ONTAP Continuous Integration/Testing

How a change becomes a product

Phil Ezolt
Netapp MTS 6 AERO/DevOps
FreeBSD Developer Summit, 5/15/19

Agenda:

- ONTAP Background
- How do we keep it working?
- Life of a Change
- Pre-submission Workflow
- Post-submission Workflow (tier 1)
- Post-submission Workflow (tier 2)
- Post-submission Workflow (coverage)
- Does it work?

ONTAP

- What is ONTAP?
  - Data Management Software: Provides fast & reliable access to data
  - Built-in storage efficiencies: snapshots, dedup
  - Access your data: NFS/SAN/CIFS/more.
  - Manage your data: GUI or CLI or Zapi (XML) or REST
  - Protect your data: replication & encryption
  - Runs on clustered Netapp filers, in VMs, or in the cloud

- ONTAP feature set is huge, this does a better job explaining it:
  - https://www.netapp.com/us/products/data-management-software/ontap.aspx
  - Netapp has been making ONTAP for 20+ years

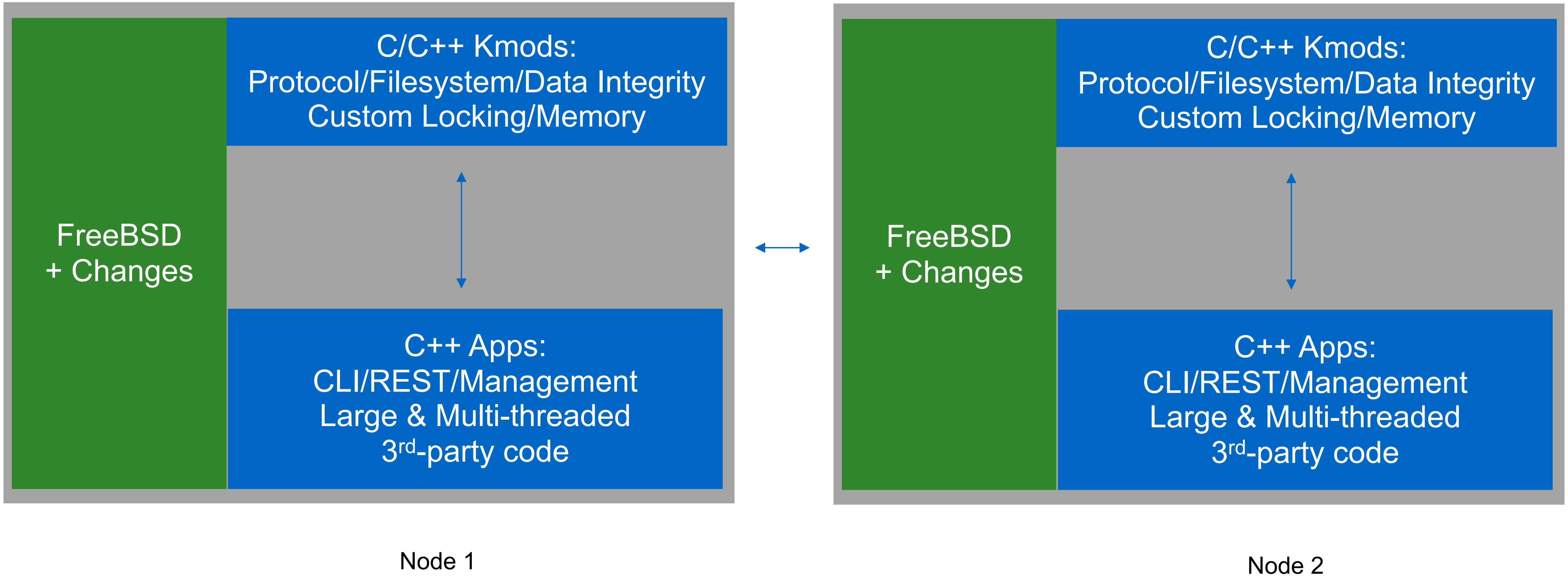
NetApp

Why is shipping ONTAP hard?

- Diverse codebase
  - >10 millions lines of executable code (Not counting some 3rd party code)
  - Kernel & User code running in FreeBSD
  - C/C++ for product, python/perl for test code.
  - Significant 3rd-party/opensource footprint

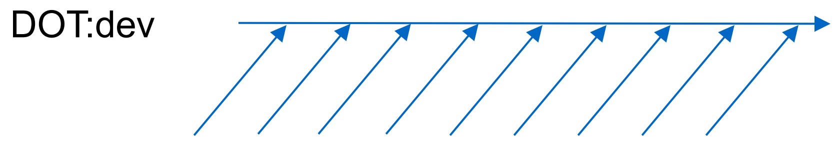
- Constant change
  - 20+ year-old code base
  - >1000 developers
  - High churn -> 38k changes submitted in 2018
  - Subtle interactions -> Changes to Feature A can break Feature B.

ONTAP complexity: Many Moving Parts


FreeBSD + Changes
C/C++ Kmods: Protocol/Filesystem/Data Integrity Custom Locking/Memory
C++ Apps: CLI/REST/Management Large & Multi-threaded 3rd-party code
Node 1
FreeBSD + Changes
C/C++ Kmods: Protocol/Filesystem/Data Integrity Custom Locking/Memory
C++ Apps: CLI/REST/Management Large & Multi-threaded 3rd-party code
Node 2

How do we keep it working?

- ONTAP uses Continuous Integration
  - All dev submits to a single perforce depot:
    - DOT:dev -> Master development branch
    - No feature branches
    - Submit risky content disabled (dark)
  - monorepo (ish)
    - Contains 3rd-party code/FreeBSD/ONTAP code/unit-tests/build-scripts/test-code/test-tools
    - Can build from scratch, but most devs use incremental builds.
    - Managed by internal build system (bedrock)… One command can build everything.
    - Fully built workspaces snapshotted and available as a flexclone (~30 secs for a full client)
    - Every file has an owner



How do we keep it working?

- ONTAP uses Continuous Integration
  - Philosophy: All regressions are reverted
    - Tests MUST always pass… Check-in test change with code change.
    - Code aggressively checked in, but aggressively reverted out.
      - ~2% of all changes are reverted
    - Lock-line if stability not achieved after 24 hours
  - Philosophy: Put the eggs in one basket
    - Focus testing in one branch
    - Focus triage in one branch
    - Focus resource use on one branch
    - Unify test environment and reporting
    - Do it one way… but do it well.

NetApp

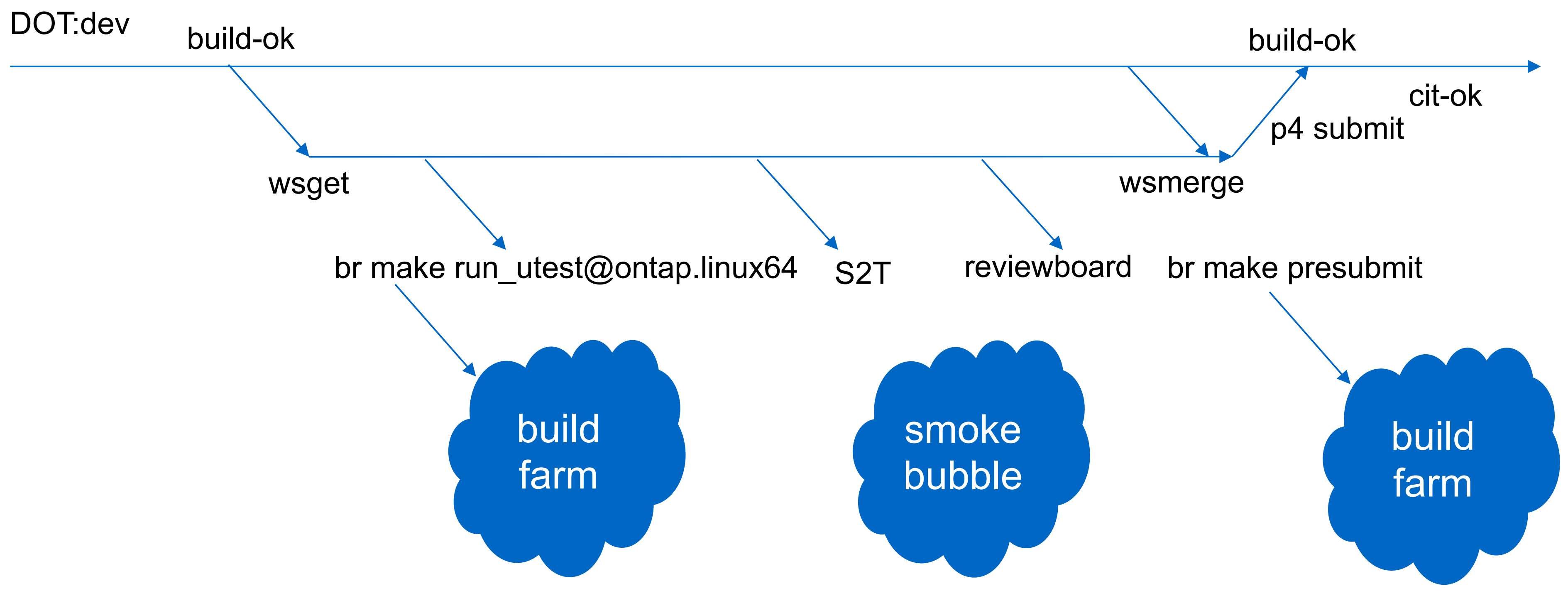
How do we keep it working?

- Known good points:
  - build-ok -> Change successfully builds most variants and passes in-build unit-tests
    - In-build unit-tests -> 28k CxxTest based ONTAP test-cases

  - cit-ok -> Change successfully passes all ~120 Continuous Integration Tests (CITs)
    - CITs -> Run for 2-hours, typically end-to-end ONTAP testing on VMs (vsims)

- Workspace (Client) pre-submission requirements
  - ‘br make presubmit’ -> build most variants, run all unit-tests
  - Source-2-test (S2T) -> run 6 CITs based on pending changes
  - Reviewboard

DOT:dev - Life of a Change



How do we keep it working?

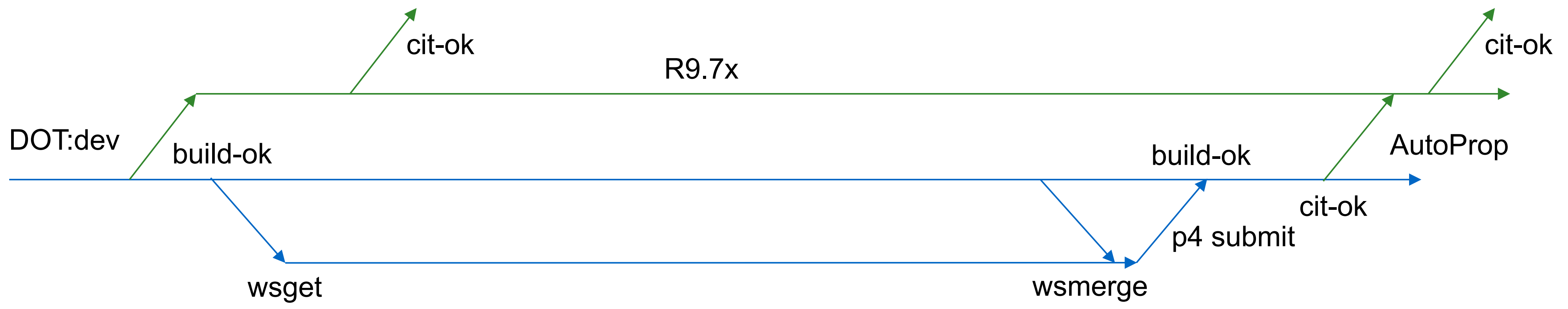
- Release branches hang off of DOT:dev
  - Release testing: focus on DOT:dev as long as possible

  - Release fixes submitted to DOT:dev first
    - We can tolerate more risk in the development branch.
    - DOT:dev is often more strict (because quality gates show up there first.)
    - Changes that pass everything in DOT:dev can be pulled back.

  - Every change: May request propagation to release branches.
    - Hit cit-ok -> individual changes are automatically propagated back (auto-prop)
    - Any future reverts of those changes are ALSO auto-proped back.

  - Release branches run CITs as well, but at a reduced cadence.

DOT:dev - Life of a Change (Release)

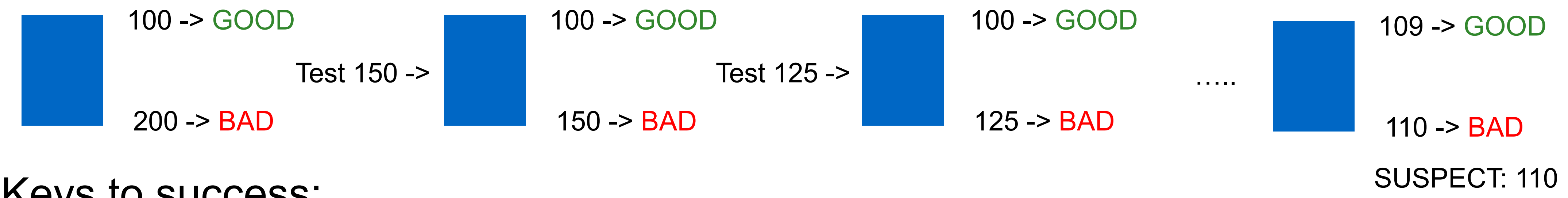


Bisect & Autoheal

- How do we keep it working?
  - Give developers known good workspaces. (build-ok, cit-ok)
  - Run builds (20m) and CITs (3 hour) on cadence.
  - Automatically find bad changes, and revert them from the line.
    - Bisect -> Find first change that broke it
    - Autoheal -> Apply ‘p4 undo’ to bad change, validate, submit
- Autoheal fundamental to maintaining + improving quality
  - Protected areas called ‘autoheal-layer’
  - Autoheal-layer enables quality ratcheting
    - Add tool/test/sanitizer to autoheal layer, autoheal keeps it clean

Bisect (details)

- Bisect:
  1. Run specific build/test on cadence.
  2. When cadence fails, record the last known good change & first failure change
  3. Pick a change in-between… see if it passes.
  4. Update last-good/first-bad. Go to 3 until we've identified the SUSPECT change that causes a failure.

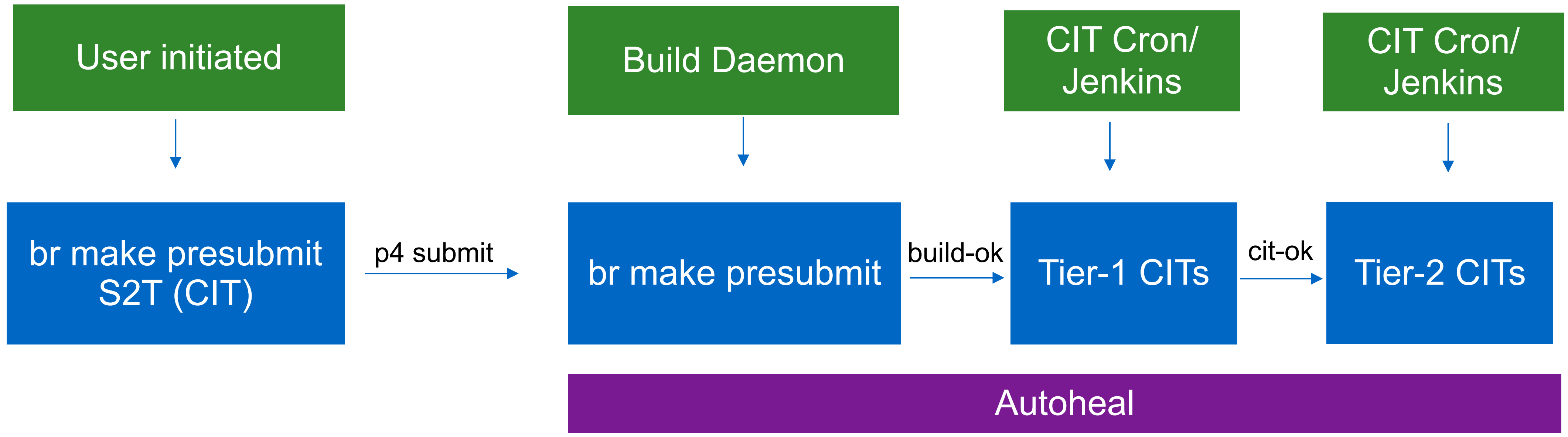


- Keys to success:
  - Minimize external dependencies… Or version them by a p4 change.
    - The same change should fail today and next week
  - Run multiple tests in parallel.
  - Premake clients at important changes. (before bisect needs them)

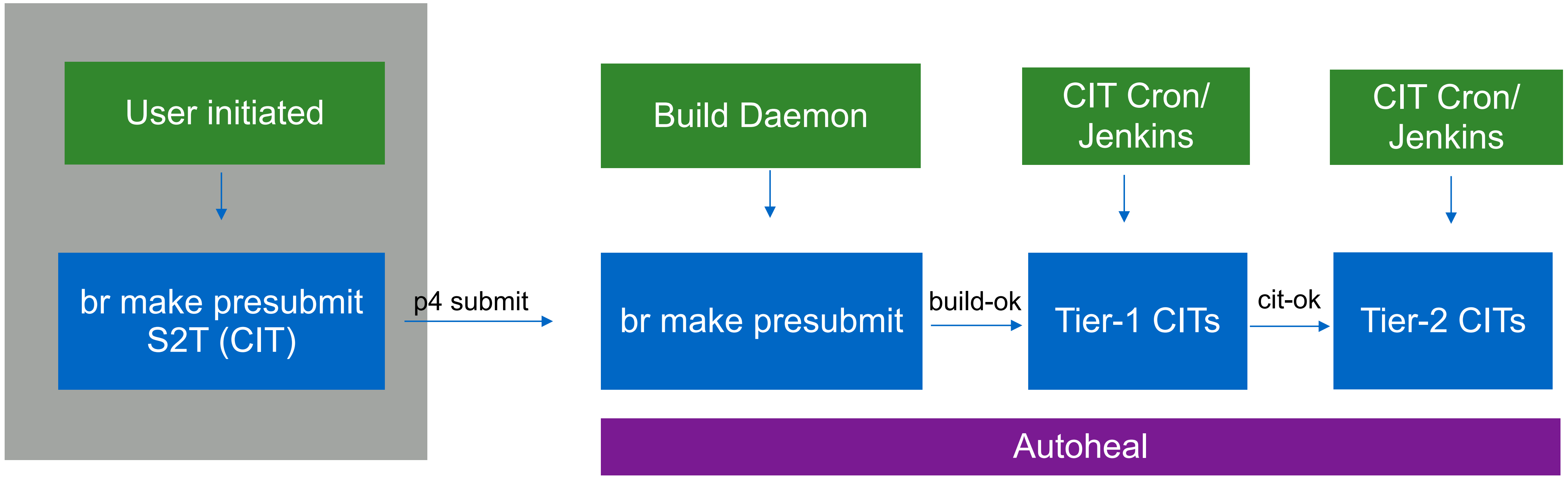
Autoheal (details)

- Autoheal
  - Validation:
    - Re-run at the first failed change, make sure it fails.
    - Re-run at head-of-line with SUSPECT change reverted, make sure it passes.
    - Validate: All changes before suspect change MUST pass, and all runs after suspect MUST fail.
  - If yes… Submit the revert, and email the user & manager:
    - Change that was reverted and test that failed
    - Instructions to recreate the client, how to run the test.
  - If no… Send message to Build/CIT team warning of intermittent error

Regression Protection layers



Regression Protection layers



‘br make presubmit’ -> Build and much more

- wsget -> get a flex-cloned client <1 minute
- br make presubmit (~10 minutes)
  - Enforce coding standard/static analysis: (fail if violated)
    - clang-format: require code in Netapp coding standard
    - include-what-you-use: remove unneeded includes
    - clang-tidy: validate C/C++ code
    - Python (pep8): passes clean
    - Man pages: missing commands?
    - Gdb macros: still work?

      …
  - Compilation: (fail on warning)
    - Compile w/ aggressive Clang warnings

‘br make presubmit’ -> Build and much more

- br make presubmit (~10 minutes)
  - Unit-test execution:
    - Run ~28k CxxTest-based linux unit-tests (<5 minute execution)
    - Address sanitizer/Undefined sanitizer for all unit-tests
    - Valgrind for a subset of unit-tests
    - Thread sanitizer for a subset of unit-tests
  - Linux-based simulator testing (<5 min)
    - Execute workflow tests on a pared-down version of ONTAP
  - Libfuzzer corpus execution (<5 min)
    - Run checked-in corpus w/address sanitizer.
  - Code coverage (<5 min)
    - Generate UT code coverage information (including coverage of pending change)

Get Ready for Submission

- source-2-test (S2T)
  - Combines client diff + CIT coverage data -> pick 6 CITs to run before submissions… Runs them.
  - Coverage analysis algorithm augmented with machine-learning results.

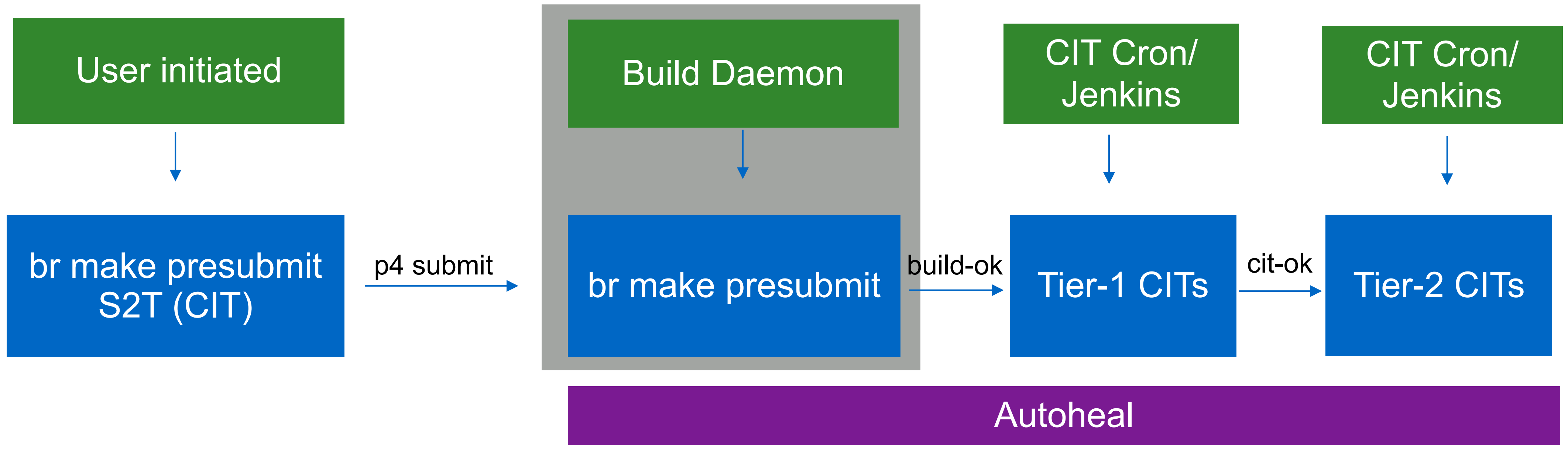
- Submit review to reviewboard

- p4 submit (w/Netapp additions)
  - Validates you've built the pending changes

  - Validates that S2T has passed

  - Checks for pending conflicting changes

Regression Protection layers



NetApp

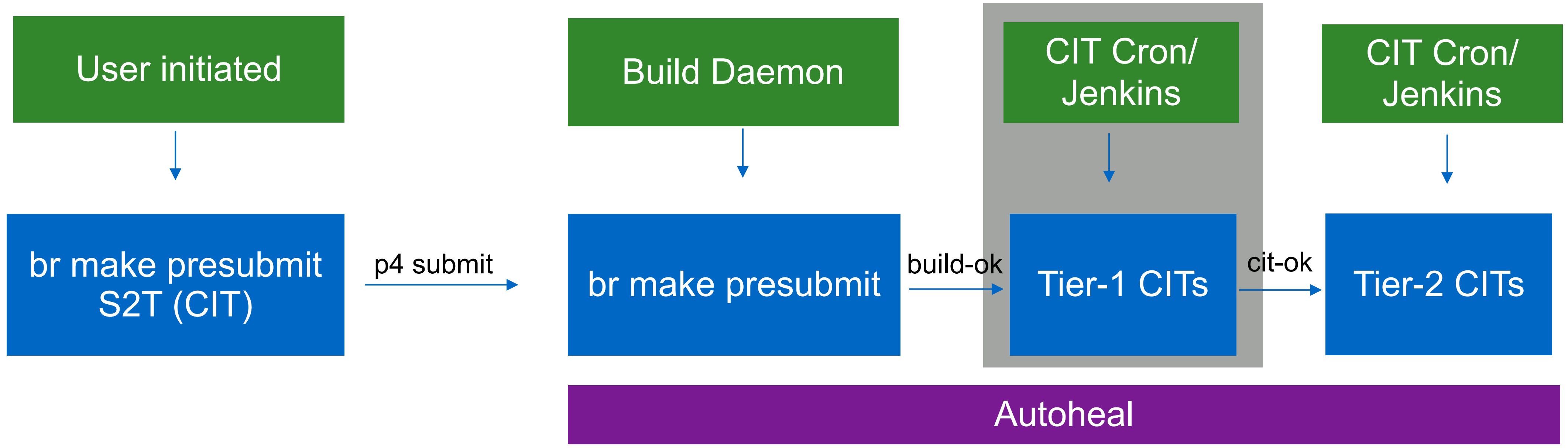
Post-submission: build-ok

- Bammbamm daemons wake up and build change. (every 20 min)
  - If it passes ‘br make presubmit’, new ws* snapshots are created, and the change is stamped ‘build-ok’
  - If it fails, bisect is started.

- Autoheal:
  - Use automation+bisect to detect which change broke the build.
  - Once verified, automatically revert change from the line. (ie. Submit an inverse of the bad change)
  - User gets email about revert and how to reapply.

- Bammbamm daemon will sync forward try again
  - If build passes @change passes, stamp change as ‘build-ok’
  - Implications: wsget clients (which use build-ok) will always build AND in-build unit-tests will always pass.

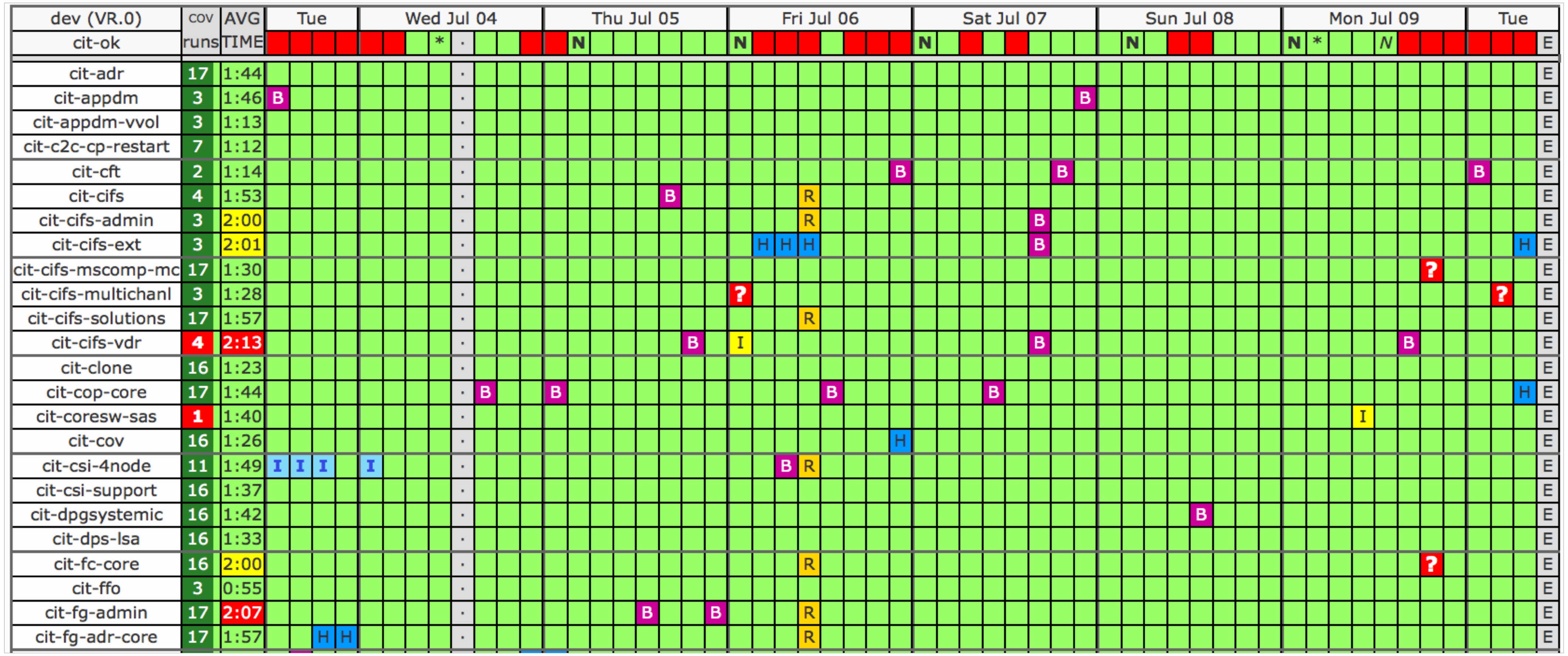
Regression Protection layers



Post-submission: CITs (tier-1)

- Continuous integration tests (CITs)
  - ~120 2-hour tests running testing ONTAP and OFFTAP in the smoke bubbles.
  - Primarily run on VSIM, with some HW.
  - Run every 3-hours on the latest build-ok.
  - If all tier-1 CITs pass on a given change, the change is stamped ‘cit-ok’
- Autoheal for CITs
  - If any CITs fail, the offending change is bisected, and autohealed out of the line.
- CITs
  - Have strict requirements on intermittent failure rates. (<5%)
  - Require a dedicated sheriff, who must triage all failures. (+ mailing list named after cit)
  - 24-hour operational support across multiple Netapp sites.
  - If cit-ok isn’t stamped within 24-hours, line is locked and fixed.

CIT: Week at a glance (WAAG)



dev (VR.0)	cov	AVG	Tue				Wed Jul 04								Thu Jul 05								Fri Jul 06								Sat Jul 07								Sun Jul 08								Mon Jul 09								Tue			
cit-ok	runs	TIME	🟥	🟥	🟥	🟥	🟥	🟥	🟩	*	·	🟩	🟩	🟥	🟥	N	🟩	🟩	🟩	🟩	🟩	🟩	N	🟥	🟥	🟥	🟩	🟥	🟥	🟥	N	🟩	🟥	🟩	🟥	🟩	🟩	🟩	🟩	N	🟩	🟥	🟥	🟩	🟩	🟩	N	*	🟩	🟩	N	🟥	🟥	🟥	🟥	🟥	🟥	E
cit-adr	17	1:44	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-appdm	3	1:46	B	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-appdm-vvol	3	1:13	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-c2c-cp-restart	7	1:12	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-cft	2	1:14	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	E
cit-cifs	4	1:53	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	R	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-cifs-admin	3	2:00	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	R	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-cifs-ext	3	2:01	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	H	H	H	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	H	E
cit-cifs-mscomp-mc	17	1:30	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	?	🟩	🟩	🟩	🟩	E
cit-cifs-multichanl	3	1:28	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	?	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	?	🟩	E
cit-cifs-solutions	17	1:57	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	R	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-cifs-vdr	4	2:13	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	I	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	E
cit-clone	16	1:23	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-cop-core	17	1:44	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	B	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	H	E
cit-coresw-sas	1	1:40	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	I	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-cov	16	1:26	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	H	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-csi-4node	11	1:49	I	I	I	🟩	I	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	R	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-csi-support	16	1:37	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-dpgsystemic	16	1:42	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-dps-lsa	16	1:33	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-fc-core	16	2:00	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	R	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	?	🟩	🟩	🟩	🟩	E
cit-ffo	3	0:55	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-fg-admin	17	2:07	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	B	🟩	🟩	B	🟩	🟩	🟩	R	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E
cit-fg-adr-core	17	1:57	🟩	🟩	H	H	🟩	🟩	🟩	🟩	·	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	R	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	E

CIT: Triage/Operation -> Jenkins

- Jenkins (stuck in the middle)
  - Clearing-house for CIT results.
  - Blends into preexisting infrastructure
    - Preexisting processes -> trigger Jenkins jobs -> trigger other Preexisting processes.
    - Can spin up Jenkins instances/slaves in different test/compute environments.
- Jenkins gathers results, and allows for triage of each failure
  - Homegrown tools wrapped around Jenkins to make common triage easier.
  - Tooling created to automatically add new CITs

Your change hit cit-ok (email)

Hello user,

The CIT-OK marker on DOT:dev has moved from 4954128 to 4954794, and these recent change(s) of yours are now CIT-OK:

Change Number	Change Description	Burt Associated
4954359	1) Create a kernel version of ems_helpers. (Since almost all of the code is the same, I just recompile the same...	1172664

This is not an absolute guarantee that your change(s) will not be reverted, but it is a good indication that it has not caused any serious issues.

Please consider using wstakechange for propagating your changes to other codelines.

E.g. To propagate change #11111 to DOT:Rfullsteam and run build/smoke tests for verification:

```
wstakechange -c 11111 -d DOT:Rfullsteam -t build,smoke
```

Alternatively, you can use "p4 take_change -state auto -c new changenum" to bring these changes into applicable prior releases.

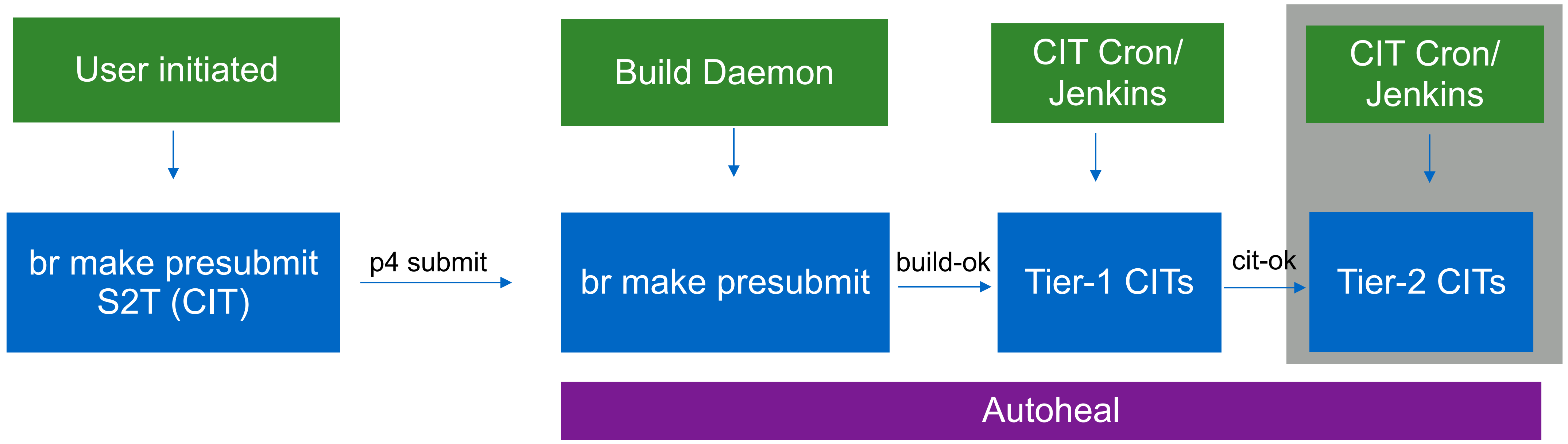
Regards,

Build Team

Your change hit cit-ok

- Autoprop starts
  - Requested changes are applied to release branch client.
  - If it can be applied and builds, it is submitted.
  - If not, user-gets an email with details and manual instructions about how to take it.

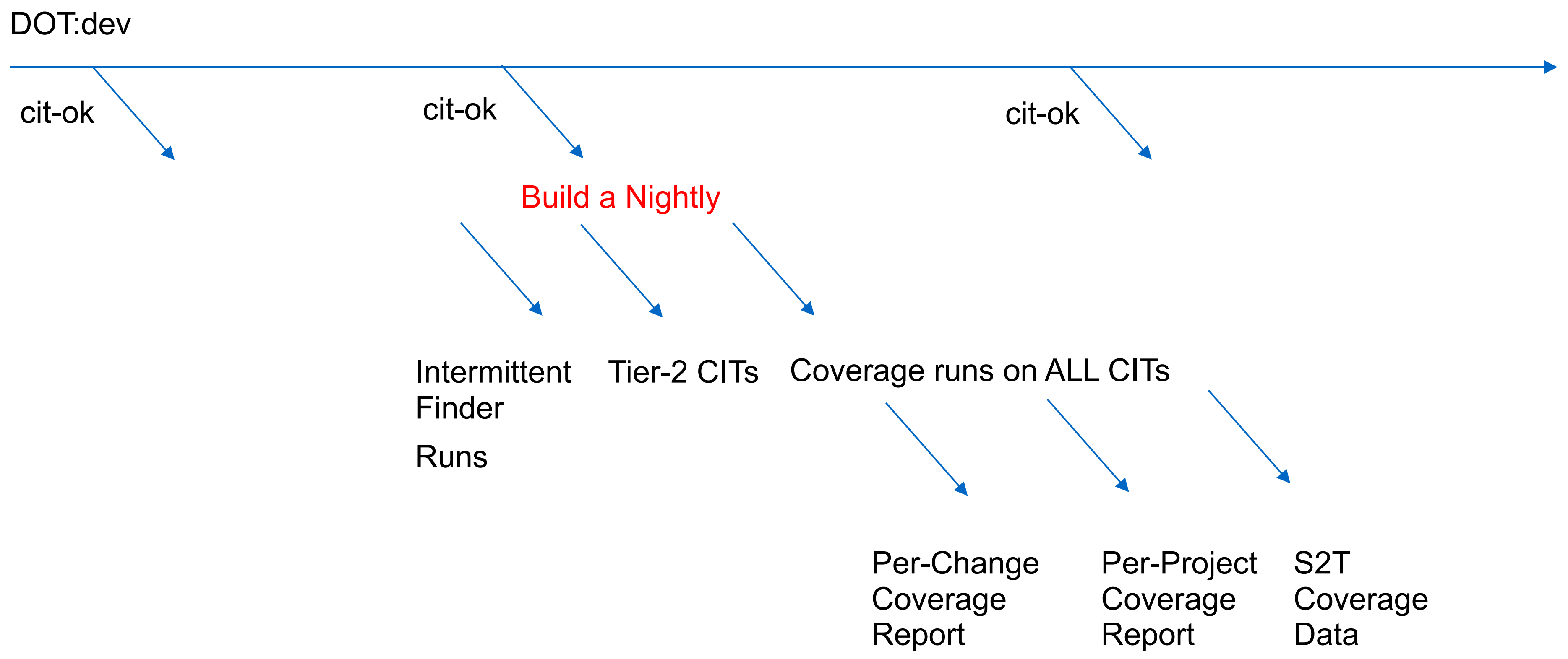
Regression Protection layers



Post-submission: CITs (tier-2)

- Tier-2 CITs run at lower cadence
  - ~525 tier-2 CITs
  - Follows all the requirements of CITs
  - Typically ‘lower-risk’ CITs. (higher-coverage tests are pushed to tier-1)
  - Runs daily on a cit-ok build.
- Failures are autohealed out of the line.
  - Bigger change range to bisect over, but will eventually be reverted. (a few days rather than hours)
  - Does NOT block cit-ok… so errors may linger longer and can be present in a cit-ok build.

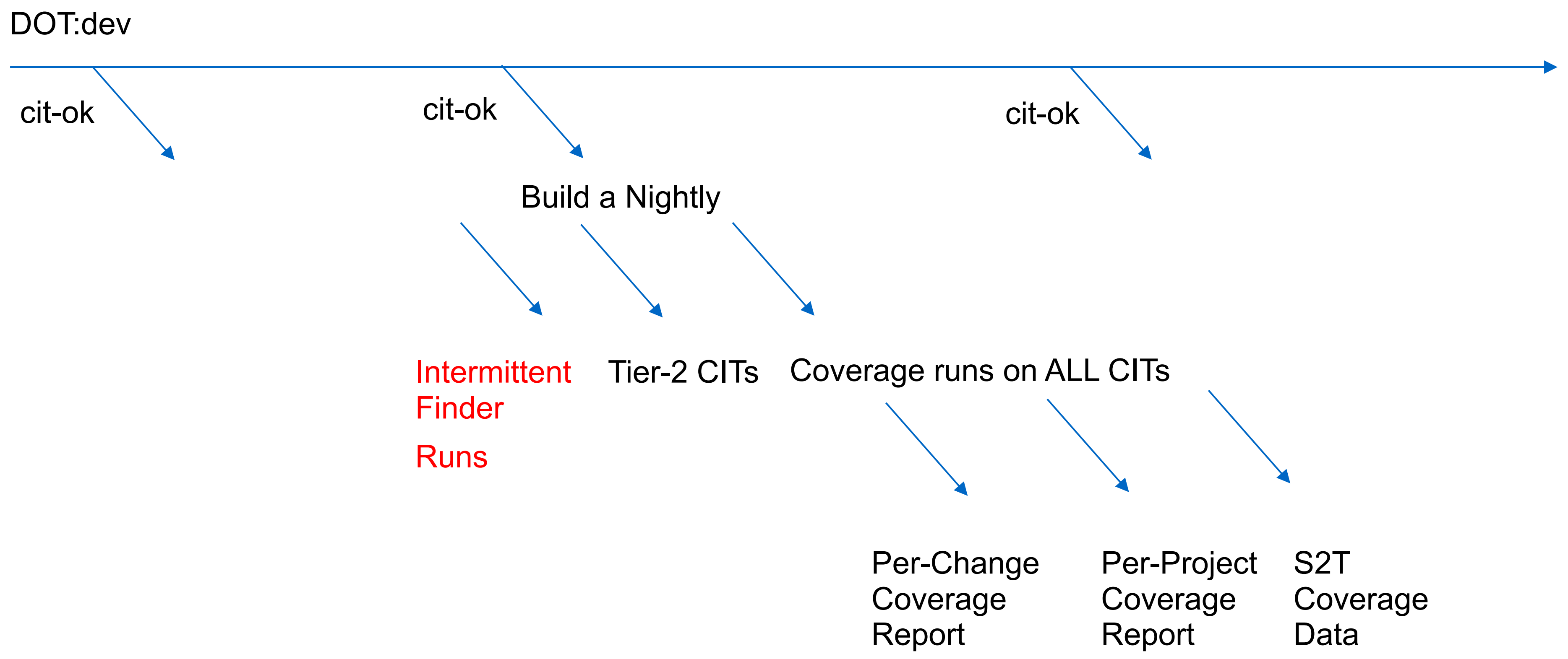
DOT:dev – Beyond cit-ok



Create a nightly

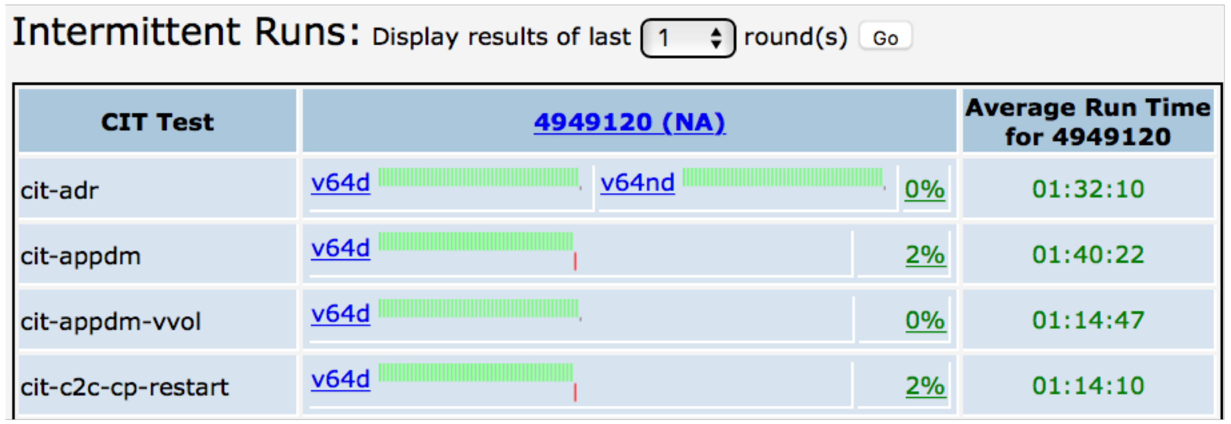
- Once a day: the latest cit-ok is built from-scratch
  - Create a long-term build (typically used by QA for deeper testing)
  - Targets beyond ‘br make presubmit’ are built. Feed-back based optimizations are performed.
  - In release branches, these are the basis for bits shipped to customers.

DOT:dev – Beyond cit-ok



Driving out intermittent errors

- Weekly: All tier-1 CITs are run 50 times on a cit-ok change.
  - This CIT must have passed at that change to be stamped cit-ok, so….
  - Any failure are due to intermittent issues in infra, product or test code.
  - Regular runs help identify WHEN issues started to occur.
- Status tracked in summary page:
  - All failures must be triaged and driven out.
- Intermittent bisect:
  - Given a failure rate, a good & bad change, a CIT + test case,
  - We can track down which change introduced an intermittent error (within a given confidence level)

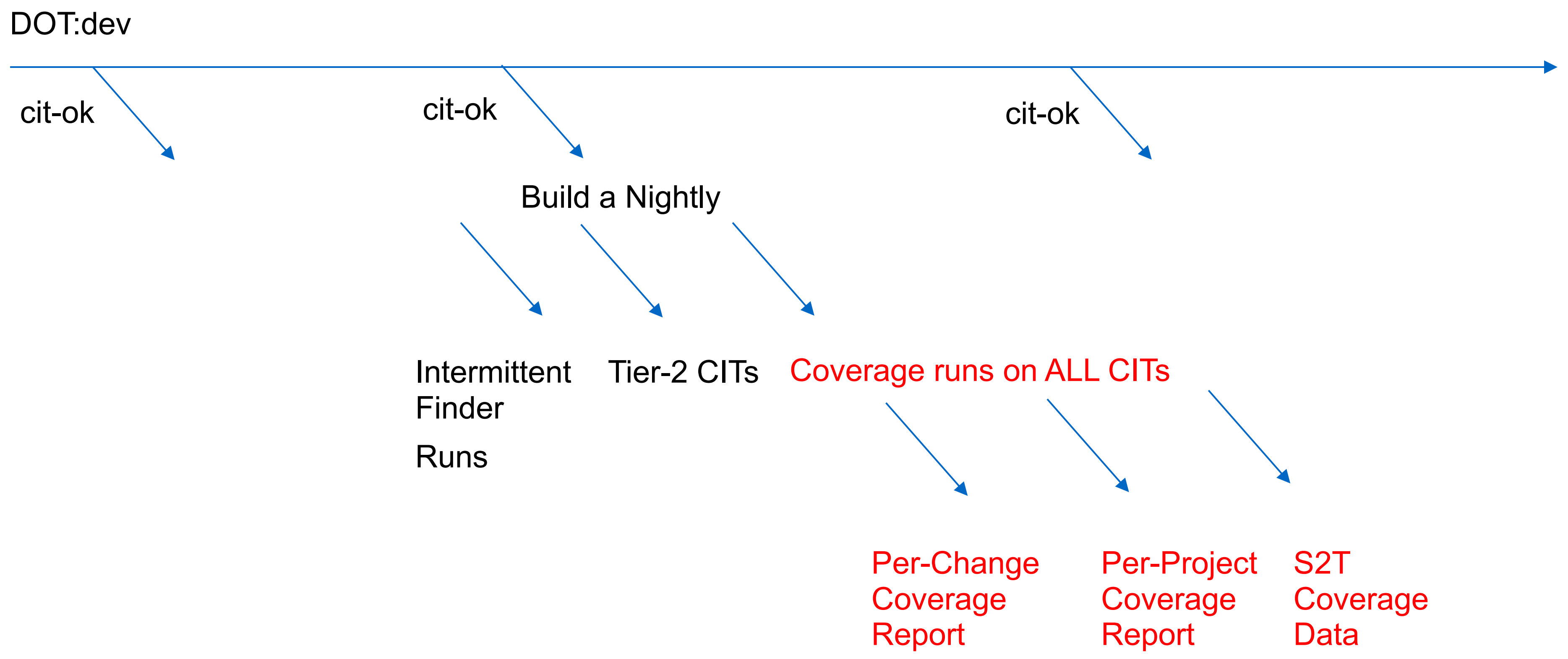


Intermittent Runs: Display results of last 1 round(s) Go

CIT Test	4949120 (NA)			Average Run Time for 4949120
cit-adr	v64d	v64nd	0%	01:32:10
cit-appdm	v64d		2%	01:40:22
cit-appdm-vvol	v64d		0%	01:14:47
cit-c2c-cp-restart	v64d		2%	01:14:10

NetApp

DOT:dev – Beyond cit-ok



Generate coverage data

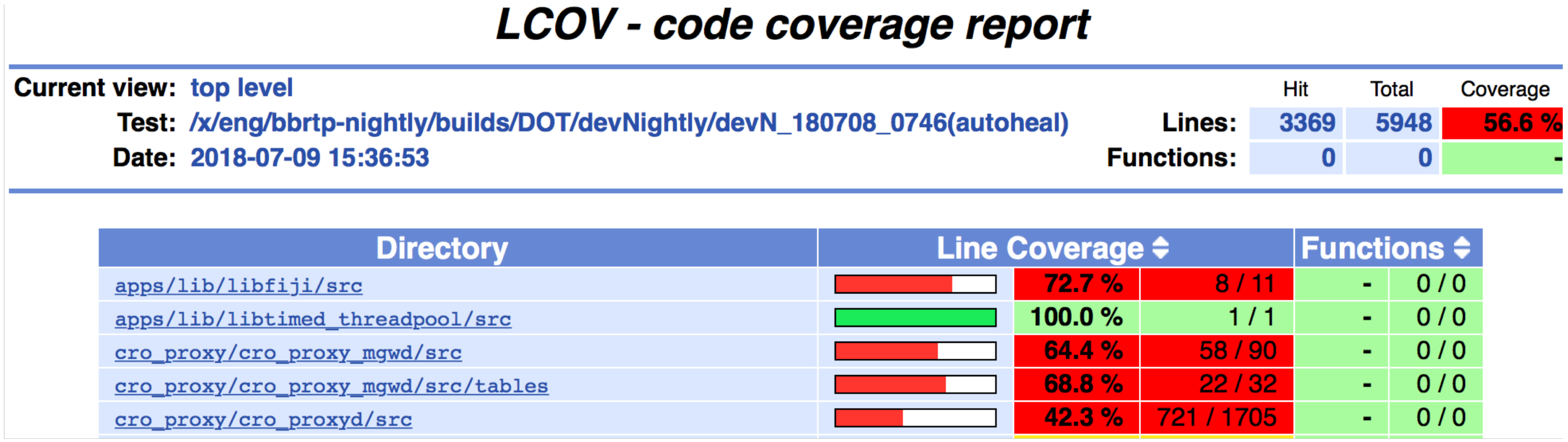
- Coverage variants of every (650+) CIT are run on the latest nightly.

- Data is gathered from the filer, combined with the in-build unit-test coverage data
  - Post processed to be human readable. (~18+ hour process)
  - Post-processed to be machine readable for quick source-to-test (S2T) analysis.

- ”Coverage in the autoheal layer” -> In-build UT + CIT tier-1 + CIT tier-2
  - Used for project release criteria

Per-Change/Per-Project Coverage Report

- Per-Change: Send developers reports on autoheal coverage of every submitted change.
- Per-Project: Aggregate coverage of all change for an ONTAP project into one report.
  - Each project has UT and Autoheal coverage goals.. Don't ship until hit.
  - Project reports are recalculated nightly with fresh code-coverage data:



LCOV - code coverage report

Current view: top level
Test: /x/eng/bbrtp-nightly/builds/DOT/devNightly/devN_180708_0746(autoheal)
Date: 2018-07-09 15:36:53

	Hit	Total	Coverage
Lines:	3369	5948	56.6 %
Functions:	0	0	-

Directory	Line Coverage			Functions	
apps/lib/libfiji/src	🟥	72.7 %	8 / 11	-	0 / 0
apps/lib/libtimed_threadpool/src	🟩	100.0 %	1 / 1	-	0 / 0
cro_proxy/cro_proxy_mgwd/src	🟥	64.4 %	58 / 90	-	0 / 0
cro_proxy/cro_proxy_mgwd/src/tables	🟥	68.8 %	22 / 32	-	0 / 0
cro_proxy/cro_proxyd/src	🟥	42.3 %	721 / 1705	-	0 / 0

Does it work?

- Yes!
  - Autoheal layer has grown 20 CITs to 650+ CITs.
  - In-build UT has grown similarly
  - Since we started CI + autoheal:
    - Each subsequent ONTAP release becomes the highest quality ONTAP release
      - Disruption/Node
      - CI + Autoheal is part of a large shift in uniformity of project reporting and expectations
    - ONTAP shifted from a multi-year release to 6-month release
    - ONTAP's backend release (from branch to ship) has shrunk (and continues to by months at a time..)
    - Other Netapp software is adopting this strategy

Summary:

- Continuous Integration + Autoheal has given ONTAP:
  - Faster cadence
  - Higher quality
  - Efficient path for new quality bars

- Success with CI requires change:
  - New processes to require it
  - New tooling to track it
  - New dev workflow (no branches)
  - Product mindset change
    - No regressions tolerated
    - Revert is a blessing.. not a curse.

NetApp

Thank You