TOWARDS ADDING VARIETY TO SIMPLICITY

CHALMERS
UNIVERSITY OF TECHNOLOGY

ELISABET LOBO

NACHIAPPAN VALLIAPPAN

ALEJANDRO RUSSO

SOLENE MIRLIAZ

rennes

MOTIVATION

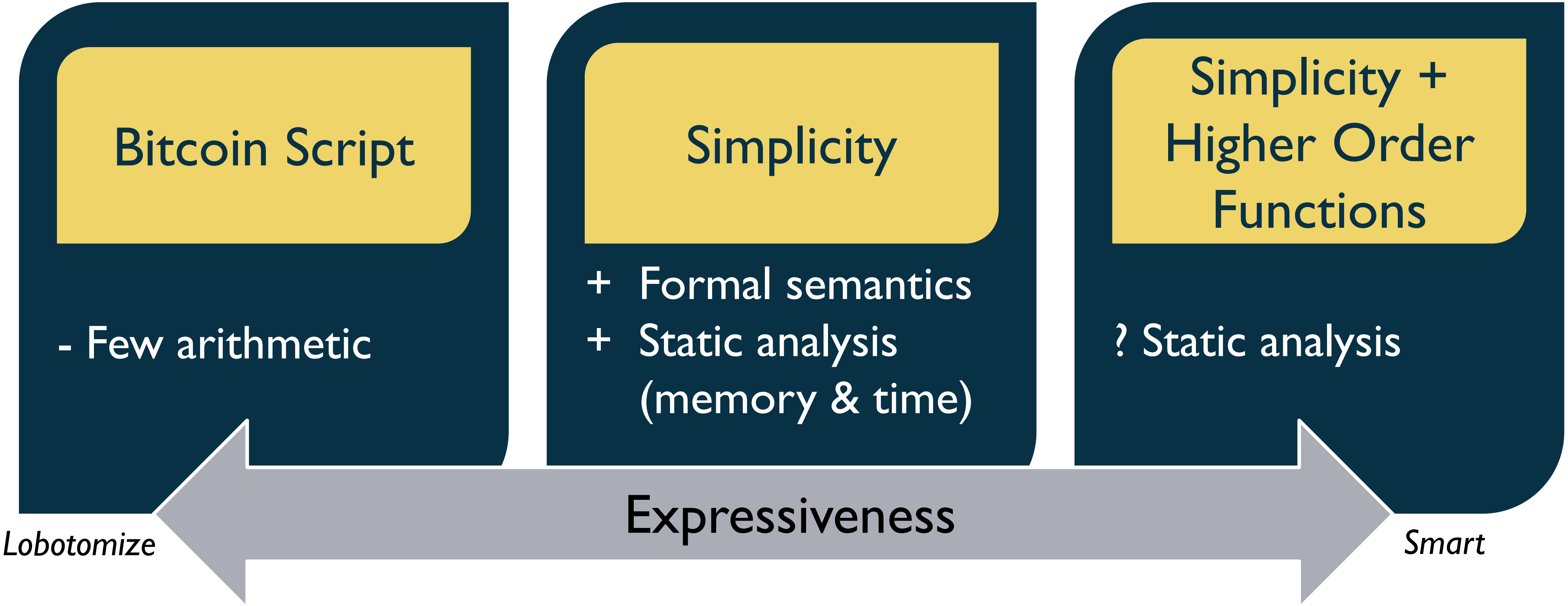
Icon made by Smashicons from www.flaticon.com

BITCOIN SCRIPT LANGUAGE

A2B.bitScript

- Stack-based
- Not Turing-complete
- No loops
- Conditionals
- Hashing and digital signature verification

BITCOIN TRANSACTIONS AS CONTRACTS



Bitcoin Script

- Few arithmetic

Simplicity

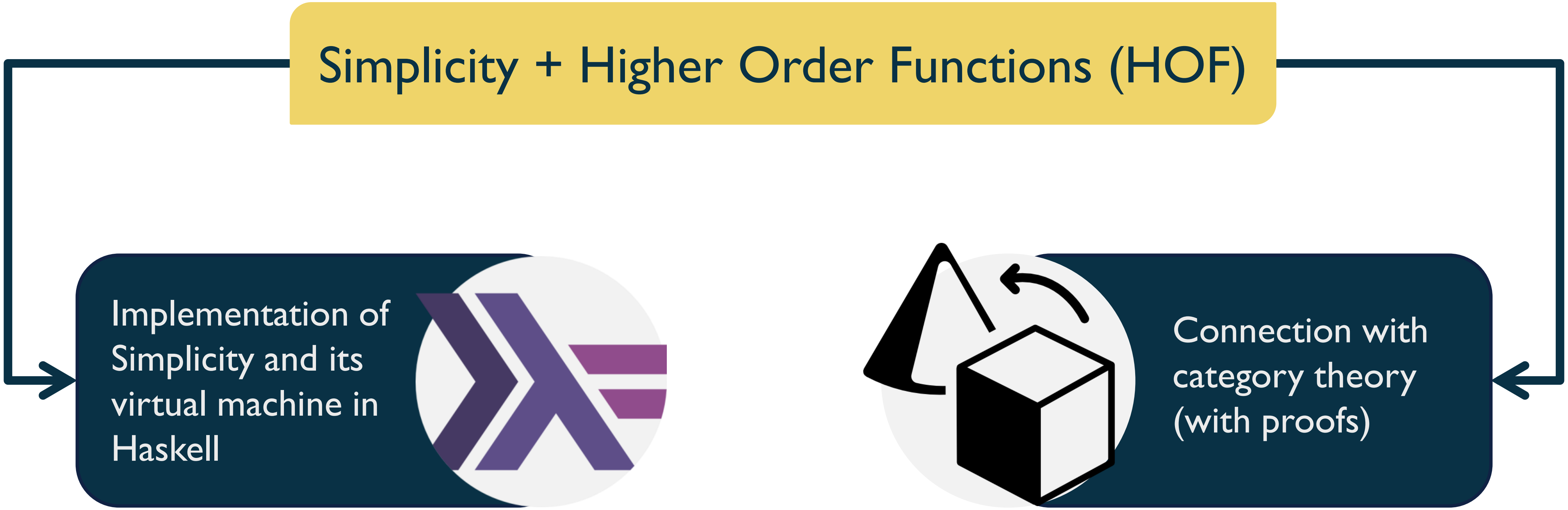
+ Formal semantics
+ Static analysis (memory & time)

Simplicity + Higher Order Functions

? Static analysis

Lobotomize Expressiveness *Smart*

CONTRIBUTIONS



Simplicity + Higher Order Functions (HOF)

Implementation of Simplicity and its virtual machine in Haskell

Connection with category theory (with proofs)

https://bitbucket.org/russo/isola-additional-material/overview

SIMPLICITY

Icon made by Freepik from www.flaticon.com

TYPED COMBINATOR LANGUAGE

Types

- Unit $= \mathbb{1}$
- Products $= A \times B$
- Coproduct $= A + B$

```
data Unit
data a :*: b
data a :+: b
```

Combinators

$$\mathbf{prog} : \mathbf{A} \vdash \mathbf{B}$$

“Program **prog** *has input type* **A** *and output type* **B***”*

- **unit** : $\mathbf{A} \vdash \mathbb{1}$
- **iden** : $\mathbf{A} \vdash \mathbf{A}$
- …

```
data Simpl a b where
  Unit :: Simpl a Unit
  Iden :: Simpl a a
  …
```

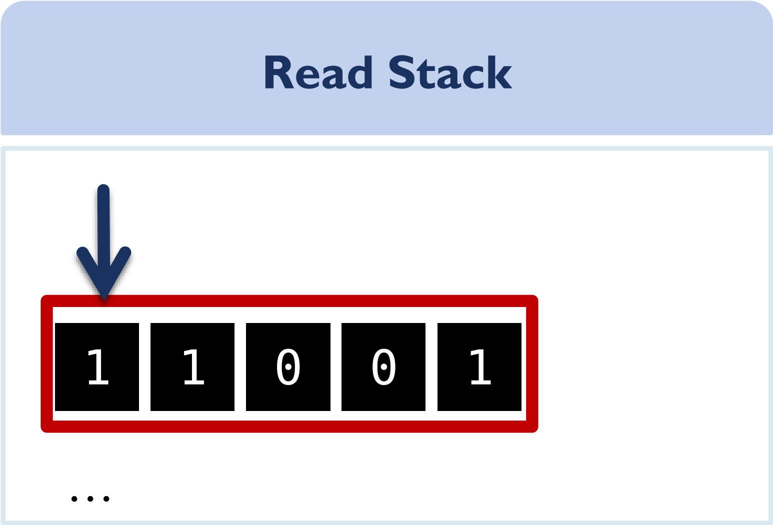
THE BIT MACHINE

Icon made by Puppets *www.flaticon.com*

EXECUTION OF TERMS

$$\mathbf{prog} : \boldsymbol{A} \vdash \boldsymbol{B}$$

EXECUTION OF TERMS



Read Stack

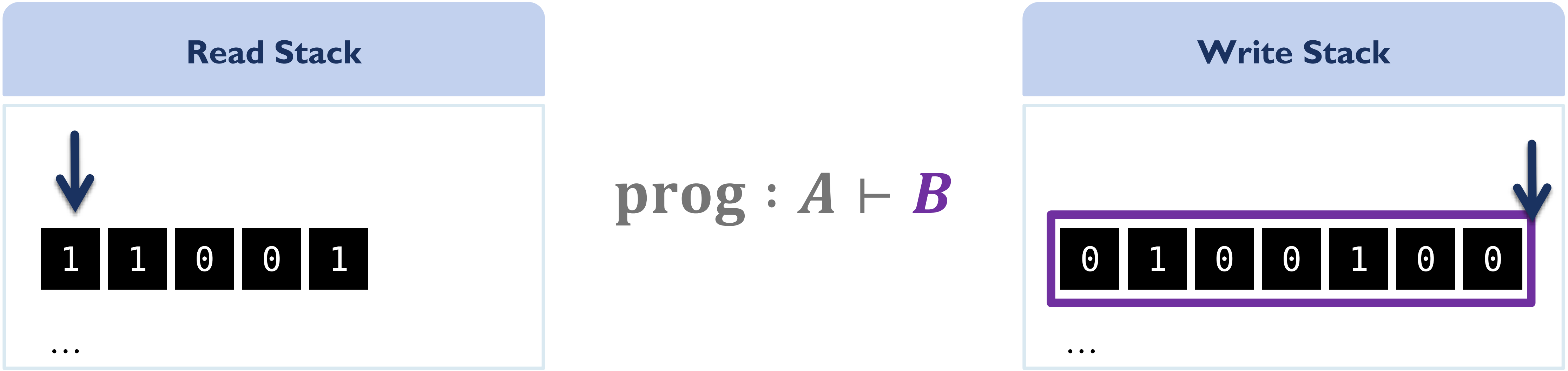
1	1	0	0	1

…

Before the execution

$$\mathrm{prog} : A \vdash B$$

EXECUTION OF TERMS



Read Stack

1 1 0 0 1

…

$$\mathbf{prog} : A \vdash B$$

Write Stack

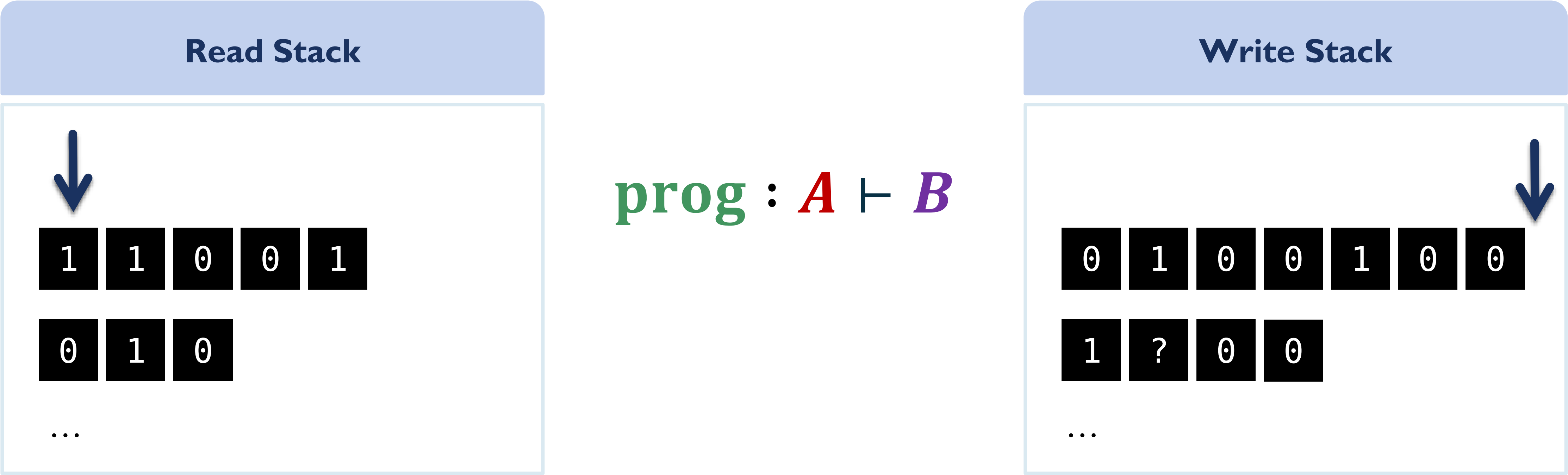
0 1 0 0 1 0 0

…

Before the execution

After the execution

EXECUTION OF TERMS



Read Stack

1 1 0 0 1

0 1 0

…

$\mathbf{prog} : A \vdash B$

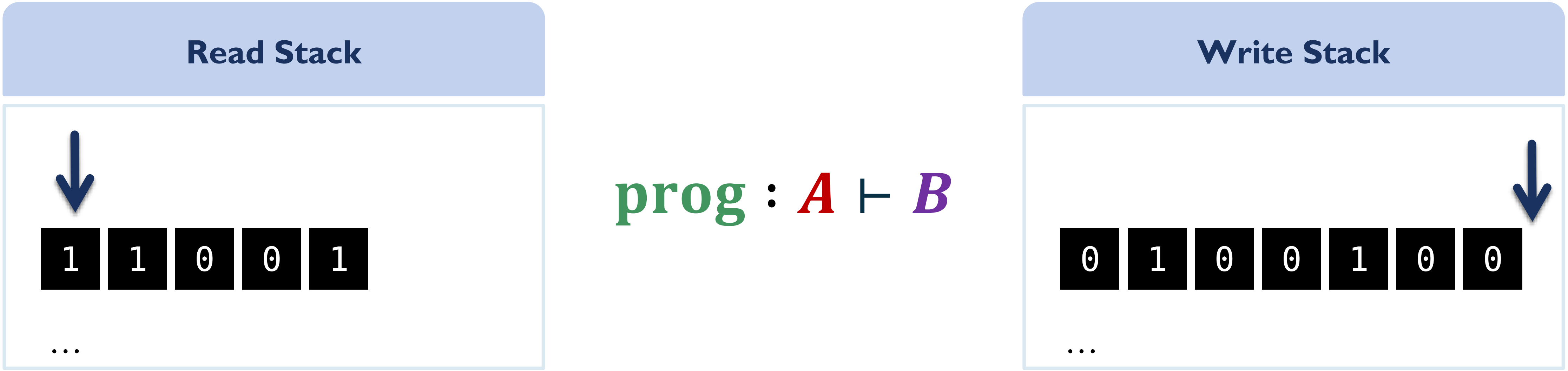
Write Stack

0 1 0 0 1 0 0

1 ? 0 0

…

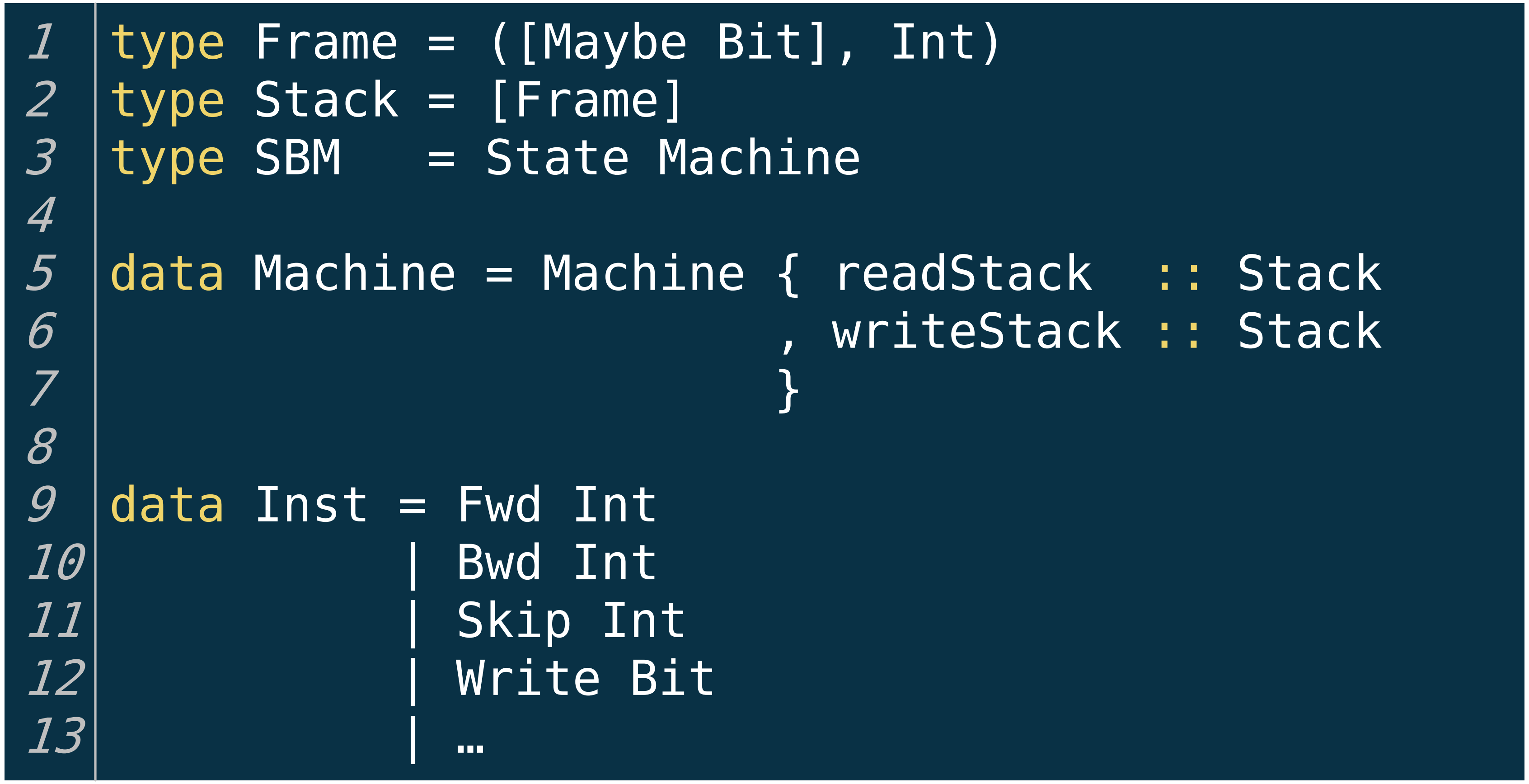
EXECUTION OF TERMS



$$\mathbf{prog} : A \vdash B$$

```
simpl2sbm :: Simpl a b → [Inst]
run :: [Inst] → SBM [Maybe Bit]
```

IMPLEMENTATION



```haskell
type Frame = ([Maybe Bit], Int)
type Stack = [Frame]
type SBM   = State Machine

data Machine = Machine { readStack  :: Stack
                       , writeStack :: Stack
                       }

data Inst = Fwd Int
          | Bwd Int
          | Skip Int
          | Write Bit
          | …
```

RESOURCE ALLOCATION

$$\mathbf{prog} : \boldsymbol{A} \vdash \boldsymbol{B}$$

$\mathrm{sizeOf}(\mathbf{A}) =$ *How many cells do we need to read*
$\mathrm{sizeOf}(\boldsymbol{B}) =$ *How many cells do we need to allocate*

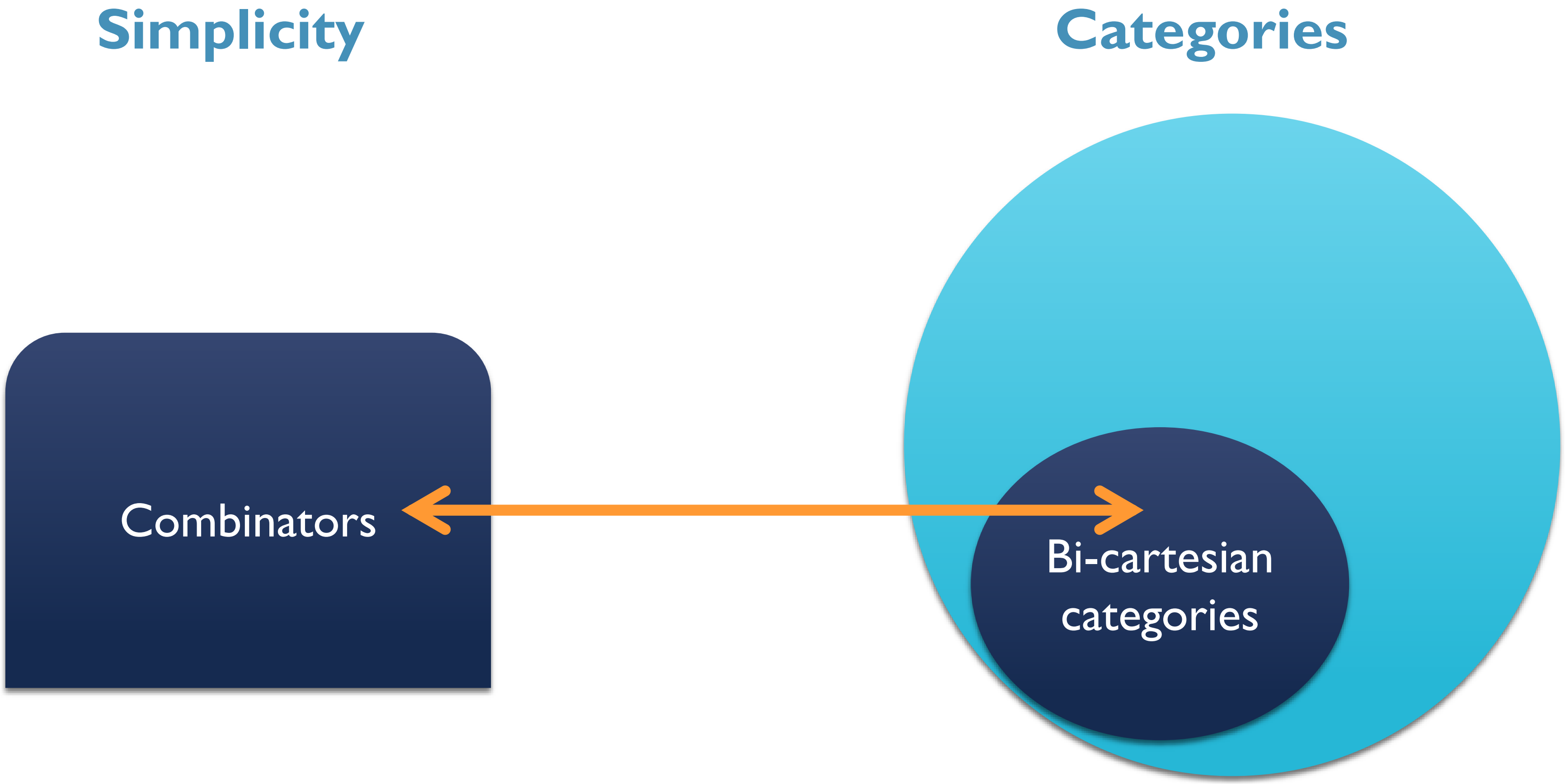
Well-typed programs have a finite representation in terms of cells

$$\mathrm{sizeOf}(\mathbb{1}) = 0$$
$$\mathrm{sizeOf}(A \times B) = \mathrm{sizeOf}(A) + \mathrm{sizeOf}(B)$$
$$\mathrm{sizeOf}(A + B) = 1 + \max(\mathrm{sizeOf}(A), \mathrm{sizeOf}(B))$$

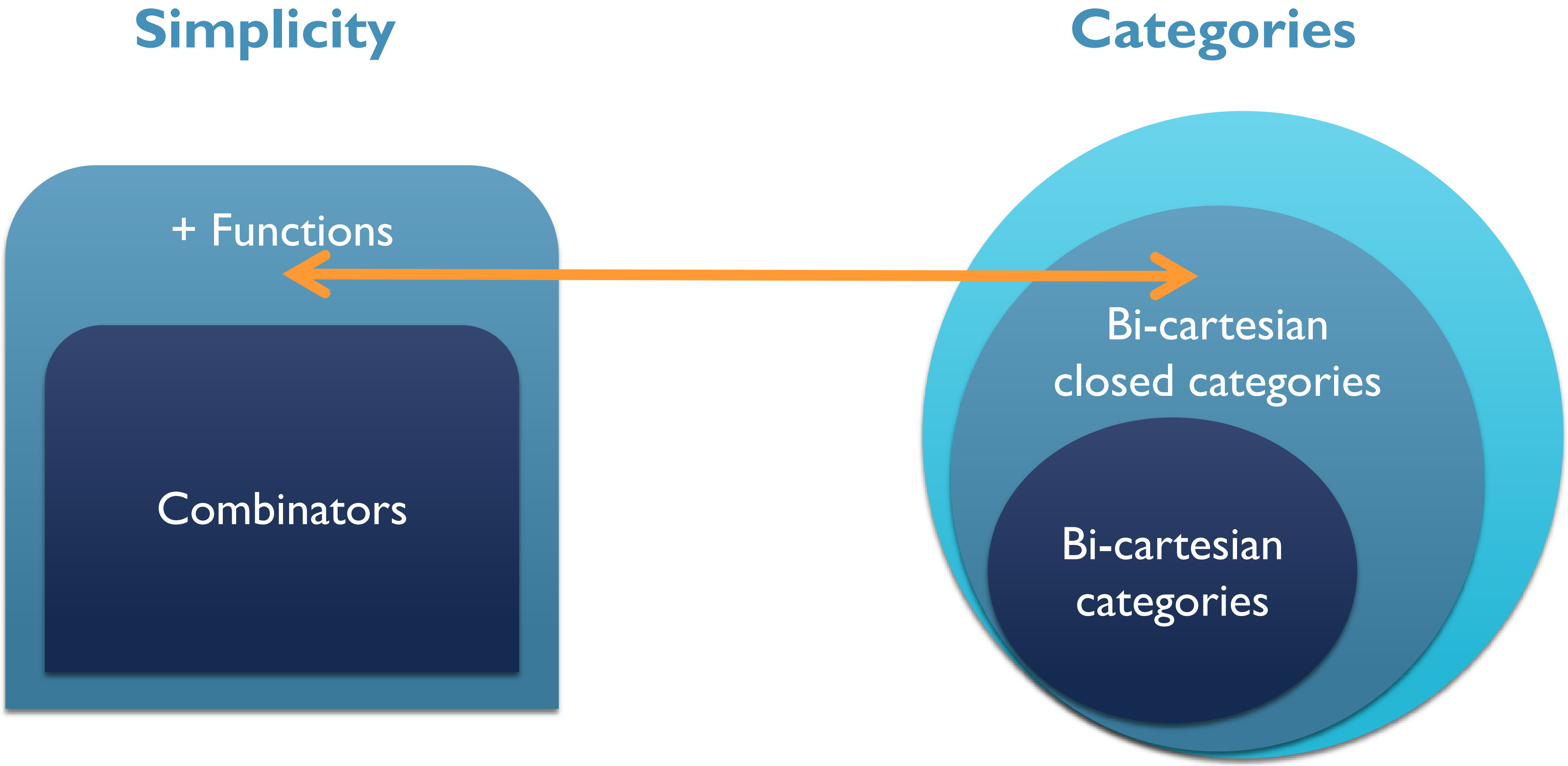
CATEGORIES

Icon made by Freepik from www.flaticon.com

ABSTRACTION



ABSTRACTION



EXTENDING THE LANGUAGE

Icon made by Freepik from www.flaticon.com

CHANGES IN SIMPLICITY

Types

- …
- Exponentials = $A \Rightarrow B$

Combinators

- …
- lam $(\mathrm{l} : \mathrm{R} \times A \vdash B) : \mathbf{R} \vdash \boldsymbol{A} \Rightarrow \boldsymbol{B}$
- app $(\mathrm{f} : \mathrm{R} \vdash \mathrm{A} \Rightarrow B)(\mathrm{x} : \mathrm{R} \vdash A) : \mathbf{R} \vdash \mathbf{B}$

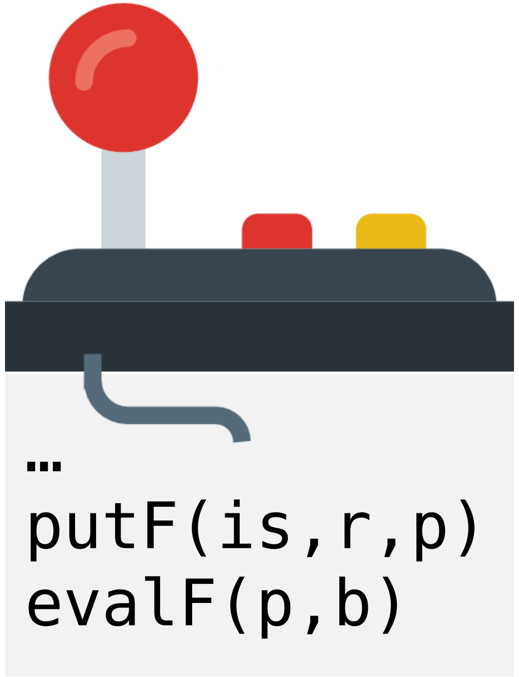
```
data a :=>: b

data Simpl a b where
  …
  Lam :: Simpl (r :*: a) b → Simpl r (a :=>: b)
  App :: Simpl r (a :=>: b) → Simpl r a → Simpl r b
```

CHANGES IN THE BITMACHINE

`prog :: Simpl r (a :=>: b)` $\xrightarrow{⚠}$ `is :: [Inst]`

$$\mathrm{sizeOf}(A \Rightarrow B) = ?$$

CHANGES IN THE BITMACHINE

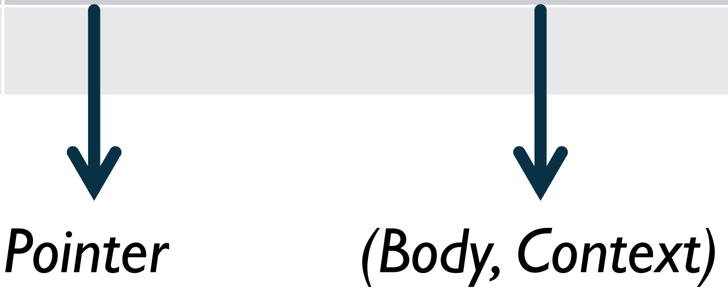


```
…
putF(is,r,p)
evalF(p,b)
```

```
data Inst = …
          | PutF [Inst] Int Int
          | EvalF Int Int
```

CHANGES IN THE BITMACHINE

Read Stack	Write Stack	List of Functions
[0 0 1 1 0 1]	[1 ? ? ? ? ?]	000 ([Fwd 3,…], [1 0 1])
[…]	[…]	



CHANGES IN THE BITMACHINE

Read Stack	Write Stack	List of Functions
[0 0 1 1 0 1]	[1 ? ? ? ? ?]	000 ([Fwd 3,…], [1 0 1])
[…]	[…]	

putF(is,r,p)

run (PutF [Write 1, …] 2 3)

CHANGES IN THE BITMACHINE

Read Stack	Write Stack	List of Functions
[0 0 1 1 0 1]	[1 ? ? ? ? ?]	000 ([Fwd 3,…], [1 0 1])
[…]	[…]	

```
run (PutF [Write 1, …] 2 3)
```

Read Stack	Write Stack	List of Functions
[0 0 1 1 0 1]	[1 0 0 1 ? ?]	000 ([Fwd 3,…], [1 0 1])
[…]	[…]	001

CHANGES IN THE BITMACHINE

Read Stack	Write Stack	List of Functions
`[0 0 1 1 0 1]`	`[1 ? ? ? ? ?]`	`000 ([Fwd 3,…], [1 0 1])`
`[…]`	`[…]`	

`run (PutF [Write 1, …] 2 3)`

Read Stack	Write Stack	List of Functions
`[0 0 1 1 0 1]`	`[1 0 0 1 ? ?]`	`000 ([Fwd 3,…], [1 0 1])`
`[…]`	`[…]`	`001 ([Write 1,…],`

CHANGES IN THE BITMACHINE

Read Stack	Write Stack	List of Functions
`[0 0 1 1 0 1]`	`[1 ? ? ? ? ?]`	`000 ([Fwd 3,…], [1 0 1])`
`[…]`	`[…]`	

↓ `run (PutF [Write 1, …] 2 3)`

Read Stack	Write Stack	List of Functions
`[0 0 1 1 0 1]`	`[1 0 0 1 ? ?]`	`000 ([Fwd 3,…], [1 0 1])`
`[…]`	`[…]`	`001 ([Write 1,…], [1,0])`

CHANGES IN THE BITMACHINE

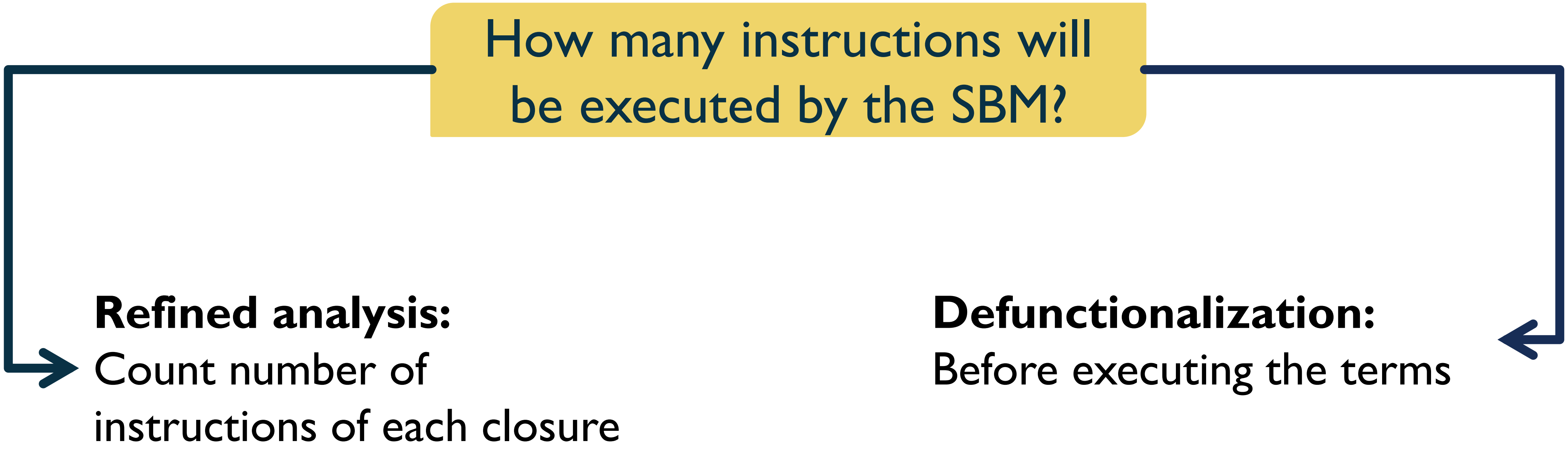
Read Stack	Write Stack	List of Functions
`[0 0 1 1 0 1]`	`[1 0 0 1 ? ?]`	`000 ([Fwd 3,…], [1 0 1])`
`[…]`	`[…]`	`001 ([Write 1,…], [1,0])`

↓ `run evalF(p,s)`

$$\mathrm{sizeOf}(A \Rightarrow B) = \mathrm{sizePtr}$$

$$\mathrm{sizePtr} = \log_2(total\_closures) + 1$$

FUTURE WORK: STATIC ANALYSIS



How many instructions will be executed by the SBM?

Refined analysis:
Count number of instructions of each closure

Defunctionalization:
Before executing the terms

FINAL REMARKS

- Glimpse of the implementation of Simplicity and its virtual machine in Haskell
- Build the intuition on how categories can model simplicity programs
- Exploit results from categories to add functions
- Change the bit machine keeping the invariant that sizeOf is finite

Simplicity + HOL:
`~115 LOC`

SBM + HOL:
`~250 LOC`

QUESTIONS

Icon made by Freepik from www.flaticon.com