Software QA w/ Generative AI (CS598): Automated Debugging

Spring 2024

Lingming Zhang

This class

- **Fault Localization:** Visualization of test information to assist fault localization
  - ICSE 2002
- **Program Repair**: Practical Program Repair via Bytecode Mutation
  - ISSTA 2019

What is fault localization?

```
int mid(int x, int y, int z) {
 1:     int m;
 2:     m = z;
 3:     if (y < z) {
 4:             if (x < y)
 5:                     m = y;
 6:             else if (x < z)
 7:                     m = y;
 8:     } else {
 9:             if (x > y)
10:                     m = y;
11:             else if (x > z)
12:                     m = x; }
13:     return m;
}
```

What is fault localization?

```
int mid(int x, int y, int z) {
 1:     int m;
 2:     m = z;
 3:     if (y < z) {
 4:             if (x < y)
 5:                     m = y;
 6:             else if (x < z)
 7:                     m = y; //m=x;
 8:     } else {
 9:             if (x > y)
10:                     m = y;
11:             else if (x > z)
12:                     m = x; }
13:     return m;
}
```

- **Fault Localization:** the process of automatically narrowing or guiding the search for faulty code to help a developer debug more quickly

A representative technique: Tarantula

Statements	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	Susp.
`int m;`							
`m = z;`							
`if(y< z){`							
`if(x< y){`							
`m = y;`							
`else if(x<z){`							
`m = y;` 							
`} else {`							
`if (x>y)`							
`m = y;`							
`else if (x>z)`							
`m = x; }`							
`return m;`							
	Pass	Pass	Pass	Pass	Pass	Fail	

- Uses dynamic Information:
  - Statements executed by each test
  - The pass/fail outcome of each test
- Performs statistical analysis:
  - Statements mainly executed by failed tests are more suspicious

A representative technique: Tarantula

Statements	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	Susp.
`int m;`							
`m = z;`							
`if(y< z){`							
`if(x< y){`							
`m = y;`							
`else if(x<z){`							
`m = y;` 							
	Pass	Pass	Pass	Pass	Pass	Fail	

- Tarantula:

$$Suspiciousness(s) = \frac{\frac{fail(s)}{totalfail}}{\frac{fail(s)}{totalfail} + \frac{pass(s)}{totalpass}}$$

A representative technique: Tarantula

Statements	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	Susp.
`int m;`							0.5
`m = z;`							0.5
`if(y< z){`							0.5
`if(x< y){`							0.63
`m = y;`							0
`else if(x<z){`							0.71
`m = y;` 							0.83
							0
							0
							0
							0
							0
							0.5
	Pass	Pass	Pass	Pass	Pass	Fail	

$$Susp(s1) = \frac{\frac{1}{1}}{\frac{1}{1} + \frac{5}{5}}$$

$$Susp(s7) = \frac{\frac{1}{1}}{\frac{1}{1} + \frac{1}{5}}$$

- Tarantula:

$$Suspiciousness(s) = \frac{\frac{fail(s)}{totalfail}}{\frac{fail(s)}{totalfail} + \frac{pass(s)}{totalpass}}$$

More formulae for fault localization

- Tarantula
  - $Suspiciousness(s) = \frac{\frac{fail(s)}{totalfail}}{\frac{fail(s)}{totalfail} + \frac{pass(s)}{totalpas}}$
- SBI
  - $Suspiciousness(s) = \frac{fail(s)}{fail(s)+pass(s)}$
- Jaccard
  - $Suspiciousness(s) = \frac{fail(s)}{totalfail+pass(s)}$
- Ochiai
  - $Suspiciousness(s) = \frac{fail(s)}{\sqrt{totalfail*(pass(s)+fail(s))}}$

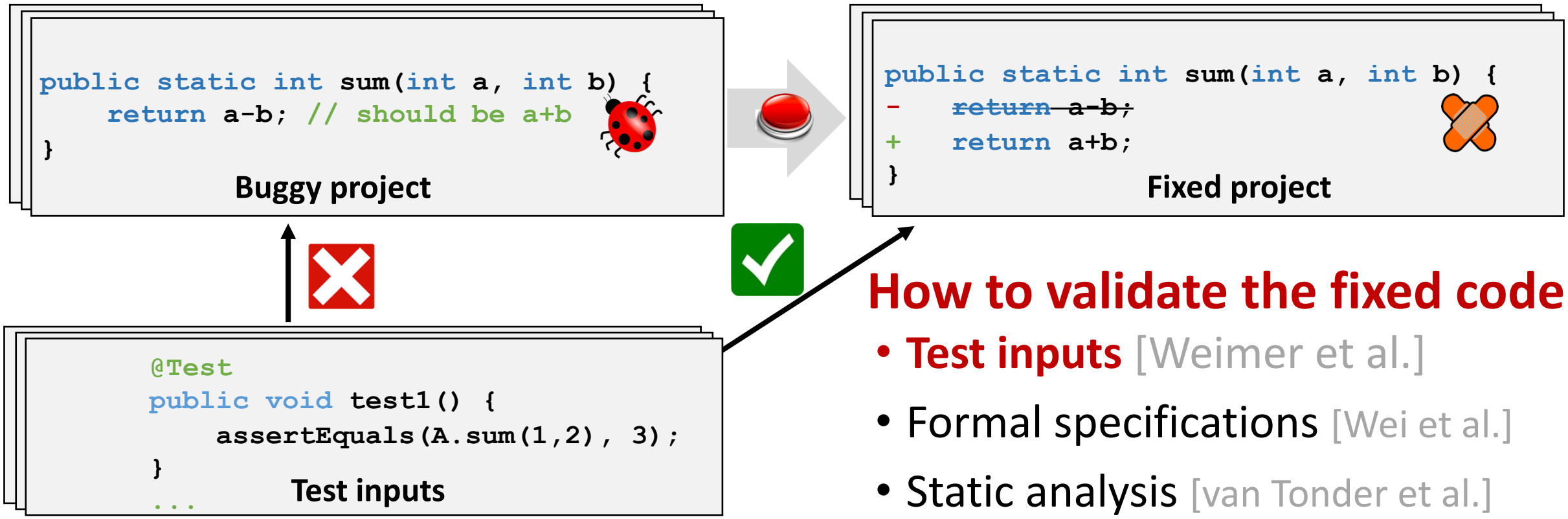
Various ML techniques have also been proposed for fault localization:
DeepFL: Integrating Multiple Fault Diagnosis Dimensions for Deep Fault Localization (ISSTA'19)

This class

- **Fault Localization:** Visualization of test information to assist fault localization
  - ICSE 2002
- **Program Repair**: Practical Program Repair via Bytecode Mutation
  - ISSTA 2019

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Automated Program Repair



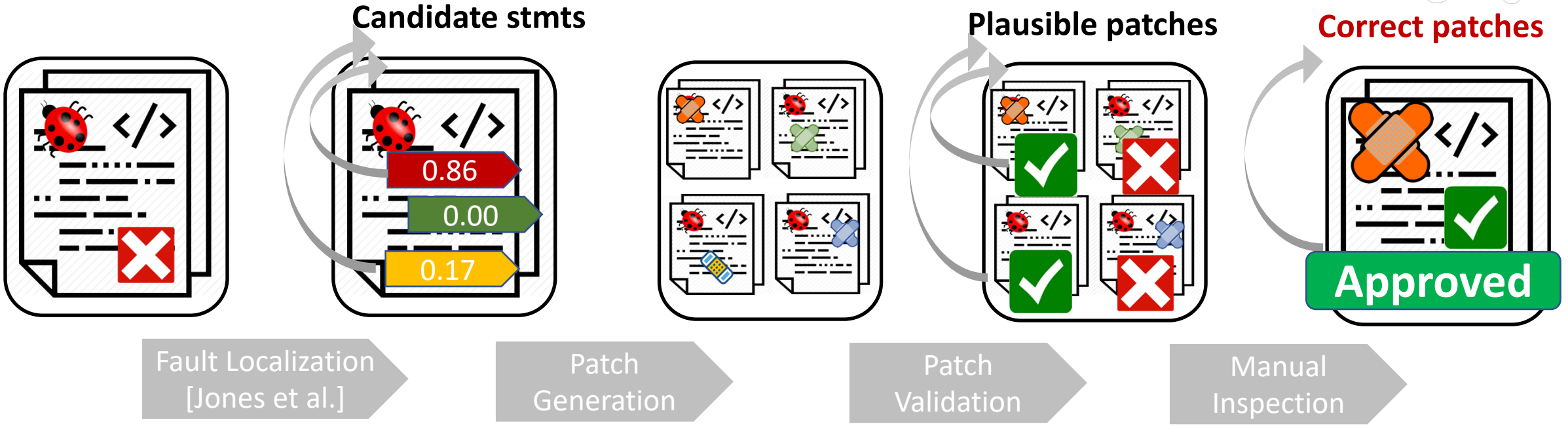
How to validate the fixed code?

- **Test inputs** [Weimer et al.]
- Formal specifications [Wei et al.]
- Static analysis [van Tonder et al.]

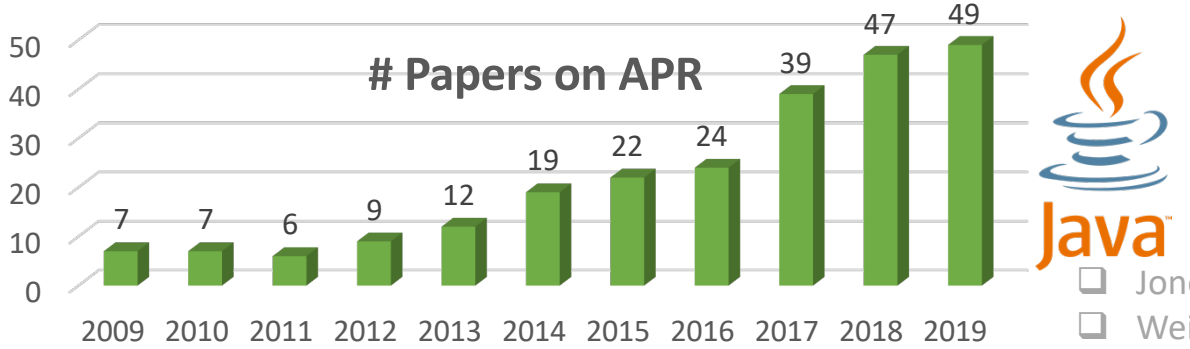
- Weimer et al., "Automatically finding patches using genetic programming". ICSE'09
- Wei et al., "Automated fixing of programs with contracts", ISSTA'10
- van Tonder et al., "Static automated program repair for heap properties", ICSE'18

Test-Driven Automated

12.27.0: “We apologize to anyone who had problems with the app. We trained a neural net to eliminate all bugs in the app and *it deleted everything*. We had to roll everything back. To be fair, *we were 100% bug-free… briefly*.” yelp



- **In academia**, APR [Weimer et al.] has been extensively studied for over a decade



- **In industry**, companies are also eager to use APR …

FUJITSU Alibaba Group

- Jones et al., “Visualization of Test Information to Assist Fault Localization”. ICSE’02
- Weimer et al., “Automatically finding patches using genetic programming”. ICSE’09

Test-Driven Automated Program Repair (APR)

Correct patches

facebook Engineering

Open Source

POSTED ON SEP 13, 2

Finding ar
Sapienz

Workfl

Sapienz
Auto Triage

FUJITSU

Services | Pr

Home > About Fujit

Press Releases

- 2020
- 2019
- 2018
- 2017
- 2016
- 2015
- 2014
- 2013
- 2012
- 2011

United States | Change

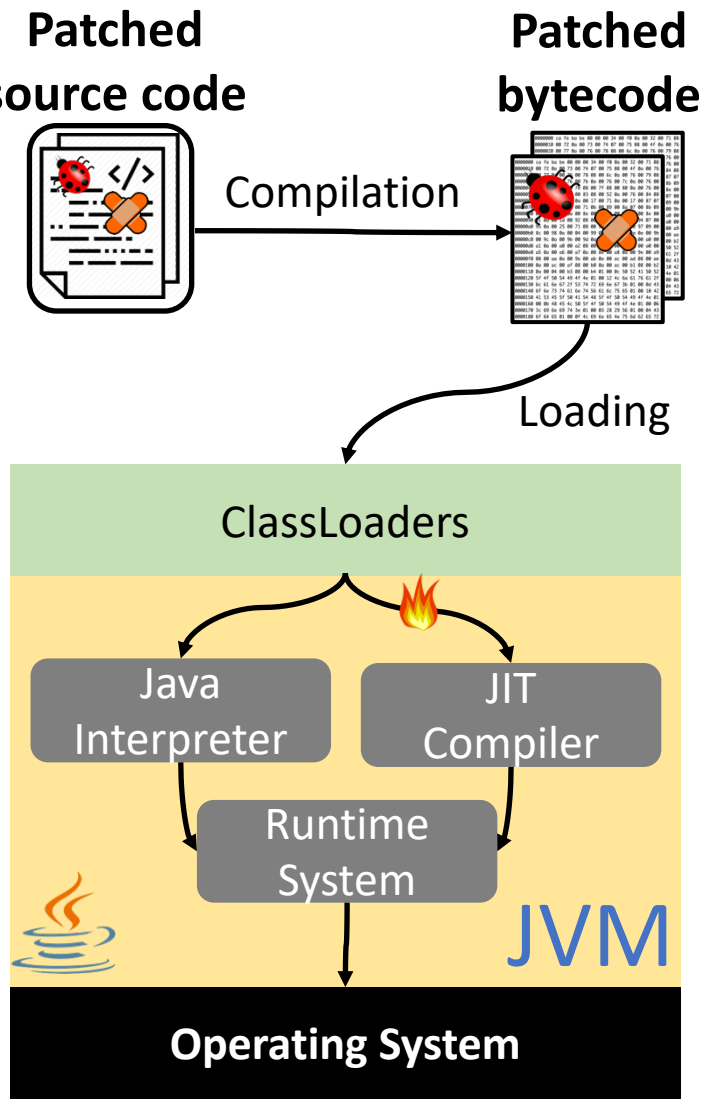
Please ente

Program that repairs programs: how to achieve 78.3 percent precision in automated program repair

August 4, 2017 | By Microsoft blog editor

02
09

However, patch validation is costly



- Traditional Java patch validation:
  - Compilation
  - Java Virtual Machine (JVM) loading
  - Execution against all test inputs

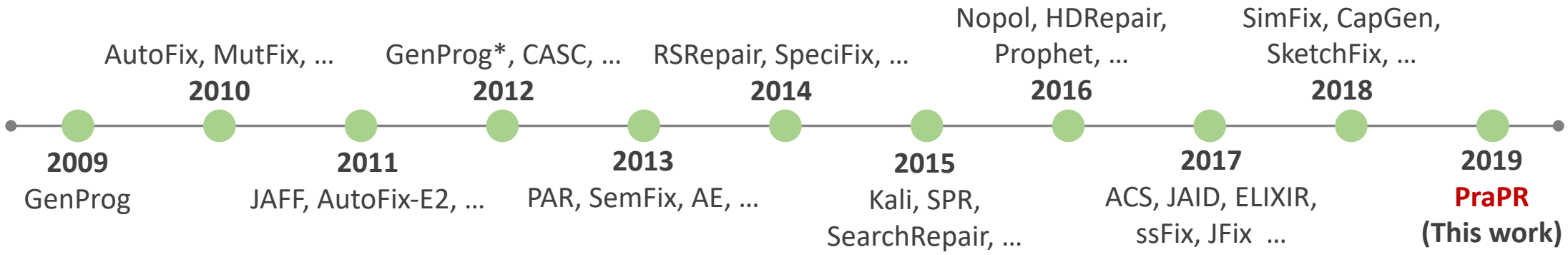
APR Tools	Validating 1 patch	50,000 patches
SimFix [ISSTA'18]	10.4s	144.5h
SketchFix [ICSE'18]	32.0s	444.5h
JAID [ASE'17]	7.7s	107.0h
…		

Google **Closure** **>250,000LOC > 7,000 Tests**

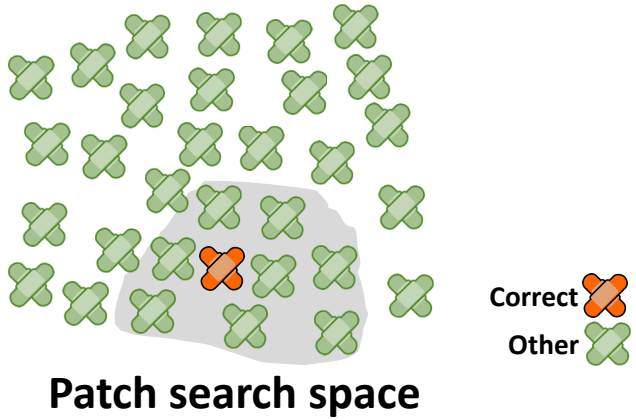
CashCore, with over **1M LOC**, costs almost **a year** for 50,000 patches!

Alipay™

Scaling APR to real-world systems



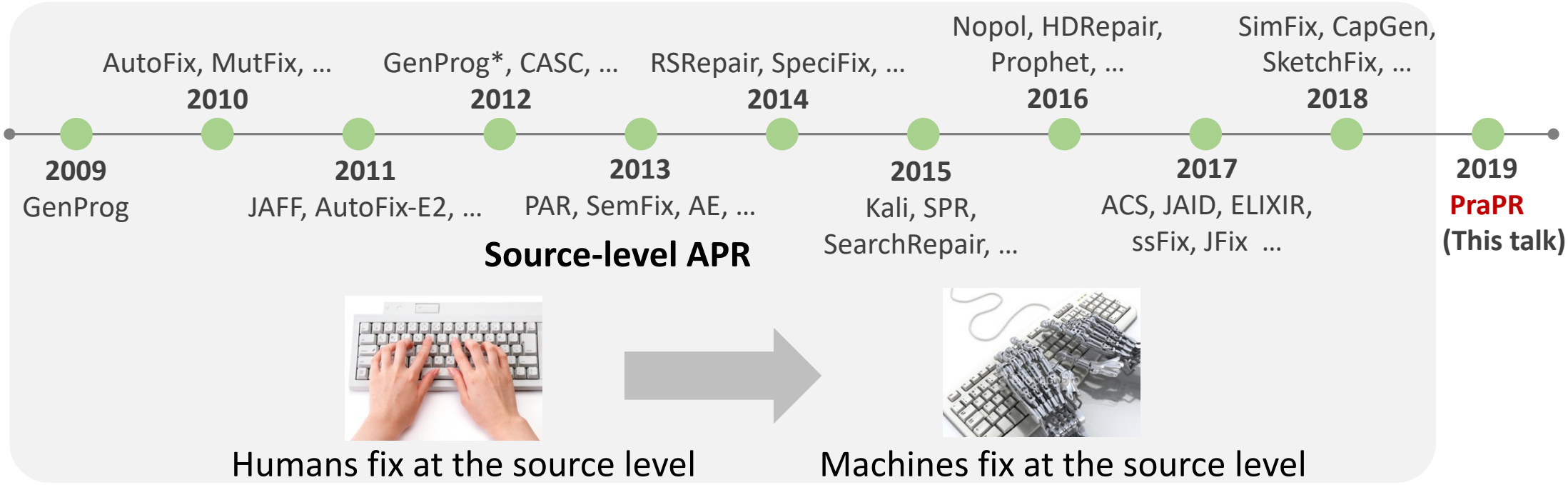
- Techniques to reduce patch search space:
  - **Code search**: SimFix, ssFix, …
  - **Machine learning**: ELIXIR, Prophet, …
  - **Constraint solving/synthesis**: Nopol, ACS, …
  - **Fixing-pattern mining**: CapGen, HDRepair, …
  - …



Patch search space

Potential side effects: dataset overfitting and/or **scalability** issues

Revisiting the APR problem after 10 years

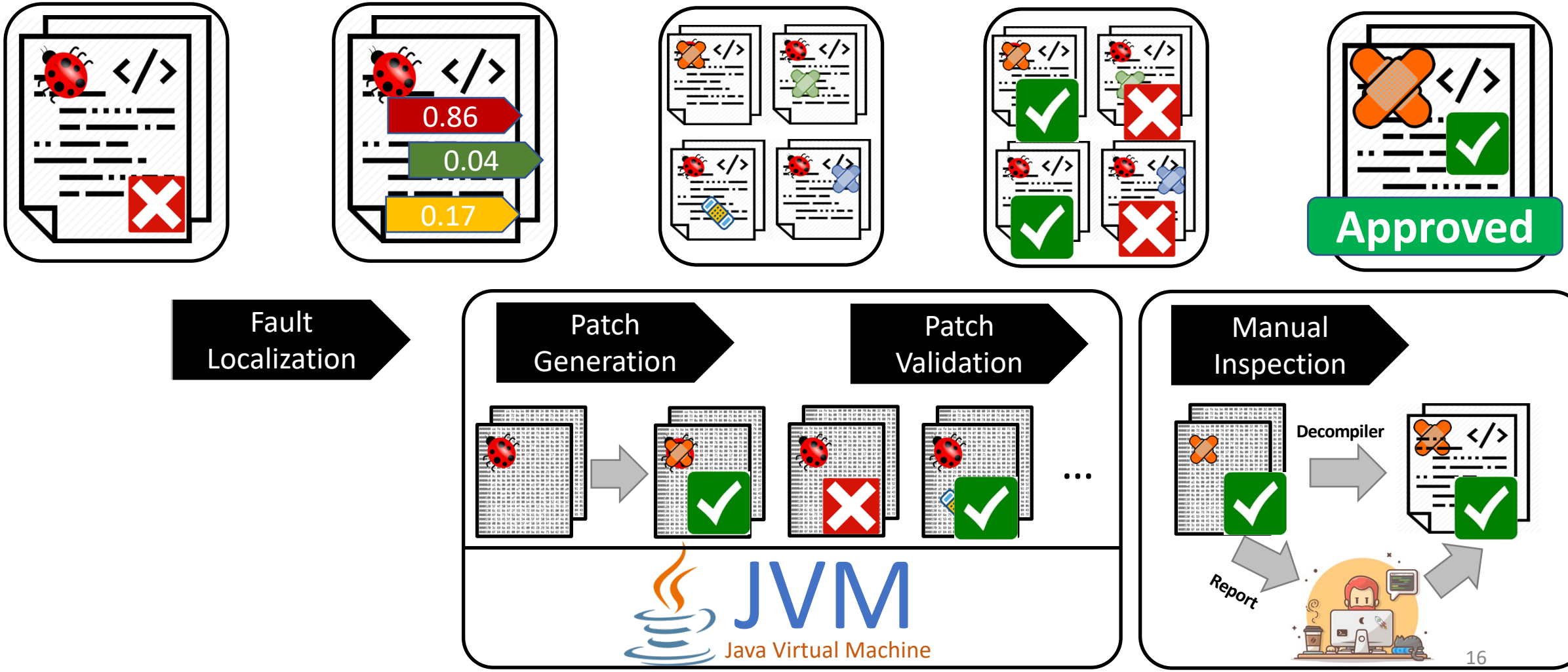


Our insight: let the machines fix at **their own level**!

Prior work: reducing the number of patches

Our work: reducing the time on each patch

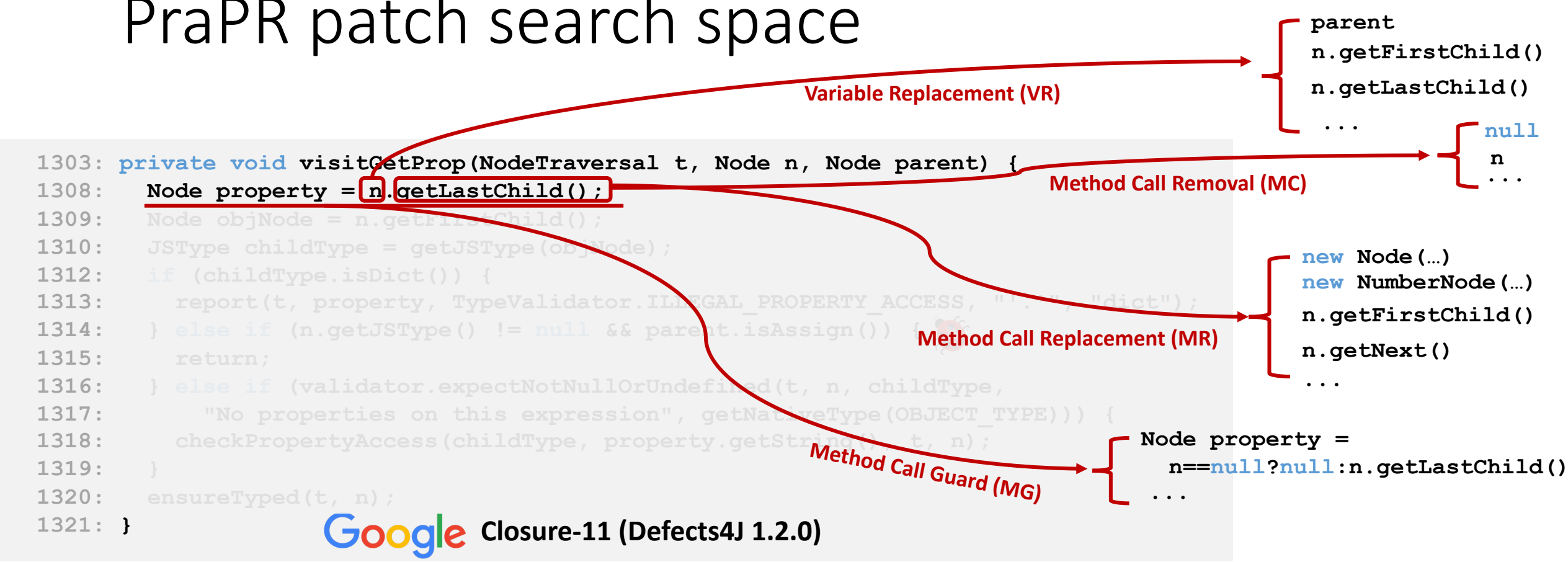
Practical Program Repair via On-the-Fly Bytecode Manipulation (PraPR)



PraPR system design

- **Patch search space**
- **Bytecode patching**
  - Handling all JVM instructions and data types
- **On-the-fly patching**
  - Sharing JVM across patches
- **Handling various class-loading mechanisms**
- **Handing modern JVM-based projects**
  - Multi-module
  - DB, network, and file accesses
- …

PraPR patch search space



20+ patches for such a simple statement!

Prior work: applying *selective* fixing patterns!

Our insight: applying *basic* fixing patterns *exhaustively*!

PraPR patch search space

```
1314:  - } else if (n.getJSType() != null && parent.isAssign()) {
1315:  -   return;
```

Developer Patch

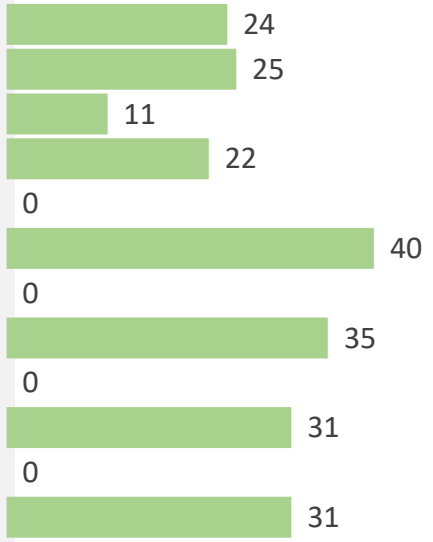
```
1314:  - } else if (n.getJSType() != null && parent.isAssign()) {
1315:  + } else if (null != null && parent.isAssign()) {
```

PraPR Patch (via MC)

```
1303: private void visitGetProp(NodeTraversal t, Node n, Node parent) {
1308:   Node property = n.getLastChild();
1309:   Node objNode = n.getFirstChild();
1310:   JSType childType = getJSType(objNode);
1312:   if (childType.isDict()) {
1313:     report(t, property, TypeValidator.ILLEGAL_PROPERTY_ACCESS, "'.'", "dict");
1314:   } else if (n.getJSType() != null && parent.isAssign()) {
1315:     return
1316:   } else i
1317:       "No
1318:     checkPropertyAccess(childType, property.getString(), t, n);
1319:   }
1320:   ensureTyped(t, n);
1321: }
```

No prior APR work can fix this bug!

Google **Closure-11 (Defects4J 1.2.0)**



Patching Frequency (tot.=219)

Over **250,000** statements for Closure-11

→ **3,744** candidate statements for patching

→ **46,926** patches

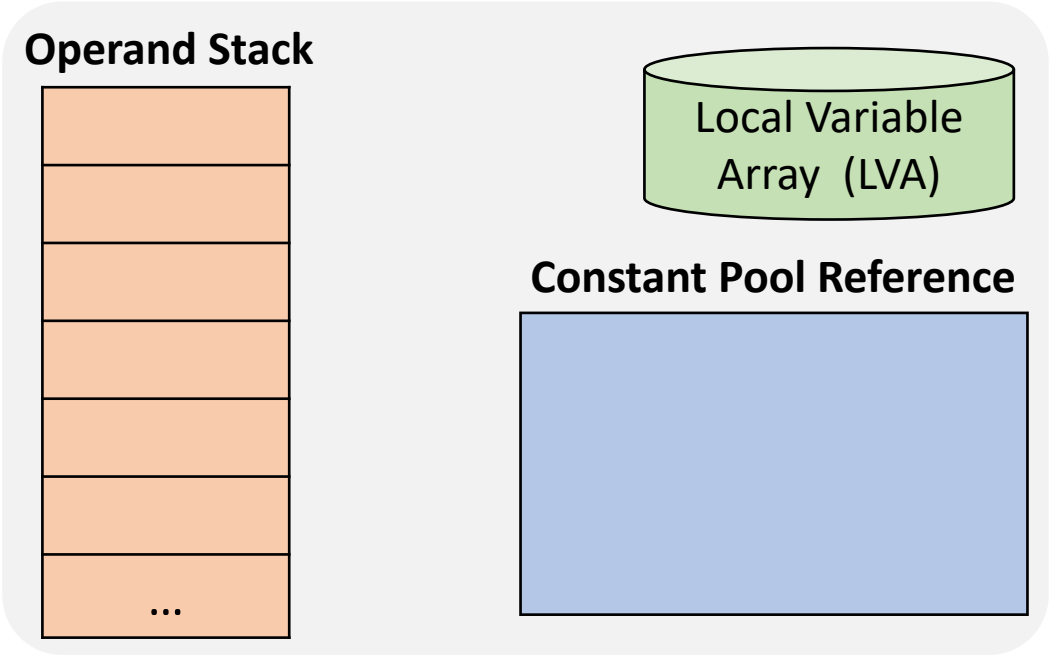
PraPR bytecode patching

Method Guard Pattern (Line 1316, Closure-11)

Google

```
- validator.expectNotNullOrUndefined(t, n, childType,"No…",getNativeType(…))
+ validator==null? true: validator.expectNotNullOrUndefined(t, n,childType,"No…",getNativeType(…))
```

JVM Stack Frame



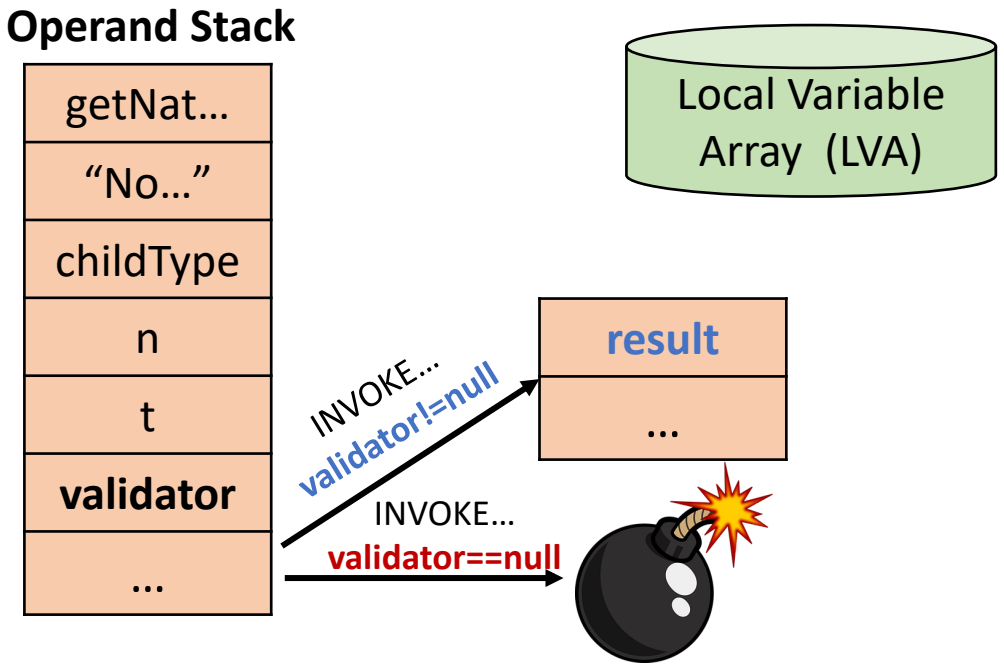
Our insight: directly manipulate JVM operand stack and LVA for fast bytecode fixing

PraPR bytecode patching

Method Guard Pattern (Line 1316, Closure-11)

Google

```
- validator.expectNotNullOrUndefined(t, n, childType,"No…",getNativeType(…))
+ validator==null? true: validator.expectNotNullOrUndefined(t, n,childType,"No…",getNativeType(…))
```

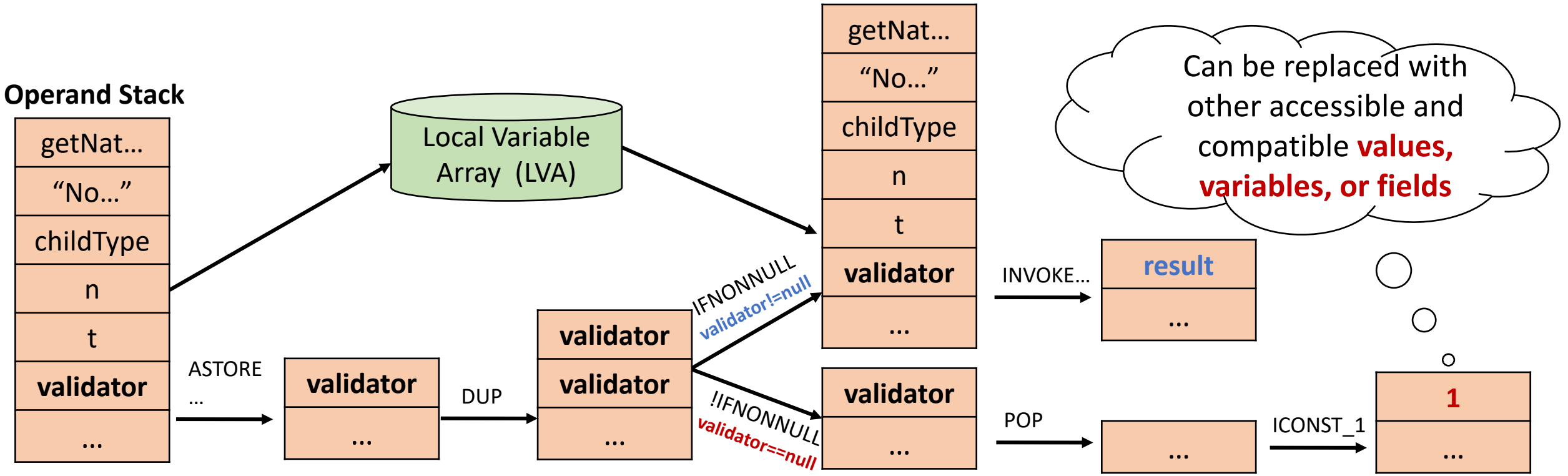


Original unpatched version

PraPR bytecode patching

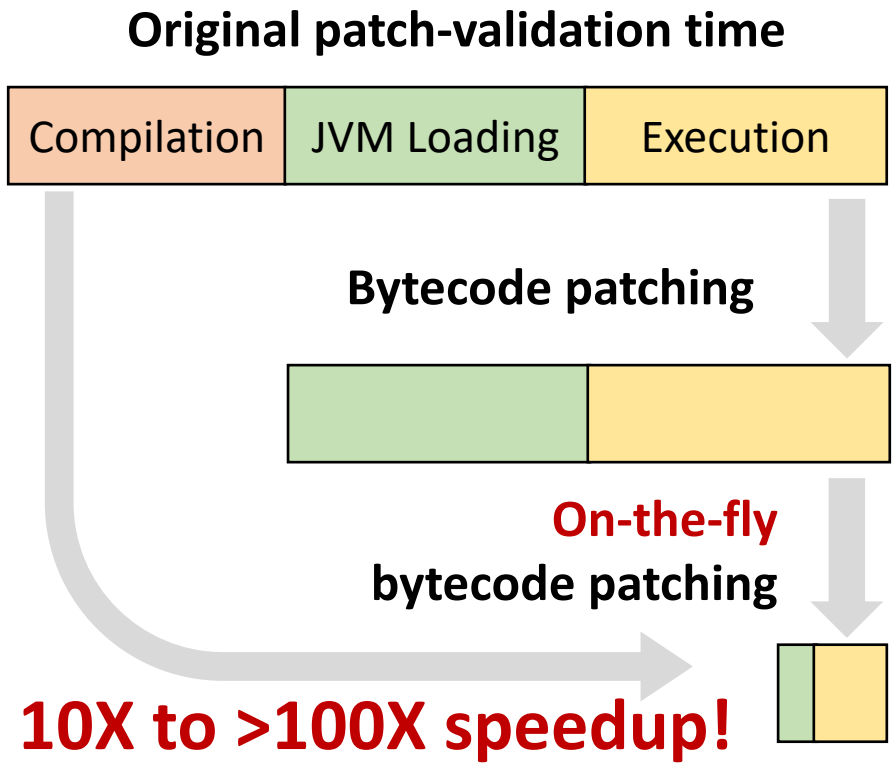
Method Guard Pattern (Line 1316, Closure-11)

```
- validator.expectNotNullOrUndefined(t, n, childType,"No…",getNativeType(…))
+ validator==null? true: validator.expectNotNullOrUndefined(t, n,childType,"No…",getNativeType(…))
```



Patched version via manipulating JVM operand stack and LVA

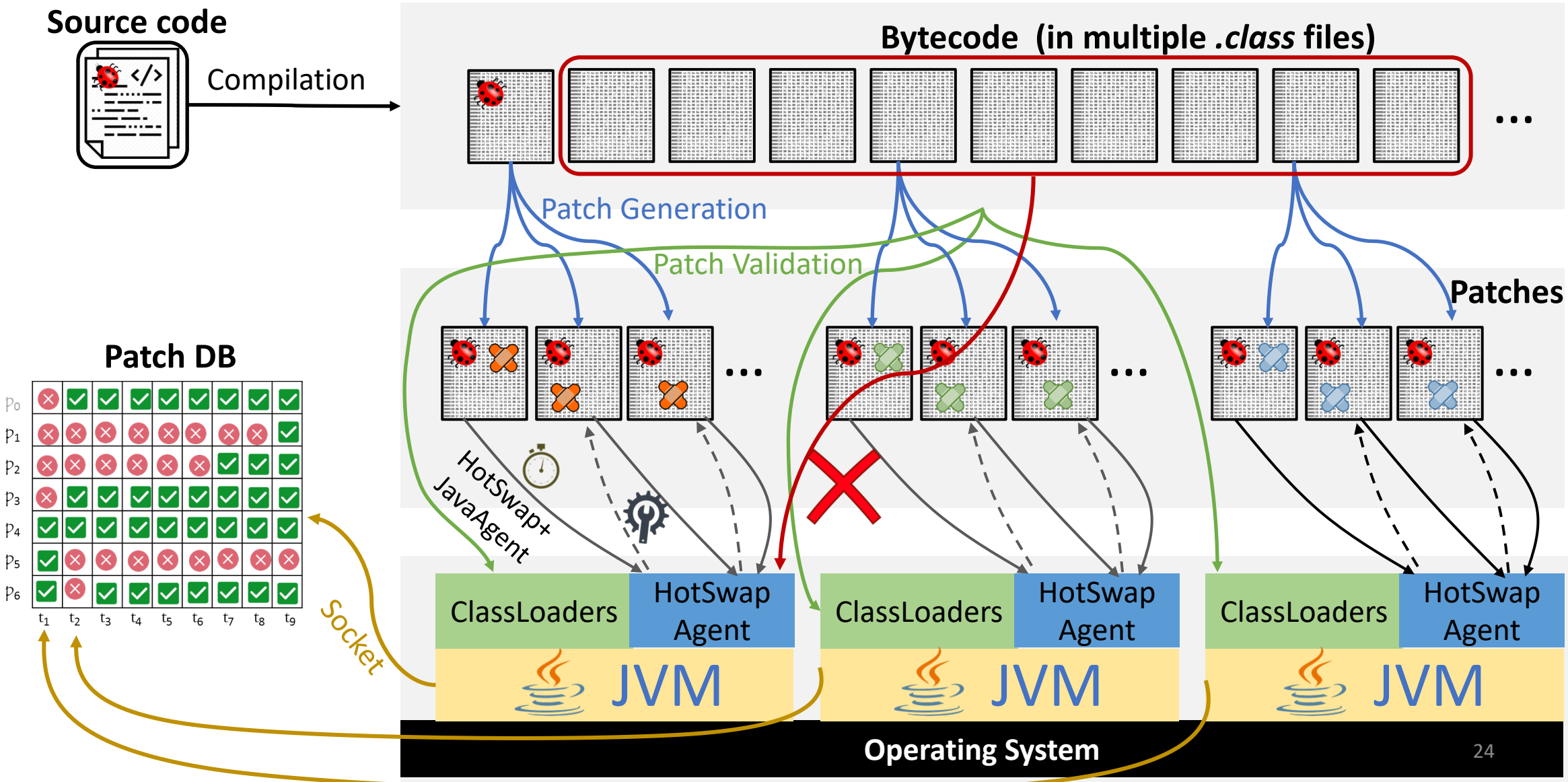
Why on-the-fly bytecode patching?



- Starting a JVM for each patch is costly [Lion et al.]
  - Load/link/initialize all used bytecode class files
    - **140,000,000+** class loadings for Closure-11
  - Deploy used bundles/services
    - AliPay projects
- **Our insight**: **share JVM across patches on-the-fly**
  - **Minimized loading**: only reload patched class(es)
  - **Faster execution**: share across patches:
    - *JVM profiling information*
    - *Already JIT-optimized* code

❑ Lion et al., "Don't Get Caught In the Cold, Warm-up Your JVM", OSDI'16

On-the-fly bytecode patching



https://github.com/prapr/prapr

PraPR system

- A **one-click** APR tool publicly available on GitHub, docker, Maven
  - Supports full set of JVM instructions and data types
- Plugin supports for modern build systems

Gradle maven

```
mvn org.mudebug:prapr-plugin:2.0.2:prapr
```

- Applicable to popular testing frameworks

JUnit TestNG Spek

- Applicable to other popular JVM languages

Java Kotlin

Modern JVM languages: Kotlin

- **No.1** preferred language for Android at **Google I/O 2019**
- Over **50%** Android developers are using Kotlin

PraPR opens the doors for:

- **Fast APR** without patch compilation and system reloading
  - Avoiding the **scalability issue** of prior APR
- **Freestanding APR** without aggressive patch pruning
  - Avoiding the **dataset-overfitting issue** of prior APR
- **Universal APR** for
  - Code with/without source information
  - **Hundreds of JVM languages**!

Resets the starting point, baseline, and scope for future APR

Benchmark projects

Benchmarks	Language	# of Bugs	Code Size (LOC)
Defects4J (1.2.0) [Just et al.]		395	60K -- 260K
Defects4J (1.4.0) [Gay et al.]		612	30K -- 260K
DefeXts [Benton et al.]		225	248 -- 170K

Total Lines of Code: >**50M**

- Defects4J (1.2.0): the most widely used APR benchmark suite
- The first APR study on Defects4J (1.4.0)
- The first APR study for Kotlin (on DefeXts)

- Just et al., "Defects4J: a database of existing faults to enable controlled testing studies for Java programs", ISSTA'14 Demo
- Gay et al., "Defect4J V1.4.0: https://github.com/Greg4cr/defects4j/tree/additional-faults-1.4", 2019
- Benton et al., "Defexts: A Curated Dataset of Reproducible Real-World Bugs for Modern JVM Languages". ICSE'19 Demo

State-of-the-art APR tools for comparison

Technical Basis	APR Tools
Code Search	SimFix [ISSTA'18], ssFix [ASE'17]
Sketching	SketchFix [ICSE'18]
Pattern Mining	CapGen [ICSE'18]
Meta-program	JAID [ASE'17]
Synthesis/Constraint Solving	NOPOL [TSE'16], ACS [ICSE'17]
Machine Learning	ELIXIR [ASE'17]
Genetic Programming	xPAR [ICSE'13], jGenProg [ICSE'09, ICSE'12]
Pattern-based	HDRepair [SANER'16], jkali [ISSTA'15], jMut [ICST'10]

PraPR effectiveness *(Defects4J 1.2.0)*

- Fixing **more bugs**
  - Fixing 43 bugs, **27%** more than the most effective **SimFix** [Jiang et al.]
  - Fixing 10 bugs not fixed by any existing APR
- Fixing bugs **over 10X faster**!

APR Tools	Validating 1 patch	50,000 patches
SimFix [ISSTA'18]	10.4s	144.5h
SketchFix [ICSE'18]	32.0s	444.5h
JAID [ASE'17]	7.7s	107.0h
…		

Google

Closure

>250,000LOC
> 7,000 Tests

❑ Jiang et al., "Shaping Program Repair Space with Existing Patches and Similar Code", ISSTA'18

PraPR effectiveness *(Defects4J 1.2.0)*

- Fixing **more bugs**
  - Fixing 43 bugs, **27%** more than the most effective **SimFix** [Jiang et al.]
  - Fixing 10 bugs not fixed by any existing APR
- Fixing bugs **over 10X faster**!

APR Tools	Validating 1 patch	50,000 patches
SimFix [ISSTA'18]	10.4s	144.5h
SketchFix [ICSE'18]	32.0s	444.5h
JAID [ASE'17]	7.7s	107.0h
…		
PraPR (this work)	**0.22s**	**3.0h**

Google

Closure

>250,000LOC
> 7,000 Tests

❑ Jiang et al., "Shaping Program Repair Space with Existing Patches and Similar Code", ISSTA'18

PraPR effectiveness *(Defects4J 1.4.0 and DefeXts)*

- Fixing **62%** more bugs than state-of-the-art APR on Defects4J 1.4.0!
  - **Avoiding the dataset overfitting issue**
- Fixing 12% studied Kotlin bugs from DefeXts
  - **The first successful Kotlin APR report**

https://github.com/lgwillmore/jenjin/commit/984f7567c83df2778b3d7887380839b757008340

```
19:    set(value) {
20:        field = value
22: -      this::class.declaredMemberProperties.forEach {
22: +      this::class.memberProperties.forEach {
23:        …
```

jenjin: a multimodule Kotlin game engine with 22,261LoC

Fixed in **1min** (exploring 1057 patches)!

Resources

- **Tarantula**: Visualization of test information to assist fault localization (ICSE'02)
  - Paper: https://faculty.cc.gatech.edu/~john.stasko/papers/icse02.pdf
- **PraPR**: PraPR: Practical Program Repair via Bytecode Mutation (ISSTA'19)
  - Paper: http://lingming.cs.illinois.edu/publications/issta2019a.pdf
  - Tool: https://github.com/prapr/prapr
- Want to know more about automated debugging?
  - https://www.computer.org/csdl/journal/ts/2016/08/07390282/13rRUwh80Dh
  - https://www.comp.nus.edu.sg/~abhik/pdf/cacm19.pdf

Thanks and stay safe!