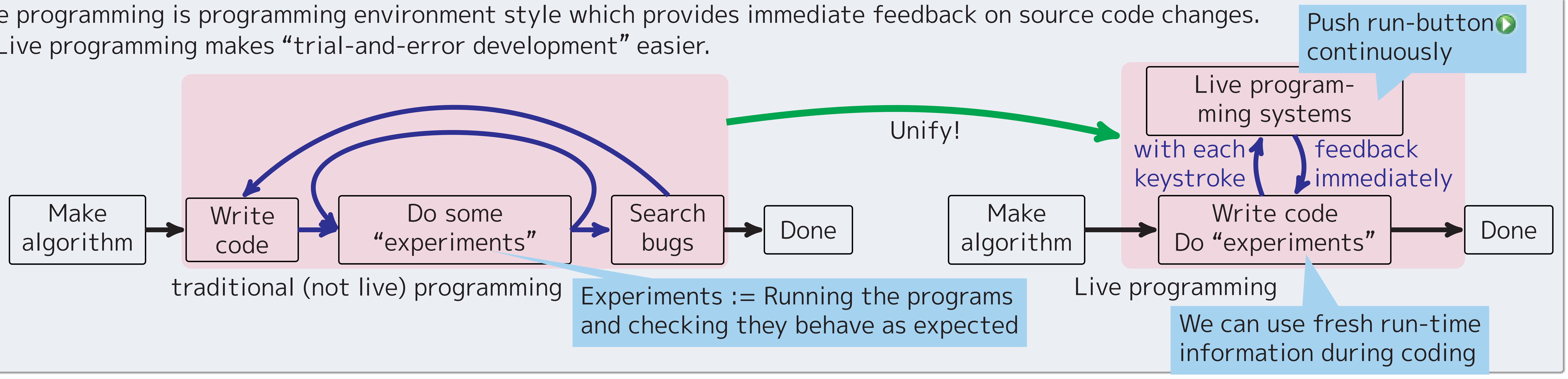
# Shiranui: a Live Programming with Support for Unit Testing

Tomoki Imai Hidehiko Masuhara Tomoyuki Aotani
(Tokyo Institute of Technology, Japan)

## Background: What is a Live Programming Environment?

Live programming is programming environment style which provides immediate feedback on source code changes.

- Live programming makes "trial-and-error development" easier.




## Our Motivation: Using Live Programming in Practical Programming

Currently, live programming environments are mainly used for:

- running samples,
- programming very small projects,
- checking functions' behavior.

We want to use live programming environments in practical programming which require:

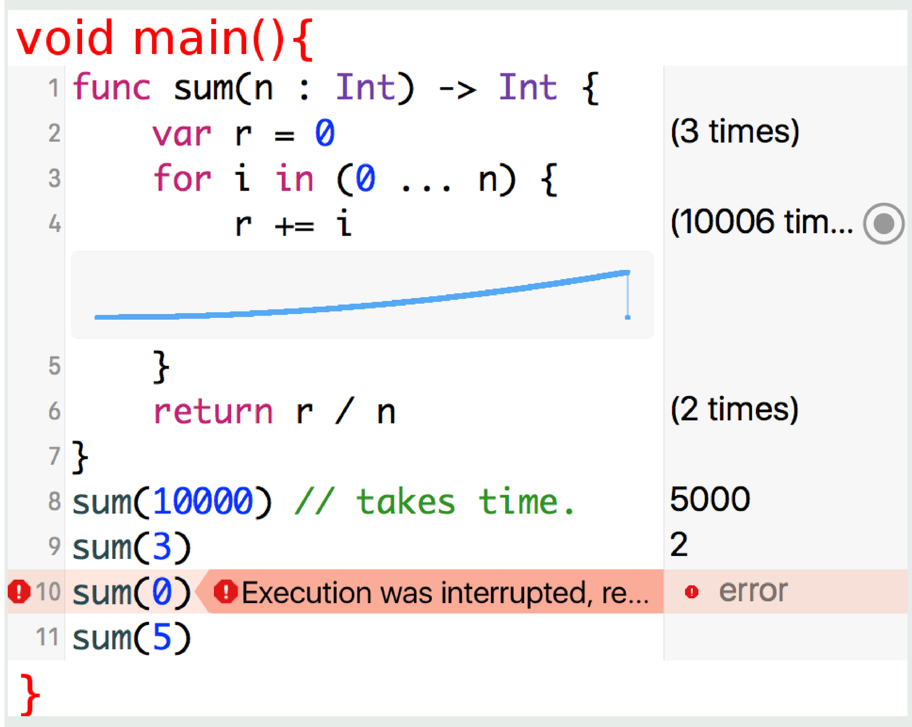
- many functions or submodules,
- ensuring that the program works well.

But, existing live programming environments has three problems.

## Problem 1: Single Entry Point

Existing live programming environments have only one entry point.

It is like a big "main" function.




It causes:

- long feedback loop,
  - ex. We cannot get sum(3)'s feedback before sum(10000).
- complex runtime log,
  - ex. sum(10000)'s log and sum(3)'s one are merged.
- lost feedback.
  - ex. sum(0) causes error, and sum(5)'s feedback is lost.

=>

Not suitable for large programs.

## Solution 1: Isolated Execution Point

Shiranui executes some parts of programs in isolated interpreters.

- Faster feedback by parallel execution
- Simpler execution logs
- Errors are not propagated to another executions.

```
#+ sum(10000) -> 5000;
#+ sum(3) -> 2;
#+ sum(0) -> "Division by 0";
#+ sum(5) -> 3;
let sum = \(n){
    let r = ref 0;
    for i in [1 .. n] {
        #* i -> 1,2,3,4,5;
        r <- !r + i;
    }
    return !r / n;
};
```

Execution flow:

1. Duplicate programs for each isolated execution point (L:1,2,3,4).
2. Run programs parallelly and record logs separately.
3. Give feedback to users.

```
let sum = \(n){
// copied body
};
sum(10000);
```

```
let sum = \(n){
// copied body
};
sum(3);
```

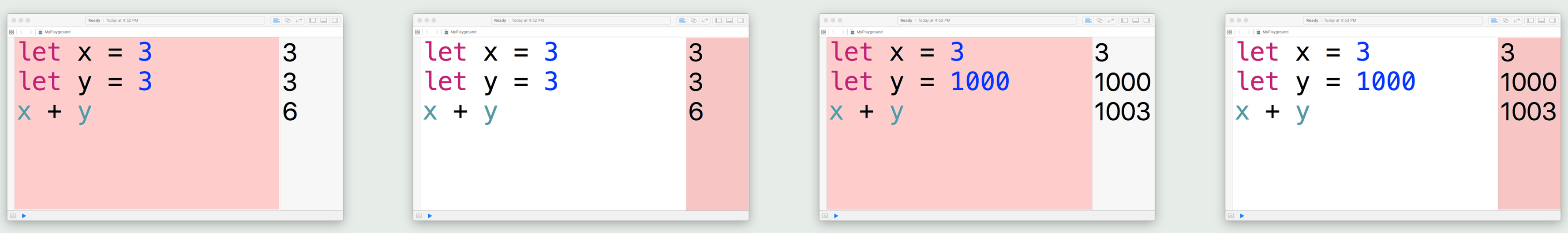
```
let sum = \(n){
// copied body
};
sum(0);
```

```
let sum = \(n){
// copied body
};
sum(5);
```

## Problem 2: No Support for Testing Frameworks

"Tests" in live programming environments are transient.

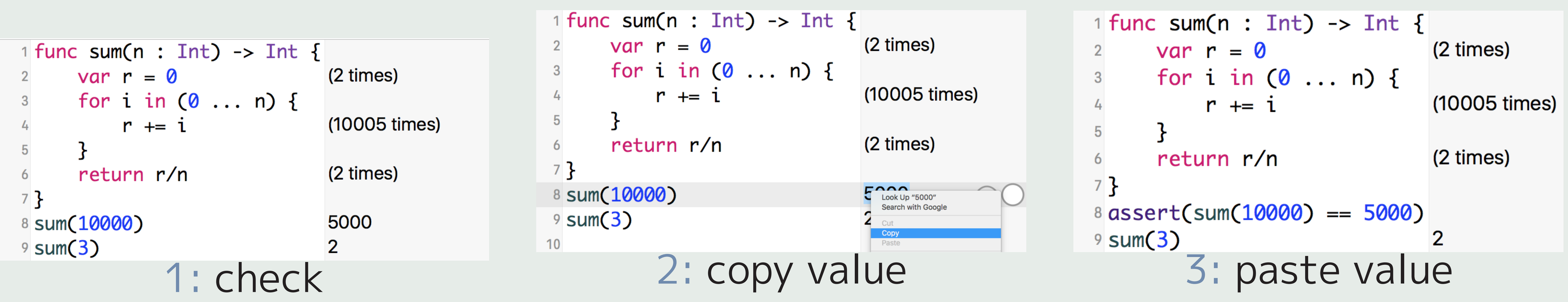
- We need to check all return values ourselves when the source code changes.




1: coding 2: checking 3: coding 4: checking

If we add testing frameworks like JUnit, "liveness" is lost.

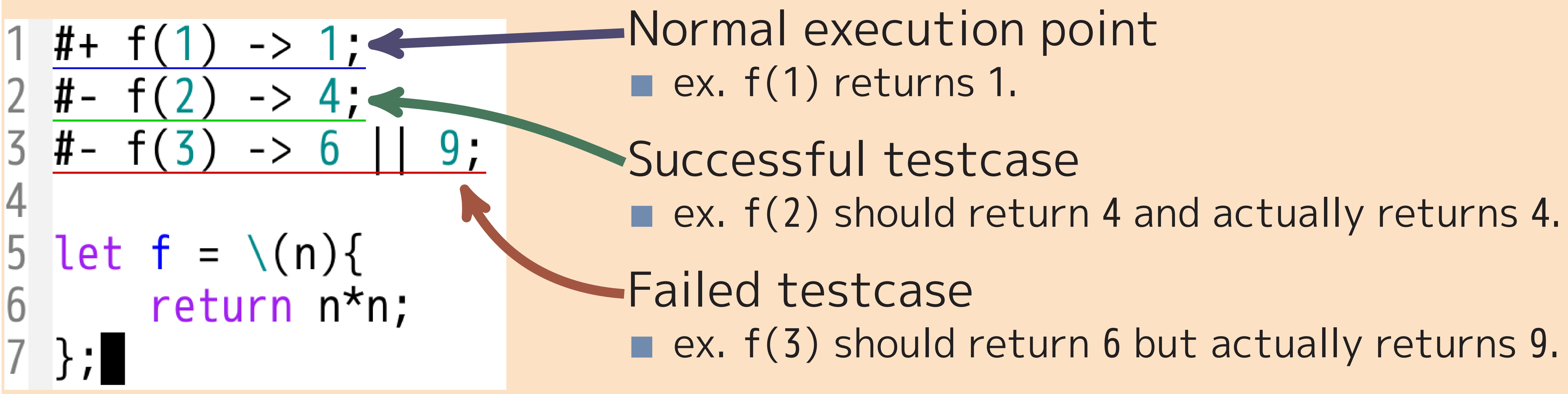
- Constructing expected values takes time.
- Promoting transient tests to persistent tests also takes time.



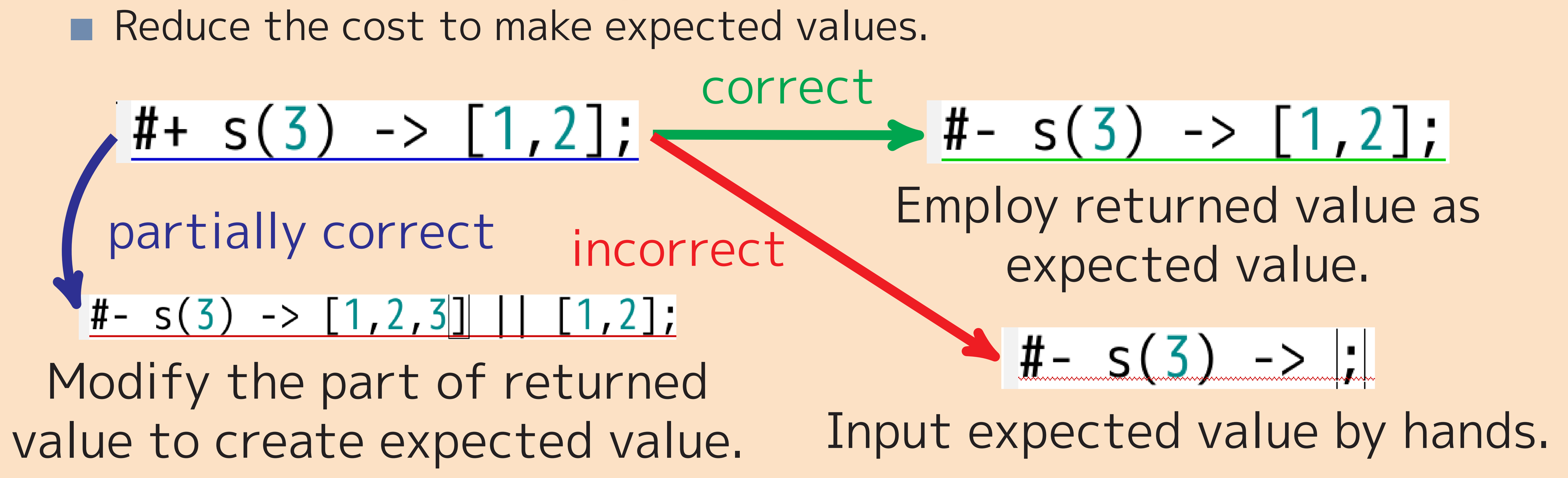

1: check 2: copy value 3: paste value

## Solution 2: Integrated Unit Testing Features

Unit testcases are expressed as isolated execution points.




- Shortcuts for promoting experiments to unit testcases.
  - Reduce the cost to make expected values.




## Problem 3: No Support to Make Small Sub Problems

When debugging programs, we must create small programs, which reproduce the problems. Live programming combines "editing" and "debugging", but do not support to make small sub problems.

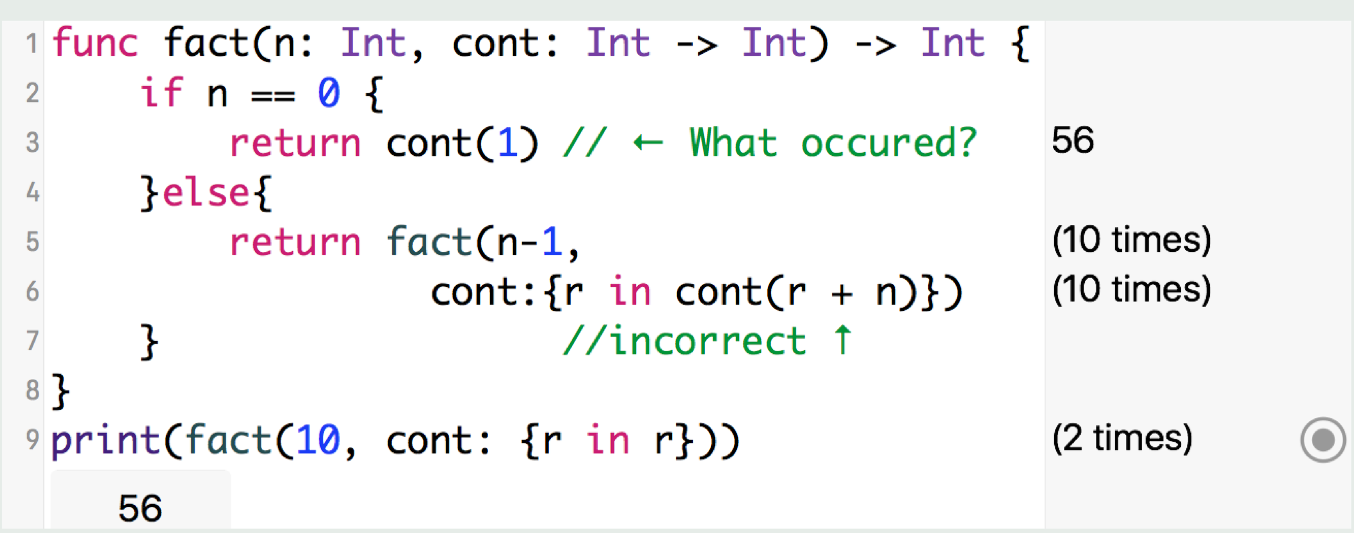
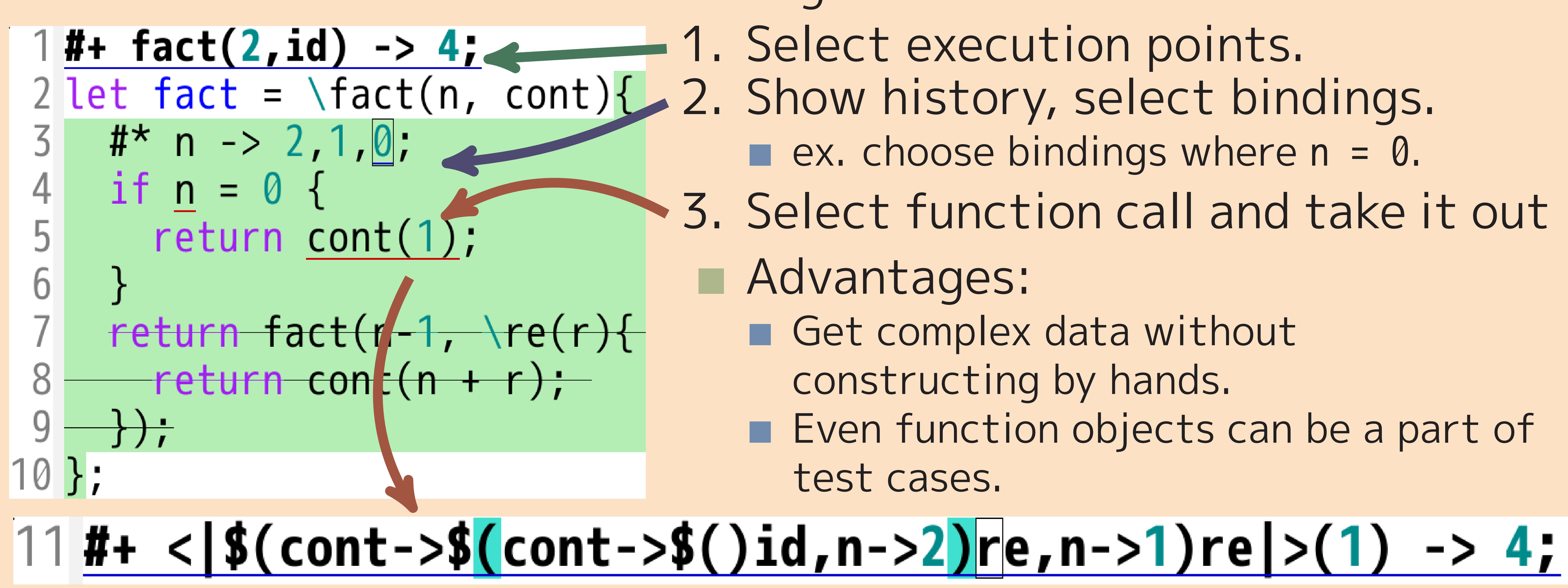



Figure: cont(1) has strange behaviors, but we cannot debug it directly.

## Solution 3: Take Out Function Call From Runtime Log

Shiranui enables user to take a function call from a runtime log.

- We can generate small sub problems by taking out function call, which seems to cause the wrong result.




1. Select execution points.
2. Show history, select bindings.
   - ex. choose bindings where n = 0.
3. Select function call and take it out
   - Advantages:
     - Get complex data without constructing by hands.
     - Even function objects can be a part of test cases.