dataport

anderScore
trust in competence

Tag 2: Vertiefung Git-Workflow, CI/CD & GitLab CI

18.06.2024, Daniel Krämer & Malte Fischer

Agenda

- **Tag 1 – Einführung in Git und GitLab, Git-Workflow im Team**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
  - Git-Workflow im Team
- **Tag 2 – Vertiefung Git-Workflow, CI/CD & GitLab CI**
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - GitLab-Runner
  - Einführung in GitLab CI/CD & gitlab.yml
- **Tag 3 – GitOps, Docker in der Entwicklung und Deployment-Strategien**
  - GitOps Grundlagen
  - Lokale Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

Agenda

- **Tag 1 – Einführung in Git und GitLab, Git-Workflow im Team**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
  - Git-Workflow im Team
- **Tag 2 – Vertiefung Git-Workflow, CI/CD & GitLab CI**
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - GitLab-Runner
  - Einführung in GitLab CI/CD & gitlab.yml
- **Tag 3 – GitOps, Docker in der Entwicklung und Deployment-Strategien**
  - GitOps Grundlagen
  - Lokale Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

Git

Gitflow-Workflow

Gitflow-Workflow

anderScore trust in competence

Inhalt

- Was ist der Gitflow-Workflow?
- Aufbau des Gitflow-Workflows
  - Branches und deren Verwendung
- Arbeiten mit Gitflow-Workflow
- Use-Cases und Fazit

Gitflow-Workflow

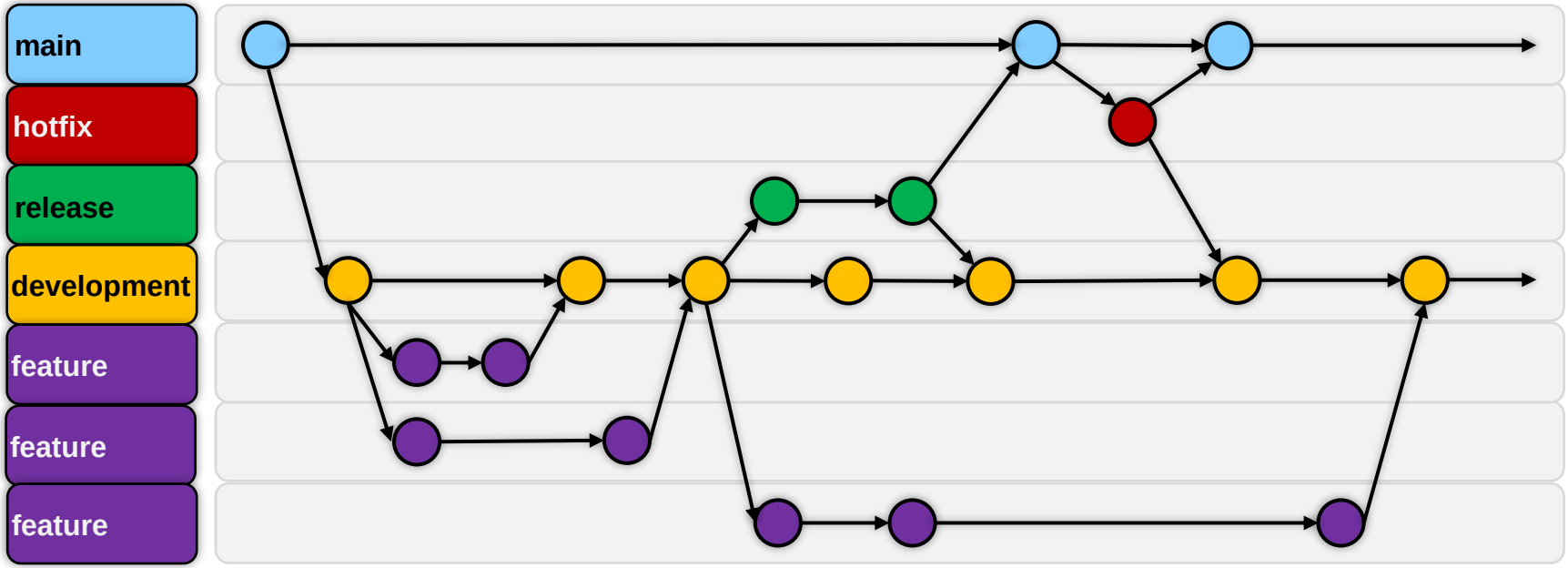
anderScore trust in competence

Was ist der Gitflow-Workflow?

- Der Gitflow-Workflow ist ein Workflow mit Feature Branches und mehreren primären Branches
- Wurde 2010 von Vincent Dreissen als Post auf nvie veröffentlicht, um seine Arbeitsweise mit Git zu teilen
- Wurde von vielen Teams übernommen und erfreute sich großer Beliebtheit
- Wird heutzutage teilweise als veraltet angesehen
- Trotzdem immer noch große Verbreitung und Verwendung

Gitflow-Workflow – Aufbau

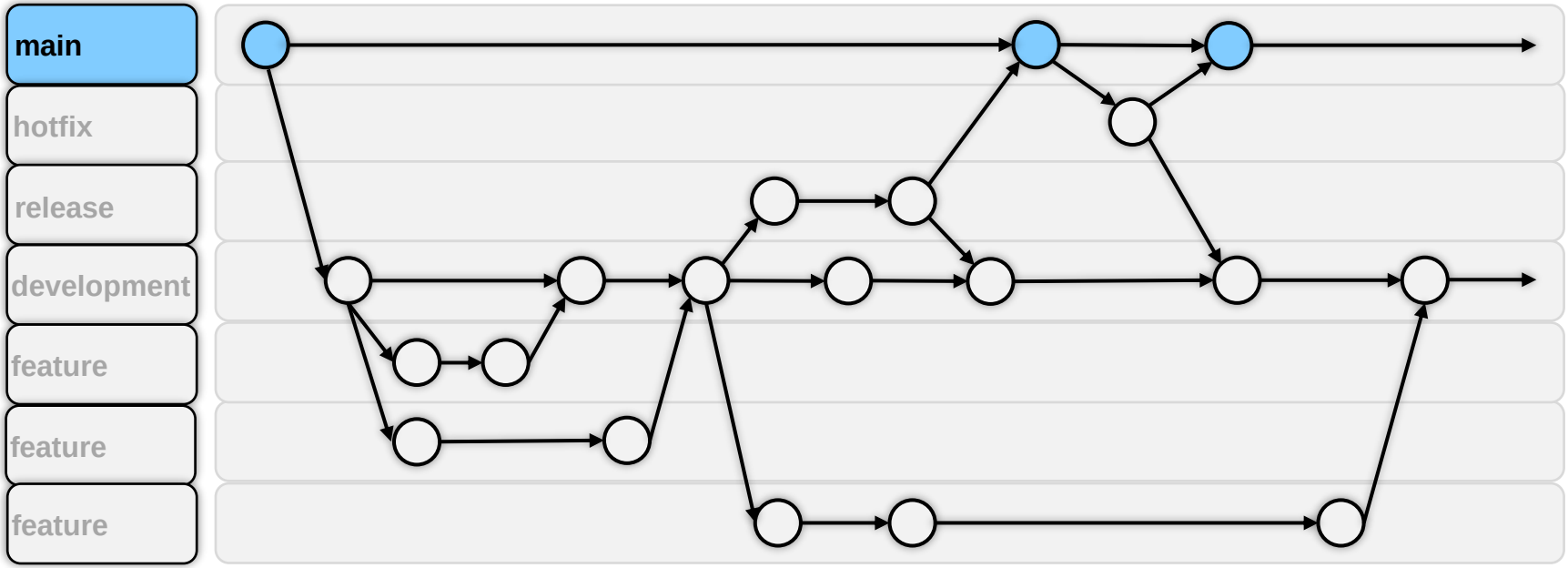
- Besteht aus mehreren Branches mit zugeteilten Rollen
- main und development werden zu Beginn des Projektes erstellt und existieren dauerhaft
- Release und Feature Branches erlauben getrenntes Arbeiten und ermöglichen isoliertes Experimentieren



Gitflow-Workflow – Aufbau

Main Branch

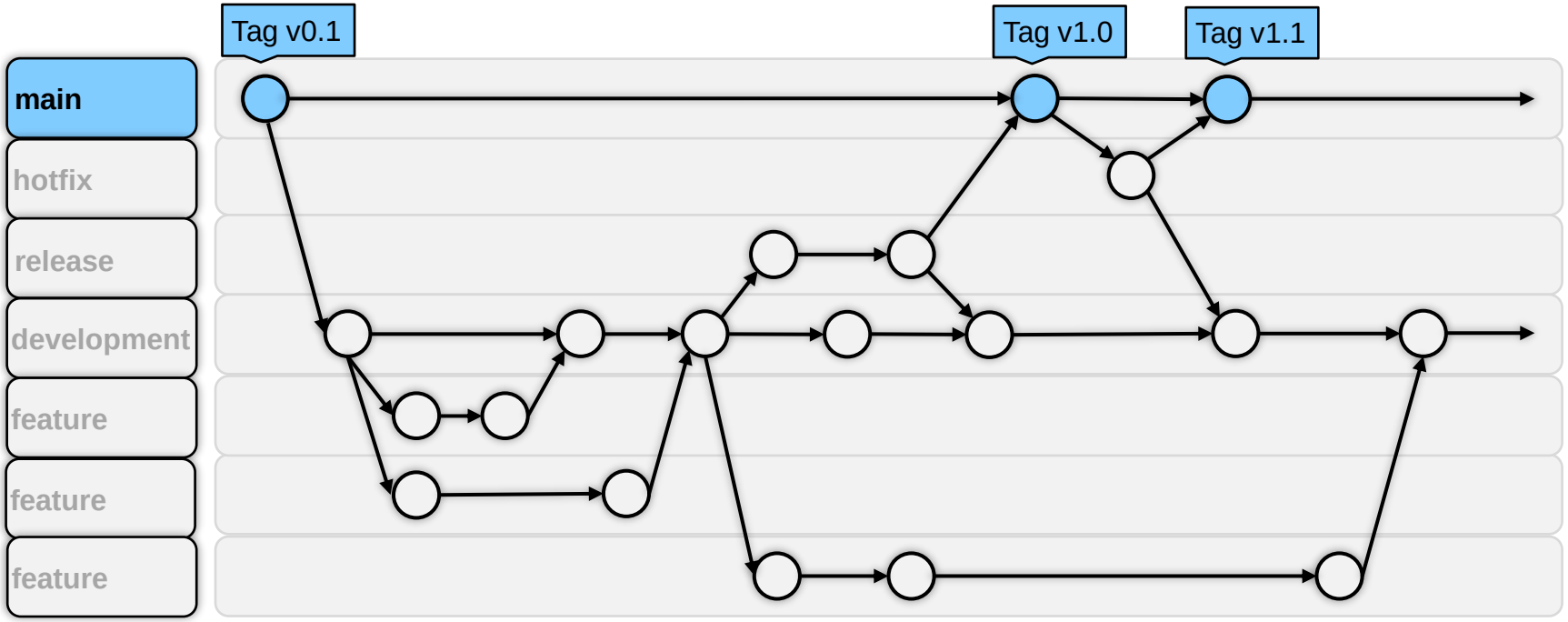
- Enthält im Gitflow-Workflow ausschließlich offizielle Releases
- Existiert fortlaufenden im Projekt
- Es wird niemals direkt auf den main Branch committet, Änderungen nur durch Merge-Requests



Gitflow-Workflow – Aufbau

Main Branch

- Commits des main-Branches werden oft mit Release-Tags versehen
- Bietet Überblick über den Release Verlauf
  - Ermöglicht auschecken älterer Version
  - Ggf. Fehlerbehebung in alten Versionen, wenn mehrere zeitgleich im Betrieb sind

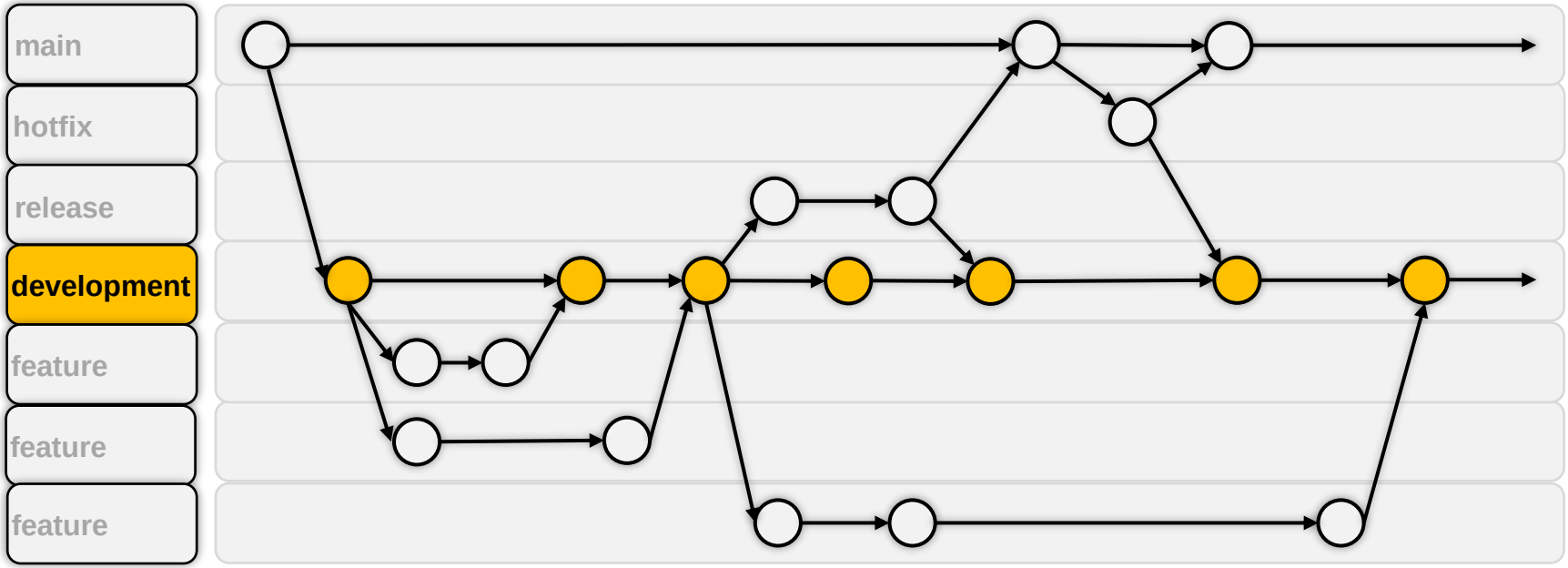


Gitflow-Workflow – Aufbau

anderScore trust in competence

Development Branch

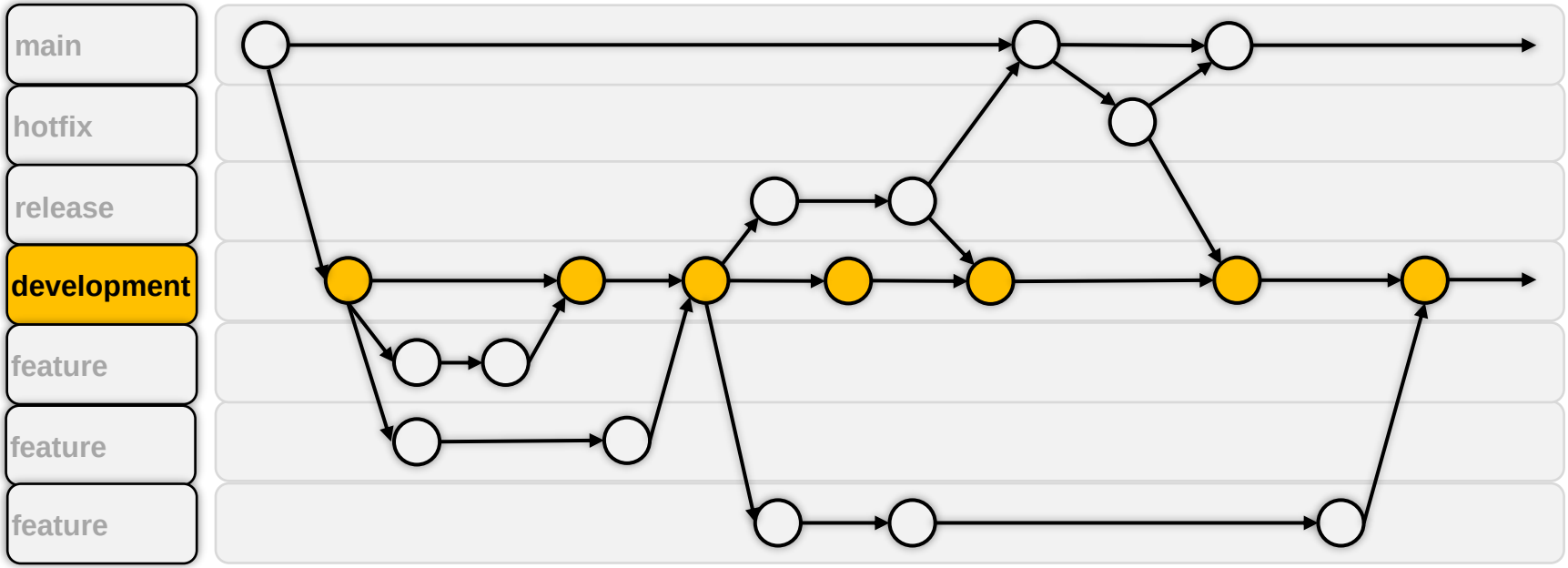
- Häufig nur kurz als **dev** bezeichnet
- Neben **main** der zweite dauerhaft existierende Branch
- Dient zur Integration von Features und Bugfixes
- Sollte immer eine vollfunktionsfähige Version der Software enthalten



Gitflow-Workflow – Aufbau

Development Branch

- Direkte Commits auf Development sind unüblich und sollten vermieden werden
- Enthält die komplette Historie des Projekts

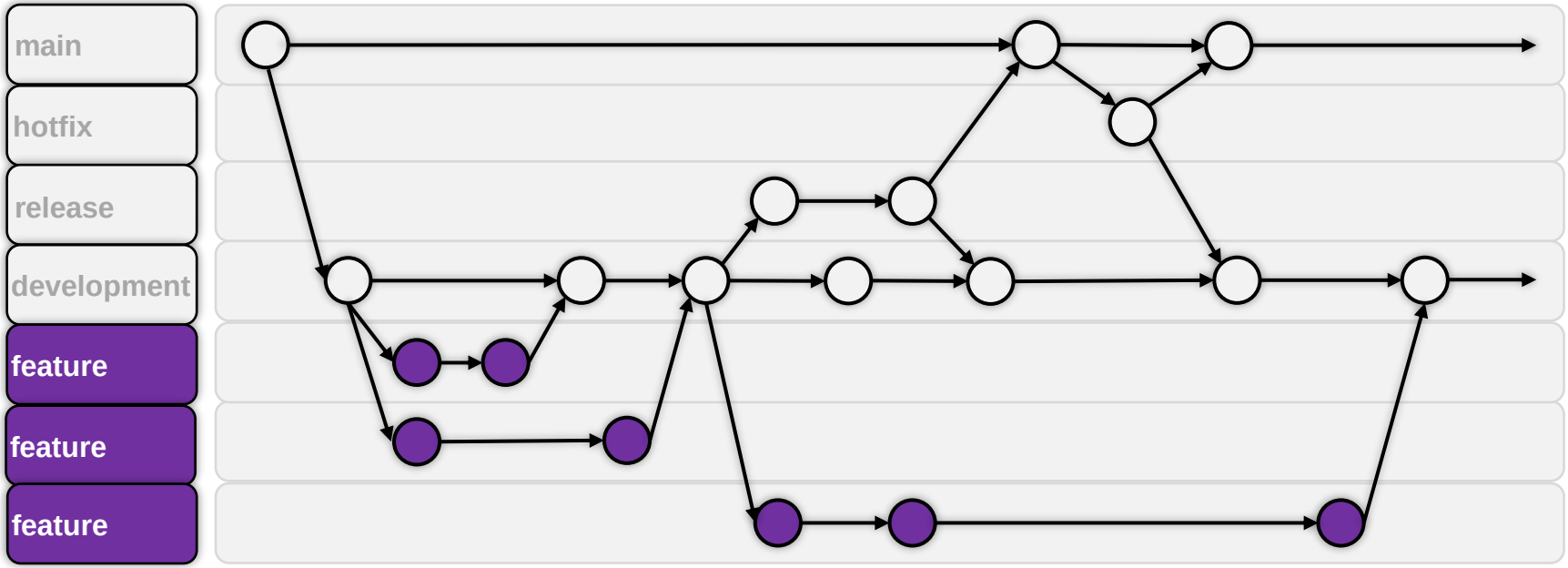


Gitflow-Workflow – Aufbau

anderScore trust in competence

Feature Branches

- Zweigen vom Development Branch ab
- Entwickler entwickelt im Feature Branch getrennt von anderen Branches sein Feature
- Isolation ermöglicht Experimente ohne andere Branches zu beeinflussen

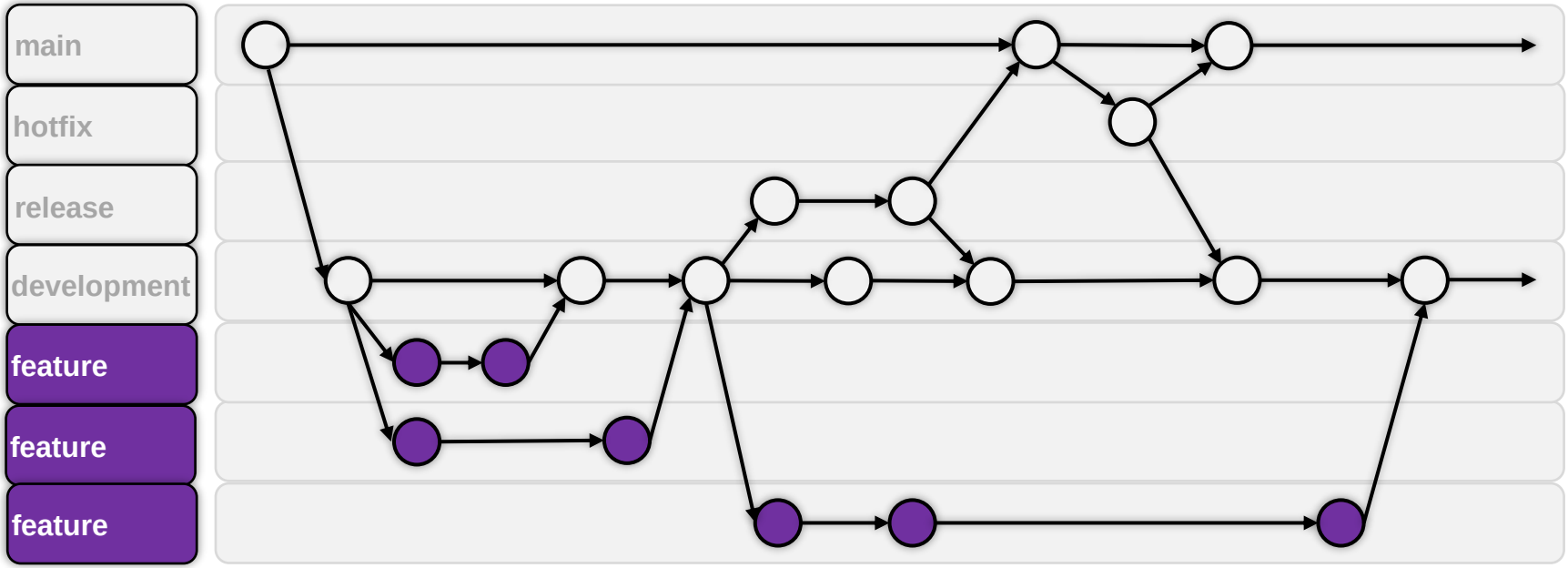


Gitflow-Workflow – Aufbau

Feature Branches

- Oft langlebigere Feature Branches als bei anderen Workflows
  - Kann Integration von Features in Development erschweren
  - Entwickler ist selbst verantwortlich Änderungen vom Development zeitnah in seinen Feature Branch einzubauen, um auf einem aktuellen Stand zu bleiben

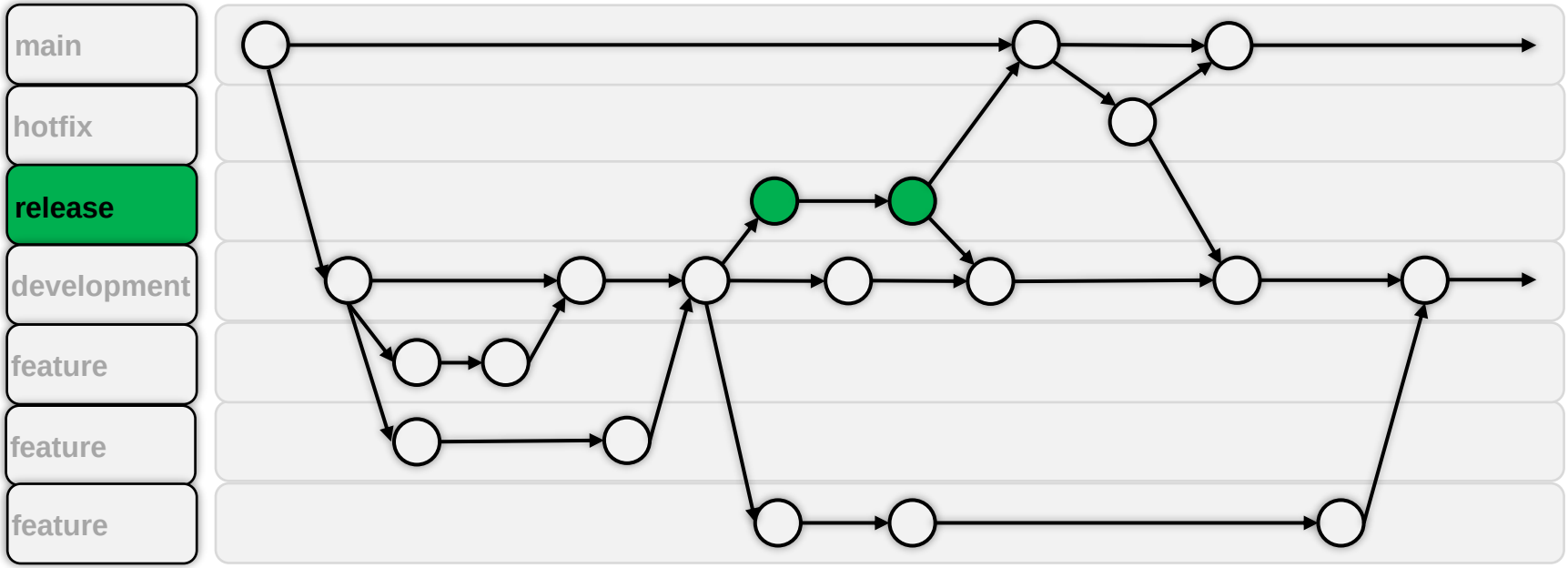
    ➔ Falls möglich Feature regelmäßig auf Development rebasen



Gitflow-Workflow – Aufbau

Release Branches

- Werden für die Release Phase verwendet
- Zweigen vom Development ab, wenn dieser zu einem Release bereit ist
- In einem Release Branch werden keine Features mehr hinzugefügt

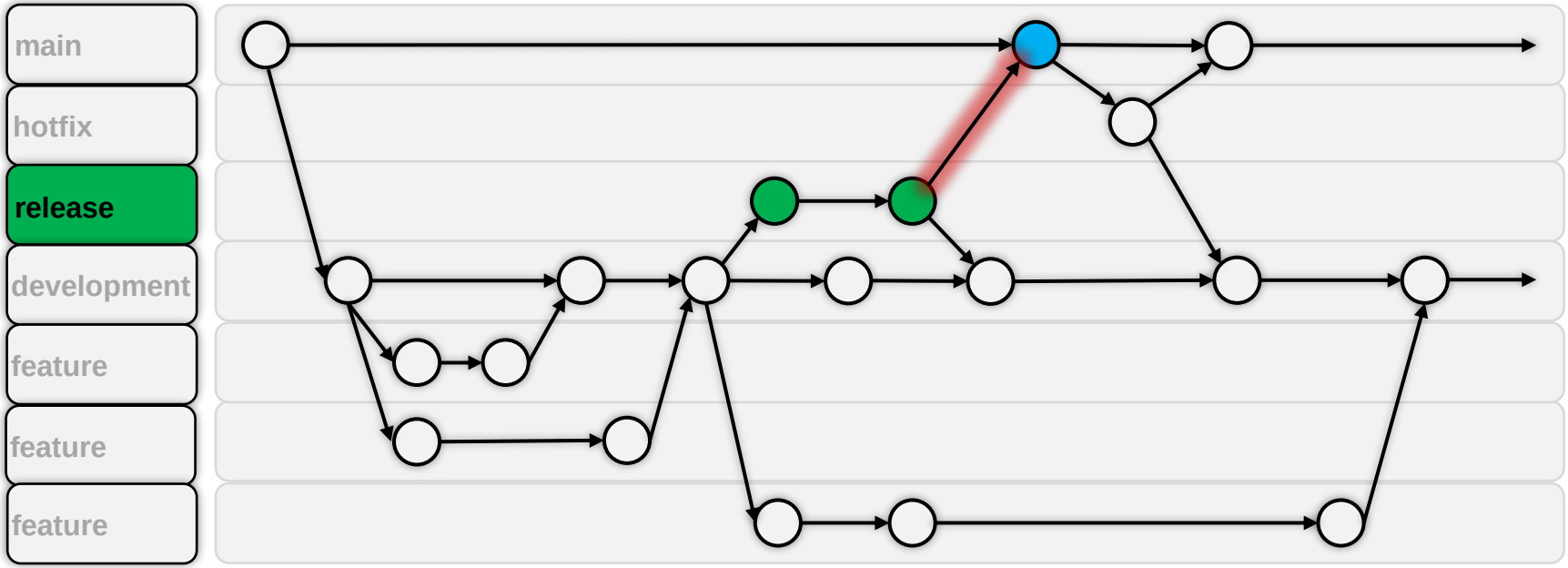


Gitflow-Workflow – Aufbau

anderScore trust in competence

Release Branches

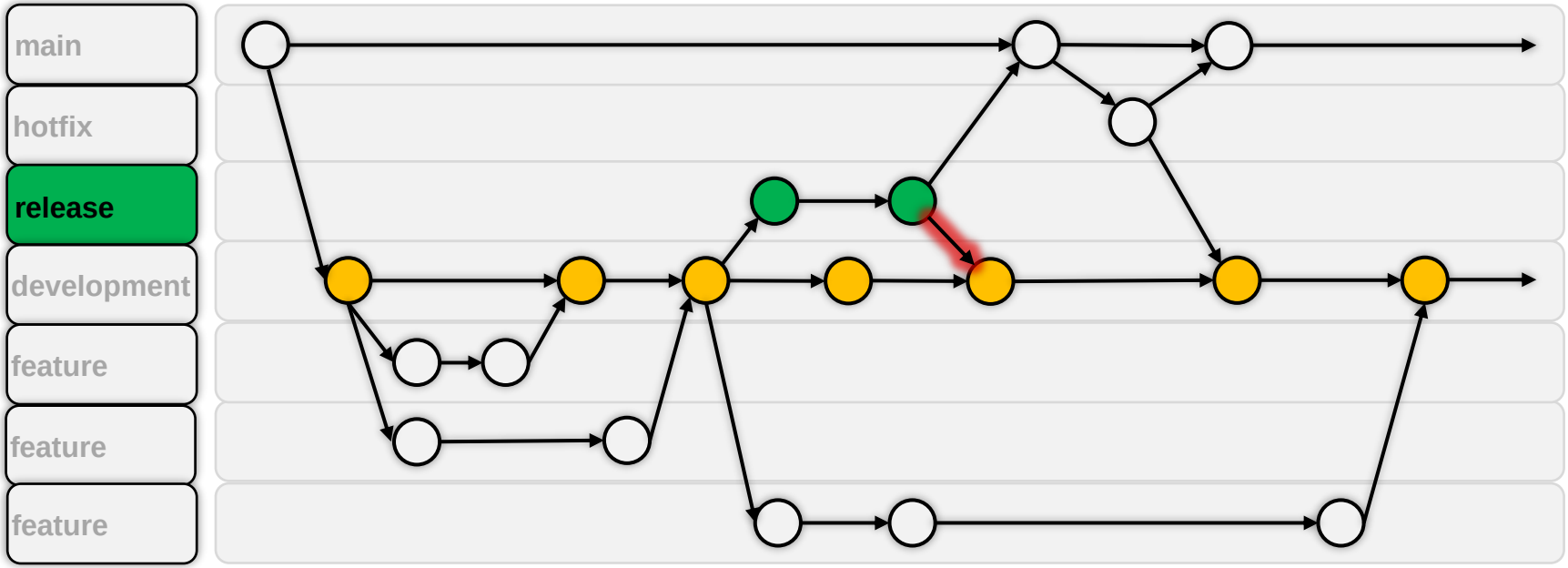
- Enthalten ausschließlich Bugfixes und für den Anpassung für das Release
- Ein Release Branch wird nach Abschluss der Anpassungen in den Main-Branch gemerged und gilt danach als abgeschlossen
- Für das nächste Release wird ein neuer Release Branch angelegt



Gitflow-Workflow – Aufbau

Release Branches

- **Wichtig**: Bugfixes und Anpassungen nach dem Release wieder in den Development Branch überführen
  - Sonst landen Anpassungen nur im Main-Branch
  - Verhindert, dass man beim nächsten Release wieder die gleichen Fehler beheben muss

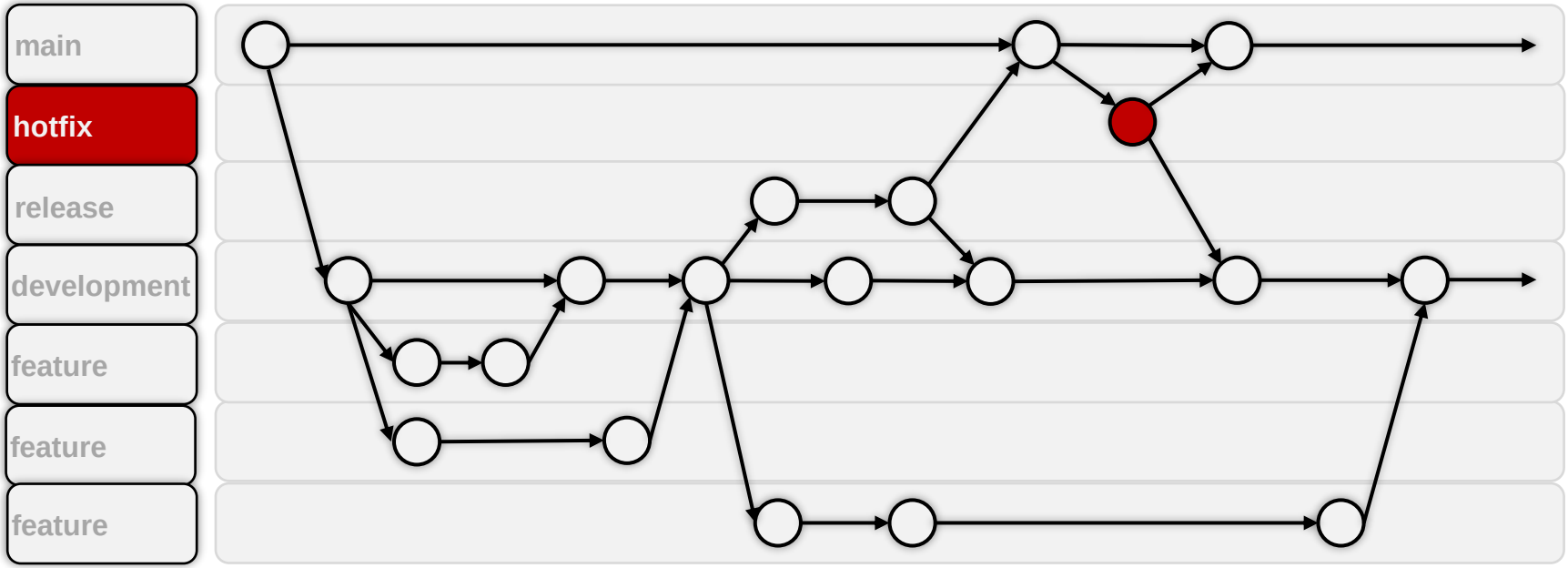


Gitflow-Workflow – Aufbau

anderScore trust in competence

Hotfix Branches

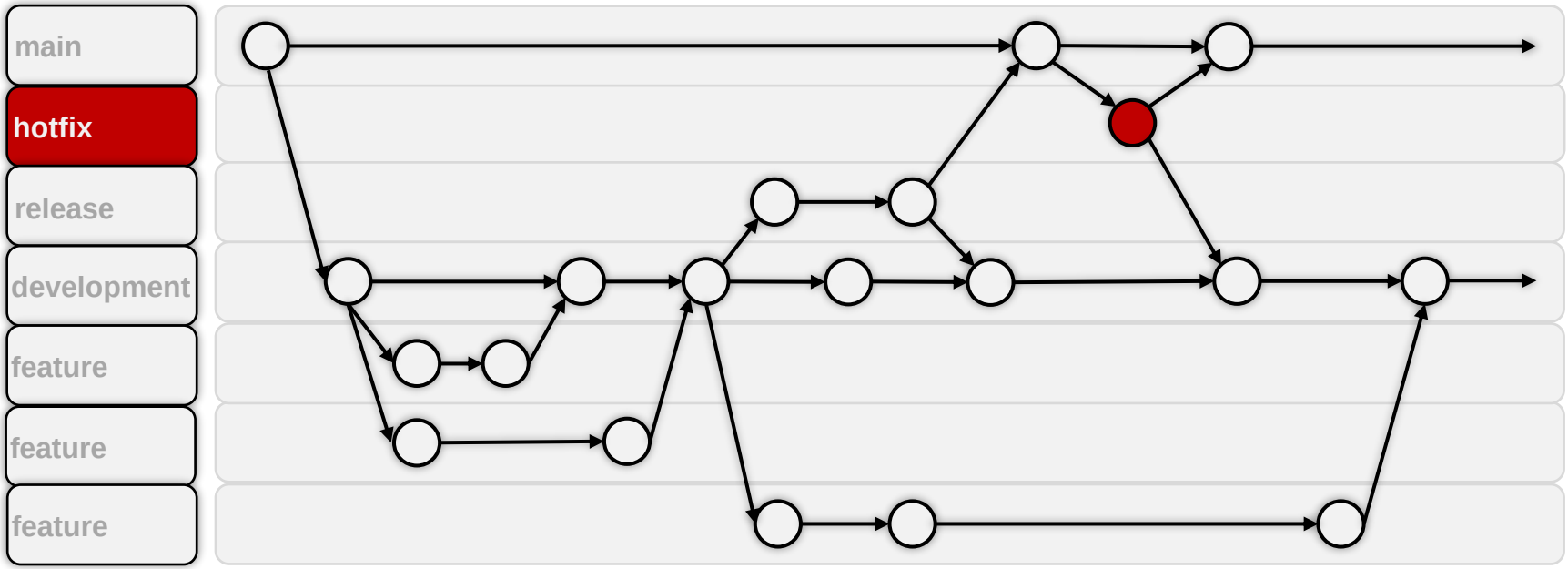
- Sind für Hotfixes der Releases gedacht
- Zweigen vom Main-Branch ab und werden nach Abschluss wieder in diesen gemerged
- Beheben Bugs, deren Fix nicht bis zum nächsten Release warten kann



Gitflow-Workflow – Aufbau

Hotfix Branches

- **Wichtig auch hier**: Unbedingt Fixes in den Development übernehmen
  Sonst läuft man Gefahr, beim nächsten Release wieder den gleichen Fehler mit auszuliefern



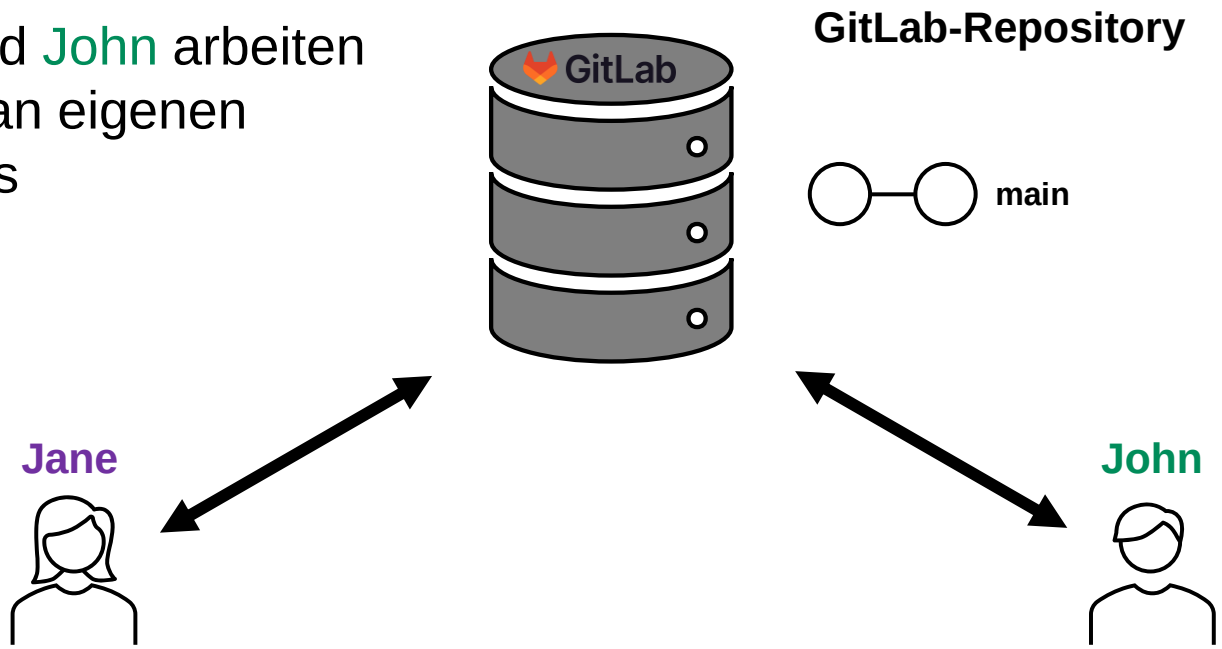
Git

Arbeiten im Gitflow-Workflow

Arbeiten im Gitflow-Workflow

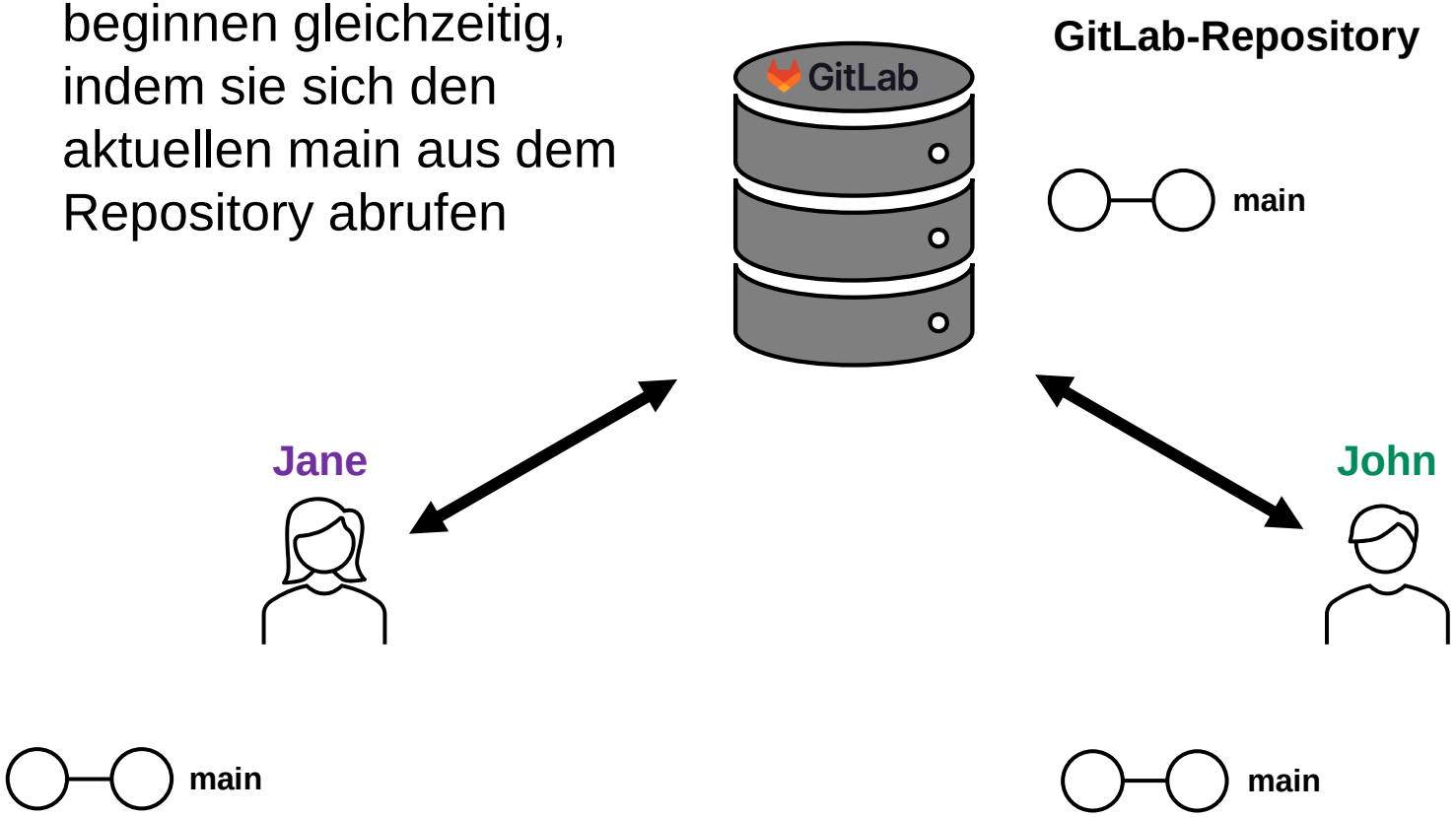
Beispielszenario

- Jane und John arbeiten jeweils an eigenen Features



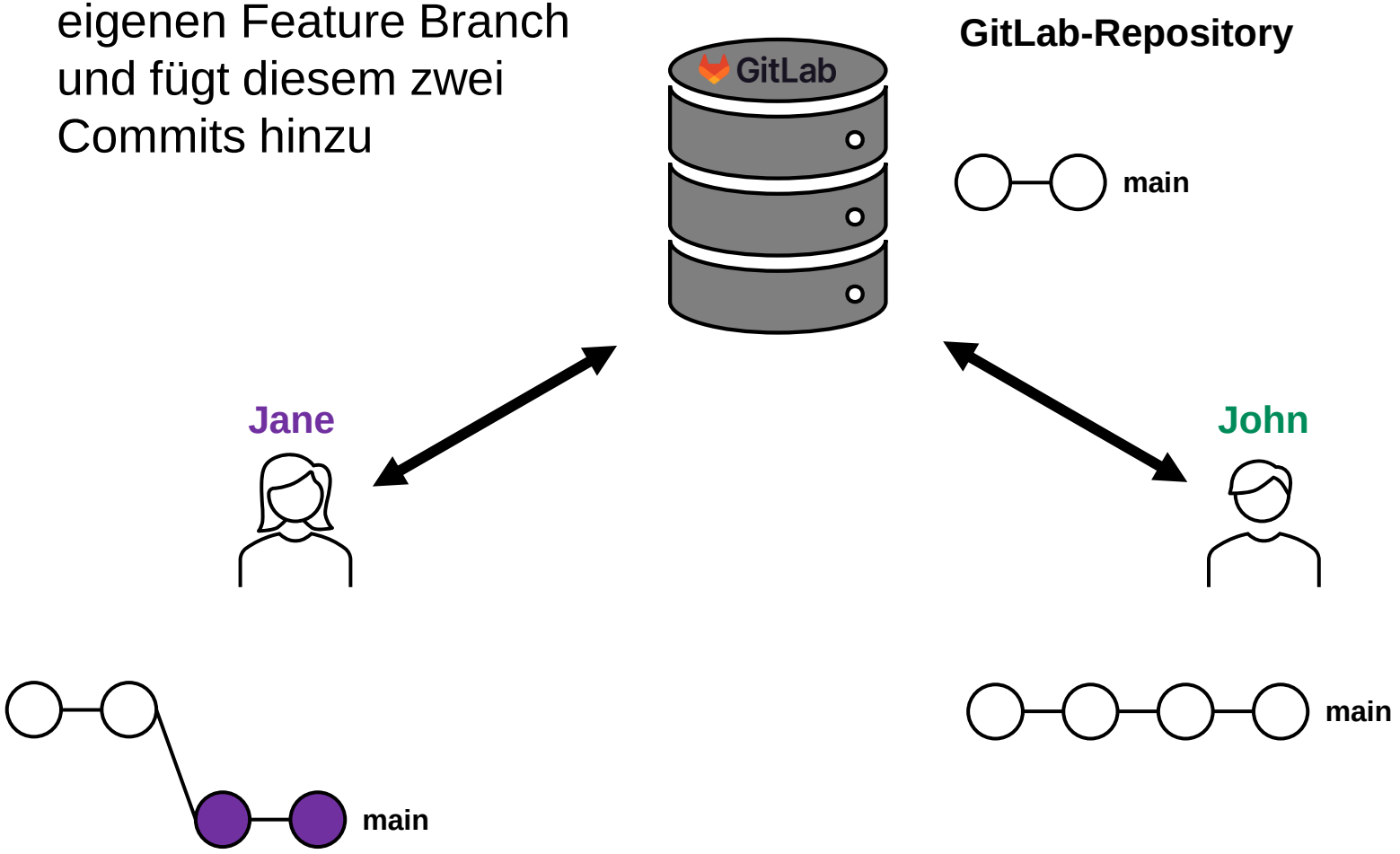
Arbeiten im Gitflow-Workflow

- Jane und John beginnen gleichzeitig, indem sie sich den aktuellen main aus dem Repository abrufen



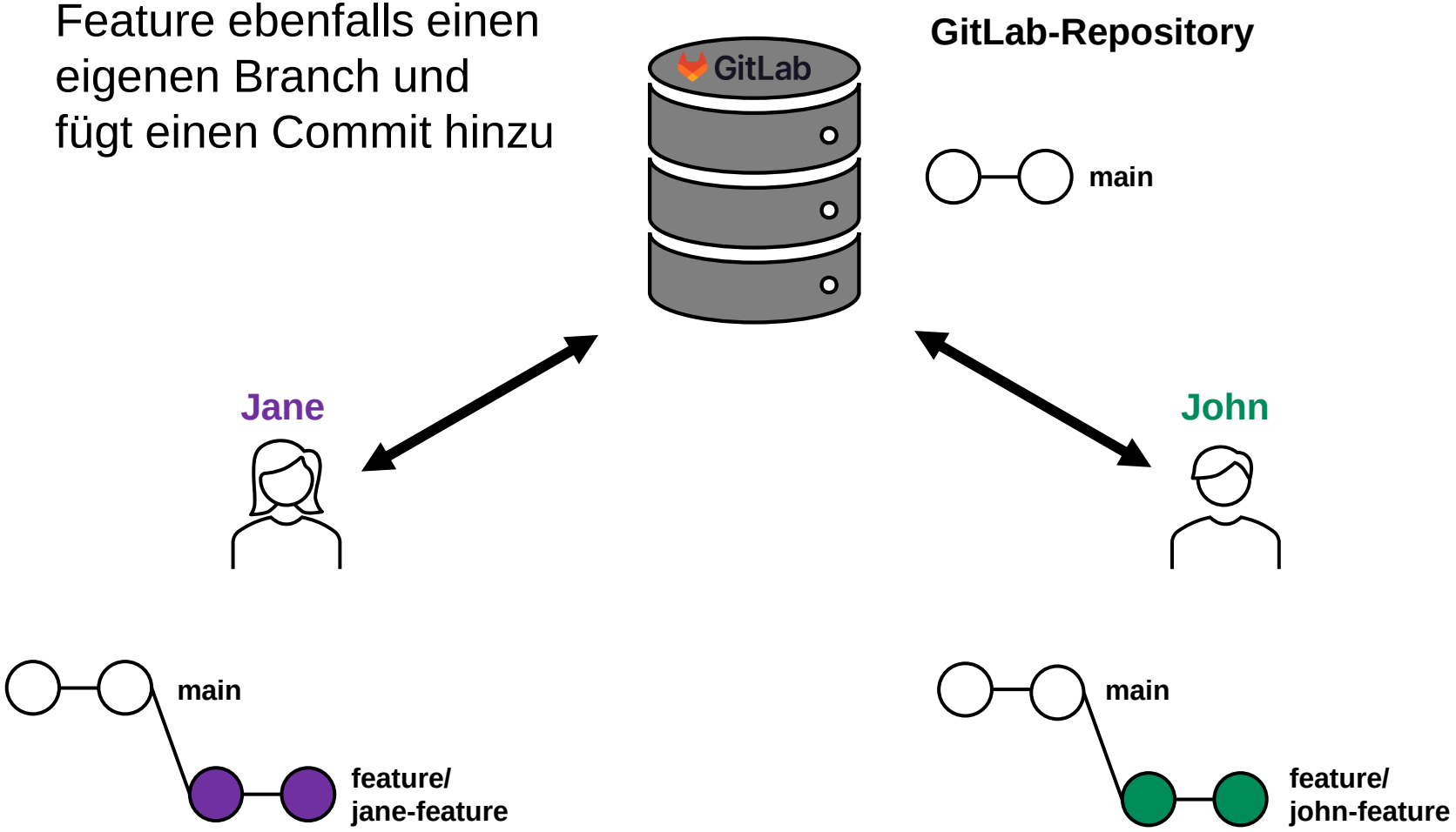
Arbeiten im Gitflow-Workflow

- Jane erstellt lokal einen eigenen Feature Branch und fügt diesem zwei Commits hinzu



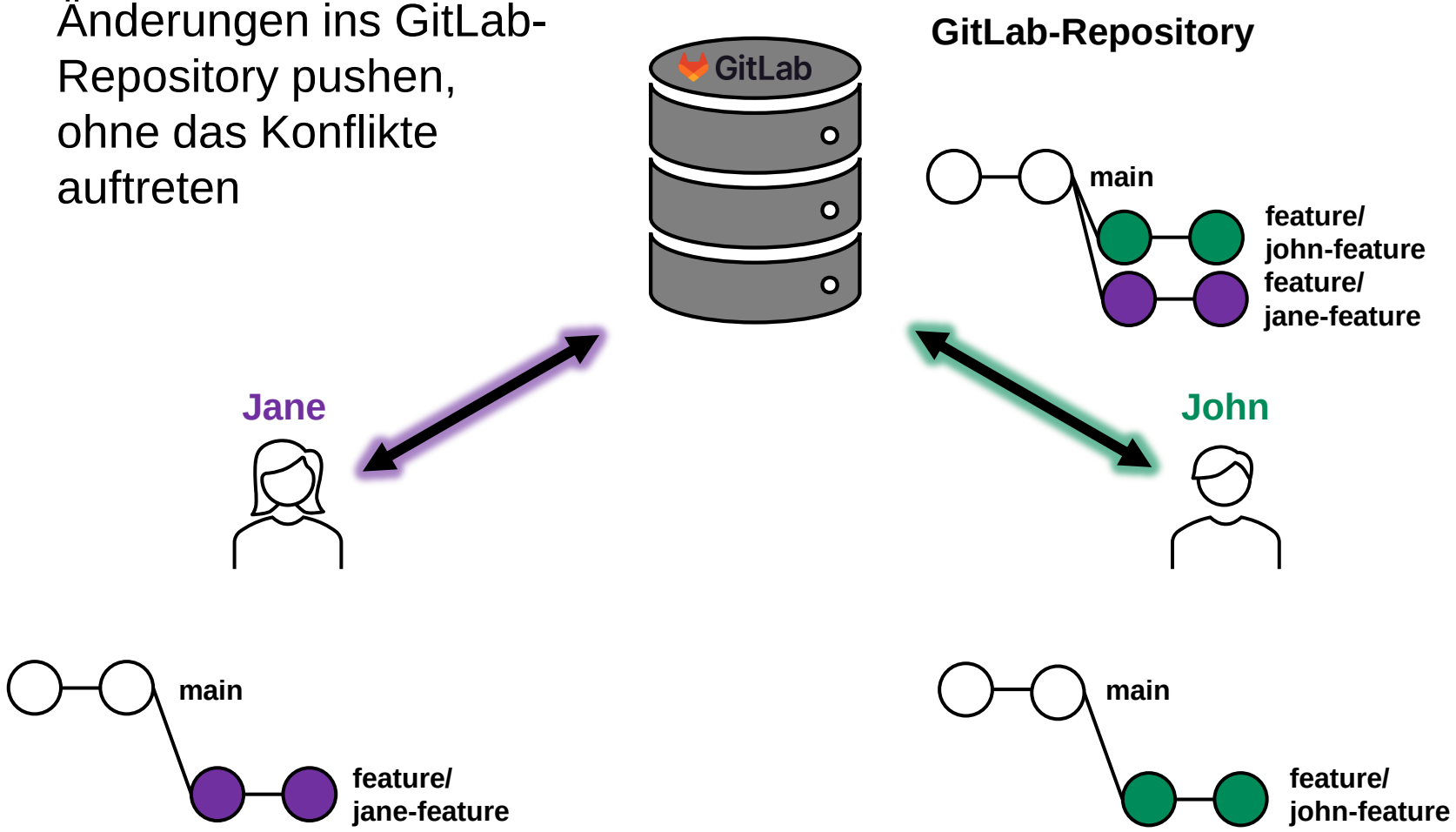
Arbeiten im Gitflow-Workflow

- John erstellt für sein Feature ebenfalls einen eigenen Branch und fügt einen Commit hinzu



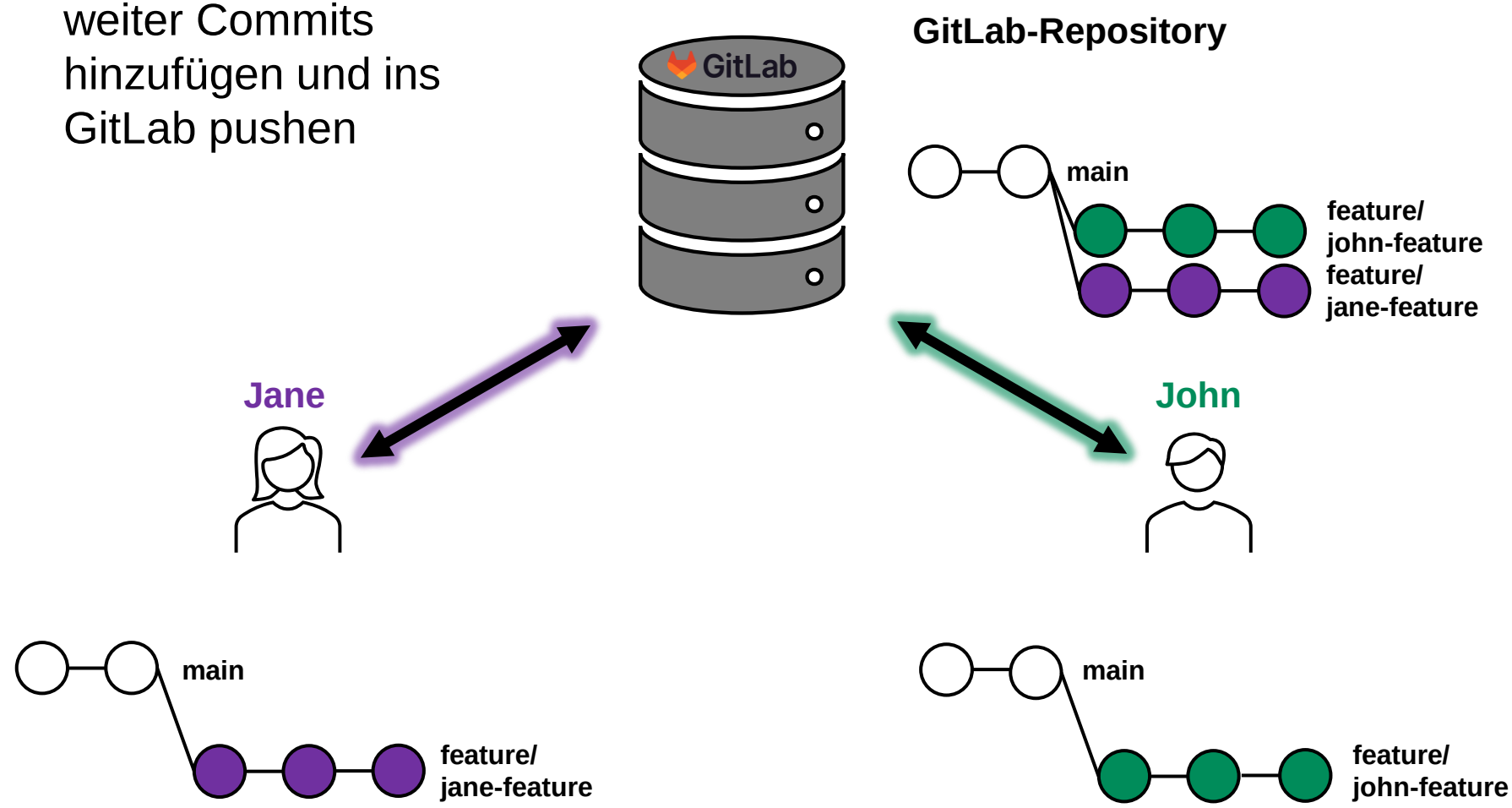
Arbeiten im Gitflow-Workflow

- Beide können ihre Änderungen ins GitLab-Repository pushen, ohne das Konflikte auftreten



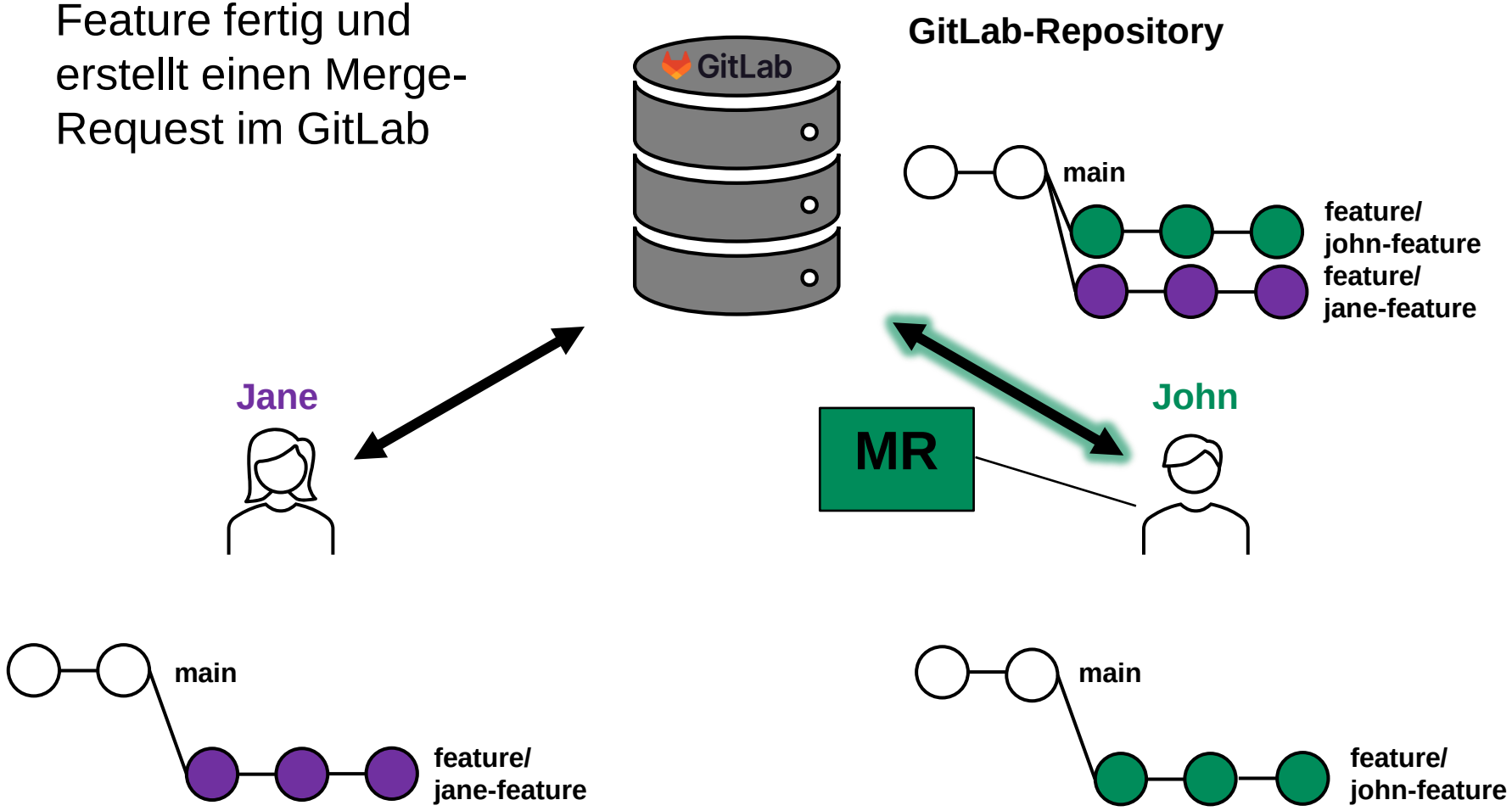
Arbeiten im Gitflow-Workflow

- Beide können beliebig weiter Commits hinzufügen und ins GitLab pushen



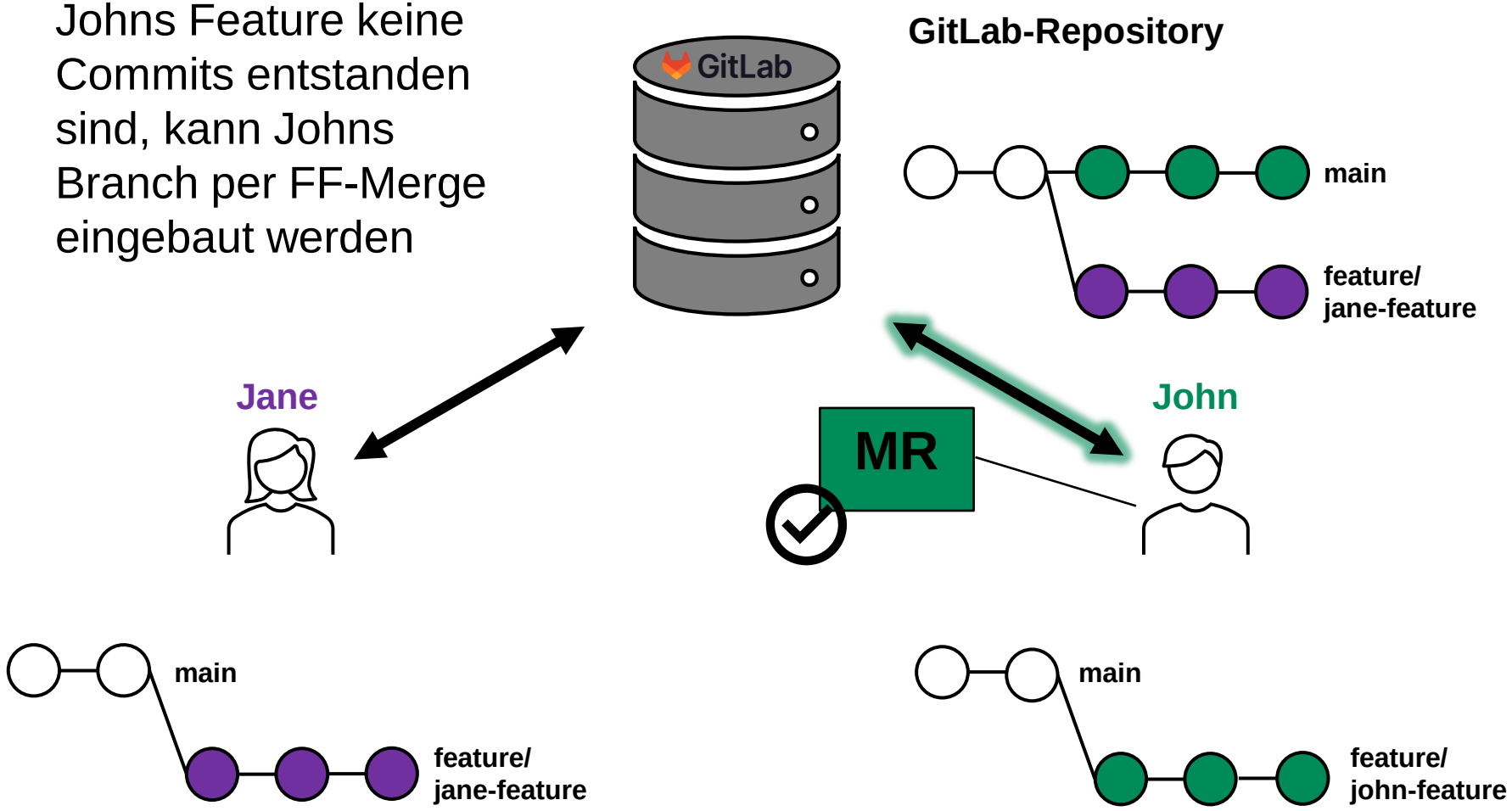
Arbeiten im Gitflow-Workflow

- John ist mit seinem Feature fertig und erstellt einen Merge-Request im GitLab



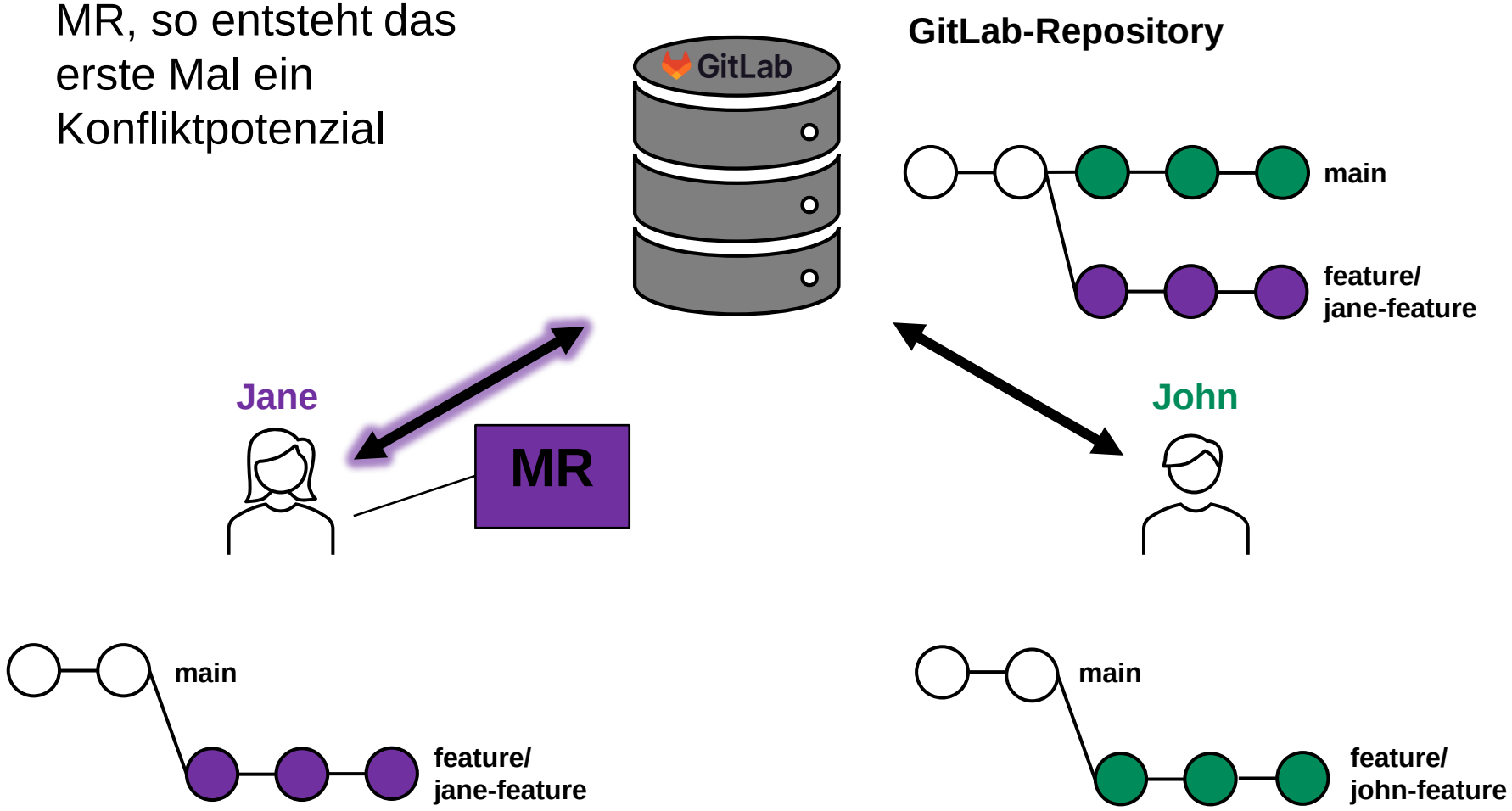
Arbeiten im Gitflow-Workflow

- Da auf dem main seit Johns Feature keine Commits entstanden sind, kann Johns Branch per FF-Merge eingebaut werden



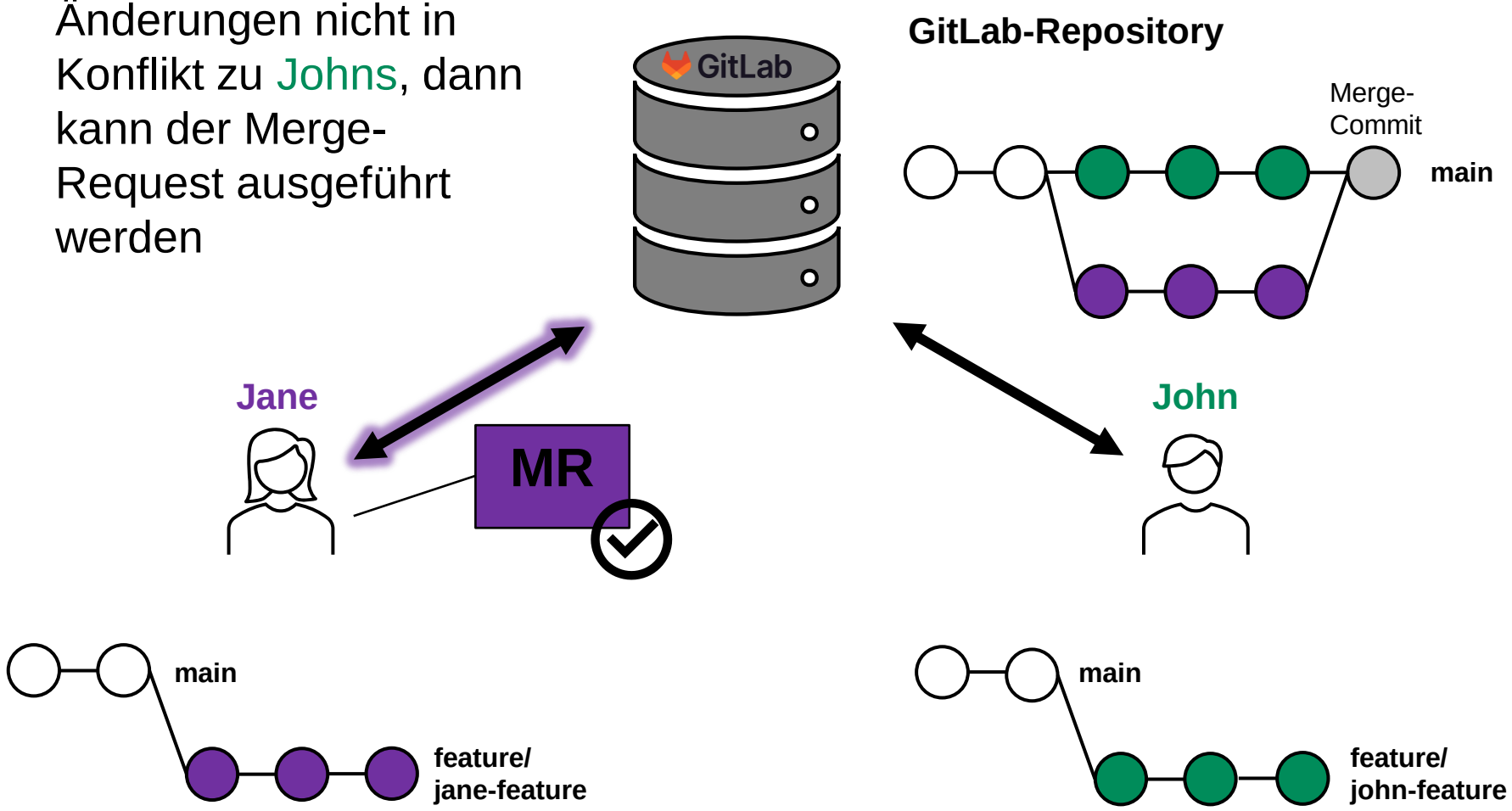
Arbeiten im Gitflow-Workflow

- Stellt nun Jane ihren MR, so entsteht das erste Mal ein Konfliktpotenzial



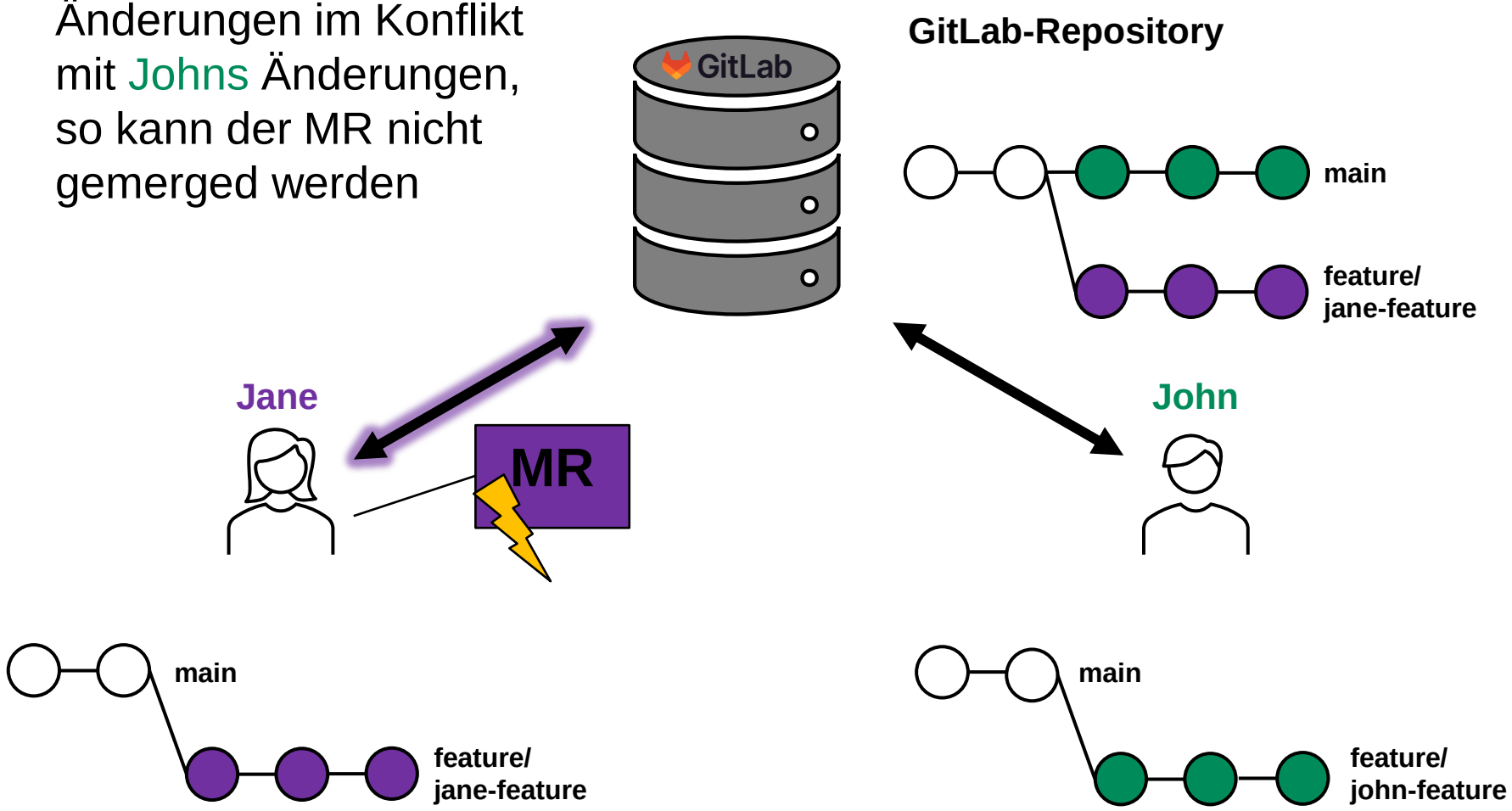
Arbeiten im Gitflow-Workflow

- Stehen Janes Änderungen nicht in Konflikt zu Johns, dann kann der Merge-Request ausgeführt werden



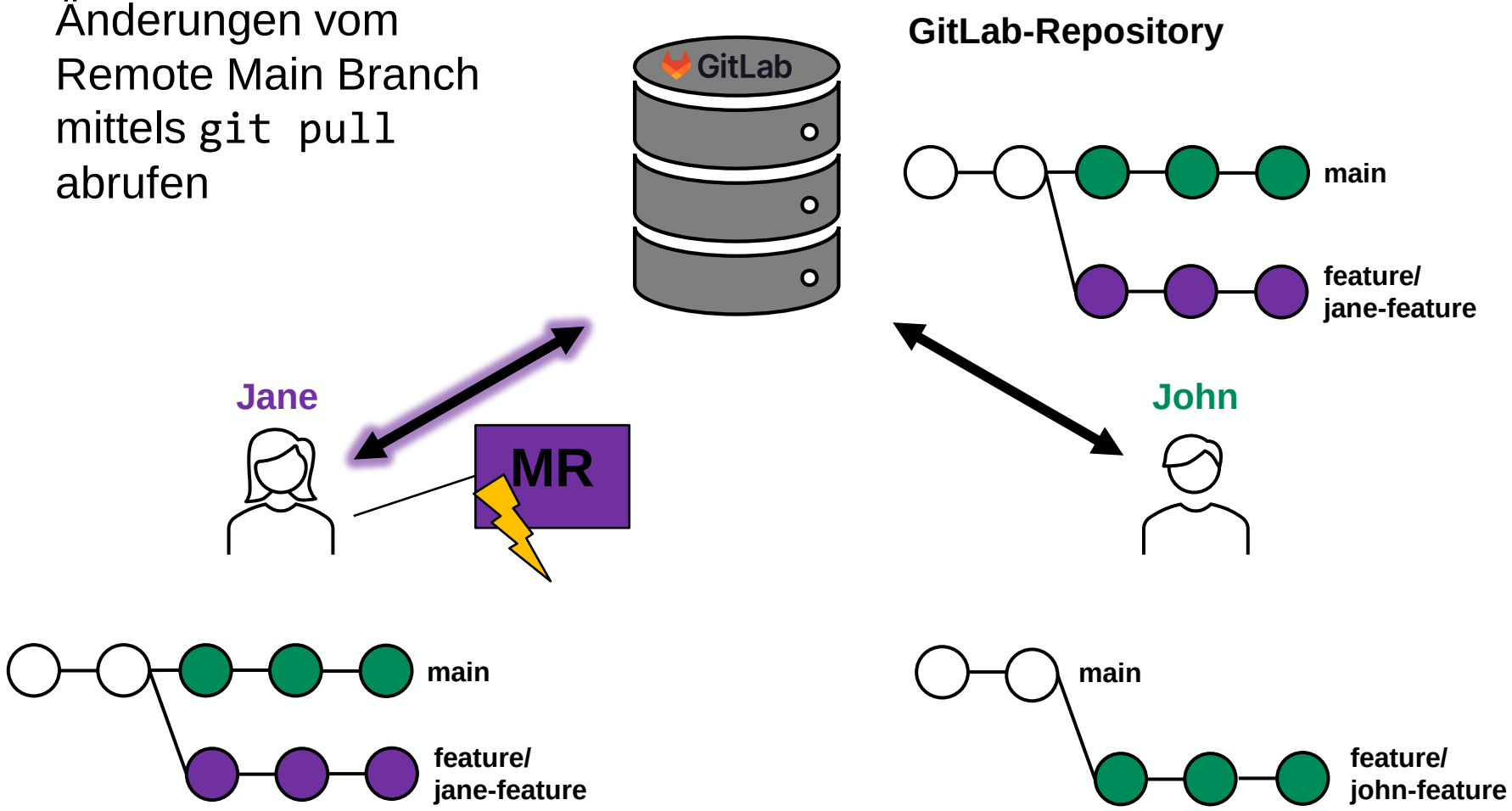
Arbeiten im Gitflow-Workflow

- Stehen Janes Änderungen im Konflikt mit Johns Änderungen, so kann der MR nicht gemerged werden



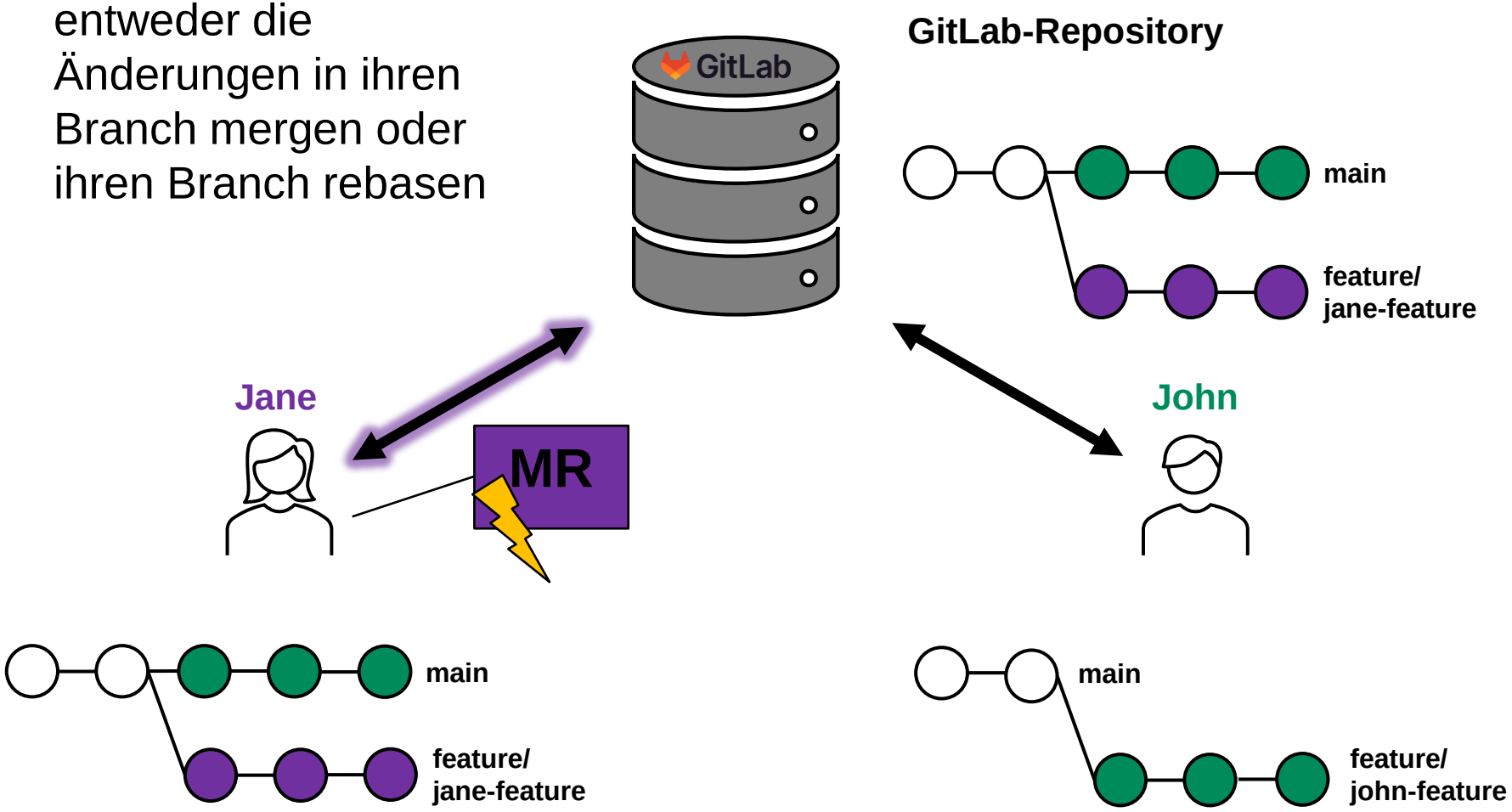
Arbeiten im Gitflow-Workflow

- Jane muss die Änderungen vom Remote Main Branch mittels `git pull` abrufen



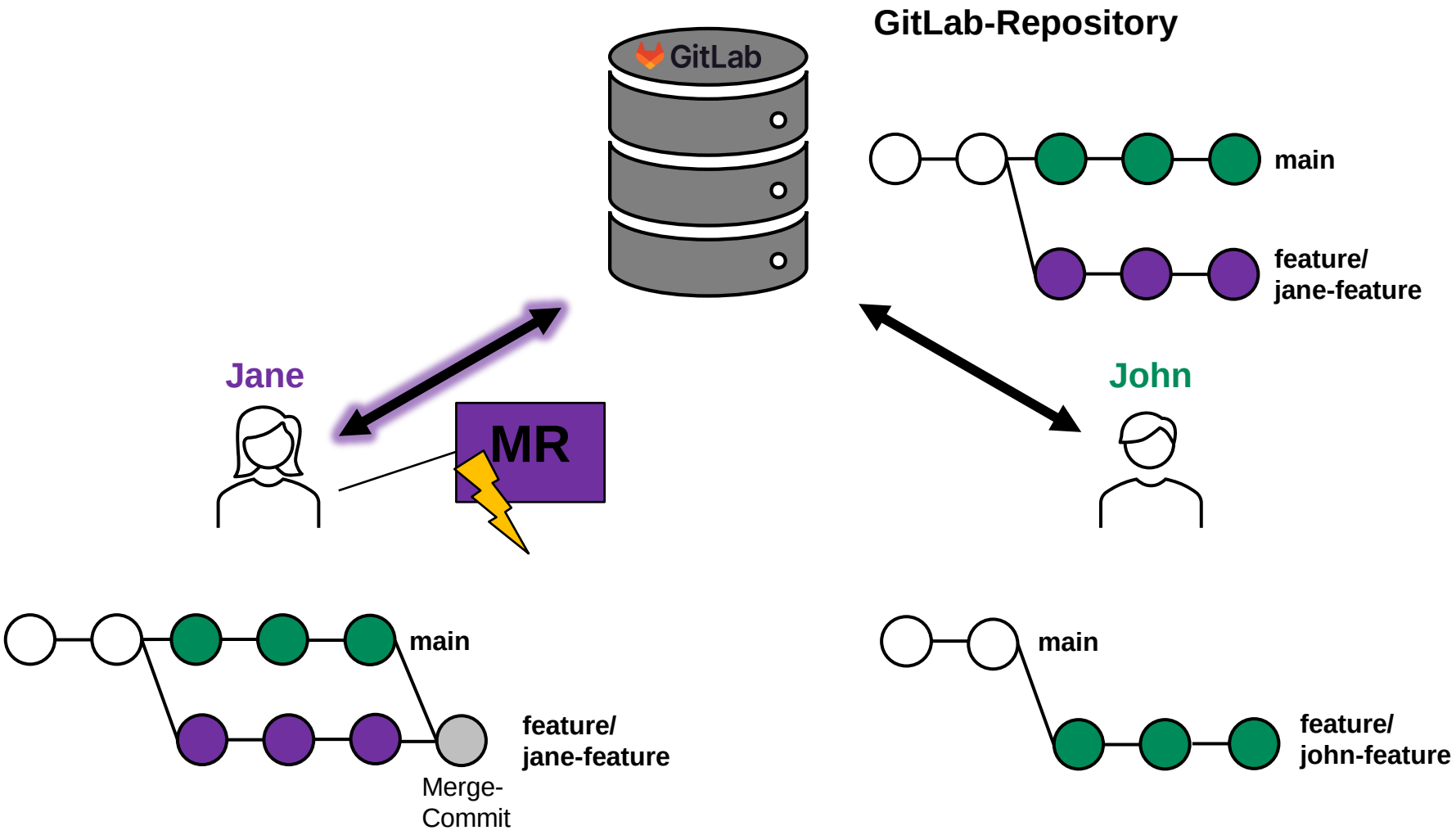
Arbeiten im Gitflow-Workflow

- Jane kann nun entweder die Änderungen in ihren Branch mergen oder ihren Branch rebasen



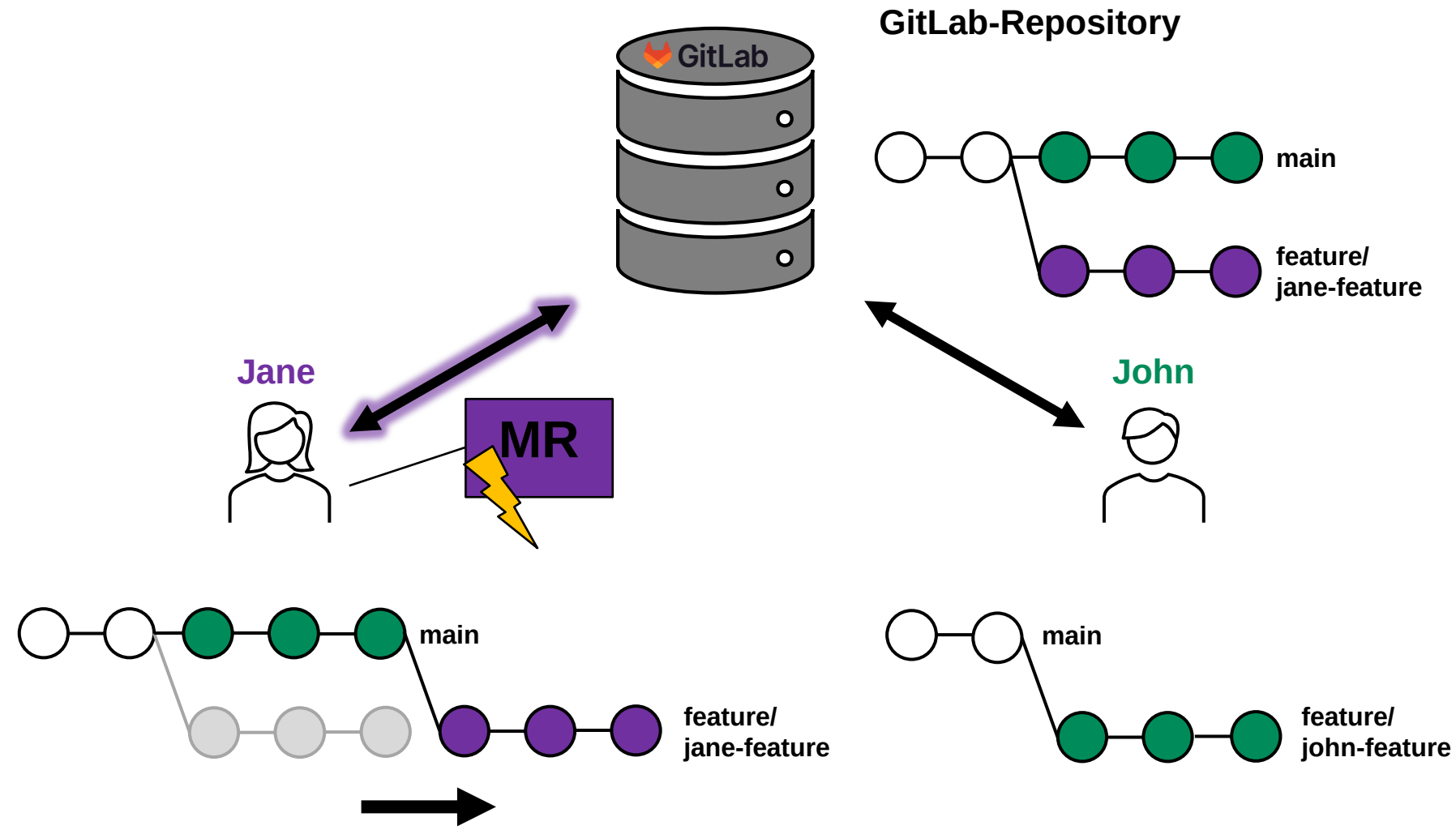
Arbeiten im Gitflow-Workflow

Merge



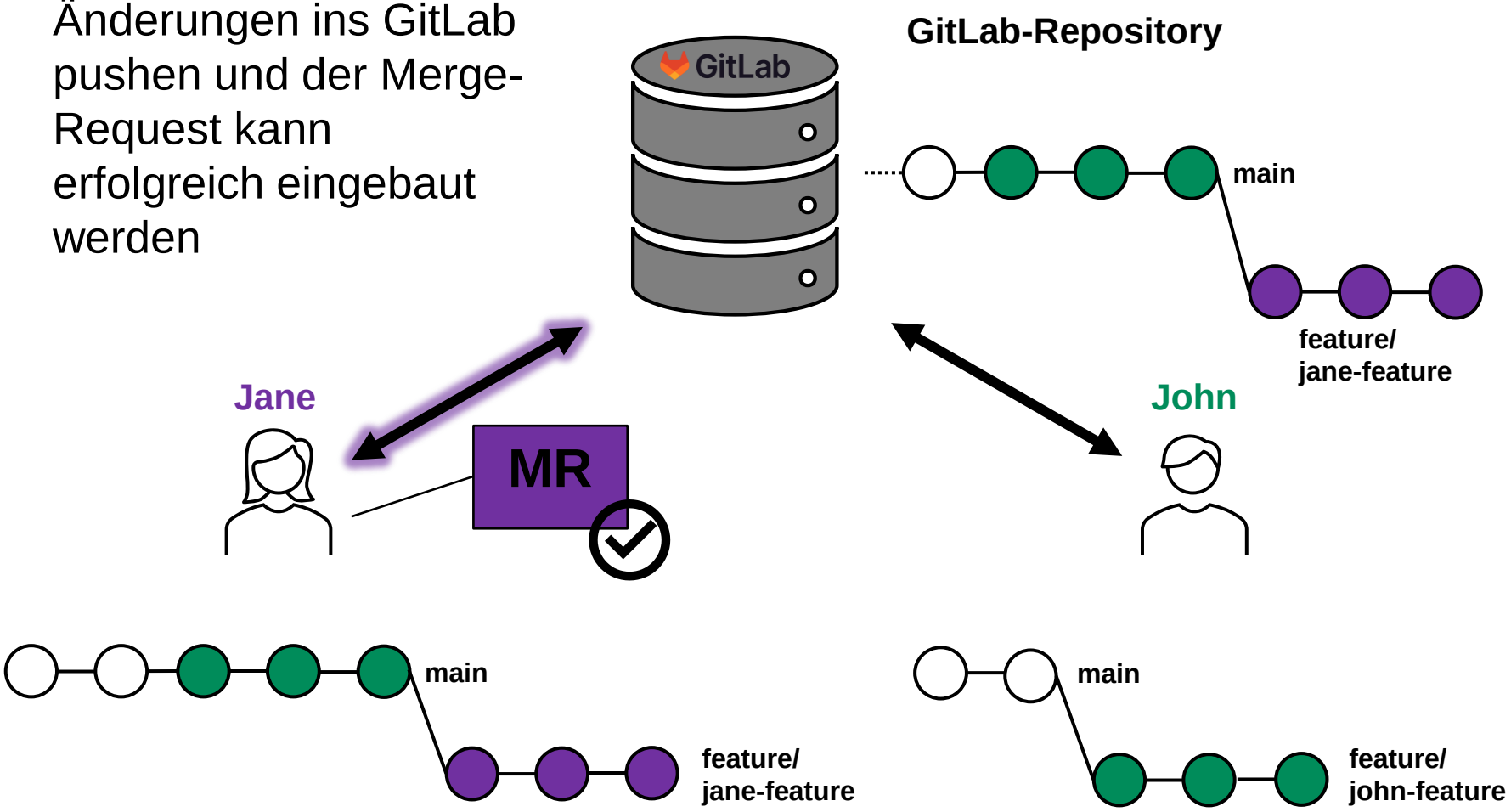
Arbeiten im Gitflow-Workflow

Rebase



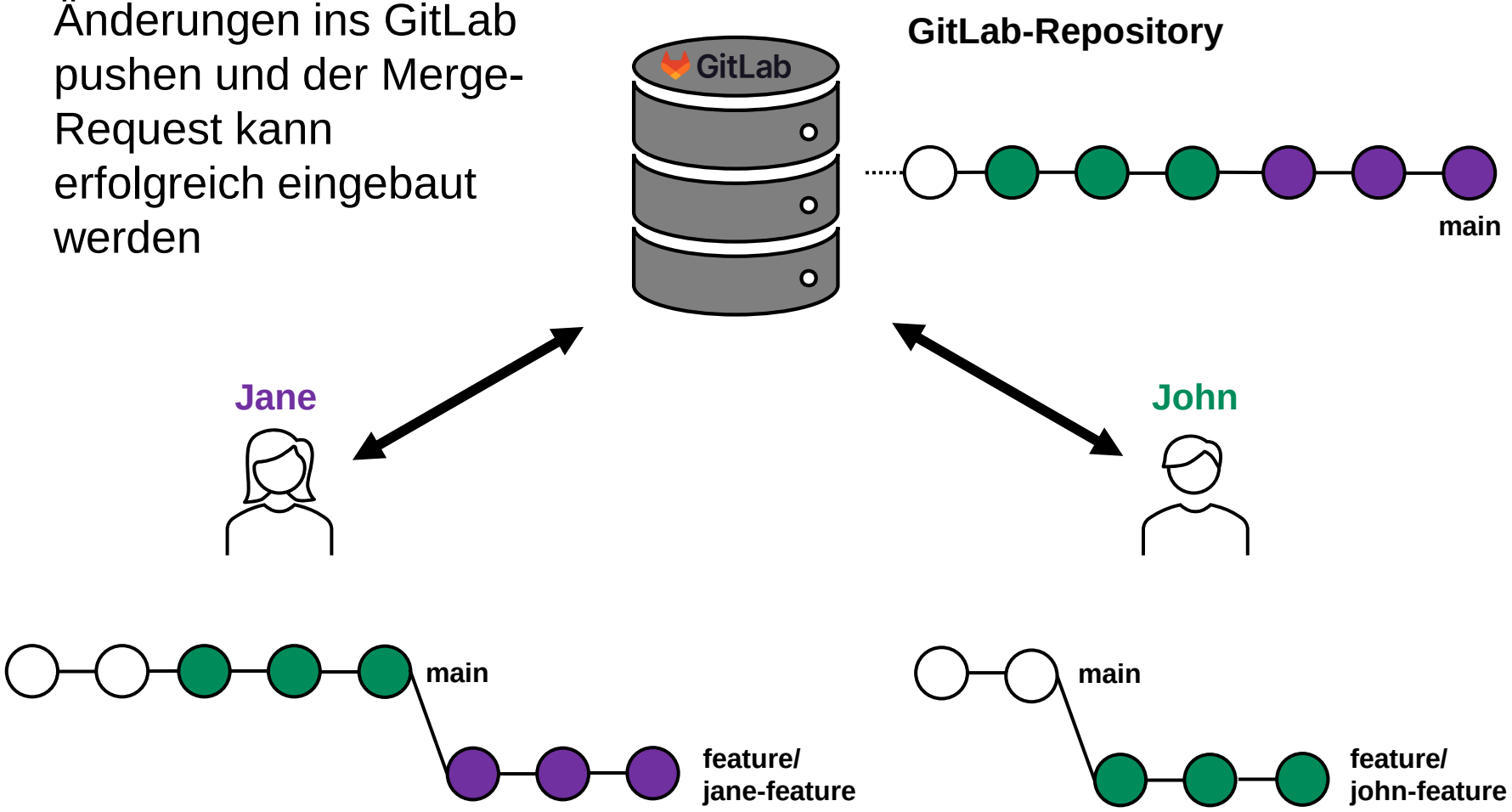
Arbeiten im Gitflow-Workflow

- Danach kann Jane ihre Änderungen ins GitLab pushen und der Merge-Request kann erfolgreich eingebaut werden



Arbeiten im Gitflow-Workflow

- Danach kann Jane ihre Änderungen ins GitLab pushen und der Merge-Request kann erfolgreich eingebaut werden



Gitflow-Workflow

Use Cases und Fazit

Gitflow-Workflow – Use Cases

anderScore trust in competence

- Der Workflow selbst stammt von 2010 und ist damit im Vergleich zu moderneren Konzepten teilweise überholt
- Hat klare Vorteile
  - Stark strukturierte Aufteilung mit starker Trennung
  - Vereinfacht parallele Entwicklung
  - Robustere Releases durch dedizierte Release Branches
  - Bessere Versionshistorie
- Aber auch klare Nachteile
  - Langlebige Branches erhöhen Konfliktpotenzial beim Mergen
  - Struktur mit seltenen Updates erschwert Continuous Deployment stark
  - Continuous Integration zwar mittels Development möglich, aber weniger effektiv
  - Zusätzlicher Overhead durch Branches
- Eignet sich eher für größere Teams oder größere/komplexere Projekte
  - Sinnvoll in Projekten ohne hochfrequente Releases
  - Vereinfacht Entwicklung, falls mehrere Versionen zeitgleich betrieben werden

Gitflow-Workflow – Veraltet?

- Vincent Driessen selbst hat im Jahre 2020 an seinen Blogpost eine Notiz zur Rekapitulation angefügt

”This model was conceived in 2010, now more than 10 years ago, and not very long after Git itself came into being. In those 10 years, git-flow (the branching model laid out in this article) has become hugely popular in many a software team to the point where people have started treating it like a standard of sorts — but unfortunately also as a dogma or panacea.

During those 10 years, Git itself has taken the world by a storm, and the most popular type of software that is being developed with Git is shifting more towards web apps — at least in my filter bubble. Web apps are typically continuously delivered, not rolled back, and you don't have to support multiple versions of the software running in the wild.

This is not the class of software that I had in mind when I wrote the blog post 10 years ago. If your team is doing continuous delivery of software, I would suggest to adopt a much simpler workflow (like GitHub flow) instead of trying to shoehorn git-flow into your team.

If, however, you are building software that is explicitly versioned, or if you need to support multiple versions of your software in the wild, then git-flow may still be as good of a fit to your team as it has been to people in the last 10 years. In that case, please read on.

To conclude, always remember that panaceas don't exist. Consider your own context. Don't be hating. Decide for yourself.”

– Vincent Driessen, https://nvie.com/posts/a-successful-git-branching-model/