

# FIRST INFRASTRUCTURE AND EXPERIMENTATION IN ECHO-DEBUGGING

Thomas Dupriez

*Université de Lille, Inria, CNRS,  
Central Lille, UMR 9189 – CRISTAL  
France*

Steven Costiou

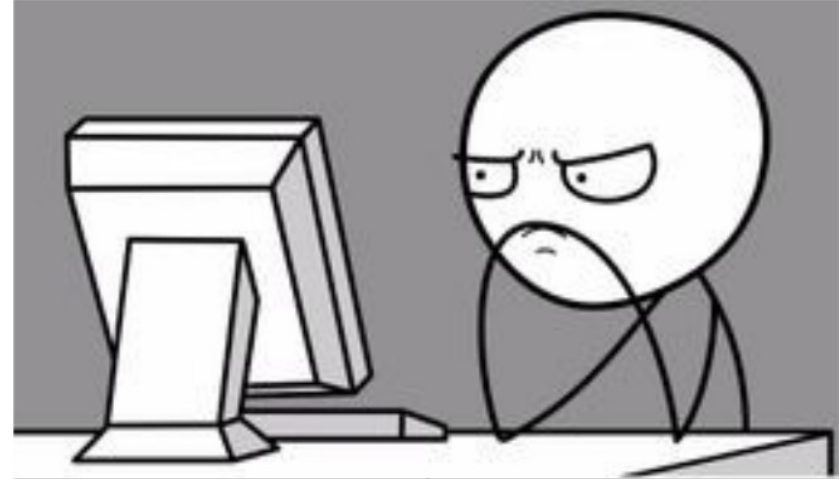
*Université de Lille, Inria, CNRS,  
Central Lille, UMR 9189 – CRISTAL  
France*

Stéphane Ducasse

*Université de Lille, Inria, CNRS,  
Central Lille, UMR 9189 – CRISTAL  
France*

# MOTIVATION

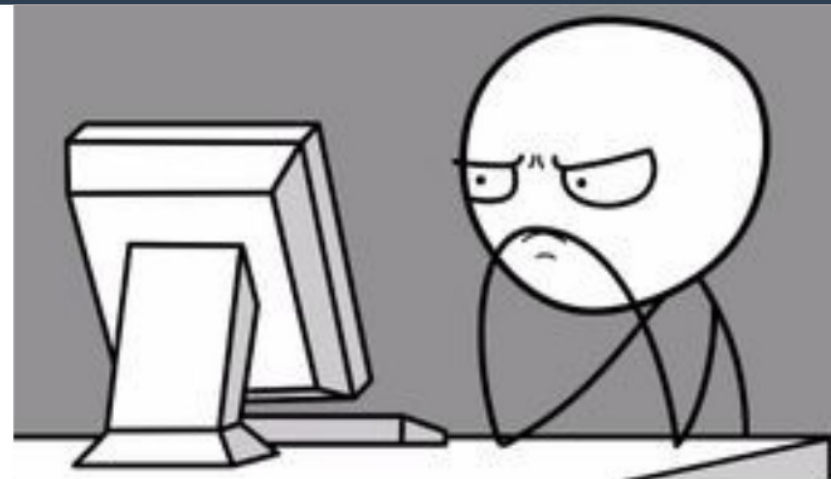
# MOTIVATION



John

# MOTIVATION

John is a serious developer.  
His code is good, and the tests are green.

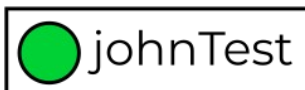


John

Good code



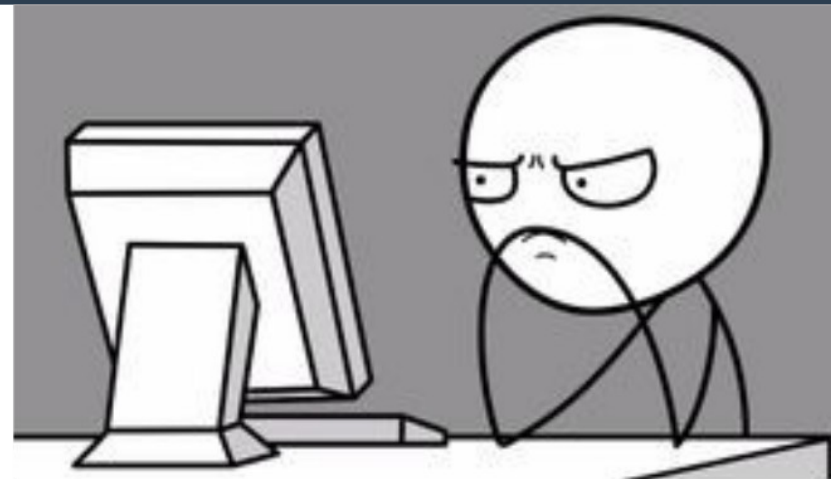
Green test



# MOTIVATION

John is a serious developer.  
His code is good, and the tests are green.

John improves his code to make it better.



John

Good code



Better code



Green test



# MOTIVATION

John is a serious developer.  
His code is good, and the tests are green.

John improves his code to make it better.

But suddenly, the tests are red.



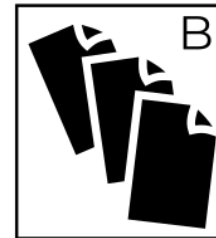
*Good code*



*Green test*



*Better code*



*Red test*



# MOTIVATION

John is a serious developer.  
His code is good, and the tests are green.

John improves his code to make it better.

But suddenly, the tests are red.

John does not understand why his change  
caused the test to fail.



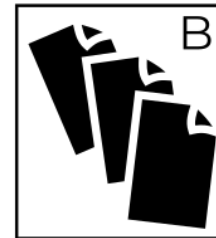
Good code



Green test



Better code



Red test



# MOTIVATION

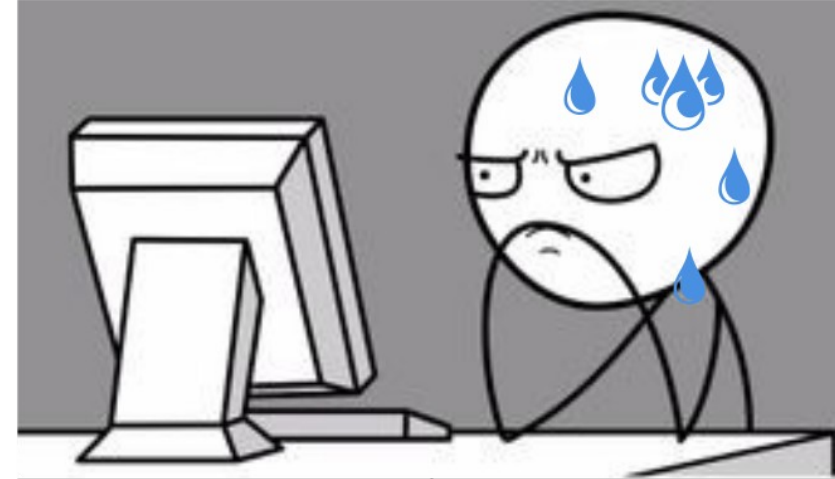
John is a serious developer.  
His code is good, and the tests are green.

John improves his code to make it better.

But suddenly, the tests are red.

John does not understand why his change  
caused the test to fail.

So John makes the only sensible decision...



John

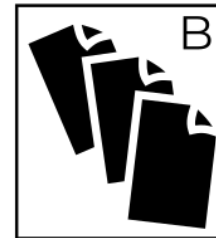
Good code



Green test



Better code



Red test





# MOTIVATION

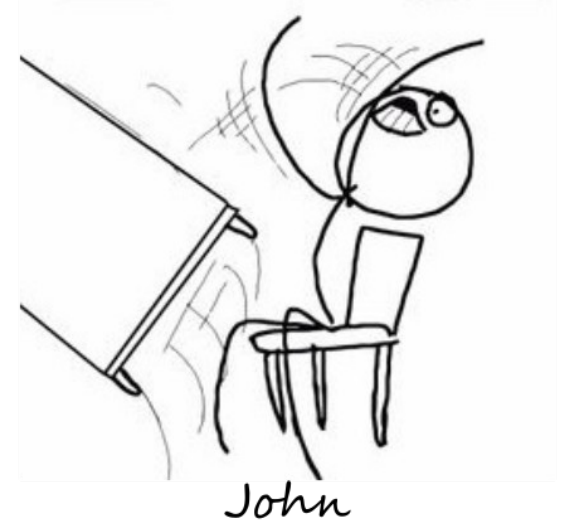
John is a serious developer.  
His code is good, and the tests are green.

John improves his code to make it better.

But suddenly, the tests are red.

John does not understand why his change  
caused the test to fail.

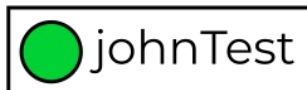
So John makes the only sensible decision...



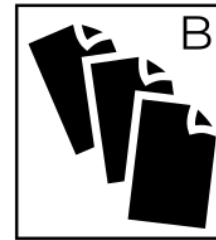
*Good code*



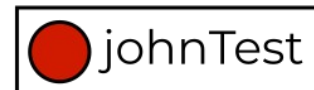
*Green test*



*Better code*



*Red test*



# MOTIVATION

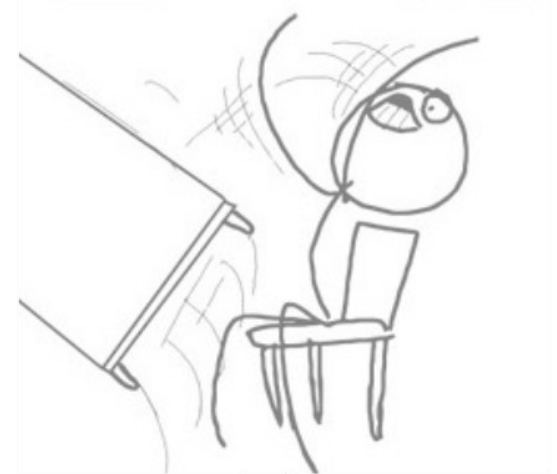
John is a serious developer.  
His code is good, and the tests are green.

John improves his code to make it better

But suddenly, the tests are red.

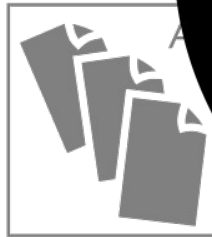
John does not understand why his change  
caused the test to fail.

So John makes the only sensible decision...



John

Good code



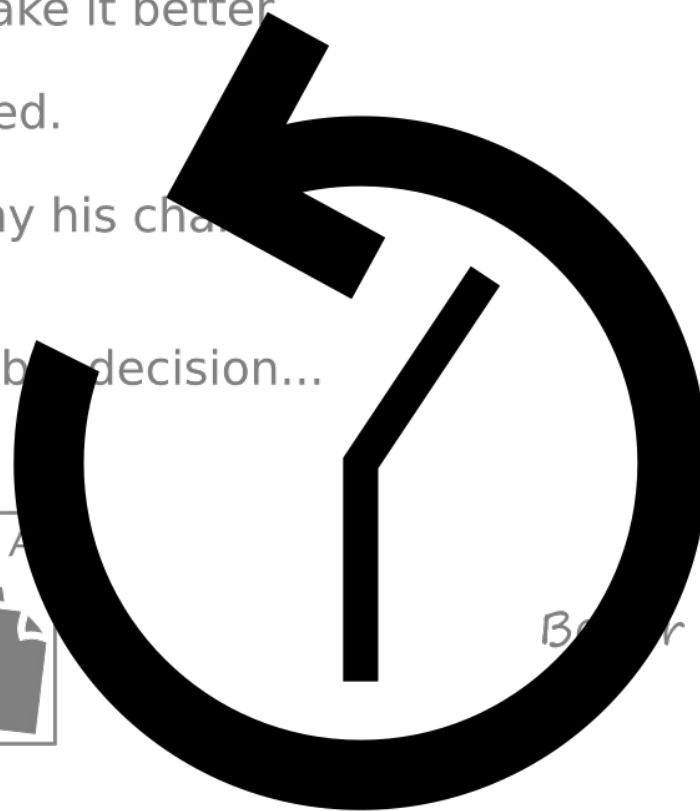
Green test



Better code



Red test



# MOTIVATION

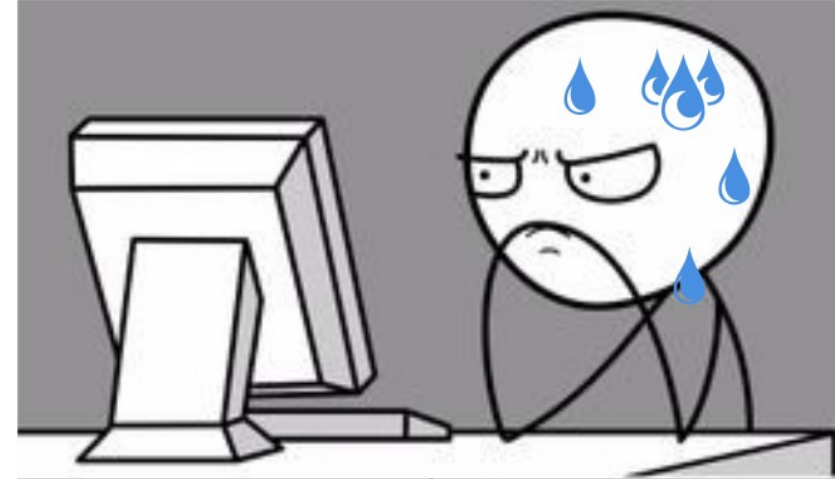
John is a serious developer.  
His code is good, and the tests are green.

John improves his code to make it better.

But suddenly, the tests are red.

John does not understand why his change  
caused the test to fail.

So John makes the only sensible decision...



John

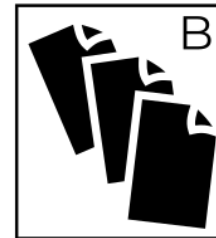
Good code



Green test



Better code



Red test



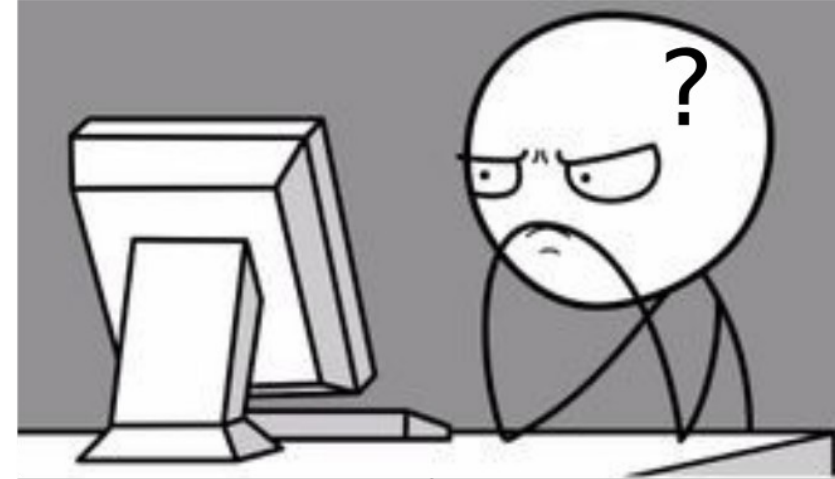
# MOTIVATION

John is a serious developer.  
His code is good, and the tests are green.

John improves his code to make it better.

But suddenly, the tests are red.

John does not understand why his change  
caused the test to fail.



John

So John makes the only sensible decision...

**Using the Echo-debugger !**

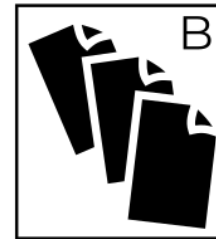
*Good code*



*Green test*



*Better code*



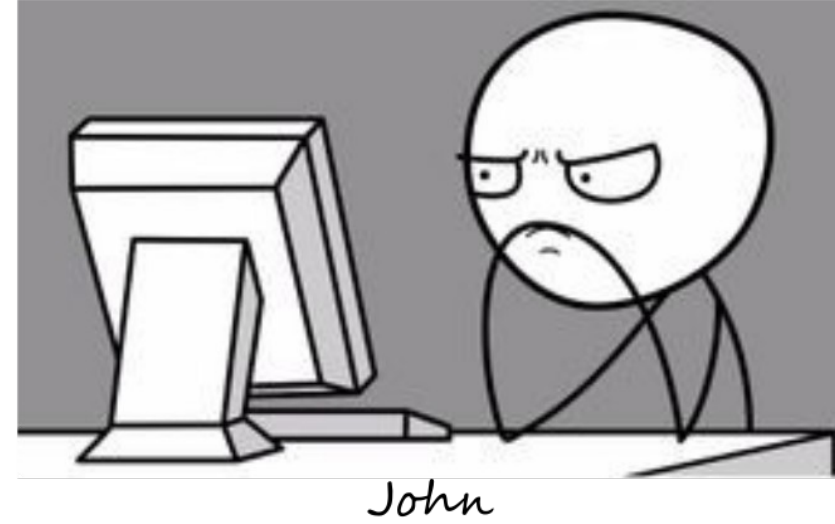
*Red test*



# MOTIVATION

The Echo-debugger lets John:

1) Debug both the green test and red test execution in parallel.

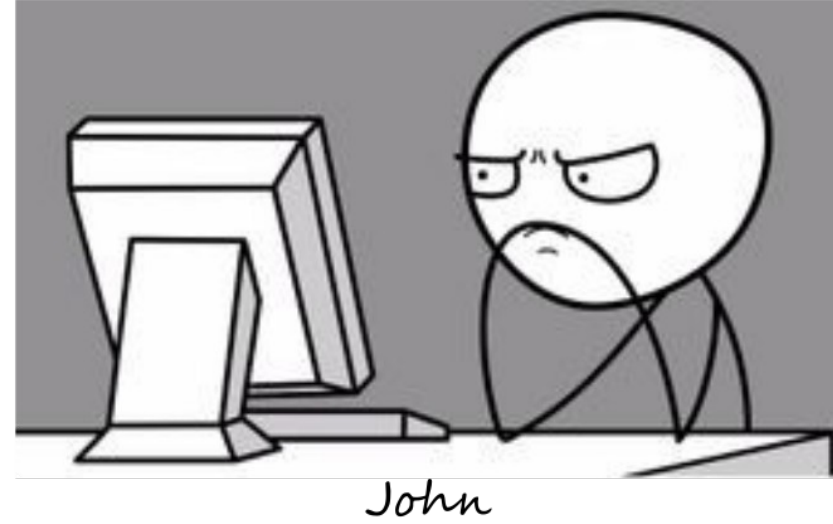


# MOTIVATION

The Echo-debugger lets John:

1) Debug both the green test and red test execution in parallel.

*Even though they are on different versions of the program.*



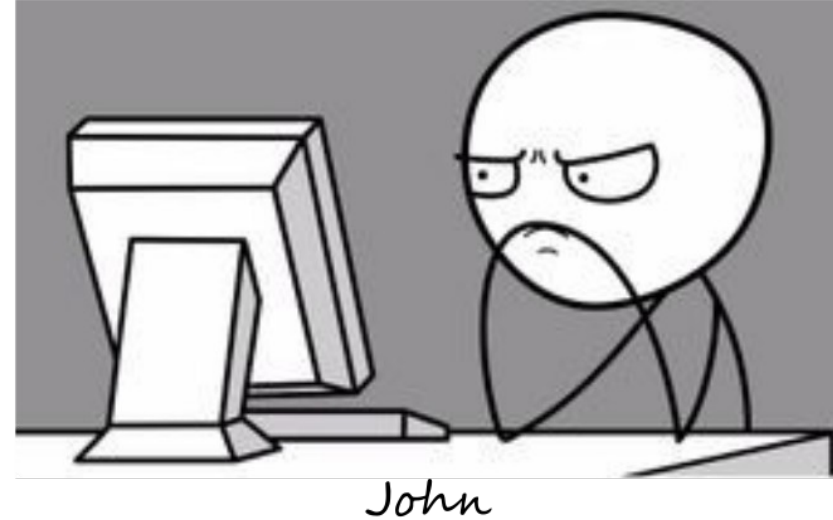
# MOTIVATION

The Echo-debugger lets John:

1) Debug both the green test and red test execution in parallel.

*Even though they are on different versions of the program.*

**=> The Echo-Debugger and its Architecture**



# MOTIVATION

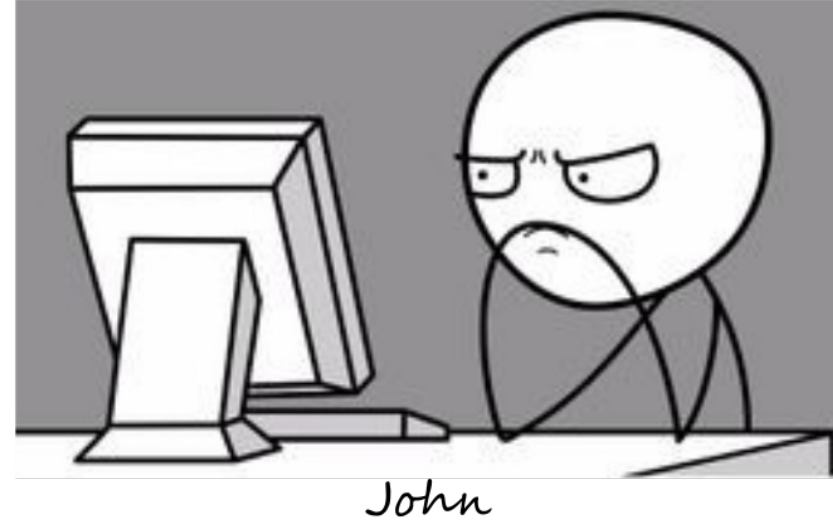
The Echo-debugger lets John:

1) Debug both the green test and red test execution in parallel.

*Even though they are on different versions of the program.*

## => The Echo-Debugger and its Architecture

2) Automatically find the control-flow divergences and convergences between the executions.





# MOTIVATION

The Echo-debugger lets John:

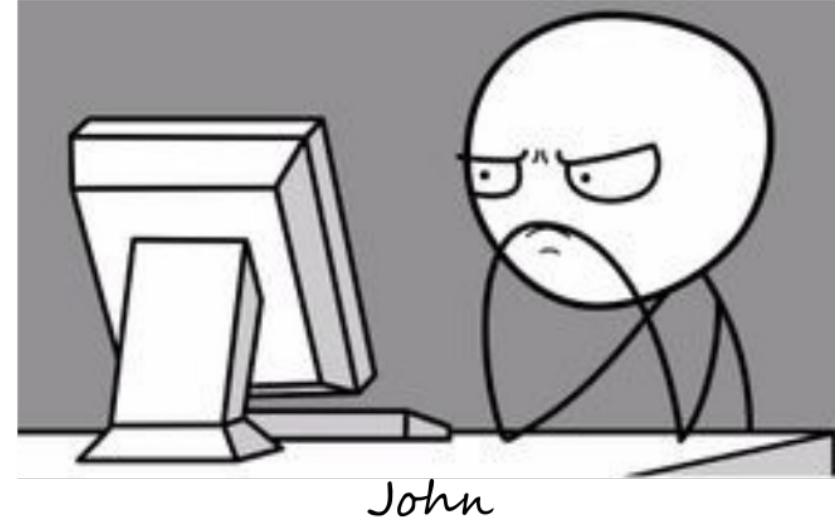
1) Debug both the green test and red test execution in parallel.

*Even though they are on different versions of the program.*

## => The Echo-Debugger and its Architecture

2) Automatically find the control-flow divergences and convergences between the executions.

*And jump the echo-debugger to them if the executions are deterministic*



# MOTIVATION

The Echo-debugger lets John:

1) Debug both the green test and red test execution in parallel.

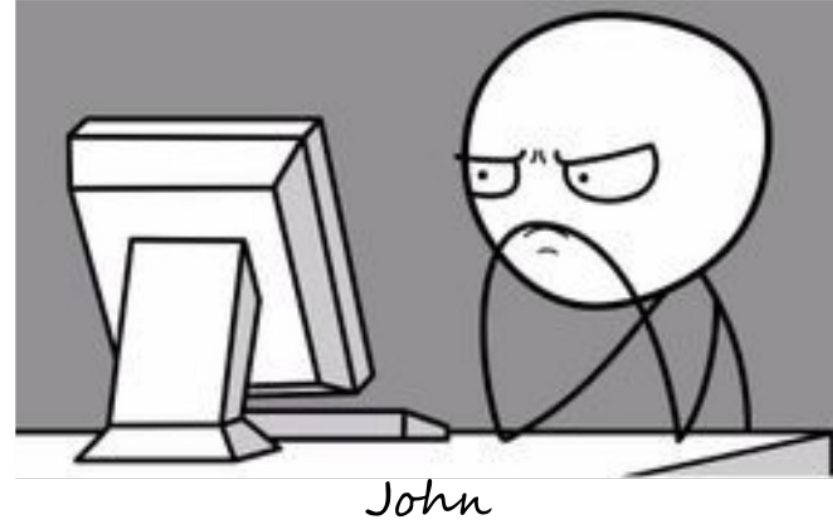
*Even though they are on different versions of the program.*

**=> The Echo-Debugger and its Architecture**

2) Automatically find the control-flow divergences and convergences between the executions.

*And jump the echo-debugger to them if the executions are deterministic*

**=> The CDM (Convergence Divergence Mapping) Algorithm**



# MOTIVATION

The Echo-debugger lets John:

1) Debug both the green test and red test execution in parallel.

*Even though they are on different versions of the program.*

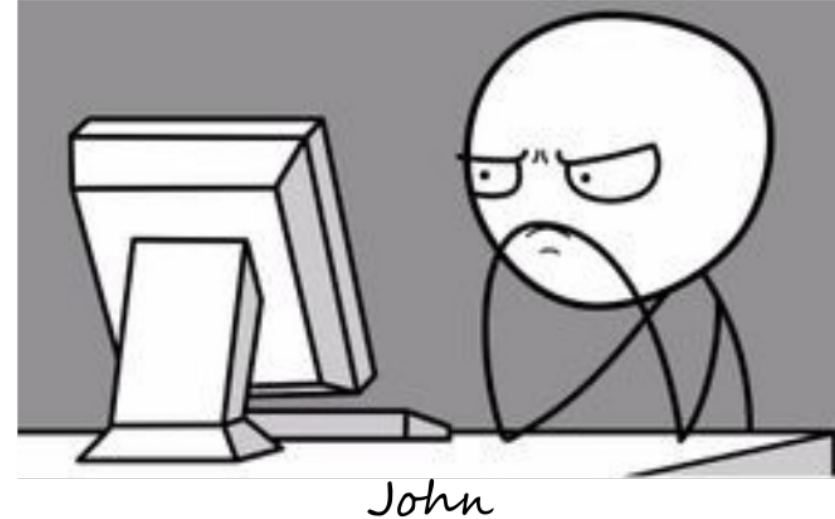
## => The Echo-Debugger and its Architecture

2) Automatically find the control-flow divergences and convergences between the executions.

*And jump the echo-debugger to them if the executions are deterministic*

## => The CDM (Convergence Divergence Mapping) Algorithm

*... Thanks to the Echo-debugger, John understood why his code change made the test red, and was able to fix the issue.*



# MOTIVATION

The Echo-debugger lets John:

1) Debug both the green test and red test execution in parallel.

*Even though they are on different versions of the program.*

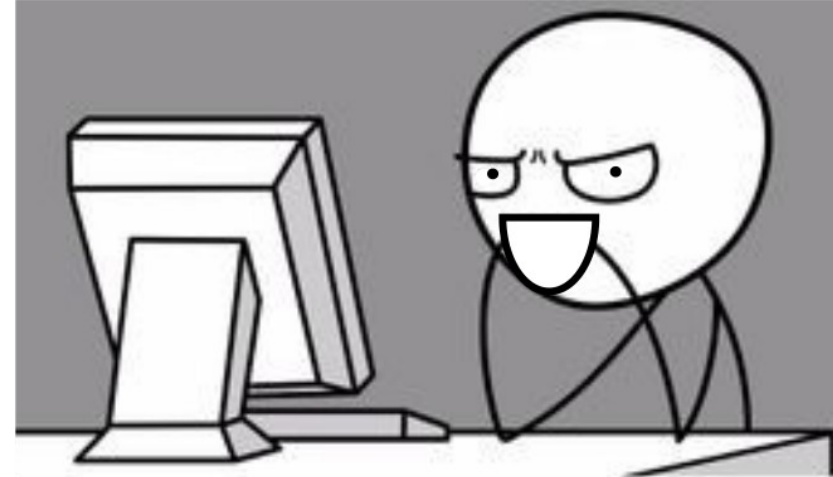
## => The Echo-Debugger and its Architecture

2) Automatically find the control-flow divergences and convergences between the executions.

*And jump the echo-debugger to them if the executions are deterministic*

## => The CDM (Convergence Divergence Mapping) Algorithm

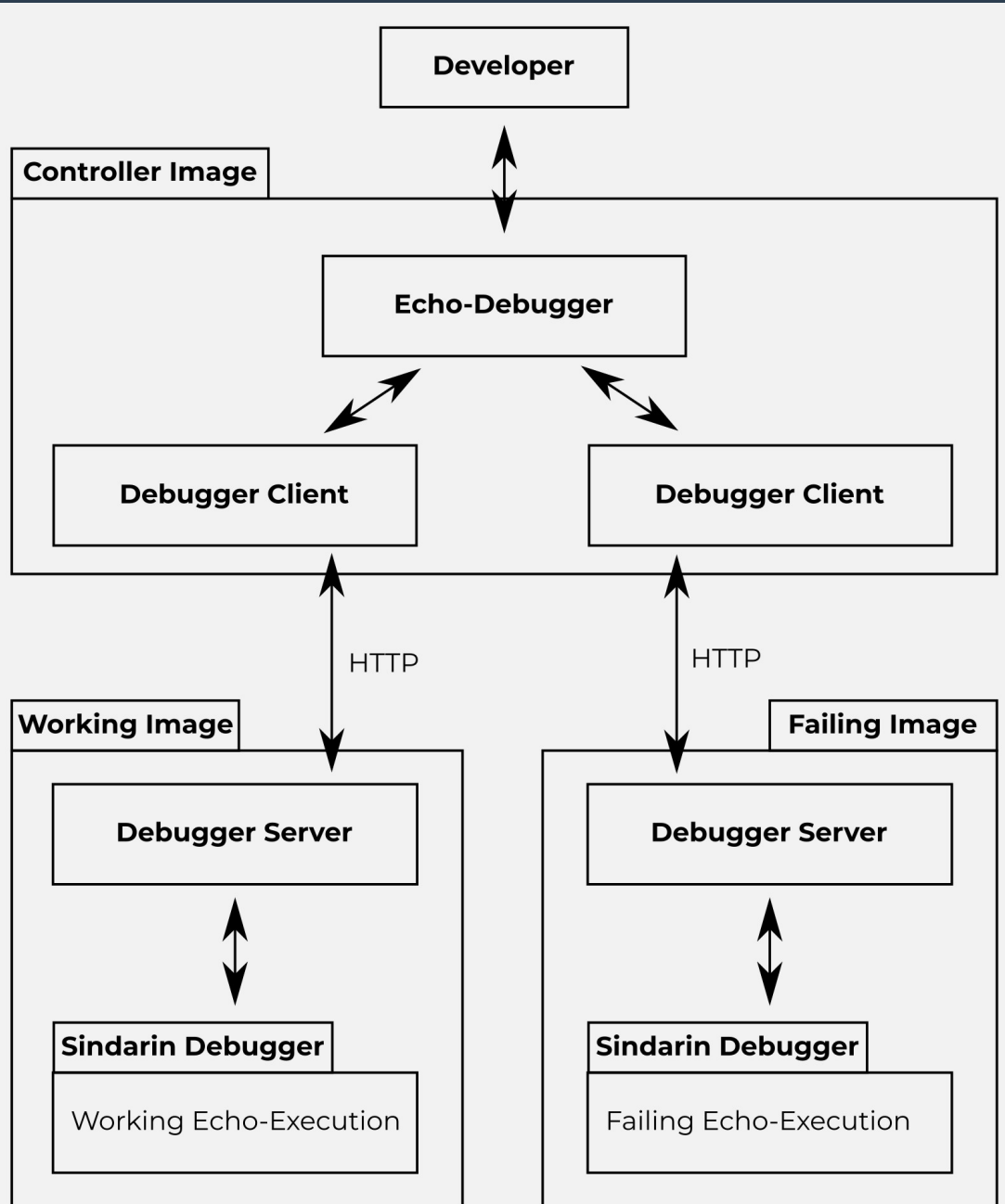
*... Thanks to the Echo-debugger, John understood why his code change made the test red, and was able to fix the issue.*



John

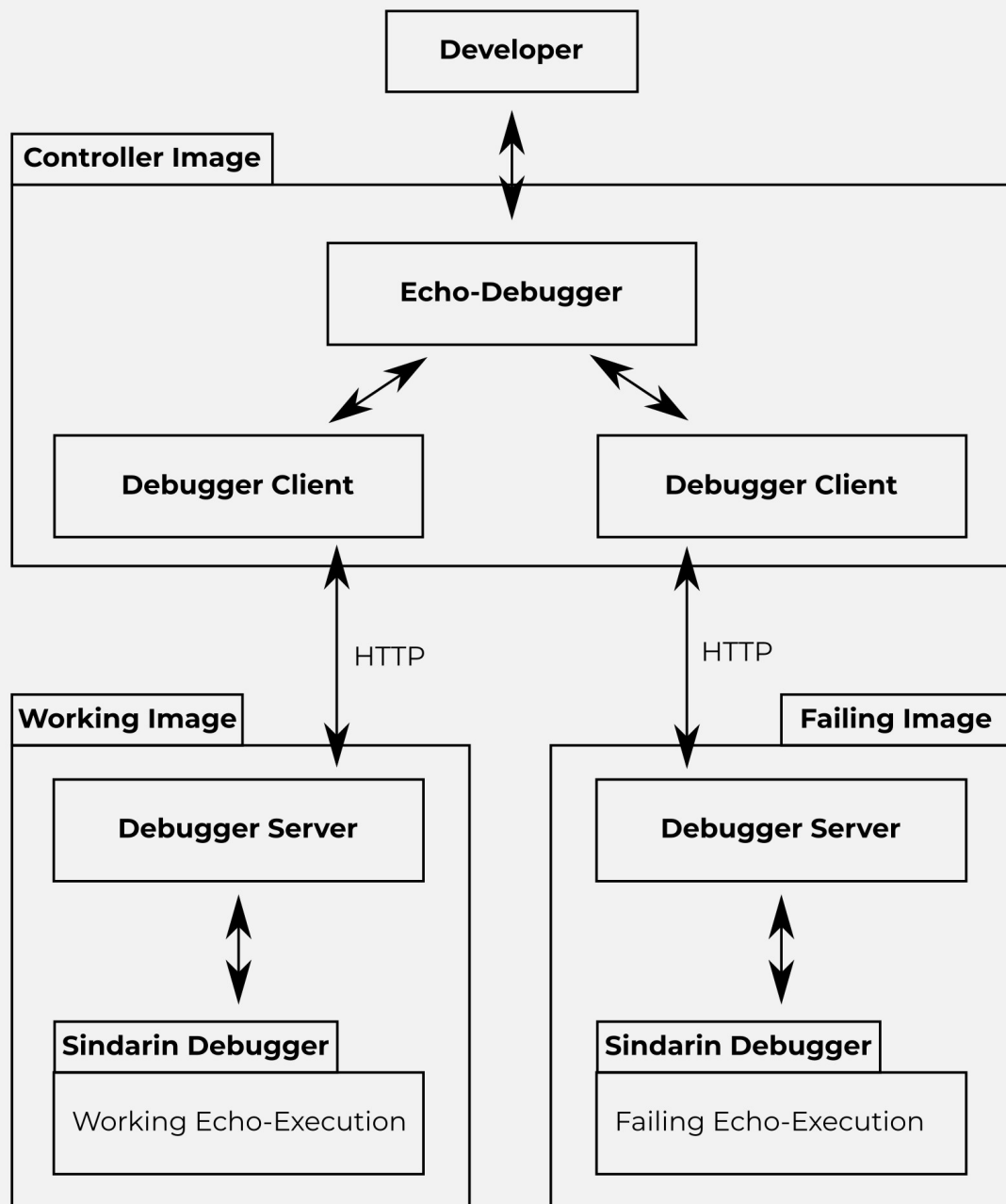
*John is now a happy serious developer!*

# ECHO-DEBUGGER: ARCHITECTURE



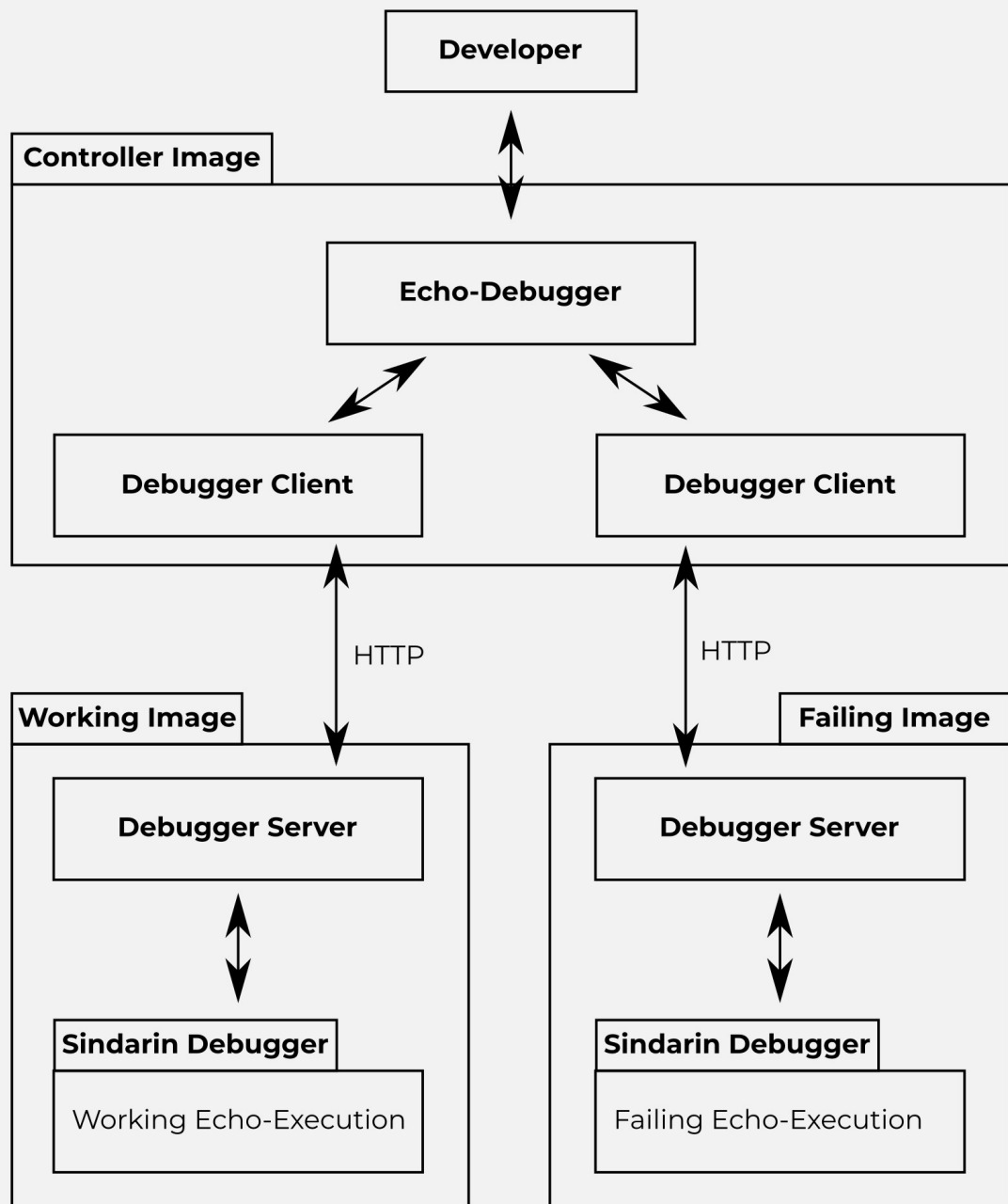
# ECHO-DEBUGGER: ARCHITECTURE

- 3 runtimes

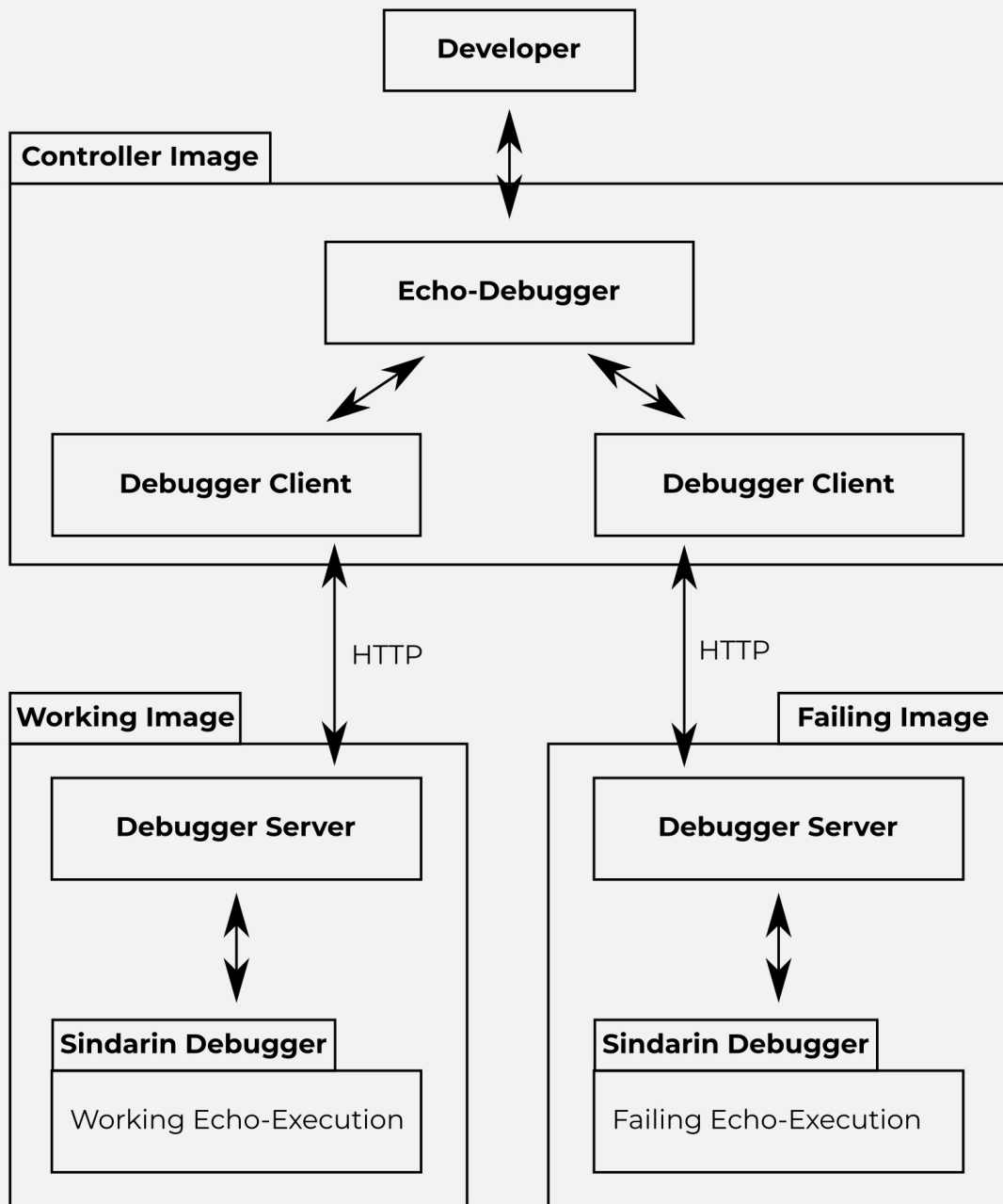


# ECHO-DEBUGGER: ARCHITECTURE

- 3 runtimes
- HTTP client/server communication



# ECHO-DEBUGGER: ARCHITECTURE



- 3 runtimes
- HTTP client/server communication
- Sindarin (scriptable) debuggers to control the echo-executions



# ECHO-DEBUGGER: CONTROLLER UI

Working Execution  
port  
1234  
Connect

Inspect  
Refresh

stack

Class	Selector
TestAsserter	assert:equals:
PCBTest	testChildConfigurationLooksUpParentConfigurat
UndefinedObject	Dolt
BlockClosure	newProcess

context

key	value
class	TestAsserter
method	assert: actual equals: expected "This method raises an AssertionFailure if actual is different (using #= message) from expected.
node	a RBProgramNodeRepresentation(#class->'TestAsserter' #methodSelector->#assert:eq
receiver	PCBTest
selector	assert:equals:

node (code) node (raw)

```
2  This method raises an AssertionFailure if actual is
3  different (using #= message) from expected.
4  Else it does nothing and execution continues.
5  "
6  ^ self
7  assert: actual = expected
8  description: [self comparingStringBetween: actual and:
expected]
```

Failing Execution  
port  
5678  
Connect

Inspect  
Refresh

stack

Class	Selector
Object	=
TestAsserter	assert:equals:
PCBTest	testChildConfigurationLooksUpParentConfigurat
UndefinedObject	Dolt
BlockClosure	newProcess

context

key	value
class	Object
method	= anObject "Answer whether the receiver and the argument represent the same object.
node	a RBProgramNodeRepresentation(#class->'Object' #methodSelector->#=#methodSource
receiver	nil
selector	=

node (code) node (raw)

```
1  = anObject
2  "Answer whether the receiver and the argument represent the
3  same
4  object. If = is redefined in any subclass, consider also
5  redefining the
6  message hash."
7
8  ^self == anObject
```

Echo Debugging

status

Node=? false

Operations

Step to next divergence

Step to sender until convergence

Analyze execution

Reset Echo Debugging

Inspect Echo Debugger

Inspect Echo Debugger Presenter

Navigation

data

Nature	W Step	F Step
start	0	0
Divergence	7	7
Convergen	106	9
Divergence	115	18
Convergen	225	18
Divergence	227	20
Convergen	227	22
Divergence	230	25

Go To

Inspect Selected

status

operation

navigation

Debugger on working execution

Debugger on failing execution

Control zone

# CDM ALGORITHM: INTRODUCTION

- 2 executions in parallel (echo-executions)
- Similar, but not identical code *CDM = Convergence  
Divergence Mapping*
- Echo-executions start on the same statement

# CDM ALGORITHM: INTRODUCTION

- 2 executions in parallel (echo-executions)
- Similar, but not identical code
- Echo-executions start on the same statement
- Goal:
  - find when the control-flows diverge and converge
  - Store the number of steps it took each echo-execution to get to each divergence/convergence

*CDM = Convergence  
Divergence Mapping*

# CDM ALGORITHM: INTRODUCTION

- 2 executions in parallel (echo-executions)
- Similar, but not identical code
- Echo-executions start on the same statement
- Goal:
  - find when the control-flows diverge and converge
  - Store the number of steps it took each echo-execution to get to each divergence/convergence

*CDM = Convergence  
Divergence Mapping*

Navigation		
data		
◄ Nature	◄ W Step Index	◄ F Step Index
start	0	0
Divergence	7	7
Convergence	106	9
Divergence	115	18
Convergence	225	18
Divergence	227	20
Convergence	227	22
Divergence	230	25

Go To

*Navigation Map*

=> Ability to visit these events in the echo-debugger, by restarting the echo-executions and stepping this many times

# CDM ALGORITHM: ALGORITHM

# CDM ALGORITHM: ALGORITHM

- **1) Echo-executions start convergent**

# CDM ALGORITHM: ALGORITHM

- 1) Echo-executions start convergent
- 2) Repeat until either echo-execution is over:

# CDM ALGORITHM: ALGORITHM

- 1) Echo-executions start convergent
- 2) Repeat until either echo-execution is over:
  - a) Step to next divergence
  - b) Register divergence event



# CDM ALGORITHM: ALGORITHM

- 1) Echo-executions start convergent
- 2) Repeat until either echo-execution is over:
  - a) Step to next divergence
  - b) Register divergence event
  - c) Step to next convergence
  - d) Register convergence event

# CDM ALGORITHM: ALGORITHM

- 1) Echo-executions start convergent
- 2) Repeat until either echo-execution is over:
  - a) Step to next divergence
  - b) Register divergence event
  - c) Step to next convergence
  - d) Register convergence event

# CDM ALGORITHM: NEXT DIVERGENCE

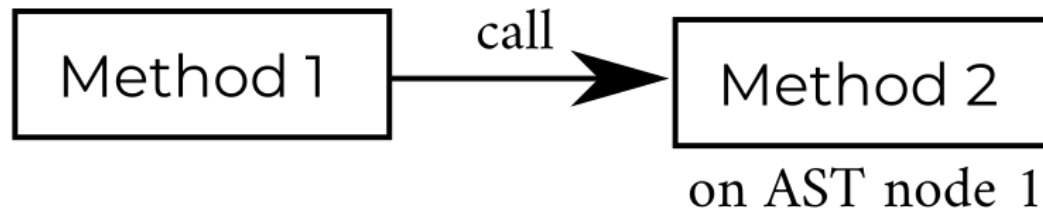
- I) Step each echo-execution once
- II) Compare their current AST node
- III) Repeat until their current AST nodes are different

# CDM ALGORITHM: NEXT CONVERGENCE

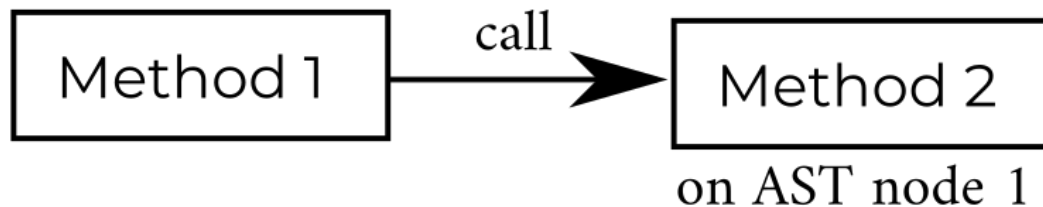
# CDM ALGORITHM: NEXT CONVERGENCE

- I) If the call stacks are not the same size, step the longest one until both call stacks are the same size

Echo-execution 1



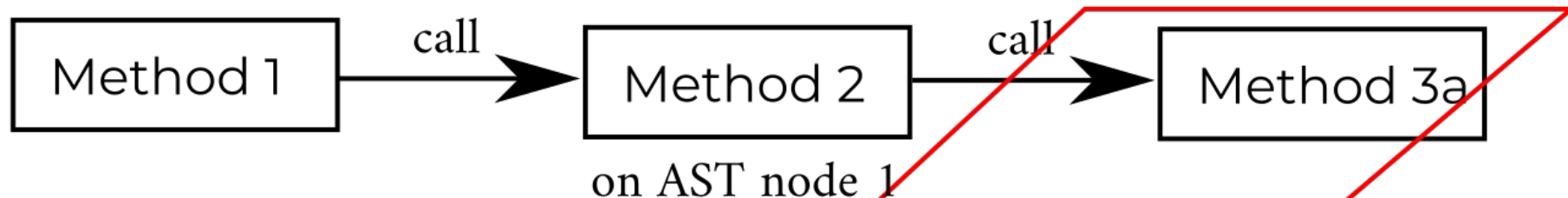
Echo-execution 2



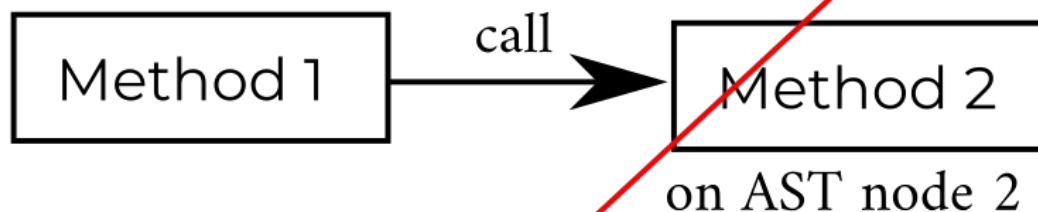
# CDM ALGORITHM: NEXT CONVERGENCE

- I) If the call stacks are not the same size, step the longest one until both call stacks are the same size

Echo-execution 1



Echo-execution 2

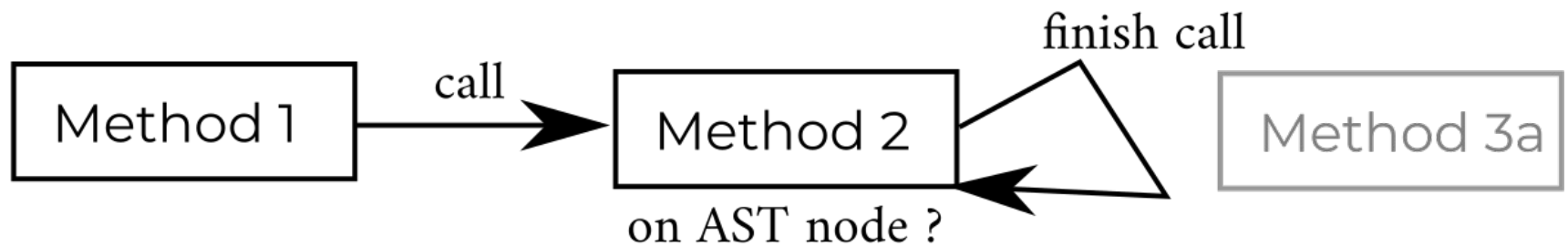


Divergence

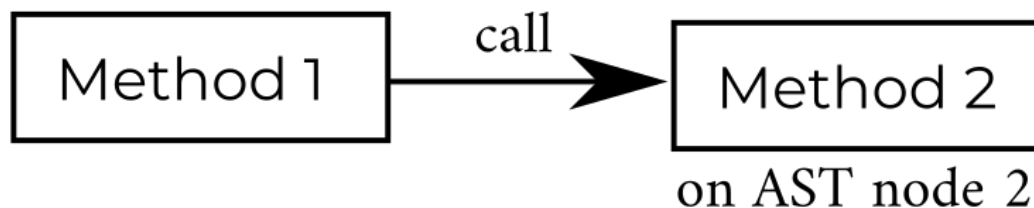
# CDM ALGORITHM: NEXT CONVERGENCE

- 1) If the call stacks are not the same size, step the longest one until both call stacks are the same size

Echo-execution 1



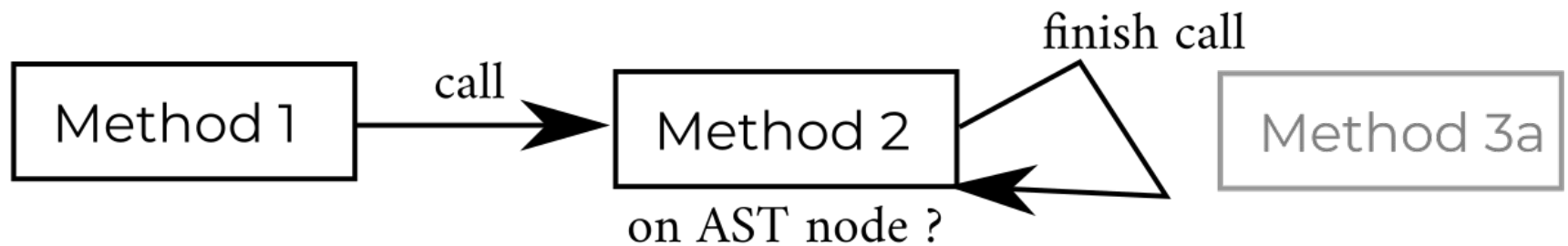
Echo-execution 2



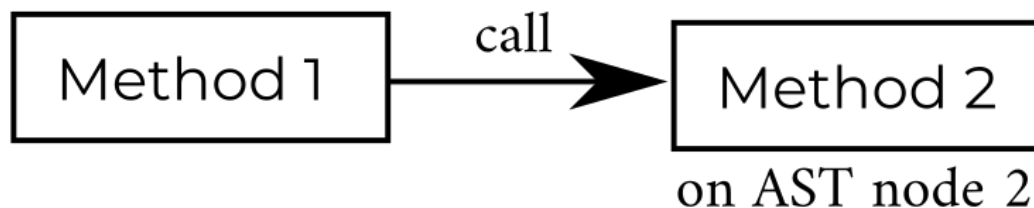
# CDM ALGORITHM: NEXT CONVERGENCE

- II) Compare the current AST nodes

Echo-execution 1



Echo-execution 2

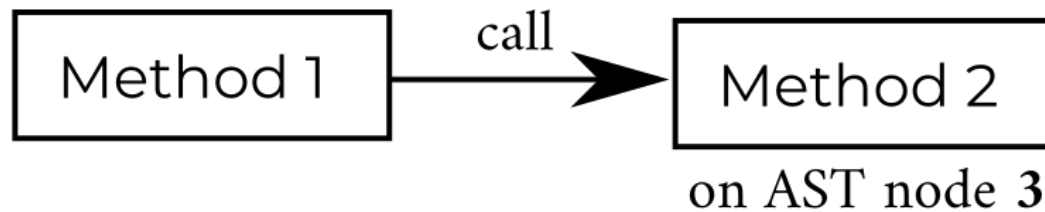




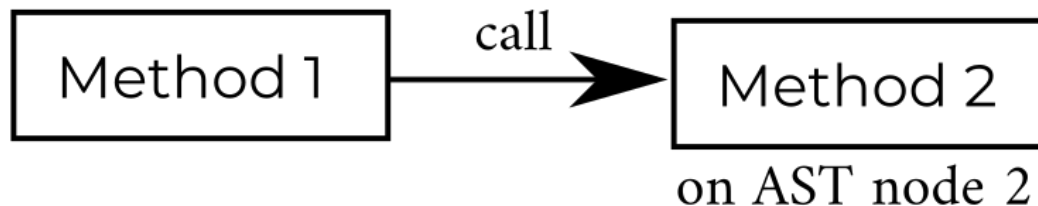
# CDM ALGORITHM: NEXT CONVERGENCE

- II) Compare the current AST nodes
- III) If different, finish the current call on both stacks and go to II)

Echo-execution 1



Echo-execution 2

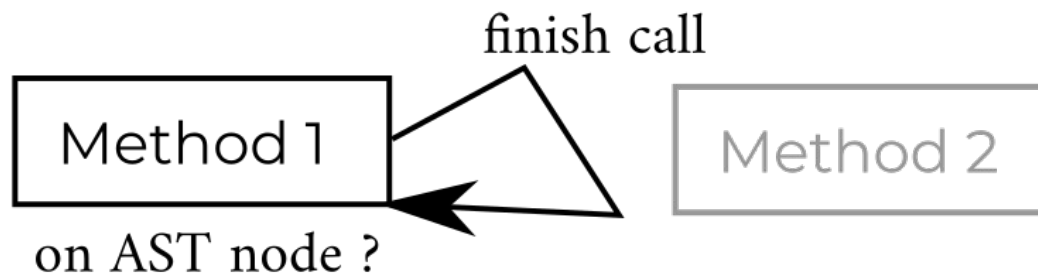


Different AST Nodes

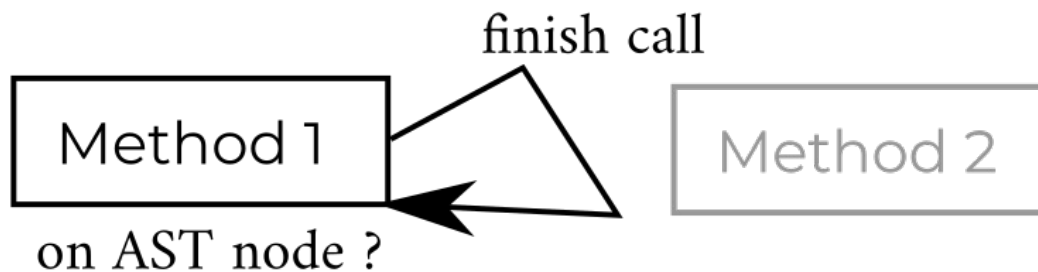
# CDM ALGORITHM: NEXT CONVERGENCE

- II) Compare the current AST nodes
- III) If different, finish the current call on both stacks and go to II)

Echo-execution 1



Echo-execution 2

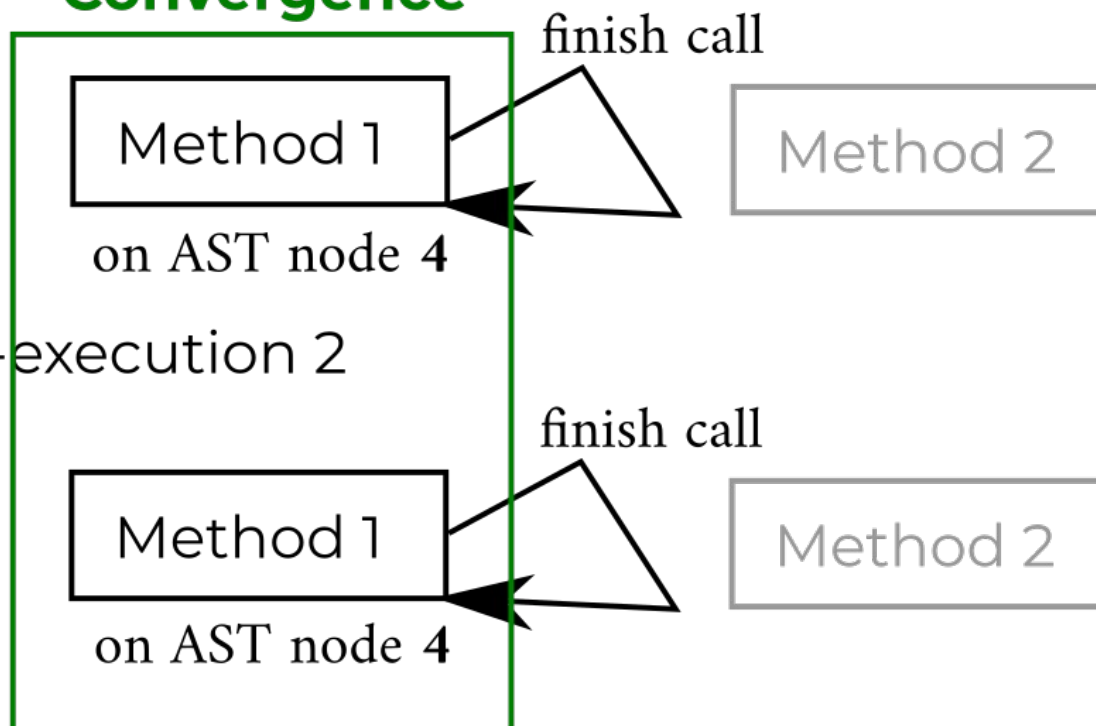


# CDM ALGORITHM: NEXT CONVERGENCE

- IV) If equal, convergence found

Echo-execution 1

**Convergence**



Same AST Nodes

# CDM ALGORITHM: NEXT CONVERGENCE

- I) If the call stacks are not the same size, step the longest one until both call stacks are the same size
- II) Compare the current AST nodes
- III) If different, finish the current call on both stacks and go to II)
- IV) If equal, convergence found

# CDM ALGORITHM: COMPARING AST NODES

# CDM ALGORITHM: COMPARING AST NODES

- With equality? (=)
  - No. Being on = AST nodes but in different methods is still a control-flow divergence

# CDM ALGORITHM: COMPARING AST NODES

- With equality? (=)
  - No. Being on = AST nodes but in different methods is still a control-flow divergence
- With identity! (==)

# CDM ALGORITHM: COMPARING AST NODES

- With equality? (=)
  - No. Being on = AST nodes but in different methods is still a control-flow divergence
- With identity! (==)
- ...except the nodes are not in the same image



# CDM ALGORITHM: COMPARING AST NODES

- With equality? (=)
  - No. Being on = AST nodes but in different methods is still a control-flow divergence
- With identity! (==)
- ...except the nodes are not in the same image
- Our AST nodes comparison operator has 4 criteria:
  - Method selector
  - Class name
  - Source code
  - Node type

# CDM ALGORITHM: COMPARING AST NODES

- With equality? (=)
  - No. Being on = AST nodes but in different methods is still a control-flow divergence
- With identity! (==)
- ...except the nodes are not in the same image
- Our AST nodes comparison operator has 4 criteria:
  - Method selector
  - Class name
  - Source code
  - Node type

```
x: xInteger y: yInteger
"Answer an instance of me with coordinates xInteger and yInteger."

^ self basicNew setX: xInteger setY: yInteger
```

# CDM ALGORITHM: COMPARING AST NODES

- With equality? (=)
  - No. Being on = AST nodes but in different methods is still a control-flow divergence
- With identity! (==)
- ...except the nodes are not in the same image
- Our AST nodes comparison operator has 4 criteria:
  - Method selector `"x:y:"`
  - Class name `"Point"`
  - Source code `"basicNew"`
  - Node type `"Message Node"`

```
x: xInteger y: yInteger
"Answer an instance of me with coordinates xInteger and yInteger."
^ self basicNew setX: xInteger setY: yInteger
```

# WRAP-UP

# WRAP-UP

- The Echo-Debugger debugs two parallel executions of the same statement in different program versions

# WRAP-UP

- The Echo-Debugger debugs two parallel executions of the same statement in different program versions
- The CDM algorithm runs both executions to find when they diverge/converge in control-flow

# WRAP-UP

- The Echo-Debugger debugs two parallel executions of the same statement in different program versions
- The CDM algorithm runs both executions to find when they diverge/converge in control-flow
- If the executions are deterministic, the Echo-Debugger can restart and step the executions to any divergence/convergence

# WRAP-UP

- The Echo-Debugger debugs two parallel executions of the same statement in different program versions
- The CDM algorithm runs both executions to find when they diverge/converge in control-flow
- If the executions are deterministic, the Echo-Debugger can restart and step the executions to any divergence/convergence
- ... Back-in-time debugger as back-end



# WRAP-UP

- The Echo-Debugger debugs two parallel executions of the same statement in different program versions
- The CDM algorithm runs both executions to find when they diverge/converge in control-flow
- If the executions are deterministic, the Echo-Debugger can restart and step the executions to any divergence/convergence
- ... Back-in-time debugger as back-end
- ... State differences

# WRAP-UP

- The Echo-Debugger debugs two parallel executions of the same statement in different program versions
- The CDM algorithm runs both executions to find when they diverge/converge in control-flow
- If the executions are deterministic, the Echo-Debugger can restart and step the executions to any divergence/convergence
- ... Back-in-time debugger as back-end
- ... State differences
- ... Automated setup tool

# WRAP-UP

- The Echo-Debugger debugs two parallel executions of the same statement in different program versions
- The CDM algorithm runs both executions to find when they diverge/converge in control-flow
- If the executions are deterministic, the Echo-Debugger can restart and step the executions to any divergence/convergence
- ... Back-in-time debugger as back-end
- ... State differences
- ... Automated setup tool

*Thank You!*

