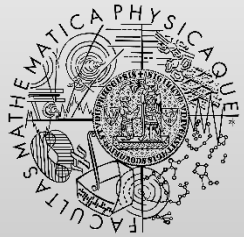
Distributed Version Control

http://d3s.mff.cuni.cz

Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Key concepts

- Each developer uses a private local repository
  - *clone*: full mirror of some existing repository
- Operations performed on the local repository
  - very fast, off-line
- Synchronization
  - Operations *push* and *pull*
  - Exchanging code patches

Department of Distributed and Dependable Systems D3S

Comparing distributed and centralized VCS

- Centralized
  - Everything visible in the central repository
  - Private branches (work) not possible
- Distributed
  - Private repositories (and branches) useful for experimental development

Tools

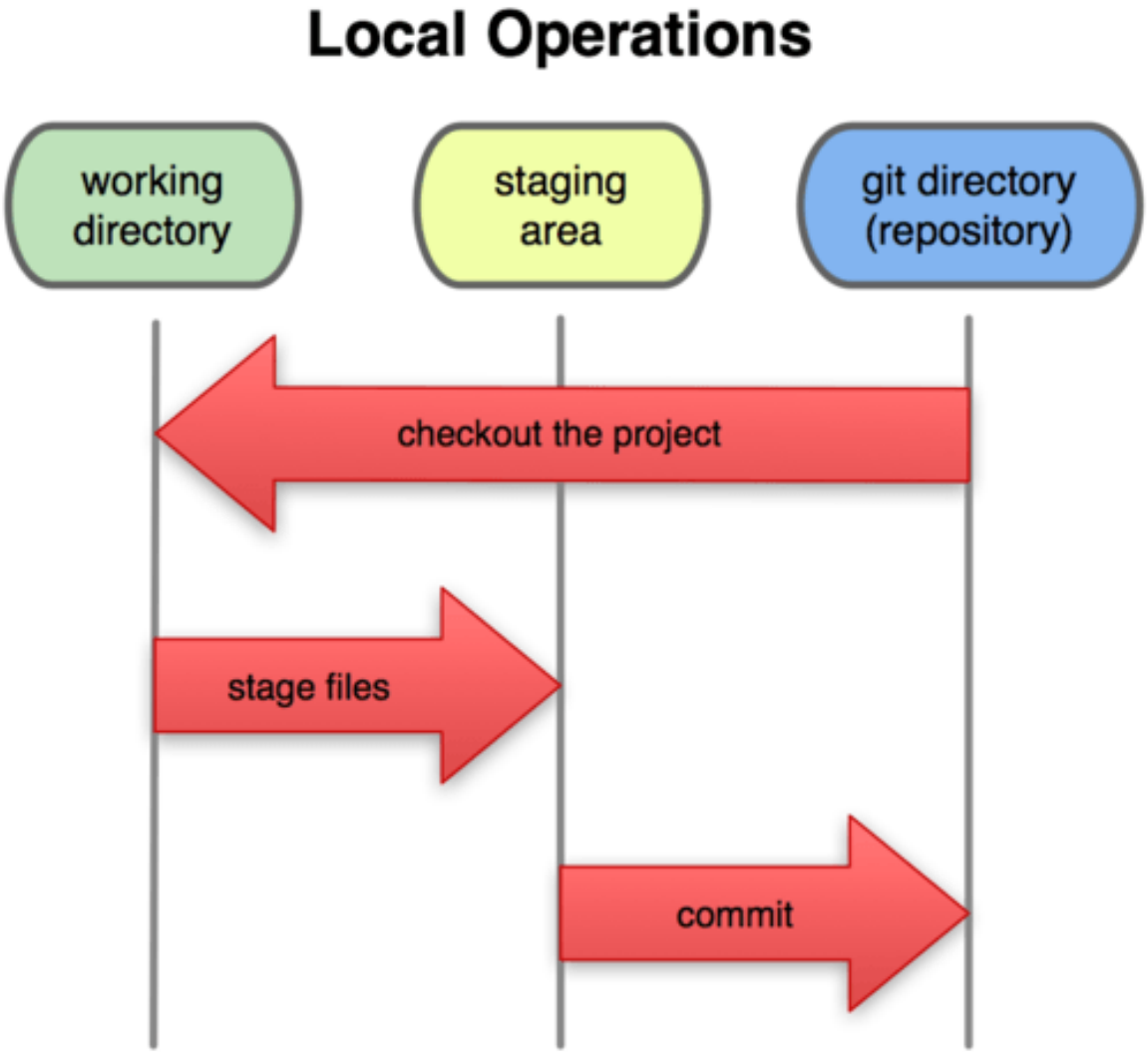
- Git
- Mercurial
- Bazaar

Git

Main features

- Versions: snapshots of the project (working dir)
- Committed revisions form a direct acyclic graph
  - Multiple “latest” versions (leaf nodes)
- Each commit has an author and committer
  - Distributing changesets via patches (email)
- Whole repository stored in `.git` (files, metadata)

Usage scenario



Picture taken from http://git-scm.com/book/

Necessary setup

- Configure your identity
  - `git config --global user.name "<your full name>"`
  - `git config --global user.email "<your email address>"`
- Stored in `$HOME/.gitconfig`

Basic commands

- Help for specific command: `git help <command>`
- Create repository in the current directory: `git init`
- Print status of the working tree: `git status`
- Start tracking new files: `git add <work dir path>`
- Add files to the staging area: `git add <path>`
- Commit staged modifications: `git commit -m "..."`
- Print uncommitted unstaged changes: `git diff`
- Print staged uncommitted changes:

  ```
  git diff --staged
  ```

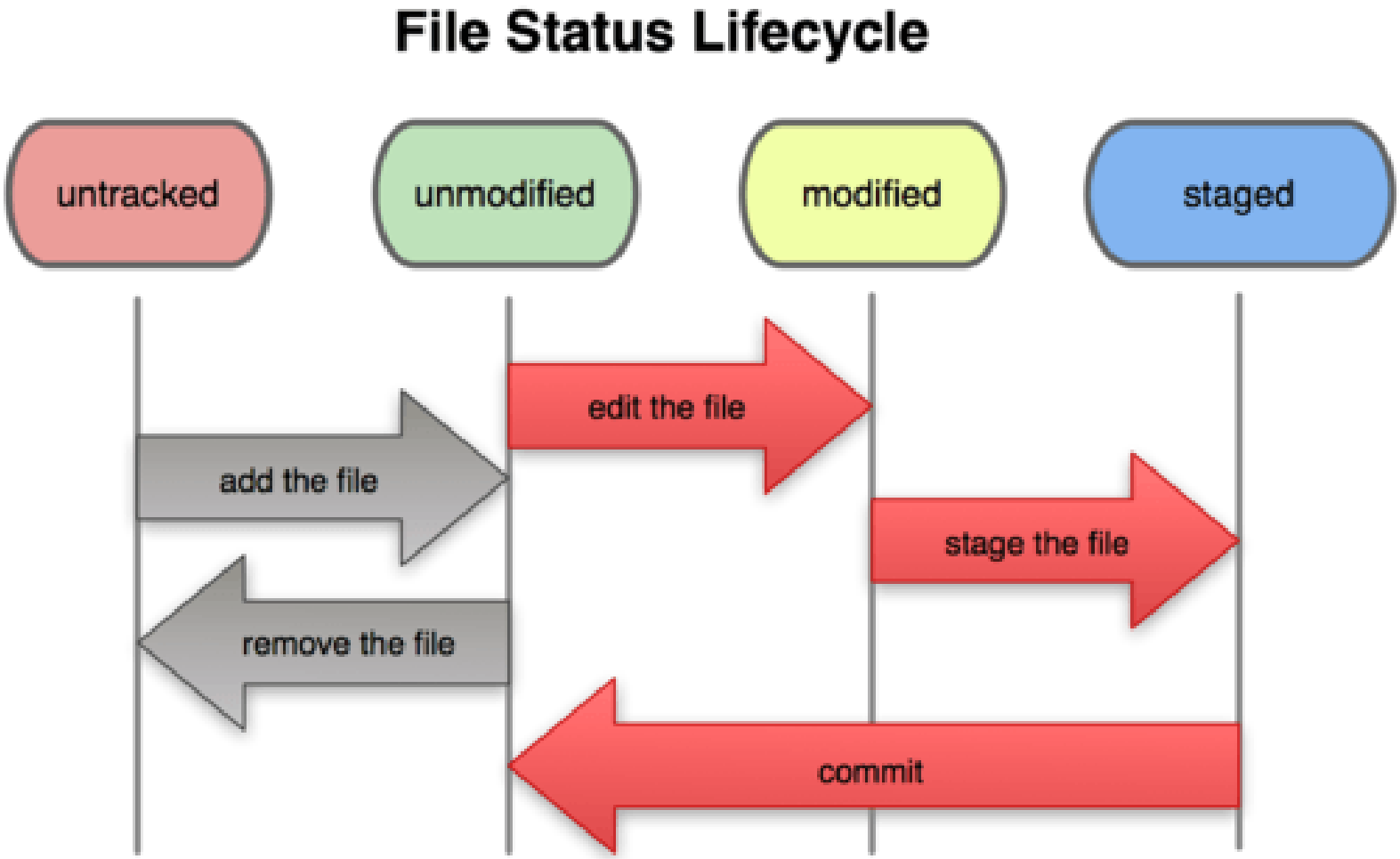
- Automatically stage every tracked file and commit

  ```
  git commit -a -m "..."
  ```

- Revert modifications: `git checkout -- <path>`
  - Alternative: `git restore <path>`

Department of Distributed and Dependable Systems D3S

File status lifecycle



Picture taken from http://git-scm.com/book/

Really basic actions

- Create repository in a specific directory
- Create some new files (e.g., hello world)
- Print current status of your repository and the working directory
- Stage all the new files
- Print current status
- Modify one of the files
- Print current status
  - Inspect differences from the previous invocation
- Commit all staged modifications
- Print current status

Department of Distributed and Dependable Systems D3S

Managing files

- Make the given file untracked

```
git rm <work dir path>
```

- Renaming file (directory)

```
git mv <old path> <new path>
```

Pick your changes

- Full interactive mode: `git add -i`
- Select patch hunks: `git add -p`
- Additional information with examples
  - https://git-scm.com/book/en/v2/Git-Tools-Interactive-Staging

Department of Distributed and Dependable Systems

Project history

- List all the commits

```
git log [-p] [-<N>] [--stat]
```

  - More options

```
[--pretty=oneline|short|full|fuller]
[--graph]
[--since=YYYY-MM-DD]
[--until=YYYY-MM-DD]
[--author=<name>]
```

- Show author name and revision for modifications

```
git blame <file path>
```

Task

- Try out file management commands (`rm`, `mv`)
- Play with the "`git log`" command
  - Explore different parameters (`-p`, `-<N>`, `--stat`, `--pretty`, `--graph`)
- Run the program "gitk" and try it
- Make some changes to a particular file and use interactive staging

Using remote repositories

- Clone a remote repository in the current local directory: `git clone <repo url>`
- Get recent changes in all branches from the remote repository: `git fetch origin`
- Get recent changes in the “master” branch and merge into your working copy: `git pull`
  - Announcements via *pull requests*
- Publish local changes in the remote repository: `git push origin master`

Department of Distributed and Dependable Systems D3S

Task

- Create new repository in your personal space at https://gitlab.mff.cuni.cz/
- Try out important commands for manipulation with remote repositories
  - clone, pull, push, fetch
- How to set (new) remote for your local Git repo

```
git remote add origin <repo url>
```

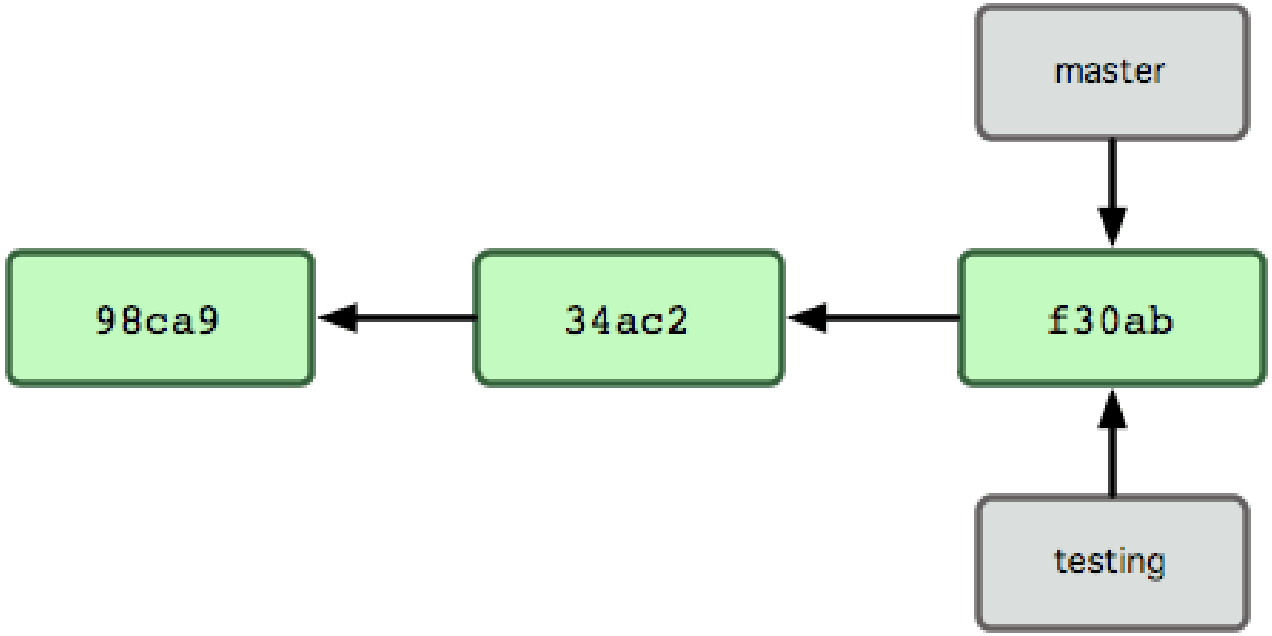
- Ask questions if something is not clear (!!)

Department of Distributed and Dependable Systems D3S

Branches in Git

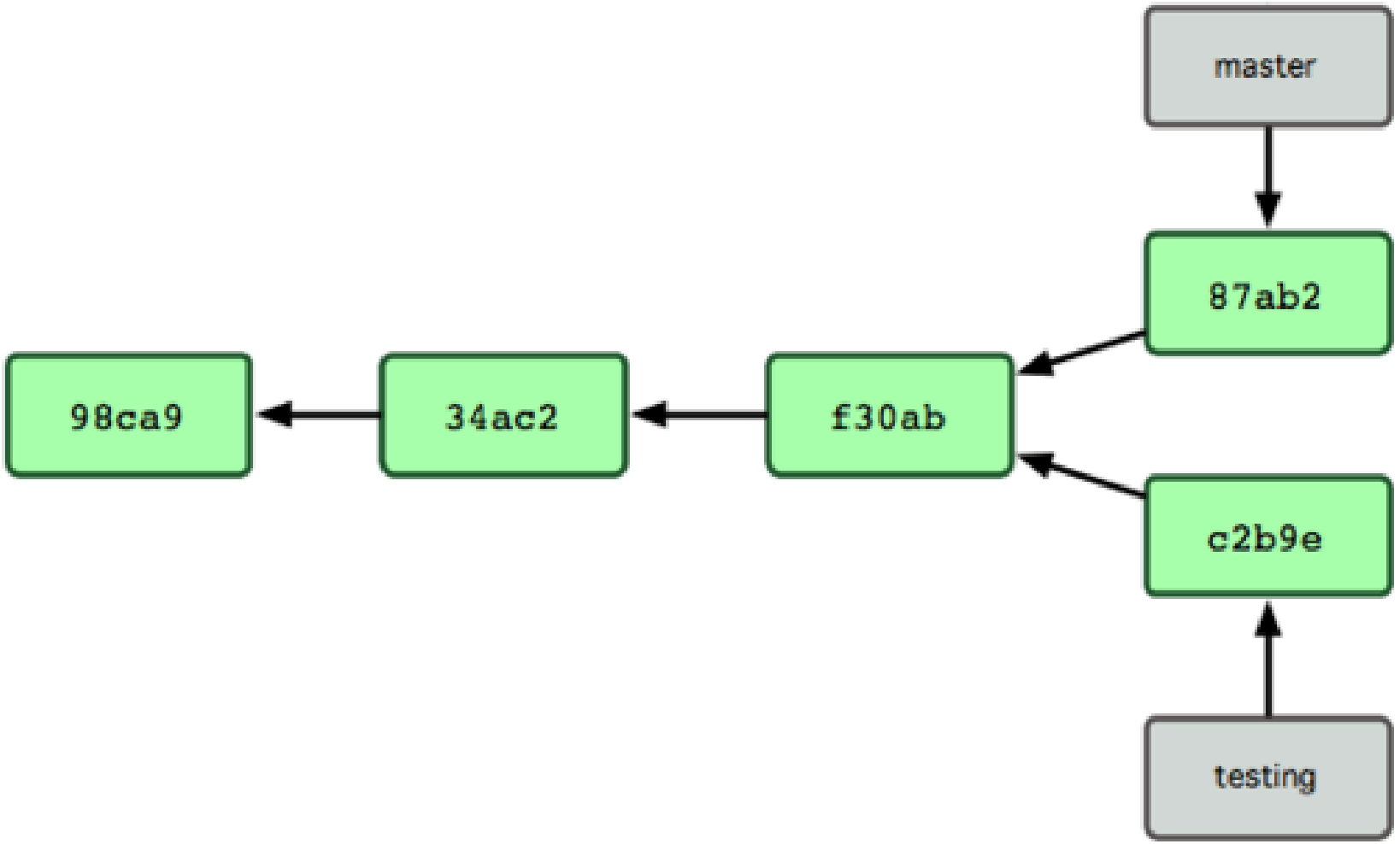
Branches in Git

- Branch: pointer to a node in the revision DAG
- Default branch: **`master`**
- Commit: branch pointer moves forward



Picture taken from http://git-scm.com/book/

What happens after concurrent modification



Picture taken from http://git-scm.com/book/

Branches in Git: commands

- Create new branch: `git branch <name>`
- Switch to given branch: `git checkout <name>`
- Shortcut: `git checkout -b <name>`
- Alternative for creating new branch and switching

```
git switch <branch name>
git switch -c <new branch>
```

- Merge branch into current working directory

```
git merge <branch name>
```

- Deleting unnecessary branch

```
git branch -d <branch name>
```

- List all branches: `git branch [-a]`
  - Current branch marked with `*`

Department of Distributed and Dependable Systems D3S

Comparing branches

- `git diff <branch 1>..<branch 2>`
  - Compare heads of the two branches
  - Note the characters '. .'
- `git diff <branch 1>...<branch 2>`
  - Print changes on the branch 2 (e.g., master) since the branch 1 (feature) was created from it
  - Note the characters '. . .'

Three-way merge

- Common ancestor
- Target branch
- Source branch

- Conflicts happen also with Git
  - Standard markers `<<<<<<<` `=======` `>>>>>>>`
  - Marking resolved files: `git add`

- Graphical merging tool: `git mergetool`

Task

- Create new branch B and switch to it
- Modify some files and commit them
- Switch back to the master branch
- Modify some files and then commit
- Merge your branch B into the master
- Delete the now unnecessary branch

- Try switching branches with uncommitted changes in the working copy
- Try graphical merging tool on some conflicts

Advanced features

- Stashing
- Undo
- Rebase

- “Squash”

Stashing

- Using stack of unfinished changes
  - `git stash [push]`
  - `git stash pop`
  - `git stash apply [<stash name>]`
  - `git stash list`

“Undoing” changes

- Symbolic names of versions
  - HEAD, HEAD~1, HEAD^2
- How to undo some changes
  - `git reset <commit>`
    - Moves the branch HEAD to a given commit
  - Several variants
    - `--soft`: undo commit (just in history of revisions)
    - `--mixed` (default): undo commit and changes in staging area
    - `--hard`: undo everything (commit, staging area, working dir)

“Undoing” changes – basic scenarios

- Drop modifications just in the working directory (before commit)
  - `git checkout -- <path>`
  - `git restore <path>`
- Remove the last commit (not yet pushed to remote) in your local repository and put the changes back to the working directory
  - `git reset –mixed HEAD~`
- Want to undo commits already pushed to the public shared repo ?
  - **NEVER EVER** drop commits in the public repo (branches) !!!
  - Make another commit that “restores” the original state (as if the particular commit never happened)
  - `git revert <commit ID>`
  - `git revert <oldest commit>..<latest commit>`

Task

- Try some approaches to “undoing” changes on your local repo and working directory
  - Use `git status` and `git diff` to observe the state of both working directory and staging area

Rebasing

- Command: `git rebase`
  - Replaying changes done in some branch onto another branch
  - Very powerful command but also tricky (be really careful !!)
  - Usage: `git rebase <source branch>` in target branch
- Modifying committed history
  - e.g., commit messages (`git commit --amend`)
- Interactive rebase
  - Command: `git rebase -i <after commit>`
  - Purpose: reordering commits, editing commit messages
  - https://git-scm.com/docs/git-rebase#_interactive_mode

Advanced features

- Ignoring certain files
  - List patterns in the file `.gitignore`
- Tagging: `git tag`
- Bare repository
  - No working copy

Merging: recommended practice

- Keep linear history
  - Rebase your branch on "main" just before merge
- Sometimes you want to "squash" multiple commits into one before merge
  - Why: eliminate work-in-progress commits from the final history ("cleaning")
  - Commands: `git reset --soft HEAD~N`, followed by `git commit`

Mercurial

- Basic principles: like Git
- Simpler learning curve
- Commands very similar
  - `init`, `clone`, `add`, `commit`, `merge`, `push`, `pull`

Work-flow models (cooperation)

Work-flow models (cooperation)

- Anything possible technically with DVCS
- “Network of trust” between developers

- Examples of possible organizations
  - Single “central” repository (branch)
  - Multiple release repositories (branches)
  - Many public repositories
  - Total anarchy

- Different workflow models
  - especially regarding branches

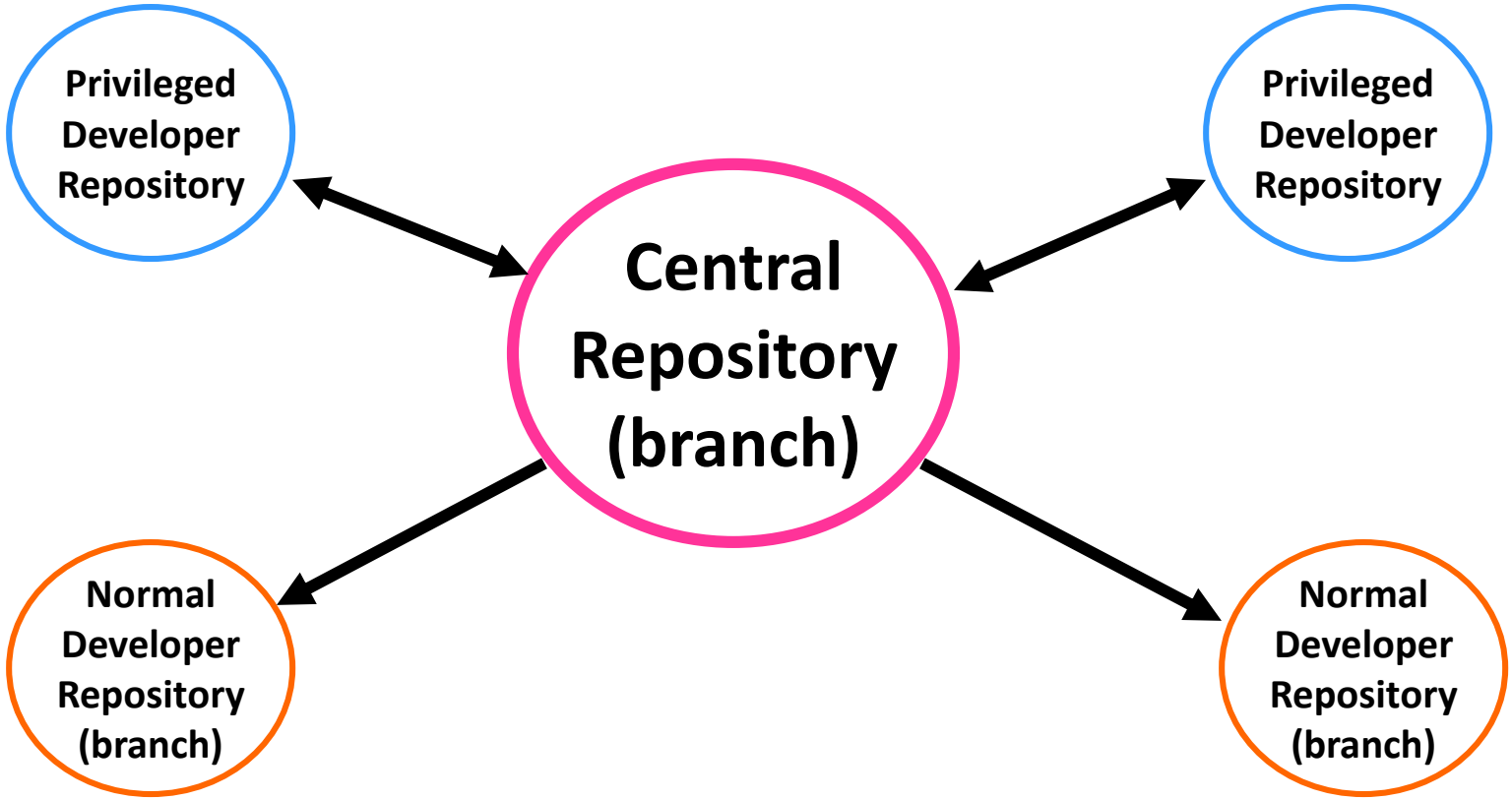
Department of Distributed and Dependable Systems D3S

Git workflow models

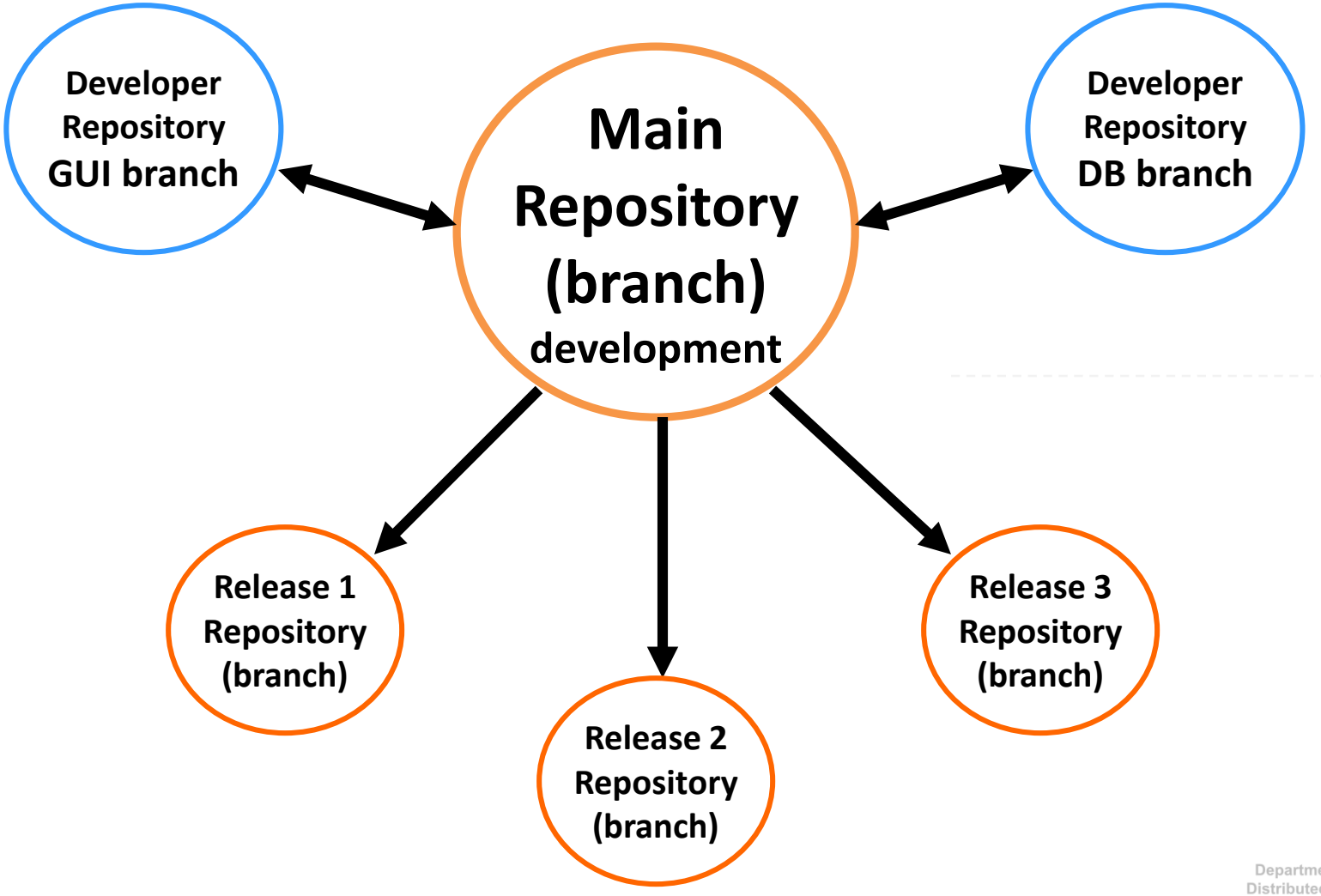
- Centralized (and comparison)
  - https://www.atlassian.com/git/tutorials/comparing-workflows
- Feature branch
  - https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow
- Trunk-based
  - https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development
- Forking
  - https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow

Department of Distributed and Dependable Systems D3S

Single “central” repository (branch)

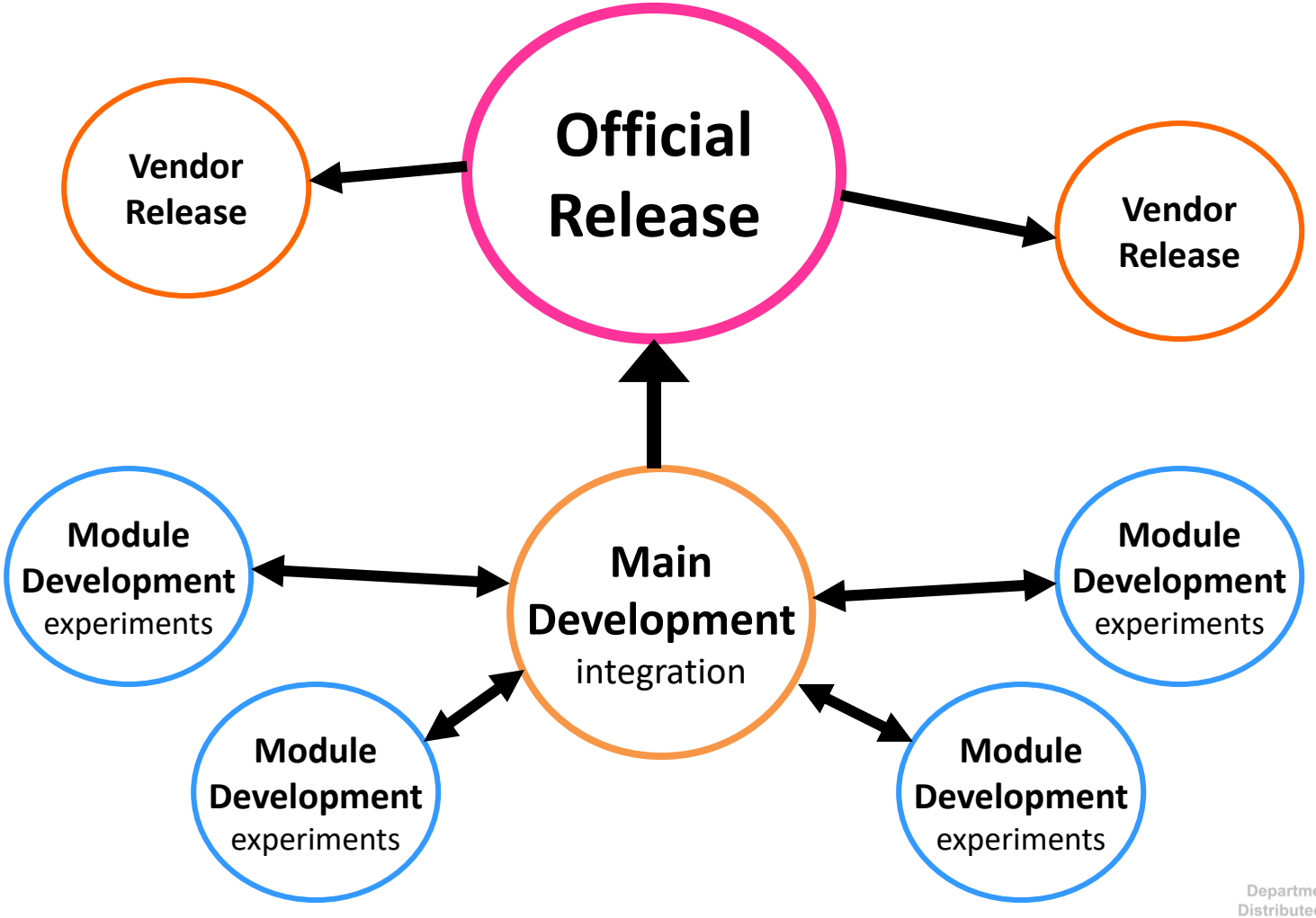


Multiple release repositories (branches)

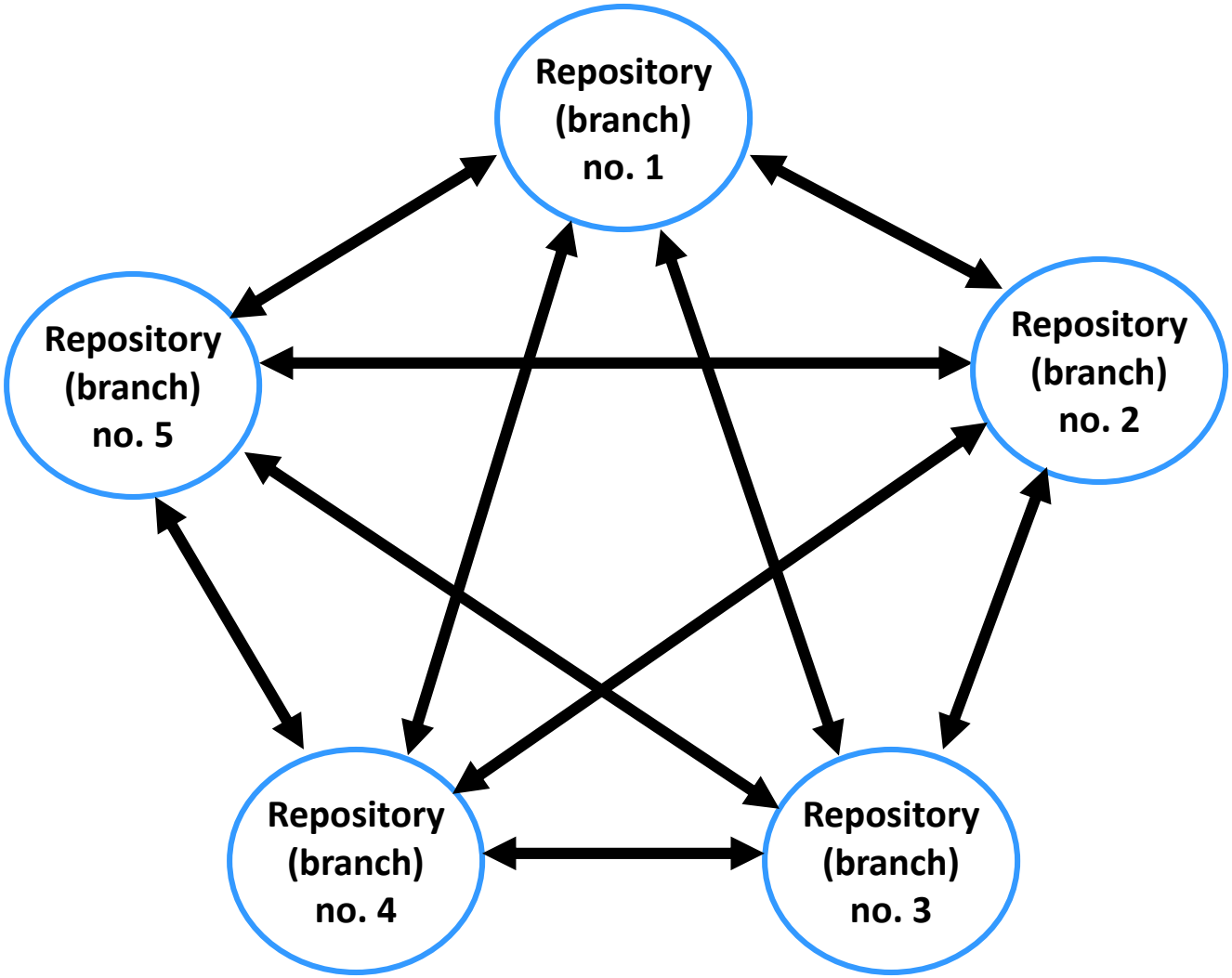


Many public repositories or branches

- Linux kernel



Total anarchy



Organization policy

- Organization
  - project, company, team
- Relevant aspects
  - Commit messages
  - Names of branches

Contributing to [open-source] projects

- Typical scenario
  - Project hosted on some public repository server
  - Write access to official repository is not possible
- Important concepts
  - Forking of the official repository
  - Publishing via pull requests

Contributing to [open-source] projects

- Official central repository (upstream)
  - https://github.com/projectname
- Fork on the same server
  - https://github.com/user/projectname
- Clone to local repository
  - From https://github.com/user/projectname to `$HOME/projectname`
- Synchronizing fork with official repository
  - `git fetch upstream`
  - `git merge upstream/master`
- Publishing changes to the upstream repository
  - Creating pull requests (processed later by maintainer)

Department of Distributed and Dependable Systems D3S

Links

- Git documentation
  - http://git-scm.com/doc
- Mercurial
  - http://www.mercurial-scm.org/, http://hgbook.red-bean.com/
- Repository servers
  - https://github.com/
  - https://bitbucket.org/
  - https://gitlab.com/
- Tools
  - Git for Windows (http://msysgit.github.io/), TortoiseGit (Win), SmartGit (http://www.syntevo.com/smartgit/)
  - TortoiseHg (Mercurial GUI, Windows)
  - SourceTree (https://www.sourcetreeapp.com/, Git and Mercurial)

Homework

- Assignment
  - ReCodEx: group associated with this course
  - Web: https://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly/
- Deadline
  - 5.3.2025