Development at the Speed and Scale of Google

Ashish Kumar
Engineering Tools

Google

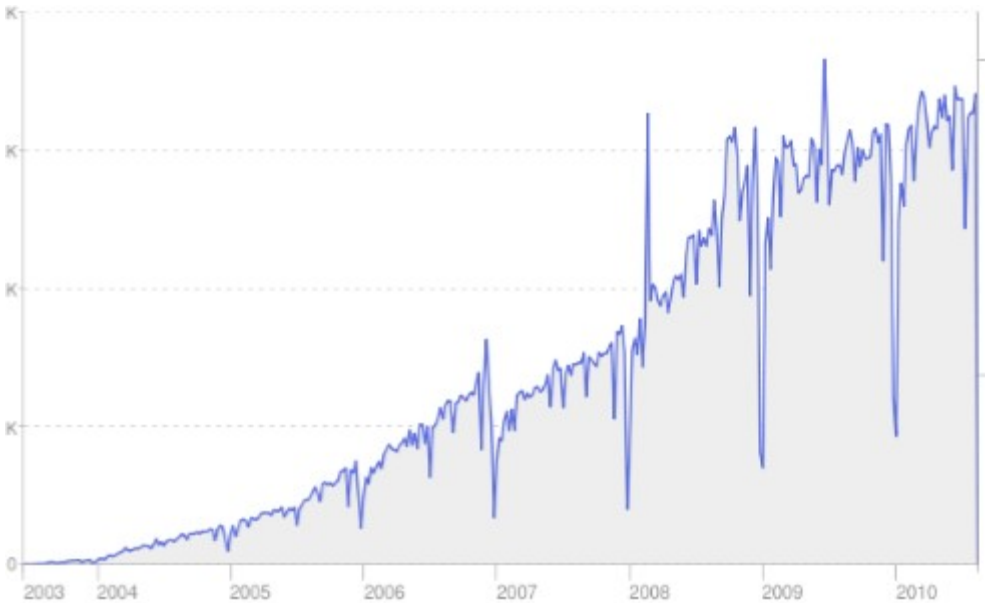
The Challenge

Speed and Scale of Google

- More than 5000 developers in more than 40 offices
- More than 2000 projects under active development
- More than 50000 builds per day on average
- More than 100 million test cases run per day
- 20+ code changes per minute; 50% of the code changes every month
- Single monolithic code tree with mixed language code
- Development on head; all releases from source

Single monolithic code tree ...

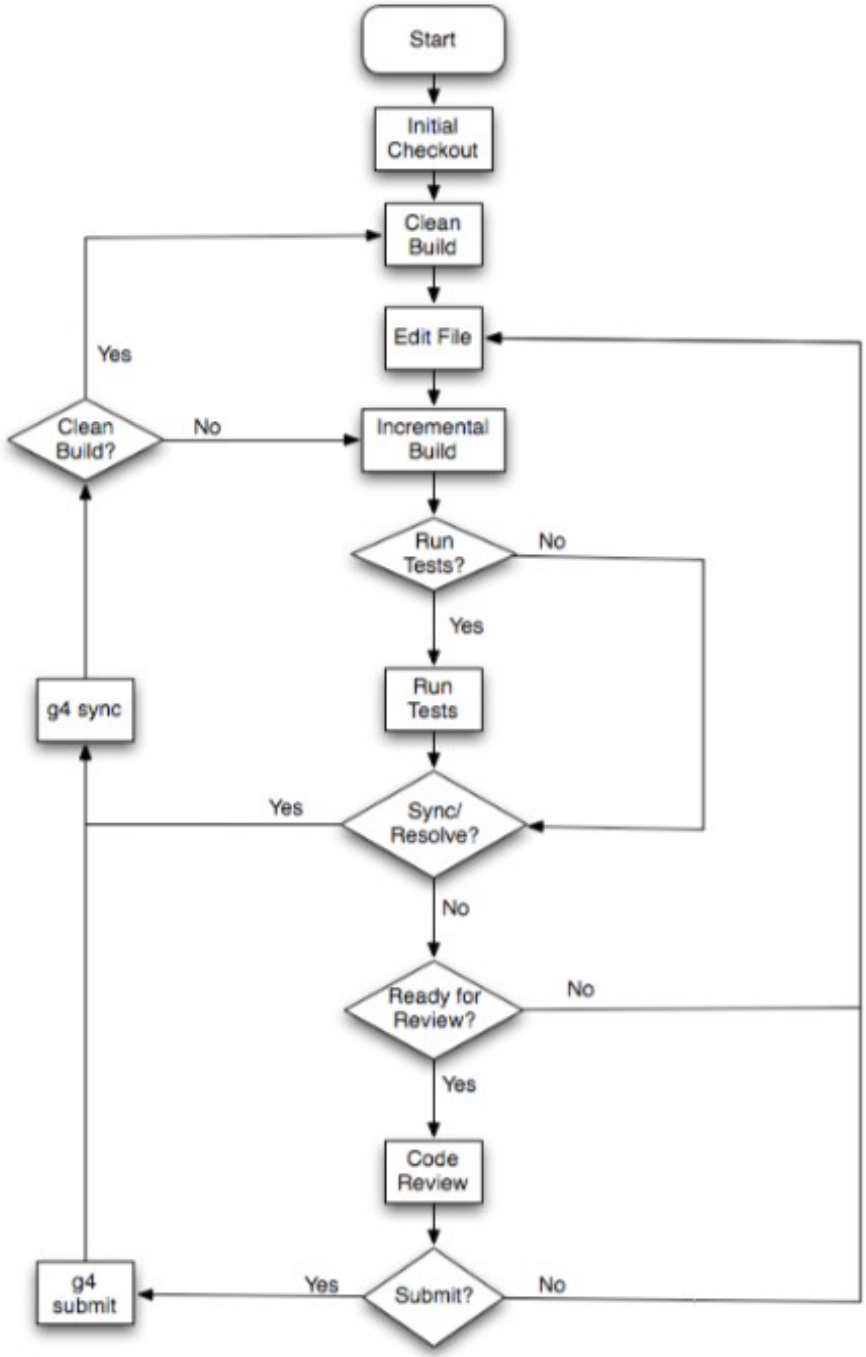
- Develop at head
- Build everything from source
- Extensive automated tests running at each changelist



- Need strong enforcement of coding style and guidelines
- Can make changes to kernel, gmail and buzz in the same changelist
- Complex dependency graph across products and libraries

Why do we care?

Rough developer workflow

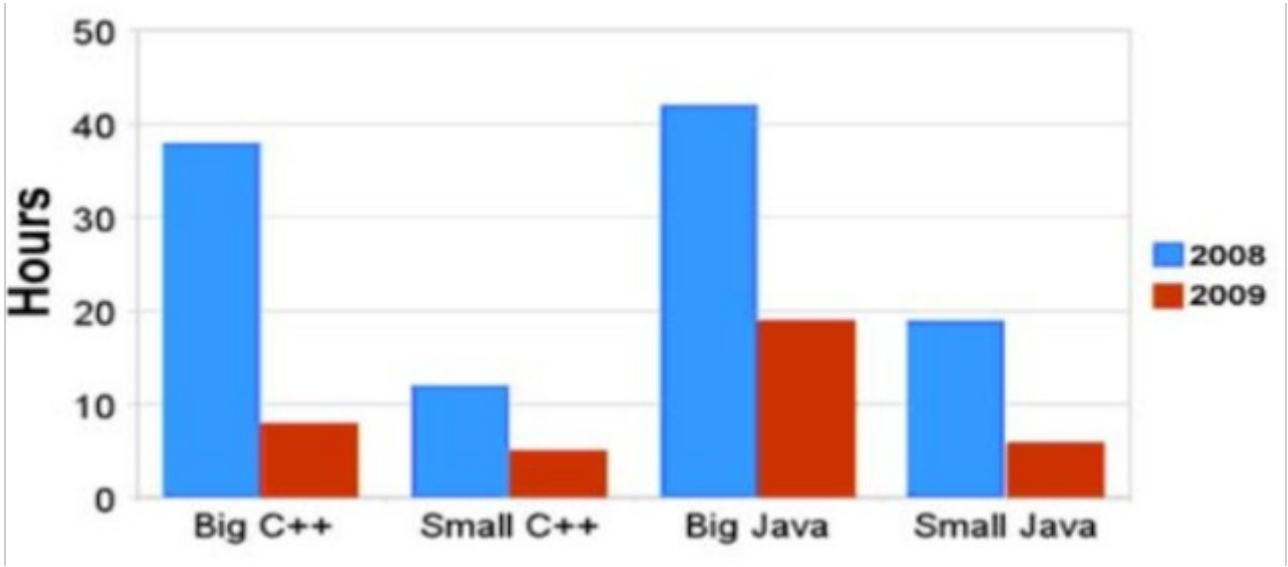


Estimating build tools savings 2008 to 2009

- Rough use case estimates

ACTIVITY	INITIAL CHECK-OUT	CLEAN BUILD	BUILD AFTER EDIT	BUILD AFTER SYNC	RUN TESTS
FREQUENCY	2	4	160	20	60

- Estimated Time waiting on build tools



- Estimated Savings: ~600 person years

Who we are

Engineering Tools and Engineering Productivity

- Google Focus Area: Engineering Productivity
  - Focus on Accelerating Google
  - Includes Test Engineering, Release Engineering, Engineering Docs and Education, ... , and Engineering Tools

- Engineering Tools
  - Focused on providing tools that accelerate Google engineers from idea to production
  - 100+ team of engineers spread across 4 major sites
  - Builds and manages tools related to Source Control, Developer Tools and IDEs, Test Infrastructure, Build Tools and Infrastructure, Project Management Tools, and others

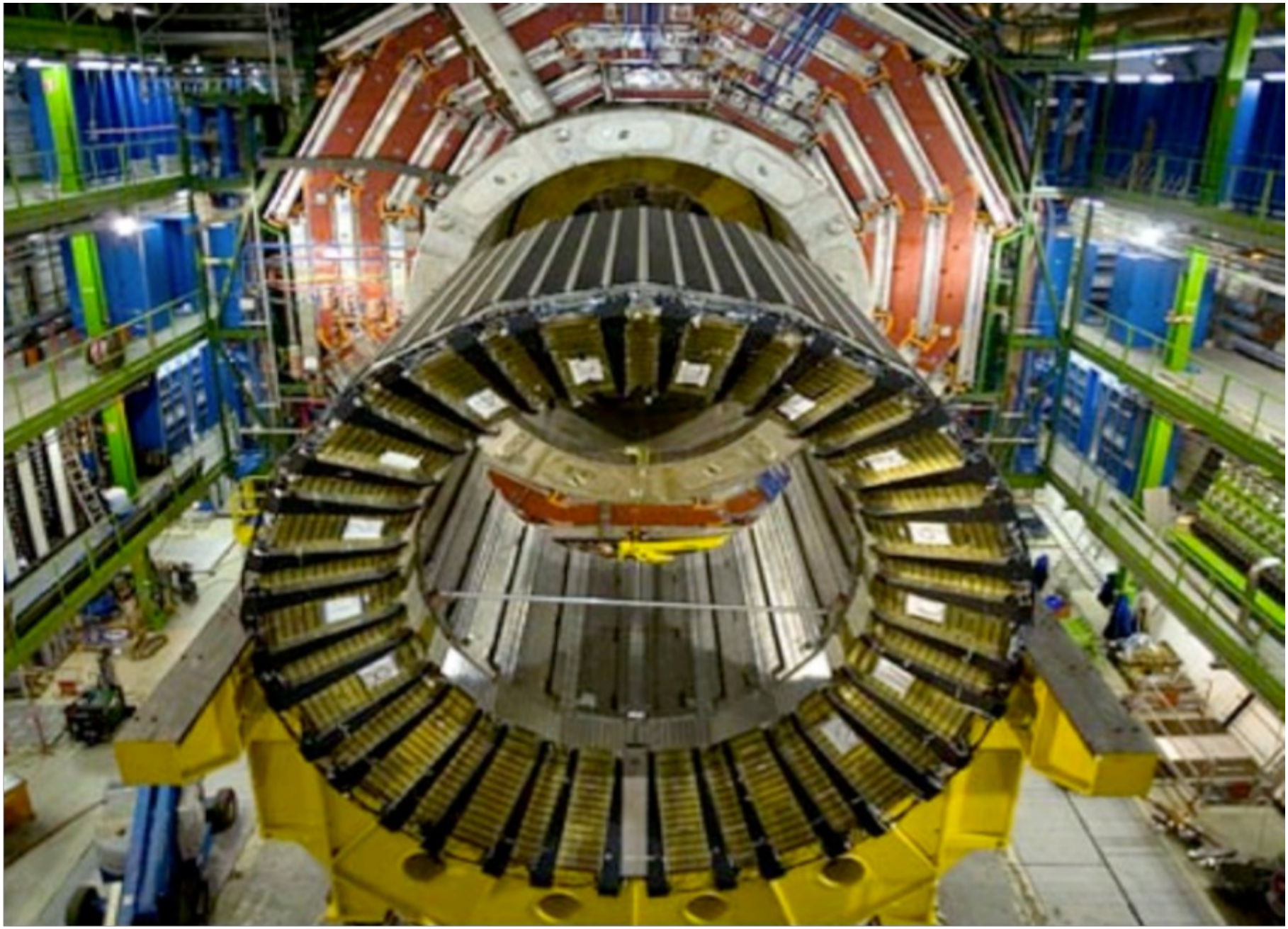
What's Unique?

- Significant investment in infrastructure for developers
  - Core infrastructure technologies like GFS, BigTable etc. that developer can quickly build systems on
  - Core tools that developers can quickly build, test and release their products / projects with
  - Tools leverage the same production infrastructure that our products do

- Continuous Improvement with Tools
  - "We can't improve what we can't measure"
  - Data-driven culture: strong focus on metrics for improvement
  - Our goal: make the tools disappear from the workflow

How we do it

Building for scale



Our version

- "Free" infrastructure for all teams
  - Transparency of code changes through centralized code review service
  - Developers can run affected tests before submitting code
  - Run every affected test at every code change
  - Run tests on all major OS / browser combinations
  - Transparently store all build and test results (including build, code analysis, and linter warnings)
  - Provide comprehensive UI, API and notification
  - Move all "compute-intensive work" to the cloud

Key Goals and Principles

- Speed: Developers spend lesser and lesser time waiting on tools e.g. builds, test systems, code analysis, ...
- High Quality Feedback: Deliver high quality feedback; more signal, less noise.
- Simplicity: Developers will ideally not need to know or understand how the underlying tools and systems work.

Measure everything

Source code at scale ...

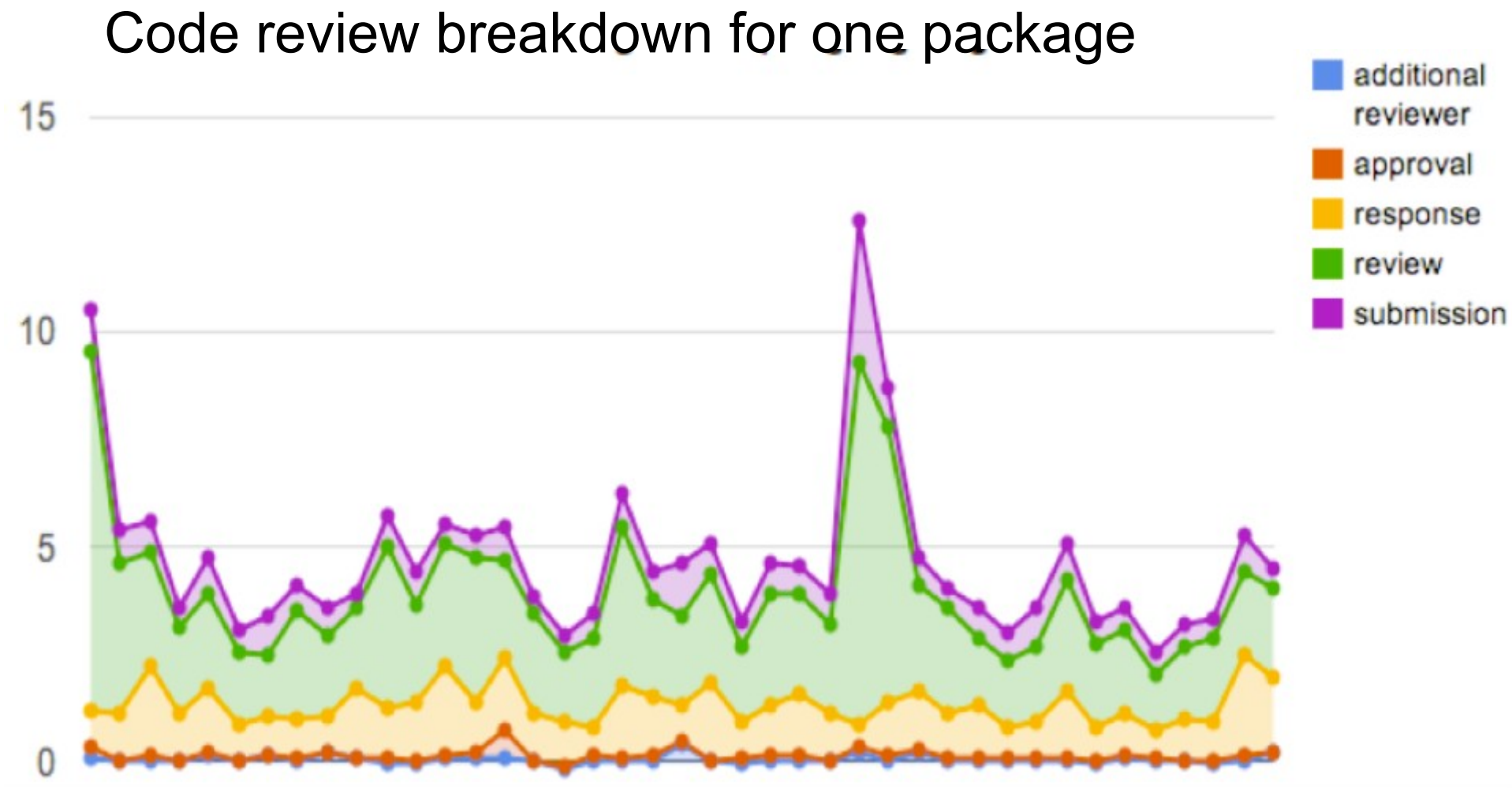
- How to allow 1000s of engineers to sync source code on a single tree with massive dependencies?
- A full checkout would take tens of minutes
  - Would easily choke any corporate network
  - Other companies create developer branches per feature
- Developers change < 10% of code they actually check out
  - Builds and tests often need the rest of the code to run
  - Deliver the rest of the code as a read-only copy, on demand
  - Implemented as a FUSE-based file system, tracks changes to main source depot and caches aggressively

Keeping the code tree consistent

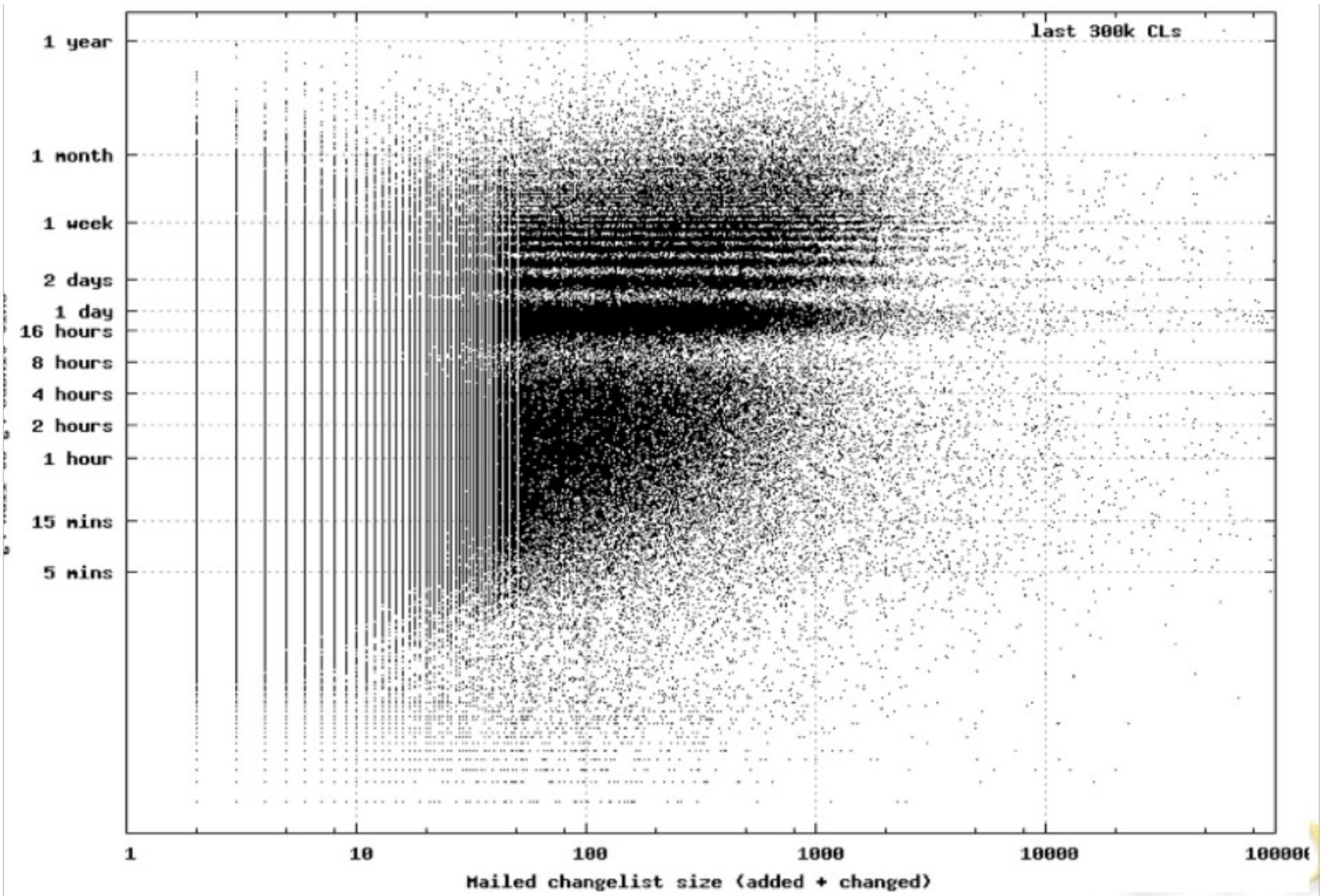
- Mandatory code reviews with central tool
  - Need code readability for languages (enforces style guide)
  - Need owners for code sub-tree that maintain consistency and correctness
  - Higher code transparency and code contributions across teams

- Reduce code review costs, provide lots of signals to reviewers
  - Lint errors
  - Code Analysis and Build warnings / errors
  - Code coverage data
  - Test results
  - Easy, web-based access - full graphical diffs available, easy to add comments
  - Future: integrate with IDEs

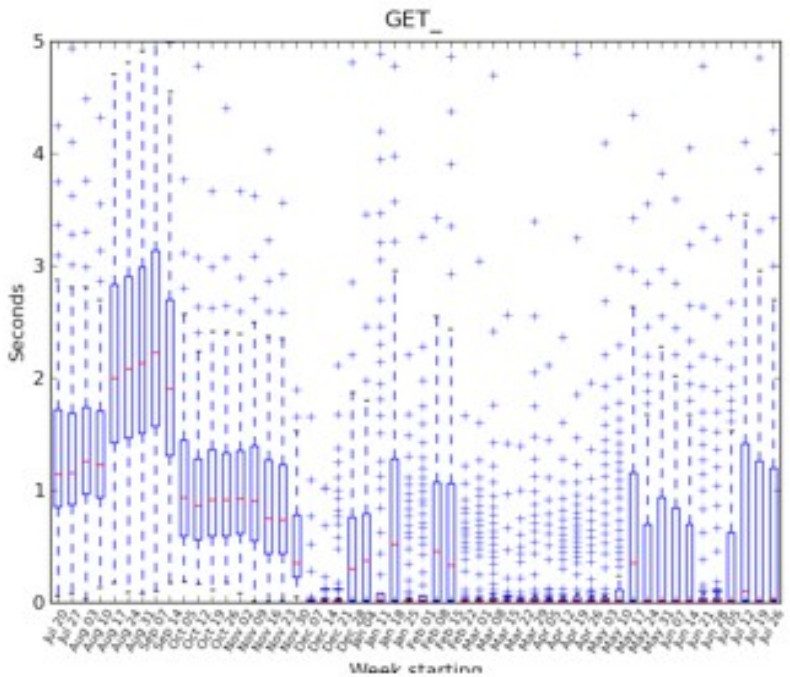
Keep code reviews efficient

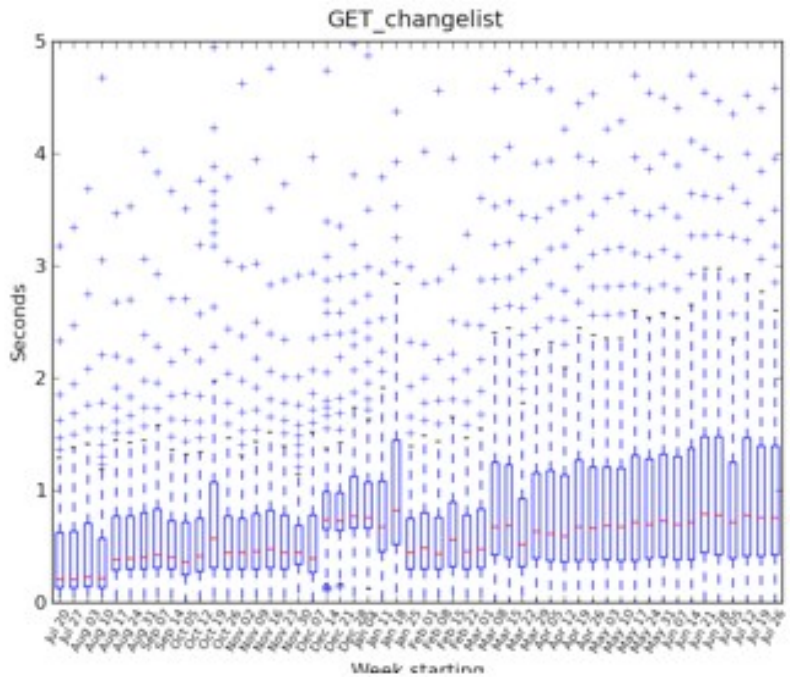


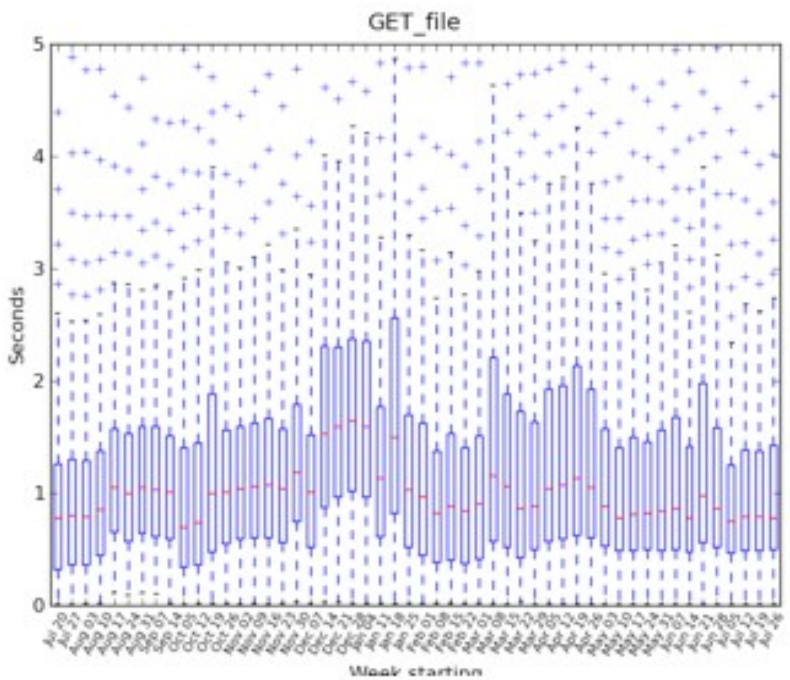
Code Review turnaround by size



Measure the tool itself







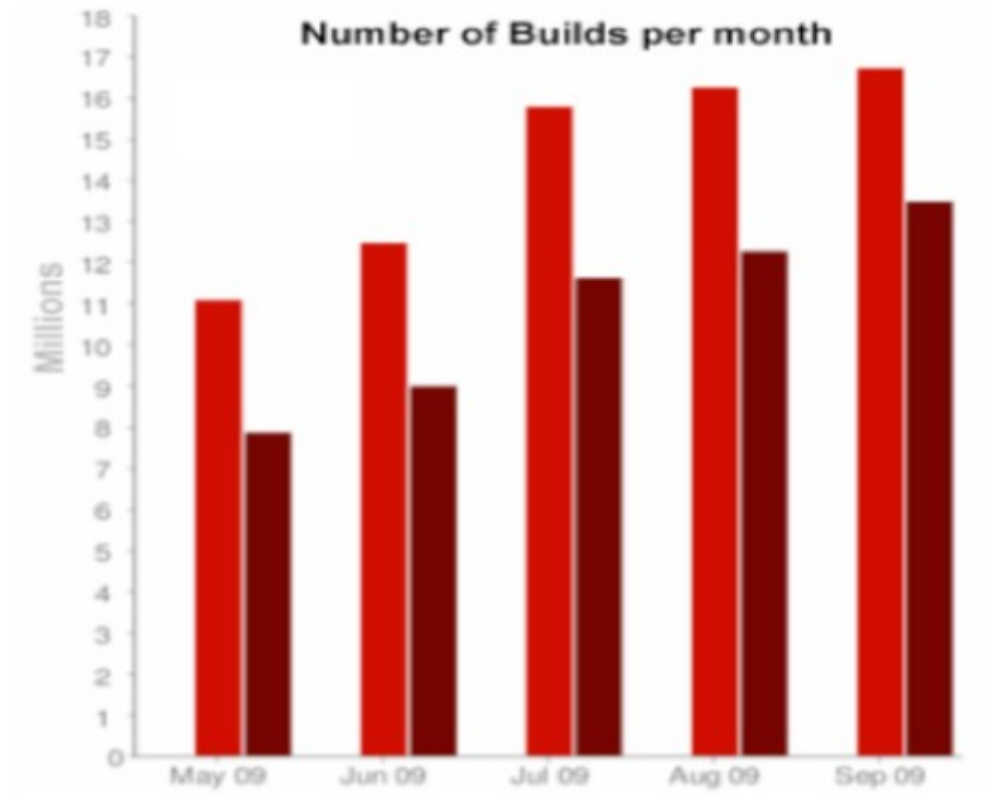
Box-plots for the Code Review tool latencies

The Build System is important

- Builds are glamour-less at most companies
- Problems with builds can result in huge productivity losses
  - Debugging build problems
  - Waiting for builds to finish
  - Feedback best attached to build systems; e.g. run tests, code analysis as part of builds
- Build metadata is equally important as source code
  - Needs to analyze and enforce dependencies, validate inputs
  - Needs to be correct and fast
  - Builds need to be hermetic to be distributed
  - Full knowledge of inputs, dependencies and outputs can allow massive parallelization of actions

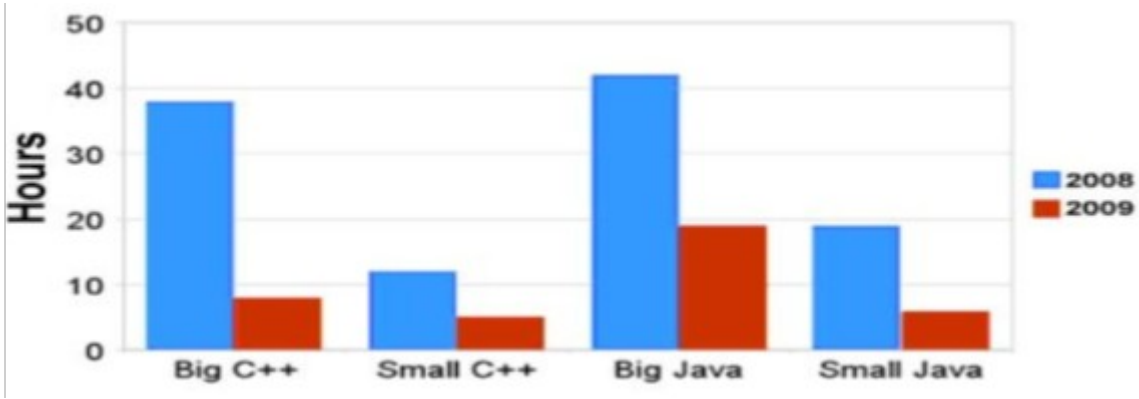
Build Systems require strong CS skills

- Deal with massive scale
  - 20 Million+ builds per year
- Massive distributed execution
  - More than 10000 cores using > 50TB of memory
  - ~1 PB 7-day cached object output



Durable metrics

- Remember this?

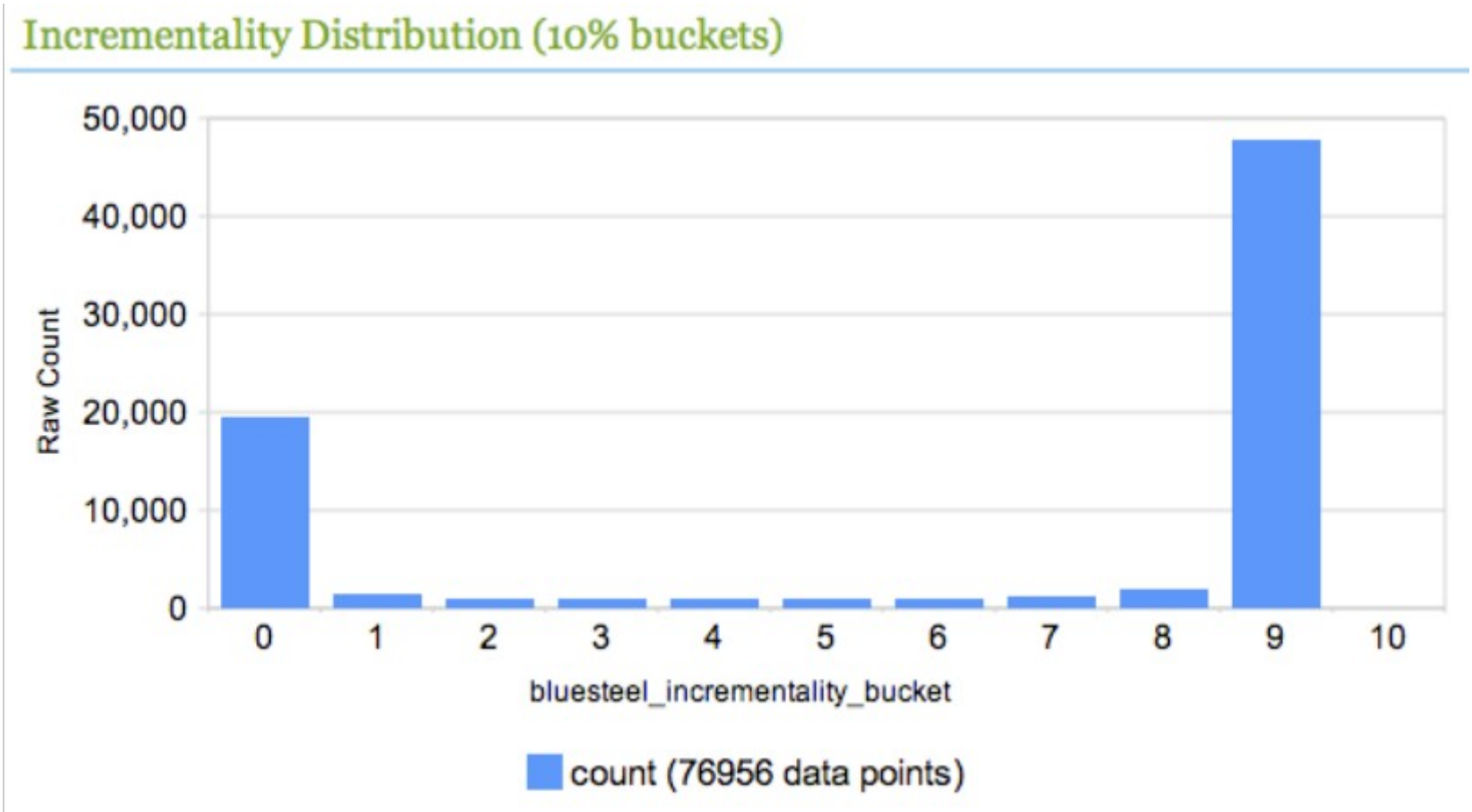


- Mostly flat between 2009 and 2010
  - Files for each (measured) target grew by 54% to 191%
  - Doing significant more work in the same time

- Needed durable metrics across time; bucket builds by:
  - Count of discrete actions and inputs
  - Office
  - Incrementality
  - ...

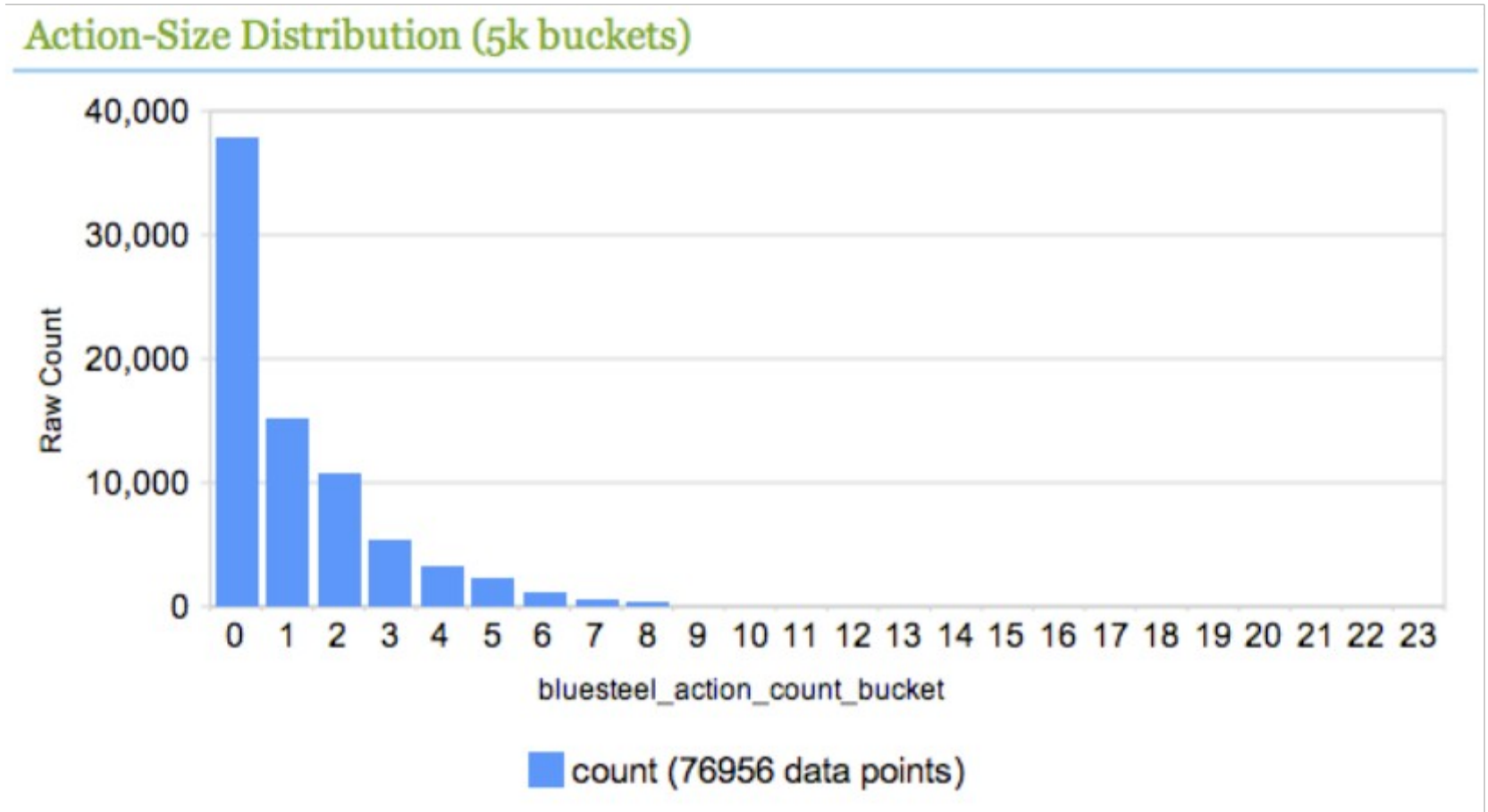
Builds by incrementality

- Many builds are clean, but most are in the 90-100% incrementality range!

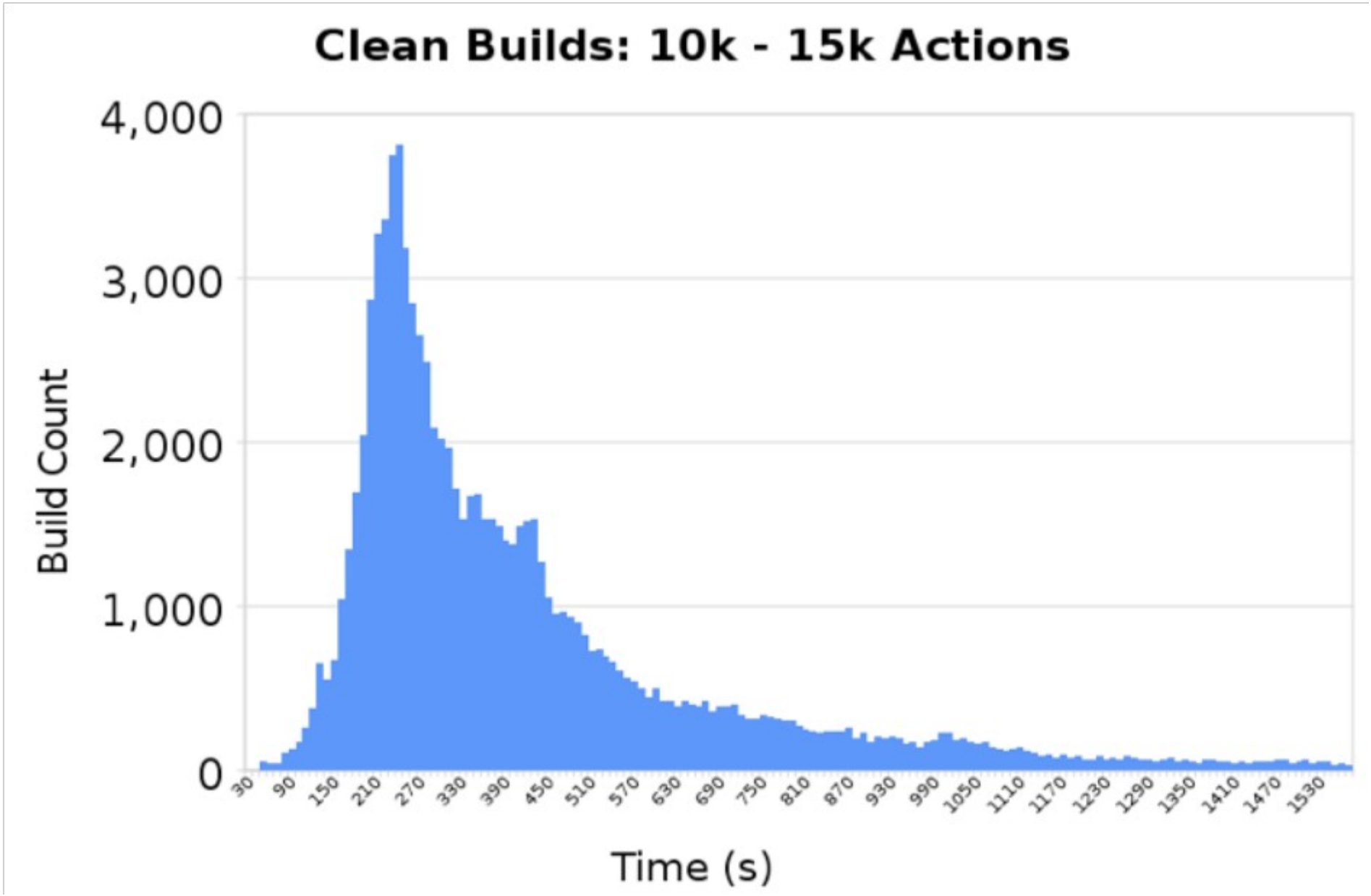


Builds by action size

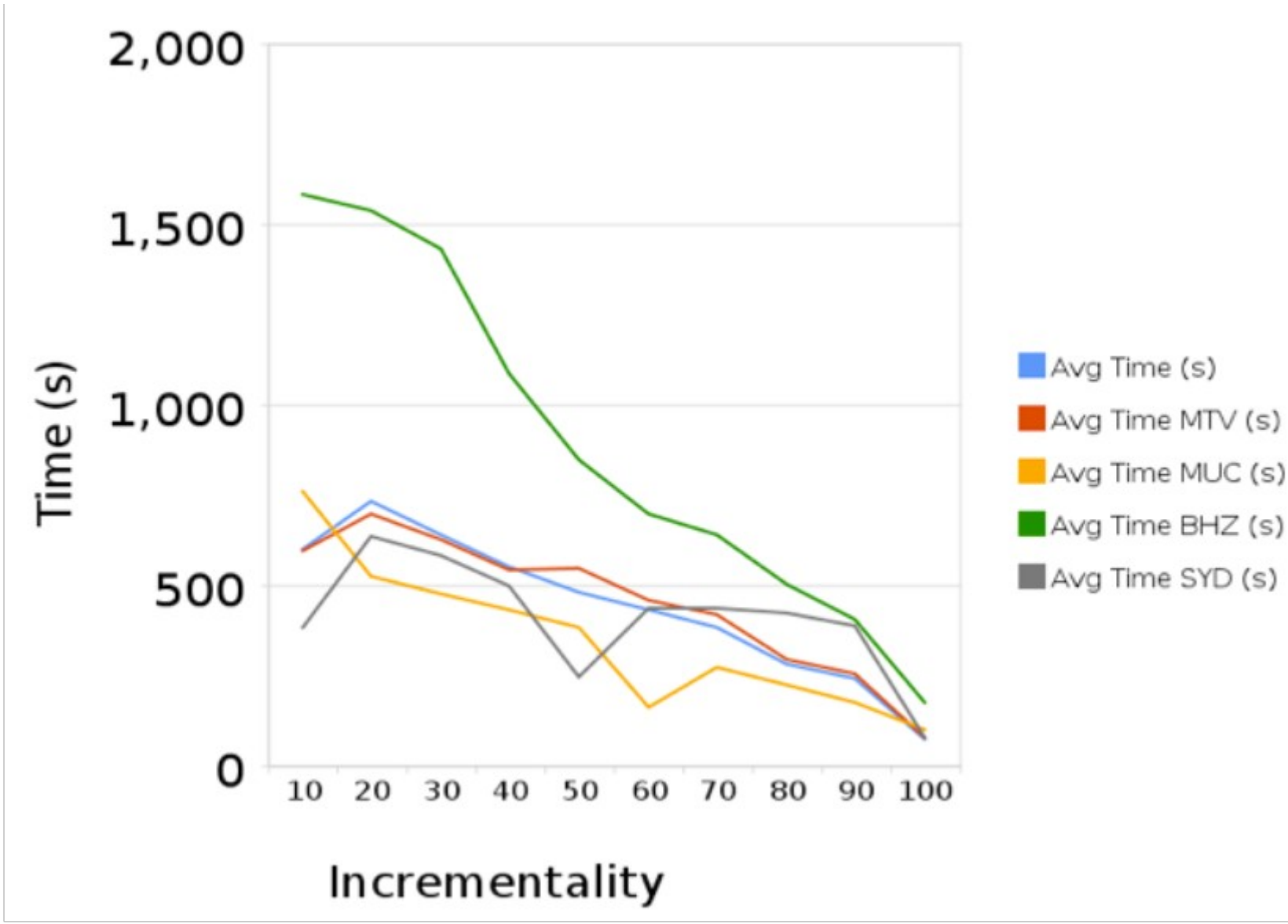
- Most builds are small, but long tail (mostly by our own automated systems)



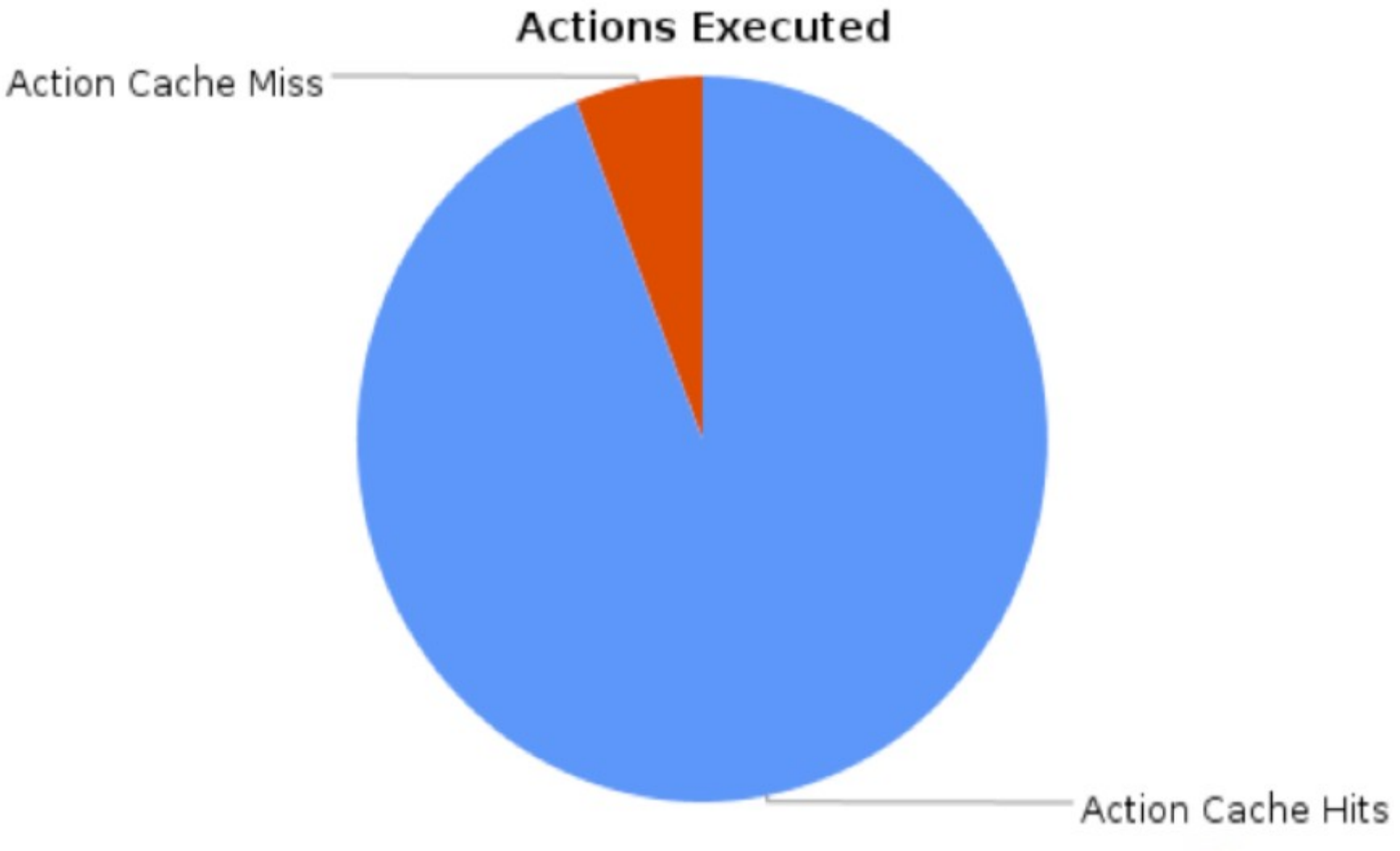
Clean Build times



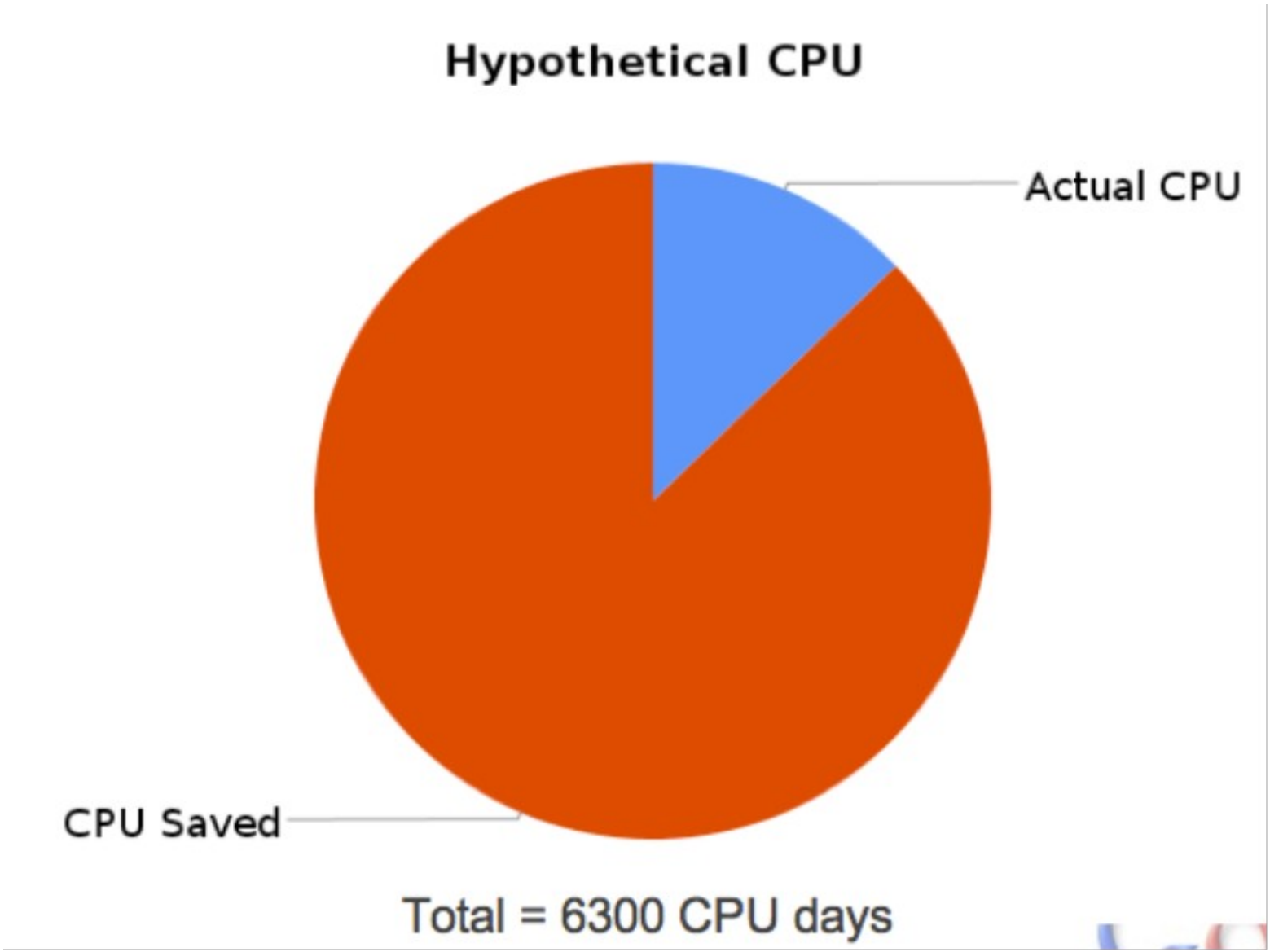
Build times by office



Action Cache



How much did we save?



Object caching wins

Statistics from a single day

- ~ 500M build actions
- 94% action cache hit rate
- 30M cache misses
- 800 CPU days (just build and test)
- 66% of actions from automated builds

Building in the cloud has costs ...

- Large builds have large outputs
- Corp-Cloud network is not as efficient as Cloud-Cloud network, transferring bits can be a significant time sink and network hog
- Solution: don't send the build outputs to the workstation till they are actually needed or read.
  - Implemented as a Fuse-based file system that allows directory operations on the output.
  - Aggressive caching for build outputs by office and workstation

Distributed builds have costs ...

- Link actions require all the input object files
  - Requires moving all object files that are built on different distributed nodes to the one node where the link action occurs
  - Can be expensive and on the critical path

- Solution: Incremental link
  - Store additional information in a binary
  - Use old binary + modified object files to build new binary
  - Only process modified object files symbol tables and relocations
  - expected 10x improvement in link speed

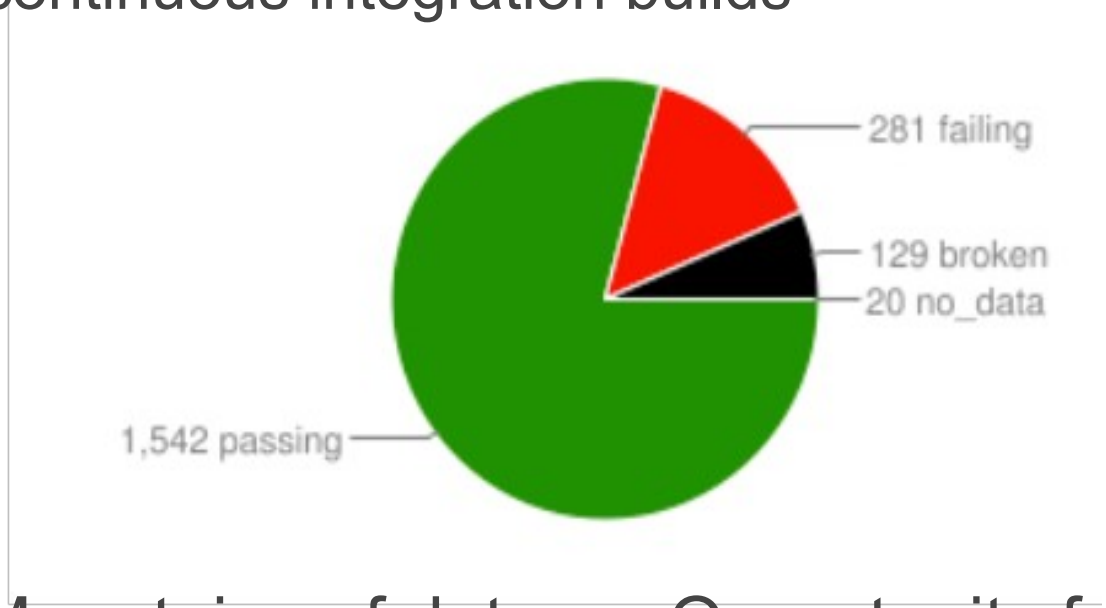
Continuous Integration at Scale

- Fail fast, report clearly, root cause
- Test early at every stage
- Reduce defect identification to fix time
- Use feedback and data to stay healthy
- Reduce complexity

"... the key is to practice
continual improvement
and think of (it) as a system,
not as bits and pieces." -
Dr. W. Edwards Deming

Continuous Integration at Scale

- 120K test suites in the code base
- Run 7.5M test suites per day
- 120M individual test cases / day and growing
- 1800+ continuous integration builds

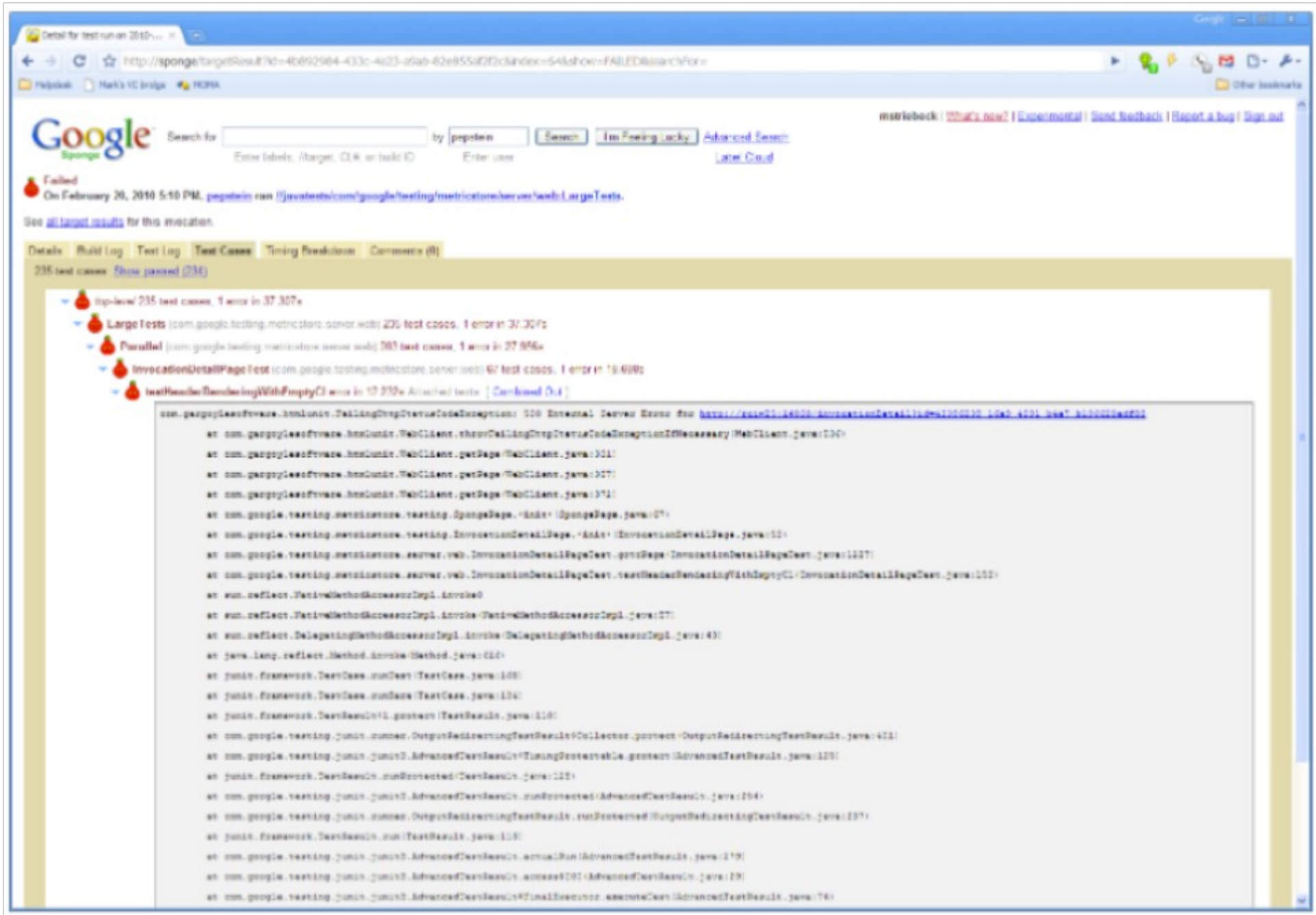


Mountains of data == Opportunity for data mining and research

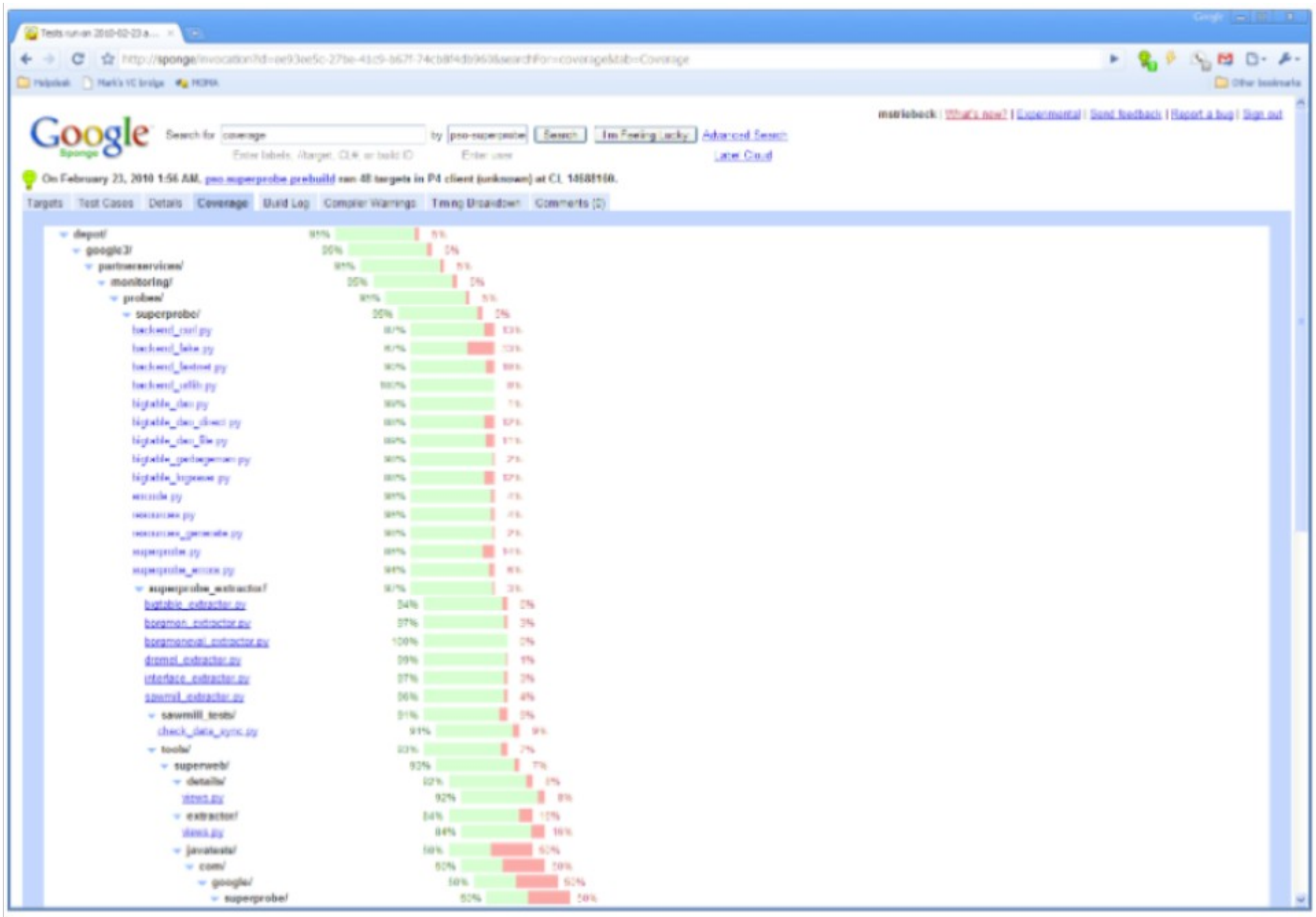
Scale requires Search

Also provides a SQL interface to query build and test results for further analysis

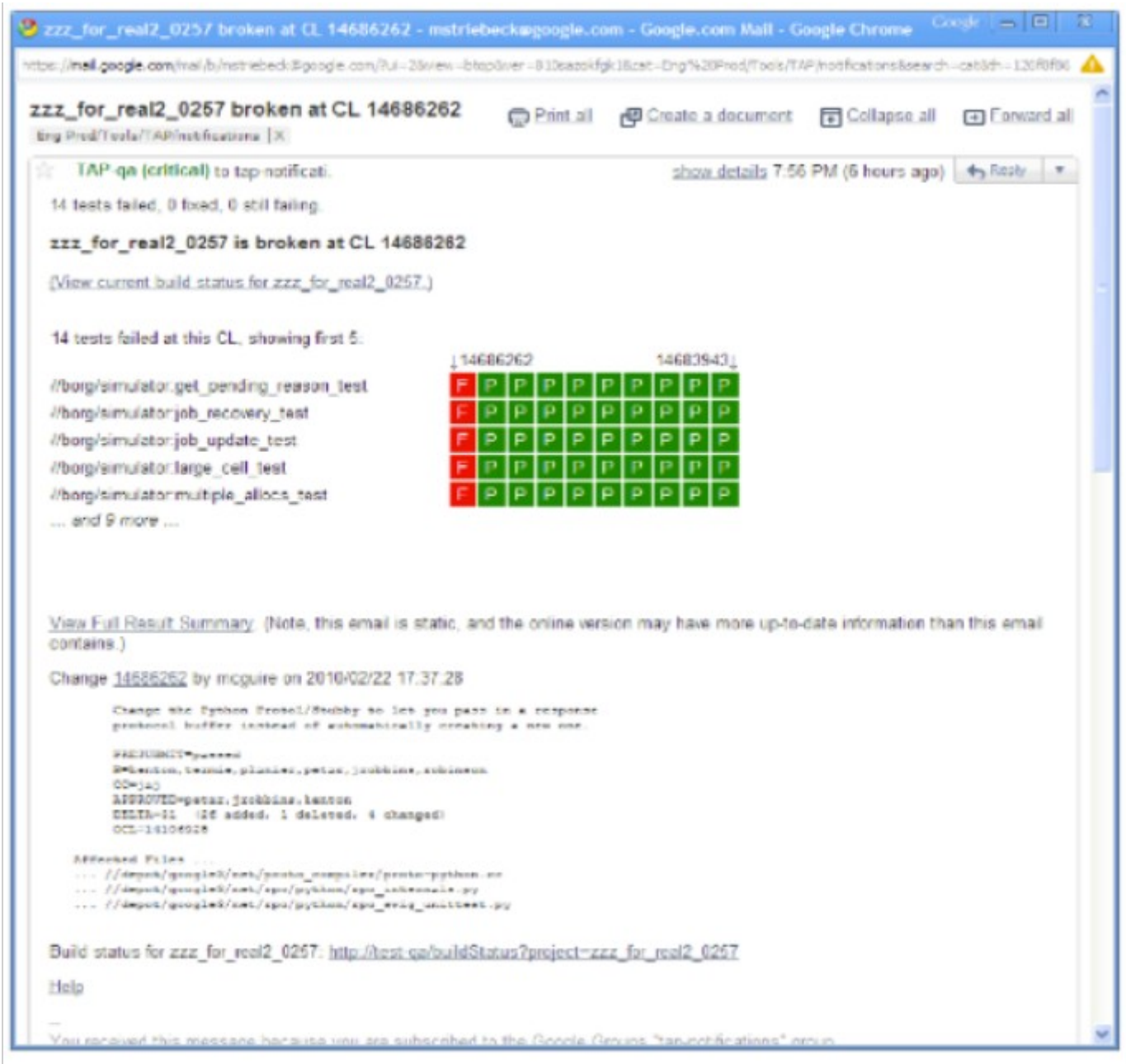
Test results repository



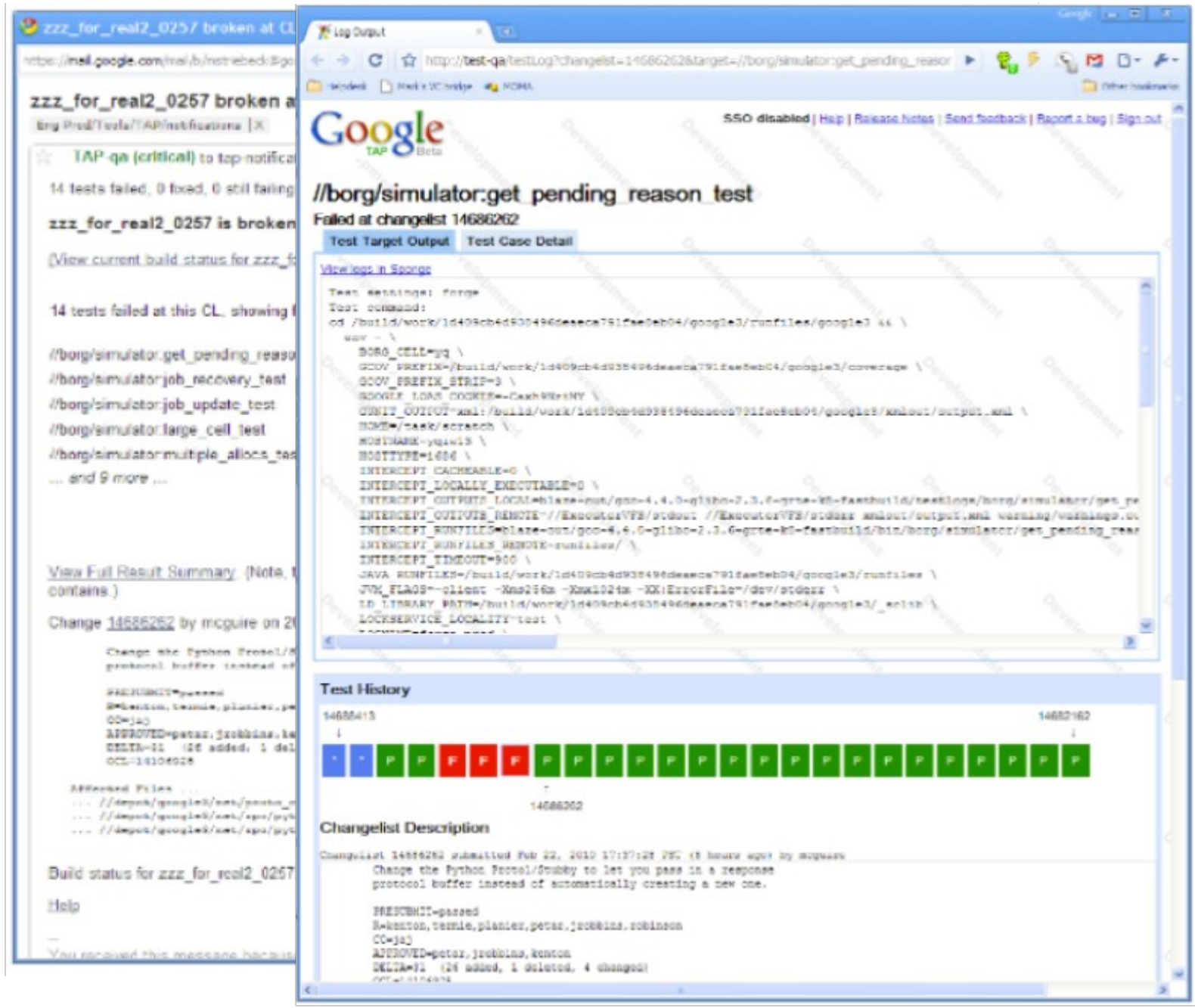
Integrated coverage view



Faster time to fix



Faster time to fix



And of course, we need more ...

- IDEs that can work at scale
- Code visualization and search
- Code Analysis and Documentation
- ... many more

Summary

What we do different

- Invest in our developer infrastructure
  - Developers can build upon common technologies
  - Significant investment in central tools team results in a measurable boost in engineer productivity

- Parallelize and Distribute where possible
  - Compute intensive operations leverage the cloud, while UI-sensitive work stays closer to the developer

- Hire the best / Design for scale
  - Developer Tools and Build Systems are tough computer science and systems problems; they need the best developers

- Measure Everything
  - Cannot improve what we don't measure