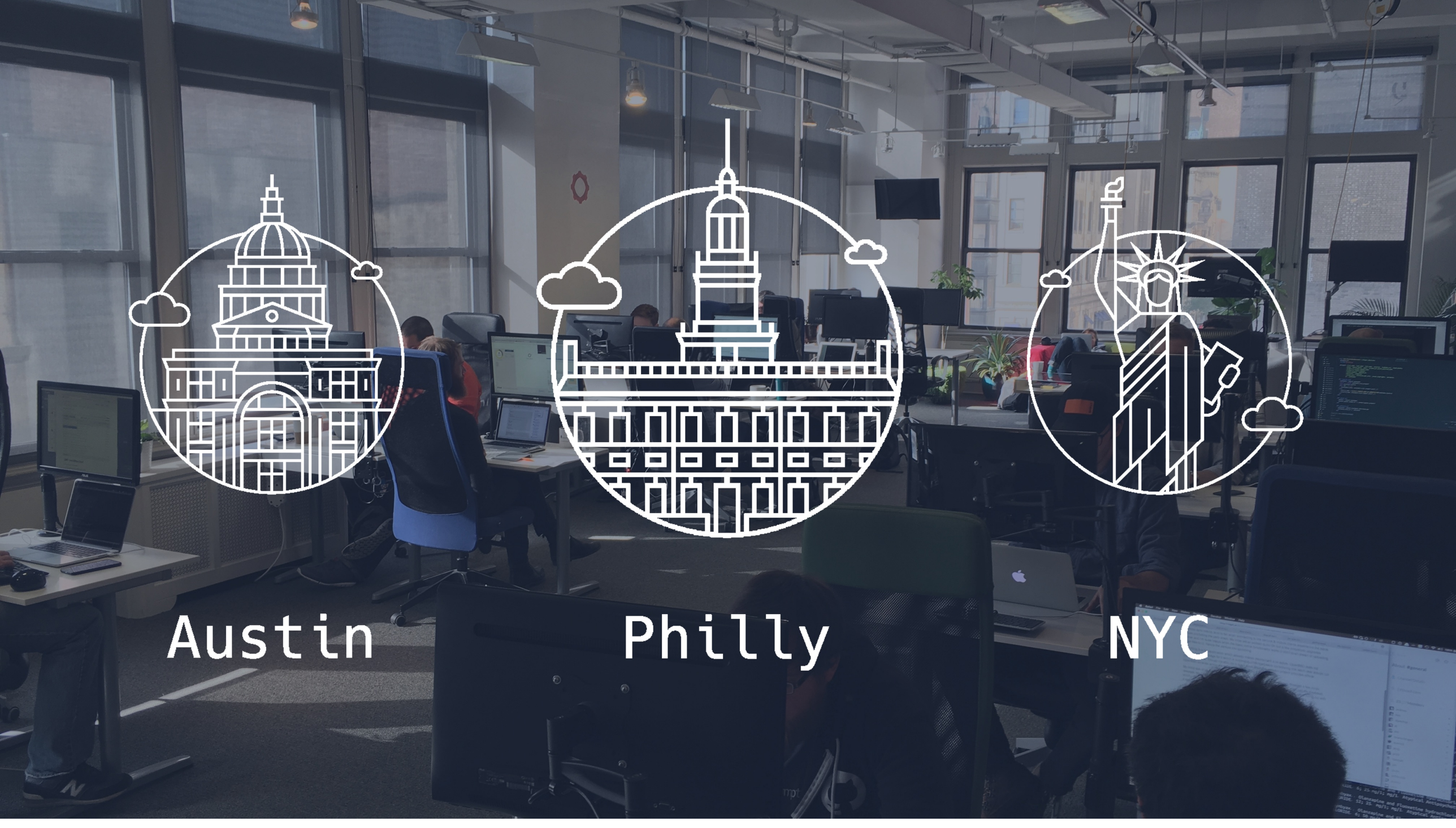
My name's Dustin Ingram

I work at PromptWorks, a software consultancy in Philadelphia


Austin
Philly
NYC

Swiss Train Deployments

Let's start with a story

Five years ago, I was traveling europe and spent a few days in Switzerland with some family friends

The first night, my host offered to take me to his office in downtown Bern the next day

Swisscom Brain Gym

The next morning, as we were walking from his house to a light rail station just like this (including the beautiful snow-capped-peaks), I saw the downtown train heading into the station

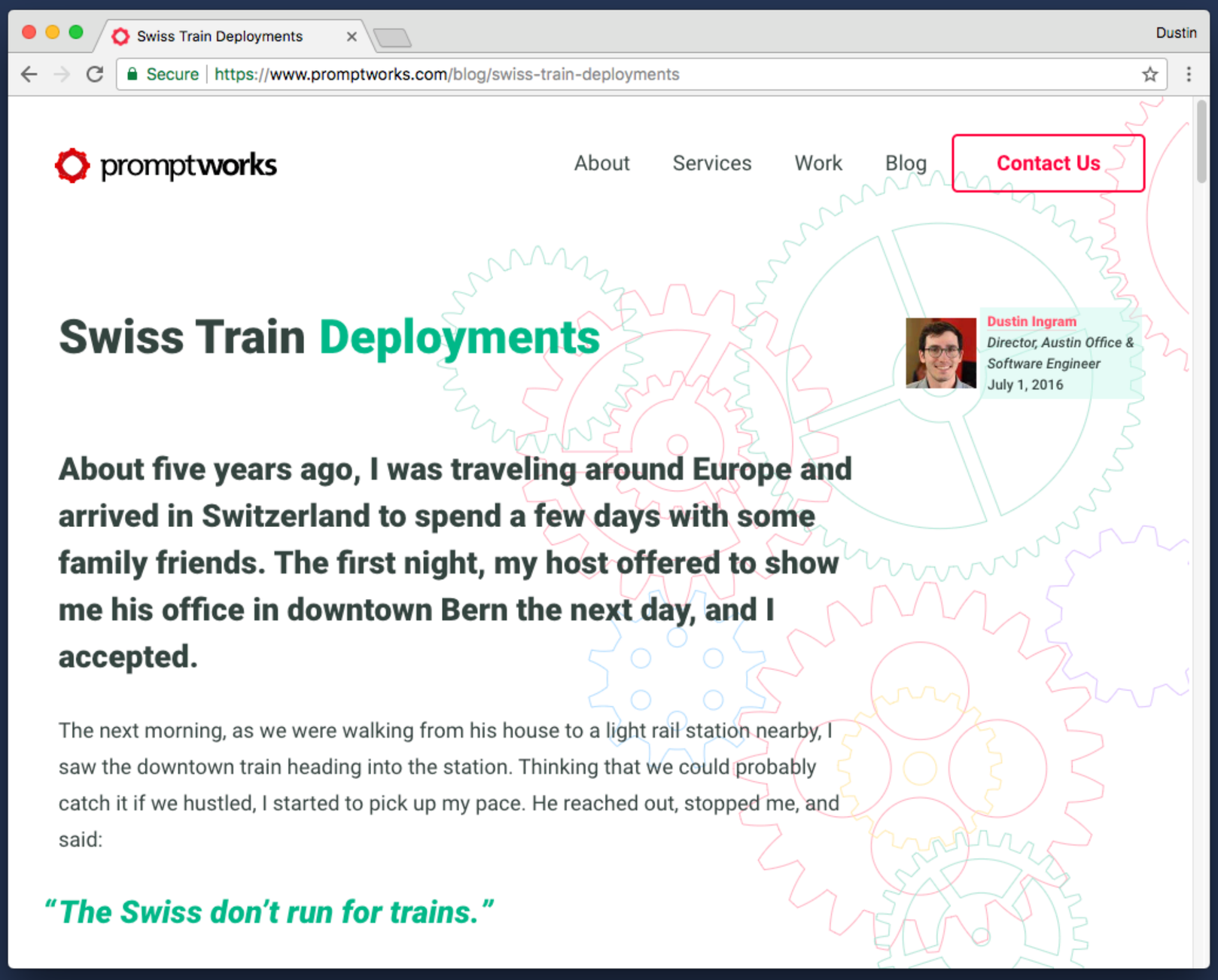
Thinking we could catch it if we hustled, I started to pick up my pace

He reached out, stopped me, and said:

"The Swiss don't run for trains."

As we walked towards the station, we watched the train that we could have caught pull away

We sat there for ~12 minutes until the next one arrived, and continued our journey



Now, this talk is also a blog post I've authored on the promptworks website

It inevitably got posted to reddit, where this was the first comment

[–] **_INTER_** 0 points 10 months ago

> The Swiss don’t run for trains.

sure do

permalink **embed**

Obviously I don't mean to generalize

This thought has always stuck with me as sort of a personal philosophy

The idea is that life is already moving pretty fast

And that there's no need to rush it any more

But on top of that, have you ever *see**n*** anyone run for a train?

They inevitably look like an idiot

EMERGING TECHNOLOGIES
FOR THE ENTERPRISE CONFERENCE

"Stability Without Stagnation:
Lessons Learned Shipping Ember'

Yehuda Katz

Day 1 - April 11, 2016 - 11:30 AM - Salon C

phillyemergingtech.com

In April I was at Philly ETE, and one of the best talks there was Yehuda Katz's talk

This is Yehuda. Most python devs aren't familiar with him, but he's a proflific developer and influential in a number of open source projects

rails, jquery, rust, and ember

Stability Without Stagnation

Yehuda's talk was called Stability without Stagnation

In it, he described a deployment process which enables rapid development, but still provides a stable platform for the people using your technology

He called his philosophy "Stability without Stagnation"

Basically, the SwS philosophy boiled down to two things, first...

#1: Ship Regularly

(Every Six Weeks)

For ember, this meant making a release every six weeks

No matter what's going on with the project

#2: Release Channels

Canary

Canary - features ASAP, no guarantees

Code might have been written the same day as the release

Beta - features after 6 weeks, bugs shook out

Release

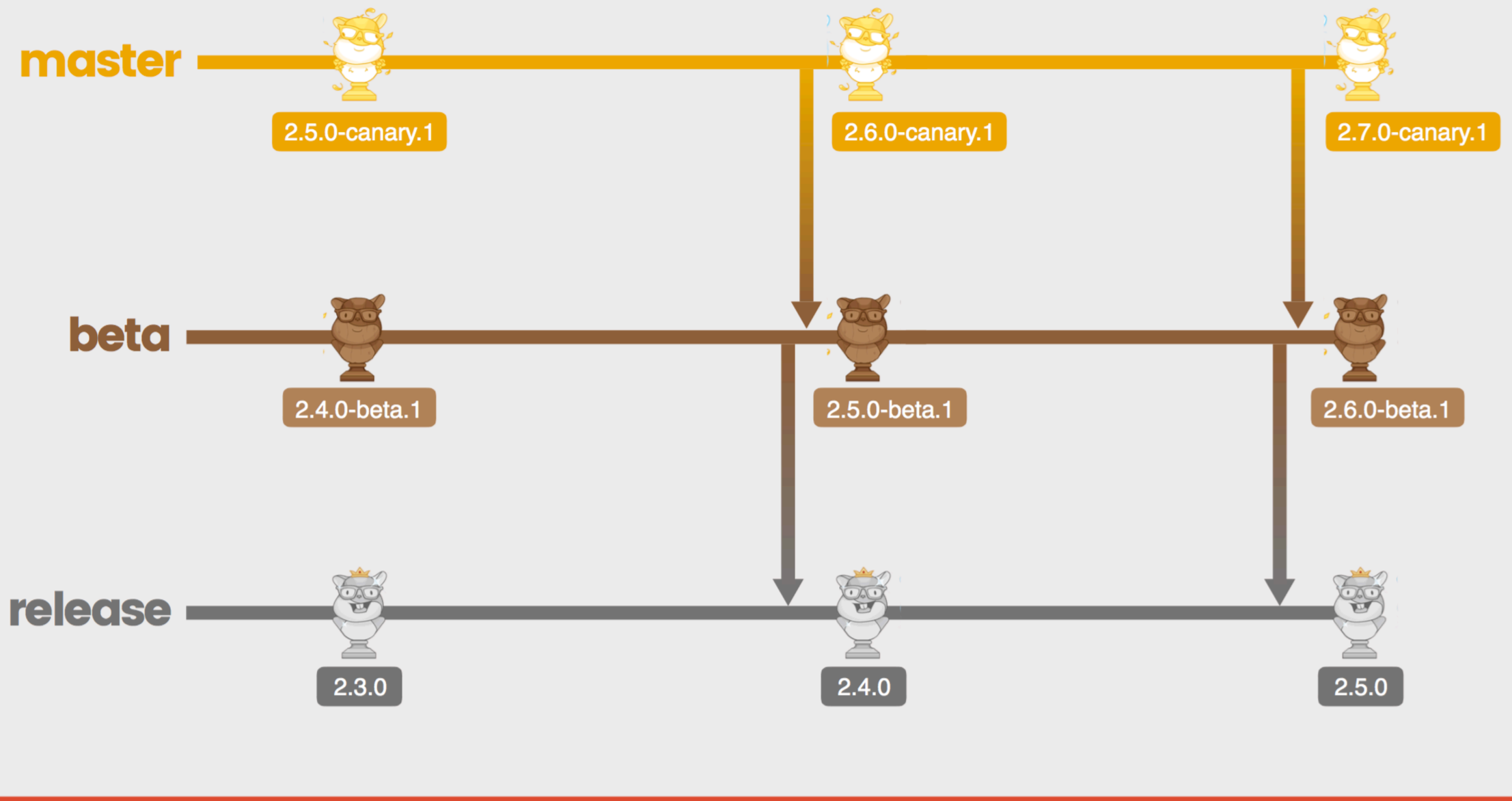
Release - features after 12 weeks, semver backwards-compatible guarantees

These are essentially tagged branches in your source

They balance new features with stability.

Release cycle means every channel cuts a release every cycle

Regardless of how many features have been shipped!
Could have nothing

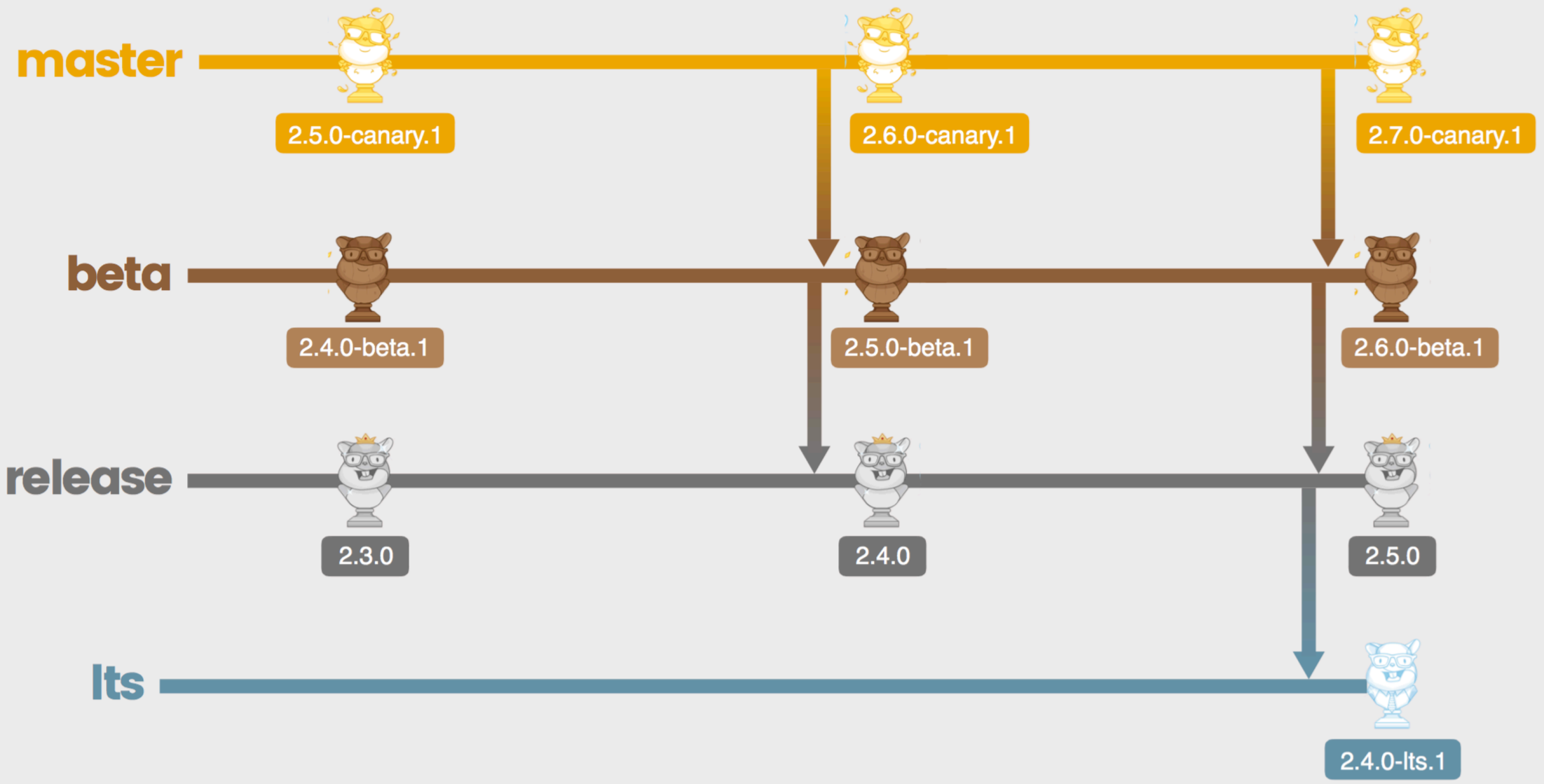


Looks something like this

A given feature is deployed to each channel, in order, one release after another

Part two is...

Have Multiple "speeds"



And making that channel's releases take 4x as long

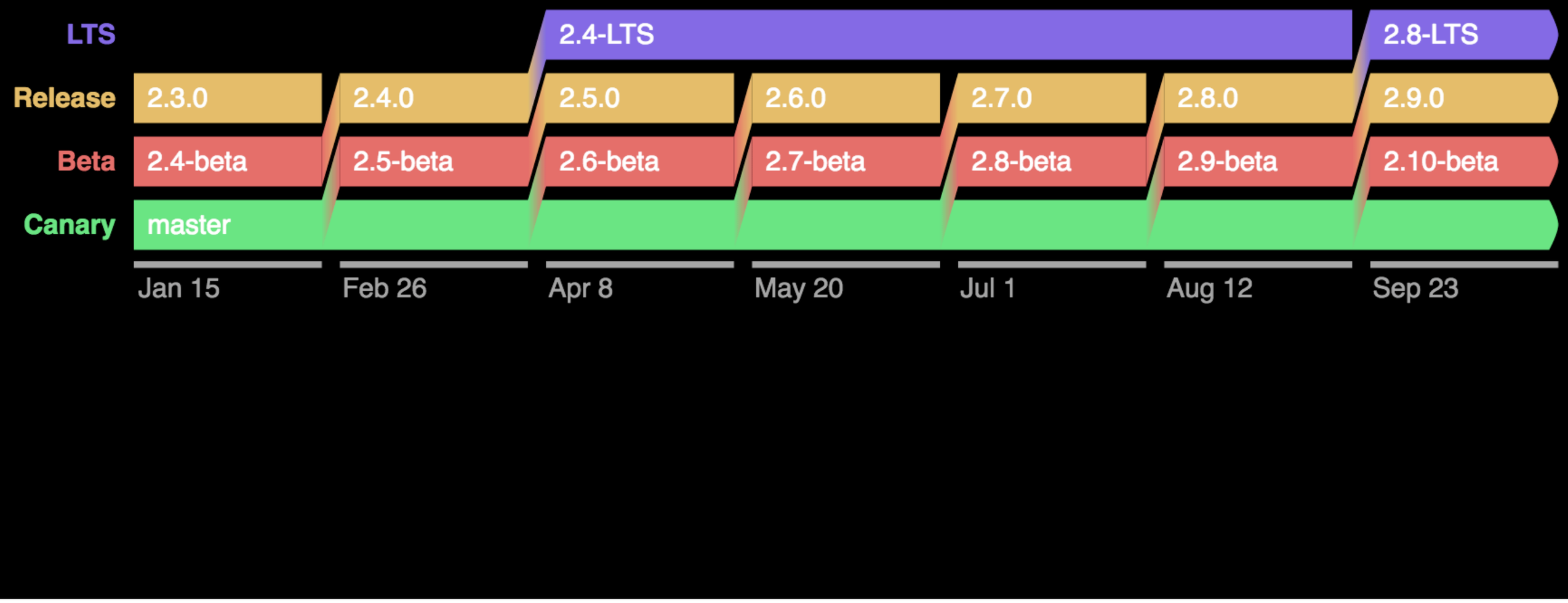
This basically amounts to an LTS release, which is a common practice in the linux/unix community

If the six-week releases are happening too quickly, the LTS release is guaranteed to exist and be stable for a longer window of time

Also, you get bugfixes for the entirety of the LTS release

Also, breaking changes are first deprecated in an LTS release

Side note: if the thought of making your release process conform to this diagram frightens you, you're doing it wrong



Here's another diagram showing how LTS was introduced

So there are some benefits to this process

Benefits

...to Maintainers

Maintainers don't feel any rush to ship a feature in for a given release. If they miss one, there's another one in six weeks.

"Ship this feature ASAP because we don't know when we'll release next!"

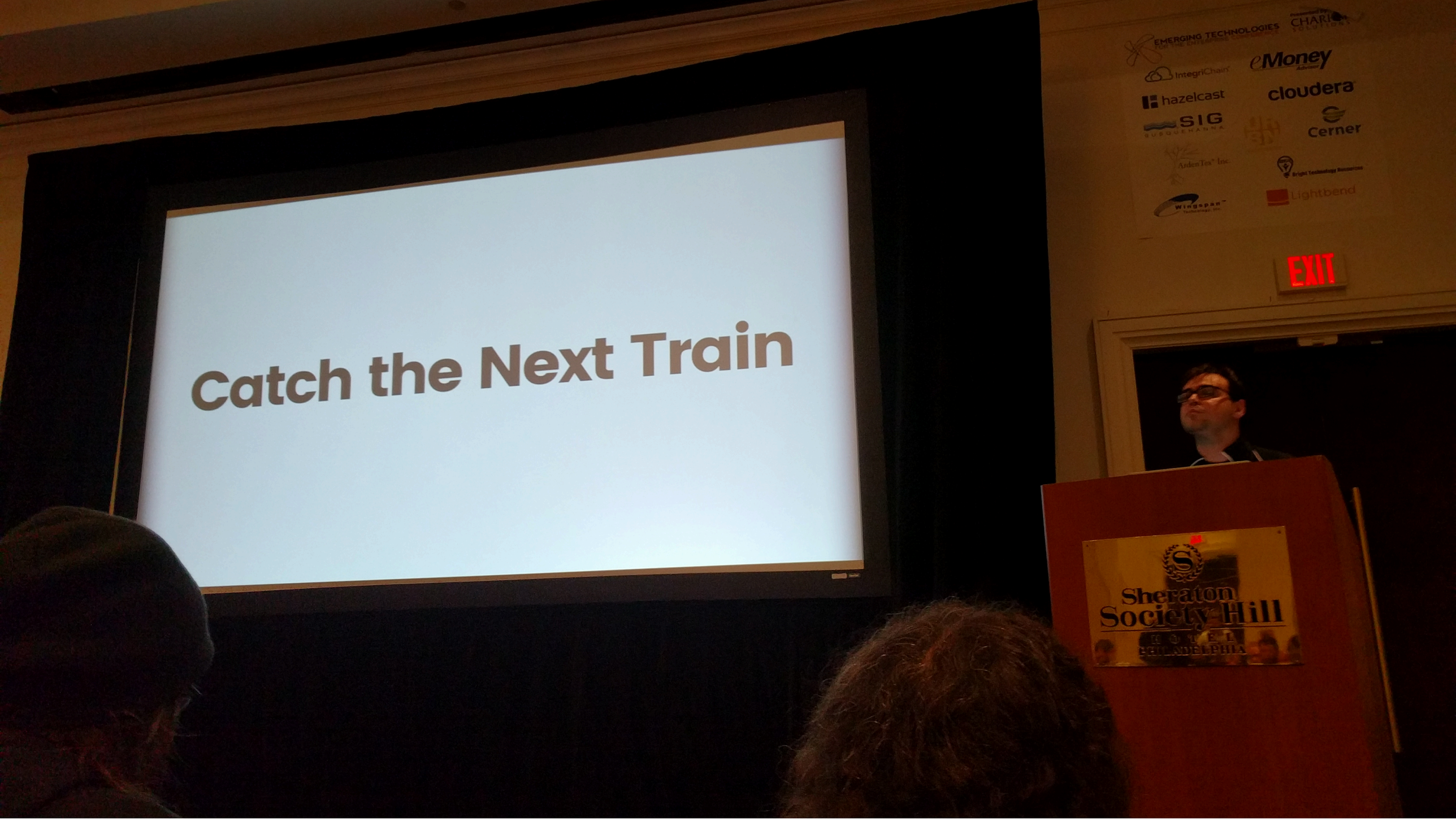
...to Contributors

Contributors can predict when their feature will be released and what the process is.

...to Add-on Authors & Developers

End users, which yehuda splits into add-on authors and developers, can make a decision about their risk tolerance easily and build against an API that they know won't change unexpectedly

Yehuda summed that up by saying that everyone can just:



The swiss don't run for trains because they want to take it easy

They don't run for trains because they're concerned about their image

They don't run for trains becaues they know there's another one coming

And because they know that, they don't stress about catching trains, and they don't look dumb

I'm from Philly

In philly, people definitely run for their trains

Why do people run for trains in Philly?

Because we're running late, maybe, but probably

Because we don't know when the next one's going to come.

We can't fix SEPTA, but we can fix our own deploy processes

Ember → You

Obviously, we are not all writing OSS with lots of contributors, users, in an early stage of development

But there are direct paralells to all kinds of development processes

Benefits

...to Management

...to the Developers

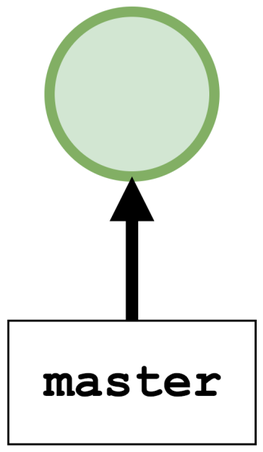
...to the Users

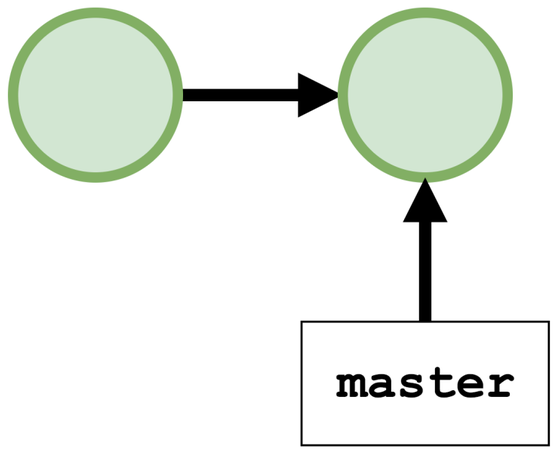
Requirements:

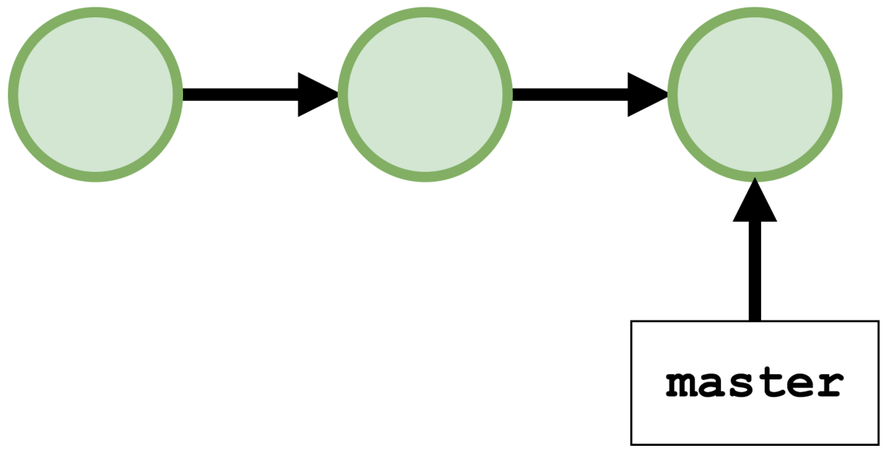
A few basic requirements you probably already have

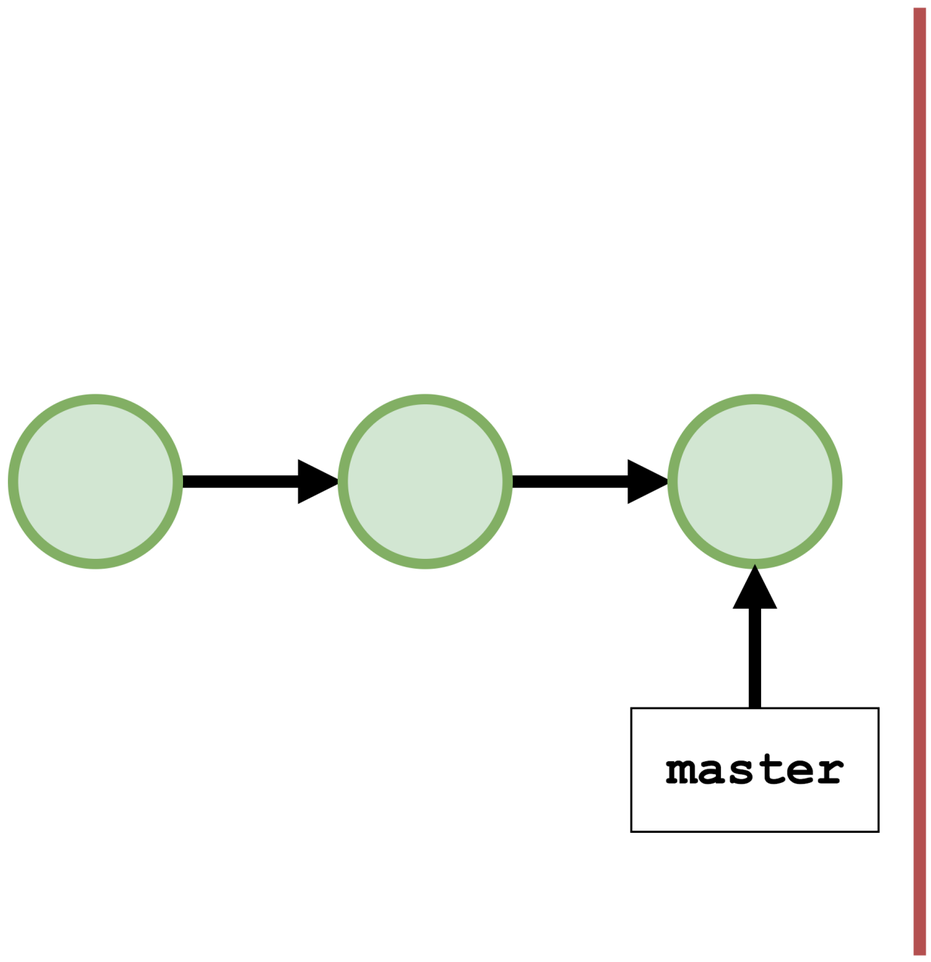
1. A reasonable VCS

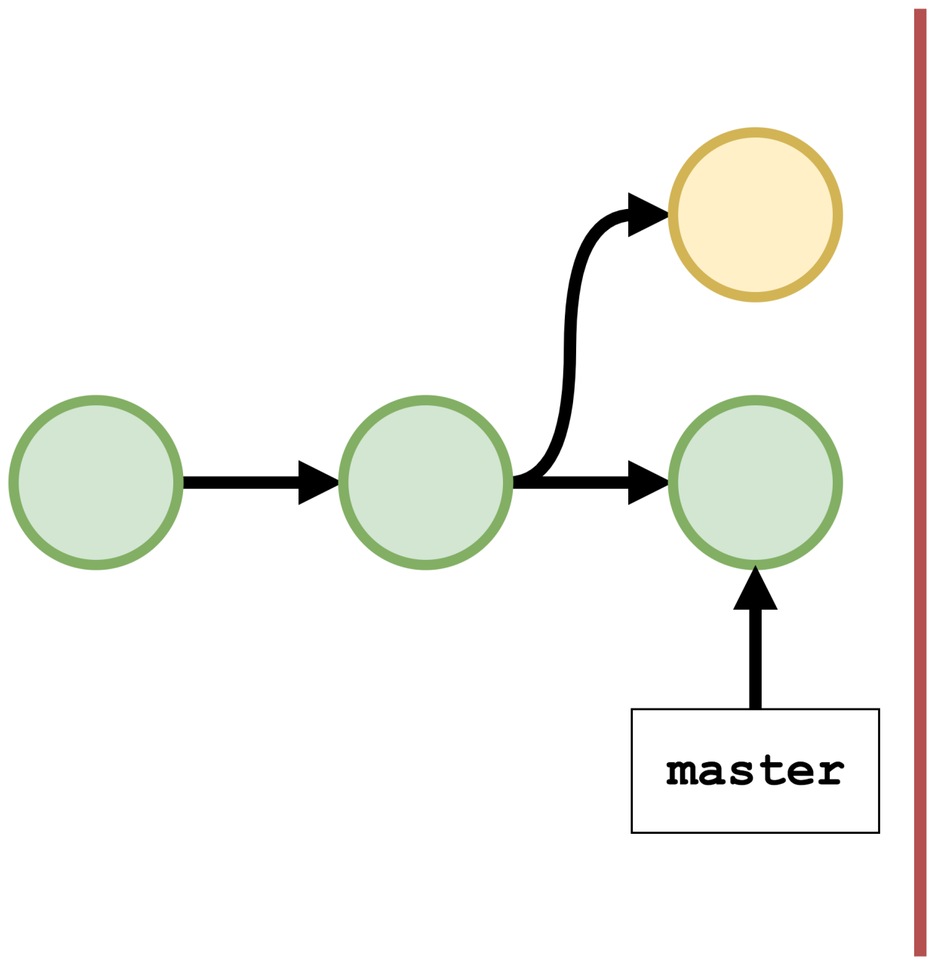
You need a VCS that lets you have branches and tags, or some equivalent *cough* git *cough*

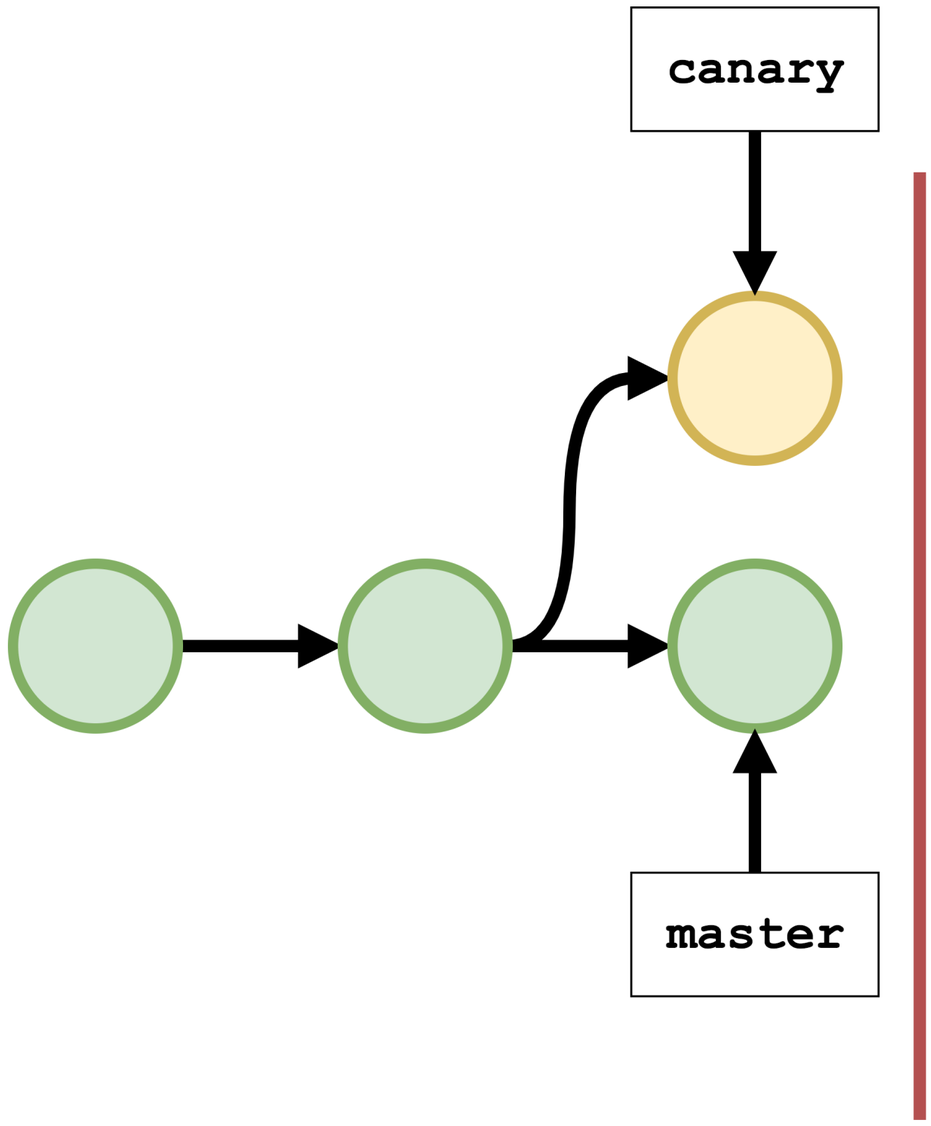

master

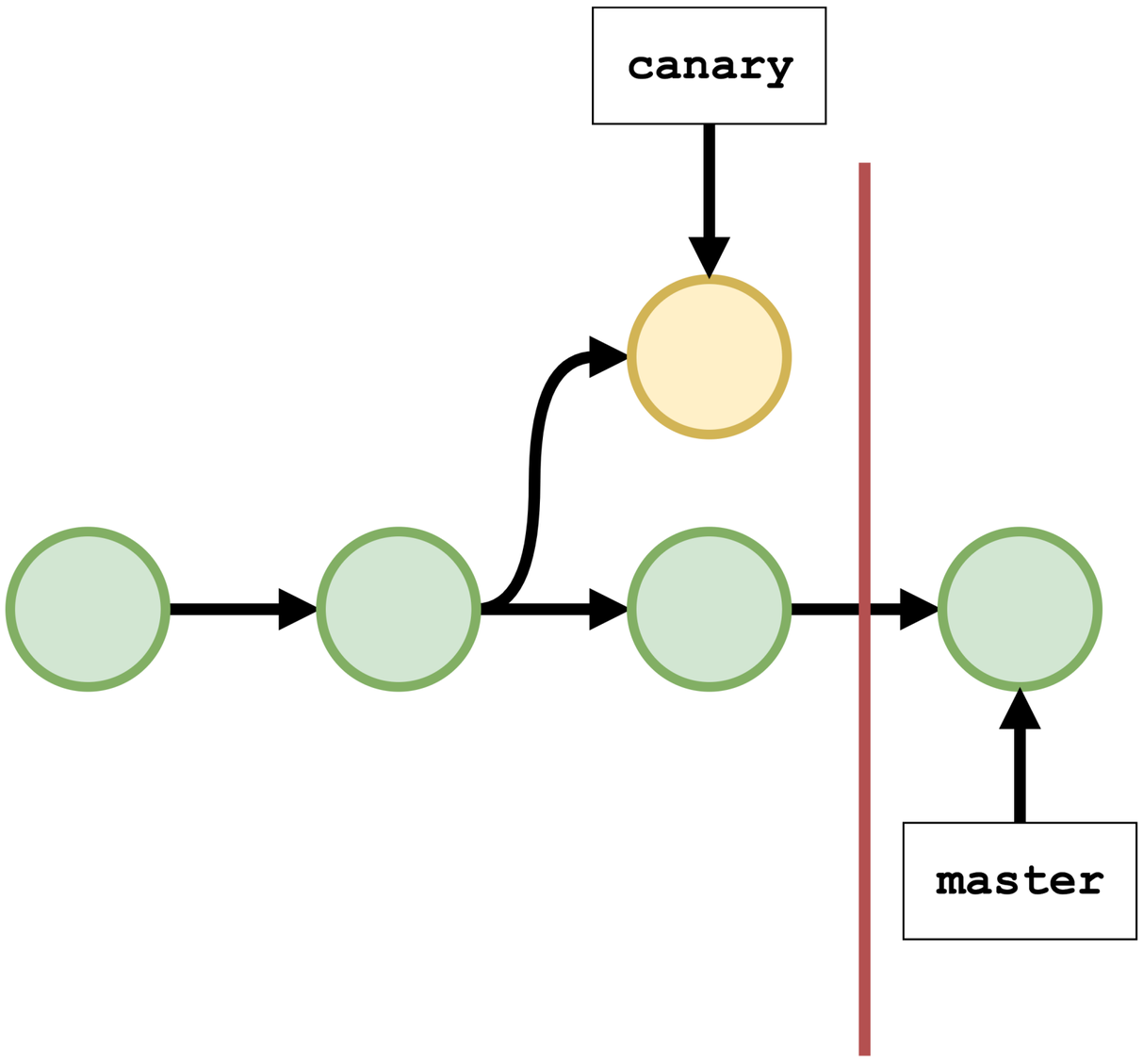

master

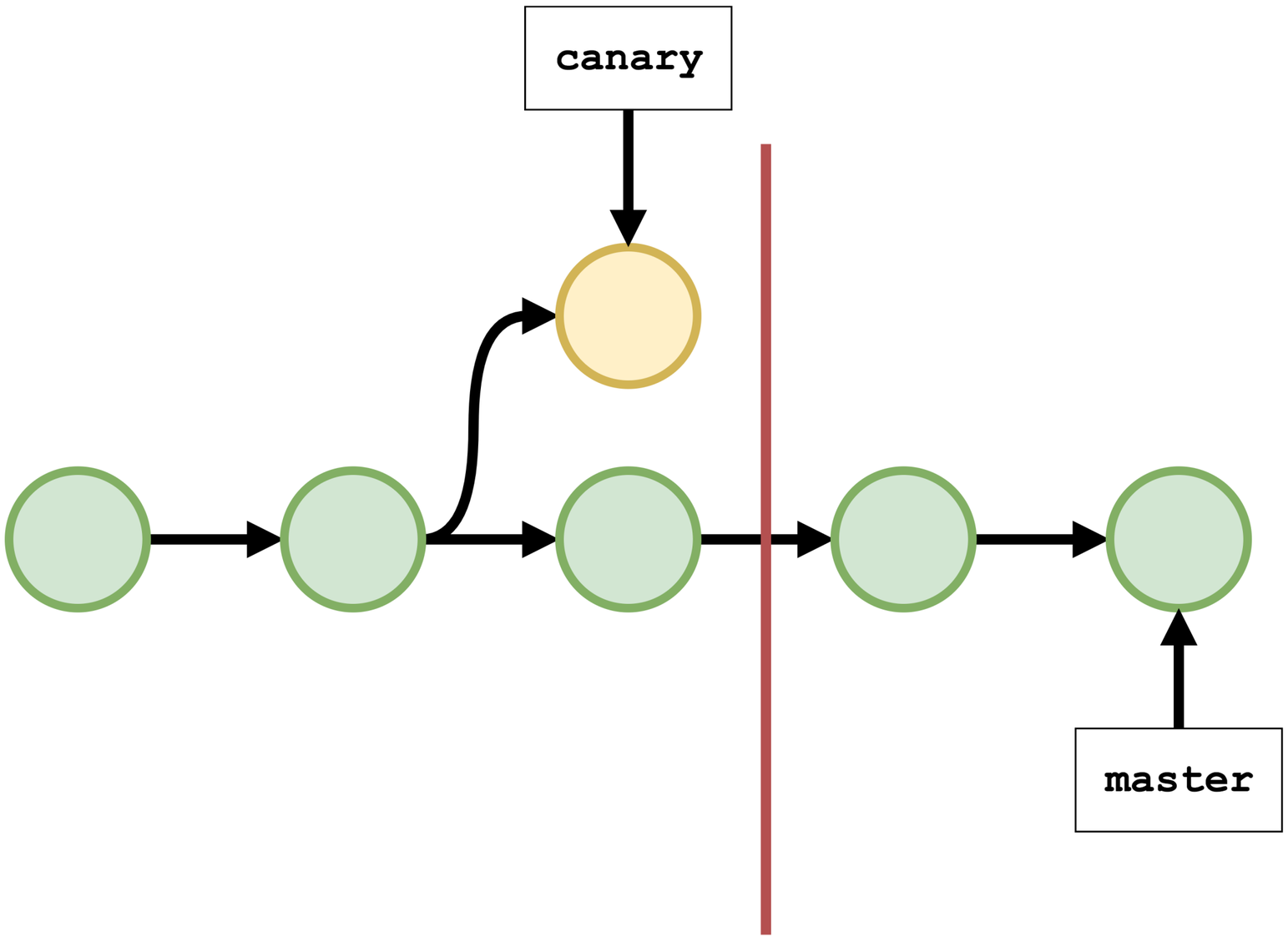

master

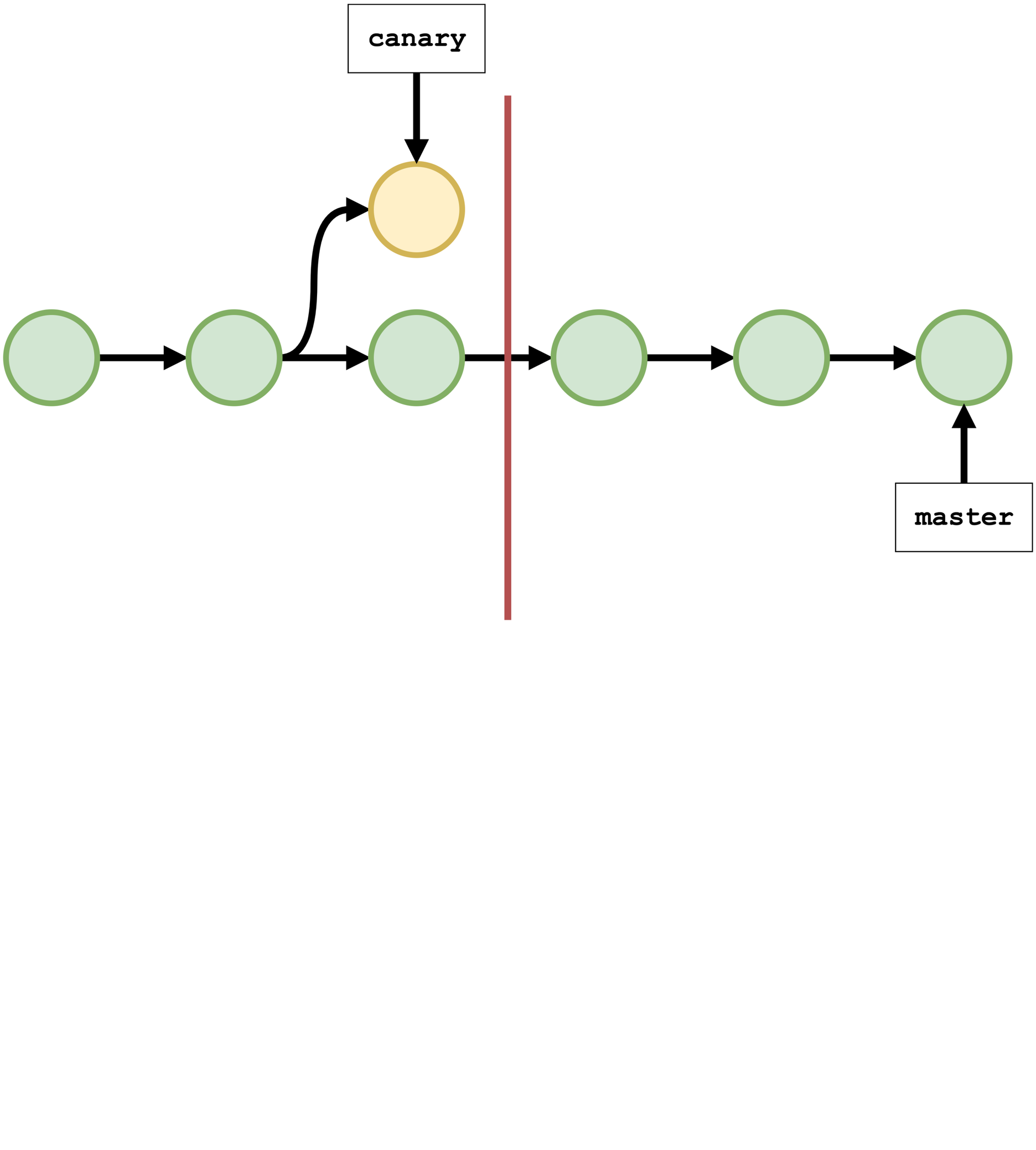

master

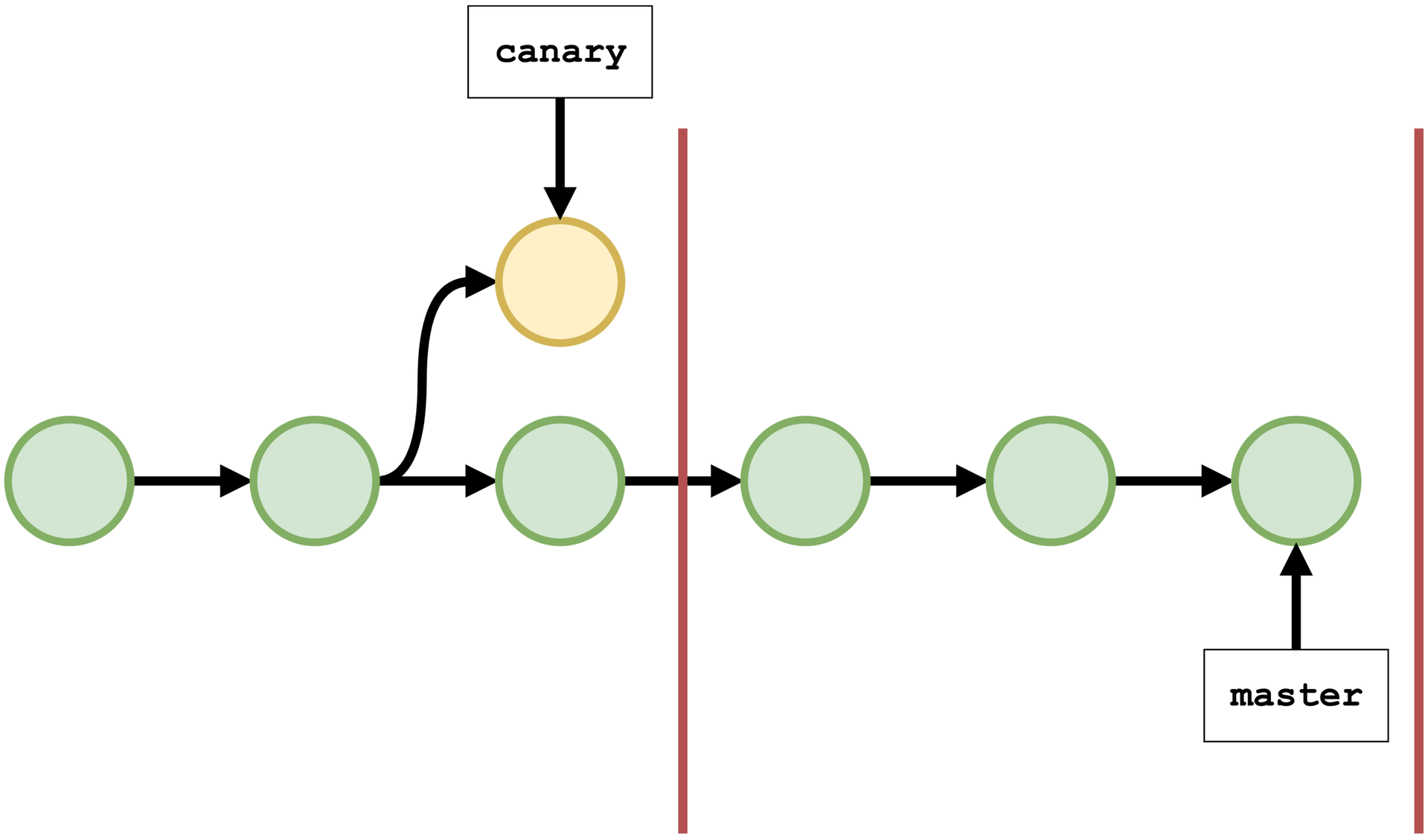

master

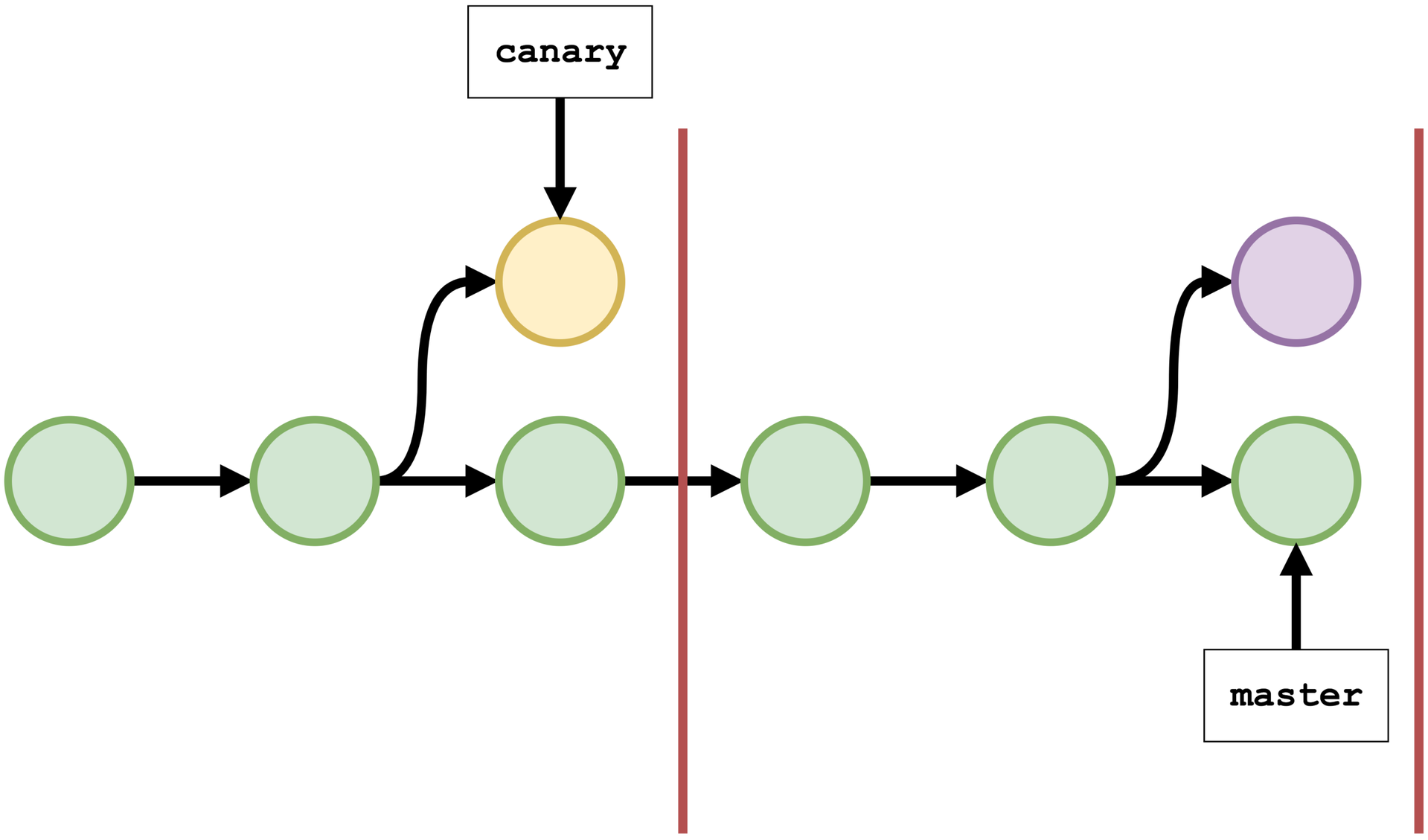

canary
master

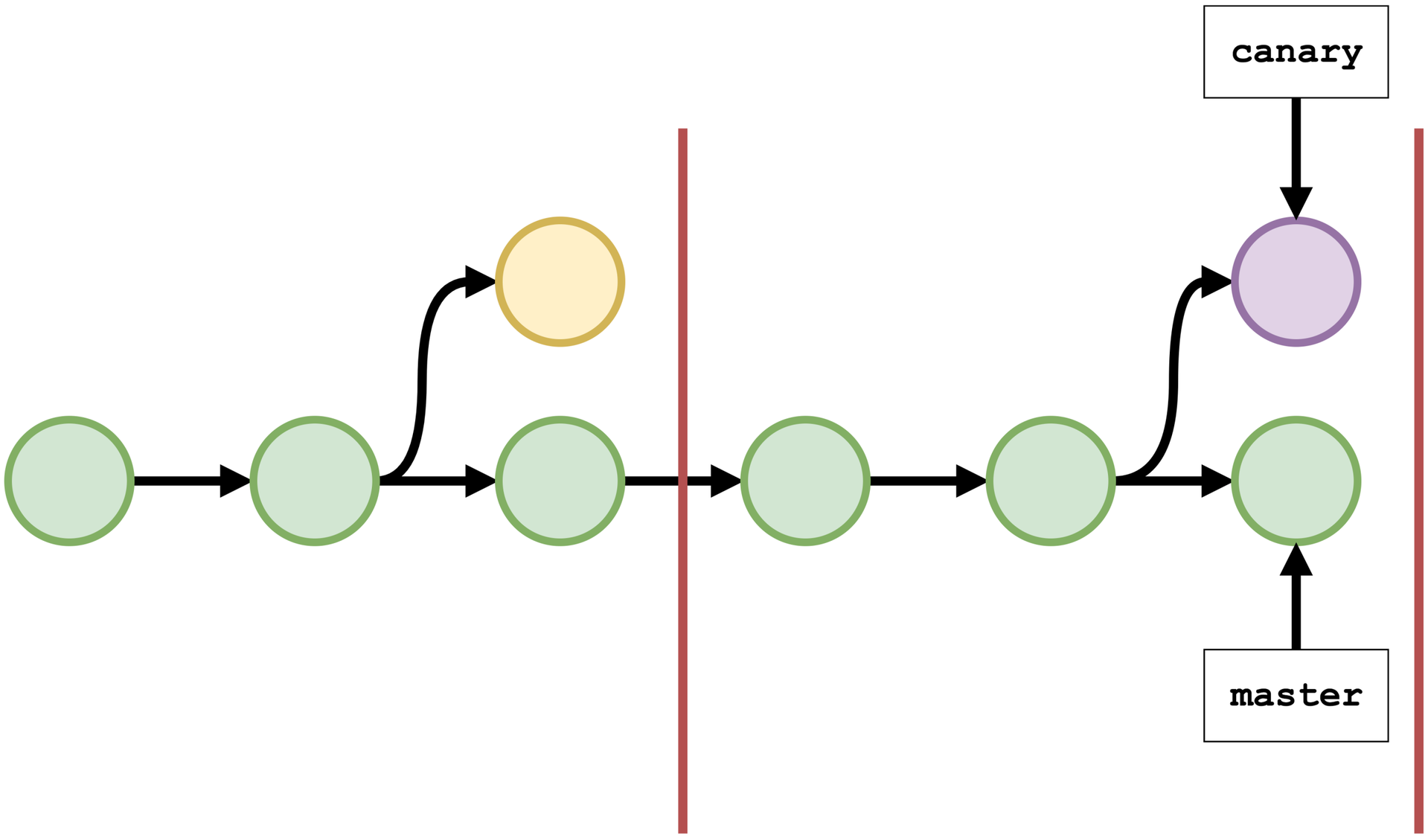

canary
master

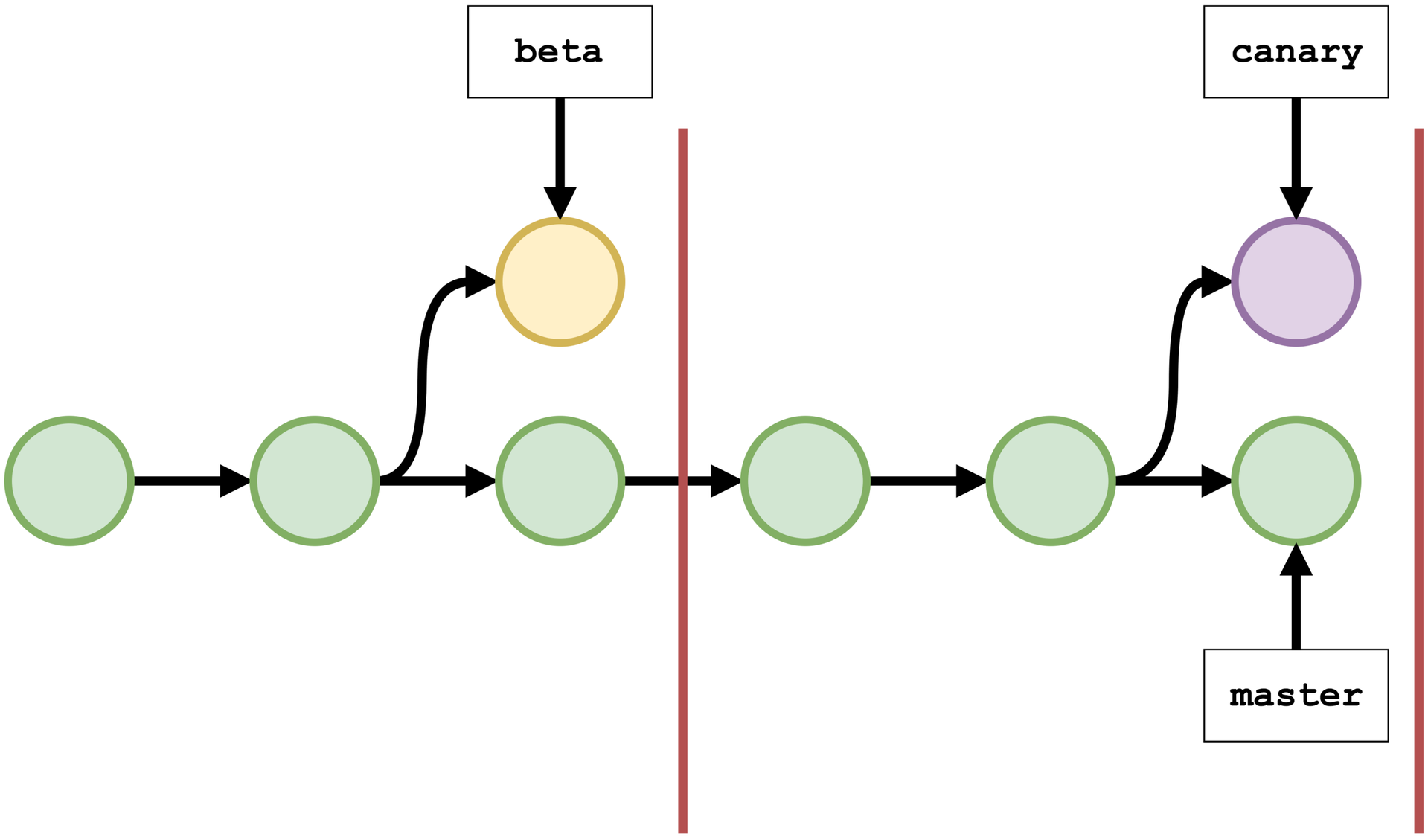

canary
master

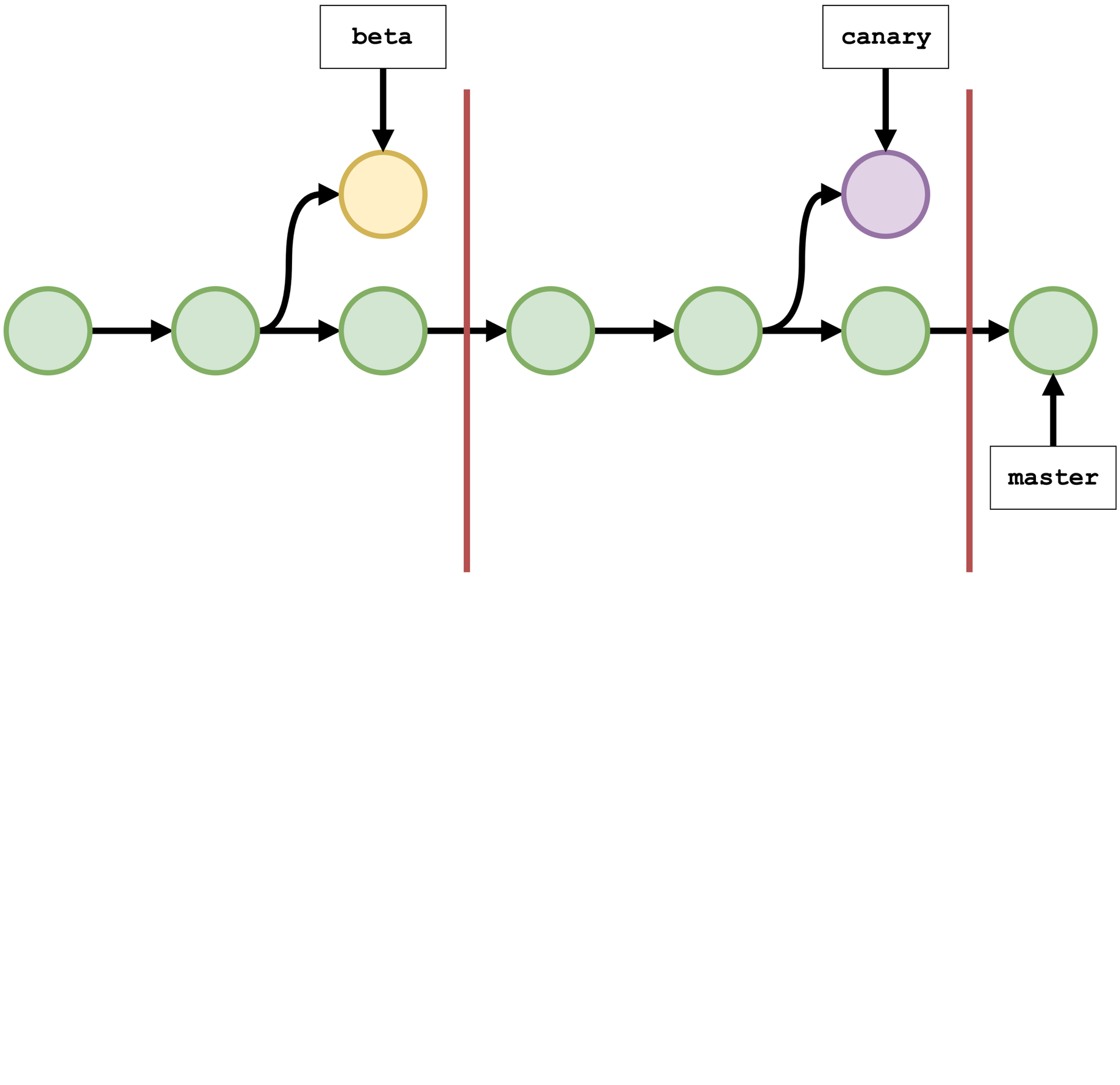

canary
master

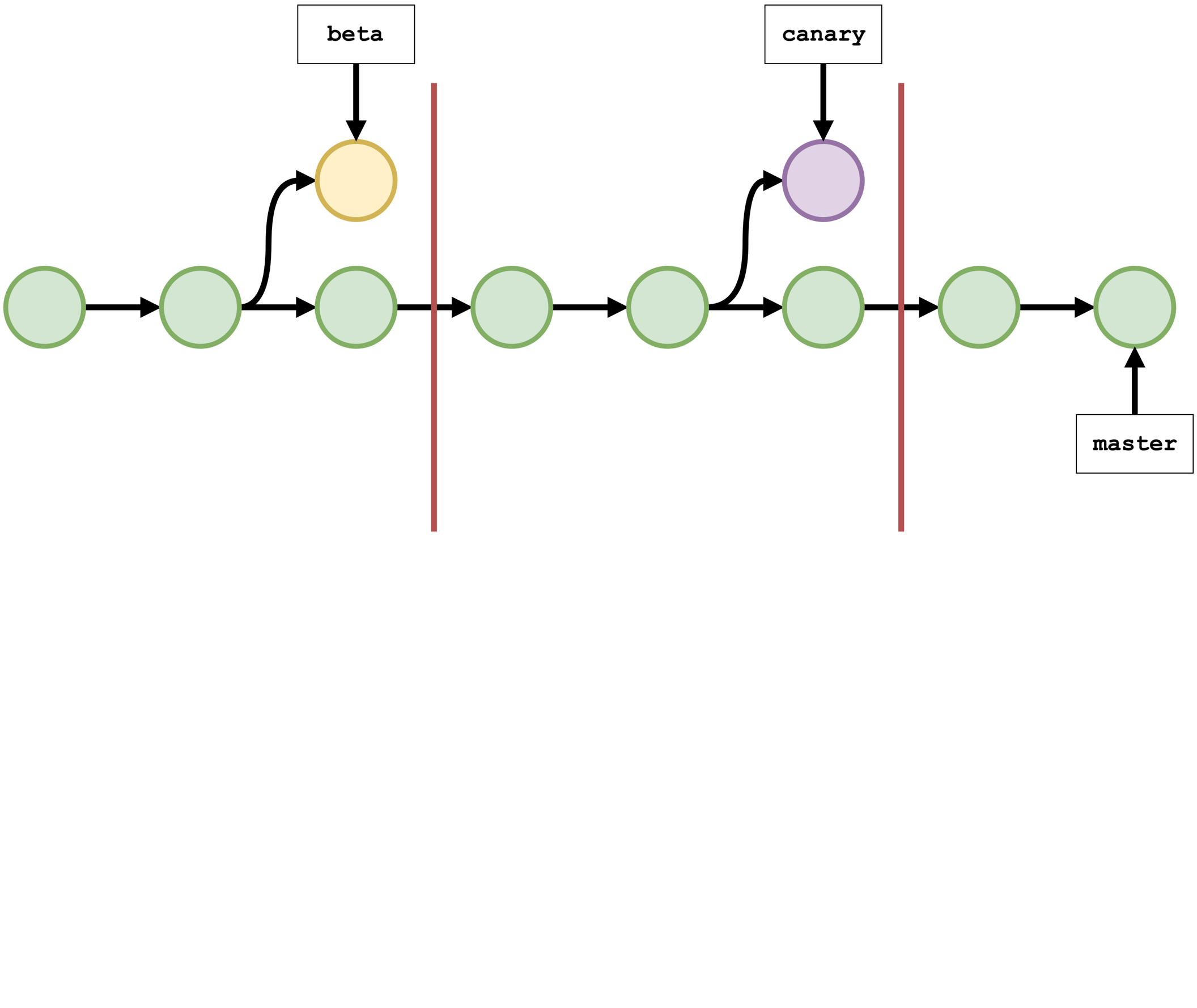

canary
master

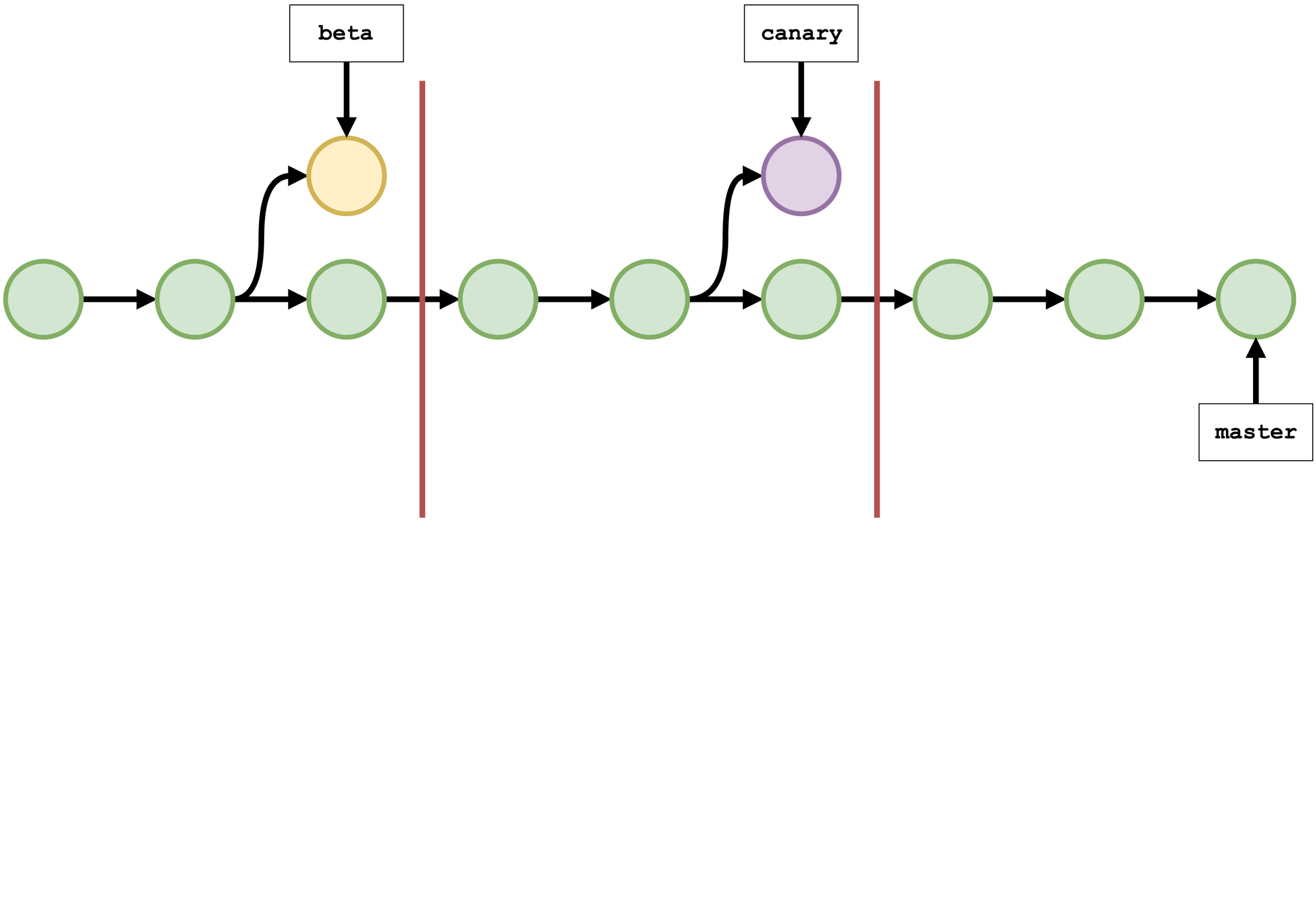

canary
master

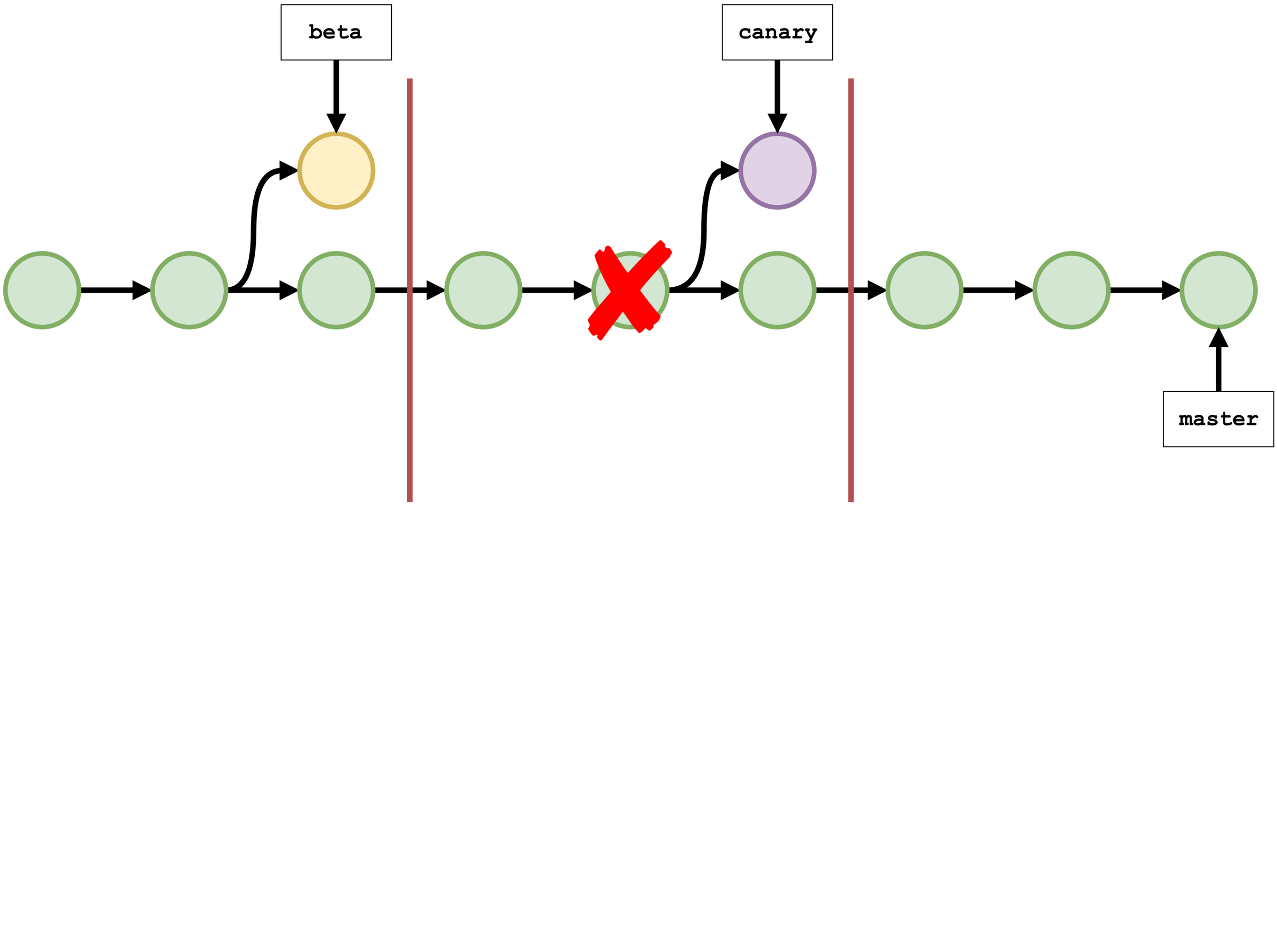

canary
master

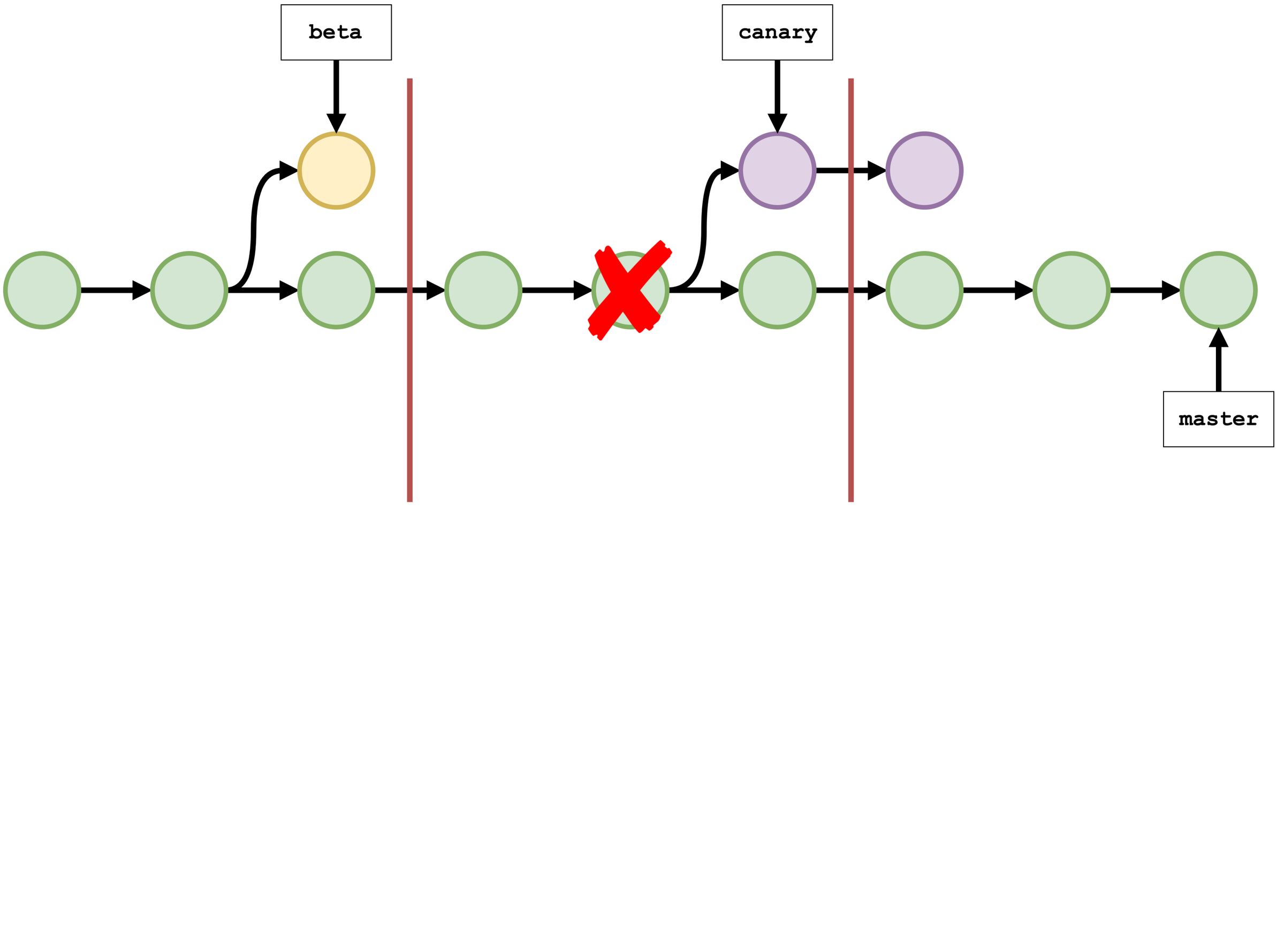

beta
canary
master

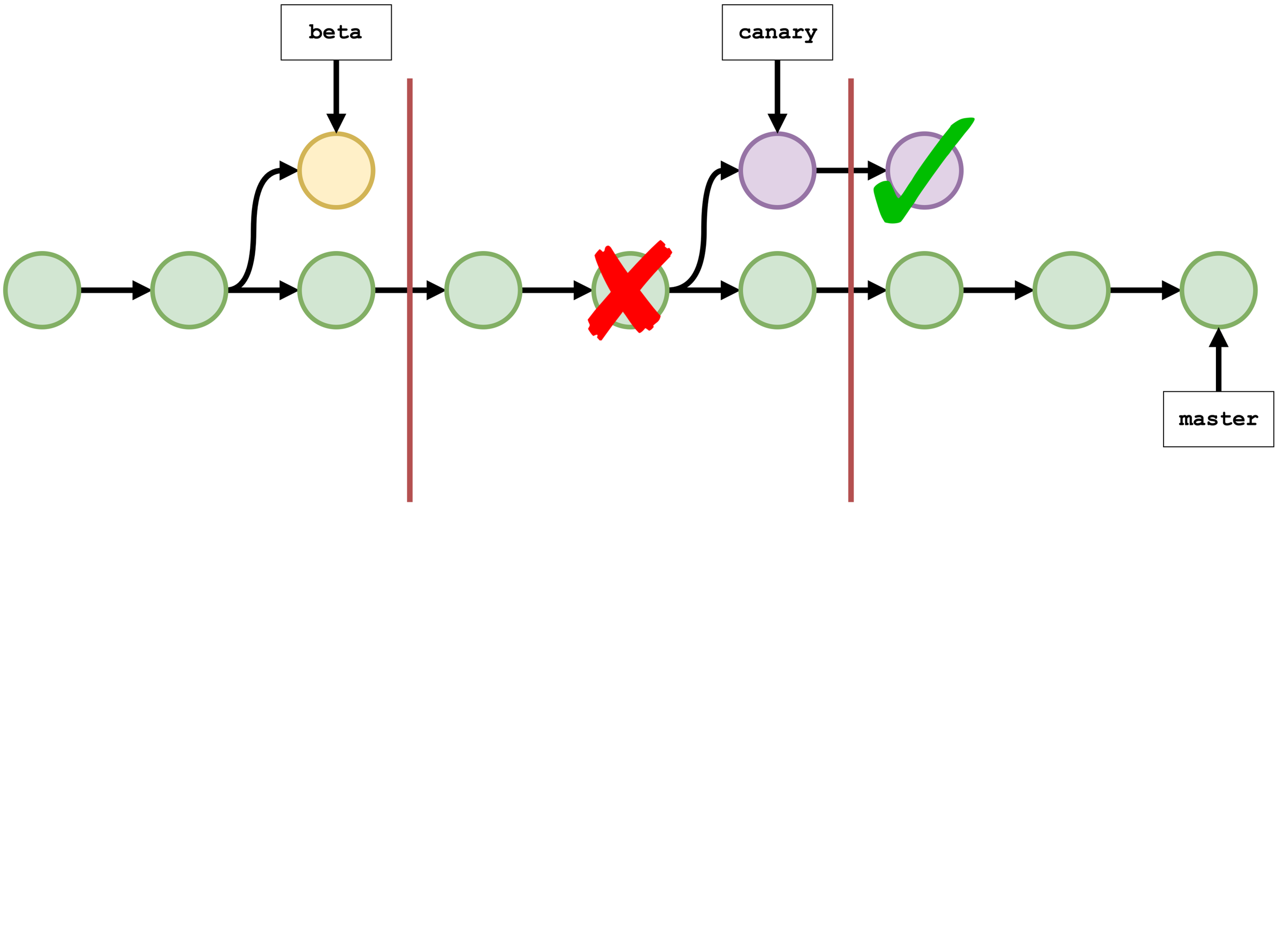

beta
canary
master

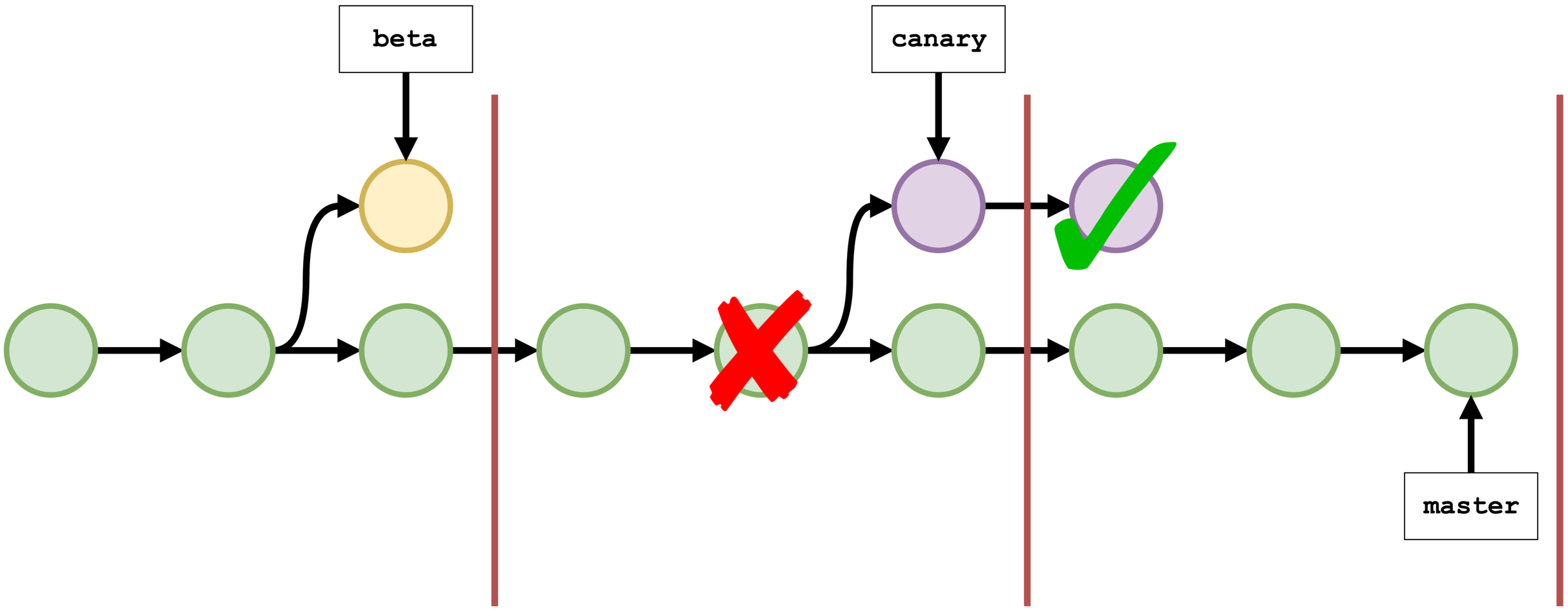

beta
canary
master

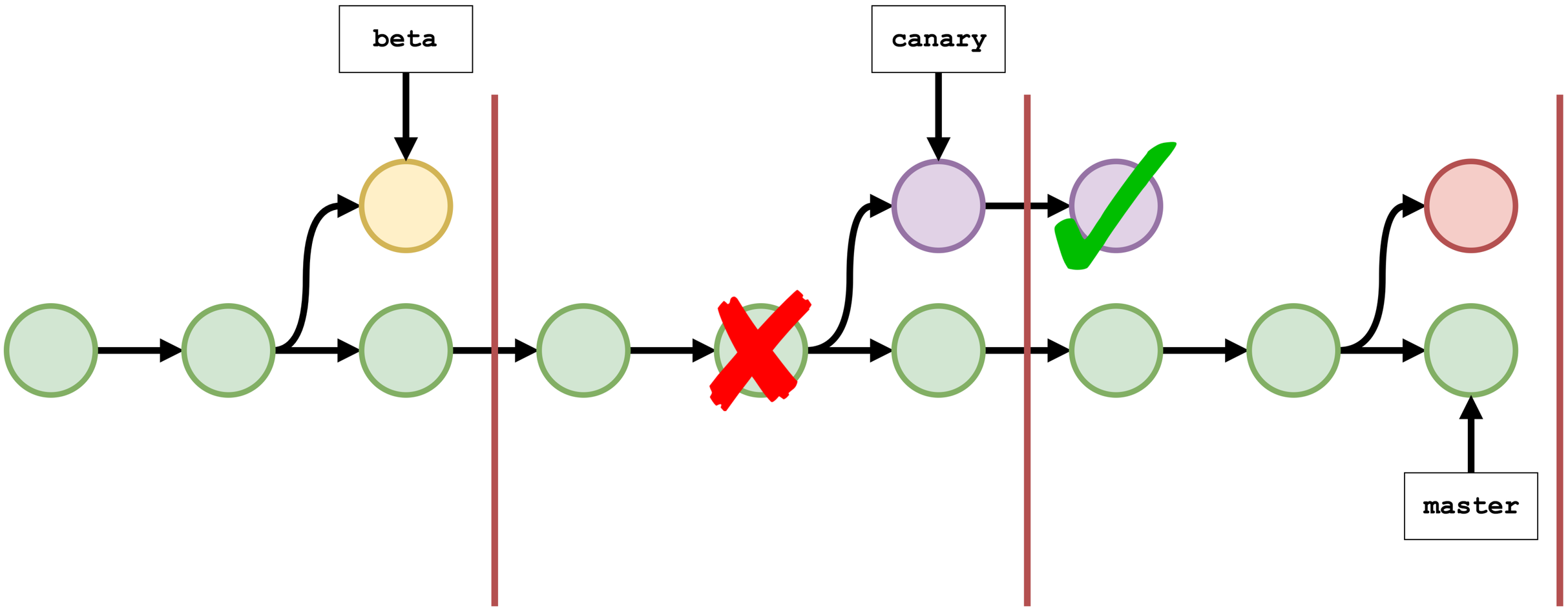

beta
canary
master

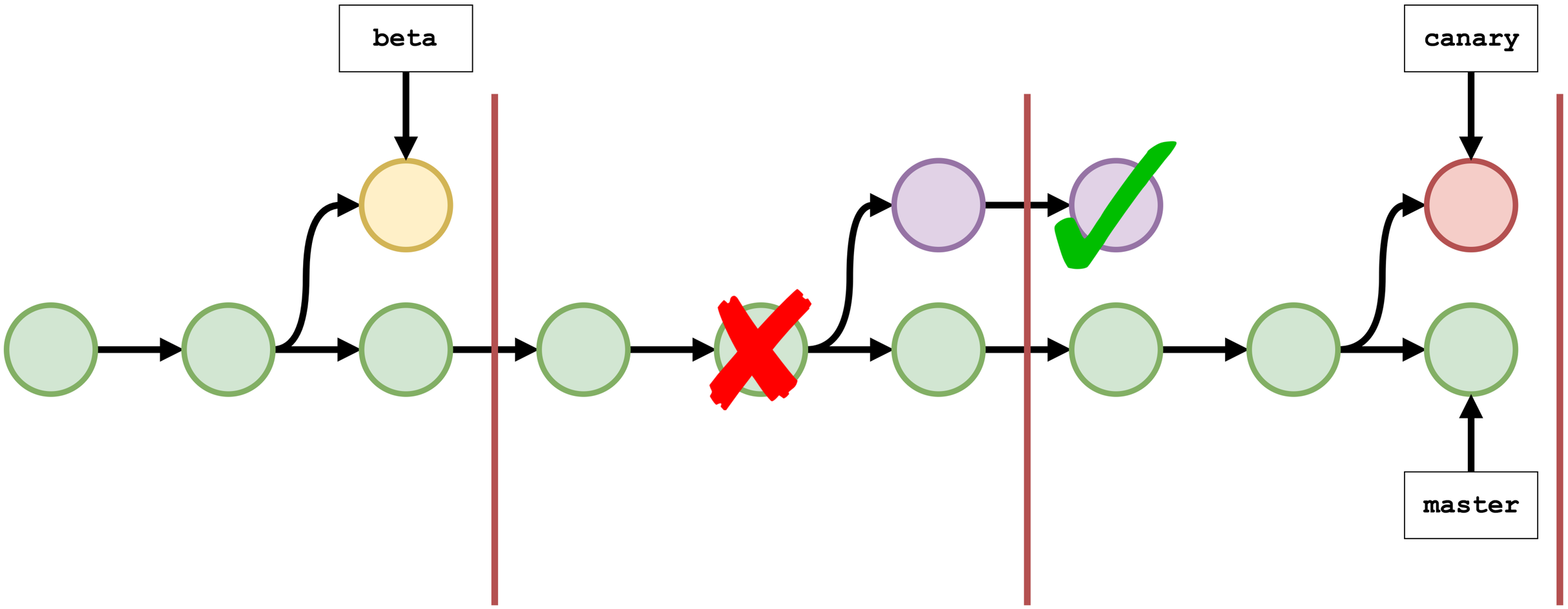

beta
canary
master

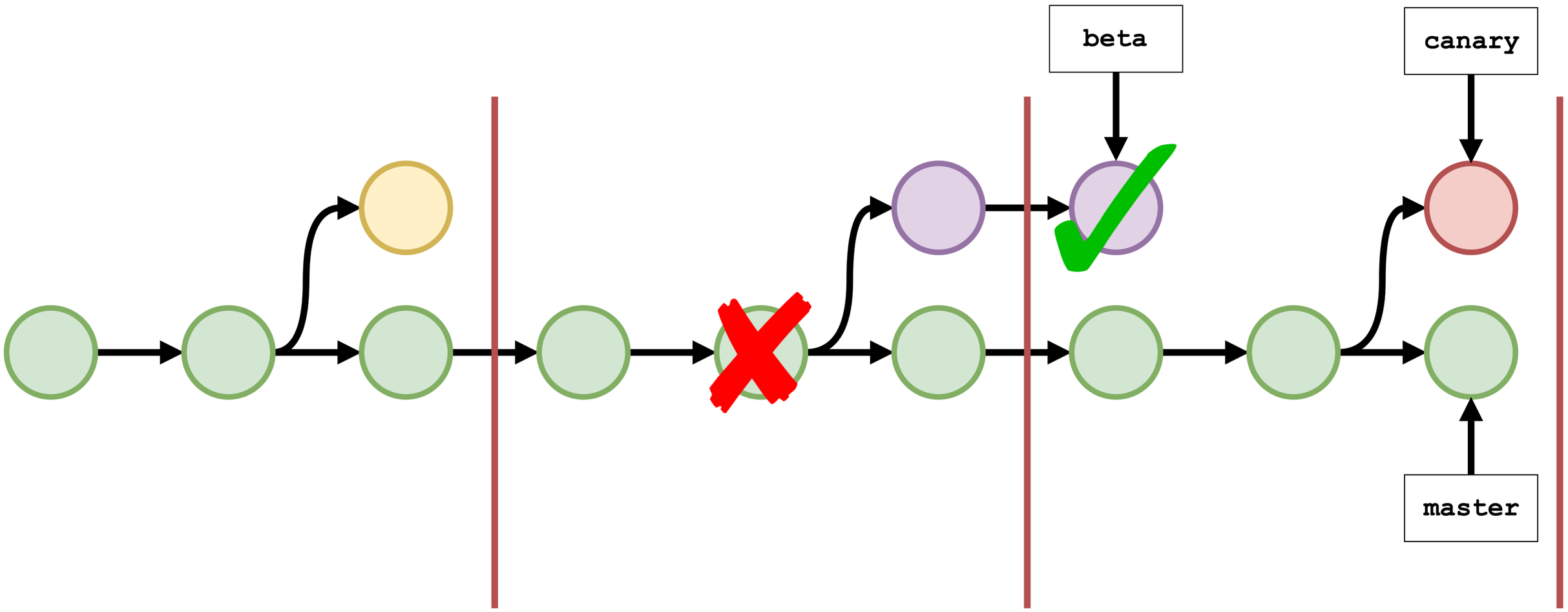

beta
canary
master

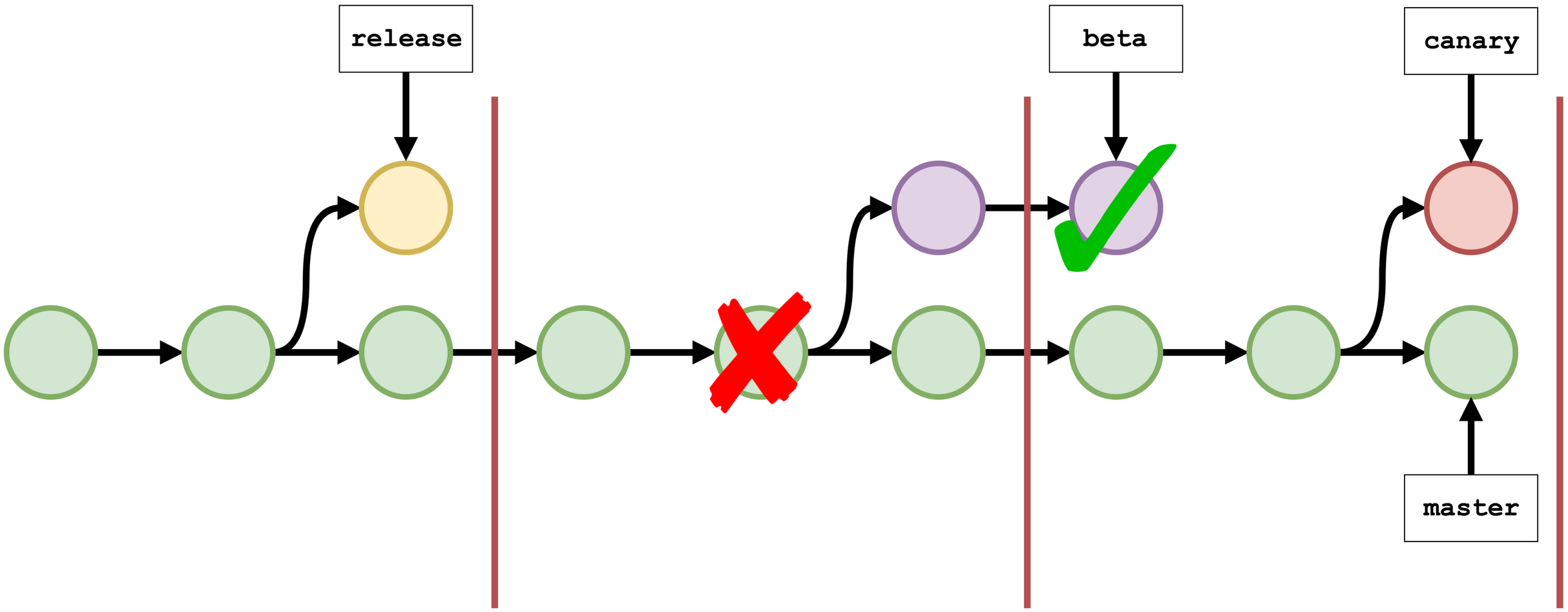

beta
canary
master


beta
canary
master


beta
canary
master


beta
canary
master


beta
canary
master


release
beta
canary
master

2. Automation

You need some kind of automation around deployment (and probably testing too)

Travis CI - Test and Deploy Yo

di@promptworks

Travis CI GmbH [DE] | https://travis-ci.org

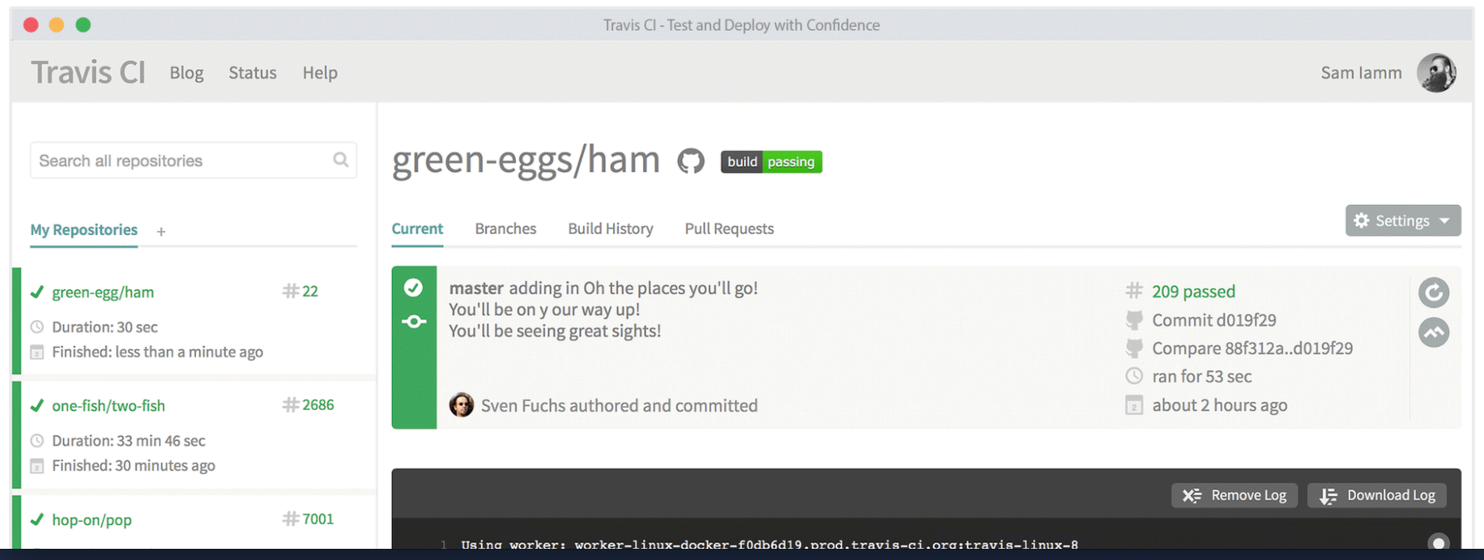
Travis CI Blog Status Help

Sign in with GitHub

Test and Deploy with Confidence

Easily sync your GitHub projects with Travis CI and you'll be testing your code in minutes!

Sign Up



Continuous Integration and De ×

di@promptworks

Secure | https://circleci.com

circleci Product Pricing Enterprise Developers Company Support | Log In **Sign Up**

Build Faster.
Test More.
Fail Less.

Automate the software development process using continuous integration and continuous delivery so you can focus on what matters—building great things—not waiting for great things to build.

Start Building for Free | Explore the Docs

facebook Segment KICKSTARTER Percolate Spotify GoPro

Jenkins

di@promptworks

Secure | https://jenkins.io

Jenkins | Blog | Documentation | Plugins | Use-cases | Participate | Sub-projects | Resources | About | Download

Jenkins

Build great things at any scale

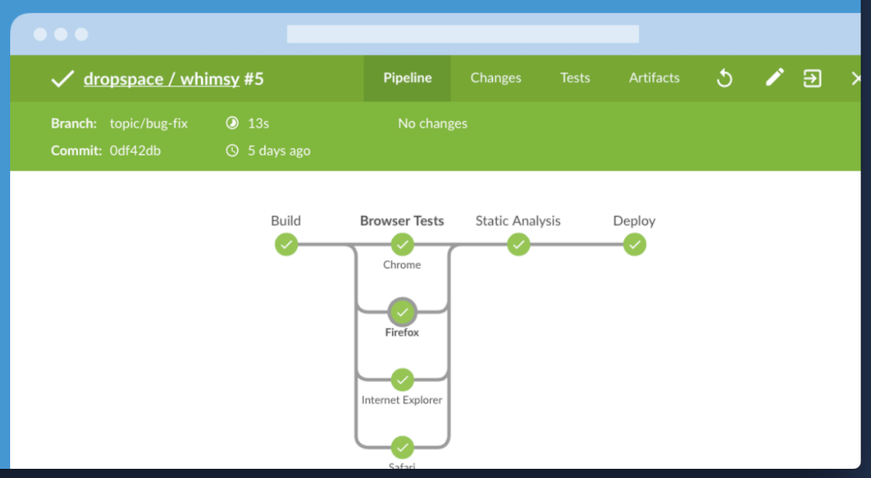
The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

Documentation | Download

Say “hello” to Blue Ocean

Blue Ocean is a new user experience that puts Continuous Delivery in reach of any team — without sacrificing the power and sophistication of Jenkins.

Get started



3. Features to ship

You need to be in active development!

Steps:

Then what do you do with it?

1. Ship each channel reguarly

This takes then "when?" out of releases! No more "when will x ship?"

2. Automate releases

This is easy when the only determinant is time. Also takes the "who?"

Both remove a lot of uncertainty and stress!

This might seem counter-intuitive with the mantra of big orgs like FB

But yehuda said they fell into the same trap

"We ourselves fell into the trap of believing the 'move fast and break things' mantra."

It's easy to do when FB says you should

"We thought that if our competitors had a feature we didn't..."

"...our users would leave en masse."

Result is paranoia about losing users

"In fact, it was our instability that alienated early adopters."

"We got a bad reputation for it."

"People criticized us because they felt burned."

Paradoxically, it causes you to lose users!

"Users don't migrate over night."

"You have a much larger window than you think..."

"...(although not infinite)"

Stability is way more important.

Even FB eventually changed their tune

This is not photoshopped! It's really their new motto

Not as sexy, but it works better if you're a dev

(Swiss) Train Deployments

But here's the thing -- this is not news to anyone

I did not invent this idea

Neither did Yehuda

HOME
CHANNELS
Release
Beta
Canary
PREV. RELEASES

Getting Started? Ember-CLI is the modern, supported way to run an Ember application. Getting Started Guide

EMBER.JS RELEASE CYCLE

Ember releases follow a "train" release model. Every six weeks, the `master` branch of Ember (sometimes referred to as "canary") branches to become the new beta. After six weeks as a beta, this version becomes the new stable release of Ember.

The Ember project adheres to semantic versioning. Releases are named according to a `MAJOR.MINOR.PATCH` scheme, and only `MAJOR` versions releases may change or remove public APIs after deprecation. `MINOR` versions may introduce new features so long as they are backwards compatible, and `PATCH` releases may include bug or security fixes.

CANARY

Canary builds are the bleeding edge of Ember development. Please do not use these builds unless you want to try out and offer feedback on new features and fixes not available in the Release or Beta builds.

Rust 1.0: Scheduling the trains

Dec 12, 2014 • Aaron Turon

As 2014 is drawing to a close, it's time to begin the Rust 1.0 release cycle!

TL;DR: we will transition to a six week release cycle on Jan 9, 2015, and produce Rust 1.0.0 final at least two cycles afterwards:

- Rust 1.0.0-alpha – Friday, Jan 9, 2015
- Rust 1.0.0-beta1 – Week of Feb 16, 2015
- Rust 1.0.0 – One or more six-week cycles later

We talked before about why Rust is reaching 1.0, and also about the 6-week train model (with Nightly, Beta, and Stable channels) that will enable us to deliver stability without stagnation. This post finishes the story by laying out the transition to this new release model and the stability guarantees it provides.

The alpha release

Reaching alpha means three things:

- The language is feature-complete. All gates are removed from features we expect to ship with 1.0.
- The standard library is nearly feature-complete. The majority of APIs that will ship in 1.0 stable will already be marked as `#[stable]`.
- *Warnings* for `#[unstable]` features are turned on by default. (Note that the `#[experimental]` stability level is going away.)

In other words, 1.0.0-alpha gives a pretty accurate picture of what 1.0 will look like, but doesn't yet institute release channels. By turning on warnings for unstable APIs but not excluding them altogether, we can get community feedback about which important APIs still need to be stabilized without those

Release Plan - The Document

di@promptworks

Secure https://wiki.documentfoundation.org/ReleasePlan

Create account Log in

Page Discussion Read View source View history Search The Document Foundatio

Release Plan

LibreOffice Home DLP TDF Community Blogs Pootle TestLink Nextcloud Redmine Ask LibreOffice Donate

Wiki Home | Development | Design | QA | Events | Documentation | Website | Localization | Accessibility | Marketing | Wiki Help

Overview | Reporting Bugs | How to build | Code Overview | Git Commands | Debugging | EasyHacks | **Release Plan** | FAQ | Developers | Extensions | II | →Open Issues

Home | 5.4 | 5.3 | 5.2 | 5.1 | 5.0 | 4.4 | 4.3 | 4.2 | 4.1 | 4.0 | 3.6 | 3.5 | 3.4 | 3.3

EN | AN | AR | AST | BE | BG | BN | BRX | CA | CA-VAL | CS | DA | DE | EL | EO | ES | FA | FI | FR | GD | GL | GUG | HE | HI | HU | ID | IS | IT | JA | JV | KO | LO-LA | LT | MN | MR | NL | NO | OC | OM | PA | PL | PT | PT-BR | RO | RU | SAH | SK | SL | SQ | SV | TA | TE | ไทย (TH) | TR | UK | VI | 正體 (ZH-TW) | 简体 (ZH-CN)

Main page
Get Involved
Recent changes
Random page

Tools

What links here
Related changes
Special pages
Printable version
Permanent link
Page information

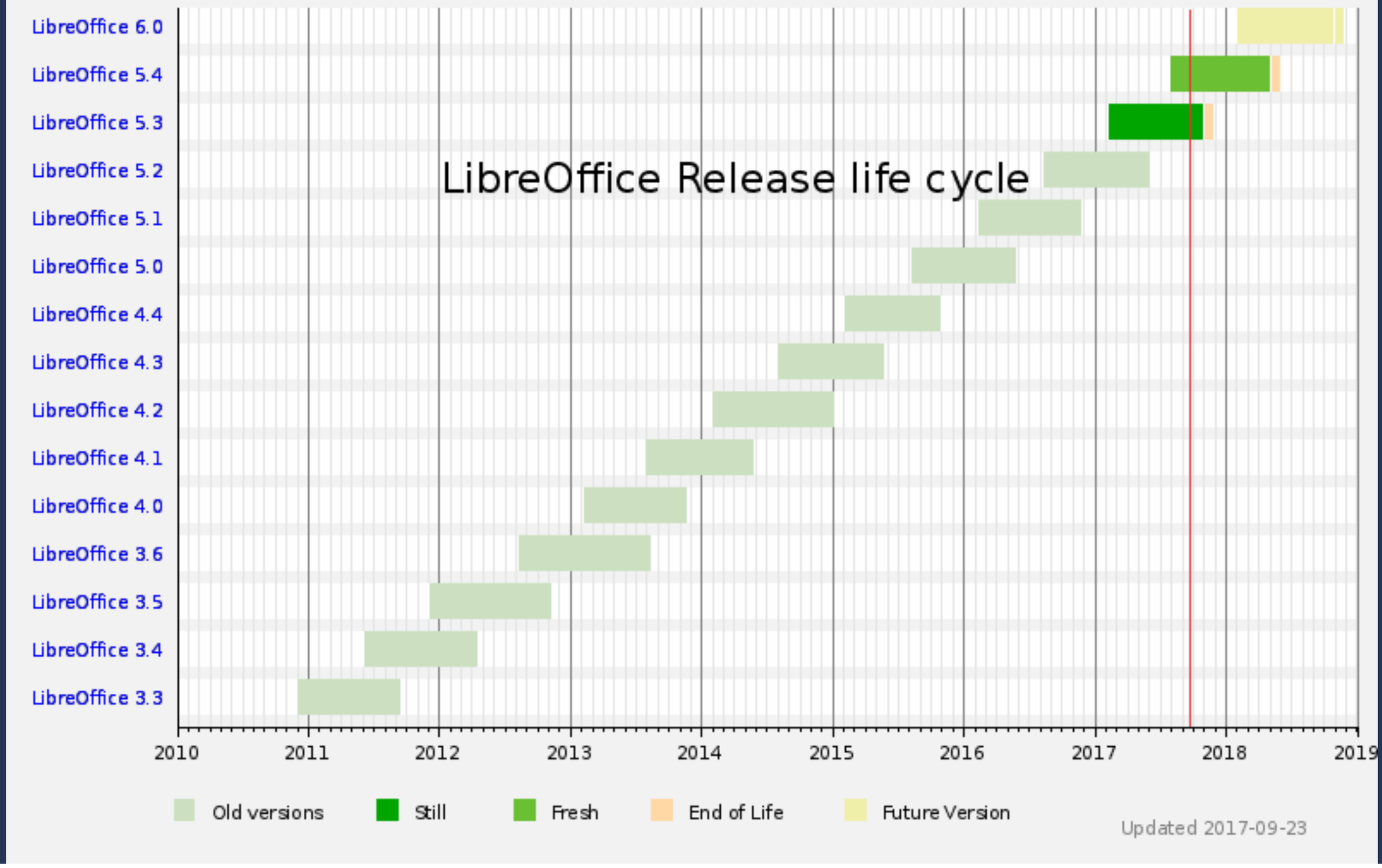
Introduction

Time-based release trains have been shown to produce the best quality Free software. A time based release is one that does not wait for either features, or bug fixes - but is based (as purely as possible) on time. This enforces discipline in introducing fixes, gives predictability, and allows more regular releasing. It is also the case that we will necessarily release earlier, and then rapidly, incrementally bug fix releases based on the previous stable version. Thus if you have a need for the very highest quality version, it can make sense to defer a move until the first or perhaps second minor point release.

Synchronizing our time-based release schedule with the wider Free Software ecosystem also has huge advantages, by getting our new features, out to users as quickly as possible – with a minimum of distribution cycle lag. In consequence, we aim at six monthly releases, and over time nudge them to align well with the March/September norms.

As a result, users get new major version every six months with a wide range of features

Contents [hide]


LibreOffice Release life cycle
LibreOffice 6.0
LibreOffice 5.4
LibreOffice 5.3
LibreOffice 5.2
LibreOffice 5.1
LibreOffice 5.0
LibreOffice 4.4
LibreOffice 4.3
LibreOffice 4.2
LibreOffice 4.1
LibreOffice 4.0
LibreOffice 3.6
LibreOffice 3.5
LibreOffice 3.4
LibreOffice 3.3
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
Old versions
Still
Fresh
End of Life
Future Version
Updated 2017-09-23

Search
Navigation
Main page
Recent changes
Random page
Help
Toolbox
Page information
Permanent link
Printable version
Special pages
Related changes
What links here

Simultaneous Release

Each year the **Eclipse Foundation** and its community of projects and contibutors produce a release on a coordinated schedule. This release is often referred to as the *simultaneous release*, *coordinated release*, *release train*, or *annual release* of Eclipse. This page provides an index of existing simultaneous releases from the current and previous years. Each release typically occurs in June, with follow-up update releases in September (*.1), December (*.2), and March (*.3).

Release name	Platform version	Main release	.1	.2	.3	Links
Photon (Next release)	4.8	June 27, 2018	September, 2018	December, 2018	March, 2019	**Wiki / Plan**
Oxygen (Current release)	4.7	June 28, 2017	September 27, 2017	December 20, 2017	March 21, 2018	**Web Site** **Wiki / Plan** **Main Download Page** **Package Download Page** **p2 Repository**
Neon	4.6	June 22, 2016	September 28, 2016	December 21, 2016	March 23, 2017	**Web Site** **Wiki / Plan** **Package Download Page** **p2 Repository**

Before Neon, each release train had two service releases in September (SR1) and February (SR2):

GNOME's Time-Based Release Schedule

Although we agree on rough aims for each major release, and attempt to achieve those aims, GNOME releases are time-based rather than feature-based. A roughly 6-month release cycle allows us to coordinate development of the features that have actually been implemented, allowing us to maintain the quality of the overall release without delaying everything because of one or two features. If a feature is not ready in time then the developers must not wait long to put it in the next major release. We have found that this encourages both high quality and rapid development compared to feature-based release schedules.

Havoc Pennington's original plan, suggesting time-based releases, might be interesting.

For instance, the current schedule is on the release start page. Once it has been created, the schedule should not be expected to change, because there is no need to wait for features to be completed.

A whole GNOME release schedule should last approximately 6 months.

The schedule should contain the following:

- Regular test release dates, approximately every 2 weeks.
  Requests for tarball releases will be sent to desktop-devel-list a few days before the release dates. The request dates should also be on the schedule, to avoid surprise. The tarballs will be made available on GNOME's ftp server, and there will be a folder of symlinks listing just the specific module tarballs for the particular GNOME release.
- Dates for the various freezes.
- A date for the final release.

The releng module contains scripts for creating the schedule and related announcements (e.g. tarballs due email). See the scripts in the releng module, tools/schedule directory.

TimeBasedReleases

Introduction

Ubuntu releases on a time based cycle, rather than a feature driven one. Sometimes this can generate confusion, especially when people ask for a new feature to be added. Here are some answers to some common questions regarding how Ubuntu's releases and should answer why your feature request was just put off or turned down.

Contents

What are time-based releases?

There are a variety of strategies for deciding when to release a new version of a piece of software, and different projects may employ different techniques. Ubuntu plans in advance to release on a certain date, and the preceding development effort is aimed at producing a high-quality release on this prearranged date.
See Ubuntu releases as well as the Releases page for a list of Ubuntu releases and their schedules.
The Ubuntu release process was heavily influenced by the release process used by the GNOME project. As an analogy, consider a play being produced in a theatre. While things may go wrong during production, even (perhaps especially) on the night, tickets have been sold and there is a very high cost to backing out: "the show must go on".

Debian decides to adopt time-based release freezes

July 29th, 2009

The Debian project has decided to adopt a new policy of time-based development freezes for future releases, on a two-year cycle. Freezes will from now on happen in the December of every odd year, which means that releases will from now on happen sometime in the first half of every even year. To that effect the next freeze will happen in December 2009, with a release expected in spring 2010. The project chose December as a suitable freeze date since spring releases proved successful for the releases of Debian GNU/Linux 4.0 (codenamed *"Etch"*) and Debian GNU/Linux 5.0 (*"Lenny"*).

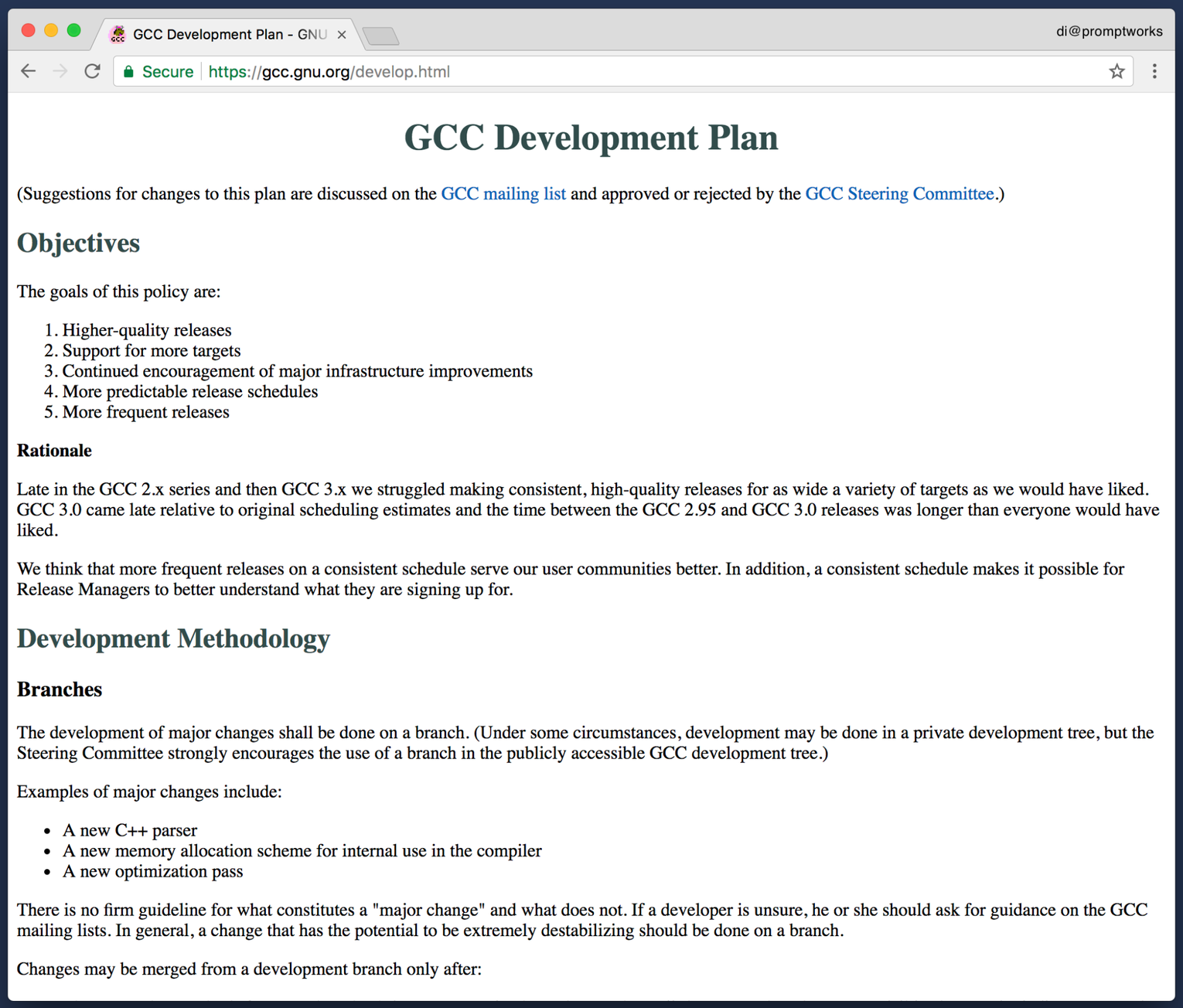
Time-based freezes will allow the Debian Project to blend the predictability of time based releases with its well established policy of feature based releases. The new freeze policy will provide better predictability of releases for users of the Debian distribution, and also allow Debian developers to do better long-term planning. A two-year release cycle will give more time for disruptive changes, reducing inconveniences caused for users. Having predictable freezes should also reduce overall freeze time.

Since Debian's last release happened on Feb. 14th 2009, there will only be approximately a one year period until its next release, Debian GNU/Linux 6.0 (codenamed *"Squeeze"*). This will be a one-time exception to the two-year policy in order to get into the new time schedule. To accommodate the needs of larger organisations and other users with a long upgrade process, the Debian project commits to provide the possibility to skip the upcoming release and do a skip-upgrade straight from Debian GNU/Linux 5.0 (*"Lenny"*) to Debian GNU/Linux 7.0 (not yet codenamed).

Although the next freeze is only a short time away, the Debian project hopes to achieve several prominent goals with it. The most important are multi-arch support, which will improve the installation of 32 bit packages on 64 bit machines, and an optimised boot process for better boot performance and reliability.

The new freeze policy was proposed and agreed during the Debian Project's yearly conference, DebConf, which is currently taking place in Caceres, Spain. The idea was well received among the attending project members.

About Debian



Earliest I could find, started in 2001

Stability without Stagnation

What is SwS reallly about? It's a philosophy, but

Developing a process which keeps you from getting stuck w/o innovating

But also giving you a stable foundation to build higher and faster on

Swiss Train Deployments

1. Don't make your users run for a release

2. Let them decide how fast to get from A to B

They can take a faster track to a feature, but they might go off the rails

Questions?

Thanks!