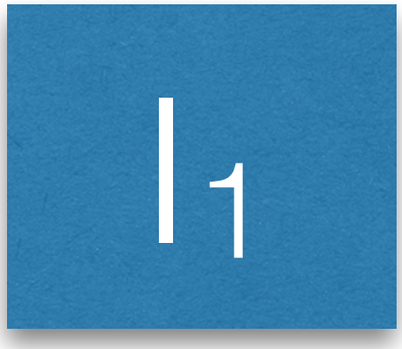
IODyn: A High-Level Language for Incremental Computation

By Kyle Headley
University of Colorado Boulder
kyleheadley.github.io

What is Incremental Computation?

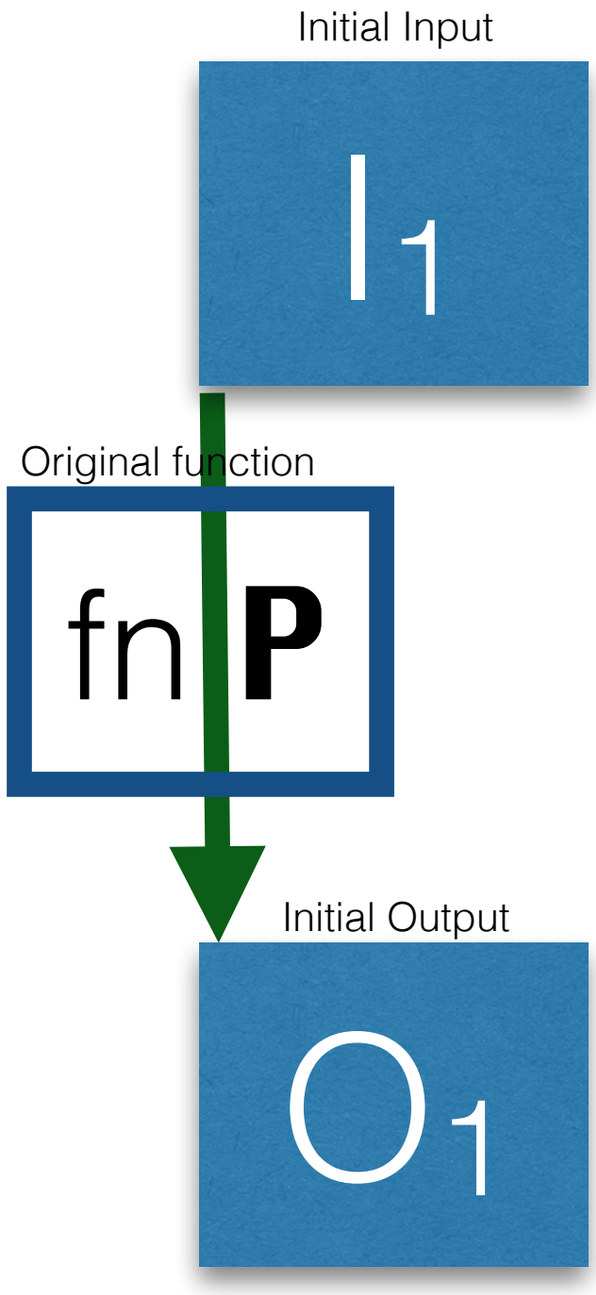
What is Incremental Computation?

Initial Input

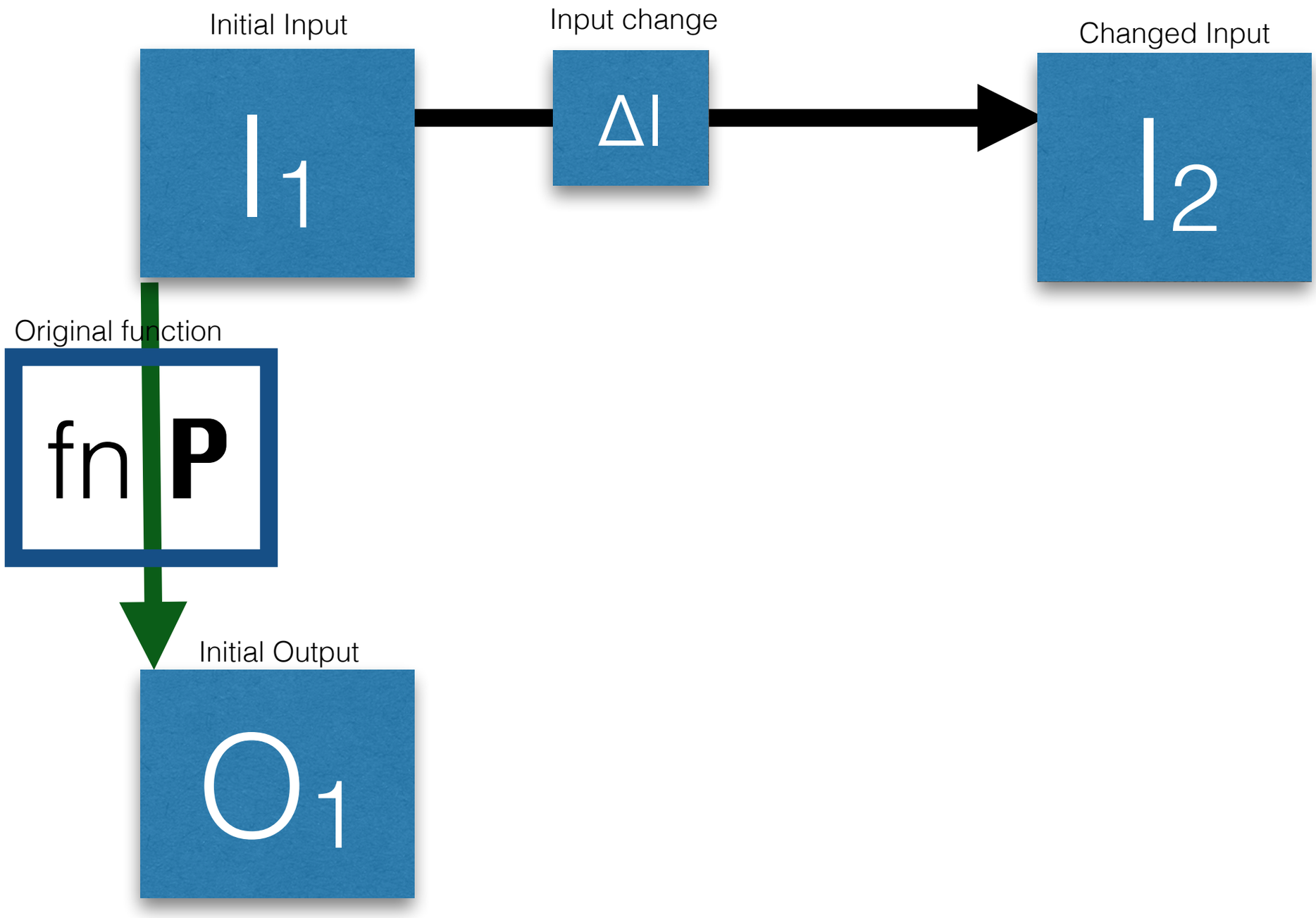


Original function

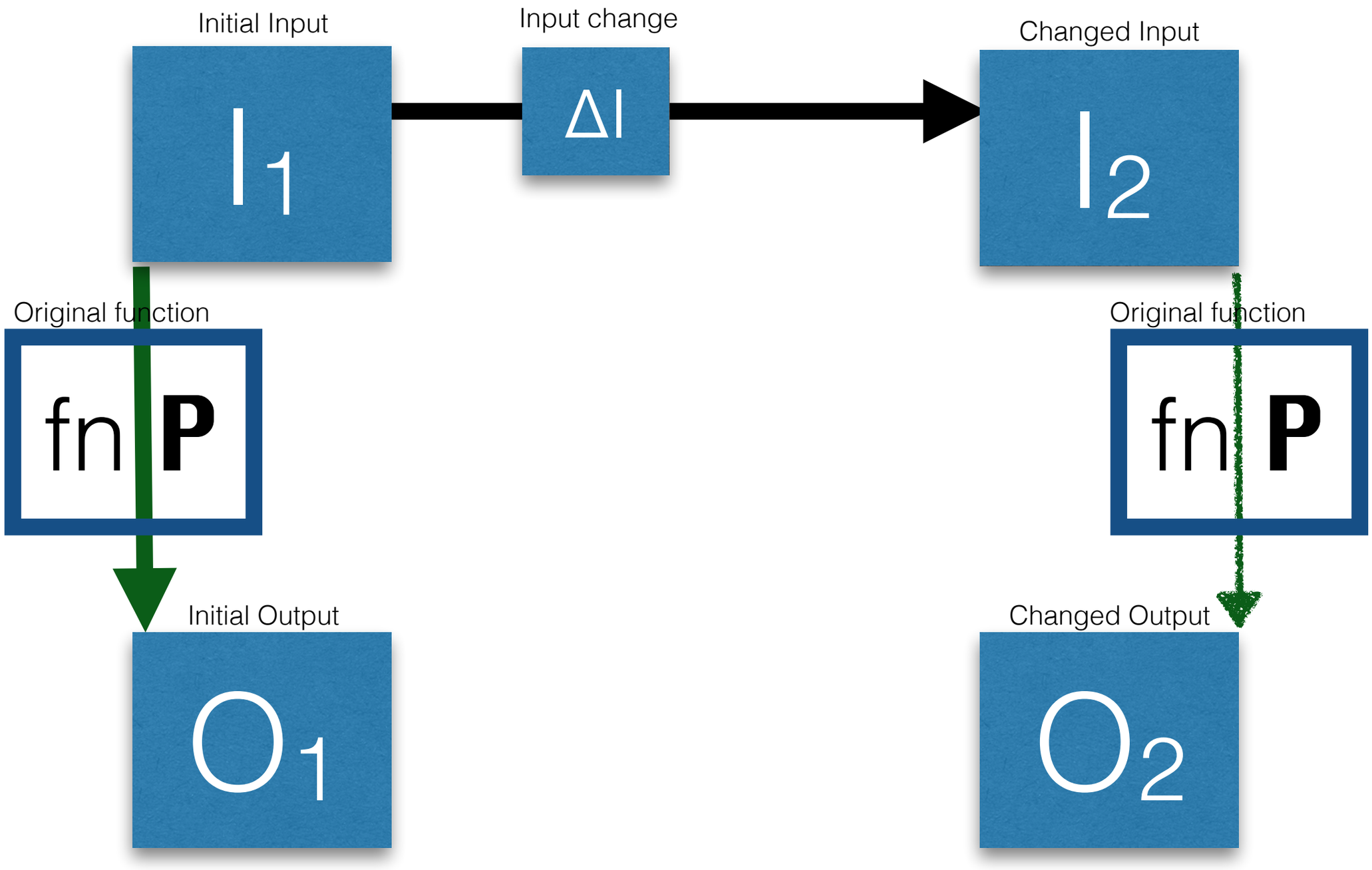
What is Incremental Computation?



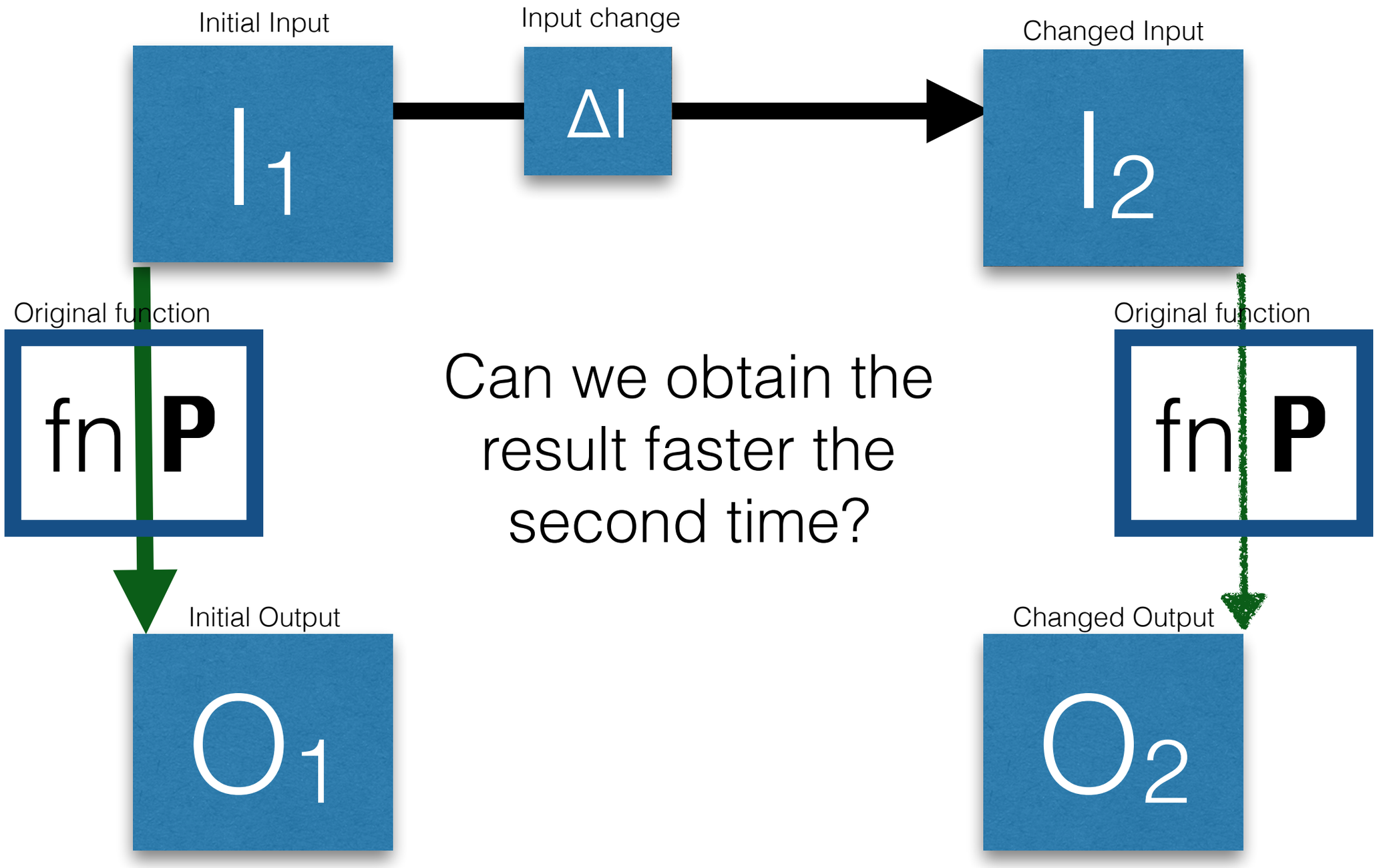
What is Incremental Computation?



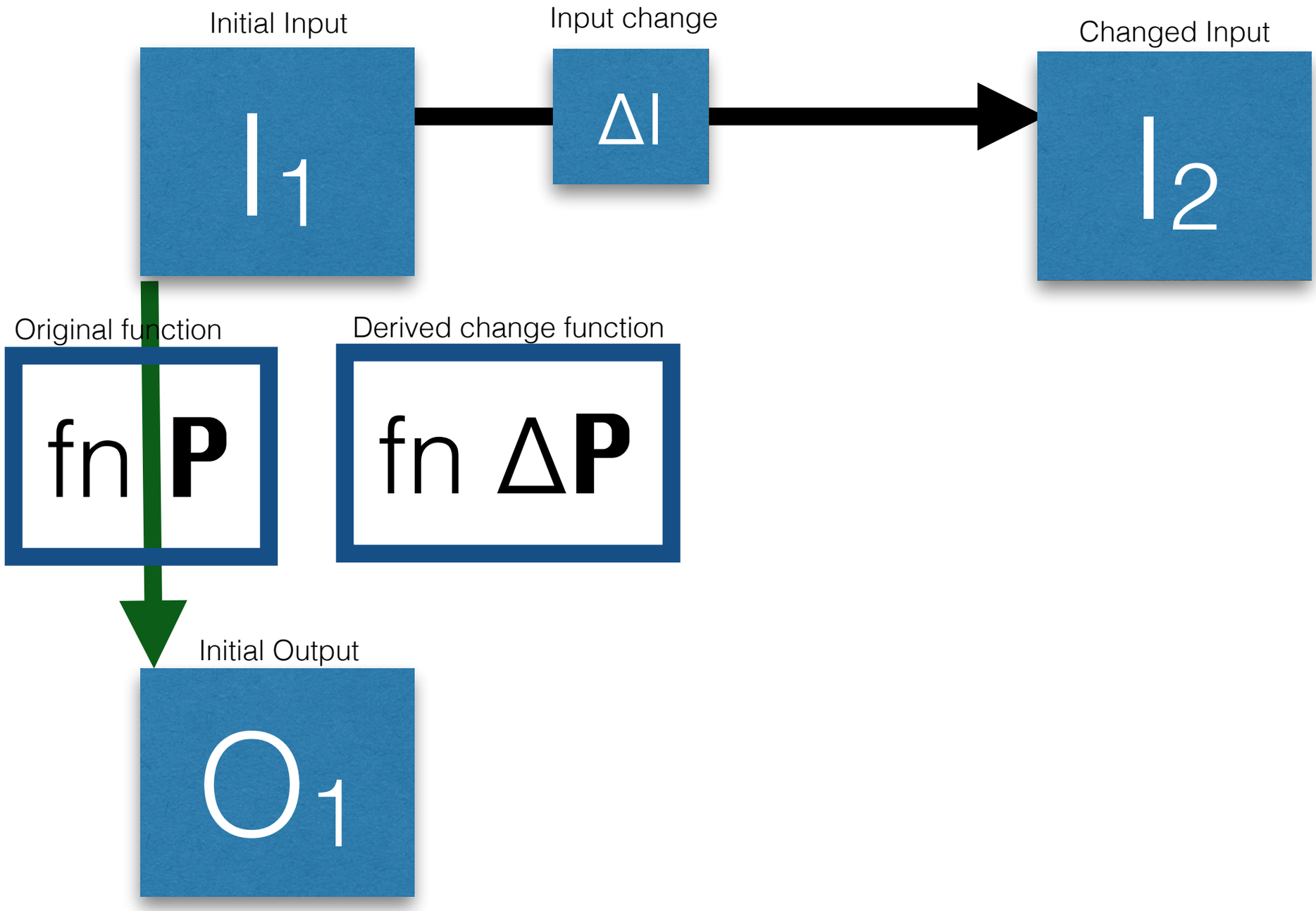
What is Incremental Computation?



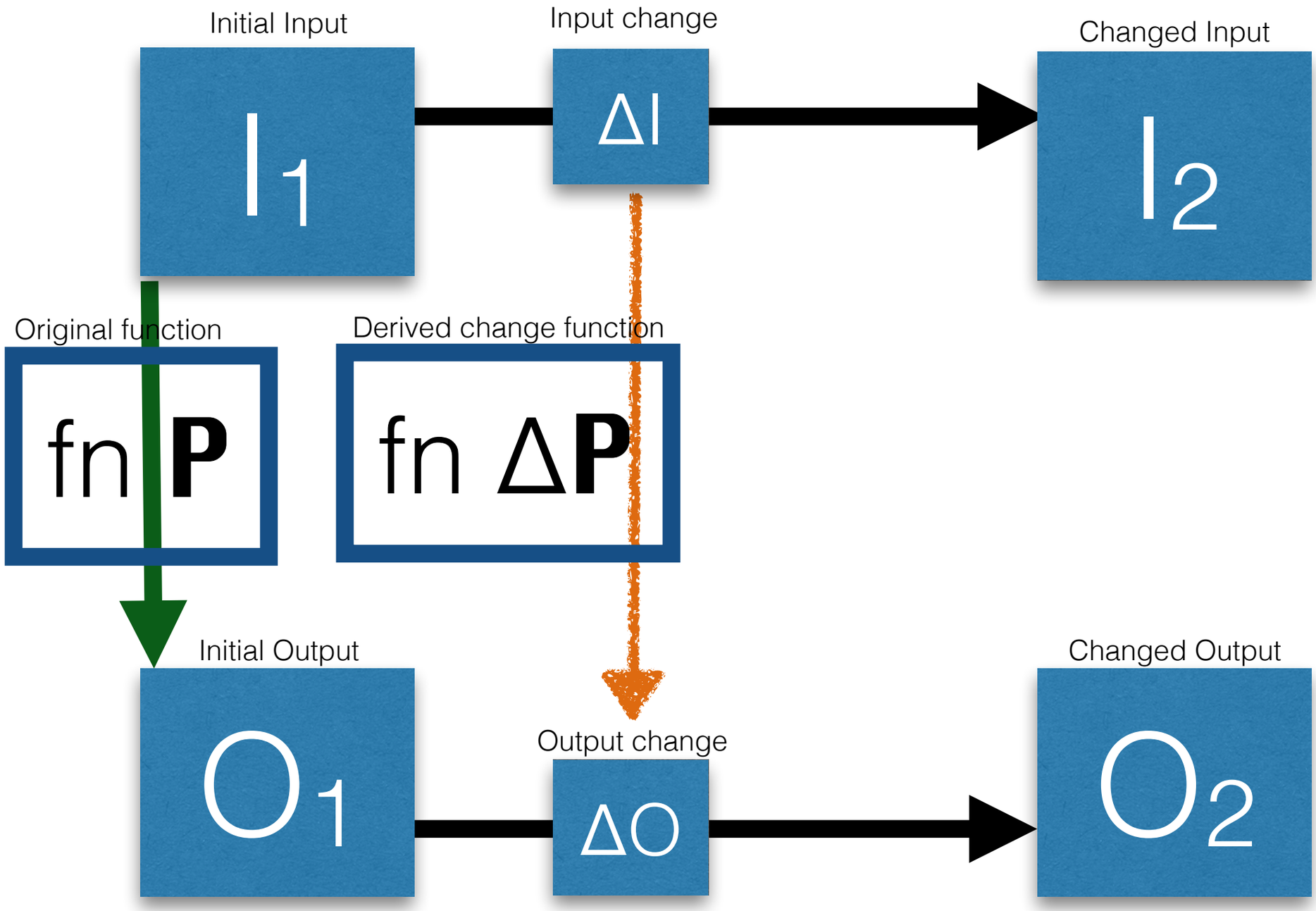
What is Incremental Computation?



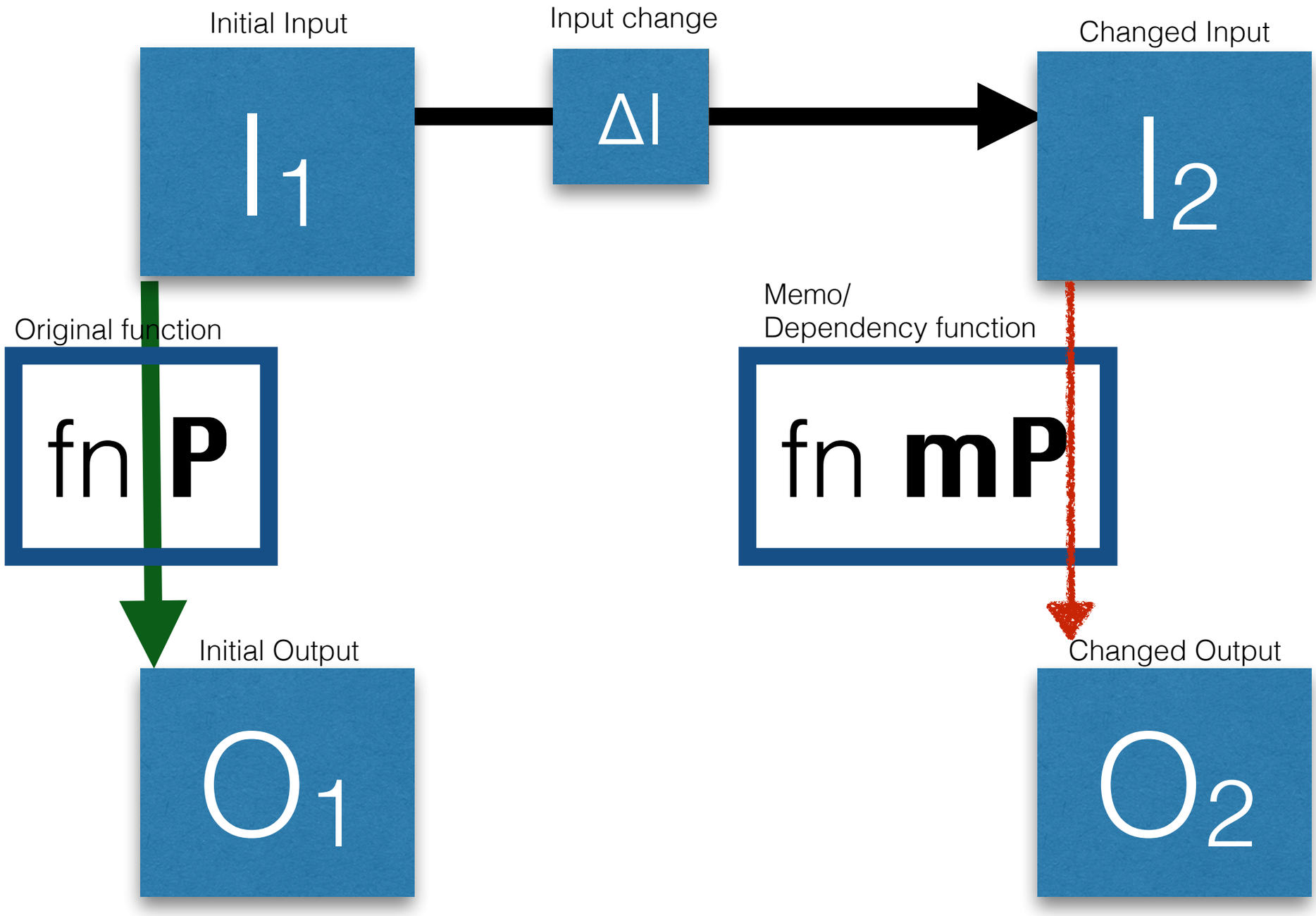
What is Incremental Computation?



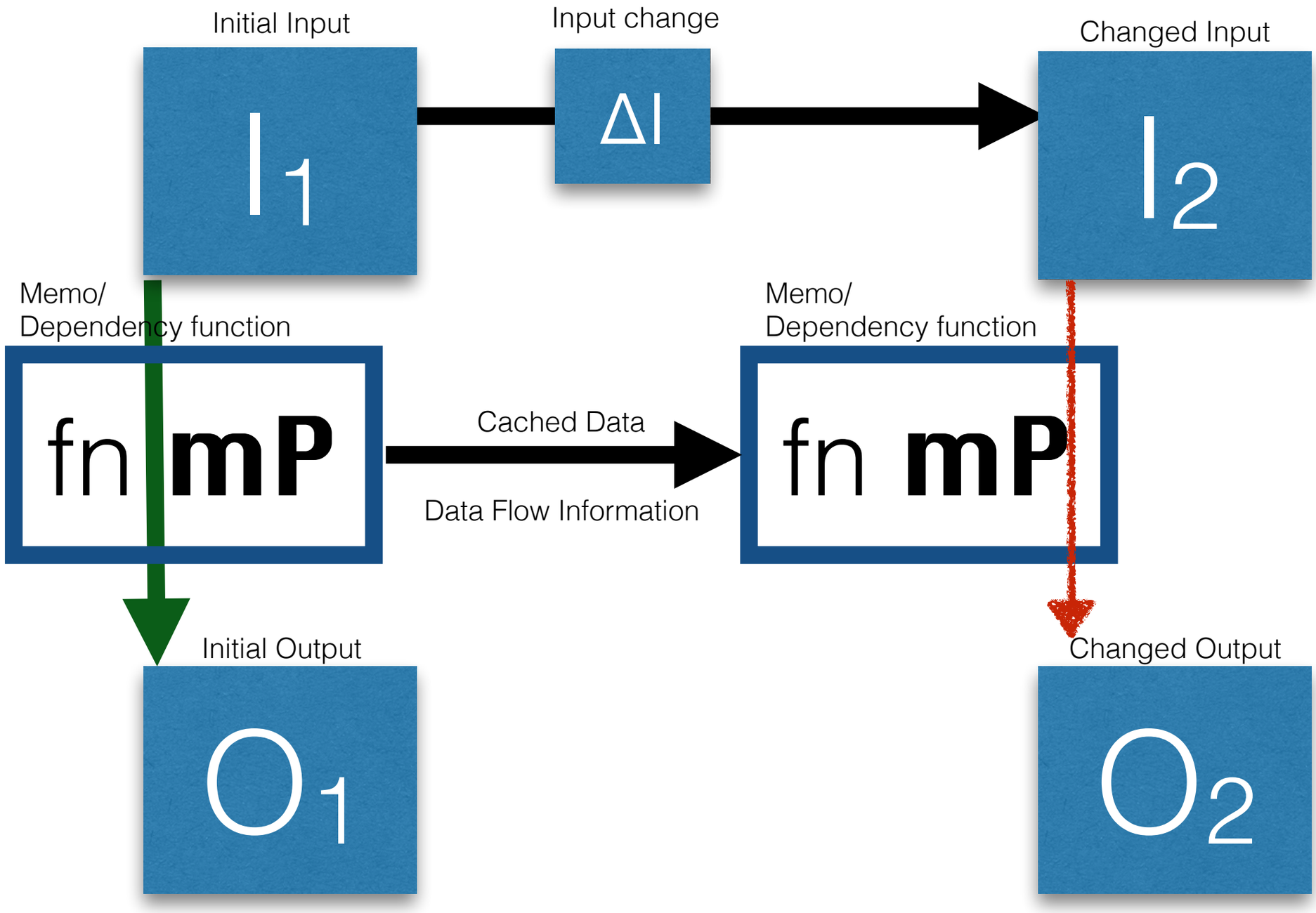
What is Incremental Computation?



What is Incremental Computation?



What is Incremental Computation?



How do we use Incremental Computation?

General-purpose incremental computation engines are accessed through libraries

Programmers annotate variables and functions based on their knowledge of the program and its incremental behavior

How do we use Incremental Computation?

General-purpose incremental computation engines are accessed through libraries

Programmers annotate variables and functions based on their knowledge of the program and its incremental behavior

But library use may not be straightforward and improper caching can lead to slowdowns or incorrect results

How do we improve Incremental Computation?

Speedups

Correctness

How do we improve Incremental Computation?

Speedups

Caching strategies that work efficiently for common computation tasks

Correctness

Static guarantees of proper use of incremental features

How do we improve Incremental Computation?

Speedups

Caching strategies that work efficiently for common computation tasks

Correctness

Static guarantees of proper use of incremental features

Our lab is working on:

Incremental Collections Libraries

Type system for use of library features

How do we improve Incremental Computation?

Speedups

Caching strategies that work efficiently for common computation tasks

Correctness

Static guarantees of proper use of incremental features

Our lab is working on:

Incremental Collections Libraries

Type system for use of library features

But these are specific solutions, with faults:

Conflicts between multiple applied incremental functions may still interfere with cache behavior

Writing the code is still difficult, and it provides no guarantees of performance

IODyn

Implicitly Incremental Language

Work in progress:

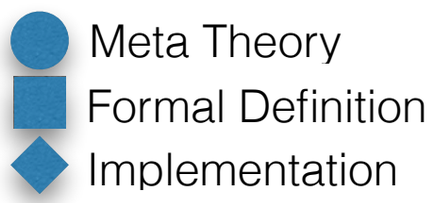
github.com/cuplv/iodyn-lang.rust

Based on the lambda-calculus

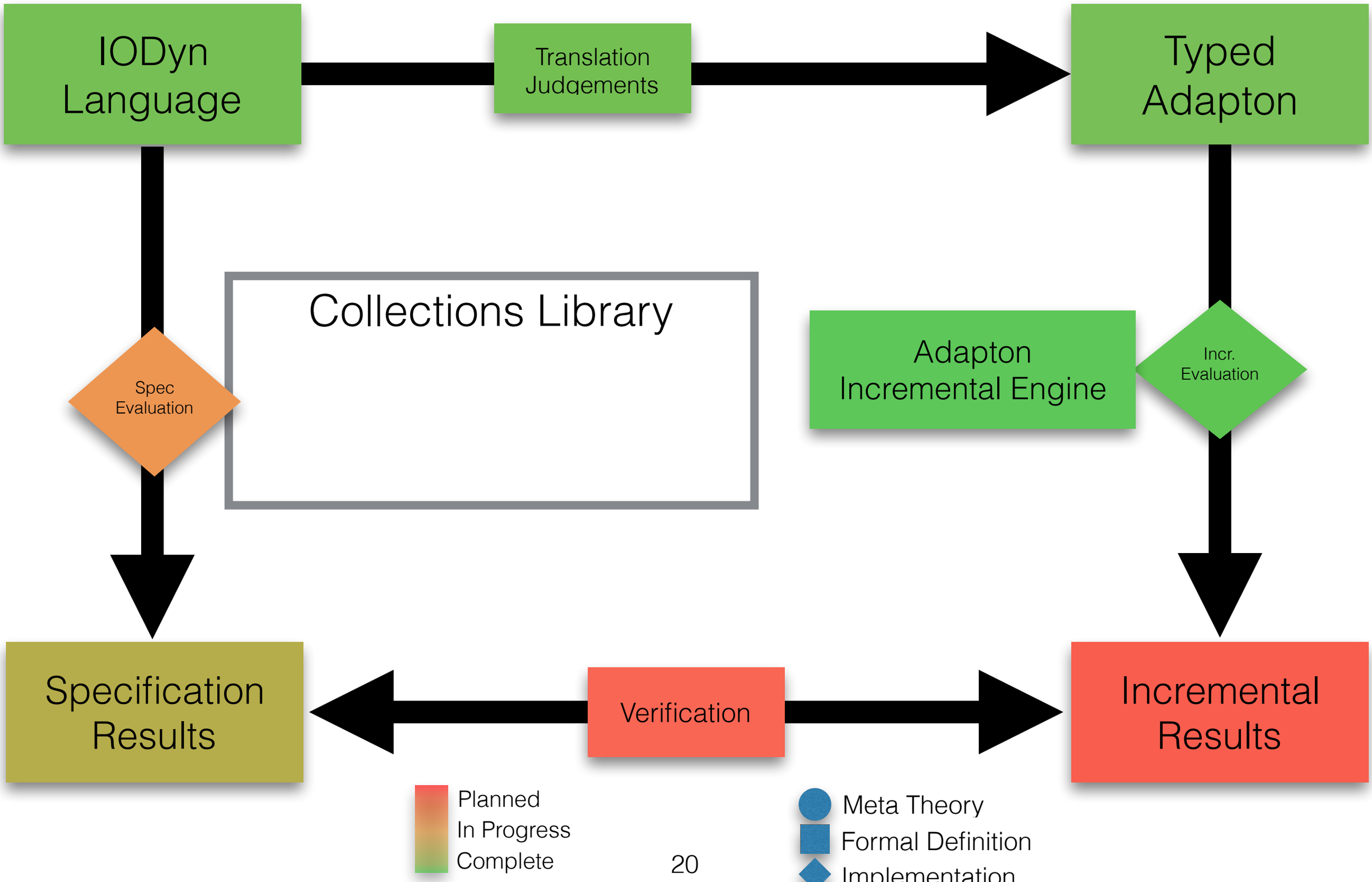
Collections primitives

Translated to Typed Adapton

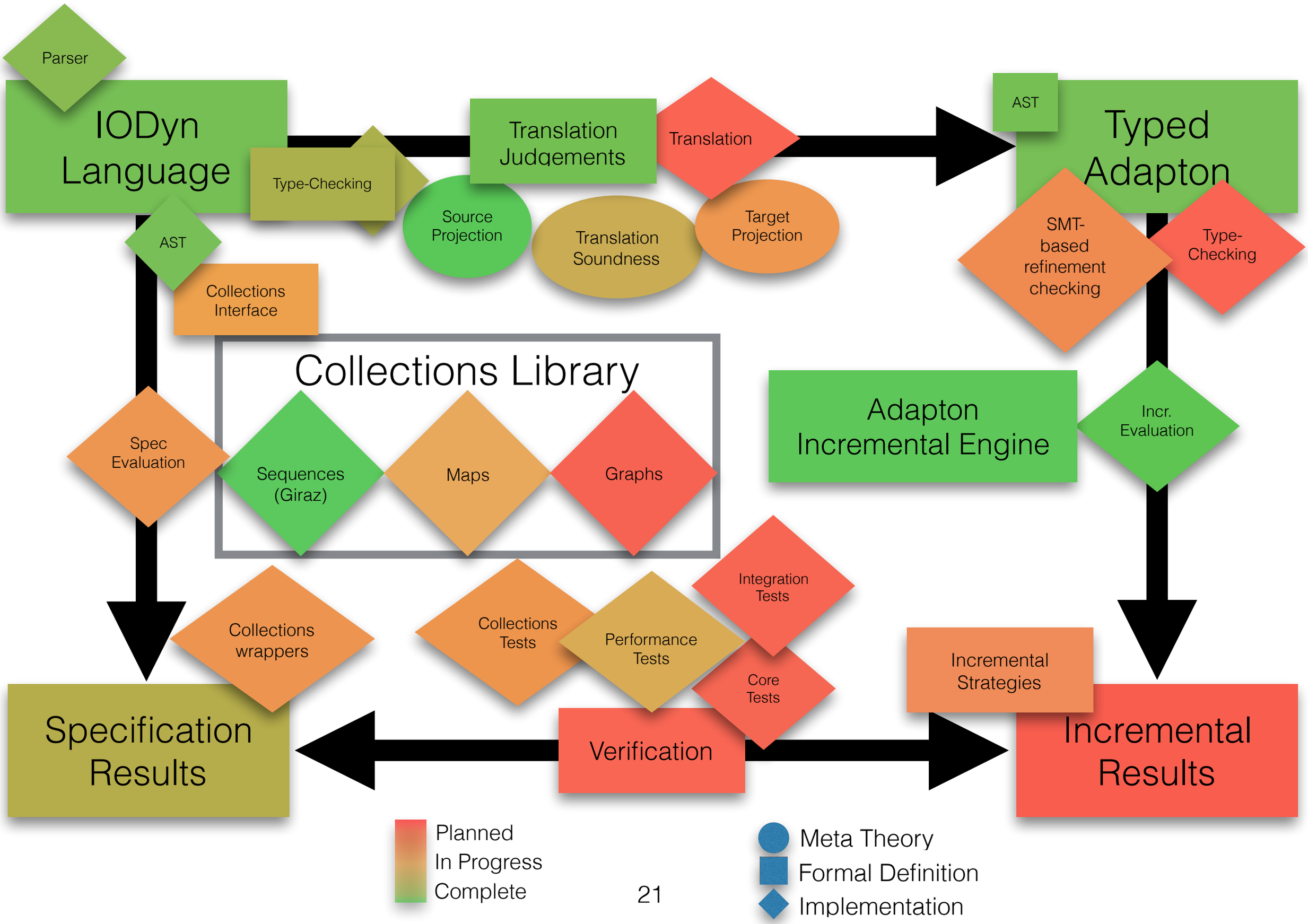
What is IODyn?



What is IODyn?

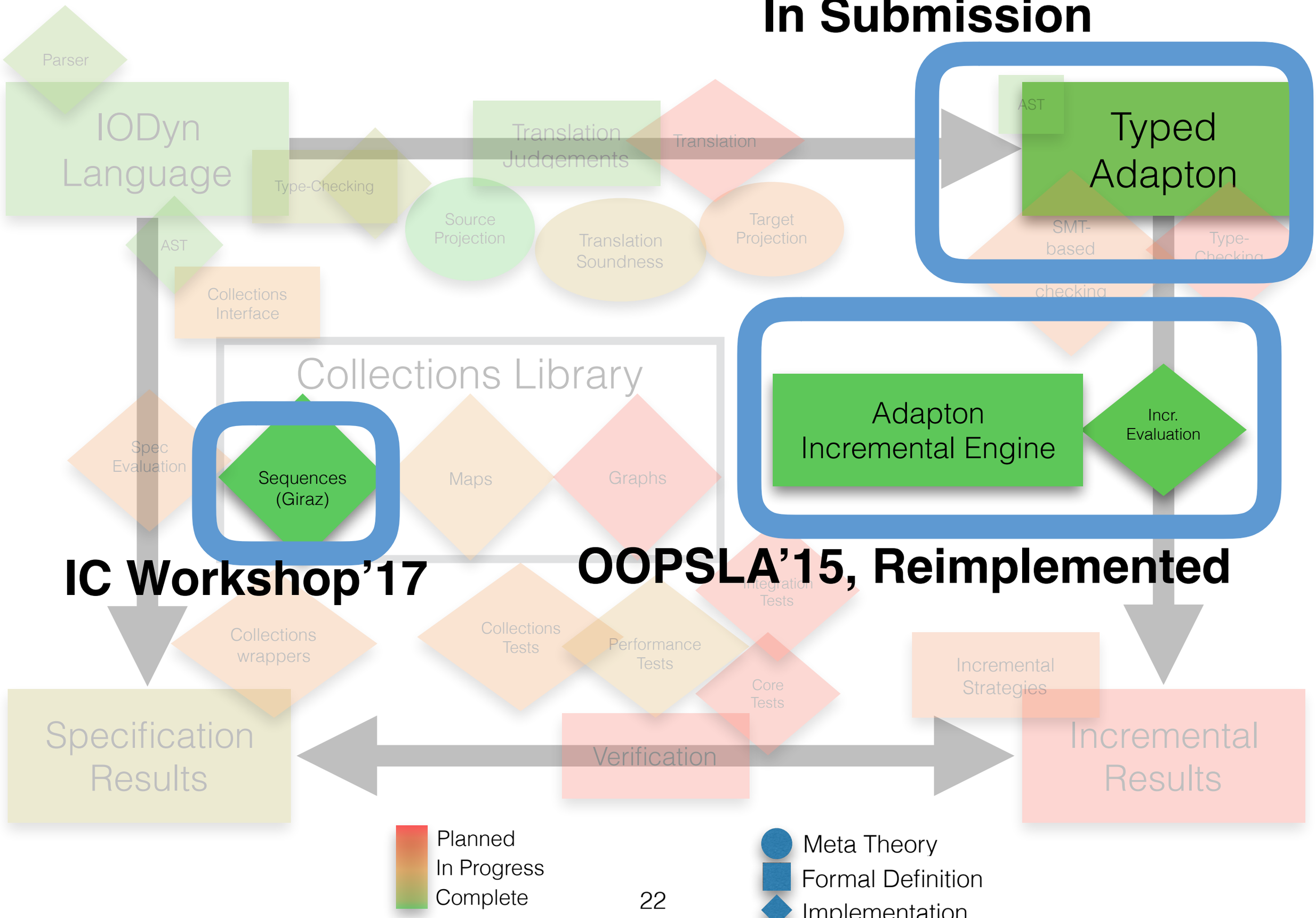


What is IODyn?



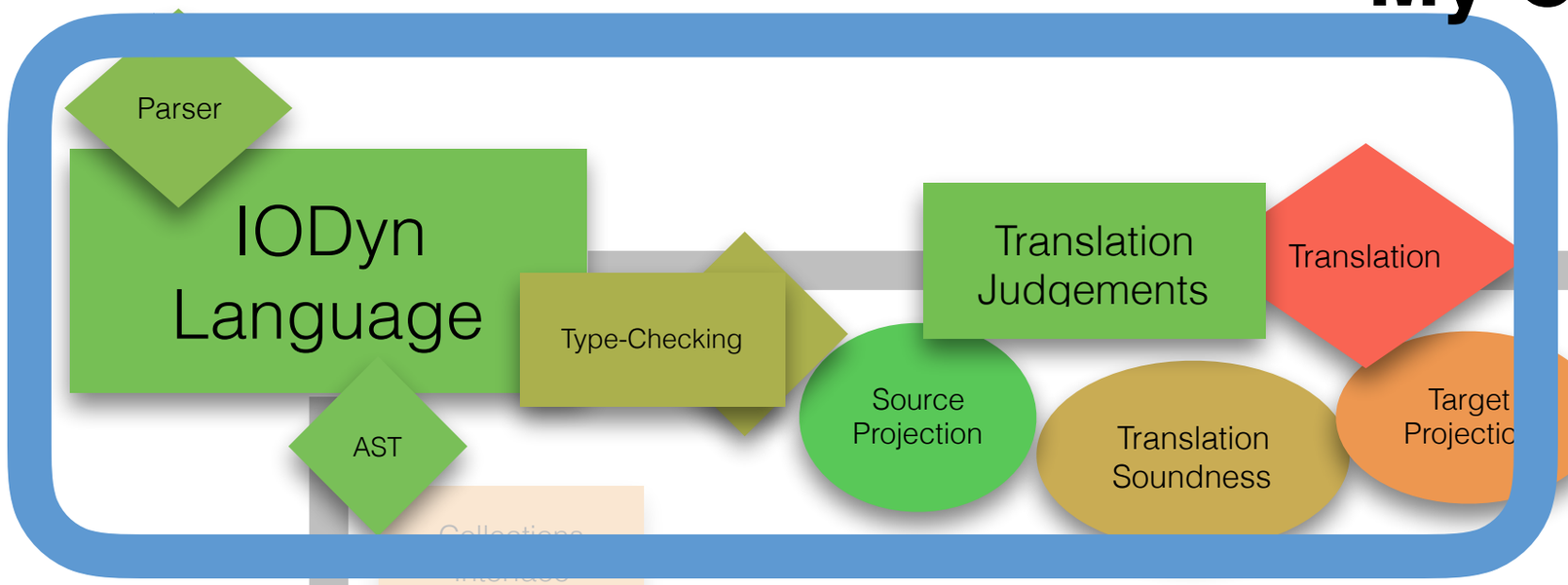
What is IODyn?

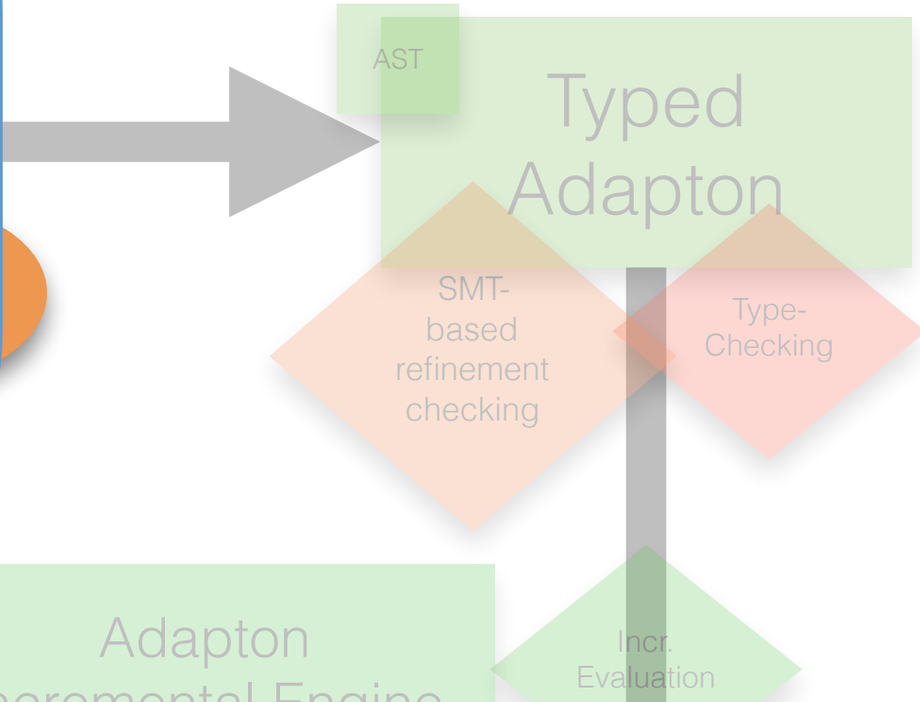
In Submission

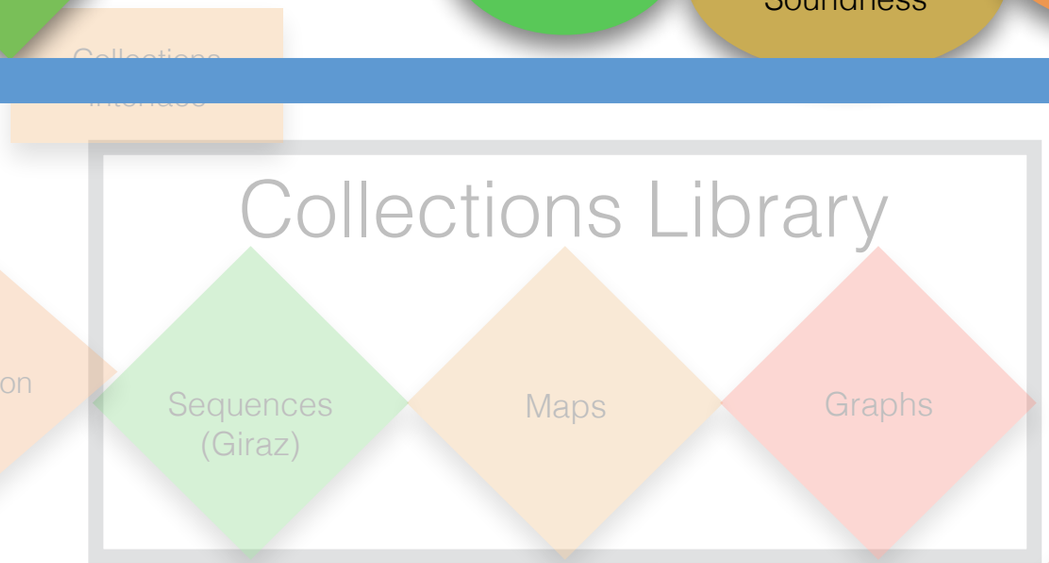


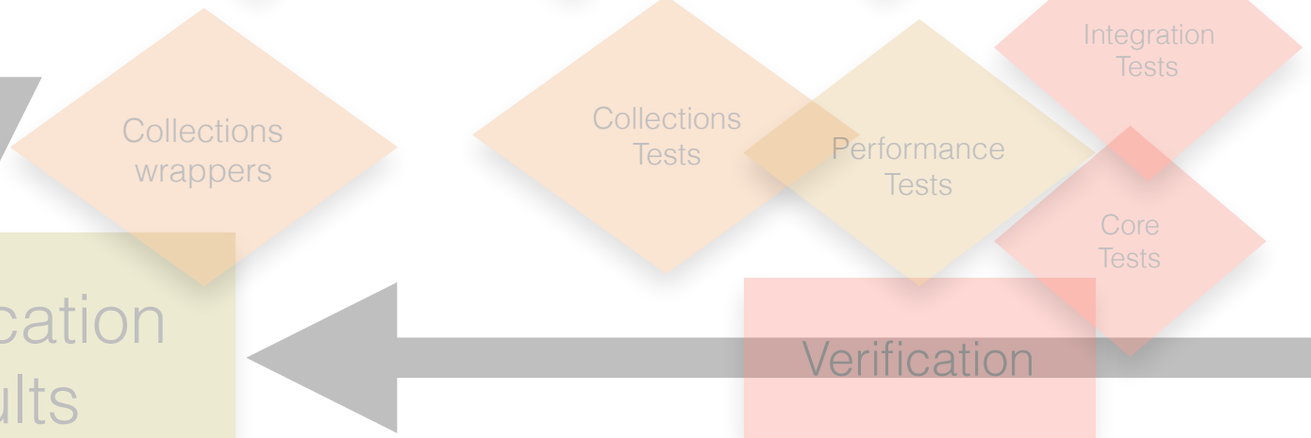
What is IODyn?

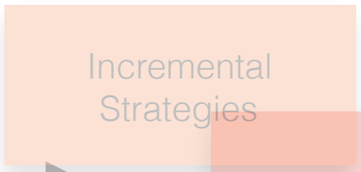
My Current Focus

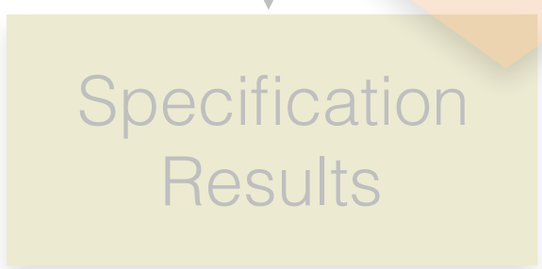


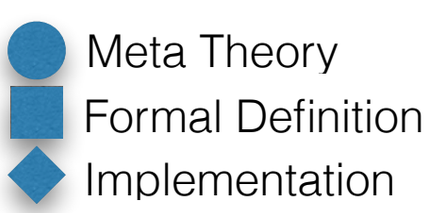






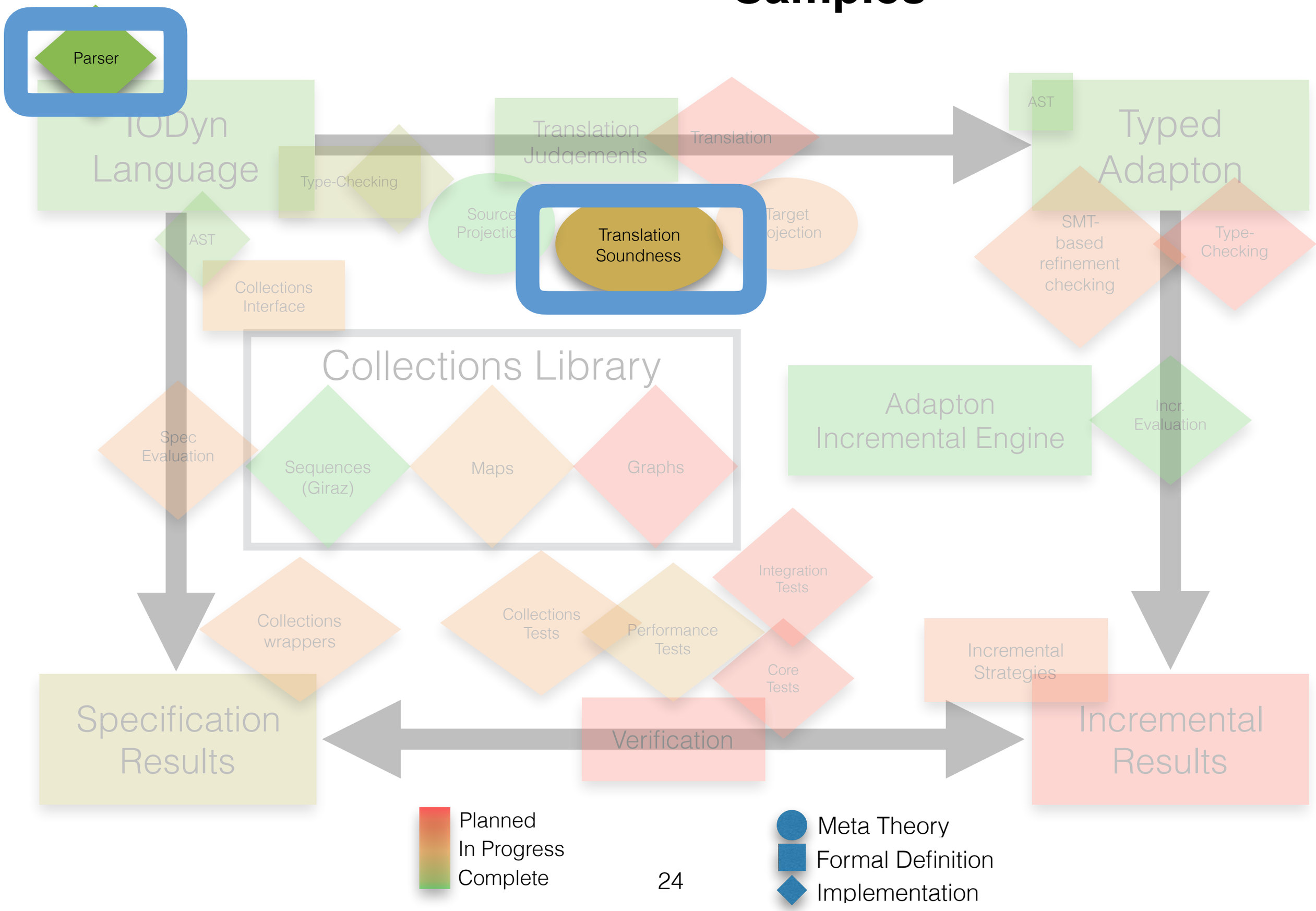






What is IODyn?

Samples



Parsing with Rust Macros

Parsing with Rust Macros

Defined by a set of rewriting rules

```
macro_rules! macro_name {
  { pattern } => { replacement };
  …
}

macro_name![ pattern ]
// replacement
```

Parsing with Rust Macros

Defined by a set of rewriting rules

```
macro_rules! macro_name {
  { literal1 literal2 } => { literal3 };
  { literal1 $variable:tt } => { $variable literal1 };
  { literal2 $($repeats:tt),+ } => $(litteral2 $repeats);+ }
}

macro_name![ literal1 something ]
// something literal1
macro_name![literal2 r, s, t]
// literal2 r; literal2 s; literal2 t
```

Parsing with Rust Macros

Sample rule

```
macro_rules! make_exp {
  // lam r.e (lambda)
  {lam $var:ident . $($body:tt)+ } =>
  {{Exp::Lam(
     stringify![$var].to_string(),
     Rc::new(make_exp![$($body)+])
   )

  …
  }};
}
```

Sample code

```
{fix qh. lam pts. lam line. lam hull.
let complete = { SeqIsEmpty(pts) }
if (complete) then { ret hull } else {

  …
}} : Seq((nat x nat)) ->
     ((nat x nat) x (nat x nat)) ->
     Seq((nat x nat)) ->
     F Seq((nat x nat))
```

Parsing with Rust Macros

```
A -> B -> C -> D
```

Not directly possible, instead,
use a fold, checking each token

Parsing with Rust Macros

```
A -> B -> C -> D
```

⇓ Fold, find `->`

```
(A)(B)(C)(D)
```

Not directly possible, instead,
use a fold, checking each token

Translation soundness

Translation soundness

if $\Gamma \vdash e : C \rightsquigarrow_x \underline{\Gamma} \vdash \underline{e} : \underline{C} \triangleright \varepsilon$

and $\Gamma \vdash \sigma : \Gamma \rightsquigarrow \underline{\Gamma} \vdash \underline{\sigma} : \underline{\Gamma}$

and $\sigma ; e \Downarrow \sigma' ; t'$

and {n is a name}

then exist {new variables}

such that

$\Gamma' \vdash t' : C \rightsquigarrow_y \underline{\Gamma'} \vdash \underline{t'} : \underline{C} \triangleright \langle\!\langle \varnothing, \varnothing \rangle\!\rangle$

and $\Gamma' \vdash \sigma' : \Gamma' \rightsquigarrow \underline{\Gamma'} \vdash \underline{\sigma'} : \underline{\Gamma'}$

and $\underline{\sigma} ; [n/x]\underline{e} \Downarrow \underline{\sigma'} ; \underline{t'}$

Translation soundness

IODyn Language

Typed Adapton

```
if Γ⊢e:C ⇝x Γ⊢e:C▷ε
and Γ⊢σ:Γ ⇝ Γ⊢σ:Γ
and σ;e ⇓ σ';t'
and {n is a name}
then exist {new variables}
such that
Γ'⊢t':C ⇝y Γ'⊢t':C▷《∅,∅》
and Γ'⊢σ':Γ' ⇝ Γ'⊢σ':Γ'
and σ;[n/x]e ⇓ σ';t'
```

Specification Results

Incremental Results

Summary

IODyn is a new incremental language that abstracts away much of the complexity of incremental code

IODyn combines the optimizations and safety of multiple previous projects

I've been working on defining and type-checking the IODyn source language

Next steps include implementing the translation and evaluating the results against the source spec

www.github.com/cuplv/iodyn-lang.rust

kyleheadley.github.io