Using Conditional Mutation to Increase the Efficiency of Mutation Analysis

René Just[1] & Gregory M. Kapfhammer[2] & Franz Schweiggert[1]

[1]Ulm University, Germany

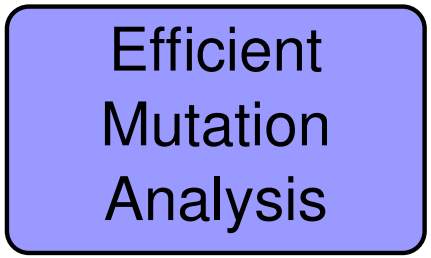
[2]Allegheny College, USA

6th International Workshop on the Automation of Software Test
Waikiki, Honolulu, Hawaii, USA
May 23 - 24, 2011

ulm university universität uulm

ALLEGHENY COLLEGE

Overview of the Presentation

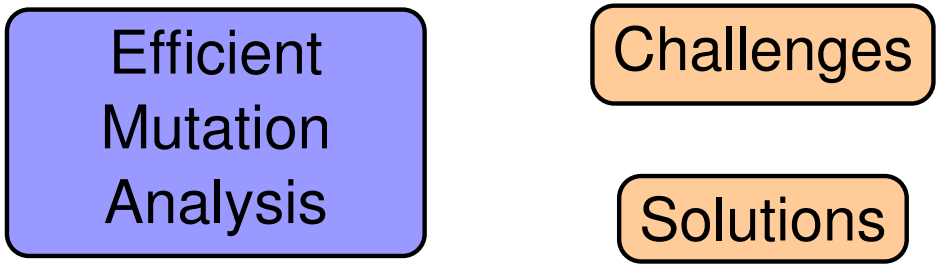


Overview of the Presentation

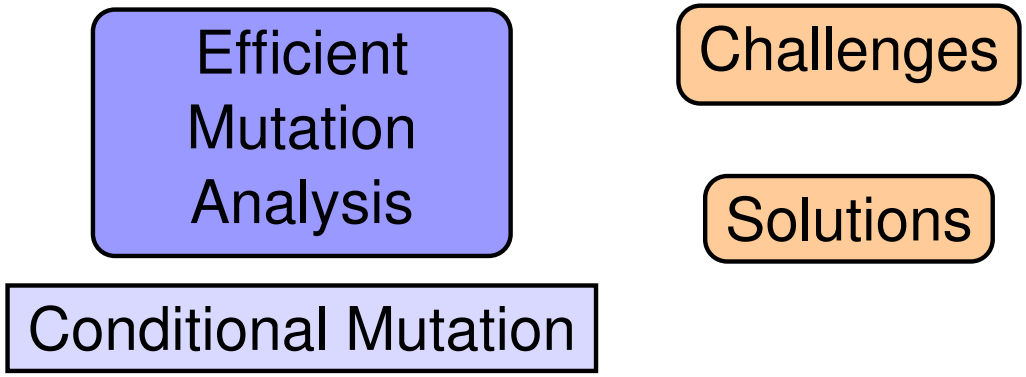
Efficient Mutation Analysis

Challenges

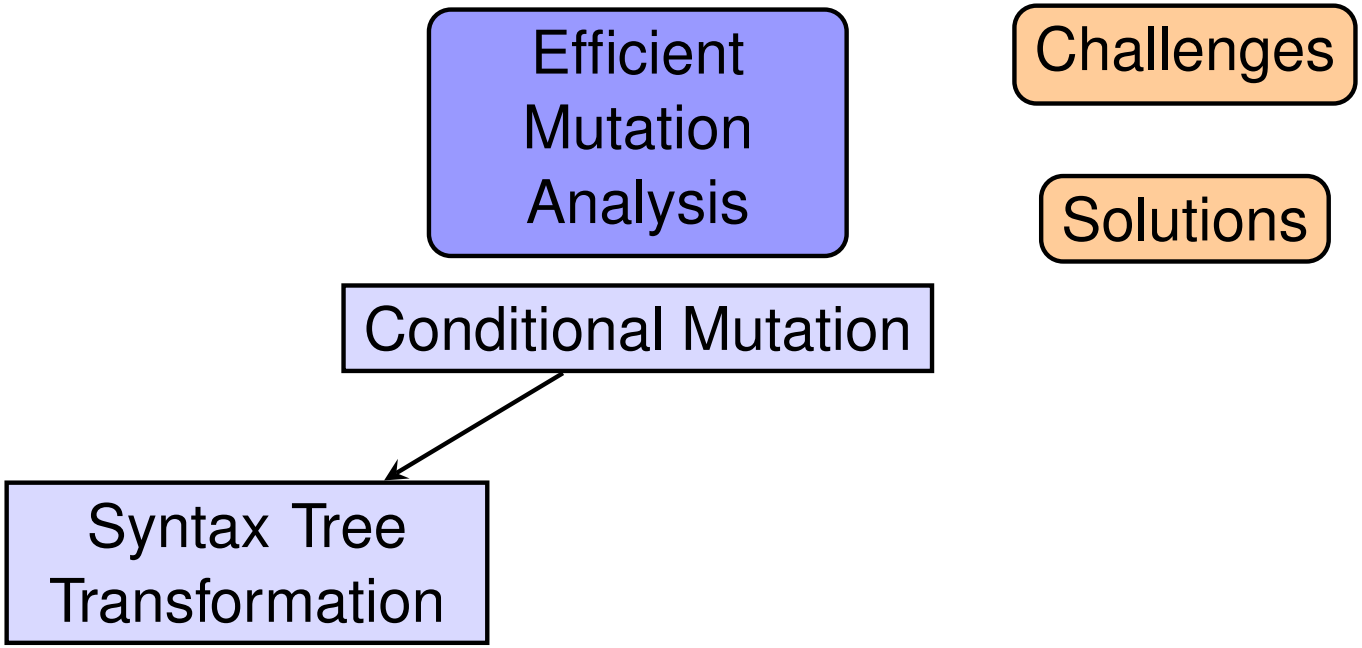
Overview of the Presentation



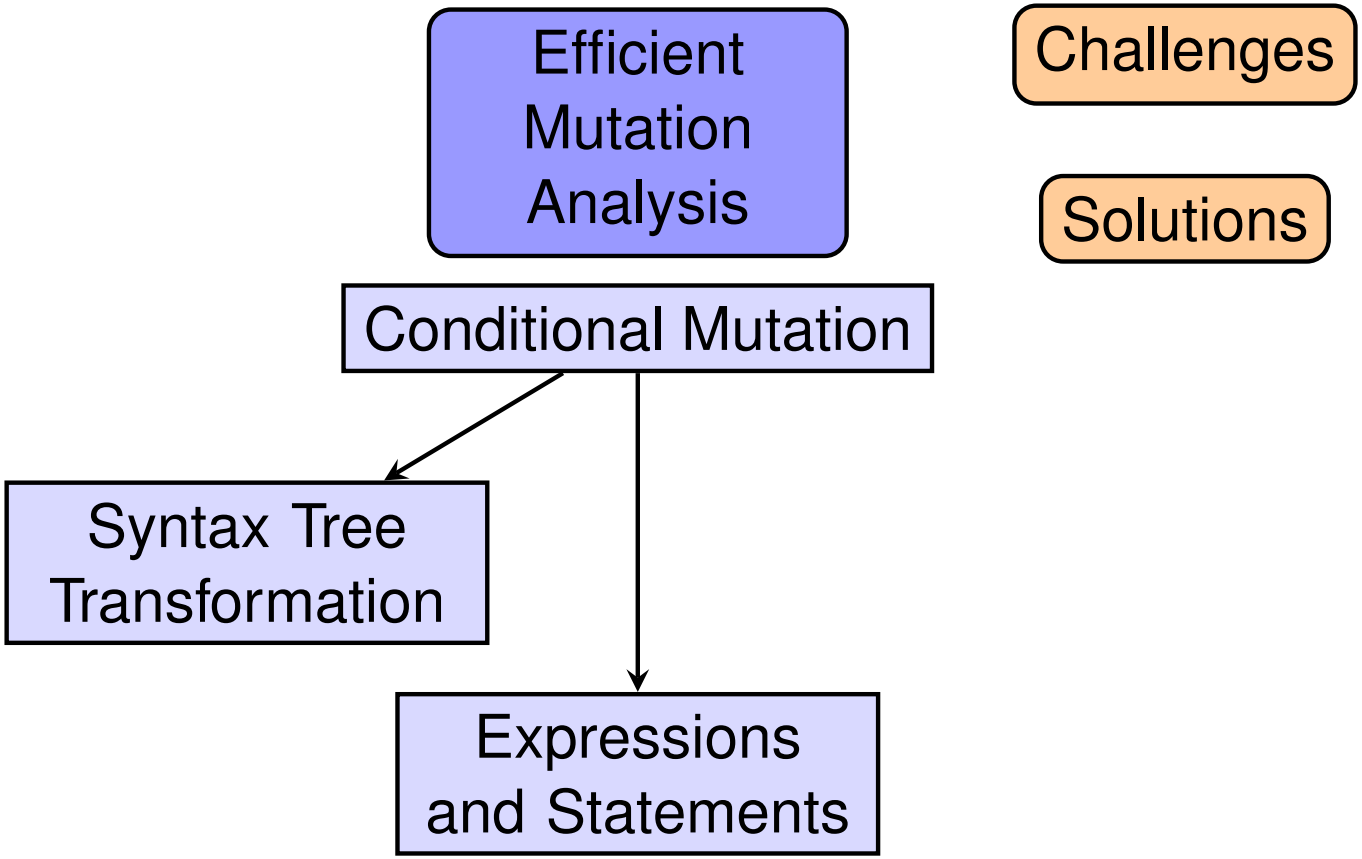
Overview of the Presentation



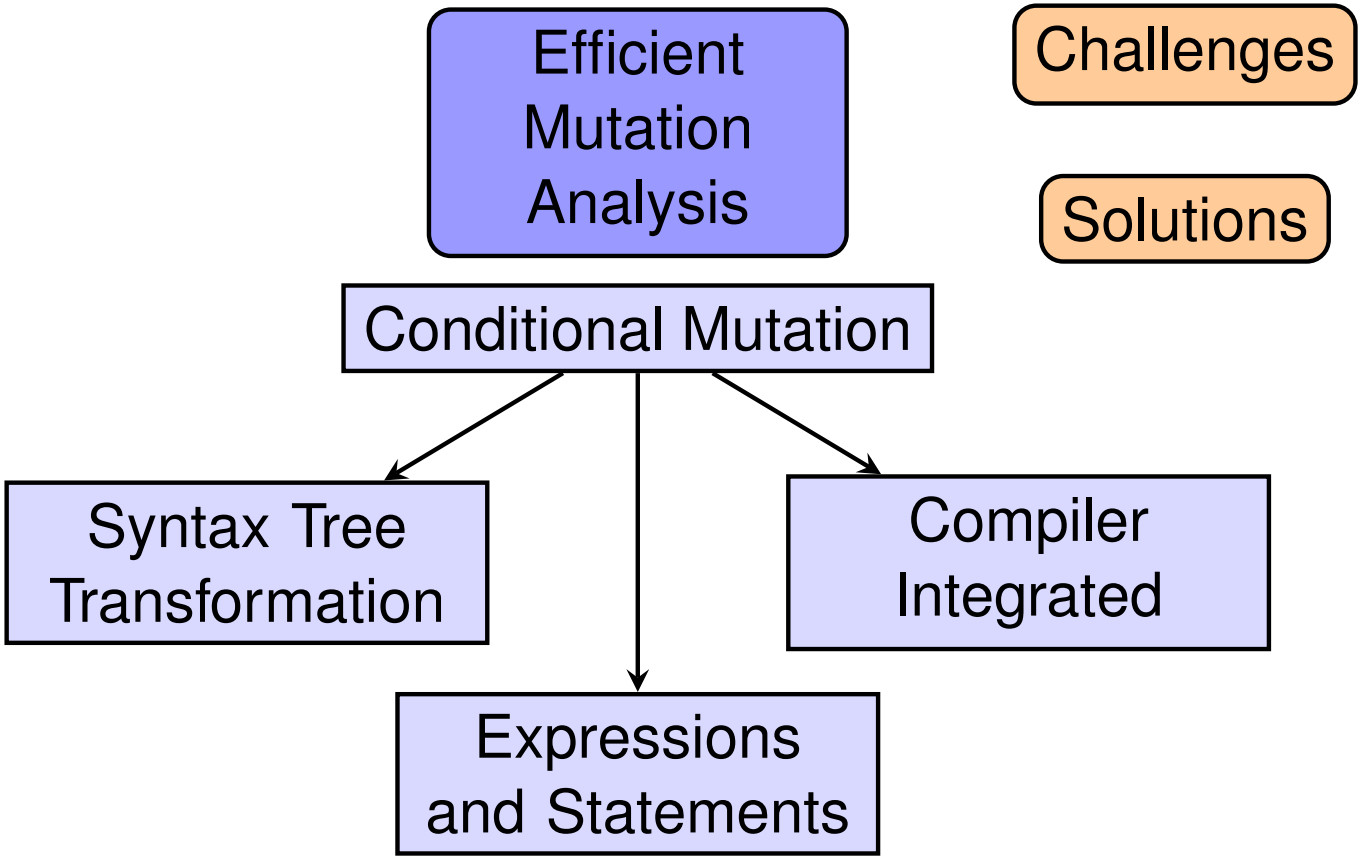
Overview of the Presentation



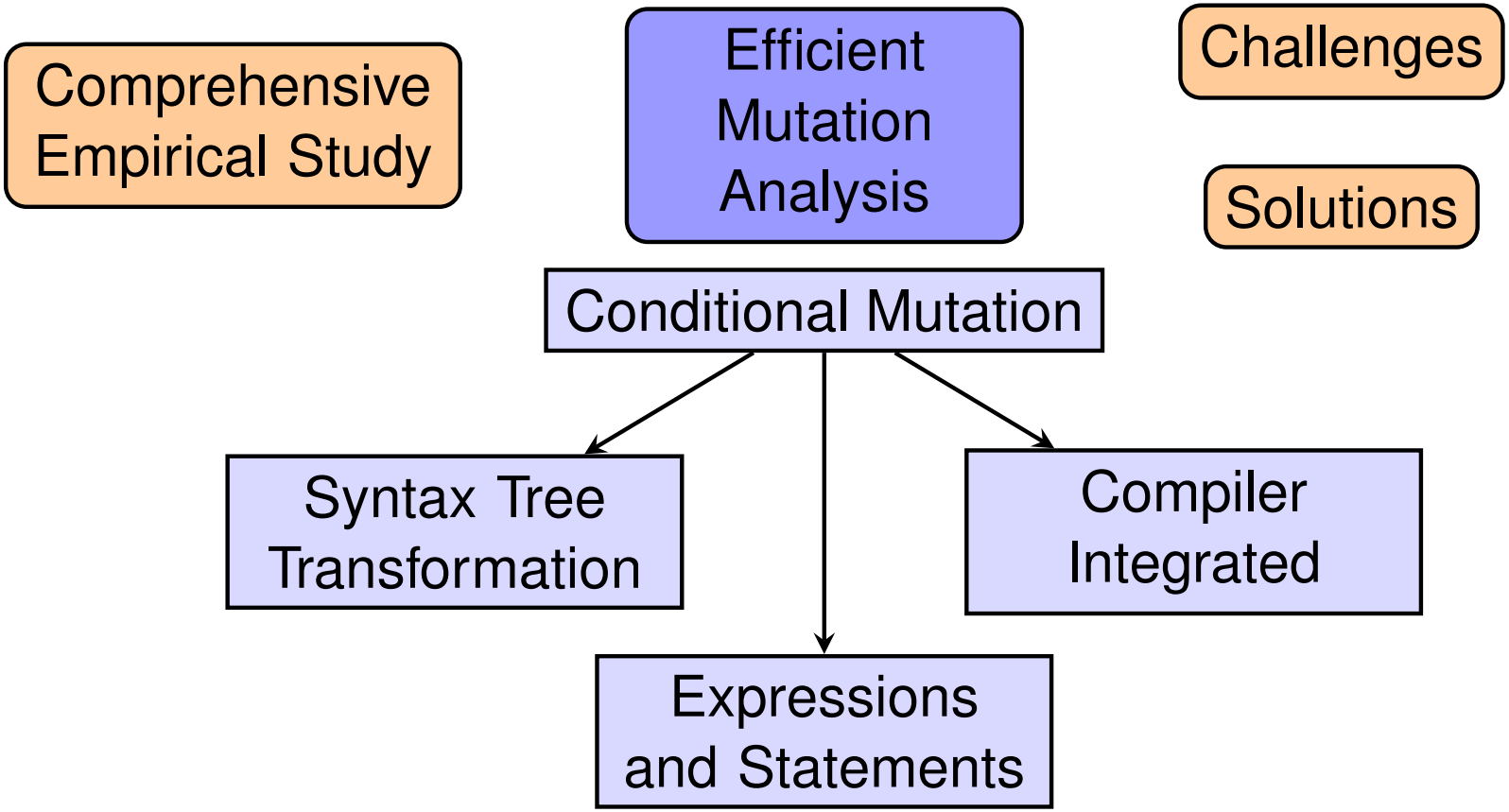
Overview of the Presentation



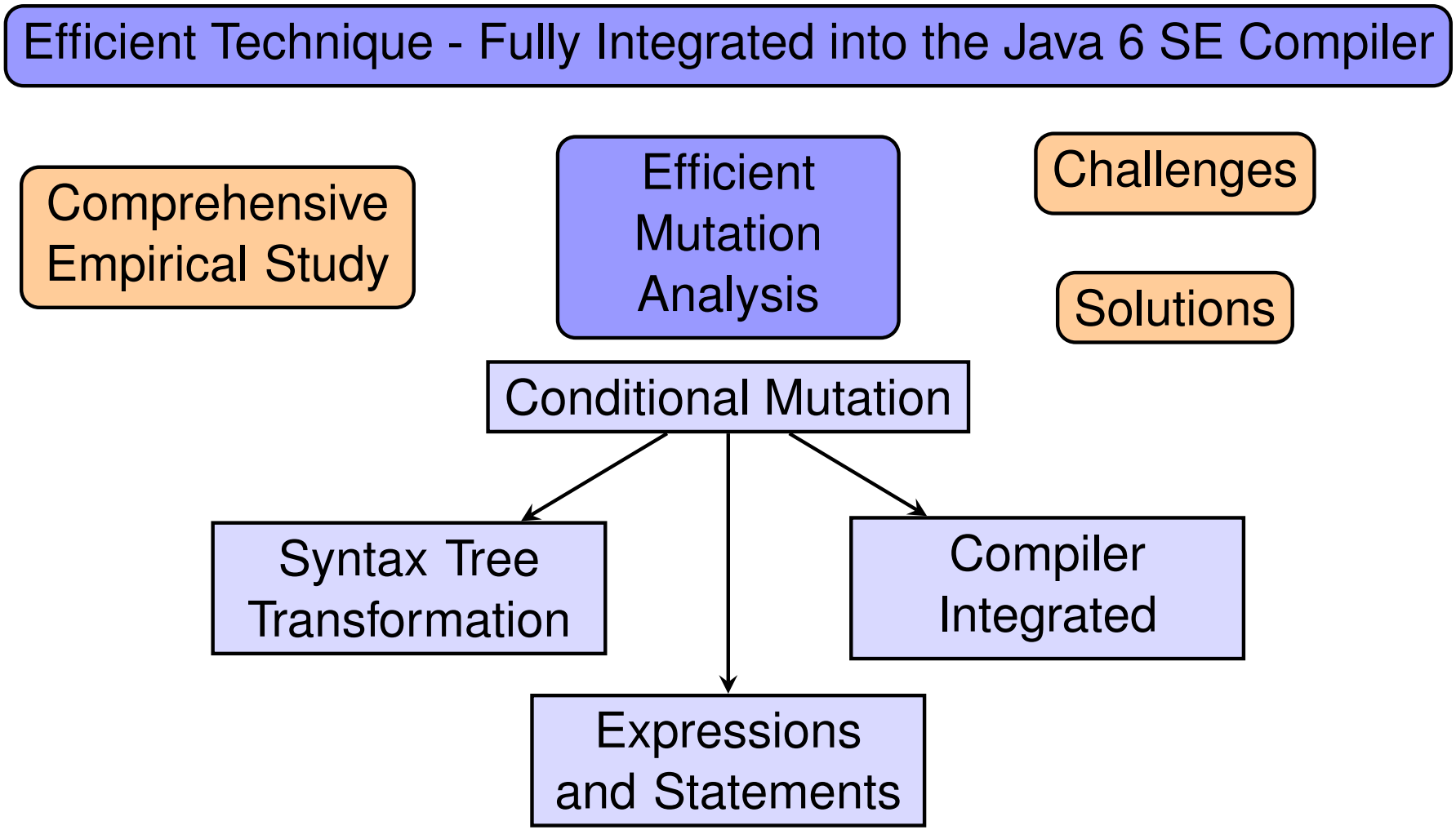
Overview of the Presentation



Overview of the Presentation



Overview of the Presentation



Overview of Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

Overview of Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

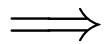
Methodically inject small syntactical faults into the program under test

Overview of Mutation Analysis

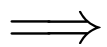
```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

Overview of Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```



- `y = a - x;`
- `y = a + x;`
- `y = a / x;`



- `if(b < a)`
- `if(b != a)`
- `if(b == a)`

Overview of Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

Unbiased and powerful method for assessing oracles and input values

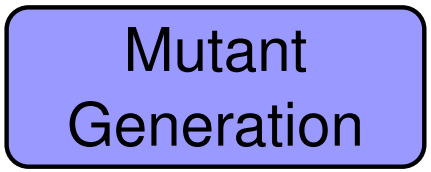
Overview of Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

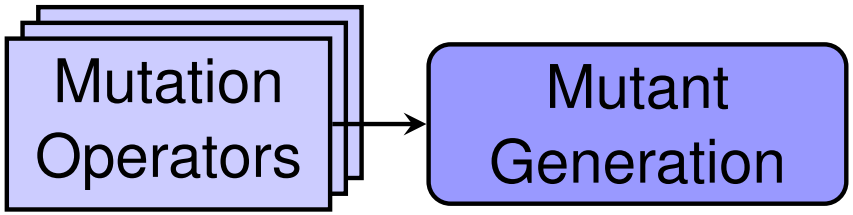
Unbiased and powerful method for assessing oracles and input values

Useful method for fault seeding during the empirical study of testing techniques

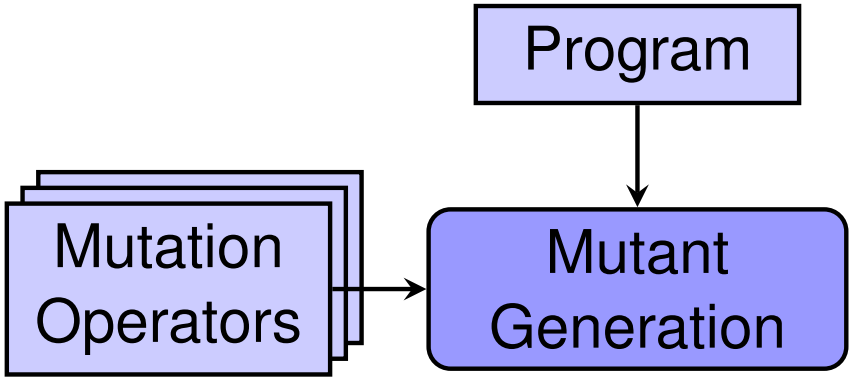
Mutation Analysis Challenges



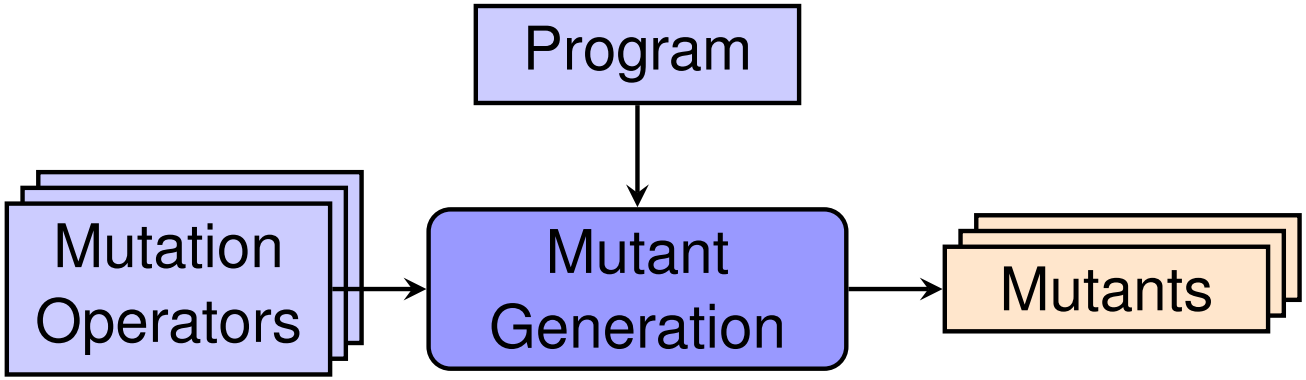
Mutation Analysis Challenges



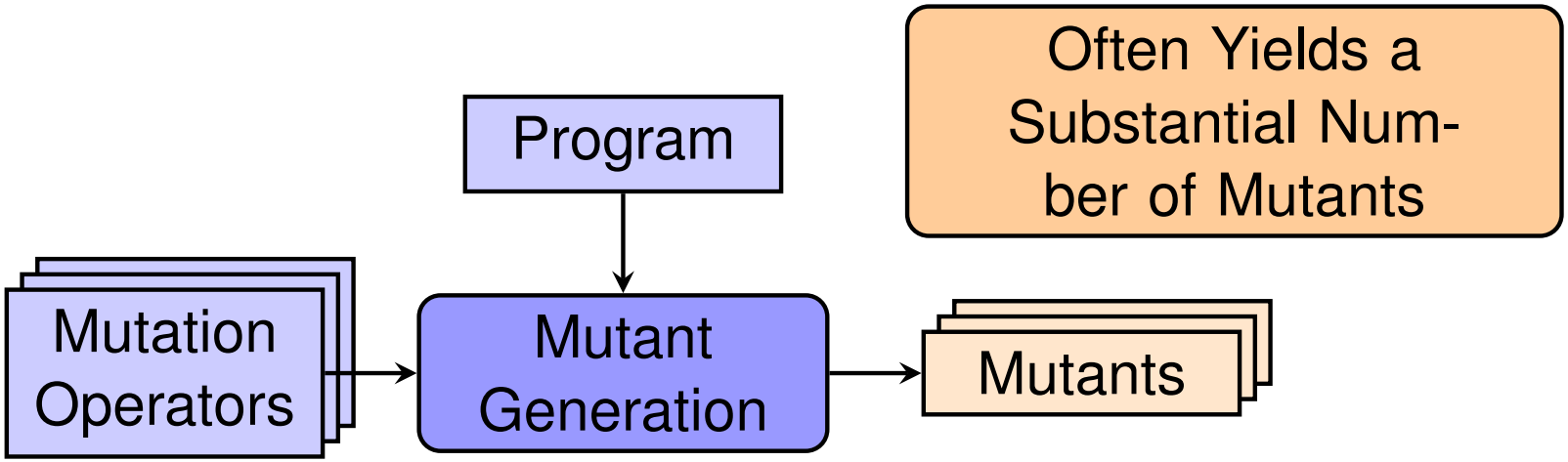
Mutation Analysis Challenges



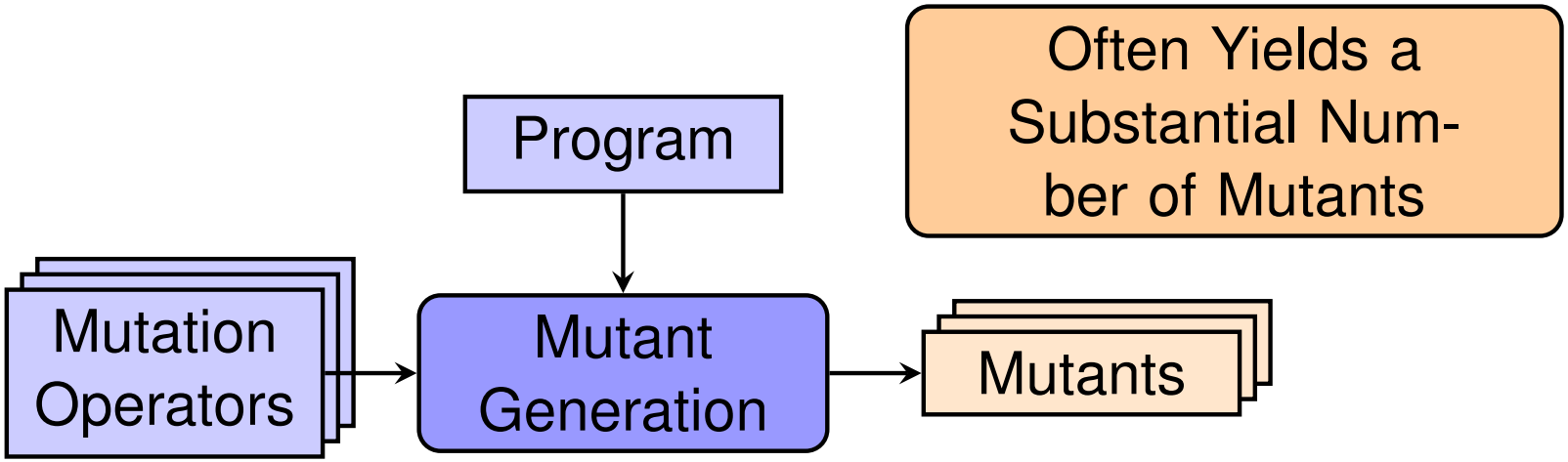
Mutation Analysis Challenges



Mutation Analysis Challenges

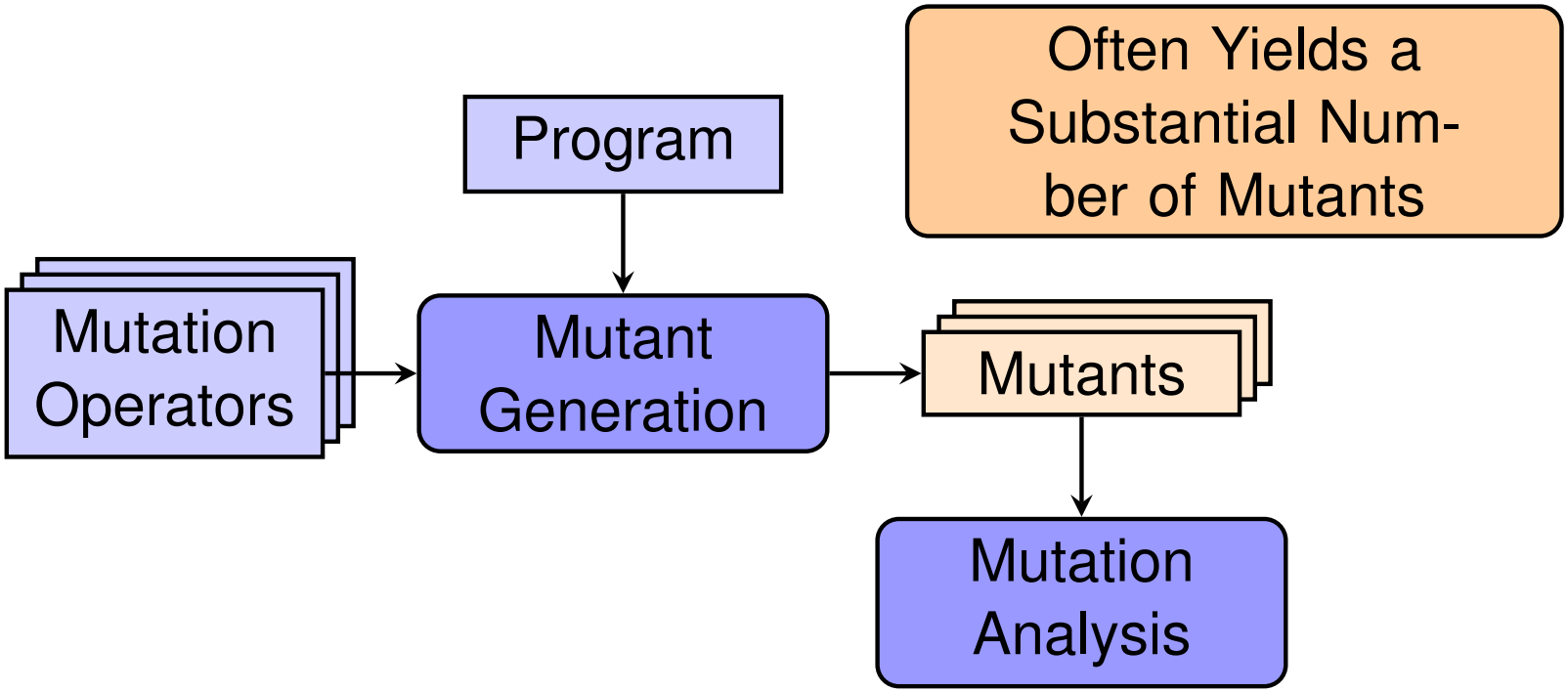


Mutation Analysis Challenges



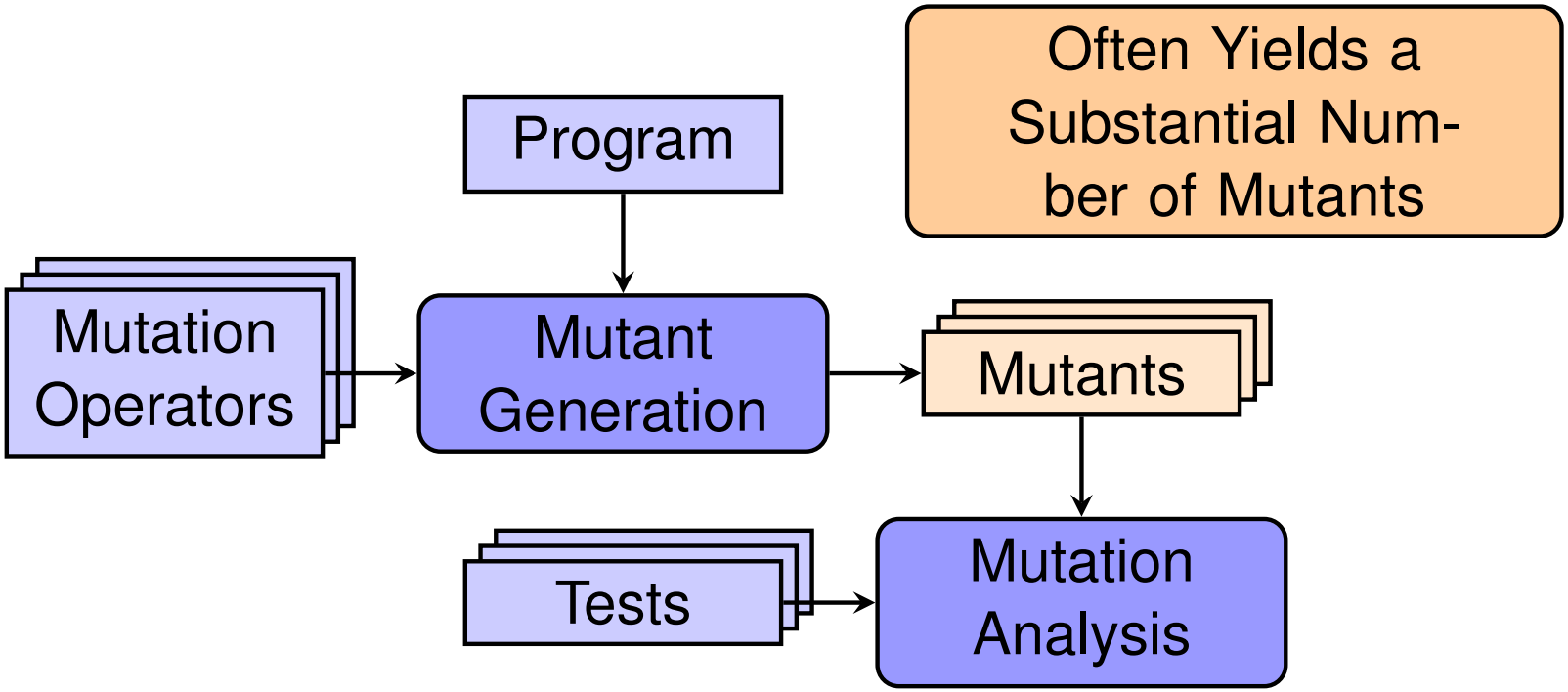
High Time Overhead for Generation

Mutation Analysis Challenges



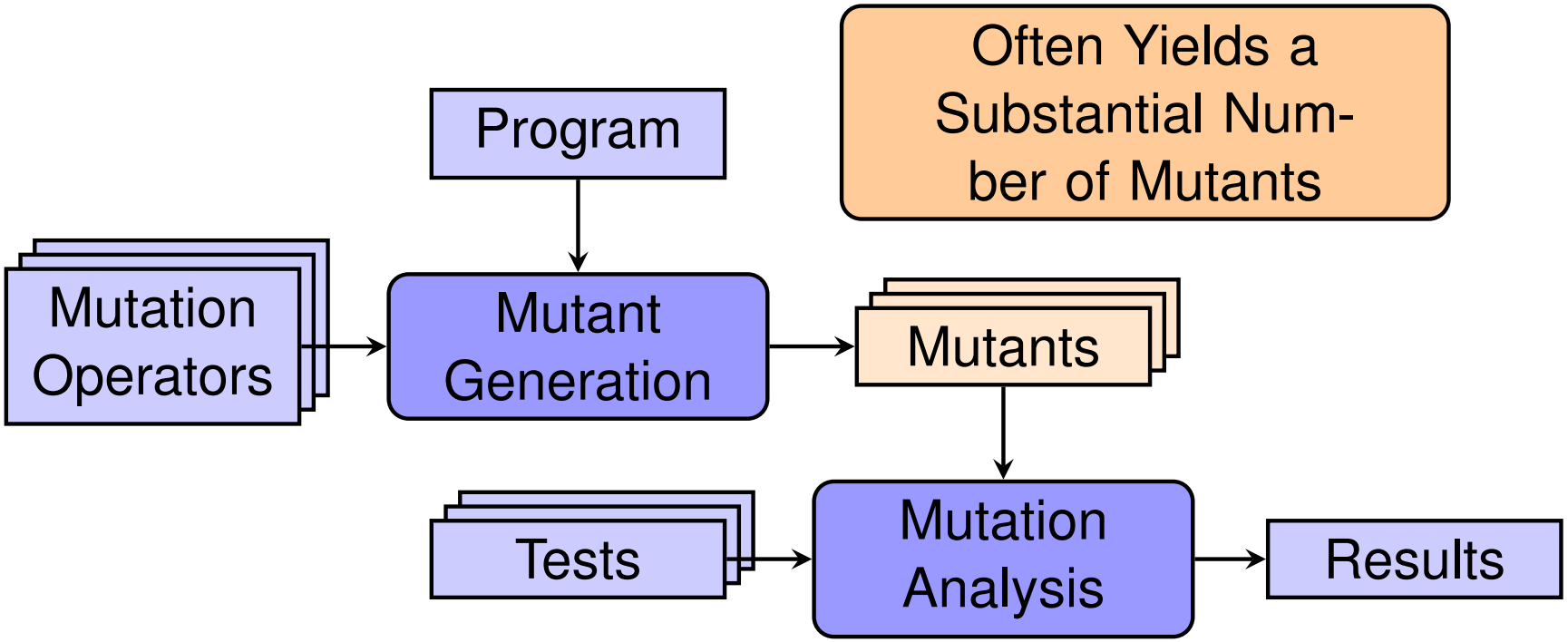
High Time Overhead for Generation

Mutation Analysis Challenges

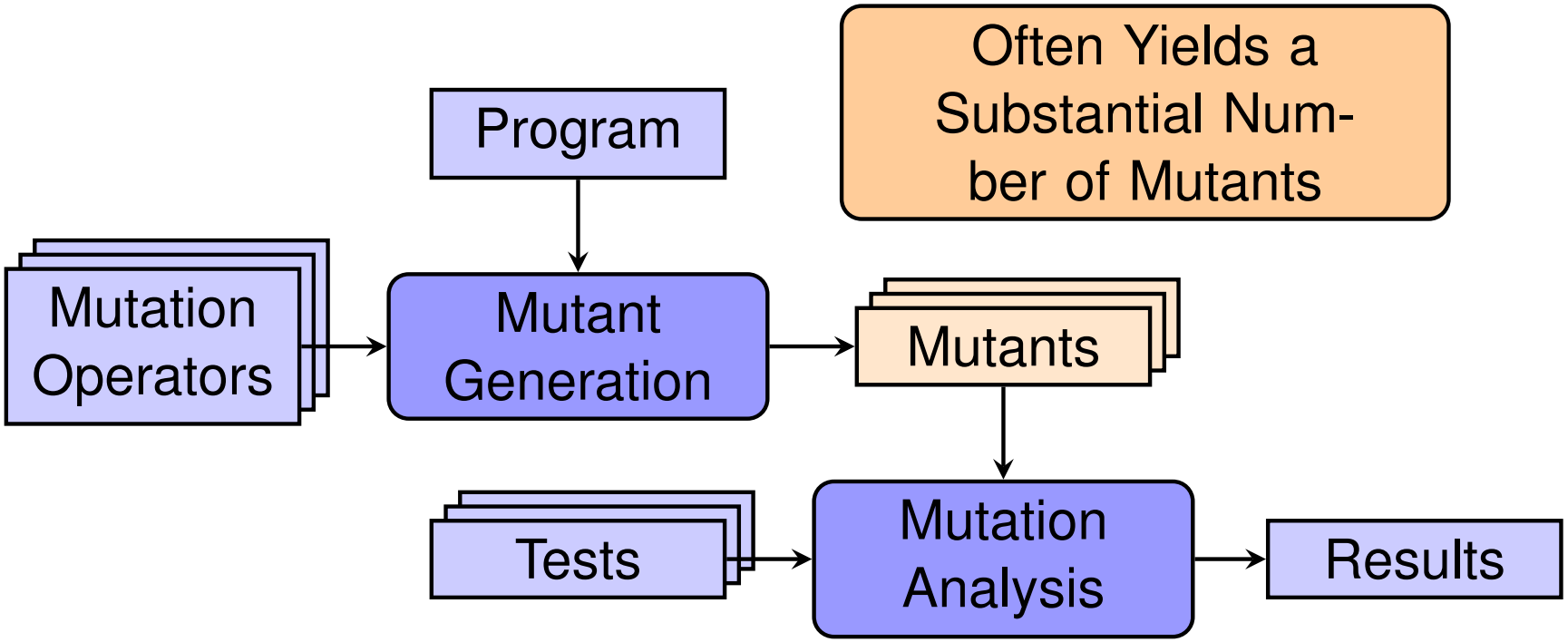


High Time Overhead for Generation

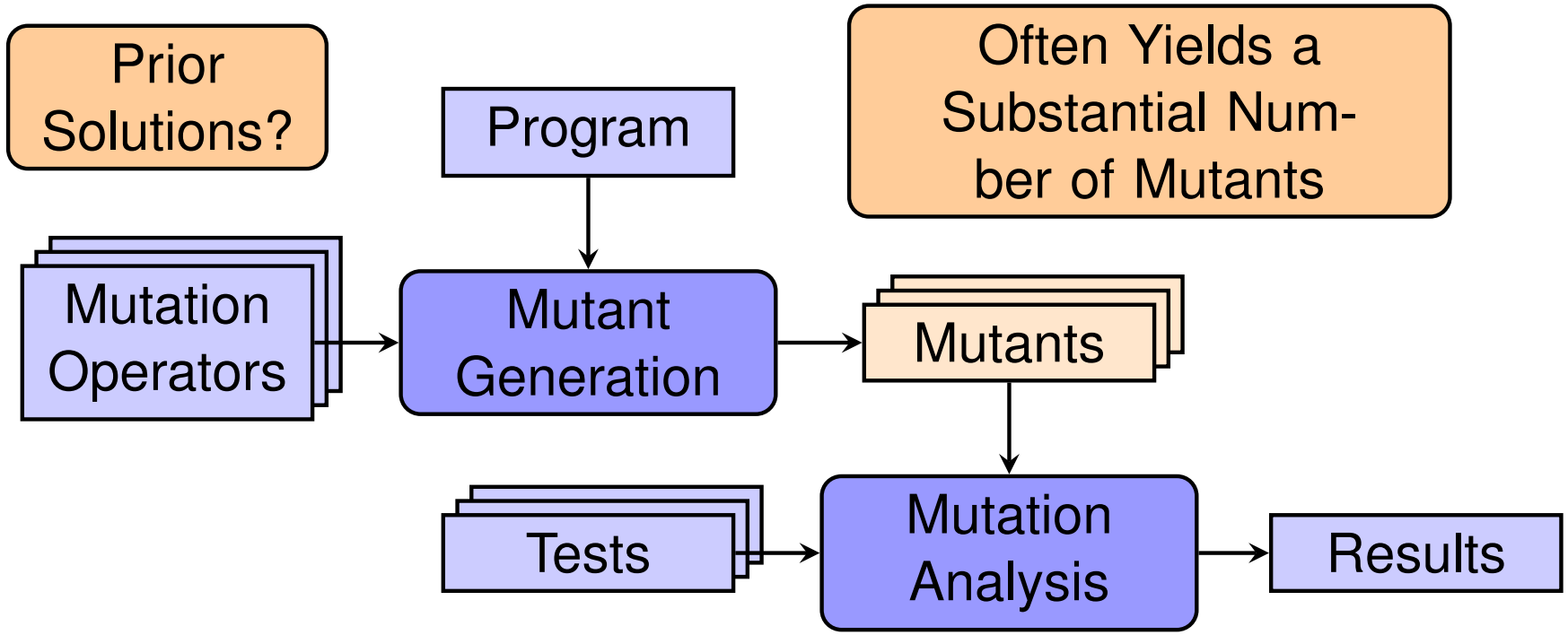
Mutation Analysis Challenges



Mutation Analysis Challenges



Mutation Analysis Challenges



Prior Work in Mutation Analysis

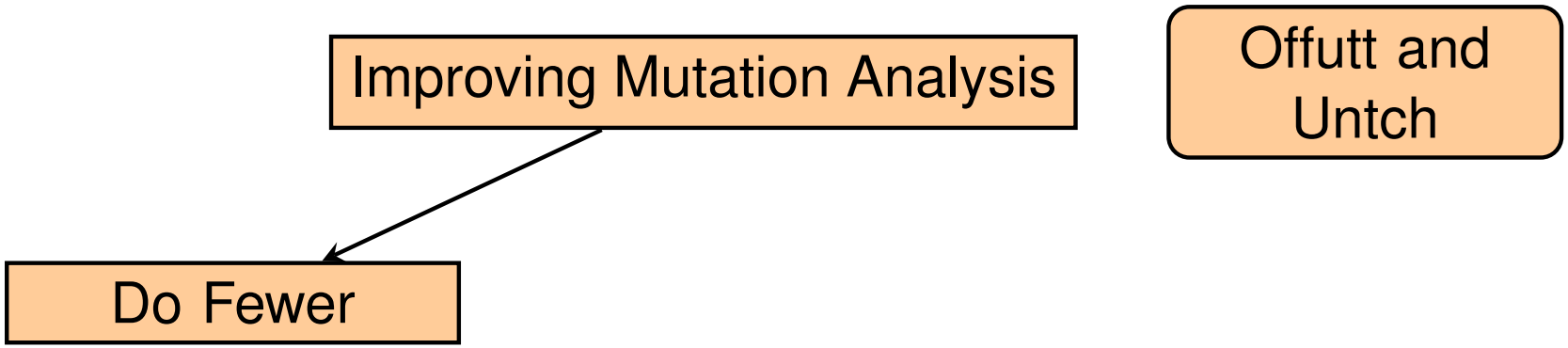
Improving Mutation Analysis

Prior Work in Mutation Analysis

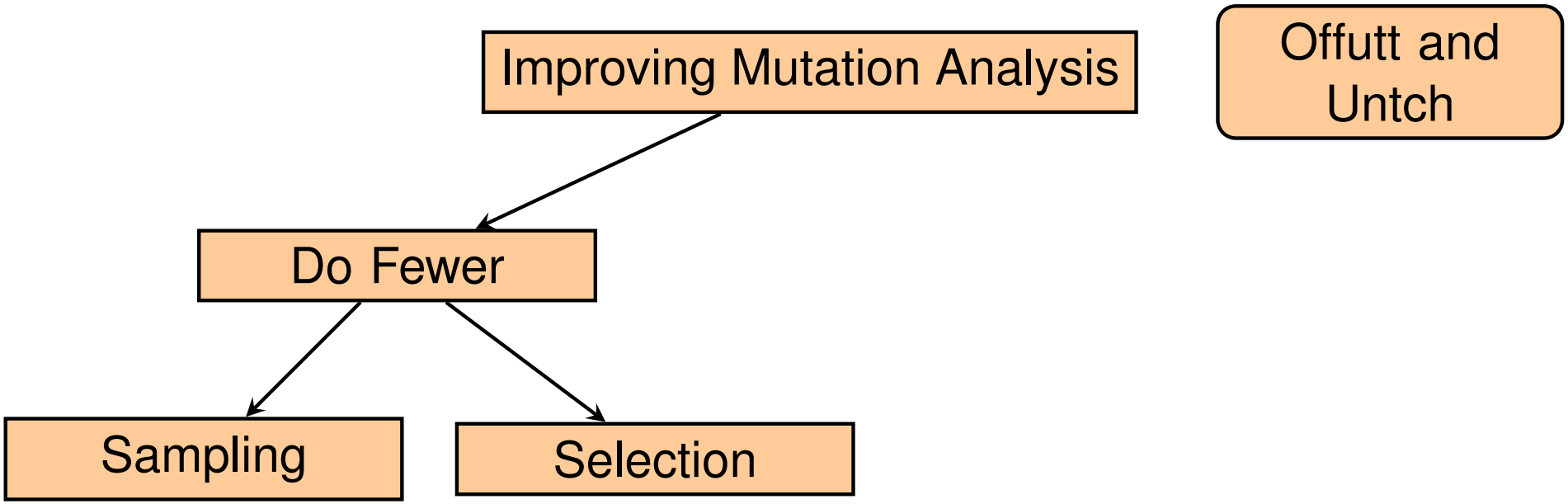
Improving Mutation Analysis

Offutt and Untch

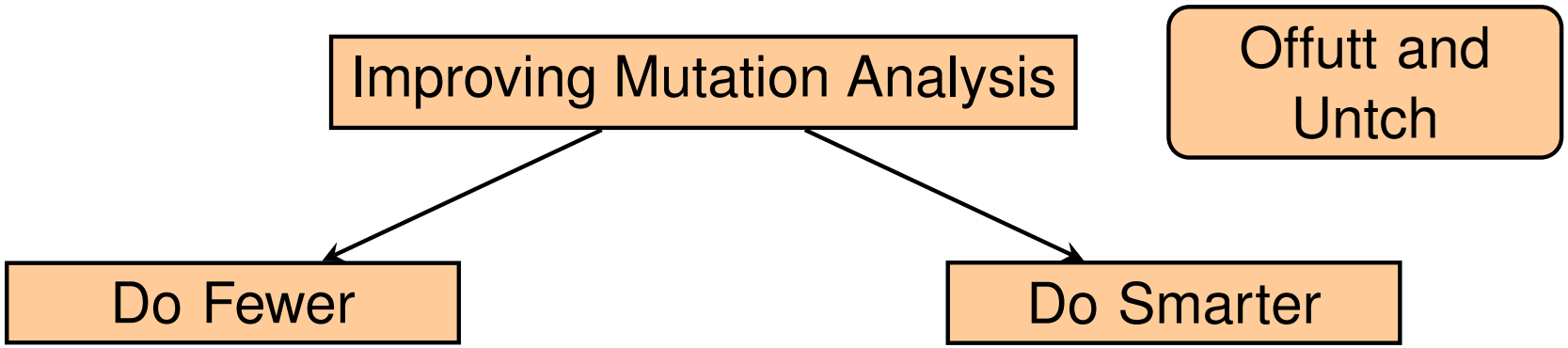
Prior Work in Mutation Analysis



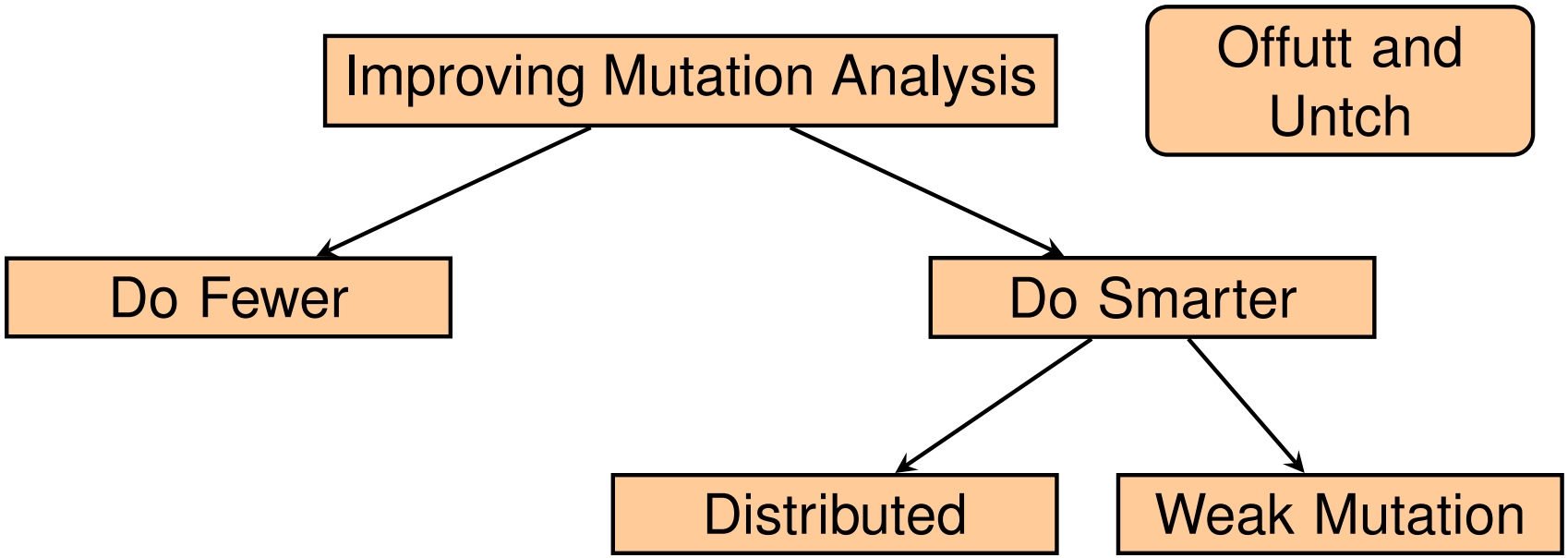
Prior Work in Mutation Analysis



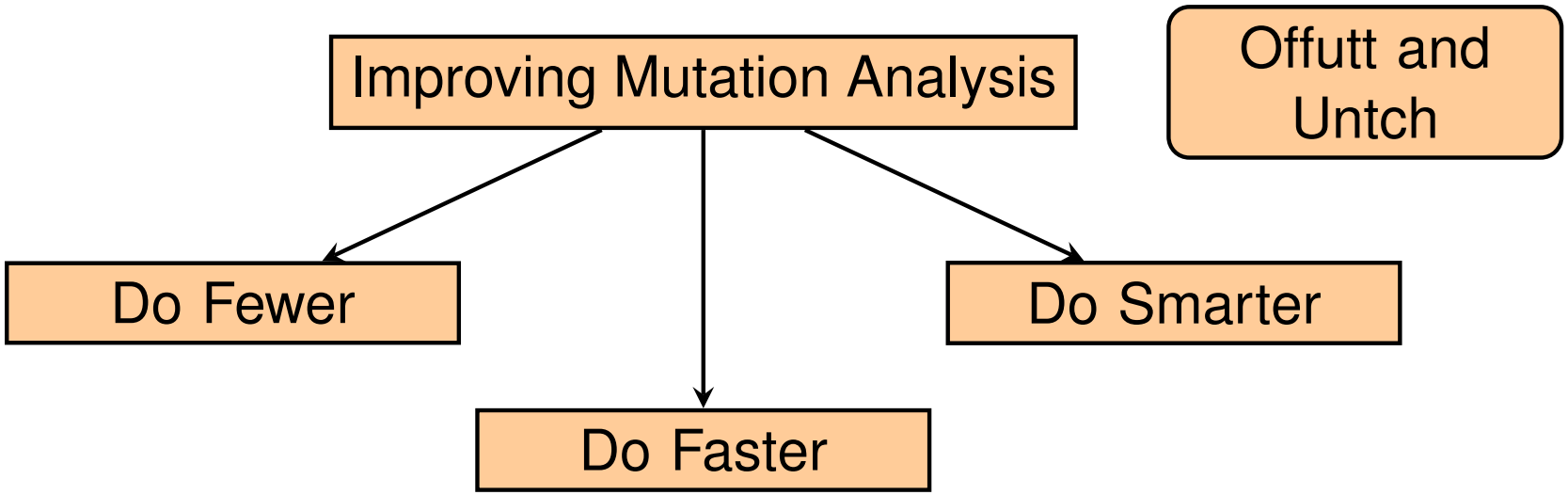
Prior Work in Mutation Analysis



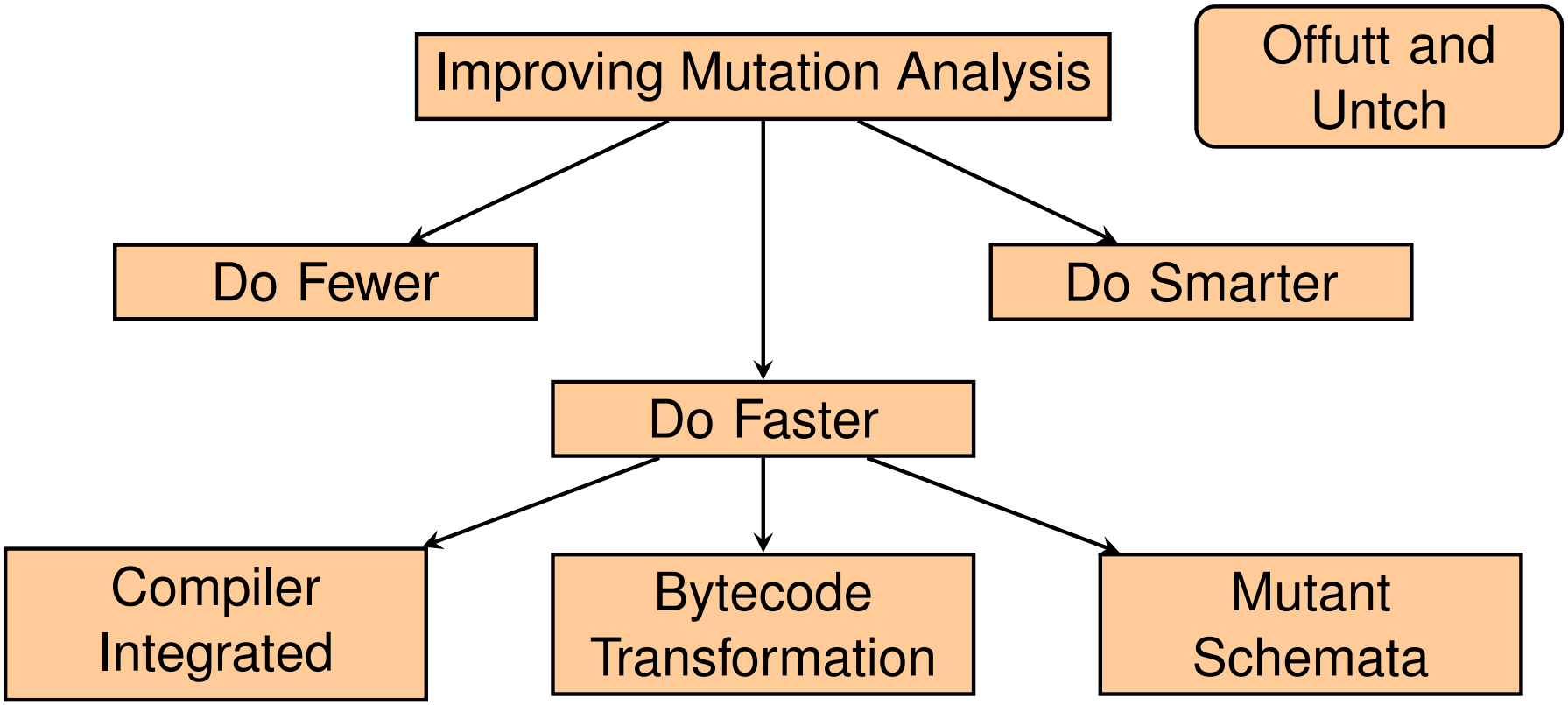
Prior Work in Mutation Analysis



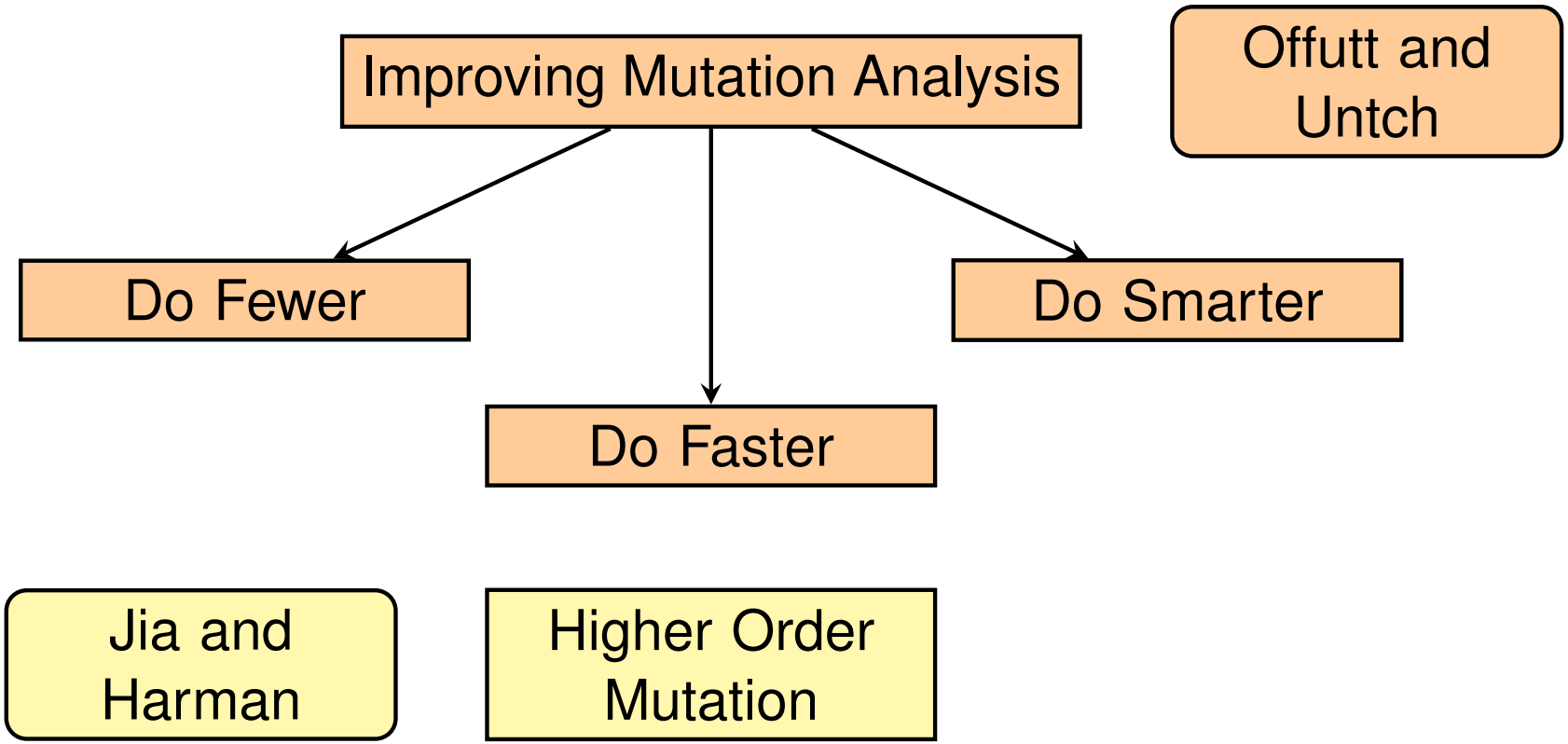
Prior Work in Mutation Analysis



Prior Work in Mutation Analysis



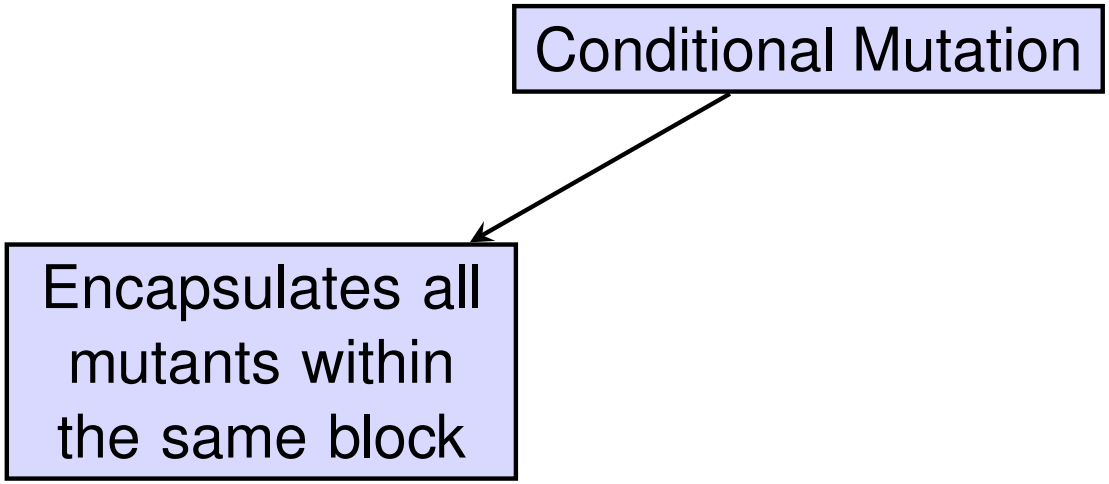
Prior Work in Mutation Analysis



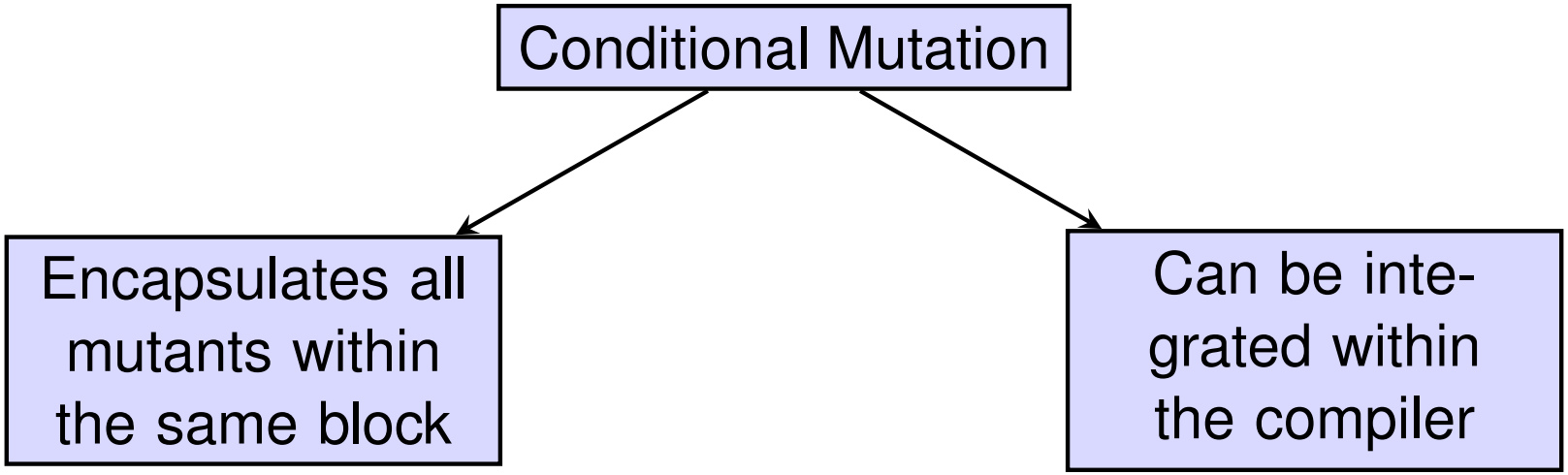
Conditional Mutation

Conditional Mutation

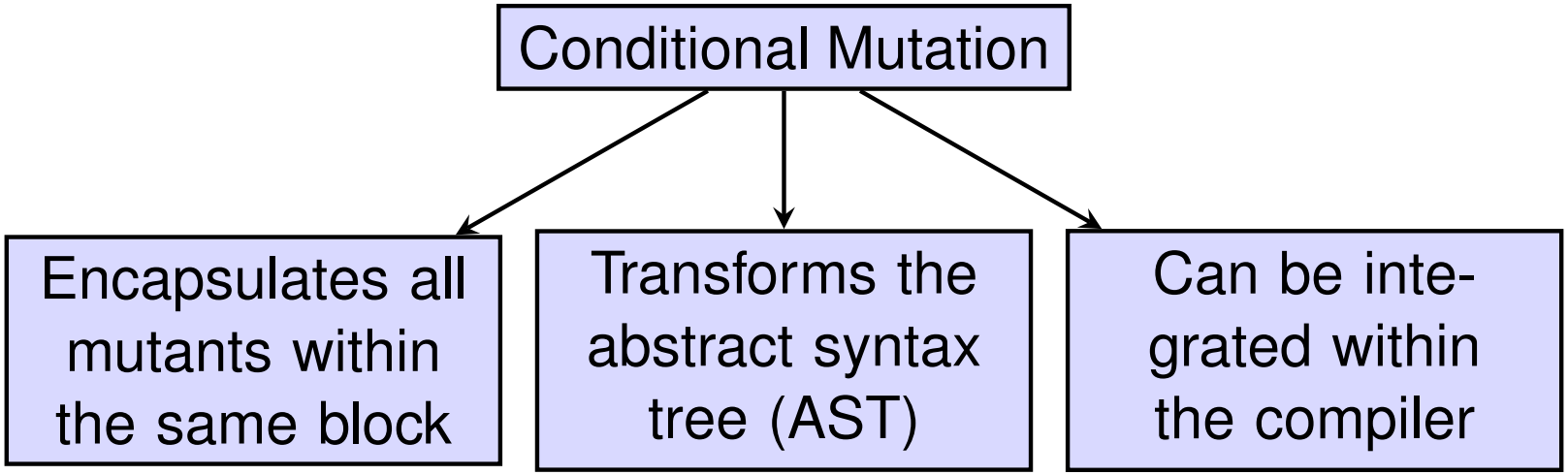
Conditional Mutation



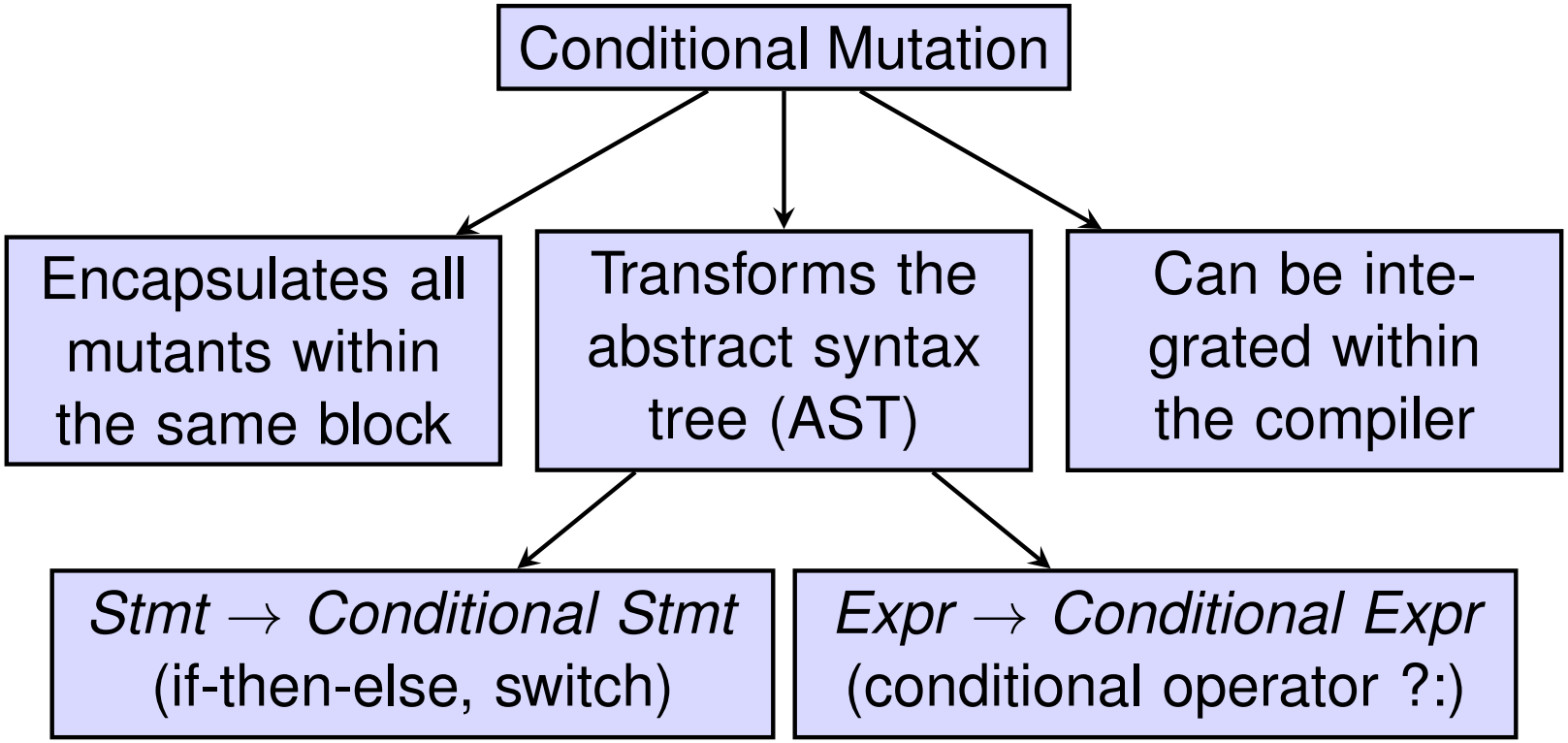
Conditional Mutation



Conditional Mutation



Conditional Mutation

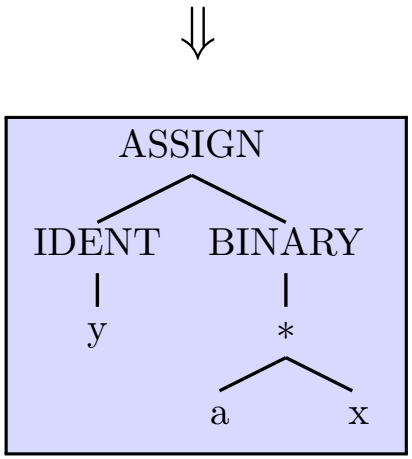


Working Example

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

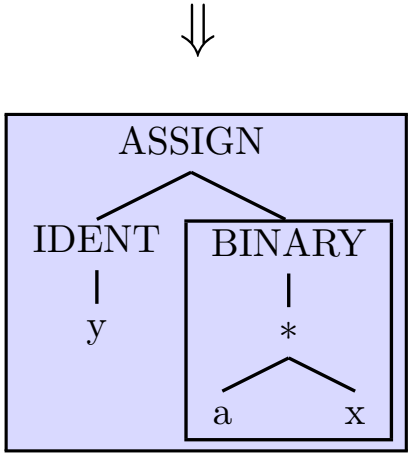
Working Example

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```



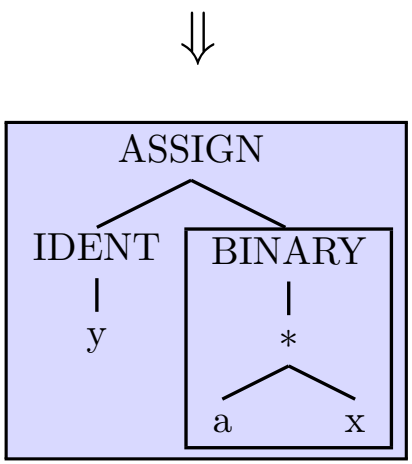
Working Example

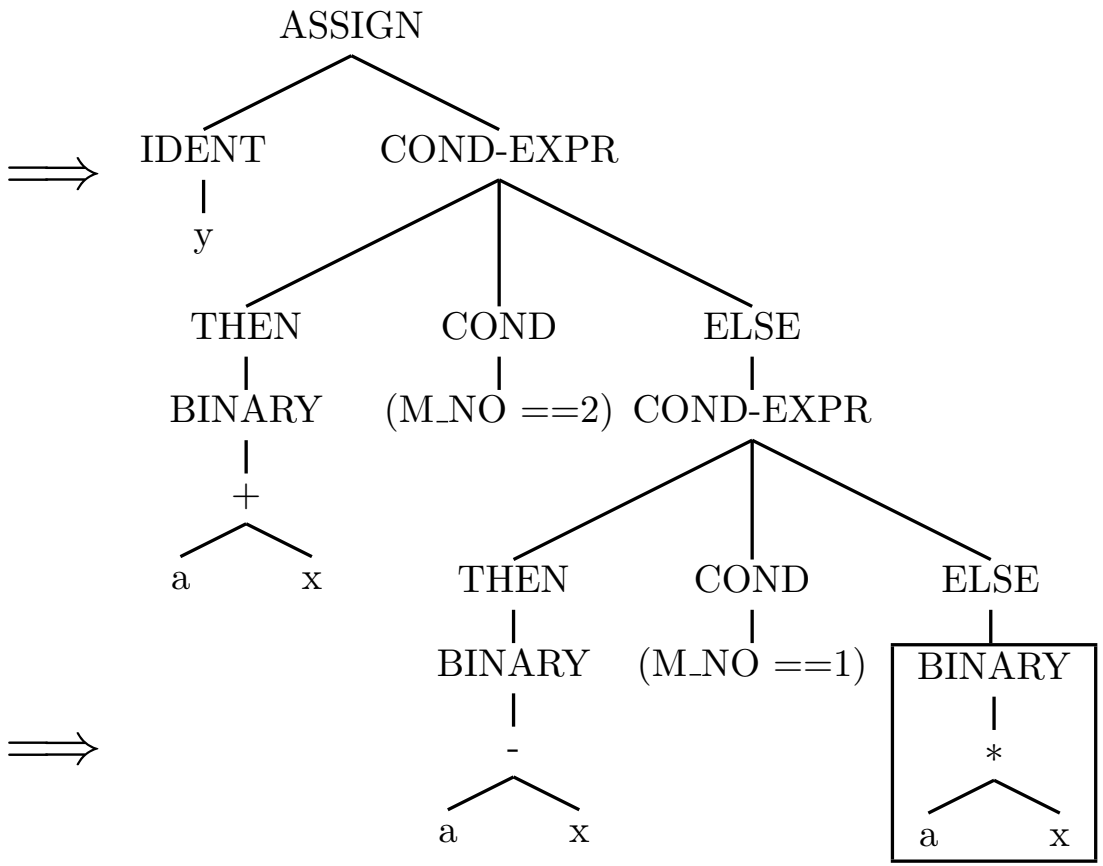
```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```



Working Example

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```





Conditional Mutation Algorithm

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

1. Define mutation operators
   $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Conditional Mutation Algorithm

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

1. Define mutation operators

   $MOP(x * y) = \{x - y, x + y, x/y\}$

2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Conditional Mutation Algorithm

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

1. Define mutation operators

   $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Conditional Mutation Algorithm

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators

   $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Conditional Mutation Algorithm

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators

   $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Conditional Mutation Algorithm

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators
   $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

- Versatile approach, can be combined with prior solutions
- Formal description and implementation details in the paper

Conditional Mutation Algorithm

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators
   $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

- Versatile approach, can be combined with prior solutions
- Formal description and implementation details in the paper

Conditional Mutation Algorithm

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators
   $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

- Versatile approach, can be combined with prior solutions
- Formal description and implementation details in the paper

Mutation Coverage

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)?  a / x :
        (M_NO==2)?  a + x :
        (M_NO==1)?  a - x :
                    a * x;

    y += b;
    return y;
}
```

Mutation Coverage

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

Mutants not executed cannot be killed

Mutation Coverage

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
        (M_NO==0 &&
         COVERED(1,3))?
            a * x : a * x;

    y += b;

    return y;
}
```

Mutants not executed cannot be killed

Determine covered mutants with additional instrumentation

Mutation Coverage

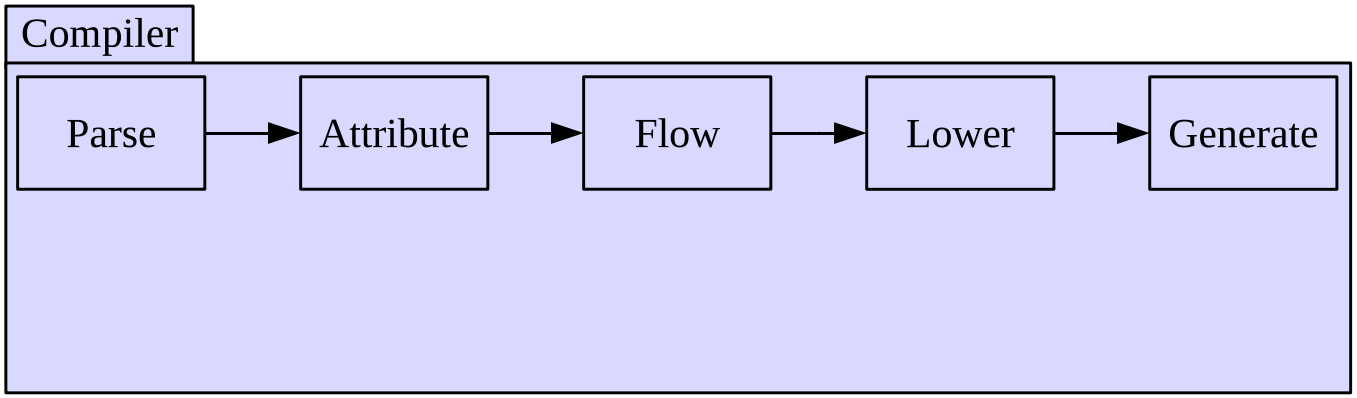
```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
        (M_NO==0 &&
         COVERED(1,3))?
            a * x : a * x;

    y += b;

    return y;
}
```

Mutants not executed cannot be killed

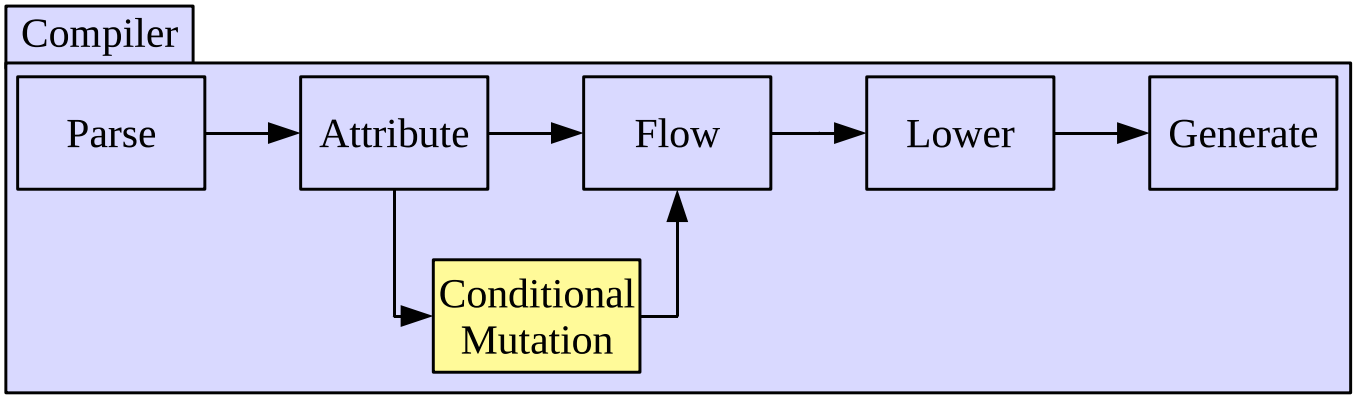
Determine covered mutants with additional instrumentation

Only execute and investigate the covered mutants

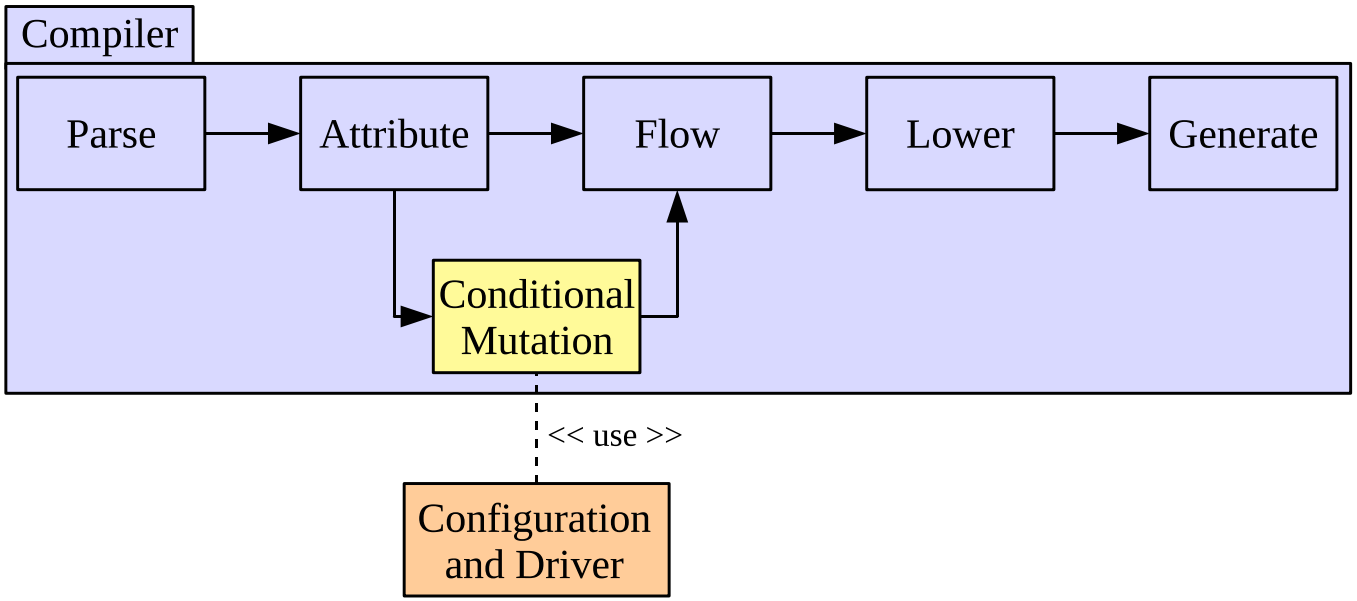
MAJOR: Mutation Analysis in a Java Compiler



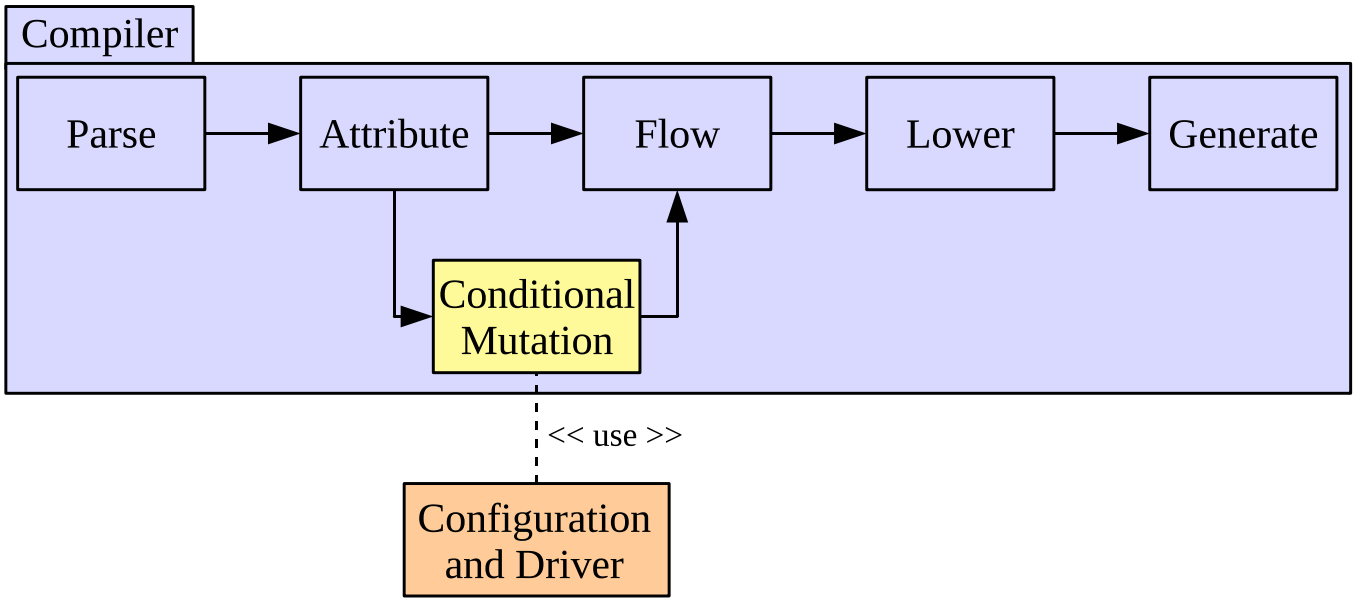
MAJOR: Mutation Analysis in a Java Compiler



MAJOR: Mutation Analysis in a Java Compiler



MAJOR: Mutation Analysis in a Java Compiler

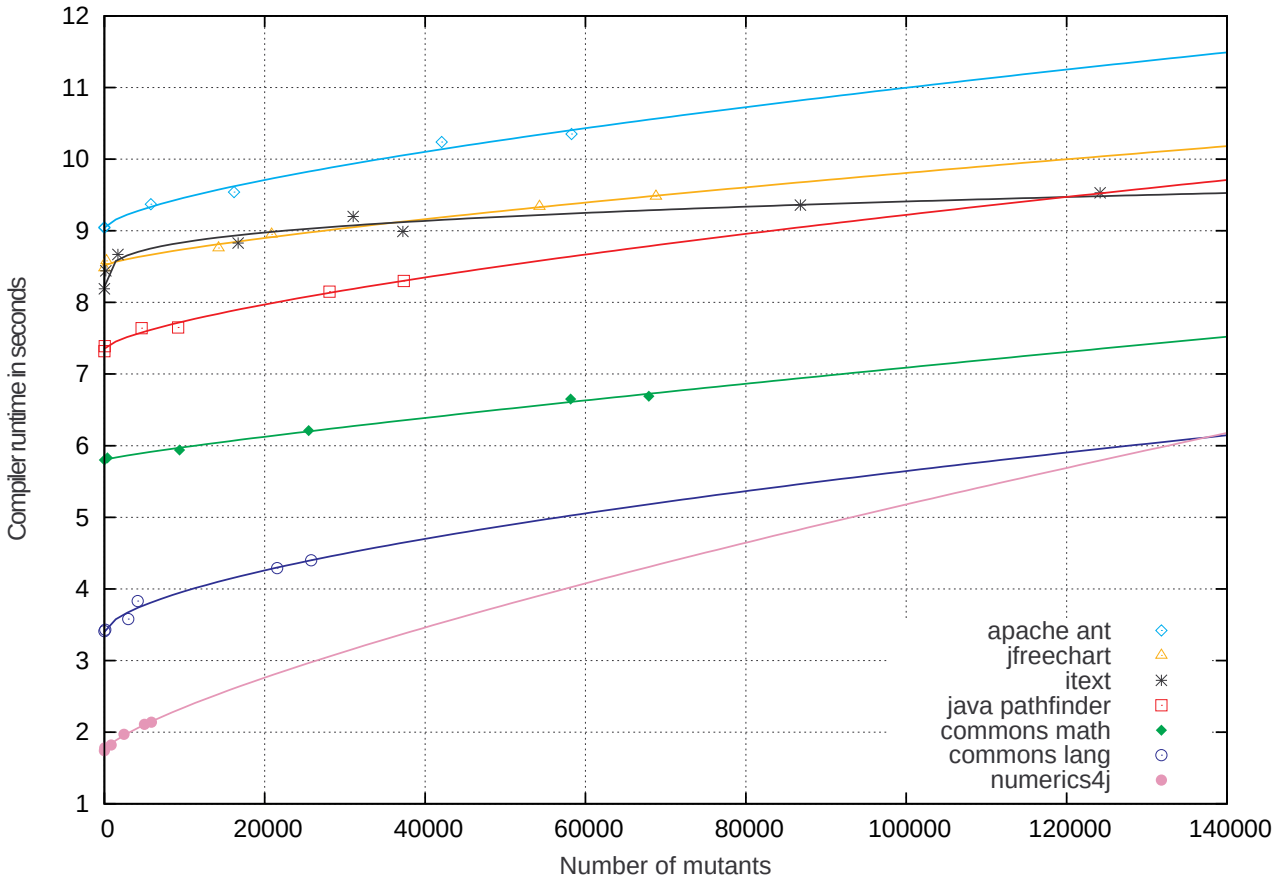


Configuration:

- Common compiler options
- Domain specific language (DSL)

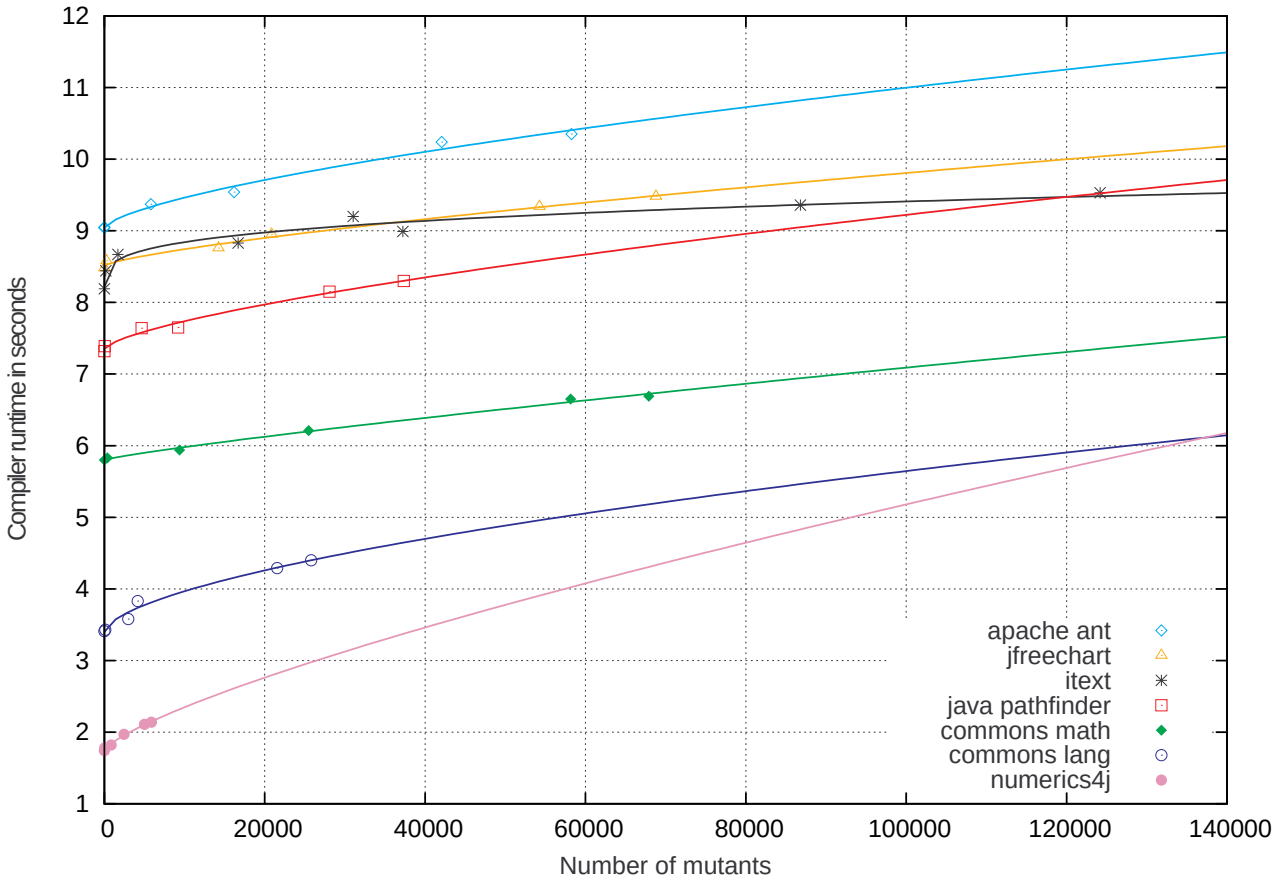
http://www.mathematik.uni-ulm.de/sai/major

Performance Analysis



- Overhead for generating and compiling mutants is negligible

Performance Analysis



- Overhead for generating and compiling mutants is negligible

Performance Analysis

Application	***Mutants***	***Runtime of test suite***			***Memory consumption***	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Performance Analysis

Application	***Mutants***	***Runtime of test suite***			***Memory consumption***	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Performance Analysis

Application	*Mutants*	*Runtime of test suite*			*Memory consumption*	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Performance Analysis

Application	*Mutants*	*Runtime of test suite*			*Memory consumption*	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

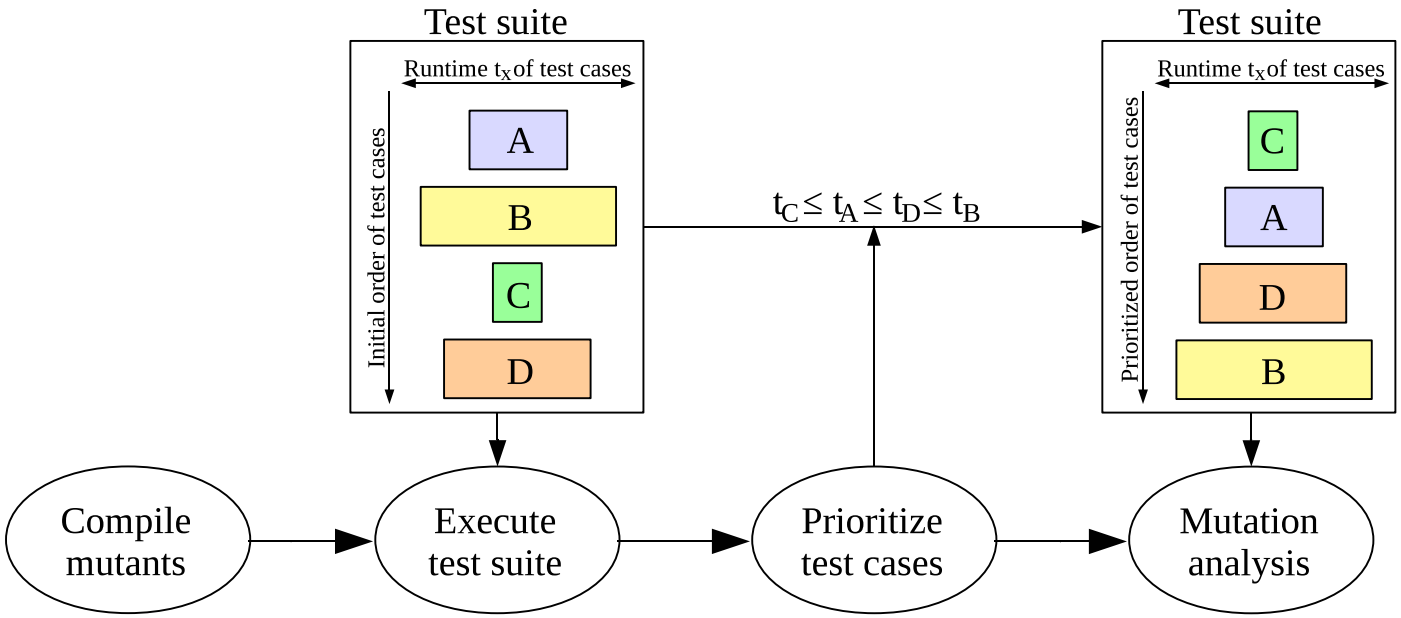
- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Performance Analysis

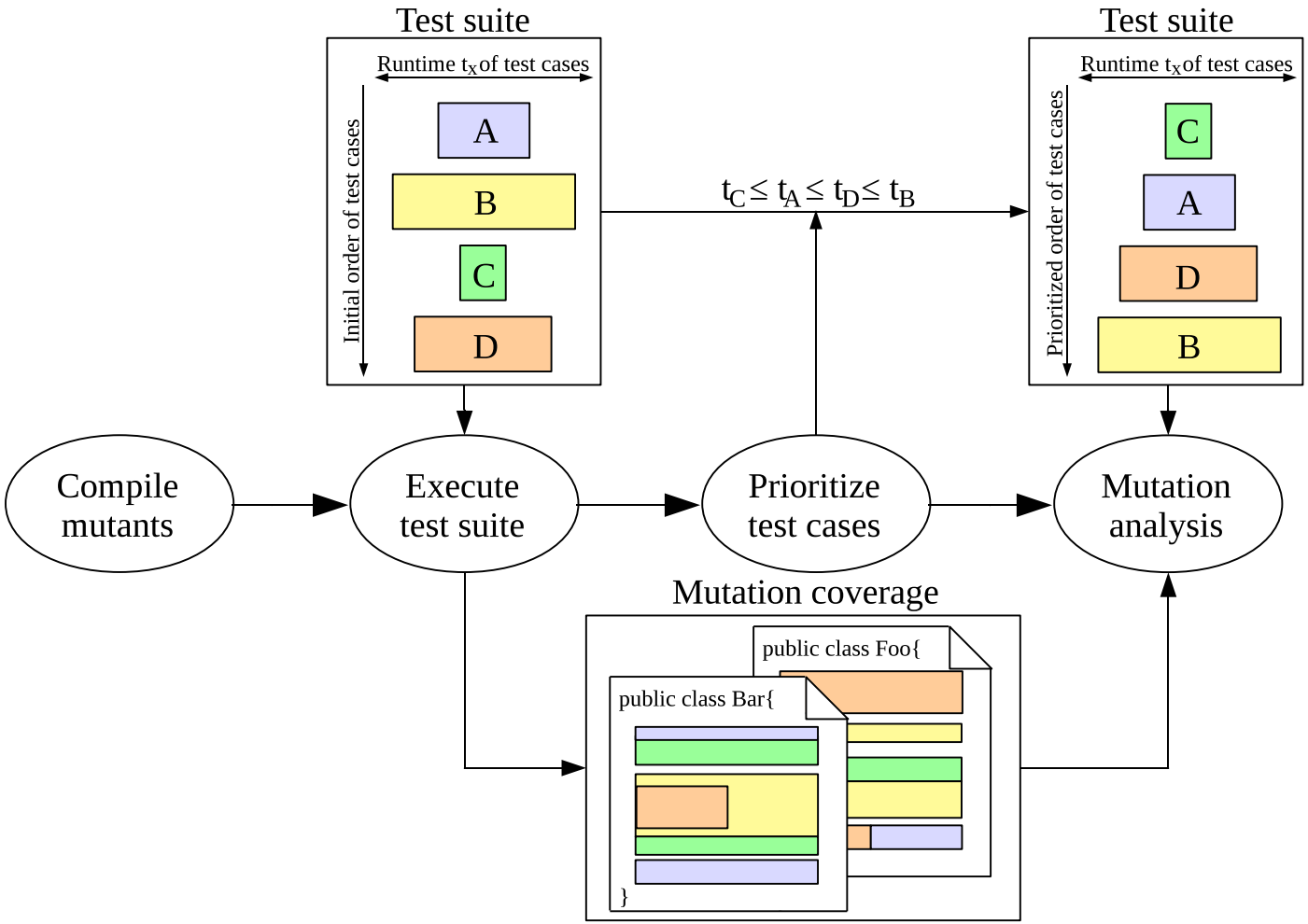
Application	***Mutants***	***Runtime of test suite***			***Memory consumption***	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Enabling Efficient Mutation Analysis



Enabling Efficient Mutation Analysis



Conclusion

Conclusion:

- Largest empirical study of mutation analysis to date
- Mutant generation time reduced to a minimum
- Mutation coverage provides runtime optimization
- Versatilely applicable in every Java-based environment
- Arbitrary conditions enable support for higher order mutation

Future Work:

- Implement new mutation operators
- Enhance the domain specific language

Conclusion

Conclusion:

- Largest empirical study of mutation analysis to date
- Mutant generation time reduced to a minimum
- Mutation coverage provides runtime optimization
- Versatilely applicable in every Java-based environment
- Arbitrary conditions enable support for higher order mutation

Future Work:

- Implement new mutation operators
- Enhance the domain specific language

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis

Thank you for your attention!

Questions?

ulm university universität uulm

ALLEGHENY COLLEGE

http://www.mathematik.uni-ulm.de/sai/major