Incremental Static Analysis

Sebastian Erdweg

Joint work with Tamás Szabó, Gábor Bergmann, Markus Voelter

TU Delft

Source code

Compile

Binary

Deploy

Customer / Production

Failure

Source code

Compile

Binary

Deploy

Customer / Production

Failure

Source code

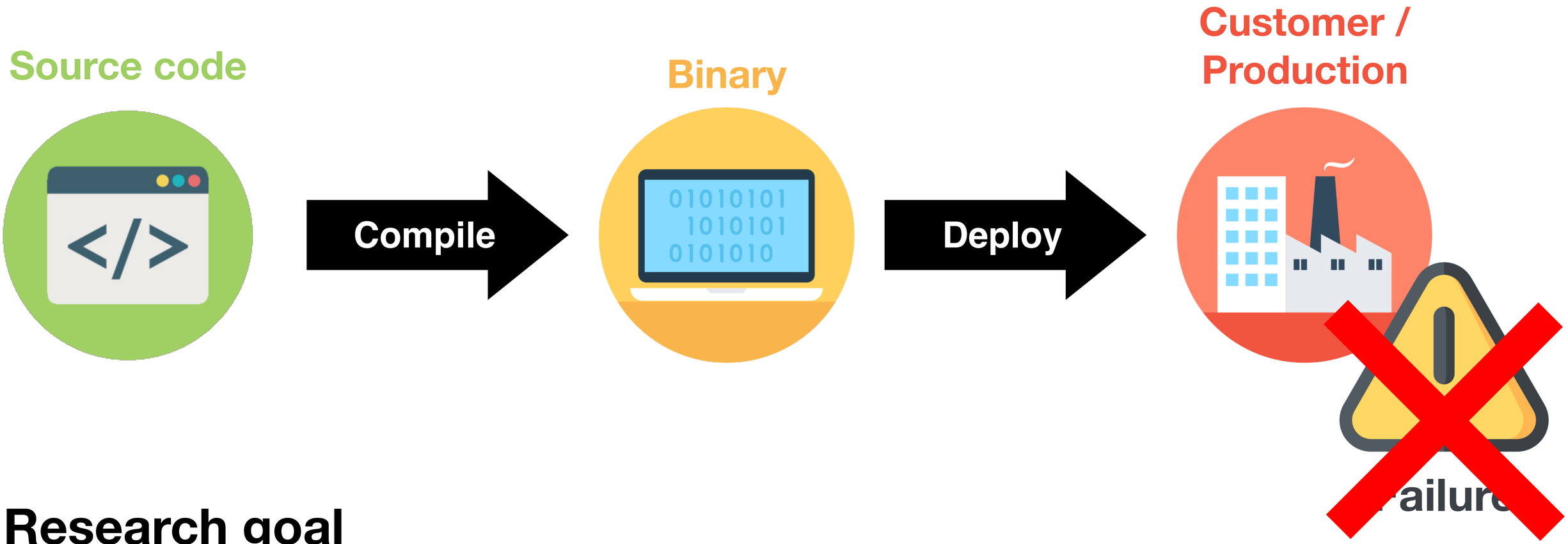
Compile

Binary

Deploy

Customer / Production

Failure



Research goal

Protect developers from security vulnerabilities, unsafe code, performance bottlenecks, and specification violations

Source code

Compile

Binary

Deploy

Customer / Production

Failure

Source code

Compile

Binary

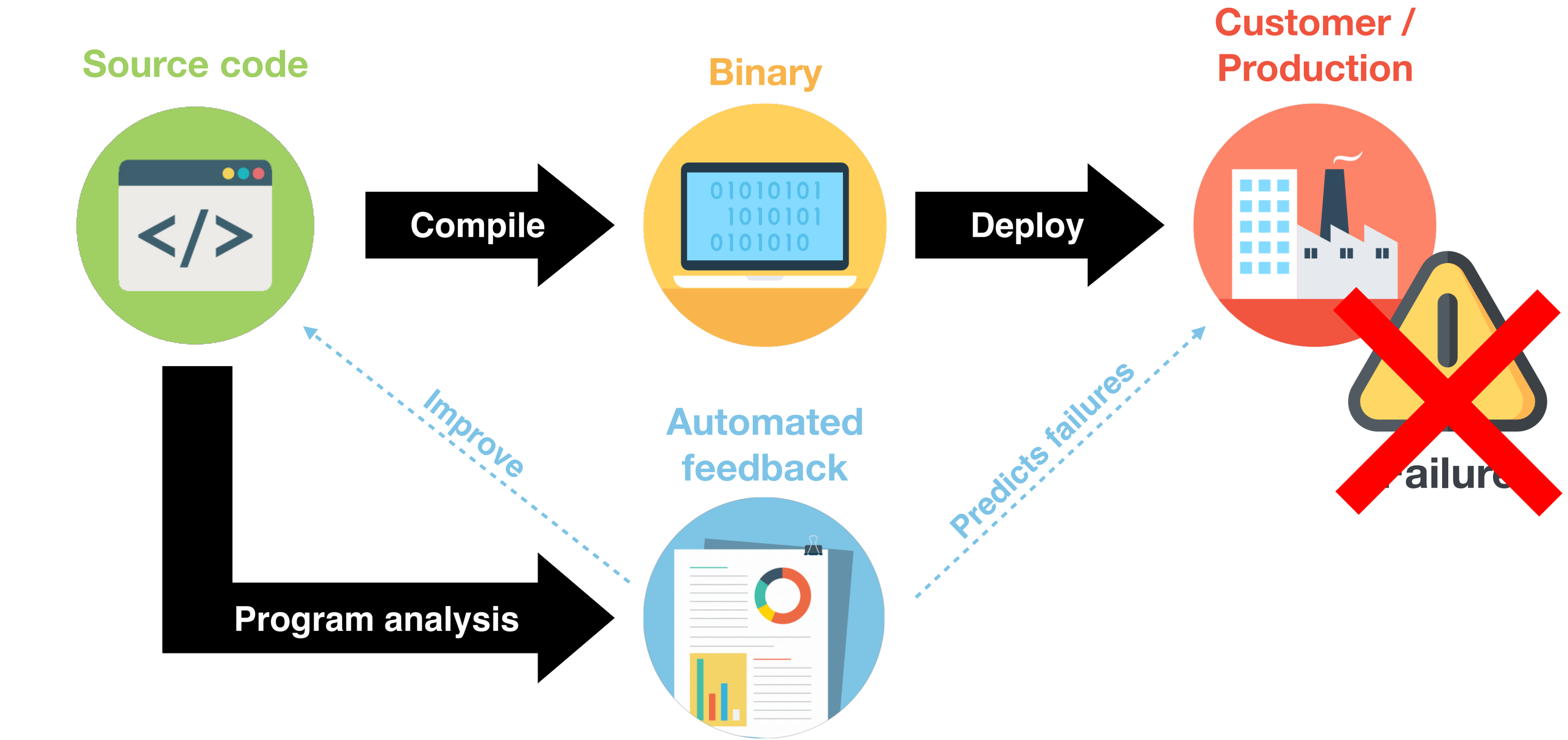
Deploy

Customer / Production

Failure

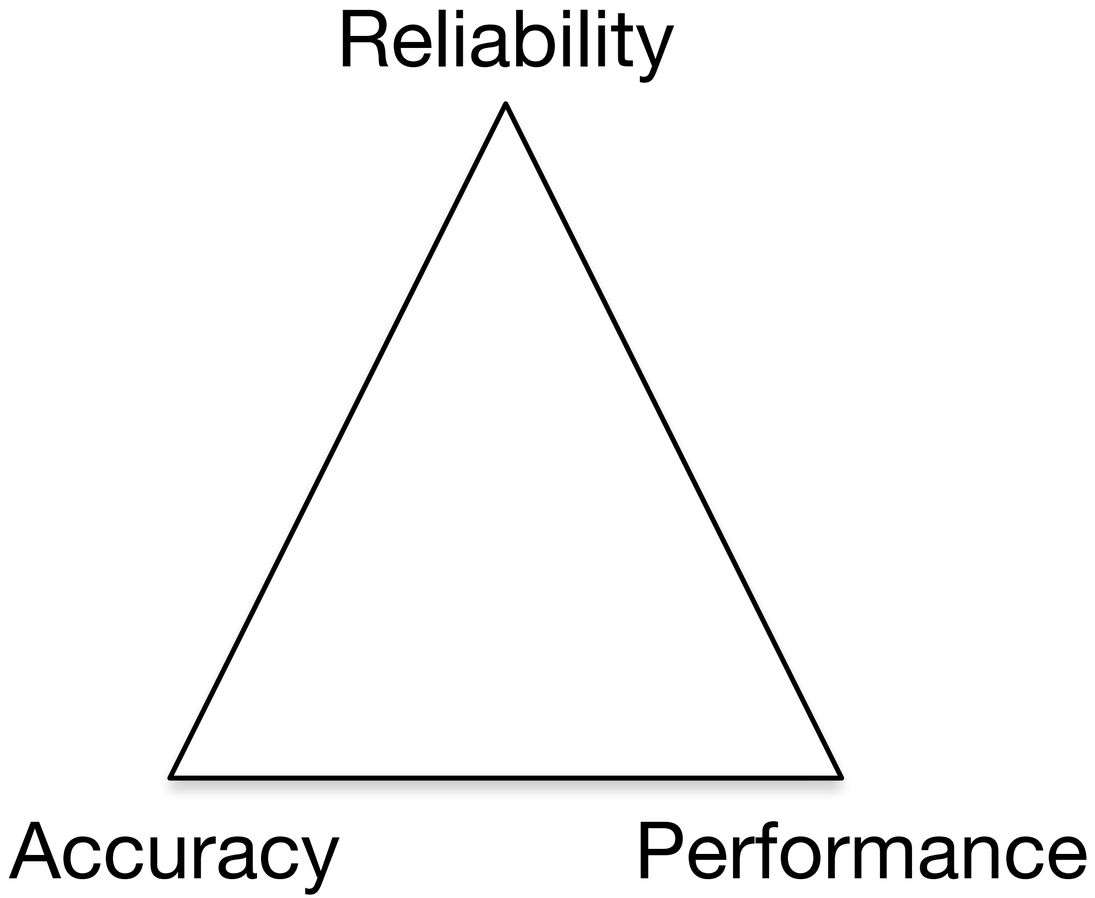
Program analysis

Automated feedback


Source code
Binary
Customer / Production
Compile
Deploy
Improve
Automated feedback
Predicts failures
Program analysis

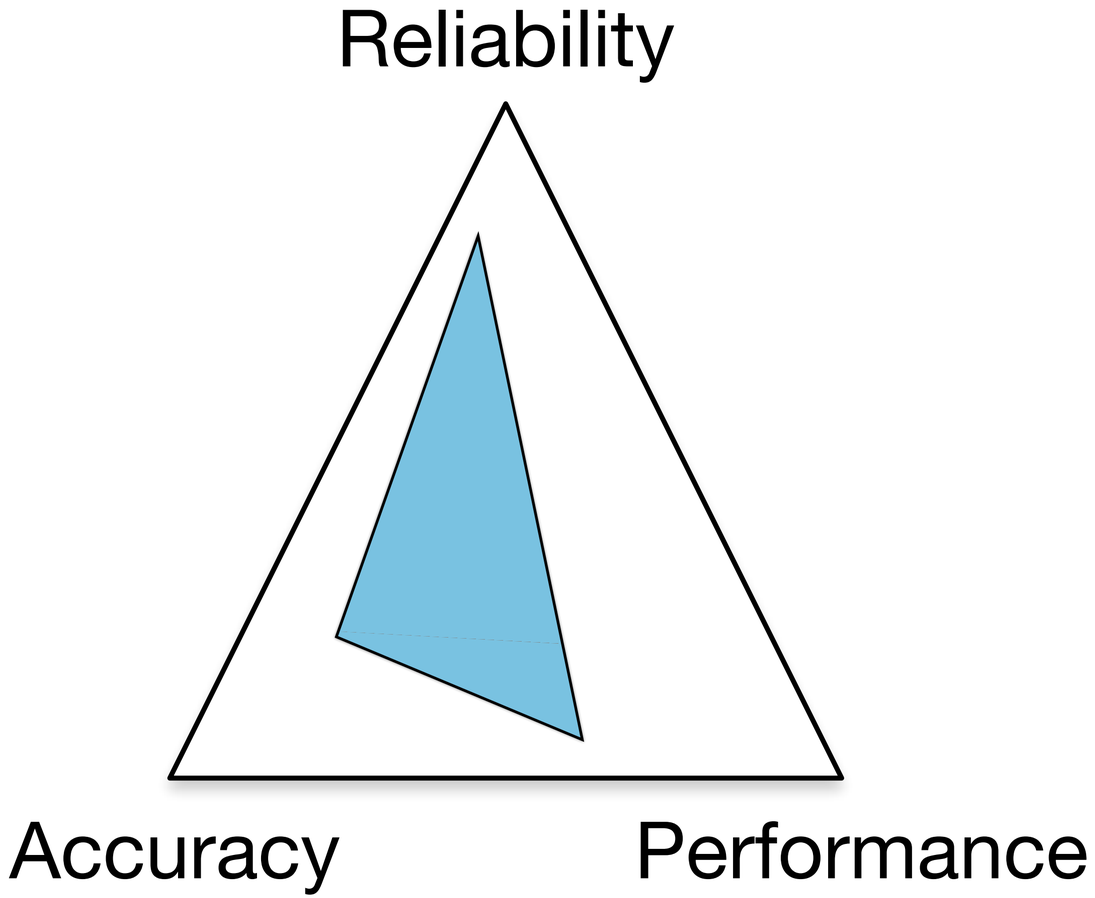
Program analysis trade-offs

More reliability = less missed bugs
More accuracy = less false alarms
More performance = faster feedback



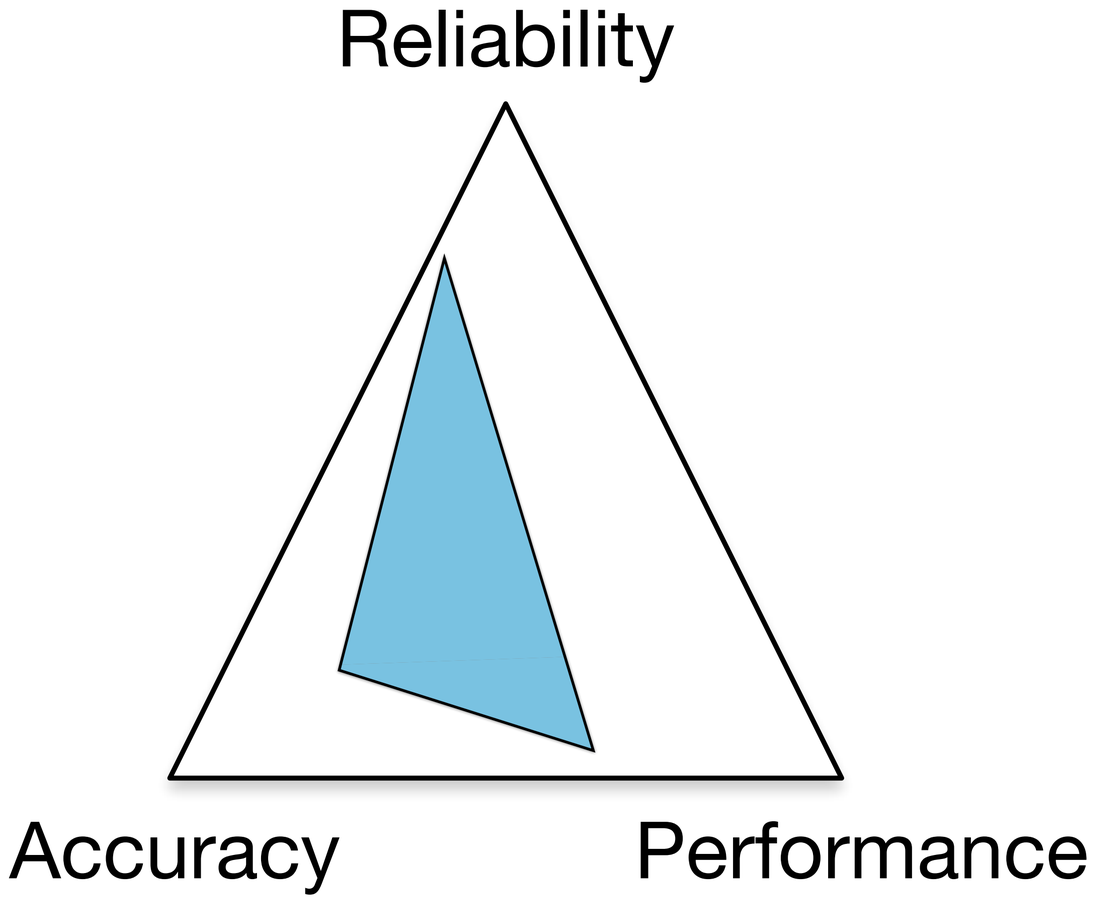
Program analysis trade-offs

More reliability = less missed bugs
More accuracy = less false alarms
More performance = faster feedback



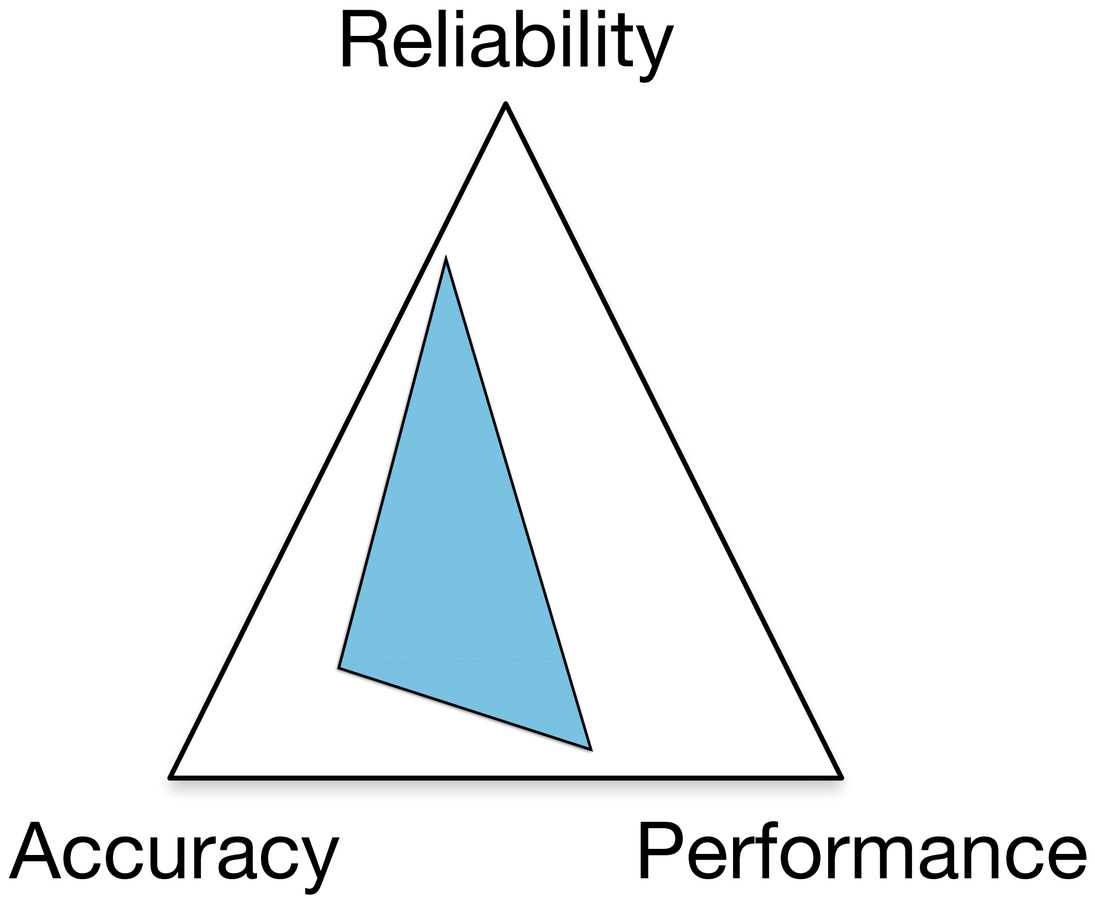
Program analysis trade-offs

More reliability = less missed bugs
More accuracy = less false alarms
More performance = faster feedback



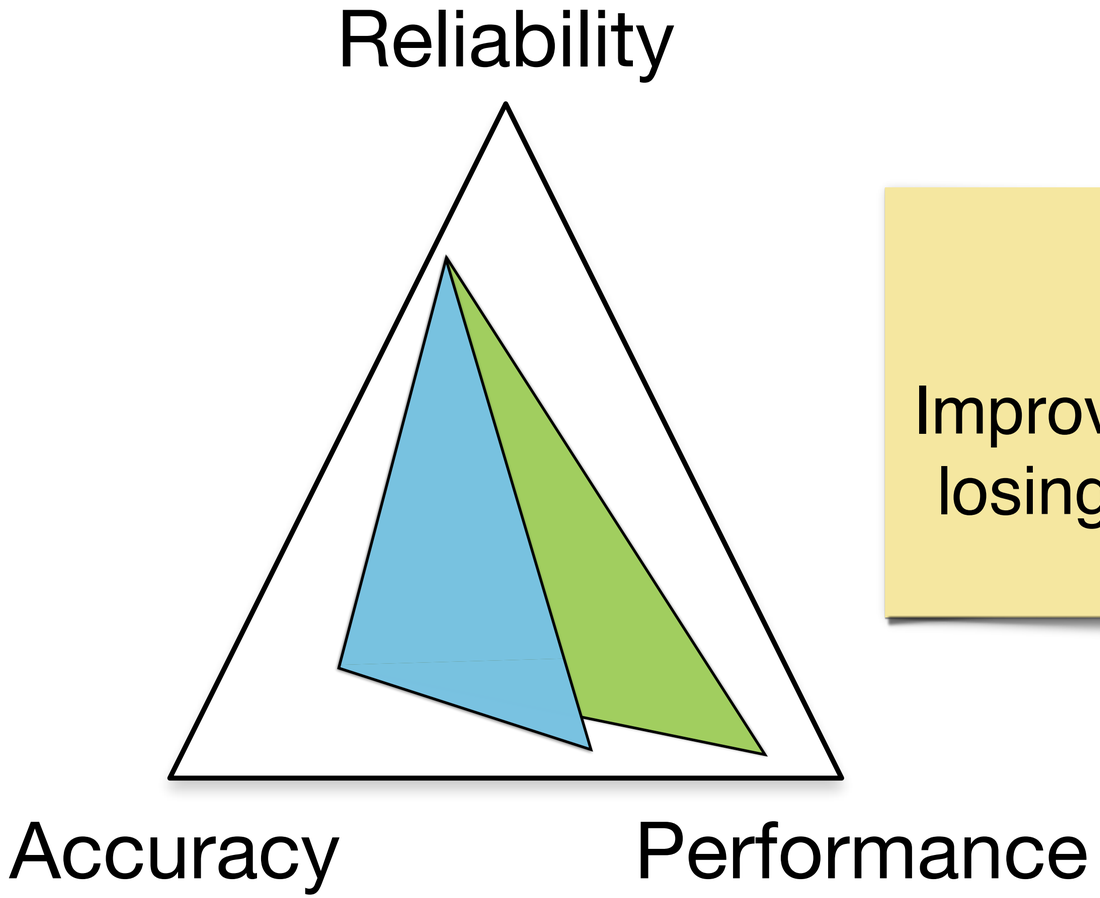
Program analysis trade-offs

More reliability = less missed bugs
More accuracy = less false alarms
More performance = faster feedback



Program analysis trade-offs

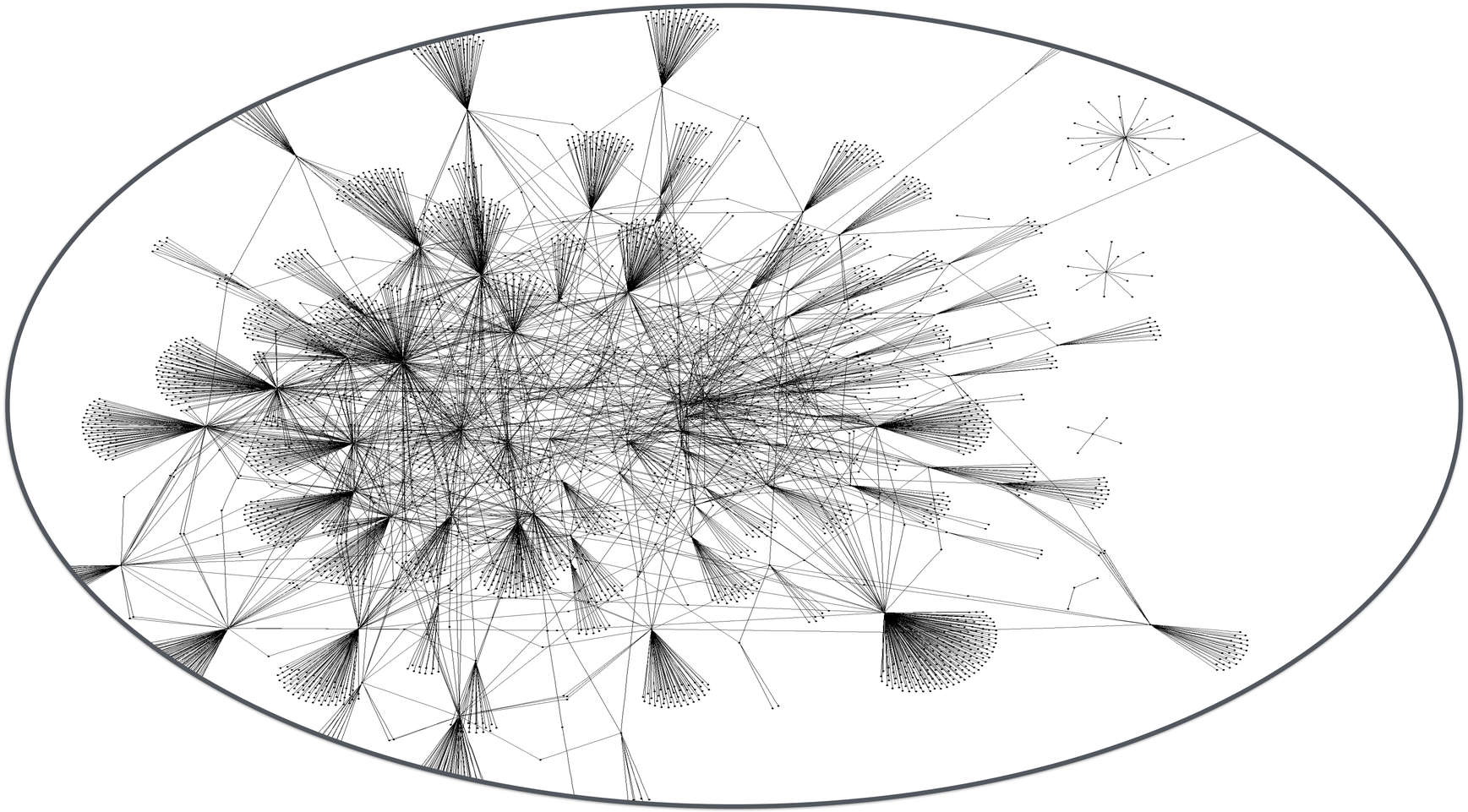
More reliability = less missed bugs
More accuracy = less false alarms
More performance = faster feedback



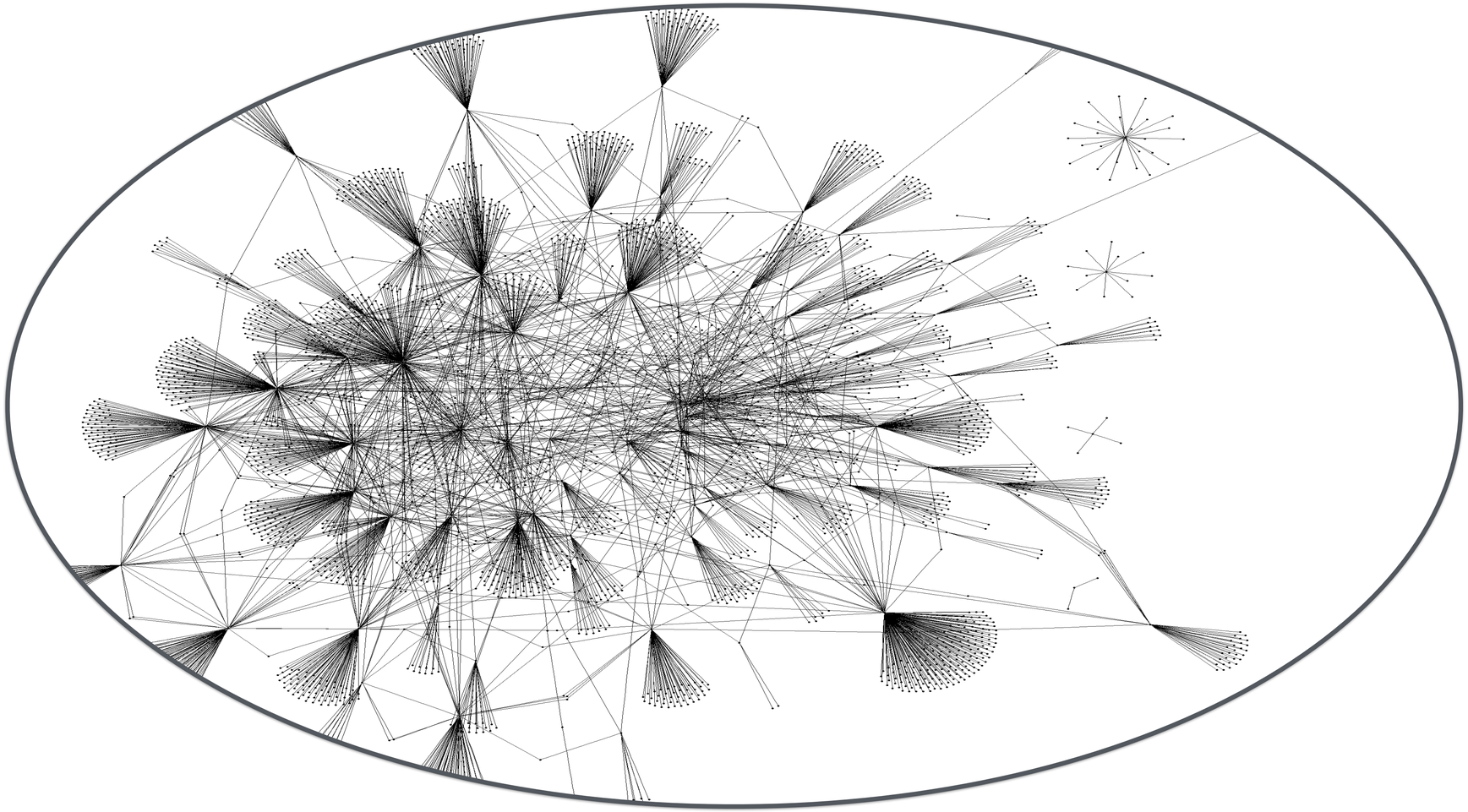
Goal

Improve performance without losing accuracy or reliability

# Code (OpenSSL = 485k SLoC
Firefox = 35.8m SLoC)



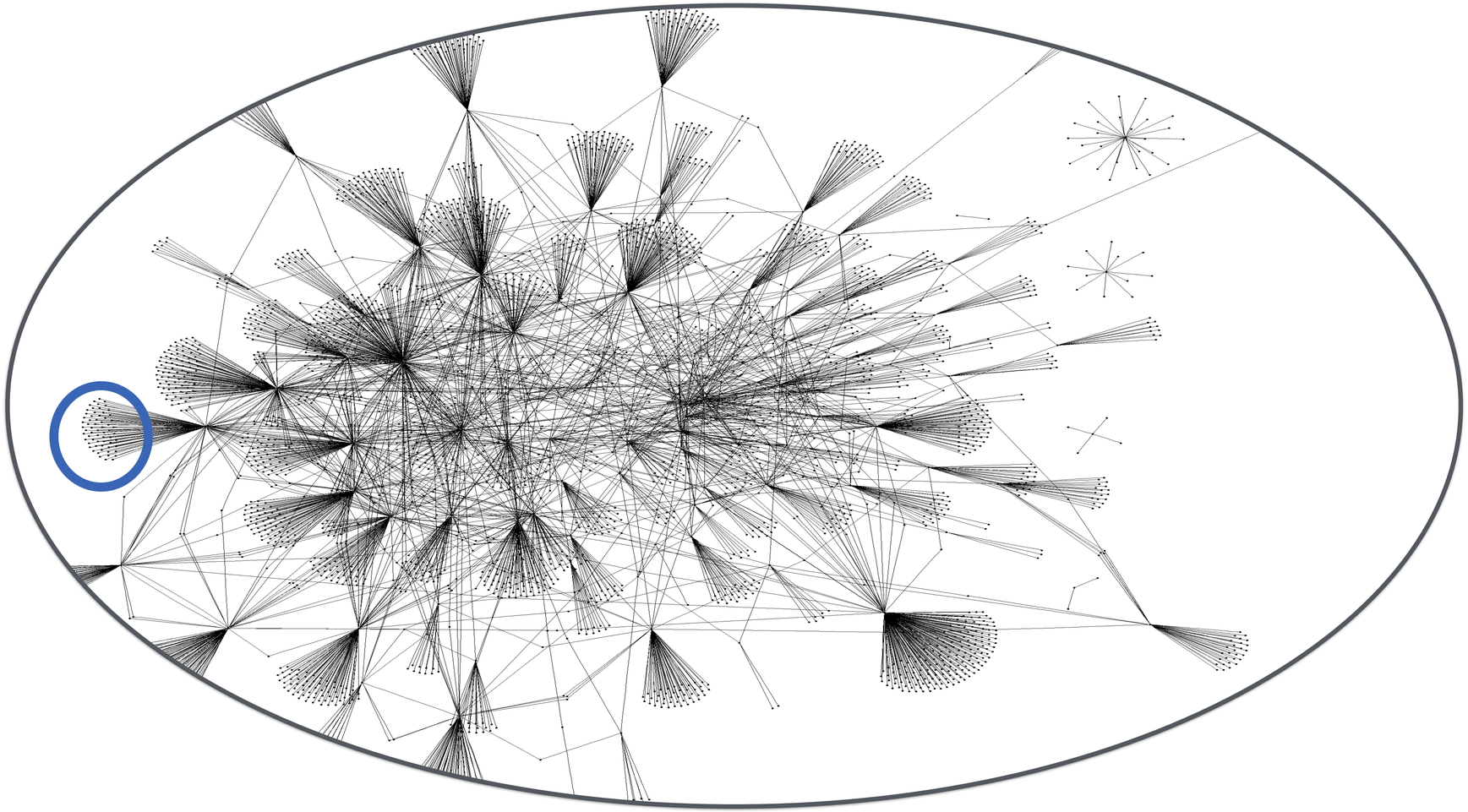
# Code (OpenSSL = 485k SLoC
Firefox = 35.8m SLoC)



Incremental analysis: only re-analyze code affected by change

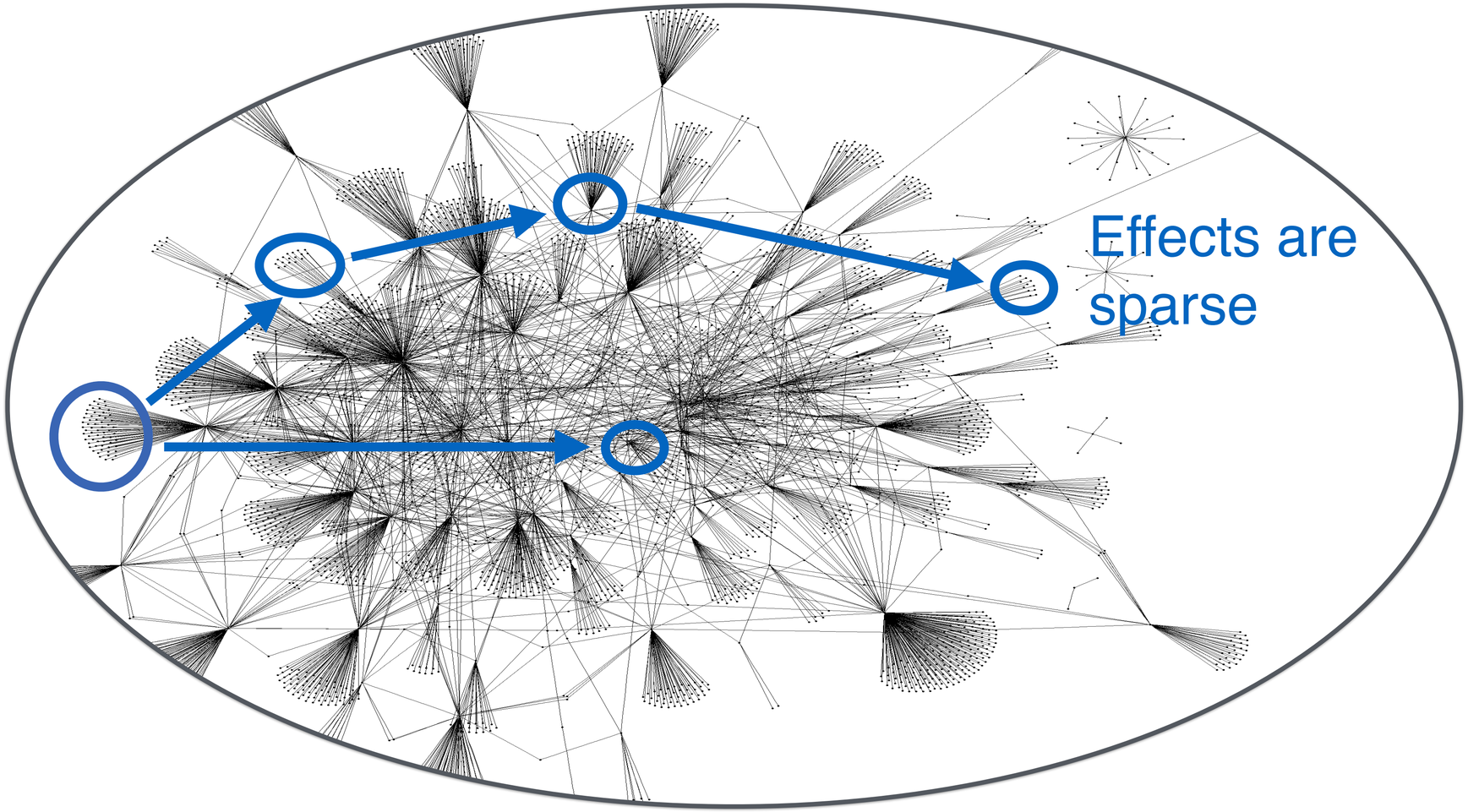
# Code (OpenSSL = 485k SLoC
Firefox = 35.8m SLoC)

Changes are local

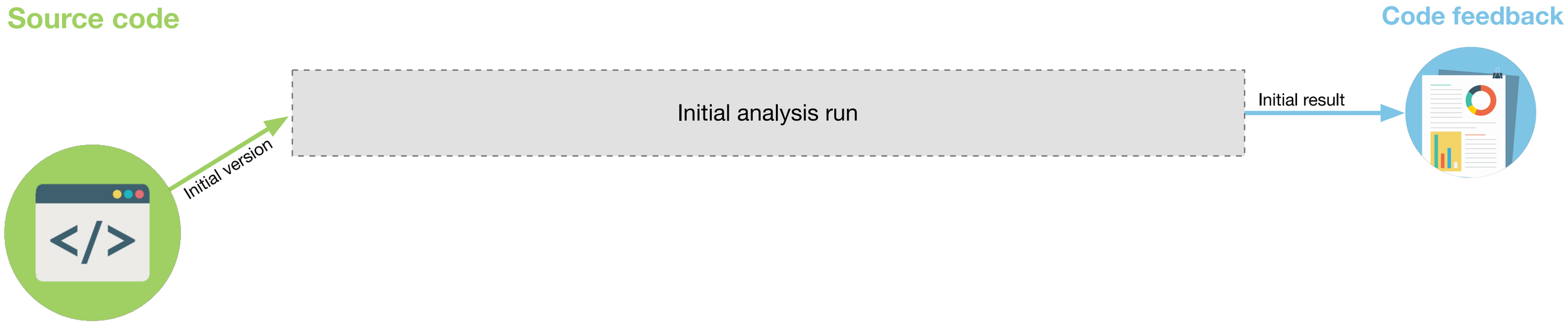


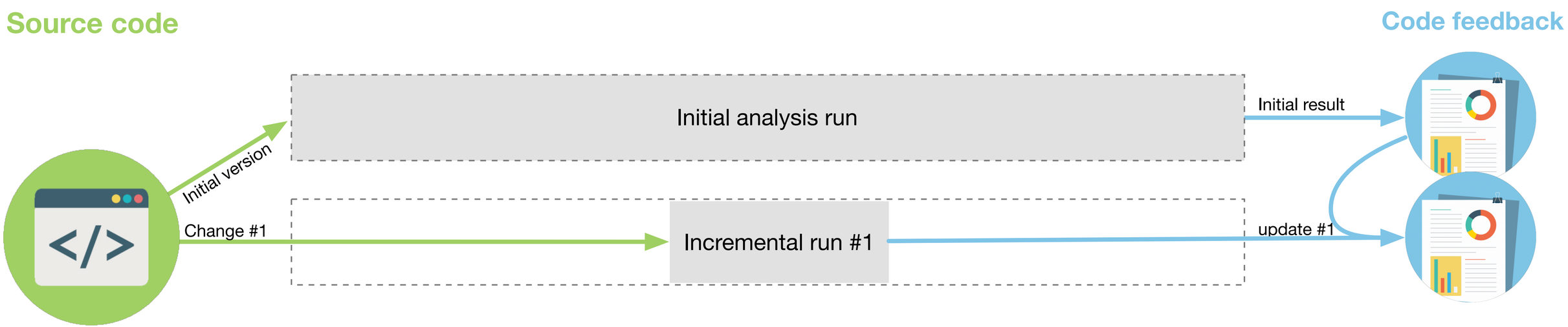
Incremental analysis: only re-analyze code affected by change

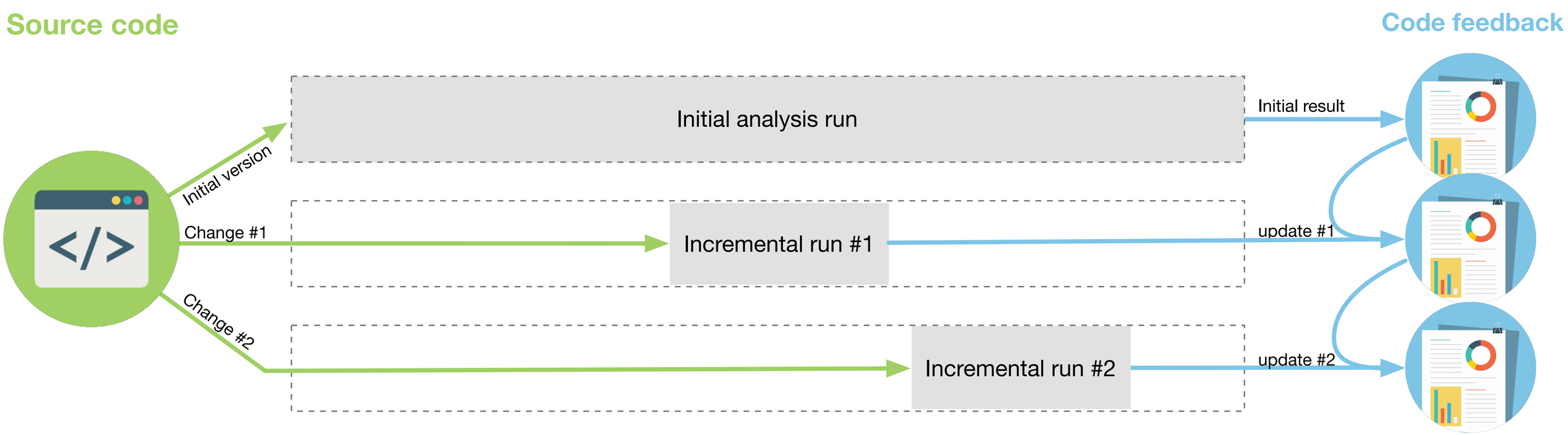
# Code (OpenSSL = 485k SLoC
Firefox = 35.8m SLoC)

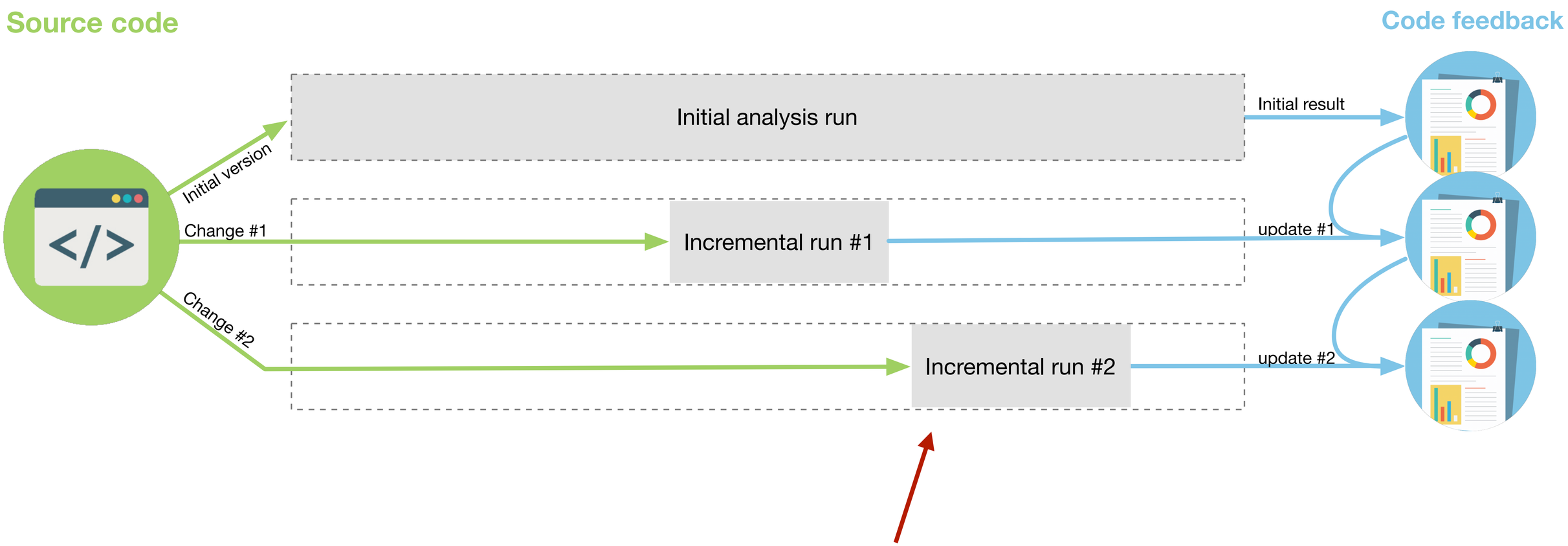


Incremental analysis: only re-analyze code affected by change

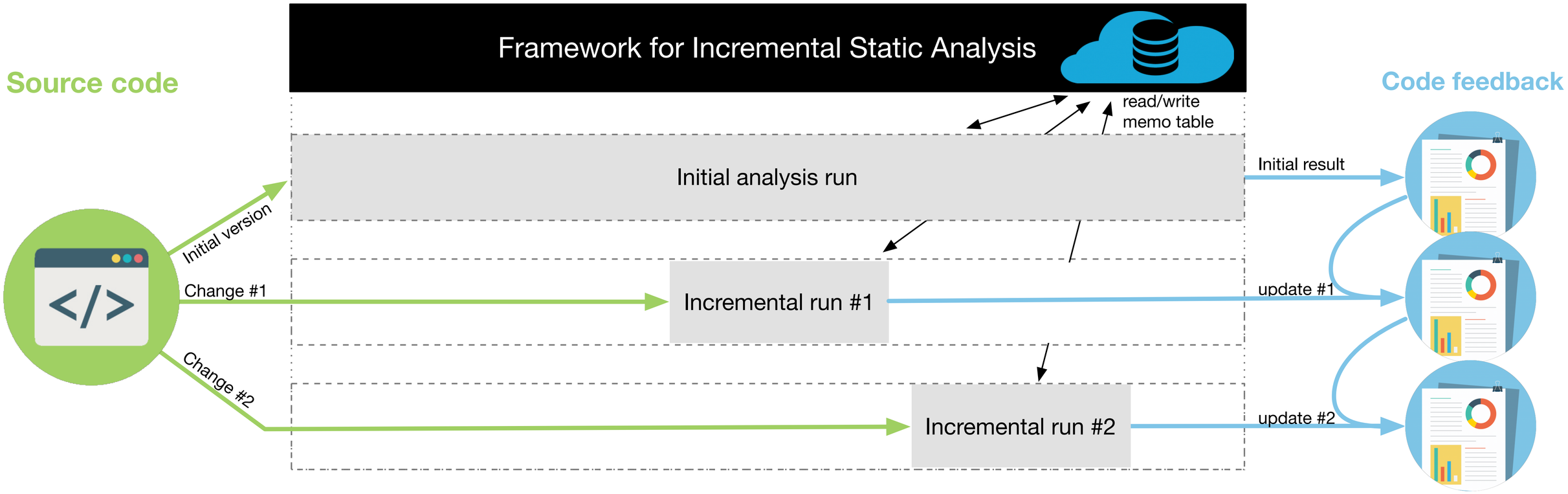

Source code
Code feedback
Initial analysis run
Initial version
Initial result


Source code
Code feedback
Initial analysis run
Initial result
Initial version
Change #1
Incremental run #1
update #1


Source code
Code feedback
Initial analysis run
Initial result
Initial version
Change #1
Incremental run #1
update #1
Change #2
Incremental run #2
update #2



Asymptotical performance improvements in theory
10x-10000x speedup observable in practice



Asymptotical performance improvements in theory
10x-10000x speedup observable in practice

Framework for Incremental Static Analysis

The IncA approach:

- Store code in deductive database
- Formulate analyses as queries against database
- Incrementalize database queries

Code = Base relations

```
(1) x = 7, y = 0;
(2) while (read()) {
(3)    x = 9;
(4)    x = x + 2;
(5)    y = cond ? y : y + 1; }
```

Code = Base relations

```
(1) x = 7, y = 0;
(2) while (read()) {
(3)    x = 9;
(4)    x = x + 2;
(5)    y = cond ? y : y + 1; }
```

```
Stmt = {1,2,3,4,5}
Assign = {1,3,4,5}
While = {2}
Next = {(1,2),(3,4),(4,5)}
Parent = {(3,2),(4,2),(5,2)}

expressions, functions, …
```

Analysis = Query

```
(1) x = 7, y = 0;
(2) while (read()) {
(3)    x = 9;
(4)    x = x + 2;
(5)    y = cond ? y : y + 1; }
```

Analysis = Query

```
(1) x = 7, y = 0;
(2) while (read()) {
(3)    x = 9;
(4)    x = x + 2;
(5)    y = cond ? y : y + 1; }
```

```
CFlow(from, to) :- Next(from, to), ¬IfElseStmt(from).
CFlow(w, start) :- WhileStmt(w), FirstChild(w, start).
CFlow(last, w)  :- WhileStmt(w), LastChild(w, last).
...
```

Analysis = Query

```
(1) x = 7, y = 0;
(2) while (read()) {
(3)    x = 9;
(4)    x = x + 2;
(5)    y = cond ? y : y + 1; }
```

```
CFlow(from, to) :- Next(from, to), ¬IfElseStmt(from).
CFlow(w, start) :- WhileStmt(w), FirstChild(w, start).
CFlow(last, w)  :- WhileStmt(w), LastChild(w, last).
...
```

```
CFlow = {(1,2),        (3,4),(4,5)        }
```

Analysis = Query

```
(1) x = 7, y = 0;
(2) while (read()) {
(3)    x = 9;
(4)    x = x + 2;
(5)    y = cond ? y : y + 1; }
```

```
CFlow(from, to) :- Next(from, to), ¬IfElseStmt(from).
CFlow(w, start) :- WhileStmt(w), FirstChild(w, start).
CFlow(last, w)  :- WhileStmt(w), LastChild(w, last).
...
```

```
CFlow = {(1,2),(2,3),(3,4),(4,5)          }
```

Analysis = Query

```
(1) x = 7, y = 0;
(2) while (read()) {
(3)    x = 9;
(4)    x = x + 2;
(5)    y = cond ? y : y + 1; }
```

```
CFlow(from, to) :- Next(from, to), ¬IfElseStmt(from).
CFlow(w, start) :- WhileStmt(w), FirstChild(w, start).
CFlow(last, w)  :- WhileStmt(w), LastChild(w, last).
...
```

```
CFlow = {(1,2),(2,3),(3,4),(4,5),(5,2)}
```

Incrementalization = Change propagation

```
(1)  x = 7, y = 0;
(2)  while if (read()) {
(3)     x = 9;
(4)     x = x + 2;
(5)     y = cond ? y : y + 1; }
```

```
- WhileStmt(2)
+ IfStmt(2)
```

```
CFlow(from, to)  :- Next(from, to), ¬IfElseStmt(from).
CFlow(w, start)  :- WhileStmt(w), FirstChild(w, start).
CFlow(last, w)   :- WhileStmt(w), LastChild(w, last).
...
```

```
CFlow = {(1,2),(2,3),(3,4),(4,5),(5,2)          }
```

Incrementalization = Change propagation

```
(1)  x = 7, y = 0;
(2)  while if (read()) {
(3)     x = 9;
(4)     x = x + 2;
(5)     y = cond ? y : y + 1; }
```

```
- WhileStmt(2)
+ IfStmt(2)
```

```
CFlow(from, to)  :- Next(from, to), ¬IfElseStmt(from).
CFlow(w, start)  :- WhileStmt(w), FirstChild(w, start).
CFlow(last, w)   :- WhileStmt(w), LastChild(w, last).
...
```

```
CFlow = {(1,2),(2,3),(3,4),(4,5),(5,2)          }
```

Incrementalization = Change propagation

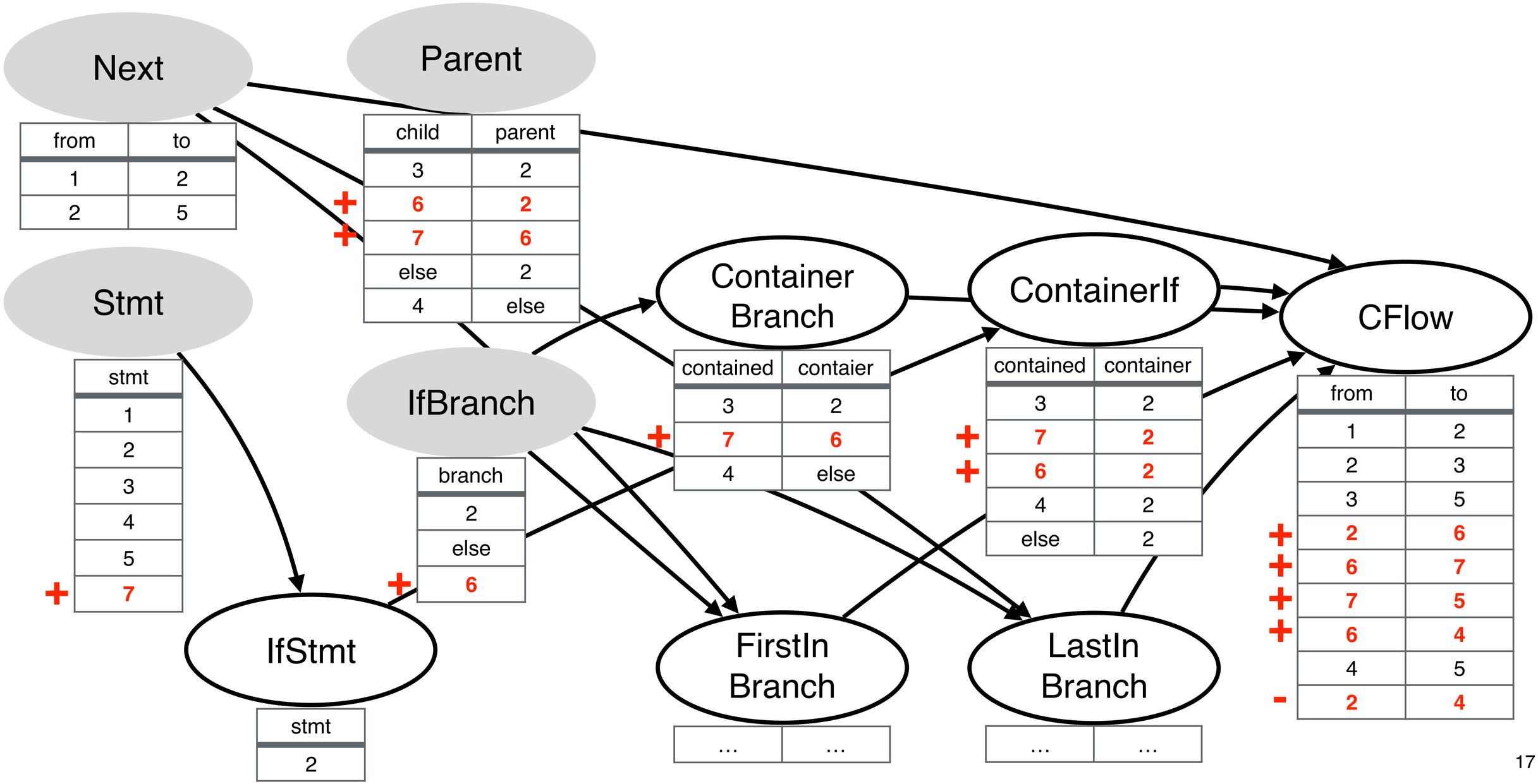
```
(1)  x = 7, y = 0;
(2)  ~~while~~ if (read()) {
(3)     x = 9;
(4)     x = x + 2;
(5)     y = cond ? y : y + 1; }
```

```
- WhileStmt(2)
+ IfStmt(2)
```

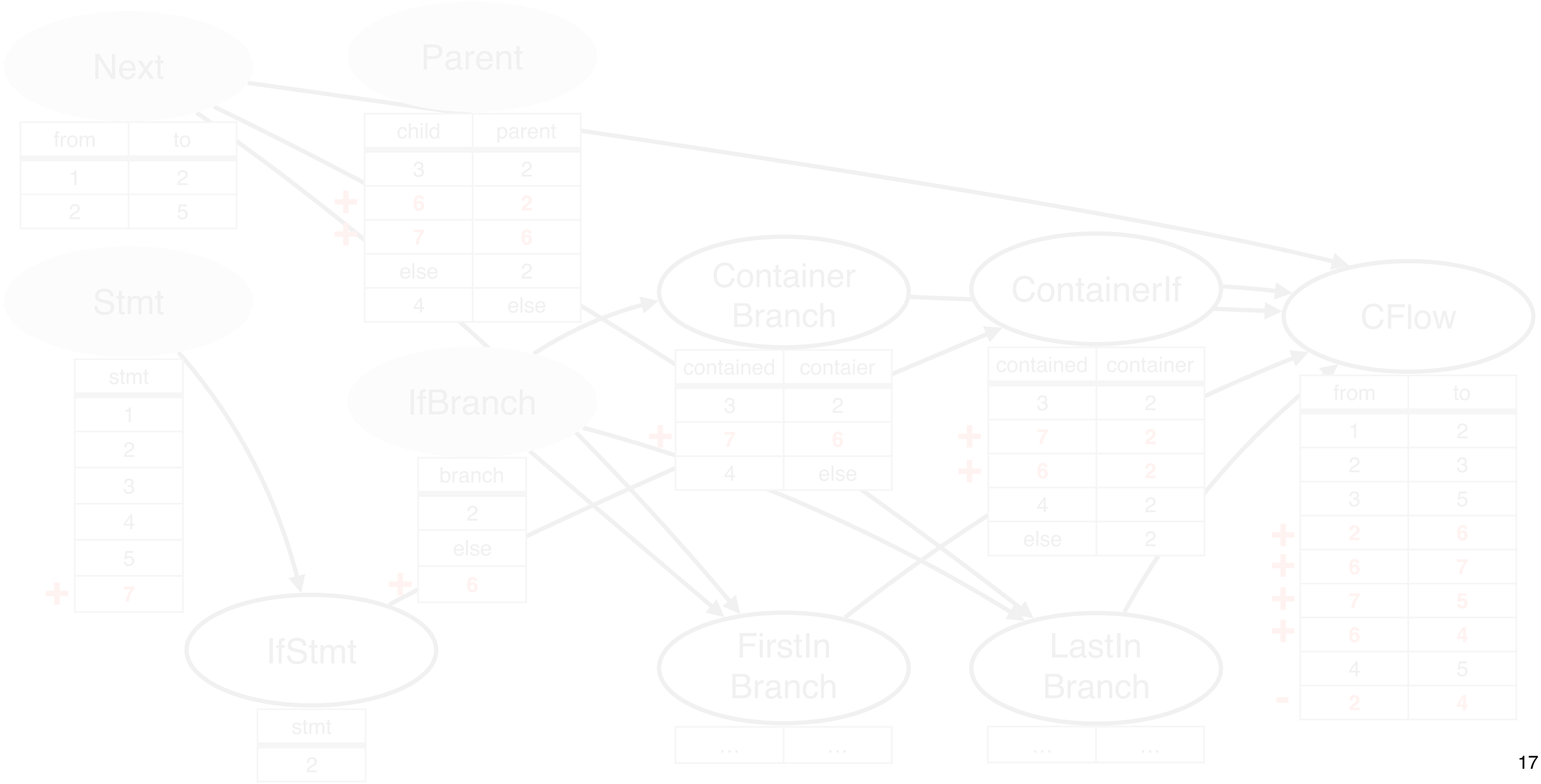
```
CFlow(from, to)  :- Next(from, to), ¬IfElseStmt(from).
CFlow(w, start)  :- WhileStmt(w), FirstChild(w, start).
CFlow(last, w)   :- WhileStmt(w), LastChild(w, last).
...
```

```
CFlow = {(1,2),~~(2,3)~~,(3,4),(4,5),~~(5,2)~~,(2,3)}
```

Technically: Computation Network



Technically: Computation Network



Abstraction: Provide regular language instead of DB queries

```
protected def resolveClassLocally(type : Type, module : Module) : ClassDec = {
  foreach clazz in module.classes {
    if (clazz.type:CName.name == type:CName.name) { return clazz }
  }
}
```

```
if (def resolveClassLocally(type, module)) {
  return resolveClassLocally(type, module)
} else {
  foreach _import in module.imports {
    return resolveClass(type, _import.module)
  }
}
```

```
match expression with {
  case Var(name:"this") =>
    method := getContainingMethodDec(expression)
    clazz := method.parent
    assert clazz instanceOf ClassDec
    return clazz.type
  case Var(name:x) =>
    method := getContainingStaticMethodDec(expression)
    parameter := method.params
    assert parameter.name == x
    return parameter.type
```

Goal: Feedback within 100ms

Goal: Feedback within 100ms

< 25ms feedback

[OOPSLA'14, ASE'16]

FindBugs

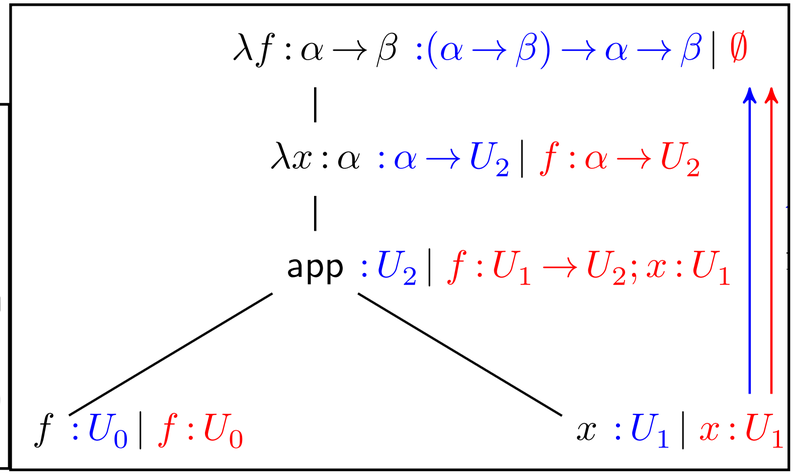
Goal: Feedback within 100ms

< 10ms feedback [OOPSLA'15, ECOOP'17]

Type checking: Java, Rust

< 25ms feedback [OOPSLA'14,ASE'16]

FindBugs



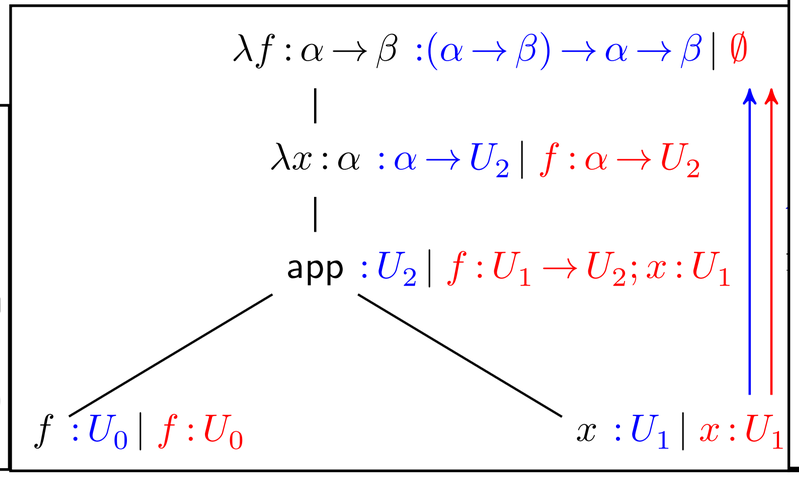
Goal: Feedback within 100ms

< 25ms feedback [OOPSLA'14,ASE'16]

FindBugs

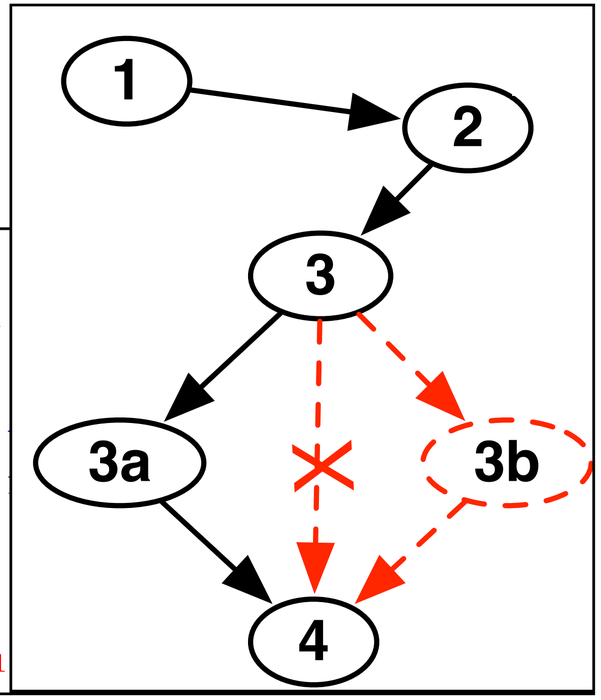
< 10ms feedback [OOPSLA'15, ECOOP'17]

Type checking: Java, Rust



< 30ms feedback [ASE'16]

Control flow: C, Java



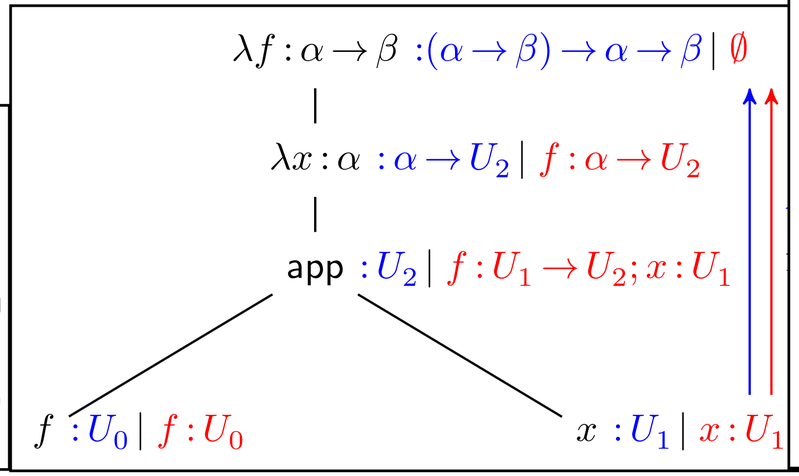
Goal: Feedback within 100ms

< 25ms feedback [OOPSLA'14,ASE'16]

FindBugs

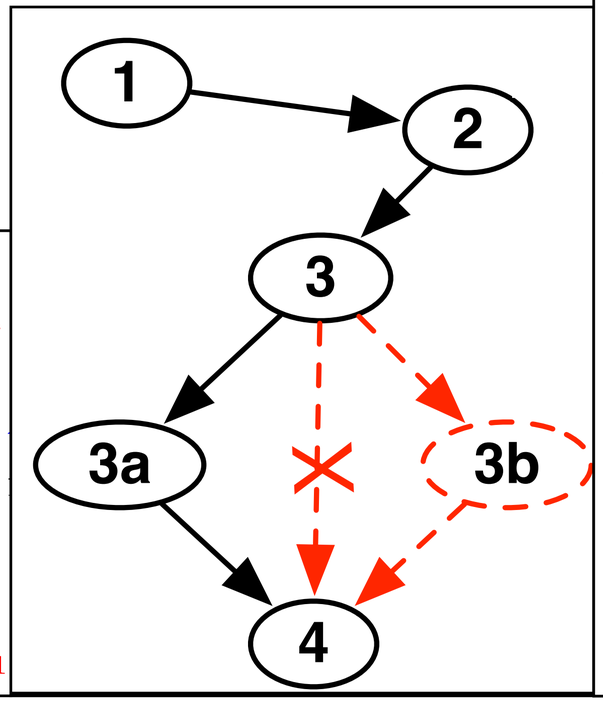
< 10ms feedback [OOPSLA'15, ECOOP'17]

Type checking: Java, Rust



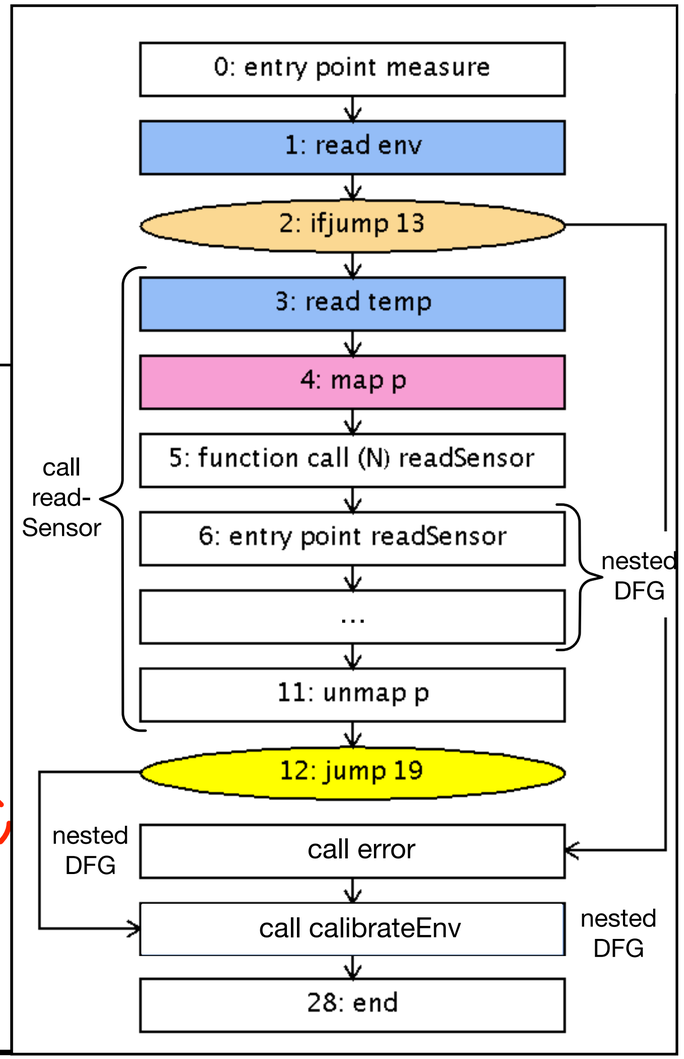
< 30ms feedback [ASE'16]

Control flow: C, Java



< 20ms feedback [OOPSLA'18]

Data flow: Java



Challenges: scalability, memory, expressiveness

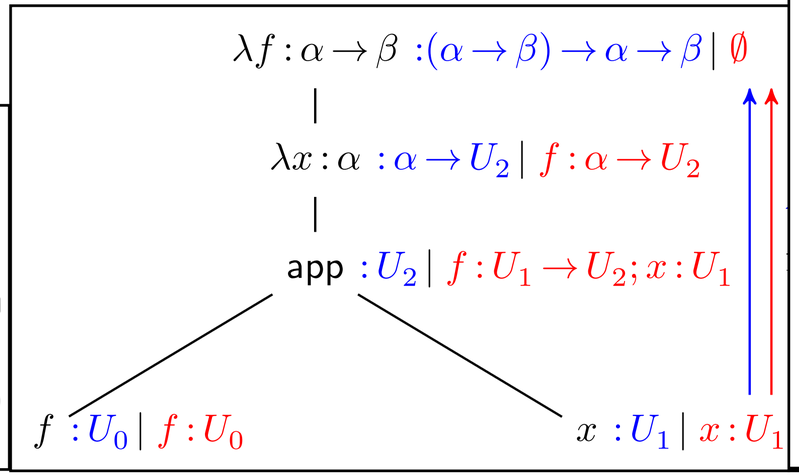
Goal: Feedback within 100ms

< 25ms feedback [OOPSLA'14,ASE'16]

FindBugs

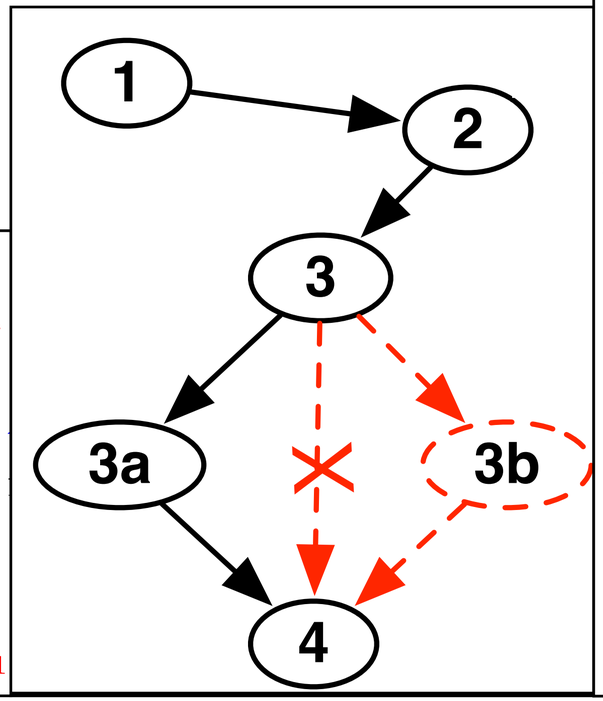
< 10ms feedback [OOPSLA'15, ECOOP'17]

Type checking: Java, Rust



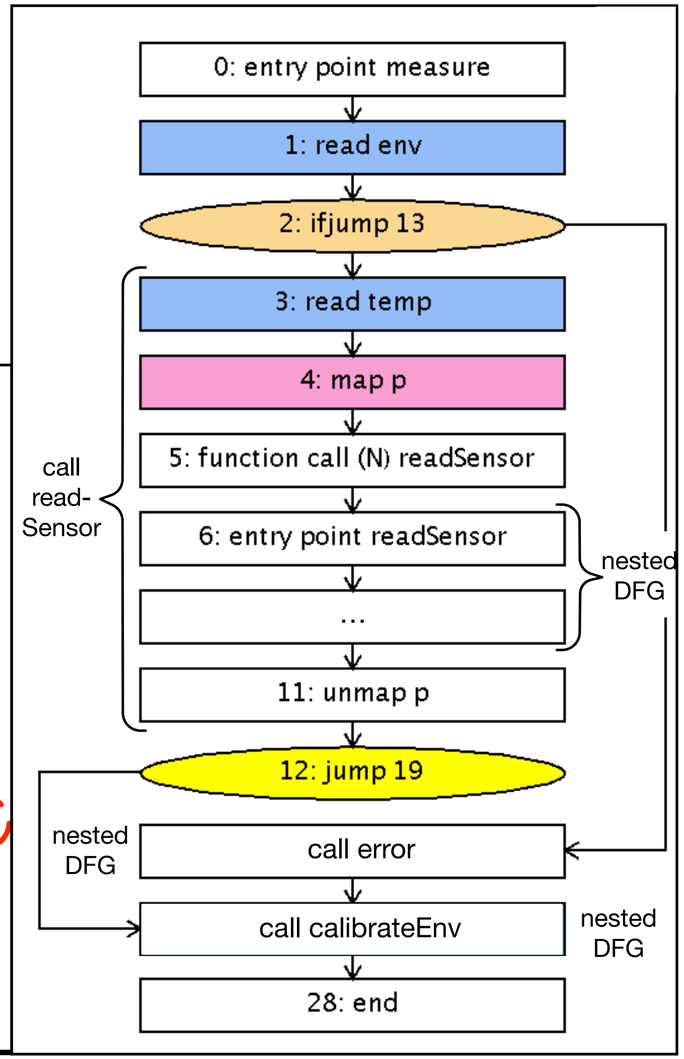
< 30ms feedback [ASE'16]

Control flow: C, Java



< 20ms feedback [OOPSLA'18]

Data flow: Java



Source code

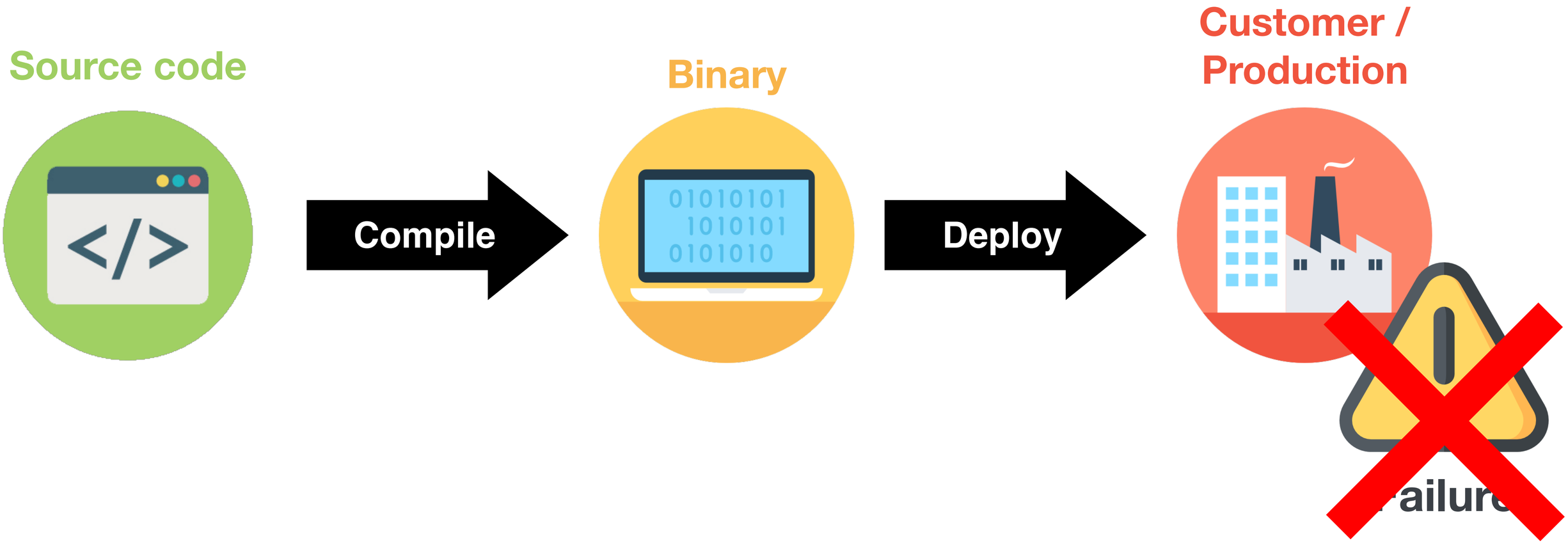
Compile

Binary

Deploy

Customer / Production

Failure



What to do with the time saved?

- Provide feedback faster – Improve adoption by developers
- Do more in same time – Improve accuracy and reliability