Git

università di ferrara
DA SEICENTO ANNI GUARDIAMO AVANTI.

Topics covered

- Introduction to Git
- Git workflows
- Git key concepts
- Hands on session
- Branching models

Introduction to Git

Version control systems

- The source files of a project changes over time (addition of functionality, error correction)
- Version control systems allow you to store and, if necessary, recover all significant versions of the sources in a repository
  - Comparison with previous versions
  - Reverting to a previous version (rollback)

Version control systems

There are 3 types of version control systems:

- Local, repository on the local machine
  - RCS
- Centralised, repository on a remote machine
  - CVS
  - Subversion (svn)
- Distributed, repository on several machines (some may act as servers)
  - **Git**
  - Mercurial

What is Git?

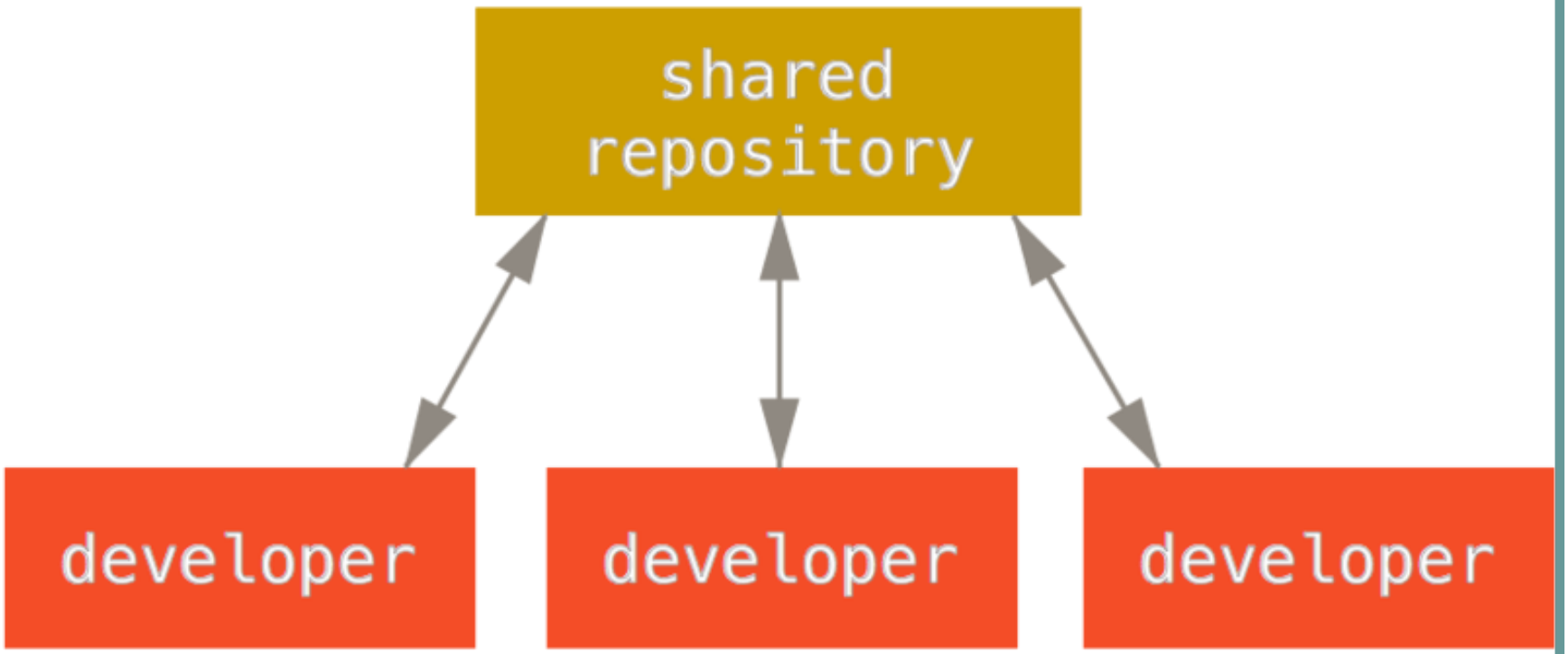
- **Distributed (Decentralized) version control system**
- Users keep entire code and history on their local machines
  - Users can make any changes without internet access
  - (Except pushing and pulling changes from a remote server)
- Started in 2005
- Created by Linus Torvald (the creator of the Linux kernel) to aid in Linux kernel development

Who uses Git


Companies & Projects Using Git
Google
facebook
Microsoft
twitter
LinkedIn
NETFLIX
PostgreSQL
android
RAILS
Qt
GNOME
eclipse

Git workflows

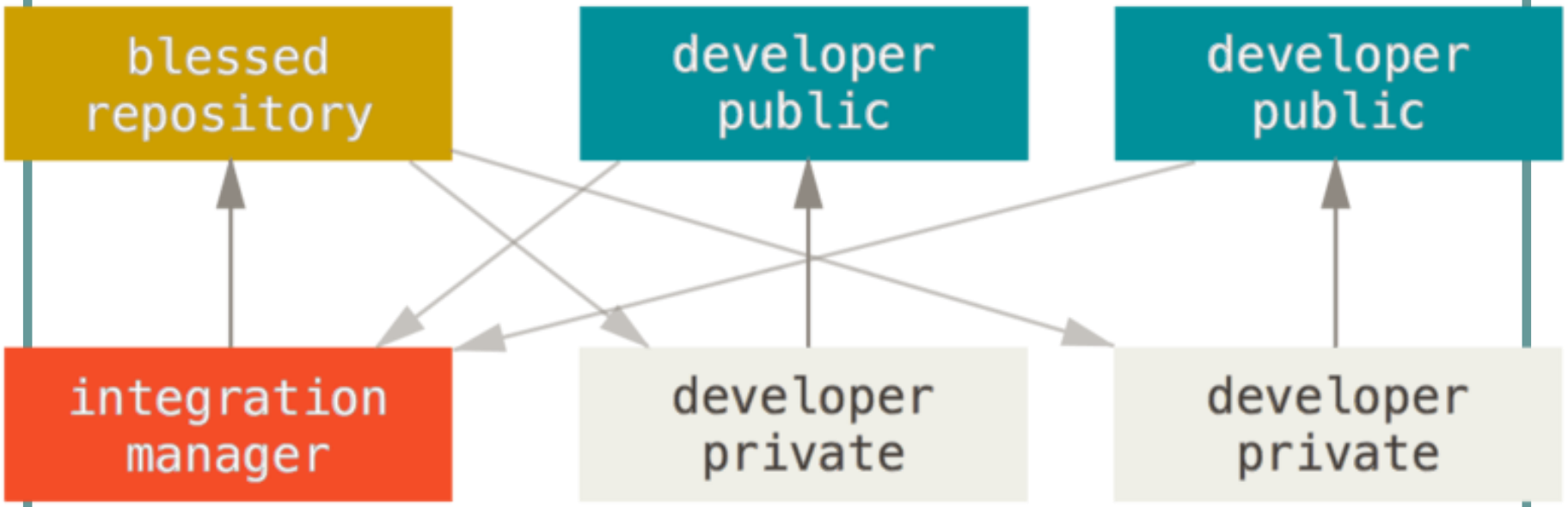
Centralised workflow



Centralised workflow

- One central repository can accept code, and everyone synchronizes their work to it. A number of developers are nodes and synchronize to that one place.

Integration-Manager Workflow

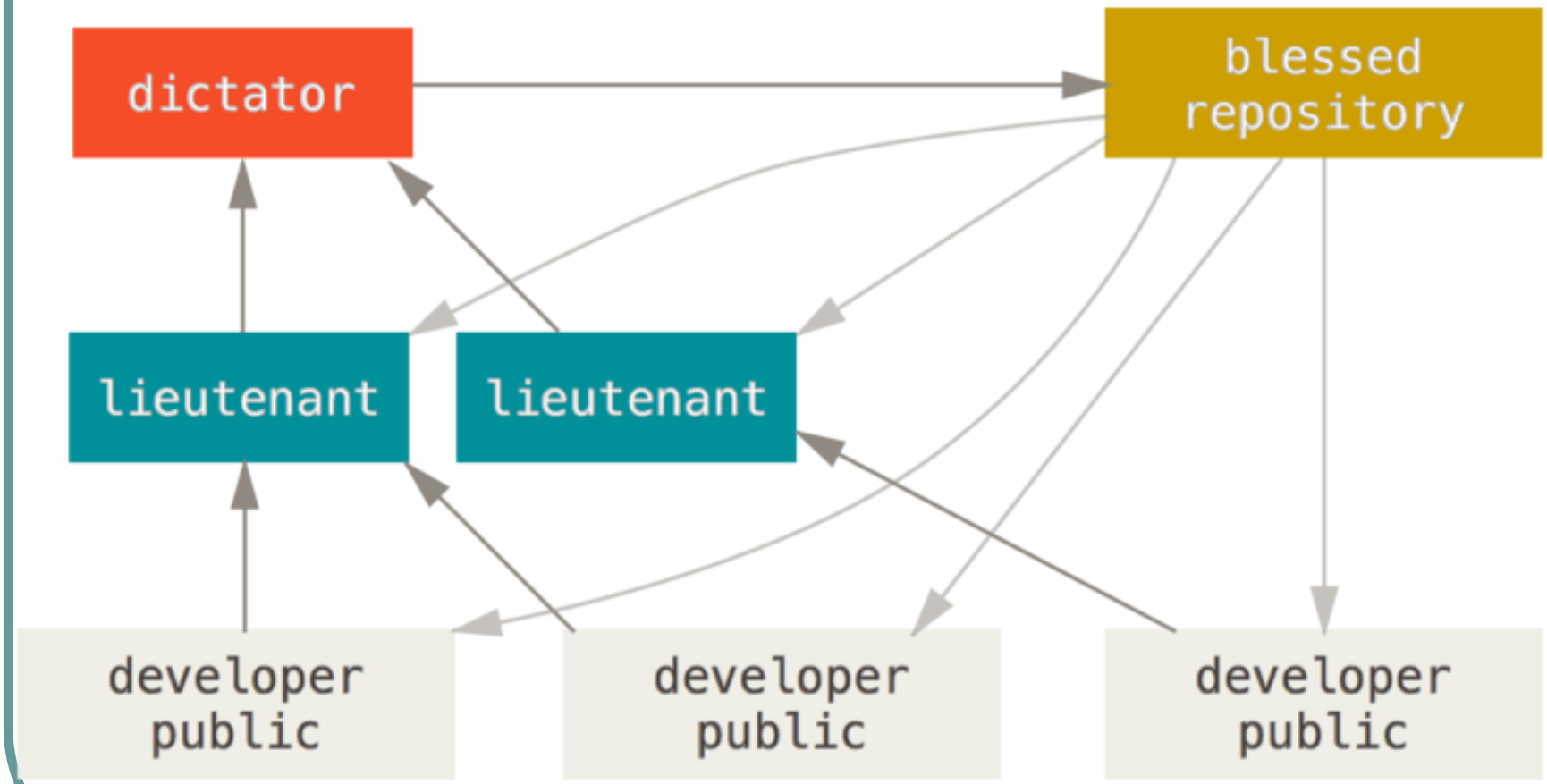


Integration-Manager Workflow

Process:

1. The project maintainer pushes to a public repository (blessed repository).
2. A contributor clones that repository and makes changes.
3. The contributor pushes to his own public copy.
4. The contributor sends the maintainer an email asking him to pull changes.
5. The maintainer adds the contributor’s repo as a remote and merges locally.
6. The maintainer pushes merged changes to the main repository.

Dictator and Lieutenants Workflow



Dictator and Lieutenants Workflow

Process:

1. Regular developers work on their topic branch and rebase their work on top of master. The master branch is that of the reference directory to which the dictator pushes.
2. Lieutenants merge the developers' topic branches into their master branches
3. The dictator merges the lieutenants‘ master branches into his master branch
4. The dictator pushes his master to the blessed repository so the other developers can rebase on it.

Remote repositories

- "In-House"
- Hosted
  - Github
  - BitBucket
  - GitLab

Git key concept

Snapshots

- The way Git keeps track of your code history
- Essentially records what all your files look like at a given point in time
- You decide when to take a snapshot, and of what files
  - The **staging area** or **index** is a file that stores information about what will go into the snapshot
- Have the ability to go back to visit any snapshot
  - Your snapshots from later on will stay around, too

Commit

- The act of creating a snapshot
- Can be a noun or verb
  - “I commited code”
  - “I just made a new commit”
- A project is made up of a set of commits
- Commits contain three pieces of information:
  - Information about how the files changed from the previous commit
  - A reference to the commit that came before it (called the parent commit)
  - A hash code name that identifies the commit

Repository

- A collection of all the files and the history of those files
  - Consists of all your commits
  - Place where all your work is stored
- Can live on a local machine or on a remote server (GitHub, Bitbucket, GitLab)
- The act of copying a repository from a remote server is called **cloning**
  - Cloning from a remote server allows teams to work together
- The process of downloading commits that don't exist on your machine from a remote repository is called **pulling** changes
- The process of adding your local changes to the remote repository is called **pushing** changes

Branches

- All commits in git live on some branch
- But there can be many, many branches
- The main branch in a project is called the master branch

Branches

What is a Git project?

- A set of commits linked together that live on some branch, contained in a repository.

Hands on session

Install Git

- Linux (Debian)
  - Command: sudo apt-get install git
- Linux (Fedora)
  - Command: sudo yum install git
- Mac
  - http://git-scm.com/download/mac
- Windows
  - http://git-scm.com/download/win

Configuration

- Identity

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

- Creating/Initializing repository

```
git init
```

- Cloning a remote repository

```
git clone <url_repository>
```

Common commands

- Add new file into the index (staging area)

```
git add README.txt
```

- Remove file from the index (staging area)

```
git rm file.c
```

- Commit changes

```
git commit –m "First commit"
```

- Updates files in the working tree to match a given commit

```
git checkout <id_commit>
```

Common commands

- Show log (commit history)

```
git log
```

- Show commit changes

```
git show <commit_hash>
```

- Show diffs between the index and the previous commit (HEAD)

```
git diff
```

Undoing things

- Unmodify modified file in the index

```
git checkout – file.c
```

- Revert a commit

```
git revert <commit_hash>
```

Commands for remote repositories

- Push to remote repository

  `git push origin <branch_name>`

- Fetch from remote repository

  `git fetch`

- Merge fetched content

  `git merge origin/<branch_name>`

- Fetch & Merge

  `git pull`

Branch management

- Create new branch

  `git branch <branch_name>`

- Switch branch

  `git checkout <branch_name>`

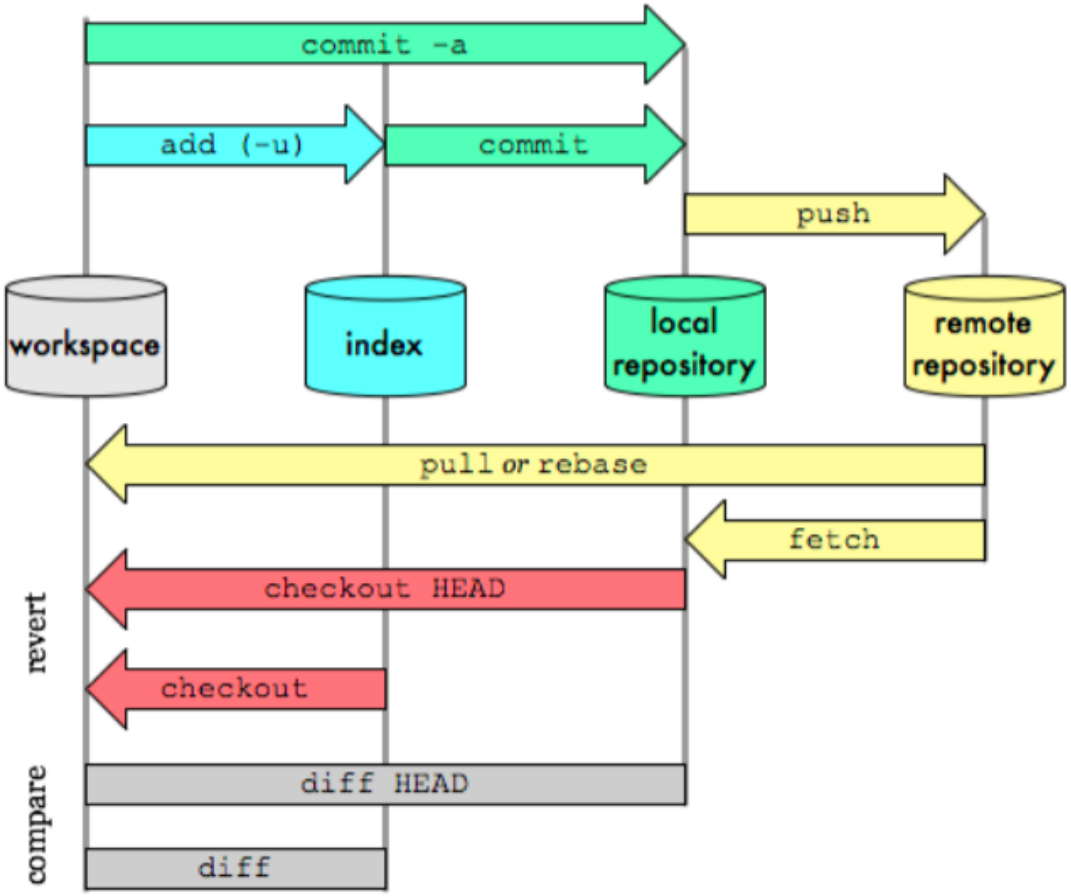
- Delete branch

  `git branch -d <branch_name>`

- Show all branches

  `git branch`

Git fundamental commands



Additional info

- Git official site
  - https://git-scm.com/
- Git 15 minutes tutorial
  - https://try.github.io/levels/1/challenges/1
- Git cheat sheet
  - https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf

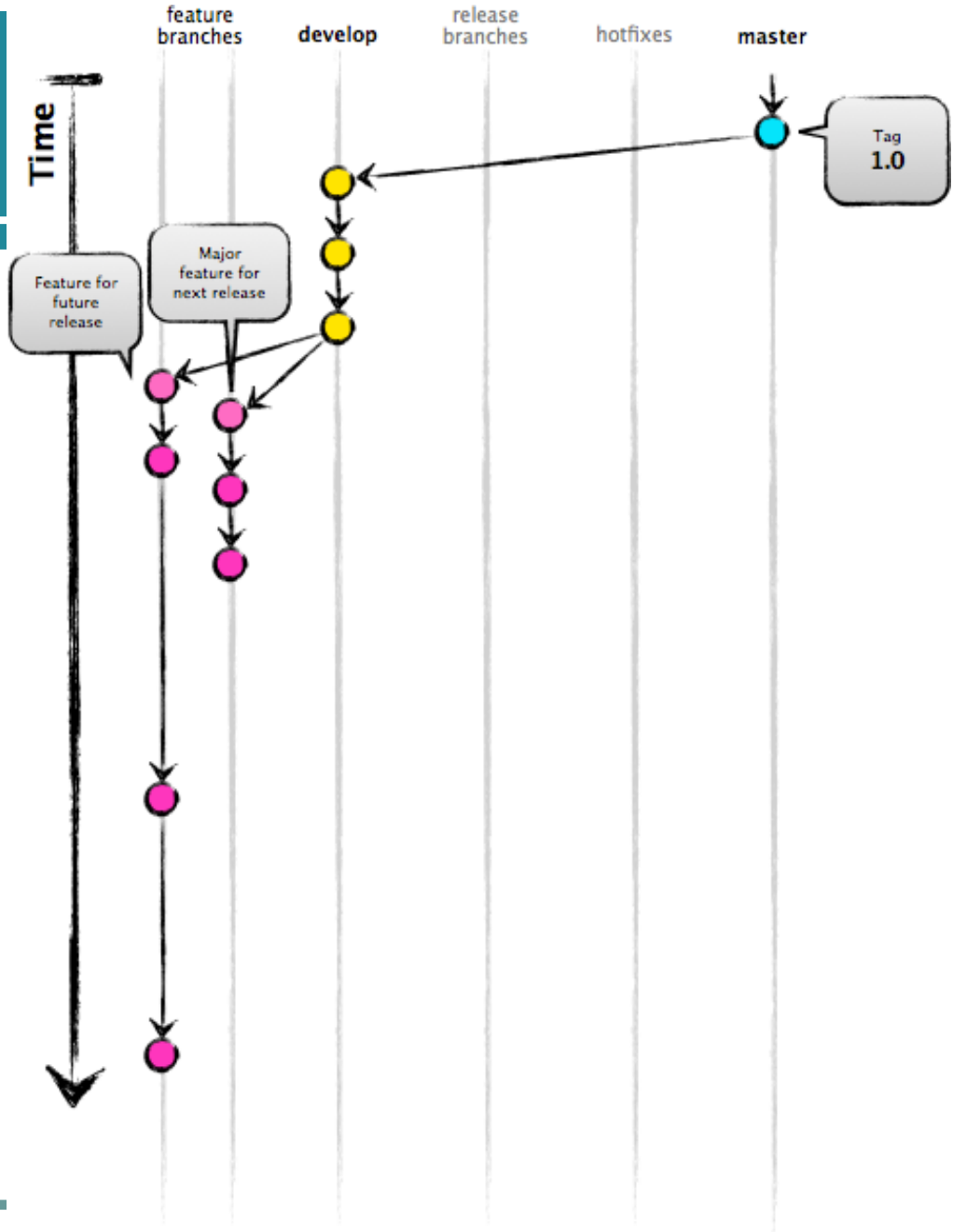
Branching models

GitFlow

- GitFlow is a branching model for Git created by Vincent Driessen
- More info:
  - https://datasift.github.io/gitflow/IntroducingGitFlow.html
  - http://nvie.com/posts/a-successful-git-branching-model/
- (Author of the pictures: Vincent Driessen)
  - Original blog post: http://nvie.com/

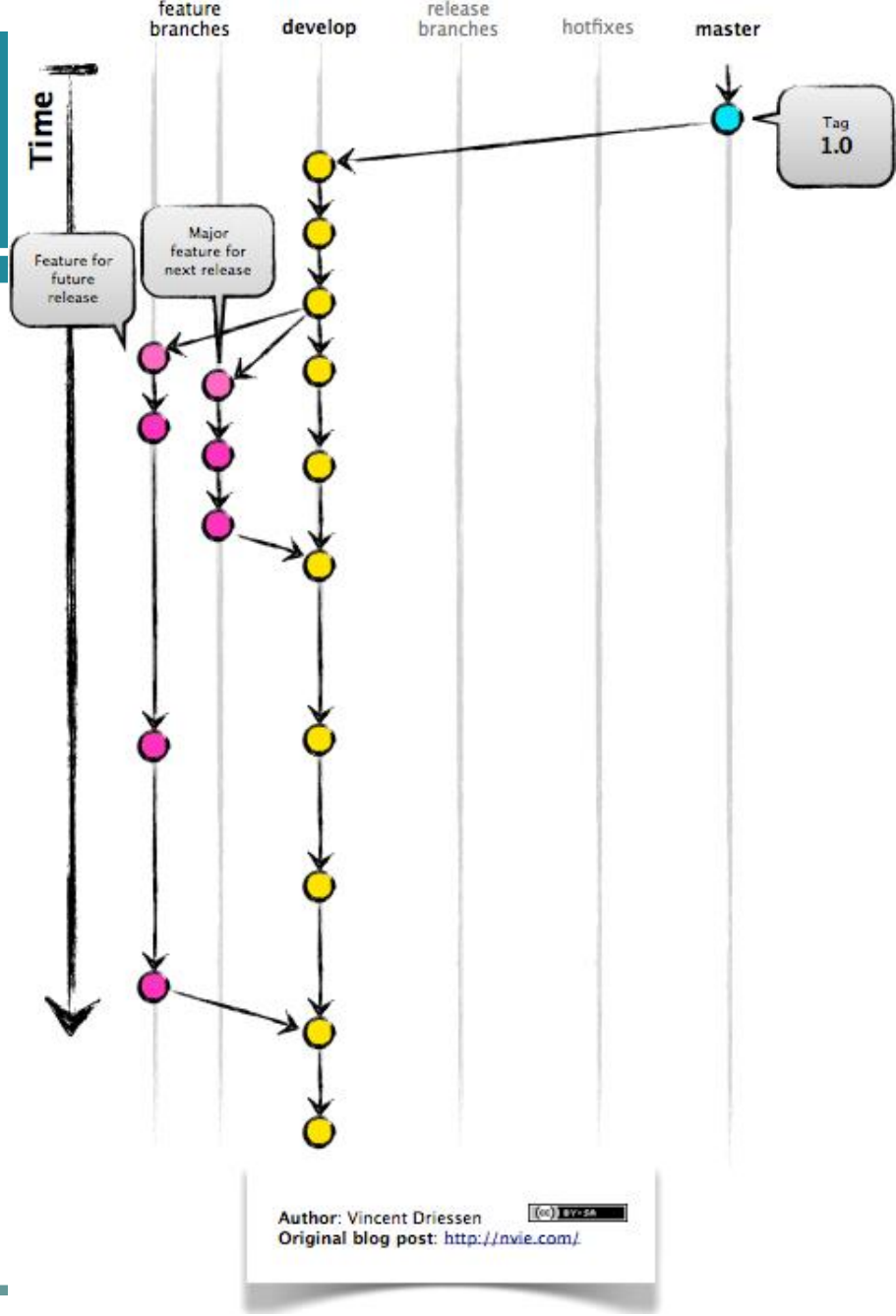
GitFlow

1. New development (new features, non-emergency bug fixes) are built in **feature branches**



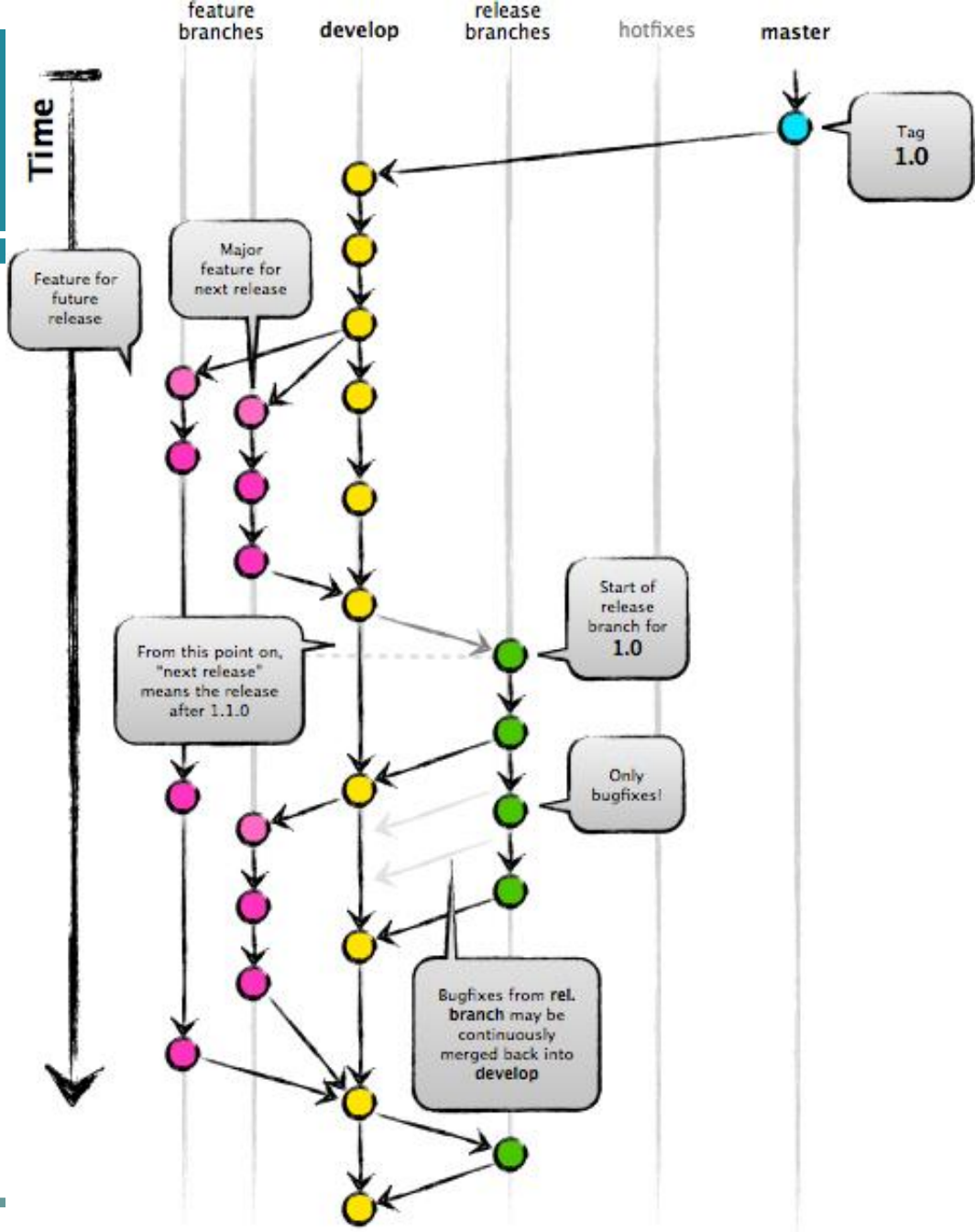
GitFlow

2. Feature branches are branched off of the **develop branch**, and finished features and fixes are merged back into the **develop branch** when they're ready for release



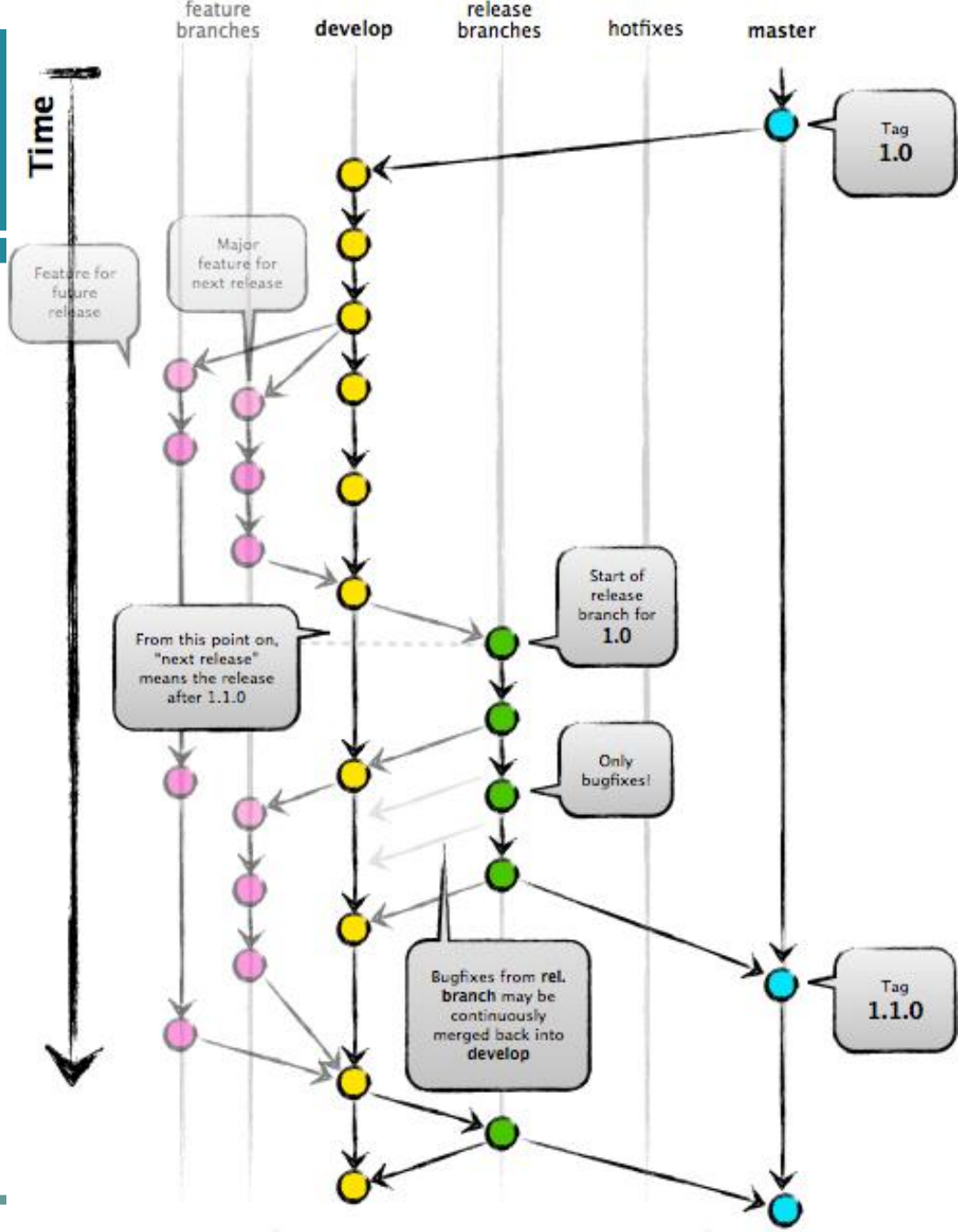
GitFlow

3. When it is time to make a release, a **release branch** is created off of **develop**
4. Test of the release branch



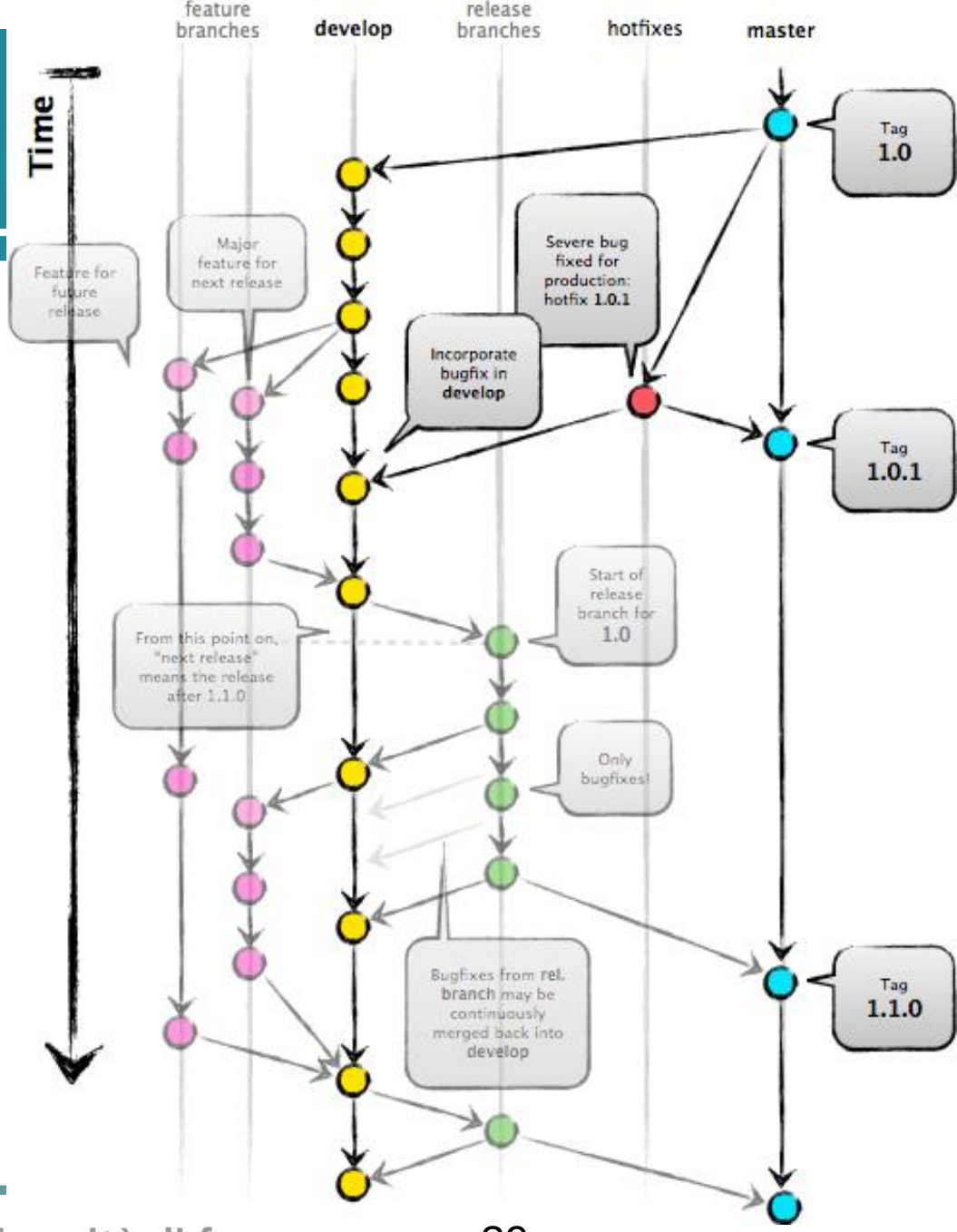
GitFlow

5. When the release is finished, the **release branch** is merged into **master and** into **develop**

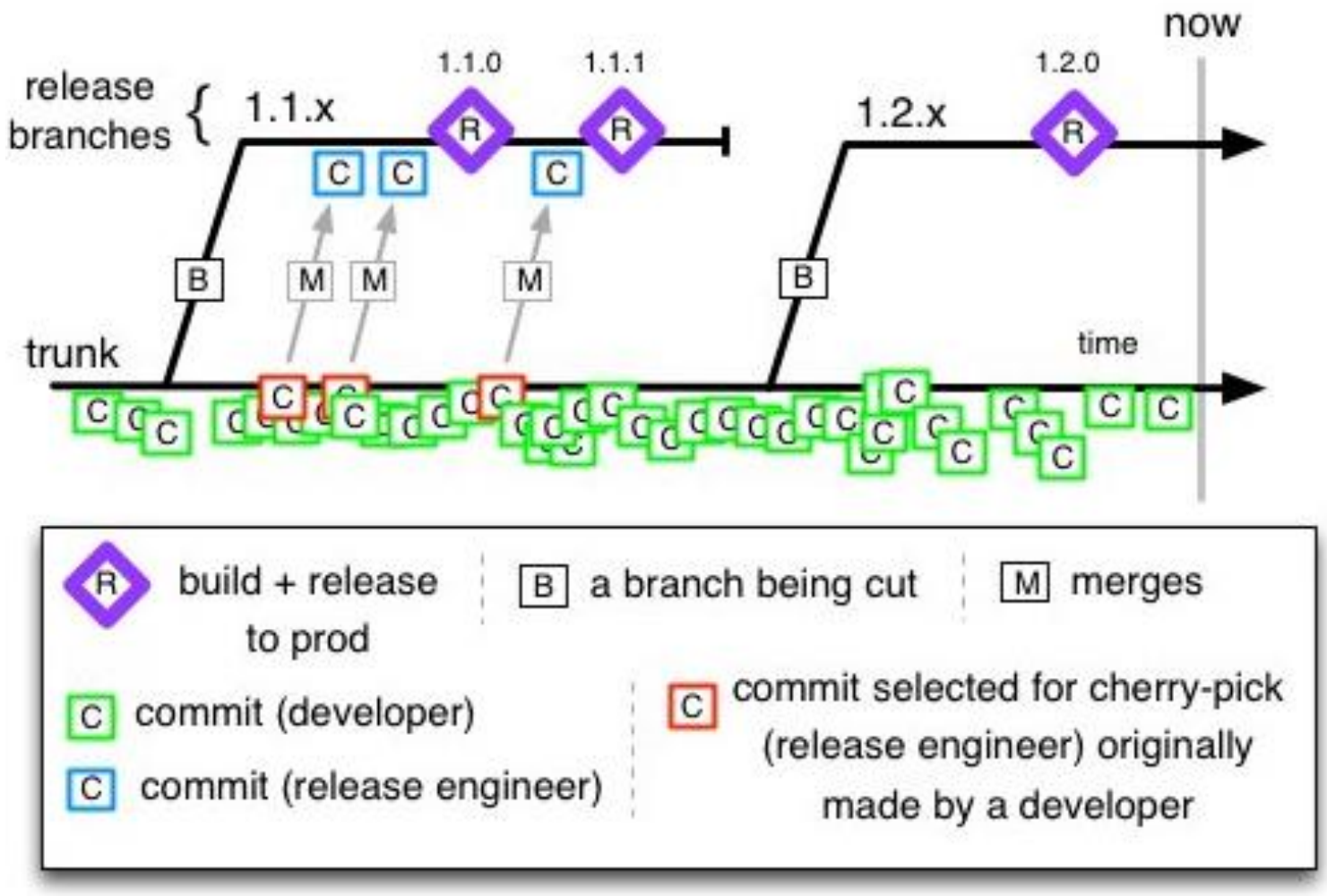


GitFlow

6. The **master branch** tracks released code only. The only commits to **master** are merges from **release branches** and **hotfix branches**.
   - **Hotfix branches** are used to create emergency fixes
7. **Hotfix branches** are branched directly from a tagged release in the **master branch**, and when finished are merged back into both **master** and **develop**.



Trunk Based Development



Trunk Based Development

- We have two main branches: **trunk** and **release**
- Developers commit to a single trunk more or less exclusively
- Release engineers (or build-cop) create branches, and cherry-pick to branches more or less exclusively
  - Only if a defect **cannot** be reproduced on trunk, is permission given to fix it on the release branch, and cherry-pick back to trunk.
- Trunk Based Development means **regular developers** don't commit to a release branch.
- Trunk Based Development means you're going to delete 'old' release branches, without merging them back to trunk
- More info: http://paulhammant.com/2013/04/05/what-is-trunk-based-development/