Péter Mileff PhD

# Integrated Systems and Testing

## Version Control Systems, Source Control

Department of Information Science
University of Miskolc

# What will it be about?

- Clarification of the concept of version control
- Why is it needed? Who uses it? Where and How?
- Version control models
- Basic concepts in version control
- Operations
- Major version control systems
- Presentation of software tools
- Practical demonstration

# ABOUT VERSION CONTROLLING IN GENERAL...

# What is version control?

**Concept:**

- a set of procedures that allow variants (versions) of a data set to be managed together
- **In case of softwares:**
  - Store changes to source code during the software lifecycle
    - Most often, version support for source code files

  Management. E.g. logging, history, change version, rollback, who / when did what, etc
- **Designations:**
  - Revision Control, Version Control, Source Control, Source Code Management (SCM)

# Why is it needed?

- **Historical data is always needed during development!**
  - The source code goes through many iterations
    - Good to know when critical things happened
  - If there is a problem, we can revert to previous versions
  - It is independent of the number of developers
- **One-person development:**
  - No parallel development (within a project)
    - it is advisable to manage each version ourselves

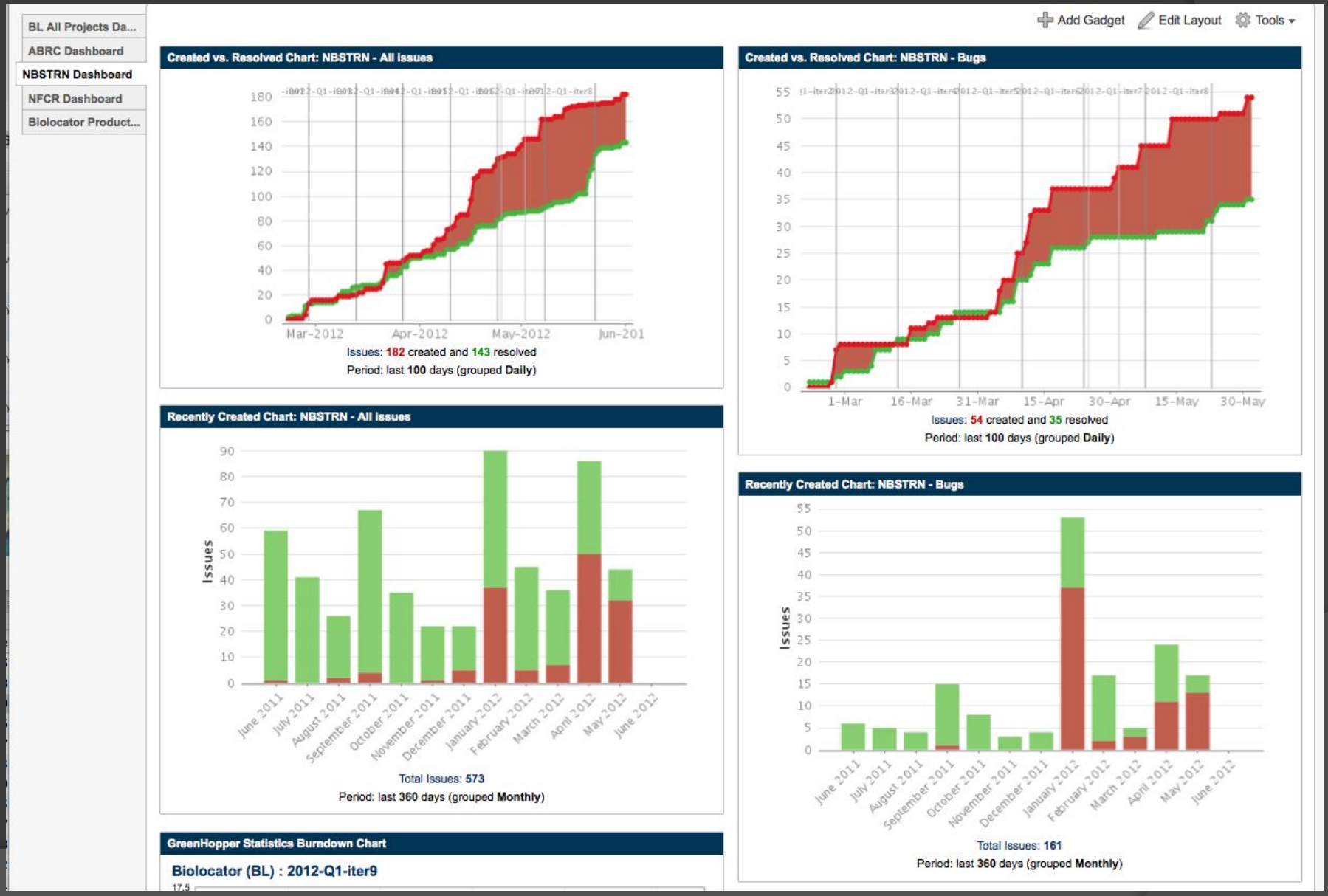
# Why is it needed?

- **Today, developing more serious software requires more people**
  - Tasks are typically done in **teams**
    - Continuous communication is required
    - The processing of tasks is parallel
      - Each member of the team works on a task
- This complex relationship needs to be managed
  - You need to see who developed what and when
  - You need to manage code merge when working on the same files
  - **Other:** marking special versions, merging versions, etc.

# Why is it needed?

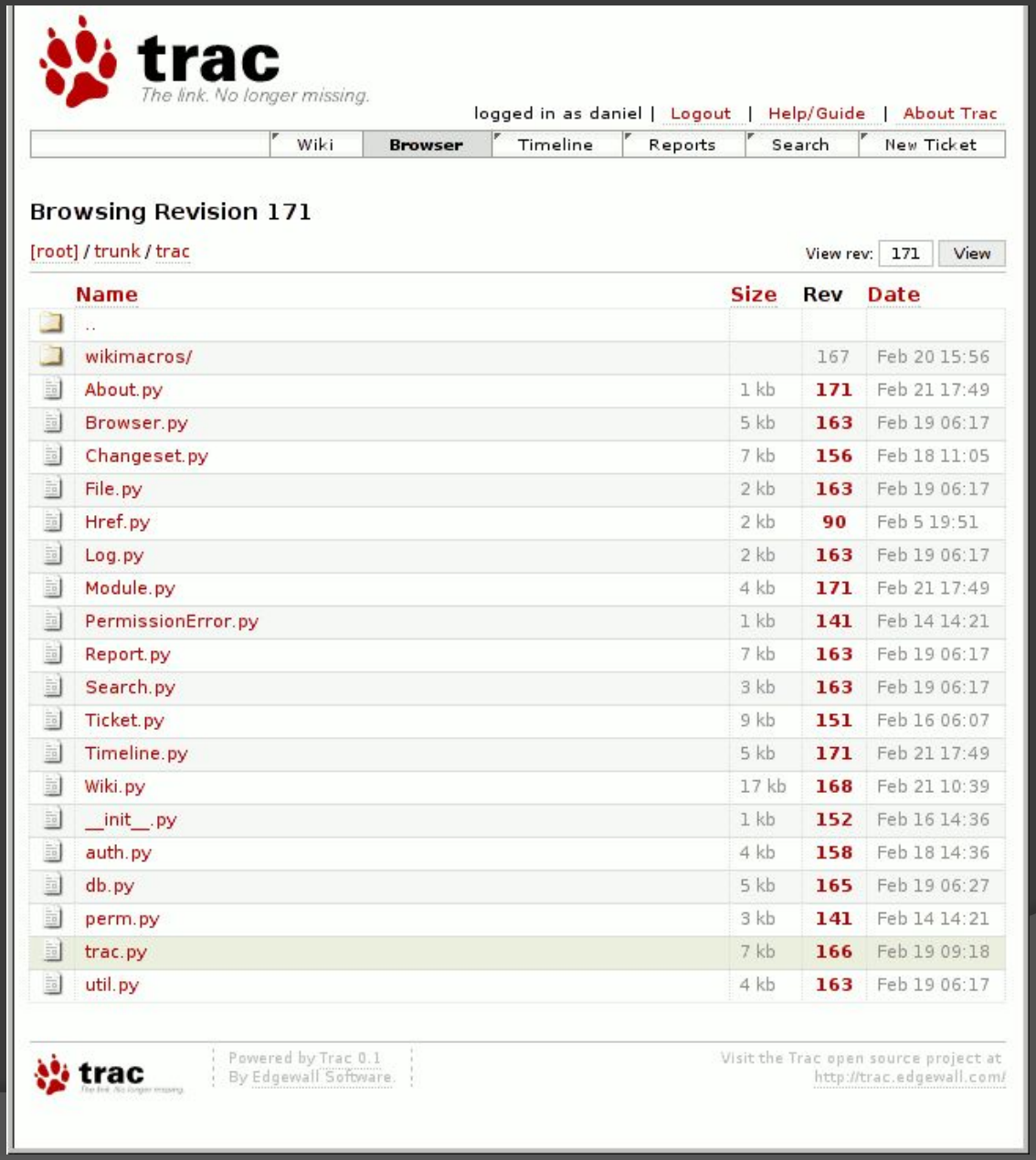
- Feedback can be provided to management
  - The process of development is clear
    - Who is working on what? Who implemented the gives feature.
- Version control systems can often be connected:
  - With task scheduler, project manager tools (eg JIRA, TRAC)
  - With Wiki systems
  - Other systems (like  Bugzilla)
- They provide a visual interface on the development process
  - Statistical data
  - Diagrams

# JIRA – Fisheye extension



# JIRA – Fisheye extension

# Trac


trac
The link. No longer missing.
logged in as daniel | Logout | Help/Guide | About Trac
Wiki
Browser
Timeline
Reports
Search
New Ticket
Browsing Revision 171
[root] / trunk / trac
View rev: 171
View
Name
Size
Rev
Date
..
wikimacros/ 167 Feb 20 15:56
About.py 1 kb 171 Feb 21 17:49
Browser.py 5 kb 163 Feb 19 06:17
Changeset.py 7 kb 156 Feb 18 11:05
File.py 2 kb 163 Feb 19 06:17
Href.py 2 kb 90 Feb 5 19:51
Log.py 2 kb 163 Feb 19 06:17
Module.py 4 kb 171 Feb 21 17:49
PermissionError.py 1 kb 141 Feb 14 14:21
Report.py 7 kb 163 Feb 19 06:17
Search.py 3 kb 163 Feb 19 06:17
Ticket.py 9 kb 151 Feb 16 06:07
Timeline.py 5 kb 171 Feb 21 17:49
Wiki.py 17 kb 168 Feb 21 10:39
__init__.py 1 kb 152 Feb 16 14:36
auth.py 4 kb 158 Feb 18 14:36
db.py 5 kb 165 Feb 19 06:27
perm.py 3 kb 141 Feb 14 14:21
trac.py 7 kb 166 Feb 19 09:18
util.py 4 kb 163 Feb 19 06:17
trac
Powered by Trac 0.1
By Edgewall Software.
Visit the Trac open source project at
http://trac.edgewall.com/

# Primitive Version Control

- The code is saved in a separate folder before each major change
  - we try to distinguish them properly
    - E.g.: give date or version number to directories
- **It works!**
  - But the least efficient version control technique
  - For one-person development only
- **Problem:**
  - over time, it becomes difficult to remember the differences in content between versions,
  - can take up a lot of storage
  - There is no software tool that provides extra features or help
    - E.g. Diff comparison

# MODELS OF VERSION TRACKING SYSTEMS...

# Main types of Version Control Systems

There are two general varieties of version control:

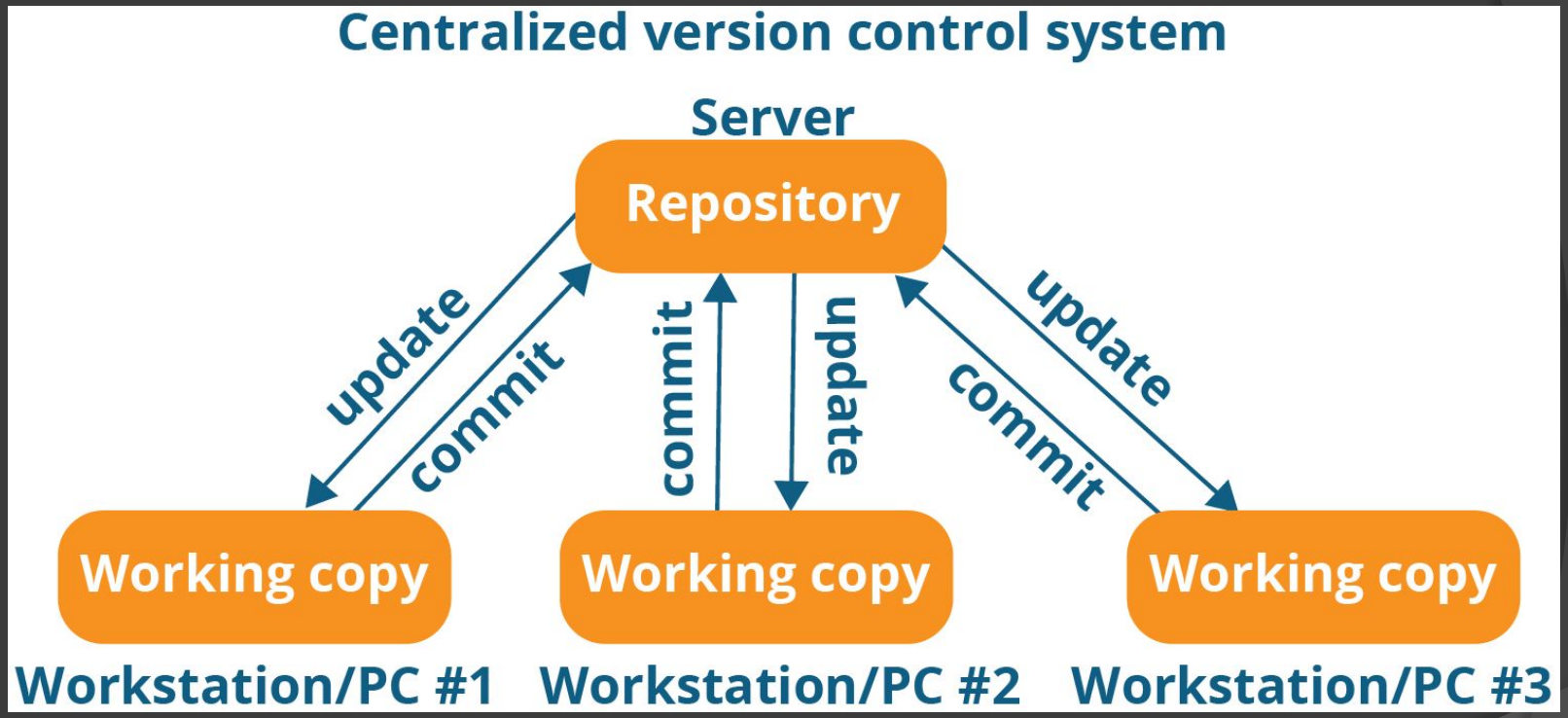
- Centralized model (traditional)
- Distributed version control systems

# Centralized Version Control System

- Server acts as the main repository which stores every version of code
- Every developer use this repository
  - Every operation is performed on the server

- **The workflow:**
  - it is practically a **commit** and **update** process
  - After each commit, it is recommended to update the working directory to get code changes created by others

# Centralized Systems




- **update:** getting the latest code from the server
- **commit:** put the local changes to the server

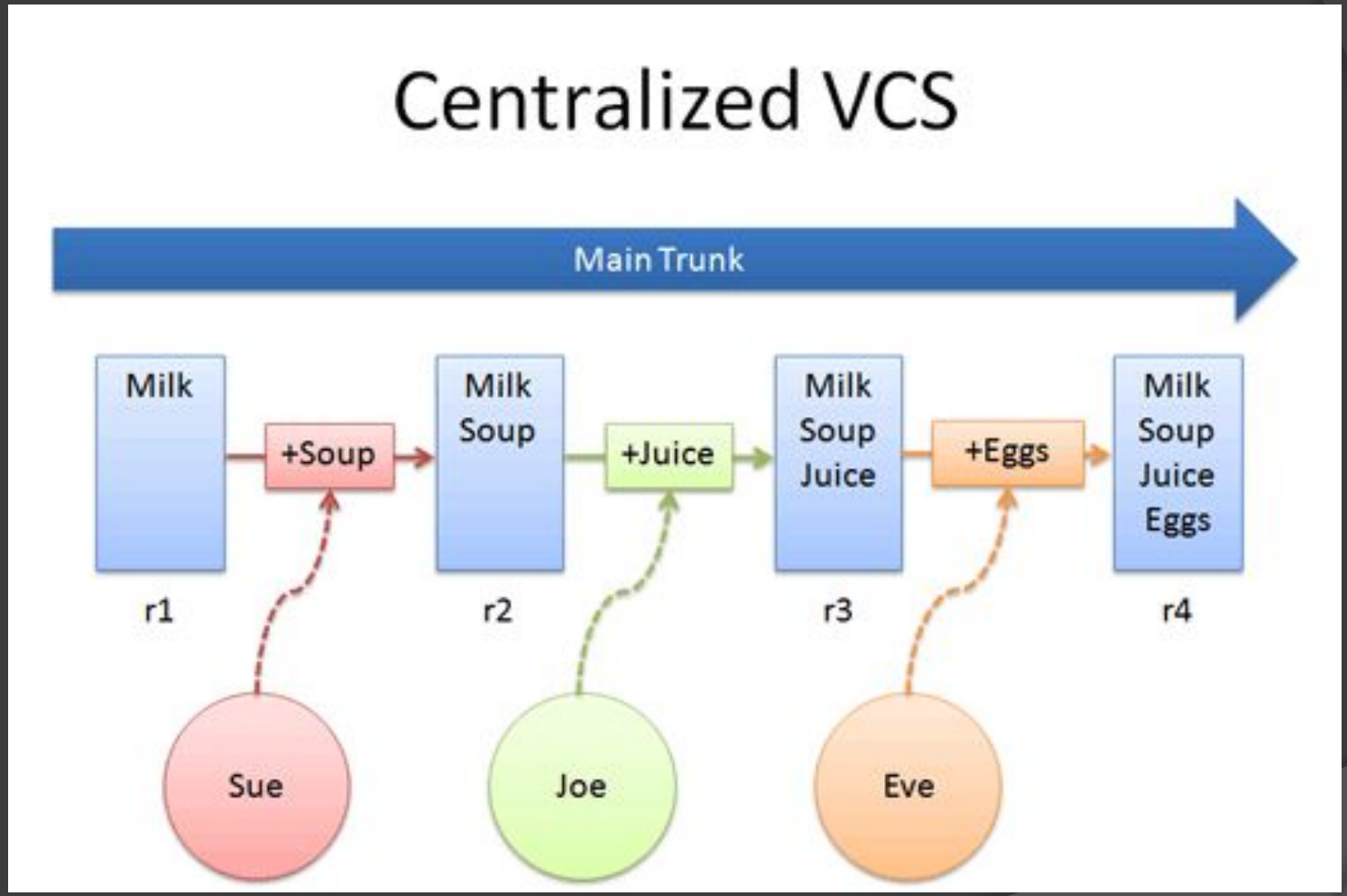
# Centralized Version Control System

- **Benefits:**
  - Centralized systems are typically easier to understand and use
    - developers of any skill level can push changes and start contributing to the code quickly
    - Setting up the system and the workflow is also simple
  - You can grant access level control on directory level
  - Performs better with binary files
  - Offers full visibility:
    - every team member has full visibility into what code is currently worked on and what changes are made
    - This knowledge helps software development teams understand the state of a project and provides a foundation for collaboration
    - centralized version control system only has two data repositories that users have to monitor: the local copy and the central server.

# Centralized Version Control System

- **Disadvantages:**
  - A single point of failure risks data:
    - If the remote server goes down, then no one can work on the code or push change
    - The lack of offline access means that any disruption can significantly impact code development and even result in code loss
    - The entire project and team comes to a standstill during an outage
  - Slow speed delays development:
    - Branching becomes a time-consuming:
      - users must communicate with the remote server for every command, which slows down code development
      - task and allows merge conflicts to appear, because developers can't push their changes to the repository fast enough for others to view

# Centralized Version Control System




Every developer synchronize and commit their changes

# Distributed Version Control Systems...

# Distributed Version Control Systems

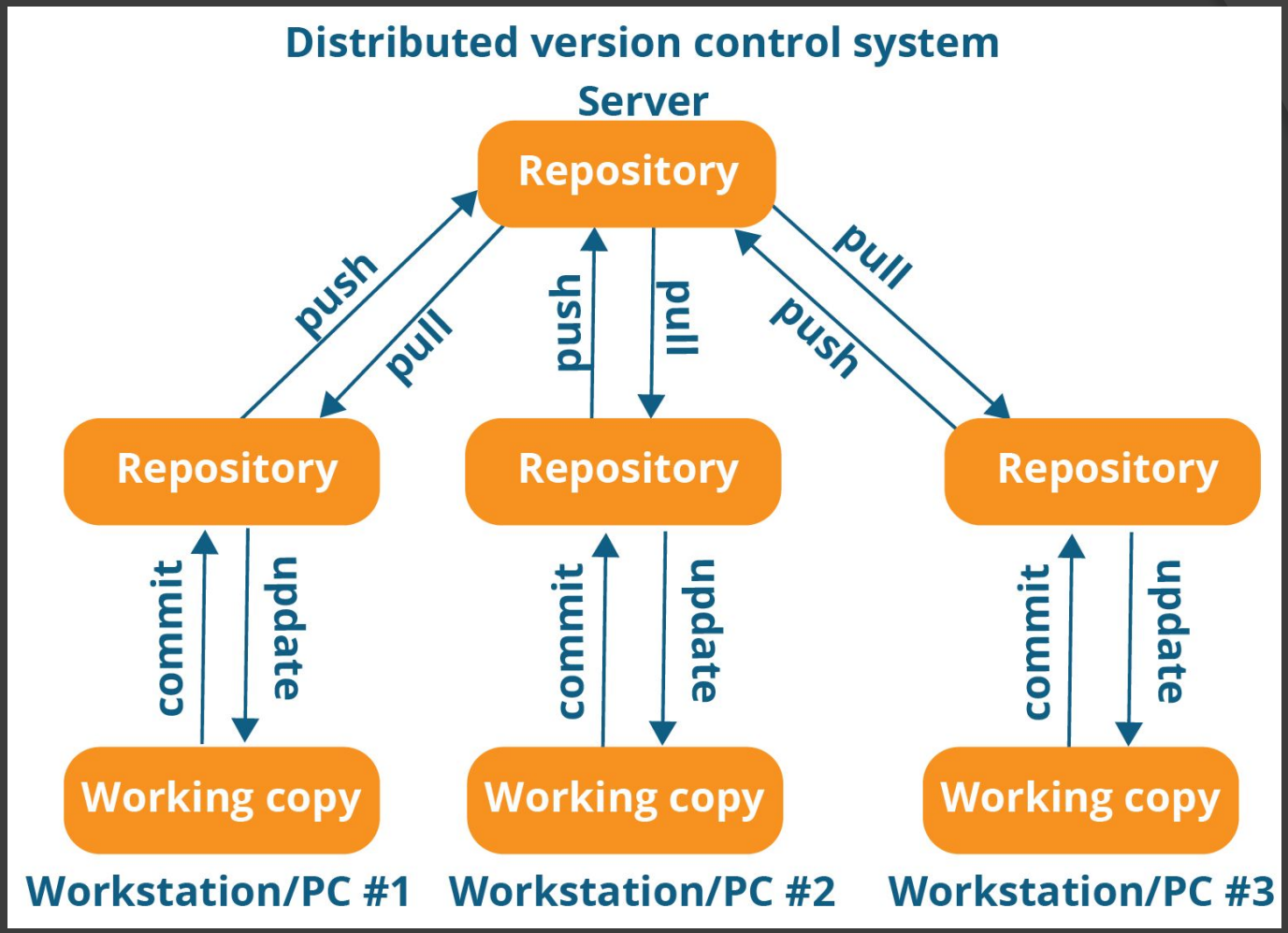
- **A type of version control where the complete codebase is mirrored on every developer's computer**,
  - including its full version history
- Synchronization is accomplished by patches sent between each machine
- **Sounds wasteful, but in practice, it's not a problem:**
  - Most programming projects consist mostly of plain text files
    - maybe a few images
  - Disk space is cheap
    - storing many copies of a file doesn't create a noticeable dent in a hard drive's free space.
  - Modern systems also compress the files to use even less space.

# Distributed Version Control Systems

- **Advantages:**
  - Branching and merging is much easier
    - Branching and merging can happen automatically and quickly
  - With a distributed system, we don't need to be connected to the network all the time
    - Developers have the ability to work offline
    - complete code repository is stored locally
  - Multiple copies of the software eliminate reliance on a single backup
  - Performance of distributed systems is better

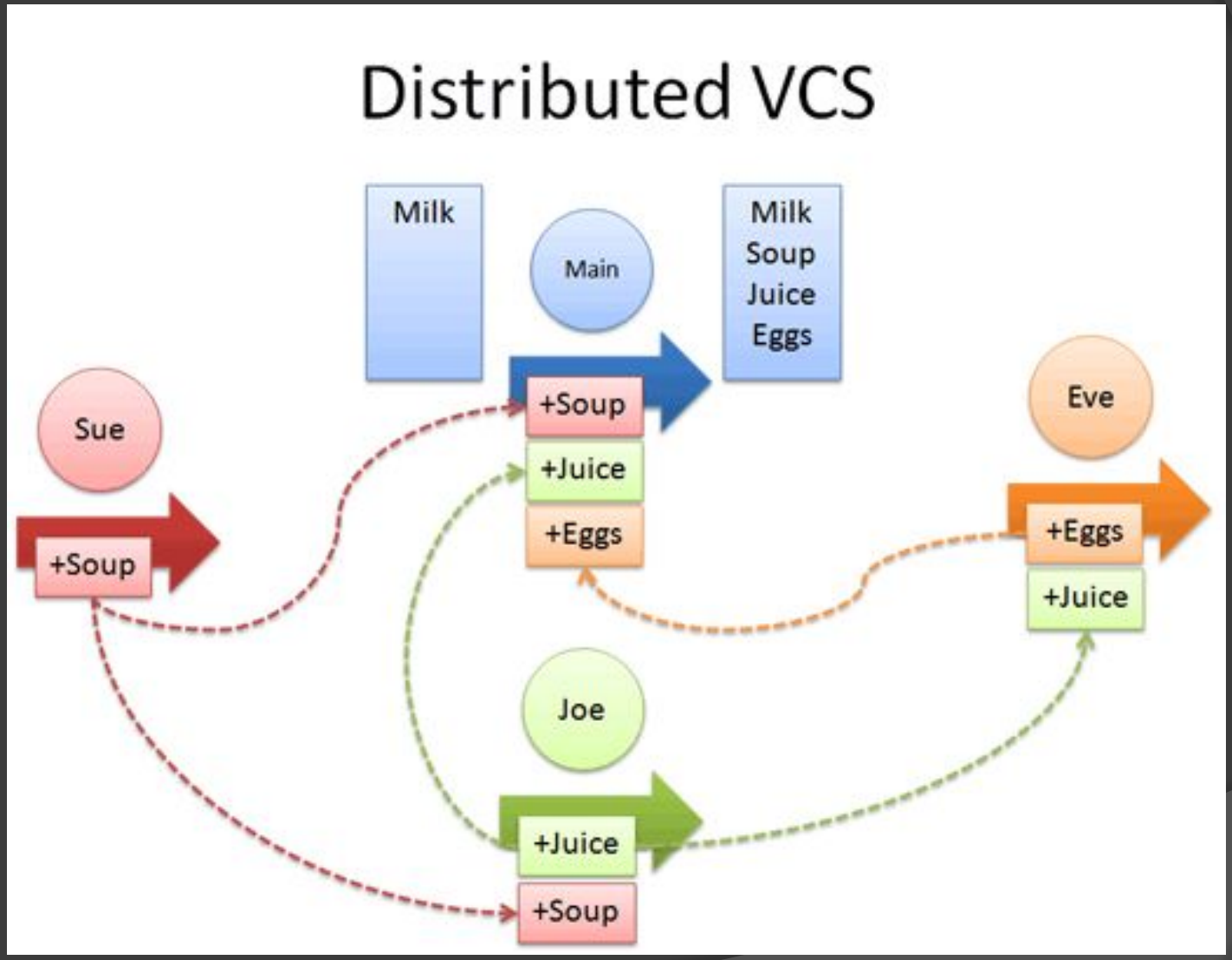
# Distributed Version Control Systems

- **Disadvantages:**
  - It may not always be obvious who did the most recent change
  - File locking doesn’t allow different developers to work on the same piece of code simultaneously.
  - It helps to avoid merge conflicts, but slows down development DVCS enables you to clone the repository – this could mean a security issue
  - Working with a lot of binary files requires a huge amount of space, and developers can't do diffs




- **pull:** getting the latest code from the server
- **update:** update local repository with the code getting with pull
- **commit:** put the local changes to the local repository
- **push:** send local commits to server

# Distributed Version Control Systems



# ALAPFOGALMAK…

# Basic definitions

- The logical operation of the version control softwares may differ
  - **But the applied definitions are the same!**

- **Repository:** **repo** in short. A data structure that stores metadata for a set of files or directory structure
  - stores all project files and their versions
  - usually a special directory structure with special files
  - so each project must be stored in a separate repo!

- **Working copy:**
  - A copy of some of the code that a developer is currently working on on their own machine.
  - Upon completion of the work, its status/change will be stored in the repository in the form of commits

# Basic definitions

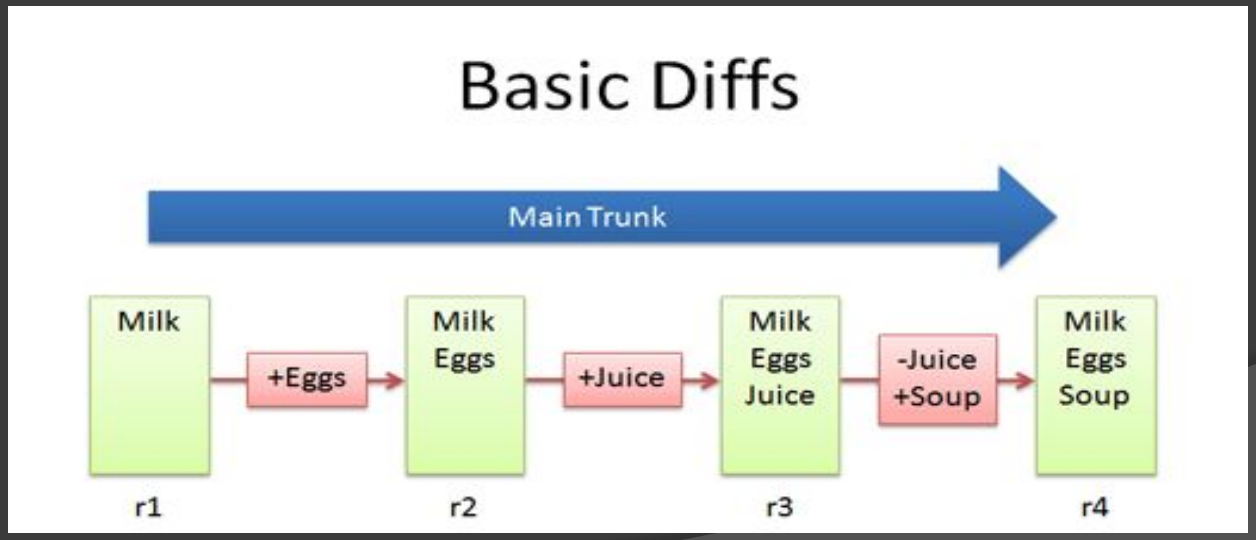
- **Commit:**
  - The “commit” command is used to save changes to the repository
  - Every set of changes implicitly creates a new, different version of the project
    - Therefore, every commit also marks a specific version.
  - It’s a snapshot of your complete project at that certain point in time.
  - It can be used to restore the project to that certain state
- Every commit item consists the following metadata:
  - unique id/hash — every commit has a unique identifier.
  - date — information when commit happened. It helps later on to lists the commits in chronological order
  - author — information who performed changes
  - message — the author of a commit has to comment what he did in a short “commit message”.

# Basic definitions

- **Revision:** a version
  - After each commit, the value of the revision in the repo increases,
    - so this is the version number. E.g: r3522
- **Checkout:**
  - Make a local copy of a versioned file.
  - By default, the user will receive the latest version,
    - but it is also possible to request a specific version based on version number
- **Head:** indicates the most recent commit (version) at the top of the current branch
- **Push(ing):** upload data to the main repository
  - only at distributed version control systems (e.g. git, mercurial)

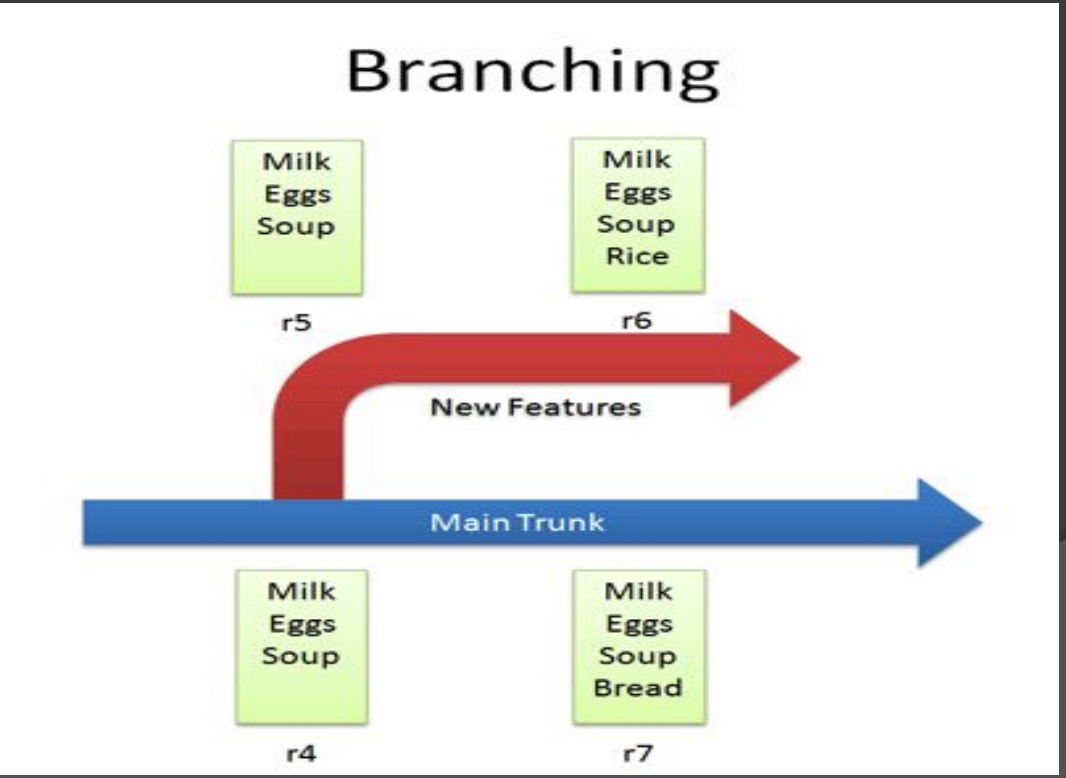
# Basic definitions

- **Trunk:** It represents the main branch of development. Essentially it is also a branch with a special name
- **Update:** it incorporates the changes in the repo into the user's working copy, i.e. the local version.
- **Diff/Change/Delta:** find / show change between two files.



# Basic definitions

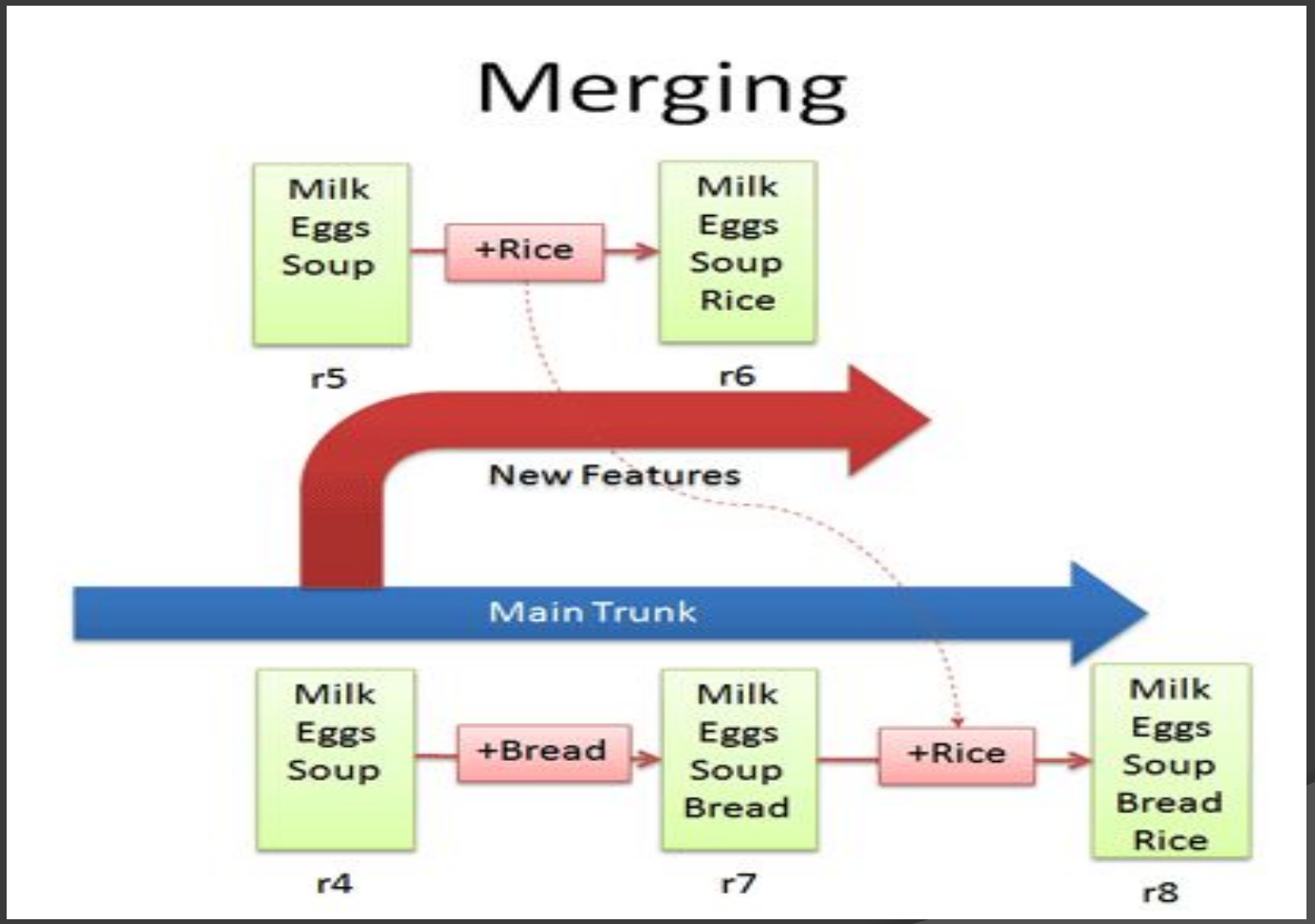
- **Branch:** is a copy of a codeline, managed by a version control system (VCS).
  - Branching helps software development teams work in parallel. It separates out “in-progress work” from tested and stable code.
  - It is an alternative development “line”



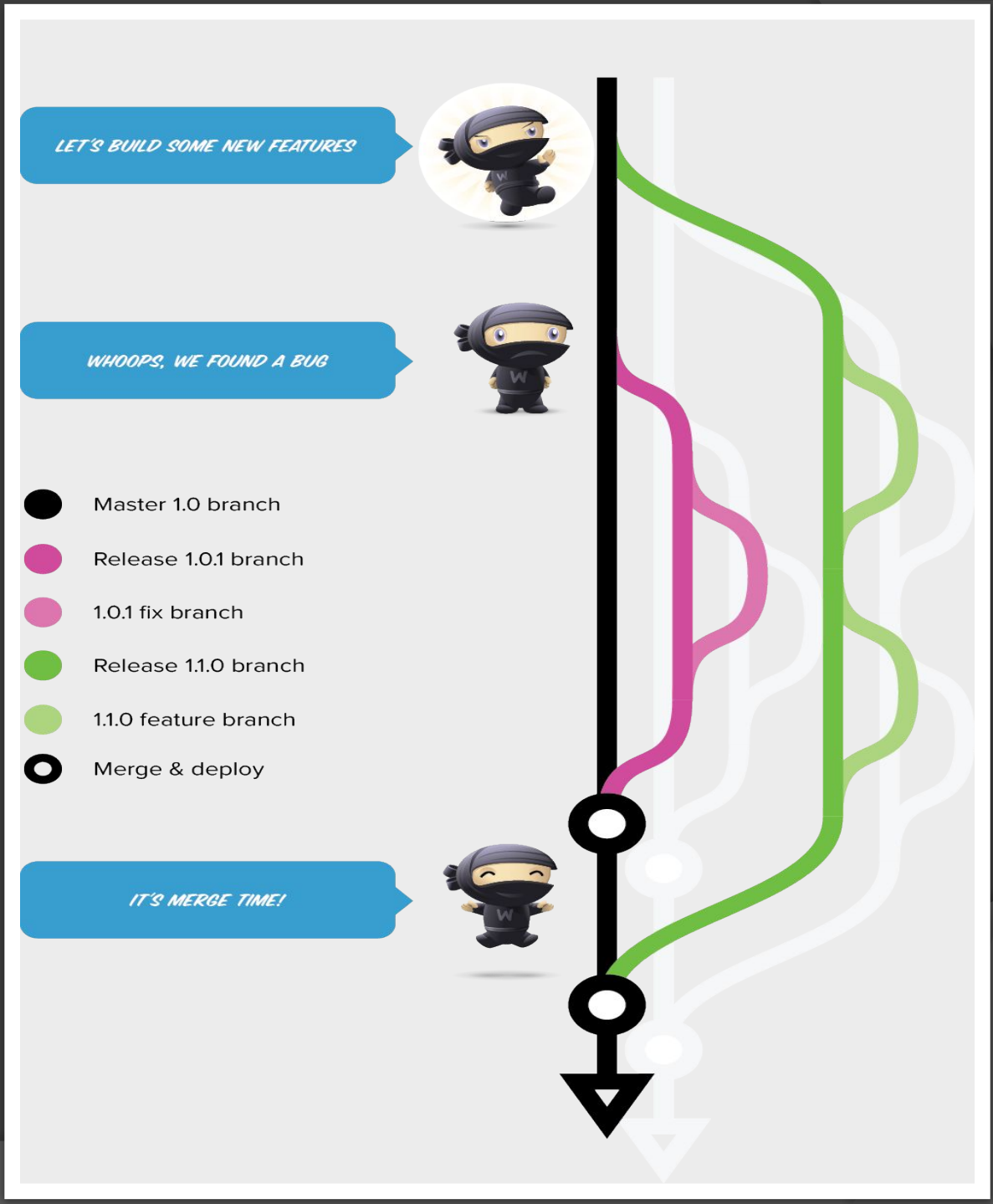
# Basic definitions

- **Merge:** there will come a time when we want to integrate changes from one branch into another
  - for example: we finished developing a feature and want to integrate it into the “production” branch
- Or the opposite:
  - we are not yet finished working on the feature,
  - but so many things have happened in the rest of the project in the meantime
  - we want to integrate these back into the feature branch.
  - Such an integration is called “merging”

# Basic definitions



# Merge


LET'S BUILD SOME NEW FEATURES
WHOOPS, WE FOUND A BUG
IT'S MERGE TIME!
Master 1.0 branch
Release 1.0.1 branch
1.0.1 fix branch
Release 1.1.0 branch
1.1.0 feature branch
Merge & deploy

# Basic definitions

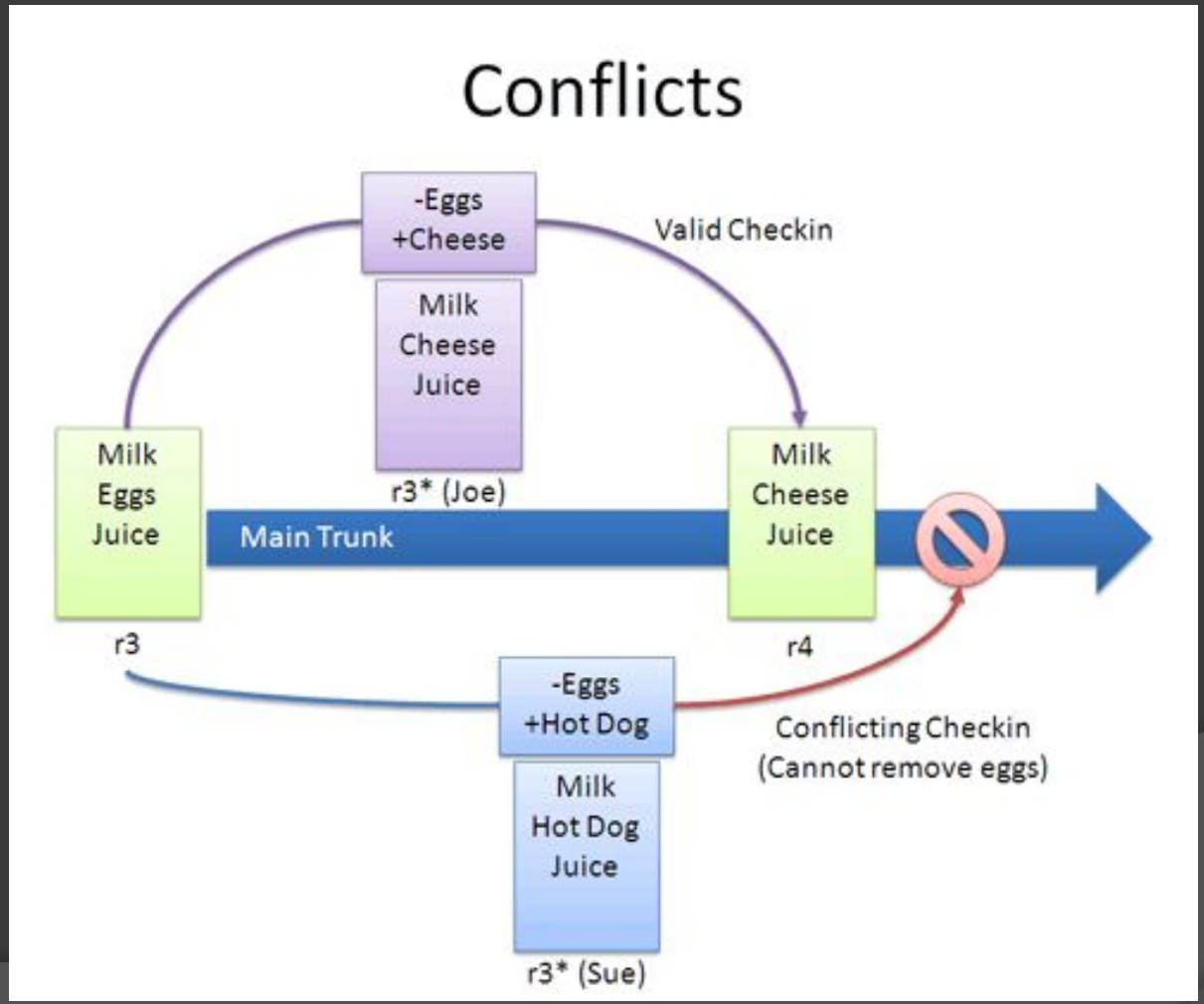
- **Conflict:**
  - Phenomenon arising from the merging of branches
  - The version of the two branches contains code that cannot be merged automatically
  - The merging process must be performed manually
  - Modern IDEs provide a graphical interface for this

- **Example (SVN):**

```
<<<<<<< .mine
This is fun stuff!
=======
This is a documentation file
>>>>>>> .r6
```

# Basic definitions

- **Conflict:**

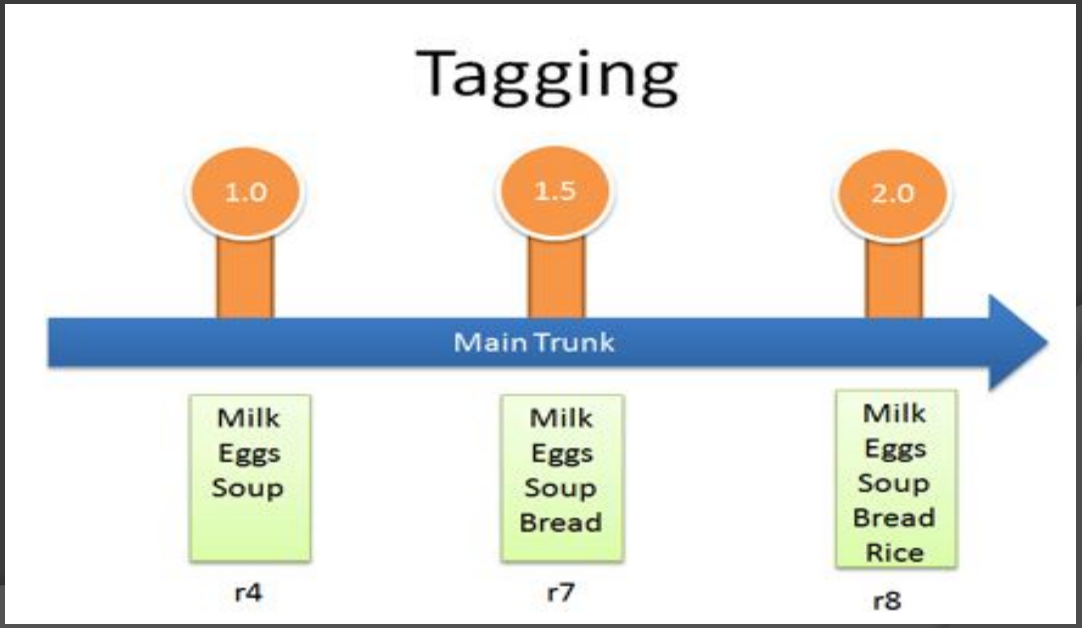


# Basic definitions

- **Another way to avoid conflict:**
  - **Lock:** tilos a konkurens hozzáférés
    - if someone starts modifying a file, it cannot be opened for writing by another user
    - In case of complex source code changes, merge conflicts can be avoided
    - Locking a file for too long can cause problems for other users

# Basic definitions

- **Version tagging:**
  - A tag represents a version of a particular branch at a moment in time (tag mark a specific commit in your repository history).
  - Tags are commonly used to mark release versions, with the release name as the tag name (i.e. v1.0.1).
  - A tag is like a branch that doesn't change.
    - Unlike branches, tags, after being created, have no further history of commits.



# Version Control Systems in practice…

# Grouping aspects of version control systems

- Repository model (central or distributed)
- Supported platforms (Linux, Windows,..)
- Price (free, paid licence)
- How the history is handled (model)
  - changeset, patch, snapshot
- Version identifier method (Revision ID: namespace, sequence, pseudorandom)
- Supported network protocol (http, https, ftp,sftp,ssh)
- Open vs. Closed source code

# Most well known VCSs

- **GIT:** free, distributed. one of the best vcs tool in the present market
  - The Linux source code is stored in git
- **Mercurial:** free, distributed, open source, one of the best
- **Concurrent Versions System(CVS):**
  - free, centralized, open source, one of the oldest
- **Subversion(SVN):** free, open source, centralized. before git, it was the most used VCS
- **Bazaar:** free, distributed, open source
- **Team Foundation Server (TFS - Microsoft):** based on client-server, distributed repository model and has a proprietary license

# Portals for developers

## GitHub

- GitHub helps software teams to collaborate and maintain the entire history of code changes.
- Weu can track changes in code, undo errors and share our efforts with other team members.
- It is a repository to host Git projects.

## GitLab

- It comes with a lot of handy features like an integrated project, a project website
- Using the continuous integration (CI) capabilities, we can automatically test and deliver the code.
- We can access all the aspects of a project, view code, pull requests, and combine the conflict resolution.

# Portals for developers

## Bitbucket

- Bitbucket is a part of the Atlassian software suite,
- it can be integrated with other Atlassian services including HipChat, Jira, and Bamboo.
- The main features of Bitbucket are code branches, in-line commenting and discussions, and pull requests.

## AWS CodeCommit

- It is a managed version control system that hosts secure and scalable private Git repositories.
- It seamlessly connects with other products from Amazon Web Services (AWS) and hosts the code in secured AWS environments.
- It is a good fit for the existing users of AWS.

# SVN overview

- Open source version control system
  - Unix, Linux, Windows, OSX, BSD, Solaris, BeOS, Haiku, etc
- Use it to manage changes to files and directories over time.
- The storage logic is similar to an average file server,
  - except that it records all changes to files and directories.
- **What it offers:**
  - full version control management from command line

http://subversion.apache.org/

# SVN overview

- **Create an SVN server:**

  **svnserve - Linux**

  **svnserve.exe - Windows**

- Built-in lightweight server:
  - It is installed with an installation package
  - Communicates via TCP / IP protocol
  - HAs an own protocol - svn: //
  - Able to communicate with ssh tunnel

**Run SVN server as a daemon:**

**svnserve.exe -d -r c:/MySVNRepo**

# SVN protocols

| Protocol | Description |
| --- | --- |
| file:// | direct access respository (on local disk) |
| http:// | We can integrate SVN into main webservers. Like: apache, ningx, lighthttpd |
| https:// | Same as http://, but with SSL secure |
| svn:// | Communicate with svnserve server through its own protocol |
| svn+ssh:// | Same as svn://, but over SSH tunnel |

# SVN usage

- Repo creation (server): the basic file structure is created

**svnadmin create MyRepo**

- Creating working copy (svn checkout):
  - create a working copy on the client side

**svn checkout place_of_repo where_to_save_it**

**E.g.:**

**svn checkout http://example.org/svn/MyRepo C:/LocalRepo**

**SSH tunel sample:**

**svn co svn+ssh://example.org/svn/MyRepo C:/LocalRepo**

# SVN usage

- Add new file to our working copy:

  **svn add sample.txt**

- Remove file from repository:

  **svn del sample.txt**

- Comitting changes into repository:
  - All changes will be sent into the repository

  **svn commit –m `Commit text`**

- Get latest code from the repository:

  **svn update**

# SVN usage

- Revert changes of a file:

```
svn  revert  test.c
```

- Creating a branch:

```
svn copy svn+ssh://example.com/svn/MyRepo/trunk
svn+ssh://example.com/svn/MyRepo /NAME_OF_BRANCH
    -m "Creating a branch of project"
```

- Merging:
  - Merge branch into revision 250 of the main branch

```
svn merge -r 250:HEAD
http://example.com/svn/MyRepo/branches/my-branch
```

# SVN usage

Make a tag:

**svn copy http://path/to/revision http://path/to/tag**

# Well know SVN clients

- Tortoise SVN, RapidSVN
- More advanced version controls allow integration with other devices
- Version control add-ons for different IDEs are often available for download
  - Graphical based diff, merge, commit, revert
  - Sync view, history, and more
- **Eclipse/Netbeans based clients:**
  - Subversive
  - Subclipse

GAME OVER