



integrata
cegos

Git

Scratch zum Webinar vom 9.-10.1.2020

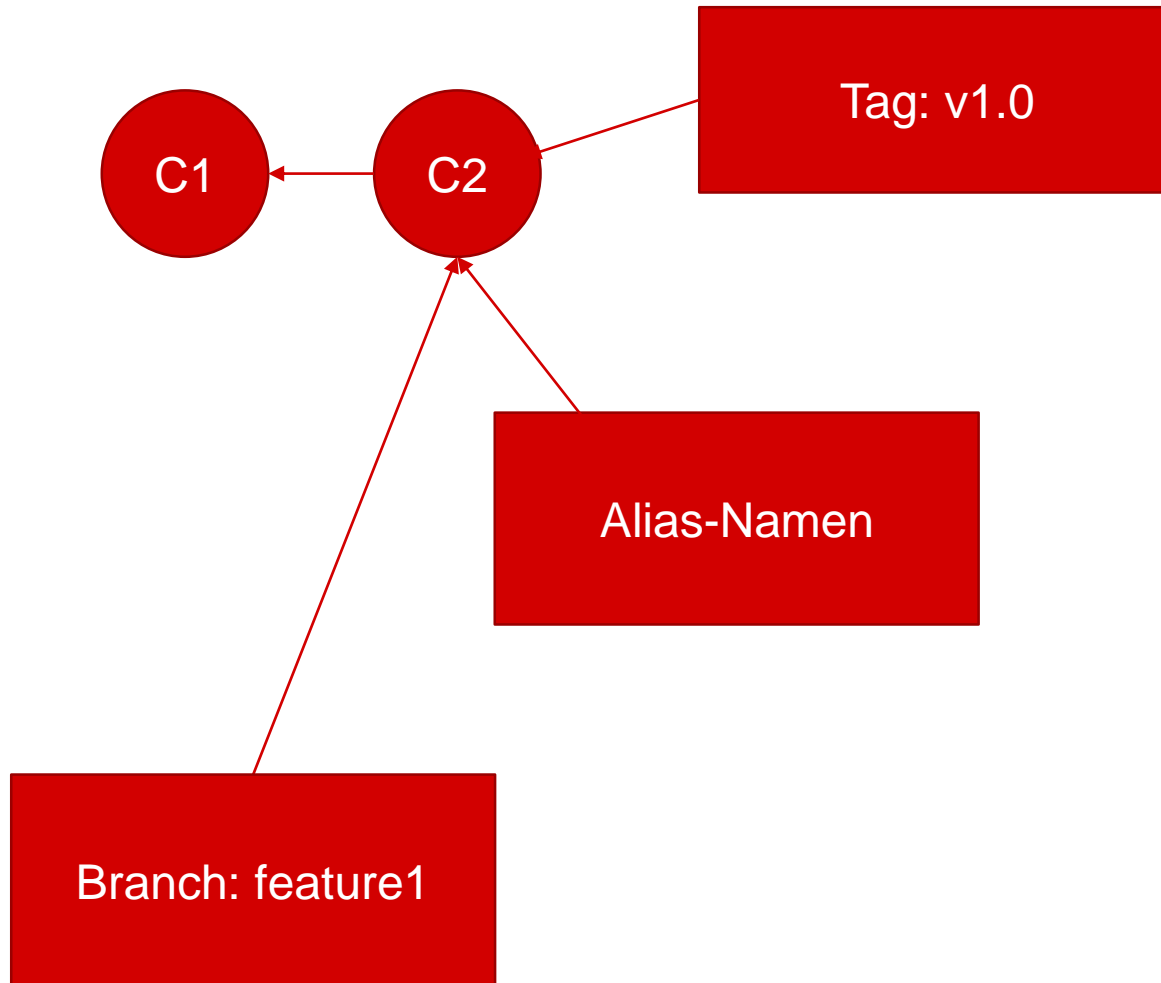
- Referent
 - Rainer Sawitzki
 - training@rainer-sawitzki.de
- 4 Sessions, jeweils 2,5 Stunden
 - Inklusive 15 Minuten Pause
- Ablauf
 - Vortrag
 - Etwa 45 Minuten
 - Präsentation
 - Etwa 45 Minuten
 - Zur Auflockerung kleinere Übungseinheiten
 - Jede 5 – 10 Minuten
 - Fragerunde
 - Nach Bedarf
 - Am Ende, etwa 15 Minuten

- Dieser Scratch
- Die Präsentation
- Online
 - <https://github.com/Javacream/org.javacream.training.gitscm>
 - Ein paar Beispiele und Skripte auf GitHub
 - <https://git-scm.com/download/win>
 - Download Git Portable
 - <https://git-scm.com/docs>
 - Dokumentation

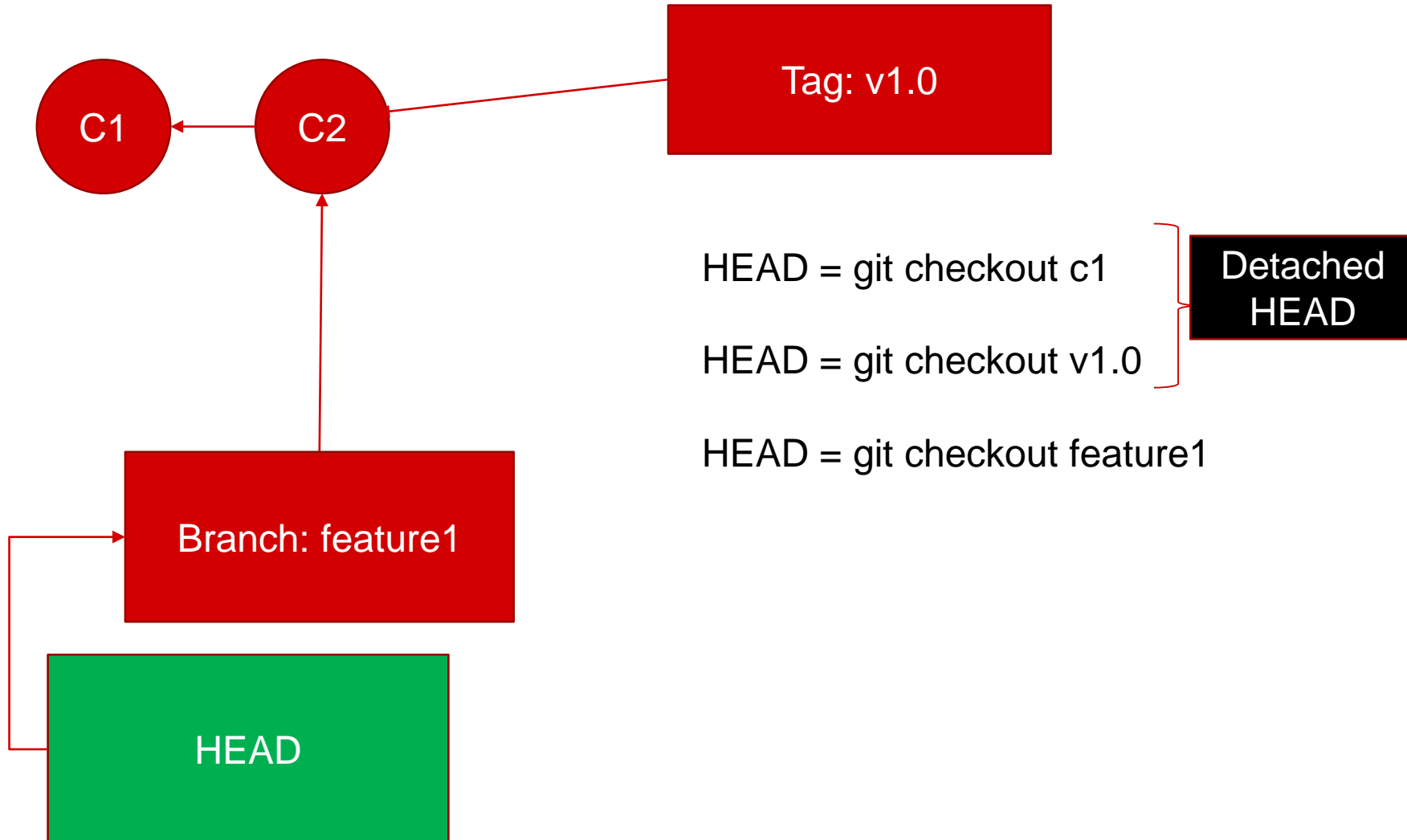
- Normales Verzeichnis
- + .git => Git Project Directory
- .git
 - Das Git Repository
- Alles andere: Workspace
- Git Repository selbst:
 - Staging Area
 - Internes Repository
 - Stashes

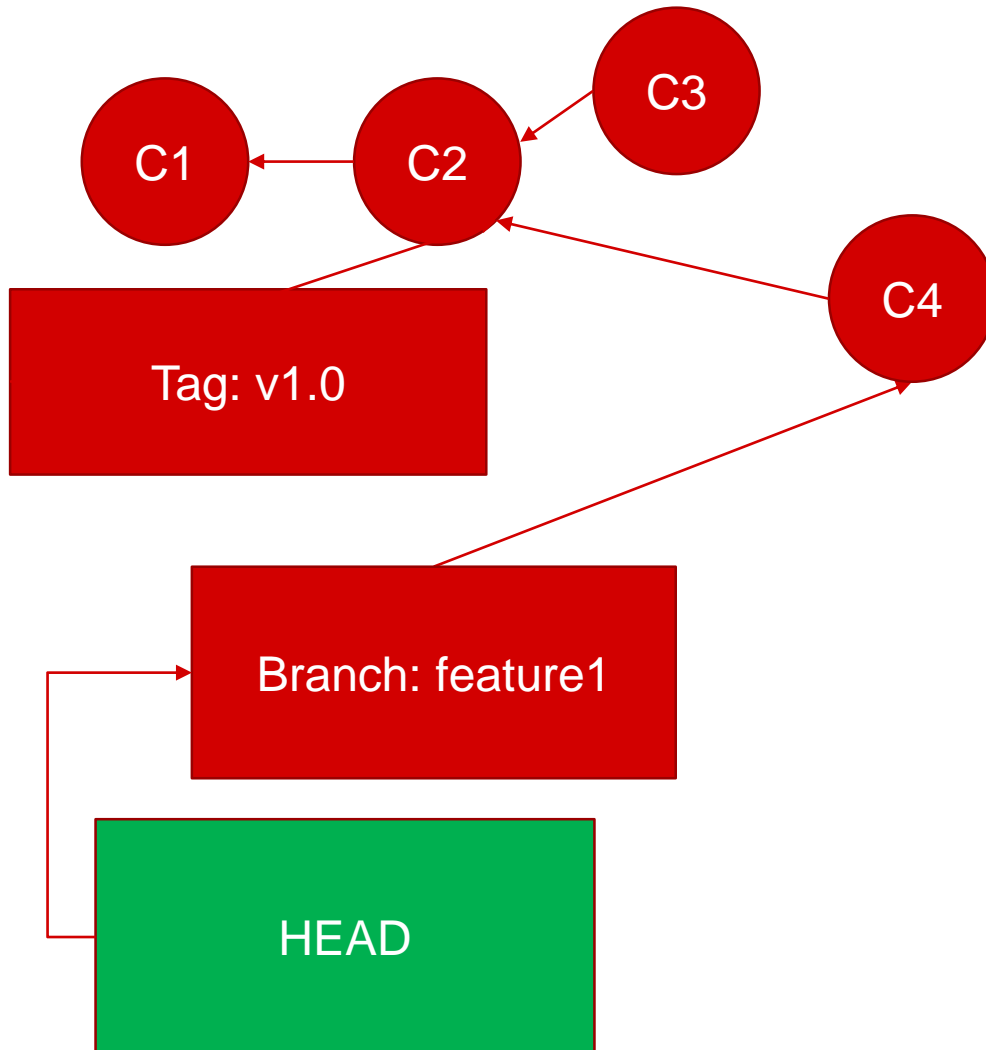
- Content-Objekte
 - Dateien
 - Tree, +++
- Commit-Objekte
 - Liste aller Content-Objekte des aktuellen Standes
 - Autor, Timestamp, Commit-Message

- git
 - config –global
 - User, eMail
 - init
 - add
 - Workspace – Staging
 - Commit
 - -m <Commit-Message>
 - Commit-Objekt wird erstellt
 - Und über Hash identifizierbar gemacht
 - status
 - Aktuellen Stand und Diskrepanzen
 - log
 - Aktuelle Commit-Objekte
 - Hash
 - Autor
 - Date
 - Commit-Message



- Unverrückbare, feste Projektstände
 - Substantive
 - Versionsnamen, v.1.0
 - Tag
 - Git checkout c2
 - Git tag v1.0
 - Git tag –list
 - Git checkout v1.0
- Aktuell laufende Aktionen
 - Verb
 - ToDos, Jira-Ticket
 - Branch
 - Git checkout c1
 - Git branch feature1
 - Git checkout feature1
 - Git branch --list

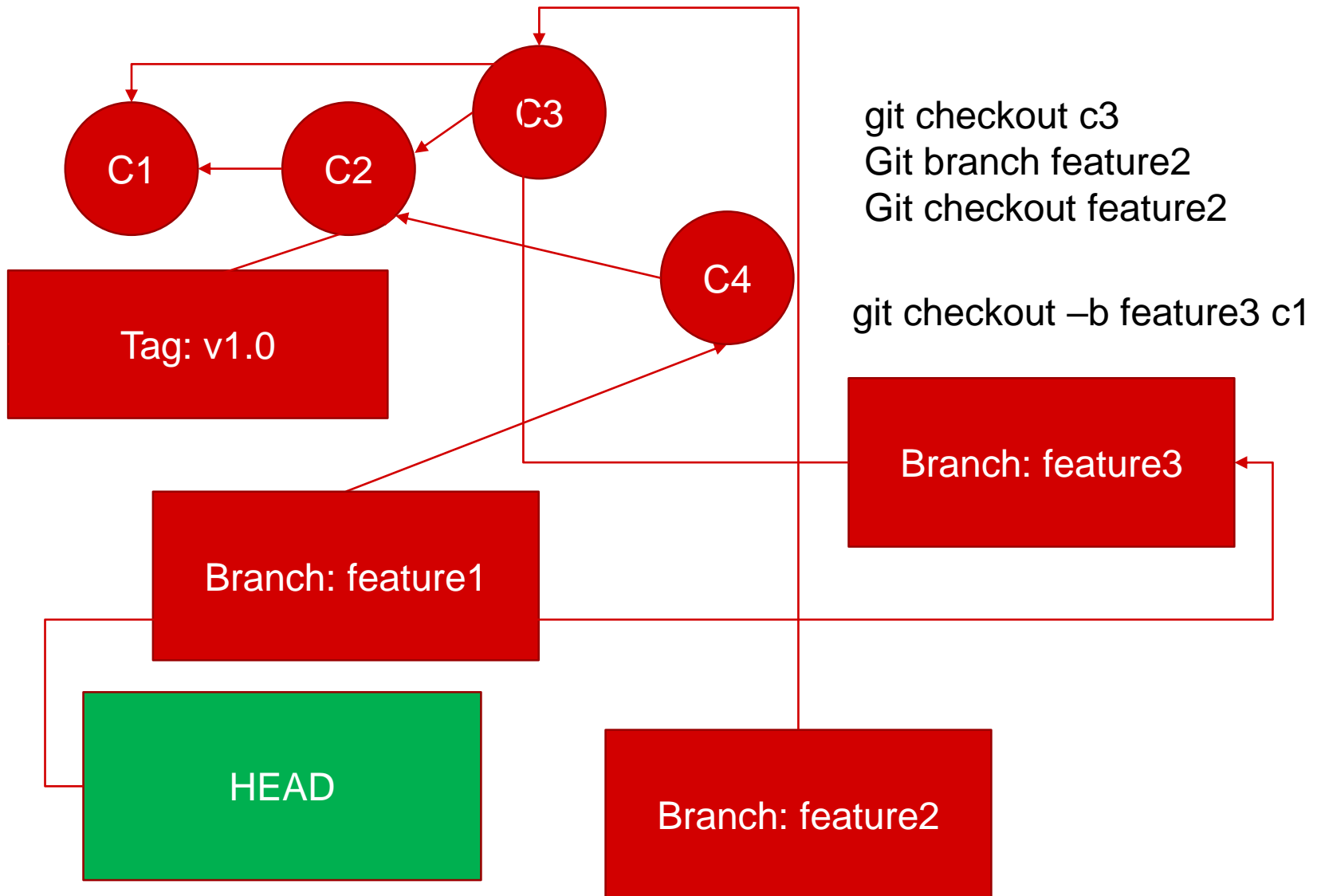


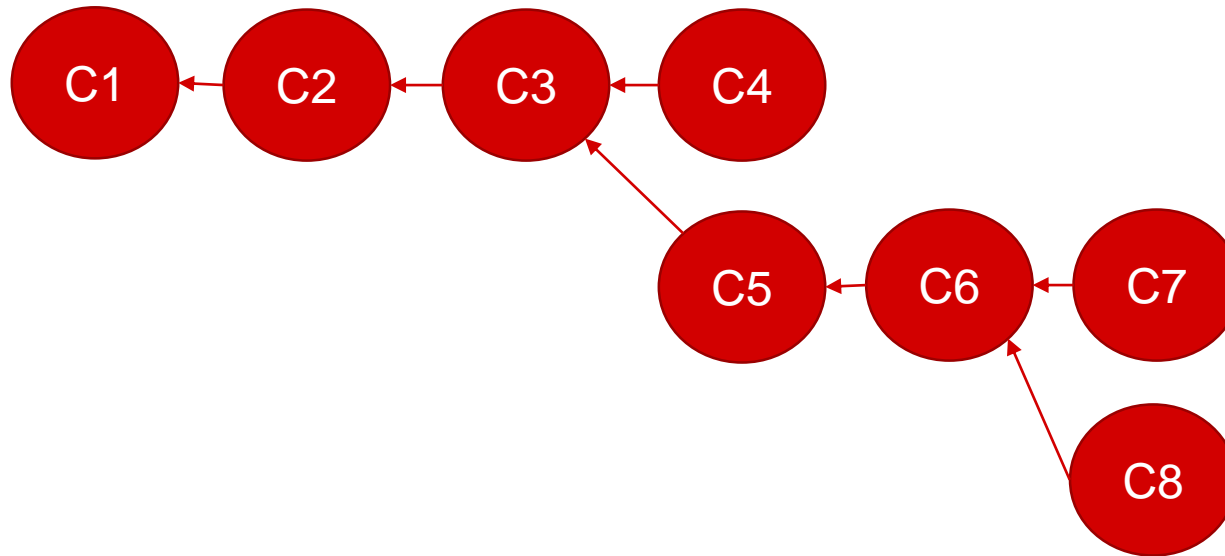


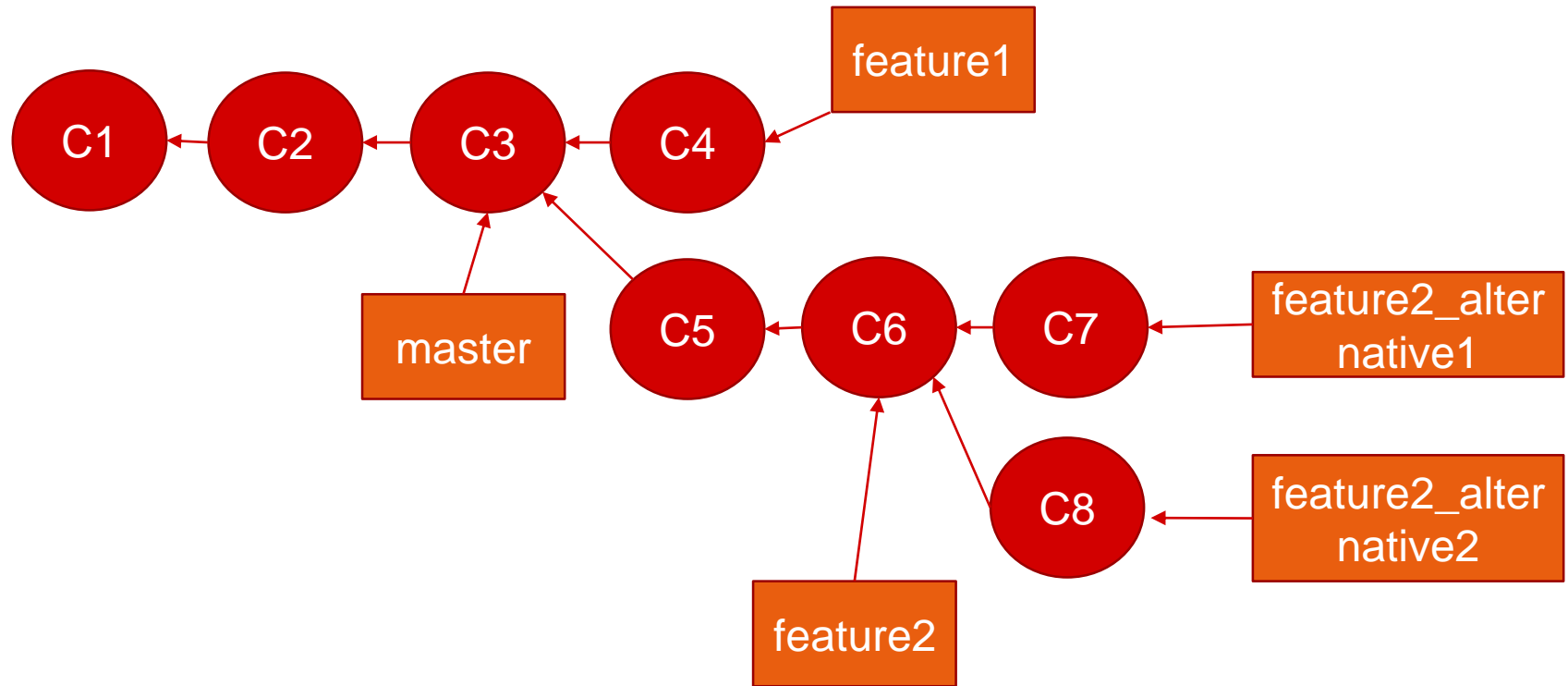
git checkout v1.0
//Changes
Add + commit

git checkout feature1
//Changes
Add + commit

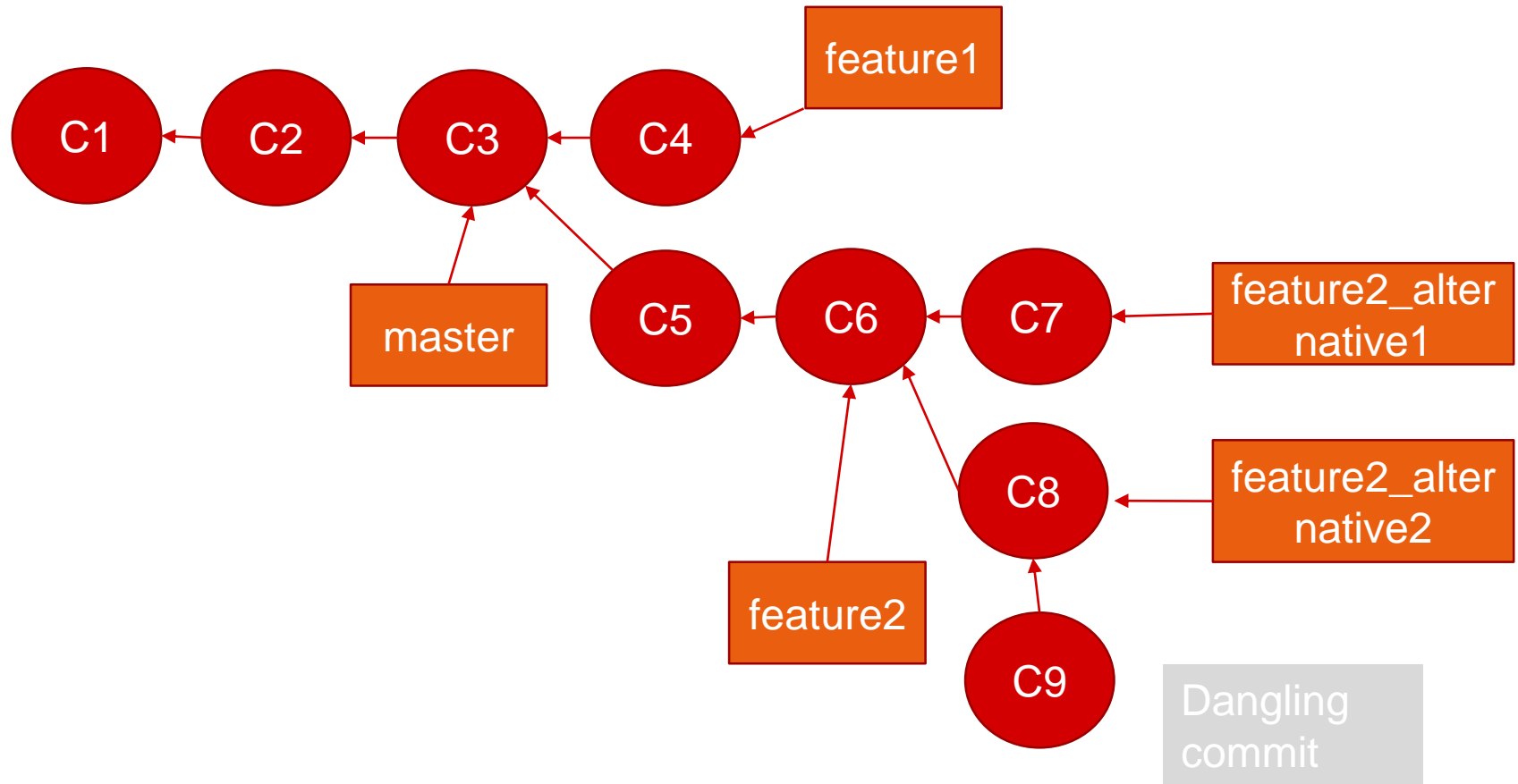
Commits und commit





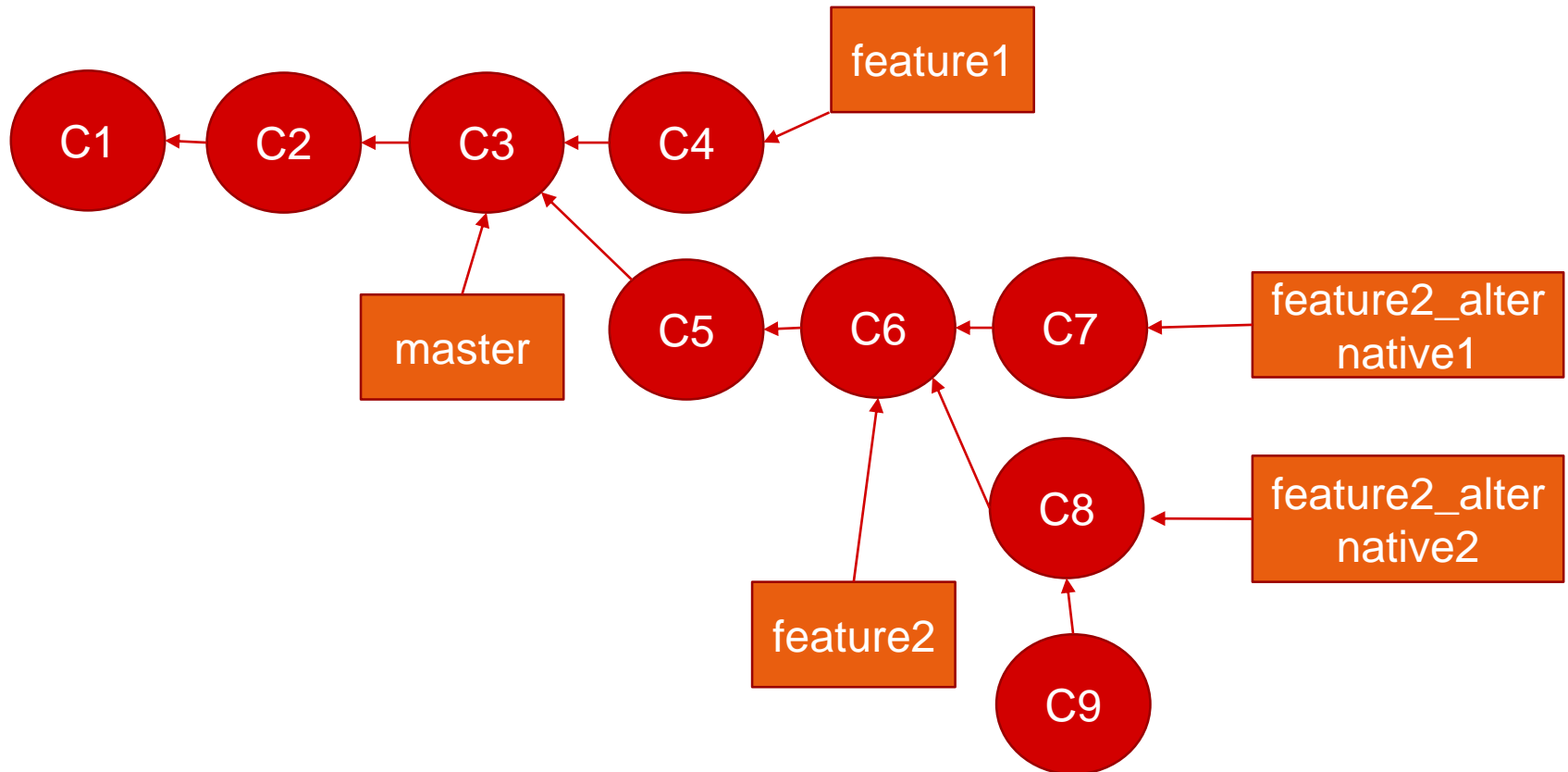


Dangling

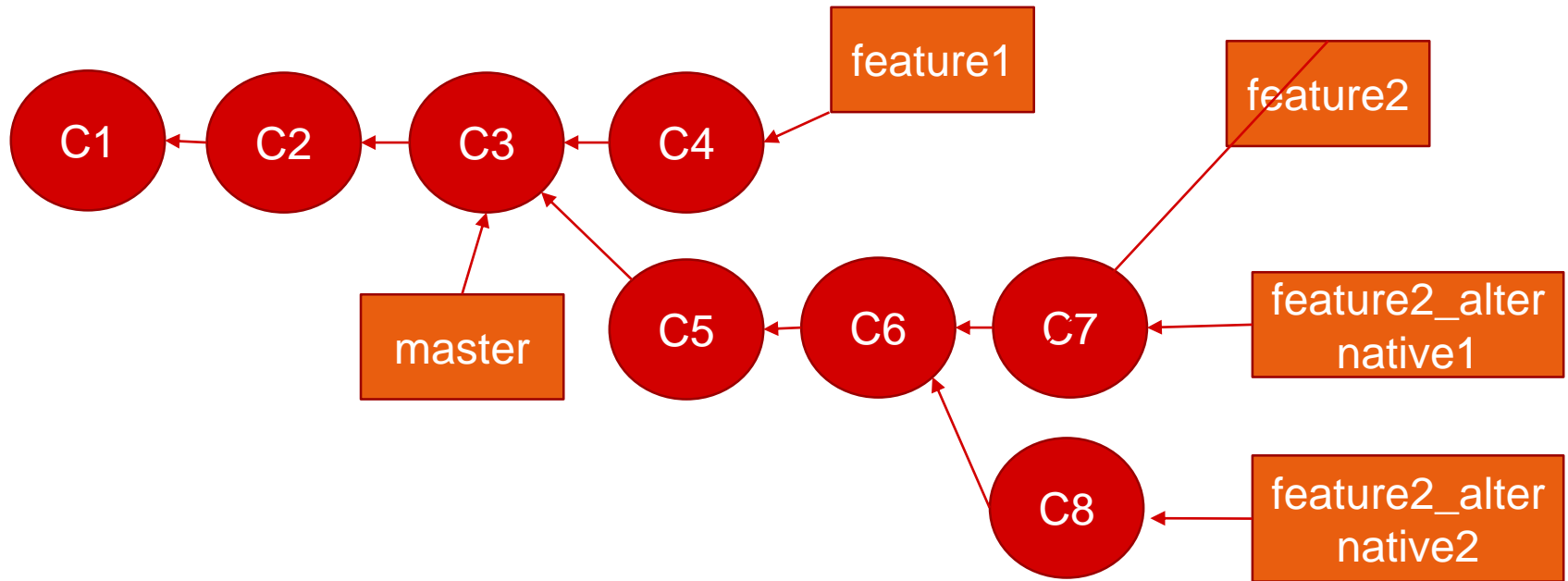


- Merge
- Rebase
- Cherry Picking
 - Aktuell in der Community eher als Anti Pattern diskutiert
- Interactive Rebasing

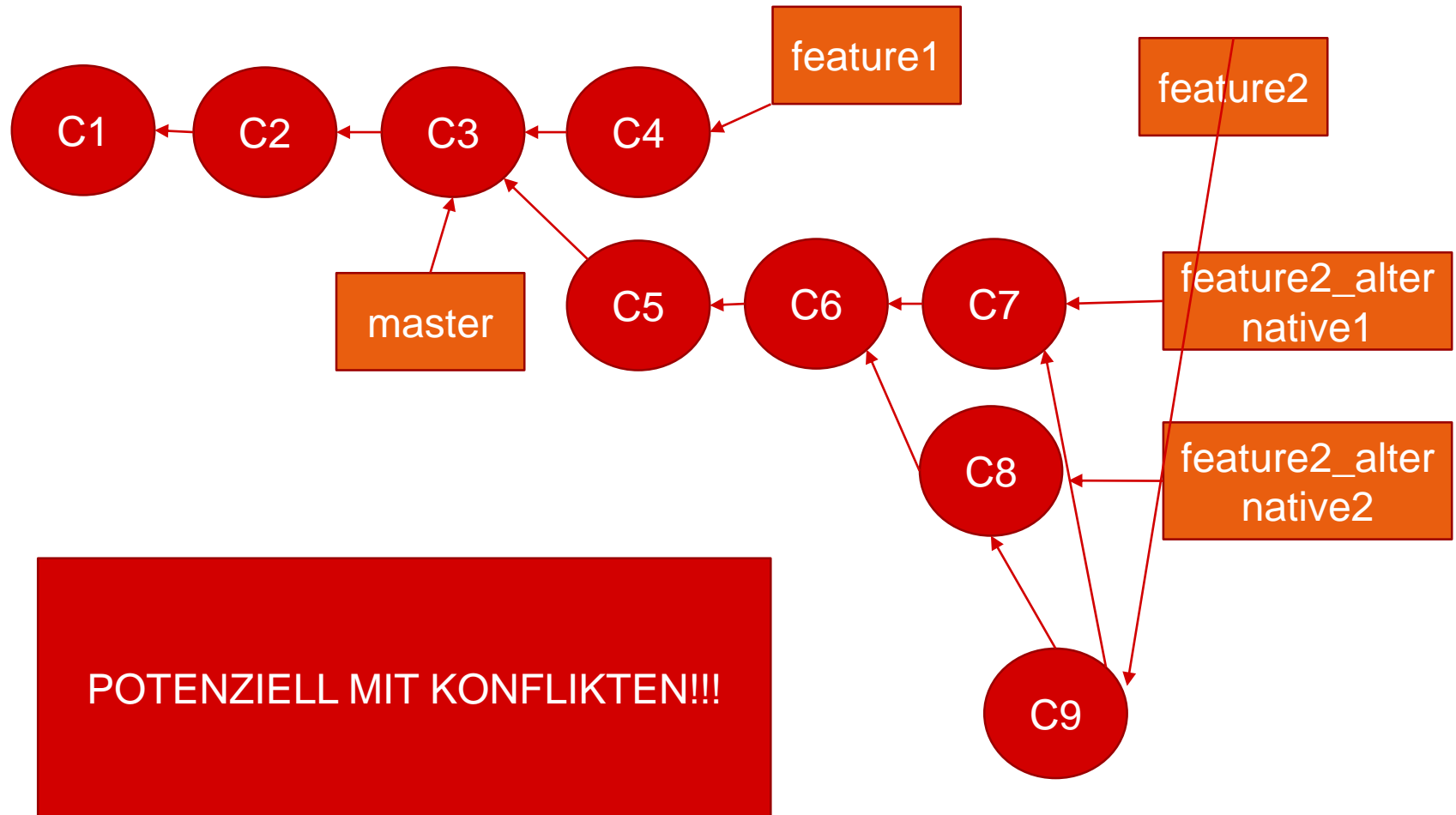
Merge



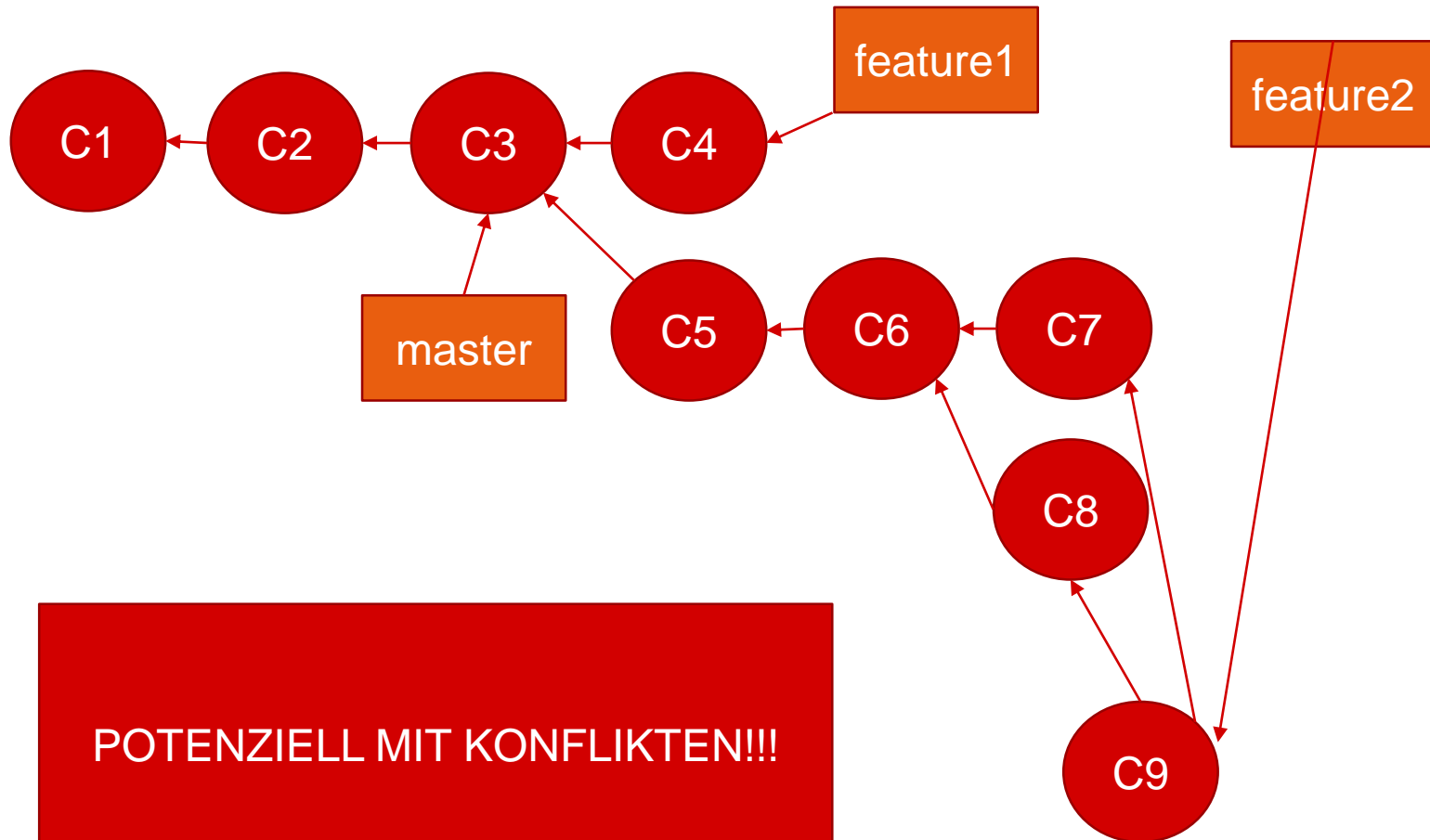
Merge feature 2 mit alt1: fast Forward



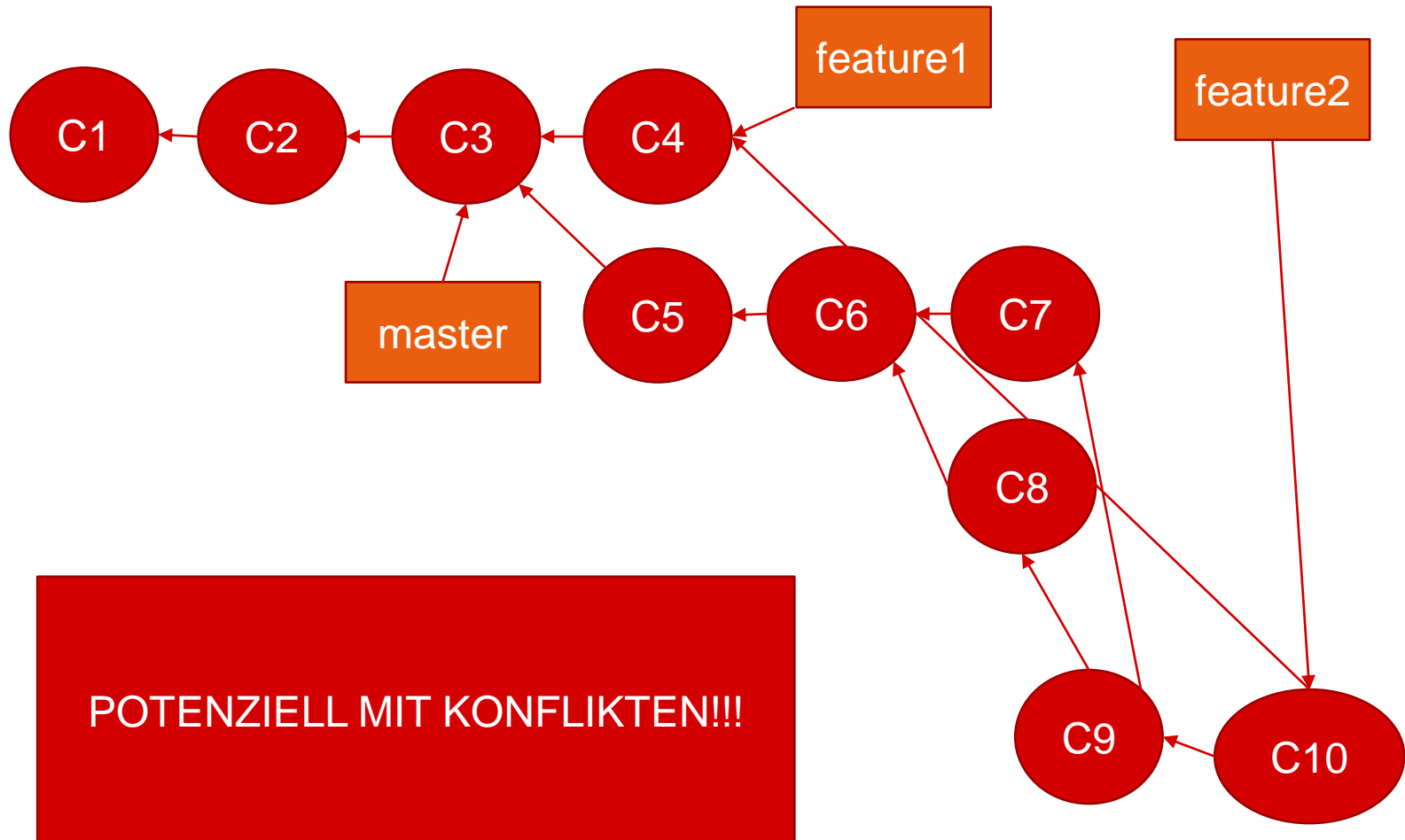
Merge feature 2 mit alt2: Recursive Merge



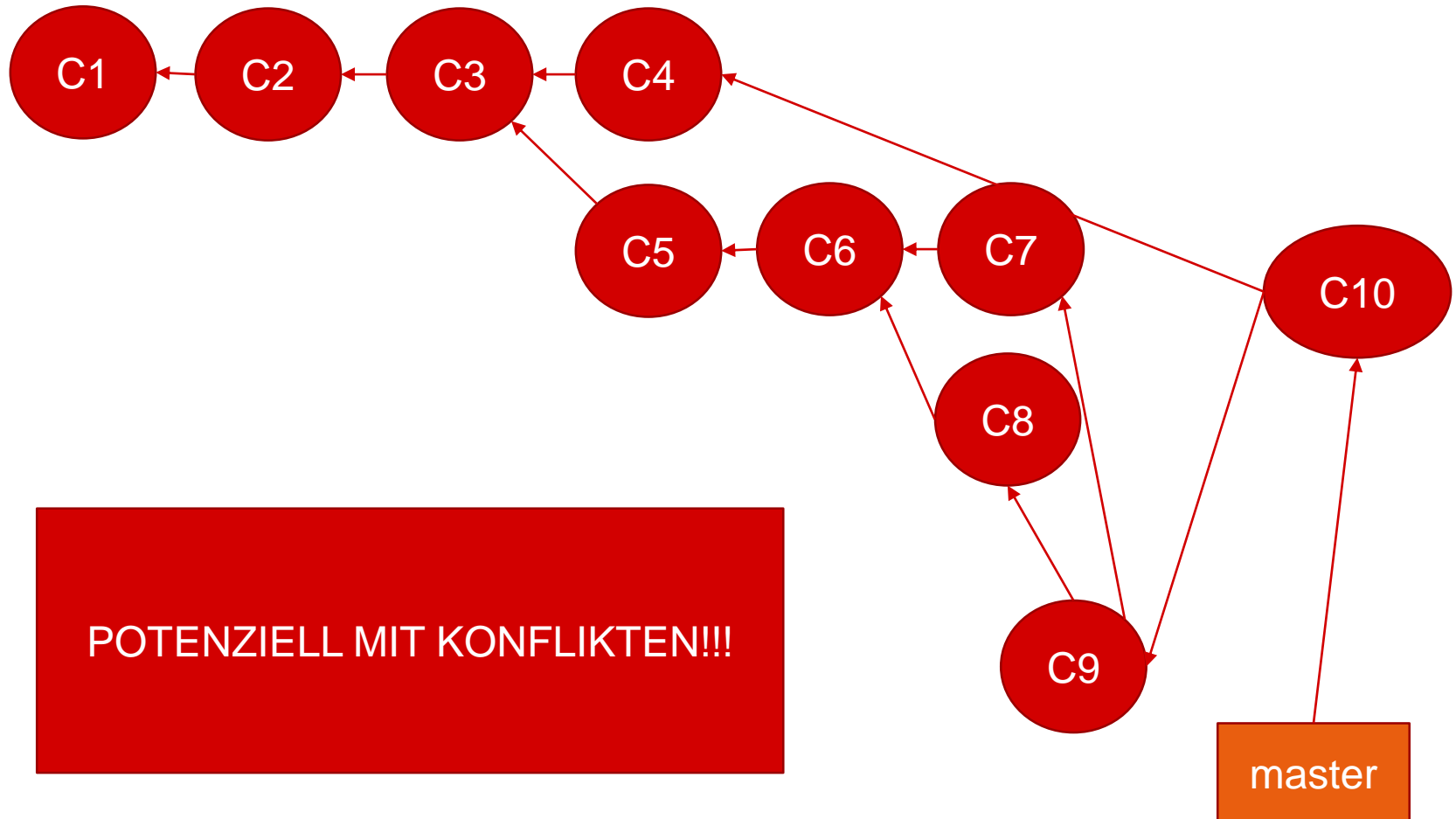
Merge feature 2 mit alt2:



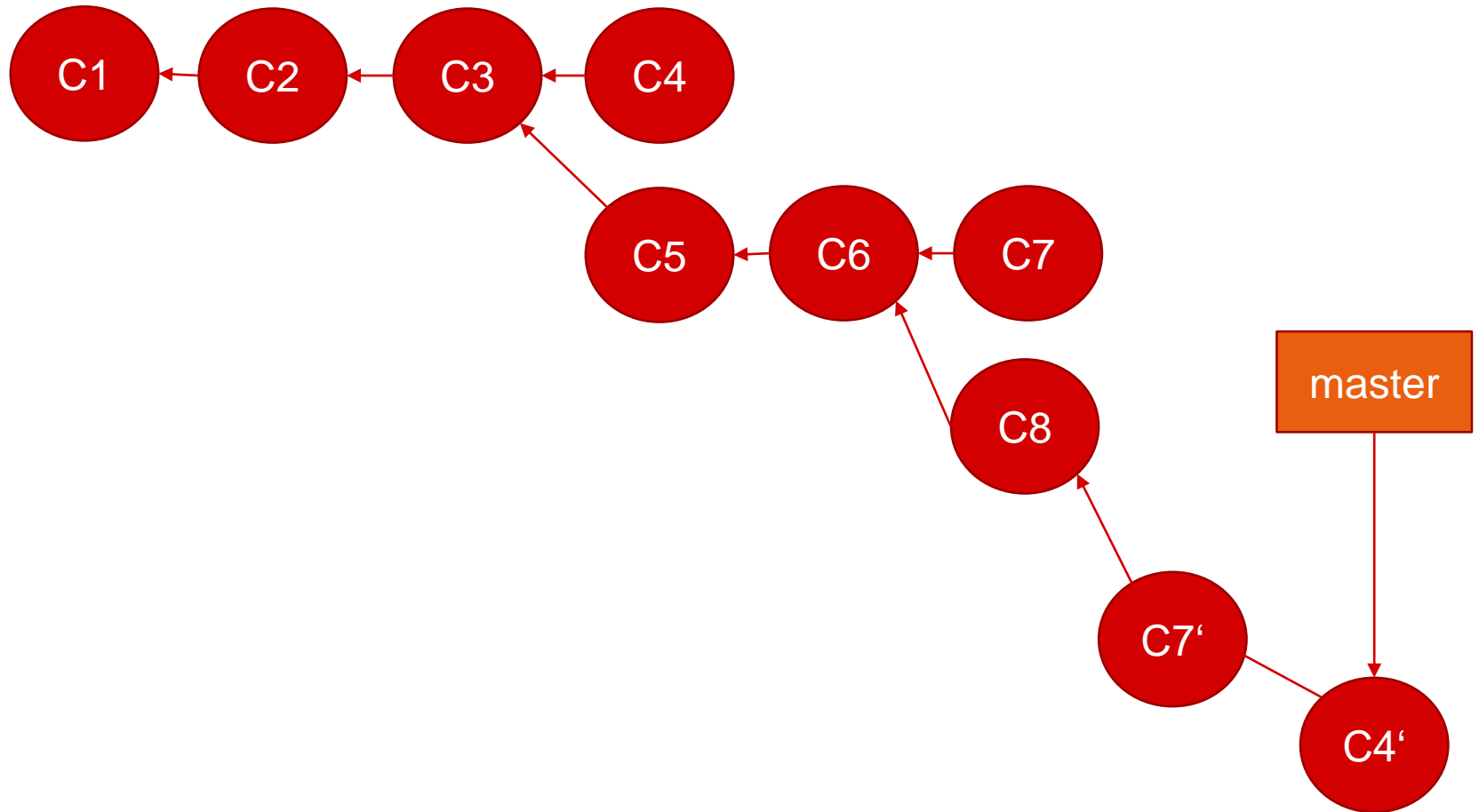
Merge feature2 mit feature1:

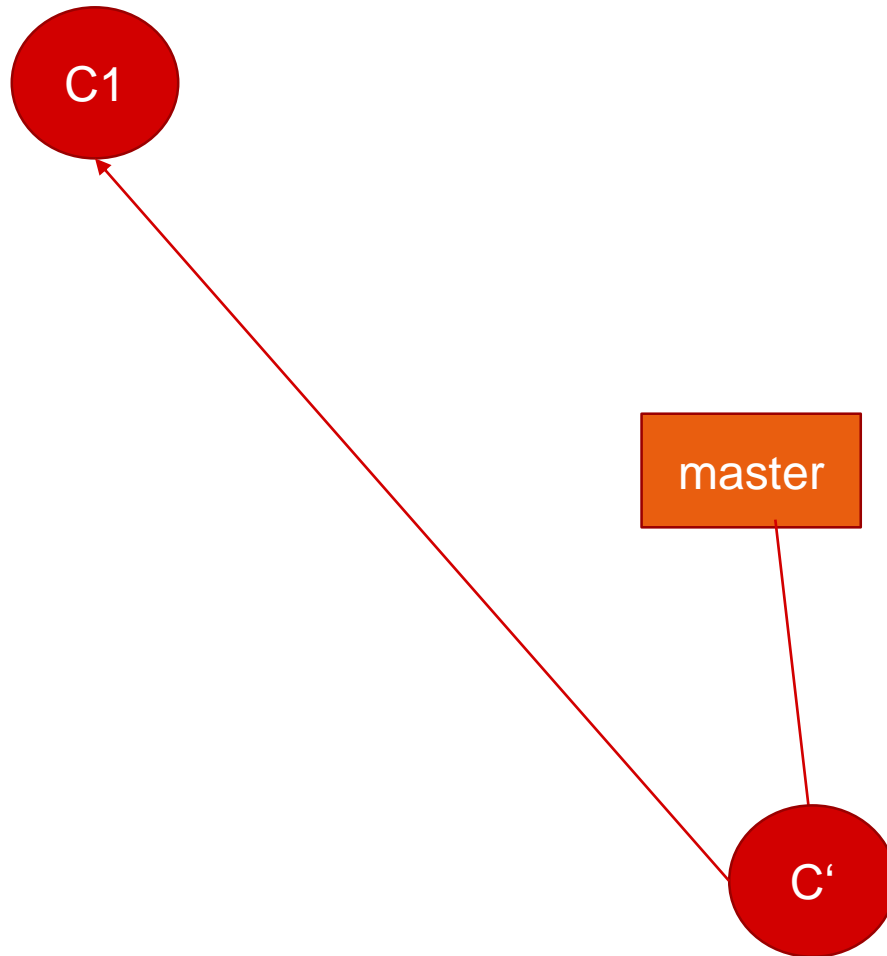


Merge master mit feature2:



Rebasing





```
git remote add file:Server s1  
git remote add file:Server2 s2
```

Repository (Developer)

master
feature1
feature2
s1/master
s2/master

Repository (Server)

master
develop
...

Repository (Server2)

master
develop
...

- http/https
- ssh
- file

- Remote Branch:
 - fetch
 - Dateitransfer aller Commits und aller Contents

- s1/master
 - Normal: Alias auf ein Commit
 - Checkout möglich
 - Aus Sicht des Developer-Repository ist der Branch read-only
 - KEIN COMMIT MÖGLICH
 - Nachträglich fetch ist immer möglich, ohne Konflikte
- Lokale Branches können mit einem Remote Branch synchronisiert werden
 - Commit-fähig

```
git remote add file:Server s1
```

Repository (Developer)

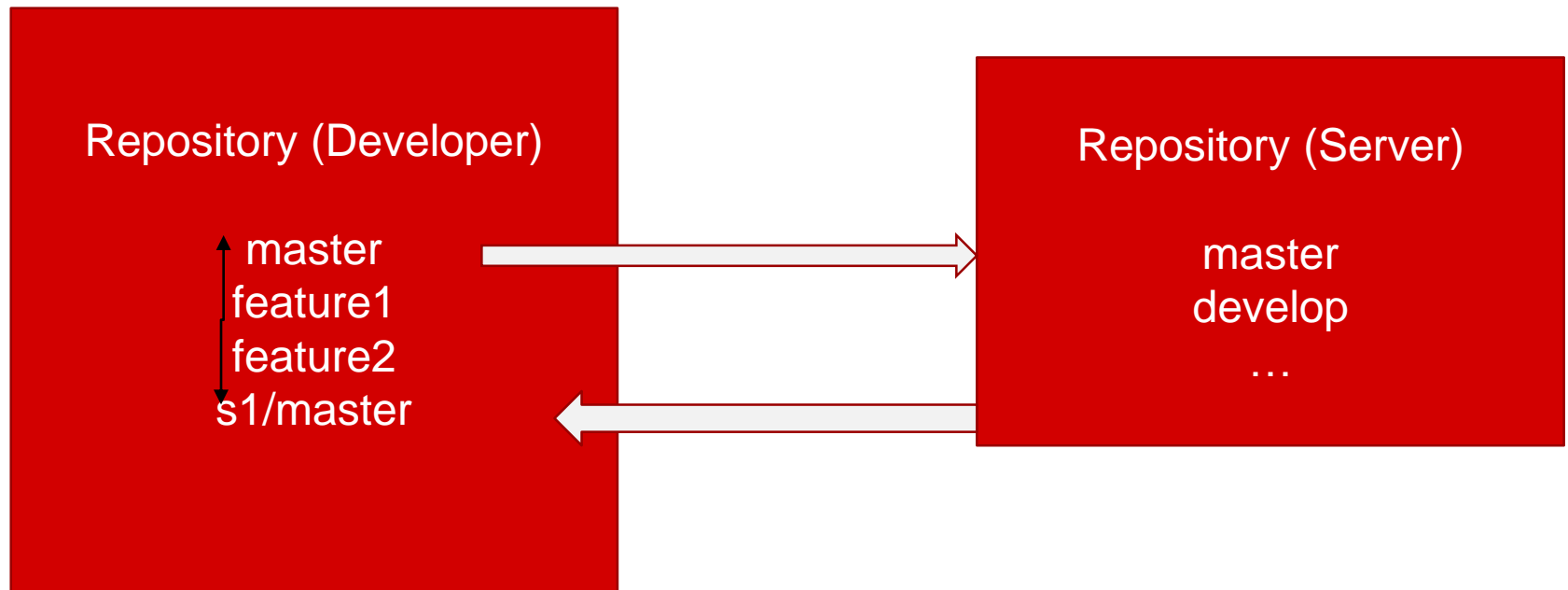
↑ master
feature1
feature2
↓ s1/master

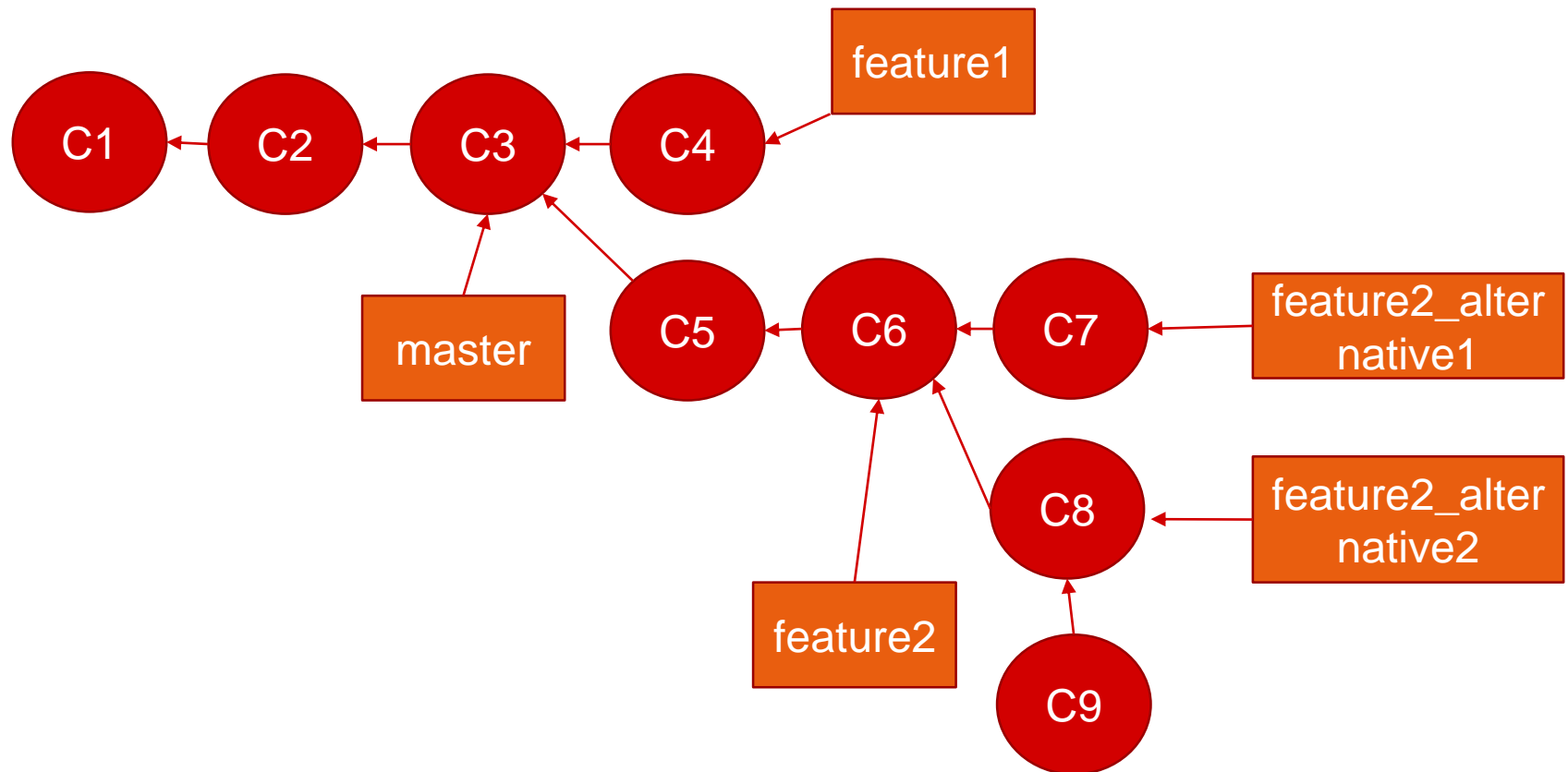
Repository (Server)

master
develop
...

- Push synchronisiert den s1/master mit dem lokalen master
 - THESE!
- In Wirklichkeit komplizierter
 - Push kann fehlschlagen!!!
- Lösung:
 - Push -> fehler
 - Fetch
 - Lokalen branch mit remote branch mergen
 - Alternativ fetch/merge: pull
 - Meistens : fetch – rebase oder pull --rebase
 - Konflikte lösen
 - Commit
 - Push
- Optimistic Locking
 - Git unterstützt kein Pessimistic Locking

- Remote Konfiguration
 - Entferntes Repository url + namespace
 - Lokaler Branch wird als „Spiegel“ des Remote Branches konfiguriert
 - Konfiguration des up- und downstreams
 - Befehle:
 - remote add
 - git clone url
 - Default Namespace: „origin“
- Befehle: fetch, pull, pull –rebase, push





- create-file-server.bat
- git clone
- Fetch, pull, push...
- Konflikte erzeugen/lösen

- Best Practice-Lösungen
- Katalog von „Design Patterns“ = Git Workflows
- Gemeinsames Element: Kurzlebige Feature Branches
- Atlassian: Git Flow
 - Release (Release-Versionen, stabil)
 - Developer (Langlebig, aktuelle Stand der Entwicklung, pontenziell inkonsistent)
 - Feature-Branches
 - Hotfixes
- GitHub Flow
 - Master , releases sind Tags auf dem Master
 - Feature-Entwicklung und HotFixes sind kurzlebige Feature-Branches

- Create-branches.bat|sh
- `git log --all --decorate --oneline --graph`
- Garbage Collection
 - `git fsck --unreachable --no-reflogs`
 - `git reflog expire --expire-unreachable=now --all`
 - `rm initialize garbage collection`
 - `git gc --prune=now`
- Ideensammlung:
 - FRAGEN STELLEN!
 - Detached HEAD
 - Neue Branches erstellen
 - Löschen feature1-> Dangling commits