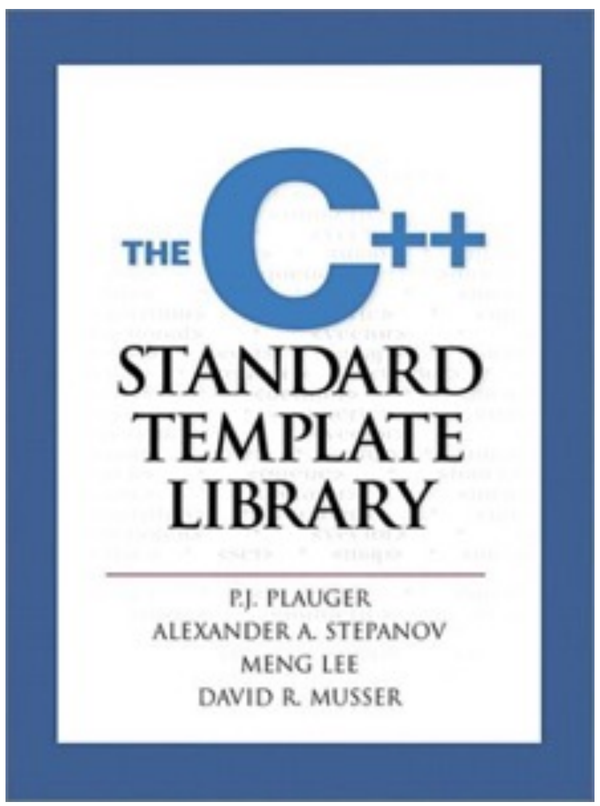
Fast Synthesis of Fast Collections

Calvin Loncaric
Emina Torlak
Michael D. Ernst

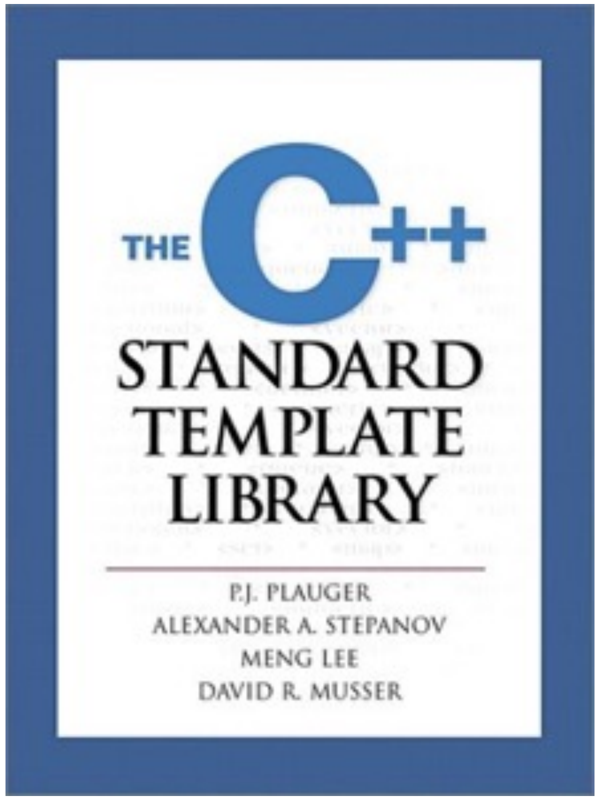
University of Washington

Data structures are everywhere

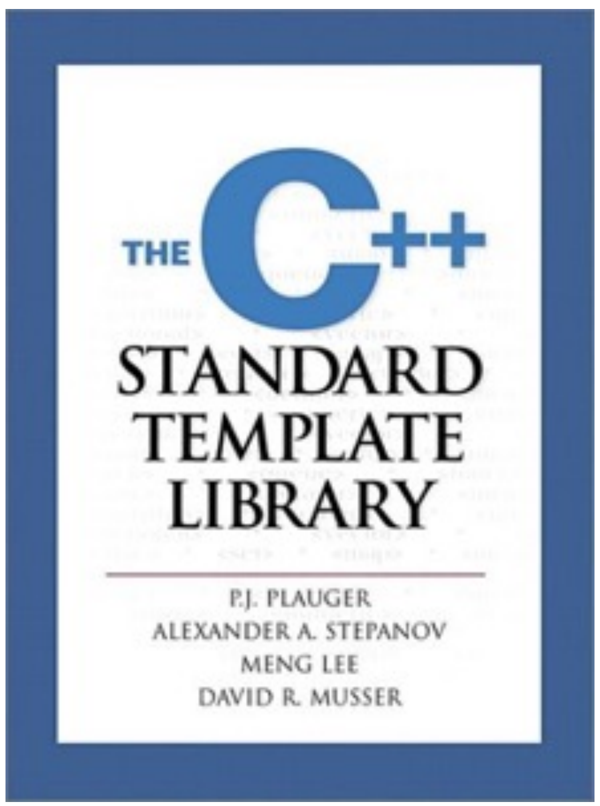
Data structures are everywhere



Data structures are everywhere



Data structures are everywhere




ORACLE Java SE Documentation

Oracle Technology Network Software Downloads

The Collections Framework

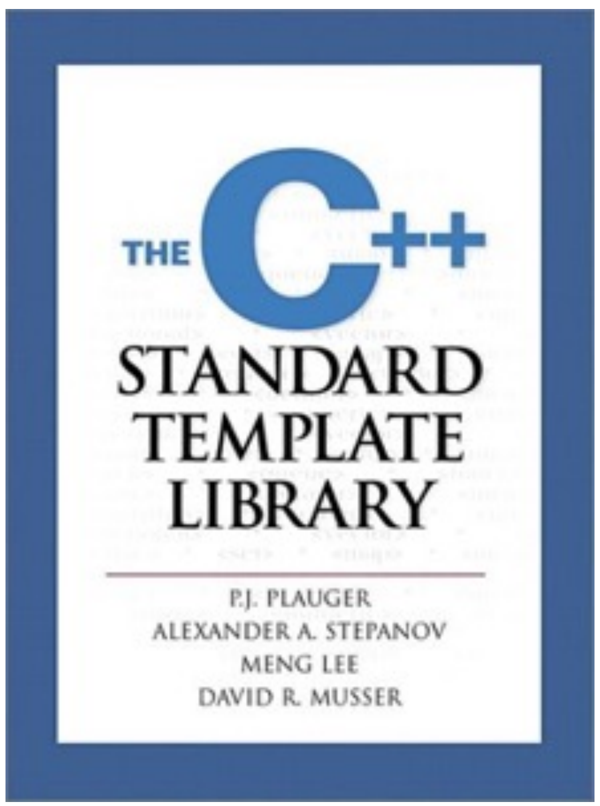
The collections framework is a unified architecture for representin
increasing performance. It enables interoperability among unrelate


8.3. collections — Container datatypes

Source code: Lib/collections/__init__.py

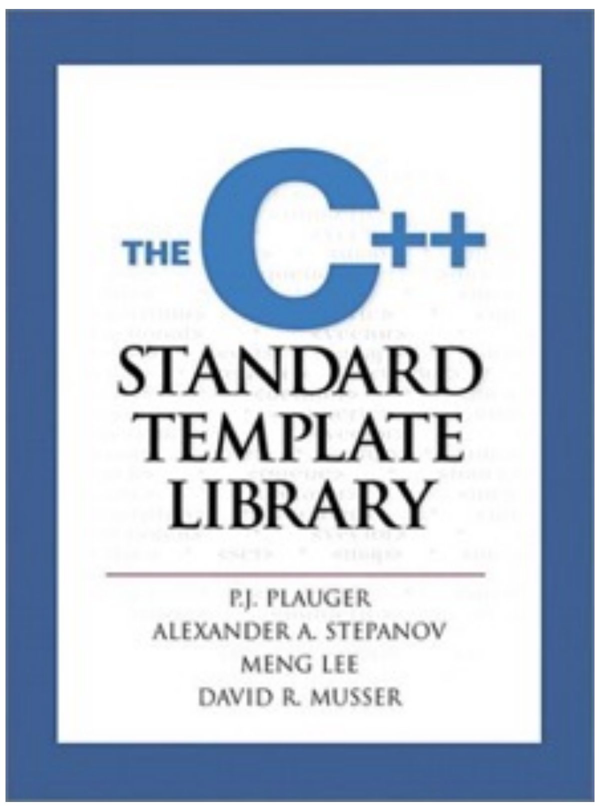
This module implements specialized container datatypes prov
to Python's general purpose built-in containers, dict, list, se

Data structures are everywhere



Lists, maps, and sets solve many problems

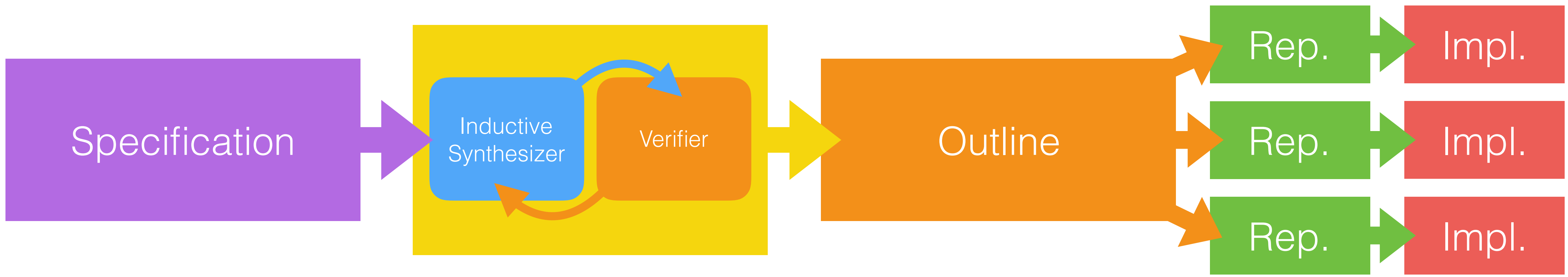
Data structures are everywhere



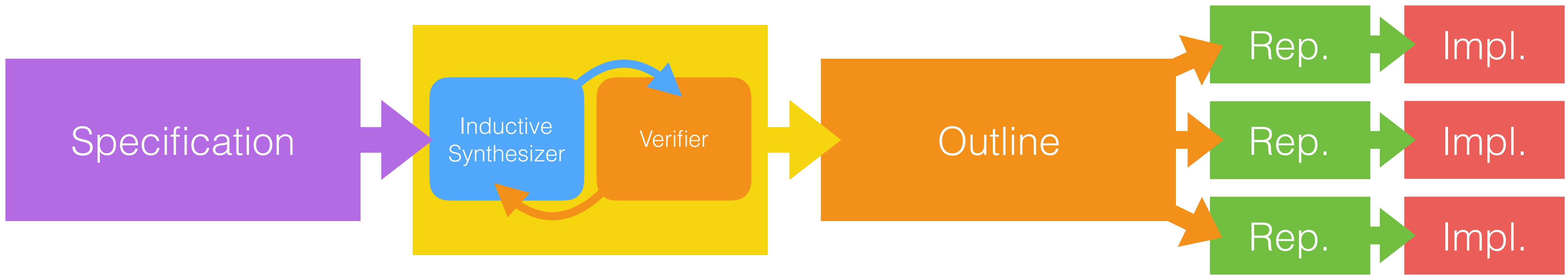
Lists, maps, and sets solve many problems

What if I need a custom data structure?

Cozy synthesizes collections

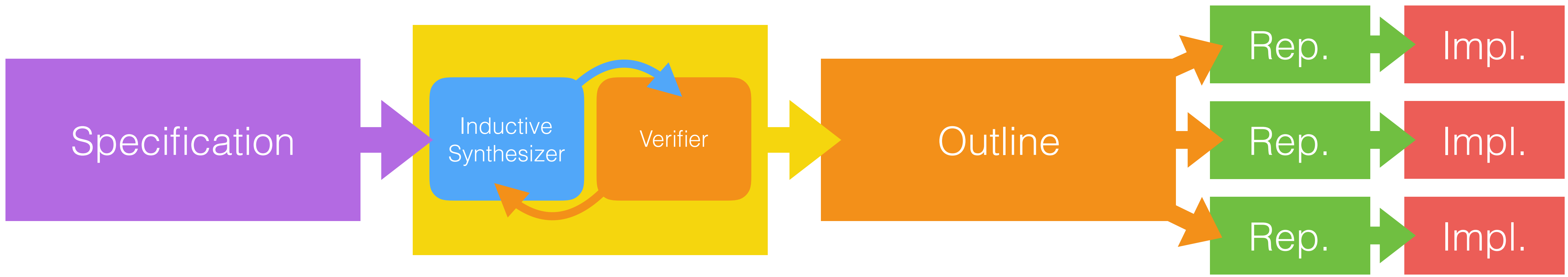


Cozy synthesizes collections



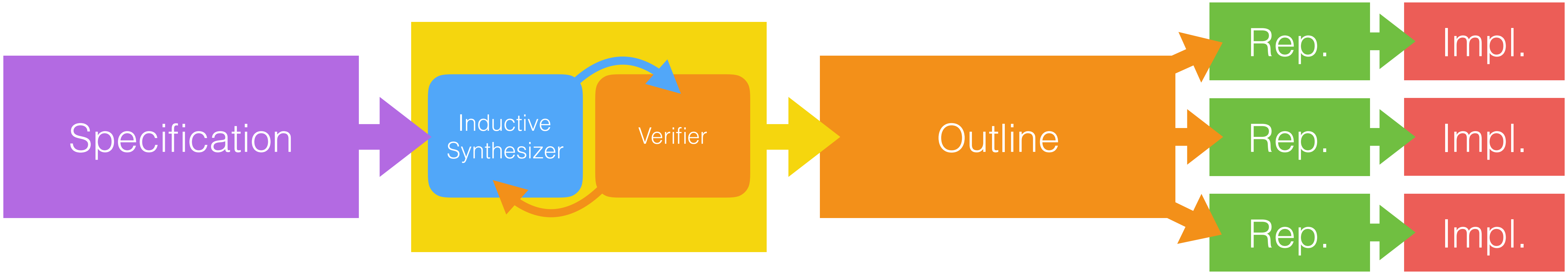
- Correct by construction

Cozy synthesizes collections



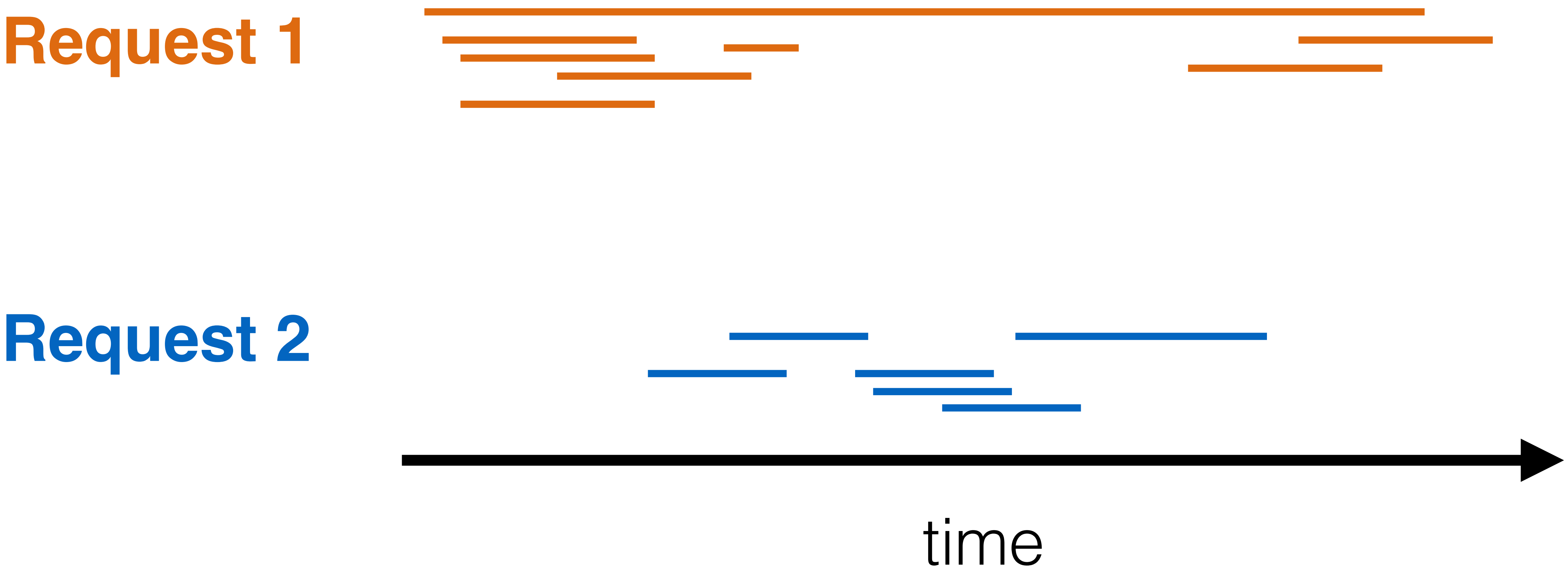
- Correct by construction
- Specifications orders-of-magnitude shorter than implementations, synthesized in < 90 seconds

Cozy synthesizes collections



- Correct by construction
- Specifications orders-of-magnitude shorter than implementations, synthesized in < 90 seconds
- Equivalent performance to human-written code

Myria Analytics Storage



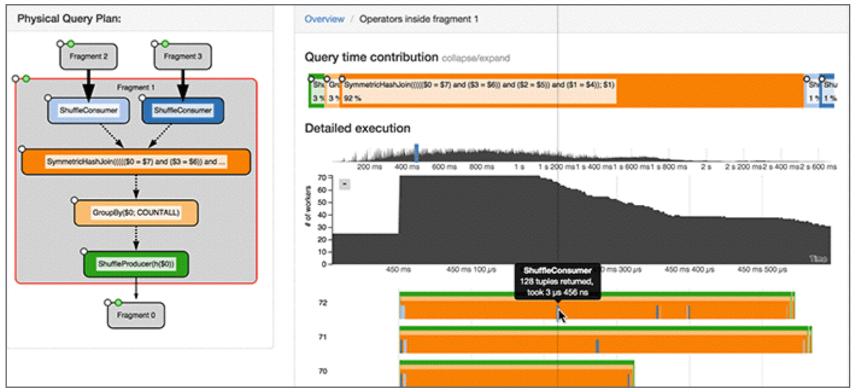
Myria Analytics Storage

Request 1

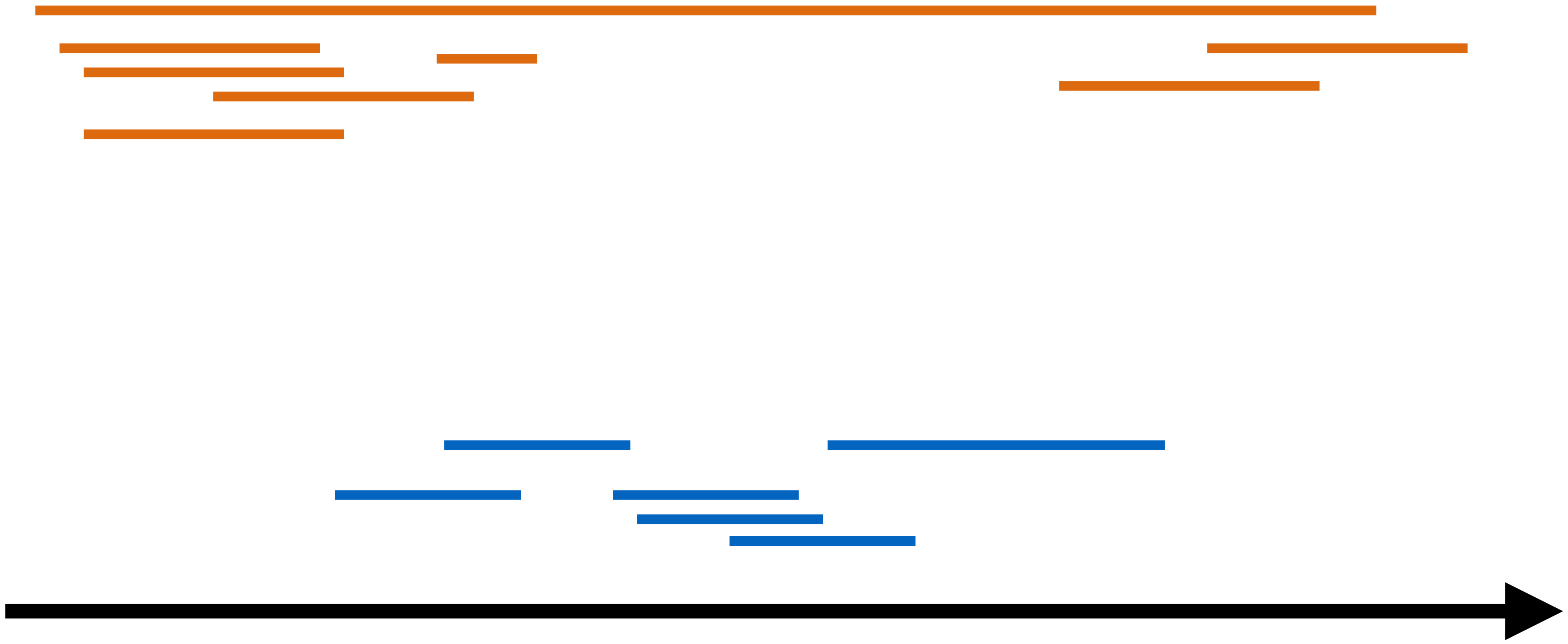
Request 2

time

Myria Analytics Storage



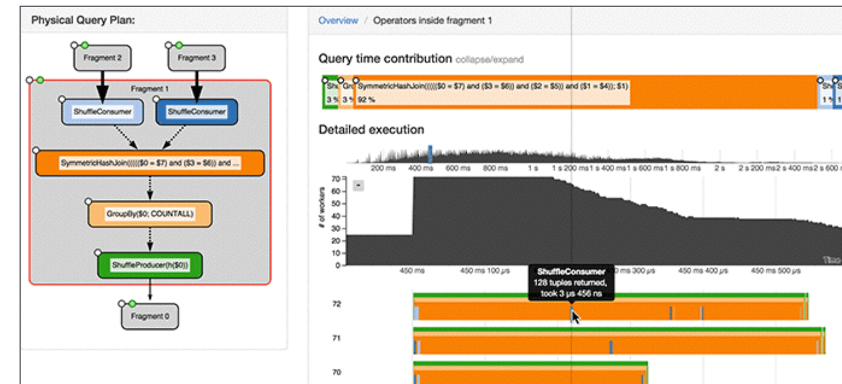
Request 1

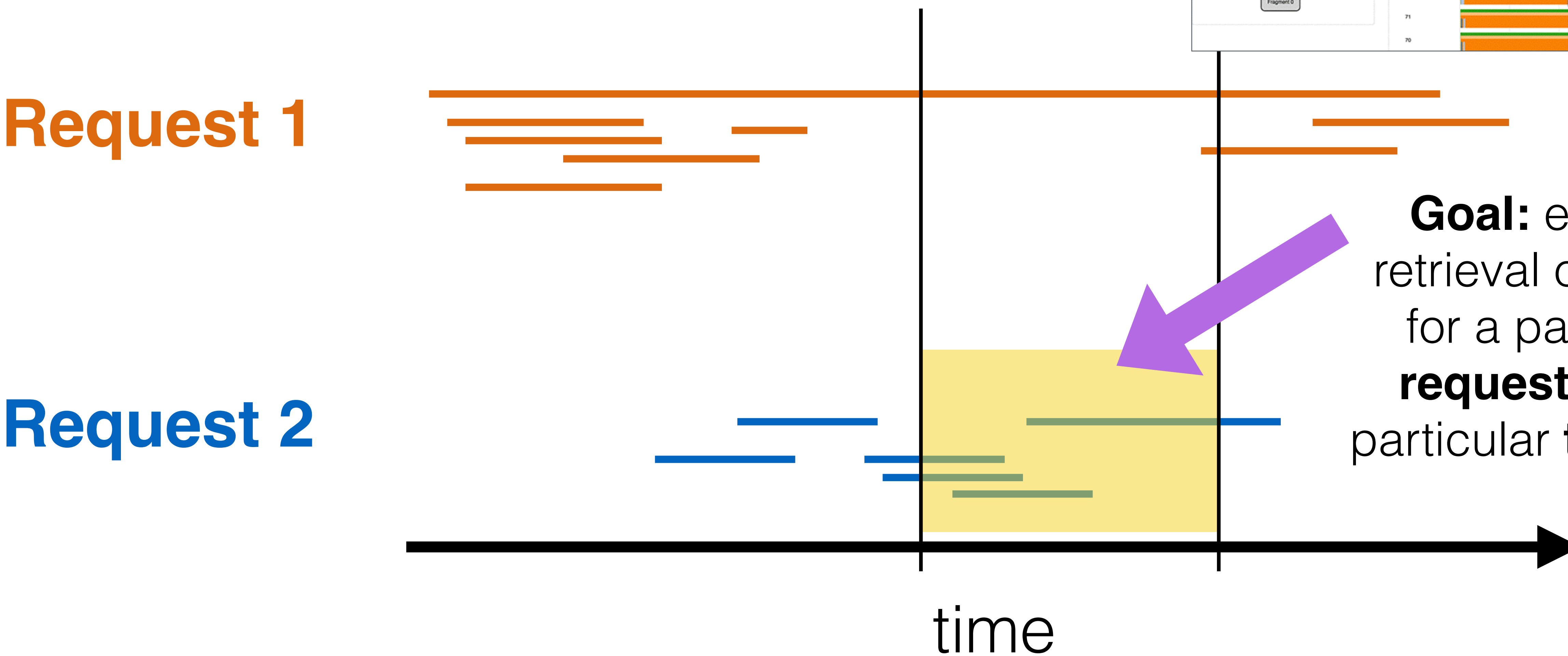


Request 2

time

Myria Analytics Storage





Goal: efficient retrieval of entries for a particular **request ID** in a particular **timespan**

Myria Analytics Storage

```
class AnalyticsLog {

    void log(Entry e)

    Iterator<Entry> getEntries(
        int   queryId,
        int   subqueryId,
        int   fragmentId,
        long  start,
        long  end)

}
```

Myria Analytics Storage

Insert an entry into
the data structure

```
class AnalyticsLog {

    void log(Entry e)

    Iterator<Entry> getEntries(
        int   queryId,
        int   subqueryId,
        int   fragmentId,
        long  start,
        long  end)

}
```

Myria Analytics Storage

Insert an entry into the data structure

Retrieve entries

```
class AnalyticsLog {

    void log(Entry e)

    Iterator<Entry> getEntries(
        int   queryId,
        int   subqueryId,
        int   fragmentId,
        long  start,
        long  end)

}
```

Myria Analytics Storage

Specification:

```
Entry has:
   queryId    : Int,
   subqueryId : Int,
   fragmentId : Int,
   start, end : Long,
   …


getEntries: all e where
   e.queryId = queryId and
   e.subqueryId = subqueryId and
   e.fragmentId = fragmentId and
   e.end >= start and
   e.start <= end
```

```
class AnalyticsLog {

     void log(Entry e)

     Iterator<Entry> getEntries(
          int    queryId,
          int    subqueryId,
          int    fragmentId,
          long   start,
          long   end)

}
```

Cozy synthesizes collections

Specification:

```
Entry has:
    field1 : Type1,
    field2 : Type2,
    …



retrieveA: all e where
    condition

retrieveB: all e where
    condition
```


```
class Structure {

    void add(Entry e)
    void remove(Entry e)
    void update(Entry e, …)

    Iterator<Entry> retrieveA(…)
    Iterator<Entry> retrieveB(…)

}
```

Trivial Solution

```
retrieve: all e where
    P(e, input)
```

Trivial Solution

retrieve: all e where
    P(e, input)

```
List<Entry> data;

Iterator<Entry> retrieve(input) {
    for e in data:
        if P(e, input):
            yield e
}
```

Trivial Solution

```
retrieve: all e where
    P(e, input)
```

```
List<Entry> data;

Iterator<Entry> re
    for e in data:
        if P(e, input):
            yield e
}
```

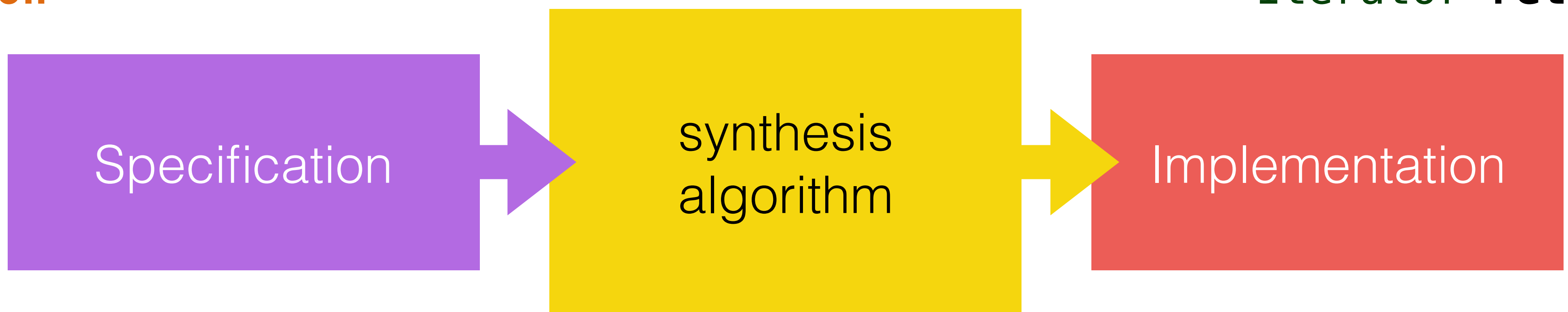
There has to be a better way!

```
Entry has:
   field1, field2, …
retrieveA: all e where
   condition
retrieveB: all e where
   condition
```

Intractable

```
void add(Entry e)
void remove(Entry e)
void update(Entry e, …)

Iterator retrieveA(…)
Iterator retrieveB(…)
```

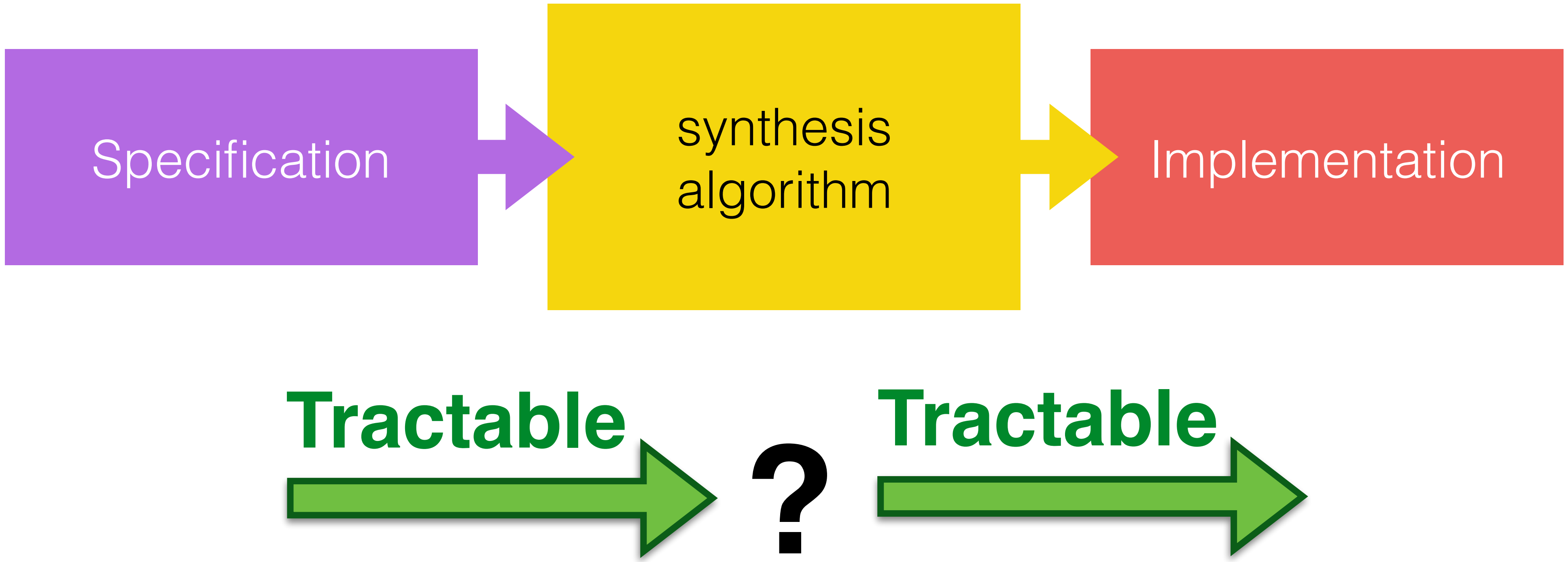


In the quest for a good solution, the search space of “all possible programs” is simply too large

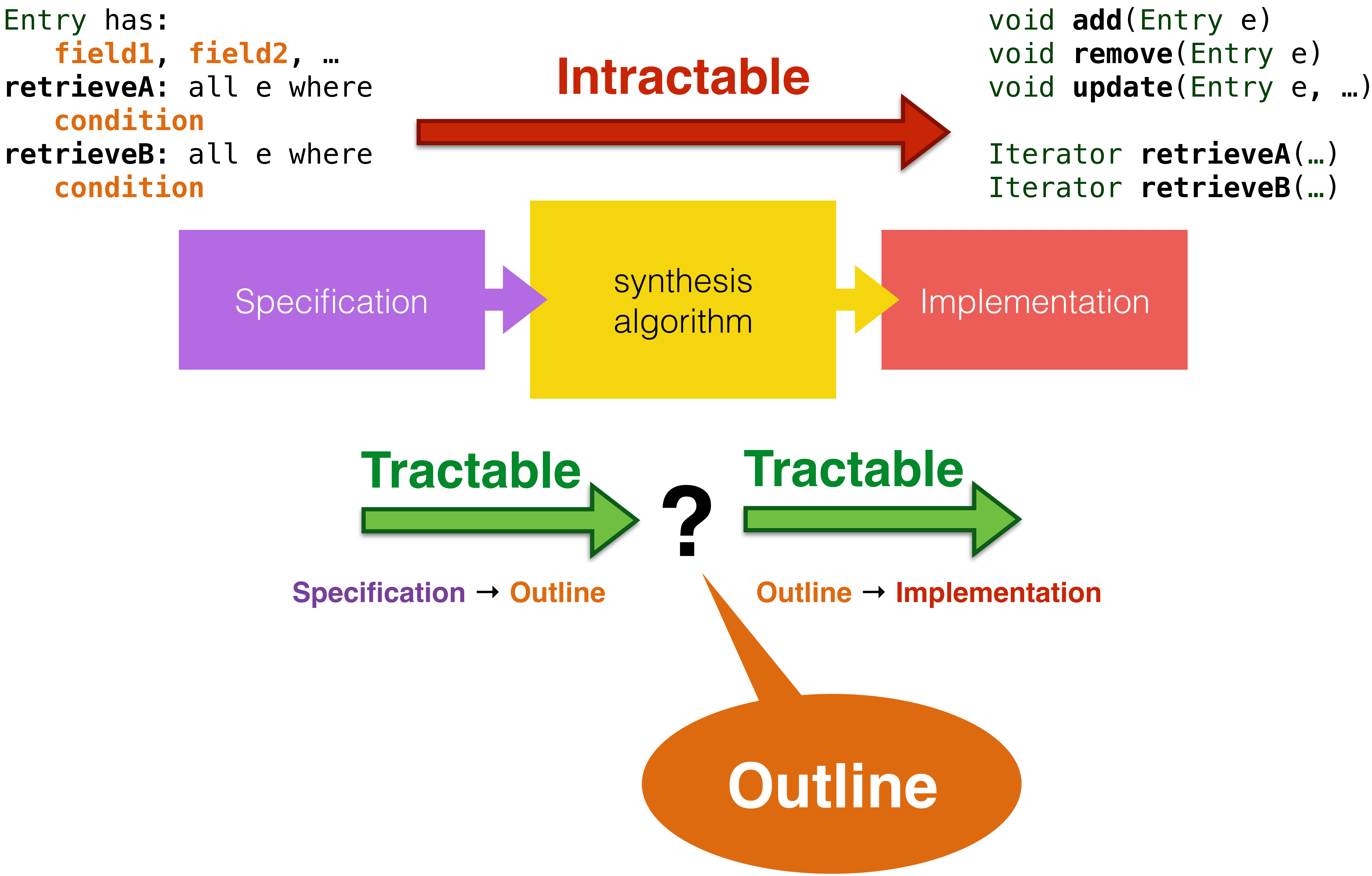
```
Entry has:
   field1, field2, …
retrieveA: all e where
   condition
retrieveB: all e where
   condition
```


```
void add(Entry e)
void remove(Entry e)
void update(Entry e, …)

Iterator retrieveA(…)
Iterator retrieveB(…)
```



In the quest for a good solution, the search space of “all possible programs” is simply too large



Entry has:
field1, field2, …
retrieveA: all e where
condition
retrieveB: all e where
condition
Intractable
void add(Entry e)
void remove(Entry e)
void update(Entry e, …)
Iterator retrieveA(…)
Iterator retrieveB(…)
Specification
synthesis algorithm
Implementation
Tractable
?
Tractable
Specification → Outline
Outline → Implementation
Outline

```
Entry has:
   field1, field2, …
retrieveA: all e where
   condition
retrieveB: all e where
   condition
```

Intractable

```
void add(Entry e)
void remove(Entry e)
void update(Entry e, …)

Iterator retrieveA(…)
Iterator retrieveB(…)
```

Specification → synthesis algorithm → Implementation

Tractable ? **Tractable**

Specification → Outline   **Outline → Implementation**

specific enough to describe
asymptotic performance

general enough to encode
a data structure succinctly

Outline

Outlines

Plans for retrieving entries

Outlines

Plans for retrieving entries

- **All** ( )

Outlines

Plans for retrieving entries

- **All** ( )
- **HashLookup** (outline, field = var)

Outlines

Plans for retrieving entries

- **All** ( )
- **HashLookup** (outline, field = var)
- **BinarySearch** (outline, field > var)

Outlines

Plans for retrieving entries

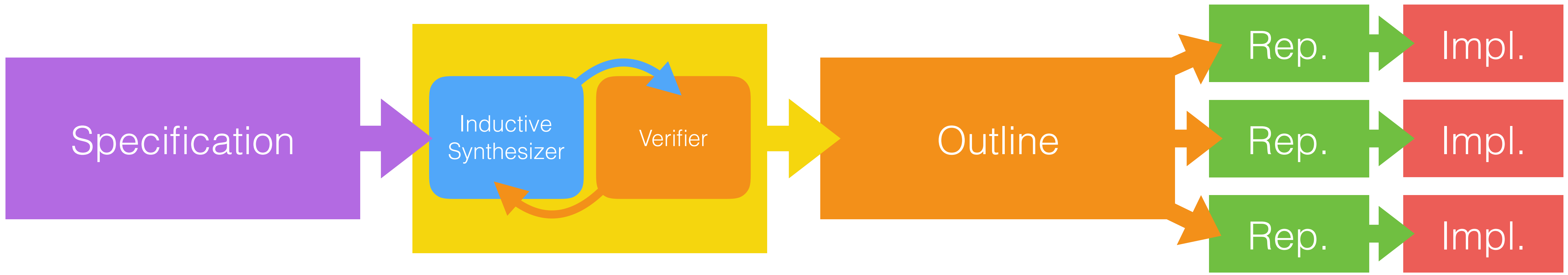
- **All** ( )
- **HashLookup** (outline, field = var)
- **BinarySearch** (outline, field > var)
- **Concat** (outline_1, outline_2)

Outlines

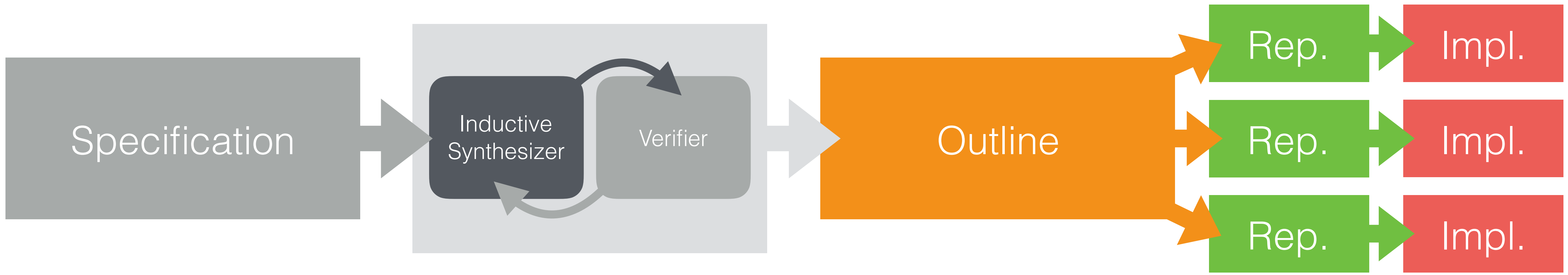
Plans for retrieving entries

- **All** ( )
- **HashLookup** (outline, field = var)
- **BinarySearch** (outline, field > var)
- **Concat** ($outline_1$, $outline_2$)
- **Filter** (outline, predicate)

Outlines → Implementations



Outlines → Implementations



Outlines → Implementations

HashLookup (
  All(),
  e.queryId = q)

```
class Structure {

    Iterator<Entry> retrieve(q) { … }

}
```

Outlines → Implementations

HashLookup (
  $\boxed{\textbf{All}(),}$
  e.queryId = q)

```
class Structure {

    Iterator<Entry>
    retrieve(q) { … }

}
```

Outlines → Implementations

HashLookup (
$\boxed{\textbf{All}(),}$
e.queryId = q)

```
class Structure {

    T data;

    Iterator<Entry>
    retrieve(q) { … }

}
```

Outlines → Implementations

HashLookup (
$\boxed{\texttt{data,}}$
e.queryId = q)

```
class Structure {

    T data;

    Iterator<Entry>
    retrieve(q) { … }

}
```

Outlines → Implementations

HashLookup (
  data,
  e.queryId = q)

```
class Structure {
    T data;
    Iterator<Entry>
    retrieve(q) { … }
}
```

Outlines → Implementations

HashLookup (
data,
e.queryId = q)

```
class Structure {
    T data;
    Iterator<Entry> retrieve(q) { … }
}
```

Outlines → Implementations

```
HashLookup (
    data,
    e.queryId = q )
```

```
class Structure {

    HMap<K,V> data;

    Iterator<Entry>
    retrieve(q) { … }

}
```

Outlines → Implementations

HashLookup (
  data,
  e.queryId = q)

```
class Structure {
    HMap<K,V> data;
    Iterator<Entry>
    retrieve(q) { … }
}
```

Outlines → Implementations

HashLookup (
  data,
  $\boxed{\text{e.queryId = q}}$)

```
class Structure {
    HMap<K,V> data;
    Iterator<Entry>
    retrieve(q) { … }
}
```

Outlines → Implementations

HashLookup (
  `data`,
  $\boxed{\text{e.queryId} = \text{q}}$)

```
class Structure {

    HMap<int,V> data;

    Iterator<Entry>
    retrieve(q) { … }

}
```

Outlines → Implementations

HashLookup (
 data,
 e.queryId = q)

```
class Structure {
    HMap<int,V> data;
    Iterator<Entry> retrieve(q) { … }
}
```

Outlines → Implementations

HashLookup (
  data,
  e.queryId = q)

```
class Structure {
    HMap<int,V> data;
    Iterator<Entry>
    retrieve(q) { … }
}
```

Outlines → Implementations

HashLookup (
  data,
  e.queryId = q)

```
class Structure {
    HMap<int,V> data;
    Iterator<Entry>
    retrieve(q) { … }
}
```

```
V = ArrayList<Entry>
```

Outlines → Implementations

HashLookup (
  data,
  e.queryId = q)

```
class Structure {
    HMap<int,V> data;
    Iterator<Entry>
    retrieve(q) { … }
}
```

`V = ArrayList<Entry>`   `V = LinkedList<Entry>`

Outlines → Implementations

HashLookup (
  data,
  e.queryId = q)

```
class Structure {
    HMap<int,V> data;
    Iterator<Entry>
    retrieve(q) { … }
}
```

Outlines → Implementations

HashLookup (
  data,
  e.queryId = q)

```
class Structure {

    HMap<int,V> data;

    Iterator<Entry>
    retrieve(q)
    {
      v = data.get(q);
      return v.iterator();
    }
```

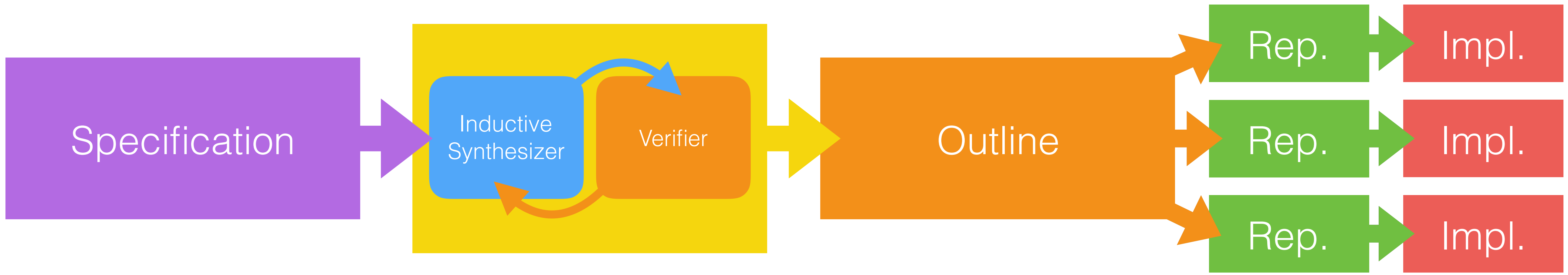
Outlines → Implementations

add, remove, update

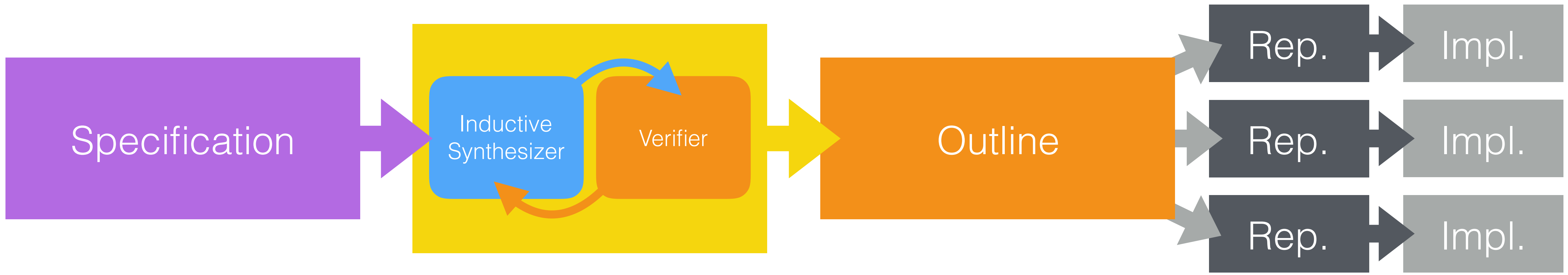
HashLookup (
 data,
 e.queryId = q)

```
class Structur
  HMap<int,V> data;
  Iterator<Entry>
  retrieve(q)
  {
    v = data.get(q);
    return v.iterator();
  }
```

Specification → Outline

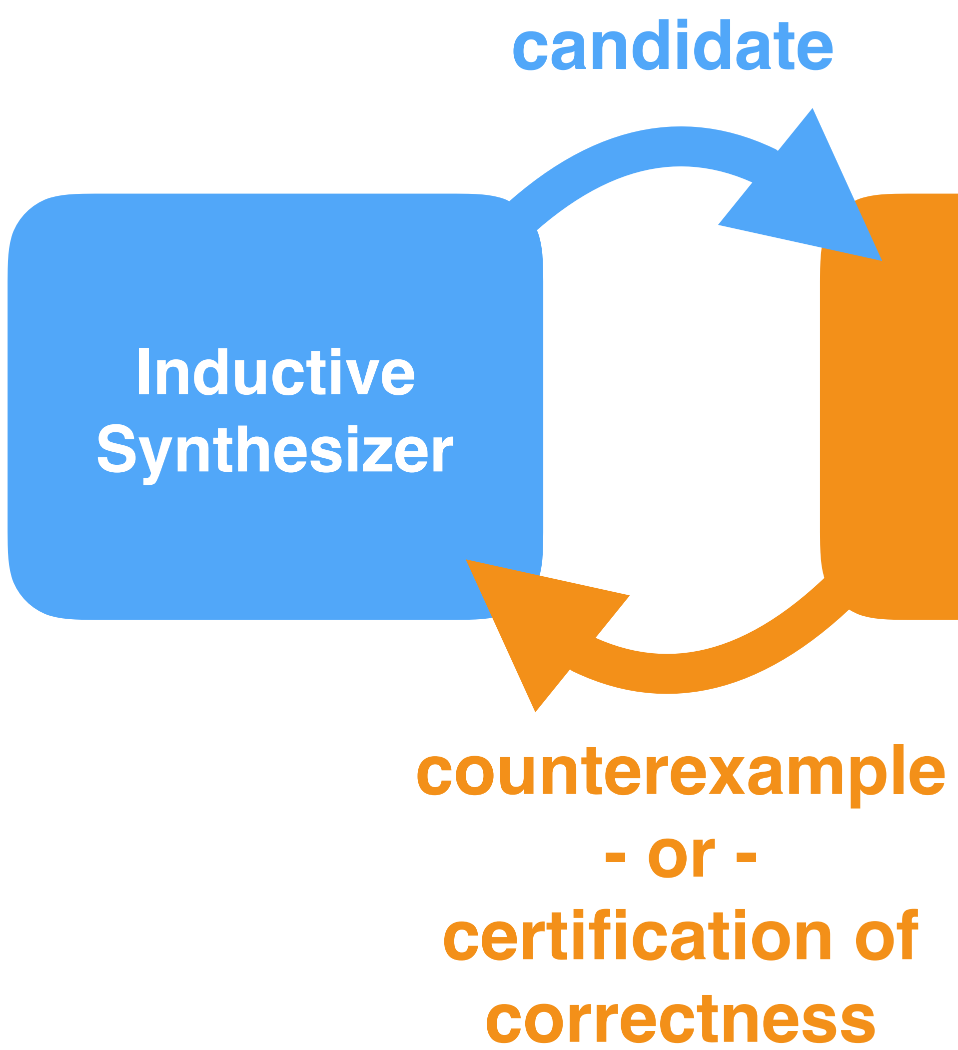


Specification → Outline



Specification → Outline

CEGIS

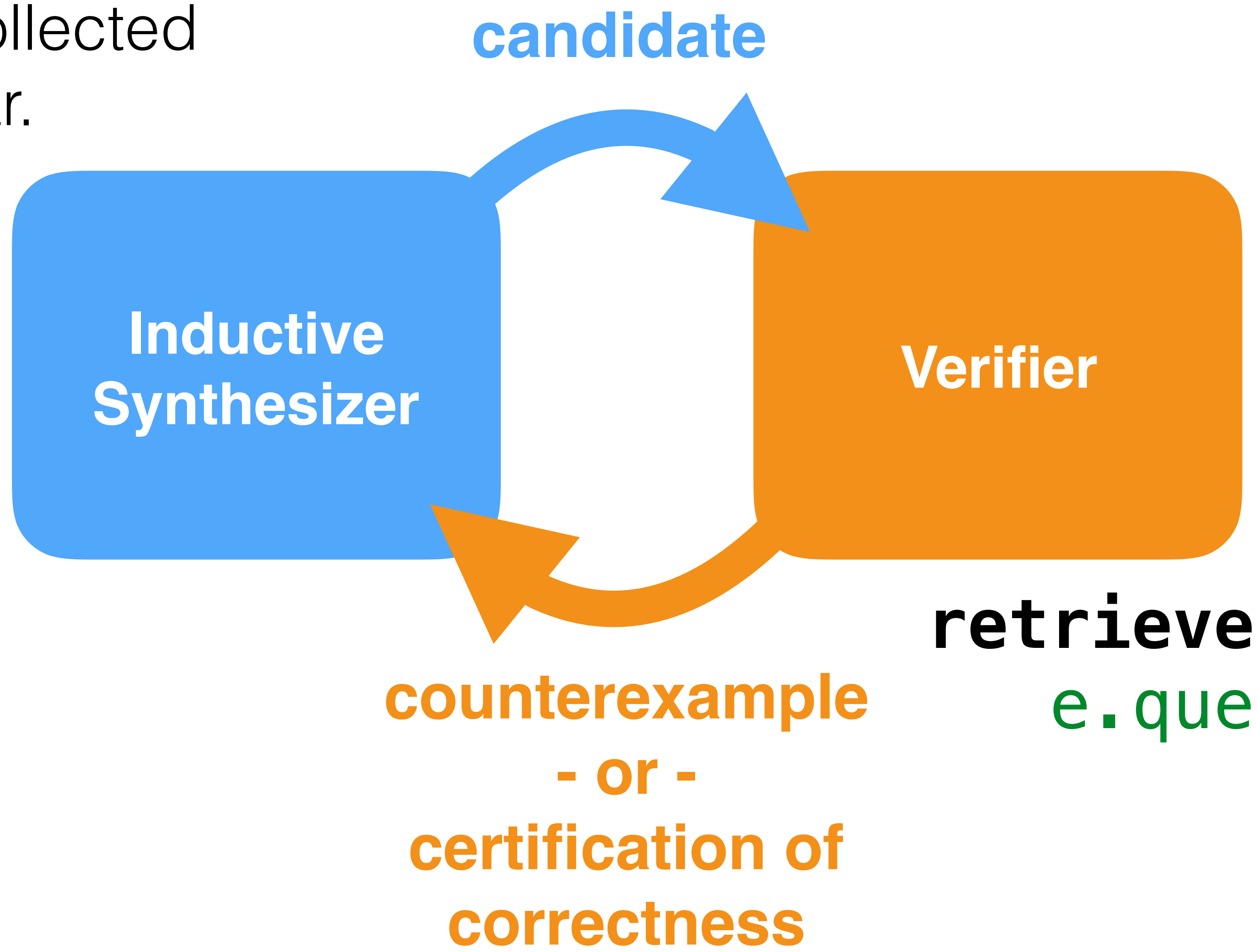



```
retrieve: all e where
   e.queryId = q and …
```

Specification → Outline

Remembers all examples; only reasons about examples collected thus far.

CEGIS




```
retrieve: all e where
    e.queryId = q and …
```

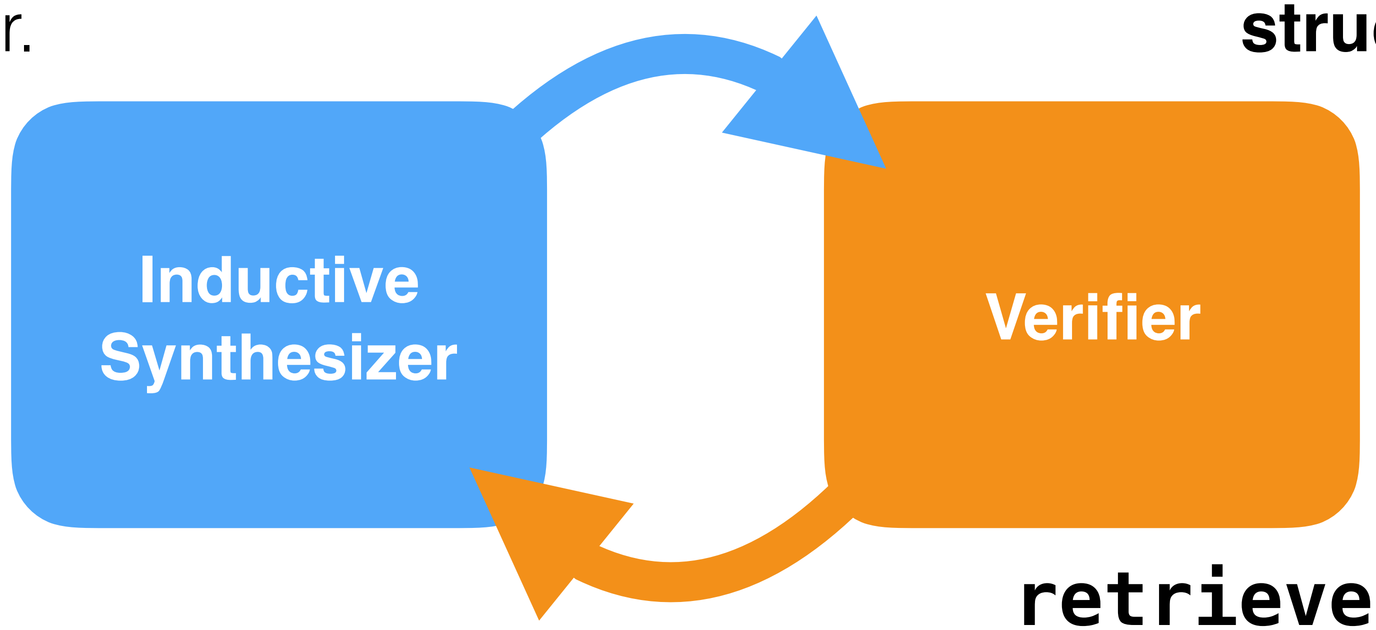
Specification → Outline

Remembers all examples; only reasons about examples collected thus far.

CEGIS

Must ensure the outline is correct for **all possible inputs** and **all possible data structure states.**

candidate



counterexample
- or -
certification of correctness

```
retrieve: all e where
    e.queryId = q and …
```

$$\forall I\ \forall S,\ out = \{\ e \mid e \in S \land P(I, e)\ \}$$

Cost Model

Filter (
    All(),
    e.queryId = q)

HashLookup (
    All(),
    e.queryId = q)

Cost Model

$O(1)$

Filter (
All(),
e.queryId = q)

HashLookup (
All(),
e.queryId = q)

Cost Model

$O(n)$

$O(1)$

Filter (
    All(),
    e.queryId = q)

HashLookup (
    All(),
    e.queryId = q)

Cost Model

$O(n)$

$O(1)$

Filter (
  All(),
  e.queryId = q)

$O(1)$

HashLookup (
  All(),
  e.queryId = q)

Cost Model

$O(n)$

$O(1)$

Filter (
  All(),
  e.queryId = q)

$O(1)$

$O(1)$

HashLookup (
  All(),
  e.queryId = q)

Cost Model

$O(n)$

$O(1)$

Filter (
  All(),
  e.queryId = q)

>

$O(1)$

$O(1)$

HashLookup (
  All(),
  e.queryId = q)

Cost Model

$O(n)$ $O(1)$

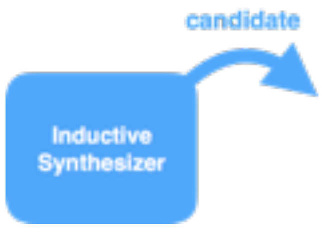
Filter (
 All(),
 e.queryId = q)

>

$O(1)$ $O(1)$

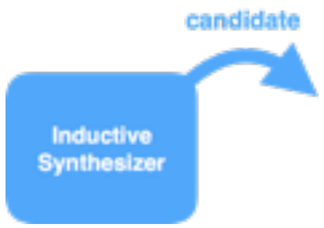
HashLookup (
 All(),
 e.queryId = q)

Cozy prefers outlines with lower cost



Inductive Synthesis

Enumerative search

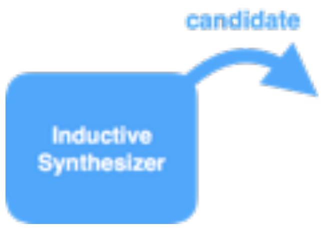


Inductive Synthesis

Enumerative search

size 1

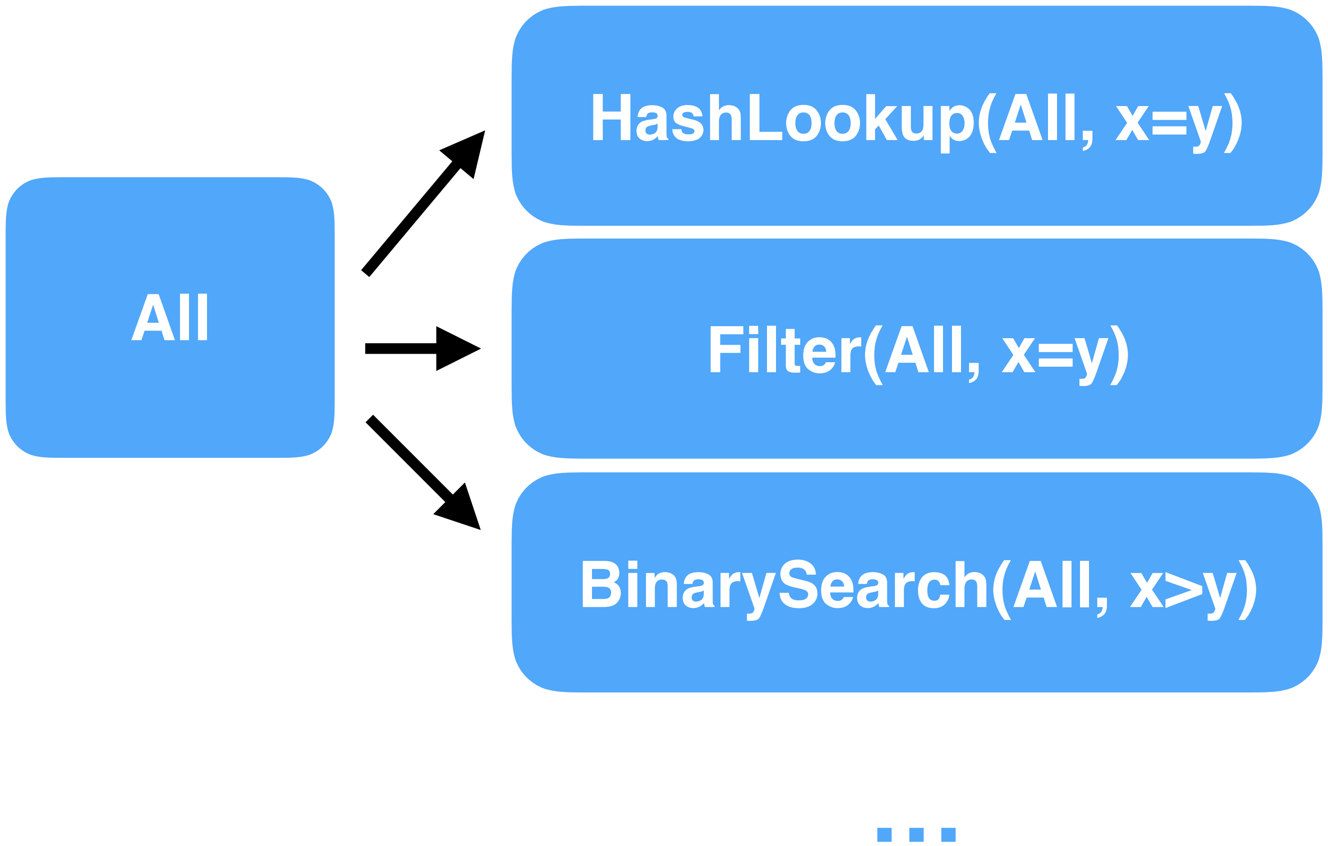
All

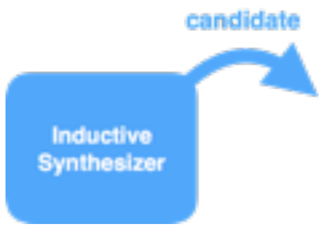


Inductive Synthesis

Enumerative search

size 1 size 2

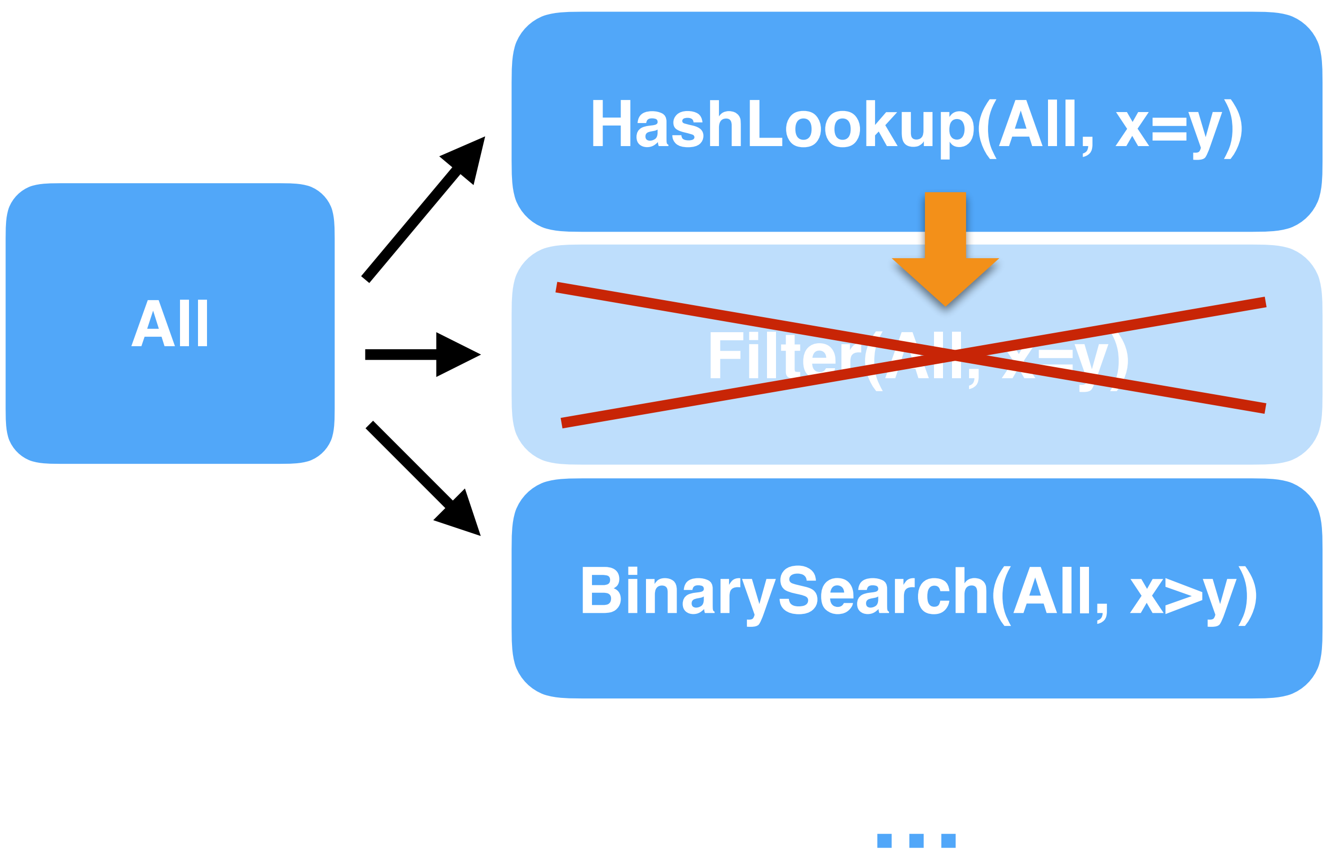


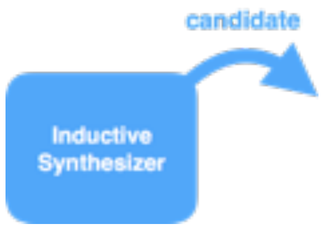


Inductive Synthesis

Enumerative search

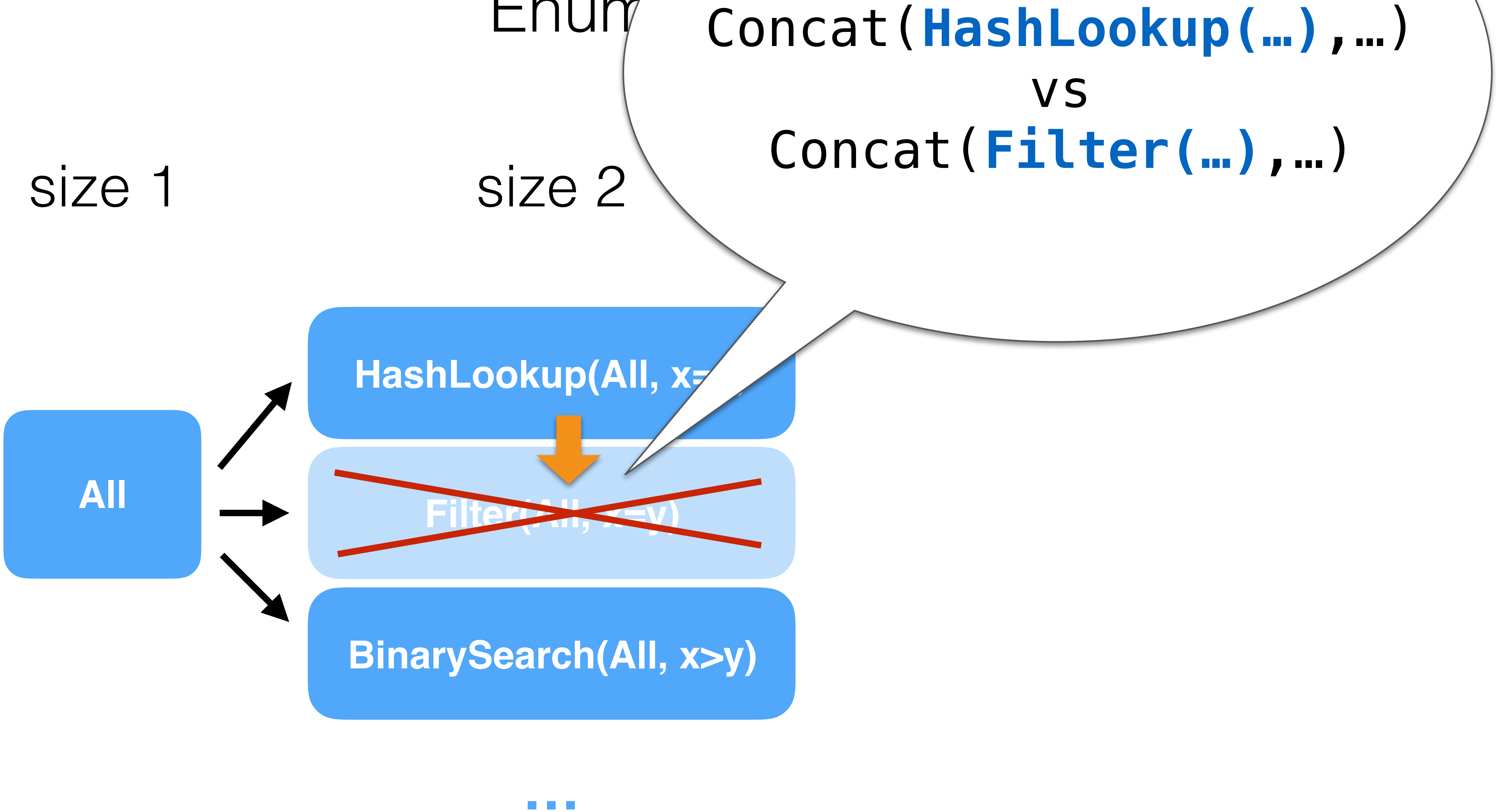
size 1 size 2

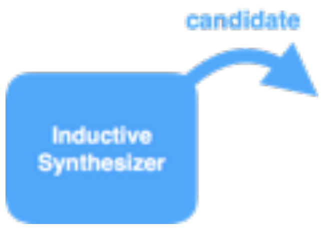




Inductive

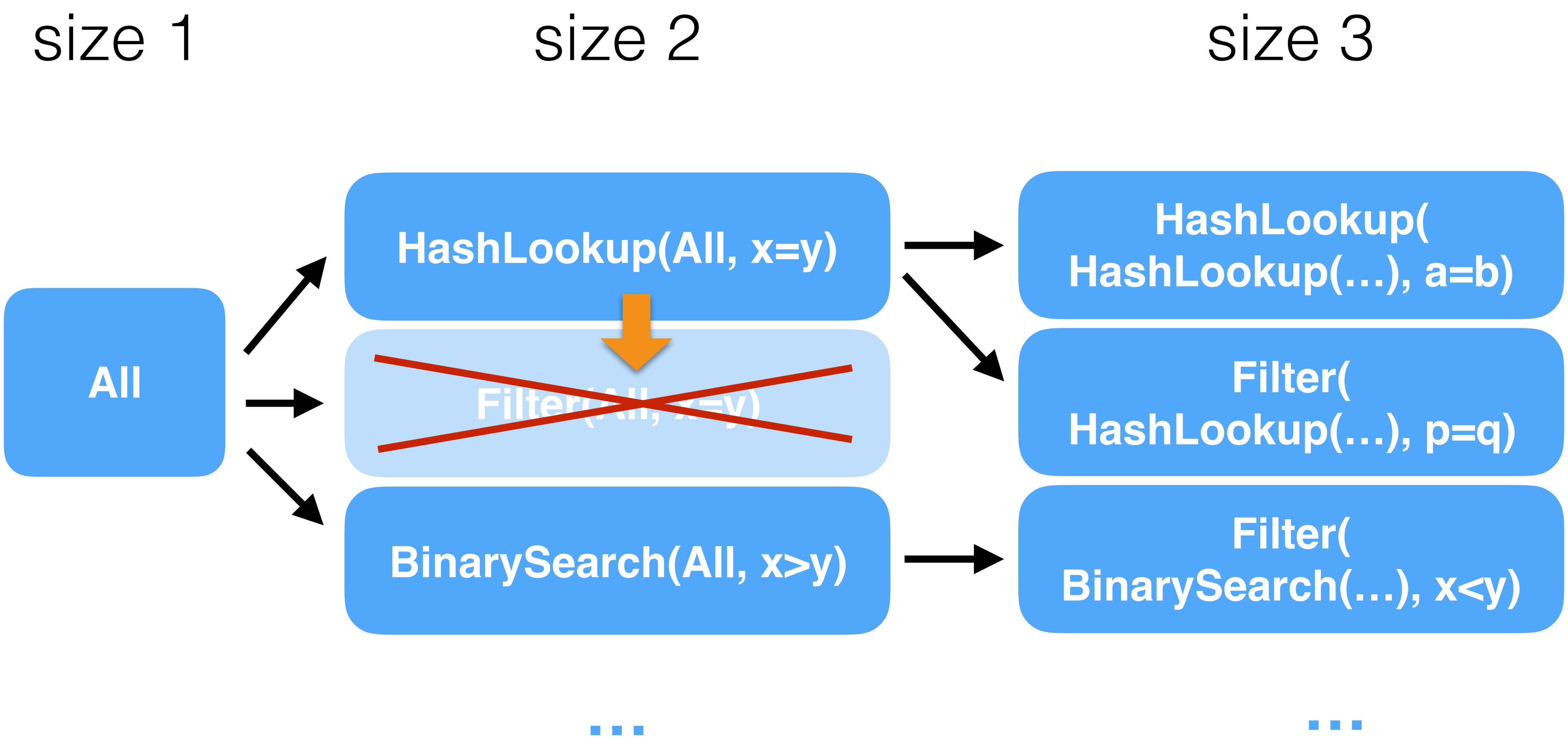
Enum

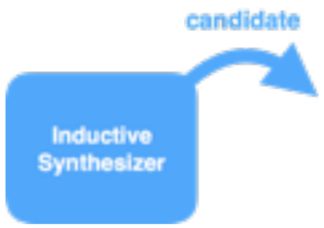




Inductive Synthesis

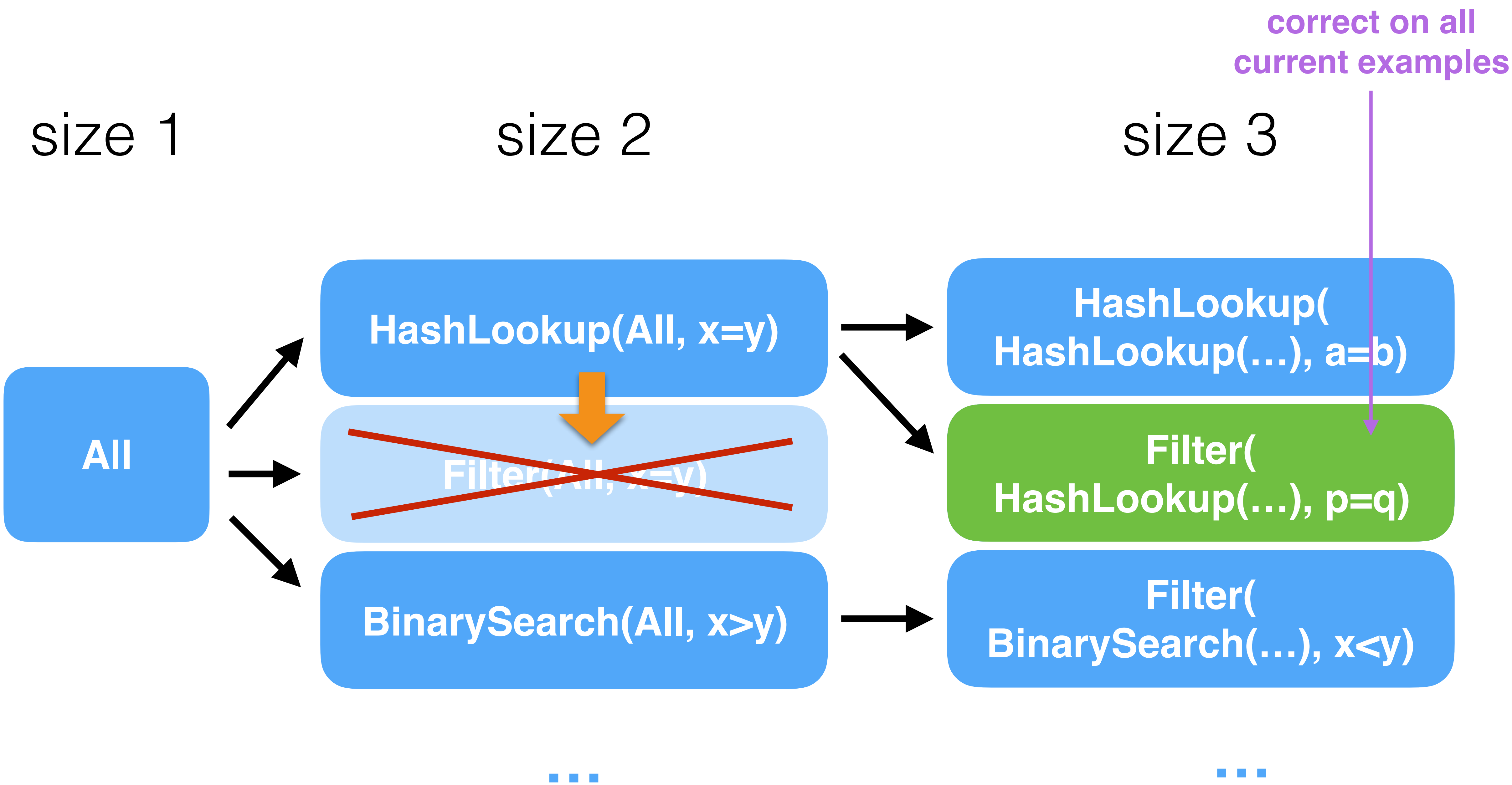
Enumerative search

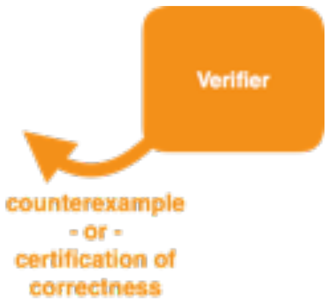




Inductive Synthesis

Enumerative search





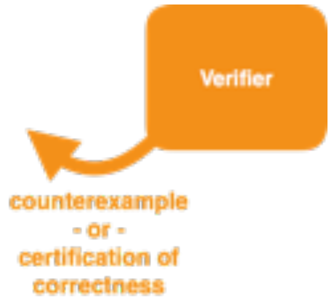
Outline Verification

Specification:

```
Entry has:
   queryId : Int,
   subqueryId : Int,
   …
```

P

```
retrieve: all e where
   e.queryId = q and …
```



Outline Verification

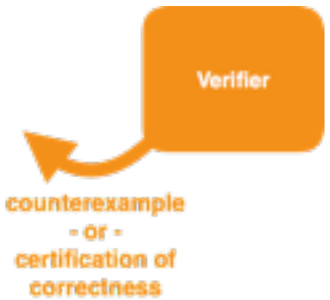
Specification:

$\{ e \mid e \in S \wedge P(I, e) \}$

```
   subquer Id : Int,
   …
```

P

```
retrieve: all e where
   e.queryId = q and …
```



Outline Verification

Specification:

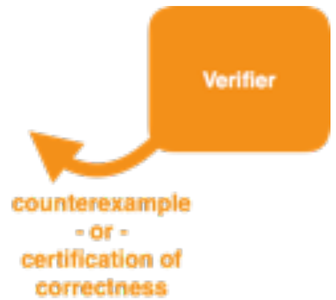
$\{\, e \mid e \in S \wedge P(I, e) \,\}$

```
subque   Id : Int,
…
```

P

```
retrieve: all e where
   e.queryId = q and …
```

HashLookup(
All(),
e.queryId = q)



Outline Verification

$$\{\, e \mid e \in S \wedge P(I, e) \,\}$$

```
    subque  Id : Int,
    …
```

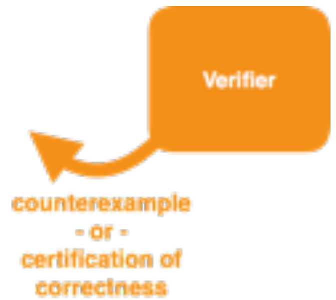
P

```
retrieve: all e where
    e.queryId = q and …
```

HashLookup(
All(),
e.queryId = q)

representative predicate Q

e.queryId = q



Outline Verification

$\{\, e \mid e \in S \wedge P(I, e) \,\}$

subque Id : Int,

…

P

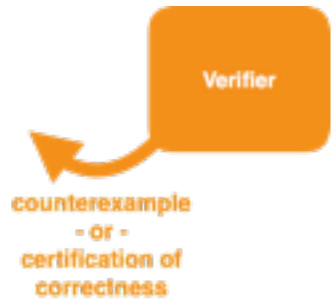
```
retrieve: all e where
    e.queryId = q and …
```

$\{\, e \mid e \in S \wedge Q(I, e) \,\}$

e.qu

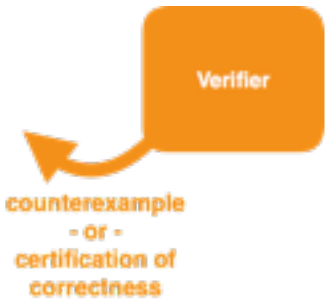
representative predicate Q

e.queryId = q



Outline Verification

$$\{ e \mid e \in S \wedge P(I, e) \} \stackrel{?}{=} \{ e \mid e \in S \wedge Q(I, e) \}$$

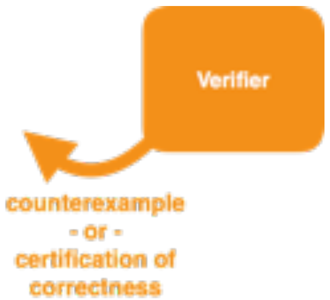


Outline Verification

$$\{ e \mid e \in S \wedge P(I, e) \} \stackrel{?}{=} \{ e \mid e \in S \wedge Q(I, e) \}$$

yes **if and only if** for all I, e:

$$P(I, e) = Q(I, e)$$



Outline Verification

$$\{\, e \mid e \in S \wedge P(I, e) \,\} \stackrel{?}{=} \{\, e \mid e \in S \wedge Q(I, e) \,\}$$

yes **if and only if** for all I, e:

$$P(I, e) = Q(I, e)$$

equivalence can be checked with an SMT solver

Evaluation

Evaluation

- Improve correctness

Evaluation

- Improve correctness
- Save programmer effort

Evaluation

- Improve correctness
- Save programmer effort
- Match performance

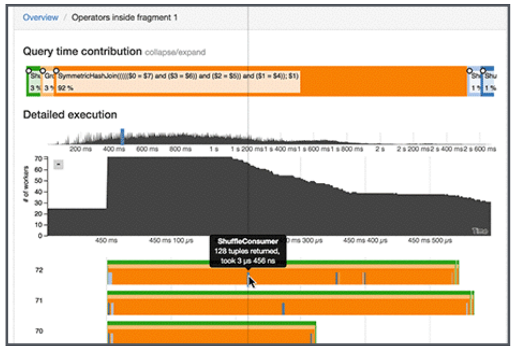
Evaluation

- Improve correctness ✓
- Save programmer effort ✓
- Match performance ✓

Case studies

Case studies

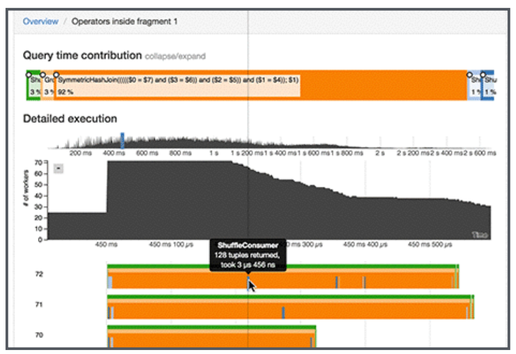
- **Myria:** analytics



Analytics data indexed by timespan and by request ID

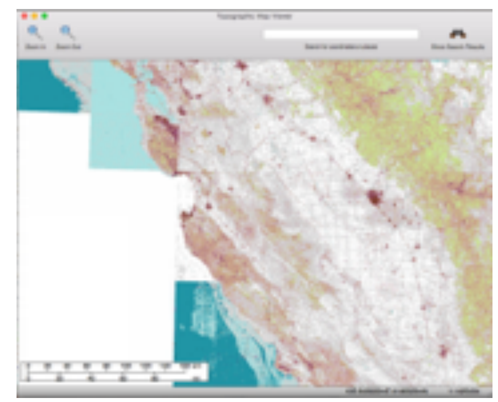
Case studies

- **Myria:** analytics



Analytics data indexed by timespan and by request ID

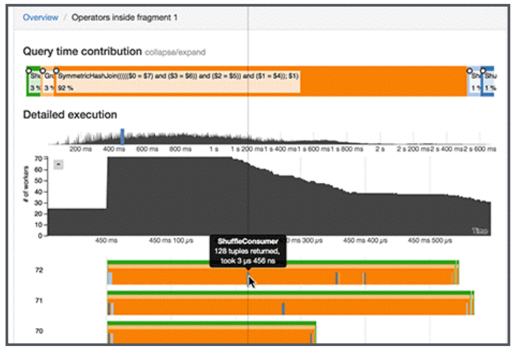
- **ZTopo:** tile cache



Tracks map tiles in a least-recently-used cache

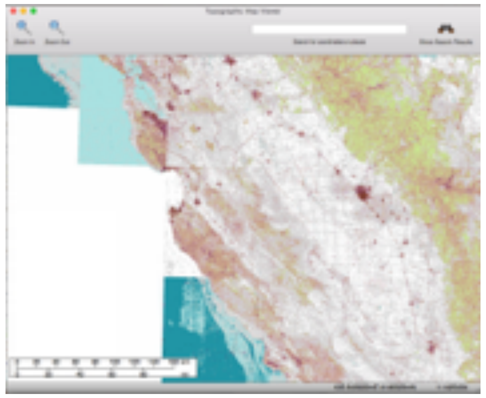
Case studies

- **Myria:** analytics



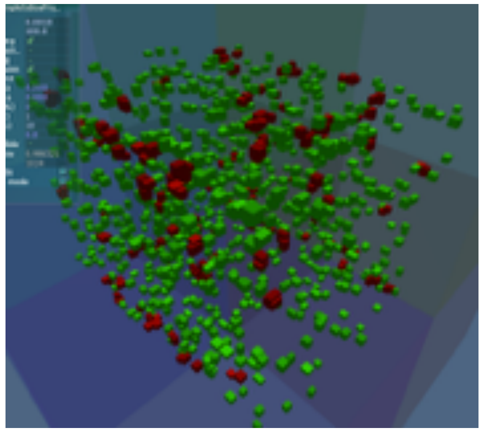
Analytics data indexed by timespan and by request ID

- **ZTopo:** tile cache



Tracks map tiles in a least-recently-used cache

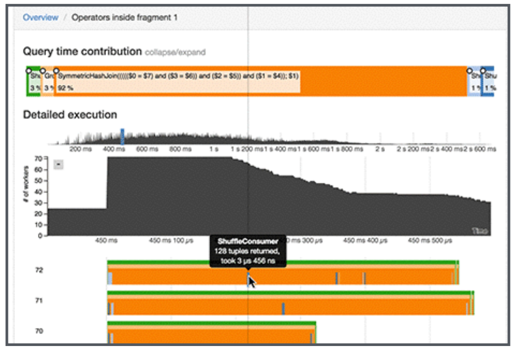
- **Bullet:** volume tree



Stores axis-aligned bounding boxes for fast collision detection

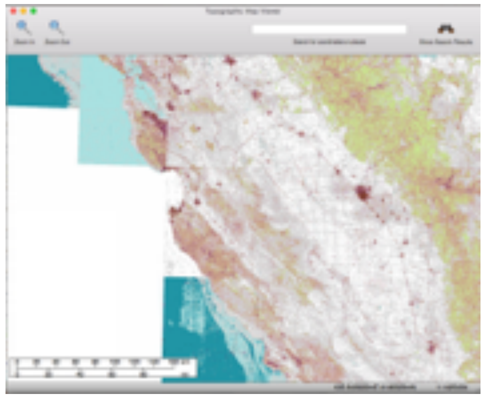
Case studies

- **Myria:** analytics



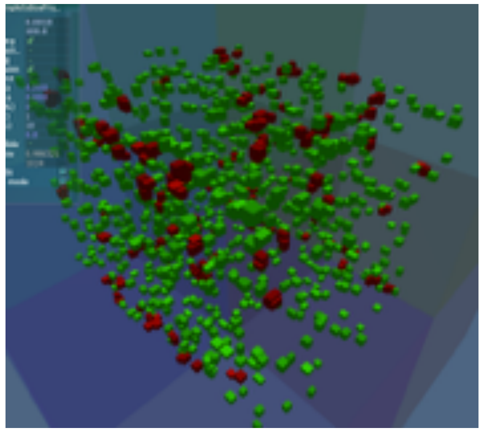
Analytics data indexed by timespan and by request ID

- **ZTopo:** tile cache



Tracks map tiles in a least-recently-used cache

- **Bullet:** volume tree



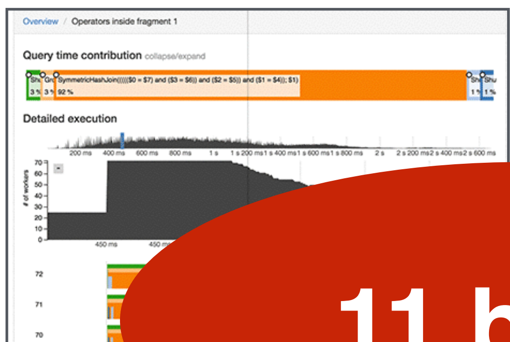
Stores axis-aligned bounding boxes for fast collision detection

- **Sat4j:** variable metadata

Tracks information about each variable in the formula

Case studies

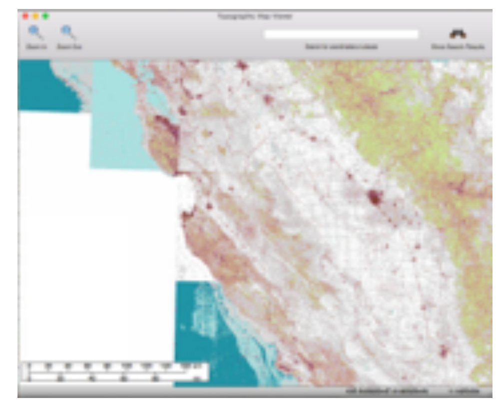
- **Myria:** analytics



Analytics data indexed by n and by

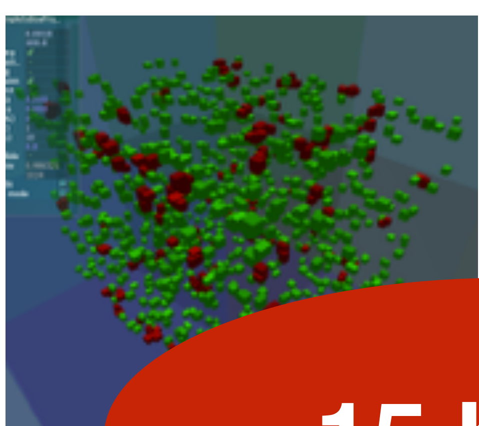
11 bugs

- **ZTopo:** tile cache



Tracks map tiles in a least-recently-used cache

- **Bullet:** volume tree



Stores axis-aligned bounding boxes for fast collision detection

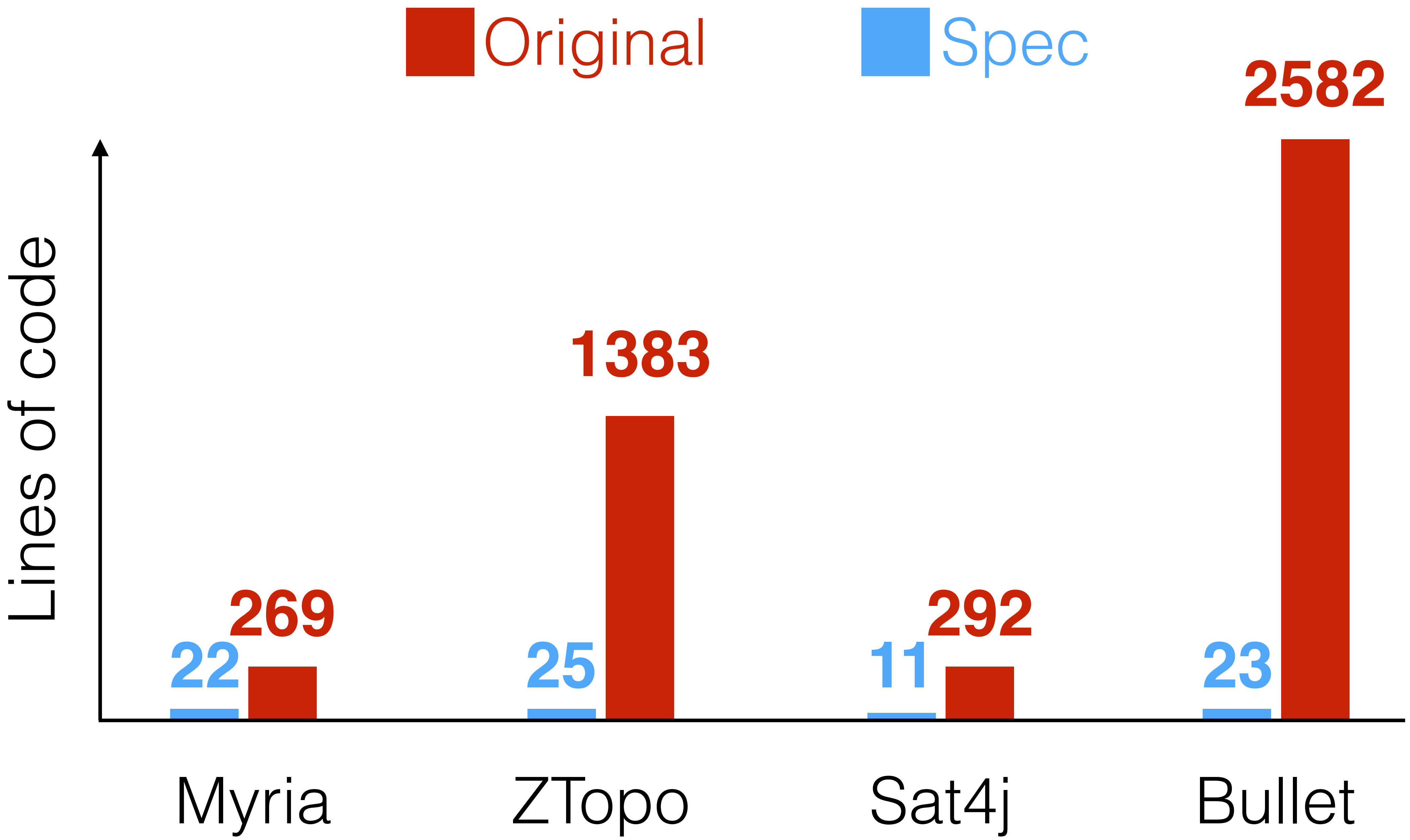
15 bugs

- **Sat4j:** variable metadata

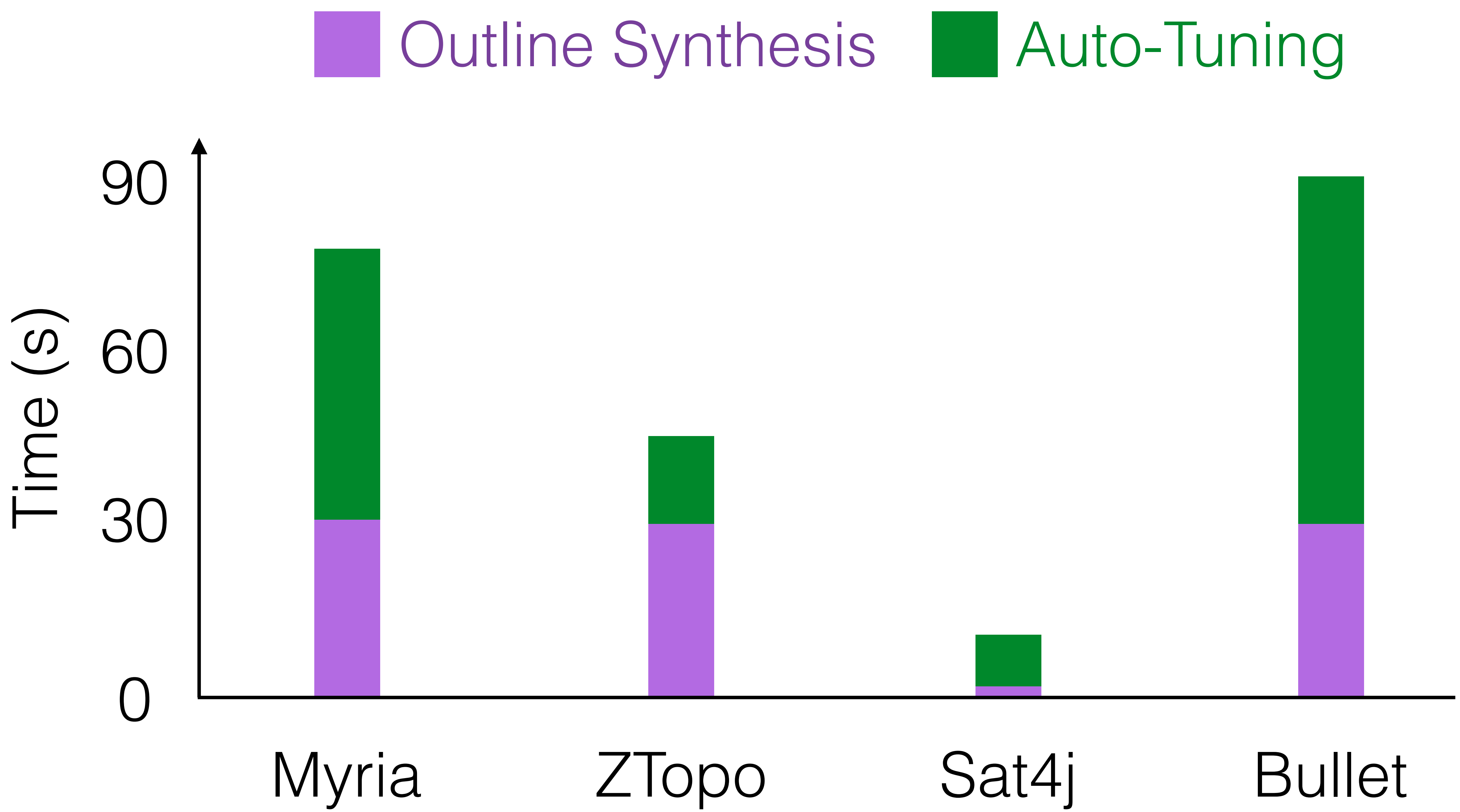
Tracks information about each variable in the formula

7 bugs

Specifications vs. Implementations



Synthesis Time



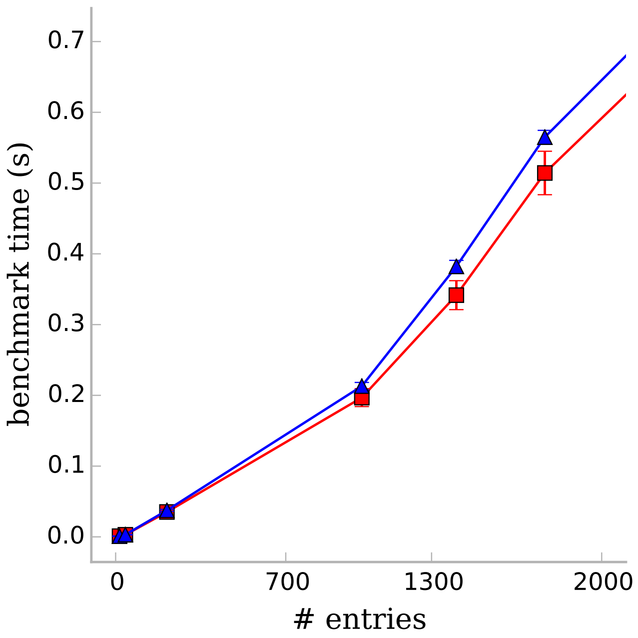
Performance

Original Synthesized

Performance

Original   Synthesized

Data structures are
nearly identical



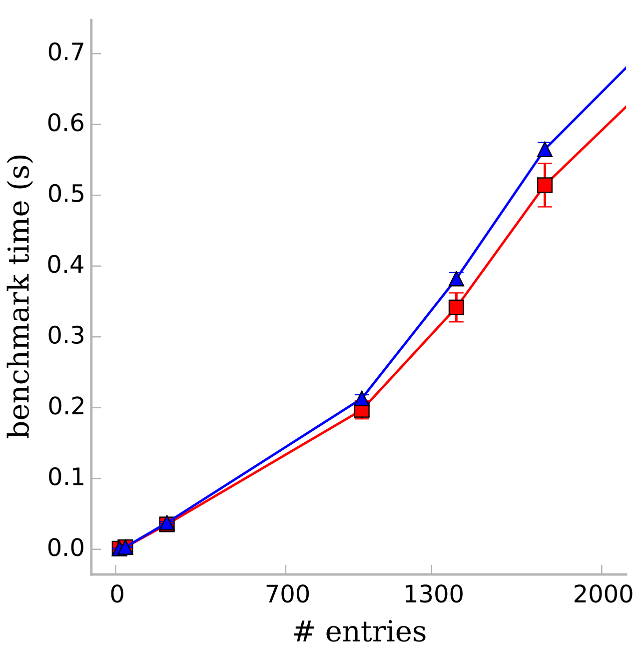
ZTopo

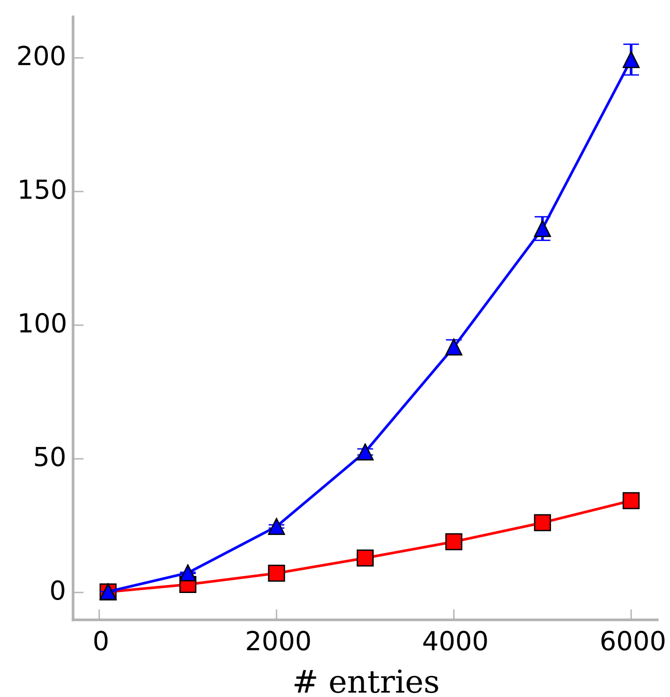
Performance

Original Synthesized

Data structures are nearly identical

Binary search tree vs. space partitioning tree





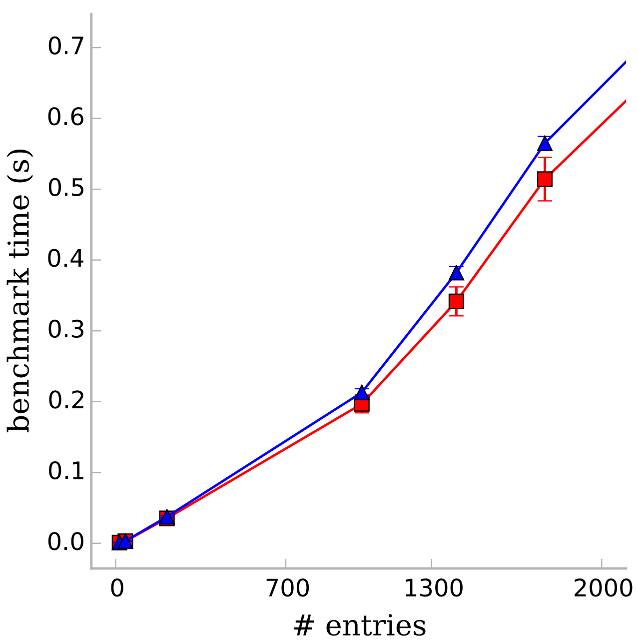
ZTopo

Bullet

Performance

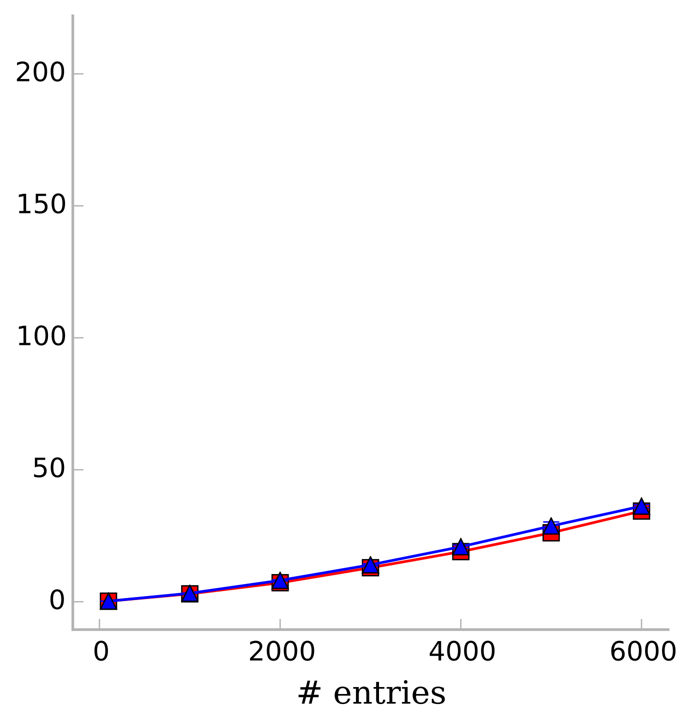
Original   Synthesized

Data structures are nearly identical



ZTopo

Binary search tree vs. space partitioning tree

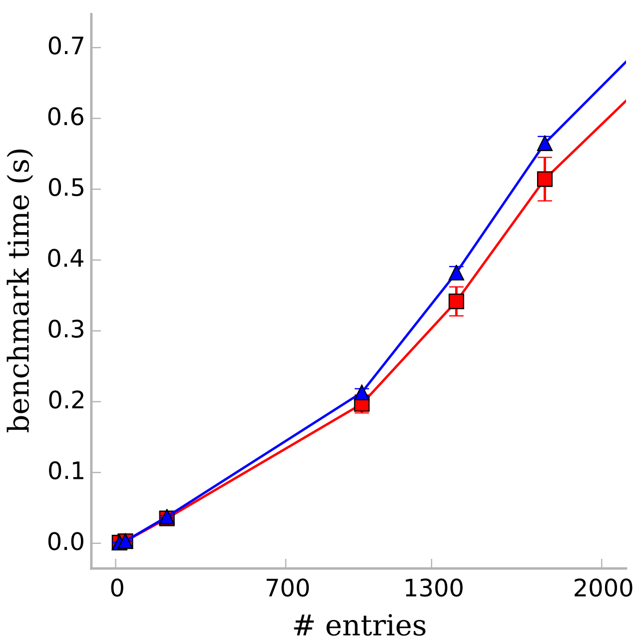


Bullet

Performance

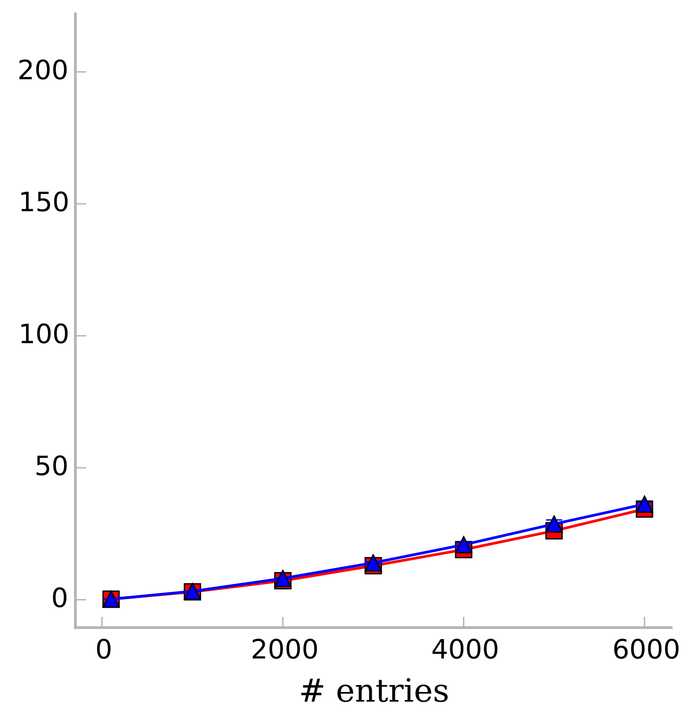
Original   Synthesized

Data structures are nearly identical



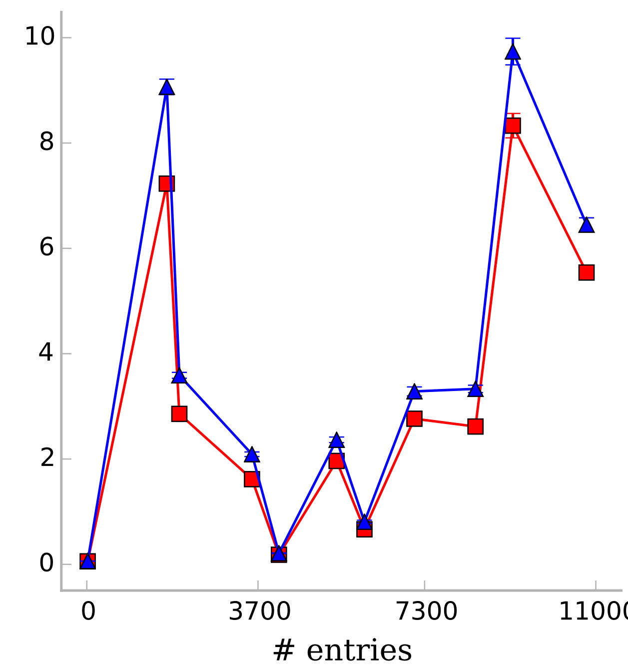
ZTopo

Binary search tree vs. space partitioning tree



Bullet

Small overhead; performance dominated by other factors

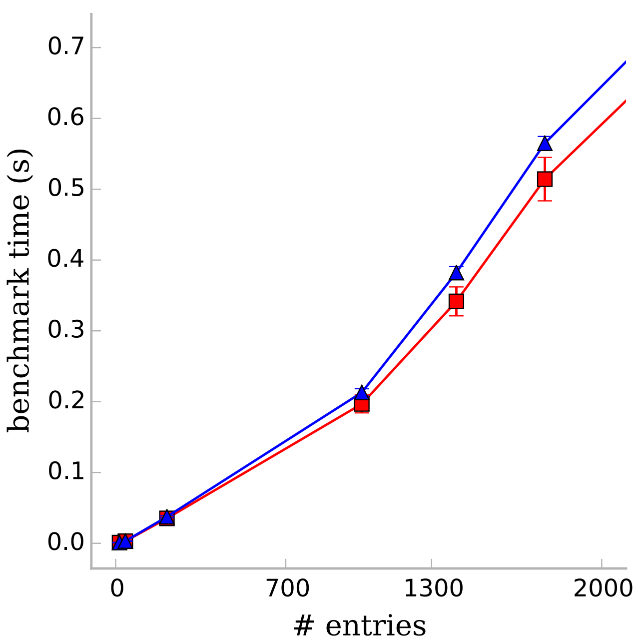


Sat4j

Performance

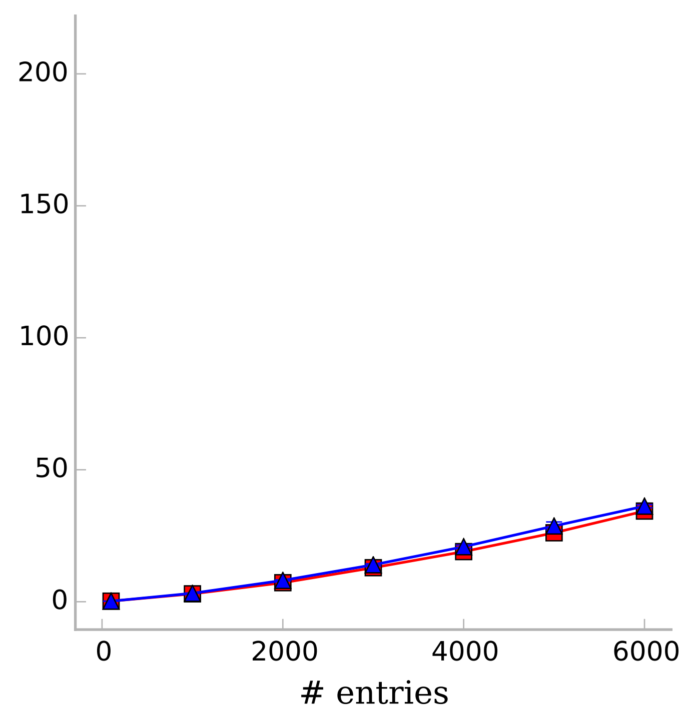
Original   Synthesized

Data structures are nearly identical



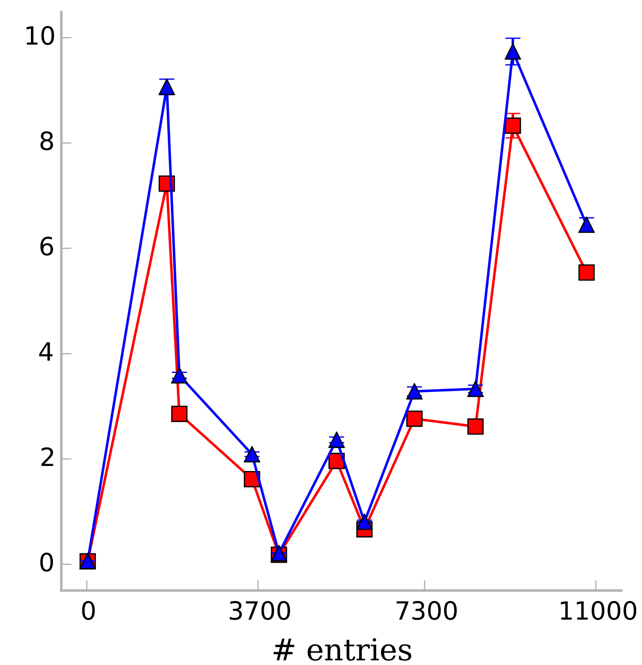
ZTopo

Binary search tree vs. space partitioning tree



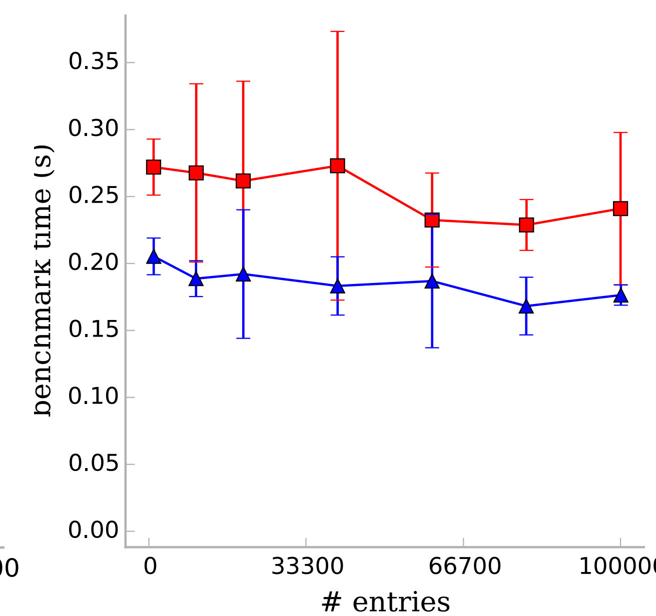
Bullet

Small overhead; performance dominated by other factors



Sat4j

Original implementation has worst-case linear time

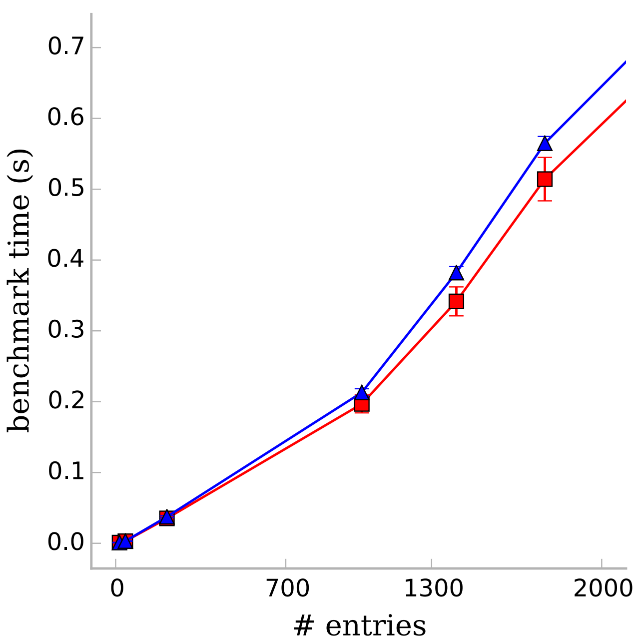


Myria

Performance

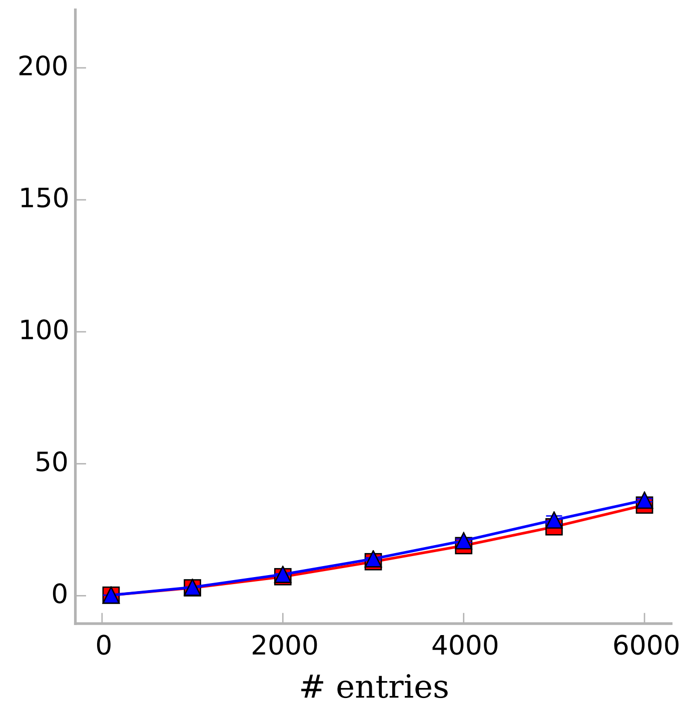
Original   Synthesized

Data structures are nearly identical



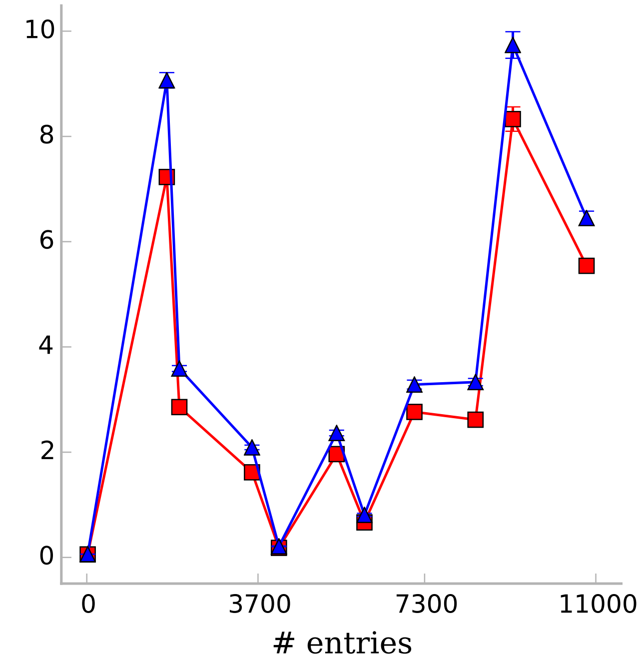
ZTopo

Binary search tree vs. space partitioning tree



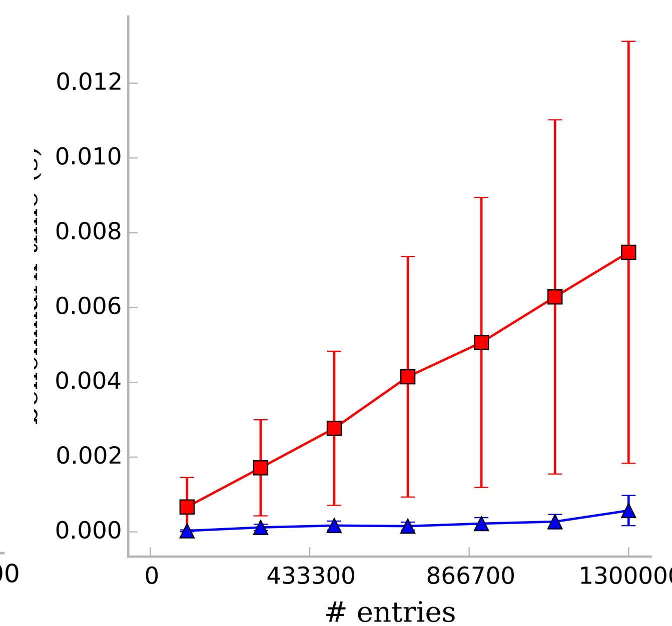
Bullet

Small overhead; performance dominated by other factors



Sat4j

Original implementation has worst-case linear time



Myria

Related Work

Related Work

- J. Earley: "**High level iterators and a method for automatically designing data structure representation**" (1974)
    - Hard-coded rewrite rules

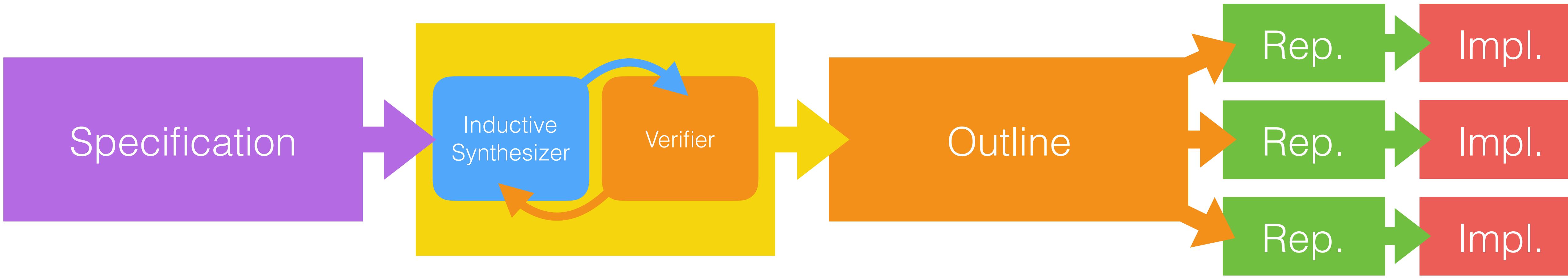
Related Work

- J. Earley: “**High level iterators and a method for automatically designing data structure representation**” (1974)
  - Hard-coded rewrite rules
- S. Agrawal et al: “**Automated selection of materialized views and indexes in sql databases**” (2000)
  - Enumerate possible views & indexes based on query syntax and use the planner to decide which ones to keep

Related Work

- J. Earley: "**High level iterators and a method for automatically designing data structure representation**" (1974)
  - Hard-coded rewrite rules
- S. Agrawal et al: "**Automated selection of materialized views and indexes in sql databases**" (2000)
  - Enumerate possible views & indexes based on query syntax and use the planner to decide which ones to keep
- P. Hawkins et al: "**Data representation synthesis**" (2011)
  - Enumerate representations and use a planner to implement retrieval operations; conjunctions of equalities only

http://cozy.uwplse.org



- Implementation outlines make the problem tractable
- Synthesis completes < 90 seconds
- Cozy generates correct code, and matches handwritten implementation performance

Special thanks to:


Michael Ernst


Emina Torlak

also Haoming Liu & Daniel Perelman