Sharing Code: Distributed Version Control and Social Development

(http://www.slac.stanford.edu/~jrbl/SCW11.pdf)

What's my point?

This is about collaboration, and having choices.

What centralized source control manages with *process*, decentralized source control systems let you manage with *software*.

My Experience

- Software development with small and medium-sized groups using centralized (CVS, SVN, Perforce) and decentralized (git) tools. Startups, universities, ISPs, nonprofits.
- I work on INSPIRE (http://inspirebeta.net) as part of an international collaboration (CERN, DESY, Fermilab, SLAC). At any given time I'm actively coordinating with 1-3 other people and our repos receive commits from ~ 12 in 3 widely separated timezones.
- Seven years as a sysadmin trained me well in human failings.

I Love git

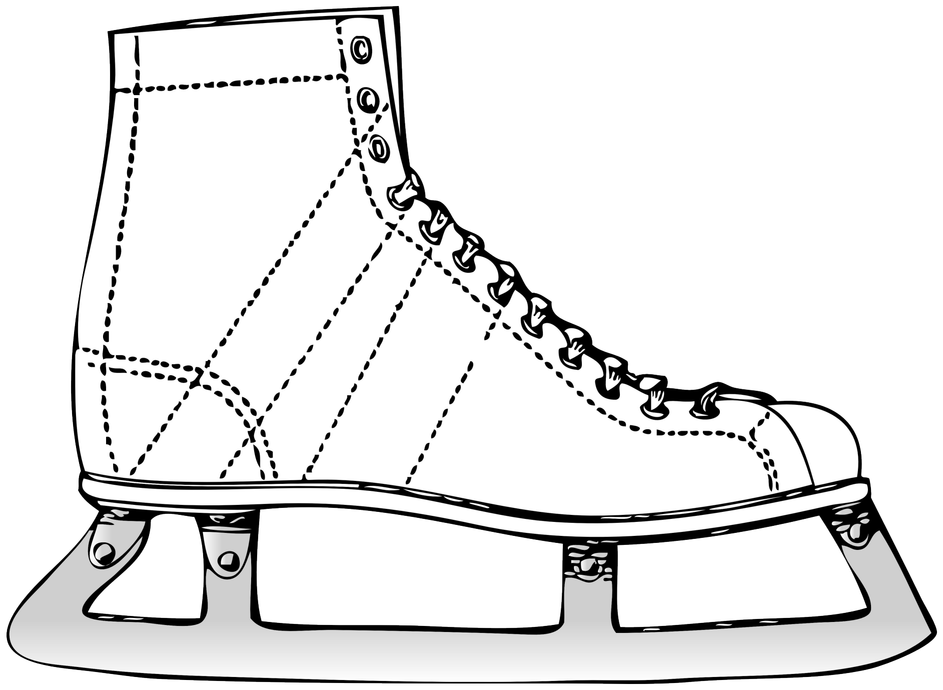
- But you're welcome to use something else.
  - Popular DVCS systems include:
    - bazaar (bzr), git, mercurial (hg)
    - Nowadays they're roughly equivalent in functionality with zealous supporters of each claiming their way is best.
      - For the record, I use vi, too.

Tools Matter

Don't believe me?

Ice.

Sucks, right?



Ice.

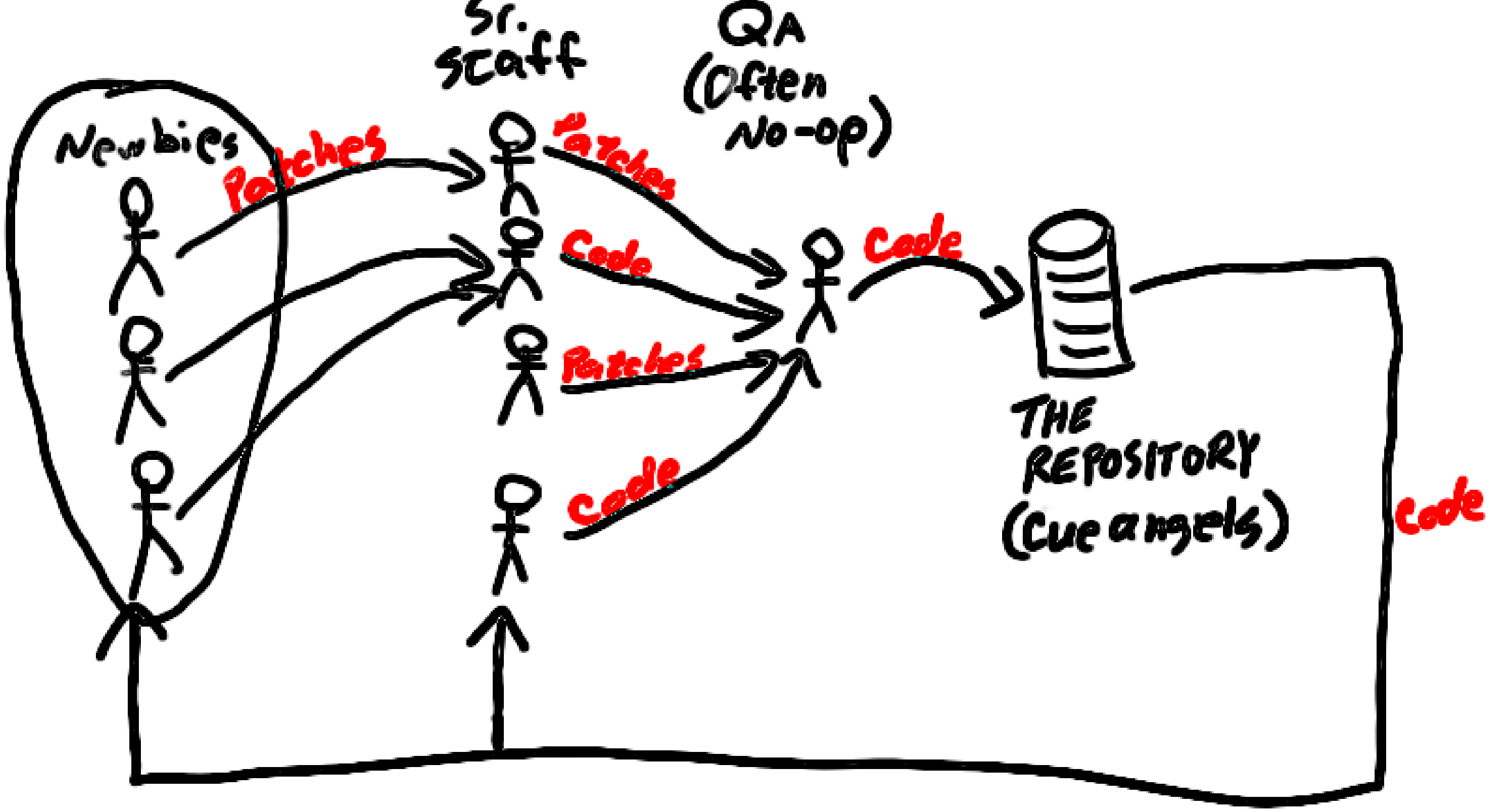
So awesome.

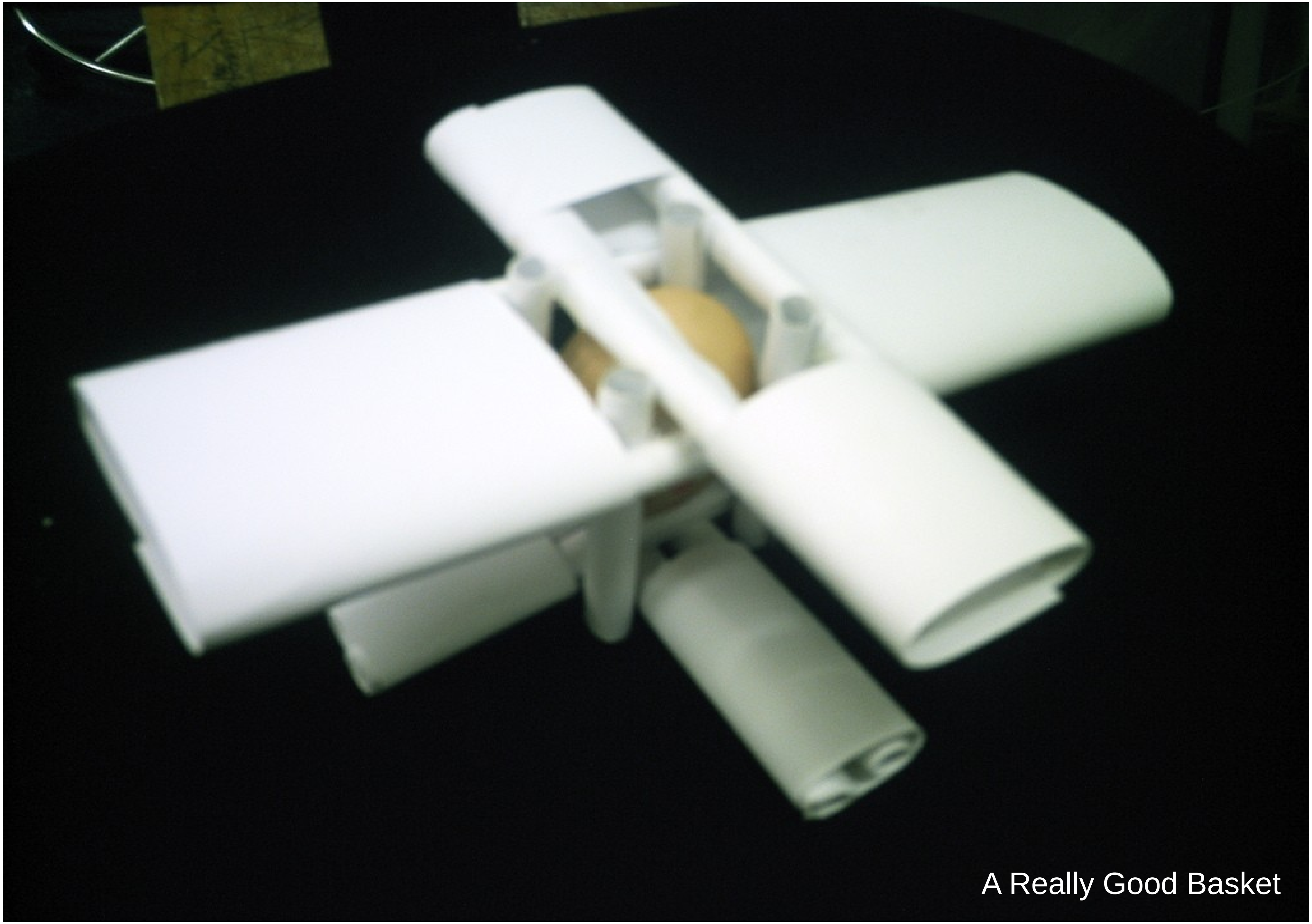
Because you had a better tool.

Collaboration Tools for Fun and Profit

- You know that way that checking your FB activity stream gives you this little hit of dopamine, so soon you find yourself continuously refreshing to see what people you haven't talked to since high school had for lunch?
  - Github is like that ...except that when it's code, it's actually useful...
  - Make code review feel like playing a game.
- Lots of projects and companies are working on this idea, but I believe the current gold standard is github.
  - See me after if you want a tour.

CENTRALIZED VERSION CONTROL

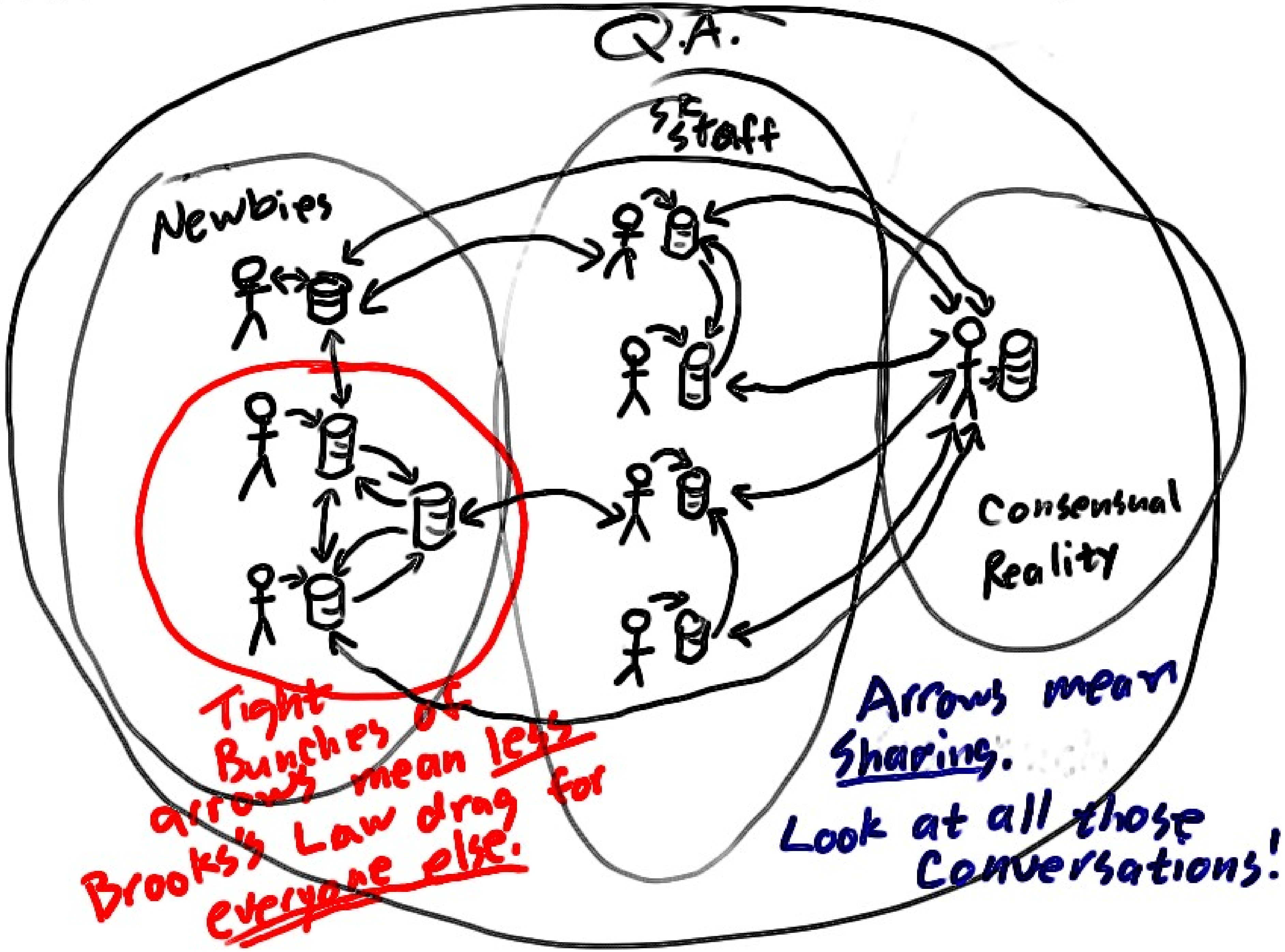


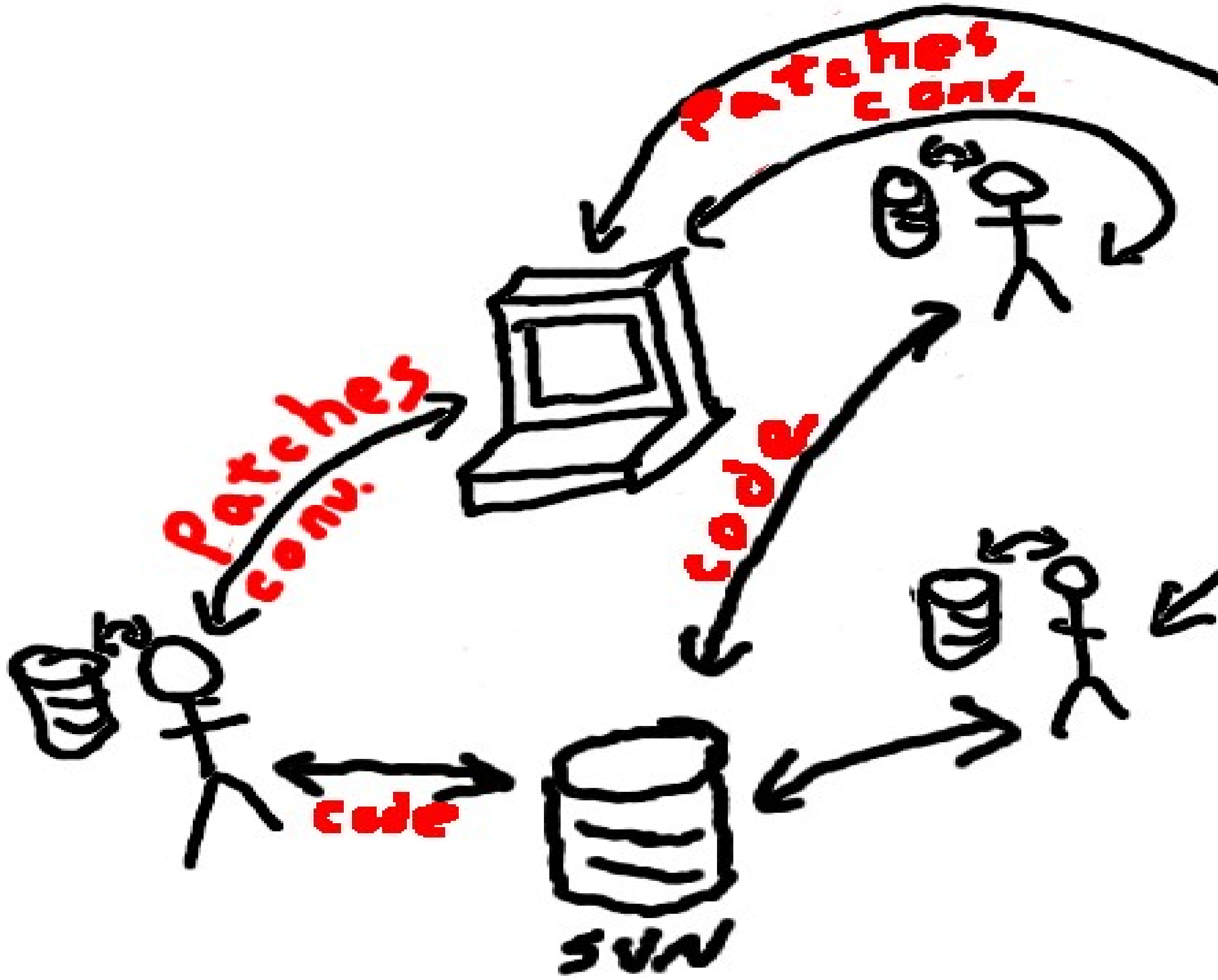

A Really Good Basket

A Quick Introduction to git

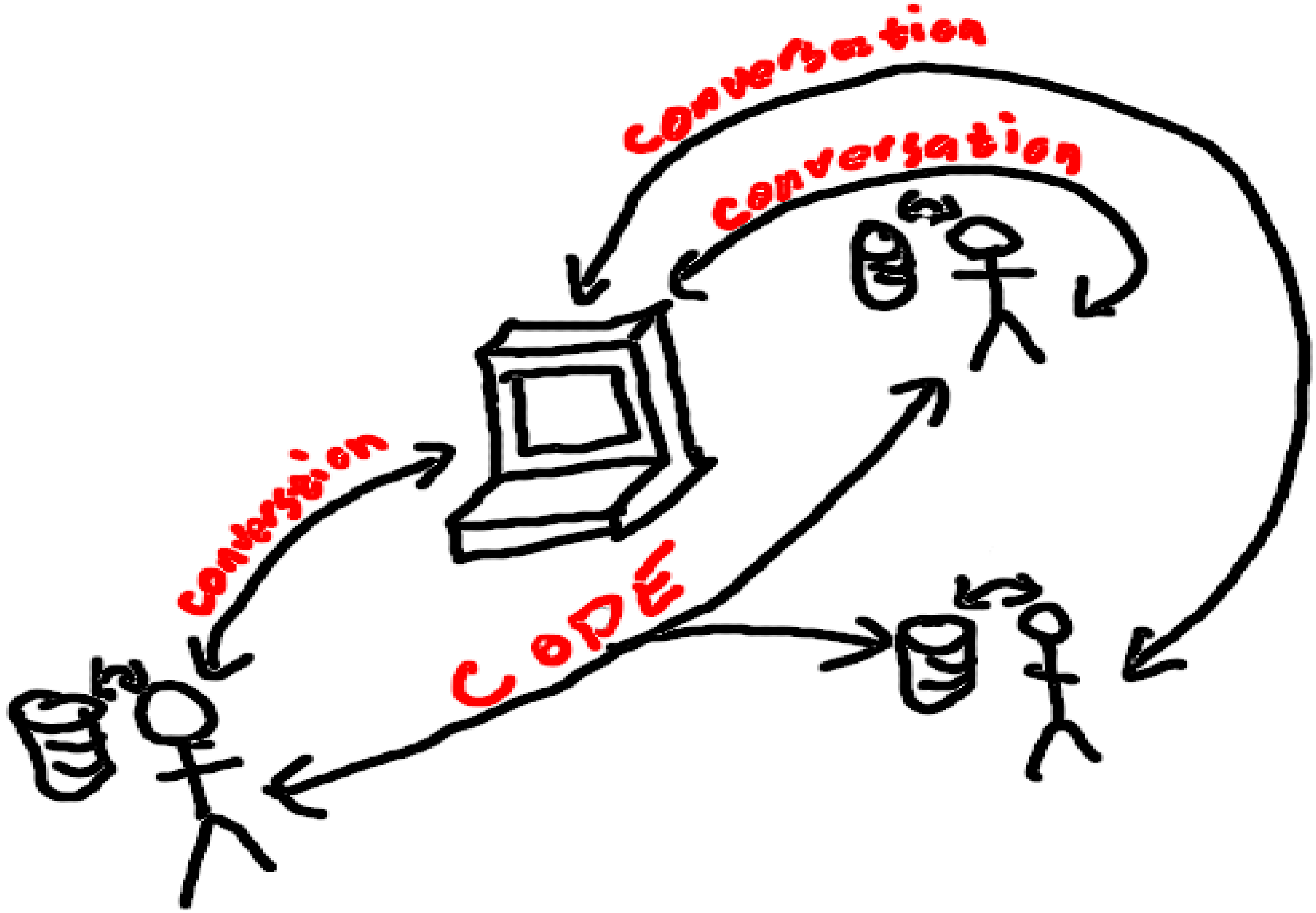
- Fully distributed
- Lockless
- Formally not that different from the darcs stack-of-patches model, but informally it's much easier to use and to reason about.
  - Being easy to reason about is a huge win.
- Developers work with *changesets*, which are sort of like patches but much smarter and generally self-applied.
- Key ideas are: developer convenience is vital, and you should make branches for *everything*.

DISTRIBUTED VERSION CONTROL


Q.A.
Sr Staff
Newbies
Consensual Reality
Tight Bunches of arrows mean less Brooks's Law drag for everyone else.
Arrows mean Sharing.
Look at all those Conversations!



Patches
conv.
Patches
conv.
code
code
SVN


Conversation
Conversation
Conversation
CODE

Better Metadata (the small stuff)

- Pulling a branch in SVN for a one-line fix is ludicrous.
  - So one-line fixes tend to accumulate on the mailing list.
- Pulling a branch in git for a one-line fix is normal.
  - So every one-line fix gets its own commit history.
- Which situation would you rather have?

And Remember Kids:

- Commit Early, Commit Often

Addenda

- http://hginit.com
- http://book.git-scm.com/ (Read Chapters 2-4!)
- http://whygitisbetterthanx.com/
- http://github.com

Thanks

- Travis Brooks, Valkyrie Savage and Evan Stratford for valuable comments
- Matt Bellis for Hg materials
- Anami Sheppard for putting up with me
- Cc-by-SA Image credits:
  - Egg Drop: Kev Meagher
  - Bigraph: Rob Zako
  - Egg Cups: Normann Copenhagen
  - Ice Skate: johnny_automatic & Open Clip Art Library

Bonus Material

- Slides that didn't make the cut

Another Point

ALWAYS Use Version Control

- I shouldn't have to say this.
  - But sad experience tells me someone's going to argue that they know better.
  - Trust me. You don't.
- Nothing – not even your .bashrc – is unworthy of source control.
  - See me after for my rant about source control for sysadmins.

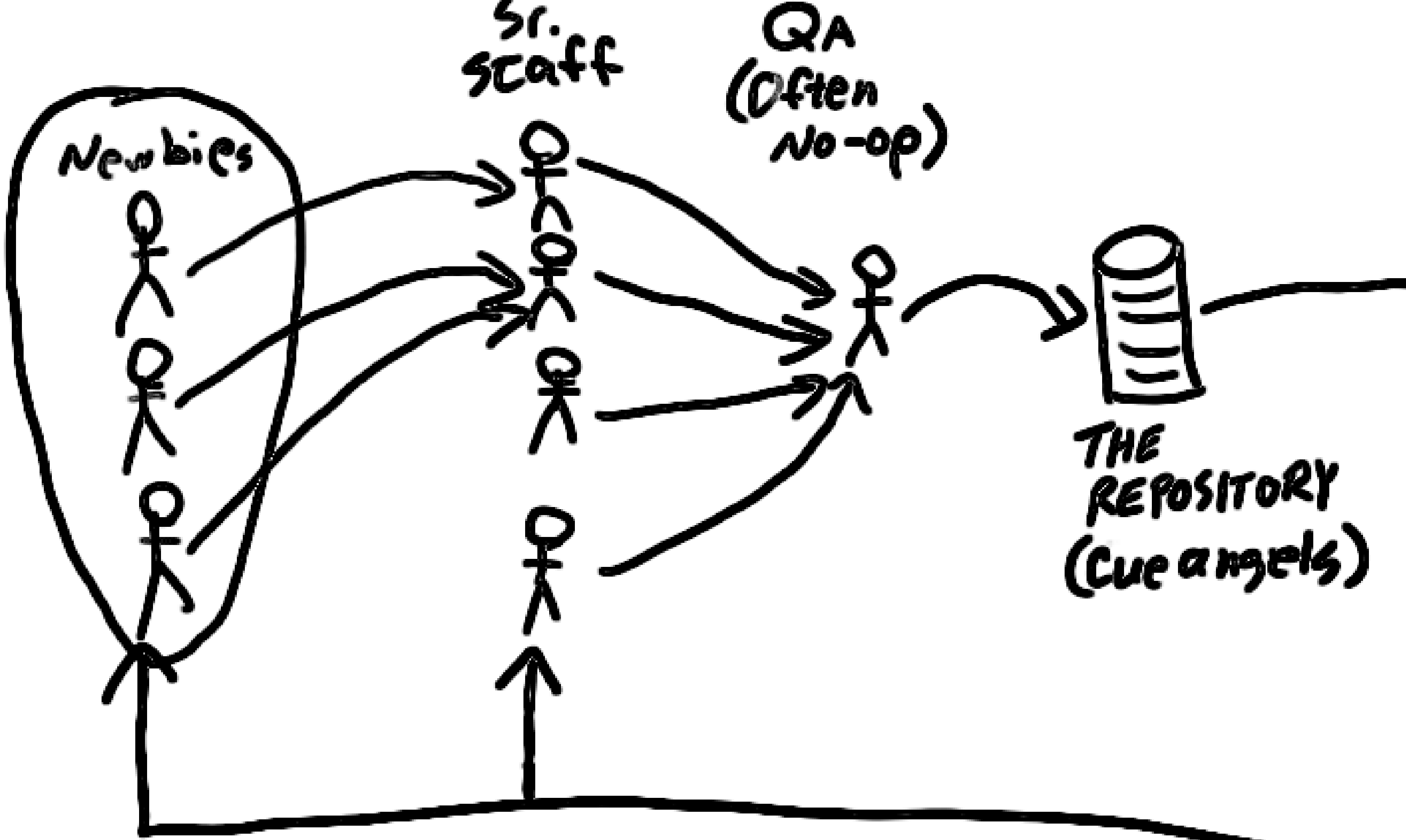
Distributed vs. Centralized Version Control

- CVCS uses a single master repository into which everyone puts their stuff.
  - Interactions mediate via trunk
- DVCS makes every developer their own master – but with great power comes great responsibility.
  - Direct interaction between devs.

CVCS: Organizing Work

- Traditional centralized source control tools suggest hierarchy. Developers work with one *blob of stuff*, so they organize themselves into *trees*.

CENTRALIZED VERSION CONTROL

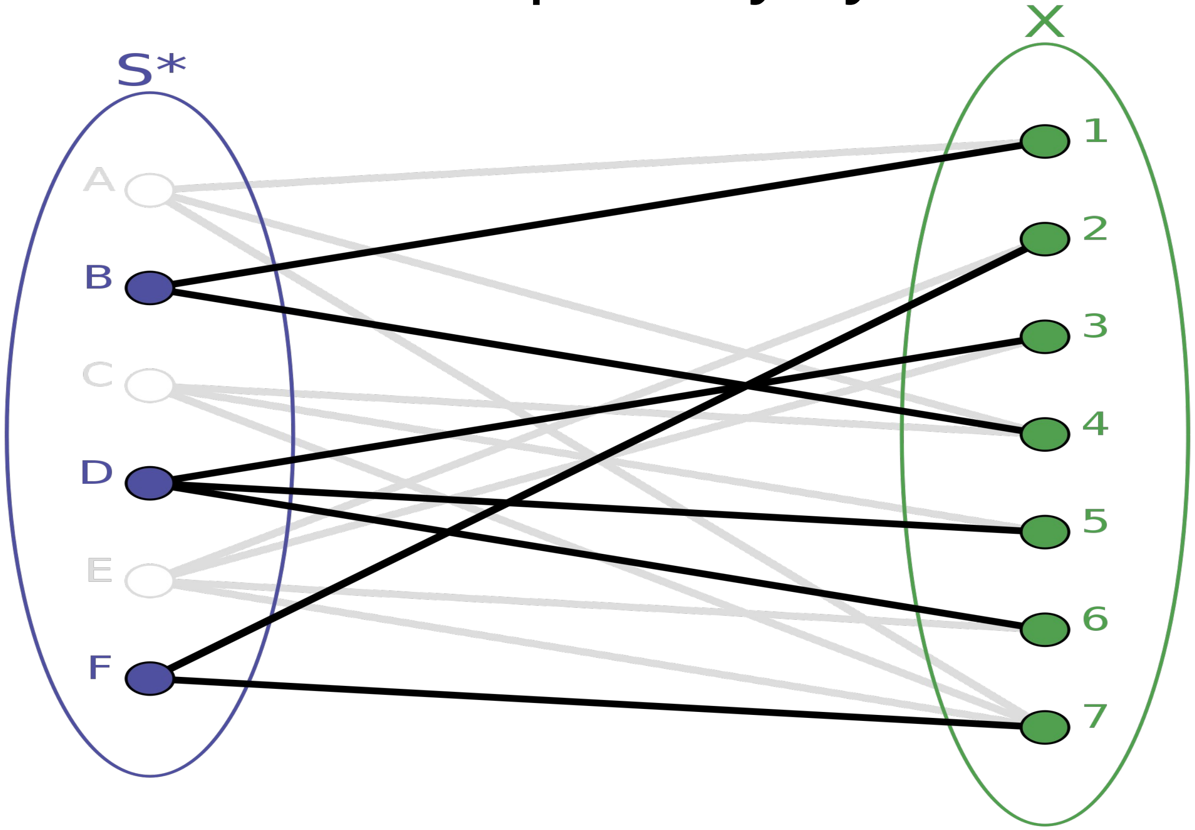


Hierarchical Teams Make Sense with SVN

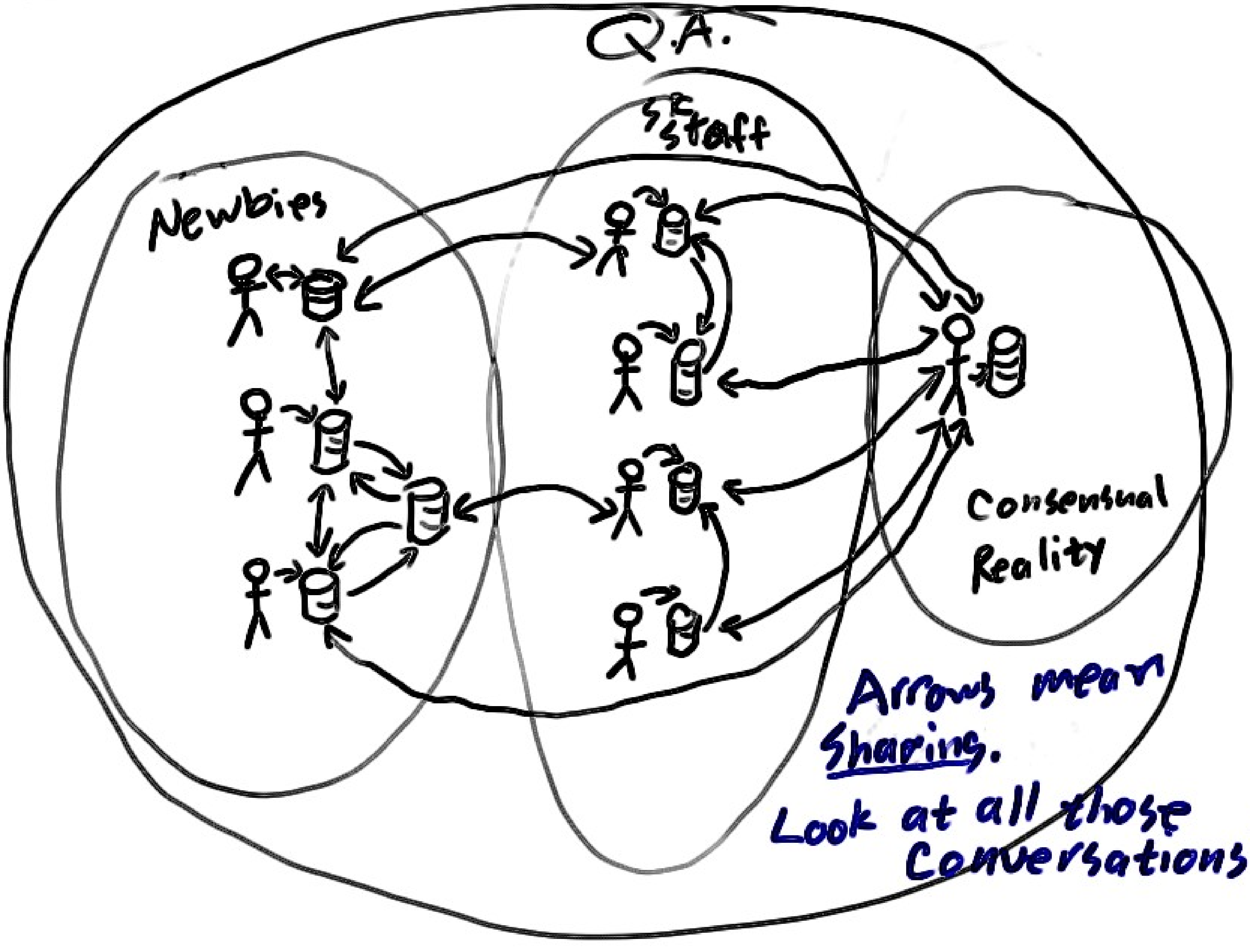
- They have to, because branching is expensive so committing to core is very serious.
- One wrong step and you could “break the build”.
  - (A motto for SVN: If you put all your eggs in one basket, it had better be a really good basket.)

DVCS: Organizing Work

- In DVCS, we make branching and merging cheap. Developers work with *trees* and teams can (self-) organize into any *graph* they want.
  - Tight collaboration works well in a partially cyclic bigraph.



DISTRIBUTED VERSION CONTROL



Arrows mean Sharing.

Look at all those Conversations!

Peer Review With SVN

- I pull from master and make my edits. When I'm done I send patches to my collaborators who read them and argue with me.
- If they want to work with my changes, they pull another copy of master and apply my patches, and then generate patches of their own to send me.

Peer Review With git

- I branch off master and make my changes. I push my branch to a public location and email my collaborators to tell them where.
- My collaborators use my URI to make diffs on-the-fly against any branch they like or a consensual reality (“master”), check out my branch, or *make commits back* to it.
- They still argue with me sometimes of course, but as often they just implement their suggestions in my branch.

Peer Review With SVN

- I pull from master and make my edits. When I'm done I send patches to my collaborators who read them and argue with me.
- If they want to work with my changes, they pull another copy of master and apply my patches, and then generate patches of their own to send me.

Peer Review With git

- I branch off master and make my changes. I push my branch to a public location and email my collaborators to tell them where.
- My collaborators use my URI to make diffs on-the-fly against any branch they like or a consensual reality (“master”), check out my branch, or *make commits back* to it.
- They still argue with me sometimes of course, but as often they just implement their suggestions in my branch.

Wild Experimentation is A-OK

- With cheap branching the cost in terms of complexity management for wild experimentation is kept low.
- So is the cost of version controlling and publishing prototypes.
  - This means it's easier to both share your prototypes or to keep them to yourself – your choice.

Your Process Can Grow as You Do

- Centralized source control encourages careful thought ahead of time about who can commit, to what and when. Once established, the development process (idea to deployment) work hardens.
- With project initialization as simple as 'git init', DVCS systems encourage jumping right in, and deciding on process later. It's technically easy to alter process at any time. (You're on your own politically though.)

Safer? Yes:

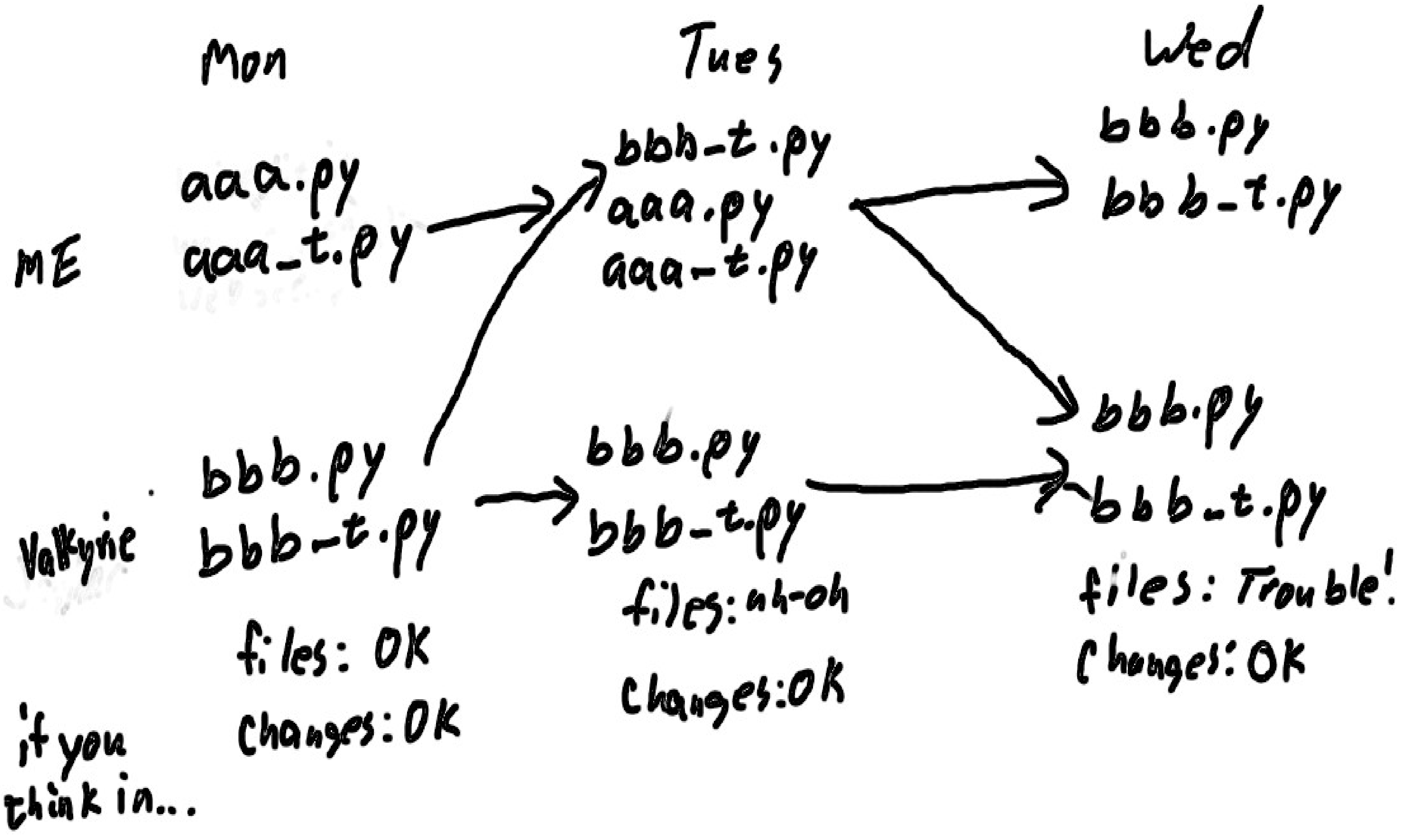
- DVCS makes every working tree a complete backup.
- Reproduction of automation infrastructure (e.g., Jenkins) to developer nodes is easy.
  - Hint: this means you get better code, not just more of it.

Faster? You betcha.

- When branching is cheap you make more branches.
- More branches means you can try more harebrained ideas.
- Trying more harebrained ideas means more wild successes.

Fast Iteration

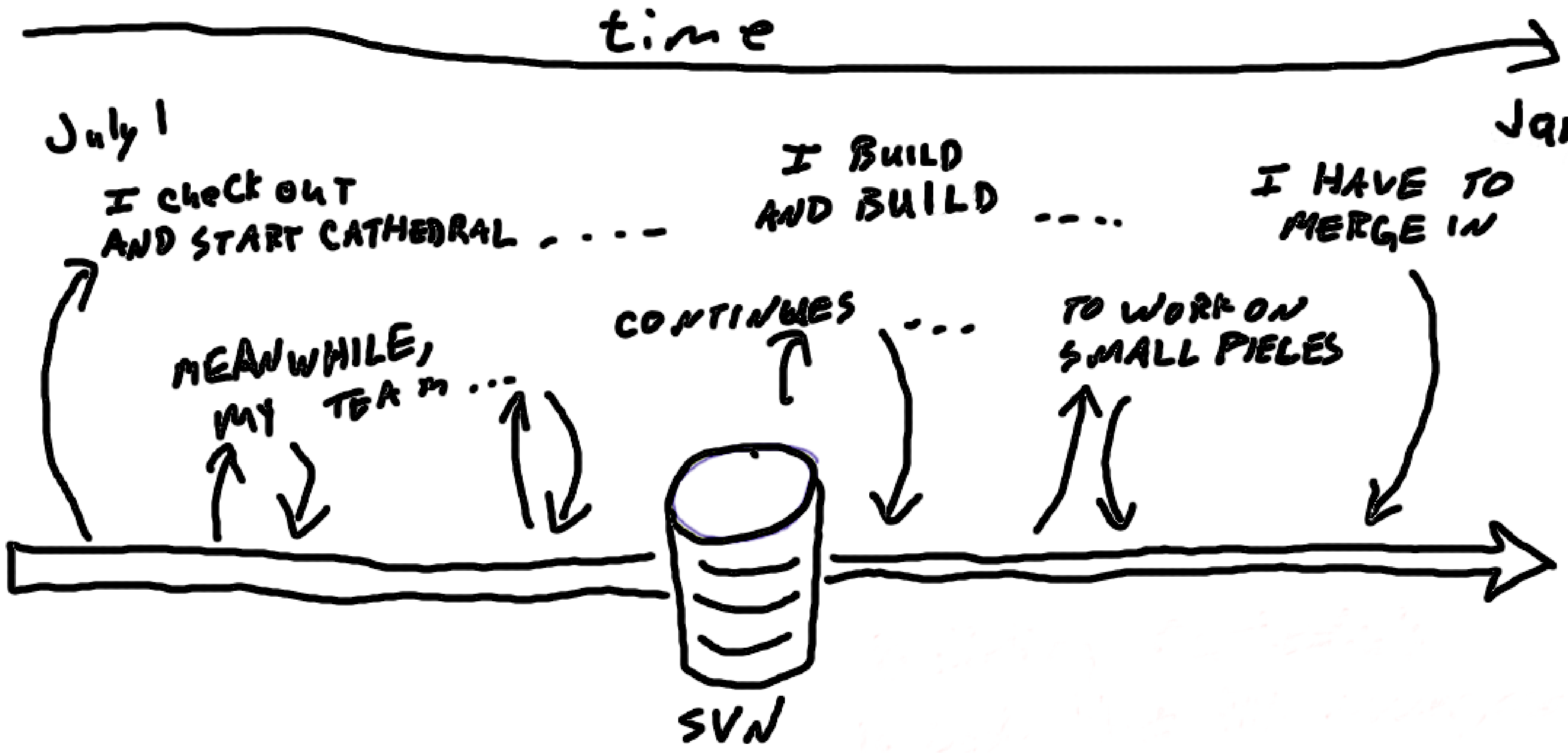
- Tight, iterative development is easy with git: you think in terms of *changes*, not *files*.
- Working in parallel (even on the same files) and merging changes at the end of the day is normal.
  - Compare this to the dread of discovering your coworker made an SVN commit of conflicting changes ten minutes before you.


Mon
Tues
Wed
ME
aaa.py
aaa_t.py
bbb_t.py
aaa.py
aaa_t.py
bbb.py
bbb_t.py
Valkyrie
bbb.py
bbb_t.py
files: OK
changes: OK
bbb.py
bbb_t.py
files: uh-oh
changes: OK
bbb.py
bbb_t.py
files: Trouble!
changes: OK
if you think in...

Infrastructure Support

- Consider Sourceforge or even Trac (VCS, wiki, forum and web integration). Tools integrating with centralized source control tend to be mature, and project-focused.
- Distributed development tools tend to focus on the developer's experience, not the project manager's. Examples include Canonical's launchpad, Google Code and github. (And plugins for Trac, too.)

CATHEDRAL BUILDING AND SOURCE CONTROL
(Never Do This)



Long-lived Branches (SVN)

- I have a branch out for six months and make 10,000 changes across 1,000 files.
- To merge my branch, I have to diff and patch each file carefully as not to clobber other peoples' work.
- And to keep them from cobbering my integration, they shouldn't check anything in while I'm integrating.
- There goes my weekend.

Long-lived Branches (git)

- I have the same 10,000 changes across 1,000 files.
- Let's say I haven't been merging in from everyone else on a regular basis (though normally I would and it would be easy.)
- To merge my branch, I have to examine each and only the individual changes that conflict with changes made in the intervening time.
- What I merge into is just another branch, and so is what you merge into. All streams merge into one, and we can take our time getting there.