dataport

anderScore
trust in competence

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

17.06.2024, 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

Workflows

Git-Workflow

Inhalt

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

Git-Workflow

Was sind Git-Workflows?

- Workflows sind Empfehlungen & Strategien im Remote-Kontext
- Sorgen im Team für konsistente und effektive Nutzung von Git & GitLab
- Workflows = Empfehlungen
  - Keine absoluten Regeln!

Git-Workflow

Was sind Git-Workflows?

- Es gibt nicht den einen Git-Workflow
- Git und GitLab vielfältige Einsatzmöglichkeiten
  - Durch verschiedene Konzepte und Features
- ➔ Viele Git-Workflows mit unterschiedlichen Konzepten
- Auswahl des passenden Workflows nach bestimmten Kriterien
  - Projektart
  - Projektgröße und Umfang
  - Teamgröße
  - Teamkultur
- Teammitglieder müssen den Workflow kennen und produktiv integrieren
- Workflow darf keinen unnötigen Overhead erzeugen

Git-Workflow

anderScore trust in competence

Zentraler Git-Workflow

- Zentraler Git-Workflow = einfacher Workflow
- Im zentralen Ansatz wird nur ein Branch benötigt
  - Häufig **main**, (auch **trunk** oder **master** (veraltet))
  - Änderungen als Commit
  - Keine Branch-Verwaltung → Weniger Overhead
- → Einfach und schnell zu verstehen
- Erleichtert Umstieg von CVCS (Subversion)
- Beliebt wegen Kompatibilität zu CI/CD

Git-Workflow

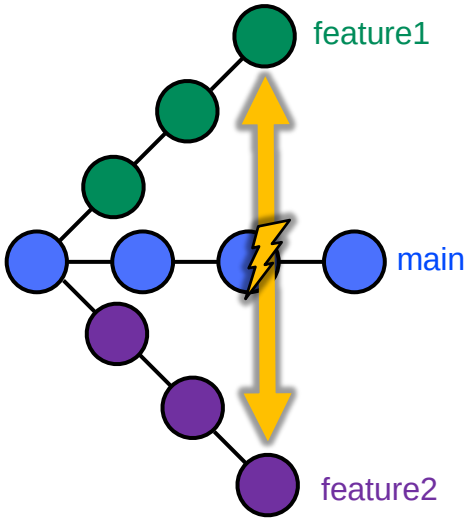
Zentraler Git-Workflow

- Entwickler committen direkt auf main
  - → oft neue Änderungen (main)
  - Keine längerlebigen Branches
- Häufige Commits unterstützen CI/CD
  - CI-Pipeline kann häufig durchlaufen
  - Automatisierte Tests (= schnelles Feedback)
  - Hochfrequente Releases möglich
- Zentraler Git-Workflow legt Fokus auf Commit-Qualität
  - = lauffähig und getestet
  - Schlechte Code-Qualität = großer Schaden

Zentraler Git-Workflow

- Erfordert regelmäßige Updates im lokalen Repository
  - Erhöht Konfliktpotenzial, reduziert Integrationskomplexität
  - Gegenbeispiel: Feature-Branch

➔ Verhindert Divergenz



Git-Workflow

Zentraler Git-Workflow

- Gemeinsame Arbeit (auf einem Branch) erhöht Konfliktpotenzial
  - Häufige Kommunikation nötig
  - (Merge-)Konflikte sauber auflösen!
- Weniger Flexibilität (als andere Workflows)
  - Komplexere Projekte, andere Workflows?
- Vorteilhaft in kleineren Teams

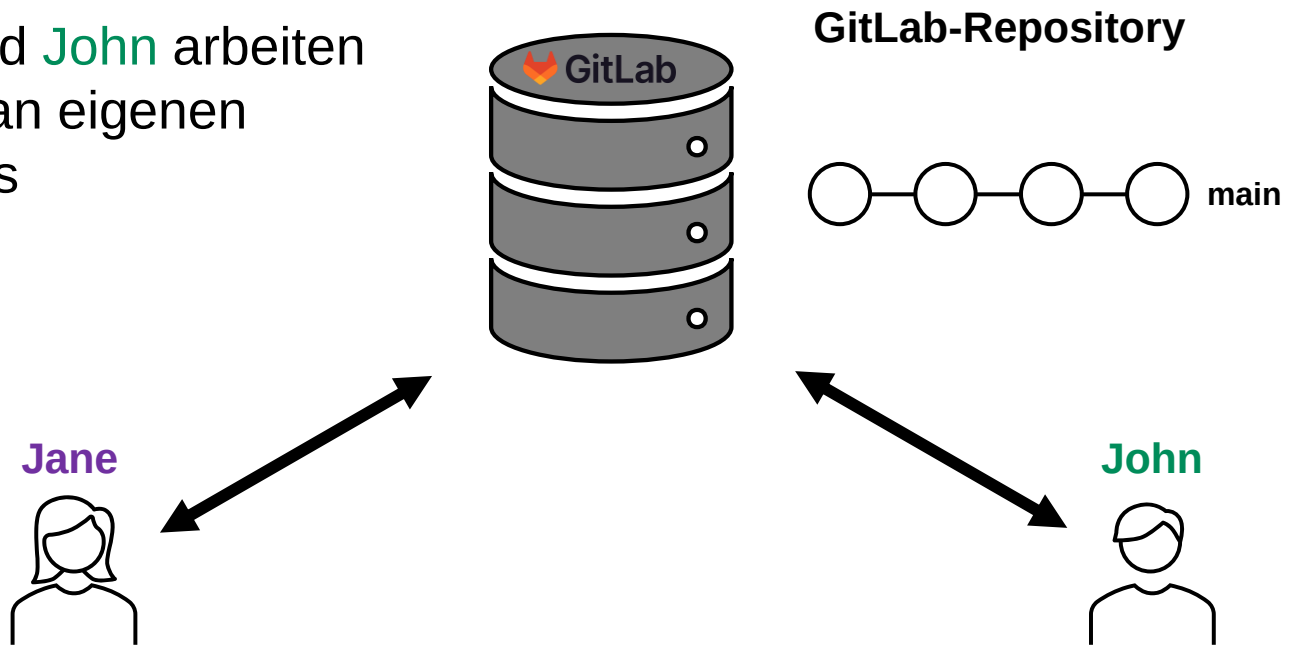
Git

Arbeiten im zentralen Workflow

Arbeiten im zentralen Workflow

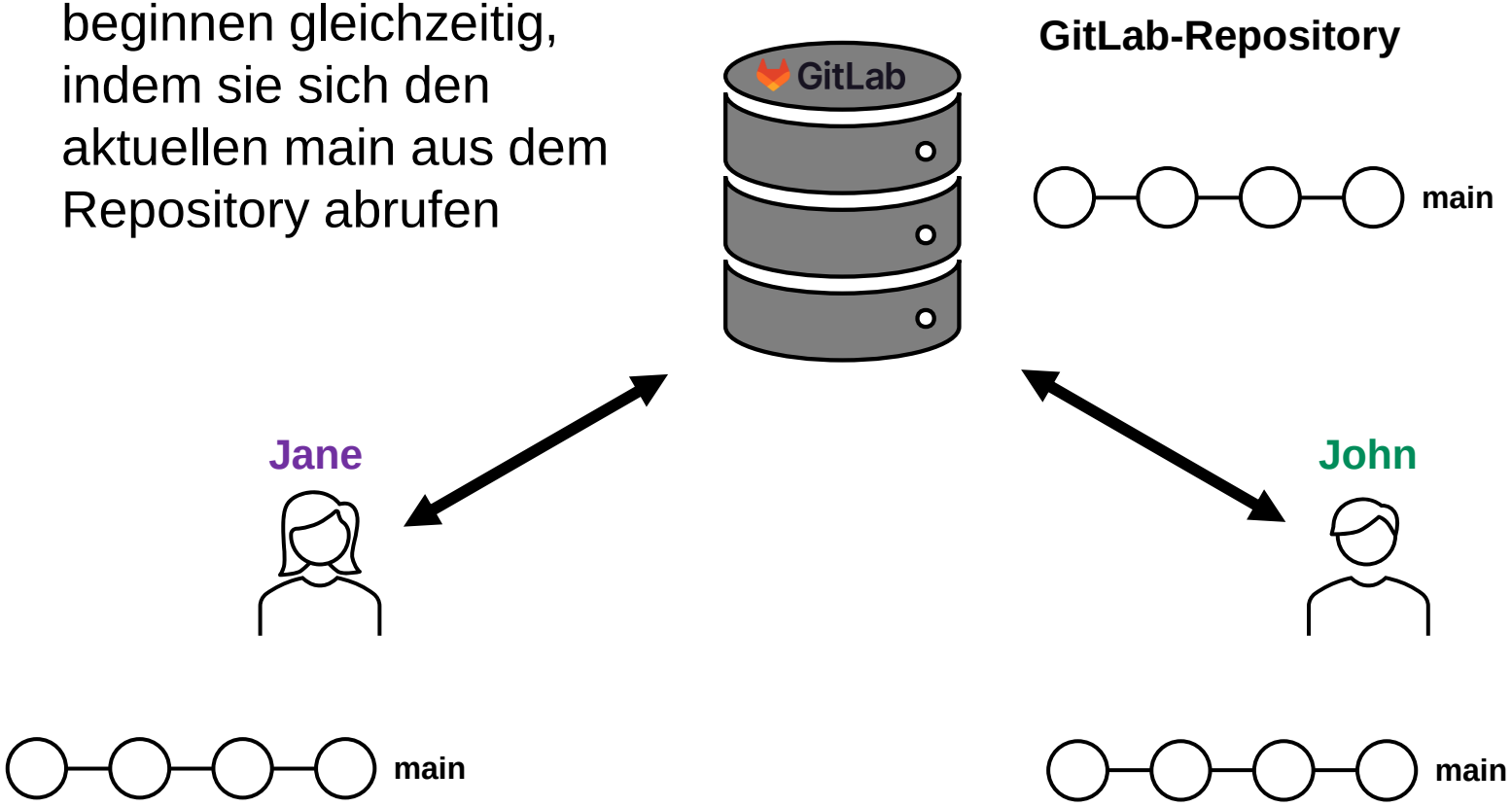
Beispielszenario

- Jane und John arbeiten jeweils an eigenen Features



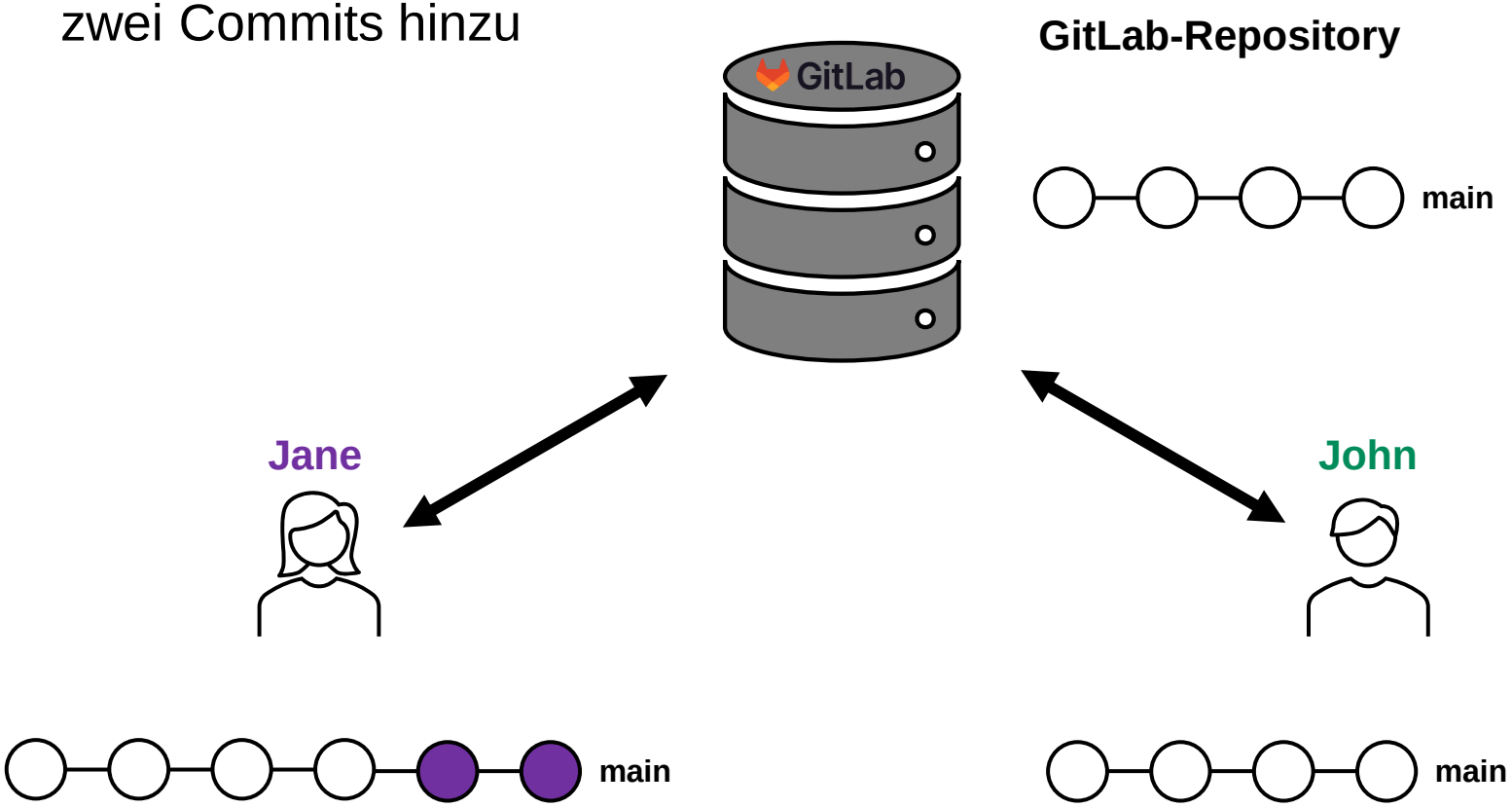
Arbeiten im zentralen Workflow

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



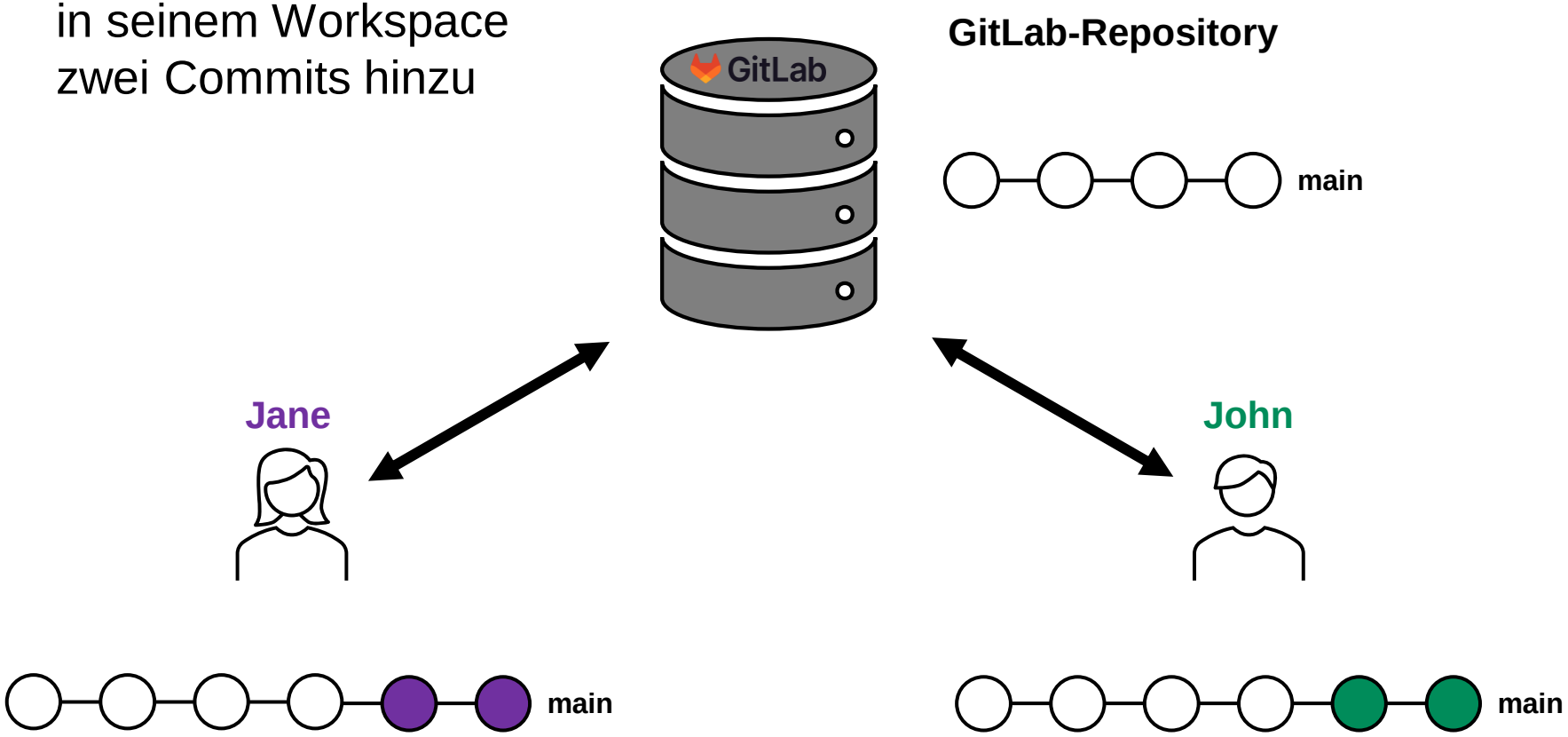
Arbeiten im zentralen Workflow

- Jane fügt bei sich lokal zwei Commits hinzu



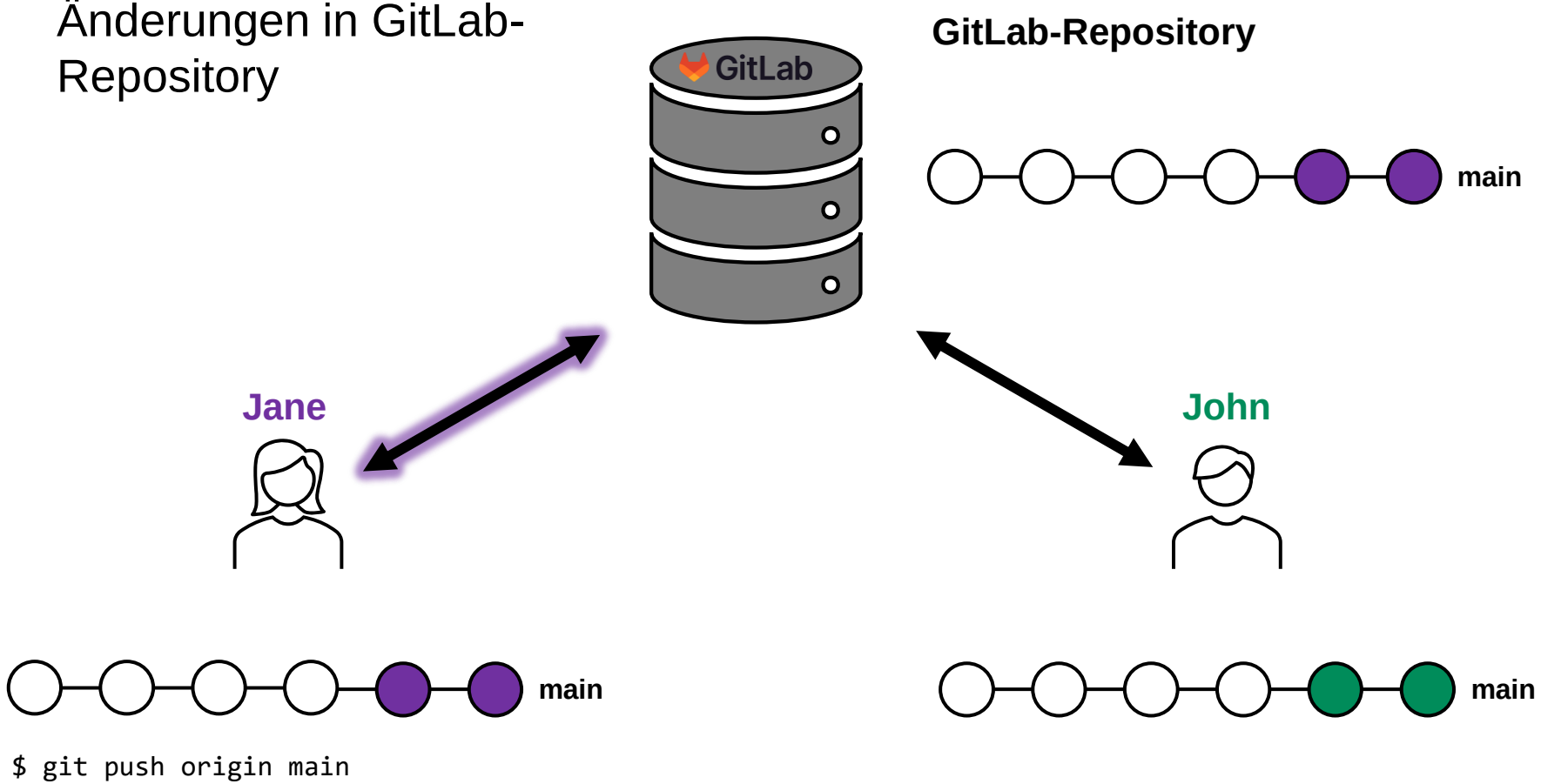
Arbeiten im zentralen Workflow

- John fügt ebenfalls lokal in seinem Workspace zwei Commits hinzu



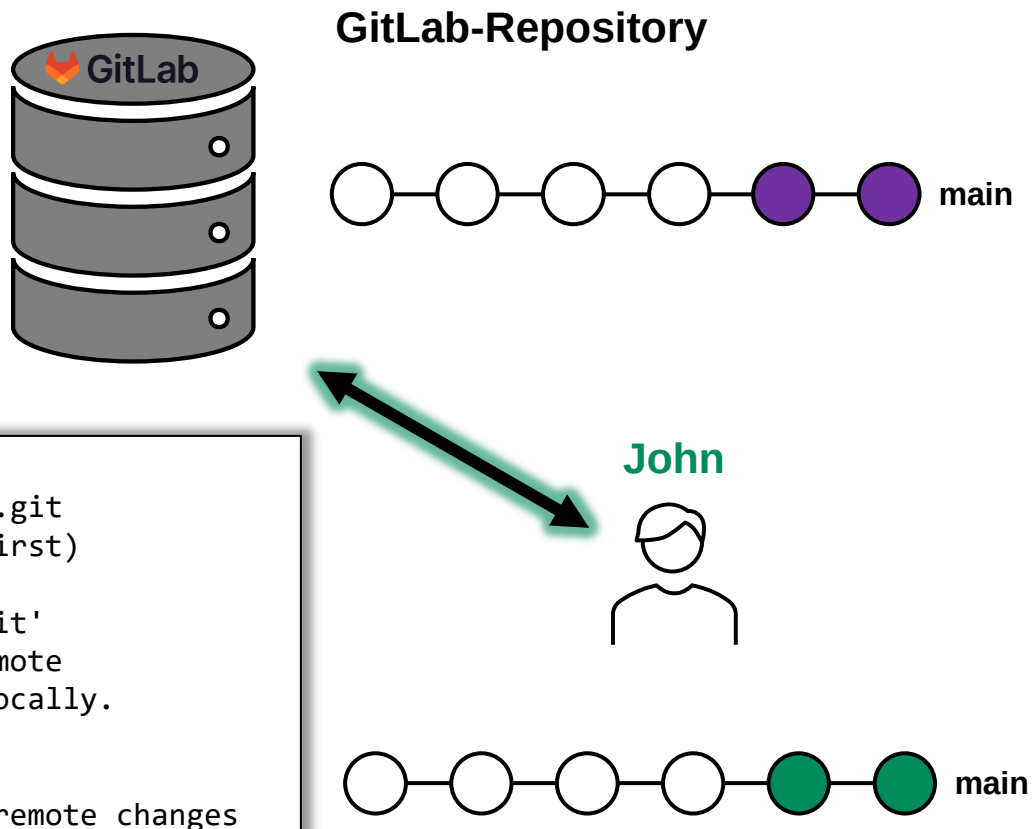
Arbeiten im zentralen Workflow

- Jane pusht ihre Änderungen in GitLab-Repository



Arbeiten im zentralen Workflow

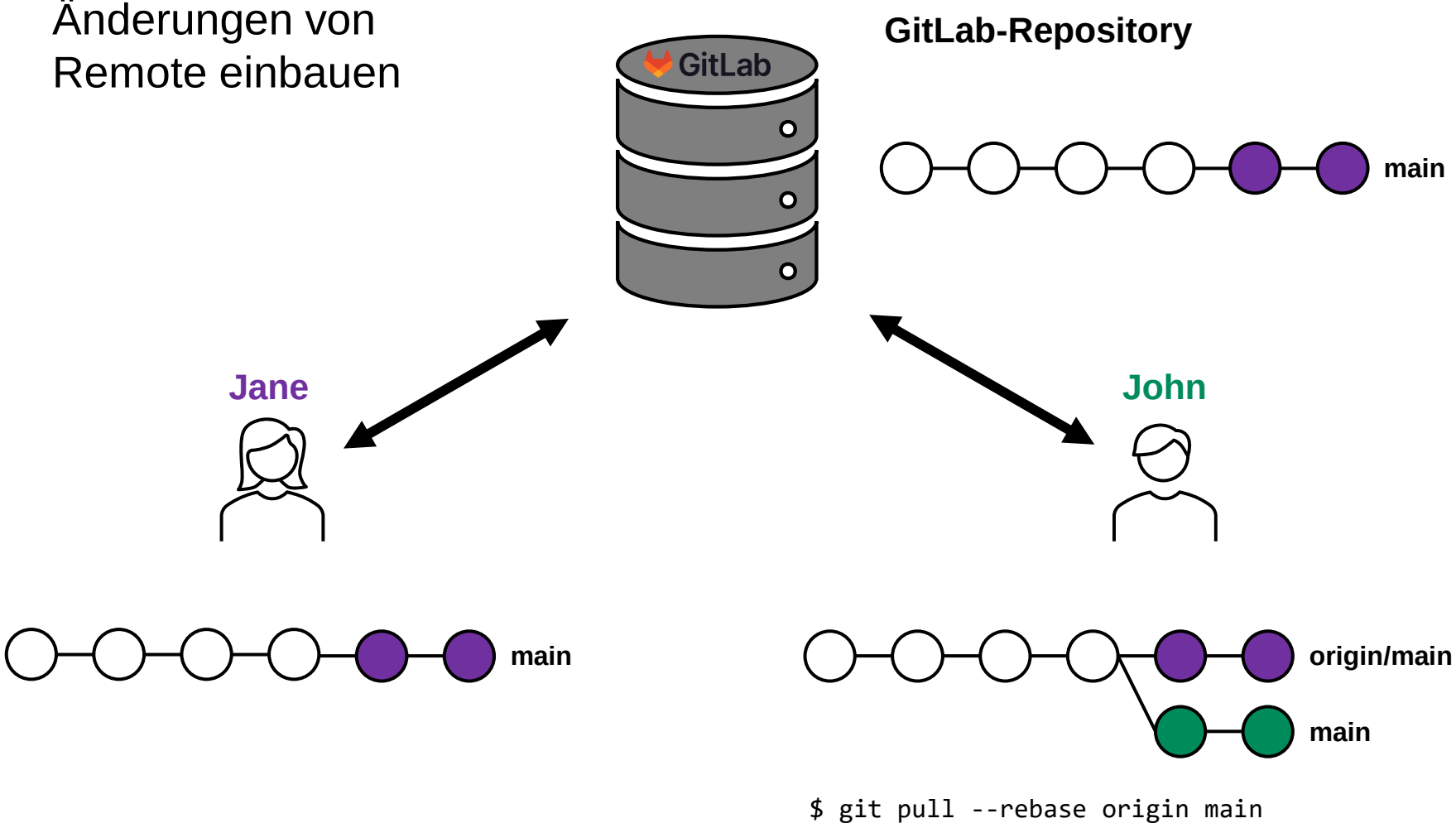
- Versucht nun John seine Änderungen ebenfalls zu pushen, erhält er einen Fehler




```
$ 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

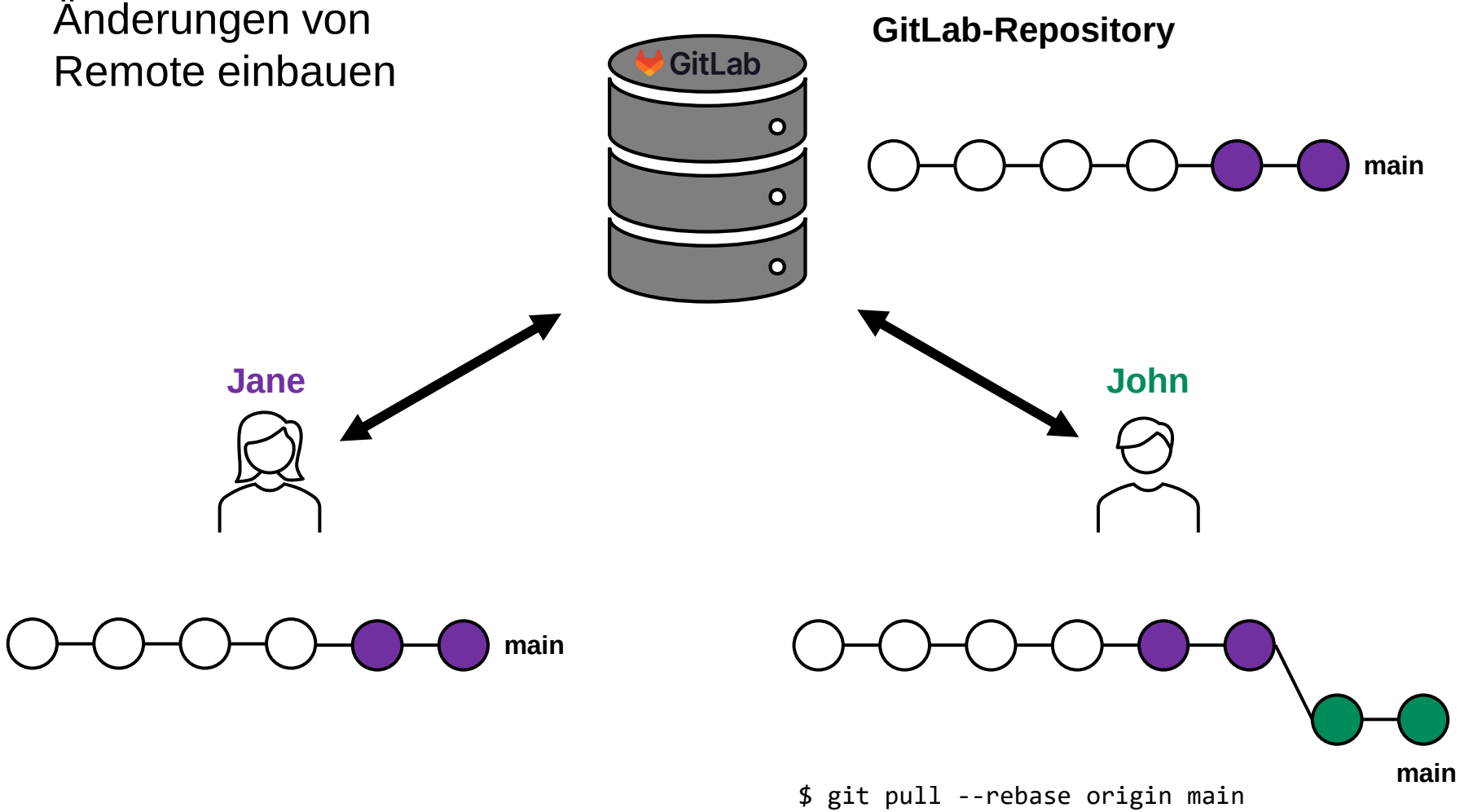
- John muss zunächst Änderungen von Remote einbauen




```
$ git pull --rebase origin main
```

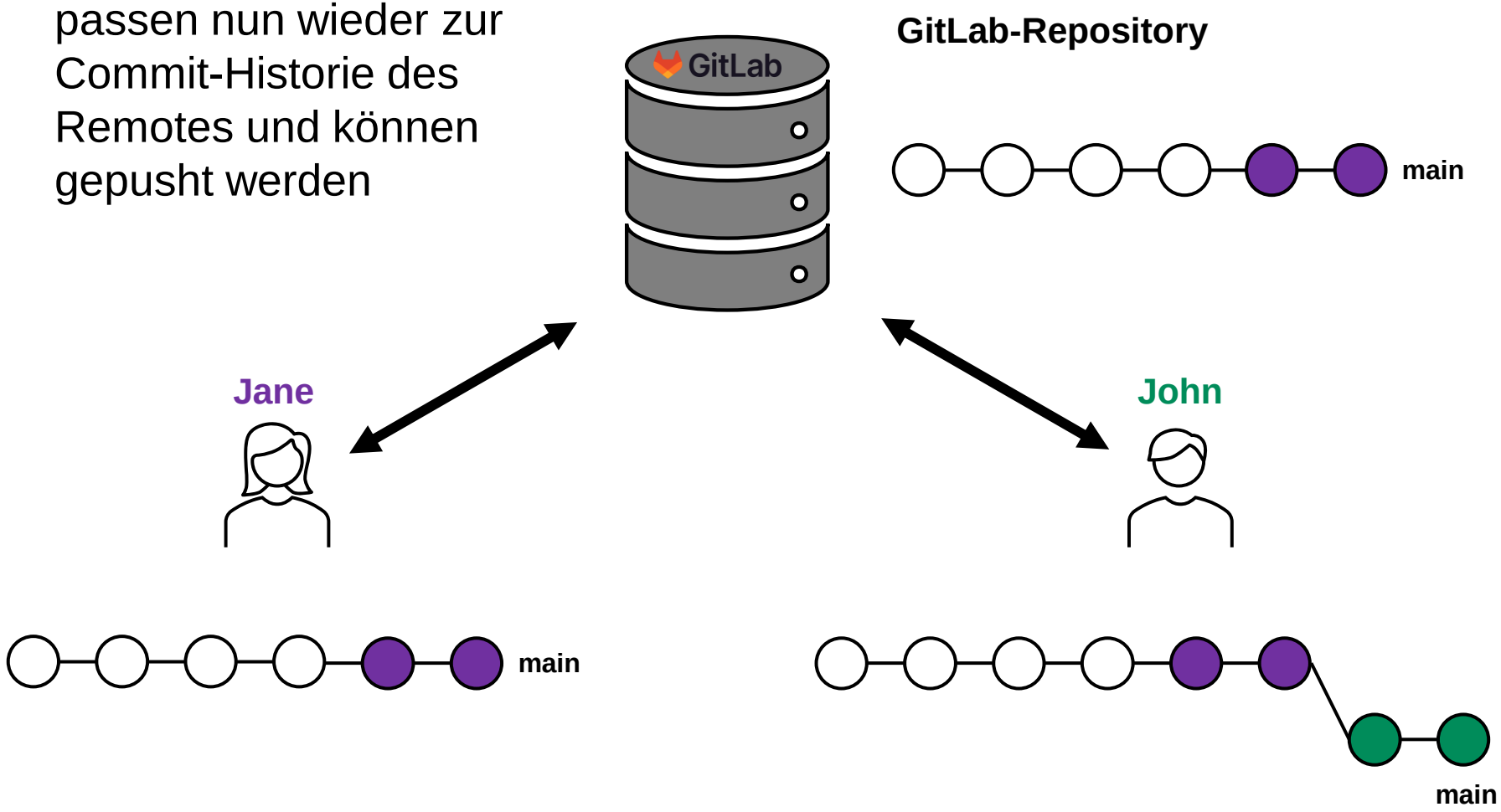
Arbeiten im zentralen Workflow

- John muss zunächst Änderungen von Remote einbauen



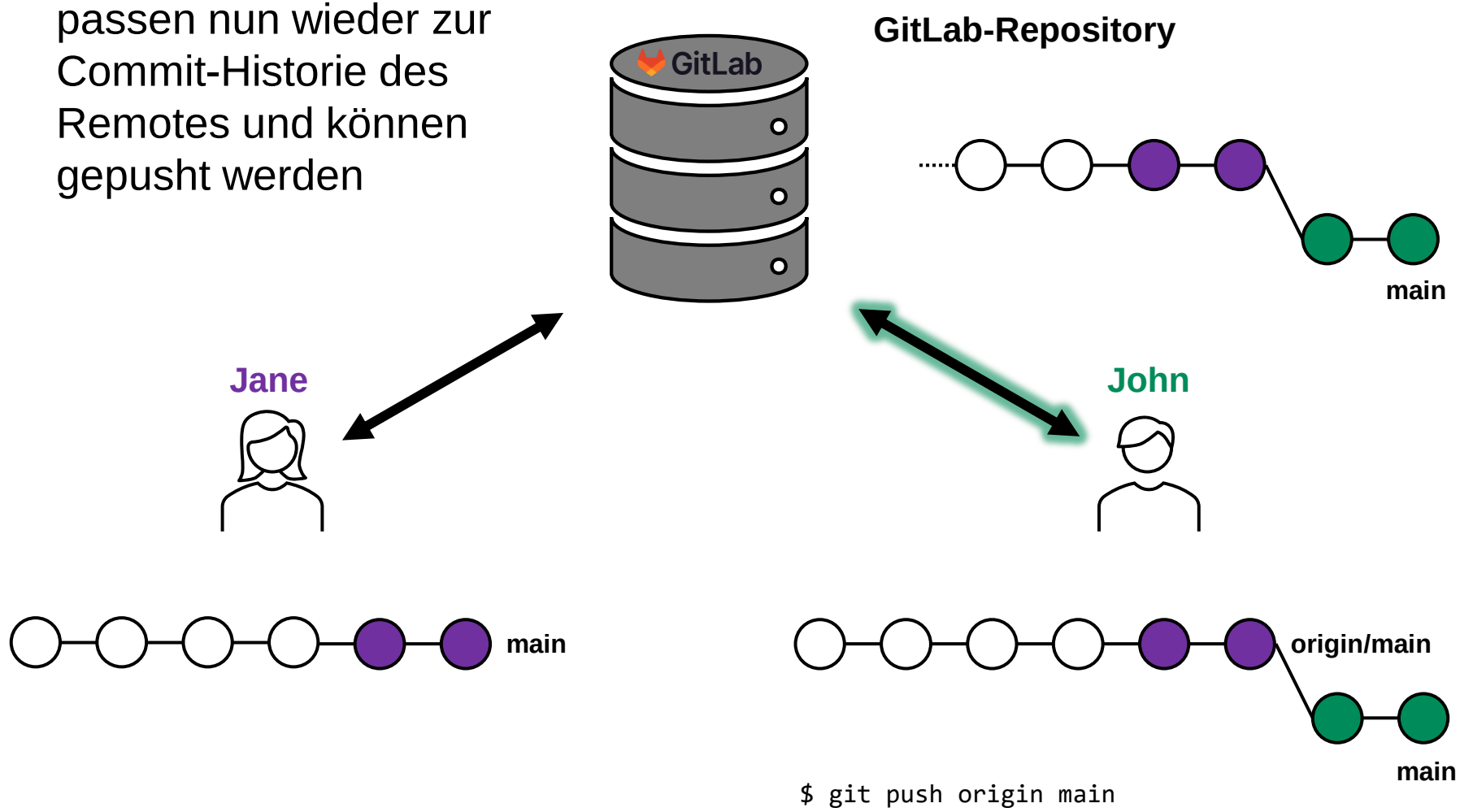
Arbeiten im zentralen Workflow

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



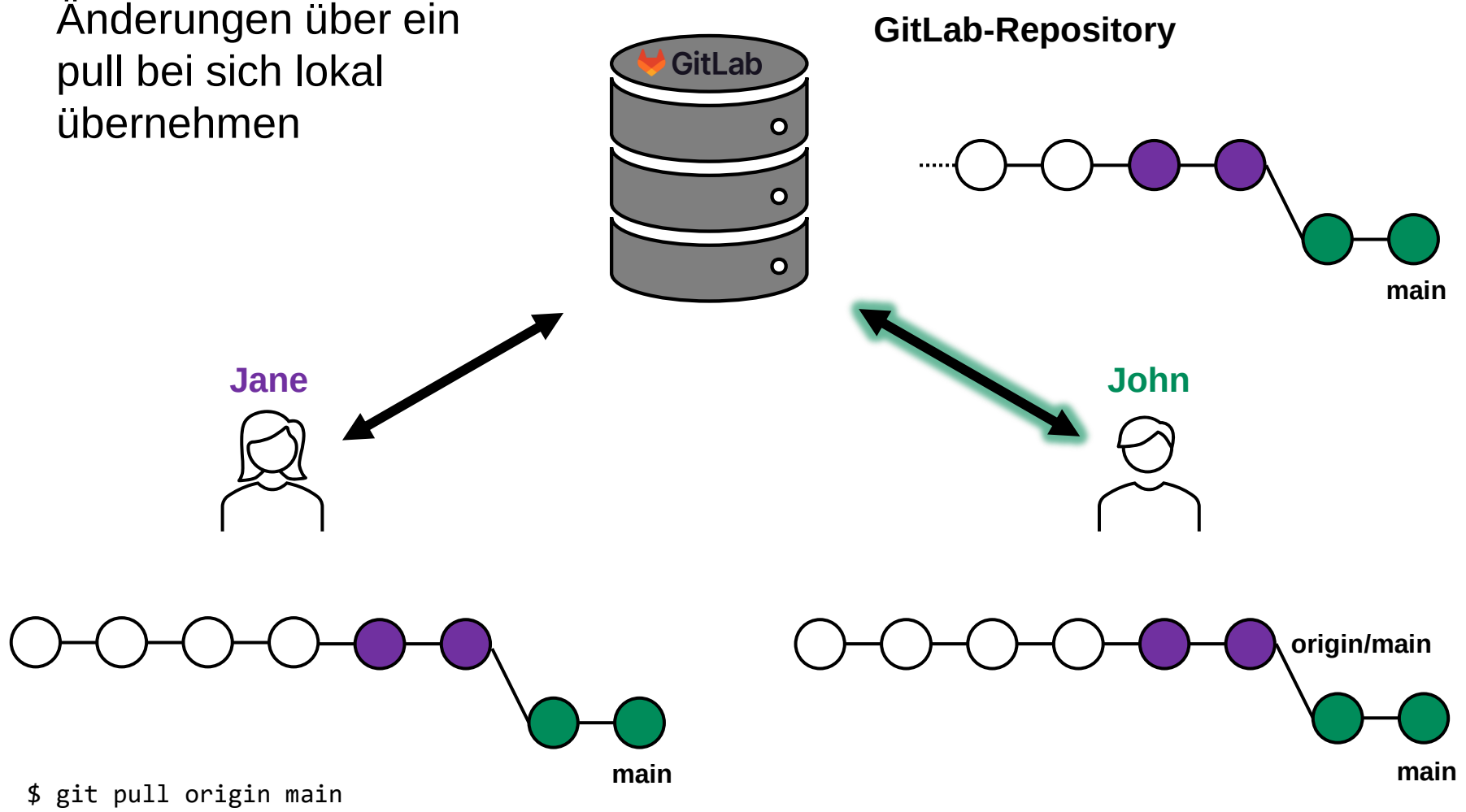
Arbeiten im zentralen Workflow

- 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 hier besser als Merging
  - → Verhindert zusätzliche Commits
- Lokal existierende Commits rebased (Remote Commits nicht)
  - → Kein Verstoß gegen Public Branch Rebasing

Git

Alternative Workflows

Alternative Workflows

- Andere Workflows = Branching-Modell komplexer
- Feature-Branch-Workflow
  - Features in eigenen Branches, nach Abschluss mergen
  - Bietet umgekehrte Vor- und Nachteile
  - Vorstellung später mit Gitflow-Workflow
- Völlig unterschiedliche Workflows
  - Forking-Workflow: Jeder Entwickler eigenes Remote-Repository
    - Projekt-Repository forken
    - Dort alleine arbeiten
    - Merge-Requests für Änderungen ins ursprüngliche Repository