Moving Boost to Git

Taking Advantage of Distributed Version Control

Beman Dawes

May 22, 2012

Show of hands

- Already familiar with Git?
- Use Git regularly?

Ask Questions! Interrupt! Please!

Motivation for this session

- To inform you about Git and about Modularization, as runup for a straw poll.
- Build concensus.

Session Coverage

- Why Change? - The big picture.
- Git Resources.
- Quick Git Tutorial.
- Pros and cons of Git.
- Workflows.
- Modularizing Boost using Git.
- Action Plan?

Why Change?

Motivations

- Boost needs to scale up for more libs
- Developer preference, driven by technical benefits
- First step of Boost 2.0 effort to improve all Boost infrastructure

Git Resources

- Pro Git book – http://progit.org/book/
- Git - SVN Crash Course – http://git.or.cz/course/svn.html
- GitHub - http://github.com/
- TortoiseGit - http://code.google.com/p/tortoisegit/
- Git Flow – see separate slide.
- Your favorite search engine.

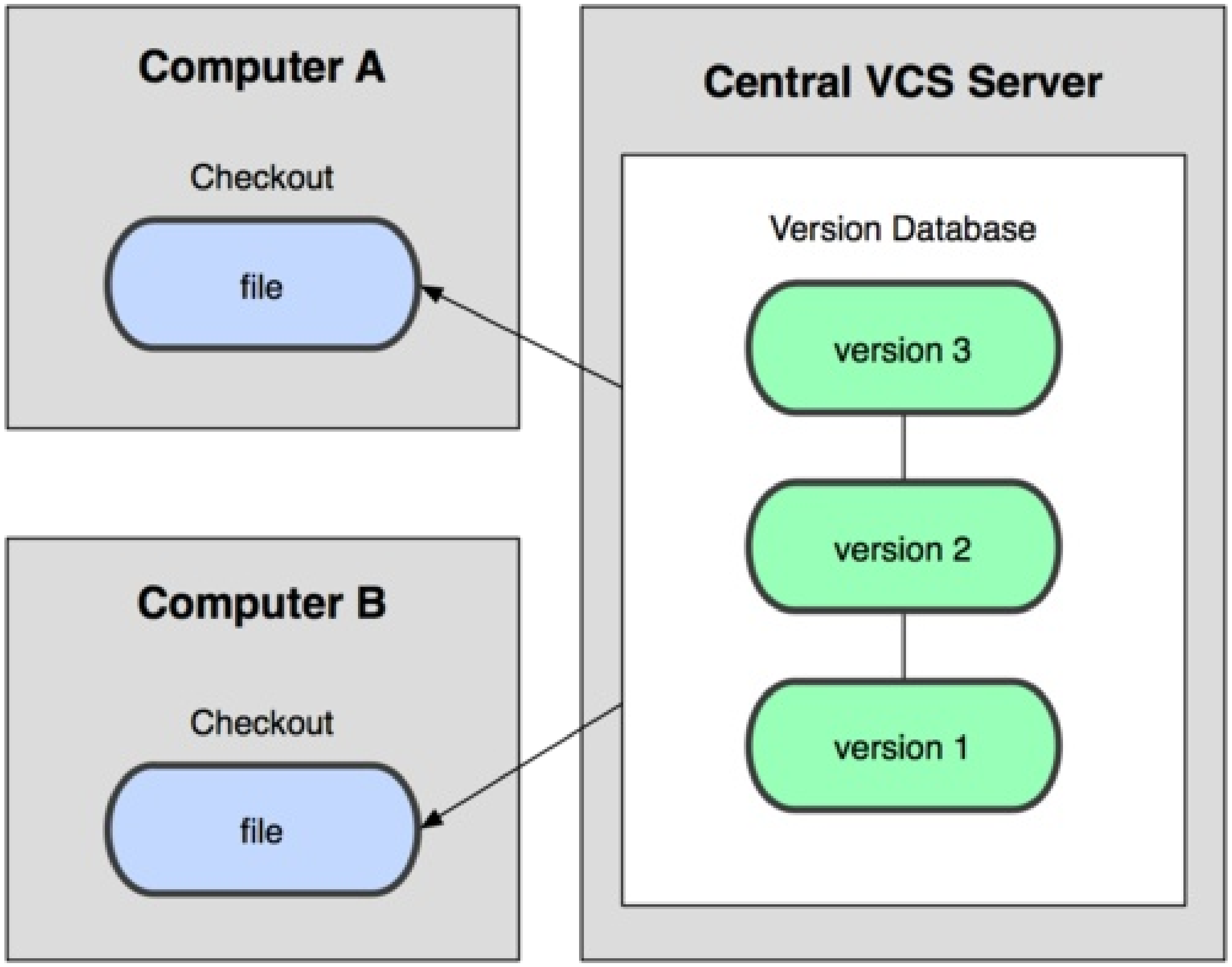
Quick Git Tutorial Design

Version Control Taxonomy

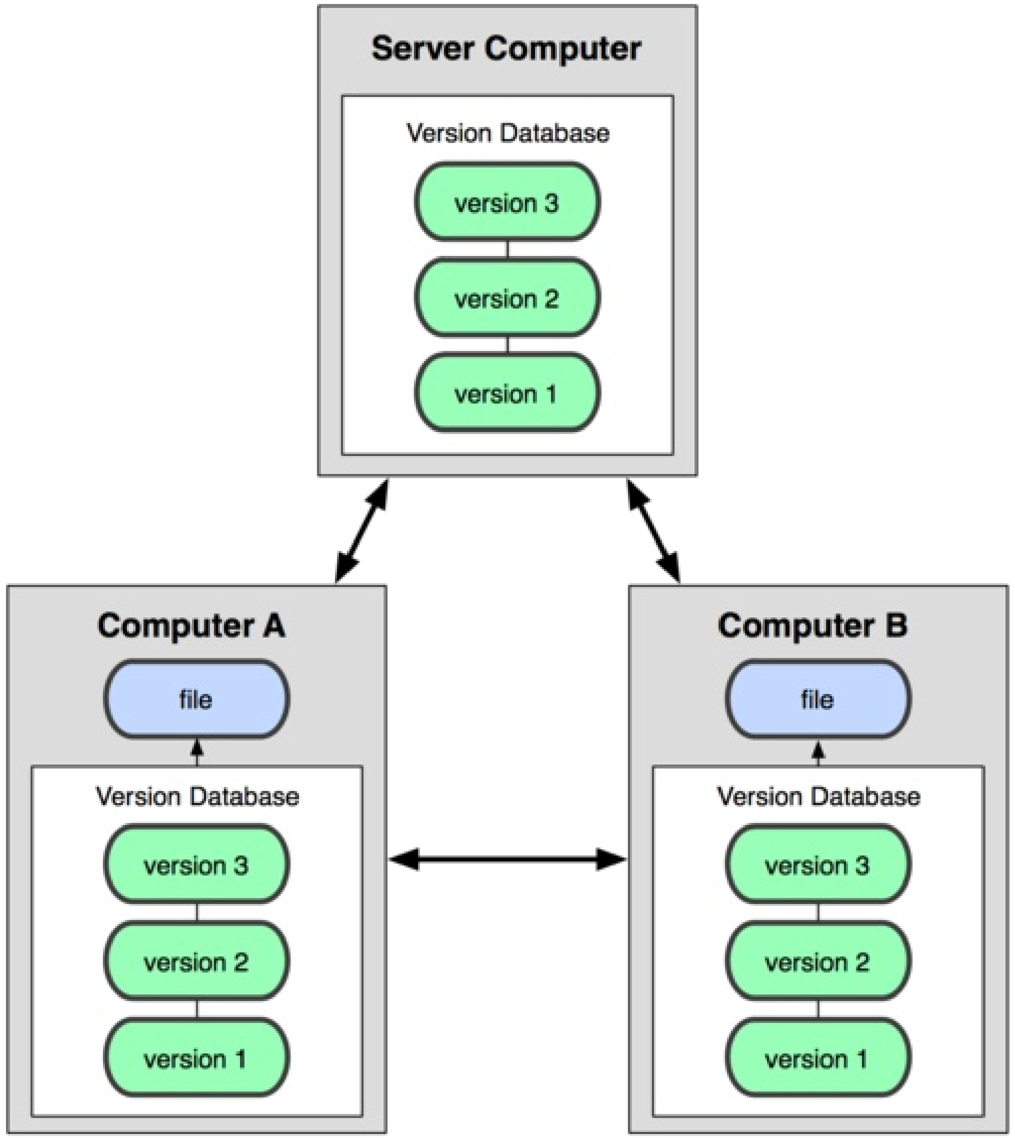
- Centralized version control system (CVCS)
  - A central repository contains all history, branches, tags.
  - A working copy contains only a view of the files as of a specific sub-tree at a specific point-in-time and nothing more.
- Distributed version control system (DVCS)
  - Every copy, both public and private, is a full repository containing all history, branches, tags, etc.
  - A working copy makes visible a view of the files as of a specific sub-tree at a specific point-in-time, but the full repo is still there in a hidden directory.

- Subversion is a Centralized Version Control System
- Git is a Distributed Version Control System

Centralized VCS (Pro Git Book)

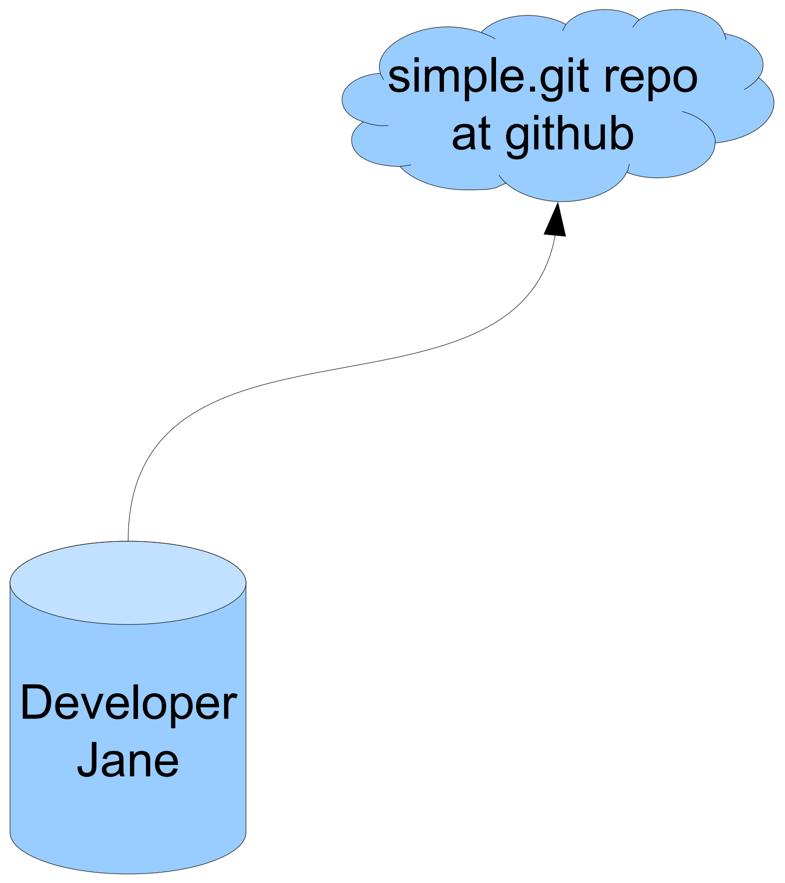


Distributed VCS (Pro Git Book)



Quick Git Tutorial Example

Simple Library

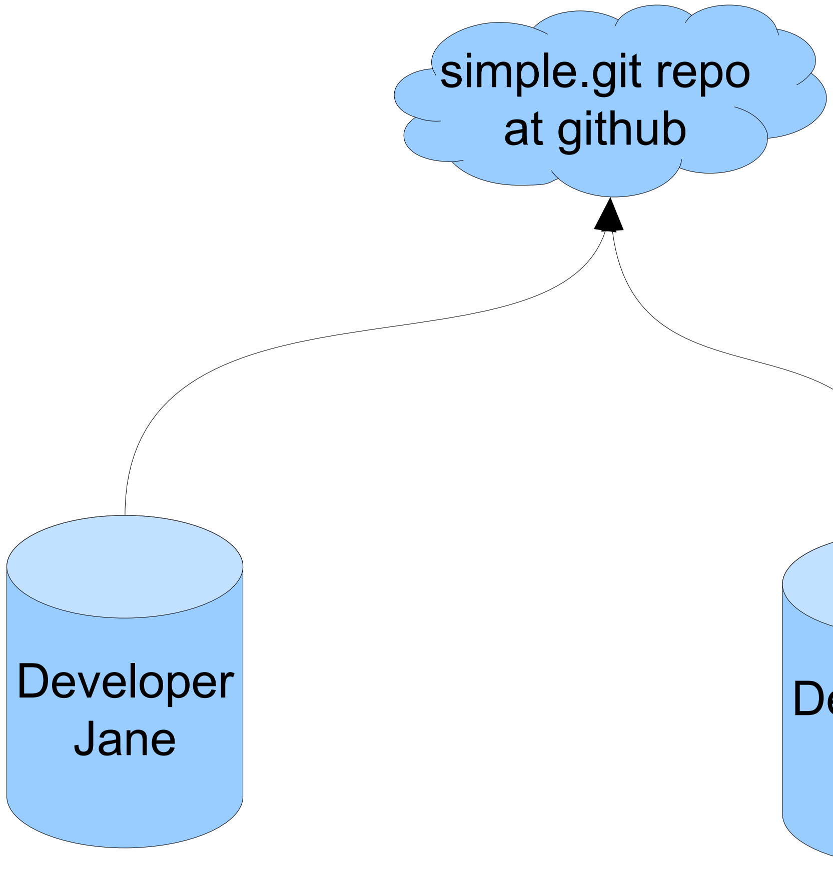


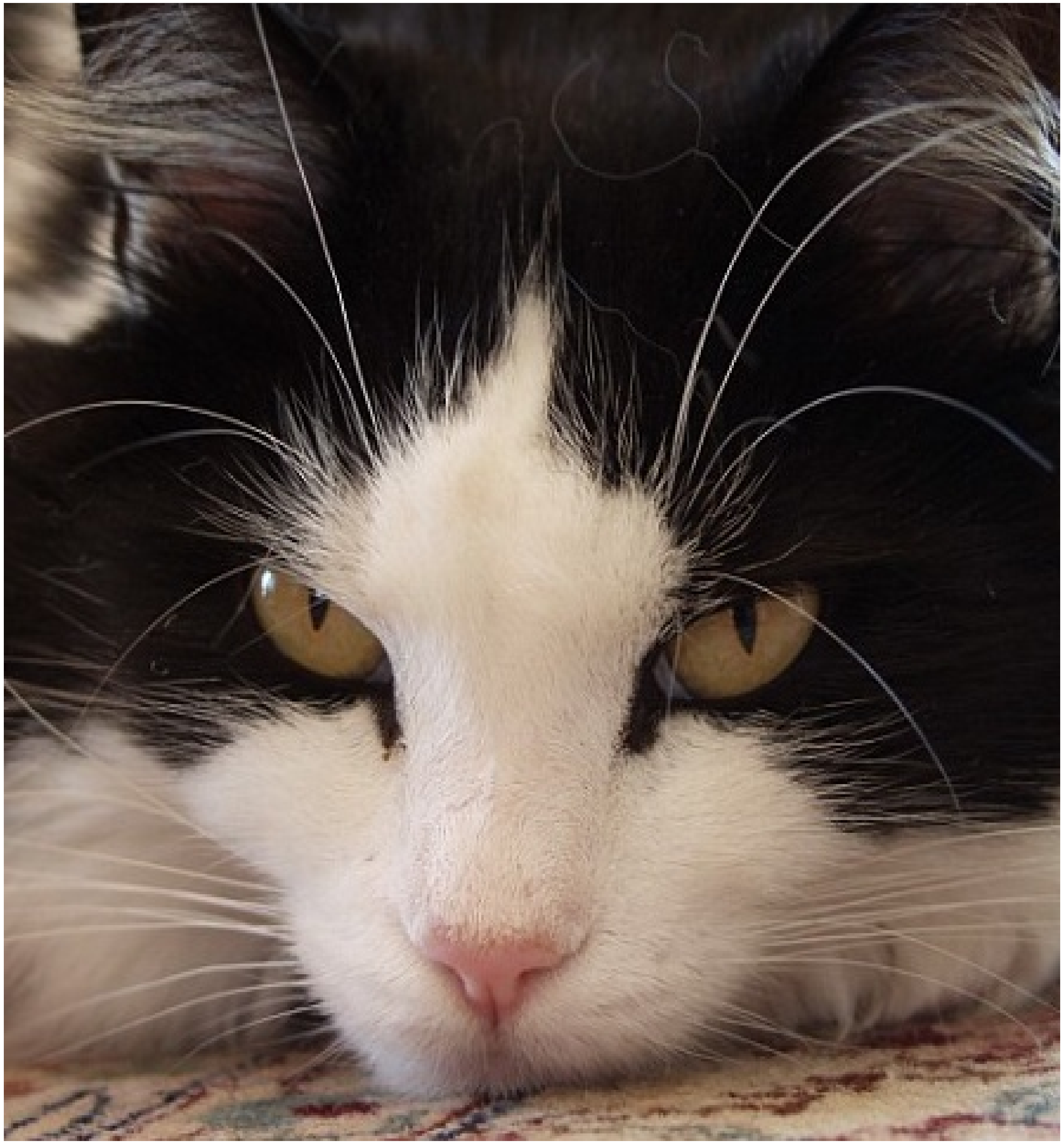
```
cd /projects
git help
git help clone
git clone git@github.com:Beman/simple
cd simple
dir /ah
ls -a
echo This is a README file >README
type README
git add README
git help commit
git commit --dry-run
git commit -m "Initial commit"
notepad README
git status
git commit -a -m "Add period"
git help push
git push --dry-run "origin" master
git push "origin" master
…
git pull
```

Aside: Local simple.git

```
cd /cloud
git init --bare simple.git
```

Simple Library





[Add Developer Harry via TortoiseGit]

Pros and Cons of Git

First Order Git Advantages over Subversion

- Faster operations; often <u>much</u> faster.
- Can work offline, including commits, branch creation, merges.
- Additional workflow support:
  - Multi-level commits ("stage/commit/push")
  - Private local branches
- No single point of failure; implicit backup.

Second order Git Advantages

- Merging often works better and is easier, particularly for non-expert users.
  - Cumulative effect of 1st order advantages.
  - Better merging was central design goal.
- Branching often used more effectively, particularly by non-expert users.
  - Cumulative effect of 1st order advantages.
  - Aids project newbies, minor feature branches
- Innovative and improved workflows

Other Git Positives

- Nice set of minor features; e.g. *stash.*
- Extensive support for moving from Subversion.
  - git svn Command
  - TortoiseGit
  - Git - SVN Crash Course http://git.or.cz/course/svn.html
  - Etc.
- Good community support, online docs, ecosystem (i.e. GitHub).
- Network effect; Git is very widely known and used, and that reduces Boost support costs.
- Pleasant to use.

Git negatives Impacting Boost

- Changing VCS is a pain
- Trac linkage may or may not be a problem

Git Negatives Not Impacting Boost

- No support for locks
- Cloning expensive for very large repos
- Some DVCS advantages moot if other developer tools only work online
- DVCS makes obliteration particularly difficult
- Access control sometimes an issue: "Users who choose decentralized version control typically must arrange things such that access control on a per-repository basis is sufficient."

Why not Mercurial?

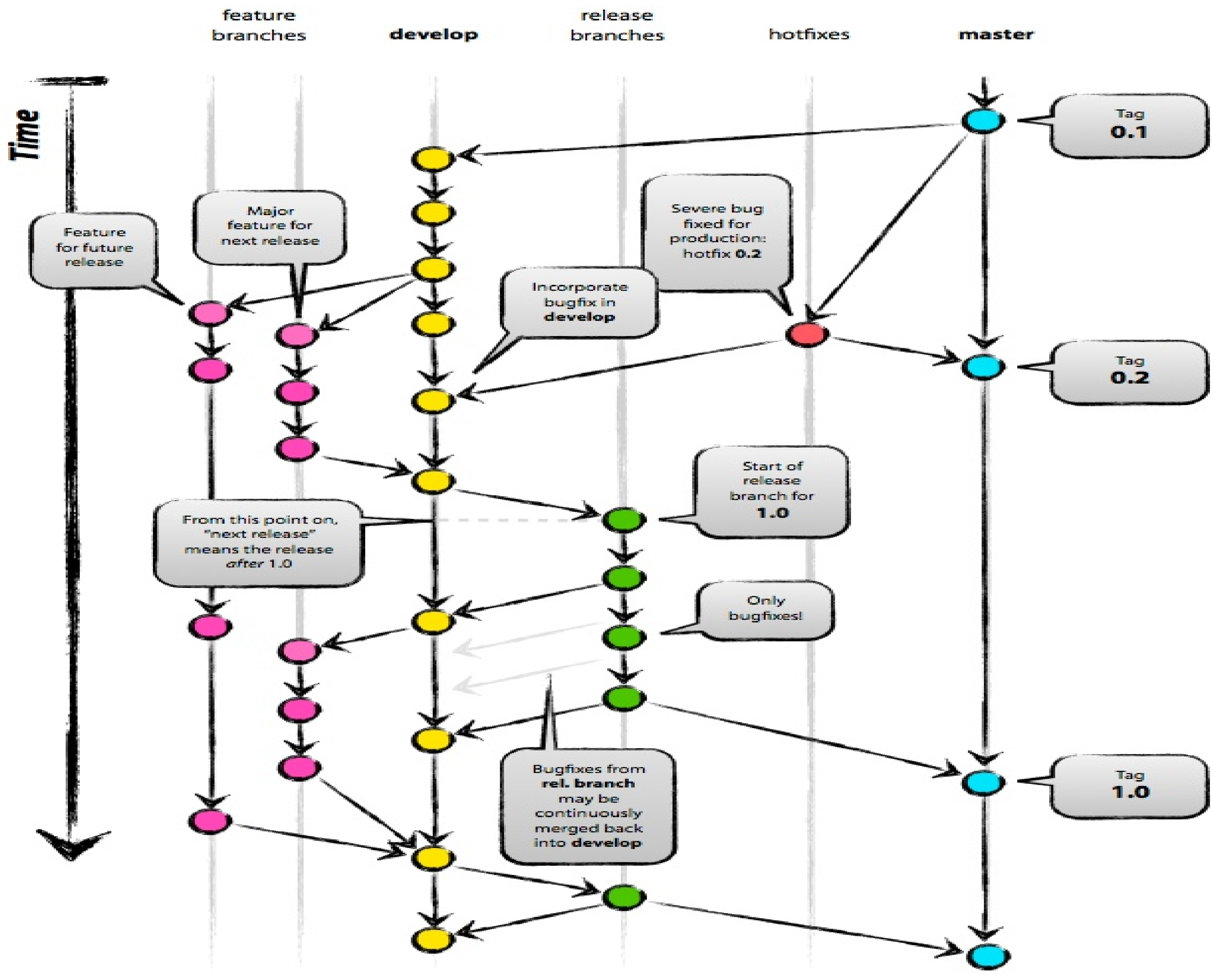
Mercurial is Git’s main competitor. It has the technical features Boost needs, has a somewhat more coherent interface, and is widely respected by DVCS aficionados.

But Git seems to be the DVCS used and preferred by more developers, both at Boost and elsewhere. So given the similarity between the two, Git wins because of network effect.

Workflows

Git Flow branching and workflow model

- Vincent Driessen's original branching model post – http://nvie.com/posts/a-successful-git-branching-model/
- Git extensions for high-level repository operations implementing the model – https://github.com/nvie/gitflow#readme (See Getting Started section to learn more about git flow)


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Severe bug fixed for production: hotfix 0.2
Major feature for next release
Feature for future release
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Tag 1.0
Bugfixes from rel. branch may be continuously merged back into develop

Modularizing Boost using Git

Motivation for Modularization

- Scaling Boost up to many more libraries implies decentralization and decoupling. Separating out per-library modules achieves that, is a well-know practice, has been studied and tested for Boost.
- Boost regression testing, release management, developer workflow, and distribution/install needs hard to meet with current structure.

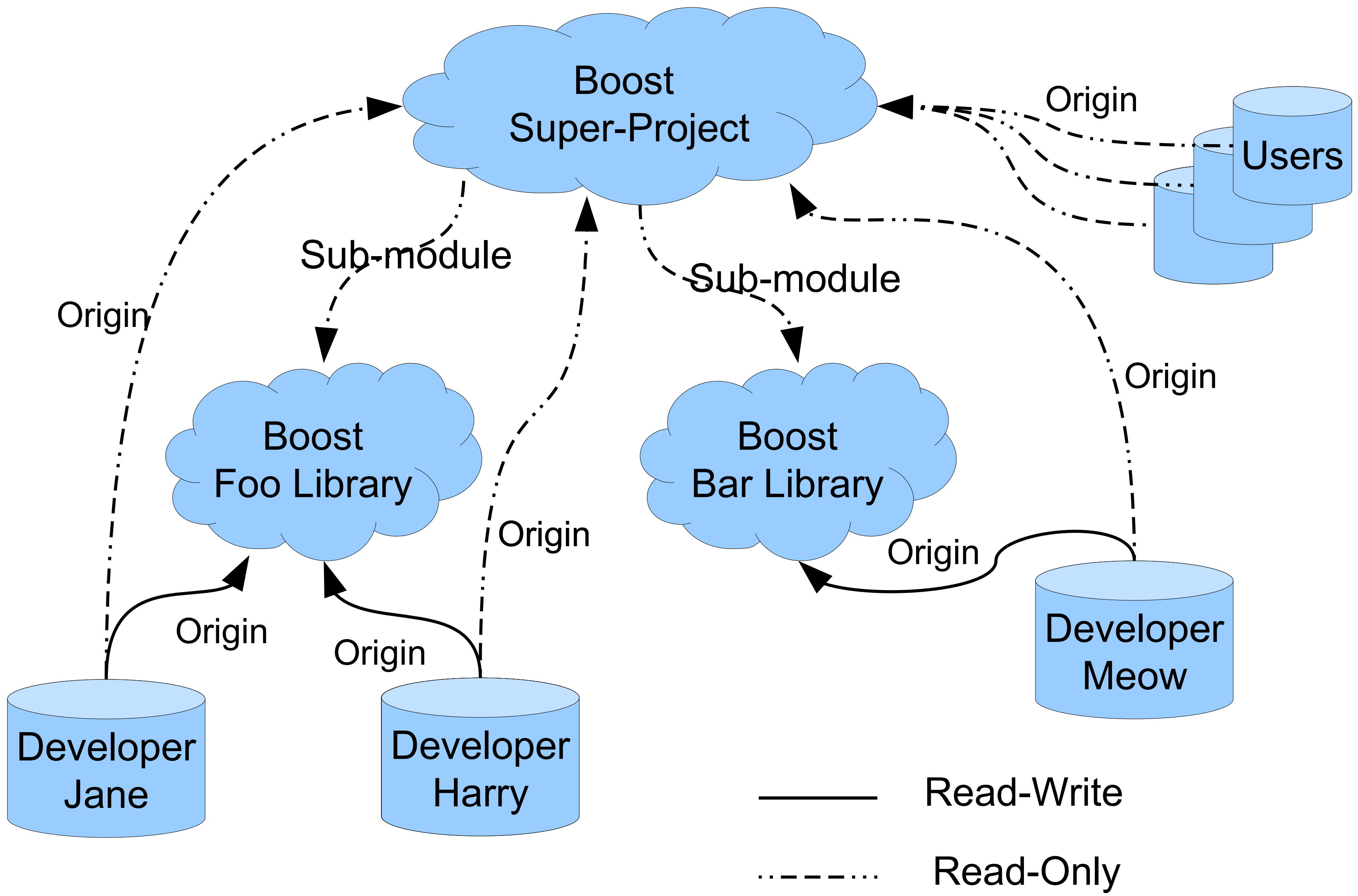
Key Modularization Requirements

- Preserve History
- Preserve User and Developer Infrastructure

Proposed Boost Modularization

- Each Boost library has its own public repository
  - Releases asynchronously from Boost super-project.
  - Write perms: The library's developers.
  - Each library chooses its own workflow and directory structure as long as basic requirements are met.
- Boost super-project with public repository
  - Each library is included as a sub-module.
  - Also includes some infrastructure, although that may eventually move to separate public repos.
  - Write perms: Boost release & infrastructure teams.
- Public repos hosted by Boost at GitHub?

Boost Repository Relationships



Try it (where network allows)

Commands…

```
git clone http://github.com/boost-lib/boost boost-modules
cd boost-modules
git submodule update --init       (16 minutes on DSL connection)
cmake -P forward_headers.cmake    (2 minutes on SSD hard-drive)
```

[Investigate with TortoiseGIT]
[Demo a bjam build]
[Show boost_test output]
[Demo a MSVC10 build]

Possible Action Plans?

Option 1a – Switch to Git, but no modularization.
Stick with current infrastructure.

Option 1b – Switch to Git and modularization.
Stick with current infrastructure.

Option 2 – Now: (1a) or (1b).
Future: Ryppl infrastructure,
modularizing first if not done yet.

Option 3 – Convert directly to Ryppl infrastructure.

Motivation for single Git + Modularization transition

- Cost of a single transition is far less than cost of two separate transitions. That holds for all cost metrics.
- The benefit of the Git transition isn’t fully realized until the modularization transition is complete.

Acknowledgments

- Troy Straszheim
- Dave Abrahams
- John Wiegley
- Daniel Pfeifer

Thank You!