Engineering Practices

Configuration Management

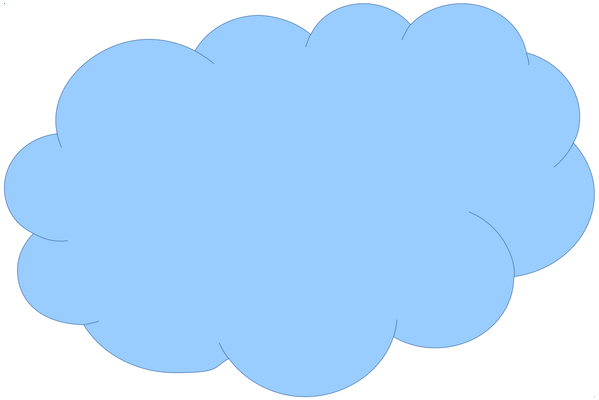
Development / Production Parity

Environments

- Example:

Developers work under OSX

Local staging environment with Red Hat



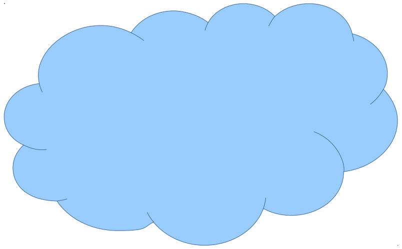
Production on AWS cloud

Environments

What changes from one environment to the other?

Dev

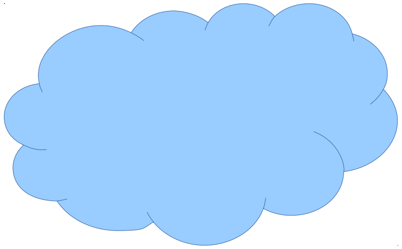
Staging



Production

Environments

What changes from one environment to the other?



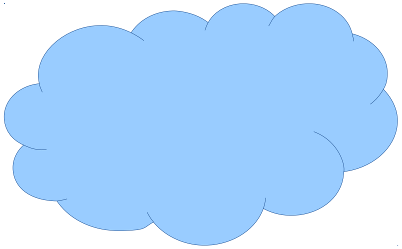
- Does all the same software run in these environments?
- Is it the same hardware?
- Should the software be configured the same way?

Environments

How does the code move around?

Dev

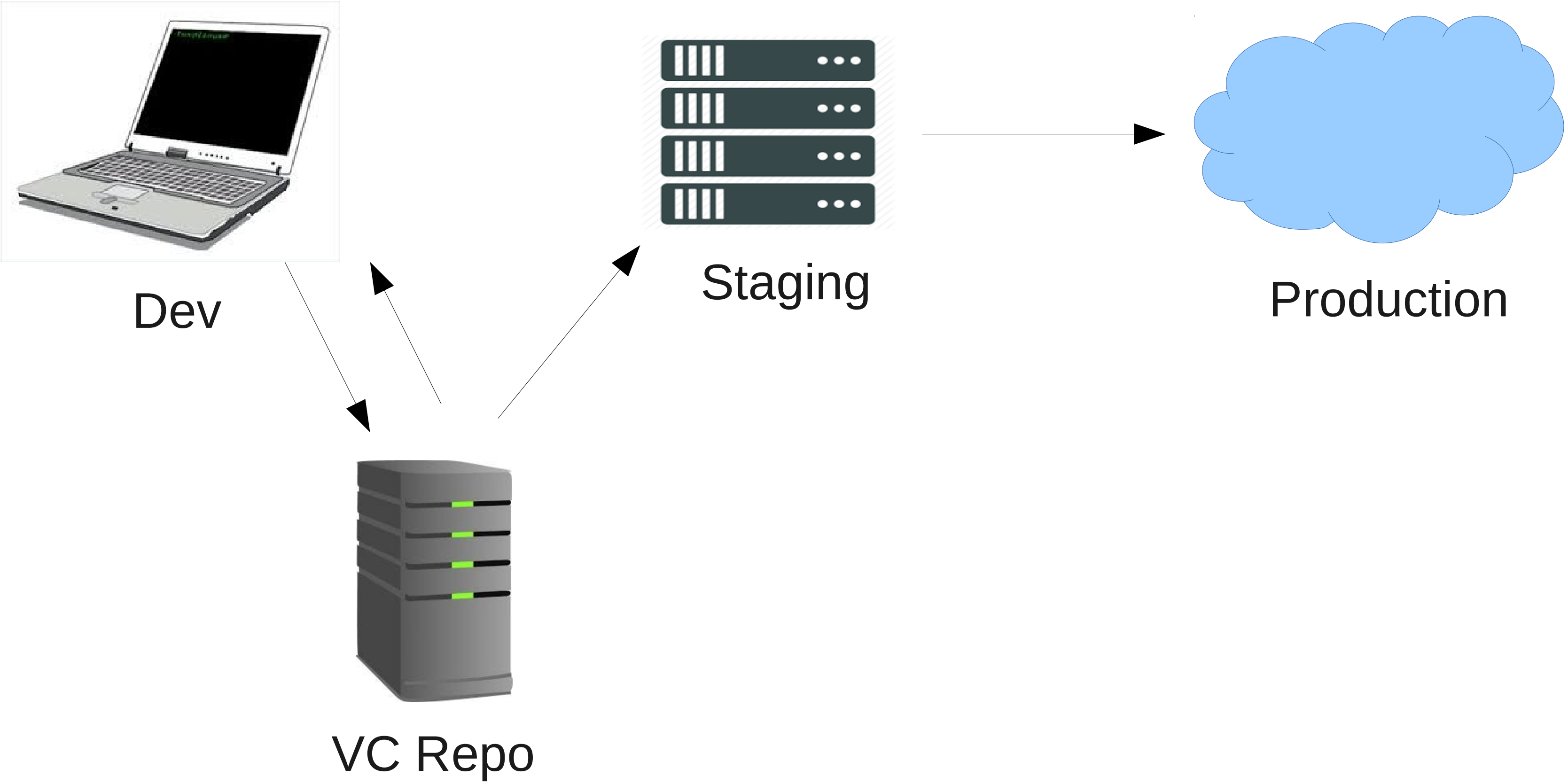
Staging



Production

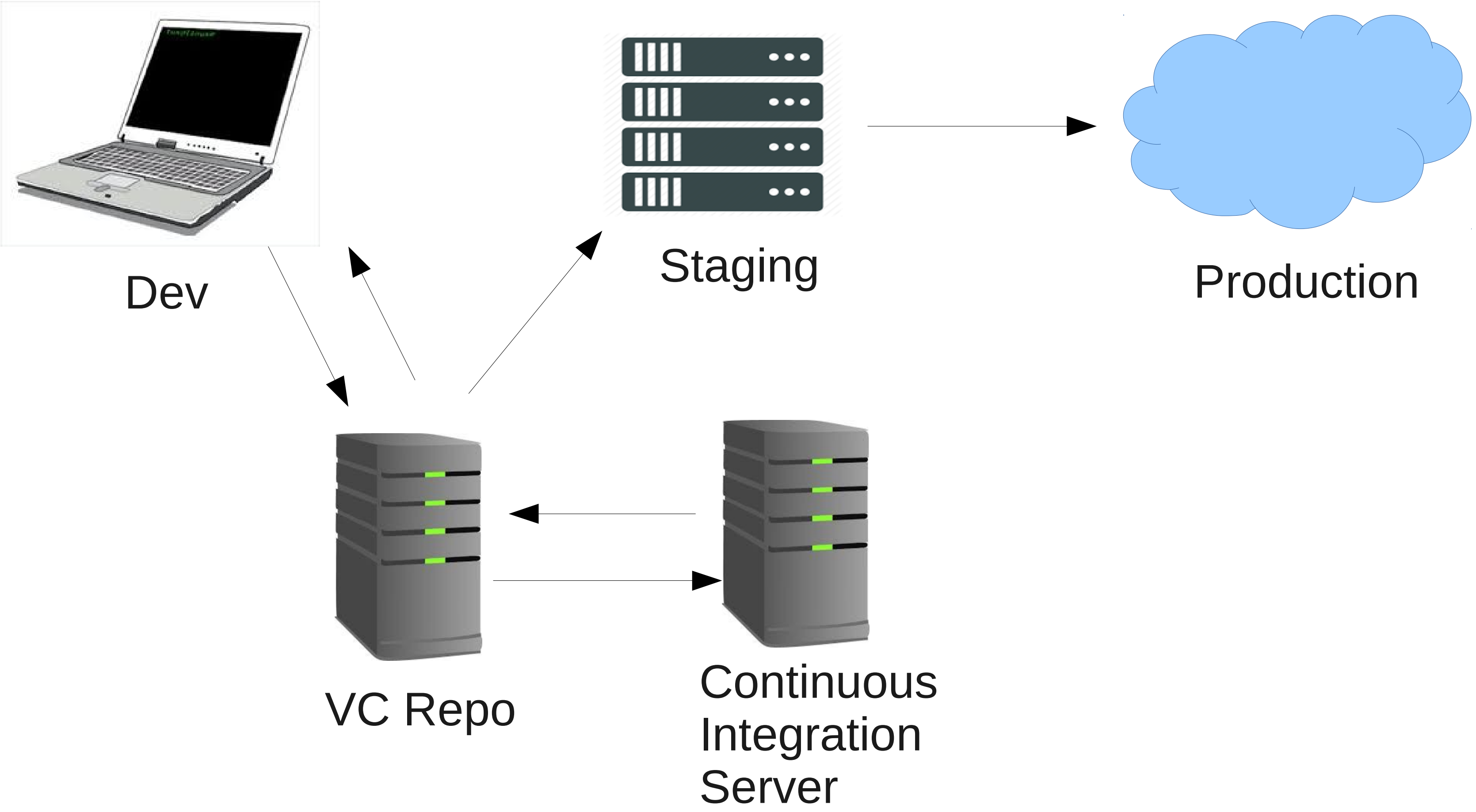
Environments

How does the code move around?



Environments

How does the code move around?



Dev

Staging

Production

VC Repo

Continuous Integration Server

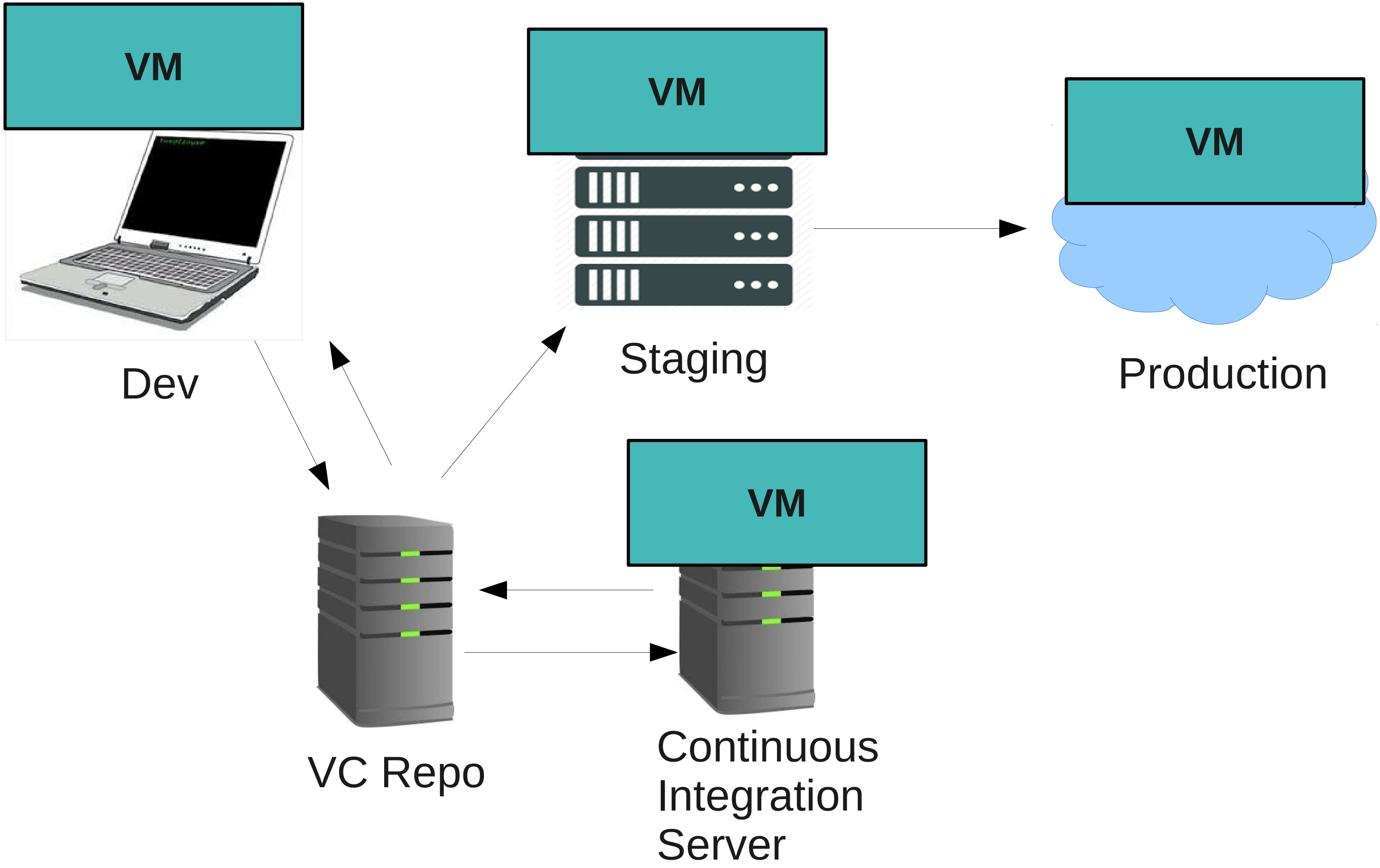
Problems

- Different environments => untested environments
- Configuration & deployment: many moving parts
  - Do it manually? => Long and error prone
  - Undo button?

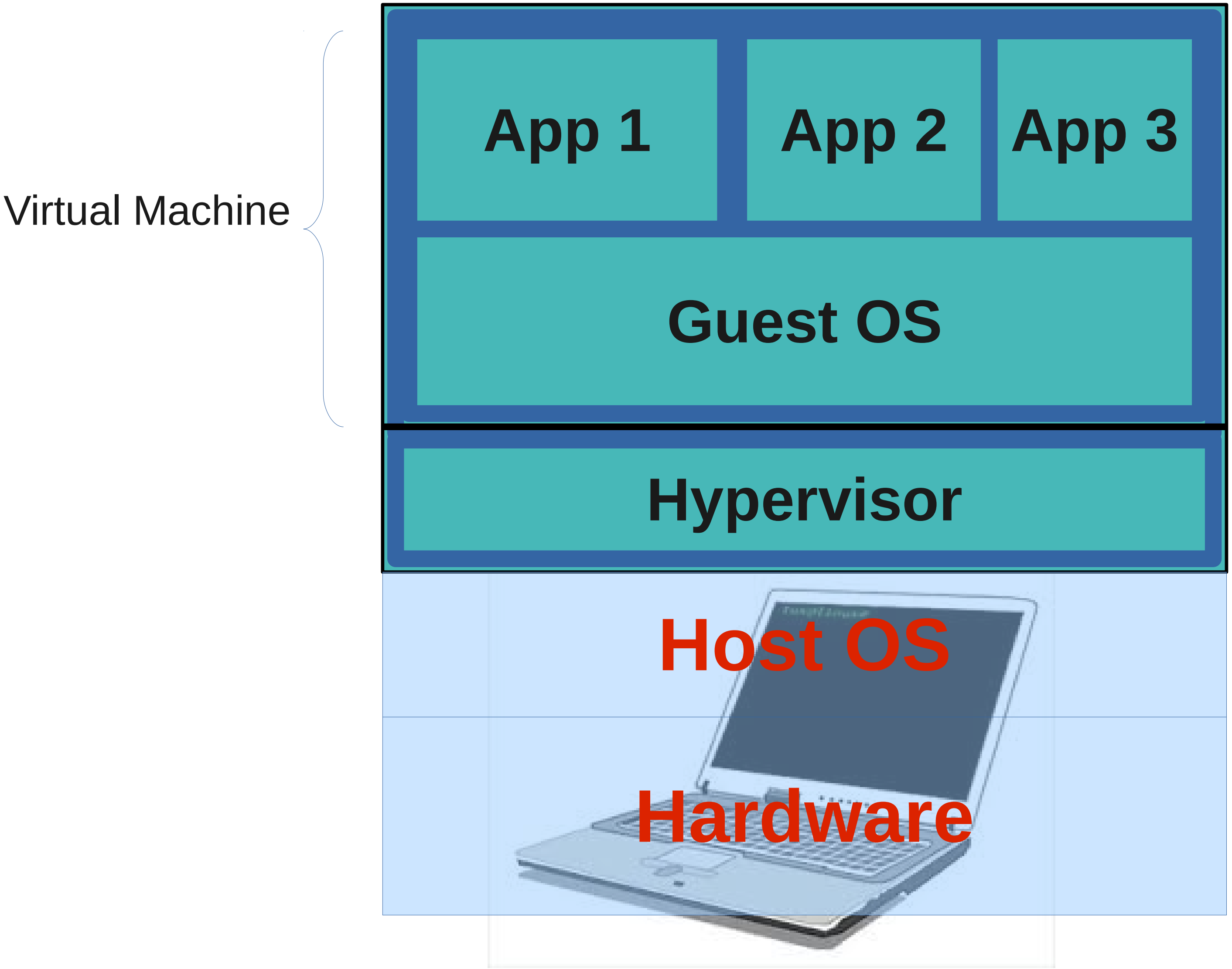
Solutions

- Virtualization
- Configuration Management
- Deploy Automation

Virtualization



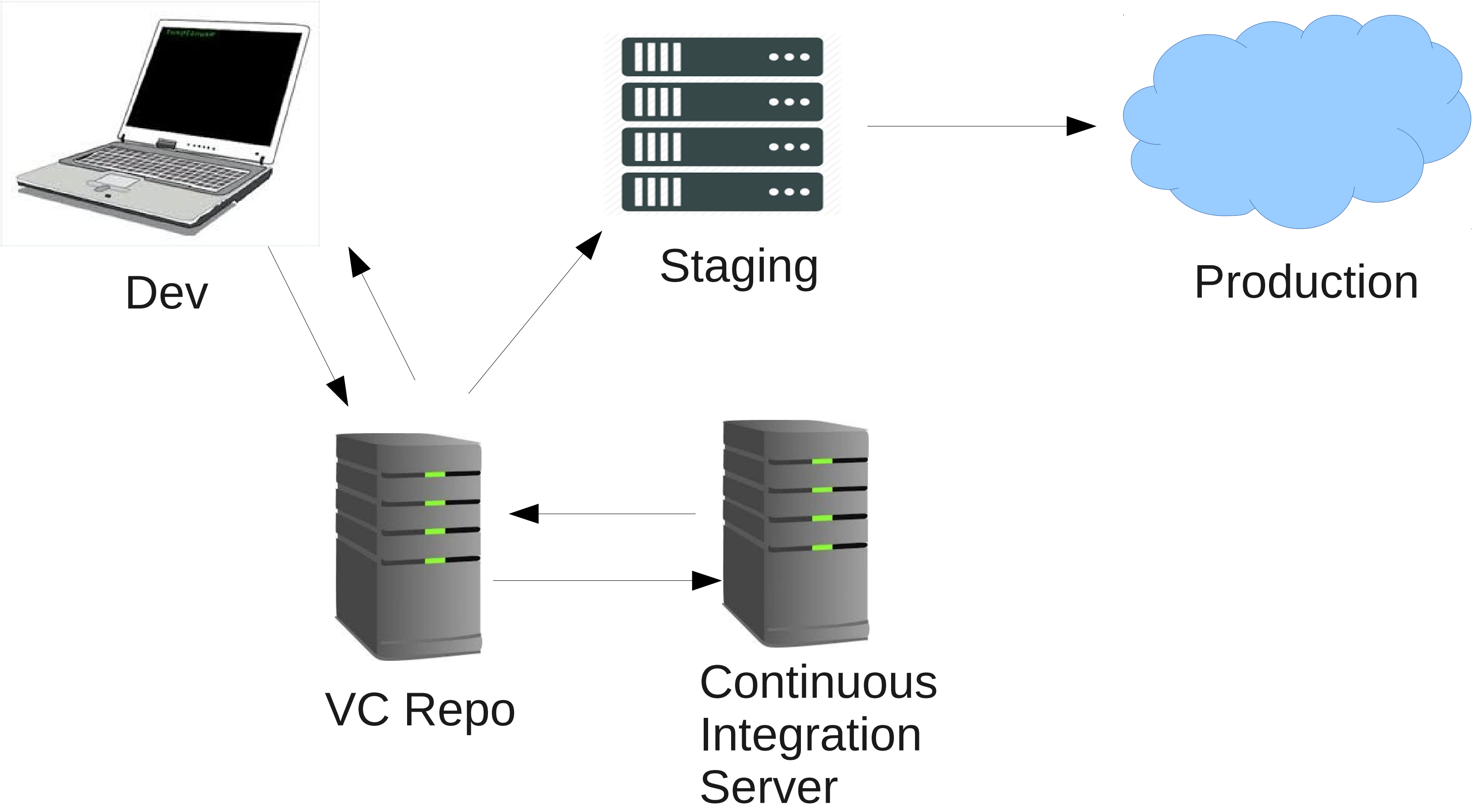
Virtualization



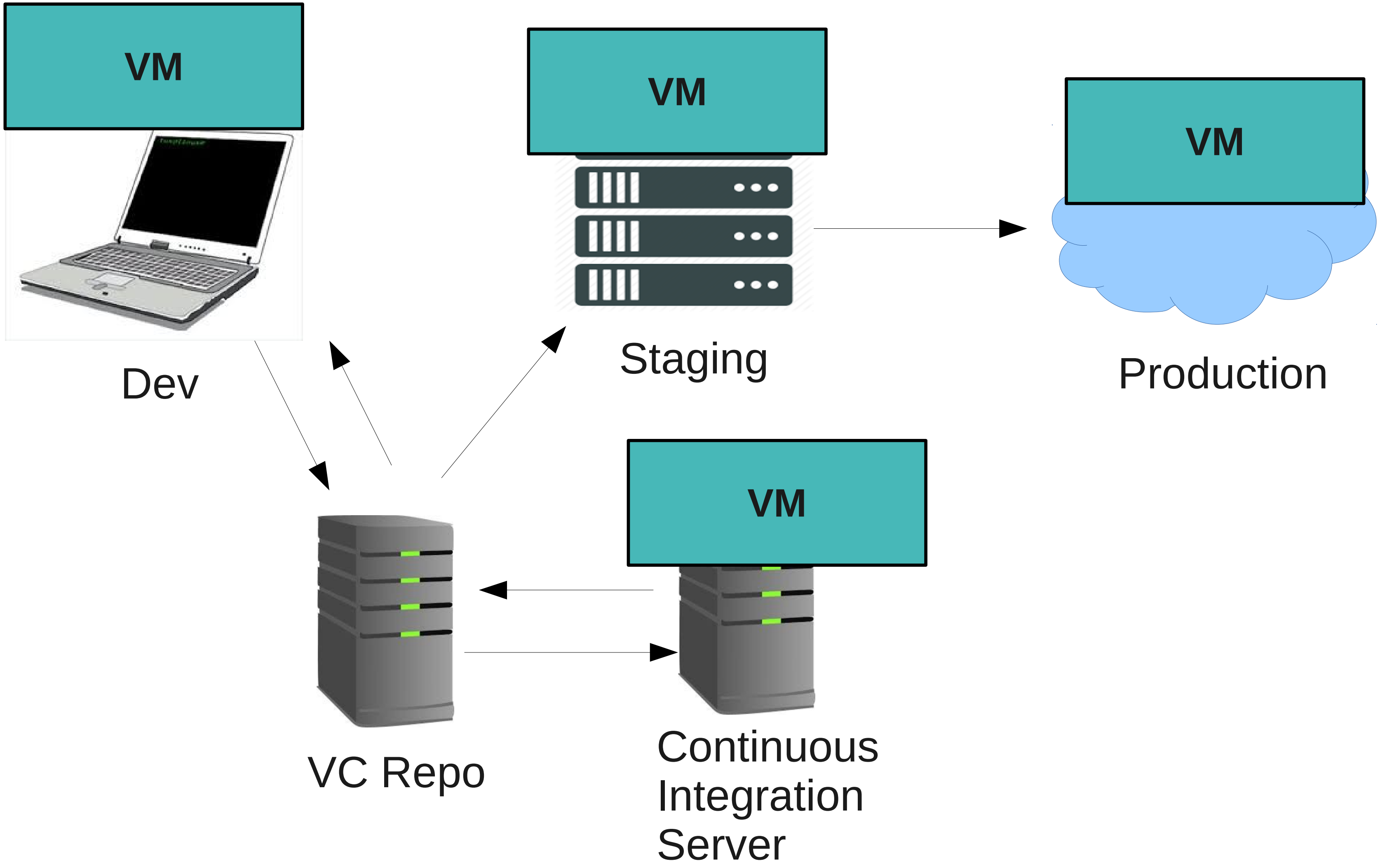
Virtualization

- Software Emulation of the Hardware
- Problems:
  - Overhead (2 OS + Hypervisor)
  - How do you provision the VMs?

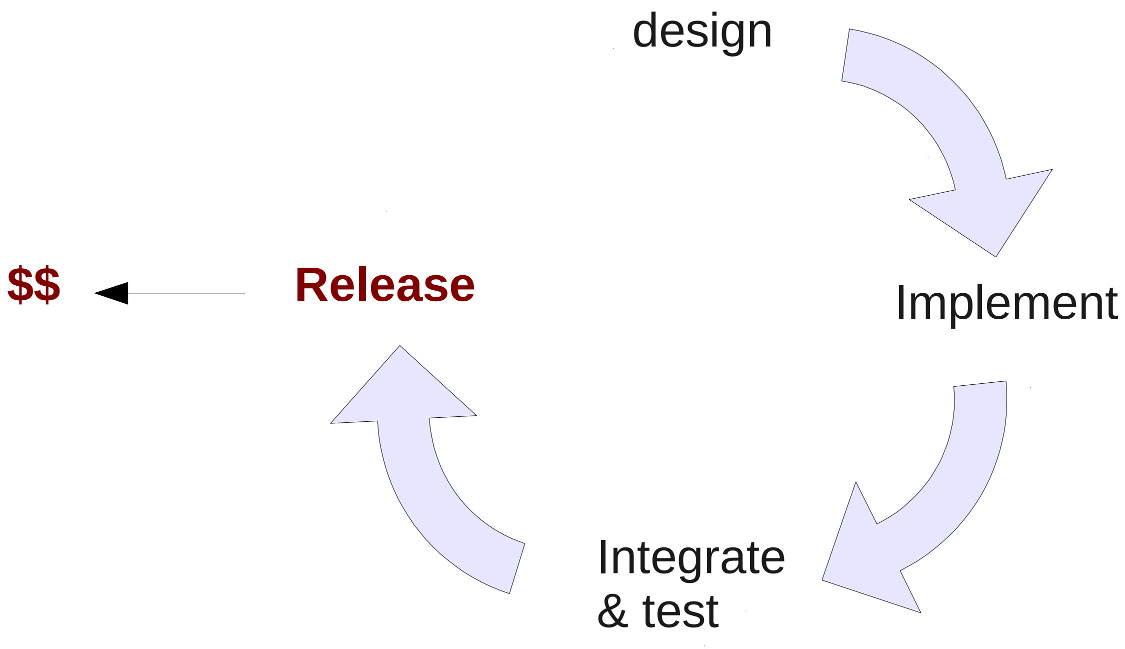
Environments



Virtualization



Software Lifecycle



design

Implement

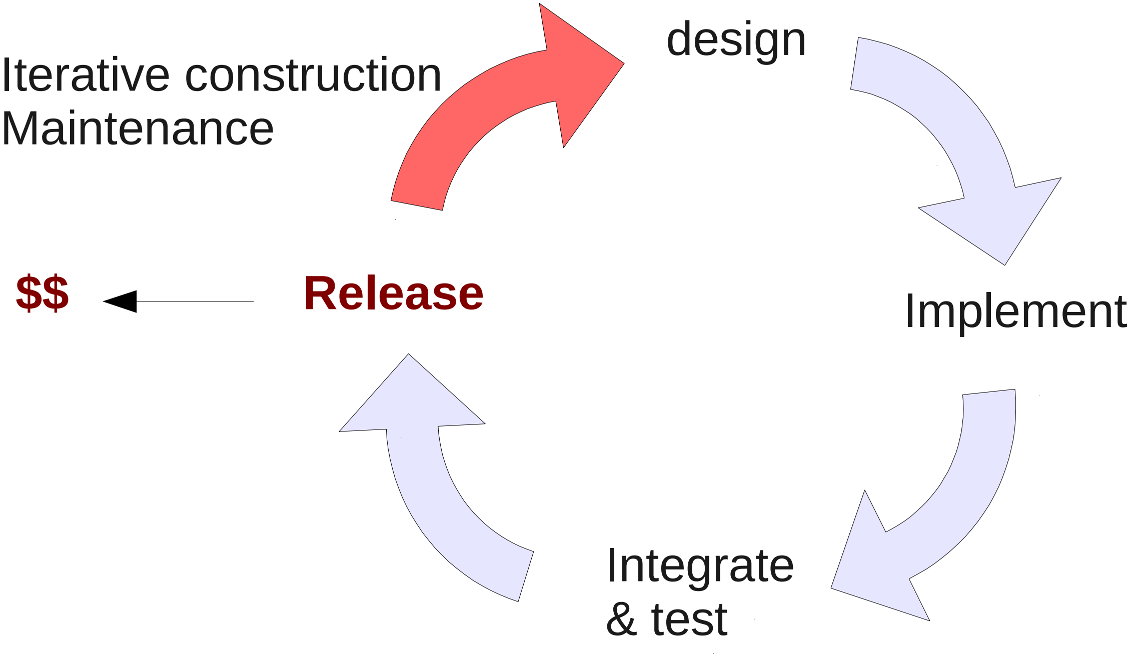
Integrate & test

Release

$$

Software lifecycle

- Iterative construction
- Maintenance



design

Implement

Integrate
& test

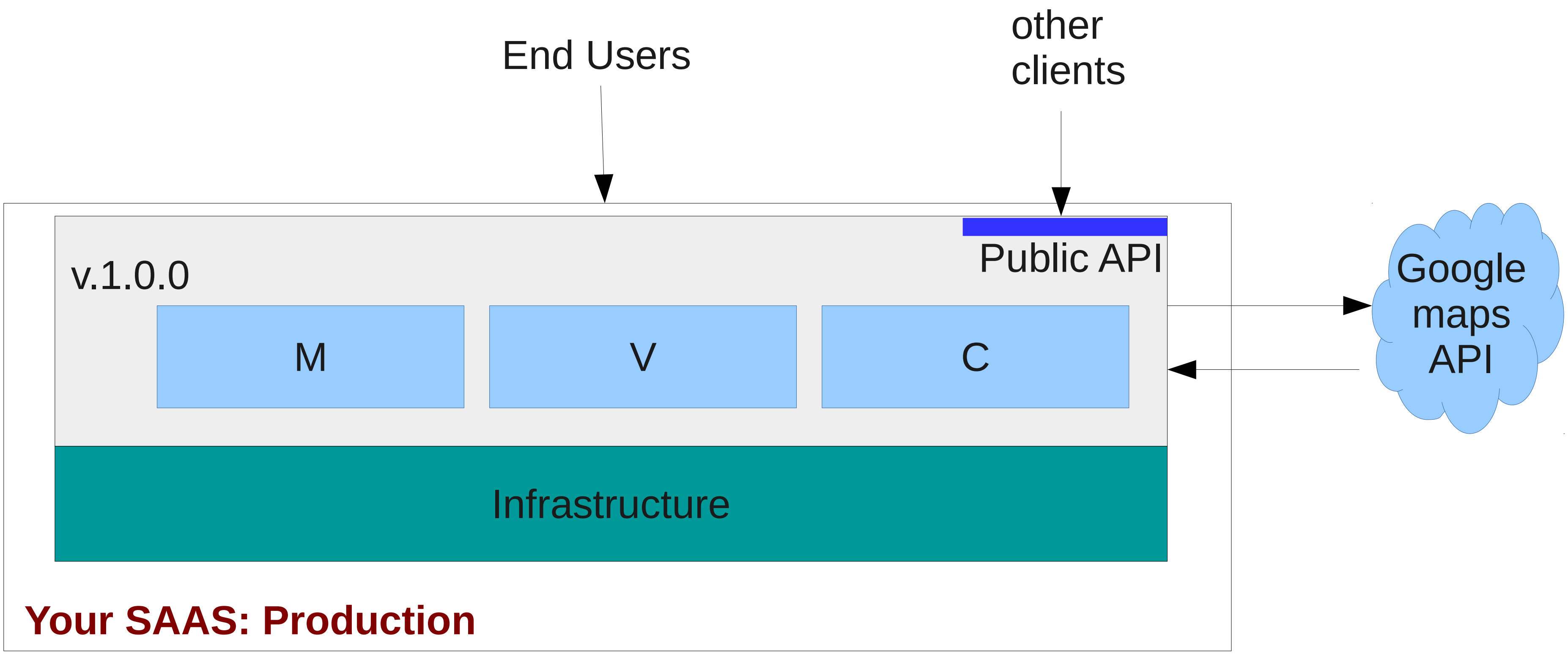
Release

$$

Software maintenance

- Maintenance:
  - Evolutions (new features)
  - Bug fixes & improvements
  - Keep up with environment
- Goal:
  - Release bug-free software
  - Release software that the client wants
  - Release it more frequently (“Continuous delivery”)

Software maintenance



End Users

other clients

Public API

v.1.0.0

M

V

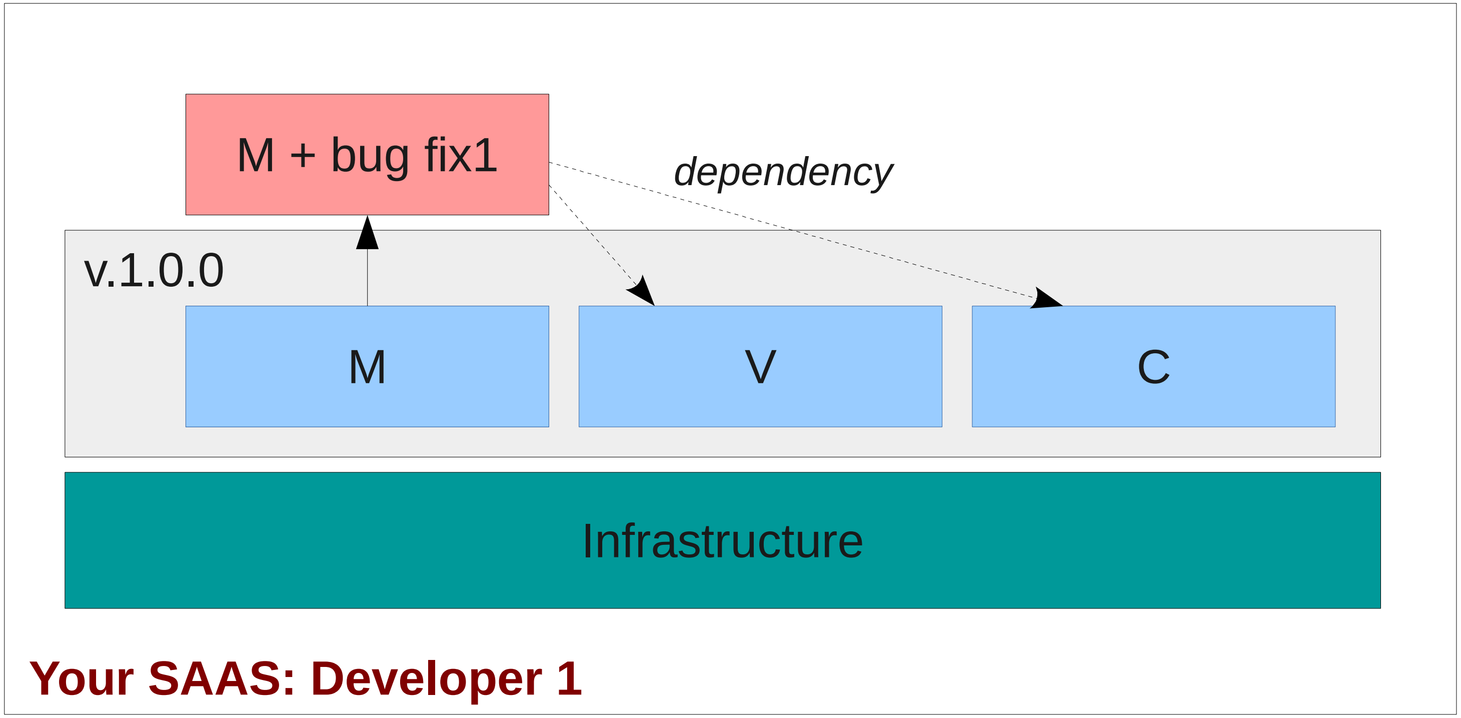
C

Google maps API

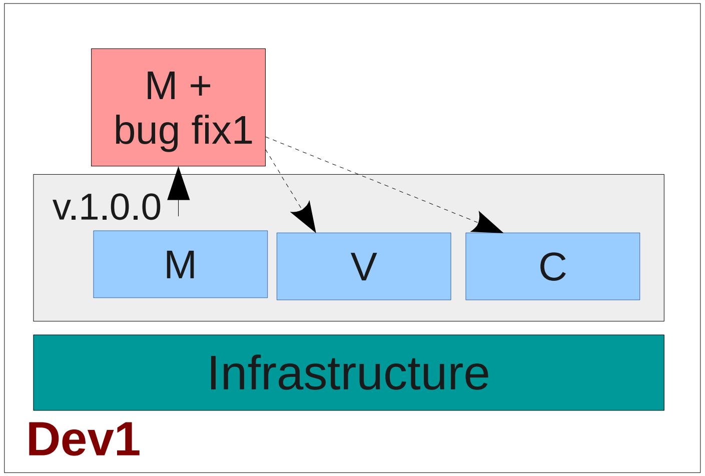
Infrastructure

Your SAAS: Production

Software maintenance



Software maintenance



M + bug fix1

v.1.0.0

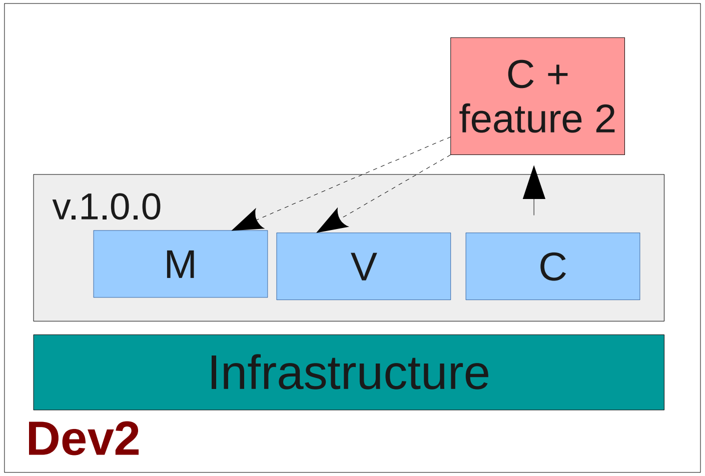
M

V

C

Infrastructure

Dev1



C + feature 2

v.1.0.0

M

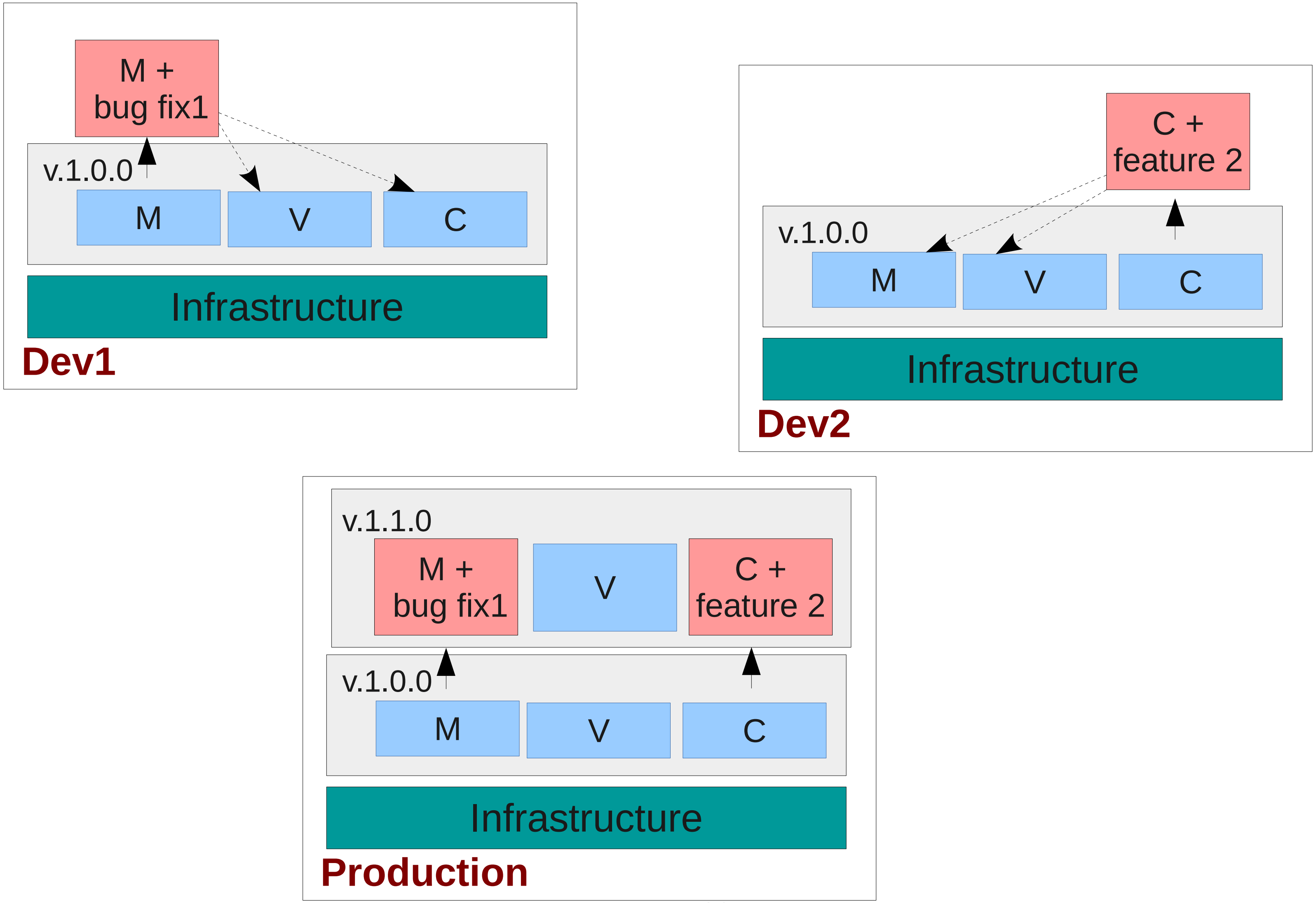
V

C

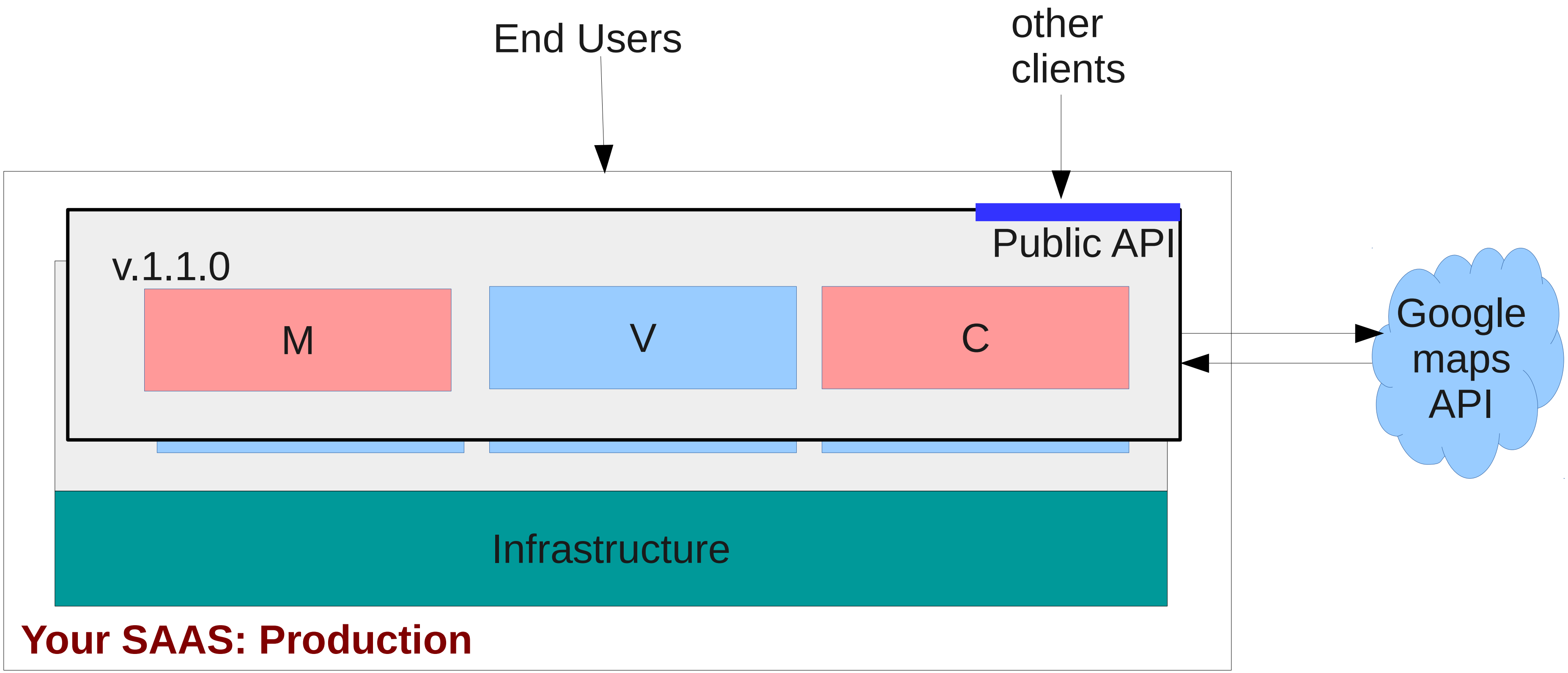
Infrastructure

Dev2

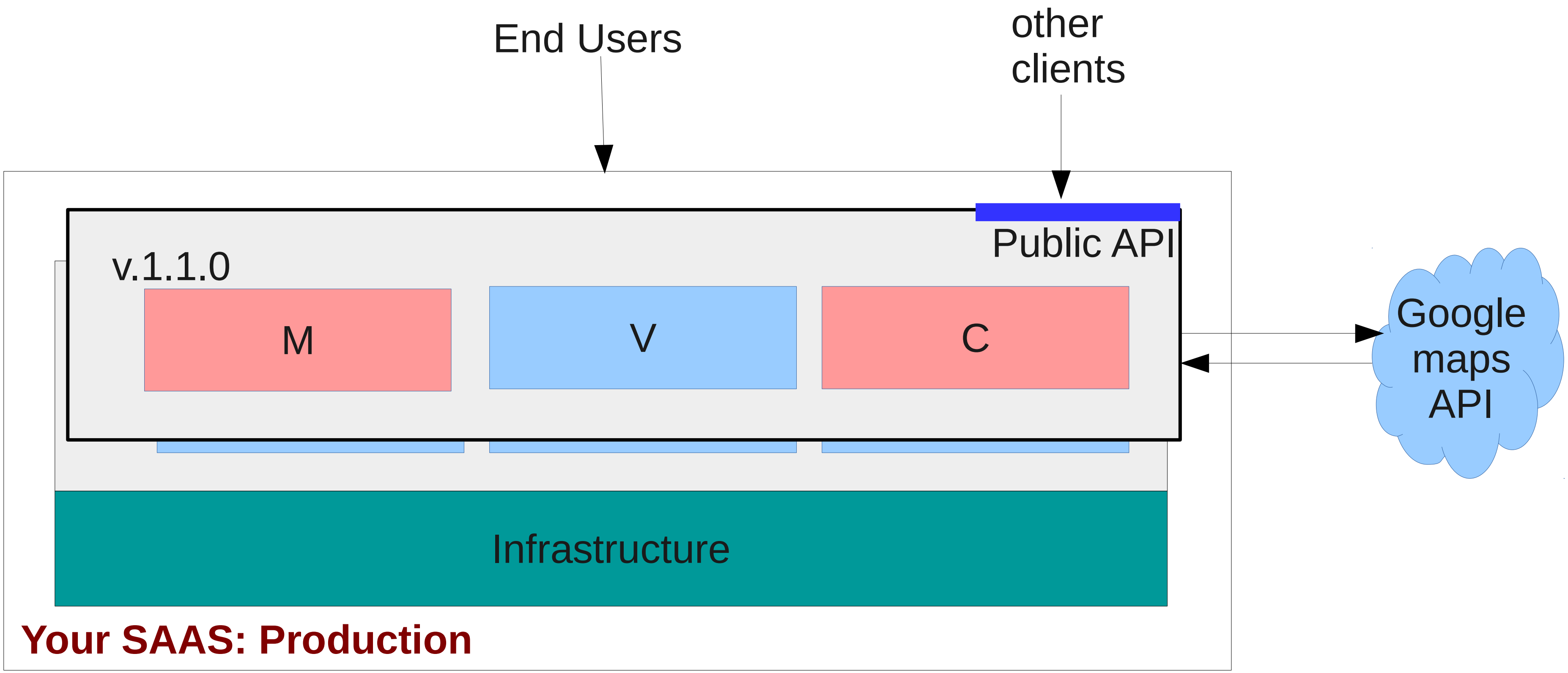
Software maintenance



Software maintenance

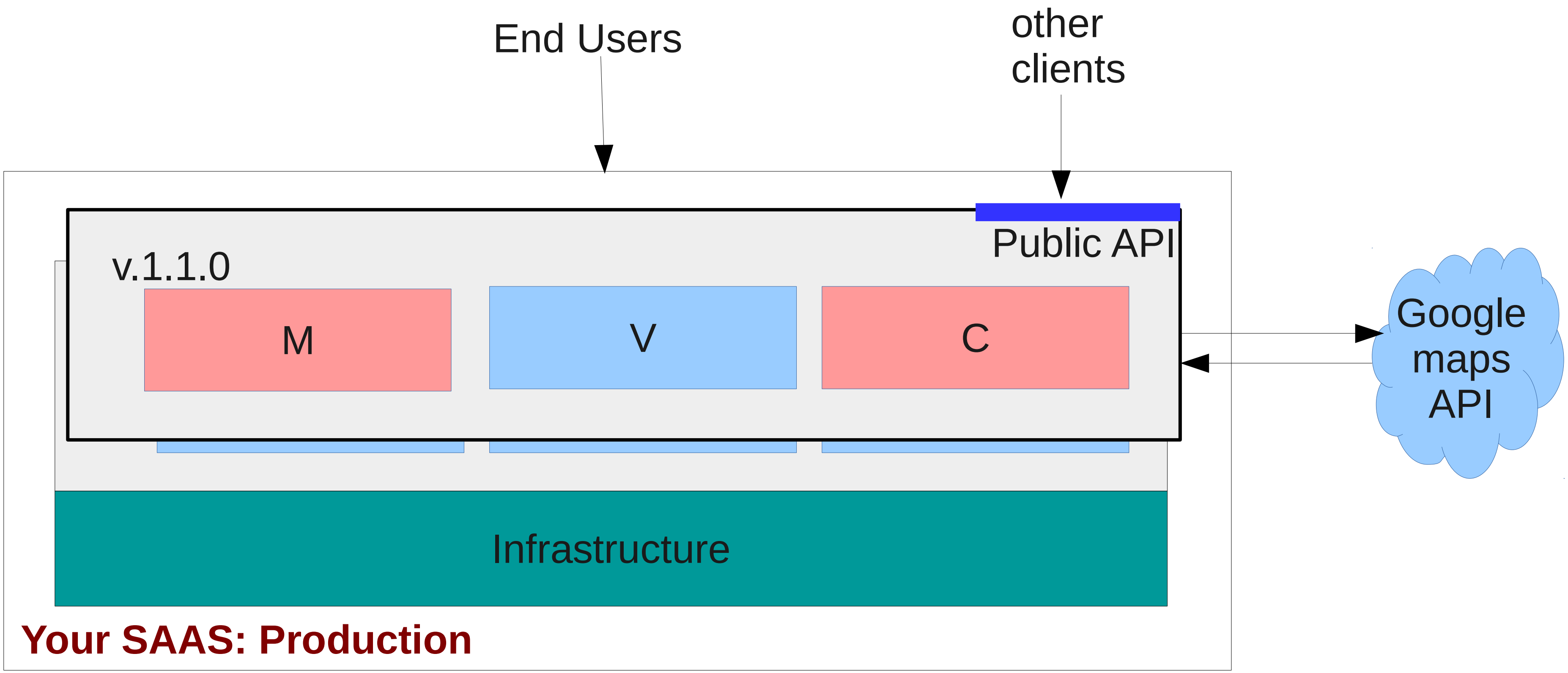


Issues



- Do **M** and **C** work together?
- Do **M** and **C** work with the production infrastructure?
  - Do **M** and **C** have new dependencies?
  - Has the infrastructure changed?

Issues (2)



- Does our product still work with Google maps API?
- Has our functionality changed?
  - Noticeable differences to end-users ?
  - Public API changes ?
    => What should our version number be?

Solutions

- Configuration Management
- Continuous integration
- Dependency Management

- Extend these practices to infrastructure management
  - Virtualization
  - Automated provisioning / deployment

Configuration Management

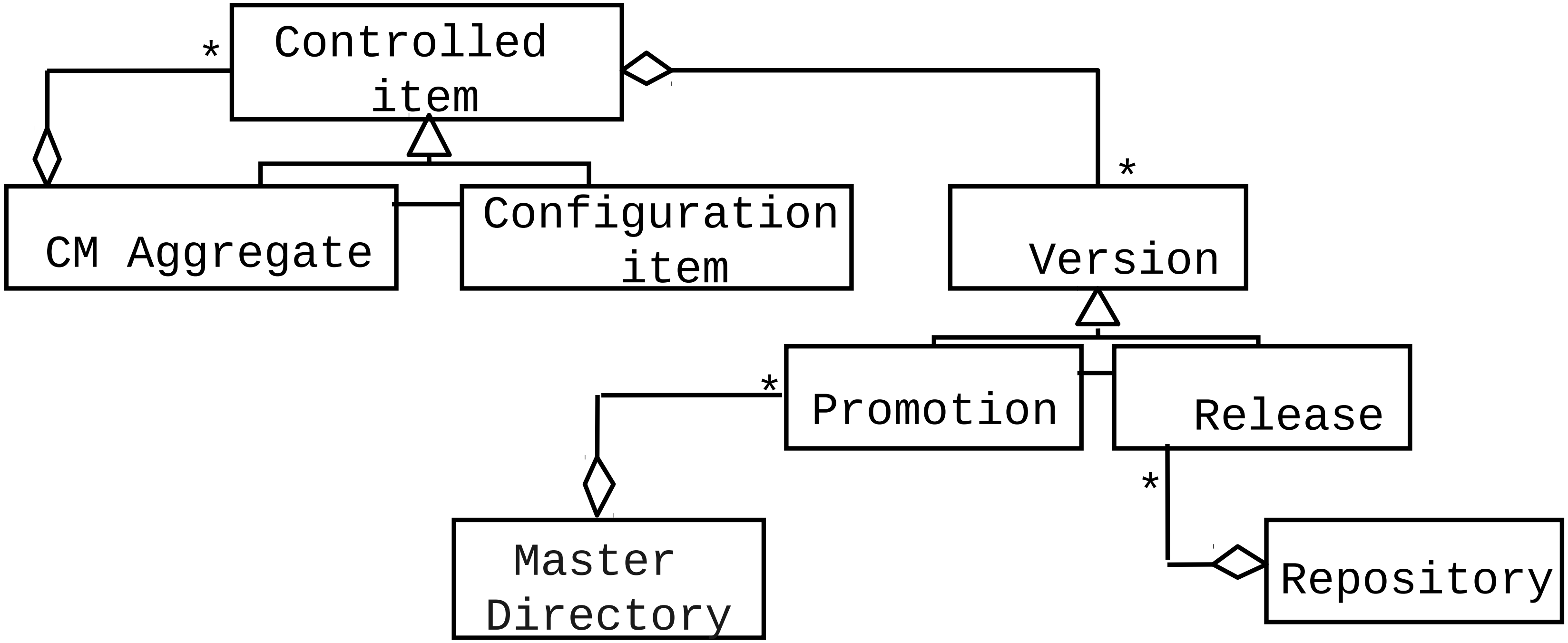
Def*:* the discipline of managing and controlling change in the evolution of software systems (IEEE, 1987)

- Identify *configuration items* and their *versions*
  - Version control (git, etc.)
- Manage / control change to these items
  - e.g. Require code reviews
- Manage releases and variants
  - What goes into a release
  - Variants example: A/B testing

Configuration Management

- Agile perspective:
  - Loosen control (embrace change, react to change)
  - Increase speed: Continuous integration, continuous delivery
- “DevOps” perspective: “Infrastructure as code”
  - Infrastructure is a configuration item
  - Infrastructure is programmable (virtualization and automatic provisioning)

Configuration Management Concepts



(Bruegge & Dutoit book)

Version Control

- Many VC systems:

  RCS, CVS, SVN, ClearCase, Mercurial, Bazaar, git...

  centralized decentralized

- Purpose: support collaborative work on files
  - No overwriting other people's changes
  - Keep track of all past versions
  - Keep track of multiple versions in parallel (branches)

Version Control

- Main tasks:

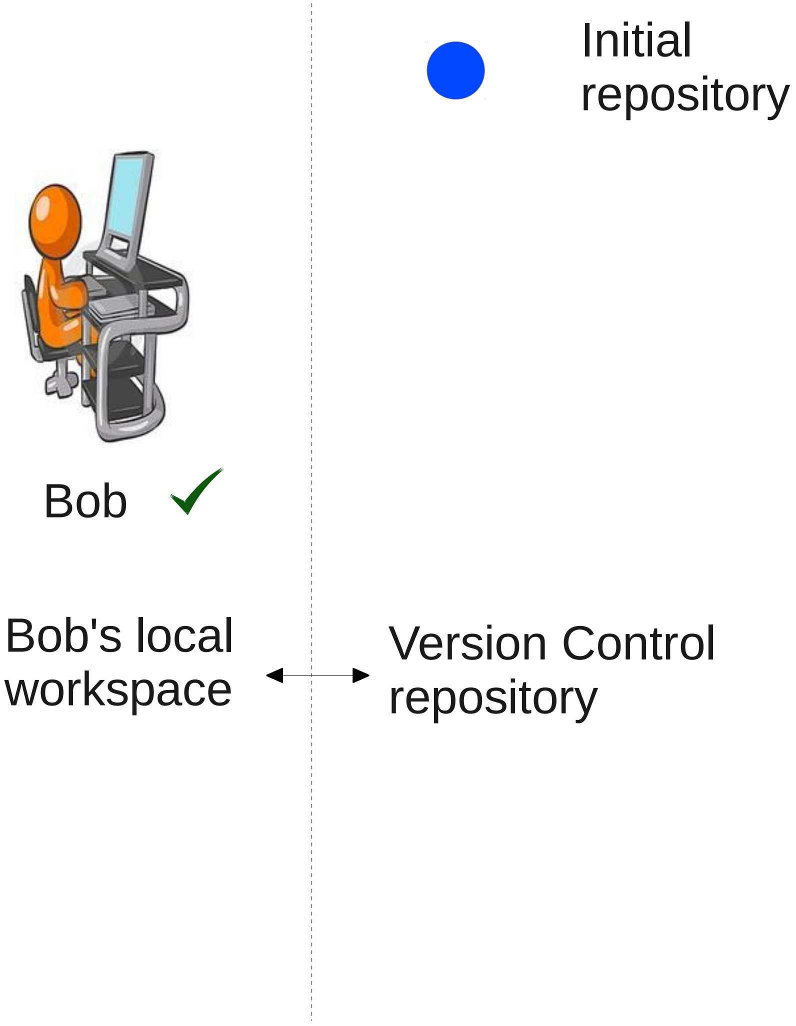
Version Control

- Main tasks:
  - Create repository

Initial repository

Version Control

- Main tasks:
  - Create repository
  - Define access control



Initial repository

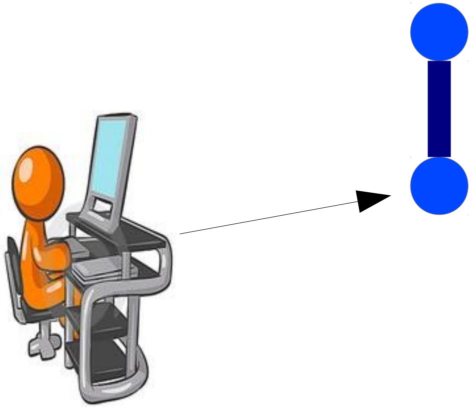
Bob ✓

Bob's local workspace

Version Control repository

Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)



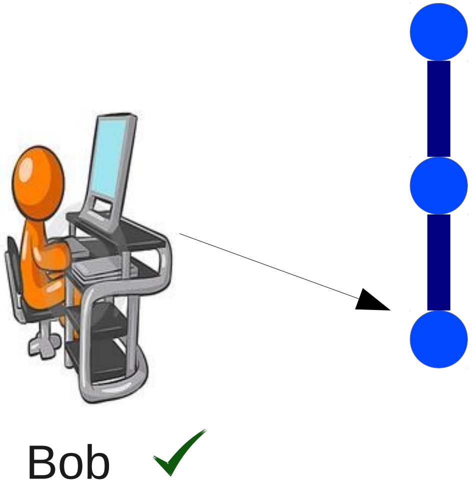
Initial repository

commit #1 by Bob
2014-06-02 09:15

Bob ✓

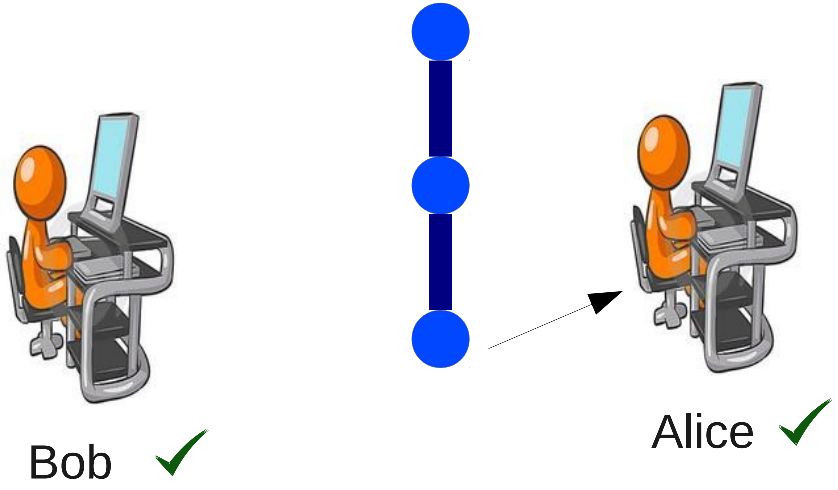
Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)



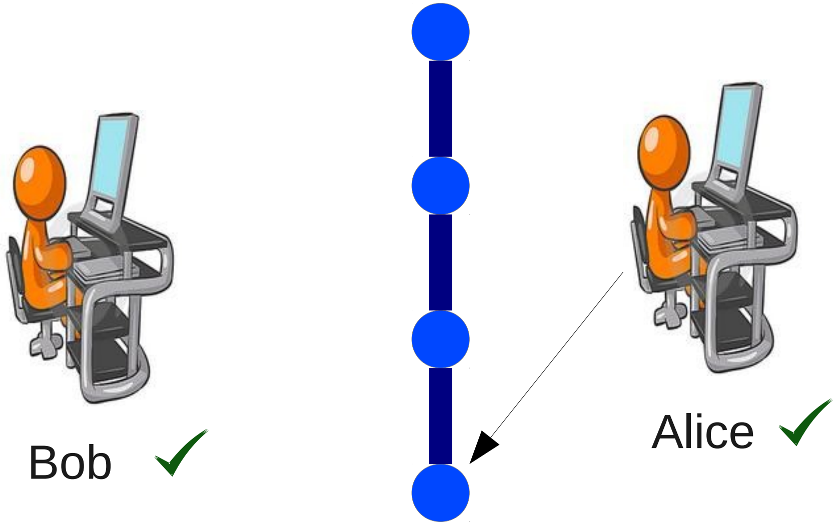
Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)



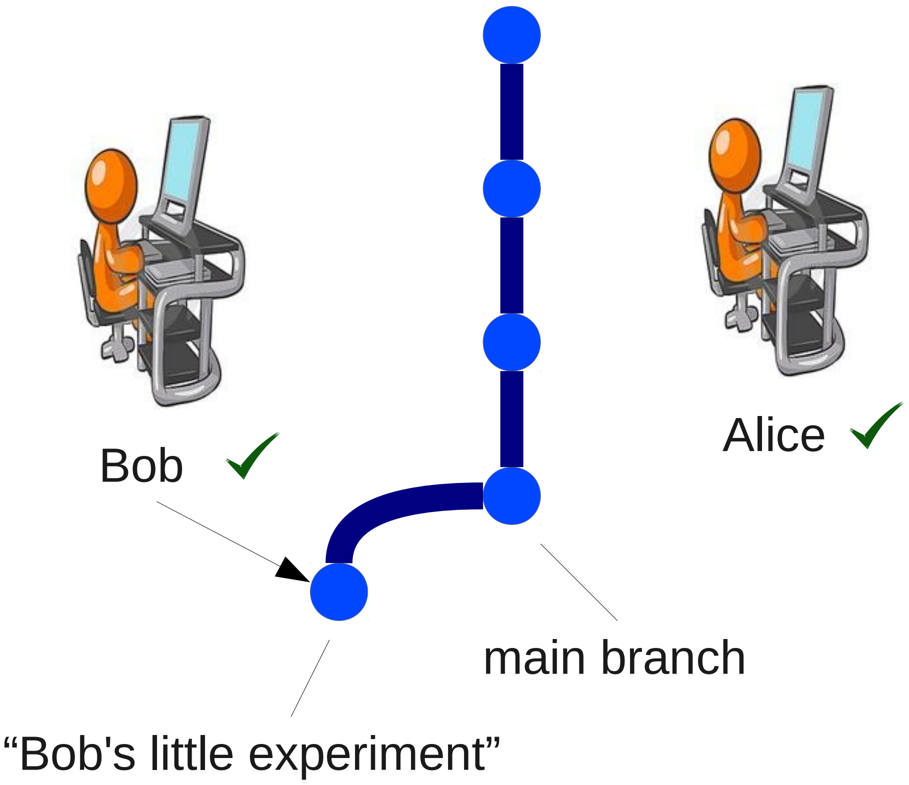
Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - (Alice commits)



Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch



Bob ✓

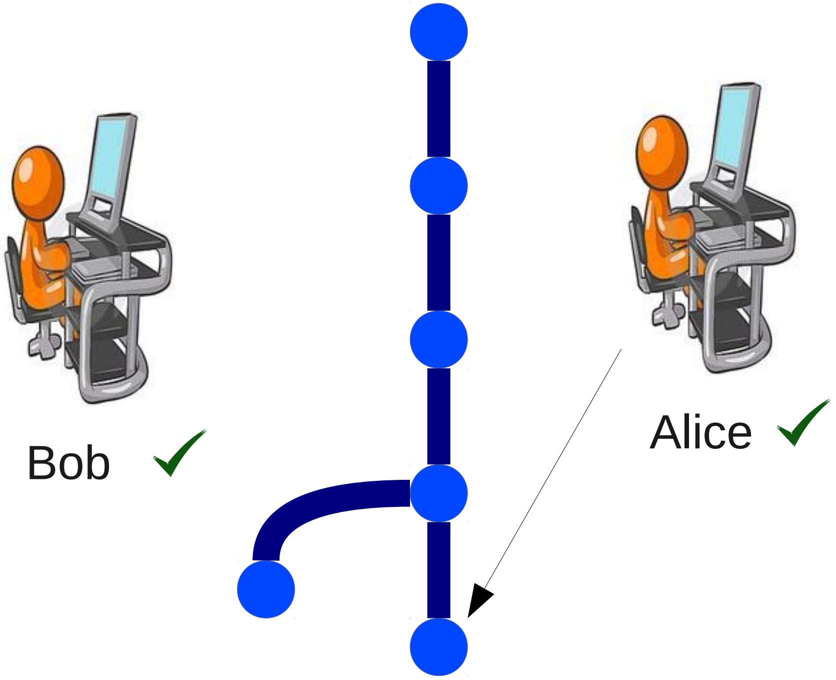
Alice ✓

main branch

“Bob's little experiment”

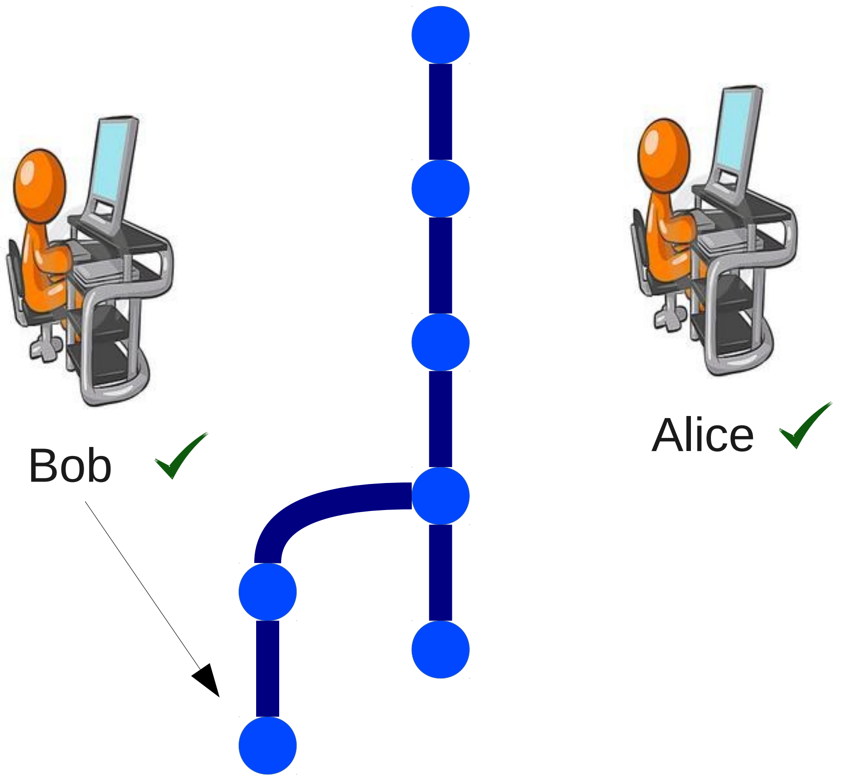
Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch



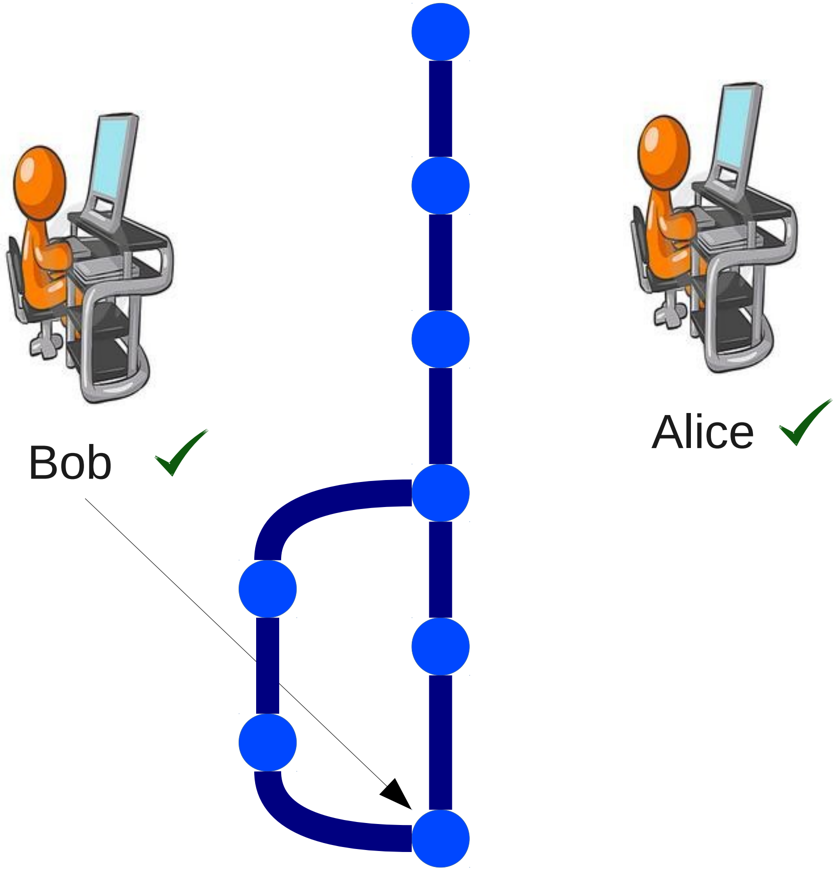
Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch



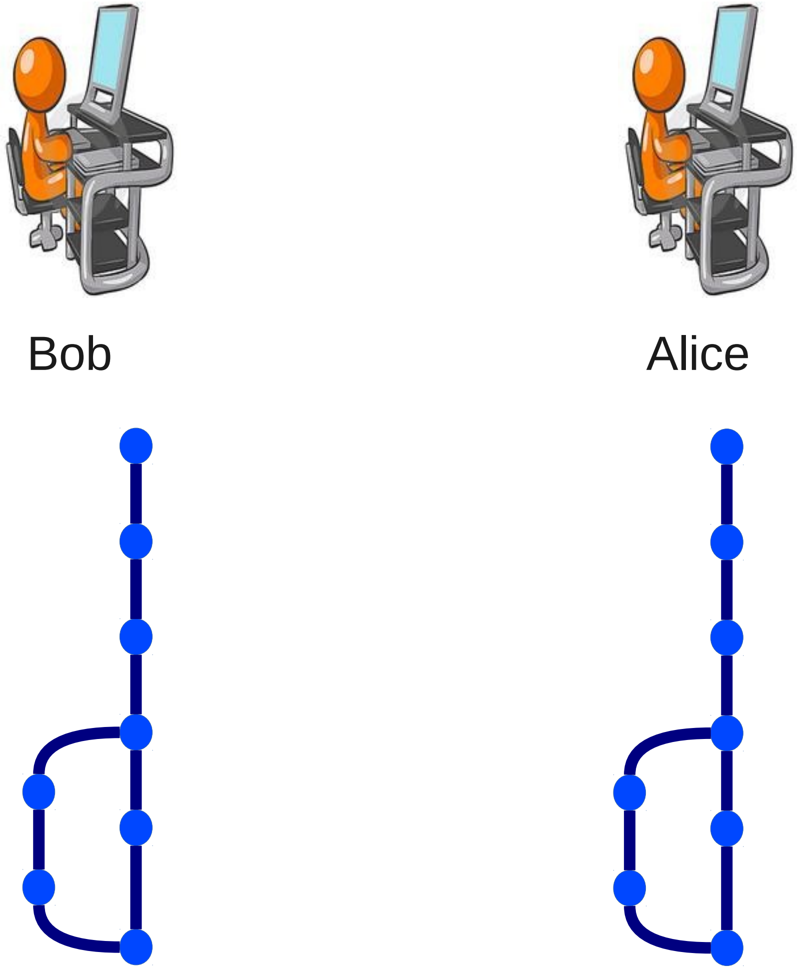
Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch
  - Merge



Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch
  - Merge
- DVCS model:



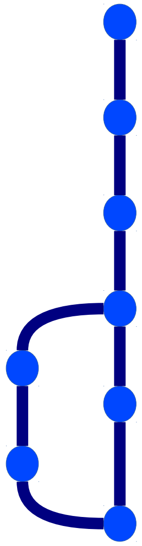
Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch
  - Merge

- DVCS model:

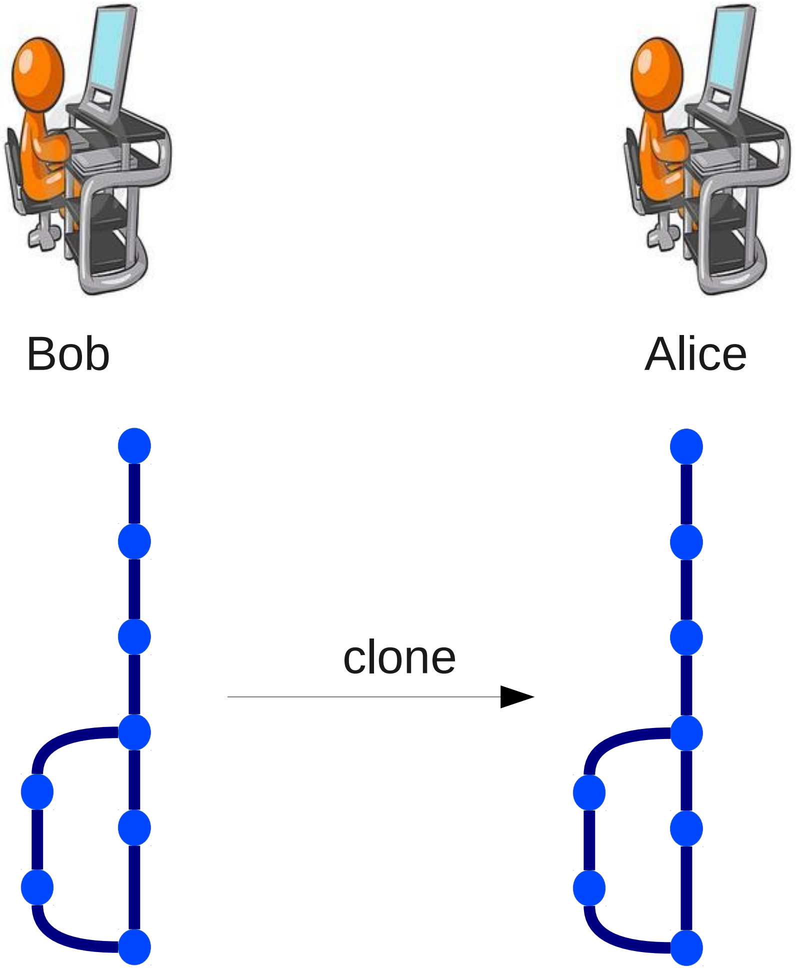
Bob

Alice



Version Control

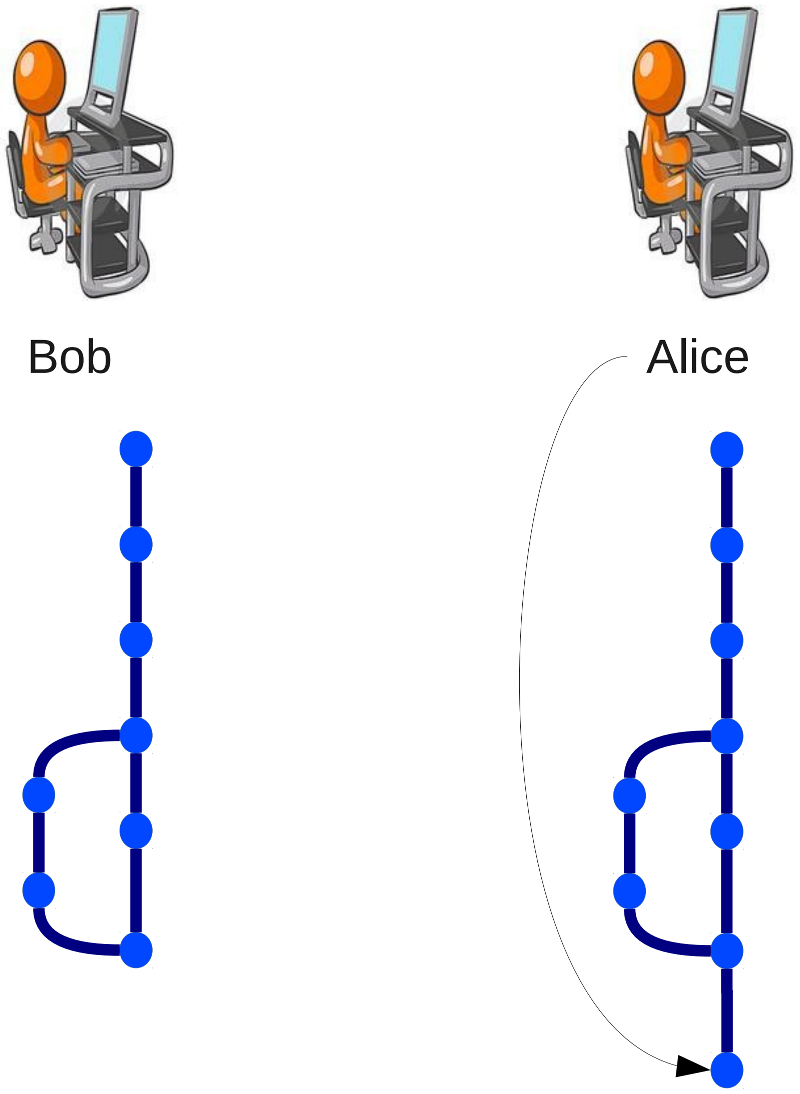
- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch
  - Merge
- DVCS model:
  - Clone/fork



Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch
  - Merge

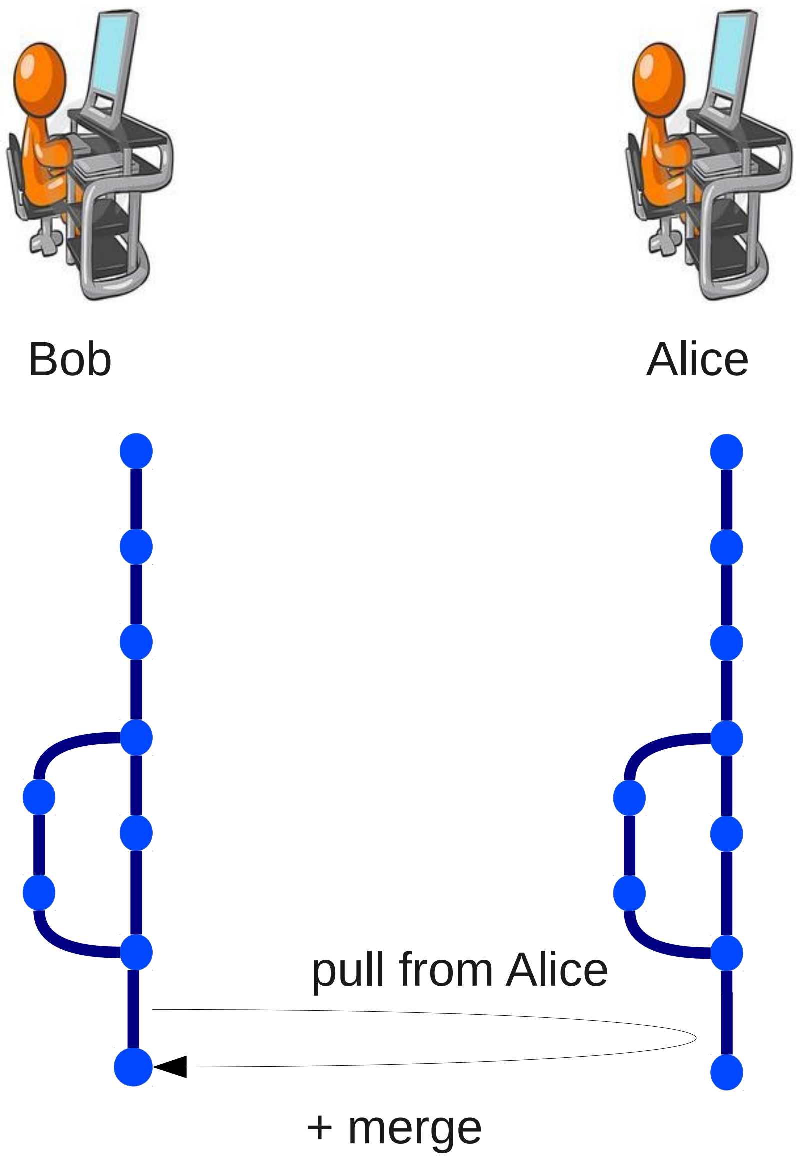
- DVCS model:
  - Clone/fork
  - Local Commits



Version Control

- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch
  - Merge

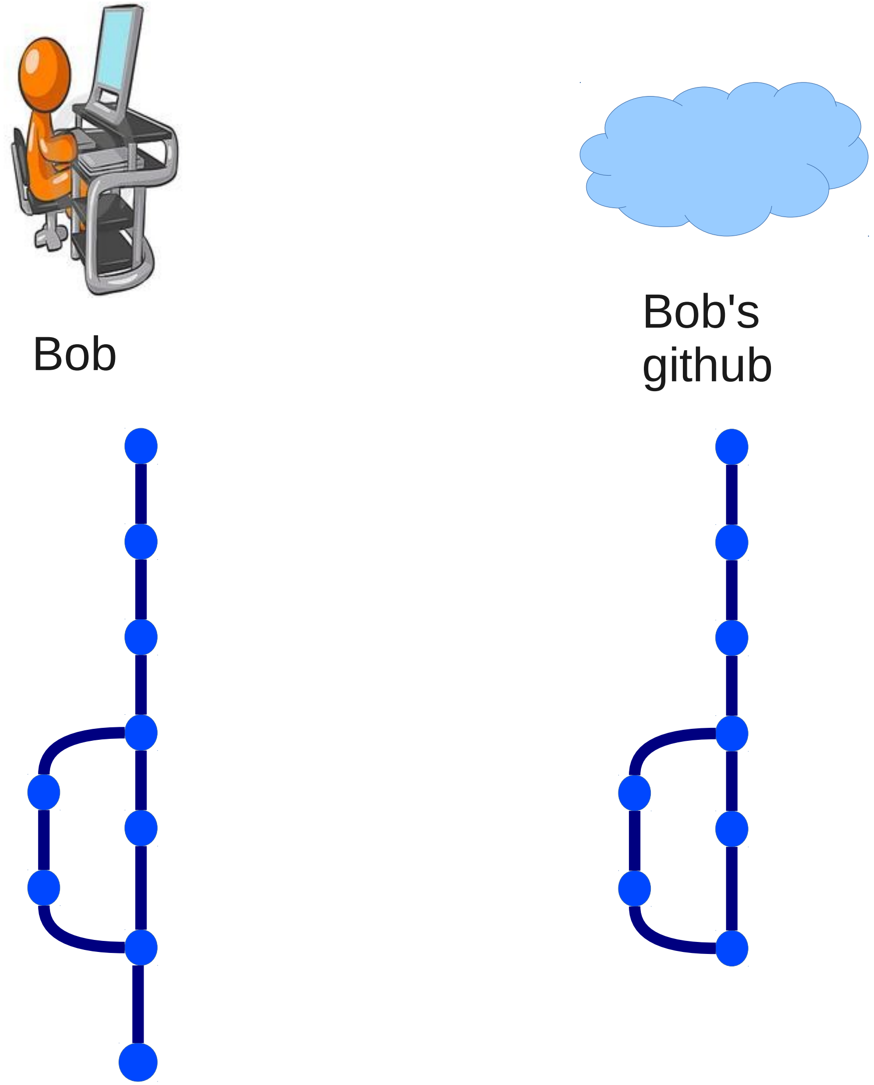
- DVCS model:
  - Clone/fork
  - Local Commits
  - Pull



Version Control

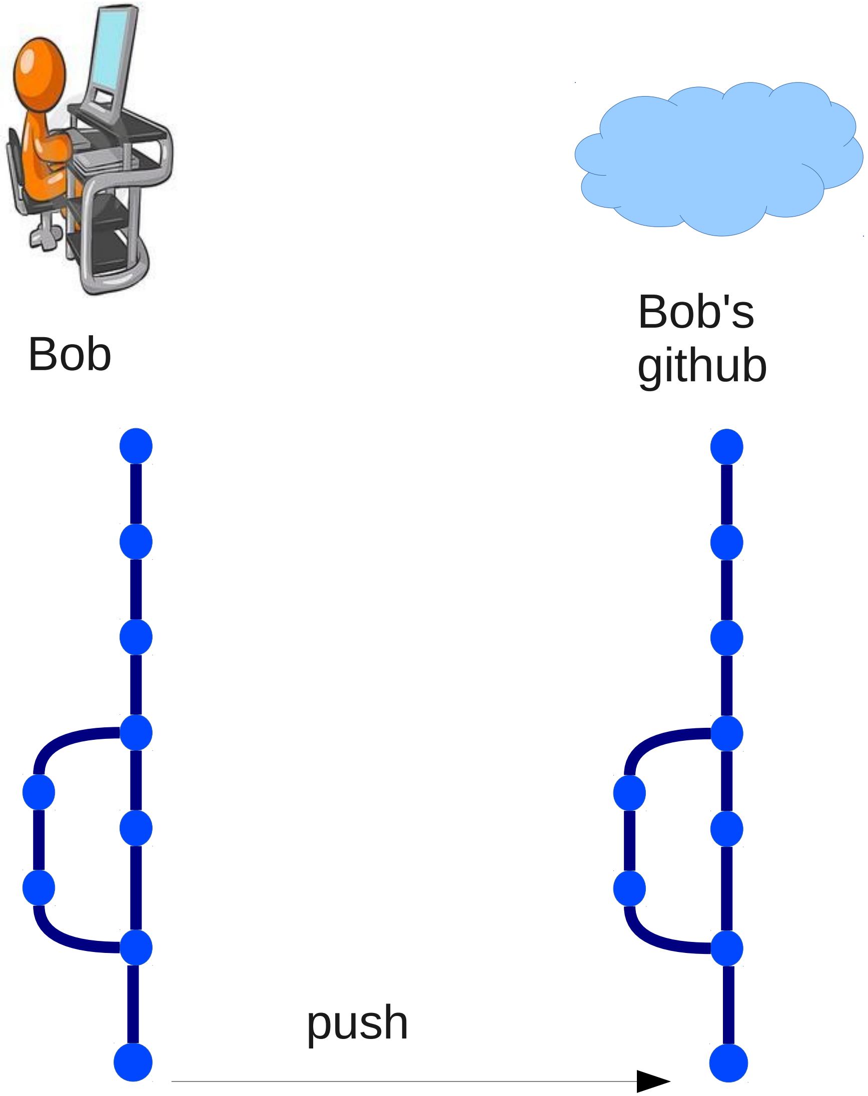
- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch
  - Merge

- DVCS model:
  - Clone/fork
  - Local Commits
  - Pull
  - Push to remote repo



Version Control

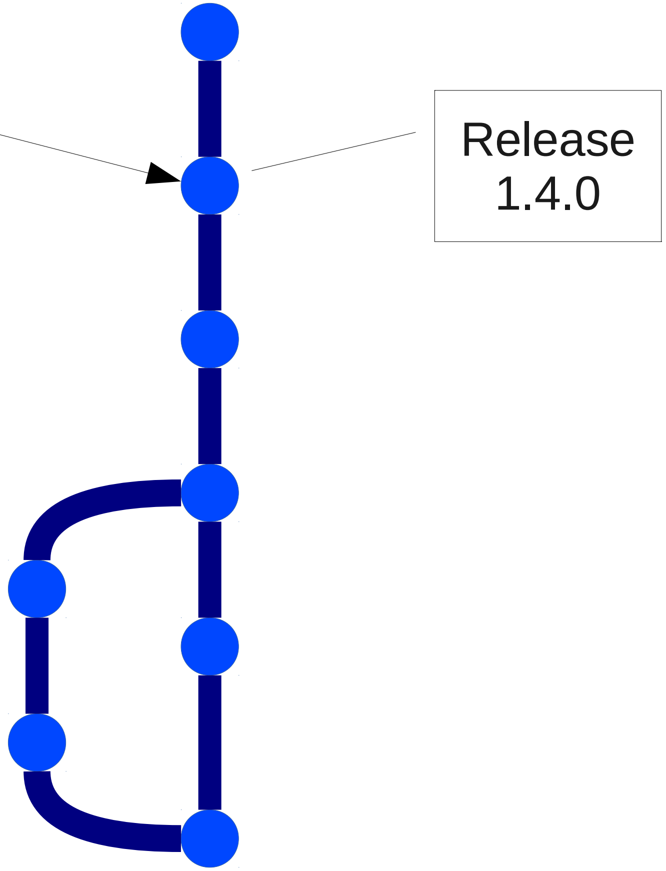
- Main tasks:
  - Create repository
  - Define access control
  - Commit (to repo)
  - Update (from repo)
  - Branch
  - Merge
- DVCS model:
  - Clone/fork
  - Local Commits
  - Pull
  - Push to remote repo



Version Control

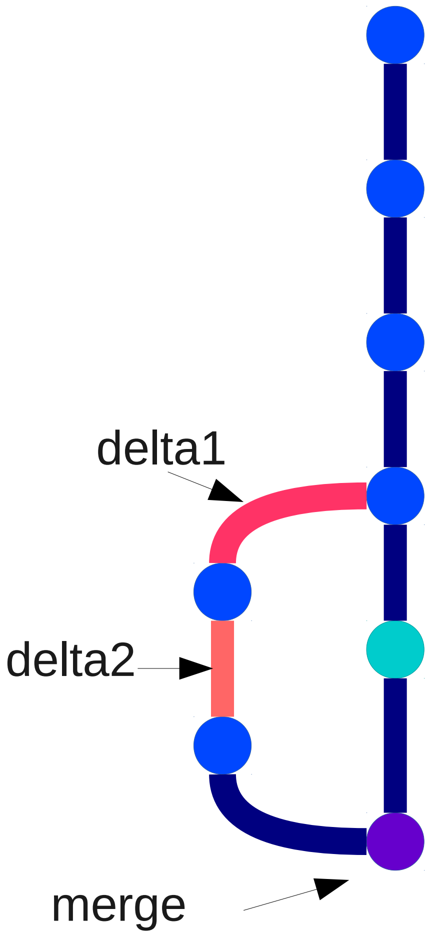
- Each node represents a snapshot of repo history

- “state” view
- Alternatively, can be recorded as “delta” from previous state



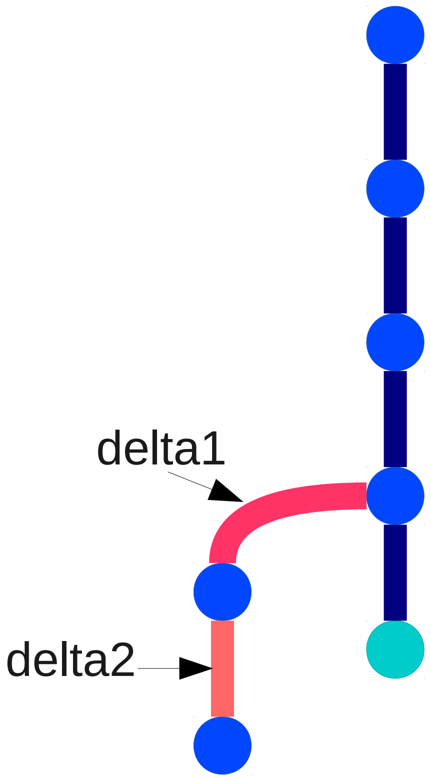
Version Control

- With deltas you can “replay” changes on a repo state (git rebase)



Version Control

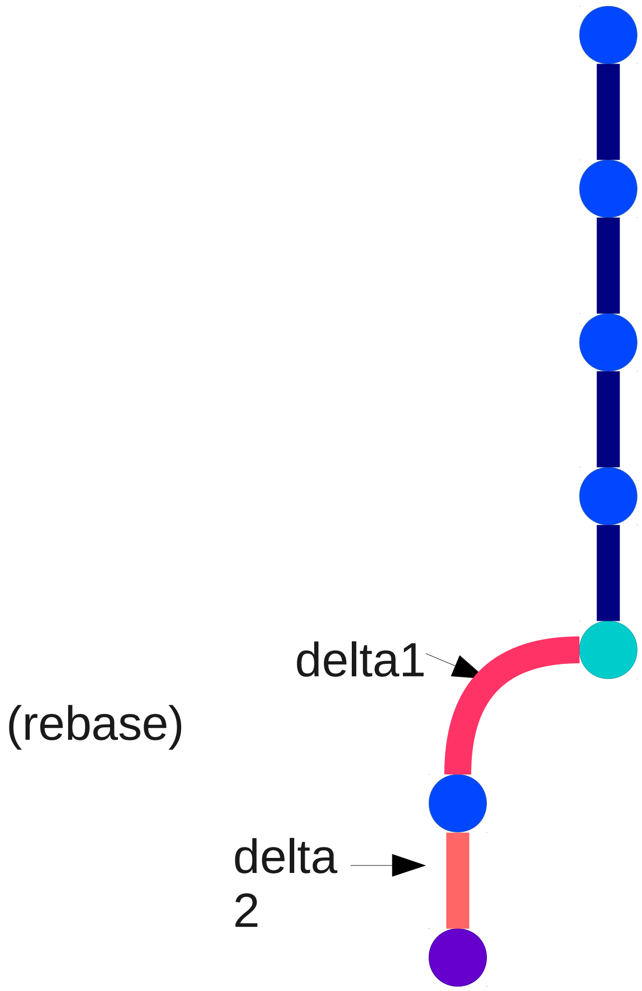
- With deltas you can “replay” changes on a repo state (git rebase)



(before merge)

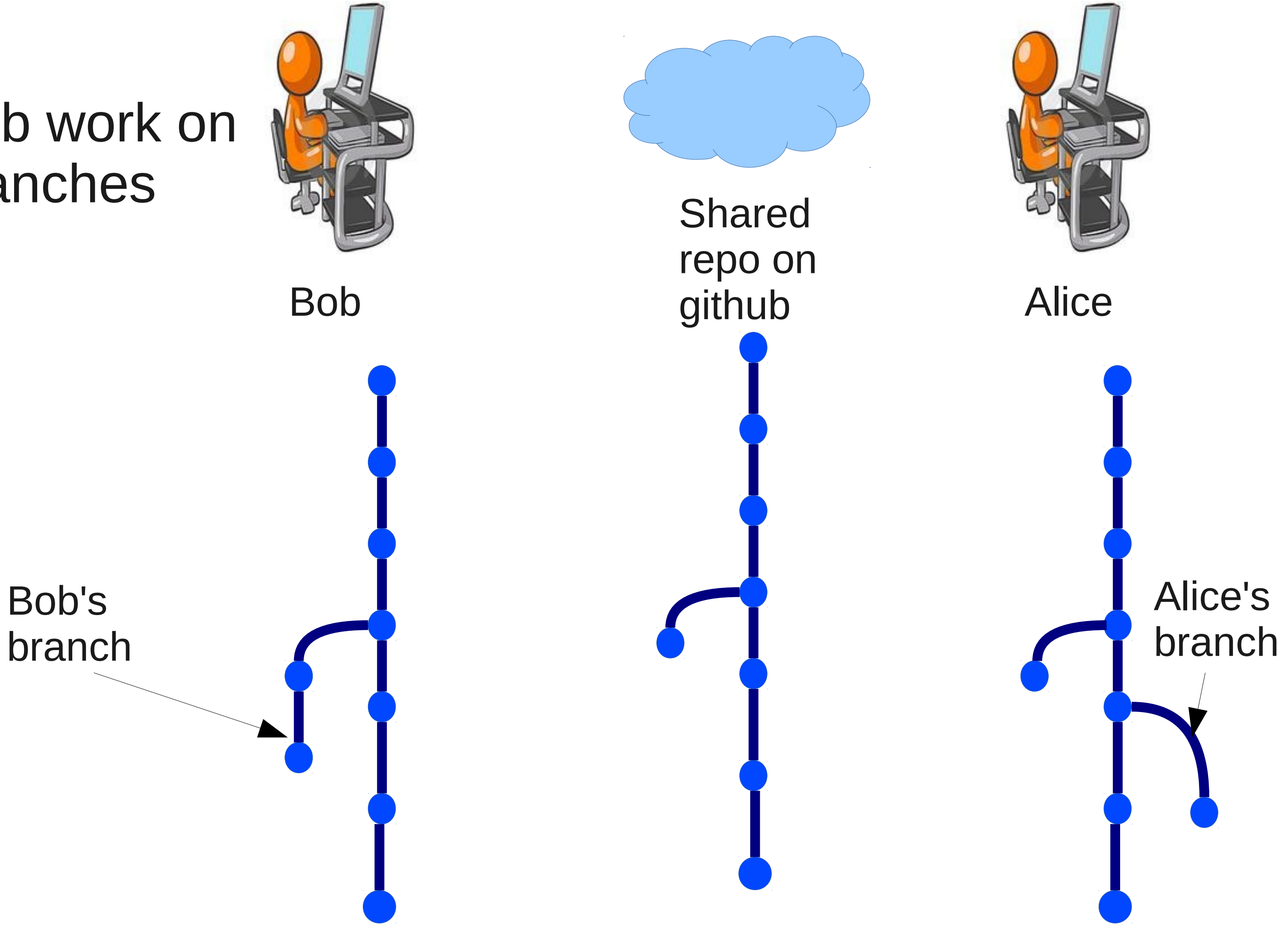
Version Control

- With deltas you can “replay” changes on a repo state (git rebase)



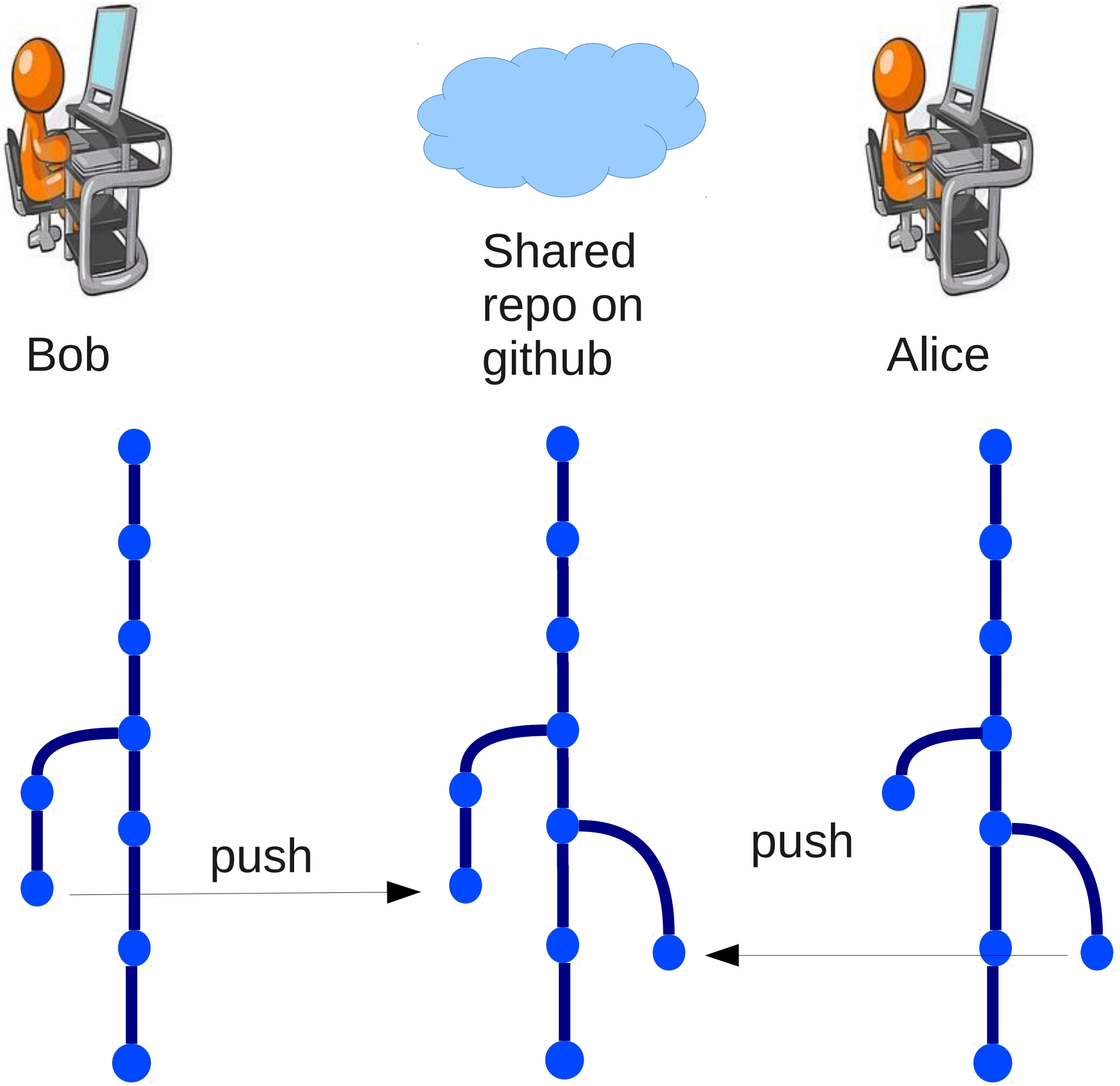
Version Control: working with github

- Alice and Bob work on their own branches



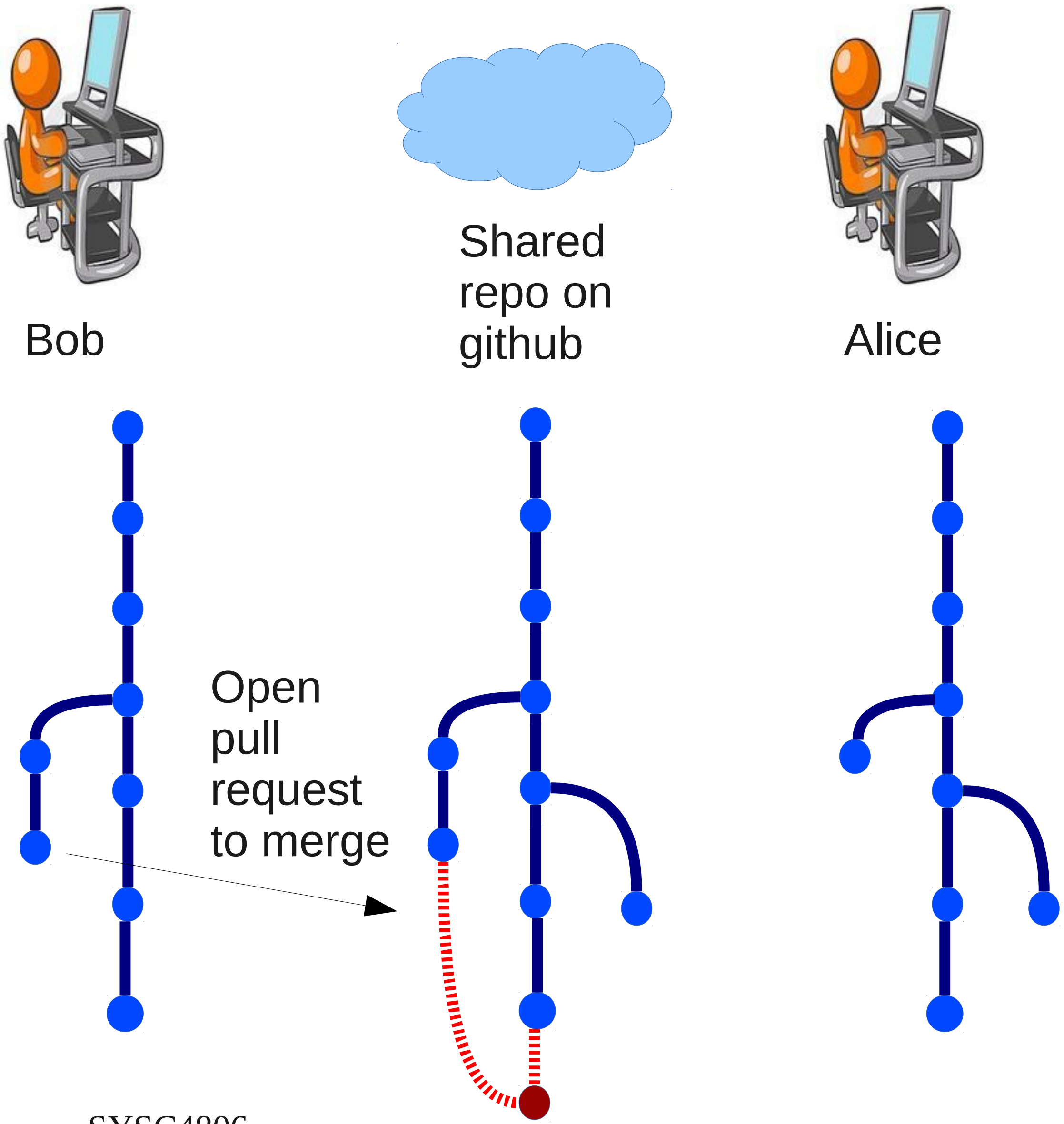
Version Control: working with github

- Alice and Bob work on their own branches
- Each push to github



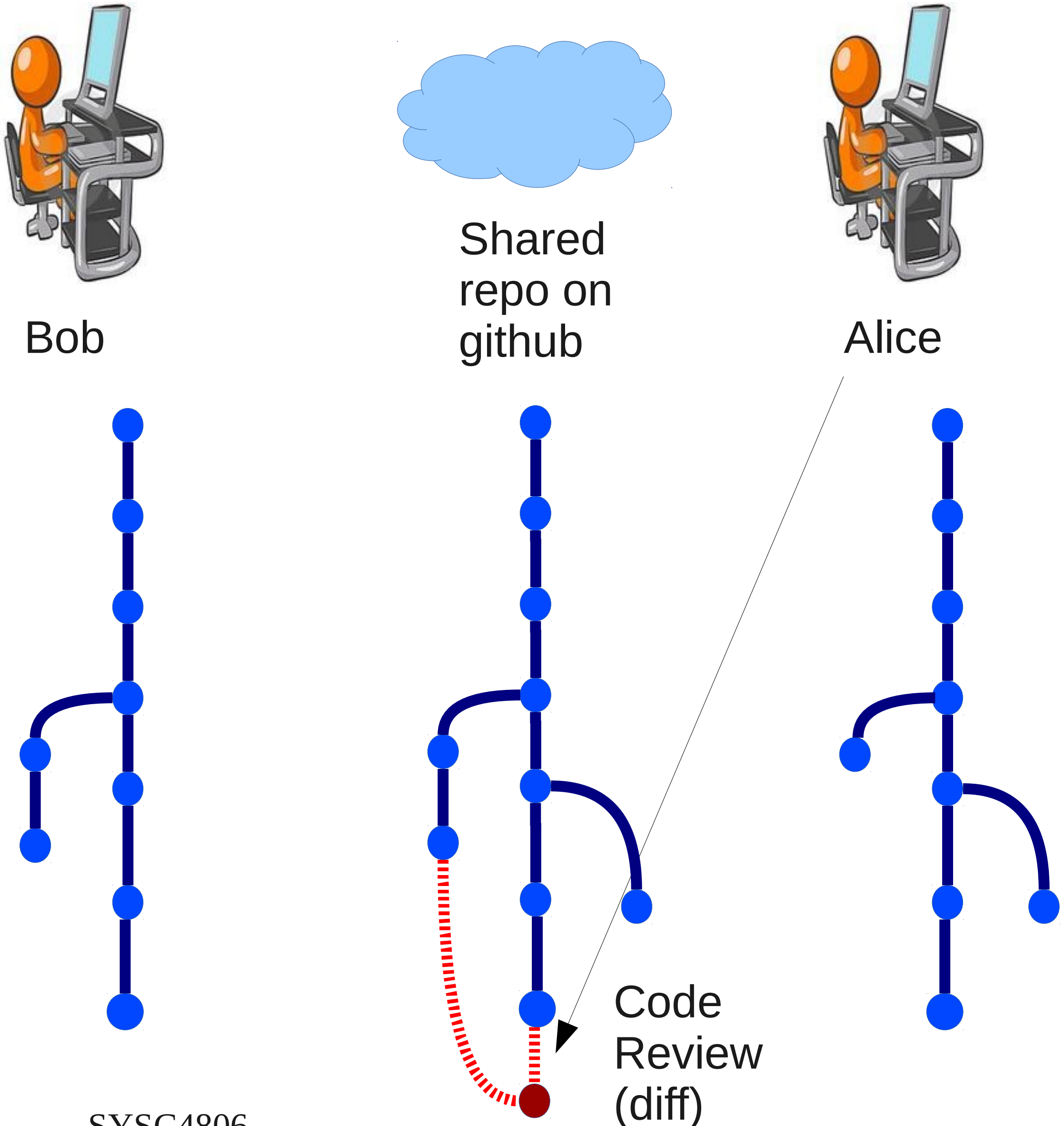
Version Control: working with github

- Alice and Bob work on their own branches
- Each push to github
- Bob opens a pull request



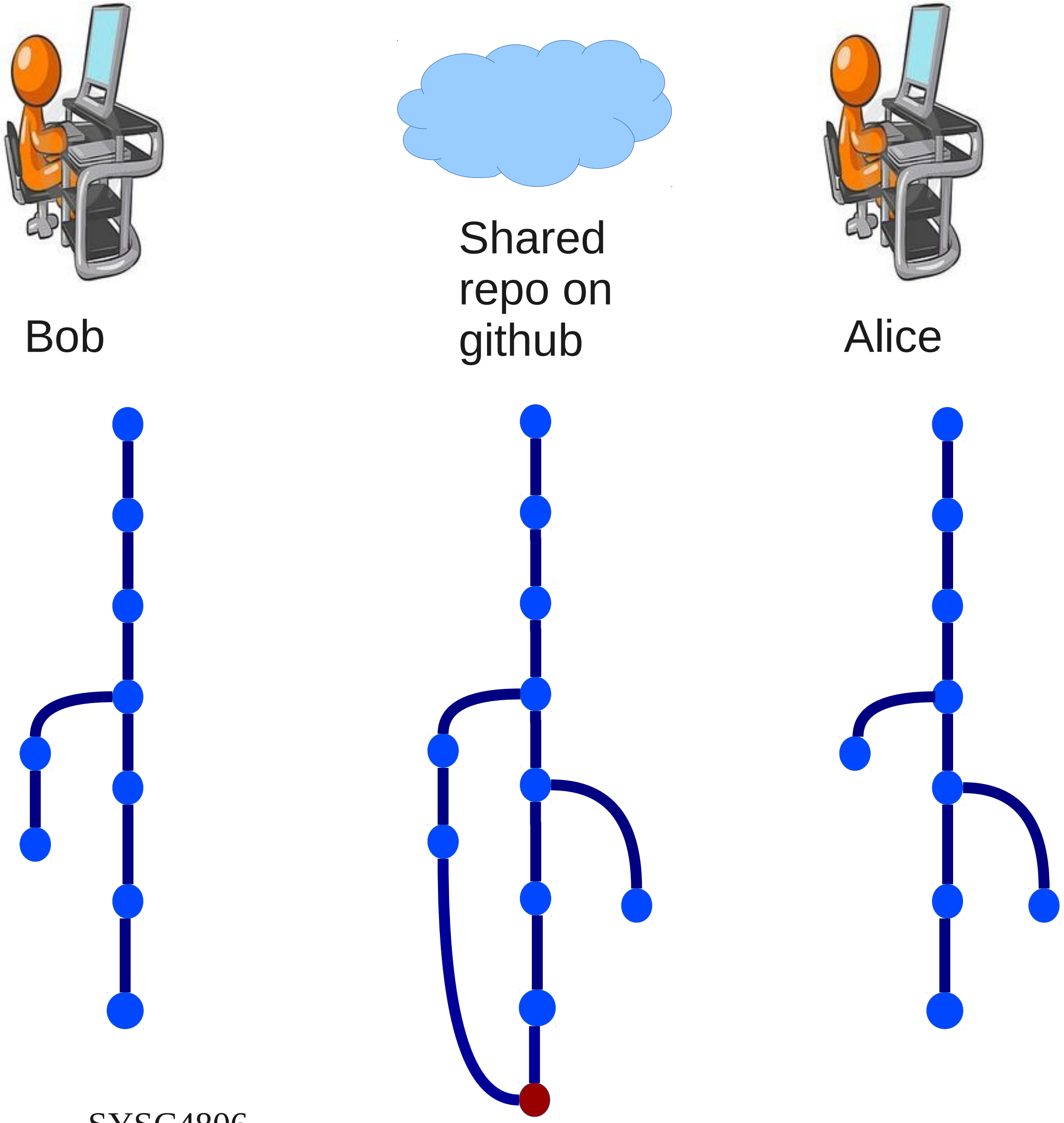
Version Control: working with github

- Alice and Bob work on their own branches
- Each push to github
- Bob opens a pull request
- Alice reviews it, comments,
  ...
  accepts



Version Control: working with github

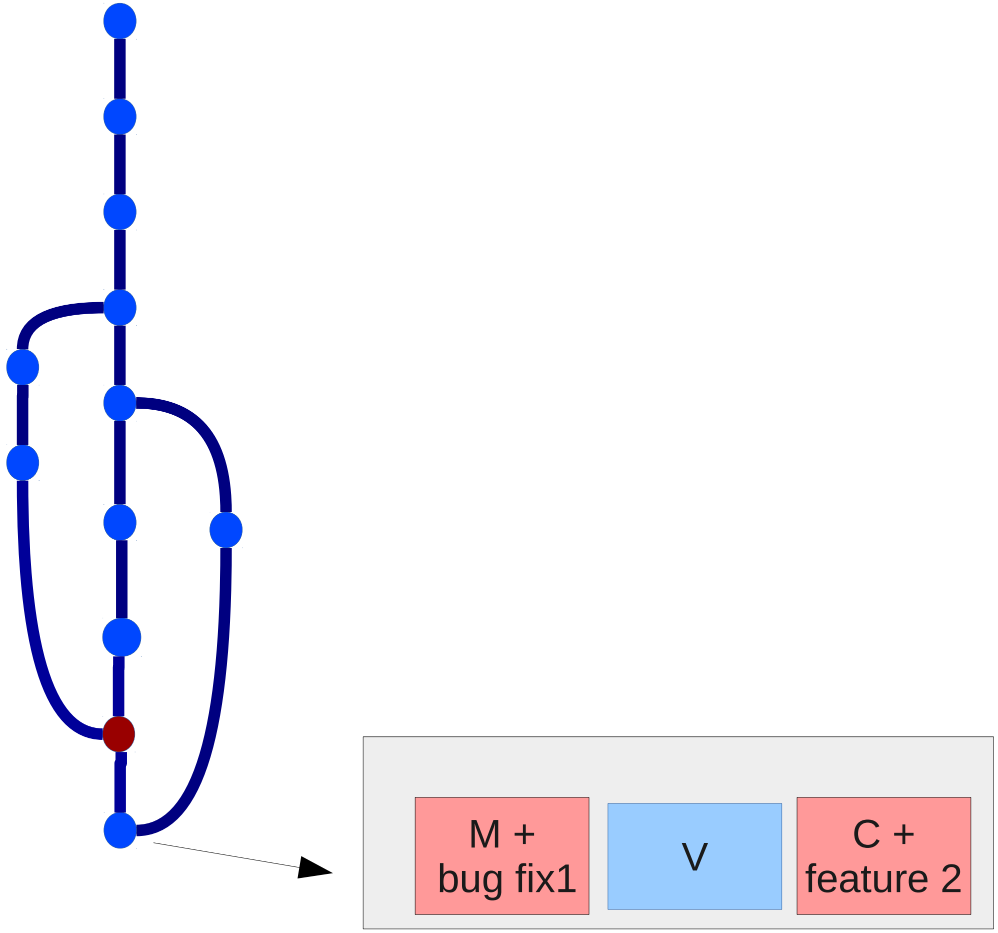
- Alice and Bob work on their own branches
- Each push to github
- Bob opens a pull request
- Alice reviews it, comments,
  ...
  accepts
- Merge is finalized.



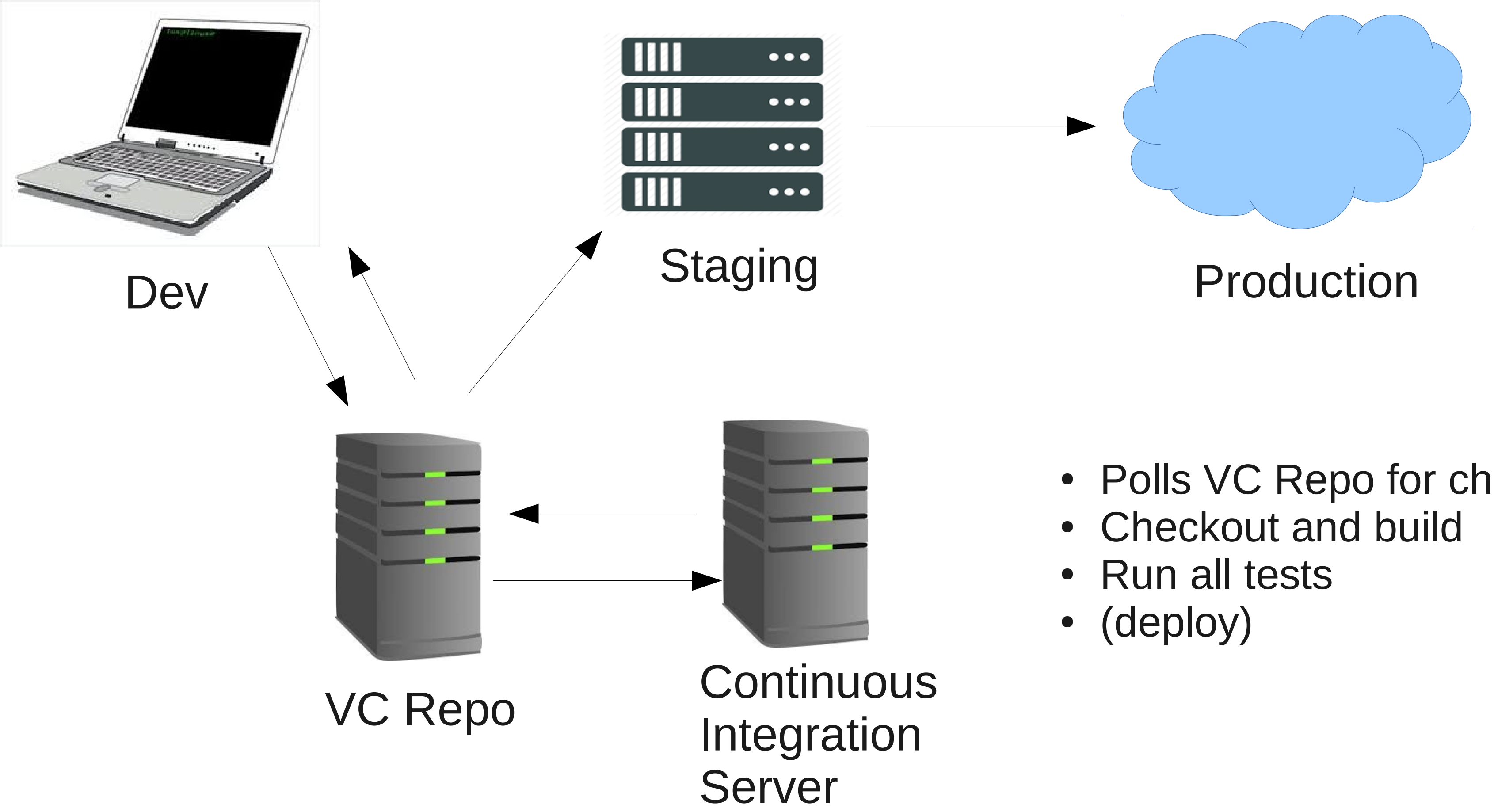
Integration

- Alice and Bob's changes have been merged.

  Can we release?



Continuous Integration



Dev

Staging

Production

VC Repo

Continuous Integration Server

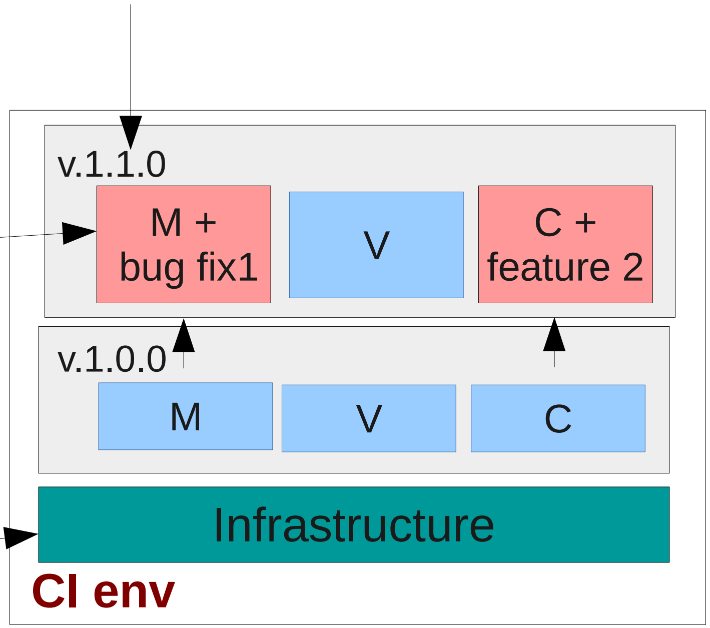
- Polls VC Repo for changes
- Checkout and build
- Run all tests
- (deploy)

Version Control + Continuous Integration

- At each change, integration tests are run
- M & C work together
- System works on prod infrastructure

Policy: only release (or merge to master) when all tests pass (CI)

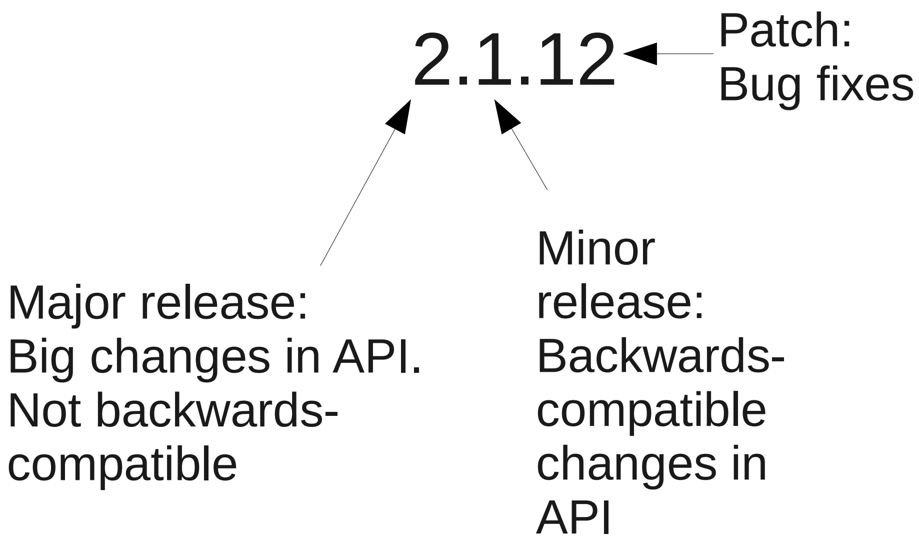
keep track of all different versions of components



Infrastructure is built from document in repo; dev = prod

Semantic Versioning

- How to number releases ?

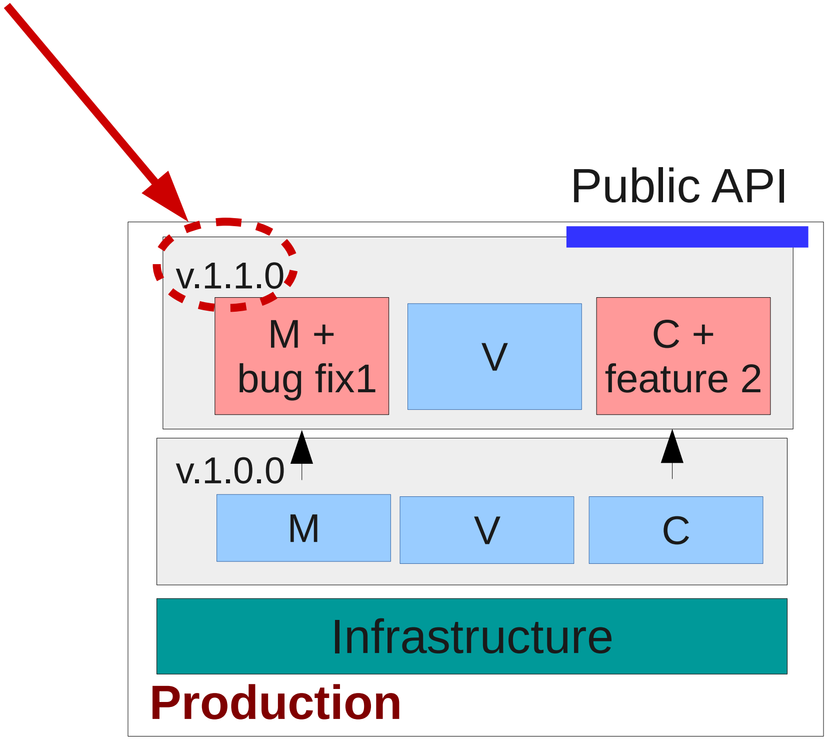


2.1.12

Patch:
Bug fixes

Major release:
Big changes in API.
Not backwards-compatible

Minor release:
Backwards-compatible changes in API



Public API

v.1.1.0

M +
bug fix1

V

C +
feature 2

v.1.0.0

M

V

C

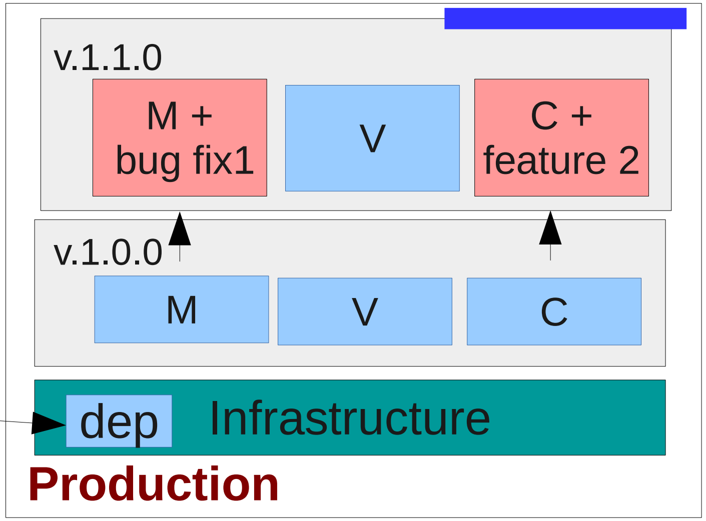
Infrastructure

Production

Dependency Management

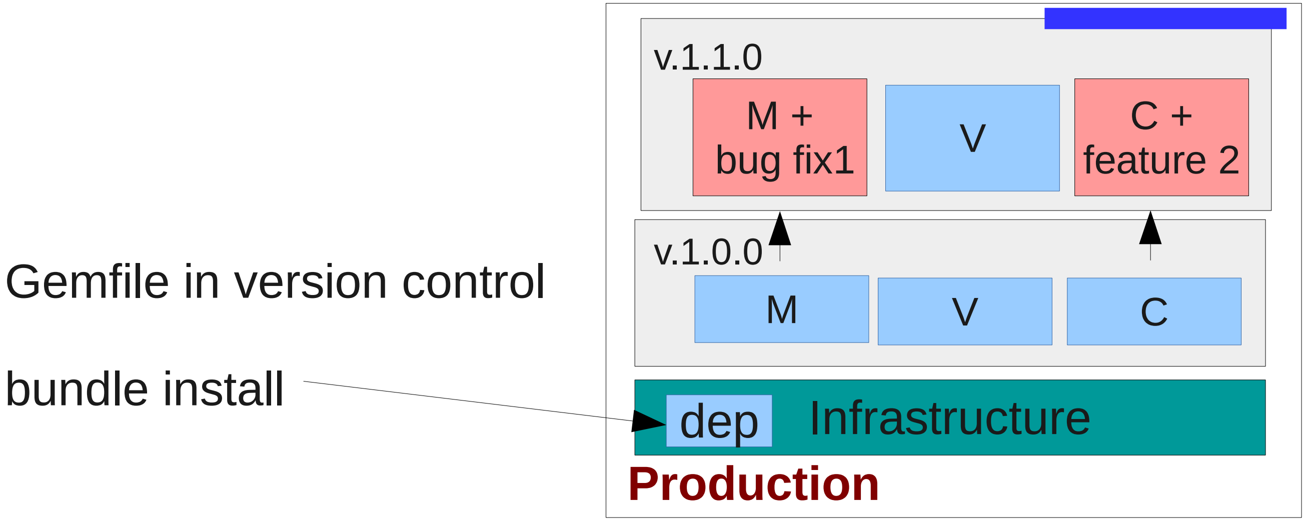
- For automated build / deployment, how are dependencies managed ?

*Do **M** and **C** have new dependencies?*



Dependency Management

- Declarative declaration of dependencies (e.g. Gemfile)
- +tools to automatically retrieve and install them (e.g. bundler)



Gemfile in version control

bundle install

Dependency Management

Other tools:

- Python pip
- Java: ant, maven (pom.xml), gradle...

(ant/maven/etc. also handle build automation)

Configuration Management

Summary:

- *Version Control* keeps track of all config. items
  - “Infrastructure as code” makes infrastructure a config item
  - Include dependency declaration
- Integration testing ensures promotions work together

  => test often (continuously) to find problems early
- Release often when limited impact on end-users

  (weekly, daily...)
- Major releases require announcements / documentation (e.g. release notes, “discover the new gmail”...)
- For interdependent systems, use semantic versioning