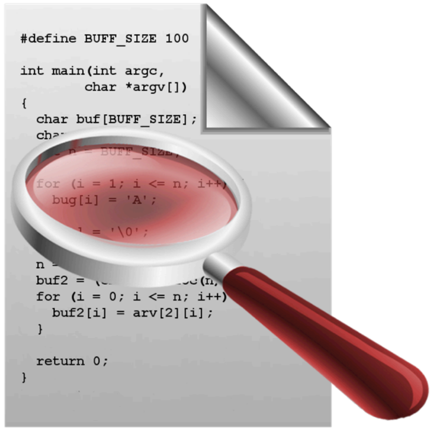
Parfait Lessons Learnt

Cristina Cifuentes, Nathan Keynes, Manuel Valdiviezo*, John Gough, Diane Corney
Oracle Labs Australia
* Oracle Parfait
17 July 2016

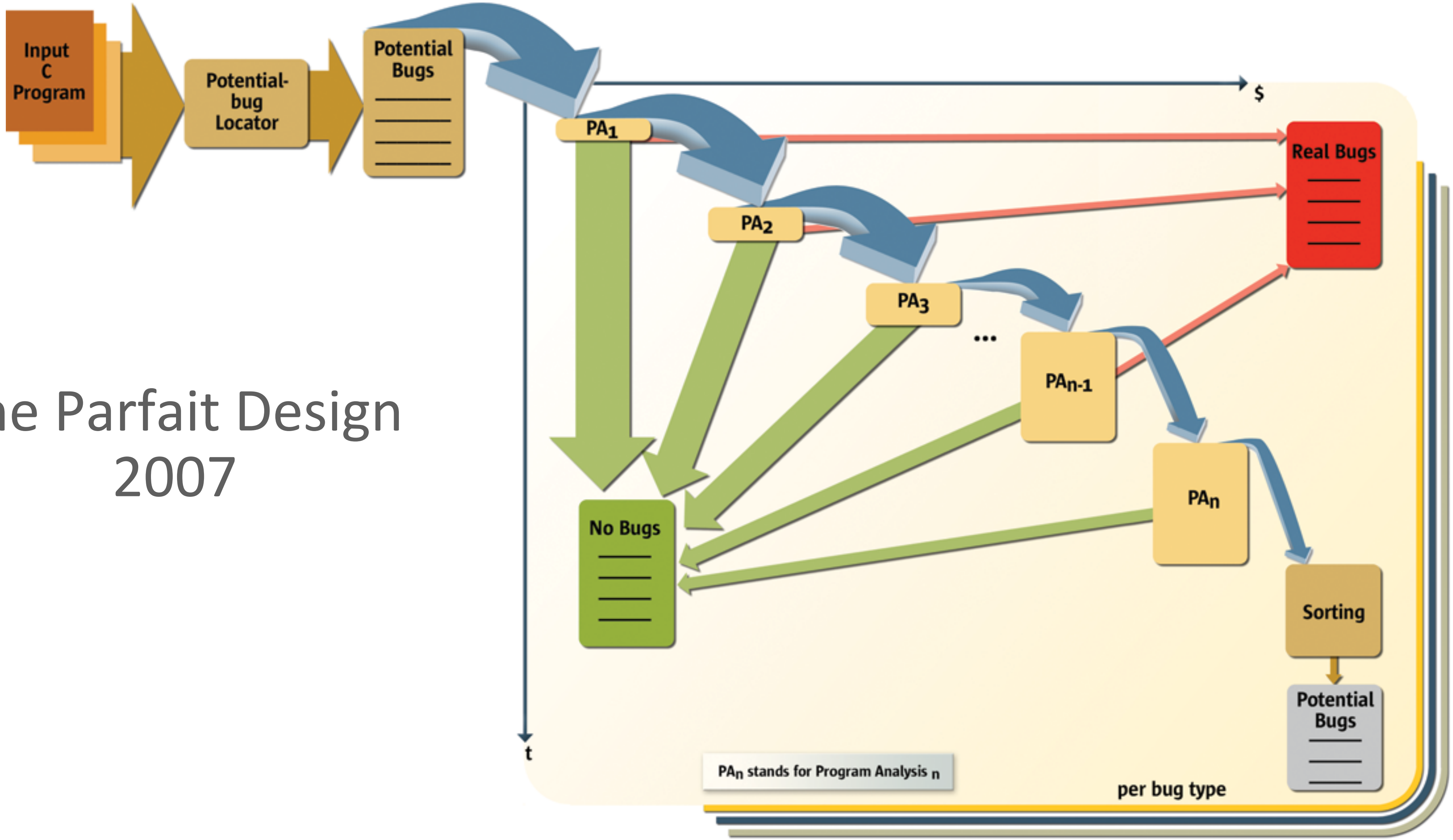
The Parfait Project

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.

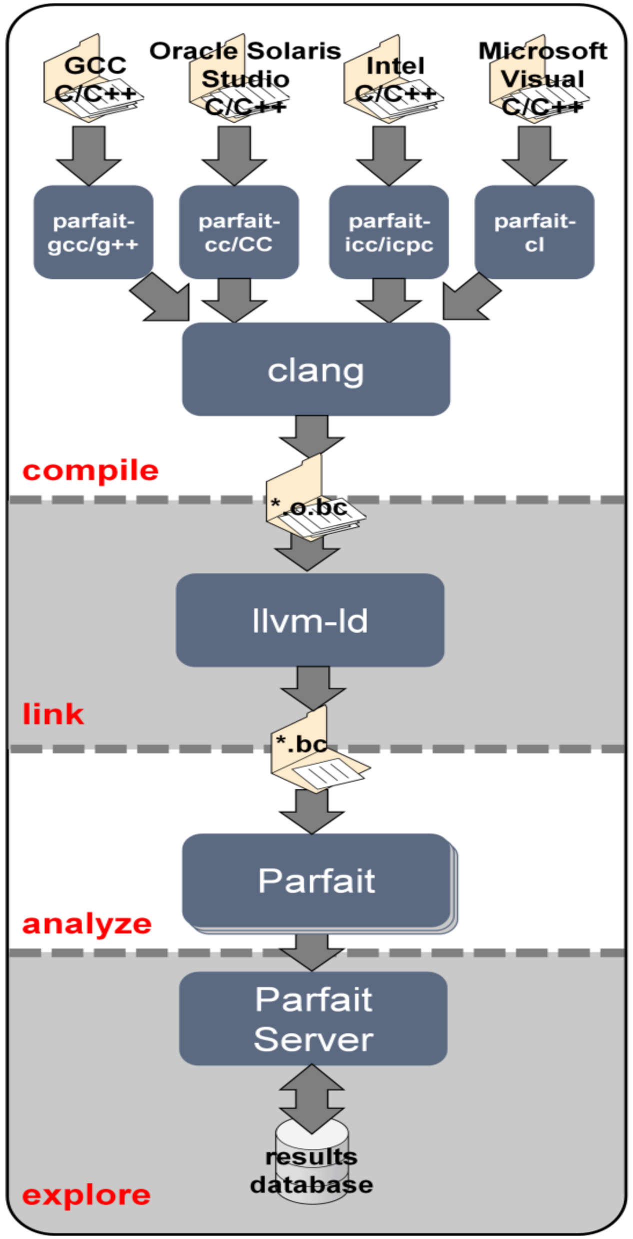


To develop a static code analysis tool that is precise (>= 90% true positives) yet scalable to millions of lines of C/C++ code in a nightly run

The Parfait Design
2007



Built on Top of LLVM



Snapshot of Parfait Results

June 2009	Kernel	Part	LOC	Buffer overrun	Bug density	Status	Time (min)
	OpenSolaris UTS b105	Core	2.1M	15	0.0069	Being fixed	5
		Device drivers	1.2M	67	0.054	Being fixed	

September 2010	ON	Part	LOC	# bug types	Memory	Time (min)
	OpenSolaris ON	All	10.4M	9	10-20x .bc	90

ORACLE

The Bells and Whistles to Enable Tech Transfer

1. Trace witness for each bug report
2. Unique bug identity via hashes
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, …

Used by thousands of developers within Oracle on a daily basis

ORACLE

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 Java Product Group

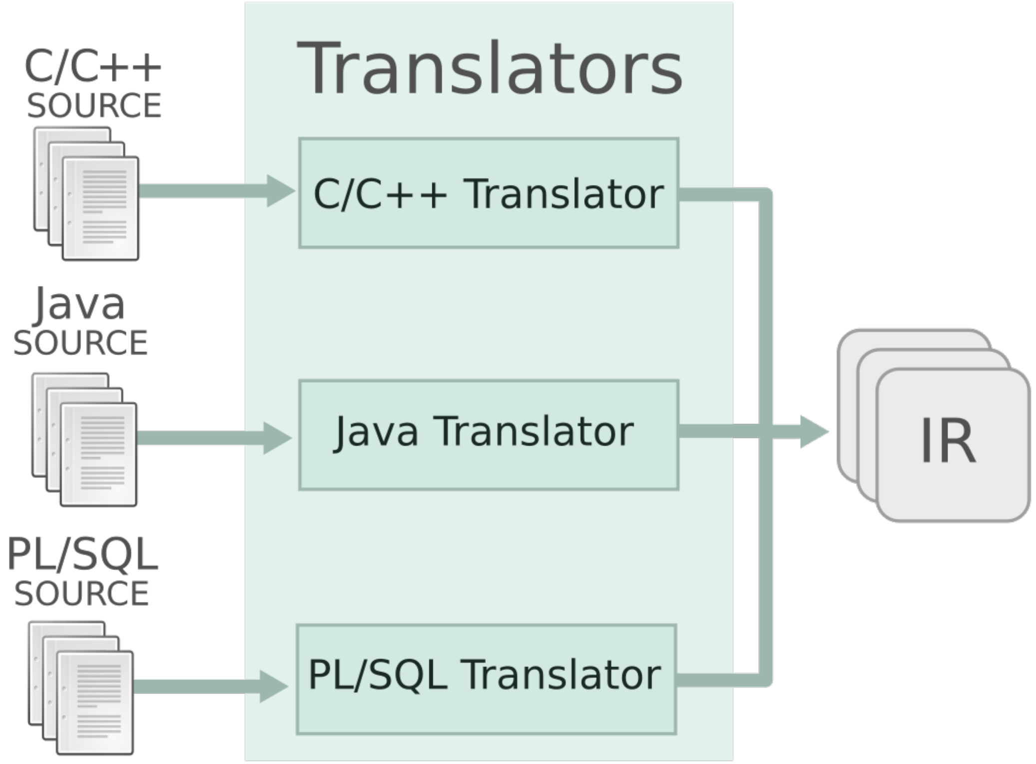
June 2015

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

ORACLE

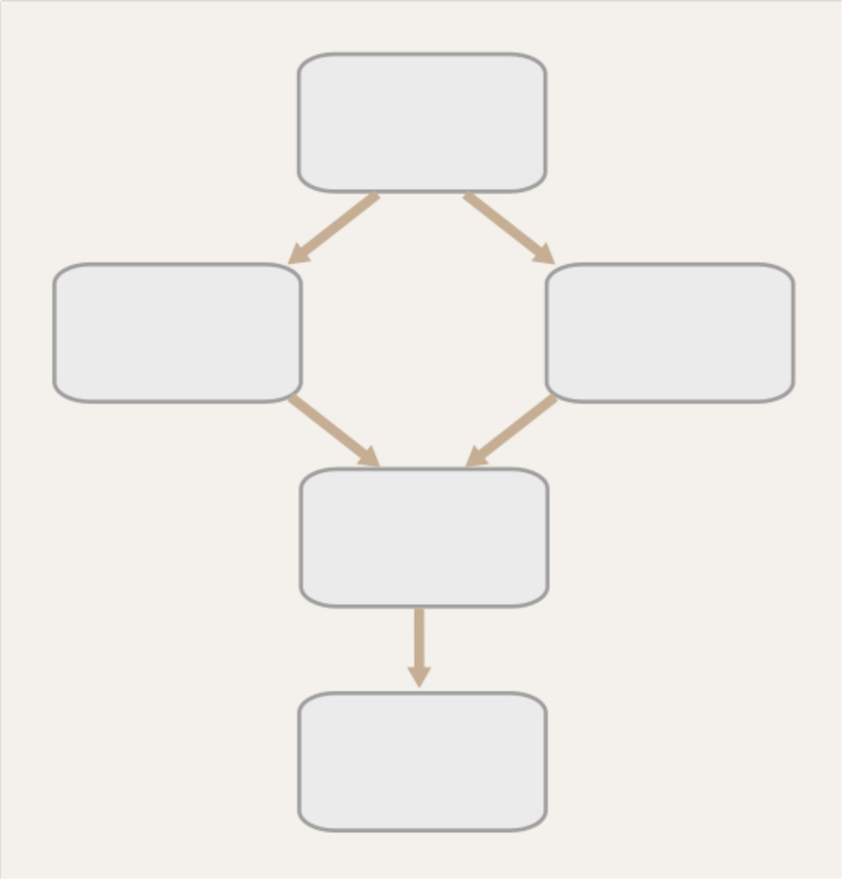
What Worked Well

Frontend



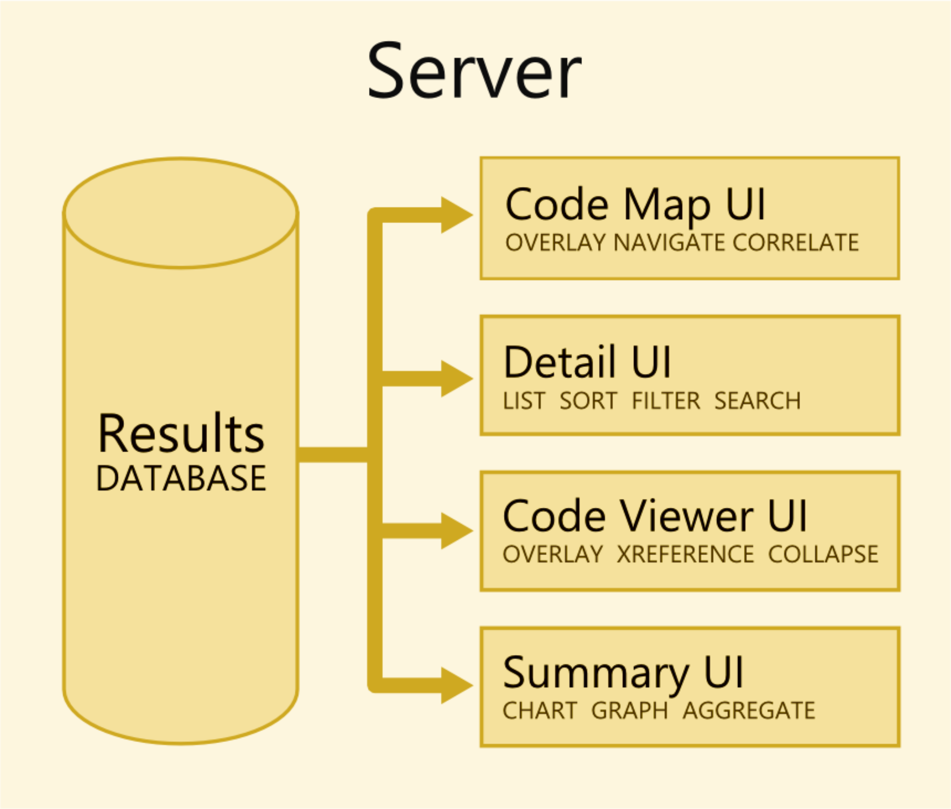
- “Loosening up” Clang
  - To support multiple C compilers and old versions of C
- Translation of language for **analysis**
  - Java, PL/SQL
- Multi-language support and reuse of analyses

Analysis



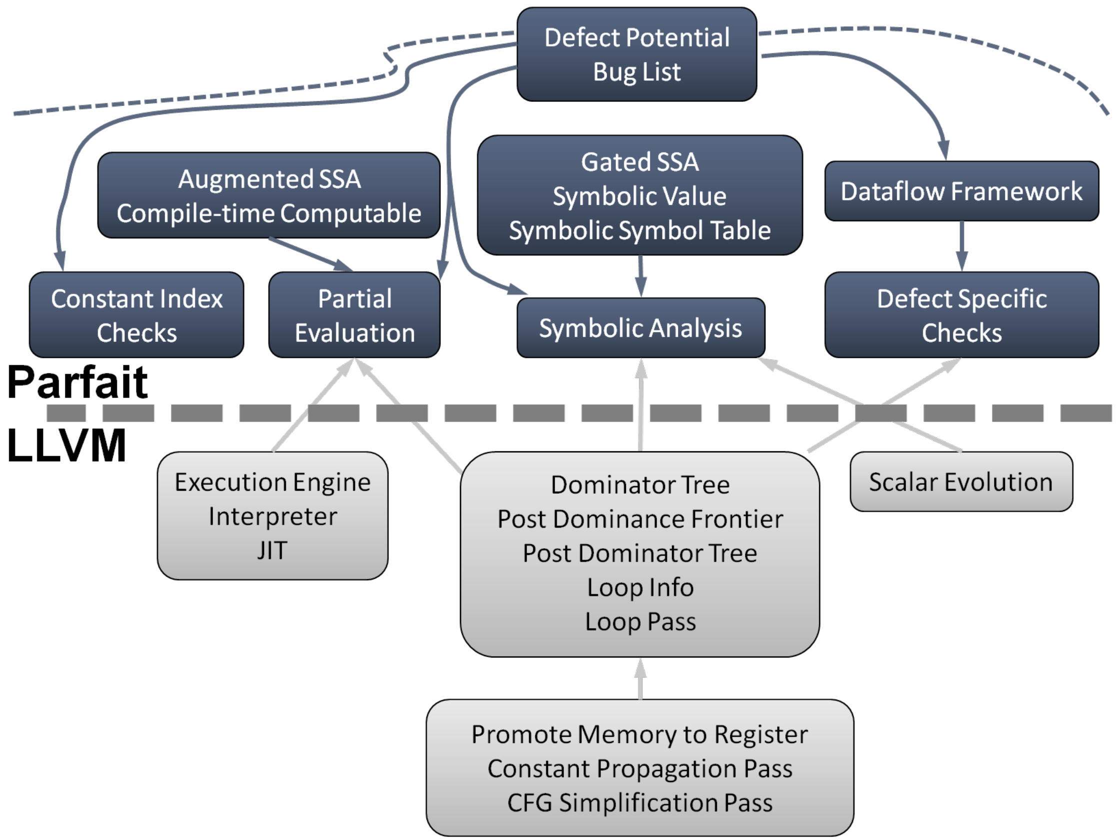
- Demand-driven analysis scales well
  - Combined with extensive caching
  - Function summaries help
- Backwards reusable frameworks
  - Dataflow
  - Symbolic analysis
- Having abstractions align well with the code under analysis
  - E.g., bit-flag operations

Presentation Framework



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

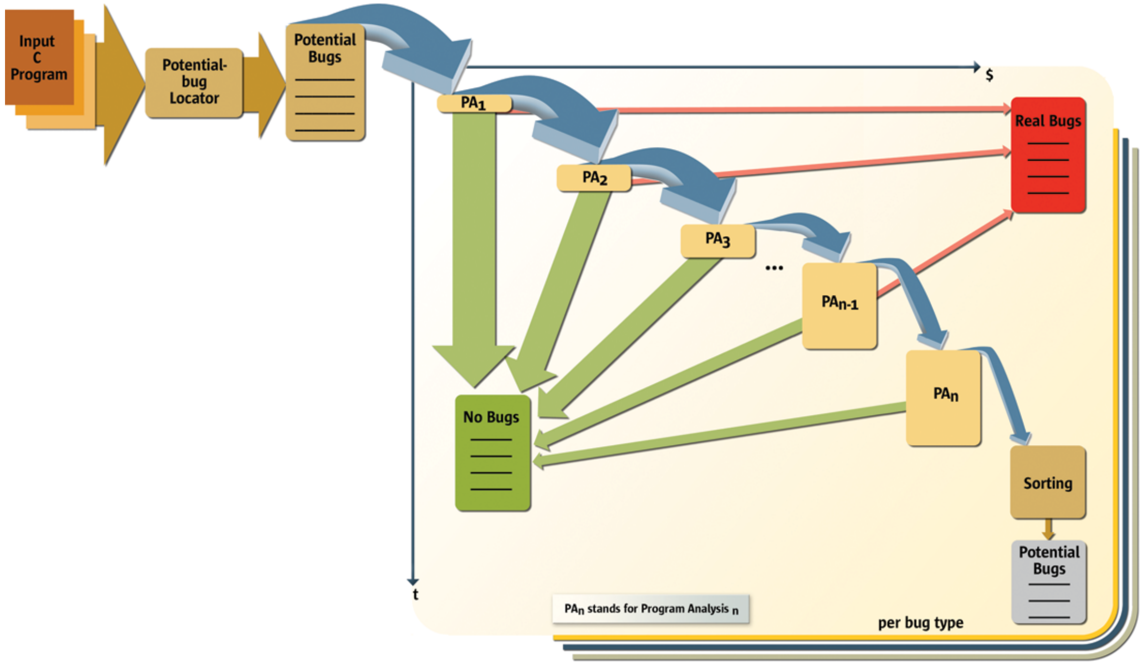
Infrastructure



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

The In Between

Original Layered Analysis Design



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

Granularity of Analysis

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

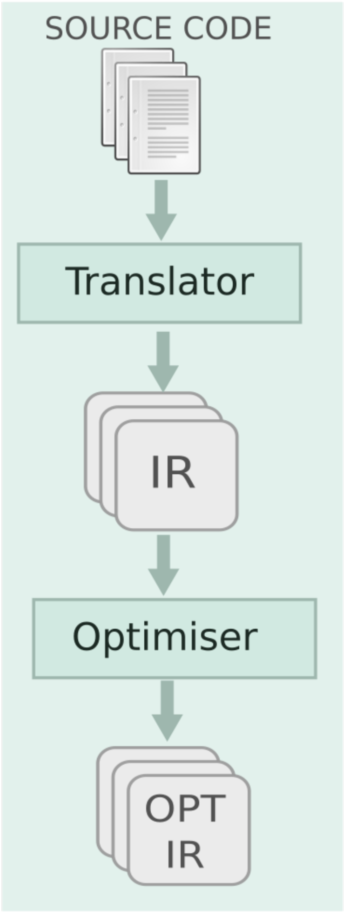
ORACLE

Parfait Infrastructure

- Replicated work due to independent development of the analyses
- 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

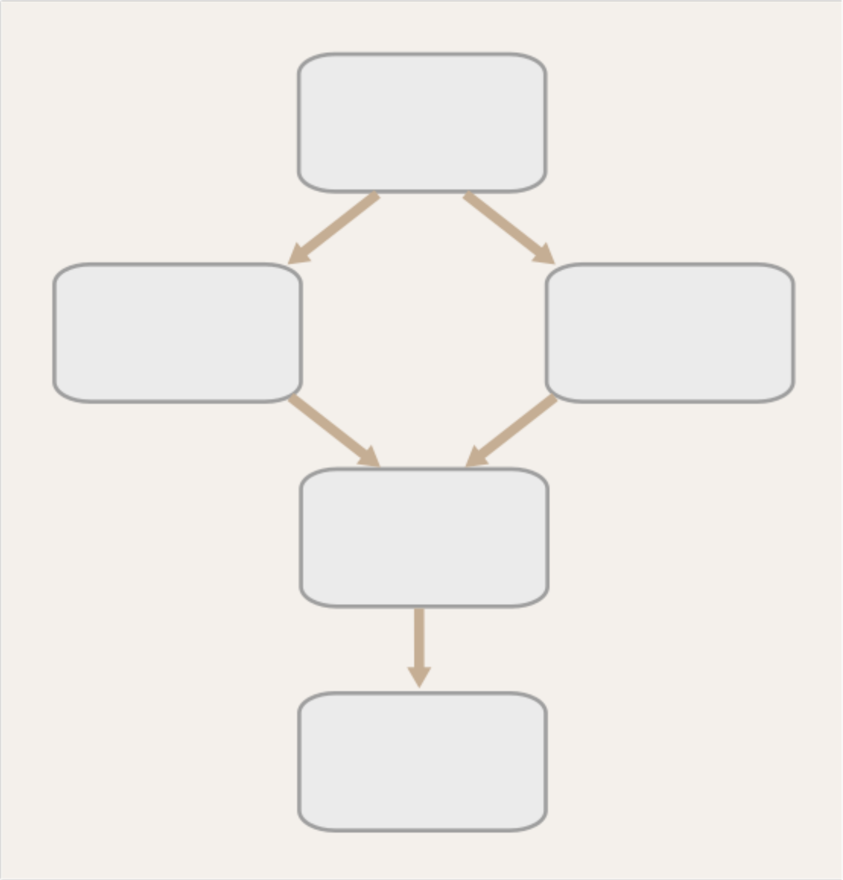
LLVM Infrastructure

- 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

- Technical debt exposed when improving analysis code coverage
- Incomplete call graph due to function pointers and virtual calls

Usability and Development Organisation's Workflow



- "Expensive" analyses are not deployed in production
  - If runtime is larger than allocated nightly integration window

Main Takeaways

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

Worked Well

- Scalability through demand-driven analyses + caching + function summaries
- Precision through unsoundness + heuristics
- Language translation for analysis
- Usability through user and organisational deployment experience

Needs More Work

- Extensibility only possible through handwritten C++
  - New languages
  - New analyses
- Infrastructure changes become challenging as time goes by

ORACLE

Many People Have Worked on Parfait Over the Years

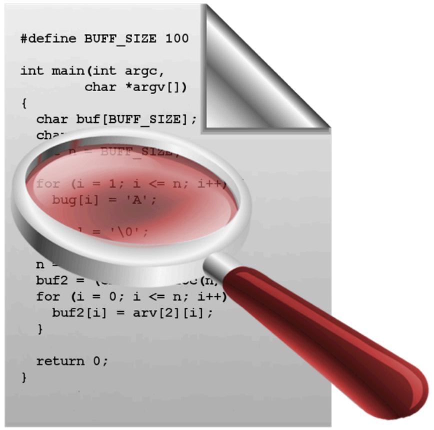
- Cristina Cifuentes
- Bernhard Scholz
- Nathan Keynes
- Lian Li
- Chenyi Zhang
- Erica Mealy
- Michael Mounteney
- Simon Long
- Nathan Hawes
- Mike Van Emmerik
- Christian Hoermann
- Manuel Valdiviezo
- Andrew Browne
- Adam Heron
- Jimmy Ti
- Jacob Zimmermann
- Andrew Craik
- Brad Moody
- Ben Barham
- 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

ORACLE

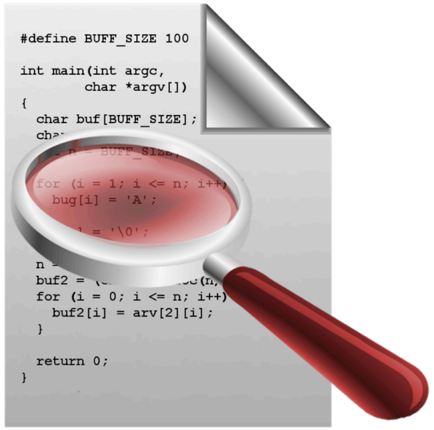
Parfait: scalable and precise bug detection for static languages

Integrated Cloud

Applications & Platform Services



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



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

Buffer Overflow Results Over Open Source OS Kernels

June 2009

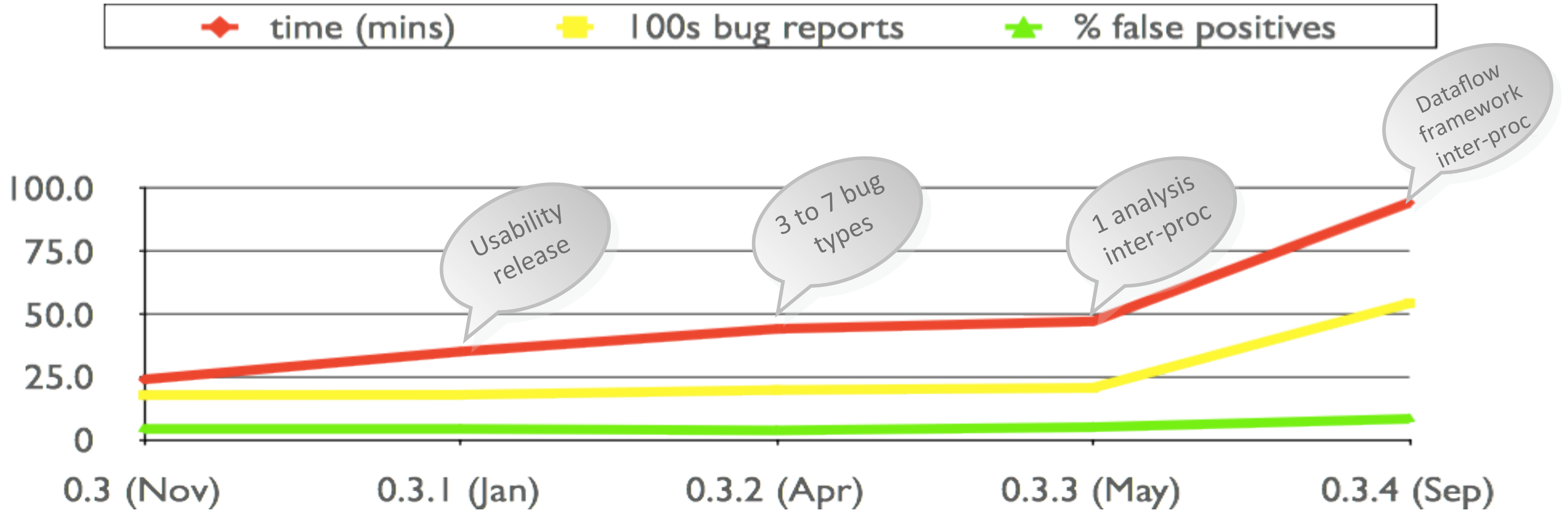
Kernel	Time (min)	Part	LOC	Buffer overrun	Bug density	Status
OpenSolaris UTS b105	5	Core	2.1M	15	0.0069	Being fixed
		Device drivers	1.2M	67	0.054	Being fixed
Linux 2.6.29*	13	Core	1.6M	12	0.0073	Fixed
		Device drivers	4.1M	85	0.020	Submitted
OpenBSD 4.4	2	Core	0.5M	3	0.0060	Fixed
		Device drivers	0.8M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

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

ORACLE

Common C Bugs Results Over OpenSolaris ON Code

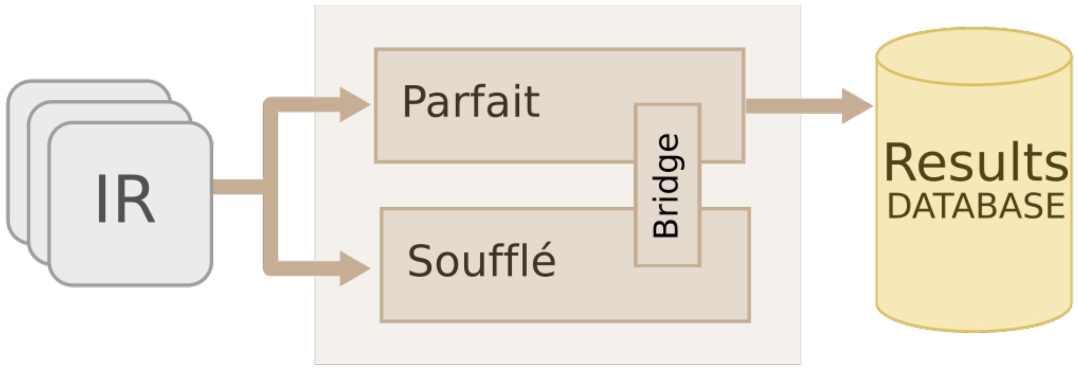
November 2009 – September 2010



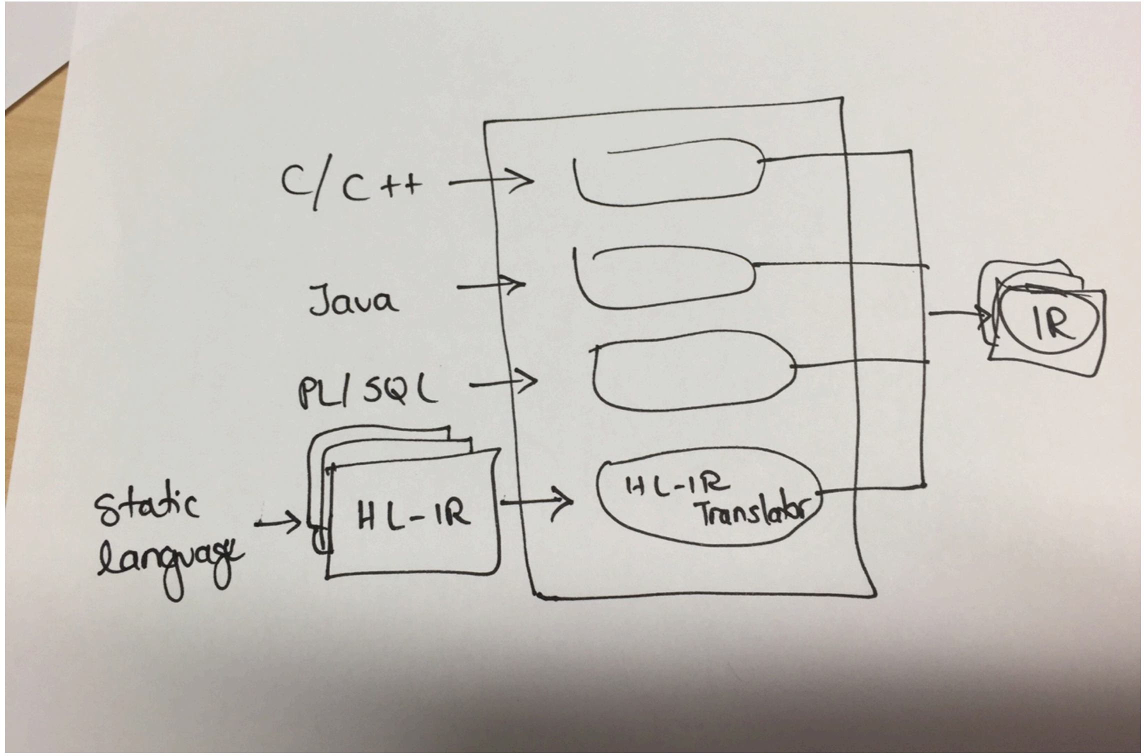
9.5 MLOC	10.3 MLOC	10.4 MLOC	10.4 MLOC	10.4 MLOC

Extensibility – Possible Solutions

Provide interface to Datalog



Provide interface to other languages



Memory Consumption

- Memory usage: 10x-20x size of .bc