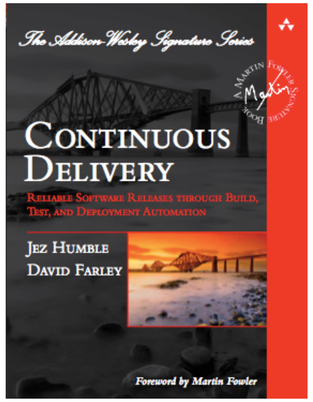
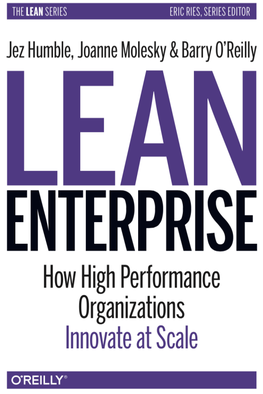
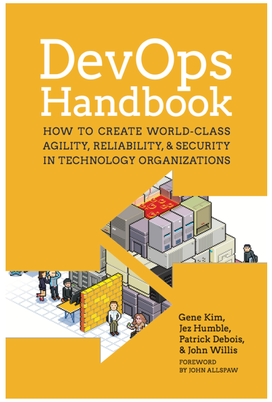
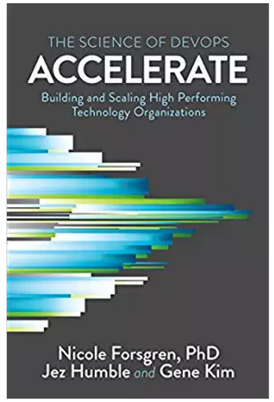
the continuous thunderdome

jez humble | ldx3 nyc | 15 september 2026







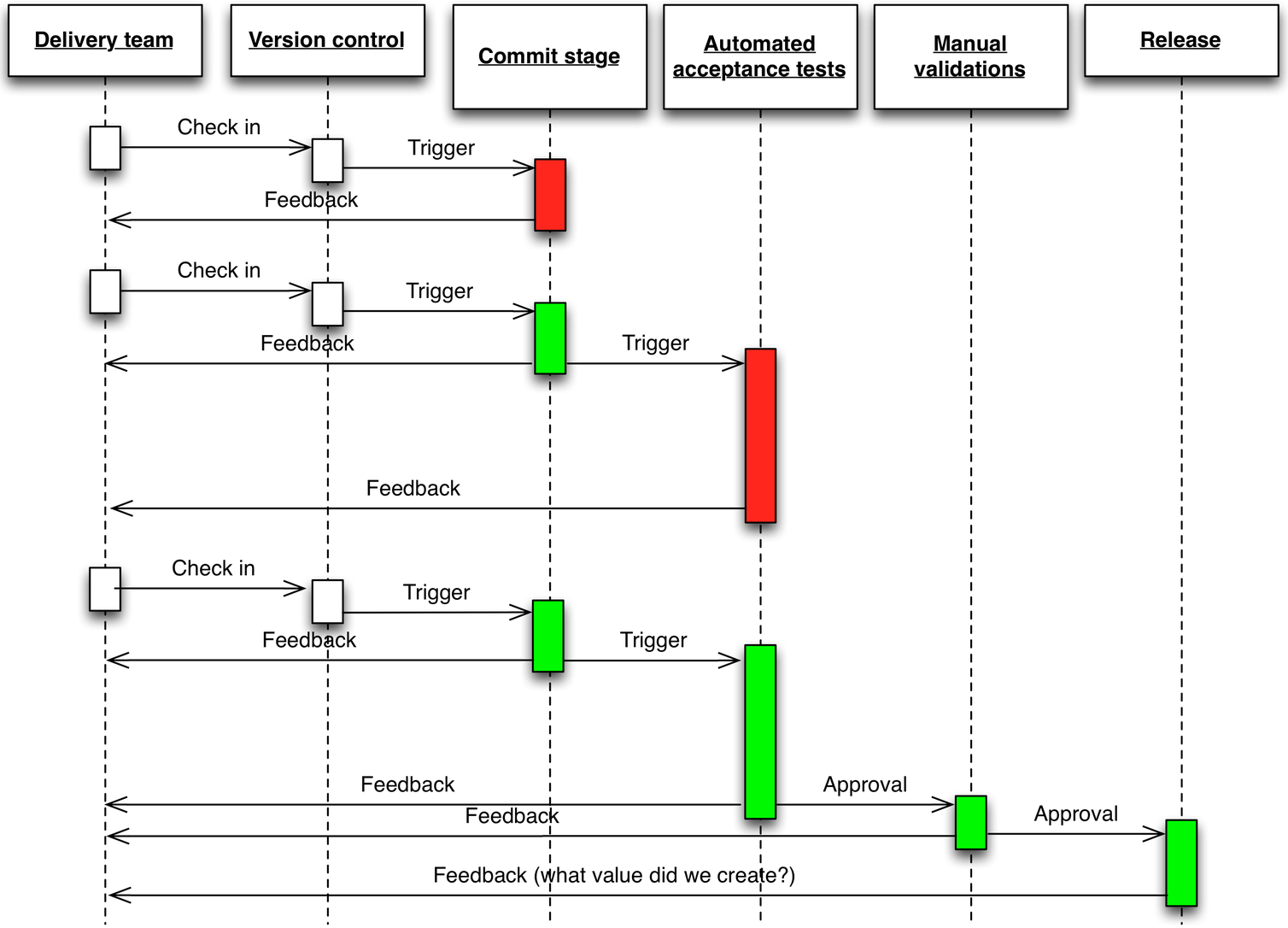


what is continuous delivery?

The ability to get changes—features, configuration changes, bug fixes, experiments—into production or into the hands of users safely and quickly in a sustainable way.

https://continuousdelivery.com/

deployment pipeline



@jezhumble

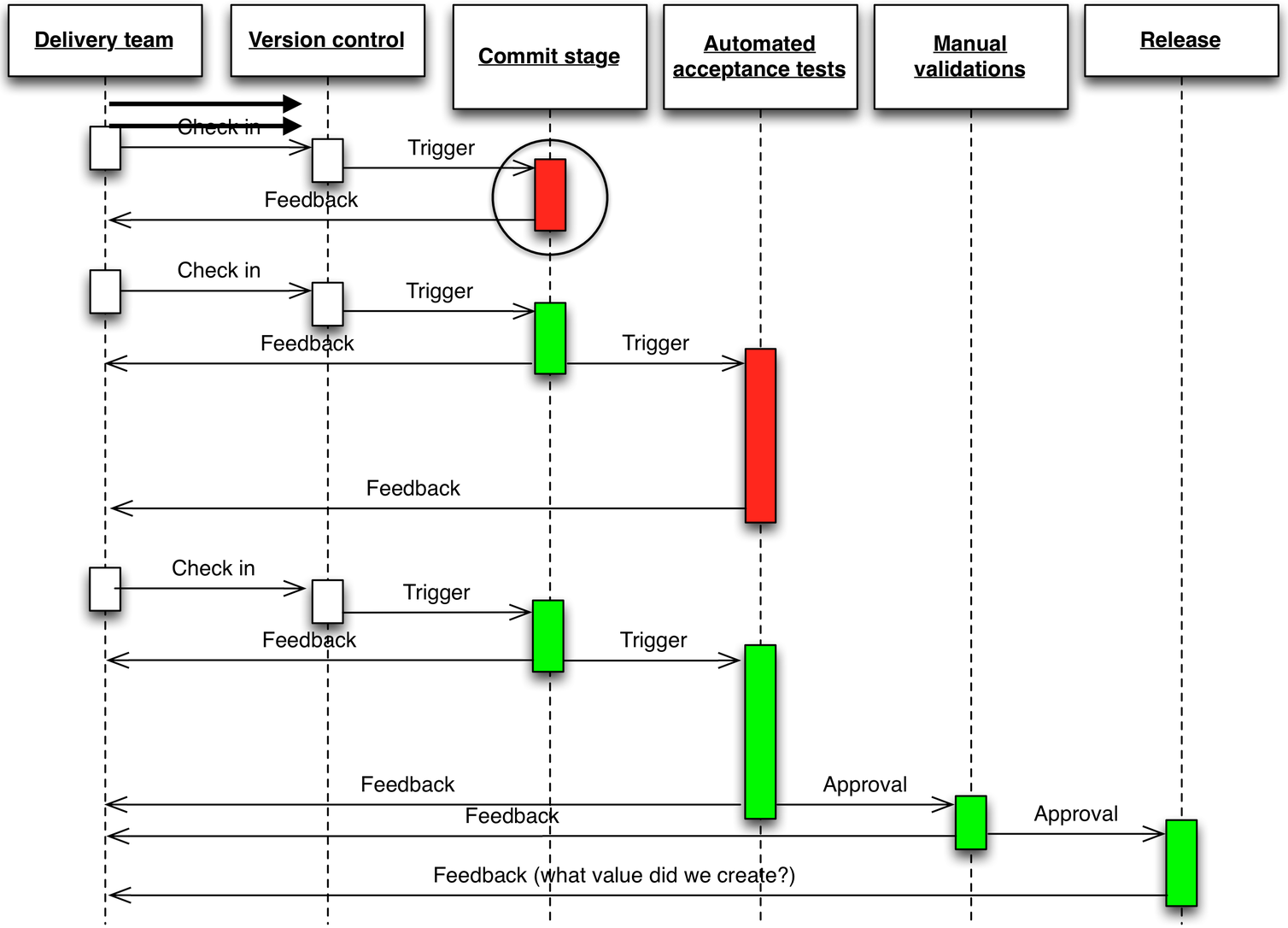
“the metamorphosis of ci/cd”

“Unfortunately, it breaks under heavy load. If you have 100 developers committing once a day, then you have 100 serial builds to run while merging their commits to main. If your build takes 30 minutes, you have 50 hours of sequential builds to run every day. Whoopsie!

“CI/CD systems solved this problem with a Merge Queue, where commits turn into queued Merge Requests (MRs) which you can intelligently reshuffle and, importantly, batch up.”

https://yegge.ai/essays/the-shape-of-things-to-come/

deployment pipeline

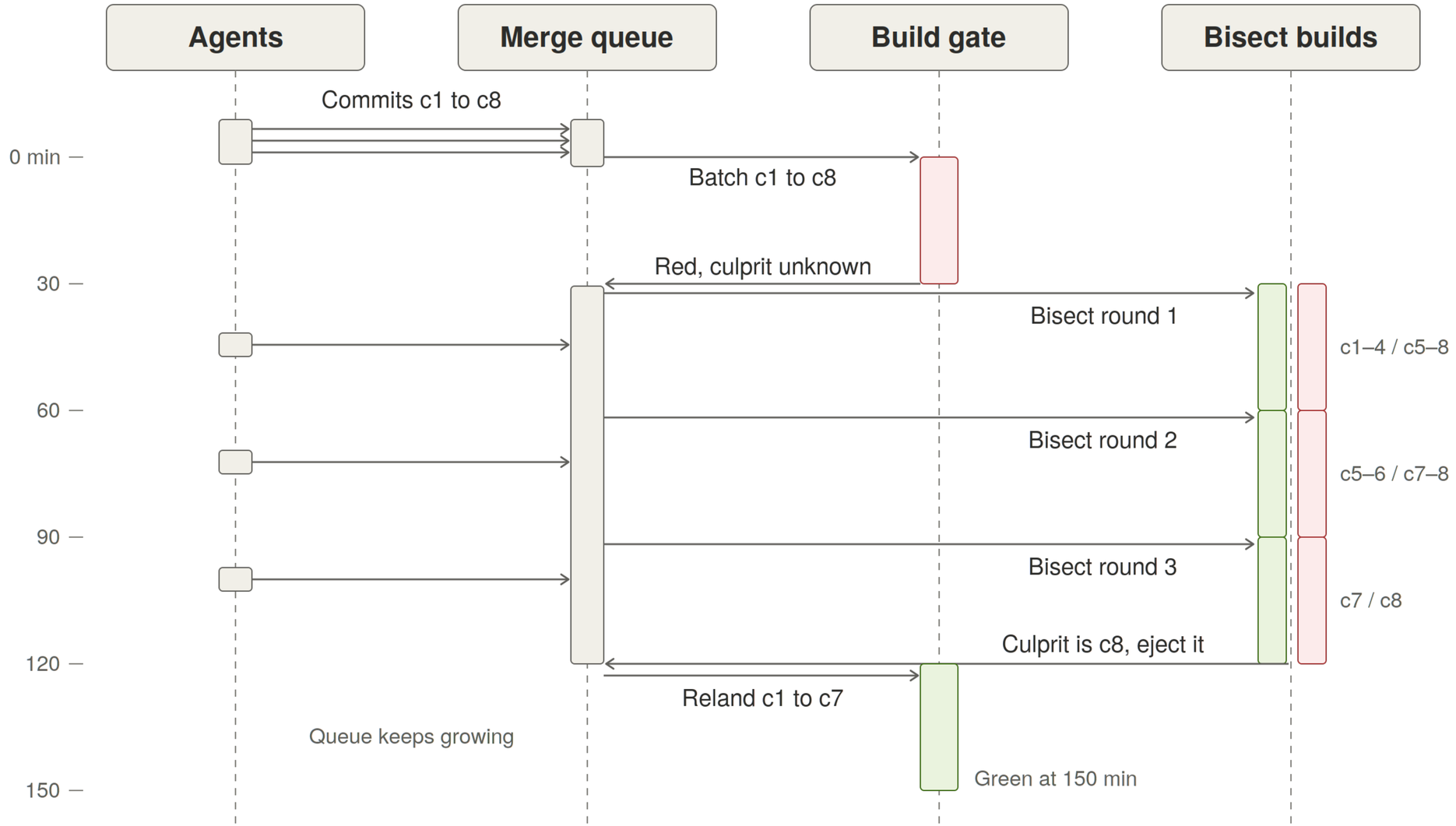


“the metamorphosis of ci/cd”

“So you bisect the batch, rerun the build on each half, and eliminate half the candidates with each run. This gives you log(N) recovery from a spoiled batch—which is just tickety-boo, but it still means that any given batch can add several hours to the queue.”

https://yegge.ai/essays/the-shape-of-things-to-come/

integration in the merge queue (don’t do this!)



@jezhumble

what is the continuous thunderdome?

“My proposal, which turned out to be a valid approach, was to Mad Max it: Just slam all the commits onto main, and then just friggin' deal with it. No bisections, no sequencing, no blame, none of that old crap. Just fix it and roll forward.”

https://yegge.ai/essays/the-shape-of-things-to-come/

continuous integration c. 2000

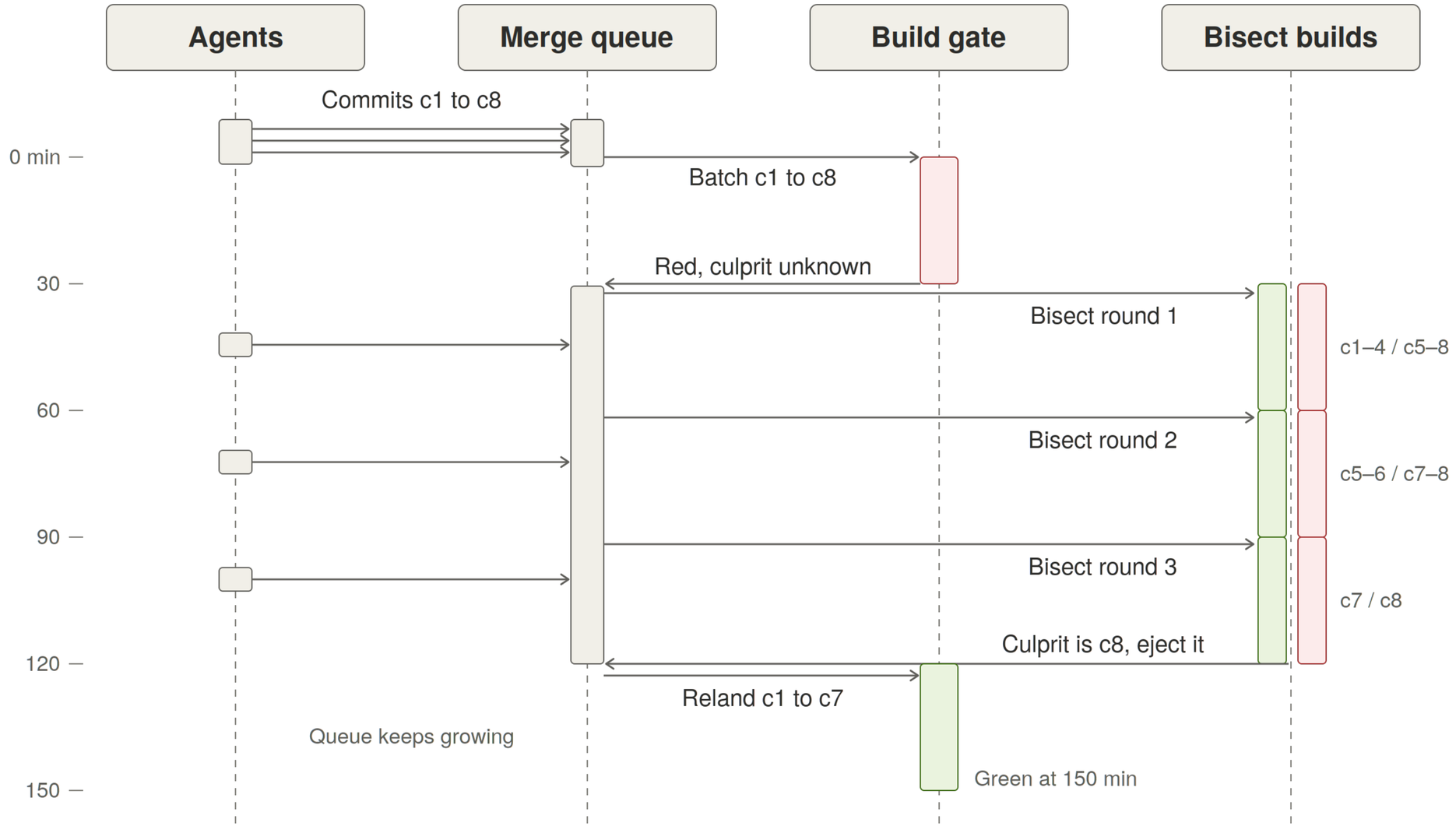
- Everyone commits to trunk / main at least daily. No long-lived branches.
- Every commit triggers an automated build and test on main.
- If it goes red, the team fixes it in minutes or reverts. Nobody commits on top of red.

Beck, Extreme Programming Explained (1999);
Fowler, "Continuous Integration" (2006, revised 2024)

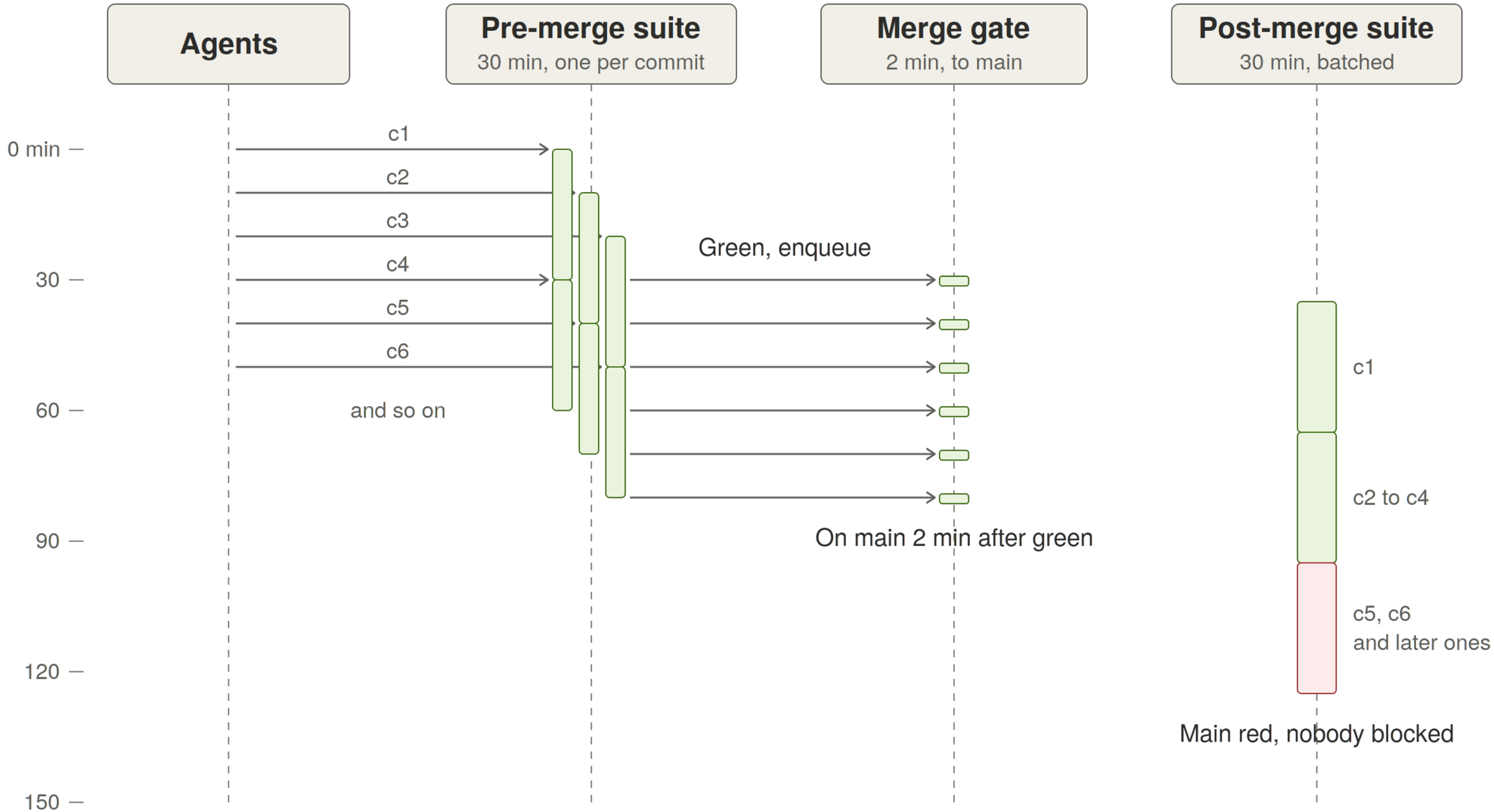
ci is a practice, not a tool

- having a tool that runs your merge queue is not doing CI; integrating to trunk/main continuously and keeping it green is.
- The build must be fast enough to fix, not just to run.

instead of integrating in the merge queue…



keep the merge gate to <2m



the merge gate

strictly 2m or less. parallelize everything in it.

compile

lint

80s of unit tests

nothing more than a few hours (max 12h) behind HEAD

post-merge build / tests

if it goes red, it doesn’t block new commits

bisect out-of-band — and/or have an LLM find the bad commit

roll back bad commits

in a monorepo you must do dep analysis & run those tests: **use bazel**

cut releases off green builds; sync from last green build rather than HEAD

does this scale?

- **>50,000 commits / day** (>1 per second); ~11-minute presubmit predicts full-suite success with 95%+ accuracy
- **>4 billion test cases/day**; postsubmit cycles batch thousands of commits
- **<1% of commits break postsubmit** (derived), but ~2 of 3 investigated failure ranges turn out to be flakes, not culprits
- **Detection and diagnosis both got smarter:** ML-scheduled speculative cycles spot breakages in 37 min vs. 107; flake-aware Bayesian culprit finding then convicts at 98% accuracy

Google (references at end)

let’s talk about tests

every component / service should have a fast suite, 10m or (much) less

verification burden moves from "did the human write it right" to "does the suite actually pin down the intended behavior."

AI-generated *tests* are a new failure mode: tautological tests, tests that codify the bug, coverage theater (all forms of reward hacking)

look at *information value* of tests: are they finding defects? At what cost?

deploy property-based testing and mutation testing

deployment pipeline

a CPU pipeline for changes. Promote each change optimistically, run every stage continuously with the latest good artifact from the previous.

optimizes for throughput and critically **instruments the process**

sustainable landing rate dependent only on fix accuracy, break rate, and verification latency

not a fixed artifact but a **process** under continuous improvement

use agents to do the process improvement!

key metrics

lead time from PR creation to approved

lead time from pushing a commit to verdict pre-merge

lead time submit to deployable

% trunk is green & time-to-green after red

standing agentic background jobs

- agents don’t like fixing the build… they love it!
- flake detection and fix
- reduce lead time
- find and fix implicit dependencies and leaked state
- write test utilities, like re-running new tests 3x pre-merge to detect flakes / run tests in random orders to detect dependencies

build quality in

"Cease dependence on inspection to achieve quality… *Build quality into the product* in the first place"

W. Edwards Deming

epilogue from steve (2026-09-09)

Sixty days later Steve sent me an update: no more megabatches. Batch size limited to 16 commits (vs up to 150 in a megabatch)

He removed the swarm & bisect: failing commits are now evicted via dependency analysis, and red trunk auto-reverts

Throughput went up, and latency went from 1 day to < 1h

While he doesn’t run presubmit tests (not recommended!), notice the direction of the change

Steve’s numbers, shared with permission

thank you!

visit http://continuousdelivery.com/ and subscribe for free stuff

google bibliography (note: this talk only contains public information!)

- Software Engineering at Google, chapter 23: https://abseil.io/resources/swe-book
- https://research.google/pubs/what-breaks-google/
- https://hackthology.com/speculative-testing-at-google-with-transition-prediction.html
- https://hackthology.com/flake-aware-culprit-finding.html
- https://dan.bodar.com/2012/02/28/crazy-fast-build-times-or-when-10-seconds-starts-to-make-you-nervous/ (not Google, but worth revisitng anyway!)