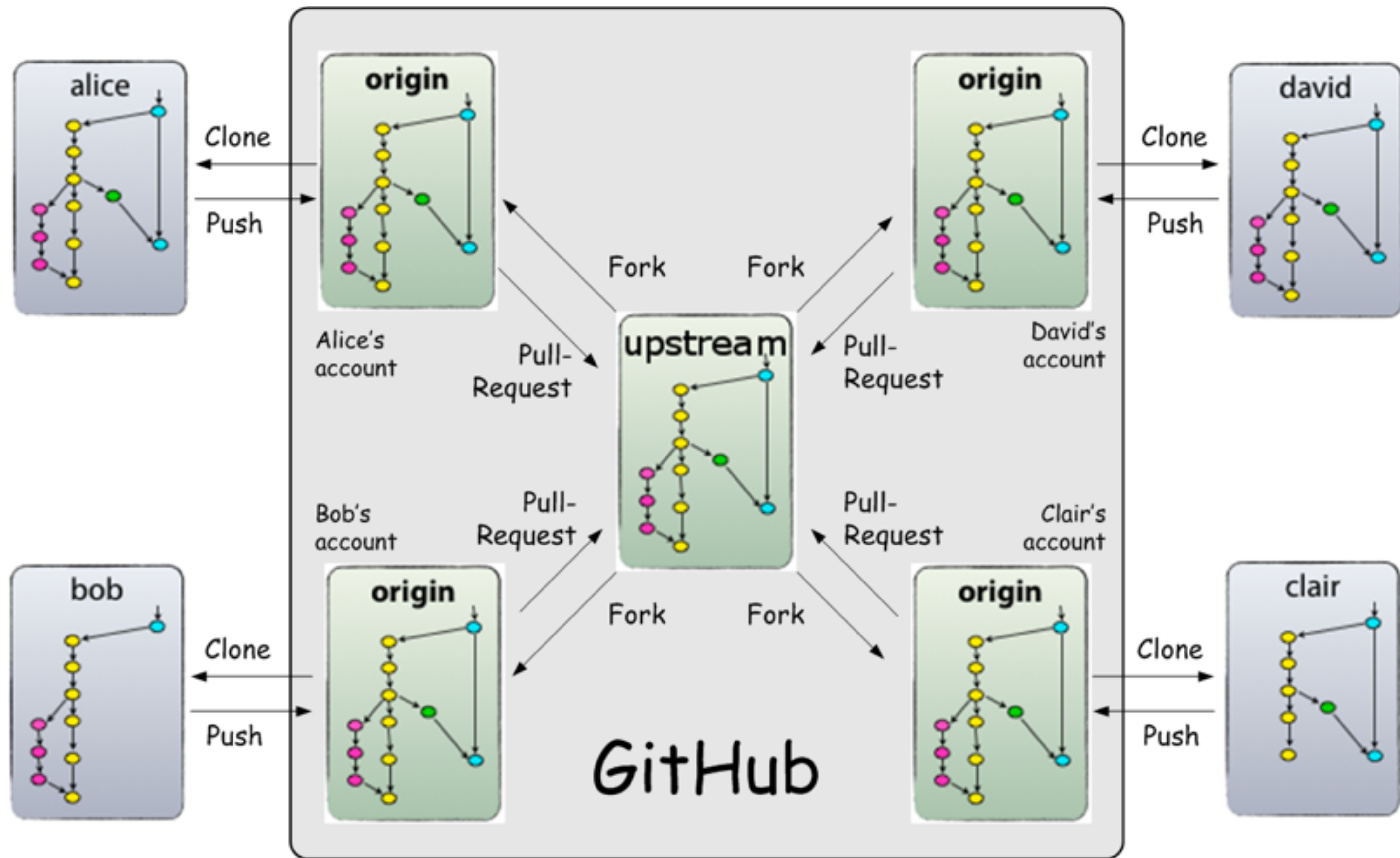
Precise Semantic History Slicing through Dynamic Delta Refinement

Yi Li / UToronto
Chenguang Zhu / UToronto
Julia Rubin / MIT
Marsha Chechik / UToronto

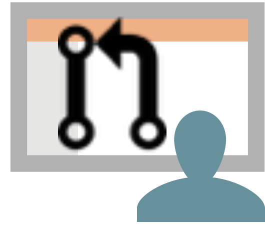
Sep 6, 2016

Fork & Pull Model



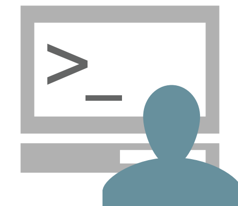
Source: http://www.dalescott.net/using-gitflow-with-githubs-fork-pull-model/

Lifetime of a Pull Request


pull request

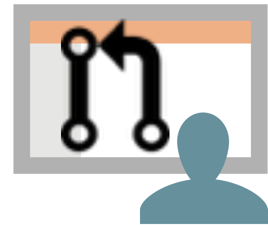
central repo

local repo



local file system

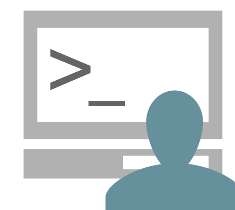
Lifetime of a Pull Request



pull request

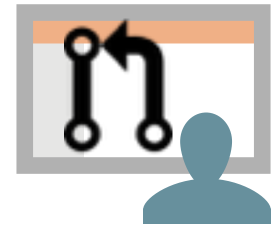
central repo

local repo



local file system

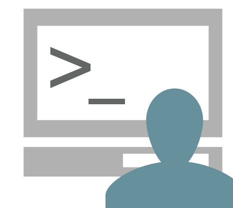
Lifetime of a Pull Request



pull request

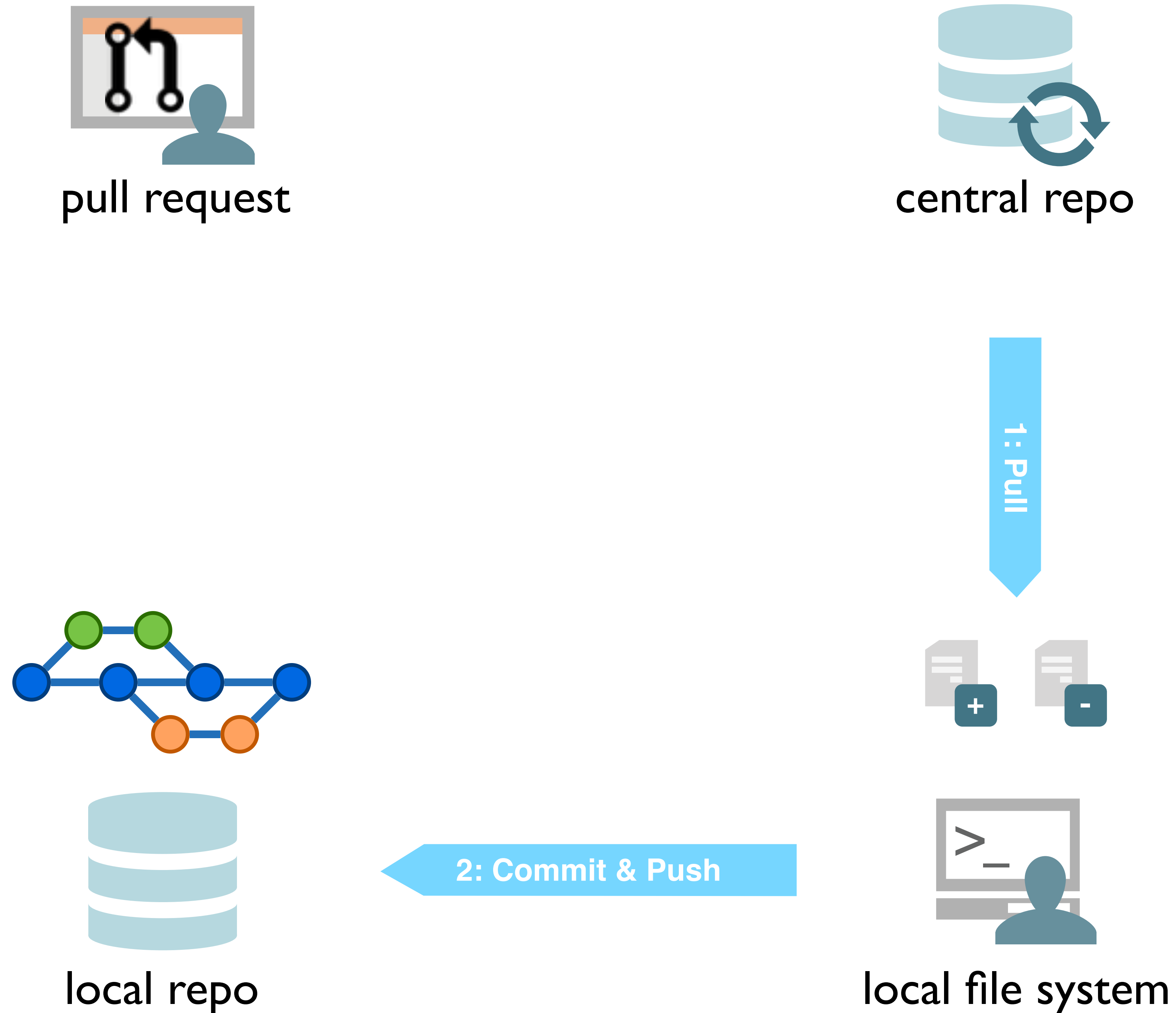
central repo

local repo

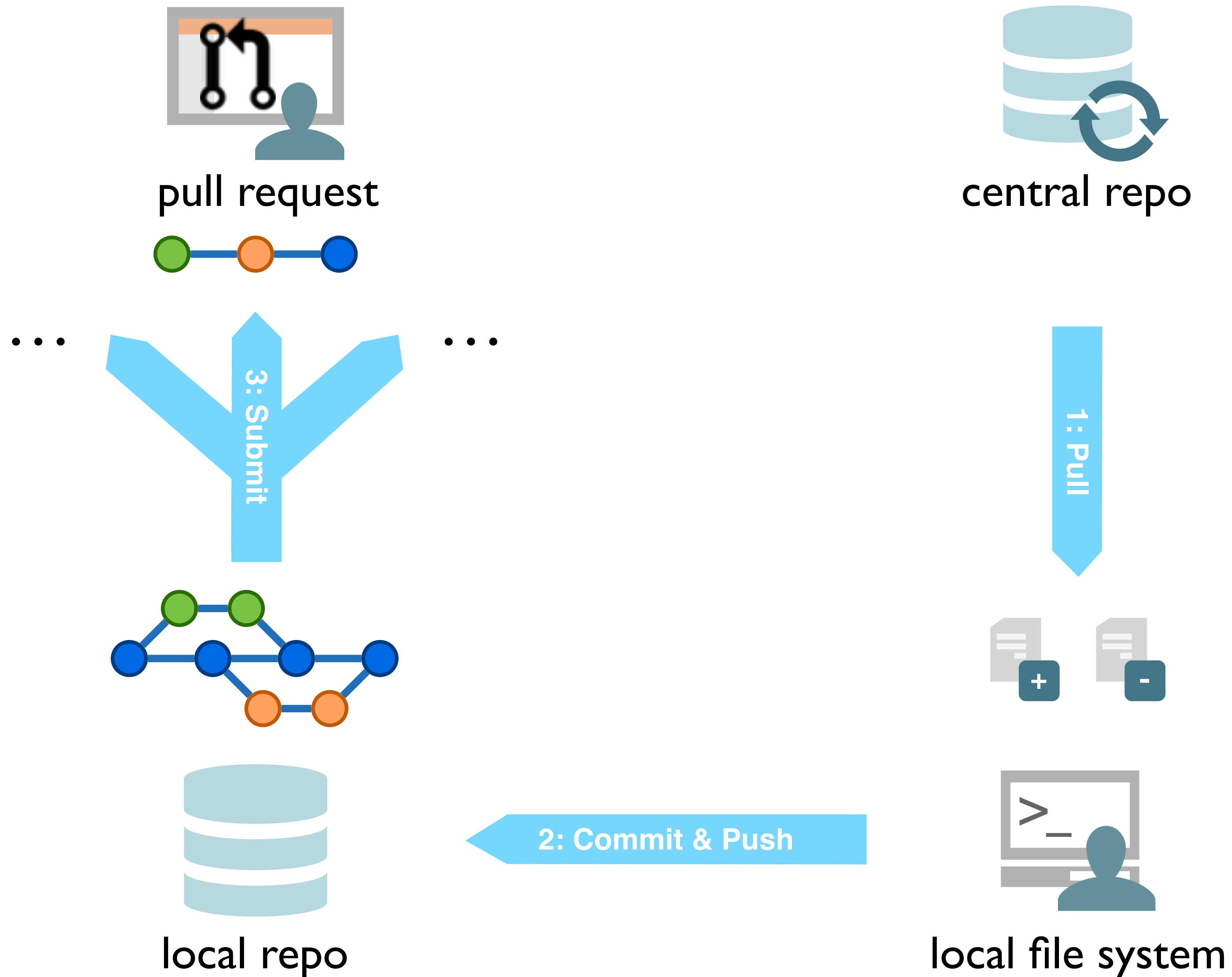


local file system

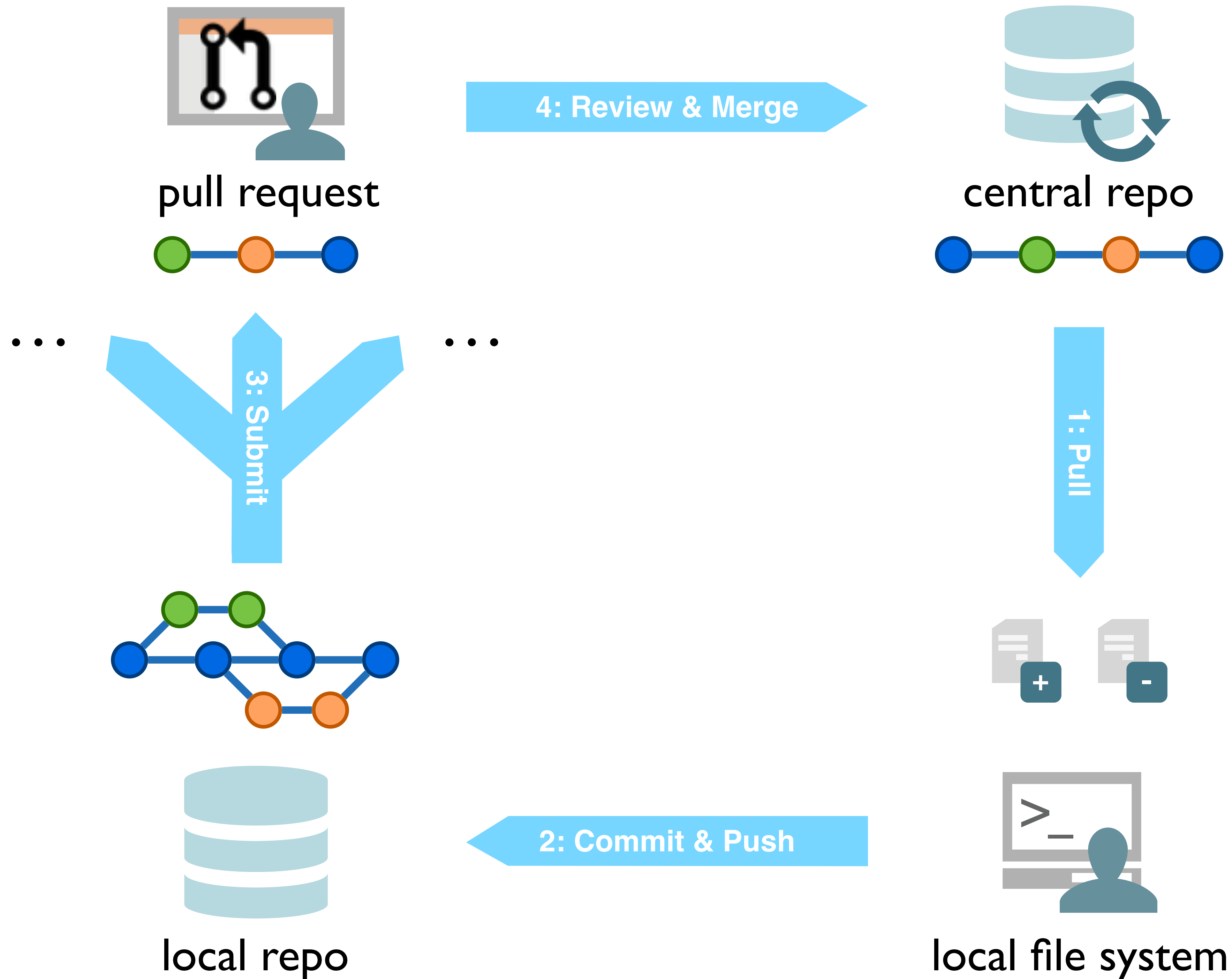
Lifetime of a Pull Request



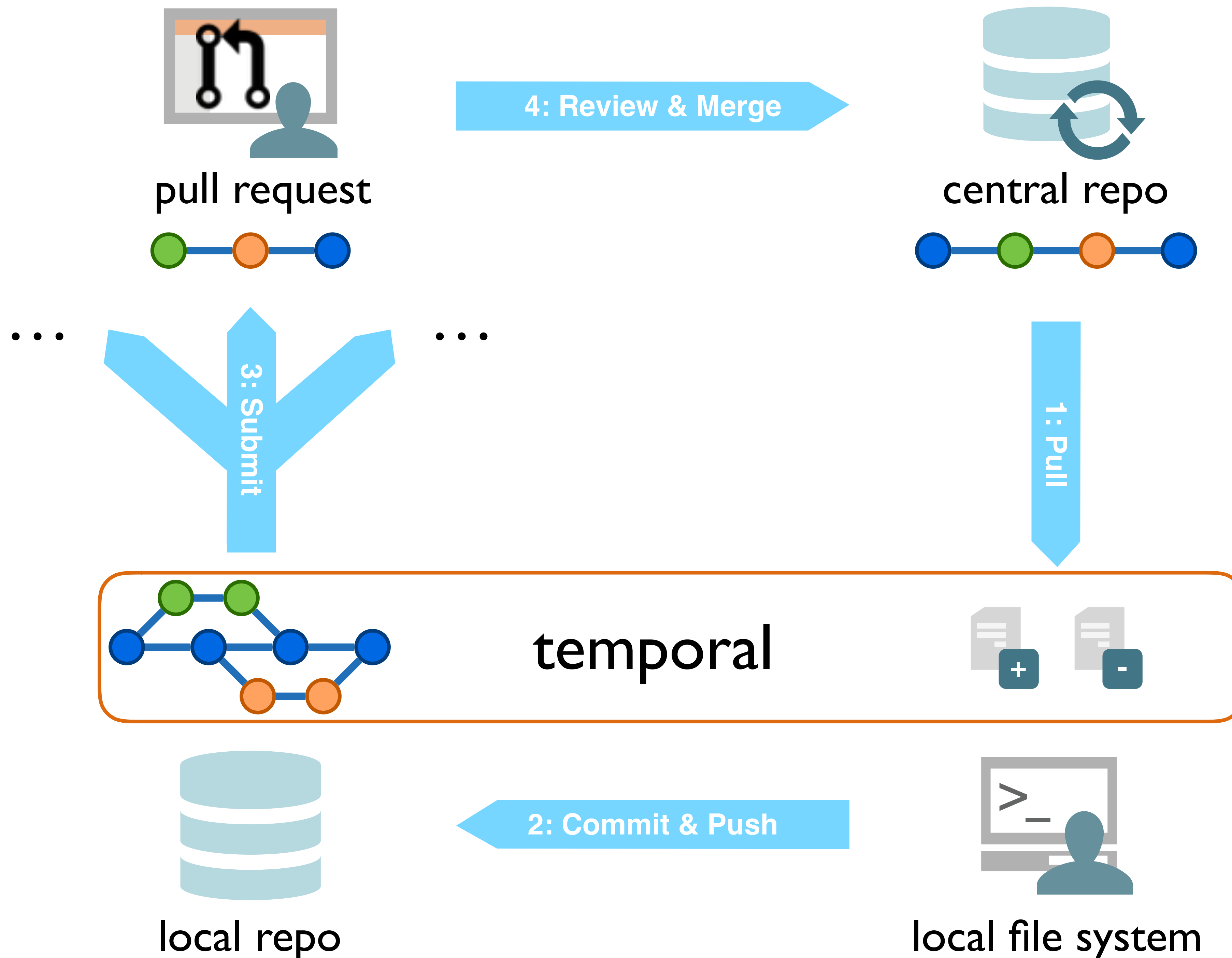
Lifetime of a Pull Request



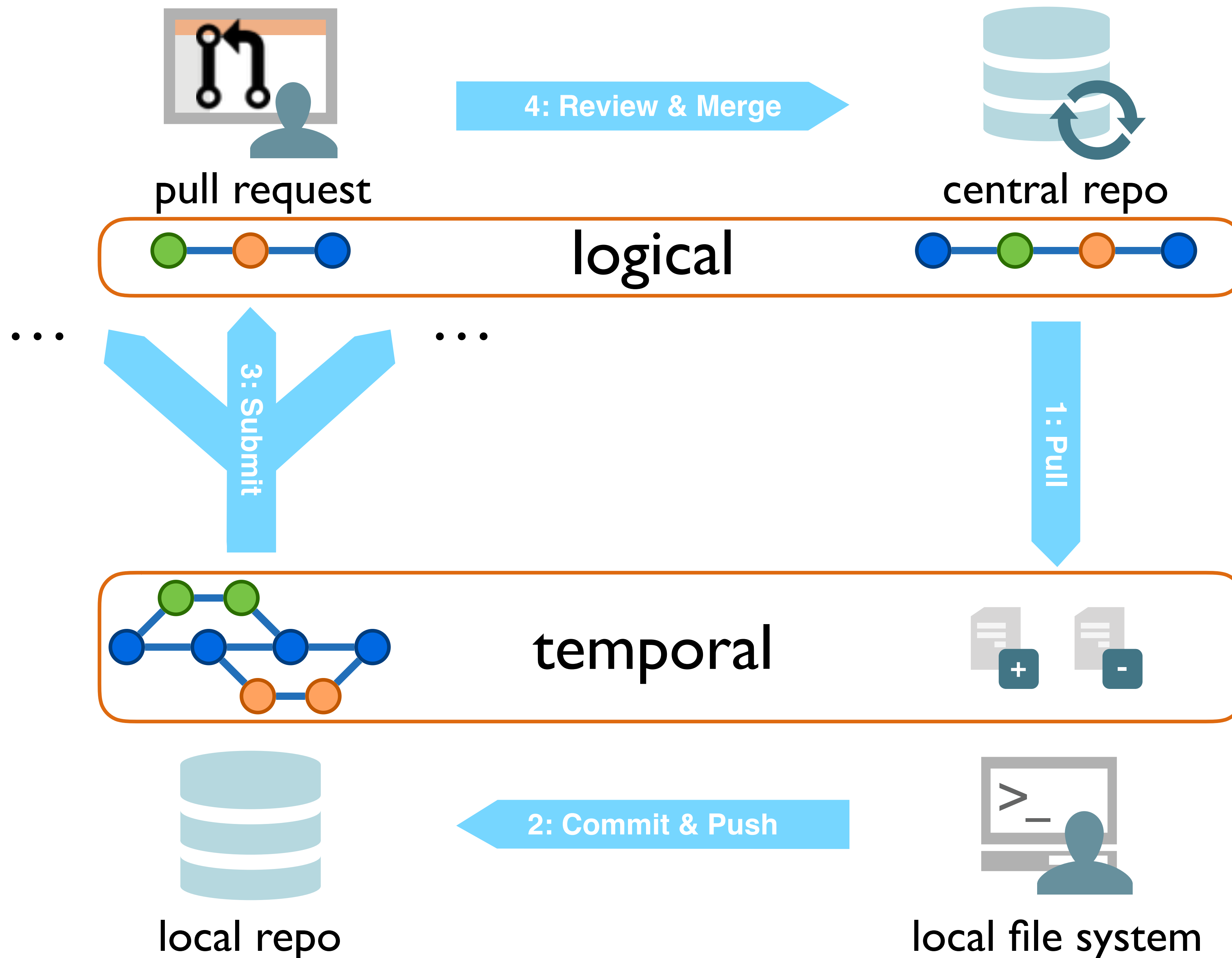
Lifetime of a Pull Request



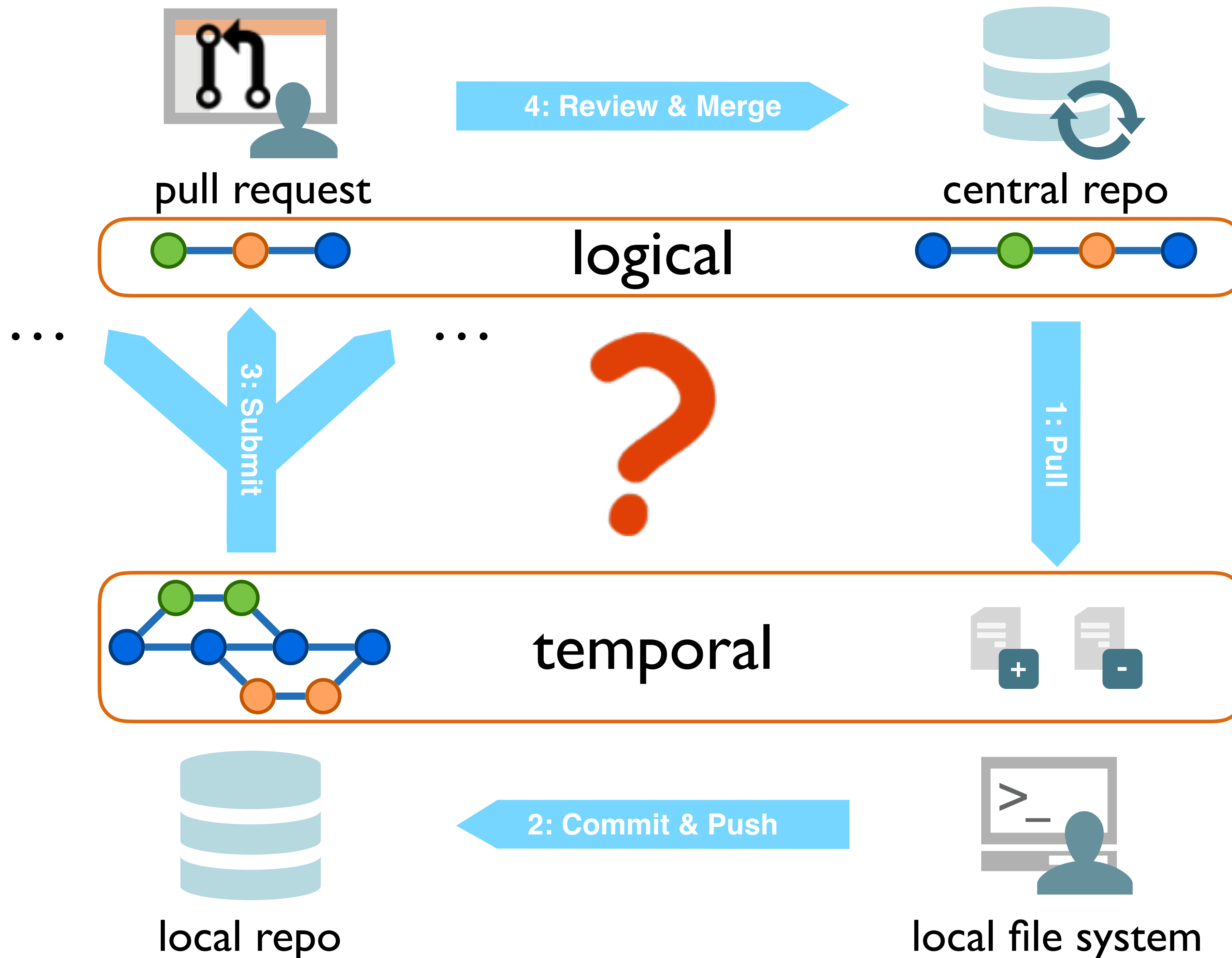
Lifetime of a Pull Request



Lifetime of a Pull Request



Lifetime of a Pull Request



Semantic History Slicing

Identify commits (change sets) w.r.t. a criterion:

- Authored by John
- Committed in last month

Semantic History Slicing

Identify commits (change sets) w.r.t. a criterion:

- Authored by John
- Committed in last month
- Corresponding to a high-level functionality [ASE'15]
  1. implementing a new feature, a hot-fix, improvement, etc.
  2. passing a test suite

Semantic History Slicing

Identify commits (change sets) w.r.t. a criterion:

- Authored by John
- Committed in last month
- Corresponding to a high-level functionality [ASE'15]
  1. implementing a new feature, a hot-fix, improvement, etc.
  2. passing a test suite

Other applications:

- Porting features, bug fixes, etc.
- History understanding, transformation [Muslu et al., ASE'15], etc.

Existing Approaches

CSLICER [ASE'15]

- Identifies an *over-approximated* set of commits essential for passing tests
- Based on static dependency analysis
- Executes tests only once, trades precision for efficiency

Existing Approaches

CSLICER [ASE'15]

- Identifies an *over-approximated* set of commits essential for passing tests
- Based on static dependency analysis
- Executes tests only once, trades precision for efficiency

Imprecise solution:

- Includes unnecessary or irrelevant changes
- E.g., tests trigger logging but don't need it to pass

Existing Approaches

Minimal semantic history slice:

- Capturing valid semantics — passing target tests
- Cannot be reduced anymore

Existing Approaches

Minimal semantic history slice:

- Capturing valid semantics — passing target tests
- Cannot be reduced anymore

Divide-and-conquer history partitioning:

- Git-bisect [https://git-scm.com/docs/git-bisect]

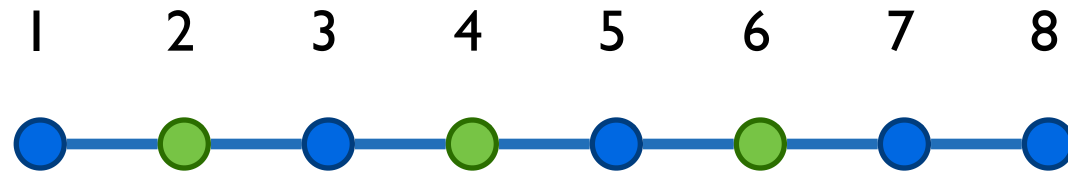
  binary search a version with defects

- Delta debugging [Zeller, ESEC/FSE-7]

  minimize fault-inducing changes

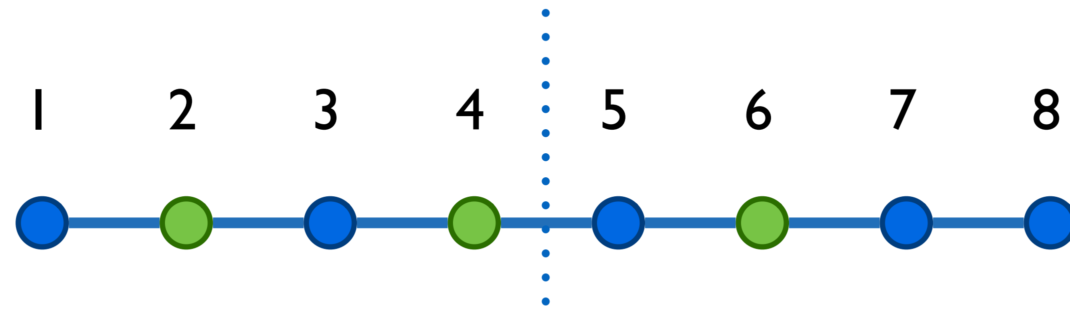
- Blind history partitioning can be slow

History Partitioning



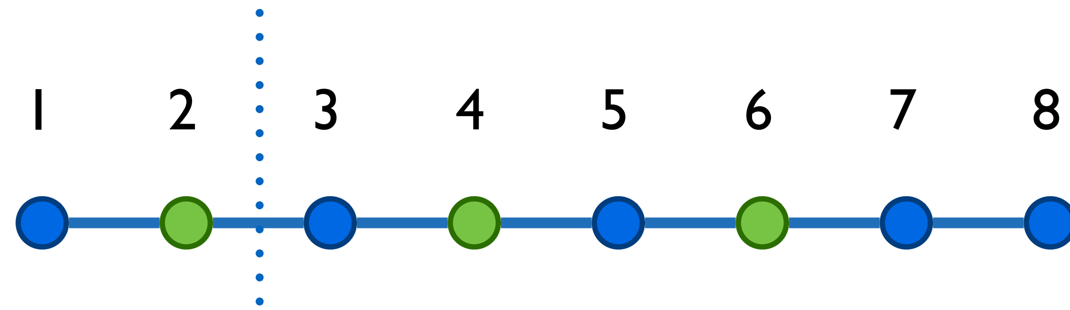
Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔

History Partitioning



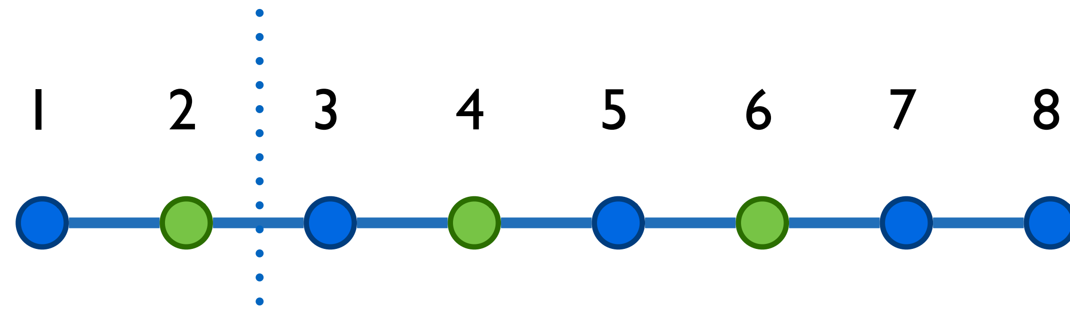
Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗

History Partitioning



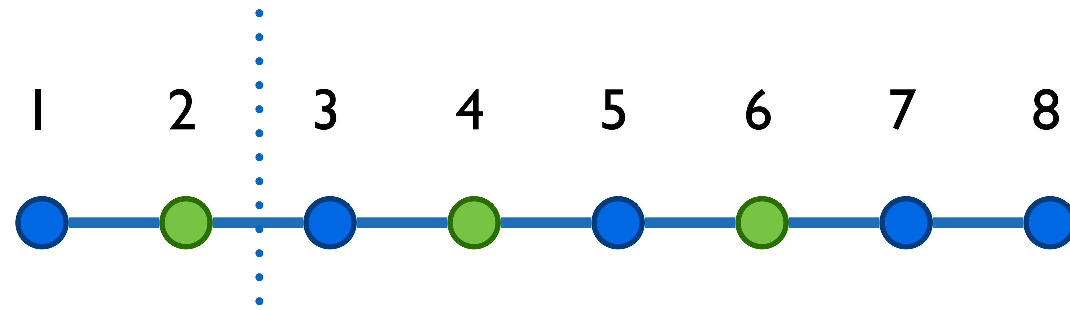
Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗

History Partitioning



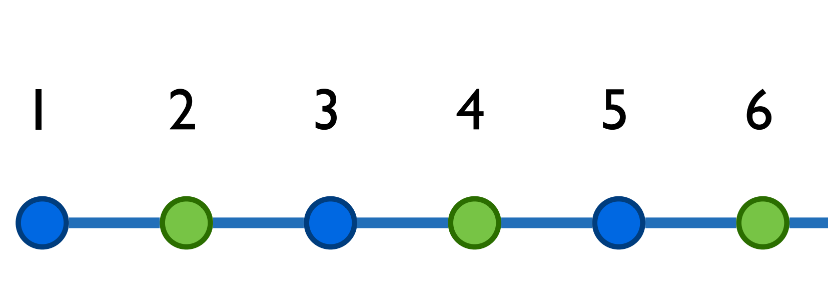
Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗
3	2	1,2	✗
4	2	3,4,5,6,7,8	✗

History Partitioning



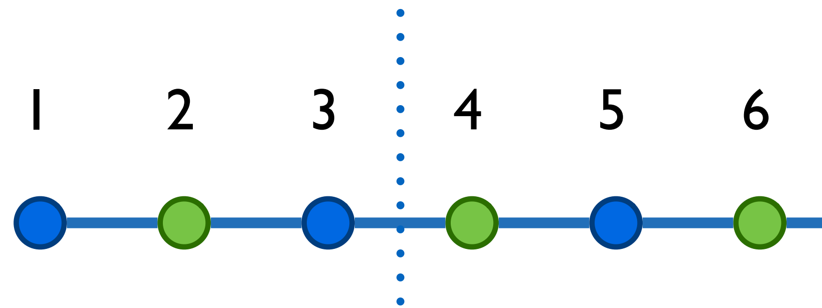
Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗
3	2	1,2	✗
4	2	3,4,5,6,7,8	✗
…	…	…	…

History Partitioning



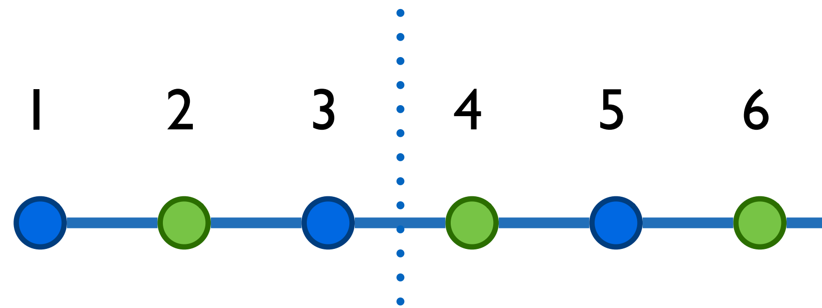
Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗
3	2	1,2	✗
4	2	3,4,5,6,7,8	✗
...	...	...	...
9	2	1,2,3,4,5,6	✔

History Partitioning



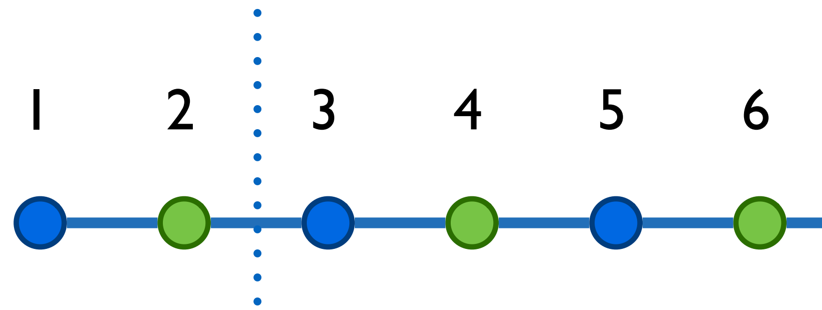
Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗
3	2	1,2	✗
4	2	3,4,5,6,7,8	✗
...	...	...	...
9	2	1,2,3,4,5,6	✔

History Partitioning



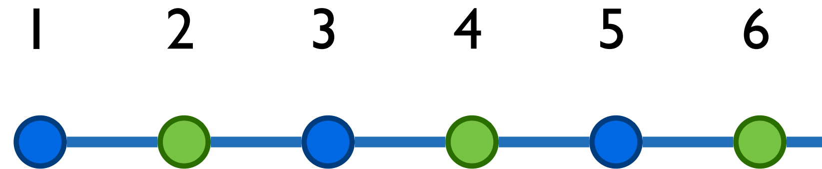
Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗
3	2	1,2	✗
4	2	3,4,5,6,7,8	✗
...	...	...	...
9	2	1,2,3,4,5,6	✔
10	3	1,2,3	✗
11	3	4,5,6	✗

History Partitioning



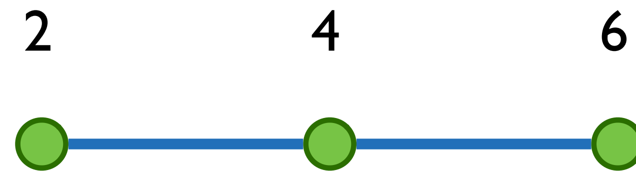
Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗
3	2	1,2	✗
4	2	3,4,5,6,7,8	✗
...	...	...	...
9	2	1,2,3,4,5,6	✔
10	3	1,2,3	✗
11	3	4,5,6	✗
-	2	1,2	-
12	2	3,4,5,6	✗

History Partitioning



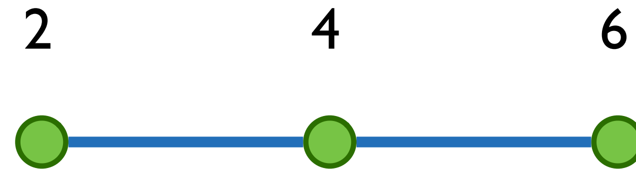
Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗
3	2	1,2	✗
4	2	3,4,5,6,7,8	✗
...	...	...	...
9	2	1,2,3,4,5,6	✔
10	3	1,2,3	✗
11	3	4,5,6	✗
-	2	1,2	-
12	2	3,4,5,6	✗
...	...	...	...

History Partitioning



Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗
3	2	1,2	✗
4	2	3,4,5,6,7,8	✗
...	...	...	...
9	2	1,2,3,4,5,6	✔
10	3	1,2,3	✗
11	3	4,5,6	✗
-	2	1,2	-
12	2	3,4,5,6	✗
...	...	...	...
25	1	2,4,6	✔

History Partitioning



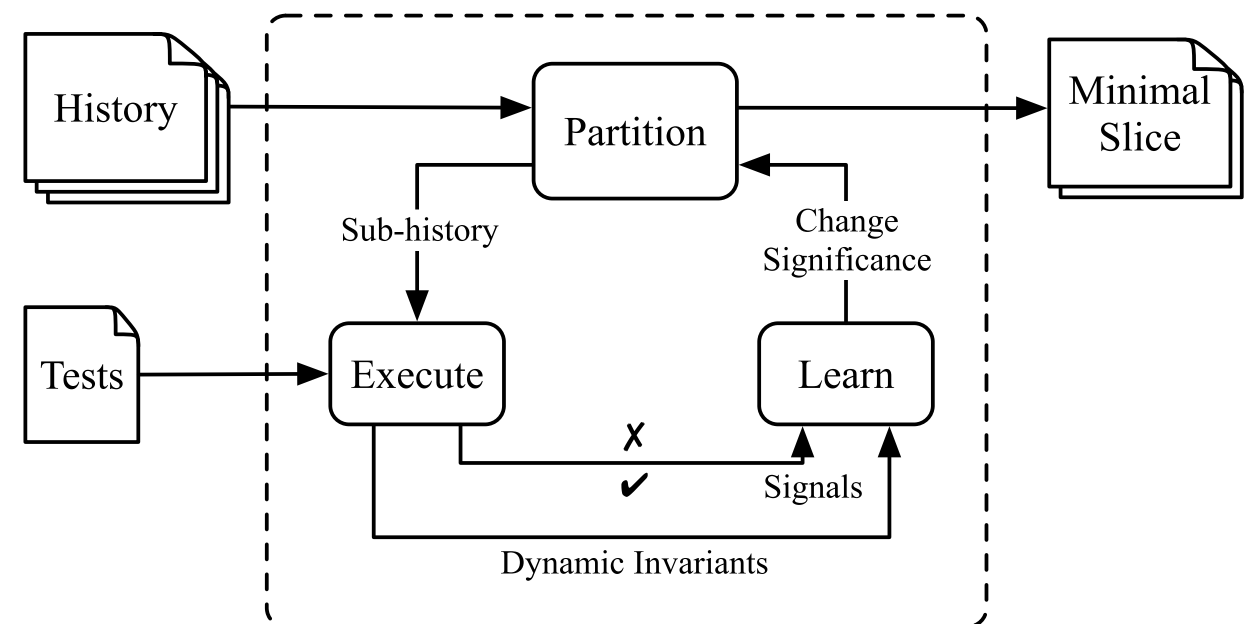
Ordering is crucial!

Iteration	Size	Partition	Outcome
0	8	1,2,3,4,5,6,7,8	✔
1	4	1,2,3,4	✗
2	4	5,6,7,8	✗
3	2	1,2	✗
4	2	3,4,5,6,7,8	✗
...	...	...	...
9	2	1,2,3,4,5,6	✔
10	3	1,2,3	✗
11	3	4,5,6	✗
-	2	1,2	-
12	2	3,4,5,6	✗
...	...	...	...
25	1	2,4,6	✔

Dynamic Delta Refinement

Precise and *efficient* semantic history slicing:

1. History partitioning
2. Iterative test executions
3. Learn *change significance* based on observable signals:
   - local change impacts per program location
   - test outcomes
4. Refine partition schemes

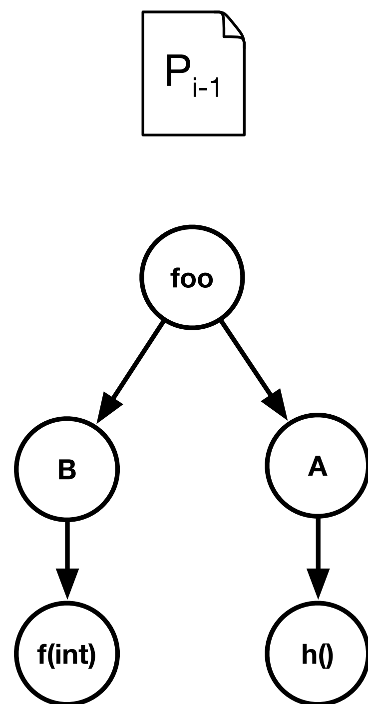


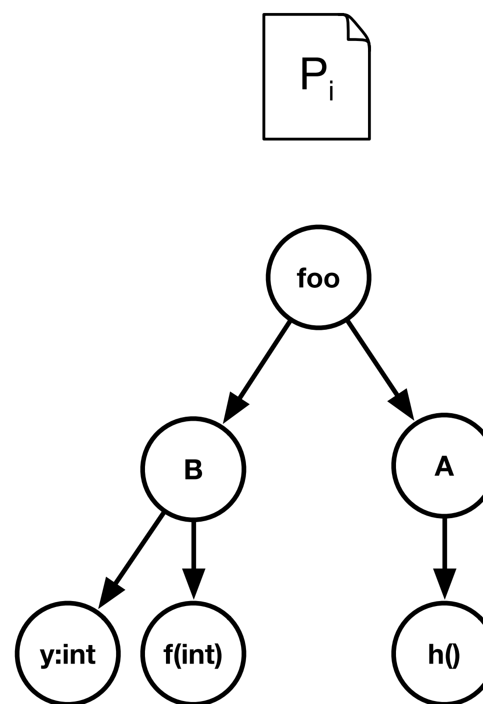
Divide-and-conquer + static analysis

Outline

1. Introduction
2. Dynamic Delta Refinement
   - Atomic changes
   - Change significance learning
3. Evaluation
4. Related Work & Conclusion

Atomic Changes





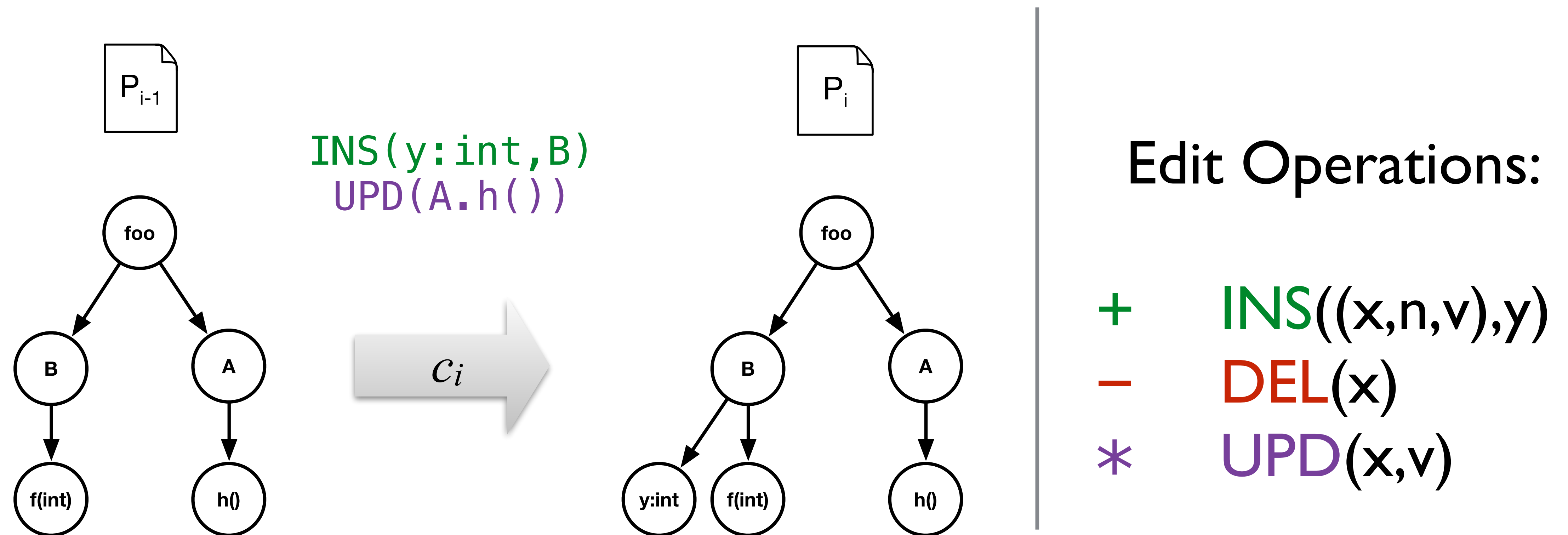
Edit Operations:

+ INS((x,n,v),y)
– DEL(x)
∗ UPD(x,v)

Compare two abstract syntax trees:

- Ignoring cosmetic changes
- Focusing on structural nodes (class, method, and field)
- Structural differencing [Fluri et al., IEEE TSE'07]

Atomic Changes



Compare two abstract syntax trees:

- Ignoring cosmetic changes
- Focusing on structural nodes (class, method, and field)
- Structural differencing [Fluri et al., IEEE TSE'07]

Change Significance

History ≔ sequence of atomic changes

✔

Change Significance

History ≔ sequence of atomic changes

- $H = \langle c_1, c_2, \ldots, c_k \rangle$

Change Significance

History ≔ sequence of atomic changes

- $H = \langle c_1, c_2, \ldots, c_k \rangle$

Significance checks for one change given a test suite T

Change Significance

History ≔ sequence of atomic changes

- $H = \langle c_1, c_2, \ldots, c_k \rangle$

Significance checks for one change given a test suite T

- Is c_i an essential change for passing test T?

Change Significance

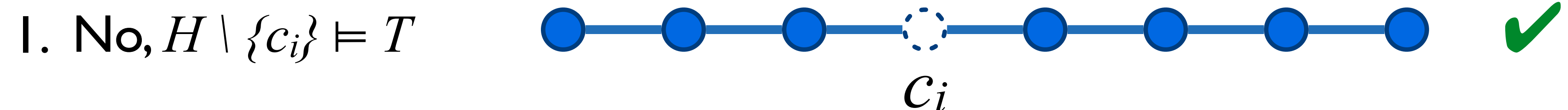
History ≔ sequence of atomic changes

- $H = \langle c_1, c_2, \ldots, c_k \rangle$

Significance checks for one change given a test suite T

- Is c_i an essential change for passing test T ?

1. No, $H \setminus \{c_i\} \vDash T$



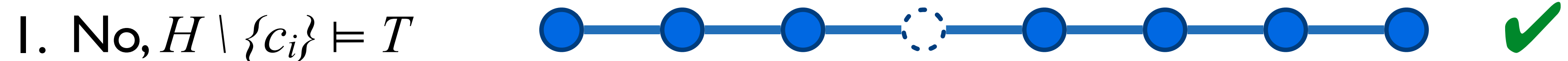
Change Significance

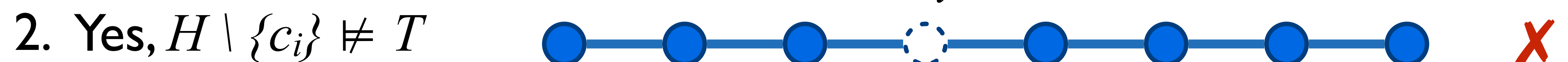
History ≔ sequence of atomic changes

- $H = \langle c_1, c_2, \ldots, c_k \rangle$

Significance checks for one change given a test suite T

- Is c_i an essential change for passing test T ?
  1. No, $H \setminus \{c_i\} \models T$
  2. Yes, $H \setminus \{c_i\} \not\models T$





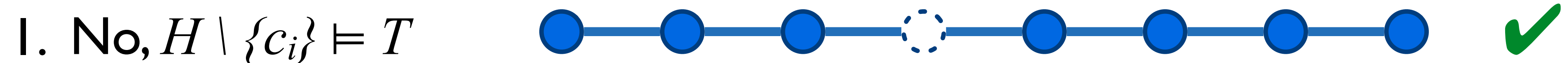
Change Significance

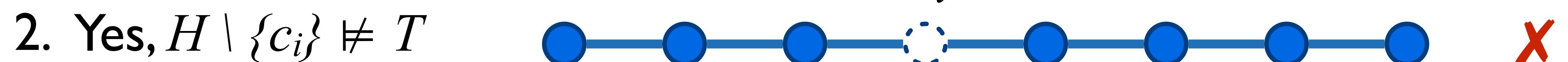
History ≔ sequence of atomic changes

- $H = \langle c_1,\ c_2,\ \ldots,\ c_k \rangle$

Significance checks for one change given a test suite T

- Is c_i an essential change for passing test T?
  1. No, $H \setminus \{c_i\} \models T$
  2. Yes, $H \setminus \{c_i\} \not\models T$



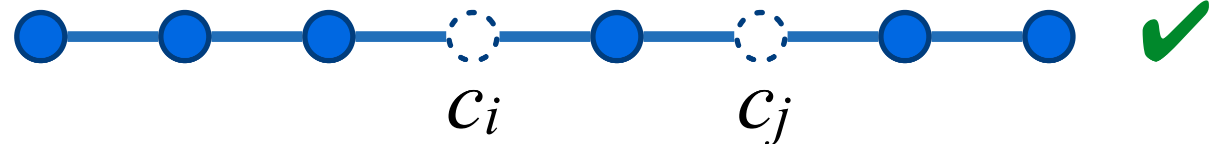


- Is c_i more important than c_j w.r.t. test T?
  3. Yes, $H \setminus \{c_j\} \models T$ but $H \setminus \{c_i\} \not\models T$
  4. Not sure, $H \setminus \{c_j\} \models T$ and $H \setminus \{c_i\} \models T$

Change Significance

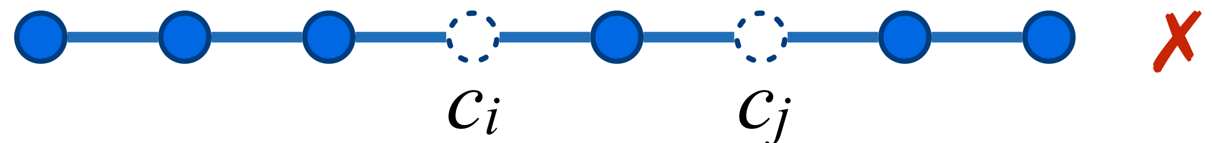
Significance checks for multiple atomic changes at once

- $H \setminus \{c_i, c_j\} \models T$



none of them is important

- $H \setminus \{c_i, c_j\} \not\models T$



some of them is important, if not all

each of them is important with **x** probability?

Generalized Change Significance

Enhanced test signals: exploiting chain of effects

```
class A {
  int f() {return (new B).g();}
}

class B {
  int x = 1;          --1. UPD-->  int x = 2;
  String s = "null";


  int g() {
    int z = h(s, x);
    return z;
  }

  int h(String v, int t) {          --2. UPD-->  int h(String v, int t) {
    return v == null ? 0 : t;                      return v != null ? 0 : t;
  }                                              }
}
```

✔ • $H \models T$ "assert f() == 2"

v == null, t == 2

ret == 2

Generalized Change Significance

Enhanced test signals: exploiting chain of effects

```
class A {
  int f() {return (new B).g();}
}

class B {
  int x = 1;
  String s = "null";


  int g() {
    int z = h(s, x);
    return z;
  }

  int h(String v, int t) {
    return v == null ? 0 : t;
  }
}
```

✔ • $H \models T$ "assert f() == 2"

• x: 2 -> 1

2. UPD

v == null, t == 2

```
int h(String v, int t) {
  return v != null ? 0 : t;
}
```

ret == 2

Generalized Change Significance

Enhanced test signals: exploiting chain of effects

```
class A {
  int f() {return (new B).g();}
}

class B {
  int x = 1;
  String s = “null”;

  int g() {
    int z = h(s, x);
    return z;
  }
  int h(String v, int t) {
    return v == null ? 0 : t;
  }
}
```

✔ • $H \models T$ "assert f() == 2"

• x: 2 –> 1

• t: 2 –> 1

v == null, t == 2

2. UPD

v == null, t == 1

```
int h(String v, int t) {
  return v != null ? 0 : t;
}
```

ret == 2

Generalized Change Significance

Enhanced test signals: exploiting chain of effects

```
class A {
  int f() {return (new B).g();}
}

class B {
  int x = 1;
  String s = "null";


  int g() {
    int z = h(s, x);
    return z;
  }
  int h(String v, int t) {
    return v == null ? 0 : t;
  }
}
```

✔ • $H \models T$ "assert f() == 2"

- x: 2 –> 1
- t: 2 –> 1
- h.ret: 2 –> 1

v == null, t == 2	2. UPD	v == null, t == 1
ret == 2		ret == 1

```
int h(String v, int t) {
  return v != null ? 0 : t;
}
```

Generalized Change Significance

Enhanced test signals: exploiting chain of effects

```
class A {
  int f() {return (new B).g();}
}

class B {
  int x = 1;
  String s = "null";

  int g() {
    int z = h(s, x);
    return z;
  }
  int h(String v, int t) {
    return v == null ? 0 : t;
  }
}
```

v == null, t == 2

ret == 2

2. UPD

```
int h(String v, int t) {
  return v != null ? 0 : t;
}
```

v == null, t == 1

ret == 1

- ✔ $H \models T$ "assert f() == 2"
- x: 2 -> 1
- t: 2 -> 1
- h.ret: 2 -> 1
- ✗ $H \setminus \{c_1\} \not\models T$

Generalized Change Significance

Enhanced test signals: exploiting chain of effects

```
class A {
  int f() {return (new B).g();}
}

class B {
  int x = 1;
  String s = "null";

  int g() {
    int z = h(s, x);
    return z;
  }

  int h(String v, int t) {
    return v == null ? 0 : t;
  }
}
```

✔ • $H \models T$ "assert f() == 2"

• x: 2 -> 1

Both changes are significant!

• h.ret: 2 -> 1

✗ • $H \setminus \{c_1\} \not\models T$

v == null, t == 2 | v == null, t == 1

2. UPD

```
int h(String v, int t) {
  return v != null ? 0 : t;
}
```

ret == 2 | ret == 1

Significance Learning

Sentinels for structural nodes — *dynamic invariants*:

- Daikon [Ernst et al., IEEE Trans. SE'01]
- Pre- and post- conditions for methods
  - parameter values: `B.h(I,S)::t >= 1, B.h(I,S)::v != null`
  - returned values: `B.h(I,S)::ret == 0`
- Values of fields:
  - fields: `B::y one of {2, 3}, B::s == “abc”`

Significance Learning

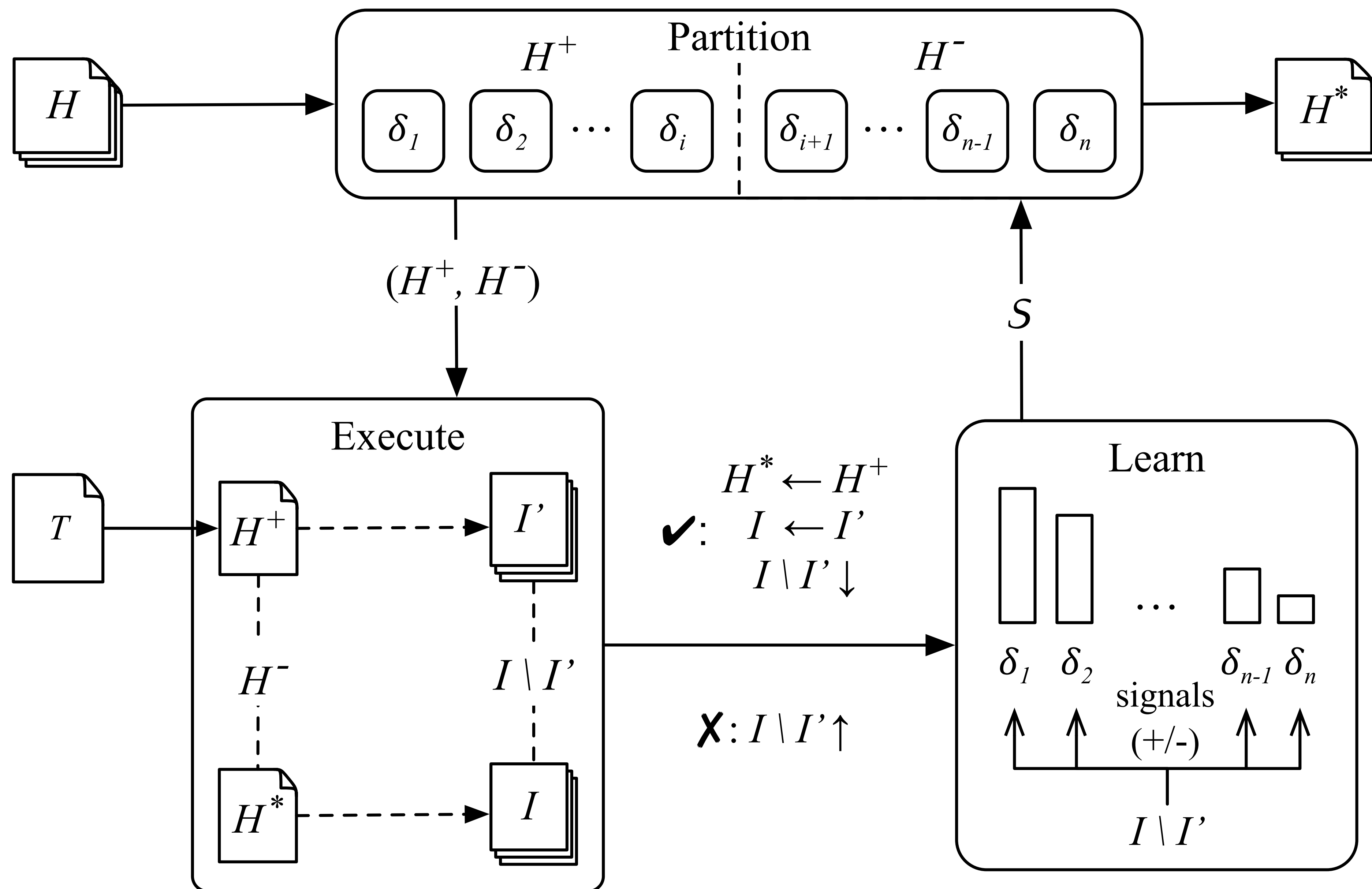
Sentinels for structural nodes — *dynamic invariants*:

- Daikon [Ernst et al., IEEE Trans. SE'01]
- Pre- and post- conditions for methods
  - parameter values: `B.h(I,S)::t >= 1, B.h(I,S)::v != null`
  - returned values: `B.h(I,S)::ret == 0`
- Values of fields:
  - fields: `B::y one of {2, 3}, B::s == "abc"`

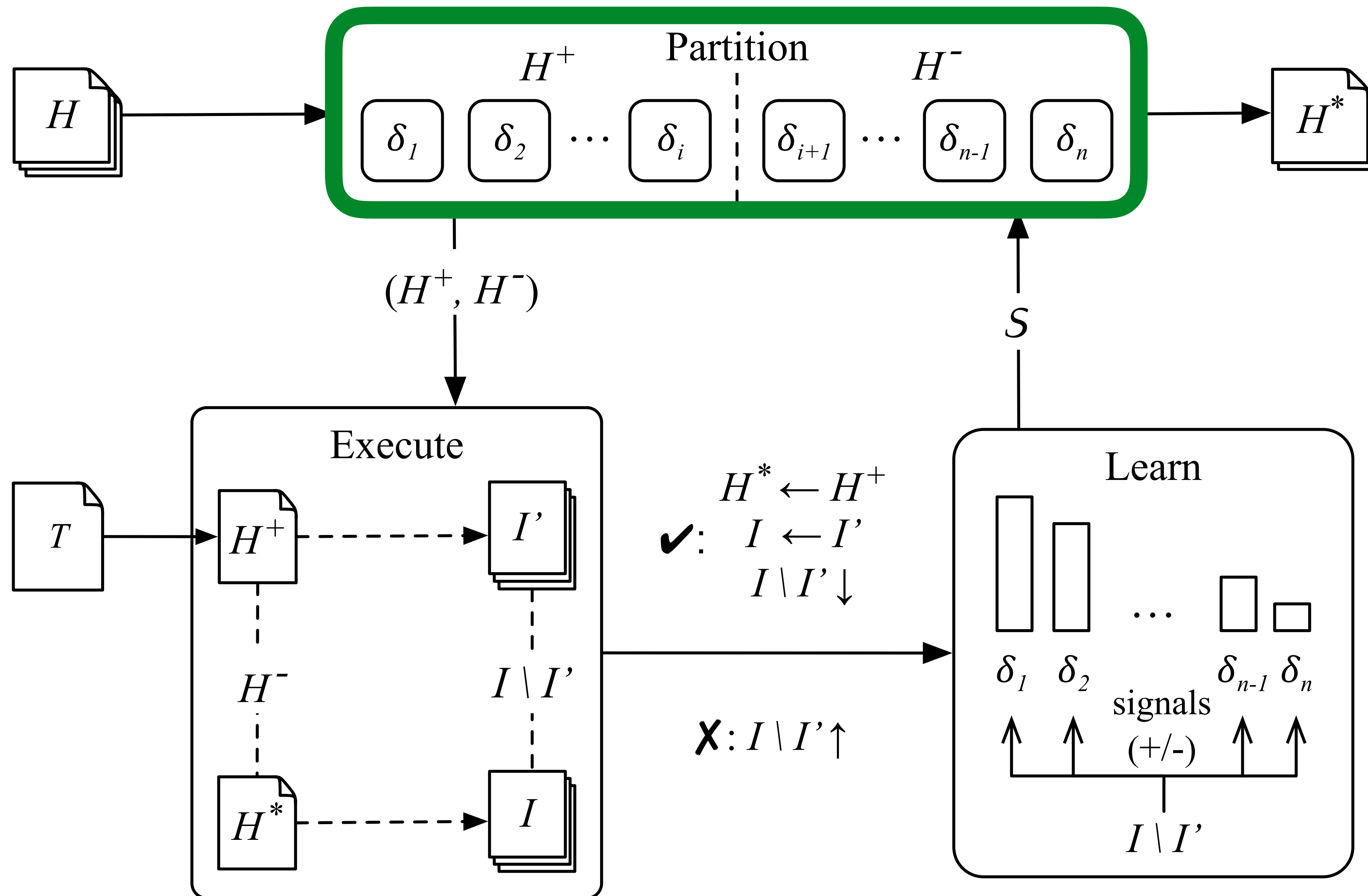
Inferring significance of changes:

- Differences of invariants: $I \setminus I'$
- Local static change impact analysis: invariants —> changes

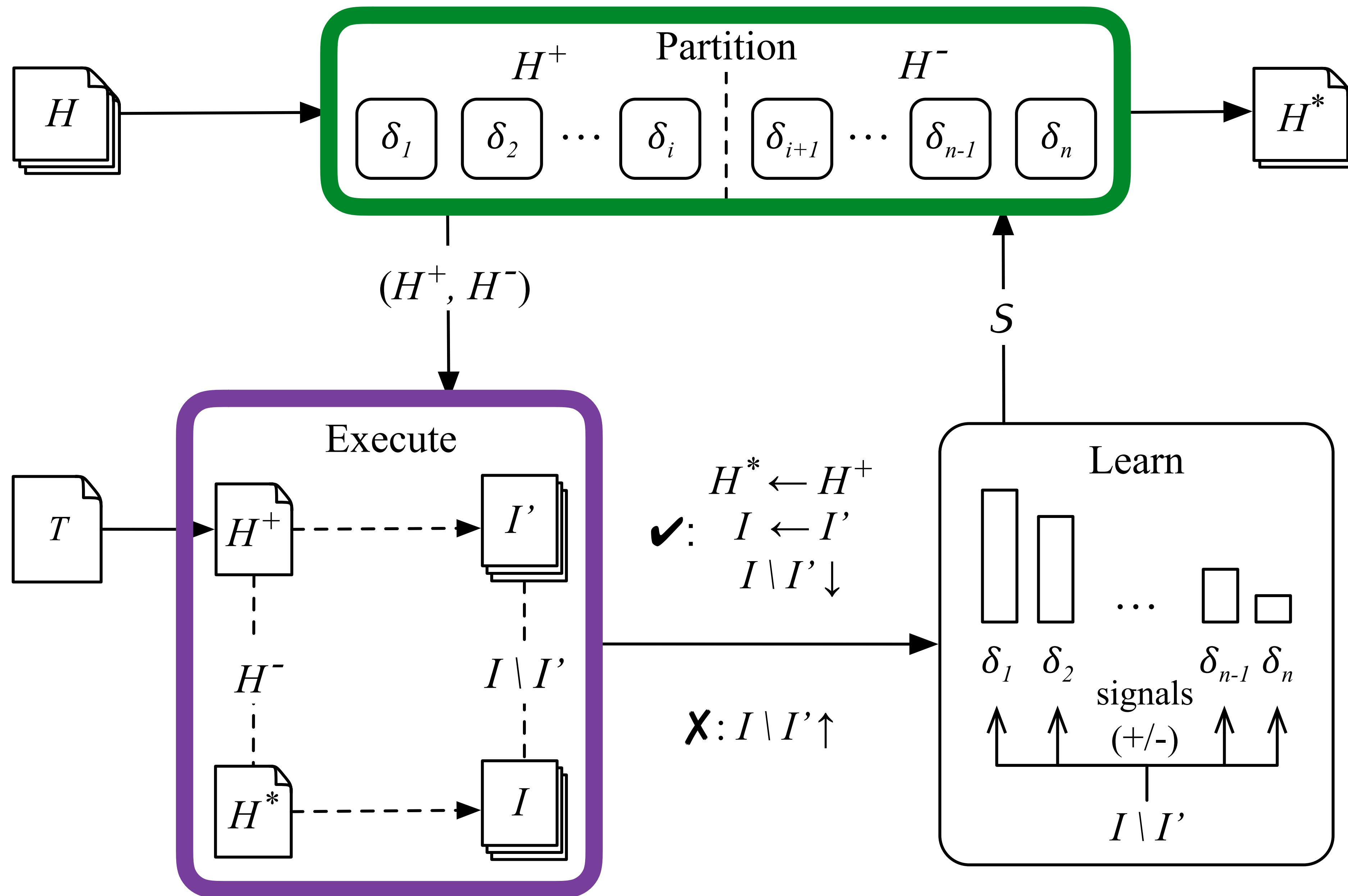
Dynamic Delta Refinement



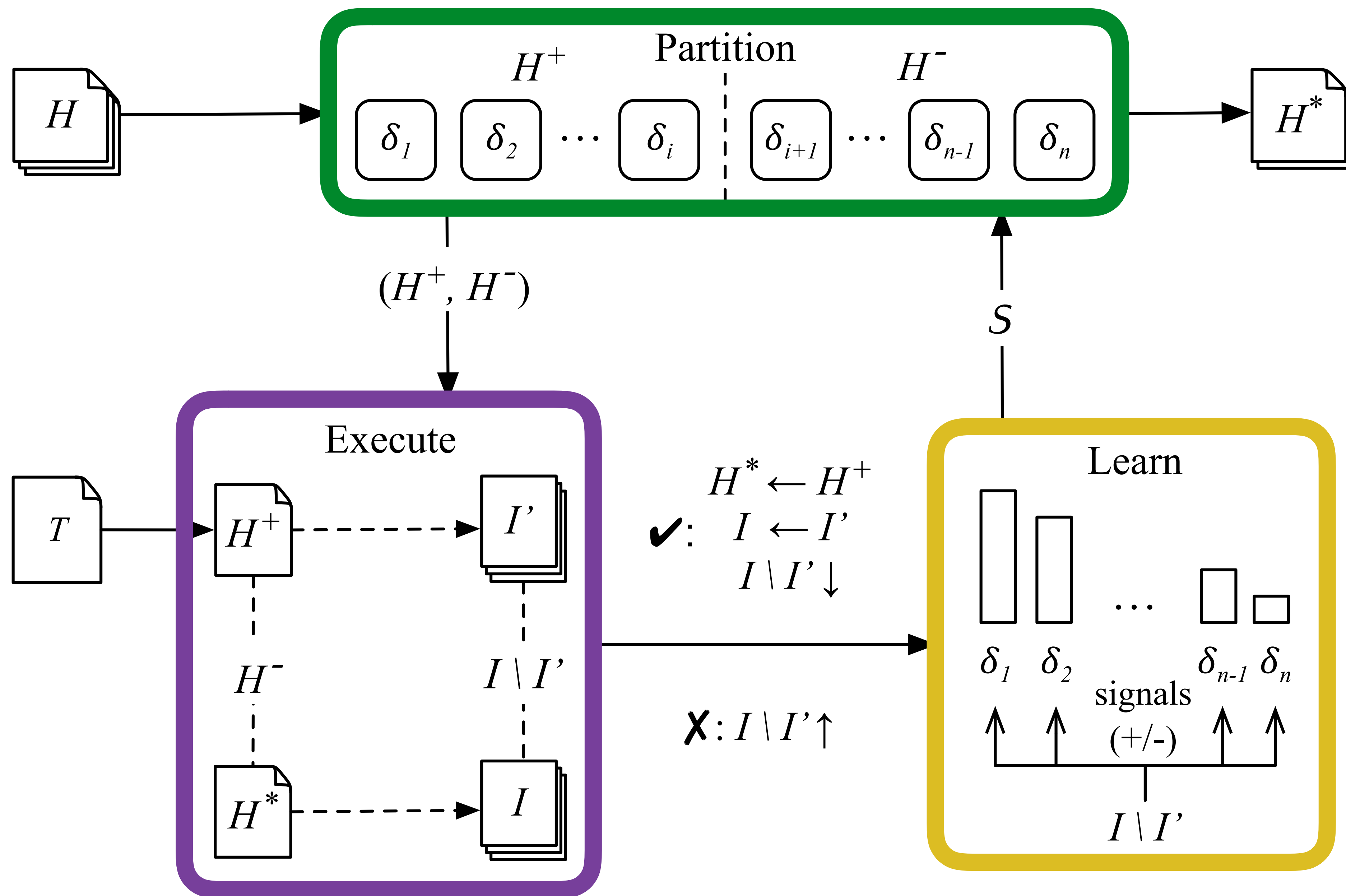
Dynamic Delta Refinement



Dynamic Delta Refinement



Dynamic Delta Refinement



Optimizations

Change Dependency Analysis [ASE'15]

- No need to run test if changes are not compatible

  e.g., method invocation depends on method declaration

- Statically predicts compilation failures without compiling

Optimizations

Change Dependency Analysis [ASE'15]

- No need to run test if changes are not compatible

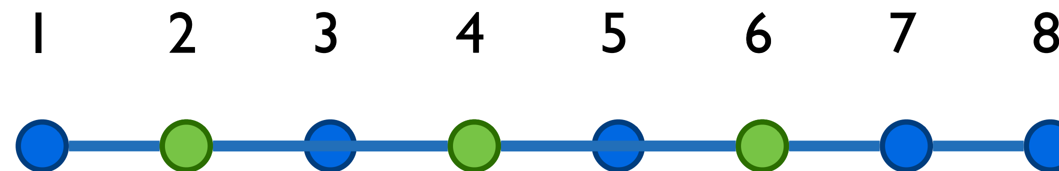
  e.g., method invocation depends on method declaration

- Statically predicts compilation failures without compiling

Decaying significance scores

- Later significance tests are more reliable
- Update significance beliefs of the remaining changes

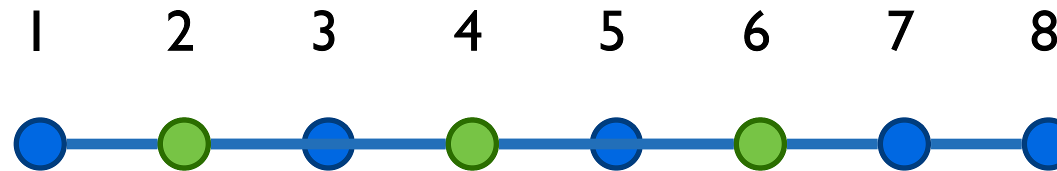
Dynamic Delta Refinement



Iteration	Partition	Outcome
0	1,2,3,4,5,6,7,8	✔
1	1,2,3,4	-
2	5,6,7,8	-
3	3,4,5,6,7,8	-
4	1,2,5,6,7,8	✗
5	1,2,3,4,7,8	-
6	1,2,3,4,5,6	✔
7	1,2,4,6	✔
8	4,6	-
9	1,2	✗
10	2,4,6	✔
11	4,6	-
12	2,6	✗

https://bitbucket.org/liyistc/gitslice/wiki/ase16-e2

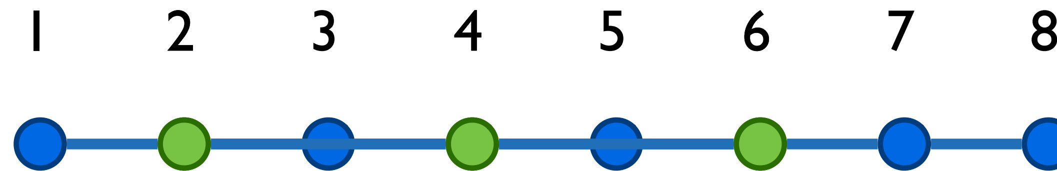
Dynamic Delta Refinement



Iteration	Partition	Outcome
0	1,2,3,4,5,6,7,8	✔
1	1,2,3,4	-
2	5,6,7,8	-
3	3,4,5,6,7,8	-
4	1,2,5,6,7,8	✗
5	1,2,3,4,7,8	-
6	1,2,3,4,5,6	✔
7	1,2,4,6	✔
8	4,6	-
9	1,2	✗
10	2,4,6	✔
11	4,6	-
12	2,6	✗

https://bitbucket.org/liyistc/gitslice/wiki/ase16-e2

Dynamic Delta Refinement



Iteration	Partition	Outcome
0	1,2,3,4,5,6,7,8	✔
1	1,2,3,4	-
2	5,6,7,8	-
3	3,4,5,6,7,8	-
4	1,2,5,6,7,8	✗
5	1,2,3,4,7,8	-
6	1,2,3,4,5,6	✔
7	1,2,4,6	✔
8	4,6	-
9	1,2	✗
10	2,4,6	✔
11	4,6	-
12	2,6	✗

#tests
6 vs. 25

https://bitbucket.org/liyistc/gitslice/wiki/ase16-e2

Outline

1. Introduction
2. Dynamic Delta Refinement
   - Atomic Changes
   - Change significance learning
3. Evaluation
4. Related Work & Conclusion

Evaluation

Delta Refinement History Slicer: DEFINER

Research Questions:

1. Precision: how good is the slicing results?

   compared with CSLICER

2. Effectiveness: does change significance learning help?

   compared with “blind” history partitioning

3. Efficiency: how does partition affect performance?

   comparing different partition schemes

Subjects

Open source Java projects hosted in Git

- Apache Commons IO
- Apache Commons Collections
- Apache Commons Mathematics

Good documentations

- Feature descriptions, developer conversations, etc.
- Test plans

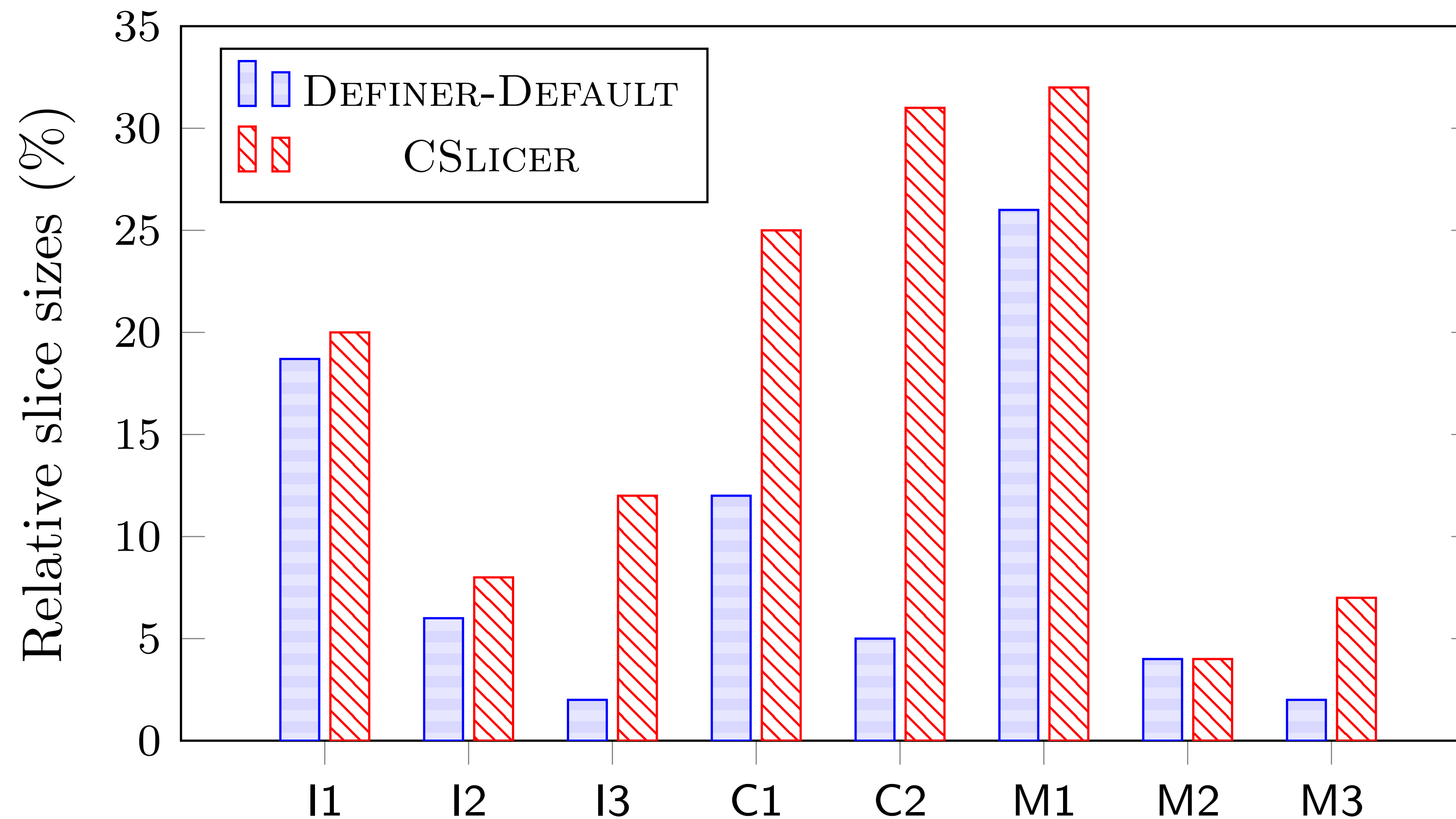
Many tests

- Designated test suite per functionality

Subjects

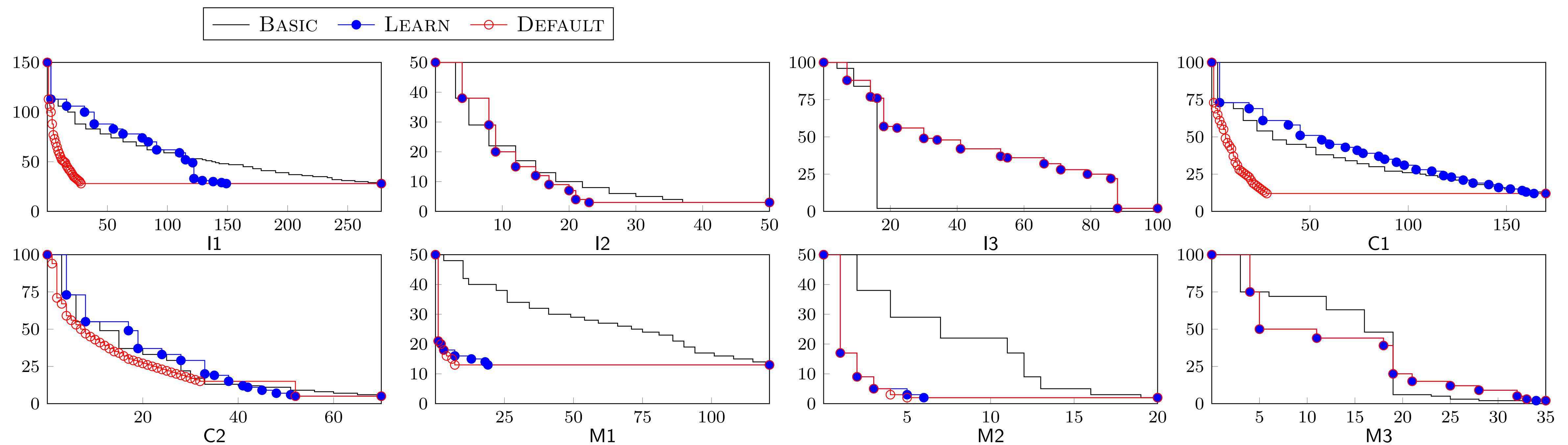
Projects	ID	Descriptions	$\|H\|$	$\|T\|$	$\|H^*\|$
io	I1	byte array output stream	150	3	28
	I2	identifies broken symlink files	50	1	3
	I3	file name utilities	100	37	2
collectio ns	C1	“indexof” function in iterable	100	1	12
	C2	“union” function in set utilities	100	1	5
math	M1	error conditions in continuous output field model	50	1	13
	M2	construct median with specific estimation type	50	1	2
	M3	large samples in polynomial fitter	100	1	2

Precision



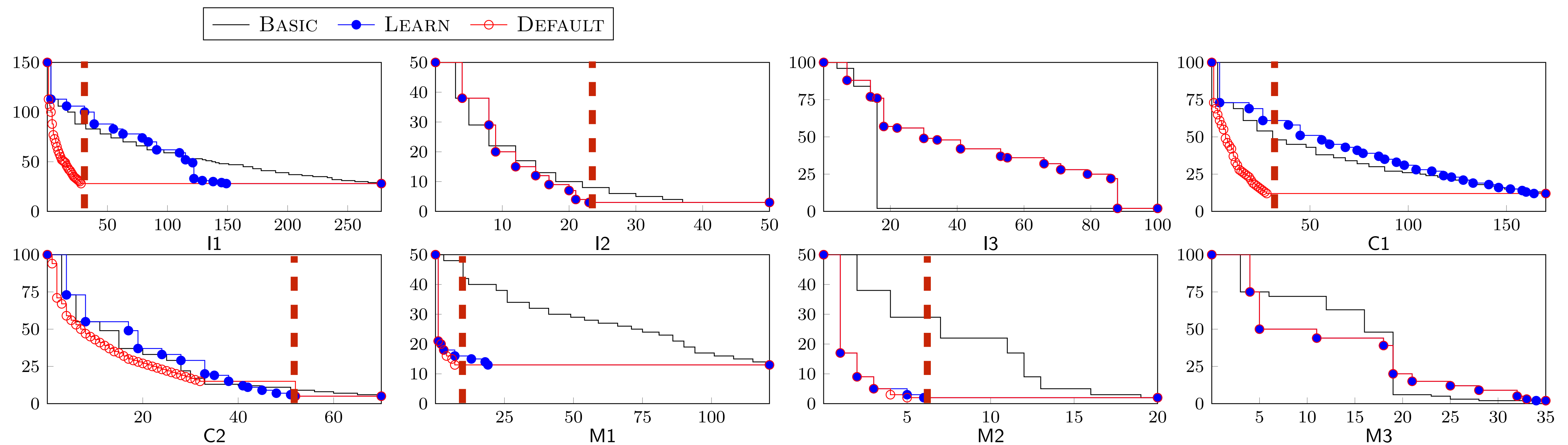
- All history slices by DEFINER are 1-minimal
- On average ~45.6% shorter than CSLICER
- Average running time: 1,162s vs. 52s

Effectiveness



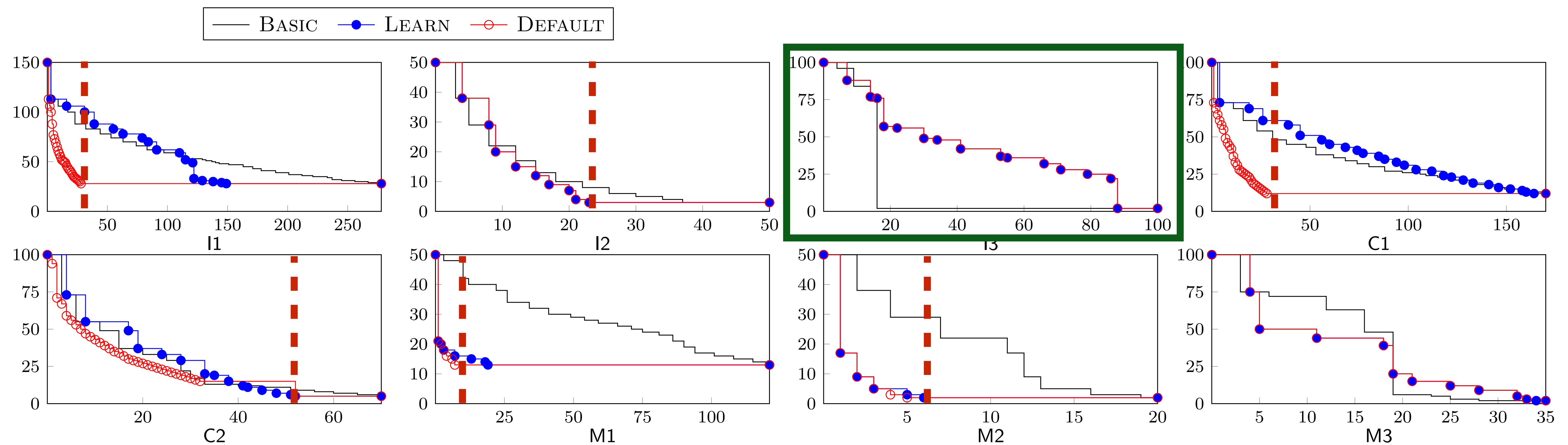
- Converges much faster in most of the cases with significance learning
- No obvious advantage when the "right answer" is too small

Effectiveness



- Converges much faster in most of the cases with significance learning
- No obvious advantage when the "right answer" is too small

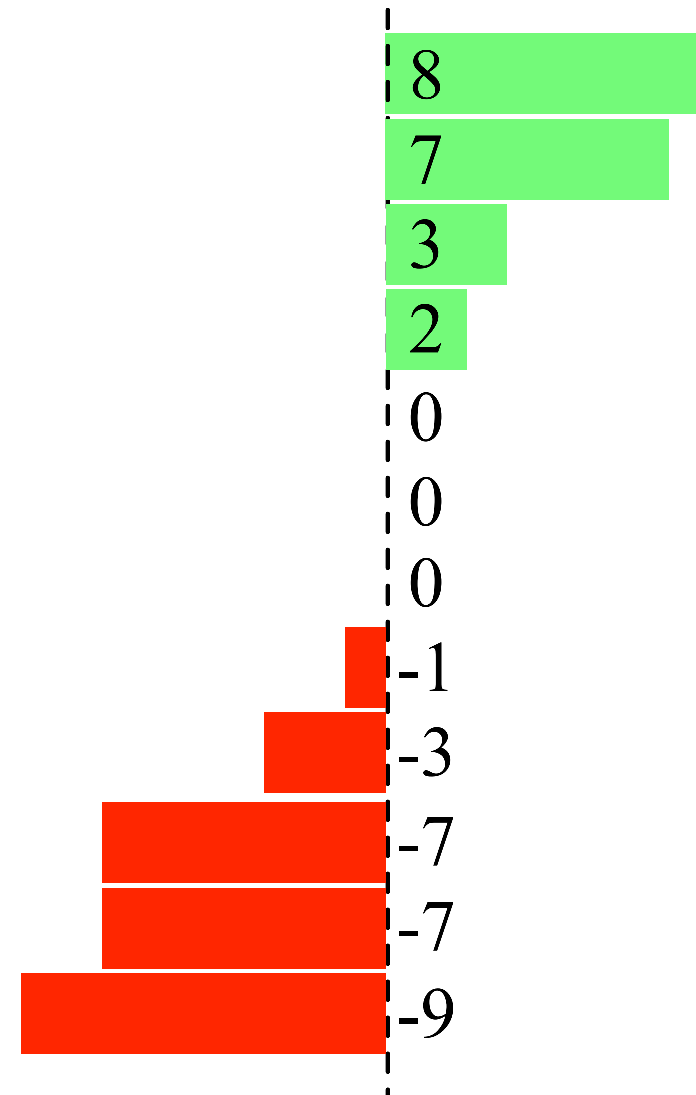
Effectiveness



- Converges much faster in most of the cases with significance learning
- No obvious advantage when the "right answer" is too small

Efficiency

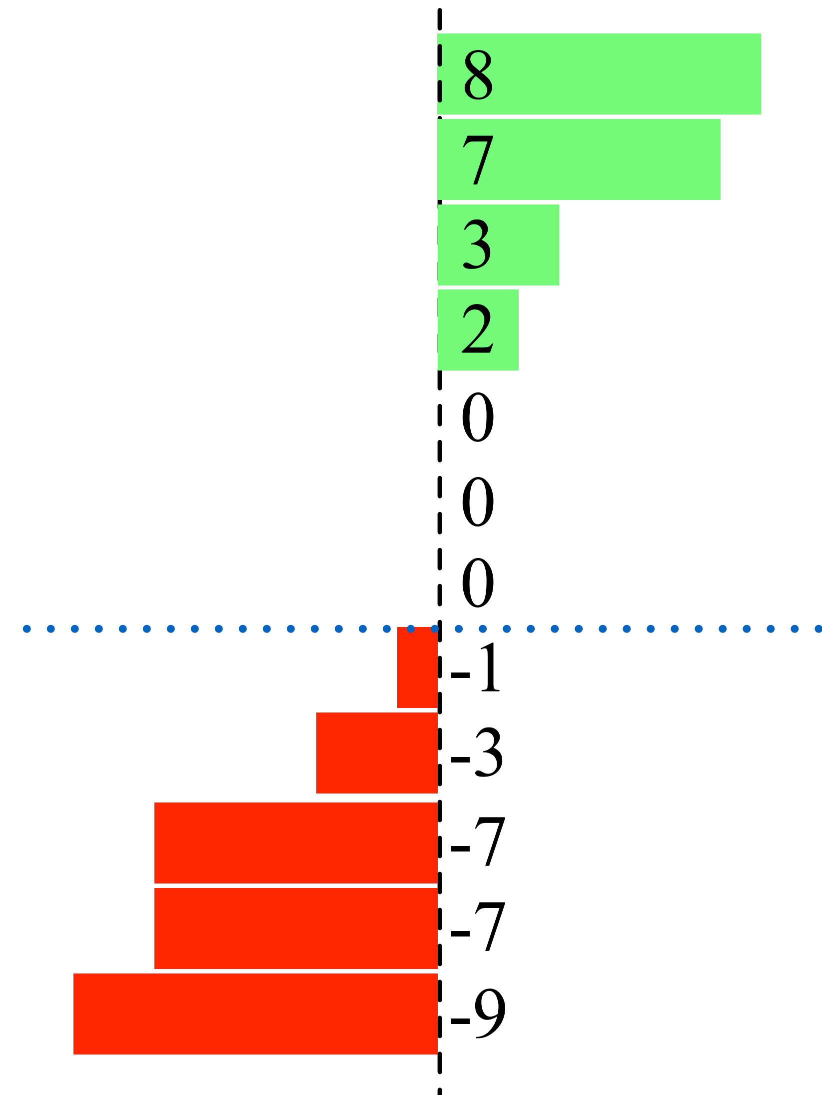
Partition Schemes:



Efficiency

Partition Schemes:

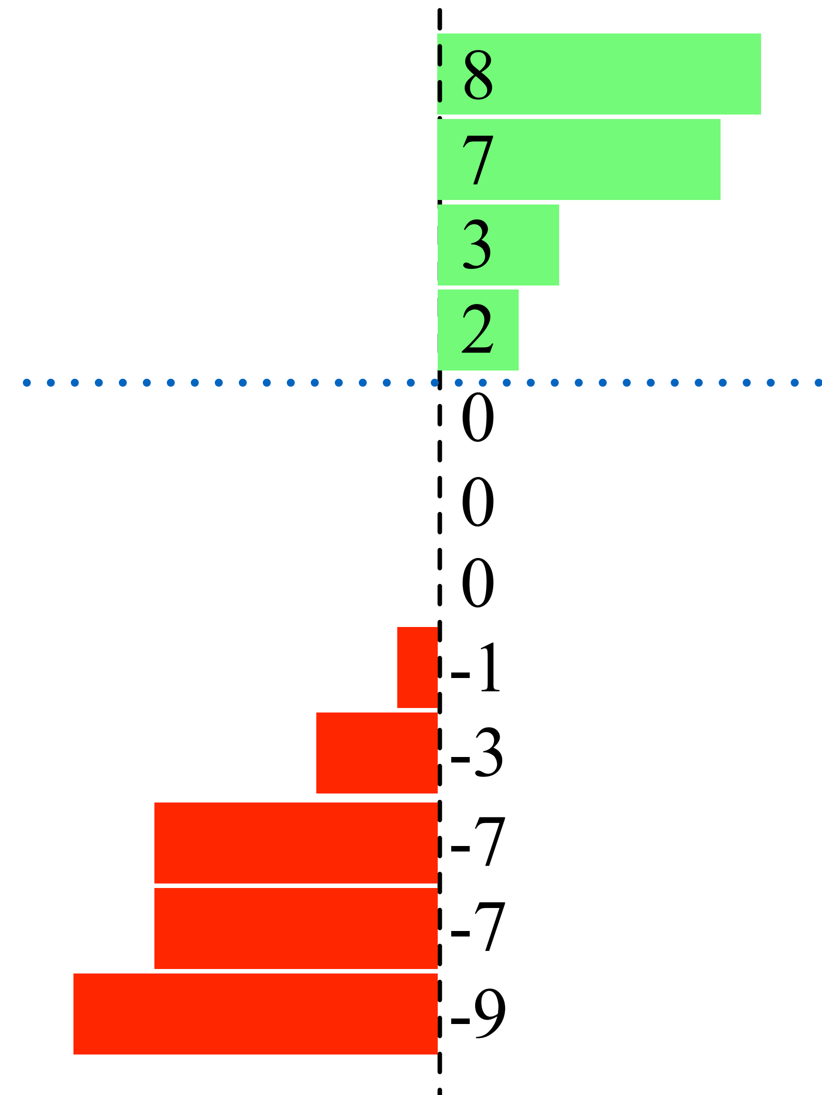
- NEG



Efficiency

Partition Schemes:

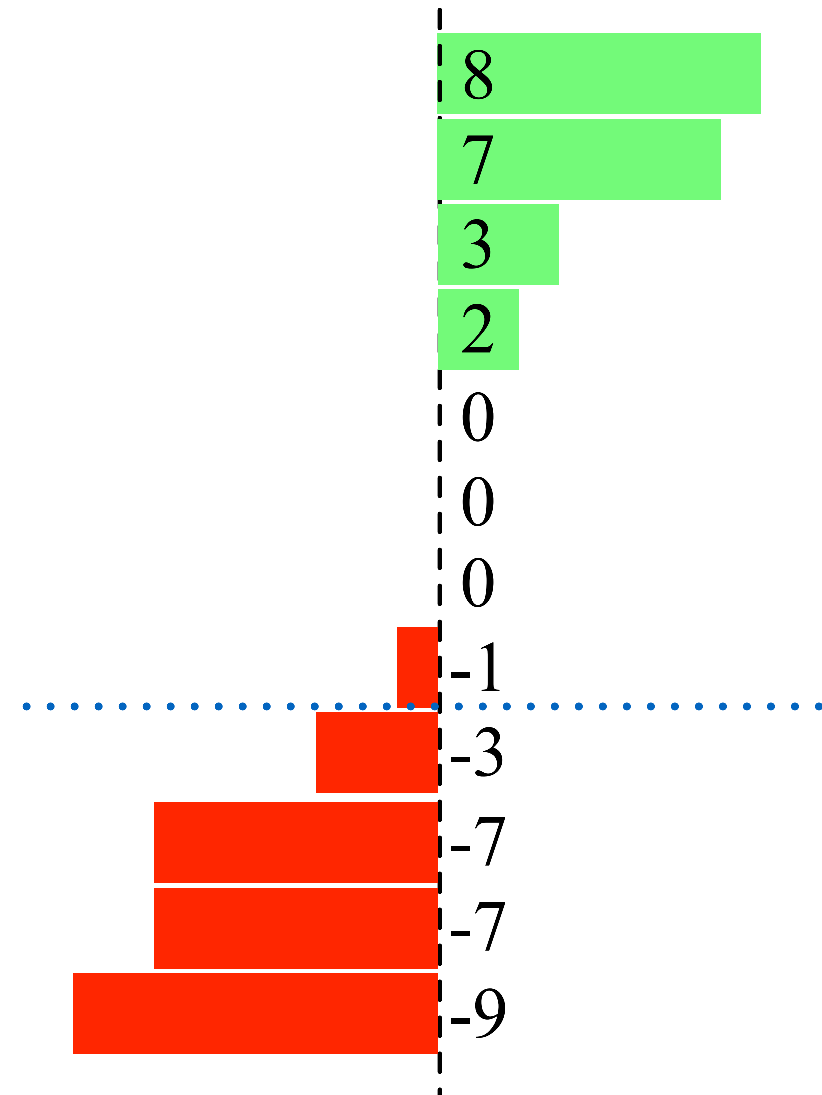
- NEG
- NonPos



Efficiency

Partition Schemes:

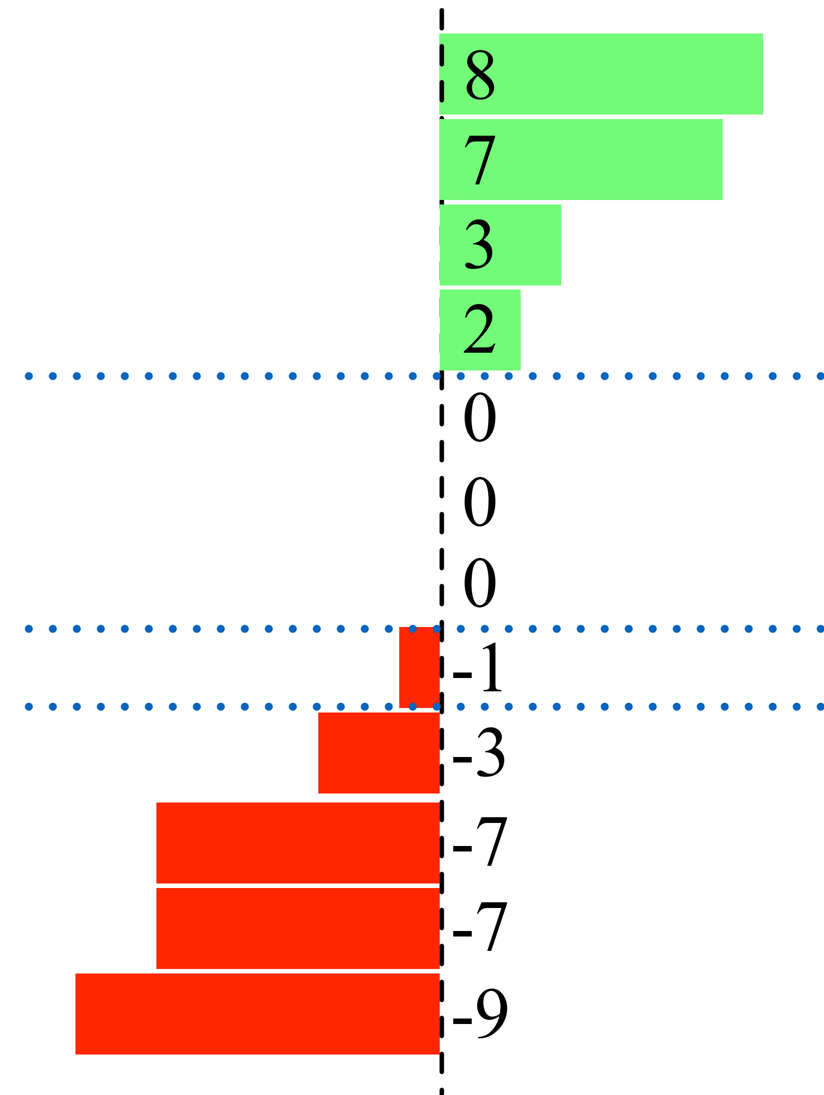
- NEG
- NonPos
- Low-3



Efficiency

Partition Schemes:

- NEG
- NonPos
- Low-3
- Combined: try all three!



Efficiency

ID	NEG	NonPos	Low-3	Combined
I1	258	258	168	168
I2	33	33	25	25
I3	89	60	89	89
C1	176	176	176	172
C2	72	72	57	57
M1	27	32	36	28
M2	7	8	11	7
M3	31	23	35	35

Efficiency

ID	NEG	NonPos	Low-3	Combined
I1	258	258	168	168
I2	33	33	25	25
I3	89	60	89	89
C1	176	176	176	172
C2	72	72	57	57
M1	27	32	36	28
M2	7	8	11	7
M3	31	23	35	35

Outline

1. Introduction
2. Dynamic Delta Refinement
   - Atomic Changes
   - Change significance learning
3. Evaluation
4. Related Work & Conclusion

Related Work

History transformation and understanding

- CSLICER [Li et al., ASE'15]
- History transformations [Muslu et al., ASE'15]
- Line-based history slicing [Servant and Jones, ASE'11, FSE'12]

Fault localization

- FaultTracer [Zhang, Kim, and Khurshid, ICSM'11]
- Delta Debugging [Zeller, ESEC/FSE7]

Conclusion & Future Work

Dynamic Delta Refinement

- Finding *minimal* semantic history slices
- Improving both *precision* and *efficiency* over state-of-the-art

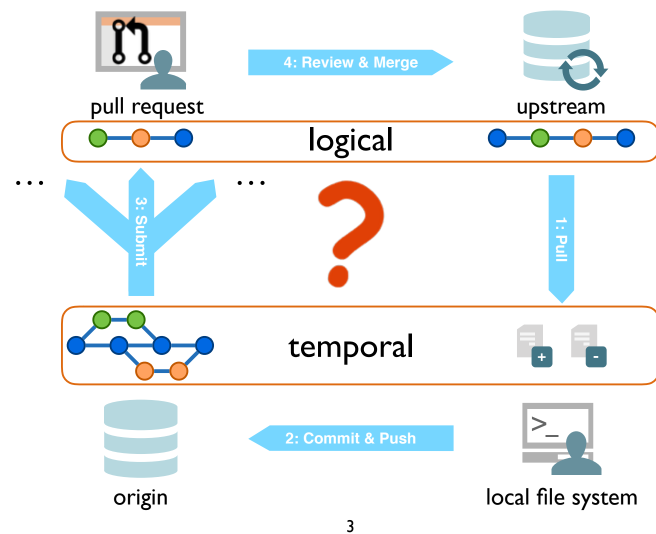
bitbucket.org/liyistc/gitslice

What's next?

- Combining with CSLICER
- Applying delta refinement to debugging and fault localization

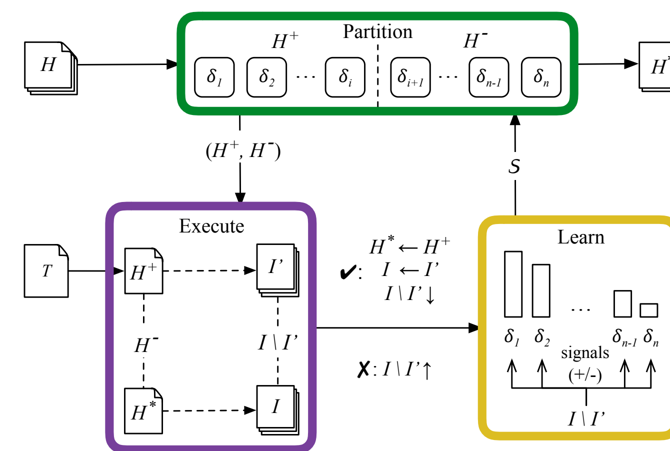
Questions?

Lifetime of a Pull Request



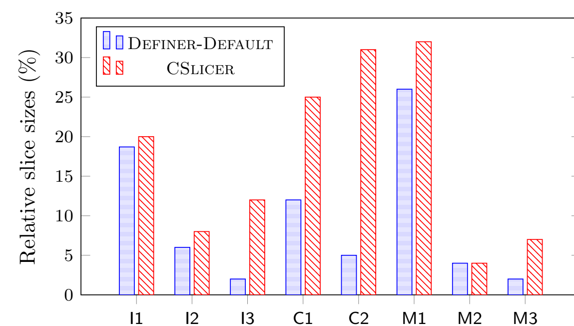
3

Dynamic Delta Refinement



15

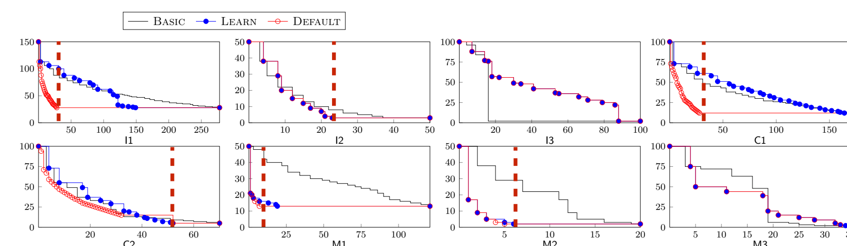
Precision



- All history slices by DEFINER are 1-minimal
- On average ~45.6% shorter than CSLICER
- Average running time: 1,162s vs. 52s

22

Effectiveness



- Converges much faster in most of the cases with significance learning
- No obvious advantage when the "right answer" is too small

23

Yi Li
liyi@cs.toronto.edu