VERSIONSKONTROLLSYSTEME IM SOFTWAREENGINEERING

GRUPPEN MITGLIEDER:

INHALTSVERZEICHNIS

INHALTSVERZEICHNIS

TEIL1: EINLEITUNG LEITFRAGE 1

Welche Vorteile bietet ein verteiltes Versionskontrollsystem im Vergleich zu einem zentralisierten Modell?

ZIELSETZUNG VON VERSIONSKONTROLLSYSTEMEN

- Änderungen nachvollziehen
- Historie & alte Versionen wiederherstellen
- Paralleles Arbeiten im Team
- Dokumentation für spätere Bearbeitung
- Zitat: „Versionsverwaltung ist ein System, das Änderungen an Dateien über die Zeit protokolliert.“ [1]

ZENTRALISIERTE SYSTEME

•Definition: Ein CVCS speichert das gesamte Projekt auf einem zentralen Server. Entwickler benötigen eine Verbindung, um Änderungen zu machen.

•Beispiele: Subversion (SVN), CVS

•Vorteile:
•Klare zentrale Kontrolle
•Einfache Rechteverwaltung
•Geeignet für kleine Teams oder Projekte mit geringer Parallelität

ZENTRALISIERTE SYSTEME

•Nachteile:
•Serverausfall > gesamte Arbeit stoppt
•Offline-Arbeiten nahezu unmöglich
•Parallelität stark eingeschränkt, da alle Entwickler auf denselben Server zugreifen

•Arbeitsweise: Commits erfolgen nacheinander auf den Server, Änderungen werden linear gespeichert

VERTEILTE SYSTEME

•Definition: DVCS speichert das komplette Repository lokal bei jedem Entwickler.

•Beispiele: Git, Mercurial

•Vorteile:
•Offline-Arbeit möglich
•Schnelle lokale Commits
•Flexible Branching-Strategien
•Hohe Ausfallsicherheit (jede Kopie ist ein Backup)

VERTEILTE SYSTEME

•Nachteile:
•Einarbeitung für neue Entwickler notwendig
•Speicherbedarf bei sehr großen Projekten

•Arbeitsweise: Jeder Entwickler kann unabhängig Änderungen commiten, testen, Branches anlegen und später in ein zentrales Repository pushen

VERGLEICH ZENTRALISIERT (CVCS) VS. VERTEILTES SYSTEM (DVCS)

Merkmale	Zentralisiert (CVCS)	Verteiltes System (DVCS)
Speicherort	Server	Lokal bei jedem Entwickler
Offline-Funktionalität	Eingeschränkt	Voll möglich
Ausfallsicherheit	Gering	Hoch
Geschwindigkeit	Netzwerkabhängig	Lokal sehr schnell
Parallelität	Begrenzt	Sehr flexible

TECHNISCHE DARSTELLUNG: GIT GRAPH

KONSEQUENZEN FÜR ARBEITSPROZESSE

- Parallele Entwicklung möglich
- Lokale Commits & Tests beschleunigen Feedbackzyklen
- Unterschiedliche Merge-Methoden (Fast-Forward, Rebase, Merge Commit)
- Weniger Absprachen notwendig als bei CVCS

ZUSAMMENFASSUNG LEITFRAGE 1

- DVCS: flexibel, offline, schnelle Commits, hohe Ausfallsicherheit
- CVCS: einfache Struktur, zentrale Kontrolle, Offline-Arbeit eingeschränkt
- DVCS ist vorteilhaft für Teams, parallele Arbeit und komplexe Projekte

TEIL2: EINLEITUNG LEITFRAGE 2

- Wie beeinflussen unterschiedliche Branching-Strategien die Softwarequalität, insbesondere im Hinblick auf Fehlerhäufigkeit, Code-Stabilität und Merge-Konflikte?

BRANCHING-GRUNDLAGEN

- Branching Strategien? Was ist das überhaupt?

  Branch=eigenständiger Entwicklungszweig VCS ohne Einfluss am Hauptentwicklungsstand

  Branching Strategien definieren damit die Regeln, wie man parallel an einem Branch arbeiten kann

- Fehlerhäufigkeit, Code-Stabilität und Merge-Konflikte? Ich versteh nur Bahnhof!

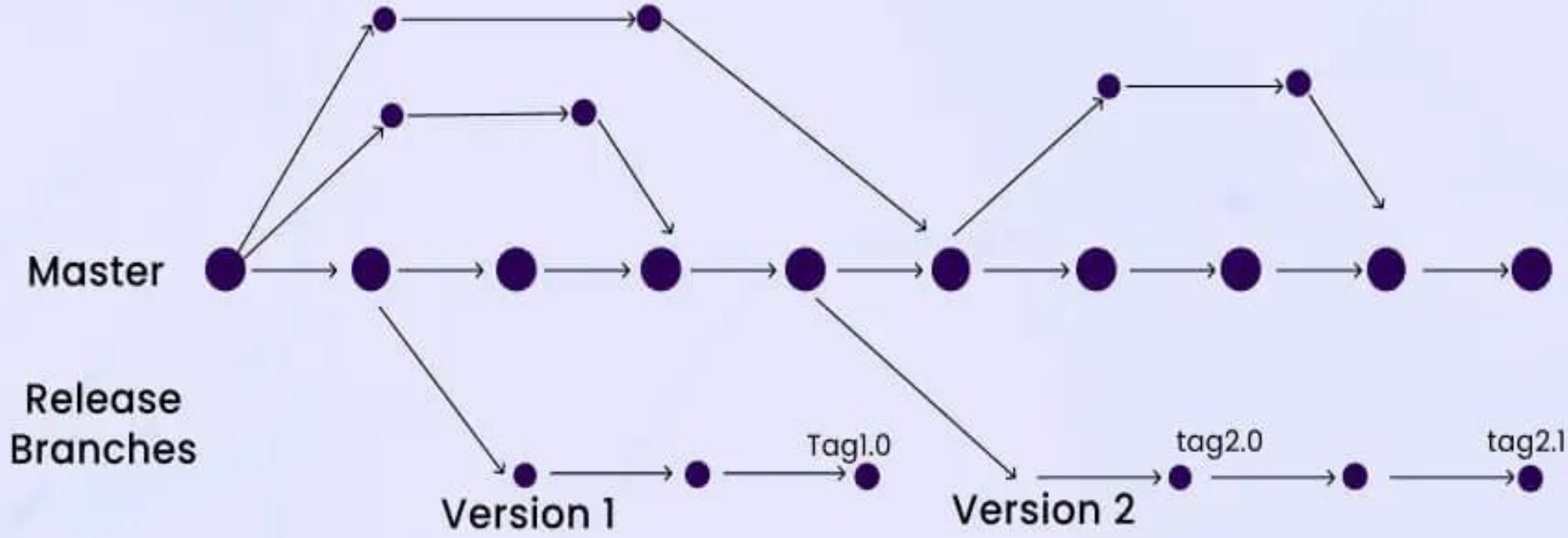
  ->Fehlerhäufigkeit: wie oft treten Softwarefehler in einem bestimmten Zeitraum auf

  ->Code-Stabilität: Wie verlässlich und vorhersehbar verhält sich Code bei Änderungen

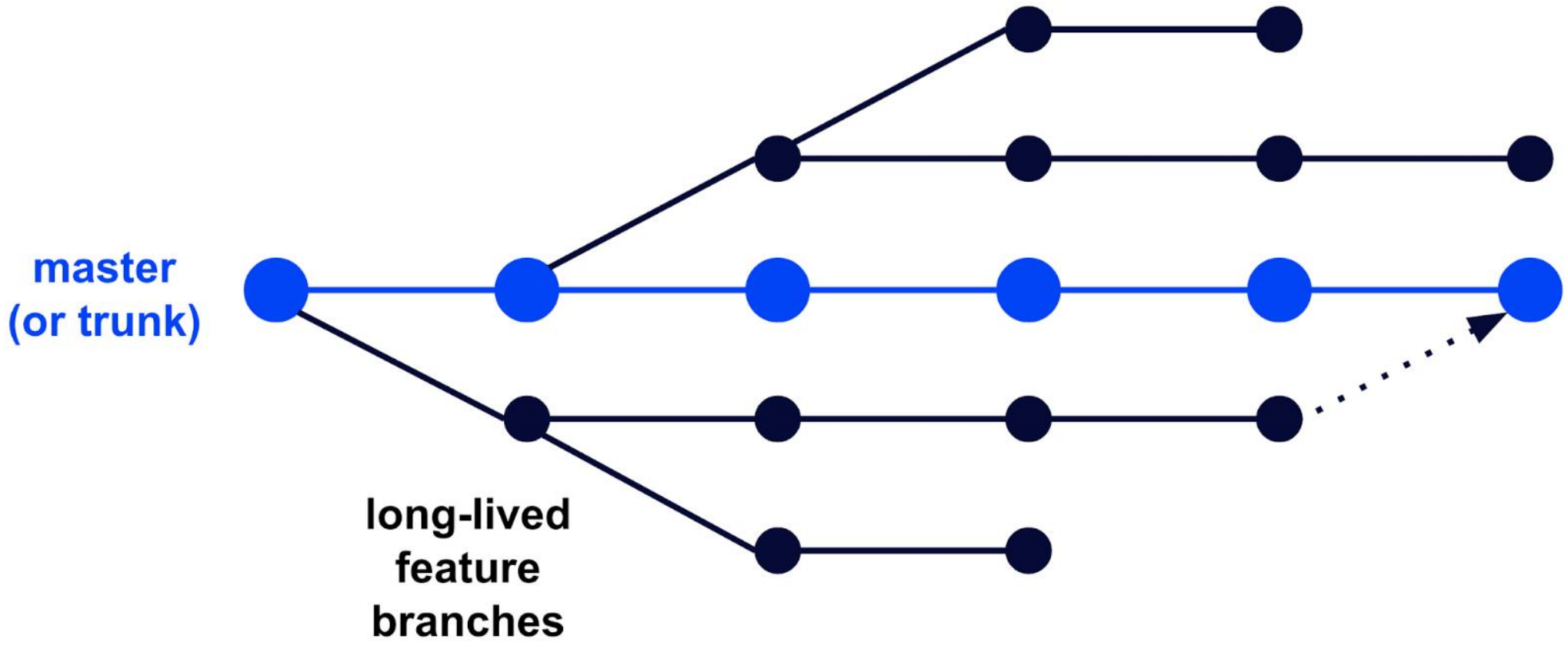
  ->Merge-Konflikte: Änderungen aus unterschiedlichen Branches können nicht automatisch zusammengeführt werden

MOON TECHNOLABS
Next-Gen Development Partner

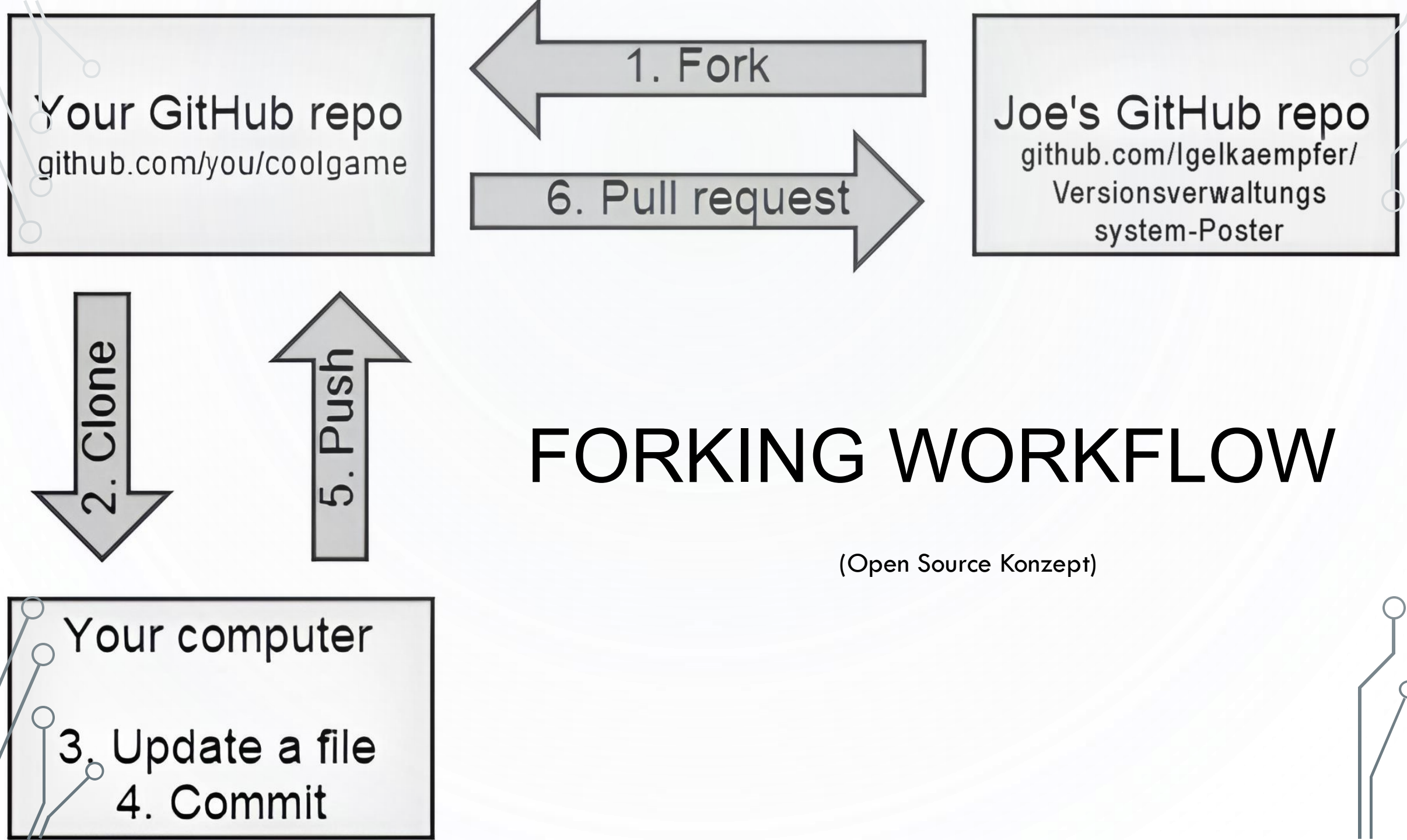
Workflow of Trunk-based Development



Feature-branched development



merging is difficult on a longer-lived feature branch



FORKING WORKFLOW

(Open Source Konzept)

AUSWIRKUNGEN AUF SOFTWAREQUALITÄT

(FEHLERHÄUFIGKEIT, CODE-STABILITÄT UND MERGE-KONFLIKTE)

TRUNK-BASED DEVELOPMENT

Fehlerhäufigkeit: Hoch

Fehler früh sichtbar

Kleine Änderung werden schneller getestet

Code-stabilität: Hoch

Kurze Branch-Lebensdauer und Automated CI Pipeline

Merge-konflikte: Niedrig

Branches, häufiges Rebased und sofortiges Auflösen

FEATURE BRANCHING DEVELOPMENT

Fehlerhäufigkeit: Mittel

Fehler oft erst beim Merge oder nach längerer Entwicklungszeit

Code-stabilität: Mittel

abhängig von Branch-Dauer und Testtiefe vor dem Merge

Merge-konflikte: Mittel

abhängig von Parallelentwicklung, Branch-Lebensdauer und Release Rhythmus

FORKING WORKFLOW DEVELOPMENT

Fehlerhäufigkeit: Hoch

Integration erfolgt verzögert

Code-stabilität: Niedrig

häufig veraltete Codebasen, umfangreiche Divergenzen

Merge-konflikte: Hoch

seltene Merges, größere Code-Divergenz zwischen Teams oder Repositories

ZUSAMMENFASSUNG LEITFRAGE 2

Trunk-Based Development am geeignetsten für kleine bis mittelgroße Teams, die eng zusammenarbeiten.

Feature Branching Development am geeignetsten mittelgroße Teams, die eng zusammenarbeiten.

Forking Workflow am geeignetsten für Open-Source-Projekte oder sehr große verteilte Teams, bei denen viele externe Mitwirkende ohne Schreibrechte am Hauptrepository arbeiten.