CSE 4513

Lec – 12

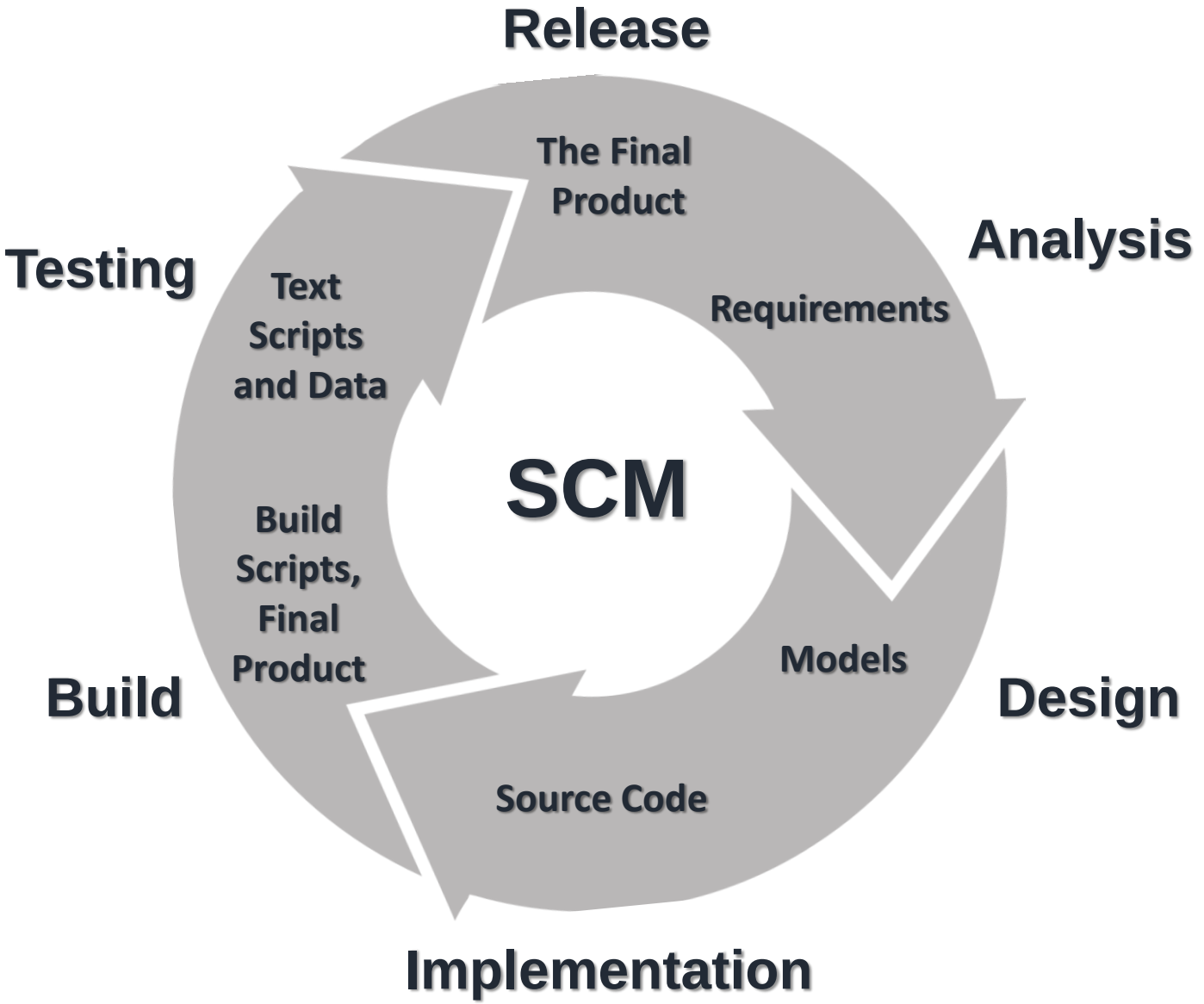
Version Control System (VCS)

SOFTWARE CONFIGURATION MANAGEMENT (SCM)

- Version Control ≈ Software Configuration Management (SCM)
  - helps organizations to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.
  - Keeps track of what is happening in the project over the time
  - Solves conflicts in the changes

The primary **goal** of the SCM process is to increase productivity with minimal mistakes.

SCM and the Software Development Lifecycle



VERSION CONTROL SYSTEMS (VCS)

Manages Different Versions of the Same File / Document

Well known products

- ✓ Concurrent Version System(CVS), Subversion (SVN) – free, open source
- ✓ Git, Mercurial – distributed, free, open source
- ✓ Perforce, Microsoft TFS – commercial

VERSION CONTROL SYSTEMS (VCS)

- **Features & Capabilities**
  - File versions control
  - Merge and differences search
  - Branch creation and deletion
  - Compare and merge version branches
  - Revert code to previous versions
  - File locking
  - Code version management (e.g. conflict resolution)
  - Parallel development streams & branches
  - Console and GUI clients

N95 3M Product Number 8210

N95 3M Product Number 8110s

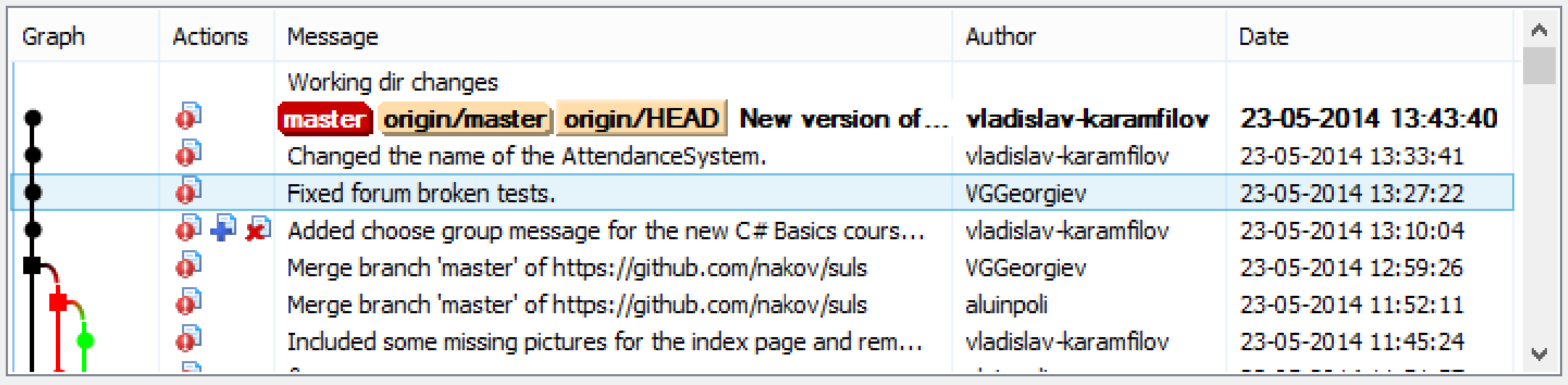
Version Control Systems (VCS)

- Constantly used in software engineering
  - ✓ During the software development
  - ✓ While working with documents
- Changes are identified with an increment of the version number
  - ✓ for example 1.0, 2.0, 2.17
- Version numbers are historically linked with the person who created them
  - ✓ Full change logs are kept

Revision	Actions	Author	Date	Message
99		nakov	March 24, 2014 21:54:09	bug fix
98		nakov	March 24, 2014 21:52:02	bug fix
97		vladkaramfilov	March 24, 2014 15:38:12	Uploaded test RAR file.
96		nakov	March 22, 2014 19:12:56	good progress: loops home…
95		nakov	March 22, 2014 11:46:18	typo fixed
94		nakov	March 22, 2014 11:44:36	Initial draft: loops homework
93		nakov	March 22, 2014 11:44:12	Loops lecture finished (exer…
92		nakov	March 22, 2014 09:49:09	removed unused file
91		nakov	March 22, 2014 09:48:27	Added TODO

VERSION CONTROL SYSTEMS (VCS)

- Systems for version control keep a complete change log (history)
  - ✓The date and hour of every change
  - ✓The user who made the change
  - ✓The files changed + old and new version
- Old versions can be retrieved, examined and compared
- It is possible to return to an old version (revert)



VOCABULARY

- Repository (source control repository)
  - A server that stores the files (documents)
  - Keeps a change log
- Revision, Version
  - Individual version (state) of a document that is a result of multiple changes
- Check-Out, Clone
  - Retrieves a working copy of the files from a remote repository into a local directory
  - It is possible to lock the files

VOCABULARY (2)

- Change
  - A modification to a local file (document) that is under version control
- Change Set / Change List
  - A set of changes to multiple files that are going to be committed at the same time
- Commit, Check-In
  - Submits the changes made from the local working copy to the repository
  - Automatically creates a new version
  - Conflicts may occur!

VOCABULARY (3)

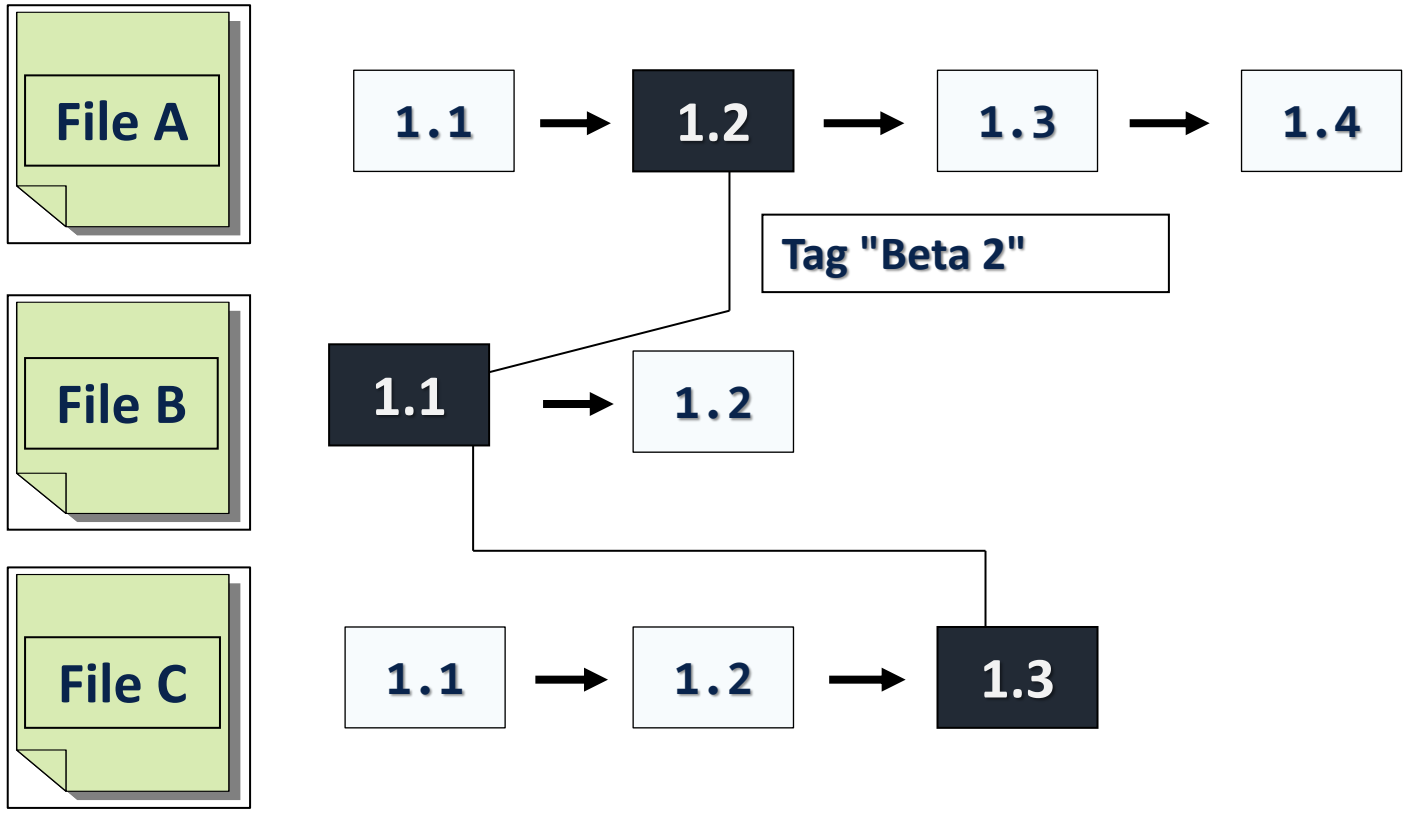
- Conflict
  - The simultaneous change to a certain file by multiple users
  - Can be solved automatically and manually
- Update, Get Latest Version, Fetch / Pull
  - Download the latest version of the files from the repository to a local working directory + merge conflicting files
- Undo Check-Out, Revert / Undo Changes
  - Cancels the local changes
  - Restores their state from the repository

VOCABULARY (4)

- Merge
  - Combines the changes to a file changed locally and simultaneously in the repository
  - Can be automated in most cases
- Label / Tag
  - Labels mark with a name a group of files in a given version
  - For example a release
- Branch / Branching
  - Division of the repositories in a number of separate workflows

TAGS

- Allows us to give a name to a group of files in a certain version



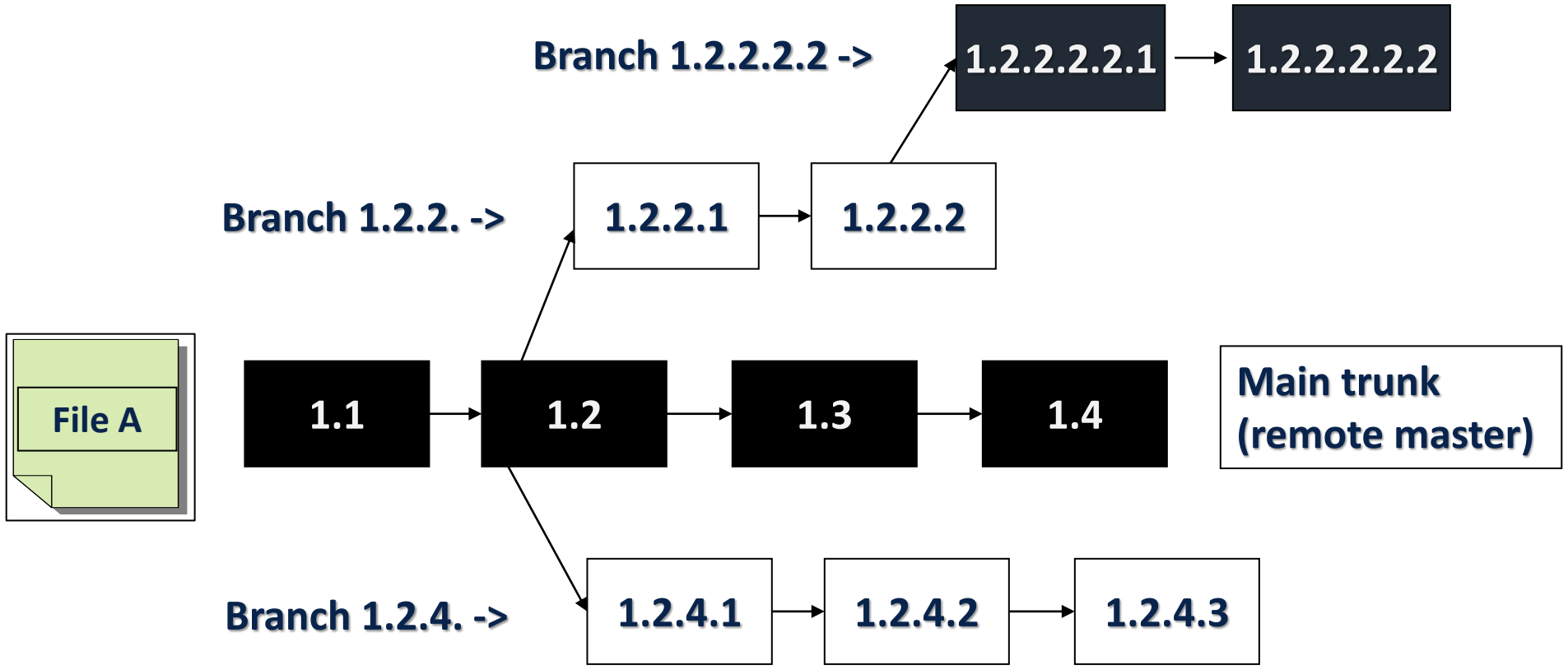
BRANCHING

- Branching allows splitting the development line into separate branches
  - Different developers work in different branches
- Branching is suitable for:
  - Development of new feature or fix in a new version of the product (for example version 2.0)
    - Features are invisible in the main development line Until merged with it
  - You can still make changes in the older version (for example version 1.0.1)

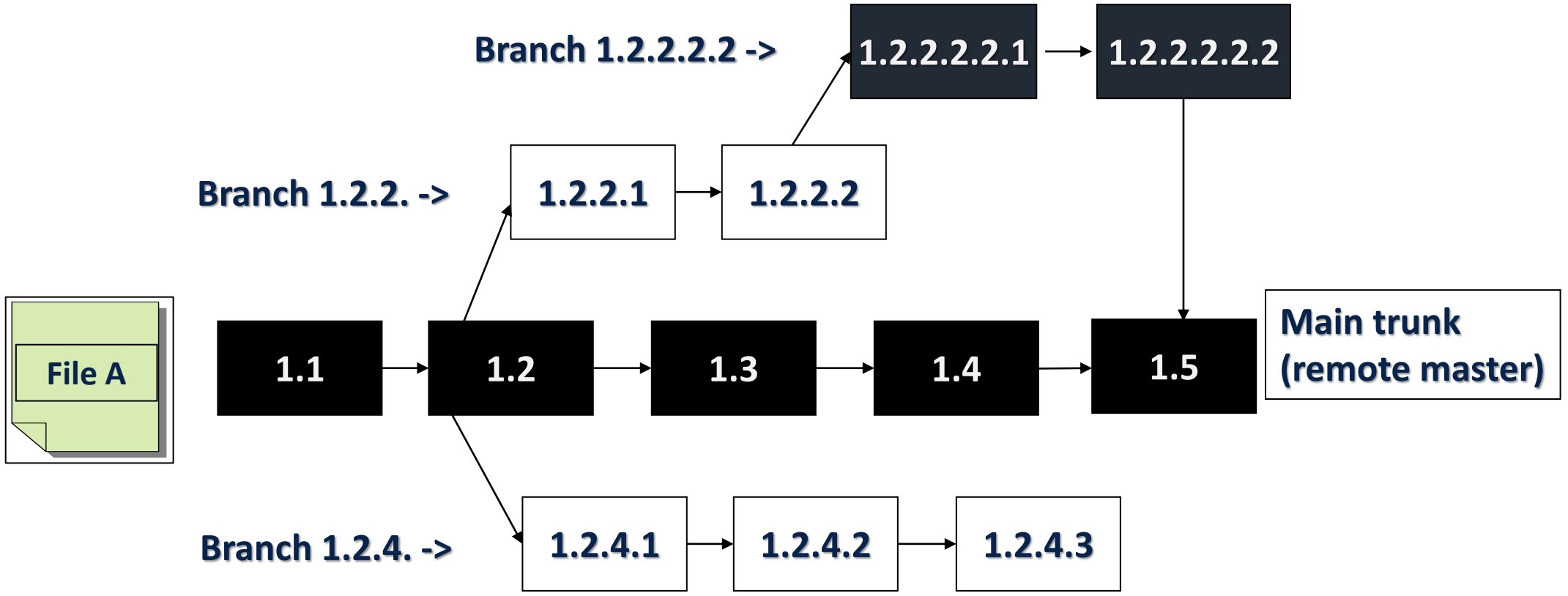
Merging Branches

- Some companies work in separate branches
  - For each new feature / fix / task
- Once a feature / fix / task is completed
  - It is tested locally and committed in its branch
- Finally it is merged into the main development line
  - Merging is done locally
  - Conflicts are resolved locally
  - If the merge is tested and works well, it is integrated back in the main development line

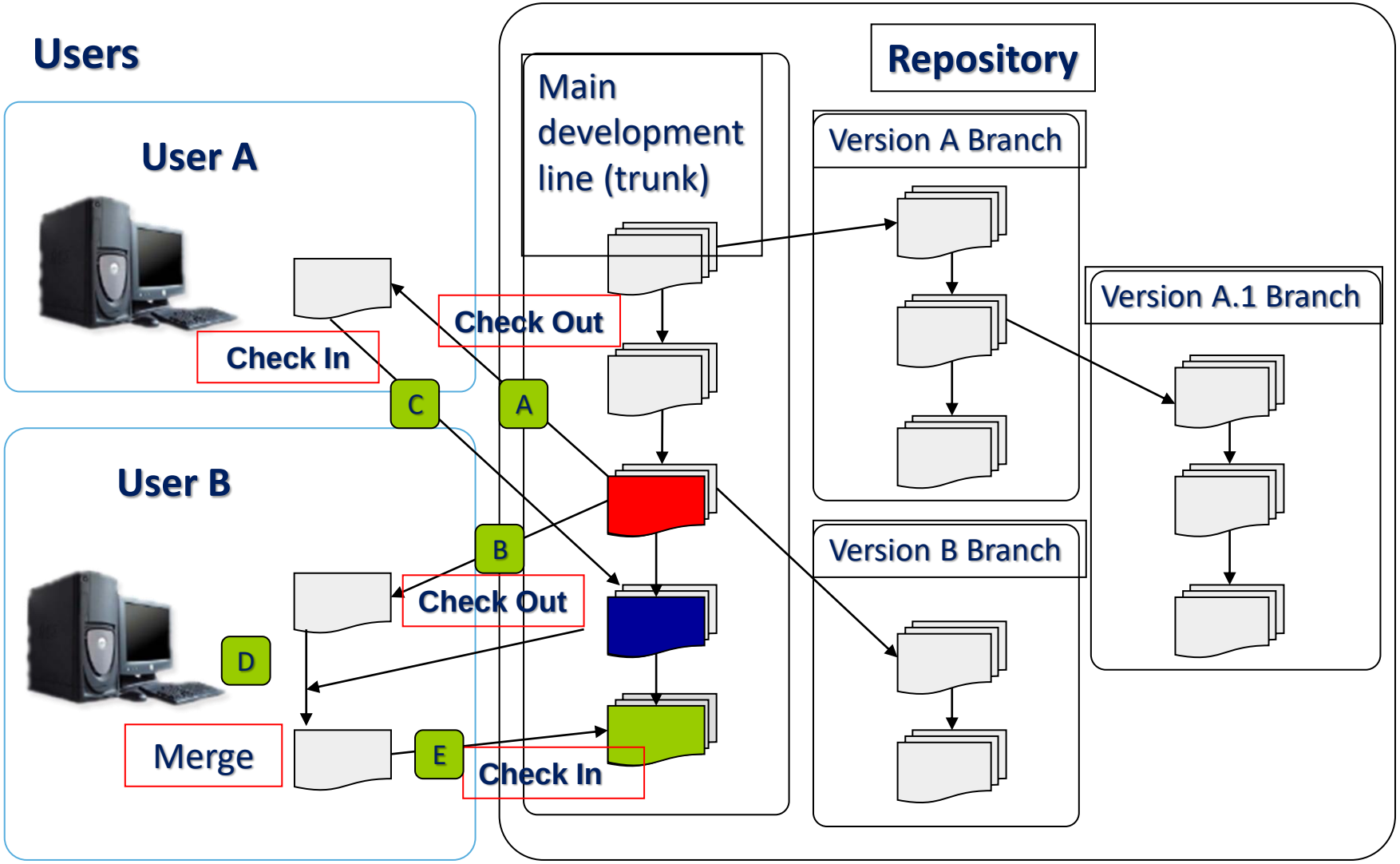
BRANCHING – EXAMPLE



Merging Branches – Example

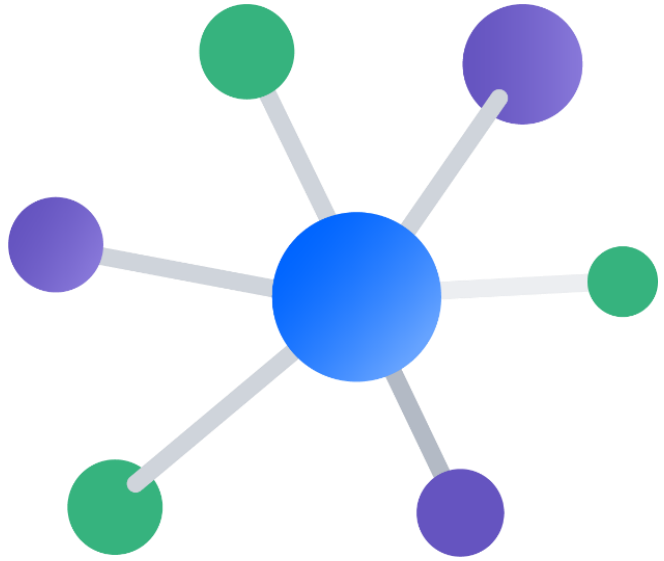


Version Control: Typical Scenario

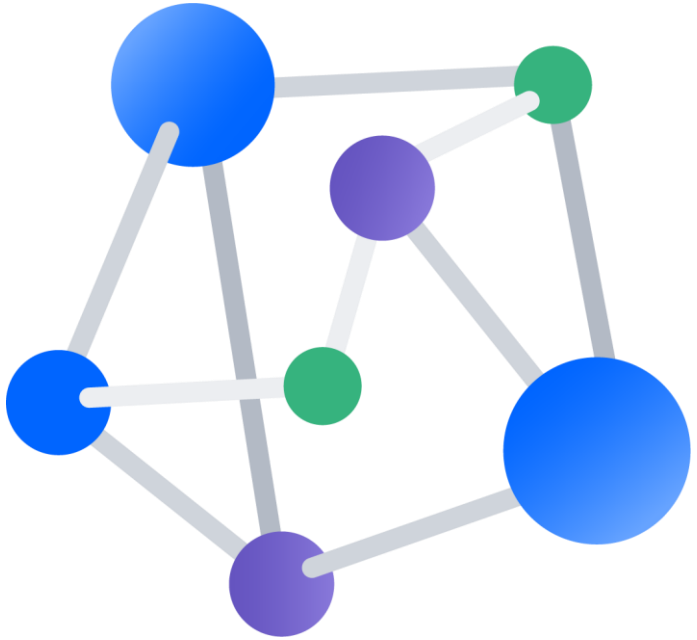


VERSIONING MODELS

- ✓VCS tools come in two primary types of remote architecture
  - ➢Centralized
    - Lock-Modify-Unlock
    - Copy-Modify-Merge
  - ➢Distributed.



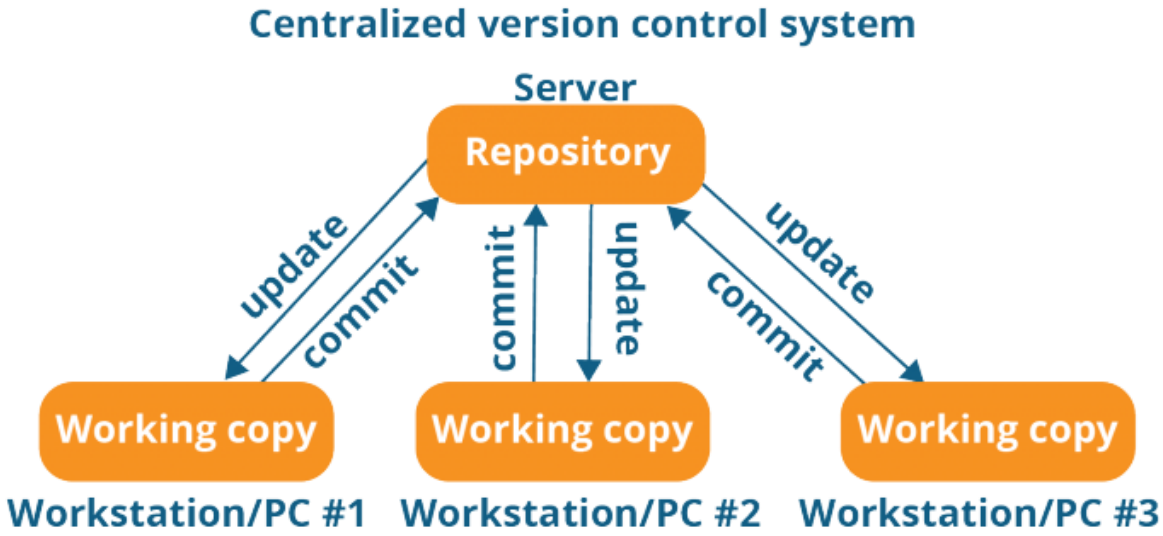
Centralized



Distributed.

CENTRALIZED VERSION CONTROL

based on the idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy.

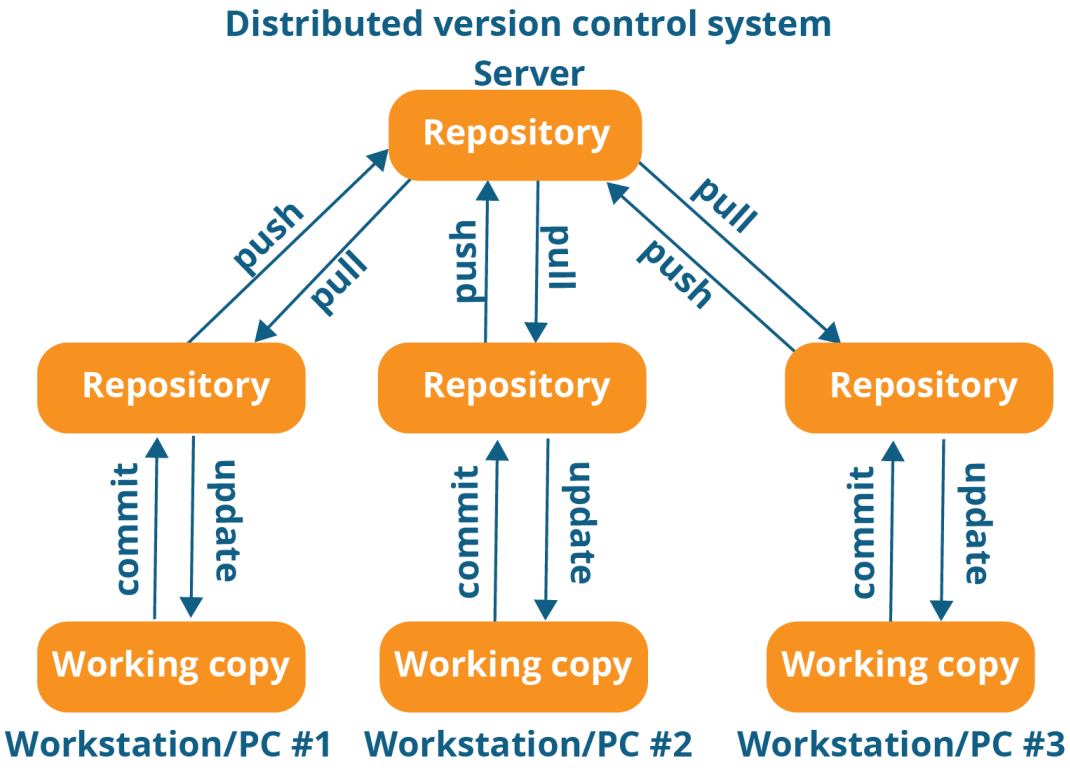


Main benefits:

- Centralized systems are typically easier to understand and use
- You can grant access level control on directory level

Distributed Version Control

In distributed version control, every developer “clones” a copy of a repository and has the full history of the project on their own hard drive. This copy (or “clone”) has all of the metadata of the original.



Main benefits:

- Performance is better
- Branching and merging is much easier
- With it, you don’t need to be connected to the network all the time (complete code repository is stored locally on PC)

VERSIONING MODELS

- Lock-Modify-Unlock
  - Only one user works on a given file at a time
    - No conflicts occur
    - Users wait each other for the locked files → works for small development teams only
  - Examples:
    - Visual SourceSafe (VSS) – old fashioned
  - Lock-modify-unlock is rarely used

VERSIONING MODELS (2)

- Copy-Modify-Merge
  - Users make parallel changes to their own working copies
  - Conflicts are possible when multiple user edit the same file
    - Conflicting changes are merged and the final version emerges (automatic and manual merge)
  - Examples:
    - SVN

Versioning Models (3)

Distributed Version Control

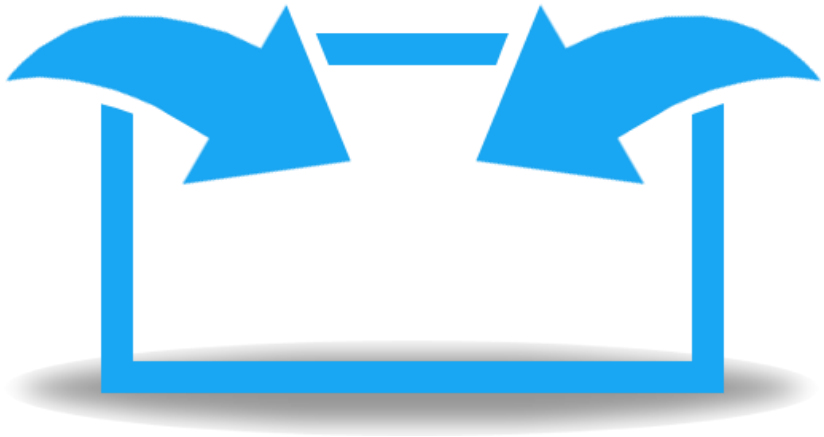
- Users work in their own repository
  - Using the Lock-Modify-Unlock model
  - Local changes are locally committed
  - No concurrency, no local conflicts
- From time to time, the local repository is pushed to the central repository
  - Conflicts are possible and merges often occur
- Example of distributed version control systems:
  - Git

Problems with Locking

- Administrative problems:
  - Someone locks a given file and forgets about it
  - Time is lost while waiting for someone to release a file → works in small teams only
- Unneeded locking of the whole file
  - Different changes are not necessary in conflict
  - Example of non-conflicting changes:
    - **A** works at the beginning of the file
    - **B** works at the end of the file

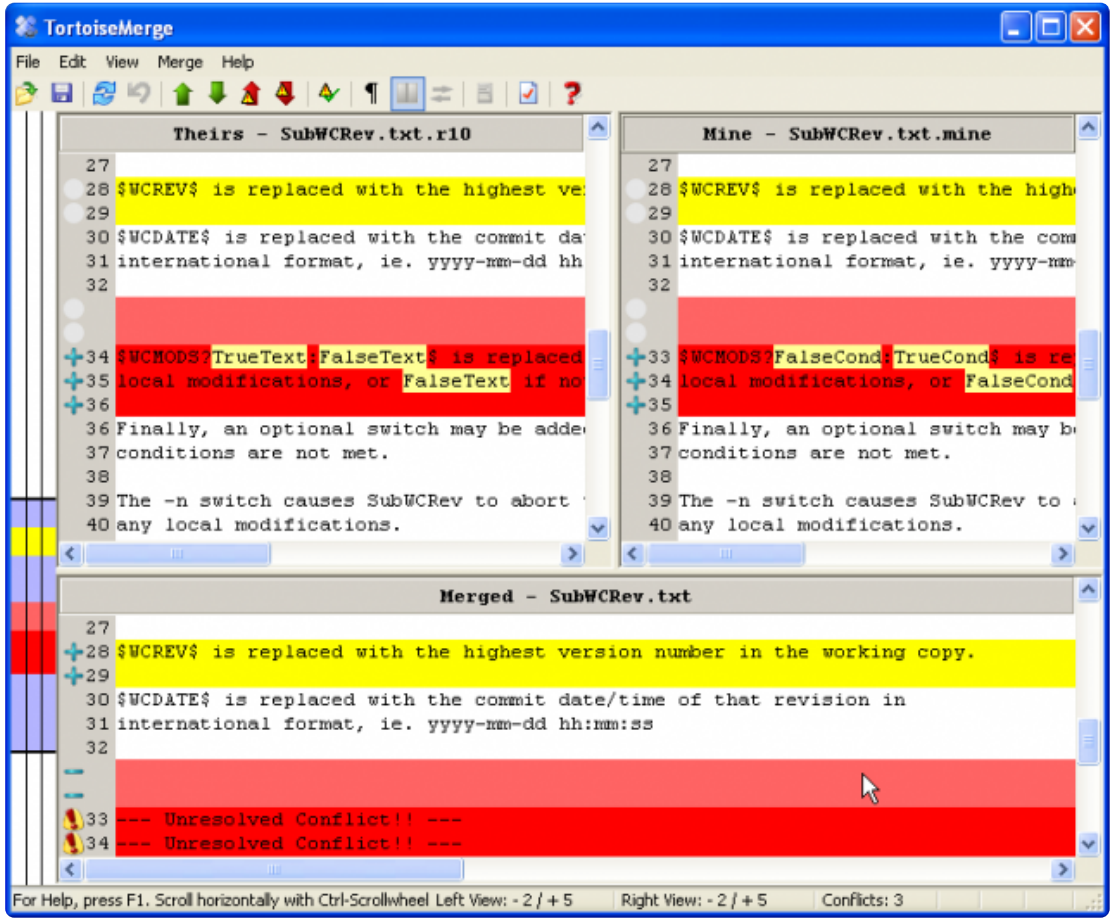
Merging Problems

- When a file is concurrently modified, changes should be merged
  - Merging is hard!
  - It is not always automatic process
- Coordination and responsibility between the developers is required
  - Commit changes as early as finished
  - Do not commit code that does not compile or blocks the work of the others
  - Leave meaningful comments at each commit



File Comparison / Merge Tools

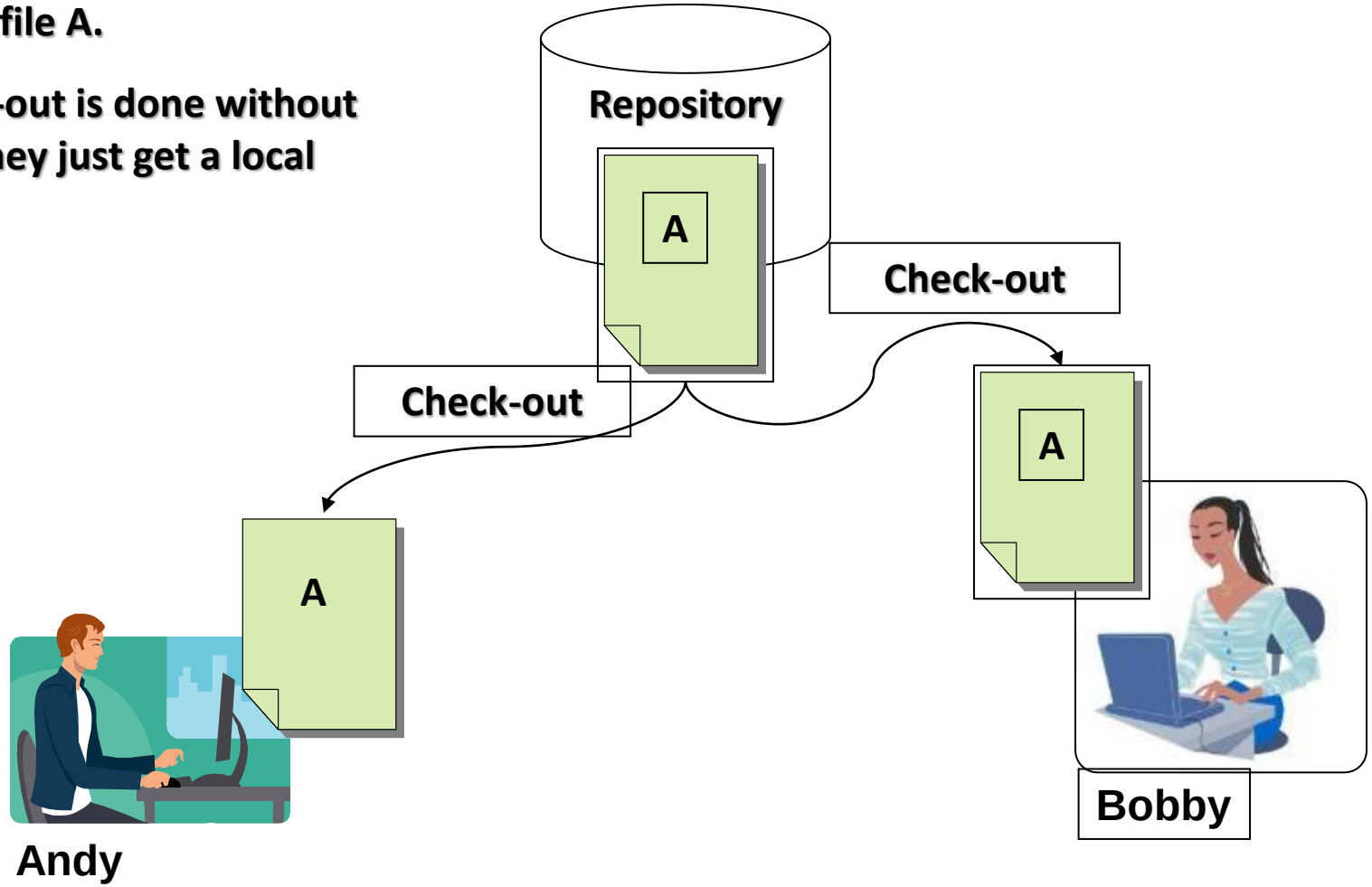
- During manual merge use file comparison
- There are visual comparison / merge tools:
  - TortoiseMerge
  - WinDiff
  - AraxisMerge
  - WinMerge
  - BeyondCompare
  - CompareIt
  - ...



The Lock-Modify-Unlock Model (1)

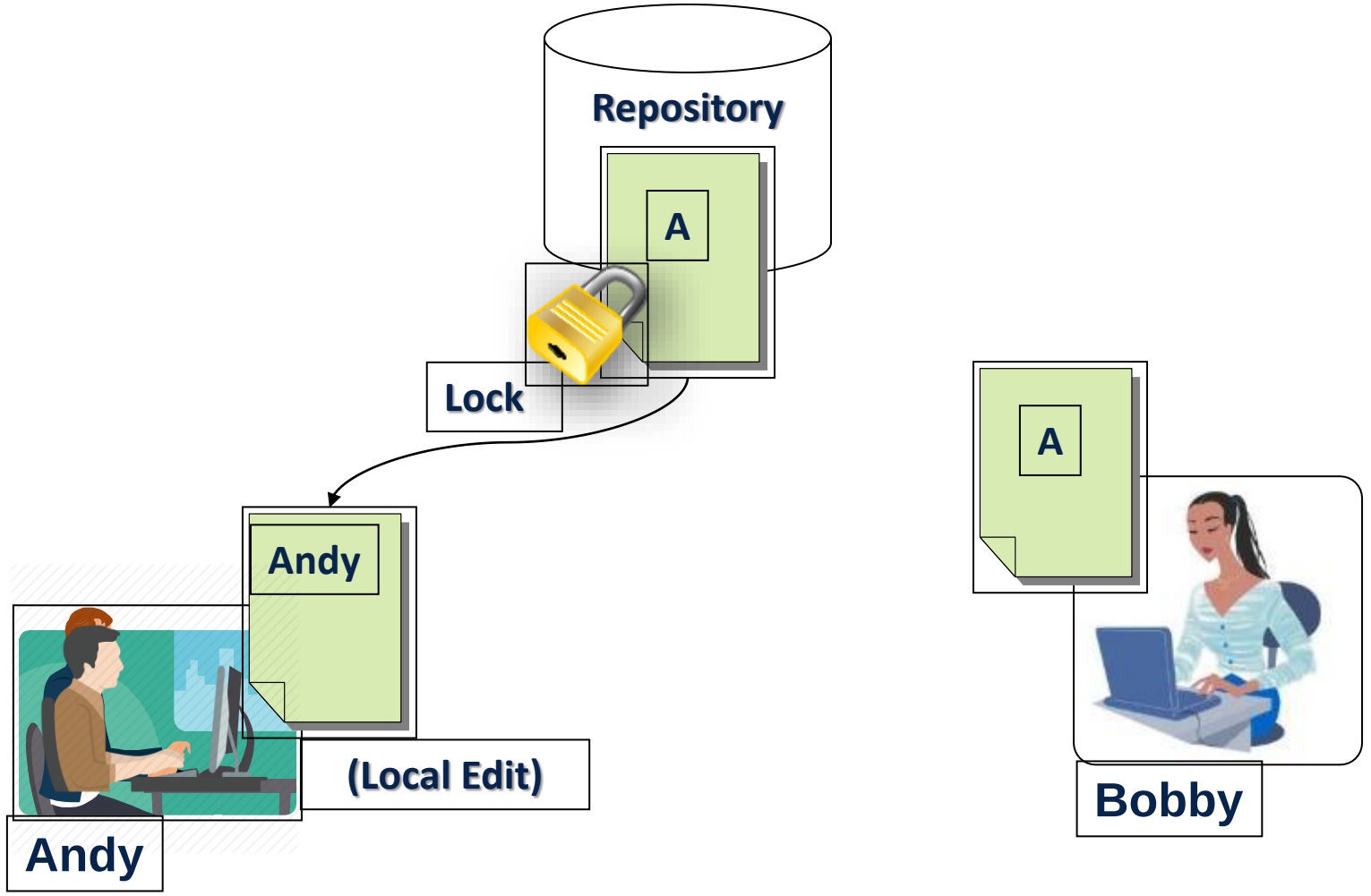
Andy and Bobby check-out file A.

The check-out is done without locking. They just get a local copy.



The Lock-Modify-Unlock Model (2)

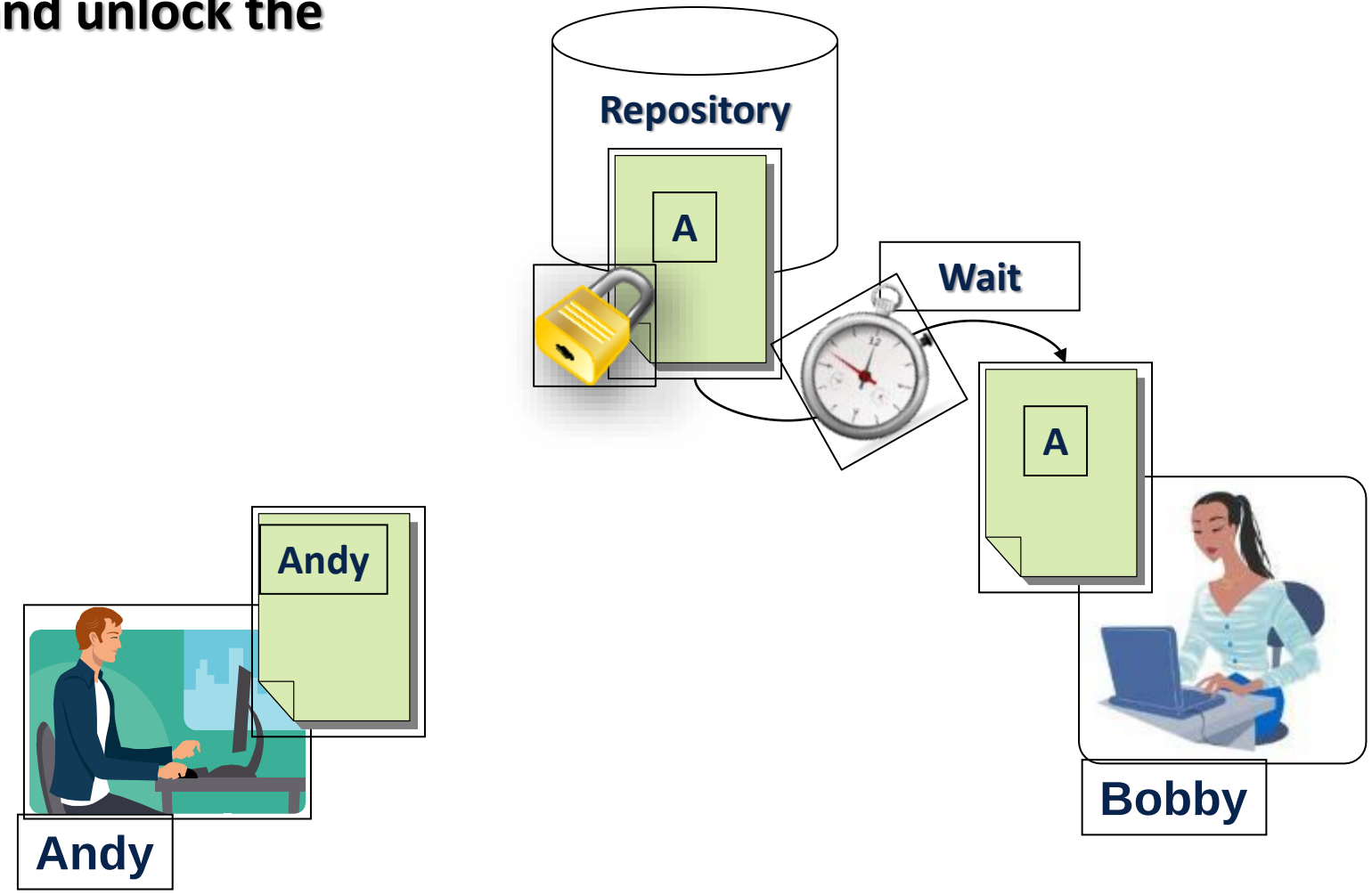
Andy locks file A and begins modifying it.



The Lock-Modify-Unlock Model (3)

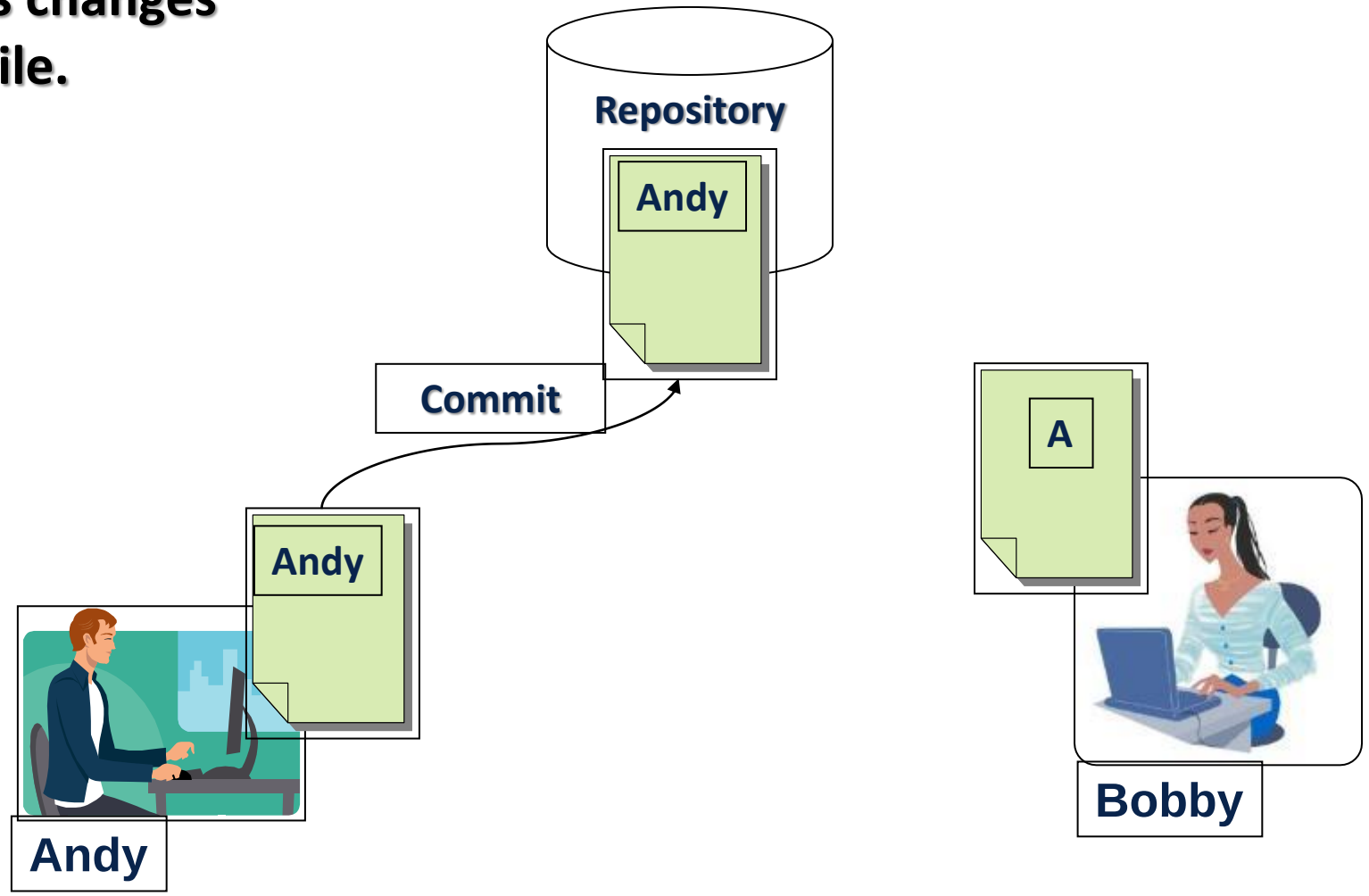
Bobby tries to lock the file too, but she can't.

Bobby waits for Andy to finish and unlock the file.



The Lock-Modify-Unlock Model (4)

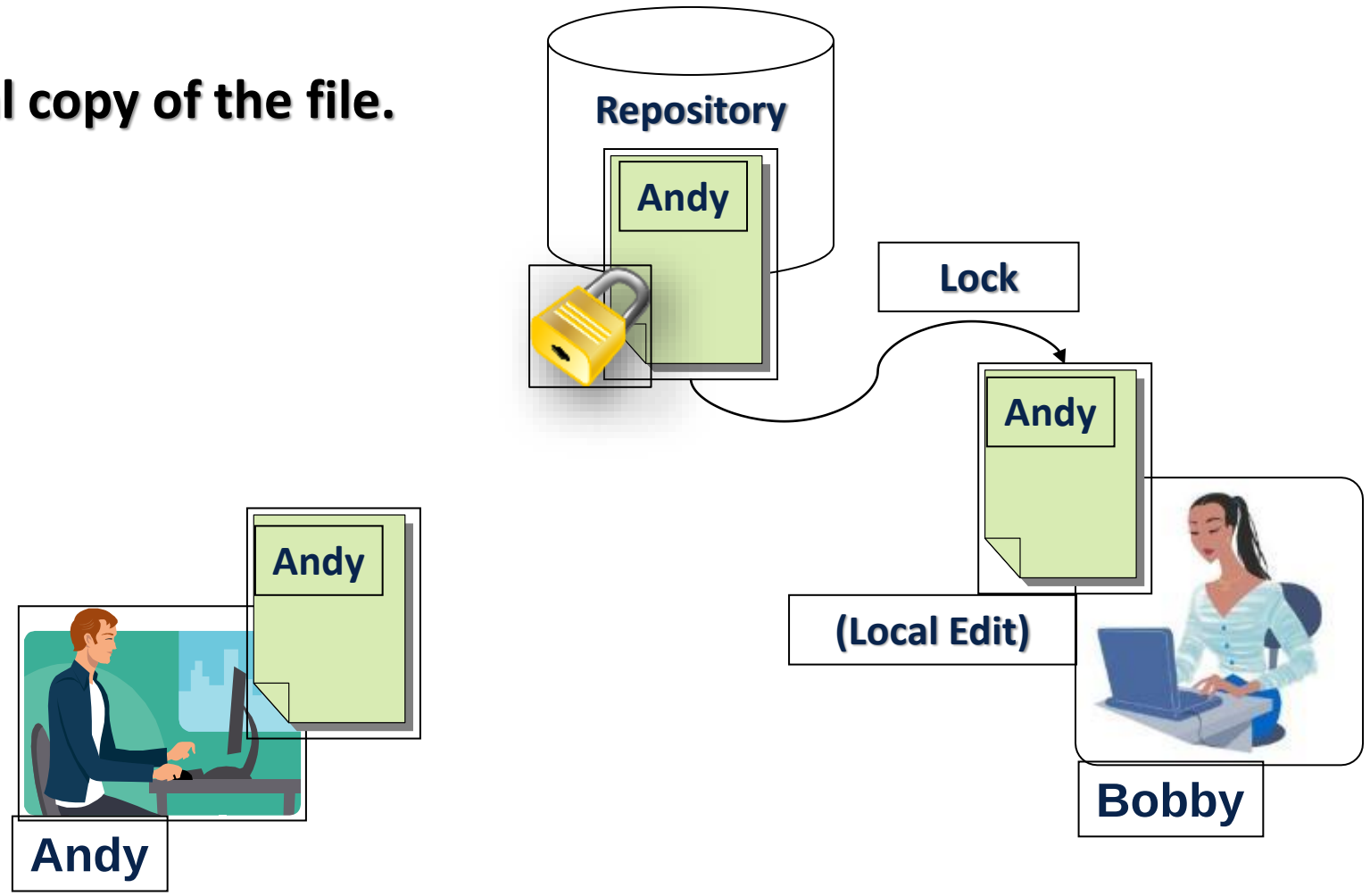
Andy commits his changes and unlocks the file.



The Lock-Modify-Unlock Model (5)

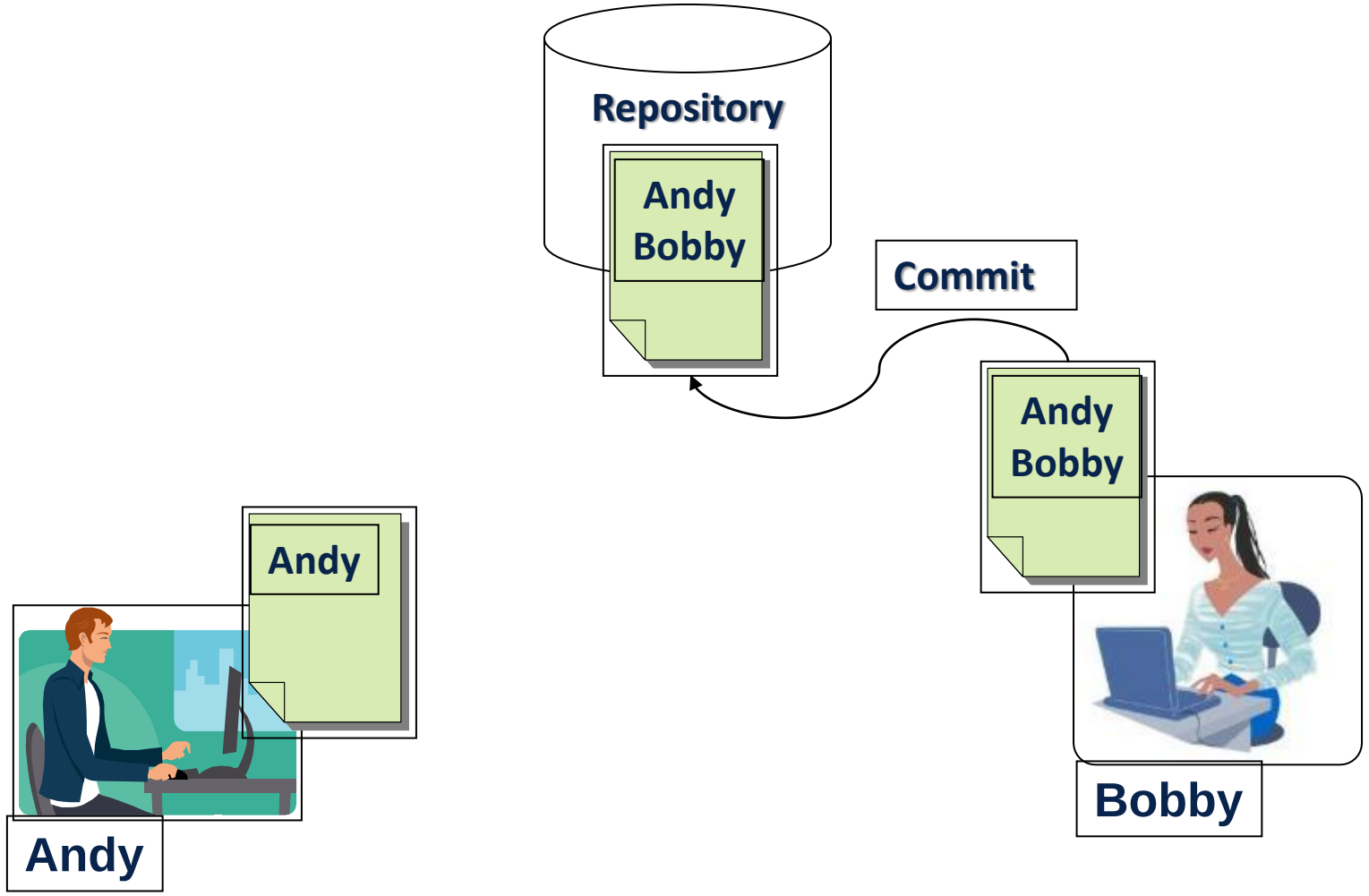
Now Bobby can take the modified file and lock it.

Bobby edits her local copy of the file.



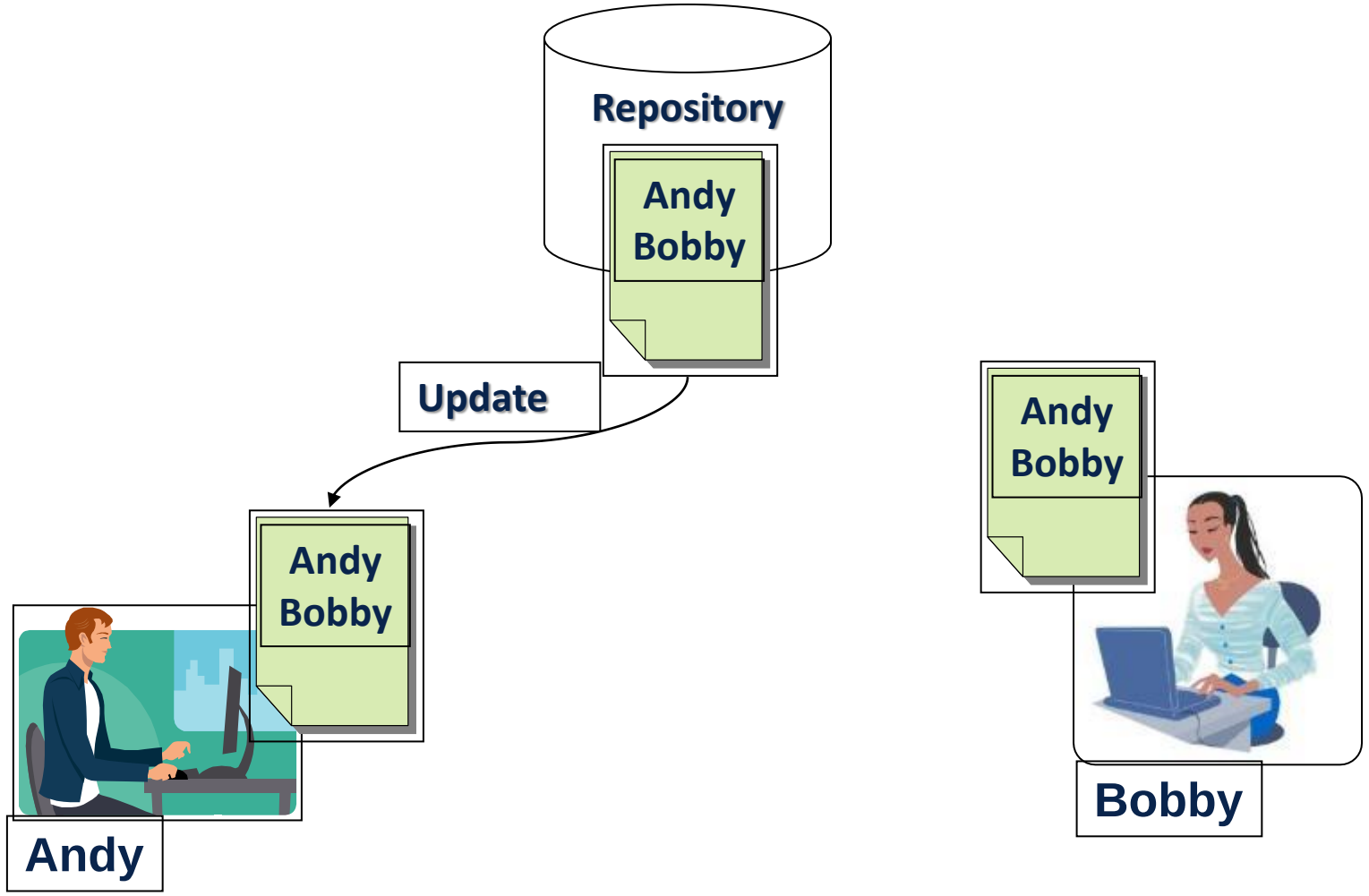
The Lock-Modify-Unlock Model (6)

Bobby finishes, commits her changes and unlocks the file.



The Lock-Modify-Unlock Model (7)

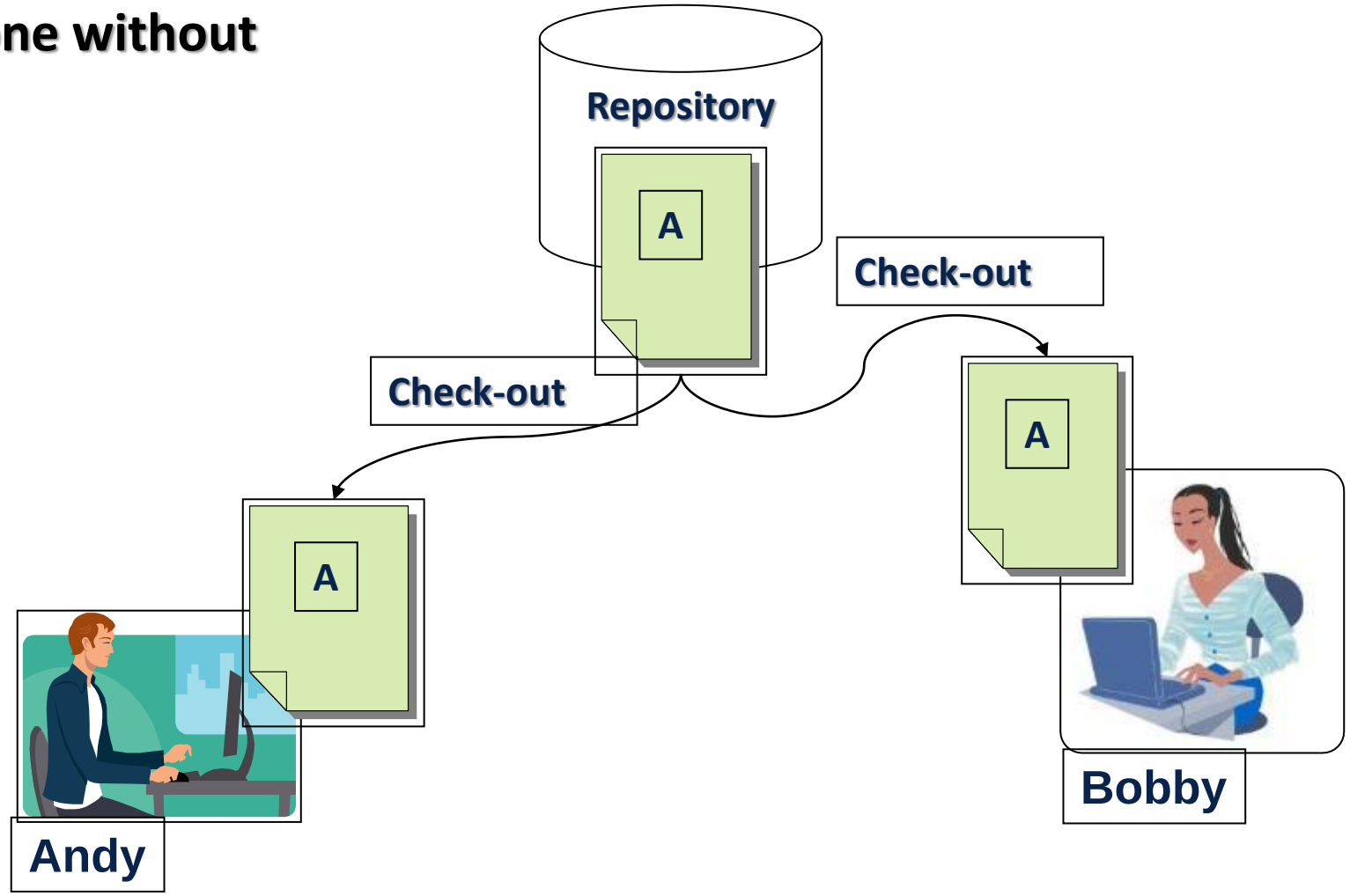
Andy updates the changes from the repository.



The Copy-Modify-Merge Model (1)

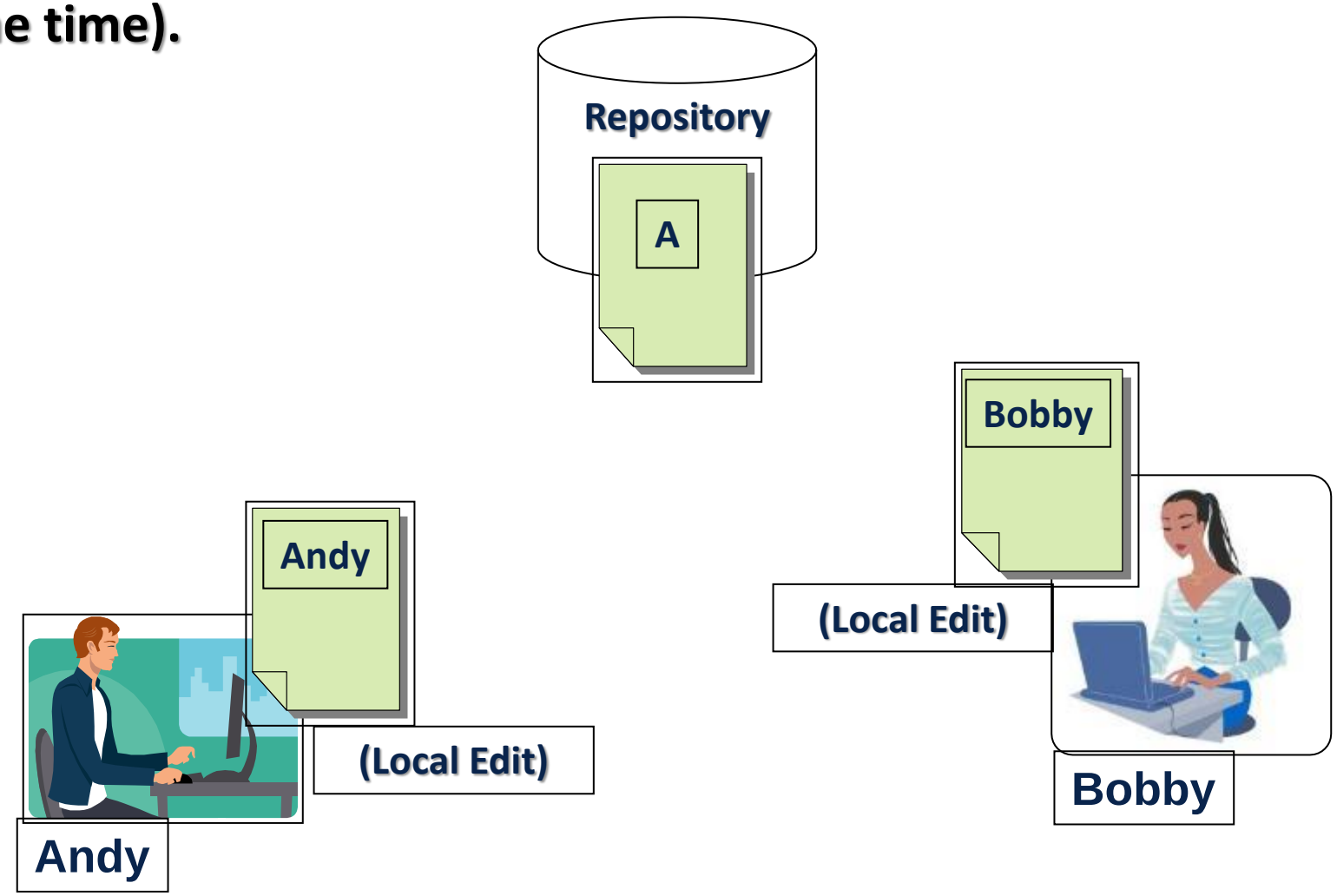
Andy and Bobby check-out a file A.

The check-out is done without locking.



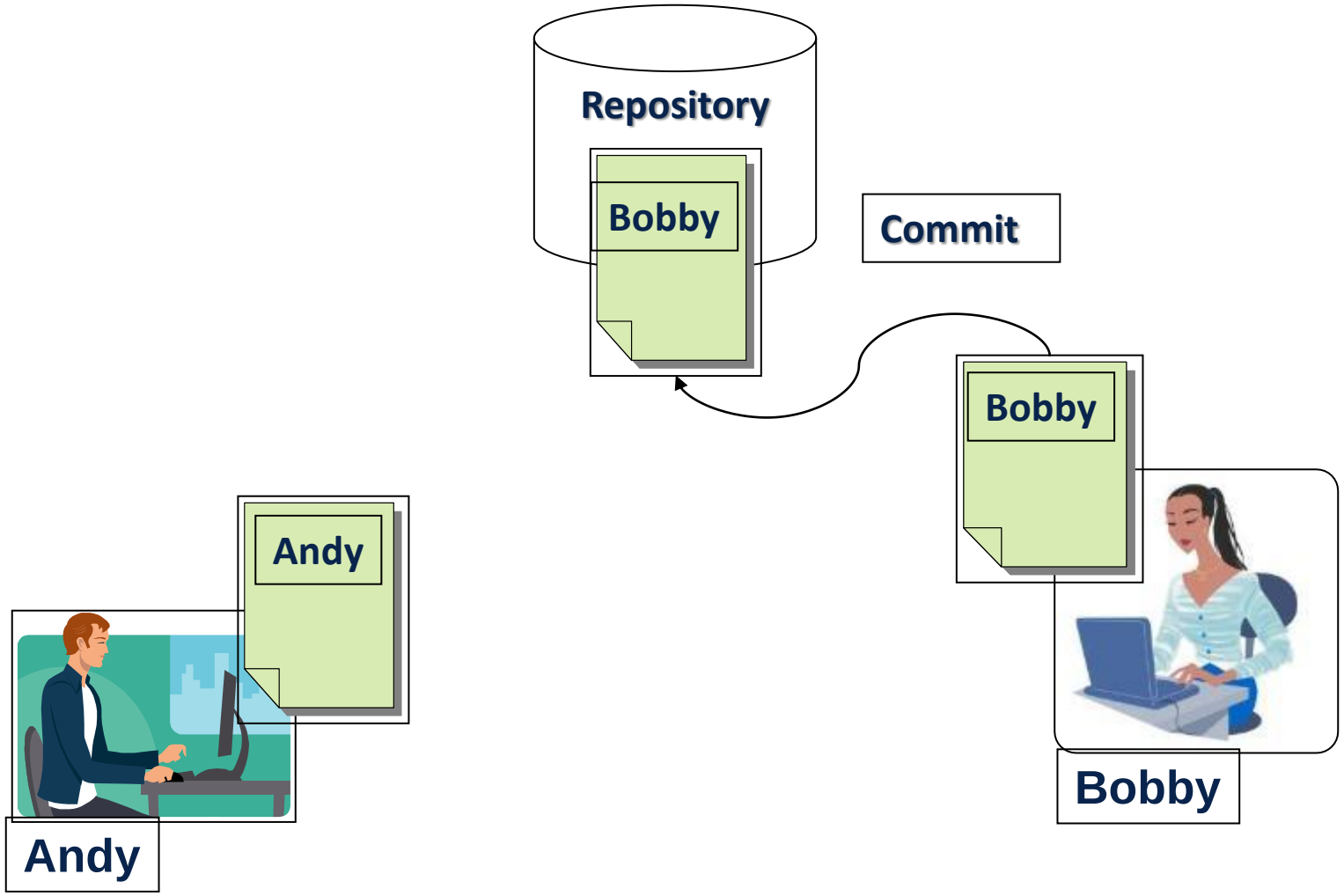
The Copy-Modify-Merge Model (2)

Both of them edit the local copies of the file (in the same time).



The Copy-Modify-Merge Model (3)

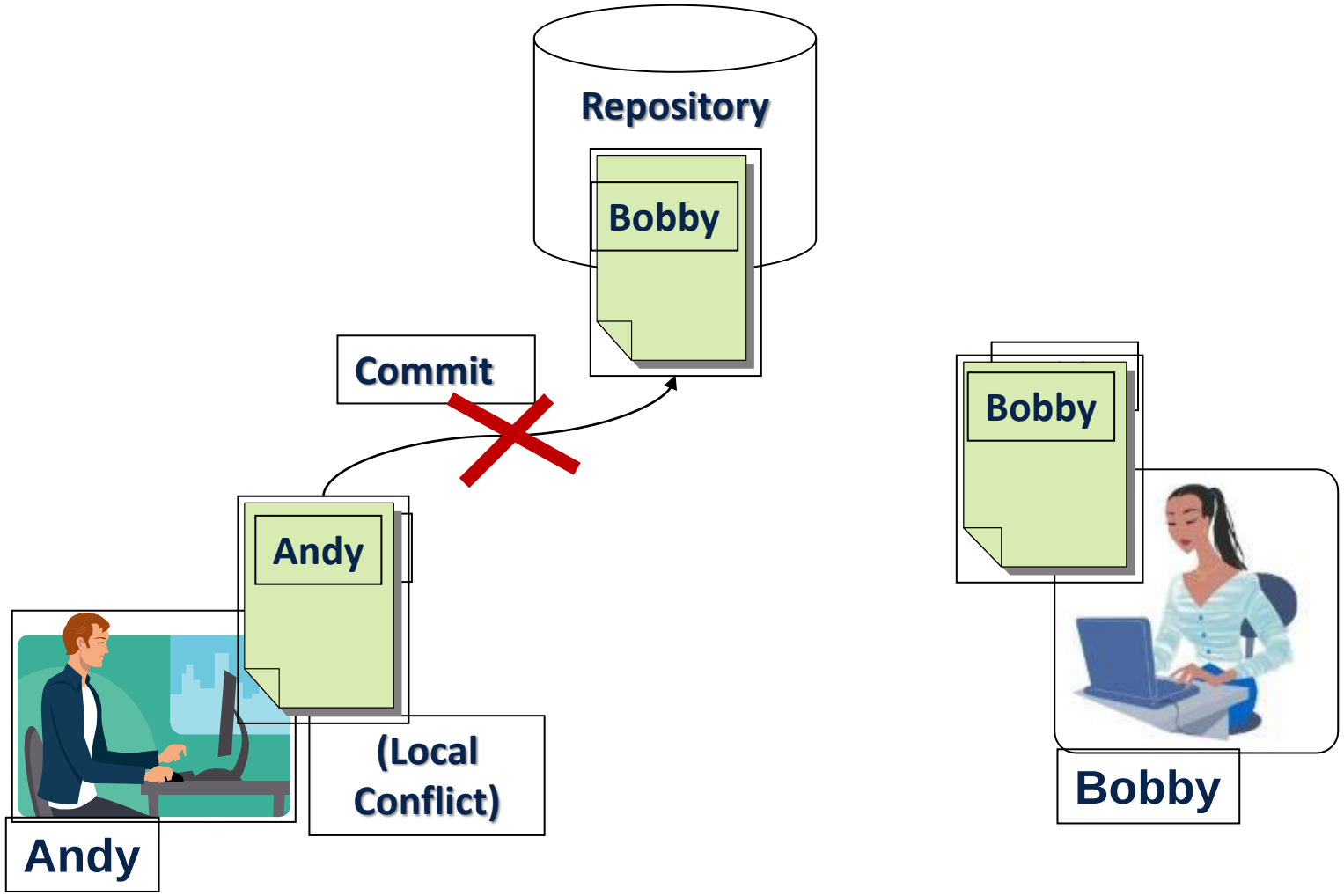
Bobby commits her changes to the repository.



The Copy-Modify-Merge Model (4)

Andy tries to commit his changes.

A conflict occurs.

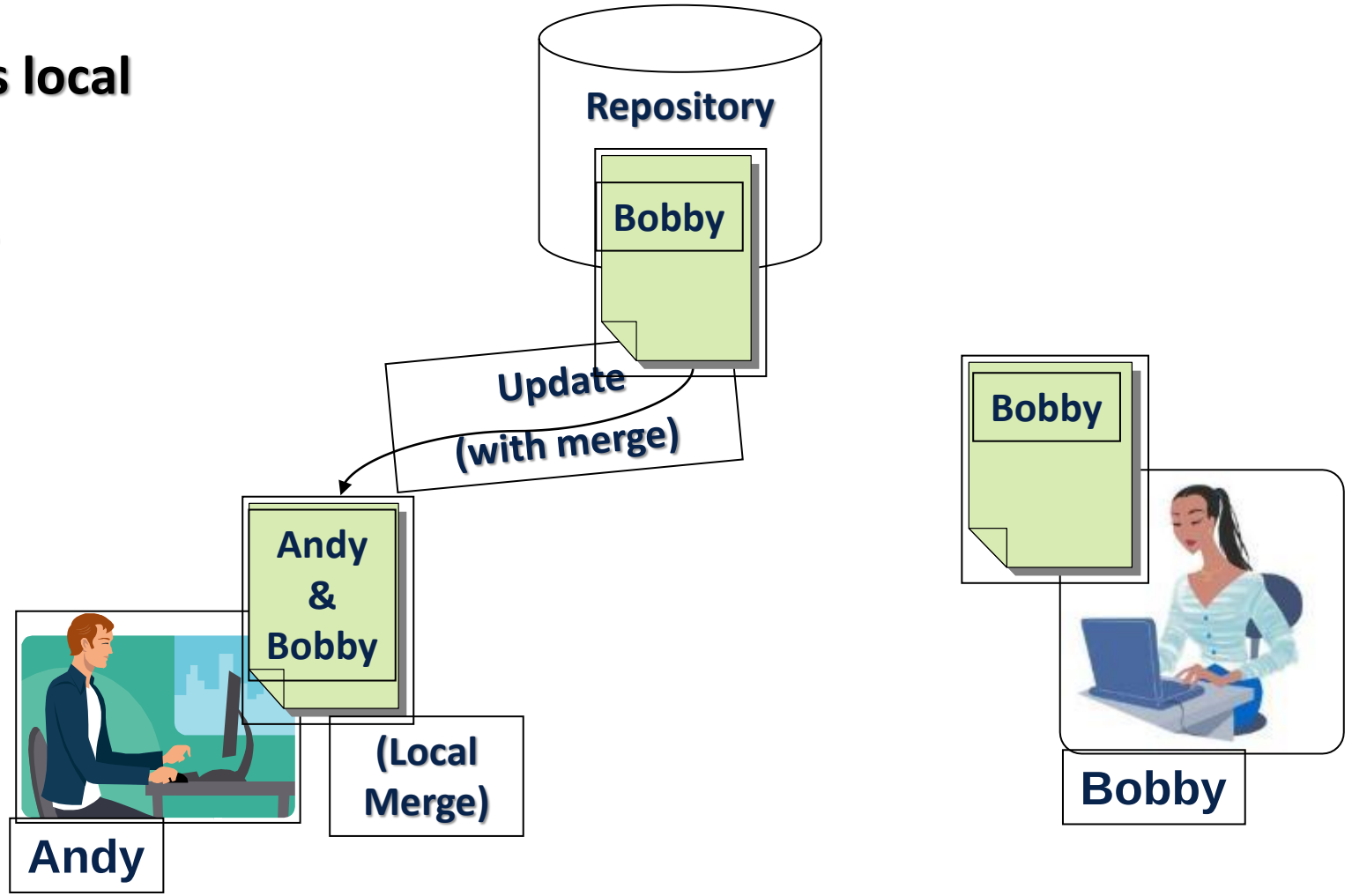


The Copy-Modify-Merge Model (5)

Andy updates his changes with the ones from the repository.

The changes merge into his local copy.

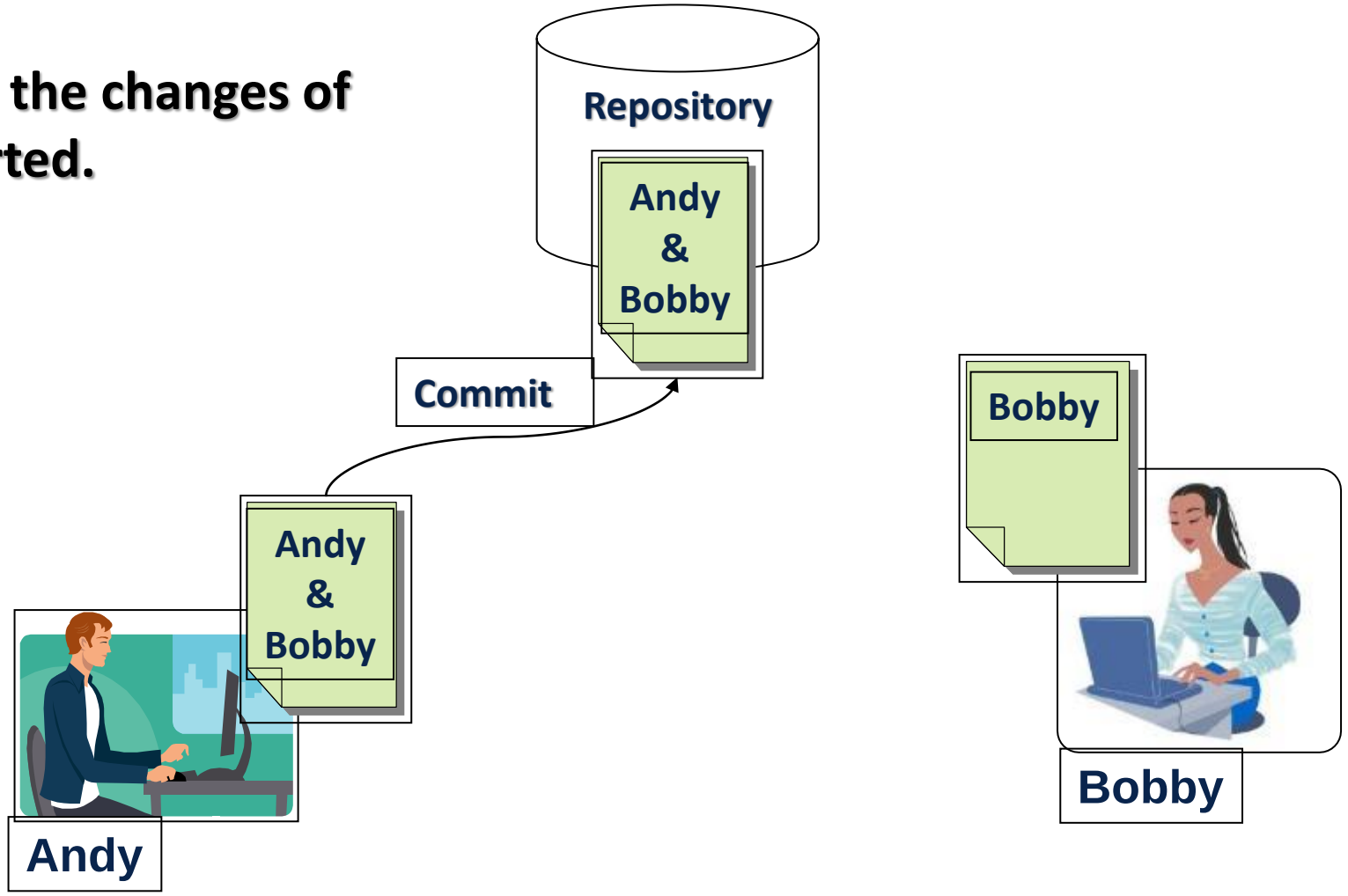
A merge conflict can occur.



The Copy-Modify-Merge Model (6)

Andy commits the merged changes to the repository.

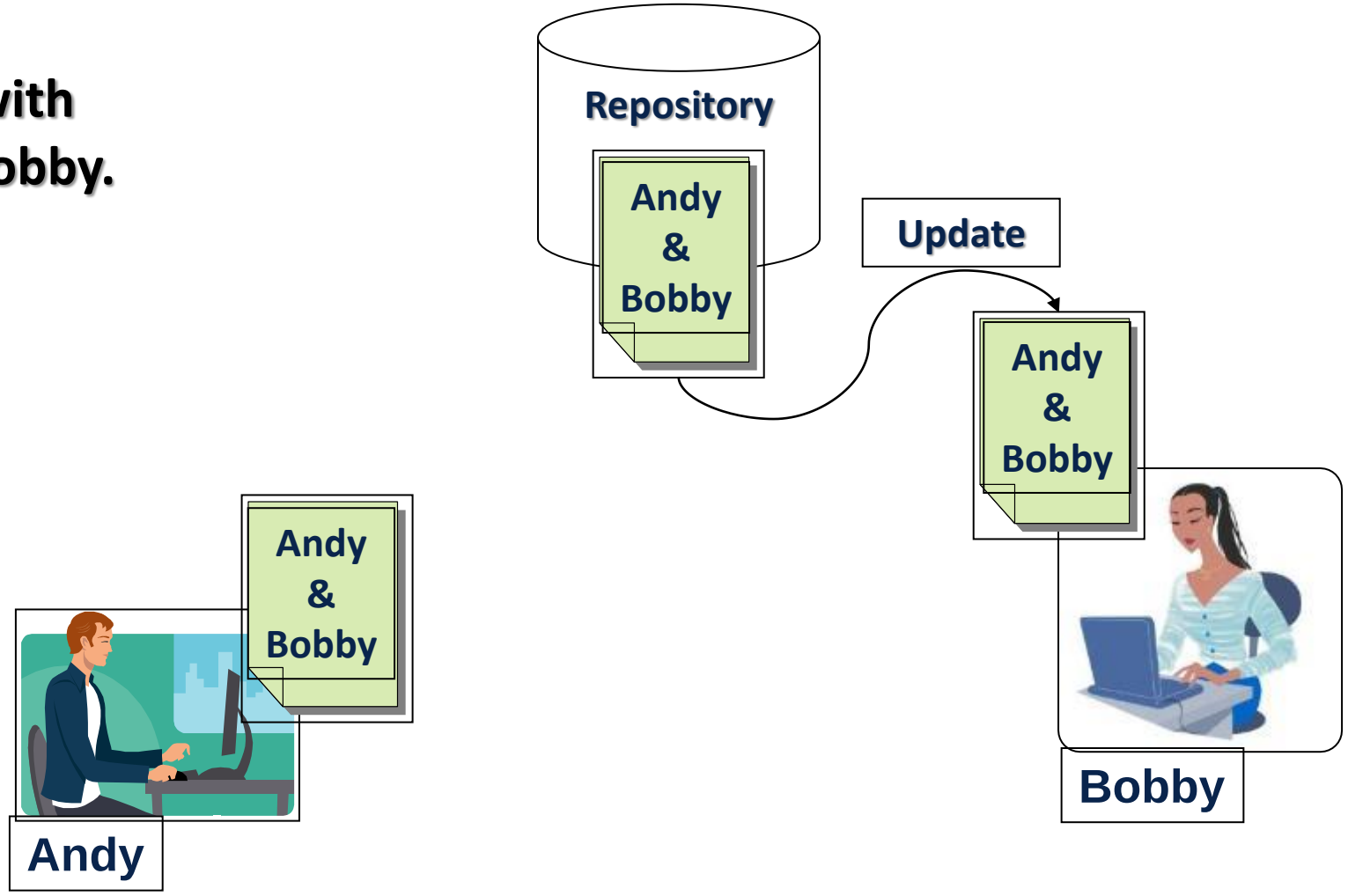
A common version with the changes of Andy and Bobby is inserted.



The Copy-Modify-Merge Model (7)

Bobby updates the changes from the repository.

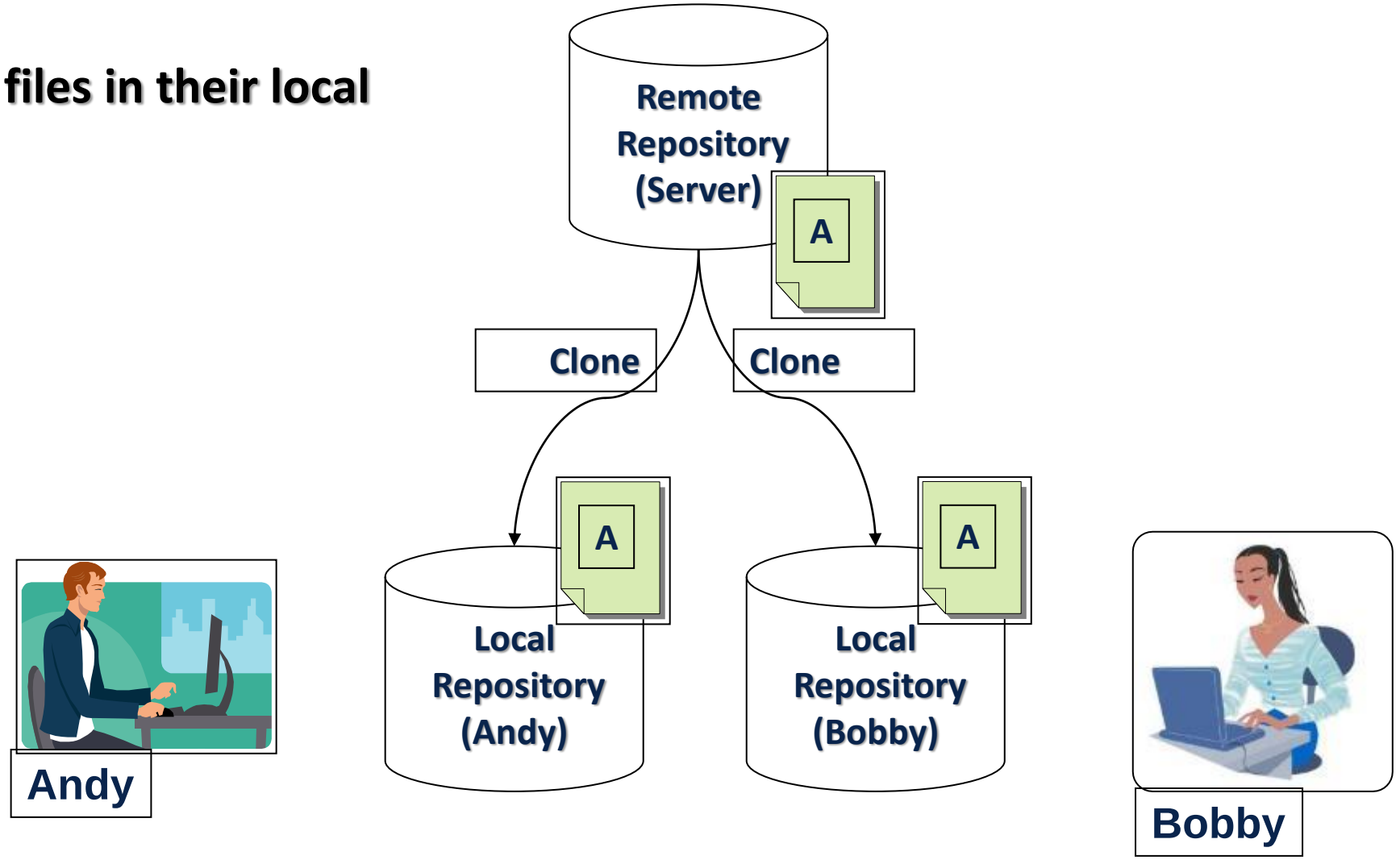
She gets the common version with both changes from Andy and Bobby.



Distributed Version Control (1)

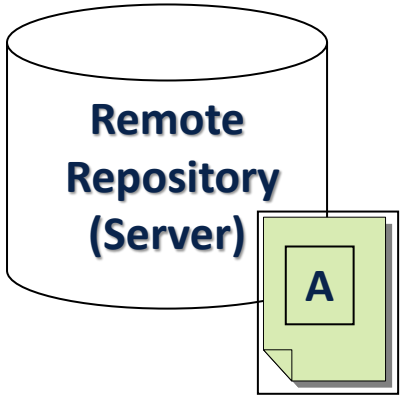
Andy and Bobby clone the remote repository locally.

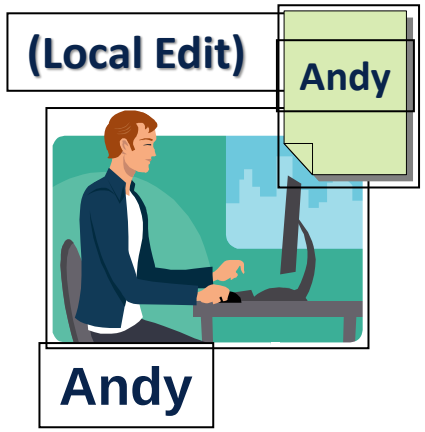
They both have the same files in their local repositories.

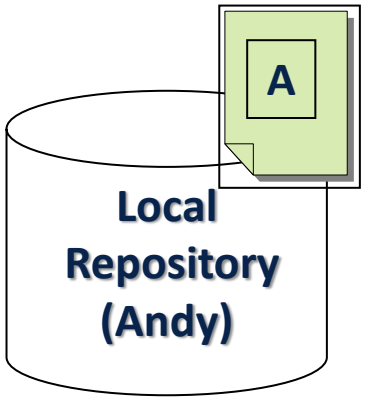


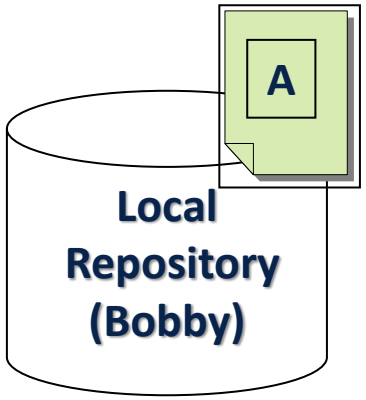
Distributed Version Control (2)

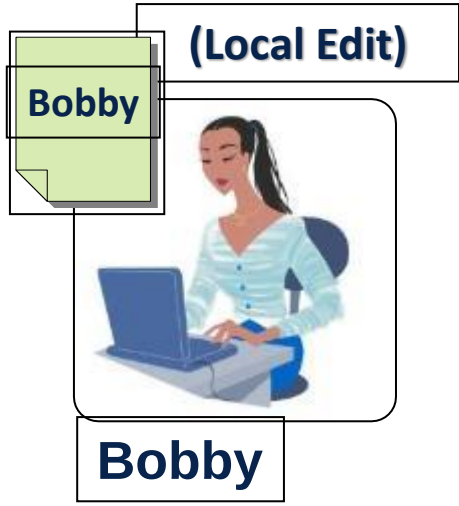
Andy and Bobby work locally on a certain file A.





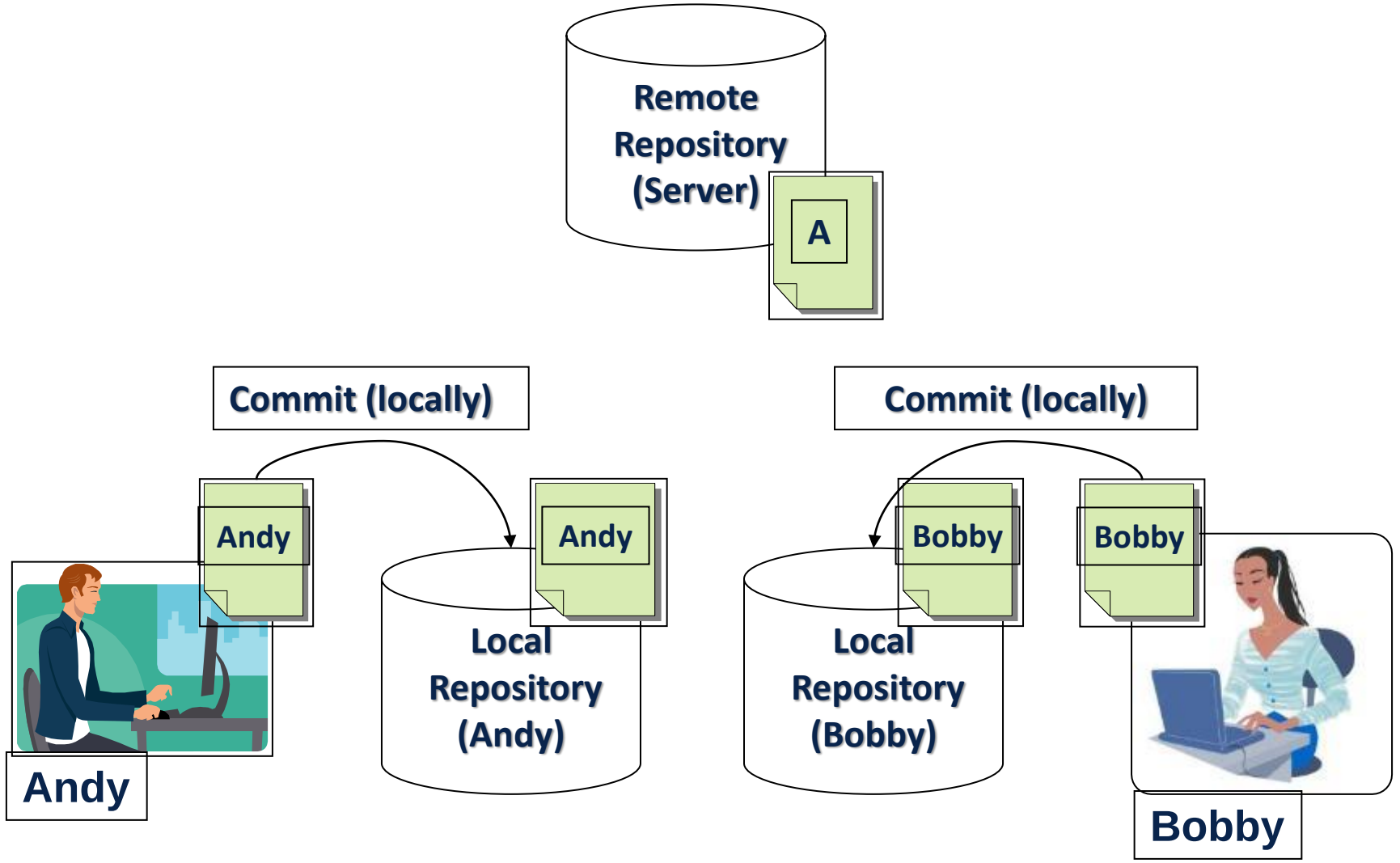






Distributed Version Control (3)

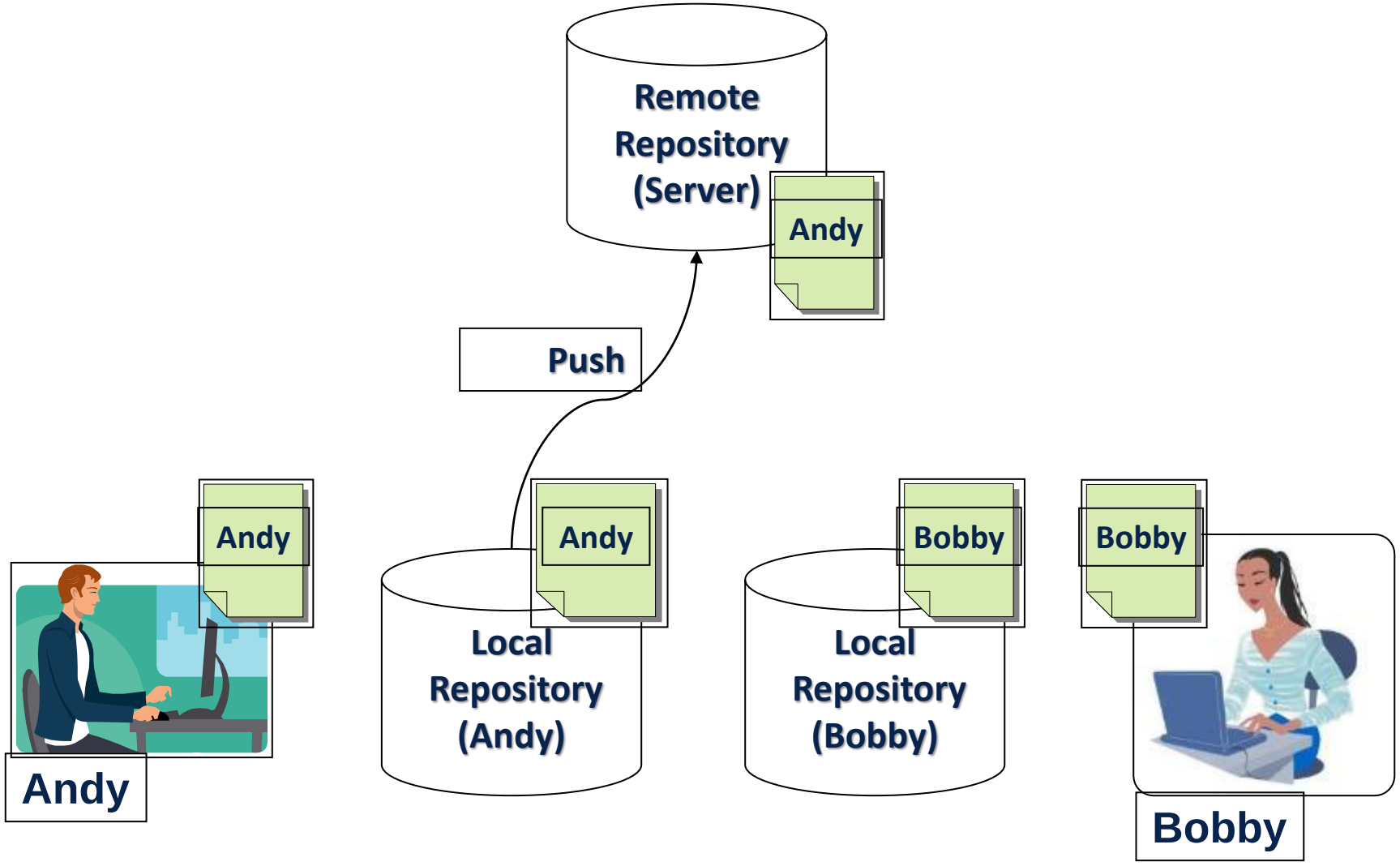
Andy and Bobby commit locally the modified file A into their local repositories.



Distributed Version Control (4)

Andy pushes the file A to the remote repository.

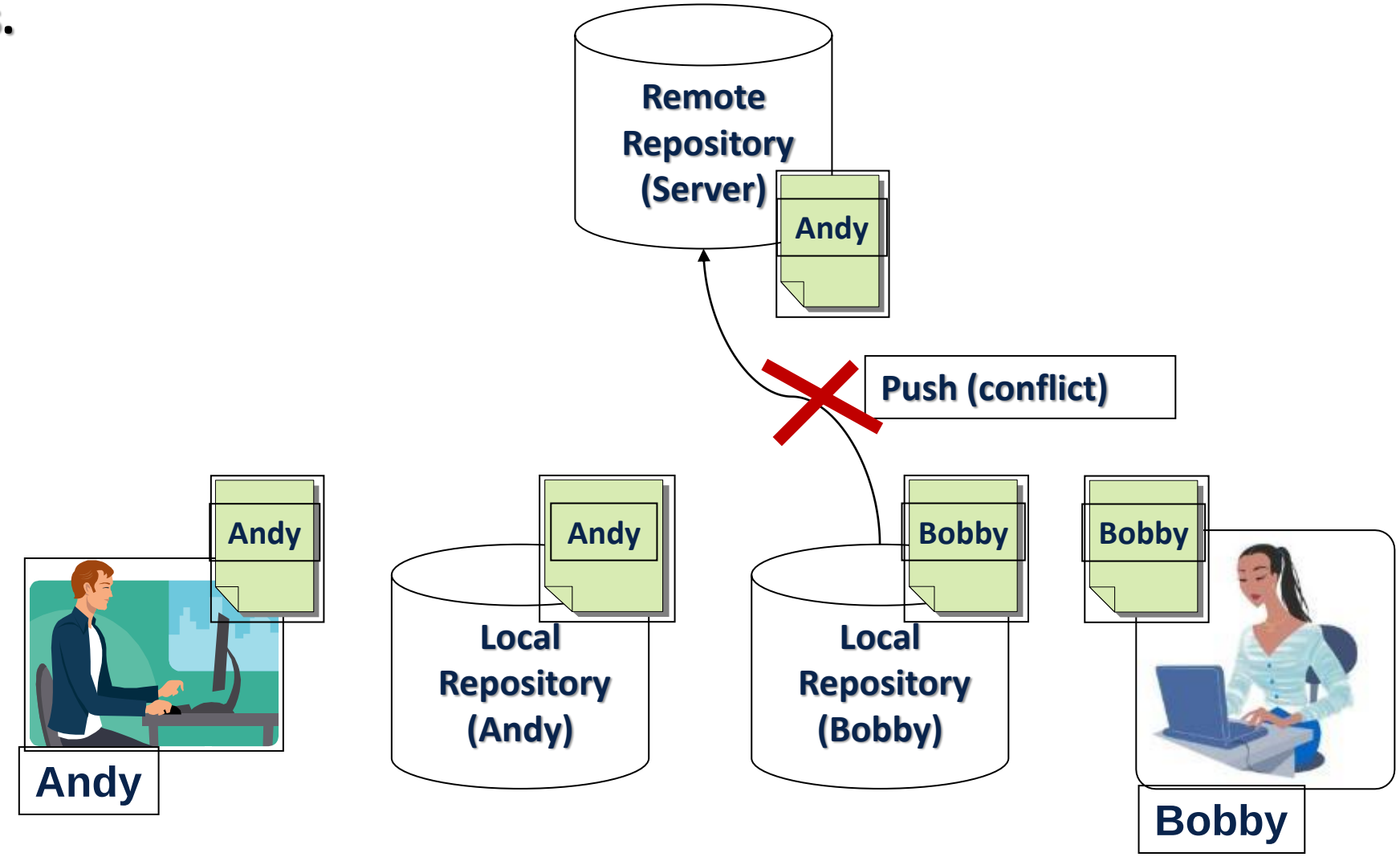
Still no conflicts occur.



Distributed Version Control (5)

Bobby tries to push her changes.

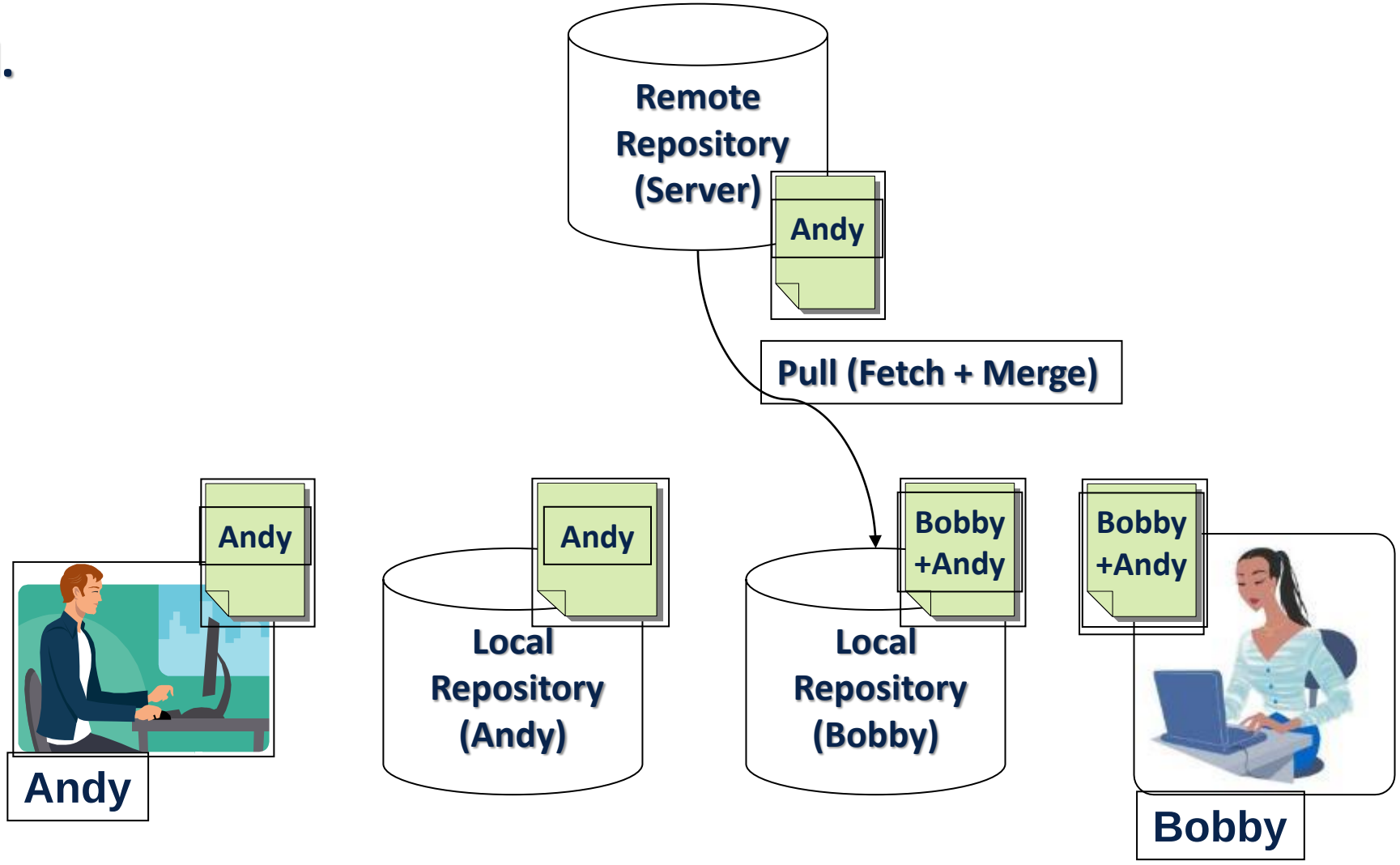
A versioning conflict occurs.



Distributed Version Control (6)

Bobby merges the her local files with the files from the remote repository.

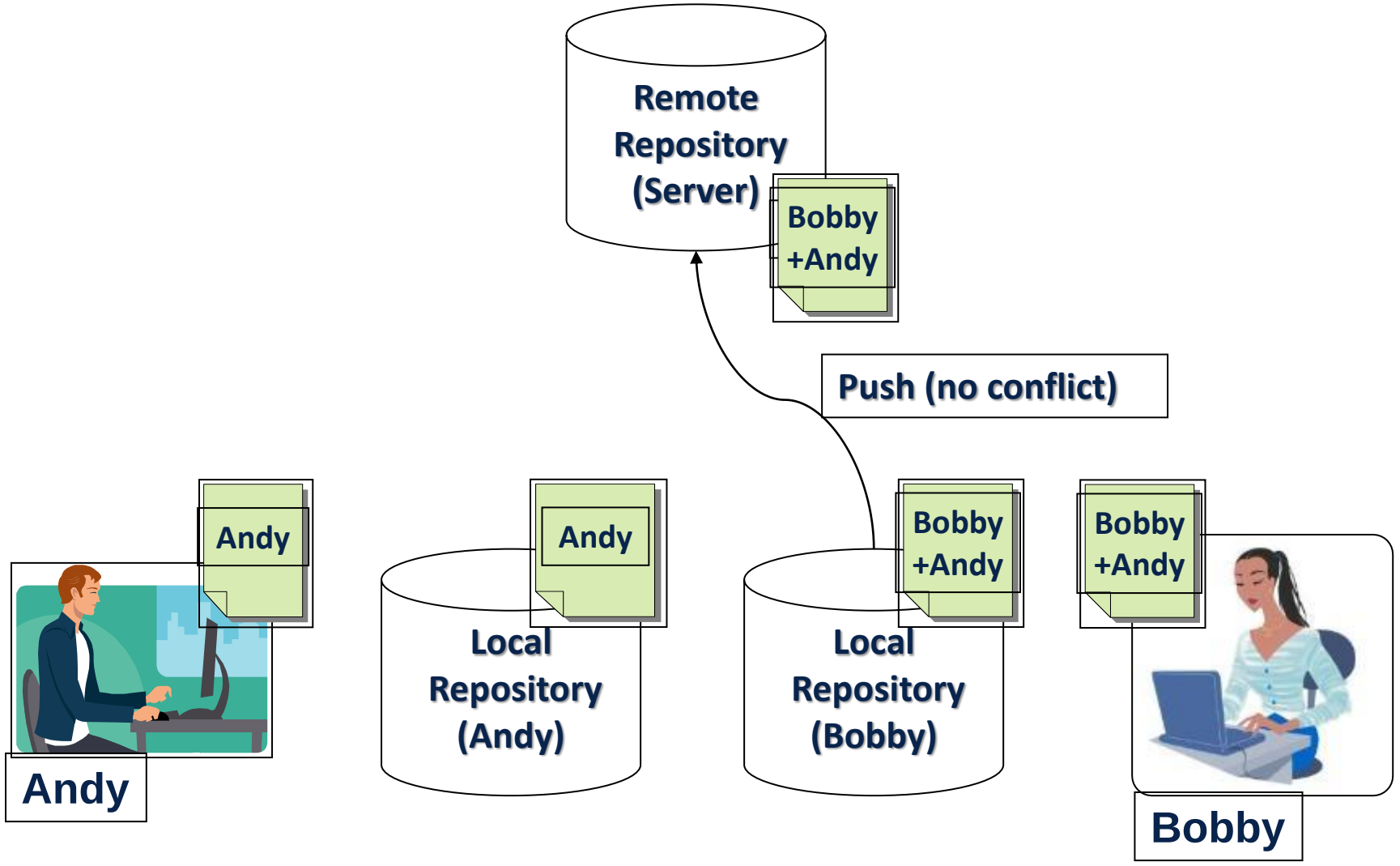
Conflicts are locally resolved.



DISTRIBUTED VERSION CONTROL (7)

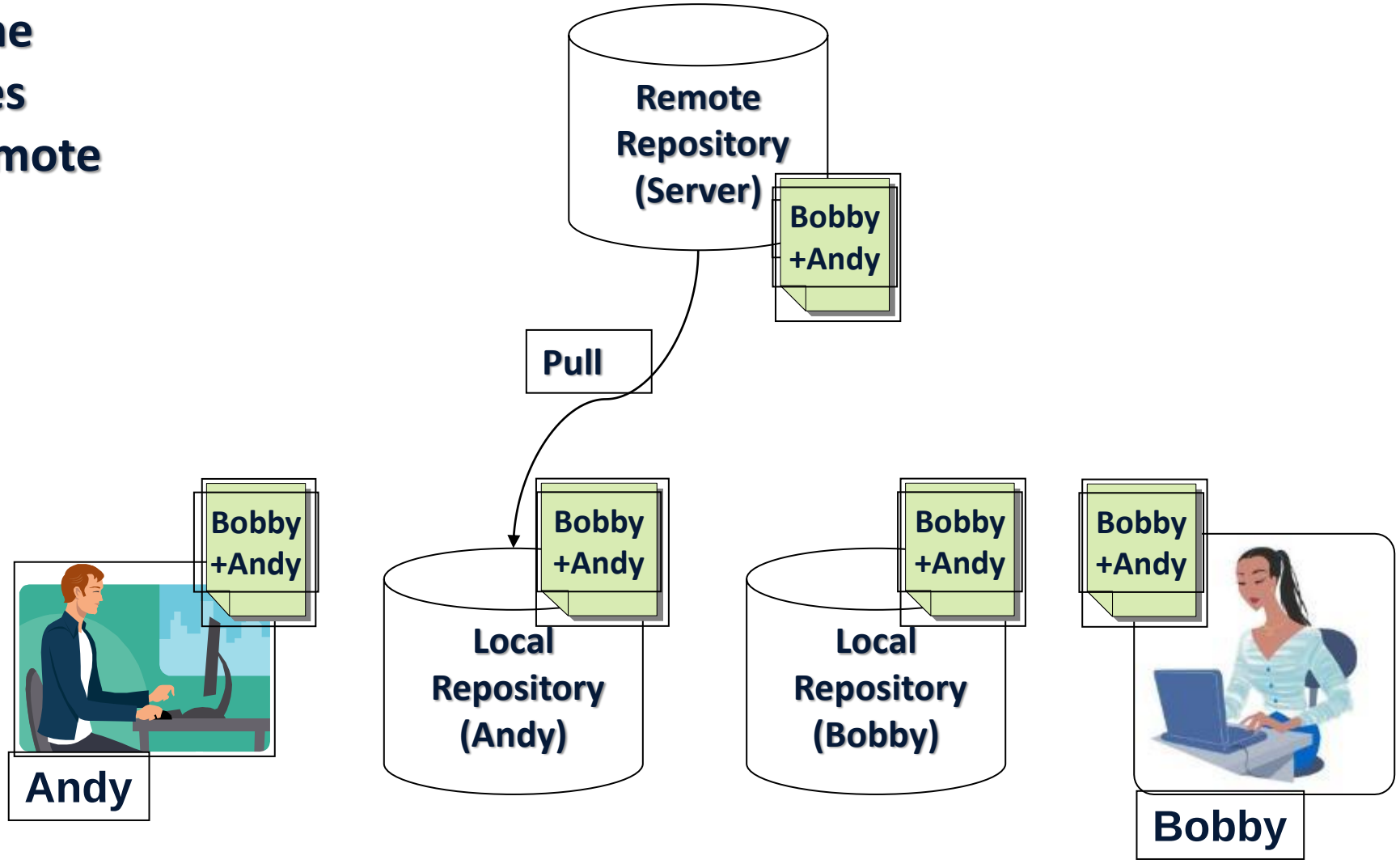
Bobby commits her merged changes.

No version conflict.



Distributed Version Control (8)

Andy pulls (updates) the changed files from the remote repository.



BENEFITS OF VCS

- The system manages the directories, files, and individual changes made over time
- allows users to find root causes for mistakes or bugs, or revert to an earlier version
- Improve team productivity and enable collaboration.
- Enhance team communication with a reliable solution.
- Reduce development errors and conflicts.
- Improve customer satisfaction with reliable software versions.

CHOOSING THE RIGHT VCS

What matters most will depend on your team's needs. But five of the top things that today's development teams are looking for:

- ✓ **Concurrent Development**
- ✓ **Automation**
- ✓ **Team Collaboration**
- ✓ **Tracked Changes — Who, What, When, Why**
- ✓ **High Availability/Disaster Recovery**

VCS Best Practices

- Save changes and back-up code consistently and frequently.
- Use a comprehensible naming convention.
- Manage appropriate controls to allow proper, productive access.
- Ensure developers can easily pull up necessary documents.
- Define team, individual, and read only access to streamline document availability and allow users to perform their functions in a timely manner.
- Use a descriptive commit message.
- Incorporate others' changes frequently
- Share your changes frequently
- Understand your merge tool