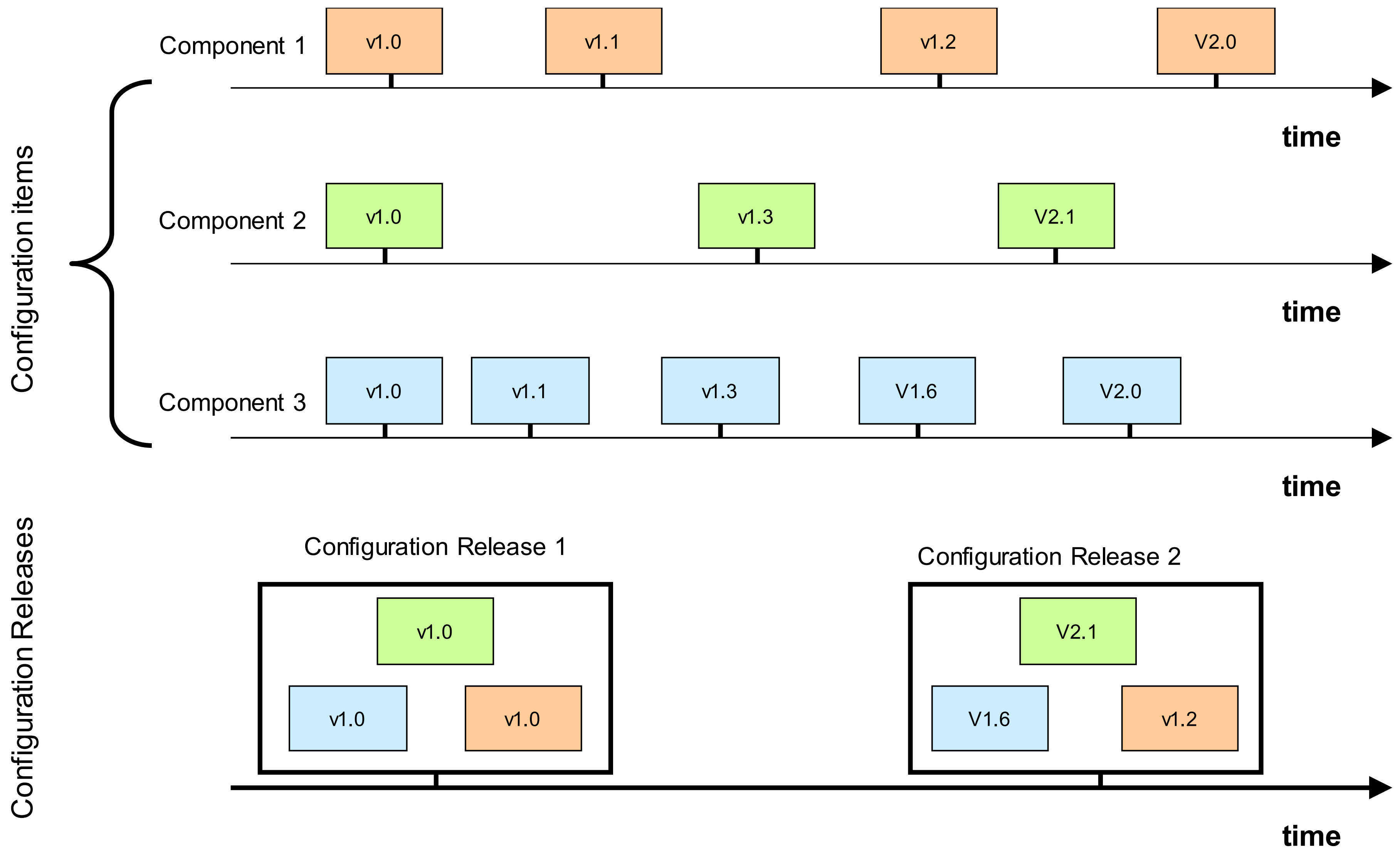
Configuration Management

VIMIMA11 Design and integration of embedded systems

Méréstechnika és Információs Rendszerek Tanszék

Configuration Items and Releases



Configuration Management CMMI Process Area

- **SG 1: Establish Baselines**
  - **SP 1.1:** Identify Configuration Items
  - **SP 1.2:** Establish a Configuration Management System
  - **SP 1.3** Create or Release Baselines
- **SG 2: Track and Control Changes**
  - **SP 2.1:** Track Change Requests
  - **SP 2.2:** Control Configuration Items
- **SG 3: Establish Integrity**
  - **SP 3.1:** Establish Configuration Management Records
  - **SP 3.2:** Perform Configuration Audits

Identifying Configuration Items

- Requirements
- Product specifications
- Architecture documentation and design data
- Plans
- Hardware and equipment
- Code and libraries
- Test results

Identifying Configuration Items

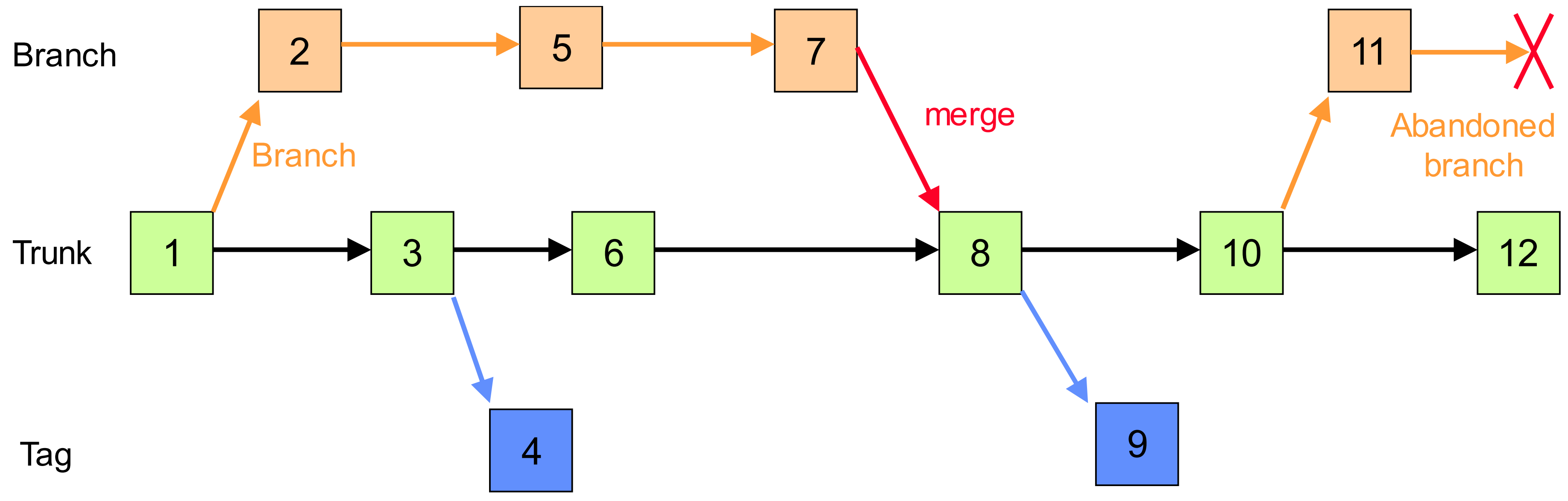
- Requirements
- Product specifications
- Architecture documentation and design data
- Plans
- Hardware and equipment
- Code and libraries
- Test results

- **Development tools**
- **Test tools**
- **Compilers, even operating systems**

Establish a Configuration Management System

- Typical storing points in a Configuration management system
  - Dynamic: Locally at the developer
  - Controlled, centralized: A central server for configuration items
  - Statically archived: Archives for the releases
- Determination of the configuration management lifecycle
- Setting user privileges and rights
  - Read, Write and Create access rights
  - User account and User groups management

Example for a typical Configuration Lifecycle



Tools of Configuration management

- Mostly some kind of version control system is used

Version Control Systems

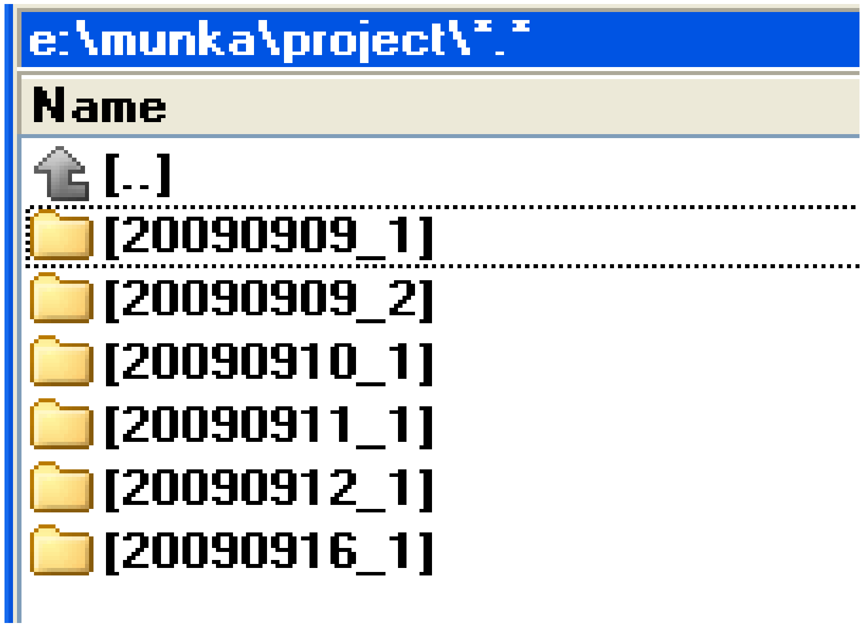
The need for such systems

- One typical day of a developer:
  - At the start of the day we have a running software
  - We add some lines to the software
  - The software freezes
  - We remove or uncomment the lines added
  - The software still freezes

- The situation is even complicated if we work in a team:
  - We add some lines to a working software
  - Someone also add few lines to the same part of the software
  - The software freezes

Trivial Version Control

- We create a new folder for every changes with the date of the changes
- Every such folder should have a *changelog* file to describe the changes



```
e:\munka\project\*.*
Name
[..]
[20090909_1]
[20090909_2]
[20090910_1]
[20090911_1]
[20090912_1]
[20090916_1]
```

Triviális Version Control

- We create a new folder for every changes with the date of the changes
- Every such folder should have a *changelog* file to describe the changes

Problems

- Requires much disk space
- How often should we create a new version?
- Should we create copy only from a working version or from an intermediate one too?
- The *changelog* file should be used very consistently, or it cause more trouble then help.

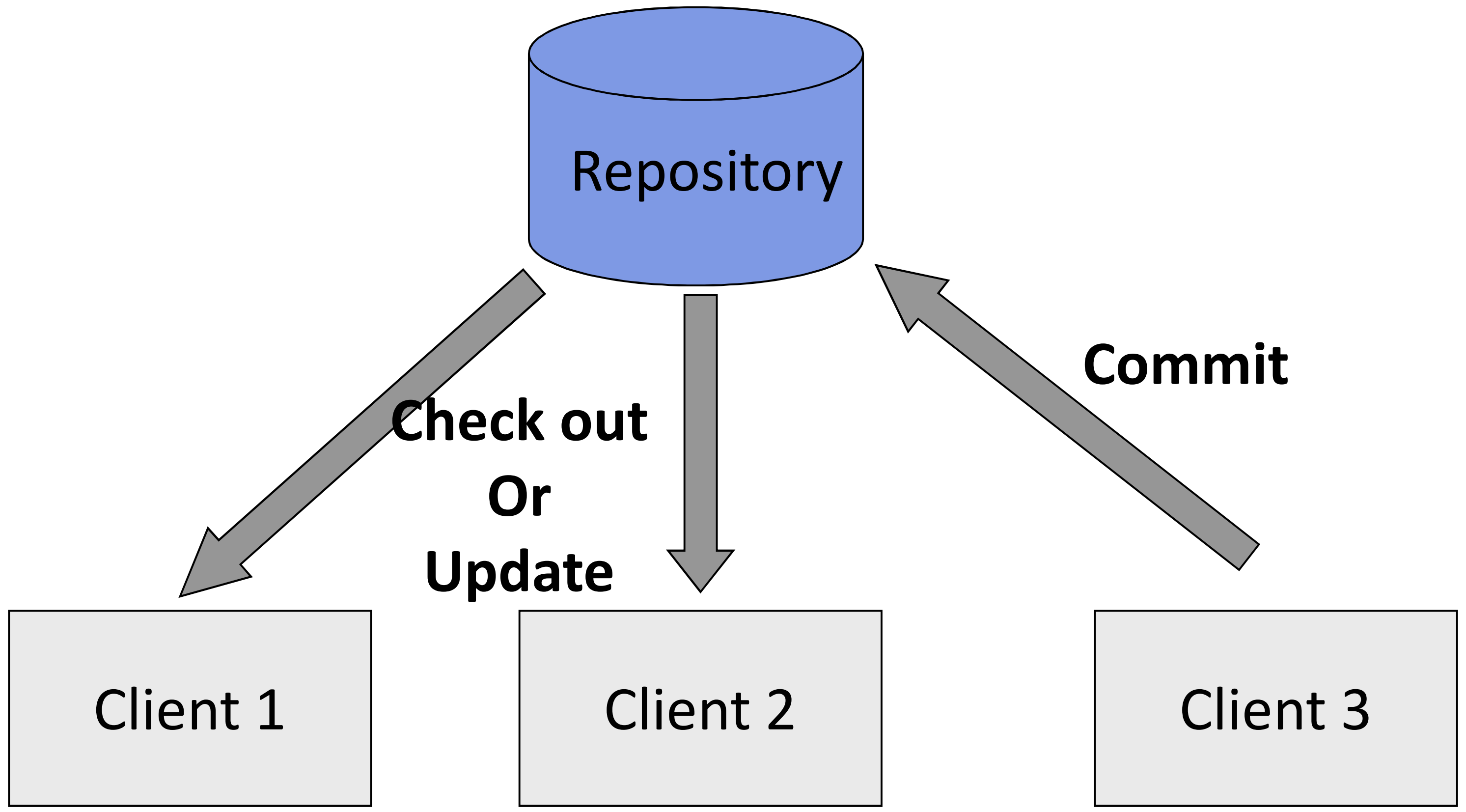
Version Control Systems
Basic Terms

- The version managements system are used to follow every changes made on a project
- The version control system logs
  - Every changes to every file assigned to version control
  - Every changes to the folder structure
- The user can
  - Check any version of a file during its version control life cycle
  - Check the reason and the committer of every changes
  - Making comments to its own changes

Centralised Version Control Systems
Basic Terms

- **Repository**: Central Storage of the current and previous versions of the project *(master copy)*.
- **Client**: user who want to work on the project
- **Working copy**: A local version of the project downloaded from the **Repository** by the **Client**

Centralised Version Control Systems Basic behavior



Version Control Strategies: *The main questions?*

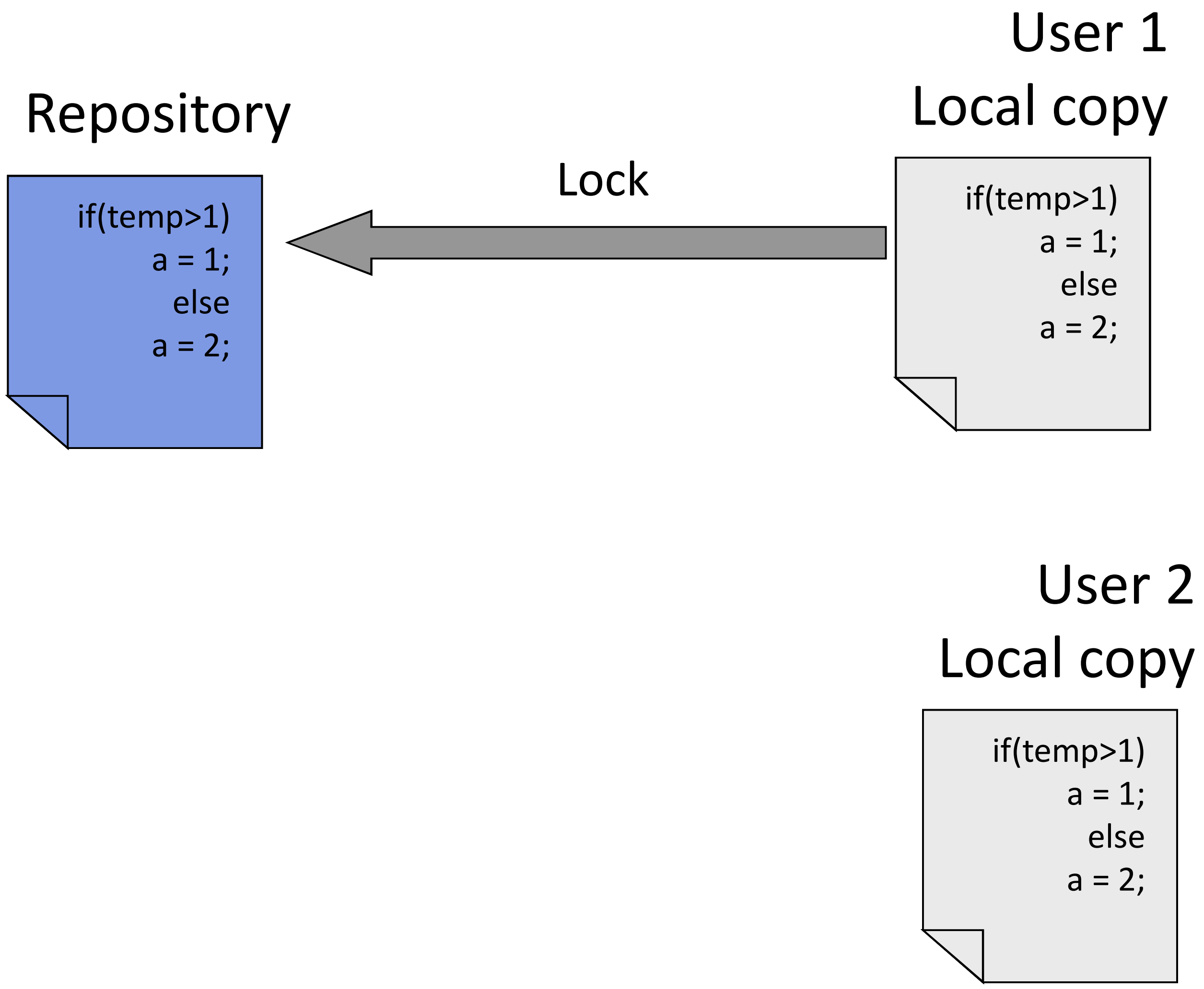
- How the version control systems support the parallel work of multiple developers?
- What is the strategy or method to avoid the inconsistency caused by the parallel work on the same file?

The Lock–Modify–Unlock approach

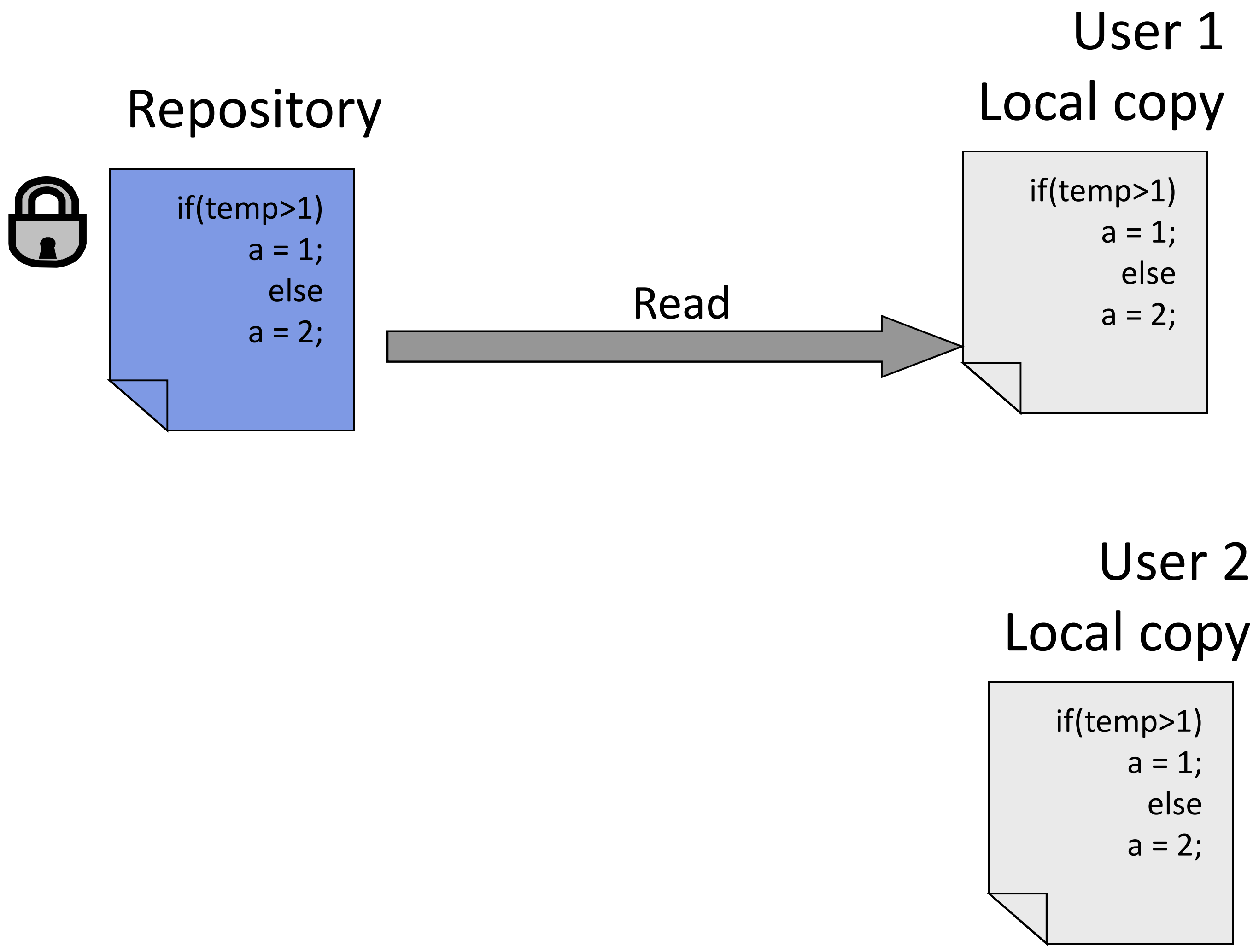
- **Before modifying** a file it have to be **locked**
- **After modification** it should be **unlocked**

- There is no parallel modification of the file: only one developer can modify the file by locking it
- Locked files can be read by other developers

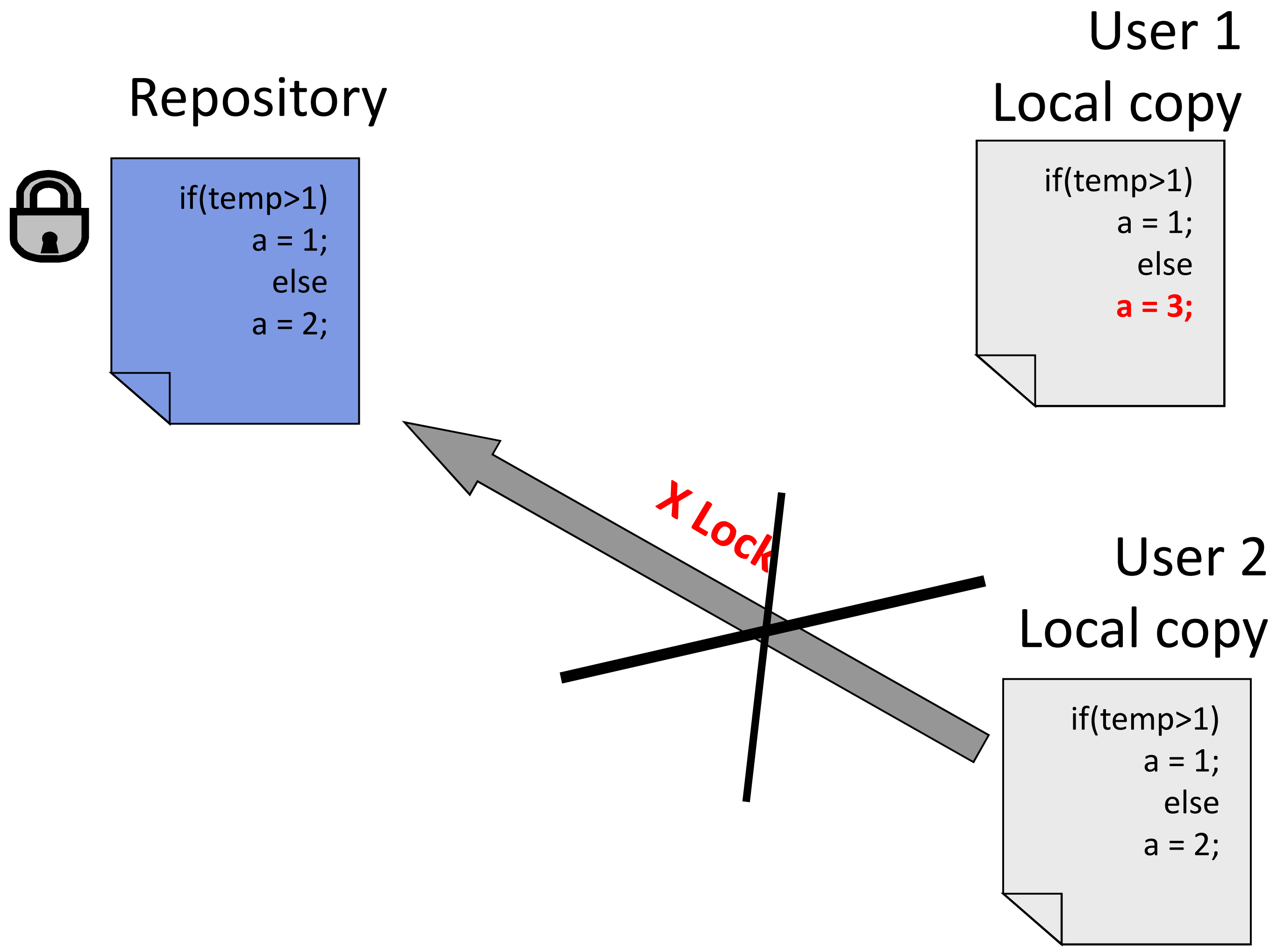
Lock–Modify–Unlock approach



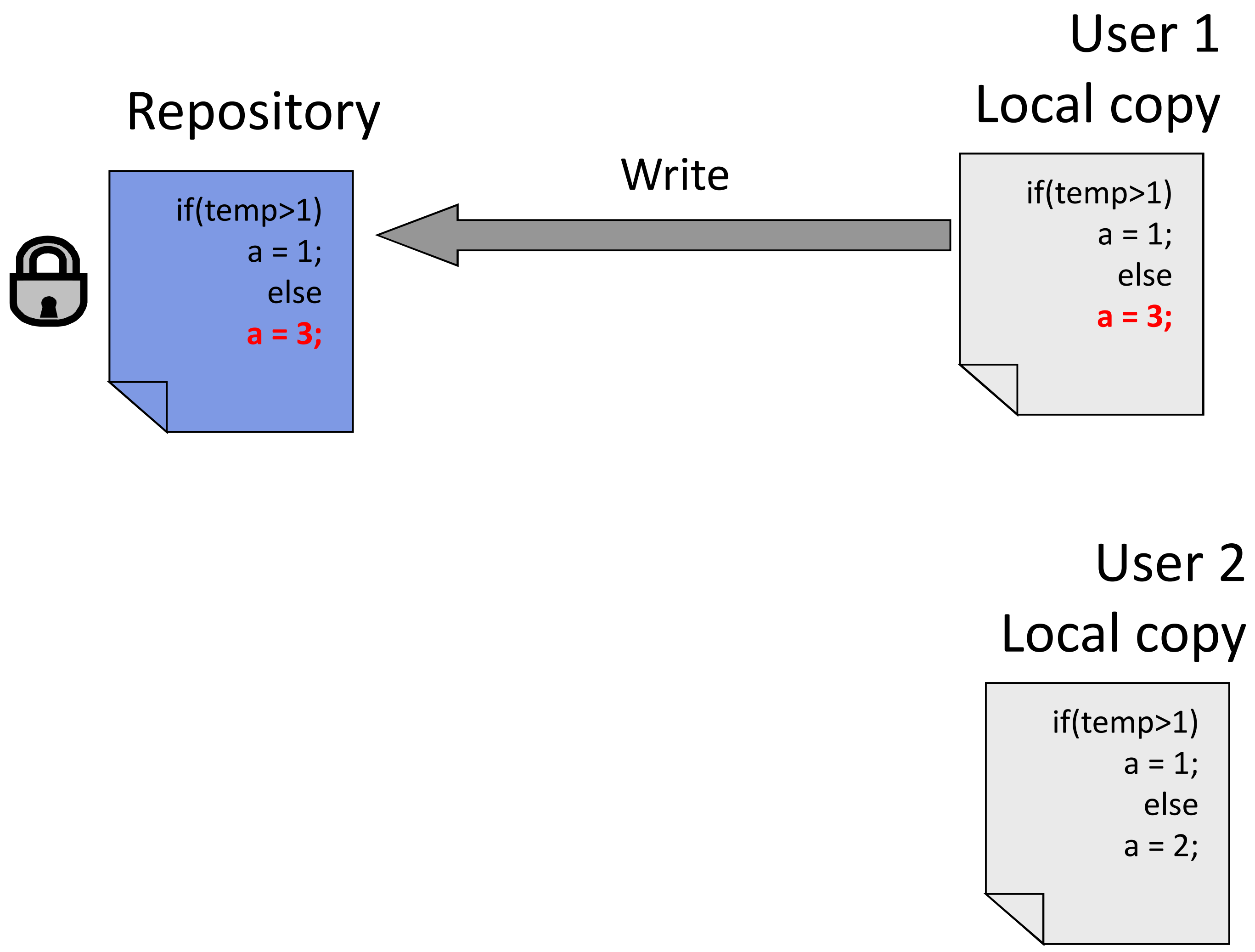
Lock–Modify–Unlock approach



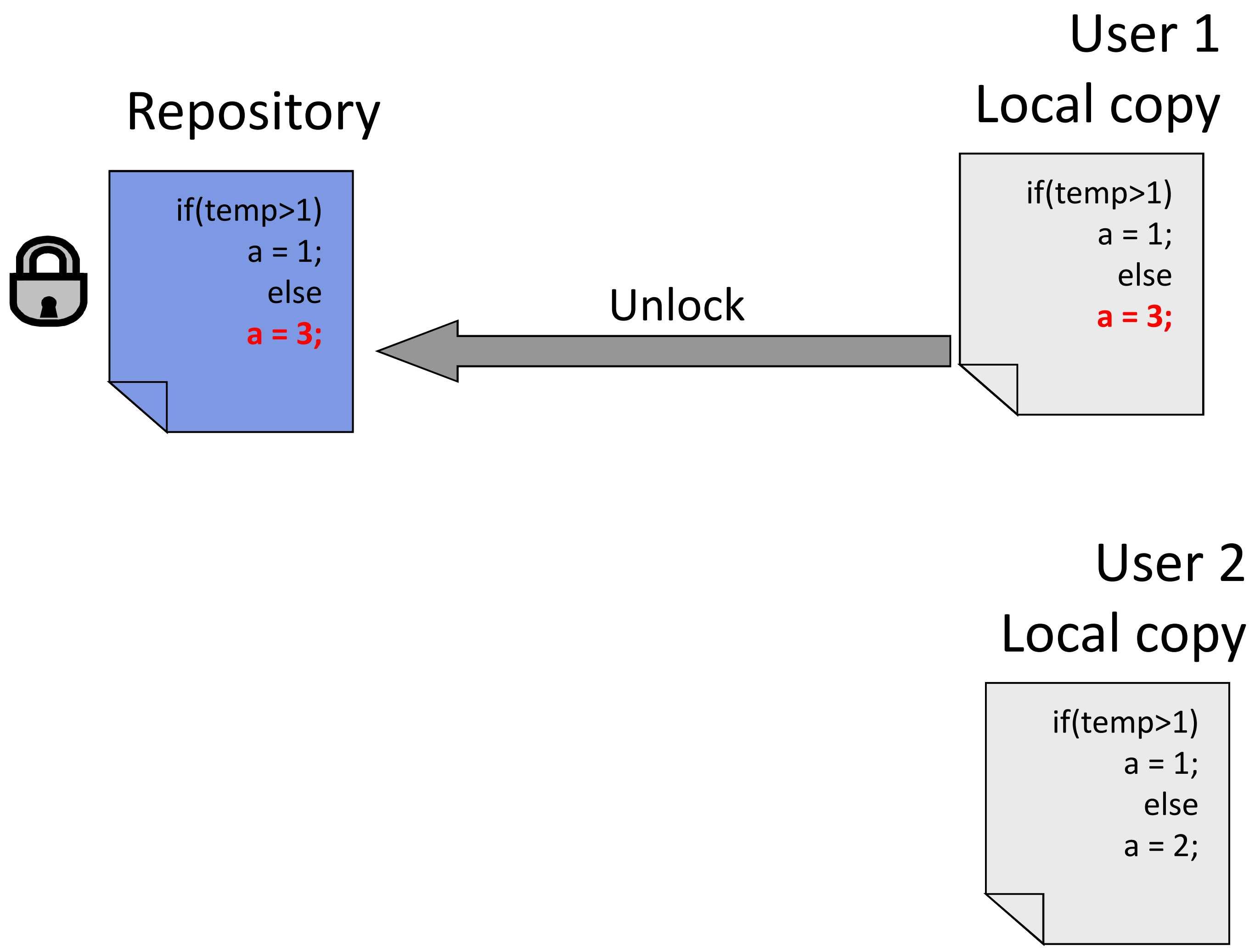
Lock–Modify–Unlock approach



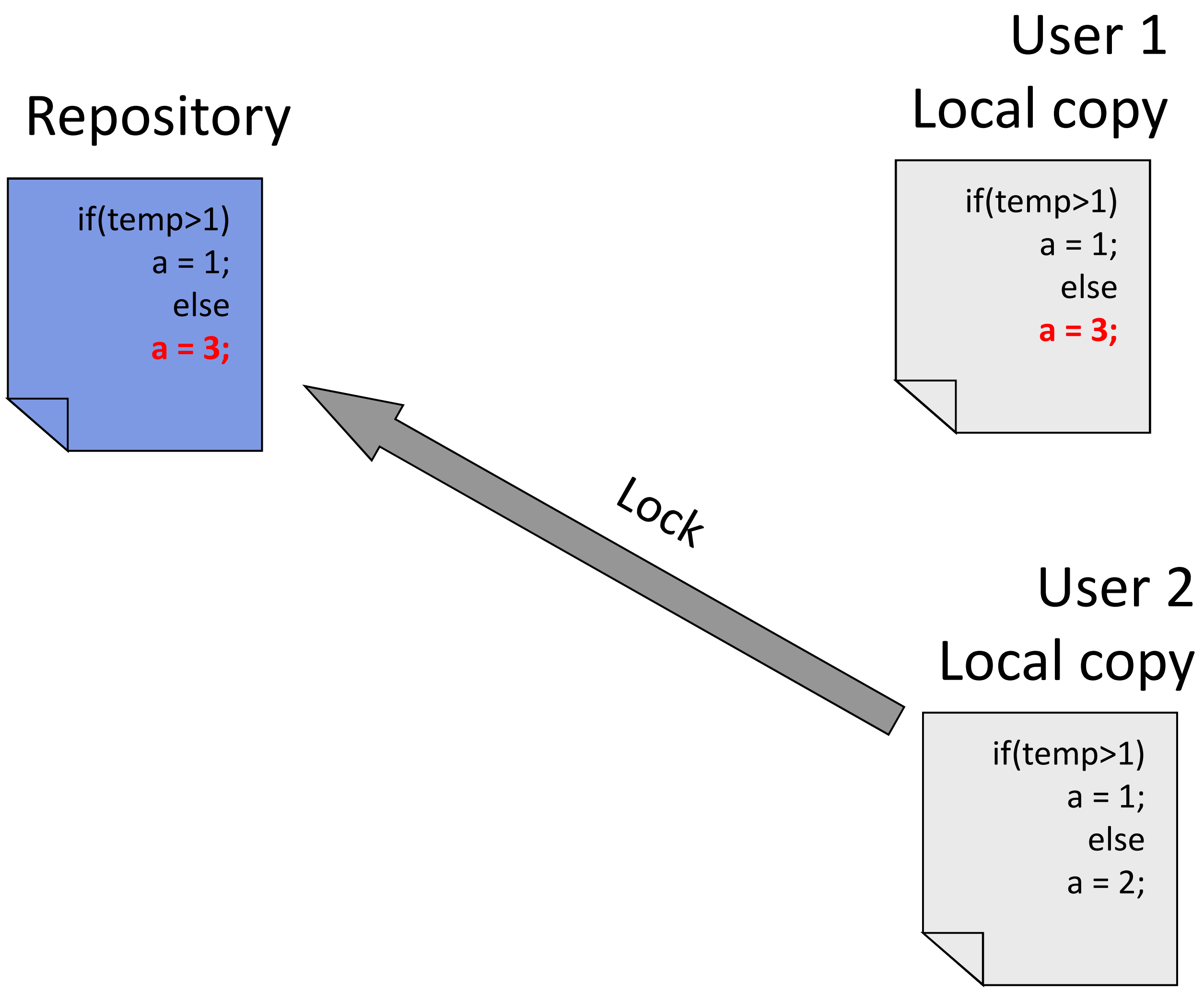
Lock–Modify–Unlock approach



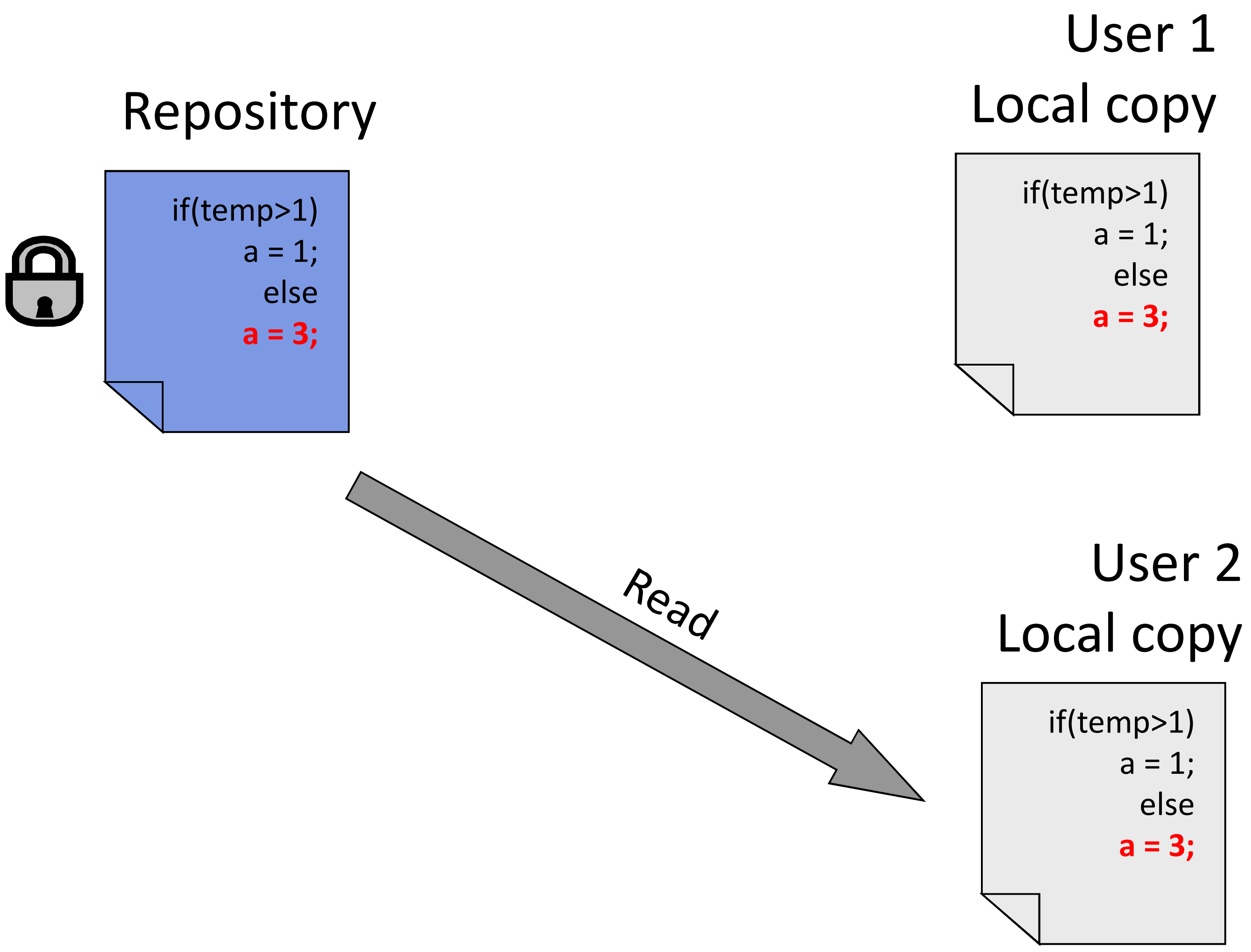
Lock–Modify–Unlock approach



Lock–Modify–Unlock approach



Lock–Modify–Unlock approach



Problems of Lock–Modify–Unlock approach

- Can lead to administrative problems:
  - One of the developers forget to unlock a file and goes to holyday …
  - System administrator is needed to unlock those files

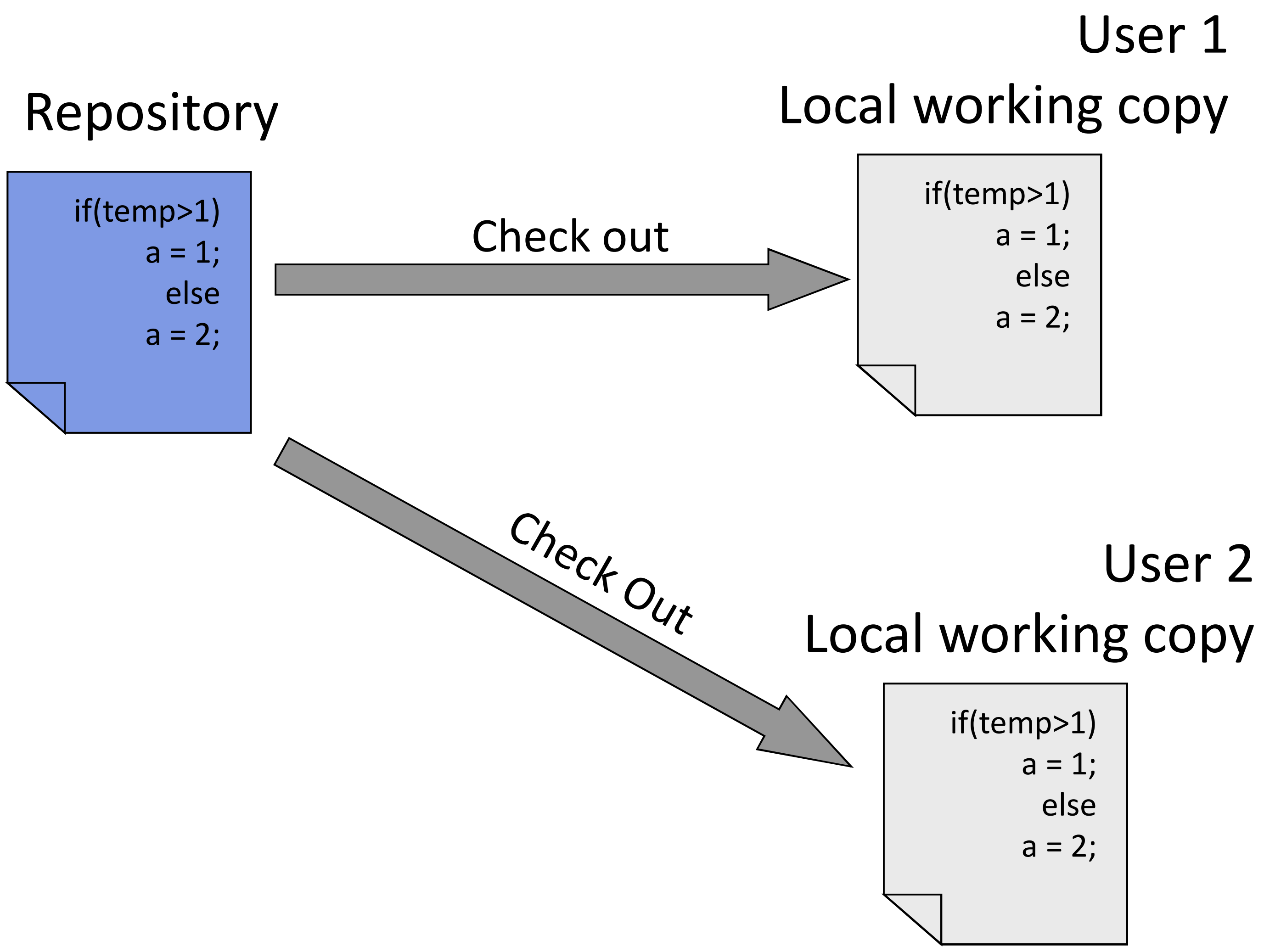
- Cause unnecessary waiting:
  - If more than one developers want to modify the same C file, but different parts of it, then there is no reason to exclude the others.

- It can lead to the false illusion of safety:
  - Developers with the lock and modify approach tends to forget the dependency of different software parts.

Copy–Modify–Merge approach

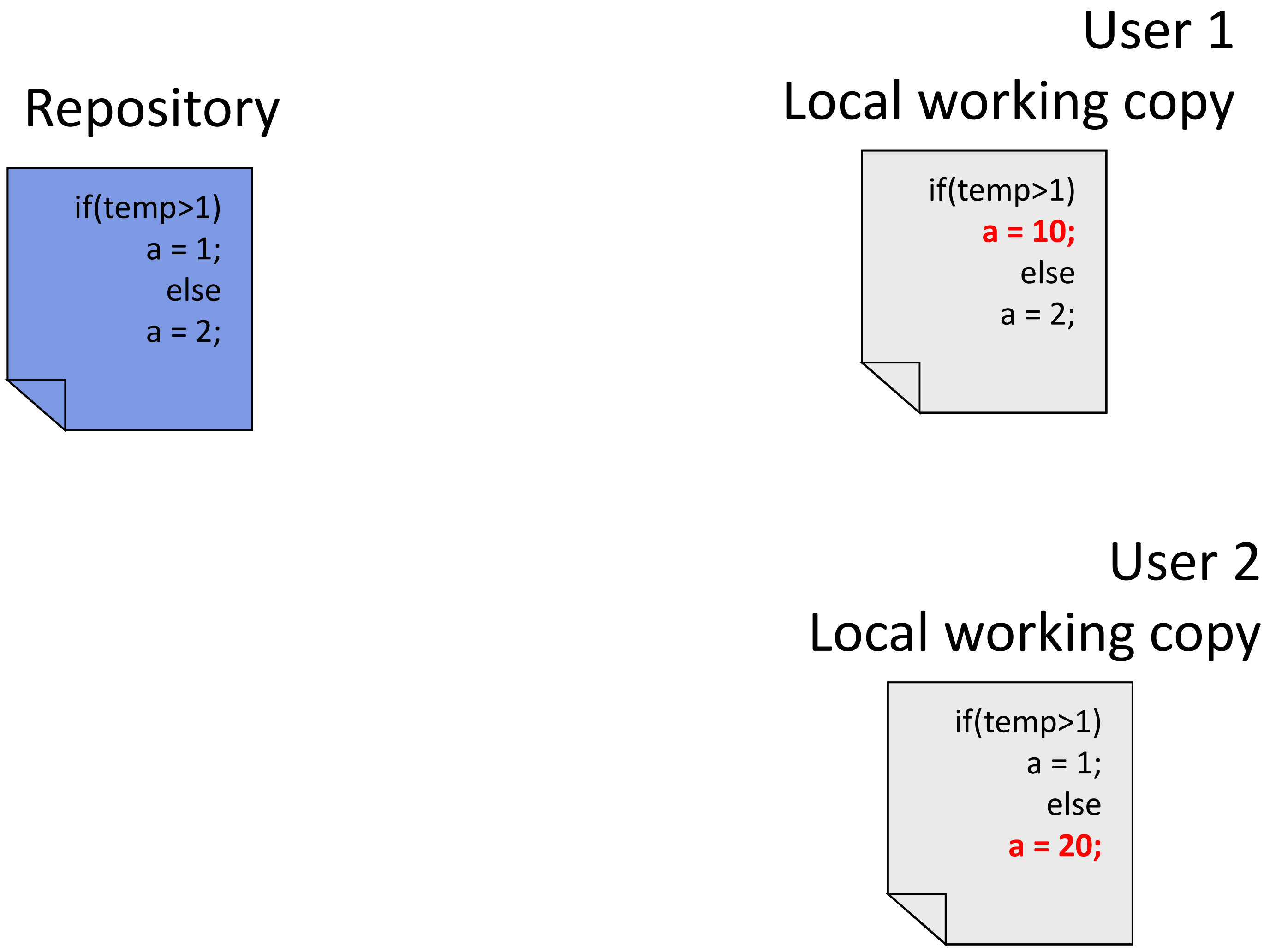
- Multiple developers **check out** from the repository to their **working copies**.
- During the **commit** phase they solve the **conflicts** by **merging** their versions.
- The Merging process is supported by the version control system, but it requires human interactions and decisions.

The Copy–Modify–Merge approach in work



```
if(temp>1)
    a = 1;
else
    a = 2;
```

The Copy–Modify–Merge approach in work



Repository:

```
if(temp>1)
    a = 1;
else
    a = 2;
```

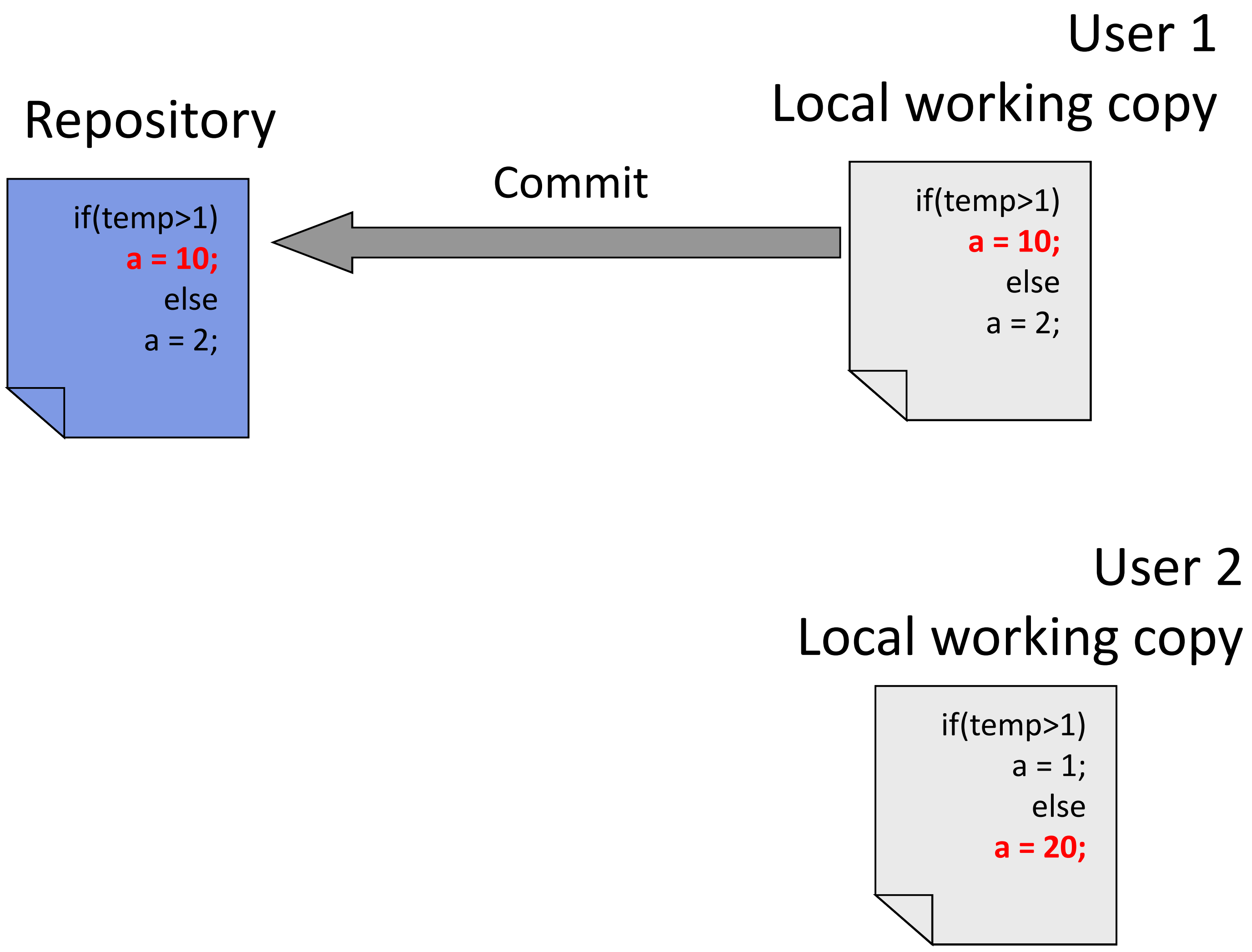
User 1 – Local working copy:

```
if(temp>1)
    a = 10;
else
    a = 2;
```

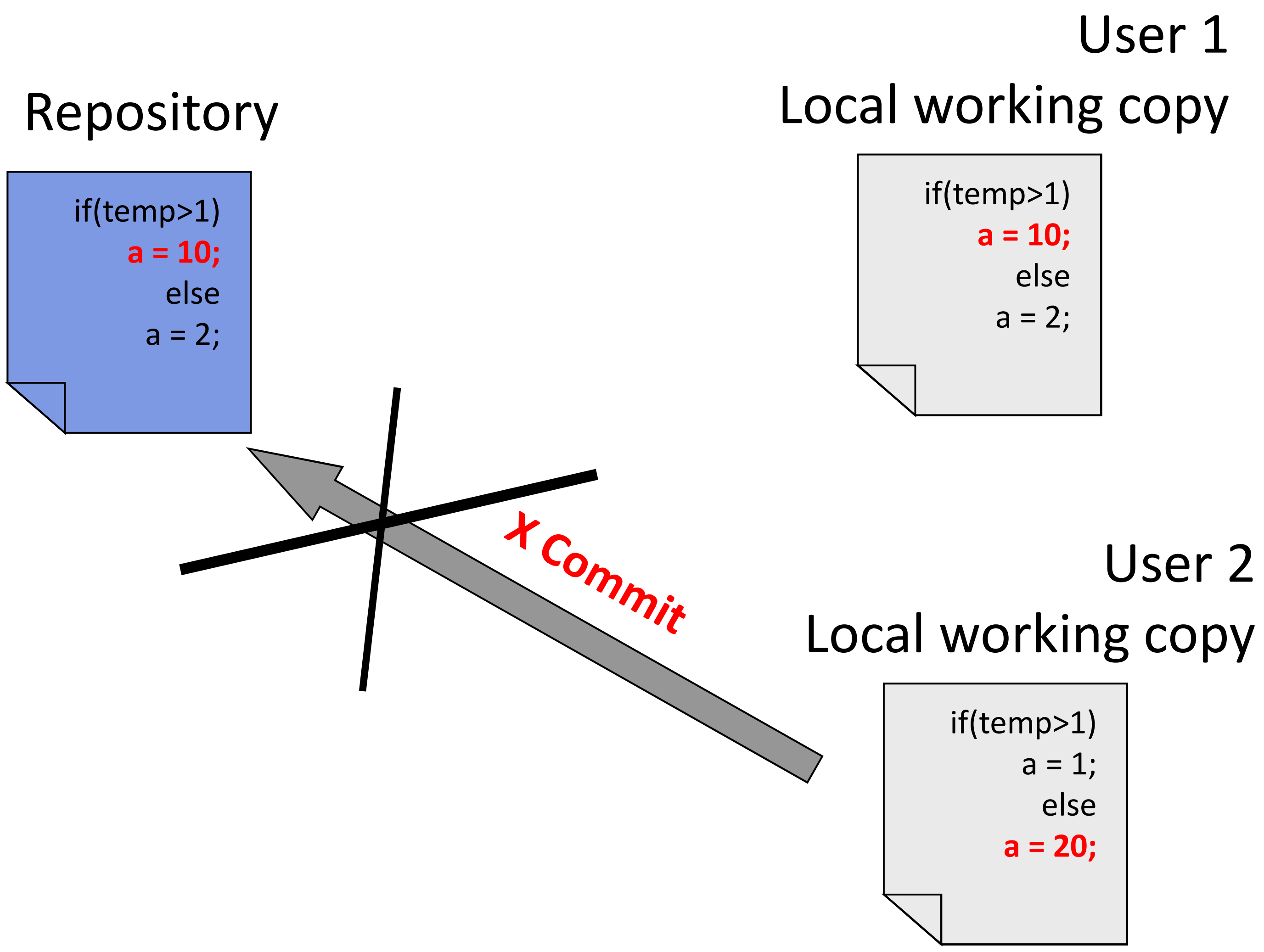
User 2 – Local working copy:

```
if(temp>1)
    a = 1;
else
    a = 20;
```

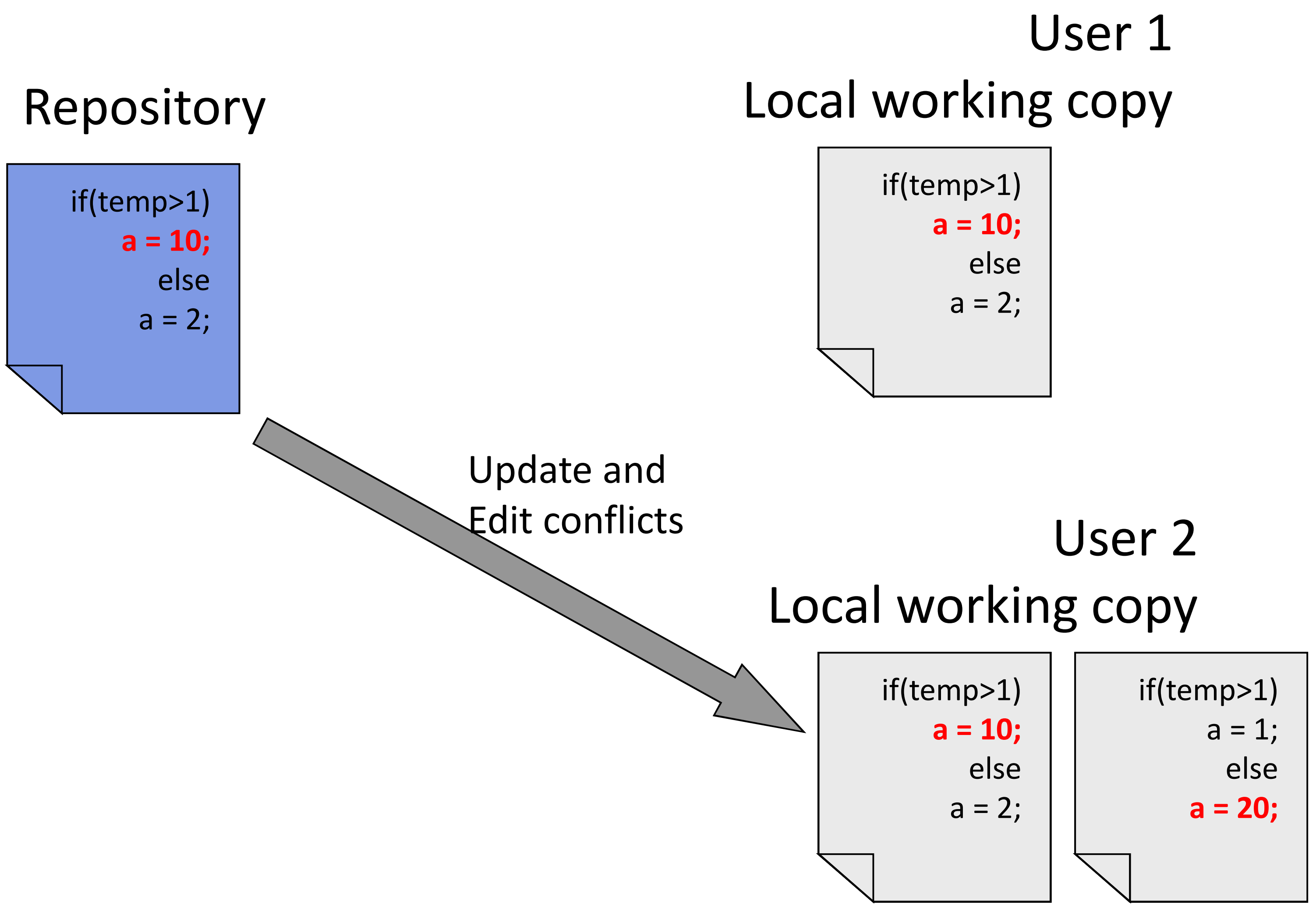
The Copy–Modify–Merge approach in work



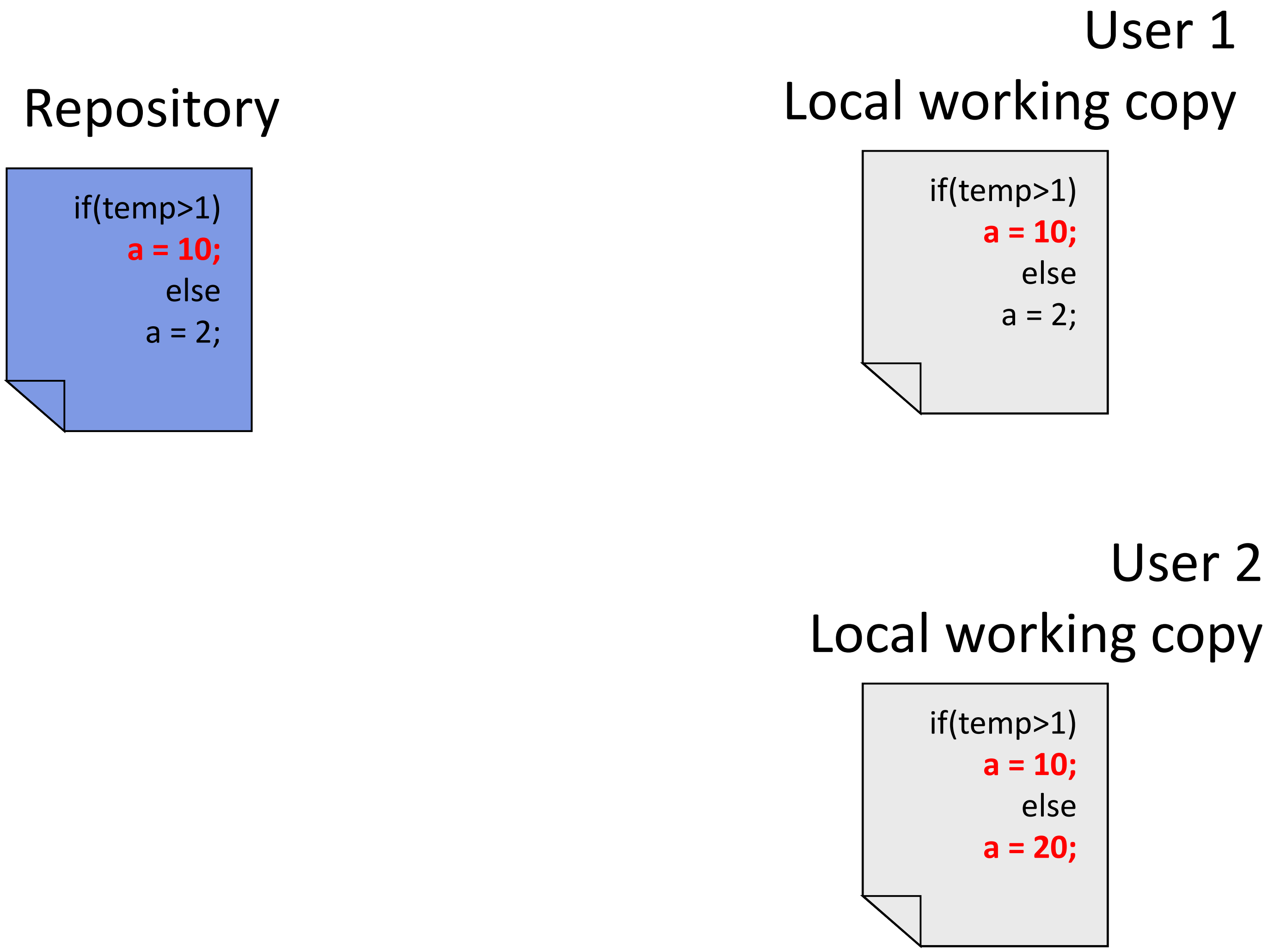
The Copy–Modify–Merge approach in work



The Copy–Modify–Merge approach in work



The Copy–Modify–Merge approach in work

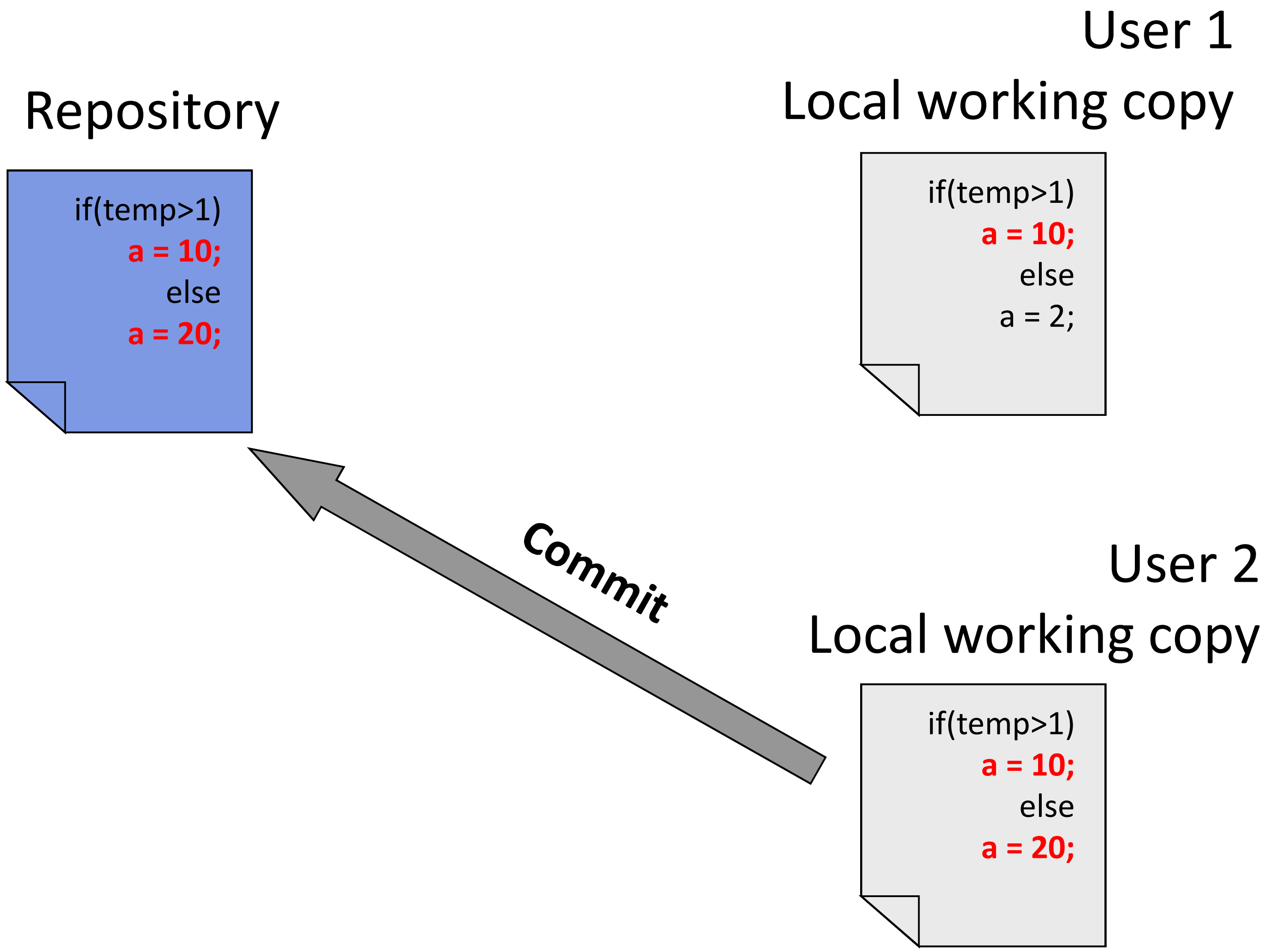


```
if(temp>1)
a = 10;
else
a = 2;
```

```
if(temp>1)
a = 10;
else
a = 2;
```

```
if(temp>1)
a = 10;
else
a = 20;
```

The Copy–Modify–Merge approach in work

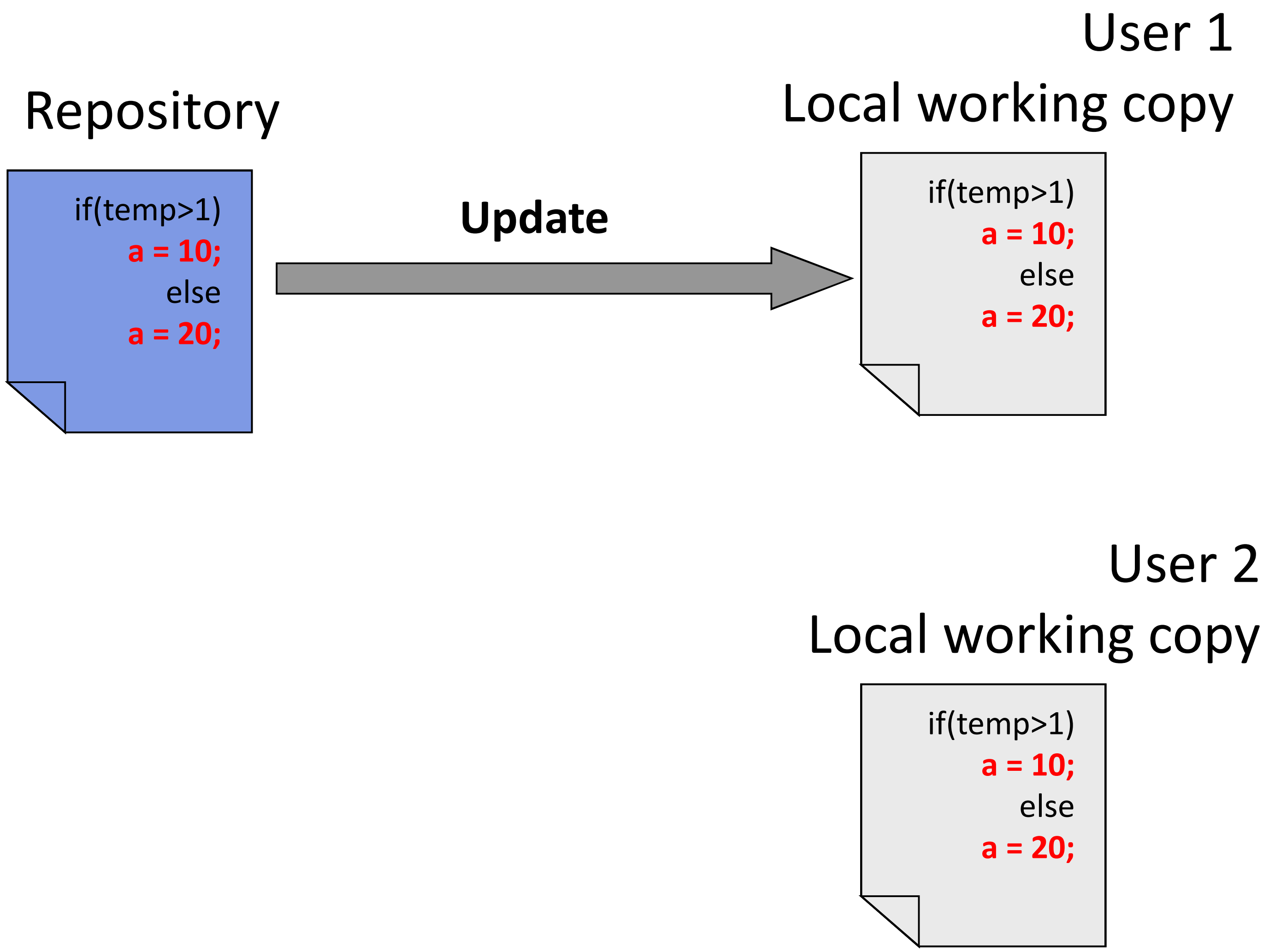


```
if(temp>1)
  a = 10;
else
  a = 20;
```

```
if(temp>1)
  a = 10;
else
  a = 2;
```

```
if(temp>1)
  a = 10;
else
  a = 20;
```

The Copy–Modify–Merge approach in work



```
if(temp>1)
    a = 10;
else
    a = 20;
```

A Copy–Modify–Merge megközelítés
merits and flaws

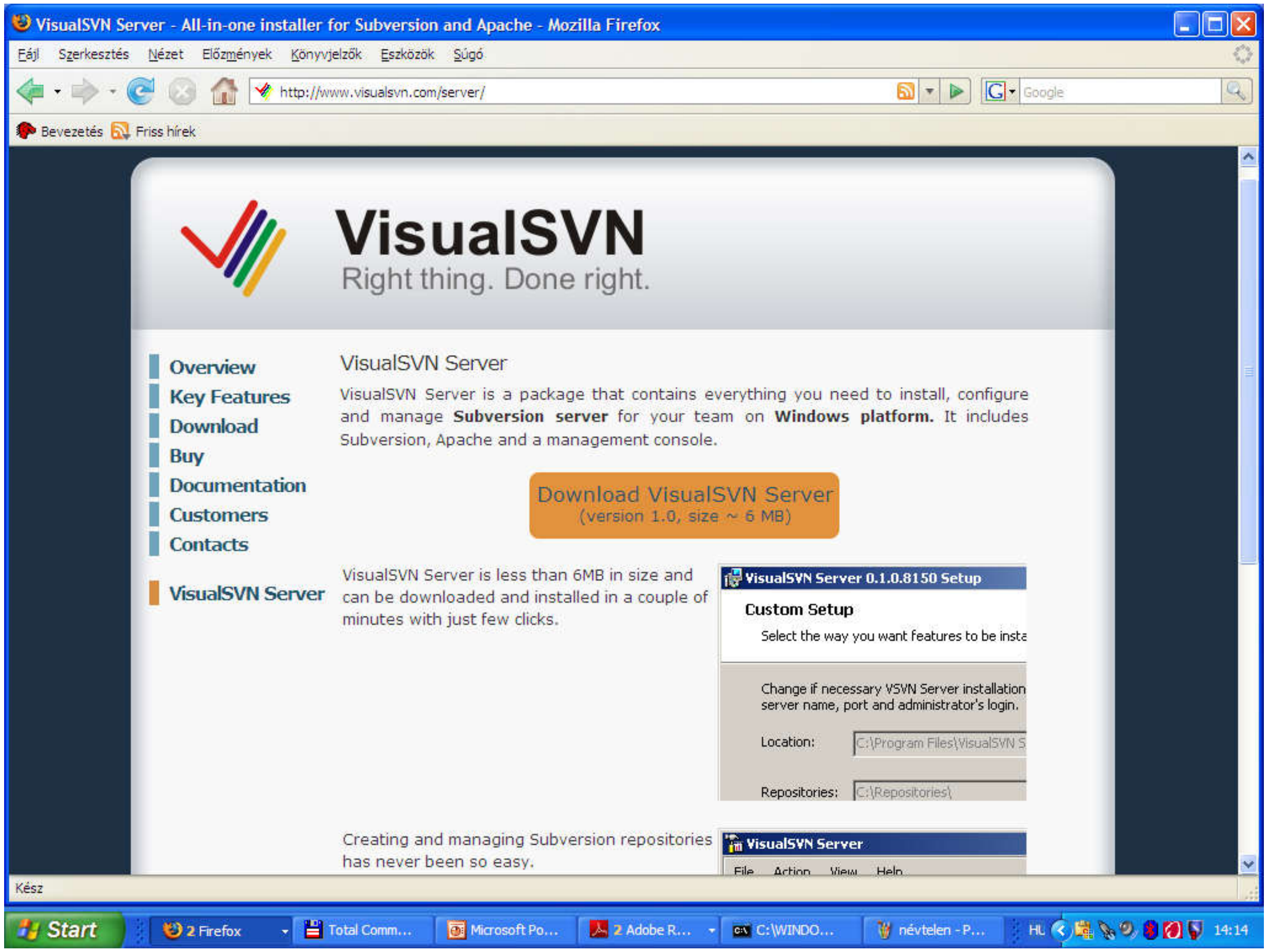
- It enables the parallel work of multiple developers
- Commit signals the conflicts
- Human interaction is needed to solve conflicts
- Version Control Systems do not replace the communication among team members

When do we need to use the lock-unlock approach?

- For binary files, where the text based merge is not possible.
  - Wav files, other raw data files
  - Outputs of some tools like PCB designers

- Therefore the lock function is available in most of the version control systems

Centralized Version Control Systems SVN, server solution example


VisualSVN Server - All-in-one installer for Subversion and Apache - Mozilla Firefox
Fájl Szerkesztés Nézet Előzmények Könyvjelzők Eszközök Súgó
http://www.visualsvn.com/server/
Bevezetés Friss hírek
VisualSVN
Right thing. Done right.
Overview
Key Features
Download
Buy
Documentation
Customers
Contacts
VisualSVN Server
Download VisualSVN Server (version 1.0, size ~ 6 MB)
Kész
Start

VisualSVN Server

VisualSVN Server is a package that contains everything you need to install, configure and manage **Subversion server** for your team on **Windows platform.** It includes Subversion, Apache and a management console.

VisualSVN Server is less than 6MB in size and can be downloaded and installed in a couple of minutes with just few clicks.

VisualSVN Server 0.1.0.8150 Setup

Custom Setup

Select the way you want features to be insta

Change if necessary VSVN Server installation server name, port and administrator's login.

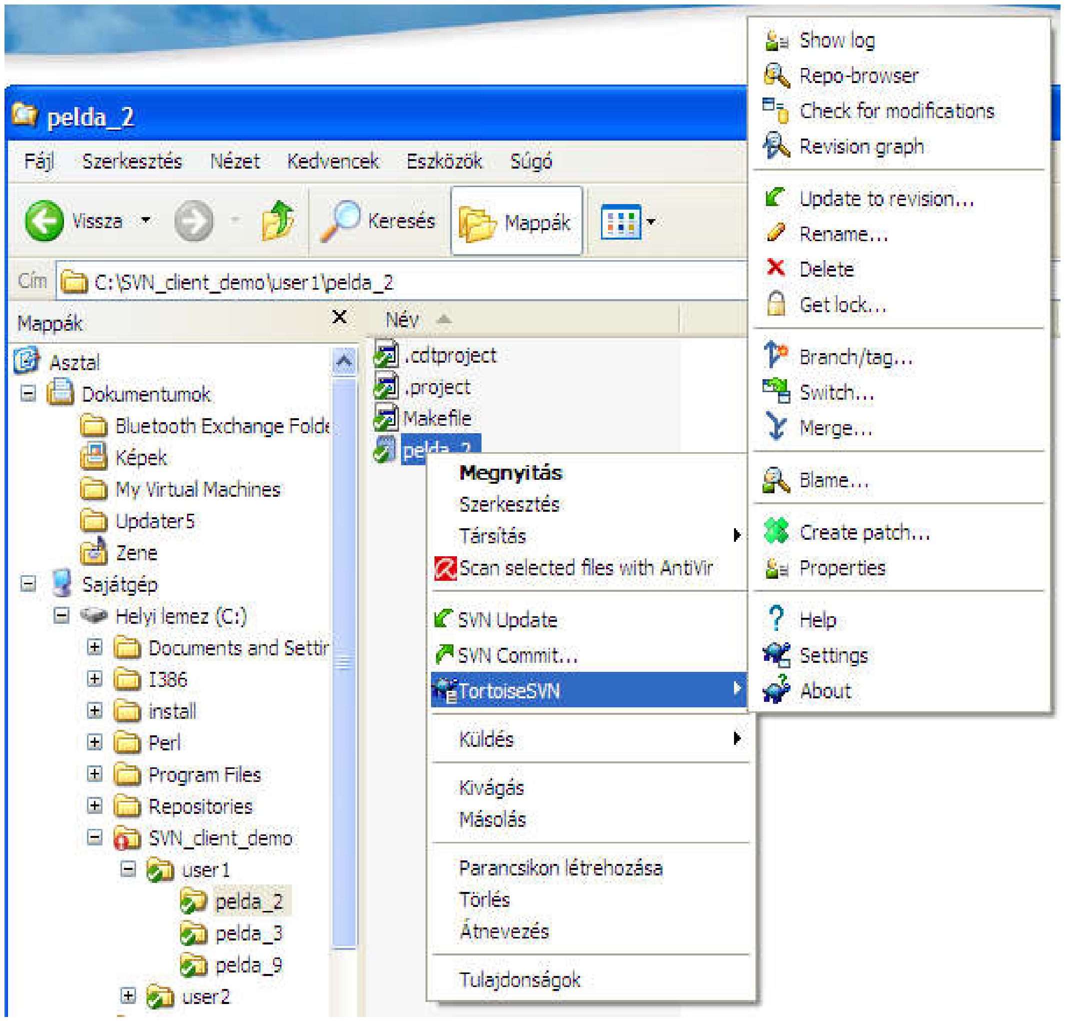
Location: C:\Program Files\VisualSVN S

Repositories: C:\Repositories\

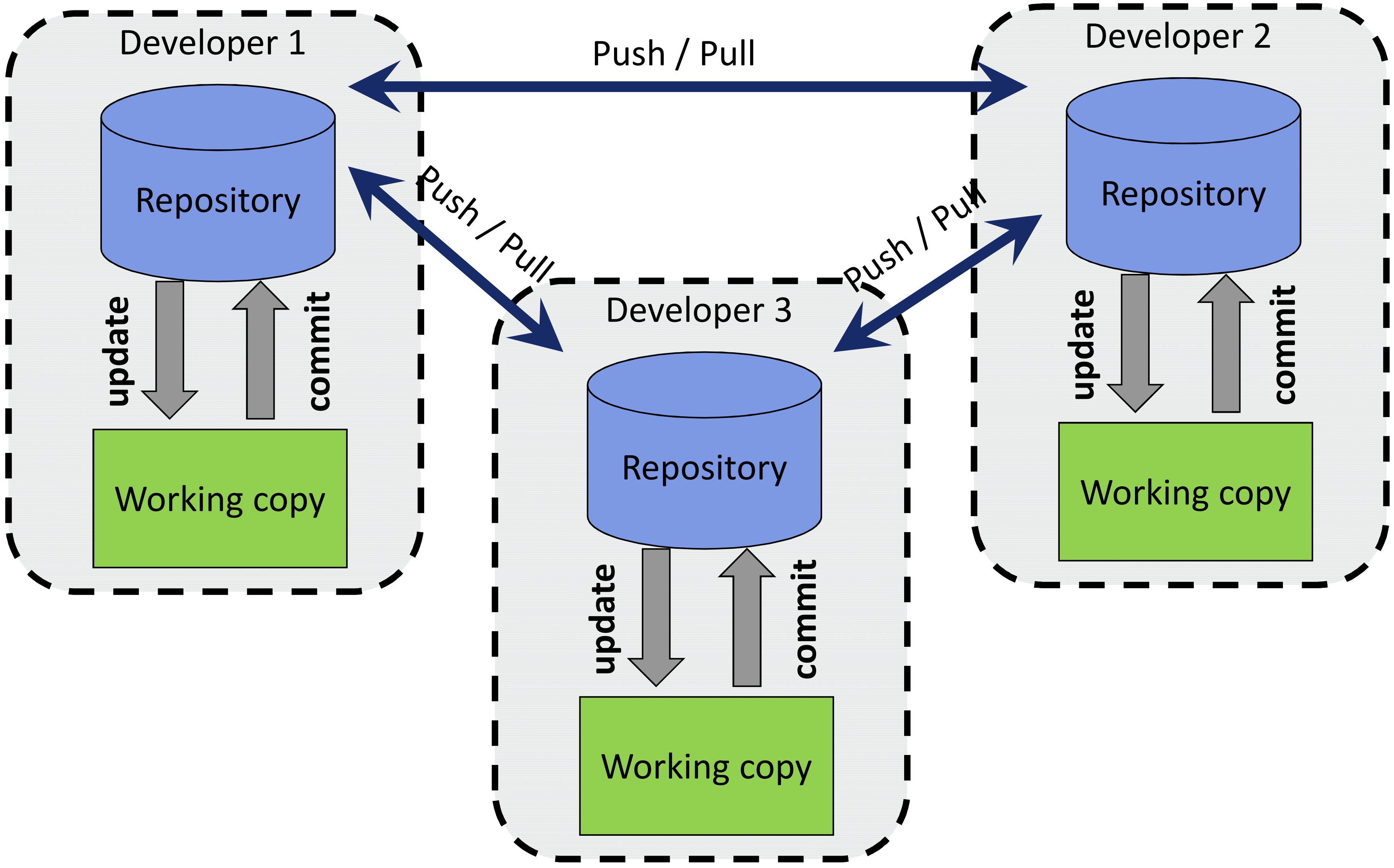
Creating and managing Subversion repositories has never been so easy.

Centralized Version Control Systems
SVN, client, TortoiseSVN

- Free SVN Client
  (there is a CVS version too)
  - http://tortoisesvn.net/
- It can overlay the icons of Windows



Distributed Revision Control



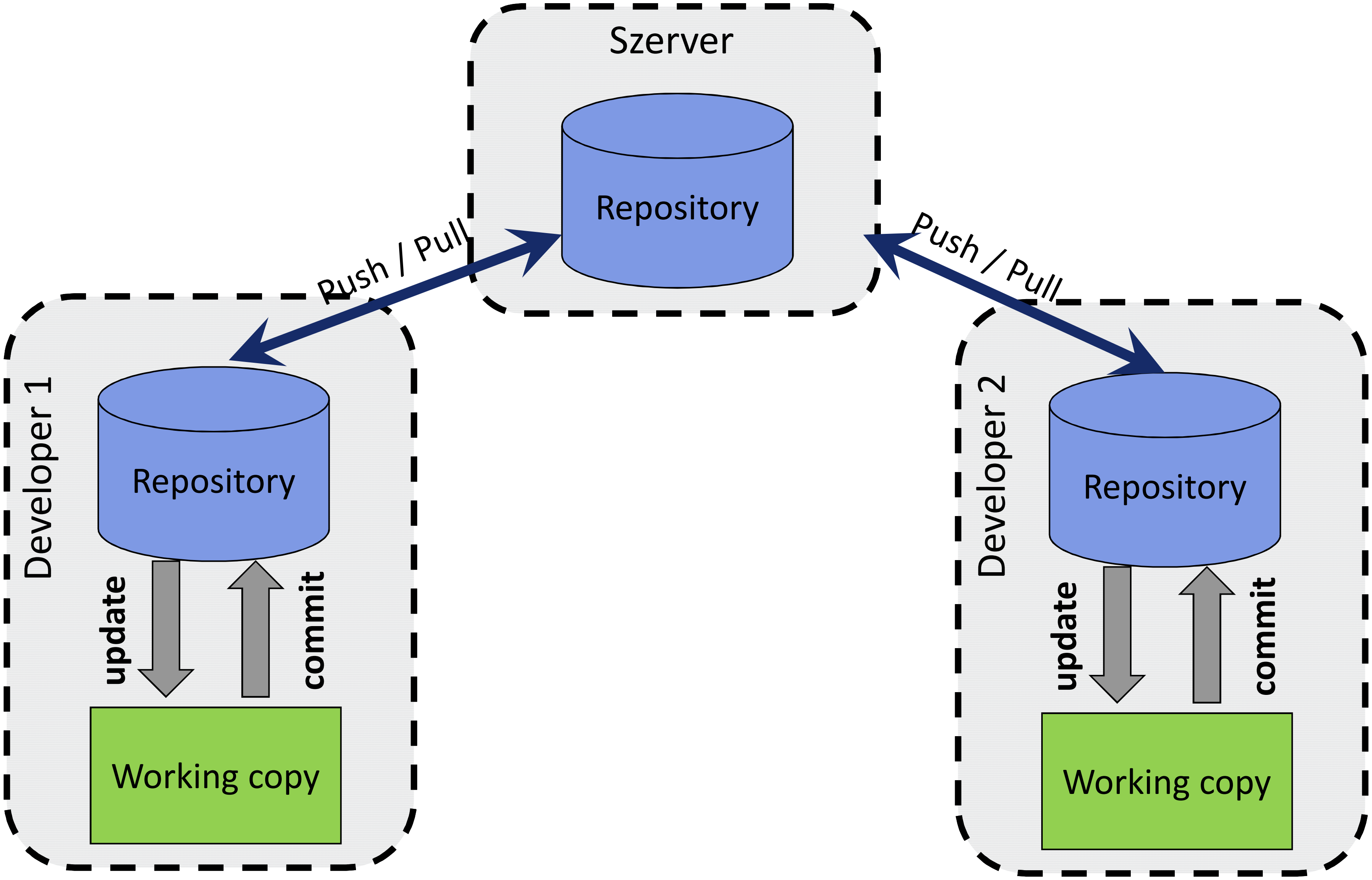
Distributed Revision Control merits

- Everyone has its own sandbox
  - Own repository, individual commit strategy
  - Easy to access the logs of own repository
- It works of line too
  - Centralized versions requires a server
- Fast
  - Don’t have to wait for the network communication
- Easy to manage
  - There is no need for a server
- Easy to make branches
  - Every developer has its own branch

Distributed Revision Control flaws

- There is still a need for back-up
  - The other developers repository cannot be considered as a back-up, because those can be very different
- There is no such us current release
  - Everybody has its own version
- There are no version numbers
  - Every change has its GUID (Globally Unique ID), but there is no such continuously like: rev 1, rev 2, rev 3

Distributed Revision Control
Usual approach



Distributed Revision Control
GIT “server side”

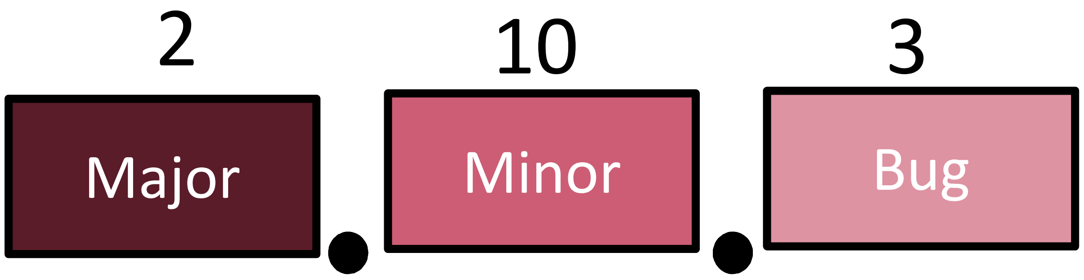
- According to terms there is not really one
- There are service providers like GitHub that can provide a centralized server for Git pushes (more then **26 million** repo)

Distributed Revision Control
GIT “client side”

- According to the terms there is not really one …
- GitHub for windows
- TortoiseGIT

tortoisegit
Windows Shell Interface to Git

Controlling Version Numbers
Semantic Versioning



- **Major:** major change that introduce incompatibility with previous verison. Like API (Application Programming Interface) change or functionality change.
- **Minor:** Change of functionality, but backwards-compatible API and features.
- **Bug:** backwards-compatible bug fixes.