Extending Existing Inference Tools to Mine Dynamic APIs

Ziyad Alsaeed and Michal Young

University of Oregon

June 2, 2018

2nd International Workshop on API Usage and Evolution (WAPI)

UNIVERSITY OF
OREGON

Motivation

- API understanding is a key to solve many software usage issues.

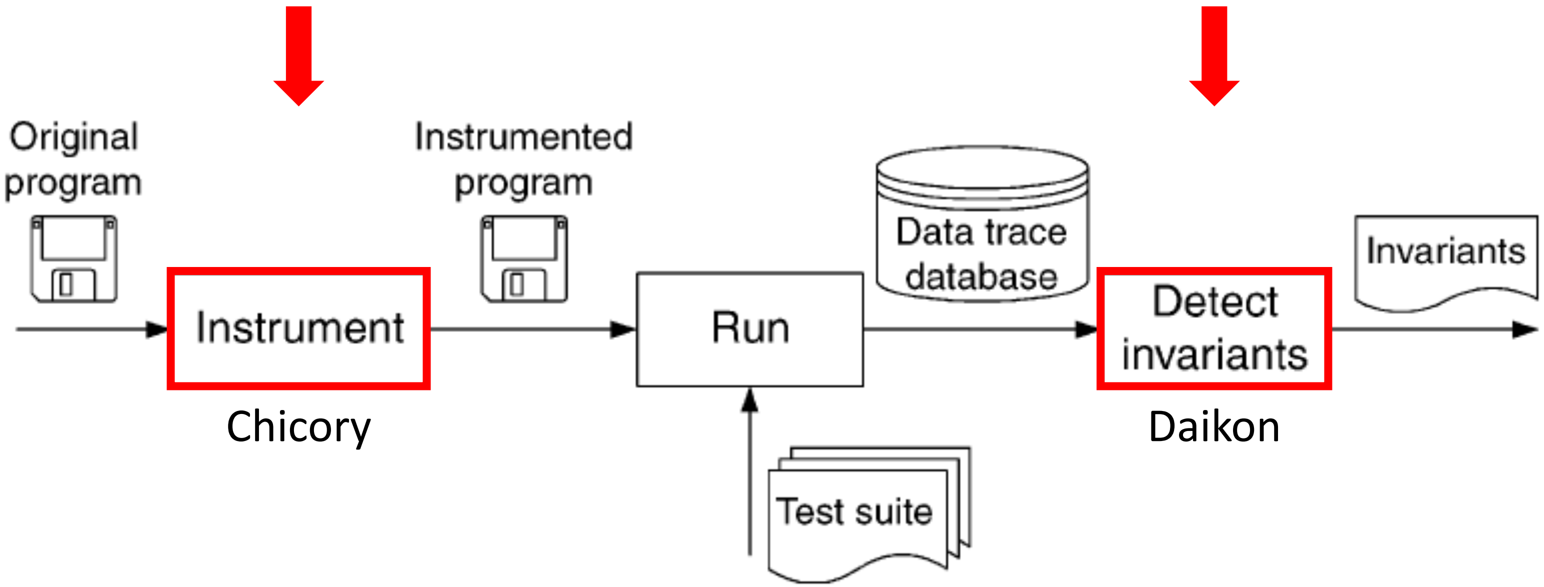
- Software documentations are rarely up-to-date and constraints associated with objects are usually in the brain of the creator.

- How do we capture the software dynamic nature.

Existing Dynamic Inference Tools (Daikon)

The Majority of our contribution

extremely tied to established

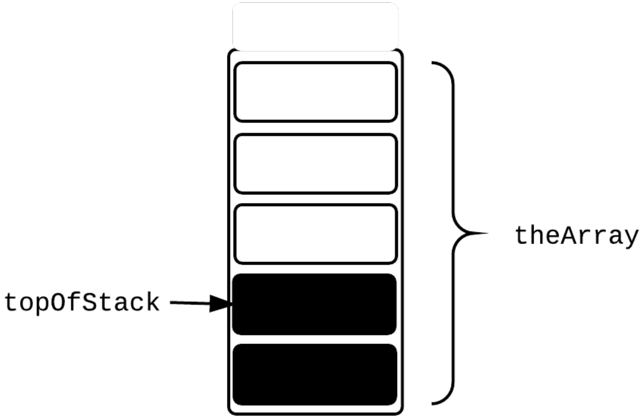


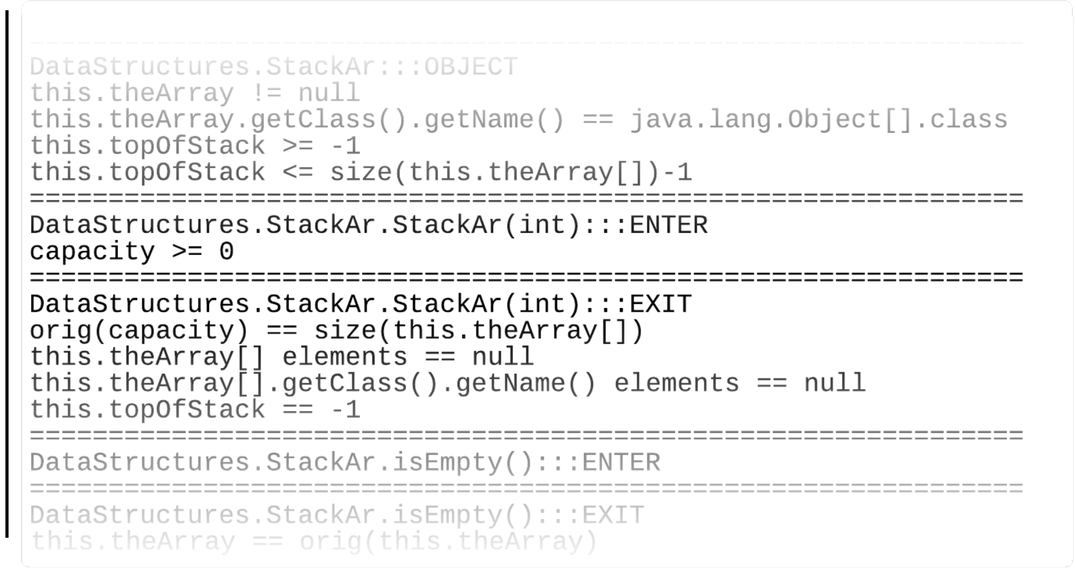
[1] Ernst et. al. ICSE’99

Mining the Well-Known StackAr by Daikon

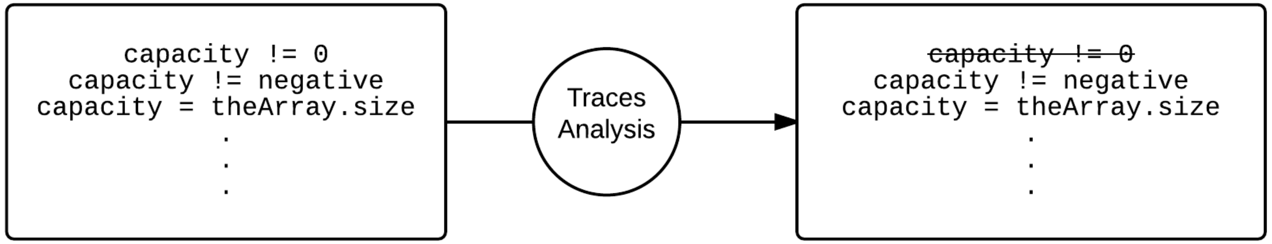
```
Constructore:
StackAr (int capacity)

Public methods:
void push( x )
void pop( )
Object top( )
Object topAndPop( )
boolean isEmpty( )
boolean isFull( )
void makeEmpty( )
```





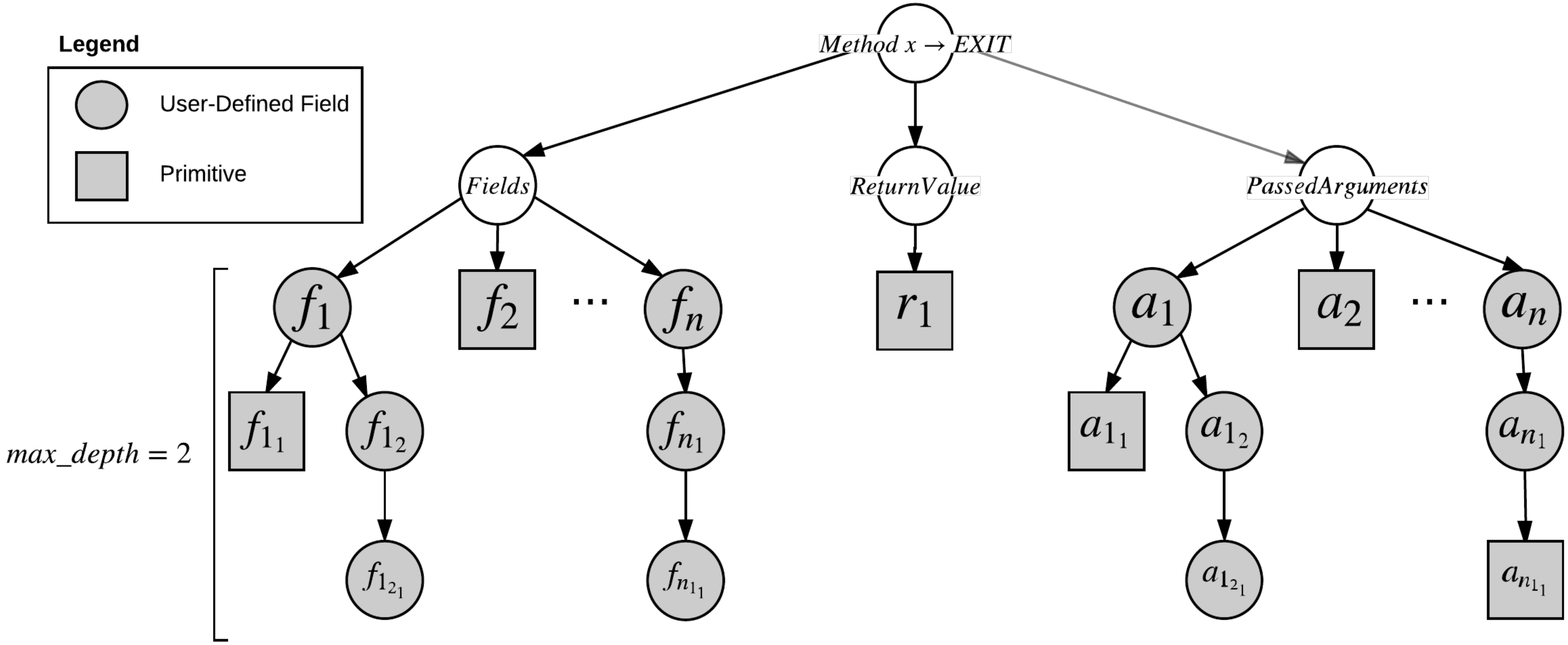
```
DataStructures.StackAr:::OBJECT
this.theArray != null
this.theArray.getClass().getName() == java.lang.Object[].class
this.topOfStack >= -1
this.topOfStack <= size(this.theArray[])-1
===============================================================
DataStructures.StackAr.StackAr(int):::ENTER
capacity >= 0
===============================================================
DataStructures.StackAr.StackAr(int):::EXIT
orig(capacity) == size(this.theArray[])
this.theArray[] elements == null
this.theArray[].getClass().getName() elements == null
this.topOfStack == -1
===============================================================
DataStructures.StackAr.isEmpty():::ENTER
===============================================================
DataStructures.StackAr.isEmpty():::EXIT
this.theArray == orig(this.theArray)
```



capacity != 0
capacity != negative
capacity = theArray.size
.
.
.

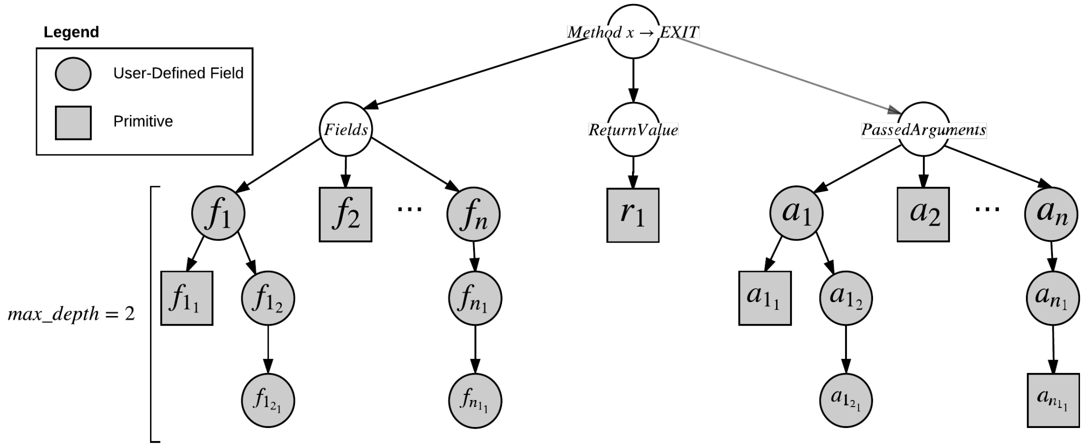
~~capacity != 0~~
capacity != negative
capacity = theArray.size
.
.
.

How Chicory Works



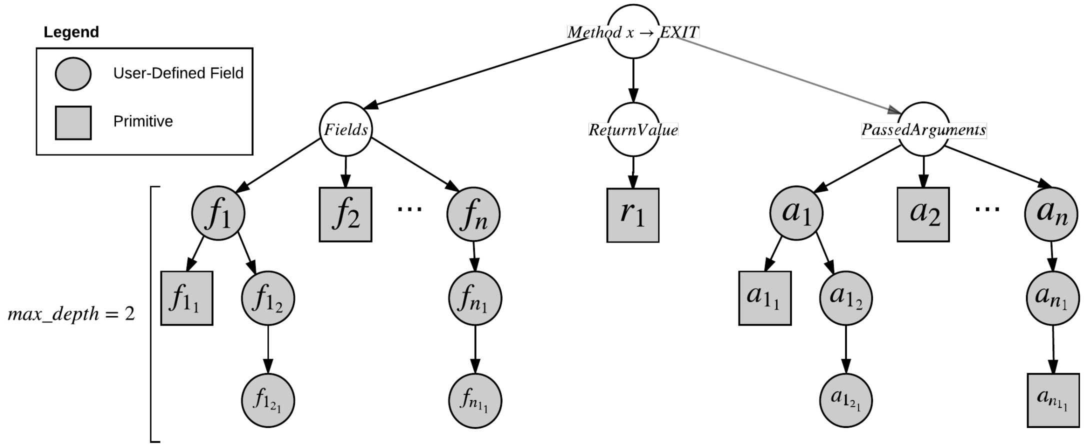
How Chicory Works

x:::EXIT	f_{1_1}	$f_{1_{2_1}}$	…	$a_{n_{1_1}}$
$invocation_1$	1	null		0.0



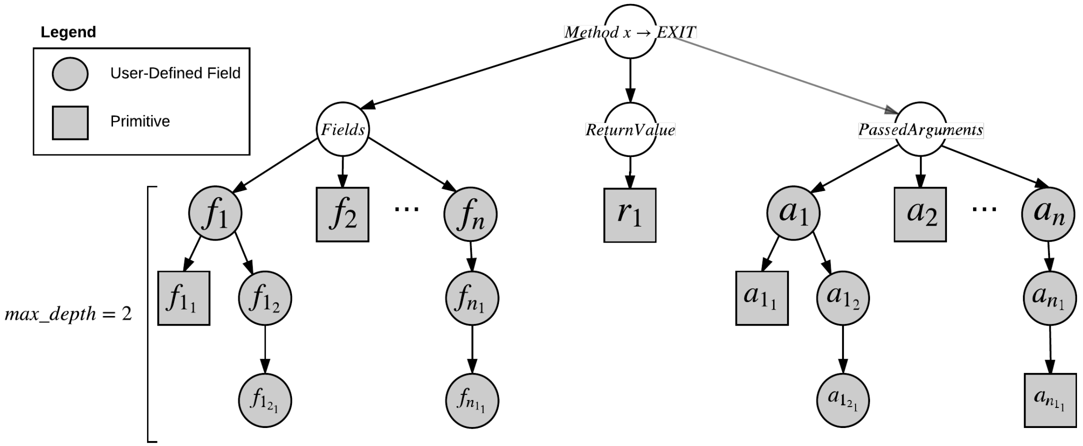
How Chicory Works

x:::EXIT	f_{1_1}	$f_{1_{2_1}}$		$a_{n_{1_1}}$
$invocation_1$	1	null	…	0.0
$invocation_2$	-1	null		0.0



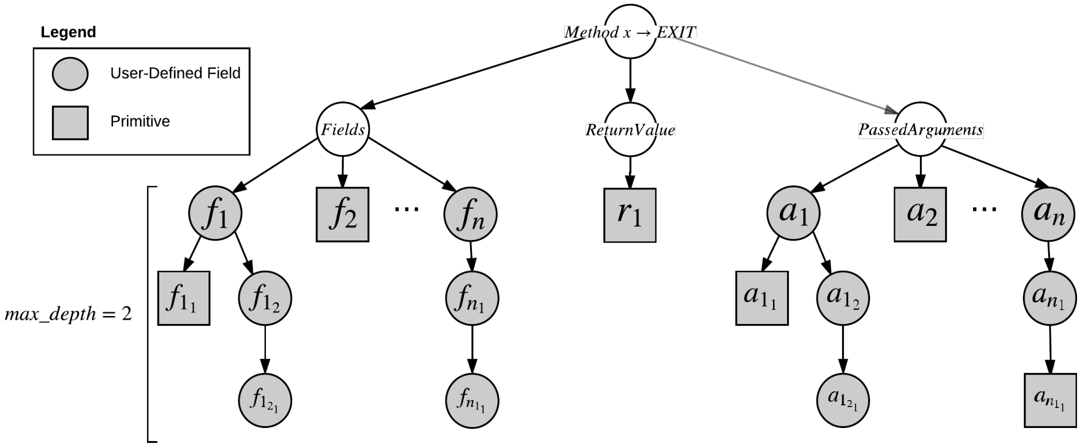
How Chicory Works

x:::EXIT	f_{1_1}	$f_{1_{2_1}}$		$a_{n_{1_1}}$
$invocation_1$	1	null	…	0.0
$invocation_2$	-1	null		0.0
$invocation_3$	-2	null		0.0

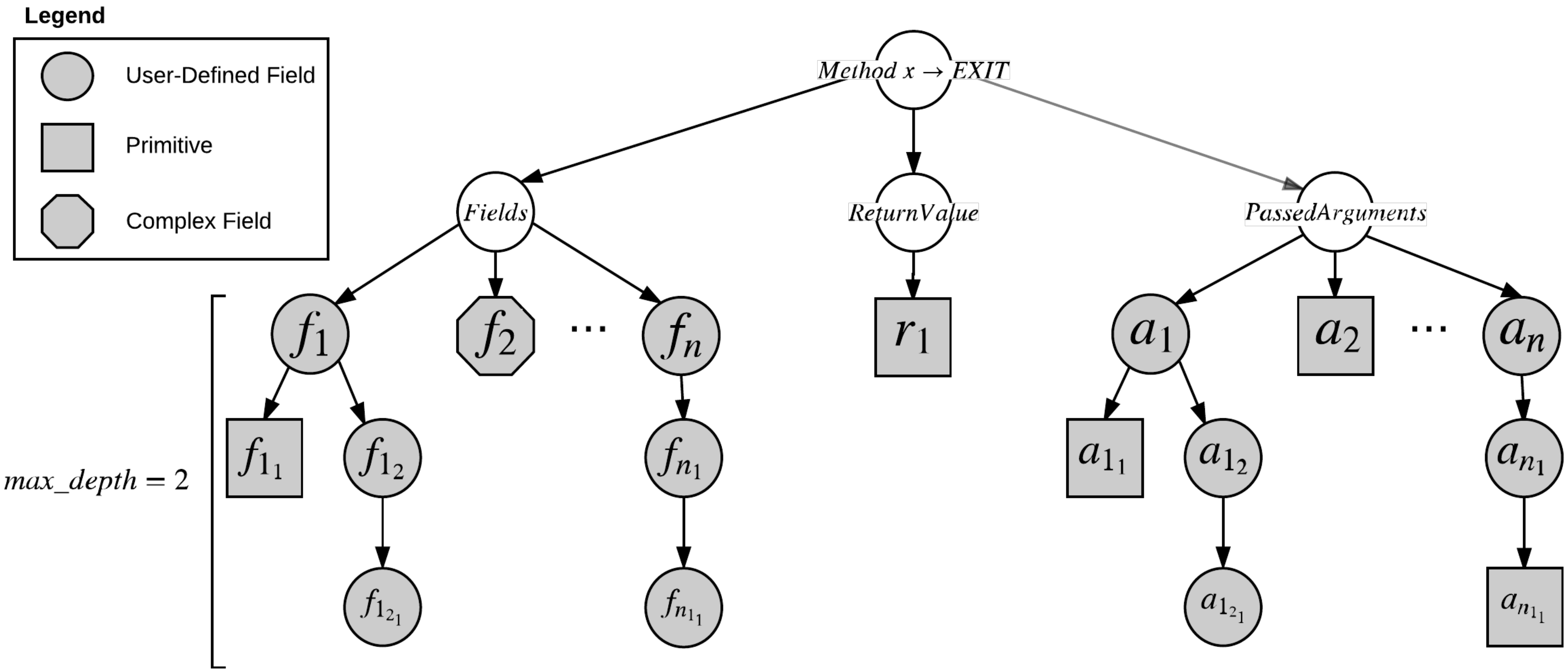


How Chicory Works

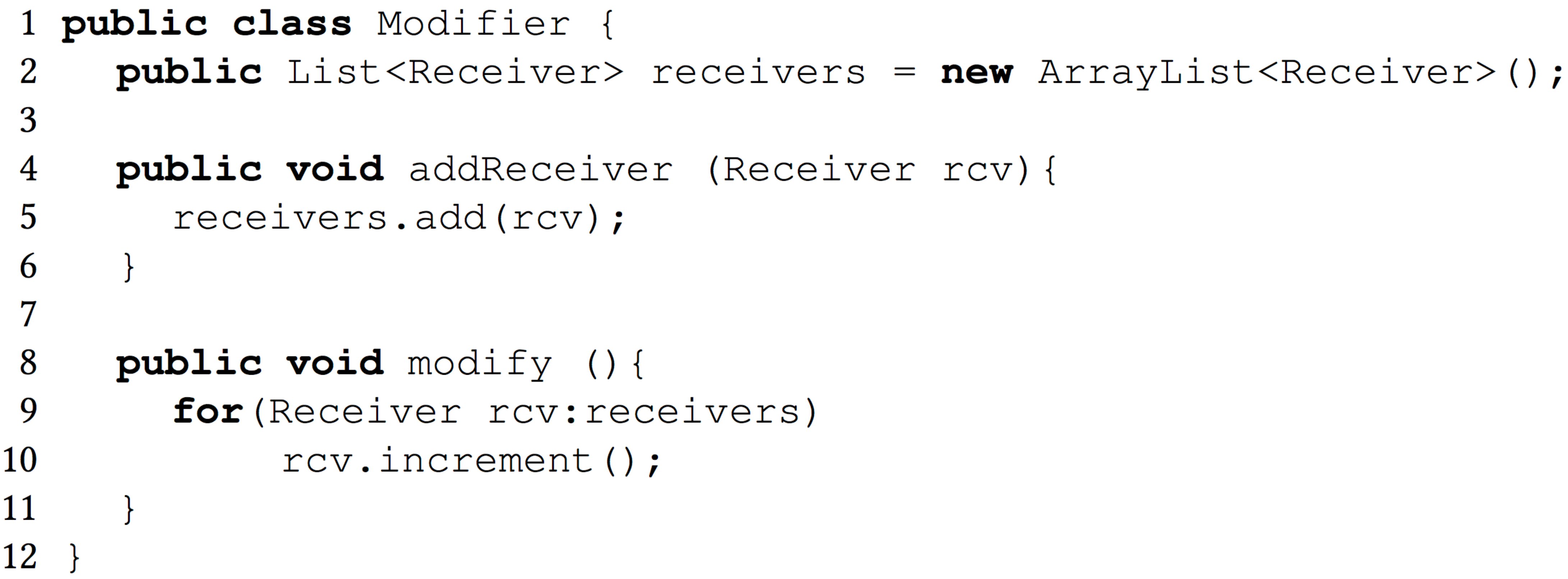
x:::EXIT	f_{1_1}	$f_{1_{2_1}}$		$a_{n_{1_1}}$
$invocation_1$	1	null	…	0.0
$invocation_2$	-1	null		0.0
$invocation_3$	-2	null		0.0
	⋮			⋮
$invocation_n$	0	null		0.0



What if?


Legend
User-Defined Field
Primitive
Complex Field
$Method\ x \rightarrow EXIT$
$Fields$
$ReturnValue$
$PassedArguments$
f_1
f_2
$\cdots$
f_n
r_1
a_1
a_2
$\cdots$
a_n
$max\_depth = 2$
f_{1_1}
f_{1_2}
f_{n_1}
a_{1_1}
a_{1_2}
a_{n_1}
$f_{1_{2_1}}$
$f_{n_{1_1}}$
$a_{1_{2_1}}$
$a_{n_{1_1}}$

Simplified Real-World Example

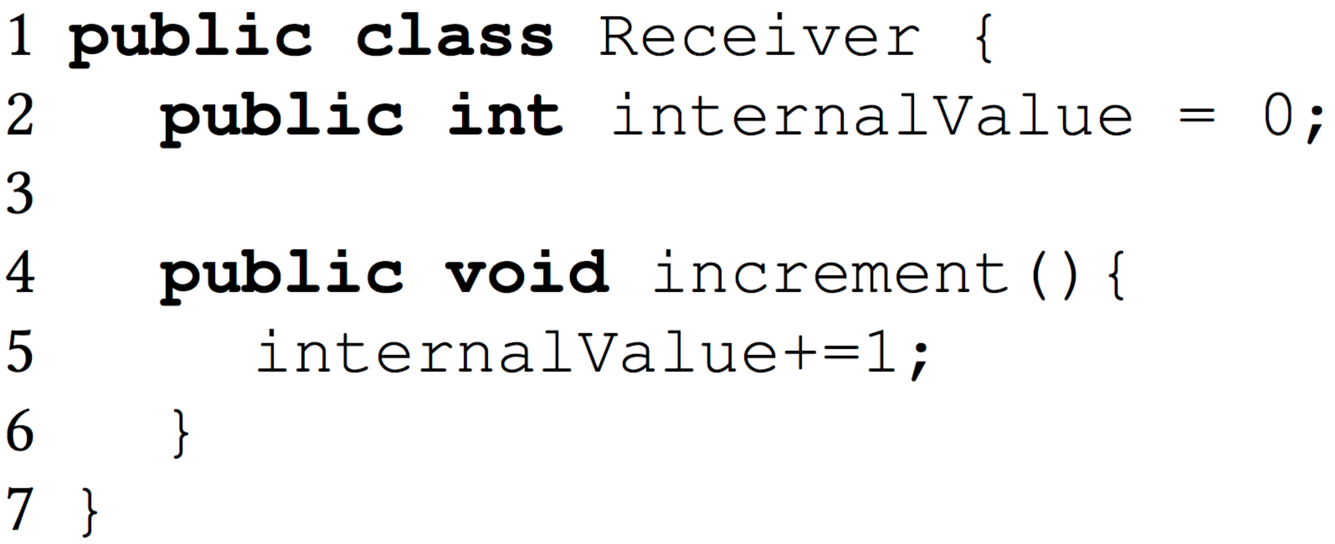


```
public class Modifier {
  public List<Receiver> receivers = new ArrayList<Receiver>();

  public void addReceiver (Receiver rcv){
    receivers.add(rcv);
  }

  public void modify (){
    for(Receiver rcv:receivers)
        rcv.increment();
  }
}
```

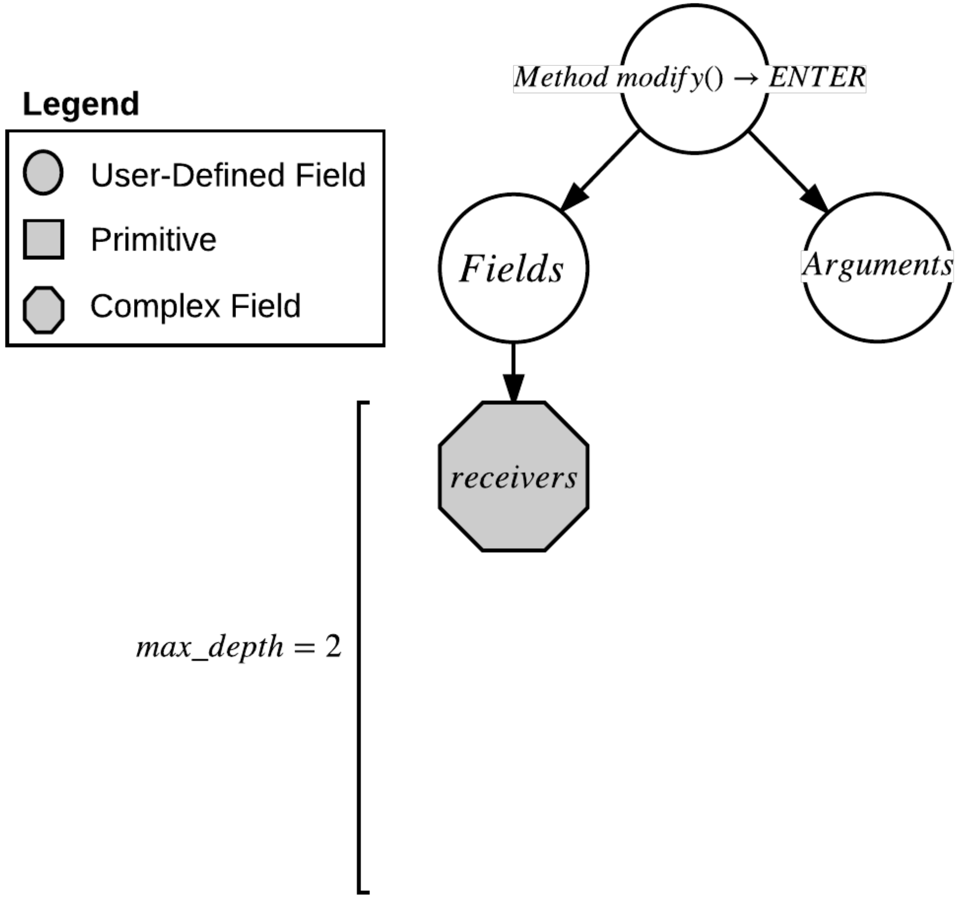
Simplified Real-World Example



```
public class Receiver {
   public int internalValue = 0;

   public void increment(){
      internalValue+=1;
   }
}
```

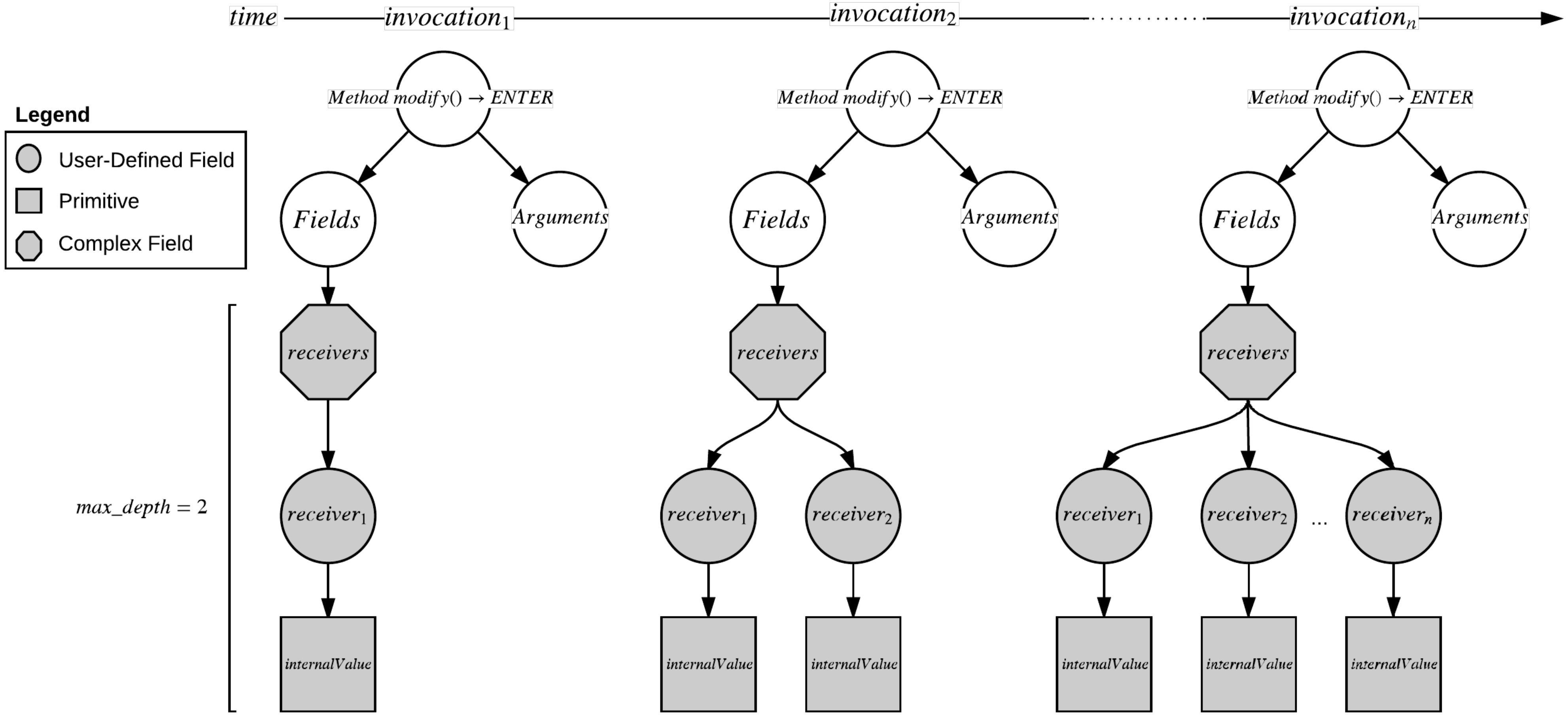
Variables Structure of *modify* based on Chicory:



Possible Scenarios When Observing a Dynamic Data Structure

- Element introduction (added to DDS):
  - An element that was not present in the DDS until a later point in the program execution and never removed thereafter.
- Element removal (removed from DDS):
  - An element that exists at some point of the program execution but removed before the last observation of the program point.
- And more …

Possible Structure Evolution

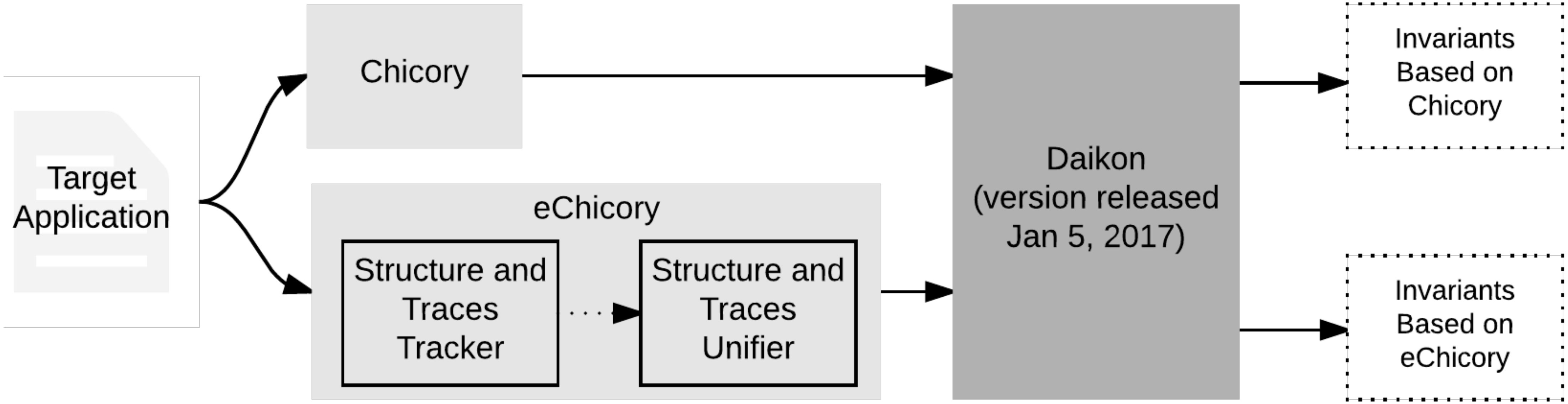


Relation to Daikon

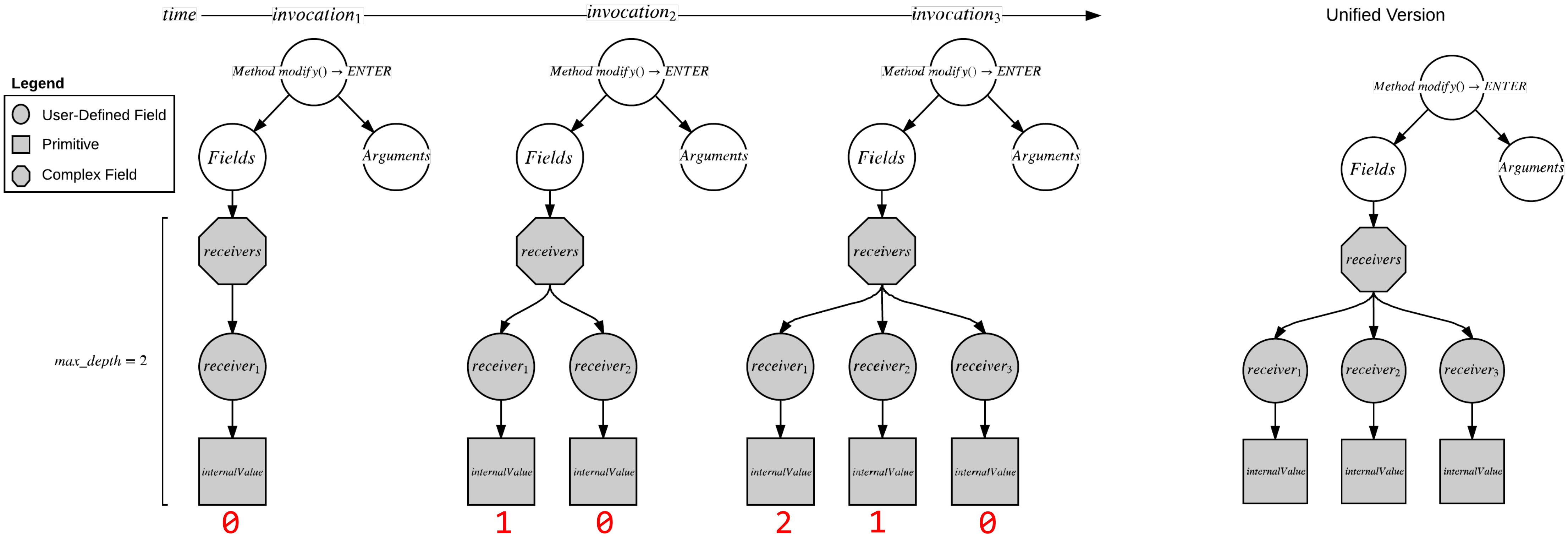
- Daikon expects a very well defined structure of a program point (method entrance or exit).

- Only one variable structure tree per program point.

eChicory Structure

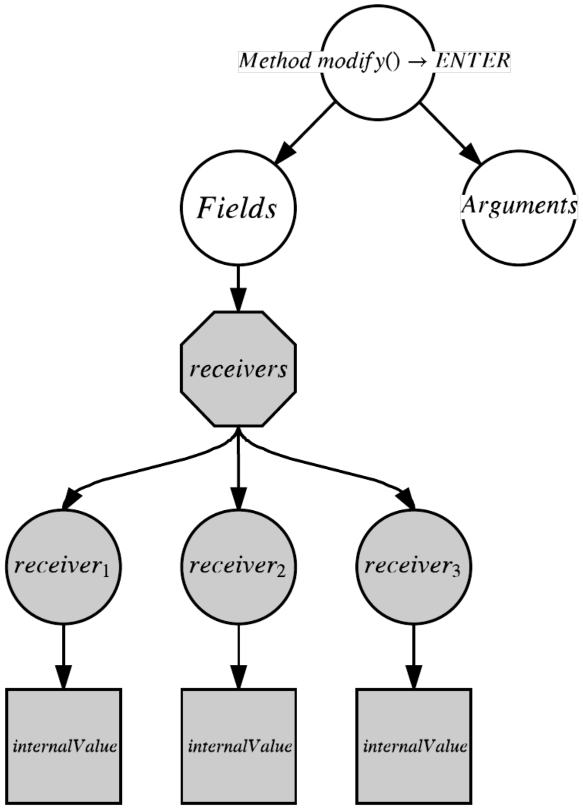

Target Application
Chicory
eChicory
Structure and Traces Tracker
Structure and Traces Unifier
Daikon (version released Jan 5, 2017)
Invariants Based on Chicory
Invariants Based on eChicory

Unification Phase in eChicory (element introduction)



Unification Phase in eChicory (element introduction)

Unified Version



modify:::ENTER	$r_1.internalValue$	$r_2.internalValue$	$r_3.internalValue$
$invocation_1$	0	nonsensical	nonsensical
$invocation_2$	1	0	nonsensical
$invocation_3$	2	1	0

Unification Phase in eChicory (element removal)

- Given Daikon design, removing a variable can only be achieved by manipulating its trace.

- Changing a trace to nonsensical after it was initialized is prohibited.

- Given it arbitrary value (e.g. stretch its last known value to the rest of instances where it was removed thereafter) would interfere with the invariants integrity.

Evaluation - Artifact Selection Criteria

- Source available in GitHub.
- Applications of size between 2K and 10K LOC.
- Has indications that one of the selected patterns is used (given the repository issues tracker, pull requests, and wiki).
- Has high test coverage (if reported).
- Popular or well maintained applications based on the star rate or managing organization.

Evaluation - Selected Artifacts

Application	Description	Selected Classes	# of Methods	Represented Design Pattern
Mockito	Mocking framework for unit tests in Java	`InvocationNotierHandler`	7	Observer Pattern
Apache Struts	Framework for creating Java web applications	`DefaultActionInvocation`	29	MVC
		`DefaultUnknownHandlerManager`	3	
		`CongurationManager`	16	
		`VelocityManager`	18	
		`SimpleTextNode`	17	
		`SimpleAdapterDocument`	43	
JabRef	BibTeX Management application	`EntryEditor`	22	MVC
		`CleanupActionsListModel`	8	
		`UndoableModifySubtree`	4	
		`ImportInspectionDialog`	21	
Zeppelin	A web based interactive data analytic tool	`Folder`	23	Observer Pattern
		`Notebook`	45	
		`NotebookRepoSync`	31	

Evaluation Criteria (Precision and Recall)

- Pros:
    - Has being the base for evaluating specification miners.
    - Shows a good insight about the accuracy of the specification miner.
- Cons:
    - A ground truth about the test subject must be defined ahead (this is done by humans, thus can’t be scaled).
    - Human defined ground truth, can differed based on the developers view or opinion.

Evaluation Criteria (Purity Analysis)

- The notion of pure (side-effect free) methods is well-defined in the static analysis domain.

- Can be generated automatically and scale with large applications.

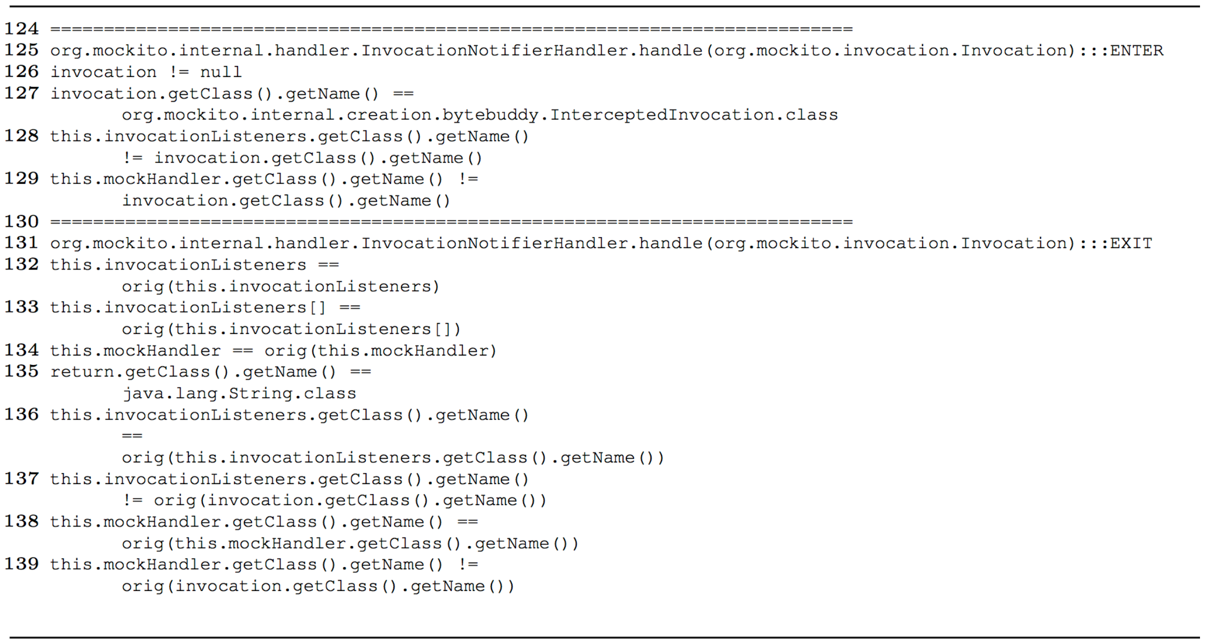
- Not the goal of dynamic analysis, but can be used to check consistency.

Mockito - InvocationNotifierHandler

Method	jPure	eChicory	Chicory
`InvocationNotifierHandler(InternalMockHandler<T>, MockCreationSettings<T>)`	!pure	!pure	!pure
`handle(Invocation)`	!pure	!pure	pure
`notifyMethodCall(Invocation, Object)`	!pure	!pure	pure
`notifyMethodCallException(Invocation, Throwable)`	!pure	!pure	pure
`getMockSettings()`	!pure	pure	pure
`getInvocationContainer()`	!pure	pure	pure
`setAnswersForStubbing(List<Answer<?>>)`	!pure	pure	pure
Total number of reported methods with no indication of effect	0	3	6

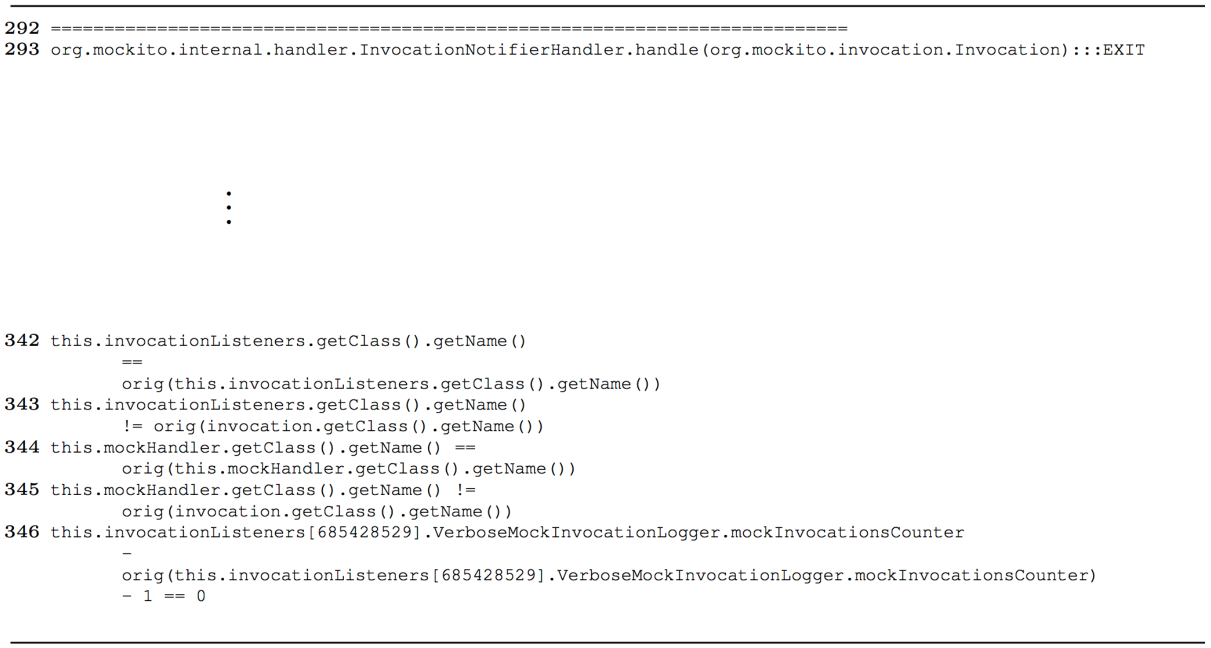
Mockito - InvocationNotifierHandler

Chicory



```
124 ===========================================================================
125 org.mockito.internal.handler.InvocationNotifierHandler.handle(org.mockito.invocation.Invocation):::ENTER
126 invocation != null
127 invocation.getClass().getName() ==
        org.mockito.internal.creation.bytebuddy.InterceptedInvocation.class
128 this.invocationListeners.getClass().getName()
        != invocation.getClass().getName()
129 this.mockHandler.getClass().getName() !=
        invocation.getClass().getName()
130 ===========================================================================
131 org.mockito.internal.handler.InvocationNotifierHandler.handle(org.mockito.invocation.Invocation):::EXIT
132 this.invocationListeners ==
        orig(this.invocationListeners)
133 this.invocationListeners[] ==
        orig(this.invocationListeners[])
134 this.mockHandler == orig(this.mockHandler)
135 return.getClass().getName() ==
        java.lang.String.class
136 this.invocationListeners.getClass().getName()
        ==
        orig(this.invocationListeners.getClass().getName())
137 this.invocationListeners.getClass().getName()
        != orig(invocation.getClass().getName())
138 this.mockHandler.getClass().getName() ==
        orig(this.mockHandler.getClass().getName())
139 this.mockHandler.getClass().getName() !=
        orig(invocation.getClass().getName())
```

eChicory



```
292 ===========================================================================
293 org.mockito.internal.handler.InvocationNotifierHandler.handle(org.mockito.invocation.Invocation):::EXIT

                    ⋮

342 this.invocationListeners.getClass().getName()
        ==
        orig(this.invocationListeners.getClass().getName())
343 this.invocationListeners.getClass().getName()
        != orig(invocation.getClass().getName())
344 this.mockHandler.getClass().getName() ==
        orig(this.mockHandler.getClass().getName())
345 this.mockHandler.getClass().getName() !=
        orig(invocation.getClass().getName())
346 this.invocationListeners[685428529].VerboseMockInvocationLogger.mockInvocationsCounter
        -
        orig(this.invocationListeners[685428529].VerboseMockInvocationLogger.mockInvocationsCounter)
        - 1 == 0
```

Impediments to Observe Other Selected Test Subjects

- Inadequate inputs (unit tests).
- Naive implementation of concrete classes.
- Absence of elements in DDS.
- Tests Failures (confirmed by repository maintainers).

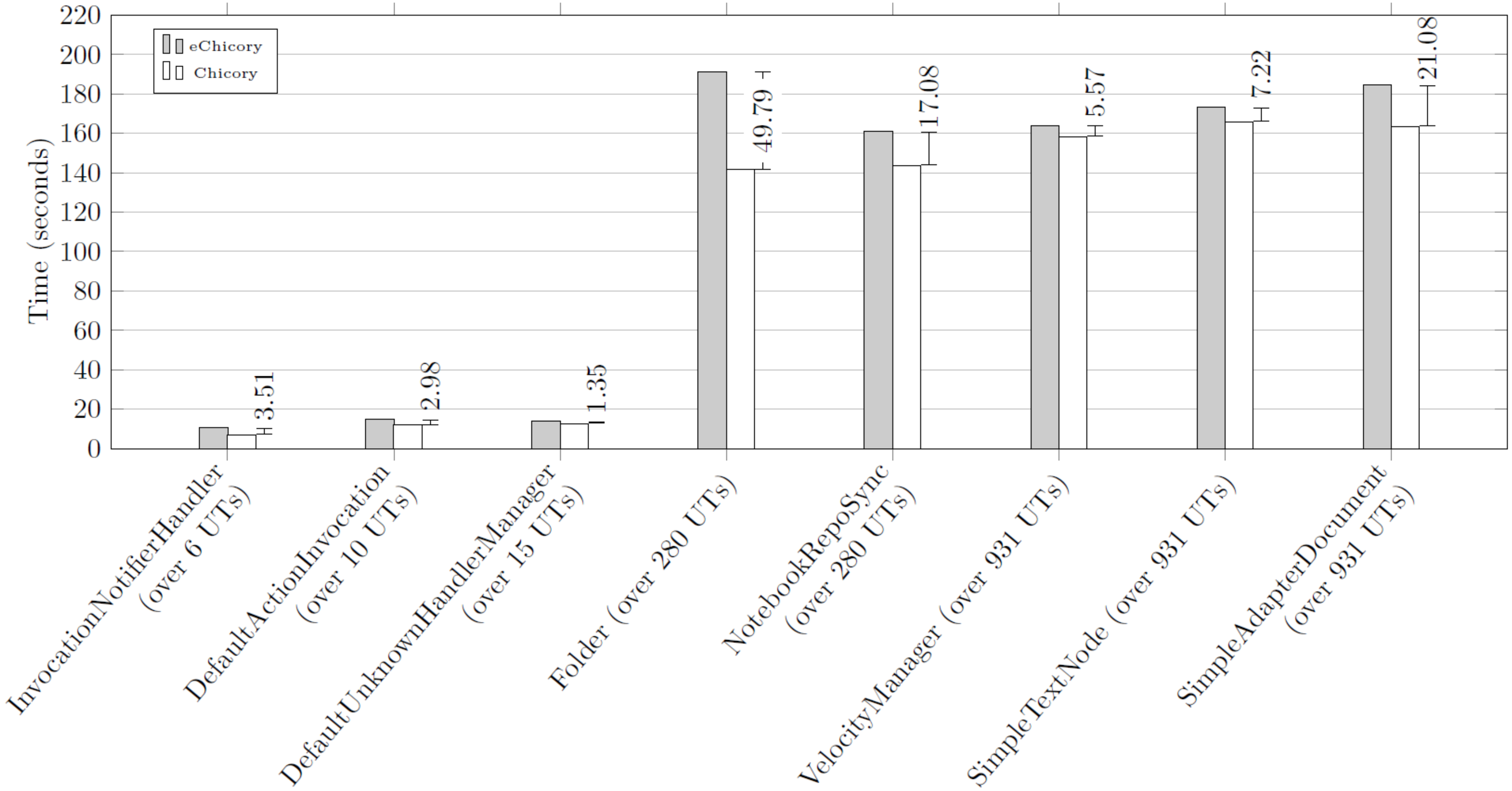
Wrote Comprehensive Tests to Prove our Approach Potential

- To prove the provided tests are the cause of result limitations, we wrote unite tests for one of the classes from Apache Struts.

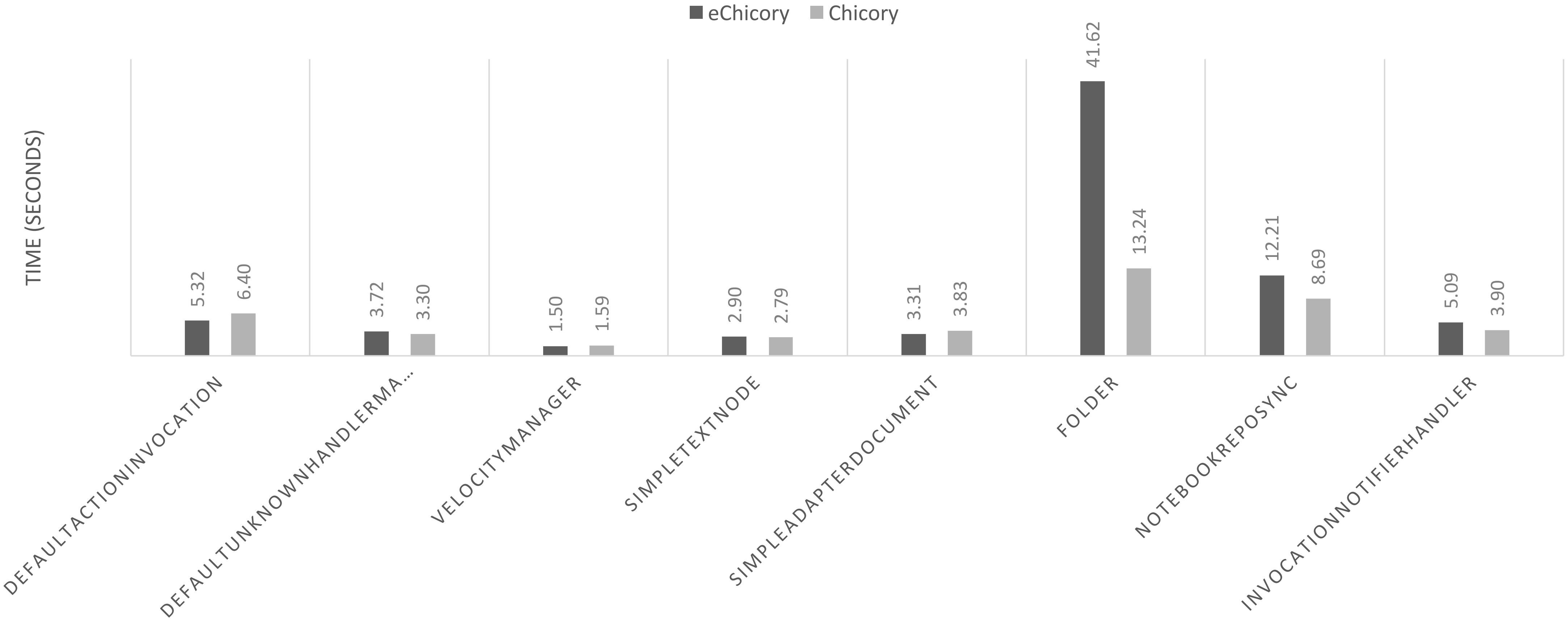
- Class `DefaultUnknownHandlerManager`
  - Method-1: `handleUnknownAction`
  - Method-2: `handleUnknownMethod`

- Written tests are reviewed and merged into Apache Struts’ main repository.

Performance - Traces Collection Phase



Performance - Inference Phase



Conclusion

- We highlighted the non-fully dynamic tracing issue and clearly identified that limitations of current instrumentation methodologies.
- We implemented a prototype as a proof of concept to fully dynamically observe complex systems.
- We showed by real world example that existing instrumentation techniques are blind to common design patterns are.
- DDSs are only one source of program structural change. There are different programming practices that leads to very dynamic structure needs to be addressed.

Thank you.

References:

[1] Ernst, M. D.; Cockrell, J.; Griswold, W. G. & Notkin, D. Dynamically Discovering Likely Program Invariants to Support Program Evolution Proceedings of the 21st International Conference on Software Engineering, ACM, 1999, 213-224