dataport

anderScore
trust in competence

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

18.06.2024, Daniel Krämer & Malte Fischer

HECKER
CONSULTING

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

Inhalt

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

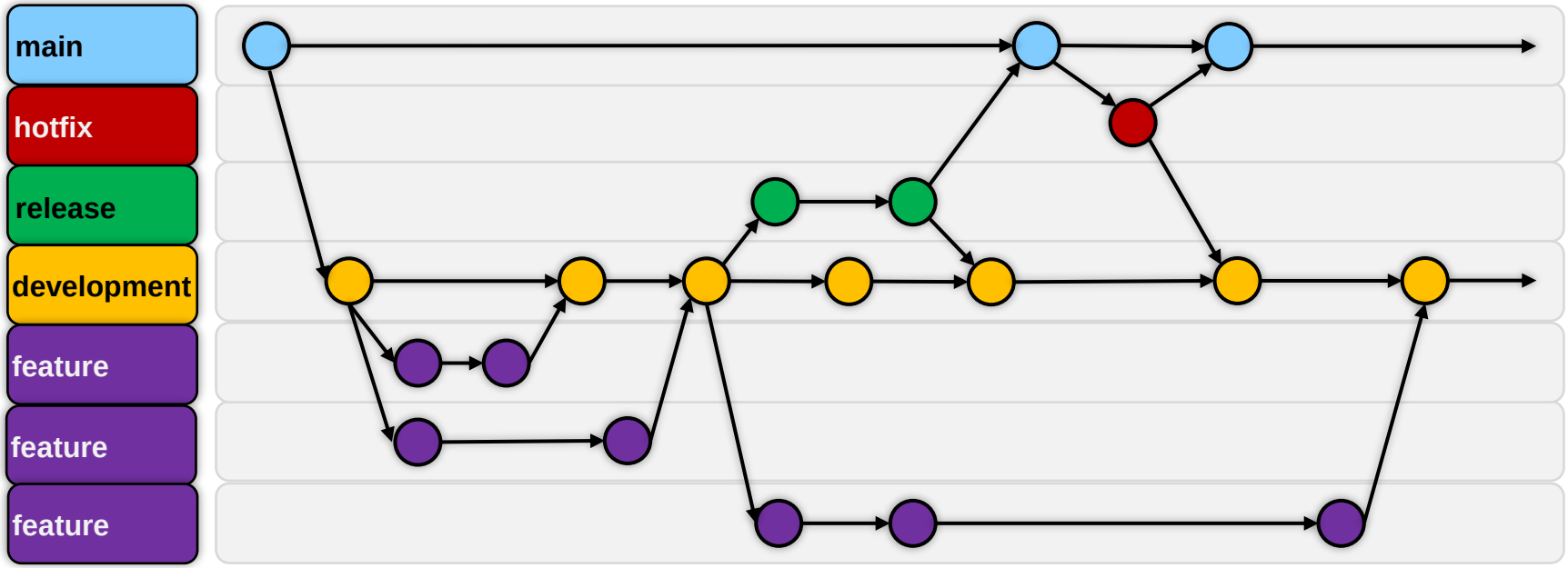
Gitflow-Workflow

Was ist der Gitflow-Workflow?

- Workflow mit Feature Branches und mehreren primären Branches
- 2010 von Vincent Driessen auf nvie veröffentlicht
- Beliebt bei vielen Teams
- Teils als veraltet angesehen, aber immernoch weit verbreitet

Gitflow-Workflow – Aufbau

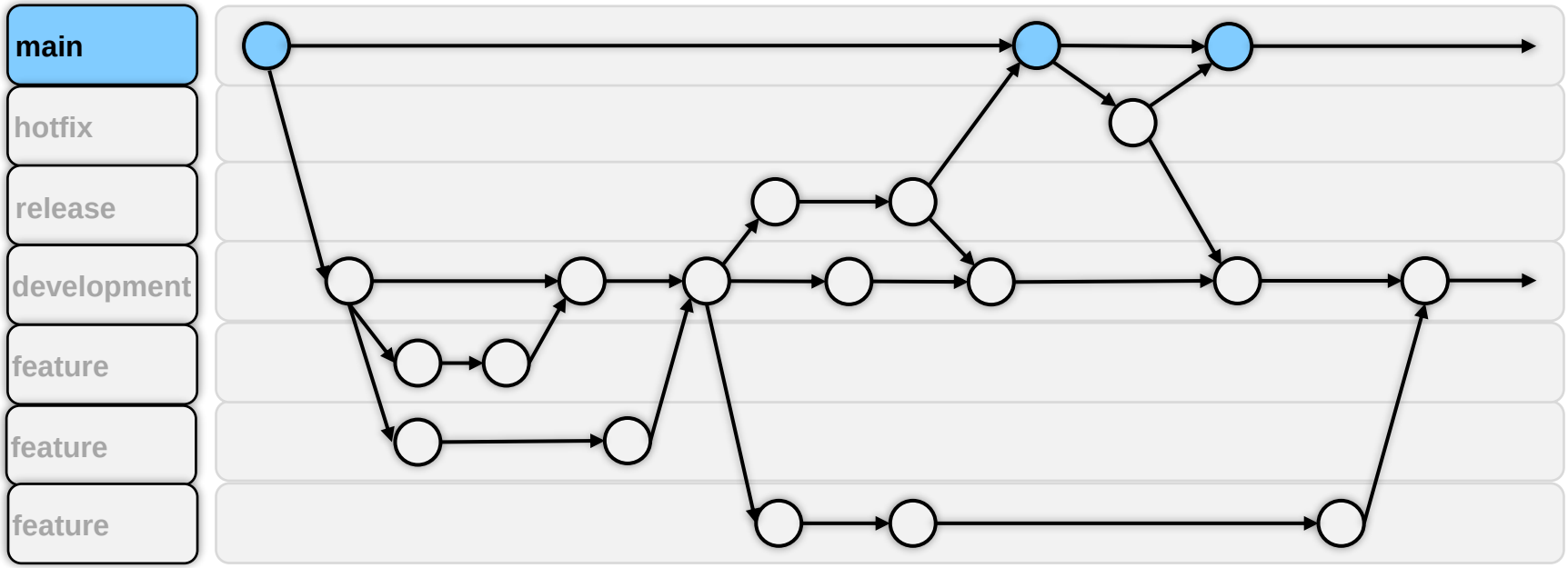
- Besteht aus mehreren Branches mit zugeteilten Rollen
- *main* und *development* zu Beginn erstellt und dauerhaft existent
- Release und Feature Branches erlauben getrenntes Arbeiten und isoliertes Experimentieren



Gitflow-Workflow – Aufbau

main Branch

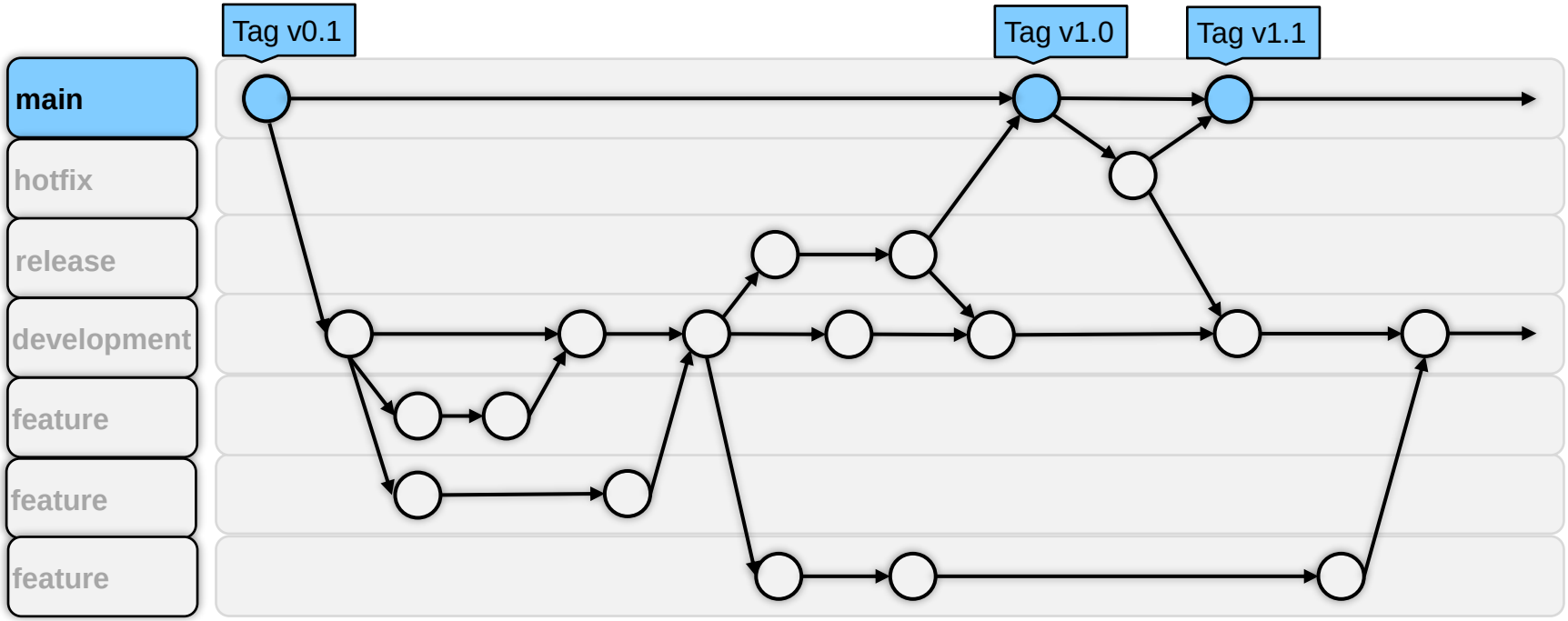
- Enthält ausschließlich offizielle Releases
- Existiert fortlaufend im Projekt
- Keine direkten Commits, nur Merge Requests



Gitflow-Workflow – Aufbau

main Branch

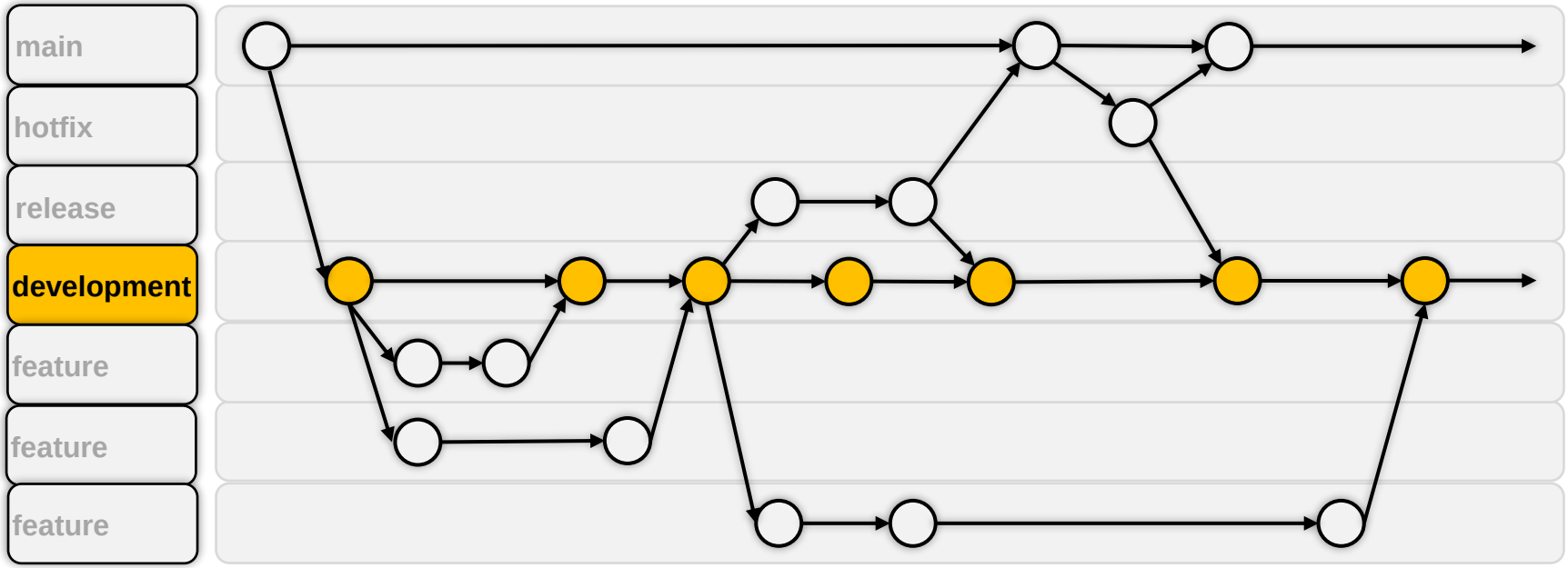
- Commits oft mit Release Tags versehen
- Bietet Überblick über alle Releases
  - Ermöglicht Auschecken älterer Versionen
  - Fehlerbehebung an alten Versionen möglich



Gitflow-Workflow – Aufbau

development Branch

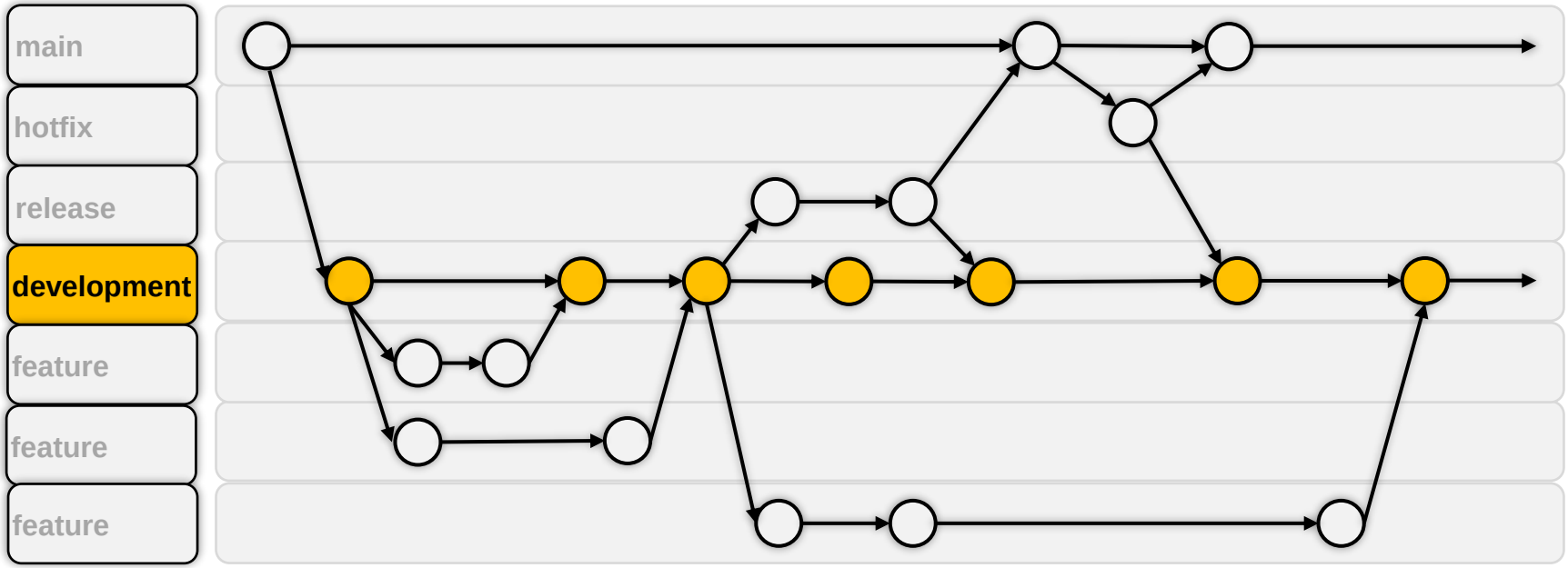
- aka **dev,** aka **develop**
- Neben **main** der zweite dauerhaft existierende Branch
- Aktueller Entwicklungsstand
- Sollte immer eine vollfunktionsfähige Version enthalten



Gitflow-Workflow – Aufbau

development Branch

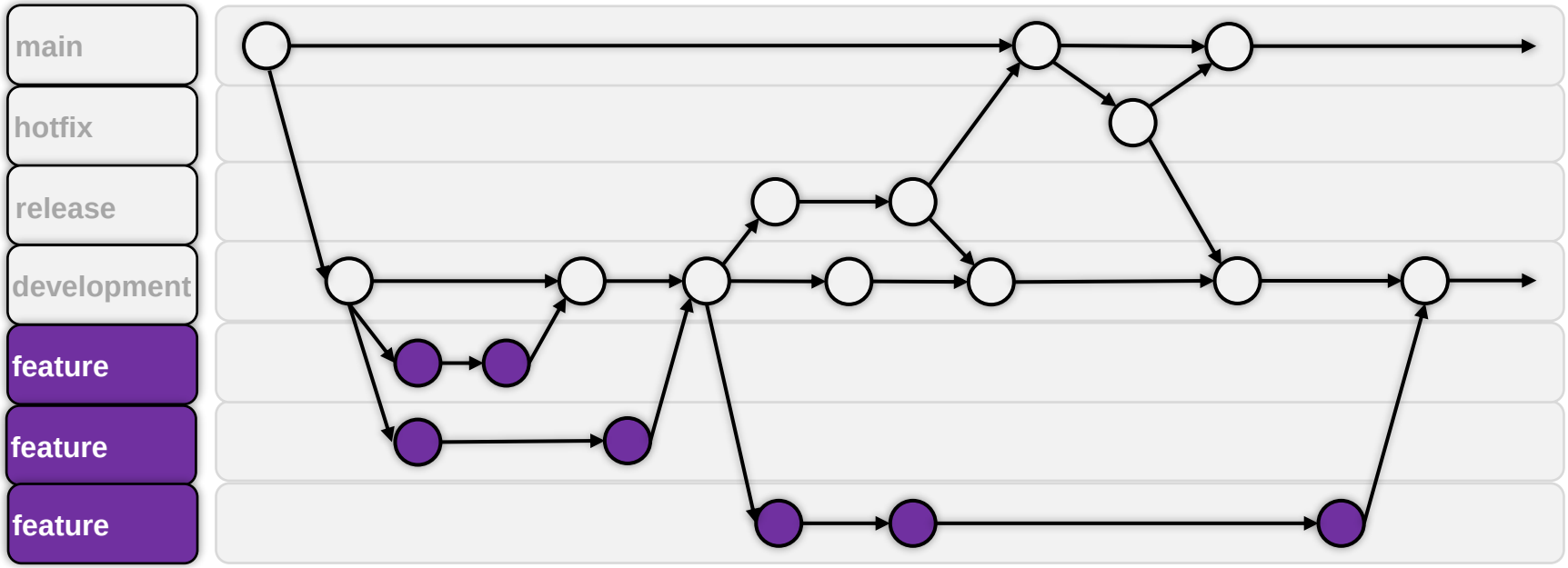
- Integration von Features und Bugfixes
- Direkte Commits unüblich und zu vermieden
- Enthält die komplette Historie des Projektes



Gitflow-Workflow – Aufbau

Feature Branches

- Zweigen von **dev** ab
- Entwickler arbeiten isoliert an Features
- Experimente, PoC, usw. möglich

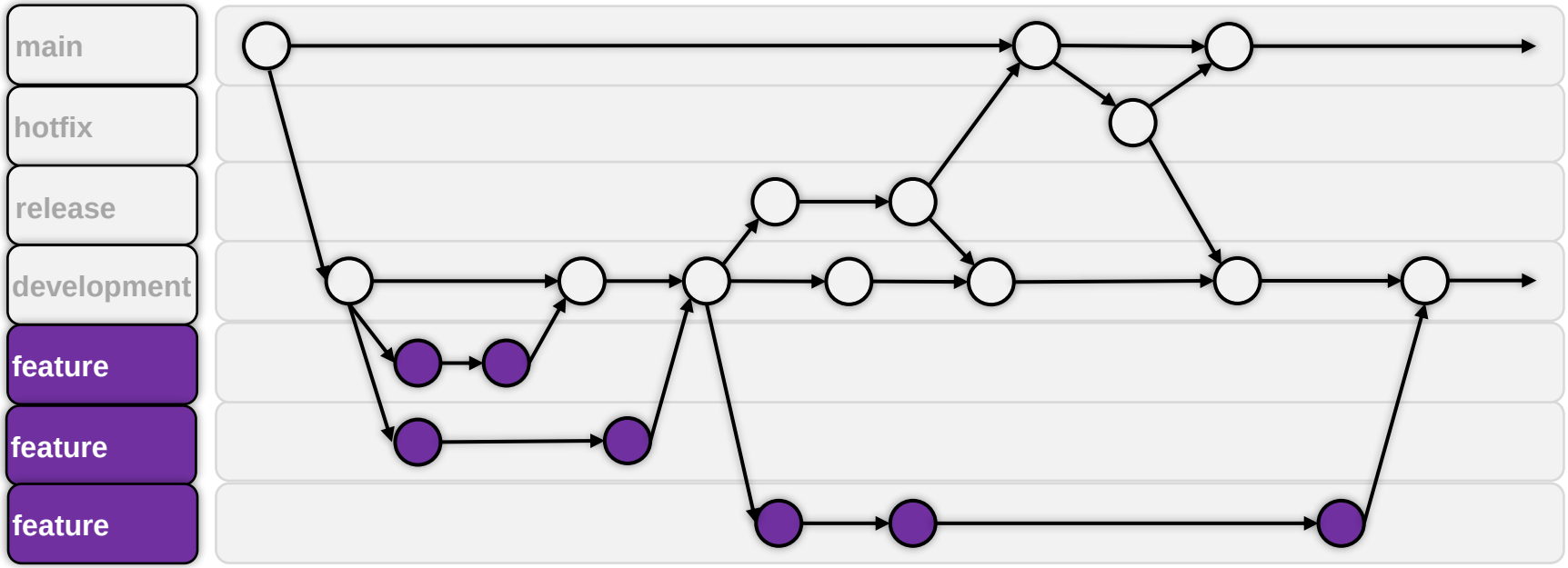


Gitflow-Workflow – Aufbau

Feature Branches

- Langlebigere Branches als bei anderen Workflows
  - Können Integration von Features erschweren
  - Entwickler für aktuellen Stand zuständig

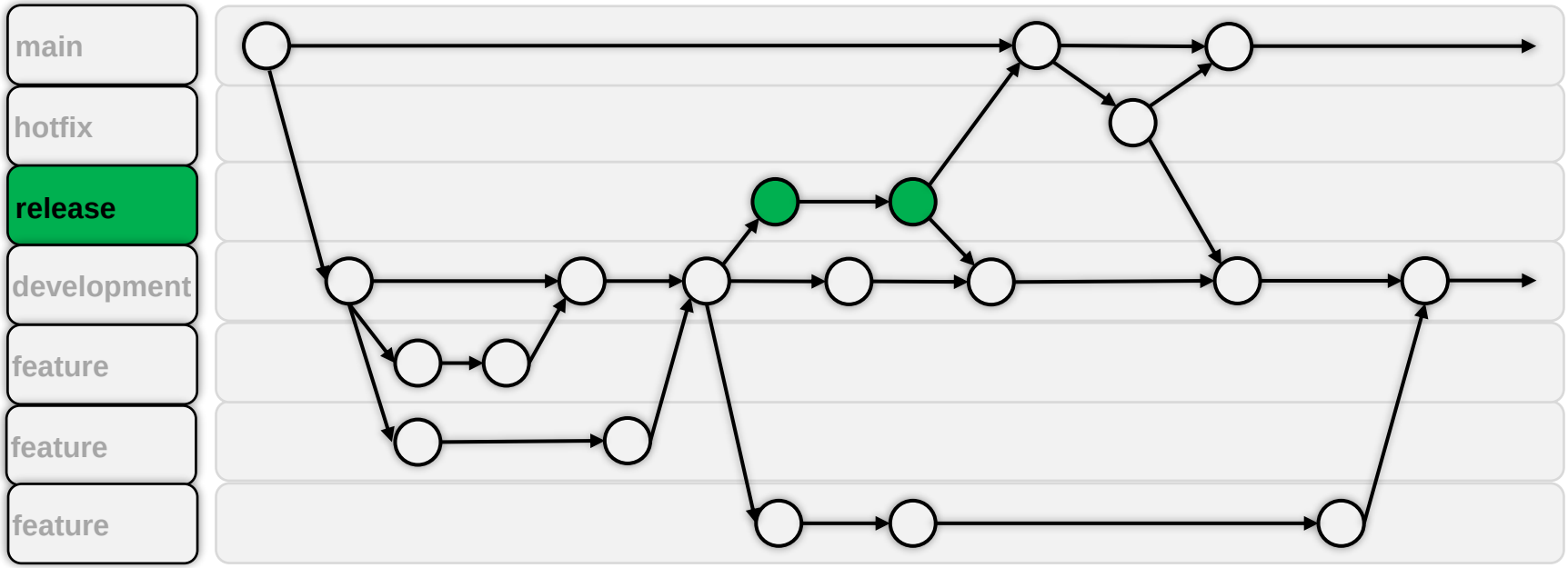
    ➔ Möglichst Feature regelmäßig auf development rebasen



Gitflow-Workflow – Aufbau

Release Branches

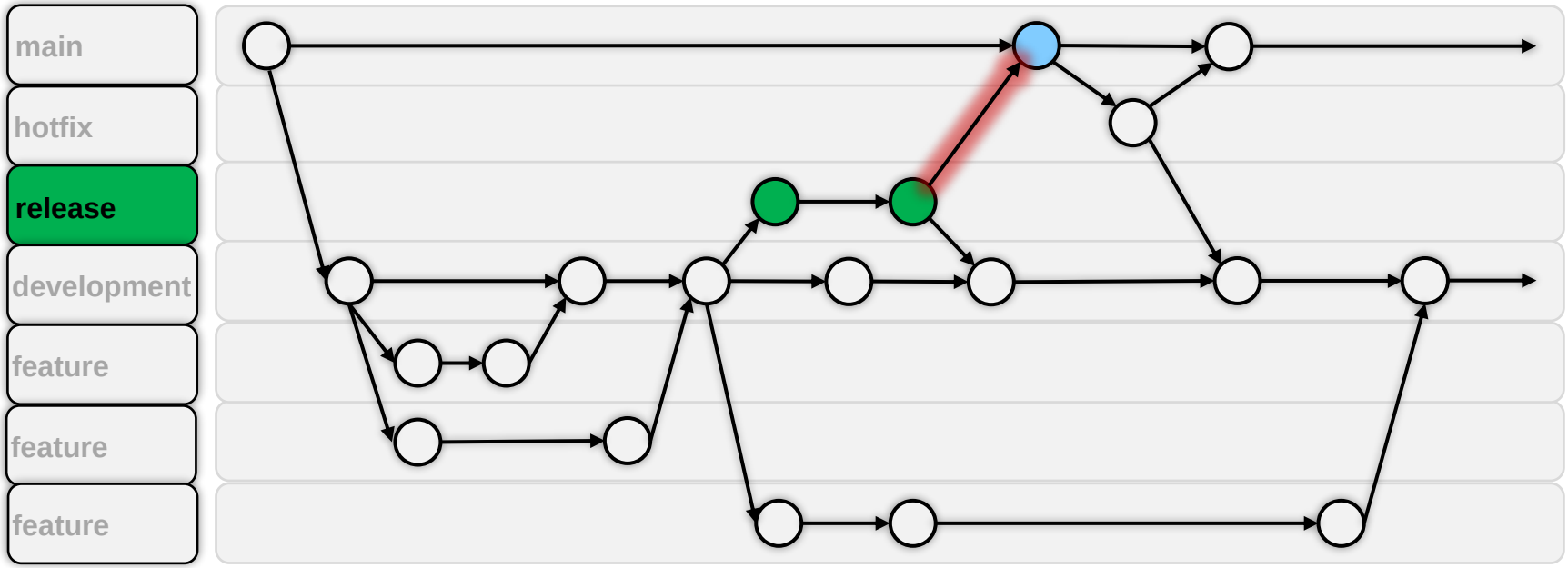
- Werden für die Release Phase verwendet
- Zweigen von **dev** ab, wenn dieser für ein Release bereit ist
- **Keine** neuen Features!



Gitflow-Workflow – Aufbau

Release Branches

- Nur Bugfixes und Anpassungen für das Release
- Nach Abschluss in **main** mergen
- Neues Release → neuer Branch

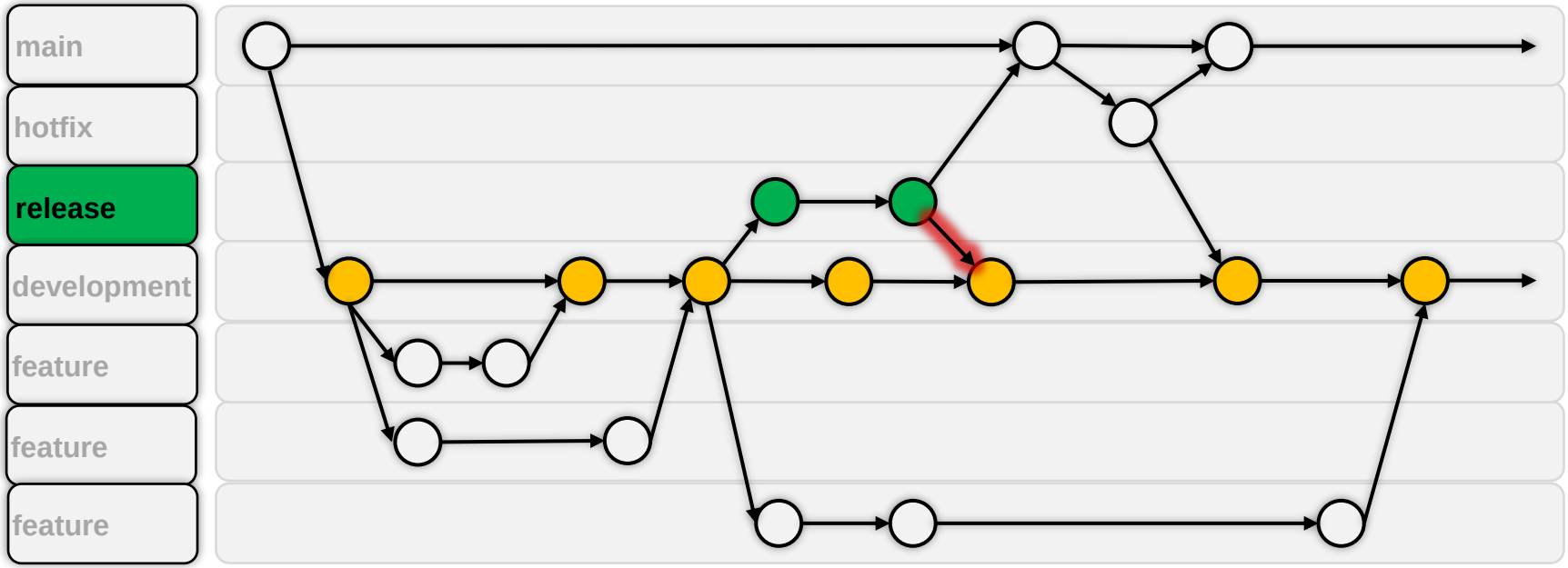


Gitflow-Workflow – Aufbau

anderScore trust in competence

Release Branches

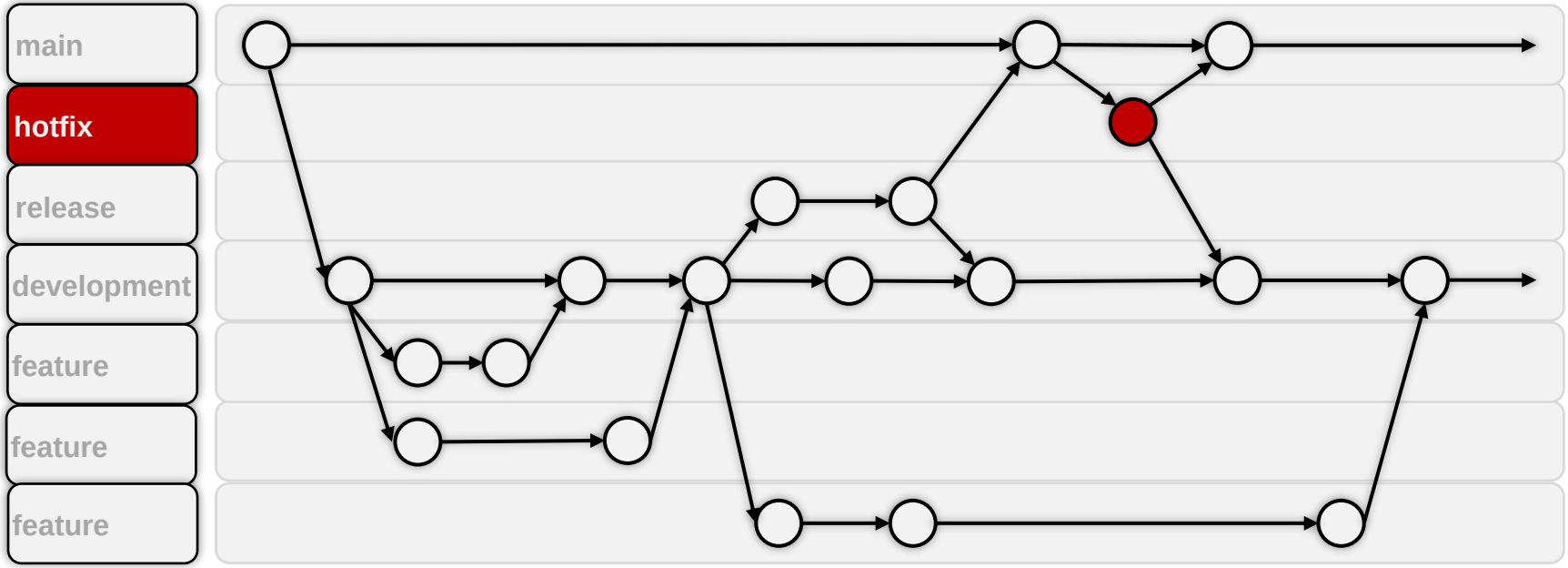
- **Wichtig**: Bugfixes und Anpassungen nach Release nach **dev** mergen
  - Anpassungen ansonsten nur in **main** enthalten
  - Regressionen dann vorprogrammiert!



Gitflow-Workflow – Aufbau

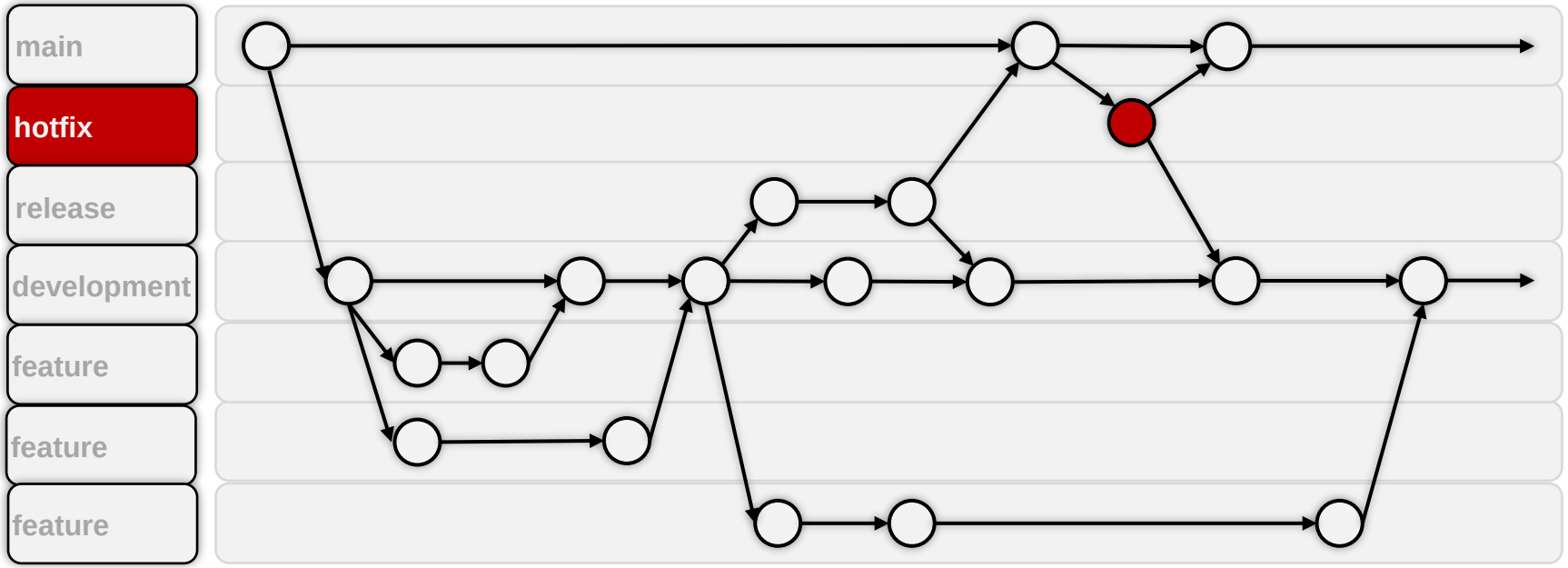
Hotfix Branches

- Hotfixes der Releases
- Zweigen vom **main** ab
- Hotfix fertig → zurück nach **main**
- Sehr dringende Bugfixes, **keine** Features



Hotfix Branches

- **Wichtig**: Bugfixes nach **dev** (analog zu **main**)



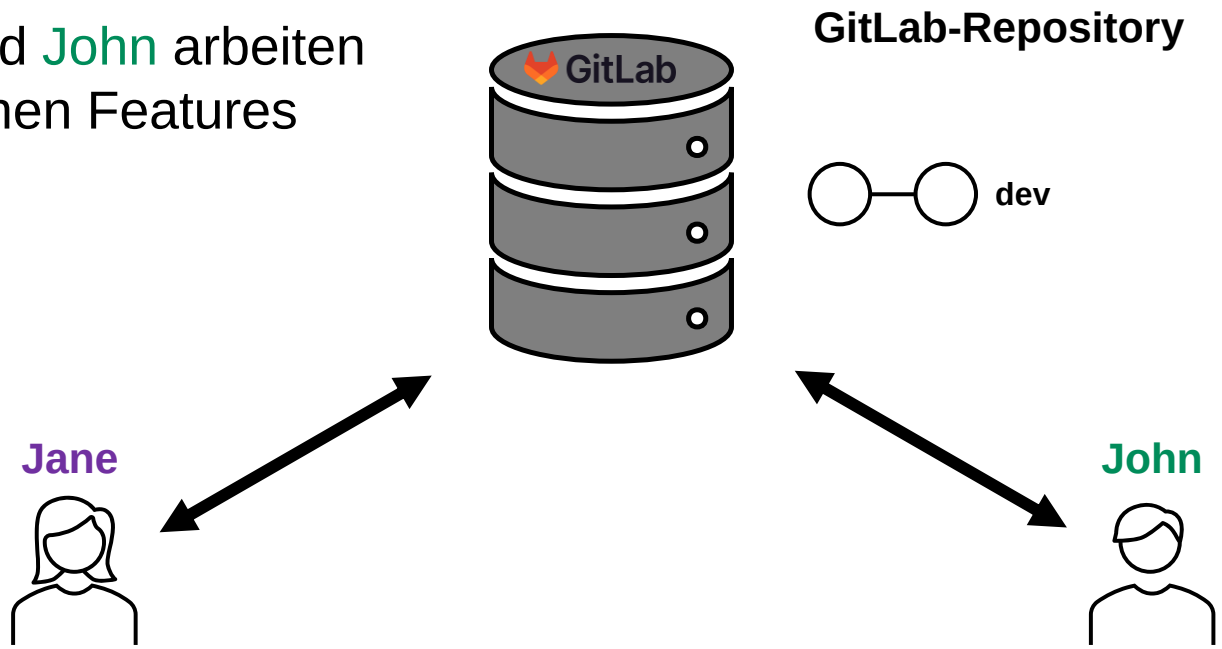
Git

Arbeiten im Gitflow-Workflow

Arbeiten im Gitflow-Workflow

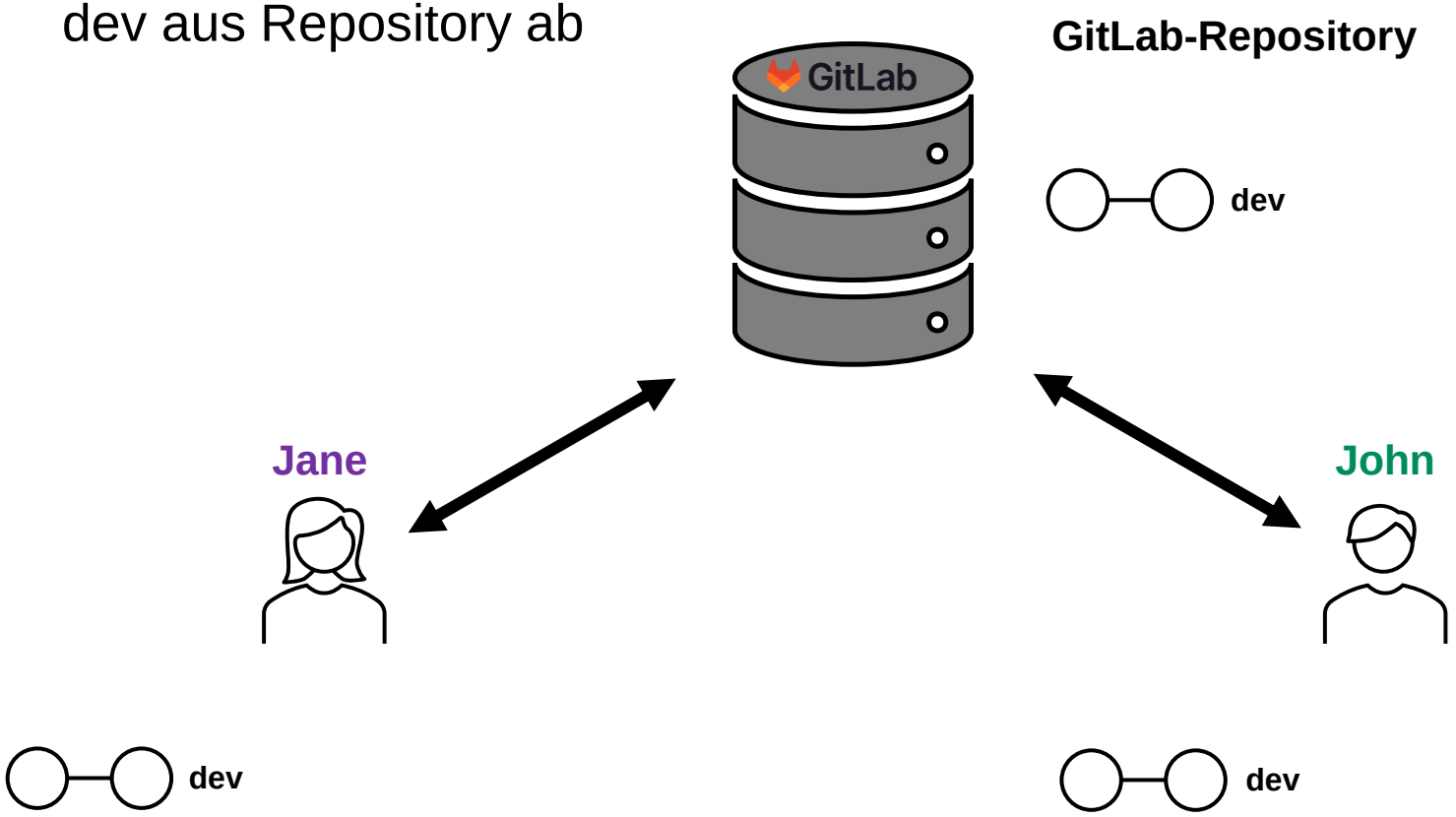
Beispielszenario

- Jane und John arbeiten an eigenen Features



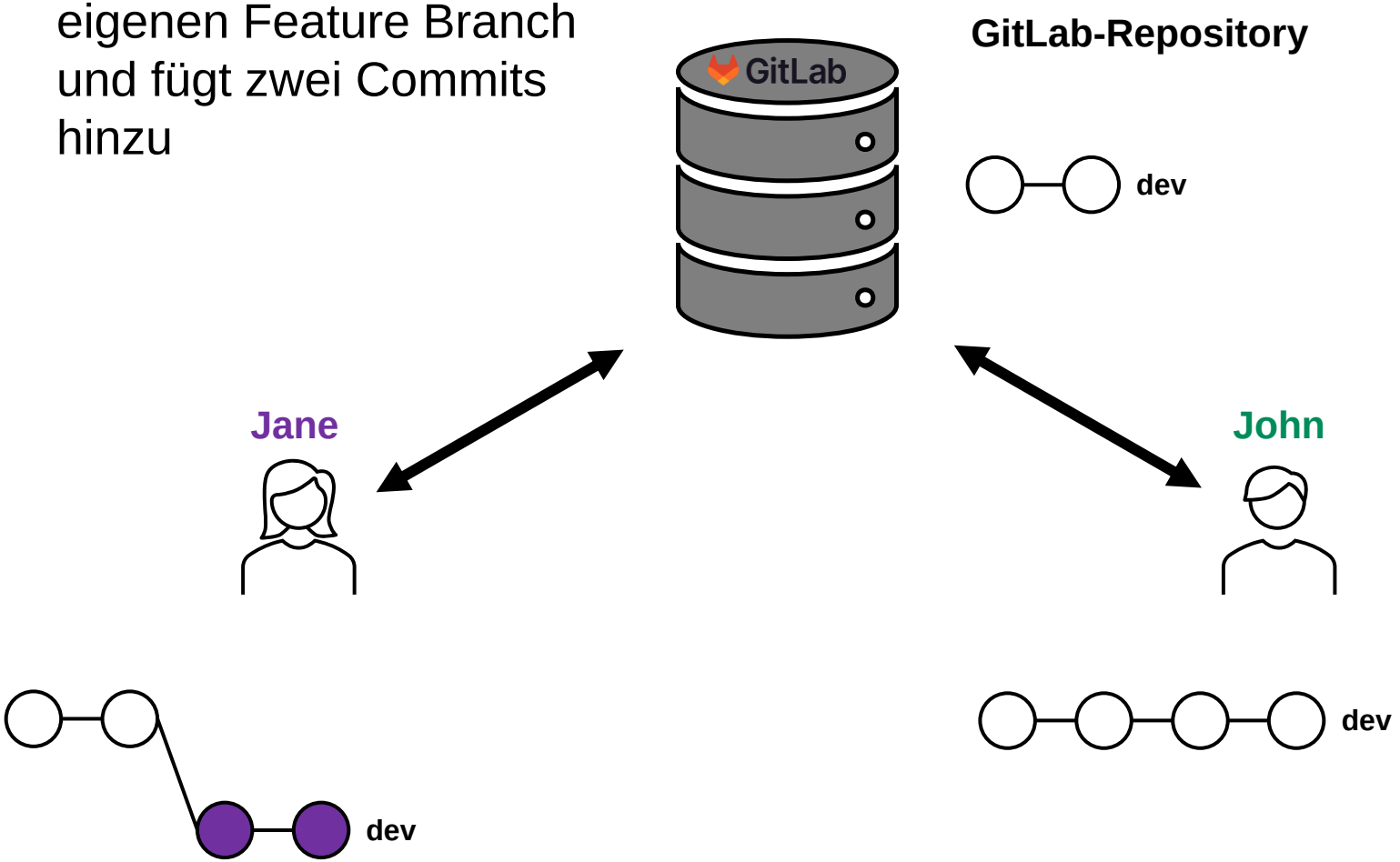
Arbeiten im Gitflow-Workflow

- Jane und John rufen dev aus Repository ab



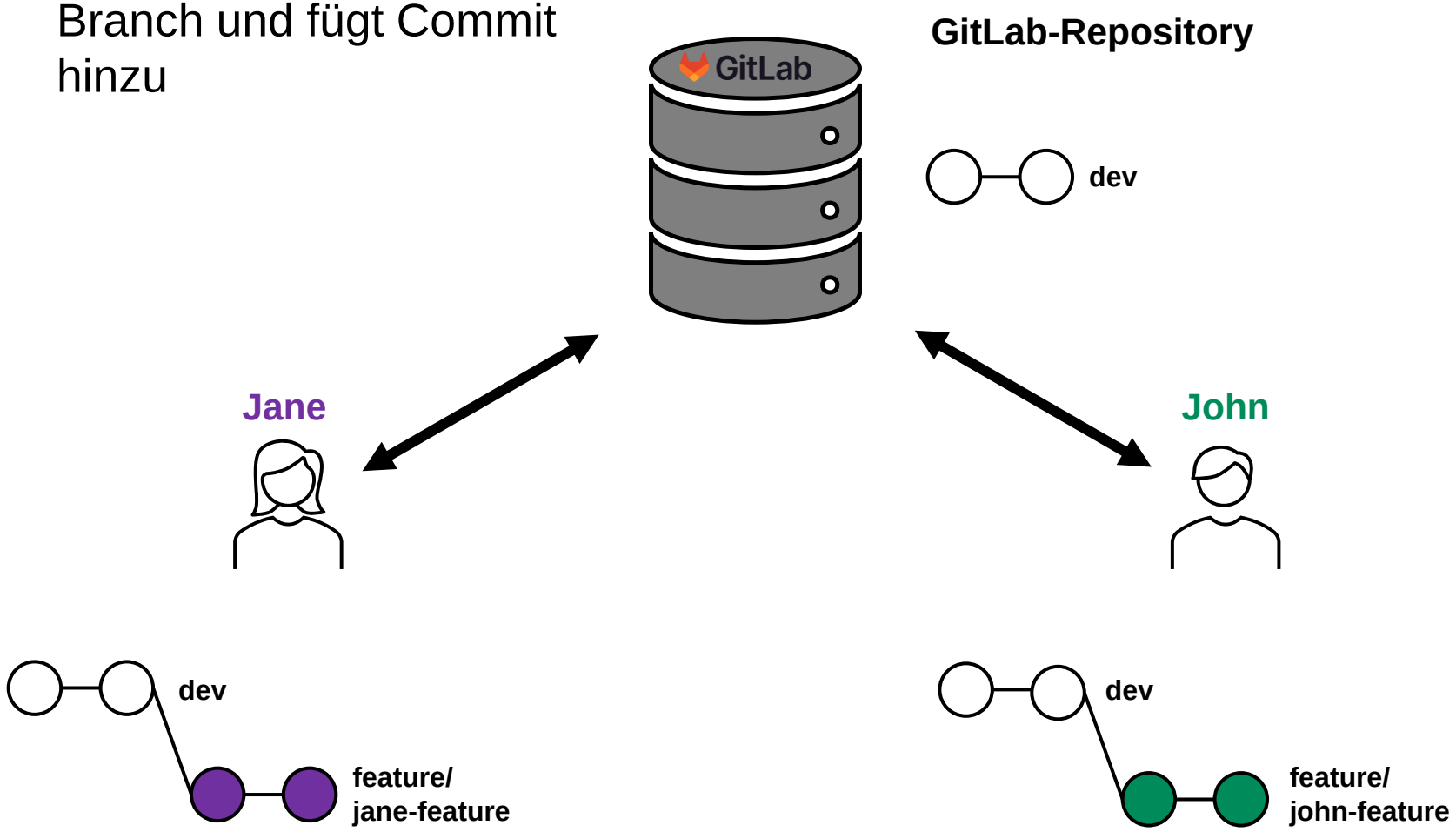
Arbeiten im Gitflow-Workflow

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



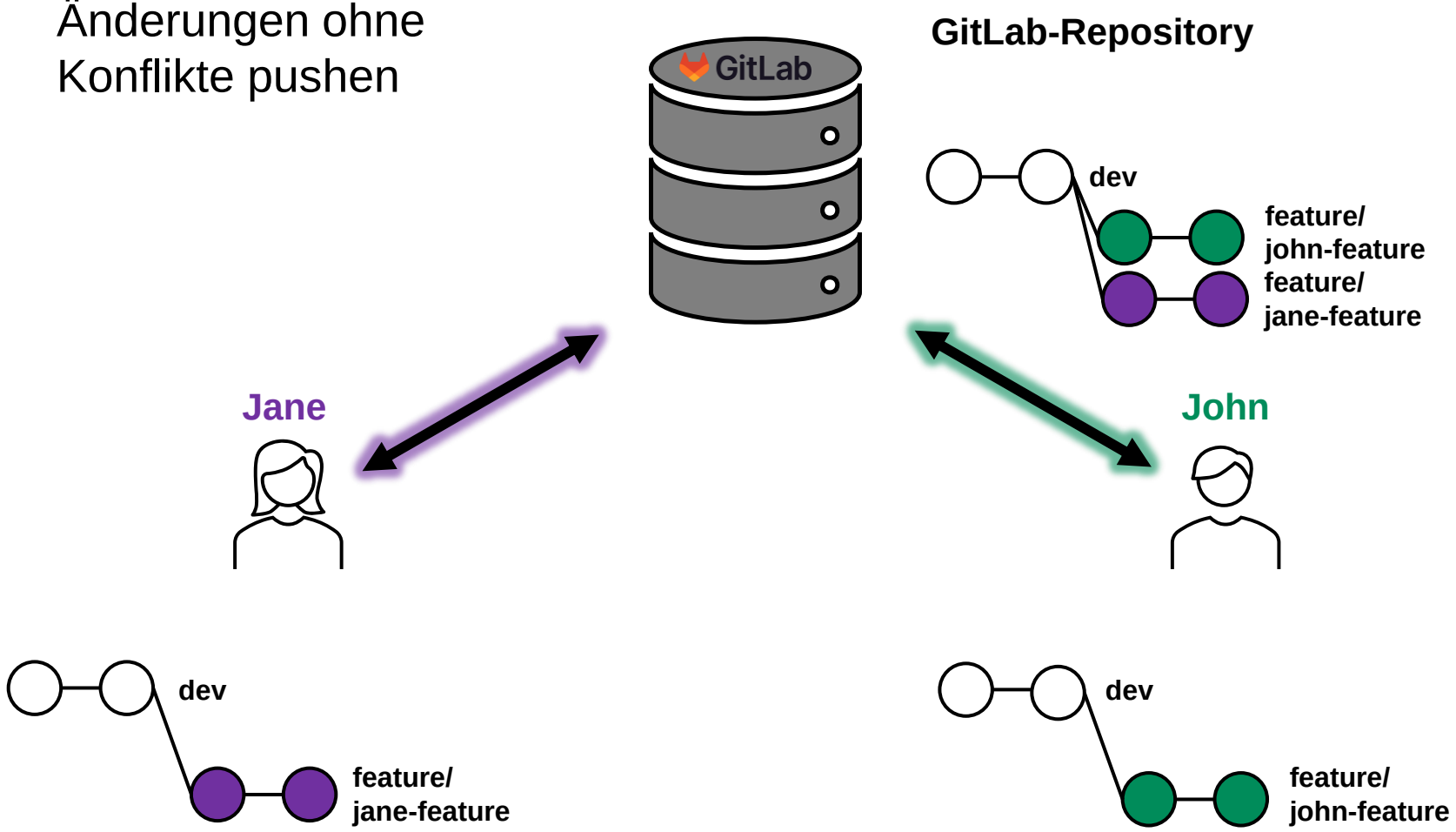
Arbeiten im Gitflow-Workflow

- John erstellt eigenen Branch und fügt Commit hinzu



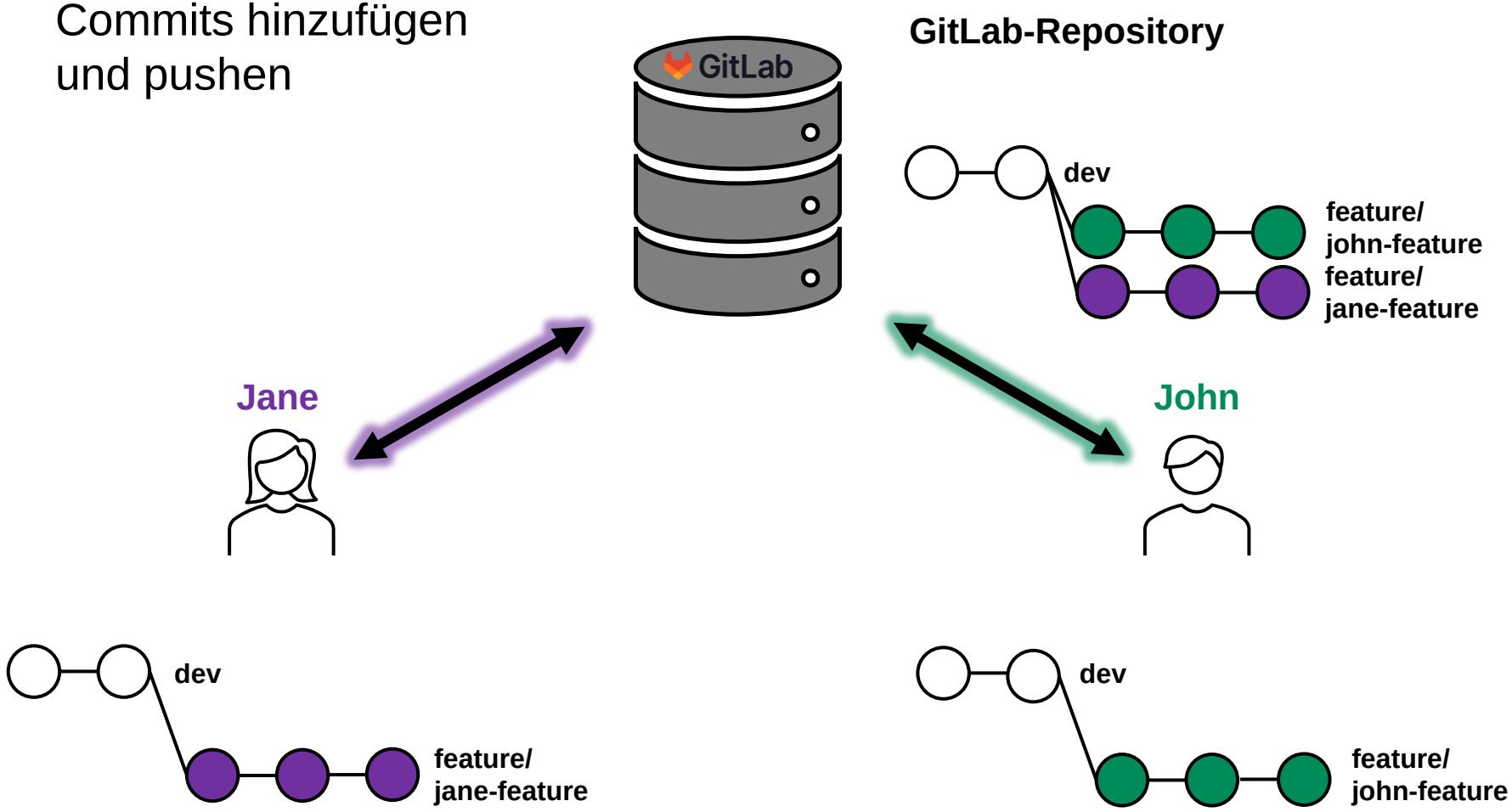
Arbeiten im Gitflow-Workflow

- Beide können Änderungen ohne Konflikte pushen



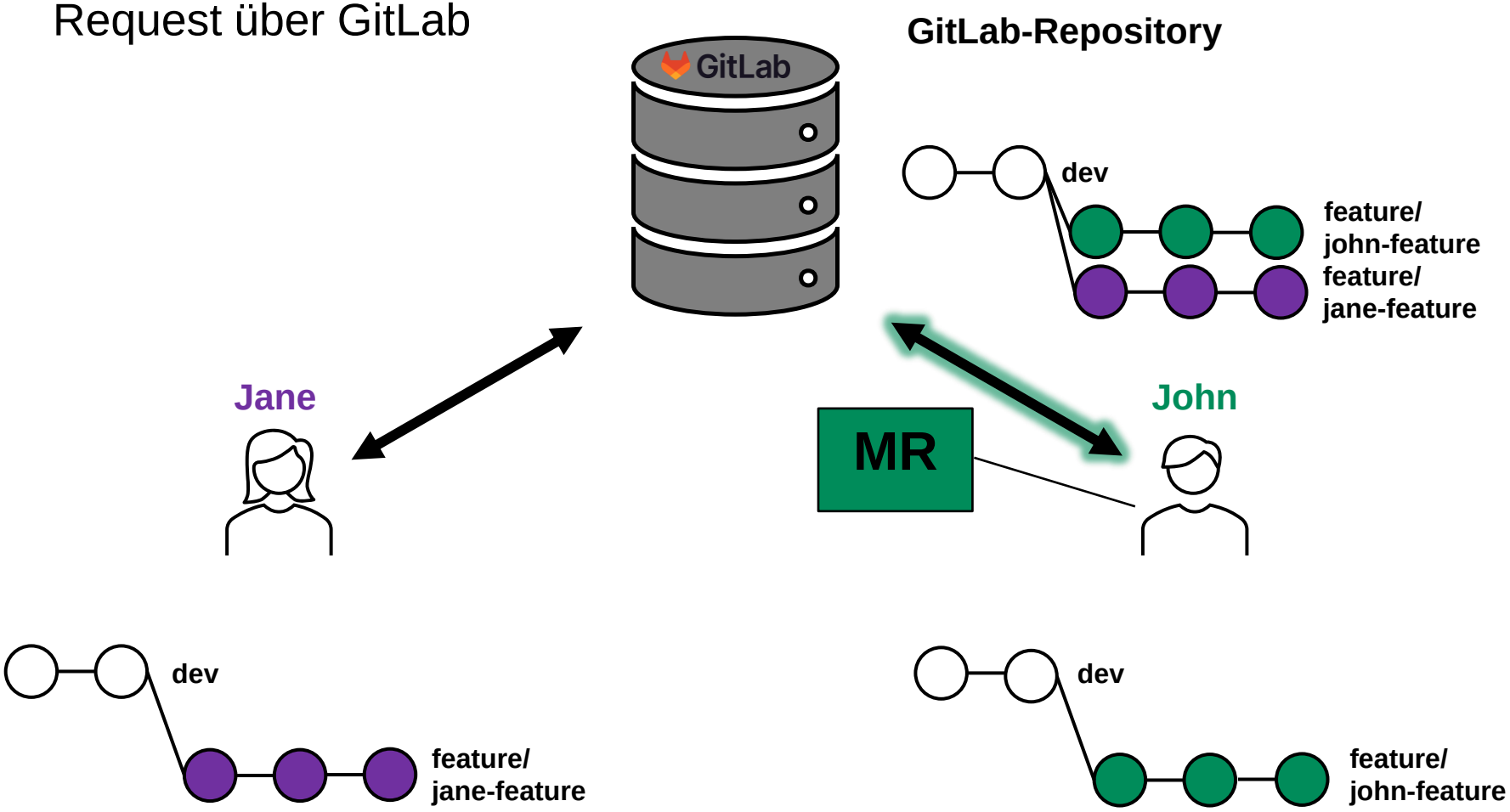
Arbeiten im Gitflow-Workflow

- Beide können beliebig Commits hinzufügen und pushen



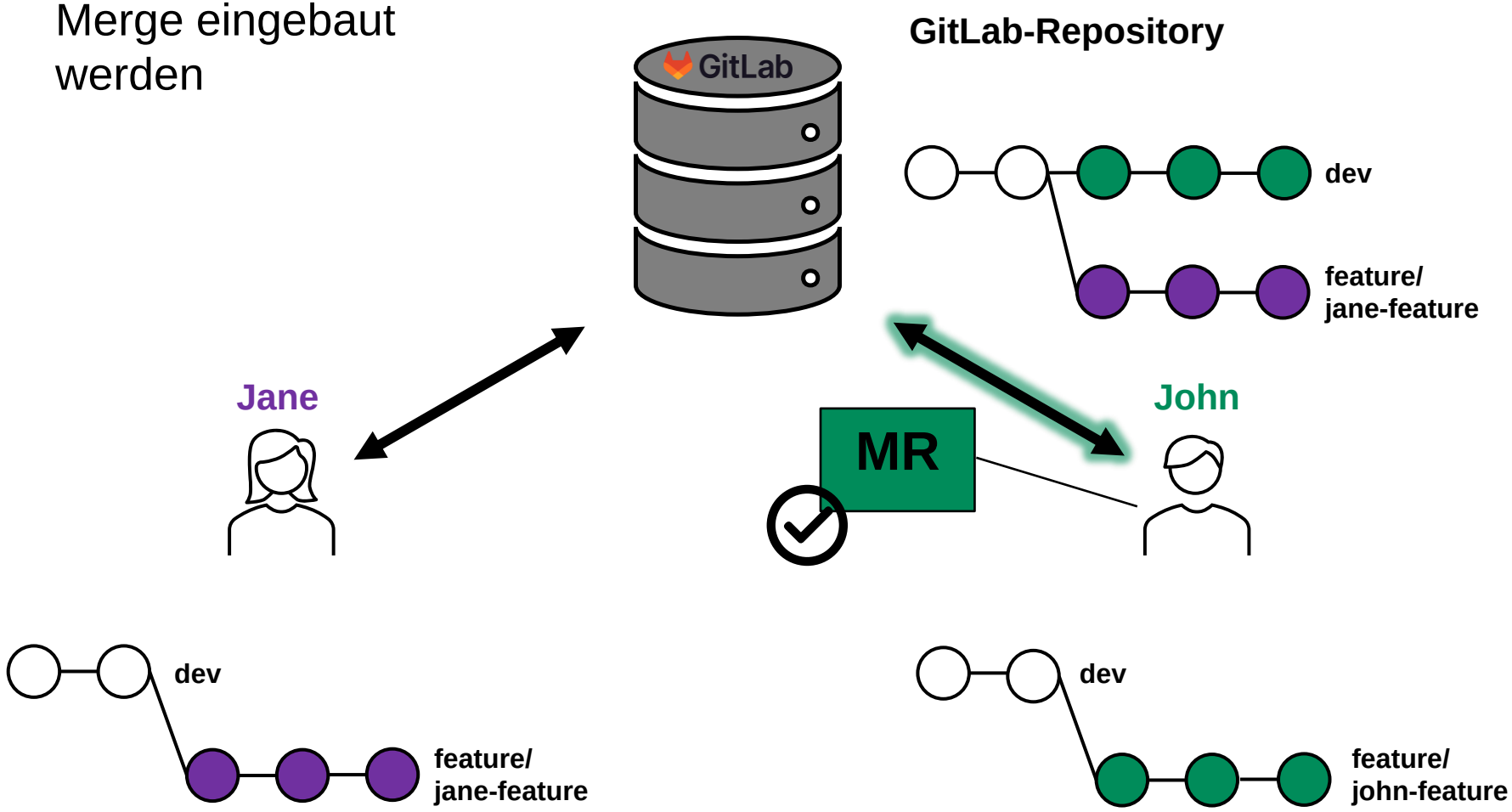
Arbeiten im Gitflow-Workflow

- John erstellt Merge Request über GitLab



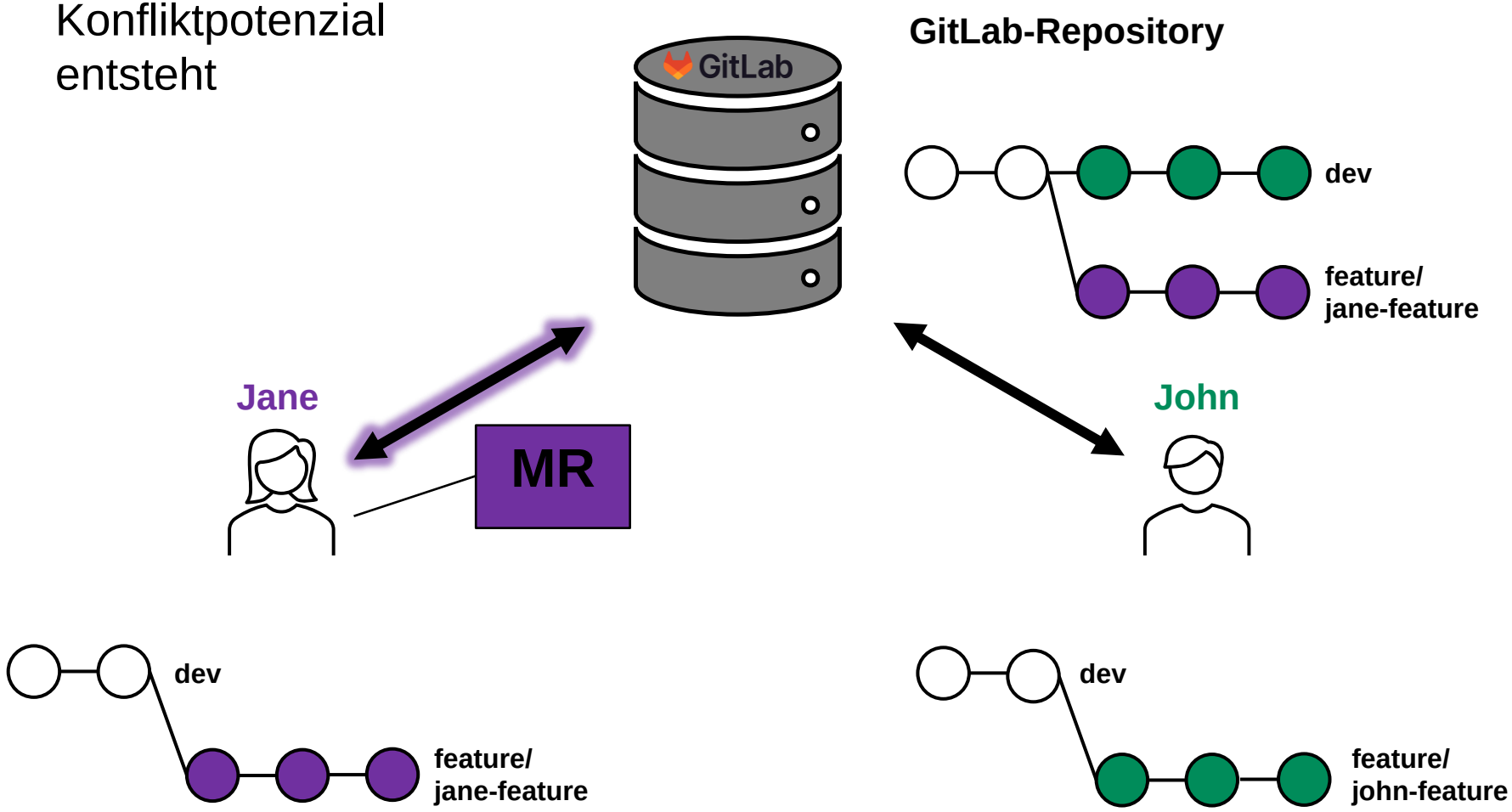
Arbeiten im Gitflow-Workflow

- Johns MR kann per FF-Merge eingebaut werden



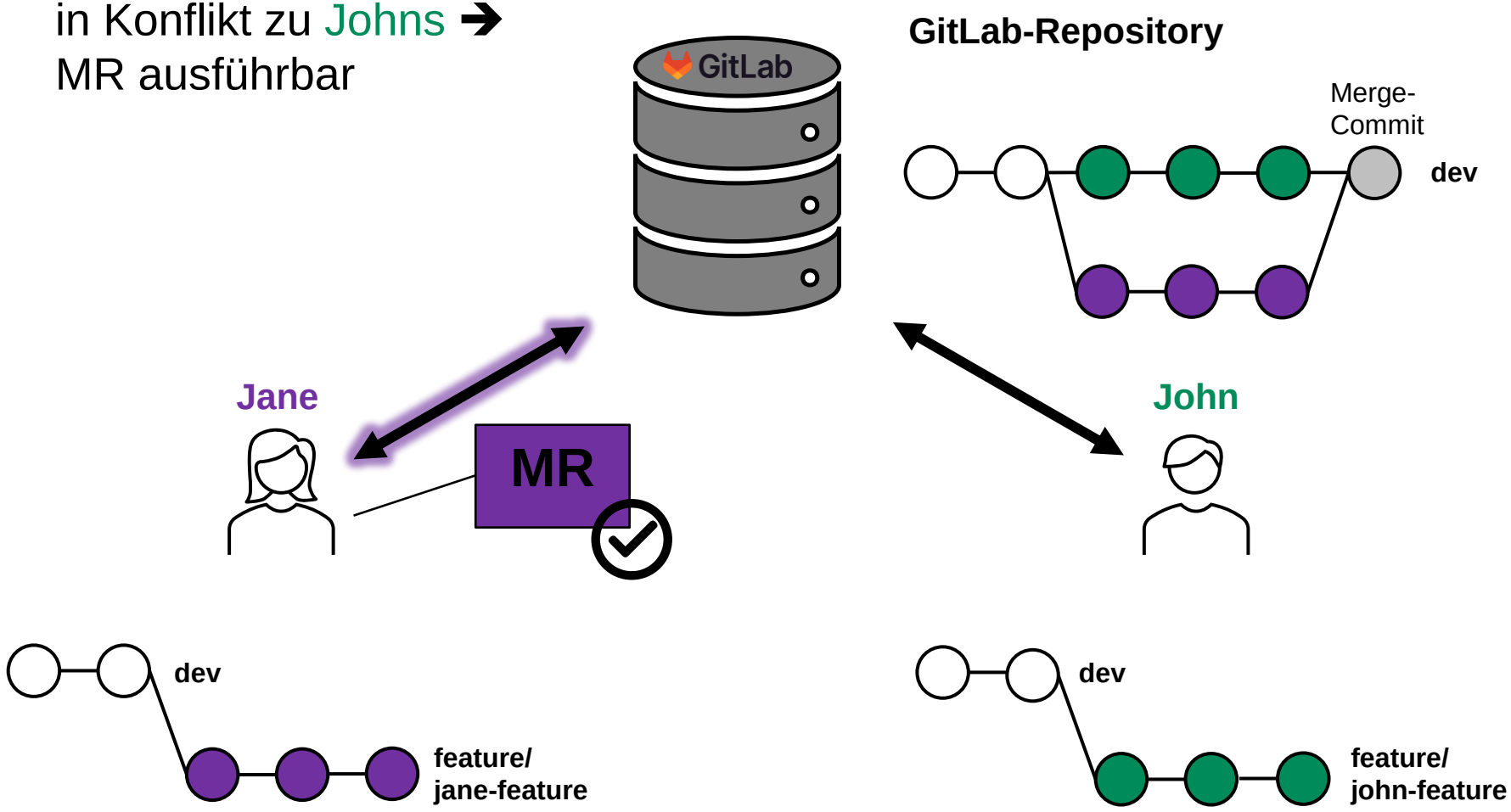
Arbeiten im Gitflow-Workflow

- Jane stellt MR, erstes Konfliktpotenzial entsteht



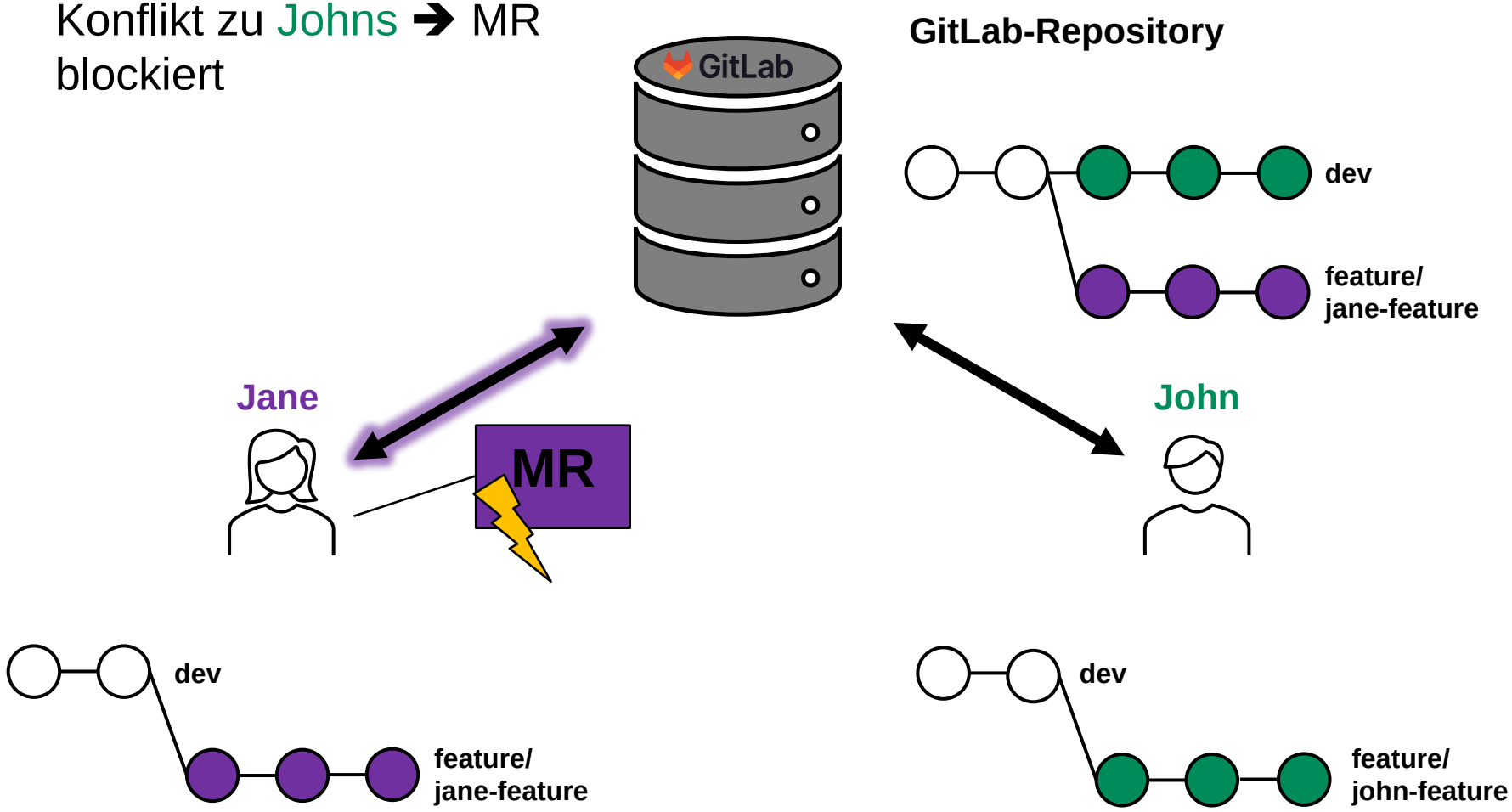
Arbeiten im Gitflow-Workflow

- Janes Änderungen nicht in Konflikt zu Johns ➔ MR ausführbar



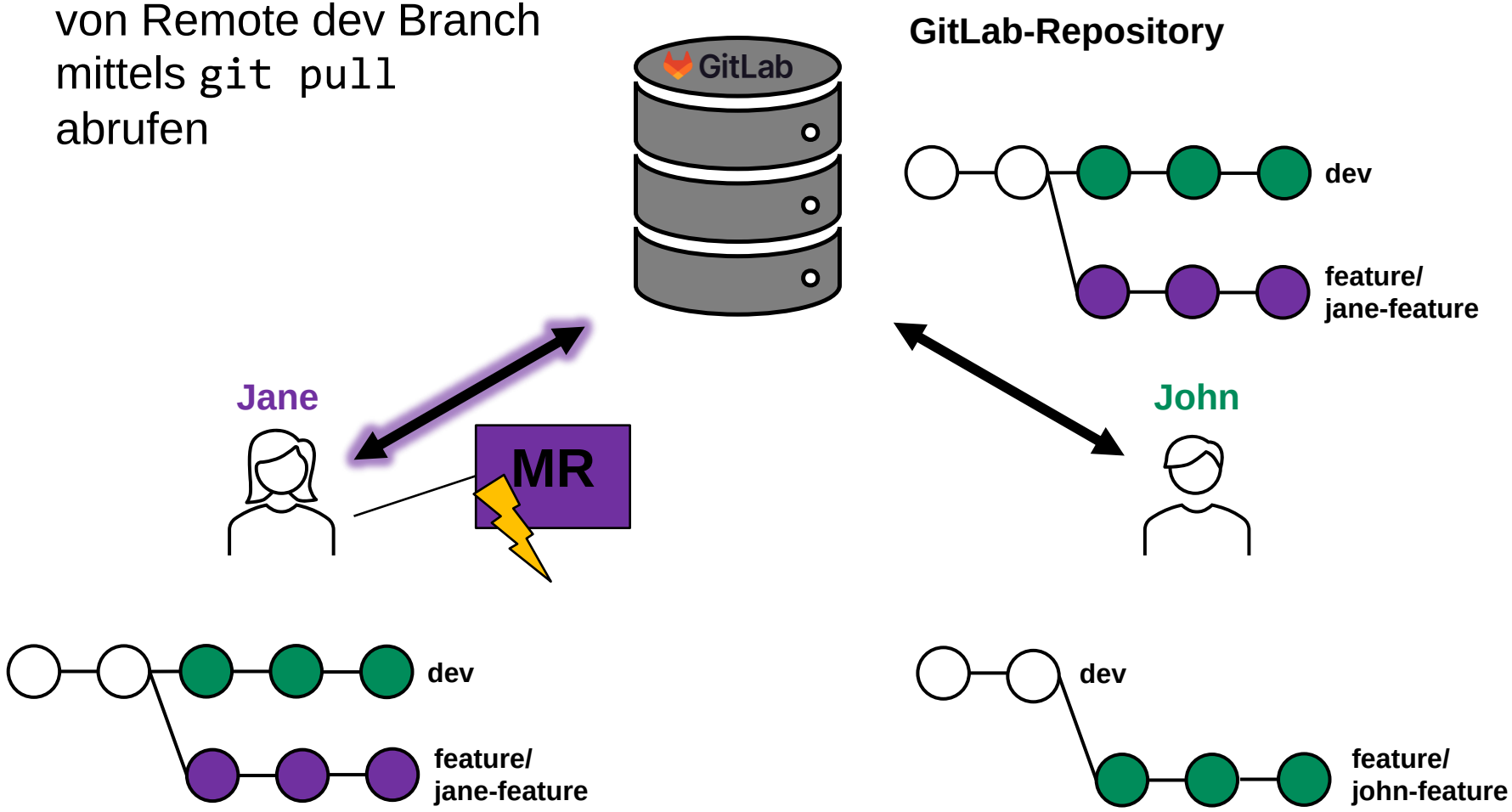
Arbeiten im Gitflow-Workflow

- Janes Änderungen in Konflikt zu Johns ➔ MR blockiert



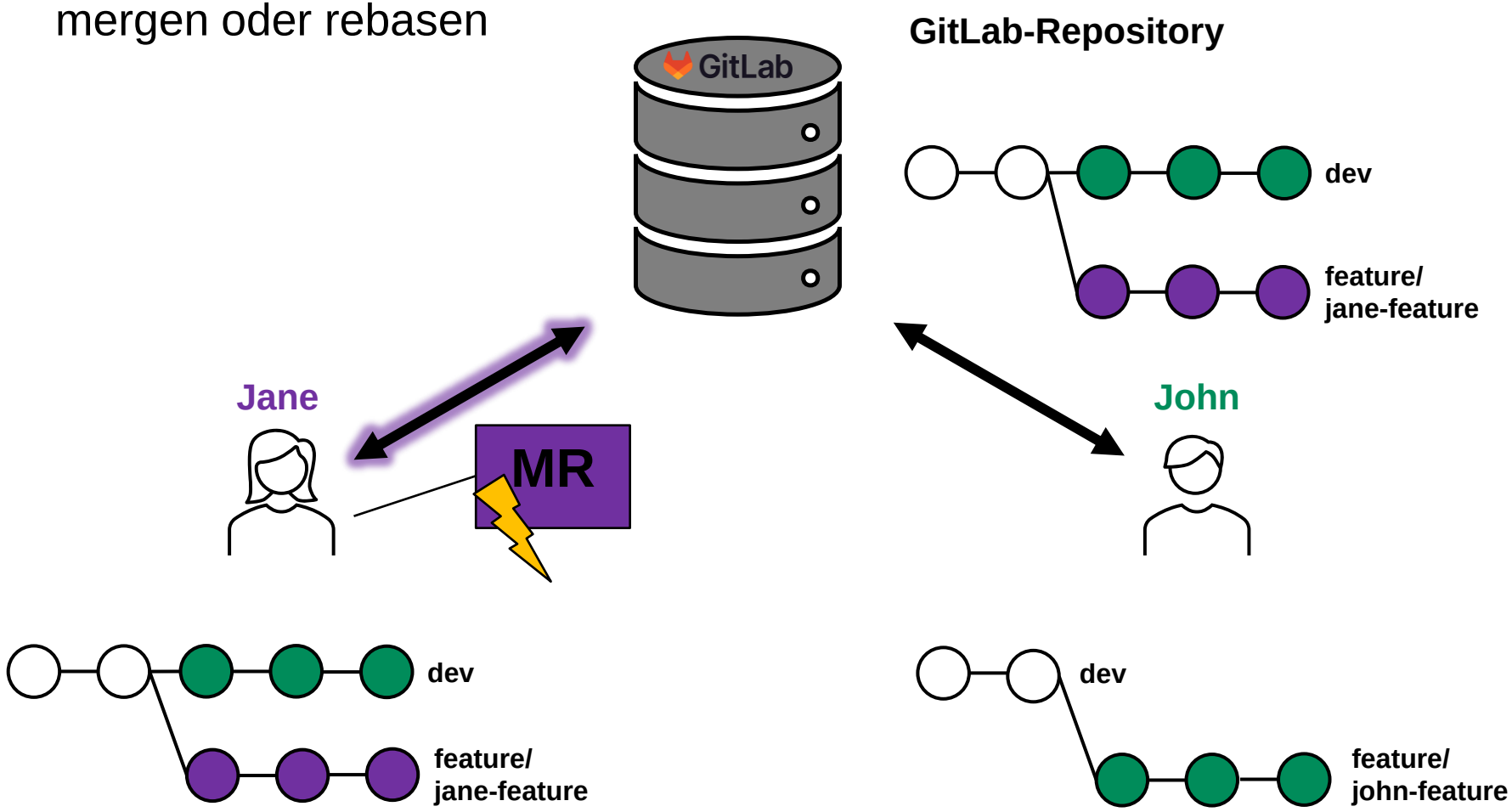
Arbeiten im Gitflow-Workflow

- Jane muss Änderungen von Remote dev Branch mittels `git pull` abrufen



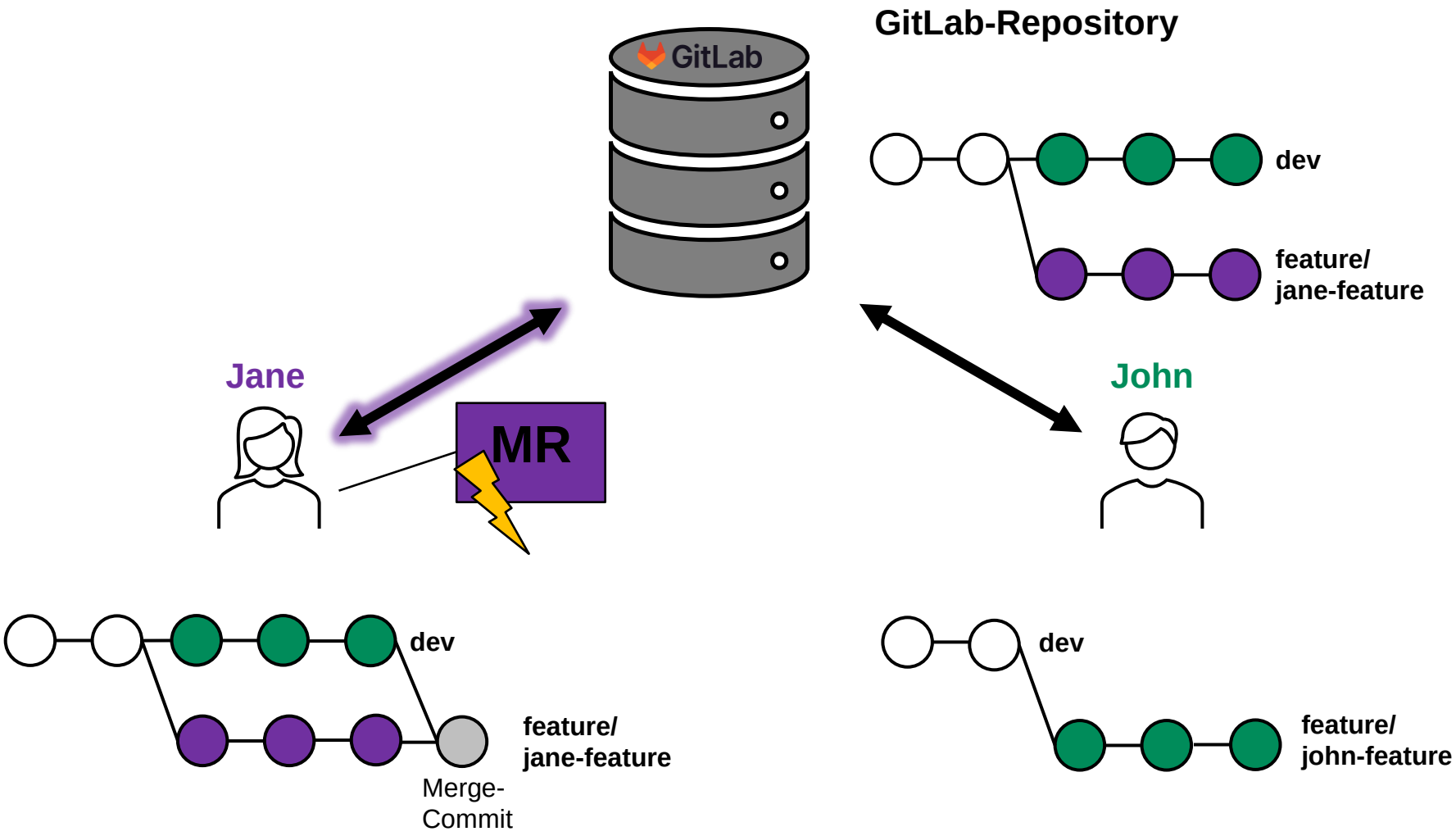
Arbeiten im Gitflow-Workflow

- Jane ihren Branch mergen oder rebasen



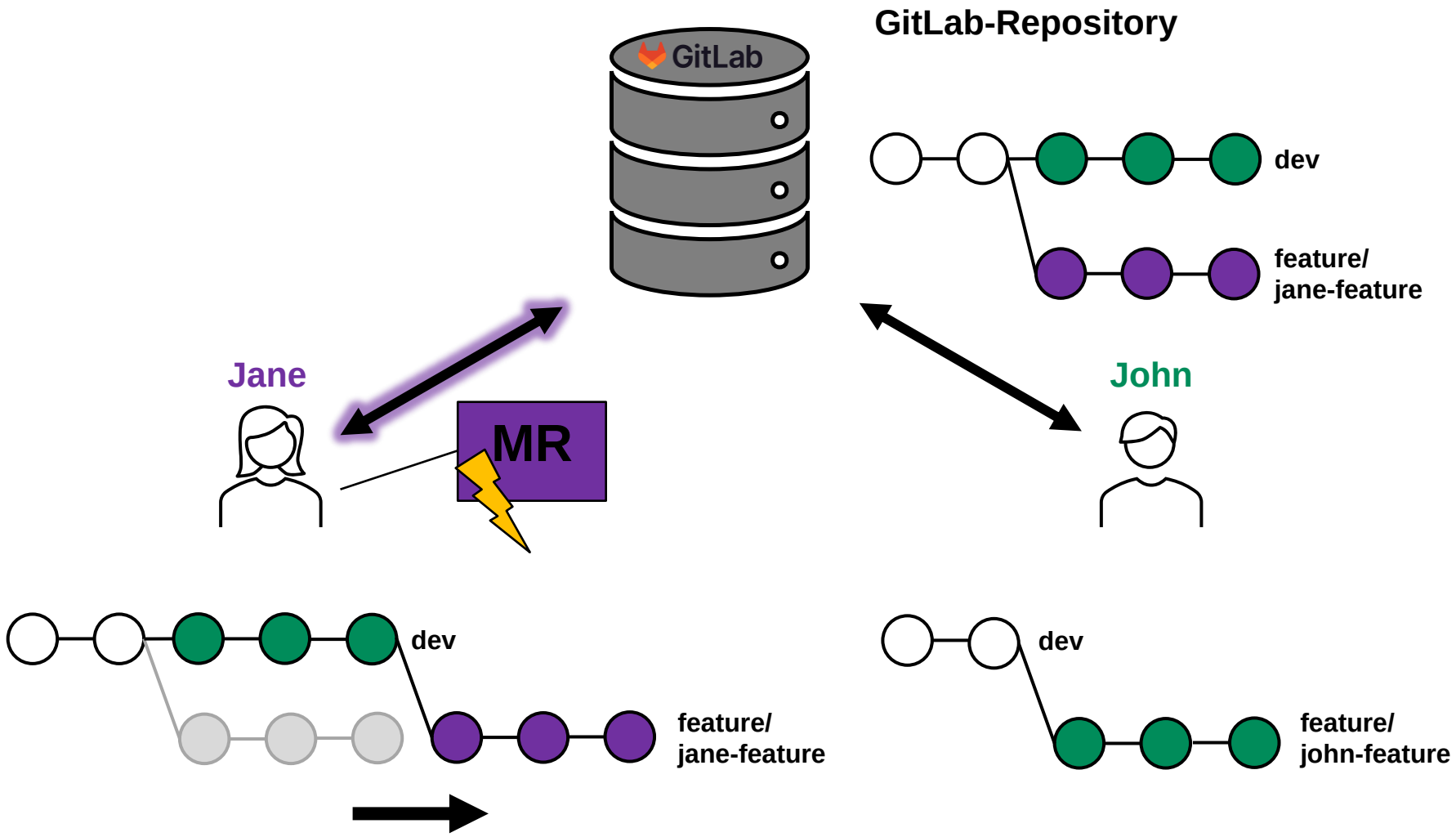
Arbeiten im Gitflow-Workflow

Merge



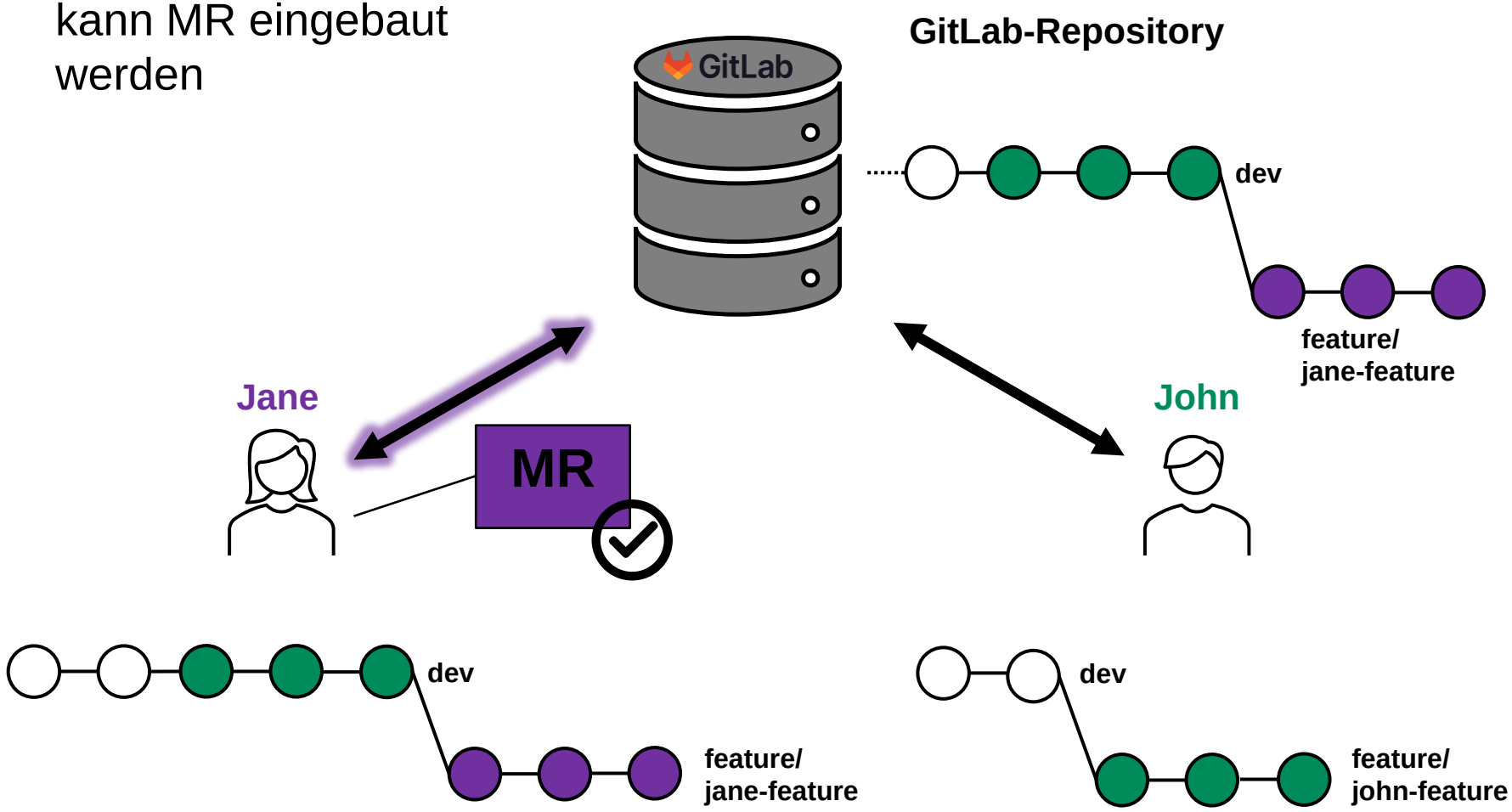
Arbeiten im Gitflow-Workflow

Rebase



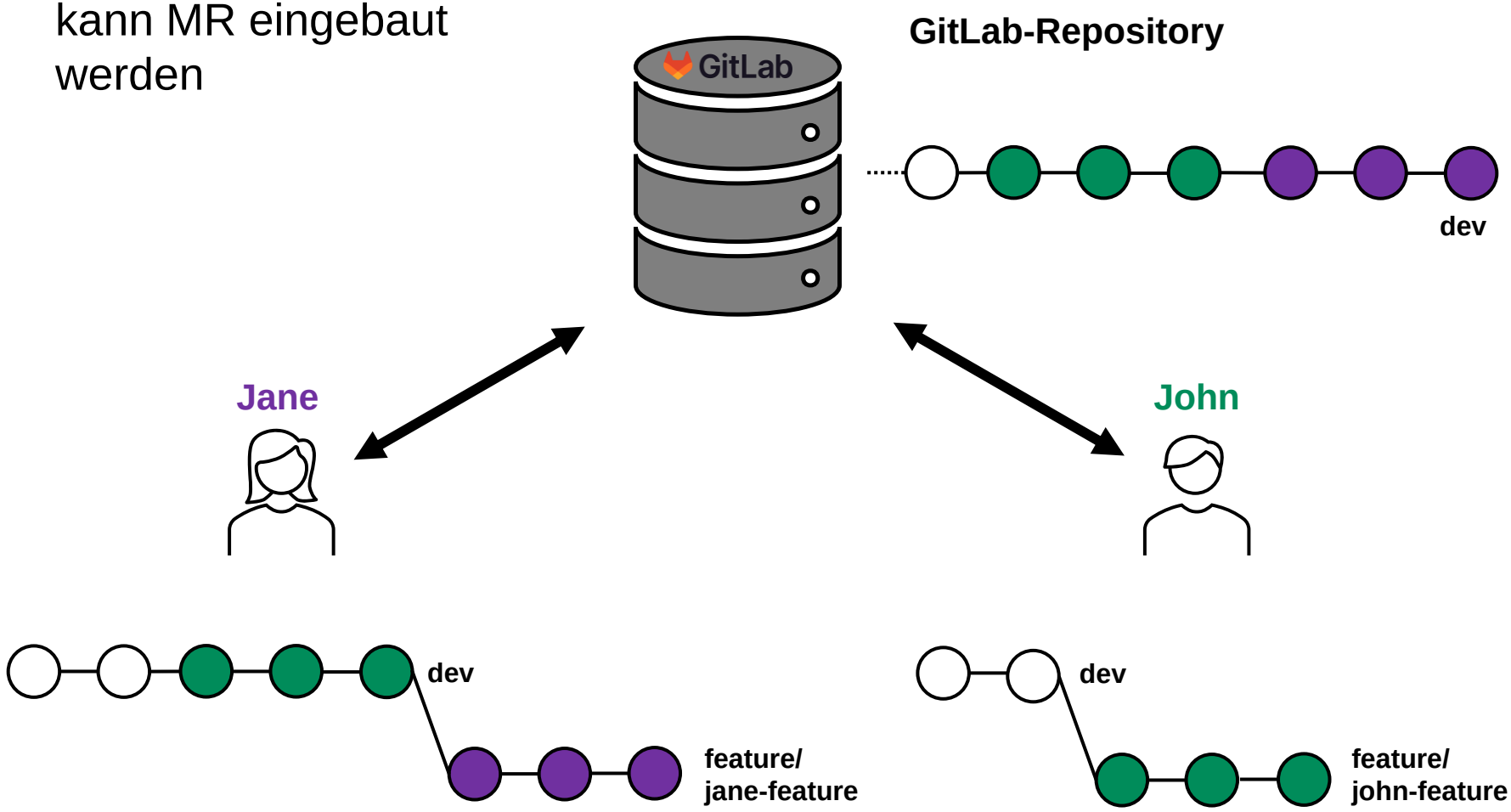
Arbeiten im Gitflow-Workflow

- Nach `git push -f` kann MR eingebaut werden



Arbeiten im Gitflow-Workflow

- Nach `git push -f` kann MR eingebaut werden



Gitflow-Workflow

Use Cases und Fazit

Gitflow-Workflow – Use Cases

- Workflow stammt von 2010 und gilt teilweise als überholt
- Vorteile
  - Klare, strukturierte Aufteilung
  - Parallele Entwicklung vereinfacht
  - Robustere Releases durch dedizierte Branches
  - Bessere Versionshistorie
- Nachteile
  - Langlebige Branches erhöhen Konfliktpotenzial
  - Erschwert Continuous Deployment
  - Continuous Integration möglich, aber weniger effektiv
  - Zusätzlicher Overhead durch Branches
- Geeignet für
  - Größere Teams oder komplexe Projekte
  - Projekte ohne hochfrequente Releases
  - Entwicklung mehrerer zeitgleich betriebener Versionsstände

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/