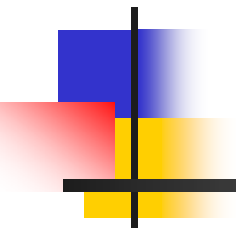
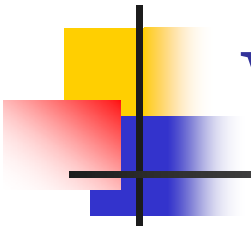


Introduction to CVS

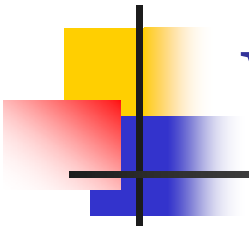
Hao-Ran Liu

2002/8/20



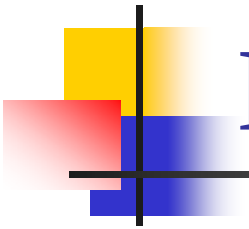
What is CVS?

- CVS stands for Concurrent Versions System
- Tools for Collaborative code development
  - Distribute code among developers
  - Aid to communication (identify changes)



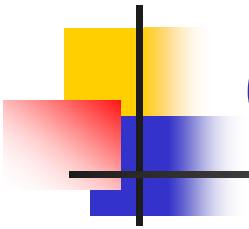
What is CVS not?

- CVS is not a build system (eg. Makefile)
- CVS is not a substitute for management
  - Code merging, branch or release date
- CVS is not a substitute for developer communication
  - It is only a tool to help you distribute and identify changes among developers.



Basic terminology

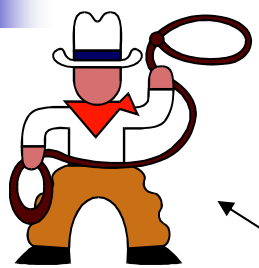
- Repository
  - Stores a complete copy of all the files and directories which are under version control.
  - Defined by $CVSROOT
- Module
  - A hierarchy of folders and files beginning at any folder in the hierarchy of the repository
- Revision
  - Version number of a file
- Tag
  - Give symbolic revision to a set of files

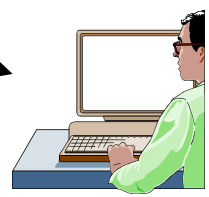


CVS Overview

- source tree in central place
  - environment variable CVSROOT
- users make copy of (parts of) this tree
  - creates subdir CVS/ in each node of tree
- users refresh their copy of the tree
- changes are made to local copy
- then merged into repository
- no locking!

Client-server architecture

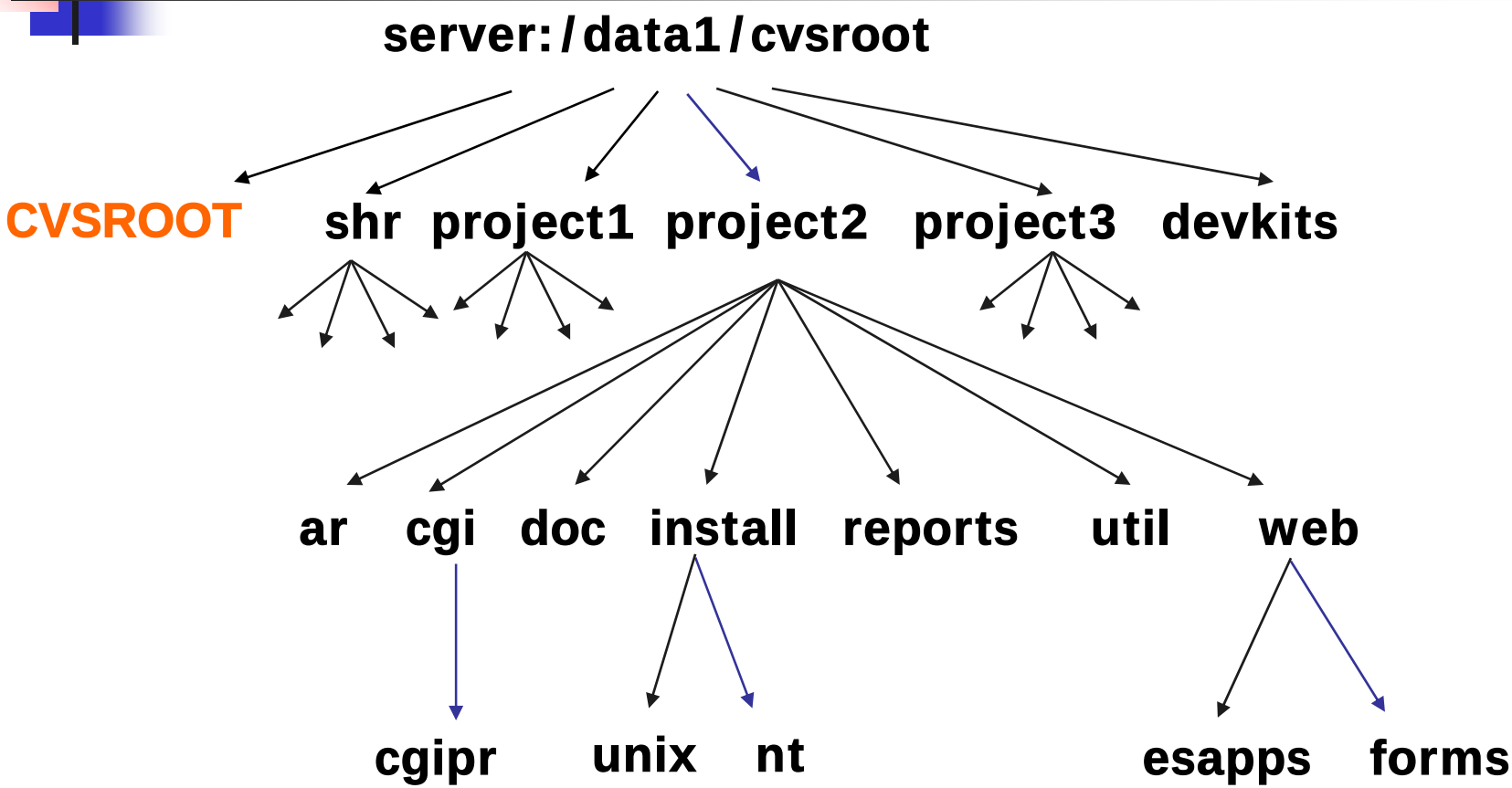




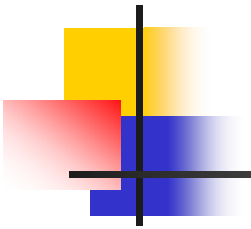
CVS server

- separate server (UNIX or NT)
- no shared filesystems
- a server process per connection

The CVS Repository



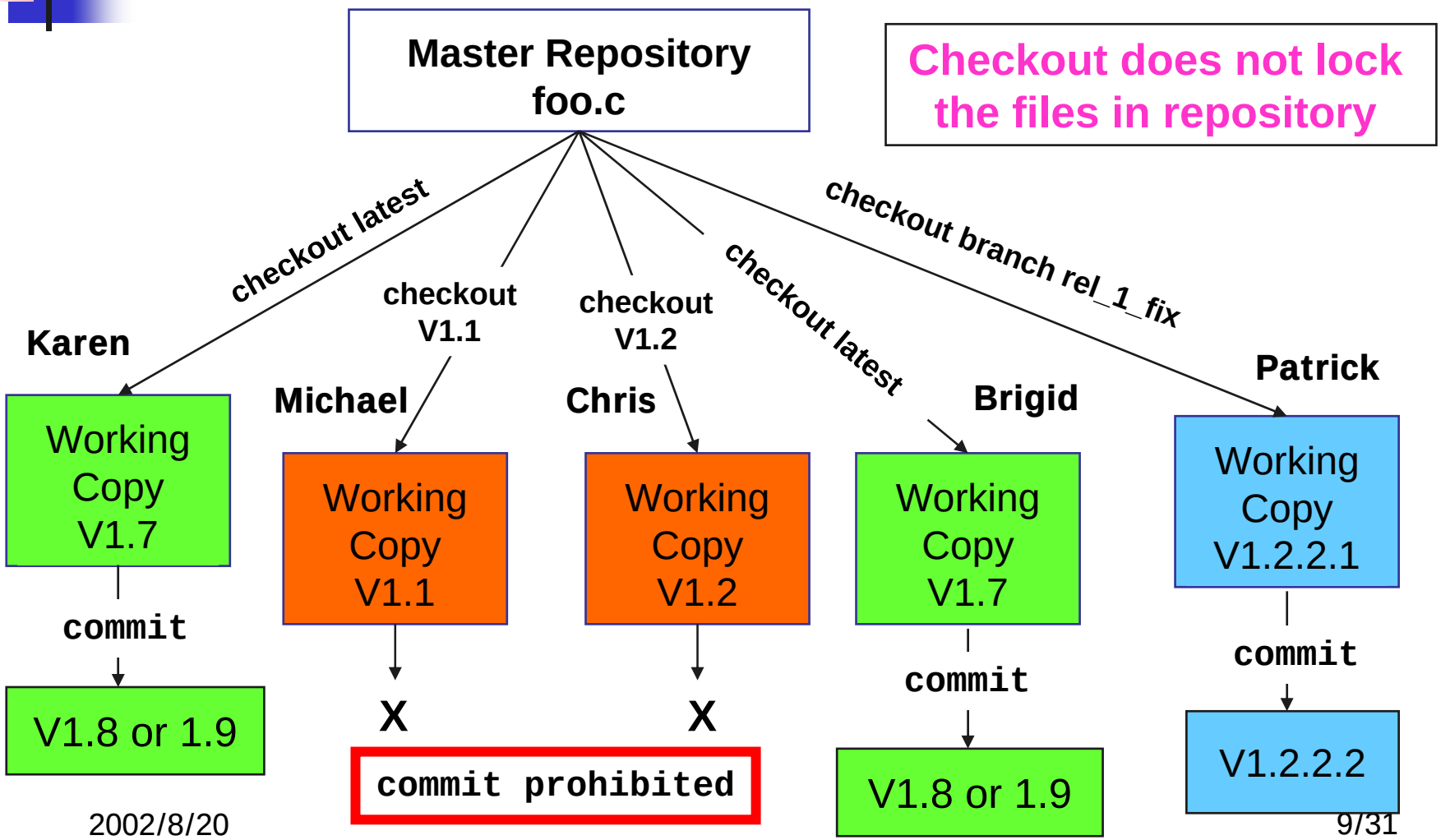
- Resides on a server
- No working files inside the repository



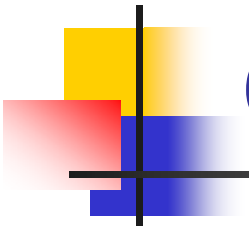
CVS Usage Model
Checkout, Commit, Update

- **Checkout**
  - Make private copy in working directory
  - Can check out anywhere
  - Check out multiple copies, multiple versions
- **Commit**
  - Commit changes to the repository when finished
  - Working copies must be up to date with repository
- **Update**
  - Bring working copy up to date with repository
  - Merge repository changes (if any) since last check out to local copy.

Concurrent checkout

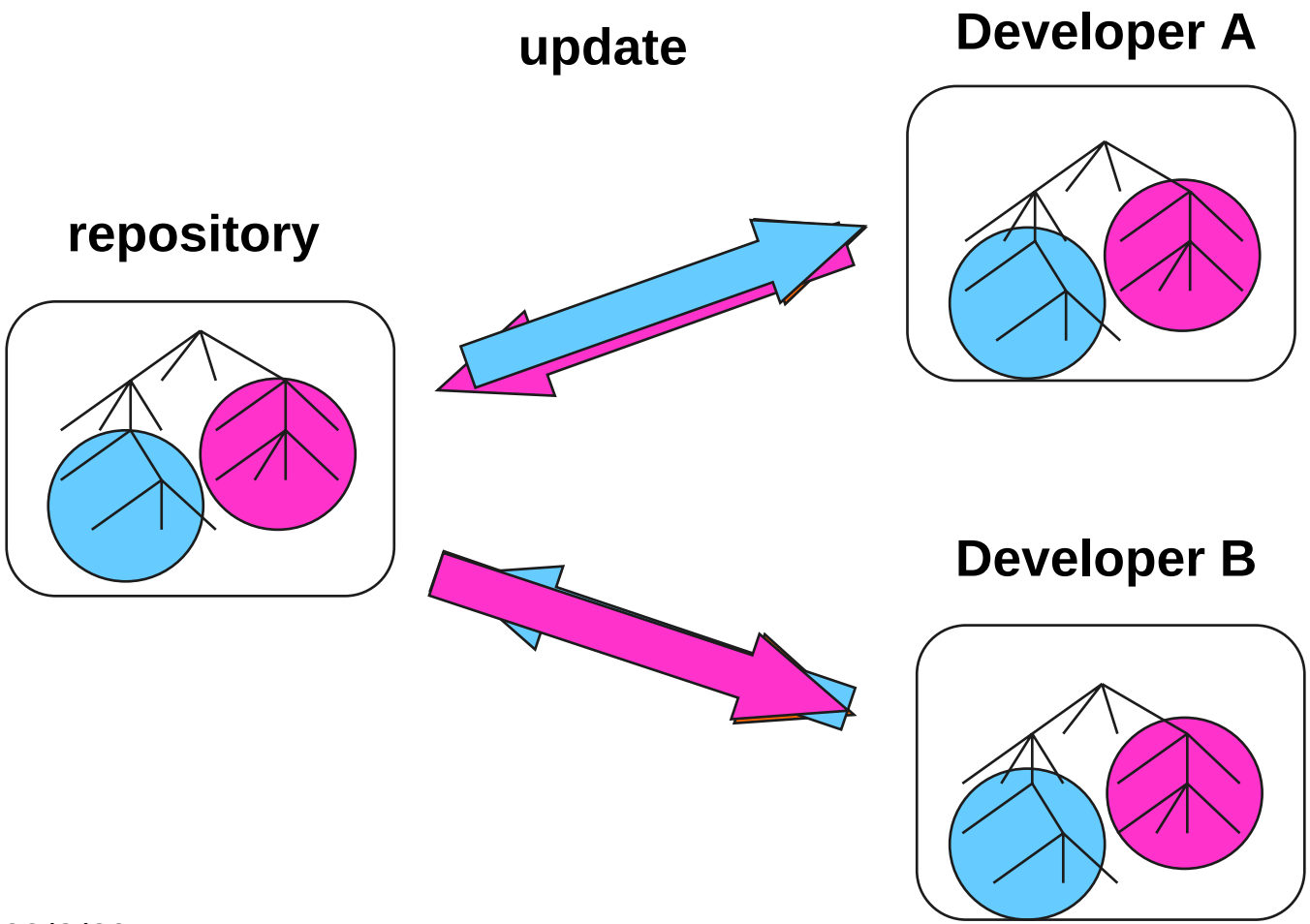


CVS and the Development Cycle

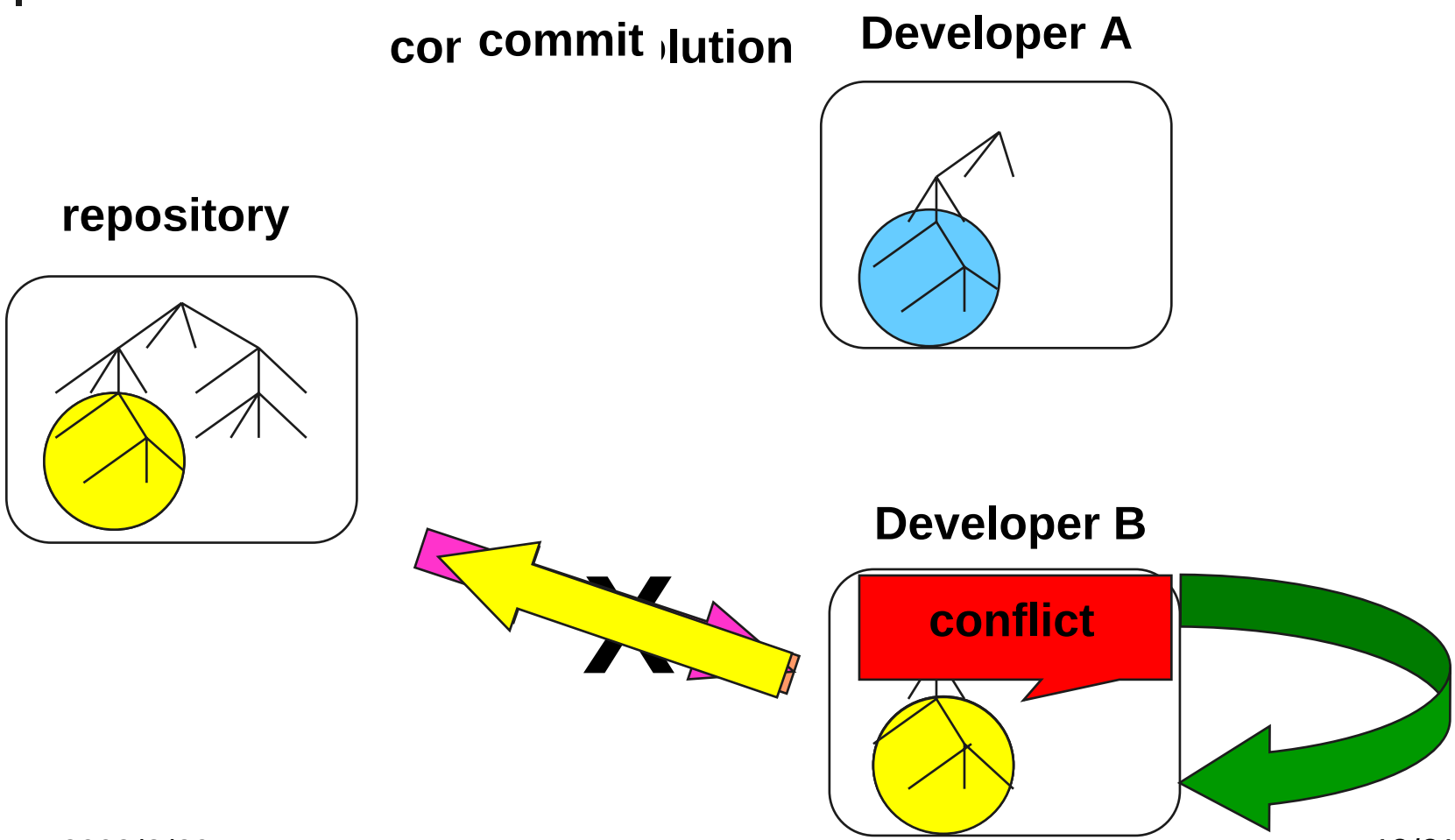


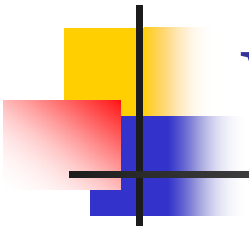
1. Check out source files in working directory.
2. Edit source files.
3. Unit test your code.
4. Update working files to merge in changes from other developers (if necessary).
5. Test again if the sources were merged on step 4.
6. Commit changes.
7. Repeat from step 2 until you have a new release.
8. Tag the release.
9. Submit the module name and release tag for integration build.

Ideal development with CVS



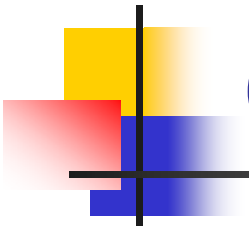
Real development with CVS





When to commit

- Commit to mark a working state that you might want to return to later.
- Commit related files in a single operation. Use a common log message for all the files.
- Commit to backup your sources.
- Commit to share latest changes with other developers.



Conflict

- Conflict happens when CVS cannot merge differences between local copy and repository one at cvs update.
- Conflict indicates
  - an overlap in the source text changes
  - Repository changes are commited by someone else since prior cvs update

Conflict Resolution

- Manually merge the difference and remove conflict markers in source code.

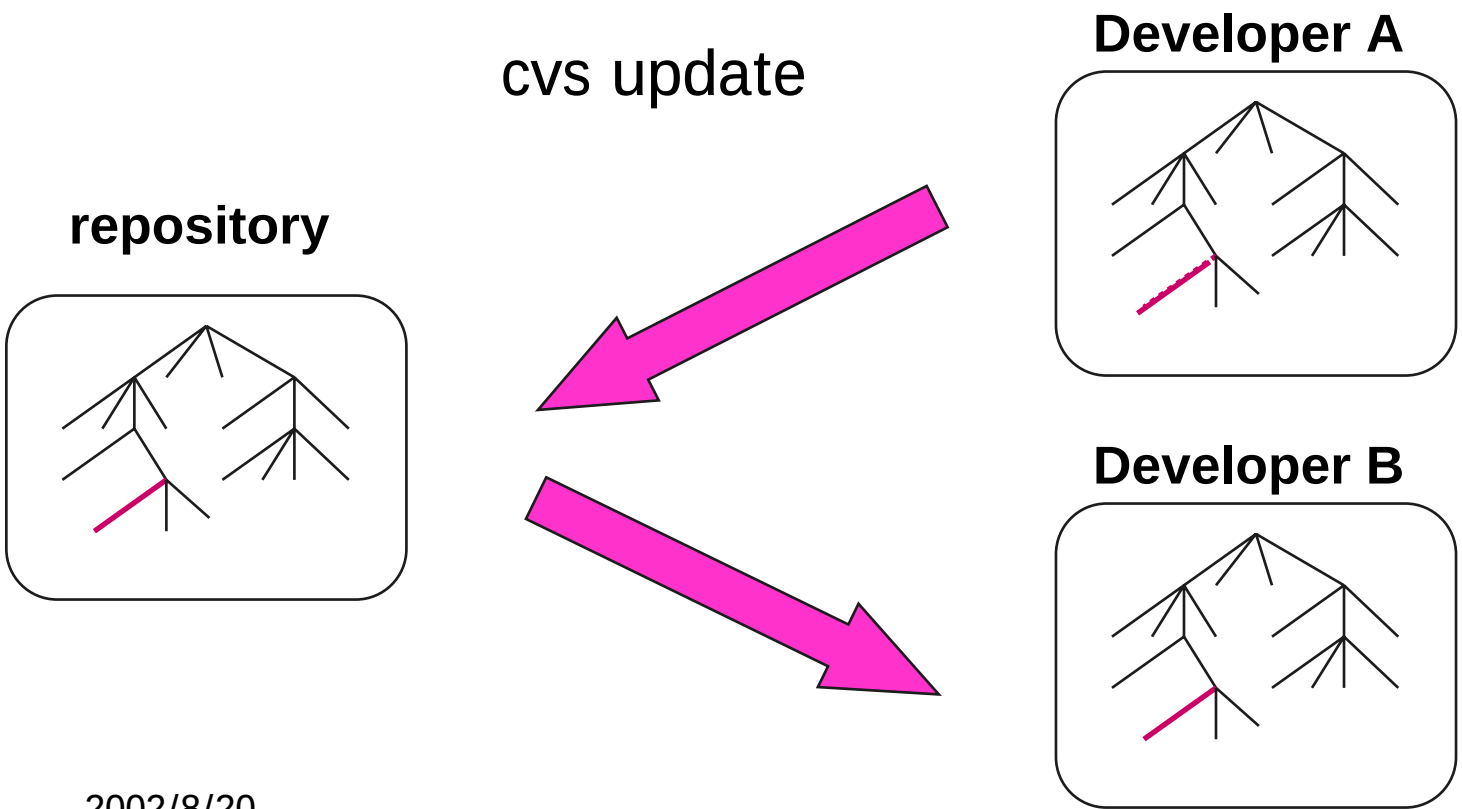
```
<<<<<<< MapReader.java
            if (l > 0)
          continue;
=======
            if (l <= 0)
          break;
>>>>>>> 1.2
```

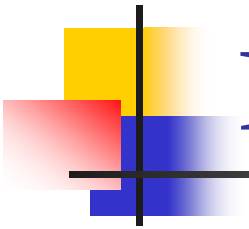
← **Local version**

← **Repository version**

Adding files or directories

- add a new file (local) to the repository
  - cvs add io.c, followed by cvs commit [io.c]
  - others need to cvs update before they see the new file

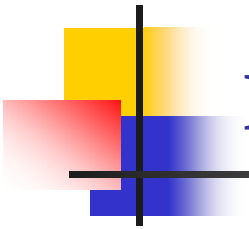




Removing files or directories

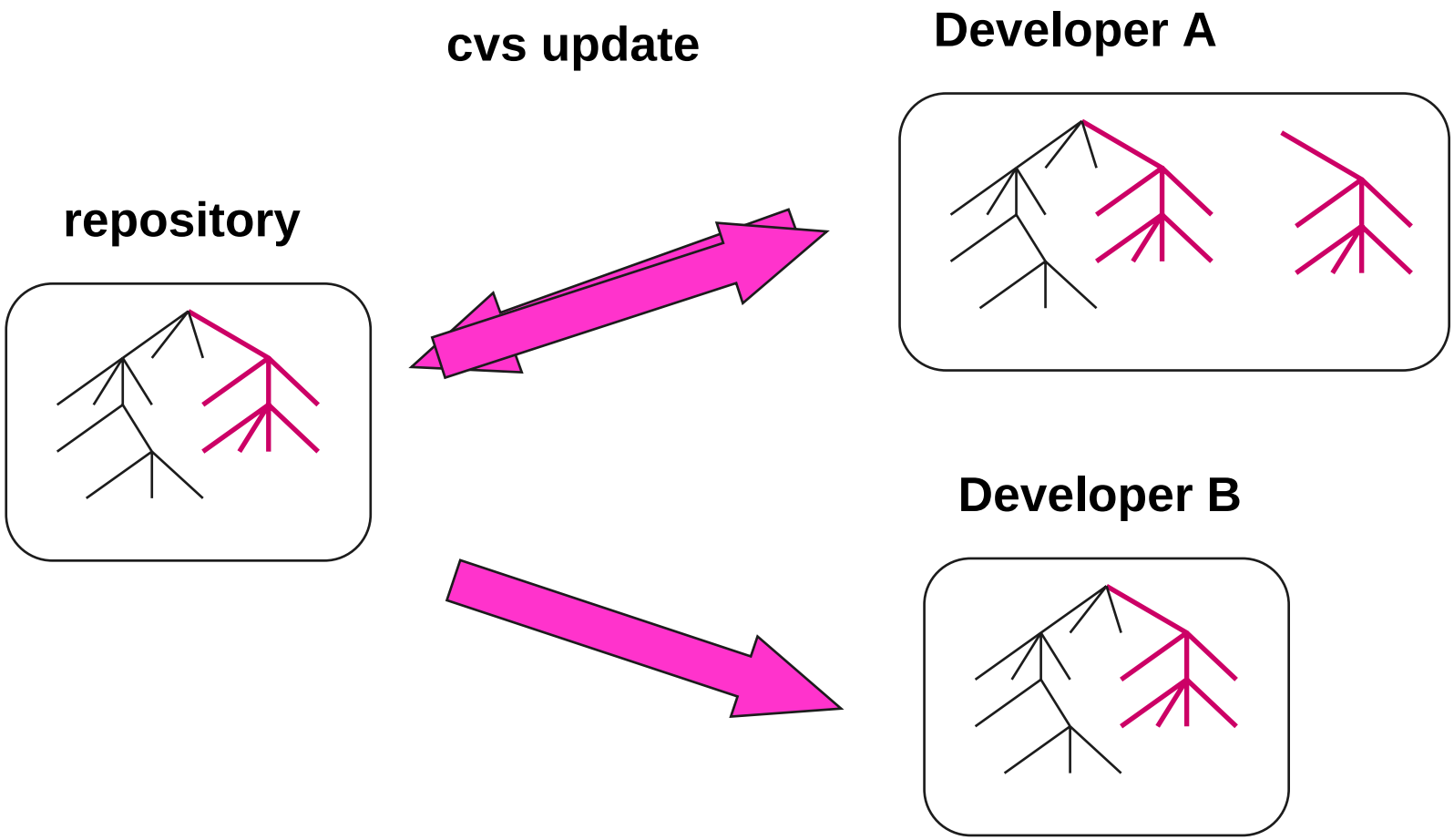
- remove a file:
  - check status: cvs status io.c, then del io.c
  - cvs remove io.c, then cvs commit [io.c]
  - (still in $CVSROOT/*dirs*/Attic/io.c,v)
- cvs add and cvs remove is NEVER recursive. Adding or removing a directory requires manual process.

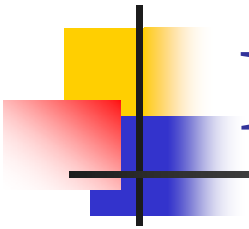
Adding a directory to repository



- cvs import – put existing hierarchy of folders and files into:
  - the repository to create a new module
  - a existing module to create a new subdirectory
- cvs import only affect remote repository
  - Need cvs update to bring the changes to local copy.

CVS import example

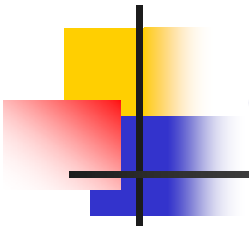




Further topic: renaming files

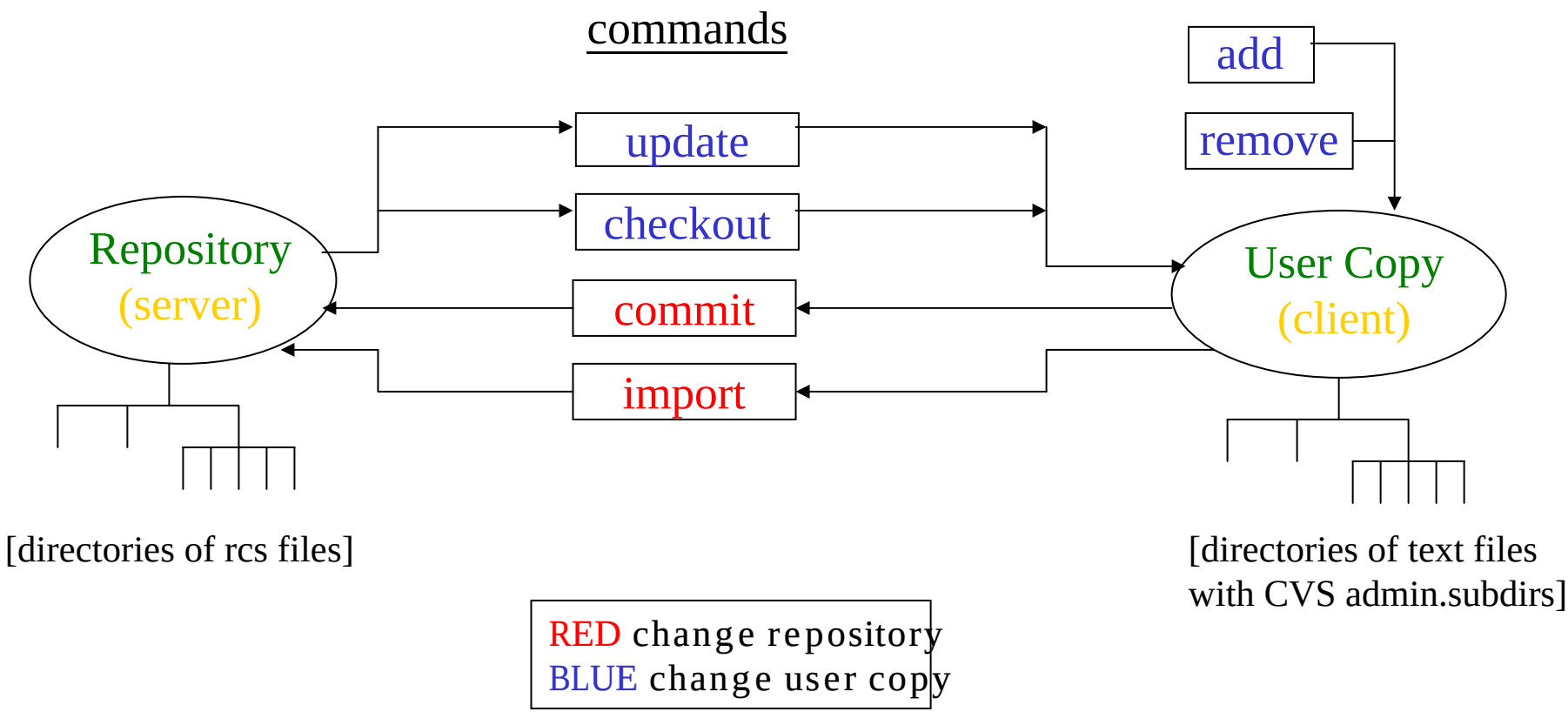
- There is no renaming command in CVS
- The only way is:
  - Rename old filename to new filename
  - cvs remove old filename
  - cvs add new filename
  - cvs commit both new and old filename with message log “Renamed oldname to newname”
- Drawback
  - To access log or retrieve old file, old filename must be supplied. (hence message log is important)

Further topic: Moving directories



- Not supported in CVS
- Manually move whole hierarchy one by one with cvs add and cvs remove.

CVS Operation Diagram



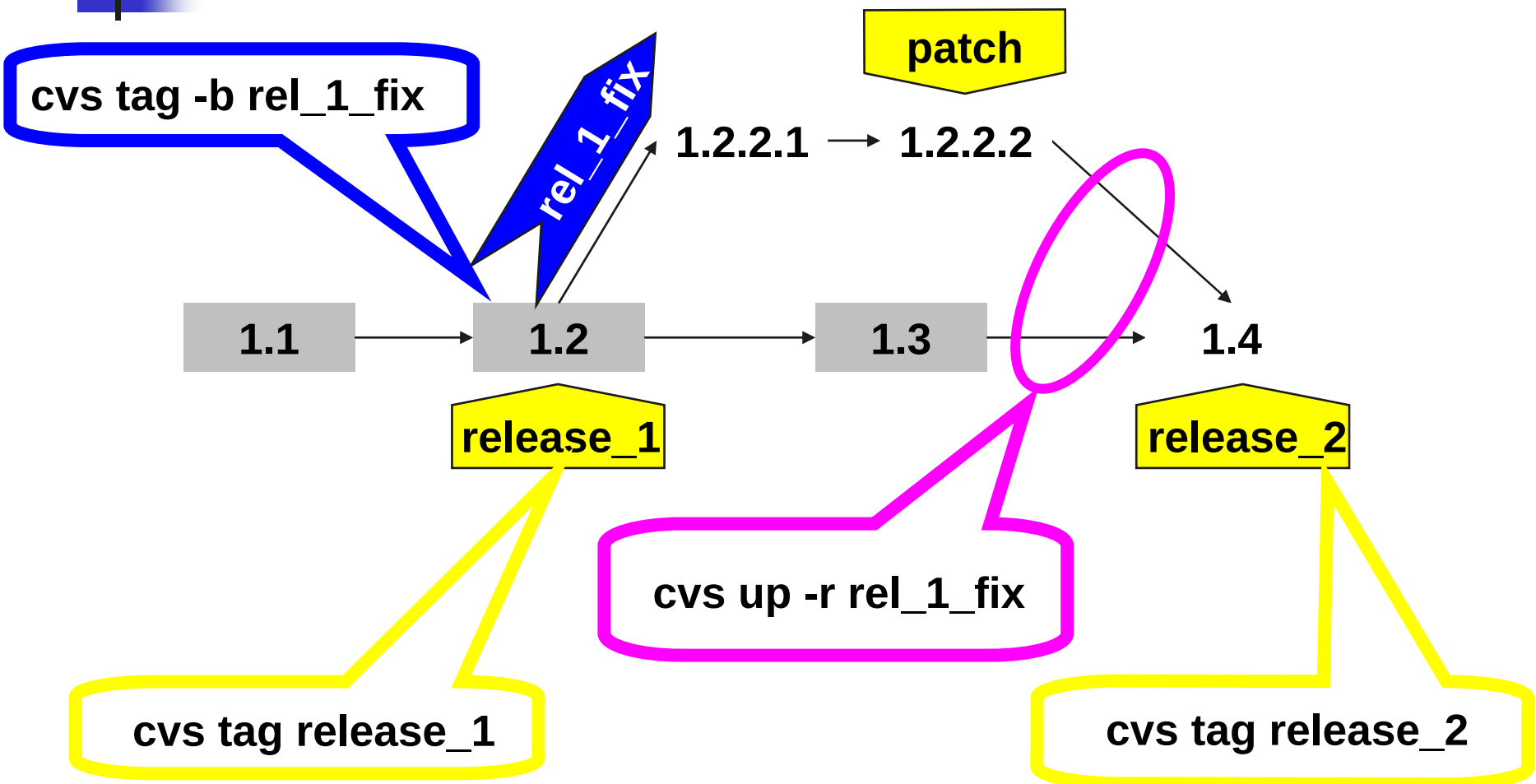
Tagging –
Create a snapshot or release on a repository

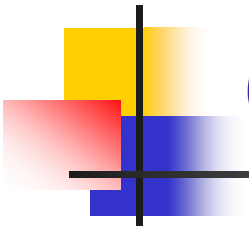
- cvs tag rel1_1 *dir*
  - creates a snapshot called rel1_1
  - consists of all versions in *dir* (usually '.')
- cvs checkout –r rel_1_1 can reproduce the snapshot at anytime.

```
file1   file2   file3   file4   file5

1.1     1.1     1.1     1.1  /--1.1*  <-*- rel1_1
1.2*-   1.2     1.2    -1.2*-
1.3  \- 1.3*-   1.3   / 1.3
1.4           \ 1.4  /  1.4
               \-1.5*-   1.5
                 1.6
```

Branching – multiple lines of development

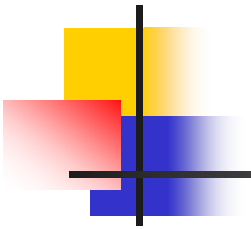




Create branch if you need ...

- to create sustaining (patch) releases
- to have multiple development lines from a single repository
- to do experimental development to merge later or forget about it
- to keep temporary state of development without affecting builds
- **ultimately: merge back**

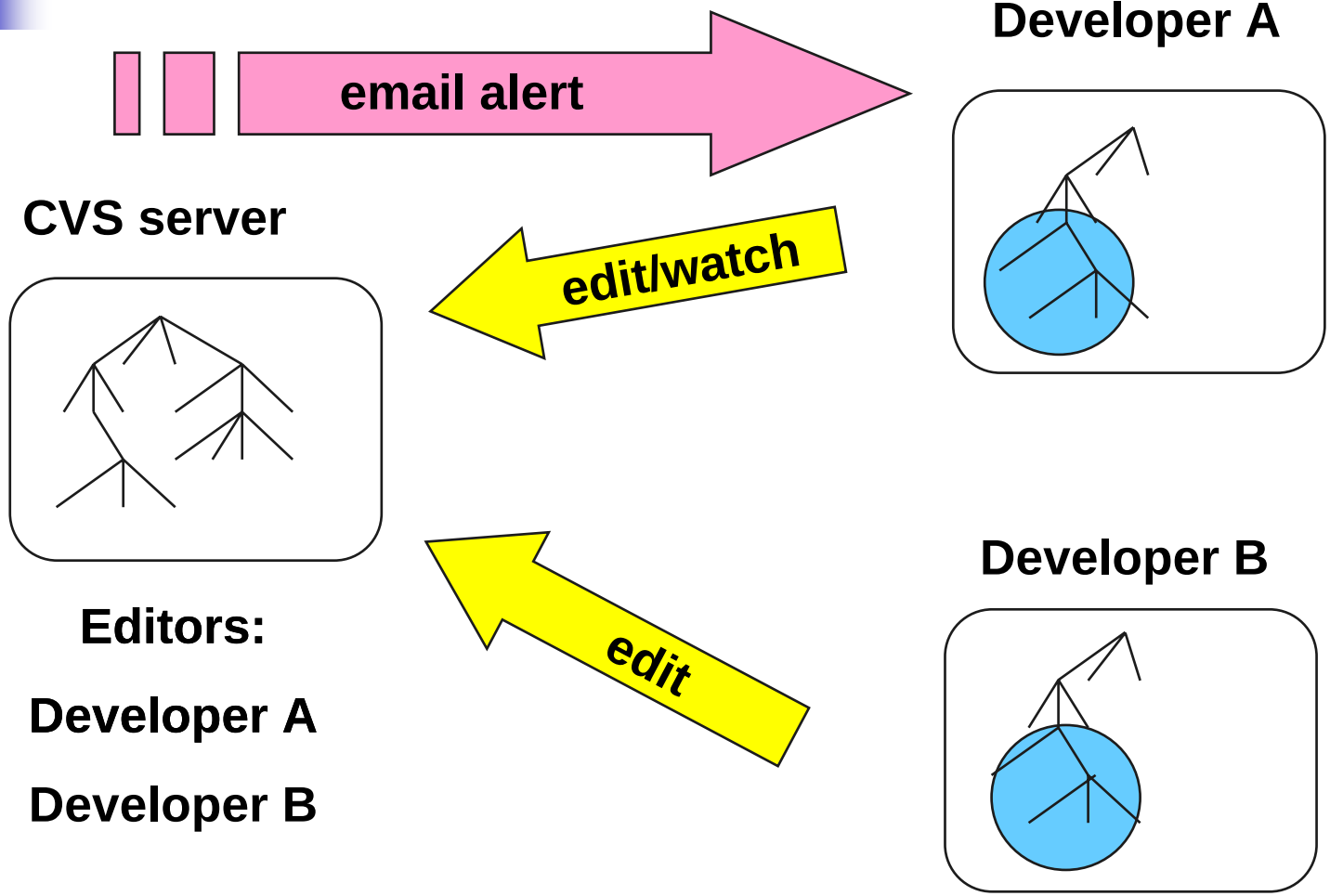
Reserved Checkouts and CVS

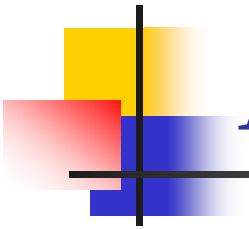


Exclusive file locking prevents parallel development and is not recommended for plain text files

- **advisory locks**: implemented via ***cvs edit*** and ***cvs watch***. Get notification when someone edits or commits the file.
- **exclusive locks** (RCS style): implemented via ***cvs admin***. You cannot commit unless you've locked the file. One lock per file per branch.

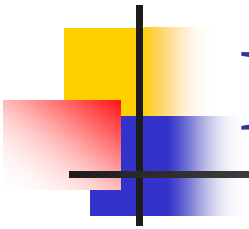
Advisory locks





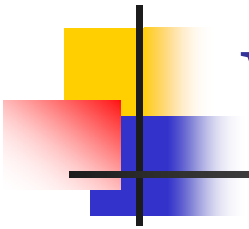
Advisory lock commands

- cvs watch on (off) *files*
  - users must cvs edit *file* before modifying
- cvs watch add (remove)
  - adds current user to those to be notified
- cvs [watchers | editors] *file*
  - See who is [watching | editing] *file*



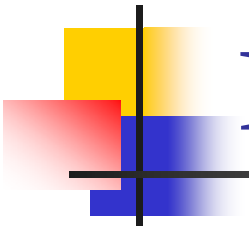
Introduction to WinCVS

- WinCVS is a GUI frontend
  - Sit on top of CVS command-line tool.
    - Command response is still in text-mode when you issue a cvs command.
    - Some command is not available through the GUI interface, knowing how to issue command to CVS command-line tool is sometimes required.
  - Provide a client view of repository as CVS does
    - Will not tell you changes in the repository until you do a cvs update or cvs query.



WinCVS demo time

- Configuration
- Main screen
- Checking out the sources
- Viewing source history
- Diff
- Commit
- Update
- Tag



Reference

- Document
  - Per Cederqvist et al, Version Management with CVS
  - Don Harper, WinCVS 1.3 User Guide
  - WinCVS Daily User Guide
- Web Link
  - http://www.cvshome.org
  - http://www.cvsgui.org