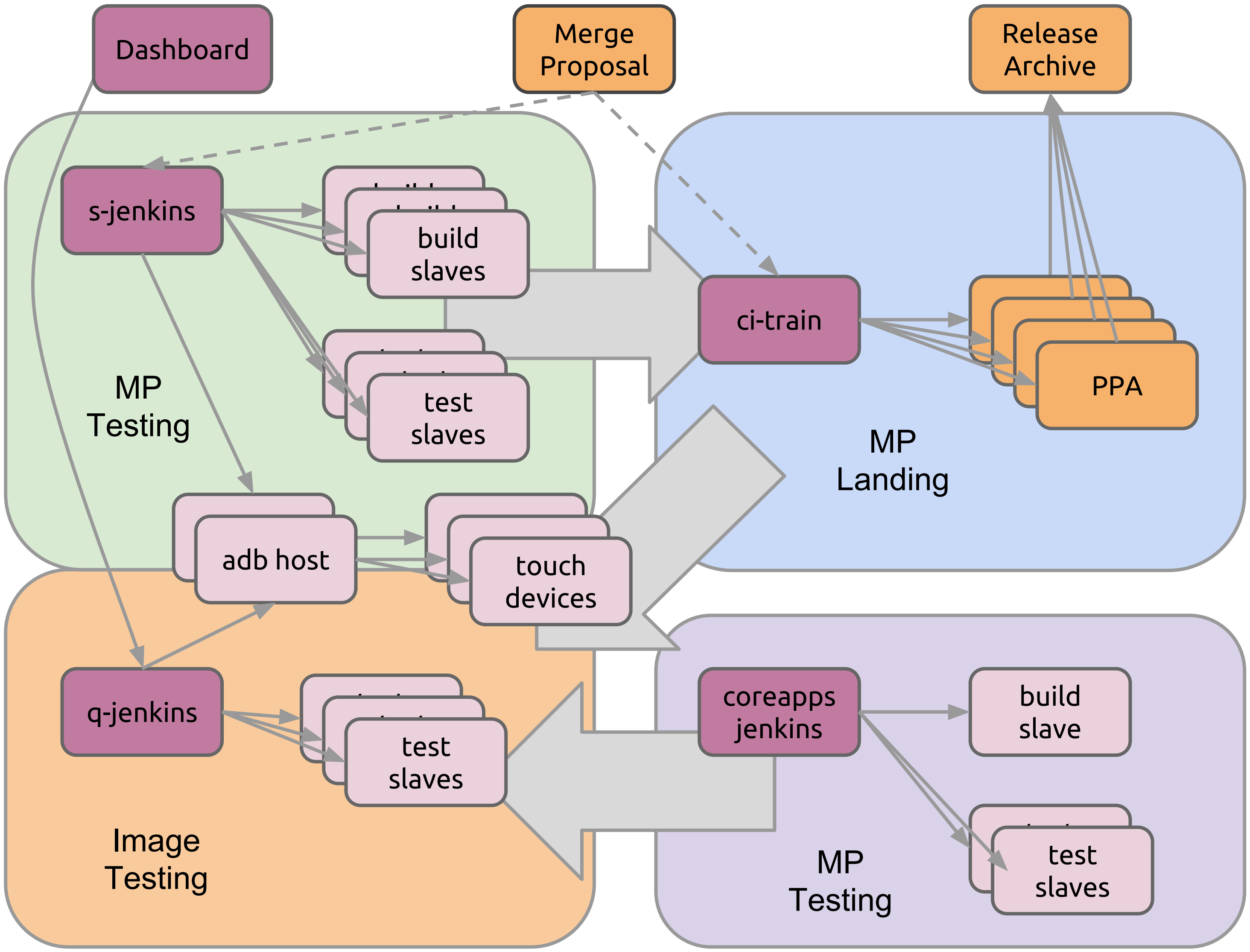
Legacy


Dashboard
Merge Proposal
Release Archive
s-jenkins
build slaves
test slaves
MP Testing
adb host
touch devices
ci-train
PPA
MP Landing
q-jenkins
test slaves
Image Testing
coreapps jenkins
build slave
test slaves
MP Testing

Functional goals

- **Enterprise-level reliability and durability**
  - Migrating away from legacy pre-Prodstack services (CI Lab) while increasing their stability and performance, and our responsiveness
  - CI Airline
- **Self-service code landing**
  - Migrating from CI Train to CI Airline
  - Increase your development velocity by lowering reliance on the LTF for paperwork tasks
  - Empowering LTF to focus more on the grey areas (blaming and coordinating response)
- **Readiness for growth**
  - Hardware profiles increasing
  - Types of software testing increasing
  - Need to retain architectural simplicity
- **Adaptability**
  - CI needs often change with time and vary by department
  - Need a lego-brick approach

Stakeholder goals

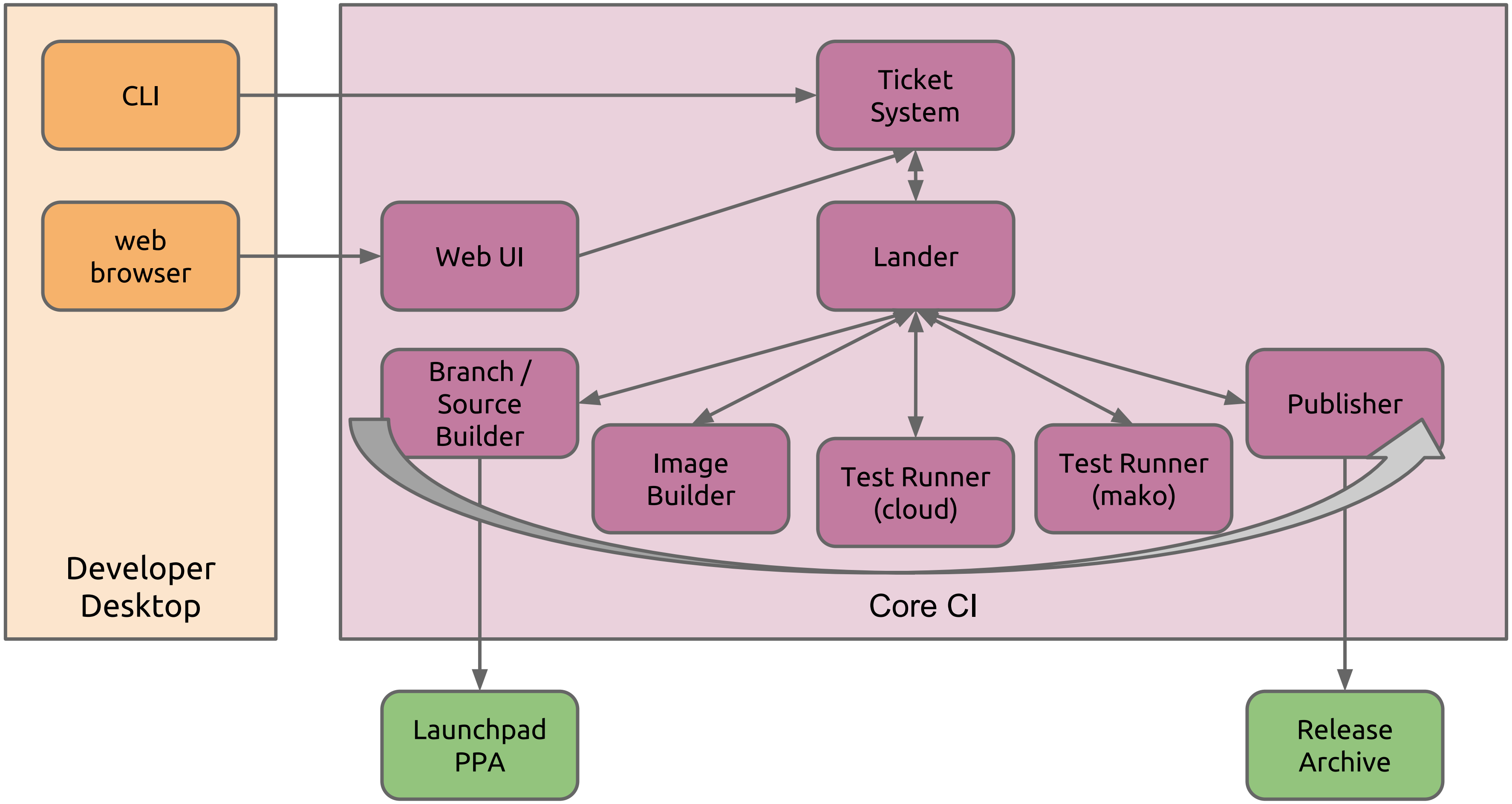
- **Development involvement**
  - Open staging deployment makes evaluating the next generation easy and provides an avenue to giving feedback on our approach
  - Having the same people support (Vanguards) and develop the software makes us acutely aware of your problems
- **Unambiguous communication**
  - Vanguard shifts ensure there is one obvious point-of-contact for the CI team at most times
  - Team-specific Asana projects and dedicated CI representatives let us work directly with you without all the noise from our other efforts
- **Frictionless experimentation**
  - An easily reproducible stack means you can easily debug issues up through CI without requiring access to the production system
  - You have the same tools and access to add new features that we do. Add some cloud credentials and you have all the resource you need
  - The architecture fits in your head. No need to understand all components

Architecture

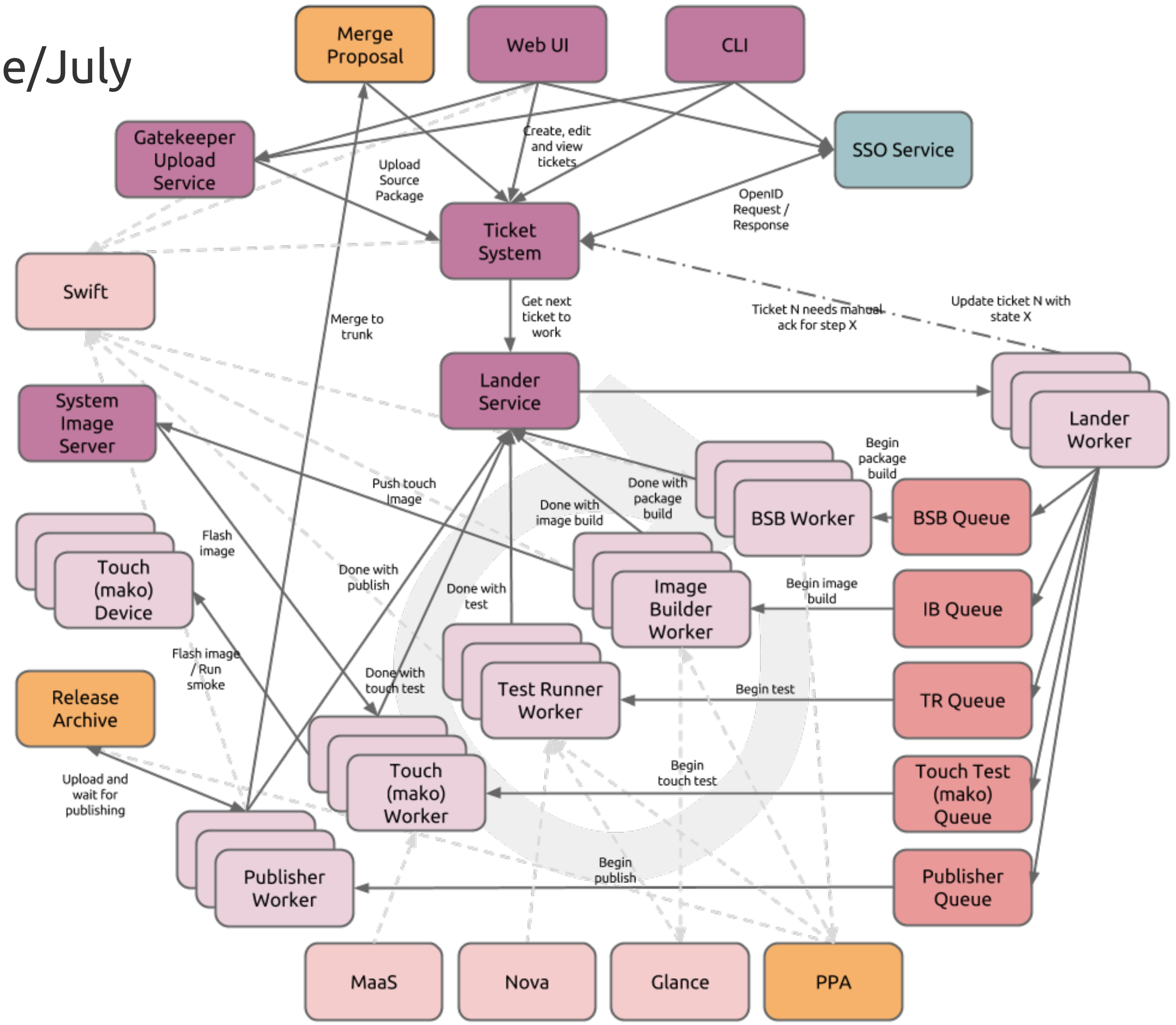
Writing a Continuous Integration system is not hard.
Writing one that does not grow into an operational nightmare is.

- **Atomicity**
  - Every step progresses a ticket or has no effect
- **Automatic failure isolation**
  - A failing ticket is automatically moved onto other hardware until it’s programmatically determined to be the source of failure
- **Resilience to failure**
  - All worker components are ephemeral
  - Work continues as far as it can, then sits patiently until missing services return without needing a manual “retry”
- **Graceful degradation**
  - Services are isolated by responsibility
  - You can still see your ticket and its artifacts, even if we lose everything but the web interface, ticket system, and Swift

June/July - CI Airline (UCE-0)



June/July


Merge Proposal
Web UI
CLI
Gatekeeper Upload Service
SSO Service
Create, edit and view tickets
Upload Source Package
OpenID Request / Response
Ticket System
Swift
Get next ticket to work
Ticket N needs manual ack for step X
Update ticket N with state X
Merge to trunk
Lander Service
Lander Worker
System Image Server
Begin package build
Push touch Image
Done with package build
Done with image build
BSB Worker
BSB Queue
Flash image
Touch (mako) Device
Done with publish
Done with test
Image Builder Worker
Begin image build
IB Queue
Flash image / Run smoke
Done with touch test
Test Runner Worker
Begin test
TR Queue
Release Archive
Touch (mako) Worker
Begin touch test
Touch Test (mako) Queue
Upload and wait for publishing
Publisher Worker
Begin publish
Publisher Queue
MaaS
Nova
Glance
PPA

List of requirements that have been completed

- CI Train on Prodstack
- Inline comments in Launchpad MPs
- Operational response improvements
  - Instant text message alerting of operational problems
  - Vanguard shifts provide a single POC for the CI team stakeholders
- Operational stability improvements
  - Nagios checks, Landscape, ksplice, centralised auth, centralised DNS, etc
  - Growing consistency in our servers (OpenID auth, single Jenkins version)
- Testing of Oxide (chrome content API) on Prodstack
- Kernel team backlog (health check, power, suspend blocker, etc)
- Test time improvements through parallelisation (~4:10 -> ~1:50)

14.10 Goals

- Deprecate the Train
- Build the emulator into our core process
- Move towards a continuous delivery model for CI development; high velocity without sacrificing quality
- Allow stakeholders to start experimenting with tomorrow’s tools today
- No poisoned tickets or lost time to CI
- Deep metrics into both the functional and human side of the CI process
- Drop-in scale out of many different hardware profiles (plug in a phone -> done)
- Get all non-testbed hardware out of the Lab and onto Prodstack (or IS-managed)

Testability Requirements

Unit testing

- Comprehensive unit test suite gating every merge

Integration testing

- Growing set of full-deployment integration tests running four times per day on trunk
- End-to-end ticket integration test in progress

In-production testing

- Some basic Nagios checks
- Nagios suite growing to cover deeper exercising of the Airline
- Production metrics support landing before Malta

In-production chaos testing

- Plan to prove we can recover from a wedged phone by sending a malicious ticket to production once a day

Risks and challenges

- **Emulator gaps**
  - We don’t yet have a complete picture of where the emulator is not a suitable replacement for a physical phone
  - Need to always finish testing on the phone to catch emulator bugs
- **Unknowns in MAAS**
  - We’re the first team to try to use it like this
- **Nested KVM**
  - We’re going to use MAAS to manage x86 Touch emulators on real hardware. Think of it as a cloud of virtual Touch devices
- **Prodstack deployment**
  - Need to mentally transition from having shell access to all services to building services that report problems well and collect sufficient debugging information