An Introduction to Mercurial Version Control Software

LANS Weekly Seminar
October 17, 2006

Satish Balay
balay@mcs.anl.gov

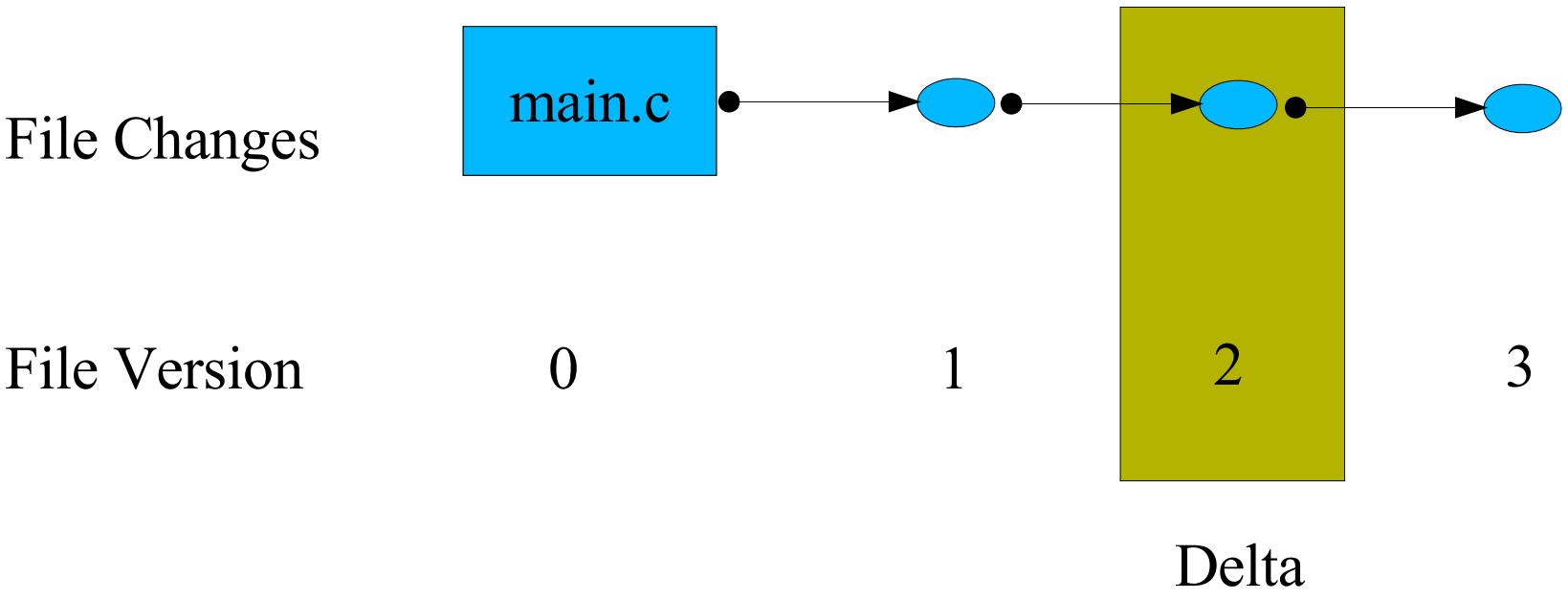
Outline

- Why use version control?
- Simple example of revisioning
- Mercurial introduction
  - Local usage
  - Remote usage
  - Normal user workflow
  - Organizing repositories [clones]
- Mercurial at MCS
- [Demo]

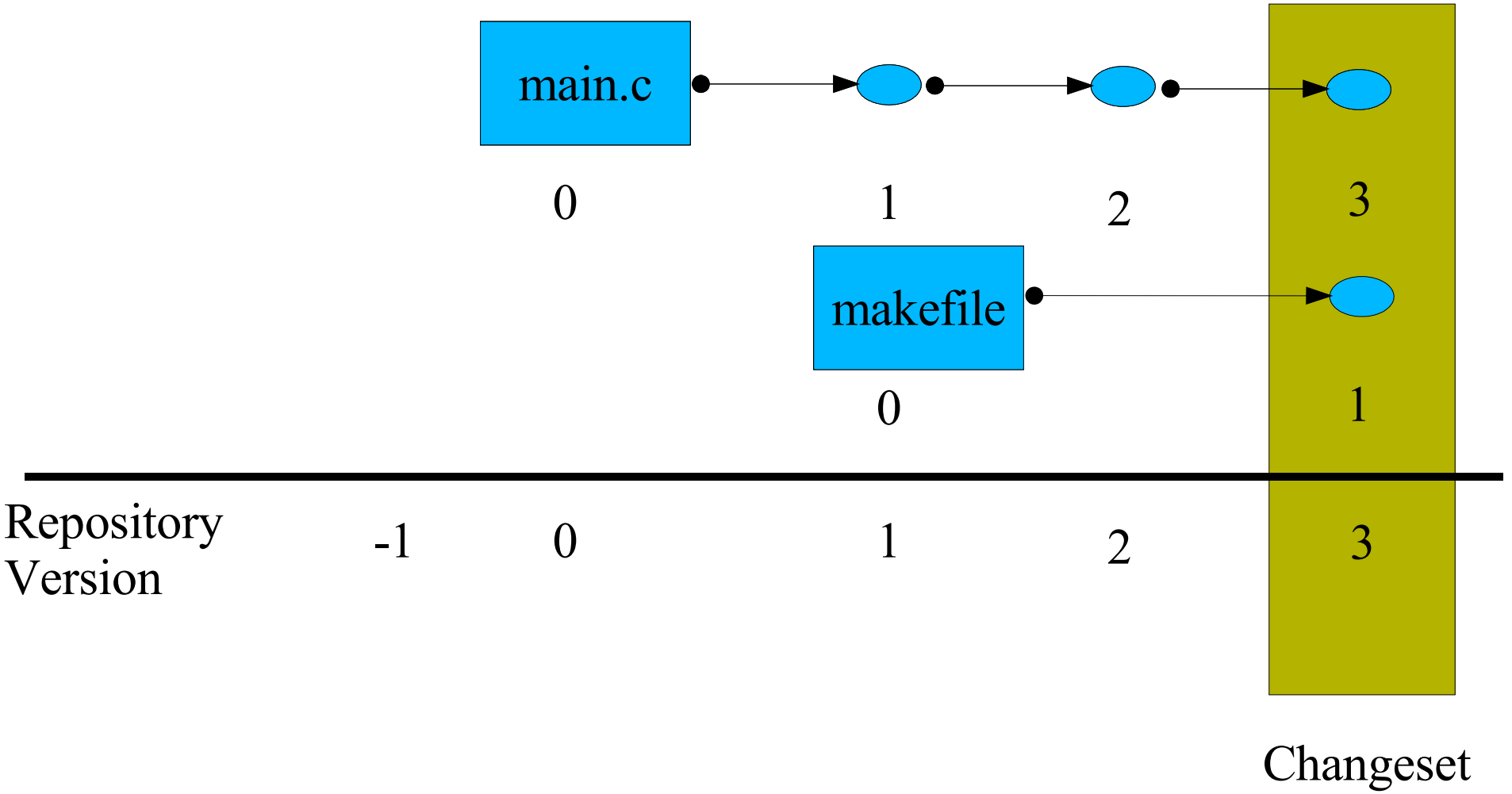
What do we use Version Control for?

- Keep track of changes to files
- Enable multiple users editing files simultaneously
- Go back and check old changes:
  * what was the change
  * when was the change made
  * who made the change
  * why was the change made
- Manage branches [release versions vs dev]

Simple Example of Revisioning



Simple Example Cont.



Some Definitions

- Delta: a single change [to a file]
- Changeset: a collection of deltas [perhaps to multiple files] that are collectively tracked.
- Repository: collection of files we intend to keep track of. This includes the revision history
- Version [or Source] Control Tool: Enables us to keep track of all the changes [to files] in a repository

Mercurial

- Distributed version control tool.
- http://www.selenic.com/mercurial
- OpenSource [GPL]
- Active mailing list : mercurial@selenic.com
- Written in python
- Works on linux, windows, and other machines
- Reasonably efficient [handles 9000+ changesets in PETSc]

Usage: Creating a Repository

- *mkdir project*
- *cd project*
- *hg init*

- Initializes the directory 'project' as a mercurial repo.
- All 'hg' commands are invoked inside the repository
- All commands are in the form 'hg command'. For example : *hg help*
- Stores metadata in the subdirectory *project/.hg*

Usage: Adding/Modifying Files

- *cd project*
- *touch main.c*
- *hg add main.c*
- *hg commit*
- *emacs main.c* [edits to file]
- *hg commit* [alternative: *hg ct*]

- 'add' indicates the file is now part of the repository.
- 'commit' creates a changeset for the current changes. [prompts the user to enter comments]

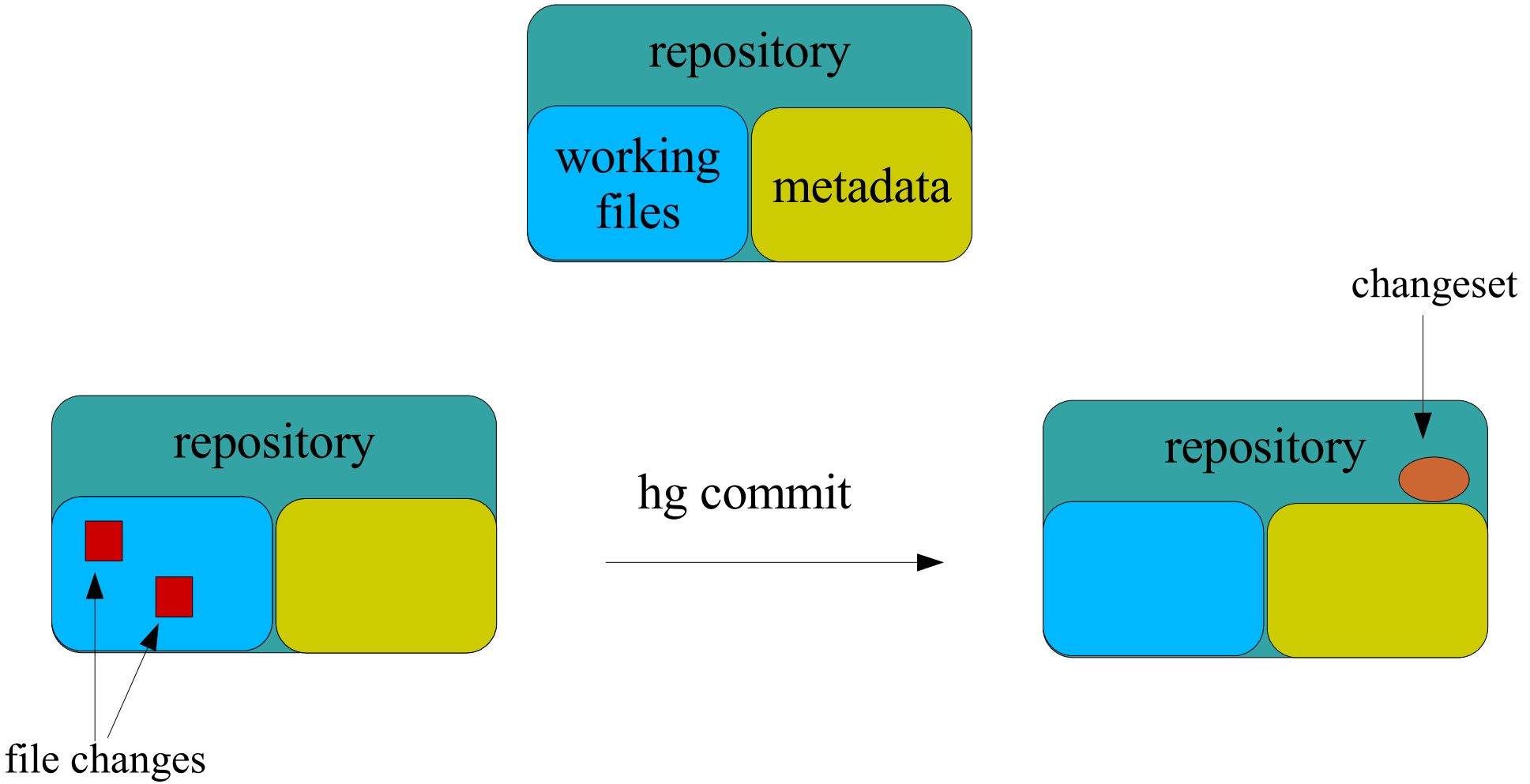
Repository Data vs Working Files

- Repository data is the revision history and graph of all the changes. Its stored in project/.hg directory
- Working files are the reset of the files in project. User edits the working files.
- *hg tip* [show the tip revision of the repository graph]
- *hg parent* [show the parent revision of the working dir]

Note: Working dir files can correspond to any revision of the repository. So one has to be careful about this point [and not assume the parent is always the tip revision]

- *hg update REV* [update working copy to REV version]

Illustration of Changes



Checking Status/History

- hg status [list the current modified, unknown files]
- hg diff [list the diff of the changed files in patch form]
- hg log [list the revision history of all changes]

- hg view [extension: GUI tool to check changeset graph]
- hg ct [extension: GUI tool to commit changes]

Note: So far we have delt with local opeations on the repository

Distributed Model

- Peer to Peer: all copies of repositories are equivalent.
- Information flows between repositories as changesets.
- Each operation is between two repositories.
- *hg clone /home/balay/old-repo new-repo*
- *cd new-repo* [Local reposiory to invoke cmds]
- *hg pull [repo]* [get remote changesets and apply locally]
- *hg push [repo]* [apply local changes to the remote repo]

Notes:

- Every repository has complete revision history [metadata]
- One can switch roles of old-repo & new-repo
- Remote operations between repositories [as oposed to local operations]

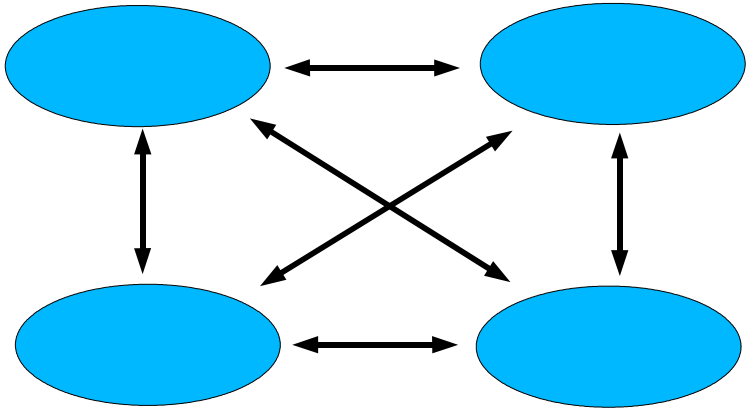
URLs

hg help pull [documentation of urls]

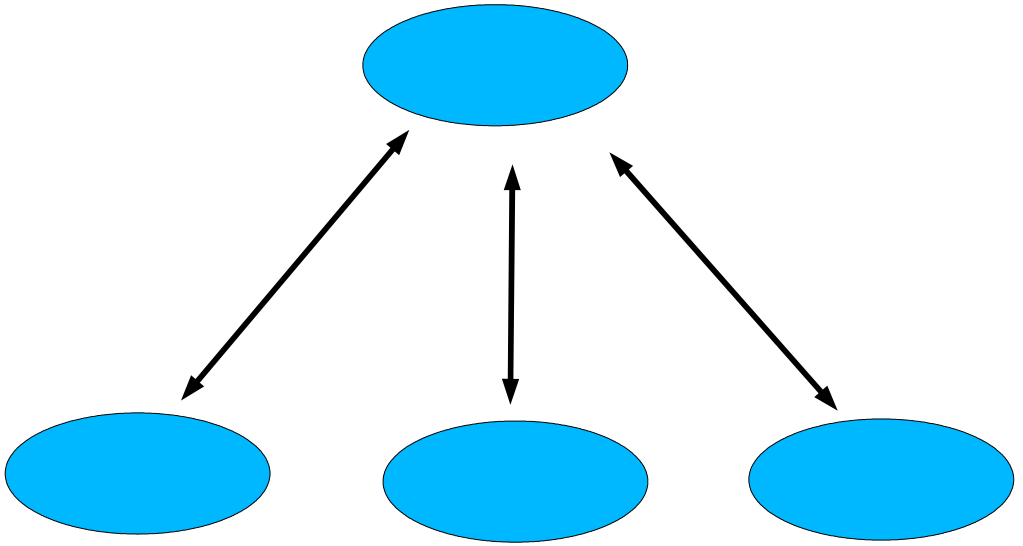
- /home/balay/petsc-dev
- ssh://petsc@harley.mcs.anl.gov//home/petsc/petsc-dev
- http://hg.mcs.anl.gov/petsc/petsc-dev [readonly]
- http-old://www.mcs.anl.gov/~petsc/project [readonly]
- https:// [read/write support in newer versions]

Note: 'hg clone' stores the URL for remote repository [in .hg/hgrc] – so, when push,pull are invoked, url is not required. [versions 0.9.1 and newer]

Organizing Repositories [clones]



Any to Any

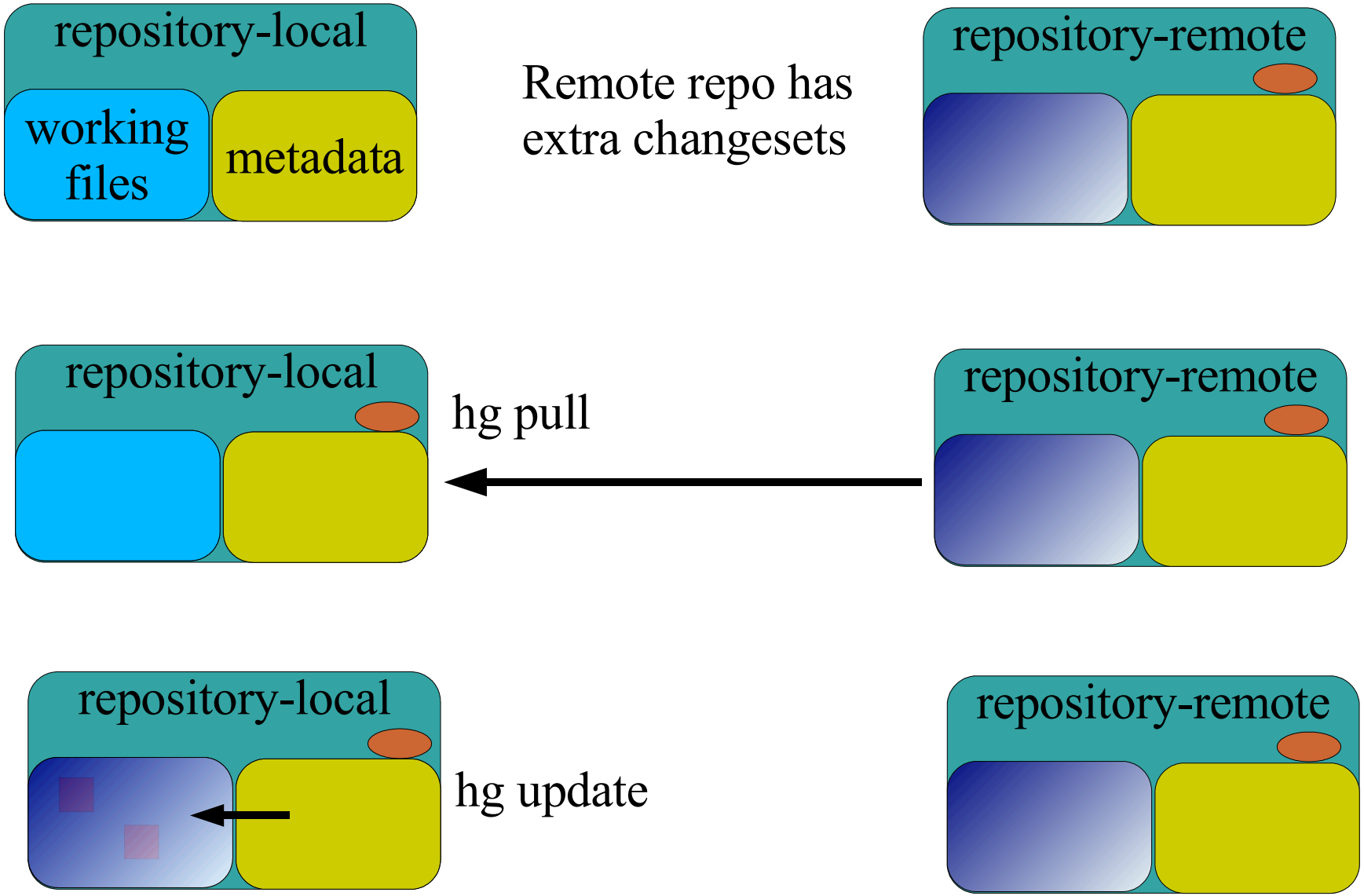


Shared Common

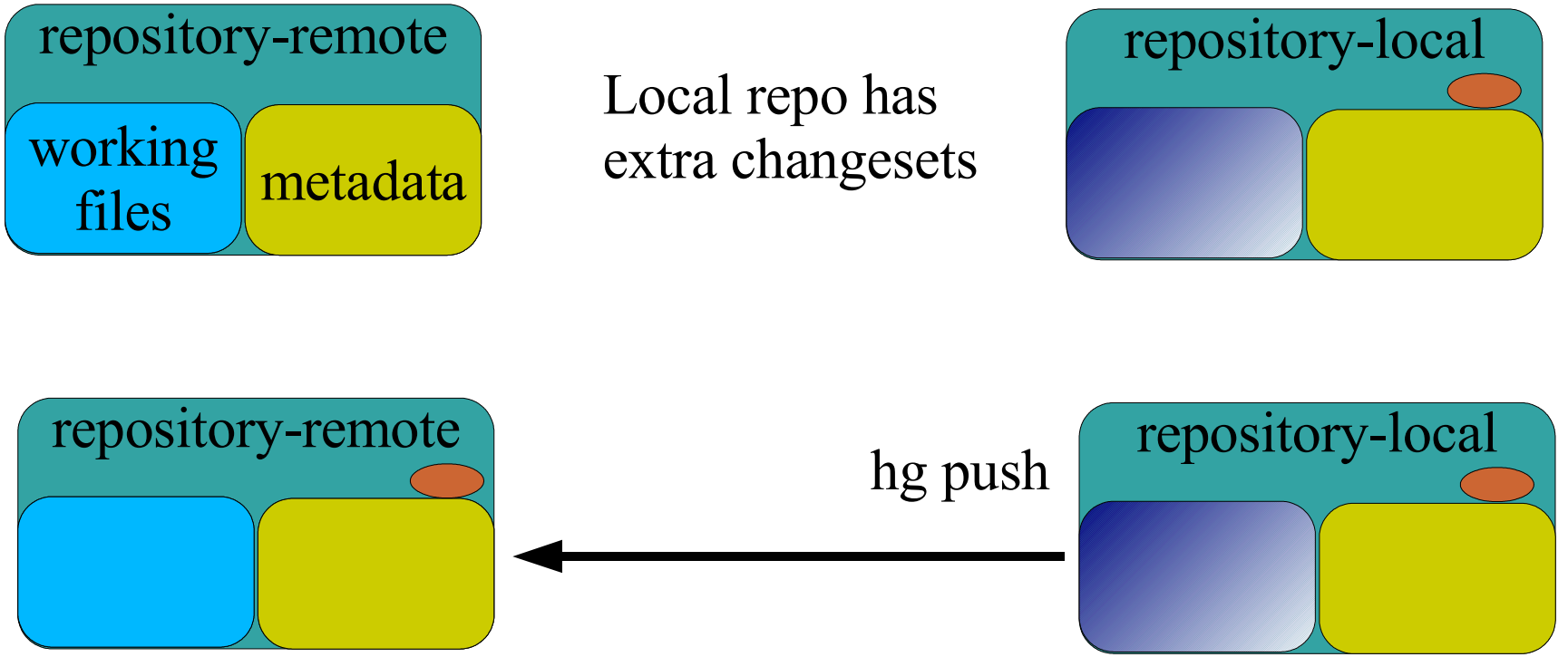
Methods of communicating changes
- clone/push/pull [changesets]
- import/export [email patch]
- bundle/unbundle [email changesets]

The relations are not hardcoded

Syncing Two Repositories with Changesets to Remote Repository

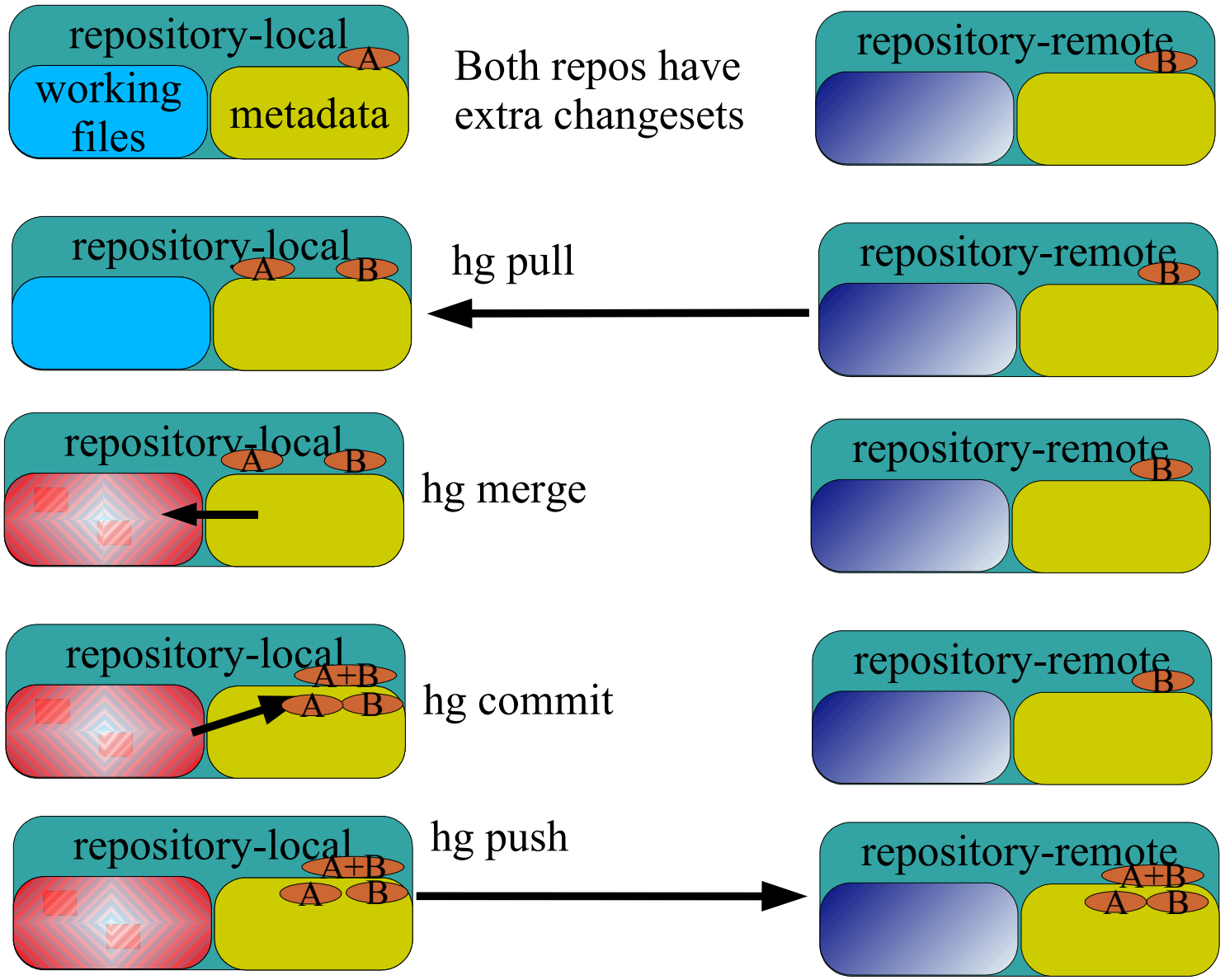


Syncing Two Repositories with Changesets to Local Repository

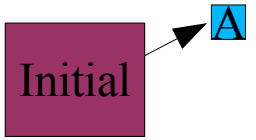


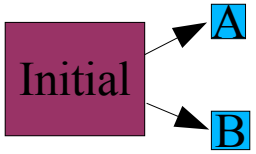
Updating Working copy of remote is not necessary.

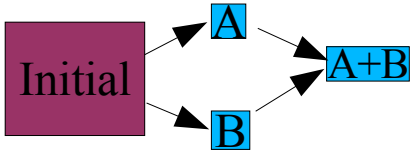
Syncing Two Repositories with Changesets to both Repositories



RevisionGraph Change







A: local repo changeset
B: remote repo changeset
A+B: merge changeset

Normal User Work Flow

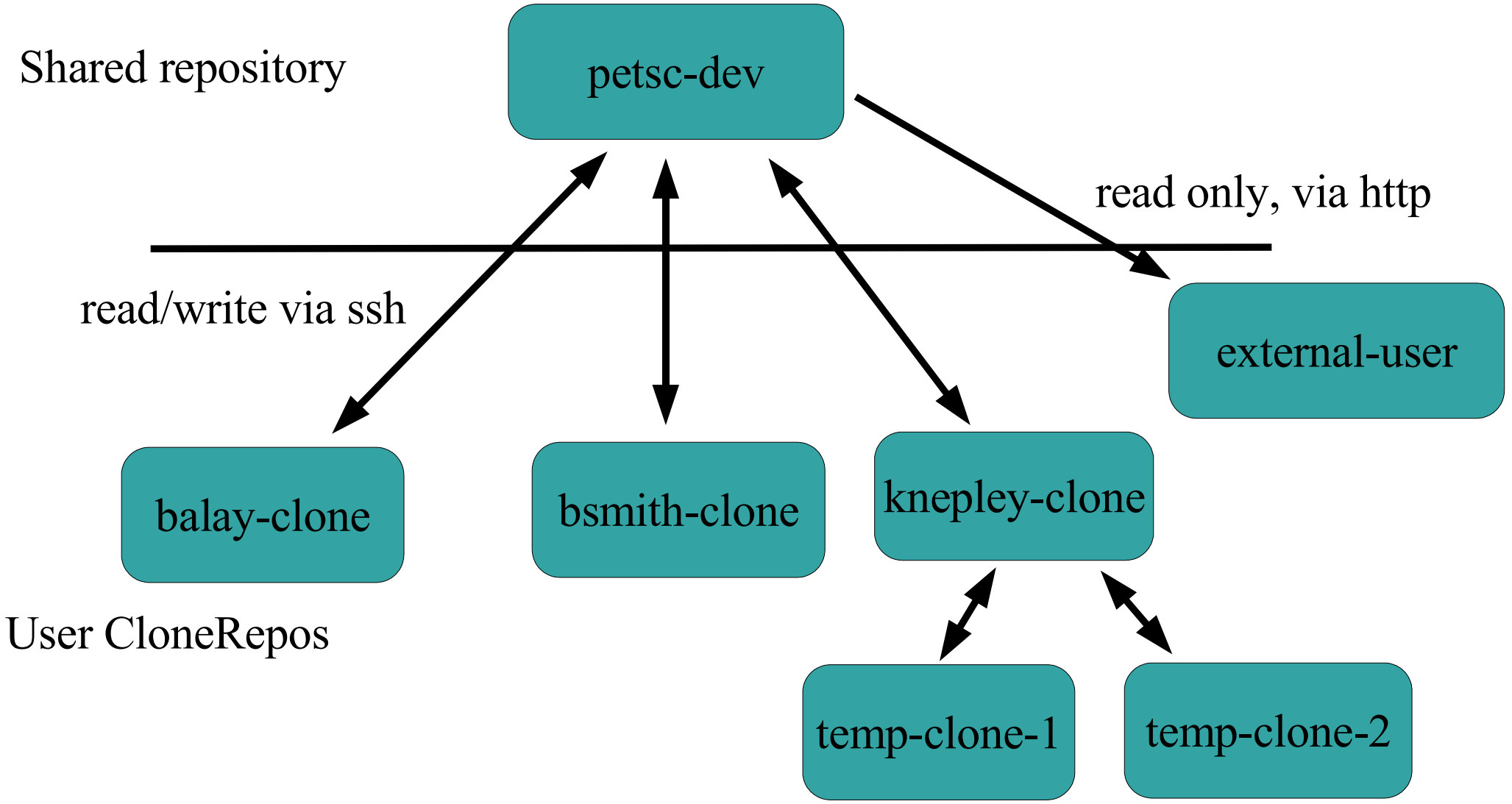
- <make changes to working files>
- hg commit [commit local changes]
- hg pull [check & obtain remote changes]
- hg merge [auto merge – if not use external merge tool for eg: kdiff3]
- hg commit [commit the merge changeset]
- hg push [push local changesets + merge changesets to the remote repository]

Handling Uncomitted Edits ?

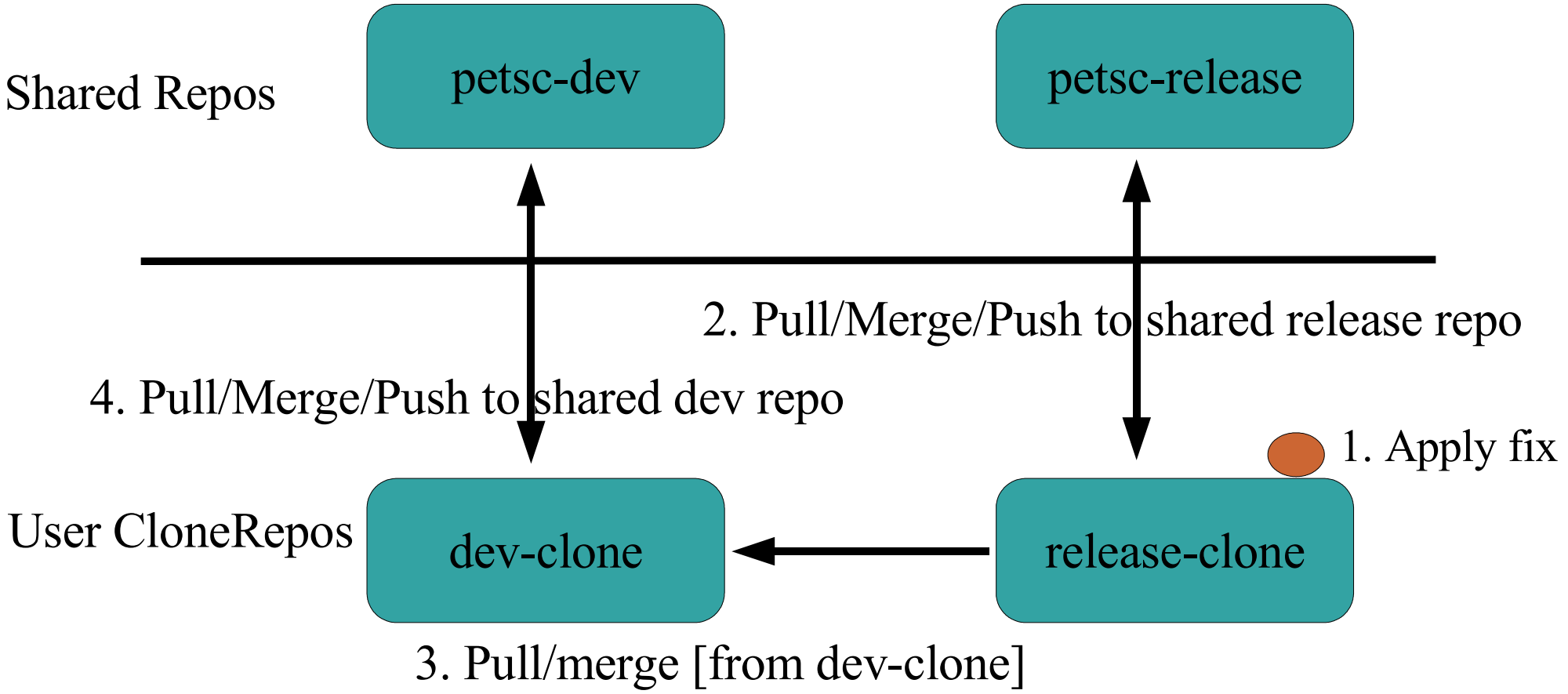
- Uncomited chages present with local changesets
- Uncomited changes present with push/pull
- Uncomited changes present during update/merge

- More things need to be kept track off [uncommited changes, commits, commits in the remote repository, merges etc..]
- This is best avoided...

Multiple Users: Communicating Changesets using a Shared Repository



Managing Patches to Release Versions



Browsing changes

- *hg view*
- *hg log*
- *hg annotate filename [REV]*
- *hg update [REV]*
- *hg serv* [starts a web server]
- Use a web browser to browse changes

Mercurial at MCS

- MCS Linux boxes has mercurial 0.9 installed
- /mcs/mercurial/project can be used for hosting repositories for web acces
- http://hg.mcs.anl.gov/project is the web url.
- For eg: some of the repositories of the PETSc project are at http://hg.mcs.anl.gov/petsc