Testing Evolving Software

Alessandro (Alex) Orso

School of Computer Science - College of Computing
Georgia Institute of Technology
http://www.cc.gatech.edu/~orso/

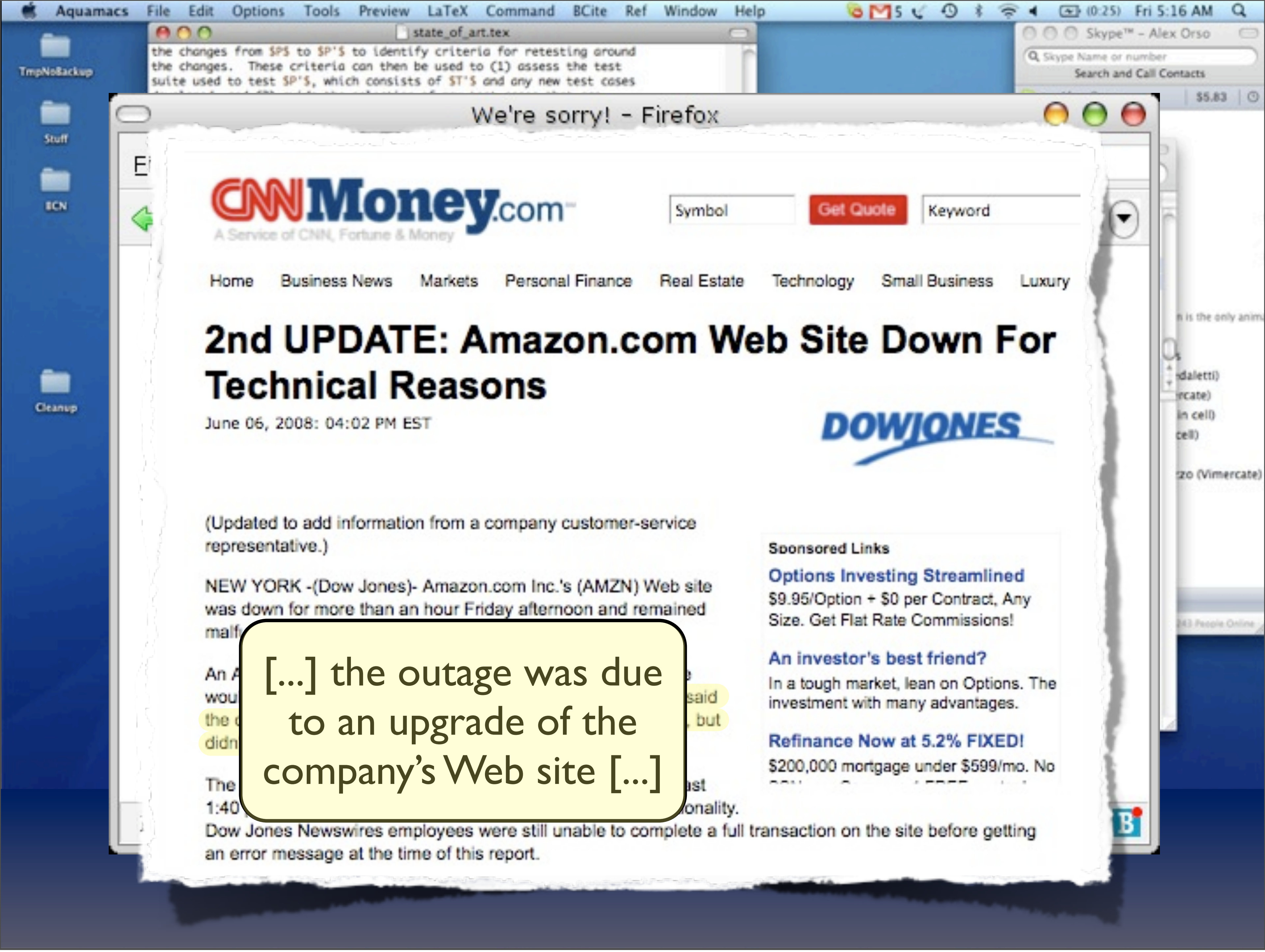
Partially supported by: NSF, IBM Research, TCS Ltd., Boeing Aerospace Corporation

Testing Evolving Software

Alessandro (Alex) Orso

Software Engineering
Static/Dynamic Program Analysis,
Software Testing, Security

Partially supported by: NSF, IBM Research, TCS Ltd., Boeing Aerospace Corporation



```
the changes from $P$ to $P'$ to identify criteria for retesting around
the changes.  These criteria can then be used to (1) assess the test
suite used to test $P'$, which consists of $T'$ and any new test cases
```

2nd UPDATE: Amazon.com Web Site Down For Technical Reasons

June 06, 2008: 04:02 PM EST

(Updated to add information from a company customer-service representative.)

NEW YORK -(Dow Jones)- Amazon.com Inc.'s (AMZN) Web site was down for more than an hour Friday afternoon and remained malf

An A … wou … said the c … but didn

The … ast 1:40 … onality. Dow Jones Newswires employees were still unable to complete a full transaction on the site before getting an error message at the time of this report.

Sponsored Links

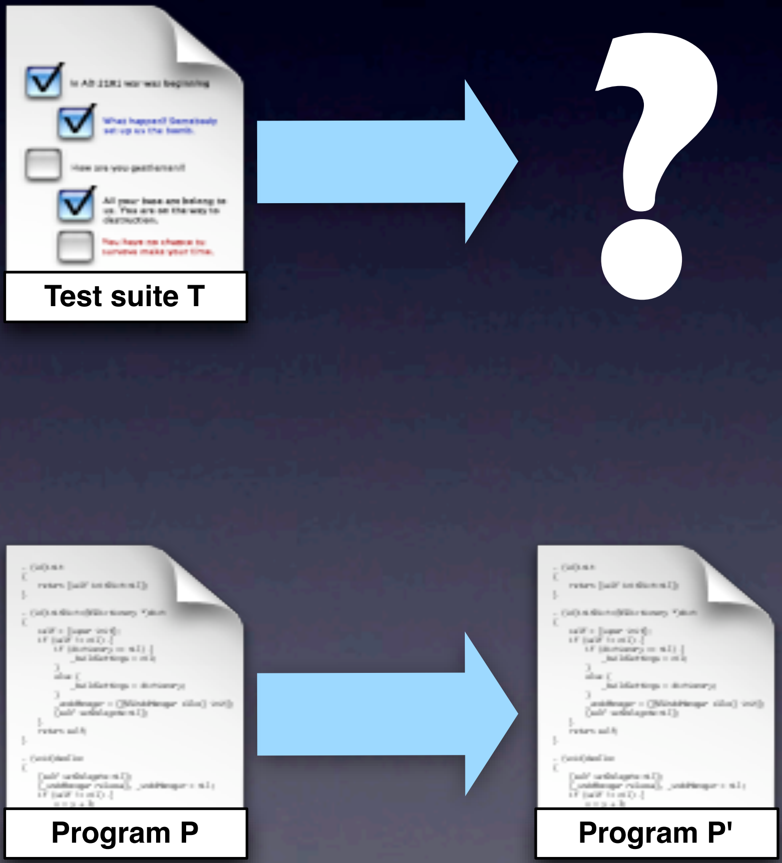
Options Investing Streamlined
$9.95/Option + $0 per Contract, Any Size. Get Flat Rate Commissions!

An investor's best friend?
In a tough market, lean on Options. The investment with many advantages.

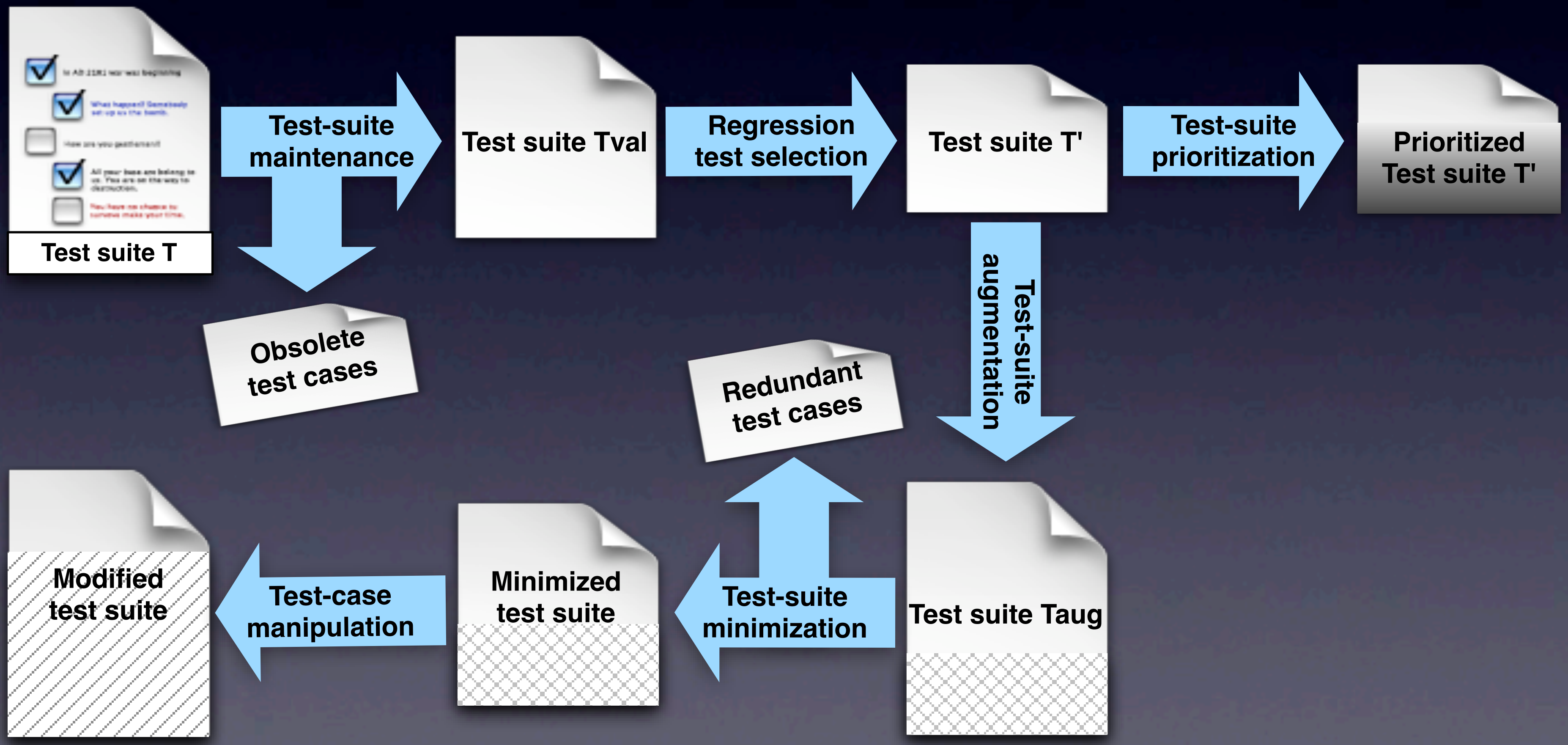
Refinance Now at 5.2% FIXED!
$200,000 mortgage under $599/mo. No

[...] the outage was due to an upgrade of the company's Web site [...]

Regression Testing Process and Issues



Regression Testing Process and Issues



Test suite T

Test-suite maintenance

Test suite Tval

Regression test selection

Test suite T'

Test-suite prioritization

Prioritized Test suite T'

Obsolete test cases

Test-suite augmentation

Redundant test cases

Test suite Taug

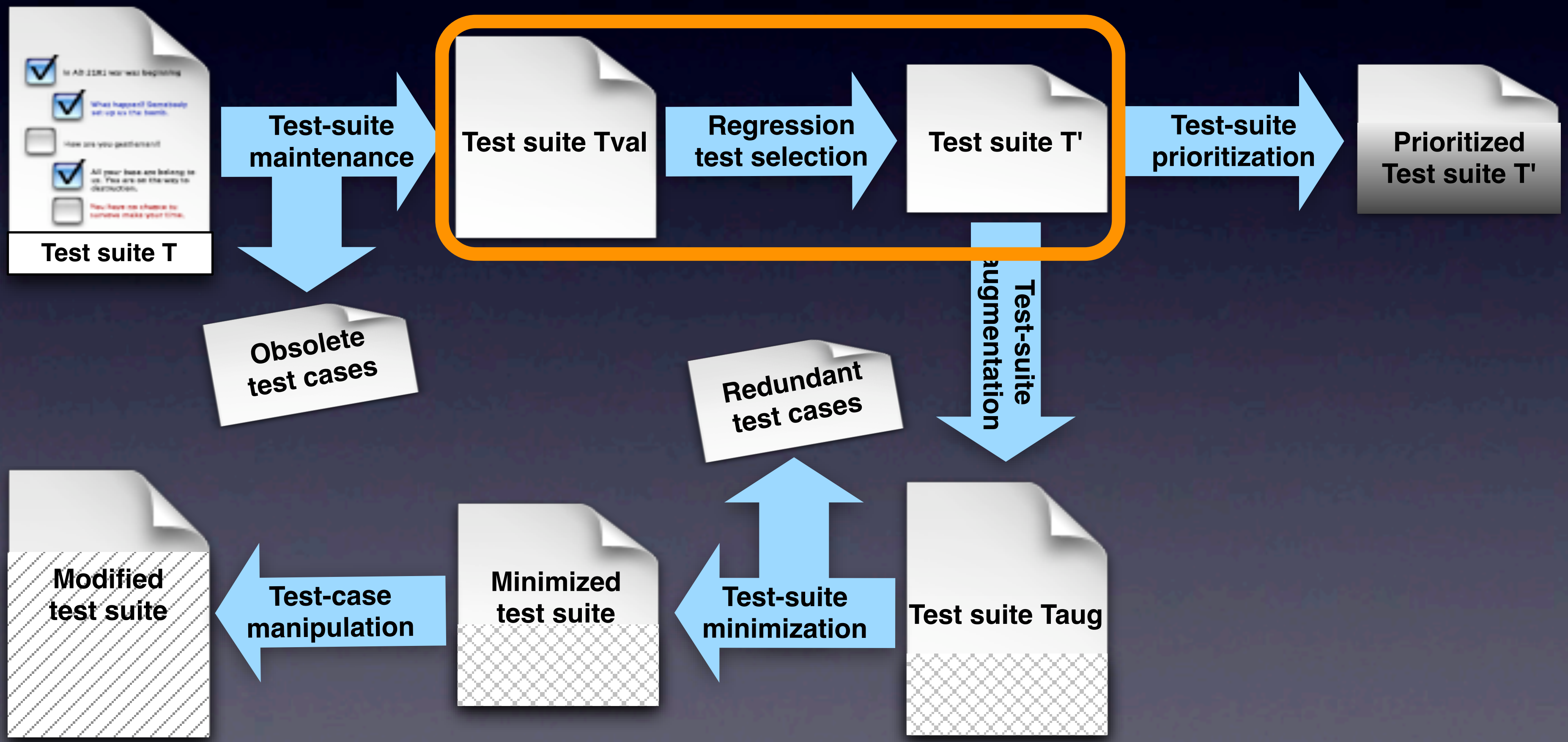
Test-suite minimization

Minimized test suite

Test-case manipulation

Modified test suite

Regression Testing Process and Issues



Test suite T

Test-suite maintenance

Obsolete test cases

Test suite Tval

Regression test selection

Test suite T'

Test-suite prioritization

Prioritized Test suite T'

Test-suite augmentation

Test suite Taug

Test-suite minimization

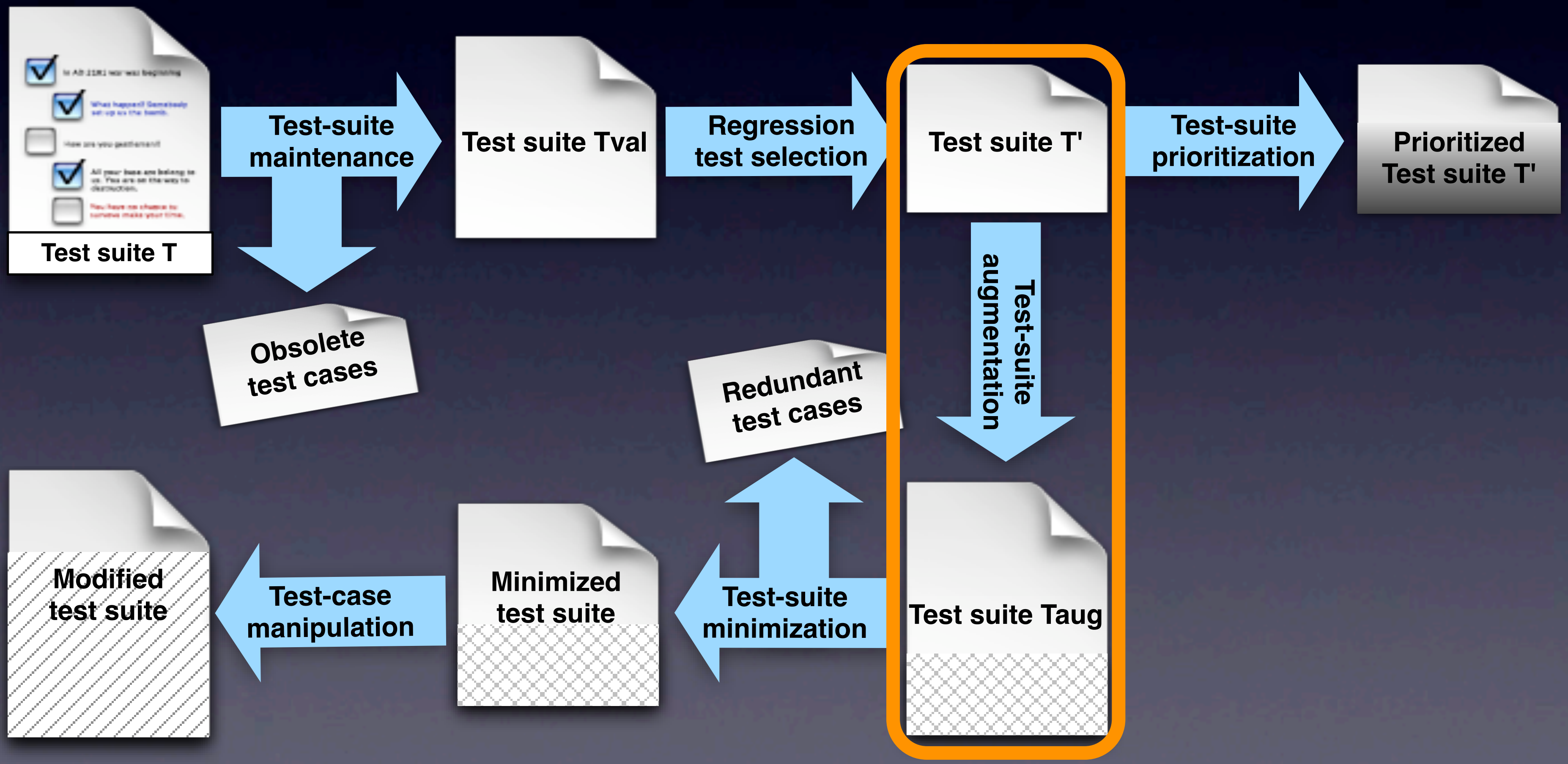
Redundant test cases

Minimized test suite

Test-case manipulation

Modified test suite

Regression Testing Process and Issues



Test suite T

Test-suite maintenance

Obsolete test cases

Test suite Tval

Regression test selection

Test suite T'

Test-suite prioritization

Prioritized Test suite T'

Test-suite augmentation

Test suite Taug

Test-suite minimization

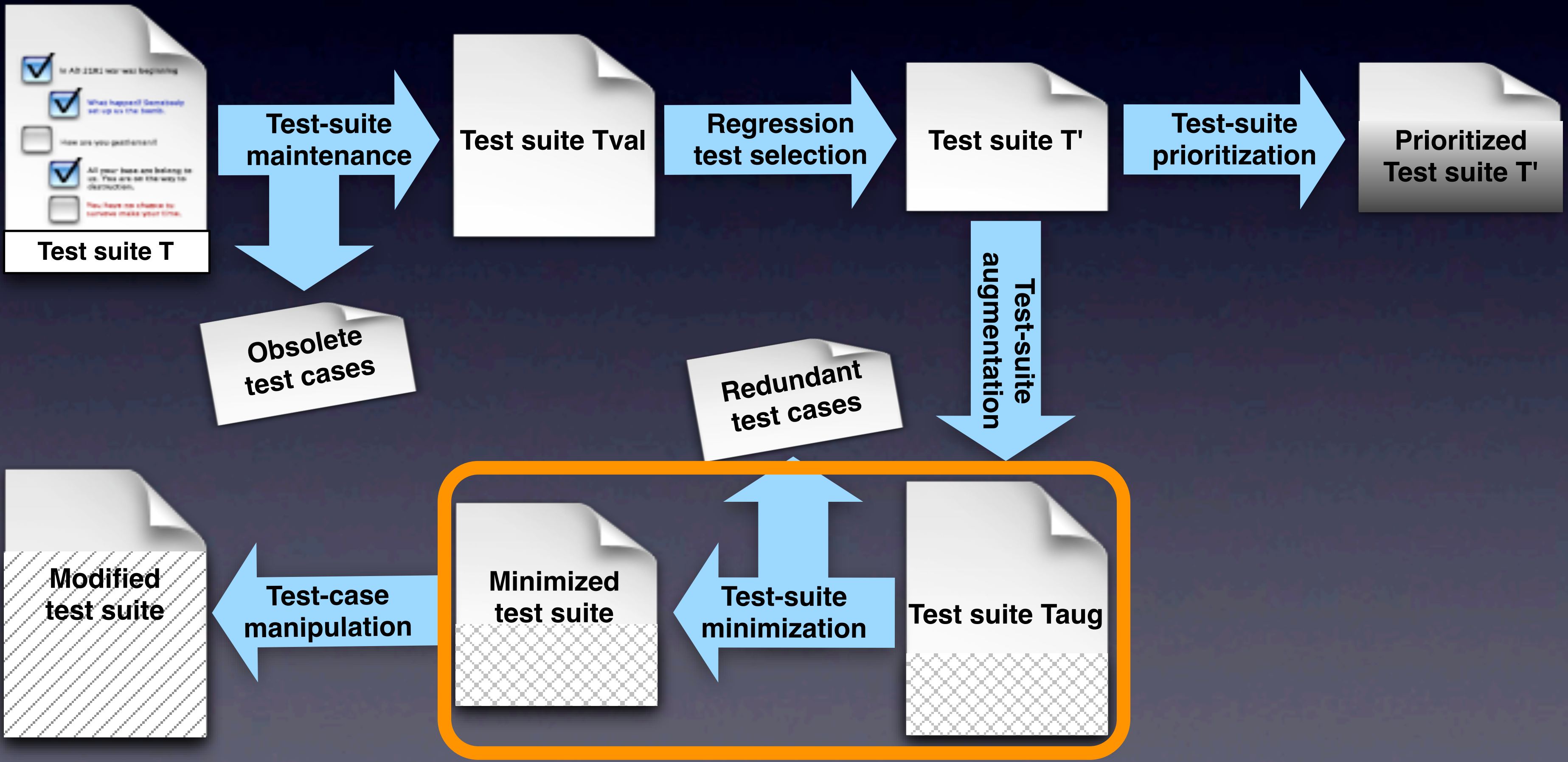
Redundant test cases

Minimized test suite

Test-case manipulation

Modified test suite

Regression Testing Process and Issues



Test suite T

Test-suite maintenance

Test suite Tval

Regression test selection

Test suite T'

Test-suite prioritization

Prioritized Test suite T'

Obsolete test cases

Test-suite augmentation

Redundant test cases

Test suite Taug

Test-suite minimization

Minimized test suite

Test-case manipulation

Modified test suite

Outline

- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion

Outline

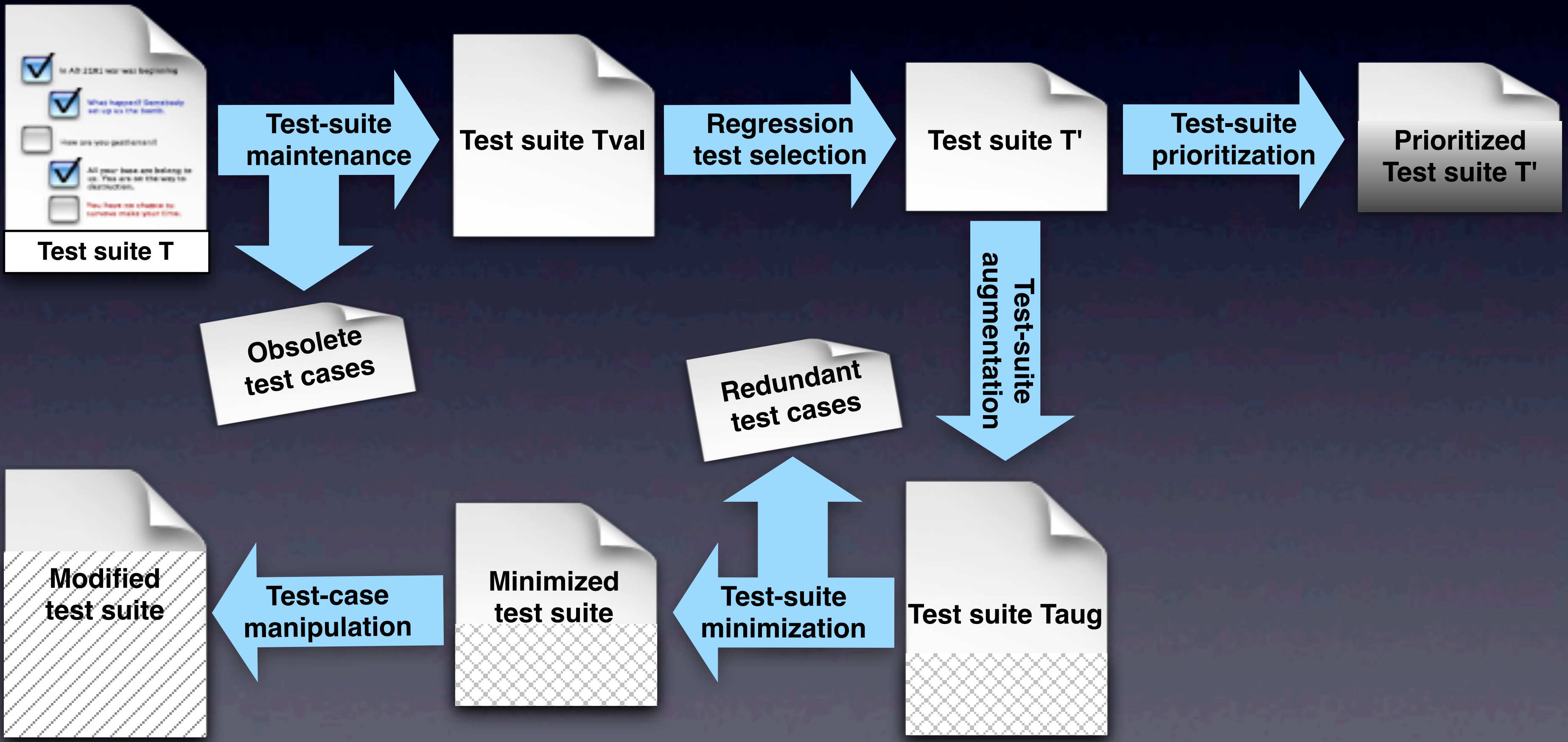
- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion

Outline

- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion

[FSE04]

Regression Test Selection



Test suite T

Test-suite maintenance

Obsolete test cases

Test suite Tval

Regression test selection

Test suite T'

Test-suite prioritization

Prioritized Test suite T'

Test-suite augmentation

Test suite Taug

Test-suite minimization

Redundant test cases

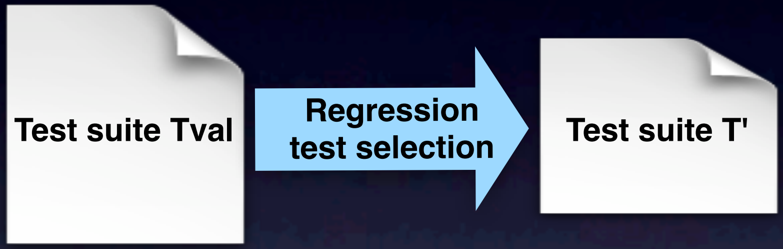
Minimized test suite

Test-case manipulation

Modified test suite

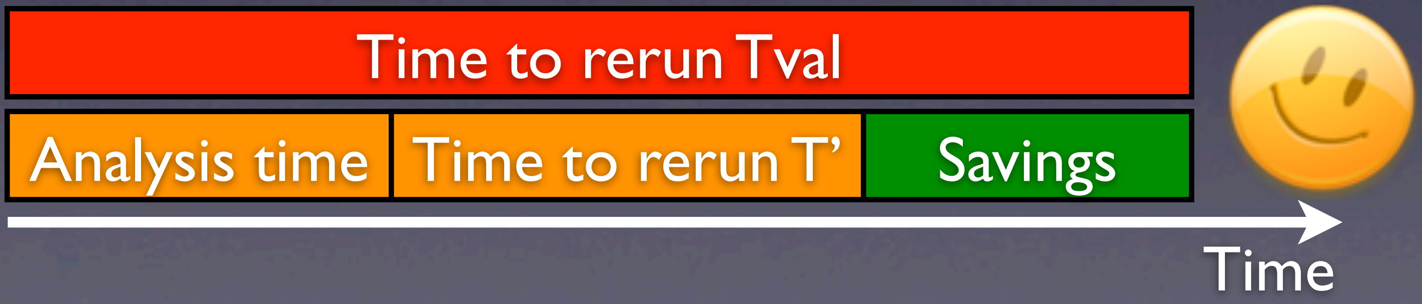
[FSE04]

Regression Test Selection


Test suite Tval

Regression test selection

Test suite T'


Time to rerun Tval

Analysis time

Time to rerun T’

Savings

Time

Motivating Example

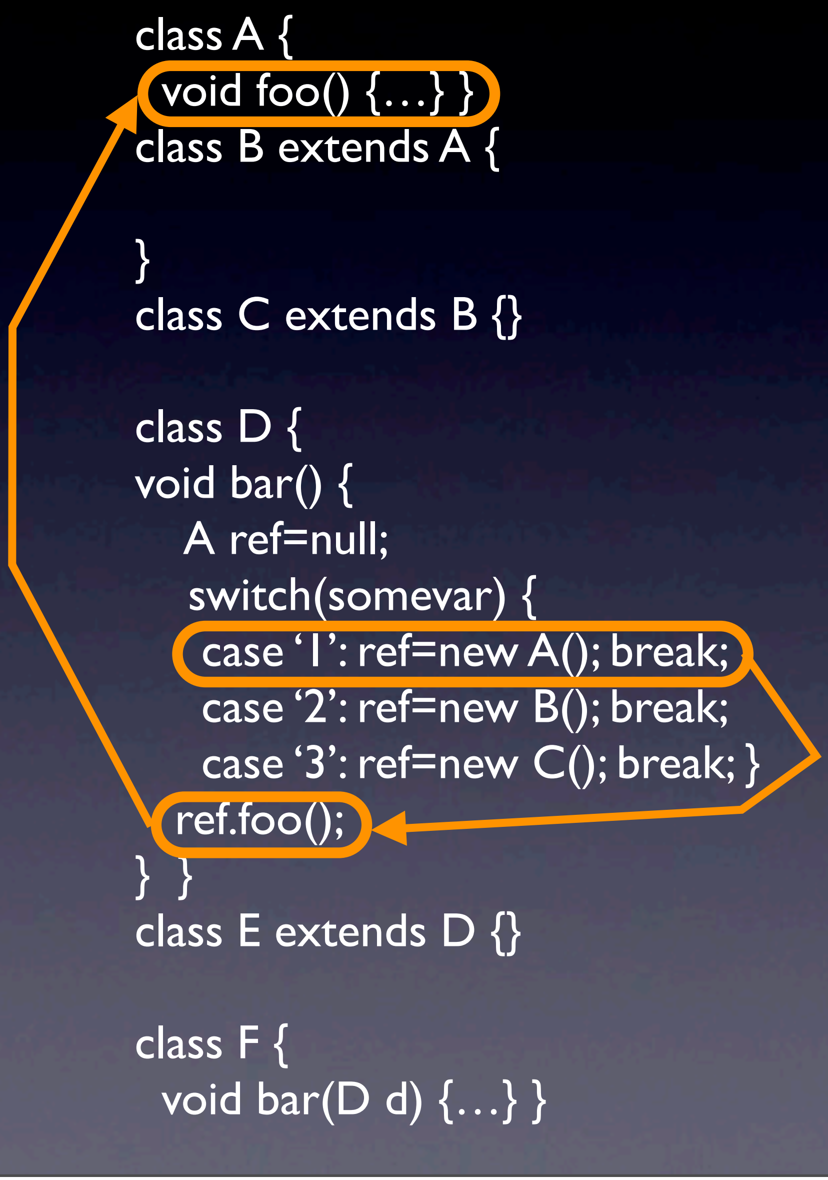
```
class A {
  void foo() {…} }
class B extends A {

}
class C extends B {}

class D {
void bar() {
    A ref=null;
     switch(somevar) {
      case ‘1’: ref=new A(); break;
      case ‘2’: ref=new B(); break;
      case ‘3’: ref=new C(); break; }
   ref.foo();
}  }
class E extends D {}

class F {
  void bar(D d) {…} }
```

Motivating Example

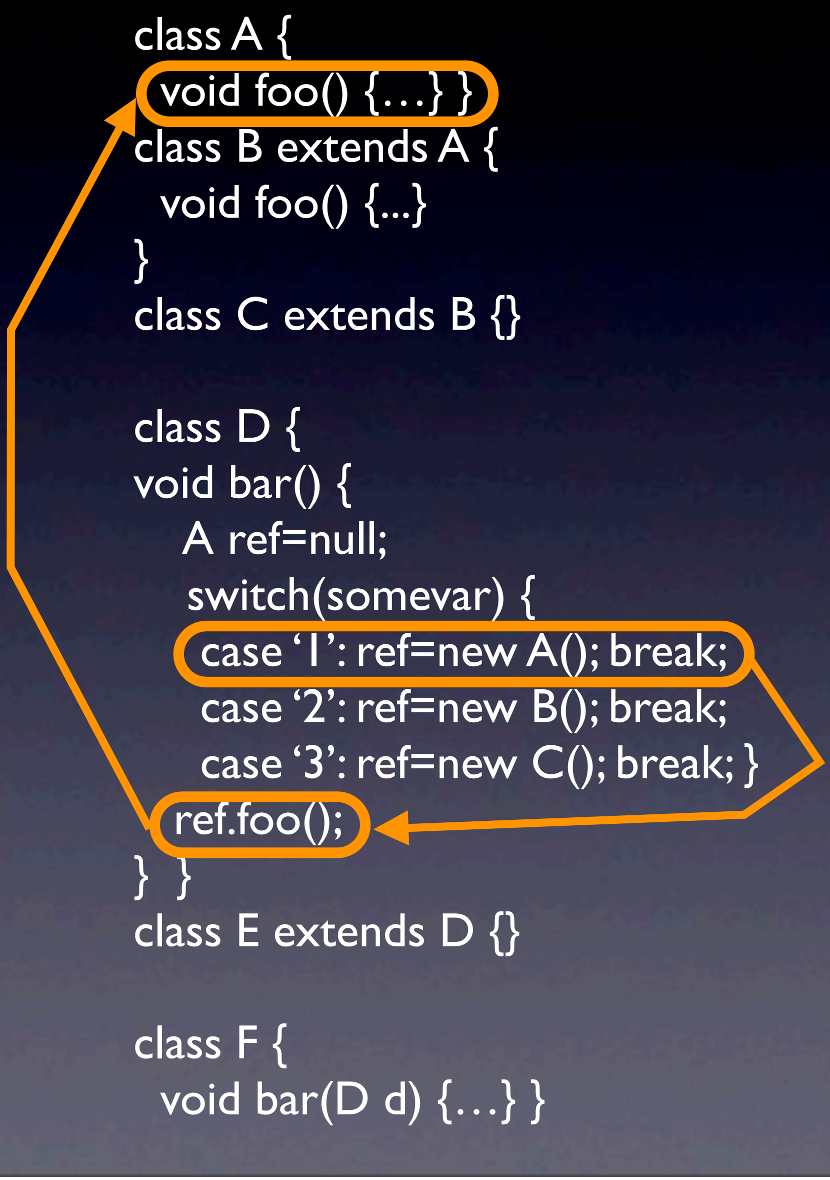
```
class A {
  void foo() {…} }
class B extends A {

}
class C extends B {}

class D {
void bar() {
    A ref=null;
    switch(somevar) {
     case ‘1’: ref=new A(); break;
     case ‘2’: ref=new B(); break;
     case ‘3’: ref=new C(); break; }
   ref.foo();
} }
class E extends D {}

class F {
  void bar(D d) {…} }
```

```
class A {
  void foo() {…} }
class B extends A {
  void foo() {...}
}
class C extends B {}

class D {
void bar() {
    A ref=null;
    switch(somevar) {
     case ‘1’: ref=new A(); break;
     case ‘2’: ref=new B(); break;
     case ‘3’: ref=new C(); break; }
   ref.foo();
} }
class E extends D {}

class F {
  void bar(D d) {…} }
```

Motivating Example

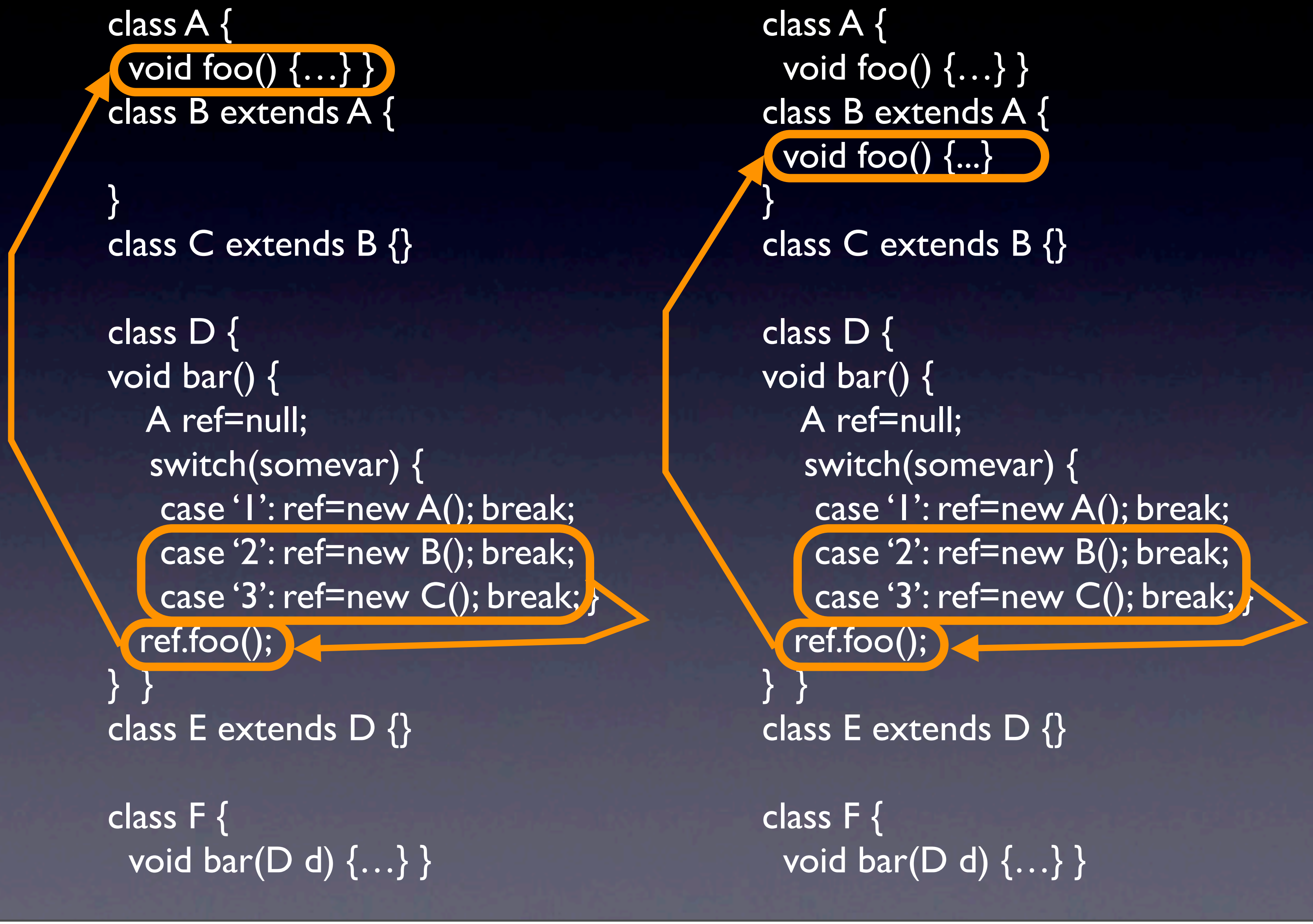


```
class A {
 void foo() {…} }
class B extends A {

}
class C extends B {}

class D {
void bar() {
    A ref=null;
     switch(somevar) {
      case ‘1’: ref=new A(); break;
      case ‘2’: ref=new B(); break;
      case ‘3’: ref=new C(); break; }
    ref.foo();
} }
class E extends D {}

class F {
  void bar(D d) {…} }
```



```
class A {
 void foo() {…} }
class B extends A {
  void foo() {...}
}
class C extends B {}

class D {
void bar() {
    A ref=null;
     switch(somevar) {
      case ‘1’: ref=new A(); break;
      case ‘2’: ref=new B(); break;
      case ‘3’: ref=new C(); break; }
    ref.foo();
} }
class E extends D {}

class F {
  void bar(D d) {…} }
```

Motivating Example



```
class A {
  void foo() {…} }
class B extends A {

}
class C extends B {}

class D {
void bar() {
    A ref=null;
     switch(somevar) {
      case ‘1’: ref=new A(); break;
      case ‘2’: ref=new B(); break;
      case ‘3’: ref=new C(); break; }
    ref.foo();
} }
class E extends D {}

class F {
  void bar(D d) {…} }
```

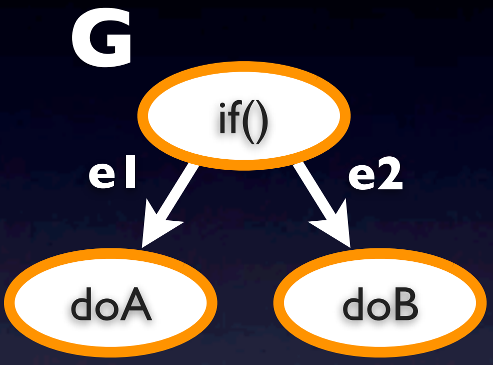
```
class A {
  void foo() {…} }
class B extends A {
  void foo() {...}
}
class C extends B {}

class D {
void bar() {
    A ref=null;
     switch(somevar) {
      case ‘1’: ref=new A(); break;
      case ‘2’: ref=new B(); break;
      case ‘3’: ref=new C(); break; }
    ref.foo();
} }
class E extends D {}

class F {
  void bar(D d) {…} }
```

Our Approach

- Handle Java features by suitably modeling them in the Java Interclass Graph (JIG)
- Use an algorithm that operates on the JIG to perform safe RTS
- Make some assumptions for safety

RTS Algorithm

1. Build JIG for P



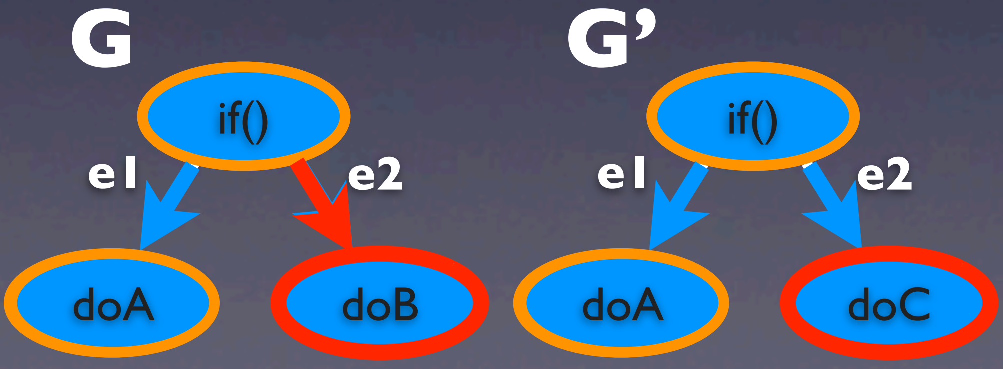
2. Collect coverage data

test cases

edges

	tc1	tc2	tc3
e1	x		
e2		x	x

3. Build G' and compare



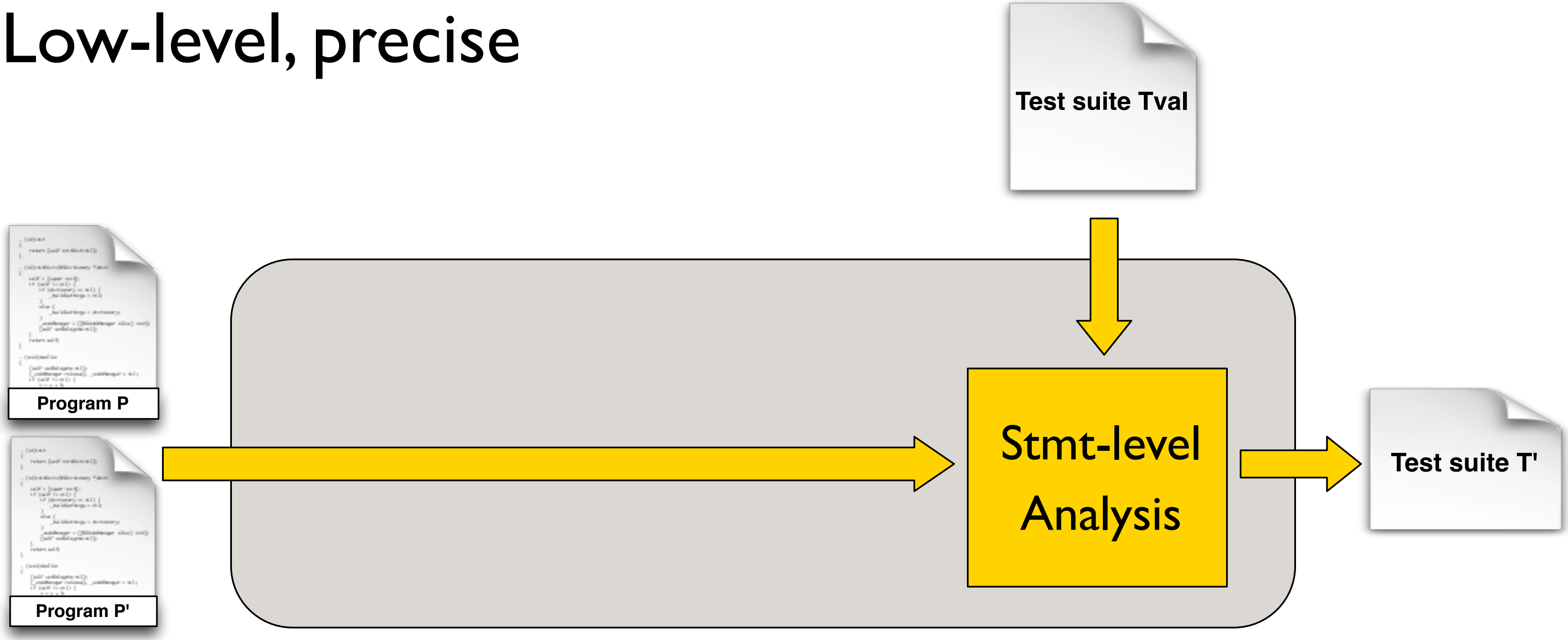
4. Select affected tests

test cases

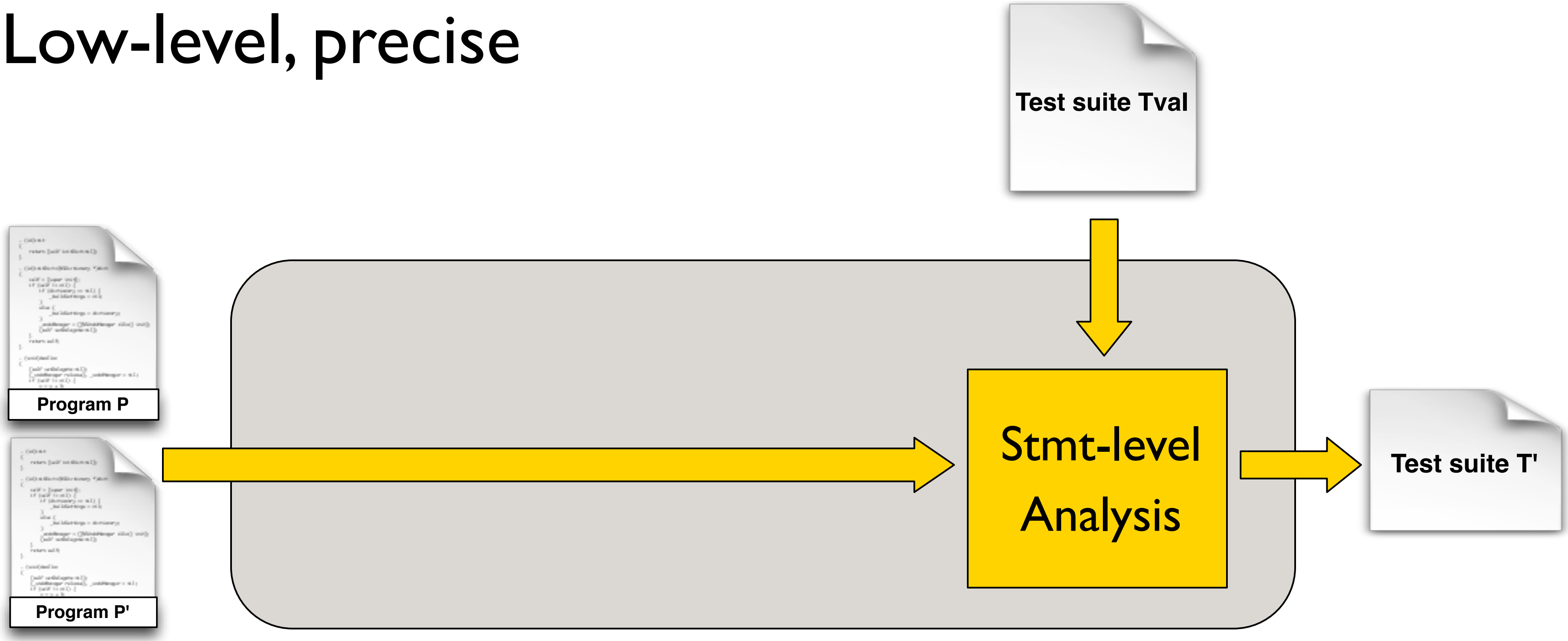
edges

	tc1	tc2	tc3
e1	x		
e2		x	x

Low-level, precise



Low-level, precise



Test suite Tval

Program P

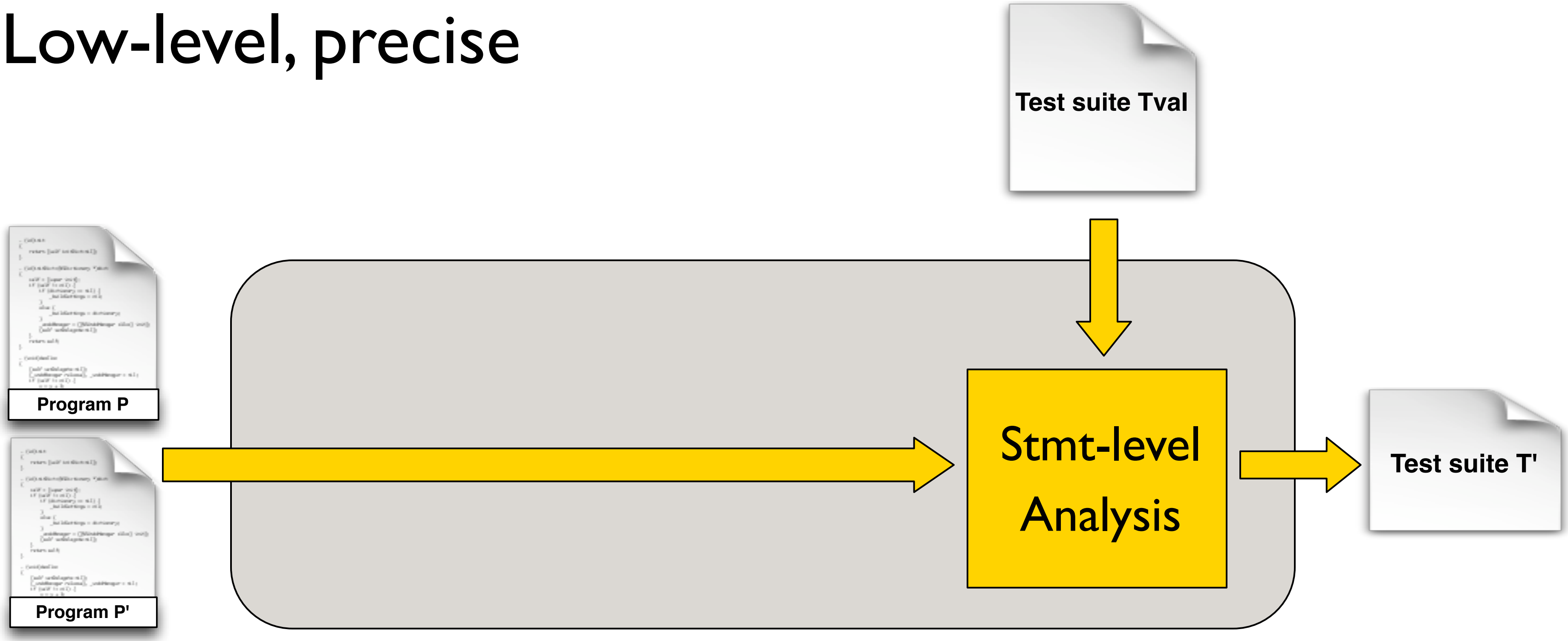
Program P'

Stmt-level Analysis

Test suite T'

Several medium-sized subjects (up to 40KLOC)

Low-level, precise



Test suite Tval

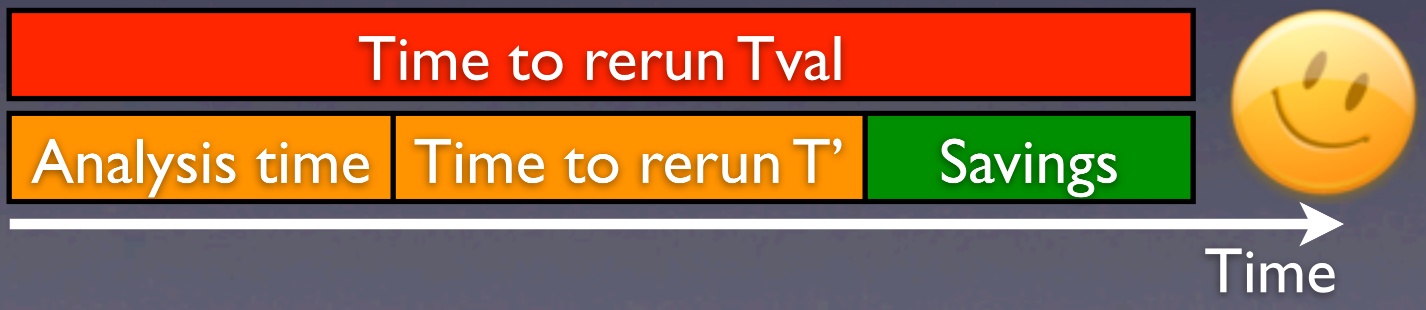
Program P

Program P'

Stmt-level Analysis

Test suite T'

Several medium-sized subjects (up to 40KLOC)



Time to rerun Tval

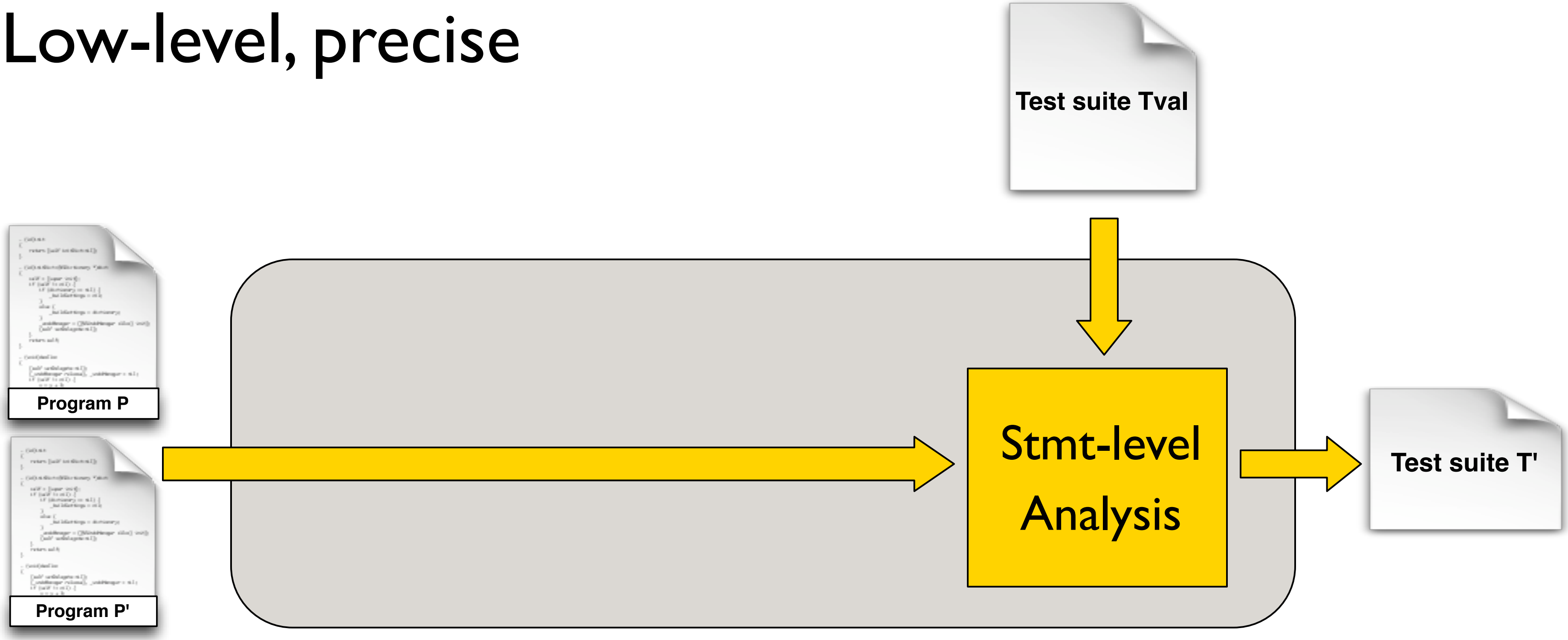
Analysis time

Time to rerun T’

Savings

Time

Low-level, precise



Test suite Tval

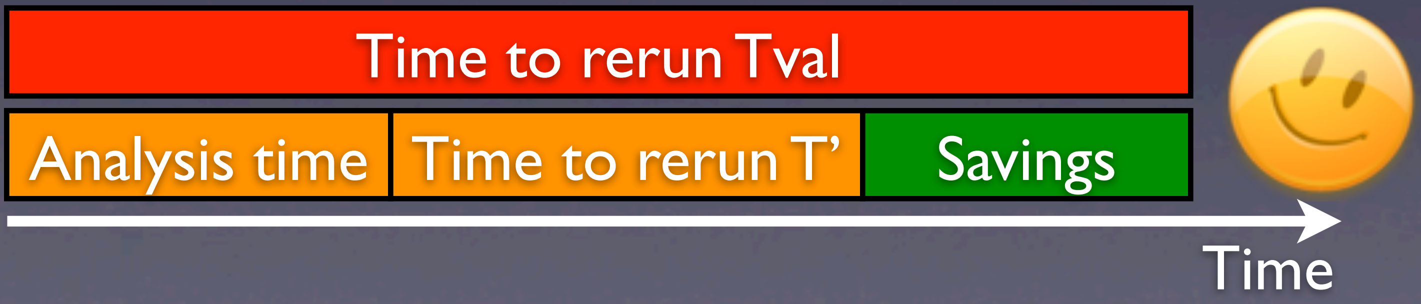
Program P

Program P'

Stmt-level Analysis

Test suite T'

JBoss – web application server, 1 million LOC



Time to rerun Tval

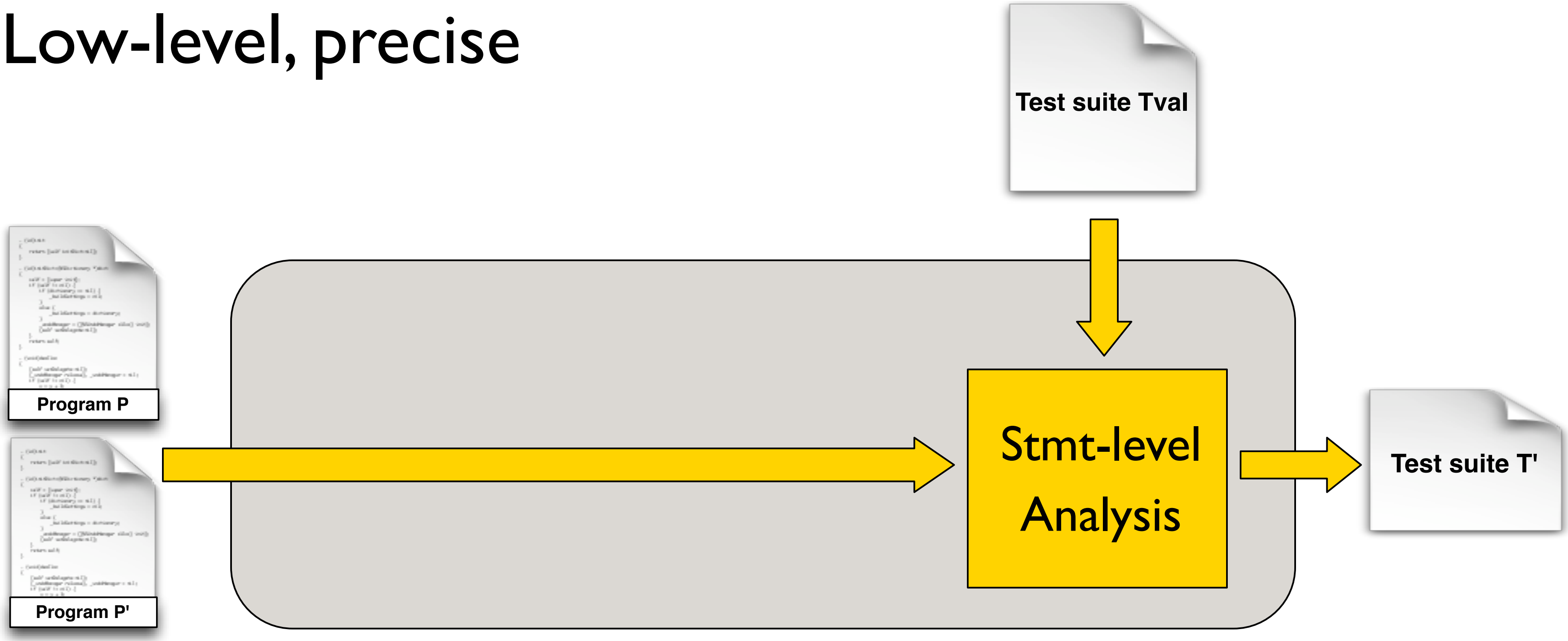
Analysis time

Time to rerun T'

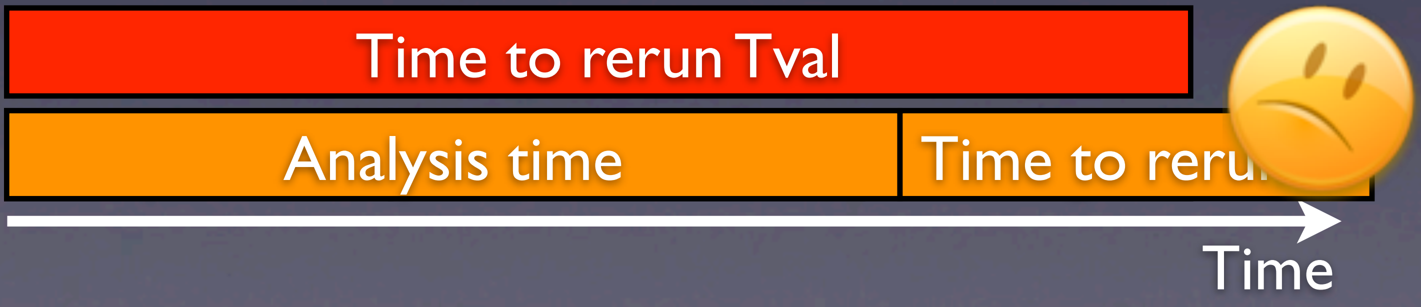
Savings

Time

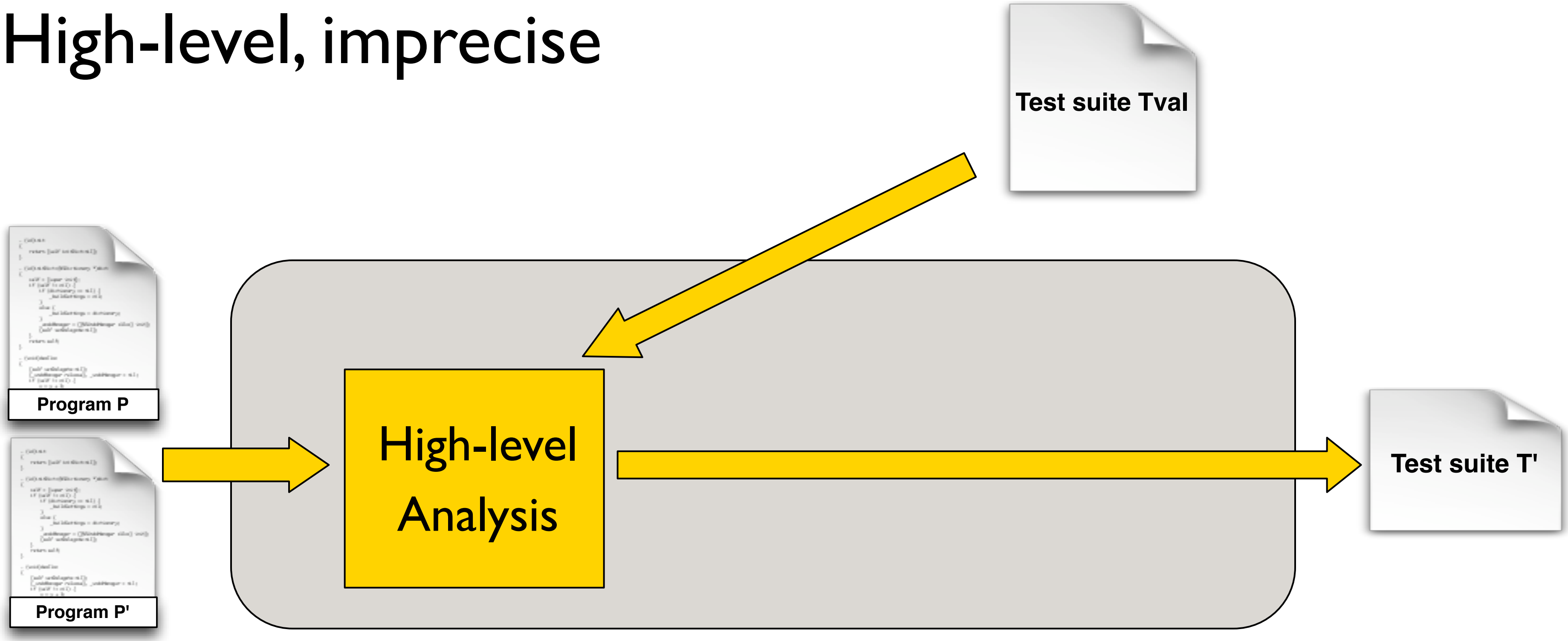
Low-level, precise

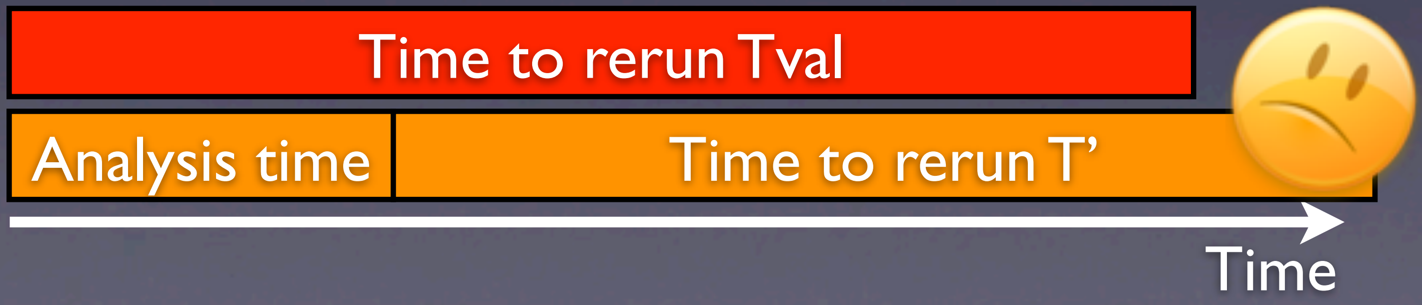


JBoss – web application server, 1 million LOC

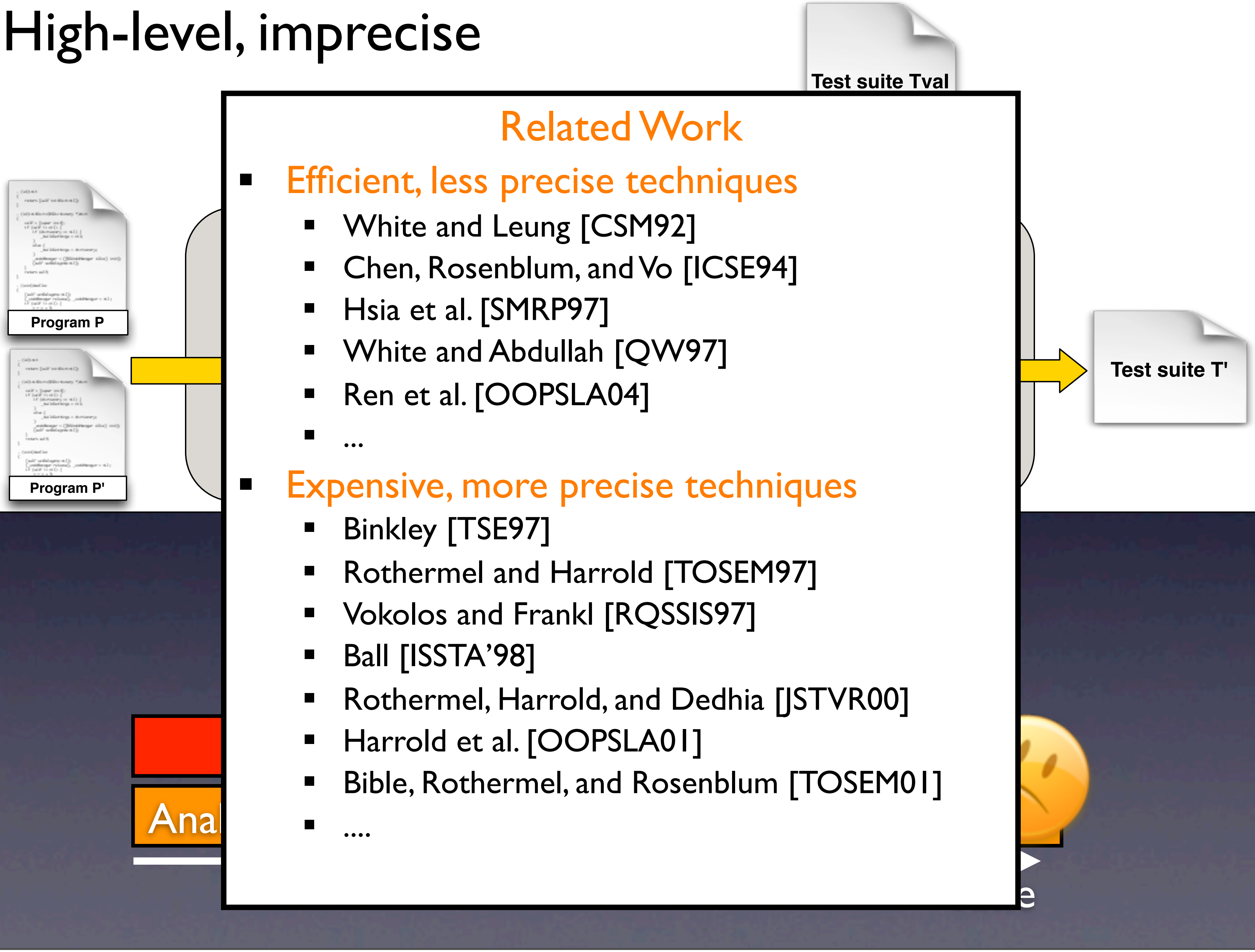


High-level, imprecise





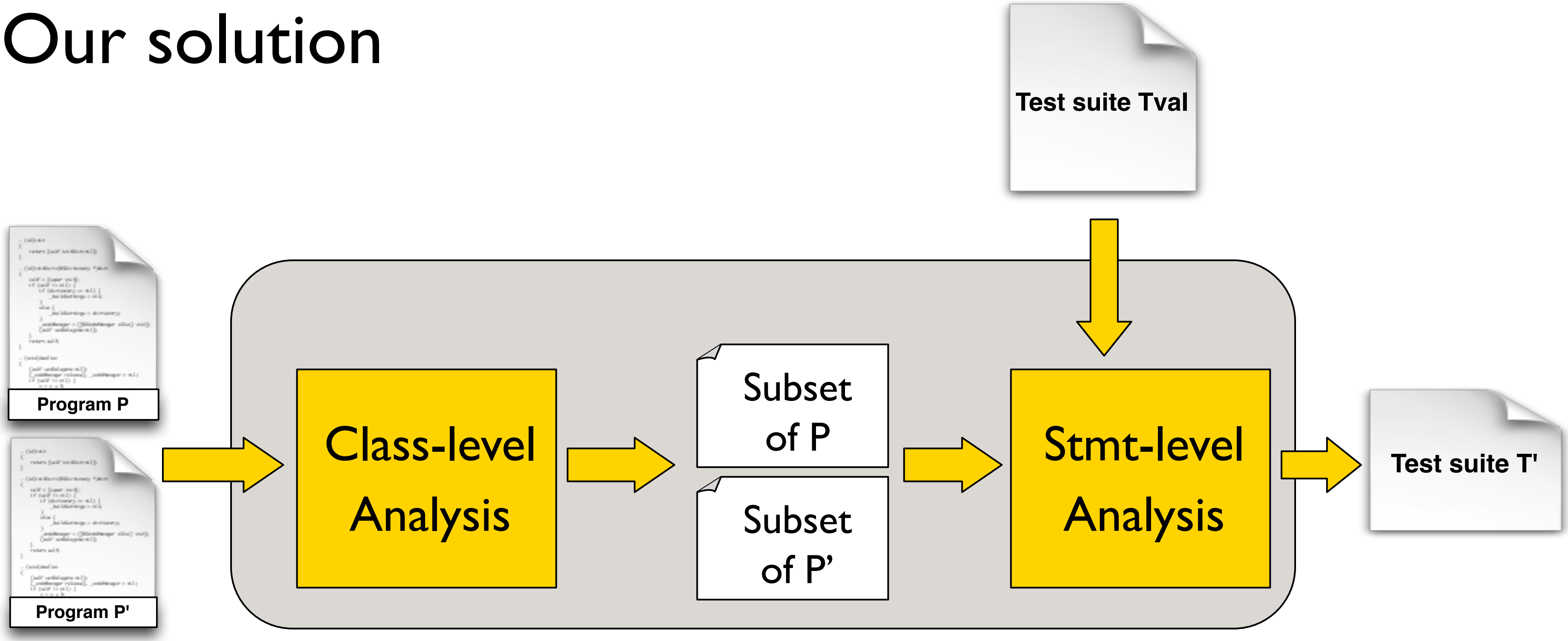
High-level, imprecise



Related Work

- Efficient, less precise techniques
  - White and Leung [CSM92]
  - Chen, Rosenblum, and Vo [ICSE94]
  - Hsia et al. [SMRP97]
  - White and Abdullah [QW97]
  - Ren et al. [OOPSLA04]
  - ...
- Expensive, more precise techniques
  - Binkley [TSE97]
  - Rothermel and Harrold [TOSEM97]
  - Vokolos and Frankl [RQSSIS97]
  - Ball [ISSTA'98]
  - Rothermel, Harrold, and Dedhia [JSTVR00]
  - Harrold et al. [OOPSLA01]
  - Bible, Rothermel, and Rosenblum [TOSEM01]
  - ....

Our solution



Two-phase approach

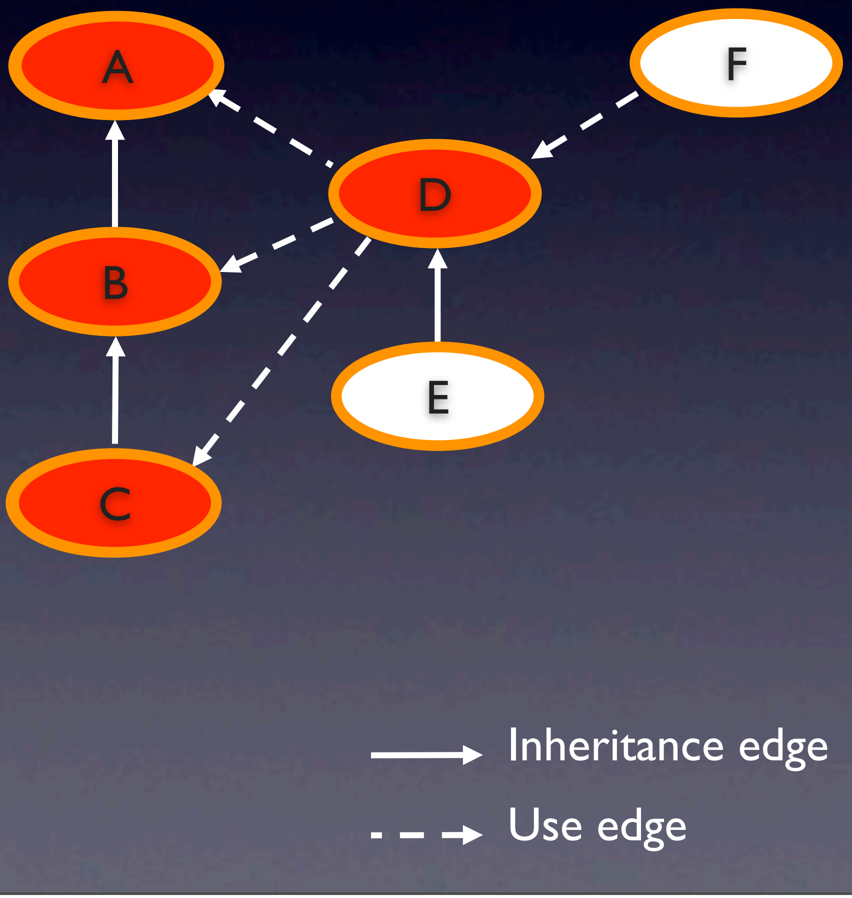
1. Class-Level analysis ➡ subset of P and P’
2. Stmt-Level analysis on the subset ➡ T’

1. Class-level Analysis

P/P’

```
class A {
  void foo() {…} }
class B extends A {
  void foo() {...}
}
class C extends B {}
class D {
void bar() {
    A ref=null;
     switch(somevar) {
      case ‘1’: ref=new A(); break;
      case ‘2’: ref=new B(); break;
      case ‘3’: ref=new C(); break; }
    ref.foo();
} }
class E extends D {}
class F {
  void bar(D d) {…} }
```

Interclass Relation Graph
(for P and P’)



2. Stmt-level Analysis

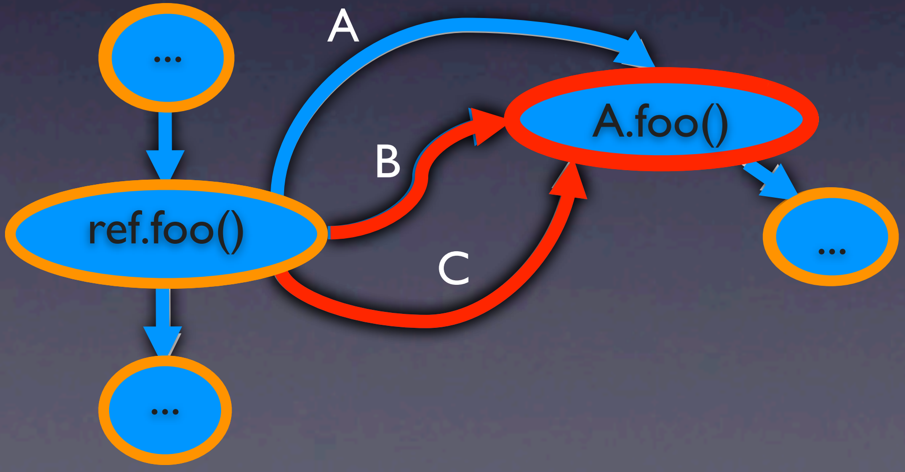
Subset of P

```
class A
class B {…}
class C
class D {
   void bar() {…; ref.foo(); …}
 } }
```

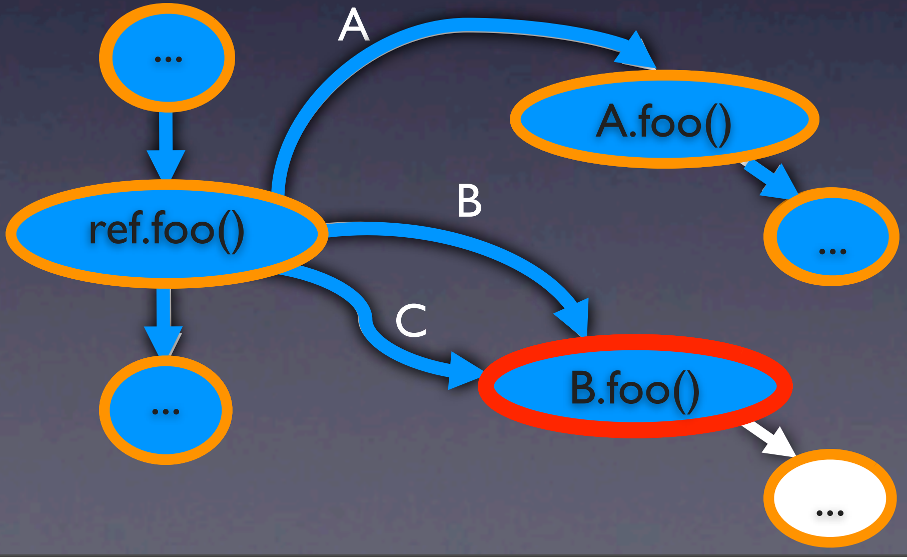
Subset of P’

```
class A
class B {… void foo() {…} … }
class C
class D {
   void bar() {…; ref.foo(); …}
 } }
```

G (excerpt)



G’ (excerpt)



2. Stmt-level Analysis

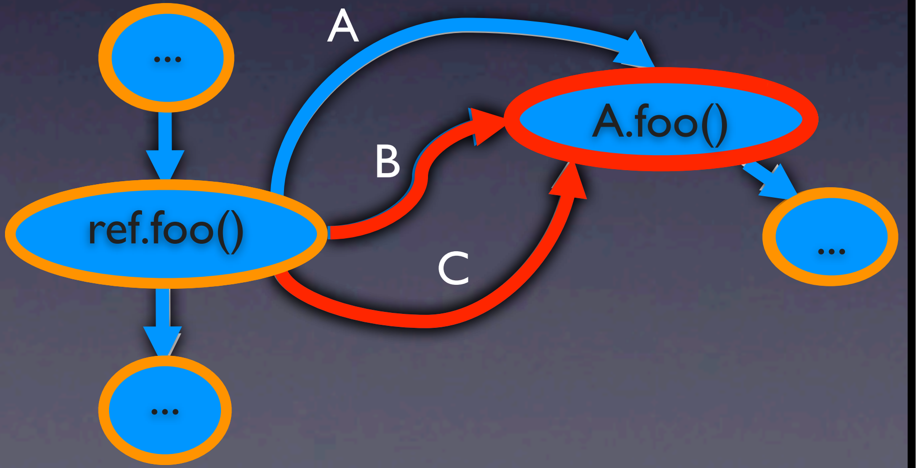
Subset of P

```
class A
class B {…}
class C
class D {
   void bar() {…; ref.foo(); …}
 } }
```

Subset of P’

```
class A
class B {… void foo() {…} … }
class C
class D {
   void bar() {…; ref.foo(); …}
 } }
```

G (excerpt)



G’ (excerpt)

Test cases to be rerun:

Test cases in Tval that execute the call node with ref’s dynamic type being B or C

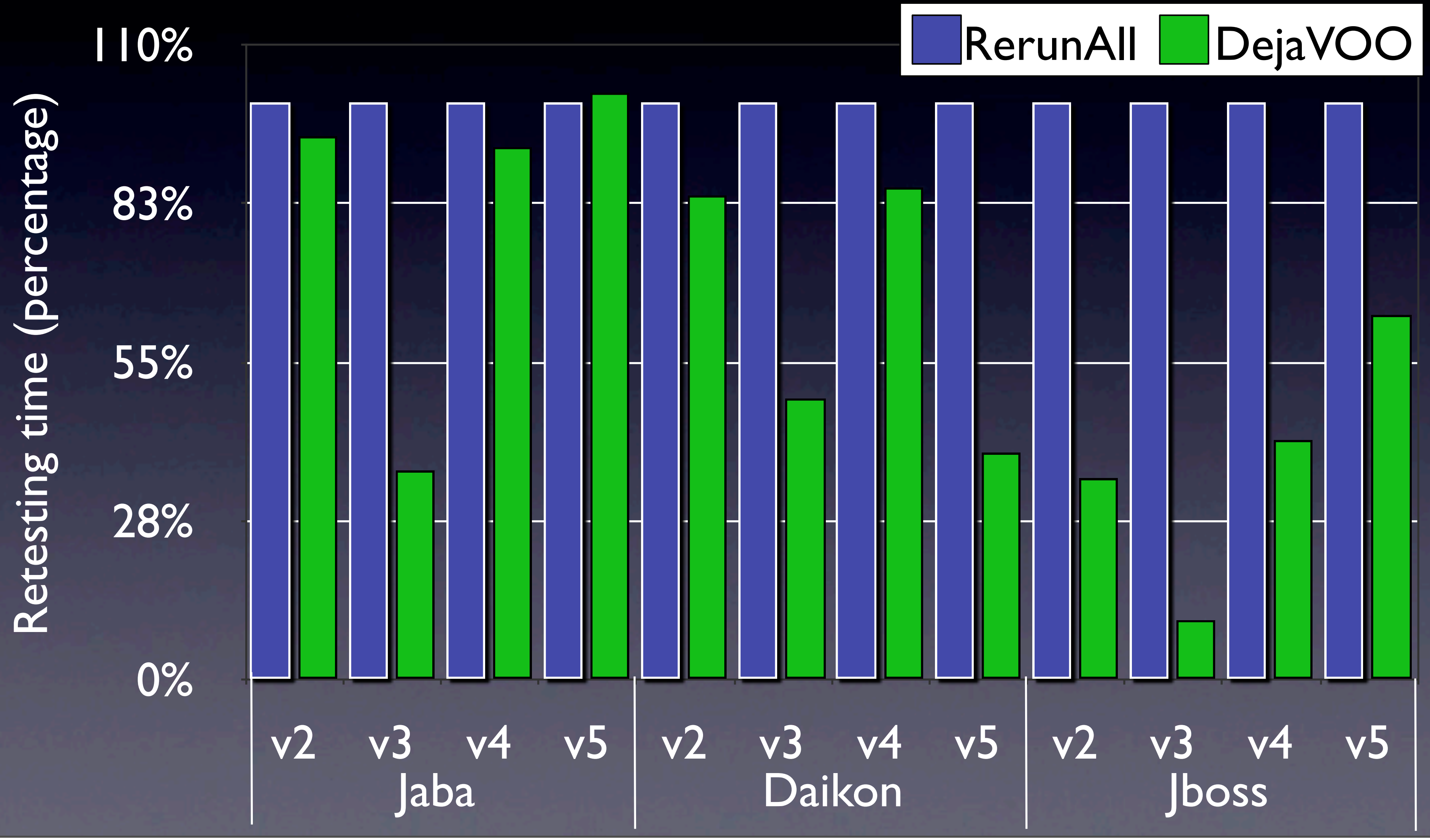
Empirical Evaluation

- Tool: DejaVOO
- Subjects:

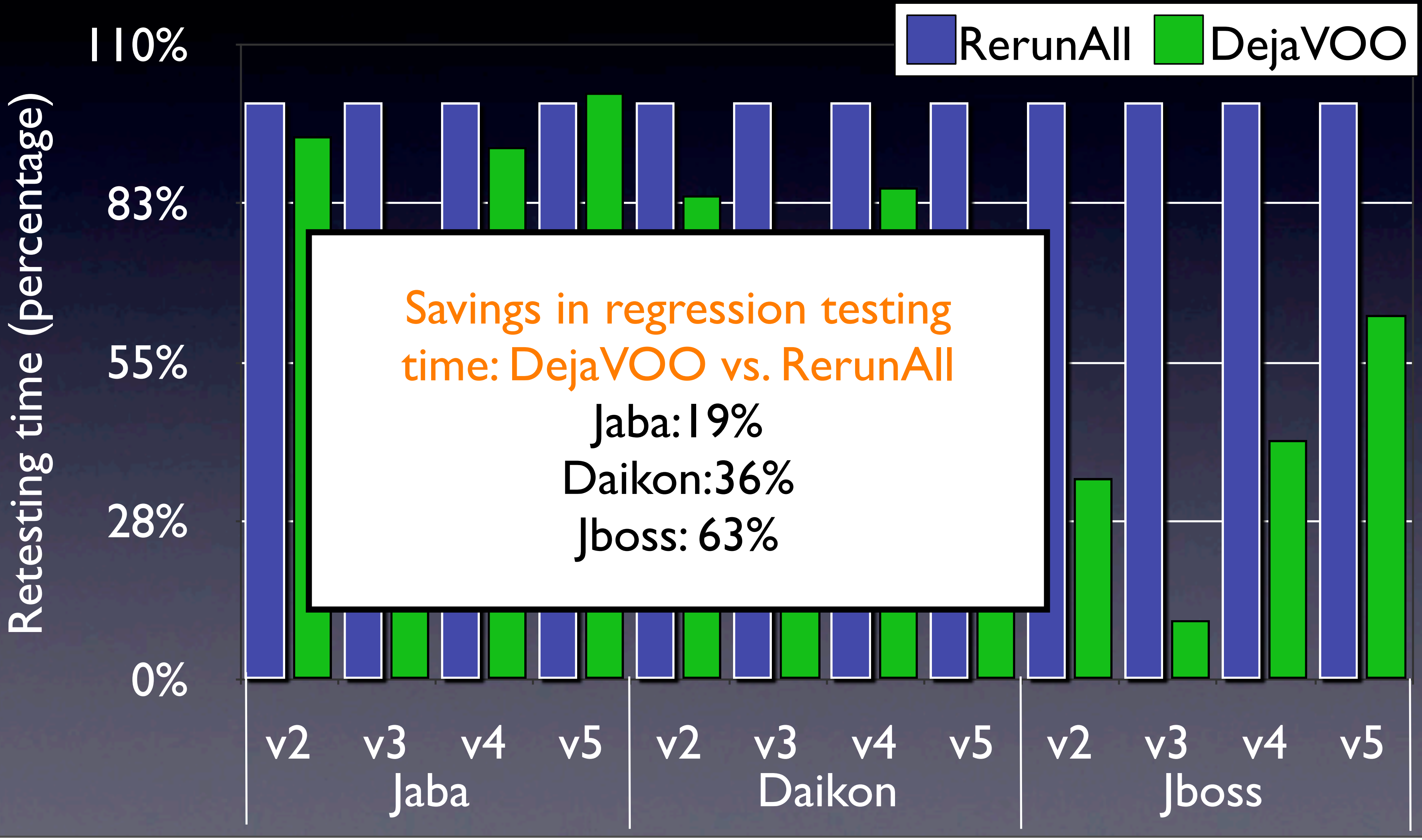
Program	#versions	#classes	KLOC	#test cases	retest time
Jaba	5	525	70	707	54 min
Daikon	5	824	167	200	74 min
Jboss	5	2,403	1,000	639	32 min

- RQ: What are the savings in testing time we can achieve using DejaVOO?

Results


RerunAll
DejaVOO
Retesting time (percentage)
110%
83%
55%
28%
0%
v2
v3
v4
v5
Jaba
v2
v3
v4
v5
Daikon
v2
v3
v4
v5
Jboss

Results



Savings in regression testing time: DejaVOO vs. RerunAll

Jaba:19%

Daikon:36%

Jboss: 63%

Regression Test Selection Summary

- DejaVOO
  - Based on the Interclass Relation Graph and Java Interclass Graph
  - First phase identifies affected classes
  - Second phase performs low-level analysis
- Benefits of our technique
  - Handles Java features
  - Handles subsystems without analyzing external classes
  - Safe (under some assumptions)

Outline

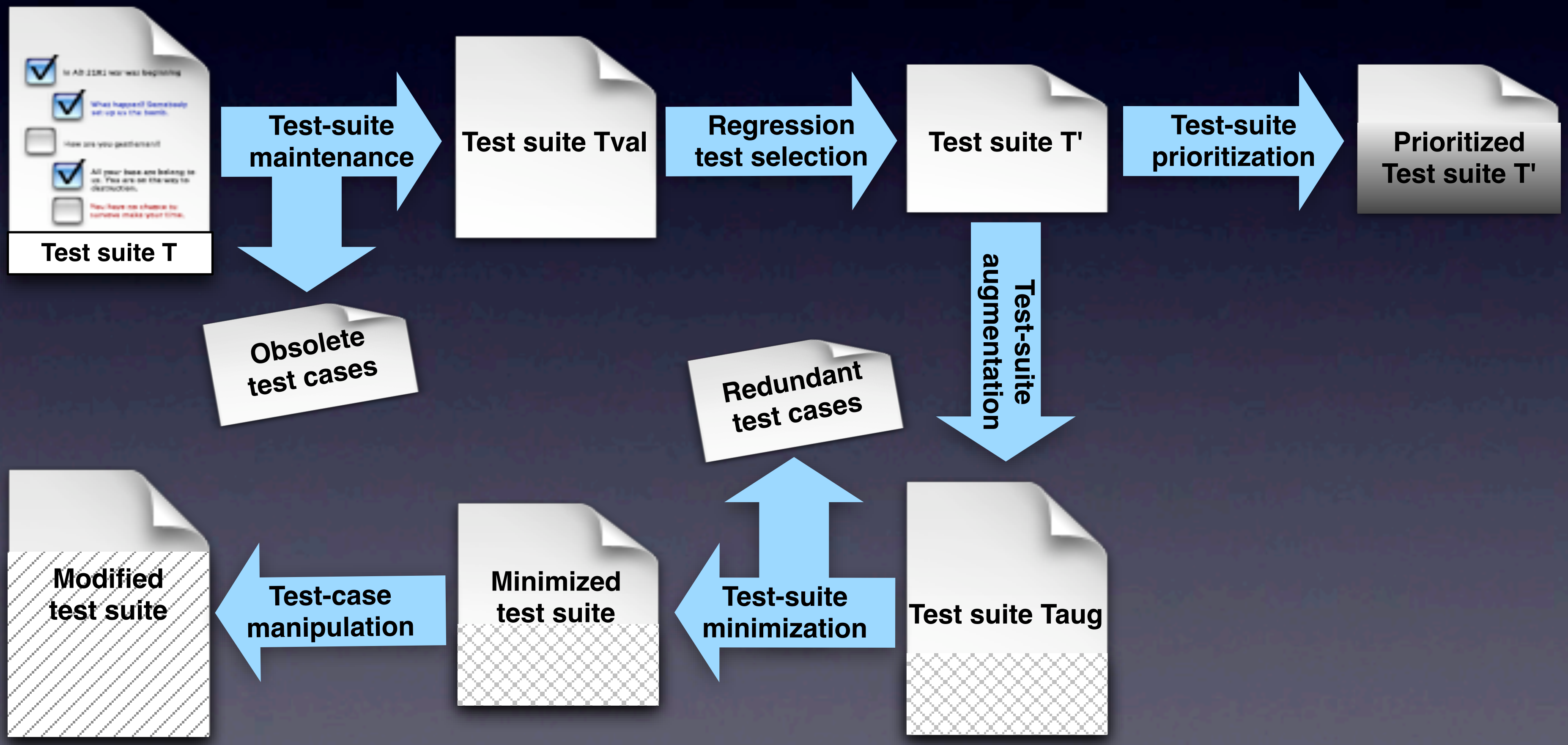
- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion

Outline

- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion

[ASE08][ICST10]

Test Suite Augmentation



Test suite T

Test-suite maintenance

Test suite Tval

Regression test selection

Test suite T'

Test-suite prioritization

Prioritized Test suite T'

Obsolete test cases

Test-suite augmentation

Redundant test cases

Test suite Taug

Test-suite minimization

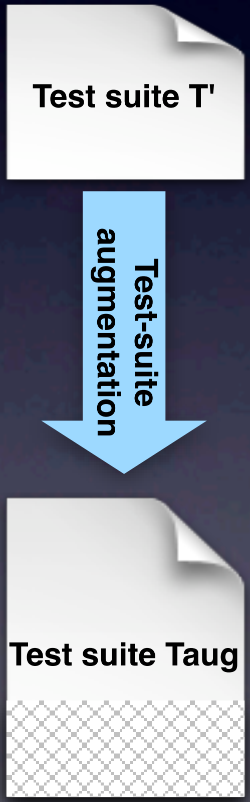
Minimized test suite

Test-case manipulation

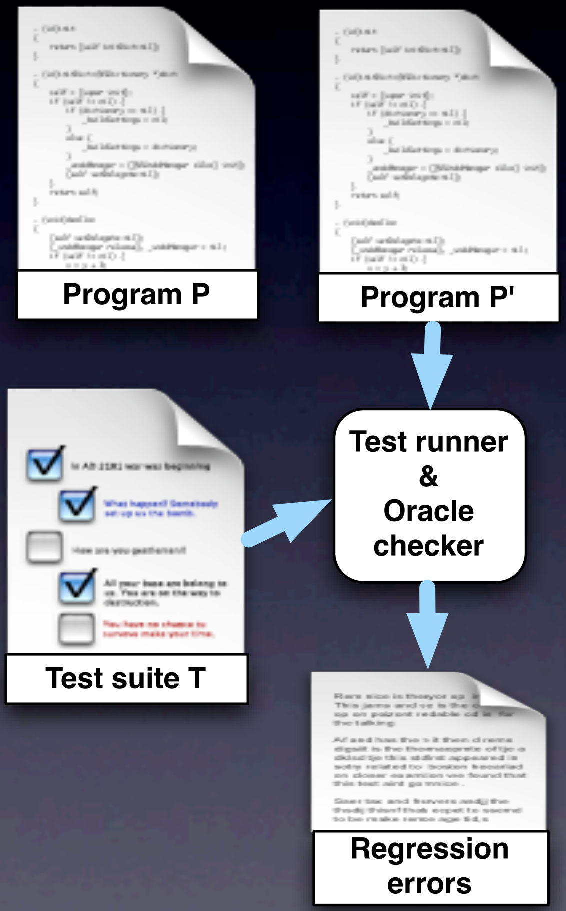
Modified test suite

[ASE08][ICST10]

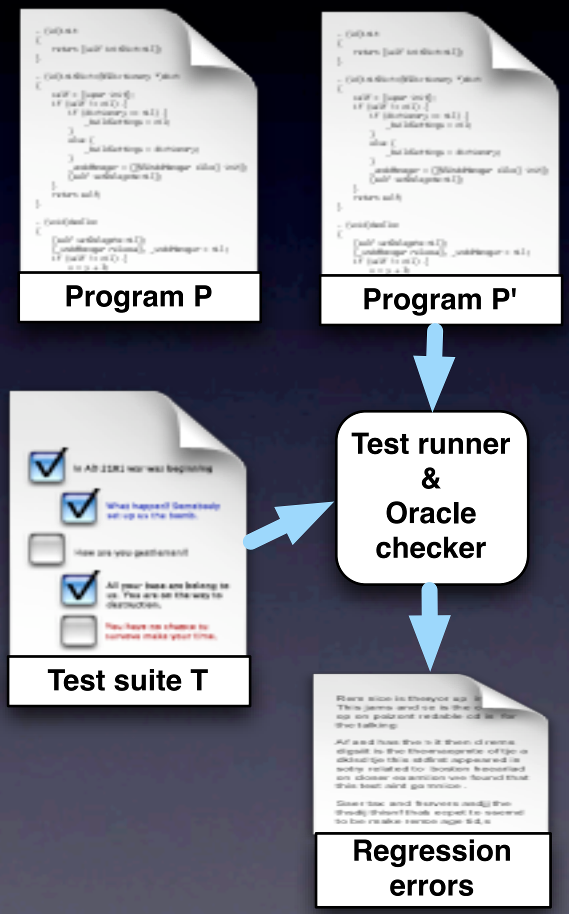
Test Suite Augmentation



Traditional regression testing



Traditional regression testing



Program P

Program P'

Test suite T

Test runner & Oracle checker

Regression errors

```
class BankAccount {
 double balance;


 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

 bool withdraw(double amount) {
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (balance < 0)
   print("account overdraft");
   return false;
  }
  balance = balance - amount;


  return true;
 }
}
```

```
class BankAccount {
 double balance;


 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

 bool withdraw(double amount) {
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (balance < 0)
   print("account overdraft");
   return false;
  }
  balance = balance - amount;


  return true;
 }
}
```

```
class BankAccount {
 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

 bool withdraw(double amount) {
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

Where is the fault?

```
class BankAccount {
 double balance;
 bool isOverdraft;
 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }
 bool withdraw(double amount) {
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
class BankAccount {
 double balance;
 bool isOverdraft;
 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }
{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false);
```

```
class BankAccount {
 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
    balance = balance + amount;
    return true;
  } else {
    print("negative amount");
    return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false);
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false); ✓
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
    balance = balance + amount;
    return true;
  } else {
    print("negative amount");
    return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false); ✓
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false); ✓
}
void test3() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, true);
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false); ✓
}
void test3() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, true); ✓
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
    balance = balance + amount;
    return true;
  } else {
    print("negative amount");
    return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
    print("negative amount");
    return false;
  }
  if (isOverdraft) {
    print("account overdraft");
    return false;
  }
  balance = balance - amount;
  if (balance < 0)
    isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false); ✓
}
void test3() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, true); ✓
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false);  ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false);  ✓
}
void test3() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, true);   ✓
}
void test4() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 a.withdraw(200.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, false);
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false); ✓
}
void test3() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, true); ✓
}
void test4() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 a.withdraw(200.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, false); ✓
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false); ✓
}
void test3() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 bool result = a.withdraw(50.00); ✓
 assertEquals(result, true);
}
void test4() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 a.withdraw(200.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, false); ✓
 result = a.deposit(200.00);
 assertEquals(result, true);
}
...
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false); ✓
}
void test3() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, true); ✓
}
void test4() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 a.withdraw(200.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, false); ✓
 result = a.deposit(200.00);
 assertEquals(result, true); ✓
}
...
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
Class BankAccountTest {
...
void test1() {
 BankAccount a=new BankAccount();
 bool result = a.deposit(-1.00);
 assertEquals(result, false); ✓
}
void test2() {
 BankAccount a=new BankAccount();
 bool result = a.withdraw(-1.00);
 assertEquals(result, false); ✓
}
void test3() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, true); ✓
}
void test4() {
 BankAccount a=new BankAccount();
 a.deposit(100.00);
 a.withdraw(200.00);
 bool result = a.withdraw(50.00);
 assertEquals(result, false); ✓
 result = a.deposit(200.00);
 assertEquals(result, true); ✓
}
...
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

{bool withdraw(double amount)
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
...
void testBehavioralDifference() {
 BankAccount a=new BankAccount();
 a.deposit(10.00);
 a.withdraw(20.00);
 a.deposit(50.00);
 bool result = a.withdraw(20.00);
 assertEquals(result, true);
}
...
```

```
class BankAccount {

 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

 bool withdraw(double amount) {
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
...
void testBehavioralDifference() {
 BankAccount a=new BankAccount();
 a.deposit(10.00);
 a.withdraw(20.00);
 a.deposit(50.00);
 bool result = a.withdraw(20.00);
 assertEquals(result, true); ✗
}
...
```

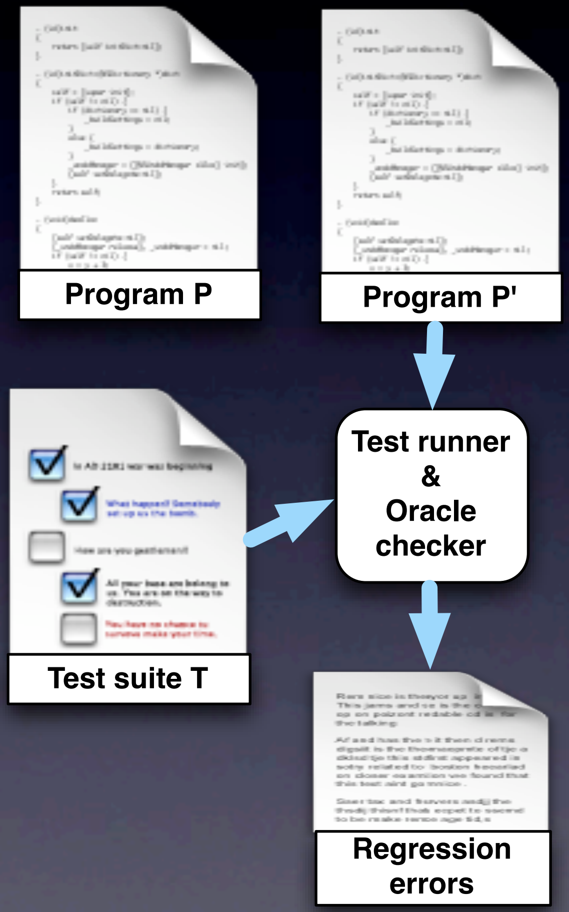
```
class BankAccount {
 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

 bool withdraw(double amount) {
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

```
...
void testBehavioralDifference() {
 BankAccount a=new BankAccount();
 a.deposit(10.00);
 a.withdraw(20.00);
 a.deposit(50.00);
 bool result = a.withdraw(20.00);
 assertEquals(result, true); ✗
}
...
```

- Such a test may not be in T
  - 100% stmt coverage without it
  - Specific sequence of calls/params
- Or its oracle may be inadequate

```
class BankAccount {
 double balance;
 bool isOverdraft;

 bool deposit(double amount) {
  if (amount > 0.00) {
   balance = balance + amount;
   return true;
  } else {
   print("negative amount");
   return false;
  }
 }

 bool withdraw(double amount) {
  if (amount <= 0) {
   print("negative amount");
   return false;
  }
  if (isOverdraft) {
   print("account overdraft");
   return false;
  }
  balance = balance - amount;
  if (balance < 0)
   isOverdraft = true;
  return true;
 }
}
```

Traditional regression testing



Program P

Program P'

Test suite T

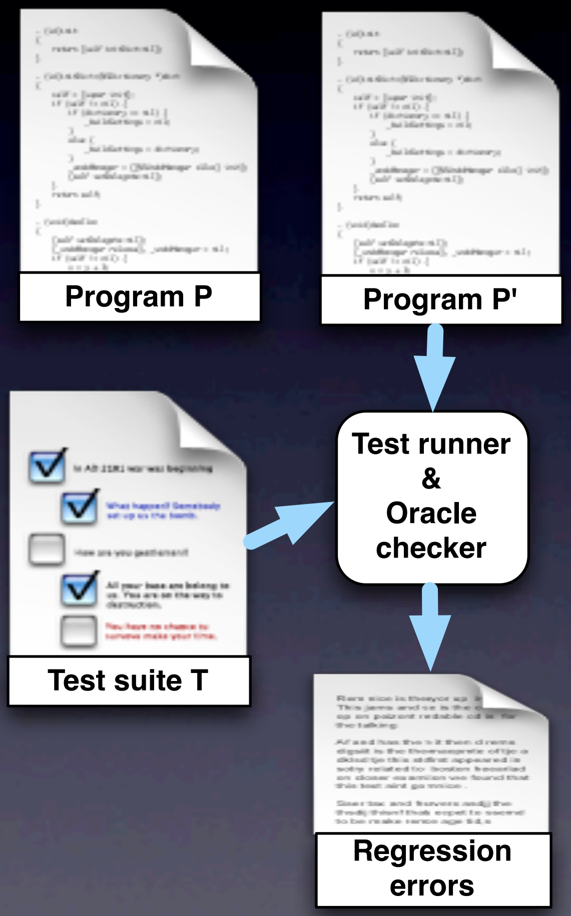
Test runner & Oracle checker

Regression errors

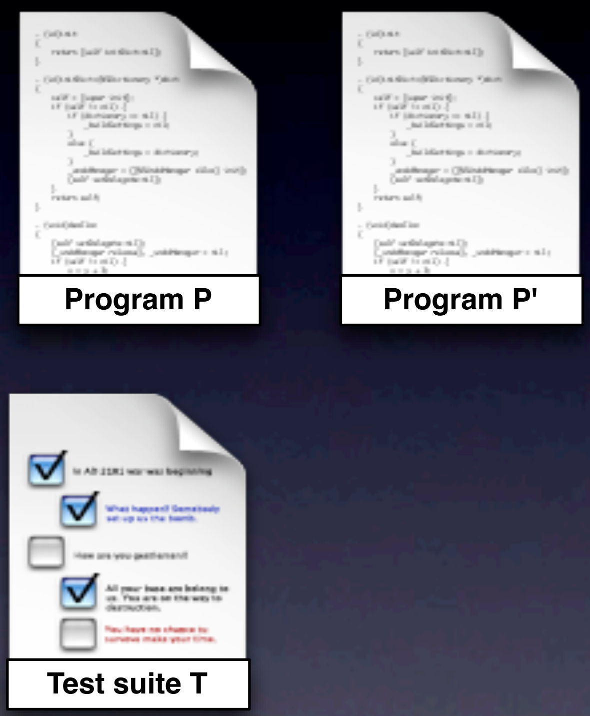
Existing test suites typically target a small subset of the program behavior

- Tests focus on core functionality
- Oracles often approximated

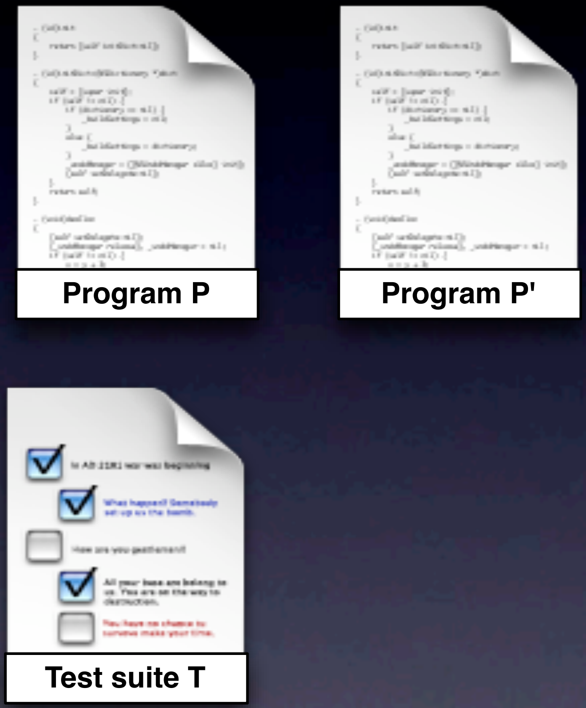
Traditional regression testing



BERT



BERT



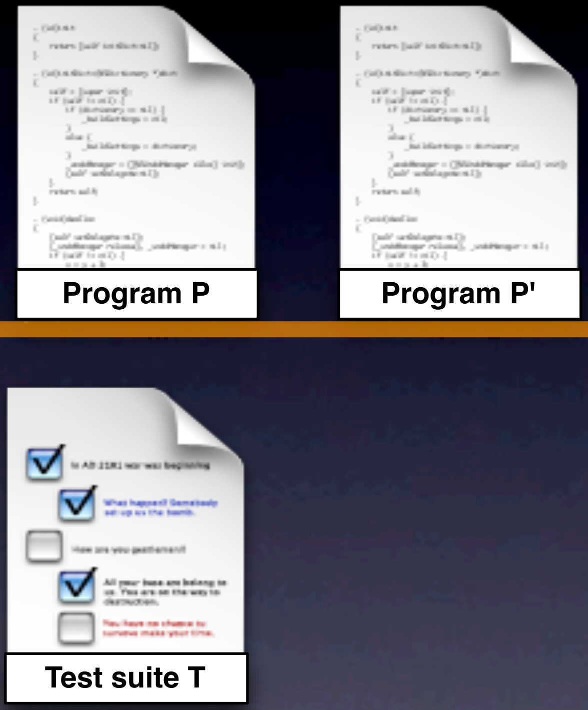
Program P

Program P'

Test suite T

BERT

Phase I:
Generation of test cases for changed code

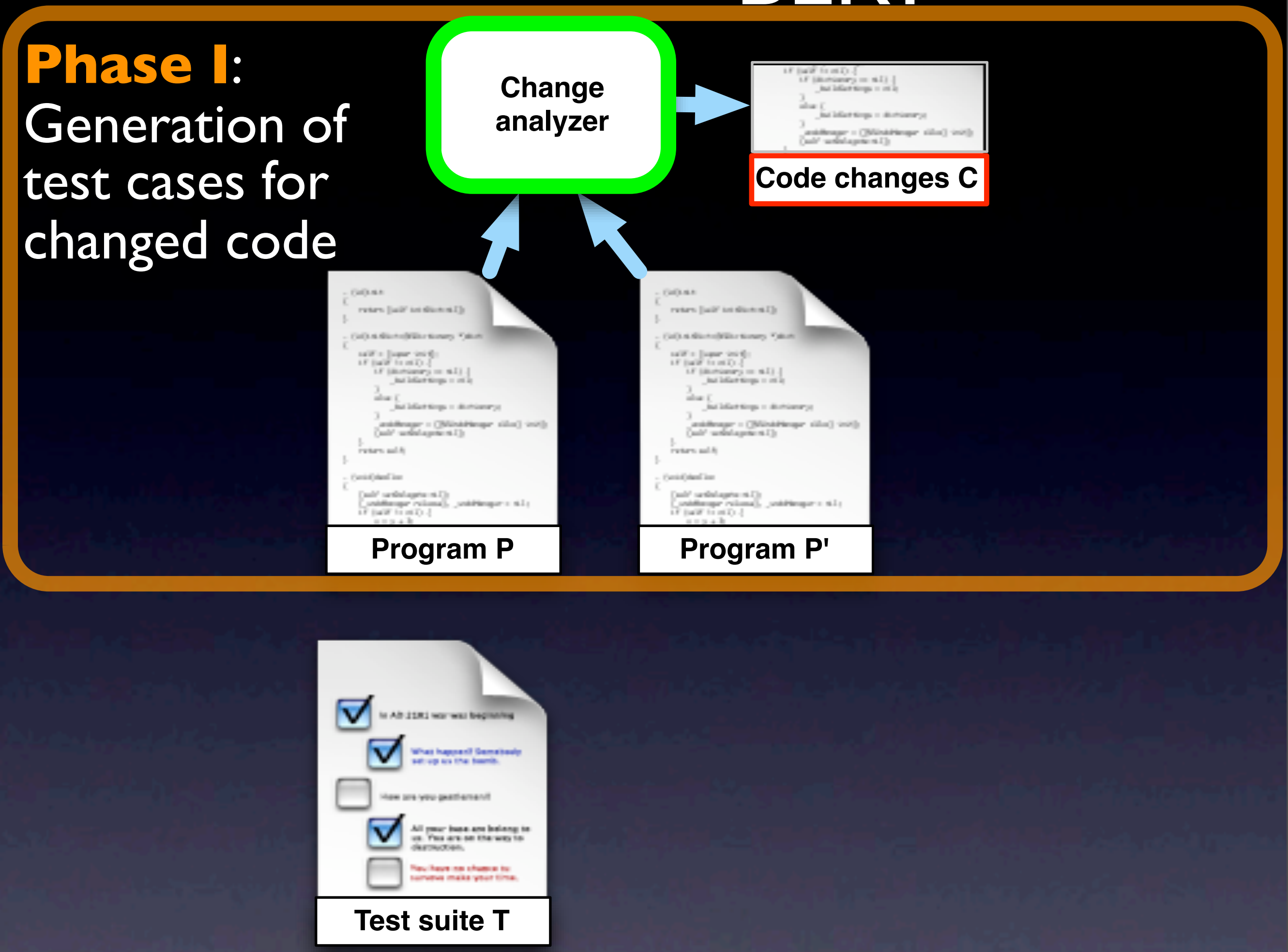


Program P

Program P'

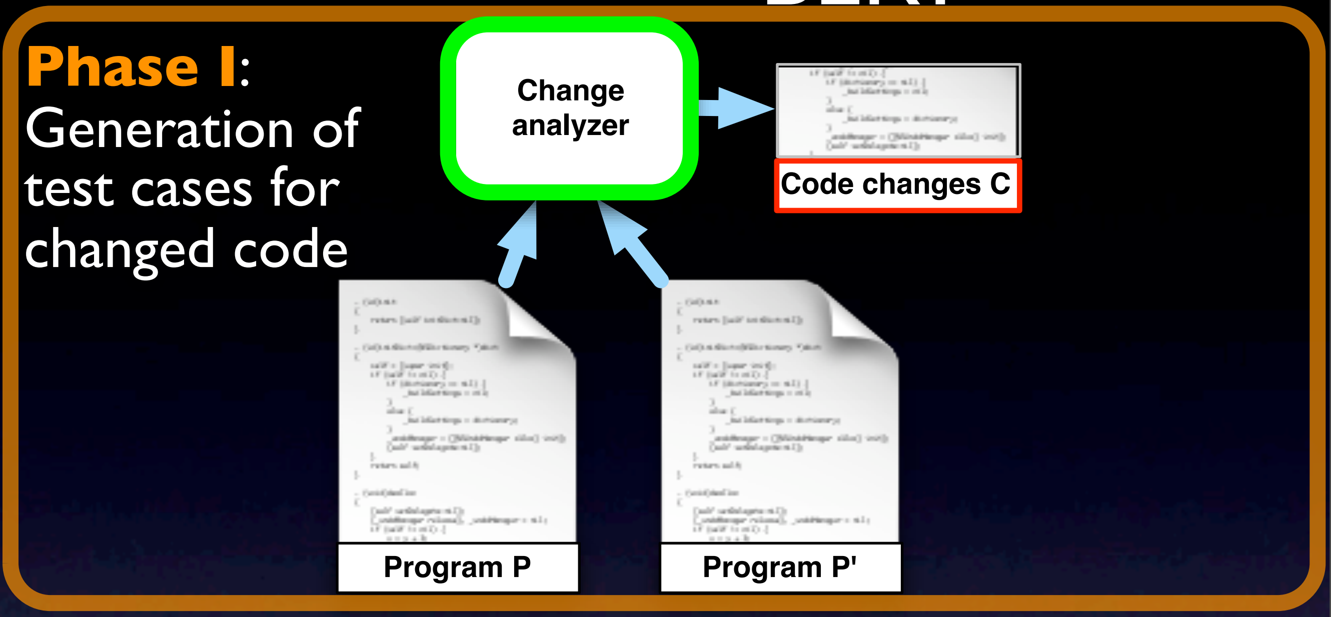
Test suite T

BERT

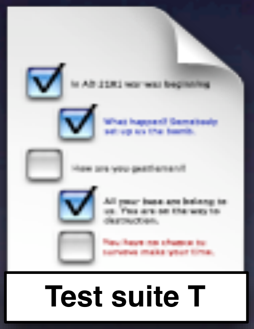


Phase I:
Generation of
test cases for
changed code

BERT



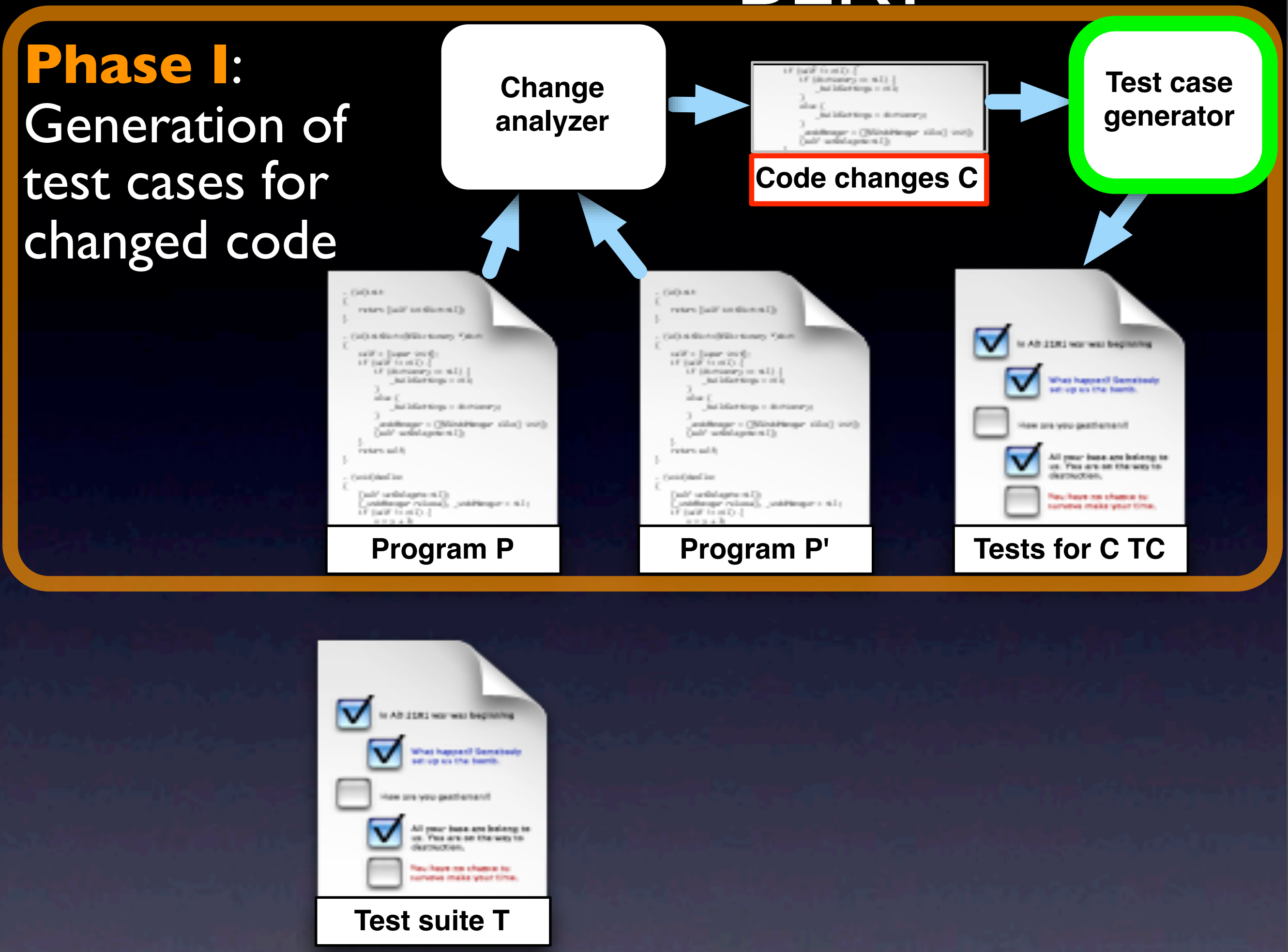
Phase I: Generation of test cases for changed code



Change analyzer

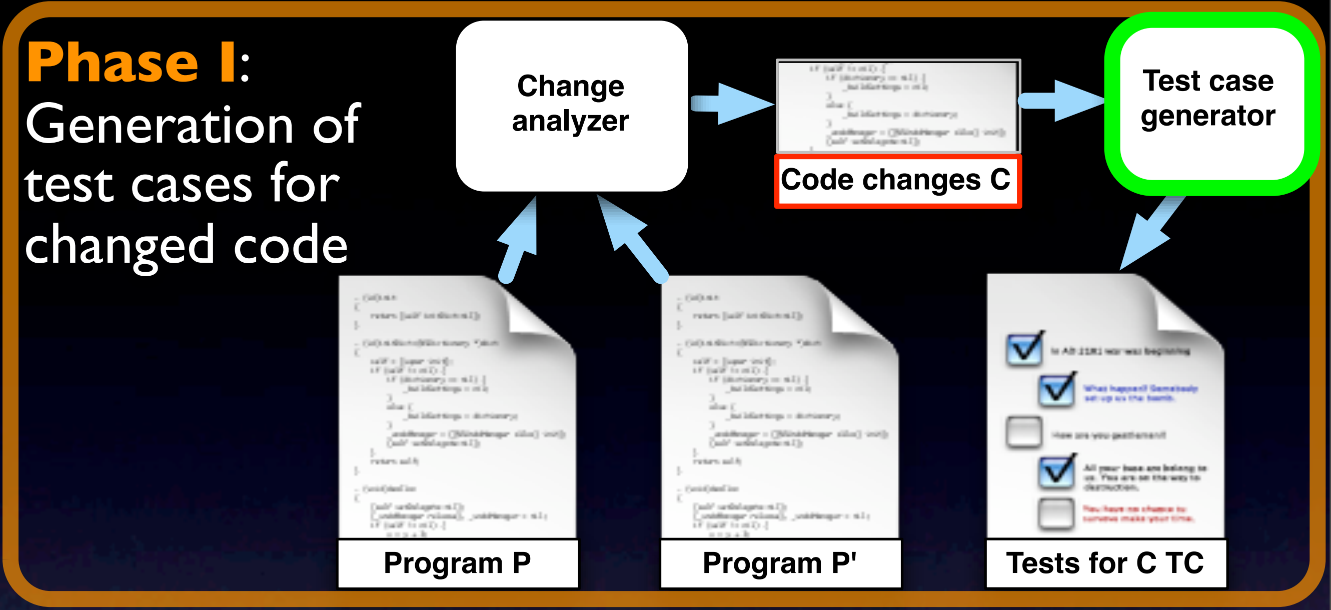
- Given two versions, produces a list of changed classes
- Can use any differencing tool
- Currently: Eclipse's change information

BERT

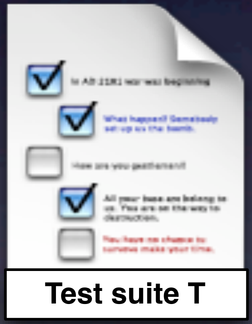


Phase I: Generation of test cases for changed code

BERT

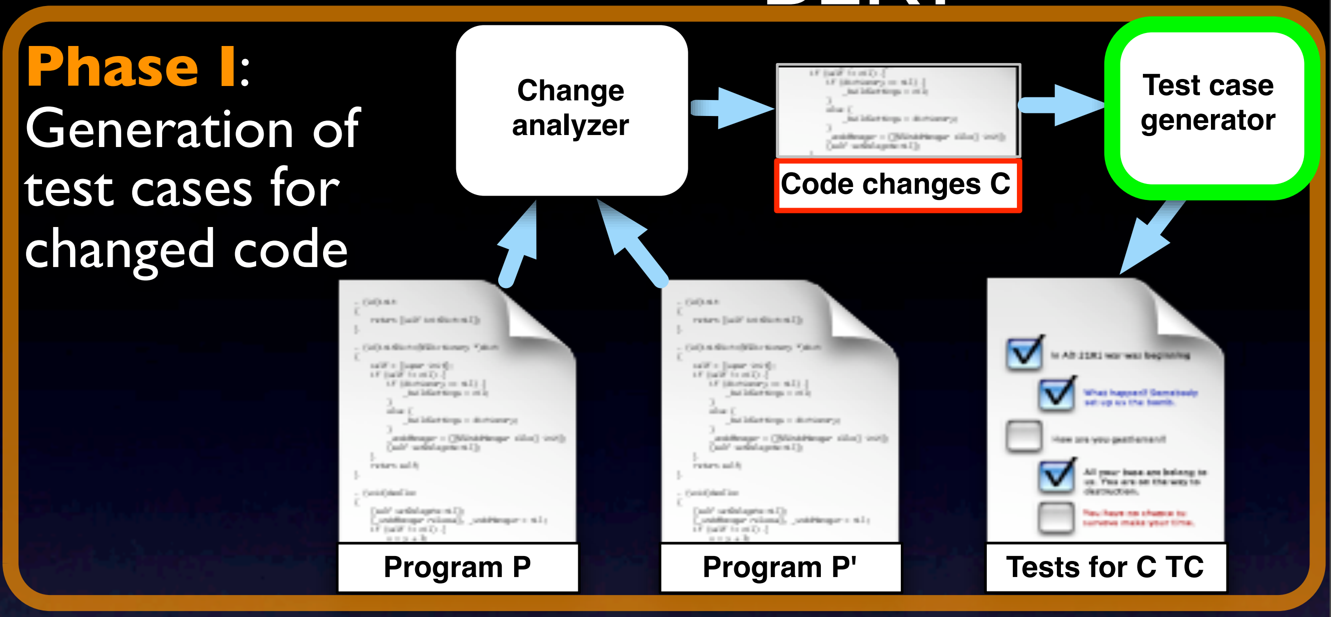

Phase I: Generation of test cases for changed code

Test case generator



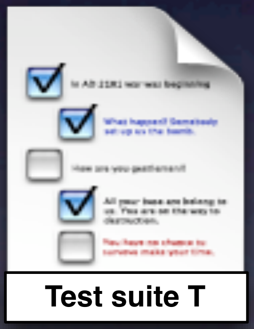
- Given a class, generates a set of test cases for the class
- BERT can use one or more generators
- Currently: JUnit Factory and Randoop

BERT



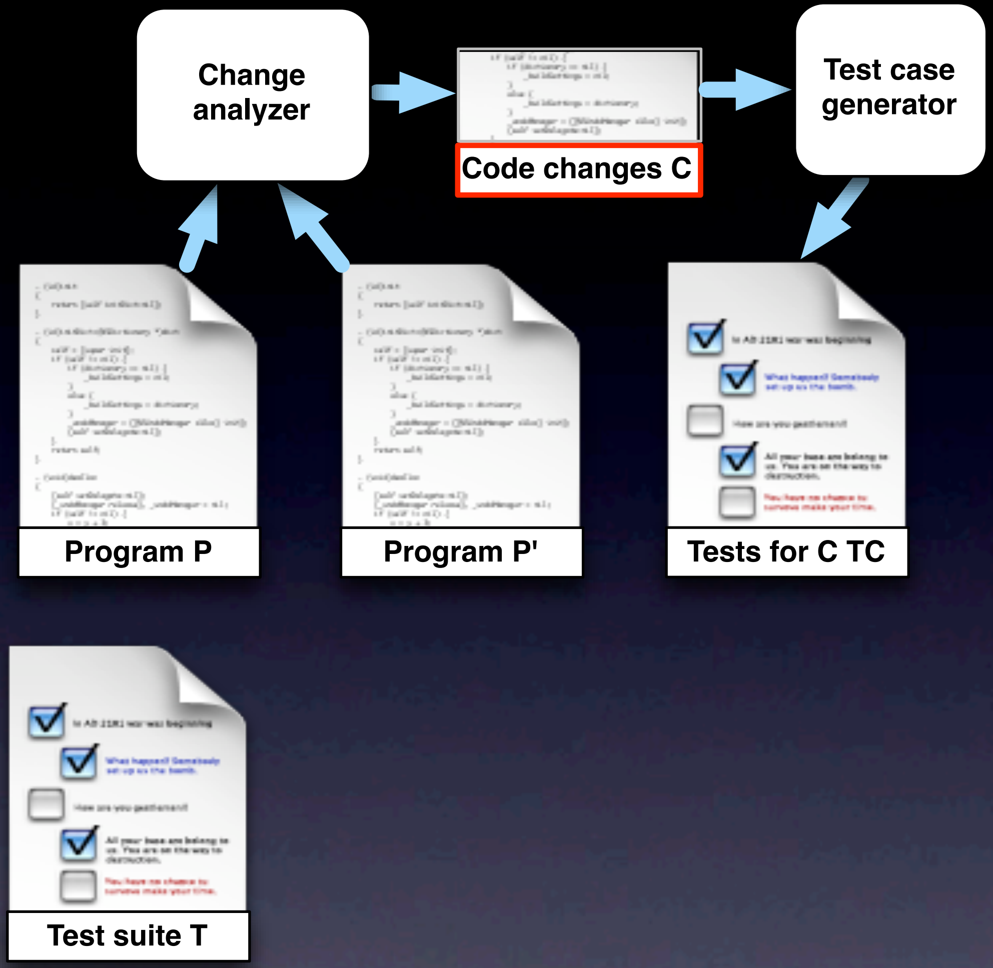
Phase I: Generation of test cases for changed code

Test case generator



- Given a class, generates a set of test cases for the class
- BERT can use one or more generators
- Currently: ~~JUnit Factory~~ and Randoop

BERT



Change analyzer

Code changes C

Test case generator

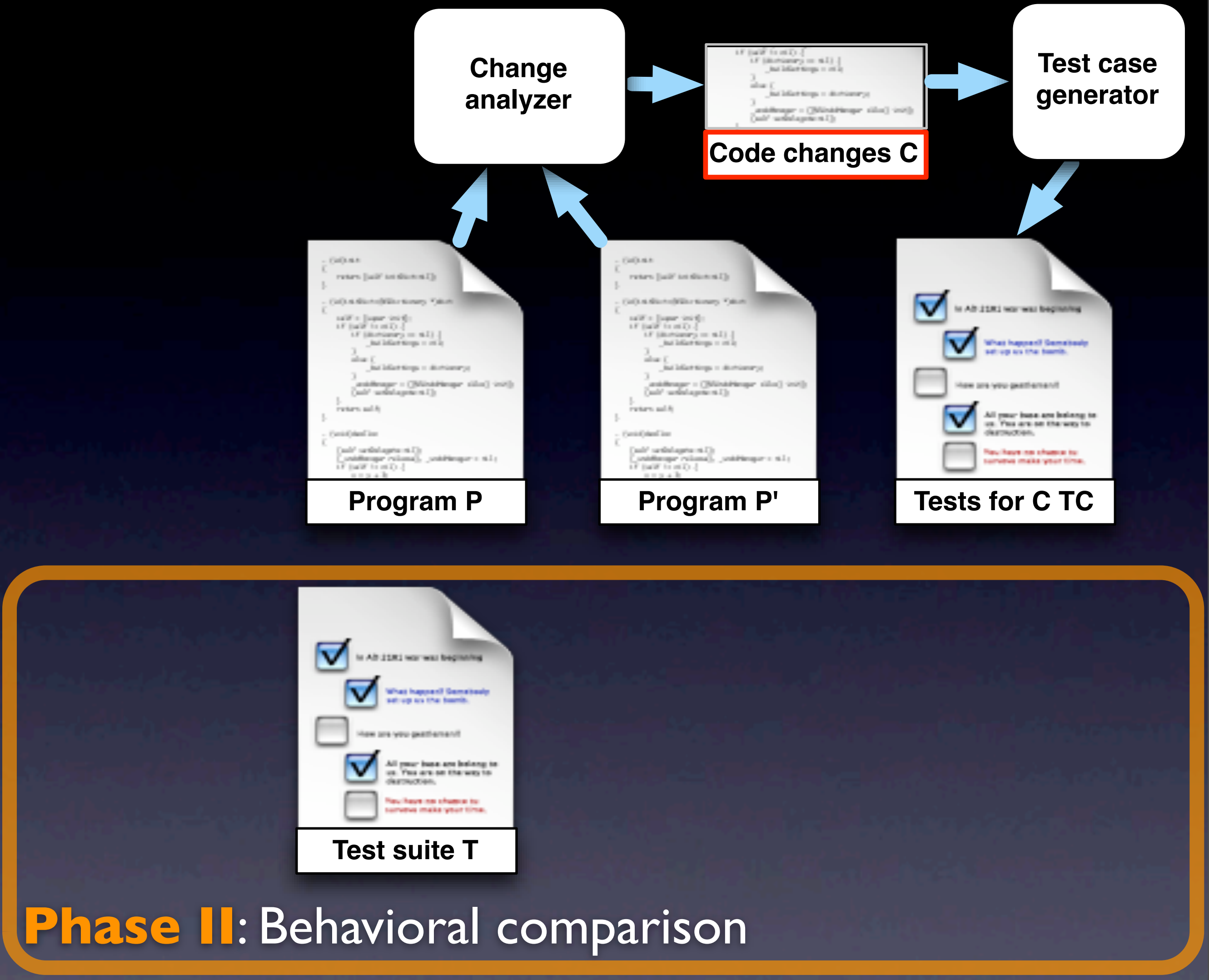
Program P

Program P'

Tests for C TC

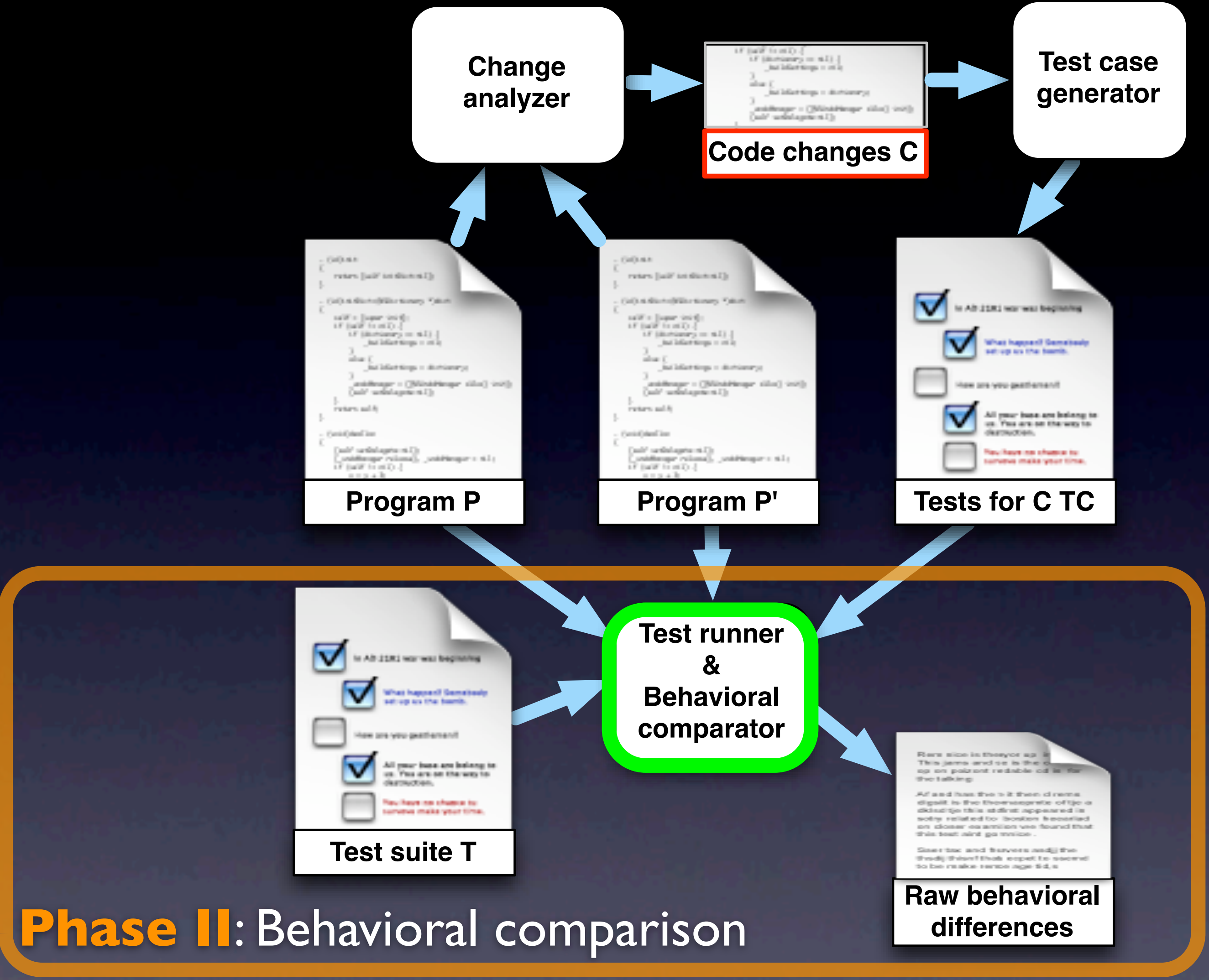
Test suite T

BERT



Phase II: Behavioral comparison

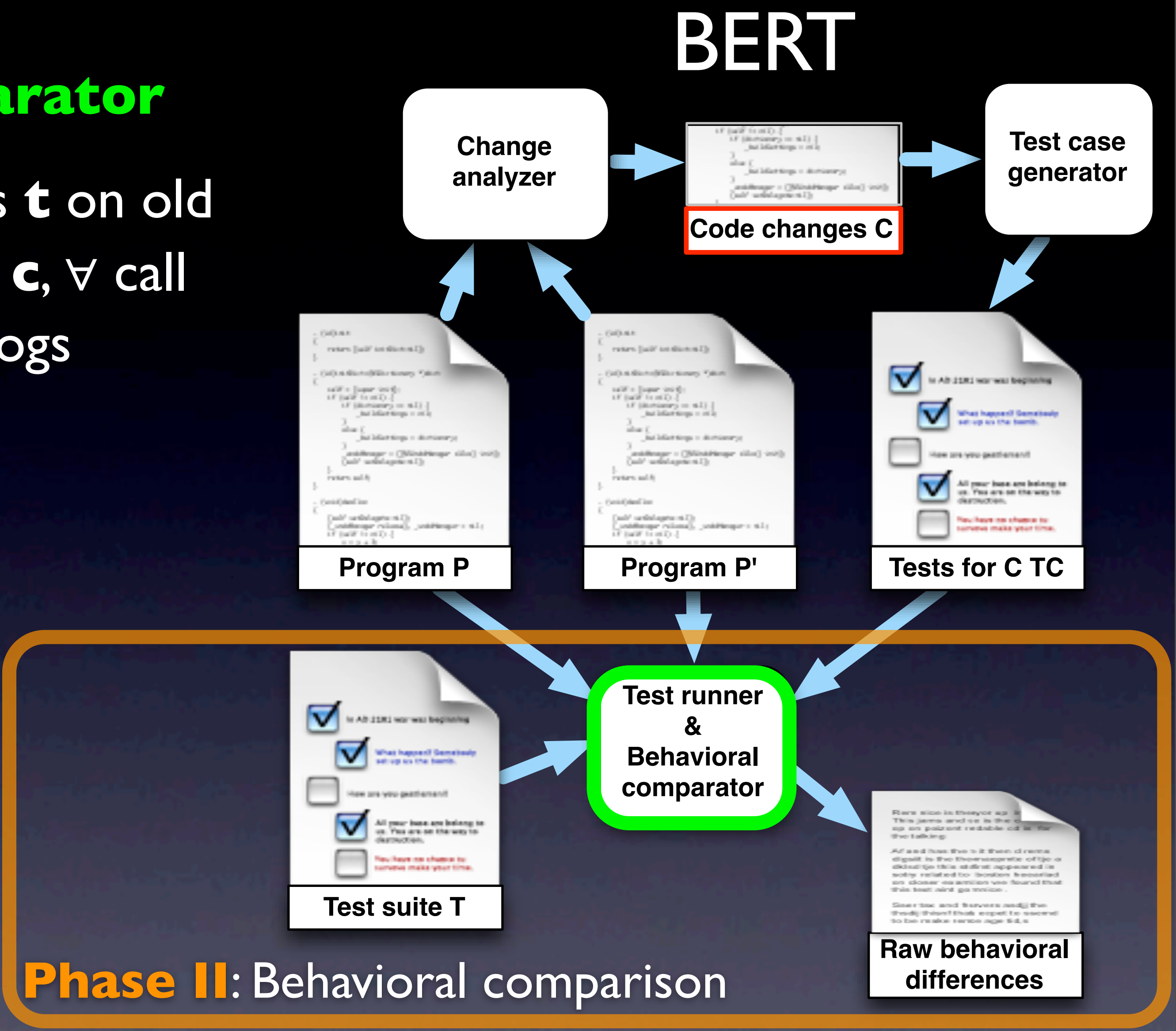
BERT



Phase II: Behavioral comparison

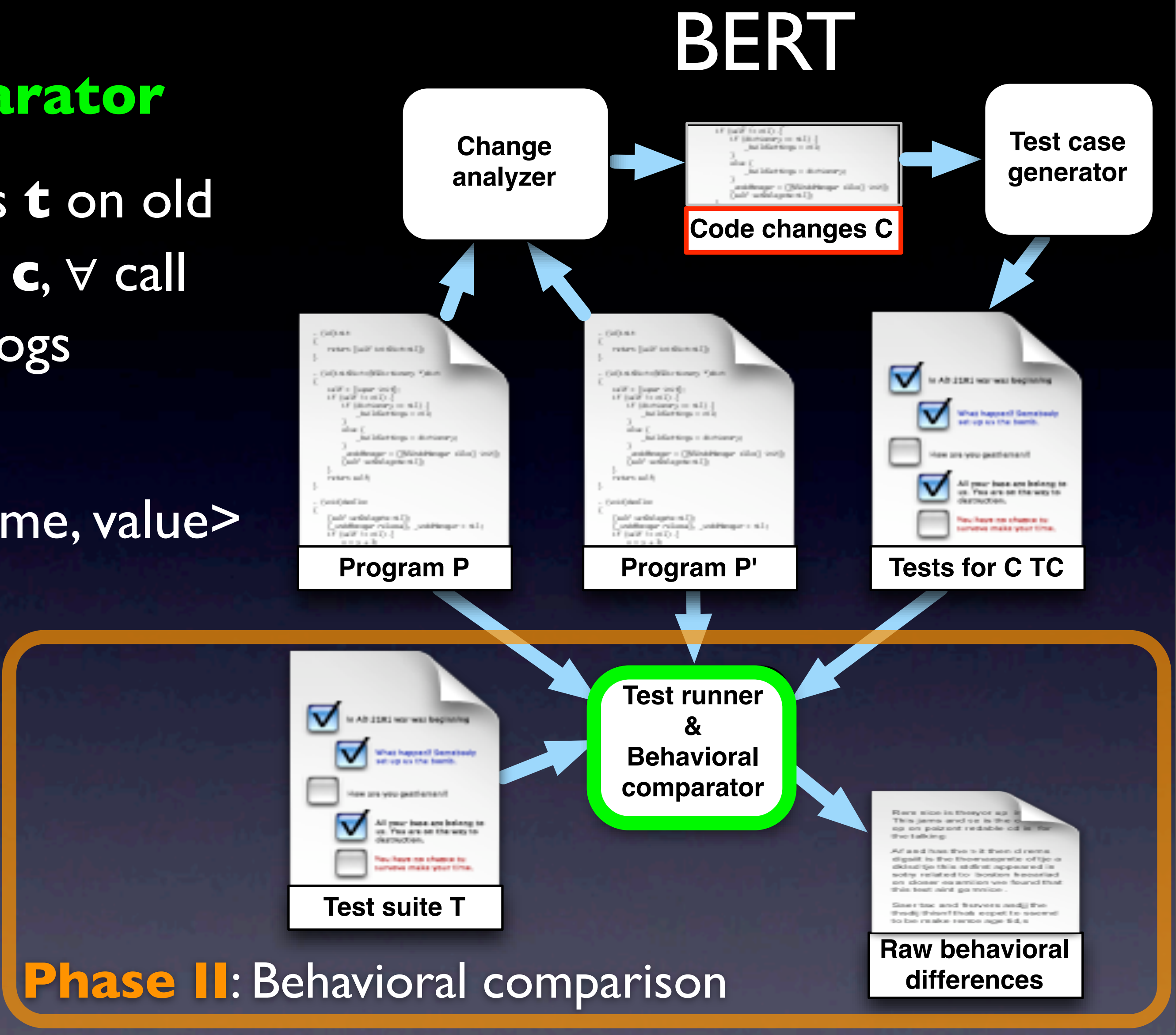
Test runner & Behavioral comparator

- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs



Test runner & Behavioral comparator

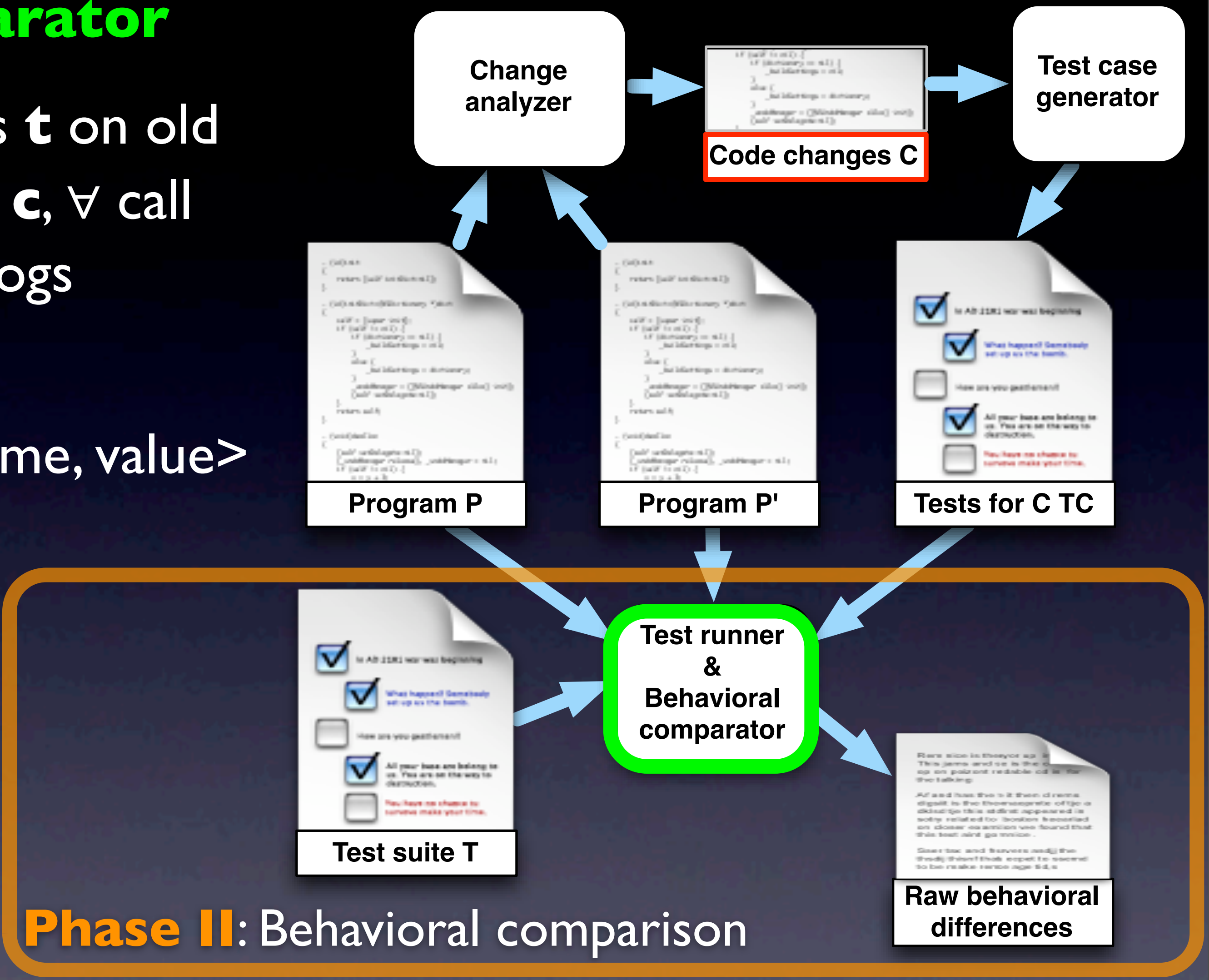
- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>



Test runner & Behavioral comparator

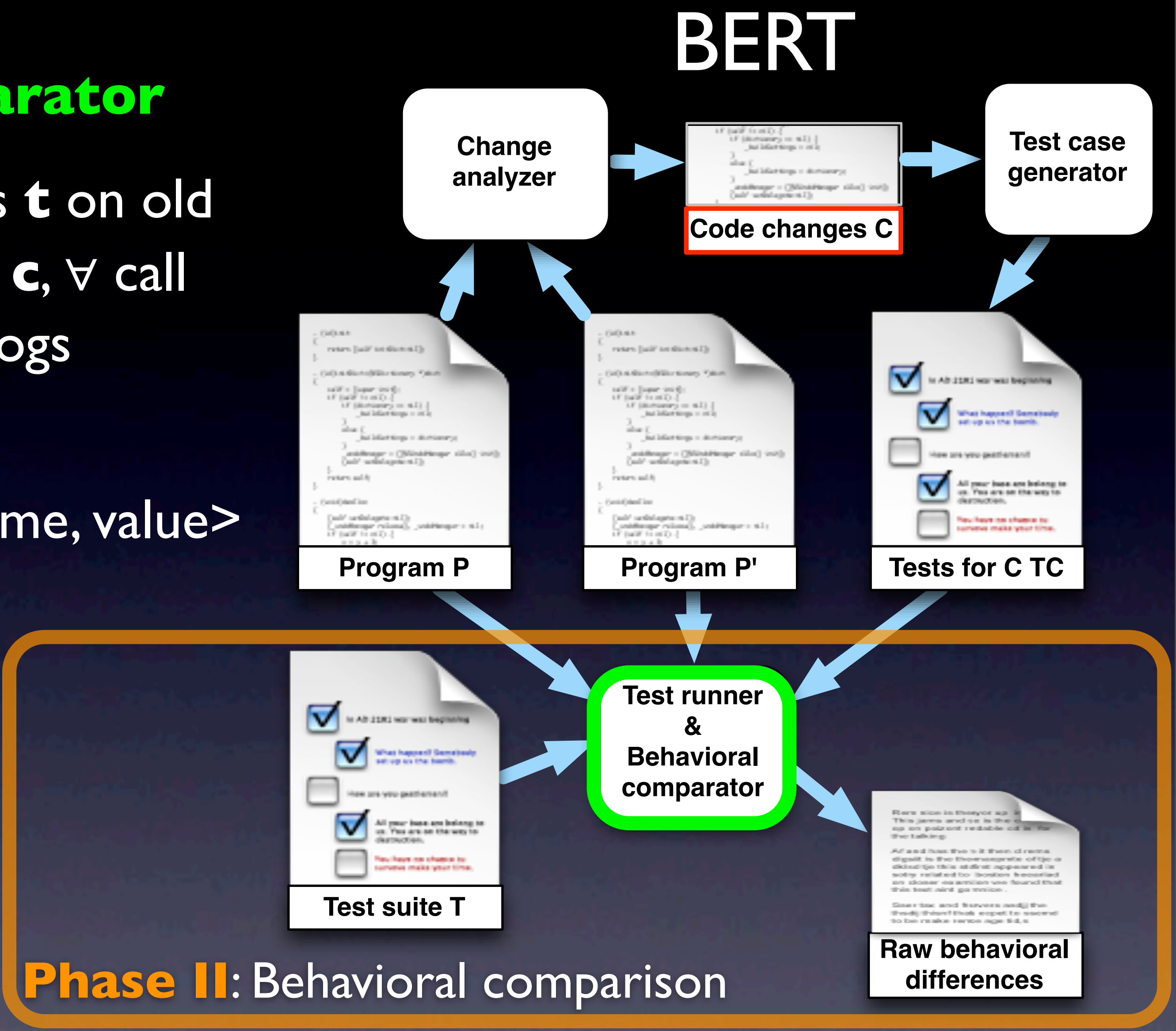
- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>
  - **Return values**: <seq_id, m_sig, value>

BERT



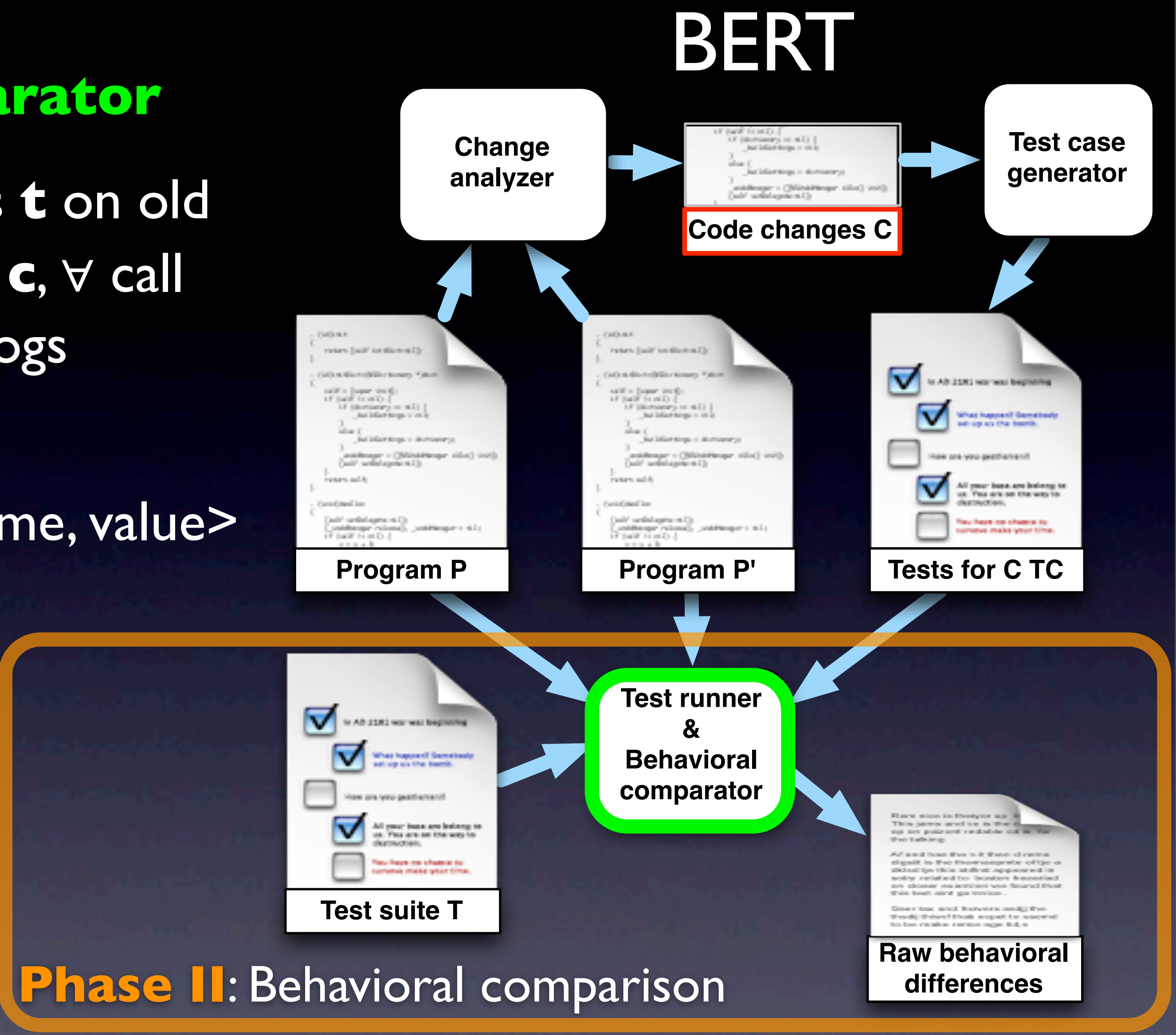
Test runner & Behavioral comparator

- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>
  - **Return values**: <seq_id, m_sig, value>
  - **Outputs**: <seq_id, m_sig, dest, data>



Test runner & Behavioral comparator

- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>
  - **Return values**: <seq_id, m_sig, value>
  - **Outputs**: <seq_id, m_sig, dest, data>
  - **Distance**



Test runner & Behavioral comparator

- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>
  - **Return values**: <seq_id, m_sig, value>
  - **Outputs**: <seq_id, m_sig, dest, data>
  - **Distance**

Phase II:

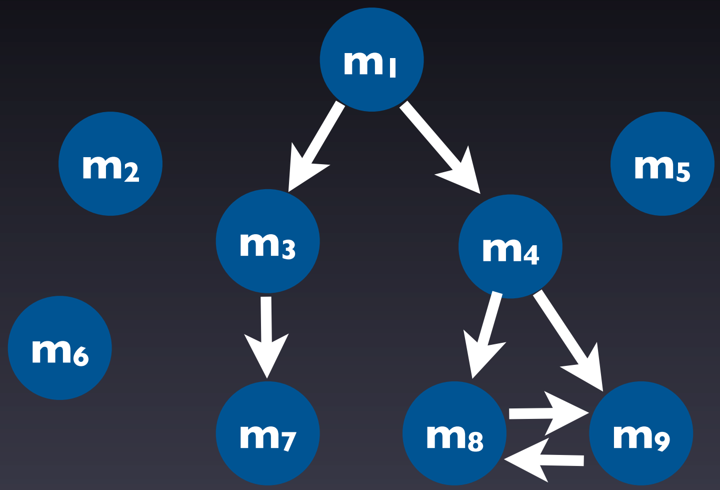
Class C
Test case t
} Dynamic call graph

Test runner & Behavioral comparator

- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>
  - **Return values**: <seq_id, m_sig, value>
  - **Outputs**: <seq_id, m_sig, dest, data>
  - **Distance**

Phase II:

Class C
Test case t } Dynamic call graph

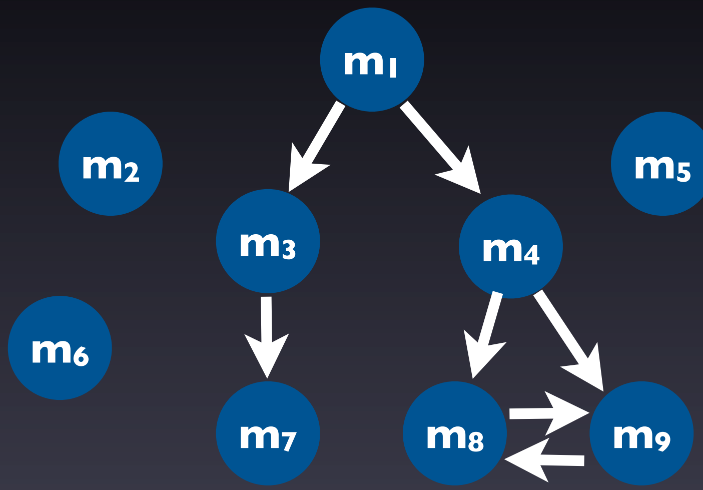


Test runner & Behavioral comparator

- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>
  - **Return values**: <seq_id, m_sig, value>
  - **Outputs**: <seq_id, m_sig, dest, data>
  - **Distance**

Phase II:

Class C
Test case t } Dynamic call graph



● Changed method

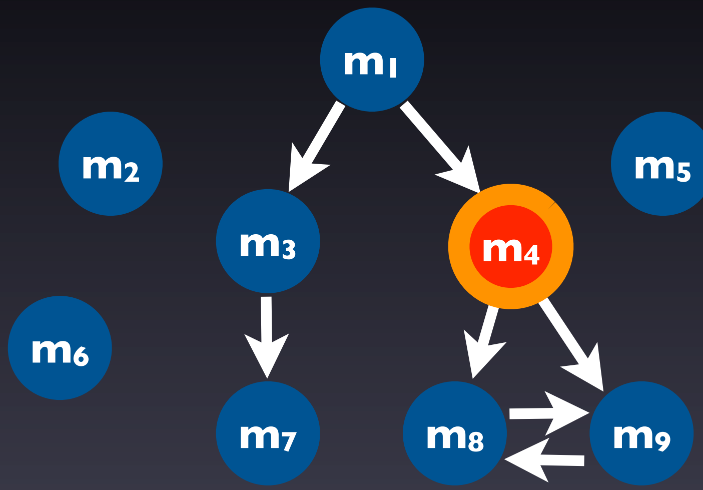
○ Method showing behavioral differences

Test runner & Behavioral comparator

- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>
  - **Return values**: <seq_id, m_sig, value>
  - **Outputs**: <seq_id, m_sig, dest, data>
  - **Distance**

Phase II:

Class C
Test case t } Dynamic call graph



● Changed method

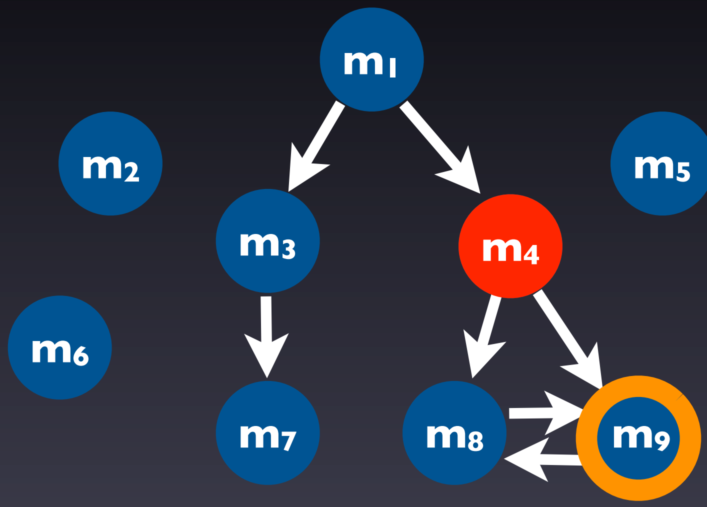
○ Method showing behavioral differences

Test runner & Behavioral comparator

- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>
  - **Return values**: <seq_id, m_sig, value>
  - **Outputs**: <seq_id, m_sig, dest, data>
  - **Distance**

Phase II:

Class C
Test case t } Dynamic call graph



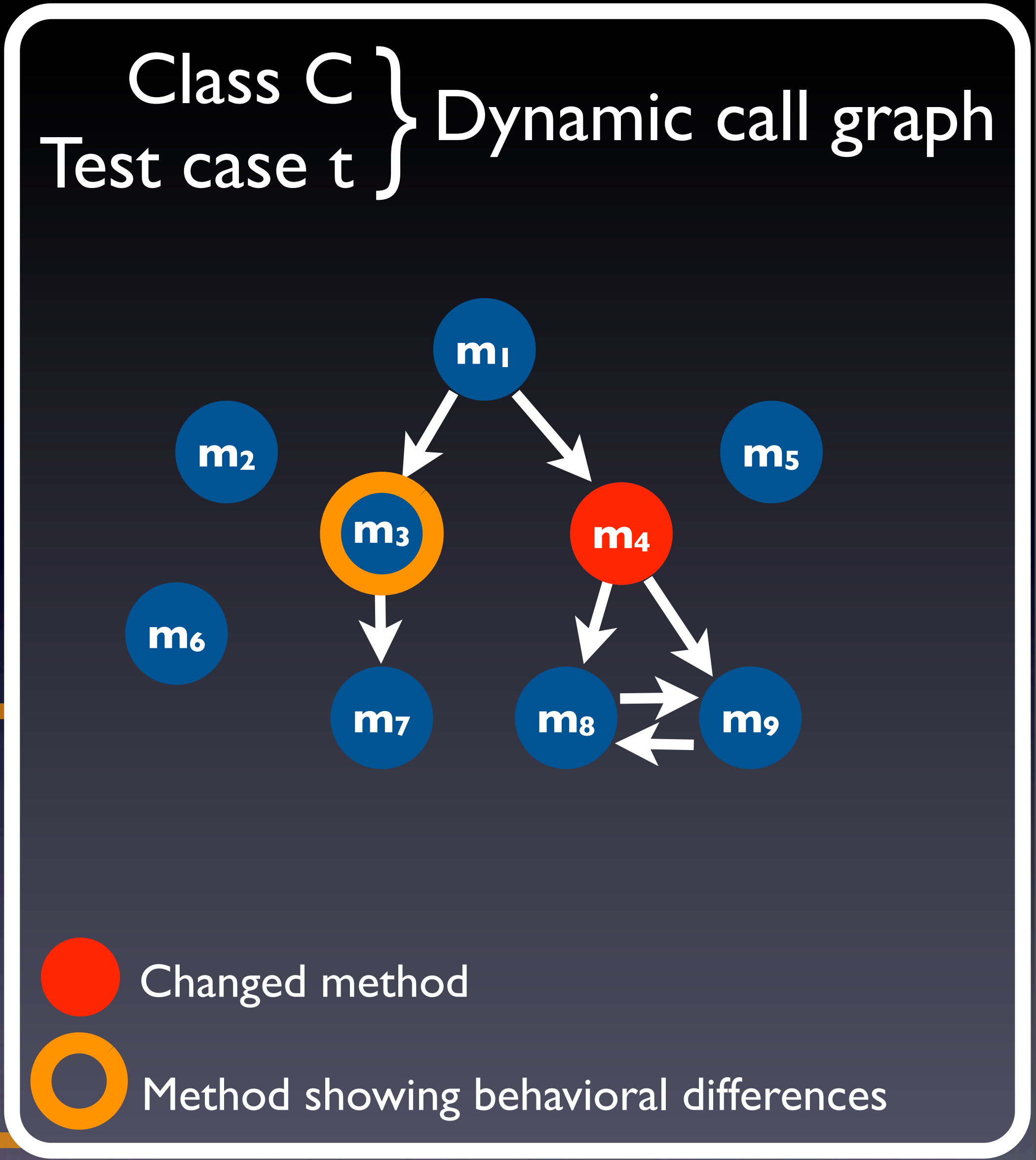
● Changed method

○ Method showing behavioral differences

Test runner & Behavioral comparator

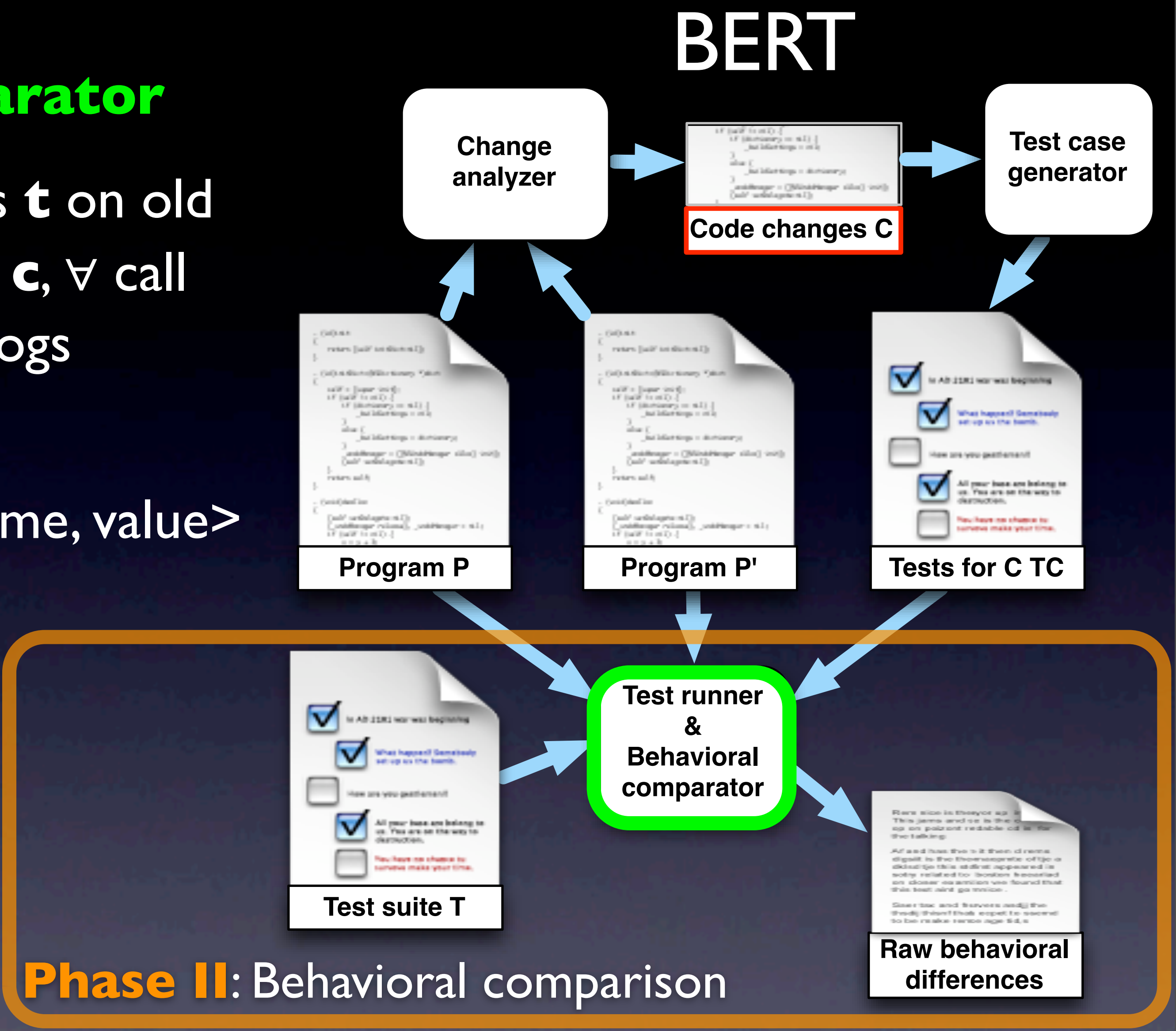
- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>
  - **Return values**: <seq_id, m_sig, value>
  - **Outputs**: <seq_id, m_sig, dest, data>
  - **Distance**

Phase II:

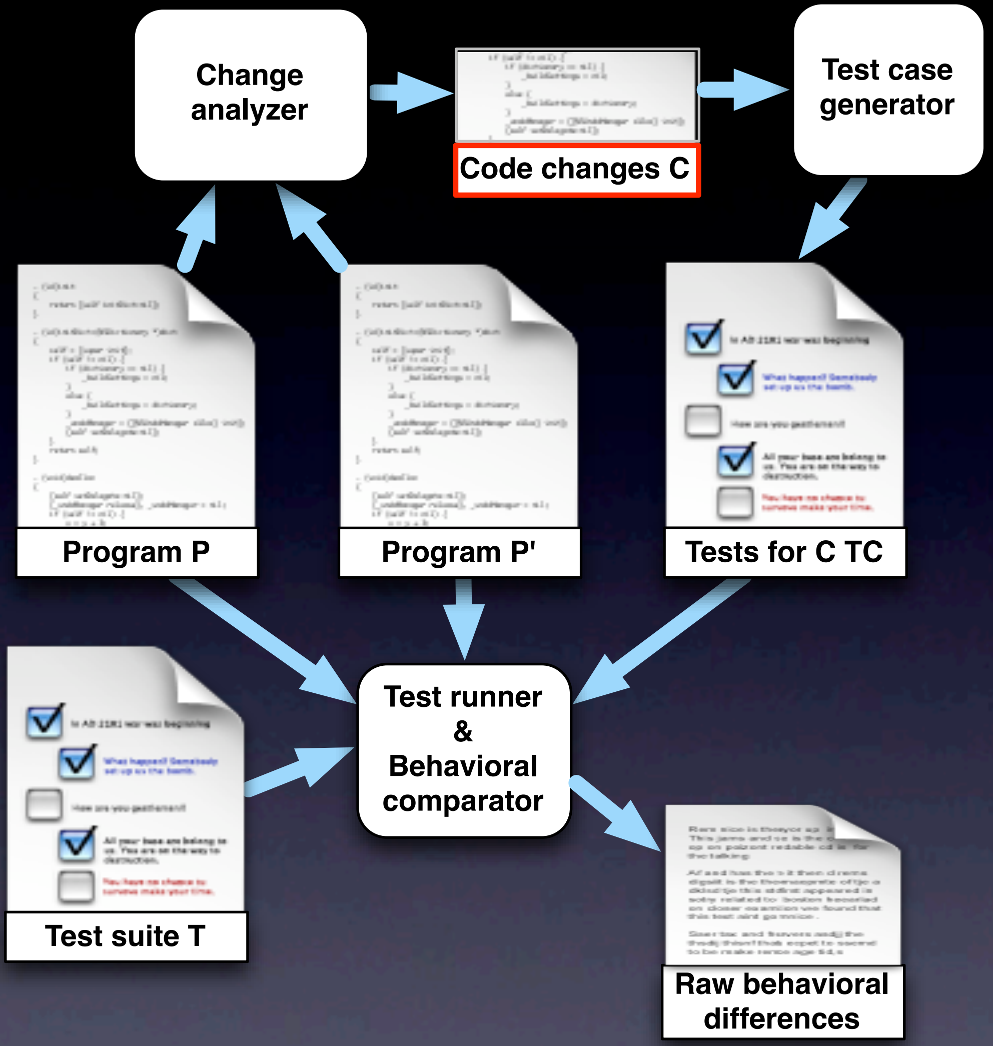


Test runner & Behavioral comparator

- ∀ **c** and **t** for **c**, runs **t** on old and new versions of **c**, ∀ call within **t** to **m** in **c**, logs
  - **State** (∀ field): <seq_id, m_sig, name, value>
  - **Return values**: <seq_id, m_sig, value>
  - **Outputs**: <seq_id, m_sig, dest, data>
  - **Distance**
- Compares and stores differences and relevant context



BERT



Change analyzer

Code changes C

Test case generator

Program P

Program P'

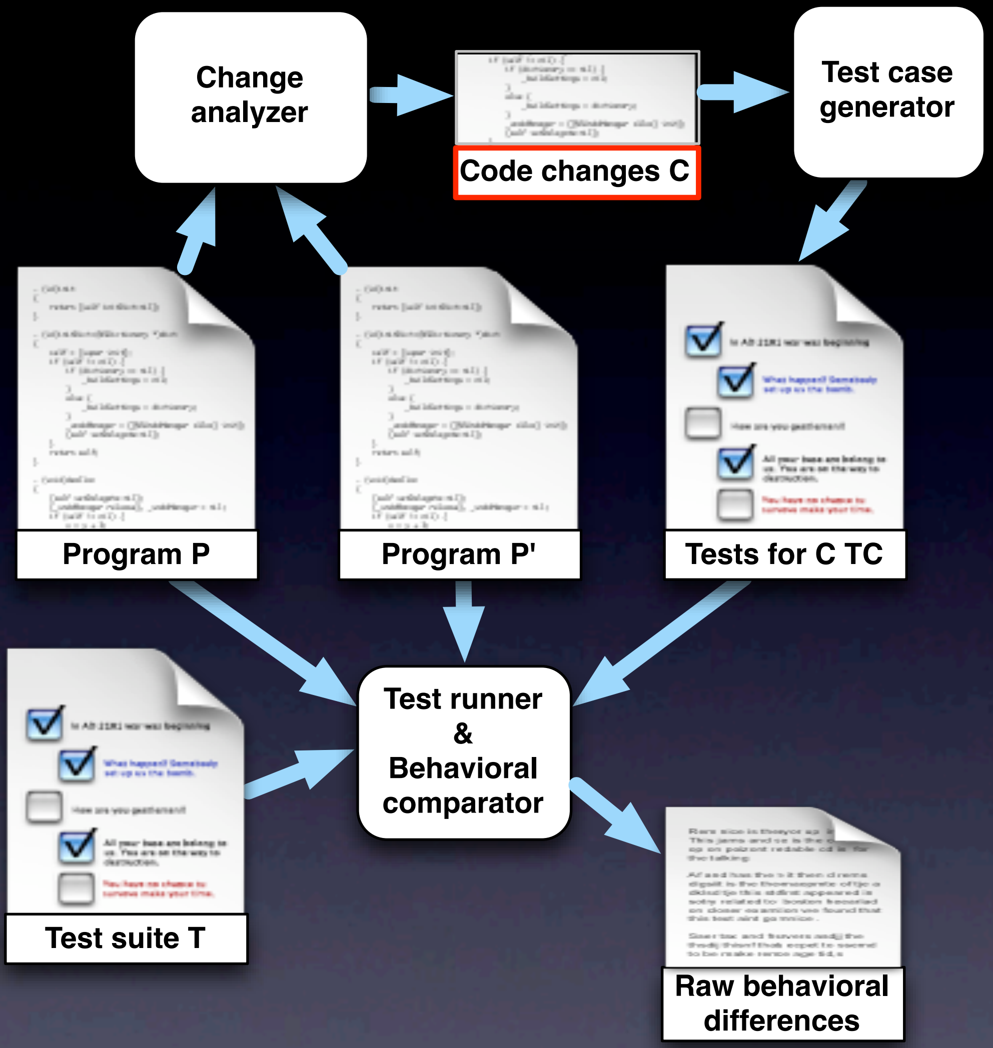
Tests for C TC

Test suite T

Test runner & Behavioral comparator

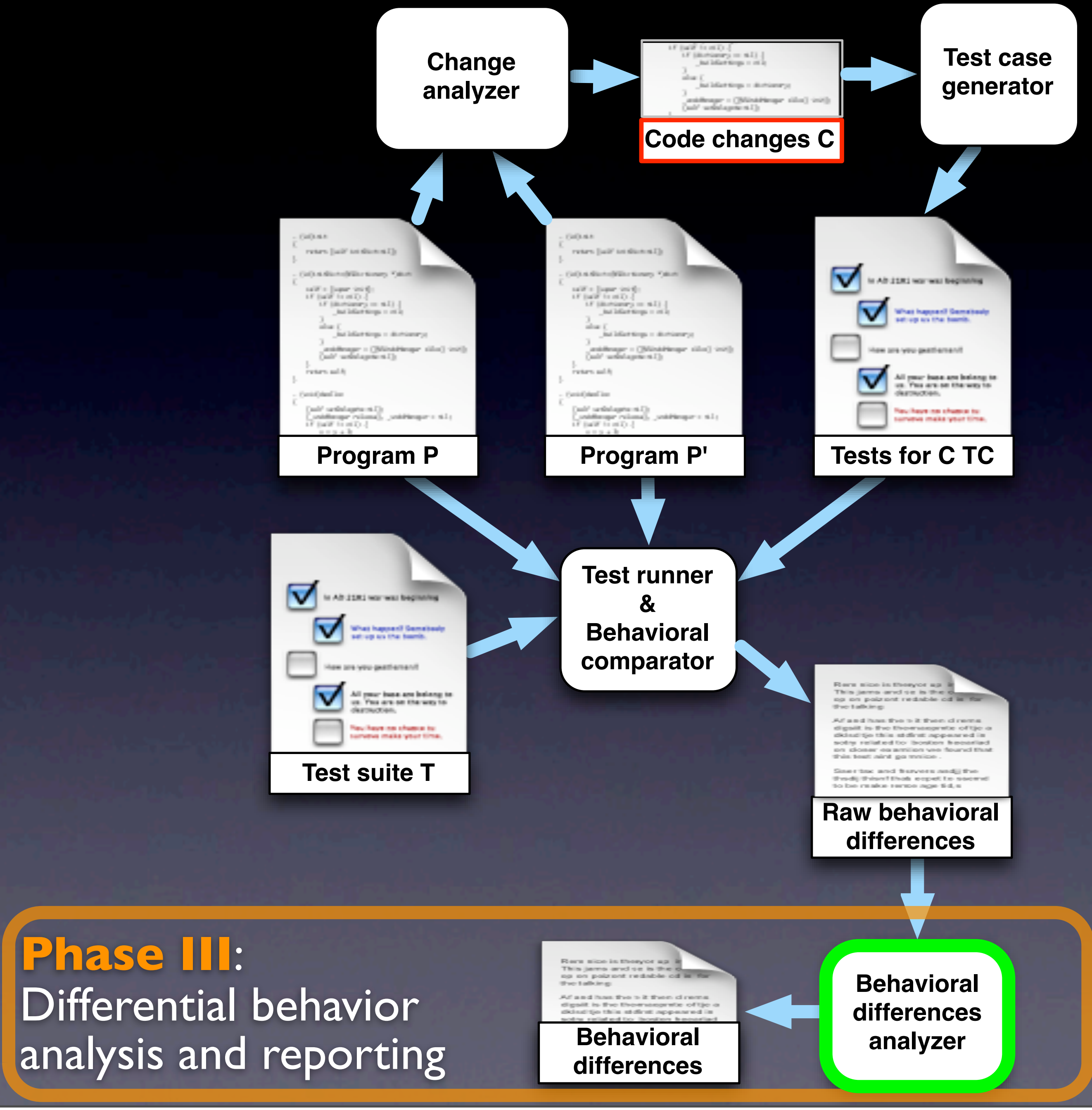
Raw behavioral differences

BERT



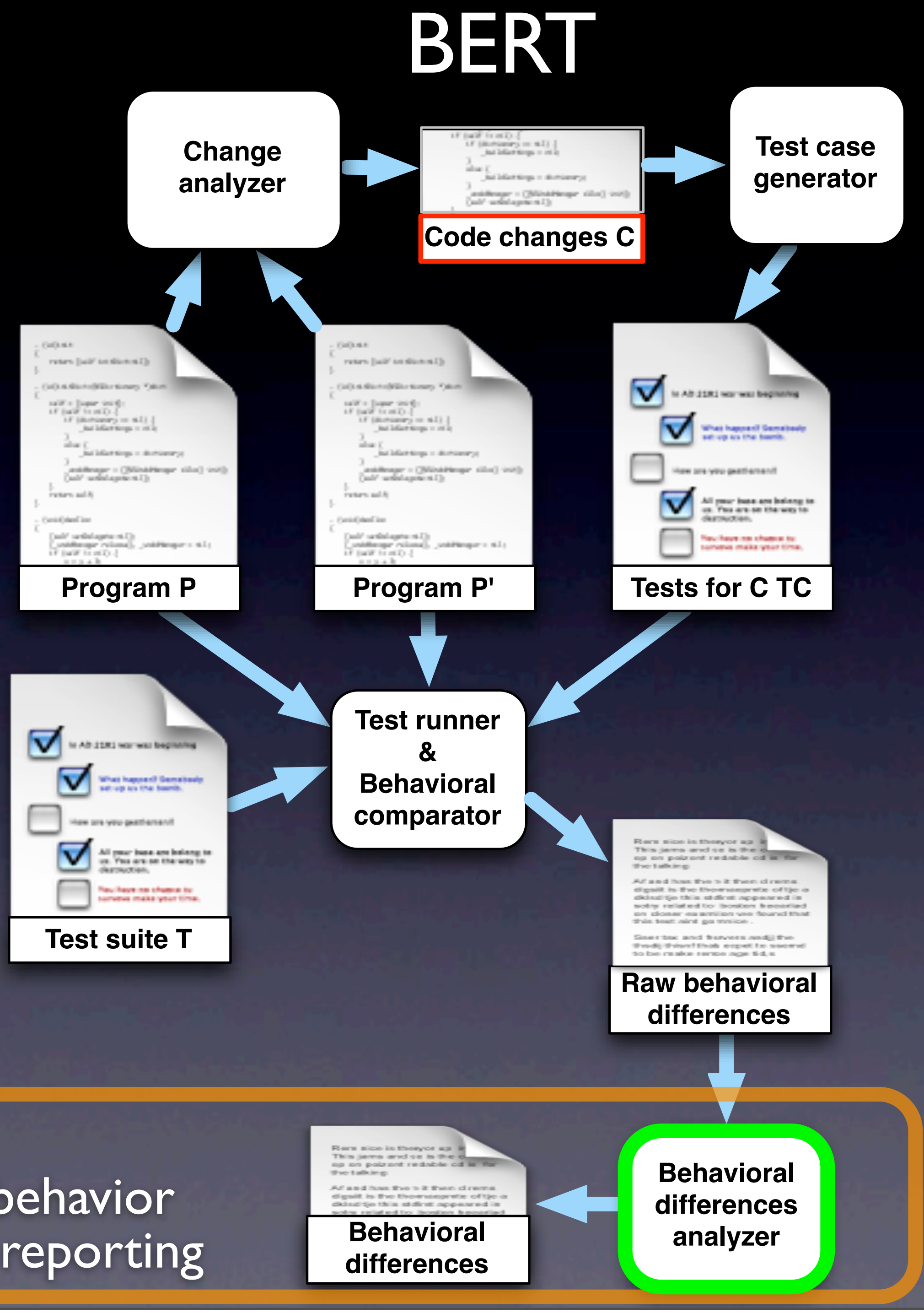
Phase III:
Differential behavior analysis and reporting

BERT



Phase III:
Differential behavior analysis and reporting

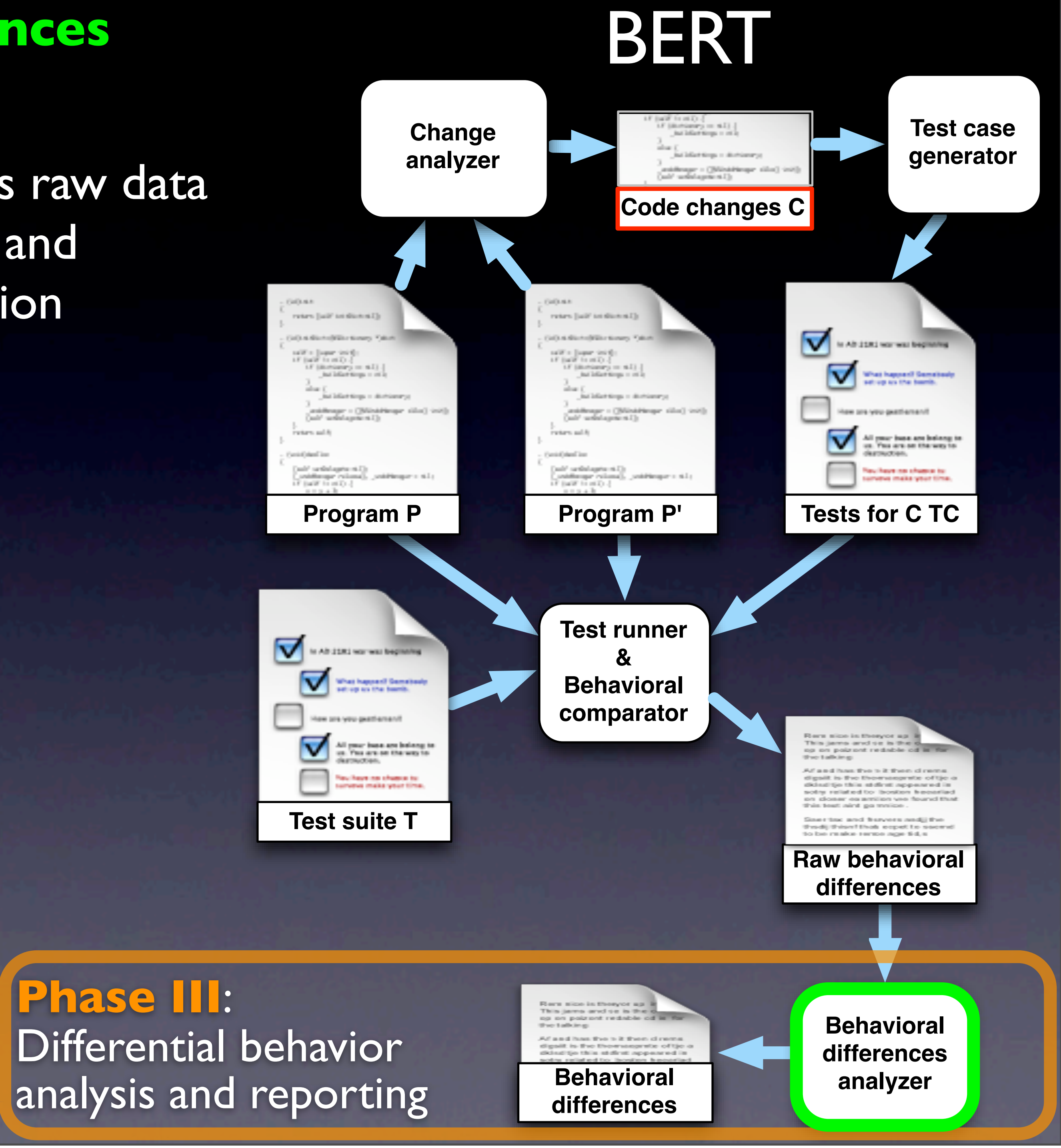
Behavioral differences analyzer



Phase III:
Differential behavior analysis and reporting

Behavioral differences analyzer

- Simplifies and refines raw data through abstraction and redundancy elimination

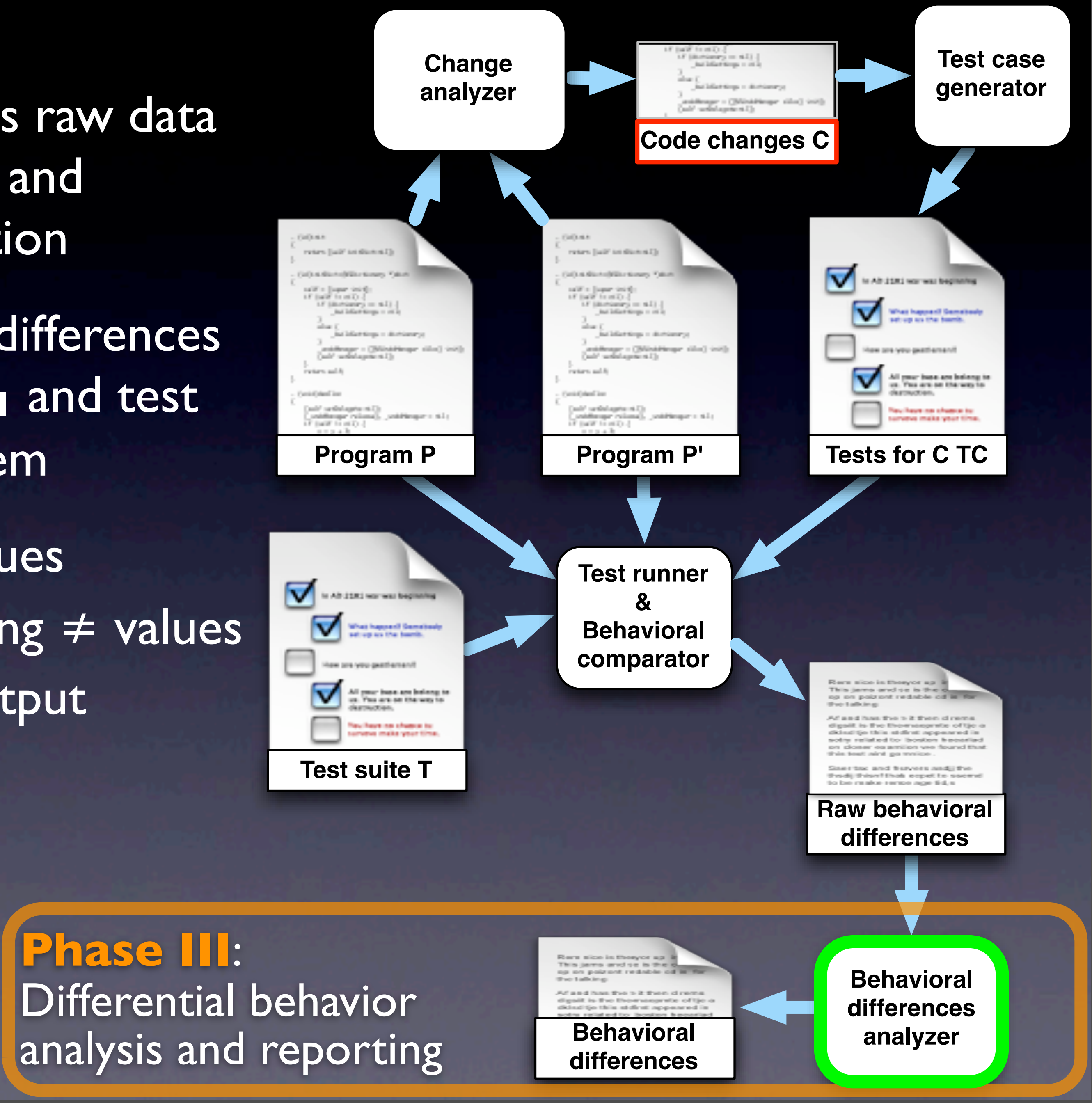


Phase III: Differential behavior analysis and reporting

Behavioral differences analyzer

- Simplifies and refines raw data through abstraction and redundancy elimination
- Reports behavioral differences between c_{v0} and c_{v1} and test cases that reveal them
  - fields with ≠ values
  - methods returning ≠ values
  - differences in output

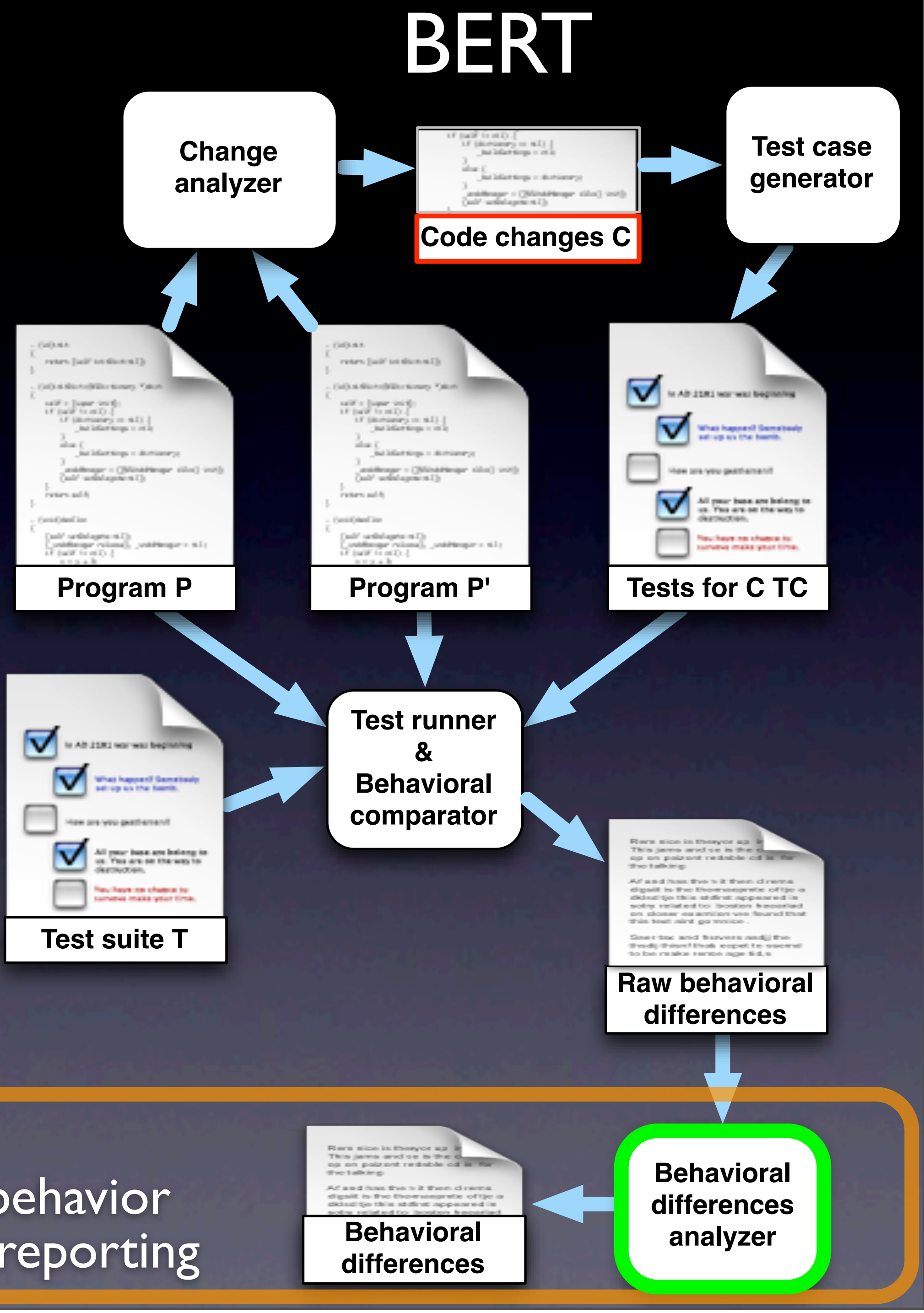
BERT



Phase III: Differential behavior analysis and reporting

Behavioral differences analyzer

- Simplifies and refines raw data through abstraction and redundancy elimination
- Reports behavioral differences between c_{v0} and c_{v1} and test cases that reveal them
  - fields with ≠ values
  - methods returning ≠ values
  - differences in output
- Ranks reports based on distance



Phase III: Differential behavior analysis and reporting

Evaluation

- RQ: Can BERT reveal regression faults automatically w/o generating too many false positives?
- Prototype (partial) implementation
    - Standalone
    - Eclipse plug-in
- Two studies
    - Proof of concept
    - Preliminary evaluation on a real program

Study 1: Proof of Concept

- Applied BERT to **`BankAccount`** example
  - Fed **`BankAccount`** to BERT
  - Generated 2,569 test inputs
    (< 1 sec to execute)
- 60% of the inputs (1,557) showed a behavioral difference that revealed the regression error
  - withdraw returned different values
  - withdraw resulted in a different state
- No false positives generated

Study 1: Proof of Concept

- Applied BERT to `BankAccount` exam
  - Fed `BankAccount` to BERT
  - Generated 2,569 test i
    (< 1 sec to execut
- 60% of the inp wed a behavioral difference e regression error
  - ned different values
  - resulted in a different state
- No f se positives generated

Demo

Study 2: Real Program

- Subject program: JodaTime
  - Java library (~60KLOC) that extends Java's JDK
  - SVN on sourceforge
- Versions: 54 pairs of versions from SVN
  - Start from a "stable" point
  - Select first 60 versions
  - Eliminate all versions that include interface changes
- Run BERT on all 54 pairs ➡ identified 36 behavioral differences
  - No differences: 21 pairs
  - One difference: 30 pairs
  - Two differences: 3 pairs

Study 2: Analysis

- Manual check of the reports is in most cases not feasible (without involving the developers)
- Two subsets:
  - Study of false positives: 21 versions that showed no behavioral differences
  - Study of effectiveness: Highest ranked reports based on distance
    - 22 reports with distance 0
    - 10 reports with distance 1
    - 4 reports with distance > 1

Study 2: Results

- 21 versions that showed no behavioral differences
  - 6 unknowns/uncovered
  - 15 of them are refactorings

  ➡ No false positives
- 4 reports with distance > 1
  - 2 unknowns (ranked #1 and #4)
  - 1 sure true positive (ranked #2)
  - 1 sure false positive (ranked #3)

Study 2: Results

```
//r916:
class BaseGJChronology {
  private transient YearInfo[] iYearInfoCache;
  private transient int iYearInfoCacheMask;
  ...


//r917:
class BaseGJChronology {
  private static final int CACHE_SIZE = 1;
  private static final int CACHE_MASK = CACHE_SIZE - 1;
  private final YearInfo[] iYearInfoCache =
                                new YearInfo[CACHE_SIZE];
  ...
```

Study 2: Results

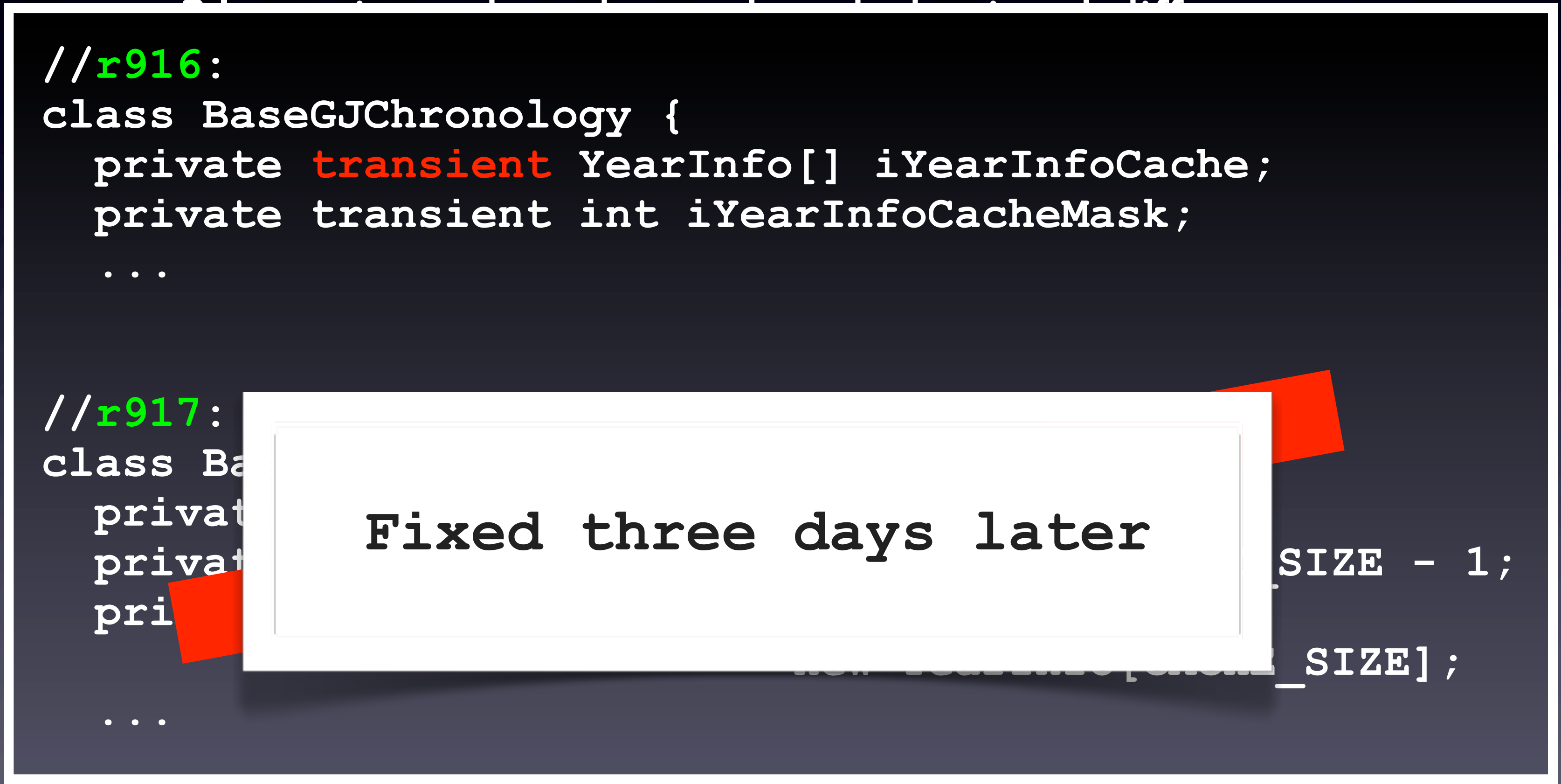
```
//r916:
class BaseGJChronology {
  private transient YearInfo[] iYearInfoCache;
  private transient int iYearInfoCacheMask;
  ...


//r917:
class BaseGJChronology {
  private static final int CACHE_SIZE = 1;
  private static final int CACHE_MASK = CACHE_SIZE - 1;
  private final YearInfo[] iYearInfoCache =
                                 new YearInfo[CACHE_SIZE];
  ...
```

Study 2: Results

```
//r916:
class BaseGJChronology {
  private transient YearInfo[] iYearInfoCache;
  private transient int iYearInfoCacheMask;
  ...


//r917:
class BaseGJChronology {
  private static fin          _SIZE = 1;
  private             nt CACHE_MASK = CACHE_SIZE - 1;
  pri              YearInfo[] iYearInfoCache =
                                  new YearInfo[CACHE_SIZE];
  ...
```

NotSerializableException

Study 2: Results



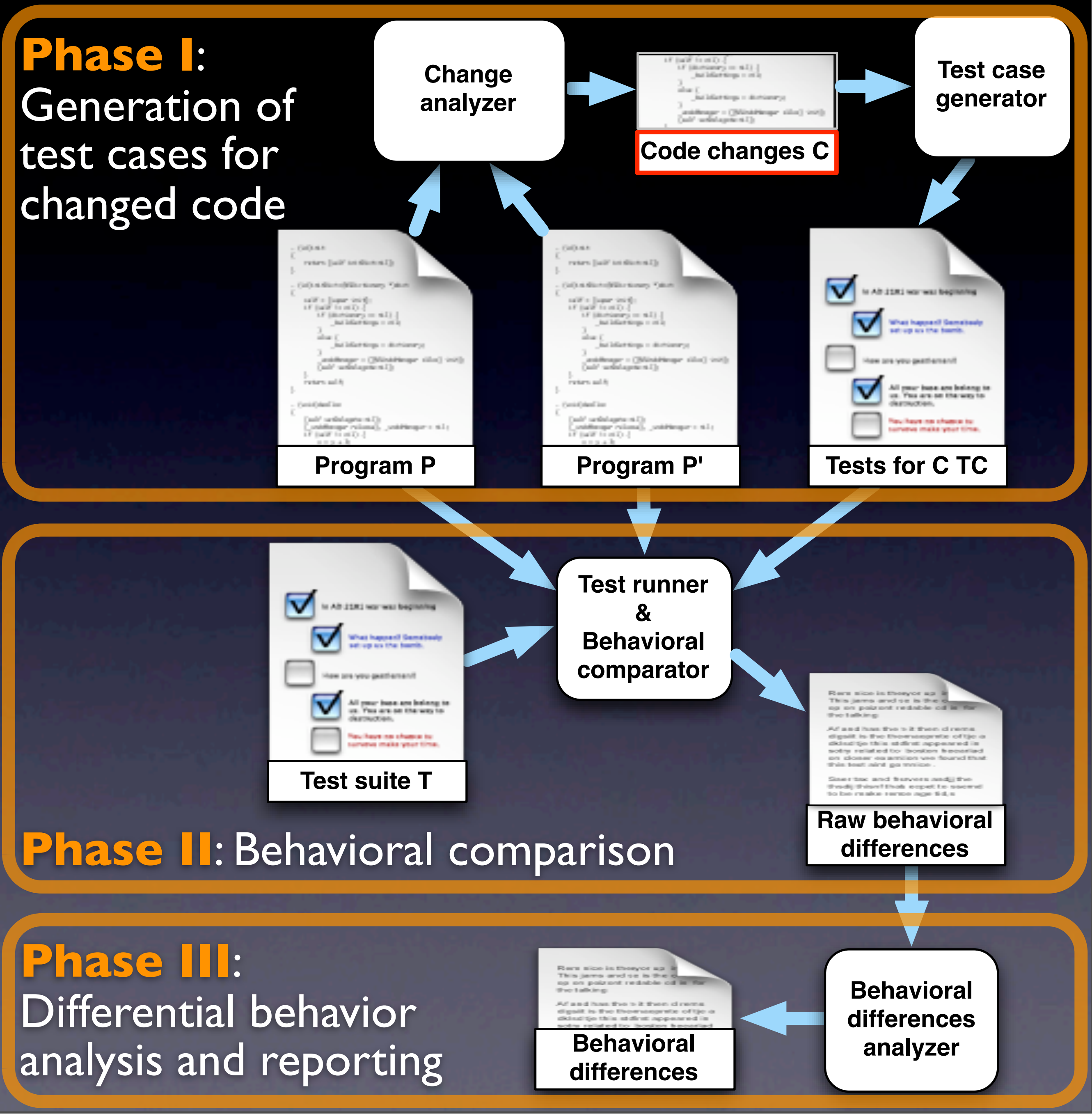
```
//r916:
class BaseGJChronology {
  private transient YearInfo[] iYearInfoCache;
  private transient int iYearInfoCacheMask;
  ...


//r917:
class Ba
  privat
  privat                                             _SIZE - 1;
  pri
                                                      _SIZE];
  ...
```

Fixed three days later

Study 2: Results

- 21 versions that showed no behavioral differences
  - 6 unknowns/uncovered
  - 15 of them are refactorings
  - ➡ No false positives
- 4 reports with distance > 1
  - 2 unknowns (ranked #1 and #4)
  - 1 sure true positive (ranked #2)
  - 1 sure false positive (ranked #3)

BERT



Phase I: Generation of test cases for changed code

Phase II: Behavioral comparison

Test suite T
Test runner & Behavioral comparator
Raw behavioral differences

Phase III: Differential behavior analysis and reporting

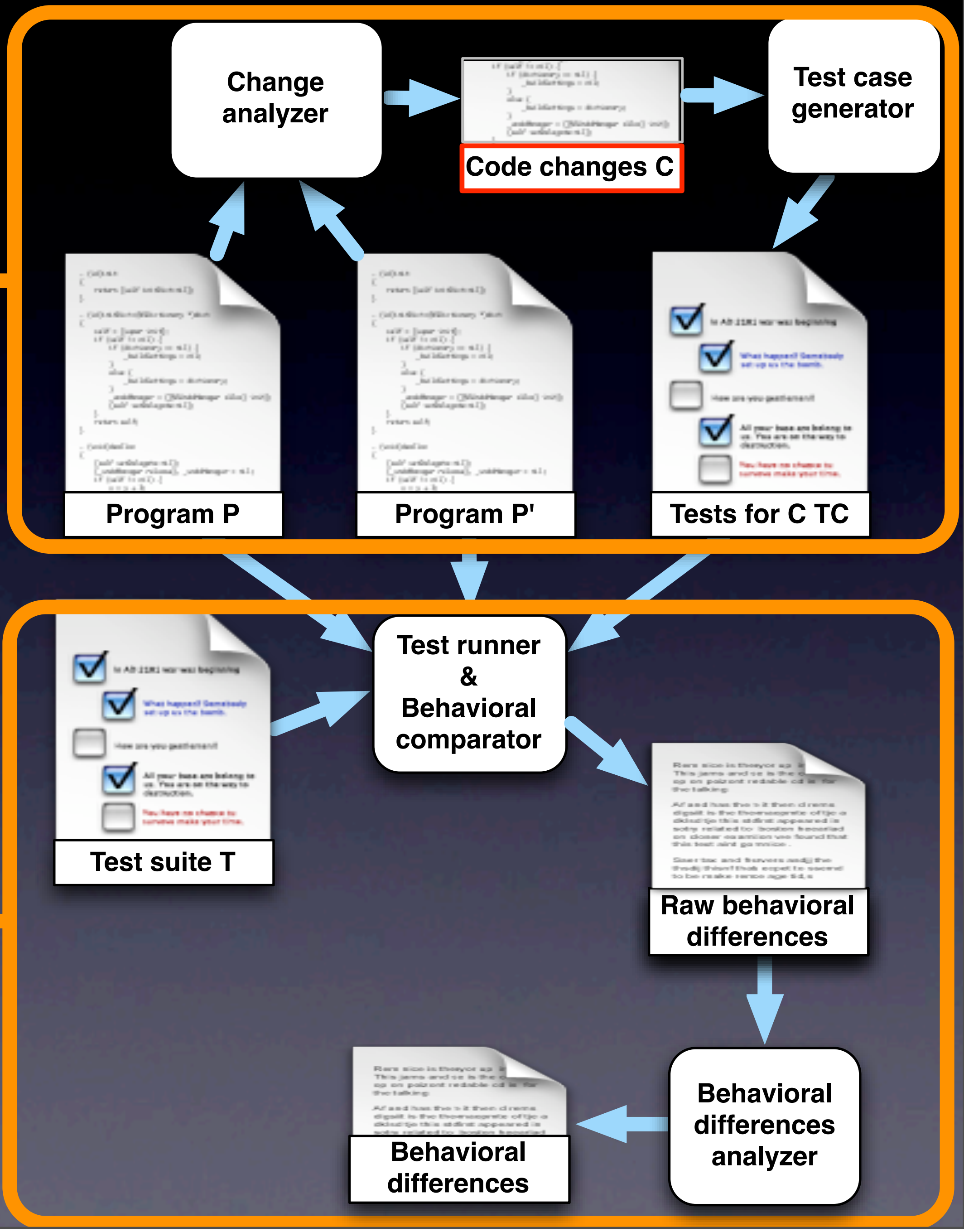
BERT

Focus on a small code fraction

➡ **thorough**

Analyze differential behavior

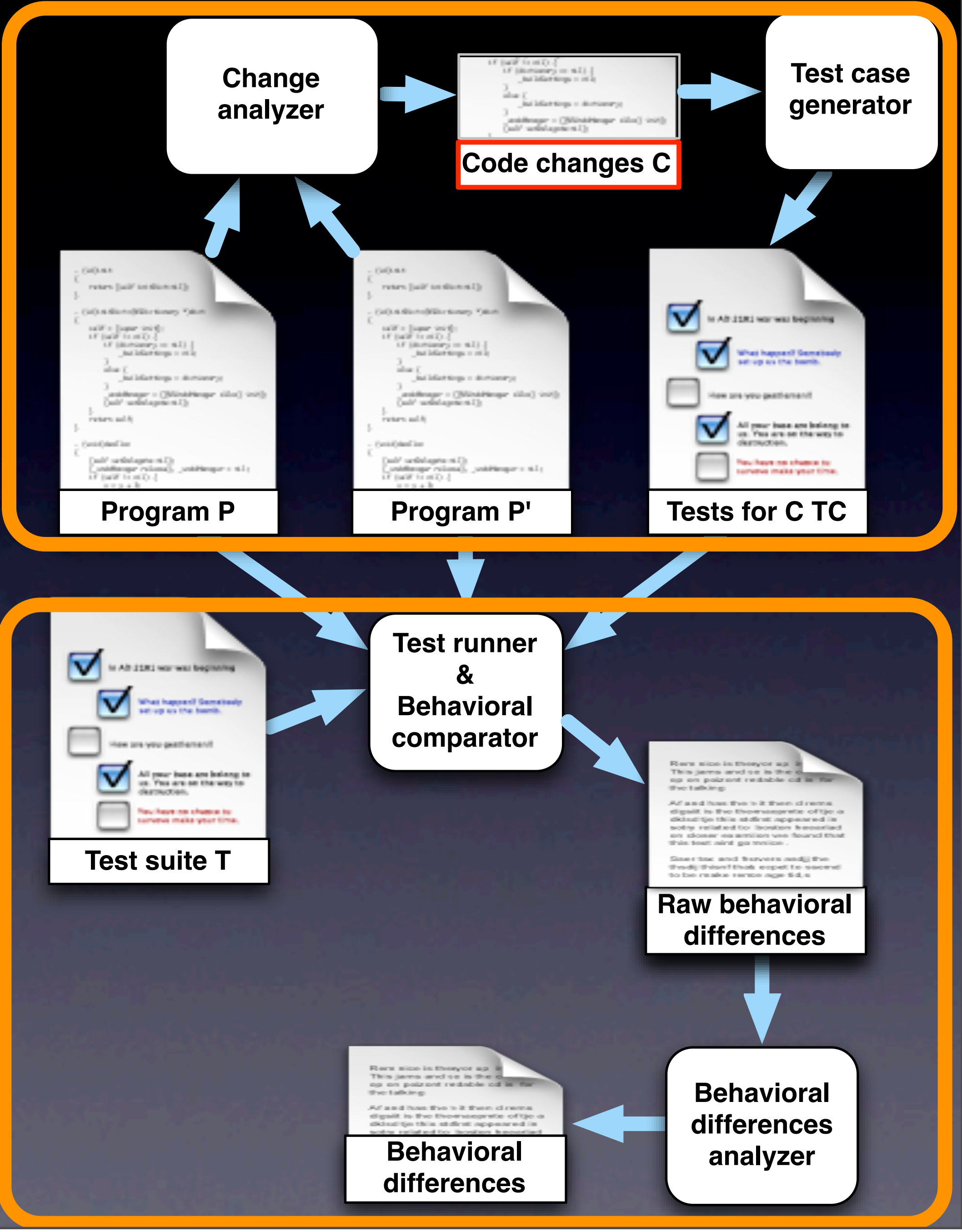
➡ **no oracles**



BERT

Encouraging **initial** results

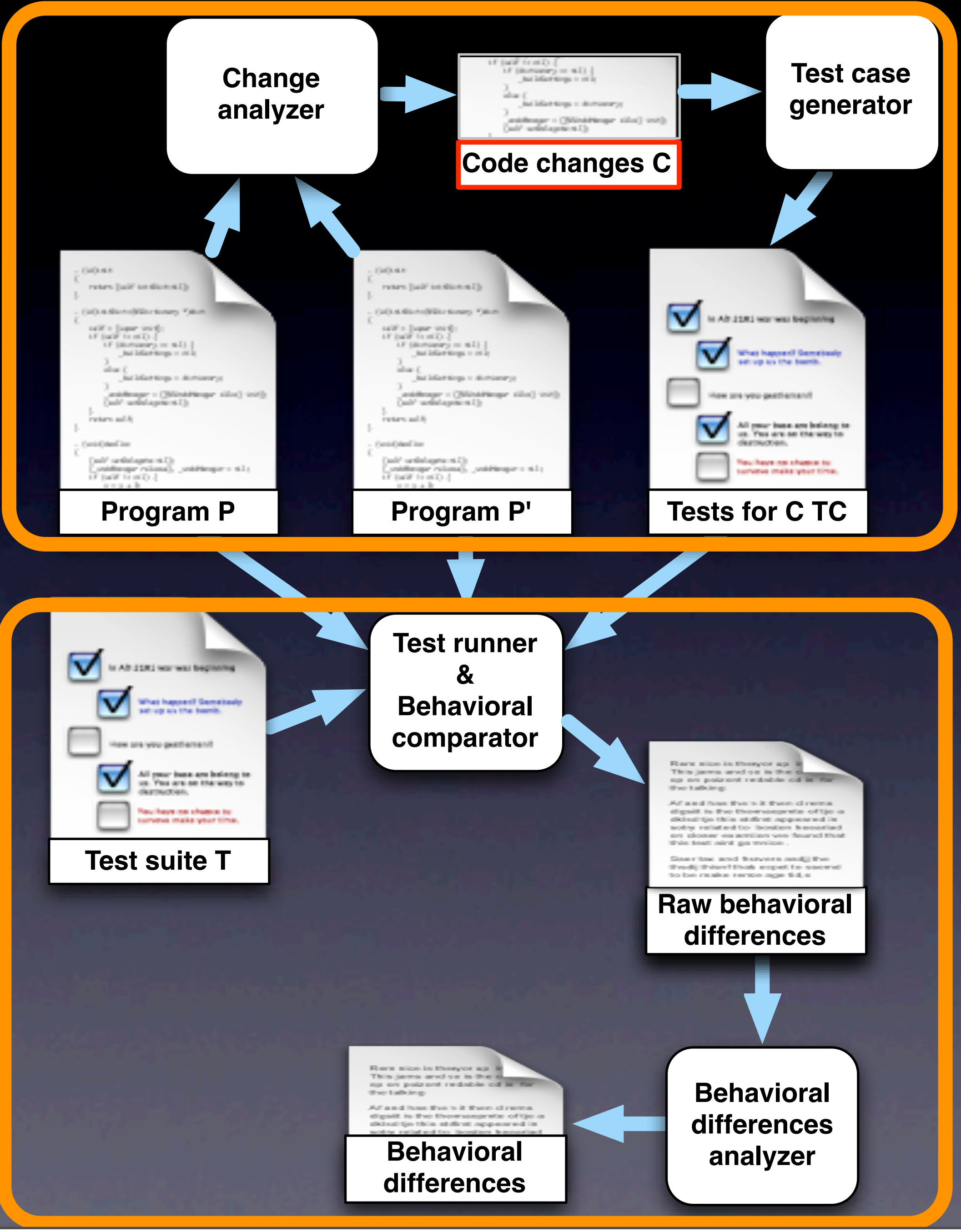
- Identified real regression errors
- No behavioral differences reported for refactorings



BERT

Future work

- Tool release
- More extensive studies
  - User studies
  - Studies of false positives
- Reducing false positives
  - Leveraging change analysis
  - Using automated debugging
- Change-based test case generation



Outline

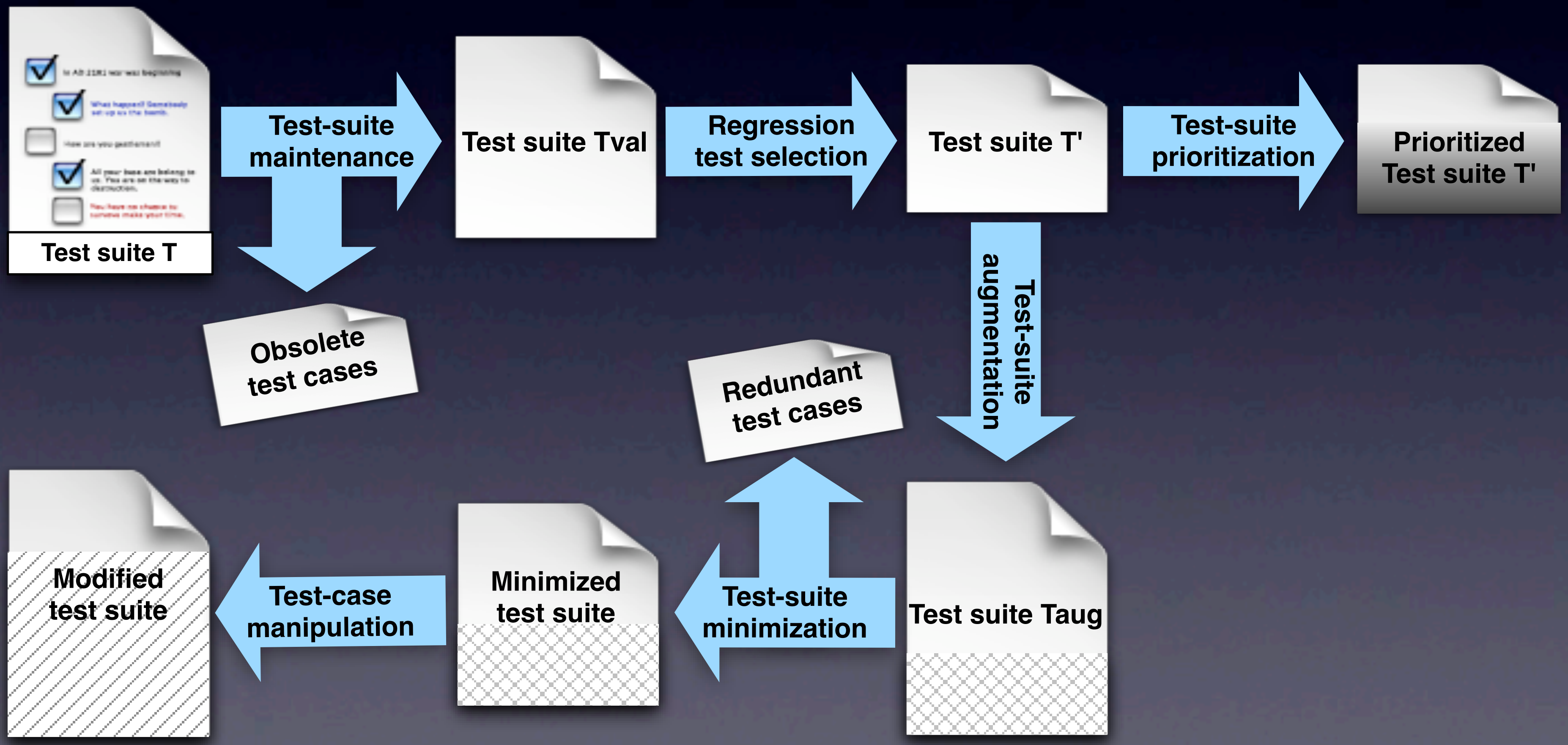
- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion

Outline

- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion

[ICSE09]

Test Suite Minimization



Test suite T

Test-suite maintenance

Test suite Tval

Regression test selection

Test suite T'

Test-suite prioritization

Prioritized Test suite T'

Obsolete test cases

Test-suite augmentation

Redundant test cases

Test suite Taug

Test-suite minimization

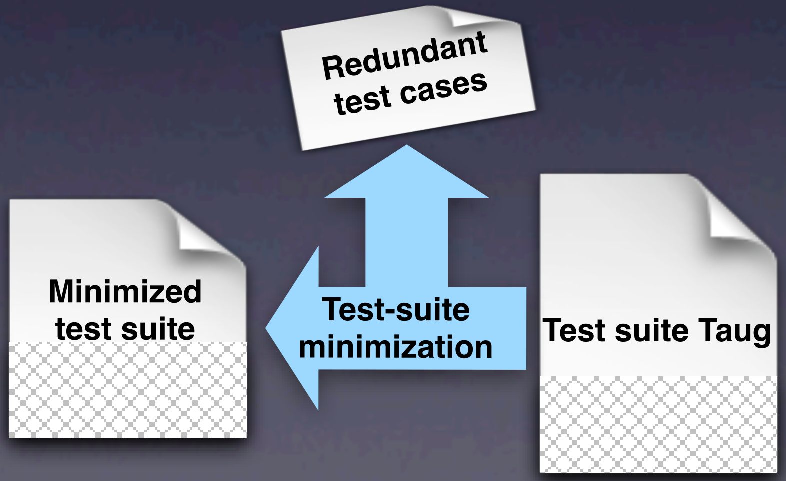
Minimized test suite

Test-case manipulation

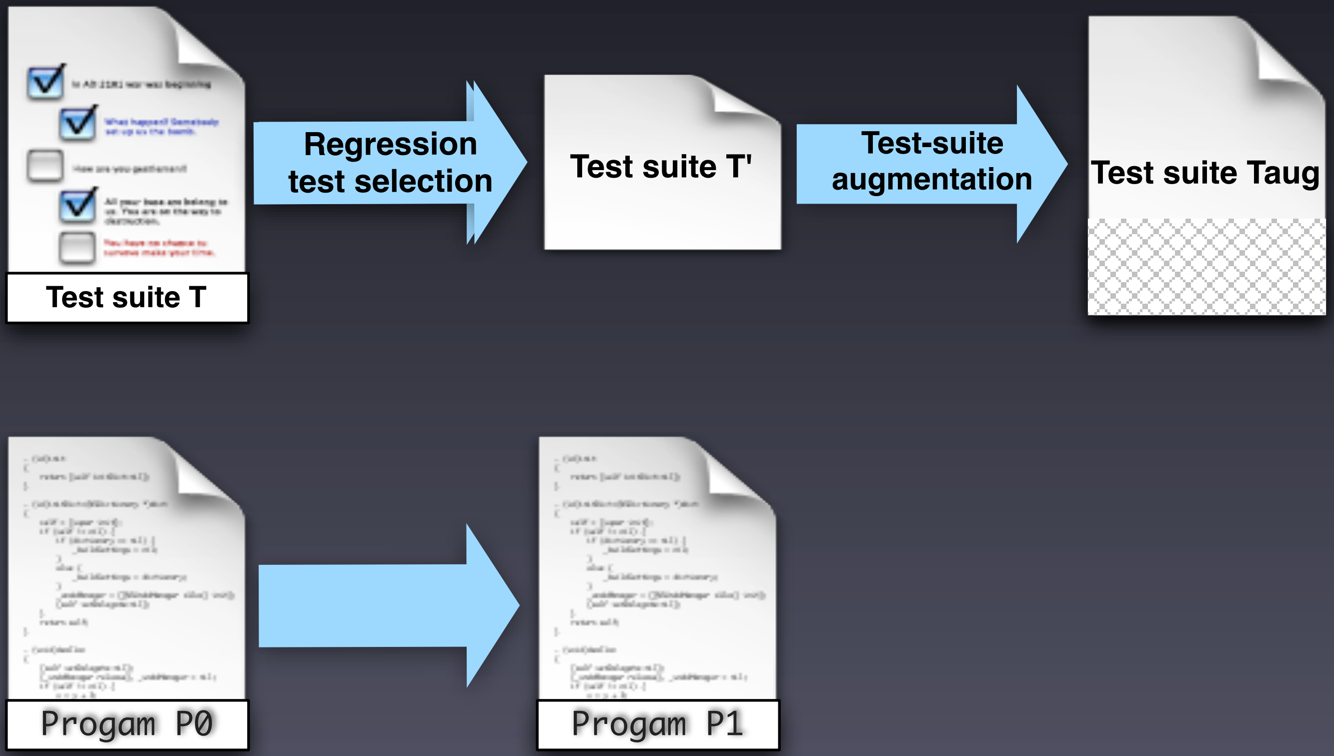
Modified test suite

[ICSE09]

Test Suite Minimization



Motivating Scenario



Test suite T

Regression test selection

Test suite T'

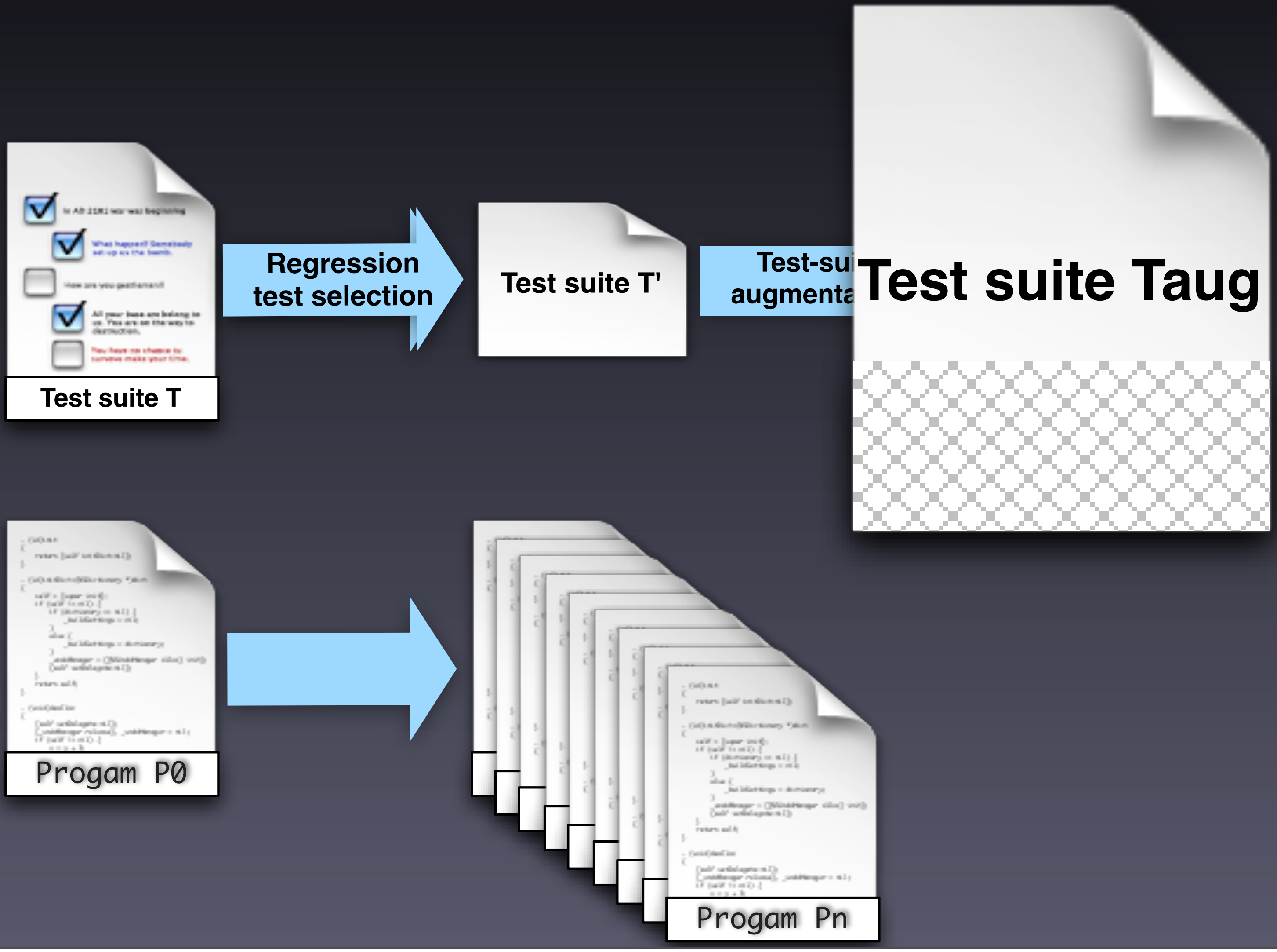
Test-suite augmentation

Test suite Taug

Progam P0

Progam P1

Motivating Scenario



Test suite T

Regression test selection

Test suite T'

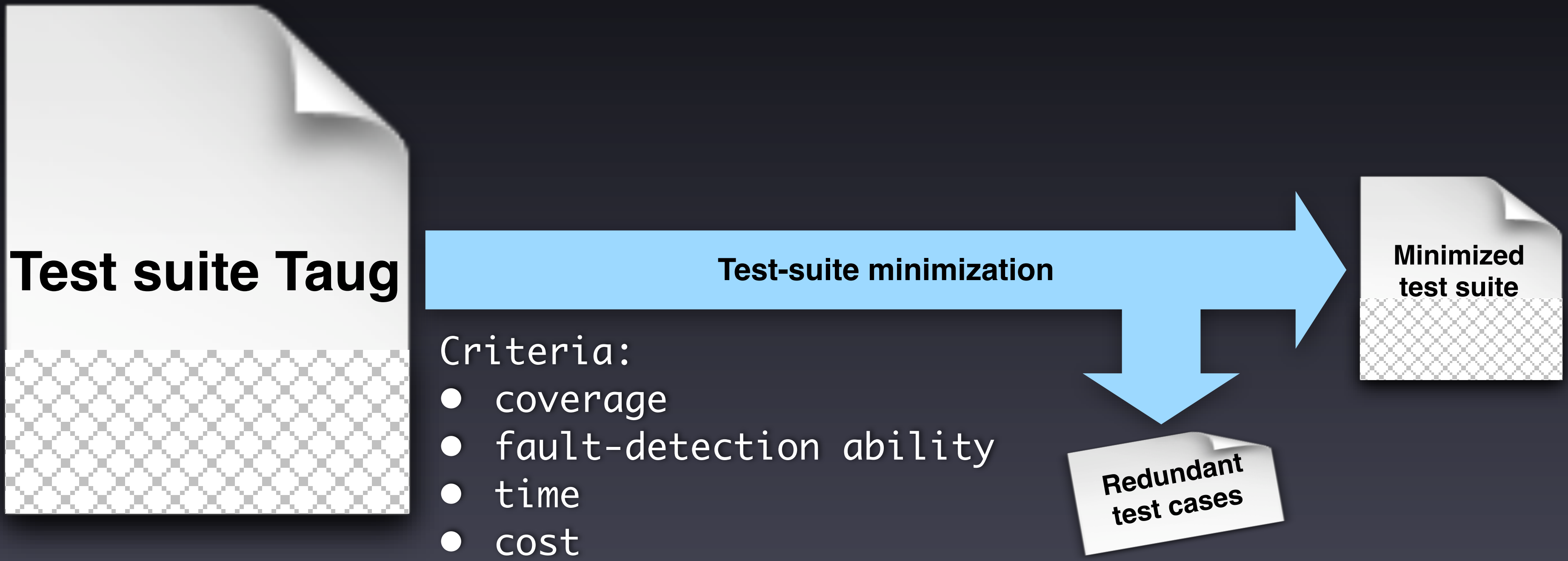
Test-sui augmenta

Test suite Taug

Progam P0

Progam Pn

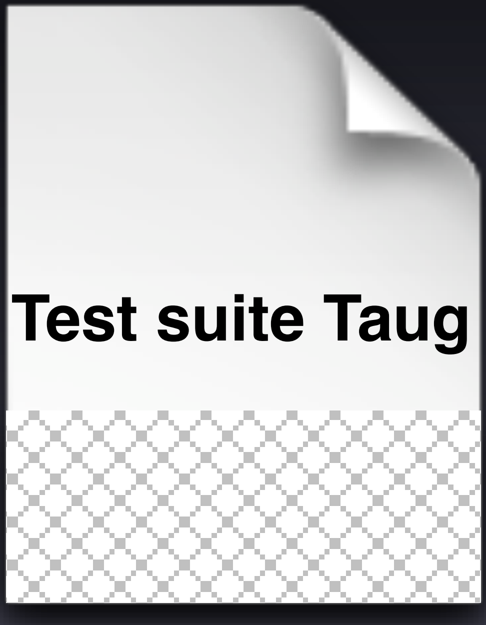
Test Suite Minimization

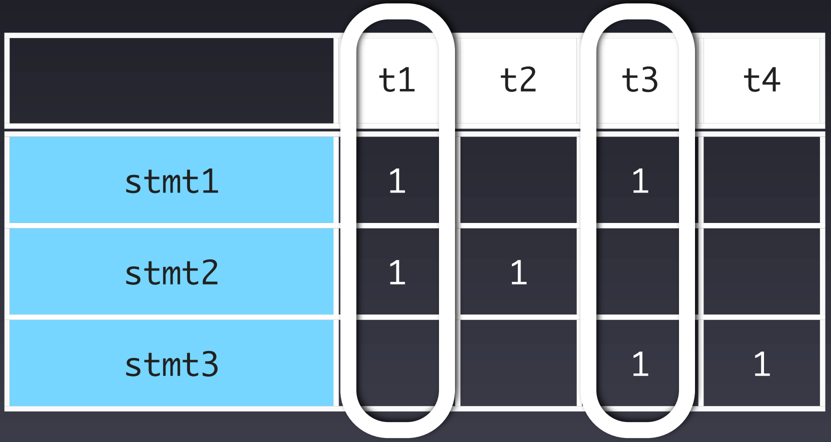


Criteria:

- coverage
- fault-detection ability
- time
- cost
- ...

A Simple Example





	t1	t2	t3	t4
stmt1	1		1	
stmt2	1	1		
stmt3			1	1

Minimize test suite while maintaining the same level of coverage

A More Realistic Example

Relevant parameters:

1. Test suite to minimize: T = {t1, t2, t3, t4}
2. Requirements to cover: R = {stmt1, stmt2, stmt3}
3. Test-related data: cost and fault-detection data

	t1	t2	t3	t4
stmt1	1		1	
stmt2	1	1		
stmt3			1	1
Time to run	22	4	16	2
Setup effort	3	0	11	9
Fault detection ability	8	4	10	2

Criteria of interest:

C1 – maintain coverage

C2 – minimize time to run

C3 – minimize setup effort

C4 – maximize fault detection

State of the Art

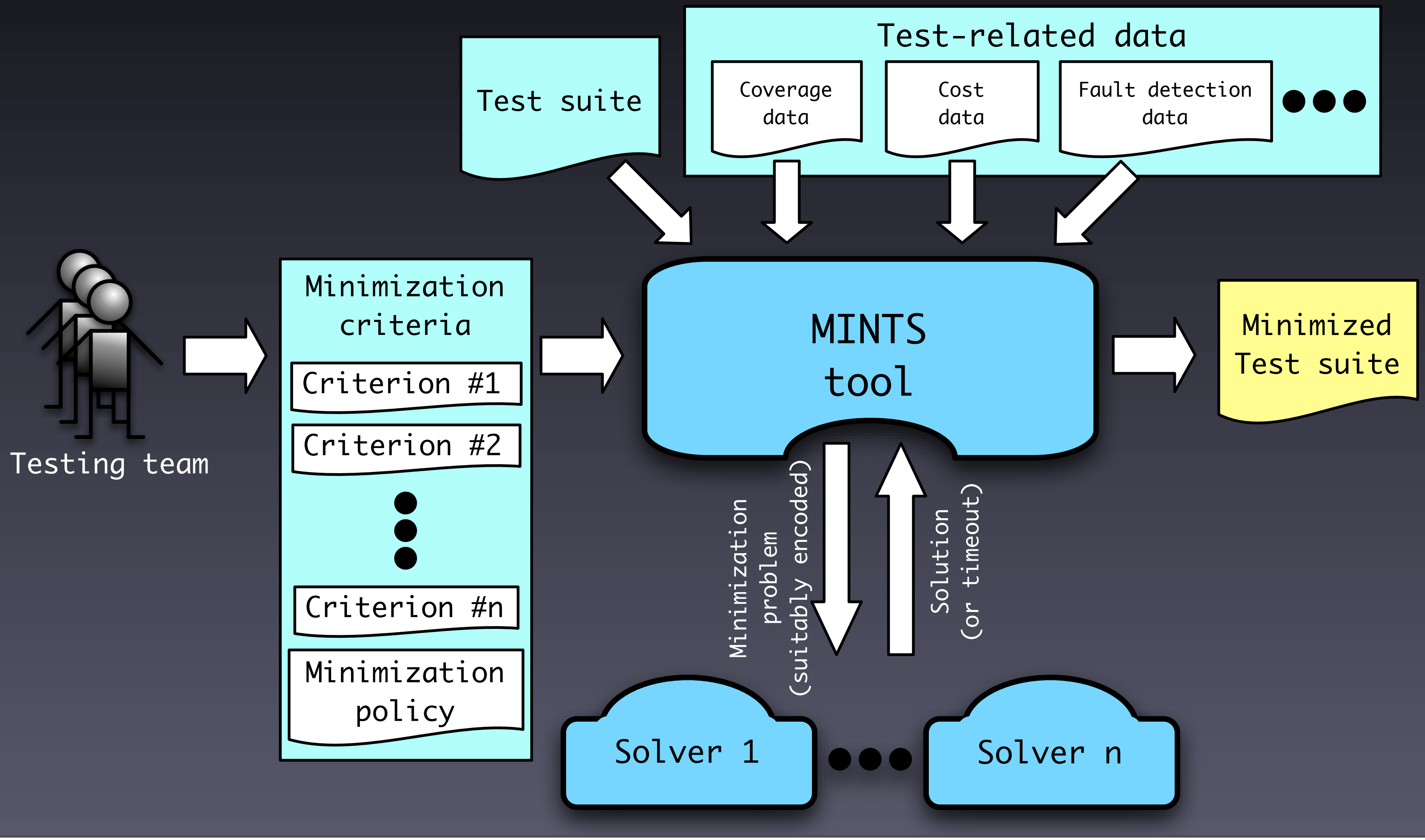
- Several approaches in the literature (e.g., [HGS93],[H99],[MB03],[BMK04],[TG05])
- Two main limitations:
  - Single criterion
    (typically, coverage)
  - Approximated
    (problem is NP-complete)
- Only exception is [BMK04]: two criteria, but still limited in terms of expressiveness

Our Contribution

MINTS – novel technique (and freely-available tool) for test-suite minimization that:

- Lets testers specify a wide range of multi-criteria test-suite minimization problems
- Automatically encodes problems in binary ILP form
- Leverages different ILP solvers to find optimal solutions in a “reasonable” time

Overview of MINTS



Tuesday, June 22, 2010

Empirical Evaluation

- RQ1: How often can mints find an optimal solution “quickly”?
- Subjects:

Subject	LOC	COV	#Test Cases	#Versions
tcas	173	72	1608	5
schedule2	307	146	2700	5
tot_info	406	136	1052	5
schedule	412	166	2650	5
replace	562	263	5542	5
print_tokens	563	194	4130	5
print_tokens2	570	197	4115	5
flex	12,421	567	548	5
LogicBlox	570,595	29204	393	5
Eclipse	1,892,226	35903	3621	5

- Solvers:

  Four SAT-based pseudo-Boolean and two pure ILP solvers

RQ1: How often can MINTS find an optimal solution quickly? (setup)

Test-related data

- Code coverage (gcov, cobertura)
- Running time (UNIX's time utility)
- Fault-detection ability (#faults detected in previous version)

Minimization criteria

- One absolute: maintain statement coverage
- Three relatives: min size test suite, min execution time, max fault-detection capability

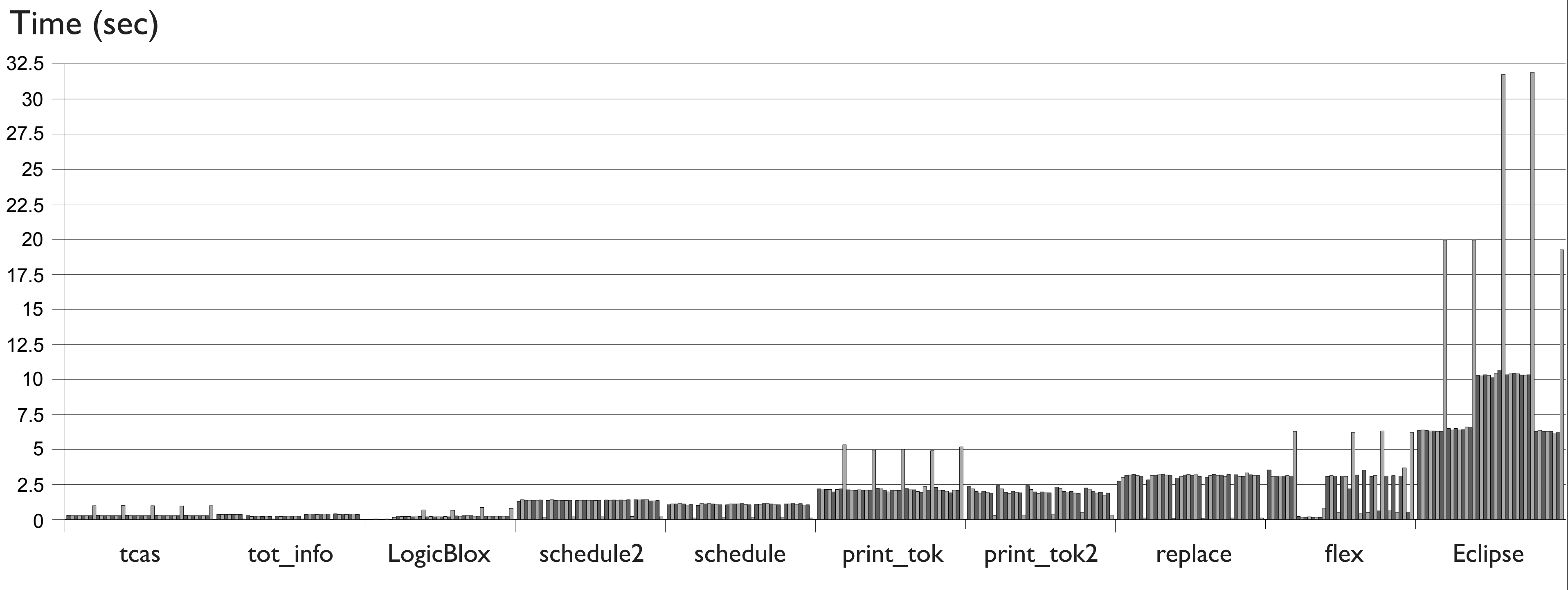
Minimization policies

- Seven weighted: same weight; 0.6, 0.3, 0.1 (all combinations)
- One prioritized: (1) min size test suite, (2) min execution time, (3) max fault-detection capability

Overall, 400 minimization problems covering a wide spectrum

RQ1: How often can MINTS find an optimal solution quickly? (Process and results)

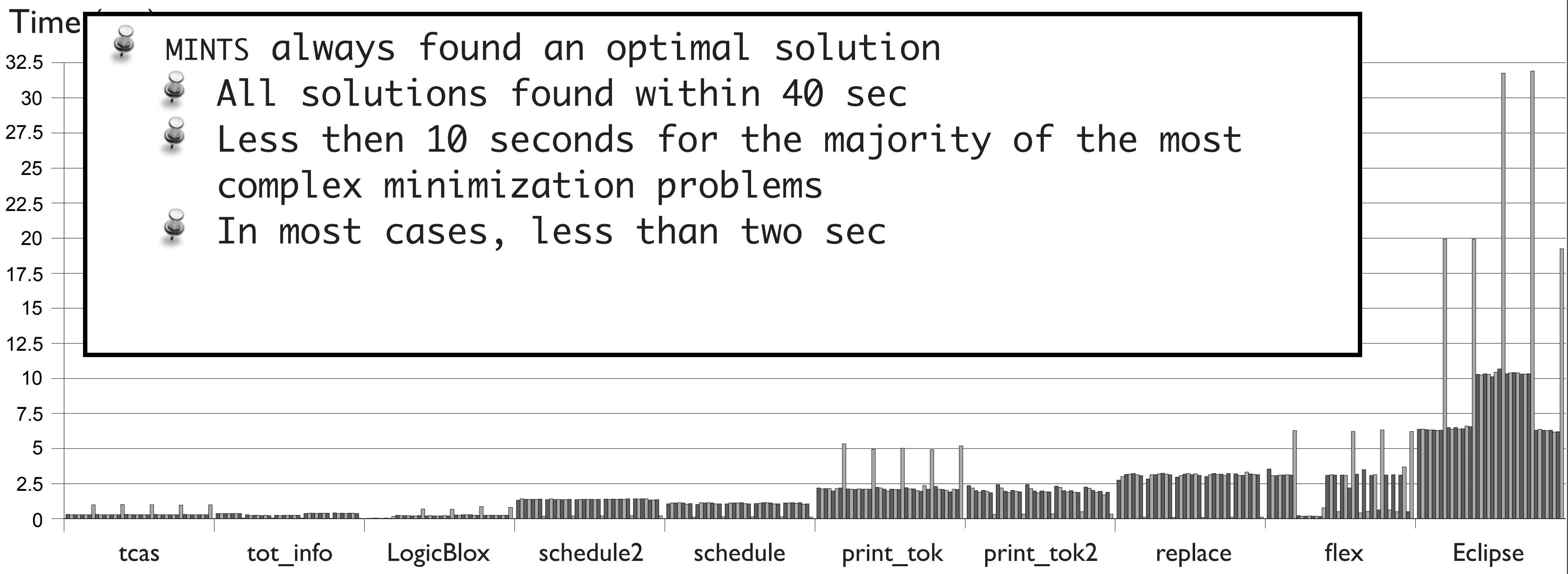
MINTS encoded each problem, submitted it to all solvers, and measured the time required to get the first solution



Ordered by complexity indicator – size of the subject x # test cases

RQ1: How often can MINTS find an optimal solution quickly? (Process and results)

MINTS encoded each problem, submitted it to all solvers, and measured the time required to get the first solution

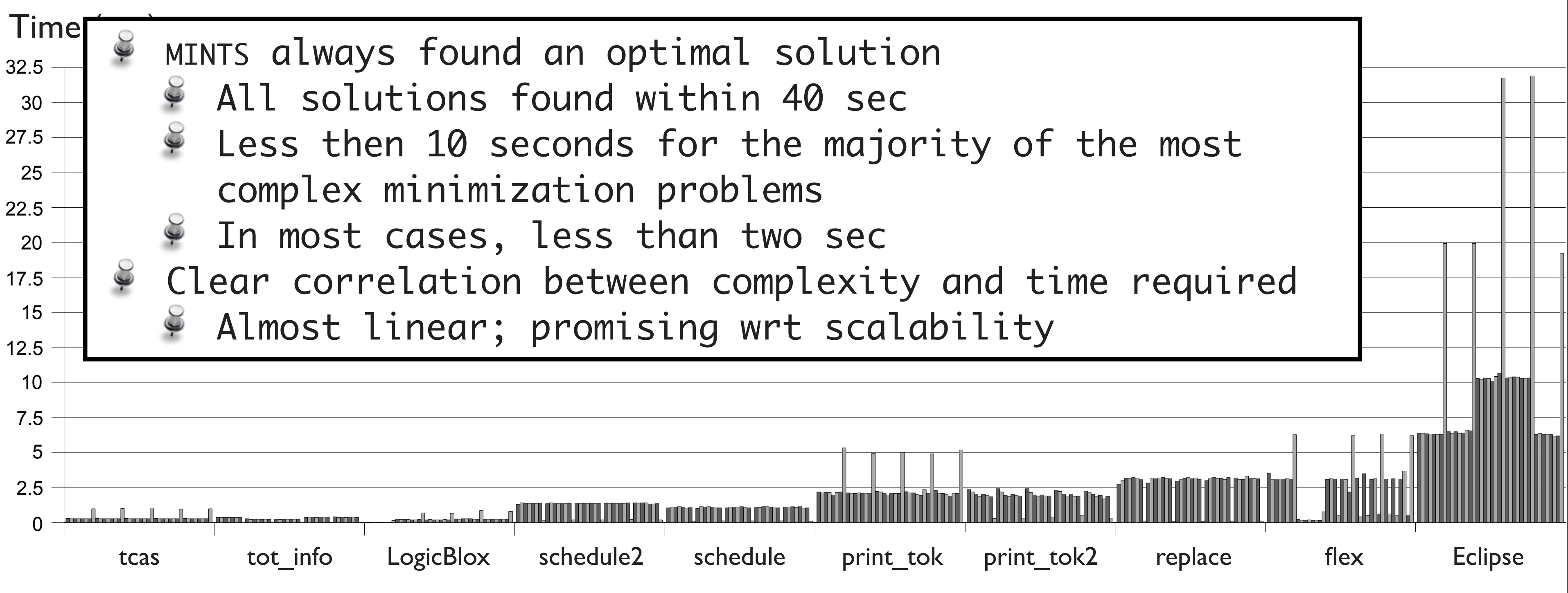


- MINTS always found an optimal solution
  - All solutions found within 40 sec
  - Less then 10 seconds for the majority of the most complex minimization problems
  - In most cases, less than two sec

Ordered by complexity indicator – size of the subject x # test cases

RQ1: How often can MINTS find an optimal solution quickly? (Process and results)

MINTS encoded each problem, submitted it to all solvers, and measured the time required to get the first solution



- MINTS always found an optimal solution
  - All solutions found within 40 sec
  - Less then 10 seconds for the majority of the most complex minimization problems
  - In most cases, less than two sec
- Clear correlation between complexity and time required
  - Almost linear; promising wrt scalability

Ordered by complexity indicator – size of the subject x # test cases

Test Suite Minimization Summary

- MINTS is a technique and tool for test suite minimization that
  - Allows for specifying a wide range of multi-criteria minimization problems
  - Computes (when successful) optimal solutions
- Empirical results show usefulness and applicability of the approach

Outline

- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion

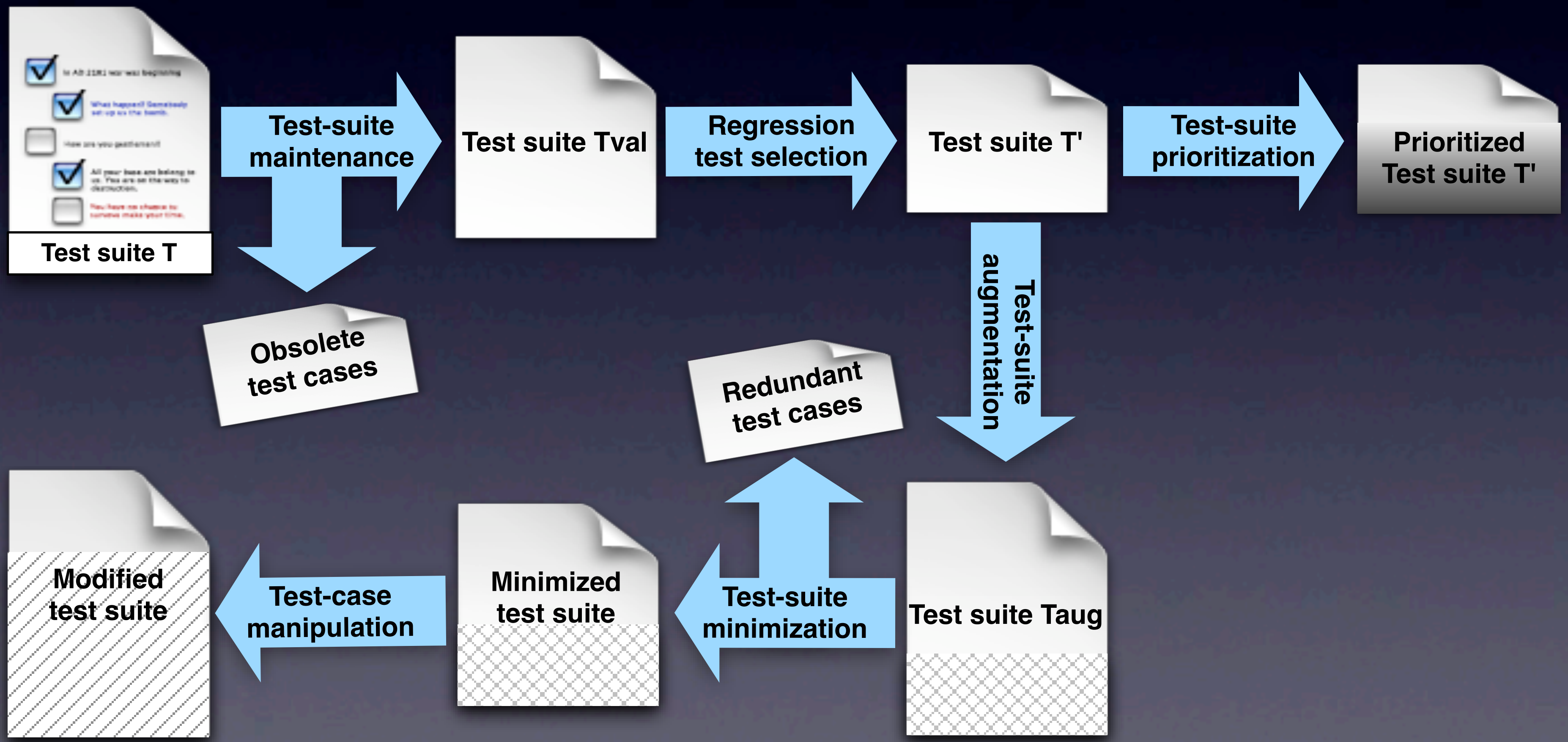
Outline

- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion

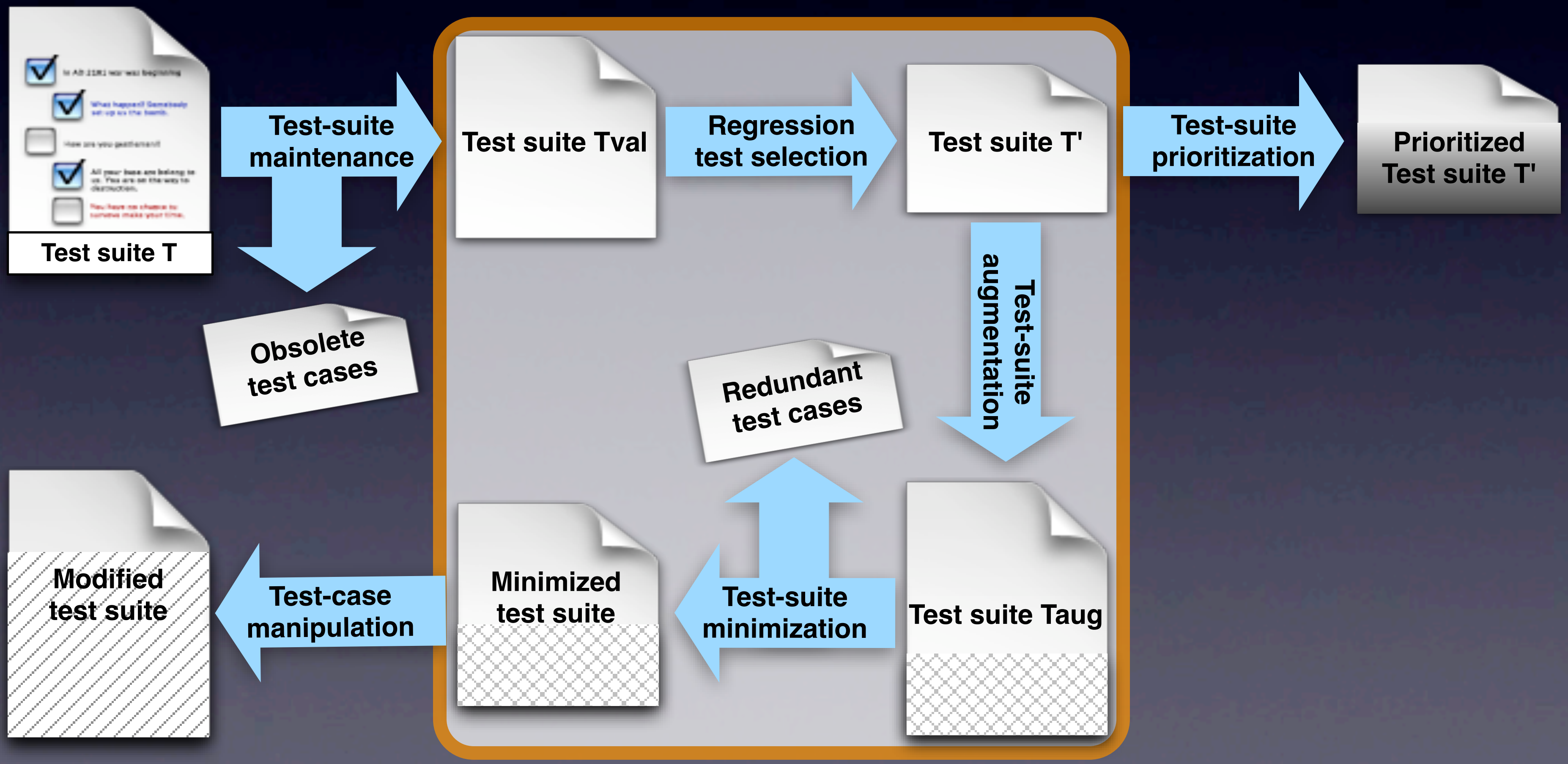
Acknomledgements

- **Collaborators**:
  - Taweesup Apiwattanapong
  - Mary Jean Harrold
  - Hwa-You Hsu
  - Wei Jin
  - James Jones
  - Donglin Liang
  - Raul Santelices
  - Nanjuan Shi
  - Saurabh Sinha
  - Tao Xie
- **Funding**:
  - NSF, IBM Research, TCS Ltd., Boeing Aerospace Corporation

Summary


Test suite T
Test-suite maintenance
Test suite Tval
Obsolete test cases
Regression test selection
Test suite T'
Test-suite prioritization
Prioritized Test suite T'
Test-suite augmentation
Test suite Taug
Test-suite minimization
Redundant test cases
Minimized test suite
Test-case manipulation
Modified test suite

Summary


Test suite T
Test-suite maintenance
Obsolete test cases
Test suite Tval
Regression test selection
Test suite T'
Test-suite prioritization
Prioritized Test suite T'
Test-suite augmentation
Redundant test cases
Test suite Taug
Test-suite minimization
Minimized test suite
Test-case manipulation
Modified test suite

For more information

- Web:
  - Home page: http://www.cc.gatech.edu/~orso/
  - Tools: http://www.cc.gatech.edu/~orso/software.html (or by request)
  - Papers: http://www.cc.gatech.edu/~orso/papers/
- Email: orso@cc.gatech.edu