Version Control

Kenneth M. Anderson

University of Colorado, Boulder

CSCI 5828 — Lecture 13 — 02/24/2009

Goals

- Review material from Chapter 6 of Pilone & Miles
  - Version Control & Configuration Management
    - Working “Without a Net”
    - Repository Management
      - Init, Add, Branch, Merge

Without a Net (I)

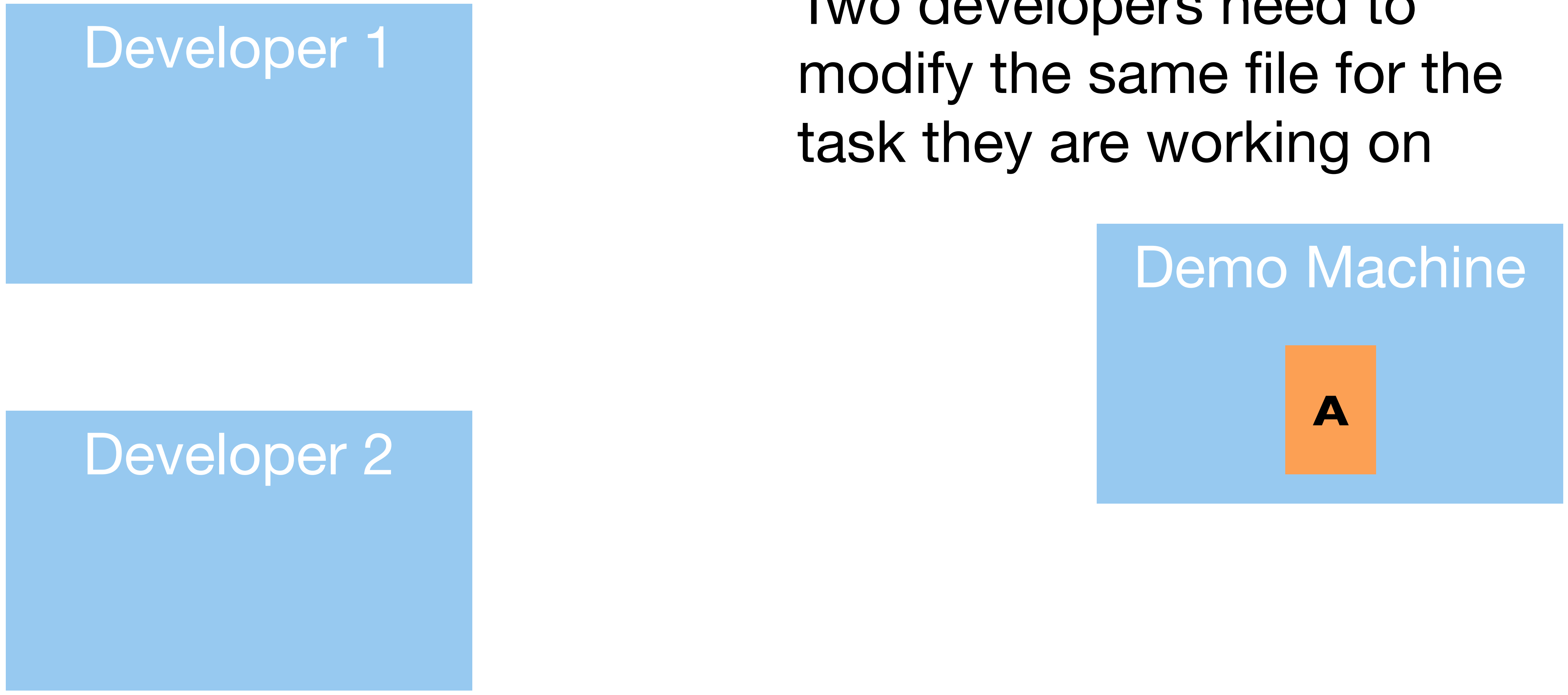
- Doing software development without configuration management is “working without a net”
  - Configuration management refers to both a process and a technology
    - The process encourages developers to work in such a way that changes to code are tracked
      - changes become “first class objects” that can be named, tracked, discussed and manipulated
    - The technology is any system that provides features to enable this process

Without a Net (II)

- If you don’t use configuration management then
  - you are not keeping track of changes
  - you won’t know when features were added
  - you won’t know when bugs were introduced or fixed
  - you won’t be able to go back to old versions of your software
- You would be “living in the now” with the code
  - There is only one version of the system: this one
- You would have no safety net

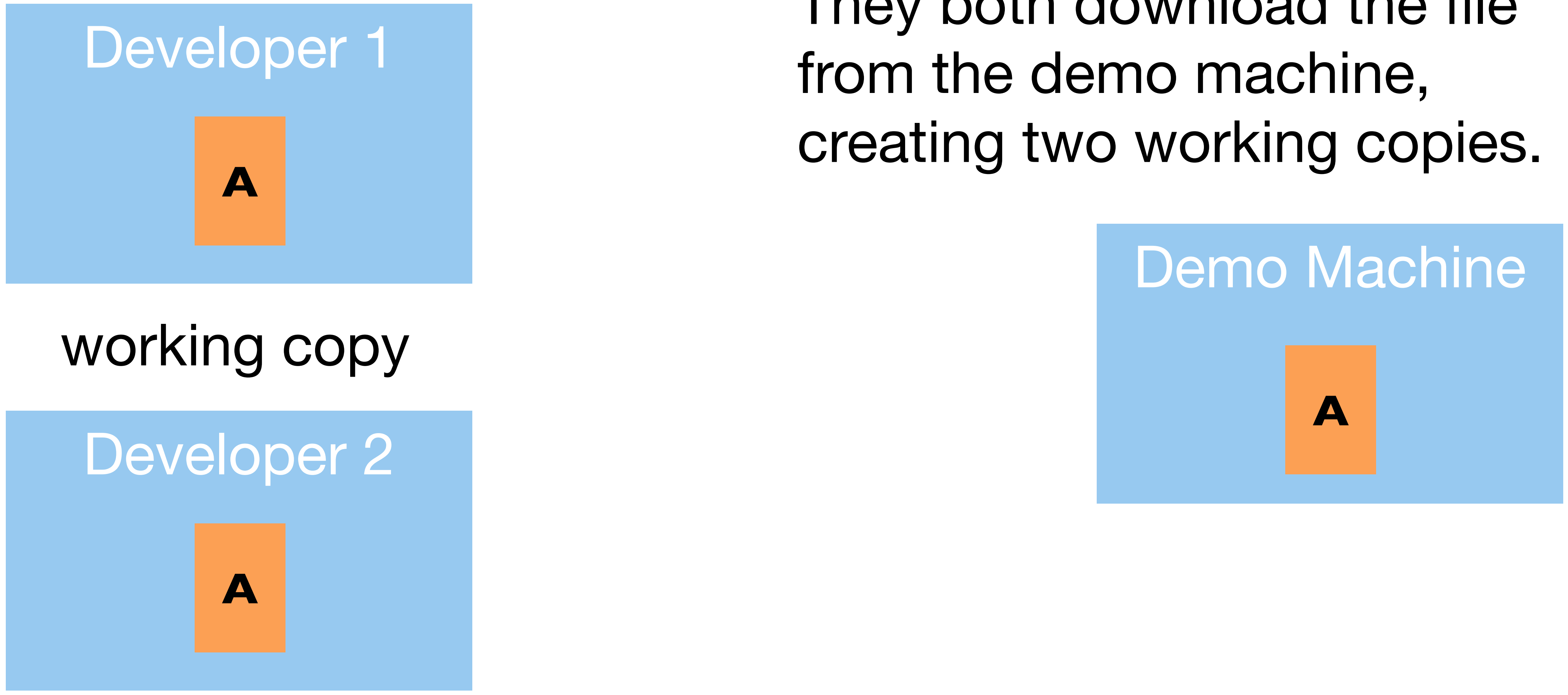
Without a Net (III)

Two developers need to modify the same file for the task they are working on



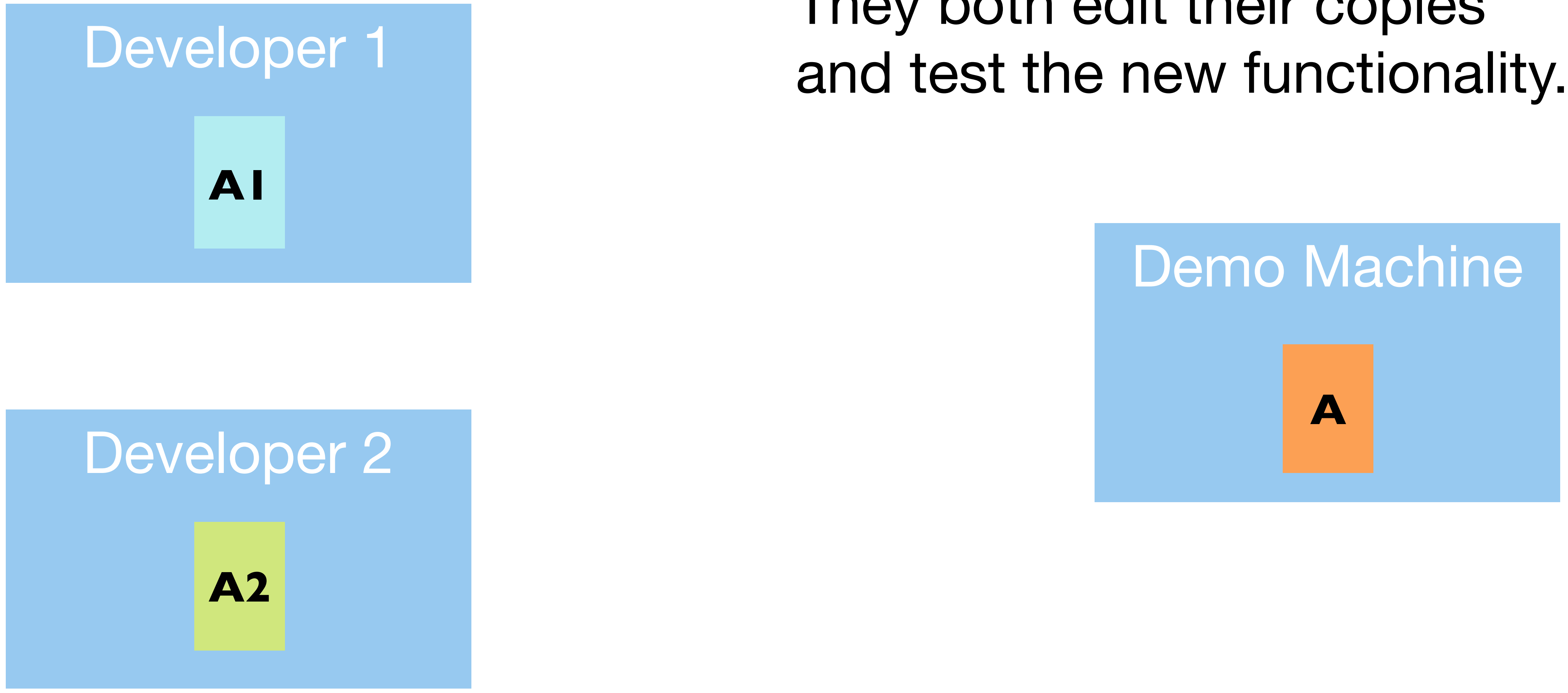
Without a Net (IV)

They both download the file from the demo machine, creating two working copies.



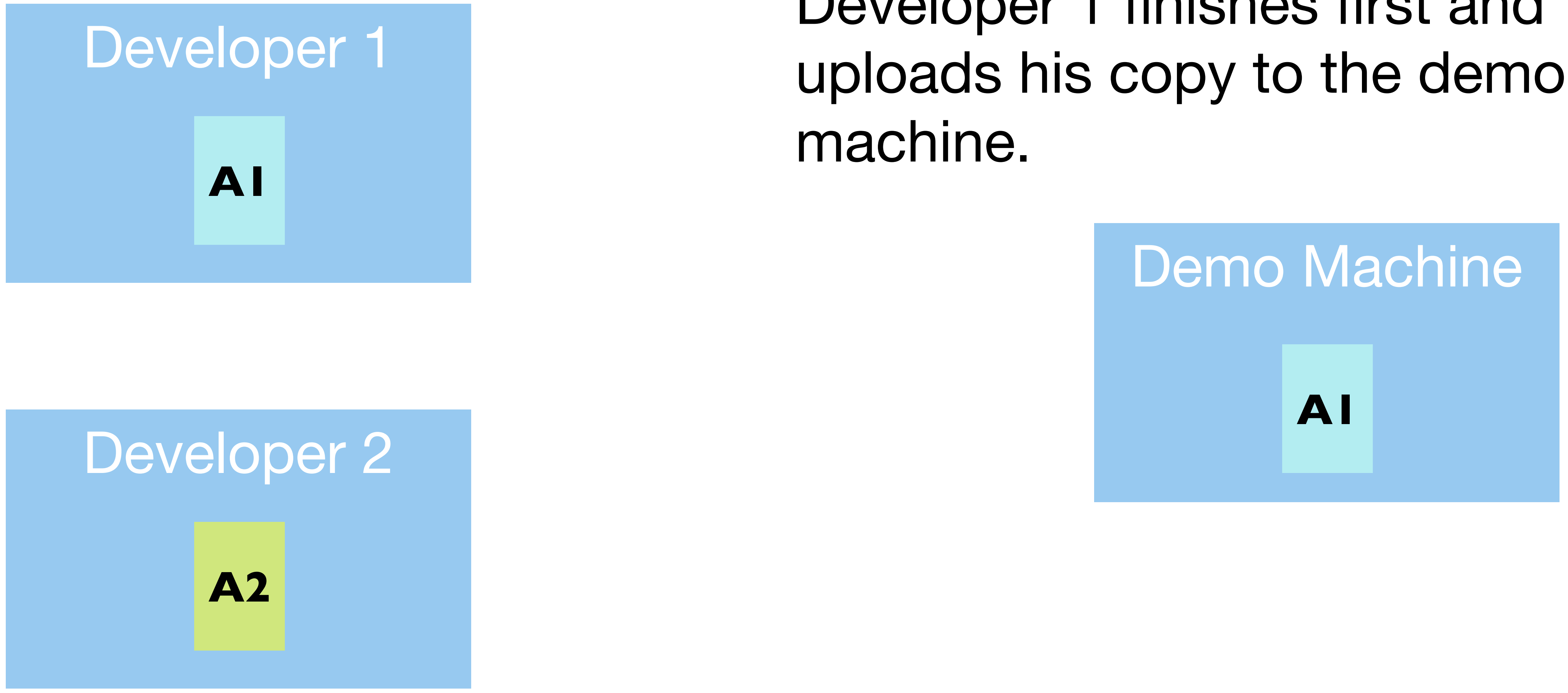
Without a Net (V)

They both edit their copies and test the new functionality.

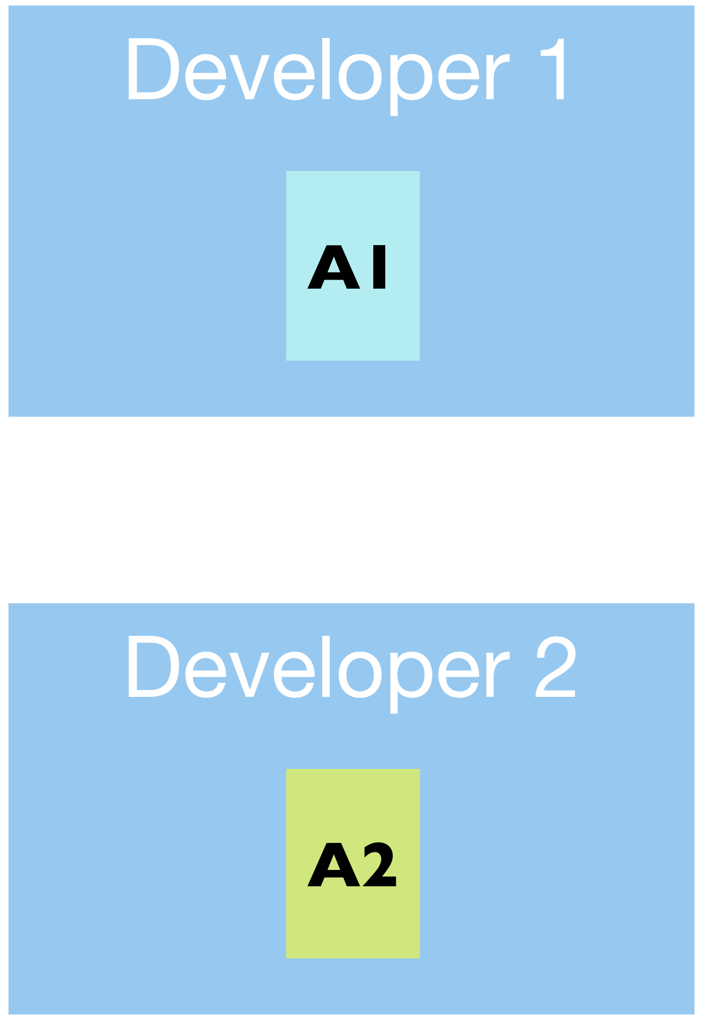


Without a Net (VI)

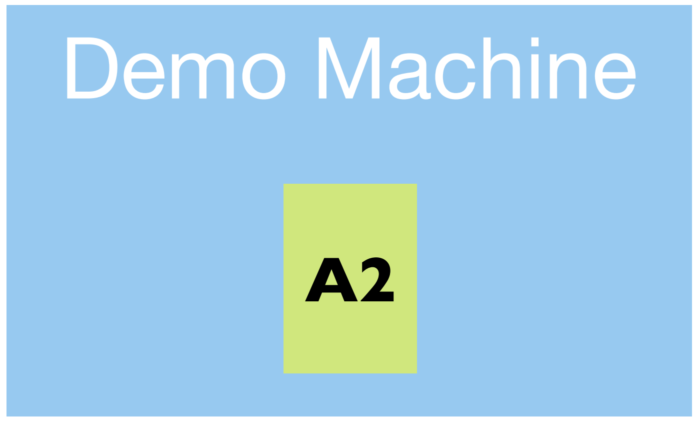
Developer 1 finishes first and uploads his copy to the demo machine.



Without a Net (VII)

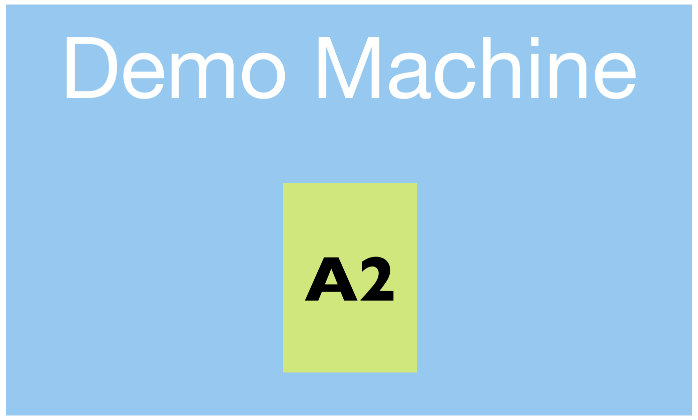


Developer 2 finishes second and uploads his copy to the demo machine.



Without a Net (VIII)

This is known as “last check in wins”

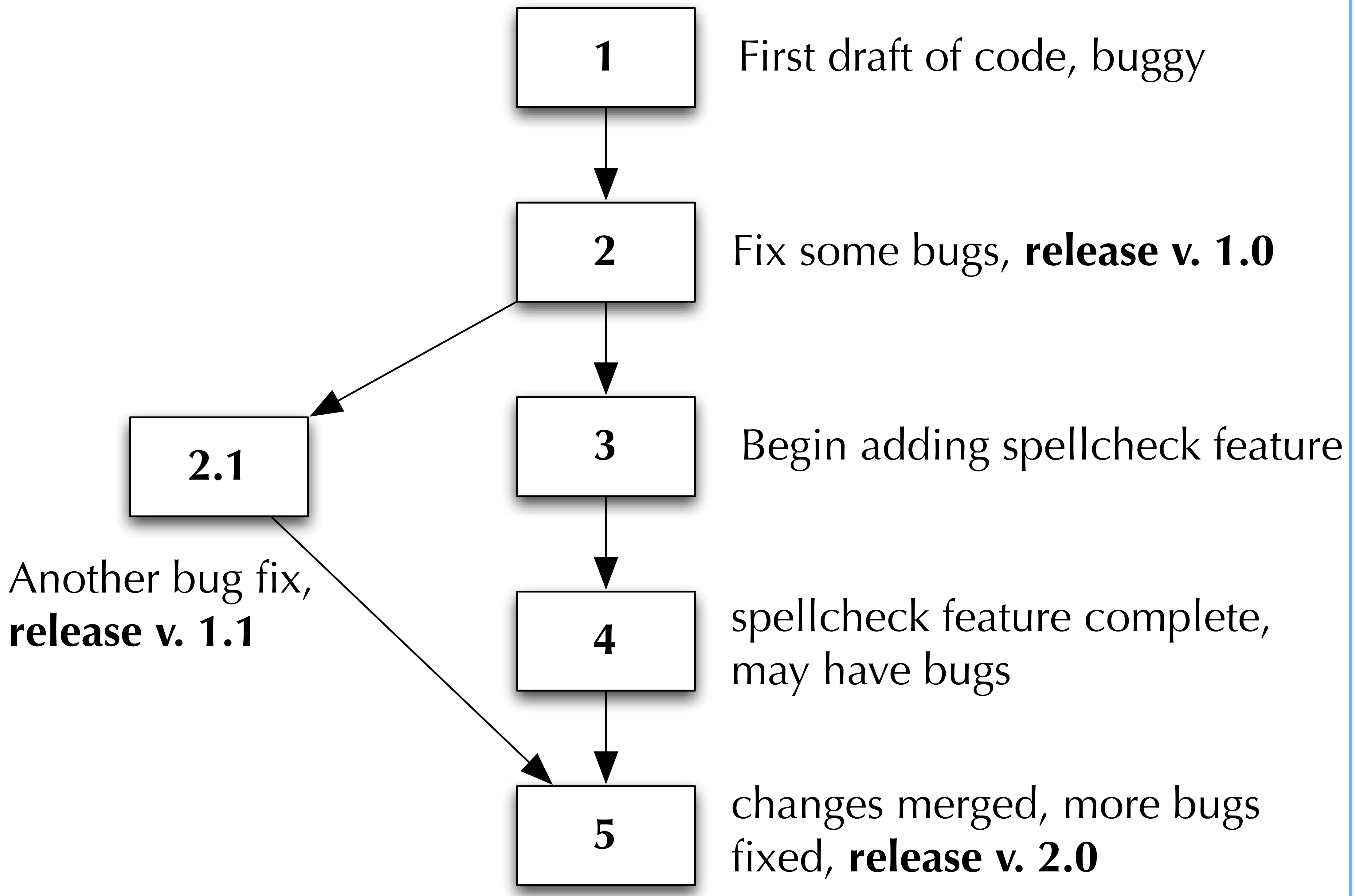


At best, developer 1’s work is simply “gone” when the demo is run; At worst, developer 1 checked in other changes, that cause developer 2’s work to crash when the demo is run.

Not Acceptable

- This type of uncertainty and instability is simply not acceptable in production software environments
  - That’s where configuration management comes in
  - The book uses the term “version control”
    - But in the literature, “version control” is “versioning” applied to a single file while “configuration management” is “versioning” applied to collections of files

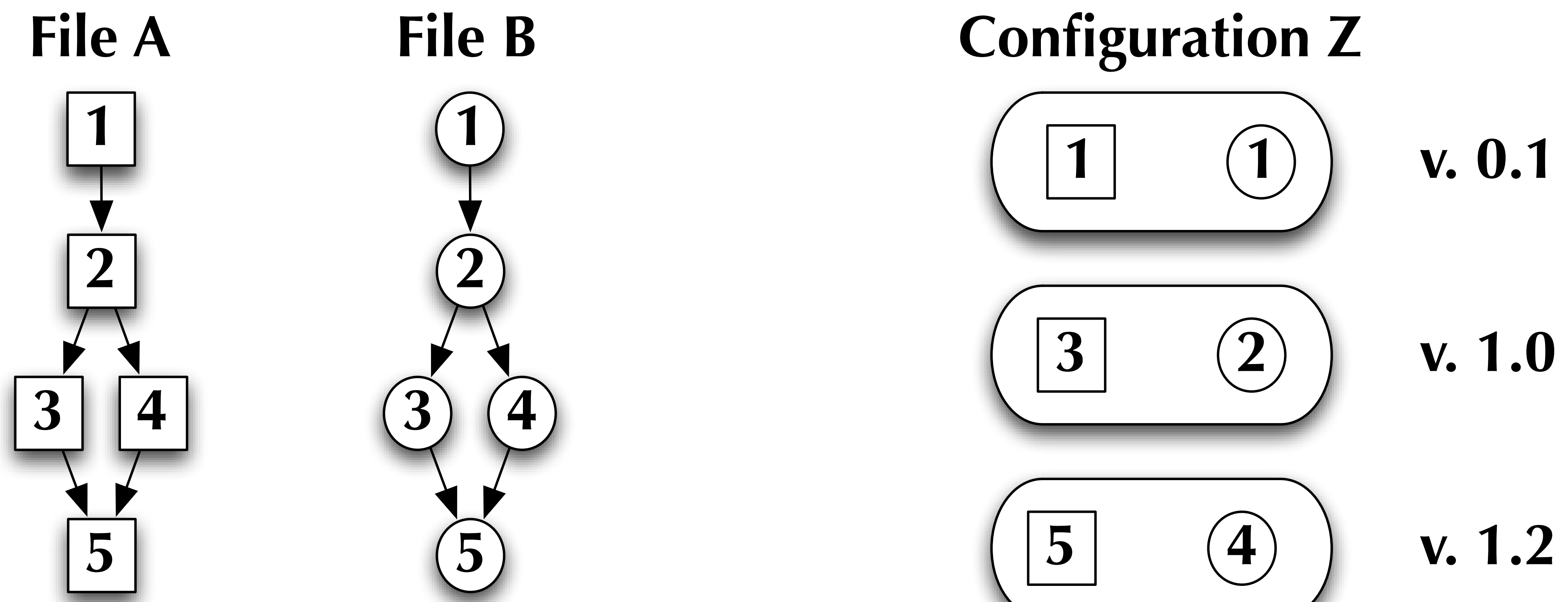
Versioning


1
First draft of code, buggy
2
Fix some bugs, **release v. 1.0**
2.1
Another bug fix, **release v. 1.1**
3
Begin adding spellcheck feature
4
spellcheck feature complete, may have bugs
5
changes merged, more bugs fixed, **release v. 2.0**

Configuration Management

Particular versions of files are included in...

... different versions of a configuration

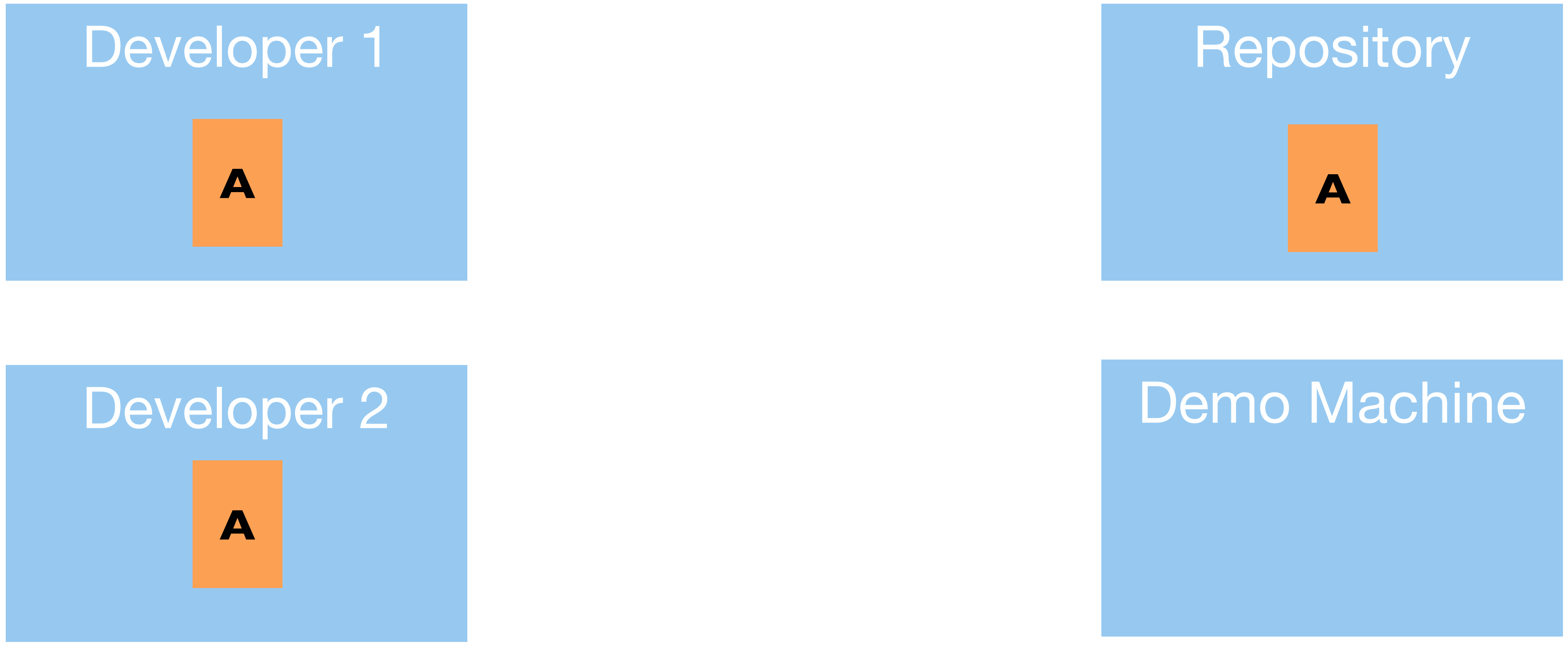


With a Net (I)

Developer 1
Repository
A
Developer 2
Demo Machine

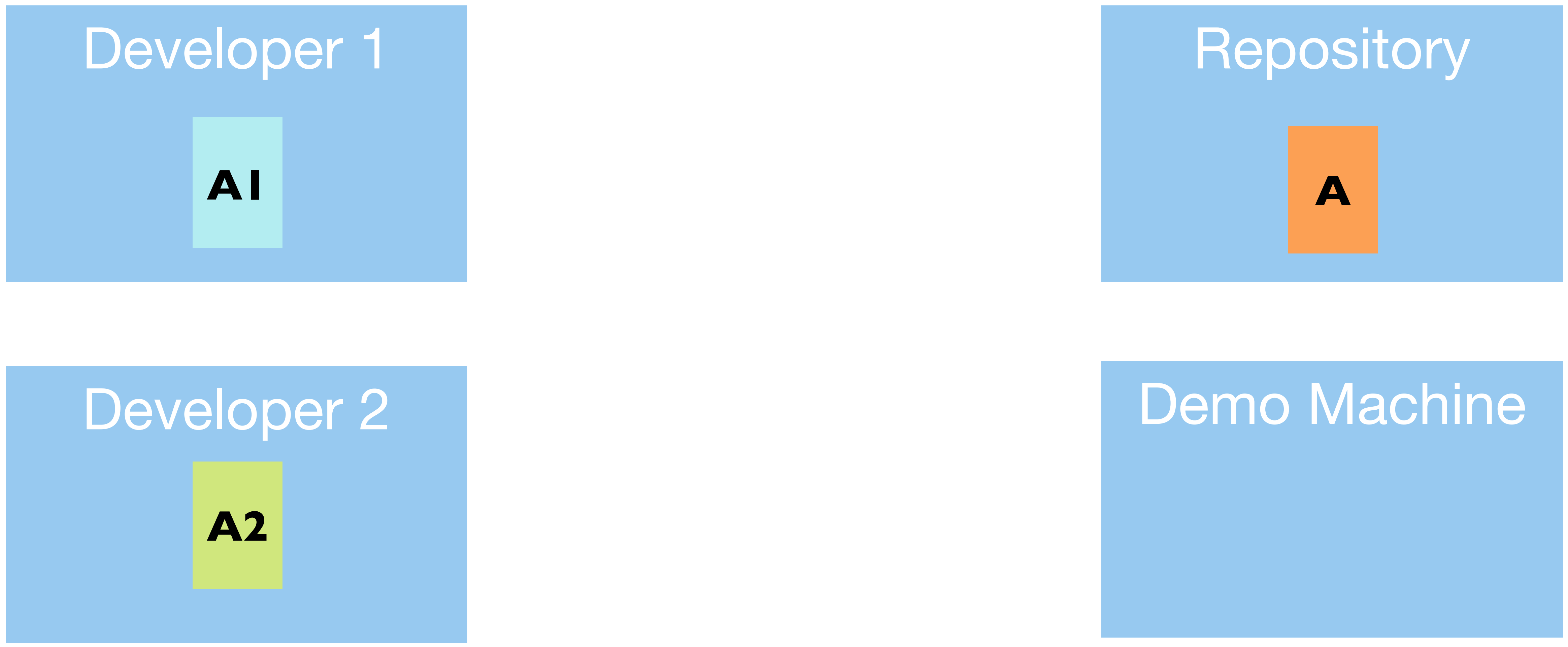
Two developers need to modify the same file for separate tasks

With a Net (II)



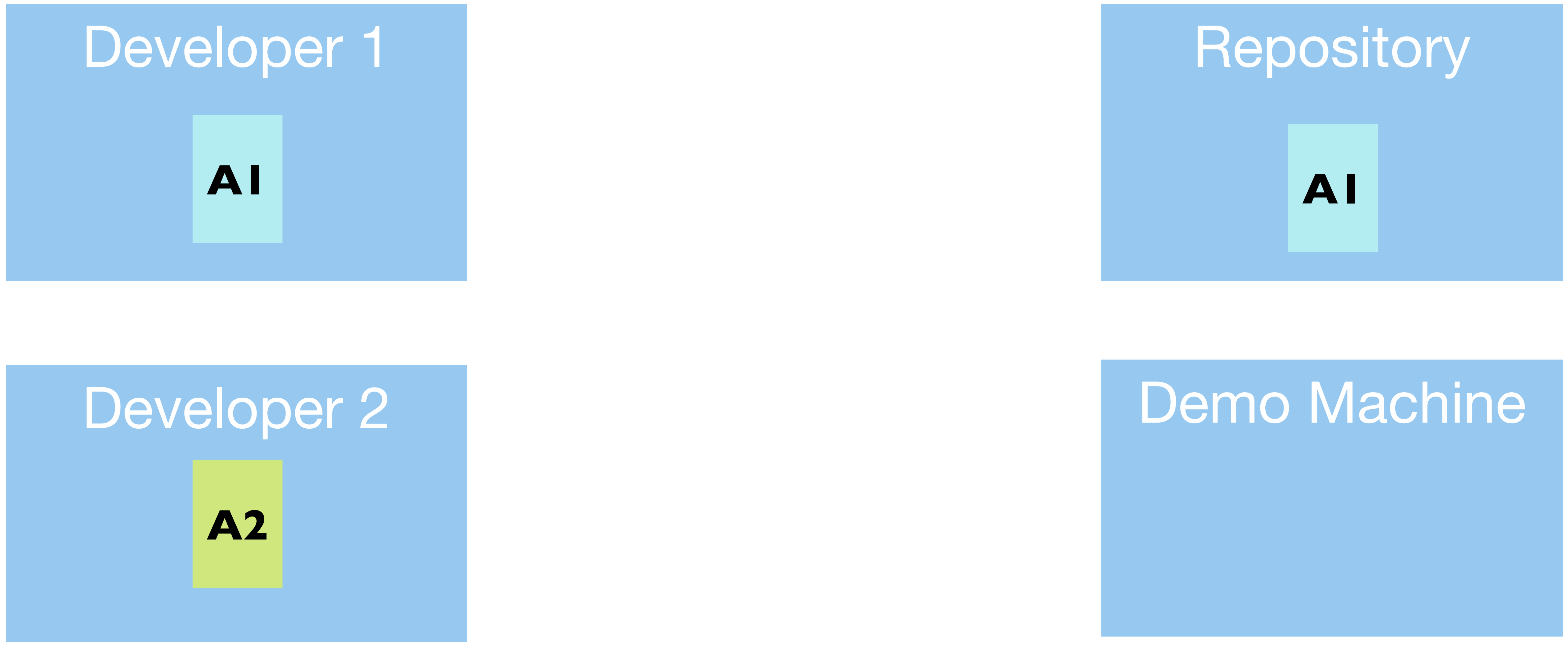
They check the file out into their own working copies

With a Net (III)



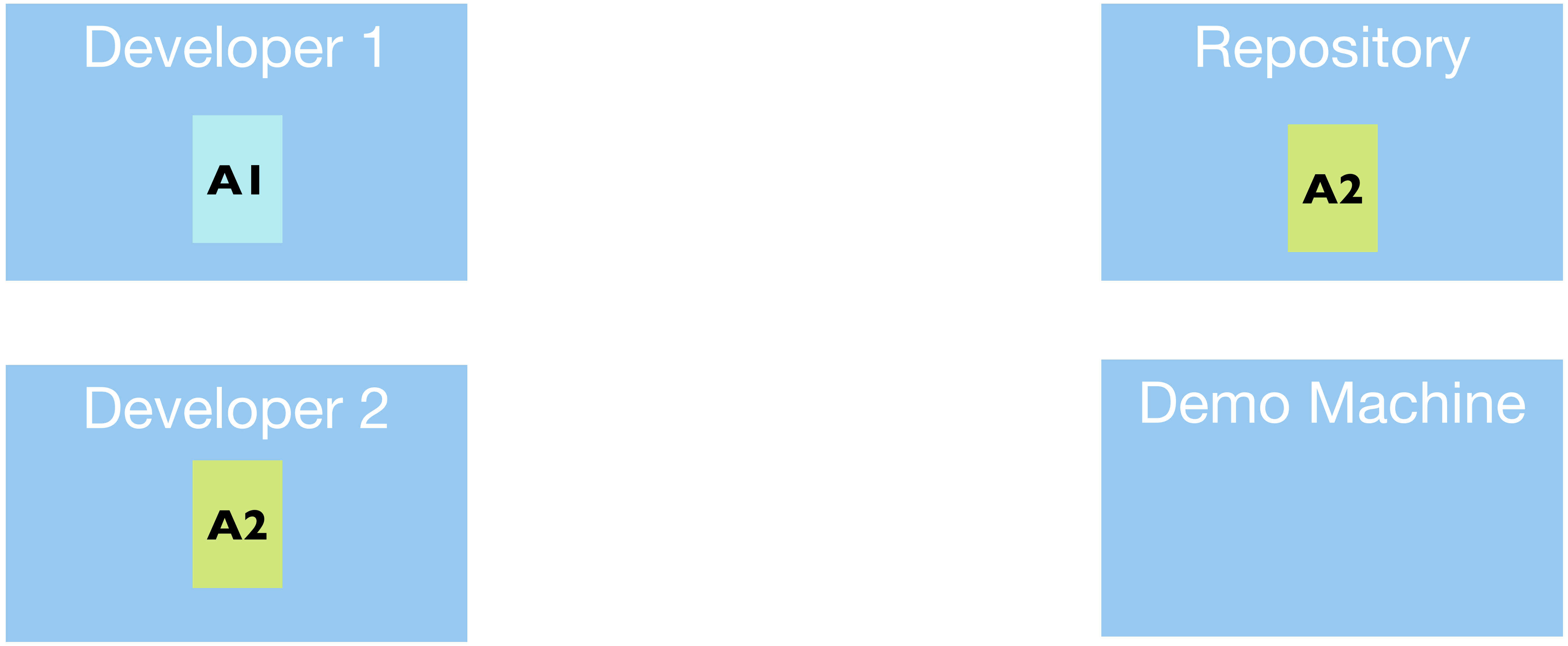
They modify their copies.

With a Net (IV)



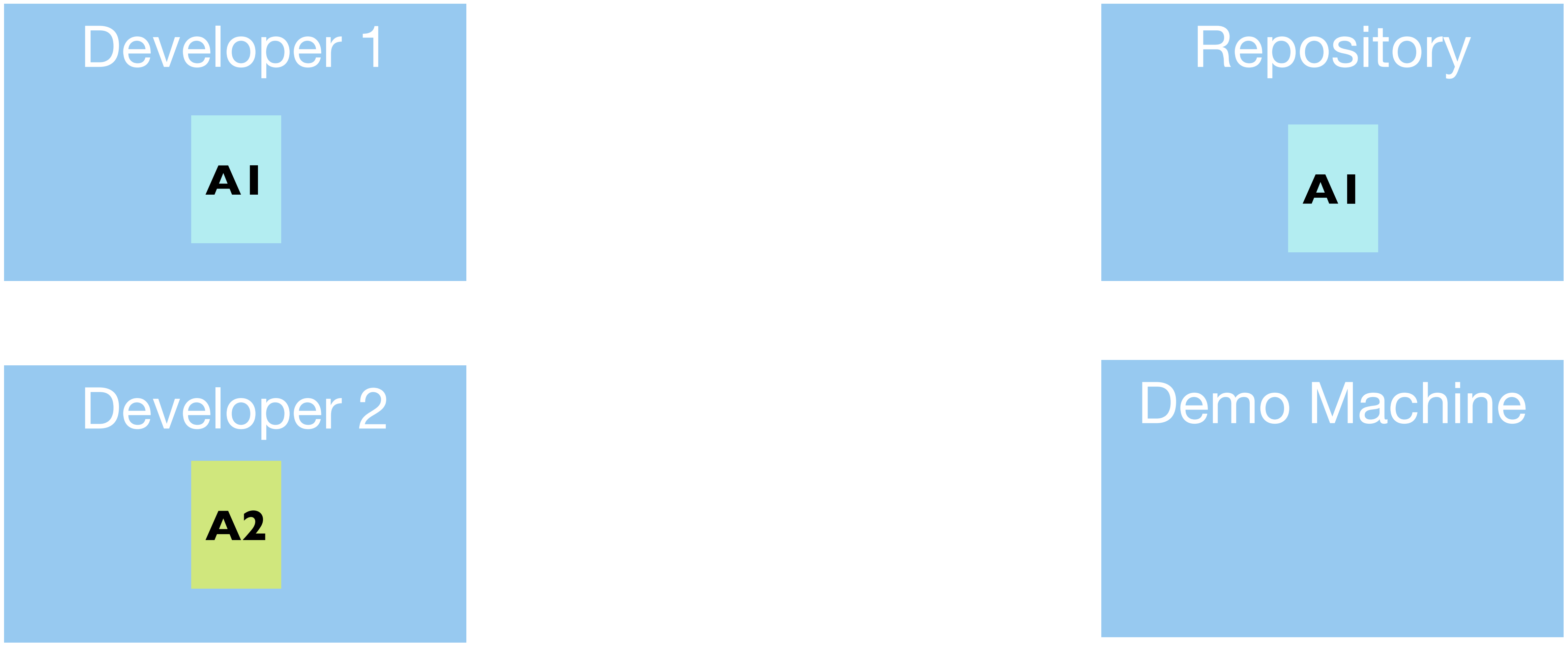
Developer 1 finishes first.

With a Net (V)



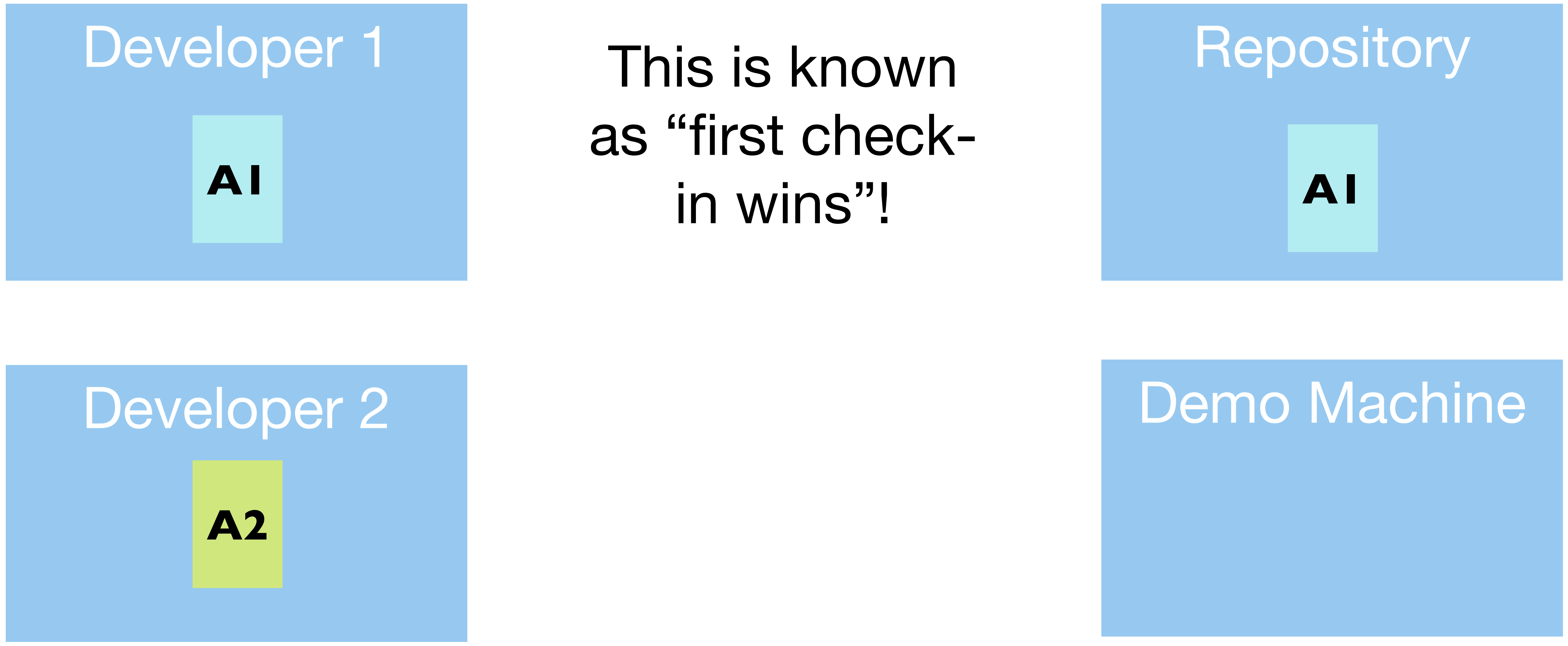
Developer 2 finishes and tries to check in, but...

With a Net (VI)



the change is rejected, because it conflicts with A1

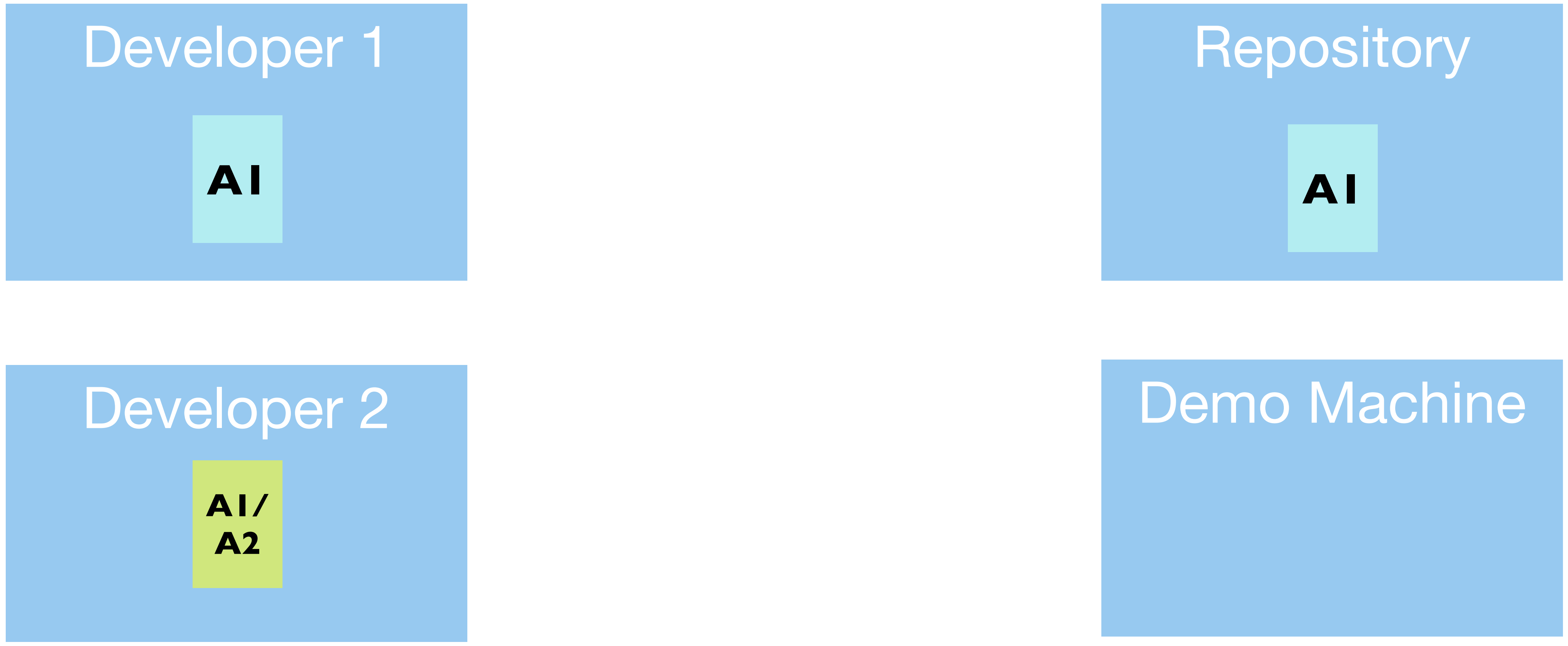
With a Net (VI)



This is known as “first check-in wins”!

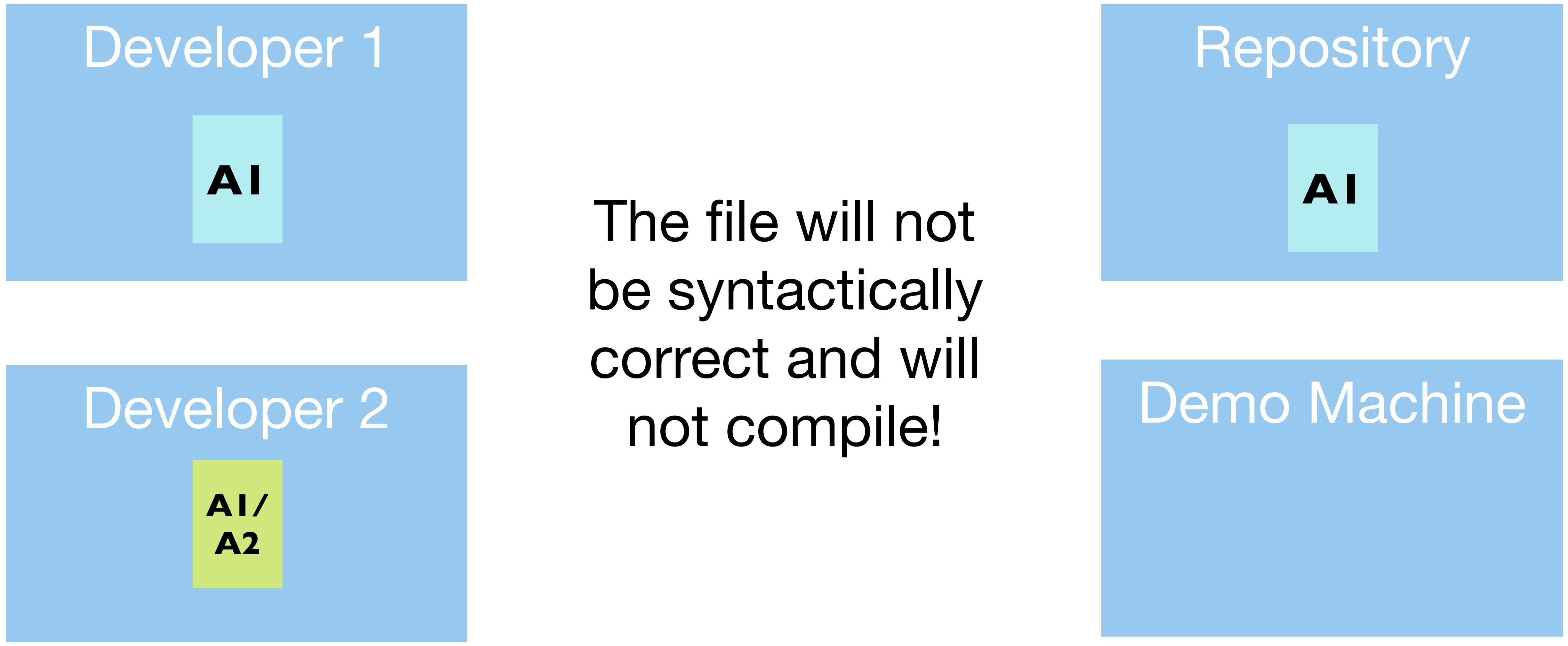
the change is rejected, because it conflicts with A1

With a Net (VII)



What is sent back is an amalgam of A1 and A2's changes

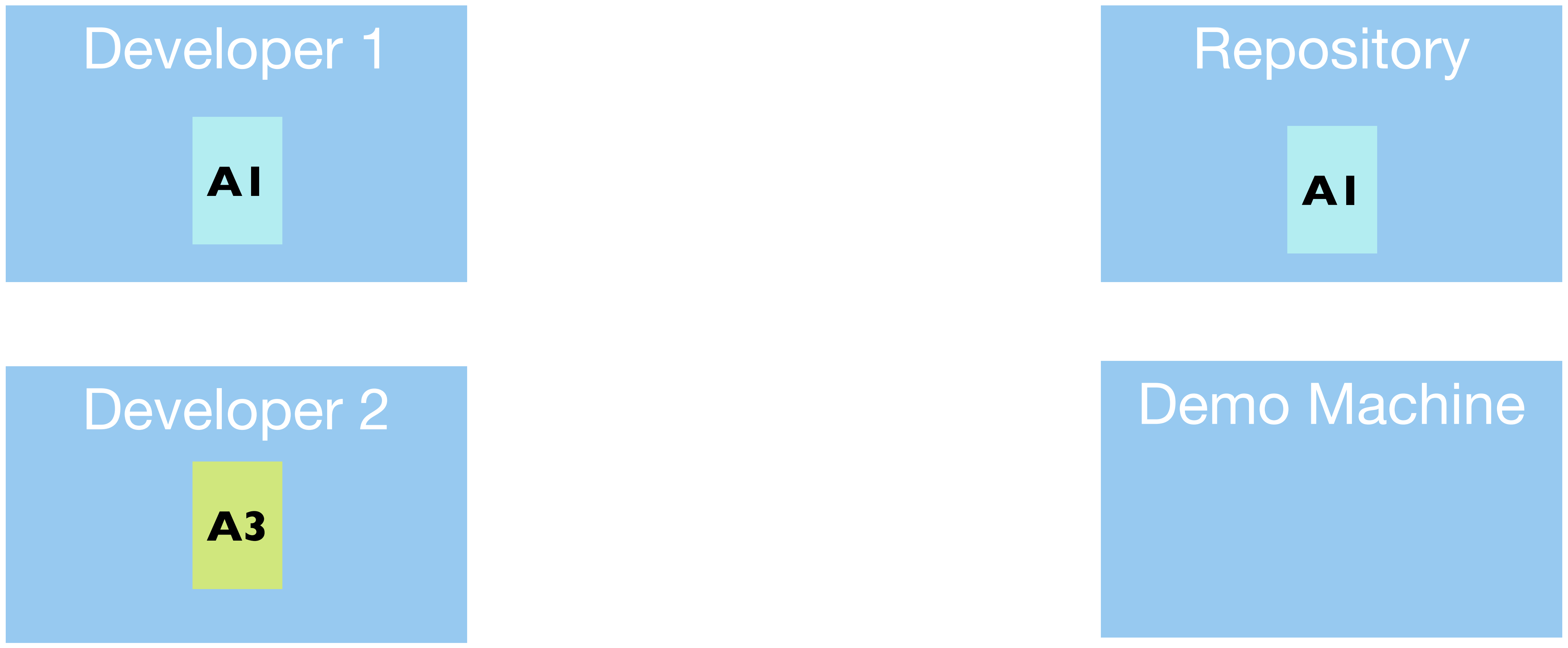
With a Net (VII)



The file will not be syntactically correct and will not compile!

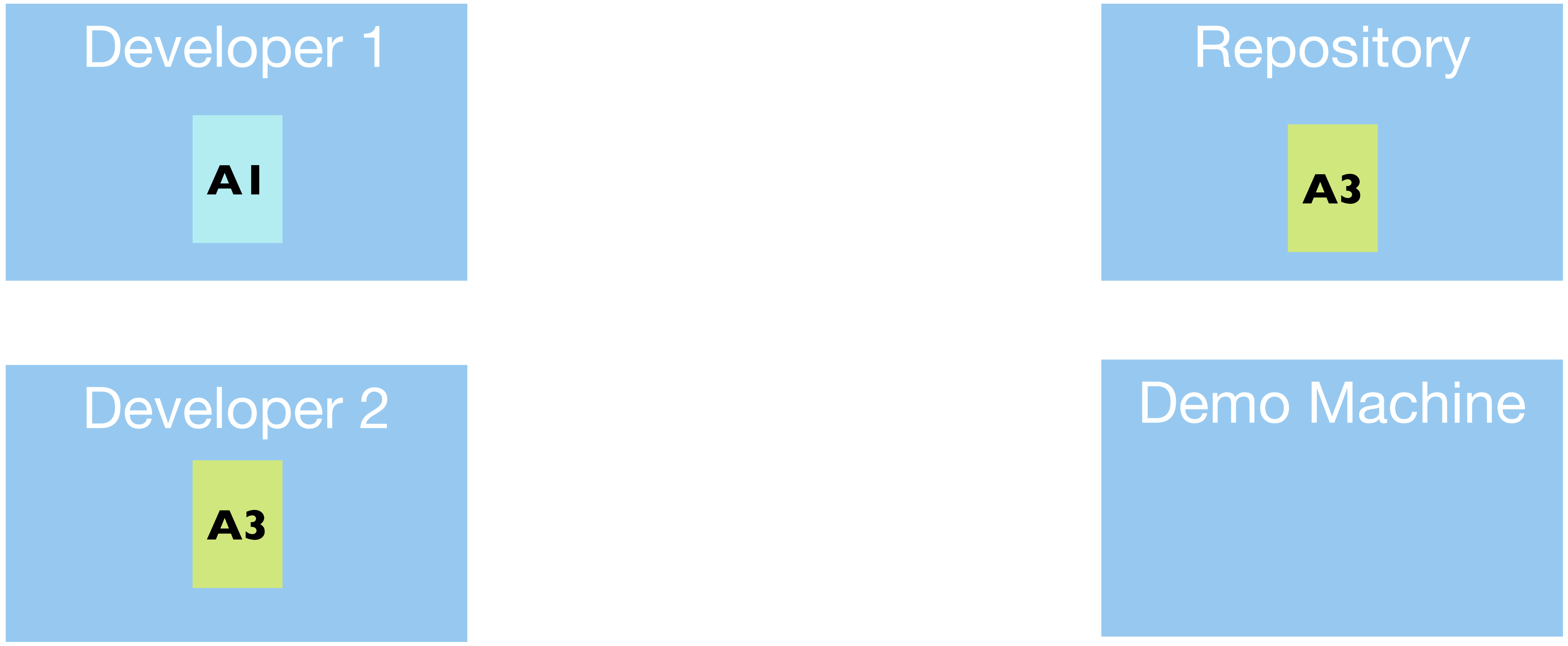
What is sent back is an amalgam of A1 and A2's changes

With a Net (VII)



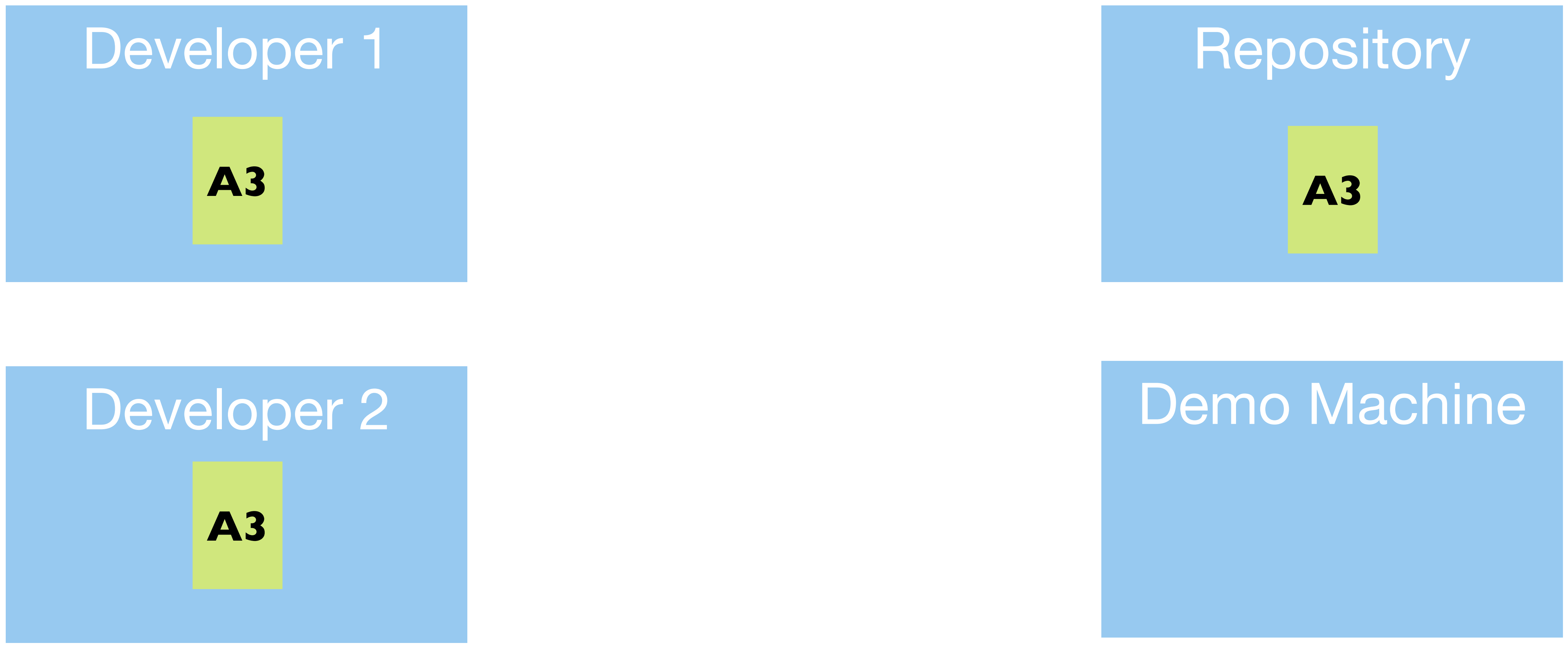
It is up to Developer 2 to merge the changes correctly!

With a Net (VII)



He tells the repository the conflict has been resolved and checks the file in again

With a Net (VII)



Developer 1 can now update his local copy and check the changes on his machine

With a Net (VII)

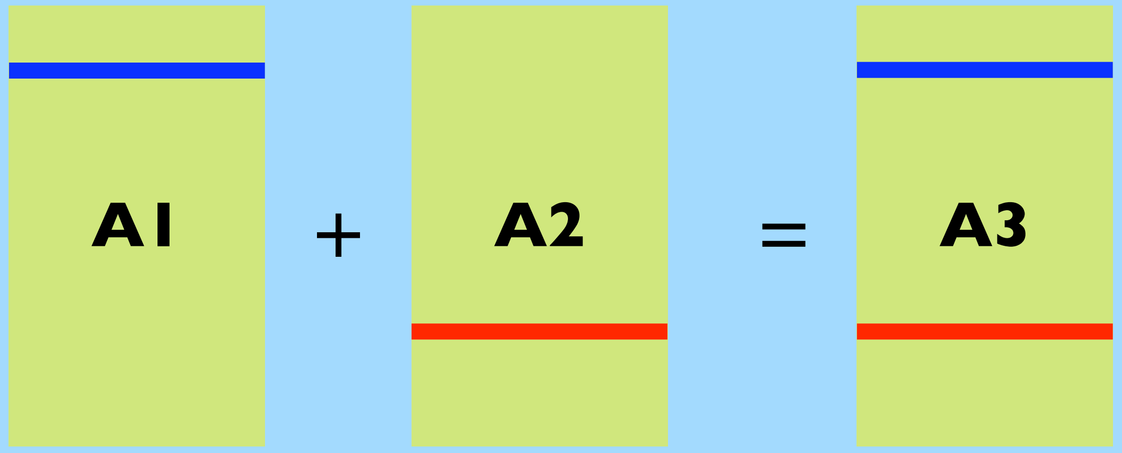
Developer 1
A3
Repository
A3
Developer 2
A3
Demo Machine
A3

When they are both satisfied, the system can be deployed to the demo machine and a successful demo occurs!

Why Multiple Copies?

- Old versioning systems (RCS) did not allow multiple developers to edit a single file at a same time
  - Only one dev. could “lock” the file at a time
- What changed?
  - The assumption that conflicts occur a lot
  - data showed they don’t happen very often!

When two developers edit the same file at the same time, they often make changes to different parts of the file; such changes can easily be merged



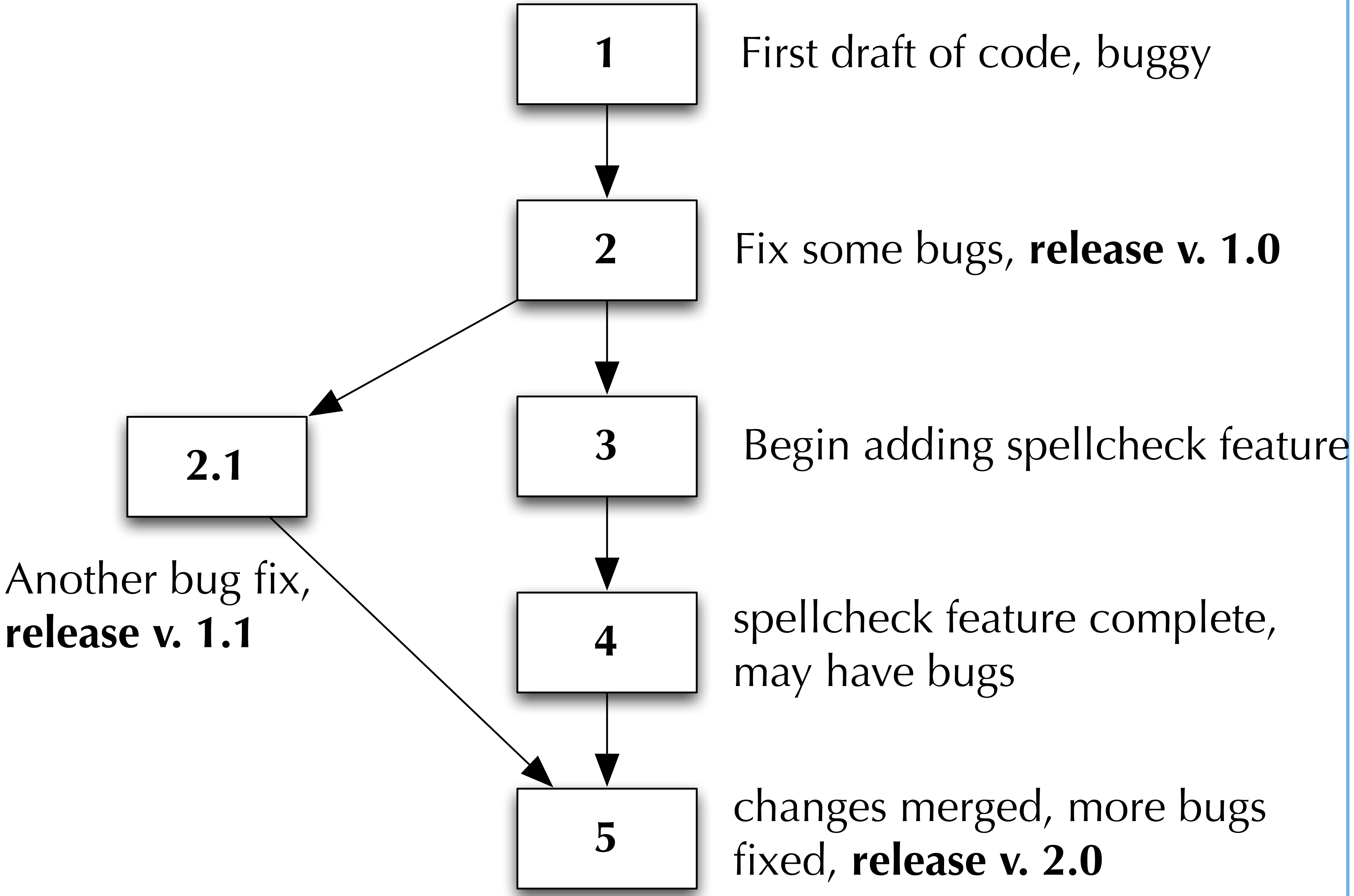
Tags, Branches, and Trunks, Oh My!

- Configuration management systems can handle the basics of checking out the latest version of a system, making changes, and checking the changes back in
  - These changes are committed to what is typically called “the trunk” or main line of development
    - git calls it the “master” branch
- But configuration management systems can do much more than handle changes to the version of a system that is under active development
  - and that’s where tags and branches come in

Scenario (I)

- In the book, a development team has released version 1.0 of a system and has moved on to work on version 2.0
  - they make quite a bit of progress when their customer reports a significant bug with version 1.0
- None of the developers have version 1.0 available on their machines and none of them can remember what version of the repository corresponded to “release 1.0”
  - This highlights the need for good “commit messages”
    - when you are checking in changes be very explicit about what it is you have done; you may need that information later

Remember this diagram? The numbers in boxes are repository versions; the text in bold represent tags


1
First draft of code, buggy
2
Fix some bugs, **release v. 1.0**
2.1
Another bug fix, **release v. 1.1**
3
Begin adding spellcheck feature
4
spellcheck feature complete, may have bugs
5
changes merged, more bugs fixed, **release v. 2.0**

Scenario (II)

- To fix the bug found in version 1.0 of their system, the developers
  - look at the log to locate the version that represented “release 1.0”
  - associate a symbolic name with that version number to “tag it”
    - In this case the tag might be “release_1.0”
  - create a branch that starts at the “release 1.0” tag
  - and fix the bug and commit the changes to the branch
    - They don’t commit to the trunk, since the associated files in the trunk may have changed so much that the patch doesn’t apply
      - once the patch is known, a developer can apply it to the trunk manually at a later point; or use a “merge/fix conflicts” approach

Branches are Cheap

- In any complicated software system, many branches will be created to support
  - bug-fixes
    - e.g. one branch for each official release
  - exploration
    - possibly one branch per developer or one per “risky” feature
      - e.g. switching to a new persistence framework
- Because of this, modern configuration management systems make it easy to create branches

Subversion Branches

- In subversion, tags and branches are made in the same way
  - by creating a copy of the trunk (or any specified revision)
  - the project can be huge, containing thousands of files, and it doesn’t matter, branch/tag creation is completed in constant time and without the size of the repository changing
    - all that subversion does on a copy is note what the copy represents by pointing at the “source” version number

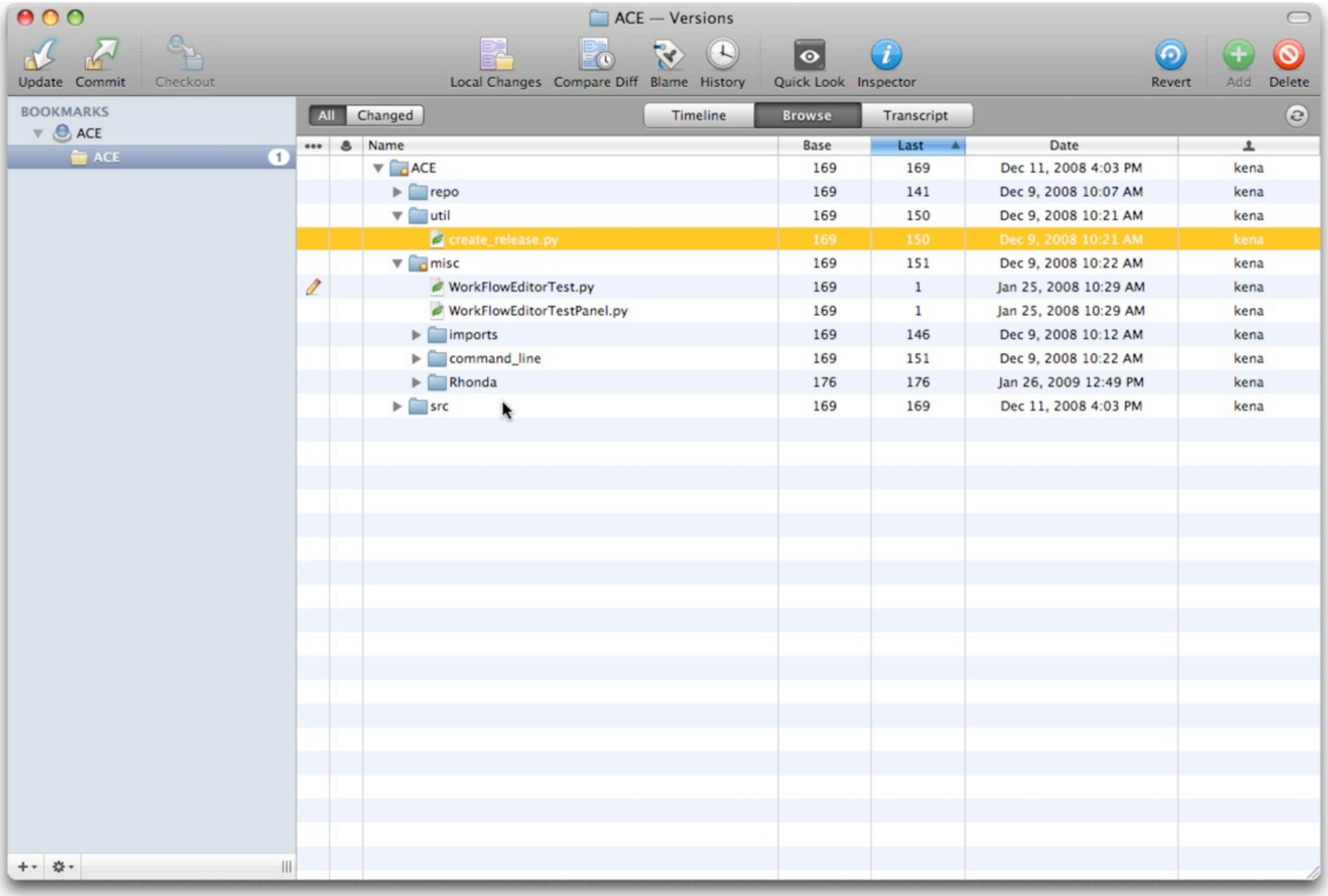
subversion cheat sheet

- Create a new repository
  - svnadmin create <repo>
- Check in new project
  - svn import <dir> <repo>/<project>/trunk
- Check out working copy
  - svn checkout <repo>/<project>/trunk <project>
- Check for updates
  - svn update
- Check in changes
  - svn commit
- Creating a tag
  - svn copy -r <version> <repo>/<project>/trunk <repo>/<project>/tags/<tag>
- Creating a branch
  - svn copy -r <version> <repo>/<project>/trunk <repo>/<project>/branches/<branch>
- tag/branch creation identical!

Many Graphical Tools

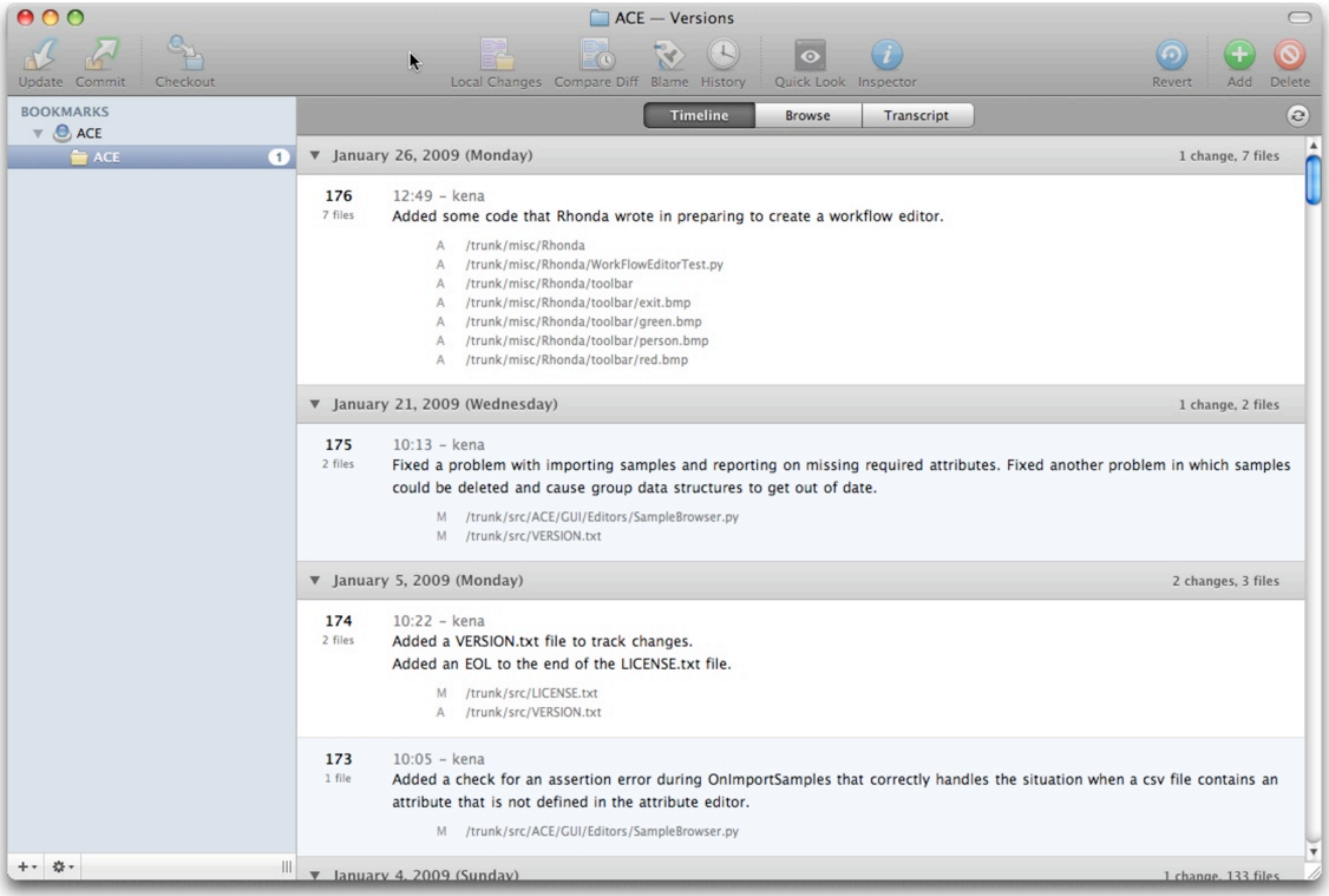
- Standalone Applications
  - Versions <http://versionsapp.com/>
- Integration into Development Environments
  - TextMate <http://macromates.com/>
- These are just examples, both for MacOS X, because that’s my primary platform
  - but there are examples of these tools for multiple platforms

Versions: Browsing Project Files



Name	Base	Last	Date	
ACE	169	169	Dec 11, 2008 4:03 PM	kena
repo	169	141	Dec 9, 2008 10:07 AM	kena
util	169	150	Dec 9, 2008 10:21 AM	kena
create_release.py	169	150	Dec 9, 2008 10:21 AM	kena
misc	169	151	Dec 9, 2008 10:22 AM	kena
WorkFlowEditorTest.py	169	1	Jan 25, 2008 10:29 AM	kena
WorkFlowEditorTestPanel.py	169	1	Jan 25, 2008 10:29 AM	kena
imports	169	146	Dec 9, 2008 10:12 AM	kena
command_line	169	151	Dec 9, 2008 10:22 AM	kena
Rhonda	176	176	Jan 26, 2009 12:49 PM	kena
src	169	169	Dec 11, 2008 4:03 PM	kena

Versions: Viewing Log Messages



▼ January 26, 2009 (Monday) — 1 change, 7 files

176 (7 files) 12:49 - kena
Added some code that Rhonda wrote in preparing to create a workflow editor.

```
A   /trunk/misc/Rhonda
A   /trunk/misc/Rhonda/WorkFlowEditorTest.py
A   /trunk/misc/Rhonda/toolbar
A   /trunk/misc/Rhonda/toolbar/exit.bmp
A   /trunk/misc/Rhonda/toolbar/green.bmp
A   /trunk/misc/Rhonda/toolbar/person.bmp
A   /trunk/misc/Rhonda/toolbar/red.bmp
```

▼ January 21, 2009 (Wednesday) — 1 change, 2 files

175 (2 files) 10:13 - kena
Fixed a problem with importing samples and reporting on missing required attributes. Fixed another problem in which samples could be deleted and cause group data structures to get out of date.

```
M   /trunk/src/ACE/GUI/Editors/SampleBrowser.py
M   /trunk/src/VERSION.txt
```

▼ January 5, 2009 (Monday) — 2 changes, 3 files

174 (2 files) 10:22 - kena
Added a VERSION.txt file to track changes.
Added an EOL to the end of the LICENSE.txt file.

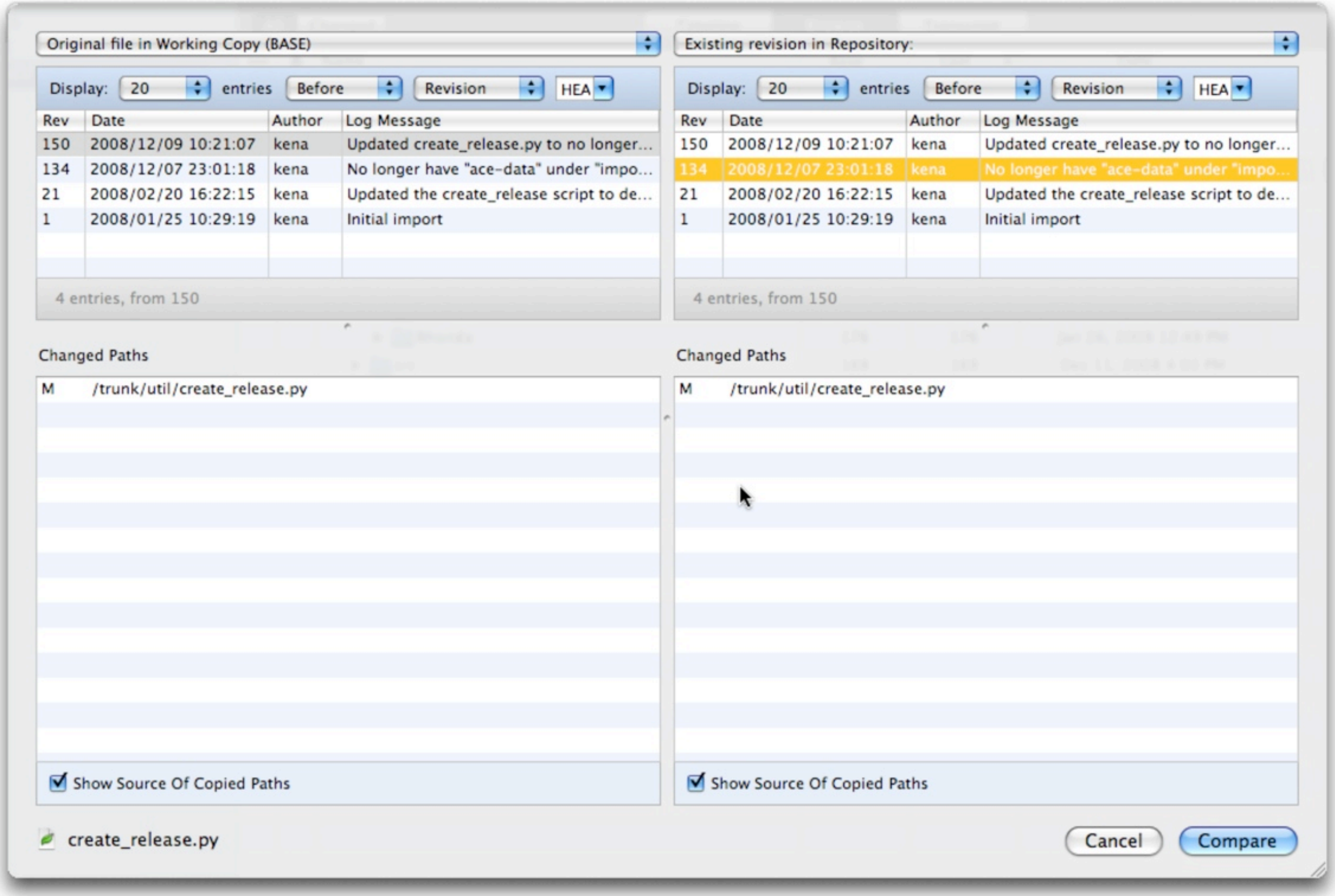
```
M   /trunk/src/LICENSE.txt
A   /trunk/src/VERSION.txt
```

173 (1 file) 10:05 - kena
Added a check for an assertion error during OnImportSamples that correctly handles the situation when a csv file contains an attribute that is not defined in the attribute editor.

```
M   /trunk/src/ACE/GUI/Editors/SampleBrowser.py
```

▼ January 4, 2009 (Sunday) — 1 change, 133 files

Versions: Selecting different versions of a file for comparison



Original file in Working Copy (BASE)

Display: 20 entries Before Revision HEA

Rev	Date	Author	Log Message
150	2008/12/09 10:21:07	kena	Updated create_release.py to no longer...
134	2008/12/07 23:01:18	kena	No longer have "ace-data" under "impo...
21	2008/02/20 16:22:15	kena	Updated the create_release script to de...
1	2008/01/25 10:29:19	kena	Initial import

4 entries, from 150

Changed Paths

M /trunk/util/create_release.py

Show Source Of Copied Paths

Existing revision in Repository:

Display: 20 entries Before Revision HEA

Rev	Date	Author	Log Message
150	2008/12/09 10:21:07	kena	Updated create_release.py to no longer...
134	2008/12/07 23:01:18	kena	No longer have "ace-data" under "impo...
21	2008/02/20 16:22:15	kena	Updated the create_release script to de...
1	2008/01/25 10:29:19	kena	Initial import

4 entries, from 150

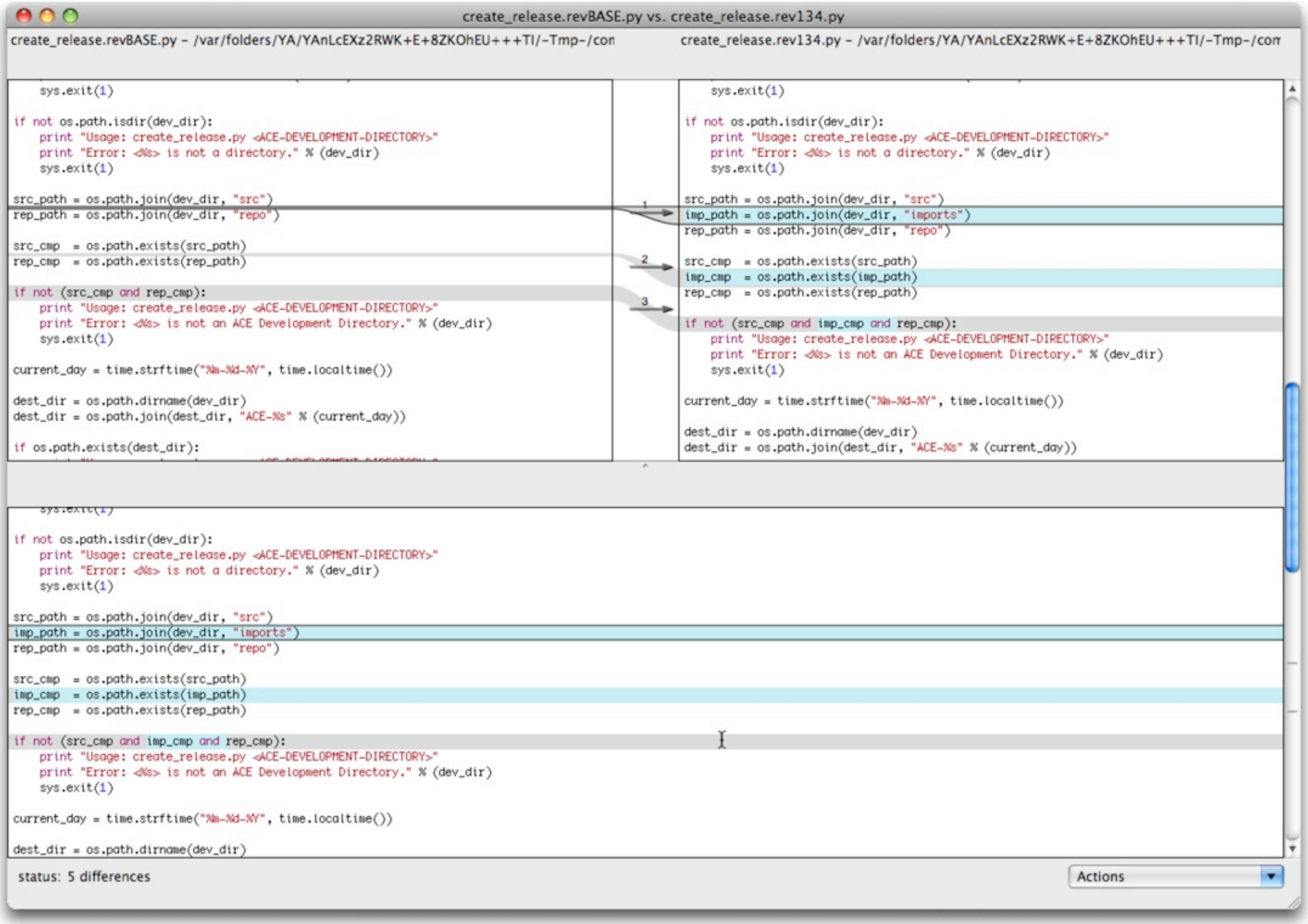
Changed Paths

M /trunk/util/create_release.py

Show Source Of Copied Paths

create_release.py

Versions: Using Apple's FileMerge to see differences



create_release.revBASE.py - /var/folders/YA/YAnLcEXz2RWK+E+8ZKOhEU+++TI/-Tmp-/con

```
    sys.exit(1)

if not os.path.isdir(dev_dir):
    print "Usage: create_release.py <ACE-DEVELOPMENT-DIRECTORY>"
    print "Error: <%s> is not a directory." % (dev_dir)
    sys.exit(1)

src_path = os.path.join(dev_dir, "src")
rep_path = os.path.join(dev_dir, "repo")

src_cmp  = os.path.exists(src_path)
rep_cmp  = os.path.exists(rep_path)

if not (src_cmp and rep_cmp):
    print "Usage: create_release.py <ACE-DEVELOPMENT-DIRECTORY>"
    print "Error: <%s> is not an ACE Development Directory." % (dev_dir)
    sys.exit(1)

current_day = time.strftime("%m-%d-%Y", time.localtime())

dest_dir = os.path.dirname(dev_dir)
dest_dir = os.path.join(dest_dir, "ACE-%s" % (current_day))

if os.path.exists(dest_dir):
```

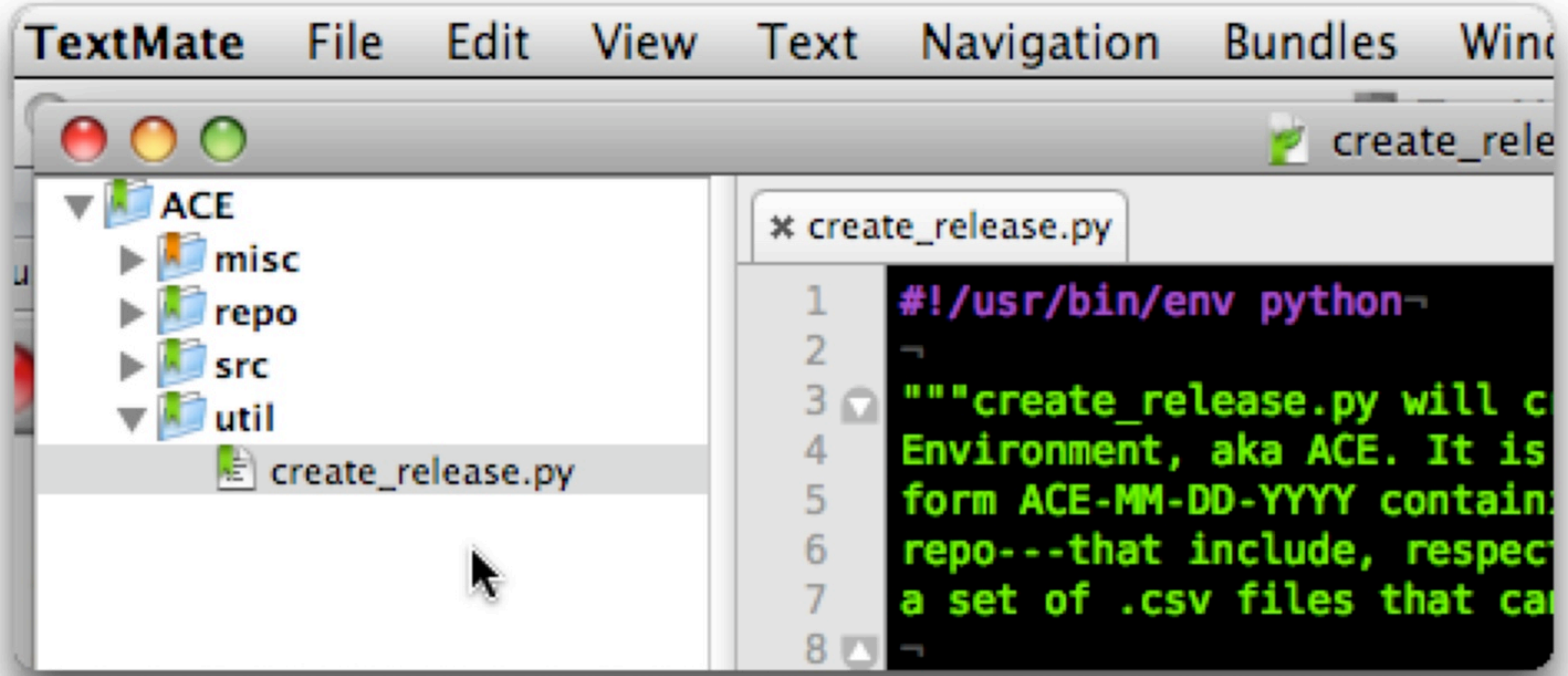
create_release.rev134.py - /var/folders/YA/YAnLcEXz2RWK+E+8ZKOhEU+++TI/-Tmp-/con

```
    sys.exit(1)

if not os.path.isdir(dev_dir):
    print "Usage: create_release.py <ACE-DEVELOPMENT-DIRECTORY>"
    print "Error: <%s> is not a directory." % (dev_dir)
    sys.exit(1)

src_path = os.path.join(dev_dir, "src")
imp_path = os.path.join(dev_dir, "imports")
rep_path = os.path.join(dev_dir, "repo")

src_cmp  = os.path.exists(src_path)
imp_cmp  = os.path.exists(imp_path)
rep_cmp  = os.path.exists(rep_path)

if not (src_cmp and imp_cmp and rep_cmp):
    print "Usage: create_release.py <ACE-DEVELOPMENT-DIRECTORY>"
    print "Error: <%s> is not an ACE Development Directory." % (dev_dir)
    sys.exit(1)

current_day = time.strftime("%m-%d-%Y", time.localtime())

dest_dir = os.path.dirname(dev_dir)
dest_dir = os.path.join(dest_dir, "ACE-%s" % (current_day))
```

```
if not os.path.isdir(dev_dir):
    print "Usage: create_release.py <ACE-DEVELOPMENT-DIRECTORY>"
    print "Error: <%s> is not a directory." % (dev_dir)
    sys.exit(1)

src_path = os.path.join(dev_dir, "src")
imp_path = os.path.join(dev_dir, "imports")
rep_path = os.path.join(dev_dir, "repo")

src_cmp  = os.path.exists(src_path)
imp_cmp  = os.path.exists(imp_path)
rep_cmp  = os.path.exists(rep_path)

if not (src_cmp and imp_cmp and rep_cmp):
    print "Usage: create_release.py <ACE-DEVELOPMENT-DIRECTORY>"
    print "Error: <%s> is not an ACE Development Directory." % (dev_dir)
    sys.exit(1)

current_day = time.strftime("%m-%d-%Y", time.localtime())

dest_dir = os.path.dirname(dev_dir)
```

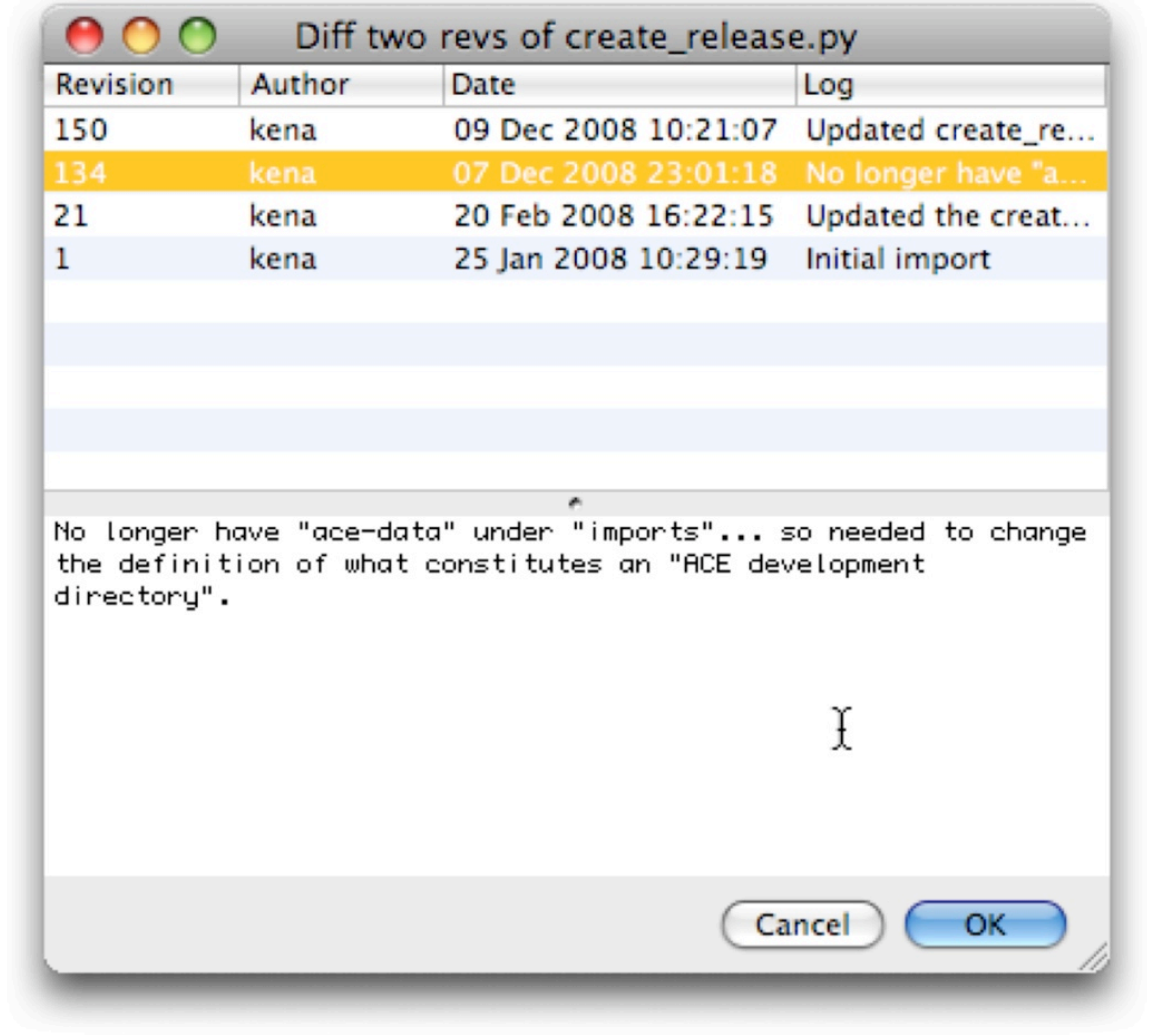
status: 5 differences

TextMate: Showing subversion information on files



```
#!/usr/bin/env python¬
¬
"""create_release.py will c
Environment, aka ACE. It is
form ACE-MM-DD-YYYY contain
repo---that include, respec
a set of .csv files that ca
¬
```

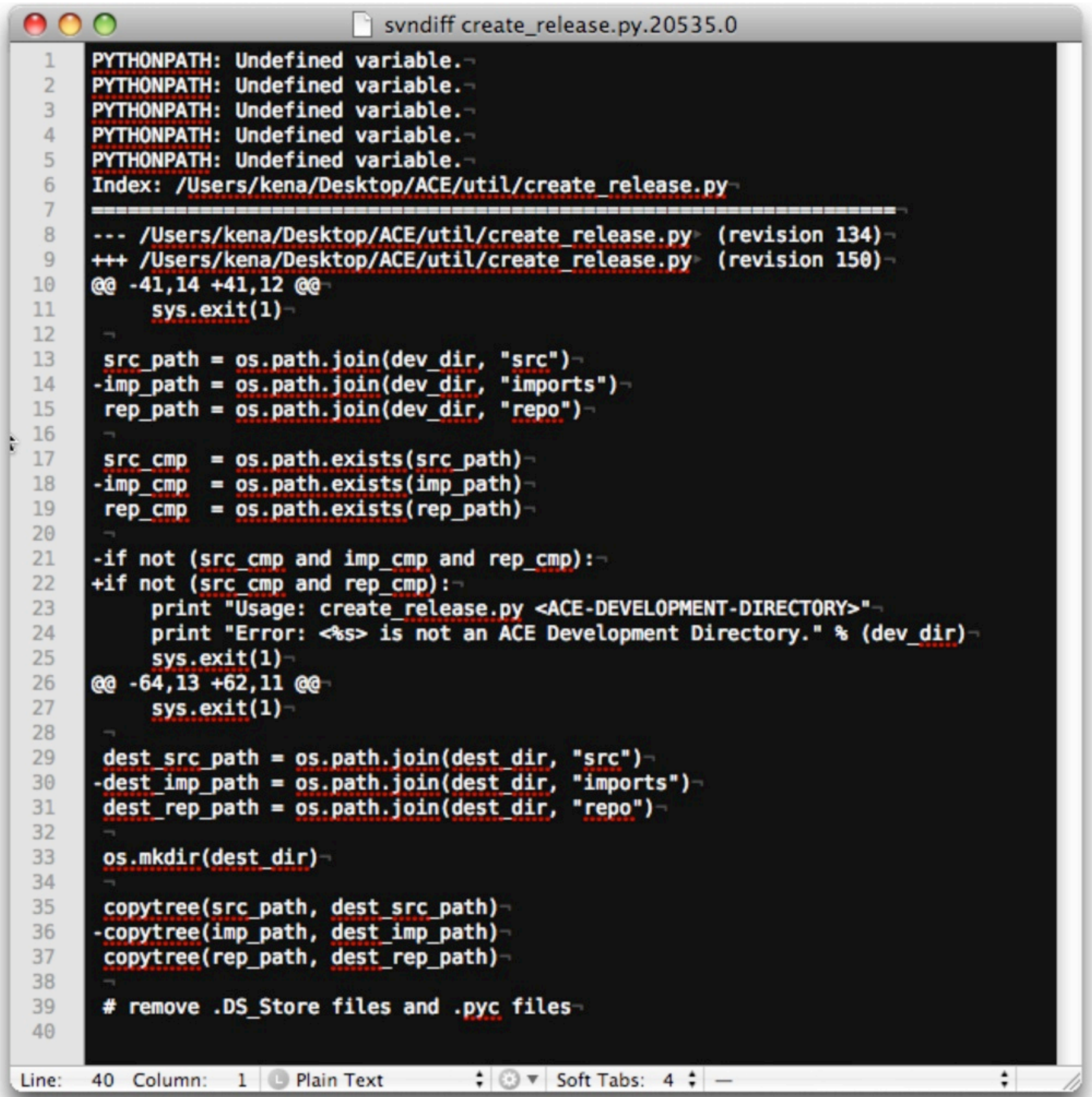
TextMate: Selecting versions of a file for comparison



Revision	Author	Date	Log
150	kena	09 Dec 2008 10:21:07	Updated create_re...
134	kena	07 Dec 2008 23:01:18	No longer have "a...
21	kena	20 Feb 2008 16:22:15	Updated the creat...
1	kena	25 Jan 2008 10:29:19	Initial import

```
No longer have "ace-data" under "imports"... so needed to change
the definition of what constitutes an "ACE development
directory".
```

TextMate: Viewing the differences as a “patch” file



svndiff create_release.py.20535.0

```
PYTHONPATH: Undefined variable.
PYTHONPATH: Undefined variable.
PYTHONPATH: Undefined variable.
PYTHONPATH: Undefined variable.
PYTHONPATH: Undefined variable.
Index: /Users/kena/Desktop/ACE/util/create_release.py
===================================================================
--- /Users/kena/Desktop/ACE/util/create_release.py	(revision 134)
+++ /Users/kena/Desktop/ACE/util/create_release.py	(revision 150)
@@ -41,14 +41,12 @@
     sys.exit(1)
 
 src_path = os.path.join(dev_dir, "src")
-imp_path = os.path.join(dev_dir, "imports")
 rep_path = os.path.join(dev_dir, "repo")
 
 src_cmp  = os.path.exists(src_path)
-imp_cmp  = os.path.exists(imp_path)
 rep_cmp  = os.path.exists(rep_path)
 
-if not (src_cmp and imp_cmp and rep_cmp):
+if not (src_cmp and rep_cmp):
     print "Usage: create_release.py <ACE-DEVELOPMENT-DIRECTORY>"
     print "Error: <%s> is not an ACE Development Directory." % (dev_dir)
     sys.exit(1)
@@ -64,13 +62,11 @@
     sys.exit(1)
 
 dest_src_path = os.path.join(dest_dir, "src")
-dest_imp_path = os.path.join(dest_dir, "imports")
 dest_rep_path = os.path.join(dest_dir, "repo")
 
 os.mkdir(dest_dir)
 
 copytree(src_path, dest_src_path)
-copytree(imp_path, dest_imp_path)
 copytree(rep_path, dest_rep_path)
 
 # remove .DS_Store files and .pyc files
```

Line: 40 Column: 1 | Plain Text | Soft Tabs: 4

I think I like FileMerge a bit better! :-)

svn demo

- Time for a demonstration of subversion in action!

Distributed Configuration Management (I)

- With subversion and cvs (and many others), configuration management depends on an “official” repository
  - There is a notion that somewhere there is a “master copy” and that all working copies are subservient to that copy
- This can be a limiting constraint in large projects with lots of developers; why?
  - so much so that the large project may be tempted to write its own configuration management system just to make progress
    - this is what happened with the Linux project; they produced git because no other configuration management system met their needs!

Distributed Configuration Management (II)

- In distributed configuration management systems, like git, the notion of a centralized repository goes away
  - each and every developer has their own "official" repository
    - with a master branch and any other branches needed by the local developer
  - then other developers can "pull" branches from publicly available git repositories and "push" their changes back to the original repository
- You can learn more about git at the git tutorial
  - <http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html>

git cheat sheet

- Create a new repository
  - git init
- Check in new project
  - git add . ; get commit
- Check out working copy
  - N/A
- Check for updates
  - N/A
- Check in changes
  - git add <file>; git commit
- Creating a tag
  - git tag <tag> <version>
- Creating a branch
  - git branch <branch>
- Collaboration
  - git clone <remote> <local>
  - Update
    - git pull <remote> <branch>
  - Commit
    - git push <remote>

Accidental Difficulties?

- svn
  - adds .svn dir to each directory in your repository
    - if you ever have supporting files stored in a directory of your repository that your application reads, it needs to be aware of the .svn dirs and ignore them
  - single repository version number even in the presence of multiple projects
    - <repo>/<project1>/trunk
    - <repo>/<project2>/trunk
      - Make a change in project 2 and the version number for project 1 is incremented!

Accidental Difficulties?

- git
  - The git FAQ seems to indicate that this tool has its own set of accidental difficulties (you can’t avoid them!)
    - <http://git.or.cz/gitwiki/GitFaq>
  - I just don’t have enough personal experience with git to detail them here.

Wrapping Up

- Version Control & Configuration Management
  - Inject safety and confidence into software development
  - Lots of tools available
    - cvs, svn, git, Mercurial, Visual Source Safe

Coming Up

- Lecture 14: Review for Midterm
- Lecture 15: Midterm
- Lecture 16: Review of Midterm