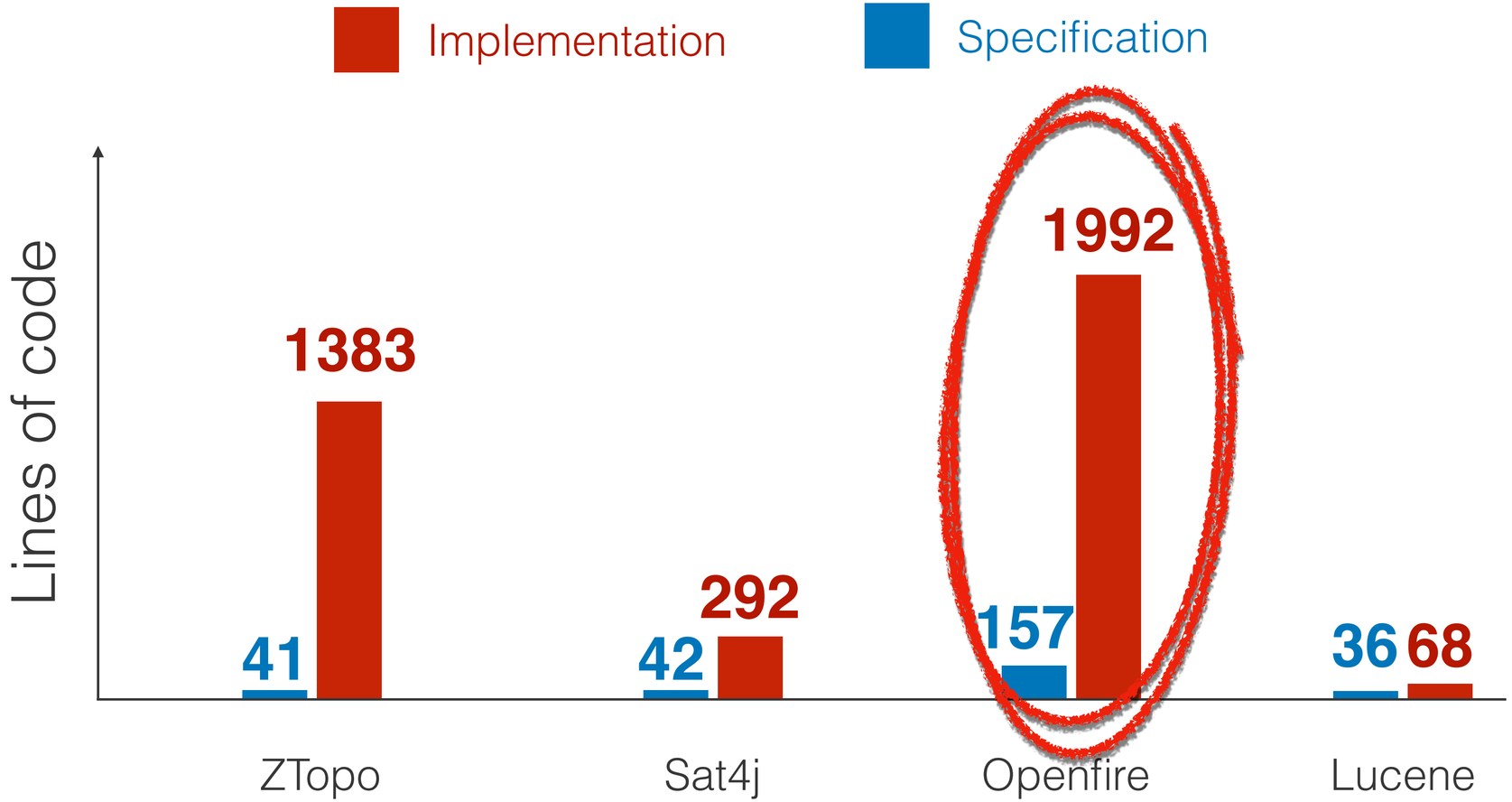
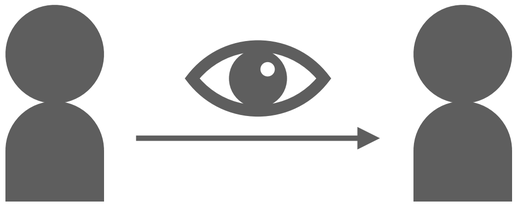
Generalized Data Structure Synthesis

Calvin Loncaric
Michael D. Ernst
Emina Torlak

Stateful modules are much more complicated than their specifications



User Visibility in Openfire



Can Alice see whether Bob is online?

Yes, if:

- Alice “subscribed” to Bob AND Bob approved it
- Bob is a member of a group g that is visible to everyone
- Alice and Bob are in a group together
- Alice is in group g_1, Bob is in group g_2, and g_2 is a “child” of g_1

User visibility in code

```
boolean isRosterItem(JID user) {
    return
        rosterItems.get(user.toBareJID()) != null ||
        implicitFrom.get(user.toBareJID()) != null;
```


```
void groupUserAdded(Group group, ..., JID addedUser) {
```

```
// Get the roster of the added user.
Roster addedUserRoster = null;
if (server.isLocal(addedUser)) {
  addedUserRoster = rosterCache.get(addedUser.getNode());
}

// Iterate on all the affected users and update their rosters
for (JID userToUpdate : users) {
  if (!addedUser.equals(userToUpdate)) {
    // Get the roster to update
    Roster roster = null;
    if (server.isLocal(userToUpdate)) {
      // Check that the user exists, if not then continue with the next user
      try {
        UserManager.getInstance().getUser(userToUpdate.getNode());
      }
      catch (UserNotFoundException e) {
        continue;
      }
      roster = rosterCache.get(userToUpdate.getNode());
    }
    // Only update rosters in memory
    if (roster != null) {
      roster.addSharedUser(group, addedUser);
    }
    // Check if the roster is still not in memory
    if (addedUserRoster == null && server.isLocal(addedUser)) {
      addedUserRoster =
          rosterCache.get(addedUser.getNode());
    }
    // Update the roster of the newly added group user.
    if (addedUserRoster != null) {
      Collection<Group> groups = GroupManager.getInstance().getGroups(userToUpdate);
      addedUserRoster.addSharedUser(userToUpdate, groups, group);
    }
    if (!server.isLocal(addedUser)) {
      // Susbcribe to the presence of the remote user. This is only necessary for
      // remote users and may only work with remote users that
      // accept presence subscription requests
      sendSubscribeRequest(userToUpdate, addedUser, true);
    }
    if (!server.isLocal(userToUpdate)) {
      // Susbcribe to the presence of the remote user. This is only necessary for
      // remote users and may only work with remote users that
      // accept presence subscription requests
```

```
void addSharedUser(Group group, JID addedUser) {
    boolean newItem = false;
    RosterItem item = null;
    try {
      // Get the RosterItem for the *local* user to add
      item = getRosterItem(addedUser);
      // Do nothing if the item already includes the shared group
      if (item.getSharedGroups().contains(group)) {
        return;
      }
      newItem = false;
    } catch (UserNotFoundException e) {
      try {
        // Create a new RosterItem for this new user
        String nickname = UserNameManager.getUserName(addedUser);
        item =
            new RosterItem(addedUser, RosterItem.SUB_BOTH, RosterItem.ASK_NONE,
                RosterItem.RECV_NONE, nickname, null);
        // Add the new item to the list of items
        rosterItems.put(item.getJid().toBareJID(), item);
        newItem = true;
      } catch (UserNotFoundException ex) {
        Log.error("Group (" + group.getName() + ") includes non-existent user (" +
            addedUser +
            ")");
        return;
      }
    }

    // If an item already exists then take note of the old subscription
    RosterItem.SubType prevSubscription = null;
    if (!newItem) {
      prevSubscription = item.getSubStatus();
    }

    // Update the subscription of the item **based on the item groups**
    Collection<Group> userGroups = GroupManager.getInstance().getGroups(addedUser);
    Collection<Group> sharedGroups = new ArrayList<>();
    sharedGroups.addAll(item.getSharedGroups());
    // Add the new group to the list of groups to check
    sharedGroups.add(group);
    // Set subscription type to BOTH if the roster user belongs to a shared group
    // that is mutually visible with a shared group of the new roster item
    if (rosterManager.hasMutualVisibility(getUsername(), userGroups, sharedGroups)) {
      item.setSubStatus(RosterItem.SUB_BOTH);
    }
    // Add the shared group to the list of shared groups of the item
    item.addSharedGroup(group);
```

```
void addSharedUser(JID addedUser, Collec
    boolean newItem = false;
    RosterItem item = null;
    try {
      // Get the RosterItem for the *local* u
      item = getRosterItem(addedUser);
      newItem = false;
    } catch (UserNotFoundException e) {
      try {
        // Create a new RosterItem for this
        String nickname = UserNameManag
            getUserName(addedUser);
        item =
            new RosterItem(addedUser, Ro
                RosterItem.ASK_NONE, n
        // Add the new item to the list of ite
        rosterItems.put(item.getJid().toBa
        newItem = true;
      } catch (UserNotFoundException ex)
        Log.error("Couldn't find a user with
            "as non-existent username (" +
      }
    }
    // Update the subscription of the item **
    Collection<Group> userGroups = GroupM
    // Set subscription type to BOTH if the re
    // that is mutually visible with a shared
    if (rosterManager.hasMutualVisibility(g
      item.setSubStatus(RosterItem.SUB_B
      for (Group group : groups) {
        if (rosterManager.isGroupVisible(gr
          // Add the shared group to the list
          item.addSharedGroup(group);
        }
      }
      // Add to the item the groups of this us
      // Note: This FROM subscription is ove
      // fact there is a TO-FROM relation bet
      // BOTH subscription
      for (Group sharedGroup : groups) {
        if (rosterManager.isGroupVisible(sh
          // Add the shared user to the list
```

User visibility in code

boolean isRosterItem(JID user)
  return
    rosterItems.get(user.toBareJ
    implicitFrom.get(user.toBare

Yes, if:

Alice “subscribed” to Bob AND Bob approved it

Bob is a member of a group g that is visible to everyone

Alice and Bob are in a group together

Alice is in group g_1, Bob is in group g_2, and g_2 is a “child” of g_1

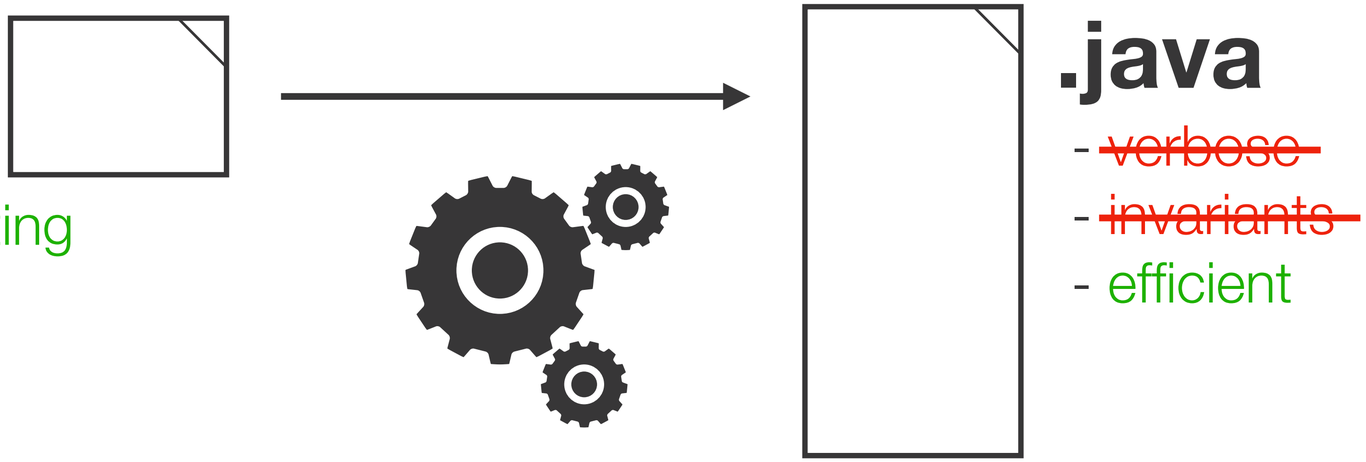
```
query hasSubscriptionTo(u1 : User, u2 : User)

  exists [ i | i <- rosterItems,
     u1.val.username == i.val.user,
     jidToUsername(i.val.target) == u2.val.username ]

  or exists [ g | g <- groups,
        (g, u2) in groupMembers and
        (g.val.rosterMode == EVERYBODY or
          (g.val.rosterMode == ONLY_GROUP and (g, u1) in groupMembers) or
          (g.val.rosterMode == ONLY_GROUP and
            exists [ subg | subg <- groups,
              (u1, subg) in groupMembers,
               (g, subg) in childGroups ])) ]
```

```
op addMember(g : Group, u : User)
  groupMembers.add((g, u));
```

Cozy

Spec

- short
- self-documenting
- ~~inefficient~~



.java

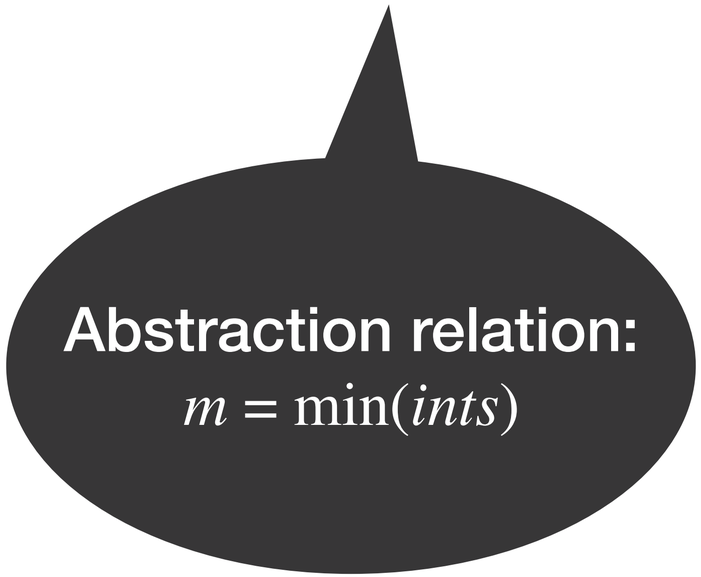
- ~~verbose~~
- ~~invariants~~
- efficient

Cozy

state **ints** : Bag<Int>

op add(**i** : Int)
  ints.add(**i**)

query findmin()
  return (min **ints**)



state **m** : Int

op add(**i** : Int)
  m = min(**m**, **i**)

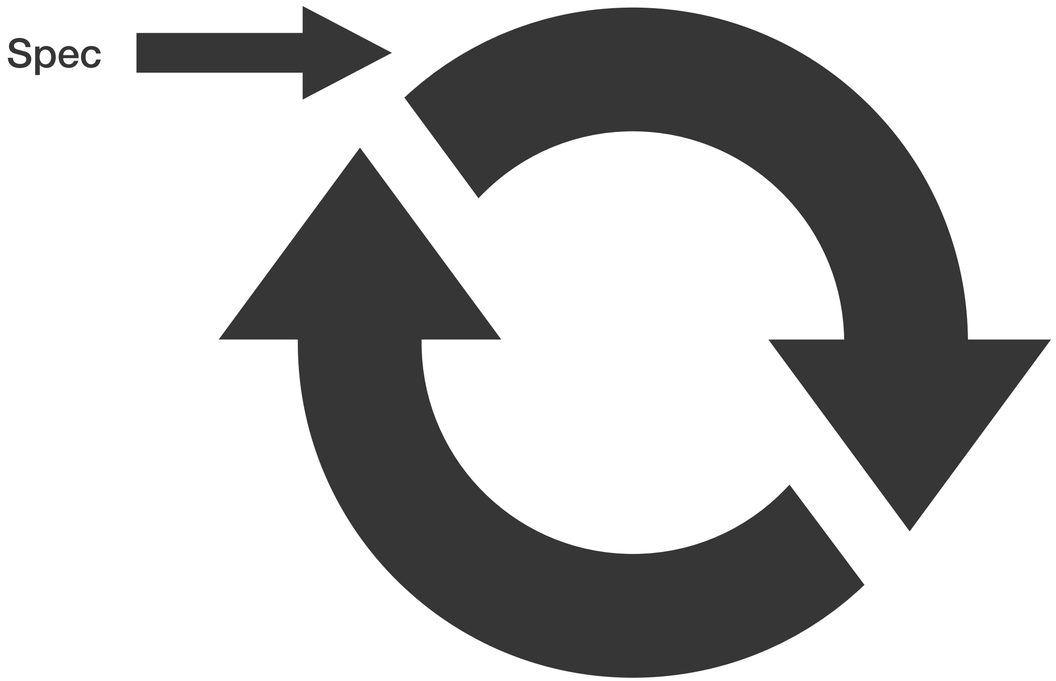
query findmin()
  return **m**

Cozy

```
state ints : Bag<Int>

op add(i : Int)
   ints.add(i)

query findmin()
   return (min ints)
```

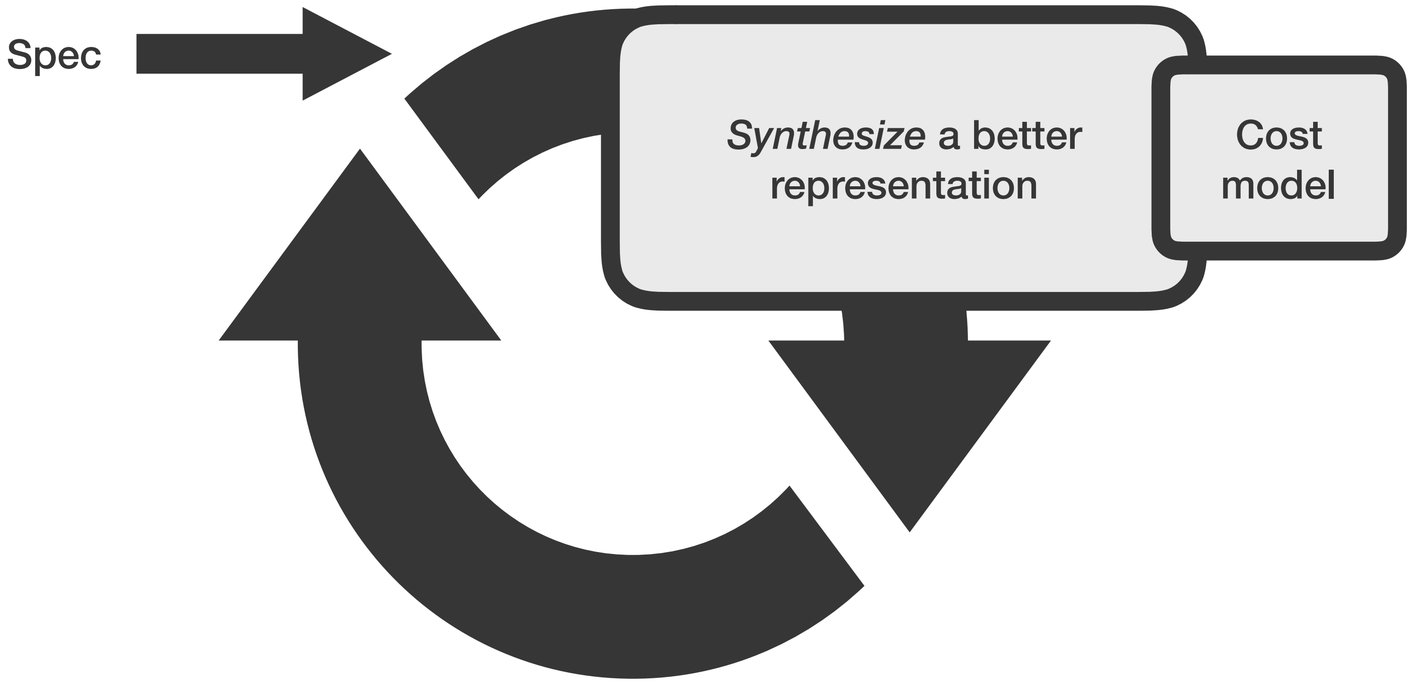
Cozy

```
state ints : Bag<Int>

op add(i : Int)
   ints.add(i)

query findmin()
   return (min ints)
```



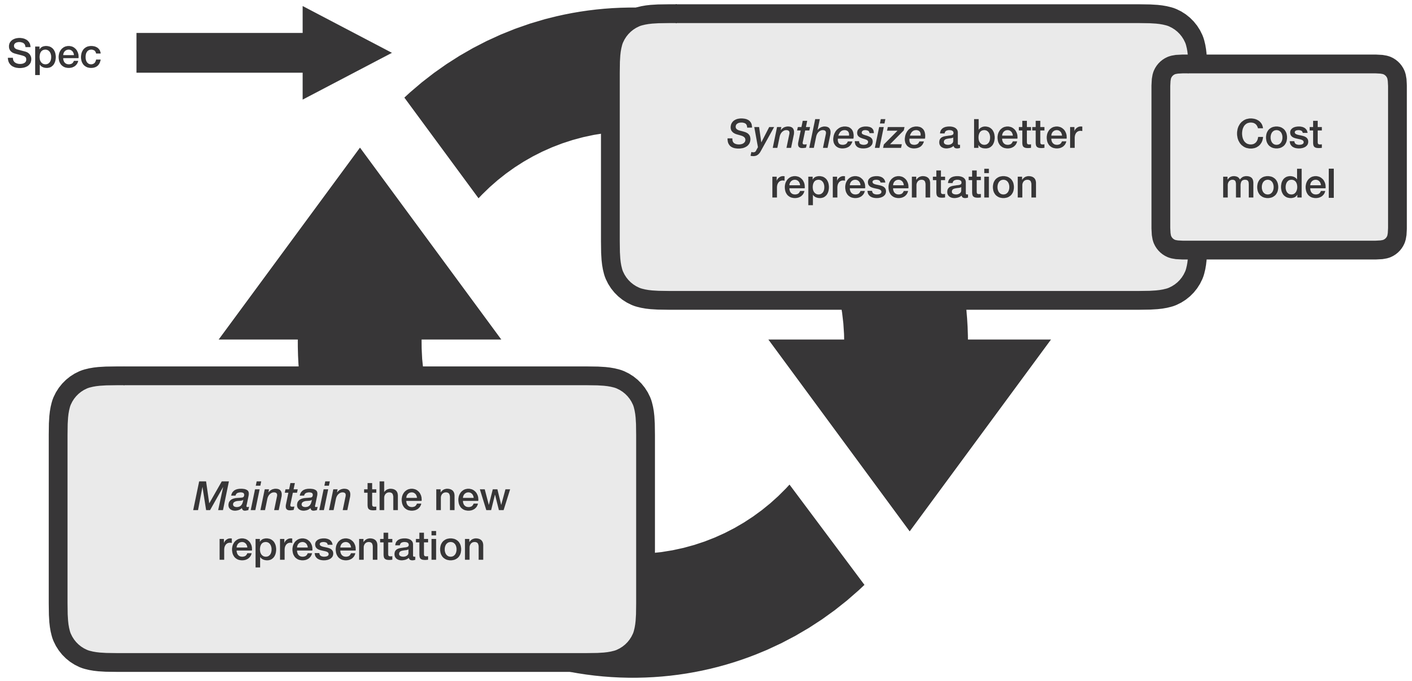
Cozy

```
state ints : Bag<Int>
state m : Int

op add(i : Int)
   ints.add(i)


query findmin()
   return m
```



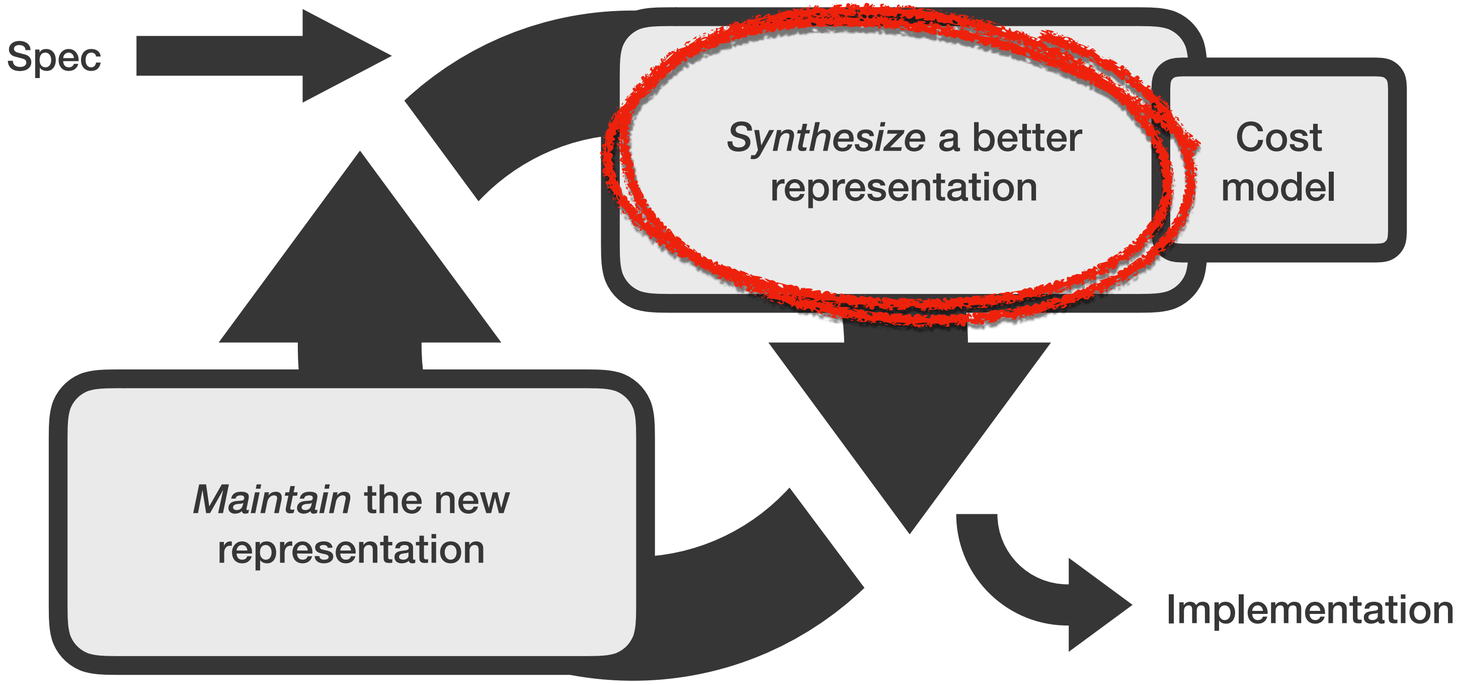
Cozy

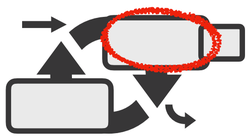
```
state ints : Bag<Int>
state m : Int

op add(i : Int)
   ints.add(i)
   m = min(m, i)

query findmin()
   return m
```



Cozy

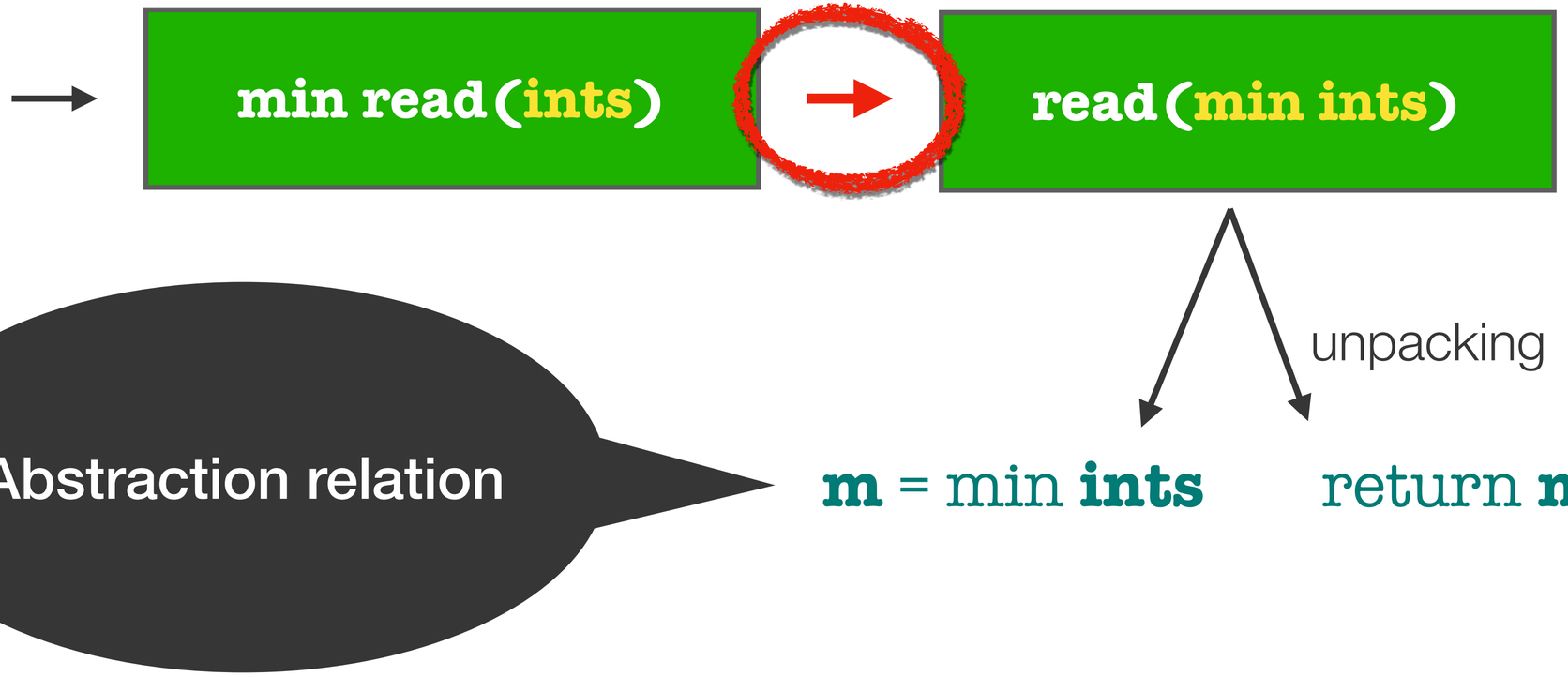
```
state m : Int

op add(i : Int)
    m = min(m, i)

query findmin()
    return m
```

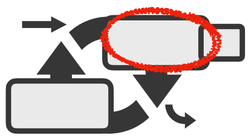




Query Synthesis

query findmin()
  return (min **ints**)





Smart Brute Force Search

skip semantically-identical exps

memoize

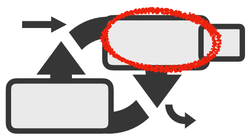
bias

```
for size in [1, 2, ...]:
  for exp in all_expressions(size):
    if correct(exp):
      return exp
```

bounded verification

yield and keep searching

“improves cost” is part of correctness



Cozy is biased toward common patterns

for **size** in [1, 2, ...]:

```
x + 1
```

AST size **3**

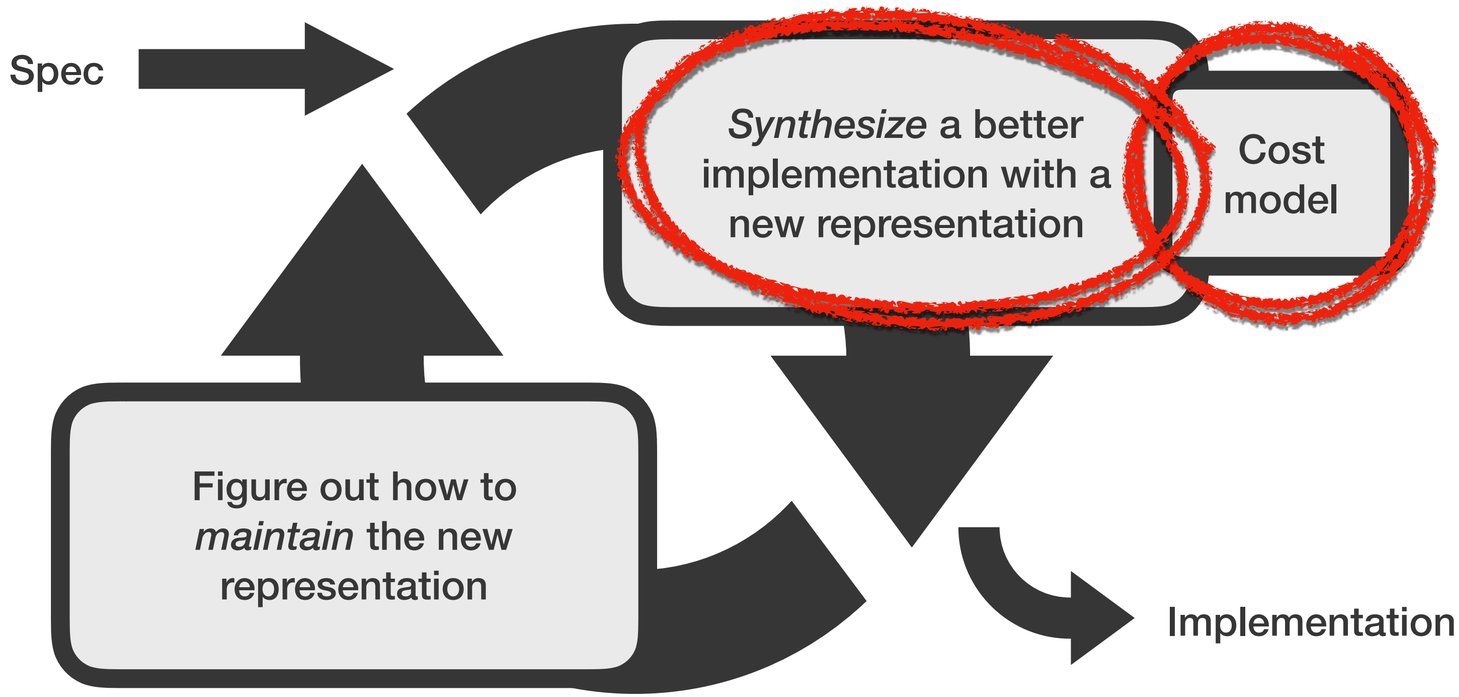
Discovered when size = **3**

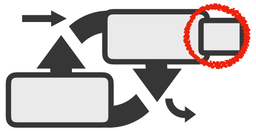
```
MakeMap(
  user_ids,
  λu . the {x|
   x ∈ users, x.id=u}
 )[user_id]
```

AST size **13**

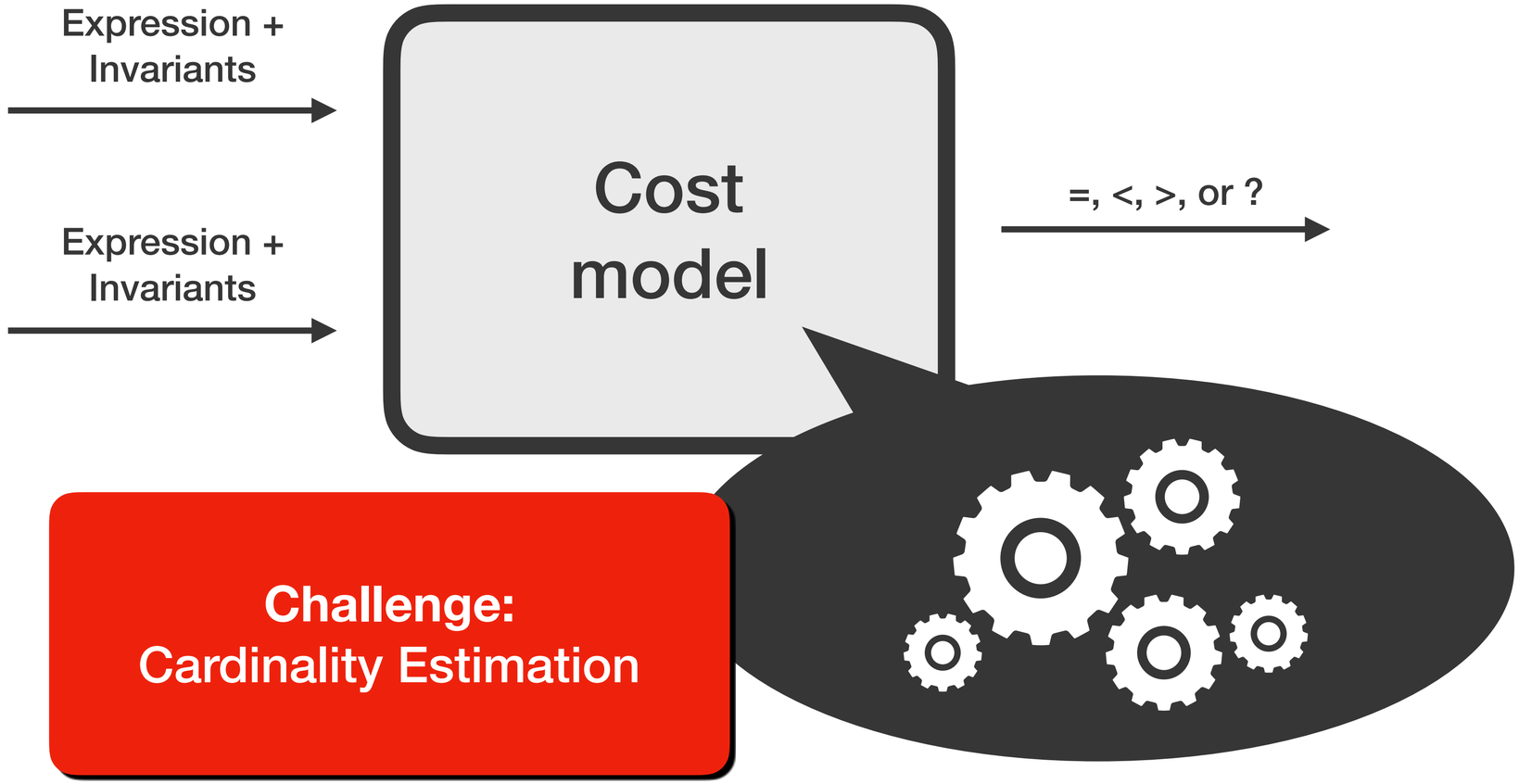
Discovered when size = **2**

Iterative Discovery

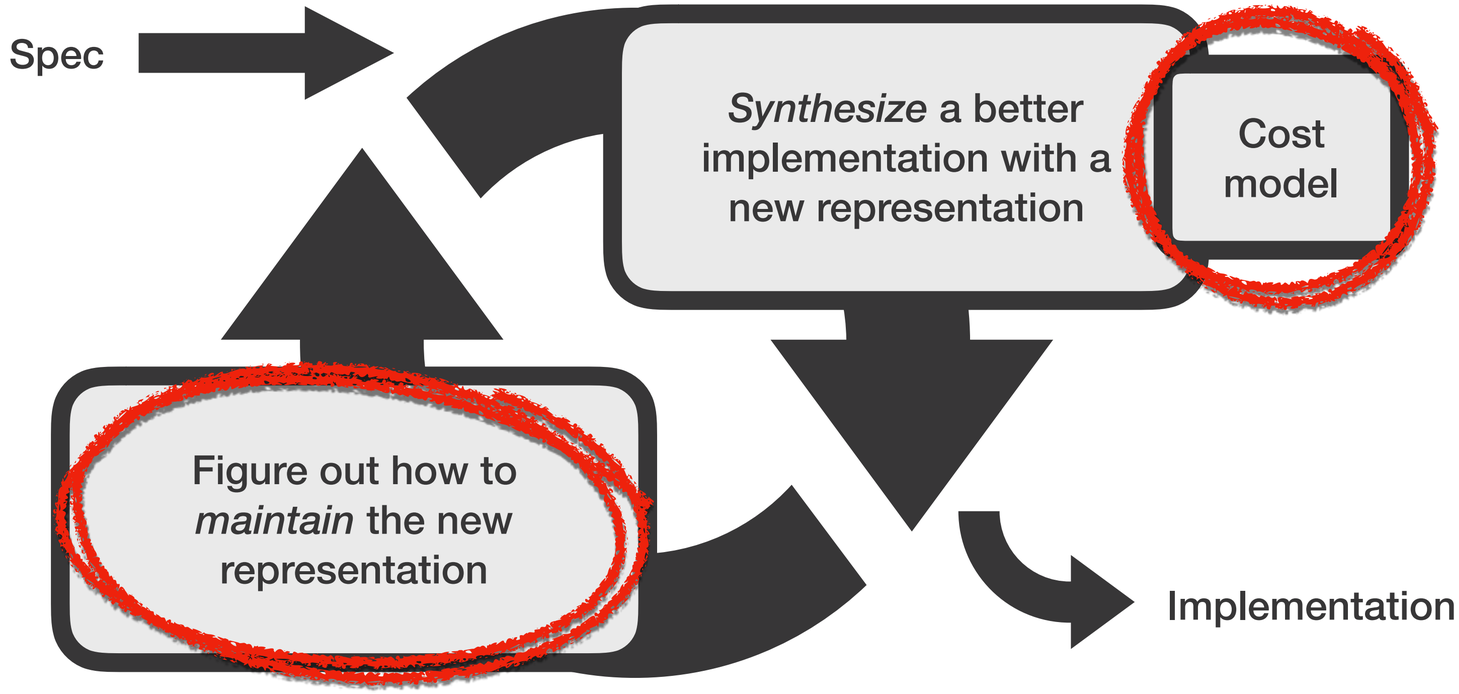


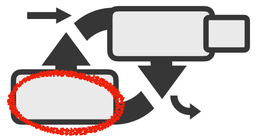


Cost Optimization



Iterative Discovery



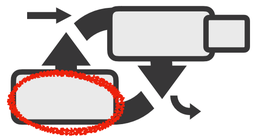


State Maintenance

```
state ints : Bag<Int>
state m : Int

op add(i : Int)
   ints.add(i)
   m =

query findmin()
   return m
```

???

Idea: leverage our incredible query synthesizer

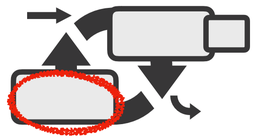


State Maintenance

```
state ints : Bag<Int>
state m : Int = min ints

op add(i : Int)
   ints.add(i)
   m = new_min(i)

query findmin()
   return m

query new_min(i)
   return (min (ints ∪ {i}))
```

Idea: leverage our incredible query synthesizer

Leave optimization to later iterations



State Maintenance

```
state ints : Bag<Int>
state m : Int

op add(i : Int)
   ints.add(i)
   m = new_min(i)

query findmin()
   return m

query new_min(i)
   return (min (ints ∪ {i}))
```

Idea: leverage our incredible query synthesizer

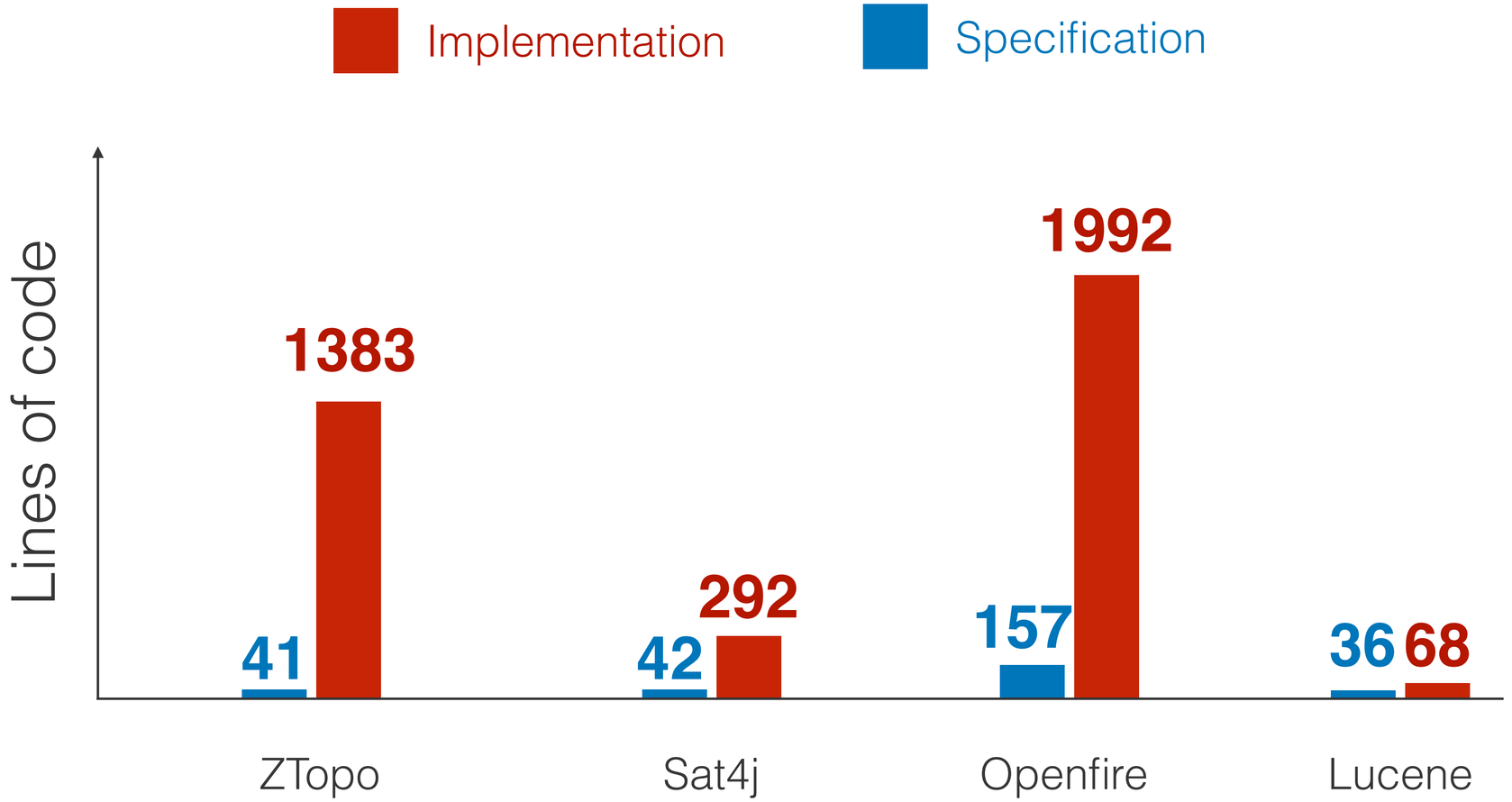
Case Studies

Goals

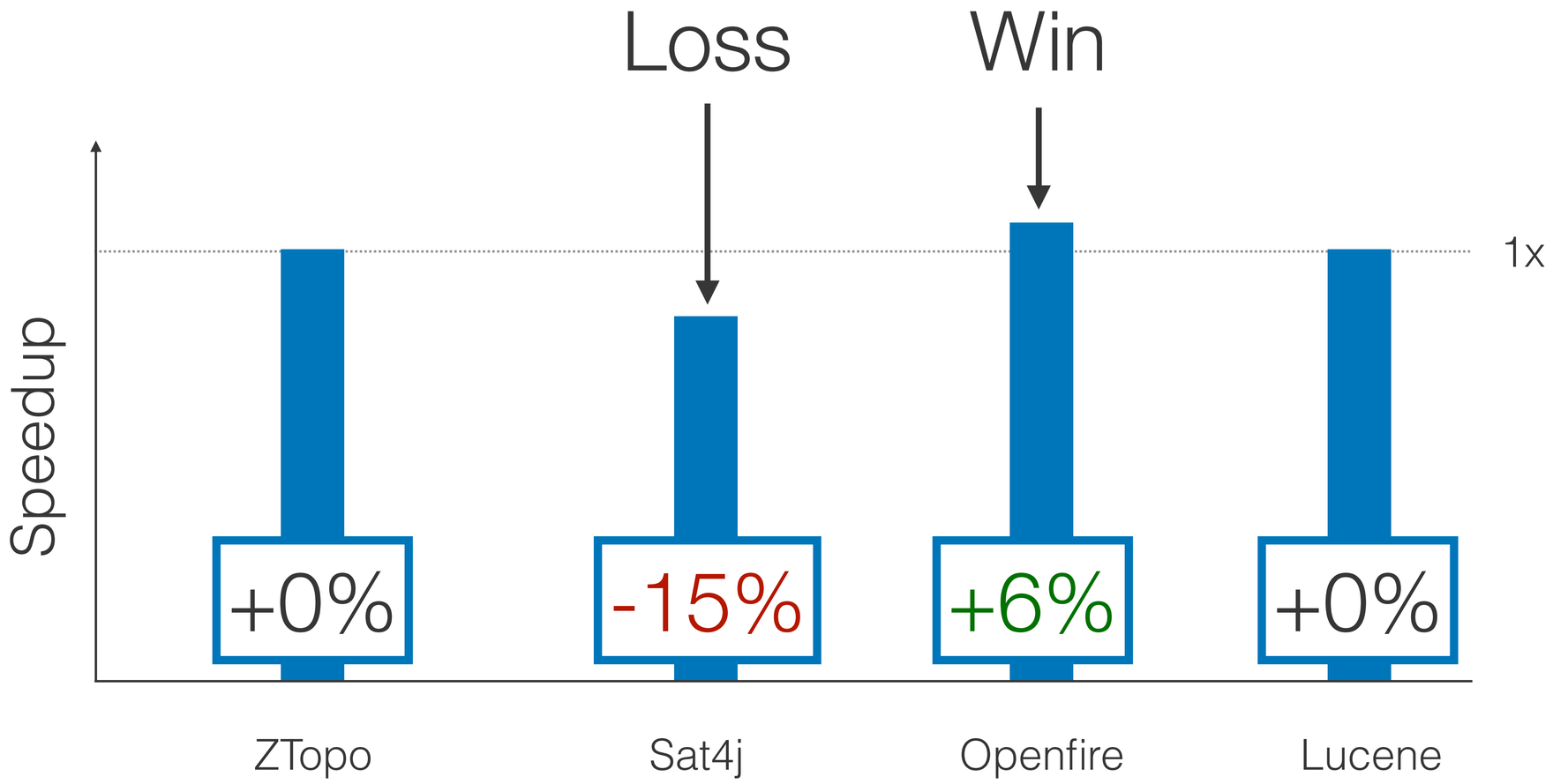
- Less effort
- Same performance
- No new bugs

ZTopo	Map tile cache
Sat4j	Internal metadata
Openfire	Visibility enforcement
Lucene	Streaming document statistics

Effort


Implementation
Specification
Lines of code
1383
41
292
42
1992
157
36
68
ZTopo
Sat4j
Openfire
Lucene

Performance



Correctness

	Handwritten		Cozy
ZTopo	?	→	?
Sat4j	7	→	0
Openfire	25	→	0
Lucene	1	→	0

Related Work

Iterator Inversion (1975)
- Rewrite rules

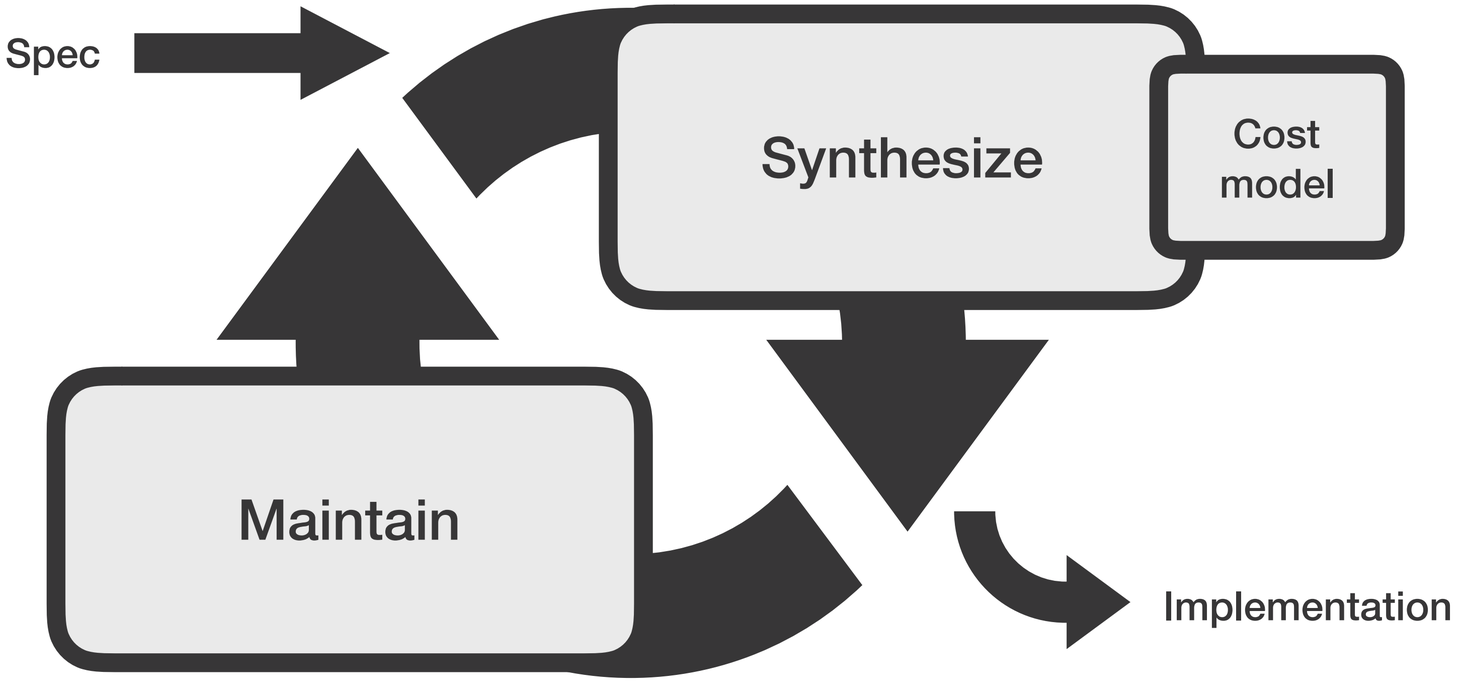
SETL (1975—)
- Manual separation of algorithm and data arrangement

Programming by Refinement (1990—)
- Manual iterative transformation

Representation Synthesis (2011)
- Limited specification language, special-purpose techniques

Cozy 1.0 (2016)
- Limited specification language, special-purpose techniques

Data Structure Synthesis



Lower effort

Equal performance

Fewer bugs

Acknowledgements
Professors Mike and Emina
Students Daniel, David, and Haoming

https://cozy.uwplse.org