ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

03
Chapter

Workflows with Git

2I1AC3 : Génie logiciel et Patrons de conception

Régis Clouard, ENSICAEN - GREYC

"Git devient plus facile une fois que vous avez compris l'idée de base selon laquelle les branches sont des endofoncteurs homéomorphes mappant des sous-variétés d'un espace de Hilbert"

Isaac Wolkerstorfer

Outline

1
Collaborating
With Git
for organizing collaborative work (4)

Workflow

- A workflow defines a plan for organizing the collaborative work of multiple developers for the production of software
- Workflow based on Git
  - **Branching strategy**: everything goes through branches
- Multiple workflows
  1) GitHub Flow
  2) Git Flow
  3) GitLab Flow

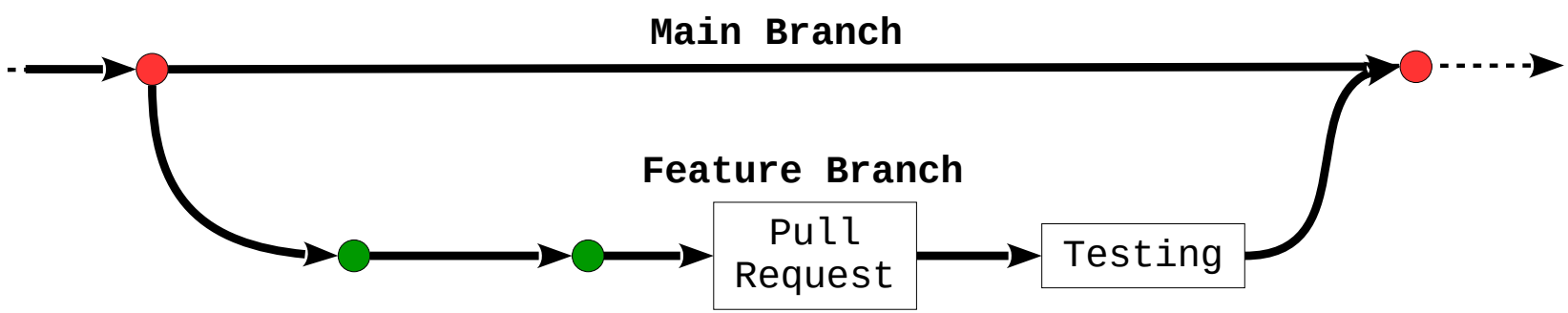
Outline

1
Collaborating With Git
for organizing collaborative work (4)

2
GitHub Flow

GitHub Flow

- The simplest: suitable for small projects like student project
- Branches: only 2 types of branch
  - **main**: permanent production branch. Contains production-ready code
  - ***x-feature***: temporary development branch, where x is the issue number and feature is the issue name (eg, 10-create-database)



Outline

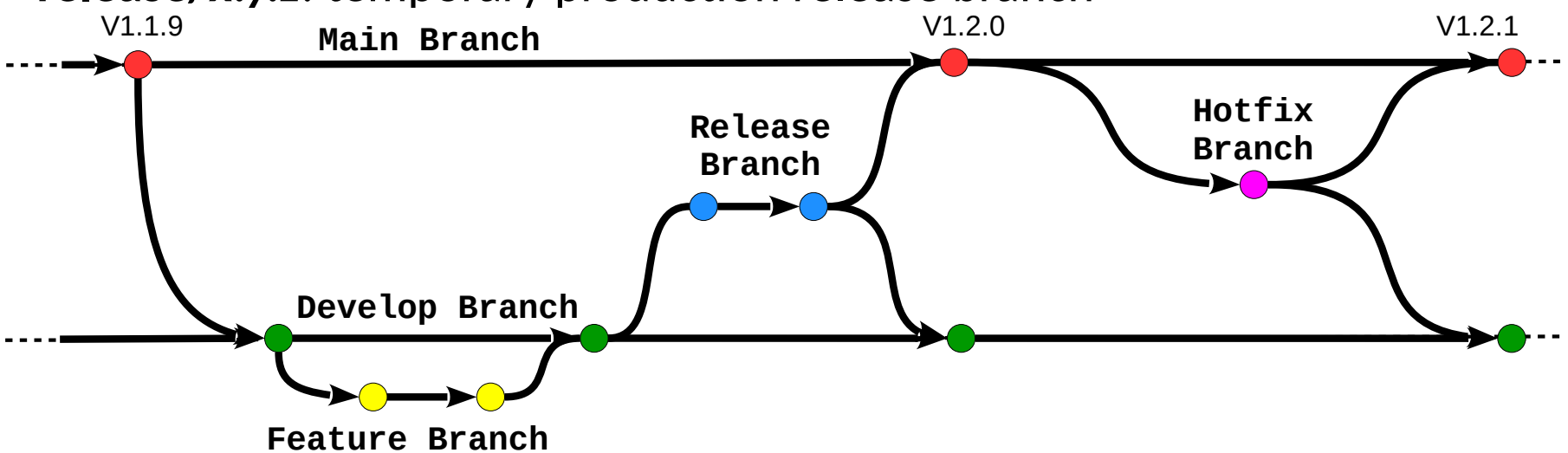
1
Collaborating With Git
for organizing collaborative work (4)

2
GitHub Flow

3
Git Flow

Git Workflow

- The most complete: suitable for large projects
- Branches:
  - **main**: permanent production branch (stable version of the software)
  - **develop**: permanent integration branch
  - **feature/x-ff**: temporary feature development branch (*x* is the issue number)
  - **hotfix/x-*issue***: temporary bug fix branch
  - **release/*x.y.z***: temporary production release branch



Outline

1
Collaborating With Git
for organizing collaborative work (4)

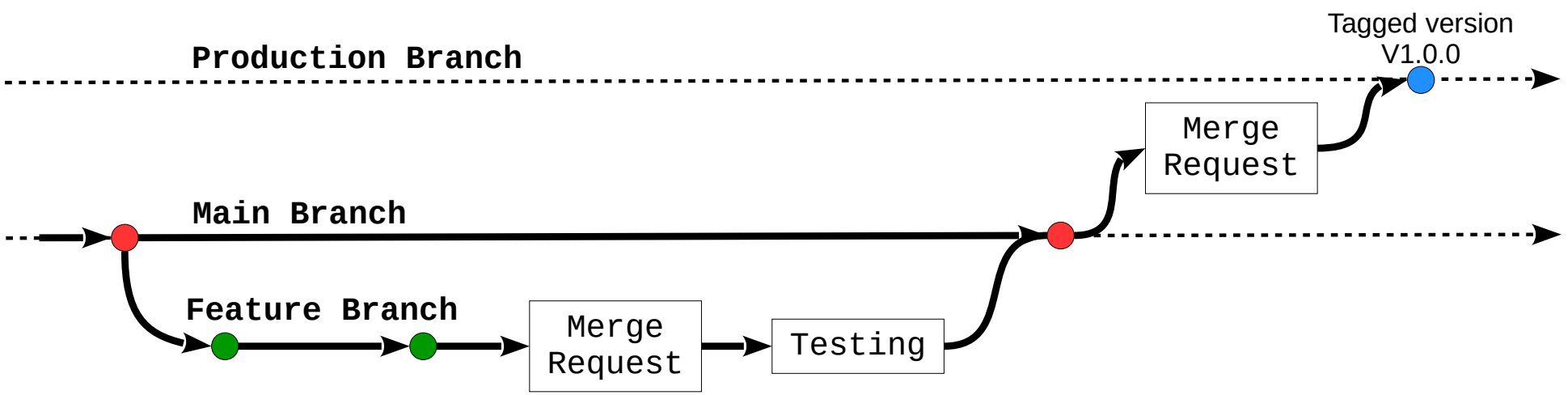
2
GitHub Flow

3
Git Flow

4
GitLab Flow

GitLab Flow

- The most efficient: includes CI/CD
- Branches
  - **main**: permanent development branch
  - **production**: permanent production branch
  - ***x-feature***: temporary development branch (x is the issue number)



GitLab Flow in Practice

Remote GitLab Repository

1) Create an issue
2) Create a Merge Request (MR + related branch) from the issue with the related button

9) Remove Draft from MR
10) Assign someone to review the MR
12) The reviewer accepts the MR, the issue is automatically closed by GitLab

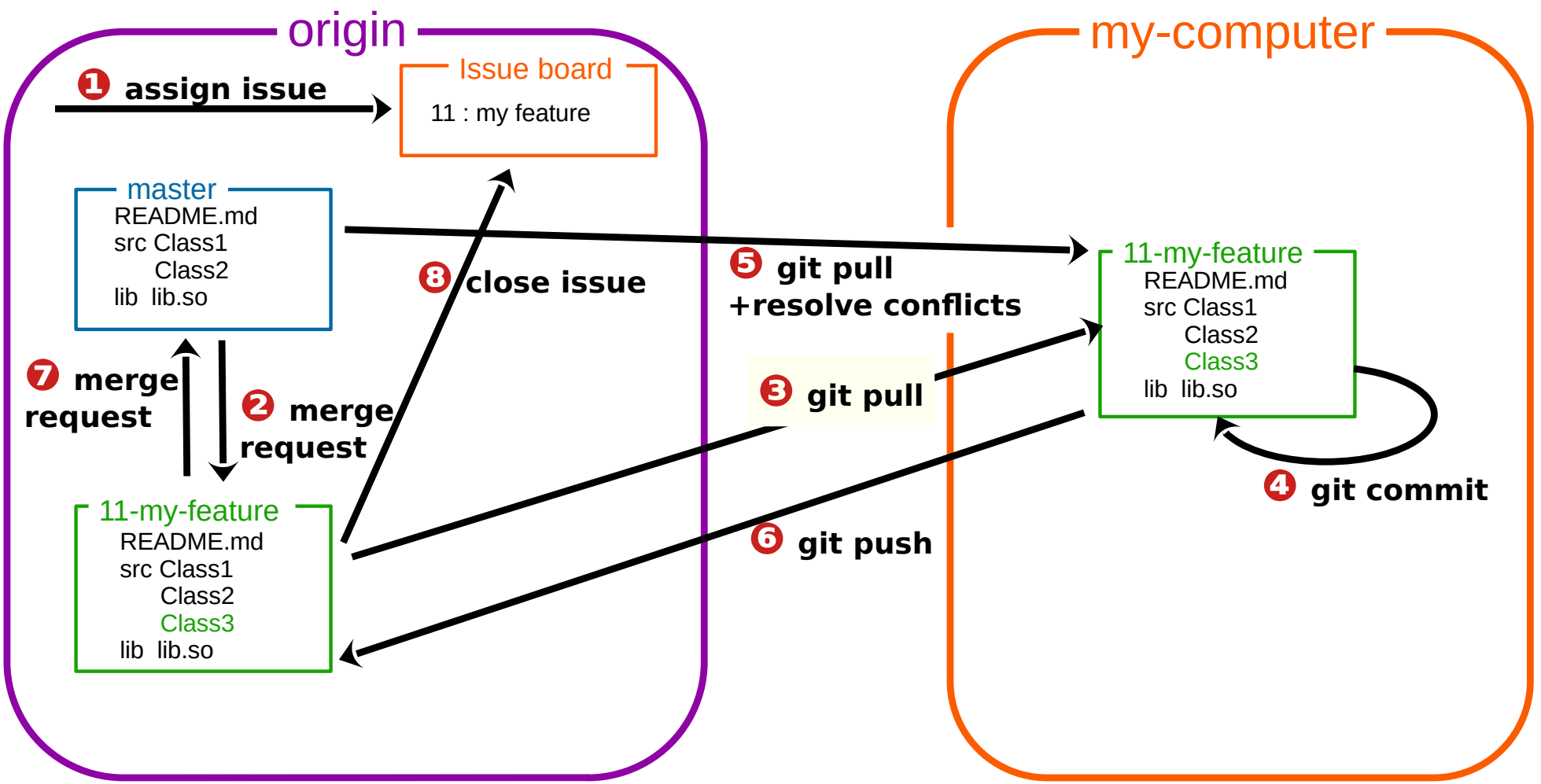
Local Repository

3) $ git fetch
4) $ git checkout -b *branch* origin/*branch*
5) Edit code, $ git commit
6) $ git pull origin master
7) Resolve conflicts
8) $ git push

11) Respond to review issues

13) Delete feature branch

The GitLab Flow



Outline

1
Collaborating With Git
for organizing collaborative work (4)

2
GitHub Flow

3
Git Flow

4
GitLab Flow

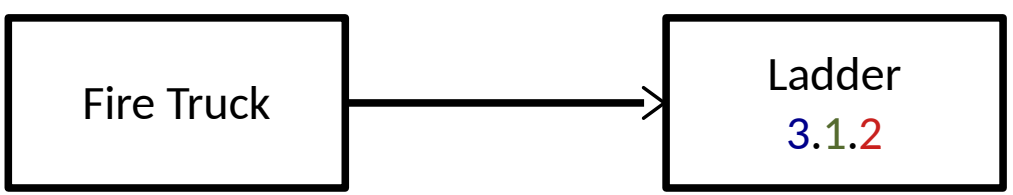
5
Best Practices

Best Practices

- **Protect permanent branches**
  - Restrict to maintainer
- **One branch = One feature/fix**
  - Develop using small steps to avoid integration problems and be able to go back to previous versions
- **Short-lived branches**
  - A branch must be short-lived (at most 1 lab session)
    - Old branch becomes difficult to merge since it diverges more and more from the main branch (and the developer cuts himself off from other teammates)
  - Create a new branch for each new contribution
  - Delete branch after merged
- **Small commit**
  - Develop using small steps to avoid integration problems and be able to go back to previous versions
  - Keep atomic commit (one thing at a time)

Best Practices: Semantic Versioning

- Version number: **MAJOR**.**MINOR**.**PATCH**
- Increment the:
  - MAJOR version when you make **incompatible changes**
  - MINOR version when you **add functionality** in a backward compatible manner
  - PATCH version when you make backward compatible **bug fixes**
- Benefit



  - You can safely specify that the version number of the Ladder ≥ 3.1.2 but < 4.0.0

Best Practices: Conventional Commits

- Format:

```
<type>[(scope)][!]: <subject>

[optional body]

[optional footer]
```

- Example

```
feat(browser): add onUrlChange event

Add new event to browser:
- forward popstate event if available
- forward hashchange event if popstate not available
- do polling when neither popstate nor hashchange available

Closes #392
```

Best Practices: Conventional Commits

- **Types**: (link to semantic versioning)
  - **feat**: add a new feature (MINOR)
  - **fix**: a bug fix (PATCH)
  - **refactor**: a code change that neither fixes a bug nor adds a feature (PATCH)
  - **test**: adding missing tests or correcting existing tests (PATCH)
- **Breaking changes**
  - !: breaking change (MAJOR)

Best Practices: Conventional Commits

- **Subject**:
  - Use the imperative, present tense: "change" not "changed" nor "changes"
    - Think of: This commit will… or This commit should…
  - Do not capitalize the first letter
  - Do not end the description with a period (.)
  - In case of breaking changes add an ! before the :