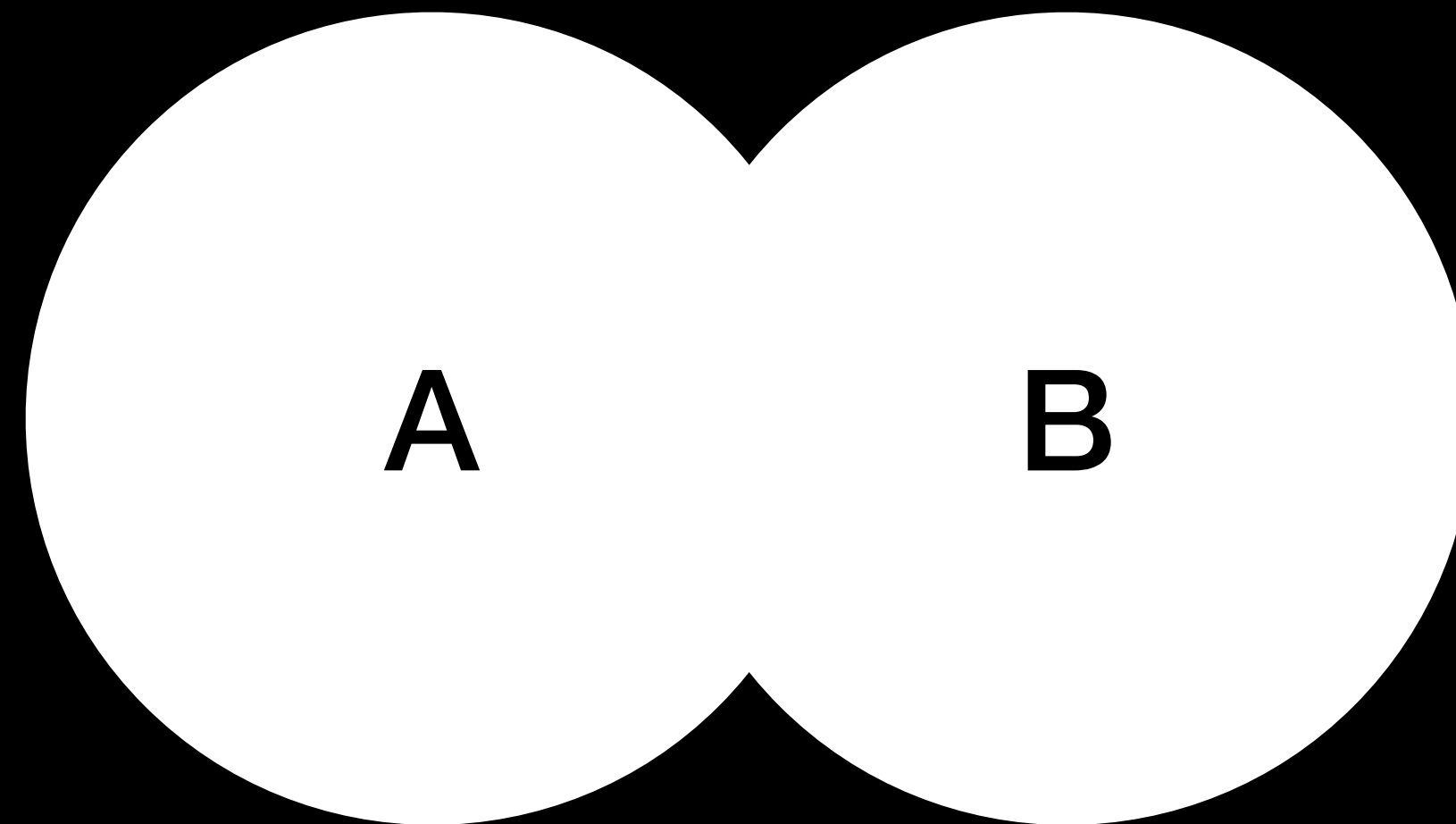
Just a thought - exponentially infinite super structures!

```
// break = 1 minute ;
```

Quick recap - inquisitive allegory

We want to draw parallels between two concepts

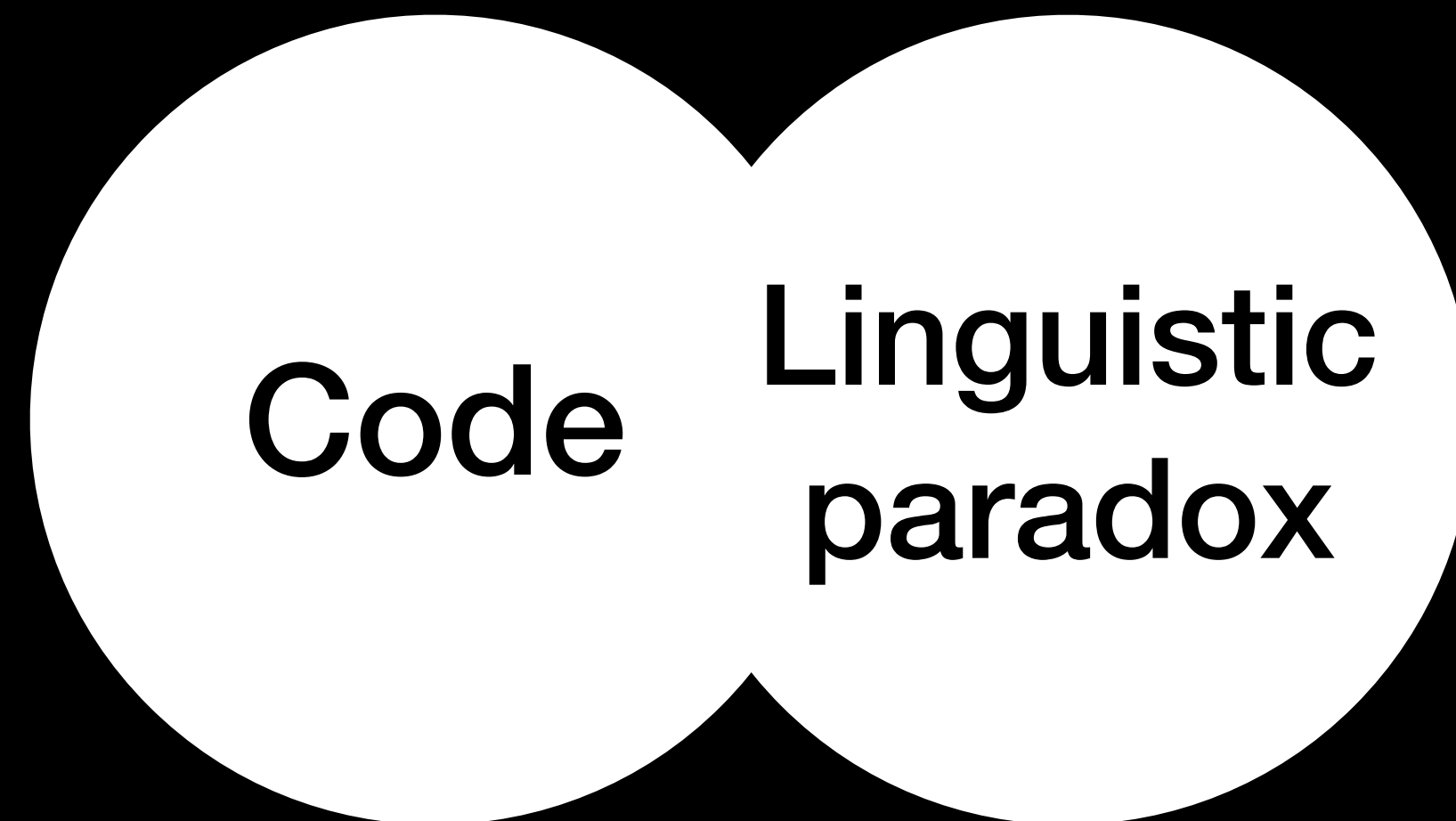
To contrast, compare & analyse. And ask cool questions



Grelling's paradox

Inquisitive allegory in action

Foundation - ideas from linguistics + logic



Two innocent looking words - “autological” & “heterological”

“Autological”

A word is autological if it describes itself

Examples:

“polysyllabic” - a word with multiple syllables

“lowercased” - a word with lowercased letters

“english” - a word which is in the English language

“Heterological”

A word is heterological if it is *not autological*, i.e. it does not describe itself

Examples:

“monosyllabic” - a word with just one syllable

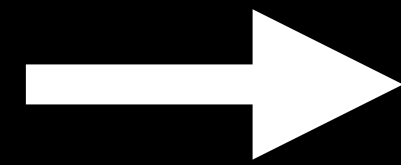
“uppercased” - a word with uppercased letters

“duck” - an animal used for debugging

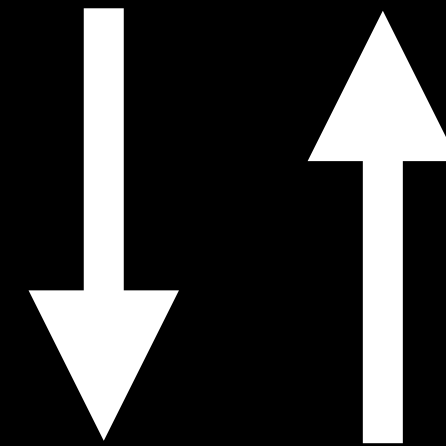
... And contradiction!

Is the word “heterological” autological or heterological?

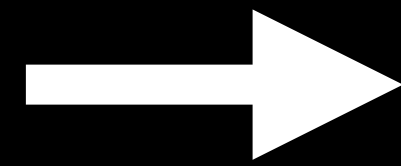
Autological



It describes itself. So, it does not describe itself.



Heterological



It does not describe itself. So, it describes itself.

Parallel with code

An autological function is a function that returns itself

`anAutologicalFunc` \Rightarrow `anAutologicalFunc`

Not to be confused with a recursive function

`aRecursiveFunc` \Rightarrow `aRecursiveFunc()`



A heterological function does not return itself

The predicate “is autological” is a function `isAutological()` which returns whether a function returns itself

`isAutological = (Func, ...) \Rightarrow boolean`

So, the predicate “is heterological” is a function that returns the negation of `isAutological()`.

... And bug ?

“is autological”	<code>isAutological = (Func, ...) \Rightarrow boolean</code>
“is heterological”	<code>isHeterological(...) \Rightarrow !isAutological(...)</code>
Where's the paradox ?	<code>isAutological(isHeterological) isHeterological(isHeterological)</code>

What kinda cool questions can we ask?

Meta question - can we *meaningfully* draw such parallels between code & logical paradoxes?

Will an infinite amount of functions as params fix our bug?

If yes, will `isAutological` or `isHeterelogical` even return?

In retrospect

We broke down computation with ideas from linguistics & logic

We drew a parallel between Grelling's Paradox and code

We came across cool questions

Just a thought - Grelling's paradox & the halting problem share the *same* logical core. Self-reference!

PS

Blog - github.com/houzyk/thephilosophicalcode

Slides - github.com/houzyk/devcon-2023-philosophy-meets-code

Thank You!