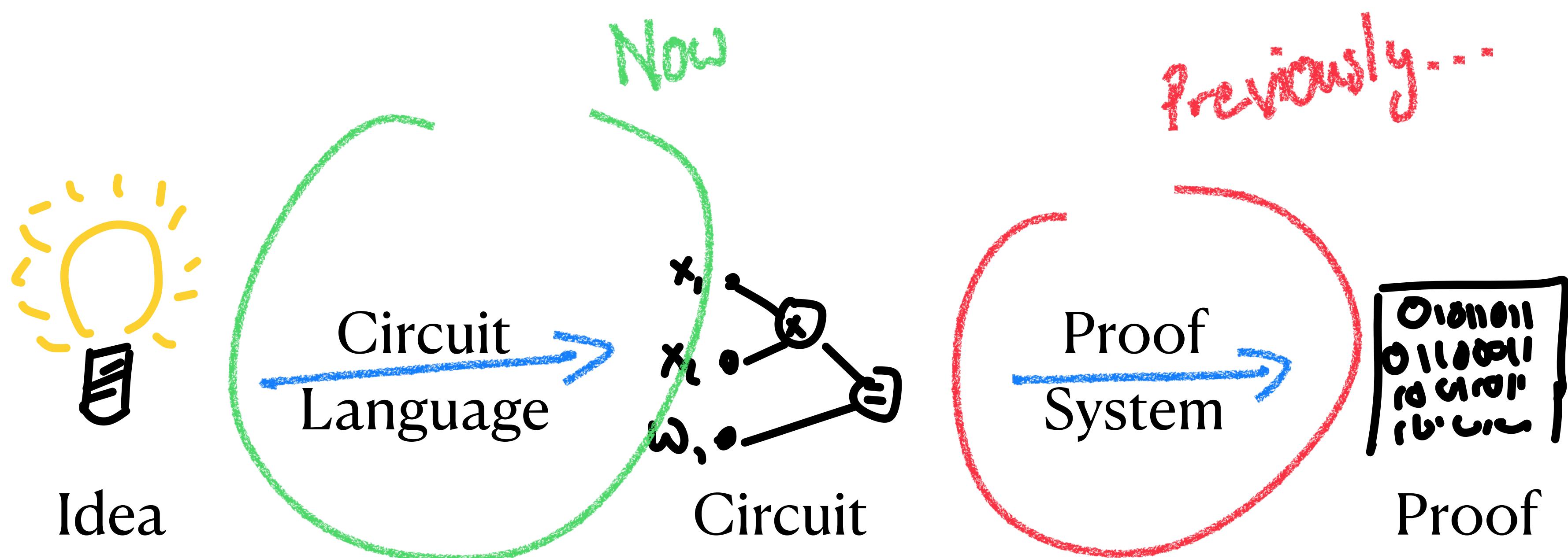


A Taxonomy of Circuit Languages

Note:
If you notice an
error in this talk's
material, please let
Alex know.

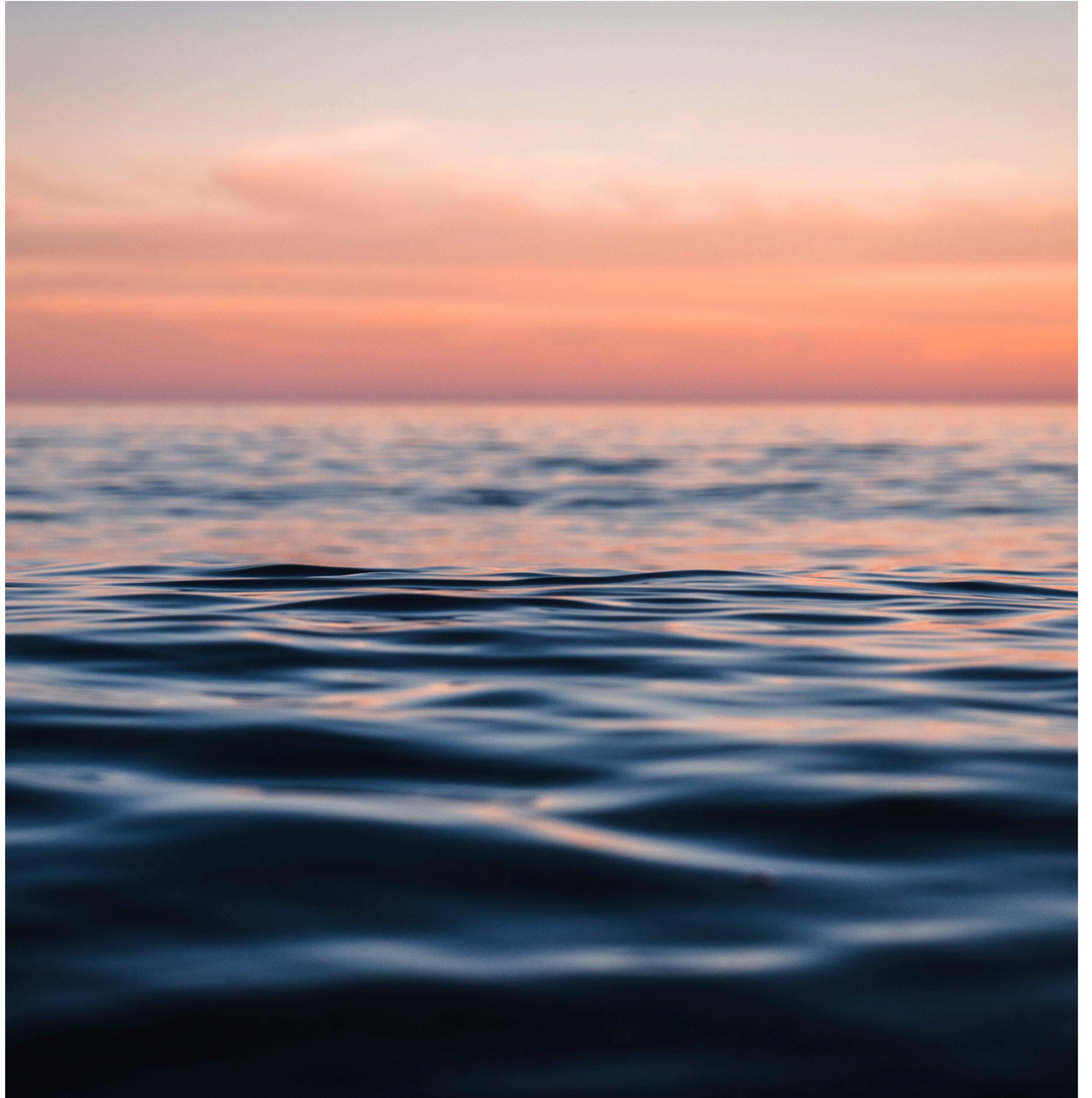
Alex Ozdemir, "ZK Hack" 2021

Core Problem

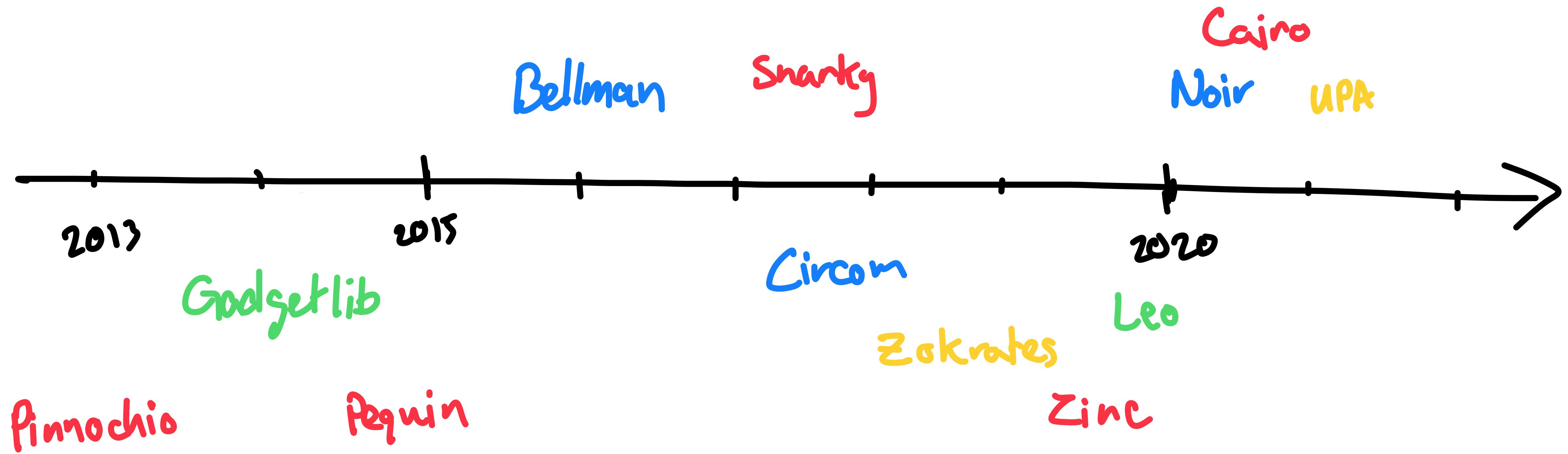


A Cambrian Explosion

2013 - present



Timeline



Circuit Target

R1CS

single circuit
unit-cost ✗
'free' +

Plonk

single circuit
unit-cost ✗
unit-cost +
custom gates
verifier randomness

AIR

repeated circuit
unit cost ✗
'free' +

True "circuits"

RAM-like

Circuit Target

R1CS

bellman

gadgetlib

pequin

Zokrates

snarky

circom

Leo

Zinc

Plonk

Noir

UPA

...

AIR

Cairo

Air Script

Distaff
Assembly

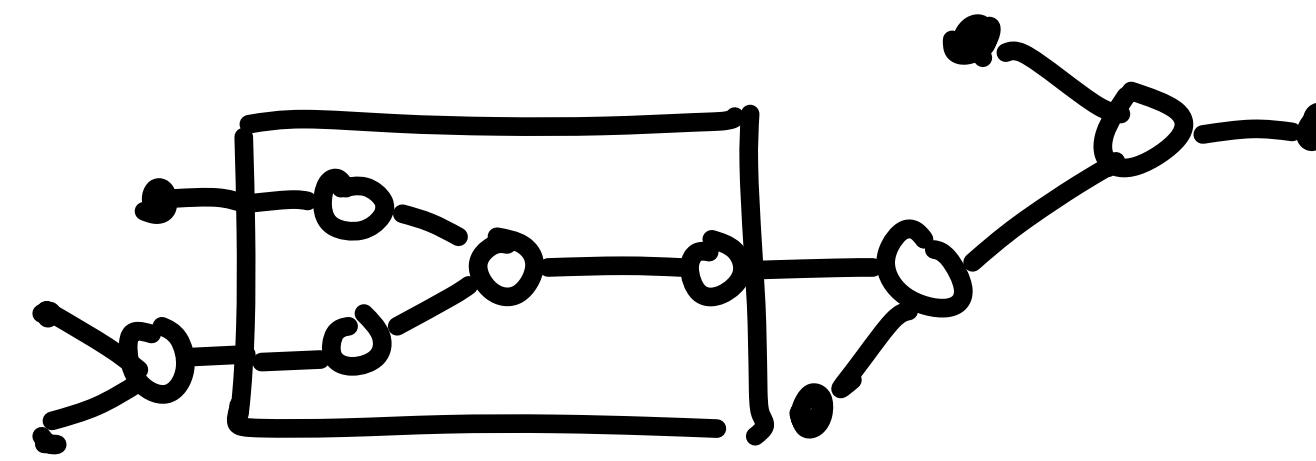
Air
Assembly

Language Type

Library

```
circ.add_wire(...)  
circ.add_gate(...)
```

HDL



(Verilog)

PL

```
fn main(...){  
...  
}
```

(Rust)

Language Type

Library

bellman

gadgetlib

Snarky

UPA

HDL

circum

PL

Zokrates

pequin

Zinc

Noir

Leo

Cairo

Feature: Mutable Variables

```
1 fn add(x: u64, y: u64) -> u64 {  
2     let mut sum = x;  
3     sum = sum + y;  
4     sum|  
5 }
```

mutating sum

Feature: Mutable Variables

No

bellman

gadgetlib

snarky
circom

UPA

Yes

Leo

Zinc

Cairo

pequin

Zokrates

Noir

Feature: Primitive Types

```
1 ▶ fn main() {  
2     let x: bool = true;  
3     let y: u32 = 5;  
4     let z: f64 = 5.0e-1;|  
5 }
```

Feature: Primitive Types

Field

Circom

Cairo

Bool

bellman

gadgetlib

UPA

Machine
Integers

snarky zokrates

pequin Zinc

Leo Noir

Feature: If-Statements

```
1 fn pow(mut base: i64, exp: bool) -> i64 {  
2     if exp {  
3         base = base * base;  
4     }  
5     base  
6 }
```

conditional branch
with side-effects

Feature: If-Statements

No

bellman

snarky

gadgetlib

circum

Zinc

Zokrates

UPA

Noir

Yes

Leo

pequin

Cairo

Feature: User Structures

```
1 struct CompressedPoint<F: Field> {  
2     x: F,  
3     neg_y: bool,  
4 }
```

Feature: User Structures

No	From Host Language	Yes
Noir	bellman	pequin
Circum	gadgetlib	Zokrates
	snarky	Zinc Cairo
	UPA	Leo

Feature: Arrays

```
1 fn foo(a: [u32; 4], i: usize) -> u32 {  
2     let x = a[i];  
3     a[i] = x + x;  
4     x  
5 }
```

variable-index
write

variable-index
reach

The diagram illustrates the scope of variable indexing for both reads and writes. A code snippet is shown in a terminal window:

```
1 fn foo(a: [u32; 4], i: usize) -> u32 {  
2     let x = a[i];  
3     a[i] = x + x;  
4     x  
5 }
```

A blue bracket under the line "x" at index 4 indicates its "variable-index write" scope. A red bracket under the line "a[i]" at index 3 indicates its "variable-index read" scope, which extends to the end of the function body. Handwritten annotations "variable-index write" and "variable-index reach" are placed near the respective brackets.

Feature: Arrays

No Arrays

bellman

gadgetlib

UPA

Constant
Indices

Circos

Zinc

Noir

Leo

Linear
Scans

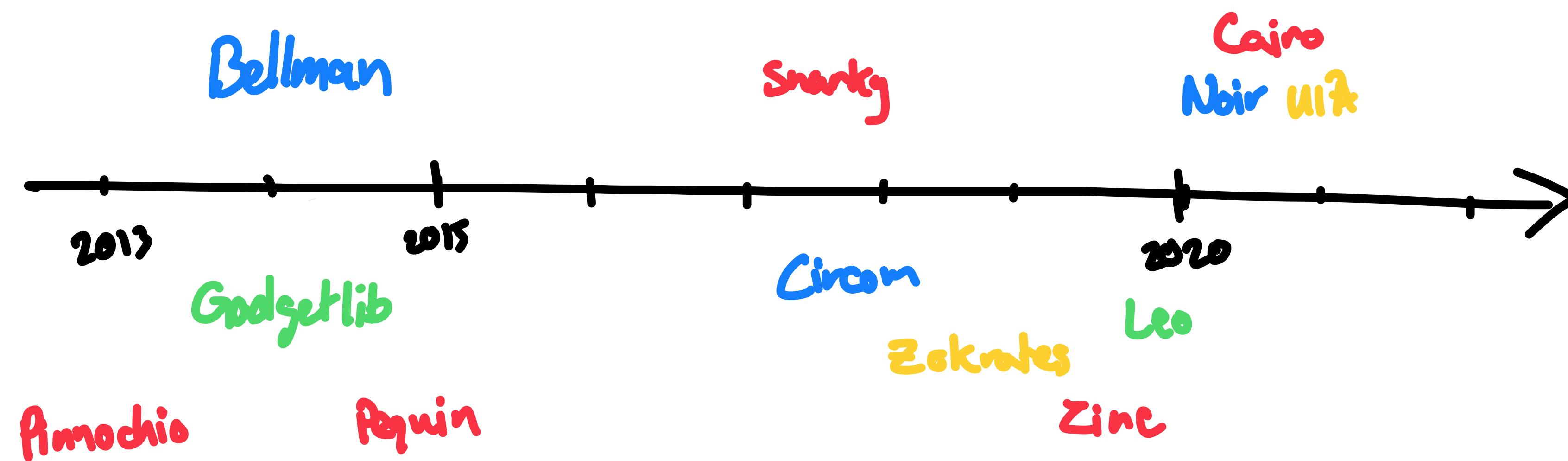
Zokrates

Better

pequin

Cairo

The Future of Circuit Languages



1. We need to continue to explore

The Future of Circuit Languages

```
let x = a[i];  
a[i] = x + x;
```

x

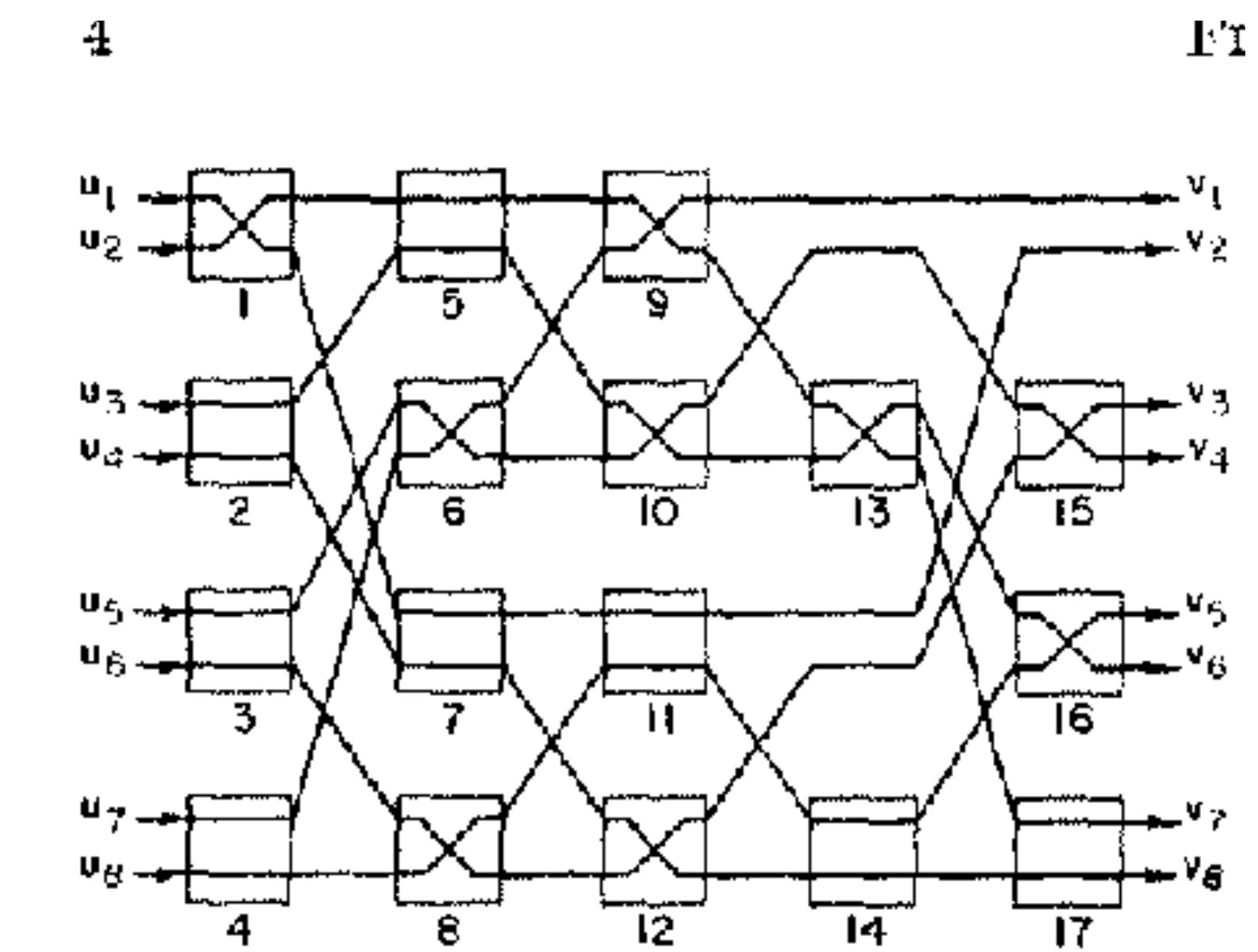
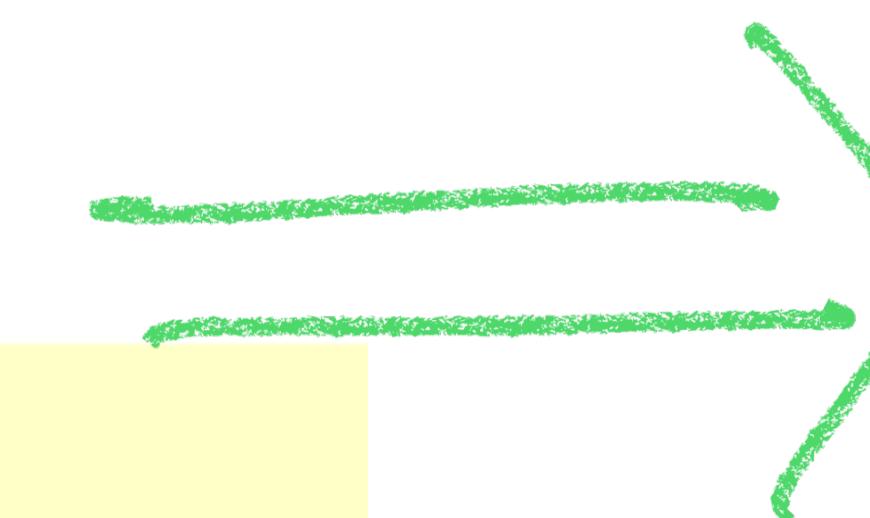
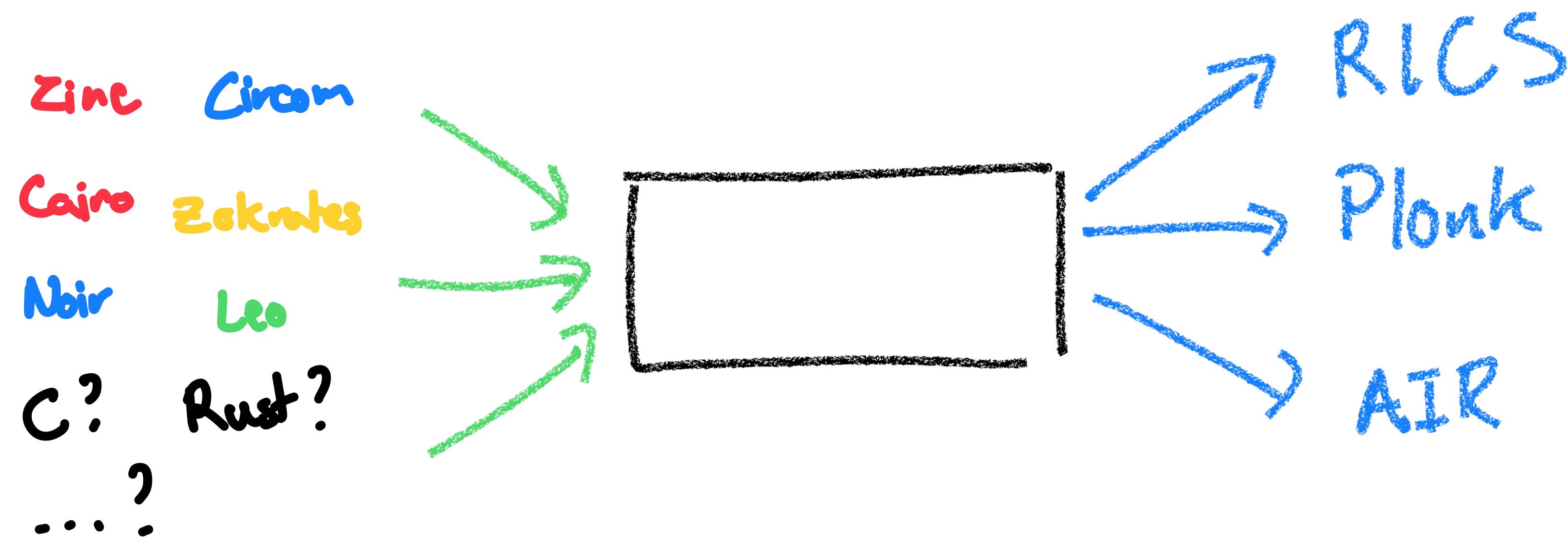


FIG. 6

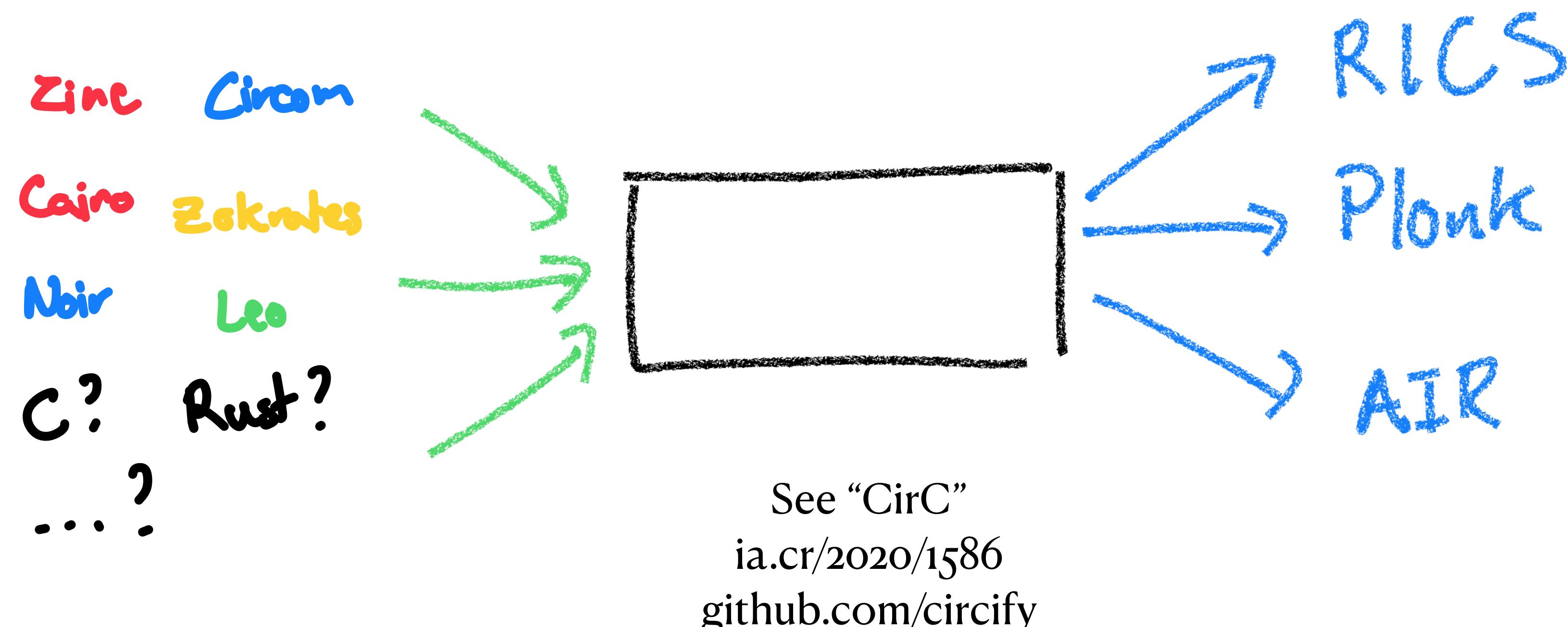
2. The compilation problem is hard

The Future of Circuit Languages



3. Perhaps we can share infrastructure?

The Future of Circuit Languages



3. Perhaps we can share infrastructure?

The Future of Circuit Languages

