ORACLE

Oracle Parfait – The Journey Towards Security Productivity

Cristina Cifuentes

Vice President of Software Assurance

Oracle Labs

June 2025

Oracle Parfait

The Oracle Parfait static code analysis tool is used by thousands of developers worldwide on a day-to-day basis over commercial and open source codebases of multi-million lines of code and tens of thousands of cloud services.

Key Competitive Advantages Of Oracle Parfait

Finds Real Defects

- Parfait finds defects in large projects without any advanced configuration

Simple Build Integration

- The Parfait build plugins integrate seamlessly in native builds
- The overhead of the compilation is only around 50%, compared to 100%-1,000% for a typical commercial SAST tool

High Rate Of True-Positives

- Parfait boasts an estimated TP rate of 80-90%
- Weekly benchmarking on more than 200 projects

It's Fast!

- Parfait analyzes projects of 1MLOC+ in minutes, not hours!

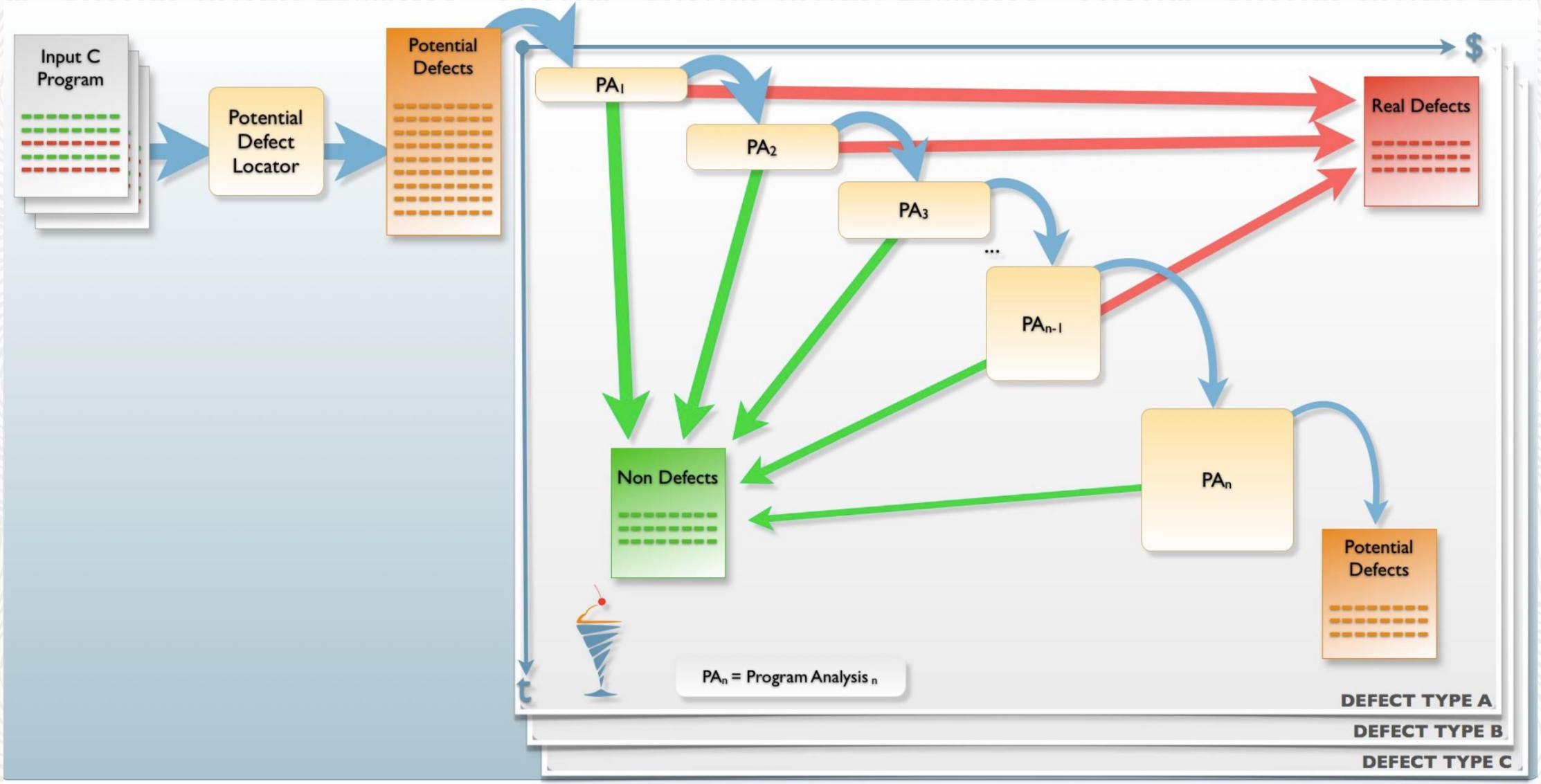
Tool of choice at Oracle

- Deployed at Oracle on more than 6,000 projects and almost billions of lines of code

The Parfait Design and Implementation

2007 Design

2007 – 2018 Implementation



Scalability

Layered approach

Demand-driven analyses

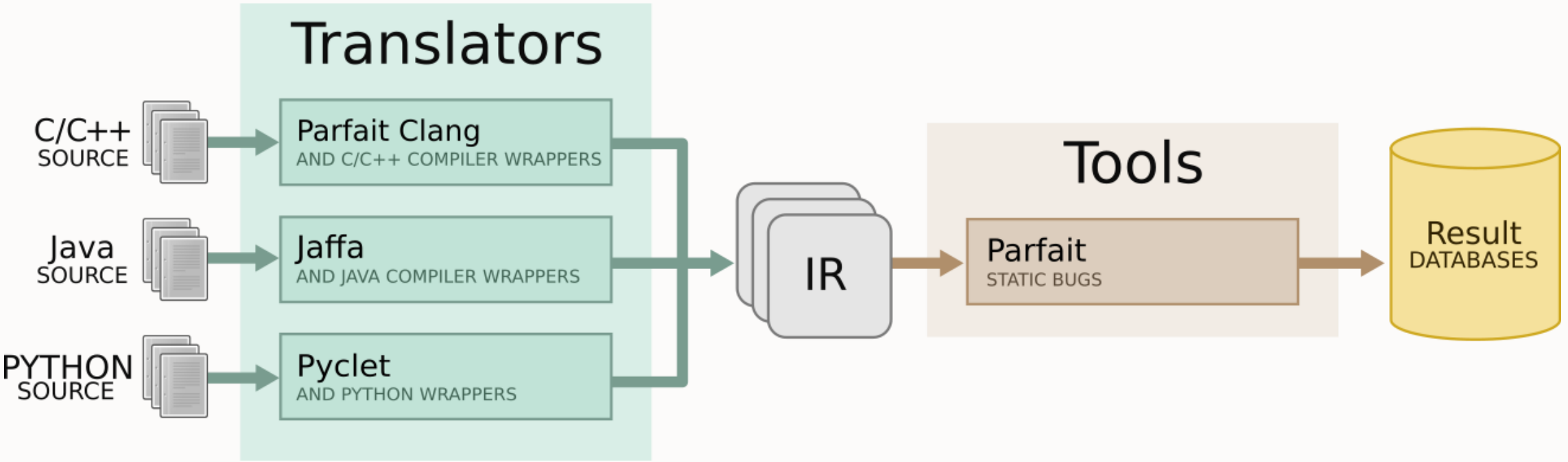
Multiple ways to parallelise the framework

Multiple lists of bugs (NoBugs, PotentialBugs, RealBugs)

Bugs moved from PotentialBugs to RealBugs list conservatively

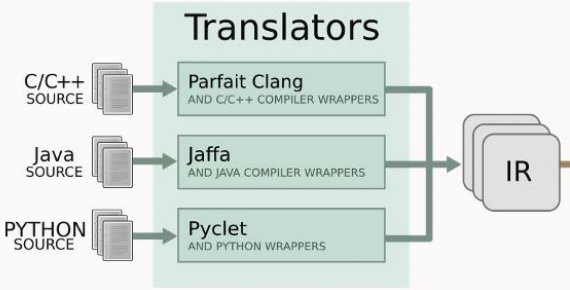
Precision

The Parfait Implementation



Built on top of LLVM

Build Integration for Make (C, Java) and Python



Drop-in replacement for C compiler

```
parfait-gcc -o test test.c
parfait test.bc
```

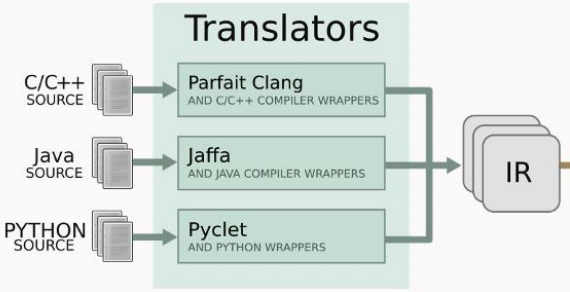
Drop-in replacement for Java compiler

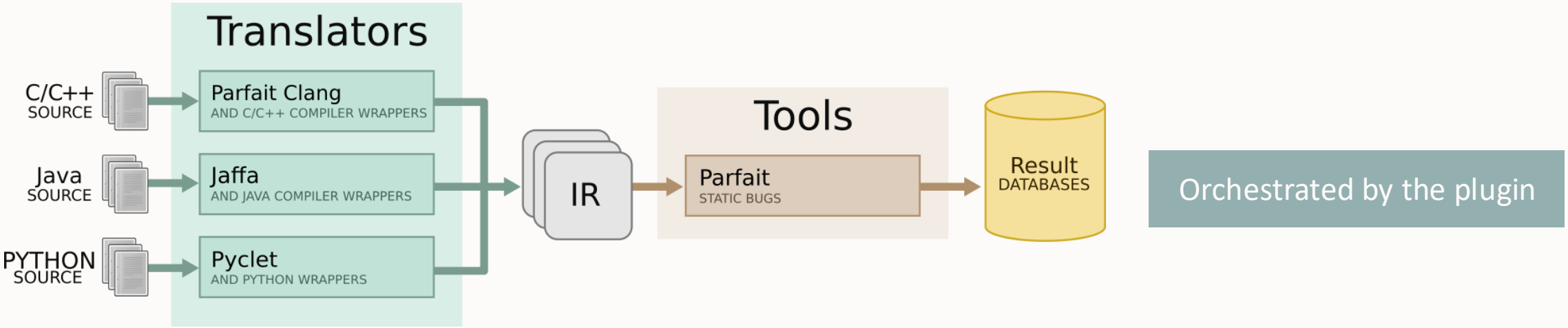
```
parfait-javac -o test test.java
parfait test.bc
```

Drop-in replacement for Python bytecode compiler

```
parfait-python -p test-dir -o test.bc test-dir
parfait test.bc
```

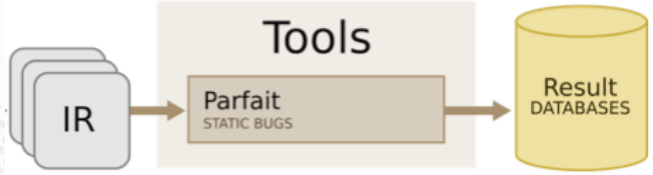
Build Integration with Maven and Gradle Plugins (Java)






```
buildscript {
    repositories {
        maven { url 'https://<artifactory-parfait-release>'} }
    dependencies {
        classpath 'oracle.parfait:gradle.plugins:1.0.5'}
}
apply plugin: 'oracle.parfait'
```

Sample Analyses



Data flow analysis

- Keeps track of data values at each point in the program

Partial evaluation

- Executes partially-evaluated slice of a potential bug

Symbolic analysis

- Symbolically tracks values of a program slice of interest

Control flow analysis

- Keeps track of flow of control through the program

Taint analysis

- Keeps track of data that is user controllable

Leak analysis

- Keeps track of sensitive data that reaches lower privileged parts of the application

Bugs and Vulnerabilities that Matter

C, C++

- Buffer overflows
- Memory/pointer bugs
  - NULL pointer dereference, use after free, double free, memory leak, …
- Integer overflow

Java EE

- SQL injection, cross-site scripting (XSS), LDAP injection, OS injection, …
- XXE/XEE
- Insecure crypto
- Insecure deserialization

Java Platform

- Unguarded caller-sensitive method calls
- Unsafe use of doPrivileged
- Call to overridable method during deserialization

Python

- SQL injection, command injection
- Insecure deserialization
- Unsafe eval

Bugs and Vulnerabilities that Matter

C, C++

- Buffer overflows
- Memory/pointer bugs
  - NULL pointer dereference, use after free, double free, memory leak, …
- Integer overflow

Java EE

- SQL injection, cross-sit
  injection, OS injection, .
- XXE/XEE
- Insecure crypto
- Insecure deserialization

Cross-language vulnerabilities (Java - C)

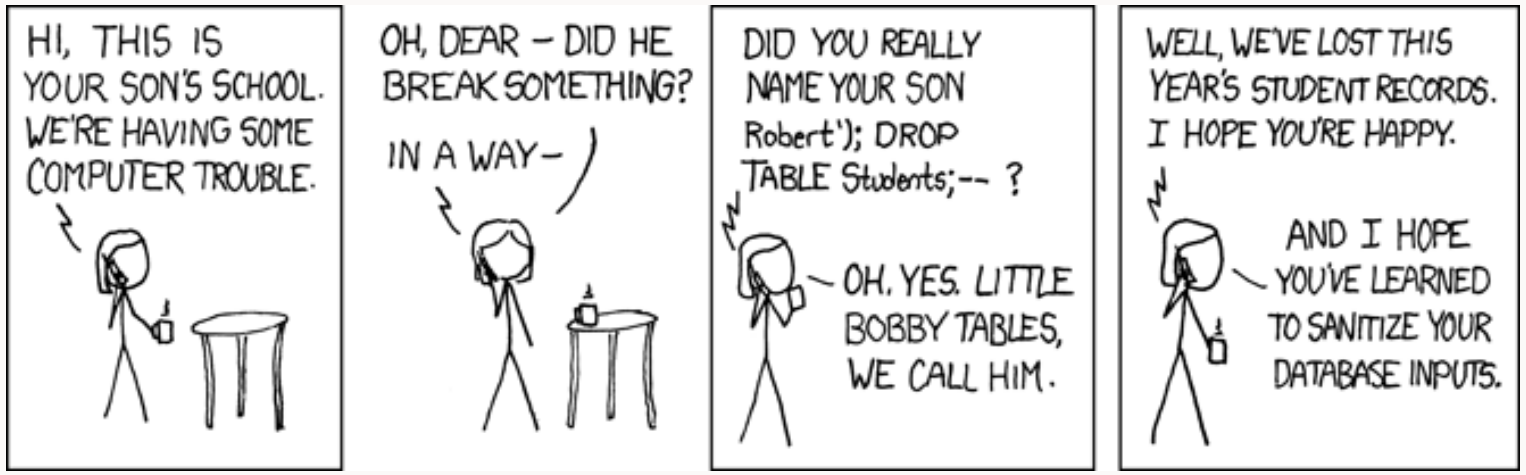
- Buffer overflows
- Dereference of untrusted pointer
- SQL injection
- All injection vulnerabilities

Java Platform

- Unguarded -sensitive method calls
- U rivileged
- method during deserialization

- QL injection, command injection
- Insecure deserialization
- Unsafe eval

Detecting SQL Injection in C, Java, and Python code



Taint Using Dataflow Analysis for SQL Injection

```
protected Element createContent(WebSession s)
{
    ...
    password = s.getParser().getRawParameter(PASSWORD);
    ...
    String query = "SELECT * FROM user_system_data WHERE user_name = '" + username +
                   "' and password = '" + password + "'";
    ...
    try {
        Statement statement =
                            connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                       ResultSet.CONCUR_READ_ONLY);
        ResultSet results = statement.executeQuery(query);
        ...
    }
    ...
}
```

https://github.com/WebGoat/WebGoat

Taint Using Dataflow Analysis for SQL Injection

```
protected Element createContent(WebSession s)
{
    ...
    password = s.getParser().getRawParameter(PASSWORD);
    ...
    String query = "SELECT * FROM user_system_data WHERE user_name = '" + username +
                   "' and password = '" + password + "'";
    ...
    try {
        Statement statement =
                            connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                       ResultSet.CONCUR_READ_ONLY);
        ResultSet results = statement.executeQuery(query);
        ...
    }
    ...
}
```

Taint Using Dataflow Analysis for SQL Injection

```
protected Element createContent(WebSession s)
{
    ...
    password = s.getParser().getRawParameter(PASSWORD);
    ...
    String query = "SELECT * FROM user_system_data WHERE user_name = '" + username +
                   "' and password = '" + password + "'";
    ...
    try {
        Statement statement =
                            connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                       ResultSet.CONCUR_READ_ONLY);
        ResultSet results = statement.executeQuery(query);
        ...
    }
    ...
}
```

Taint Using Dataflow Analysis for SQL Injection

```
protected Element createContent(WebSession s)
{
    password = s.getParser().getRawParameter(PASSWORD);
    ...
    String query = "SELECT * FROM user_system_data WHERE user_name = '" + username +
                   "' and password = '" + password + "'";
    ...
    try {
        Statement statement =
                            connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                       ResultSet.CONCUR_READ_ONLY);
        ResultSet results = statement.executeQuery(query);
        ...
    }
    ...
}
```

Taint Using Dataflow Analysis for SQL Injection

```
public String getRawParameter(String name) throws ParameterNotFoundException {
    String[] values = request.getParameterValues(name);
    if (values == null) {
        throw new ParameterNotFoundException(name + “not found”);
    else if (values[0].length() == 0) {
        throw new ParameterNotFoundException(name + “was empty”);
    }

    return (values[0]);
}
```

Taint Using Dataflow Analysis for SQL Injection

A source of tainted data

```
public String getRawParameter(String name) throws ParameterNotFoundException {
    String[] values = request.getParameterValues(name);
    if (values == null) {
        throw new ParameterNotFoundException(name + "not found");
    else if (values[0].length() == 0) {
        throw new ParameterNotFoundException(name + "was empty");
    }

    return (values[0]);
}
```

Taint Using Dataflow Analysis for SQL Injection

```
public String getRawParameter(String name) throws ParameterNotFoundException {
    String[] values = request.getParameterValues(name);
    if (values == null) {
        throw new ParameterNotFoundException(name + "not found");
    else if (values[0].length() == 0) {
        throw new ParameterNotFoundException(name + "was empty");
    }

    return (values[0]);
}
```

No sanitisation of String values

Taint Using Dataflow Analysis for SQL Injection

```
public String getRawParameter(String name) throws ParameterNotFoundException {
    String[] values = request.getParameterValues(name);
    if (values == null) {
        throw new ParameterNotFoundException(name + "not found");
    else if (values[0].length() == 0) {
        throw new ParameterNotFoundException(name + "was empty");
    }

    return (values[0]);
}
```

Returns tainted String values[0]

Taint Using Dataflow Analysis for SQL Injection

```
protected Element createContent(WebSession s)
{
    password = s.getParser().getRawParameter(PASSWORD);
    ...
    String query = "SELECT * FROM user_system_data WHERE user_name = '" + username +
                   "' and password = '" + password + "'";
    ...
    try {
        Statement statement =
                            connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                       ResultSet.CONCUR_READ_ONLY);
        ResultSet results = statement.executeQuery(query);
        ...
    }
    ...
}
```

tainted String

Taint Using Dataflow Analysis for SQL Injection

```
protected Element createContent(WebSession s)
{
    ...
    password = s.getParser().getRawParameter(PASSWORD);
    ...
    String query = "SELECT * FROM user_system_data WHERE user_name = '" + username +
                   "' and password = '" + password + "'";
    ...
    try {
        Statement statement =
                            connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                       ResultSet.CONCUR_READ_ONLY);
        ResultSet results = statement.executeQuery(query);
        ...
    }
    ...
}
```

A sink of tainted data

org.owasp.webgoat.session.ParameterParser.getRawParameter

org/owasp/webgoat/session/ParameterParser.java

c1c5914ec99978da1a99c7ca7a04dd0e13a7e3a1-nightly

```
513.      * @throws org.owasp.webgoat.session.ParameterNotFoundException
514.      */
515.     public String getRawParameter(String name) throws ParameterNotF
516.         String[] values = request.getParameterValues(name);
```

#1: 'values[...]' is tainted by user-provided input returned by method javax.servlet.ServletRequest.getParameterValues

```
517.
518.         if (values == null) {
519.             throw new ParameterNotFoundException(name + " not found
520.         } else if (values[0].length() == 0) {
521.             throw new ParameterNotFoundException(name + " was empty
522.         }
523.
524.         return (values[0]);
```

#2: '' is tainted (load of tainted 'values[...]')

#3: Tainted '' returned

```
525.     }
526.
527.     /**
528.      * Gets the named parameter value as a short
```

org.owasp.webgoat.plugin.DOS_Login.createContent

org/owasp/webgoat/plugin/DOS_Login.java

Line # | File | Expand | Collapse

c1c5914ec99978da1a99c7ca7a04dd0e13a7e3a1-nightly

```
86.             String password = "";
87.             username = s.getParser().getRawParameter(USERNAME);
88.             password = s.getParser().getRawParameter(PASSWORD);
```

#4: 'password' is tainted by return value of method org.owasp.webgoat.session.ParameterParser.getRawParameter

```
89.
90.             // don;t allow user name from other lessons. it would be too simple.
98.             Connection connection = DatabaseUtilities.getConnection(s);
99.
100.            String query = "SELECT * FROM user_system_data WHERE user_name = '" +
username + "' and password = '"
```

#5: 'java.lang.StringBuilder.append()' is assumed to be tainted by the return value of method java.lang.StringBuilder.append, due to tainted argument 'password'

#6: 'query' is assumed to be tainted by the return value of method java.lang.StringBuilder.toString, due to tainted method receiver 'java.lang.StringBuilder.append()'

```
101.                     + password + "'";
102.            ec.addElement(new StringElement(query));
103.
104.            try
105.            {
106.                Statement statement =
connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
107.
ResultSet.CONCUR_READ_ONLY);
108.            ResultSet results = statement.executeQuery(query);
```

SQL INJECTION
Run SQL query with tainted input 'query'

A source to sink trace for SQL injection example

Finding Spectre Variant 1 Vulnerabilities in C, C++ Code

CVE-2017-5753

Spectre (v1)

CVE-2017-5753

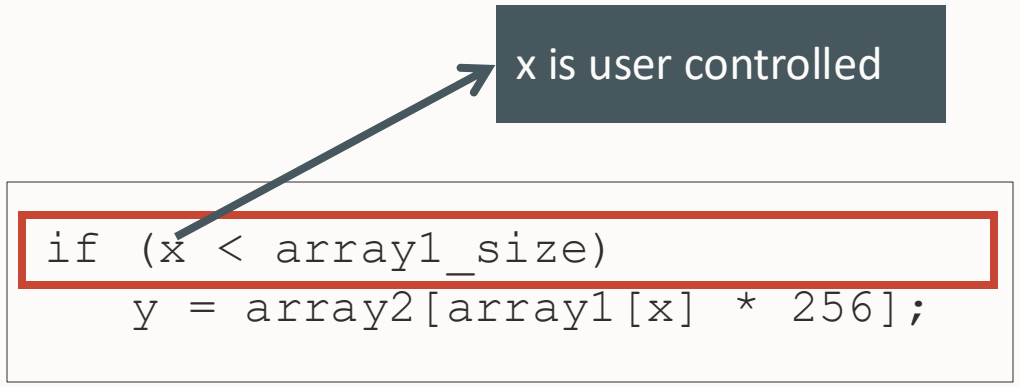
“Systems with microprocessors utilizing speculative execution and branch prediction may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis.”

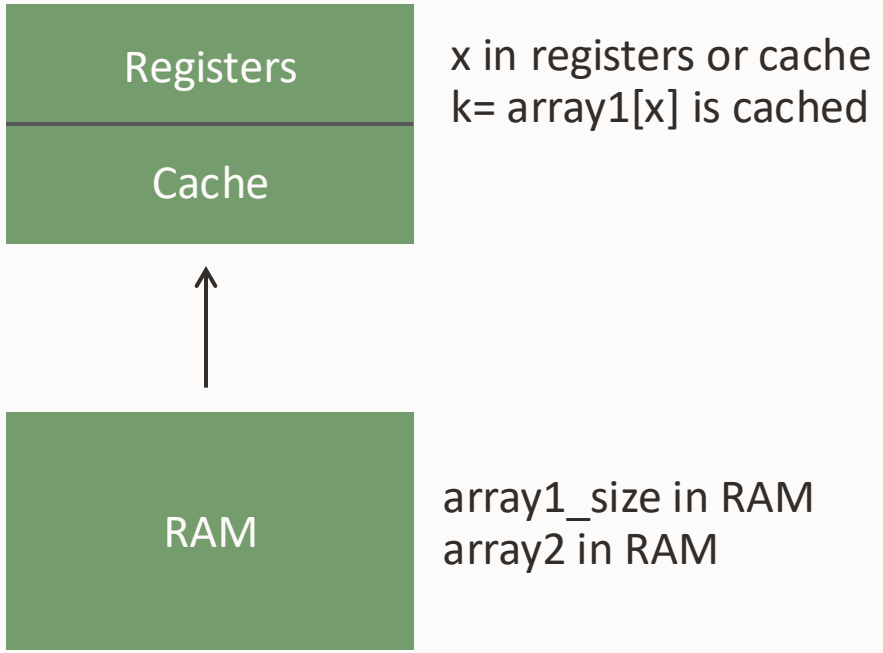
Meltdown

CVE-2017-5754

“Systems with microprocessors utilizing speculative execution and indirect branch prediction may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis of the data cache.”

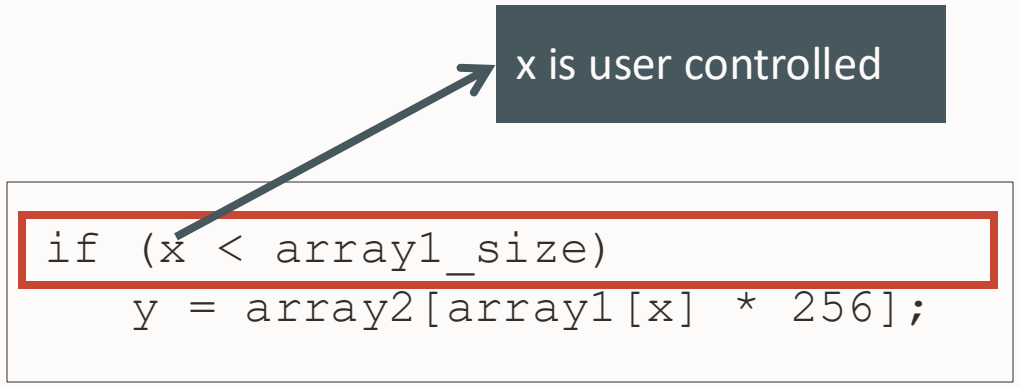
Spectre v1 in a Nutshell





P. Kocher et al, Spectre Attacks: Exploiting Speculative Execution: https://spectreattack.com/spectre.pdf

Spectre v1 Pattern



Branch-load-load

- Branch is a bounds check on first load
- Offset to second load based on first load
- No LFENCE/MEMBAR/array_index_nospec() in the pattern
- Heuristics to determine whether array2 cannot be held in one cache line

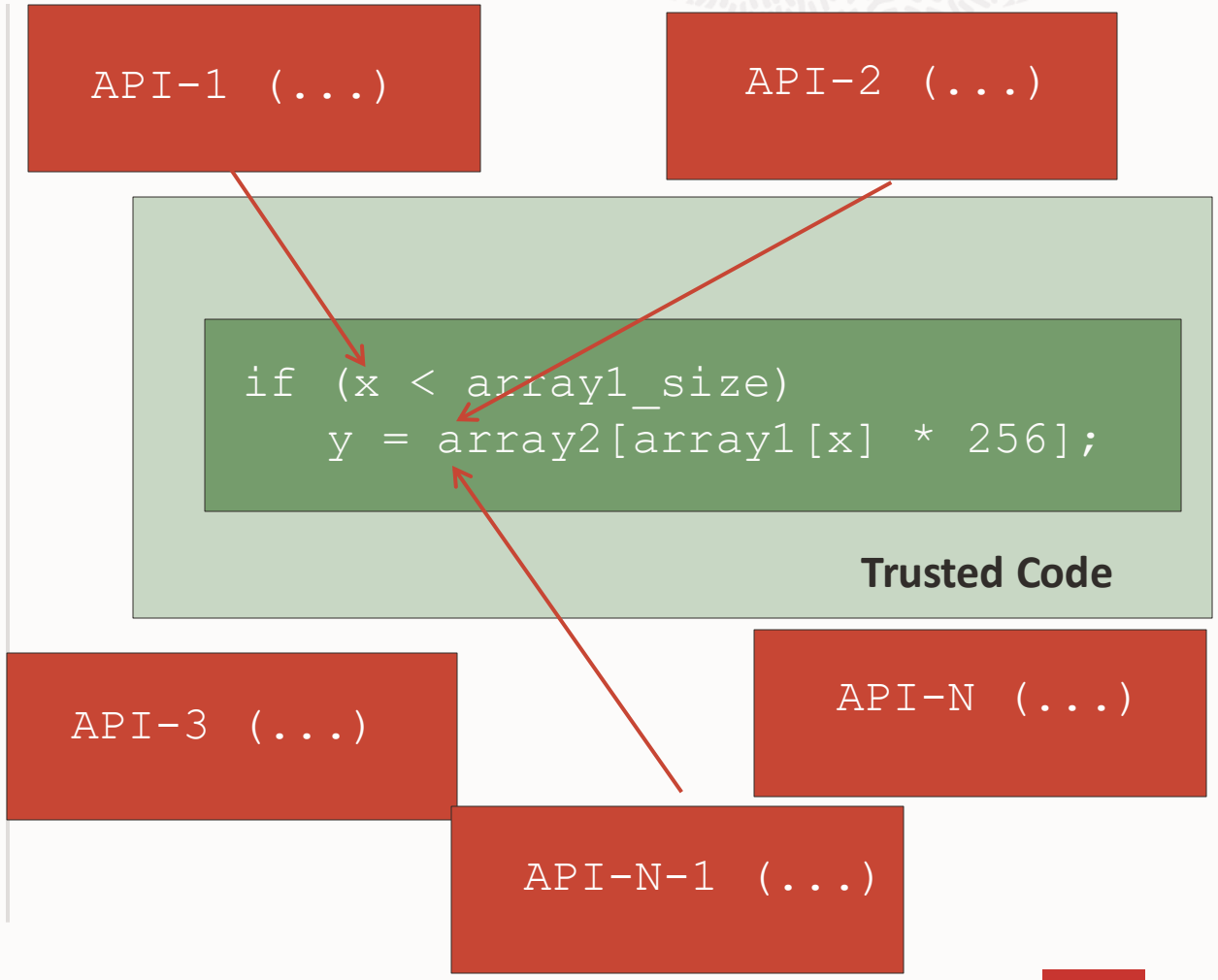
User-controllable offset to first load

Load-load is reachable from less privileged code

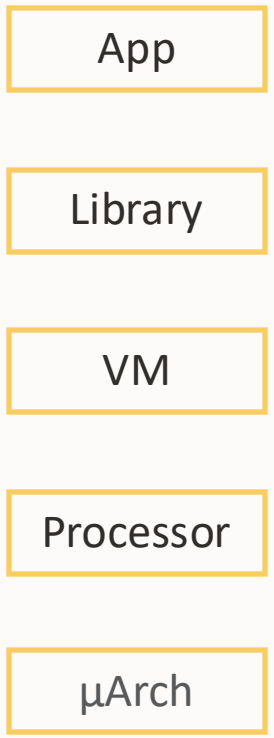
P. Kocher et al, Spectre Attacks: Exploiting Speculative Execution: https://spectreattack.com/spectre.pdf

Rules to Detect Spectre Variant 1

1. Identify branch-load-load pattern
2. Identify the API boundary between more and less privileged code, e.g., syscalls
3. Check (interprocedurally) for reachability and taint from the API entry points to the potential defect



Security Issues Can Arise at Any Level of Abstraction



Each level *provides* a service under certain *assumptions*

Each level *consumes* a service with certain *expectations*

Mismatch between assumptions and expectations can be exploited

Examples

- App and Library: Sanitisation not performed: SQL Injection
- Library and VM: Isolation not guaranteed: information leakage
- Processor and µ$Arch$: Spectre/Meltdown

A Sample of Results

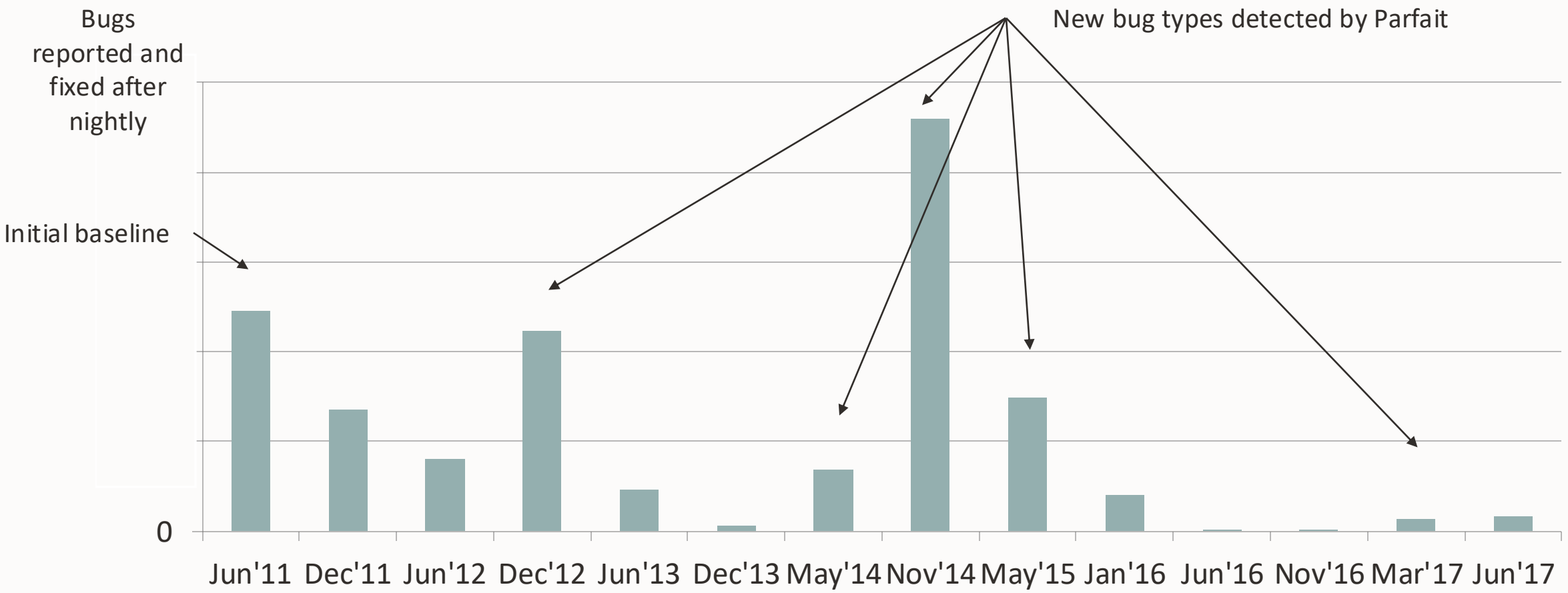
Parfait – Scalable, Deep Static Code Analysis

Language Benchmarks	Lines of Code	Number of Bug Types	Analysis time	Analysis time in KLOC/min
C	48,442,242	74	543m	161 KLOC/min
C++	59,986,380	74	1,089m	69 KLOC/min
Go	7,859,818	22	44m	191 KLOC/min
Java	63,959,798	49	249m	179 KLOC/min
Python	1,666,760	24	47m	47 KLOC/min

Parfait version 2025.04

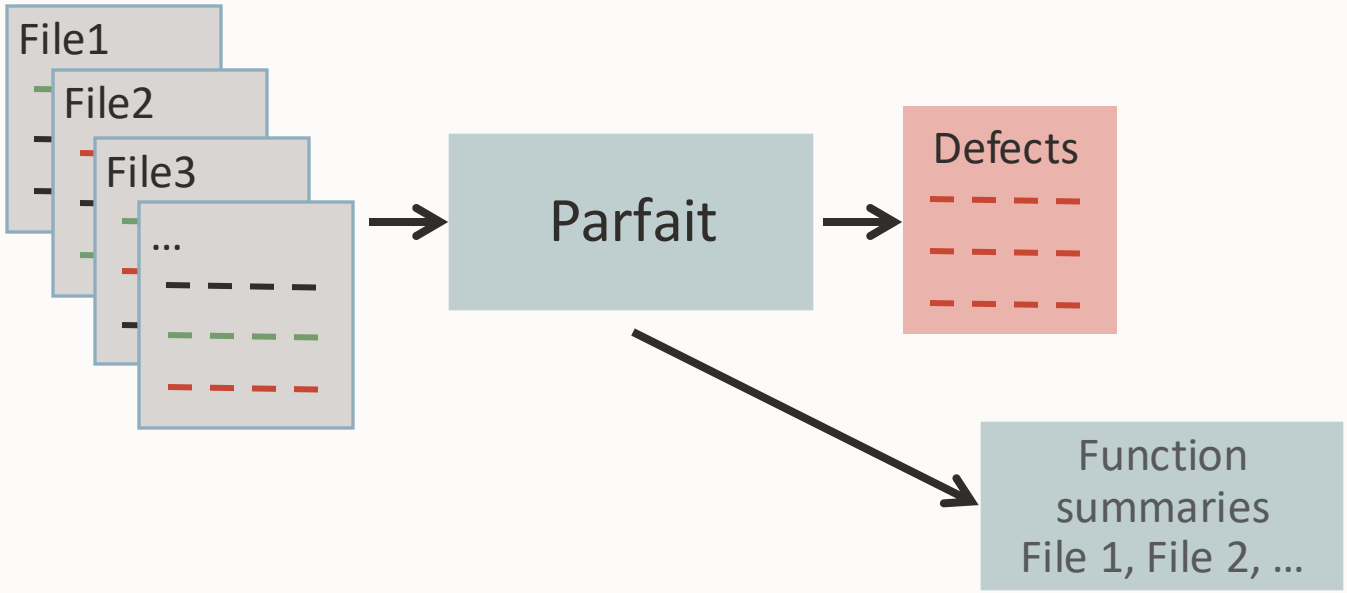
Parfait – Precise, Deep Static Code Analysis

Bugs fixed by developers once **baseline** had been established

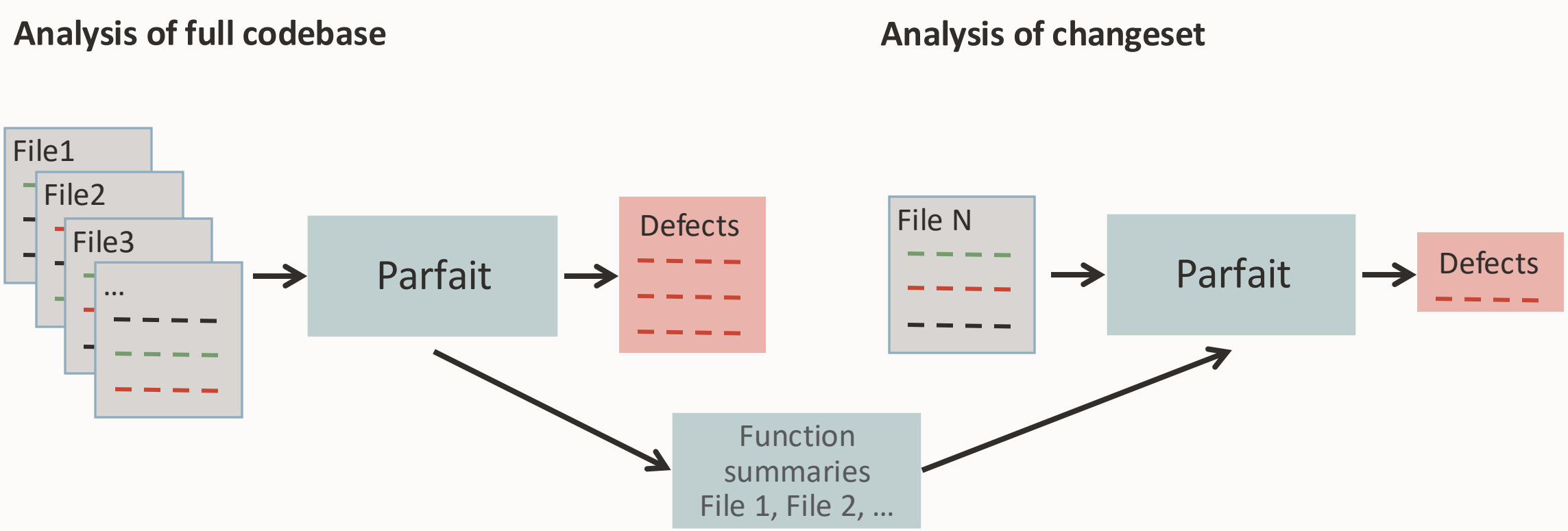


Analysis of Full Codebase vs Analysis of Commit/Push/Pull/Merge Request

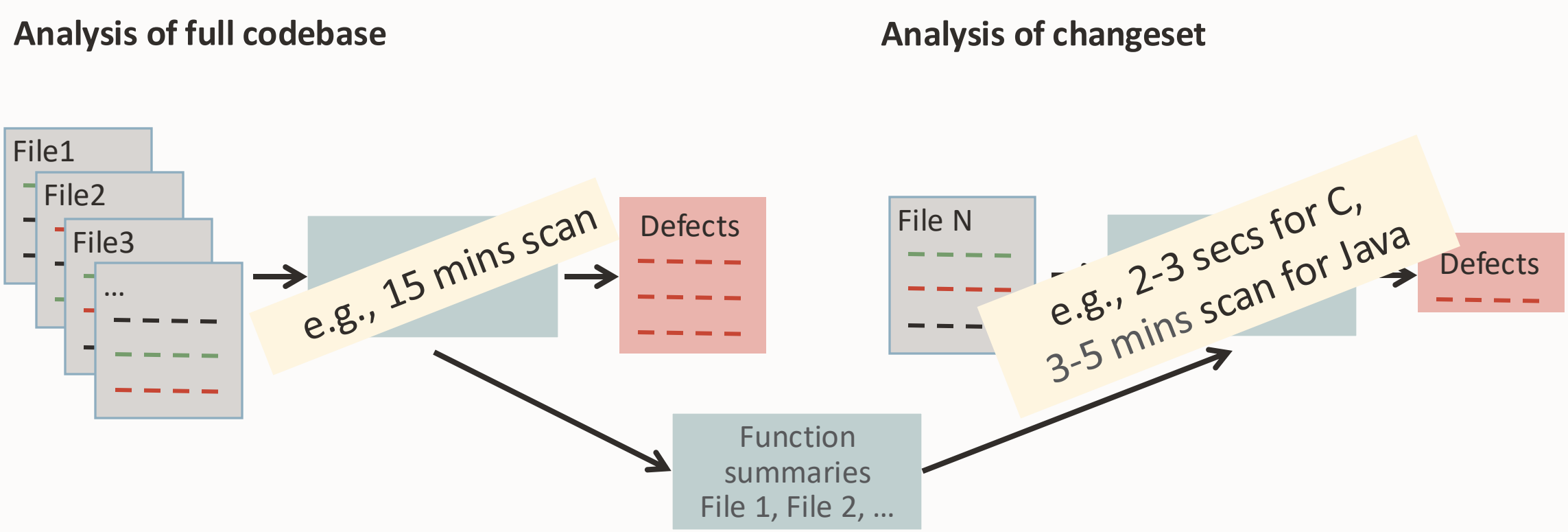
Analysis of full codebase



Analysis of Full Codebase vs Analysis of Commit/Push/Pull/Merge Request

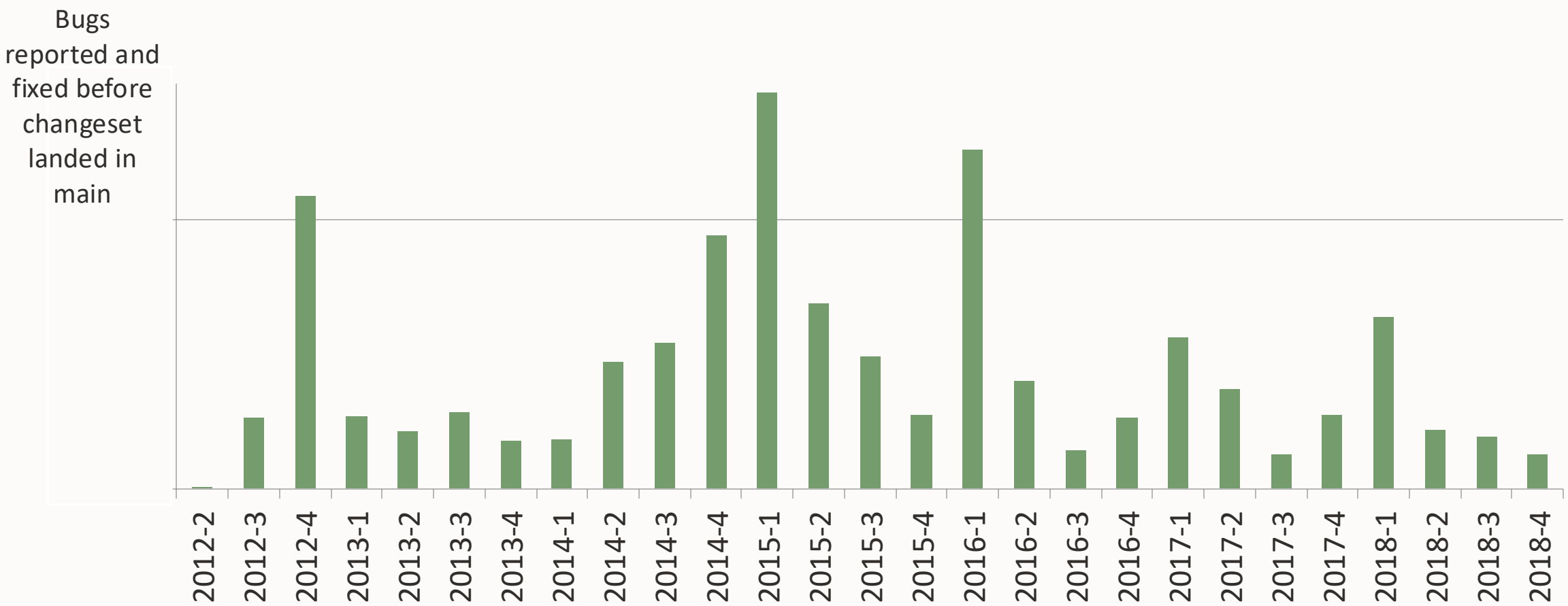


Analysis of Full Codebase vs Analysis of Commit/Push/Pull/Merge Request



Bugs Prevented from Being Introduced into the Codebase

Changeset analysis prevents **80%** of new bugs (compared to baseline)

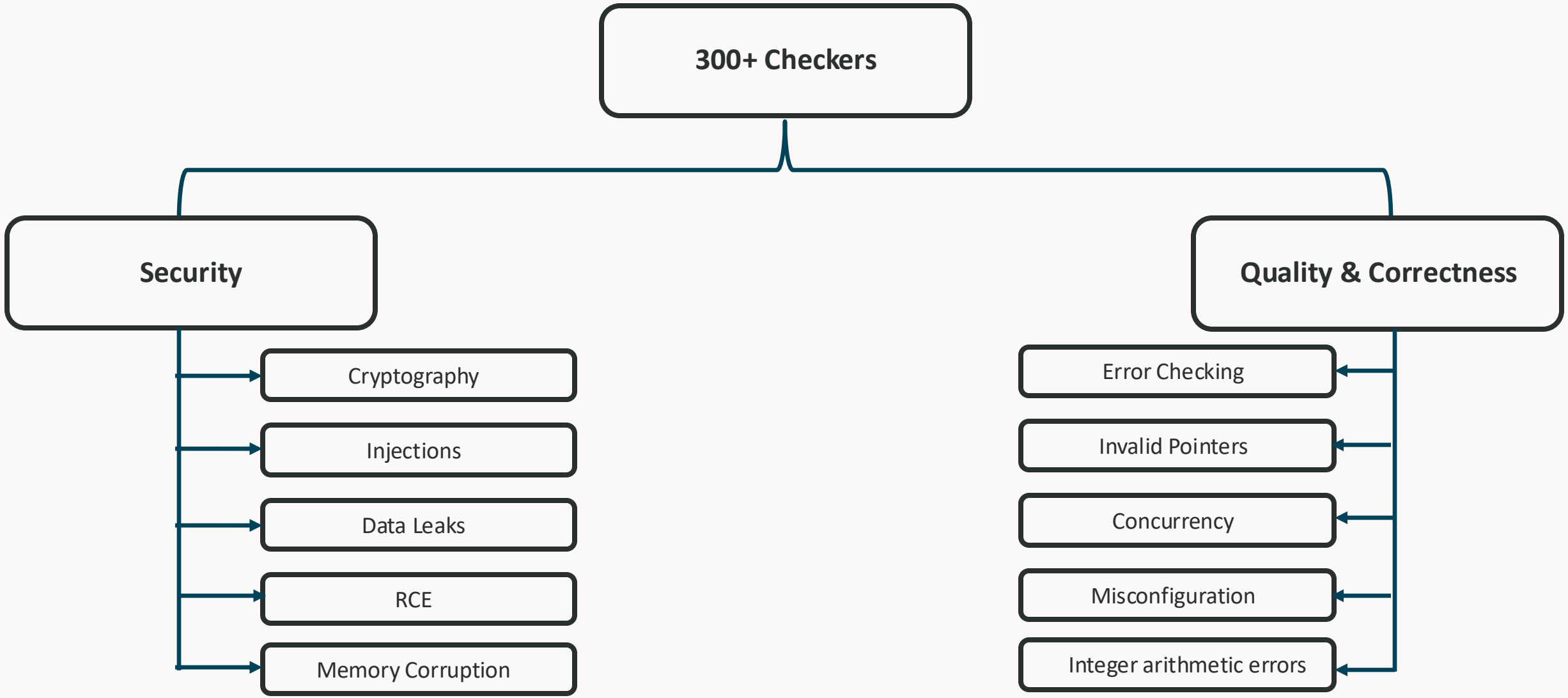


Innovations During the Past 15 Years

- Efficient analysis of full codebase
  - Used to be nightly runs
  - Now part of Continuous Integration
- Efficient analysis of changeset
  - Prevent bugs from being introduced into the codebase
  - Can be hooked into the commit, push, pull request or merge request

Parfait innovations
- Precise results
- Scalable, can integrate early in the development cycle

Supported Checkers



Supported Languages

The Road to Scanning Tens of Thousands of Cloud Services Automatically

Build integration at scale

Key Requirement

Automatic onboarding of Oracle Cloud Infrastructure (OCI) projects with minimal developer effort required

Automatic Onboarding of Cloud Services

Use pre-existing OCI Build Service conventions & configurations to run Oracle Parfait's translation step:

- OCI build config
  - Language information
  - JDK versions
- Security scanning service config
  - Build parameters required by Parfait plugins & compiler wrappers

Test Deployment Snapshot

Scans summary over the past 90 days

Language/Build Tool	Projects	Successfully Scanned Projects	Scan Success Rate
Maven	6,984	4,245	61%
Gradle	216	113	52%
Go	575	385	67%
Python	11,071	10,577	96%

15K unique OCI repositories scanned; ~10K successful automatically atm (WIP)

Over 8 Billion LOC scanned

Findings reported to central ticketing system

Test Deployment Snapshot

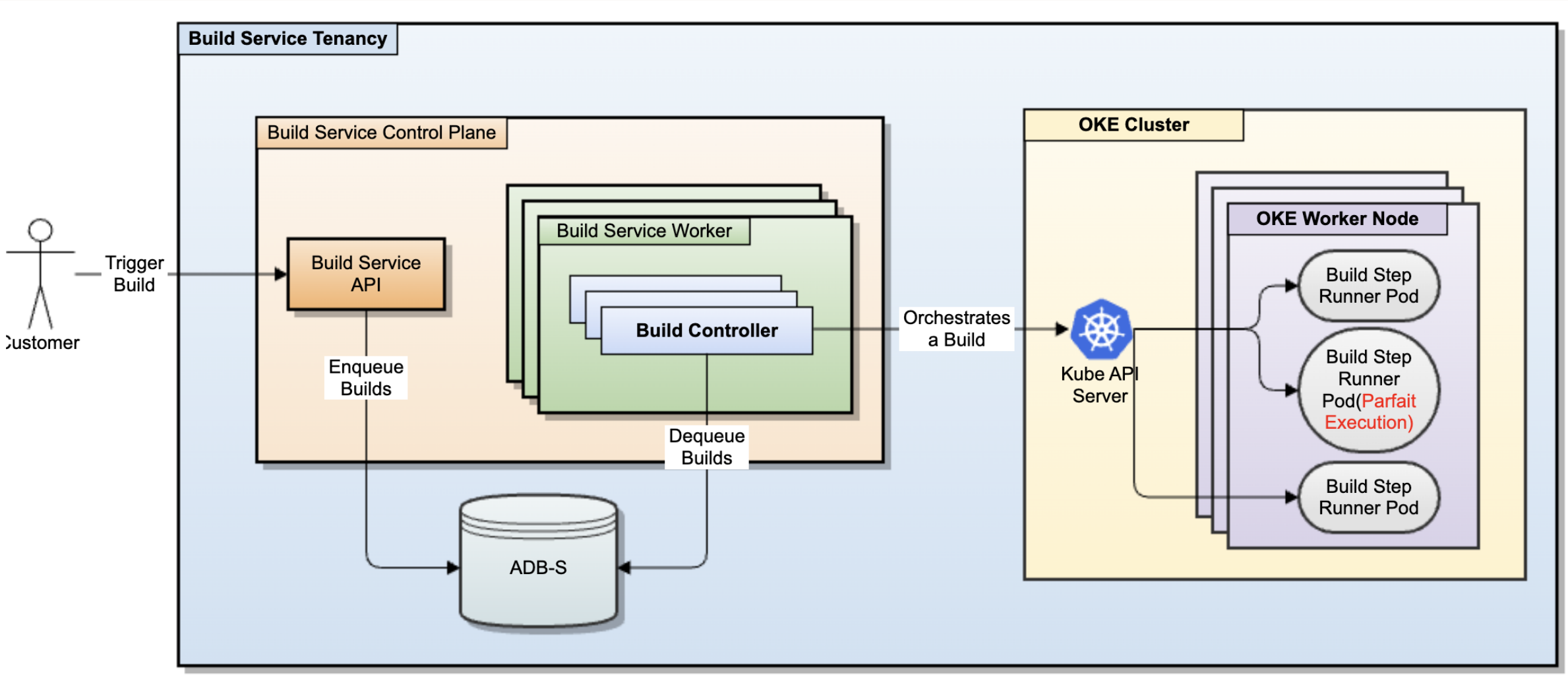
Runtime performance

Language	Lines of Code (M)	Number of Bug Types	Average Analysis Time (secs)	Average Analysis Time (KLOC/min)
Java	1,390	49	44.0	86.2
Go	119	22	49.9	113.0
Python	138	24	31.0	9.5

Data based on recent 7-day time period

Instance Shape: VM.Standard.E3.Flex (AMD)
Default K8S Pod : 5GB Memory and 1 OCPU

Parfait Integration Into OCI Build Service

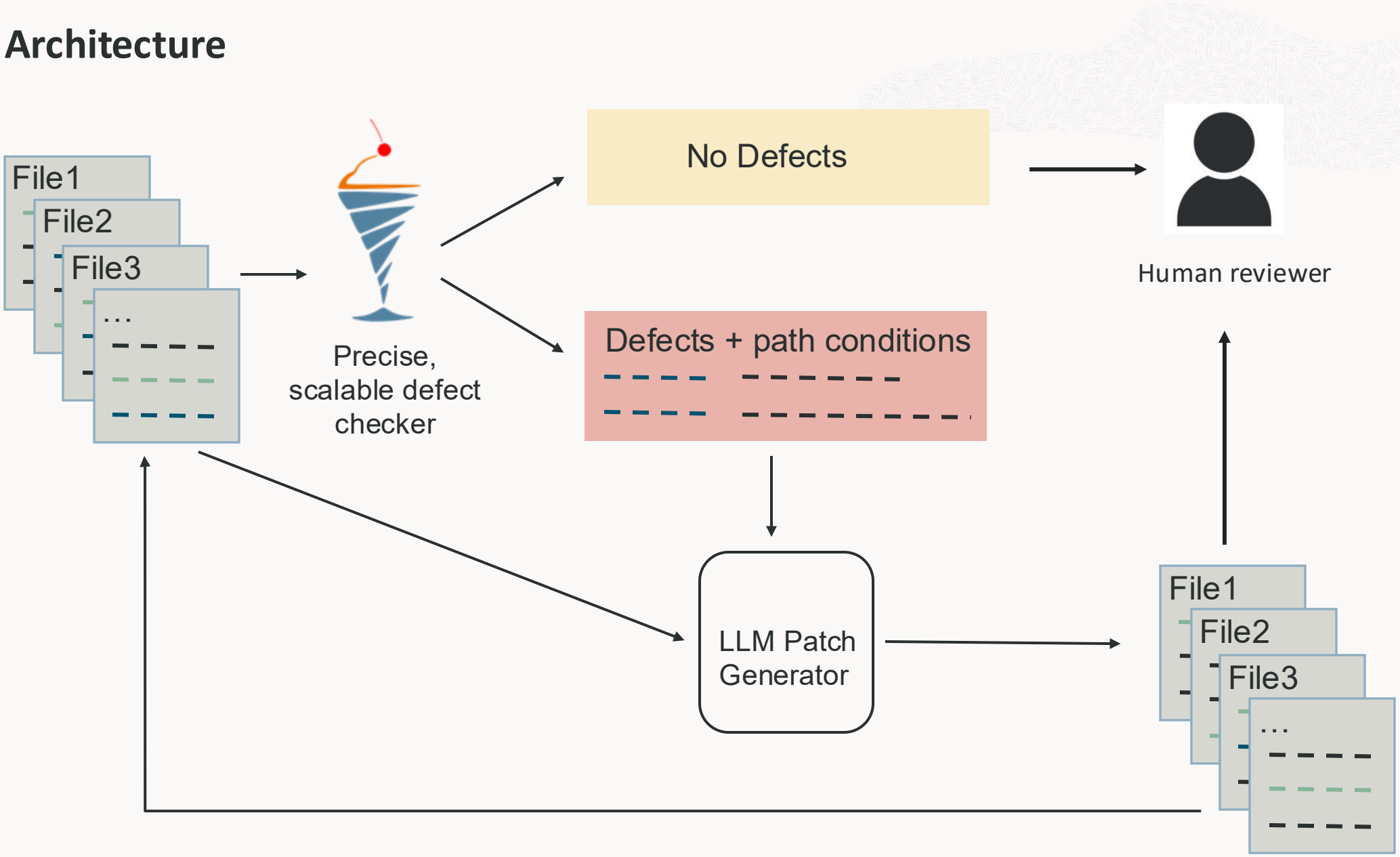


Automation is a requirement for build integration of cloud services

Autonomously Generating Patches for Vulnerabilities

Exploratory work

Architecture



Preliminary Results – AI-assisted Automatic Patch Generation for Buffer Overflows

Type of Benchmark	Average function lines of code	Number of buffer overflows	Average lines altered by generated patch	Vulnerabilities resolved after patch	Functionality uncompromised after patch
Intra-procedural	30	18	7	100%	100%
Inter-procedural, first-party code	100	6	21	100%	67%
Inter-procedural, library	300	4	29	75%	100%

Example – libpng Buffer Overflow

Pre-Patch

```
#include <stdio.h>
...
#include "postdaisy.h"

char htabstops[COLUMNS];   /* horizontal tab stops */
char vtabstops[ROWS];      /* vertical tab stops   */

....

static void inittabs(void)
{
    int i; /* loop index */
    for (i = 0; i < ROWS; i++)
        htabstops[i] = ((i % 8) == 0) ? ON : OFF;
    for (i = 0; i < COLUMNS; i++)
        vtabstops[i] = ((i * ovmi) > BOTTOMMARGIN) ? ON : OFF;
}
/* Clears all horizontal and vertical tab stops. */
static void cleartabs(void)
{
    int i; /* loop index */
    for (i = 0; i < ROWS; i++)
        htabstops[i] = OFF;
    for (i = 0; i < COLUMNS; i++)
        vtabstops[i] = OFF;
}

```

Parfait Report

```
Error: Buffer overrun
   Buffer overflow [buffer-overflow] (CWE 120):
      In array dereference of htabstops[i] with index i
      Array size is 200 bytes, index is 200
        at line 70 of /home/nicholas-chay/t1/testcases/SIMINTERpostdaisy.c in function 'inittabs'.
Error: Buffer overrun
   Buffer overflow [buffer-overflow] (CWE 120):
      In array dereference of htabstops[i] with index i
      Array size is 200 bytes, index is 200
        at line 92 of /home/nicholas-chay/t1/testcases/SIMINTERpostdaisy.c in function 'cleartabs'
```

Patch Example – libpng Buffer Overflow

Pre-Patch

```c
#include <stdio.h>
...
#include "postdaisy.h"

char htabstops[COLUMNS];   /* horizontal tab stops */
char vtabstops[ROWS];      /* vertical tab stops   */

....

static void inittabs(void)
{
    int i; /* loop index */
    for (i = 0; i < ROWS; i++)
        htabstops[i] = ((i % 8) == 0) ? ON : OFF;
    for (i = 0; i < COLUMNS; i++)
        vtabstops[i] = ((i * ovmi) > BOTTOMMARGIN) ? ON : OFF;
}
/* Clears all horizontal and vertical tab stops. */
static void cleartabs(void)
{
    int i; /* loop index */
    for (i = 0; i < ROWS; i++)
        htabstops[i] = OFF;
    for (i = 0; i < COLUMNS; i++)
        vtabstops[i] = OFF;
}

```

Patch-Diff

```
--- testcases_temp/SIMINTERpostdaisy.c  2024-01-10 14:27:01.000000000 +1000
+++ temp_patched/SIMINTERpostdaisy_patch.c  2025-04-11 16:29:14.156776731 +1000
@@ -66,35 +66,36 @@
  */
-    for ( i = 0; i < ROWS; i++ )
+    for ( i = 0; i < sizeof(htabstops) / sizeof(htabstops[0]); i++ )
        htabstops[i] = ((i % 8) == 0) ? ON : OFF;

-    for ( i = 0; i < COLUMNS; i++ )
+    for ( i = 0; i < sizeof(vtabstops) / sizeof(vtabstops[0]); i++ )
        vtabstops[i] = ((i * ovmi) > BOTTOMMARGIN) ? ON : OFF;
 }

 static void  cleartabs(void)
 {
    int  i;   /* loop index */
-    for ( i = 0; i < ROWS; i++ )
+    for ( i = 0; i < sizeof(htabstops) / sizeof(htabstops[0]); i++ )
        htabstops[i] = OFF;
-    for ( i = 0; i < COLUMNS; i++ )
+    for ( i = 0; i < sizeof(vtabstops) / sizeof(vtabstops[0]); i++ )
        vtabstops[i] = OFF;
 }
```

Patch Example – libpng Buffer Overflow

Pre-Patch

```
#include <stdio.h>
...
#include "postdaisy.h"

char htabstops[COLUMNS];   /* horizontal tab stops */
char vtabstops[ROWS];      /* vertical tab stops   */

....

static void inittabs(void)
{
    int i; /* loop index */
    for ( i = 0; i < sizeof(htabstops) / sizeof(htabstops[0]); i++ )
        htabstops[i] = ((i % 8) == 0) ? ON : OFF;
    for ( i = 0; i < sizeof(vtabstops) / sizeof(vtabstops[0]); i++ )
        vtabstops[i] = ((i * ovmi) > BOTTOMMARGIN) ? ON : OFF;
}
/* Clears all horizontal and vertical tab stops. */
static void cleartabs(void)
{
    int i; /* loop index */
    for ( i = 0; i < sizeof(htabstops) / sizeof(htabstops[0]); i++ )
        htabstops[i] = OFF;
    for ( i = 0; i < sizeof(vtabstops) / sizeof(vtabstops[0]); i++ )
        vtabstops[i] = OFF;
}
```

Parfait Report

None

Preliminary Results

- The framework is particularly effective at addressing intraprocedural vulnerabilities.
- Patches rely on the key contextual information given by Oracle Parfait (buffer bounds, path conditions, etc.).
- Larger benchmarks require context reduction techniques (natural language summary was used in our case).
- The framework works well with other types of vulnerabilities.

Lessons Learned

Subhead goes here.

Analyses need to be **precise**, **scalable** and **incremental** in order to be useful to developers and practical for DevSecOps and CI/CD integration.

Analyses need to **explain** why the tool reports a bug at a given line; i.e., provide a trace/witness.

Automatic integration into existing CI build pipelines is essential.

Aiding developers to **fix** defects will contribute to improve security and developer productivity.

Oracle Parfait
Precise, scalable and incremental static analysis for today’s giga-scale code

Let’s Connect

drcristinacifuentes

@criscifuentes

http://labs.oracle.com/locations/australia


ORACLE