

CSC207H

Version Control Systems

Yes, Boss. Everything is under control!

Change Tracking

What strategies do you use for tracking changes to files?

- Tracking what, now?
- *Ad hoc snapshots.*
- Application-specific change-tracking.

It Gets Me From Here to There

How do you move your electronic work?

- Work?
- Work from external storage device.
- Copy everything.
- Synchronization software.

Taming Teamwork

- Exchanging files
- Coordinating changes
- Tracking progress

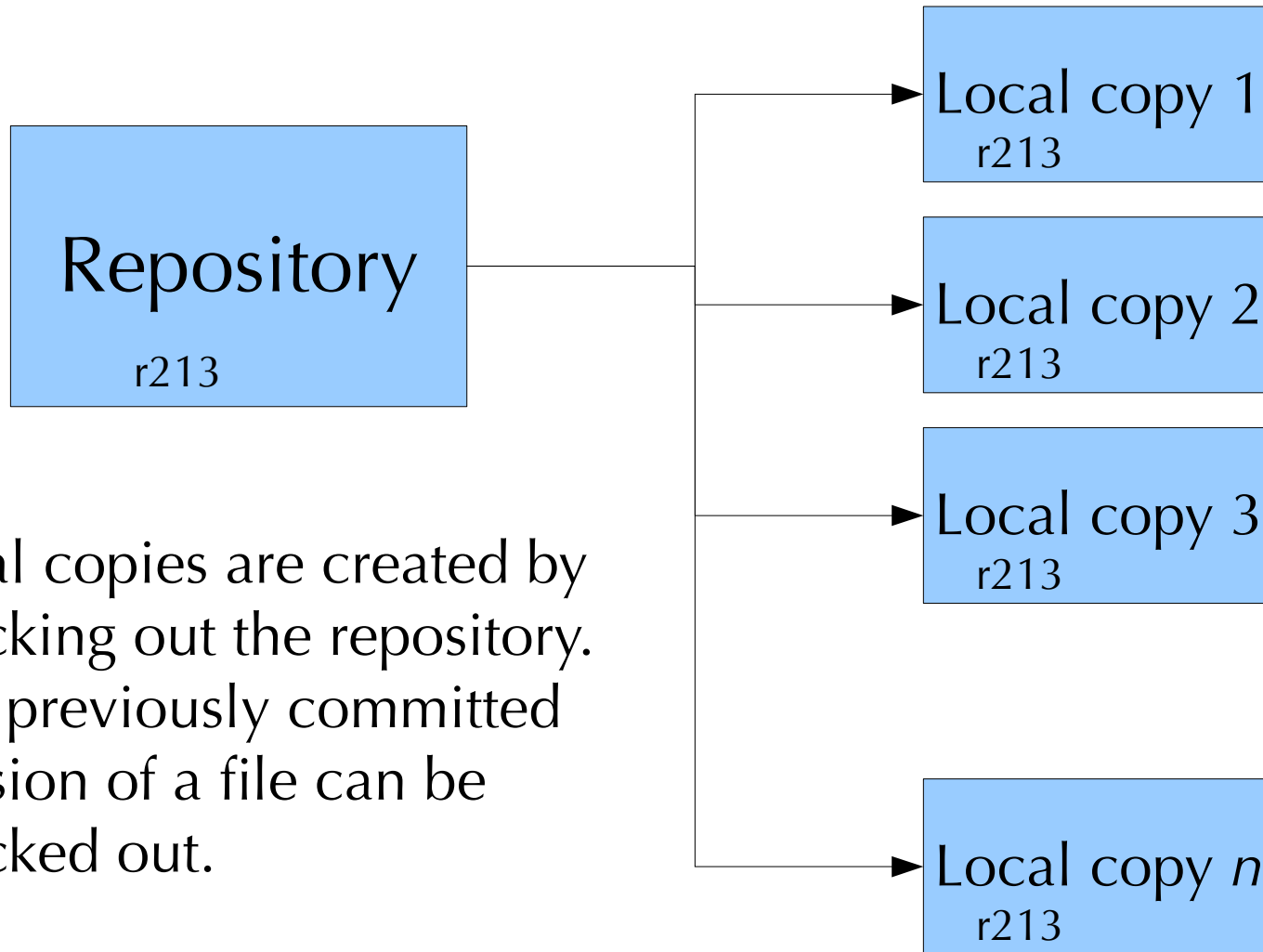
Version Control

- Files kept in an electronic library known as a *repository*.
 - The repository is the master copy.
 - It should not be changed directly!
- A copy of the library can be *checked out*.
- Changes can be made to these local copies.
- These changes can then be *committed* to the repository.

Version Control

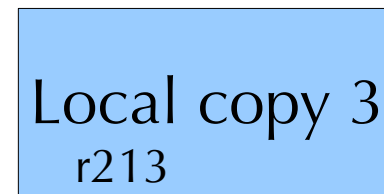
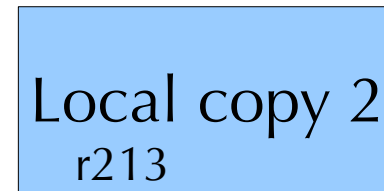
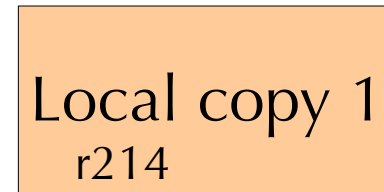
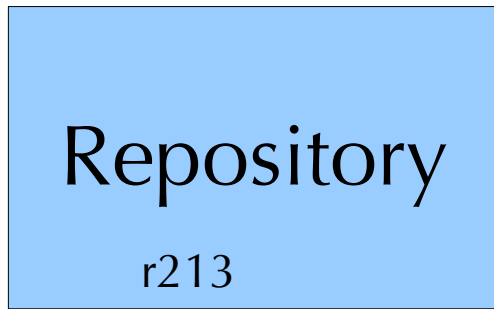
- All changes are recorded.
 - No changes are “permanent” since older revisions can be retrieved.
 - Log messages can and should be included with every commit.
- In this course, we will be using the Subversion (svn) revision control system.

Version Control

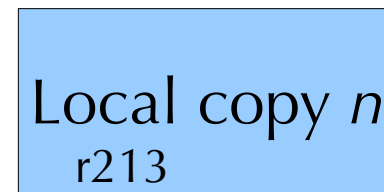


Local copies are created by checking out the repository. Any previously committed revision of a file can be checked out.

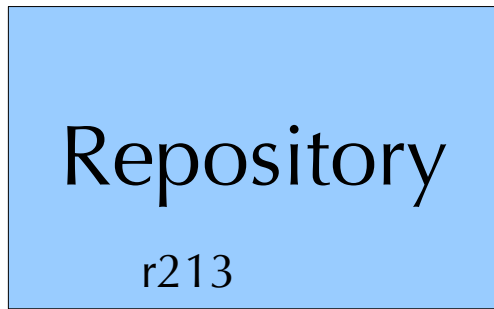
Version Control



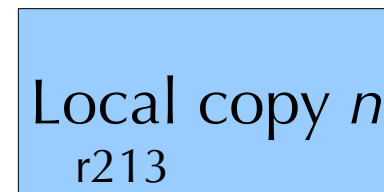
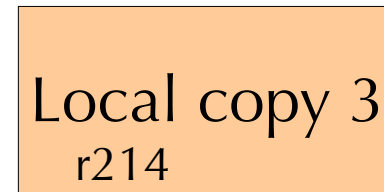
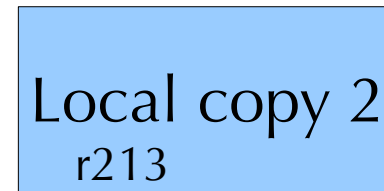
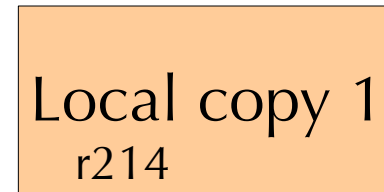
Local changes (file changes, additions, and deletions) are not seen by the repository or other local copies.



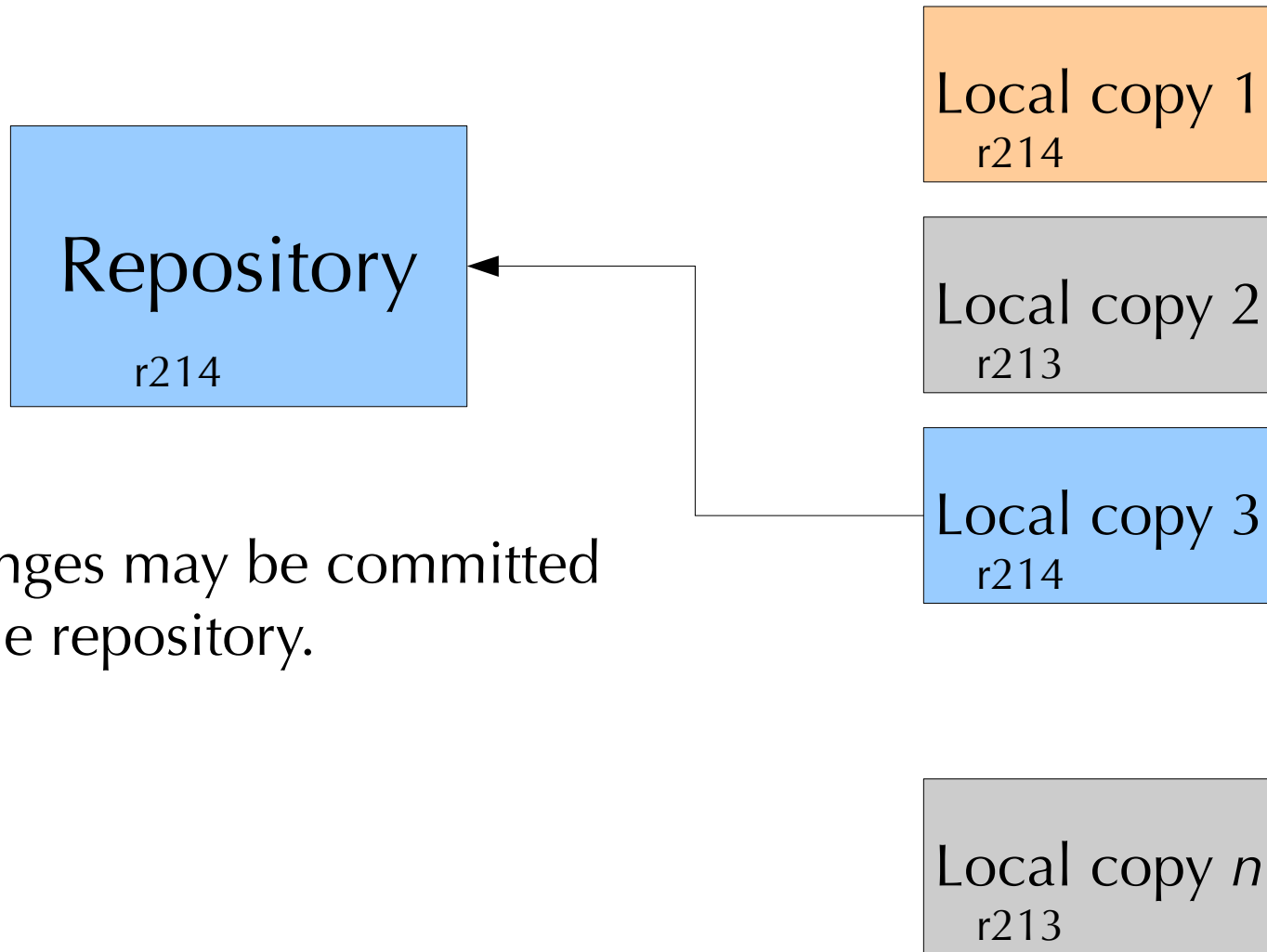
Version Control



If more than one local copy is modified, they may share (locally) the same revision number, but are still independent.

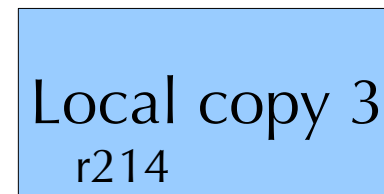
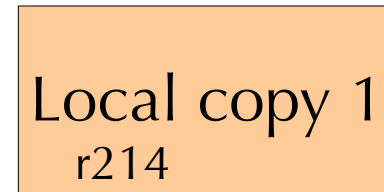


Version Control



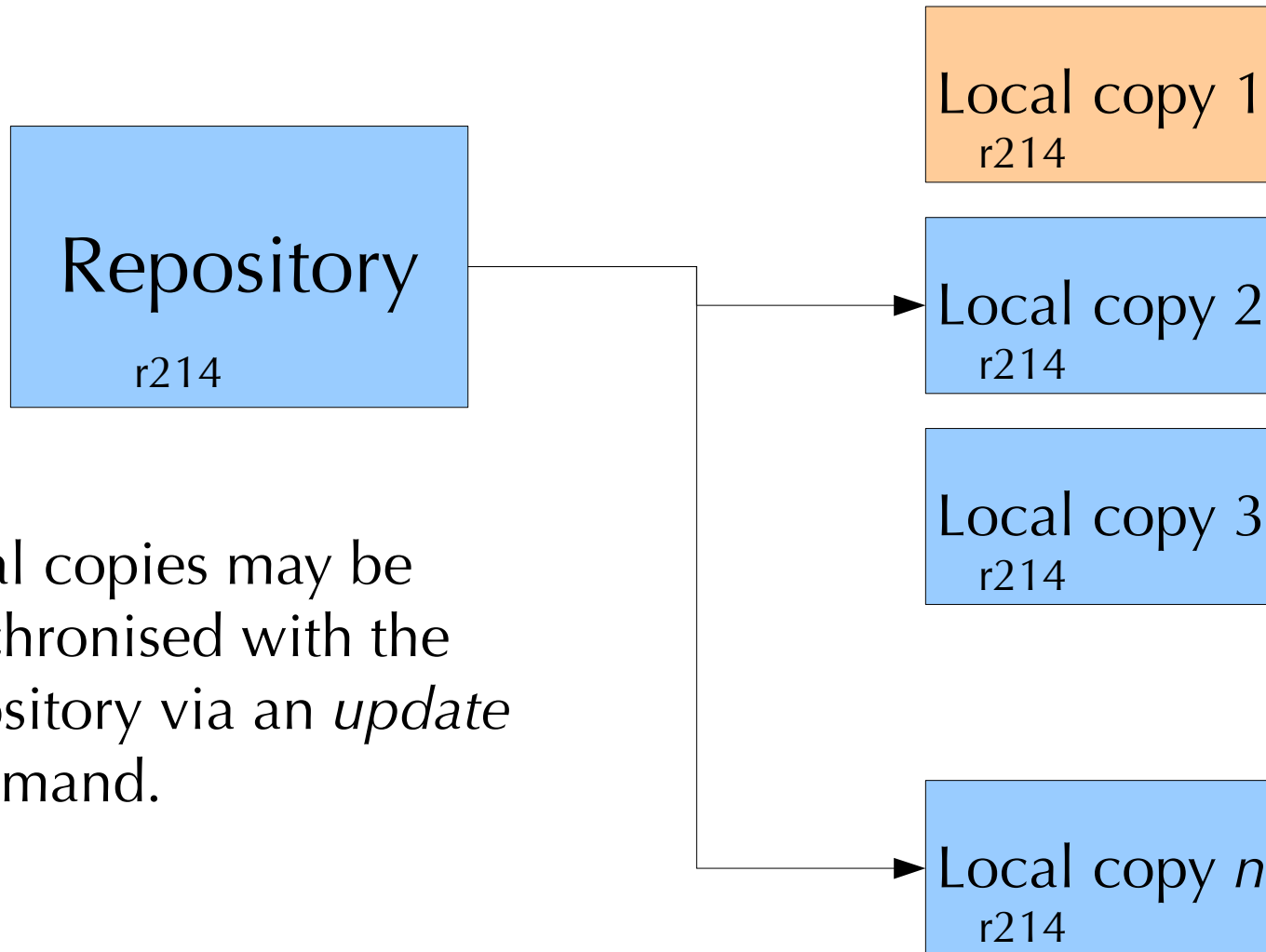
Changes may be committed to the repository.

Version Control



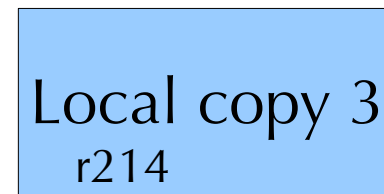
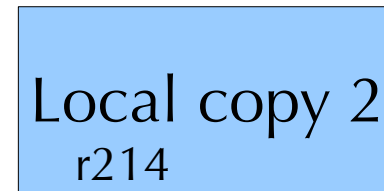
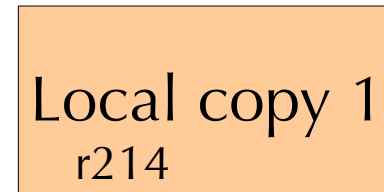
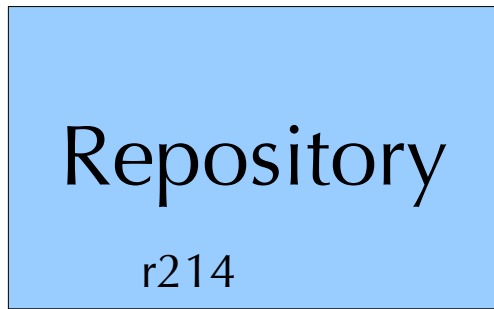
Local copy 1's r214 is not the same as r214 in the repository or local copy 3. Local copies 1, 2 and n are out-of-date with respect to the repository.

Version Control

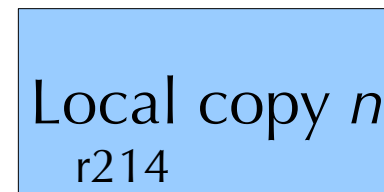


Local copies may be synchronised with the repository via an *update* command.

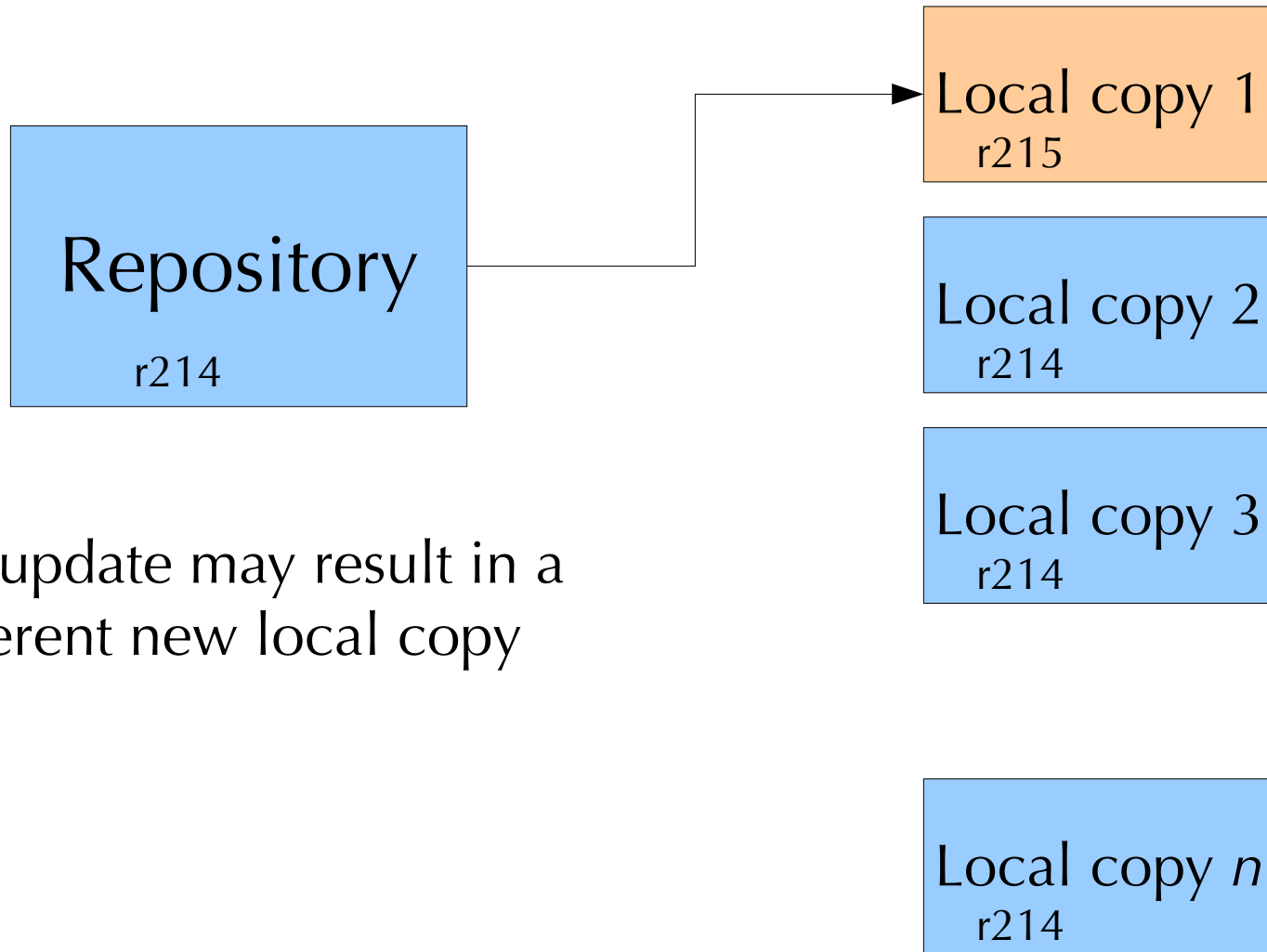
Version Control



Local copy 1 cannot commit because it has outdated versions of files. It must be updated before its changes can be committed.

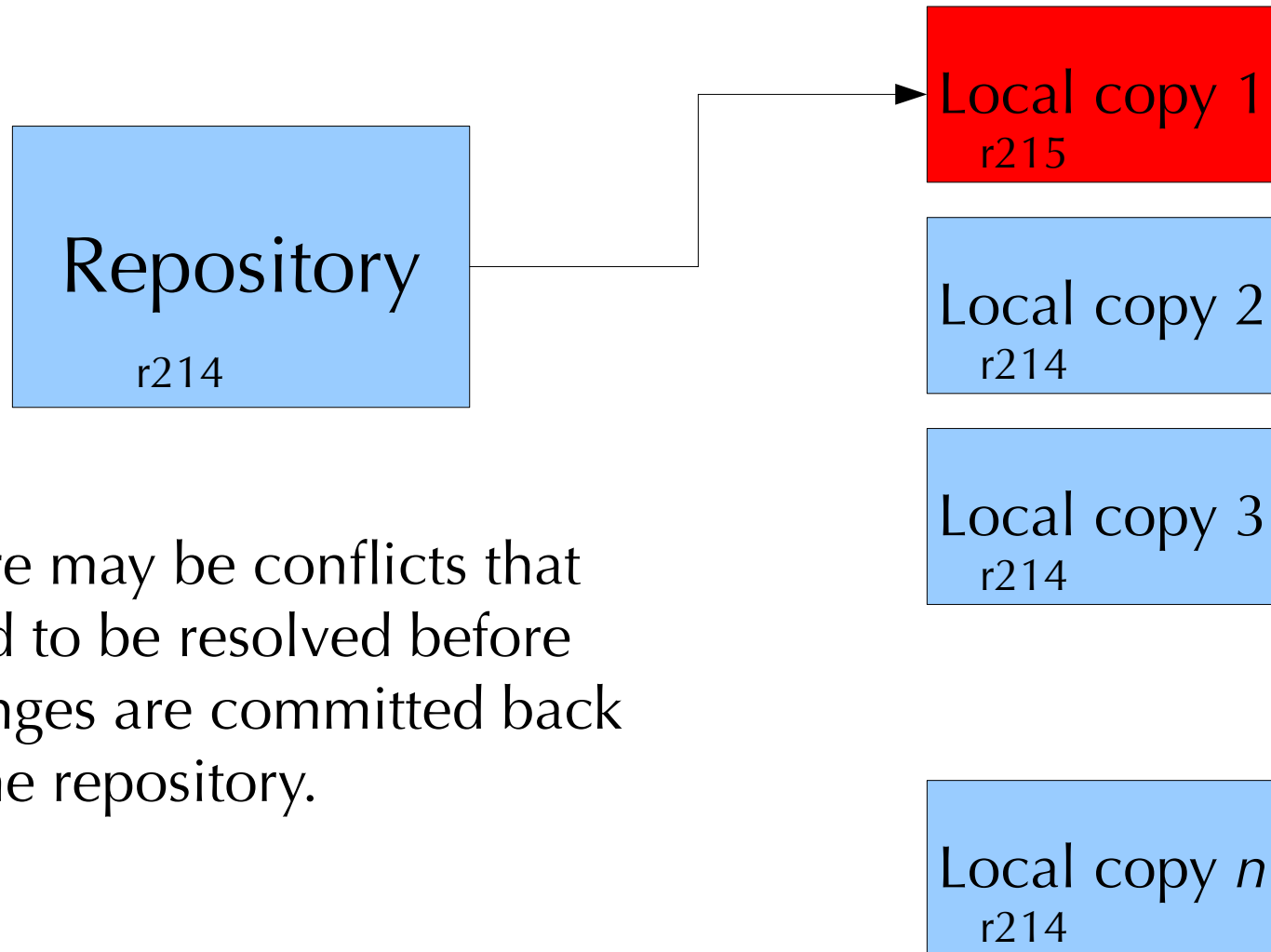


Version Control



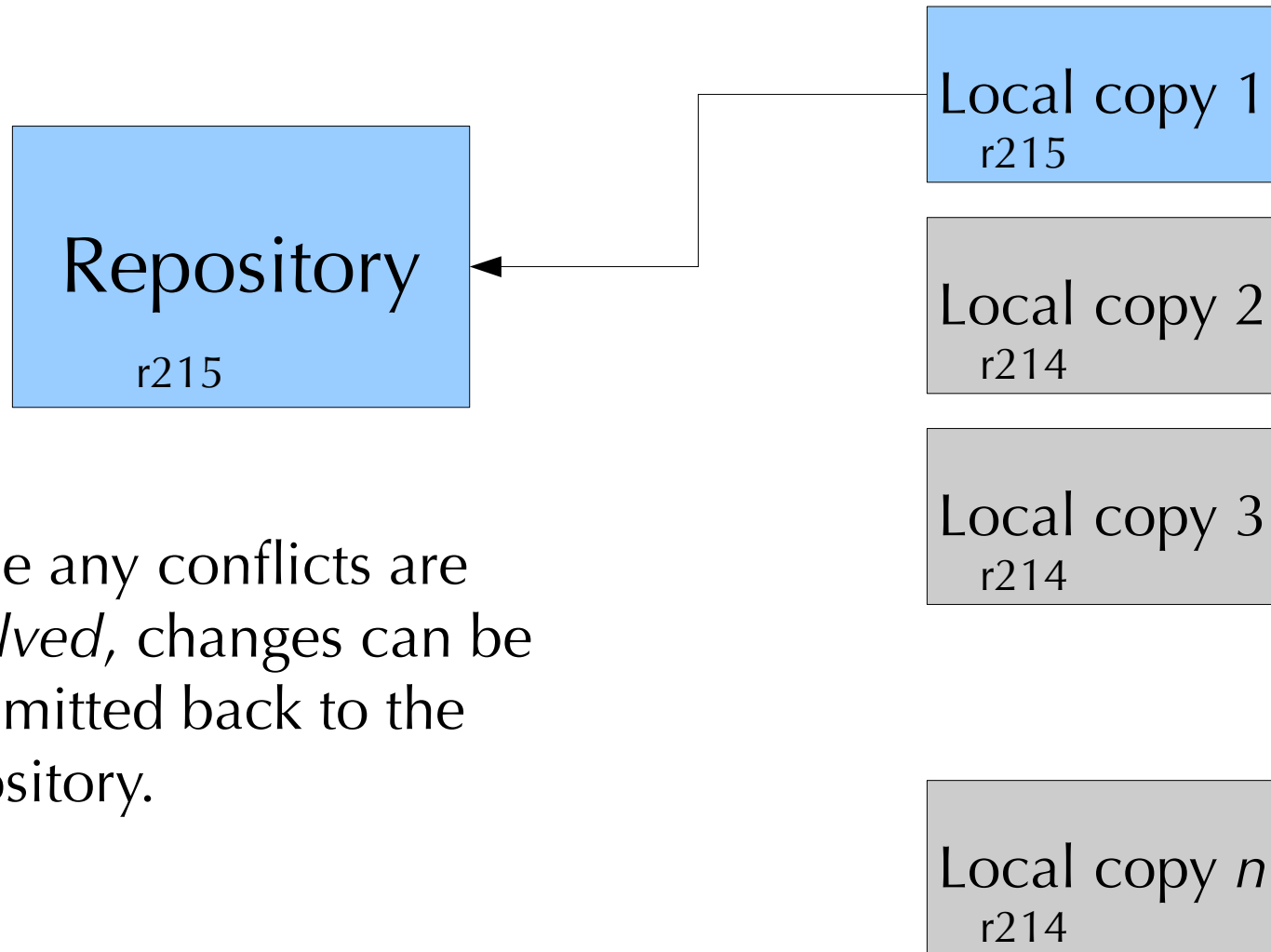
The update may result in a coherent new local copy or...

Version Control



There may be conflicts that need to be resolved before changes are committed back to the repository.

Version Control



Once any conflicts are *resolved*, changes can be committed back to the repository.

Compare and Contrast

Design choice	Differences
Distribution model: Client-server vs. <u>distributed</u>	In a client-server model, there is a “master” version; clients usually need to have network access to commit. In a distributed system, everyone has a “master” copy of the repository.
Concurrency: Merging vs. locking	Locking prevents multiple people from writing to the same file(s) whereas merging combines concurrent changes — sometimes manually, if necessary.
Versioning: <i>Per-file</i> vs. per-commit	Per-file versioning increments a file's version when changed; in per-commit versioning, all files at a given point in time have the same version number.
Revision IDs: Namespace vs. <u>hashes</u> vs. pseudorandom vs. timestamp	Namespace revision IDs increment file versions sequentially (e.g. 1, 2, 3). Content hashes give IDs based on the contents of a revision; revisions with the same contents will have the same hash value. Pseudorandom IDs carry no meaning. Timestamp IDs are based on when a changeset is committed.

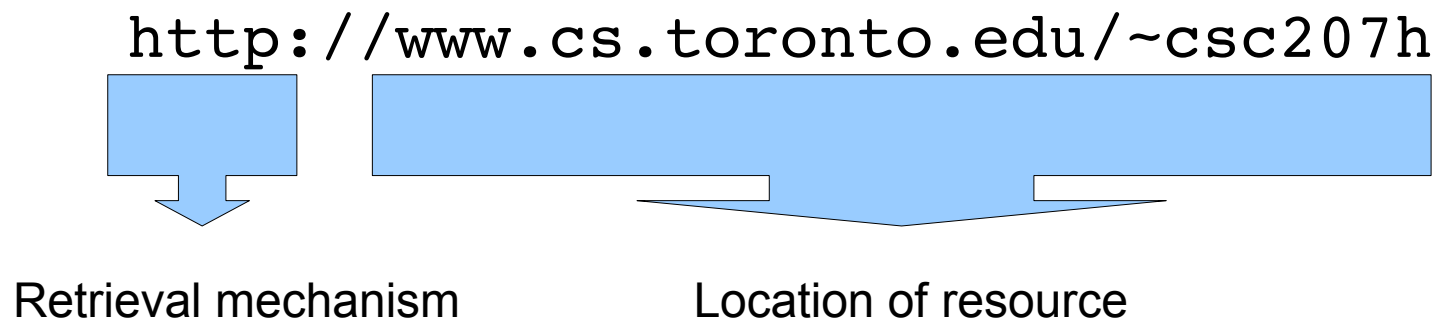
CVS

Subversion

Monotone

Subversion URLs

- A uniform resource locator (URL) is a type of address that points to a resource and indicates a retrieval mechanism.
- Subversion uses URLs to point to repositories and items in repositories.



Subversion URLs

Subversion can retrieve files via

- a *file system* (`file://`),
- *http* and *https* (`http://` and `https://`),
- the *svn* protocol (`svn://`), or
- the *svn* protocol through *ssh* (`svn+ssh://`).

Common subversion commands

- `svn checkout URL`
 - Check out a repository from the given URL.
 - `co` can be used in place of `checkout` to save typing.
- `svn add files`
 - Adds the given files to the local copy.
 - A `commit` is still required to make the changes available in the repository.

Common subversion commands

- `svn update [files]`
 - Brings local files in sync with the repository.
 - Conflicts requiring manual merging are flagged.
 - `up` can be used in place of `update` to save typing.
- `svn commit [files] [-m message]`
 - Pushes local changes to the repository.
 - Only allowed if local copy is up-to-date.
 - `ci` can be used in place of `commit` to save typing.

Common subversion commands

- `svn remove files`
 - Removes files from local copy.
 - `rm` can be used in place of `remove` to save typing.
 - A commit is still required to make the changes available in the repository.
- `svn status [files]`
 - Shows the status of files in the local copy with respect to the repository.

Common subversion commands

- `svn diff [-r revision] [files]`
 - Lists differences in files compared to the current version of files in the repository or a specific revision, if specified.
- `svn log [files]`
 - Shows the history of commits including the committer, time, and commit message for each commit.

Common subversion commands

- `svn blame [-r revision] [files]`
 - Displays files line-by-line with the user name of the last person to change a line and the revision in which the change was made.
- `svn revert [files]`
 - Revert files the state they were in after the last update.

Common subversion commands

- `svn move fromfile tofile`
 - Renames or moves a file in the local copy.
 - `mv` can be used in place of `move` to save typing.
 - A commit is still required to make the changes available in the repository.
- `svn help [command]`
 - Get on-line help for Subversion.

Common subversion commands

- There are some more useful commands when working in teams:
 - `merge`
 - `lock`
 - `unlock`
 - `resolved`
 - `copy`
- We will cover these when you begin your projects.

File Organization

- `root`
 - `trunk` Main code development here.
 - `branches` Side development here.
 - `tags` Important releases here.

What to Check In

- Things to check in:
 - Your own test programs,
 - Unit tests/expected output,
 - Readme files, notes, logs, etc.
- Things **not** to check in:
 - Automatically generated files (e.g., Python `pyc` and Java `class` files); they waste disk space and bandwidth.
 - Temporary files.
 - Dead/commented-out code; you can retrieve this from older revisions of a file, so do not clutter your code!

When to Check In

- After a new feature is implemented.
- After a new test is passed.
- When you need to move files to another location.
- Revisions that do not run should not be checked into the trunk of a repository.

CSC207 Check-ins

- You will each be given access to a repository for submitting homework.
- Checking in is generally very fast.
- The first time may take longer.
- Check in early!
- Check in often! It is easier to track and fix problems if check-ins are made after small changes.
- Expect to lose all arguments about submission problems if your first check-in was on the due-date or the day before.

Course Repository

- Code examples and other text documents will be made available through a read-only Subversion repository:

```
svn://greywolf.cdf.toronto.edu:7796/200905/csc207h
```

- You may wish to check out the repository as soon as possible.