MEA

Informatik Bachelor

Stefan Bente

Branching-Strategien, oder: **Wie man kollaborativ entwickelt**

SS 2024, 05.09.2024

Technology
Arts Sciences
TH Köln

Motivation

Größe von Software Systemen (Wdh.)

- 145.000 - **Apollo 11**
  - https://www.synopsys.com/blogs/software-security/apollo-11-software-development.html
- 710.000 - **Otto.de Plattform**
  - Fürstenau, D., Rothe, H., Baiyere, A., Schulte-Althoff, M., Masak, D., Schewina, K., & Anisimova, D. (2019). Growth, Complexity, and Generativity of Digital Platforms: The Case of Otto.de. *Fortieth International Conference on Information Systems*. https://aisel.aisnet.org/wi2019/track07/papers/10
- 1,3 Mio - **Zalando Logistics System (Zalos)**
  - https://engineering.zalando.com/posts/2018/04/migrating-java-8.html
- 50 Mio. - **Windows 10**
  - 3 Mio. - Kernel: 3 Mio
  - 25 Mio. - Applications (Office, Outlook, Store, …)
  - 22 Mio. – 3rd Party (Treiber, …)
  - https://softkeys.uk/blogs/blog/how-many-lines-of-code-in-windows-10
- 2 Mrd. - **Google Monorepo**
  - https://qeunit.com/blog/how-google-does-monorepo

Herausforderungen von kollaborativer Entwicklung

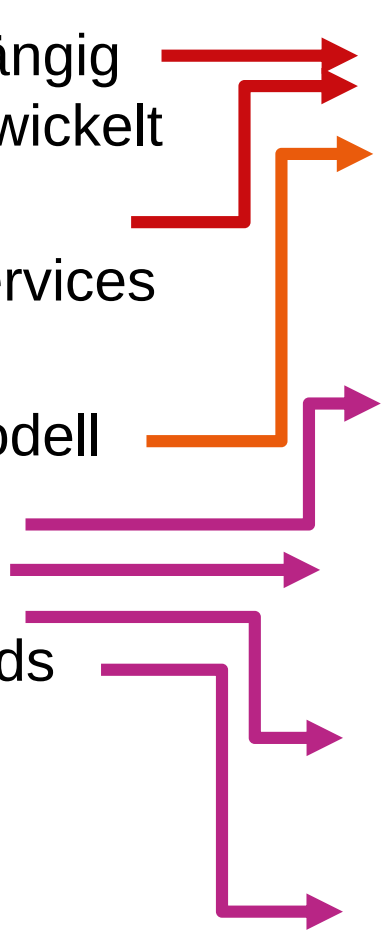
- Microservice Dungeon zählt als (kleines) „großes System“
  - 80.000+ LoC
  - 20+ Repos
  - 20+ Entwickler:innen, die parallel arbeiten (sehr lose gekoppelt)
    - ~15 in this course
    - 5+ „Praxisprojekte“ etc.
    - 4 „Admins“

Herausforderungen

a. Features unabhängig voneinander entwickelt
b. Features über verschiedene Services (Repos) verteilt
c. Open Source Modell
   - Features werden evtl. nicht fertig
   - … oder stellen sich als ungeeignet für weitere Entwicklung heraus
d. Governance: Einheitliche Qualitätsstandards

Was hilft wofür?

a. Features unabhängig voneinander entwickelt
b. Features über verschiedene Services (Repos) verteilt
c. Open Source Modell
d. Governance: Einheitliche Qualitätsstandards



1. Branching-Strategien
2. Merge Request Governance
3. Zentrale Dokumentation und Wissensmanagement
4. API-Dokumentation
5. Architectural Principles als "Leitplanken" für Entwicklungsteams
6. Architectural Decision Records (ADR)

Wie ist das mit den anderen Elementen?

2. Merge Requests Governance
   - https://the-microservice-dungeon.gitlab.io/docs/docs/contribution-guidelines/
   - „Currently Worked On“ Pages: https://the-microservice-dungeon.gitlab.io/docs/docs/roadmap/currently-worked-on/
   - 3er Team als Governance Board
     - Sichtung der „Currently Worked On“ Features mit Entscheidung, ob in den Main Branch gemerged wird
3. Zentrale Dokumentation und Wissensmanagement
   - https://the-microservice-dungeon.gitlab.io/docs/
4. API-Dokumentation
   - https://the-microservice-dungeon.gitlab.io/docs/docs/reference/
5. Architectural Principles als "Leitplanken" für Entwicklungsteams
   - haben wir nicht im MSD
   - Oft gewählter Ansatz in großen agilen Organisationen
6. Architectural Decision Records (ADR)
   - https://the-microservice-dungeon.github.io/decisionlog/
     - (allerdings nicht gepflegt)

1. Branching-Strategien

Quelle: Martin Fowler, Patterns for Managing Source Code Branches
https://martinfowler.com/articles/branching-patterns.html

Lese-Empfehlung!

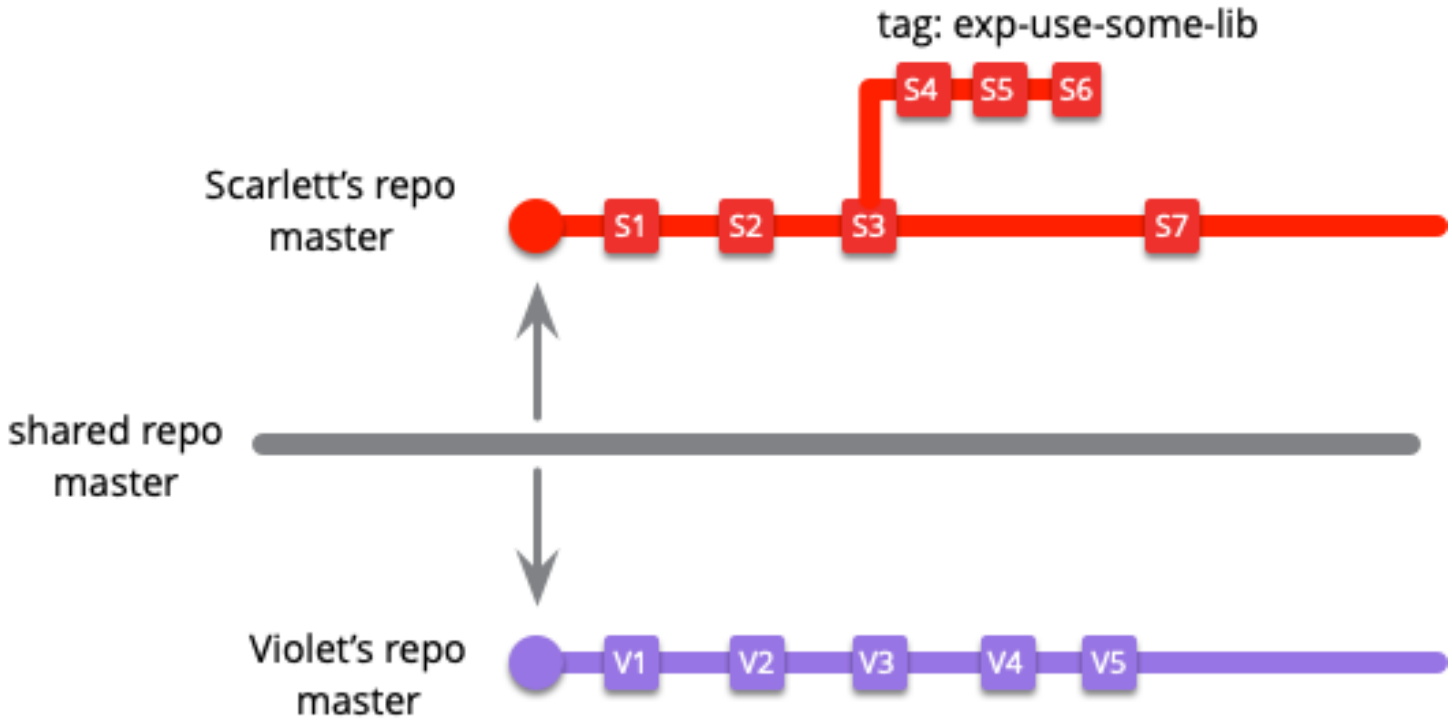
Alle nachfolgenden Abbildung aus dieser Quelle.

Die 4 Optionen nach Fowler

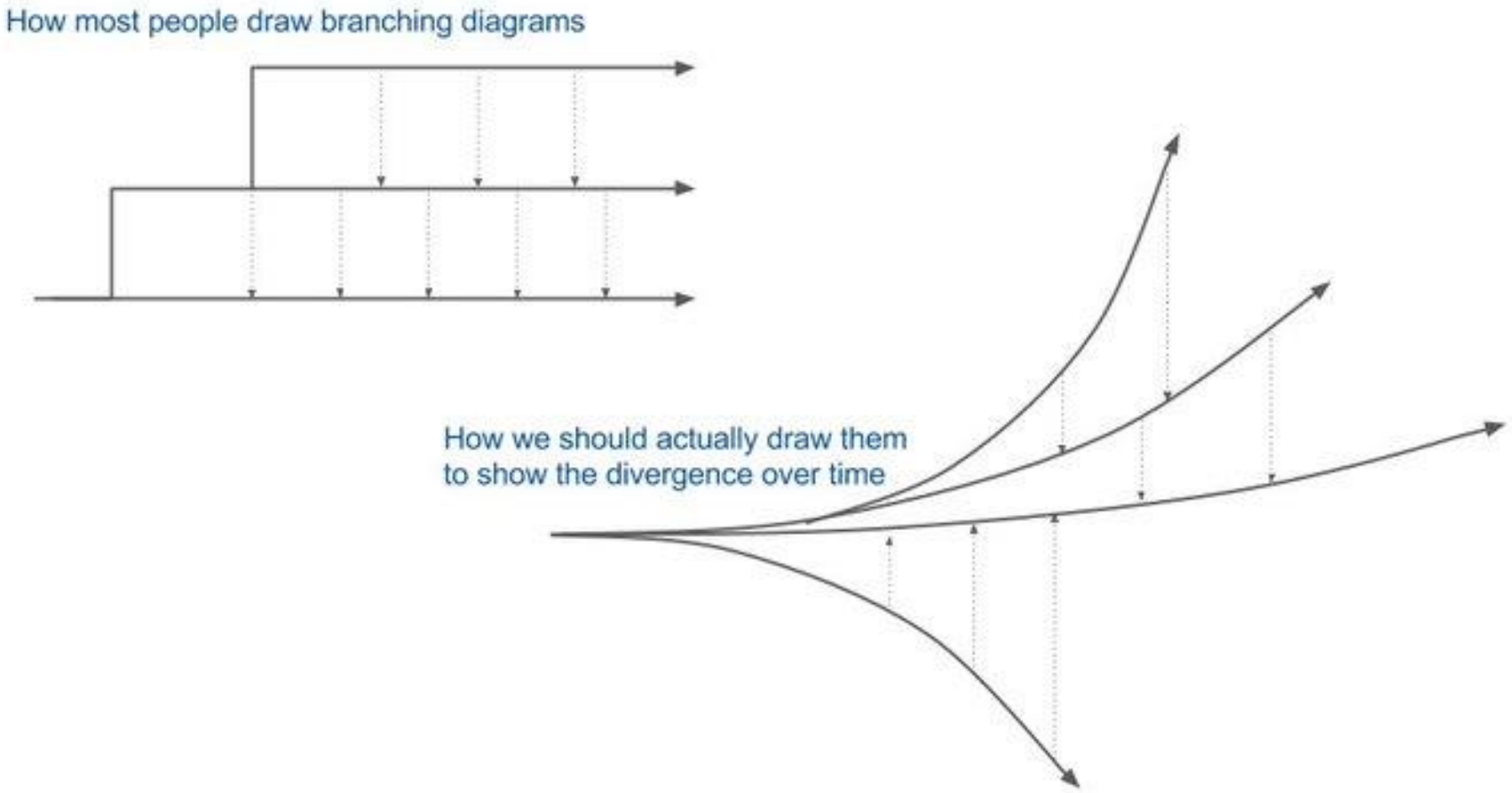
- Haupt-Optionen
  1. Ungeregeltes Arbeiten
  2. Arbeiten mit einem Master-Branch
  3. Feature Branches
  4. Continous Integration

- Low vs. High Frequency

1) Ungeregeltes Arbeiten

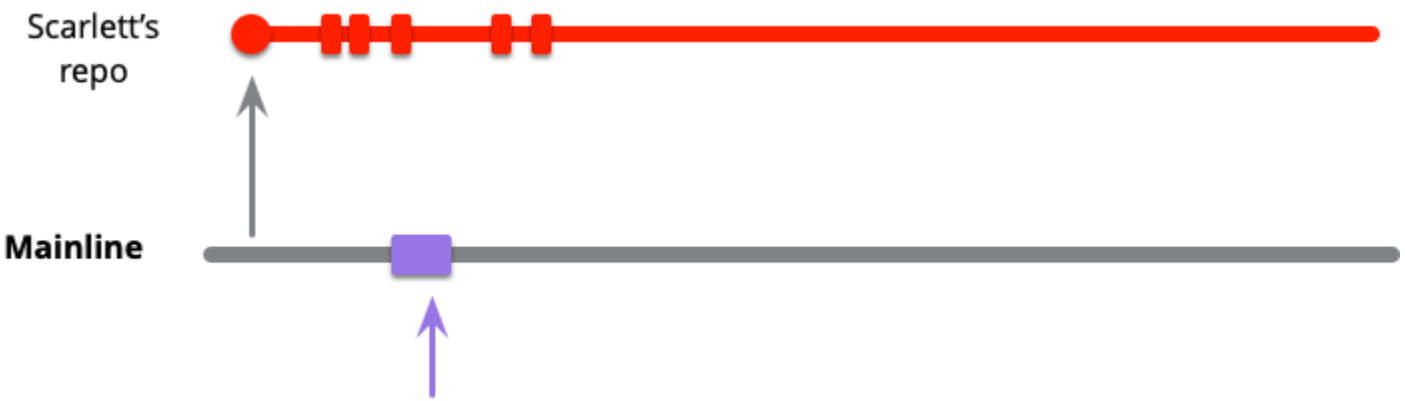


… und der Effekt

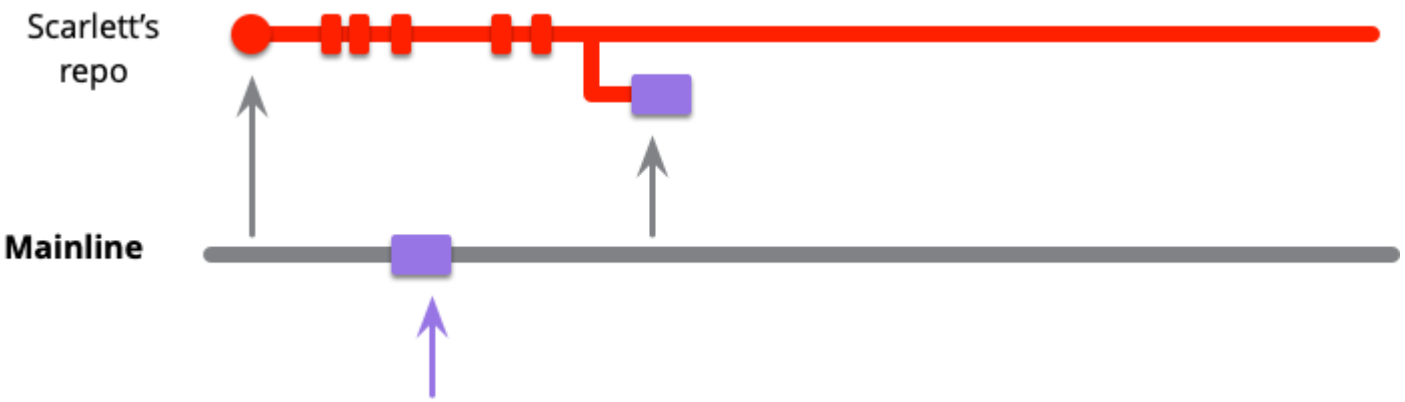


2) Arbeiten mit einem Master-Branch

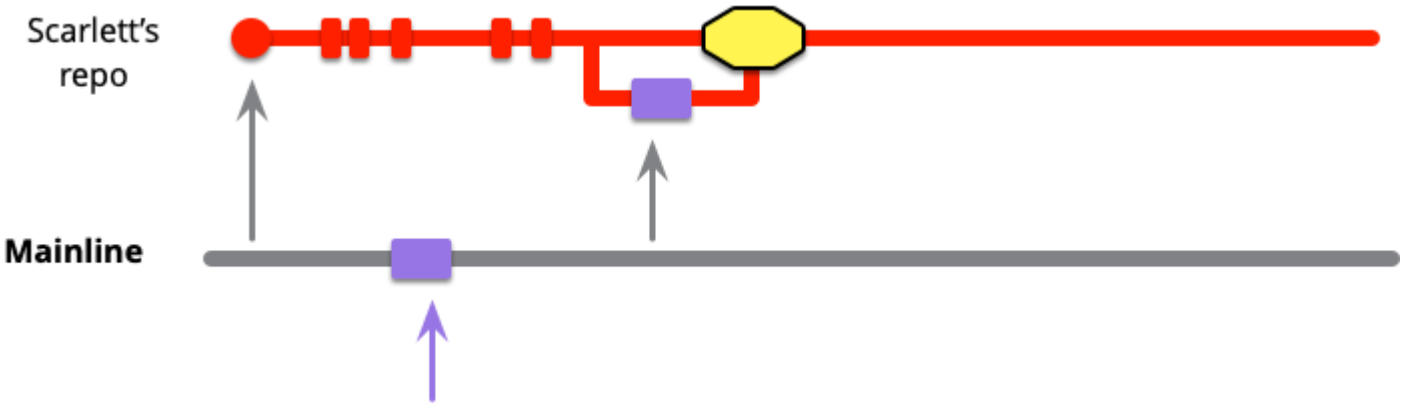
2) Arbeiten mit einem Master-Branch



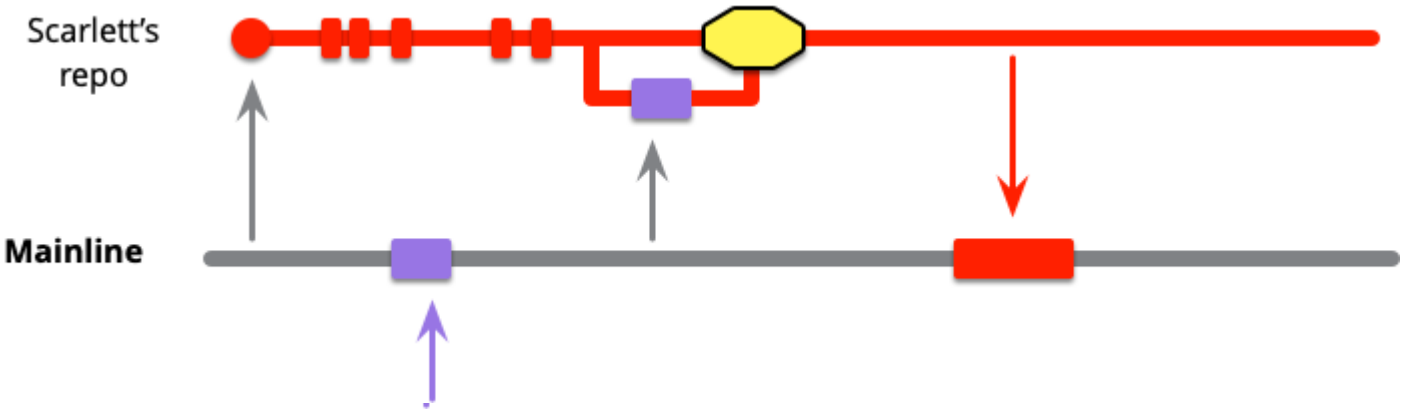
2) Arbeiten mit einem Master-Branch



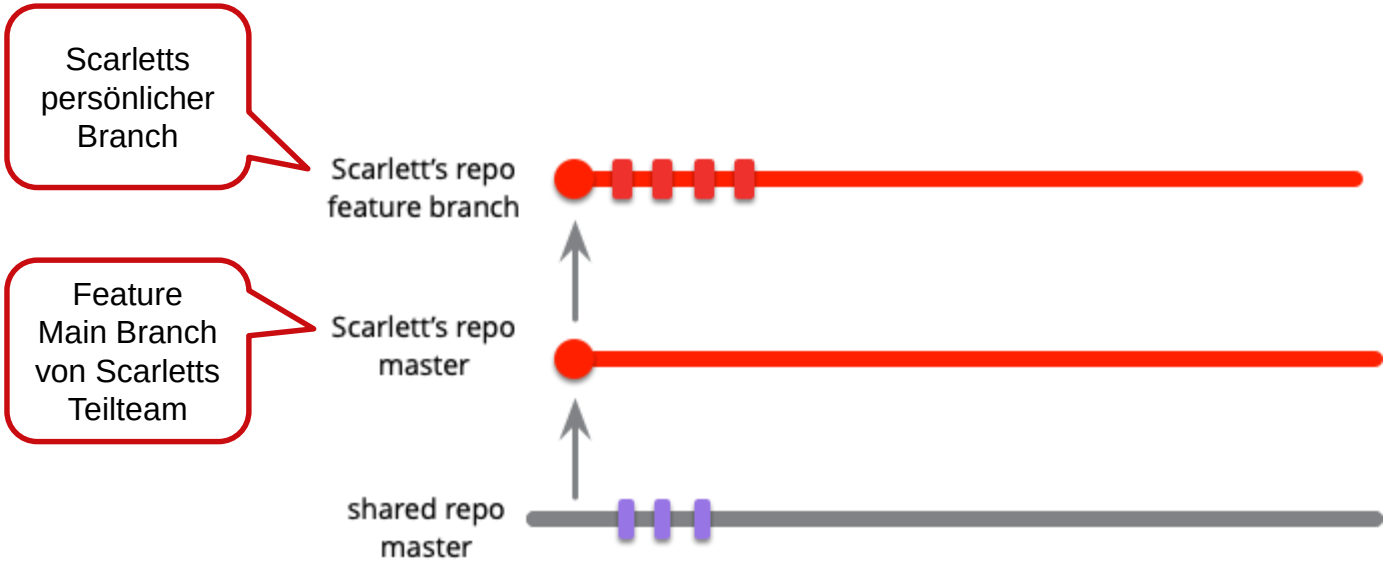
2) Arbeiten mit einem Master-Branch



2) Arbeiten mit einem Master-Branch

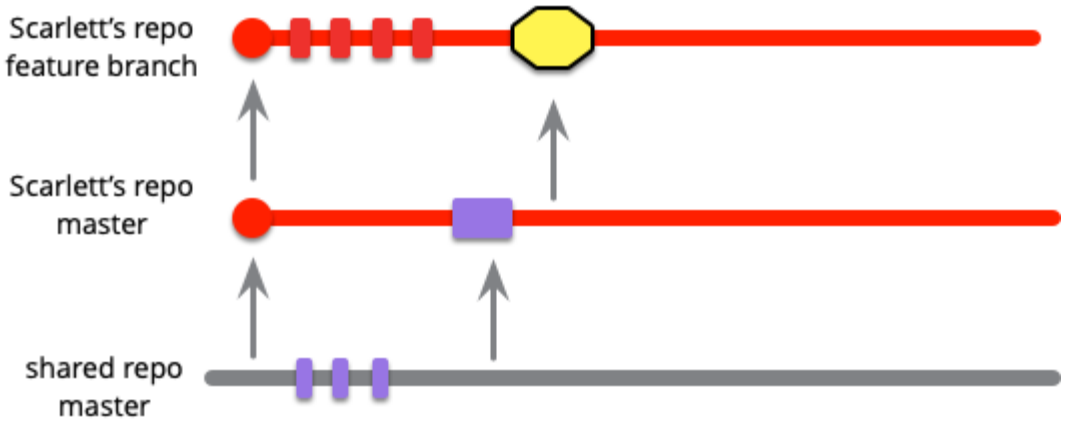


3) Feature Branches

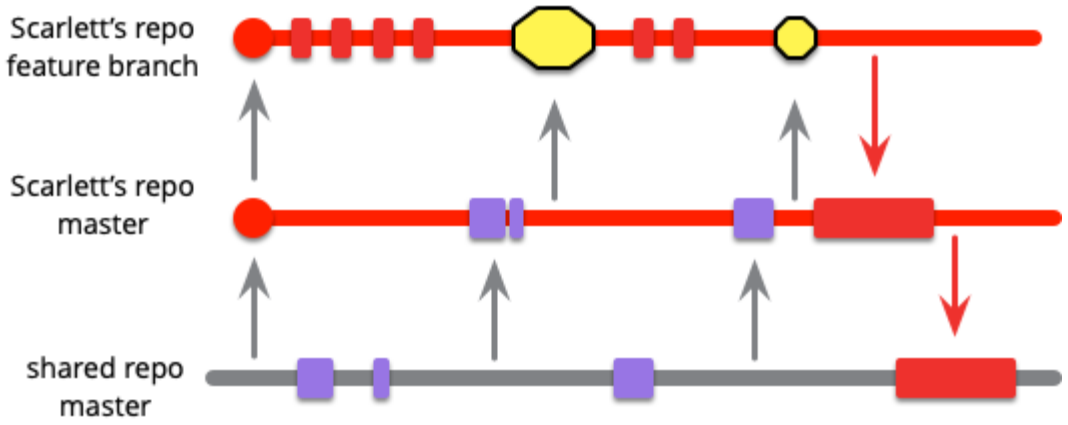


Annahme: Scarlett arbeitet mit einem Teilteam, sagen wir noch zwei Kollegen, an einem bestimmten Feature.

3) Feature Branches

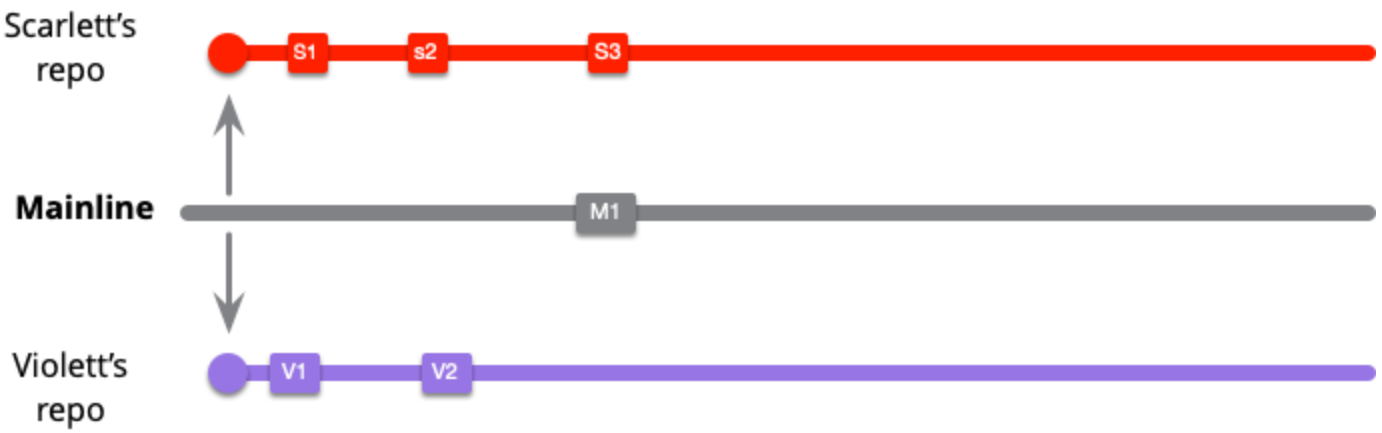


3) Feature Branches

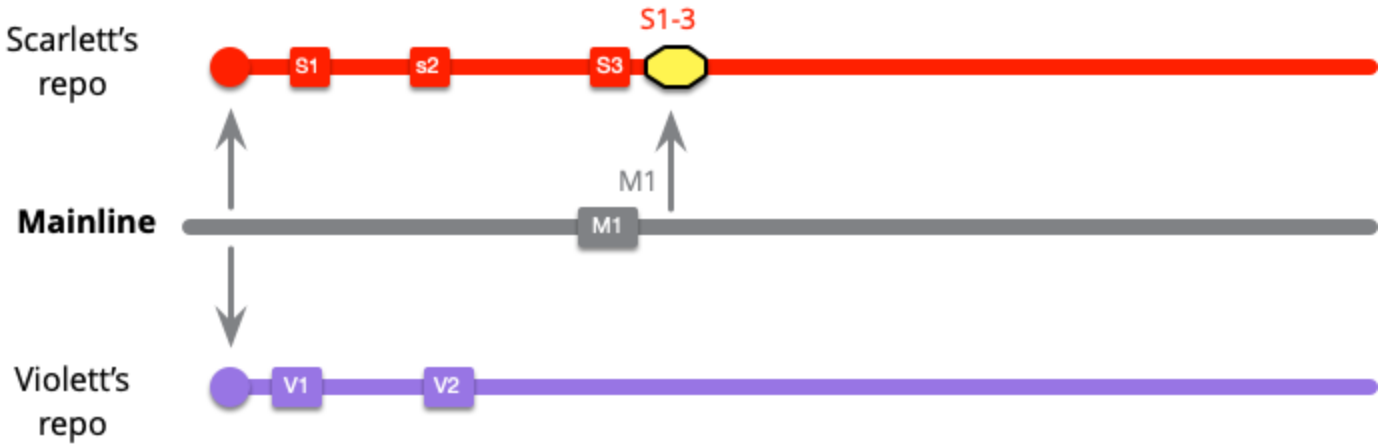


Low Integration Frequency

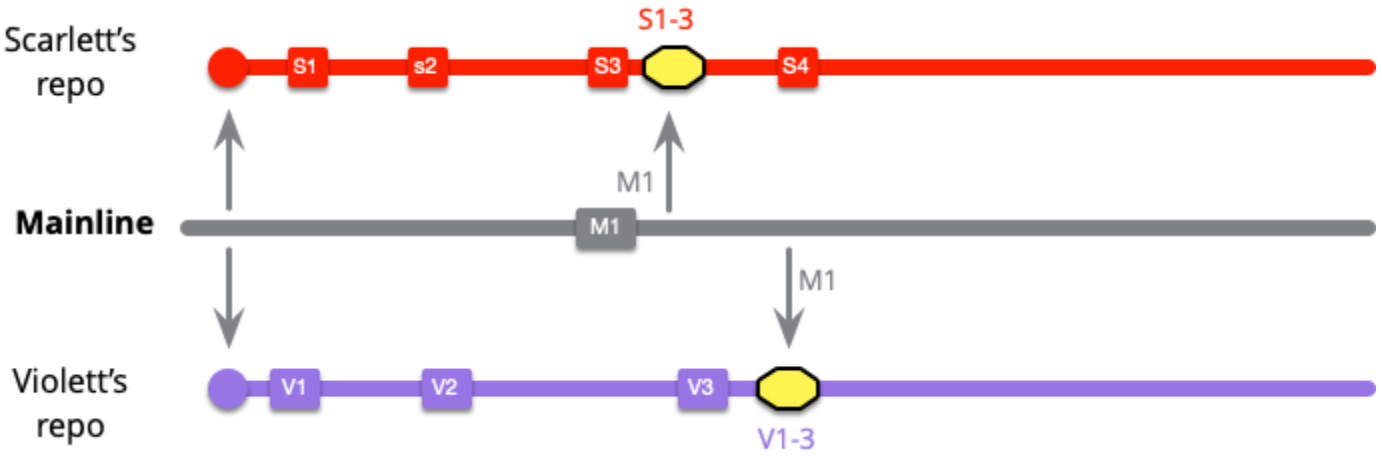
Low Integration Frequency



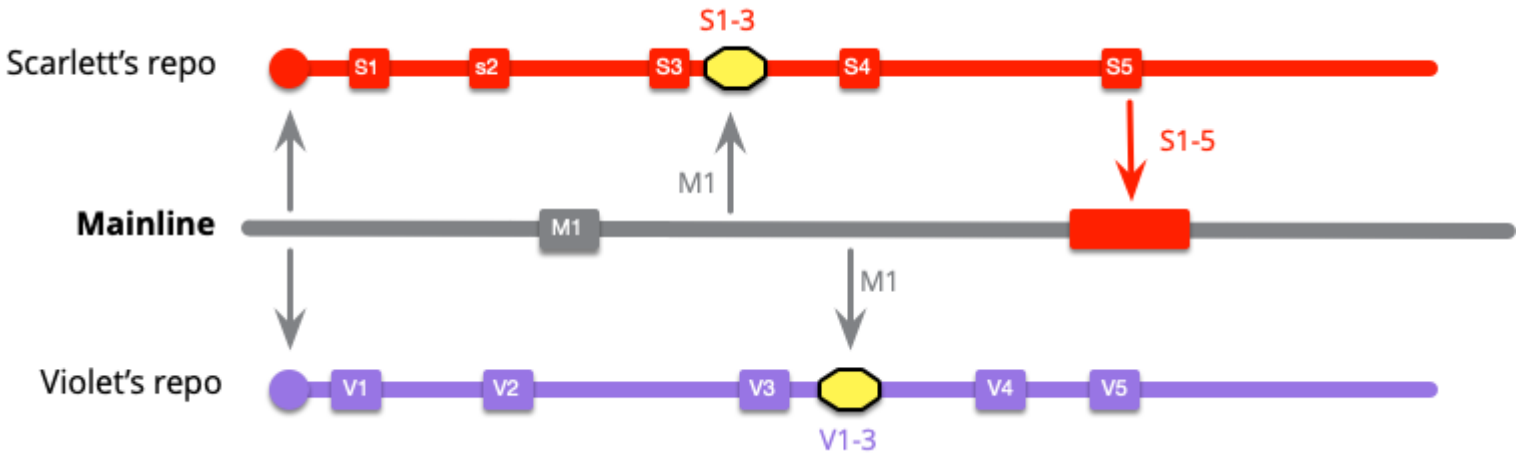
Low Integration Frequency



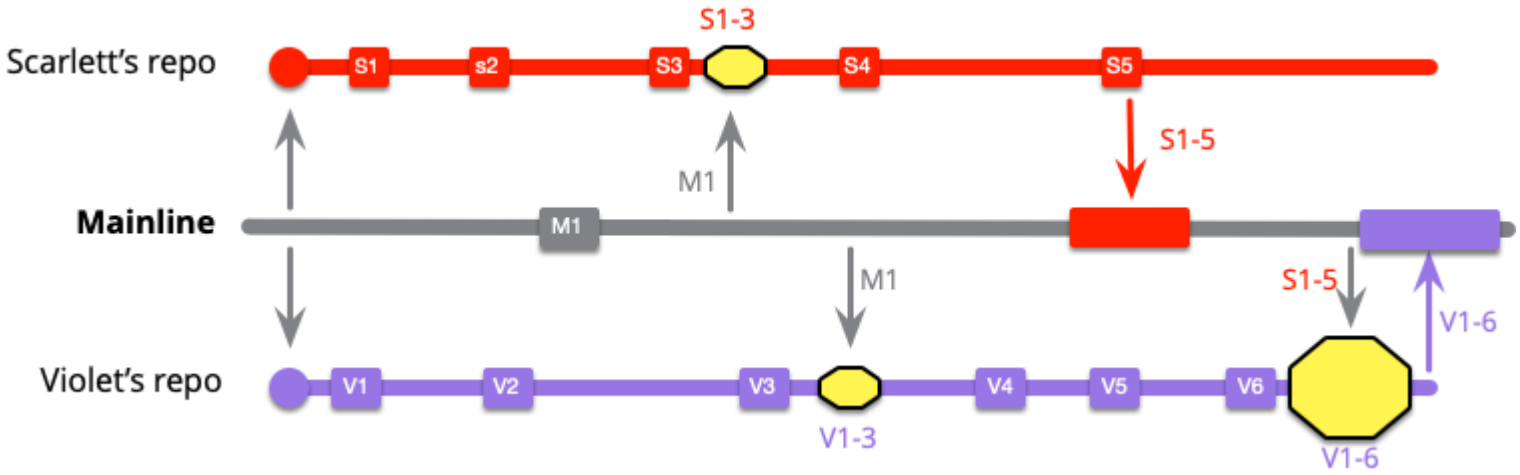
Low Integration Frequency



Low Integration Frequency



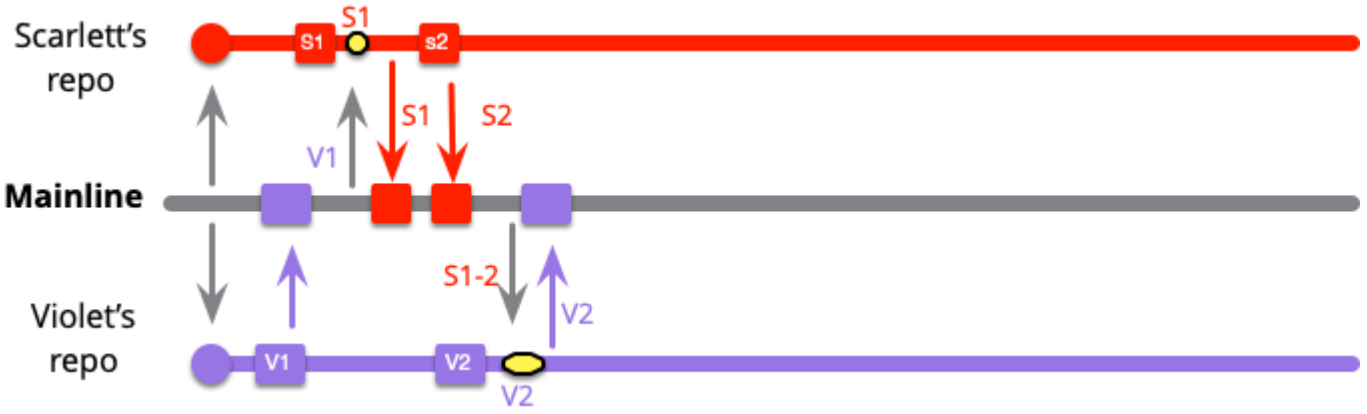
Low Integration Frequency



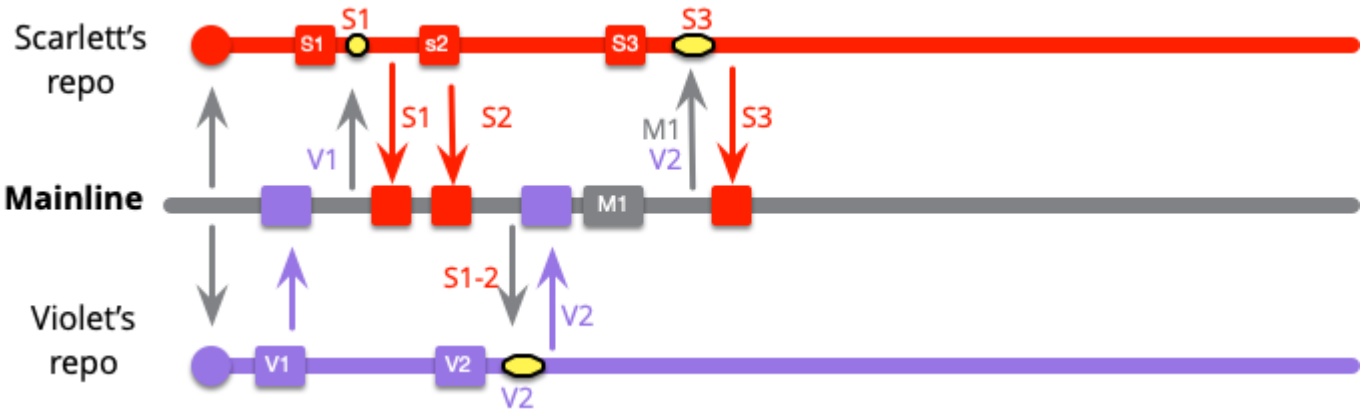
High Integration Frequency

High Integration Frequency

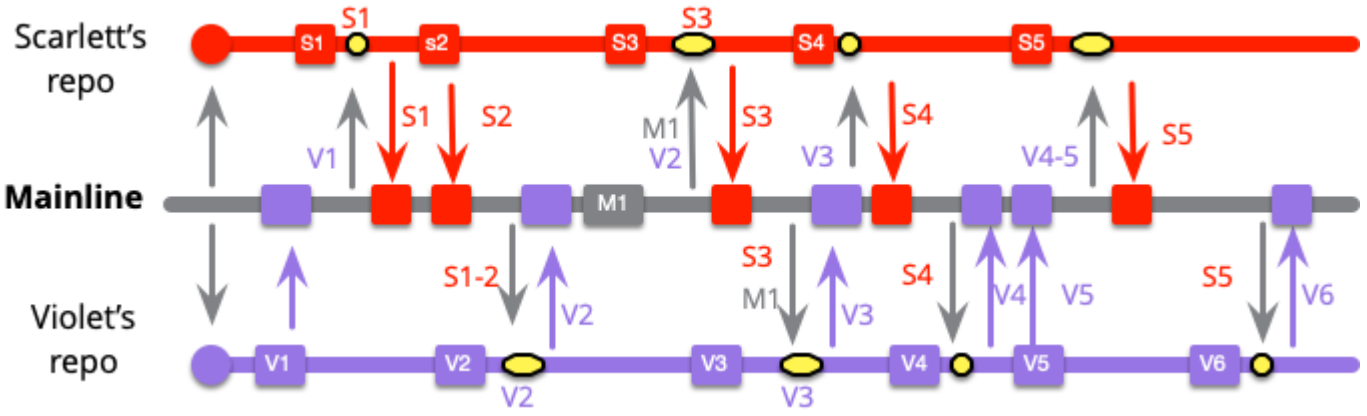
High Integration Frequency



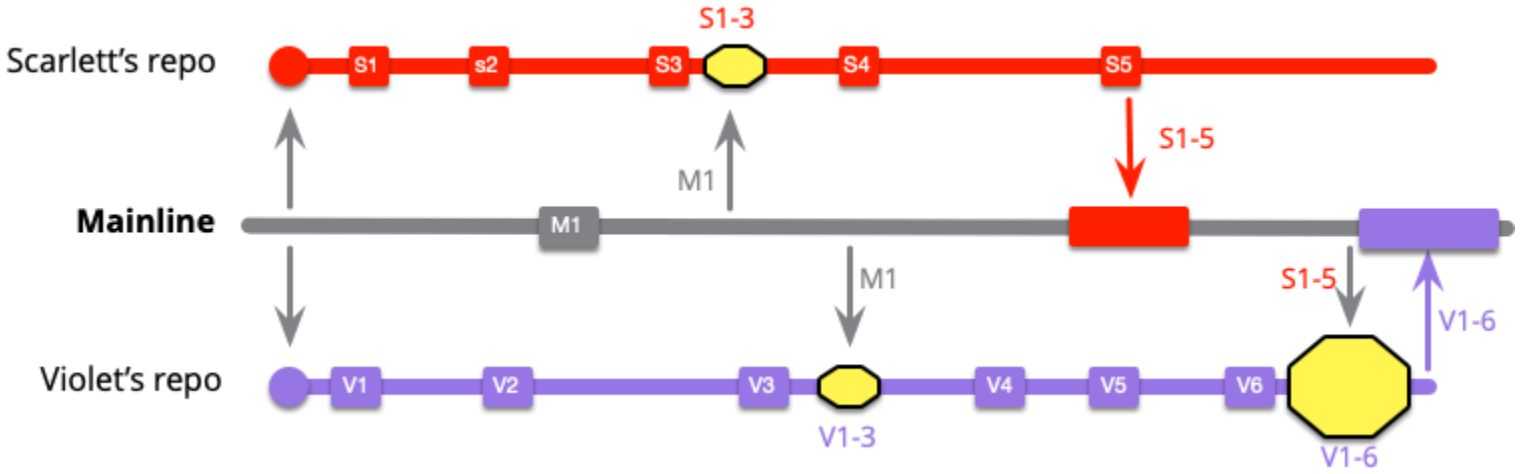
High Integration Frequency

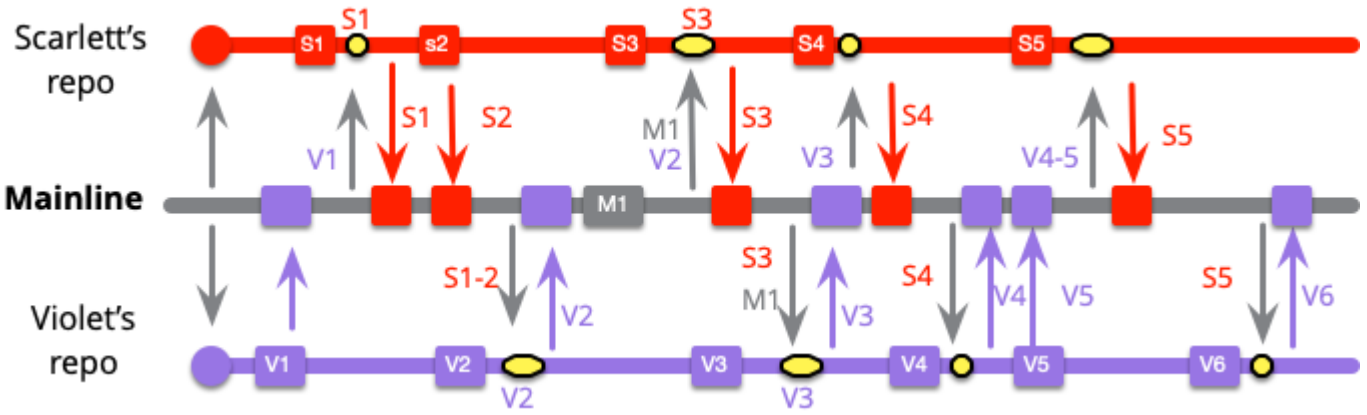


High Integration Frequency

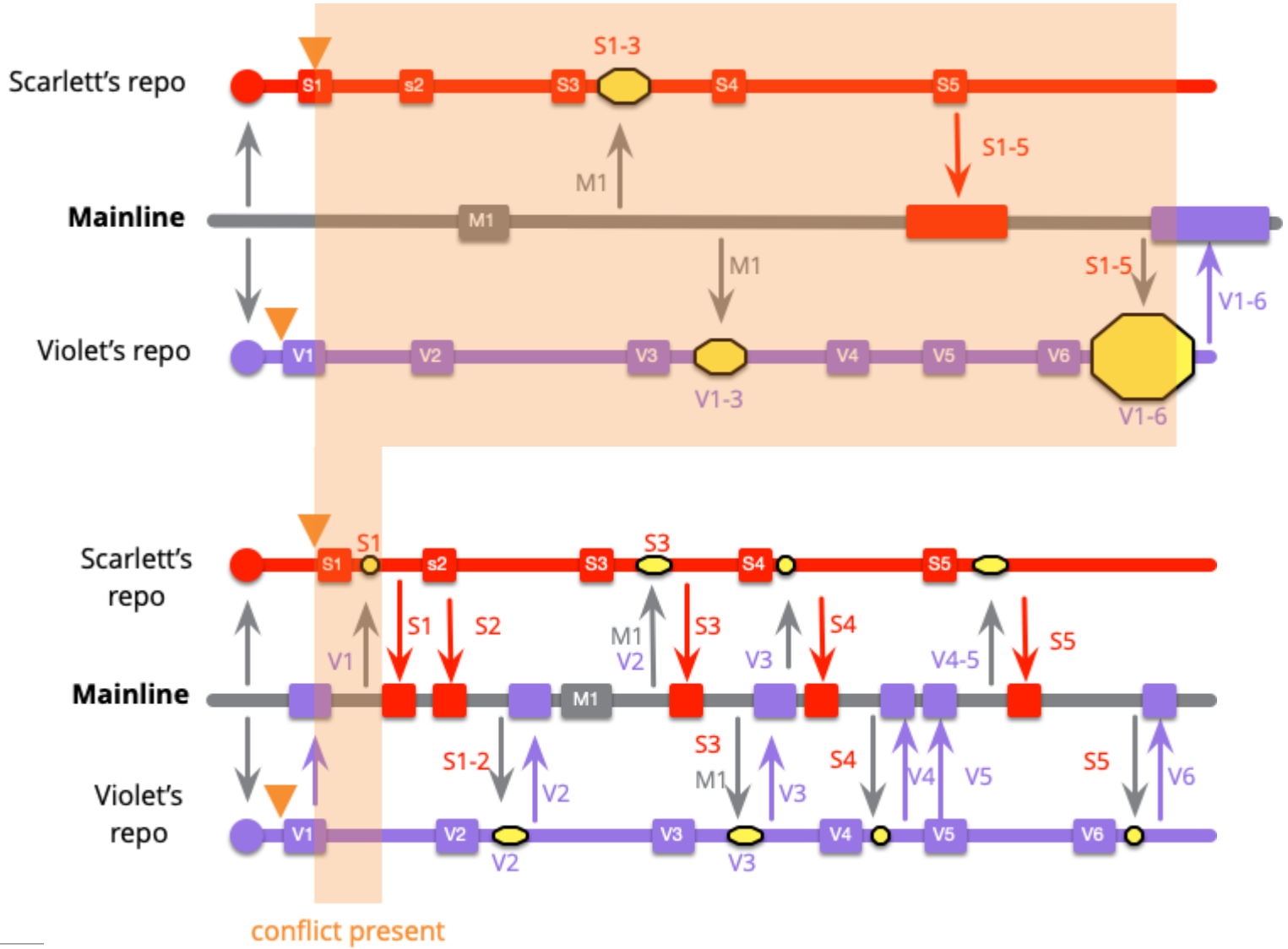


Low vs. High Integration Frequency





Konflikte bei Low vs. High Integration Frequency



Continuous Integration

Continuous Integration =

- „Heathy Branches“
  - gründlich getestet
  - werden mit Sicherheit in den Main Branch übernommen

  +

- High Frequency Integration

Feature Branches vs. Continuous Integration

Feature Branches

- Funktioniert auch für lose zusammenarbeitende Teil-Teams
- Erlaubt Quality-Gate vor Übernahme nach Main
- Insgesamt weniger, aber schlecht planbare Merges
  - Haben Potential zu scheitern!
- Seltenere Integration => insgesamt weniger produktiv wegen höherer Reibungs-Verluste

Continuous Integration

- Häufigere Merges, aber geringerer Merge-Aufwand
- Höhere Produktivität
- Macht Refactoring einfacher (niederschwelliger)
- Funktioniert nur bei einem disziplinierten, eingespielten und eng zusammen-arbeitenden Team
  - Healthy Branches mit guter Test-Abdeckung
  - Sicherheit, dass der Branch weiterlebt