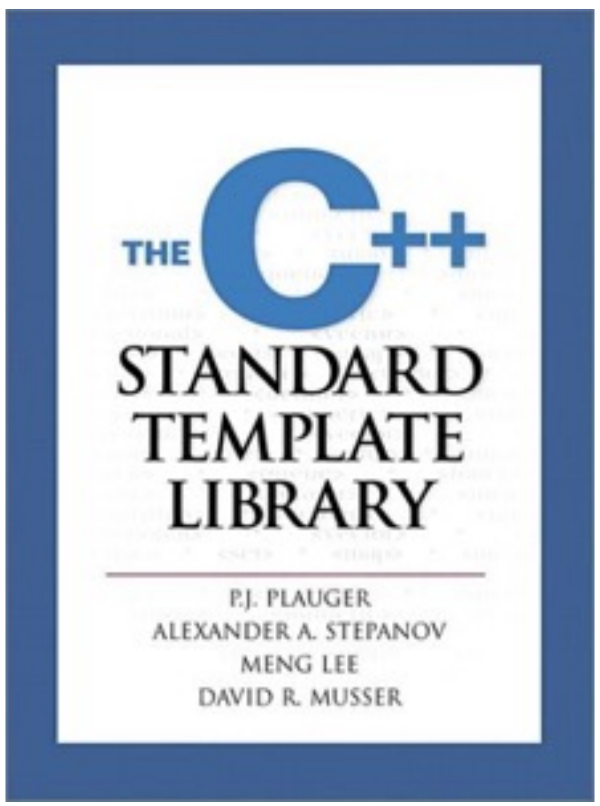
Data Structure Synthesis

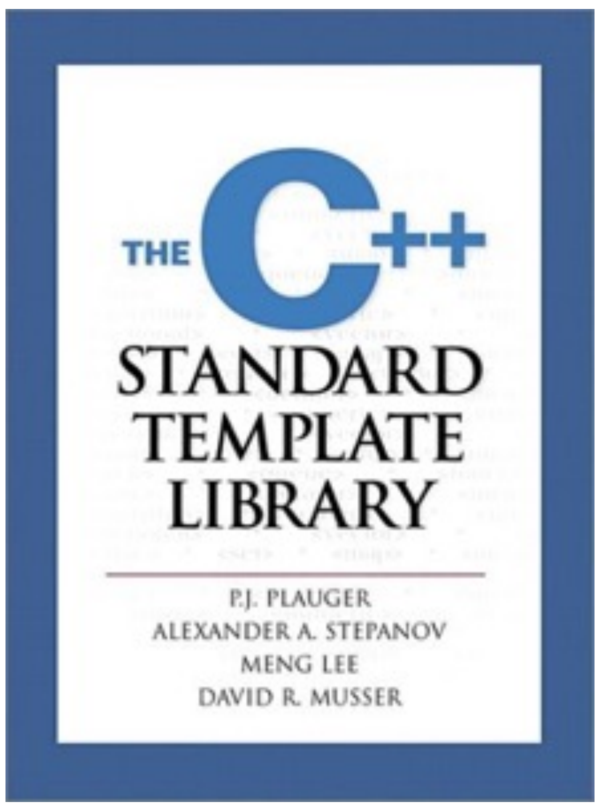
Calvin Loncaric
Emina Torlak
Michael D. Ernst

Data structures are everywhere

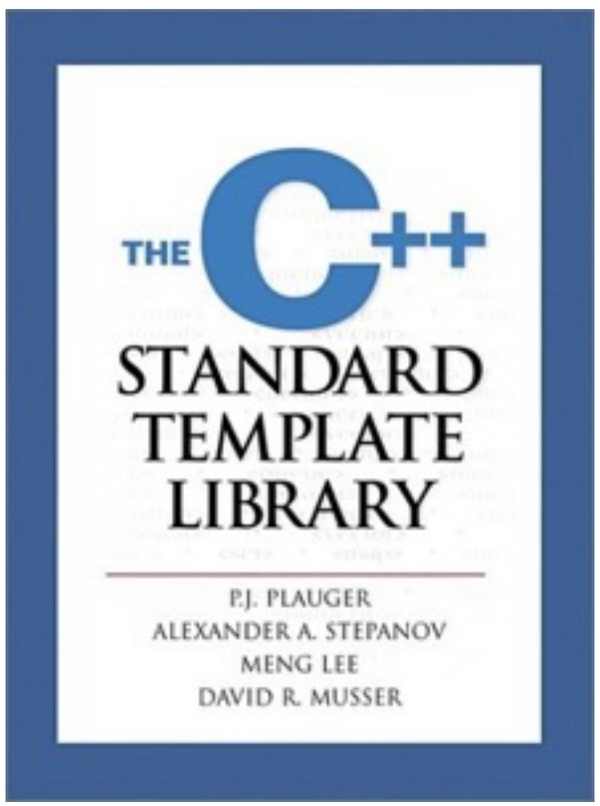
Data structures are everywhere



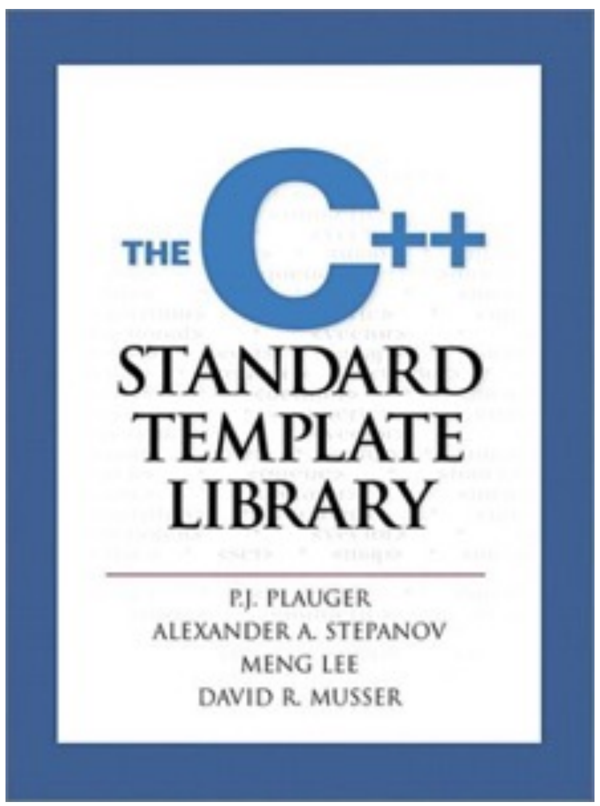
Data structures are everywhere



Data structures are everywhere

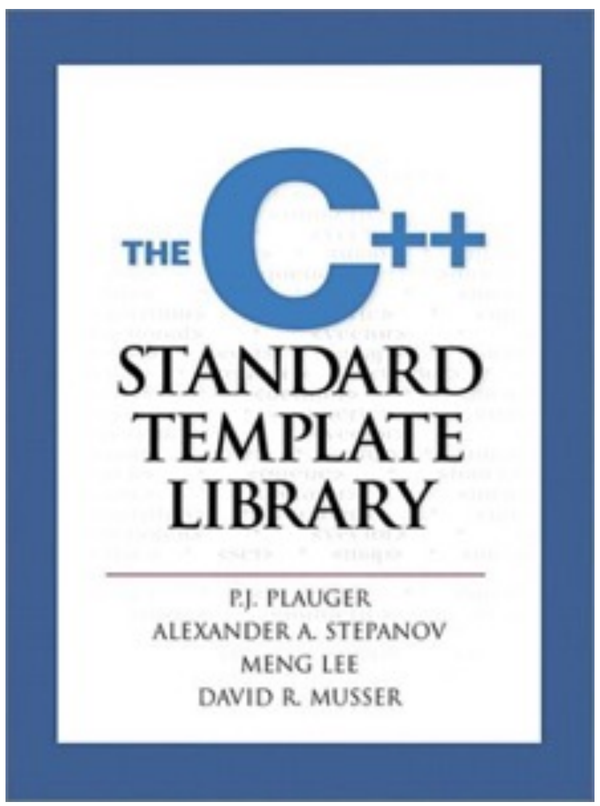


Data structures are everywhere



Lists, maps, and sets solve many problems

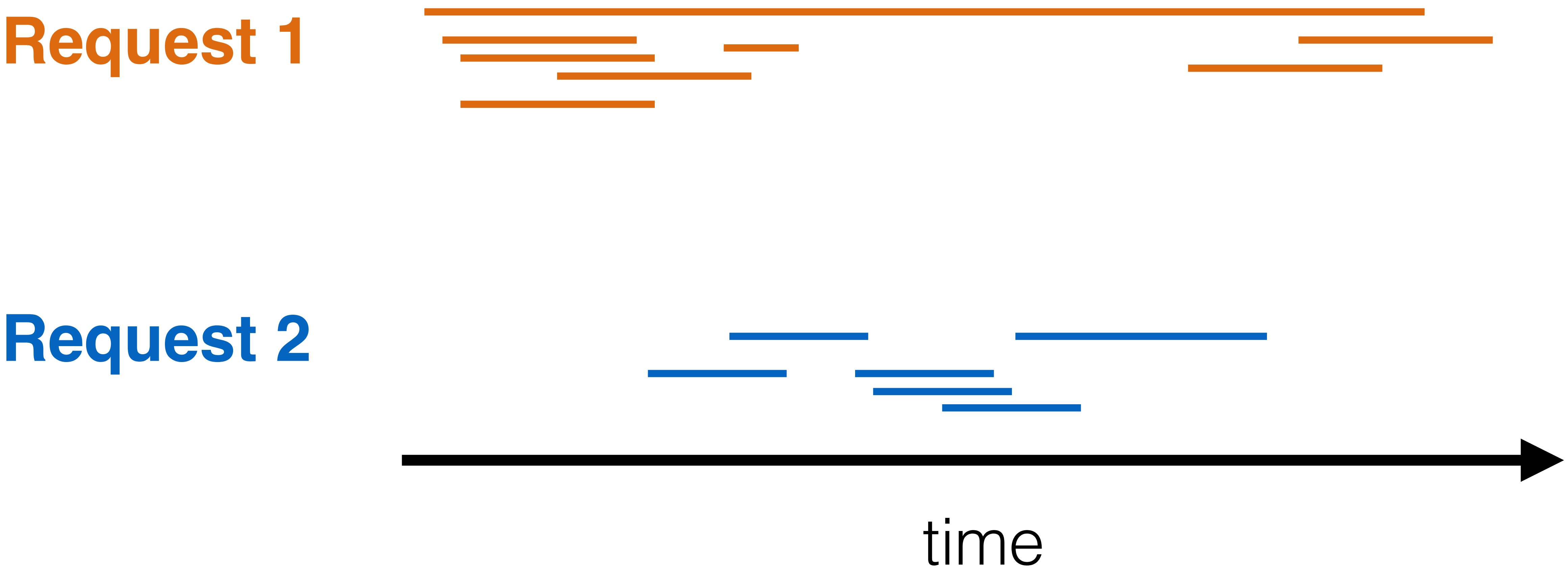
Data structures are everywhere



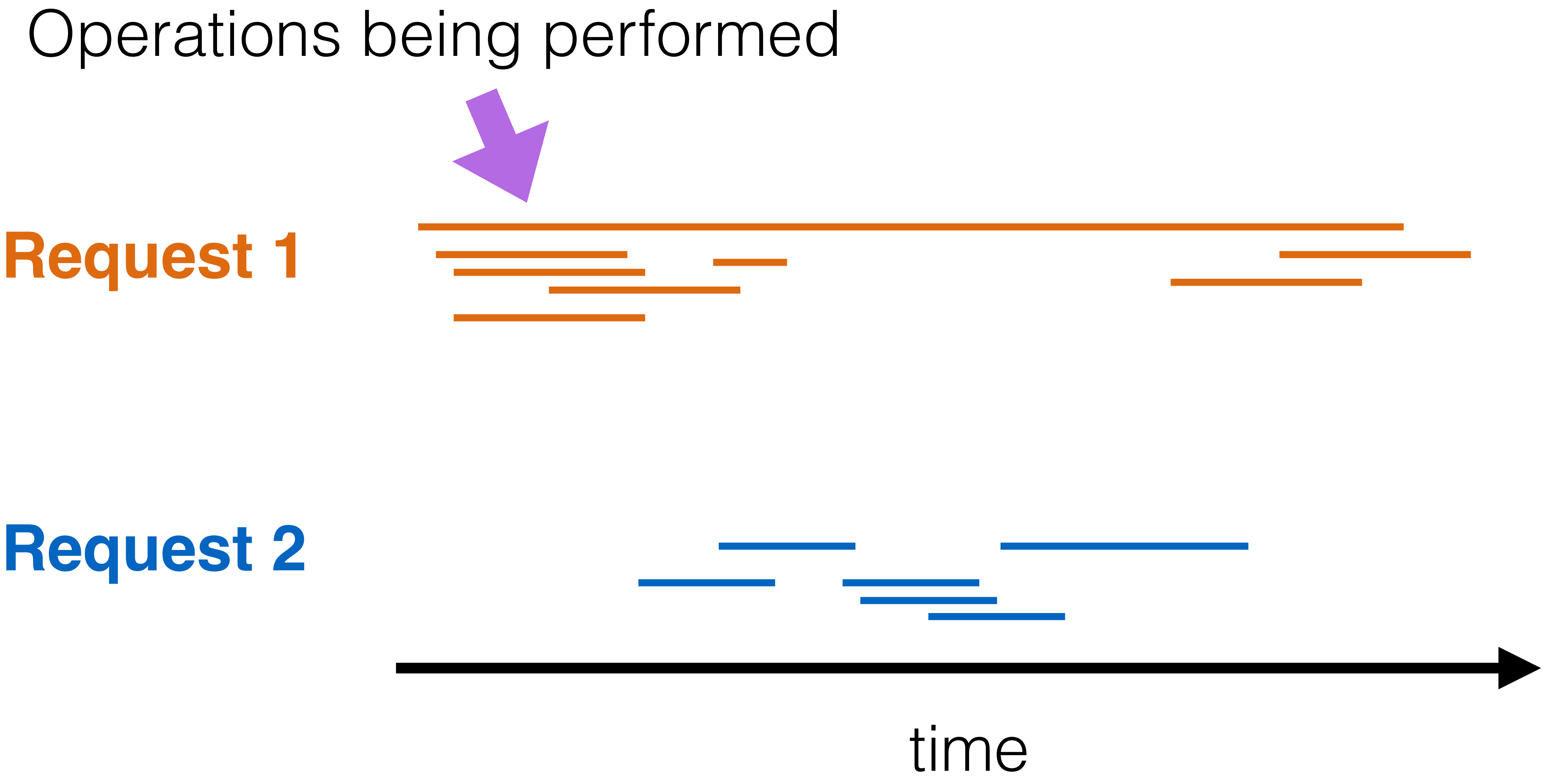
Lists, maps, and sets solve many problems

What if I need a custom data structure?

Myria Analytics Storage



Myria Analytics Storage



Myria Analytics Storage

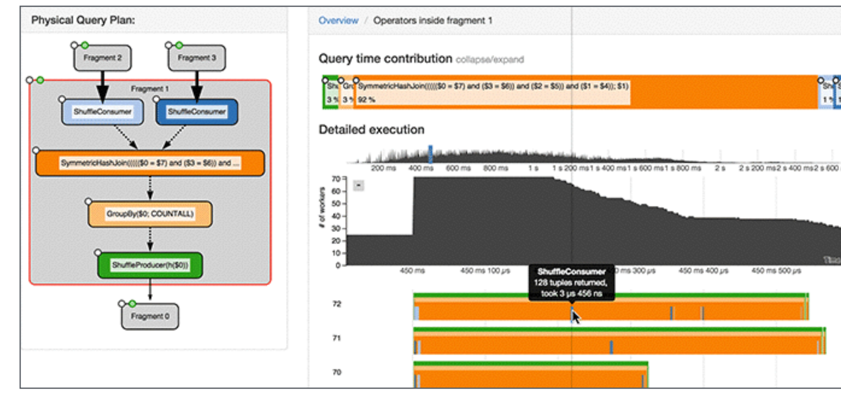
Operations bei

Request 1

Request 2

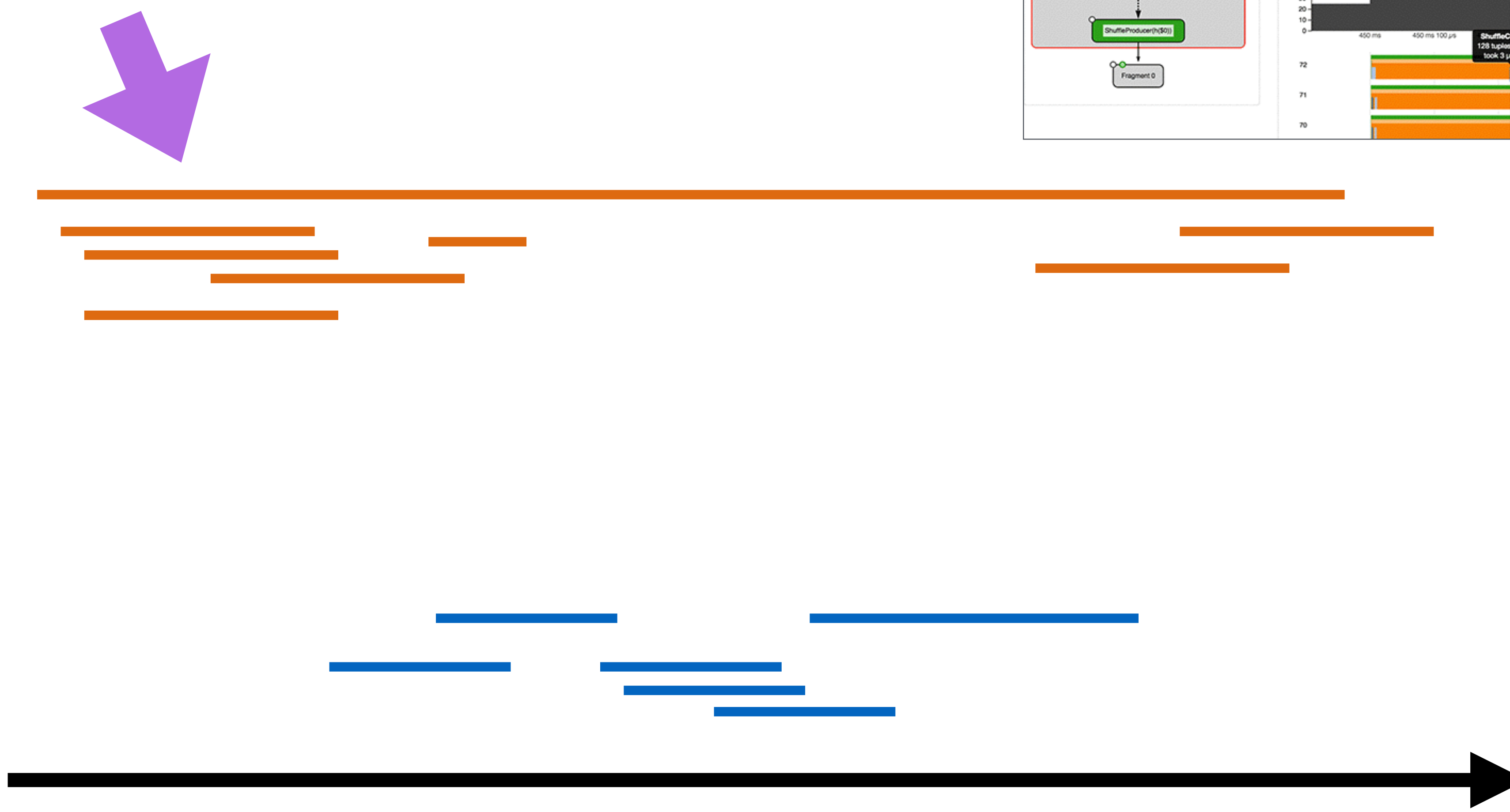
time

Myria Analytics Storage



Operations being performed

Request 1

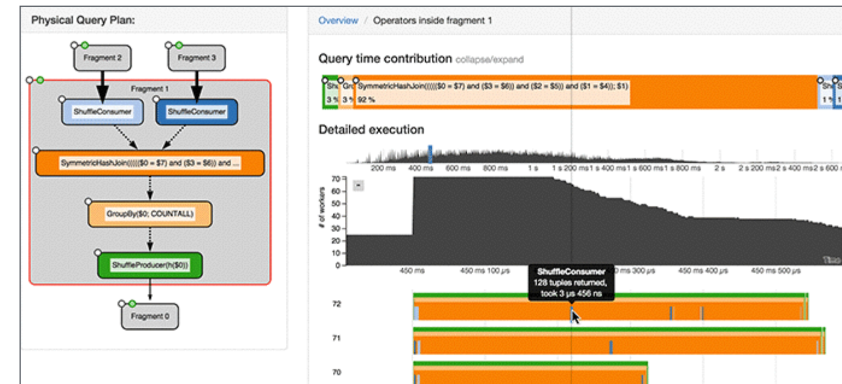


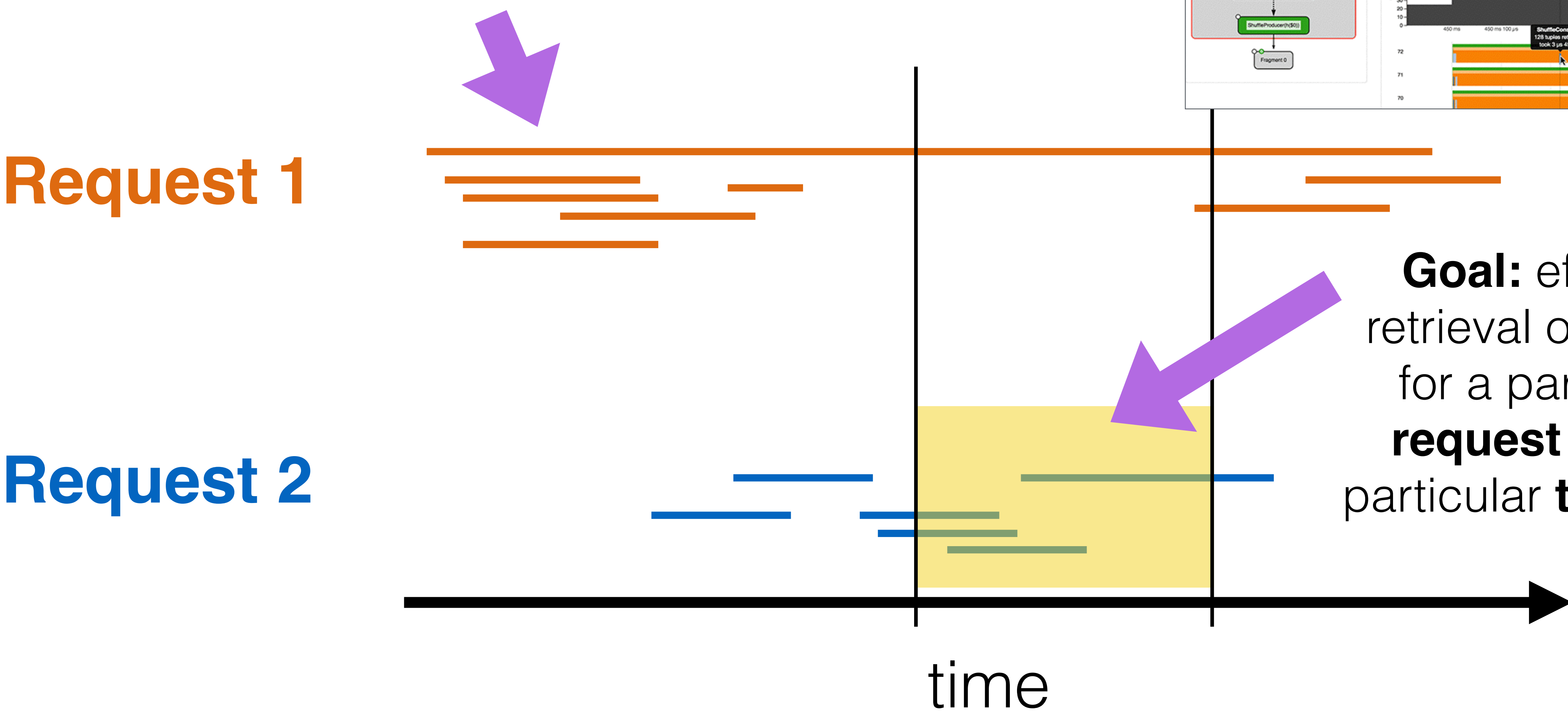
Request 2

time

Myria Analytics Storage

Operations being performed





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

```
class AnalyticsLog {

    void log(Entry e)

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

}
```

Retrieve entries

Myria Analytics Storage

Specification:

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

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)

}
```

Myria Analytics Storage

Specification:

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

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

Cozy

```
class AnalyticsLog {

    void log(Entry e)

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

}
```

Cozy synthesizes collections

Cozy synthesizes collections

Specification:

```
Entry has:
   field1
   field2
   …

retrieveA: all e where
   e.field1 < var1 and …

retrieveB: all e where
   e.field1 > var1 and …
```

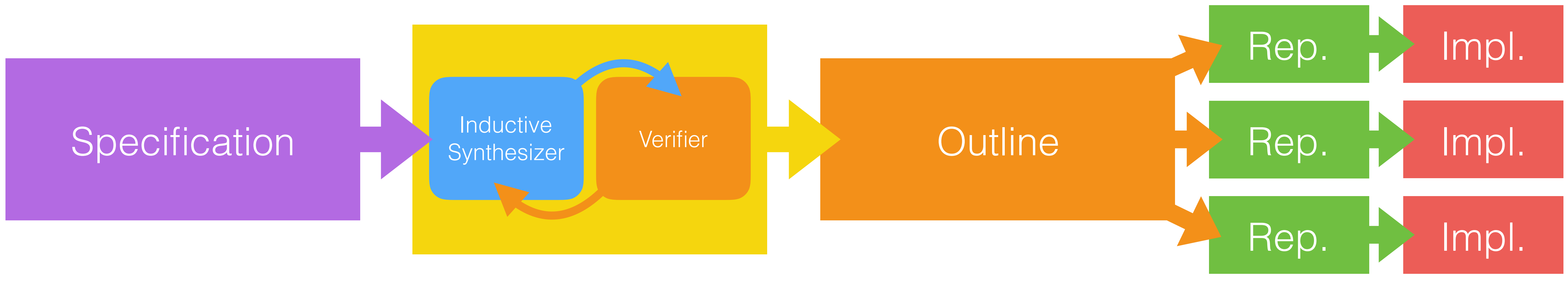
Cozy synthesizes collections

Specification:

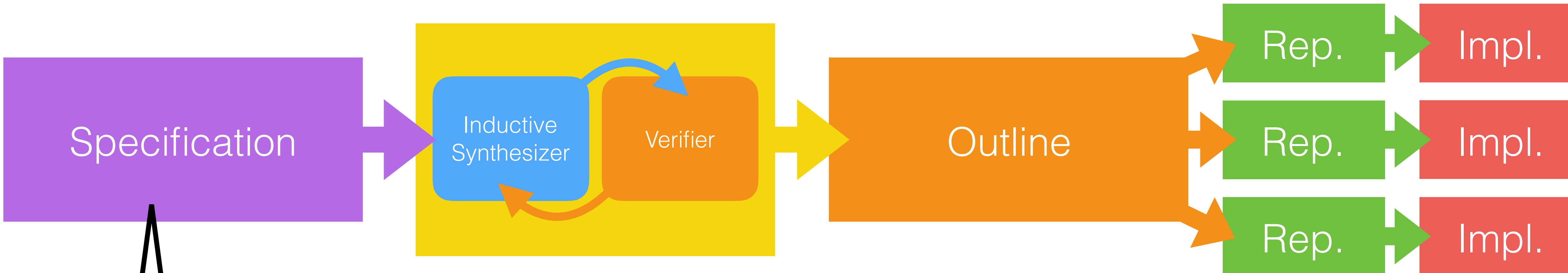
```
Entry has:
   field1
   field2
   …

retrieveA: all e where
   e.field1 < var1 and …

retrieveB: all e where
   e.field1 > var1 and …
```


```
class Structure {

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

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

}
```

Architecture



Architecture



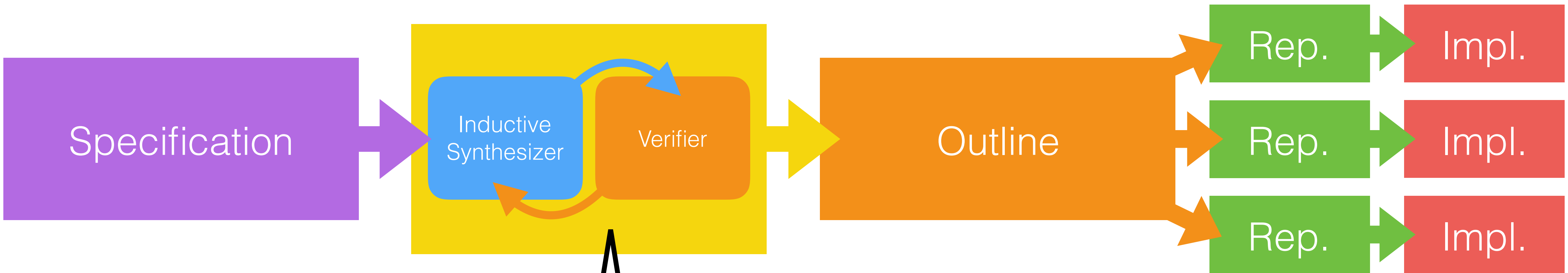
What fields does each entry have?

What query operations are needed?

Specification:

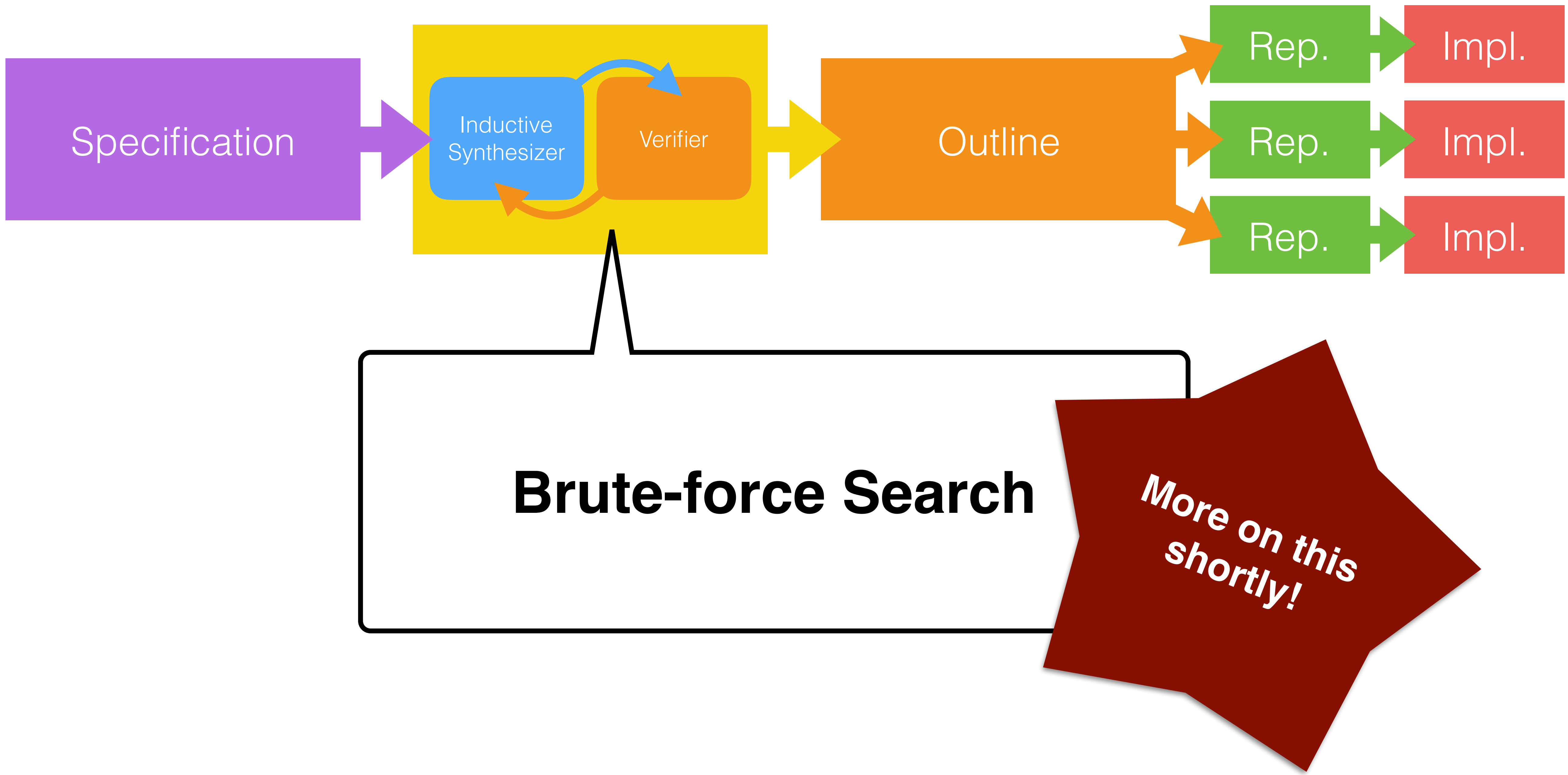
```
Entry has:
    field1
    field2
    …


retrieveA: all e where
    e.field1 < var1 and …


retrieveB: all e where
    e.field1 > var1 and …
```

Architecture

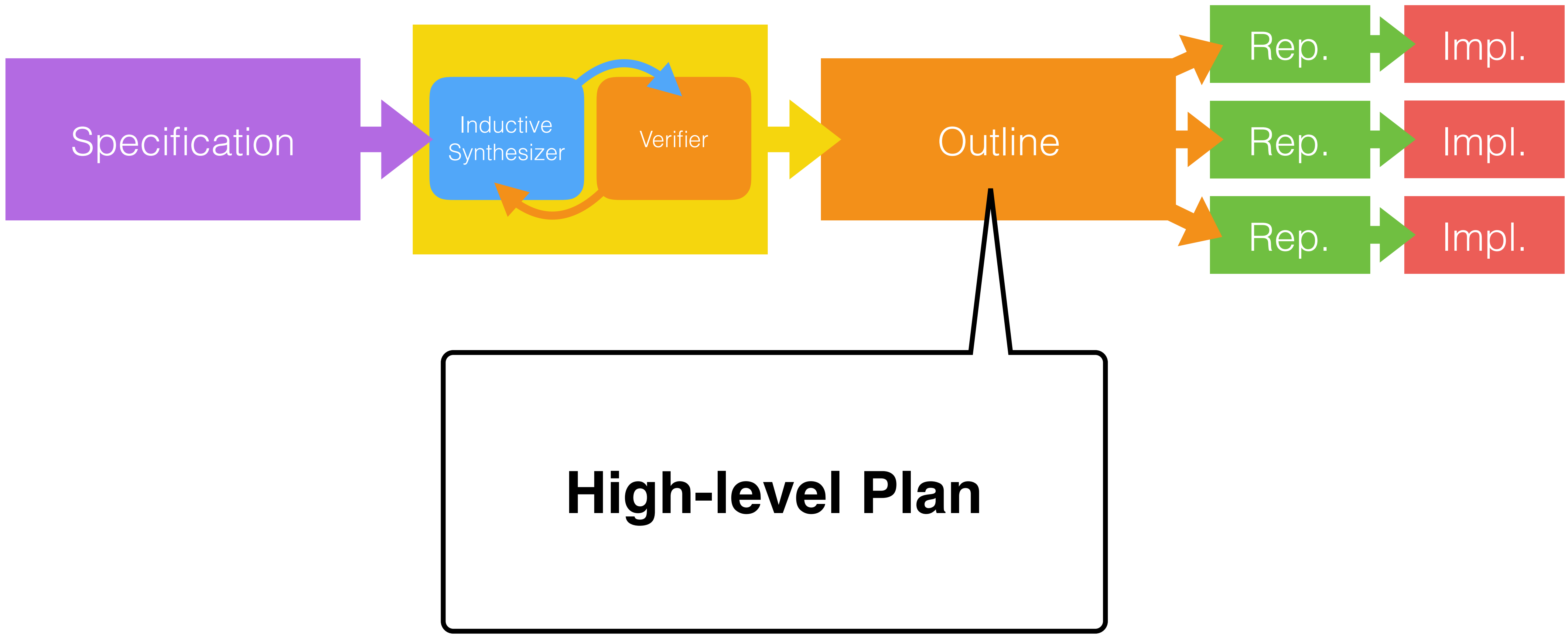


Brute-force Search

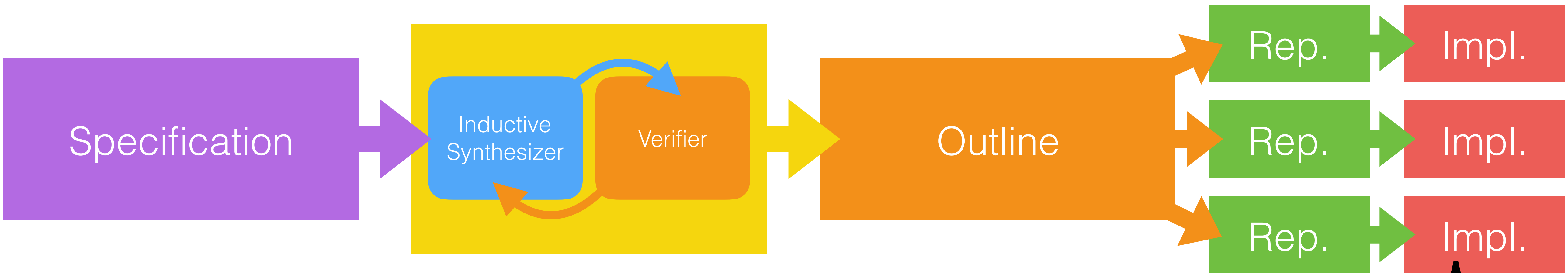
Architecture



Architecture



Architecture



Benchmark to make low-level choices

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, outline)

Outlines

Plans for retrieving entries

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

Outlines

Plans for retrieving entries

- **All** ( ) **1**
- **HashLookup** (outline, field = var)
- **BinarySearch** (outline, field > var)
- **Concat** (outline, outline)
- **Filter** (outline, predicate)

Outlines

Plans for retrieving entries

- **All** ( ) **1**
- **HashLookup** (outline, field = var) $c + 1$
- **BinarySearch** (outline, field > var)
- **Concat** (outline, outline)
- **Filter** (outline, predicate)

Outlines

Plans for retrieving entries

- **All** ( ) $\mathbf{1}$
- **HashLookup** (outline, field = var) $\boldsymbol{c} + \mathbf{1}$
- **BinarySearch** (outline, field > var) $\boldsymbol{c} + \mathbf{log}\ \boldsymbol{n}$
- **Concat** (outline, outline)
- **Filter** (outline, predicate)

Outlines

Plans for retrieving entries

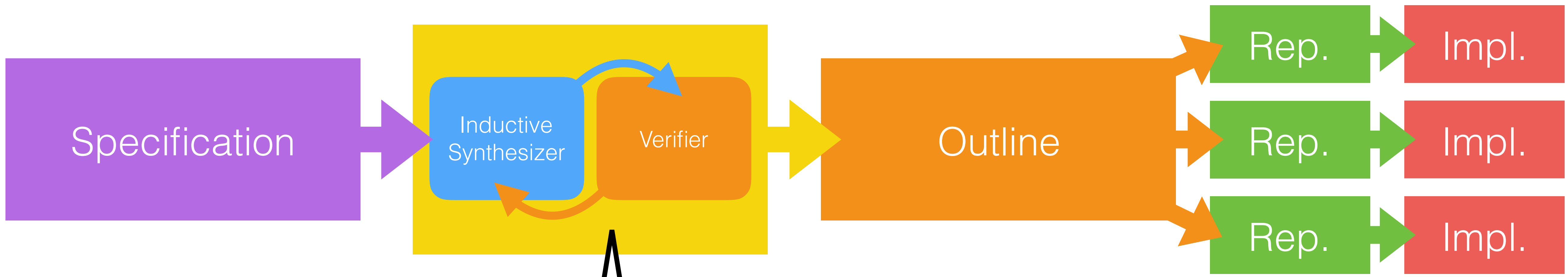
- **All** ( ) — $\mathbf{1}$
- **HashLookup** (outline, field = var) — $c + 1$
- **BinarySearch** (outline, field > var) — $c + \log n$
- **Concat** (outline, outline) — $c_1 + c_2$
- **Filter** (outline, predicate)

Outlines

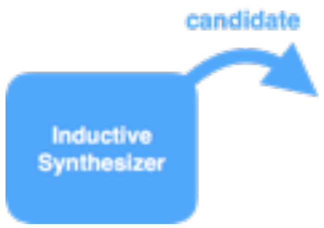
Plans for retrieving entries

- **All** ( ) $\mathbf{1}$
- **HashLookup** (outline, field = var) $c + 1$
- **BinarySearch** (outline, field > var) $c + \log n$
- **Concat** (outline, outline) $c_1 + c_2$
- **Filter** (outline, predicate) $c + n$

Architecture

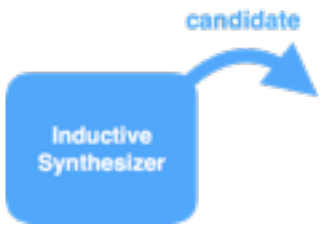


Brute-force Search



Specification —> Outline

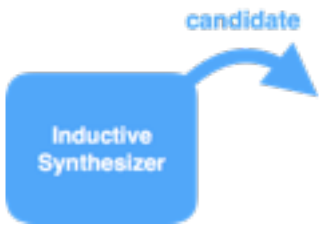
Enumerative search



Specification —> Outline

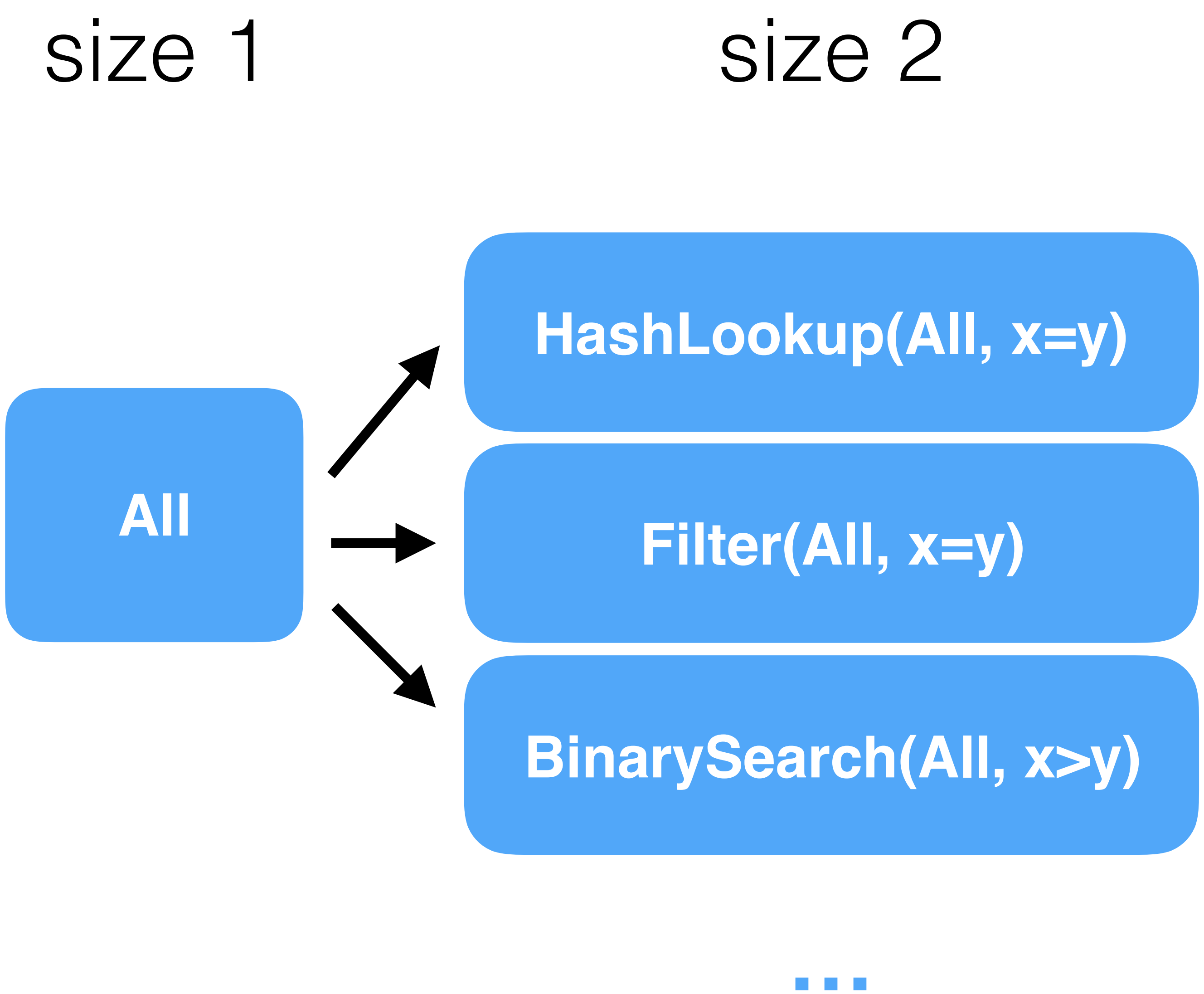
Enumerative search

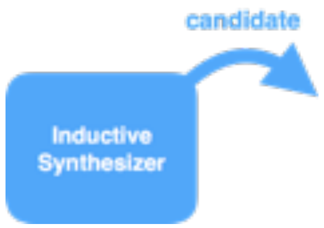
size 1



Specification —> Outline

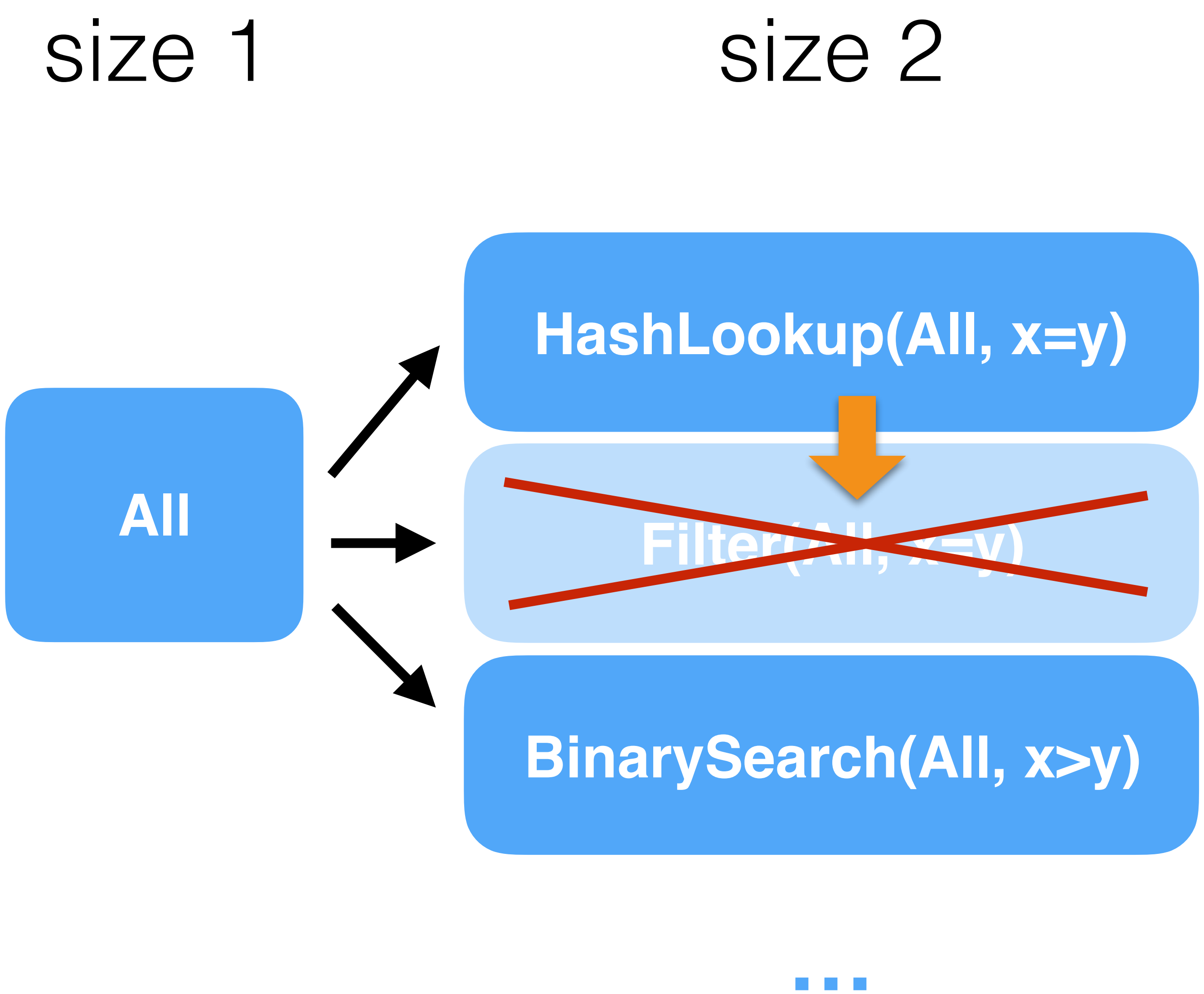
Enumerative search

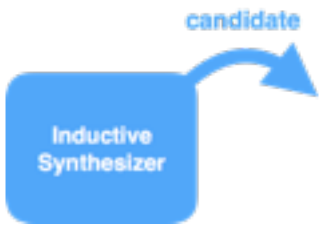




Specification —> Outline

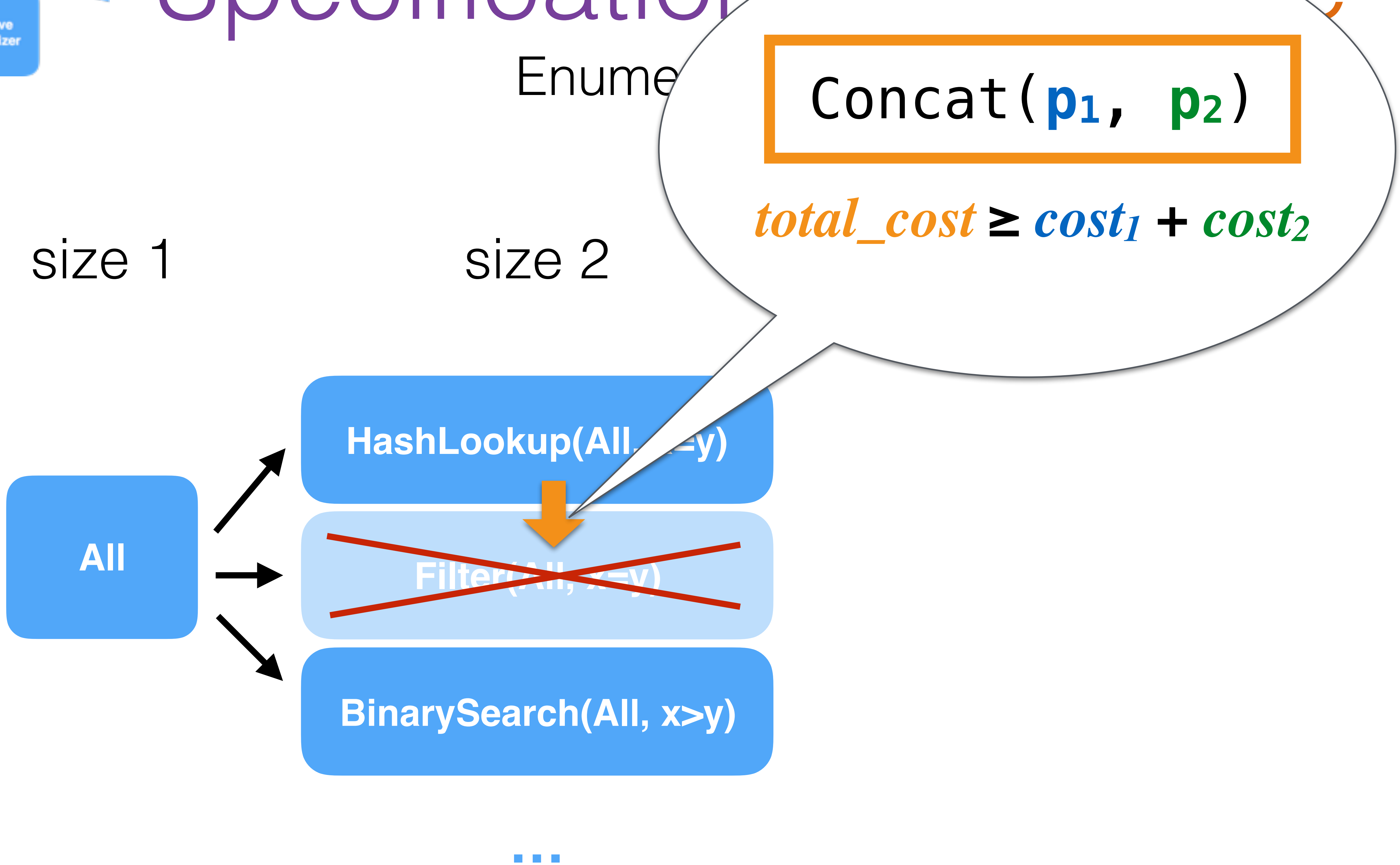
Enumerative search

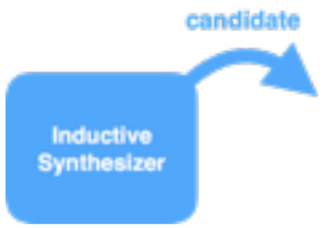




Specification ...se

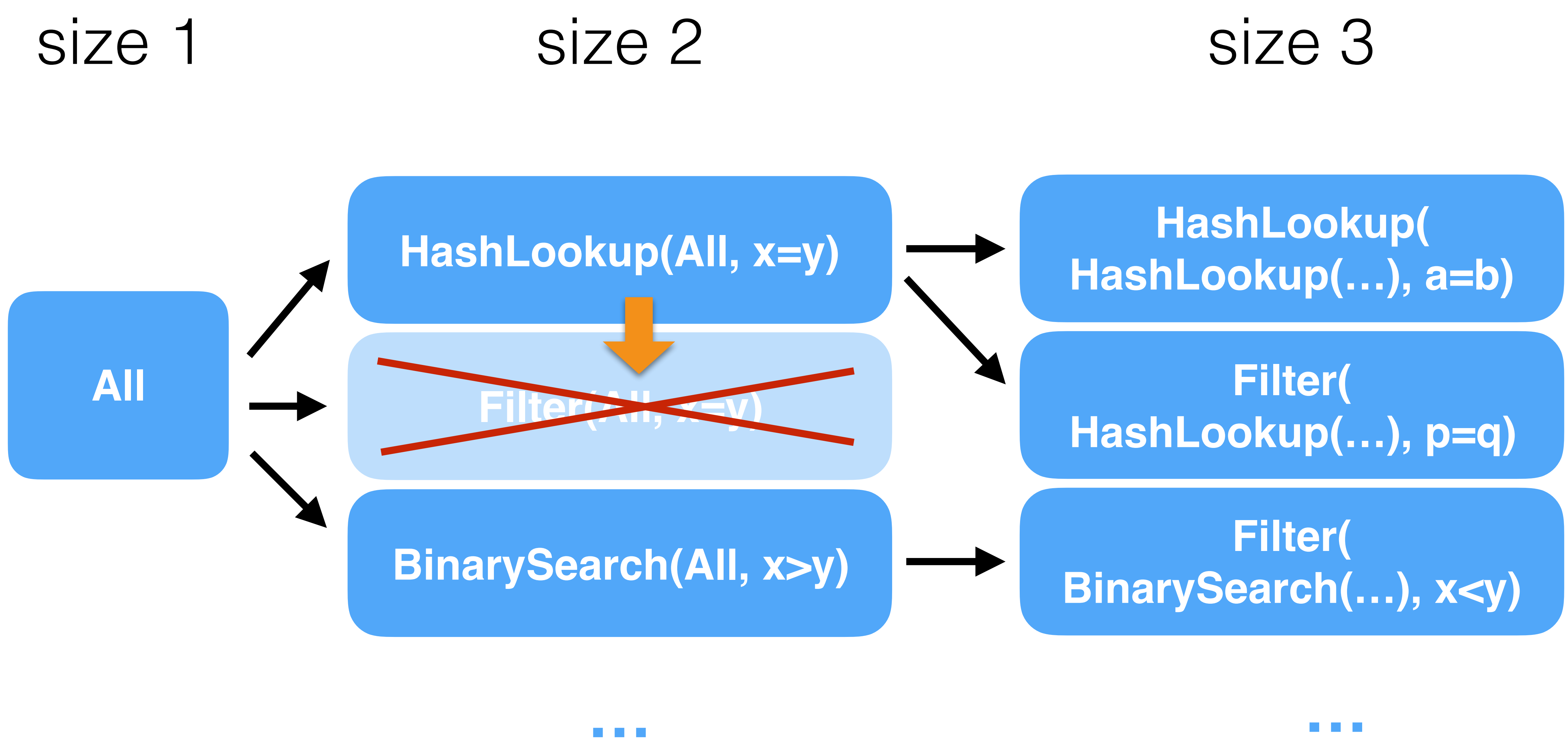
Enume...

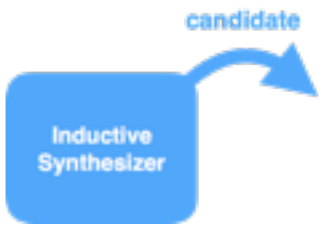




Specification —> Outline

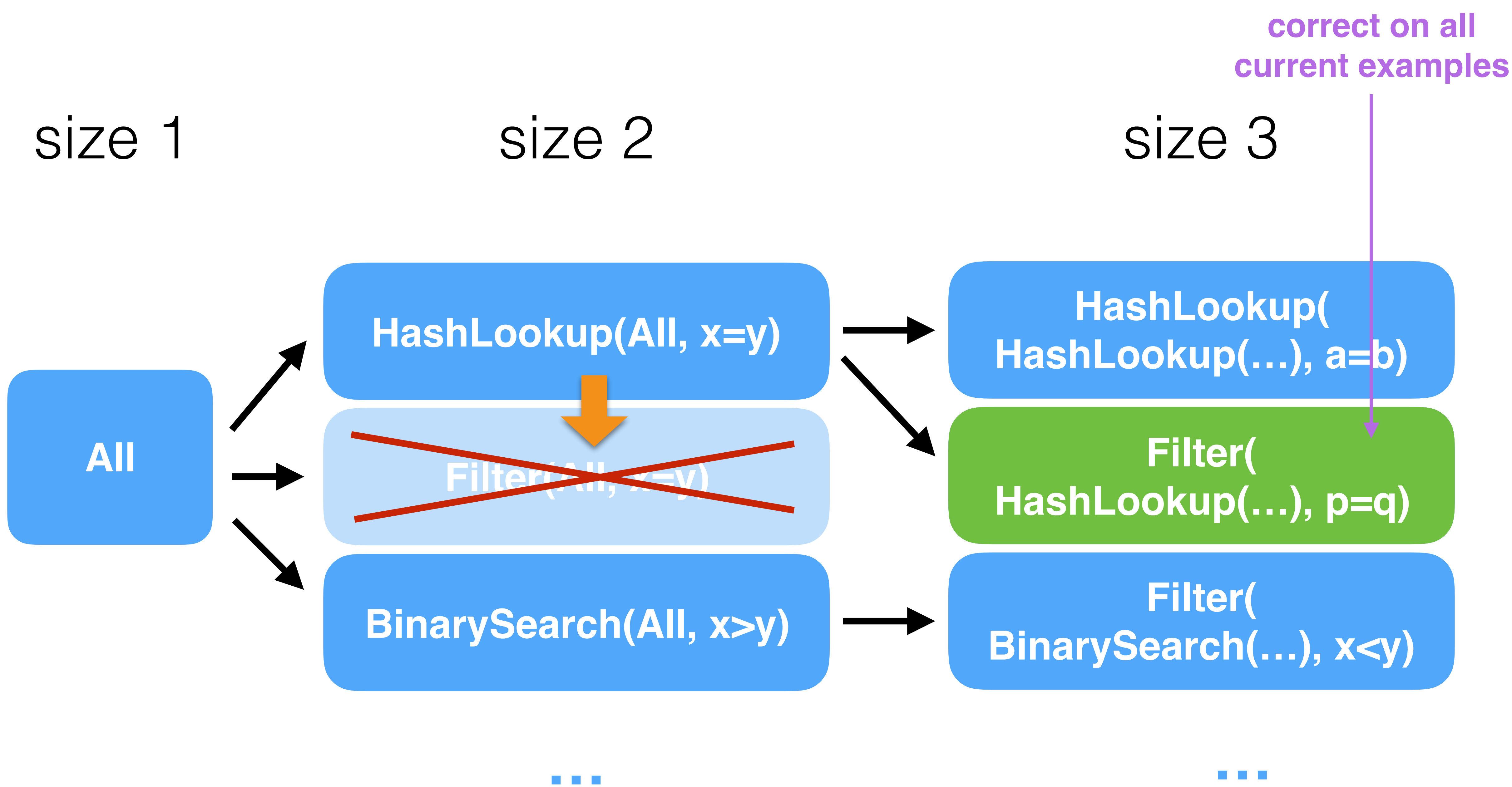
Enumerative search

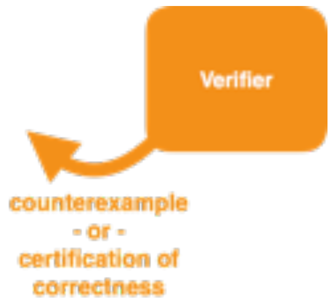




Specification —> Outline

Enumerative search

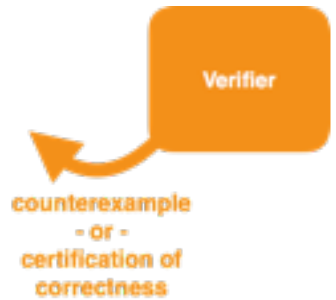




Outline Verification

Specification:

```
Entry has:
   queryId
   subqueryId
   …

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



Outline Verification

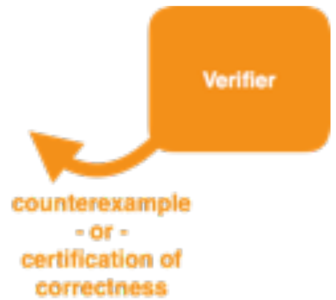
$$\forall I\ \forall S,\quad out = \{\, e \mid e \in S \wedge P(I, e) \,\}$$

S

E

```
    qu
    subquery
    …
```

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



Outline Verification

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

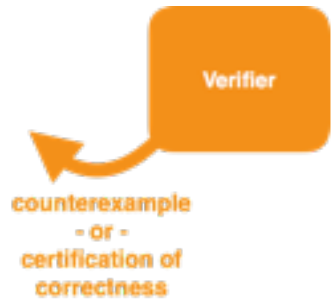
S

E

```
    qu
    subquery.
    …
```

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

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



Outline Verification

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

S

E

qu

subquery.

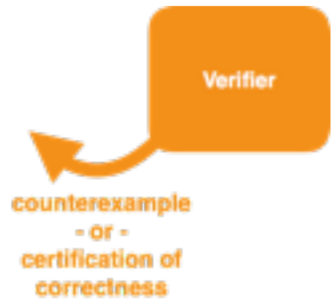
…

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

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

representative predicate

e.queryId = q



Outline Verification

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

S

E

qu

subquery.

…

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

HashLoo

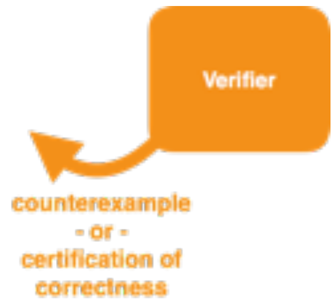
All(),

e.queryId = q

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

representative predicate

e.queryId = q



Outline Verification

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

S

E

```
    qu
    subquery.
    …
```

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

HashLoo
 All(),
 e.queryId = q

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

representative predicate

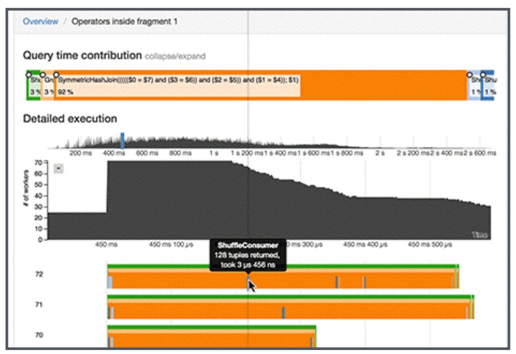
e.queryId = q

check equivalence with an SMT solver

Evaluation

Evaluation

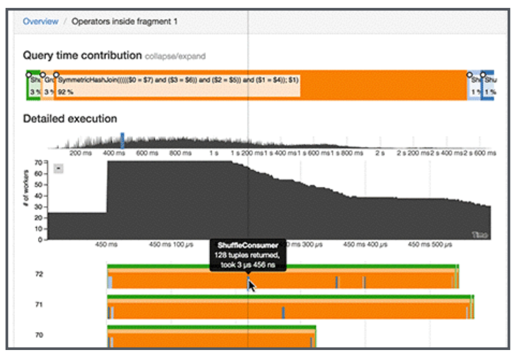
- **Myria:** analytics



Bugs at the SQL/Java interface. Unpredictable query planner.

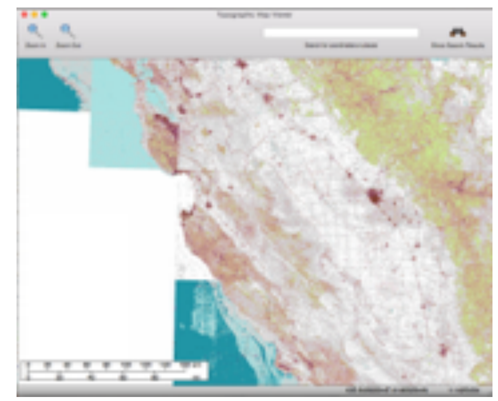
Evaluation

- **Myria:** analytics



Bugs at the SQL/ Java interface. Unpredictable query planner.

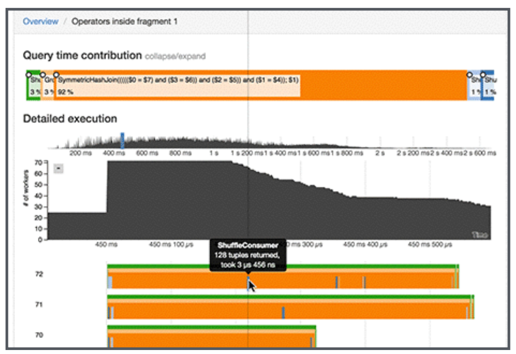
- **ZTopo:** tile cache



Tricky invariant: “state” field on entries reflects its position in the data structure.

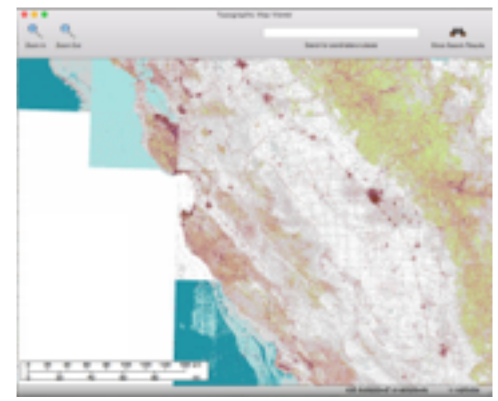
Evaluation

- **Myria:** analytics



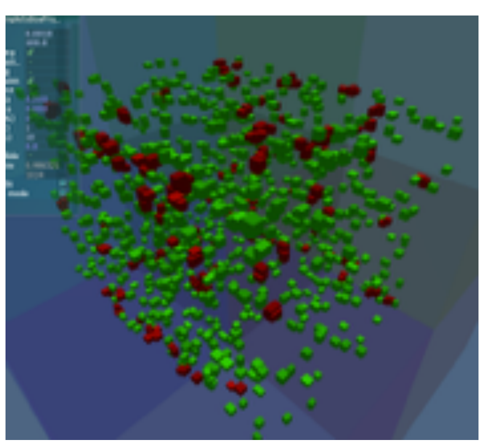
Bugs at the SQL/Java interface. Unpredictable query planner.

- **ZTopo:** tile cache



Tricky invariant: “state” field on entries reflects its position in the data structure.

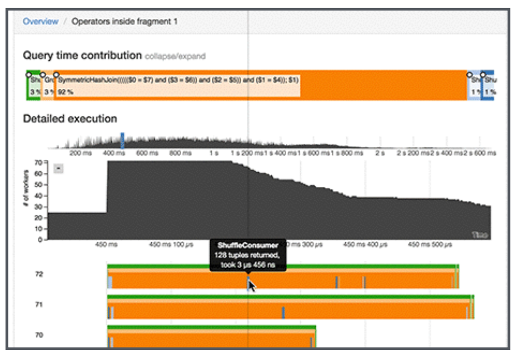
- **Bullet:** volume tree



Lots of verbose, handwritten C++ pointer manipulation code. Custom memory allocator.

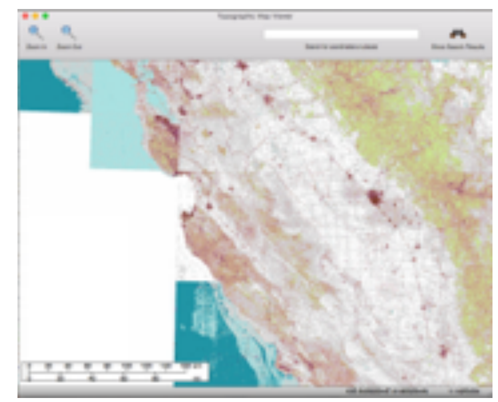
Evaluation

- **Myria:** analytics



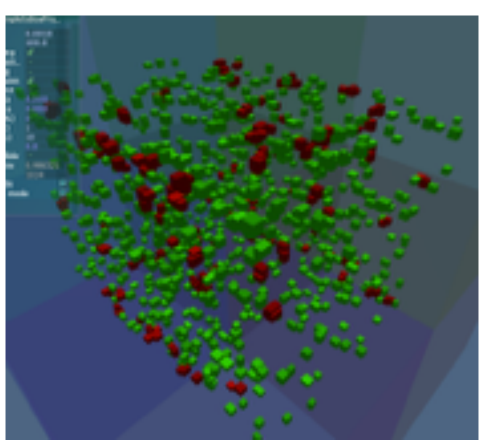
Bugs at the SQL/ Java interface. Unpredictable query planner.

- **ZTopo:** tile cache



Tricky invariant: "state" field on entries reflects its position in the data structure.

- **Bullet:** volume tree



Lots of verbose, handwritten C++ pointer manipulation code. Custom memory allocator.

- **Sat4J:** variable metadata

Custom map implementation for faster lookups

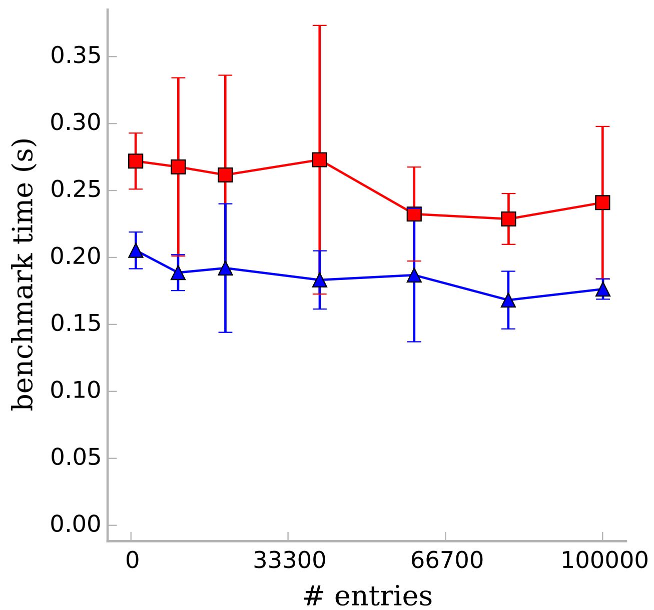
Performance

Original Synthesized

Performance

Original　　Synthesized

Original implementation has worst-case linear time

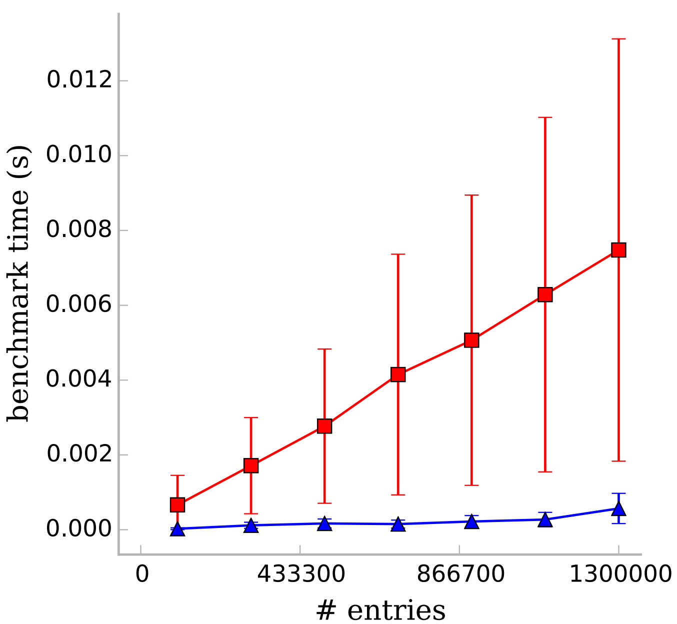


Myria

Performance

Original   Synthesized

Original implementation has worst-case linear time



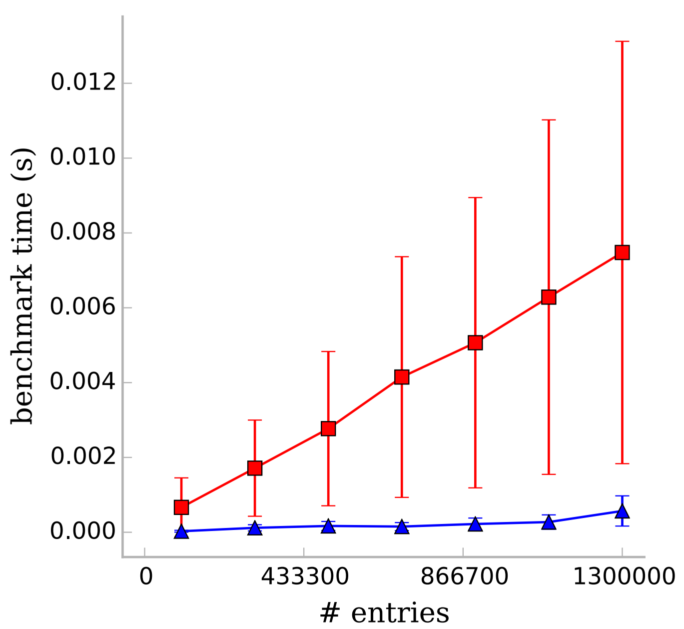
Myria

Performance

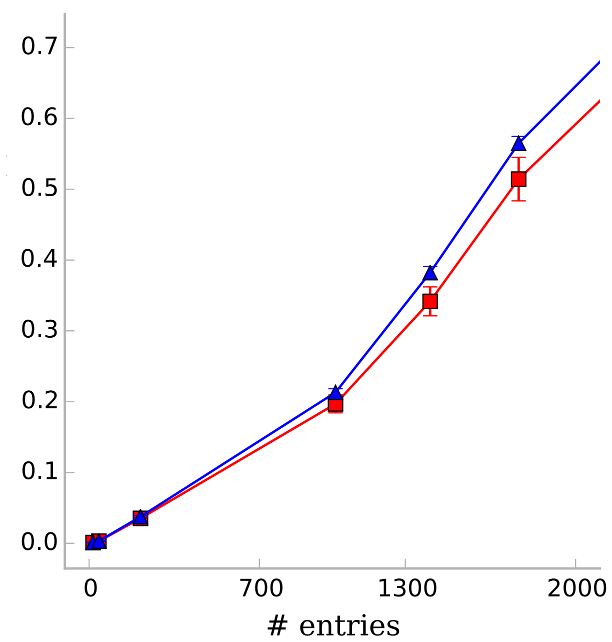
Original Synthesized

Original implementation has worst-case linear time

Data structures are nearly identical



Myria

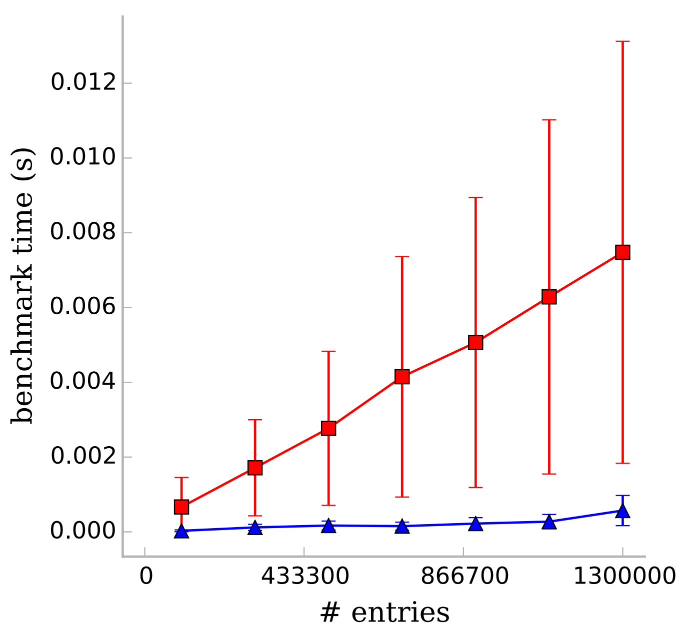


ZTopo

Performance

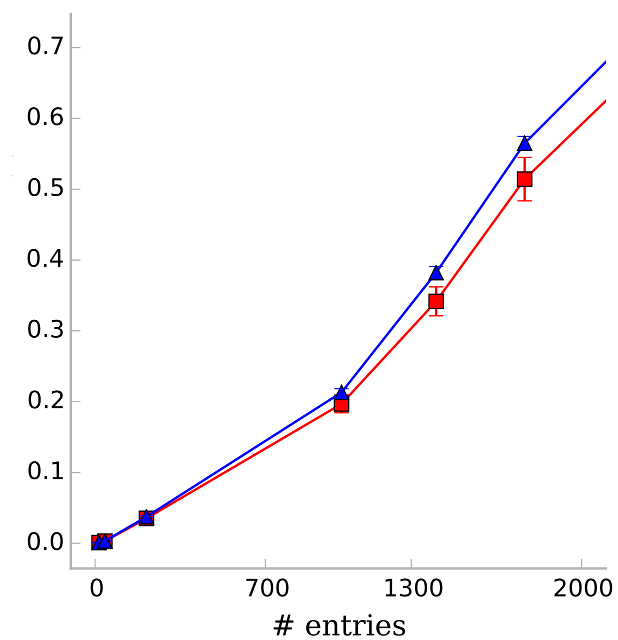
Original Synthesized

Original implementation has worst-case linear time



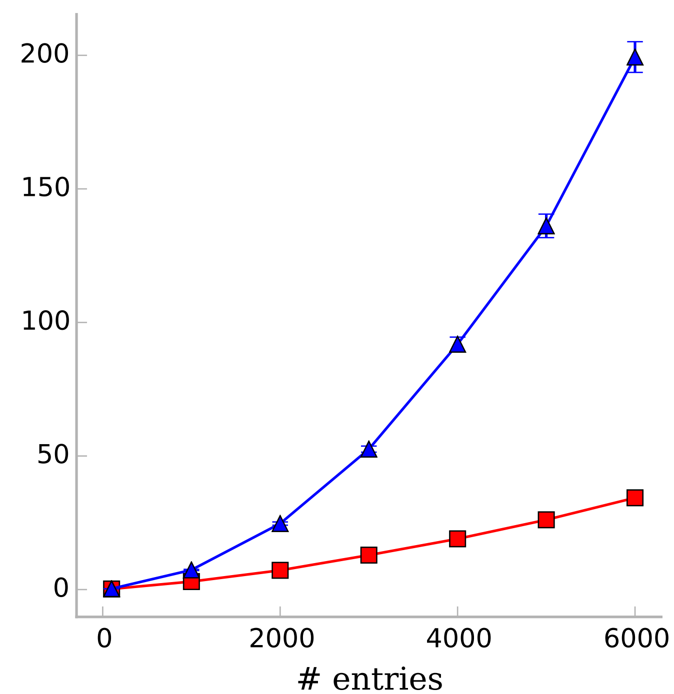
Myria

Data structures are nearly identical



ZTopo

Binary search tree vs. space partitioning tree

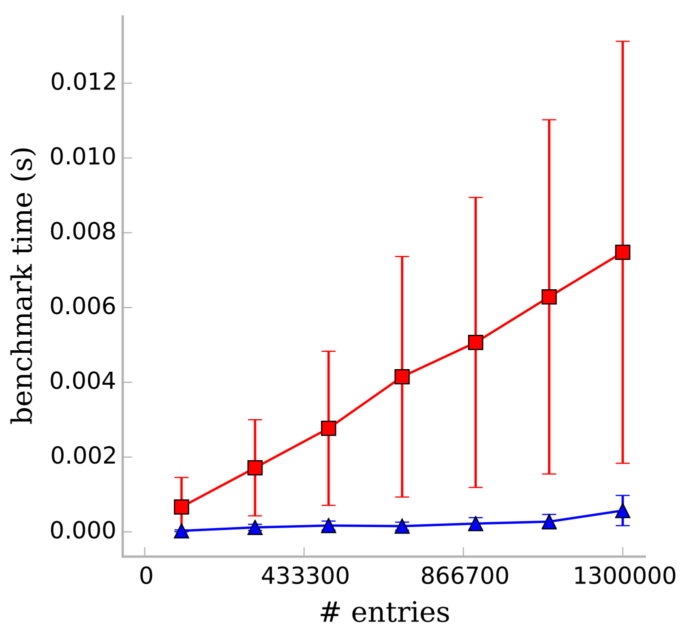


Bullet

Performance

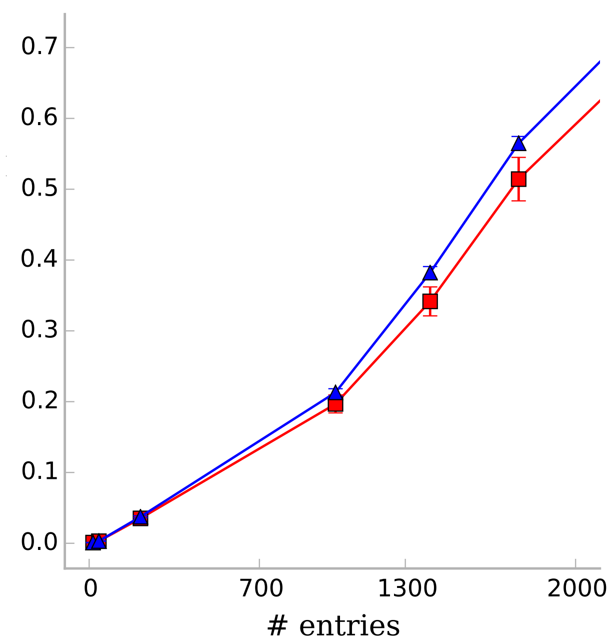
Original   Synthesized

Original implementation has worst-case linear time



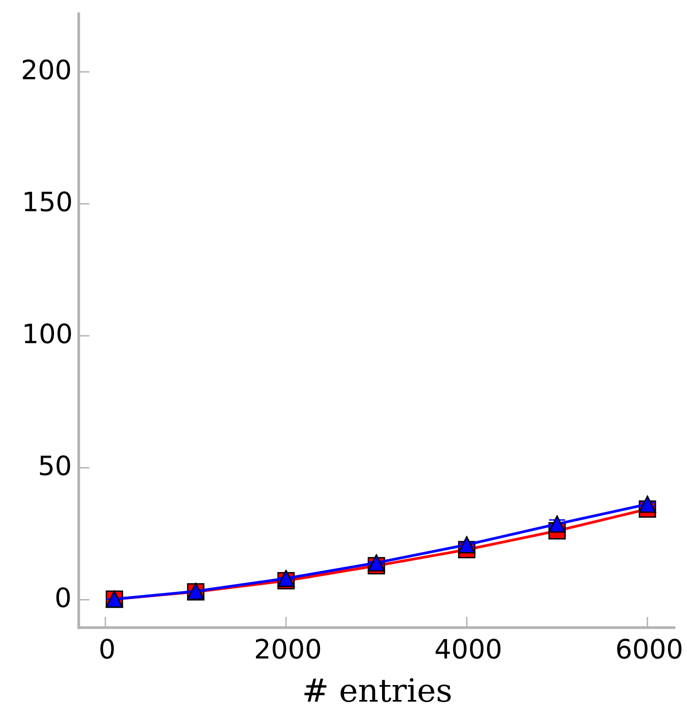
Myria

Data structures are nearly identical



ZTopo

Binary search tree vs. space partitioning tree

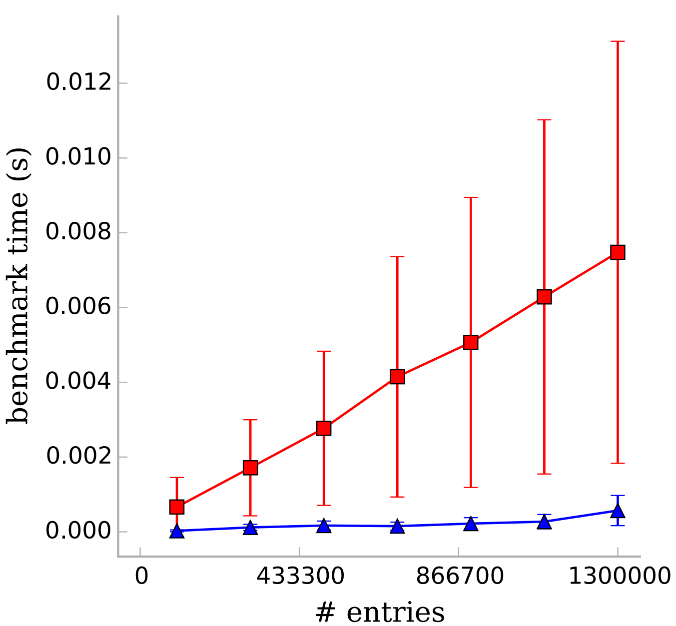


Bullet

Performance

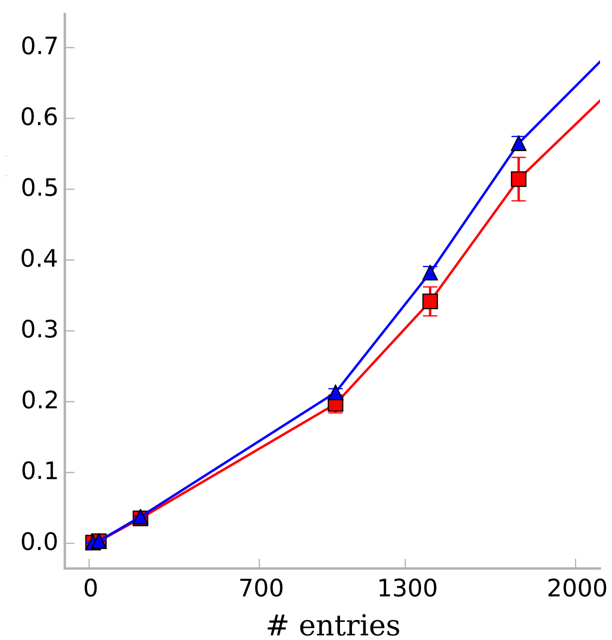
Original   Synthesized

Original implementation has worst-case linear time



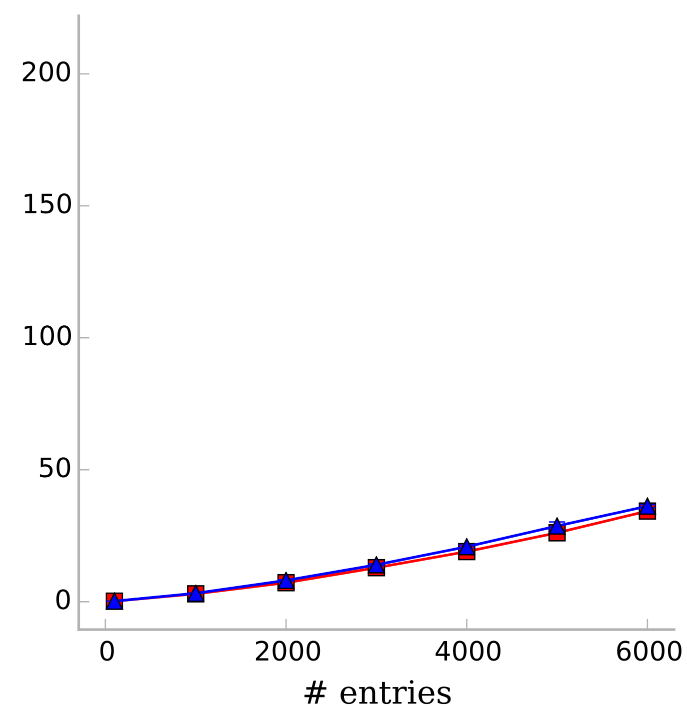
Myria

Data structures are nearly identical



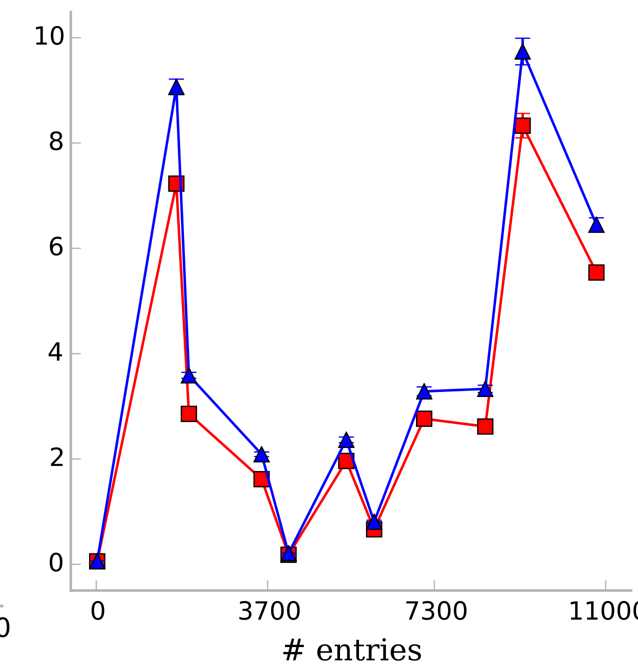
ZTopo

Binary search tree vs. space partitioning tree



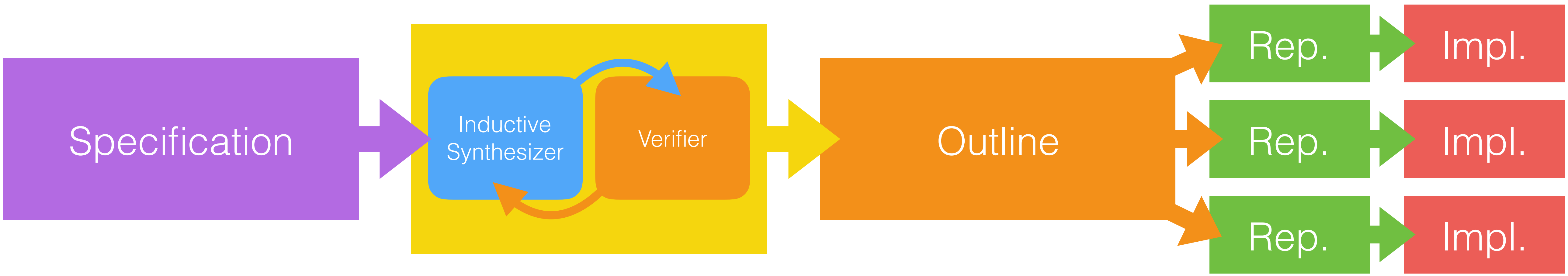
Bullet

Small overhead; performance dominated by other factors



Sat4J

Data Structure Synthesis



- Implementation outlines make the problem tractable

- This approach is extensible

- Cozy generates correct code, and can sometimes surpass handwritten implementations

Special thanks to:

Michael Ernst

Emina Torlak

also Haoming Liu & Daniel Perelman