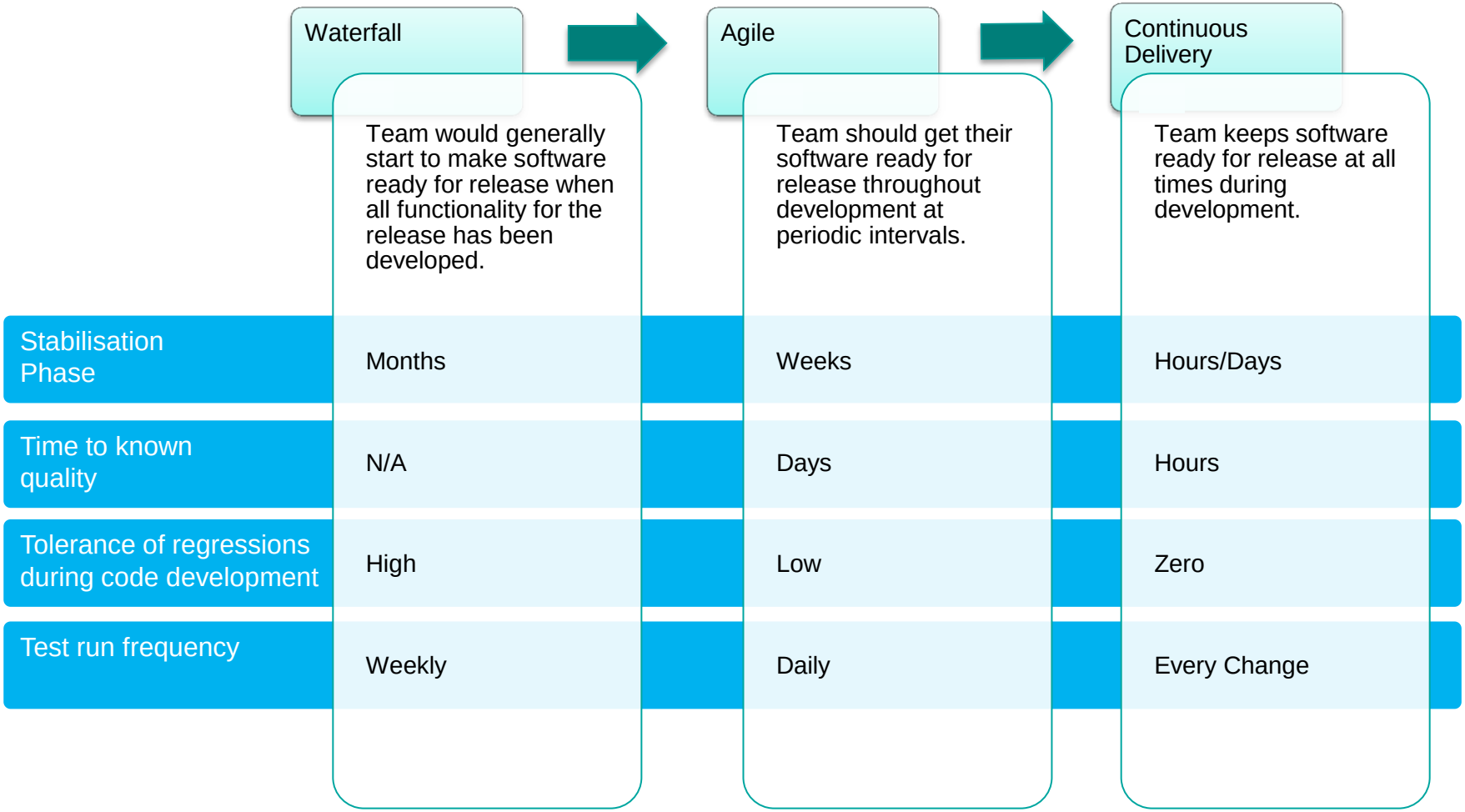
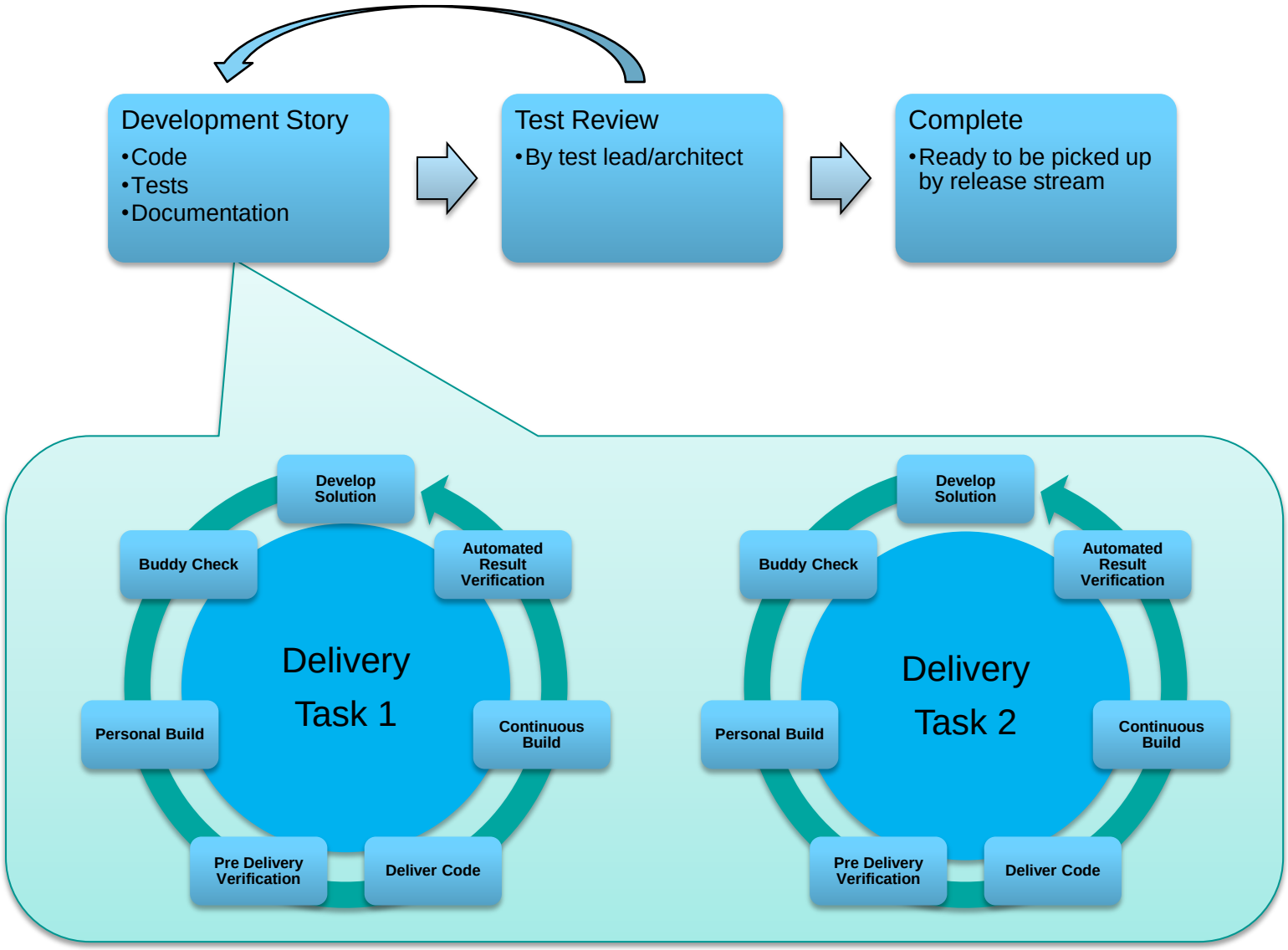
Building a better product with data analytics and the cloud

Kevin Smith (smithk6@uk.ibm.com)
Test Architect, WebSphere Application Server – Liberty Profile

The drive towards continuous delivery is driving changes in how we ensure quality



Continuous delivery lifecycle



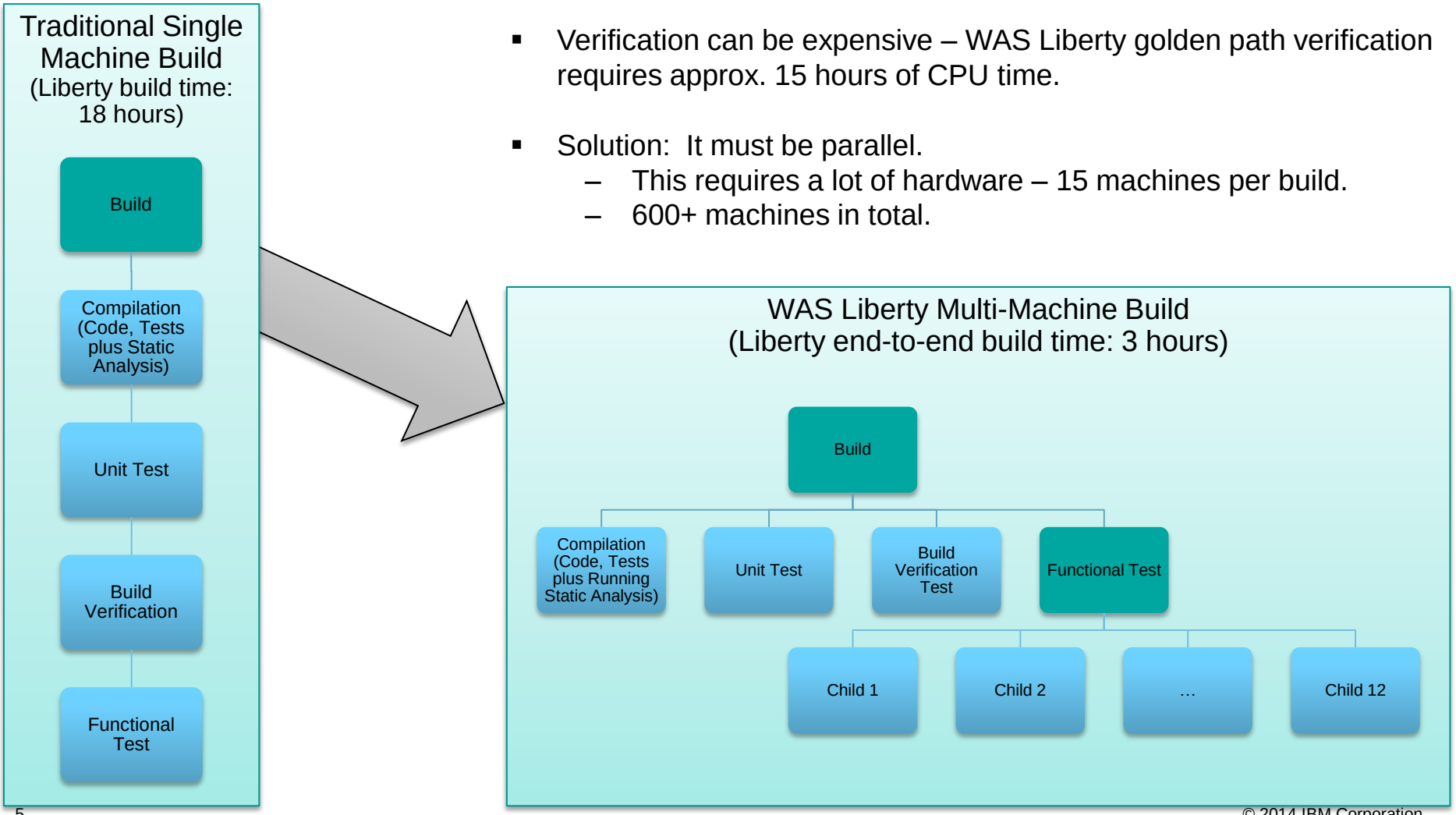
Different “builds” for different use cases

Build Type	Coverage	Hours of functional test per build	Frequency	Peak Daily Runs	Comment
Personal	Golden Path + Common Error Paths	15	On demand	200	Required prior to code delivery
Continuous	Golden Path + Common Error Paths	15	Every 2 hours	12	Verify new code deliveries to ensure there are no hard regressions.
Release	Golden Path + Common Error Paths	15	3 / day	3	Release Candidates Iterations can only close when release build has 100% of tests passing.
Full	Entire Corpus	36	2 / day	2	Initial sanity check of full corpus prior to launching SOE testing
Supported Operating Environment	Entire Corpus	36	1-2 / week	140	One SOE execution runs the entire corpus on 140 different OS/JDK platforms.
			Total	**357**	

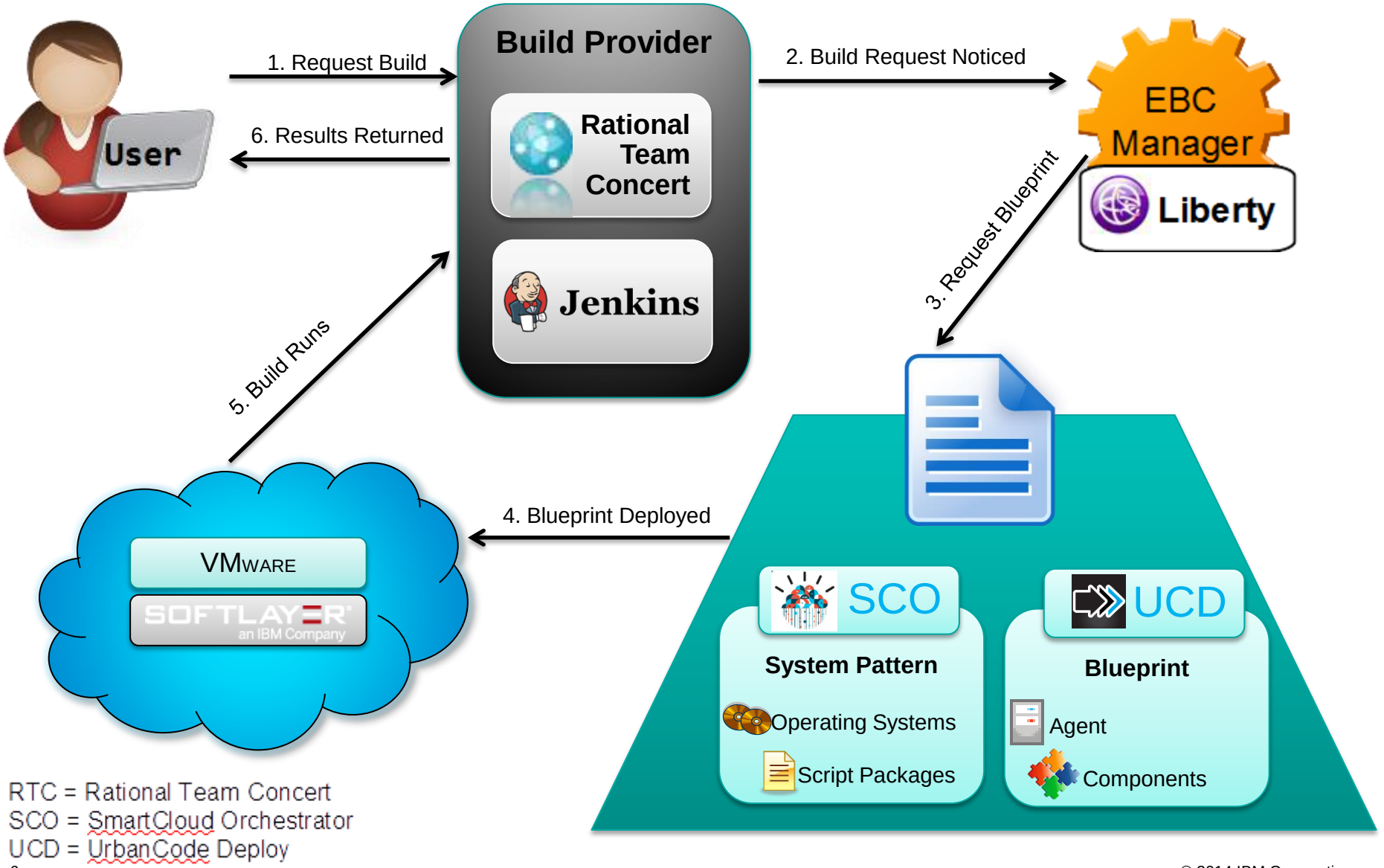
Note: 357 build per day equates to nearly a year of testing in a single 24 hour period.

Anatomy of a build

- In order to ensure quality at all times each build must verify there are no regressions.
- Verification can be expensive – WAS Liberty golden path verification requires approx. 15 hours of CPU time.
- Solution: It must be parallel.
  - This requires a lot of hardware – 15 machines per build.
  - 600+ machines in total.



Introducing the Elastic Build Cloud



RTC = Rational Team Concert
SCO = SmartCloud Orchestrator
UCD = UrbanCode Deploy

Scaling the Elastic Build Cloud – Liberty EBC in numbers

24 hours in the Liberty EBC

6 Blueprints

6 * OS / UCD agent / Jazz Build Engine / Liberty Build Scripts

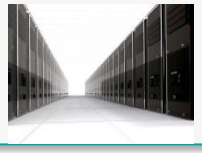
Coming soon: More platforms, Jenkins, complex topologies.

35 Definitions

25% of Liberty cross platform testing.

Target: 100% of cross platform testing plus developer builds.

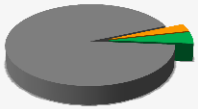
200 Systems



Concurrently each with 2 vCPUs, 6GB RAM, 80GB Disk on one x86 hypervisor with 10 nodes.

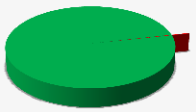
Goal: 600 x86, 400 PPC, 150 zVM + static pools.

420 Jobs



Provision:	**4 minutes**	**Total execution time:**	**1260 hours (52 Days)**
Teardown:	**3 minutes**		
Job Execution:	**180 minutes**	**Concurrent provisions:**	**50**

>2,000,000 Tests



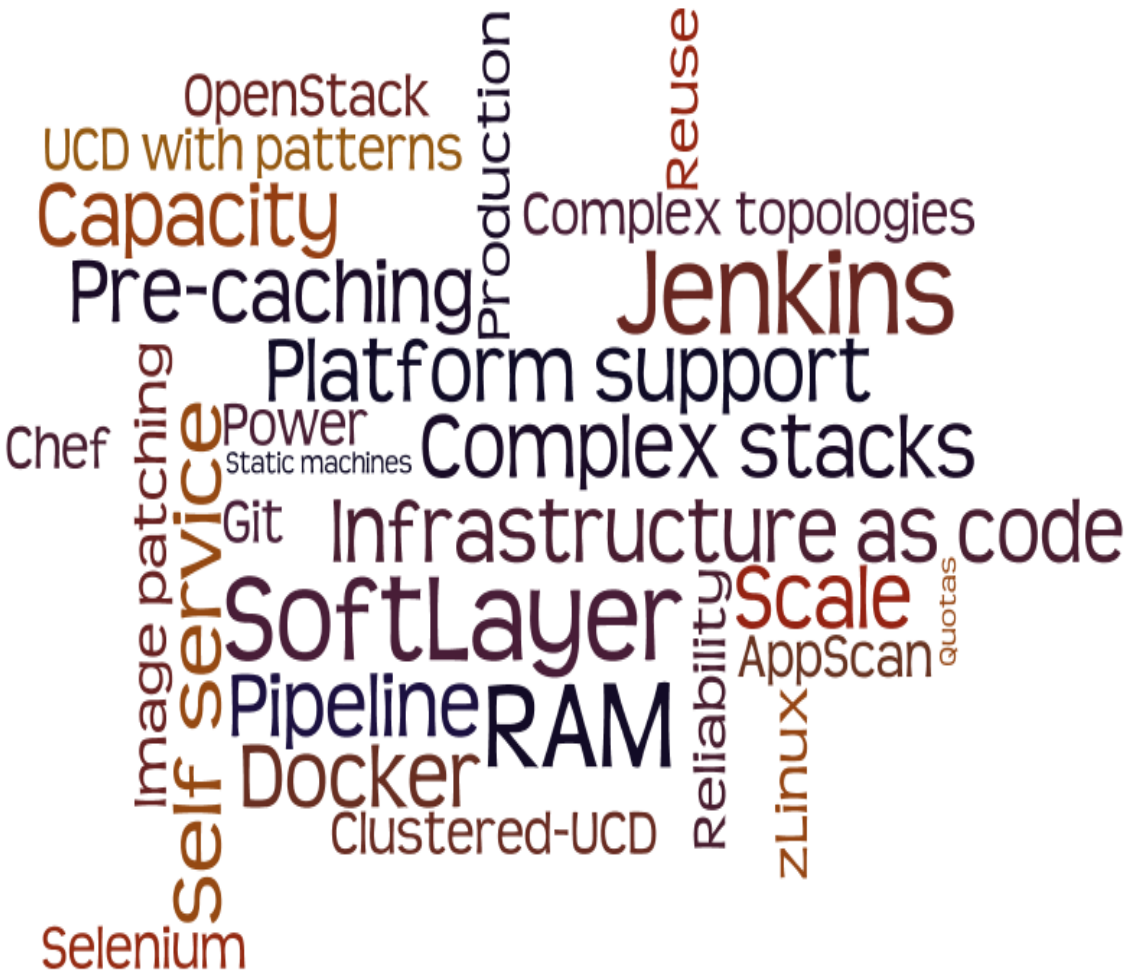
99.8% Pass

0.2% Fail/Error

7070 test execution buckets

~250 GB of test artifacts

Future for the Elastic Build Cloud



Why does Liberty need to perform test analytics?

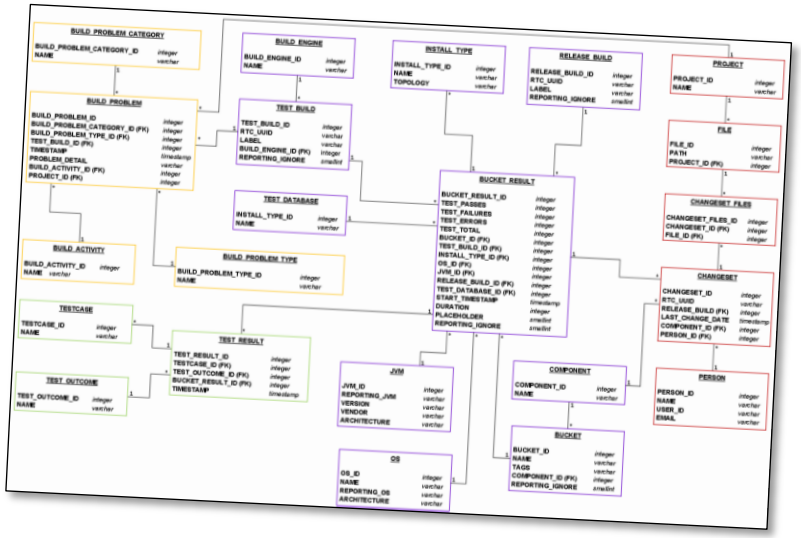
- **Liberty generates a lot of test data…**
  - On a busy day we can expect to run over 300 builds with each executing between 15 -36 hours of testing.
  - The Supported Operating Environments (SOE) builds cover ~140 platforms (OS/JDK combinations).
  - We are approaching running 1 year of testing in 24 hours!
- **The test data we generate is complex…**
  - Each failure may have a different cause requiring analysis.
  - Liberty is complex and thus so are its tests.
- **Analysis of the test data is expensive…**
  - We expect test failures to be investigated as quickly as possible.
  - Investigation is often a manual process.
- **We need fast feedback on regressions…**
  - The faster we find issues, the cheaper it is to fix.
  - Short release cycles mean limited time to identify and fix issues.

What data to we collect and how do we collect it?

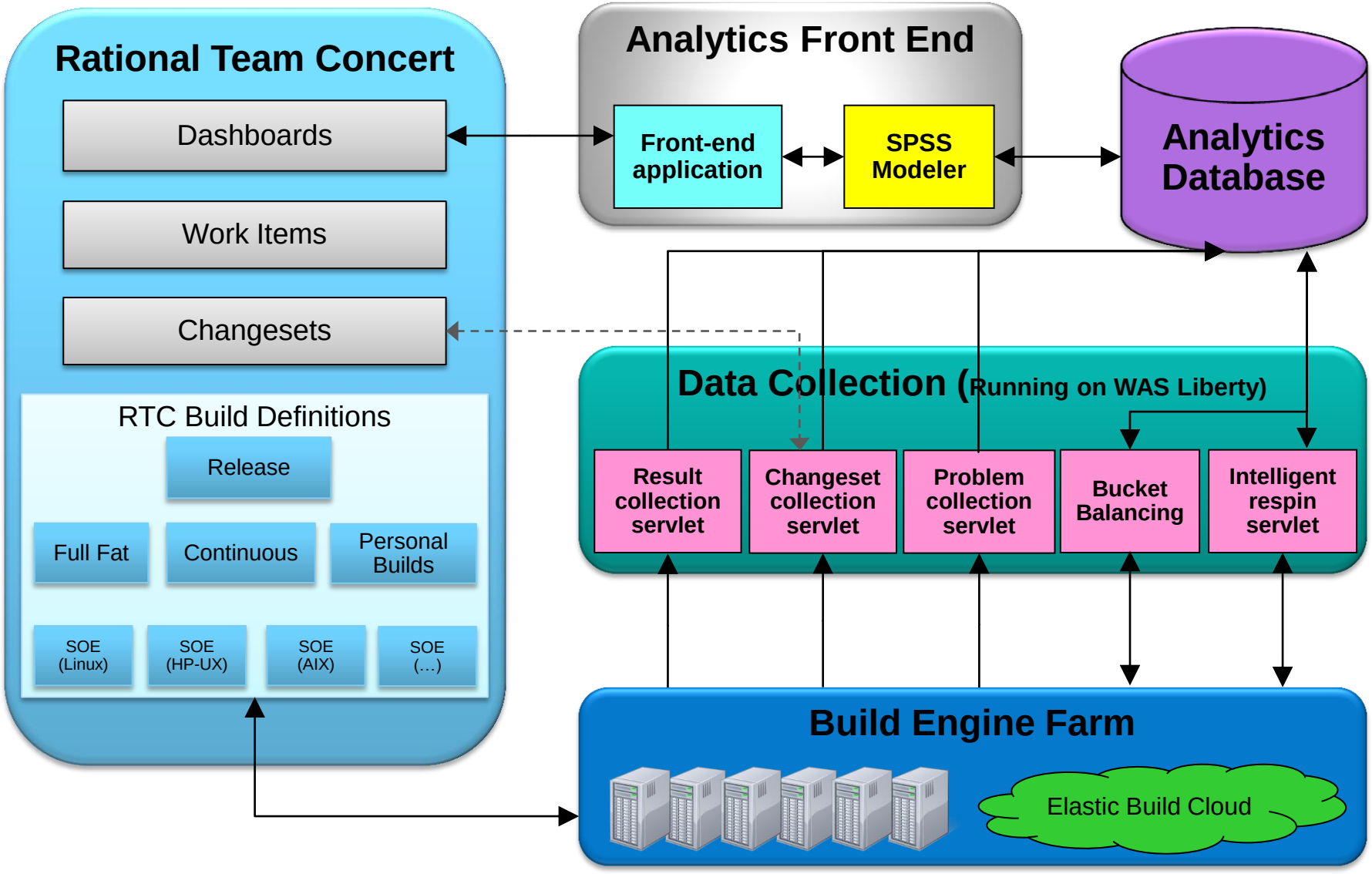
- **The key element of data we collect is a *bucket result*: the outcome of running a group of related tests.**

- **For a *bucket result* we collect (an abridged list):**
  - The *bucket* which ran and the number of its tests which passed, failed or encountered an error.
  - Details of every change made in the code base since we last ran:
    - What changed.
    - Who changed it and when.
  - Details of the environment we executed on.

- **Additionally we collect details of what execution issues were encountered (infrastructure failures).**

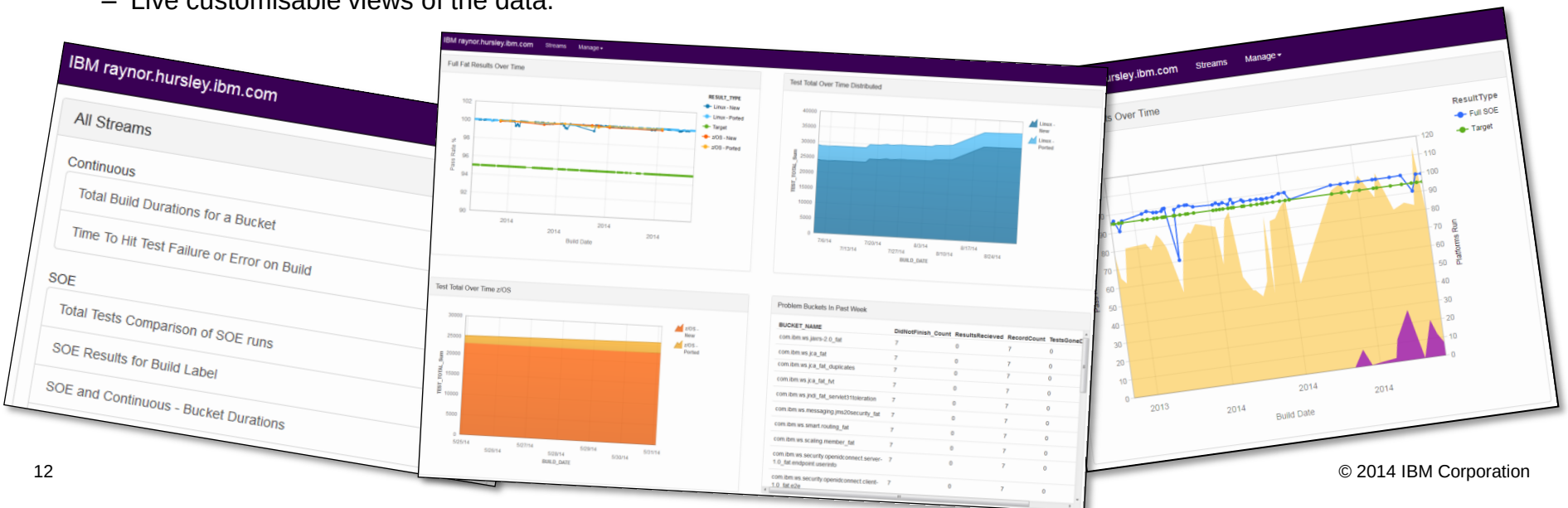


Analytics infrastructure

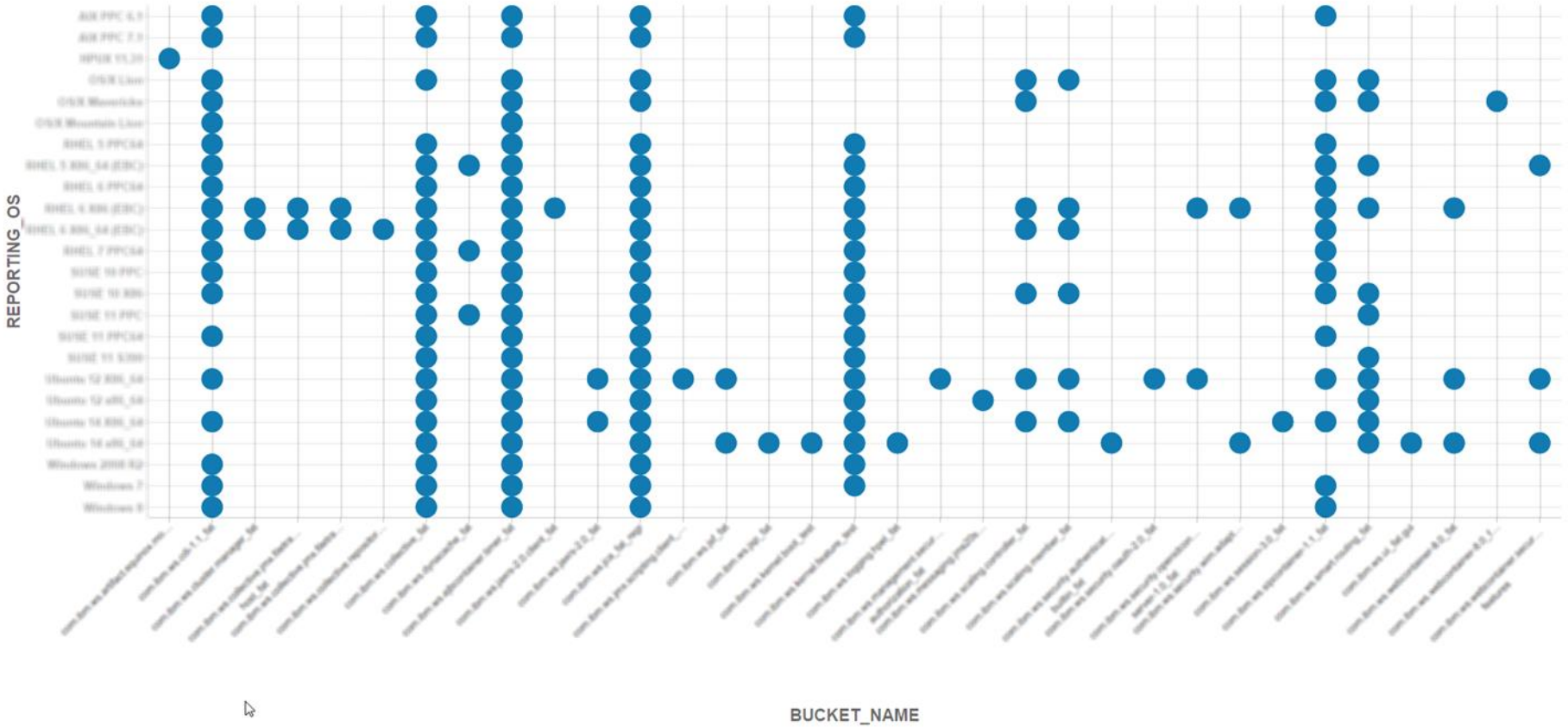


How are we using analytics?

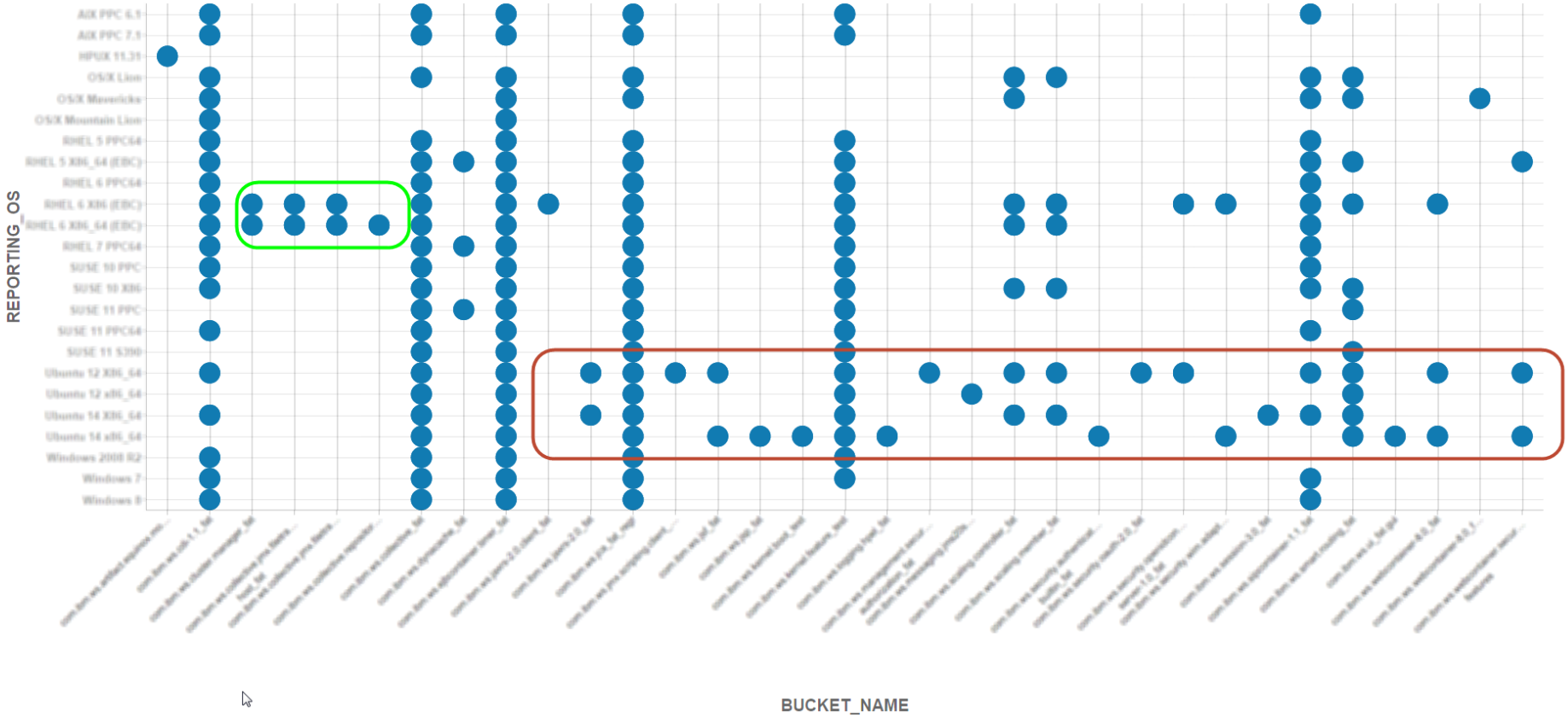
- **Automated reporting:**
  - Generate real time view of our quality:
    - Trends over time.
    - Complete view of quality for single build of the product.
  - Self-service frontend application running on Liberty which allows anyone to view/create reports.
  - Integration with Rational Team Concert.
- **The benefits of automated reporting:**
  - Analytics has allowed us to explore our data and produce visualizations we hadn't previously considered.
  - Live customisable views of the data.



How are we using analytics?

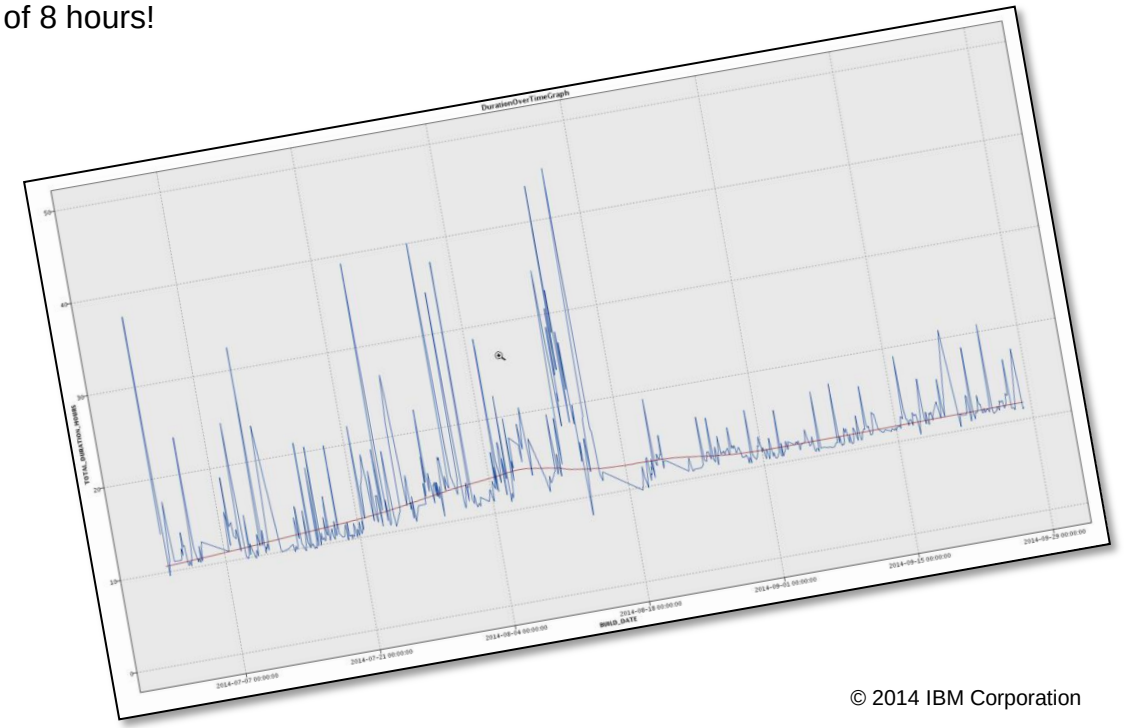


How are we using analytics?



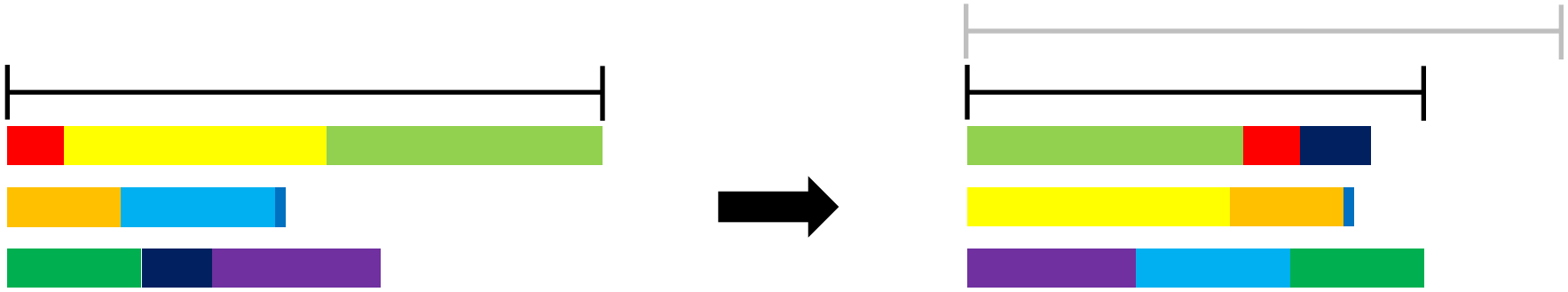
How are we using analytics?

- **Advanced interrogation of our data:**
  - Analytics has allowed us to quickly and easily interrogate our data by creating new streams.
- **A recent example:**
  - Our builds were taking longer than normal to run, causing build queues, intermittent failures and slowing development.
  - We used analytics to identify a relationship between build times and the virtualized hardware running the build.
  - This entire process had a turn-around time of 8 hours!
  - 25% reduction in execution time.
  - 150% increase in “green” builds.



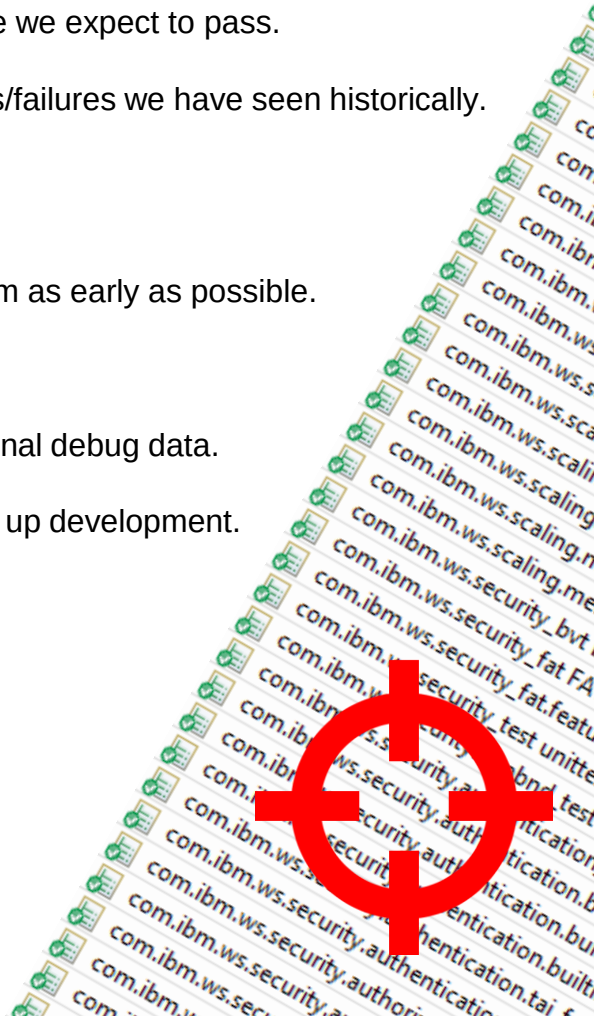
How are we using analytics?

- **Intelligent bucket distribution – Using data to dynamically change our build process:**
  - The majority of our builds run our test buckets asynchronously in a number of child builds (usually 15).
  - A build does not finish until all of its child builds have finished.
  - The buckets all take varying amounts of time to run.
  - We previously distributed the buckets amongst the child builds via a naïve round-robin algorithm.
  - The build now calls out to the analytics which uses historical data to bin the buckets and returns an optimal distribution.
- **The benefits of intelligent bucket distribution:**
  - End-to-end build time reduced by 20%.



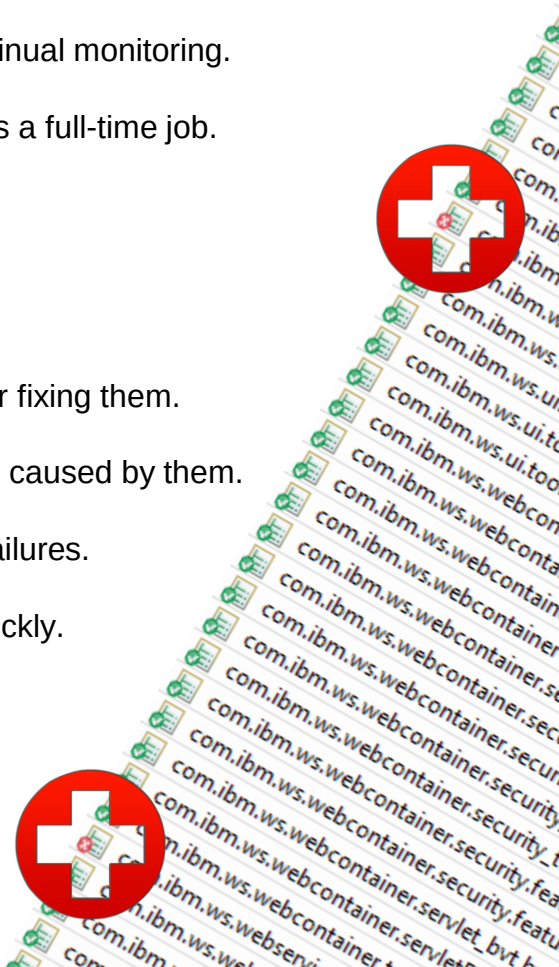
What's next for our analytics

- **Targeted testing (extending the intelligent bucket distribution):**
  - The earlier we identify a test failure the earlier we can begin to triage it and fix it.
  - We are working to use analytics to prioritize buckets we expect to fail before those we expect to pass.
  - We can do this by examining the changes in the build comparing them to changes/failures we have seen historically.
- **The benefits of targeted testing:**
  - By identifying and triaging test failures as early as possible we can start fixing them as early as possible.
  - We may opt to run buckets with high risk of failure multiple times to be thorough.
  - We may opt to run buckets which fail intermittently multiple times to gather additional debug data.
  - It may eventually be possible to omit low-risk buckets in personal builds to speed up development.



What's next for our analytics

- **Auto-Triage:**
  - Test failures are currently triaged manually by build monitors.
  - Our developers take turns at being build monitors for 1 iteration (2 weeks) at a time.
  - We have 3 build monitors working full time in 3 different geographies to provide continual monitoring.
  - Identifying known test failures and raising defects is currently a manual process - it is a full-time job.
- **The benefits of auto-triage:**
  - We can automate the more mundane build monitoring tasks such as raising defects.
  - This frees up build monitors to dedicate their time to investigating complex failures or fixing them.
  - As we collect data on infrastructure issues as they occur we can identify test failures caused by them.
  - Analytics can provide additional insight into test failures and identify trends among failures.
  - By raising defects quickly (and including more information) we can fix them more quickly.
  - We can potentially verify defects automatically when they are fixed.



Questions