dataport

anderScore
trust in competence

Tag 1: Einführung in Git und GitLab, Git-Workflow im Team

17.06.2024, Daniel Krämer & Malte Fischer

© Copyright 2024 anderScore GmbH

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

Workflows

Git-Workflow

Inhalt

- Was sind Git-Workflows?
- Zentraler Git-Workflow
  - Konzept
  - Ablauf
- Andere Git-Workflows

Git-Workflow

anderScore trust in competence

Was sind Git-Workflows?

- Workflows sind Empfehlungen und Strategien zum Arbeiten mit Git, speziell im Remote Kontext
- Sollen im Team für eine konsistente und effektive Nutzung von Git und Git-Plattformen wie GitLab sorgen
- Workflows sind eher als Empfehlungen und nicht als absolute Regeln zu verstehen

Git-Workflow

Was sind Git-Workflows?

- Es gibt nicht den einen Git-Workflow
- Git und GitLab bieten durch eine Vielzahl an Konzepten und Feature ganz unterschiedliche Möglichkeiten zum Einsatz
- Daher existieren auch viele Git-Workflows mit unterschiedlichen Konzepten
- Auswahl des passenden Workflows richtet sich nach bestimmten Kriterien
  - Art des Projekts, indem Git verwendet wird
  - Projektgröße und Umfang
  - Teamgröße
  - Teamkultur
- Teammitglieder müssen mit dem gewählten Workflow vertraut sein und diesen produktiv in ihre Arbeit integrieren
- Workflow darf keinen unnötigen Overhead erzeugen

Git-Workflow

anderScore trust in competence

Zentraler Git-Workflow

- Ein zentraler Git-Workflow ist einfacher Workflow
- Im zentralen Ansatz wird nur ein Branch benötigt
  - Häufig **main**, andere Bezeichnungen sind **trunk** oder **master** (veraltet)
  - Sämtliche Änderungen werden hier committet
  - Vermeidet Overhead durch Verwaltung von Branches
- Dadurch einfach und schnell zu verstehen
- Eignet sich gut bei Umstieg von einem CVCS wie bspw. Subversion
- Erfreut sich in verschiedenen Varianten durch gute Kompatibilität zu Continuous Integration und Continuous Deployment (CI/CD) großer Beliebtheit

Git-Workflow

anderScore trust in competence

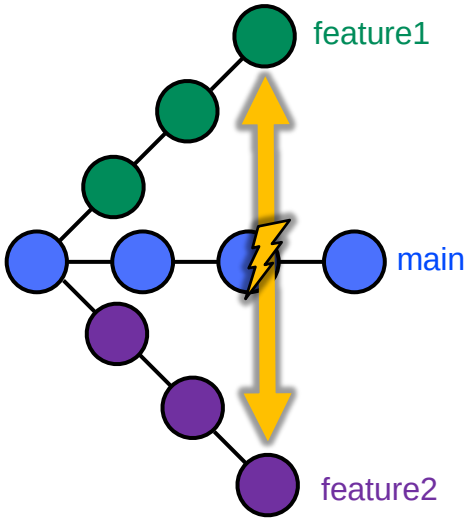
Zentraler Git-Workflow

- Entwickler committen ihre Änderungen direkt auf main
- main-Branch erhält dadurch oft neue Änderungen, da diese nicht längerlebigen Branches erfolgen
- Häufige Commits auf main-Branch helfen bei CI/CD
  - CI-Pipeline kann häufig durchlaufen
  - Automatisierte Tests geben schnell Feedback
  - Ermöglicht hochfrequente Releases
- Beim zentralen Git-Workflow liegt ein besonderer Fokus auf die Qualität der Commits
  - Generell sollte in Git nur lauffähiger und getesteter Code committet werden
  - Commits mit schlechtem Code verursachen im zentralen Git-Workflow großen Schaden

Zentraler Git-Workflow

- Entwickler werden gezwungen, oft Änderungen von neuen Commits in ihr lokales Repository einzubauen
  - Erhöht zwar direktes Konfliktpotenzial, reduziert aber Komplexität bei der Integration von Features
  - Gegenbeispiel wäre ein Feature-Branch, der über Wochen nicht zwingend aktualisiert werden muss und nachher wieder in den main-Branch zurückgeführt wird

➔ Verhindert Divergenz



Zentraler Git-Workflow

- Durch die gemeinsame Arbeit auf einem Branch steigt das Konfliktpotenzial
  - Häufige Kommunikation nötig
  - Entwickler sollten mit Umgang von Konflikten in Git vertraut sein und diese sauber auflösen können
- Ein zentraler Git-Workflow bietet weniger Flexibilität als andere Workflows
  - Komplexere Projekte mit vielen parallelen und inhaltlich getrennten Features können hier von anderen Workflows profitieren
- Zentraler Git-Workflow entfaltet seine Vorteile meinst in kleineren Teams

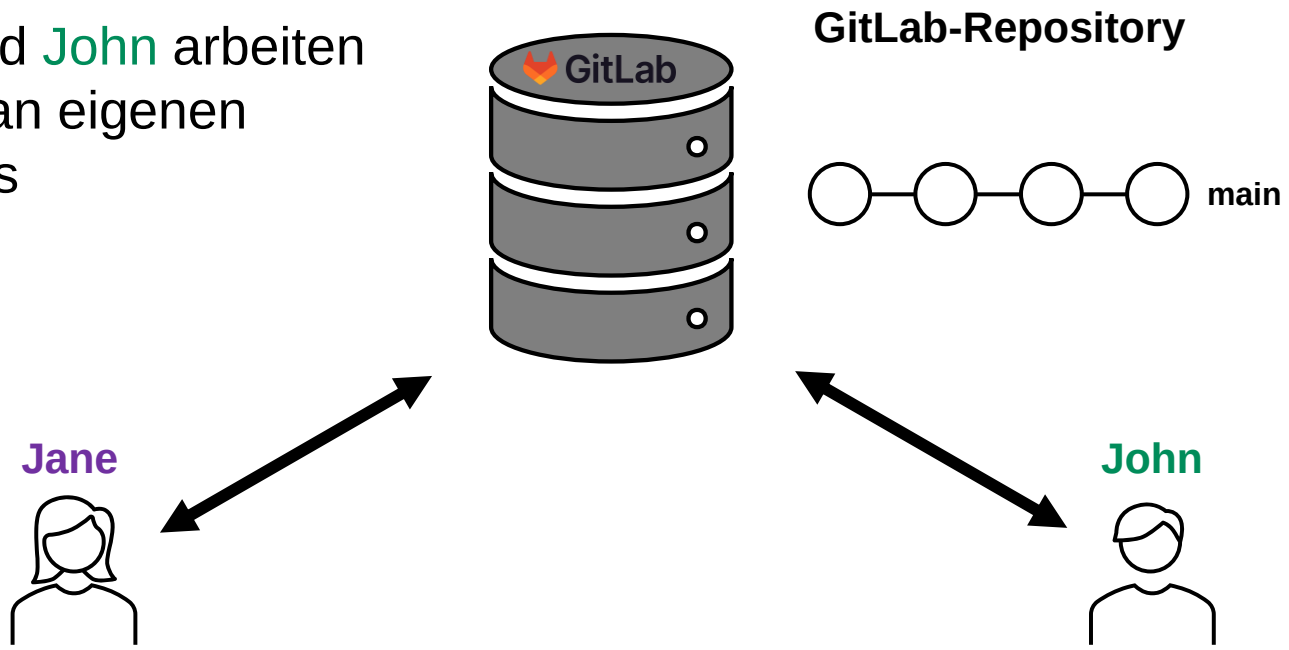
Git

Arbeiten im zentralen Workflow

Arbeiten im zentralen Workflow

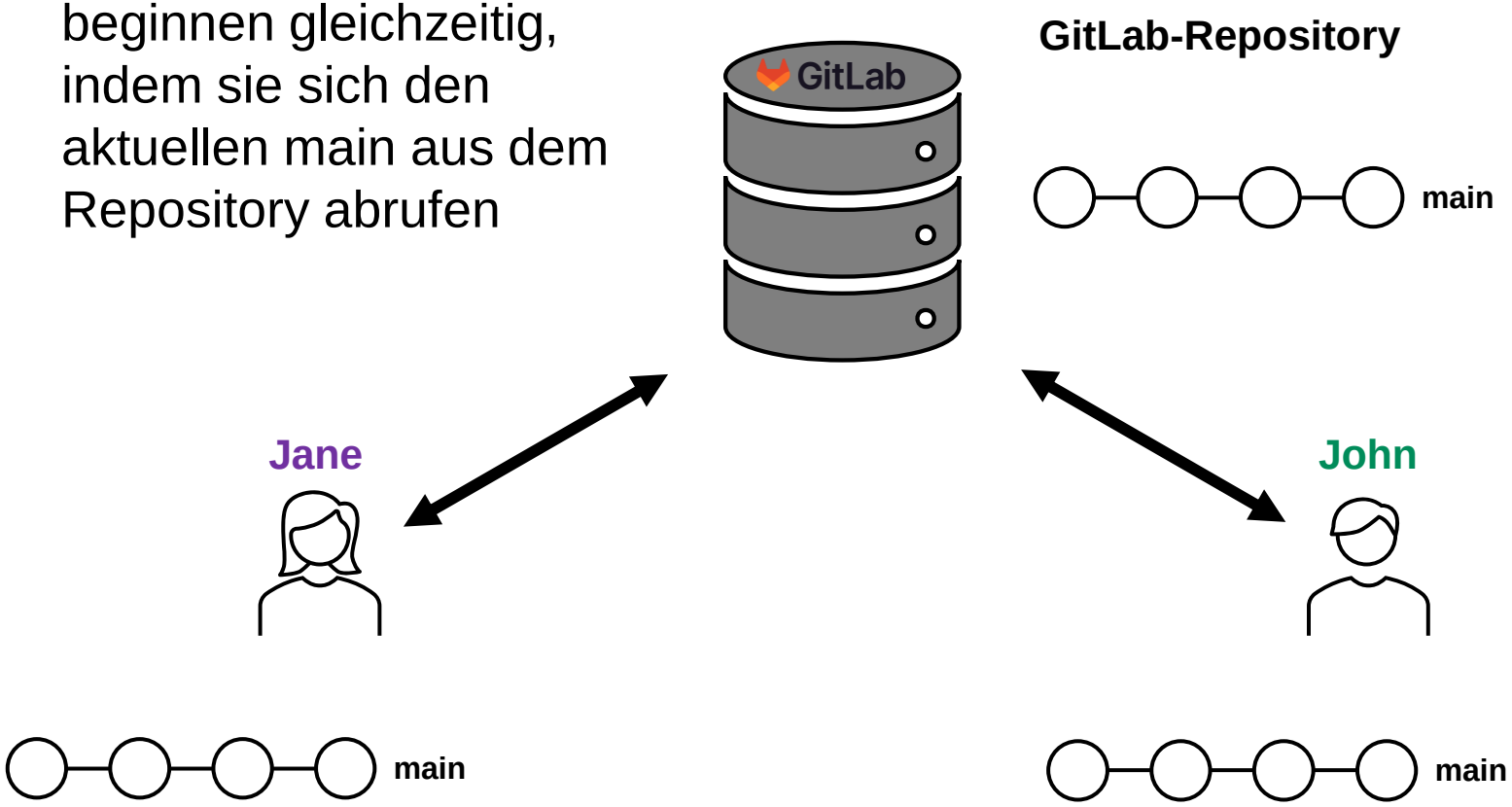
Beispielszenario

- Jane und John arbeiten jeweils an eigenen Features



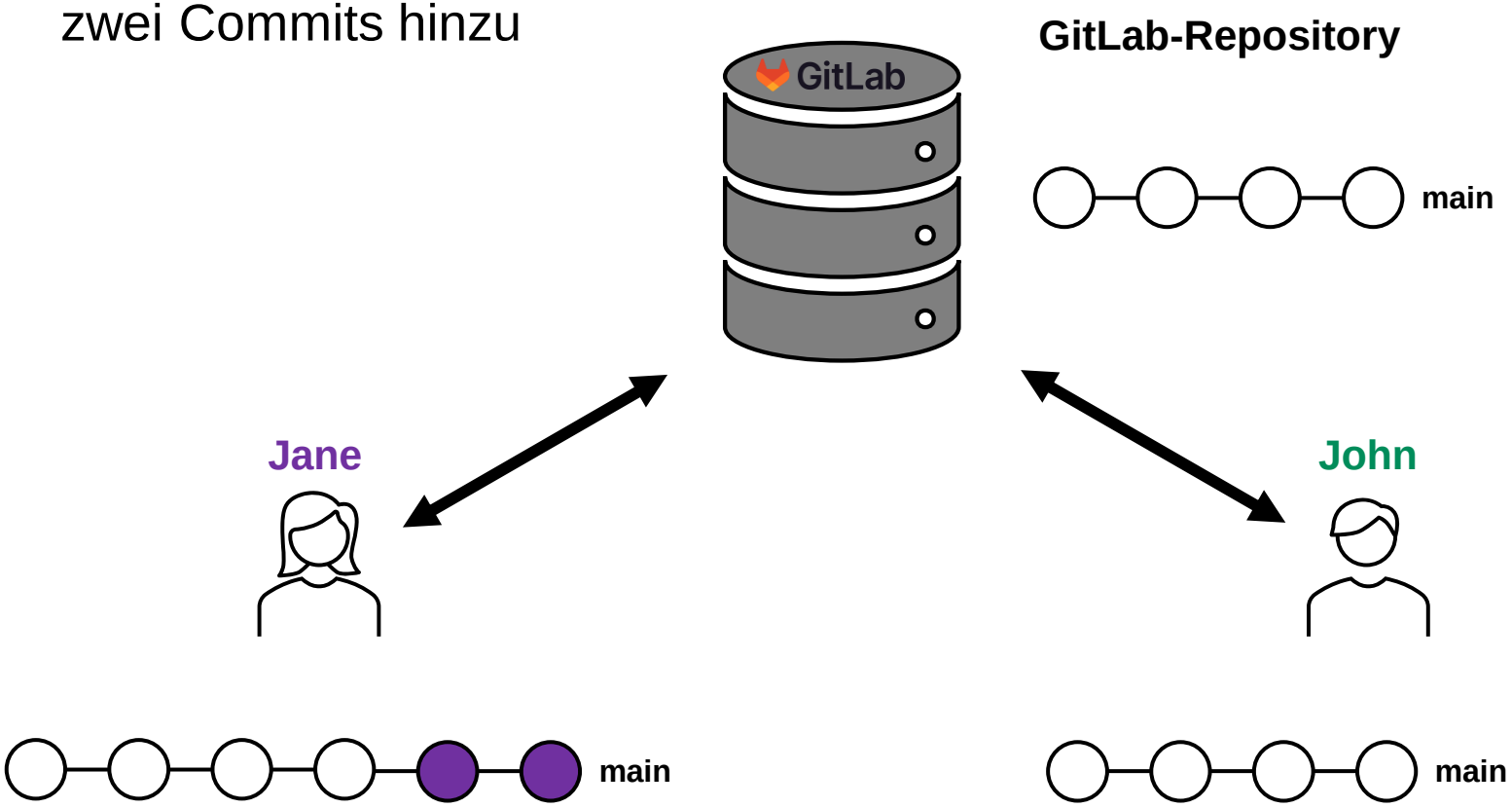
Arbeiten im zentralen Workflow

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



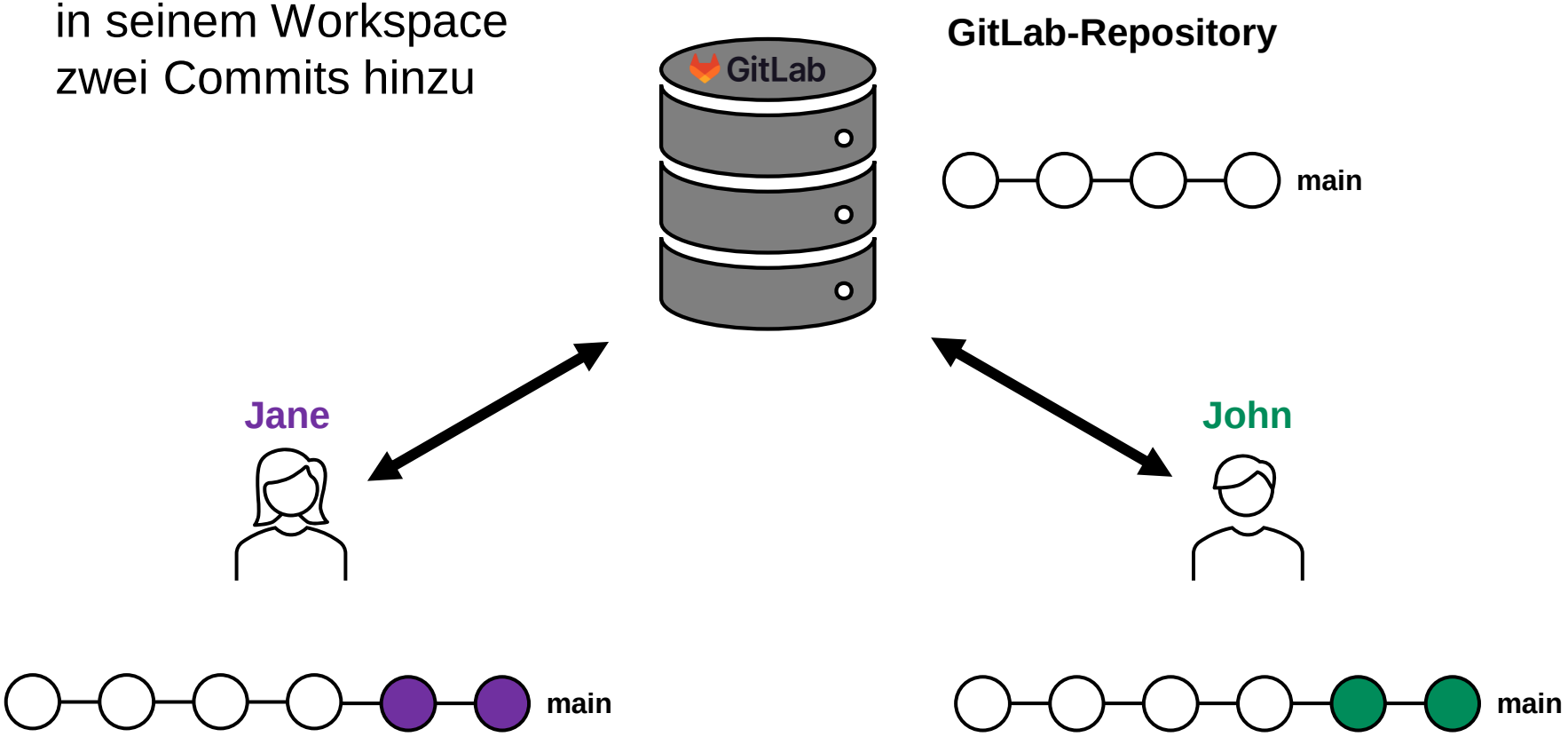
Arbeiten im zentralen Workflow

- Jane fügt bei sich lokal zwei Commits hinzu



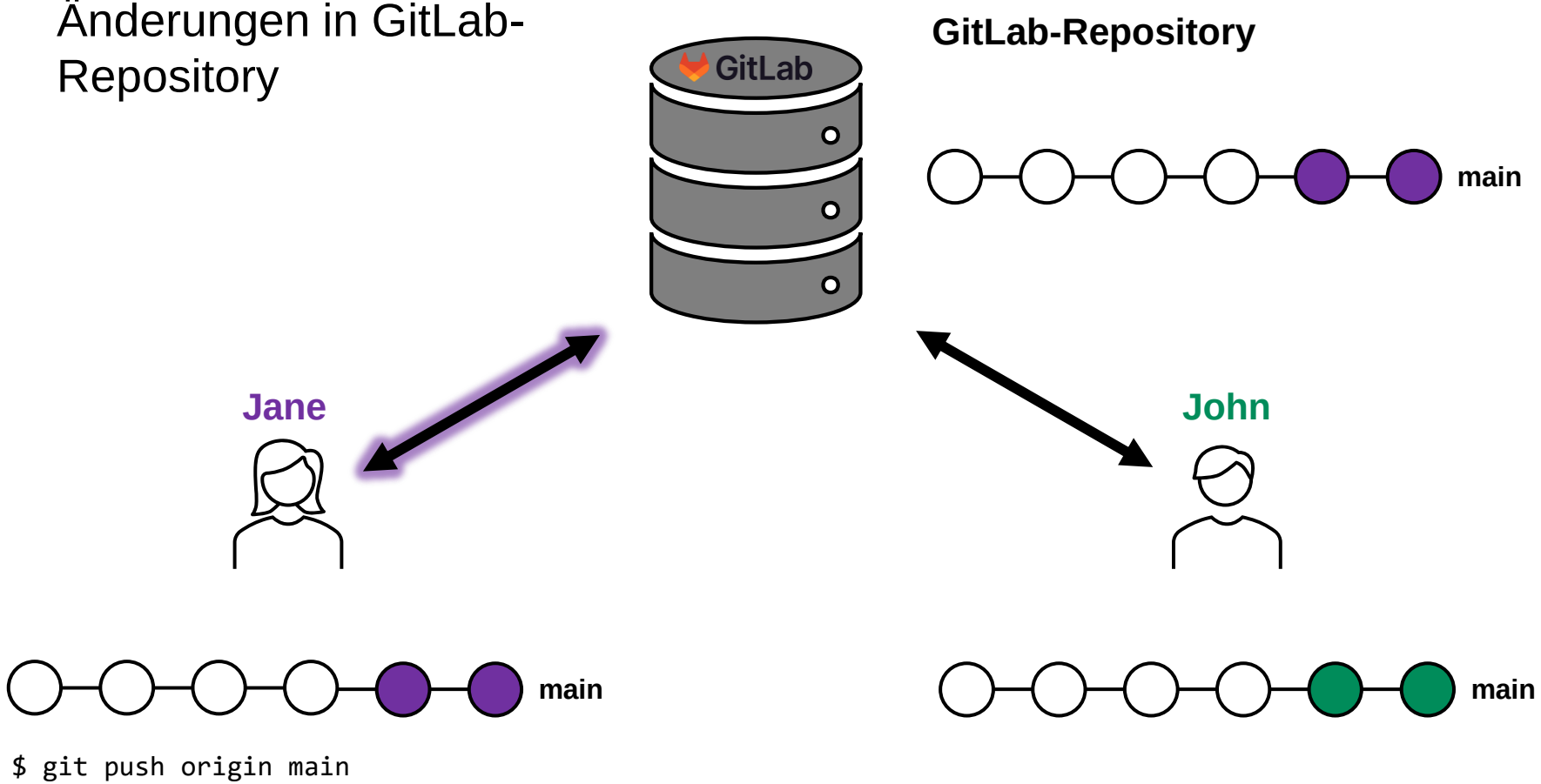
Arbeiten im zentralen Workflow

- John fügt ebenfalls lokal in seinem Workspace zwei Commits hinzu



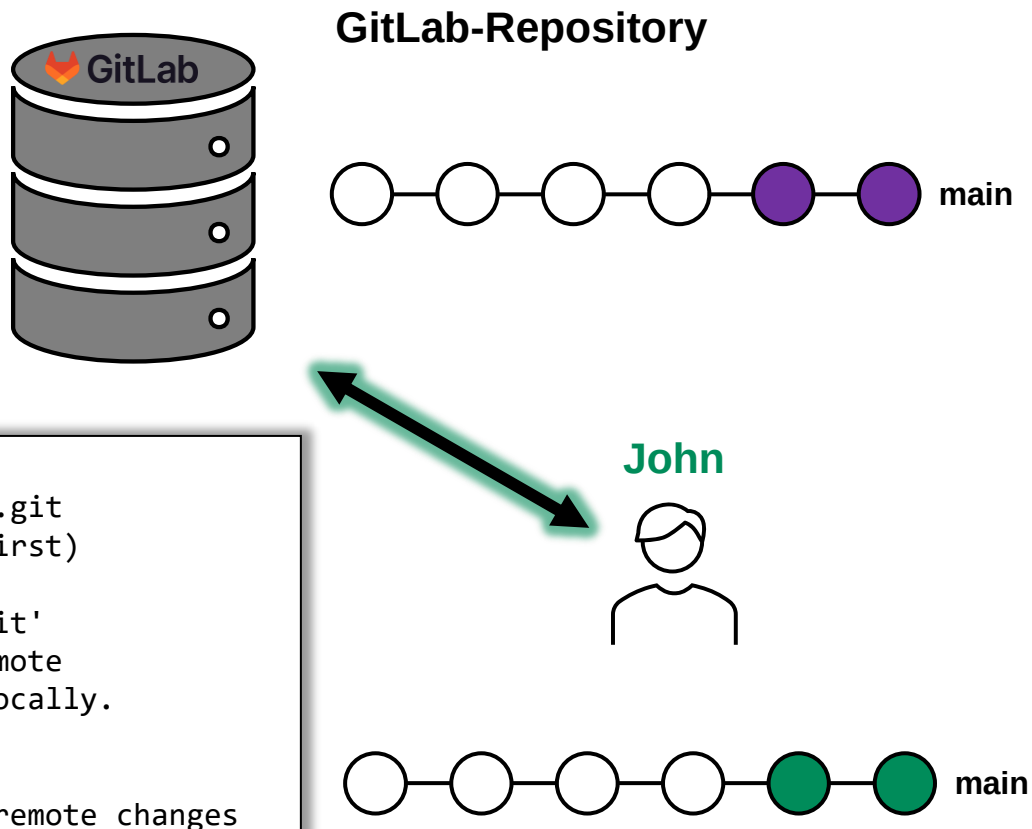
Arbeiten im zentralen Workflow

- Jane pusht ihre Änderungen in GitLab-Repository



Arbeiten im zentralen Workflow

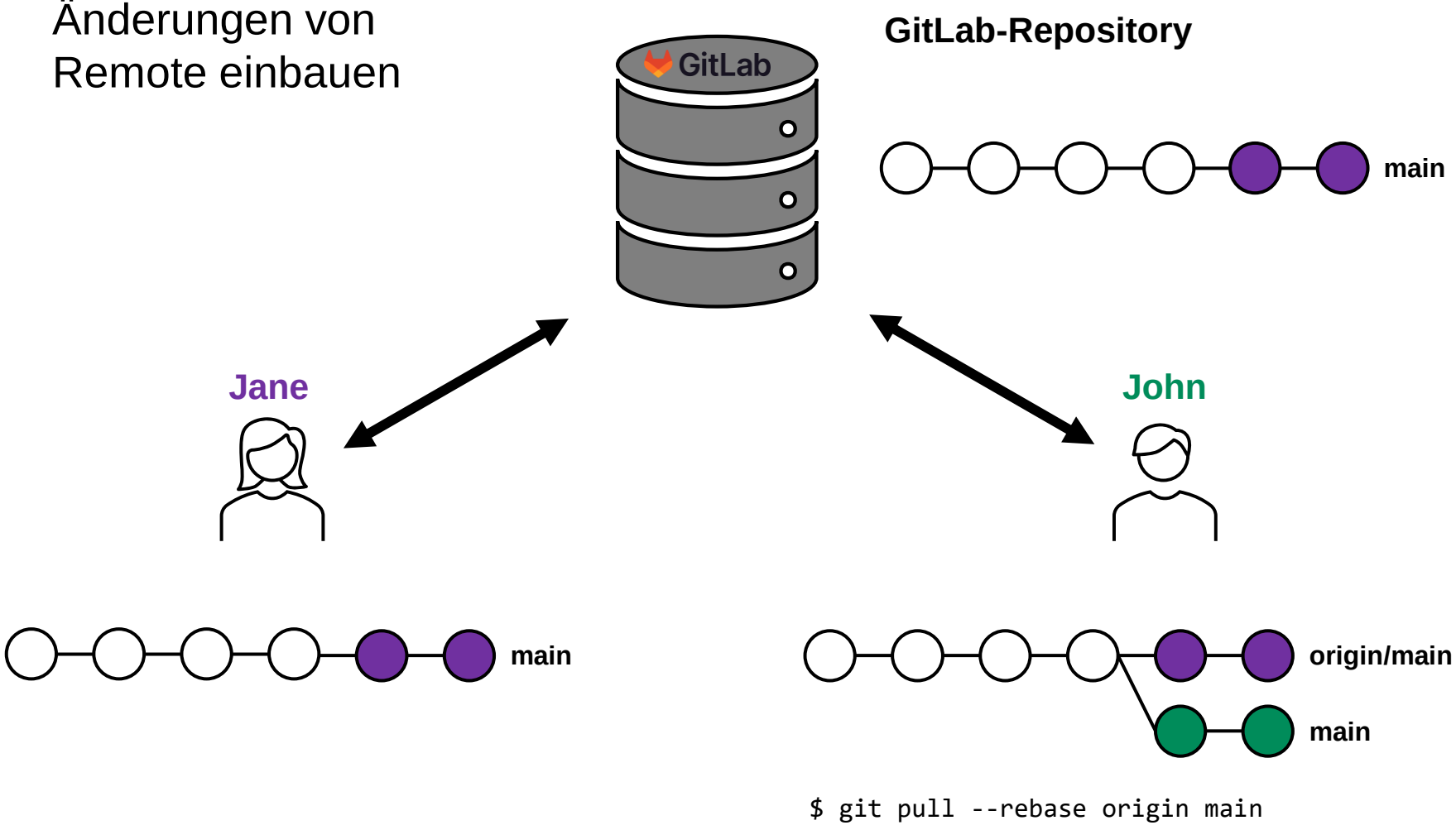
- Versucht nun John seine Änderungen ebenfalls zu pushen, erhält er einen Fehler




```
$ git push origin main
To https://gitlab.com/john/example-project.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to
'https://gitlab.com/john/example-project.git'
hint: Updates were rejected because the remote
hint: contains work that you do not have locally.
Hint: This is usually caused by another
hint: repository pushing to the same ref.
hint: You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards’
hint: in 'git push --help' for details.
```

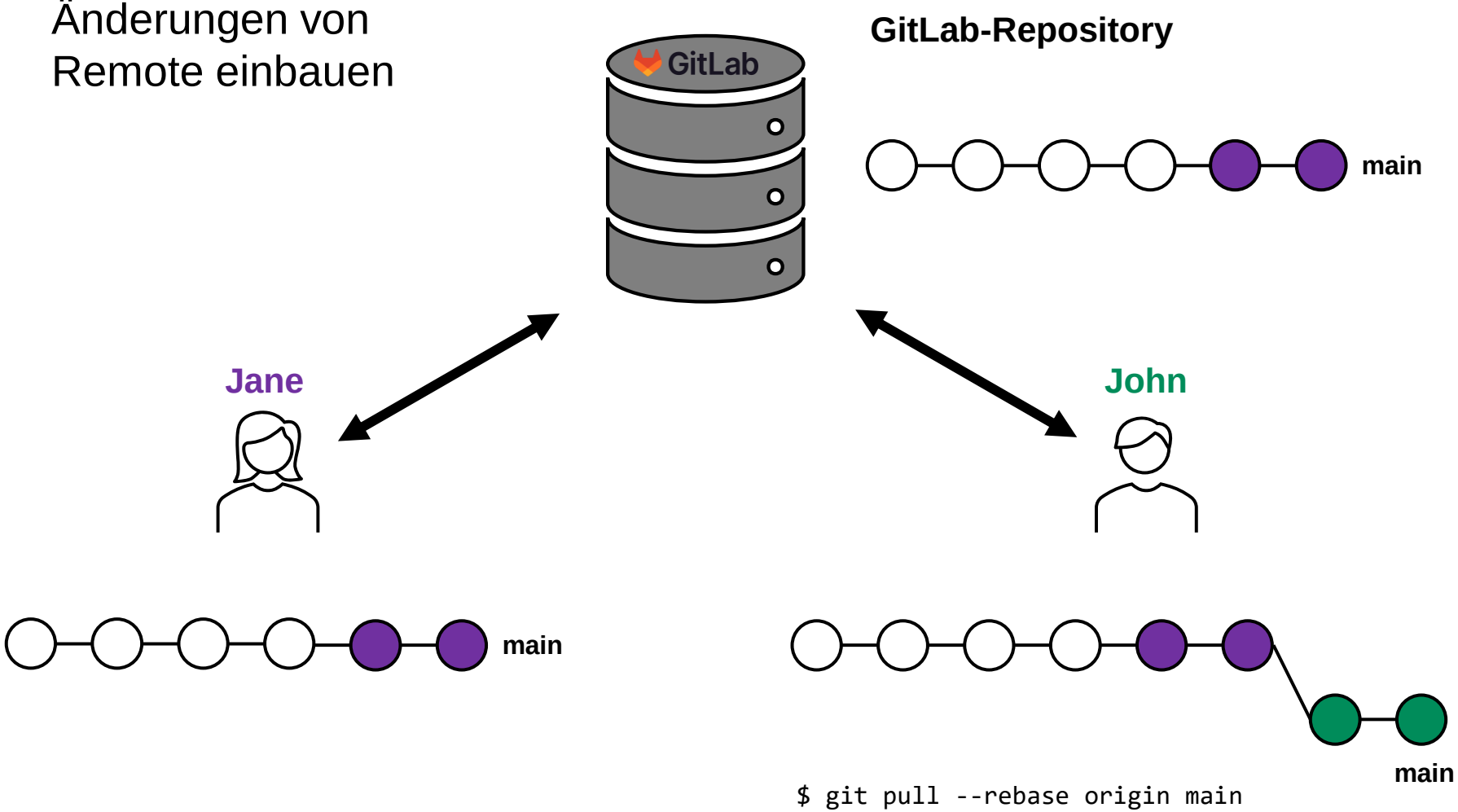
Arbeiten im zentralen Workflow

- John muss zunächst Änderungen von Remote einbauen



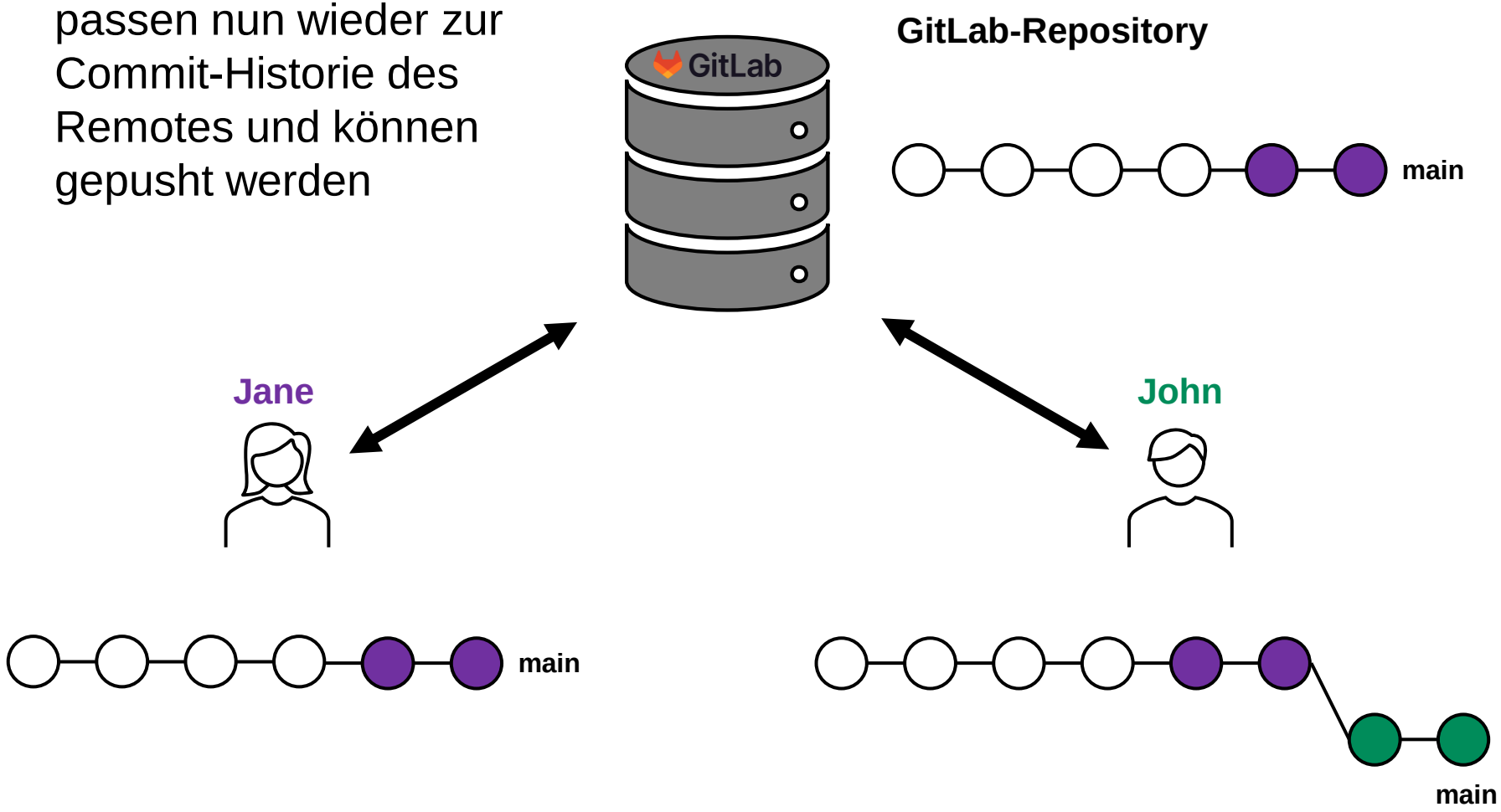
Arbeiten im zentralen Workflow

- John muss zunächst Änderungen von Remote einbauen



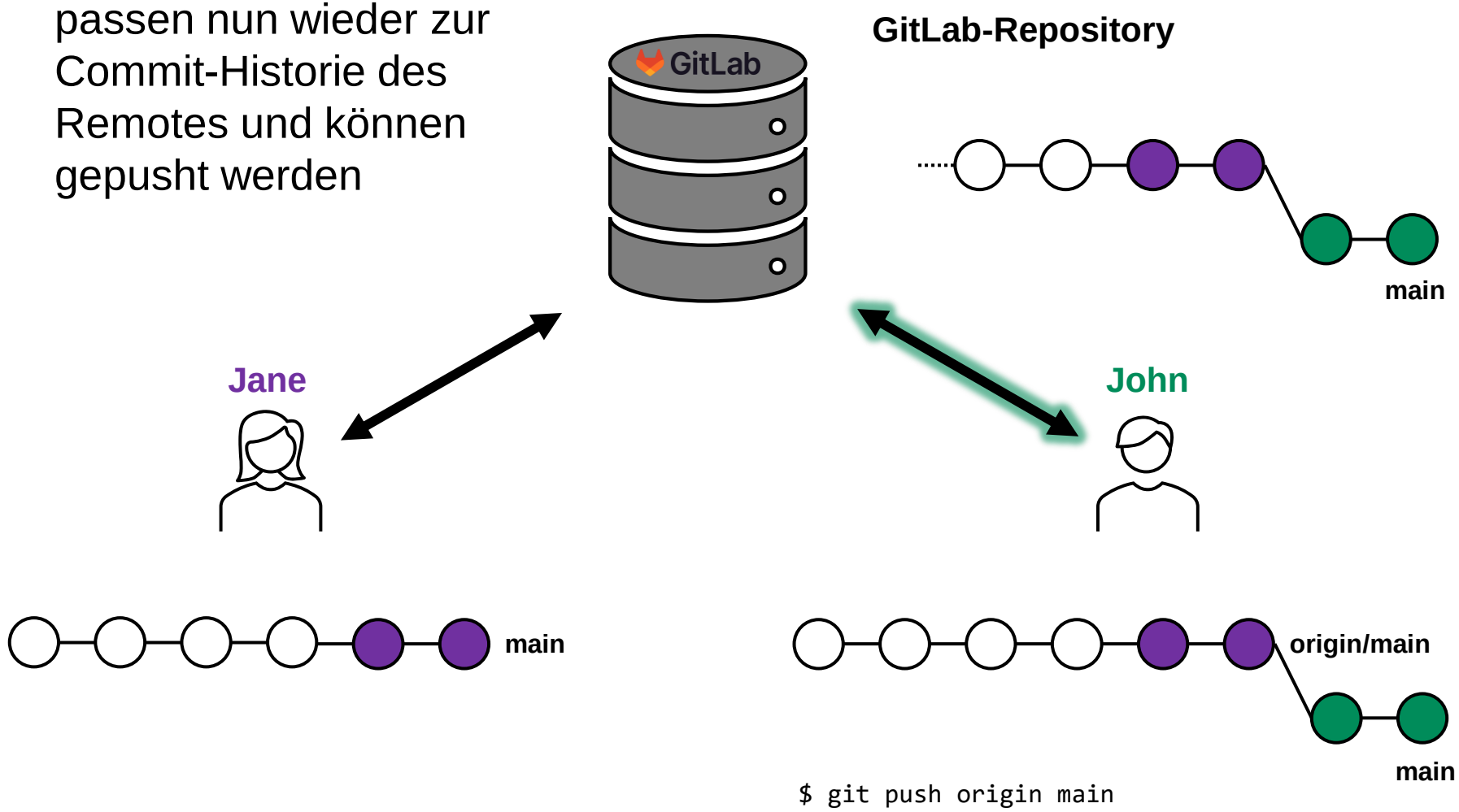
Arbeiten im zentralen Workflow

- Johns Änderungen passen nun wieder zur Commit-Historie des Remotes und können gepusht werden



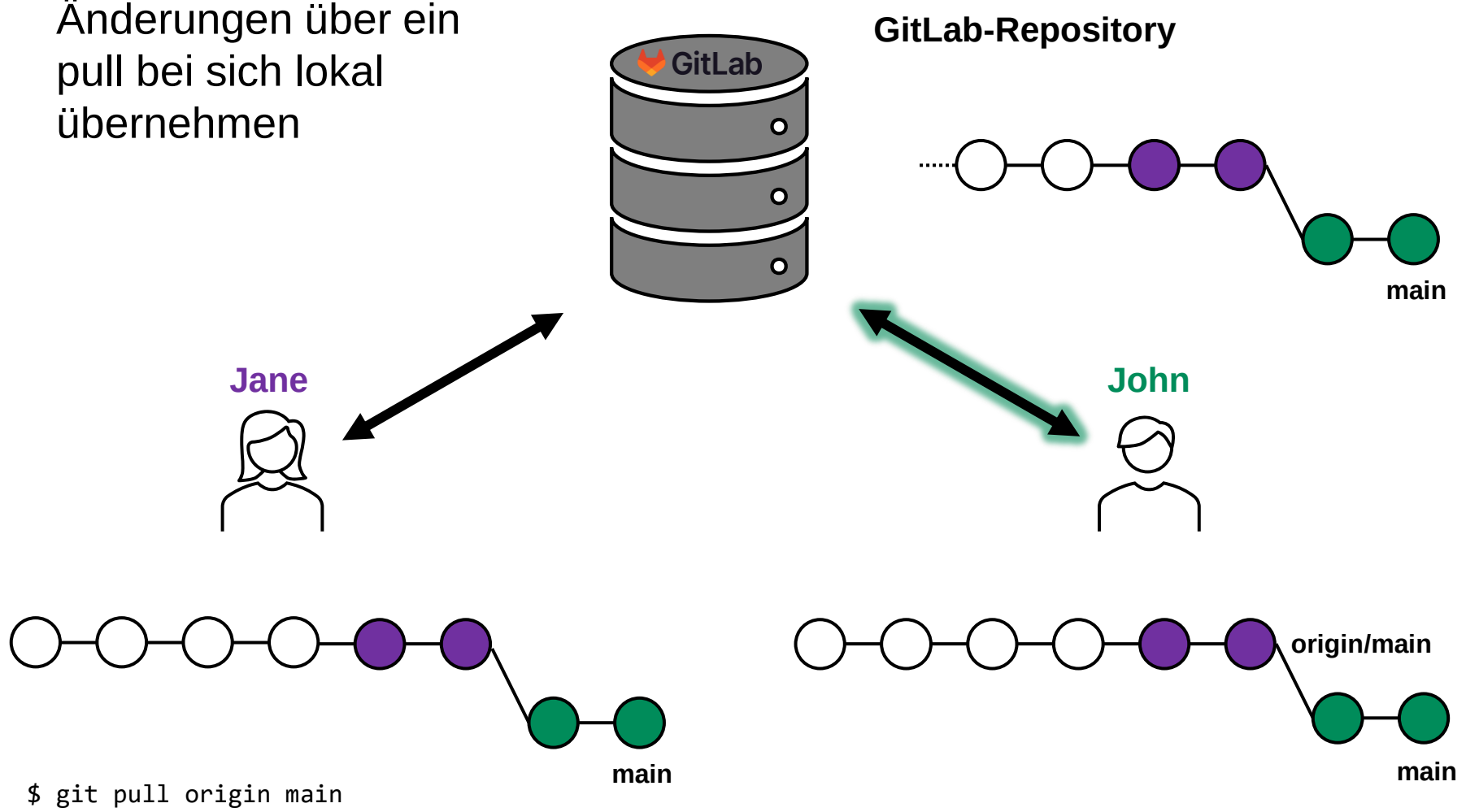
Arbeiten im zentralen Workflow

- Johns Änderungen passen nun wieder zur Commit-Historie des Remotes und können gepusht werden



Arbeiten im zentralen Workflow

- Jane Kann nun Johns Änderungen über ein pull bei sich lokal übernehmen



Arbeiten im zentralen Workflow

- Häufiges Rebasing nötig
- Rebasing zum Übernehmen der Remote Änderungen eignet sich hier besser als Merging, da zusätzliche Commits verhindert werden
- Nur die Commits, welche lokal aber nicht Remote existieren, werden rebased. Daher verstößt ein Rebase hier auch nicht gegen das Prinzip, keine Public Branches zu rebasen

Git

Alternative Workflows

Alternative Workflows

- Die meisten anderen Workflows sind vom Branching Modell her komplexer
- Ein weiterer bekannte Workflow ist der Feature-Branch-Workflow
  - Features werden in eigenen Branches entwickelt und nach Abschluss in den Hauptbranch gemerged
  - Bietet genau umgekehrte Vor- und Nachteile gegenüber dem zentralen Git-Workflow
  - Wird mit dem Gitflow-Workflow als Variante noch genauer betrachtet
- Es existieren auch Workflows, die sich vollkommen Unterscheiden
  - Beispielsweise der Forking-Workflow sieht auch für jeden Entwickler ein eigenes Remote Repository
  - Dabei forkt ein Entwickler ein Projekt-Repository und arbeitet dort alleine
  - Später werden mit Merge-Request Änderungen ins ursprüngliche Repository übernommen