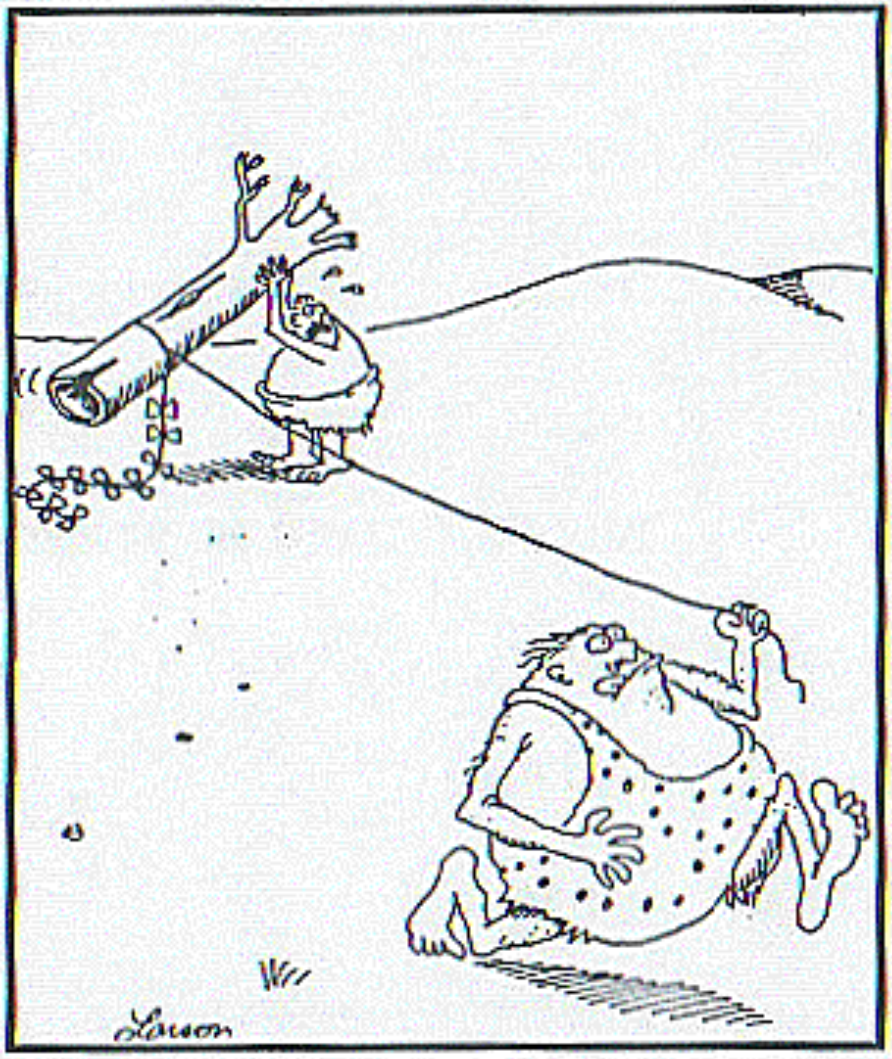
Big things are different from small things



"Okay, Bob! Go! Go!"

The life time of HEP software

Software is a long-term commitment

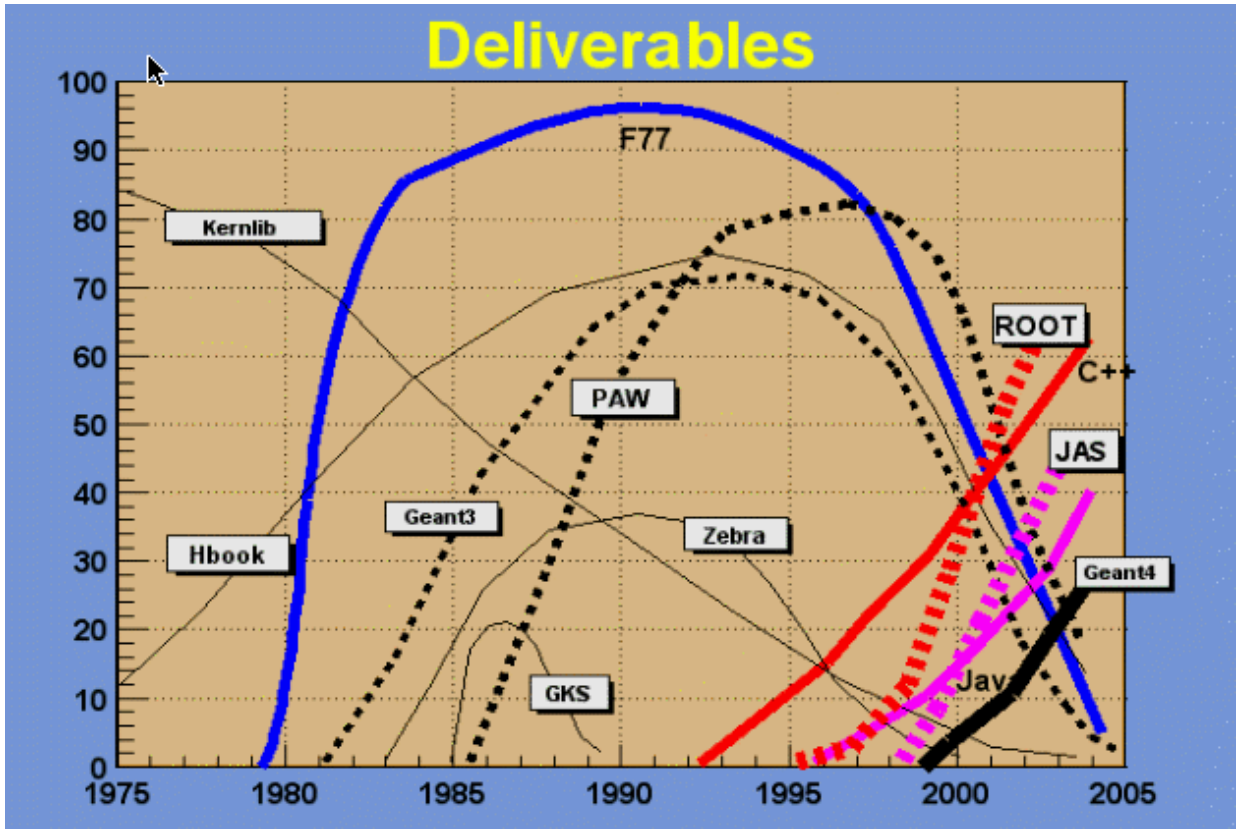
Users like stable and maintained systems

Vote with their feet

It takes time to develop a new system

- Geant3 6+ yrs 3 people 300 KLOCs
- PAW 6+ yrs 5 people 300
- Zebra 4+ yrs 2 people 100
- ROOT 5* yrs 3 people 630

Real work is after that !!



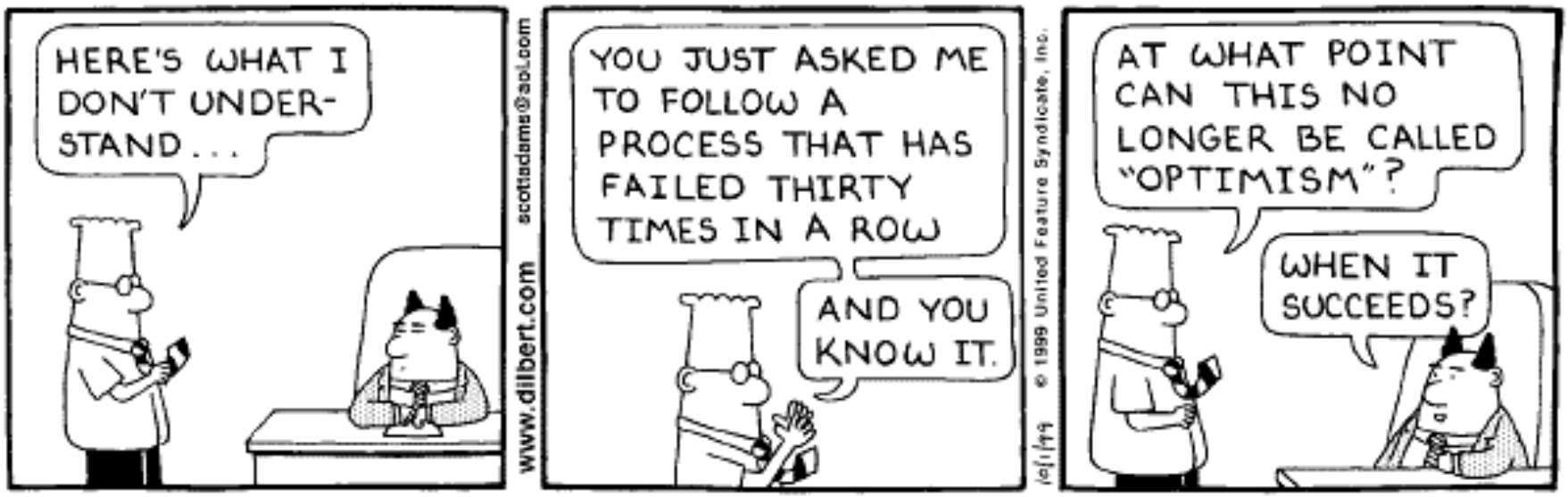
Many releases of the software are needed over its lifetime to fix bugs, add new features, support new platforms etc

How do we cope?

We try to find a way of working that leads to success

- We create a "process" for building systems
- We devise methods of communicating and record keeping: "models"
- We use the best tools & methods we can lay our hands on

And we engage in denial:



Can't technology save us?

CERN School of Computing

More

We've built a series of ever-larger tools to handle large code projects:

- **CVS, SVN, Git for controlling and versioning code**
- **Tools for building "releases" of systems**
- **Tools for "configuration management"**

But we struggle against three forces:

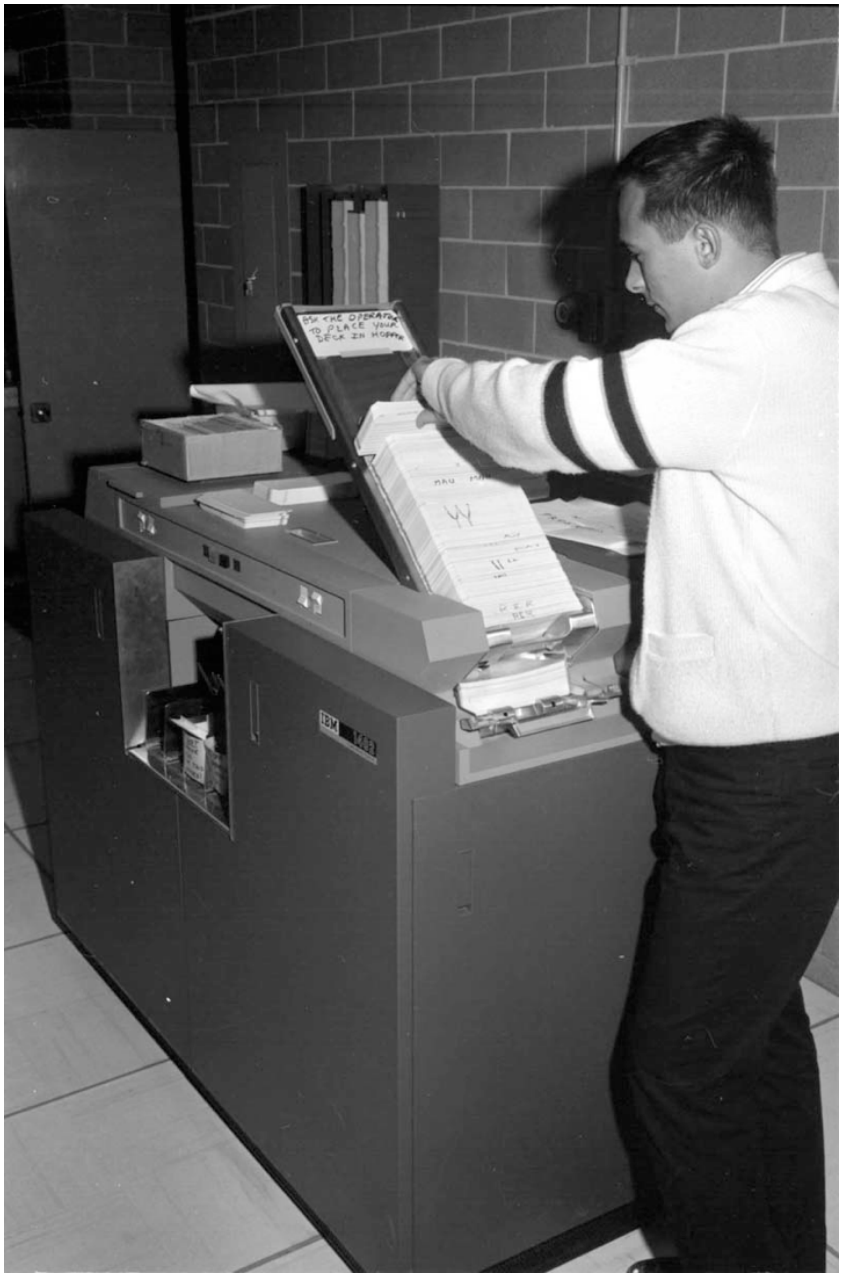
- **We're always building bigger & more difficult systems**
- **We're always building bigger & more difficult collaborations**
- **And we're the same old people**

Net effect: We're always pushing the boundary of what we can do

Stupidity got us into this mess; why can't it get us out? - Will Rogers

How we got here:

First, you just wrote a big program



CERN School of Computing

How we got here:

First, you just wrote a big program

But soon it was so big you wanted help

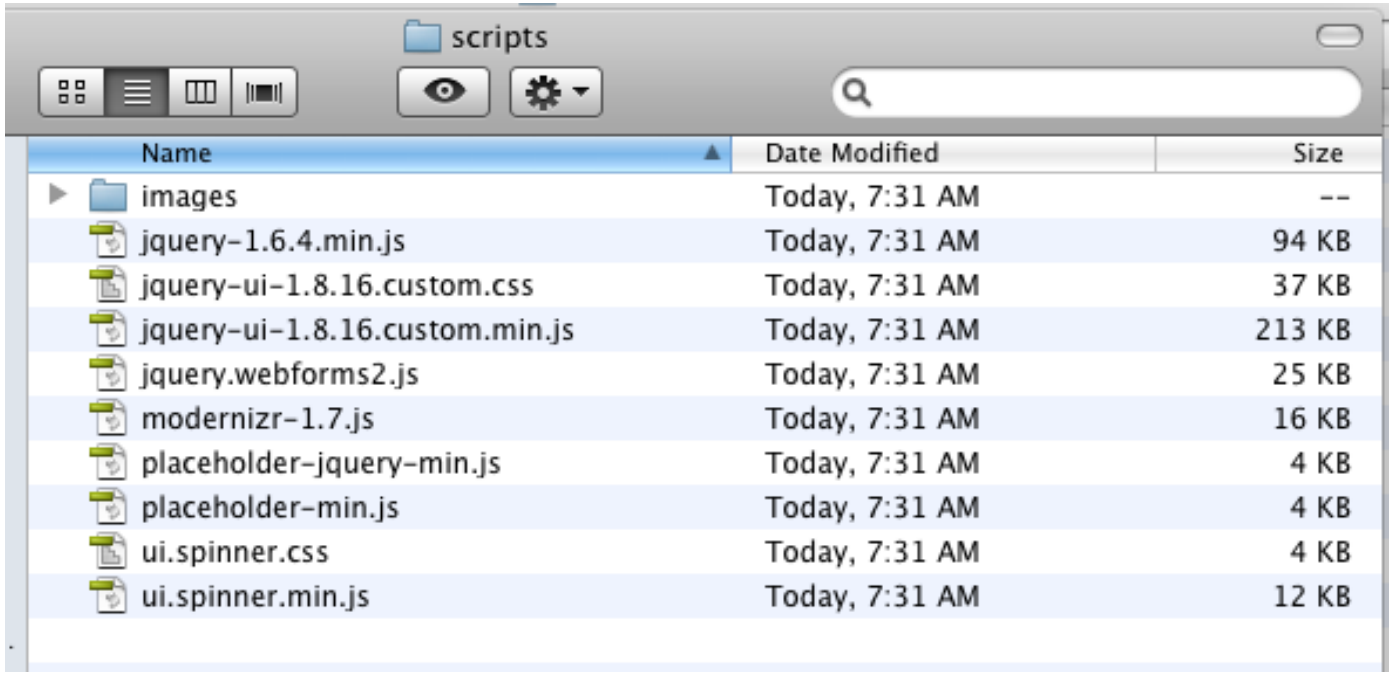
CERN School of Computing

How we got here:

First, you just wrote a big program

But soon it was so big you wanted help

So you broke it into pieces/files/modules



CERN School of Computing

How we got here:

First, you just wrote a big program

But soon it was so big you wanted help

So you broke it into pieces/files/modules

But how do you share work on those?

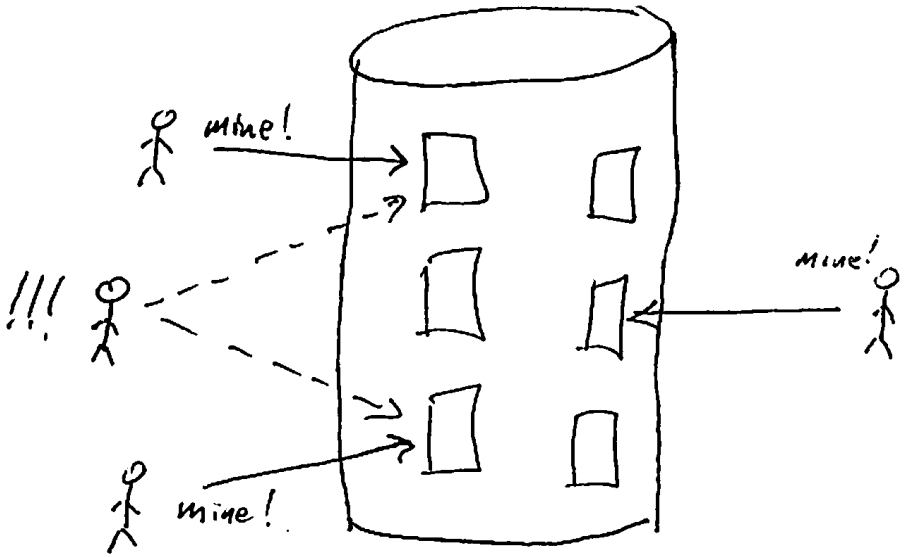
How we got here:

First, you just wrote a big program

But soon it was so big you wanted help

So you broke it into pieces/files/modules

But how do you share work on those?



Revision Control System (RCS)

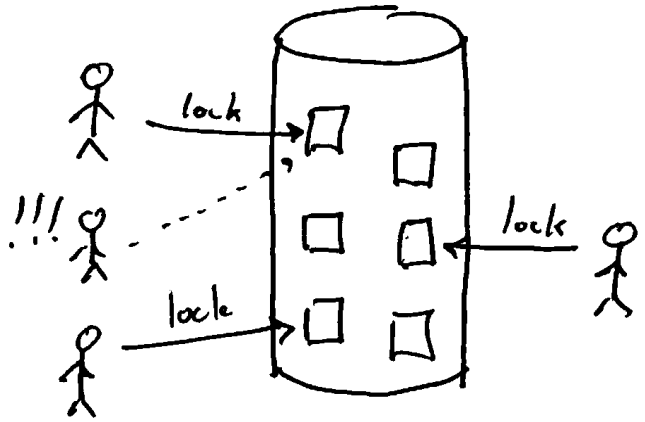
Maintains a repository of text files

- Allows users to check-out, edit, check-in changed text
- Old code remains available

  Each checked-in change defines a new revision

  You can retrieve, ask for differences with any of them
- Revisions can be tagged for easy reference

Anybody can get a specific set of source code file versions



Revision Control System (RCS)

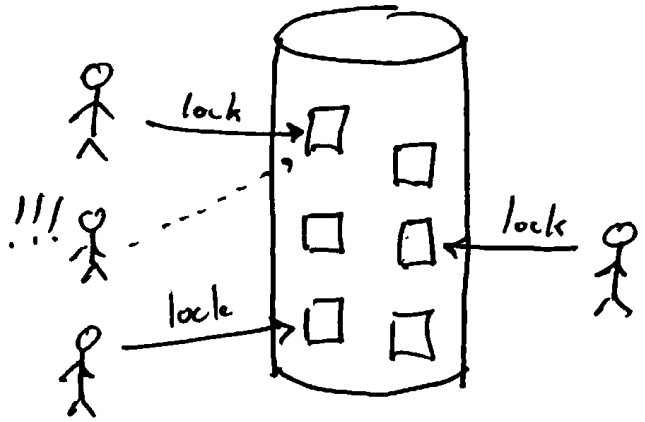
Maintains a repository of text files

- Allows users to check-out, edit, check-in changed text
- Old code remains available

  Each checked-in change defines a new revision

  You can retrieve, ask for differences with any of them
- Revisions can be tagged for easy reference

Anybody can get a specific set of source code file versions



But only one person working on a file at a time!

Problem: This serializes development

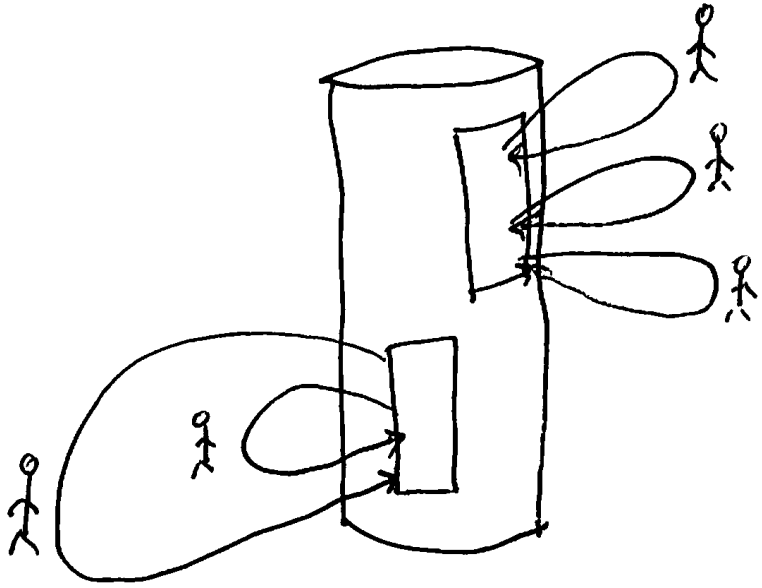
Workarounds, but with problems of their own

Concurrent Versions System (CVS)

More

As systems & collaborations grow, efficiency goes down

"Version" idea: Track changes from one version to next



Big advantage: checkout is not exclusive

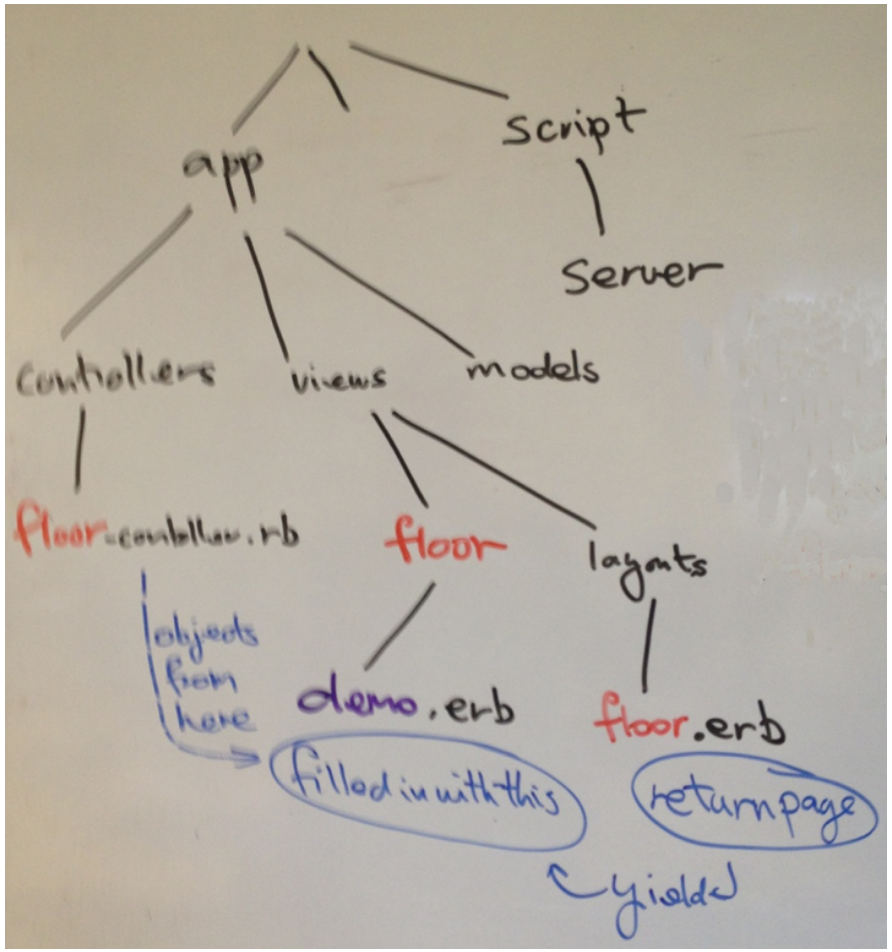
- More than one developer can have the same file checked out
- Developers can control their own use of the code for read, write
- Changes can come from multiple sources
- Tool handles (most) of the conflict resolution

And systems still grow

You broke the code into pieces/files/modules

And things got more and more complicated

You needed an organization above the level of the file



Directory Tree

```
.
|-- code
|  |-- perl
|  |  |-- clean_fasta.pl
|  |  `-- run_glam2.pl
|  |-- R
|  |  |-- gse21350.R
|  |  |-- gse7275.R
|  |  `-- gse9589.R
|  |-- ruby
|  |  `-- seq2svm.rb
|  `-- sh
|      `-- ens_uniq.sh
|-- data
```

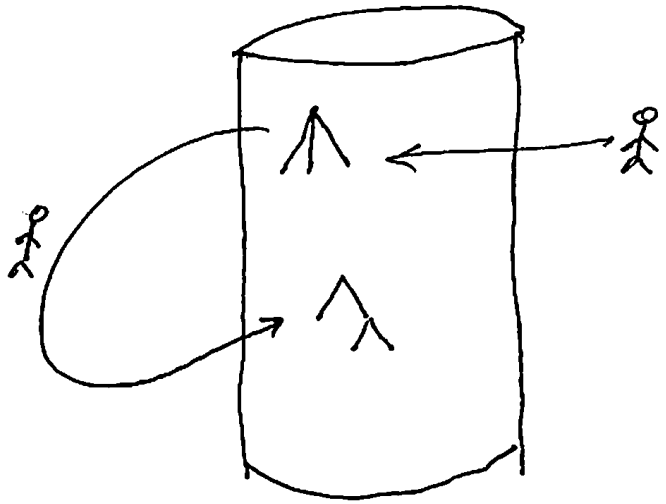
Subversion (svn)

So you broke it into pieces/files/modules

And things got more and more complicated

You needed an organization above the level of the file

Want to be able to collaborate on that:



Subversion (svn) brings tools for doing that

Why isn't that enough?

CVS, SVN lets me "check out" complete source code. Then just compile!

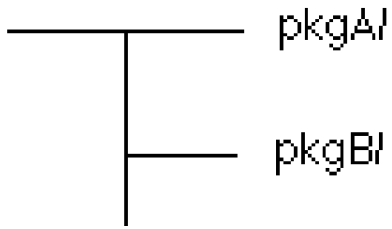
- Works great for small projects
- But builds run into several levels of scaling problems

1) **Want to attach to external code**

- We don't write everything (though tempted)
- Sometimes don't get source for external code
- Need some way to connect to specific external libraries:
  Both specific product, and a specific version of that product

2) **Want to separate code into multiple parts**

- So people/institutions can take responsibility for parts
- But software has cross-connections
- Need structure that works for both

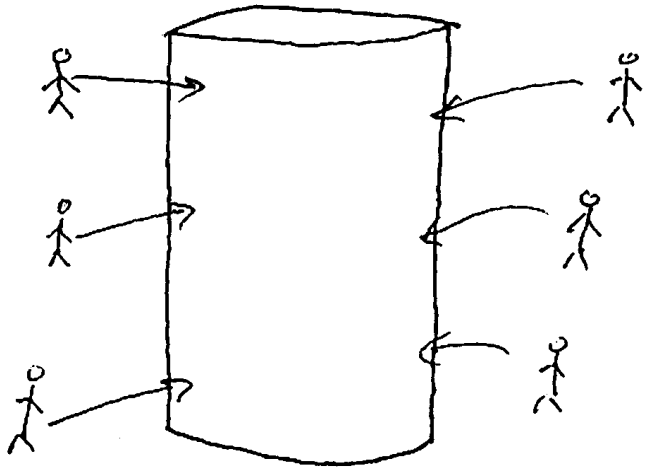


And still need to be able to build the code

Scaling is still an issue

Everybody is sharing a single repository

Every commit is immediately visible to everybody else

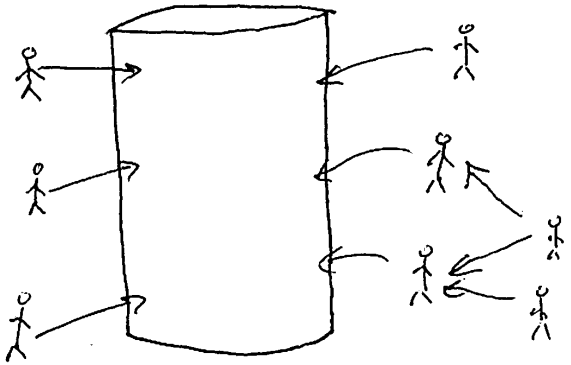


Development stands on shifting sand

- **Detailed records, but little understanding**

Workarounds!

- **Package Coordinators**
- **Tags and Branches**
- **External record keeping tools**



Scaling: Handling complicated builds

CERN School of Computing

More

Multiple "packages" require cross connects while compiling

- Typing the compile command gets boring fast

```
g++ -c -I"/afs/cern.ch/user/s/scherzer/public/1001/InstallArea/include/PixelDigitization"
-I"/afs/cern.ch/user/s/scherzer/public/1001/InstallArea/include/SiDigitization"
-I"/afs/cern.ch/atlas/software/dist/10.0.1/InstallArea/include/InDetSimEvent"
-I"/afs/cern.ch/atlas/software/dist/10.0.1/InstallArea/include/HitManagement"
-I"/afs/cern.ch/atlas/software/dist/10.0.1/InstallArea/include/TestTools"
-I"/afs/cern.ch/atlas/software/dist/10.0.1/InstallArea/include/TestPolicy"
-I"/afs/cern.ch/atlas/offline/external/Gaudi/0.14.6.14-pool201/GaudiKernel/v15r7p4"
-I"/afs/cern.ch/sw/lcg/external/clhep/1.8.2.1-atlas/slc3_ia32_gcc323/include"
-I"/afs/cern.ch/sw/lcg/external/Boost/1.31.0/slc3_ia32_gcc323/include/boost-1_31"
-I"/afs/cern.ch/sw/lcg/external/cernlib/2003/slc3_ia32_gcc323/include"  -O2 -pthread
-D_GNU_SOURCE -pthread -pipe -ansi -pedantic -W -Wall -Wwrite-strings -Woverloaded-virtual
-Wno-long-long -fPIC -march=pentium -mcpu=pentium -pedantic-errors -ftemplate-depth-25
-ftemplate-depth-99  -DHAVE_ITERATOR -DHAVE_NEW_IOSTREAMS  -D_GNU_SOURCE
-o PixelDigitization.o  -DEFL_DEBUG=0 -DHAVE_PRETTY_FUNCTION -DHAVE_LONG_LONG
-DHAVE_BOOL -DHAVE_EXPLICIT -DHAVE_MUTABLE -DHAVE_SIGNED -DHAVE_TYPENAME
-DHAVE_NEW_STYLE_CASTS -DHAVE_DYNAMIC_CAST -DHAVE_TYPEID
-DHAVE_ANSI_TEMPLATE_INSTANTIATION -DHAVE_CXX_STDC_HEADERS '
-DPACKAGE_VERSION="PixelDigitization-00-05-16"'  -DNDEBUG  -DCLHEP_MAX_MIN_DEFINED
-DCLHEP_ABS_DEFINED -DCLHEP_SQR_DEFINED     ../src/PixelDigitization.cxx
```

Build tools: "make", "Ant", etc

- Manually create a "makefile" that forwards include options to the compiler

  g++ -IpkgA -IpkgB

- Lets you adapt to various internal structures

  g++ -IpkgA -IpkgB/include -IpkgC/headers

- Also lets you add other options to control localization, debugging, etc

Size keeps getting in the way

Small experiment (offline production code only):

- 430 packages
- 17,000 files
- 7 million lines of source

Some of these are large "for historical reasons"

But that's true of just about any project

Repository checkout: 13 minutes

Build from scratch: 6 hours

Spread across multiple production machines; never did complete on laptop

"gmake" with one change: about 4-12 minutes to think about dependencies

And I don't even want to think about the size of a monolithic Makefile

And everybody will need multiple copies…

Old ones, new ones, …

"But I just want to run the program!"

Issue arises at large & small level

At the level of developers, need way to manage this

- Both tools and procedures
  - We'll be discussing & exercising typical tools; many exist!
  - Individual collaborations have their own ways of sharing info

At the collaboration level, need procedures to ensure it works

- "Nightly builds"
  - Now common in HEP - Give early feedback on consistency problems
  - Many in industry moving toward "continuous integration"
- Not a complete solution by itself
  - Only works when people actually integrate early and often
- Reduces problems, but integration is still a lot of work

More

When Boeing wanted to design the 747, they had two choices:

1. Hire "SuperEngineer", who could do it alone
2. Hire 7,200 engineers and organize them to cooperate

Which did they choose?

Why?

What can we learn from this?

Two Approaches: (1) Organize people to match the work

1) You want a "release system"

Use tags in the repository to mark "package versions"

"Package Coordinators" are people with the local knowledge

Build tools that record relations between packages, external requirements:

- pull out proper consistent versions,
- combine make files,
- control the build

Complicated tools that need to know a bunch of stuff

Configuration Management Tool (CMT)

- Based on 'requirements file' with custom syntax and contests

Lots of others (SCRAM, ETICS, cloud-based tools)

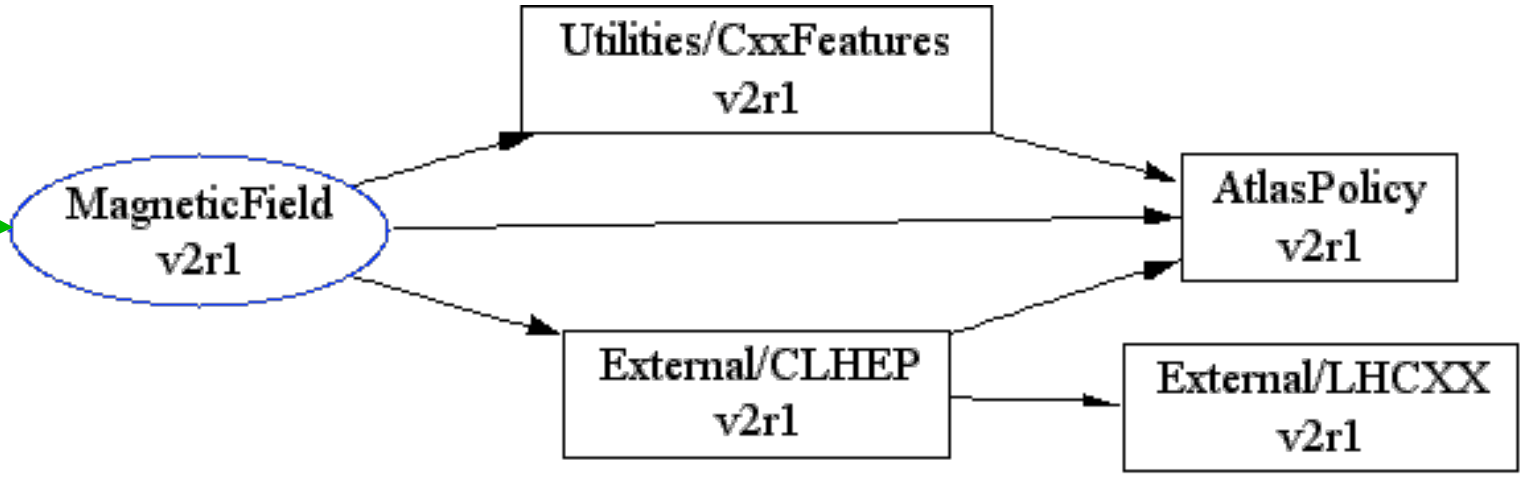
- Showing CMT because easy to see how it works

More

CMT: a simple release and consistency tool

Requirements file provides custom language for expressing our needs

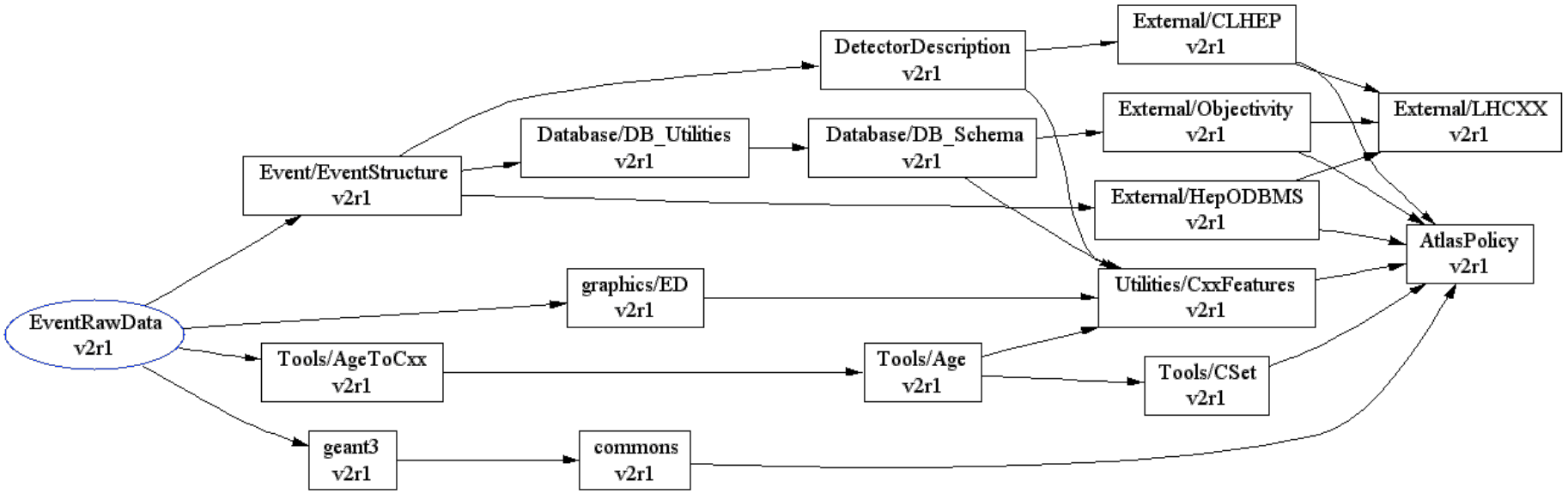
More




```
package MagneticField

author  Laurent Chevalier <laurent@hep.saclay.cea.fr>
author  Marc Virchaux <virchau@hep.saclay.cea.fr>

use AtlasPolicy v2r1
use CxxFeatures v2r1 Utilities
use CLHEP v2r1 External

include_dirs $(MAGNETICFIELDROOT)/MagneticField

branches MagneticField doc src test
...
```

Example from C. Arnault (LAL and Atlas)

"Consistency" scales poorly

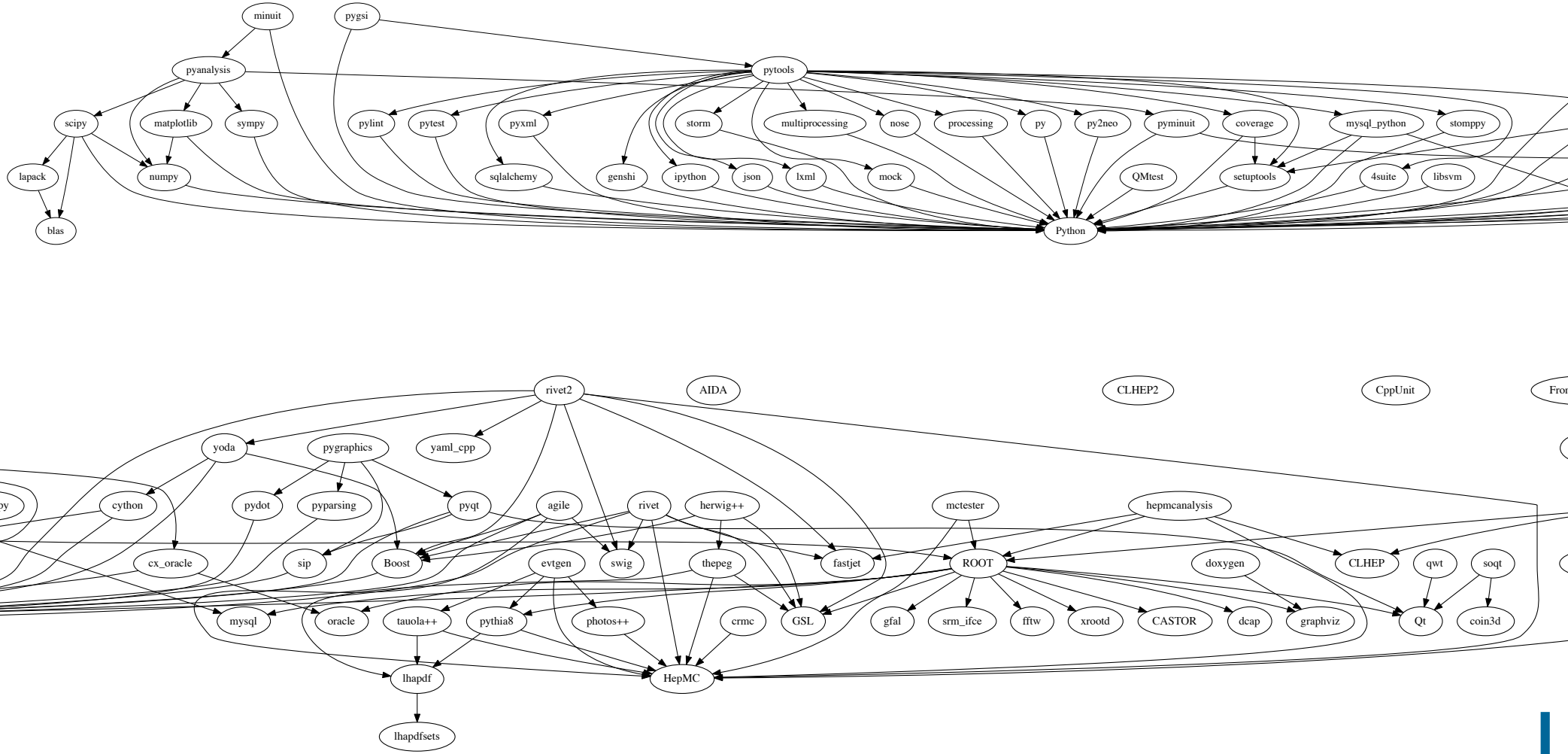


Software strongly depends on other software

- Usually managed at the package level

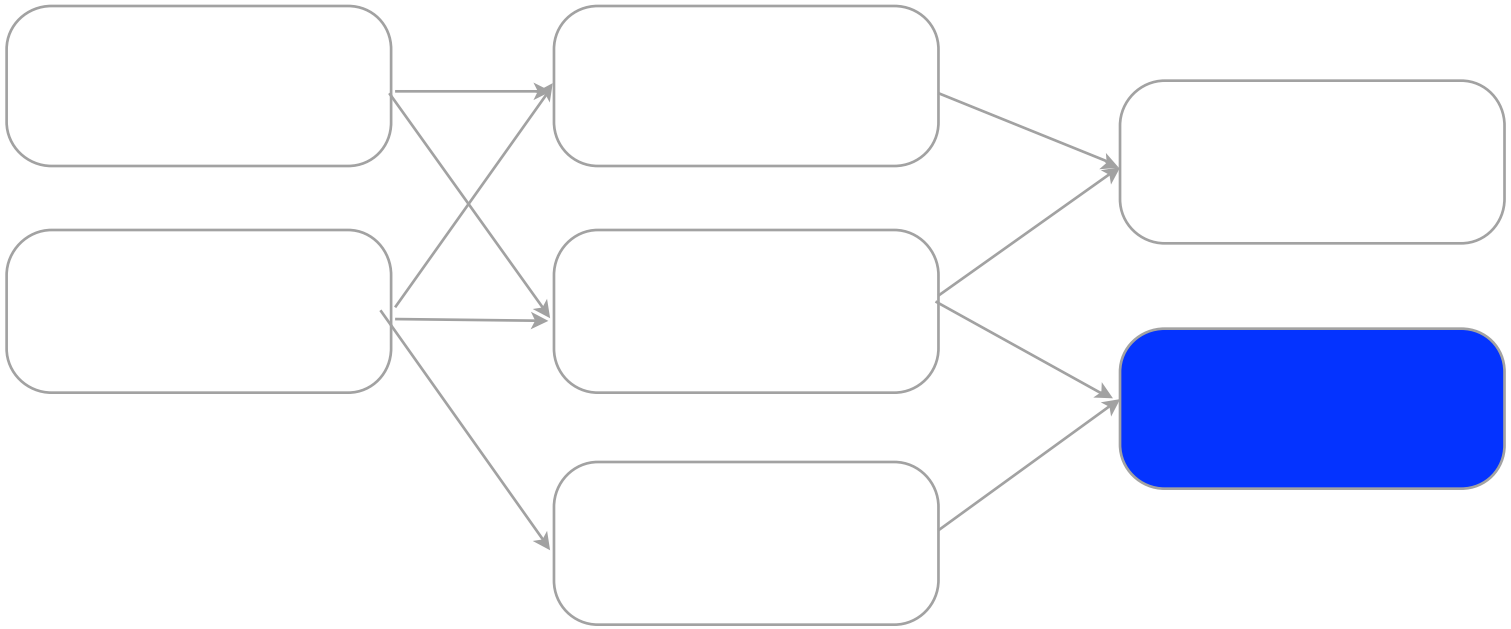
  (This can result in lots of packages, as you subdivide over and over)

- Expresses how changes in one piece can drive changes in another

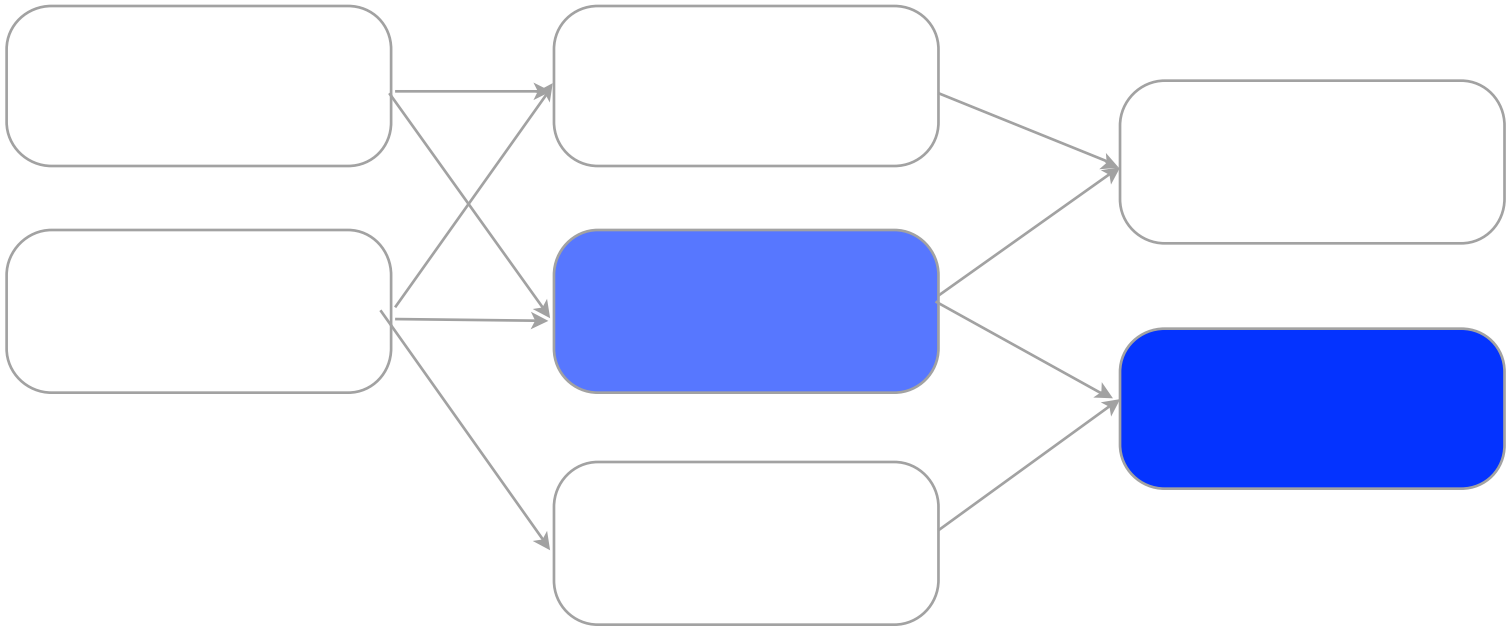

minuit
pygsi
pyanalysis
pytools
scipy
matplotlib
sympy
pylint
pytest
pyxml
storm
multiprocessing
nose
processing
py
py2neo
pyminuit
coverage
mysql_python
stomppy
lapack
numpy
sqlalchemy
genshi
ipython
json
lxml
mock
QMtest
setuptools
4suite
libsvm
blas
Python
rivet2
AIDA
CLHEP2
CppUnit
yoda
pygraphics
yaml_cpp
cython
pydot
pyparsing
pyqt
agile
rivet
herwig++
mctester
hepmcanalysis
cx_oracle
sip
Boost
evtgen
swig
thepeg
fastjet
ROOT
doxygen
CLHEP
qwt
soqt
mysql
oracle
tauola++
pythia8
photos++
crmc
GSL
gfal
srm_ifce
fftw
xrootd
CASTOR
dcap
graphviz
Qt
coin3d
lhapdf
HepMC
lhapdfsets

More

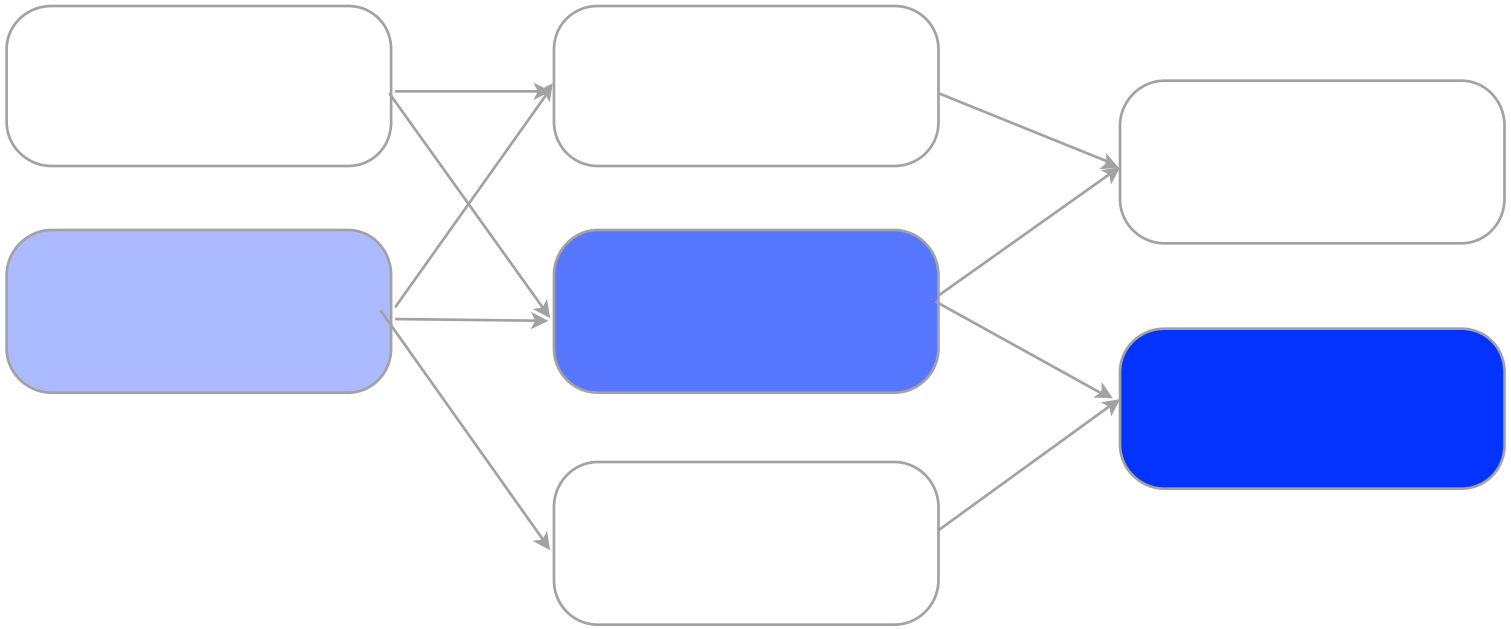
How change propagates through dependencies



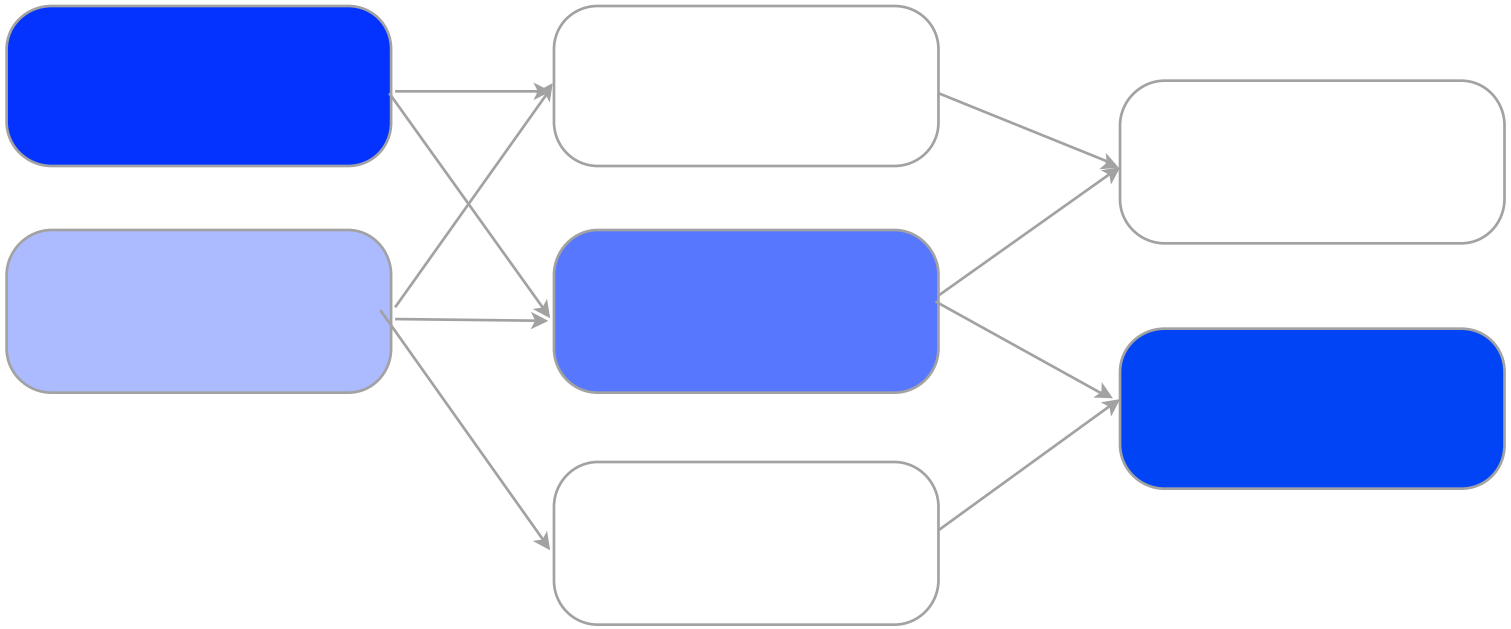
How change propagates through dependencies



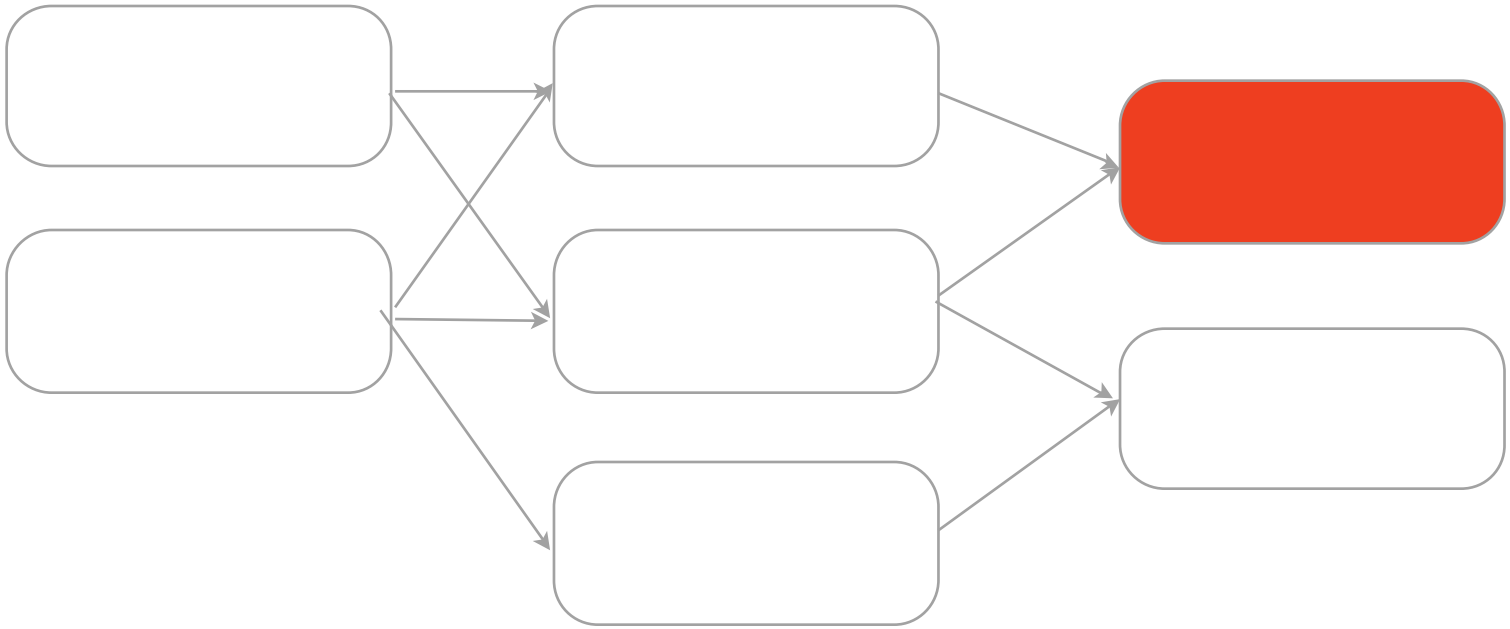
How change propagates through dependencies

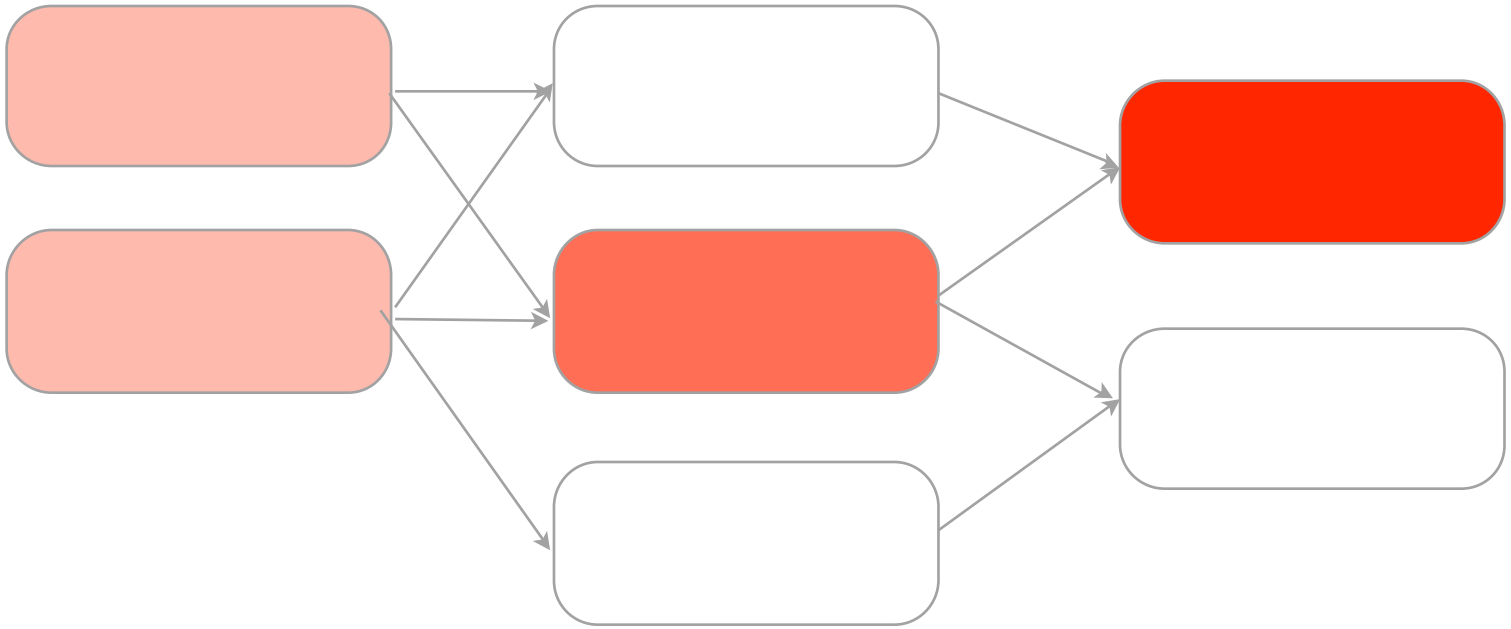


Changes don't always stay small

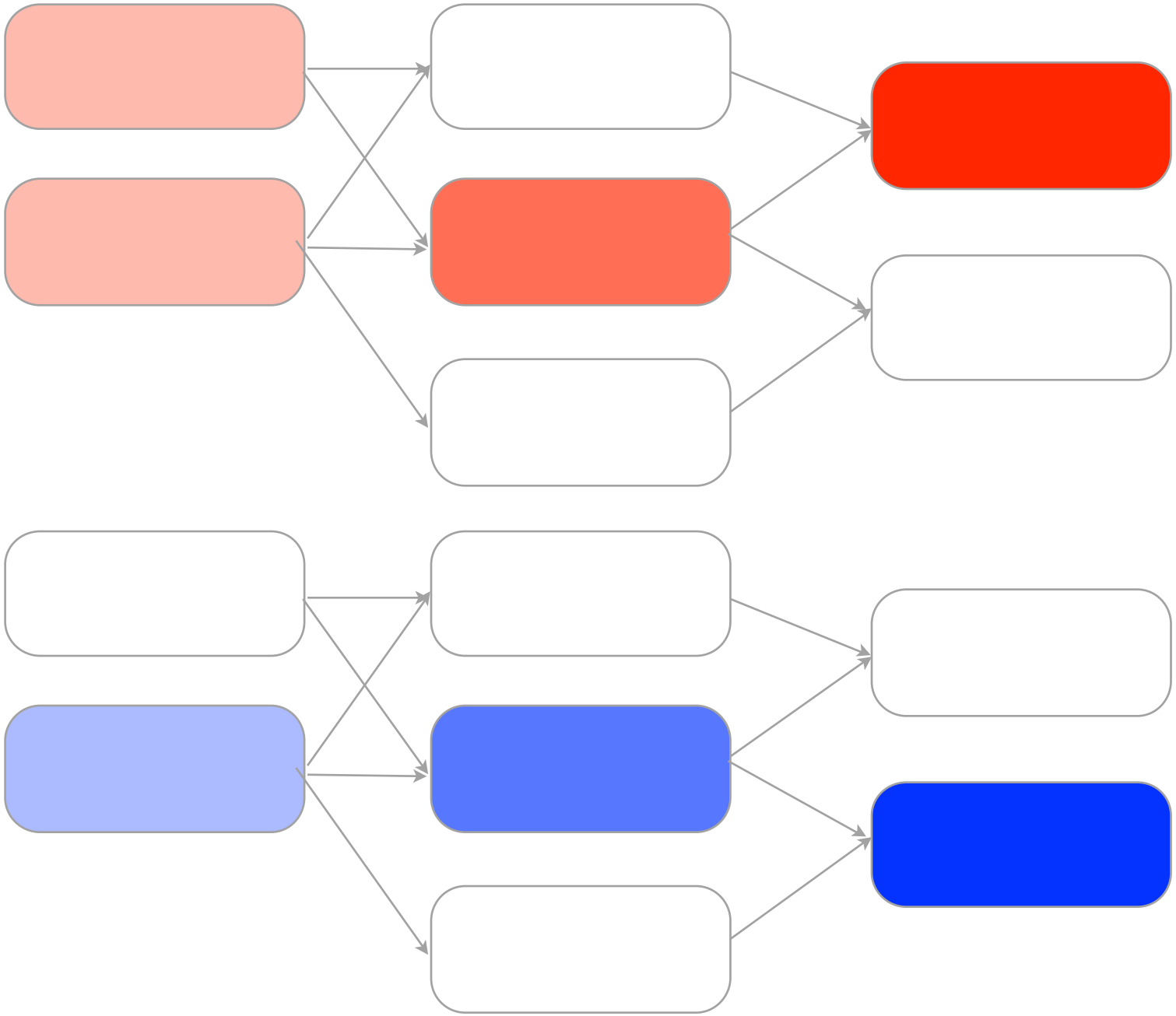


Another change:

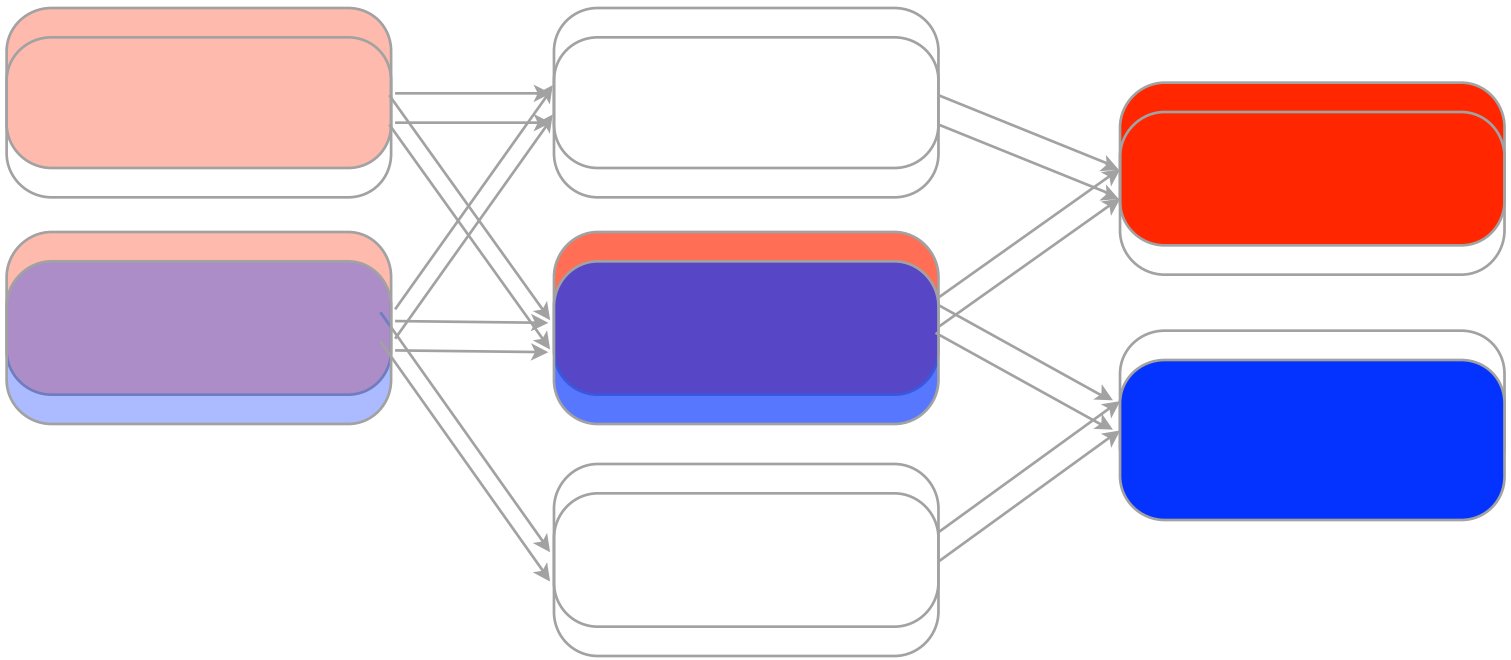




Change management requires people



Change management requires people



2nd approach: People handle consistency, machines build

The repository is where the code should be consistent

Create tools that let you do that!

Allow developers to work on their own content until it's right

- Not every change should go to everybody

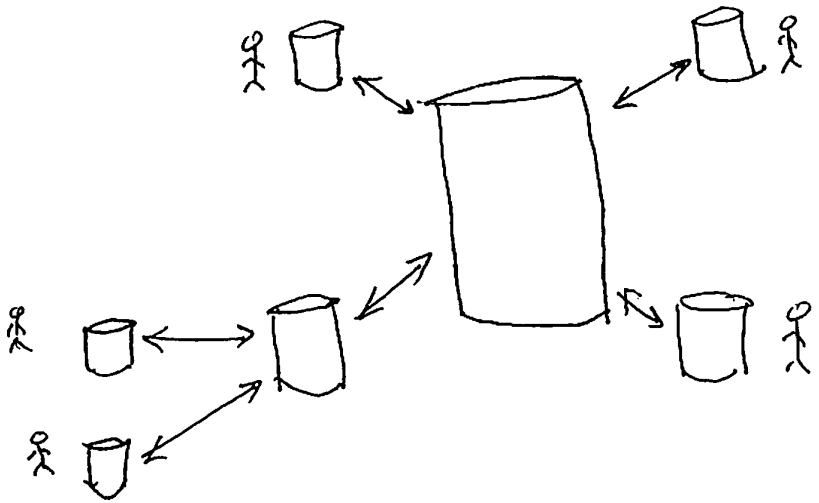
Allow developers to collaborate in small groups

- Put together a sub-system

It's not the versions that are interesting, it's the updates!

- Development becomes a story instead of a series of snapshots
- "Here's our complete contribution"

Enter Git (not an acronym)



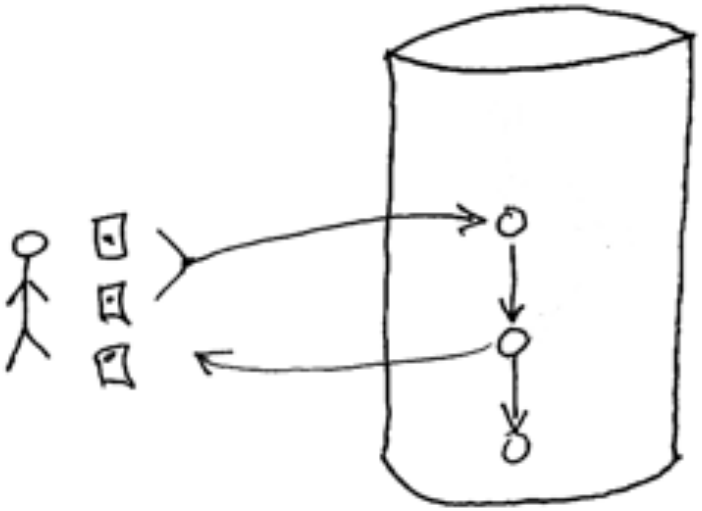
More

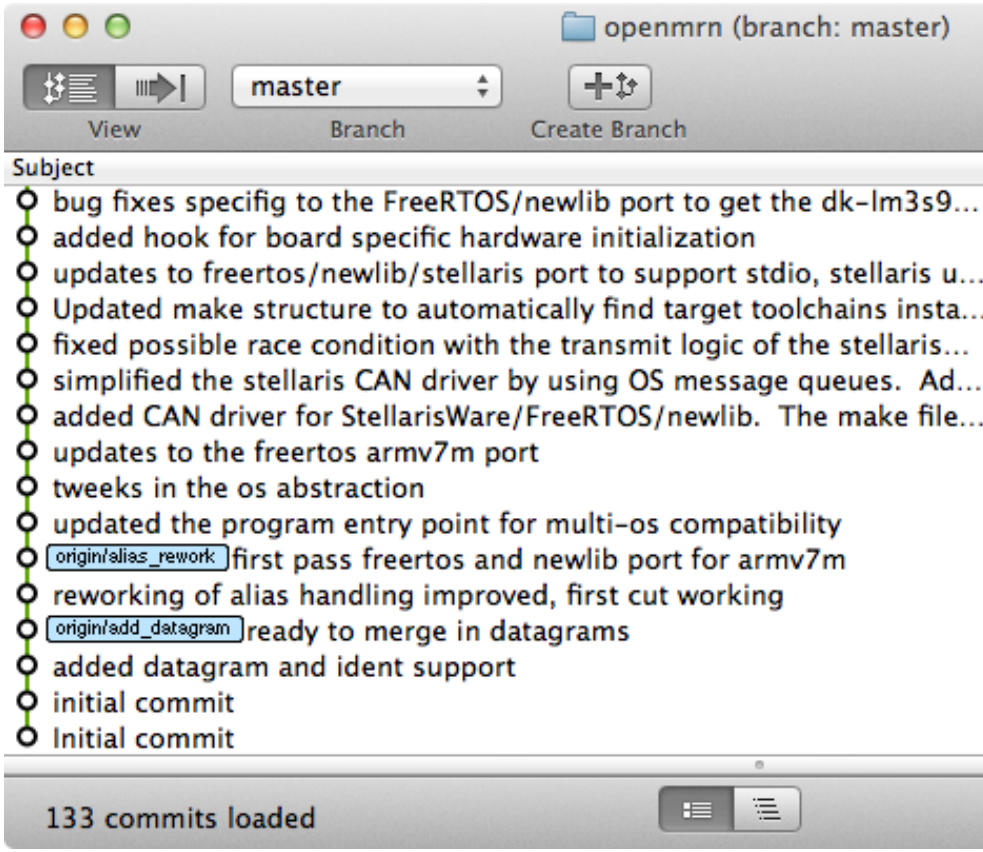
At first, Git looks like SVN

You bring out a copy, work on it, and commit

Git repository contains all that history

More



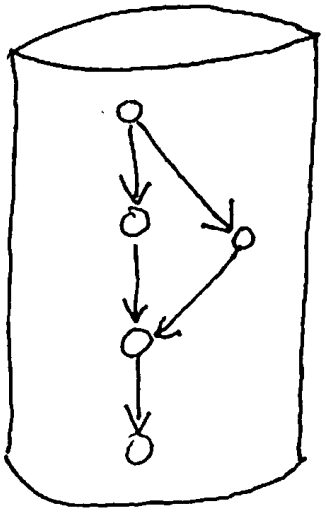


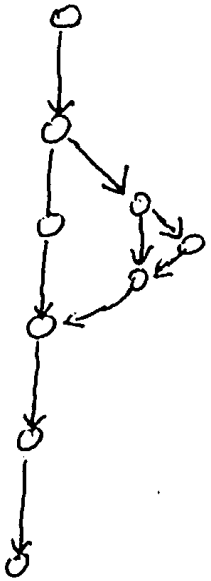
"Scratchpad" idea lets you control what you commit: Shaping the story

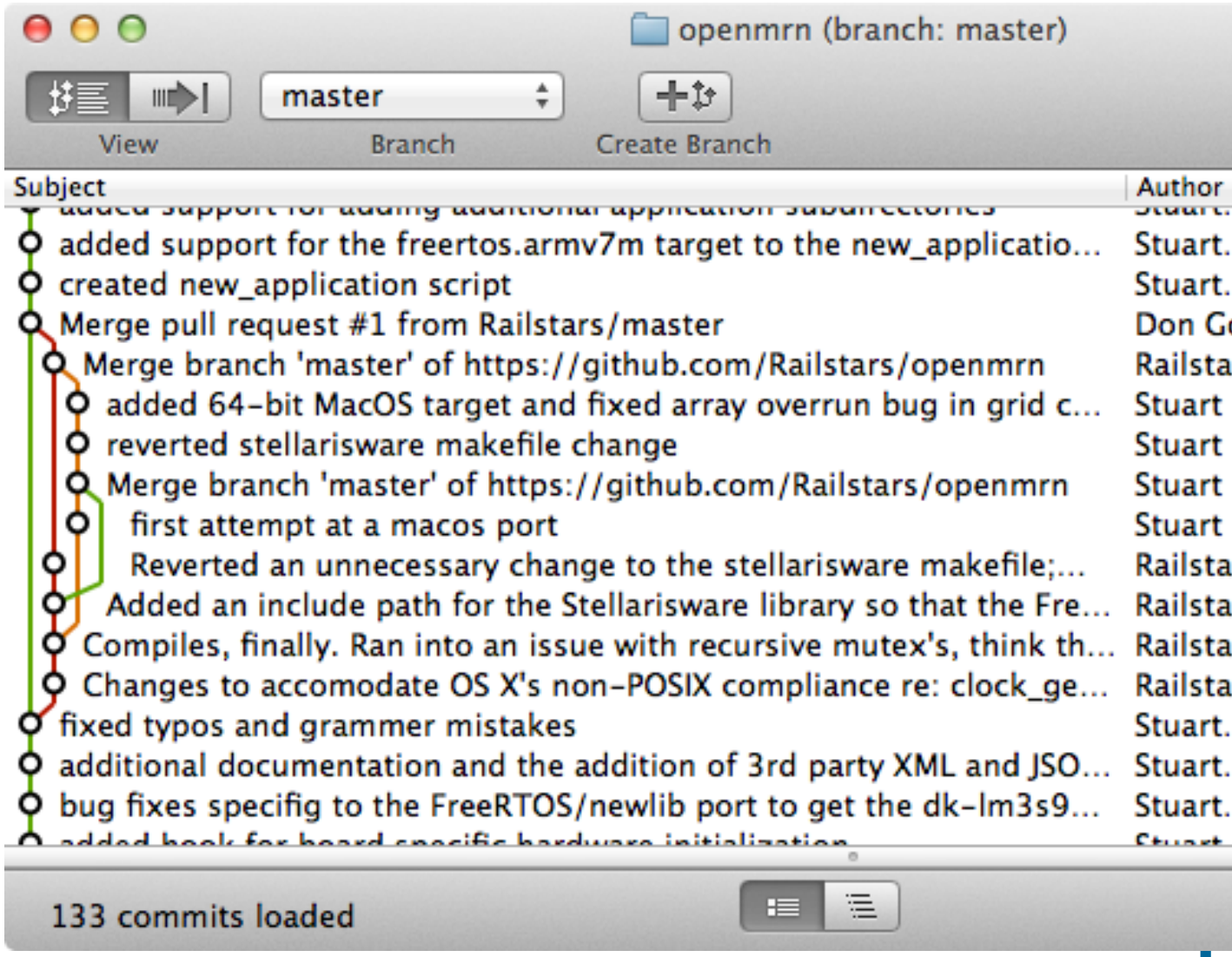
More

Merging

Because Git focuses on commits, not on versions, very powerful merging

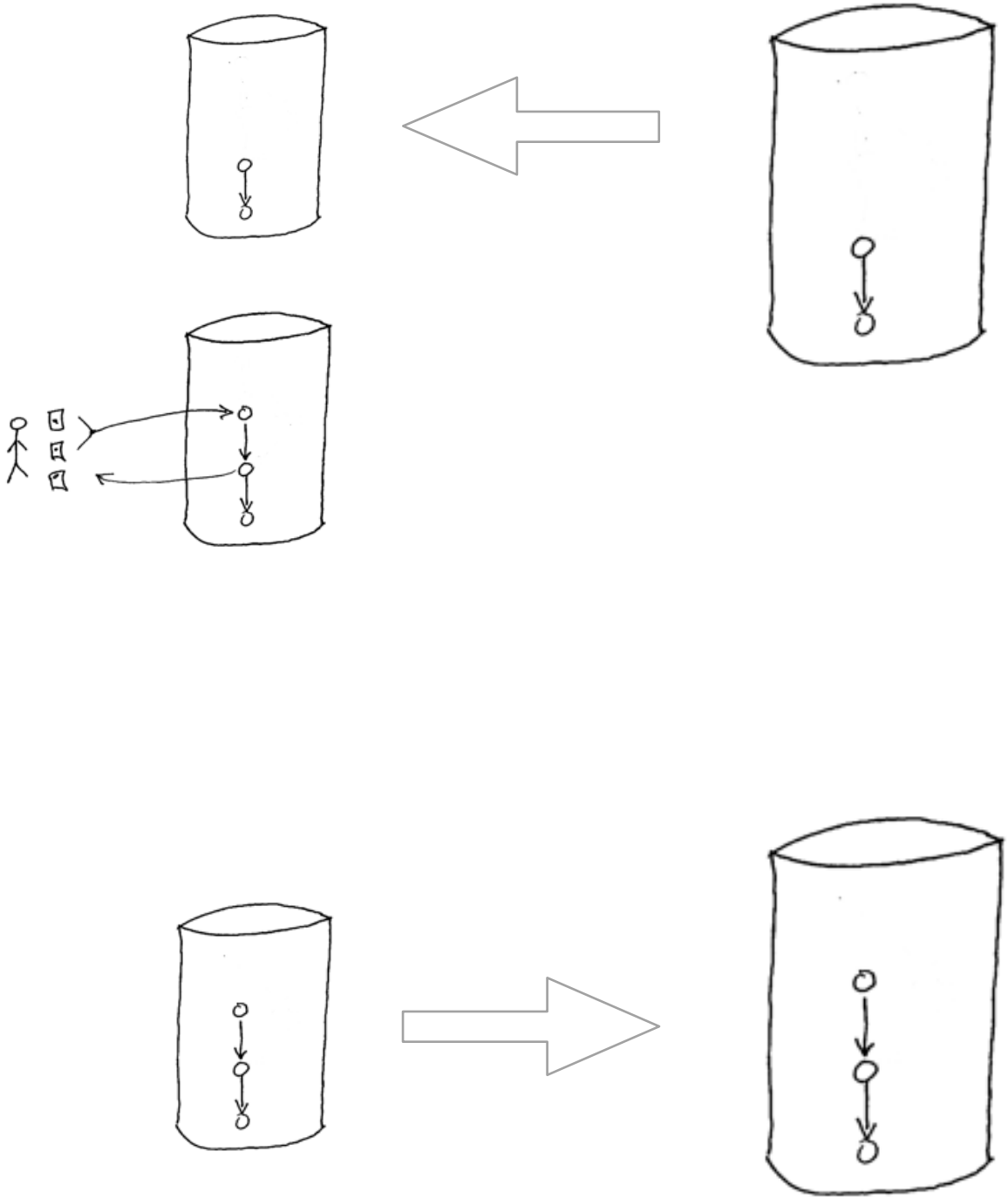




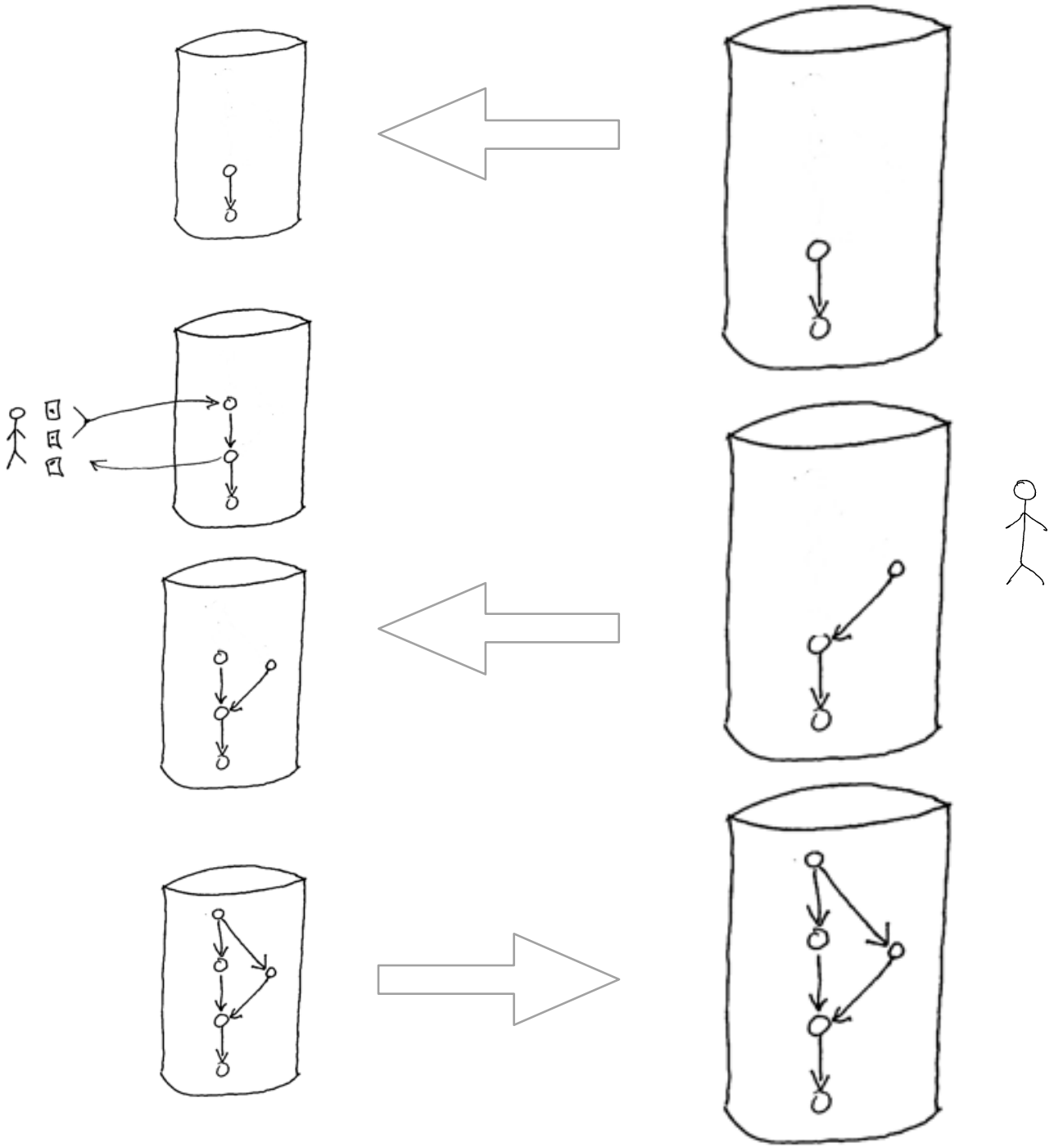


Multiple repositories with easy transfer of commits between

More

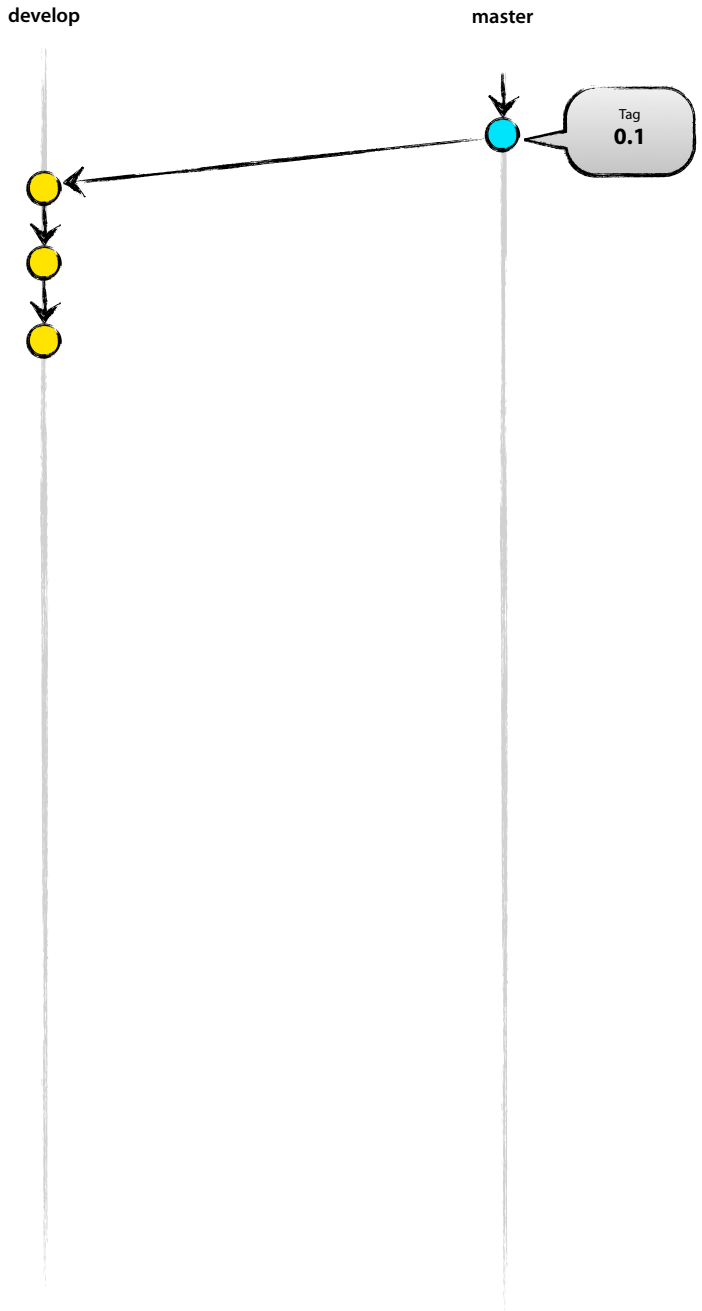


More than just mirroring



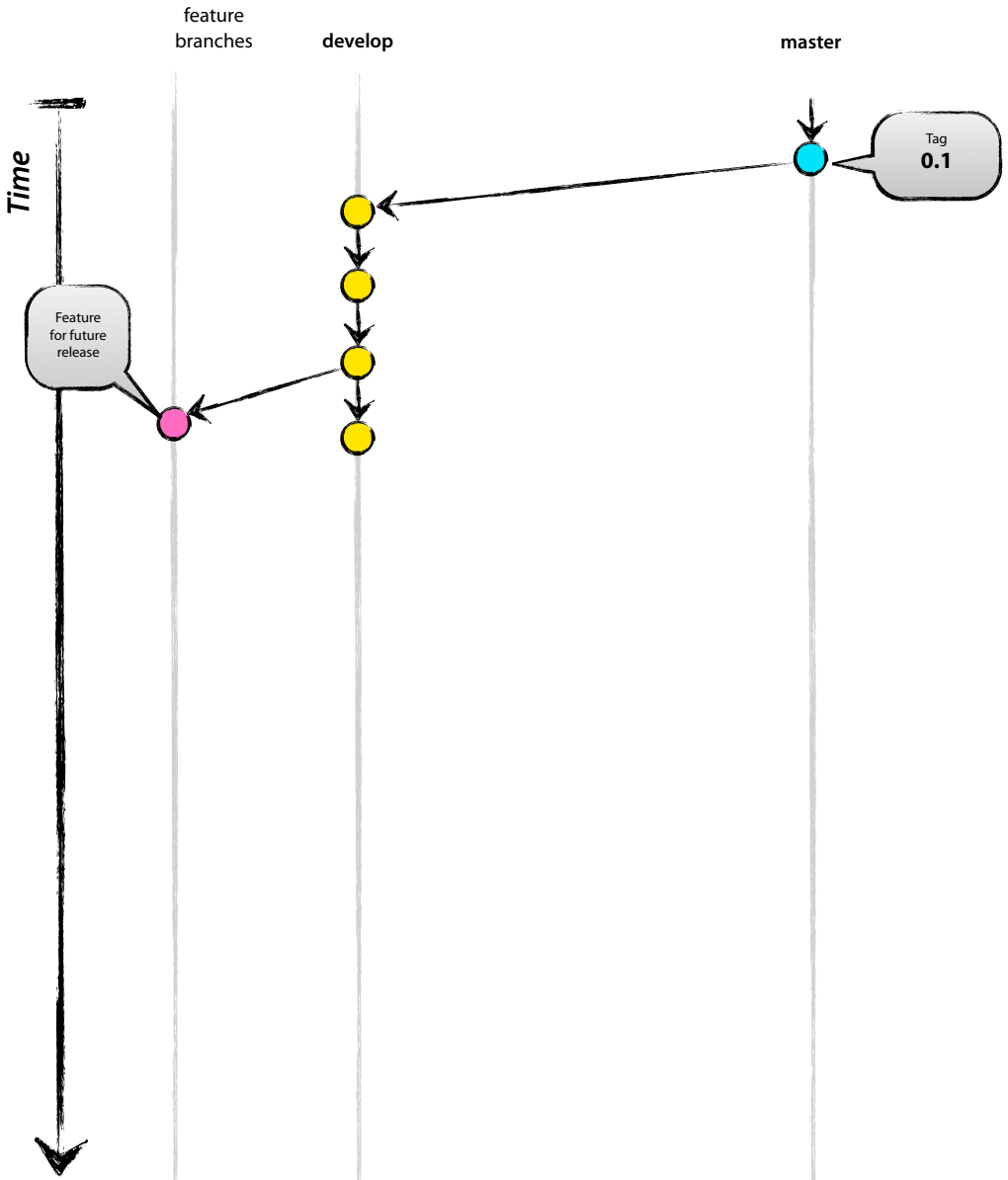
Branches are key

- **Develop on a separate branch**



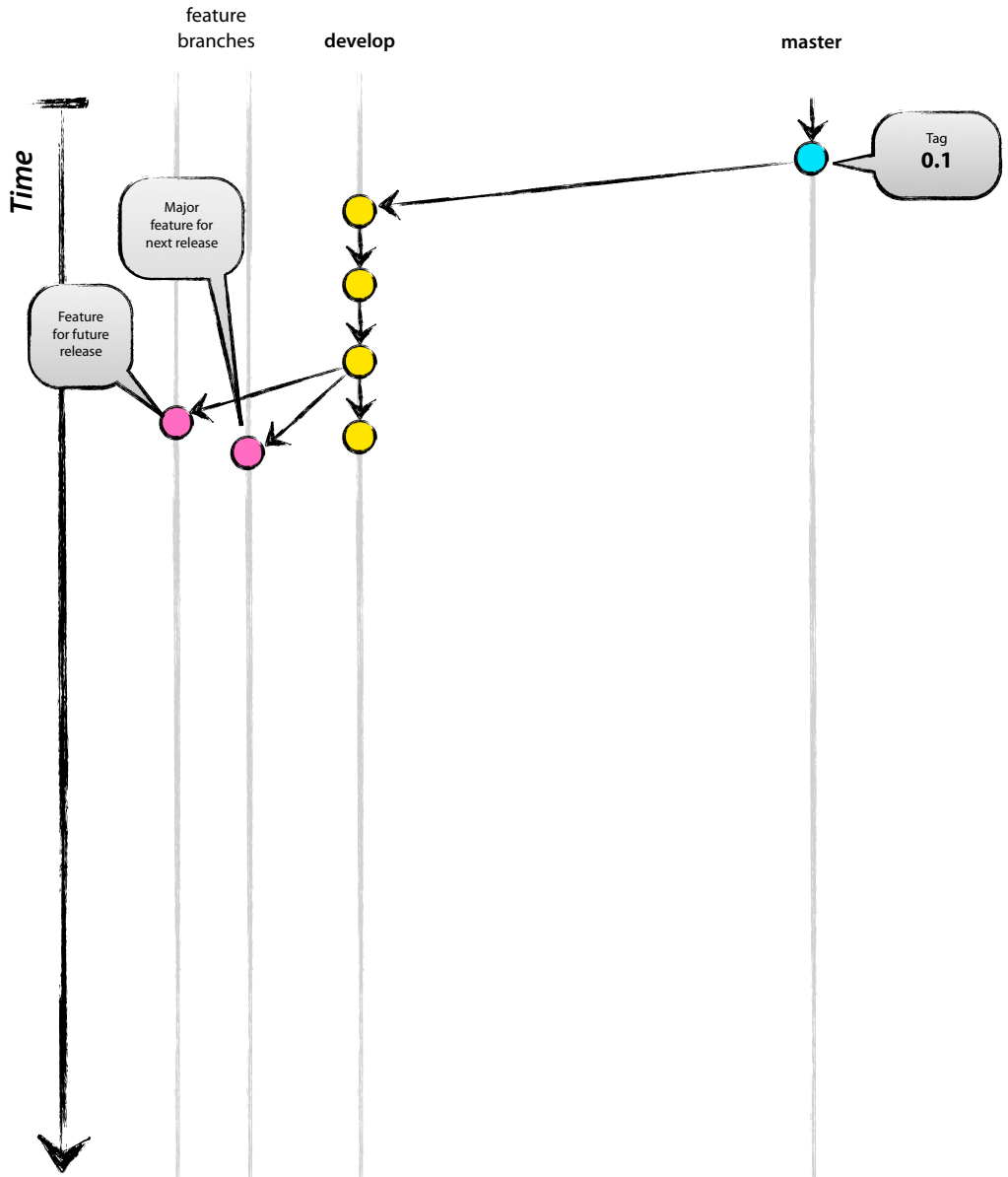
Branches are key

- Develop on a separate branch
- Future Big Feature on branch



Branches are key

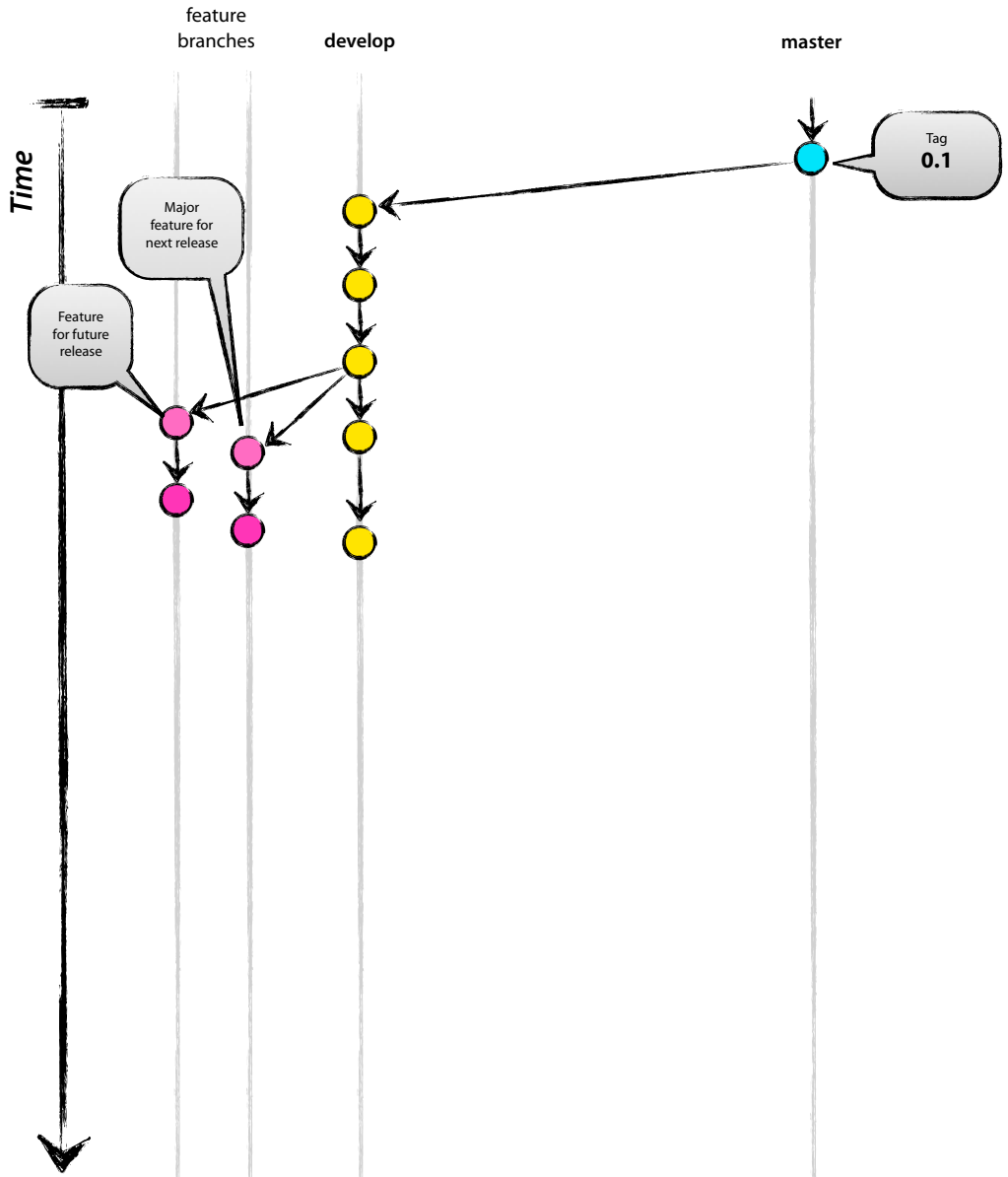
- Develop on a separate branch
- Future Big Feature on branch
- And another one



Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323

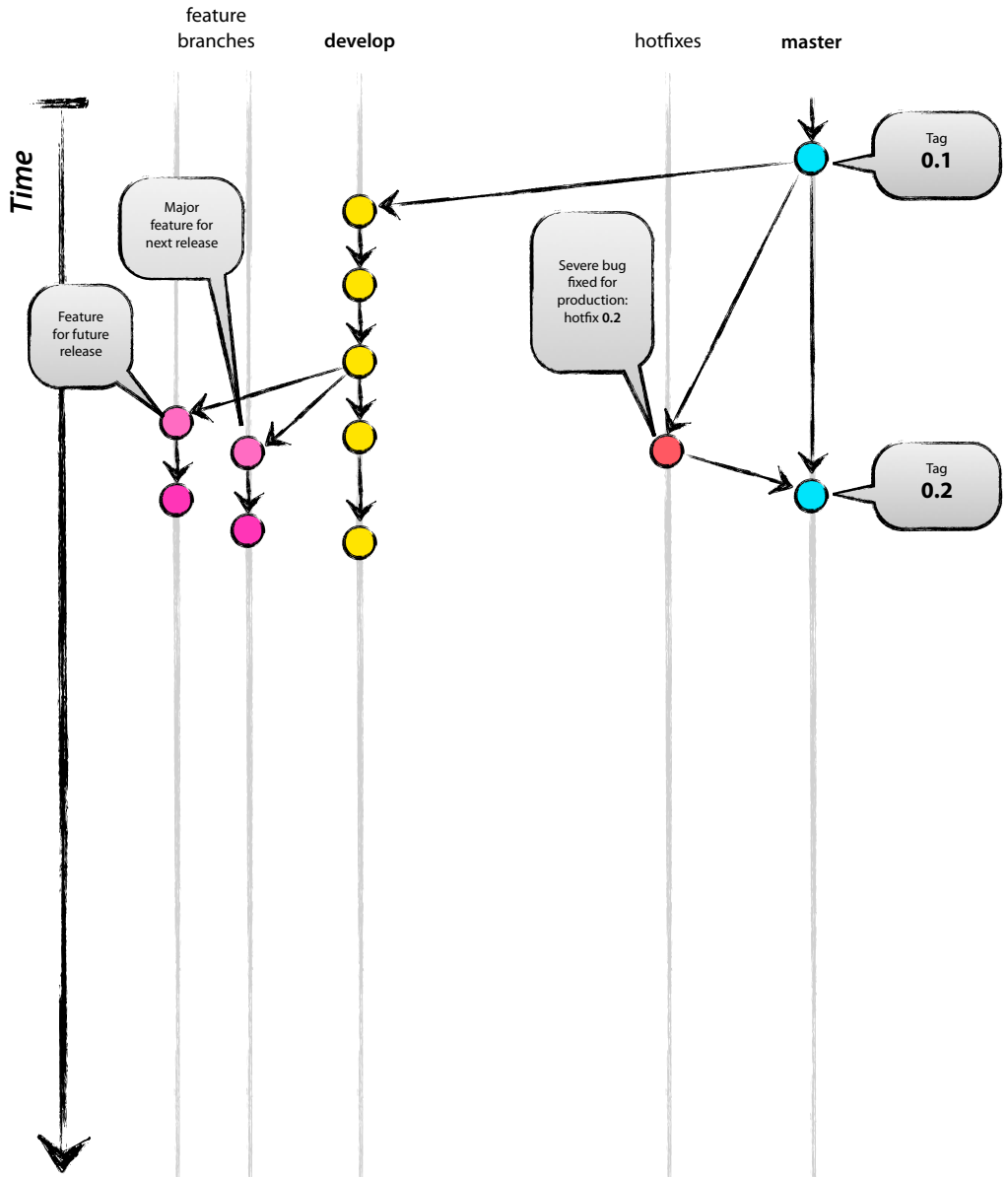
Branches are key

- Develop on a separate branch
- Future Big Feature on branch
- And another one for ll work



Branches are key

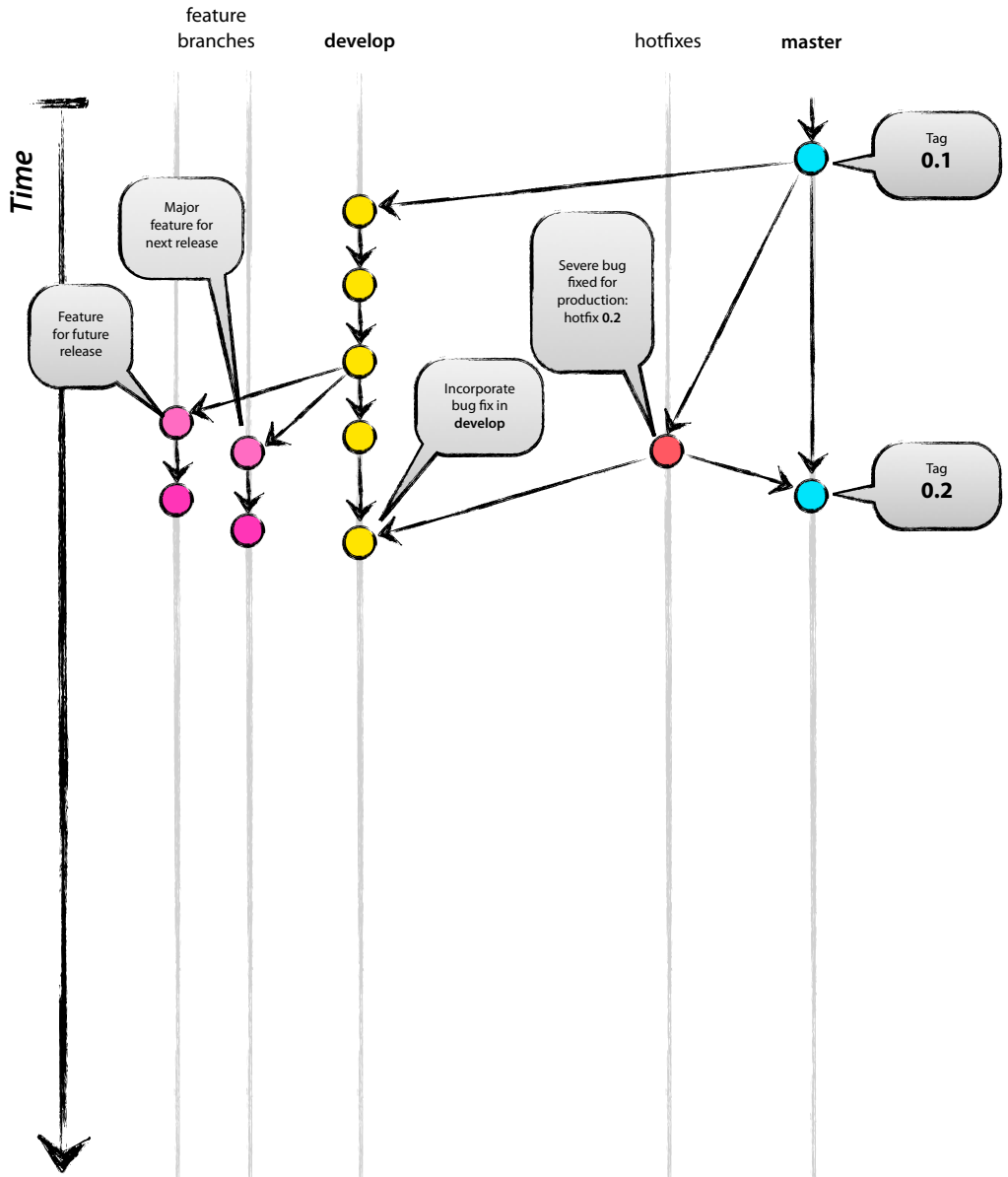
- Develop on a separate branch
- Future Big Feature on branch
- And another one for || work
- Pays off for bug fix!



Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323

Branches are key

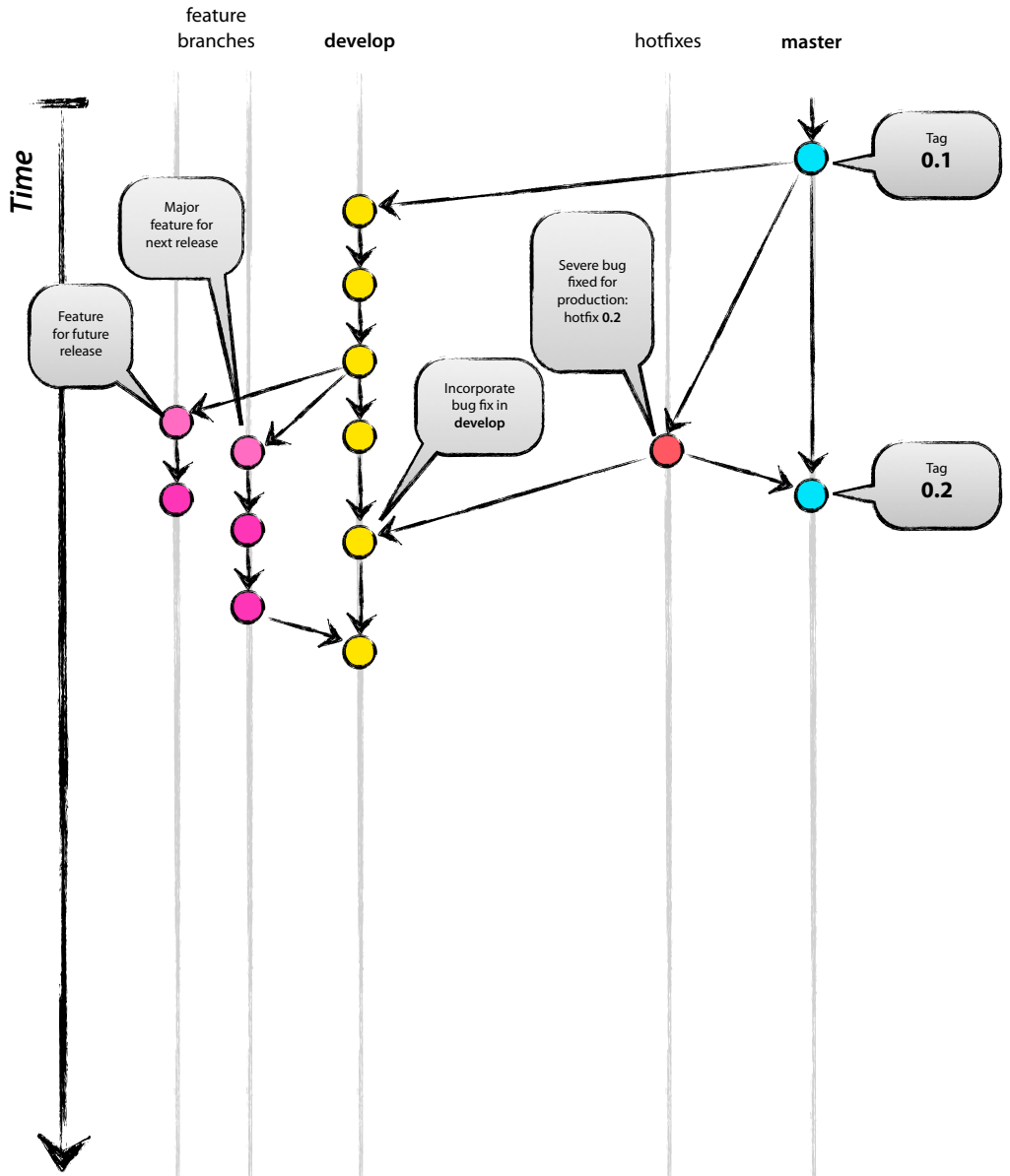
- Develop on a separate branch
- Future Big Feature on branch
- And another one for || work
- Pays off for bug fix!
- Git merge to get fix across



Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323

Branches are key

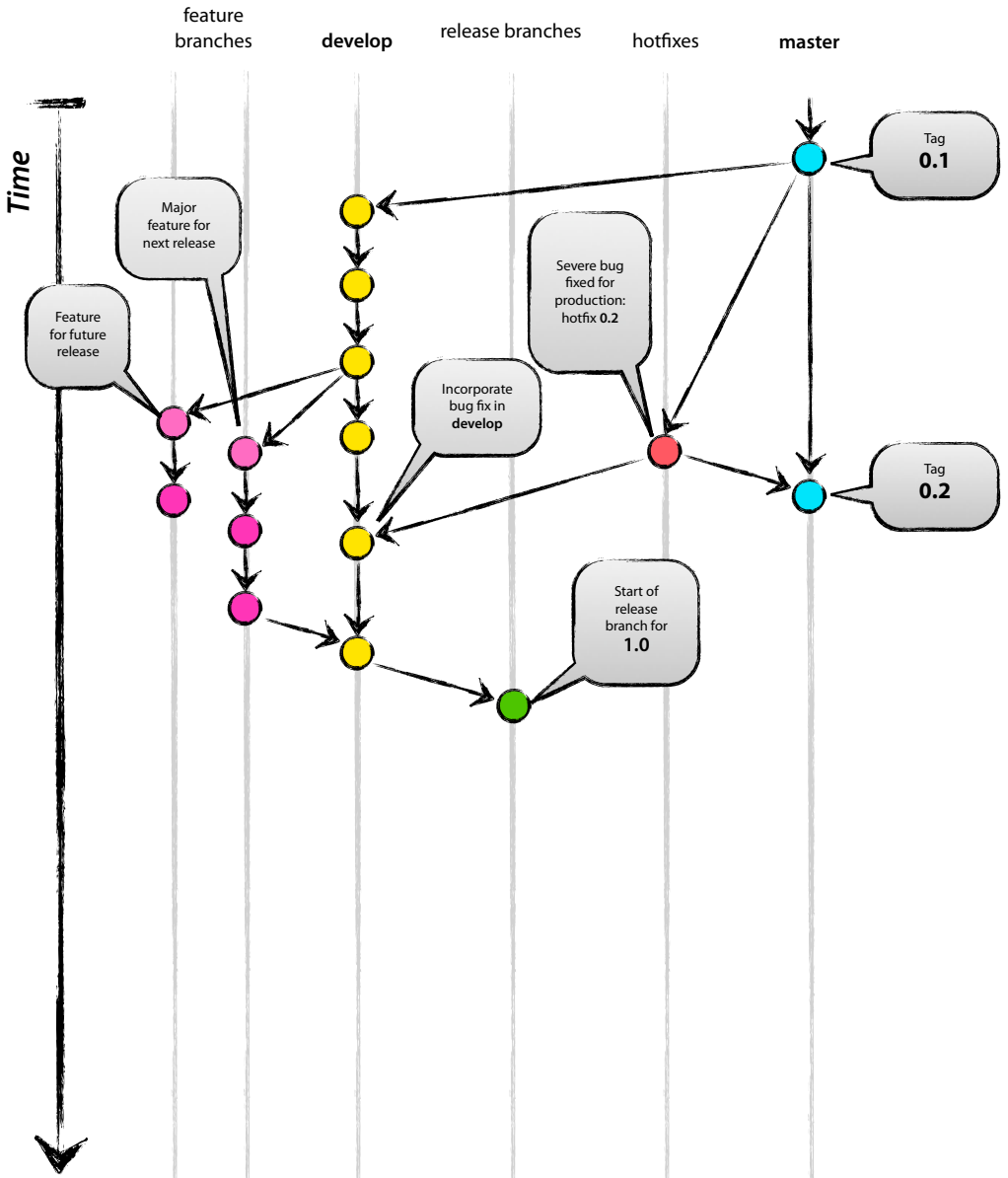
- Develop on a separate branch
- Future Big Feature on branch
- And another one for || work
- Pays off for bug fix!
- Git merge to get fix across
- Feature done, merges in



Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323

Branches are key

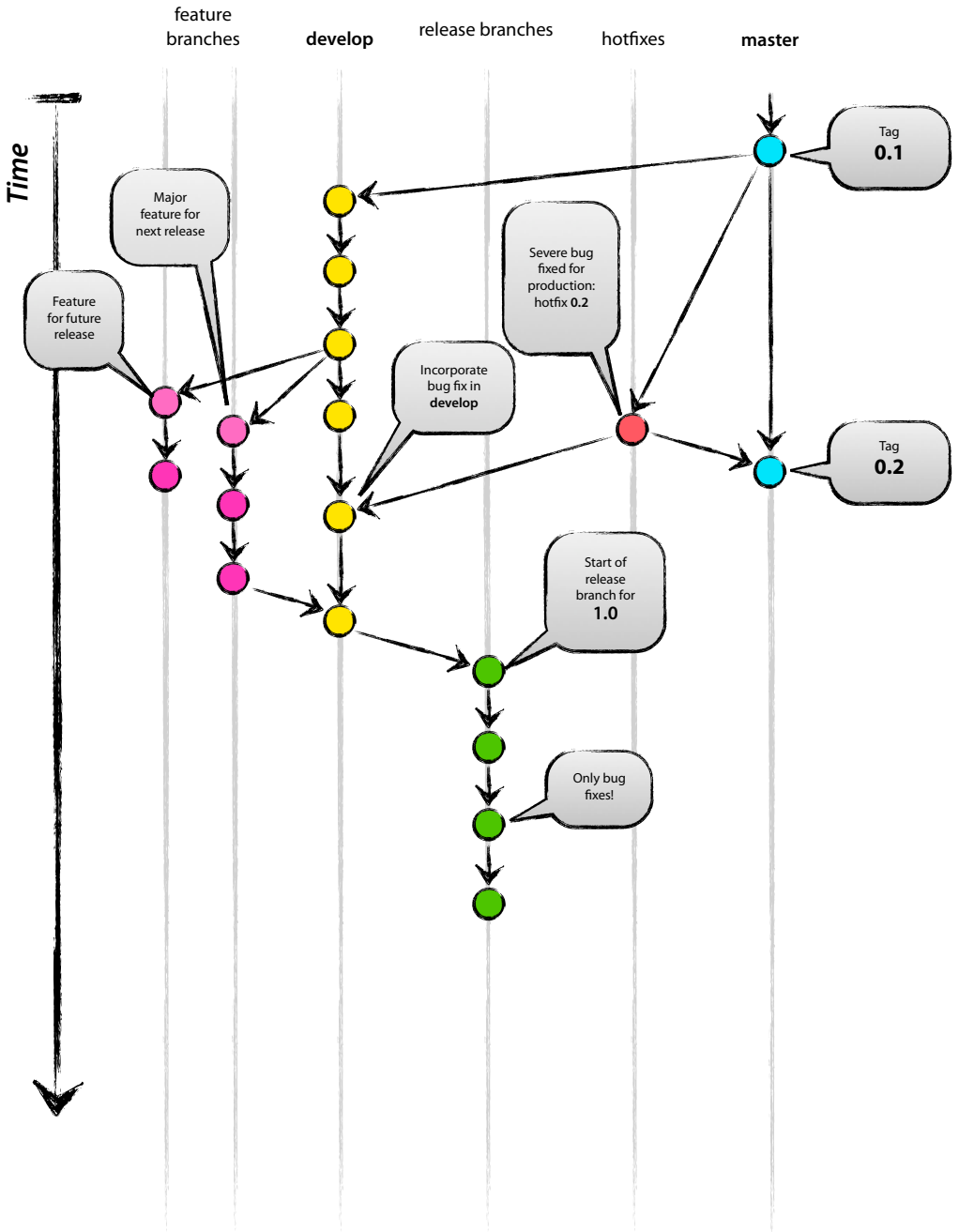
- Develop on a separate branch
- Future Big Feature on branch
- And another one for || work
- Pays off for bug fix!
- Git merge to get fix across
- Feature done, merges in
- New branch holds release



Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323

Branches are key

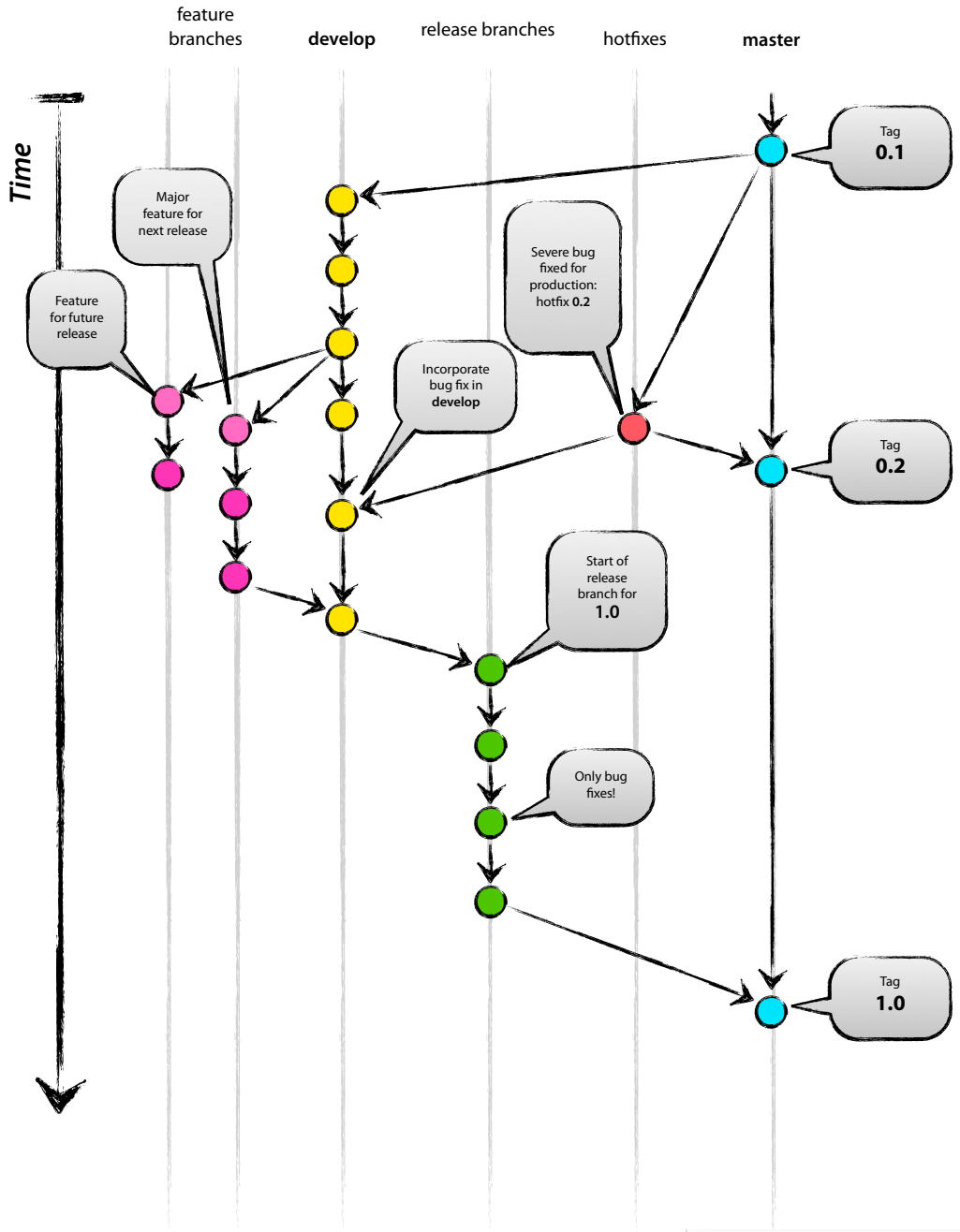
- Develop on a separate branch
- Future Big Feature on branch
- And another one for || work
- Pays off for bug fix!
- Git merge to get fix across
- Feature done, merges in
- New branch holds release
- and it's inevitable fixes



Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323

Branches are key

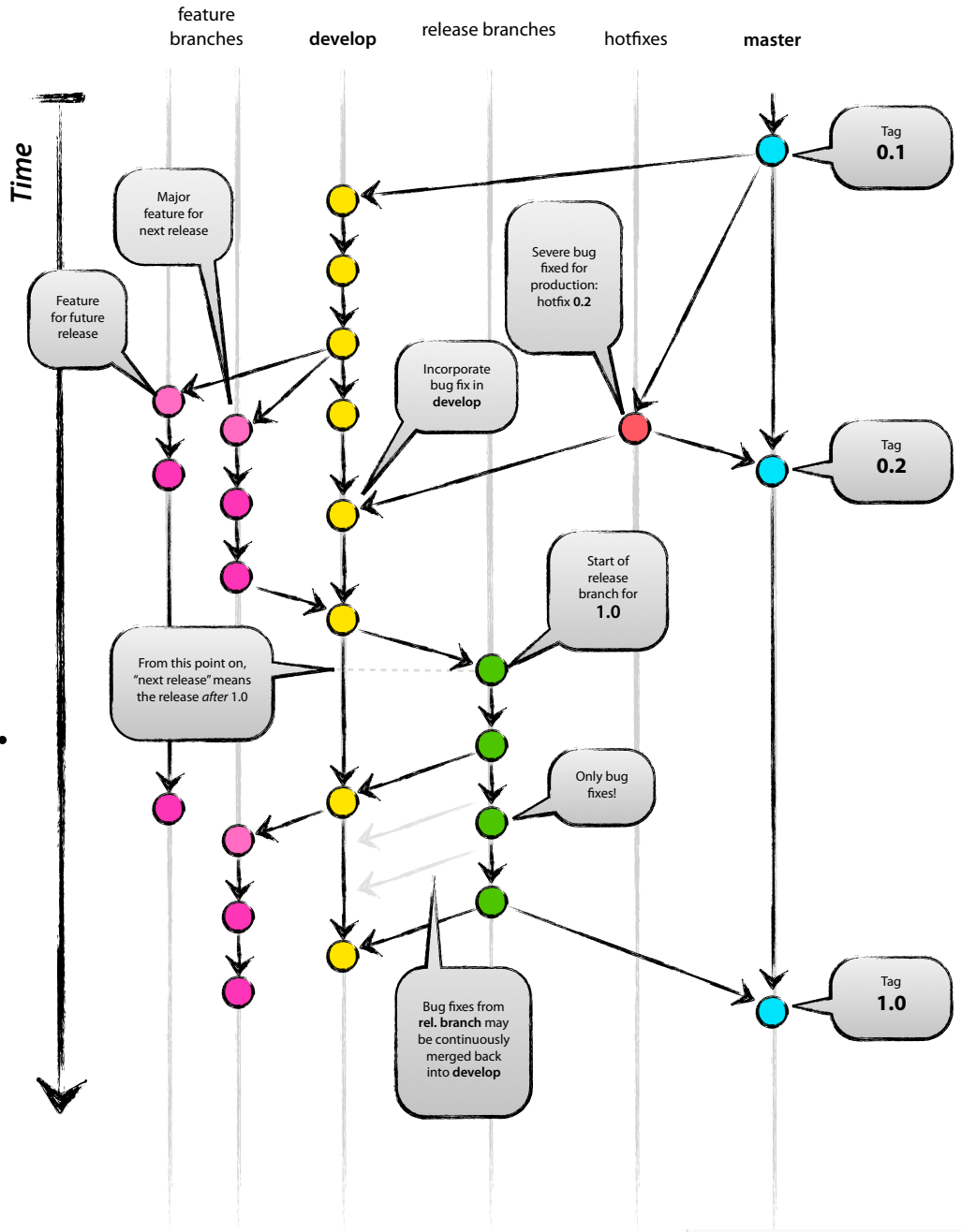
- Develop on a separate branch
- Future Big Feature on branch
- And another one for || work
- Pays off for bug fix!
- Git merge to get fix across
- Feature done, merges in
- New branch holds release
- and it's inevitable fixes
- until merge and release master



Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323

Branches are key

- Develop on a separate branch
- Future Big Feature on branch
- And another one for || work
- Pays off for bug fix!
- Git merge to get fix across
- Feature done, merges in
- New branch holds release
- and it's inevitable fixes
- until merge and release master.
- Meanwhile, work proceeds



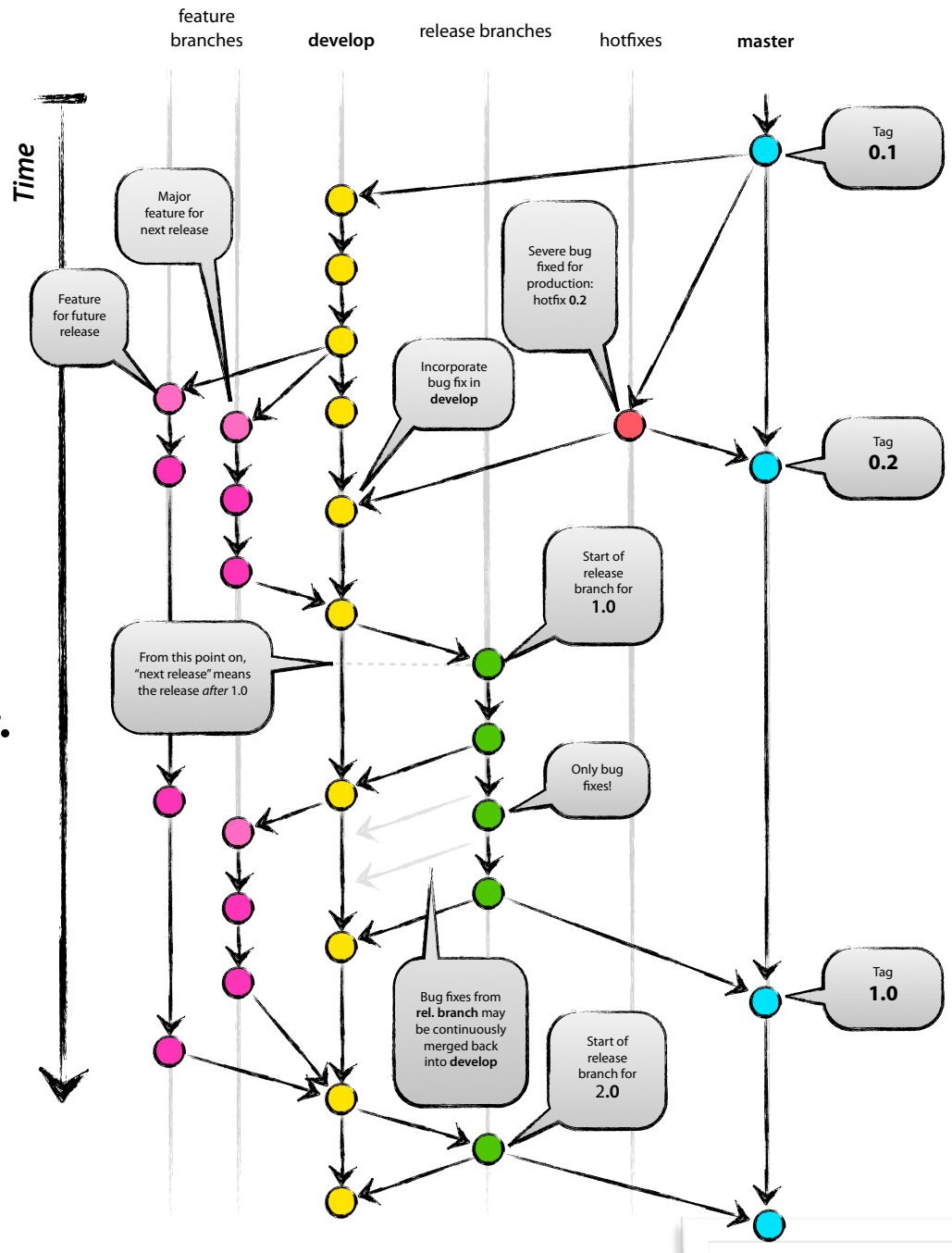
Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323

Branches are key

- Develop on a separate branch
- Future Big Feature on branch
- And another one for || work
- Pays off for bug fix!
- Git merge to get fix across
- Feature done, merges in
- New branch holds release
- and it's inevitable fixes
- until merge and release master.
- Meanwhile, work proceeds
- And the process repeats

Keys: cheap branches,
reliable merges

Gives understandable story

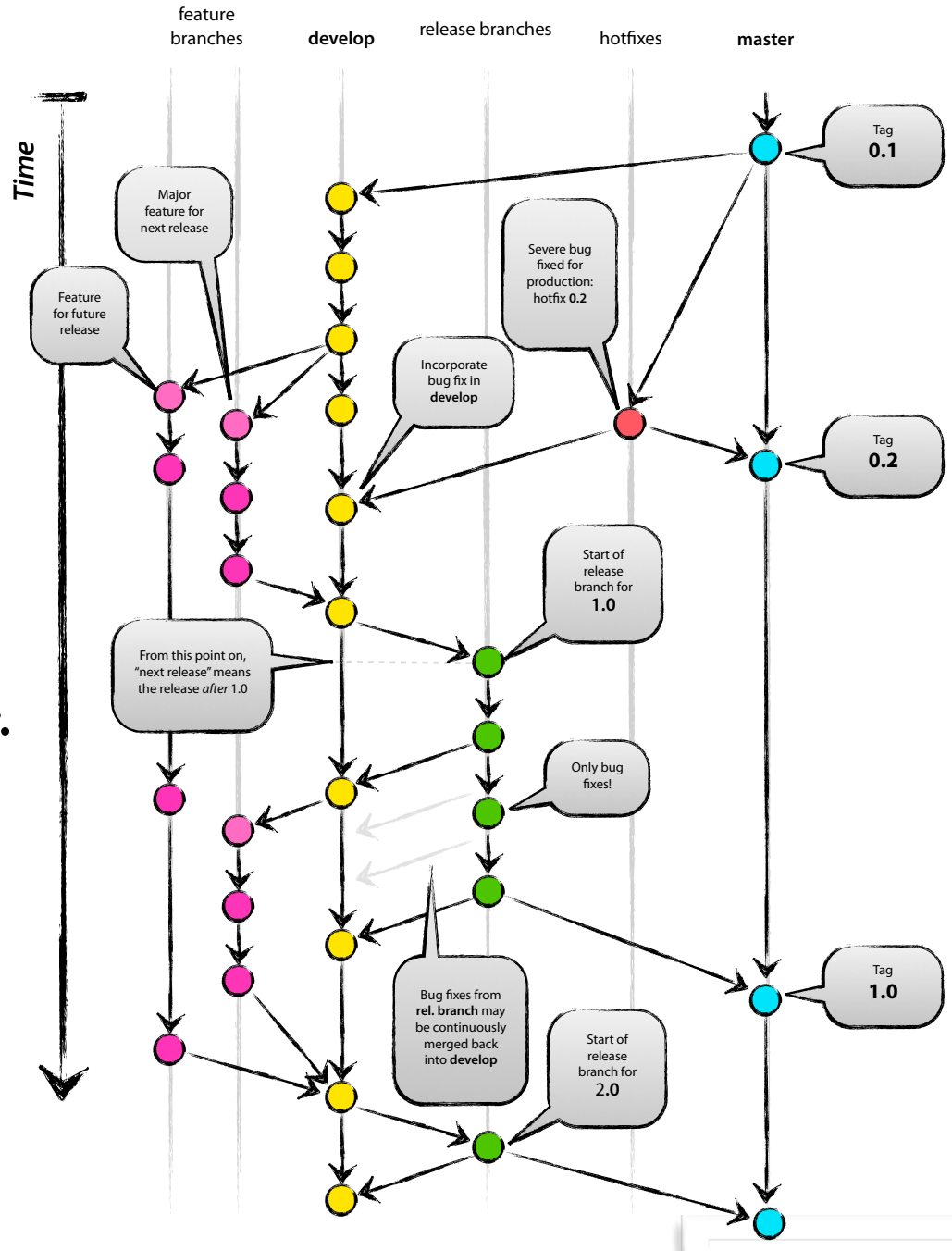


Branches are key

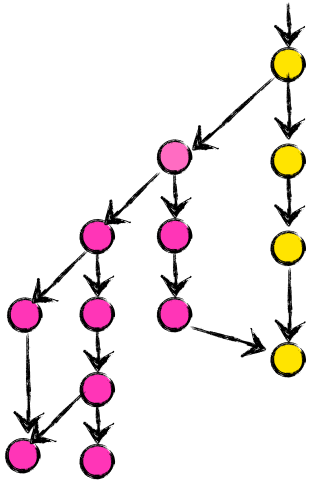
- Develop on a separate branch
- Future Big Feature on branch
- And another one for || work
- Pays off for bug fix!
- Git merge to get fix across
- Feature done, merges in
- New branch holds release
- and it's inevitable fixes
- until merge and release master.
- Meanwhile, work proceeds
- And the process repeats

Keys: cheap branches,
reliable merges

Gives understandable story

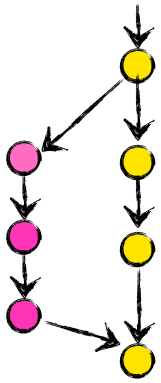


Rebase: An Editor for the Story

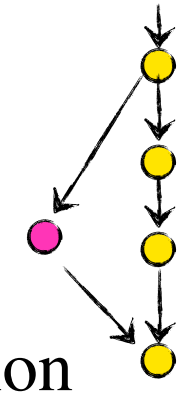


Finished difficult development task, after several dead ends, lots of little bits of progress & dead ends

More

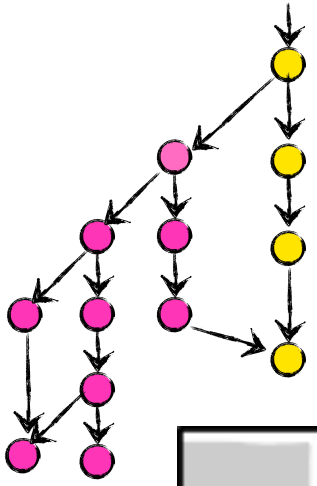


Deleting only gets you so far



"Rebase" operation

Rebase: An Editor for the Story



Finished difficult development task, after several dead ends, lots of little bits of progress & dead ends

More

You want me to trust how many people?

CERN School of Computing

More

How do you give 6,000 people access to a central repository?

A) Don't! Have them submit patches to Package Coordinators (PBs)

```
Index: java/src/jmri/jmrix/openlcb/swing/downloader/LoaderPane.java
===================================================================
--- java/src/jmri/jmrix/openlcb/swing/downloader/LoaderPane.java        (revision 29731)
+++ java/src/jmri/jmrix/openlcb/swing/downloader/LoaderPane.java        (working copy)
@@ -186,18 +186,19 @@
          */
         void sendNext() {
             byte[] temp = new byte[SIZE];
-            int i;
-            for (i = 0; i < SIZE; i++) {
-                if (!inputContent.locationInUse(location+i))
+            int count;
+            for (count = 0; count < SIZE; count++) {
+                if (!inputContent.locationInUse(location+count)) {
+                    break;
-                temp[i] = (byte)inputContent.getLocation(location+i);
+                }
+                temp[count] = (byte)inputContent.getLocation(location+count);
             }
-            byte[] data = new byte[i];
-            System.arraycopy(temp, 0, data, 0, i);
+            byte[] data = new byte[count];
+            System.arraycopy(temp, 0, data, 0, count);

             int addr = location; // next call back might be instantaneous
-            location = location + i;
-            log.info("Sending write to 0x{}", Integer.toHexString(location).toUpperCase());
+            location = location + count;
+            log.info("Sending write to 0x{} length {}", Integer.toHexString(location).toUpperCase(), count);
             mcs.request(new MemoryConfigurationService.McsWriteMemo(destNodeID(), space, addr, data) {
                 public void handleWriteReply(int code) {
                     // update GUI intermittently
```

Enough info for reliable commit, but not a lot of context, and no reliable way to merge back if commit is delayed

You want me to trust how many people?

How do you give 6,000 people access to a central repository?

B) Find reliable people and give them access, log commits

```
Revision: 29733
        http://sourceforge.net/p/jmri/code/29733
Author:   jacobsen
Date:     2015-08-09 23:20:19 +0000 (Sun, 09 Aug 2015)
Log Message:
-----------
Better index variable name; improve logging message

Modified Paths:
--------------
   trunk/jmri/java/src/jmri/jmrix/openlcb/swing/downloader/LoaderPane.java

Modified: trunk/jmri/java/src/jmri/jmrix/openlcb/swing/downloader/LoaderPane.java
===================================================================
--- trunk/jmri/java/src/jmri/jmrix/openlcb/swing/downloader/LoaderPane.java       2015-08-08 23:10:01 UTC (rev 29732)
+++ trunk/jmri/java/src/jmri/jmrix/openlcb/swing/downloader/LoaderPane.java       2015-08-09 23:20:19 UTC (rev 29733)
@@ -186,18 +186,19 @@
         */
        void sendNext() {
           byte[] temp = new byte[SIZE];
-          int i;
-          for (i = 0; i < SIZE; i++) {
-              if (!inputContent.locationInUse(location+i))
+          int count;
+          for (count = 0; count < SIZE; count++) {
+              if (!inputContent.locationInUse(location+count)) {
                 break;
-              temp[i] = (byte)inputContent.getLocation(location+i);
+              }
+              temp[count] = (byte)inputContent.getLocation(location+count);
           }
-          byte[] data = new byte[i];
-          System.arraycopy(temp, 0, data, 0, i);
+          byte[] data = new byte[count];
+          System.arraycopy(temp, 0, data, 0, count);

           int addr = location; // next call back might be instantaneous
-          location = location + i;
-          log.info("Sending write to 0x{}", Integer.toHexString(location).toUpperCase());
+          location = location + count;
+          log.info("Sending write to 0x{} length {}", Integer.toHexString(location).toUpperCase(), count);
           mcs.request(new MemoryConfigurationService.McsWriteMemo(destNodeID(), space, addr, data) {
              public void handleWriteReply(int code) {
                 // update GUI intermittently
```

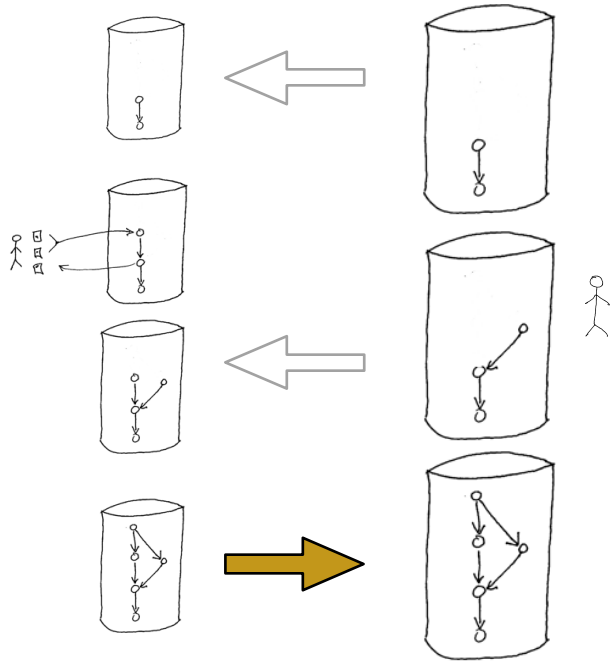
Solves to context & merge-back problem,

but do you really have 6,000 reliable friends?

You want me to trust how many people?

How do you give 6,000 people access to a central repository?

C) Use a distributed repository and "pull requests"



Git-based developers have a full local repository

Commits have full context

"Push" moves all that to target

A "pull request" sends all that to somebody at the target, who can accept or not

When accepted, the merge is completed & both repositories in sync

(Pull requests rarely rejected outright - usually it's "fix these things and resend")

Strong tools exist to make pull requests easy: CI test results, etc automated

More

2nd approach: People handle consistency, machines build

Once consistency is managed, building is easier to automate

More

Enter "CMake"

Two phase process:

- (Zeroth: Pull a consistent set of code from managed repository)
- First, automatically build localized control files
- Second, do a platform-specific build using those files

```
cmake path
make
```

CMake:

- Scales very well (builds entire Linux distributions, LCGsoft LHC software)
- Well integrated with other tools (Eclipse, Visual Studio, the whole world)
- Powerful capabilities

Are you sure I don't need a version system for packages?

We use version control outside SVN because it's too hard to have lots of independent, controlled versions inside SVN.

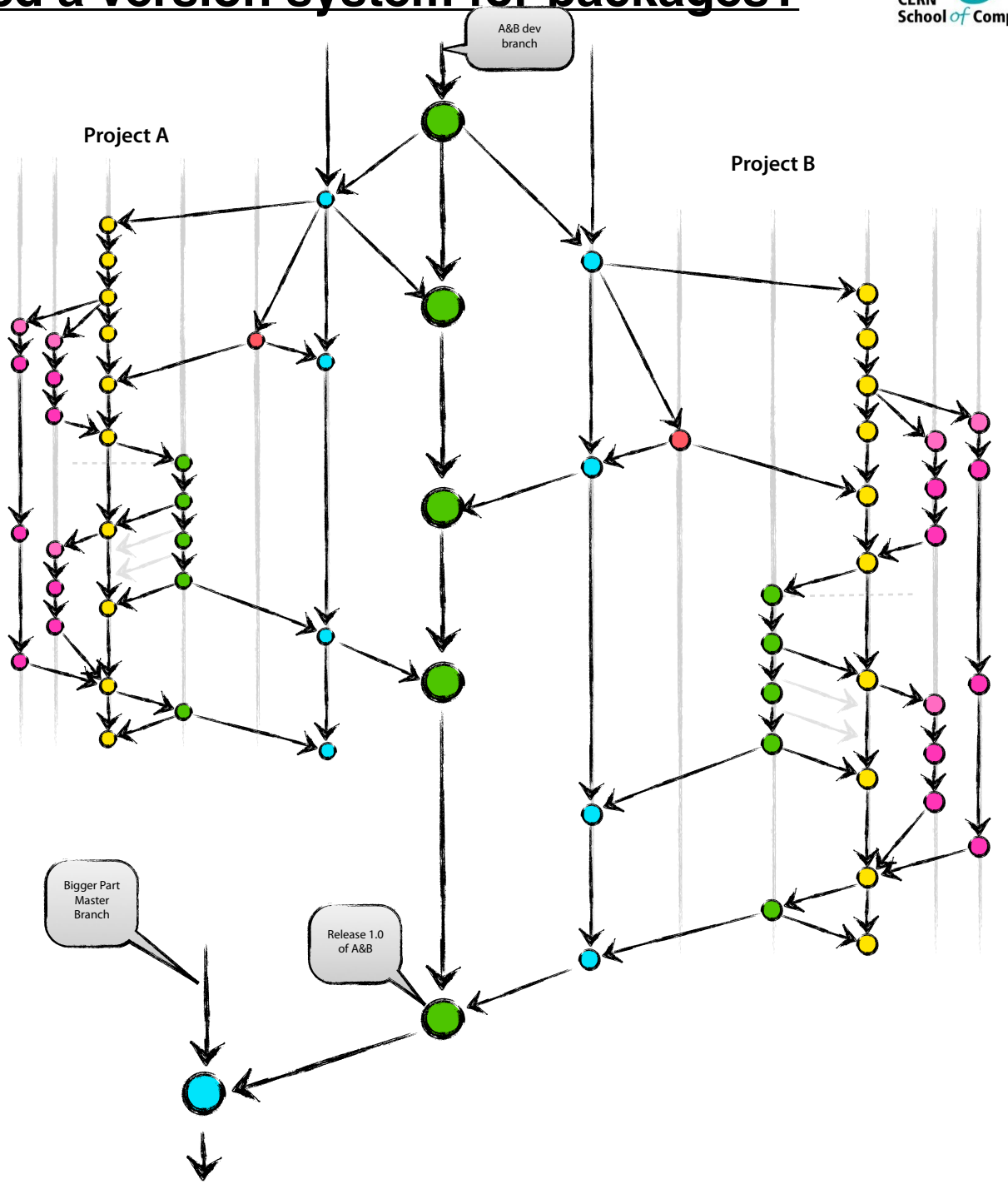
Git's "Lots of branches"

+

strong & easy merging

is

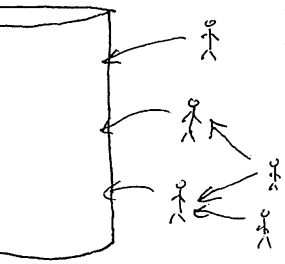
qualitatively different



Compare Approaches:

SVN & CMT

- **Code in repository**
- **Unit of organization: Package**



Package Coordinator decides:

- Release time & contents
- Dependency rules

- **Tools create releases**

Resolve dependencies

- At package level

Specify localization

- **Build and distribute**

Pre-made Makefiles

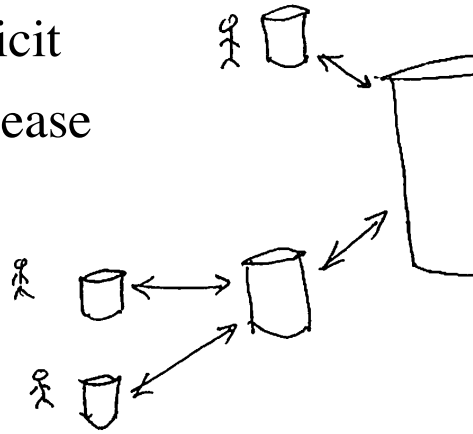
Git & CMake

- **Code in repository**
- **Unit of organization: Branch**

Fractal organization within

- Common time, contents
- Dependency implicit

Cooperate to define release



- **Check out and build**

Consistent release from branch

CMake handles localization

Exercises

Test Frameworks

Performance Profiling

Memory Issues

Code Management

Release Management

More

Instructions at: http://server/jake/CSC/index.html

You'll work in pairs. Try to find somebody with complementary skills!

Learn about each topic, spend more time on the ones that interest you.

Speed is not the issue: no reward for first done, no complaint about last.

Think about what you're doing: There are larger lessons to be found!

Lecture summary

Software engineering is the art of building complex computer systems

It's ideas and techniques spring from our need to handle size & complexity

As you do your own work & develop your own skills, consider:

- How your effort effects or contributes to things 10X, 100X, 1000X larger
- How you'll do things different/better when it's your problem