


- Motivation
 - Konfig Management – Überblick/Definition
 - Aufgaben des Kernprozesses
 - Auswahl der Konfig Elemente
 - Erstellung des KM Handbuchs
 - Beschreibung der Konfig Elemente
 - Festlegung der Projektstruktur
 - Verwaltung der Konfig Elemente
 - Projektautomatisierung
 - Änderungs- und Fehlermanagement
 - Workflow- und Raumkonzept
-  Arbeiten mit speziellen Tools



Versionsverwaltung

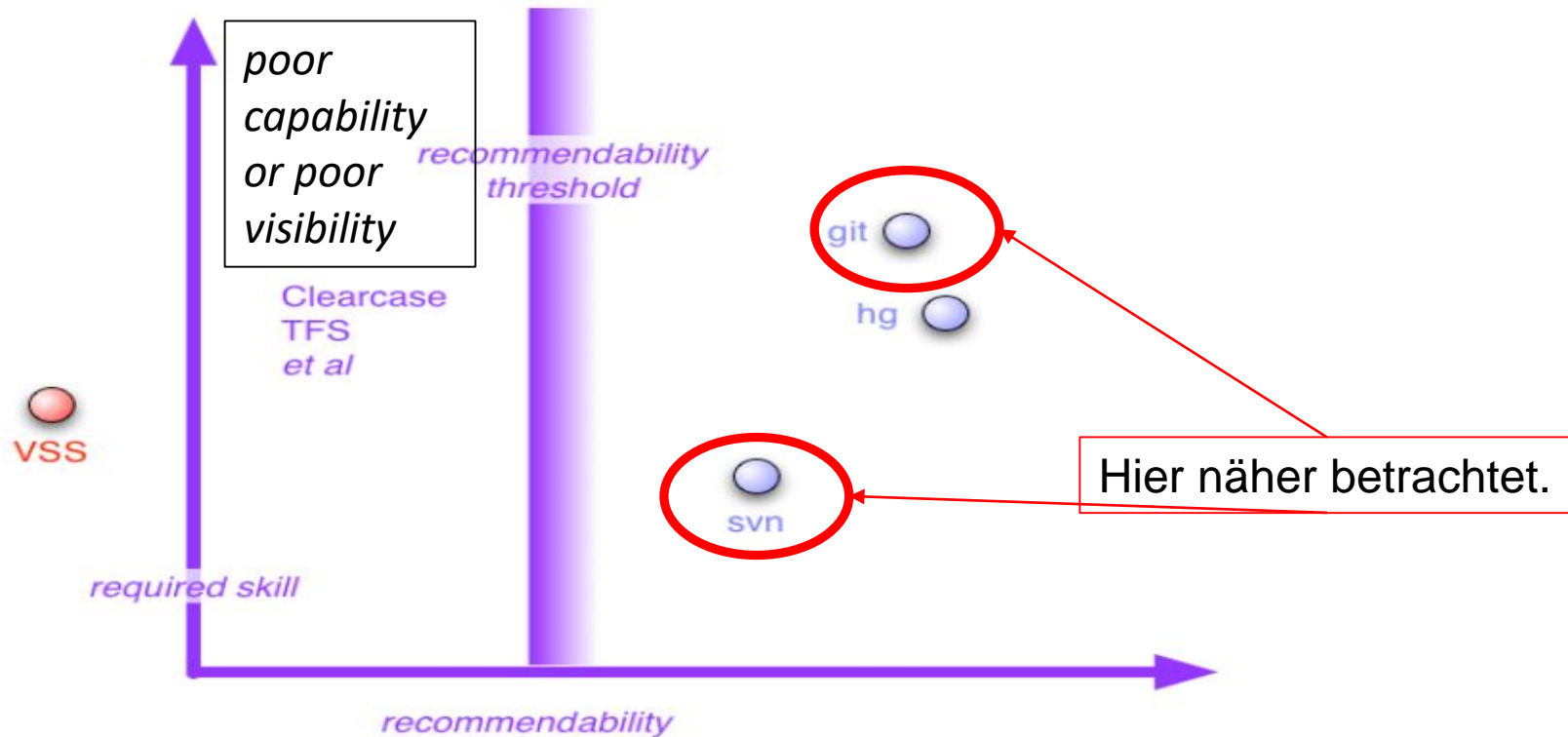
Quelle: <https://de.wikipedia.org/wiki/Versionsverwaltung>

	Open-Source-Systeme	Proprietäre Systeme
Zentrale Systeme	<ul style="list-style-type: none"> • MediaWiki • SCCS • RCS • CVS • Subversion (SVN) 	<ul style="list-style-type: none"> • Alienbrain • Perforce • Team Foundation Server • Visual SourceSafe • ClearCase • IBM Rational Synergy • PTC Integrity • SAP Design Time Repository (DTR) • versiondog (für SPS, Roboter, Automatisierungsgeräte) • Sourcegear Vault (ehem. Fortress)
Verteilte Systeme	<ul style="list-style-type: none"> • Bazaar • BitKeeper • Darcs • Fossil • Git • GNU arch • Mercurial • Monotone 	<ul style="list-style-type: none"> • Rational Team Concert



Martin Fowler zu VCS (2010)

<http://martinfowler.com/bliki/VersionControlTools.html>



Neuere Vergleiche unter

<https://www.dev-insider.de/5-freie-versionskontrollsysteme-im-ueberblick-a-835250/>

https://en.wikipedia.org/wiki/Comparison_of_version-control_software

Es gibt verschiedenste Tools zur Unterstützung der Konfigurationsprozesse.

Hier werden einige Open Source Tools vorgestellt, die weit verbreitet sind:

- **Subversion, GIT** - Versionskontrolle
- **Maven** – Build Prozess
- **Jenkins** – Continuous Integration
- **Redmine** – Kollaboration



- Subversion: Top Level Projekt der Apache Software Foundation
(<https://subversion.apache.org/>)
- Logischer Nachfolger von CVS
- Aktuelle (Januar 2020) Version : 1.13.0
- Literatur:
 - <http://svnbook.red-bean.com/>
 - G. Popp: Konfigurationsmanagement, d.punkt Verlag, 2013



Vision von Subversion

Subversion exists to be universally recognized and adopted as an **open-source, centralized version control system** characterized by its **reliability** as a safe haven for valuable data; the **simplicity** of its model and usage; and its ability to support the needs of a **wide variety of users and projects**, from individuals to large-scale enterprise operations.

Quelle: <https://subversion.apache.org/>



Vorteile von subversion

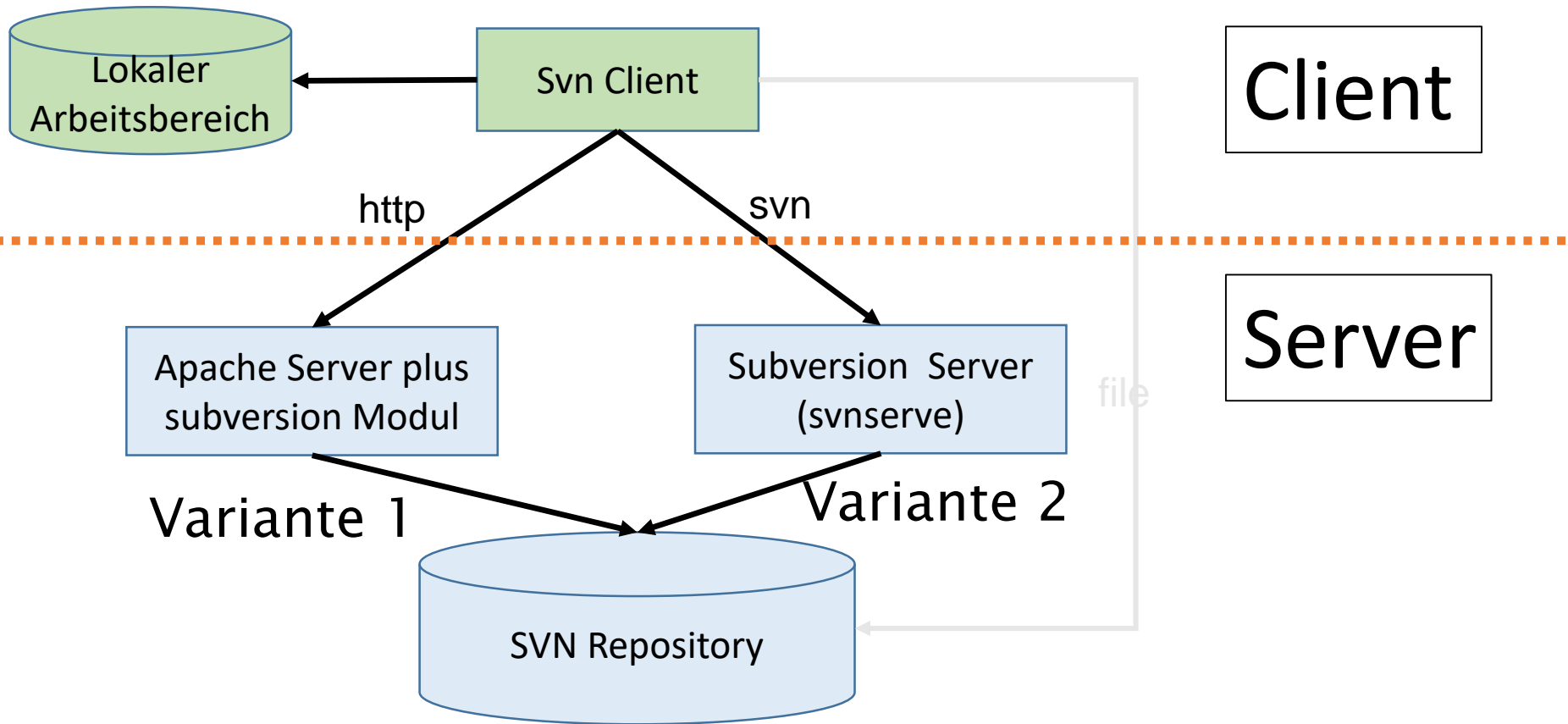
- Kostenfrei erhältlich
- Gut erprobt
- Stetige Weiterentwicklung als Apache Projekt
- Weithin akzeptiert
- Unterstützt von vielen Tools
- Einfach
- Komplexe Szenarien abbildbar
- Gut für CI geeignet



- Versionierung von Dateien **und Verzeichnissen**
- Versionierung von Kopier-, Lösch- und Umbenennungsvorgängen
- Atomare Check-Ins
- Unterstützung für Metadaten (Properties): Dateien können mit Schlüssel-Wert Paaren versehen werden.
- Flexible Client Server Architektur
- Effizienter Algorithmus zur Delta Bildung
- Branching und Tagging mit Cheap Copies



Subversion - Architektur



Clients in verschiedenen Ausprägungen

Siehe z.B.

http://en.wikipedia.org/wiki/Comparison_of_Subversion_clients

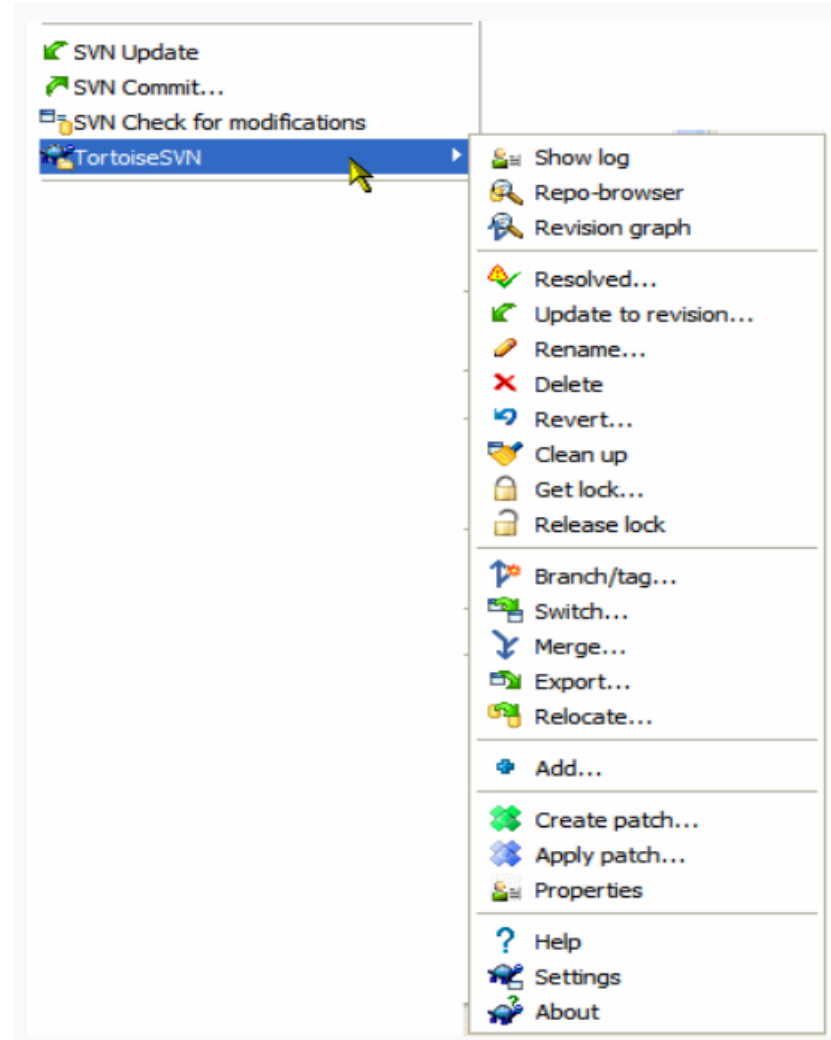
Häufig verwendet:

- Command line clients
- Tortoise: <http://tortoisesvn.net/>
- Eclipse subversive:
<http://www.eclipse.org/subversive/>



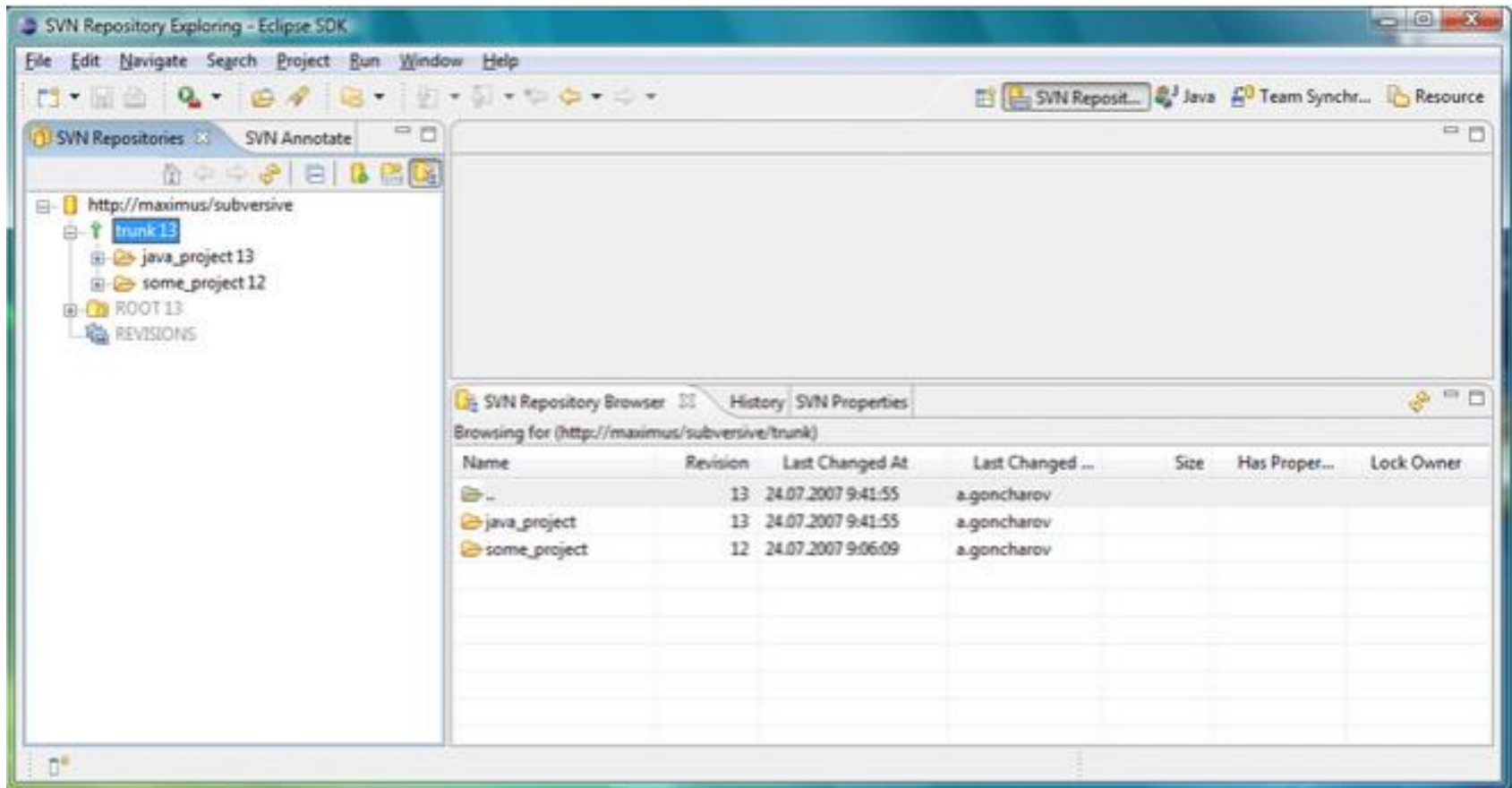
Subversion Clients - Beispiele

Tortoise:
Arbeiten mit dem
Kontextmenü im
Explorer



SVN Clients - Beispiel

Subversive – Integration in Eclipse



http://www.eclipse.org/subversive/documentation/teamSupport/repos_persp.php

Unter

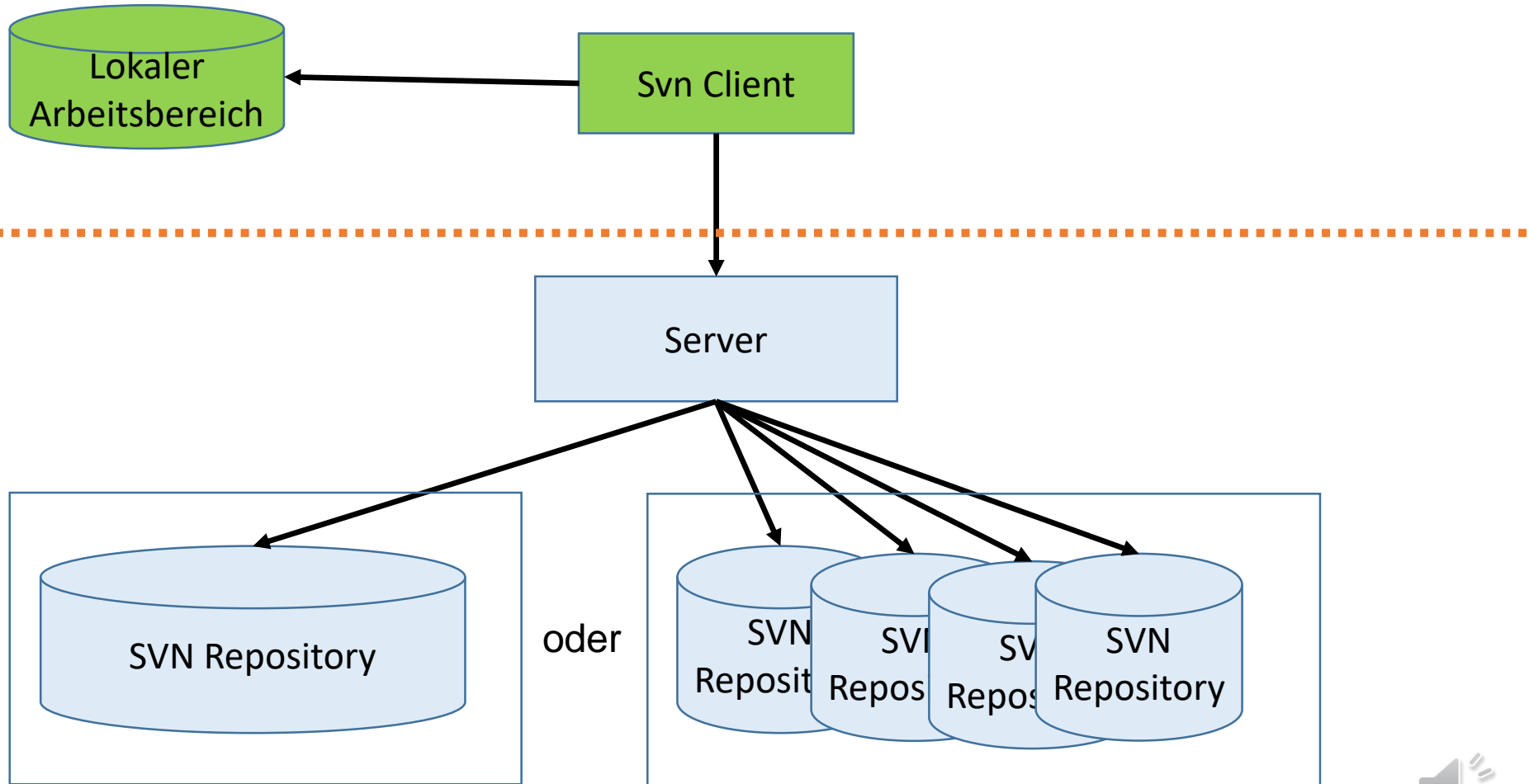
<http://subversion.apache.org/packages.html> :

- Installationspakete für Client und/oder Server für verschiedene Plattformen
- und Anleitungen.

Client als Command Line Client oder mit GUI.
Die folgenden Folien verwenden einen
Command Line Client.



Einrichten des Repositories



Einrichten des Repositories

Repository wird **nicht** mit der svn Installation angelegt.

Erste Entscheidung: zentrales oder projektspezifisches Repository möglich



Ein Repository
für alle Projekte
im Unternehmen

oder

Ein Repository
pro Projekt

Normalerweise ein Repository pro Projekt leichter organisatorisch in den Griff zu kriegen.



Einrichten des Repositories

Erstellung eines Repositories (ab jetzt für die Beispiele ein Repository pro Projekt)

→ Verwendung eines eigenen Admin Tools (svnadmin)

```
svnadmin create D:\svn-repos\meinProjekt
```

Default: Storage System ist das Filesystem.
Alternative: Berkeley DB



Berkeley DB vs Filesystem

Aus *svn-book*, Tabellenausschnitt

Tabelle 5.1. Vergleich der Projektarchiv-Datenspeicherung

Kategorie	Funktion	Berkeley DB	FSFS
Zuverlässigkeit	Unversehrtheit der Daten	Höchst zuverlässig, wenn es richtig aufgesetzt wird; Berkeley DB 4.4 bringt automatische Wiederherstellung	Ältere Versionen hatten einige selten aufgetretene, jedoch Daten zerstörende Fehler
	Empfindlichkeit gegenüber Unterbrechungen	Sehr; Abstürze und Berechtigungsprobleme können die Datenbank „verklemmt“ hinterlassen, was eine Wiederherstellung mithilfe des Journals erfordert	Ziemlich unempfindlich



- **Revisionen:** Subversion vergibt für jede schreibende Transaktion eine Revisionsnummer (automatisch und nicht änderbar).
- Diese Revisionsnummer bezieht sich nicht nur auf die Änderungen sondern auf alle Elemente des Repositorys (**globale Versionierung**)
- Ein **Changeset:** Alle im Rahmen der Transaktion getätigten Änderungen. D.h. ein Changeset kann einer Revision zugeordnet werden und umgekehrt.



- **Standard:** Einstellung der Benutzerauthentifizierung in der Datei `conf/svnserve.conf`. **Pwd im Klartext!**
- **Alternativen:**
 - Einsatz des Apache Servermoduls → Verwendung der Infrastruktur des Apache Servers zur Authentifizierung.
 - CYRUS SASL Authentifizierung (<http://asg.web.cmu.edu/sasl/>) (dadurch z.B. Anbindung an LDAP Server möglich)
→ siehe dazu die svn Dokumentation.



Benutzer und Zugriffsrechte festlegen

svnserve.conf

```
# Konfiguration des repositorys
[general]
password-db = users.conf
authz-db = access.conf
```

users.conf

```
[users]
root = rootpwd
bulenda = bulendapwd
adent = adentpwd
```

access.conf

```
[groups]
admins = root
developers = bulenda, adent

[/]
* = r
@admin = rw
@developers = rw
```

Benutzer

Zugriffsrechte



Server starten:

- `svnserve -d -r d:\svn-repos`
 - d: als Hintergrundprozess (in UNIX)
 - r : Basisverzeichnis des Repositories

Zugriff vom Client via Client Kommandos

- `svn <Kommando> [<Optionen>] [<Ziel>]`

Ziel:

- *<Protokoll>://<Hostname>/<Repository>/<Pfad>*
Oder
- Lokale Datei oder Verzeichnis



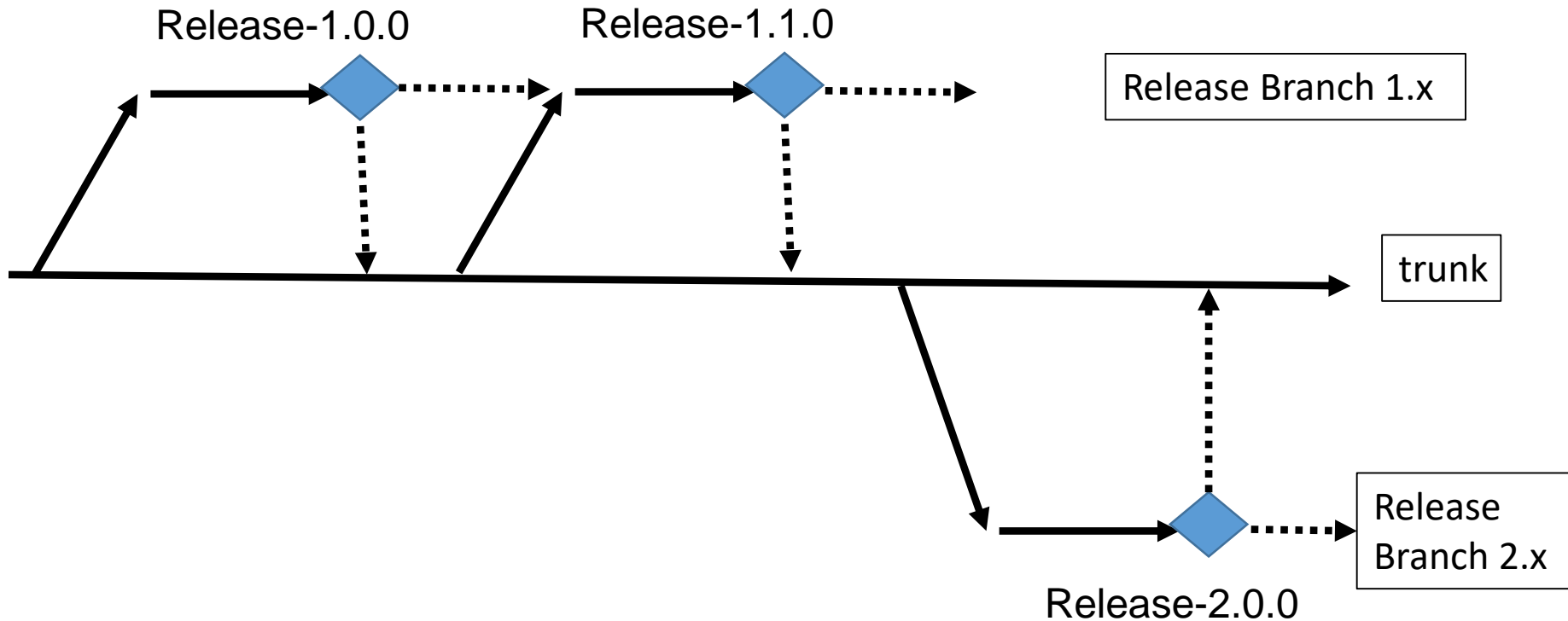
Projektstruktur festlegen

Die Projektstruktur in einem svn Projekt ist nicht nur durch die allgemeinen Überlegungen aus dem letzten Kapitel bestimmt. Es fließen ein:

- Die identifizierten Konfig Elemente
- Die gewünschte logische Struktur
- Das tagging/branching/release Konzept



Beispielhafter Release Plan



Tags und Branches

Template	Beschreibung	Bsp
Trunk	Hauptentwicklungspfad	trunk
RB-<release>	Release Branch	RB-1.0.0 RB-1.1.0
PREP-<release>	Tag zu Beginn der Releasevorbereitung	PREP-1.0.0
REL-<release>	Tag zur Fertigstellung des Releases	REL-1.0.0

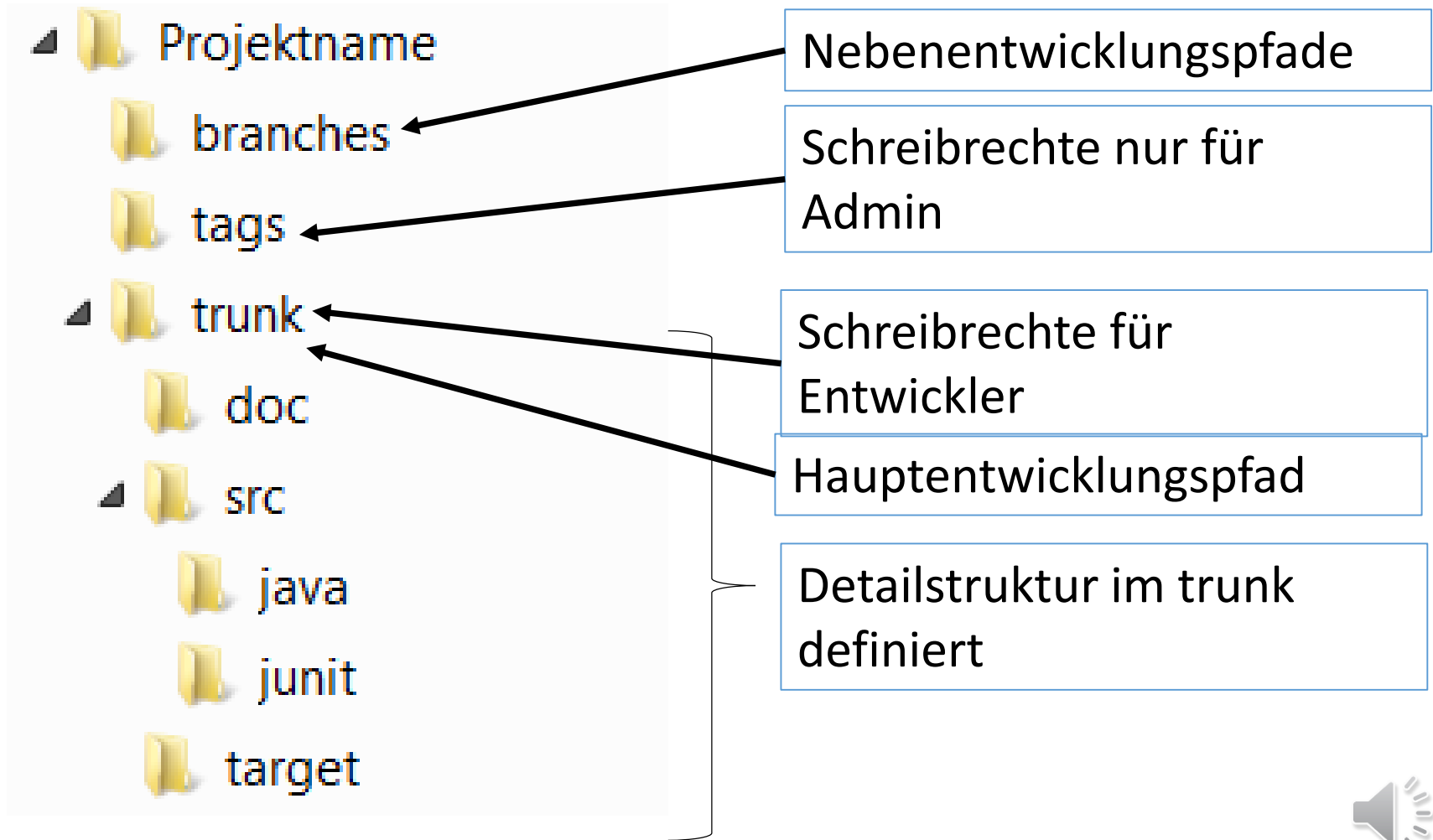
Projektstruktur festlegen

- svn kennt keine echten tags und branches.
- Beide Konzepte werden als normale Verzeichnisse im repository abgebildet.
- Semantik der tags und branches muss berücksichtigt werden:
 - Tags und branches im repository eindeutig identifizieren und abgrenzen
 - Tags nach Erstellung unveränderlich machen.
 - Branches als echte Verzweigung

➔ Realisierung durch Repository Struktur und Rechtevergabe



Projektstruktur festlegen



Inhalt der erweiterten access.conf

[groups]

admins = root

developers = bulenda, adent

[/]

** = r*

@admins = rw

[/branches]

[/tags]

[/trunk]

@developers = rw

Entwickler dürfen nur
noch auf dem trunk schreiben



Konfiguration des svn Clients

- ➔ Datei *config*, Ablagepfad abhängig von der Plattform.
- ➔ Inhalt z.B. `svn:needs-lock` für binäre Dateien, `svn:mime-type` für die korrekte Darstellung im Apache
- ➔ Beispiele siehe Buch von G. Popp Seite 110 und subversion Doku



Anlegen der Struktur im Repository

Anlegen direkt im Repository oder einfacher:

- 1. Anlegen der Struktur lokal auf Clientseite.**
- 2. Importieren der Struktur ins repository:**

```
svn import svn://localhost/meinProjekt/ \  
--message "Import der Projektstruktur" \  
--username root --password root
```



➔ lokale Kopie des Repositories.

- Enthält ein Unterverzeichnis namens `.svn`, dessen Inhalt die Clientseite verwaltet;
 - Auf welcher Revision baut Ihre Arbeitskopie auf
 - Zeitstempel der letzten Aktualisierung

➔ `svn` kann durch Kommunikation mit dem Rep feststellen, in welchem Zustand sich eine Arbeitsdatei befindet:

- Unverändert und aktuell
- Lokal geändert und aktuell
- Unverändert und veraltet
- Lokal geändert und veraltet



Anlegen des Arbeitsbereichs

- `svn checkout svn://localhost/<repo>/trunk`
- Hat nichts mit einem checkout im Sinne von Sperren zu tun, sondern erzeugt lediglich eine Arbeitskopie.



- `svn checkout ...` → Arbeitsbereich anlegen
- `svn update ...` → Arbeitskopie aktualisieren
- `svn commit ...` → Änderungen ins rep schreiben



Gemischte Revisionen

- Aktualisierungen und Übertragungen sind getrennt.
- Gemischte Revisionen sind normal.
- Gemischte Revisionen können nützlich sein.
- Grenzen der gemischten Revisionen
 - Löschen einer nicht aktuellen Datei kann nicht an das Repository übertragen werden
 - Keine Änderungen an den Metadaten eines nichtaktuellen Verzeichnisses möglich.



Konflikte – svn update

➔ Arbeitskopie aus dem repository holen

Aktionszeichen

A

Hinzugefügt

B

Aufgebrochene Sperre

D

Gelöscht

U

Aktualisiert

C

In Konflikt

G

Zusammengeführt

E

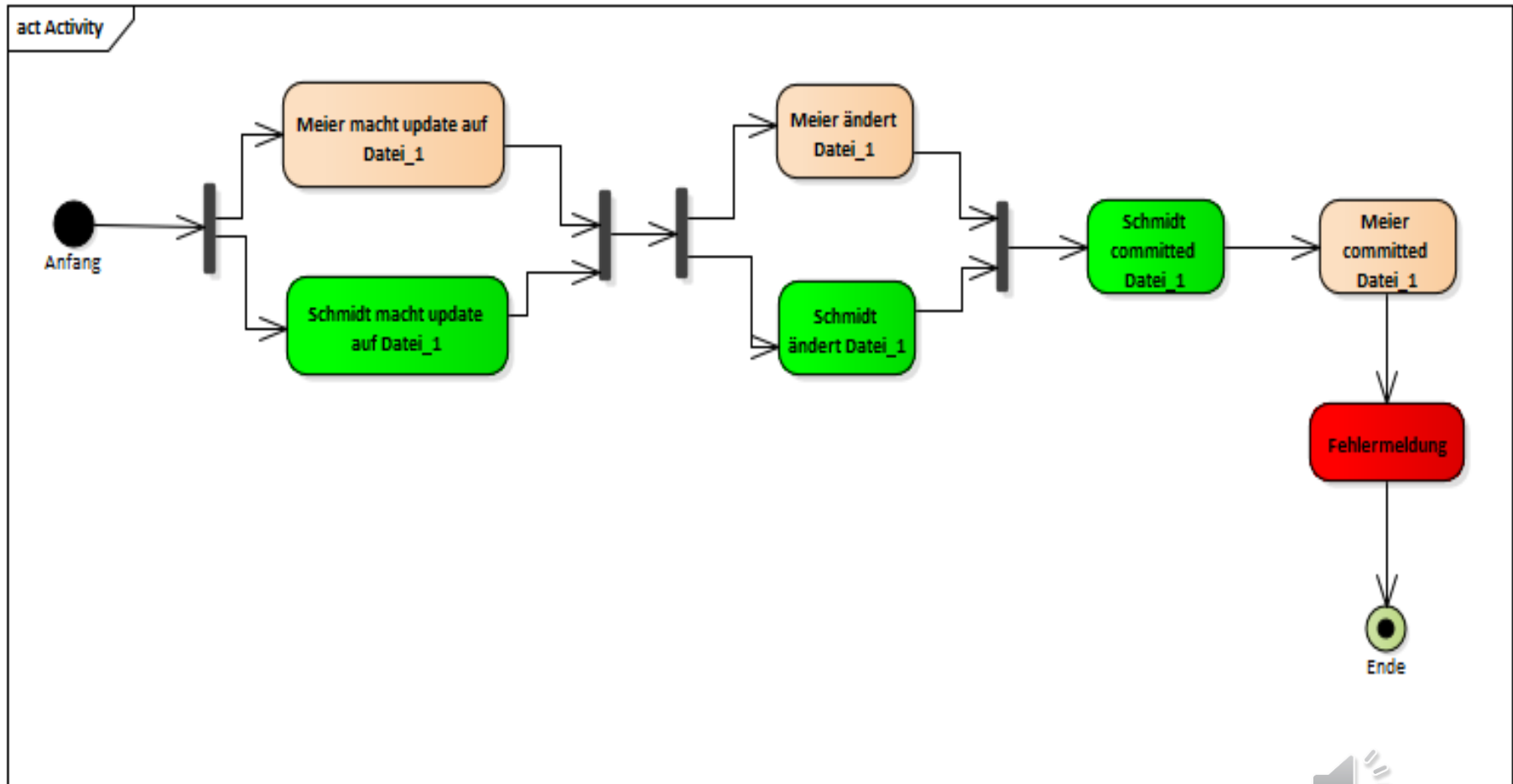
Existierte

➔ **Automatischer Merge ➔ Überprüfen!**



Konflikte – svn commit

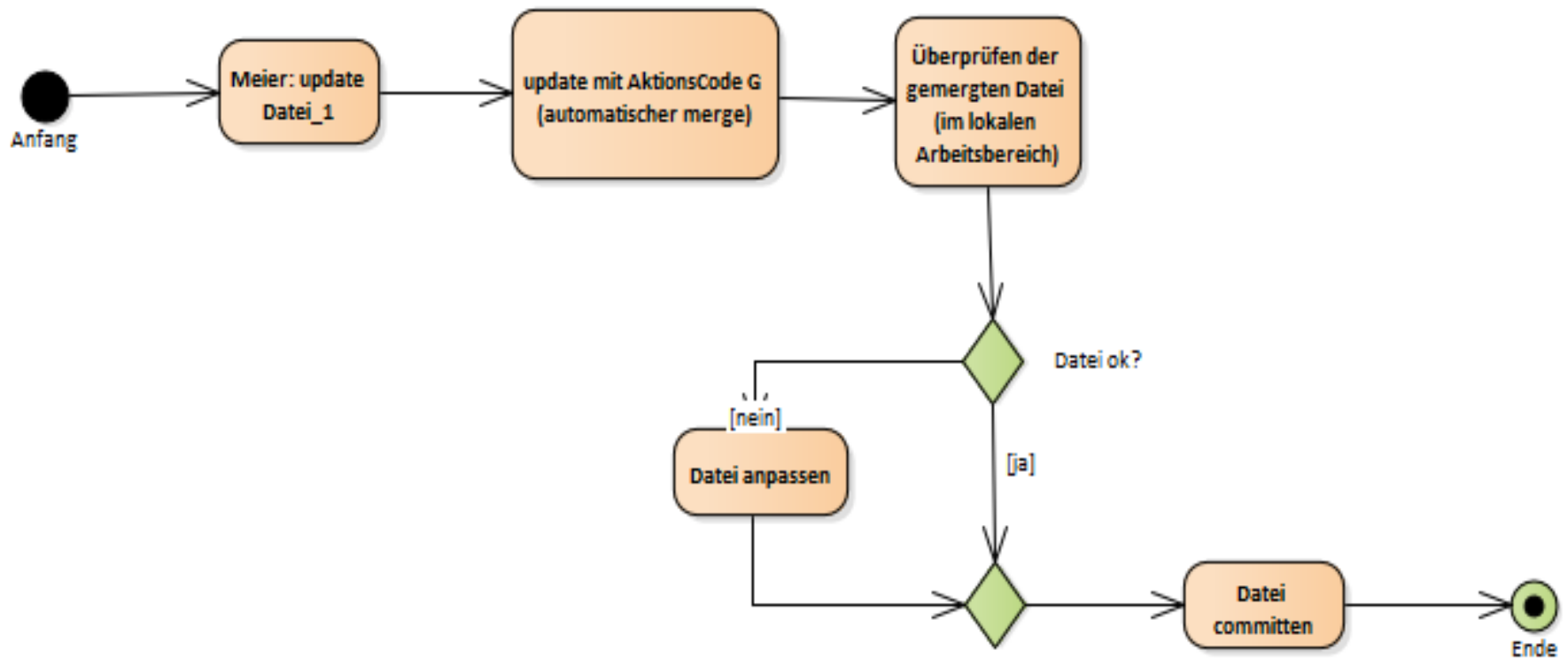
Standard: copy-modify-merge ➔ Konflikte sind normal



Auflösen von Konflikten

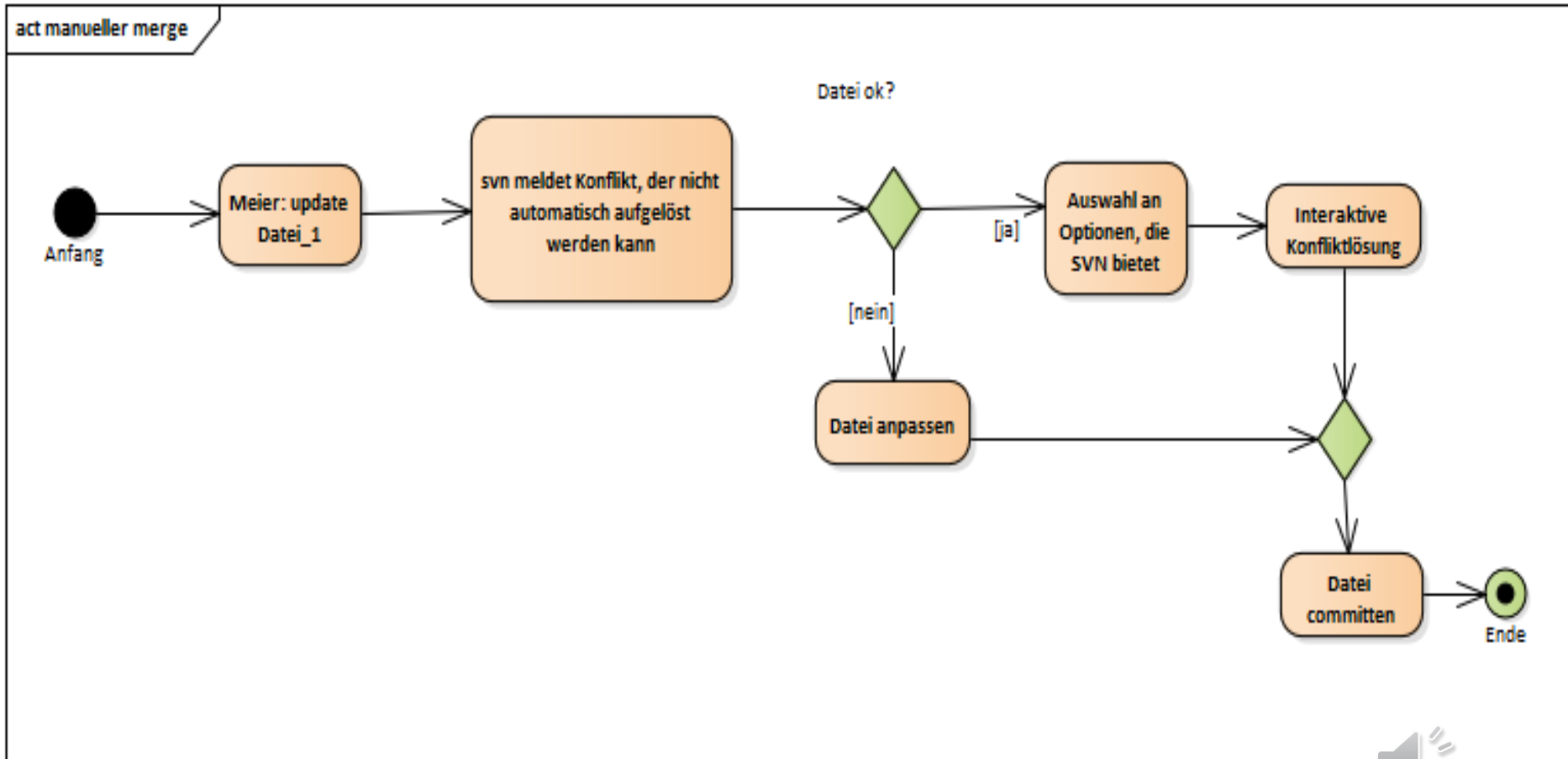
Automatisch oder manuell

act Autoamtische Konfliktlösung



Auflösen von Konflikten

Automatisch oder **manuell** bei echten Konflikten



Arbeitsprozess

1. Arbeitsbereich aktualisieren
2. Prüfung auf Probleme
3. Änderungen durchführen
4. Auf Änderungen im Repository prüfen
5. Änderungen ins Repository schreiben



- Arbeitsbereich aktualisieren
 - svn update ...
- Datei Status erfahren
 - svn status ...
- Elemente hinzufügen
 - svn add <Dateiname>
- Elemente löschen
 - svn delete <Dateiname>



Weitere Aktionen

- Änderungen rückgängig machen
 - **svn revert ...**
- Elemente kopieren
 - **svn copy ...**
- Elemente verschieben und umbenennen
 - **svn move ...**



- Versionshistorie ansehen:
svn log ...
- Unterschiede zwischen Versionen ermitteln
svn diff ...
- Bsp: Verzeichnisbäume vergleichen

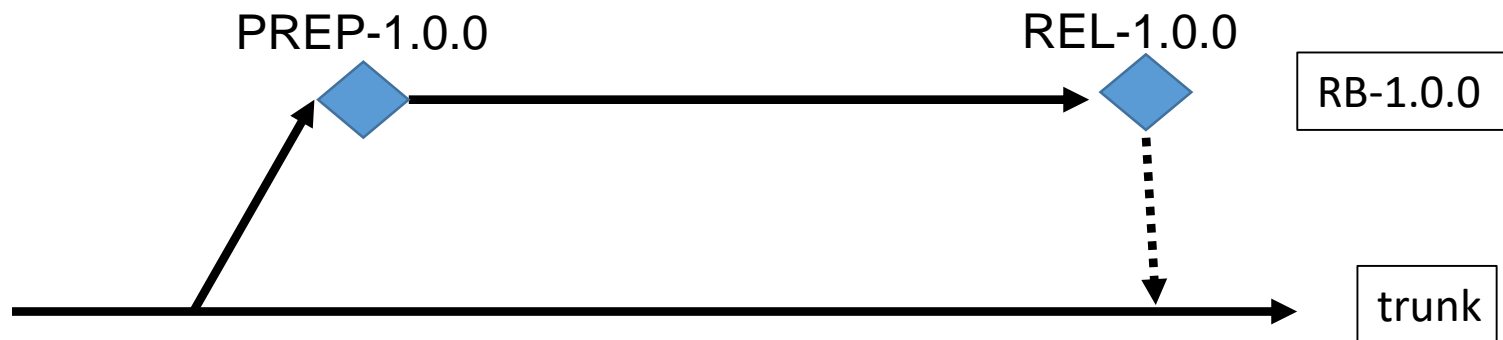
svn diff -revision 4:5 src/java/demo/irgendwas



Festlegung von Tags

Markierung eines bestimmten Entwicklungsstands erfolgt in svn durch einen simplen Kopiervorgang.

Bsp: Vorbereitung des Releases 1.0.0



1. Erzeugen eines Tags für die Releasevorbereitung

```
svn copy -message "Erstelle Tag PREP-1.0.0" \  
--username root - password root \  
svn://localhost/meinProjekt/trunk \  
svn://localhost/meinProjekt/tags/PREP-1.0.0
```

dann noch die Rechte anpassen!

2. Erzeugen eines Release Branches

```
svn copy -message "Erstelle Branch RB-1.0.0" \  
--username root - password root \  
svn://localhost/meinProjekt/tags/PREP-1.0.0 \  
svn://localhost/meinProjekt/branches/RB-1.0.0
```

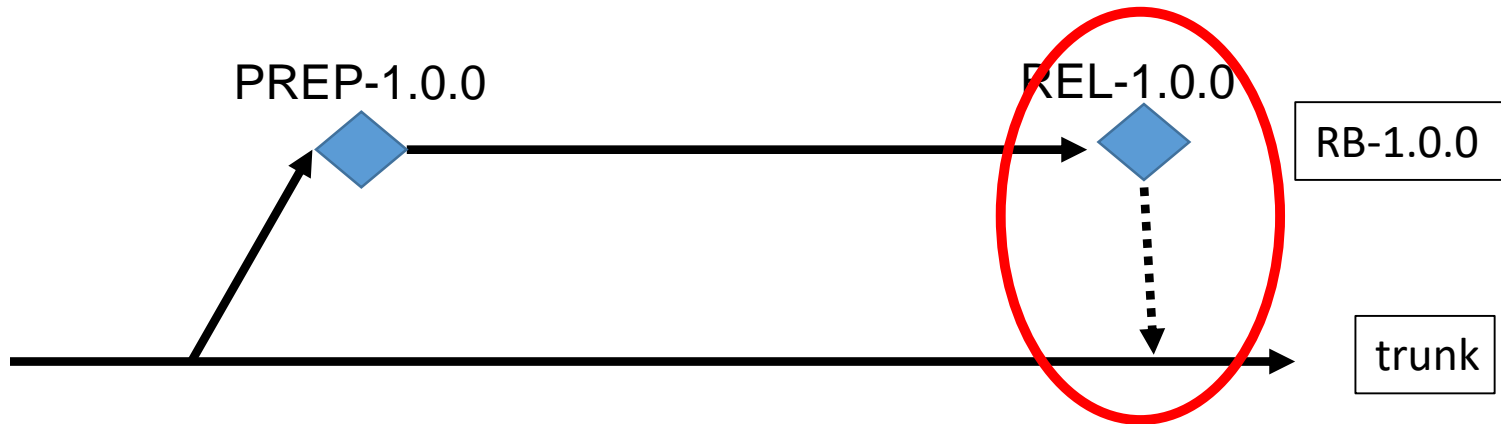


Analog zu trunk:

1. Mithilfe von checkout einen neuen lokalen Arbeitsbereich erstellen.
2. In diesem Arbeitsbereich analog wie im trunk arbeiten:
 1. Update
 2. Change
 3. commit



Branches zusammenführen



In svn möglich:

- Automatisches Zusammenführen von branches
- Gezielte Übernahme einzelner Changesets
- Gezielte Blockade einzelner Changesets



- **Zentrales** Version Control System
- Etabliert
- Sicher
- Für CI (siehe später) gut geeignet

