Quantifying the Potential to Automate the Synchronization of Variants in Clone-and-Own

Alexander Schultheiß

In collaboration with

Paul Maximilian Bittner

ulm university universität ulm

Timo Kehrer

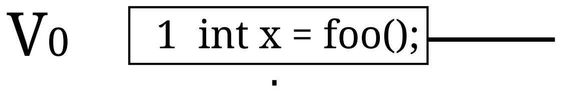
UNIVERSITÄT BERN

Thomas Thüm

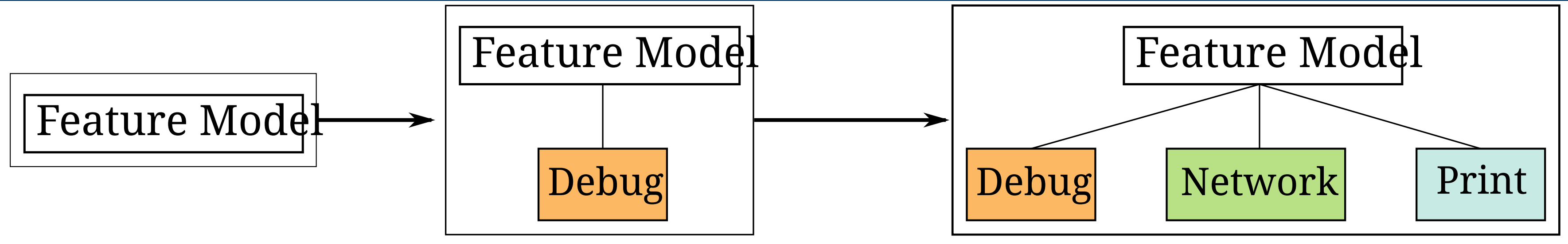
ulm university universität ulm

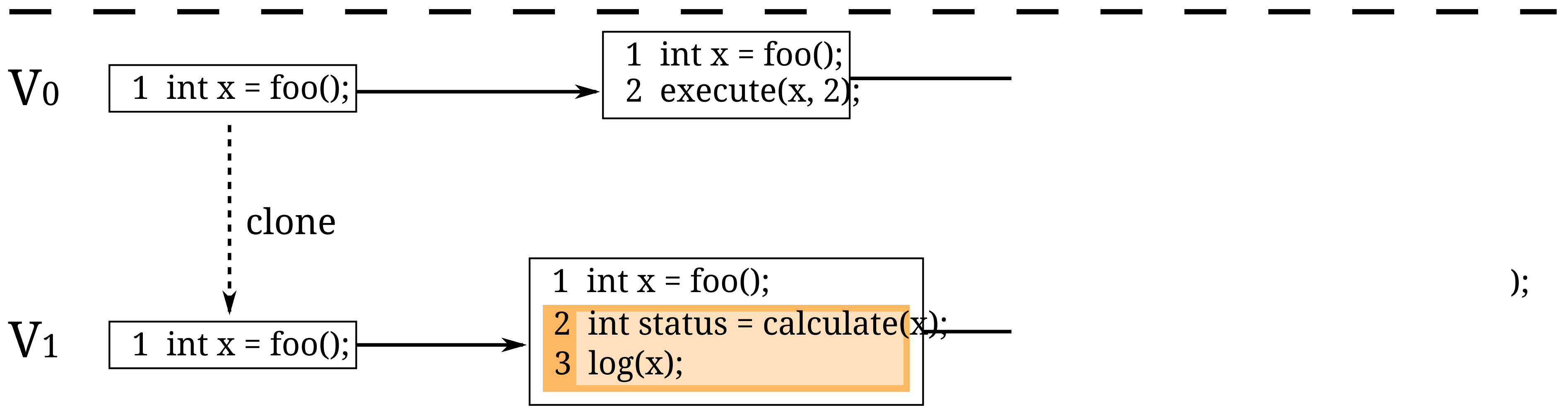
Clone-and-Own Development

Variants are created by copying and adapting existing variant



);



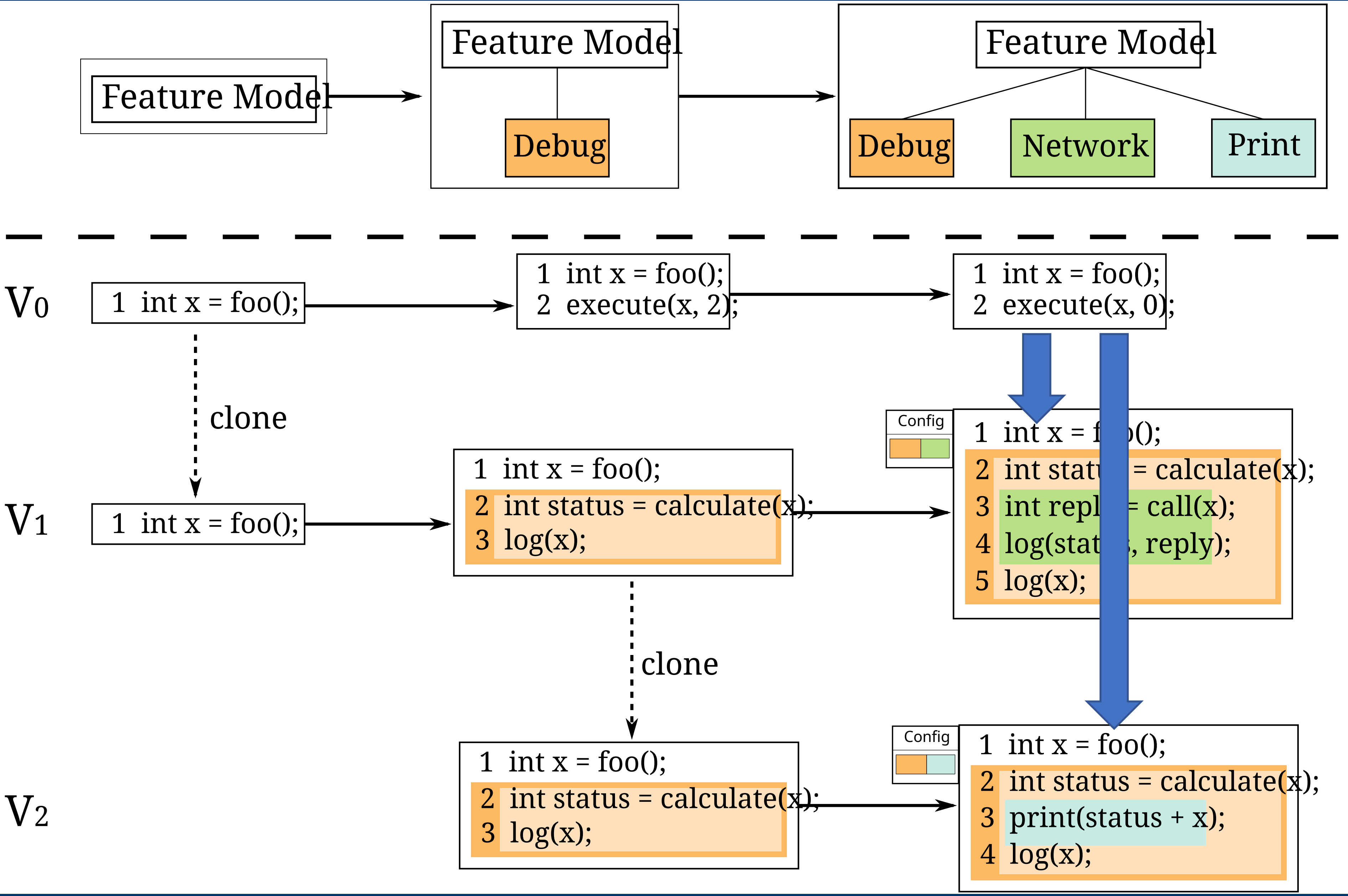


```
1  int x = foo();
```

```
1  int x = foo();
2  execute(x, 2);
```

```
1  int x = foo();
```

```
1  int x = foo();
2  int status = calculate(x);
3  log(x);
```



```
1  int x = foo();
```

```
1  int x = foo();
2  execute(x, 2);
```

```
1  int x = foo();
2  execute(x, 0);
```

```
1  int x = foo();
```

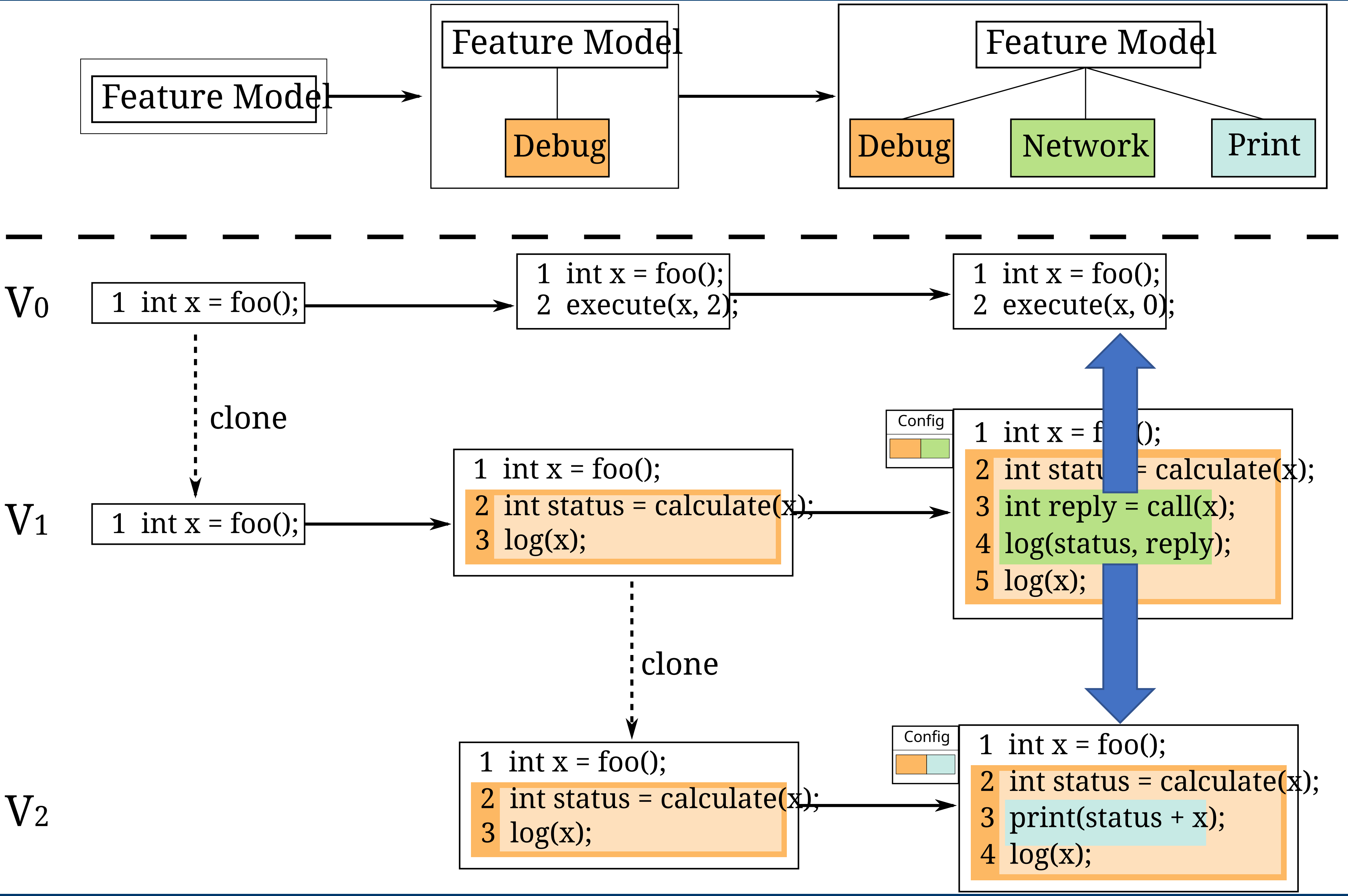
```
1  int x = foo();
2  int status = calculate(x);
3  log(x);
```

```
1  int x = foo();
2  int status = calculate(x);
3  int reply = call(x);
4  log(status, reply);
5  log(x);
```

```
1  int x = foo();
2  int status = calculate(x);
3  log(x);
```

```
1  int x = foo();
2  int status = calculate(x);
3  print(status + x);
4  log(x);
```

Where to apply a change?



Which changes to propagate?

Quantifying the Automation Potential

Goals of the study

Goals:

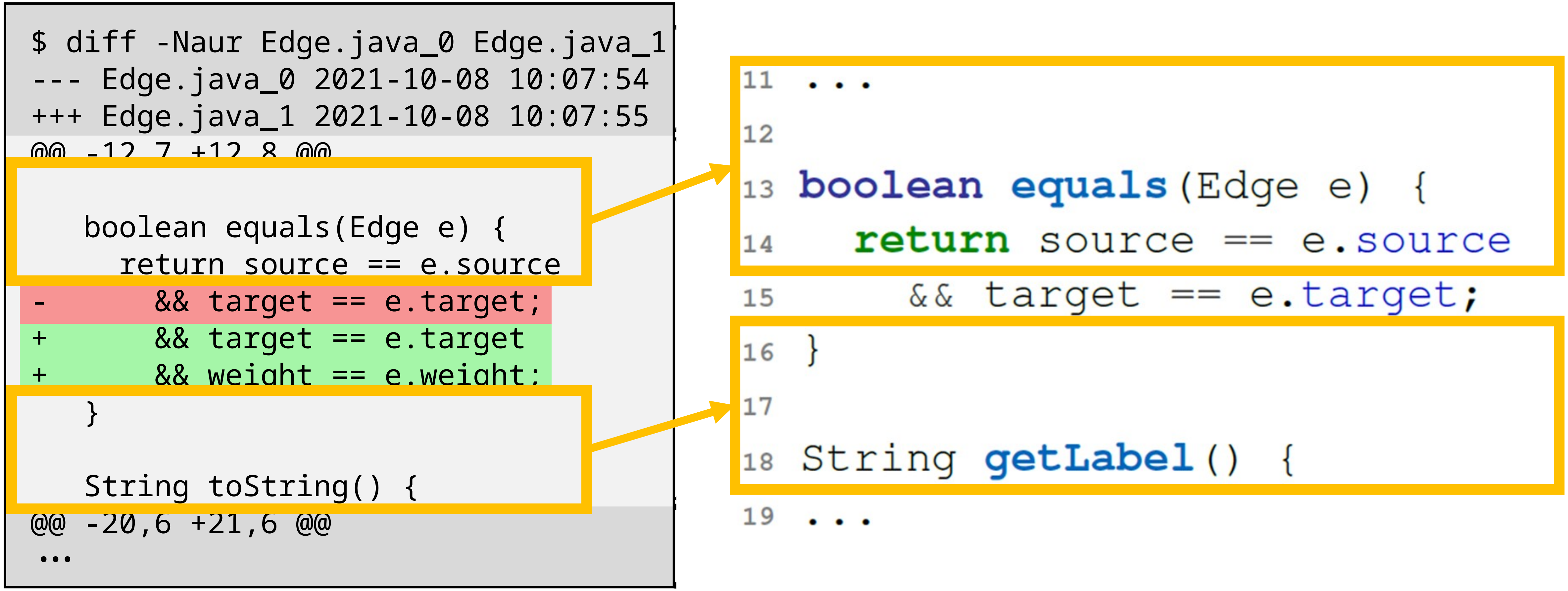
- Simulate automated change propagation
- Evaluate applicability and correctness
- Assess the benefit of knowledge about features

What we need:

- Change propagation technique
- Software variants
- History of the variants

Change Propagation with *Diff* and *Patch*

Patch applies changes based on a *diff*



```
$ diff -Naur Edge.java_0 Edge.java_1
--- Edge.java_0 2021-10-08 10:07:54
+++ Edge.java_1 2021-10-08 10:07:55
@@ -12,7 +12,8 @@

   boolean equals(Edge e) {
     return source == e.source
-        && target == e.target;
+        && target == e.target
+        && weight == e.weight;
   }

   String toString() {
@@ -20,6 +21,6 @@
 ...
```

```
...

boolean equals(Edge e) {
   return source == e.source
         && target == e.target;
}

String getLabel() {
...
```

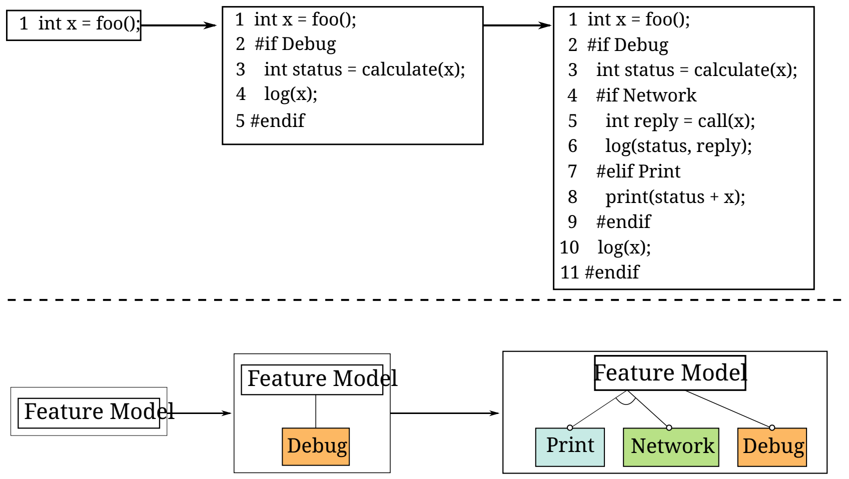
Variants and their histories

We consider the evolution of variants in BusyBox

Software product line development

We can analyze the product line…

Software Product Line

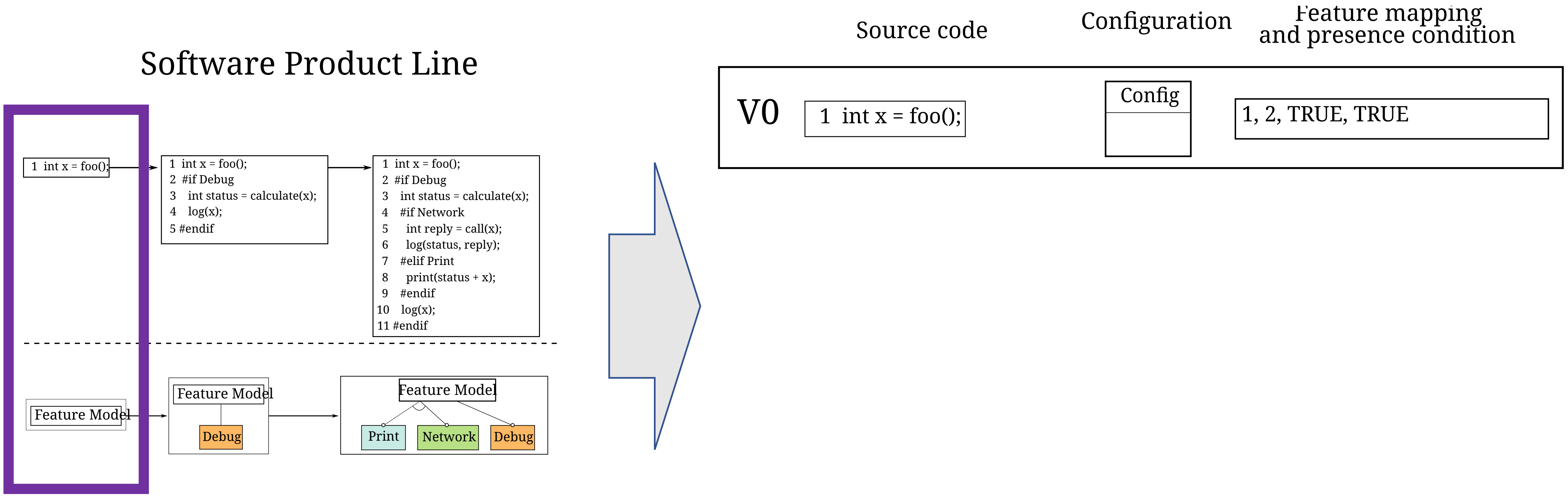


```
int x = foo();
```

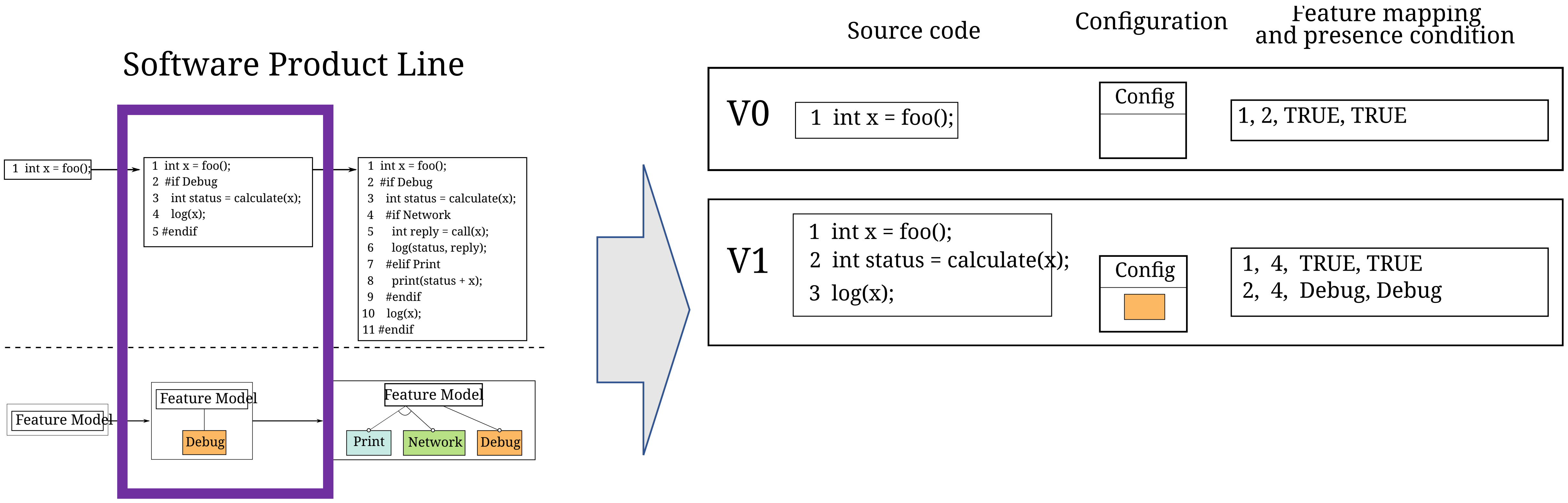
```
int x = foo();
#if Debug
  int status = calculate(x);
  log(x);
#endif
```

```
int x = foo();
#if Debug
  int status = calculate(x);
  #if Network
    int reply = call(x);
    log(status, reply);
  #elif Print
    print(status + x);
  #endif
  log(x);
#endif
```

… to generate variants…



... to generate variants for each revision



```
1  int x = foo();
```

```
1  int x = foo();
2  #if Debug
3    int status = calculate(x);
4    log(x);
5 #endif
```

```
1  int x = foo();
2  #if Debug
3    int status = calculate(x);
4    #if Network
5      int reply = call(x);
6      log(status, reply);
7    #elif Print
8      print(status + x);
9    #endif
10   log(x);
11 #endif
```

V0

```
1  int x = foo();
```

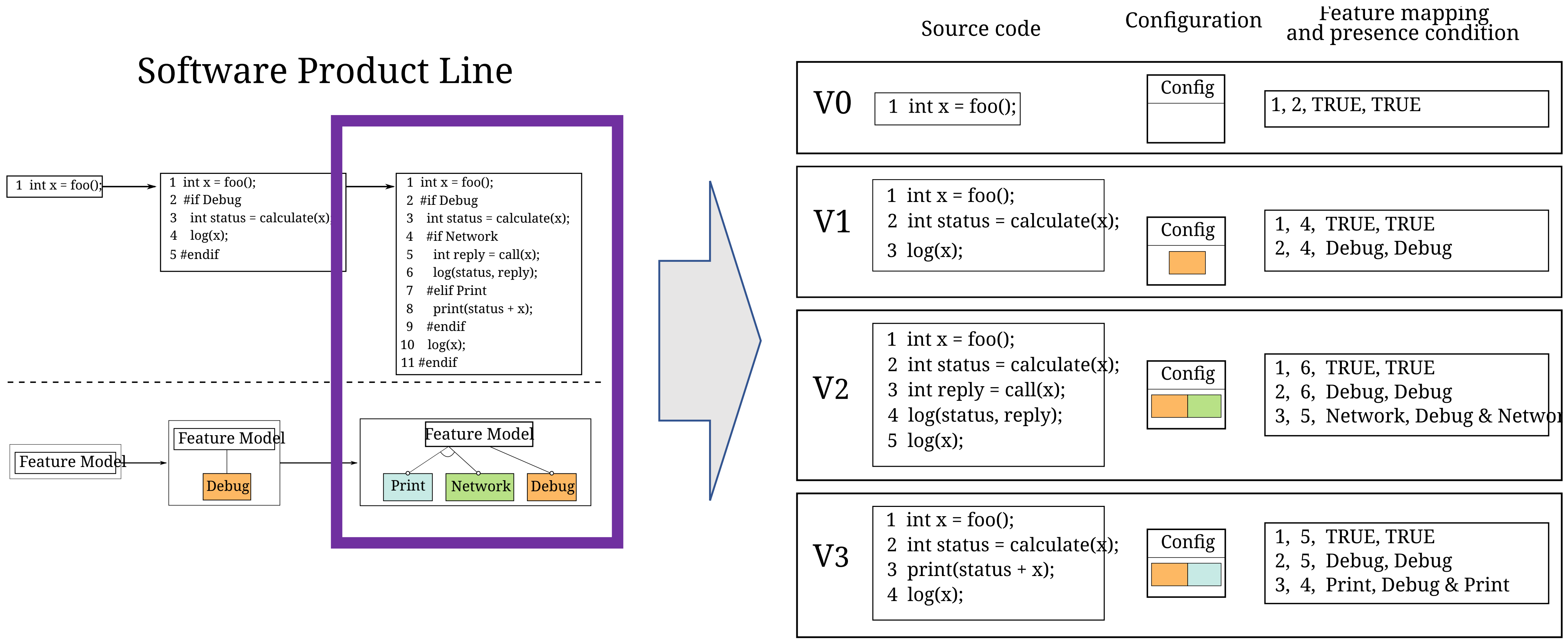
1, 2, TRUE, TRUE

V1

```
1  int x = foo();
2  int status = calculate(x);
3  log(x);
```

1, 4, TRUE, TRUE
2, 4, Debug, Debug

… to generate variants for each revision



```
1  int x = foo();
```

```
1  int x = foo();
2  #if Debug
3    int status = calculate(x);
4    log(x);
5 #endif
```

```
1  int x = foo();
2  #if Debug
3    int status = calculate(x);
4    #if Network
5      int reply = call(x);
6      log(status, reply);
7    #elif Print
8      print(status + x);
9    #endif
10   log(x);
11 #endif
```

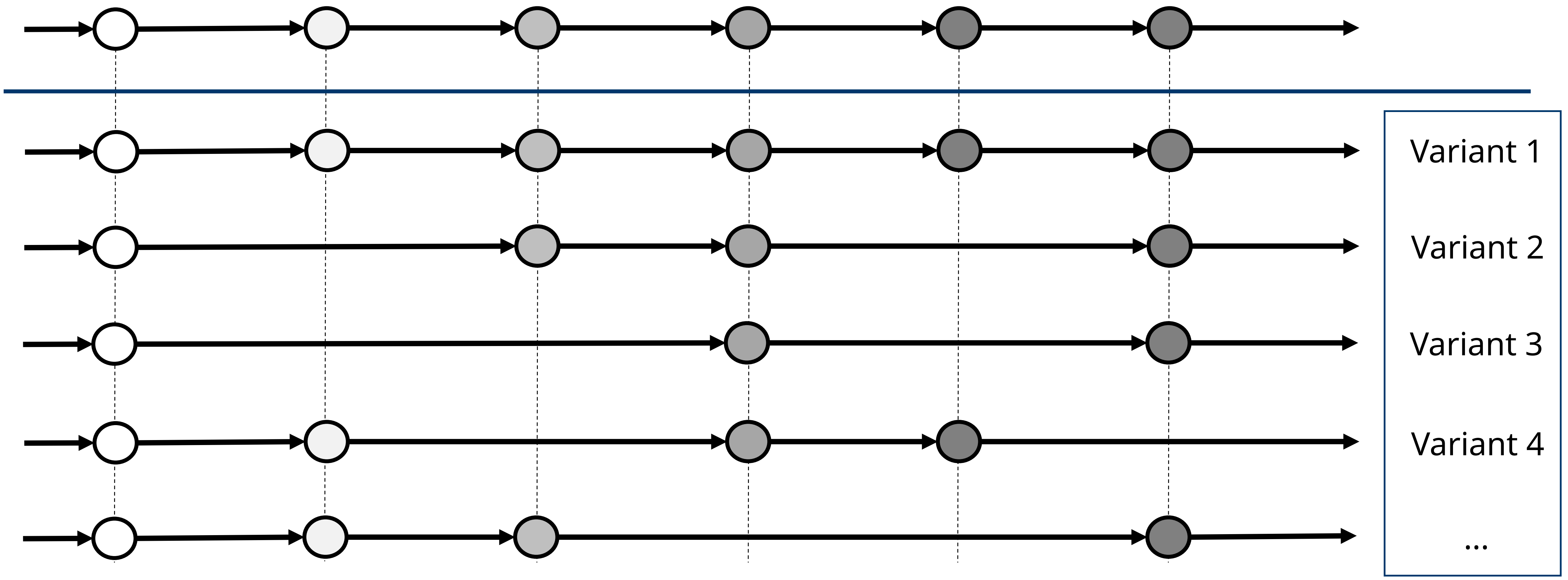
	Source code	Configuration	Feature mapping and presence condition
V0	1 int x = foo();	Config	1, 2, TRUE, TRUE

V1	1 int x = foo();
	2 int status = calculate(x);
	3 log(x);
		Config 🟧	1, 4, TRUE, TRUE
			2, 4, Debug, Debug

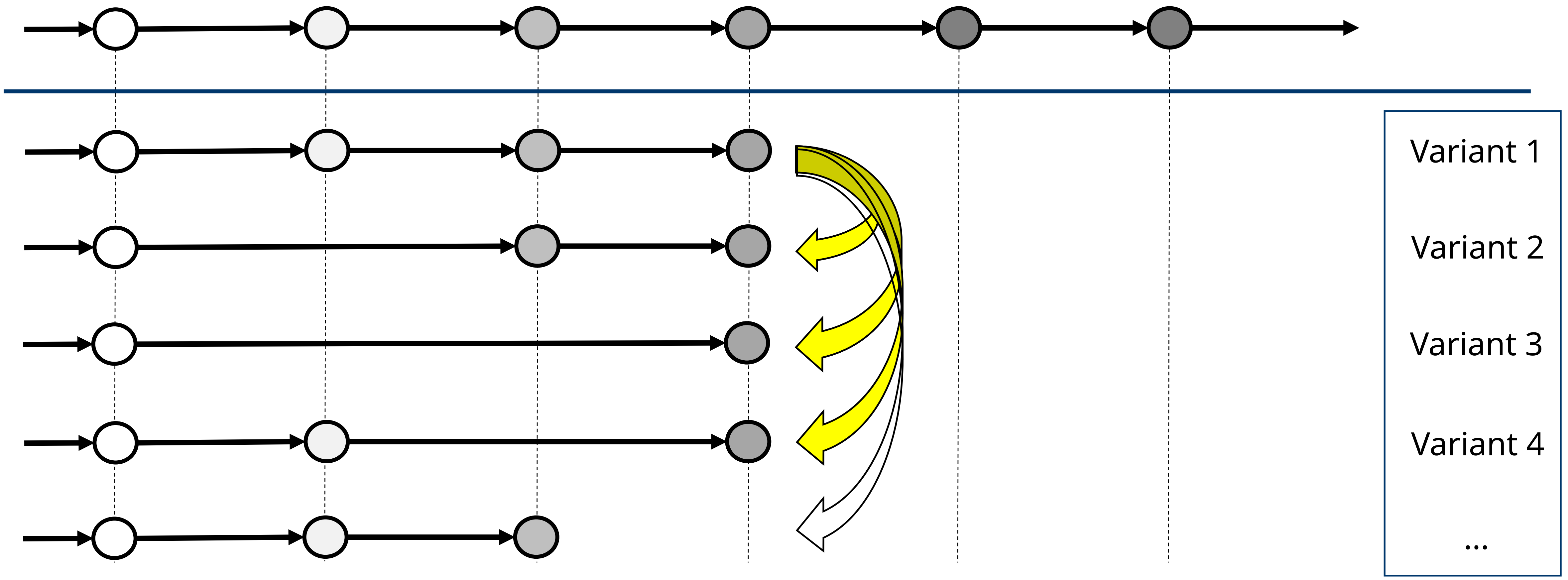
V2	1 int x = foo();
	2 int status = calculate(x);
	3 int reply = call(x);
	4 log(status, reply);
	5 log(x);
		Config 🟧🟩	1, 6, TRUE, TRUE
			2, 6, Debug, Debug
			3, 5, Network, Debug & Networ

V3	1 int x = foo();
	2 int status = calculate(x);
	3 print(status + x);
	4 log(x);
		Config 🟧🟦	1, 5, TRUE, TRUE
			2, 5, Debug, Debug
			3, 4, Print, Debug & Print

We can simulate a history

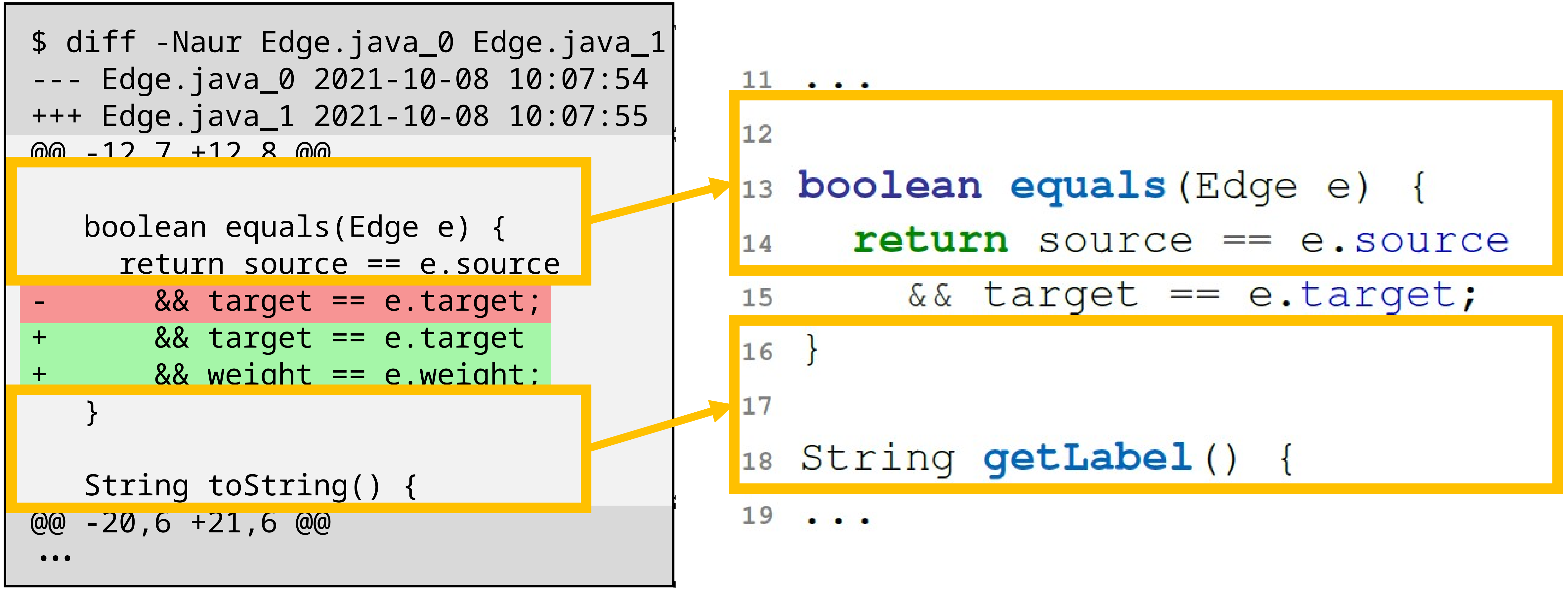


We can conduct our study



Results

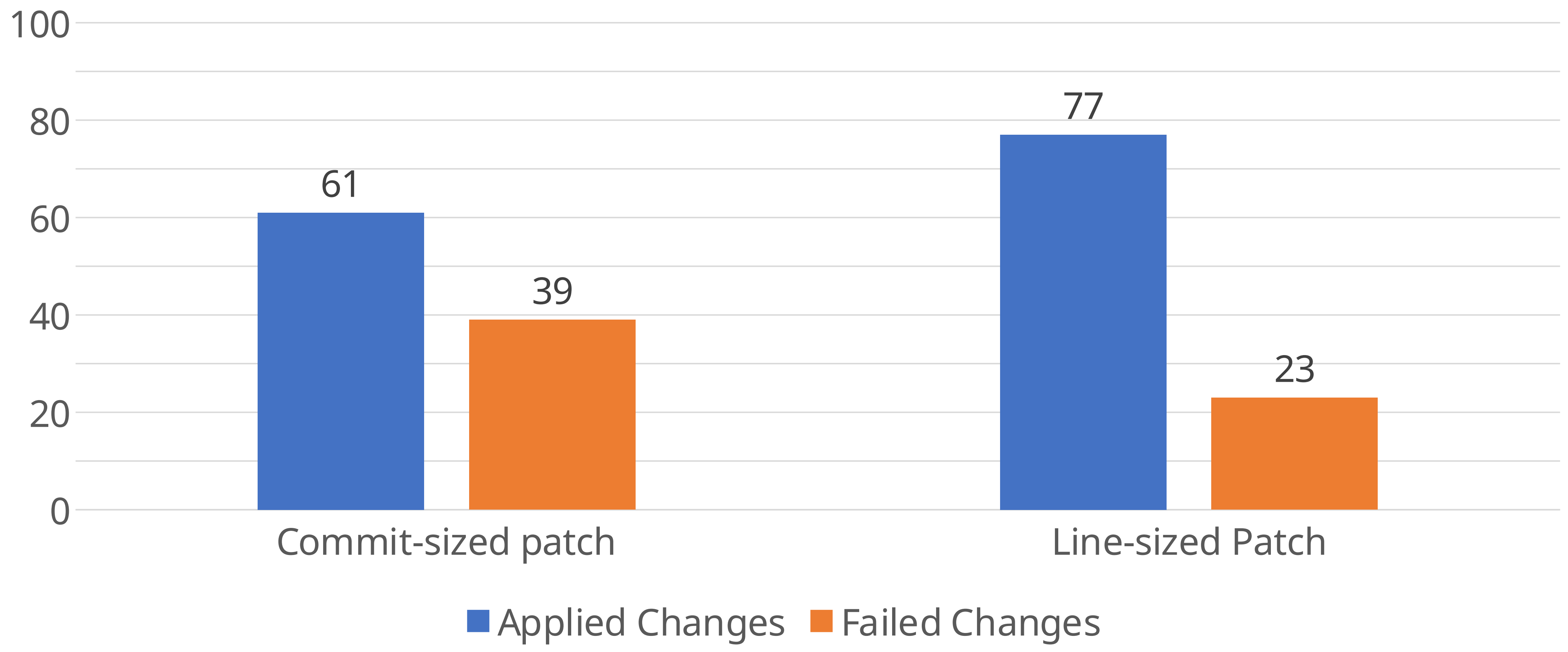
RQ1: How often can changes be applied?



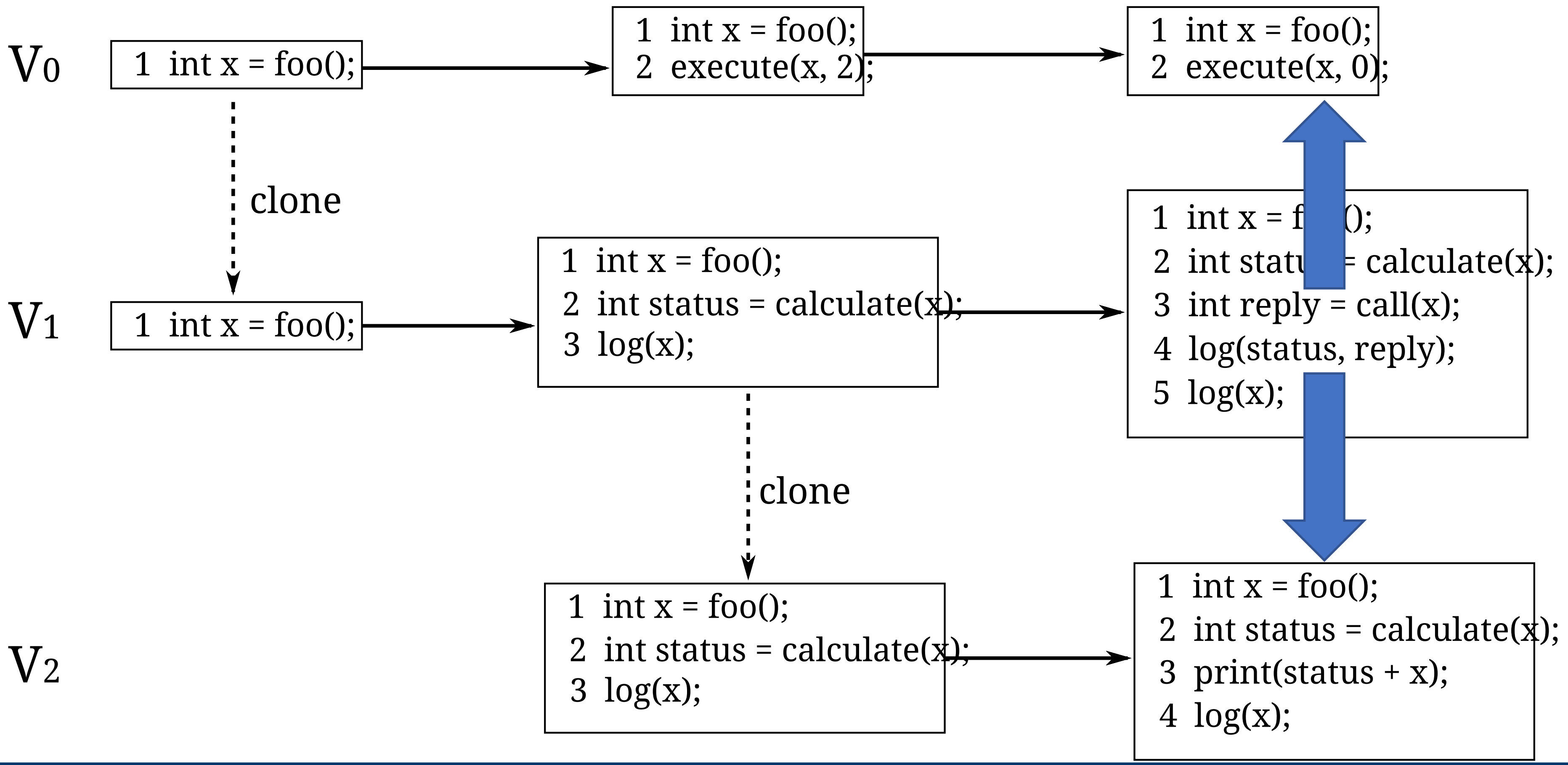
```
$ diff -Naur Edge.java_0 Edge.java_1
--- Edge.java_0 2021-10-08 10:07:54
+++ Edge.java_1 2021-10-08 10:07:55
@@ -12,7 +12,8 @@

   boolean equals(Edge e) {
     return source == e.source
-        && target == e.target;
+        && target == e.target
+        && weight == e.weight;
   }

   String toString() {
@@ -20,6 +21,6 @@
 ...
```

```
...

boolean equals(Edge e) {
   return source == e.source
      && target == e.target;
}

String getLabel() {
...
```

RQ1: Applicability of Blindly Propagated Changes (in %)



RQ2: How often is blind change propagation correct?



```
int x = foo();
```

```
int x = foo();
execute(x, 2);
```

```
int x = foo();
execute(x, 0);
```

```
int x = foo();
```

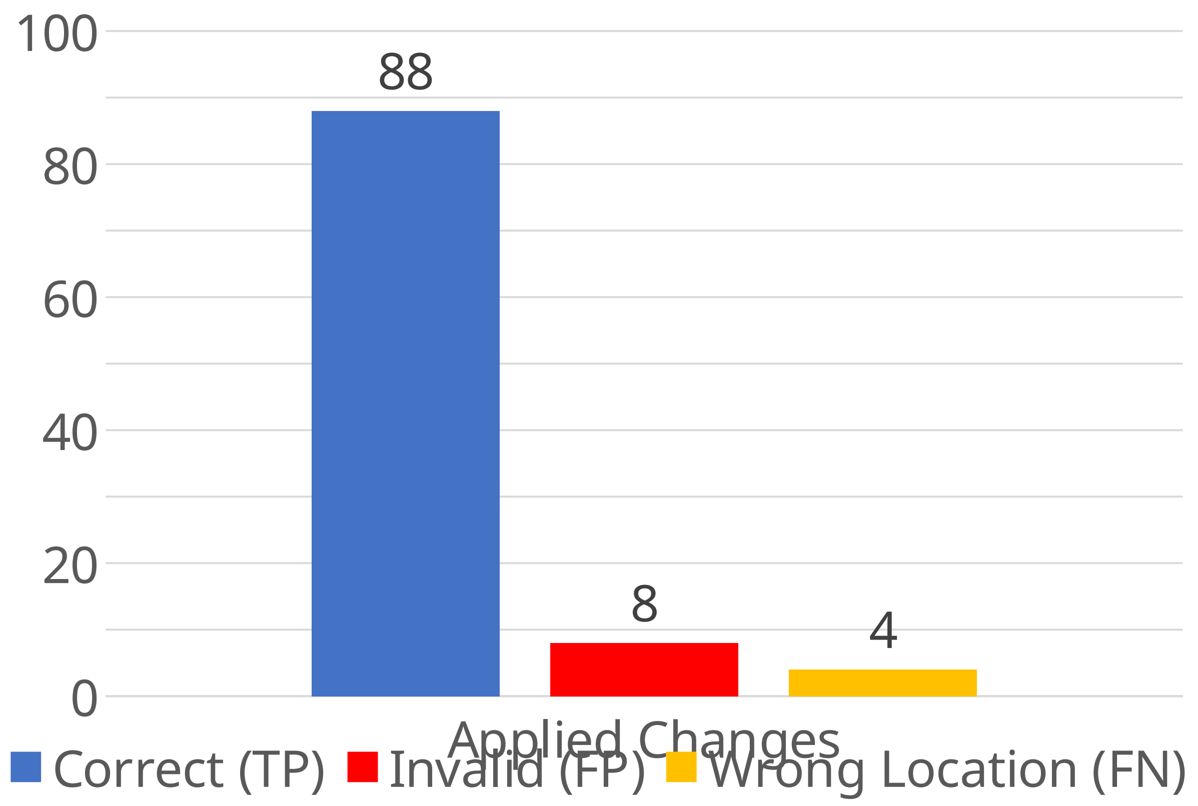
```
int x = foo();
int status = calculate(x);
log(x);
```

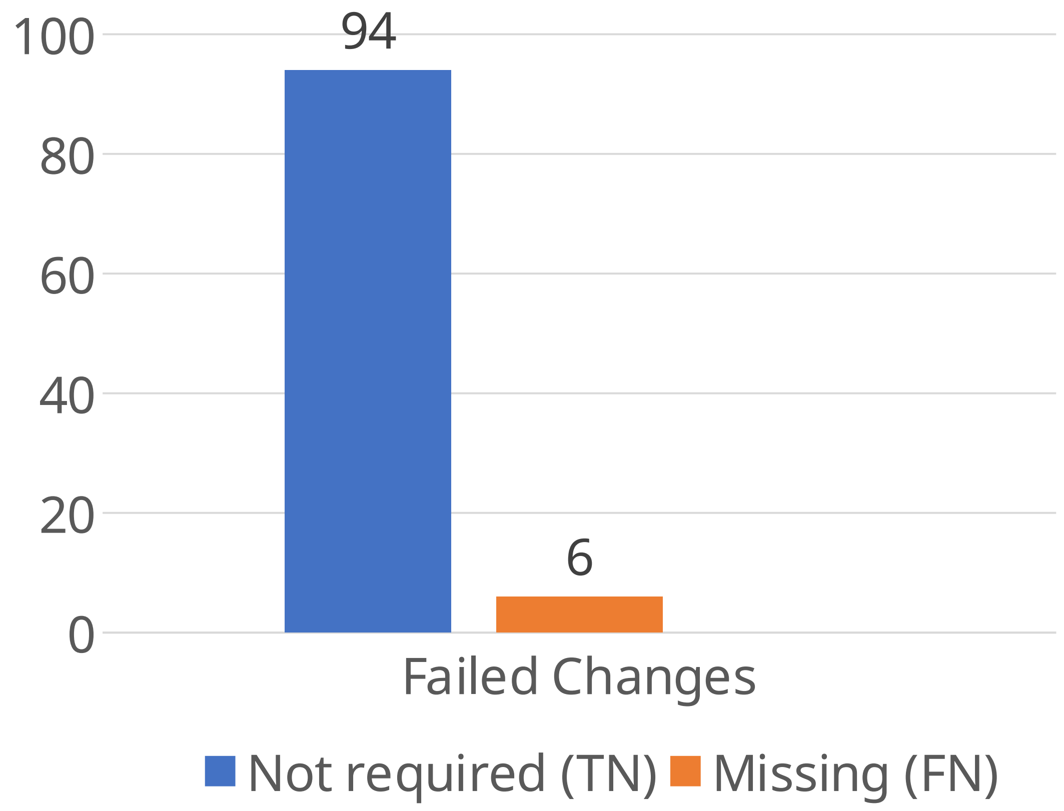
```
int x = foo();
int status = calculate(x);
int reply = call(x);
log(status, reply);
log(x);
```

```
int x = foo();
int status = calculate(x);
log(x);
```

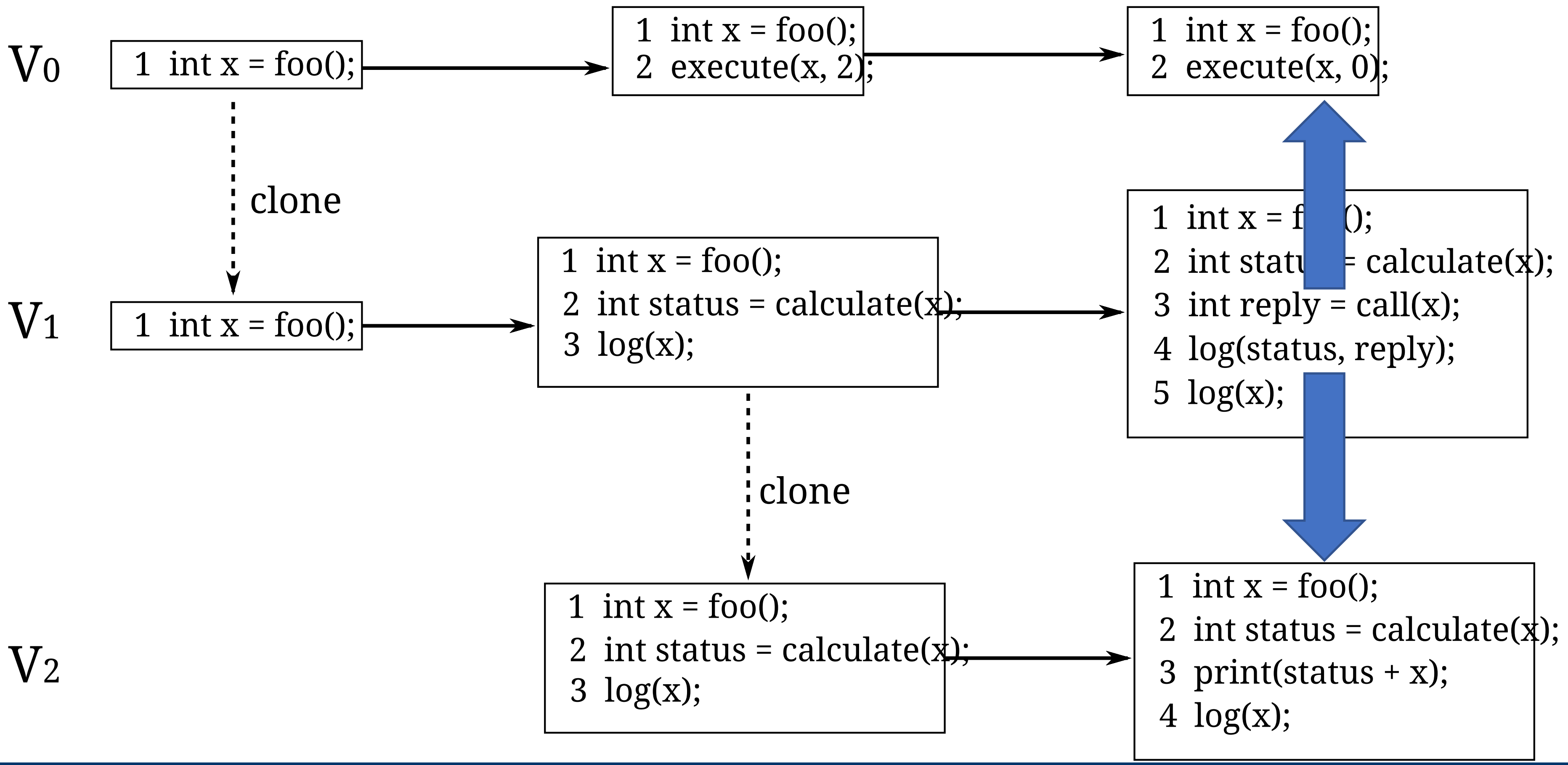
```
int x = foo();
int status = calculate(x);
print(status + x);
log(x);
```

RQ2: Correctness of Blindly Propagated Changes (in %)





RQ3: What is the impact of considering knowledge about features?



```
int x = foo();
```

```
int x = foo();
execute(x, 2);
```

```
int x = foo();
execute(x, 0);
```

```
int x = foo();
```

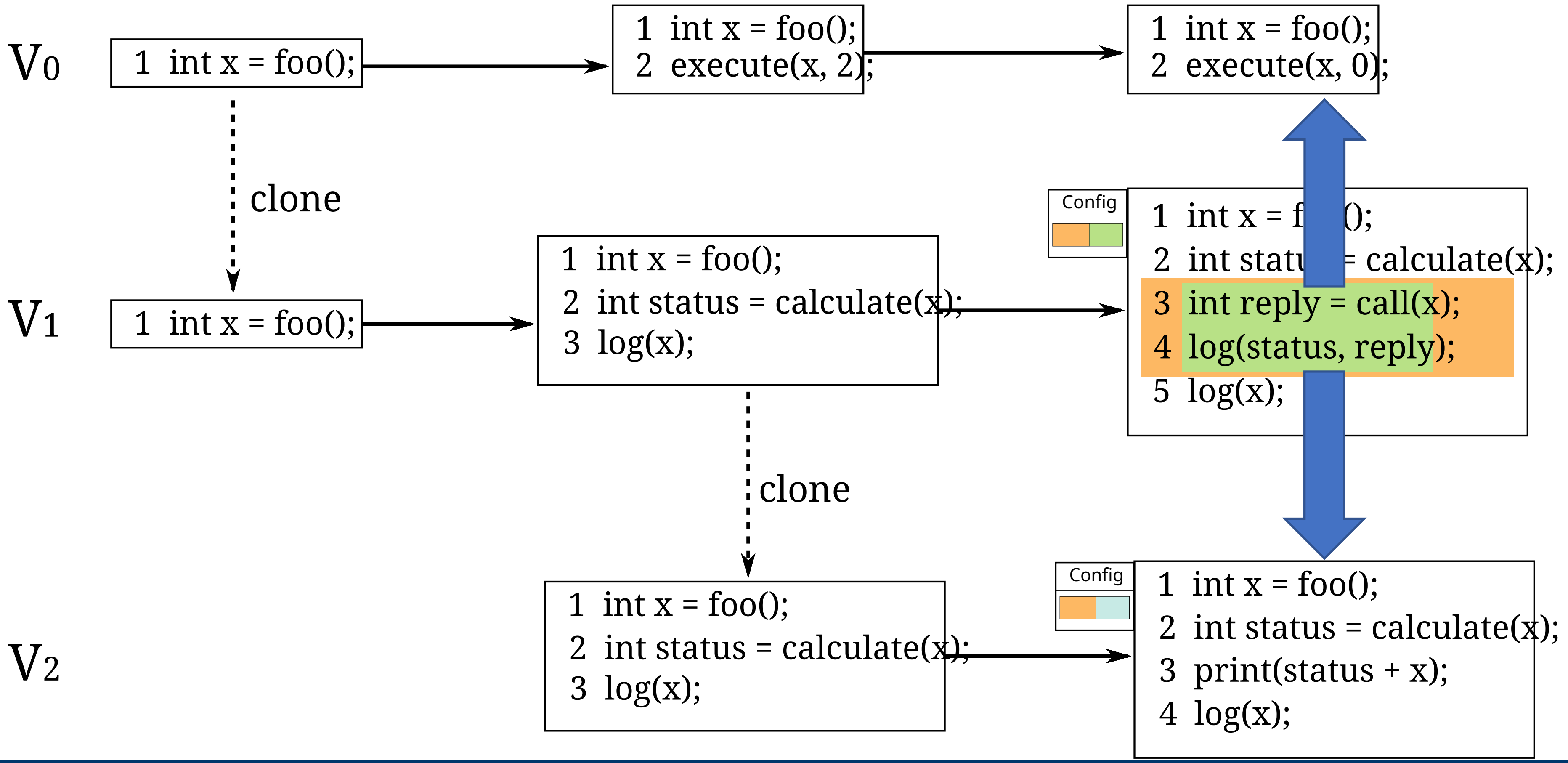
```
int x = foo();
int status = calculate(x);
log(x);
```

```
int x = f[illegible]();
int statu[illegible] = calculate(x);
int reply = call(x);
log(status, reply);
log(x);
```

```
int x = foo();
int status = calculate(x);
log(x);
```

```
int x = foo();
int status = calculate(x);
print(status + x);
log(x);
```

RQ3: What is the impact of considering knowledge about features?



```
1  int x = foo();
```

```
1  int x = foo();
2  execute(x, 2);
```

```
1  int x = foo();
2  execute(x, 0);
```

```
1  int x = foo();
```

```
1  int x = foo();
2  int status = calculate(x);
3  log(x);
```

```
1  int x = foo();
2  int status = calculate(x);
3  int reply = call(x);
4  log(status, reply);
5  log(x);
```

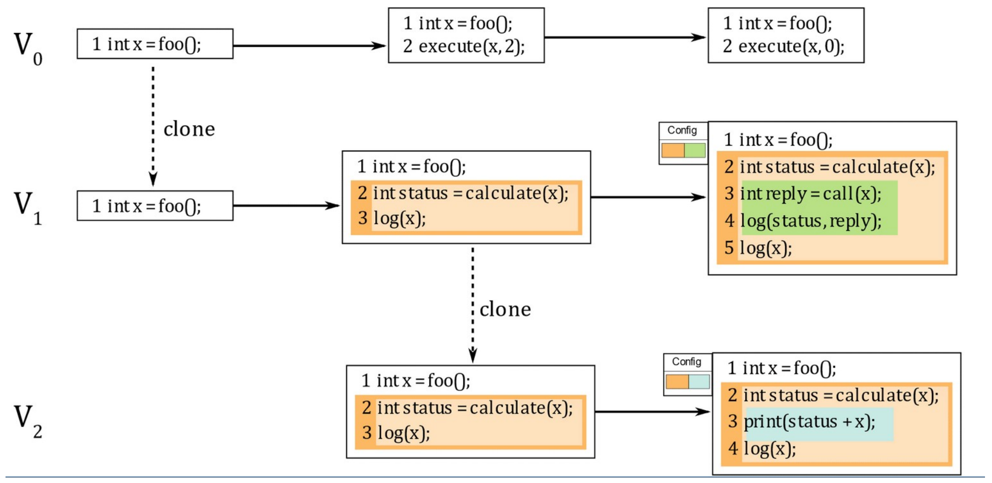
```
1  int x = foo();
2  int status = calculate(x);
3  log(x);
```

```
1  int x = foo();
2  int status = calculate(x);
3  print(status + x);
4  log(x);
```

RQ3: Impact of considering knowledge about features

	Precision (are applied changes correct?)	**Recall (have relevant changes been applied?)**	**Balanced Accuracy (degree of correct results)**
Blind propagation	0.92	0.93	0.85
Feature-aware propagation	0.97	0.93	0.93

In summary…



```
1 int x = foo();
```

```
1 int x = foo();
2 execute(x, 2);
```

```
1 int x = foo();
2 execute(x, 0);
```

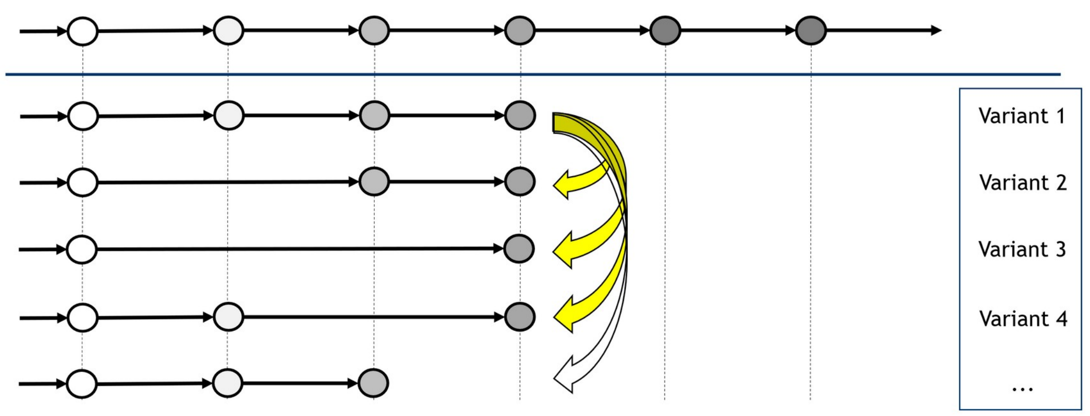
```
1 int x = foo();
```

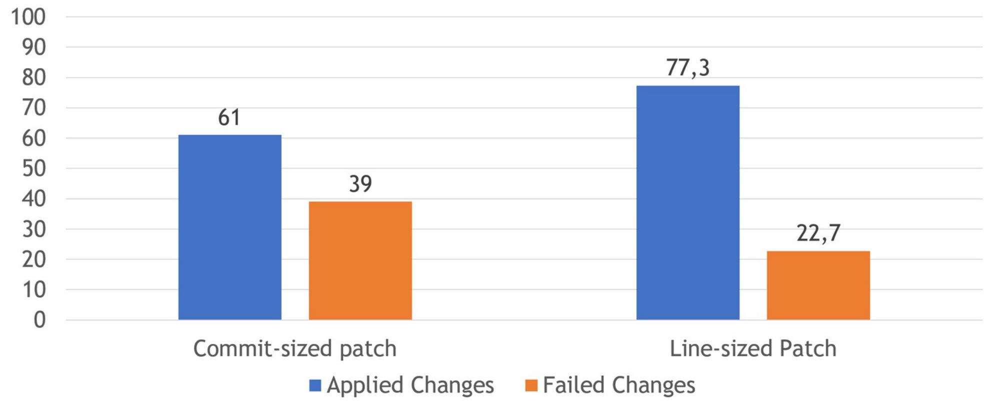
```
1 int x = foo();
2 int status = calculate(x);
3 log(x);
```

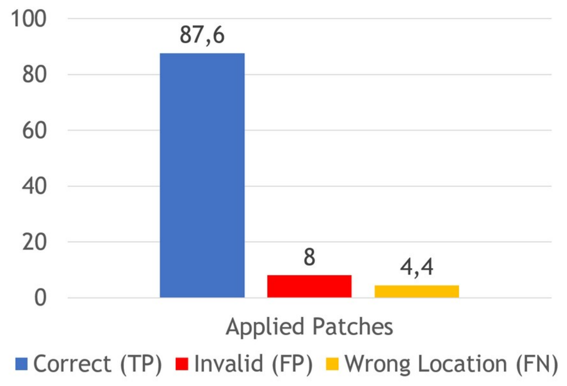
```
1 int x = foo();
2 int status = calculate(x);
3 int reply = call(x);
4 log(status, reply);
5 log(x);
```

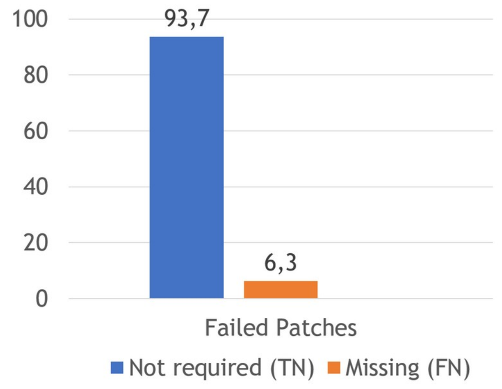
```
1 int x = foo();
2 int status = calculate(x);
3 log(x);
```

```
1 int x = foo();
2 int status = calculate(x);
3 print(status + x);
4 log(x);
```

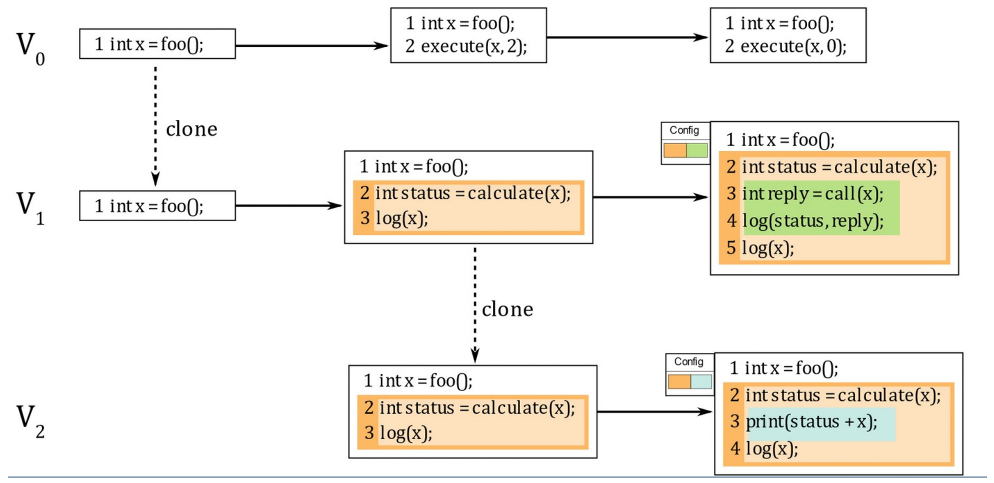




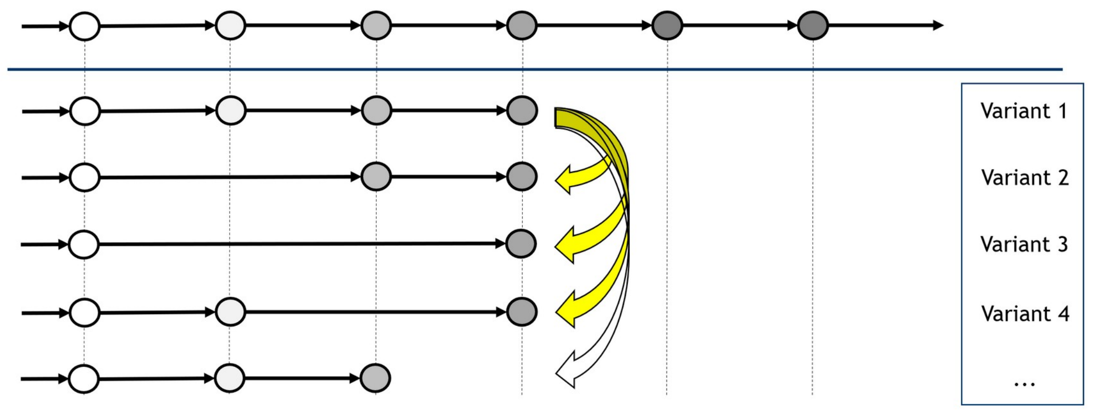




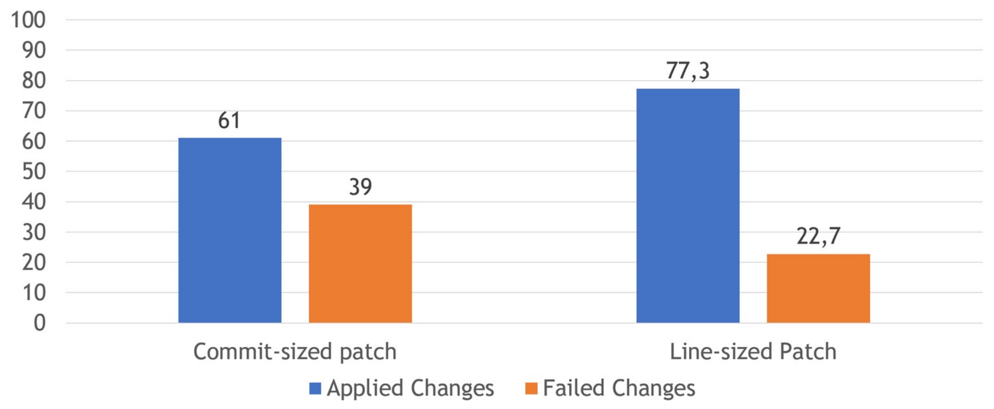
Questions?



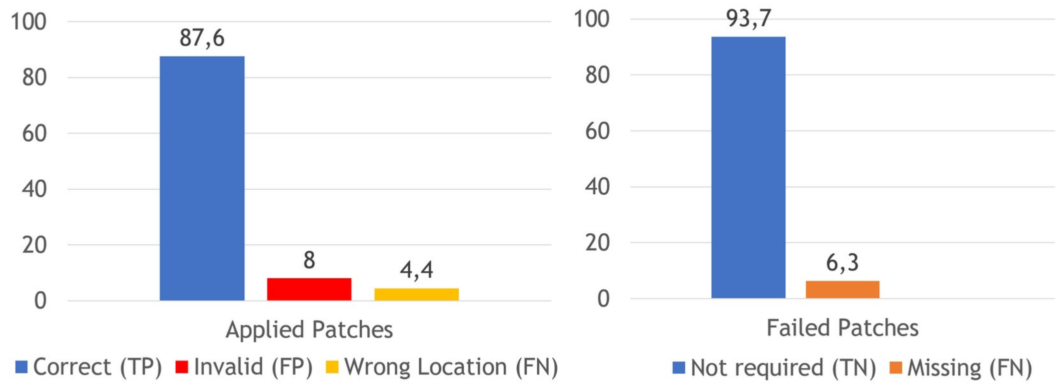
V0
1 int x = foo();
1 int x = foo();
2 execute(x, 2);
1 int x = foo();
2 execute(x, 0);
clone
V1
1 int x = foo();
1 int x = foo();
2 int status = calculate(x);
3 log(x);
Config
1 int x = foo();
2 int status = calculate(x);
3 int reply = call(x);
4 log(status, reply);
5 log(x);
clone
V2
1 int x = foo();
2 int status = calculate(x);
3 log(x);
Config
1 int x = foo();
2 int status = calculate(x);
3 print(status + x);
4 log(x);



Variant 1
Variant 2
Variant 3
Variant 4
...

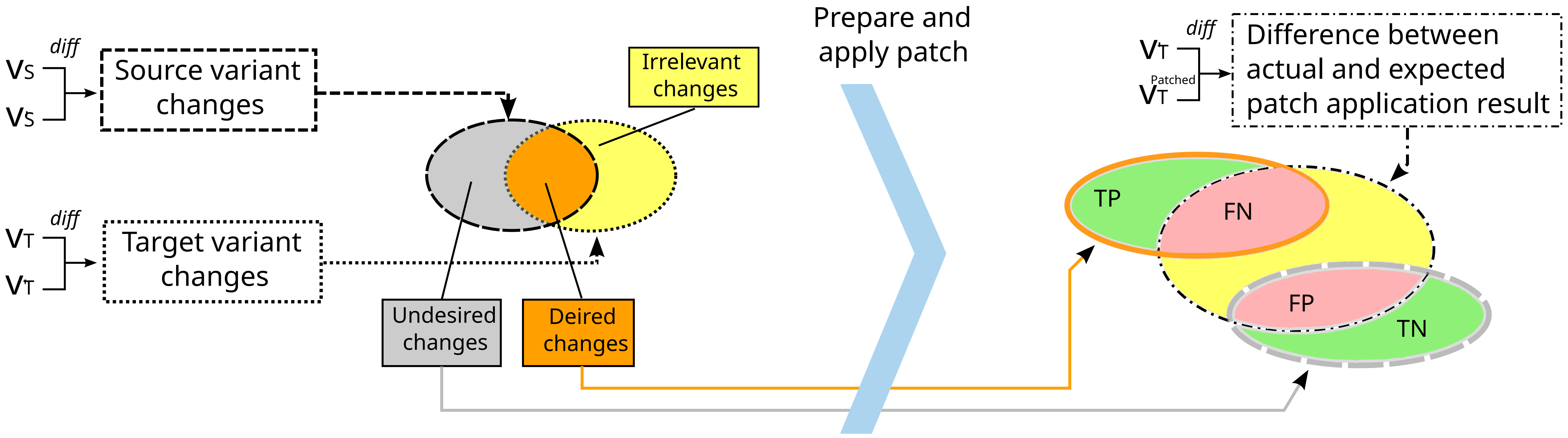


100
90
80
70
60
50
40
30
20
10
0
61
39
77,3
22,7
Commit-sized patch
Line-sized Patch
Applied Changes
Failed Changes

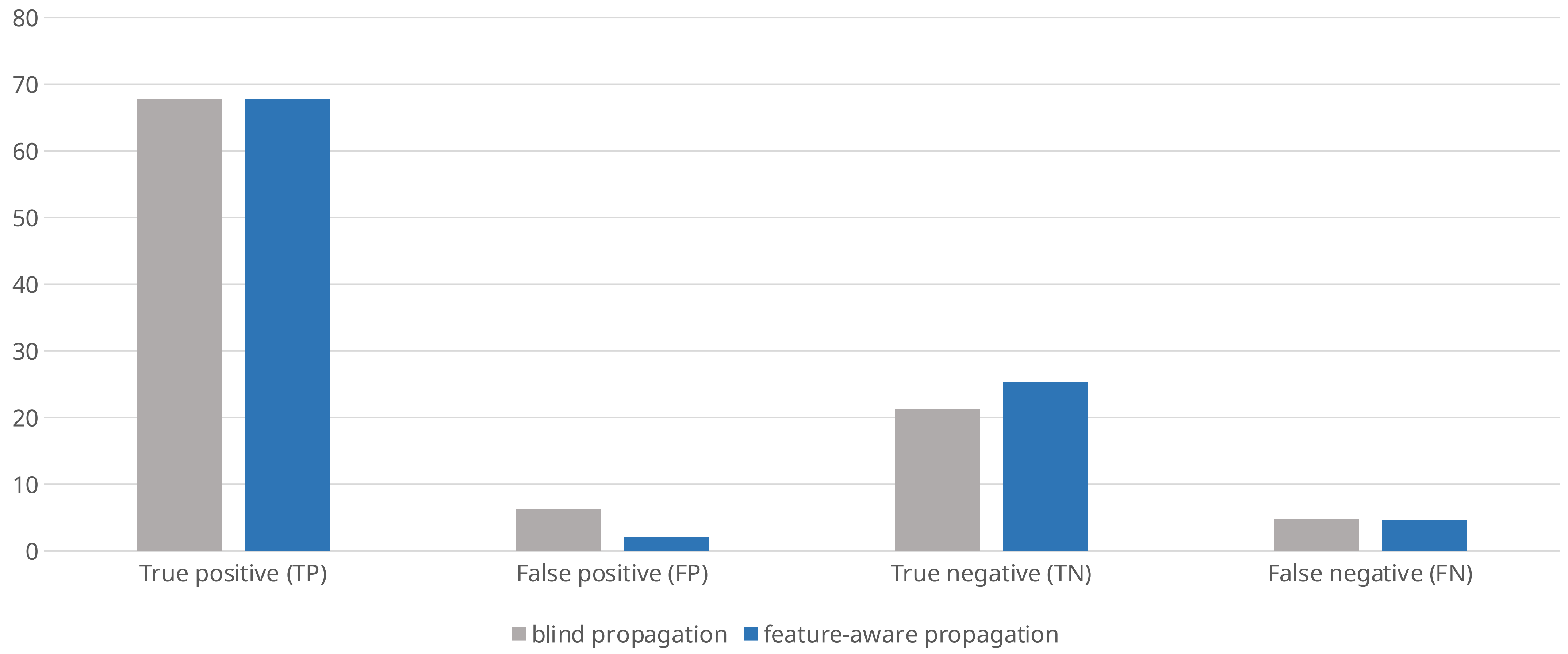


100
80
60
40
20
0
87,6
8
4,4
Applied Patches
Correct (TP)
Invalid (FP)
Wrong Location (FN)
93,7
6,3
Failed Patches
Not required (TN)
Missing (FN)

Evaluation of patch outcomes



RQ3: Impact of considering knowledge about features



The VariantSync project

Central Goal:

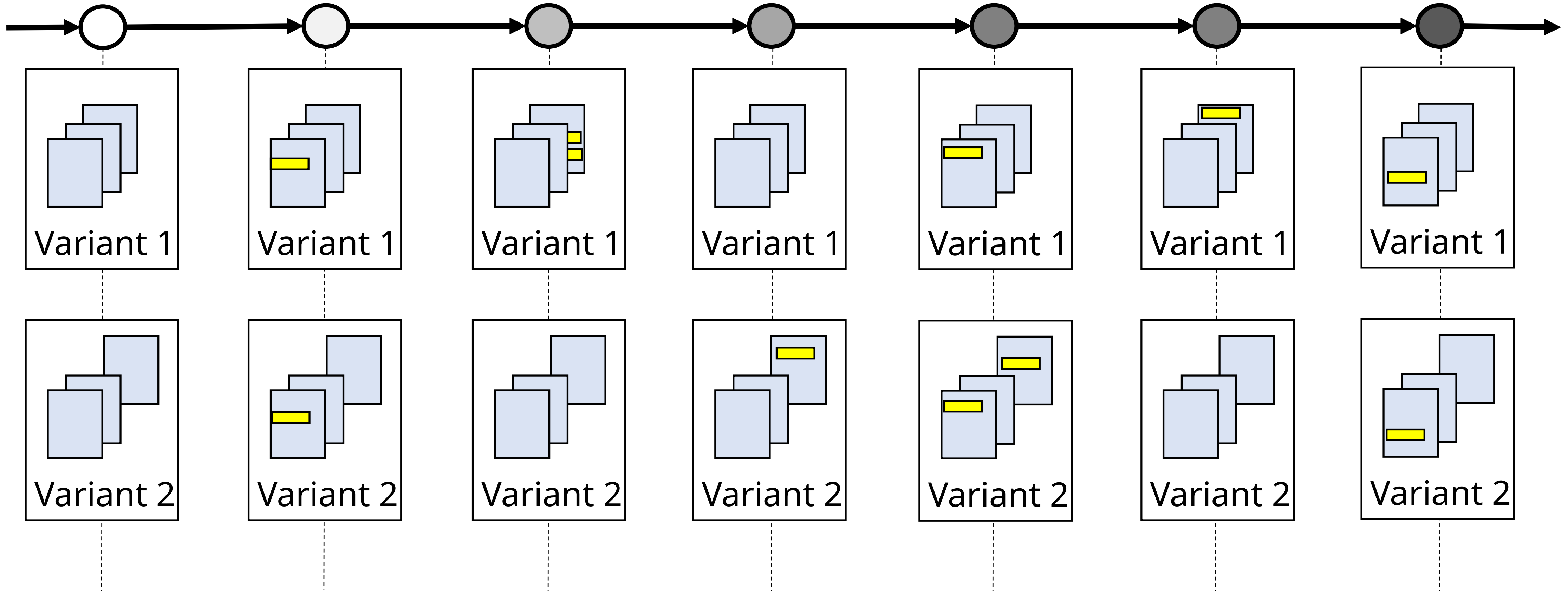
- Automatic synchronization of variants

Basic Idea:

- Trace features to their implementation
- Propagate feature changes to other variants automatically

Simulation of Automated Change Propagation

Not all variants are changed in a commit


Variant 1
Variant 1
Variant 1
Variant 1
Variant 1
Variant 1
Variant 1
Variant 2
Variant 2
Variant 2
Variant 2
Variant 2
Variant 2
Variant 2