

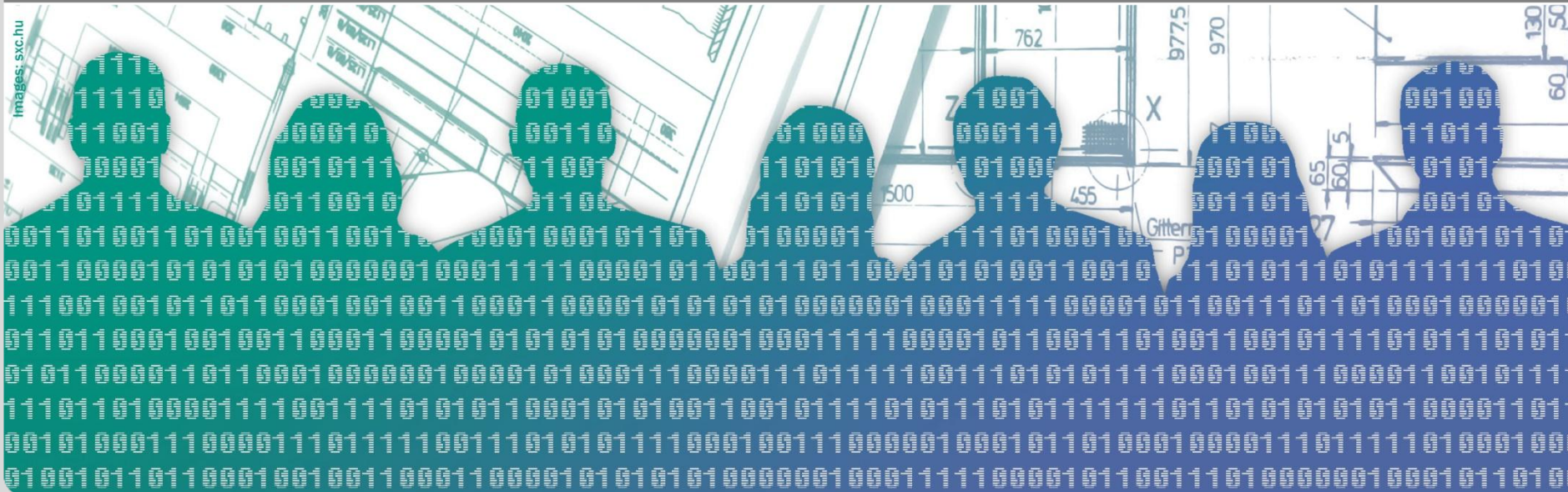
Kapitel 9

Konfigurationsverwaltung

SWT I – Sommersemester 2010

Walter F. Tichy, Andreas Höfer, Korbinian Molitorisz

IPD Tichy, Fakultät für Informatik



Inhalt

- Grundzüge der Konfigurationsverwaltung (KV)
- Revision Control System (RCS)
- Subversion (SVN)

Ursprung der KV

- US-amerikanische Raumfahrtindustrie in 50er und 60er Jahren
 - Zahlreiche nicht dokumentierte Änderungen an Prototypen
 - Prototypen bei Tests zerstört, Pläne veraltet, daher Nachbau unmöglich

→ Konfigurationsverwaltung

Weitere Argumente für KV

- Verschiedene Dateisysteme
- unterschiedliche Rechner
- Diverse Plattformen
- Komponenten von Drittherstellern
- Mehrere Teams/Firmen
- Geographisch verteilte Entwicklerteams

Symptome des Änderungschaos

■ Identifikation und Verfolgung:

- “Dieses Programm funktionierte gestern noch!”
- “Diesen Fehler habe ich schon letzte Woche korrigiert!”
- “Wo sind meine Änderungen von letzter Woche?”
- “Das ist eine offensichtliche Verbesserung. Wurde sie schon probiert?”
- “Wer ist für diese Änderung verantwortlich?”

Mehr Änderungschaos

■ Versionsselektion

- “Wurde alles übersetzt? Nachgetestet?”
- “Wie konfiguriere ich für Test, mit meinen Änderungen aber keinen anderen?”
- “Wie schließe ich diese fehlerhafte Änderung wieder aus?”
- “Ich kann den Fehler in dieser Konfiguration nicht rekonstruieren!”
- “Welche Änderungen sind denn nun in dieser Freigabe?”

Noch mehr Chaos...

■ Software-Lieferung

- “Welche Konfiguration hat dieser Kunde?”
- “Haben wir eine konsistente Konfiguration geliefert?”
- “Hat der Kunde modifiziert?”
- “Der Kunde hat die letzten zwei Freigaben nicht installiert. Was passiert, wenn wir ihm die neue schicken?”

■ Diese Probleme klingen für jeden vertraut, der in der Software-Entwicklung gearbeitet hat.

Konfigurationsverwaltung (KV)

- ISO 9001:
„Konfigurationsmanagement stellt einen Mechanismus zur **Identifizierung**, **Lenkung** und **Rückverfolgung** der Versionen jedes Software-Elementes dar.“

(Software-) Konfiguration

- Eine (Software-) Konfiguration ist eine **benannte** und formal freigegebene Menge von **Software-Elementen**, mit den jeweils gültigen **Versionsangaben**, die zu einem bestimmten Zeitpunkt im Produktlebenszyklus in ihrer Wirkungsweise und ihren Schnittstellen aufeinander abgestimmt sind und gemeinsam eine vorgesehene Aufgabe erfüllen sollen.

Software-Element (SE)

- Ein Software-Element ist jeder **identifizierbare Bestandteil** des entstehenden Produktes oder der entstehenden Produktlinie.

Software-Element

- Besitzt systemweit eindeutigen **Bezeichner**
- Änderung am Element erzeugt neuen Bezeichner, um Fehlidentifikation zu vermeiden
- Unterscheide
 - Quellelement: manuell erzeugt, z.B. mit Editor
 - Abgeleitetes Element: automatisch generiert, z.B. durch Übersetzer

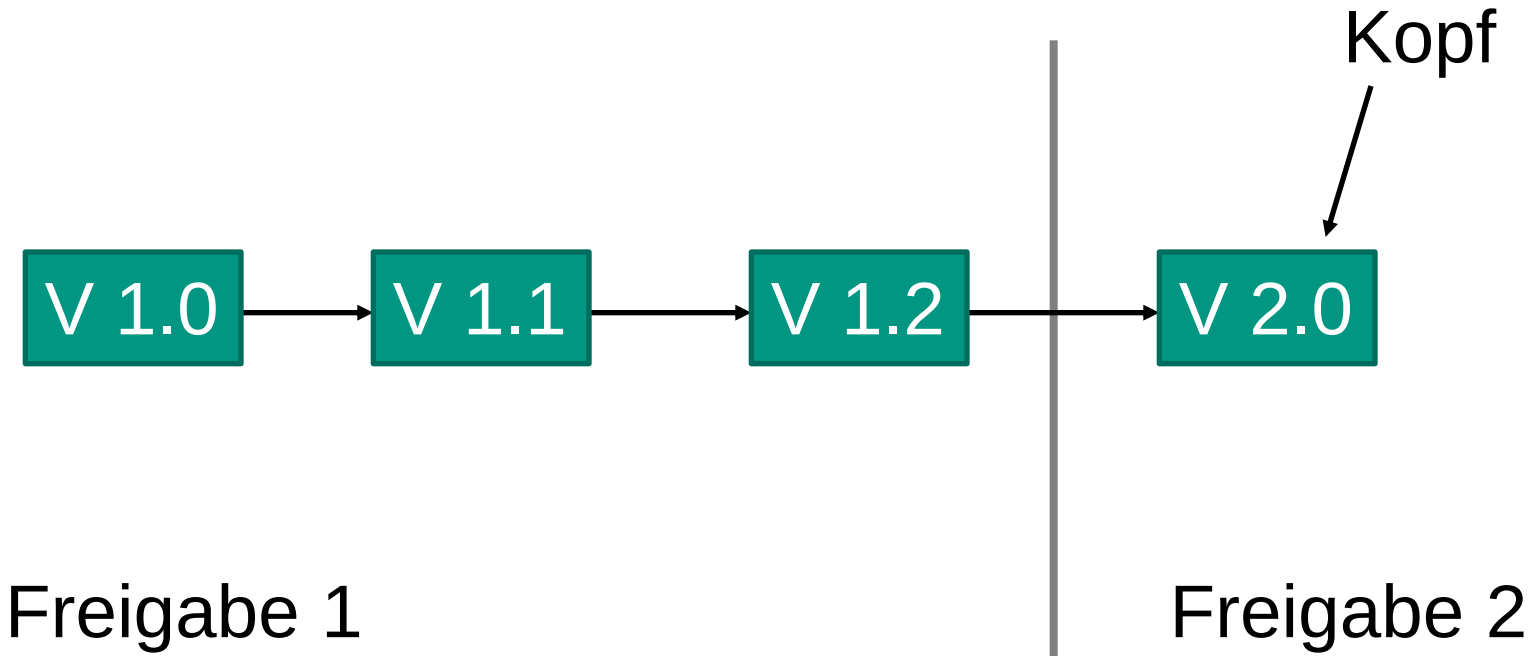
Versionen

- Eine **Version** ist die Ausprägung eines Software-Elementes zu einem bestimmten Zeitpunkt.
- **Revisionen** sind zeitlich nacheinander liegende Versionen (Entwicklungsstände).
- **Varianten** sind alternative Versionen (Anpassungen, oder mit alternativen Datenstrukturen/Algorithmen).

Versionsnummern (VN)

- Bestehen aus mindestens zwei Teilen
 - **Freigabe**-Nummer (engl. *release*)
 - **Laufende** Nummer (engl. *level*)
- Versionsnummer: **Release.Level**
- Neues SE erhält VN 1.0
- Bei jeder Änderung wird laufende Nummer erhöht: z.B. 1.1 auf 1.2

Versionsnummern



Einbuchen/Ausbuchen (Check-In/Check-Out)

- SE werden in Archiven gesammelt
- **Ausbuchen (Check-Out)**
 - Holt Kopie aus Archiv
 - Reserviert Kopie für den Ausbucher, was heißt, dass nur dieser die nächste Revision ablegen darf (**striktes Ausbuchen**)
 - Kopie darf geändert und wieder eingebucht werden

Einbuchen/Ausbuchen

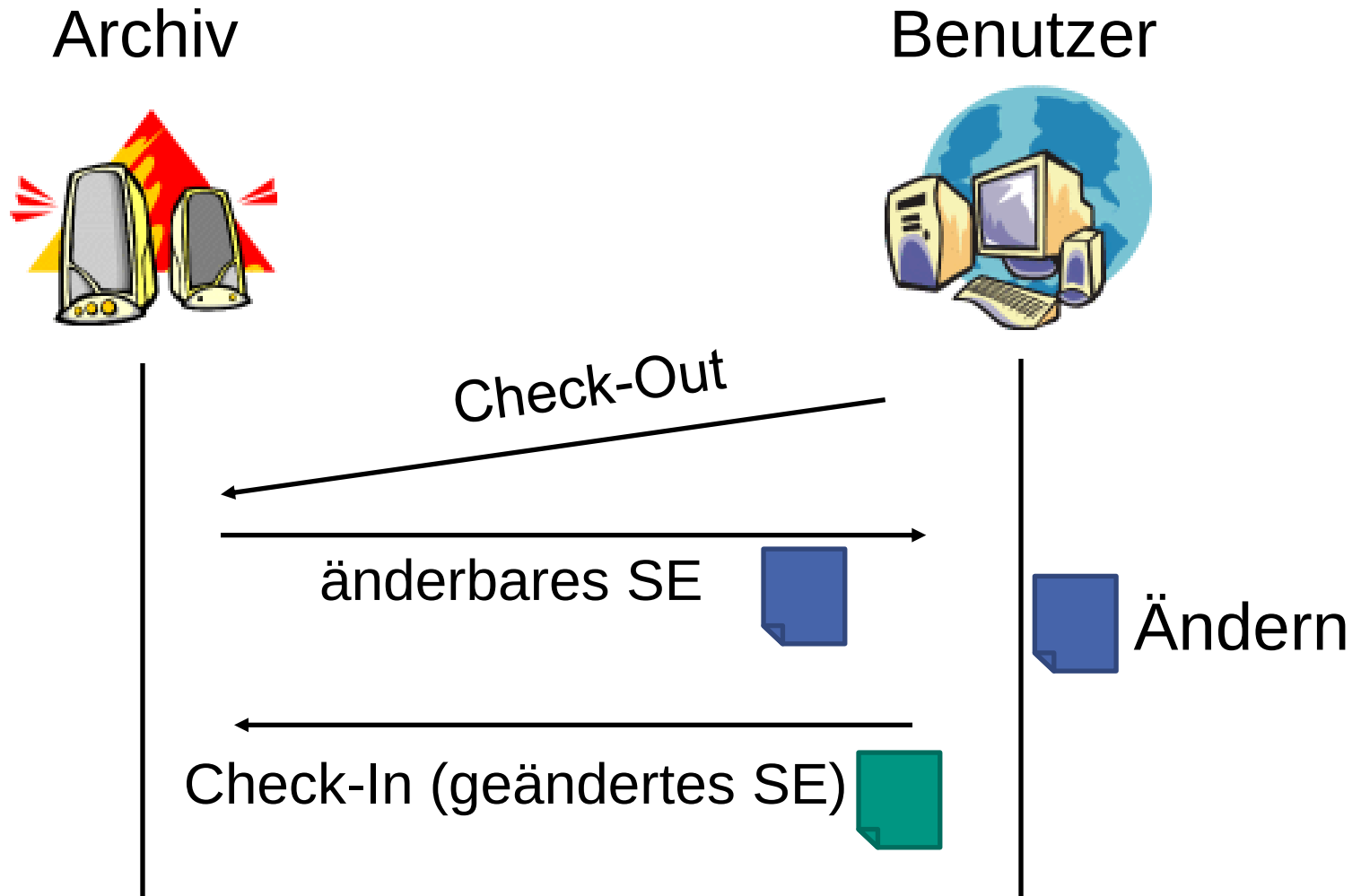
■ Einbuchen (Check-In)

- Schreibt Kopie in Archiv zurück
- Löscht Reservierung
- Erweitert Historie um
 - Autor des Elements/der Änderung
 - Einbuchungszeitpunkt
 - Logbucheintrag, der Änderungen zusammenfasst

Einbuchen/Ausbuchen

- Eingebuchtes Element ist nicht mehr änderbar
- Erst erneutes Ausbuchen erlaubt Änderungen

Check-In / Check-Out



Einbuchen/Ausbuchen

- Im System können mehrere Dateien gleichzeitig ausgebucht sein
- Lesezugriff durch Reservierung nicht verhindert
- Unterscheide
 - **striktes** Ein-/Ausbuchen
 - **Optimistisches** oder mehrfaches Ein-/Ausbuchen

Striktes Ausbuchen

- Nur eine Ausbuchung gleichzeitig ist erlaubt
- Ausbucher hat exklusives Änderungsrecht
- Vorteil:
 - kein Verschmelzungsaufwand beim Zurückschreiben
- Nachteil:
 - immer nur einer kann eine Version ändern

Optimistisches Ausbuchen

- Mehrere Ausbuchungen gleichzeitig erlaubt
- Mehrere Entwickler Arbeiten an der gleichen Programmversion
- Vorteil:
 - Mehrere Entwickler können eine Version ändern
- Nachteil:
 - Aufwand beim Zusammenführen der Versionen (der Schnellere gewinnt)

Mehrfaches Ausbuchen

- Zusammenführen der Versionen
 - Stand der ausgebuchten Version kann sich im Archiv geändert haben
 - Verschmelzen (engl. *merging*) der Änderungen nötig

Varianten

- Sequentielle Versionsstämme reichen für Praxis oft nicht aus
- Varianten erlauben das Ändern ähnlicher Versionen eines SE

Varianten können ...

- Parallele **Entwicklungslinien** darstellen
- Unterschiedliche Implementierung derselben Schnittstelle sein, bei gleicher Funktion
- Anpassungen an unterschiedliche Bibliotheken, Geräte, GUIs, Nutzer sein.

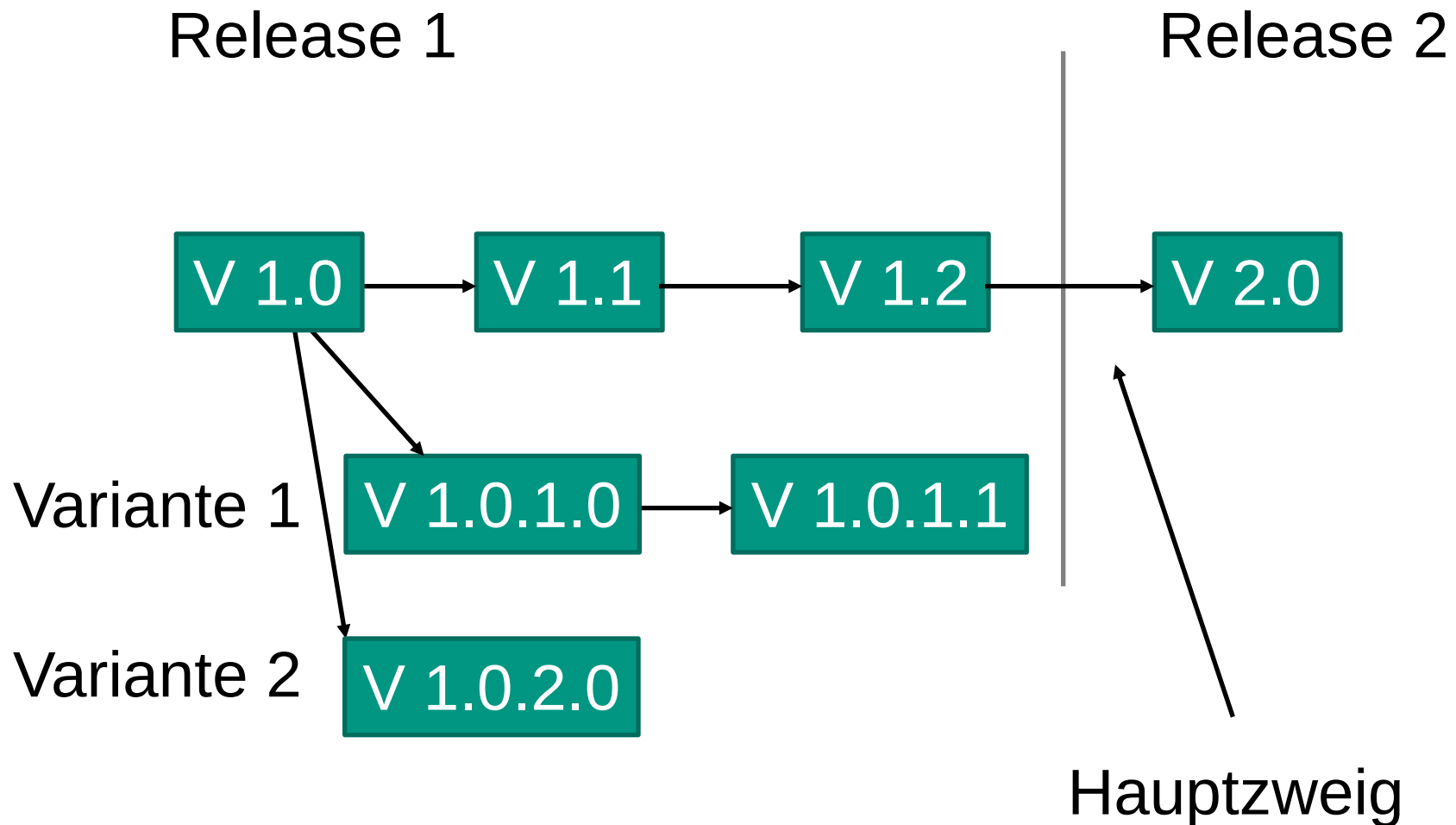
Varianten können ...

- Auf unterschiedliche Hardware oder Systemsoftware-Konstellationen zugeschnitten sein
- Ab bestimmten Abstraktionsniveau nicht mehr unterschieden werden

Variantennummern

- Setzen sich zusammen aus Nummer der Grundversion und Nummer der Variante
 - Release.Level.**Variante**.Level
- Varianten werden ab 1 gezählt
- Erster Variantenlevel ist 0
- 1. Variante von Version 2.3 ist 2.3.1.0
- 3. Level der 4. Variante von Version 1.7 ist 1.7.4.3

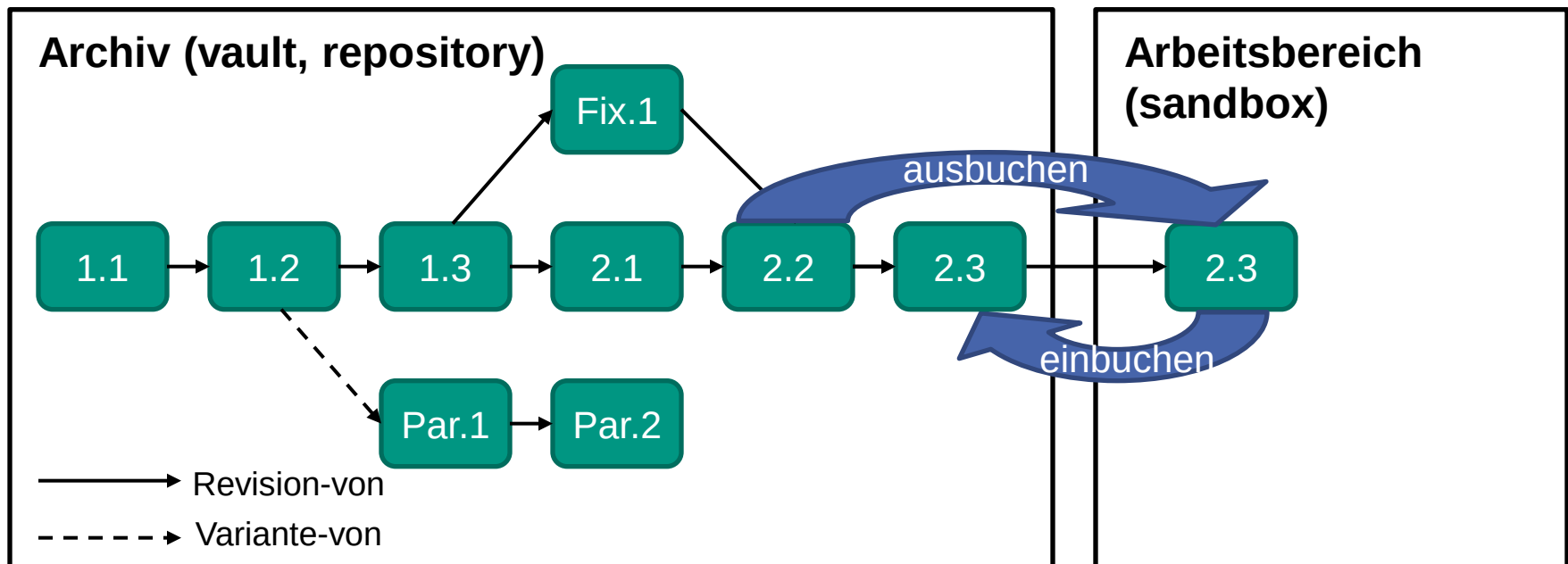
Varianten



Organisation von Versionsgruppen

■ Änderungszyklus:

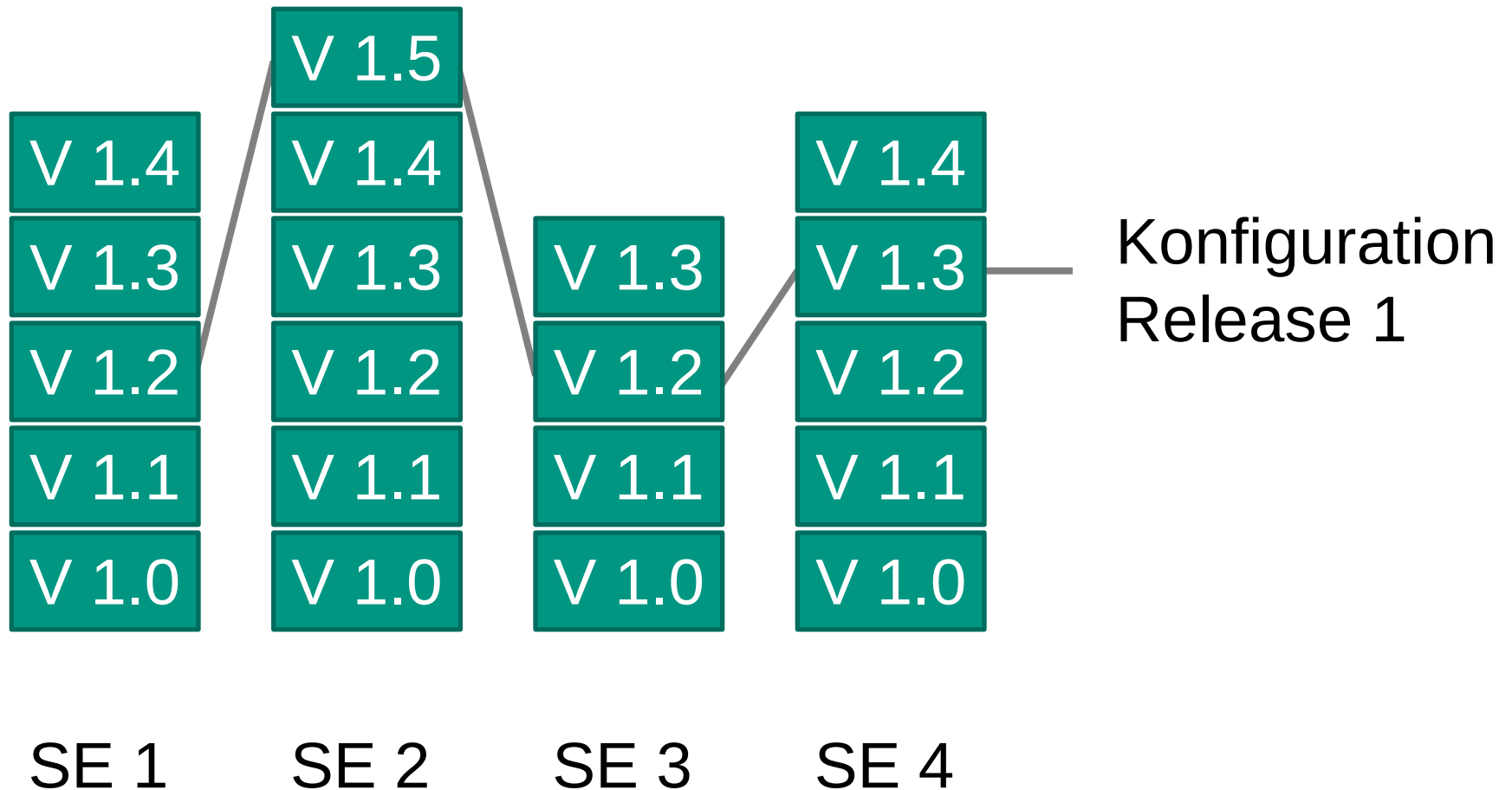
- ausbuchen (erzeuge Kopie in Arbeitsbereich, Platzhalter in DB)
- modifiziere Kopie in Arbeitsbereich
- einbuchen (befördere Kopie in Platzhalter)



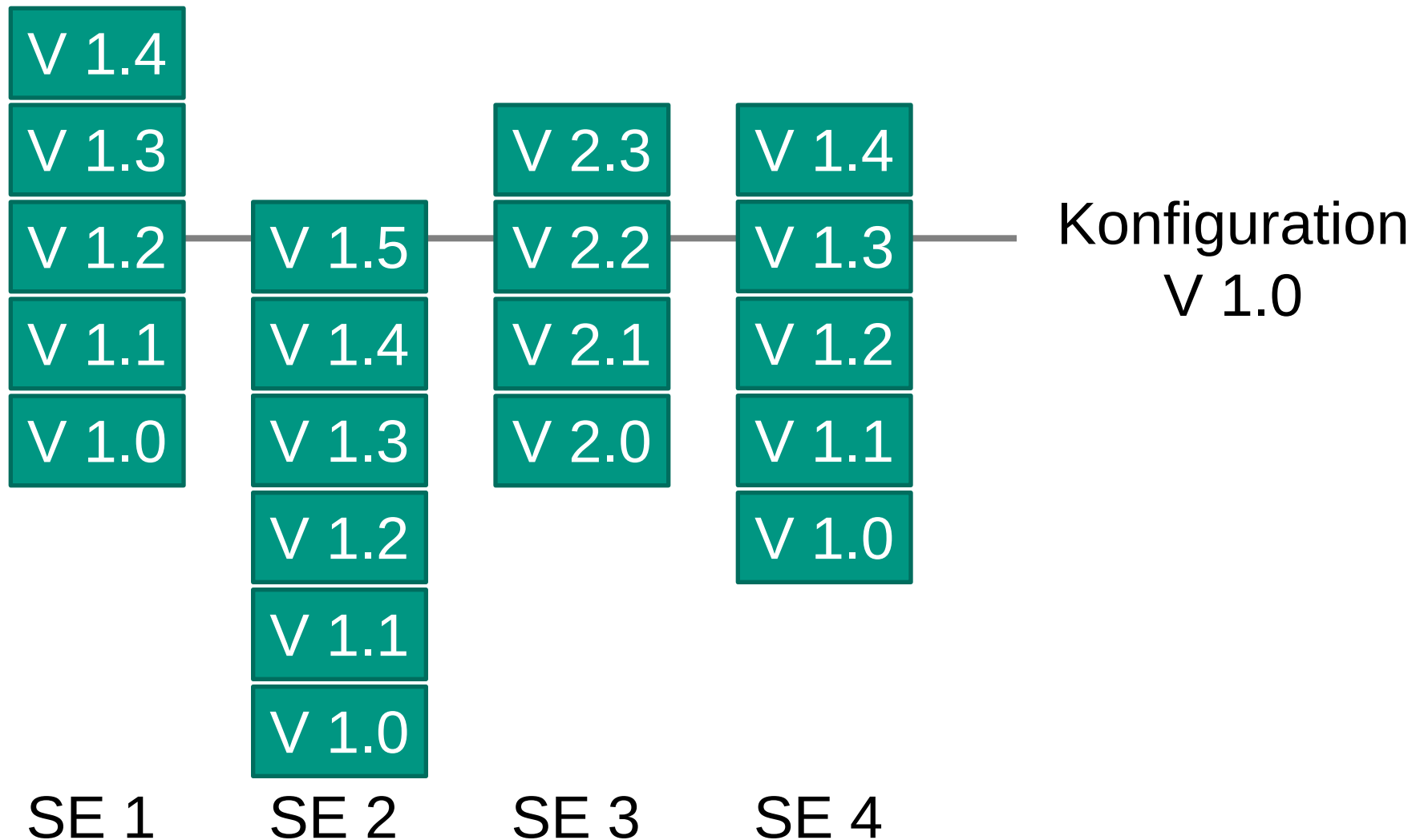
Zurück zur Konfiguration

- Konfiguration umfasst **bestimmte** Versionen von SE
- Konfigurations-Identifikationsdokument (KID) bestimmt **Zuordnung** von Konfiguration zu Versionen der SE
- KID ist selber im KV, d.h. besitzt Versionsnummer

Konfiguration



Konfiguration



■ Konfiguration V 1.0

- SE 1, V 1.2
- SE 2, V 1.5
- SE 3, V 2.2
- SE 4, V 1.3

Bestandteile einer (Software-) Konfiguration

- Quellelemente, z.B.
 - Programmtext
 - Dokumentation
 - Konfigurationsdateien für zu bauendes System
- Abgeleitete Elemente, wenn sie teuer in Erstellung sind

Bestandteile einer (Software-) Konfiguration

- Werkzeuge (Übersetzer ...)
 - Systemrelevante, wichtig für Ausführung des Systems (z.B. Laufzeitübersetzer)
 - Entwicklungsrelevante, bekommt Kunde nicht zu sehen
- KID von Subsystemen (hierarchische Struktur einer Konfiguration – siehe Kompositum)

Konfiguration für Zwischenschritte des SW-Entwicklungsprozesses

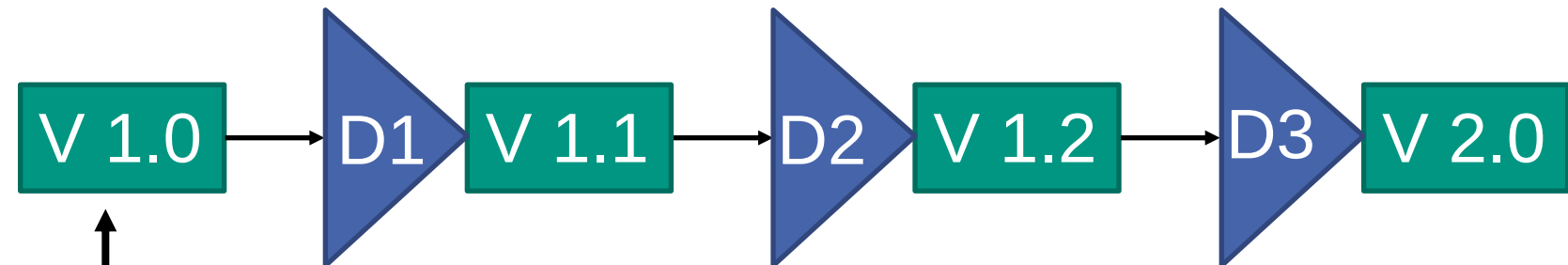
- Konfiguration für Ergebnis der Entwicklungsphasen, z.B.
 - Anforderungsanalyse
 - Entwurf
- Ermöglichen
 - Rückverfolgen von Änderungen
 - Zurücksetzen auf korrekte Konfiguration

Ausblick: Wie werden Versionen verwaltet

- Vollständiges Abspeichern jeder Versionen ist platzaufwendig.
- Alternative
 - **Vorwärts-Deltas:** Speichere Grundversion und die daran durchgeführten Änderungen
 - **Rückwärts-Deltas:** Speichere aktuelle Version und die Änderungen für frühere Versionen
- Ein Delta ist der Unterschied zwischen zwei Versionen; eine komprimierter Änderungs-Skript, der eine Version in die andere überführt. Bei Software ist ein Delta i.d.R. etwa 1-2% des Umfanges einer (Voll-)Version.

Vorwärts-Deltas

blau markierte Elemente werden gespeichert !



Version V 1.0
vollständig gesichert

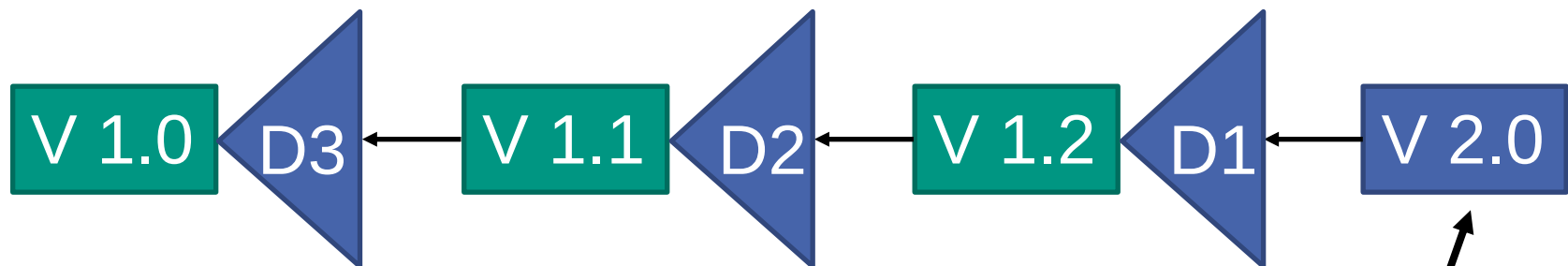
Für weitere Versionen
speichere Änderungen
in Deltas:
 $V\ 1.2 = D2(D1(V1.0))$

Vorwärts-Deltas

- Vorteile
 - Schneller Zugriff auf frühere Versionen
- Nachteile
 - Langsamer Zugriff auf aktuellere Version
- Aktuelle Version wird häufiger benötigt
- Drehe Anwendung der Deltas um: Rückwärts-Deltas

Rückwärts-Deltas

blau markierte Elemente werden gespeichert !



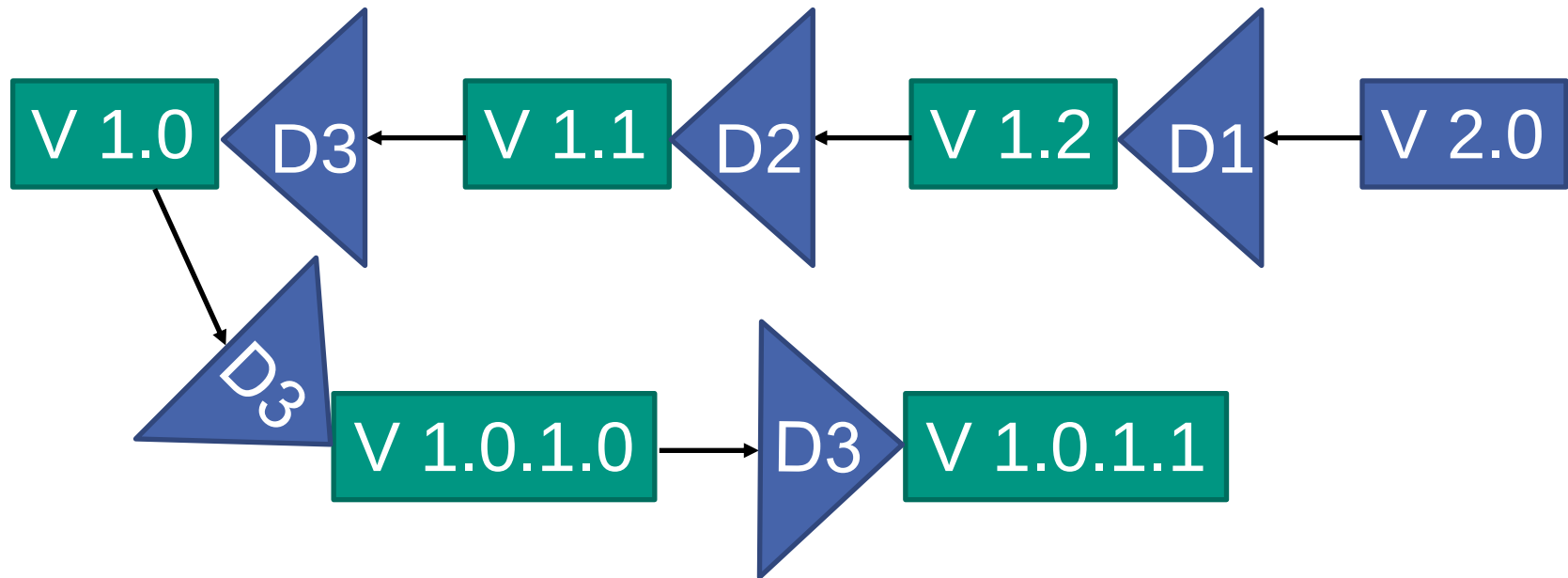
Für weitere Versionen
speichere Änderungen
in Deltas:

$$V\ 1.1 = D2(D1(V2.0))$$

Version V 2.0
vollständig gesichert



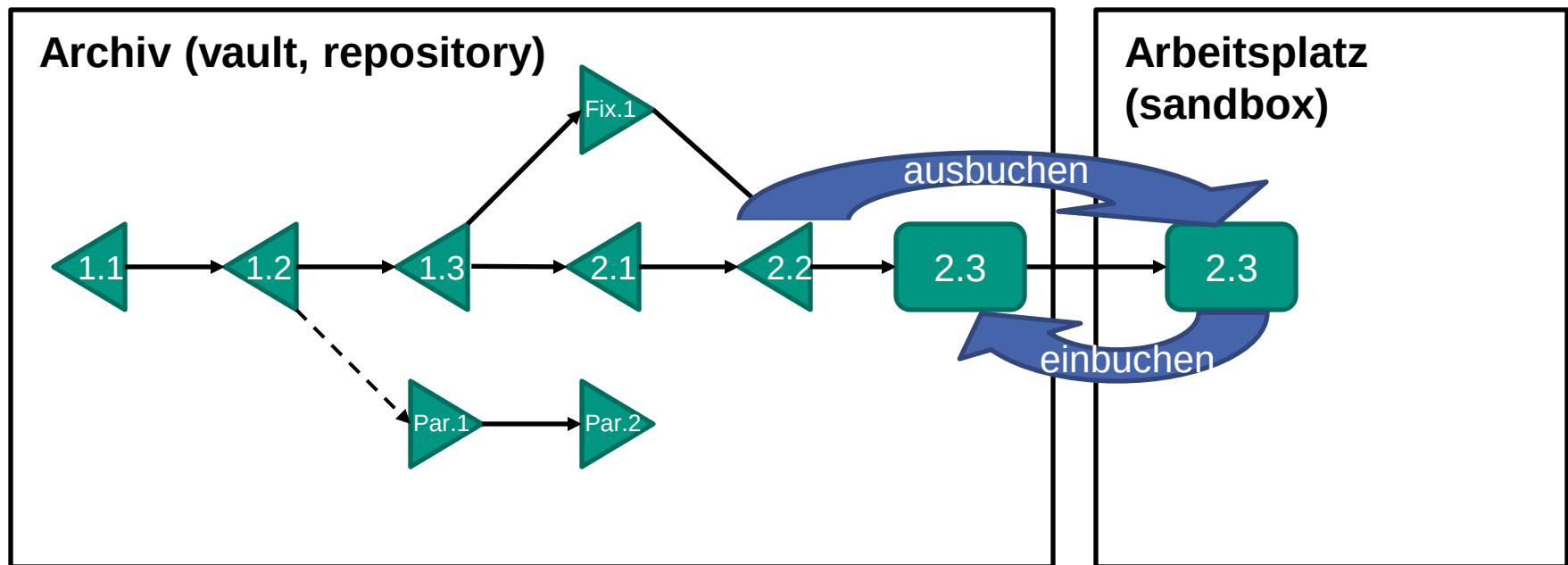
Was ist bei Varianten?



Verwende Vorwärts-Deltas für Varianten
Problem: etwas teurere Variantenberechnung
(Methode von RCS)

Deltas sparen Platz

- Ein Delta ist der Unterschied zw. zwei Revisionen. Kann vorwärts oder rückwärts berechnet werden. Wird zur Wiederherstellung von Revisionen benutzt.



Revision Control System (RCS)

- Autor: Tichy, 1983
- Verwaltet **mehrere Versionen** einer Datei
- Automatisiert
 - Aufbewahrung
 - Wiederherstellung
 - Logging
 - Identifikation
 - Verschmelzen von Versionen

Benutzerschnittstelle

- Grundlegende Kommandos
 - **ci** – Check-In (Bedeutung wie bisher erklärt)
 - **co** – Check-Out (Bedeutung wie bisher erklärt)
 - **rcs** – Verwaltung des Archives
 - **rlog** – Zeige Verwaltungsinformation und Logbuch
- RCS-Archiv zur Datei *<Dateiname>* heißt *<Dateiname>,v*
- Dateien im Archiv heißen RCS-Dateien
- Alle anderen Dateien heißen Arbeitsdateien

Bedienung: Anlegen eines Archivs

■ `rcs -i <Dateiname>`

- Legt neues Archiv für Datei *<Dateiname>* an
- Archiv ist leer
- Archiv wird in Unterverzeichnis RCS angelegt, falls existent, sonst im lokalen Verzeichnis
- Beschreibung der Datei wird verlangt, nicht der Logbucheintrag!

Bedienung: Einbuchen

■ `ci <Dateiname>`

- Legt Datei *<Dateiname>* im RCS-Archiv ab
- Löscht Datei im Arbeitsverzeichnis
- Existiert Archiv nicht, erst `rcs -i` ausführen mit zusätzlichem Einbuchen, oder `ci -i` initialisiert das Archiv beim ersten Einbuchen.

Bedienung: Ausbuchen

■ `co <Dateiname>`

- Bucht Datei *<Dateiname>* aus RCS-Archiv heraus
- Die neueste Version wird ausgebucht
- Version in Arbeitsverzeichnis abgelegt
- Datei ist nicht zum editieren geeignet, da keine Reservierung angelegt wurde !

Bedienung: Ausbuchen mit Lock

■ `co -l <Dateiname>`

- Vorgehen wie bei `co <Dateiname>`
- Zusätzlich bekommt die Version im Archiv eine Reservierung (lock)
- Diese Version kann von niemand anderen mehr bearbeitet werden (striktes Check-Out)
- Erst nach `ci <Dateiname>` kann anderer diese Version zur Veränderung ausbuchen

Zugriffsschutz

- Ausgebuchte Version ist mit einer Sperre (engl. lock) geschützt (`co -1`)
- RCS-Datei ist durch Zugriffsliste geschützt: nur Personen in der Zugriffsliste können Datei verändern

Logbuch ansehen

- `rlog <Dateiname>` zeigt folgende Informationen an
 - Pfad zu RCS-Datei
 - Pfad zu Arbeitsdatei
 - Kopf-Version
 - Zweige
 - Zugriffsliste
 - Symbolische Namen

Logbuch – Beispiel: \$ rlog App.java

RCS file: RCS/App.java,v

Working file: App.java

head: 1.3

branch:

locks: strict

access list:

symbolic names:

keyword substitution: kv

total revisions: 3; selected revisions: 3

description:

First Java Application

revision 1.3

date: 2003-05-27 03:11:54-04; author: dfaeckt;

state: Exp; lines: +1 -0

Change request R456 implemented.

revision 1.2

date: 2003-05-27 03:08:56-04; author: dfaeckt;

state: Exp; lines: +1 -0

Moved method getName to class App.

revision 1.1

date: 2003-05-27 03:07:51-04; author: dfaeckt;

state: Exp;

Initial revision

=====



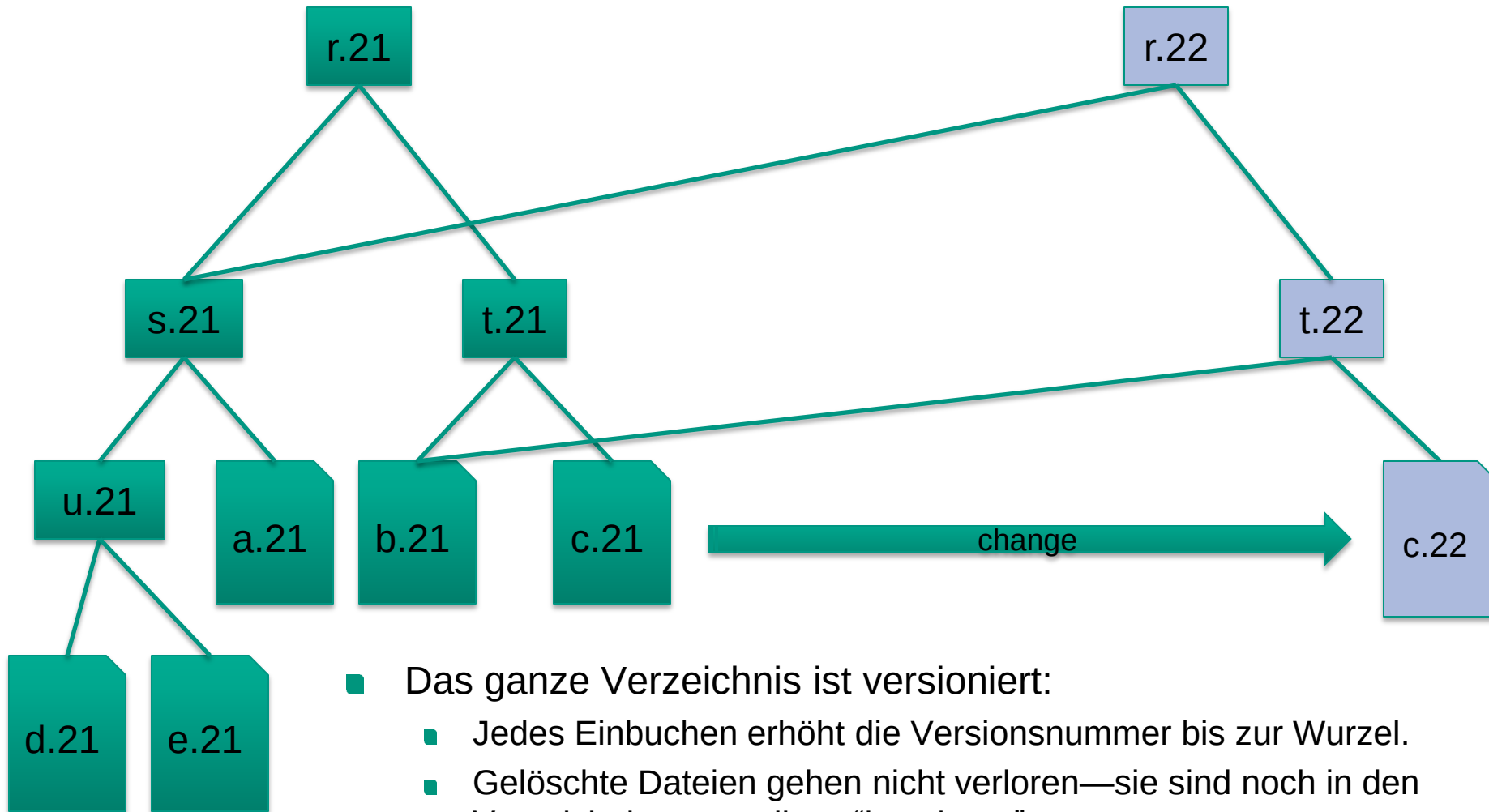
RCS - Zusammenfassung

- Einfache Schnittstelle für Versionierung einzelner Dateien
- Durch Kommandozeilenoptionen Zugriff auf alle Versionen (hier nicht gezeigt)
- Sicht auf mehrere Dateien nicht vorhanden
- Nicht Netzwerkfähig
- Unterstützt keine Versionierung von Verzeichnissen

Subversion (SVN)

- Weiterentwicklung von CVS (Concurrent Versions System)
- Internetfähig, d.h. das Archiv wird auf einem Dienstgeber abgelegt, auf den Arbeitsplatzrechner über das Internet zugreifen.
- SVN versioniert das gesamte Projektarchiv, inklusive dem Verschieben, Umbenennen und Kopieren von Verzeichnissen und Dateien.
- Das Einbuchen von Änderungen geschieht atomar, d.h. eine Änderung wird ganz oder gar nicht in das Projektarchiv übertragen. Schlägt das Einbuchen einer Datei fehl, werden auch Änderungen an anderen Dateien nicht übertragen.

Versionierung in SVN



- Das ganze Verzeichnis ist versioniert:
 - Jedes Einbuchen erhöht die Versionsnummer bis zur Wurzel.
 - Gelöschte Dateien gehen nicht verloren—sie sind noch in den Verzeichnissen vor ihrer “Löschung”.

Verzweigung für Varianten

- SVN bietet keine eingebaute Semantik für Verzweigungen oder Variantenbildung.
- Statt dessen erzeugt man Unterverzeichnisse, die die Alternativen enthalten.
- Per Konvention benutzt man folgende Verzeichnisse:
 - **trunk**: Hauptentwicklungslinie
 - **branches**: alternative Entwicklungslinien, evtl. später mit Hauptentwicklungslinie verschmolzen.
 - **tags**: Kopien von anderen Verzeichnissen zu einem bestimmten Entwicklungsstand, benutzt um Freigaben einzufrieren.

Benutzerschnittstelle

- `svn` *<Kommando>*, z.B.
 - `import` – Einpflegen in Archiv
 - Projekt in Archiv erstellen
 - `checkout` – Verzeichnis/Projekt aus Archiv laden
 - `commit` – Änderungen in das Archiv übernehmen
 - `update` – Lokale Kopie aktualisieren
 - `add` – Neue Datei hinzufügen
 - `delete` – Datei aus Archiv löschen
 - `move` – Datei/Verzeichnis verschieben
 - `copy` – Datei kopieren
 - `mkdir` – Verzeichnis anlegen

Anlegen eines SVN Archivs

- Liegt auf bestimmten Rechner im Netz, auf den alle zugreifen können.
- Der SVN-Server kann entweder ein eigenständige Installation sein oder in eine bestehende Apache-Webserver-Installation integriert werden.

■ `svnadmin create`

- Legt neues Archiv auf dem Server an.
- Beispiel:
`svnadmin create /var/svn/repos`

Verzeichnis/Datei einbuchten

■ `svn import [<Verzeichnis>] <URL>`

- Pflegt alle Dateien und Unterverzeichnisse des lokalen Verzeichnisses ein. Standardmäßig wird für **<Verzeichnis>** das aktuelle Verzeichnis „.“ angenommen.
- **<URL>** ist die Adresse zum Projektarchiv, z.B. `https://.../repos`
- Mittels der Option `-m` kann eine zusätzliche Nachricht zur aktuellen Aktion angegeben werden.

Ausbuchen eines SVN Archivs

- Ausbuchen eines Archivs bspw. mit dem Befehl:
`svn checkout https://.../repos /tmp/projects`
- Das Archiv kann auch über weitere Protokolle wie bspw. **http://** etc. zur Verfügung gestellt werden.
- Benutzername und Passwort können mit Hilfe der Optionen **--username** und **--password** übergeben werden:
`svn checkout https://.../repos /tmp/projects
--username chuck --password sirron`

Ausbuchen eines SVN Archivs

- `svn checkout <URL> [<Verzeichnis>]`
 - Legt lokal *<Verzeichnis>* an
 - Kopiert alle Dateien des ausgewählten Verzeichnisses im Archiv nach *<Verzeichnis>*
 - Legt Verzeichnis *<Verzeichnis>/svn* mit Verwaltungsinformationen an
 - *svn* nicht löschen, nicht anfassen!

Lokale Dateien mit Archiv synchronisieren

- `svn update <Verzeichnis/Dateien>`
 - Aktualisiert lokale Dateien mit neuen Versionen aus dem Archiv, die seit Ausbuchen neu abgelegt wurden.
 - Mögliche Konflikte werden in separater Datei angezeigt (eigene Modifikationen werden nicht überschrieben!)
- Auflösung von Konflikten durch Verschmelzung muss man vor Einbuchen durchführen, da **svn commit** nicht synchronisiert!

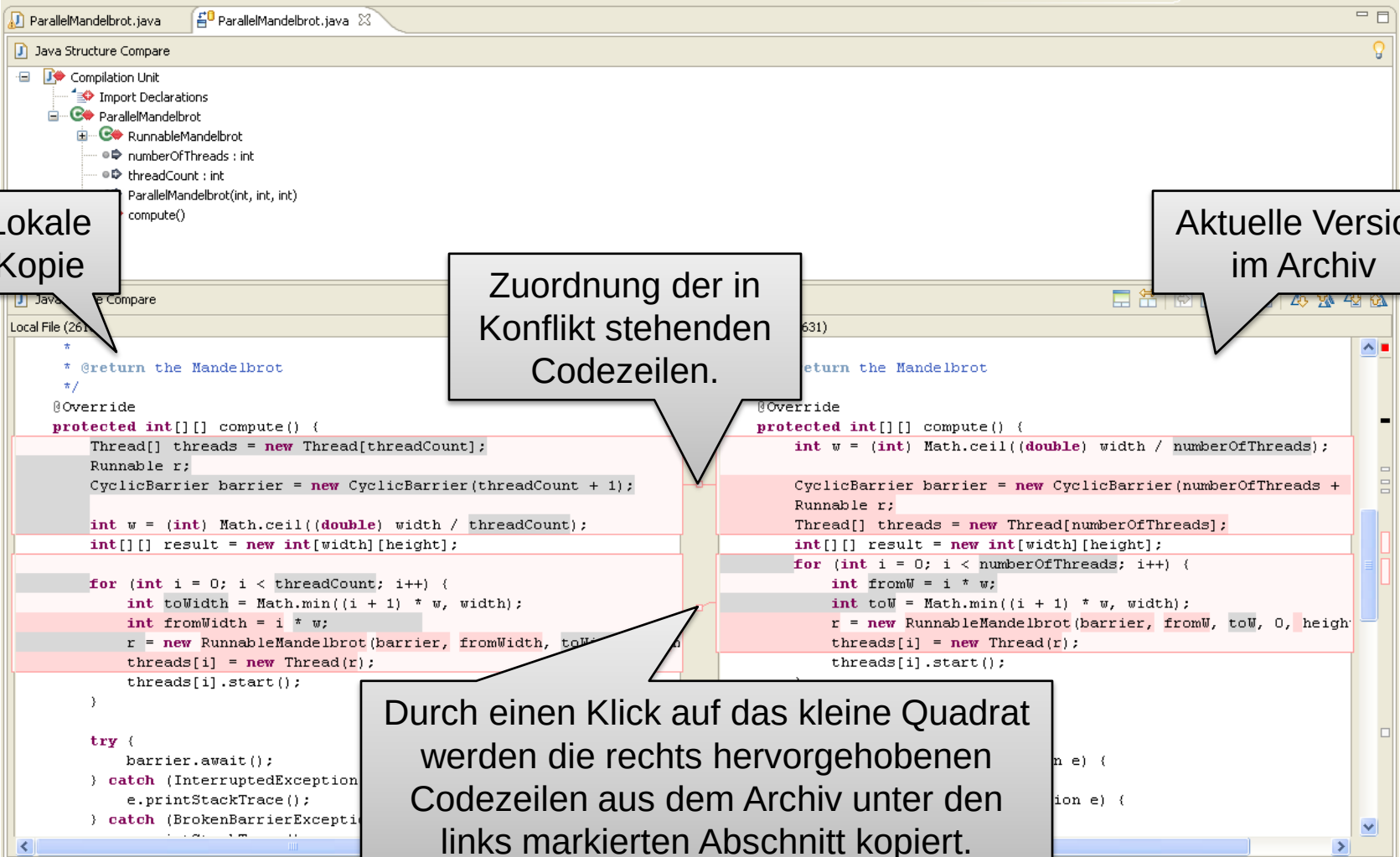
Einbuchen I

- `svn commit <Verzeichnis/Dateien>`
 - Schreibt **<Verzeichnis/Dateien>** in Archiv zurück
 - Bei Konflikten, bricht SVN ab → benutze **update** zuerst
 - Erhöht Revisionsnummer der veränderten Dateien.
 - Beim Einbuchen werden nur Änderungen der geänderten Dateien übertragen:
 - Im **.svn/** Verzeichnis befindet sich eine Kopie der letzten Version aller Dateien.
 - Zum Einbuchen werden die Änderungen zwischen diesen Kopien und den geänderten Dateien gesucht.
 - Diese Änderungen werden dann zum SVN-Server übertragen.

Einbuchen II

- Hat sich seit dem letzten Aktualisieren der lokalen Kopie die entsprechende Datei im Archiv geändert, muss vor dem Einbuchen die **neue Version** im Archiv mit der lokalen Kopie verschmolzen werden.
- Dazu bieten die graphischen Schnittstellen Werkzeugunterstützung an.

Einbuchen III – Zusammenführen mit Subclipse



Lokale Kopie

Aktuelle Version im Archiv

Zuordnung der in Konflikt stehenden Codezeilen.

Durch einen Klick auf das kleine Quadrat werden die rechts hervorgehobenen Codezeilen aus dem Archiv unter den links markierten Abschnitt kopiert.

```
Java Structure Compare
- Compilation Unit
  - Import Declarations
  - ParallelMandelbrot
    - RunnableMandelbrot
      - numberOfThreads : int
      - threadCount : int
      - ParallelMandelbrot(int, int, int)
      - compute()

Local File (261...)
* @return the Mandelbrot
*/
@Override
protected int[][] compute() {
    Thread[] threads = new Thread[threadCount];
    Runnable r;
    CyclicBarrier barrier = new CyclicBarrier(threadCount + 1);

    int w = (int) Math.ceil((double) width / threadCount);
    int[][] result = new int[width][height];

    for (int i = 0; i < threadCount; i++) {
        int toWidth = Math.min((i + 1) * w, width);
        int fromWidth = i * w;
        r = new RunnableMandelbrot(barrier, fromWidth, toWidth, 0, height);
        threads[i] = new Thread(r);
        threads[i].start();
    }

    try {
        barrier.await();
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (BrokenBarrierException e) {
        e.printStackTrace();
    }

    return result;
}

Archive
@Override
protected int[][] compute() {
    int w = (int) Math.ceil((double) width / numberOfThreads);
    CyclicBarrier barrier = new CyclicBarrier(numberOfThreads + 1);
    Runnable r;
    Thread[] threads = new Thread[numberOfThreads];
    int[][] result = new int[width][height];
    for (int i = 0; i < numberOfThreads; i++) {
        int fromW = i * w;
        int toW = Math.min((i + 1) * w, width);
        r = new RunnableMandelbrot(barrier, fromW, toW, 0, height);
        threads[i] = new Thread(r);
        threads[i].start();
    }
}
```

Beispiel: SWT I Folien

■ Mitarbeiter:

- Archiv ausbuchen (falls noch nicht geschehen):
`svn checkout`
`https://svn.ipd.uni-karlsruhe.de/lehre/vorlesung/SWT1/SS09/studies/ \`
`D:\SWT1Folien --username swt1 --password ***`
- Datei hinzufügen
`svn add .\KapXYZ.pdf`
- Änderungen einbuchen:
`svn commit -m „Neues Kapitel ‚XYZ‘ veröffentlicht.“ \`
`--username swt1 --password ***`

■ Student:

- Lokale Kopie aktualisieren
`svn update`

Weitere Kommandos / Optionen

- `diff` – Finde Differenzen zwischen lokaler Kopie und Archiv
- Zu jedem Kommando gibt es eine Reihe von Optionen, die z.B. die genaue Spezifikation von Versionen erlauben.

Graphische Schnittstellen

- TortoiseSVN: <http://tortoisesvn.tigris.org>
- RapidSVN: <http://rapidsvn.tigris.org>
- Subclipse: <http://subclipse.tigris.org>

SVN Details

- Administrative Informationen befinden sich lokal im Unterverzeichnis **.svn/** (ähnlich wie bei cvs in **CVS/**)
- Von jeder Datei befindet sich eine unveränderte Version im **.svn**-Verzeichnis. Dadurch können einige Befehle (wie bspw. **revert**) lokal ausgeführt werden.
- Das Einpflegen (**commit**) geschieht bei SVN atomar, d.h. entweder werden alle Änderungen in das Archiv übernommen, oder gar keine.
- SVN unterstützt binäre Dateien. Es wird beim Einpflegen von Dateien automatisch erkannt, ob es sich um eine Text- oder Binärdatei handelt.
- Zur Zeichenkodierung verwendet SVN UTF-8, wodurch internationale Zeichen unterstützt werden.

Literatur

- Bruegge, B., Dutoit, A., *Object-Oriented Software Engineering: Using UML, Patterns and Java*, 2004, Pearson Prentice Hall, Kap. 12
- Manual Seiten zu RCS
 - Handbuch: <http://agave.garden.org/~aaronh/rcs/tichy1985rcs/rcs.html>
 - Projektseite: <http://www.gnu.org/software/rcs/rcs.html>
- SVN:
 - Ausführliches Handbuch: <http://svnbook.red-bean.com>
 - Projektseite: <http://subversion.tigris.org>
- GIT, verteilte Versionsverwaltung:
 - Projektseite: <http://git-scm.com>