dataport

anderScore
trust in competence

Tag 2: Git-Workflows, CI/CD, GitLab CI

09.07.2024, Daniel Krämer

HECKER
CONSULTING

Agenda

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

Agenda

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

Git

Workflows

Git-Workflow

Inhalt

- Was sind Git-Workflows?
- Zentralisierter 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 bieten vielfältige Einsatzmöglichkeiten
- ➔ Viele Git-Workflows mit unterschiedlichen Konzepten
- Auswahl des passenden Workflows nach bestimmten Kriterien
  - Unternehmensprozesse
  - Teamkultur
  - Teamgröße
  - Projektgröße und Umfang
- Teammitglieder müssen den Workflow kennen und produktiv integrieren
- Workflow sollte keinen unnötigen Overhead erzeugen

Zentralisierter Git-Workflow

- Auch bekannt als *trunk-based Development*
- Nur ein Branch benötigt
  - Häufig **main**, (auch **trunk** oder **master** (veraltet))
  - Änderungen als Commit
  - Keine Verwaltung weiterer Branches → weniger Overhead
- → Einfach und schnell zu verstehen
- Erleichtert Umstieg von CVCS (z.B. SVN)

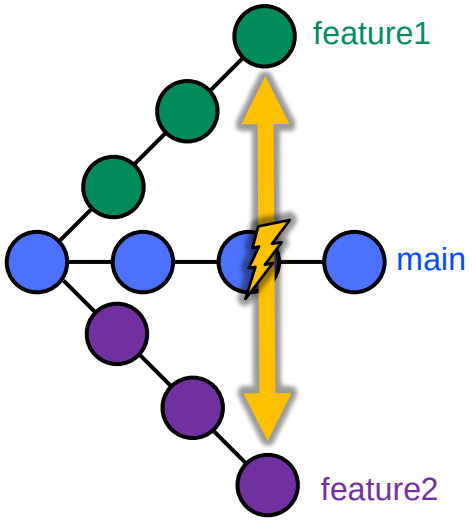
Zentralisierter Git-Workflow

- 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
- Fokus auf Commit-Qualität
  - = lauffähig und getestet
  - Schlechte Code-Qualität = großer Schaden

Zentralisierter Git-Workflow

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

➔ Verhindert Divergenz



Zentralisierter 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 und Projekten

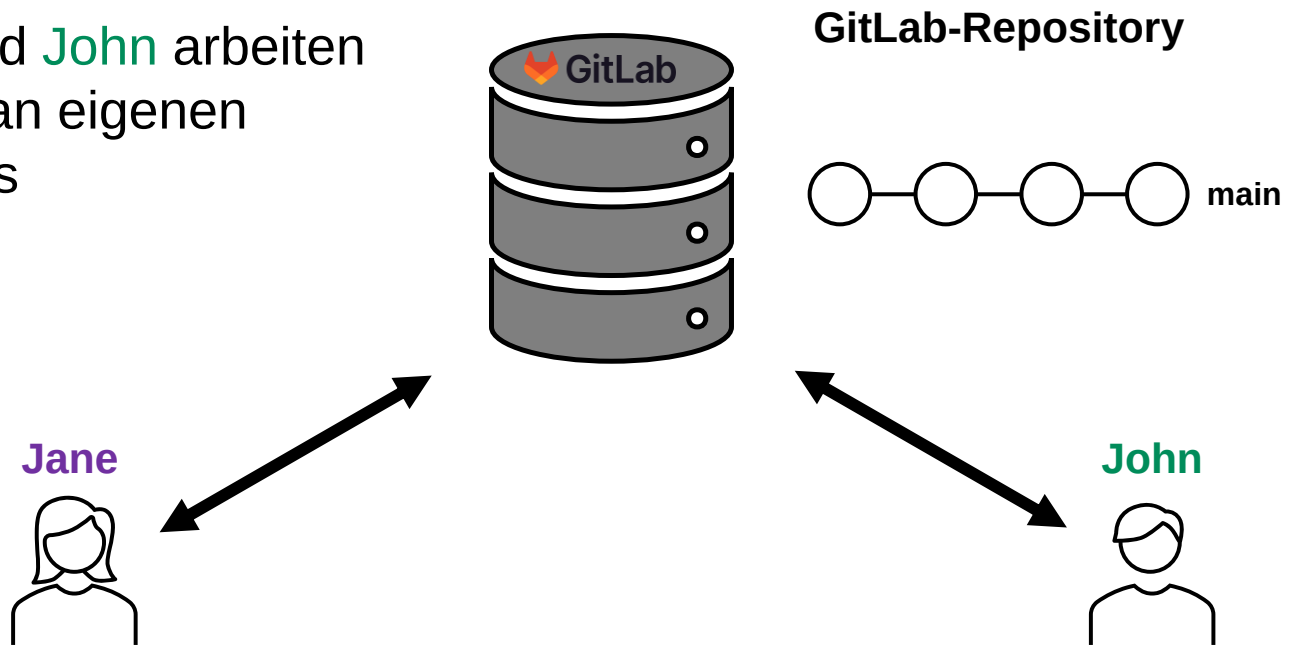
Git

Arbeiten im zentralisierten Workflow

Arbeiten im zentralisierten Workflow

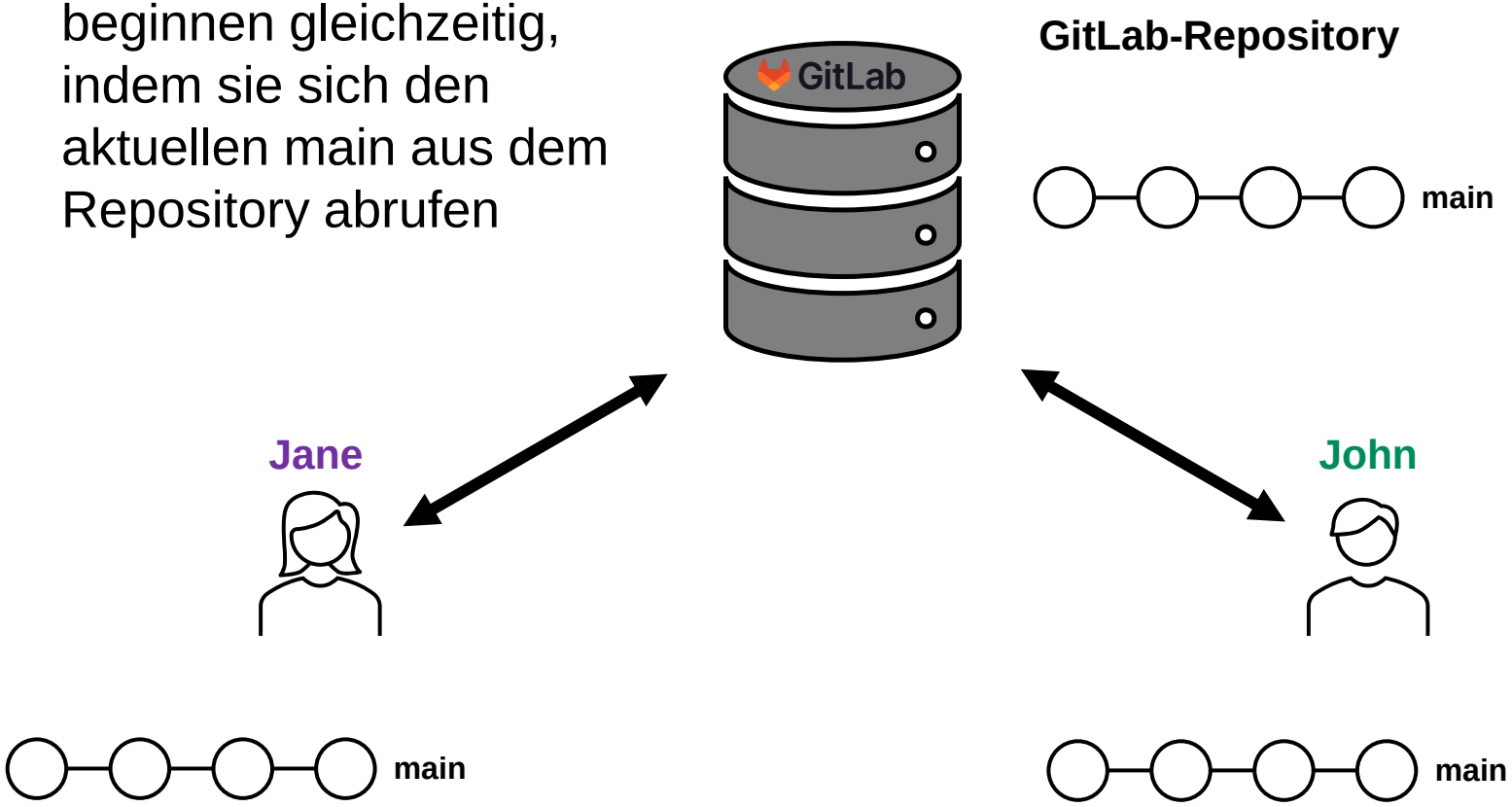
Beispielszenario

- Jane und John arbeiten jeweils an eigenen Features



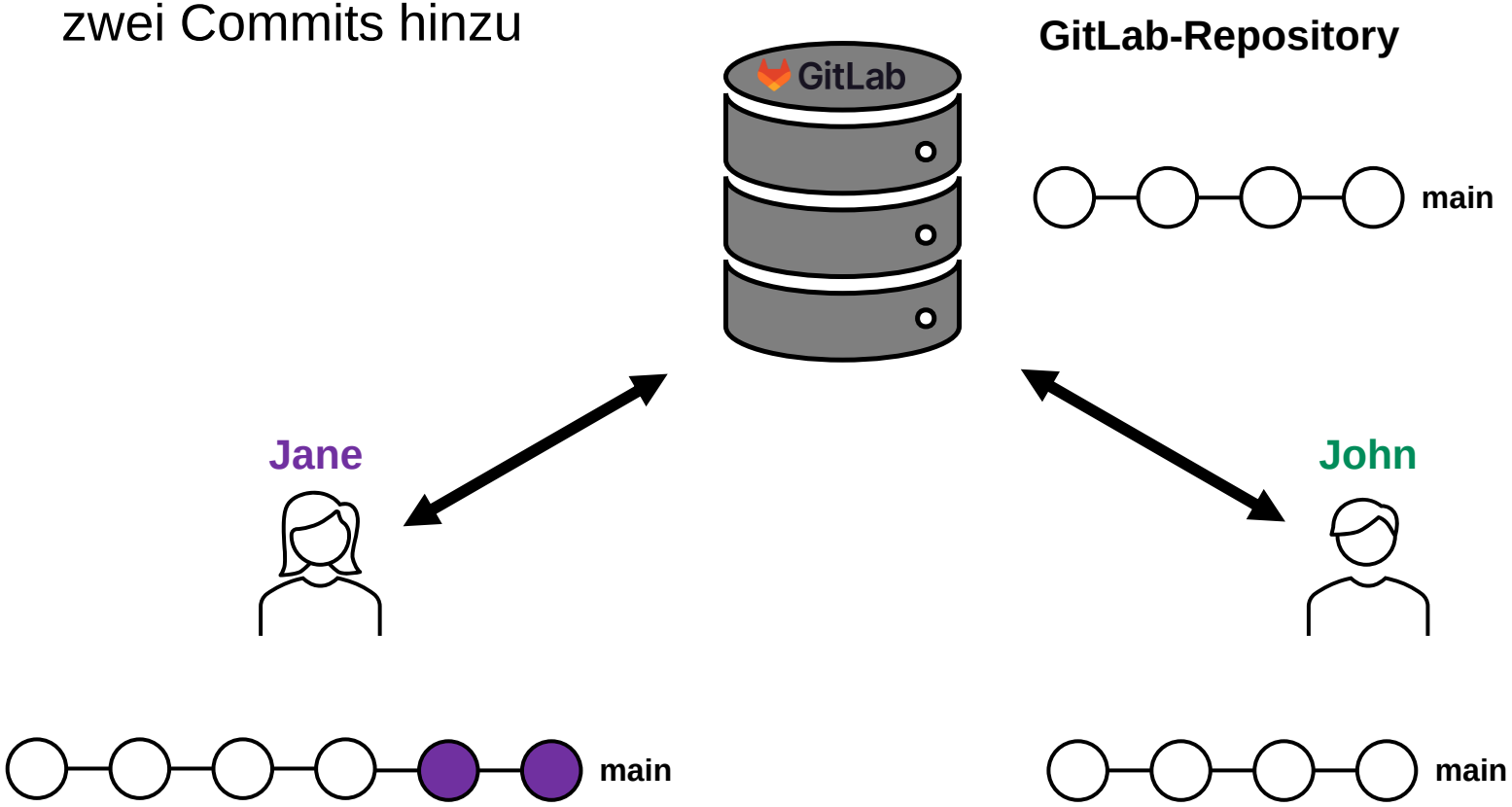
Arbeiten im zentralisierten Workflow

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



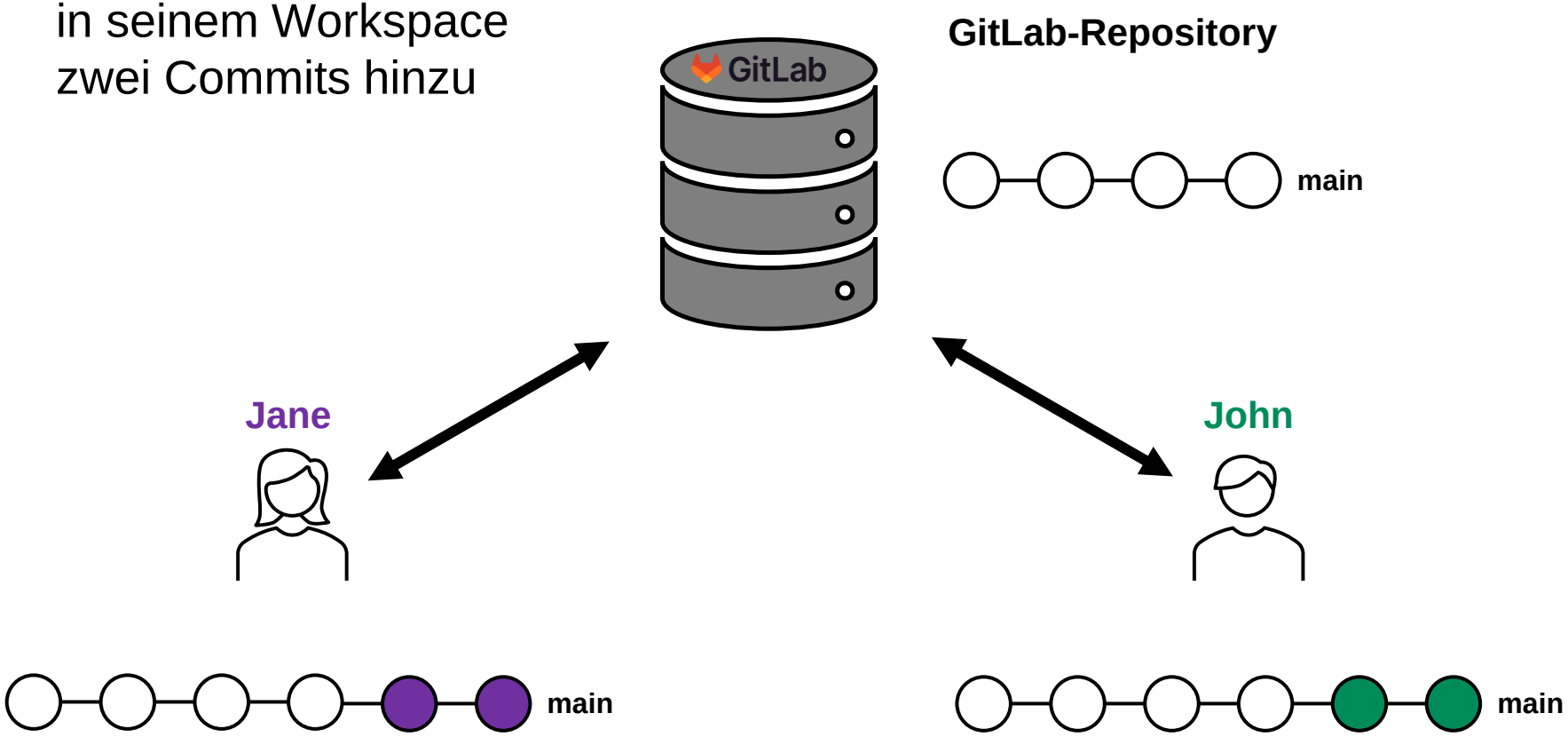
Arbeiten im zentralisierten Workflow

- Jane fügt bei sich lokal zwei Commits hinzu



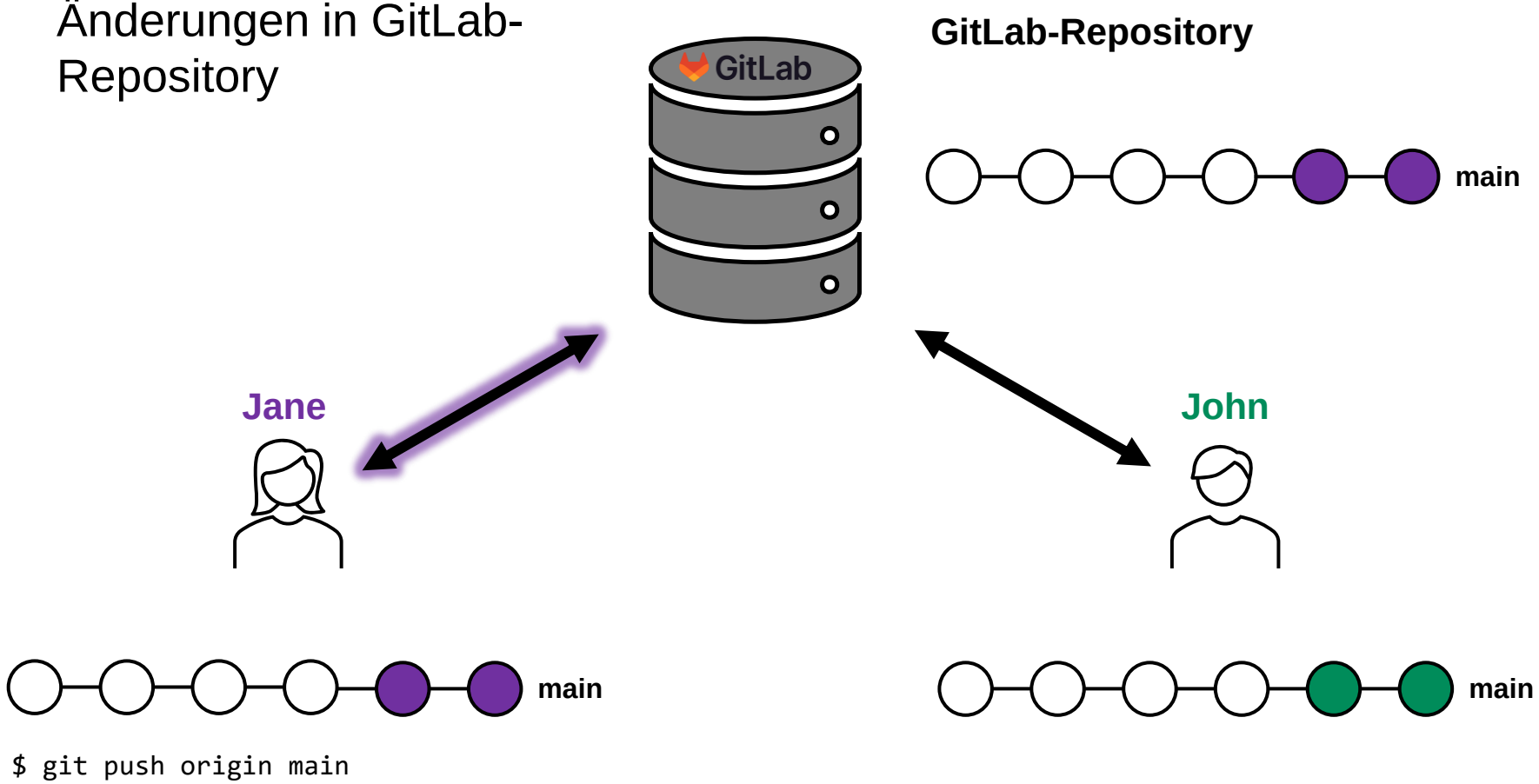
Arbeiten im zentralisierten Workflow

- John fügt ebenfalls lokal in seinem Workspace zwei Commits hinzu



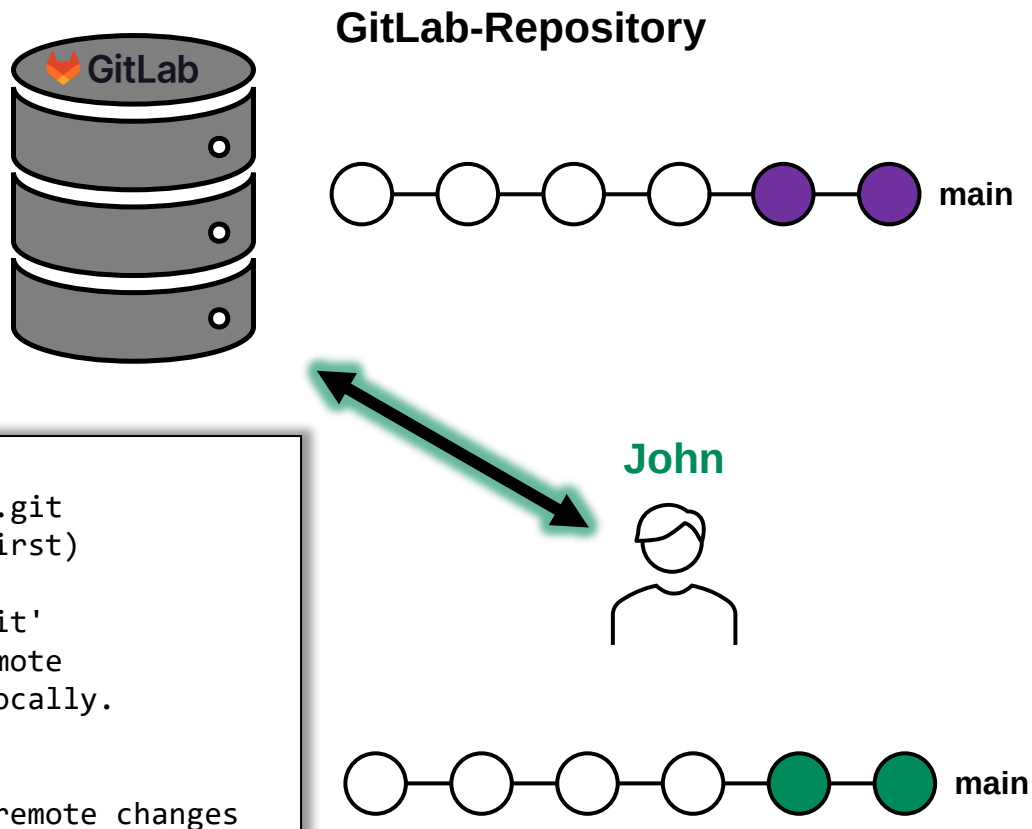
Arbeiten im zentralisierten Workflow

- Jane pusht ihre Änderungen in GitLab-Repository



Arbeiten im zentralisierten Workflow

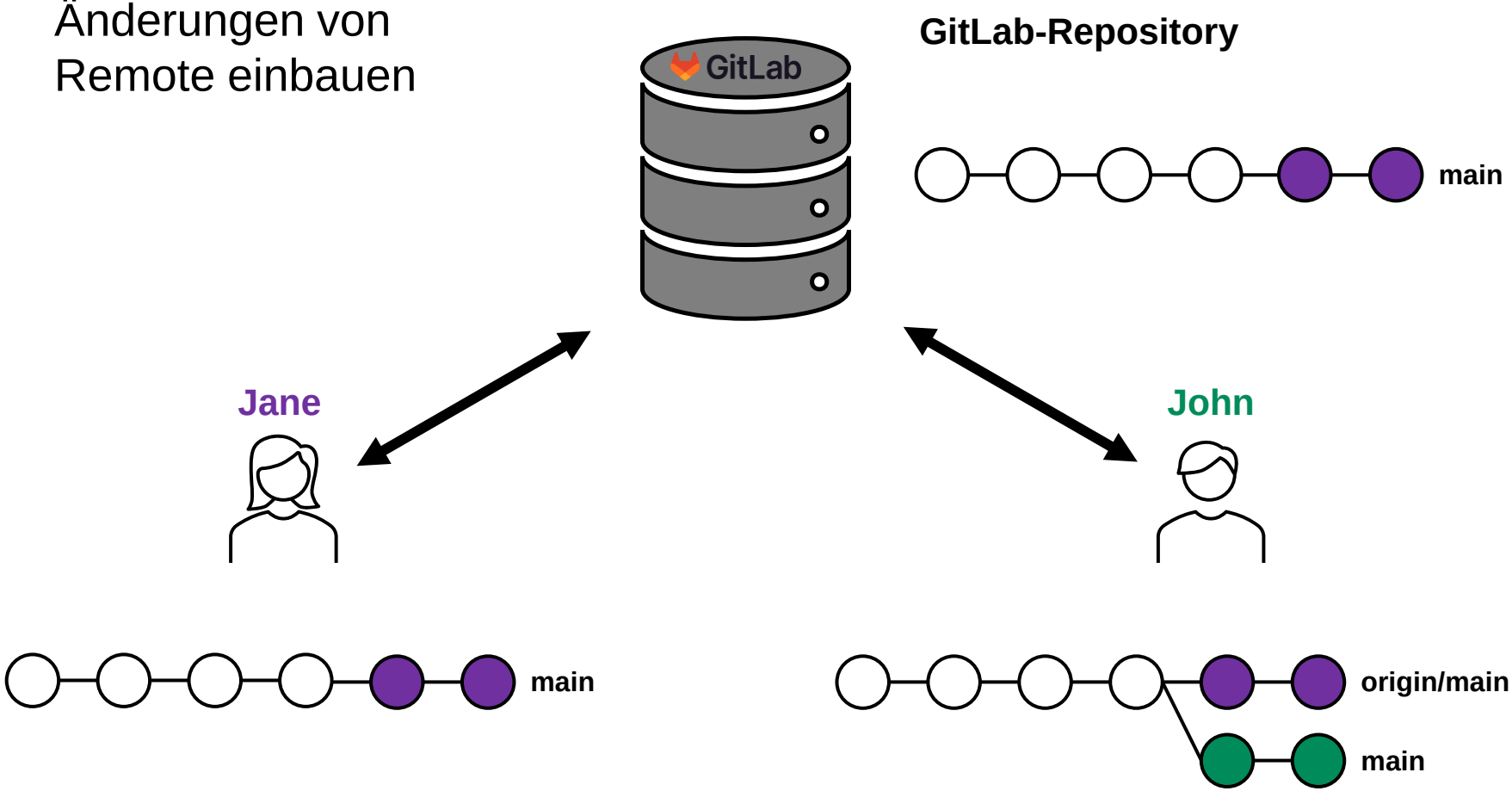
- Versucht nun John seine Änderungen ebenfalls zu pushen, erhält er einen Fehler




```
$ 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 zentralisierten Workflow

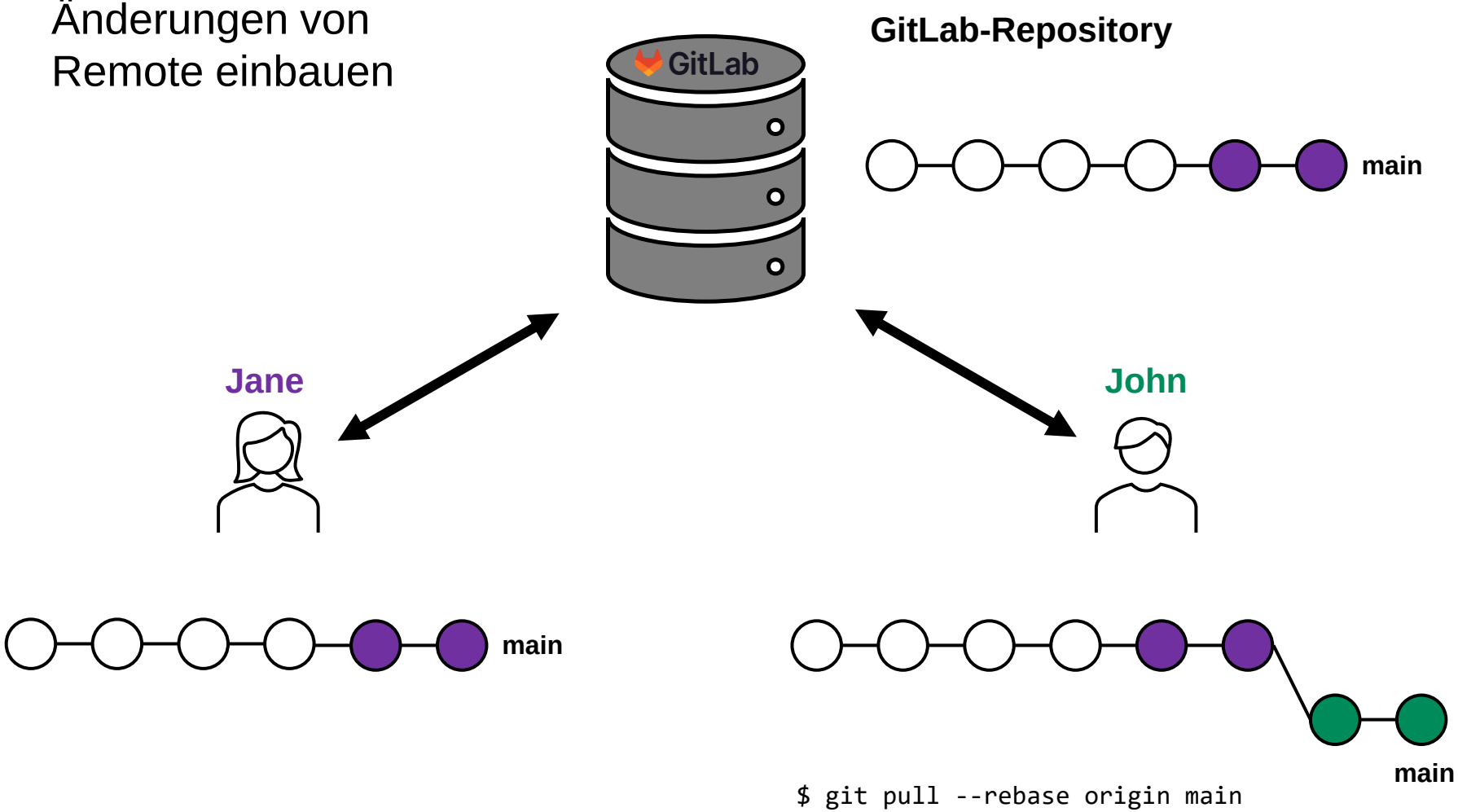
- John muss zunächst Änderungen von Remote einbauen




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

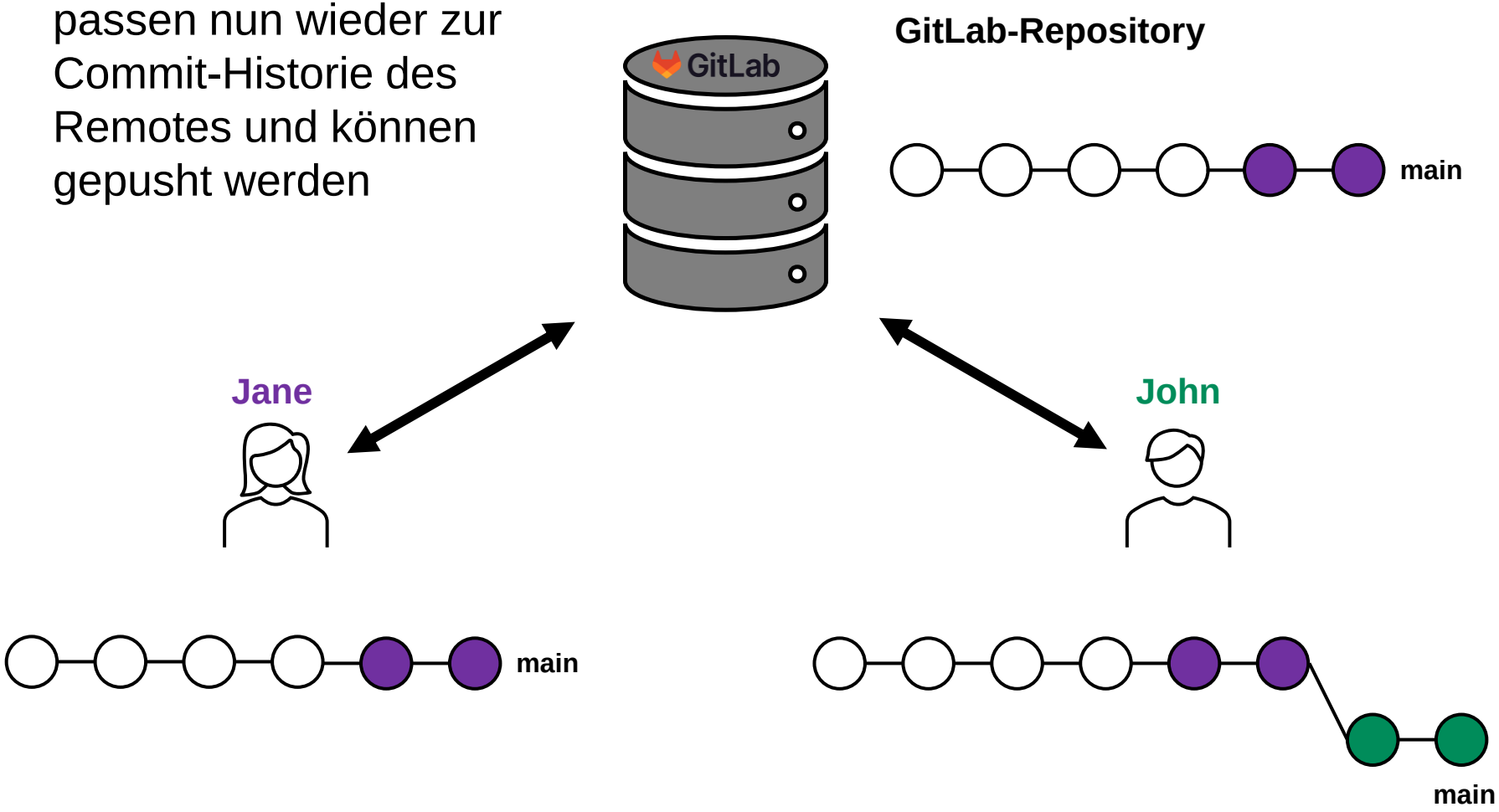
Arbeiten im zentralisierten Workflow

- John muss zunächst Änderungen von Remote einbauen



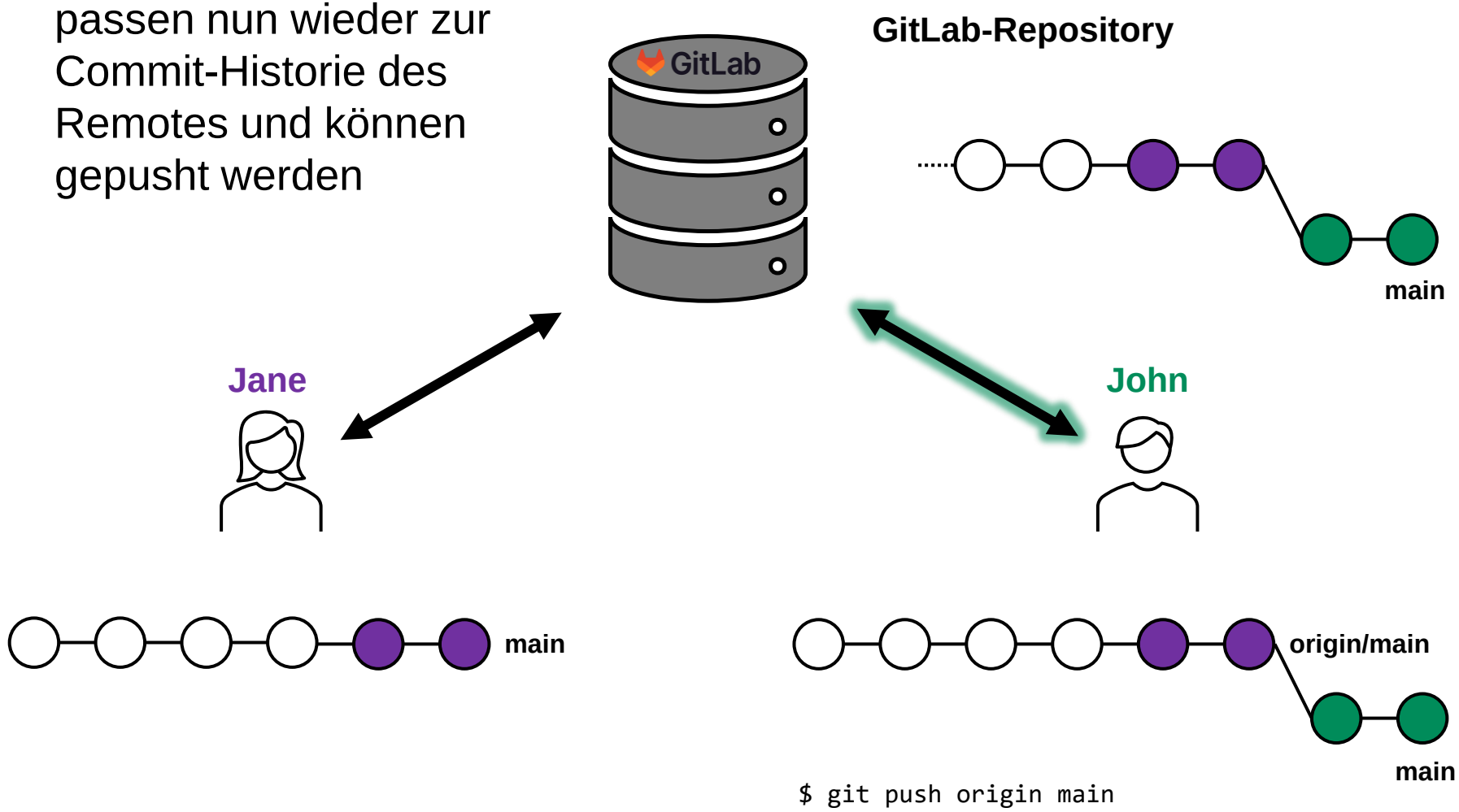
Arbeiten im zentralisierten Workflow

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



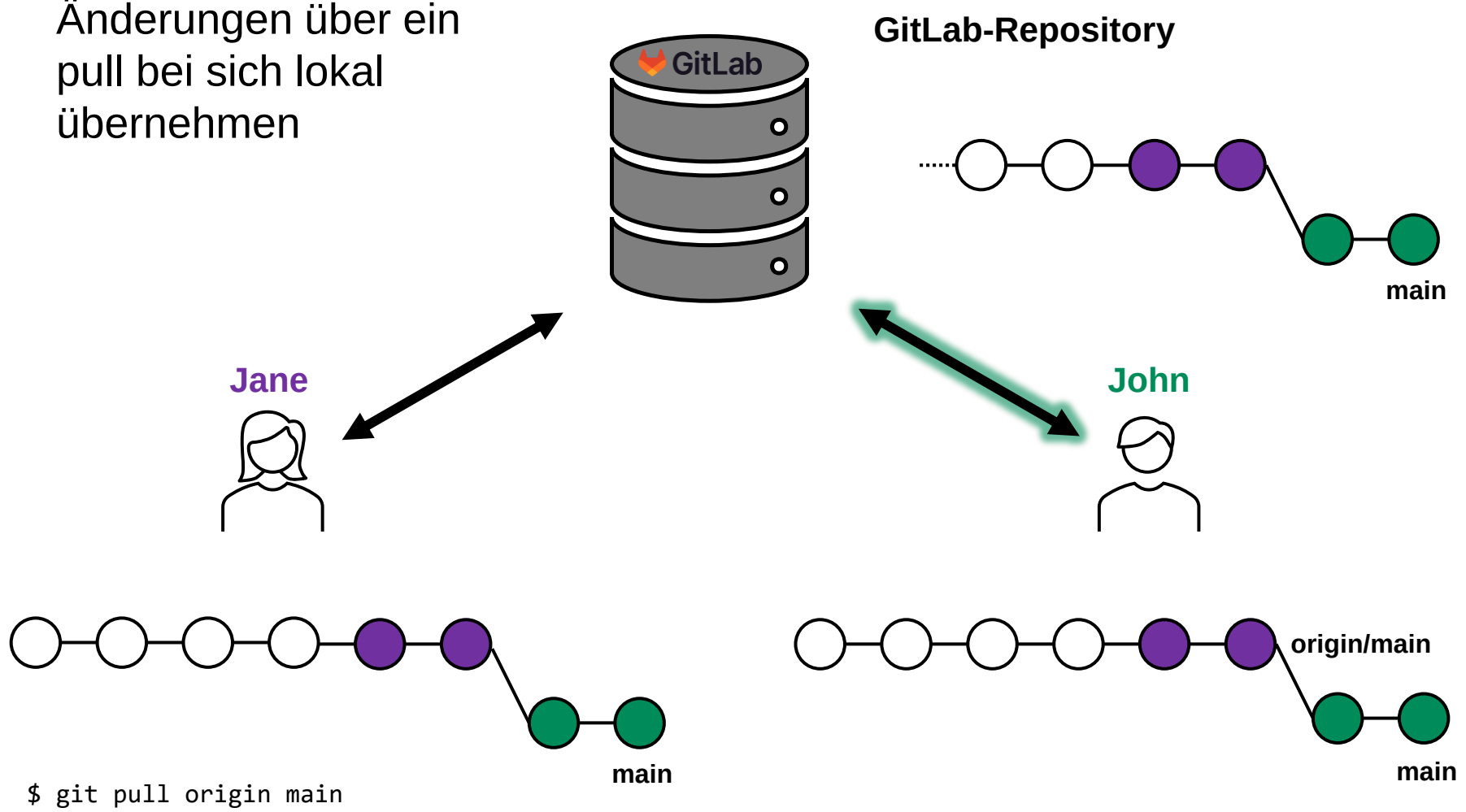
Arbeiten im zentralisierten Workflow

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



Arbeiten im zentralisierten Workflow

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



Arbeiten im zentralisierten Workflow

- Häufiges Rebasing nötig
- Rebasing hier besser als Merging
  - → Verhindert zusätzliche Commits
- Ausschließlich Rebasing lokal existierender Commits
  - → Kein Verstoß gegen Public Branch Rebasing

Git

Alternative Workflows

Alternative Workflows

- Zumeist komplexeres Branching-Modell
- Feature-Branch-Workflow
  - Features in eigenen Branches
  - Nach Abschluss mergen
  - Bietet umgekehrte Vor- und Nachteile
  - Vorstellung später mit Gitflow-Workflow
- Forking-Workflow
  - Jeder Entwickler nutzt eigenes Remote-Repository
  - Projekt-Repository forken
  - Dort alleine arbeiten
  - Pull Requests für Änderungen ins ursprüngliche Repository stellen
  - In Open Source Projekten üblich