ORACLE®

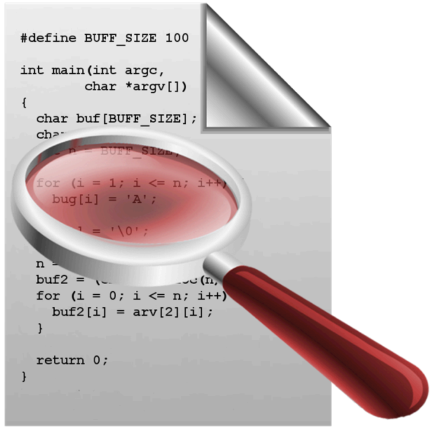
Parfait Lessons Learnt

Cristina Cifuentes, Nathan Keynes, Manuel Valdiviezo*, John Gough, Diane Corney
Oracle Labs Australia
* Oracle Parfait

17 July 2016

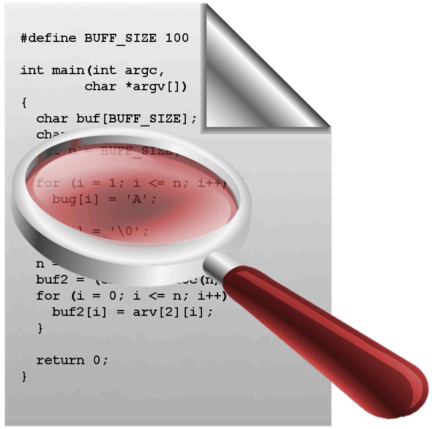
The Parfait Project

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.



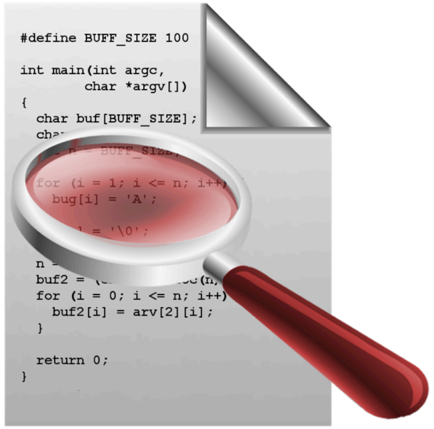
```
#define BUFF_SIZE 100

int main(int argc,
        char *argv[])
{
  char buf[BUFF_SIZE];
  cha
    n = BUFF_SIZE;
  for (i = 1; i <= n; i++)
    bug[i] = 'A';
      = '\0';

  n
  buf2 =          (n,
  for (i = 0; i <= n; i++)
    buf2[i] = arv[2][i];
  }

  return 0;
}
```

Parfait research goal:
To develop a static code analysis tool that is precise yet scalable to millions of lines of C/C++ code



```
#define BUFF_SIZE 100

int main(int argc,
        char *argv[])
{
  char buf[BUFF_SIZE];
  cha
  t n = BUFF_SIZE;

  for (i = 1; i <= n; i++)
    bug[i] = 'A';

  ] = '\0';

  n =
  buf2 = (c        c(n,
  for (i = 0; i <= n; i++)
    buf2[i] = arv[2][i];
  }

  return 0;
}
```

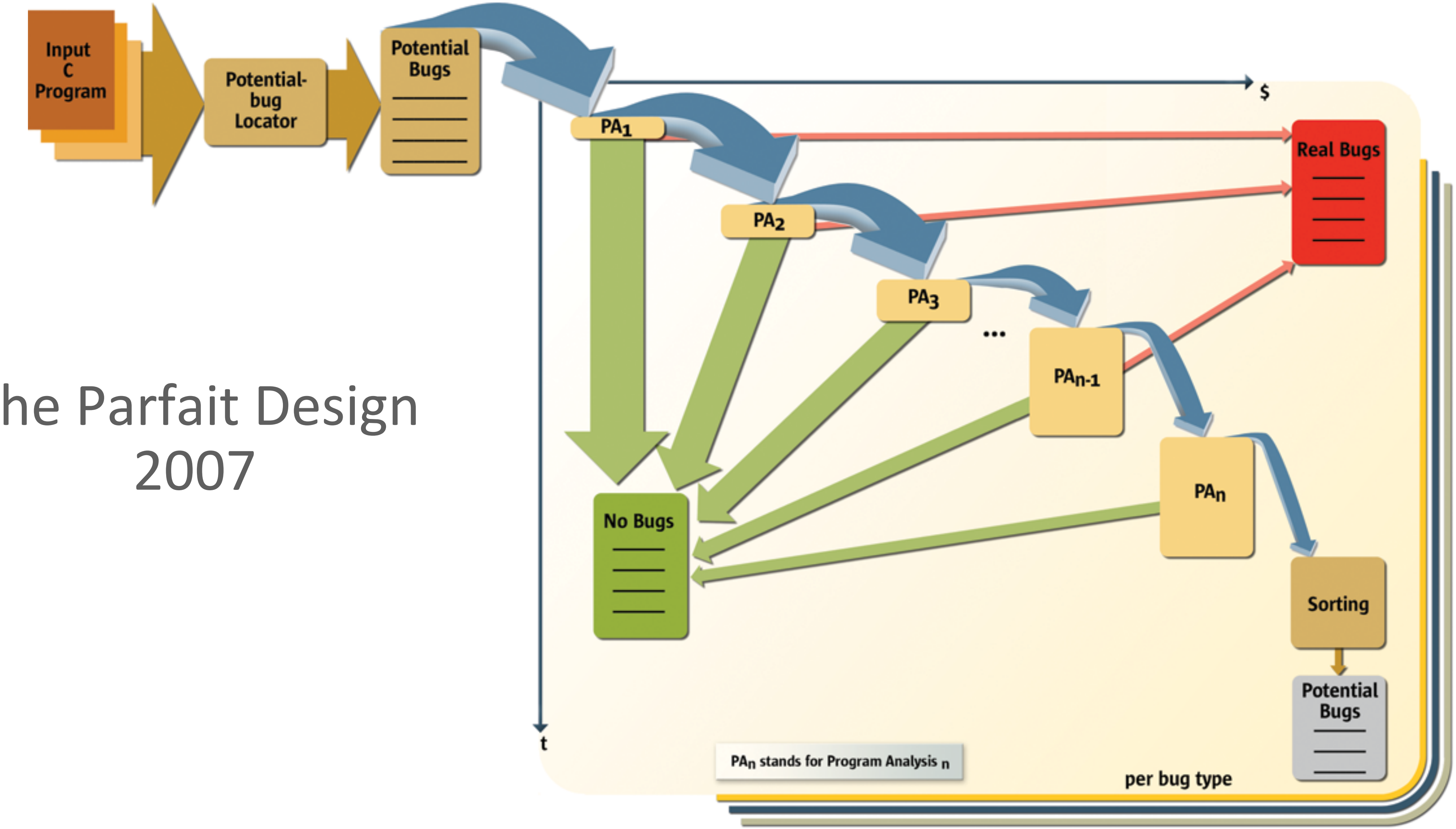
Observation 1: some bugs are easy to find, others are hard to find



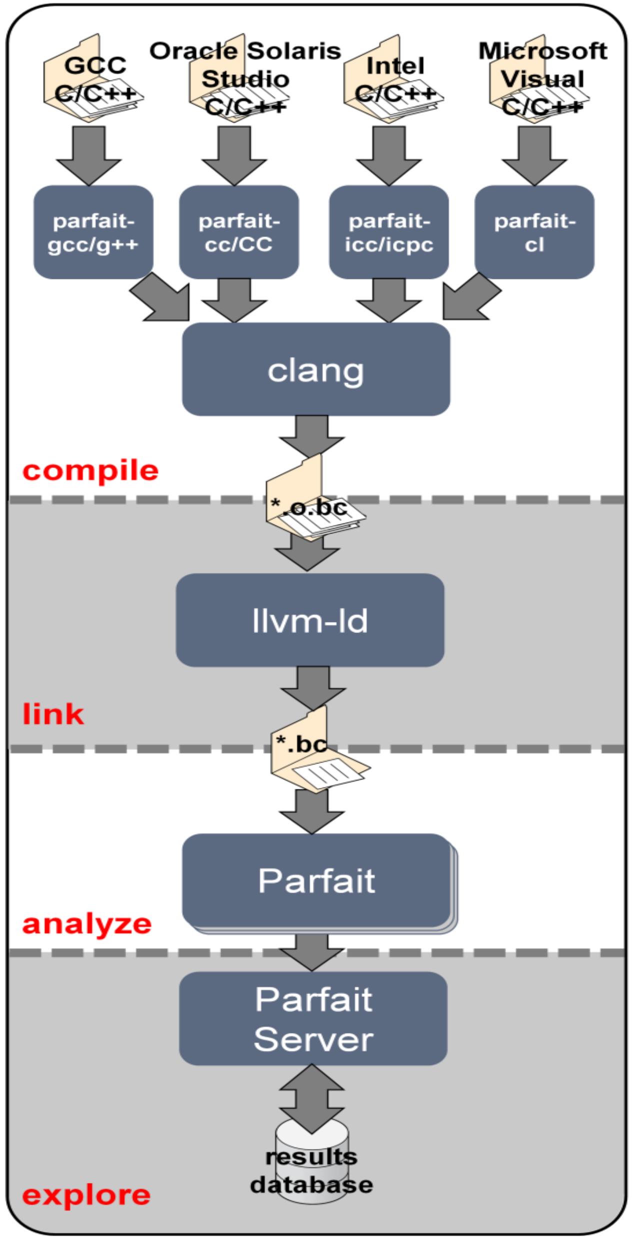
```
#define BUFF_SIZE 100

int main(int argc,
        char *argv[])
{
  char buf[BUFF_SIZE];
  cha
    n = BUFF_SIZE;

  for (i = 1; i <= n; i++)
    bug[i] = 'A';

      ] = '\0';

  n =
  buf2 = (c          c(n,
  for (i = 0; i <= n; i++)
    buf2[i] = arv[2][i];
  }

  return 0;
}
```

Observation 2: cheap program analyses can find easy bugs, expensive program analyses can find complex bugs

The Parfait Design 2007



Built on Top of LLVM



Buffer Overflow Results Over Open Source OS Kernels

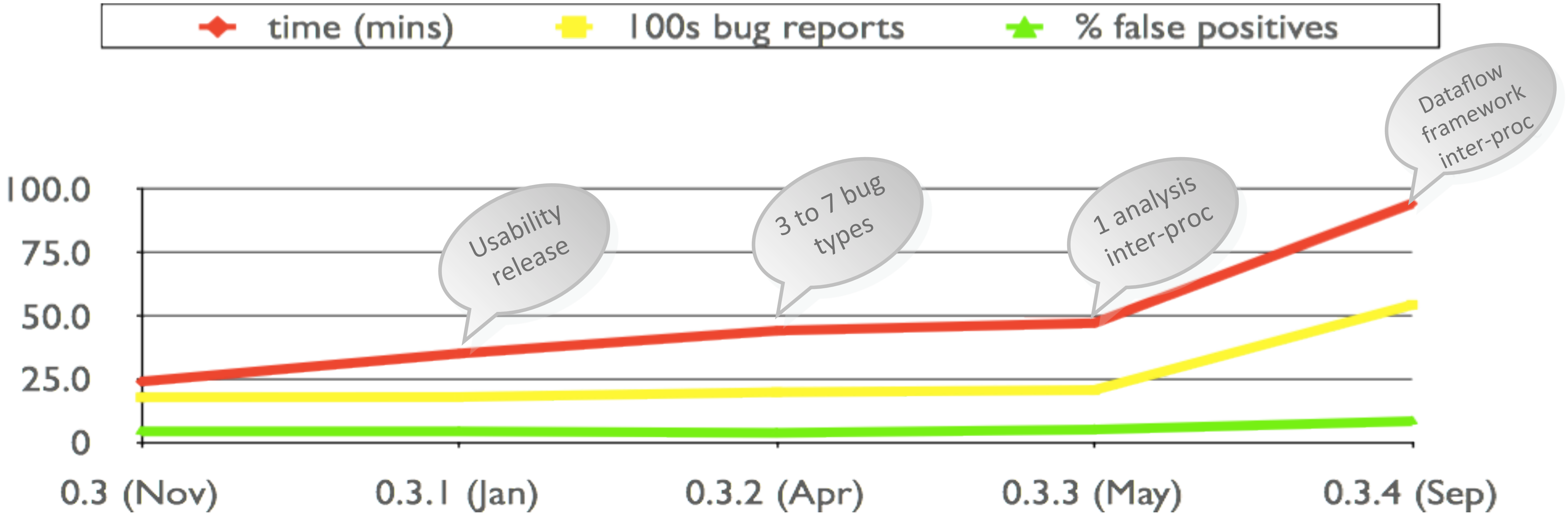
June 2009

Kernel	Time (min)	Part	LOC	Buffer overrun	Bug density	Status
OpenSolaris UTS b105	5	Core	2.1M	15	0.0069	Being fixed
		Device drivers	1.2M	67	0.054	Being fixed
Linux 2.6.29*	13	Core	1.6M	12	0.0073	Fixed
		Device drivers	4.1M	85	0.020	Submitted
OpenBSD 4.4	2	Core	0.5M	3	0.0060	Fixed
		Device drivers	0.8M	26	0.029	Fixed

* Linux has the benefit of two separate scans already made by Coverity over their kernel code

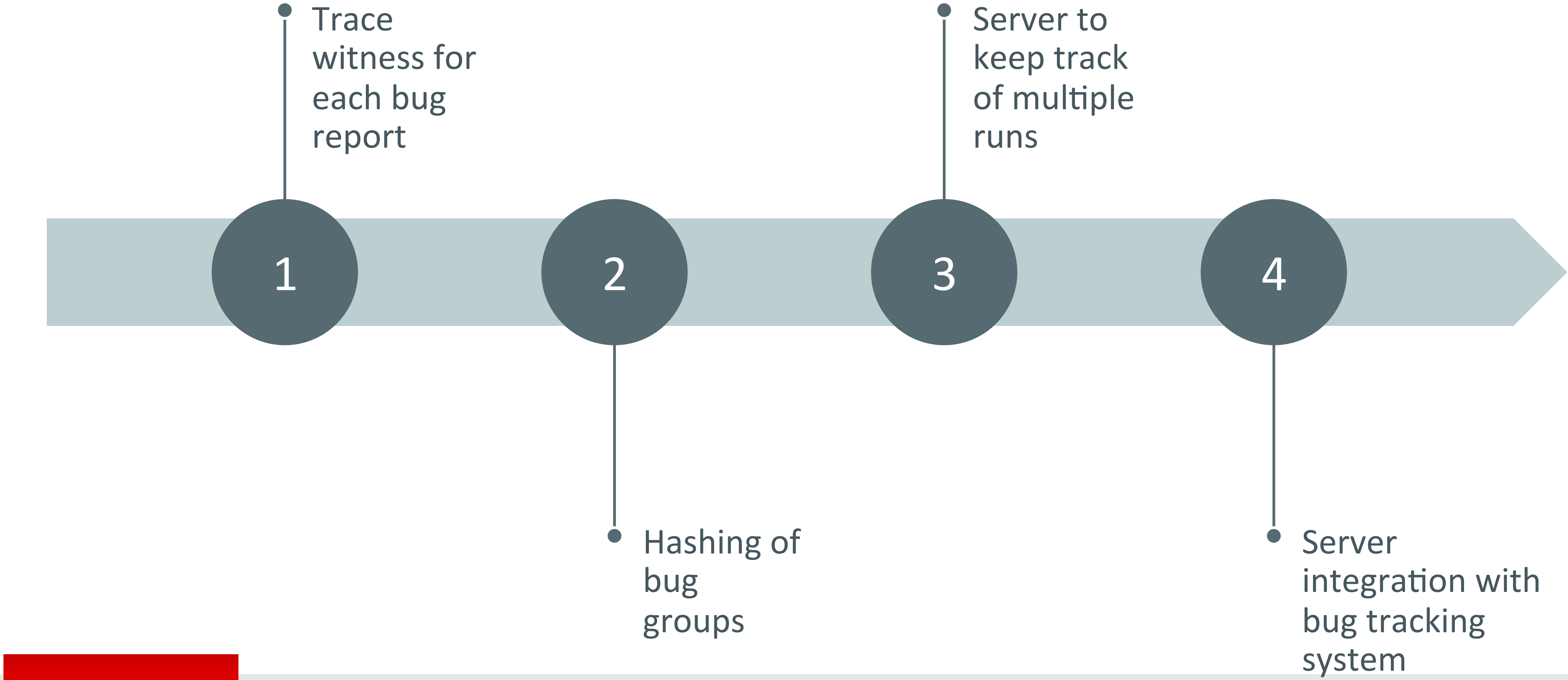
Common C Bugs Results Over OpenSolaris ON Code

November 2009 – September 2010



9.5 MLOC | 10.3 MLOC | 10.4 MLOC | 10.4 MLOC | 10.4 MLOC

The Bells and Whistles to Enable Tech Transfer



1

Trace witness for each bug report

2

Hashing of bug groups

3

Server to keep track of multiple runs

4

Server integration with bug tracking system

The Transfer

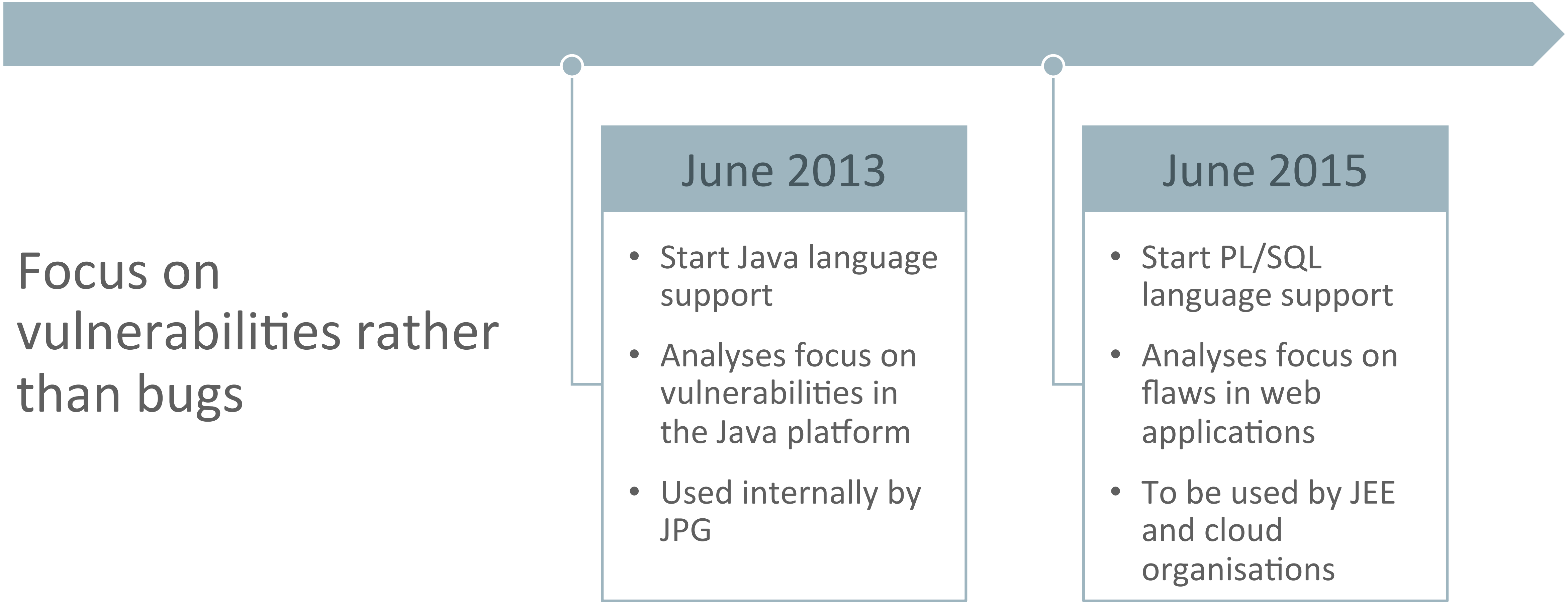
June 2012

- Parfait becomes an internal Oracle product
- Used internally by RDBMS, Solaris, OEL, TimesTen, …
- Memory usage: 10x-20x size of .bc

Used by thousands of developers within Oracle on a daily basis

New Language and Analysis Support

Focus on vulnerabilities rather than bugs



June 2013

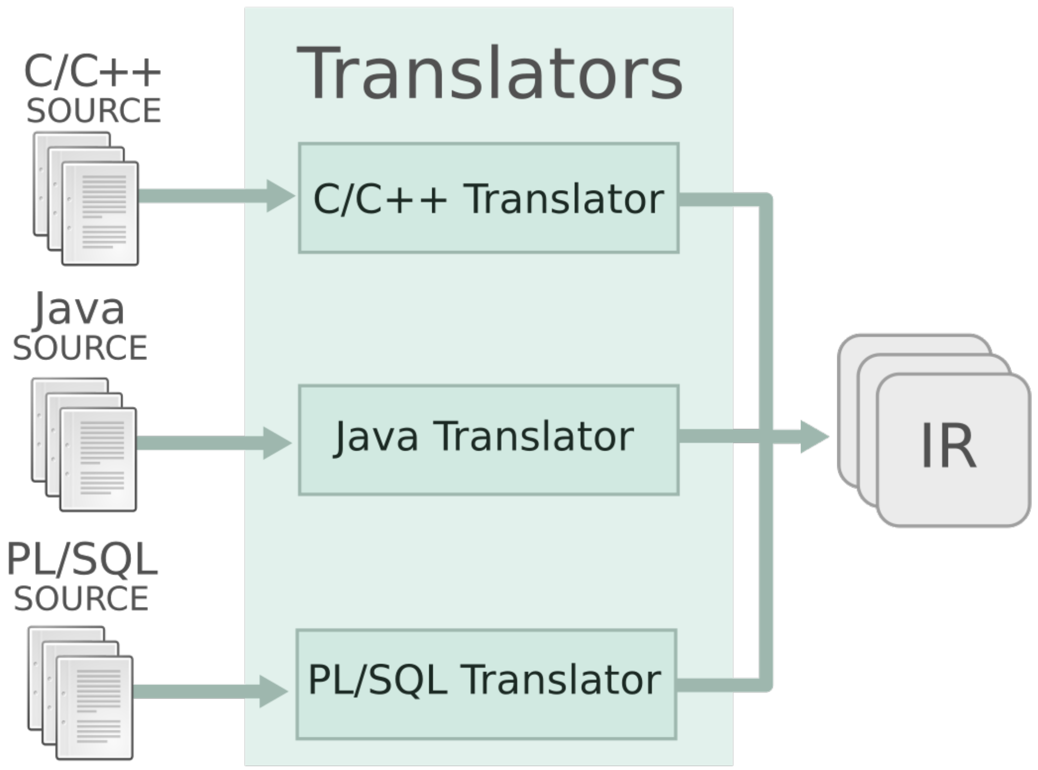
- Start Java language support
- Analyses focus on vulnerabilities in the Java platform
- Used internally by JPG

June 2015

- Start PL/SQL language support
- Analyses focus on flaws in web applications
- To be used by JEE and cloud organisations

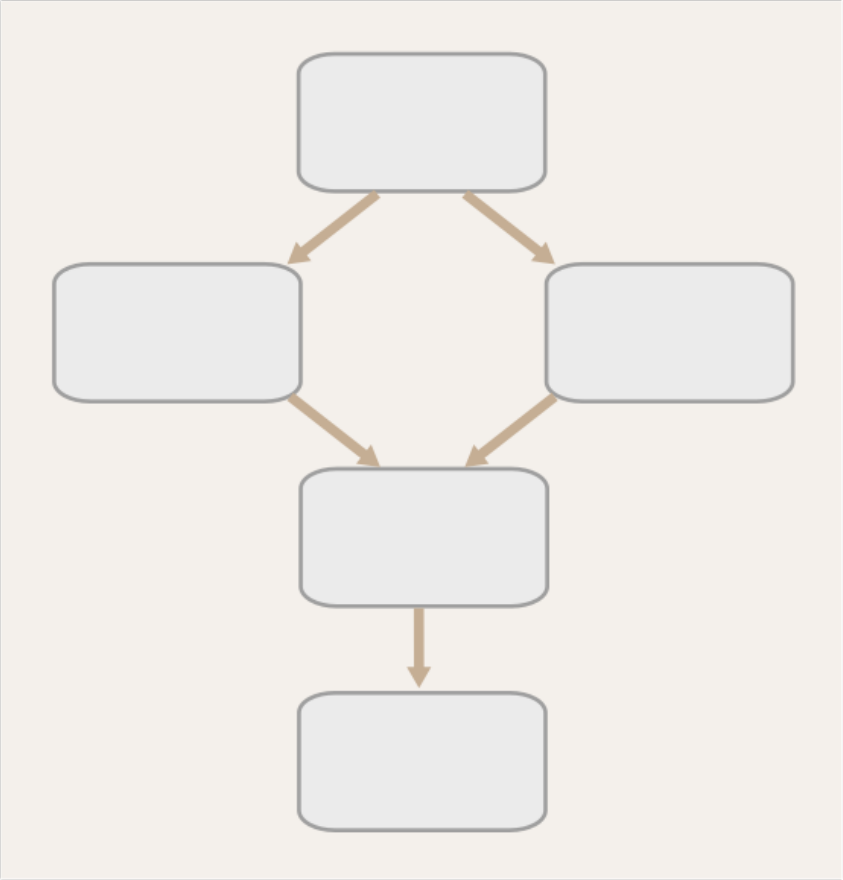
What Worked Well

Frontend



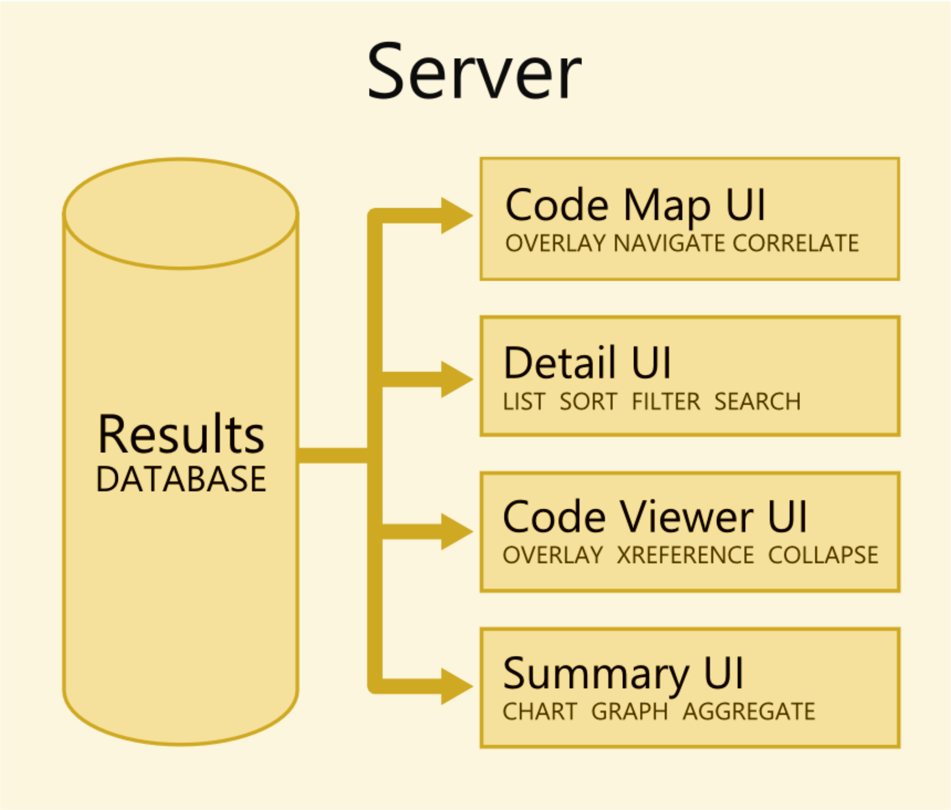
- “Loosening up” Clang
  - To support multiple C compilers and old versions of C
- Translation of language for **analysis**
  - Java, PL/SQL
- Multi-language support enables
  - JNI analyses
  - PL/SQL to C and PL/SQL to Java analyses

Analysis



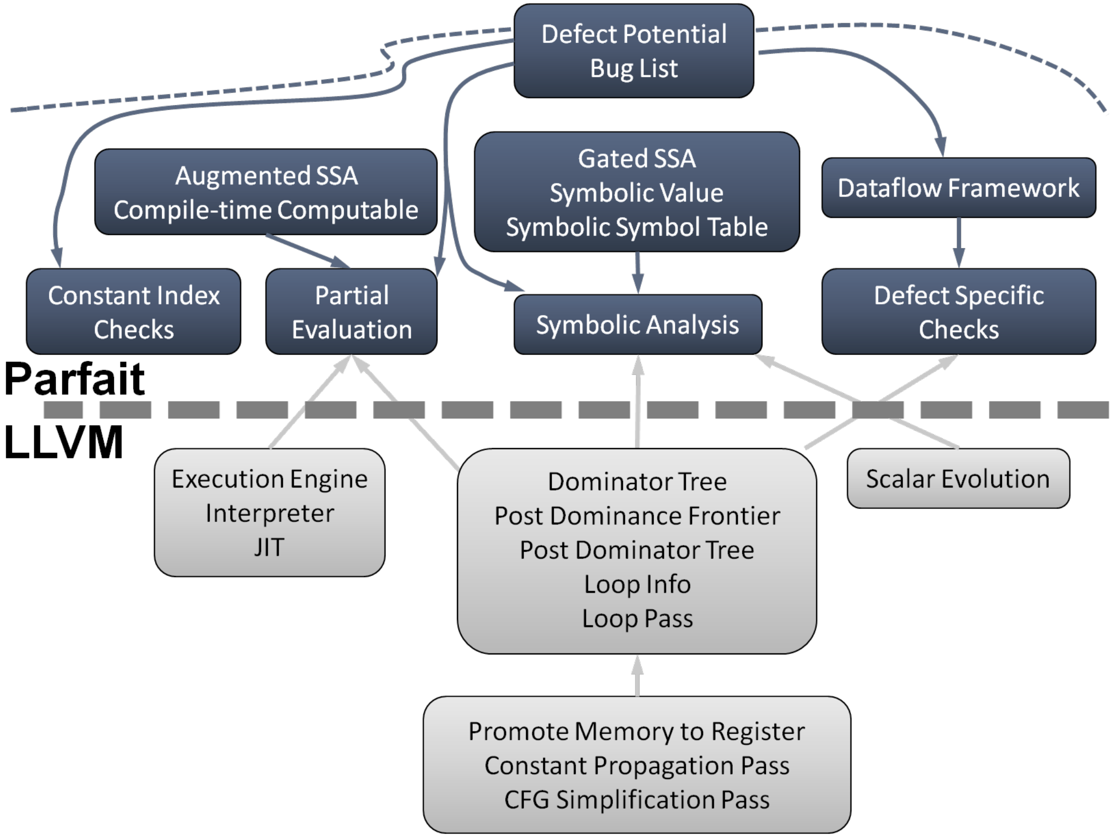
- Demand-driven analysis scales well
  - Combined with extensive caching
  - Function summaries help
- Backwards reusable frameworks
  - Dataflow
  - Symbolic analysis
- Having abstractions align well with the code under analysis
  - E.g., bit-flag operations

Backend



- Usability
  - Server to keep track of multiple runs
  - Mechanism (hashes) to
    - compare results from different runs, and
    - group bugs
  - Trace witness for each bug report

Infrastructure

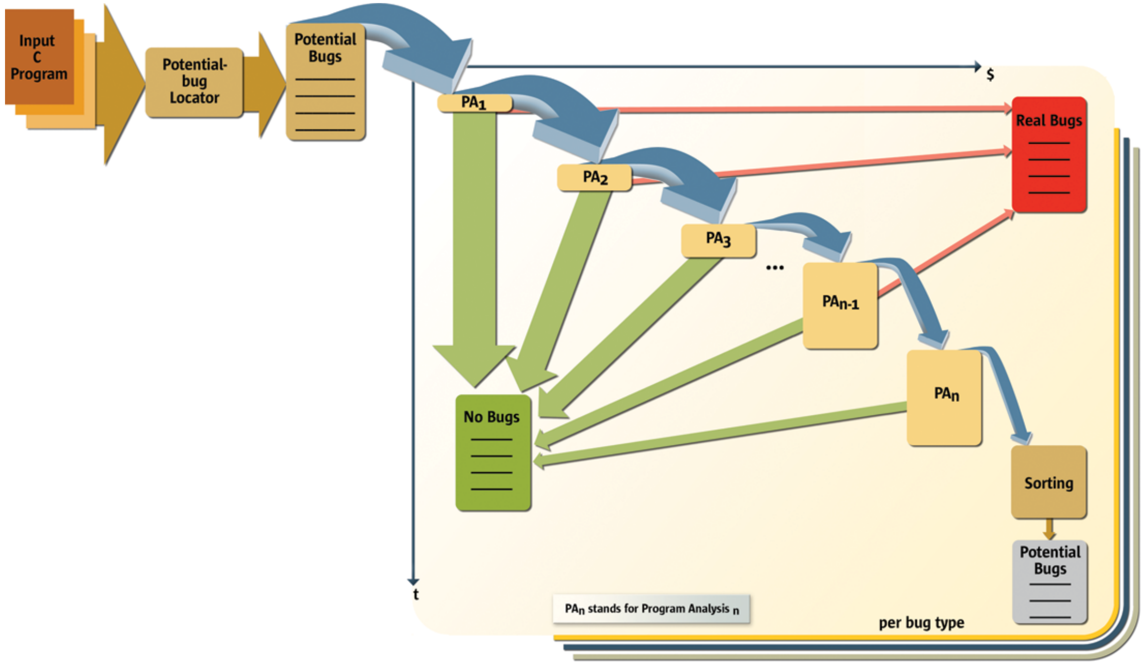


Defect Potential Bug List
Augmented SSA Compile-time Computable
Gated SSA Symbolic Value Symbolic Symbol Table
Dataflow Framework
Constant Index Checks
Partial Evaluation
Symbolic Analysis
Defect Specific Checks
Parfait
LLVM
Execution Engine Interpreter JIT
Dominator Tree Post Dominance Frontier Post Dominator Tree Loop Info Loop Pass
Scalar Evolution
Promote Memory to Register Constant Propagation Pass CFG Simplification Pass

- LLVM works well as the underlying infrastructure
  - IR
  - Analysis support

The In Between

Original Layered Analysis Design



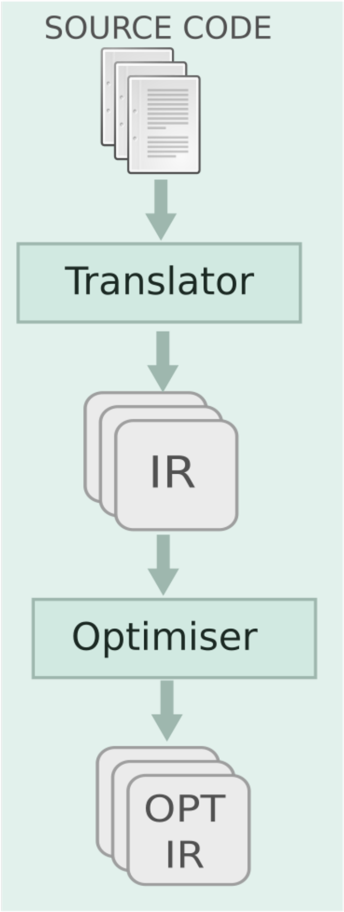
- Layered analysis works but not used as originally planned
  - Most analyses have multiple exit points
  - Promotions of one bug type to another

Parfait Infrastructure

- Replicated work due to independent development of the analyses
- Infrastructure needs to be upgraded as larger codebases are supported
- Bug hashes essential but hard to keep consistent

What Didn’t Work Well

IR

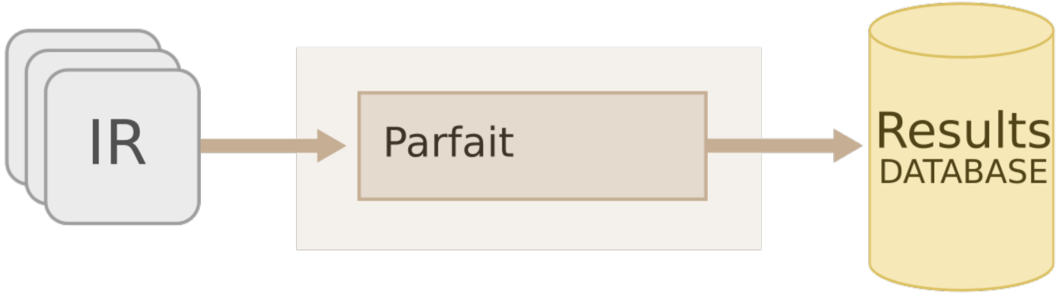


- Use of optimisations to simplify IR
  - Removed in favour of useful bug reports
- Requires data from the AST
  - Needed for useful bug reports
- Cannot represent dynamic features of languages

LLVM Infrastructure

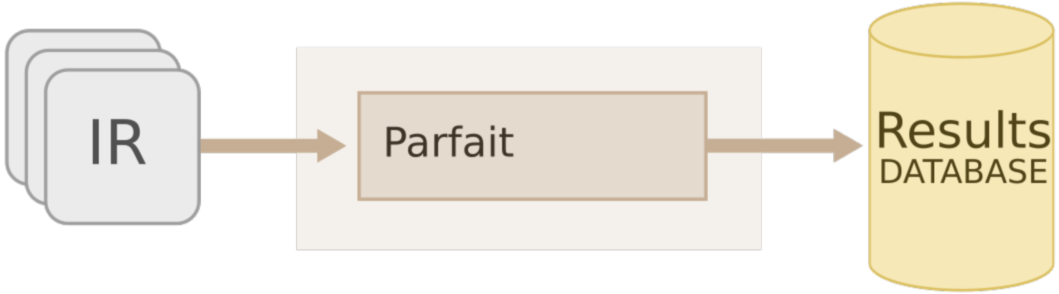
- llvm-ld doesn’t scale well
- .bc format is not indexable
  - Now using file format that supports random access
- Support for other C compilers not of interest to the Clang community

Analysis



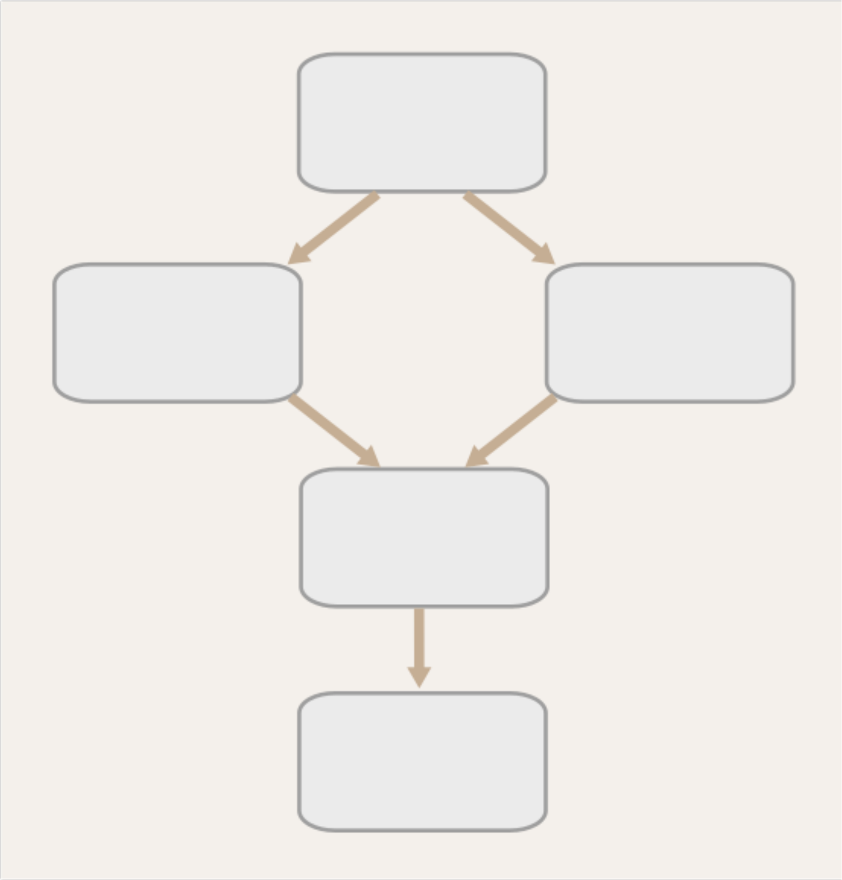
- Hard to implement top-down analysis as libraries and architecture designed to work primarily bottom-up
- Technical debt exposed when improving analysis code coverage
- Incomplete call graph due to function pointers
- Not having a framework to manage the bottom-up analyses

Granularity of Analysis



- Analysing one LLVM module at a time doesn’t work for large monolithic codebases
  - E.g., 200GB RAM to process one .bc file
- Incremental analysis at the LLVM module doesn’t work for everyone
  - Some teams want incremental at subcomponent levels
- Reuse of results of analysis of dynamic libraries linked into multiple binaries is needed

Usability and Development Organisation’s Workflow



- “Expensive” analyses are not deployed in production
  - If runtime is larger than allocated nightly integration window
- Specification language for new bug types is not expressive enough
  - E.g., can’t express TOCTOU issues
  - Other issues with how an organisation deploys such new bug types

Main Takeaways

Parfait for C/C++, Java and PL/SQL – Main Takeaways

Worked Well

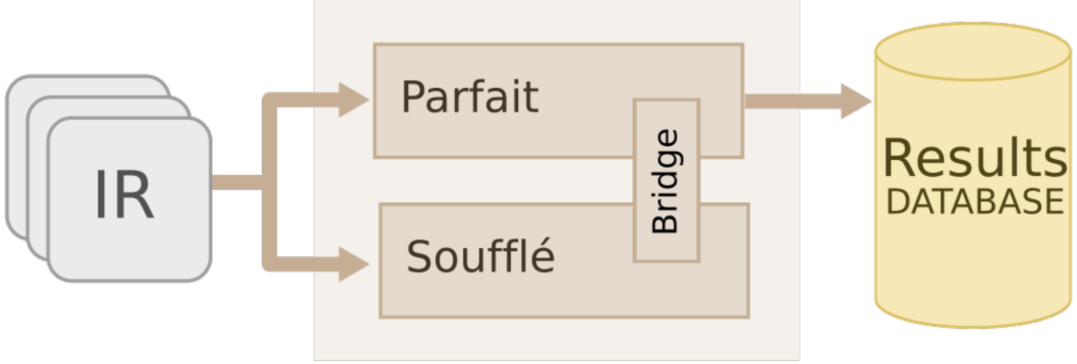
- Scalability through demand-driven analyses + caching
- Precision through unsoundness + heuristics
- Usability through user and organisational deployment experience

Needs More Work

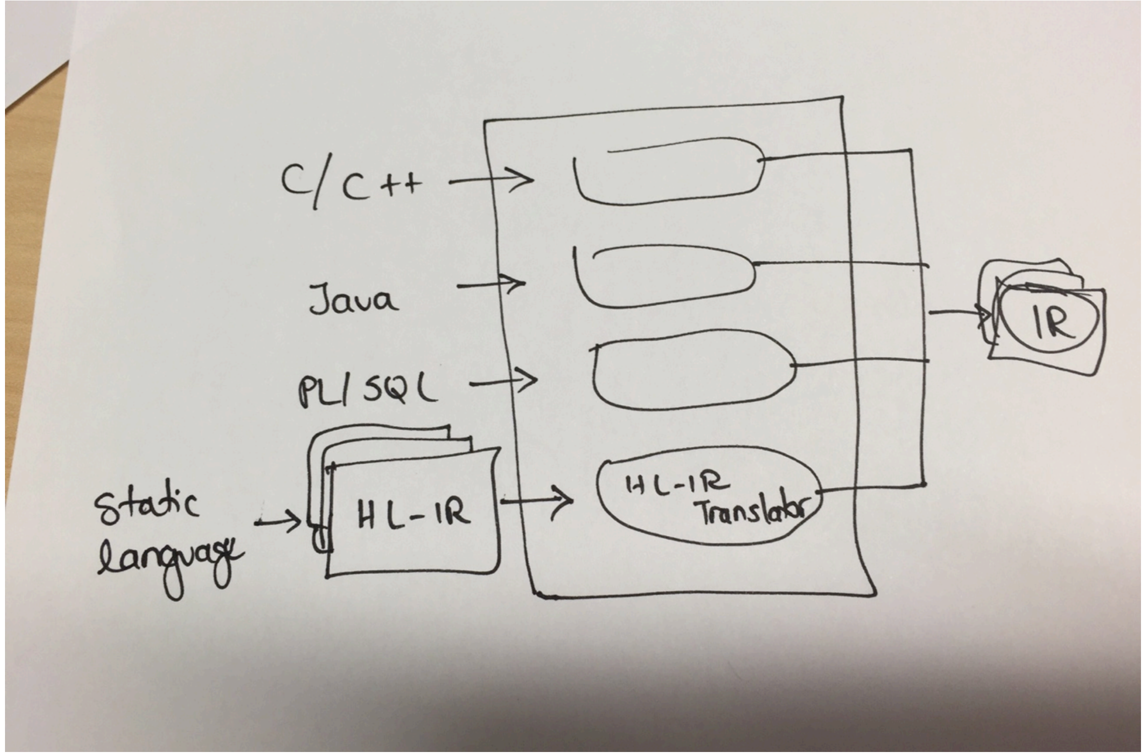
- Extensibility only possible through internal team
  - New analyses
    - Provide interface to Datalog analyses?
  - New languages
    - Provide API/interface to high-level IR?
- Infrastructure changes become challenging as time goes by

Extensibility – Possible Solutions

Provide interface to Datalog



Provide interface to other languages



Many People Have Worked on Parfait Over the Years

- Cristina Cifuentes
- Bernhard Scholz
- Nathan Keynes
- Lian Li
- Chenyi Zhang
- Erica Mealy
- Michael Mounteney
- Simon Long
- Nathan Hawes
- Mike Van Emmerik
- Christian Hoermann
- Manuel Valdiviezo
- Andrew Browne
- Adam Heron
- Jimmy Ti
- Jacob Zimmermann
- Andrew Craik
- Brad Moody
- Ben Barham
- Douglas Teoh
- Duc Hoai Nguyen
- Edward Evans
- Dominic Ferreira
- Ijaz Faiz
- Ben Dean
- Ben Jones
- Daniel Dawson
- Adam Heron
- Kostyantyn Vorobyov
- Diane Corney
- John Gough
- Daniel Wainwright
- Nicholas Allen
- Brian Modra
- Matthew Johnson
- Paddy Krishnan
- Tomas Kotal
- Vince Chiang
- Lin Gao
- Richard Marks
- Minhtri Pham
- François Gauthier
- Alexander Jordan?
- Vladimir Silchanka
- Tom King
- Ramon Millsteed

Parfait: scalable and precise bug detection for static languages

Integrated Cloud

Applications & Platform Services

ORACLE®