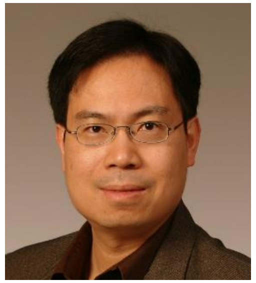
香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Coverage-Driven Test Code Generation for Concurrent Classes

Valerio Terragni



Shing-Chi Cheung

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
{vterragni, scc}@cse.ust.hk

20 May 2016

Automated Test Code Generation for Concurrent Classes

Input

Class Under Test (CUT)

```
public class CUT{
 int x= 0;
 public void setX(int n){
    x = n;
 }
public synchronized void inc(){
    $temp = x;
     x = $temp + 1;
 }
}
```

Automated Test Code Generation for Concurrent Classes

Input
Class Under Test (CUT)

```
public class CUT{
 int x= 0;
 public void setX(int n){
     x = n;
 }
public synchronized void inc(){
     $temp = x;
      x = $temp + 1;
 }
}
```

Output
Test code that expose concurrency bugs (if any)

Automated Test Code Generation for Concurrent Classes

Input

Class Under Test (CUT)

```
public class CUT{
 int x= 0;
 public void setX(int n){
     x = n;
 }
public synchronized void inc(){
    $temp = x;
     x = $temp + 1;
 }
}
```

Output

Test code that expose concurrency bugs (if any)

```
private void runTest() throws Throwable {

  final CUT sout = new CUT();
Thread T1 = new Thread(new Runnable() {
        public void run() { sout.setX(0);}
   });

Thread T2 = new Thread(new Runnable() {
        public void run() { sout.inc(); }
   });
 T1.start();
 T2.start();
```

Automated Test Code Generation for Concurrent Classes

Input
Class Under Test (CUT)

```
public class CUT{
 int x= 0;
 public void setX(int n){
    x = n;
 }
public synchronized void inc(){
    $temp = x;
     x = $temp + 1;
 }
}
```

Output
Test code that expose concurrency bugs (if any)

Shared Object Under Test

```
private void runTest() throws Throwable {

   final CUT sout = new CUT();
Thread T1 = new Thread(new Runnable() {
       public void run() { sout.setX(0); }
   });

Thread T2 = new Thread(new Runnable() {
       public void run() { sout.inc(); }
   });
 T1.start();
 T2.start();
```

Automated Test Code Generation for Concurrent Classes

Input

Class Under Test (CUT)

```
public class CUT{
 int x= 0;
 public void setX(int n){
    x = n;
 }
public synchronized void inc(){
    $temp = x;
     x = $temp + 1;
 }
}
```

Output

Test code that expose concurrency bugs (if any)

```
private void runTest() throws Throwable {

   final CUT sout = new CUT();
Thread T1 = new Thread(new Runnable() {
        public void run() { sout.setX(0); }
    });

Thread T2 = new Thread(new Runnable() {
        public void run()  { sout.inc(); }
    });
  T1.start();
  T2.start();
```

Shared Object Under Test

method call sequences

Automated Test Code Generation for Concurrent Classes

Input

Class Under Test (CUT)

```
public class CUT{
 int x= 0;
 public void setX(int n){
    x = n;
 }
public synchronized void inc(){
    $temp = x;
     x = $temp + 1;
 }
}
```

Output

Test code that expose concurrency bugs (if any)

```
final CUT sout = new CUT();

T1                          T2
sout.setX(0);       sout.inc();
```

Automated Test Code Generation for Concurrent Classes

Input

Class Under Test (CUT)

```
public class CUT{
 int x= 0;
 public void setX(int n){
    x = n;
 }
public synchronized void inc(){
    $temp = x;
     x = $temp + 1;
 }
}
```

Output

Test code that expose concurrency bugs (if any)

```
final CUT sout = new CUT();
```

T1 **T2**

```
sout.setX(0);      sout.inc();
```

```
                $temp = x;
x = n;
                x = $temp + 1;
```

Automated Test Code Generation for Concurrent Classes

Input

Class Under Test (CUT)

```
public class CUT{
 int x= 0;
 public void setX(int n){
    x = n;
 }
public synchronized void inc(){
    $temp = x;
     x = $temp + 1;
 }
}
```

Output

Test code that expose concurrency bugs (if any)

```
final CUT sout = new CUT();

T1                    T2
sout.setX(0);     sout.inc();

                  $temp = x;
x = n;
                  x = $temp + 1;
```

buggy interleaving

serializability violation

Coverage-driven Test Code Generation

Random generation [Pradel et. al. PLDI 2012, Nistor et. al. ICSE 2012]

- **Many** randomly generated tests are needed for effective bug detection
  - Explore the interleaving space of many tests is **too expensive**!

Coverage-driven Test Code Generation

Random generation [Pradel et. al. PLDI 2012, Nistor et. al. ICSE 2012]

- Many randomly generated tests are needed for effective bug detection
  - Explore the interleaving space of many tests is too expensive!

Research Problem

Generate fewer tests that collectively achieve the highest coverage with respect to a given interleaving coverage criterion

Coverage-driven Test Code Generation

Random generation [Pradel et. al. PLDI 2012, Nistor et. al. ICSE 2012]

- Many randomly generated tests are needed for effective bug detection
  - Explore the interleaving space of many tests is too expensive!

Research Problem

Generate fewer tests that collectively achieve the highest coverage with respect to a given interleaving coverage criterion

An example of interleaving coverage criterion are the 11 problematic access patterns violating atomic-set serializability [Vaziri POPL 2006]

T1	T2
	R(x)
W(x)	
	W(x)

W(x) write on shared memory location x

R(x) write on shared memory location x

Coverage Requirements

- **Interleavings** that match a problematic access pattern
  - Usually computed with respect to **a given test code**

Coverage Requirements

- Interleavings that match a problematic access pattern
  - Usually computed with respect to a given test code
- Coverage driven test code generation requires to compute coverage requirements for all possible test codes

Coverage Requirements

- Interleavings that match a problematic access pattern
  - Usually computed with respect to a given test code
- Coverage driven test code generation requires to compute coverage requirements for all possible test codes

problematic access pattern

T1	T2
	R(x)
W(x)	
	W(x)

Coverage Requirements

- Interleavings that match a problematic access pattern
  - Usually computed with respect to a given test code
- Coverage driven test code generation requires to compute coverage requirements for **all possible test codes**

problematic access pattern

T1	T2
	R(x)
W(x)	
	W(x)

Class Under Test

```
public class CUT{
 int x= 0;
 public void setX(int n){
   x = n;
}
public synchronized void inc(){
     $temp = x;
     x = $temp + 1;
} }
```

Coverage Requirements

- Interleavings that match a problematic access pattern
  - Usually computed with respect to a given test code
- Coverage driven test code generation requires to compute coverage requirements for **all possible test codes**

problematic access pattern

T1	**T2**
	R(x)
W(x)	
	W(x)

+

Class Under Test

```
public class CUT{
 int x= 0;
 public void setX(int n){
   x = n;
}
public synchronized void inc(){
     $temp = x;
     x = $temp + 1;
} }
```

=

coverage requirements

Challenge

Compute the executable domain of interleaving coverage criteria is machine undecidable [Ramalingam, TOPLAS 2000]

- It requires Context-sensitive and synchronization-sensitive analysis

Challenge

Compute the executable domain of interleaving coverage criteria is machine undecidable [Ramalingam, TOPLAS 2000]

- It requires Context-sensitive and synchronization-sensitive analysis

Challenge

Compute the executable domain of interleaving coverage criteria is machine undecidable [Ramalingam, TOPLAS 2000]

It requires Context-sensitive and synchronization-sensitive analysis

ConSuite [Steenbuck et al., ICST 2013]

EVOSUITE

- Compute context-insensitive and synchronization-insensitive coverage requirements statically

ConSuite [Steenbuck et al., ICST 2013]

Step 1 Collect context-insensitive coverage requirements **(statically)**

problematic access pattern

T1 **T2**

R(x)

W(x)

W(x)

ConSuite [Steenbuck et al., ICST 2013]

Step 1 Collect **context-insensitive coverage requirements (statically)**

problematic access pattern

T1	T2
	R(x)
W(x)	
	W(x)

\+

Class Under Test

```
public class CUT{
 int x= 0, z = 0;
 public void m1(int n){
   x = n;
}
private void m4(int v1){
     $temp = x;
     x = $temp + v1;
}
}
```

ConSuite [Steenbuck et al., ICST 2013]

Step 1 Collect **context-insensitive coverage requirements (statically)**

problematic access pattern

T1 **T2**

R(x)

W(x) **W(x)**

Class Under Test

```
public class CUT{
 int x= 0, z = 0;
 public void m1(int n){
   x = n;
 }
private void m4(int v1){
    $temp = x;
    x = $temp + v1;
 }
}
```

context-insensitive
coverage requirement

x = n;

static match of bytecode instructions

ConSuite [Steenbuck et al., ICST 2013]

Step 1 Collect **context-insensitive coverage requirements (statically)**

problematic access pattern

T1	T2
	R(x)
W(x)	
	W(x)

\+

Class Under Test

```
public class CUT{
 int x= 0, z = 0;
 public void m1(int n){
   x = n;
 }
private void m4(int v1){
   $temp = x;
   x = $temp + v1;
 }
}
```

=

context-insensitive
coverage requirement

x = n;

$temp = x;

static match of bytecode instructions

ConSuite [Steenbuck et al., ICST 2013]

Step 1 Collect **context-insensitive coverage requirements (statically)**

problematic access pattern

T1 **T2**

R(x)

W(x)

W(x)

Class Under Test

```
public class CUT{
 int x= 0, z = 0;
 public void m1(int n){
   x = n;
 }
private void m4(int v1){
   $temp = x;
   x = $temp + v1;
 }
}
```

context-insensitive coverage requirement

x = n;

$temp = x;

x = $temp + v1;

static match of bytecode instructions

ConSuite [Steenbuck et al., ICST 2013]

Step 1 Collect **context-insensitive coverage requirements (statically)**

problematic access pattern

T1 **T2**

R(x)

W(x)

W(x)

+

Class Under Test

```
public class CUT{
 int x= 0, z = 0;
 public void m1(int n){
    x = n;
 }
private void m4(int v1){
    $temp = x;
    x = $temp + v1;
 }
}
```

=

context-insensitive
coverage requirement

x = n;

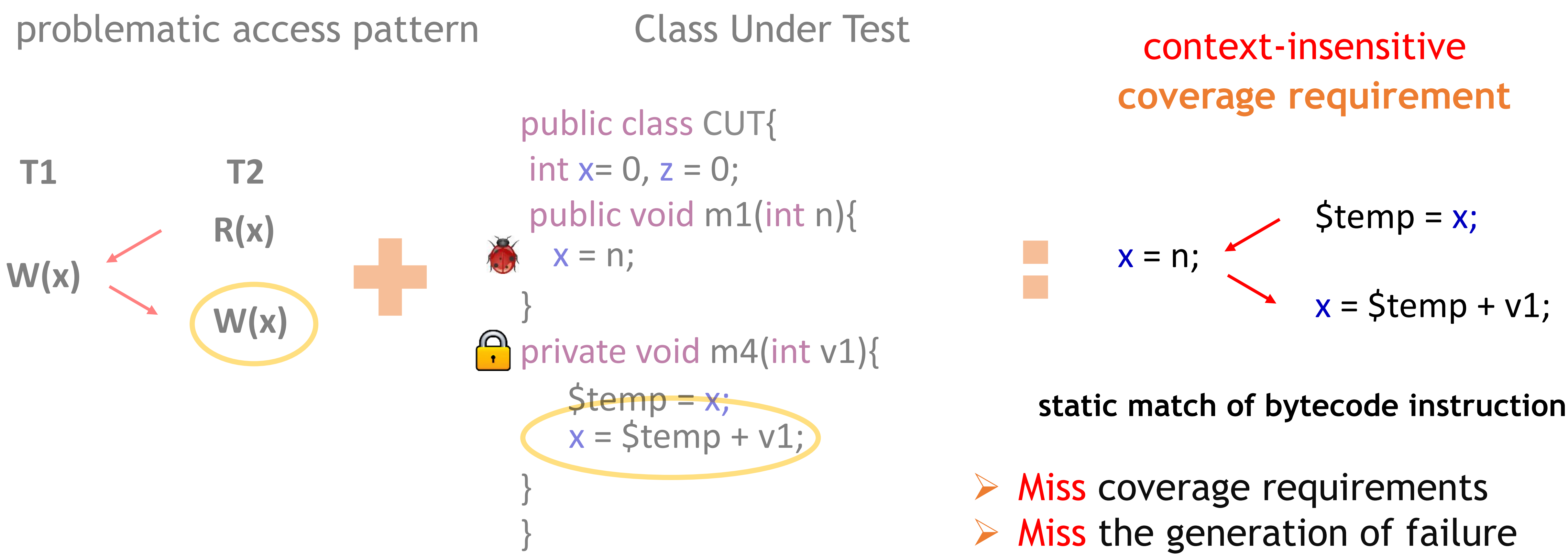
$temp = x;

x = $temp + v1;

static match of bytecode instructions

ConSuite [Steenbuck et al., ICST 2013]

Step 1 Collect **context-insensitive coverage requirements (statically)**



static match of bytecode instructions

- Miss coverage requirements
- Miss the generation of failure inducing test codes

Step 1 Collect **context-insensitive coverage requirements (statically)**

Step 2 **Coverage-driven** test code generation

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int n){
     x = n;
}

public void m2(){
      z = 1;
}

public void m3(int v1){
    if(z ==0){
     synchronized (this){
          m4(v1);
       }
    } else
          m4(v1);
 }

private void m4(int v1){
     $temp = x;
     x = $temp + v1;

}
```

- **Context-insensitive** coverage requirement

$temp = x;

x= n

x; = $temp + v1;

Step 1 Collect context-insensitive coverage requirements (statically)

Step 2 Coverage-driven test code generation

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int n){
    x = n;
}

public void m2(){
    z = 1;
}

public void m3(int v1){
    if(z ==0){
     synchronized (this){
        m4(v1);
     }
    } else
        m4(v1);
}

private void m4(int v1){
    $temp = x;
    x = $temp + v1;

}
```

➢ Context-insensitive coverage requirement

```
          $temp = x;
x= n
          x; = $temp + v1;
```

➢ Test code generation

```
final CUT sout = new CUT();
T1
sout.m1(0);
```

Step 1 Collect **context-insensitive coverage requirements (statically)**

Step 2 **Coverage-driven** test code generation

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int n){
    x = n;
}

public void m3(int v1){
   if(z ==0){
   synchronized (this){
         m4(v1);
      }
   } else
         m4(v1);
}
```

```
public void m2(){
     z = 1;
}

private void m4(int v1){
   $temp = x;
   x = $temp + v1;

}
```

- **Context-insensitive** coverage requirement

$temp = x;

x= n

x; = $temp + v1;

- Test code generation

```
final CUT sout = new CUT();
T1                     T2
sout.m1(0);    sout.m3(10);
```

Step 1 Collect context-insensitive coverage requirements (statically)

Step 2 Coverage-driven test code generation

ConSuite [Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int n){
     x = n;
}

public void m2(){
     z = 1;
}

public void m3(int v1){
    if(z ==0){
    synchronized (this){
         m4(v1);
      }
    } else
         m4(v1);
}

private void m4(int v1){
   $temp = x;
   x = $temp + v1;

}
```

- **Context-insensitive** coverage requirement

```
                 $temp = x;
x= n
                 x; = $temp + v1;
```

- Test code generation

```
final CUT sout = new CUT();
T1                    T2
sout.m1(0);   sout.m3(10);

                 $temp = x;
                 x = $temp + v1;
x= n
```

infeasible

Context and synchronization **insensitivity misses** the generation of this failure inducing test code

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int x){
     x = n;
}

public void m3(int v1){
    if(z ==0){
       synchronized (this){
           m4(v1);
       }
    } else
           m4(v1);
 }
```

```
public void m2(){
      z = 1;
}

private void m4(int v1){
    $temp = x;
    x = $temp + v1;

}
```

```
final CUT sout = new CUT();

T1                        T2
sout.m1(0);          sout.m2()
                     sout.m3(10);
```

Context and synchronization **insensitivity** misses the generation of this failure inducing test code

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int x){
     x = n;
}

public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

```
public void m2(){
    z = 1;
}

private void m4(int v1){
    $temp = x;
    x = $temp + v1;

}
```

```
final CUT sout = new CUT();
```

T1

```
sout.m1(0);
```

T2

```
sout.m2()
sout.m3(10);
```

z = 1

Context and synchronization **insensitivity misses** the generation of this failure inducing test code

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int x){
     x = n;
}

public void m3(int v1){
    if(z ==0){
       synchronized (this){
          m4(v1);
        }
    } else
          m4(v1);
 }
```

```
public void m2(){
      z = 1;
}

private void m4(int v1){
    $temp = x;
    x = $temp + v1;

}
```

final CUT sout = new CUT();

T1 **T2**

sout.m1(0);

sout.m2() → **z = 1**

sout.m3(10);

Context and synchronization insensitivity misses the generation of this failure inducing test code

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int x){
     x = n;
}

public void m3(int v1){
   if(z ==0){
       synchronized (this){
          m4(v1);
       }
   } else
          m4(v1);
 }
```

```
public void m2(){
      z = 1;
}

private void m4(int v1){
     $temp = x;
     x = $temp + v1;

}
```

```
final CUT sout = new CUT();
T1                       T2
sout.m1(0);        sout.m2()      → z = 1
                   sout.m3(10);
```

Context and synchronization insensitivity misses the generation of this failure inducing test code

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int x){
     x = n;
}

public void m3(int v1){
    if(z ==0){
       synchronized (this){
          m4(v1);
       }
    } else
          m4(v1);
 }
```

```
public void m2(){
      z = 1;
}

private void m4(int v1){
    $temp = x;
    x = $temp + v1;

}
```

```
final CUT sout = new CUT();
T1                        T2
sout.m1(0);      sout.m2()        z = 1
                sout.m3(10);
```

Context and synchronization insensitivity misses the generation of this failure inducing test code

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int x){
     x = n;
}

public void m2(){
     z = 1;
}

public void m3(int v1){
    if(z ==0){
      synchronized (this){
         m4(v1);
      }
    } else
         m4(v1);
}

private void m4(int v1){
    $temp = x;
    x = $temp + v1;

}
```

```
final CUT sout = new CUT();
T1                      T2
sout.m1(0);        sout.m2()        → z = 1
                   sout.m3(10);
```

Context and synchronization **insensitivity** misses the generation of this failure inducing test code

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int x){
     x = n;
}

public void m3(int v1){
    if(z ==0){
       synchronized (this){
          m4(v1);
       }
    } else
          m4(v1);
 }
```

```
public void m2(){
      z = 1;
}

private void m4(int v1){
   $temp = x;
   x = $temp + v1;

}
```

```
final CUT sout = new CUT();
```

T1 **T2**

sout.m1(0);

sout.m2() → **z = 1**
sout.m3(10);

FEASIBLE interleaving

$temp = x;

x= n

x; = $temp + v1;

Context and synchronization **insensitivity misses** the generation of this failure inducing test code

ConSuite
[Steenbuck et al., ICST 2013]

```
public class CUT{
 int x= 0, z =0;

public synchronized void m1(int x){
     x = n;
}

public void m2(){
        z = 1;
}

public void m3(int v1){
    if(z ==0){
       synchronized (this){
          m4(v1);
       }
    } else
          m4(v1);
 }

private void m4(int v1){
   $temp = x;
   x = $temp + v1;

}
```

```
final CUT sout = new CUT();
T1                     T2
sout.m1(0);        sout.m2()      → z = 1
                   sout.m3(10);
```

FEASIBLE interleaving

```
                 $temp = x;
x= n
                 x; = $temp + v1;
```

Without context and synchronization sensitivity this test is unlikely generated

Our Intuition

Step 1 Collect **context-insensitive** **coverage requirements** **(statically)**

Step 2 **Coverage-driven** test code generation

Our Intuition

~~**Step 1** Collect **context-insensitive coverage requirements** **(statically)**~~

Step 2 **Coverage-driven** test code generation

- Context-sensitive and synchronization-sensitive **information** can be collected precisely and efficiently **during sequential** test code generation

Our Intuition

~~**Step 1** Collect **context-insensitive coverage requirements (statically)**~~

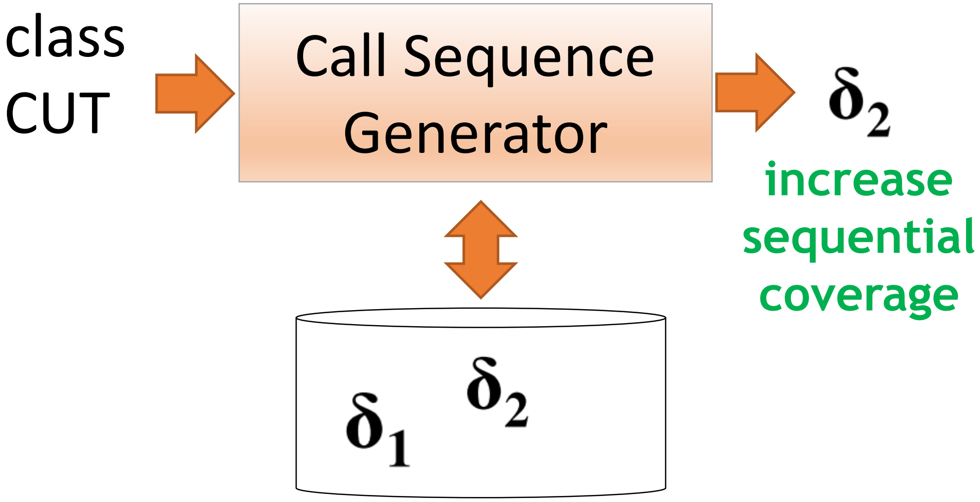
Step 2 Coverage-driven test code generation

- Context-sensitive and synchronization-sensitive **information** can be collected precisely and efficiently **during sequential** test code generation

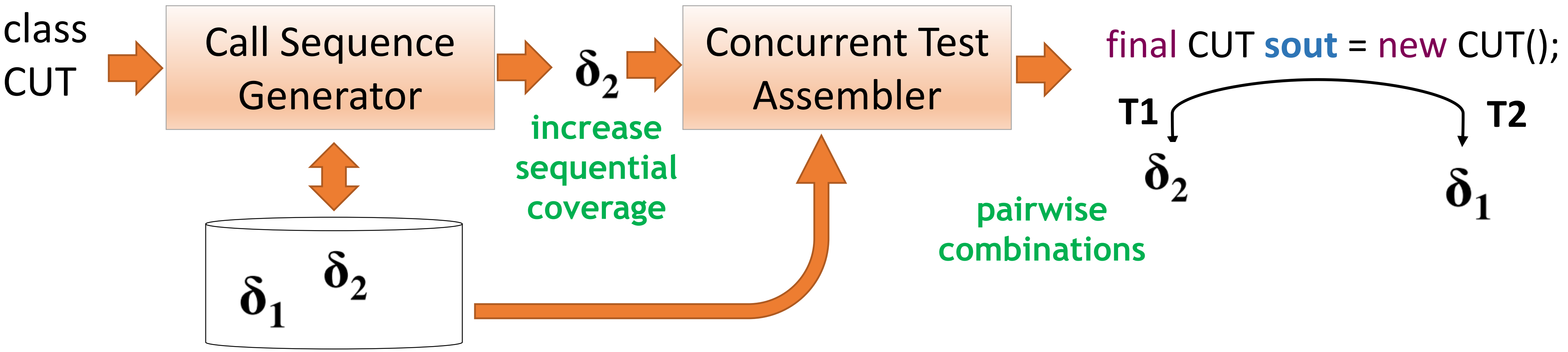
- **Coverage metric (Sequential coverage)**
  - Granularity at outer-most method call
  - Ordered sequence of object’s fields accesses
  - Context and synchronization sensitive
    - lock acquires and releases
    - calling context

AutoConTest

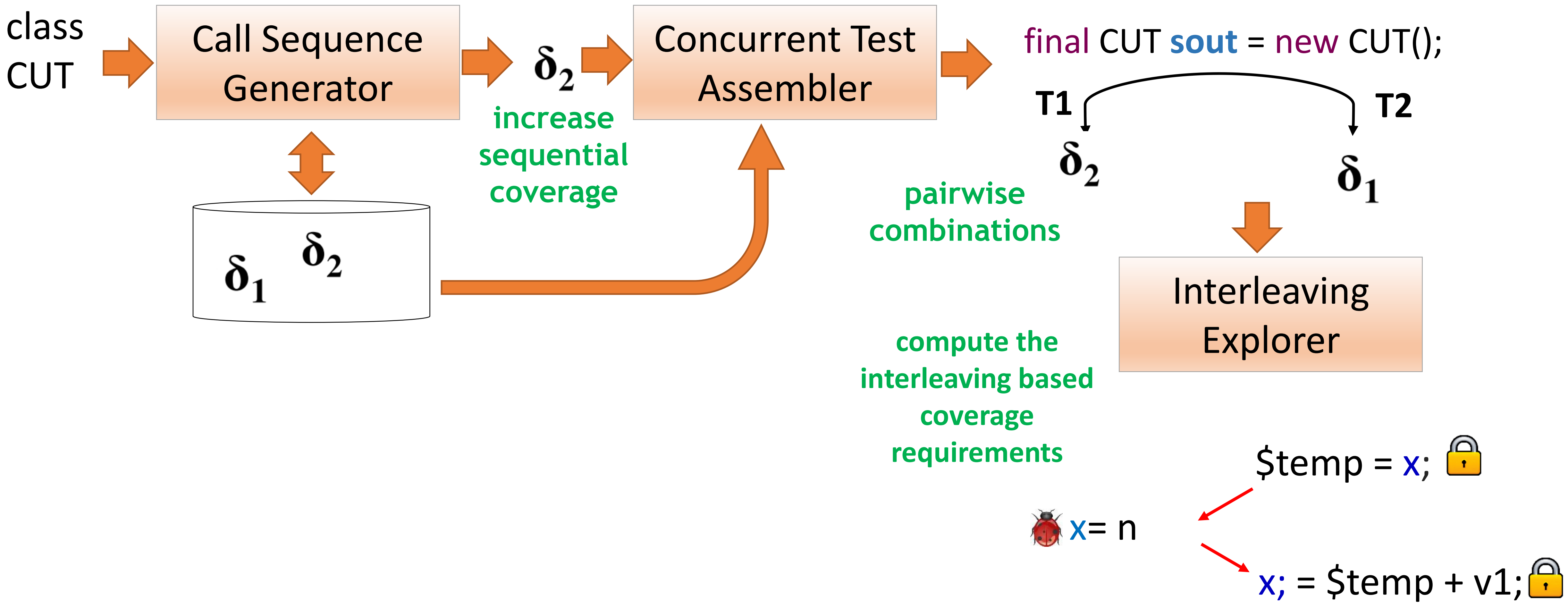
AutoConTest



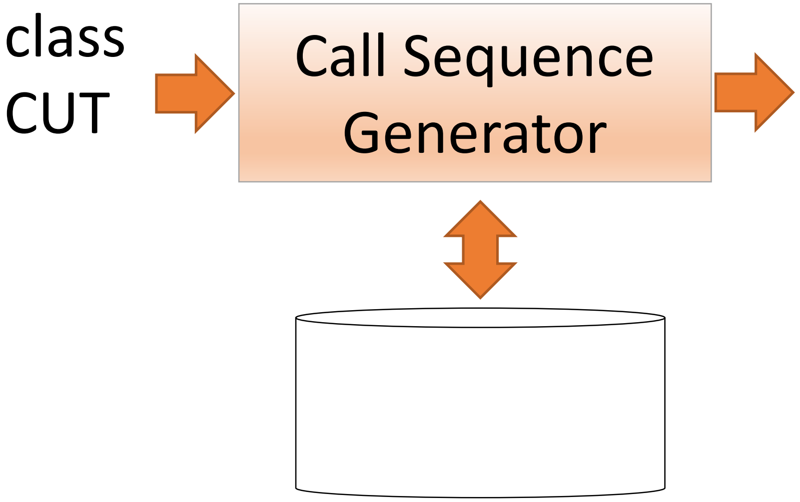
AutoConTest



AutoConTest



Call Sequence Generator

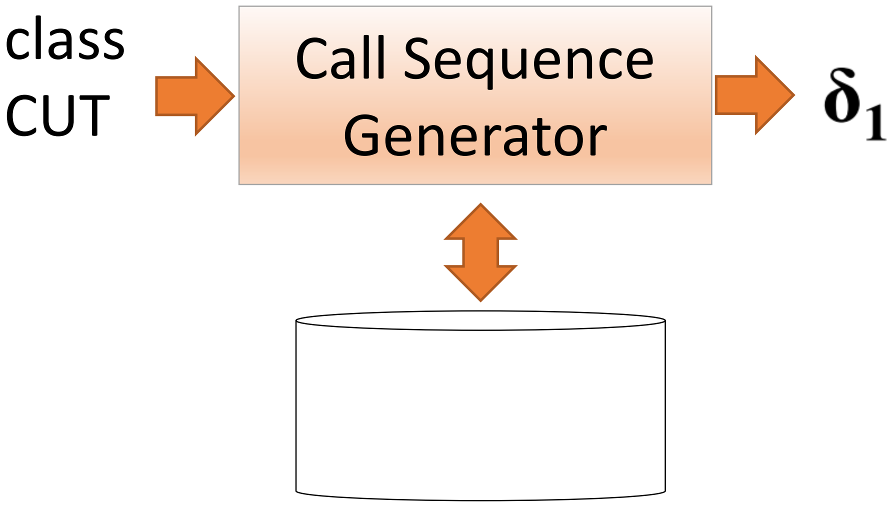



```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

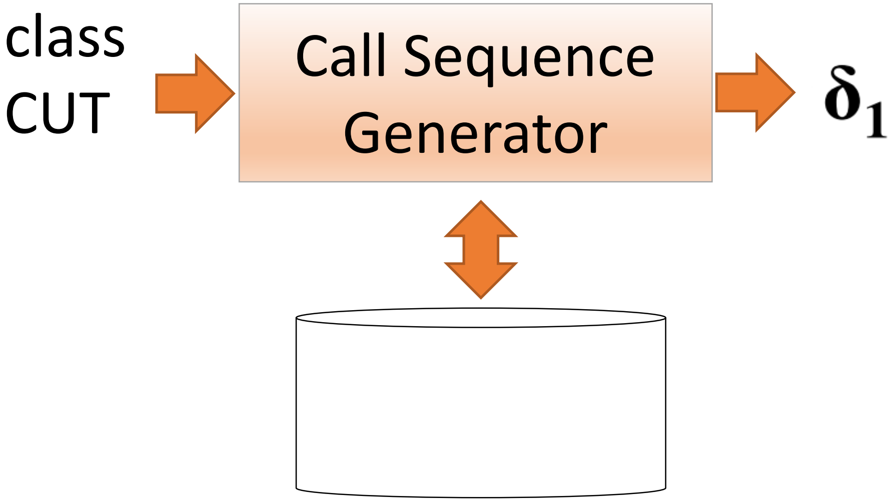
```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

δ_1

```
CUT sout = new CUT();
sout.m3(10);
sout.m2();
```

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

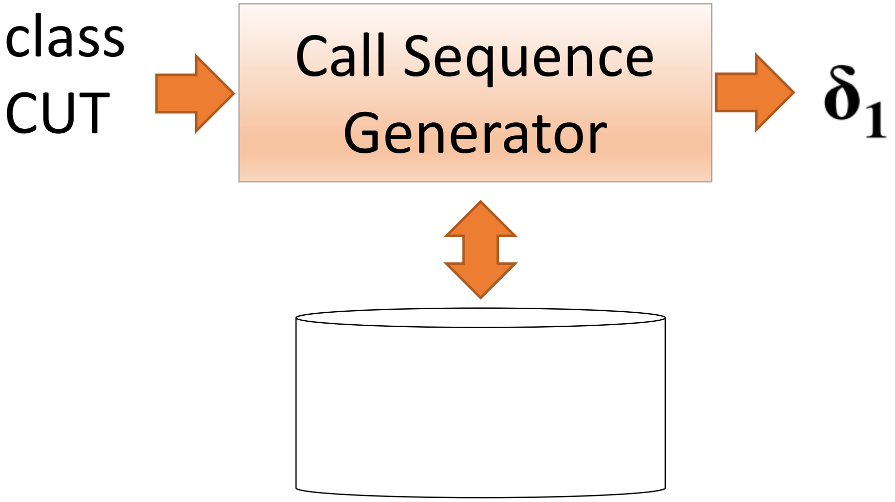
```
private void m4(int v1){
        $temp = x;
        x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

δ_1

```
CUT sout = new CUT();
    sout.m3(10);
    sout.m2();
```

Call Sequence Generator



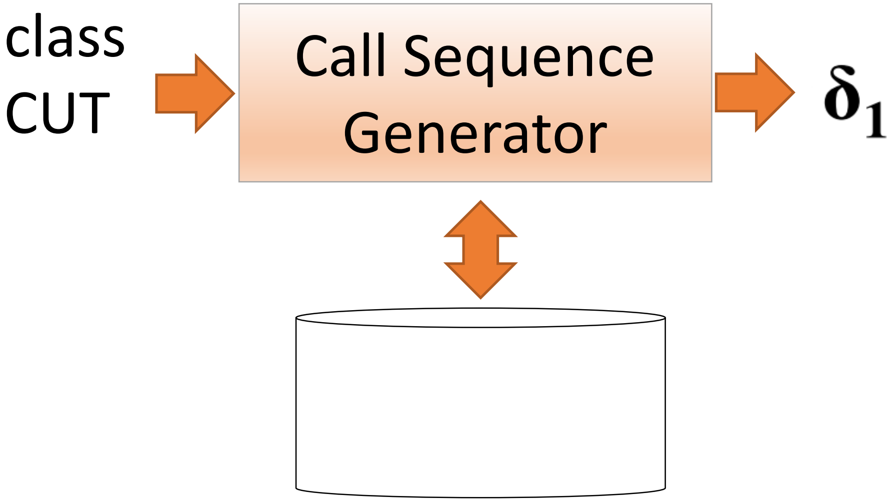

```
public void m3(int v1){
  if(z ==0){
      synchronized (this){
          m4(v1);
      }
  } else
          m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
public void m2(){
    z = 1;
}
```

δ_1

```
CUT sout = new CUT();
    sout.m3(10);
    sout.m2();
```

Call Sequence Generator




```
public void m3(int v1){
  if(z ==0){
    synchronized (this){
        m4(v1);
    }
  } else
        m4(v1);
}
```

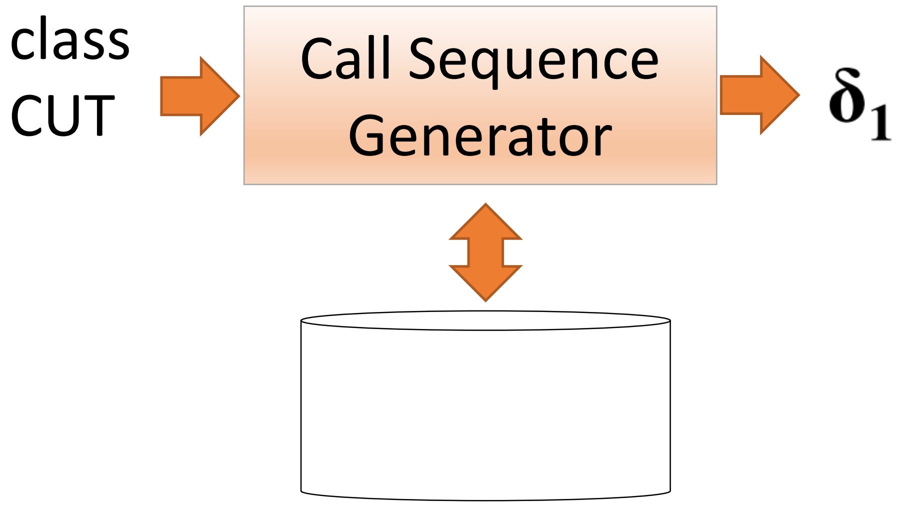
```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

δ_1

```
CUT sout = new CUT();
    sout.m3(10);
    sout.m2();
```

Call Sequence Generator




```
public void m3(int v1){
  if(z ==0){
    synchronized (this){
        m4(v1);
    }
  } else
        m4(v1);
}
```

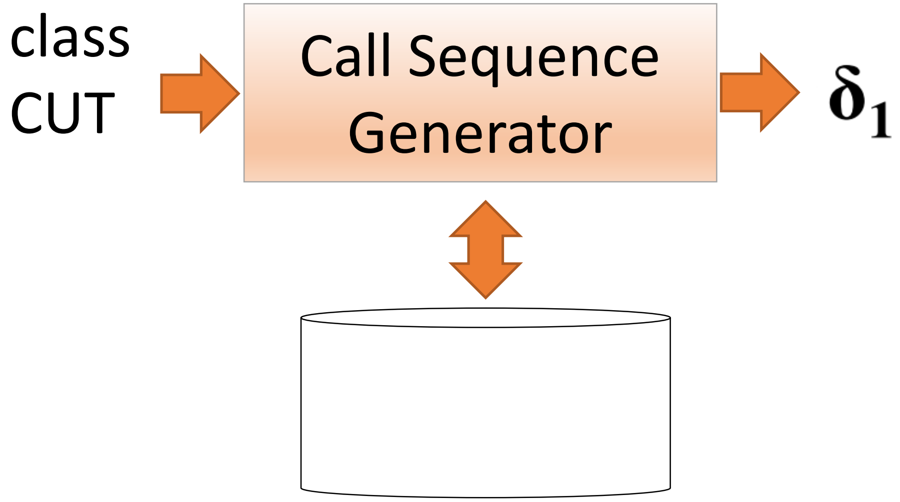
```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

δ_1

```
CUT sout = new CUT();
    sout.m3(10);
    sout.m2();
```

Call Sequence Generator




```
public void m3(int v1){
  if(z ==0){
    synchronized (this){
        m4(v1);
    }
  } else
        m4(v1);
}
```

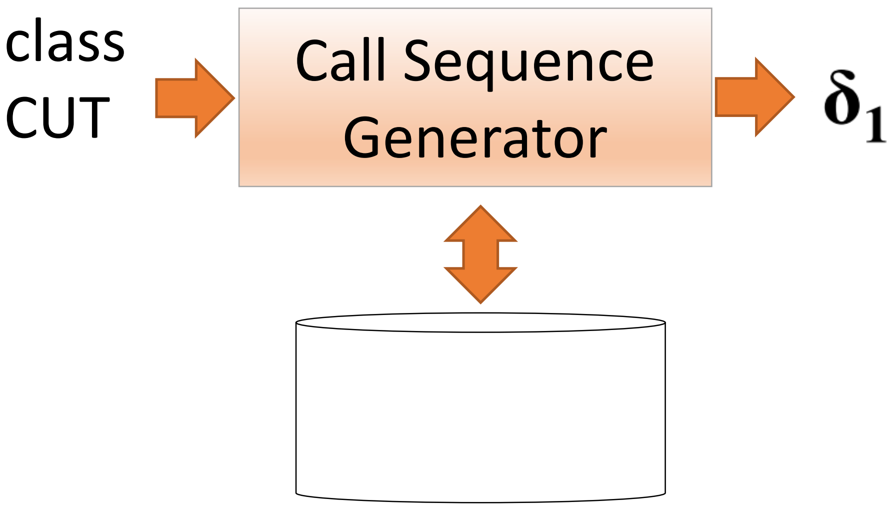
```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

δ_1

```
CUT sout = new CUT();
    sout.m3(10);
    sout.m2();
```

Call Sequence Generator




```
public void m3(int v1){
  if(z ==0){
    synchronized (this){
        m4(v1);
    }
  } else
        m4(v1);
}
```

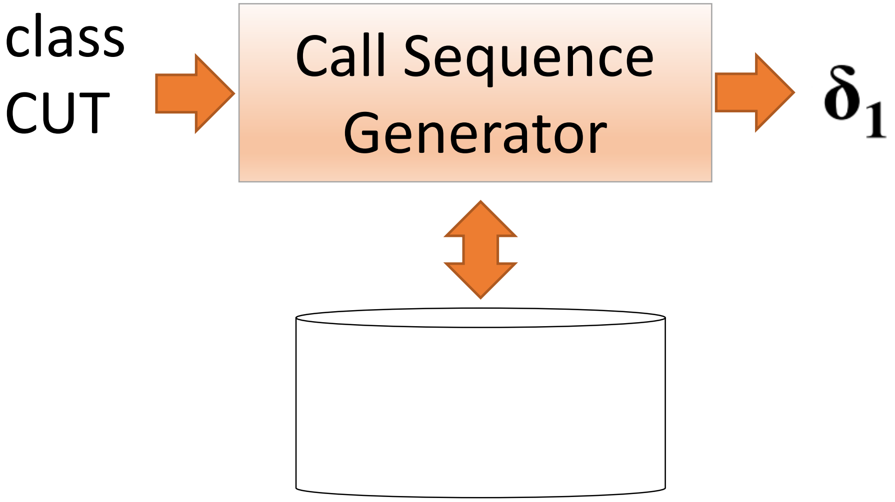
```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

δ_1

```
CUT sout = new CUT();
  sout.m3(10);
  sout.m2();
```

Call Sequence Generator




```
public void m3(int v1){
  if(z ==0){
    synchronized (this){
       m4(v1);
    }
  } else
       m4(v1);
}
```

```
private void m4(int v1){
   $temp = x;
   x = $temp + 1;
}
```

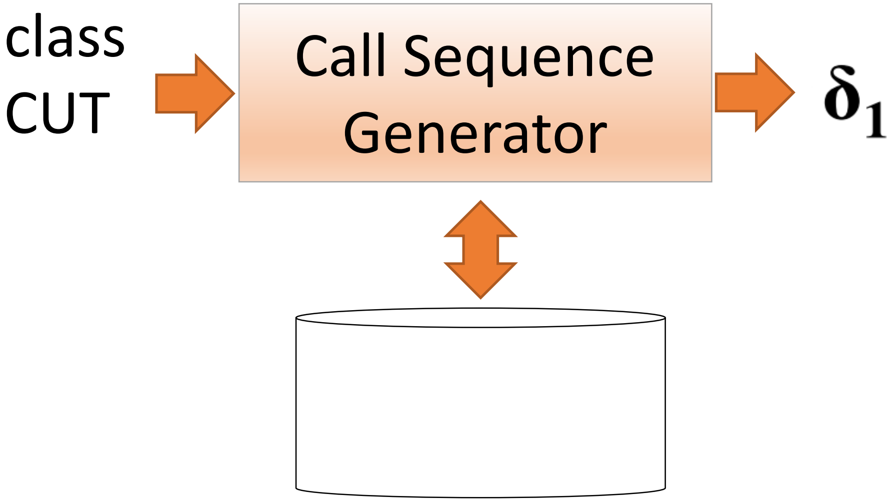
```
public void m2(){
   z = 1;
}
```

δ_1

```
CUT sout = new CUT();
    sout.m3(10);
     sout.m2();
```

new m3's behaviour

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

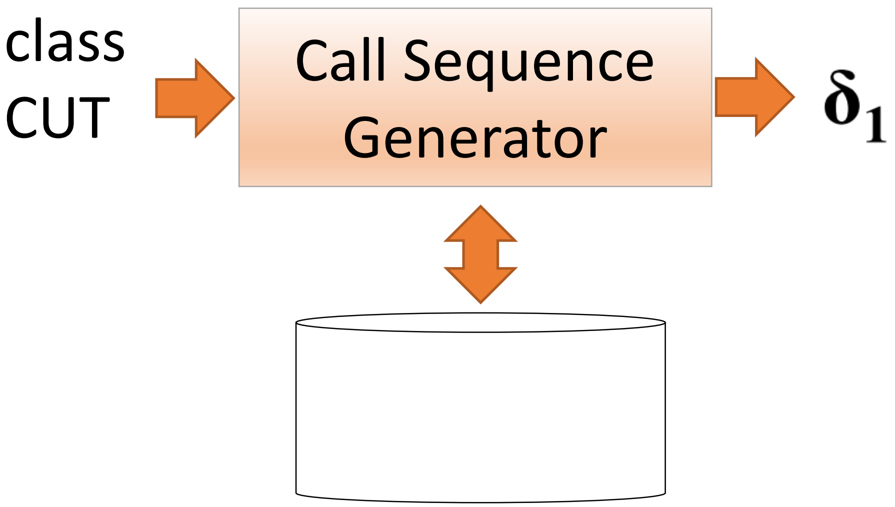
```
public void m2(){
    z = 1;
}
```

δ_1

```
CUT sout = new CUT();
    sout.m3(10);
    sout.m2();
```

new m3's behaviour

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

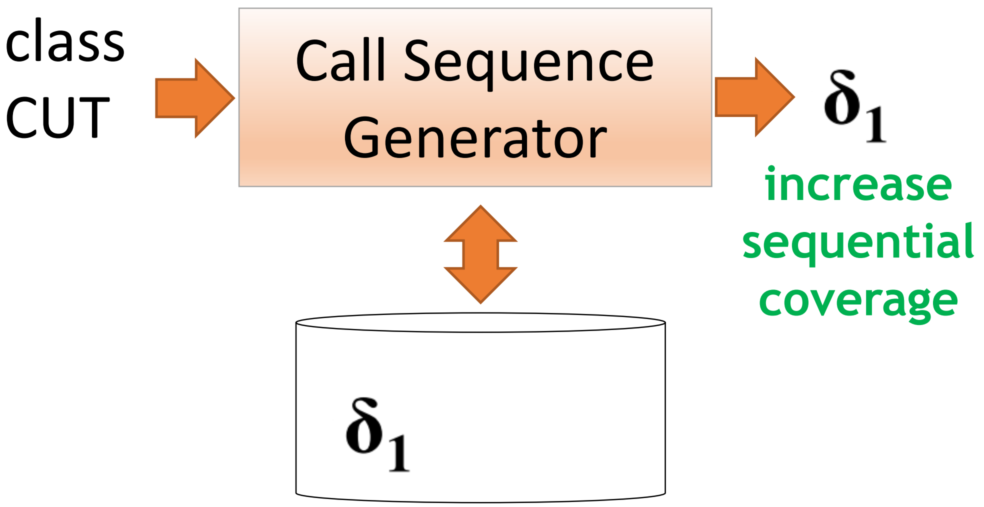
δ_1

```
CUT sout = new CUT();
    sout.m3(10);
    sout.m2();
```

new m3’s behaviour

new m2’s behaviour

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

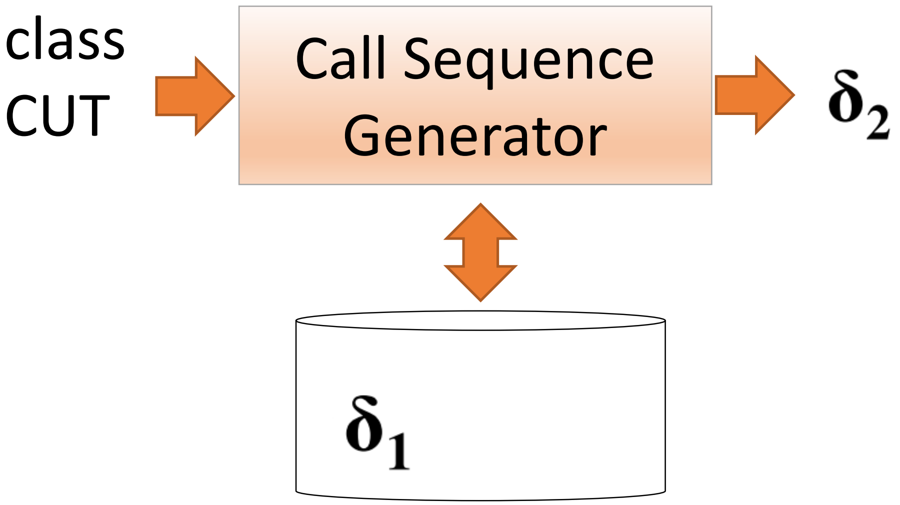
δ_1

```
CUT sout = new CUT();
    sout.m3(10);
     sout.m2();
```

new m3's behaviour

new m2's behaviour

Call Sequence Generator




```java
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

```java
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```java
public void m2(){
    z = 1;
}
```

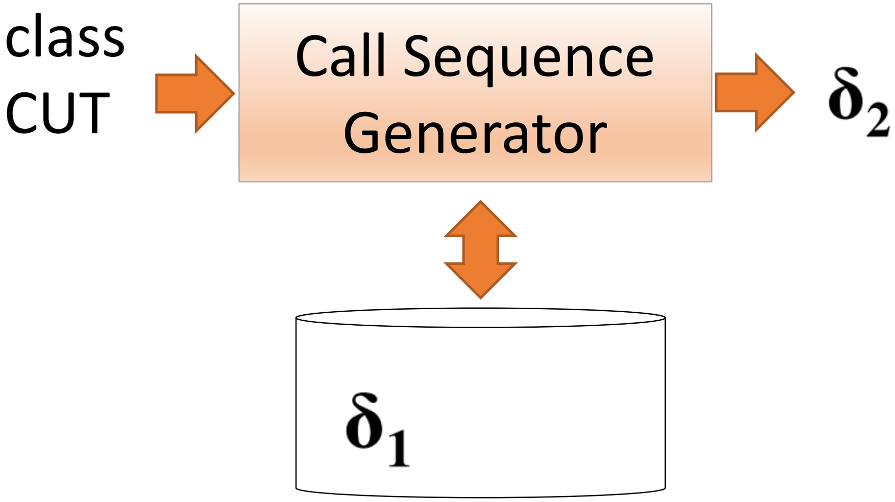
δ_1

```java
CUT sout = new CUT();
    sout.m3(10);      new m3's behaviour
    sout.m2();        new m2's behaviour
```

δ_2

```java
CUT sout = new CUT();
    sout.m2();
    sout.m3(10);
```

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

δ1

```
CUT sout = new CUT();
    sout.m3(10);
    sout.m2();
```

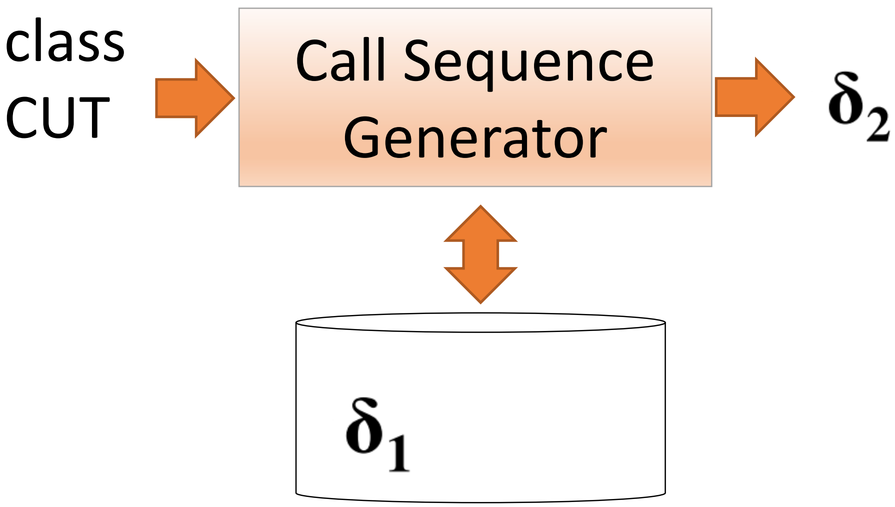
new m3's behaviour

new m2's behaviour

δ2

```
CUT sout = new CUT();
    sout.m2();
    sout.m3(10);
```

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

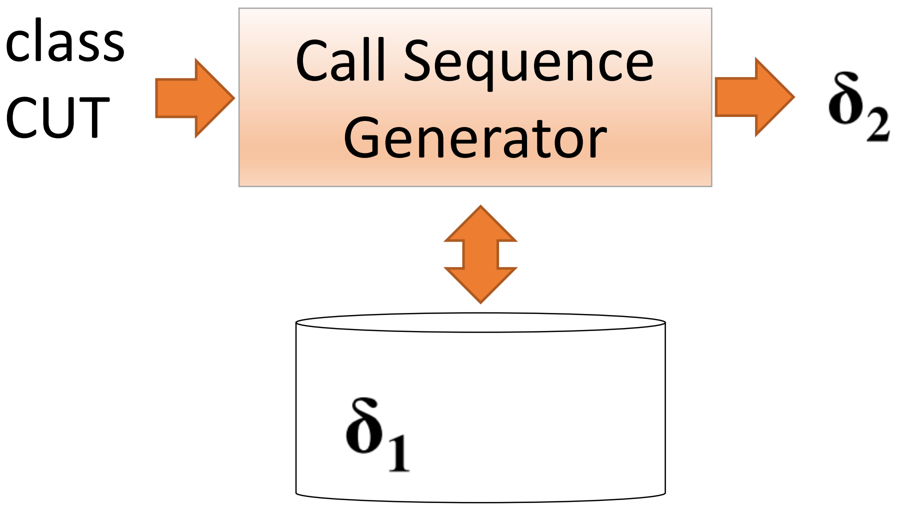
δ_1

```
CUT sout = new CUT();
    sout.m3(10);      new m3's behaviour
    sout.m2();        new m2's behaviour
```

δ_2

```
CUT sout = new CUT();
    sout.m2();
    sout.m3(10);
```

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

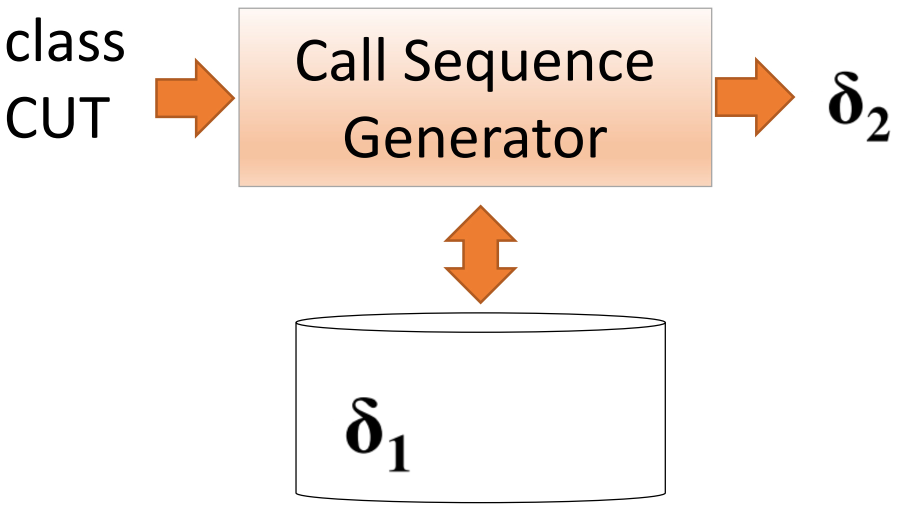
δ_1

```
CUT sout = new CUT();
    sout.m3(10);      new m3's behaviour
    sout.m2();        new m2's behaviour
```

δ_2

```
CUT sout = new CUT();
    sout.m2();
    sout.m3(10);
```

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
            m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

δ_1

```
CUT sout = new CUT();
sout.m3(10);
sout.m2();
```

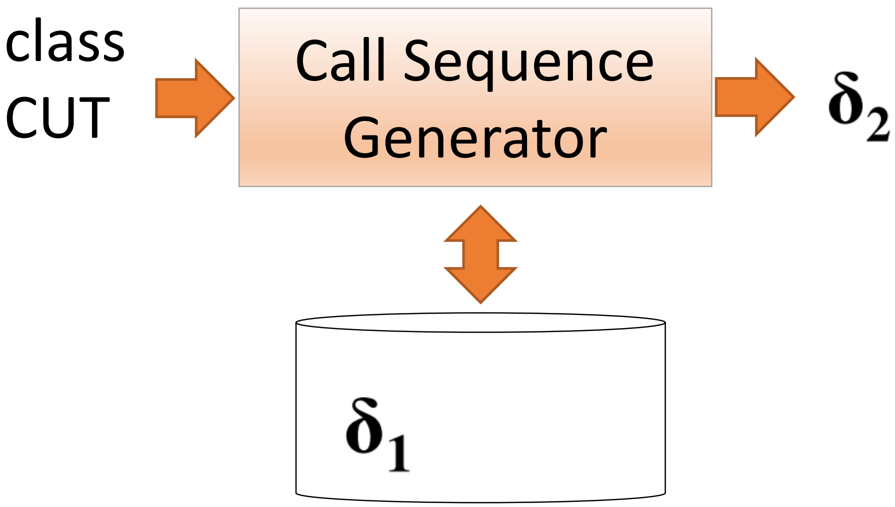
new m3's behaviour

new m2's behaviour

δ_2

```
CUT sout = new CUT();
sout.m2();
sout.m3(10);
```

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
        m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

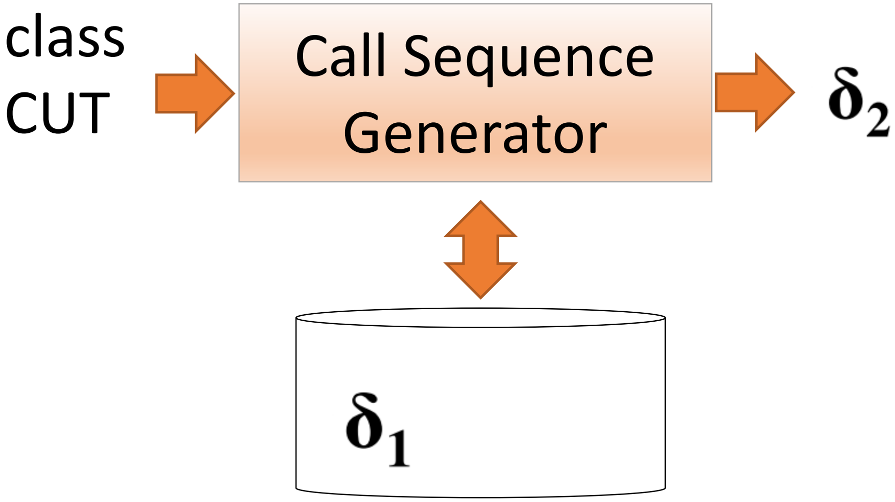
δ_1

```
CUT sout = new CUT();
    sout.m3(10);      new m3's behaviour
    sout.m2();        new m2's behaviour
```

δ_2

```
CUT sout = new CUT();
    sout.m2();
    sout.m3(10);
```

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
        m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

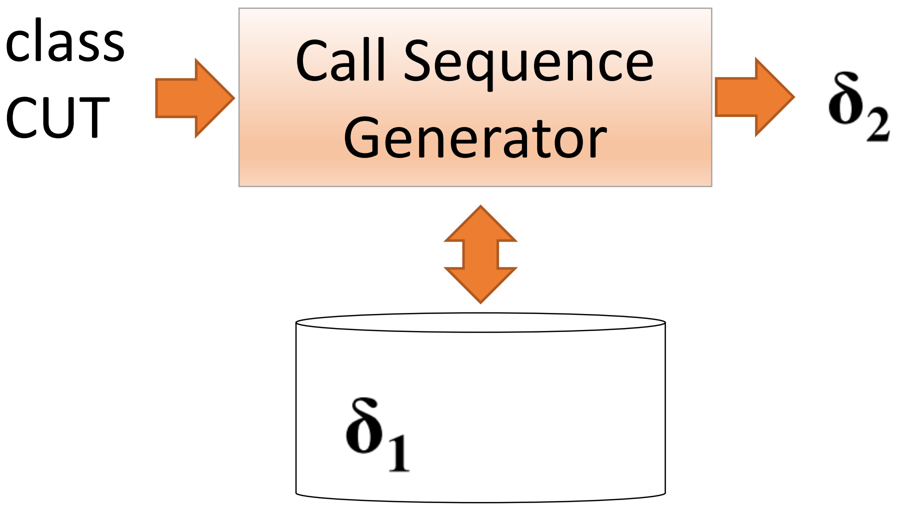
δ_1

```
CUT sout = new CUT();
sout.m3(10);    new m3's behaviour
sout.m2();      new m2's behaviour
```

δ_2

```
CUT sout = new CUT();
sout.m2();
sout.m3(10);
```

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
        m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

δ_1

```
CUT sout = new CUT();
sout.m3(10);
sout.m2();
```

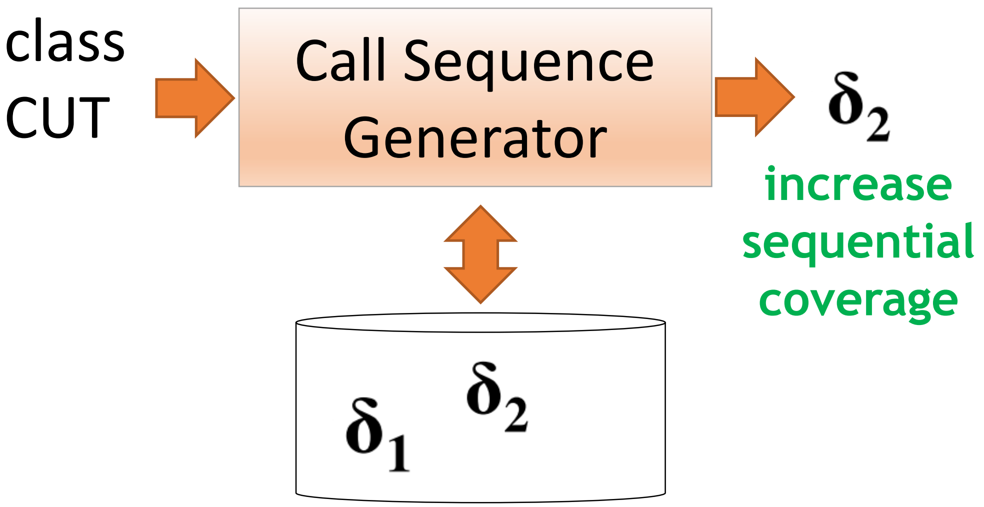
new m3's behaviour

new m2's behaviour

δ_2

```
CUT sout = new CUT();
sout.m2();
sout.m3(10);
```

Call Sequence Generator




```
public void m3(int v1){
    if(z ==0){
        synchronized (this){
            m4(v1);
        }
    } else
        m4(v1);
}
```

```
private void m4(int v1){
    $temp = x;
    x = $temp + 1;
}
```

```
public void m2(){
    z = 1;
}
```

δ1

```
CUT sout = new CUT();
sout.m3(10);
sout.m2();
```

new m3's behaviour

new m2's behaviour

δ2

```
CUT sout = new CUT();
sout.m2();
sout.m3(10);
```

new m3's behaviour

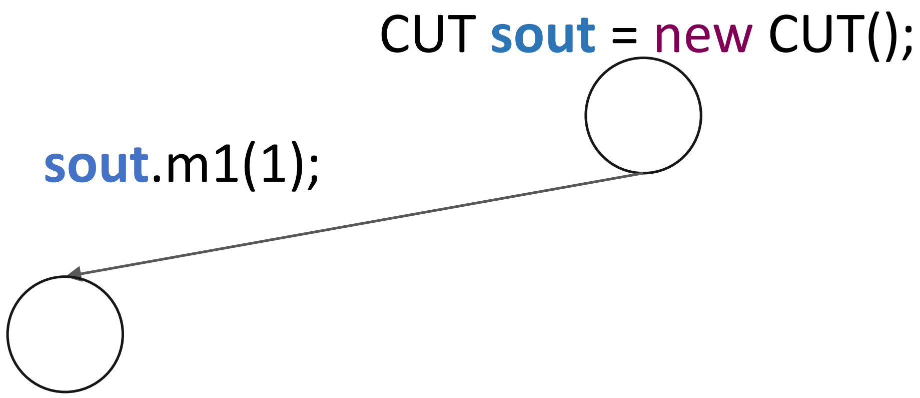
Call Sequence Generator

- **Search Space**
  - CUT methods
  - fixed pool of parameters values (at each iteration)

```
CUT sout = new CUT();
```

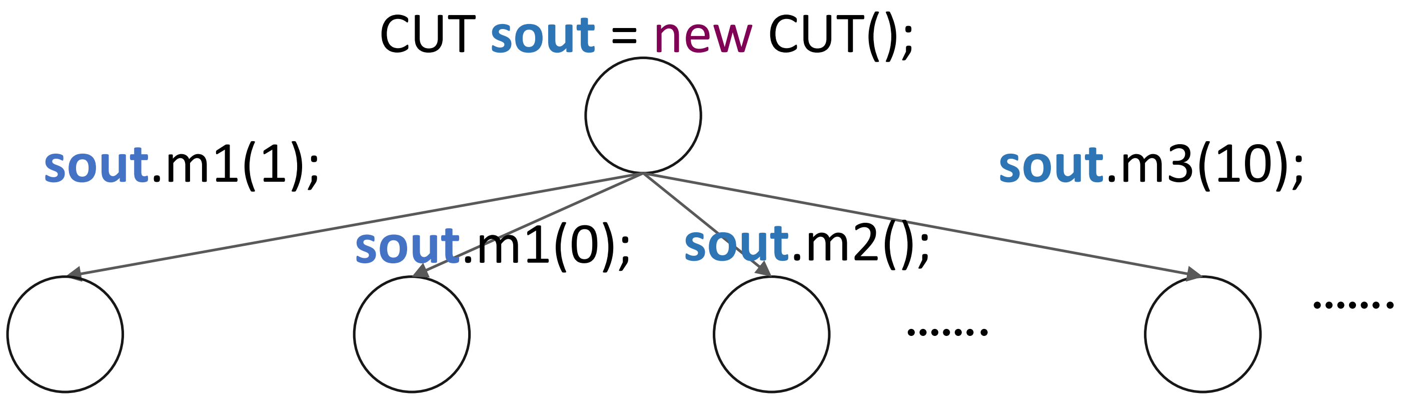
Call Sequence Generator

- **Search Space**
  - CUT methods
  - fixed pool of parameters values (at each iteration)



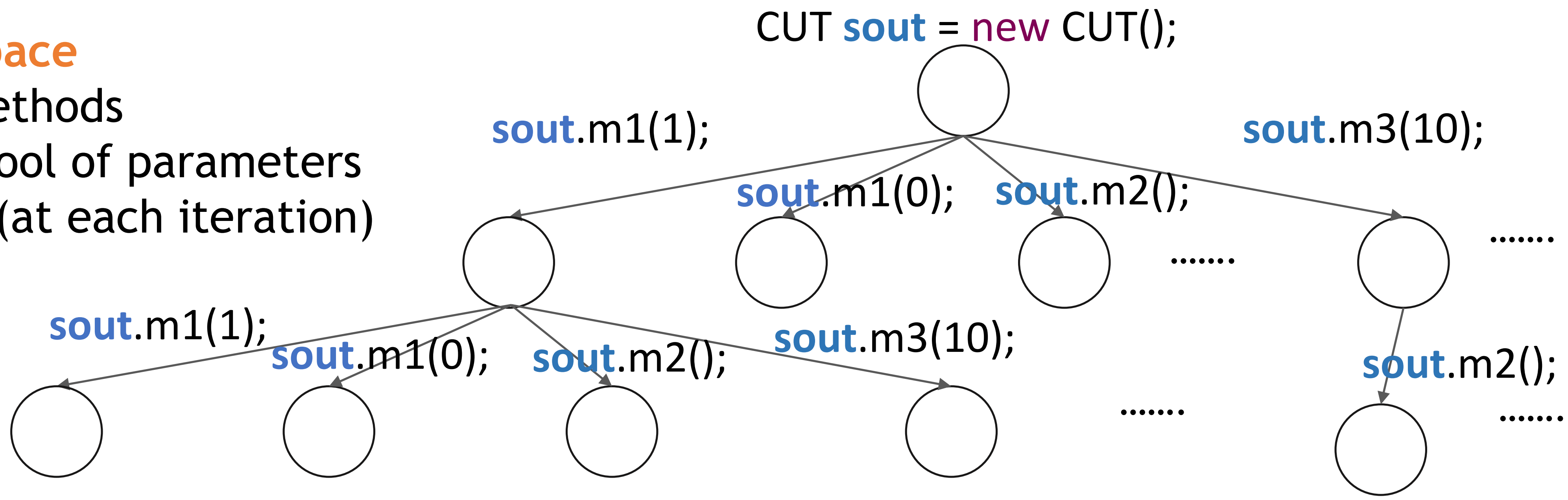
Call Sequence Generator

- **Search Space**
  - CUT methods
  - fixed pool of parameters values (at each iteration)



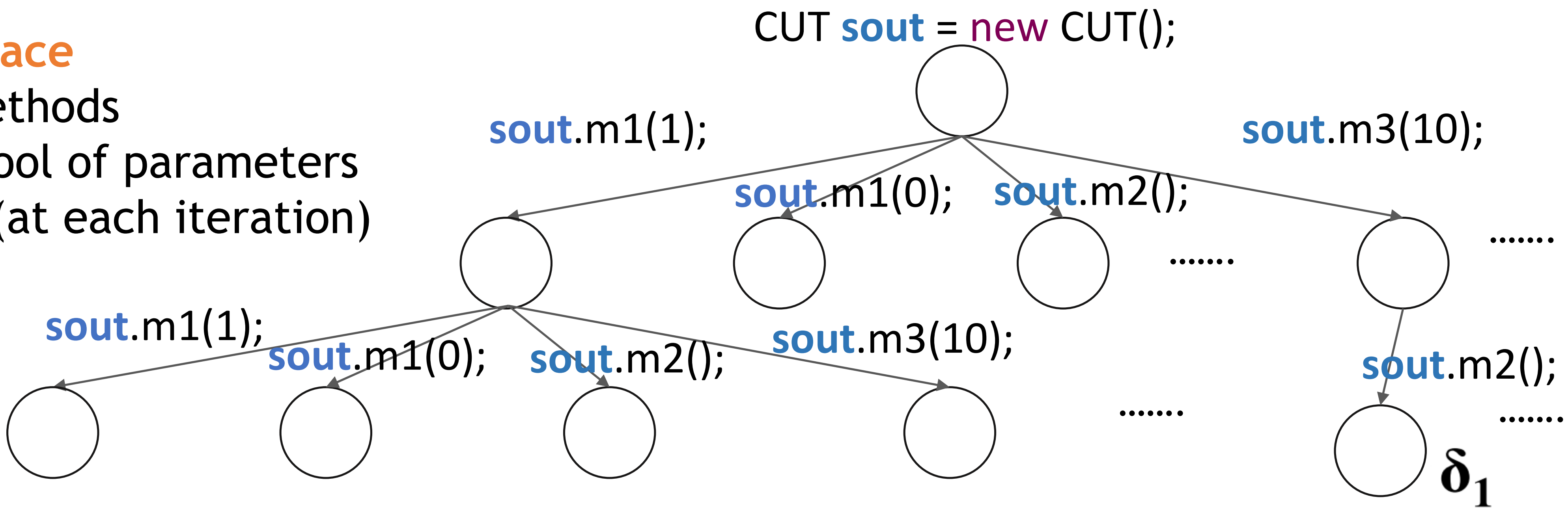
Call Sequence Generator

- **Search Space**
  - CUT methods
  - fixed pool of parameters values (at each iteration)



Call Sequence Generator

- **Search Space**
  - CUT methods
  - fixed pool of parameters values (at each iteration)

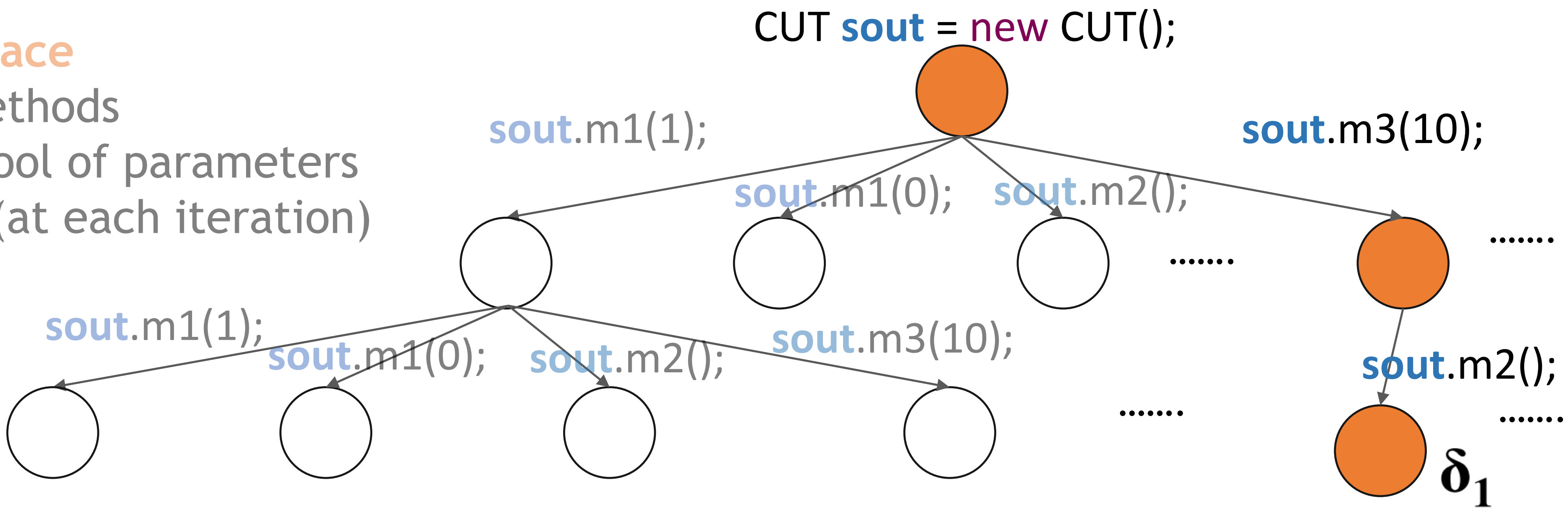


δ_1

```
CUT sout = new CUT();
sout.m3(10);
sout.m2();
```

Call Sequence Generator

- Search Space
  - CUT methods
  - fixed pool of parameters values (at each iteration)

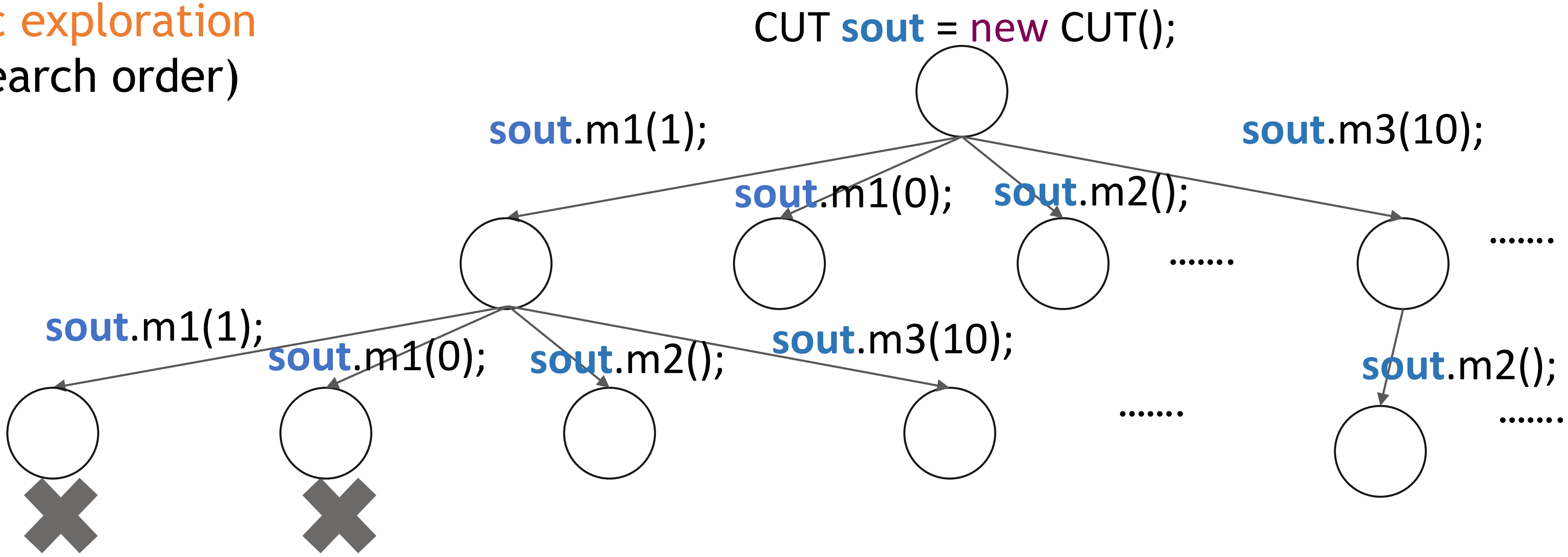


δ_1

```
CUT sout = new CUT();
sout.m3(10);
sout.m2();
```

Call Sequence Generator

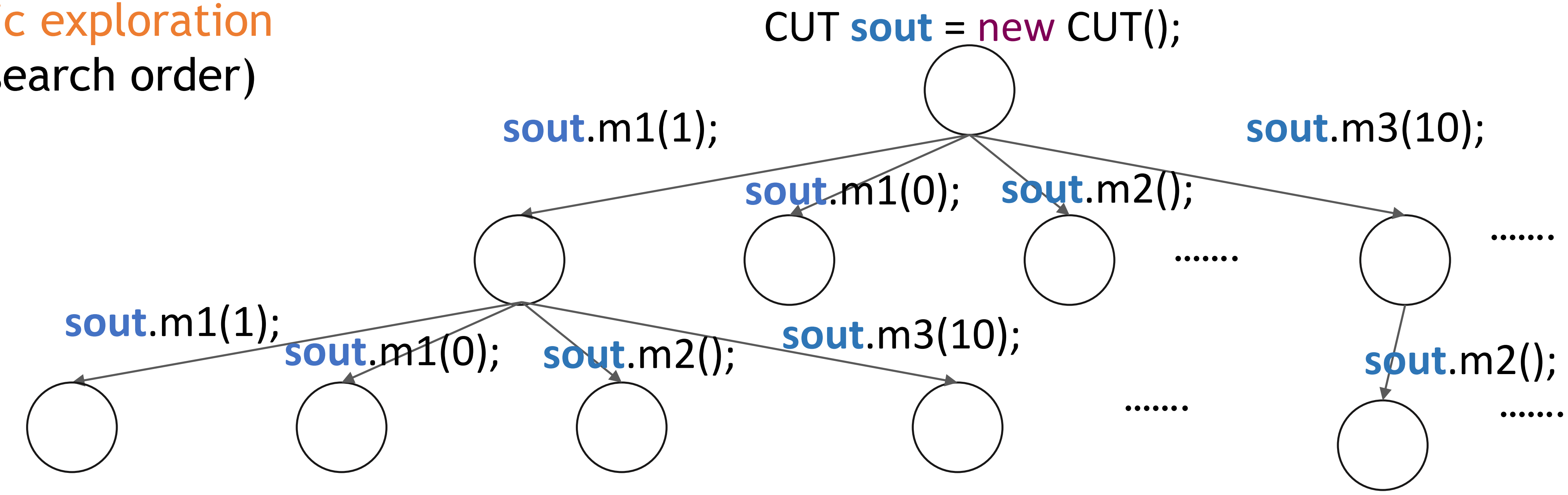
- Systematic exploration
  - DFS (search order)



- Saturation-based stopping criterion
  - Stop extending a node if the latest K extensions have not increased the **sequential coverage**

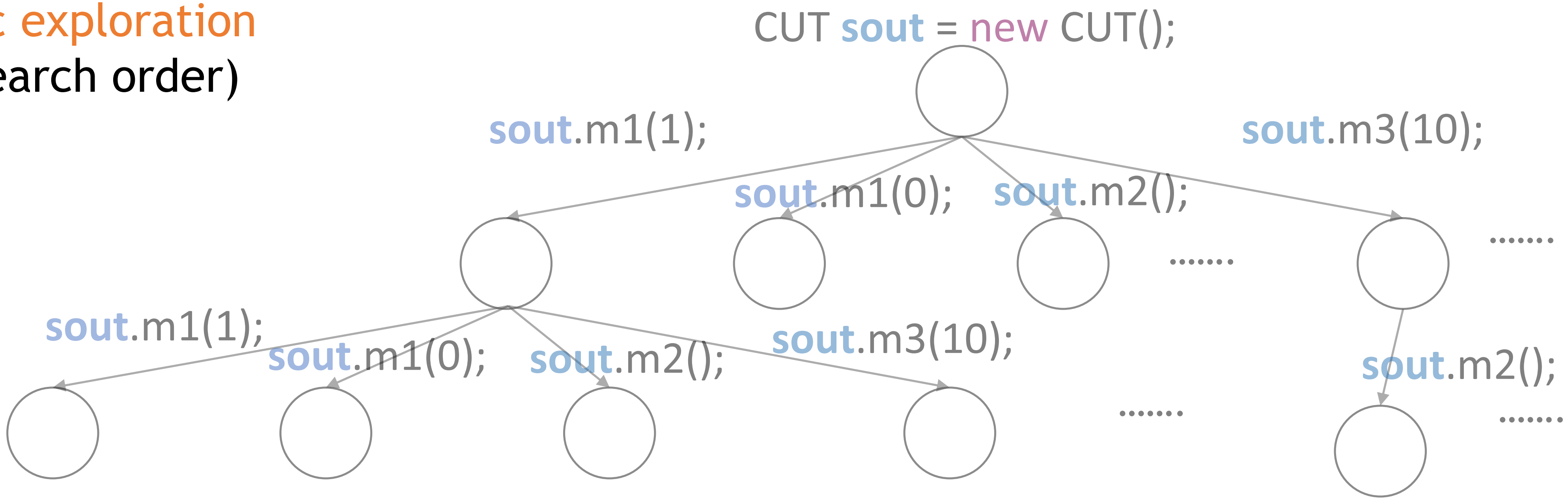
Call Sequence Generator

- Systematic exploration
  - DFS (search order)



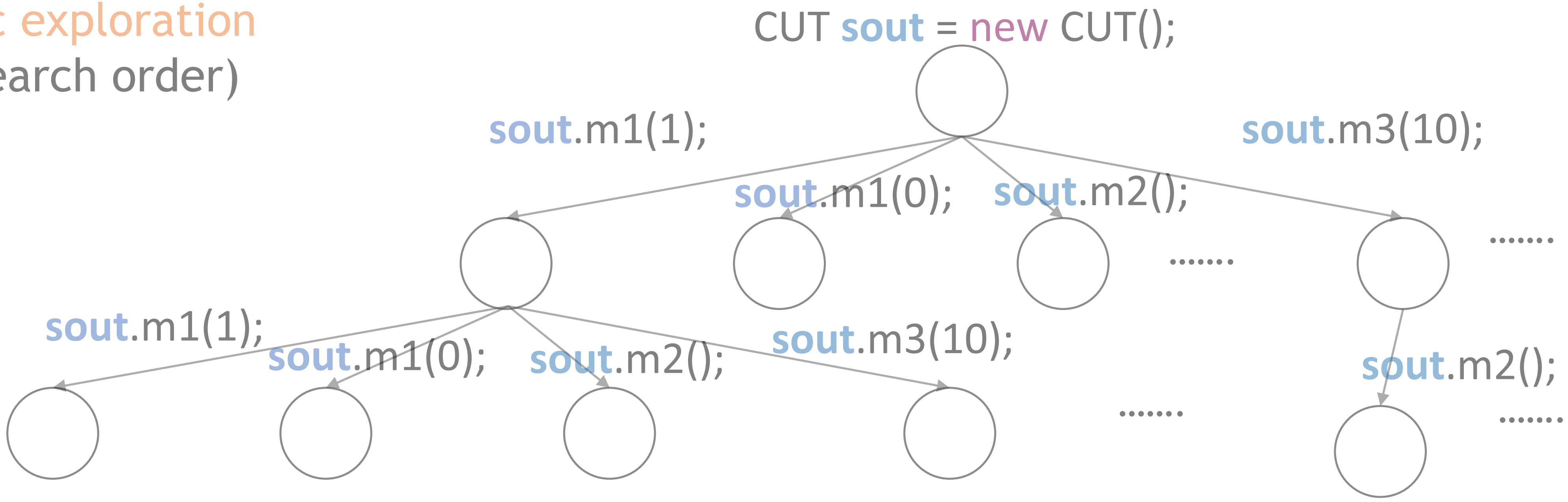
Call Sequence Generator

- Systematic exploration
  - DFS (search order)



Call Sequence Generator

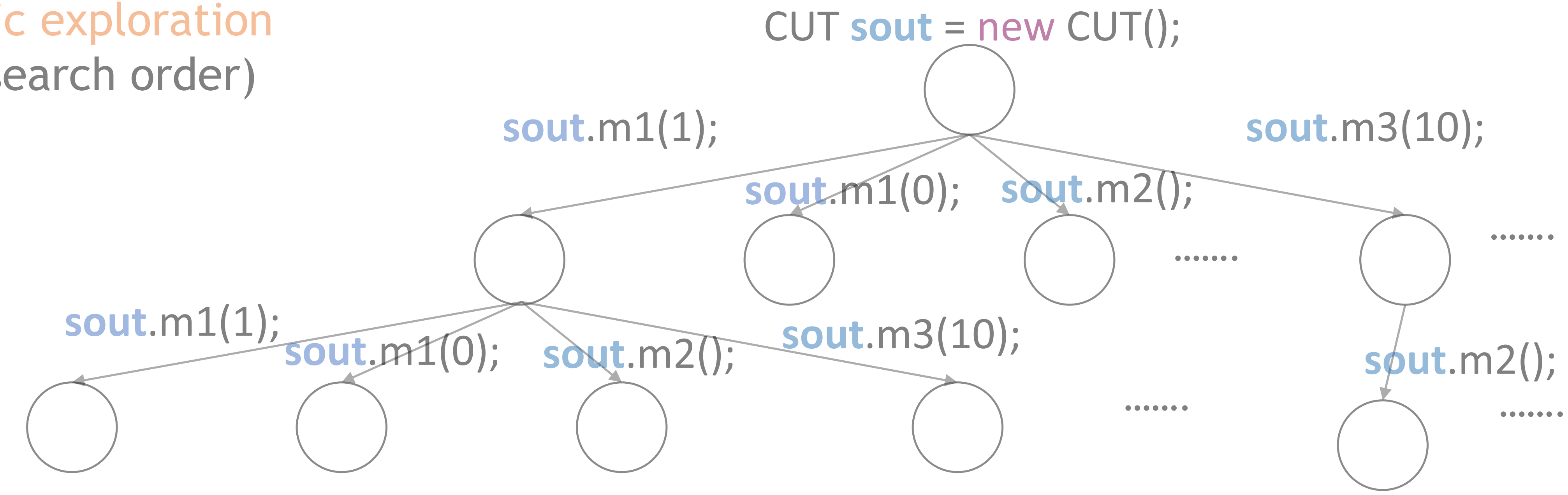
- Systematic exploration
  - DFS (search order)



- To **minimize** the number of tests and **maximize** the coverage
- At each iteration identify an **optimal sequence** with the highest coverage improvement
  - # method calls that **increase sequential coverage**

Call Sequence Generator

- Systematic exploration
  - DFS (search order)

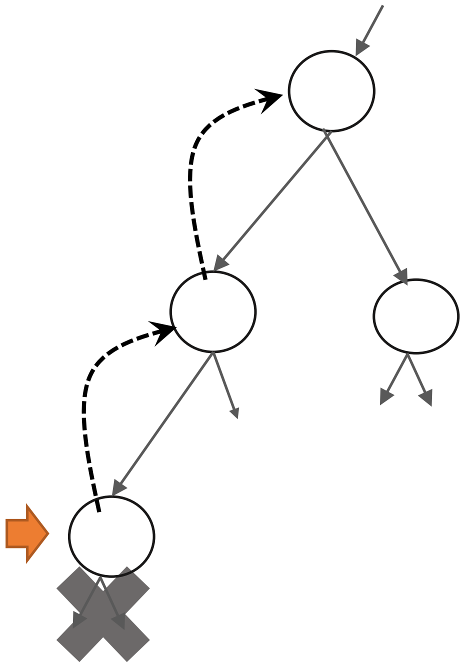


- To **minimize** the number of tests and **maximize** the coverage
- At each iteration identify an **optimal sequence** with the highest coverage improvement

  # method calls that **increase sequential coverage**

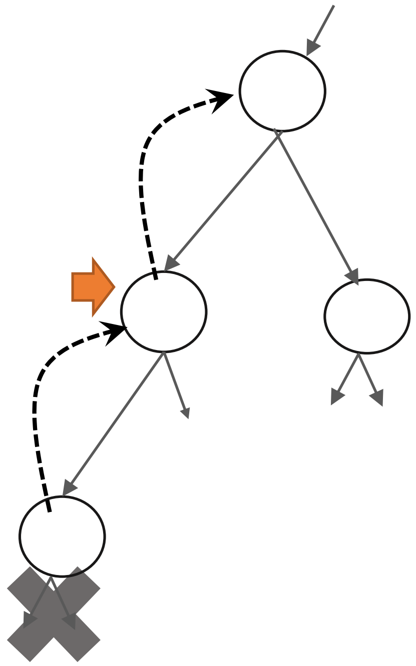
search space is too huge!

Search Space Pruning Strategies



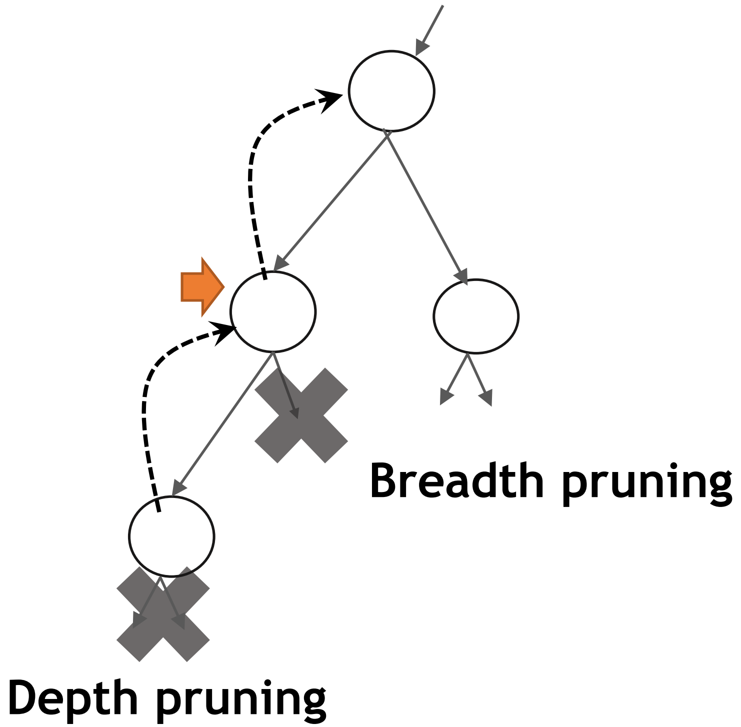
Depth pruning

Search Space Pruning Strategies

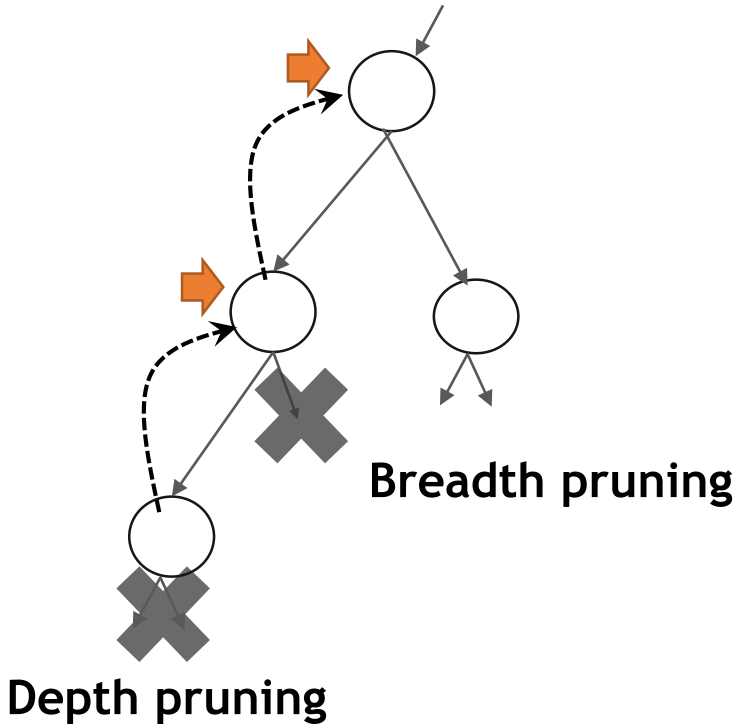


Depth pruning

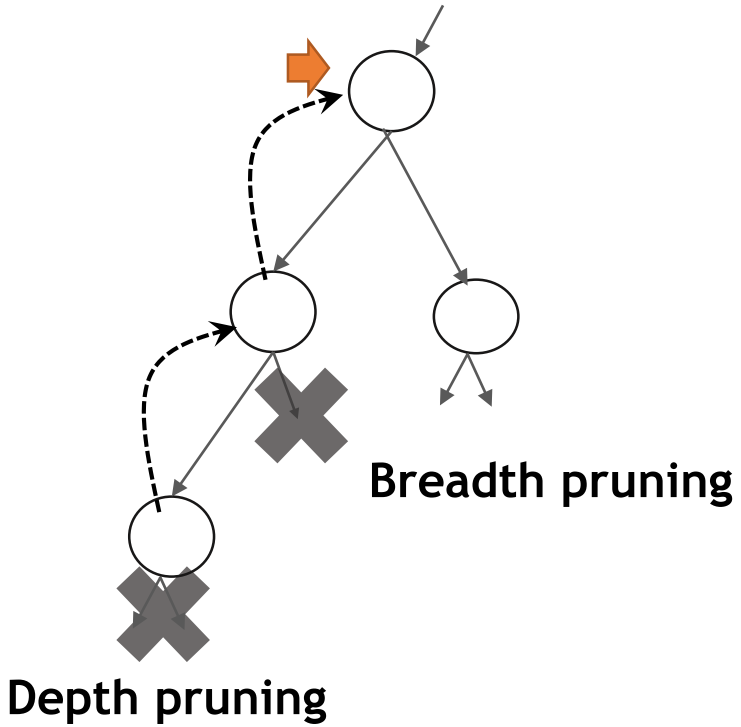
Search Space Pruning Strategies



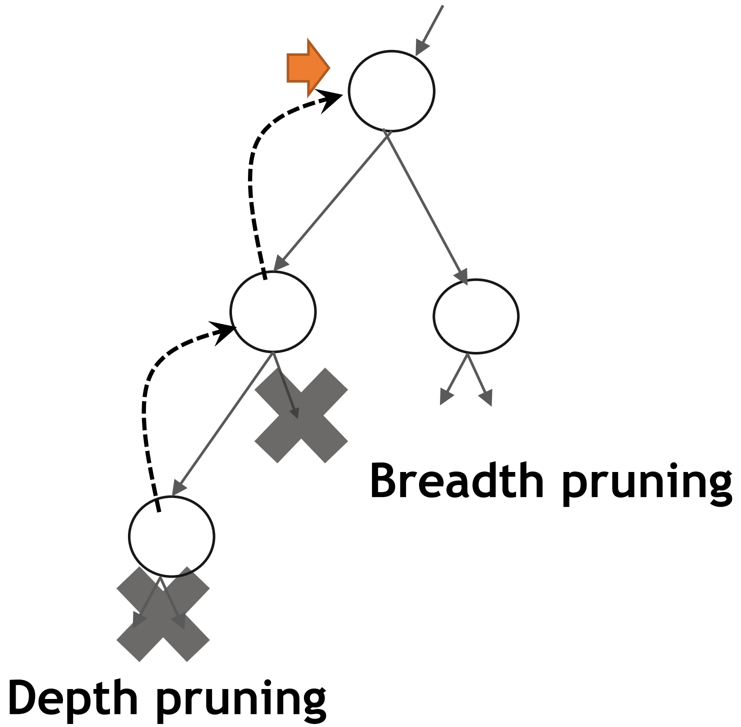
Search Space Pruning Strategies



Search Space Pruning Strategies

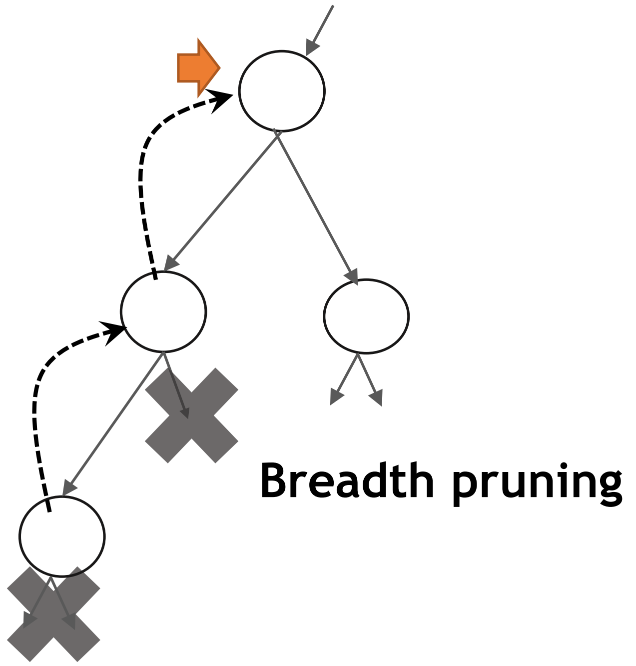


Search Space Pruning Strategies



Assumption: The sequential execution of call sequences is deterministic

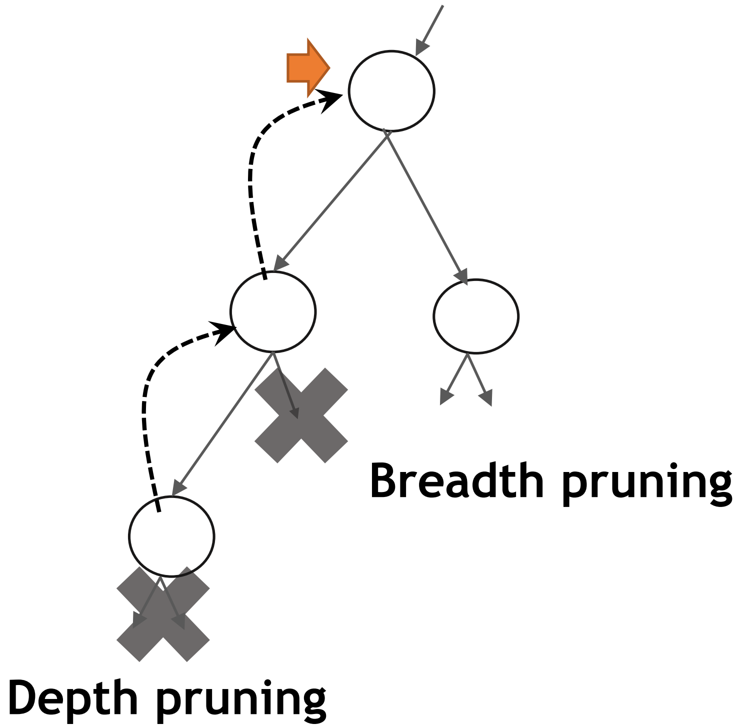
Search Space Pruning Strategies



Assumption: The sequential execution of call sequences is deterministic

- **Redundancy**
  - Based on **program states** and sequential coverage

Search Space Pruning Strategies



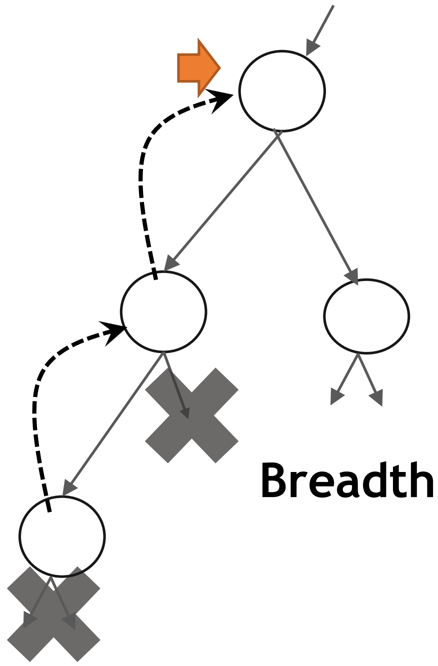
Assumption: The sequential execution of call sequences is deterministic

- **Redundancy**
  - Based on **program states** and sequential coverage

Theorem 1

The unexplored descendants of a node representing a **redundant** call sequence do not need to be explored in order to reach the optimal solution

Search Space Pruning Strategies



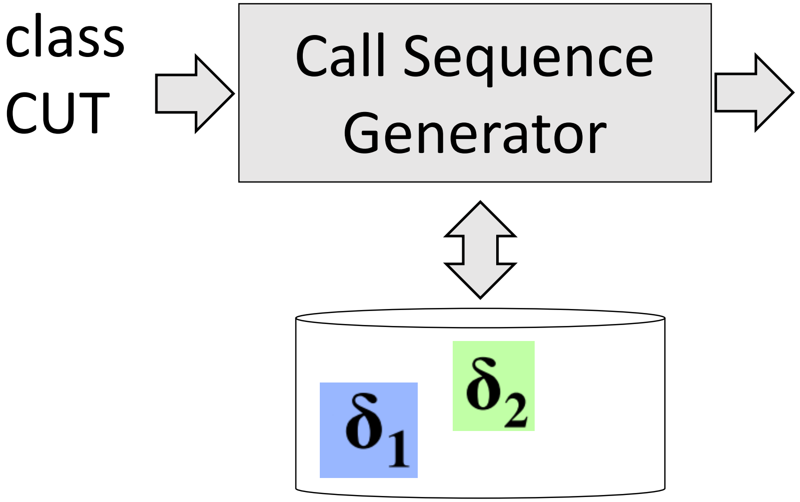
Assumption: The sequential execution of call sequences is deterministic

- Redundancy
  - Based on program states and sequential coverage

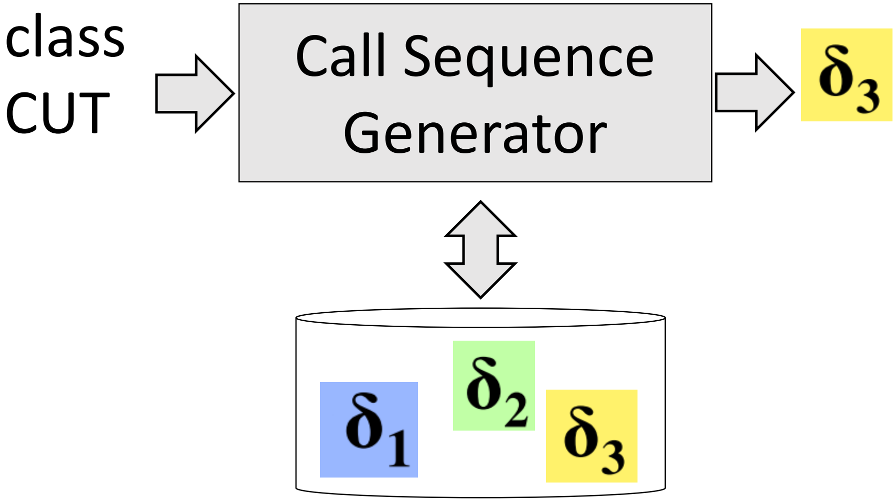
Theorem 1

The unexplored descendants of a node representing a redundant call sequence do not need to be explored in order to reach the optimal solution

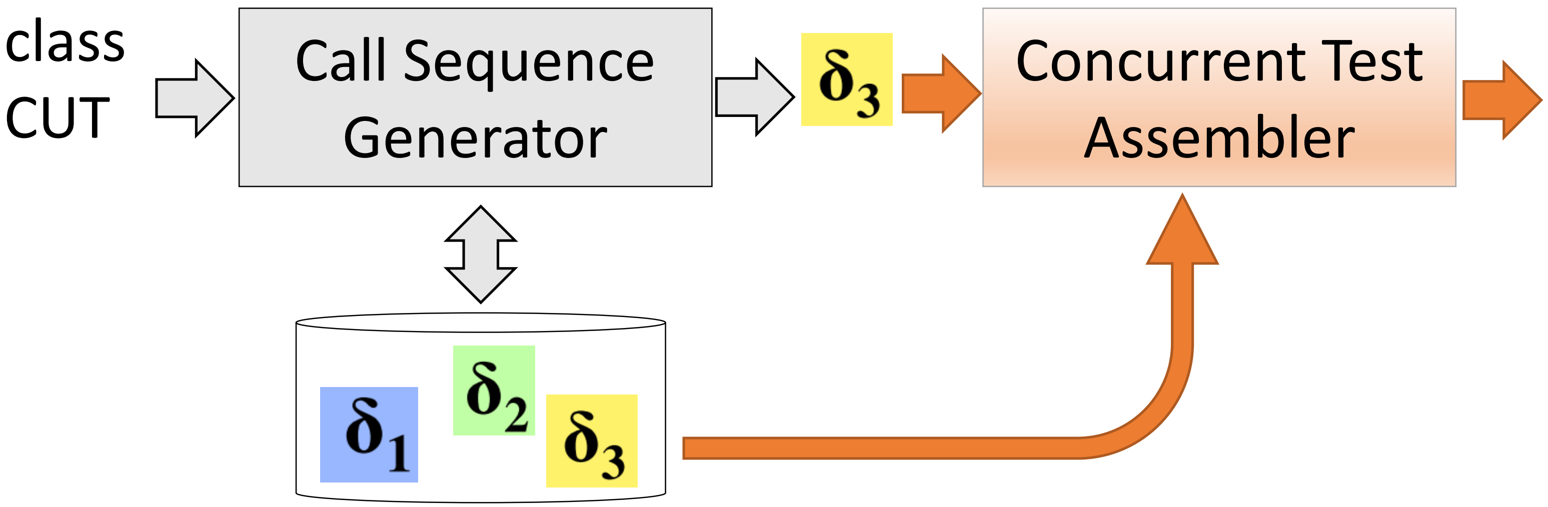
Concurrent Test Assembler



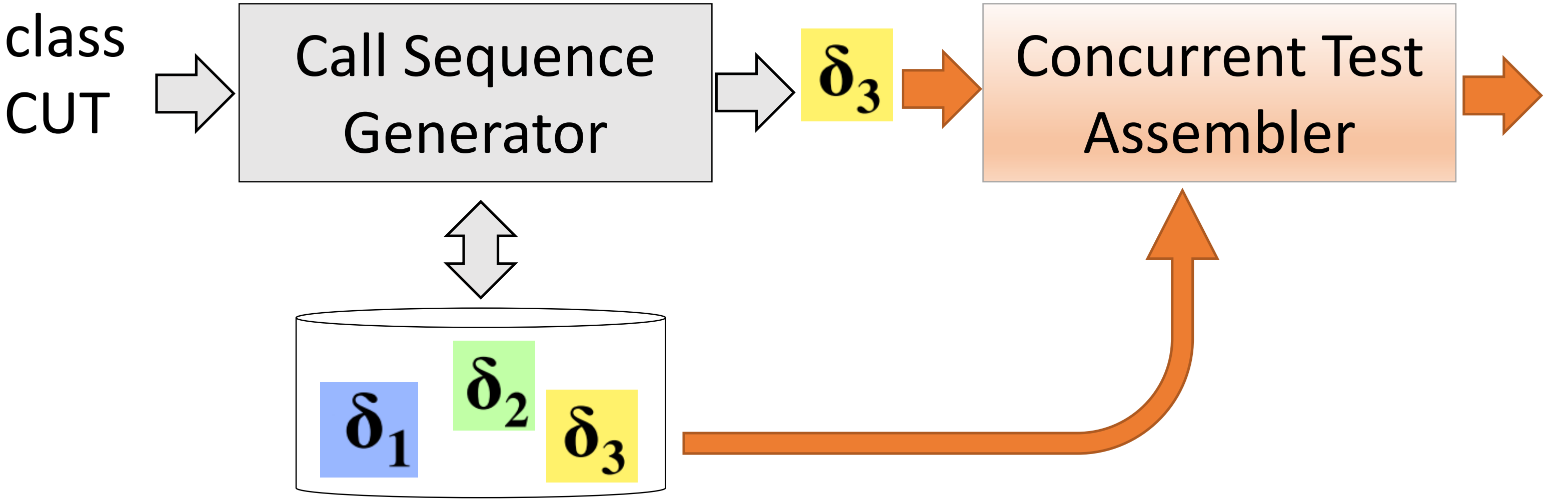
Concurrent Test Assembler

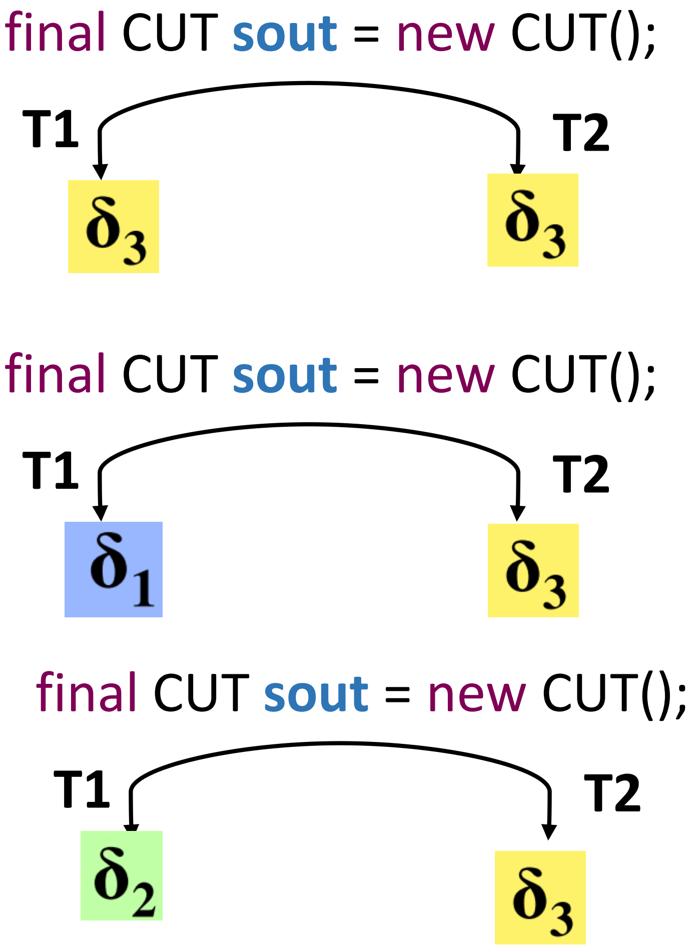


Concurrent Test Assembler



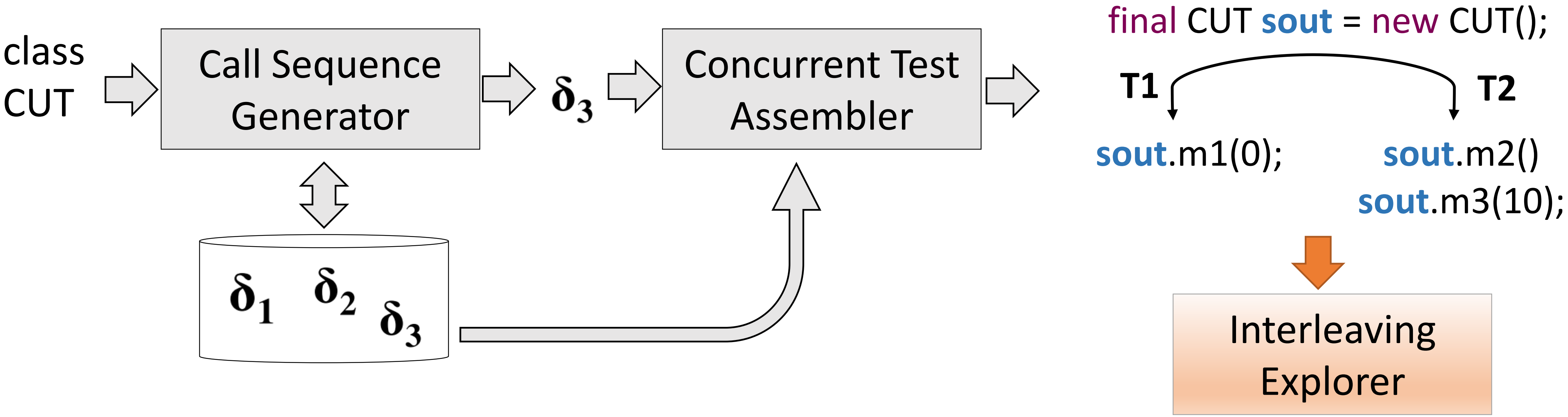
Concurrent Test Assembler



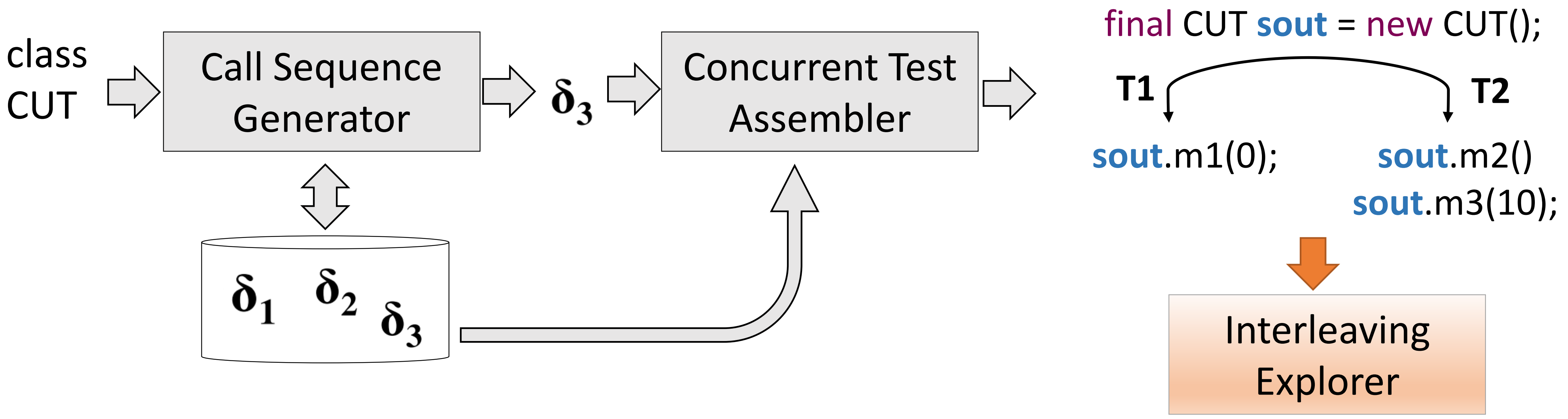


- The returned call sequences returned call sequences are concurrently **pair-wise tested**
- **Necessary condition** for increasing interleaving coverage (Theorem 2)

Interleaving Explorer



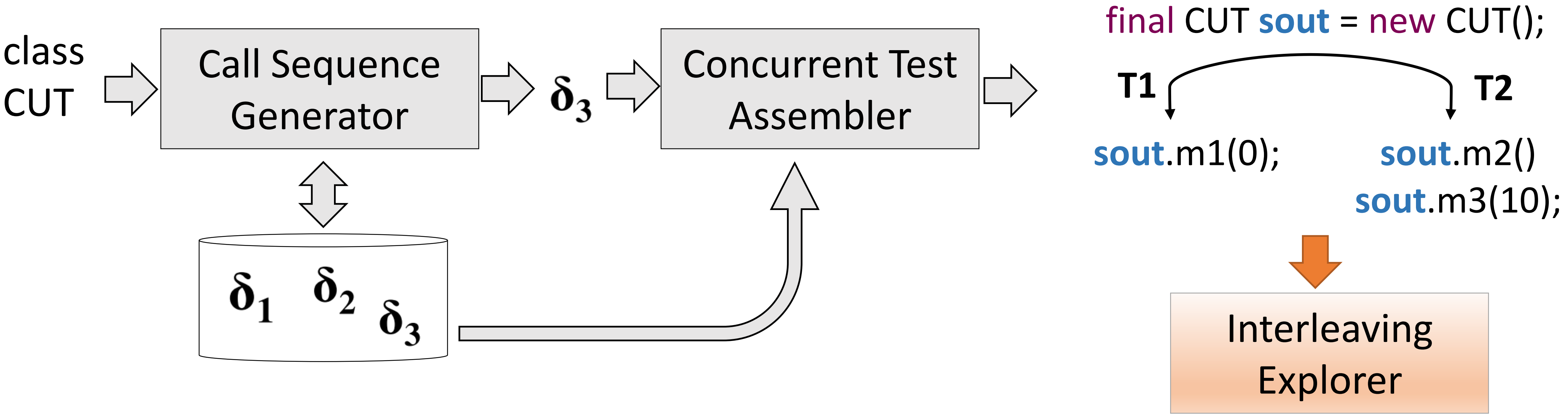
Interleaving Explorer



Predictive Trace Analysis (PTA) [Lai et al. ICSE 2016]

To compute the **interleaving coverage requirements** of the test code

Interleaving Explorer



Predictive Trace Analysis (PTA) [Lai et al. ICSE 2016]

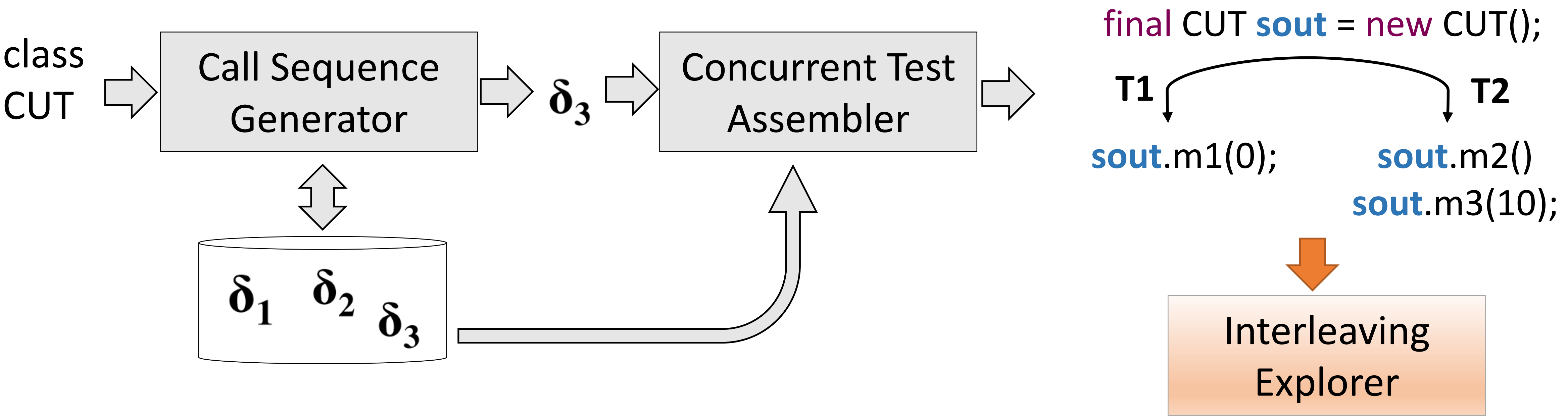
+

Thread Scheduler

To compute the interleaving coverage requirements of the test code

To check if the requirements are feasible or infeasible

Interleaving Explorer



Predictive Trace Analysis (PTA) [Lai et al. ICSE 2016]

+

Thread Scheduler

To compute the interleaving coverage requirements of the test code

To check if the requirements are feasible or infeasible

Challenge: non-deterministic interferences

Subjects

BUG ID	Code Base	Class Under Test	CUT SLOC
1	Apache Commons 2.4	[..].lang.math.IntRange	278
2	Google Commons 1.0	[...]AbstractMultiMap$AsMap	1125
3	Java JDK 1.1.7	java.util.Vector	216
4	JFreeChart 0.9	[...]chart.axis.NumberAxis	1298
5	Java JDK 1.1.7	java.util.logging.Logger	992
6	Java JDK 1.4.2	java.util.Vector	326

6 real, known concurrency bugs
atomic-set serializability violations

RQ1:Cost-Effectiveness

BUG ID	Code Base	CUT SLOC	Time first fault
1	Apache Commons 2.4	278	22 sec
2	Google Commons 1.0	1125	29 sec
3	Java JDK 1.1.7	216	65 sec
4	JFreeChart 0.9	1298	35 sec
5	Java JDK 1.1.7	992	45 sec
6	Java JDK 1.4.2	326	30 sec

RQ1:Cost-Effectiveness

BUG ID	Code Base	CUT SLOC	Time first fault
1	Apache Commons 2.4	278	22 sec
2	Google Commons 1.0	1125	29 sec
3	Java JDK 1.1.7	216	65 sec
4	JFreeChart 0.9	1298	35 sec
5	Java JDK 1.1.7	992	45 sec
6	Java JDK 1.4.2	326	30 sec

RQ1:Cost-Effectiveness

BUG ID	Code Base	CUT SLOC	Time first fault
1	Apache Commons 2.4	278	22 sec
2	Google Commons 1.0	1125	29 sec
3	Java JDK 1.1.7	216	65 sec
4	JFreeChart 0.9	1298	35 sec
5	Java JDK 1.1.7	992	45 sec
6	Java JDK 1.4.2	326	30 sec

generate the first failing test code
and
trigger the first faulty interleaving

RQ1:Cost-Effectiveness

BUG ID	Code Base	CUT SLOC	Time first fault
1	Apache Commons 2.4	278	22 sec
2	Google Commons 1.0	1125	29 sec
3	Java JDK 1.1.7	216	65 sec
4	JFreeChart 0.9	1298	35 sec
5	Java JDK 1.1.7	992	45 sec
6	Java JDK 1.4.2	326	30 sec

generate the first failing test code
and
trigger the first faulty interleaving

For all subjects the first generated test manifested the bug

RQ2:Comparison with ConTeGen

AutoConTest

ConTeGen

BUG ID	Code Base	CUT SLOC	Time first fault
1	Apache Commons 2.4	278	22 sec
2	Google Commons 1.0	1125	29 sec
3	Java JDK 1.1.7	216	65 sec
4	JFreeChart 0.9	1298	35 sec
5	Java JDK 1.1.7	992	45 sec
6	Java JDK 1.4.2	326	30 sec

ConTeGen [Pradel et. al. PLDI 2012]
http://thread-safe.org/

- State-of-The-Art in Random based concurrent test code generation
- We used the same interleaving explorer of AutoConTest
- different random seeds best result
- Time-budget of one hour

RQ2:Comparison with ConTeGen

			AutoConTest	ConTeGen
BUG ID	**Code Base**	**CUT SLOC**	**Time first fault**	**Time first fault**
1	Apache Commons 2.4	278	22 sec	66 sec
2	Google Commons 1.0	1125	29 sec	*1hr
3	Java JDK 1.1.7	216	65 sec	1,014 sec
4	JFreeChart 0.9	1298	35 sec	156 sec
5	Java JDK 1.1.7	992	45 sec	*1hr
6	Java JDK 1.4.2	326	30 sec	2,254 sec

RQ2:Comparison with ConTeGen

BUG ID	Code Base	CUT SLOC	AutoConTest Coverage	ConTeGen Coverage
1	Apache Commons 2.4	278	19	91
2	Google Commons 1.0	1125	1	0
3	Java JDK 1.1.7	216	2	2
4	JFreeChart 0.9	1298	23	3
5	Java JDK 1.1.7	992	1	0
6	Java JDK 1.4.2	326	33	1

Coverage =
unique and feasible interleaving coverage requirements (atomic-set violations)

The same bug could lead to different atomic-set violations

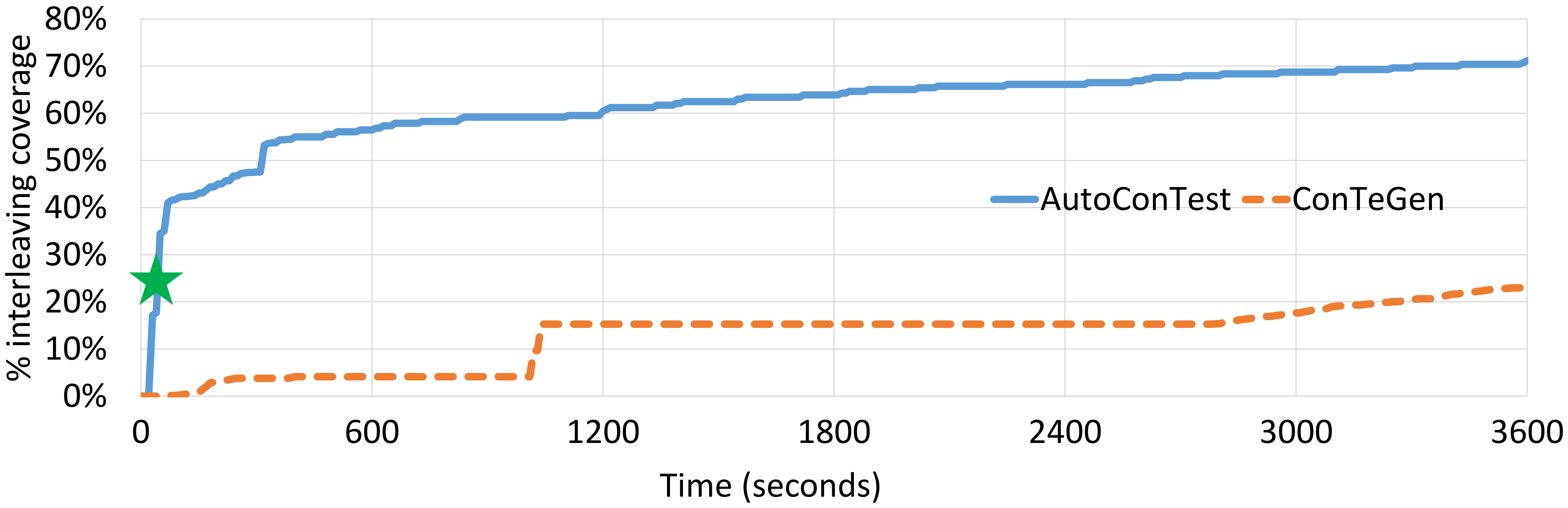
RQ2:Comparison with ConTeGen

BUG ID	Code Base	CUT SLOC	AutoConTest Coverage	ConTeGen Coverage
1	Apache Commons 2.4	278	19	91
2	Google Commons 1.0	1125	1	0
3	Java JDK 1.1.7	216	2	2
4	JFreeChart 0.9	1298	23	3
5	Java JDK 1.1.7	992	1	0
6	Java JDK 1.4.2	326	33	1

Coverage =
unique and feasible interleaving coverage requirements (atomic-set violations)

The same bug could lead to different atomic-set violations

RQ2:Coverage Comparison



On average (for all subjects), **AutoConTest** achieved in **less than 40** seconds the same percentage of coverage achieved by **ConTeGen** in **one hour**.

RQ2:Test Code Size Comparison

			AutoConTest		ConTeGen	
BUG ID	Code Base	CUT SLOC	# tests	Test code SLOC	# tests	Test code SLOC
1	Apache Commons 2.4	278	7	2,157	110	1,742
2	Google Commons 1.0	1125	3	19	130	1,898
3	Java JDK 1.1.7	216	17	1,437	185	1,232
4	JFreeChart 0.9	1298	1	114	99	1,351
5	Java JDK 1.1.7	992	3	105	230	3,161
6	Java JDK 1.4.2	326	1	104	167	2,729

RQ2:Test Code Size Comparison

			AutoConTest		ConTeGen	
BUG ID	**Code Base**	**CUT SLOC**	**# tests**	**Test code SLOC**	**# tests**	**Test code SLOC**
1	Apache Commons 2.4	278	7	2,157	110	1,742
2	Google Commons 1.0	1125	3	19	130	1,898
3	Java JDK 1.1.7	216	17	1,437	185	1,232
4	JFreeChart 0.9	1298	1	114	99	1,351
5	Java JDK 1.1.7	992	3	105	230	3,161
6	Java JDK 1.4.2	326	1	104	167	2,729

RQ2:Test Code Size Comparison

			AutoConTest		ConTeGen	
BUG ID	Code Base	CUT SLOC	# tests	Test code SLOC	# tests	Test code SLOC
1	Apache Commons 2.4	278	7	2,157	110	1,742
2	Google Commons 1.0	1125	3	19	130	1,898
3	Java JDK 1.1.7	216	17	1,437	185	1,232
4	JFreeChart 0.9	1298	1	114	99	1,351
5	Java JDK 1.1.7	992	3	105	230	3,161
6	Java JDK 1.4.2	326	1	104	167	2,729

Or generated test suites are more effective than much larger test suites generated randomly

RQ3:Redundancy-based Pruning Strategies

	ENABLED		DISABLED	
BUG ID	Time (ms)	# unique method calls that increase coverage	Time (ms)	# unique method calls that increase coverage
1	1,598	25	1 hr (time-out)	25
2	1,741	6	1 hr (time-out)	6
3	1,931	23	1 hr (time-out)	24
4	7,098	56	1 hr (time-out)	56
5	2,911	24	1 hr (time-out)	24
6	2,866	44	1 hr (time-out)	44

Optimal sequence at the first iteration

saturation based stopping criterion $k = 3$

RQ3:Redundancy-based Pruning Strategies

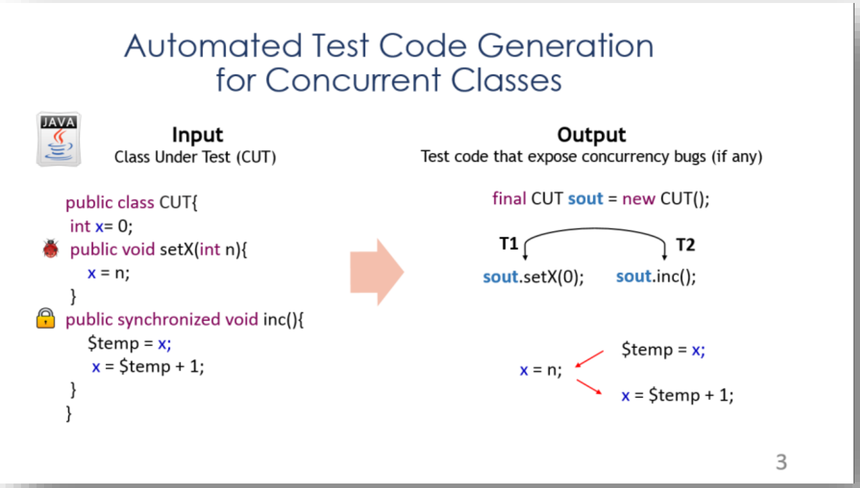
	ENABLED		DISABLED	
BUG ID	Time (ms)	# unique method calls that increase coverage	Time (ms)	# unique method calls that increase coverage
1	1,598	25	1 hr (time-out)	25
2	1,741	6	1 hr (time-out)	6
3	1,931	23	1 hr (time-out)	24
4	7,098	56	1 hr (time-out)	56
5	2,911	24	1 hr (time-out)	24
6	2,866	44	1 hr (time-out)	44

Optimal sequence at the first iteration

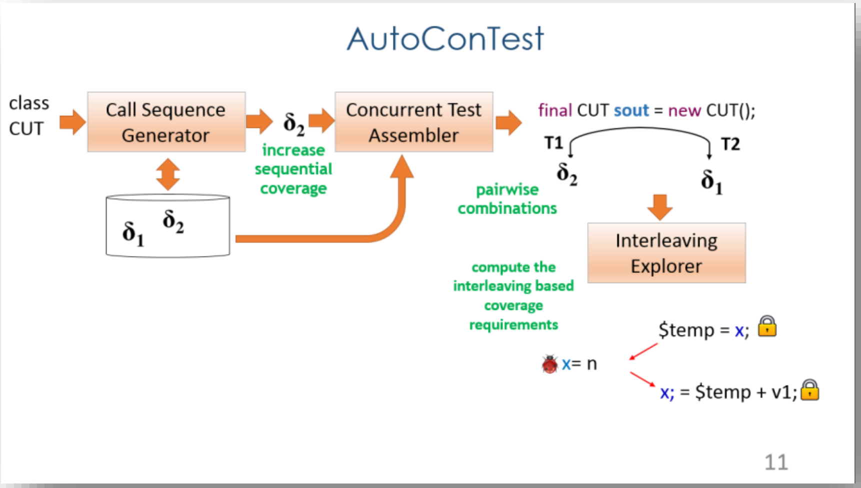
saturation based stopping criterion k = 3

The pruning strategies detected optimal sequence more than **1530× faster**, in only one case the solution was sub-optimal.

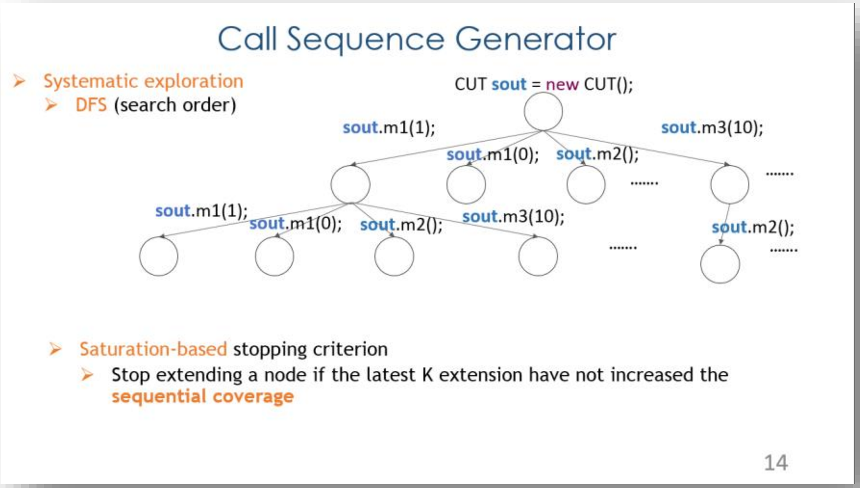
Conclusion



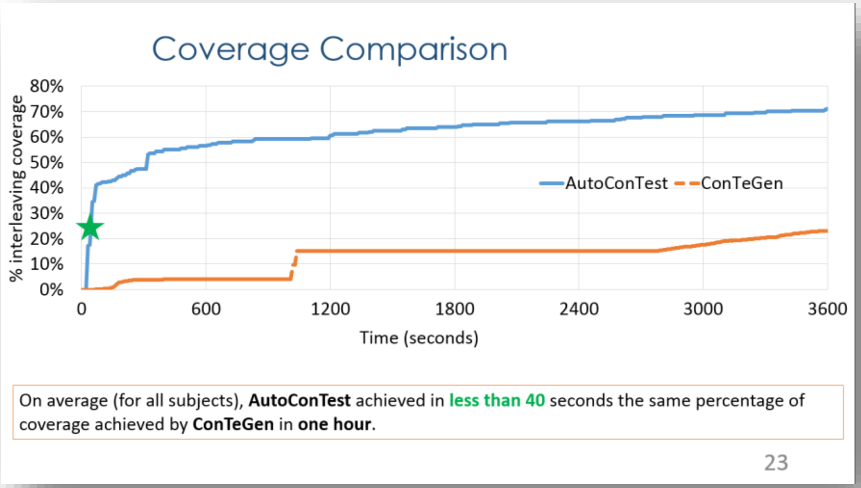
Automated Test Code Generation for Concurrent Classes
JAVA
Input
Class Under Test (CUT)
public class CUT{
int x= 0;
public void setX(int n){
x = n;
}
public synchronized void inc(){
$temp = x;
x = $temp + 1;
}
}
Output
Test code that expose concurrency bugs (if any)
final CUT sout = new CUT();
T1
T2
sout.setX(0);
sout.inc();
$temp = x;
x = n;
x = $temp + 1;
3



AutoConTest
class CUT
Call Sequence Generator
δ2
increase sequential coverage
Concurrent Test Assembler
δ1 δ2
final CUT sout = new CUT();
T1
T2
δ2
δ1
pairwise combinations
Interleaving Explorer
compute the interleaving based coverage requirements
$temp = x;
x= n
x; = $temp + v1;
11



Call Sequence Generator
Systematic exploration
DFS (search order)
CUT sout = new CUT();
sout.m1(1);
sout.m3(10);
sout.m1(0);
sout.m2();
sout.m1(1);
sout.m1(0);
sout.m2();
sout.m3(10);
sout.m2();
Saturation-based stopping criterion
Stop extending a node if the latest K extension have not increased the sequential coverage
14



Coverage Comparison
% interleaving coverage
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
600
1200
1800
2400
3000
3600
Time (seconds)
AutoConTest
ConTeGen
On average (for all subjects), AutoConTest achieved in less than 40 seconds the same percentage of coverage achieved by ConTeGen in one hour.
23