

15-410

“...Goals: Time Travel, Parallel Universes...”

Source Control
Feb. 16, 2006

Dave Eckhardt

Zach Anderson (15-412, S '03)

Outline

Motivation

Repository vs. Working Directory

Conflicts and Merging

Branching

PRCS – Project Revision Control System

Goals

Working together should be easy

Time travel

- Useful for challenging patents
- **Very** useful for reverting from a sleepless hack session

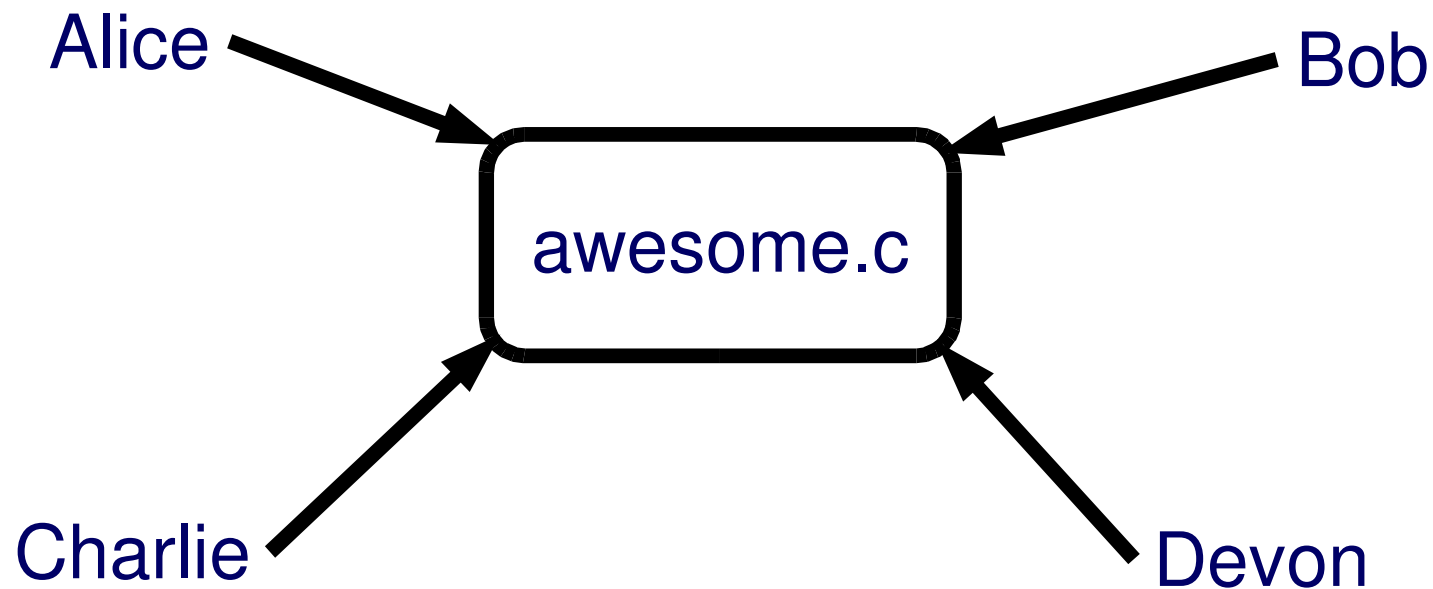
Parallel universes

- Experimental universes
- Product-support universes

Goal: Shared Workspace

Reduce development latency via parallelism

– [But: Brooks, Mythical Man-Month]



Goal: Time Travel

Retrieving old versions should be easy.

Once Upon A Time...

Alice: What happened to the code? It doesn't work.

Charlie: Oh, I made some changes. My code is 1337!

Alice: Rawr! I want the code from last Tuesday!

Goal: Parallel Universes

Safe process for implementing new features.

- Develop bell in one universe
- Develop whistle in another
- Don't inflict B's core dumps on W
- Eventually produce bell-and-whistle release

How?

Keep a global repository for the project.

The Repository

Version

- Contents of some files at a particular point in time
- aka “Snapshot”

Project

- A “sequence” of versions
 - (not really)

Repository

- Directory where projects are stored

The Repository

Stored in group-accessible location

- Old way: file system
- Modern way: “repository server”

Versions *in repository* visible group-wide

- Whoever has read access
- “Commit access” often separate

How?

Keep a global repository for the project.

Each user keeps a working directory.

The Working Directory

Many names (“sandbox”)

Where revisions happen

Typically belongs to *one* user

Versions are *checked out* to here

New versions are *checked in* from here

How?

Keep a global repository for the project.

Each user keeps a working directory.

Concepts of checking out, and checking in

Checking Out. Checking In.

Checking out

- A version is copied from the repository
 - Typically “Check out the latest”
 - Or: “Revision 3.1.4”, “Yesterday noon”

Work

- Edit, add, remove, rename files

Checking in

- Working directory \Rightarrow repository *atomically*
- Result: new version

Checking Out. Checking In.

Repository

Working Directory

○
○
○



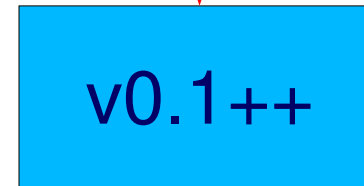
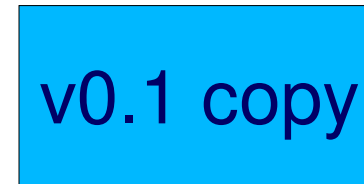
Checking Out. Checking In.

Repository

○
○
○



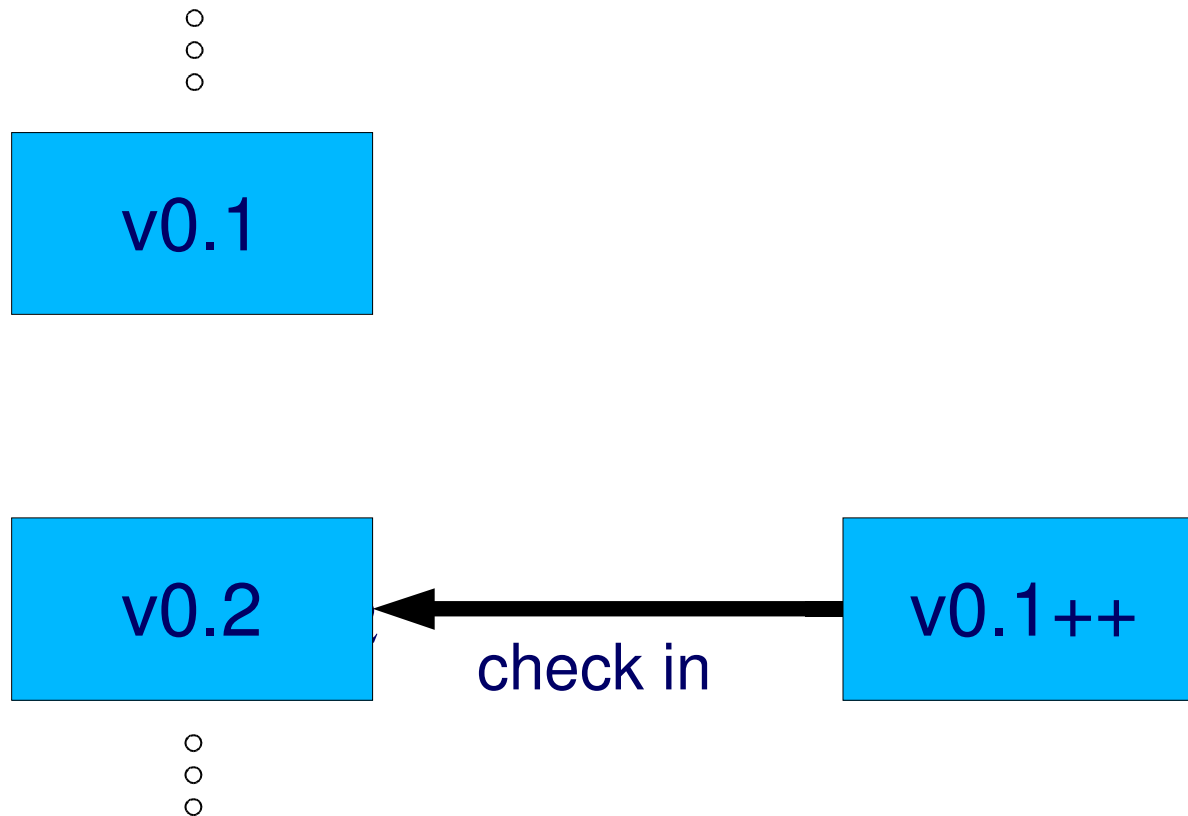
Working Directory



Checking Out. Checking In.

Repository

Working Directory



How?

Keep a global repository for the project.

Each user keeps a working directory.

Concepts of *checking out*, and *checking in*

Mechanisms for merging

Conflicts and Merging

Two people check out.

- Both modify `foo.c`

Each wants to check in a new version.

- Whose is the *correct* new version?

Conflicts and Merging

Conflict

- Independent changes which “overlap”
- *Textual* overlap detected by revision control
- *Semantic* conflict cannot be

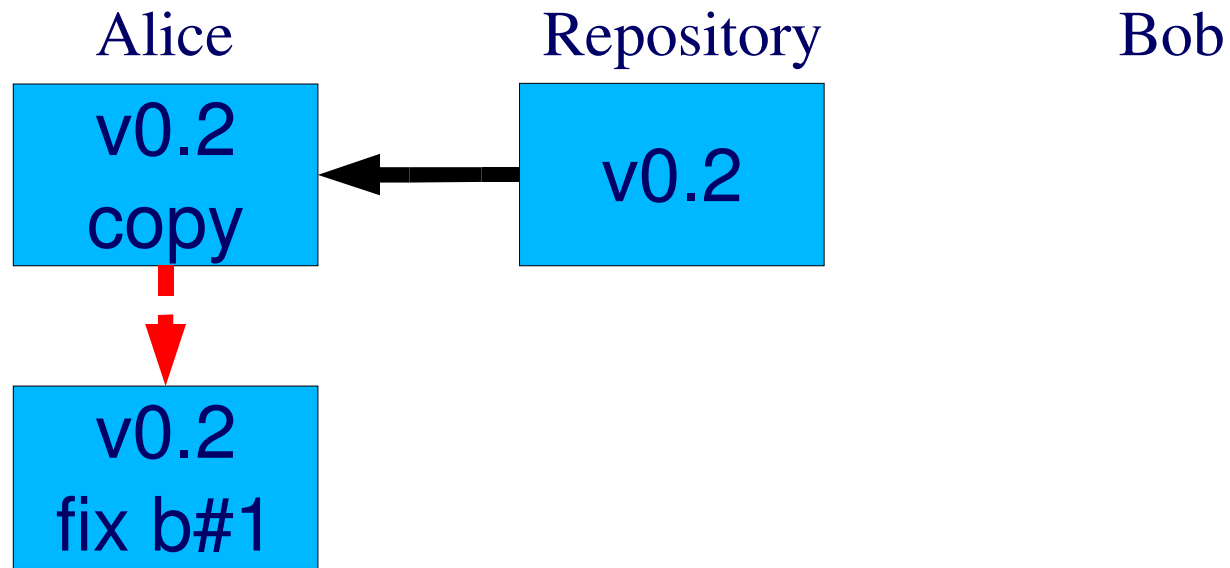
Merge displays conflicting updates per file

Pick which code goes into the new version

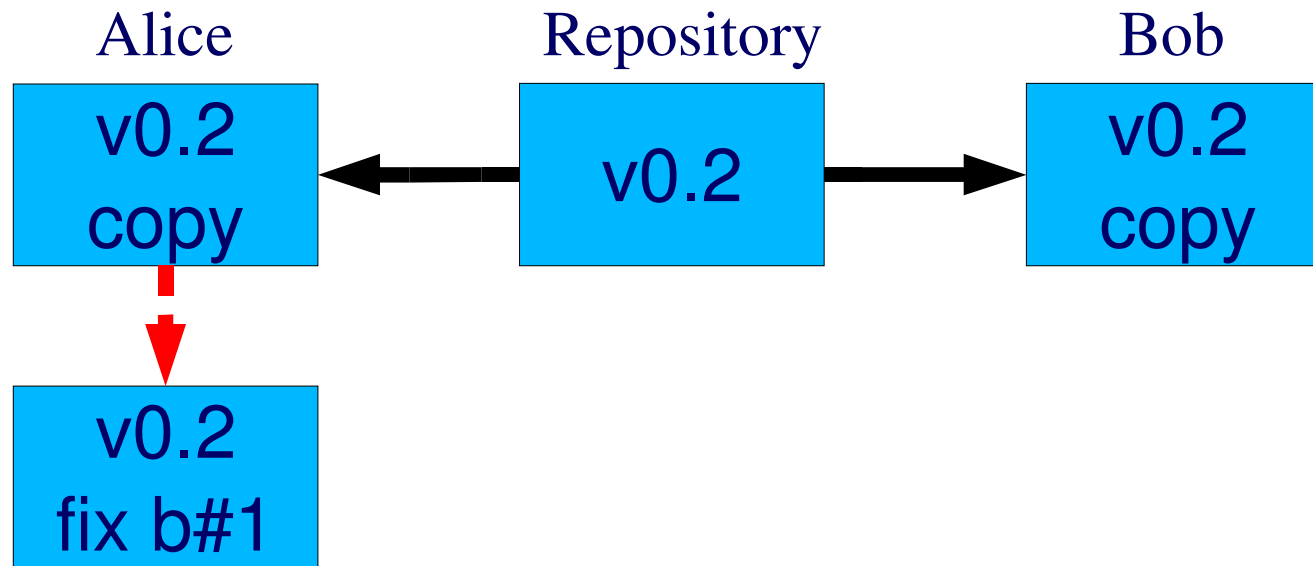
- A, B, NOTA

- 19 - Story now, real-life example later

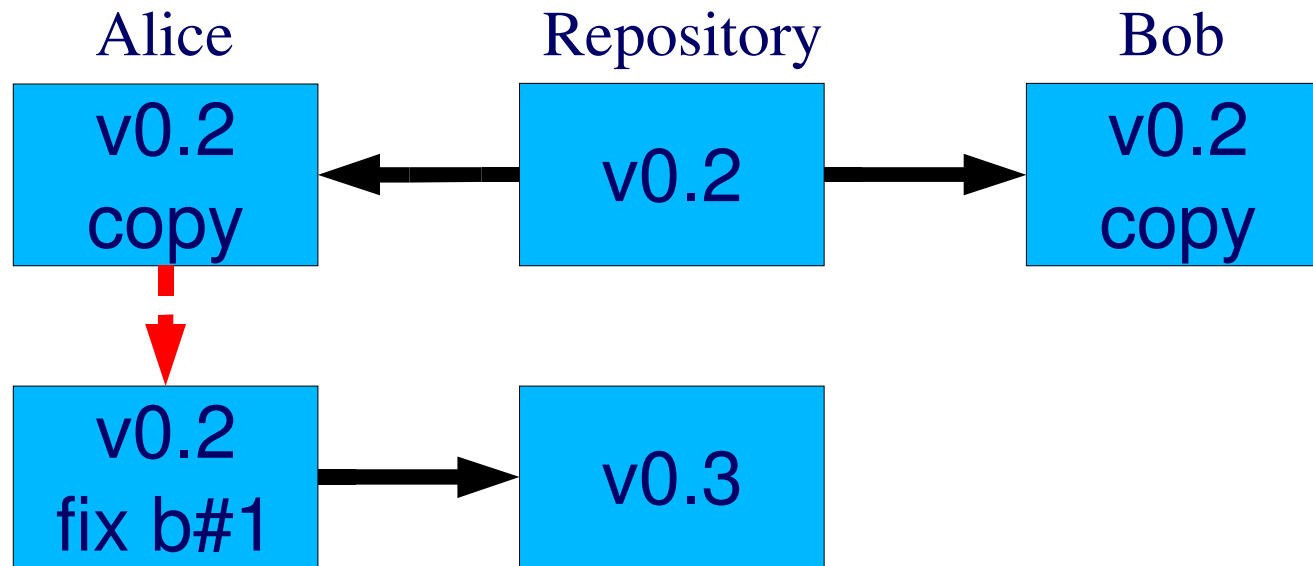
Alice Begins Work



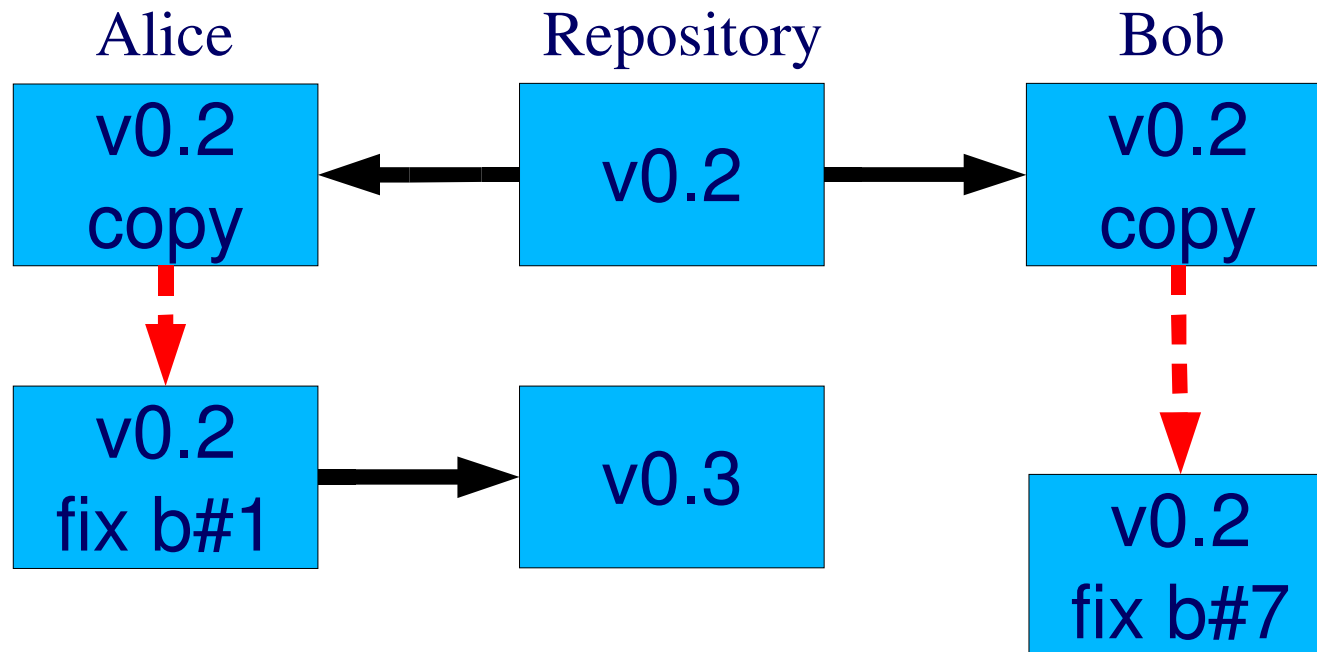
Bob Arrives, Checks Out



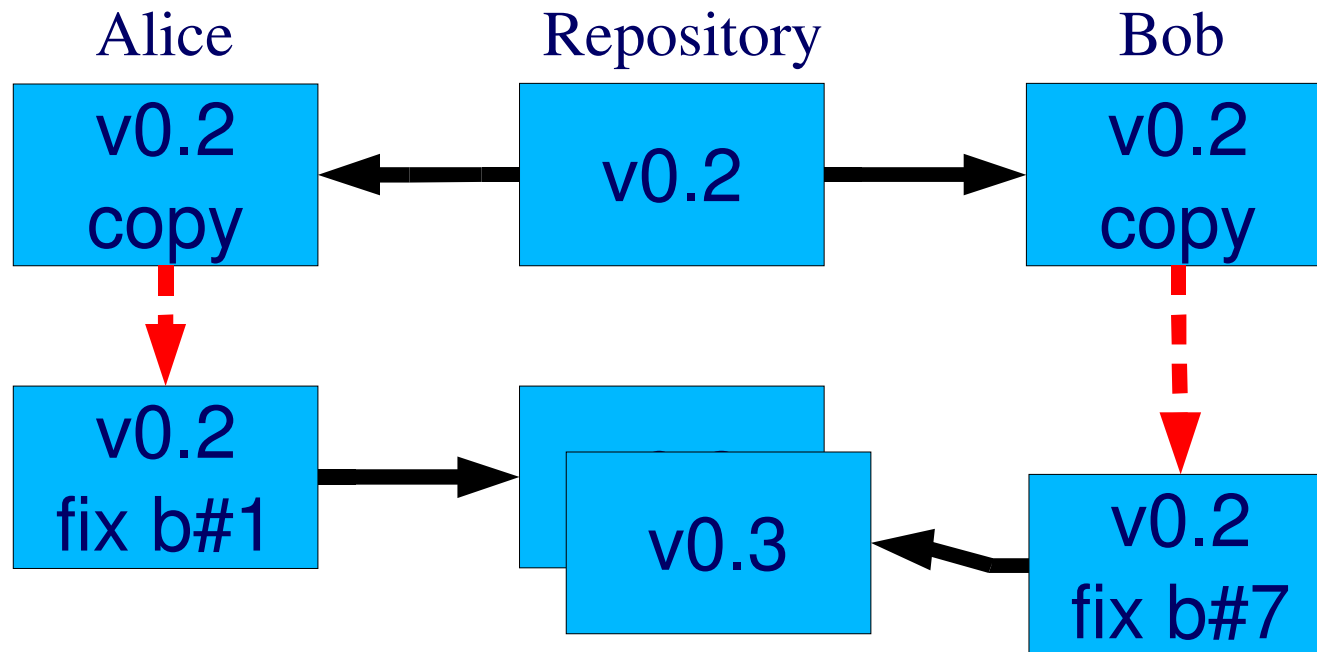
Alice Commits, Bob Has Coffee



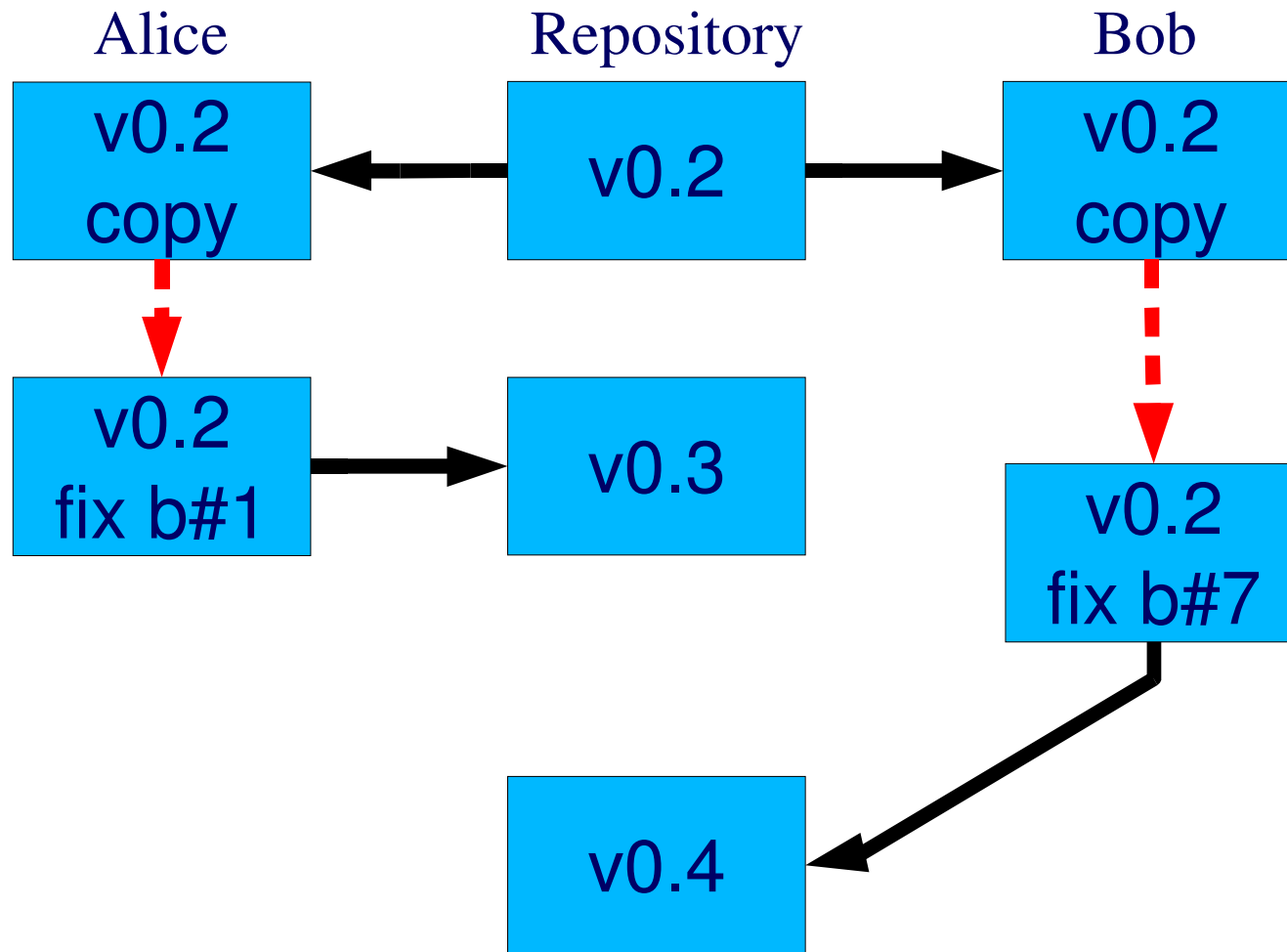
Bob Fixes Something Too



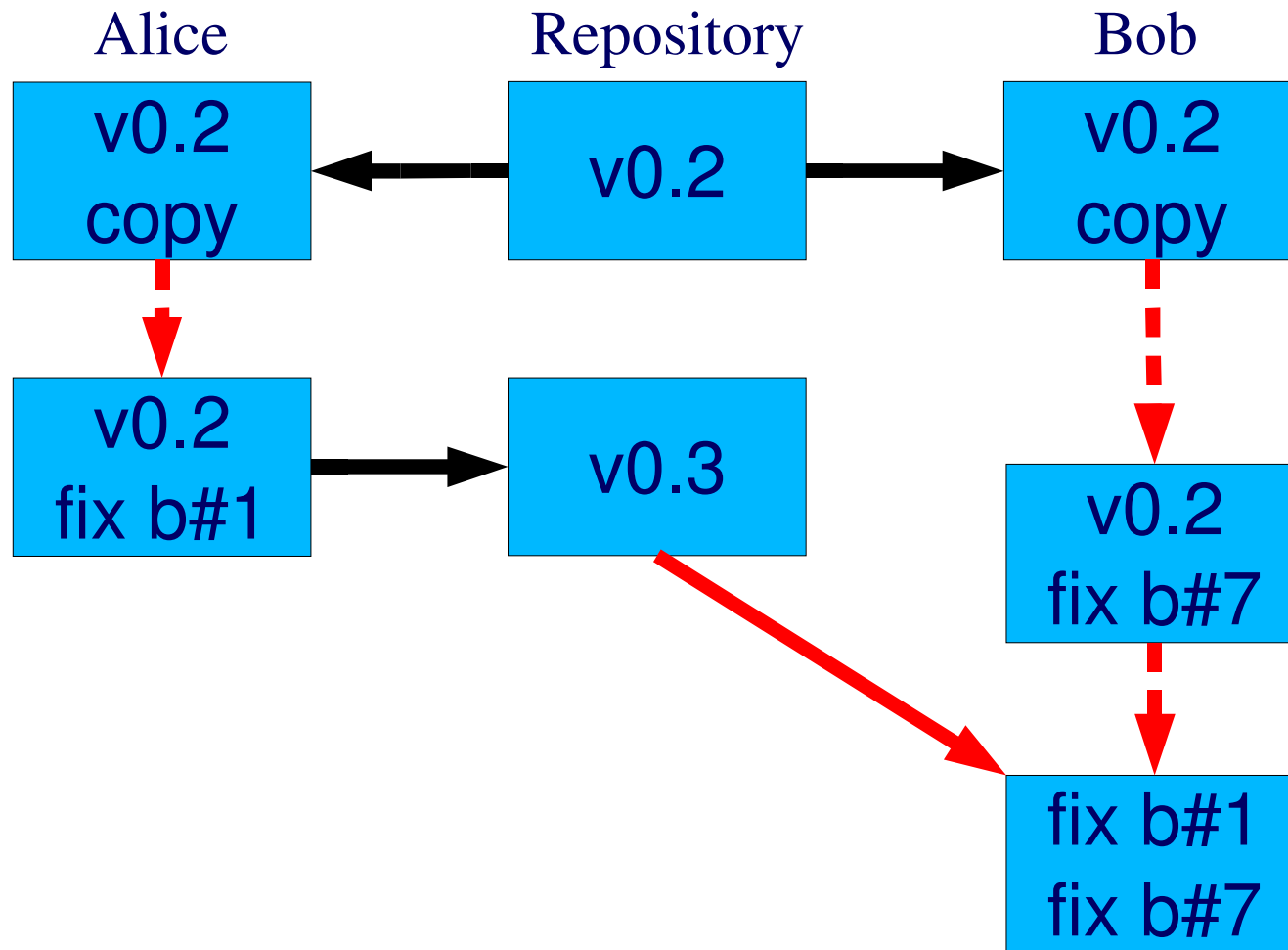
Wrong Outcome



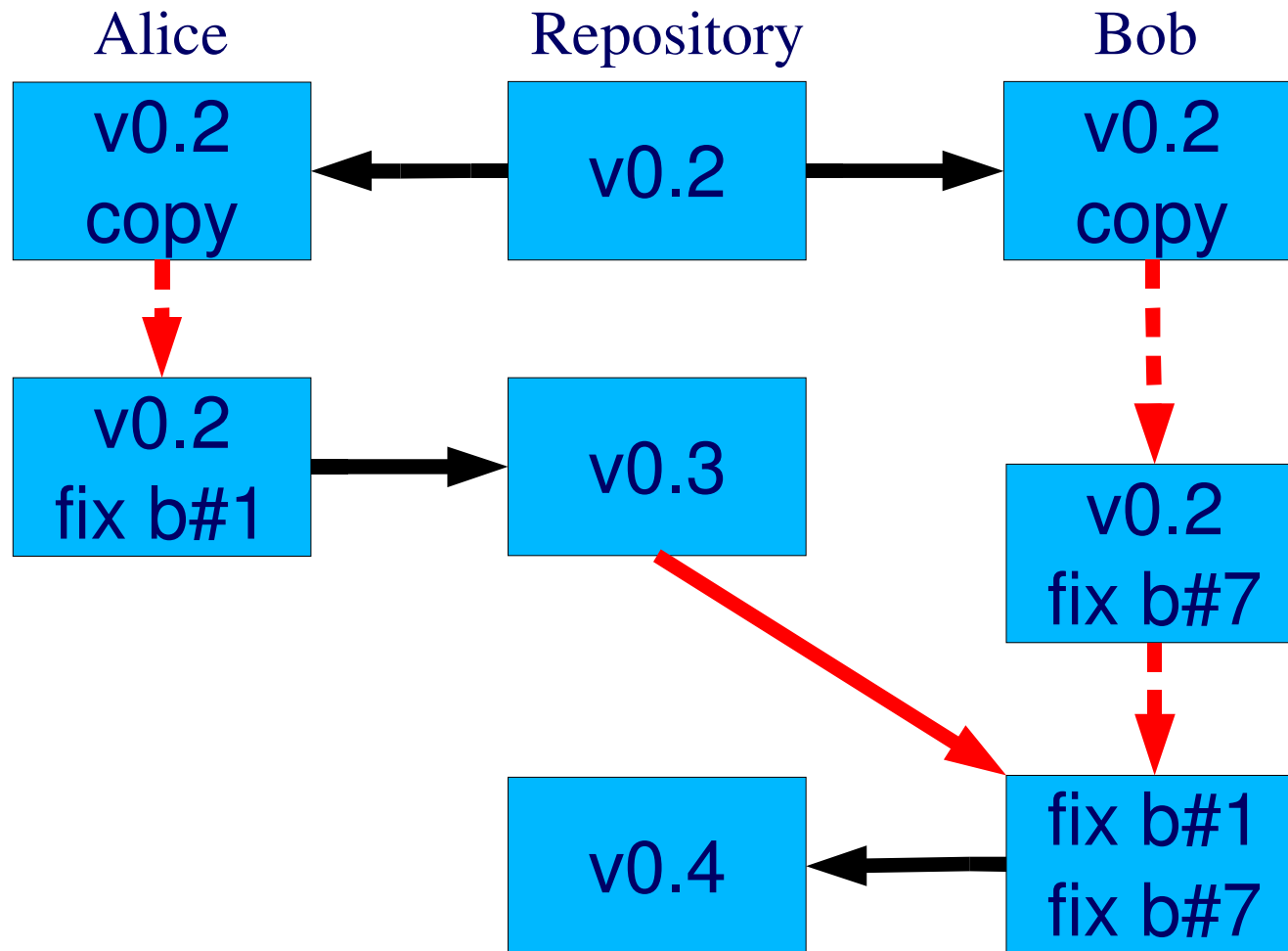
“Arguably Less Wrong”



Merge, Bob, Merge!



Committing Genuine Progress



How?

Keep a global repository for the project.

Each user keeps a working directory.

Concepts of *checking out*, and *checking in*

Mechanisms for *merging*

Mechanisms for branching

Branching

A branch is a *sequence of versions*

- (not really...)

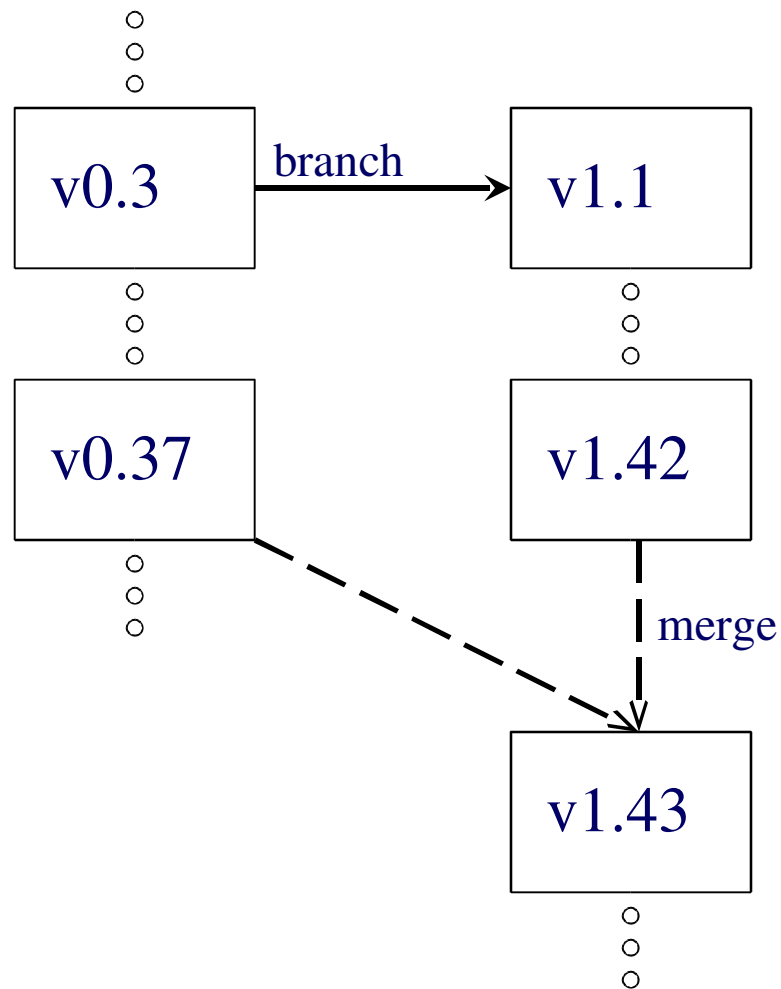
Changes on one branch don't affect others

Project may contain many branches

Why branch?

- Implement a new “major” feature
- Begin an independent sequence of development

Branching



The actual branching and merging take place in a particular user's working directory, but this is what such a sequence would look like to the repository.

Branch Life Cycle

Successful development branch

- Merged back to parent
- No further versions

Unsuccessful development branch

- Some changes pulled out?
- No further versions

Maintenance branch

- “End of Life”: No further versions

Recommendation for 15-441

You can use CVS if you're used to it

- Also: SVN, arch, ...

PRCS, Project Revision Control System

- Small “conceptual throw weight”
- Easy to use, state is visible (single text file)
- No bells & whistles

Setting to learn revision control *concepts*

- Quick start when joining research project/job
 - (They will probably not be using PRCS)

Getting Started

Add programs to your path (.bashrc):

```
$ export  
PATH=/afs/cs.cmu.edu/academic/class/15410-  
s06/bin:$PATH
```

Set environment variables (also .bashrc):

```
$ export  
PRCS_REPOSITORY=/afs/cs.cmu.edu/academic/c  
lass/15441-s06-users/group-99/REPOSITORY  
$ export PRCS_LOGQUERY=1
```

Creating A New Project

In a working directory:

```
$ prcs checkout P
```

– P is the name of the project

Creates a file: P.prj

The Project File

```
;; -*- Prcs -*-  
(Created-By-Prcs-Version 1 3 0)  
(Project-Description "")  
(Project-Version P 0 0)  
(Parent-Version -*- -*- -*-)  
(Version-Log "Empty project.")  
(New-Version-Log "")  
(Checkin-Time "Wed, 15 Jan 2003 21:38:47 -0500")  
(Checkin-Login zra)  
(Populate-Ignore ())  
(Project-Keywords)  
(Files  
;; This is a comment.  Fill in files here.  
;; For example: (prcs/checkout.cc ())  
)  
(Merge-Parents)  
(New-Merge-Parents)
```

Description of project.

Make notes about changes before checking in a new version

List of files

Using the Project File

Adding Files

```
$ prcs populate P file1 file2 ... fileN
```

- To add *every* file in a directory

```
$ prcs populate P
```

- Rarely what you want – *expensive* to source-control core files!

Removing, renaming files

- See <http://www.cs.cmu.edu/~410/prcs-cmu>

Checking In

Checking in

```
$ prcs checkin P
```

- Check-in will fail if there are conflicts.
- Hey, we forgot to talk about conflicts!

Conflicts and Merging

Suppose this file is in the repository for project P:

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    printf("Hello World!\n");
    return 0;
}
```

Conflicts and Merging

Suppose Alice and Charlie check out this version, and make changes:

Alice's Version

```
#include <stdlib.h>
#include <stdio.h>

#define SUPER 0

int main(void)
{
    /* prints "Hello World"
       to stdout */
    printf("Hello World!\n");
    return SUPER;
}
```

Charlie's Version

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    /* this, like, says
       hello, and stuff */
    printf("Hello Hercules!\n");
    return 42;
}
```

Conflicts and Merging

Suppose Alice checks in first.

```
$ prcs checkin
```

Now Charlie must perform a merge

```
$ prcs checkin ⇒ will fail
```

```
$ prcs merge
```

- **Default merge option performs a CVS-like merge.**

```
$ prcs checkin ⇒ should work now
```


Merge Mutilation

```
#include <stdlib.h>
#include <stdio.h>

#define SUPER 0

int main(void)
{
<<< 0.2(w)/hello.c Wed, 19 Feb 2003 21:26:36 -0500 zra (P/0_hello.c 1.2 644)
    /* this, like, says hello, and stuff */
    printf("Hello Hercules!");
    return 42;
===
    /* prints "Hello World" to stdout */
    printf("Hello World!");
    return SUPER;
>>> 0.3/hello.c Wed, 19 Feb 2003 21:36:53 -0500 zra (P/0_hello.c 1.3 644)
}
```

Conflicts and Merging

Pick/create the desired version

- Check that into the repository.**

Branching

To create the first version of a new branch:

```
$ prcs checkin -rExperimental_VM  
Kern.prj
```

To merge with branch X version 37:

```
$ prcs merge -rX.37 Kern.prj
```

Information

To get a version summary about P:

```
$ prcs info P
```

– with version logs:

```
$ prcs info -l P
```

CVS

Features

- What your roommate probably uses
- *Much* more popular than PRCS!
- Commit: atomic if you are careful
 - Must commit in the top-level directory
- Named snapshots: if you “cvs tag” often
- Branching: works if you are careful
- *Core operations* require care & expertise!!!

SVN - “SubVersion”

Features

- Cool name
- Somebody on your hall probably uses it
- *Much* more popular than PRCS!
- Probably less local hand-holding than CVS

RCS – Revision Control System

Features

- The data structures CVS & PRCS use internally
- **Very** reliable (in wide use since 1980's)
- Core orientation is single directories, not trees
- Lock-based - “no merge conflicts!”

RCS – Storage

Each file stored two places

- “working file” - foo.c
- RCS file – foo.c,v
 - Or, more conveniently, RCS/foo.c,v
 - If you conveniently “mkdir RCS”

RCS file contains multiple versions of working file

- Format is basically diffs against current version

Creating RCS Files

After making 'RCS' subdirectory...

Initialize RCS for your file `mysource.c` (assume you have already created it) by checking it in (`ci`)

```
[geoffl@ux3 ~/tmp]$ ci mysource.c
RCS/mysource.c,v <-- mysource.c
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> This source file contains a simple algorithm to solve the Halting
Problem.
>> .
initial revision: 1.1
done
```

Result: version 1.1

Checking out files

In previous example, after `ci`, `mysource.c` is *gone*!

To retrieve `mysource.c`, use `co` command (“check out”)

```
[geoffl@ux3 ~/tmp]$ ls -l
total 2
drwxr-xr-x    2 geoffl  users          2048 Jan 19 15:39 RCS
[geoffl@ux3 ~/tmp]$ co mysource.c
RCS/mysource.c,v --> mysource.c
revision 1.1
done
[geoffl@ux3 ~/tmp]$ ls -l
total 3
-r--r--r--    1 geoffl  users           137 Jan 19 15:39 mysource.c
drwxr-xr-x    2 geoffl  users          2048 Jan 19 15:39 RCS
```

Note: permissions don't let us change `mysource.c`

- Our copy is read-only—“this is a feature, not a bug”

Locking files

To change `mysource.c`, must acquire lock

- `co -l mysource.c` does checkout with lock
- `rcs -l mysource.c` acquires lock after the fact

Obtaining lock sets file to read/write

- **Plan:** edit the file quickly, check it in
 - `ci -u mysource.c` means “check in and unlock”
- **Not the plan**
 - Lock the file, go home and go to sleep
 - Now your partner can't lock the file

Locking files

To change `mysource.c`, must acquire lock

- `co -l mysource.c` does checkout with lock
- `rcs -l mysource.c` acquires lock after the fact

Obtaining lock sets file to read/write

- **Plan:** edit the file quickly, check it in
 - `ci -u mysource.c` means “check in and unlock”
- **Not the plan**
 - Lock the file, go home and go to sleep
 - Now your partner can't lock the file
 - ☞ Guess why CVS was invented?

Locking files

To change `mysource.c`, must acquire lock

- `co -l mysource.c` does checkout with lock
- `rcs -l mysource.c` acquires lock after the fact

Obtaining lock sets file to read/write

- **Plan:** edit the file quickly, check it in
 - `ci -u mysource.c` means “check in and unlock”
- **Not the plan**
 - Lock the file, go home and go to sleep
 - Now your partner can't lock the file
 - ☞ **Guess why CVS was invented?**
 - **Solution:** break your partner's lock
 - Now your partner gets to merge in your changes
 - (Luckily, RCS doesn't have merge conflicts like CVS, PRCS)

Versions

Each version of the file has a version number

- “release.revision” format –
 - 4.2 is release 4, revision 2
- Doesn't necessarily correspond to anything about real world version numbers
- By default, each `ci` of a changed file increments revision number by 1

Can use `-r` flag to specify version numbers

- Use this with `co` to retrieve old versions
- Use this with `ci` to name new versions
 - `ci -r1.8 mysource.c` will check in `mysource.c` as 1.8
 - `ci -r2 mysource.c` will check in `mysource.c` as 2.1

Info On “Alternatives”

More information about CVS (my personal choice)

- <http://www.nongnu.org/cvs/>
- <http://www.tortoisecvs.org/> for Windows users

RCS

- Look at `man rcs`, `man rcsintro`
- Official RCS homepage:
 - <http://www.cs.purdue.edu/homes/trinkle/RCS/>
- Other useful features
 - `ci -l`: check-in a version but keep the file and the lock
 - `ci -u`: check-in a version but keep a read-only version of file
 - `rcsdiff`: display differences between versions
 - `rcsmerge`: merge changes in different versions of a file