π

A Pattern Language

WHAT IS PROGRAMMING?

WHAT IS PROGRAMMING?

The essence of programming is communicating behavior from a sending system to a receiving system.

WHAT IS PROGRAMMING?

The essence of programming is communicating behavior from a sending system to a receiving system.

WHAT IS PROGRAMMING?

The essence of programming is communicating behavior from a sending system to a receiving system.

… and vice versa.

COMMUNICATION

No matter of what kind these systems are…

COMMUNICATION

No matter of what kind these systems are…

COMMUNICATION

No matter of what kind these systems are…

COMMUNICATION

No matter of what kind these systems are…

SYMBOLS

Communication is done by sending symbols.

SYMBOLS

Communication is done by sending symbols.

PARAMETRIZED SYMBOLS

We parametrize symbols to create generic symbols.

PARAMETRIZED SYMBOLS

We parametrize symbols to create generic symbols.

```
print("Hello Onward!");
```

PARAMETRIZED SYMBOLS

We parametrize symbols to create generic symbols.

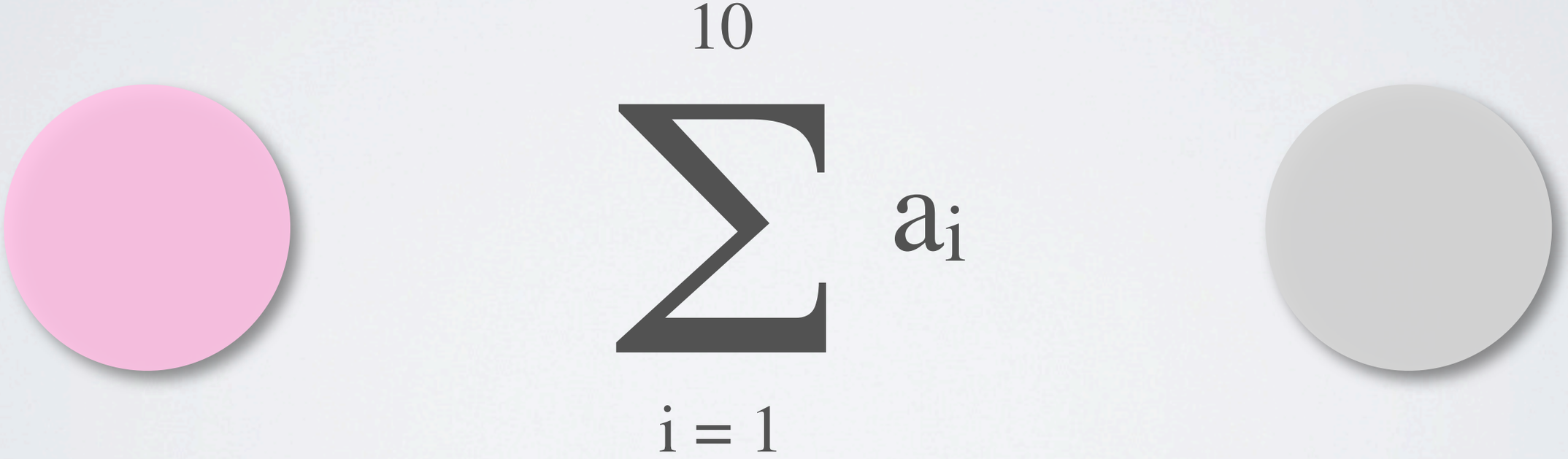
```
print(                    );
```

PARAMETRIZED SYMBOLS

We parametrize symbols to create generic symbols.

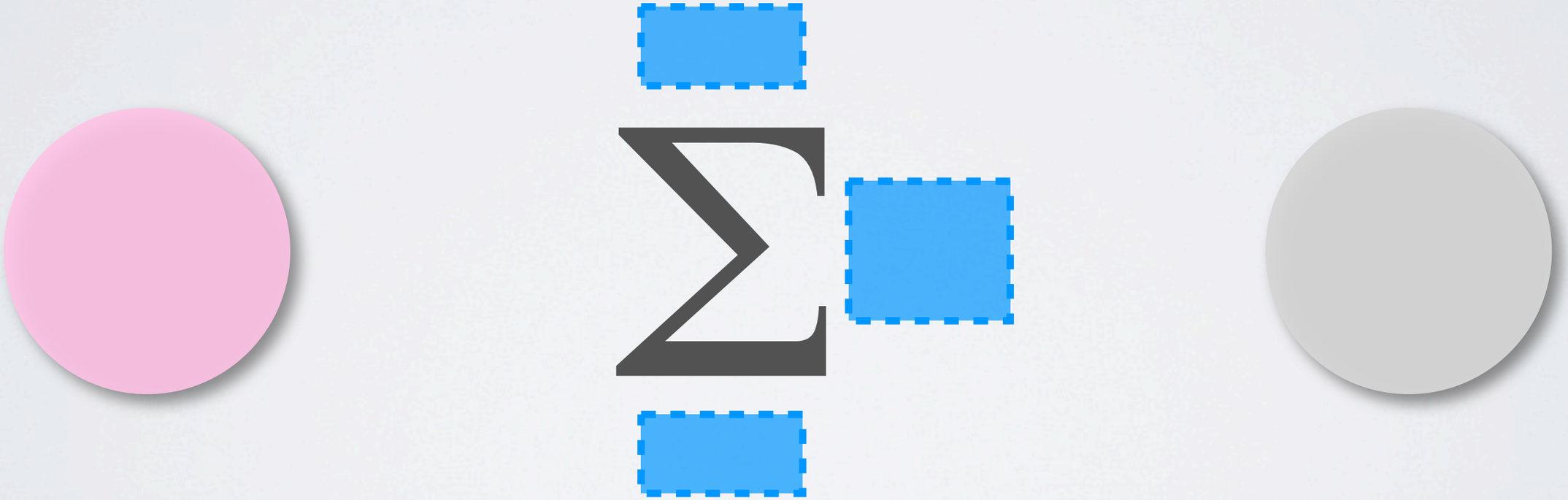
PARAMETRIZED SYMBOLS

We parametrize symbols to create generic symbols.



PARAMETRIZED SYMBOLS

We parametrize symbols to create generic symbols.



PARAMETRIZED SYMBOLS

We parametrize symbols to create generic symbols.

PARAMETRIZED SYMBOLS

We parametrize symbols to create generic symbols.

PATTERNS

Parametrized symbols having a meaning are patterns.

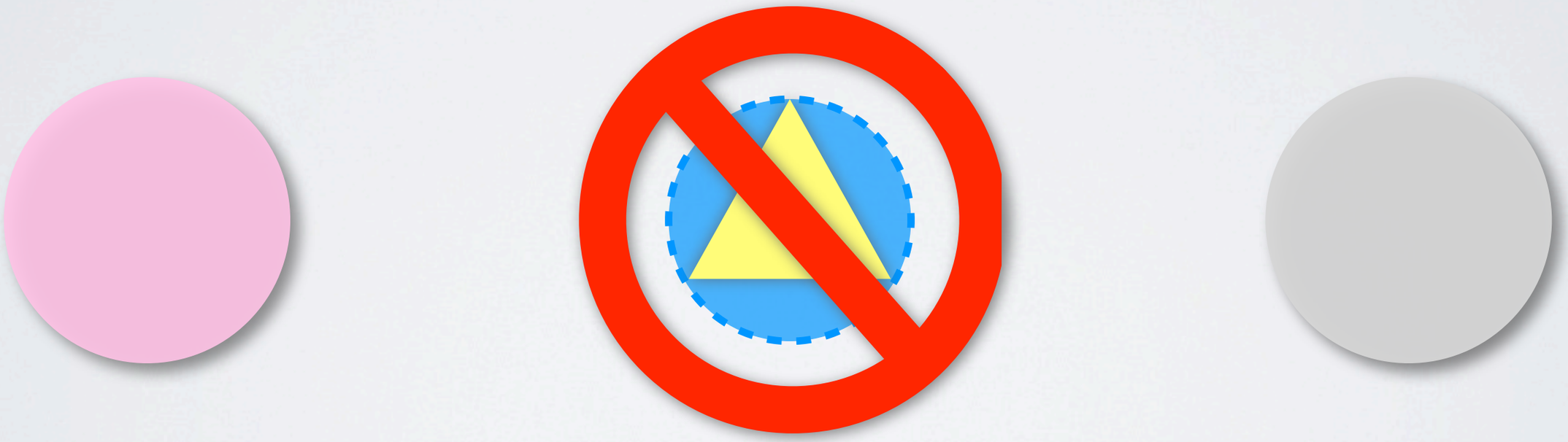
PATTERNS

Parametrized symbols having a meaning are patterns.

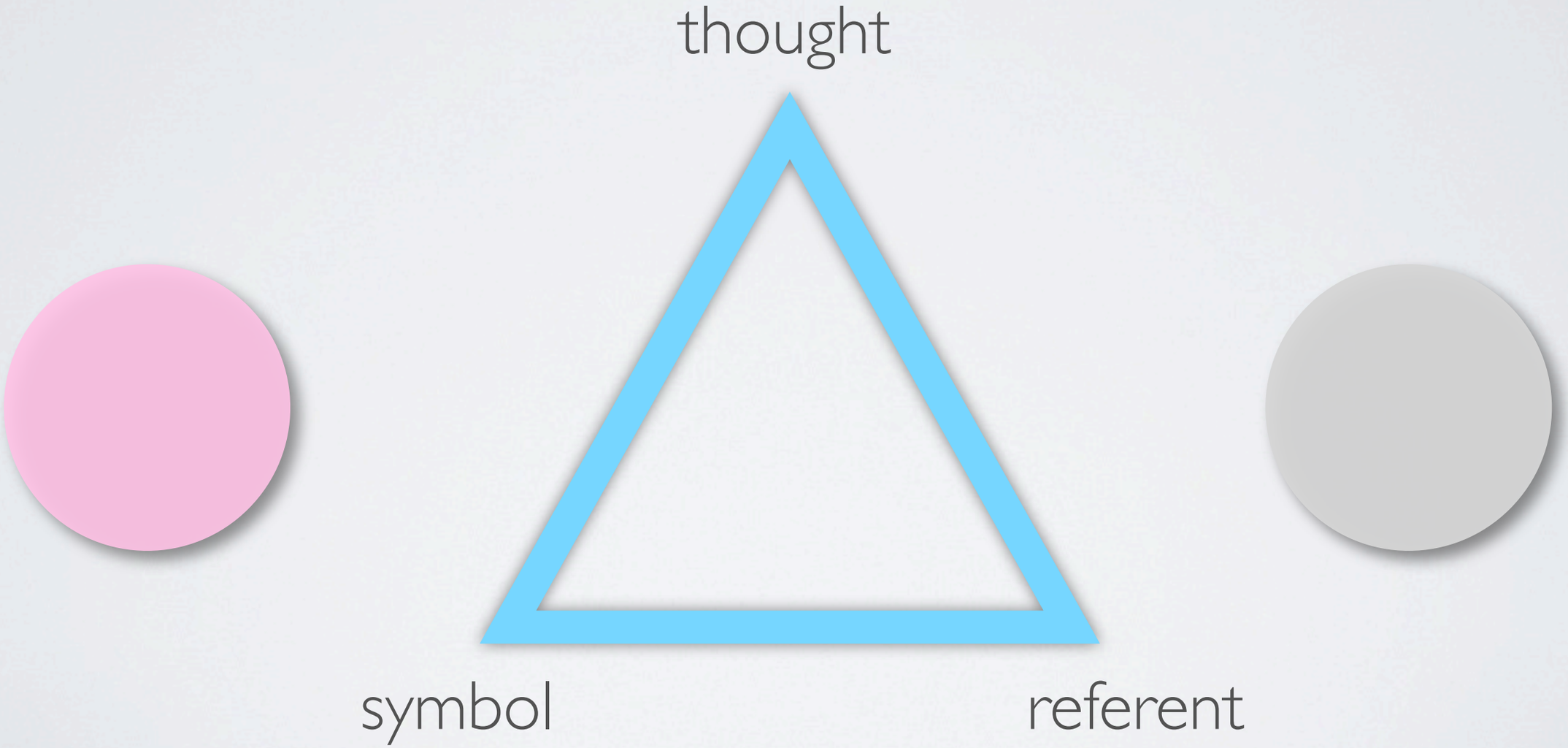
PATTERNS

Parametrized symbols having a meaning are patterns.

SEMIOTICS



SEMIOTICS



PROGRAMMING

A PATTERN LANGUAGE

Every programming language has and uses patterns.

A PATTERN LANGUAGE

Every programming language has and uses patterns.

But only a pattern language could create new patterns.

A PATTERN LANGUAGE

Every programming language has and uses patterns.

But only a pattern language could create new patterns.

There would be a single abstraction mechanism: patterns.

A PATTERN LANGUAGE

Every programming language has and uses patterns.

But only a pattern language could create new patterns.

There would be a single abstraction mechanism: patterns.

It would be homogeneously syntactically extensible.

A PATTERN LANGUAGE

Every programming language has and uses patterns.

But only a pattern language could create new patterns.

There would be a single abstraction mechanism: patterns.

It would be homogeneously syntactically extensible.

It would be reflection-complete and meta-complete.

π

$$\Pi$$

>

Π

```
> int a,b,c;
>
```

Π

```
> int a,b,c;
> a = 3; b = -2; c = 67;
>
```

Π

```
> int a,b,c;
> a = 3; b = -2; c = 67;
> print (a^3);
27
>
```

Π

```
> int a,b,c;
> a = 3; b = -2; c = 67;
> print (a^3);
27
> if ( (a ≠ 2) ∧ (|b^7| > 121) )
      print ("yes!");
yes!
>
```

Π

```
> int a,b,c;
> a = 3; b = -2; c = 67;
> print (a^3);
27
> if ( (a ≠ 2) ∧ (|b^7| > 121) )
      print ("yes!");
yes!
> unless (50 < c ≤ 70)
      print ("out of range!");
>
```

π

π

absolute_value := "|" *integer*:i "|" ⇒ integer ➞ i ≥ 0 ? i : -i

π

absolute_value = "|" *integer*:i "|" ⇒ integer → i ≥ 0 ? i : -i

π

absolute_value := "|" *integer*:i "|" ⇒ integer ➞ i ≥ 0 ? i : -i

π

absolute_value ::= "|" *integer*:i "|" ⇒ integer → i ≥ 0 ? i : -i

π

absolute_value := "|" *integer*:i "|" ⇒ integer ➞ i ≥ 0 ? i : -i

π

absolute_value := "|" *integer*:i "|" = integer → i ≥ 0 ? i : -i

π

absolute_value := "|" *integer*:i "|" ⇒ integer ➞ i ≥ 0 ? i : -i

π

absolute_value := "|" *integer*:i "|" ⇒ integer ⟼ i ≥ 0 ? i : -i

π

```
absolute_value := "|" integer:i "|" ⇒ integer → i ≥ 0 ? i : -i
```

```
potentiation :=
    integer "^" integer
    ⇒ integer
    → {
        int result = i;
        for (int k = 1; k ≤ j-1; k++)
            result *= i;
        return result;
    }
```

π

```
absolute_value := "|" integer:i "|" ⇒ integer → i ≥ 0 ? i : -i
```

```
potentiation :=
    integer "^" integer
    ⇒ integer
    → {
        int result = i;
        for (int k = 1; k ≤ j-1; k++)
            result *= i;
        return result;
    }
```

π

```
absolute_value := "|" integer:i "|" ⇒ integer → i ≥ 0 ? i : -i
```

```
potentiation :=
    integer "^" integer
    ⇒ integer
    → {
        int result = i;
        for (int k = 1; k ≤ j-1; k++)
            result *= i;
        return result;
    }
```

π

```
absolute_value := "|" integer:i "|" ⇒ integer → i ≥ 0 ? i : -i
```

```
potentiation :=
    integer "^" integer
    ⇒ integer
    → {
        int result = i;
        for (int k = 1; k ≤ j-1; k++)
            result *= i;
        return result;
    }
```

```
operator_chain :=
    integer:f [ ("<":s | "≤") integer:i ]:ff
    ⇒ boolean → {
        … if (f < ff[0].i) … }
```

π

```
absolute_value := "|" integer:i "|" ⇒ integer → i ≥ 0 ? i : -i
```

```
potentiation :=
    integer "^" integer
    ⇒ integer
    → {
        int result = i;
        for (int k = 1; k ≤ j-1; k++)
            result *= i;
        return result;
    }
```

```
operator_chain :=
    integer:f [ ("<":s | "≤") integer:i ]:ff
    ⇒ boolean → {
        … if (f < ff[0].i) … }
```

π

```
absolute_value := "|" integer:i "|" ⇒ integer → i ≥ 0 ? i : -i
```

```
potentiation :=
    integer "^" integer
    ⇒ integer
    → {
        int result = i;
        for (int k = 1; k ≤ j-1; k++)
            result *= i;
        return result;
    }
```

```
operator_chain :=
    integer:f [ ("<":s | "≤") integer:i ]:ff
    ⇒ boolean → {
        … if (f <s ff[0].i) … }
```

π

```
absolute_value := "|" integer:i "|" ⇒ integer → i ≥ 0 ? i : -i
```

```
potentiation :=
    integer "^" integer
    ⇒ integer
    → {
        int result = i;
        for (int k = 1; k ≤ j-1; k++)
            result *= i;
        return result;
    }
```

```
operator_chain :=
    integer:f [ ("<":s | "≤") integer:i ]:ff
    ⇒ boolean → {
        … if (f < ff[0].i) … }
```

π

```
absolute_value := "|" integer:i "|" ⇒ integer → i ≥ 0 ? i : -i
```

```
potentiation :=
    integer "^" integer
    ⇒ integer
    → {
        int result = i;
        for (int k = 1; k ≤ j-1; k++)
            result *= i;
        return result;
    }
```

```
operator_chain :=
    integer:f [ ("<":s | "≤") integer:i ]:ff
    ⇒ boolean → {
        … if (f < ff[0].i) … }
```

```
"unless" "(" expression ")" instruction
    ⇒ instruction → if (¬expression)
        evaluate (instruction);
```

π

```
absolute_value := "|" integer:i "|" ⇒ integer → i ≥ 0 ? i : -i
```

```
potentiation :=
    integer "^" integer
    ⇒ integer
    → {
        int result = i;
        for (int k = 1; k ≤ j-1; k++)
            result *= i;
        return result;
    }
```

```
operator_chain :=
    integer:f [ ("<":s | "≤") integer:i ]:ff
    ⇒ boolean → {
        … if (f < ff[0].i) … }
```

```
"unless" "(" expression ")" instruction
    ⇒ instruction → if (¬expression)
        evaluate (instruction);
```

π

```
absolute_value := "|" integer:i "|" ⇒ integer → i ≥ 0 ? i : -i
```

```
potentiation :=
    integer "^" integer
    ⇒ integer
    → {
        int result = i;
        for (int k = 1; k ≤ j-1; k++)
            result *= i;
        return result;
    }
```

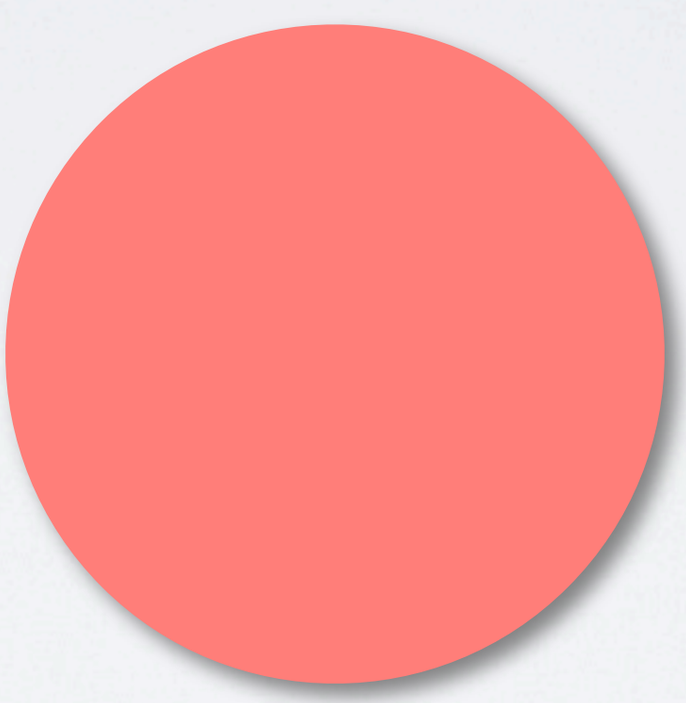
```
operator_chain :=
    integer:f [ ("<":s | "≤") integer:i ]:ff
    ⇒ boolean → {
        … if (f < ff[0].i) … }
```

```
"unless" "(" expression ")" instruction
    ⇒ instruction → if (¬expression)
        evaluate (instruction);
```

π

π

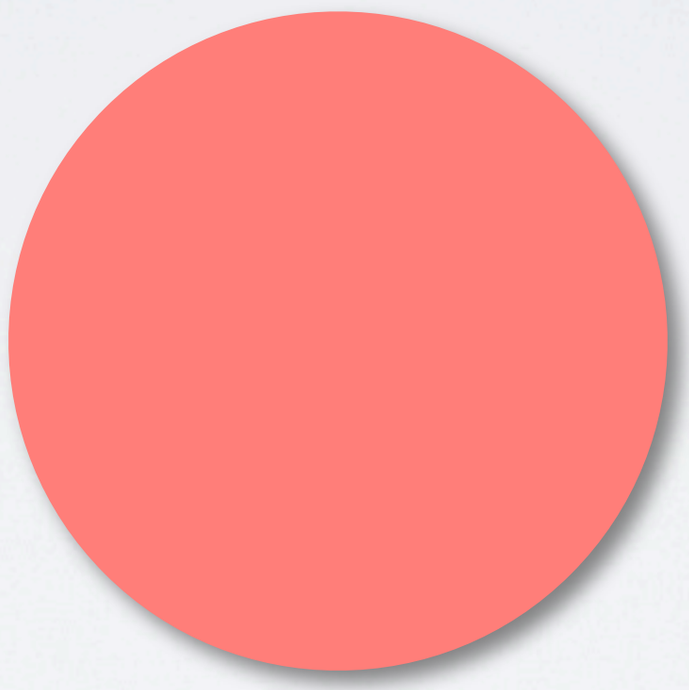
π-interpreter



π

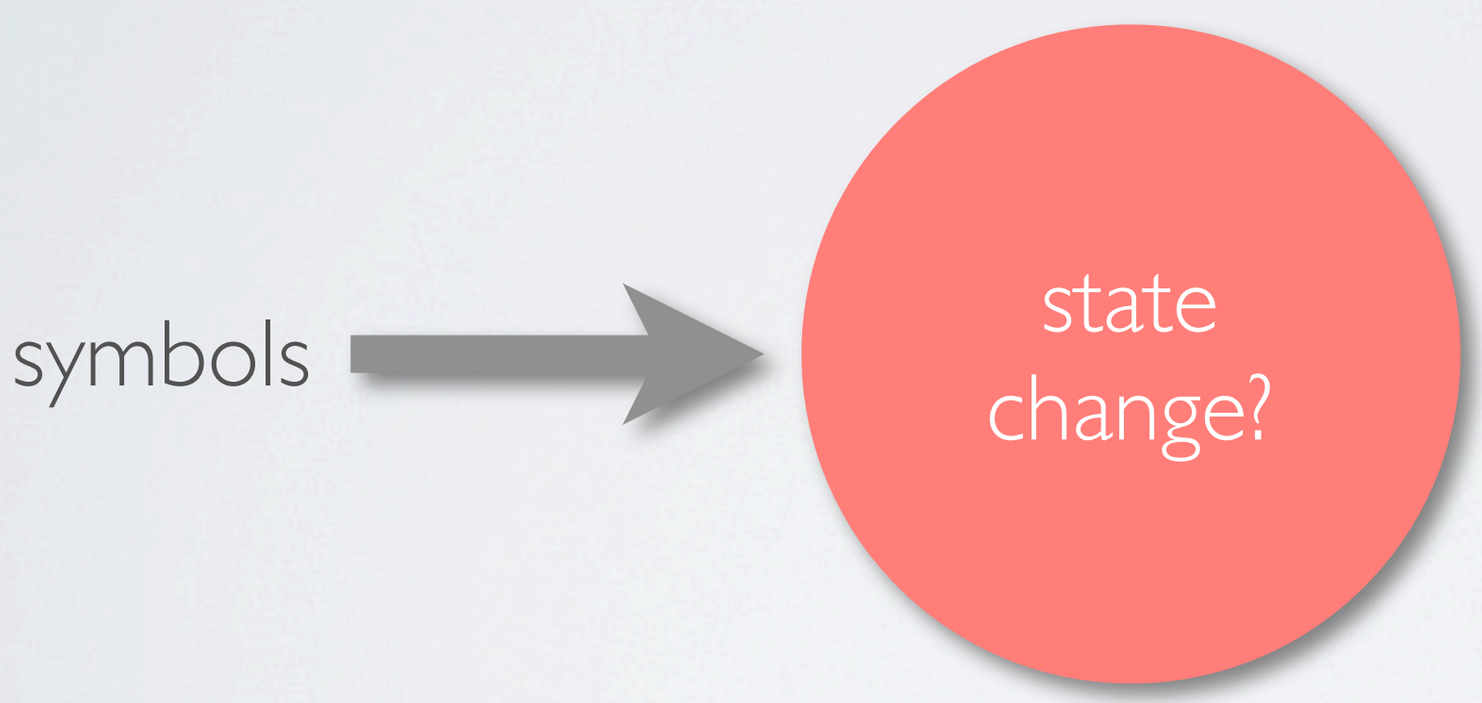
π-interpreter

symbols



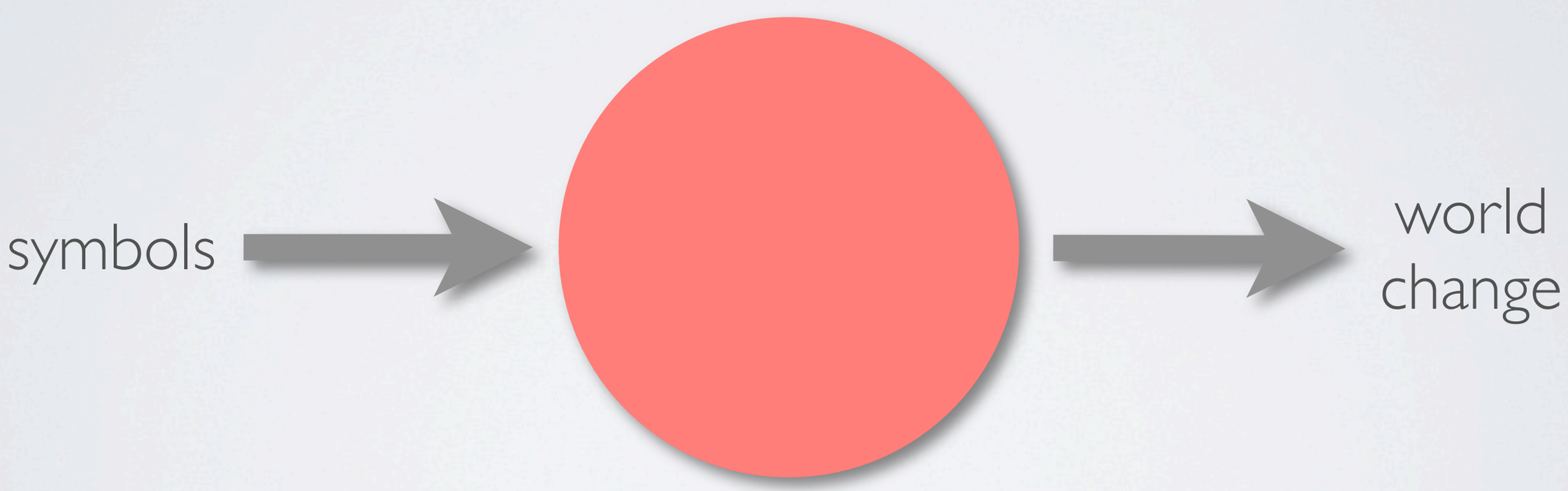
π

π-interpreter



π

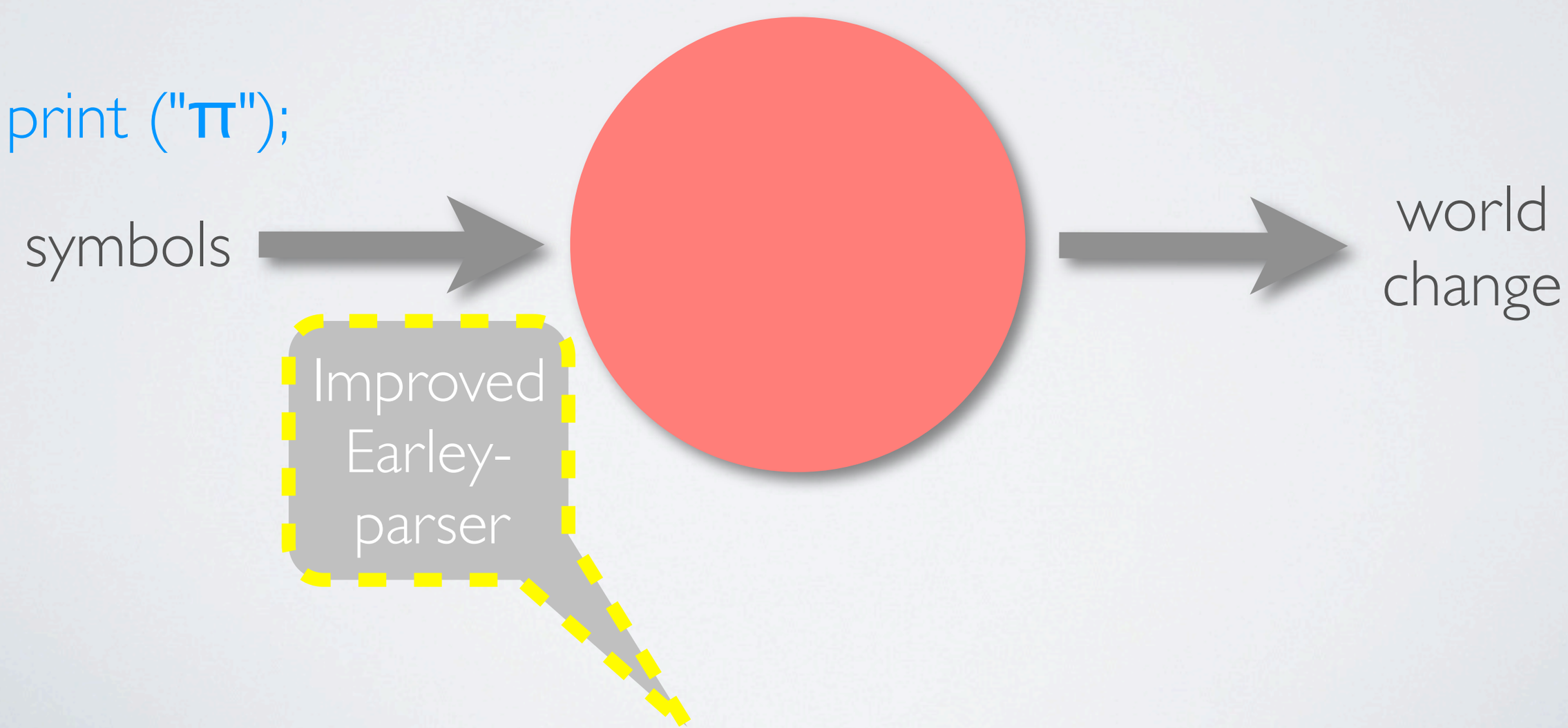
π-interpreter



π

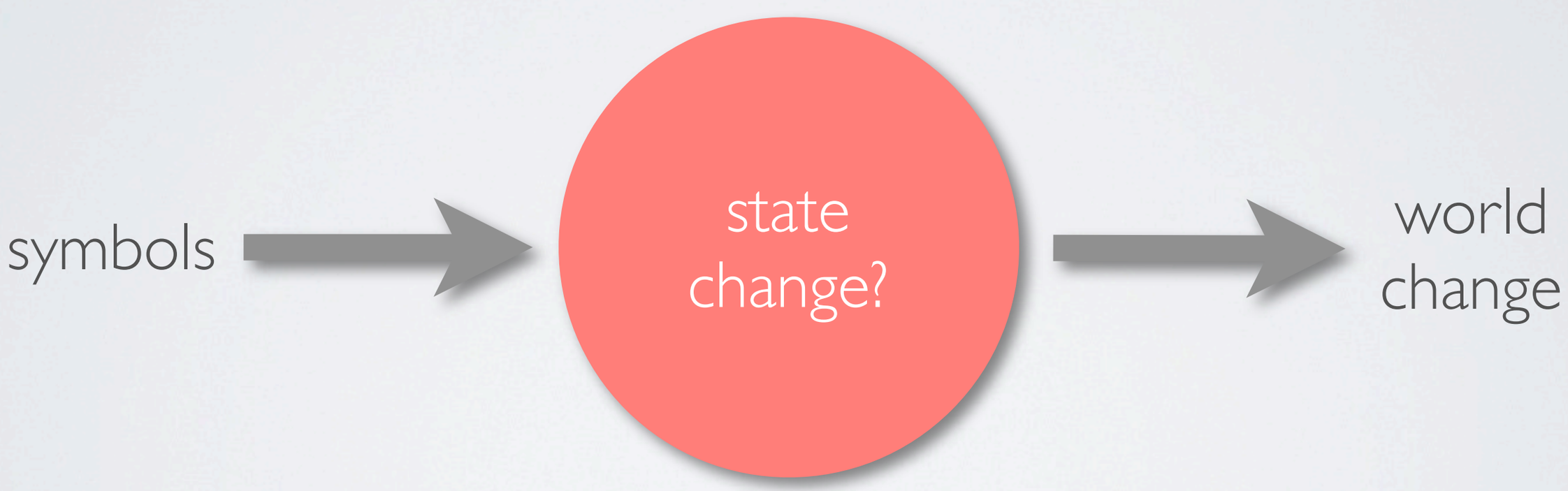
π-interpreter

print ("π");



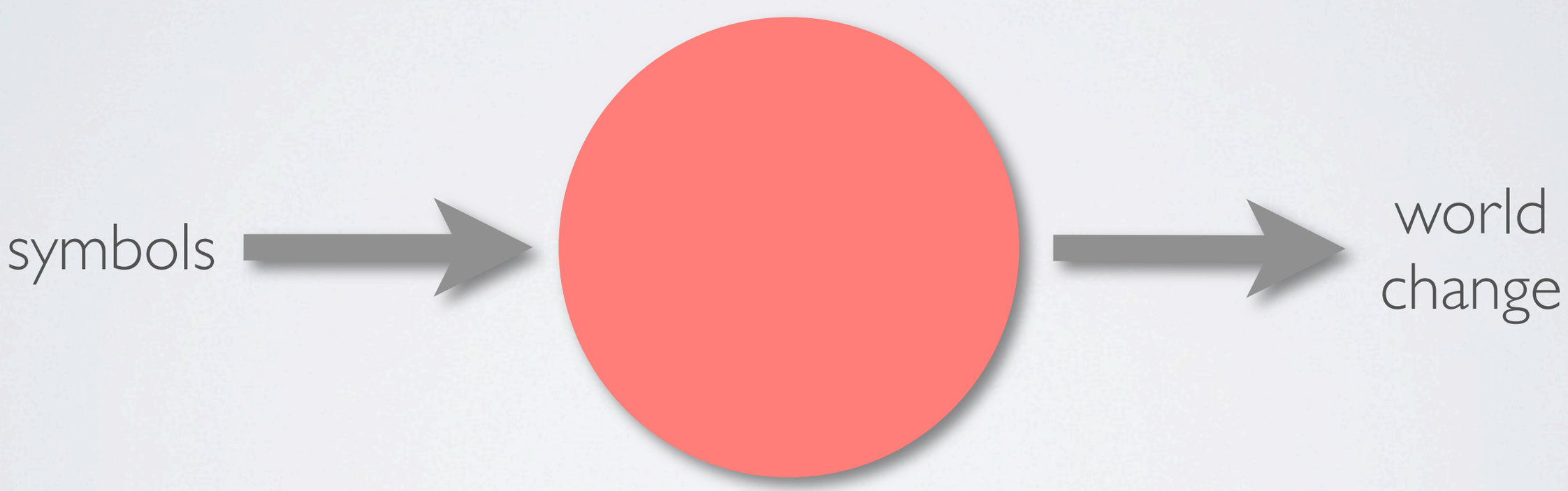
π

π-interpreter



π

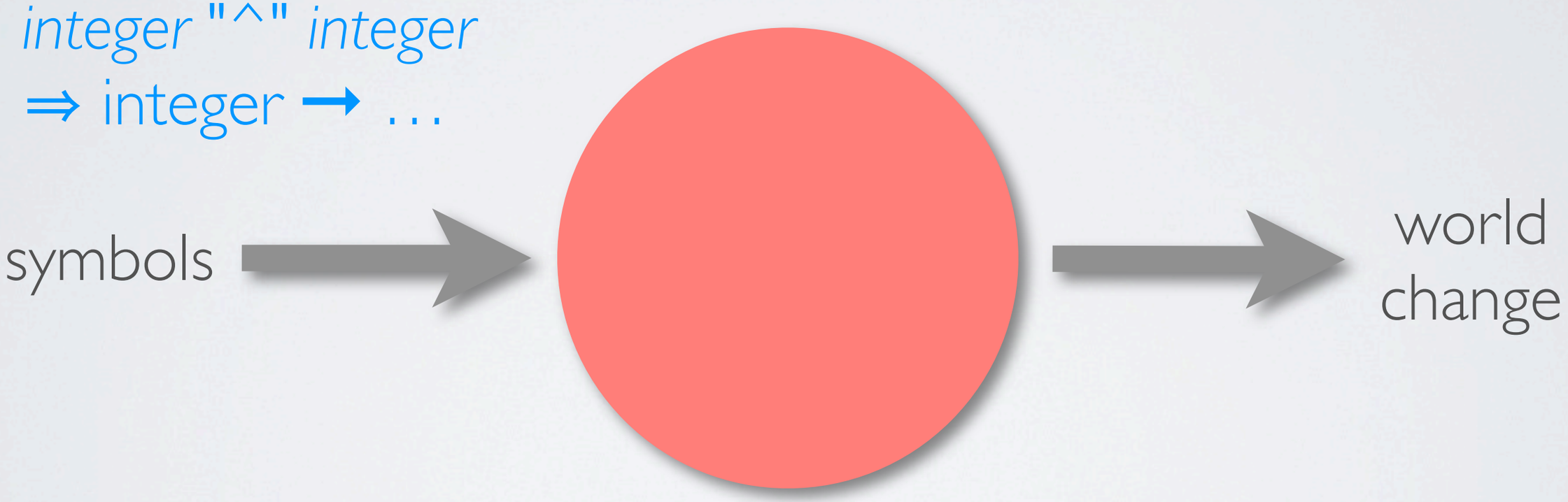
π-interpreter



π

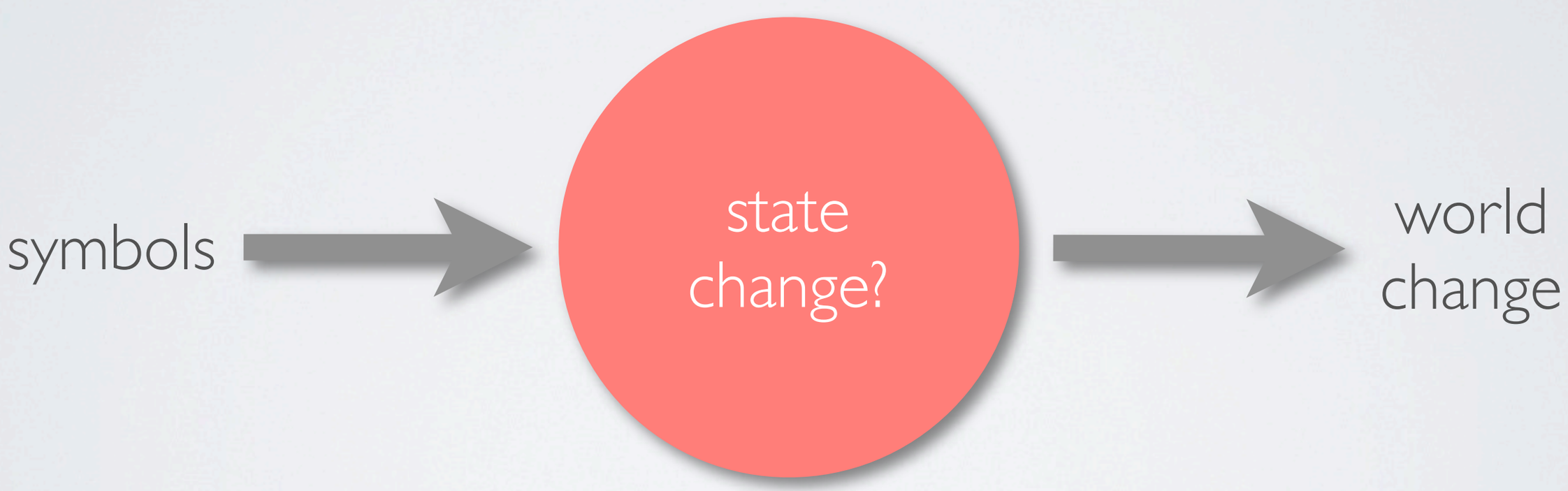
potentiation :=
integer "^" *integer*
⇒ integer ⟶ …

π-interpreter



π

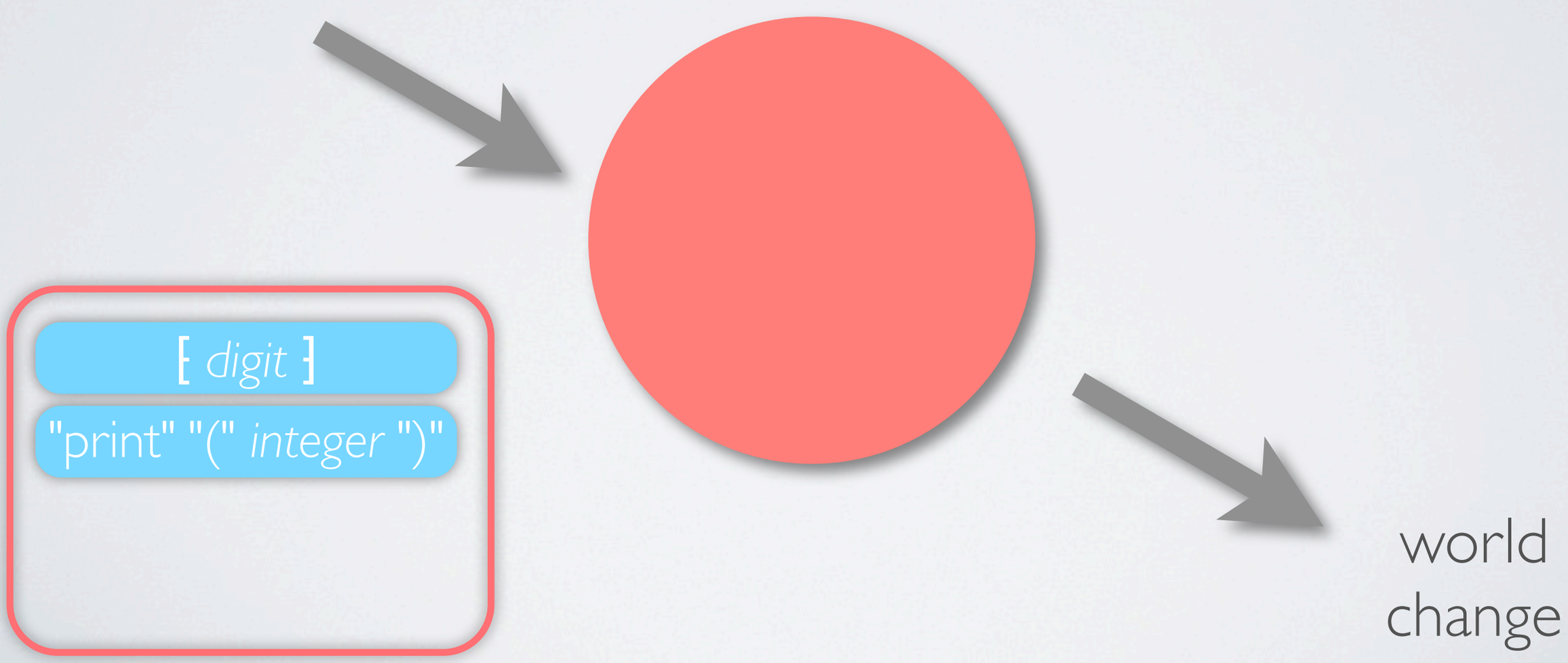
π-interpreter



π

symbols

π-interpreter

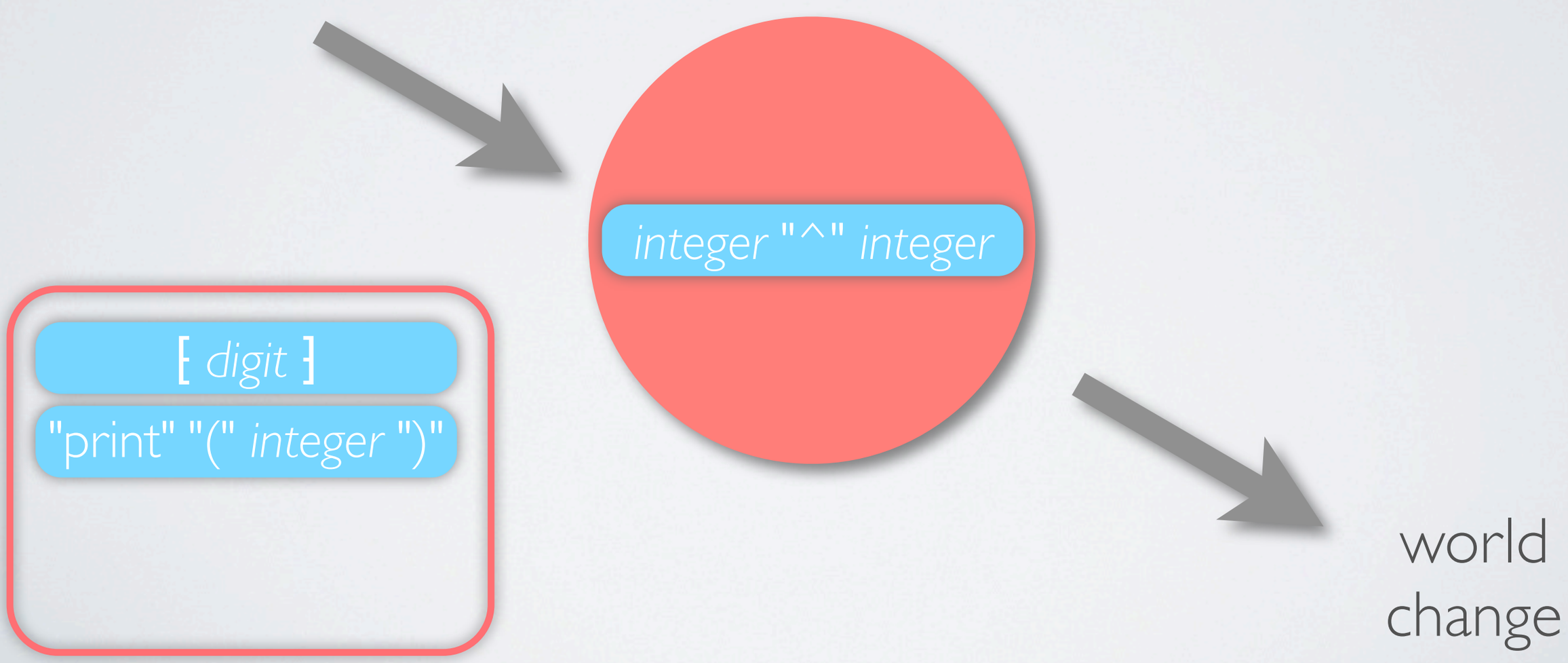


world
change

π

symbols

π-interpreter

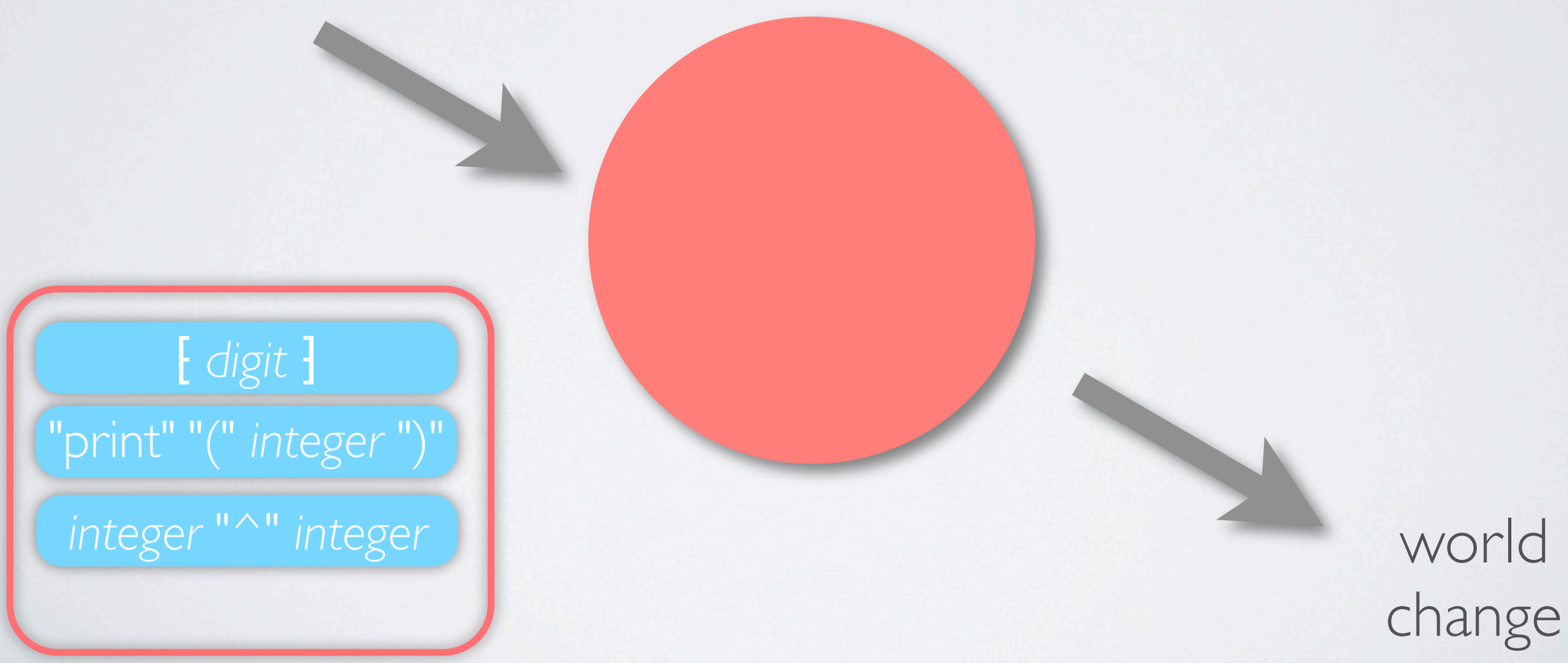


world
change

π

π-interpreter

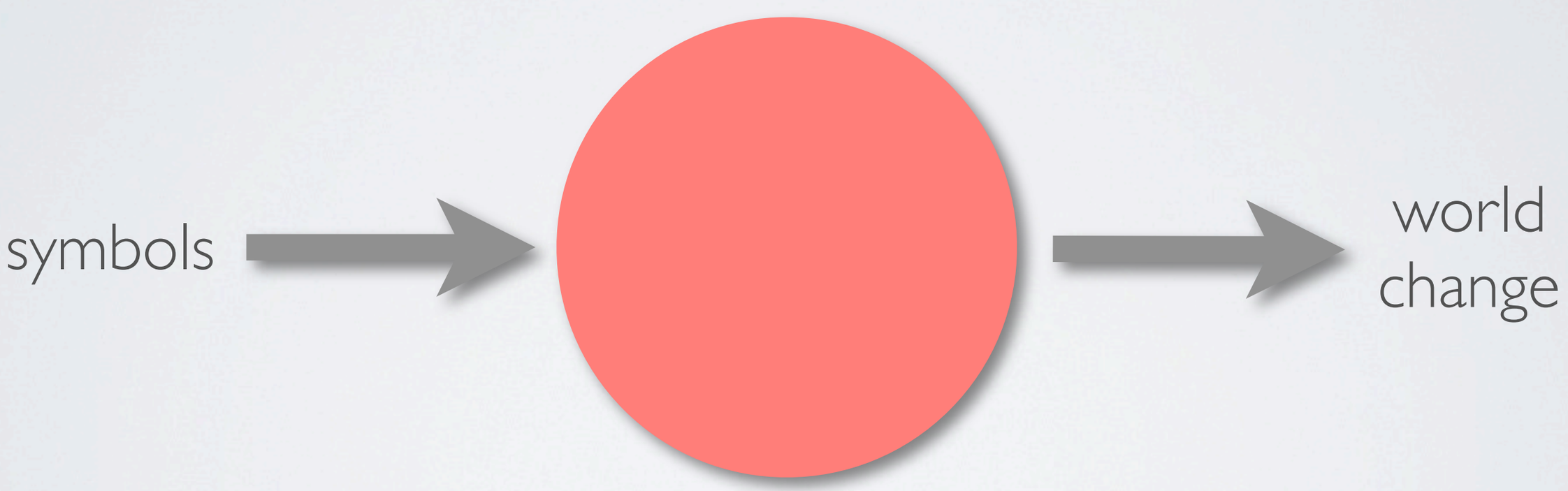
symbols



world
change

π

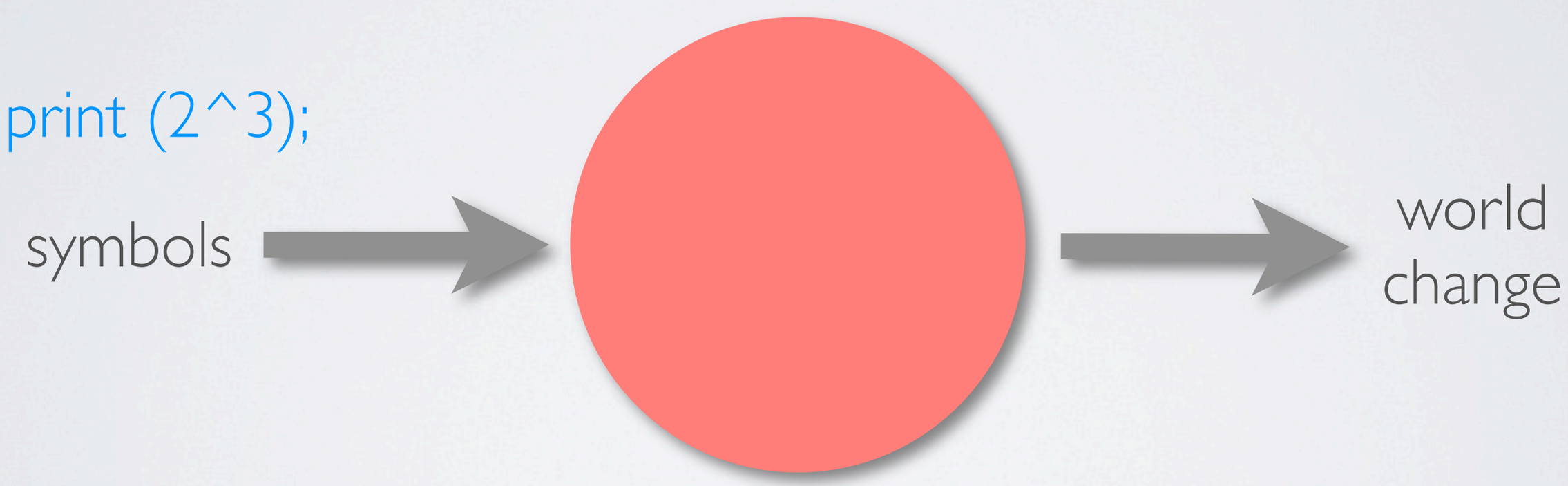
π-interpreter



π

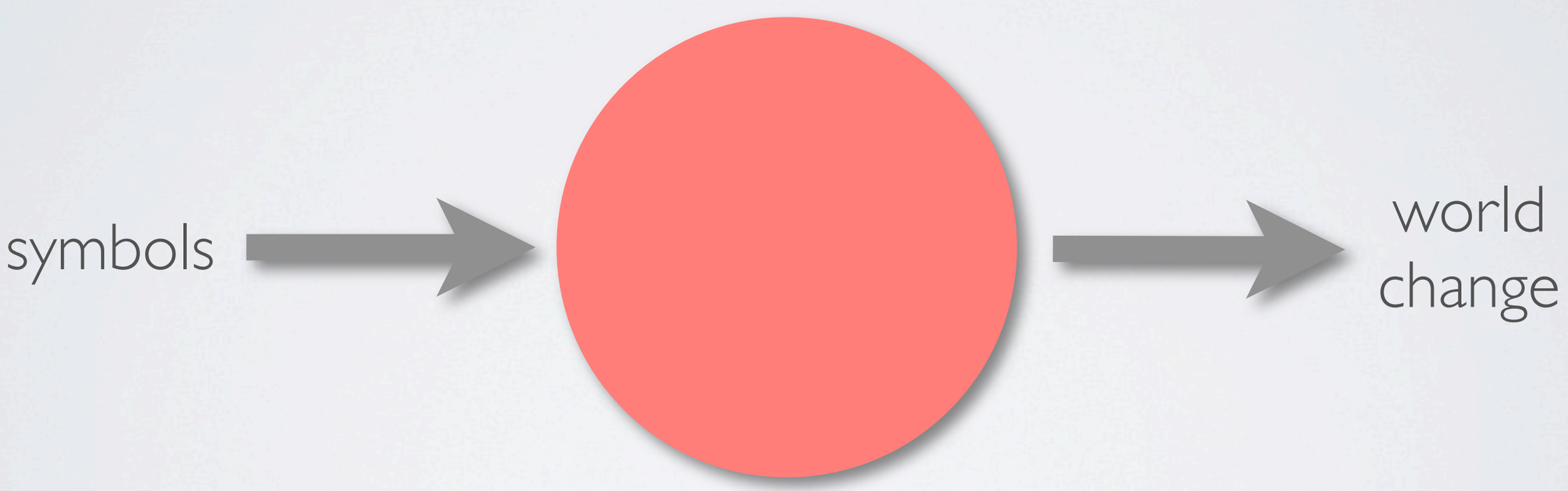
π-interpreter

print (2^3);



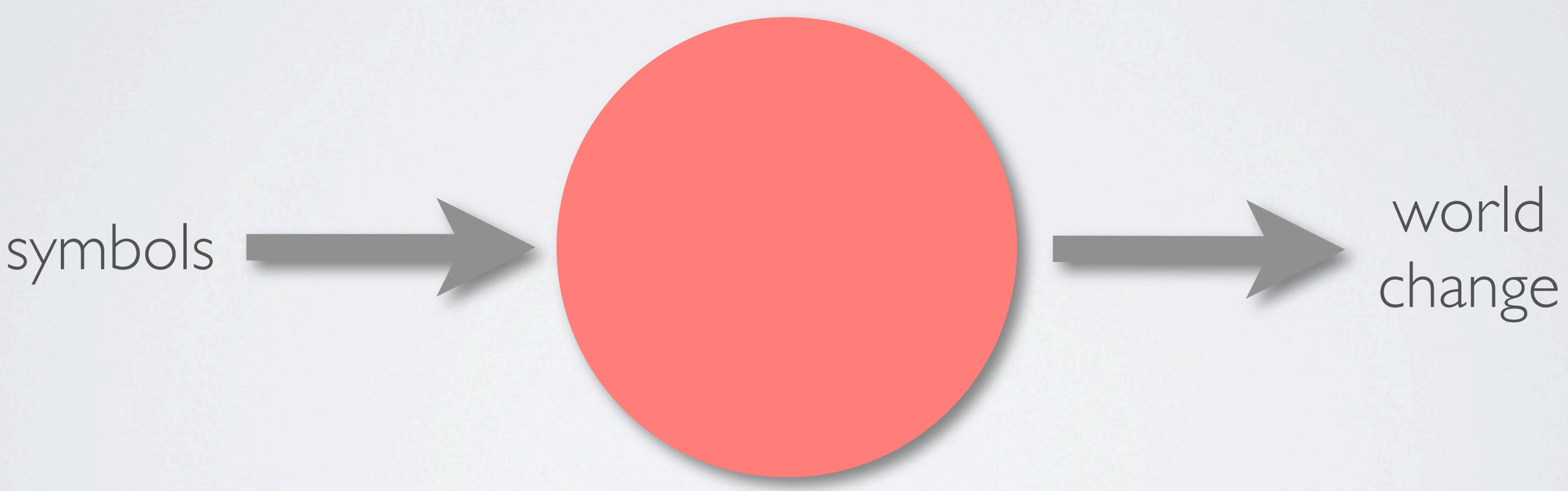
π

π-interpreter



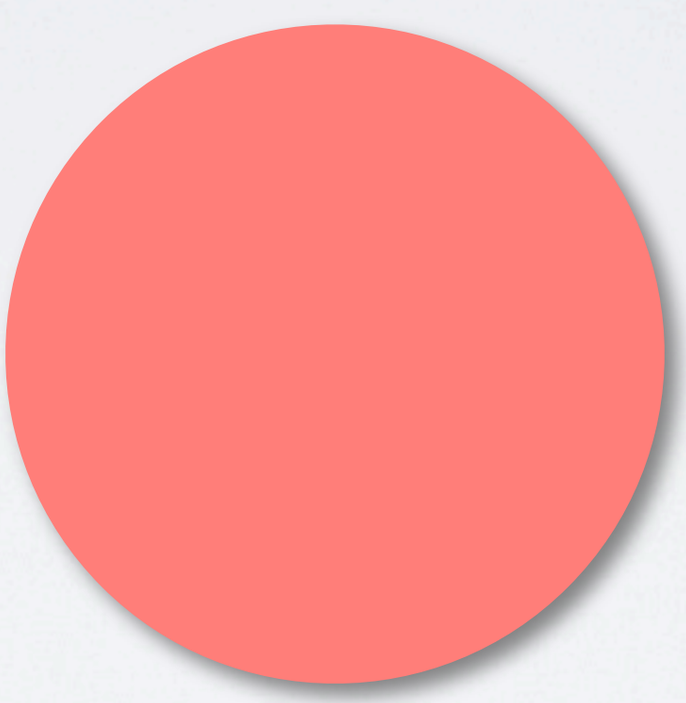
π

π-interpreter



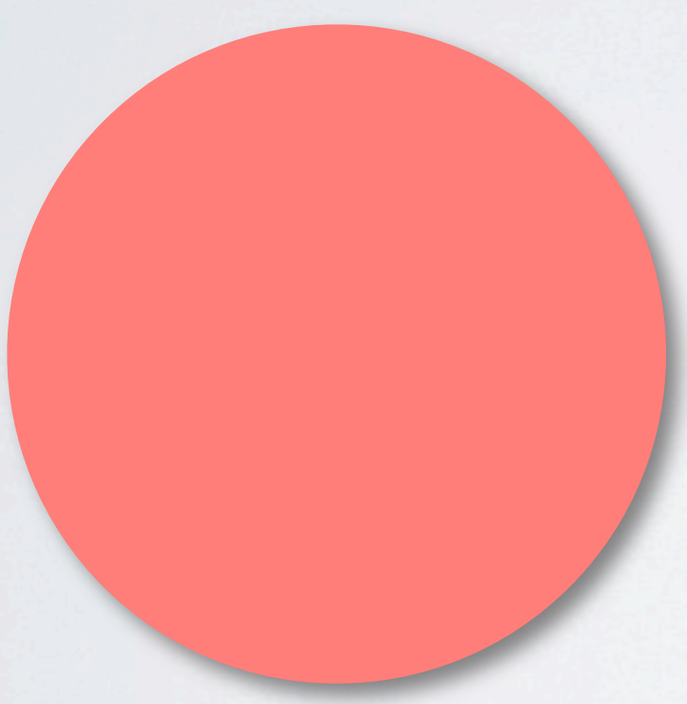
EVALUATION

π-interpreter



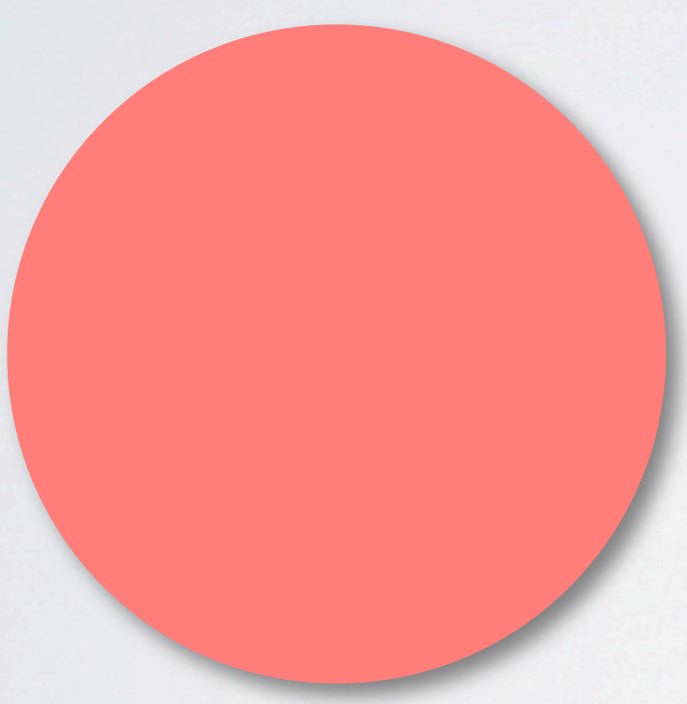
EVALUATION

π-interpreter



EVALUATION

π-interpreter

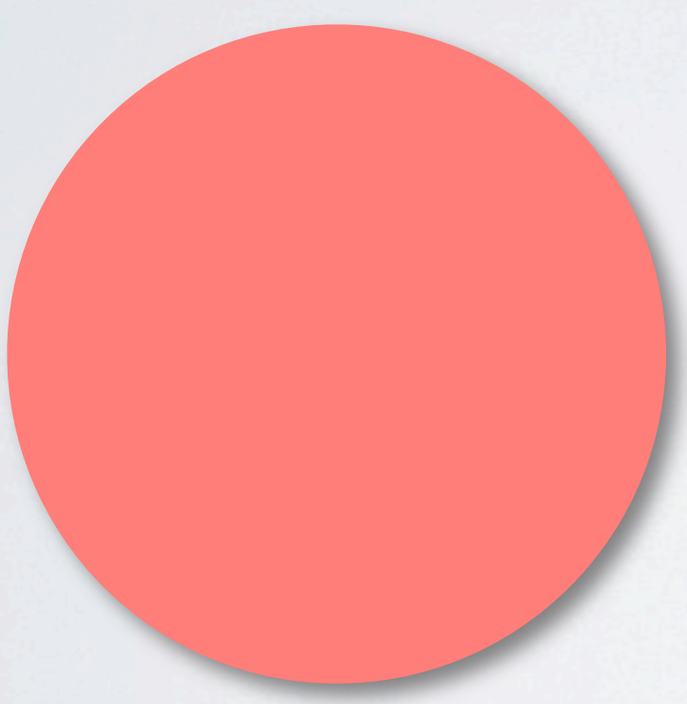


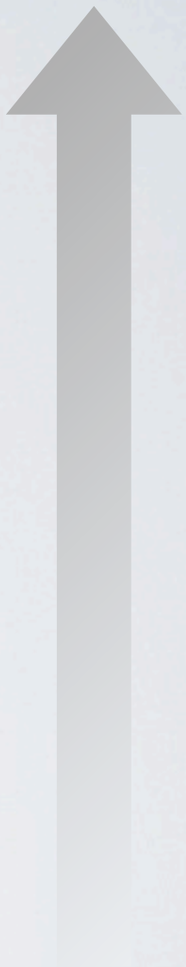
Level 1
Language Constructs

EVALUATION

π-interpreter

if (a|9) print (max (|a|, |b|));

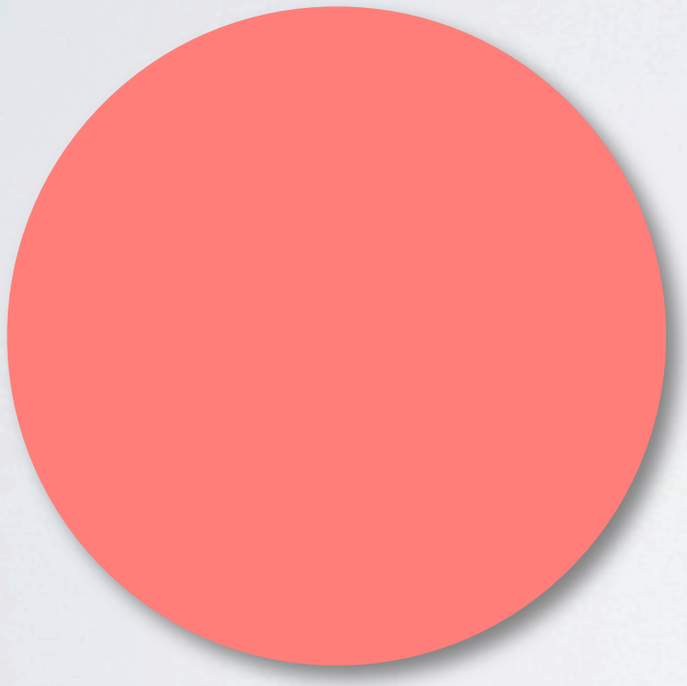




Level 1
Language Constructs

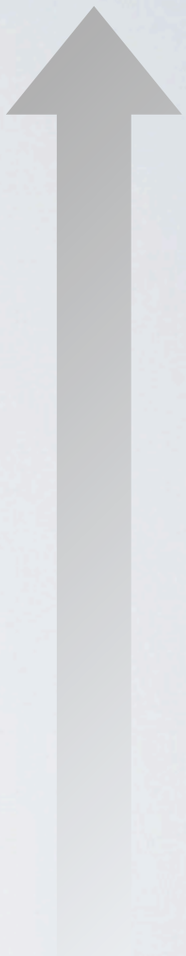
EVALUATION

π-interpreter



if (a|9) print (max (|a|, |b|));

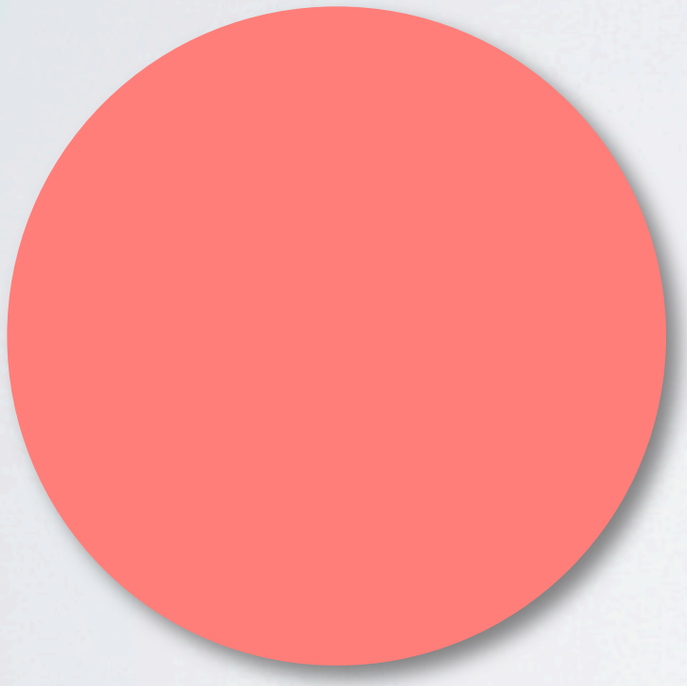
```
abs_value ::= | integer:i |
  ⇒ integer → i ≥ 0 ? i : -i
```



Level I
Language Constructs

EVALUATION

π-interpreter



if (a|9) print (max (|a|, |b|));

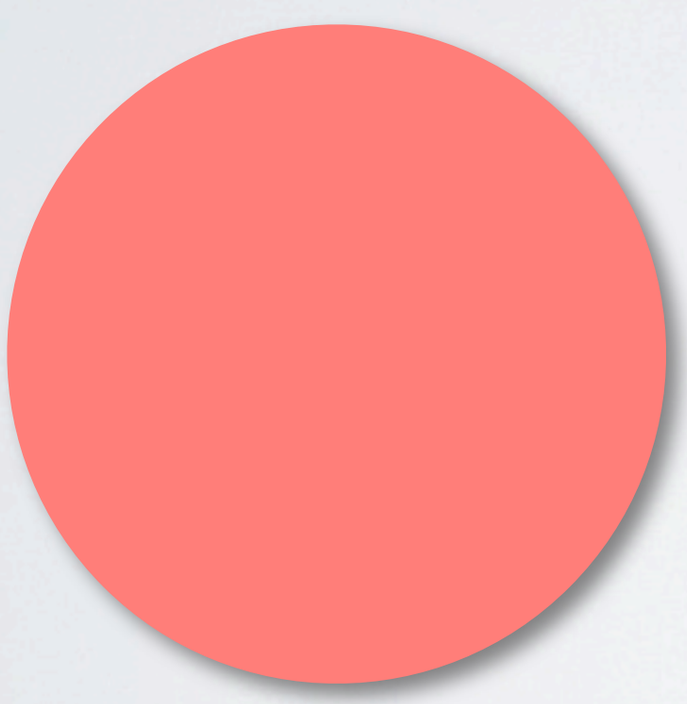
abs_value := "|" *integer*:i "|"
⇒ integer → i ≥ 0 ? i : -i

divisible := integer:i "|" integer:j
⇒ boolean → i % j = 0

Level I
Language Constructs

EVALUATION

π-interpreter

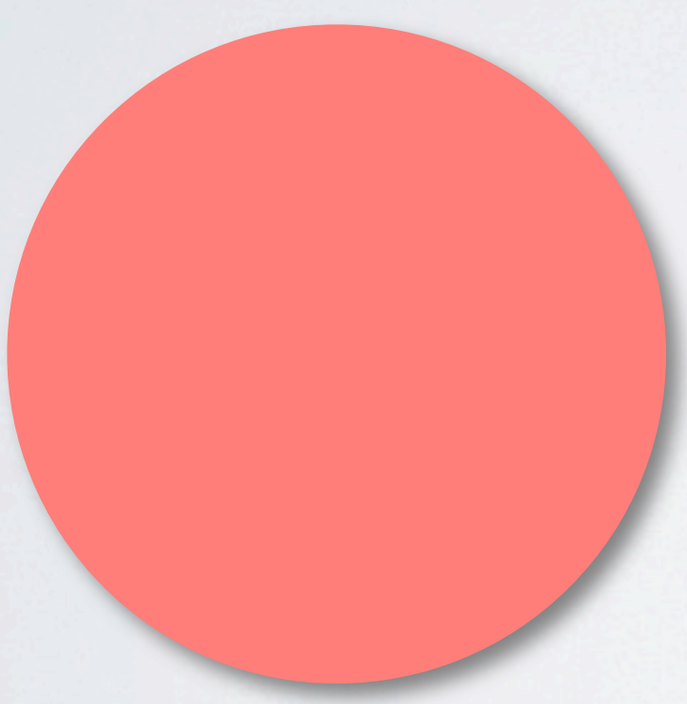


Level 1
Language Constructs

EVALUATION

π-interpreter

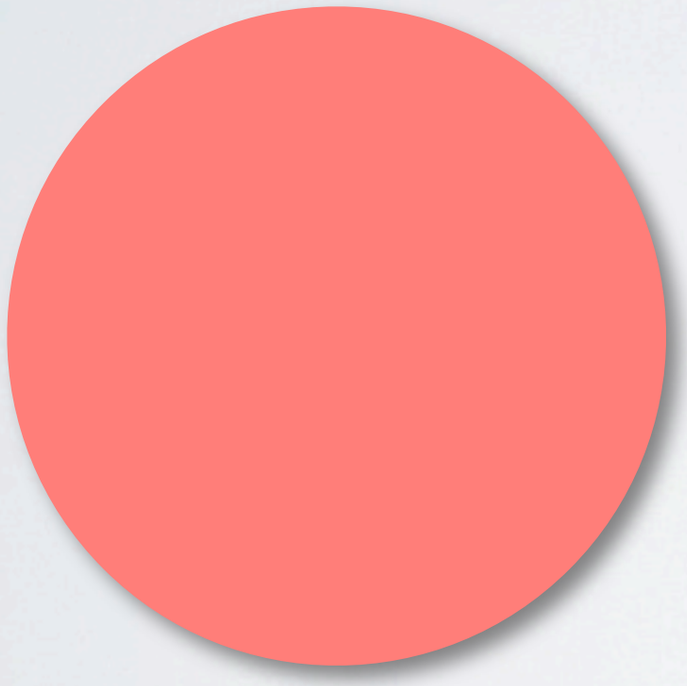
do { print ("hello!"); } (10) times;



Level 1
Language Constructs

EVALUATION

π-interpreter



do { print ("hello!"); } (10) times;

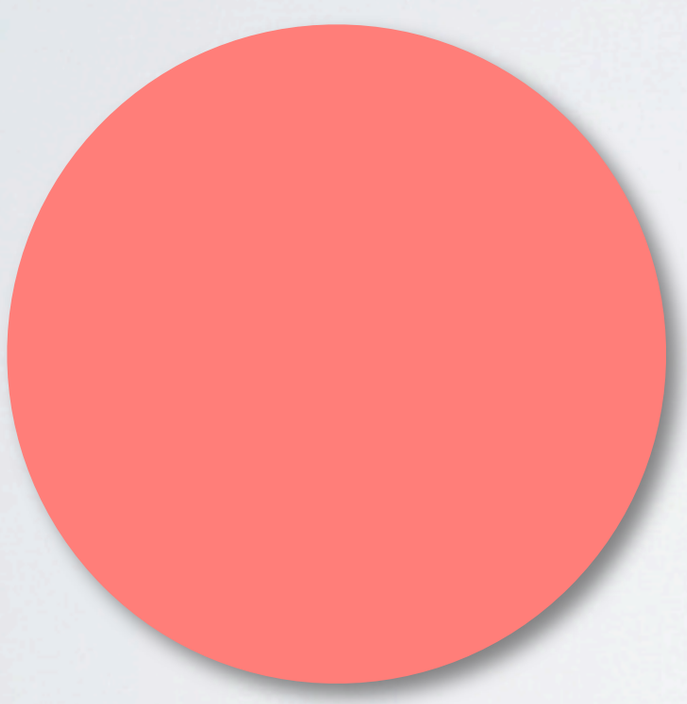
```
do_times_loop :=
   "do" block
   "(" integer:times ")" "times" ";"
   ⇒ loop → {

   for (int i = 1; i ≤ times; i++)
      execute (block);
}
```

Level 1
Language Constructs

EVALUATION

π-interpreter

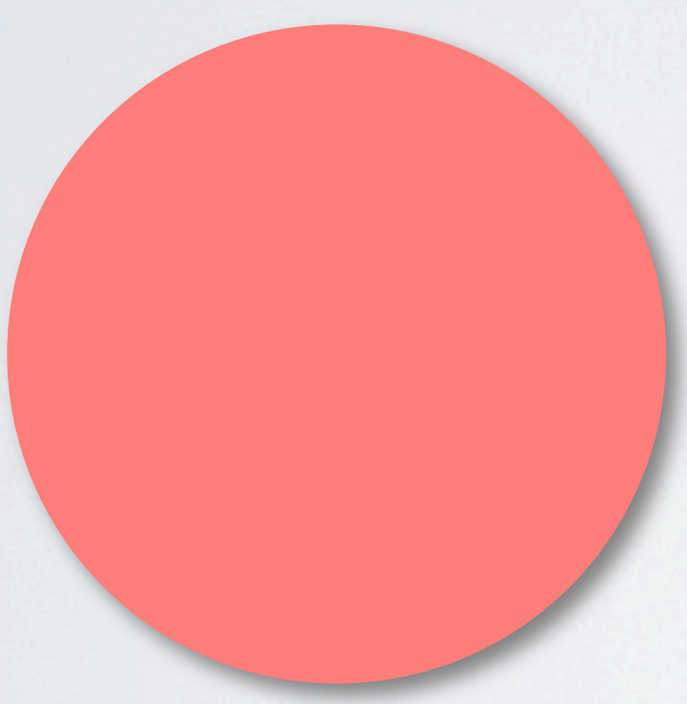


Level 1
Language Constructs

EVALUATION

π-interpreter

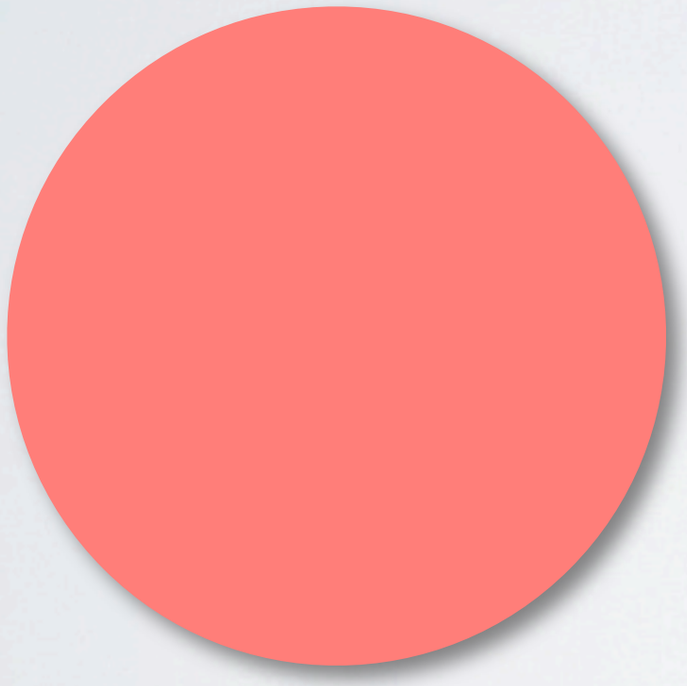
(2 7 3
5 1 9)



Level 1
Language Constructs

EVALUATION

π-interpreter



(2 7 3
5 1 9)

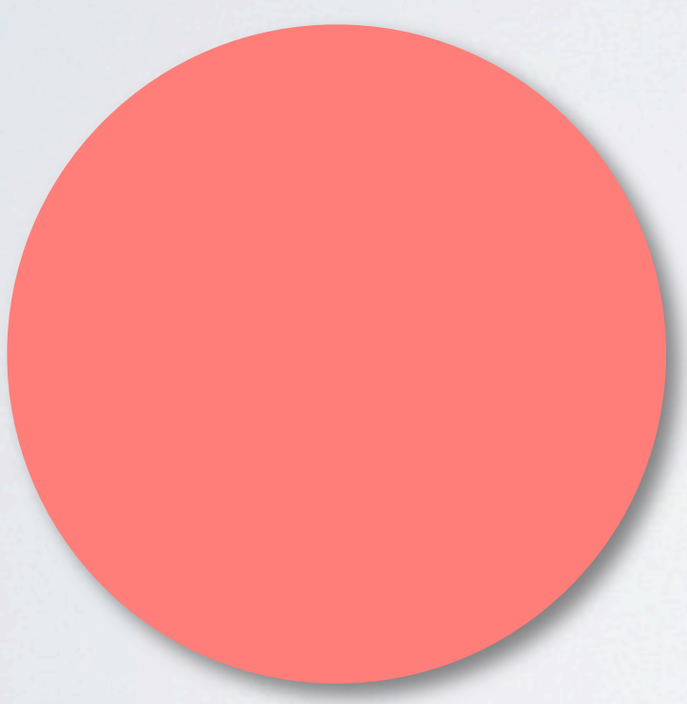
```
matrix :=
   "(" row { %LB row } ")"
   ⇒ data

row :=
   integer { %S integer }
   ⇒ data
```

Level 1
Language Constructs

EVALUATION

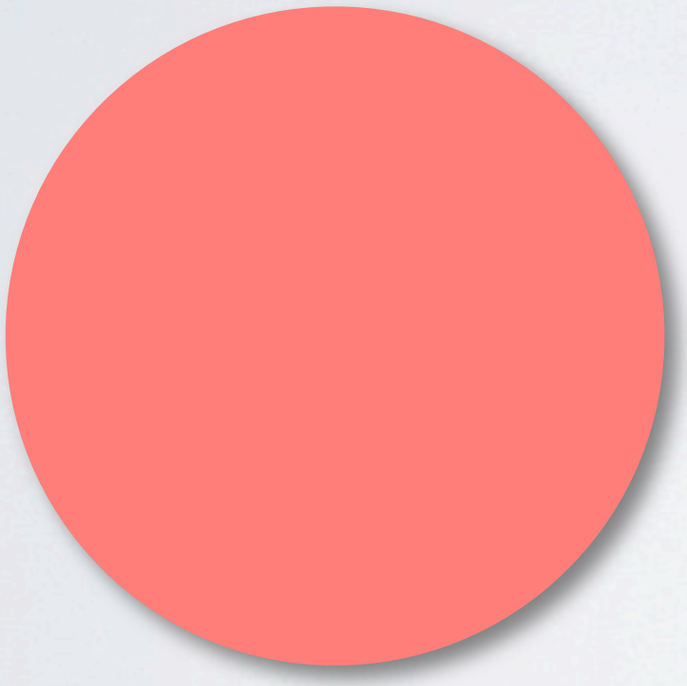
π-interpreter

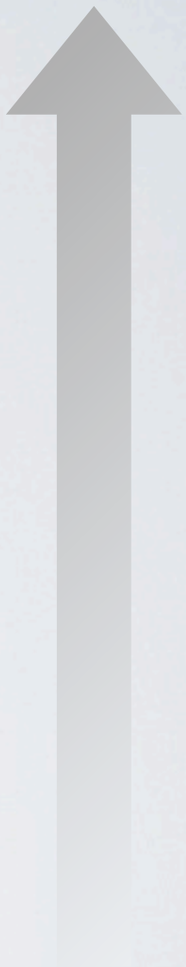


Level 1
Language Constructs

EVALUATION

π-interpreter



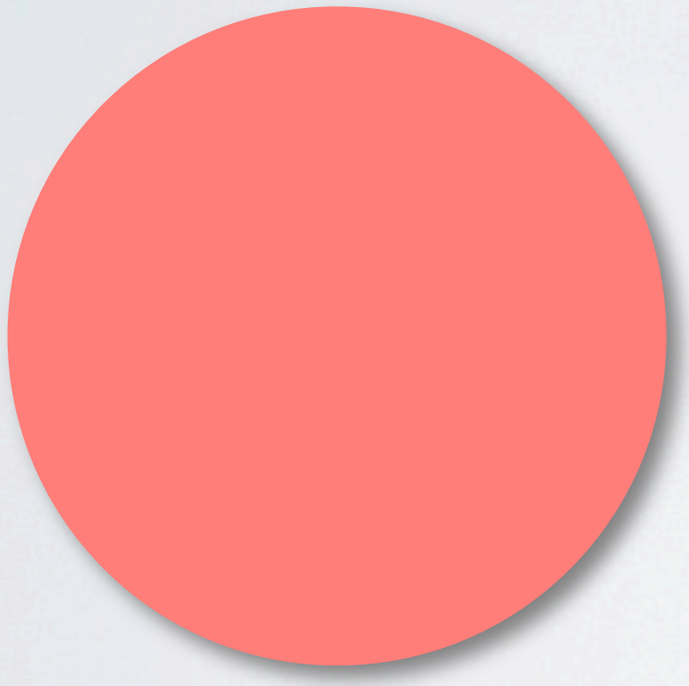


Level II
Meta Constructs

EVALUATION

π-interpreter

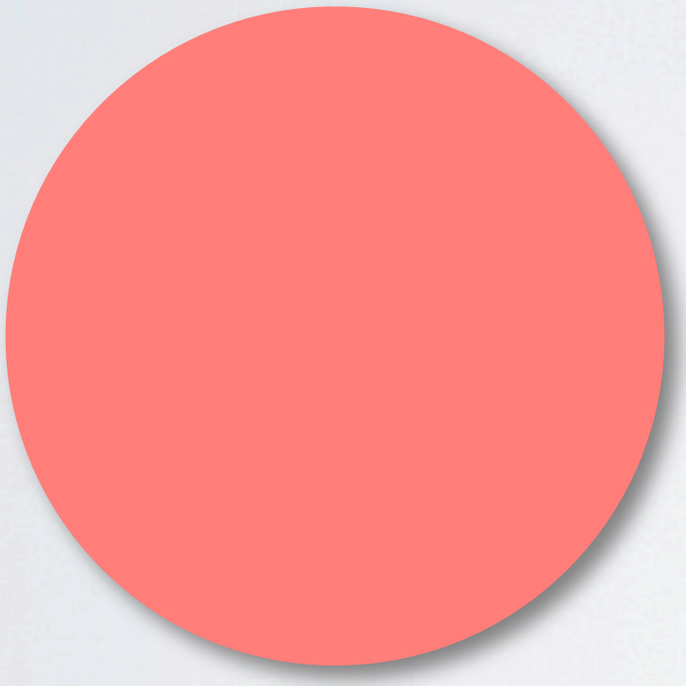
function fib (n:$\mathbb{N}$) $\mapsto$ $\mathbb{N}$
= { n = 0 ∨ n = 1 : 1
n ≥ 2 : fib(n-1) + fib(n-2);



Level II
Meta Constructs

EVALUATION

π-interpreter



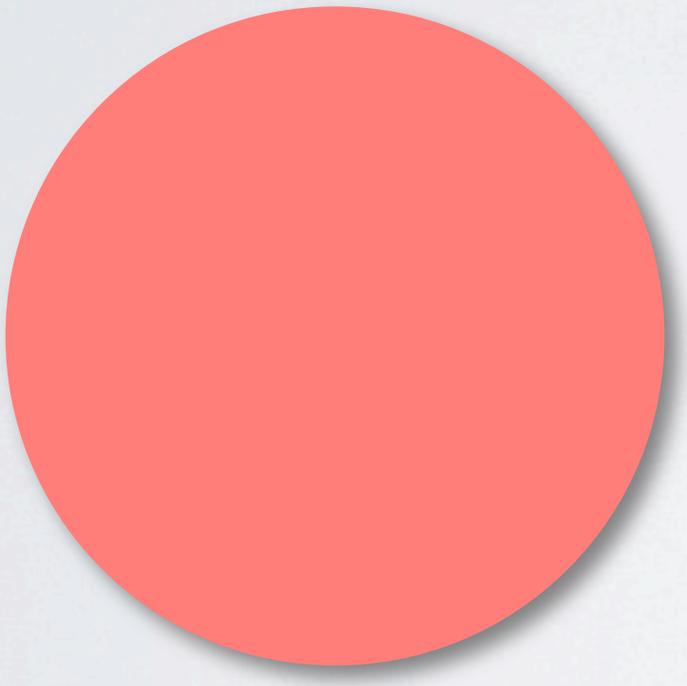
function fib (n:$\mathbb{N}$) $\mapsto \mathbb{N}$
= { n = 0 ∨ n = 1 : 1
n ≥ 2 : fib(n-1) + fib(n-2);

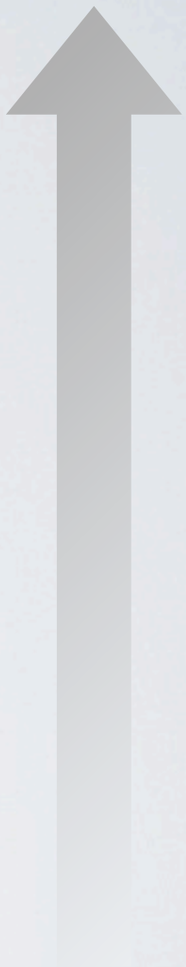
```
function := "function" name …
   ⇒ declaration → {
   declare_pattern name …
}
```

Level II
Meta Constructs

EVALUATION

π-interpreter

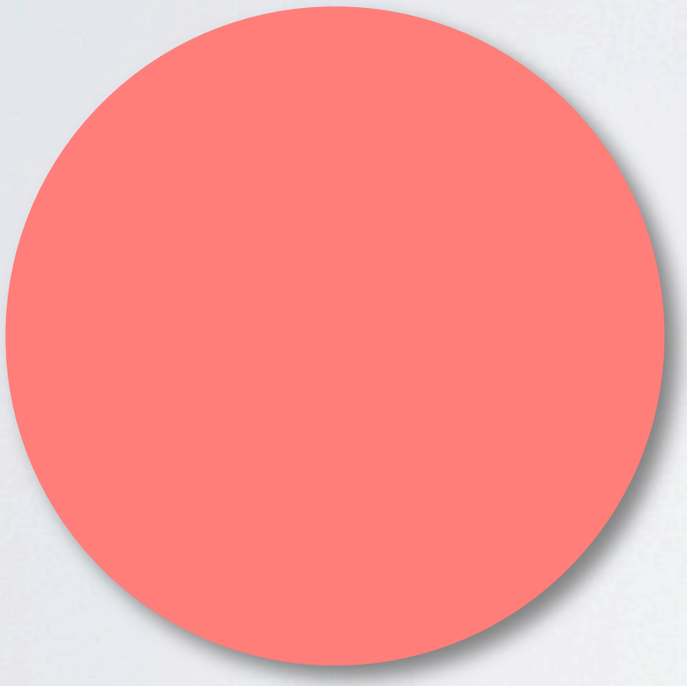


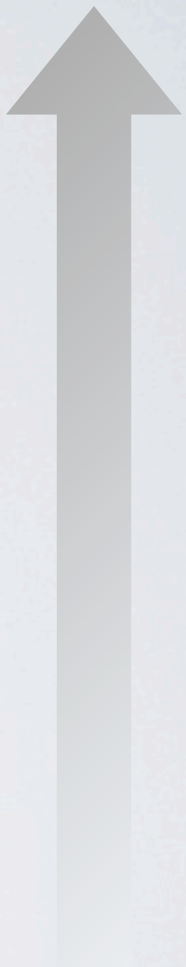


Level II
Meta Constructs

EVALUATION

π-interpreter

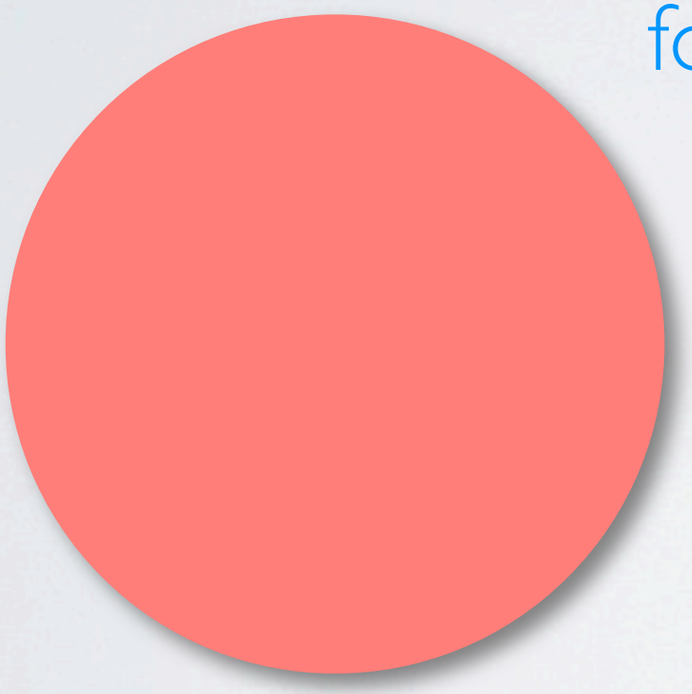




Level III
Libraries and Frameworks

EVALUATION

π-interpreter

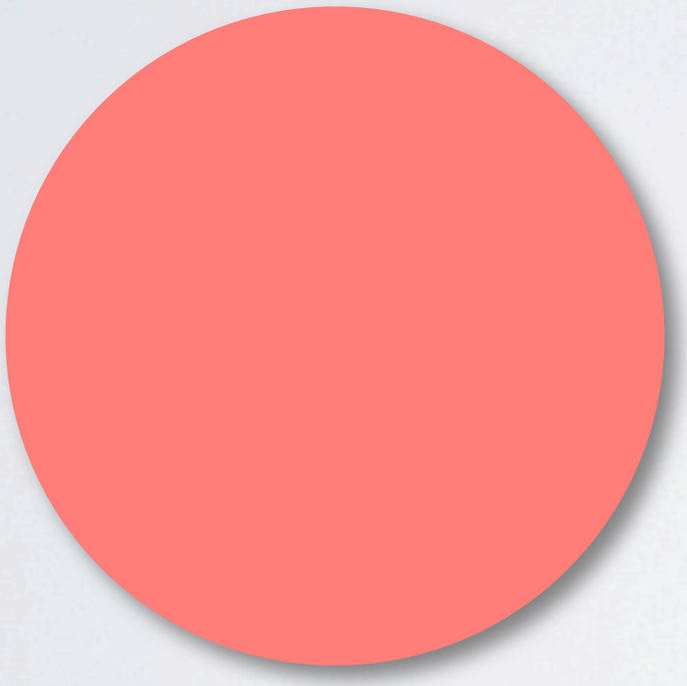


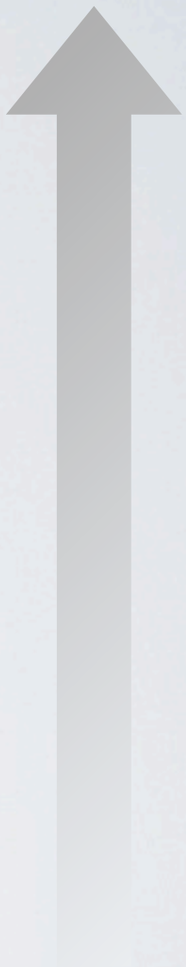
```
for_each_in (SELECT * FROM people)
  print (current.forename);
```

Level III
Libraries and Frameworks

EVALUATION

π-interpreter

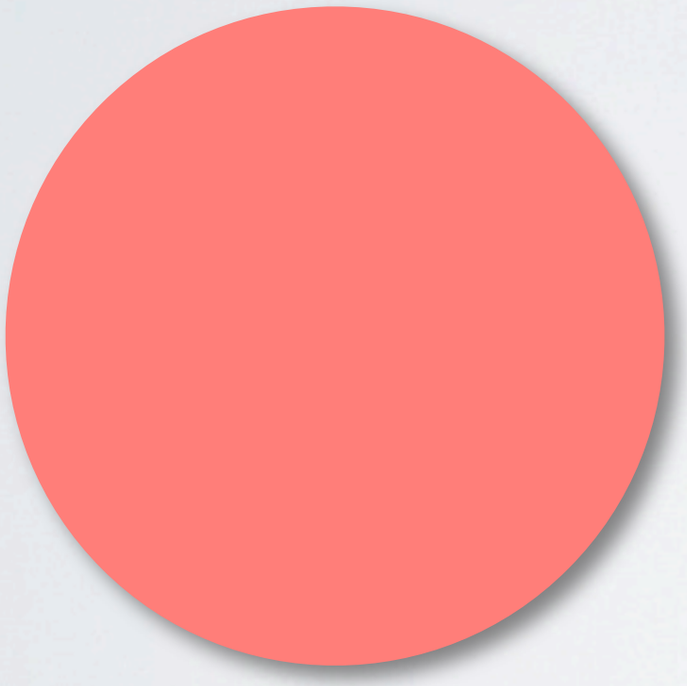


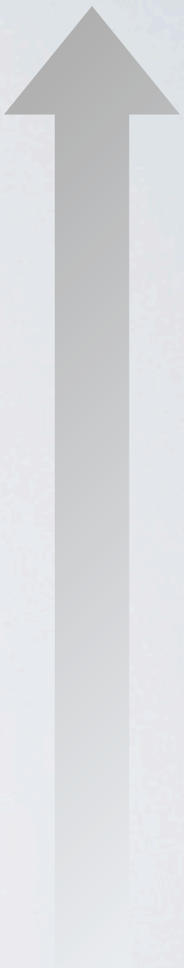


Level III
Libraries and Frameworks

EVALUATION

π-interpreter

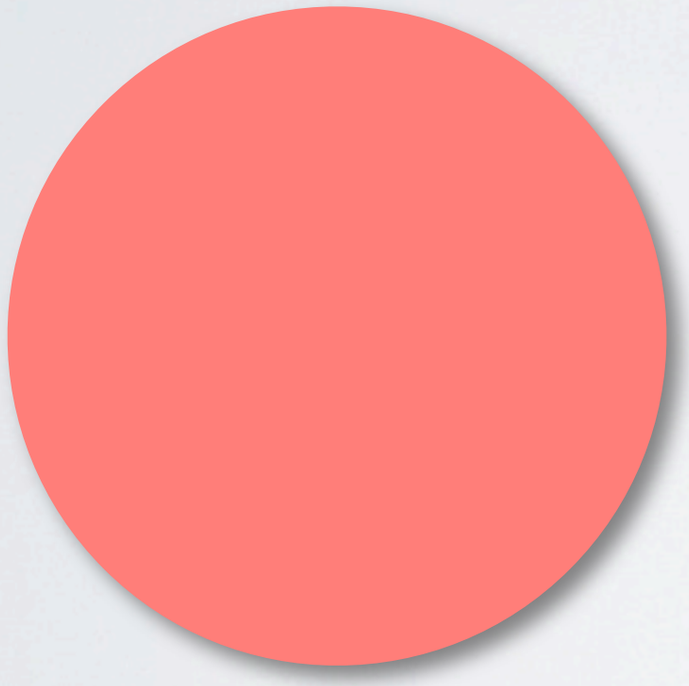




Level IV
Full Languages

EVALUATION

π-interpreter

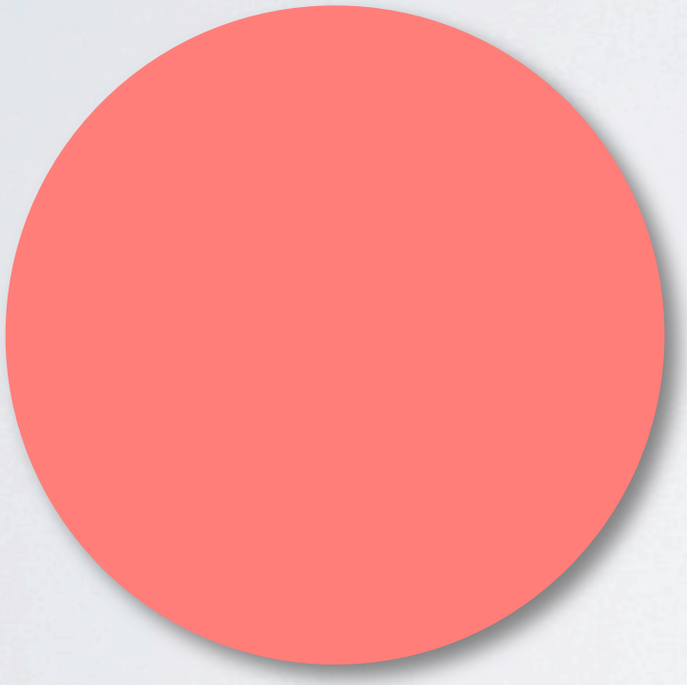


(λn.n+1 3)

Level IV
Full Languages

EVALUATION

π-interpreter



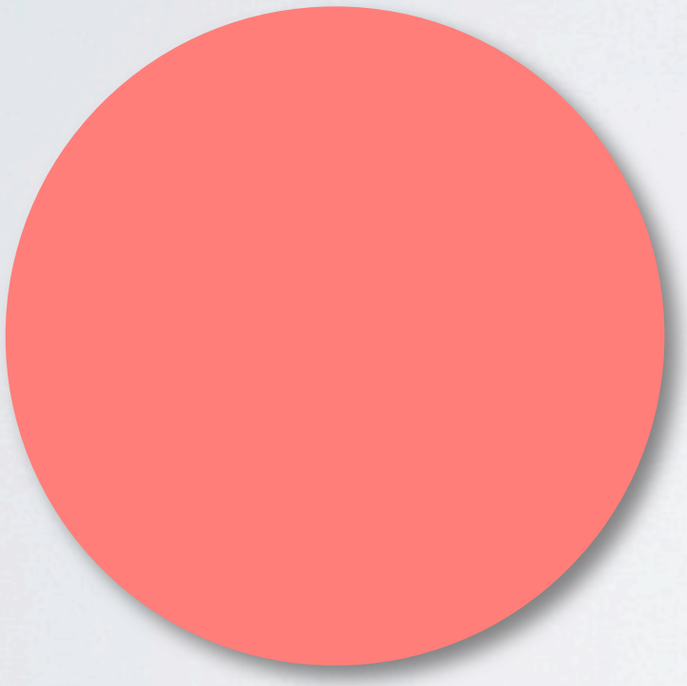
$(\lambda n.n+1\ 3)$

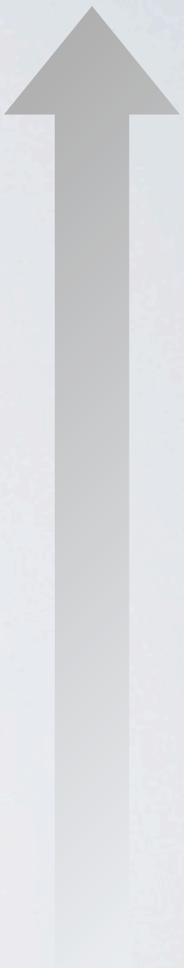
λ_expression :=
λ_variable |
λ_abstraction |
λ_application
⇒ λ_calculus

Level IV
Full Languages

EVALUATION

π-interpreter

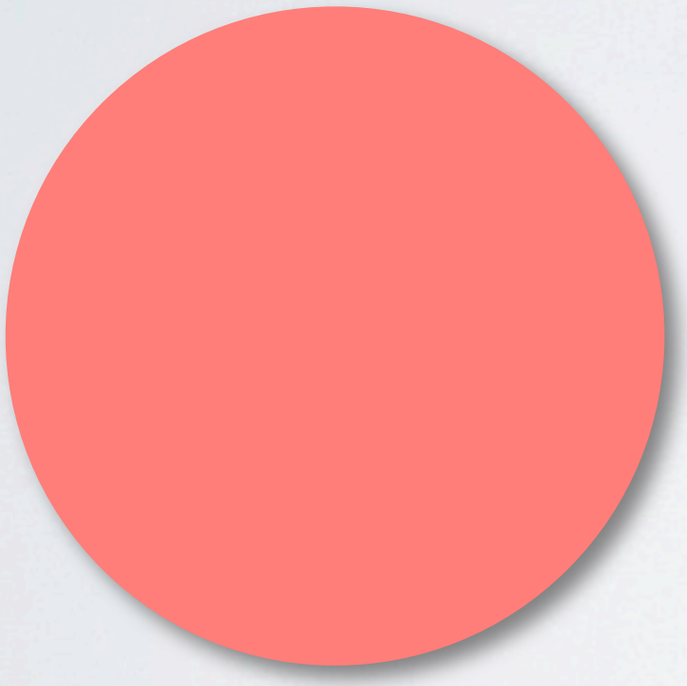


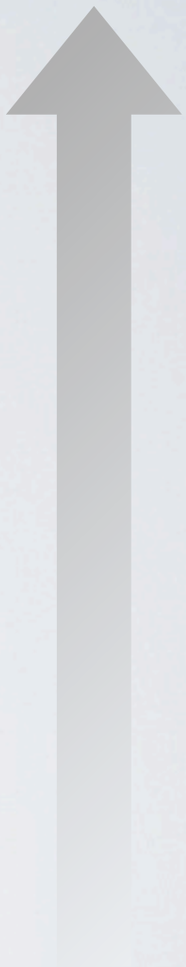


Level IV
Full Languages

EVALUATION

π-interpreter

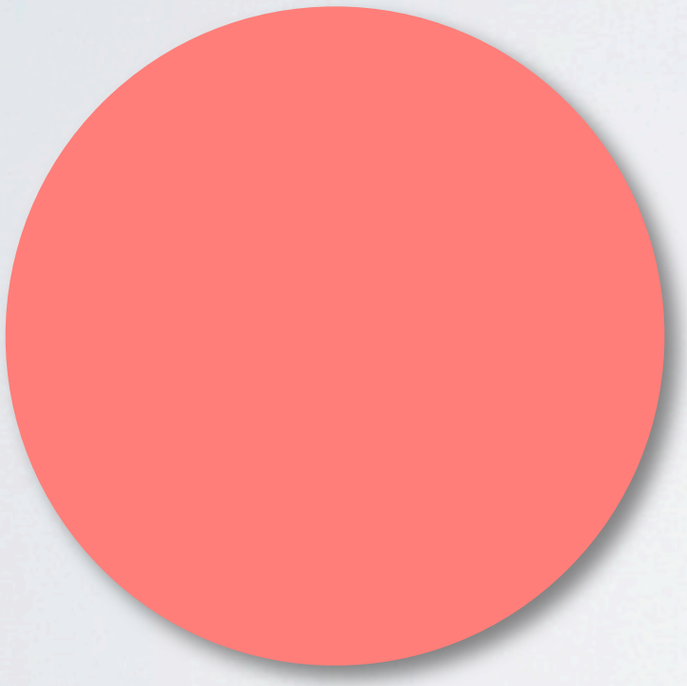




Level V
Meta Languages

EVALUATION

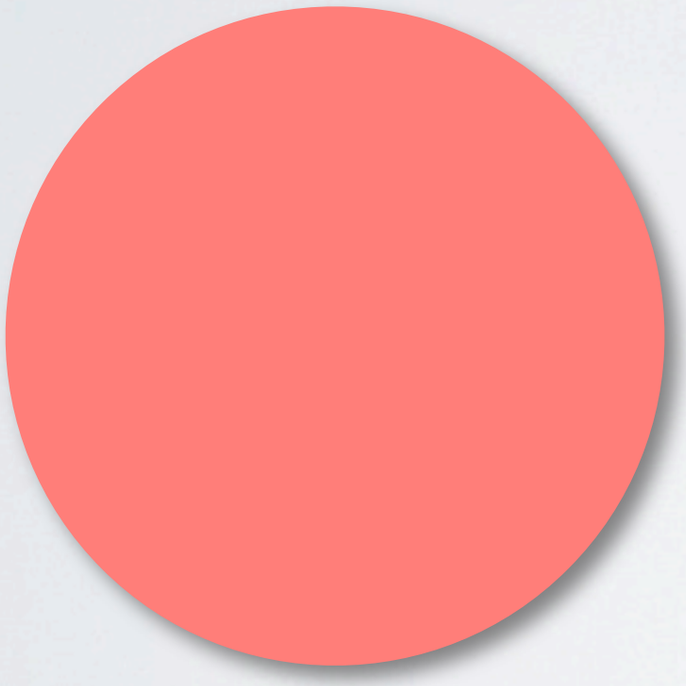
π-interpreter



product lines?
language families?

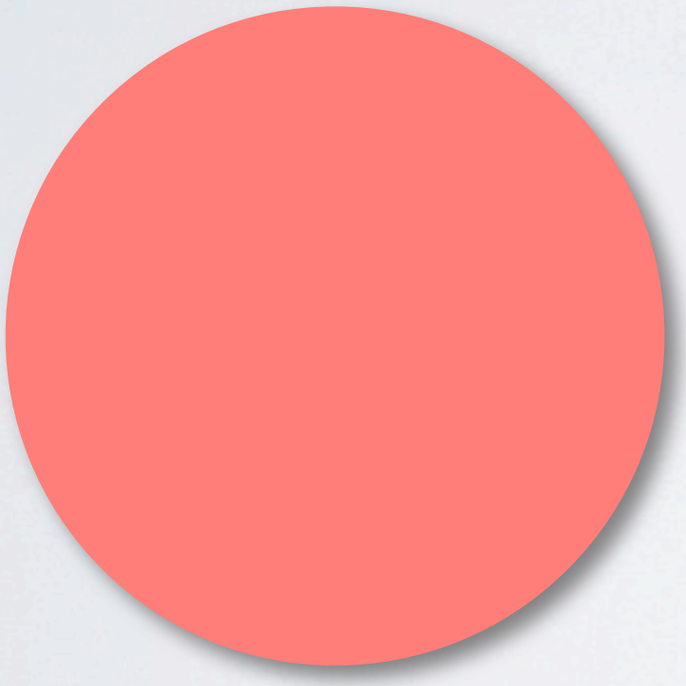
Level V
Meta Languages

A PLAN FOR GROWTH



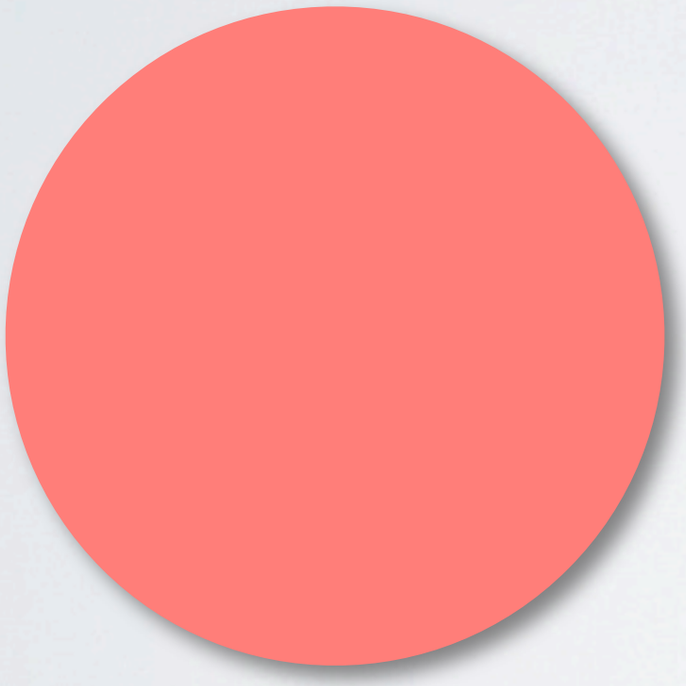
A PLAN FOR GROWTH

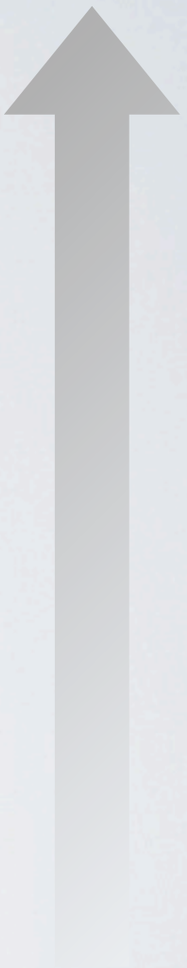
π-interpreter

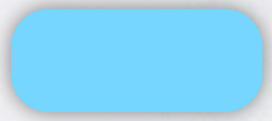


A PLAN FOR GROWTH

π-interpreter

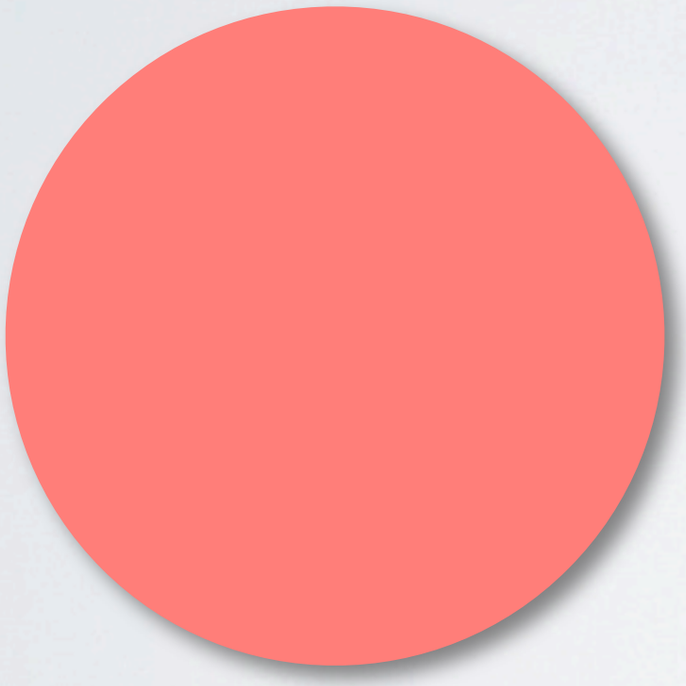


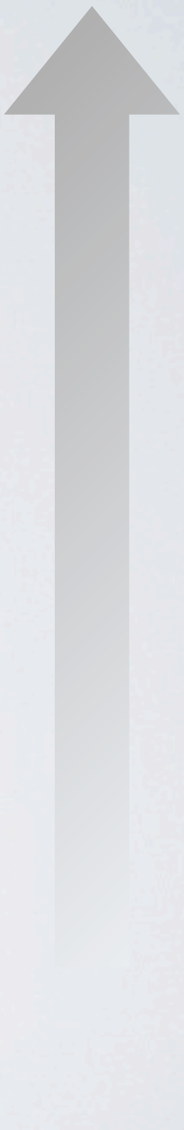




A PLAN FOR GROWTH

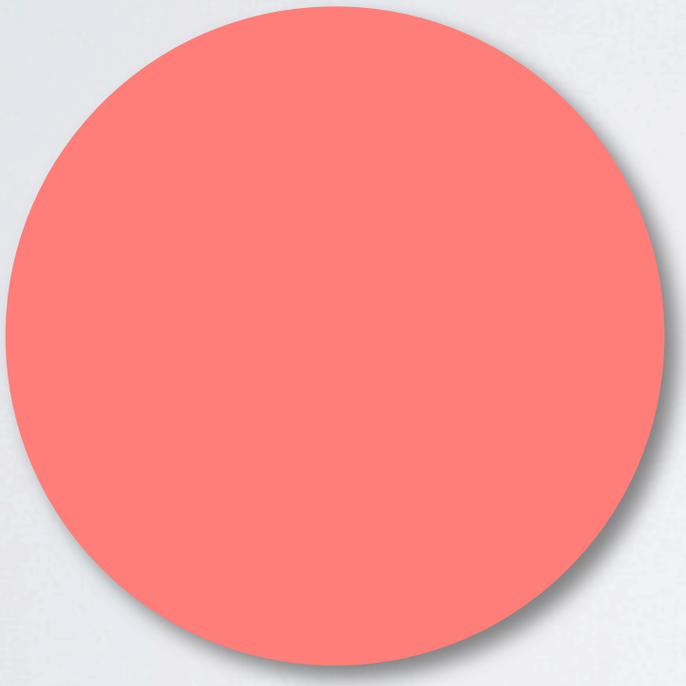
π-interpreter

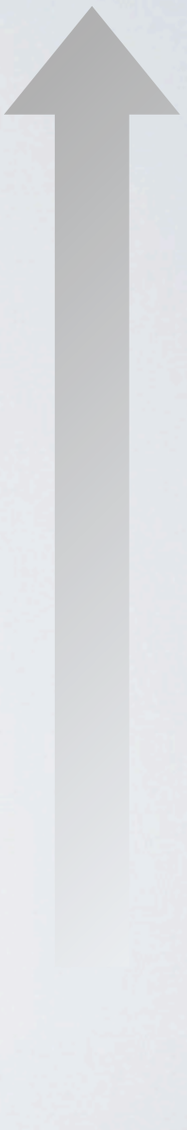




A PLAN FOR GROWTH

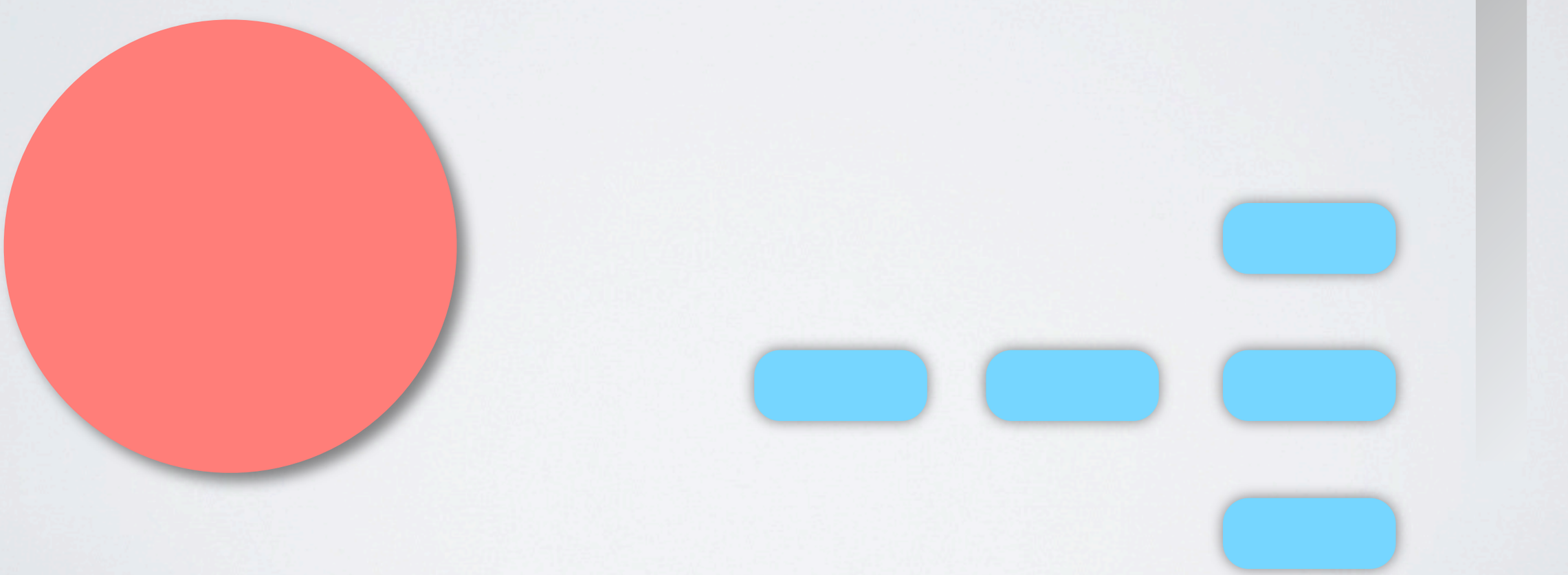
π-interpreter





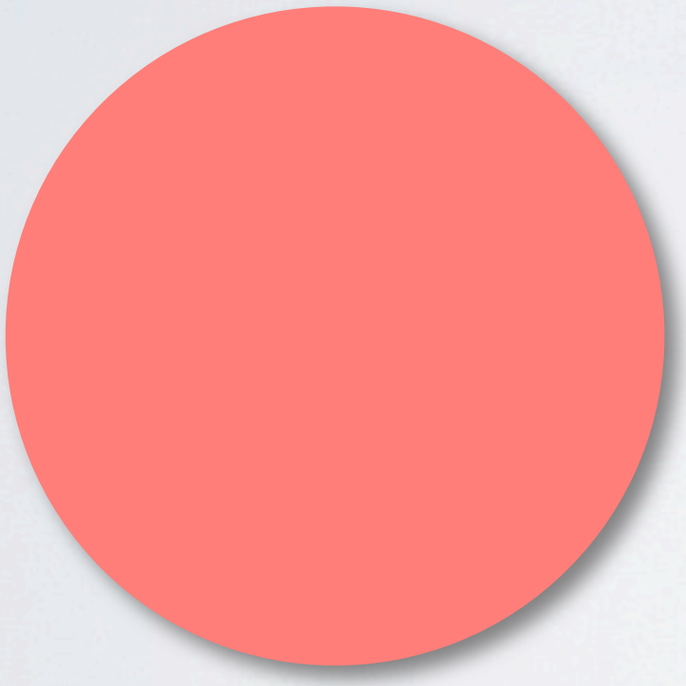
A PLAN FOR GROWTH

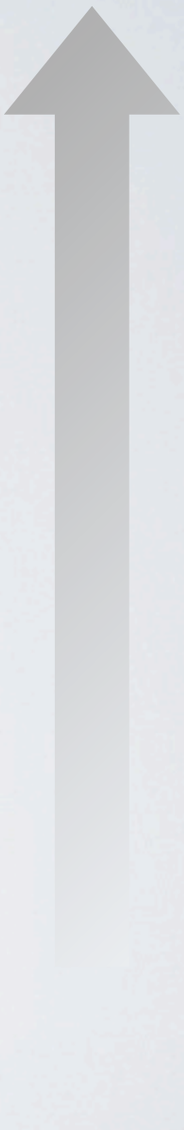
π-interpreter



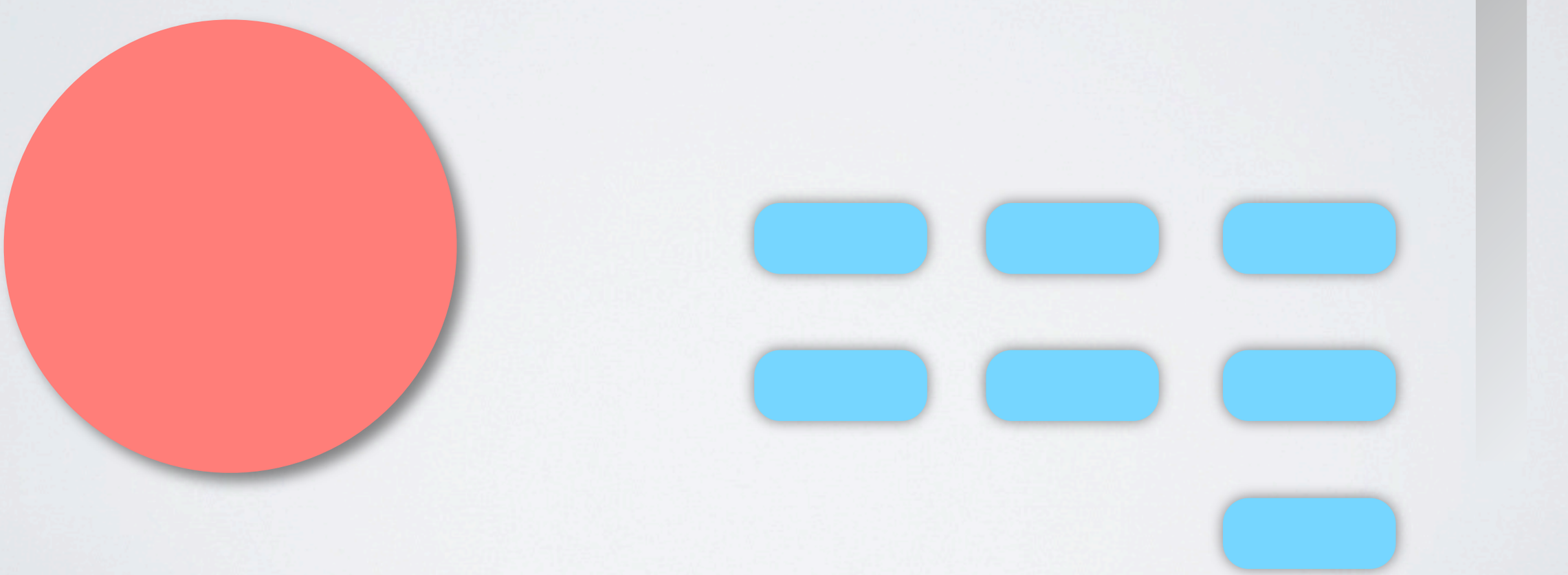
A PLAN FOR GROWTH

π-interpreter





A PLAN FOR GROWTH

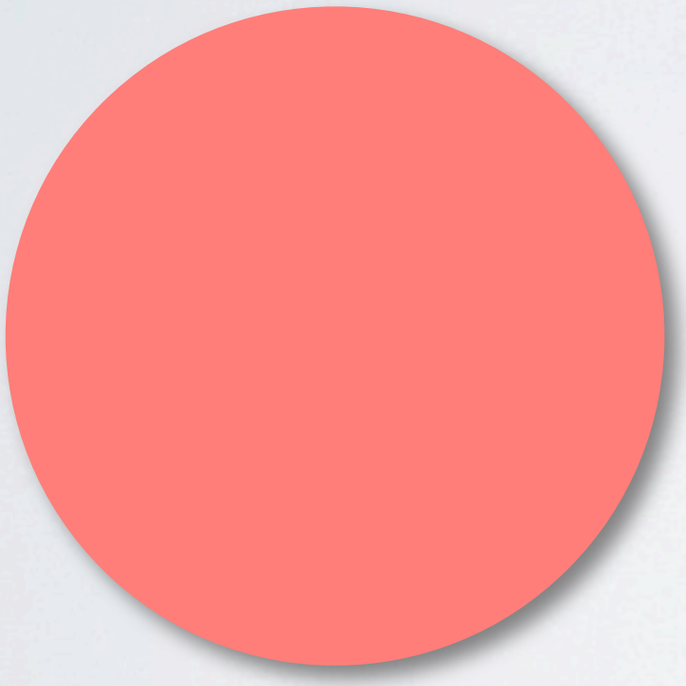


A PLAN FOR GROWTH

π-interpreter

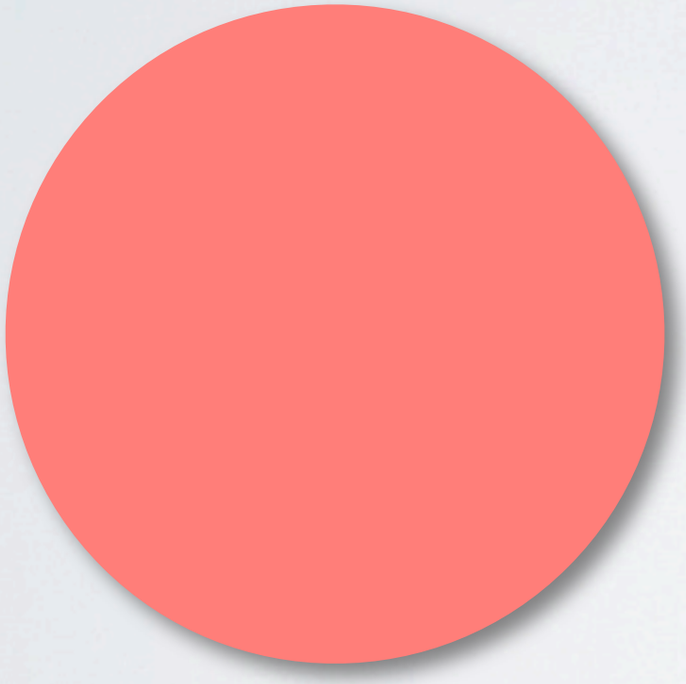
A PLAN FOR GROWTH

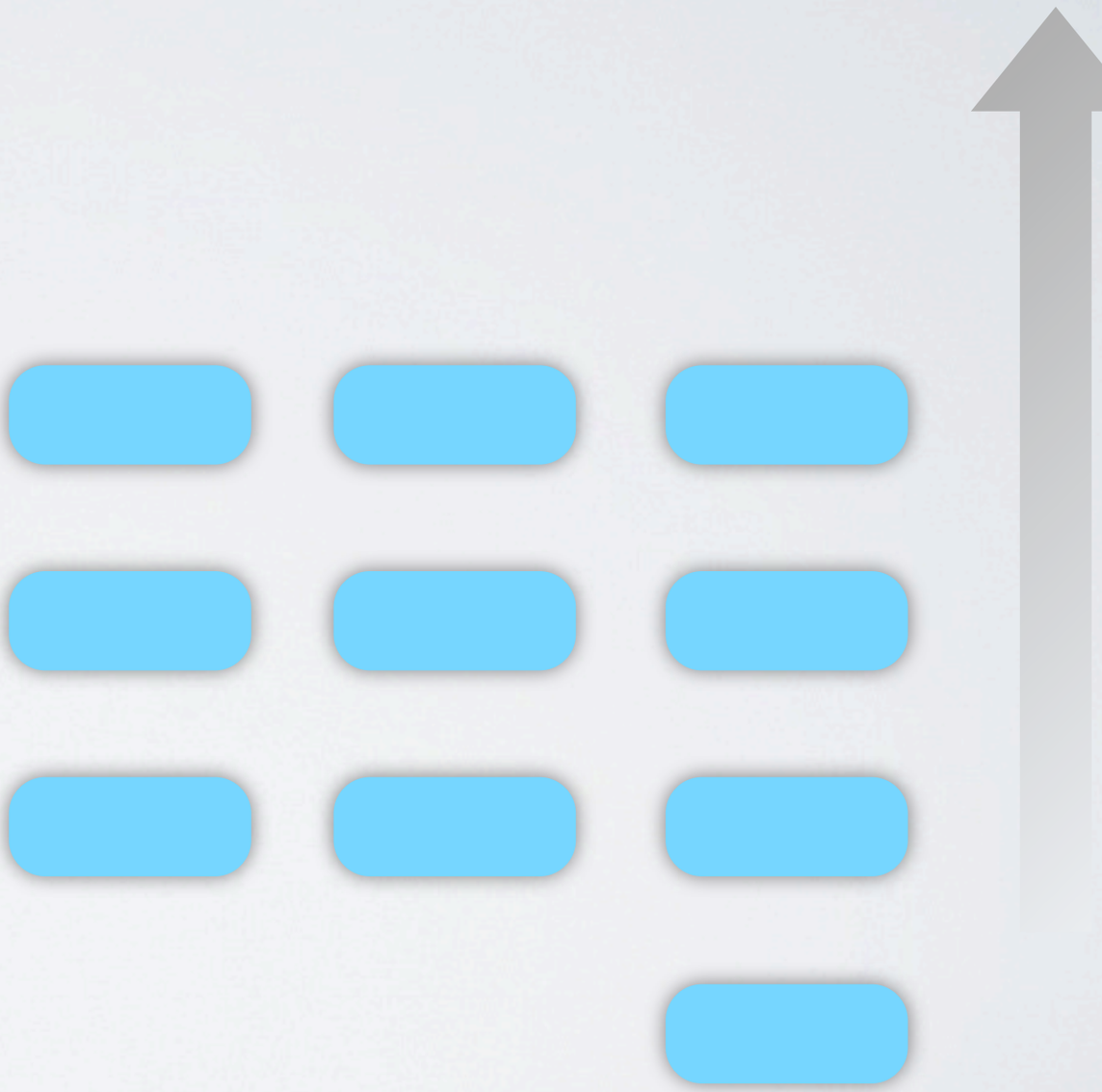
π-interpreter



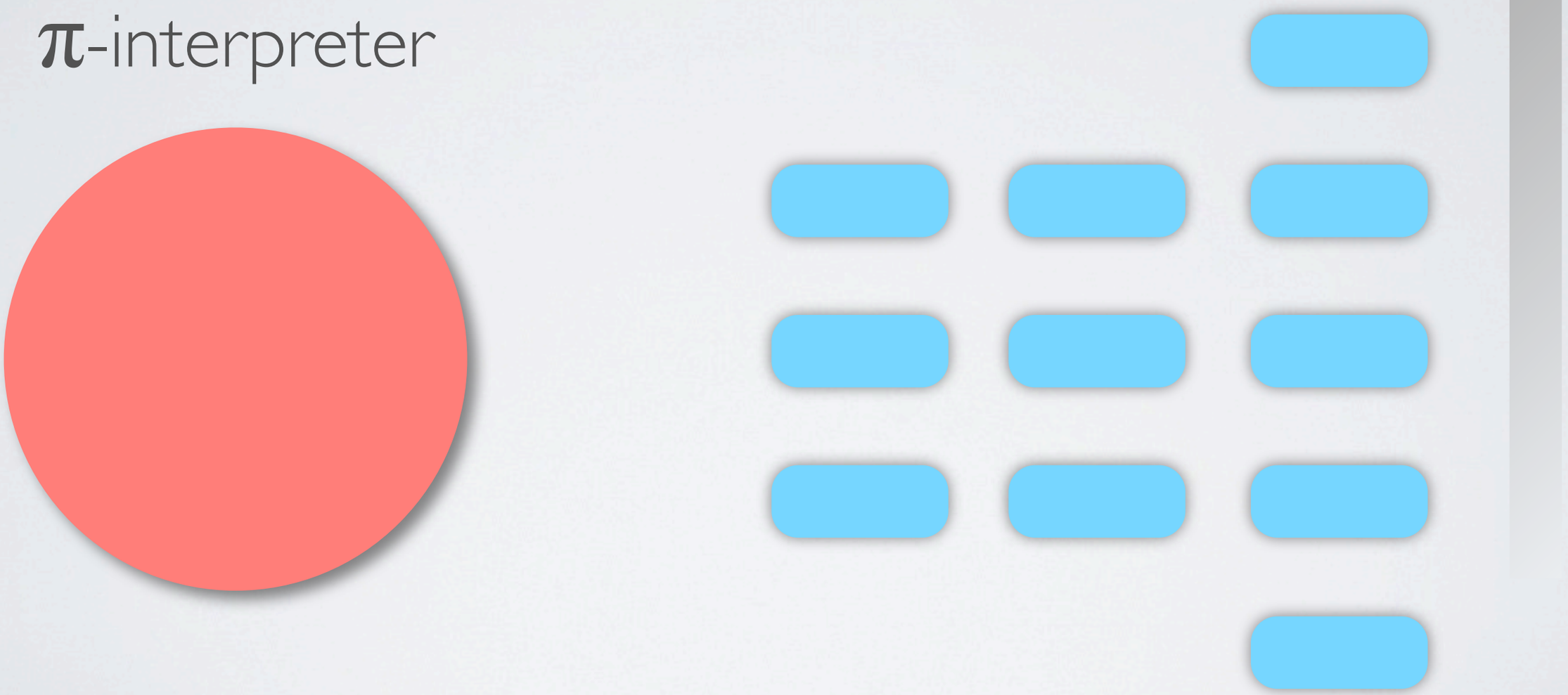
A PLAN FOR GROWTH

π-interpreter





A PLAN FOR GROWTH



A PLAN FOR GROWTH

patterns

RELATED IDEAS

patterns

RELATED IDEAS

extensible languages

super languages
Katahdin
XMF

patterns

RELATED IDEAS

extensible languages

language oriented programming

patterns

RELATED IDEAS

extensible languages

patterns

language design tools

extension tools
JSE
Stratego/XL
Ometa

RELATED IDEAS

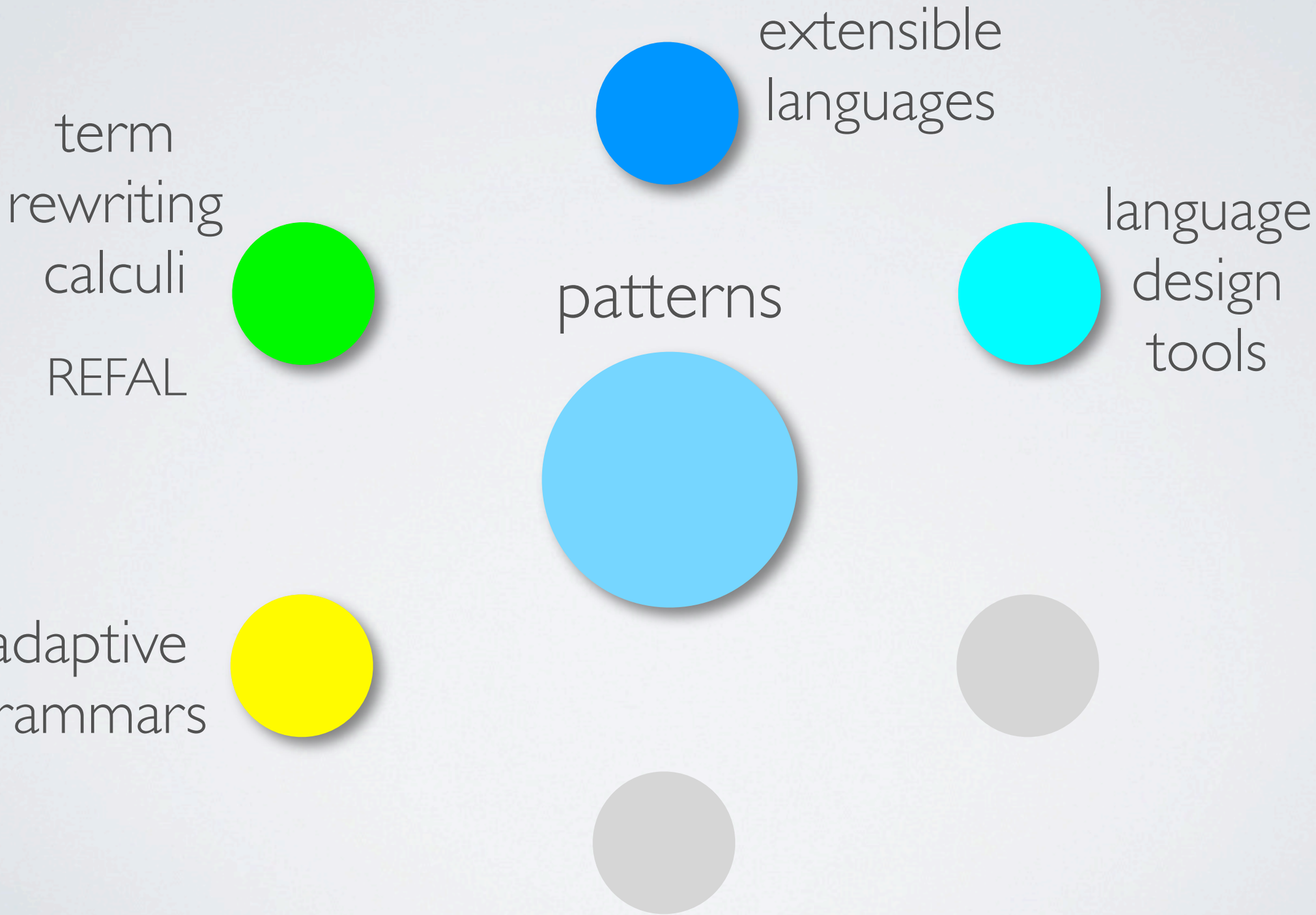
extensible languages

patterns

language design tools

compiler-compiler
JastAdd
ANTLR

RELATED IDEAS



RELATED IDEAS

extensible languages

term rewriting calculi

language design tools

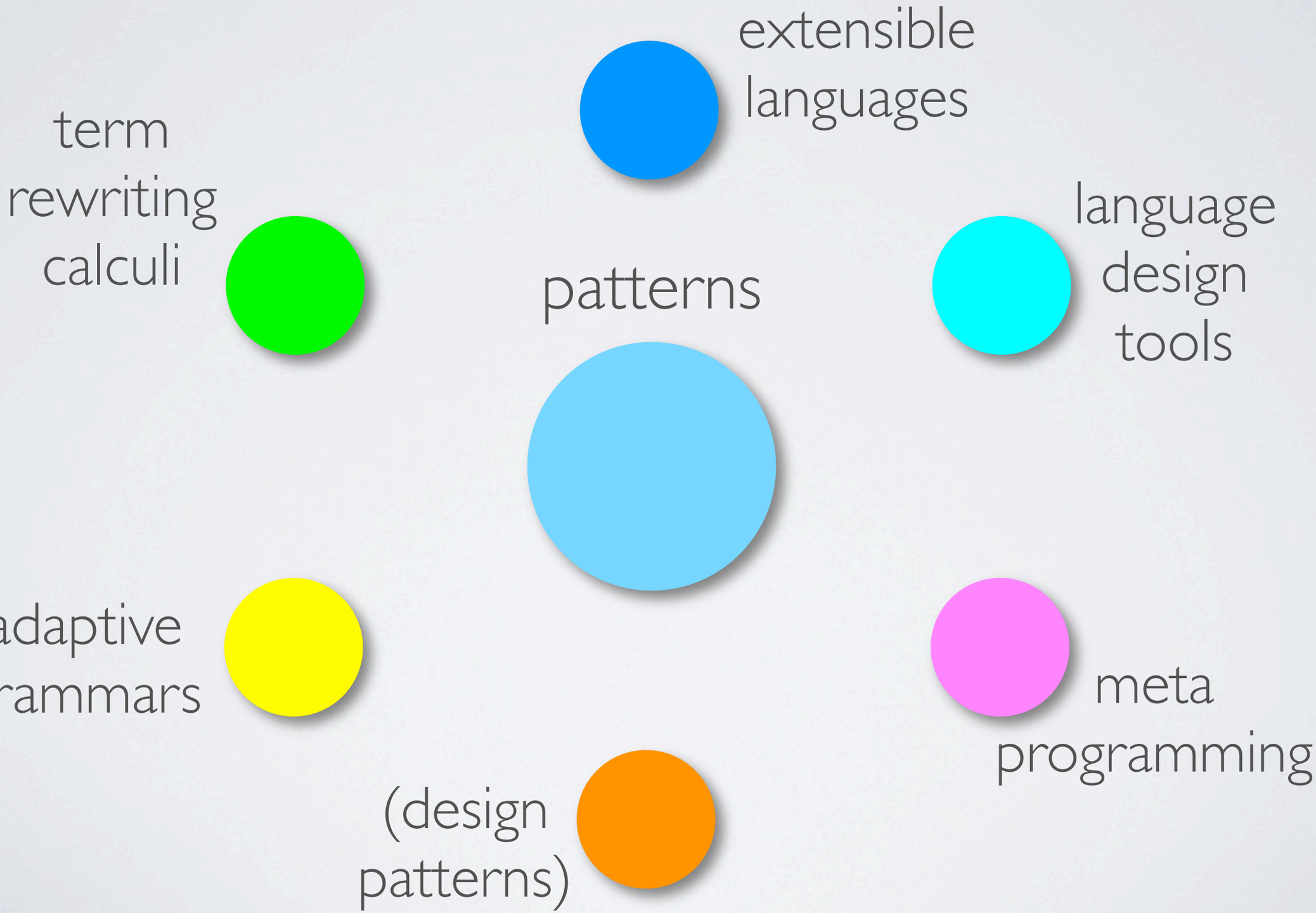
patterns

adaptive grammars

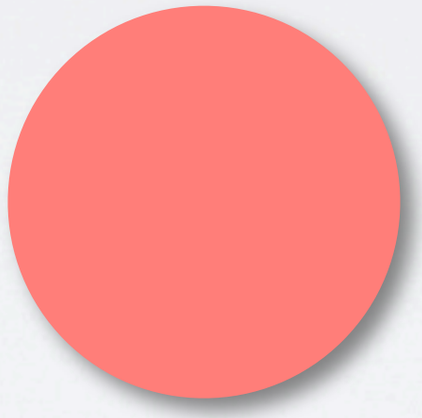
meta programming

Lexical/Syntactic Macros; Generative, Multi-Stage-, Generic Programming

RELATED IDEAS

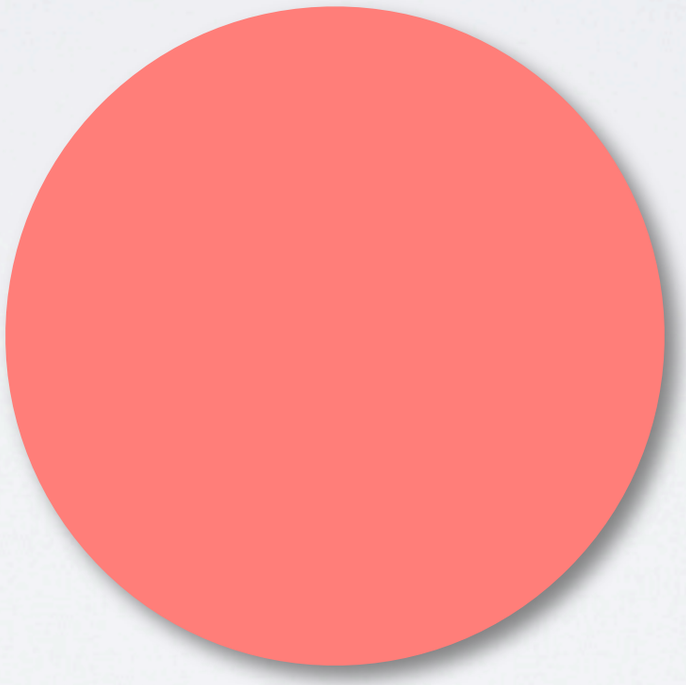


TODAY



TODAY

π-interpreter



TODAY

π-interpreter

semantic
recursion

TODAY

π-interpreter

semantic
recursion

operator
precedence

TODAY

π-interpreter

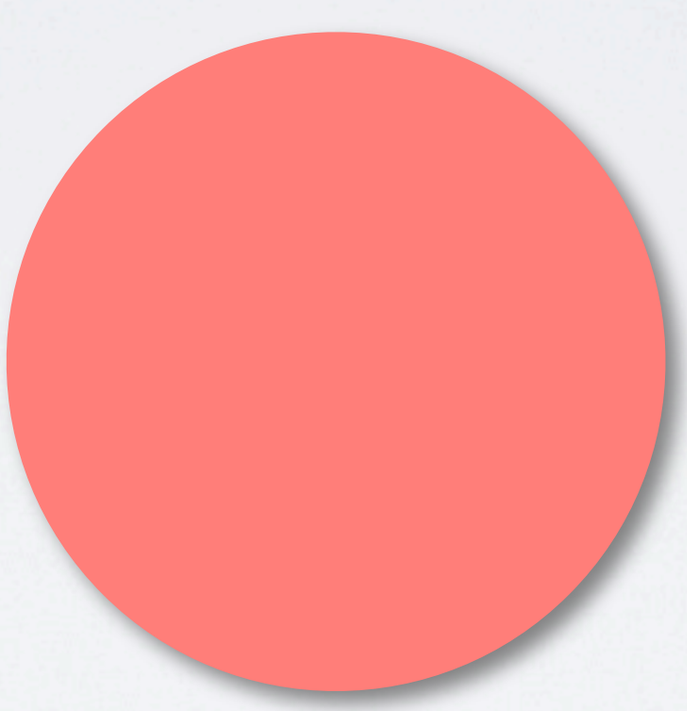
semantic
recursion

nested
dynamic
pattern
declarations

operator
precedence

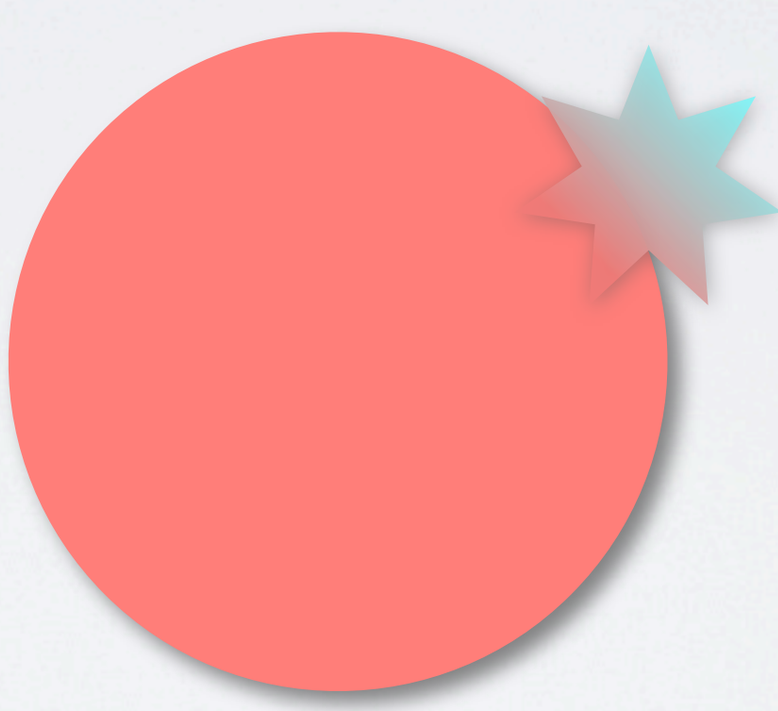
TOMORROW

π-interpreter



TOMORROW

π-interpreter



static (syntax) analysis?

TOMORROW

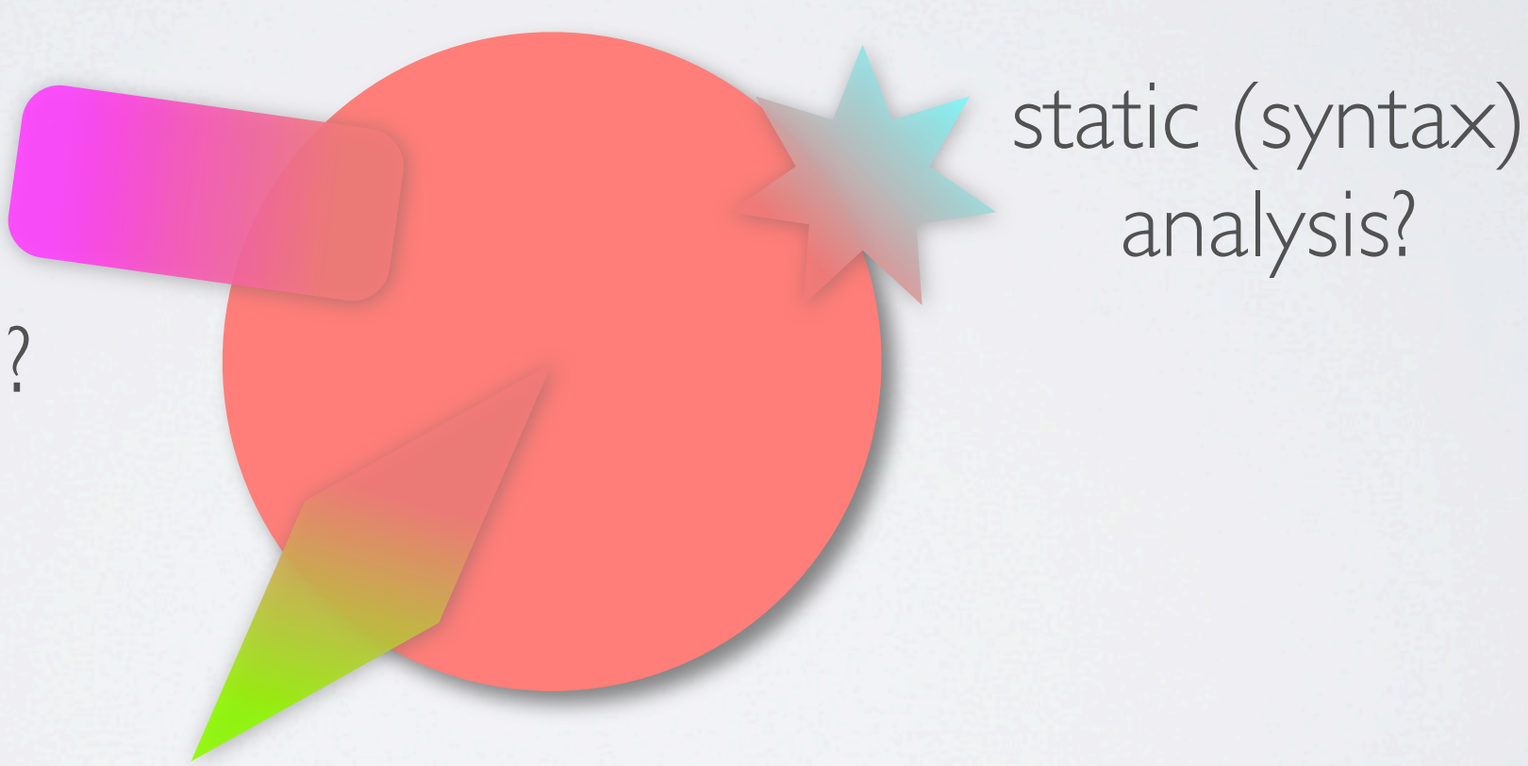
π-interpreter

static (syntax) analysis?

a **π**-calculus?

TOMORROW

π-interpreter



IDE integration?

static (syntax) analysis?

a **π**-calculus?

TOMORROW

π-interpreter

IDE integration?

static (syntax) analysis?

a **π**-calculus?

pattern sharing community?

THE CHANCE

THE CHANCE

a hyperlanguage

THE CHANCE

abstraction of related ideas

THE CHANCE

learnability & presentability

THE CHANCE

understandability &
sustainability

THE CHANCE

structuring & abstraction

THE CHANCE

productivity & expressibility

THE CHANCE

individuality? & freedom?

THE CHANCE

evolutionary progress

THE CHANCE

a renaissance of the origins?

THE CHANCE

"democratization"
of language design?

THE CHANCE

a plea for the importance of syntax

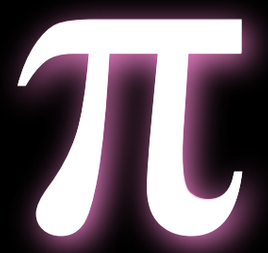
THE CHANCE

… against "meta-ization"

THE CHANCE

… for the focus on language design
& against "tool-erism"

THE CHANCE



Onward! 2009

TUD - Technische Universität Darmstadt

Roman Knöll & Mira Mezini

pi-programming.org

This research is part of the Pegasus Project at TUD
for creating a natural language programming system.
http://www.pegasus-project.org