Static Analysis in Industry

Andy Chou, CTO and Cofounder
Coverity, Inc.

About Coverity

- World's largest independent company focused on static analysis development testing tools.
- Coverity founded in 2003 by four CS PhD students and Prof. Dawson Engler.
- As of 2014:
    - 300 employees
    - 1100 customers
- One round of venture funding in 2007
- Cash flow positive
- Headquarters in San Francisco with offices in Boston, Seattle, Calgary, Tokyo, London.

Coverity Headquarters, San Francisco

About This Talk

- How is static analysis used in industry?
- What attributes does the market want in a static analysis tool?
- How does Coverity analysis work?

- …I don't know all the answers!

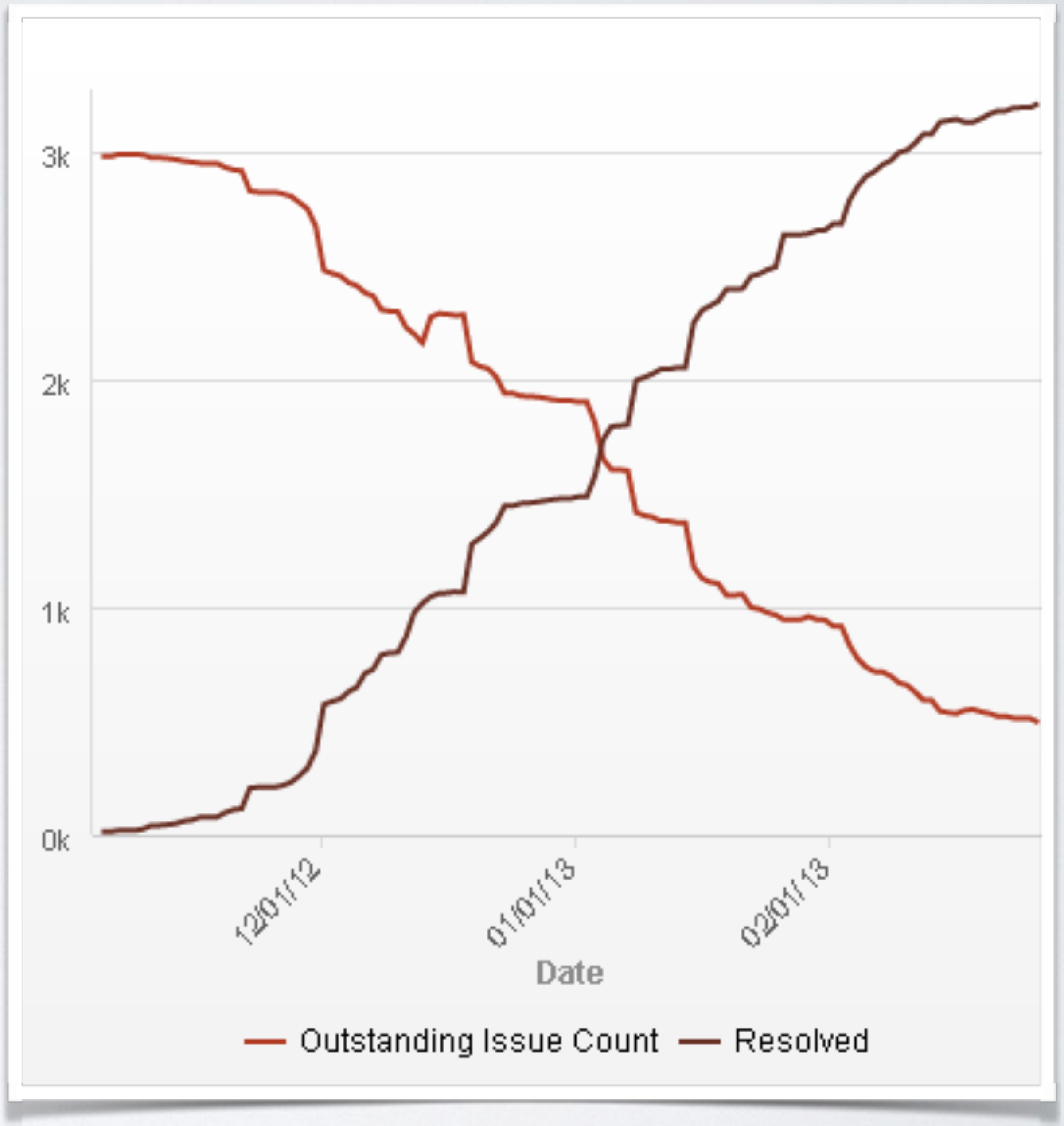
It’s not about finding bugs. It’s about fixing them.

Case Study: Company X

- In 2009, Company X evaluated Coverity on 9m lines of C/C++ code.
- Over 10,000 defects were discovered and Company X licensed Coverity for 3 years.
- But mistakes were made:
  - There was no plan for addressing defects.
  - Defects had no owners.
  - Management did not set clear expectations.
  - Slow build/analysis times.
  - Infrequent weekly analysis runs.
- At the end of year 3, no progress was made, and the renewal business was in jeopardy.

Crisis and Renewal

- Despite this failure, Company X decided to try again with new champions who believed in static analysis.
- Criteria were established for success:
  - Must fit into workflow without distraction.
  - All new defects must be automatically assigned an owner.
  - Defects assignment results in notification, with clear expectations for triage and resolution by the assignee.
  - Added release criteria to enforce zero new defects.
  - Focused effort on the most critical defects (10,000 down to 3,000).



Social Pressure

- A baseline was set at the prior release version.
- New defects introduced after the baseline were assigned an owner by using SCM integration.
- A daily top 10 new defect owner list was broadcast to the entire development team.
- This put social pressure on individuals to now appear on the list. Keeping “Coverity Clean” became a priority.

Coverity

Top 10 offenders

(sorted by defects)

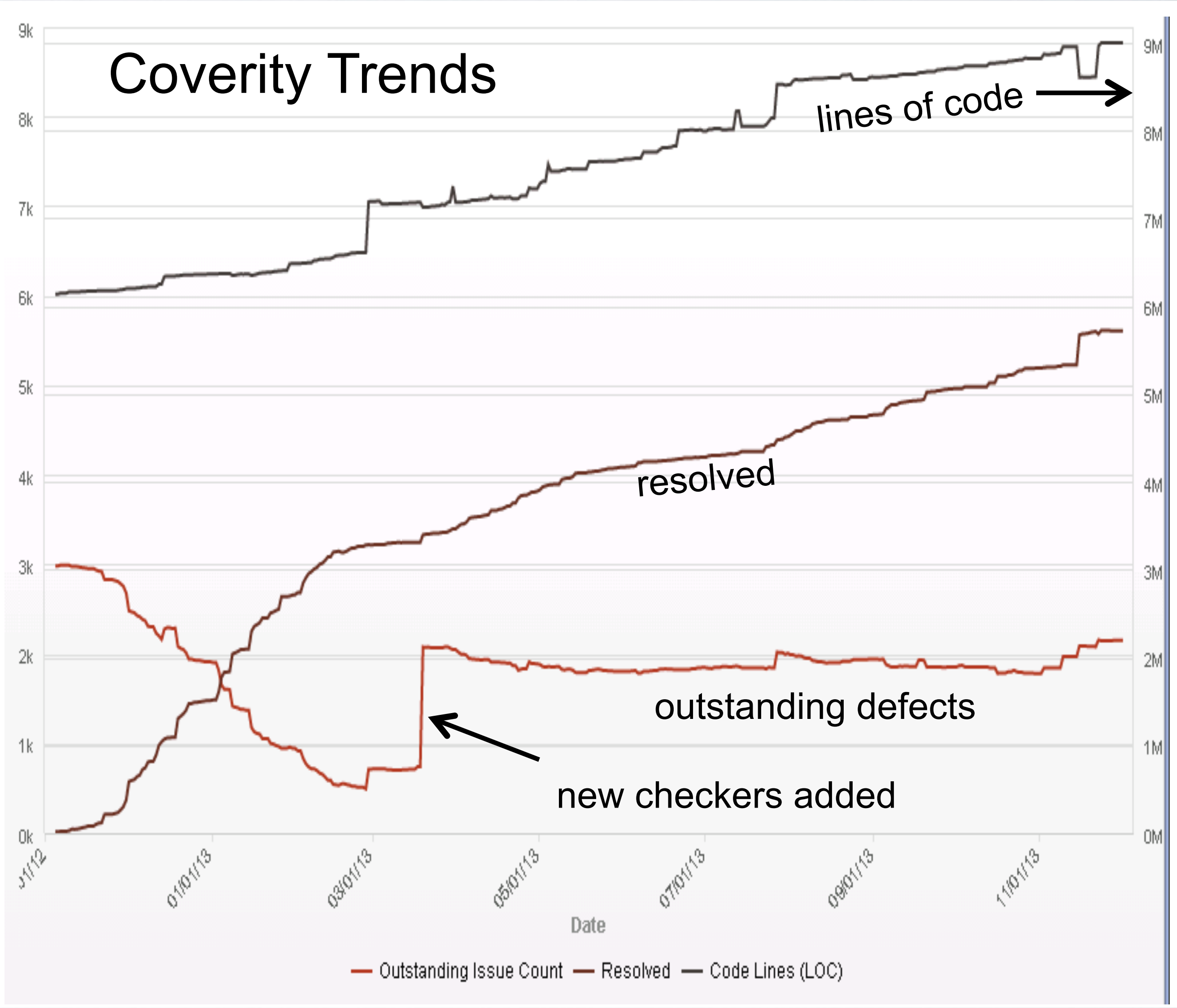
NOTE: The Coverity results below are based on a dif

OWNER	DEFECTS
mu[redacted]	168
gu[redacted]	39
an[redacted]	11
bh[redacted]	10
pr[redacted]	8
sc[redacted]	8
eu[redacted]	8
pr[redacted]	7
sa[redacted]	7
sr[redacted]	5

A full list of defects by owners may be found here, a

Constant Vigilance

- After the push, Coverity was upgraded and improved checkers uncovered 1,000 new defects.
- New defects from the upgrade were not addressed immediately.
- However, defects in new code were continually resolved as code was added or changed.



Company Y: The Value of a Memorable Bug

```
if(tRate != 0.0 && p->Qty() != 0.0) {
    mFactor = fabs(p->Cost()/tRate/p->Qty());
}
```

```
if(tRate != 0.0 && p->Qty() != 0.0) {
    mFactor = fabs(p->Cost()/eRate/p->Qty());
}
```

Company Y: The Value of a Memorable Bug

```
if(tRate != 0.0 && p->Qty() != 0.0) {
    mFactor = fabs(p->Cost()/tRate/p->Qty());
}
```

```
Copy-paste error: "tRate" in "p->Cost()/tRate" looks like a
copy-paste error.  Should it say "eRate" instead?
```

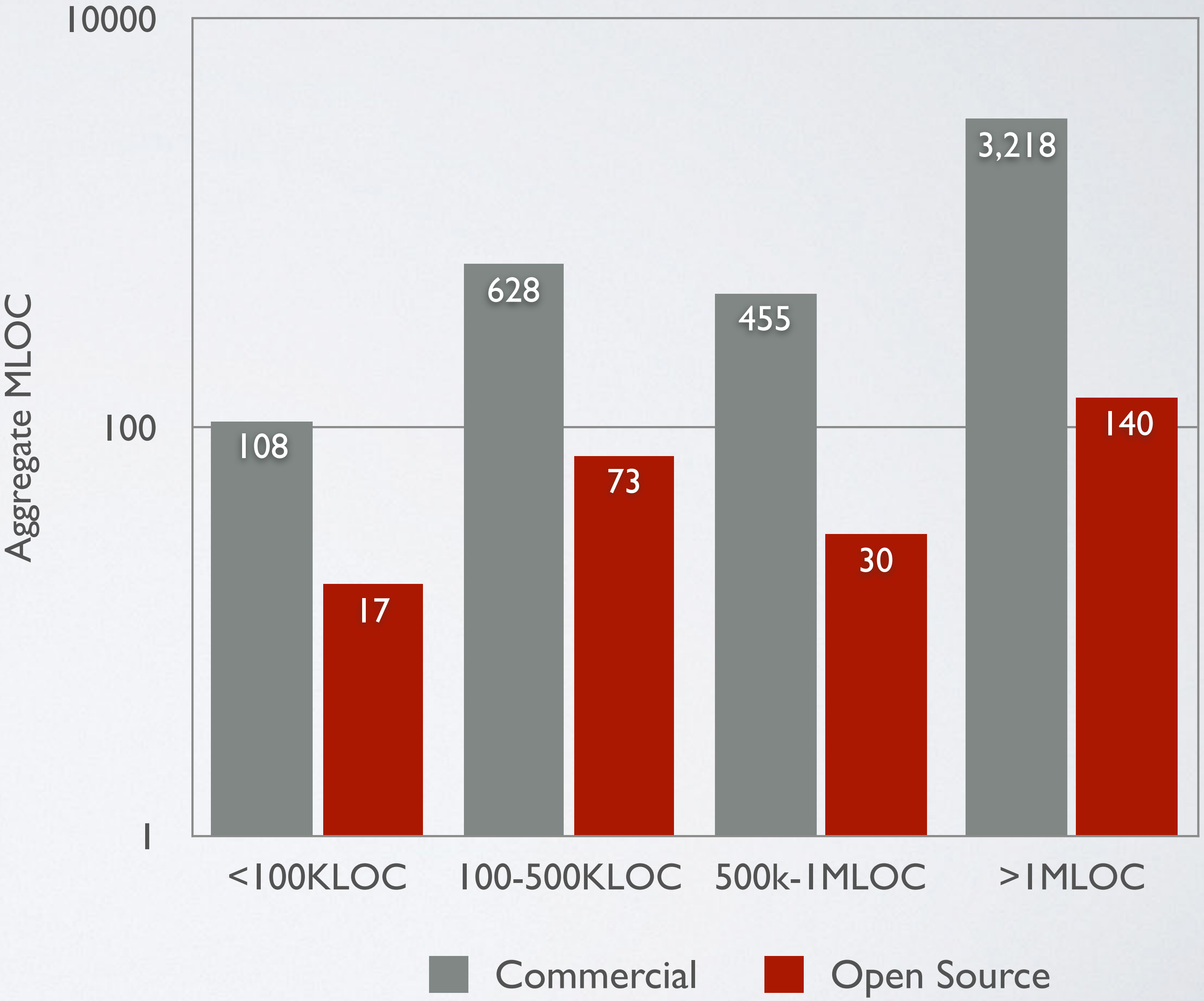
```
if(eRate != 0.0 && p->Qty() != 0.0) {
    mFactor = fabs(p->Cost()/tRate/p->Qty());
}
```

"We decided that you guys deserve a beer for this one"

Customers with bigger code bases have more money.

MLOCs and BLOCs

- This represents ~30% of Coverity customers
  - 4.5 billion LOC
  - 314 customers
  - 7,535 projects
  - Duplicate projects within 5% LOC eliminated.
  - 95% of the code is C/C++
- Open Source data is collected from the Coverity Scan project
  - 260 million LOC
  - 789 open source projects

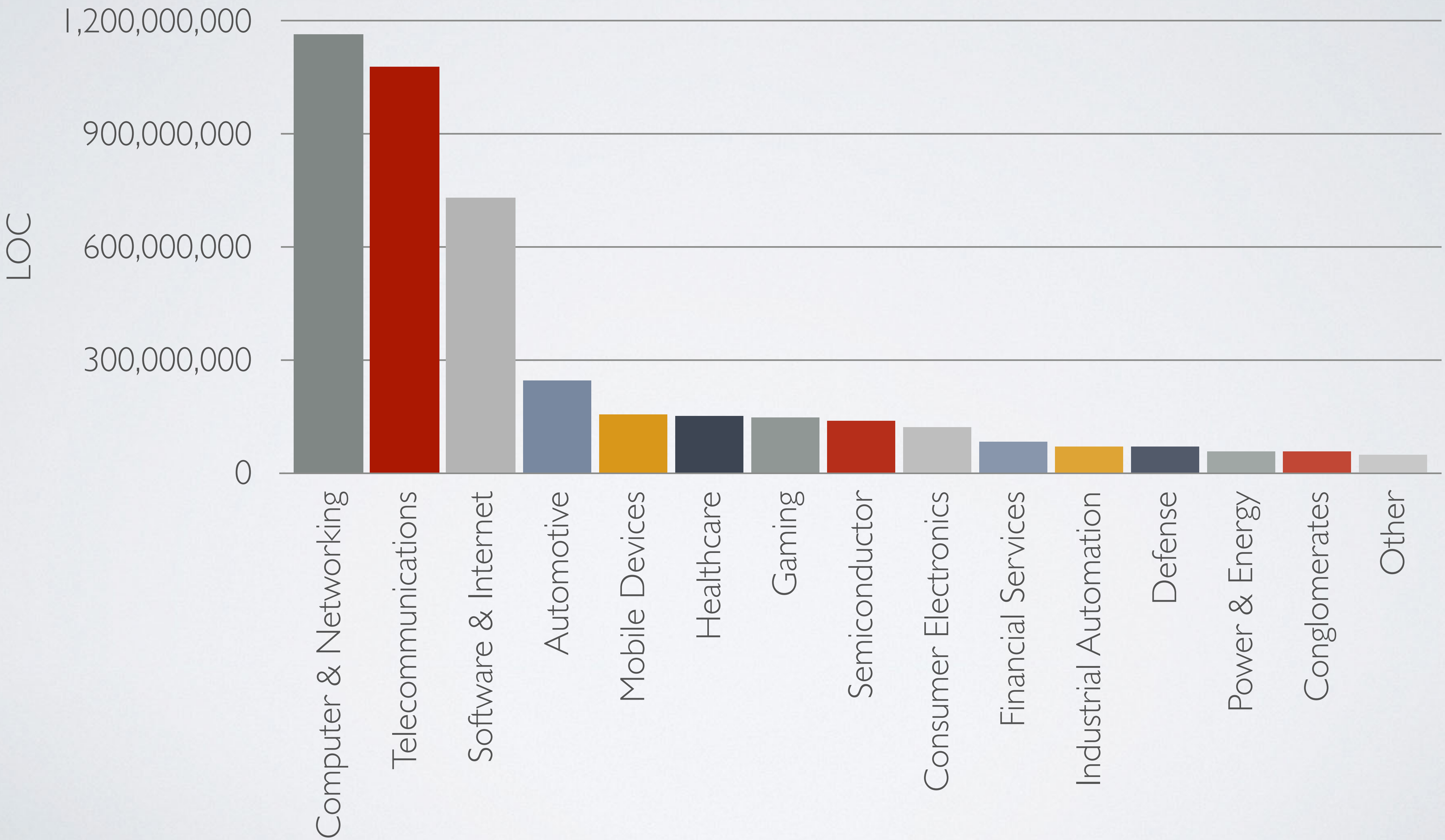


Open Source Projects in Scan with >1MLOC

Project	LOC
NetBSD	16,068,290
FreeBSD	12,649,589
LibreOffice	9,017,270
Linux	**8,578,254**
ACE+TAO+CIAO +DAnCE	7,626,092
OpenOffice	7,357,498
PostgreSQL	**6,649,825**
Thunderbird	5,066,354
Firefox	**4,997,817**
reactos	4,875,945
haiku	4,164,654
llvm	**4,014,963**
Wine	3,682,735
Wireshark	2,878,801
FxOS	2,561,607
osadl-realtime	2,548,656
GNURadio	2,340,142
globus	2,270,022
logfs	2,236,126
gcc	**1,898,975**
Samba	1,871,346
blender	1,739,394
XenProject	1,546,718

Project	LOC
InsightSoftwareConsortium	1,528,932
ScummVM	1,339,755
mantid	1,273,032
TortoiseGit	1,267,639
XBMC	1,261,638
RyzomCore	1,203,776
MariaDB	1,183,763
digiKam	1,158,653
Postgresql9	1,144,407
FreeSWITCH	1,132,527
TC	1,125,919
TrinityCore	1,116,027
MPC-HC	1,108,151
Mesa	1,104,103
0 A.D.	1,103,170
KDE	1,071,697
NuPIC	1,070,950
openWNS	1,042,486
cloudstack	1,027,683
gstreamer	1,023,454

C/C++ Code by Industry



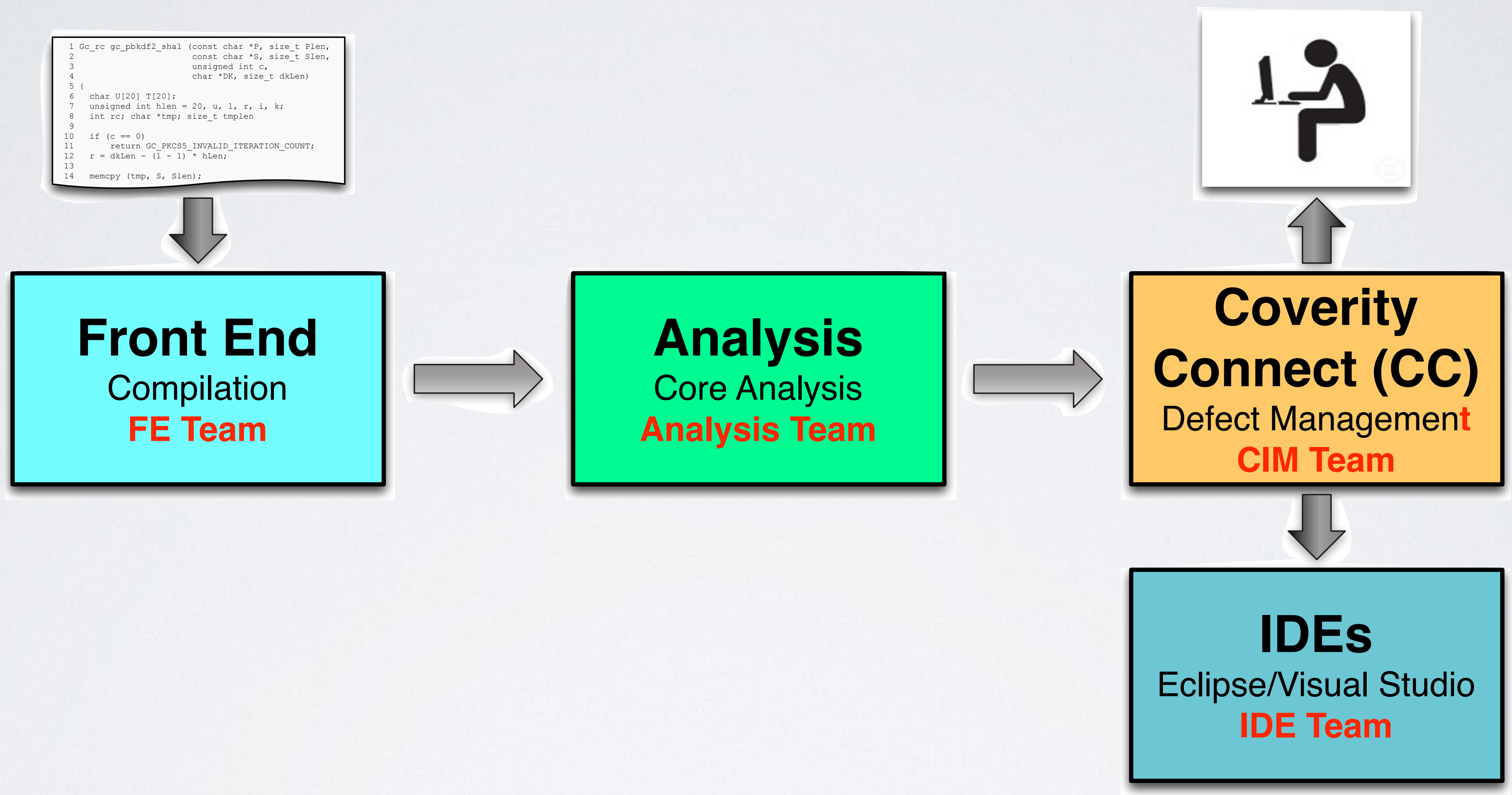

GARMIN
KYOCERA
Sony Mobile Communications
TOSHIBA
Panasonic
ideas for life
EA
NOKIA
LG
SAMSUNG
SEGA
CISCO
amazon.com
intel
hp
WILLIAMS-SONOMA
EMC2
where information lives
vmware
YAHOO!
Adobe
DELL
Google
SAP
GM
TOYOTA
FDA
NASA
NEC
SIEMENS
Hospira
Medtronic
NASDAQ
Bloomberg
NYSE Euronext
Deutsche Bank
BARCLAYS
AVIVA
WELLS FARGO
HSBC
DirectEdge

What are the Market Segments?

- C/C++ static analysis for quality and security accounts for > 85% of Coverity's revenue. This segment of the market is likely north of $100m.
- Security static analysis, especially for web applications in Java and C#, is also a large market dominated by HP/Fortify and is likely north of $120m.

Customers want a ~~product~~ solution.

The Analysis Workflow



55 compiler translators

158 checkers

4 Languages

3 IDEs

Breadth of Defect Coverage

Resource Leaks
- Memory leaks
- Resource leak in object
- Incomplete delete
- Microsoft COM BSTR memory leak

Uninitialized variables
- Missing return statement
- Uninitialized pointer/scalar/array read/write
- Uninitialized data member in class or structure

Concurrency Issues
- Deadlocks
- Race conditions
- Blocking call misuse

Integer handling issues
- Improper use of negative value
- Unintended sign extension

Improper Use of APIs
- Insecure chroot
- Using invalid iterator
- printf() argument mismatch

Memory-corruptions
- Out-of-bounds access
- String length miscalculations
- Copying to destination buffers too small
- Overflowed pointer write
- Negative array index write
- Allocation size error

Memory-illegal access
- Incorrect delete operator
- Overflowed pointer read
- Out-of-bounds read
- Returning pointer to local variable
- Negative array index read
- Use/read pointer after free

Control flow issues
- Logically dead code
- Missing break in switch
- Structurally dead code

Error handling issues
- Unchecked return value
- Uncaught exception
- Invalid use of negative variables

Breadth of Defect Coverage

Program hangs
- Infinite loop
- Double lock or missing unlock
- Negative loop bound
- Thread deadlock
- sleep() while holding a lock

Null pointer differences
- Dereference after a null check
- Dereference a null return value
- Dereference before a null check

Code maintainability issues
- Multiple return statements
- Unused pointer value

Web Security
- Cross-site Scripting
- SQL Injection
- App server misconfiguration
- LDAP Injection
- Script injection
- Other forms of injection

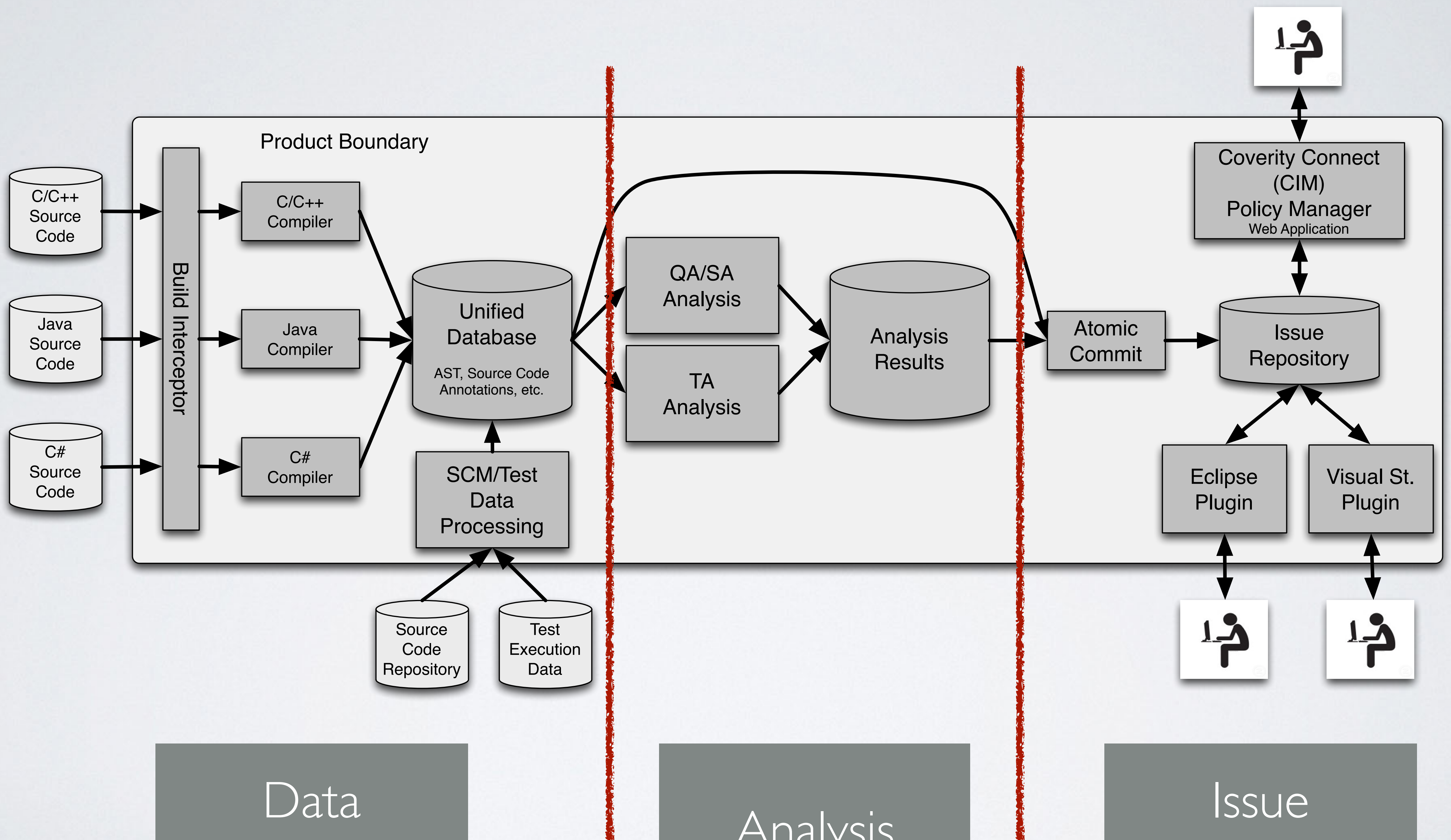
C/C++ Security
- Integer overflow
- Loop bound by untrusted source
- Write/read array/pointer with untrusted value
- Format string with untrusted source

Performance inefficiencies
- Big parameter passed by value
- Large stack use

Security best practices
- Possible buffer overflow
- Copy into a fixed size buffer
- Calling risky function
- Use of insecure temporary file
- Time of check/time of use
- User pointer dereference

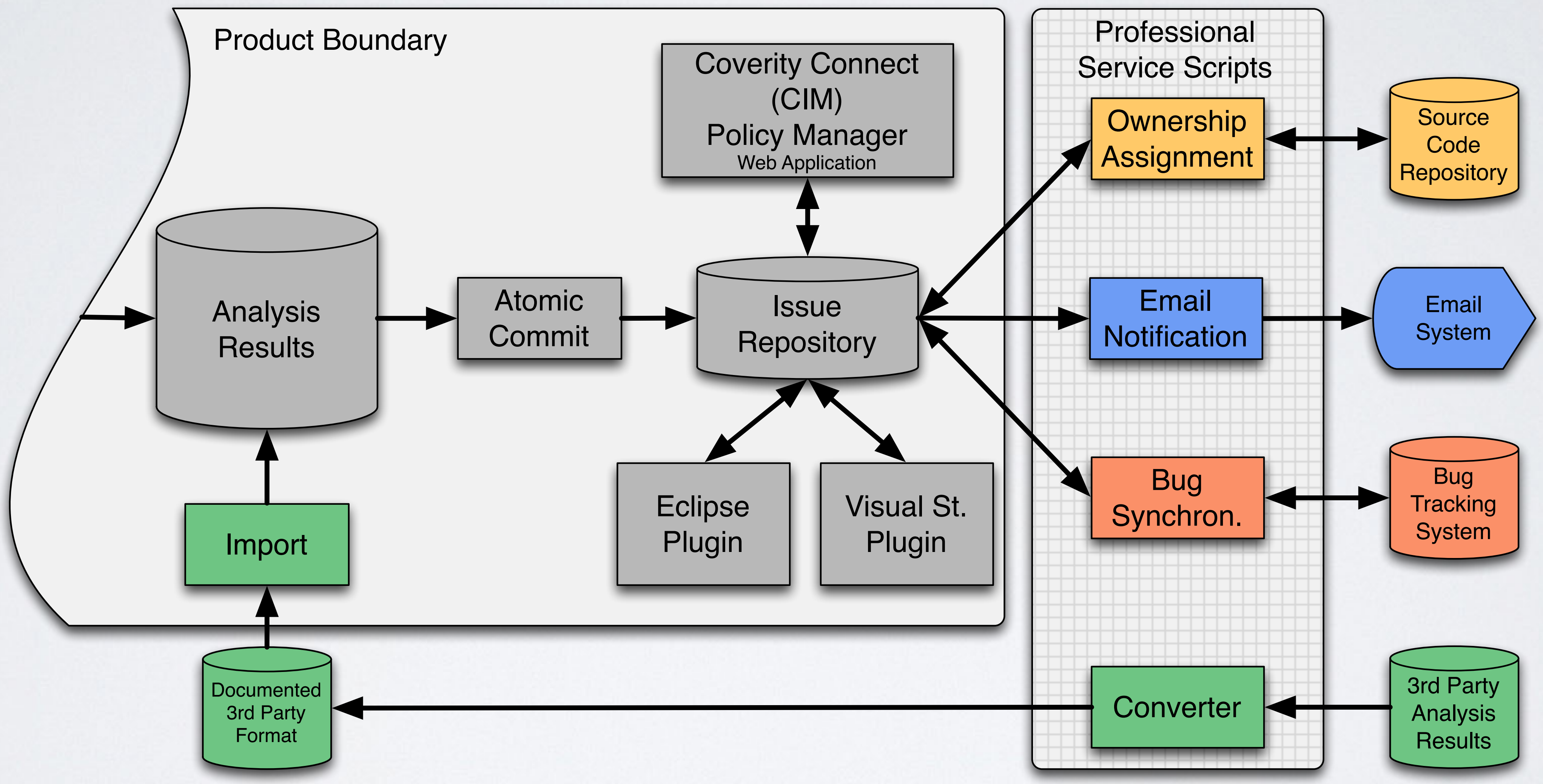
Other
- Copy-paste errors

Product Architecture



Data Collection | Analysis | Issue Management

Product Extensions



Performance

Code Base	MLOC	Time (min)
qt-x11-free-3.3.2	0.6	9.4
Proprietary X	1.1	31
firefox-2.0	1.8	14
Proprietary Y	2.4	35
kde-3.5.5	6.0	42
openoffice-2.4	6.7	130
Linux-3.x	7.7	38
Proprietary Z	8.0	107

Memory: 1GB + 0.5GB per worker

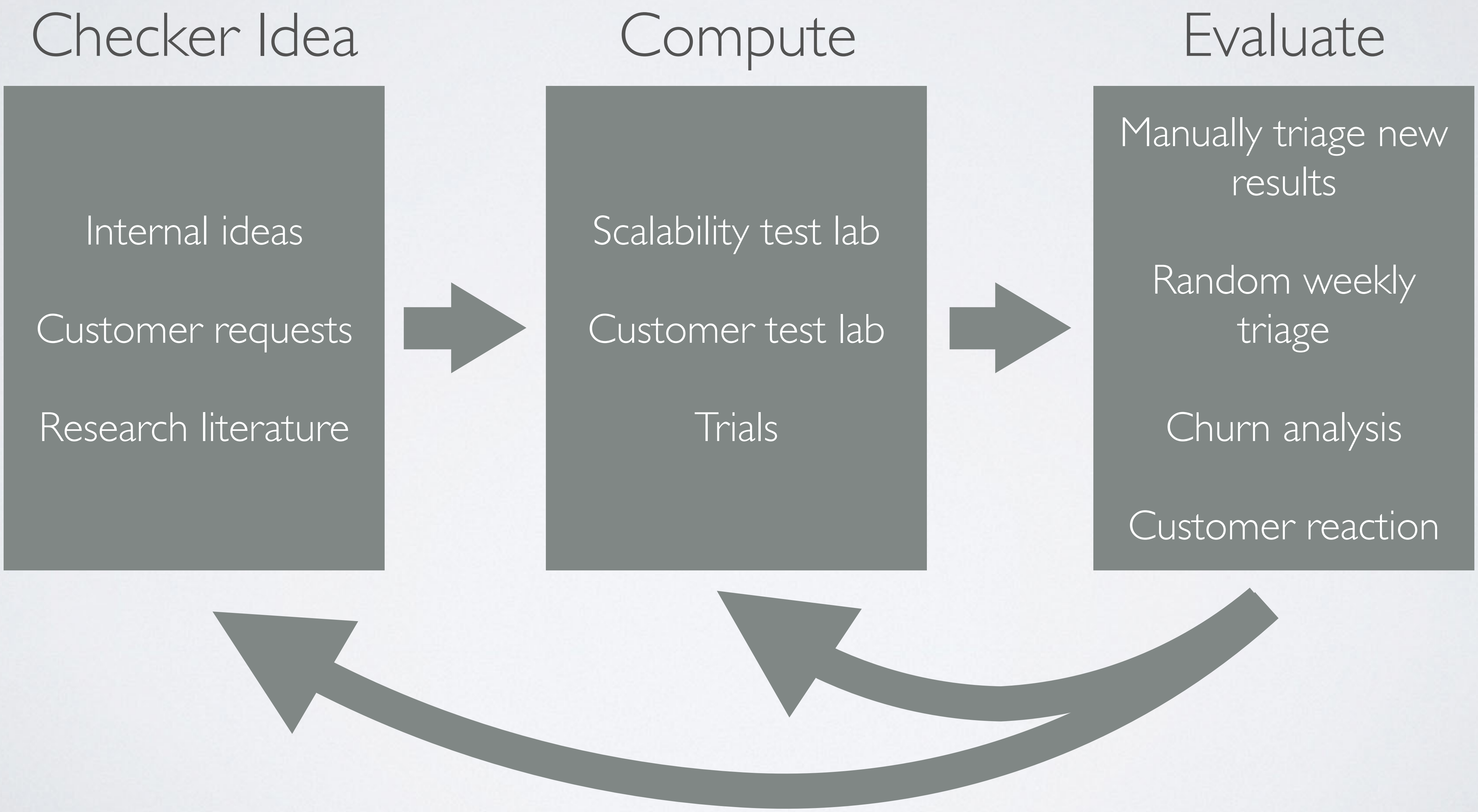
Choosing what *not* to report is at least as important as finding more defects.

Checker Development Methodology

Guess → Compute Consequences → Compare Experiment

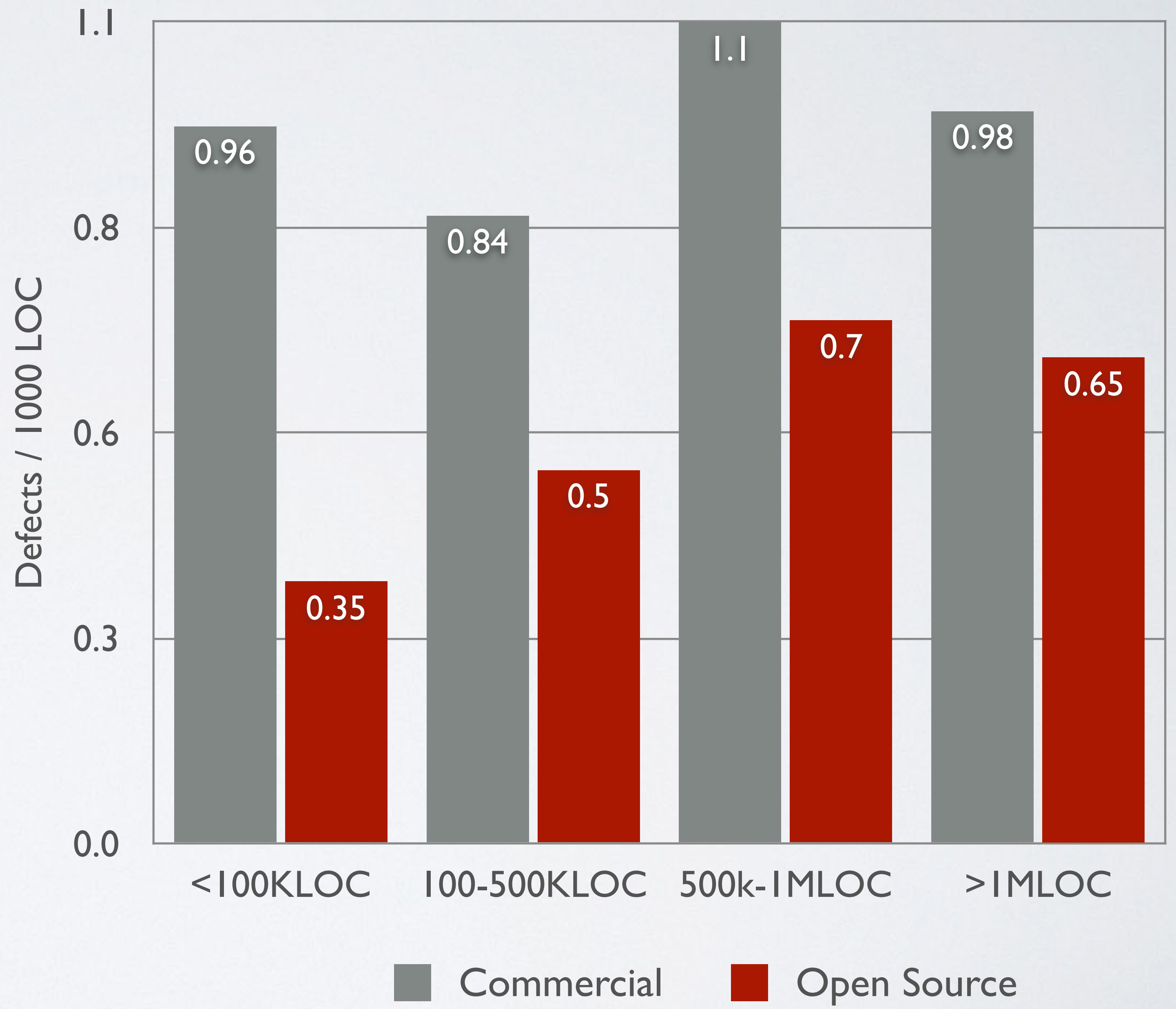
(Richard Feynman)

Checker Development Methodology



C/C++ Defect Density

- Commercial defect density is in the same ballpark as open source.
- Open source is used to tune the analysis, so a lower FP rate out of the box is to be expected.
- Some open source projects have a long history of fixing defects.
- Commercial data is biased towards newer customers who have turned on data collection.



Demo

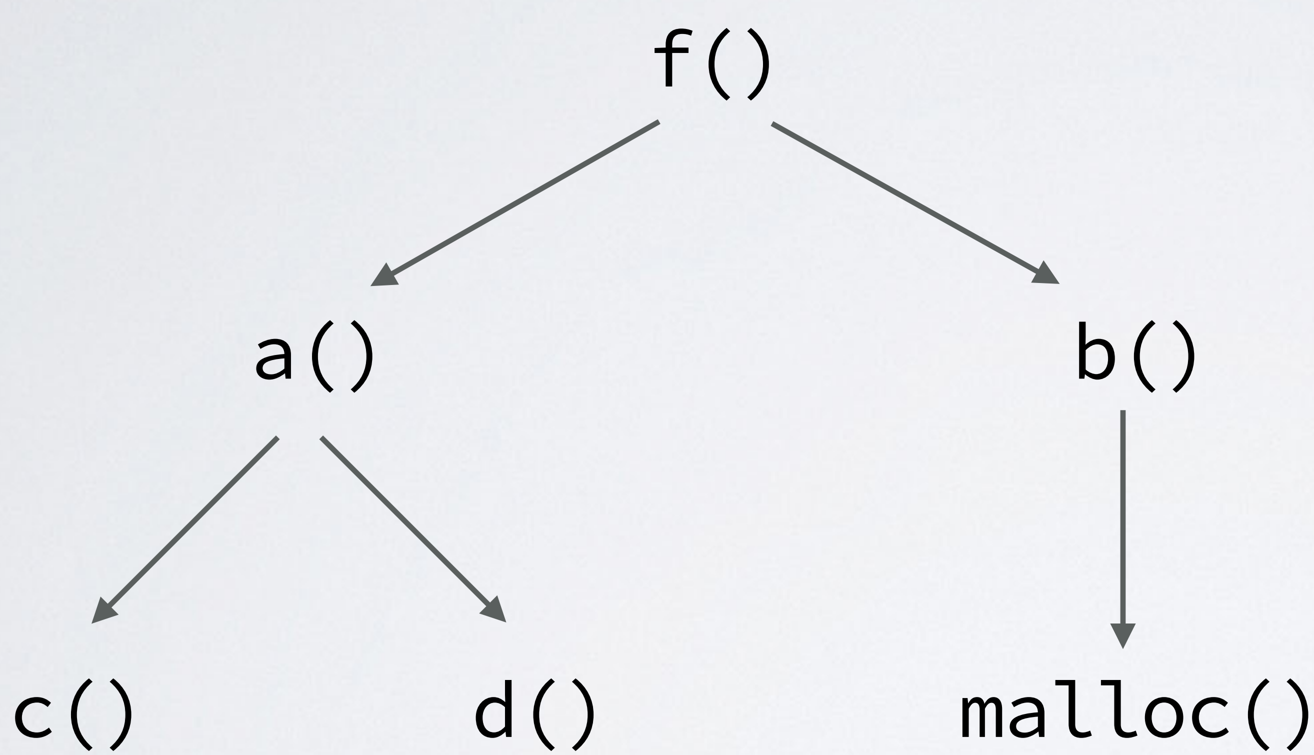
Styles of Analysis

- Intraprocedural checks, both flow sensitive and insensitive
- Interprocedural control flow based
  - Bottom-up, context sensitive, path sensitive
  - Examples: Null pointer dereferences, buffer overruns
- Statistical
  - Adds global statistical data as evidence in addition to visible control/data flows.
  - Examples: Return value checking, race conditions
- Global dataflow
  - Geared towards security checks around use of tainted data
  - Examples: XSS, SQL injection

Interprocedural Analysis

`a()` `b()` `c()` `d()` `f()` `malloc()`

Callgraph Construction



- Class Hierarchy Analysis (CHA) for callgraph construction
- Unsound, custom alias analysis for function pointers
- Recursive cycles broken, with heuristics to detect likely false edges from inaccurate virtual call resolution

Intraprocedural Analysis

c()

Checkers	Pass 1	Pass 2
OVERRUN	1	0
NULL_RETURNS	1	2
...		

Defect Reports

- Each function is analyzed by a sequence of independent *checkers*, which analyze each function in turn.
- Each checker has its own abstraction of program state. States between checkers are not mingled, but there are some parameterized, reusable abstractions that some checkers share.
- The most common checker architecture explicitly traverses control flow, avoids widening and merging of states, and uses a 2-pass mechanism for FPP.

False Path Pruning

Pass 1

Pass 2

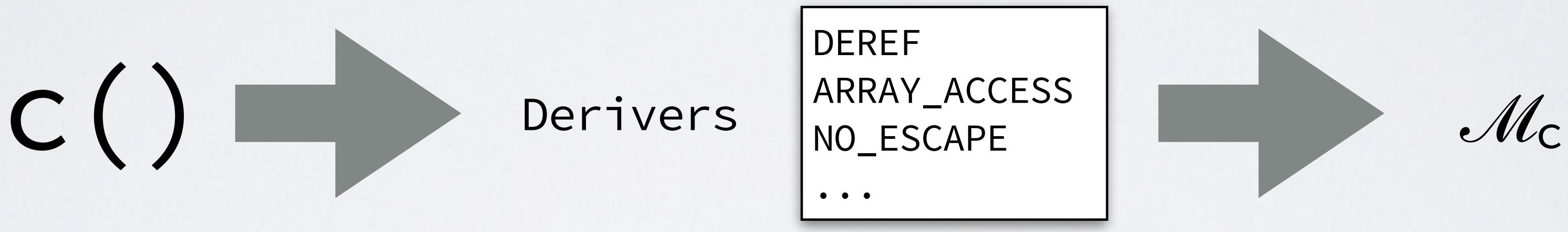
$\texttt{checker(c)} \xrightarrow{\text{defect}} (\texttt{checker} \circ \texttt{FPP})\texttt{(c)}$

$\texttt{checker(c)} \xrightarrow{\text{no defect}} \text{Next function}$

```
Intervals: x ∈ [Min, Max]
Disequality: x ∉ { Values }
Masks: (x & MASK) = 0
Branches: if(expr) … if(expr)
Exceptions: throw new E -> catch(E)
Increment: for(i=0; i<12; i+=2)
Non-null: *p ... if(p)
Types: vcall consistency on paths
```

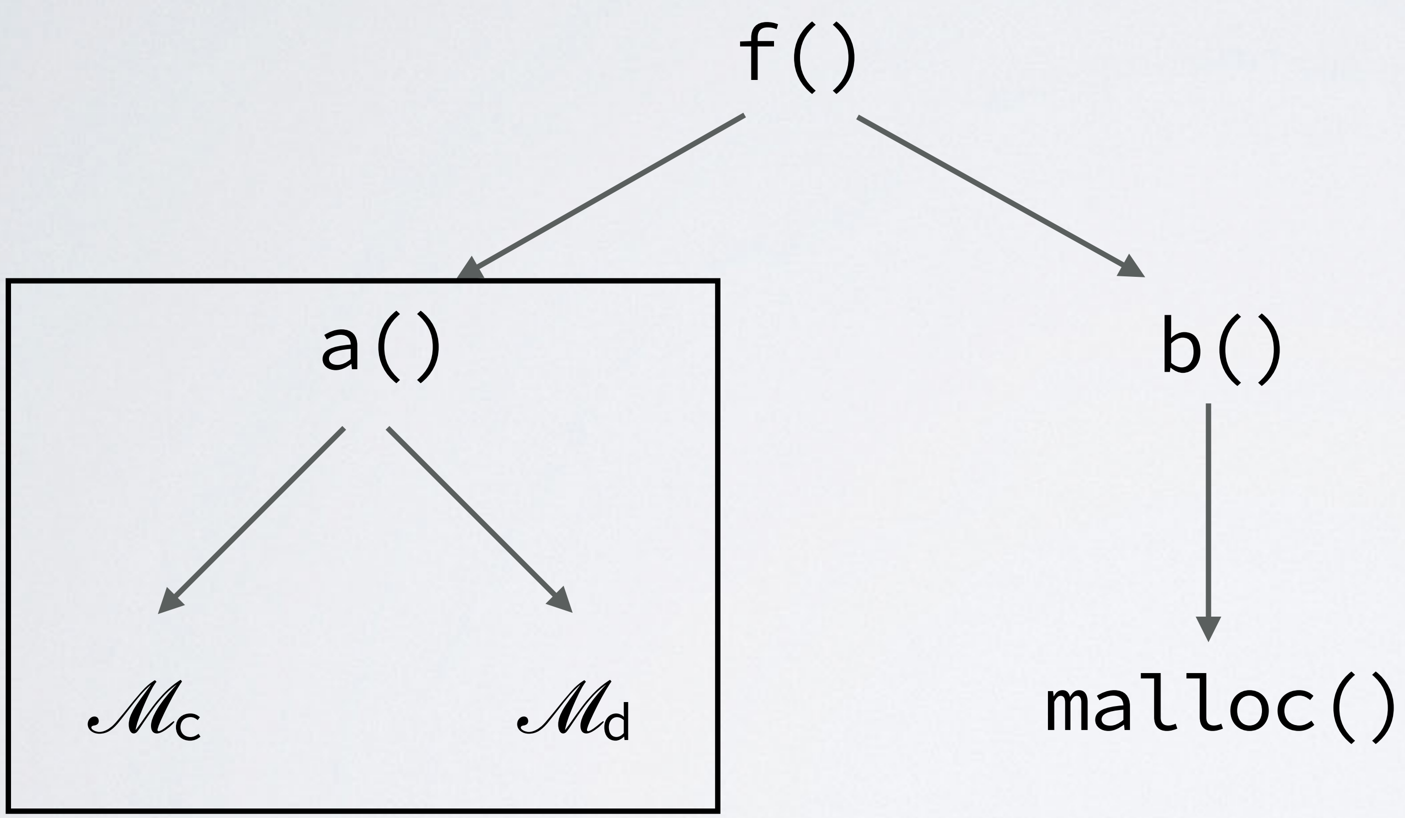
An alternative implementation uses a SAT solver.

Summaries



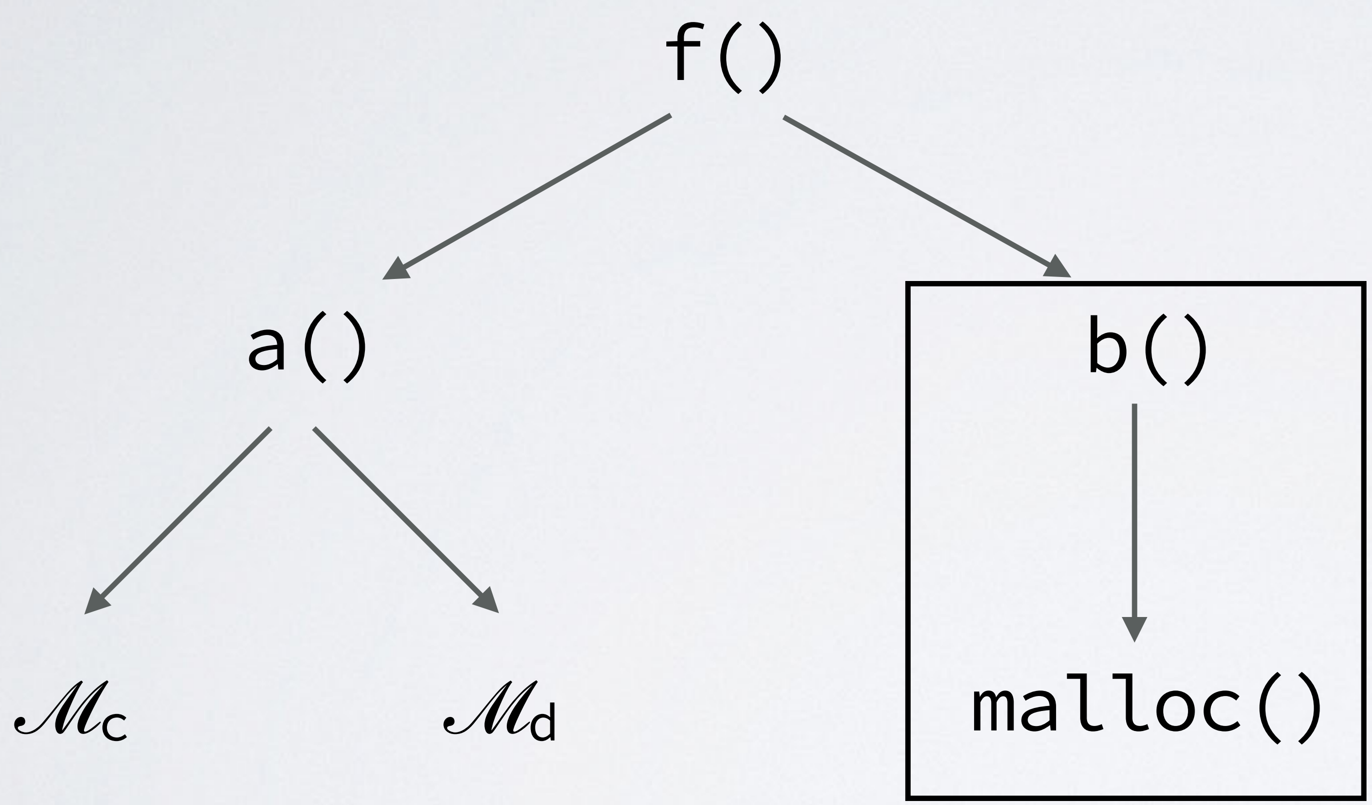
- Each function is also analyzed by a set of *derivers* that generate summaries that called *models*

Interprocedural Analysis



- Models contain control flow edges and *model events.*
- Models are substituted for function calls by modifying the control flow graph at each callsite.
- Model size is aggressively limited by restricting the locations that can be addressed to a limited number of *interfaces*
- Memory use constrained by limiting analysis to one function at a time, along with the models for all callees.
- Highly parallelizable based on the callgraph structure

Library Models



- Library functions have built-in models
- These models are manually written in C code using special primitives
- Users can also create library models to address libraries where no source is available to override the automatically derived summary.

“We have to put off fixing static analysis bugs because we’ve got fires to fight.”

Do People Care About Quality?

	Measurable?	Visible?	Cost or Opportunity?	Best Practice?
Improve Quality	No	No	Cost	No
Speed up Development	Yes	Yes	Opportunity	No
Reduce Risks	No	No	Cost	Some
Increase Agility	Some	Yes	Opportunity	Yes
Lower Costs	Yes	Yes	Cost	Some

This is a coarse generalization based on observations. Specific companies will differ.

Challenges

- Economies of scale are hard to achieve with a highly fragmented market
  - Diversity of languages, frameworks, coding styles, defect types, libraries, design patterns, compilers
  - Keeping up with changes in all dimensions is expensive
- It's hard to tune analysis without a large corpus of source code
  - Companies are *very* reluctant to share their commercial code.
  - Outside of C/C++, representative open source code is not always available. E.g. Java and C# web applications.
  - Large companies are reluctant to put critical source code in the cloud

Opportunities

- Integrating multiple sources of evidence
- Leveraging the software supply chain
- The rise of GitHub
- De facto standards
- Collaboration between academia and industry

"We came to find bugs. We stayed because it made us better software developers."

–Software Development Manager, Intuit

Engage with Coverity

- Academic licensing program
    - Access to Coverity Quality Advisor for classroom use and limited research use.
- Coverity Scan
    - Free scanning of open source projects.
    - http://scan.coverity.com

Q&A

Andy Chou
andy@coverity.com
Twitter @_achou

Appendix

Coverity on Coverity