Programmiermethodik 1
Programmiertechnik

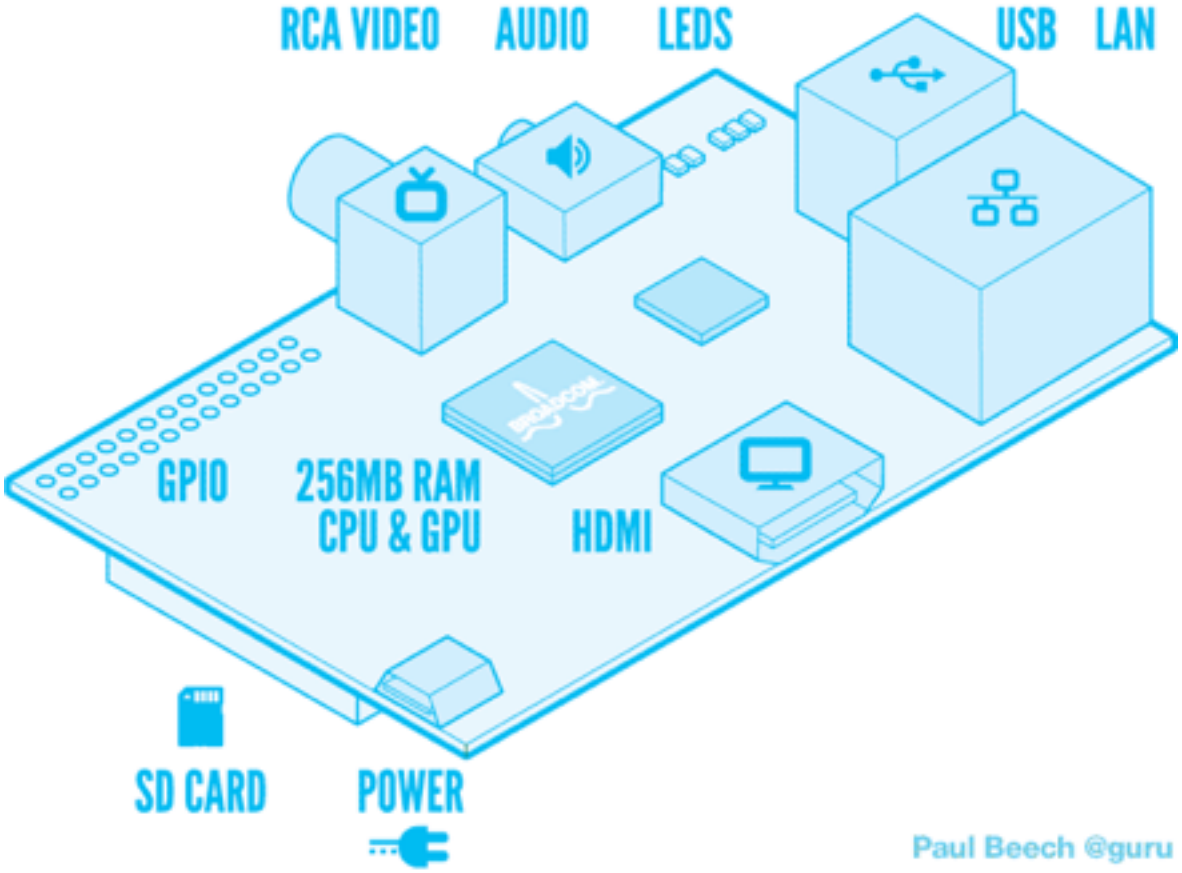
Collections, Git, Demo

Wiederholung

- Collections-Framework
- Verkettete Liste und Array-Liste
- Iteratoren
- Vergleichen

Ausblick

Worum gehts?


RCA VIDEO
AUDIO
LEDS
USB
LAN
GPIO
256MB RAM
CPU & GPU
HDMI
SD CARD
POWER
Paul Beech @guru


Java 8

Agenda

- Collections
    - Menge
    - Map
    - Collections-Operationen
- Git
- Demo

Menge

Motivation

- Erinnerung:
  - Container aus Interface List können das gleiche Element mehrfach beinhalten (mit unterschiedlichem Index)
- manchmal gewünscht:
  - jedes Element nur einmal in Container
  - mathematische Bezeichnung: Menge
  - Umsetzung in Java: Interface Set

Interface Set

- ebenfalls vom Interface Collection abgeleitet
    - wie das Interface List
- Erweiterung der Collection-Schnittstelle
    - zusätzliche Anforderung an die implementierenden Klassen
    - eine Duplikate zulassen
    - zu keinem Zeitpunkt darf es zwei Element-Objekte x und y geben, für die x.equals(y) == true gilt!
- Einträge sind nicht geordnet

Achtung Aufpassen

- Achtung
  - sollten Element-Objekte "von außen" so verändert werden, dass x.equals(y) eintritt, gerät ein Set in einen undefinierten Zustand!

Referenzimplementierung: HashSet

- Klasse HashSet implementiert des Interface Set
- wieder: Verwendung von generischen Typen
    - HashSet<Elementtyp>

Beispiel: Anzahl verschiedener Wörter in Text

```
public class WoerterZaehlen {

  public int zaehleWoerter(String text) {
    Set<String> wordSet = new HashSet<String>();
    for (String word : text.split(" ")) {
      wordSet.add(word);
    }
    return wordSet.size();
  }

  public static void main(String[] args) {
    WoerterZaehlen woerterZaehlen = new WoerterZaehlen();
    String text =
        "Wenn Fliegen fliegen fliegen Fliegen Fliegen nach.";
    System.out.format("Text: '%s'.\n", text);
    System.out.format("Anzahl Wörter in dem Text: %d.\n",
        woerterZaehlen.zaehleWoerter(text));
  }
}
```

Map

Motivation

- Informationen lassen sich häufig so darstellen:
  - Menge von Schlüsseln (z.B. String oder Zahl)
  - zu jedem Schlüssel ein Wert (kann auch Objekt sein)
- Liste und Menge für diese Anforderung noch optimal geeignet
- besser
  - Wörterbuch oder Map/Abbildung
  - Java: Interface Map
- Hinweis: falls Schlüssel = aufsteigender Zahlenwert → Array oder Liste auch geeignet
  - eindeutige Abbildung: Index → Element
  - Beispiele

```
myArray[0]
myList.get(5);
```

Interface Map

- Map-Idee: Verallgemeinerung des Indextyps
  - Index = Schlüssel ("key")
  - Element = Wert ("value")
  - Schlüssel und Wert können jeweils beliebigen Datentyp haben
  - eindeutige Abbildung: Schlüssel auf Wert
  - Speicherung von Paaren (Schlüssel, Wert)
- Beispiel Telefonliste:
  - Name: Schlüssel
  - Nummer: Wert

Referenzimplementierung HashMap

- Einträge sind nicht geordnet
- Schlüssel und Werte müssen Referenztypen sein
- wieder: Verwendung von generischen Typen:

  `HashMap<KeyType,ValueType>`

Wichtige Methoden

```
public ValueType put(KeyType key, ValueType value)
```

- speichert den Wert value unter dem Schüssel key in der HashMap
- Rückgabe: ein evtl. vorhandener alter Wert oder null

```
public ValueType get(Object key)
```

- liefert den unter dem Schlüssel key gespeicherten Wert
- oder null, falls der Schlüssel in der HashMap nicht vorkommt

```
public ValueType remove(Object key)
```

- löscht den unter dem Schlüssel key gespeicherten Eintrag (Schlüssel, Wert) und liefert den Wert
- falls der Schlüssel key in der HashMap nicht vorhanden ist, passiert nichts und null wird zurückgegeben

Beispiel: Telefonliste mit Map

```
public class TelefonlisteMap {

   private Map<String, String> eintraege =
      new HashMap<String, String>();

   public void eintragHinzufuegen(String name, String nummer) {
      eintraege.put(name, nummer);
   }

   public String getNummer(String name) {
      return eintraege.get(name);
   }

   public void eintragEntfernen(String name) {
      eintraege.remove(name);
   }
}
```

Gleichheit

- alle Referenztypen können als Schlüssel (key) verwendet werden
- alle Collections-Klassen verwenden die equals()-Methode eines Objekts zur Feststellung der Gleichheit
    - eine geeignete equals()-Implementierung für den Typ des Schlüssels muss vorhanden sein!
    - Default der Klasse Object: Test auf Identität der Objekte
    - Telefonbuch: equals()-Methode der Klasse String wird verwendet
    - also keine Redefinition nötig
- viele Collection-Klassen verwenden auch die hashCode()-Methode zur Optimierung
    - Vorsicht bei Redefinition

Umwandlung HashMap

```
public Set<KeyTyp> keySet()
```

- liefert die Menge aller Schlüssel einer Map als Set (generischer Typ!)

```
public Collection<ValueTyp> values()
```

- liefert die Werte einer Map als allgemeine Collection

```
public Set<Map.Entry<KeyTyp,ValueTyp>> entrySet()
```

- liefert alle Einträge der Map als Menge mit Elementtyp Map.Entry<KeyTyp,ValueTyp>
- Klasse Map.Entry definiert die beiden Methoden
- KeyTyp getKey(): liefert den Schlüssel des Eintrags
- ValueTyp getValue(): liefert den Wert des Eintrags

Interface Map

- Umwandlungen in andere Container liefern nur eine Sicht auf die Map, d.h.
  - Änderungen an der Collection wirken sich auf die Map aus
  - sehr effiziente Erzeugung, weil keine Daten kopiert werden
- Beispiele für Collection-Sichten auf eine Map
- Namen und Nummern jeweils getrennt sammeln:

```
Set<String> namen = eintraegeMap.keySet();
Collection<String> nummern = eintraegeMap.values();
```

- Das gesamte Telefonbuch ausgeben:

```
for (Map.Entry<String, String> eintrag:
   eintraegeMap.entrySet()){
   System.out.format("%s: %s\n", eintrag.getKey(),
      eintrag.getValue());
}
```

Übung: Menge und Map

- Schreiben Sie eine Klasse SchweizerNummernKontoBank
- eine solche Bank verwaltet Konten
- ein Konto besteht aus einer Kontonummer (int) und einem Kontostand (float)
- Schreiben Sie Methoden
    - zum Hinzufügen eines Kontos
    - zum Setzen des Kontostands eines Kontos
    - zum Auslesen des Kontostands eines Kontos

Collections-Operationen

Collections

- Sammlung von Algorithmen als statische Methoden der Klasse java.util.Collections (ähnlich wie Klasse Arrays)
- Beispielauswahl:

```
static void sort(List<Elementtyp> list)
```

- Liste list nach aufsteigender Elementgröße sortieren
- Elementtyp muss das Interface Comparable implementieren

```
static Elementtyp max(Collection<Elementtyp> coll)
```

- liefert das größte Element der Collection
- Elementtyp muss das Interface Comparable implementieren

```
static int binarySearch(List<Elementtyp> list, <Elementtyp> key)
```

- key in der sortierten Liste list suchen (Vergleiche: Comparable)
- Ergebnis
    - >= 0 Index der ersten Fundstelle von key in list
    - < 0 key in list nicht gefunden

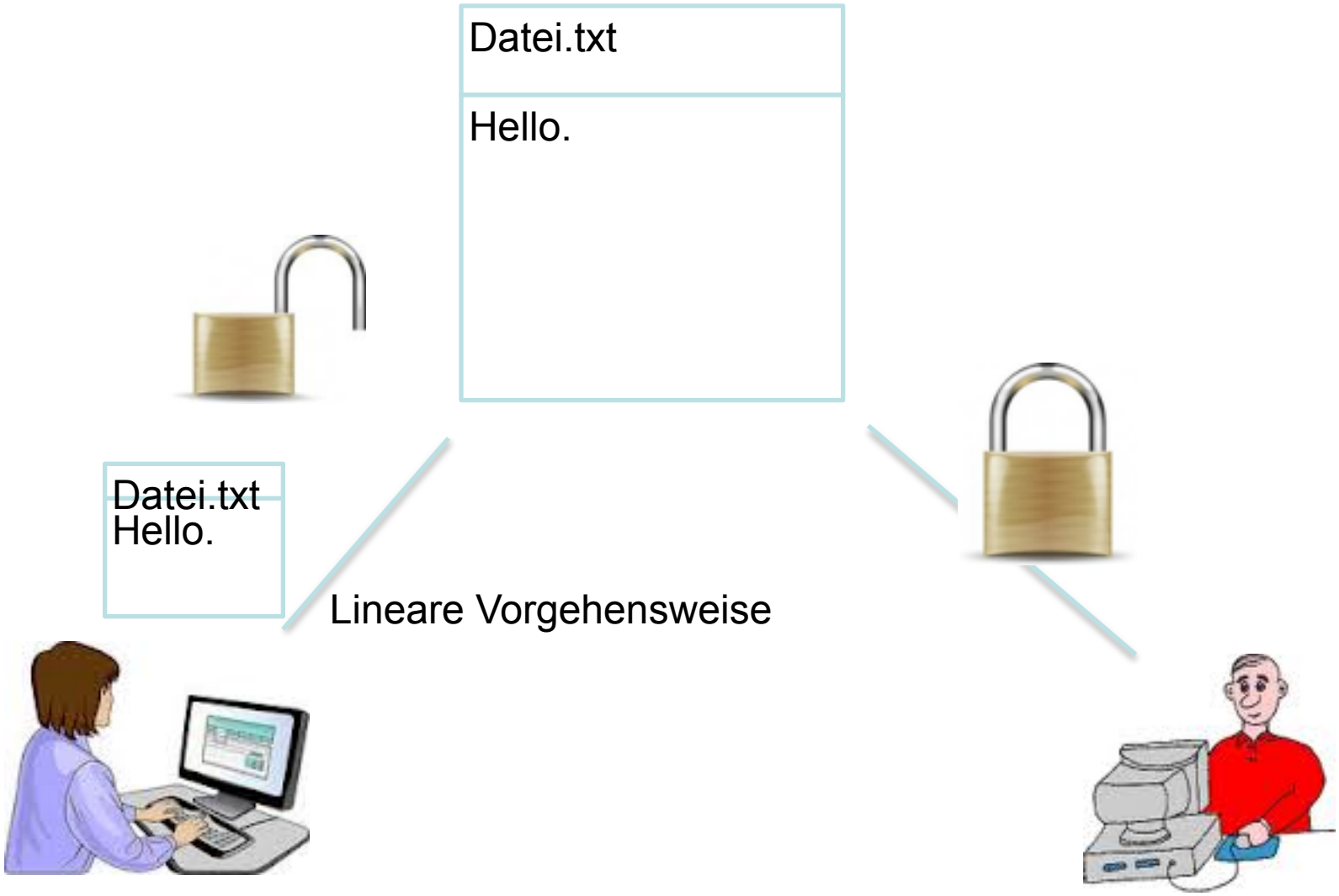
Versionsverwaltung

Versionsverwaltung

Datei.txt
Hello.

Beide wollen eine
Datei bearbeiten

Versionsverwaltung



Versionsverwaltung

Datei.txt

„Hello. Worlde.“

Editieren und auf den Server zurückschreiben (commit)

Versionsverwaltung

Datei.txt

„Hello. World!“

Lokale Kopie holen,
editieren, committen.

Versionsverwaltung

Datei.txt

Hello.

Datei.txt
Hello.

Datei.txt
Hello.

Nicht-linearer Ansatz:
Beide holen sich eine
Working-Version.

Versionsverwaltung

Datei.txt
Hello.

Datei.txt
„Hello.
Worlde“

Beide editieren gleichzeitig.

Datei.txt
Hello
World

Versionsverwaltung

Datei.txt

„Hello. Worlde“

Datei.txt
„Hello.
Worlde“

commit

Datei.txt
Hello
World

Versionsverwaltung

Datei.txt

„Hello. Worlde“

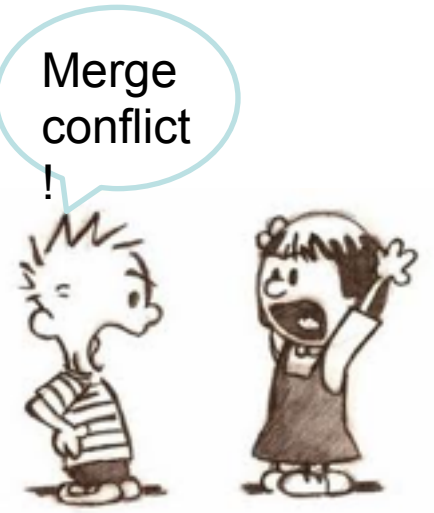
commit

Datei.txt
Hello
World

Versionsverwaltung

Datei.txt

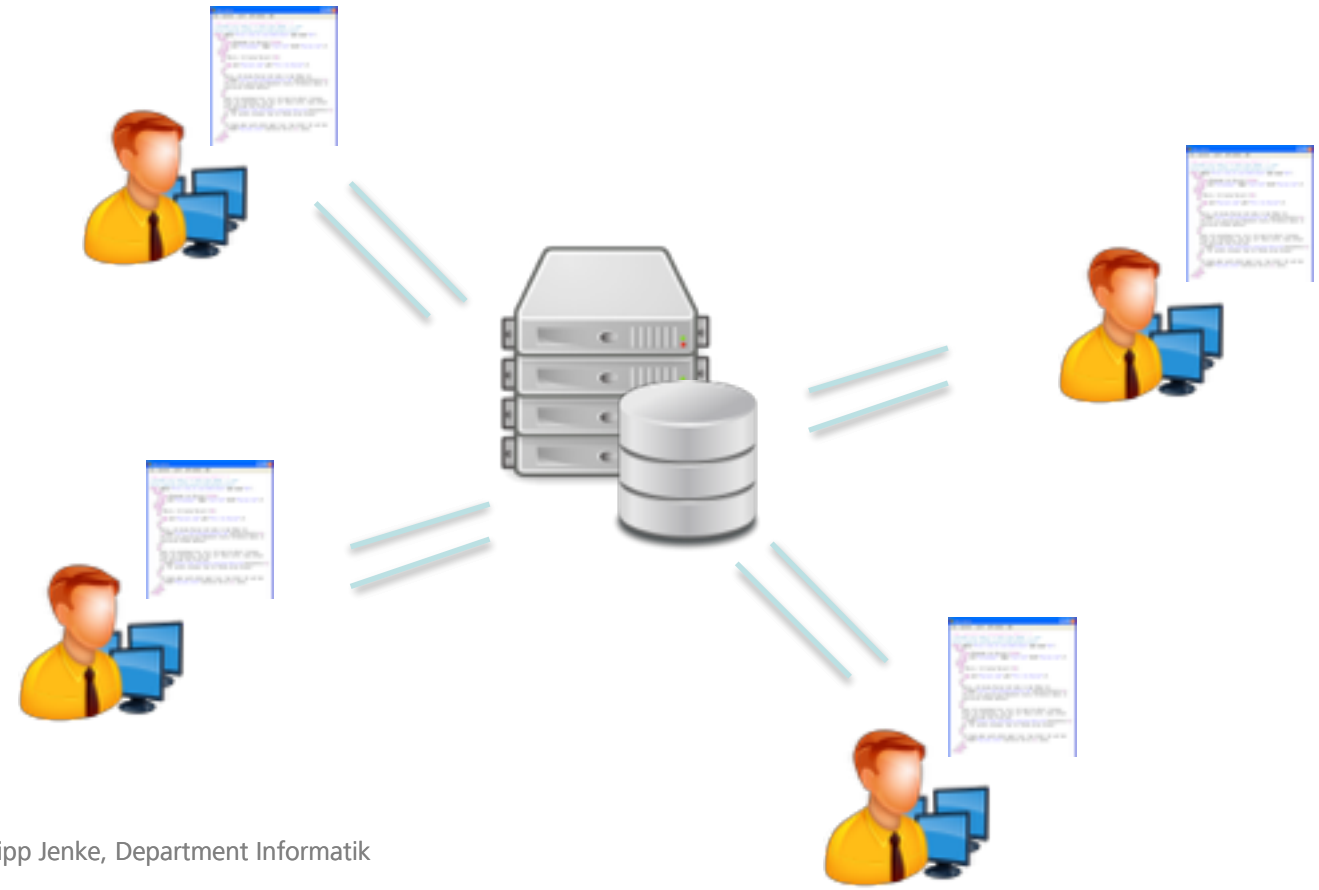
„Hello. Worlde“



Datei.txt
Hello
World

Versionsverwaltung

- Zentraler (Server-)Ansatz
- Server synchronisiert die lokalen Kopien aller Entwickler



Versionsverwaltung

- mittlerweile üblich
  - dezentrale Versionsverwaltung
  - z.B. Mercurial, GIT

Entwickler, Entwicklungsumgebung, Arbeitskopie

lokales Repository des Entwicklers (Bildquelle: [3])

allgemein zugängliches Repository auf Server

Git

GIT

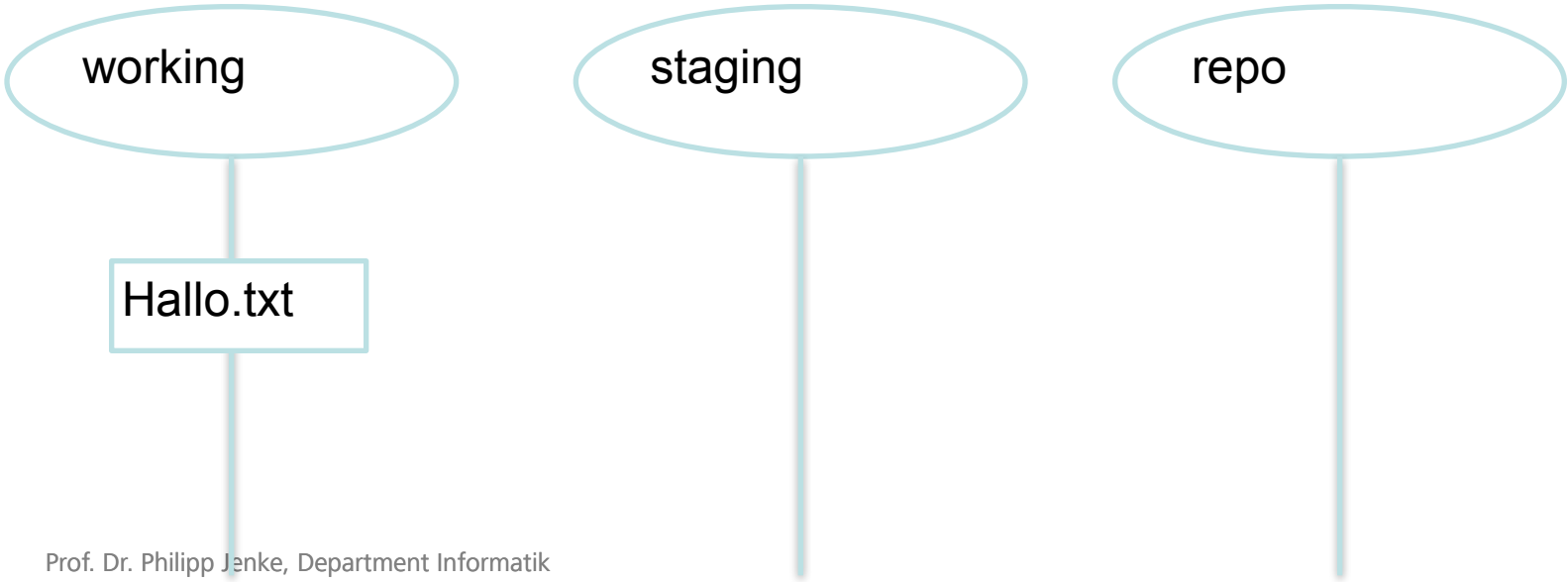
```
$ git config --global user.name "Your Name“
$ git config --global user.email "your_email@whatever.com"
```

Name und E-Mail-Adresse angeben

GIT

```
$ git init
```

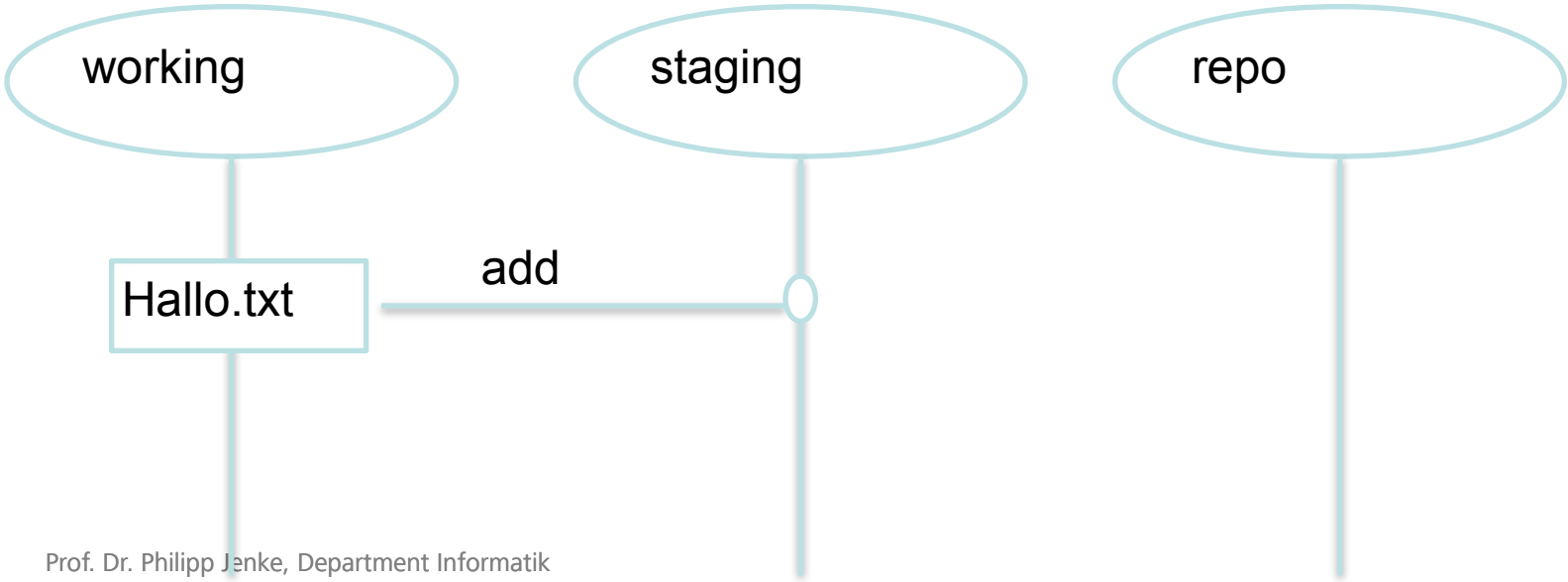
Lokales Git-Repository erstellen



GIT

```
$ git add Hallo.txt
```

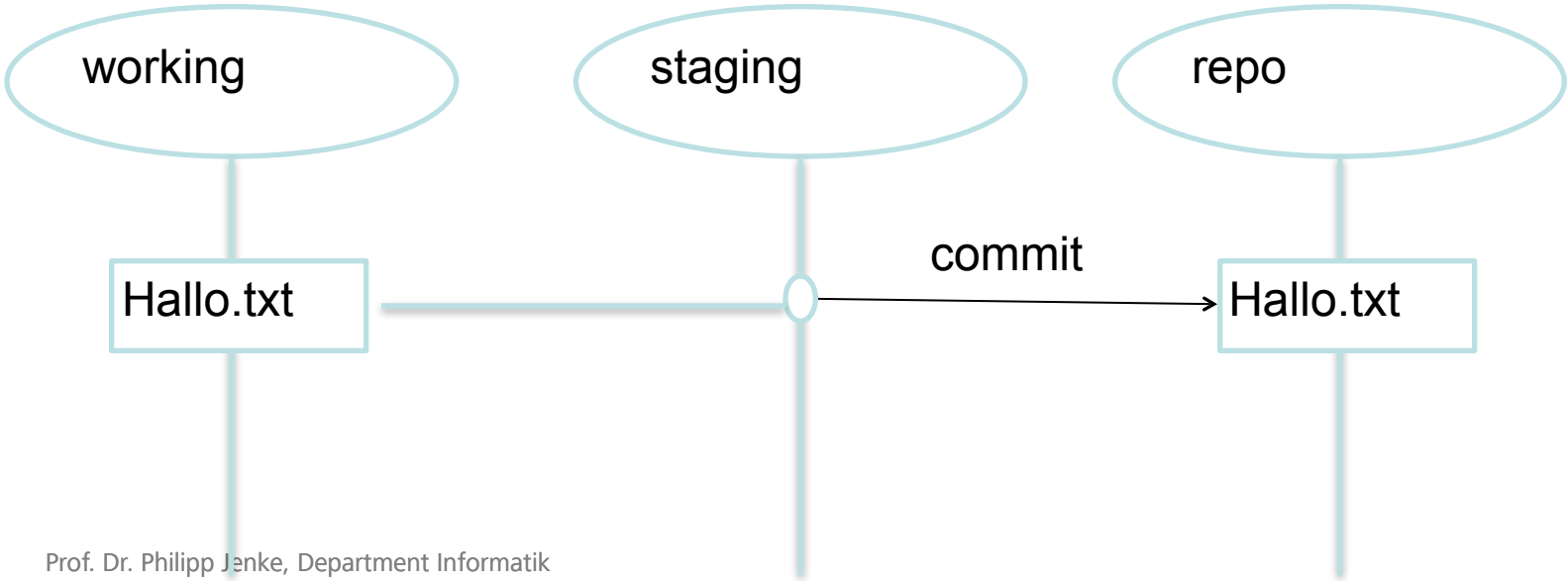
Zum Index hinzufügen (stagen)



GIT

```
$ git commit –m “first Hallo“
```

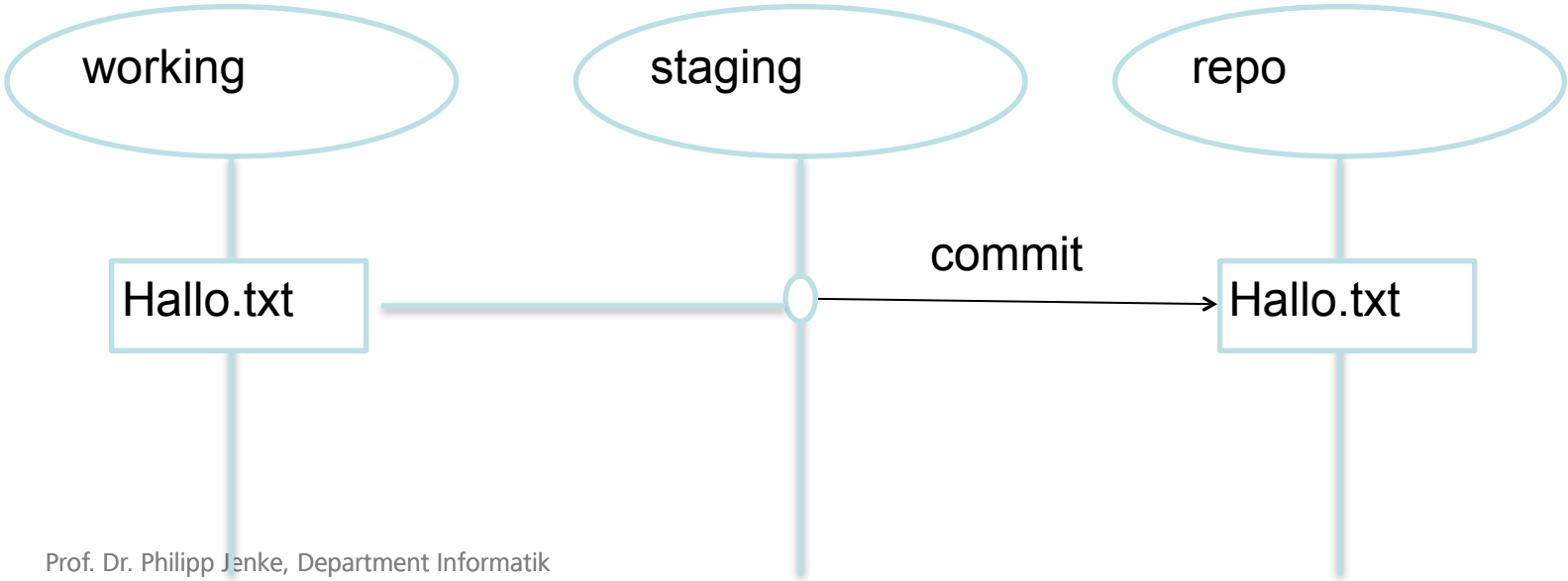
Auf das lokale Repo schreiben



GIT

```
$ git status
# on branch master
Nothing to commit
```

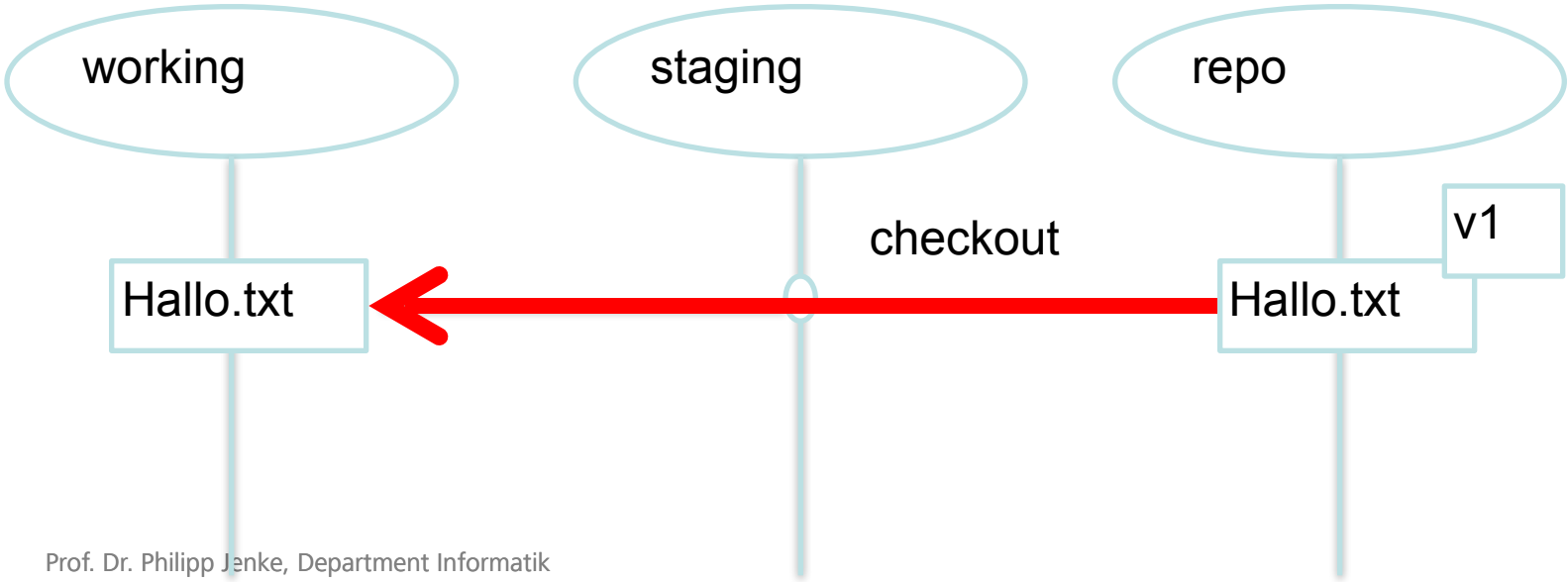
Status des Staging-Bereichs angeben



GIT

```
$ edit Hallo.txt
$ git checkout master .
```

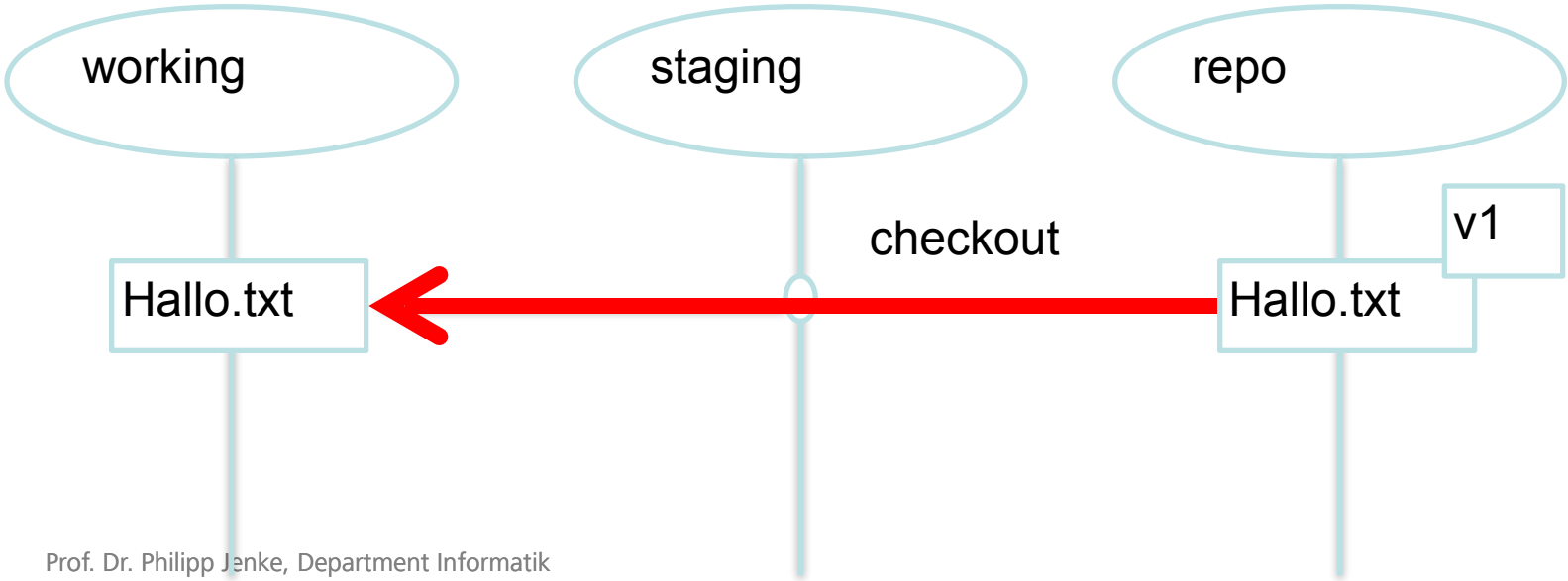
Hallo.txt verändern und Veränderung rückgäng machen



GIT

```
$ edit Hallo.txt
$ git checkout master
$ git checkout “Hallo.txt“
```

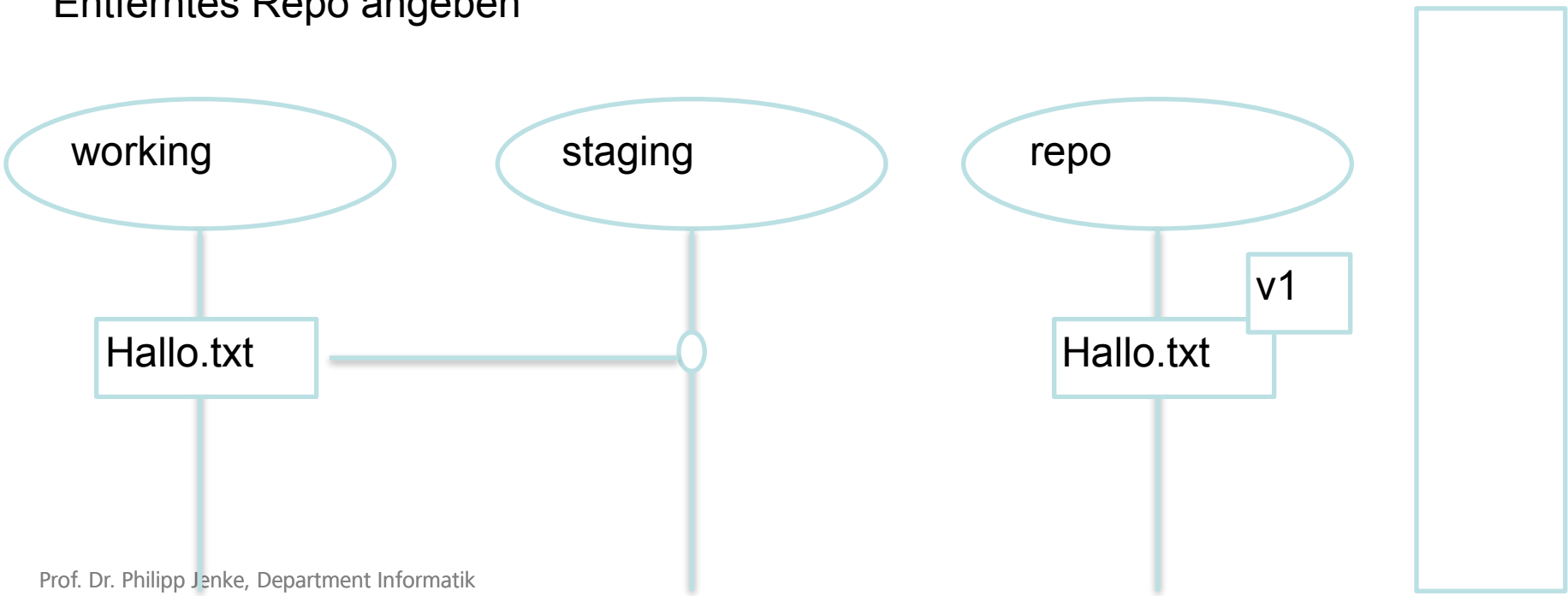
Hallo.txt verändern und Veränderung rückgäng machen



GIT

```
$ git remote add origin https://github.com
```

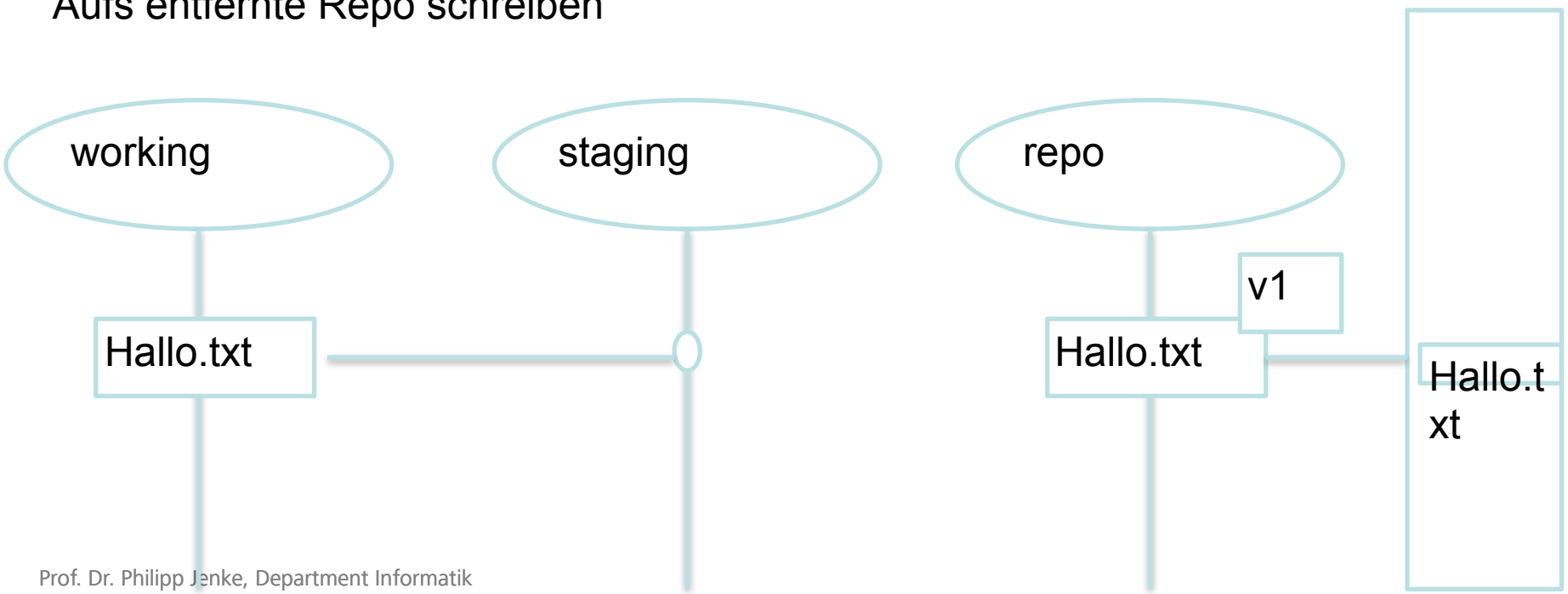
Entferntes Repo angeben



GIT

```
$ git push –u origin master
```

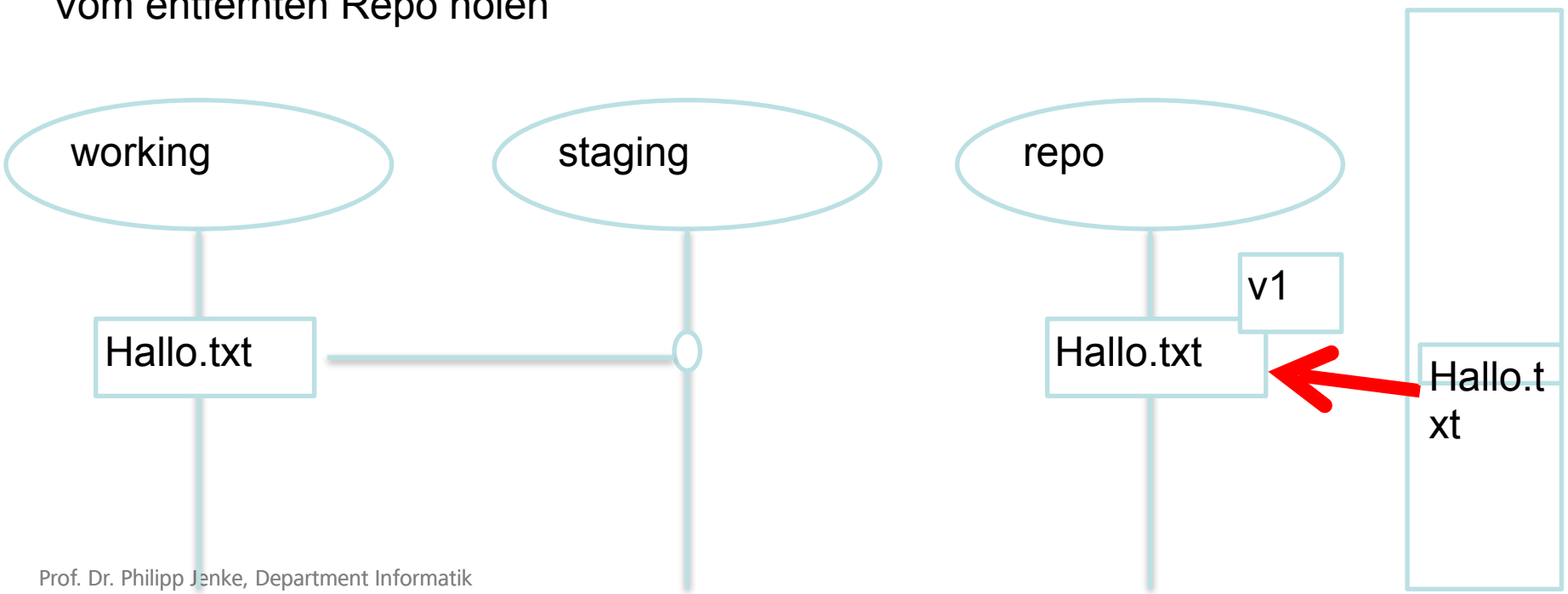
Aufs entfernte Repo schreiben



GIT

```
$ git pull origin master
```

Vom entfernten Repo holen

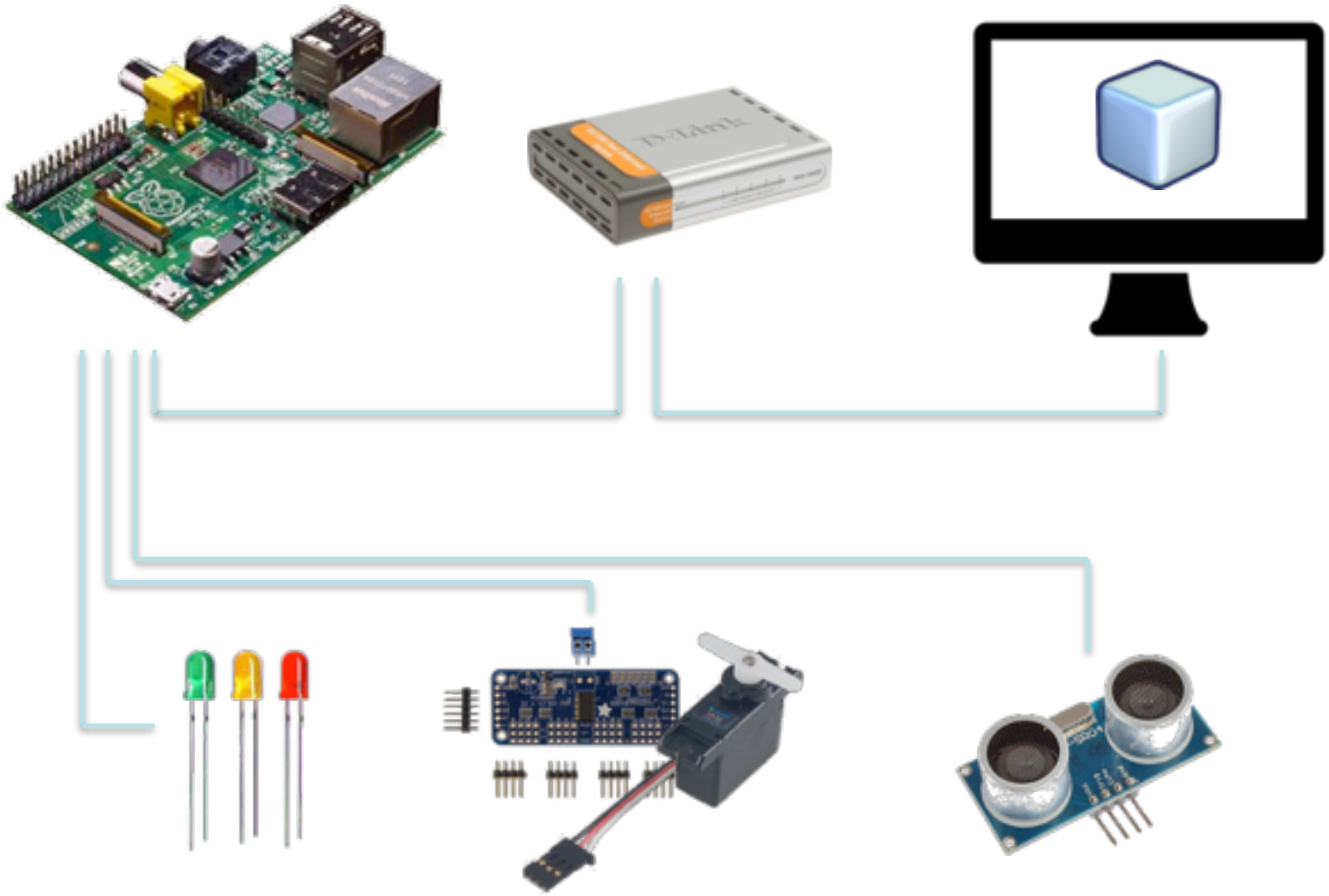


GIT-Repositories

- Eigenes Netzlaufwerk
- Bitbucket (https://bitbucket.org/): Kostenlos für private Nutzung
- Github (https://github.com/): Kostenlos bei öffentlichen Repositories
- Sourceforge (http://sourceforge.net/): Open Source Projekte
- Angebot der HAW Informatik

Live Demo: Raspberry Pi

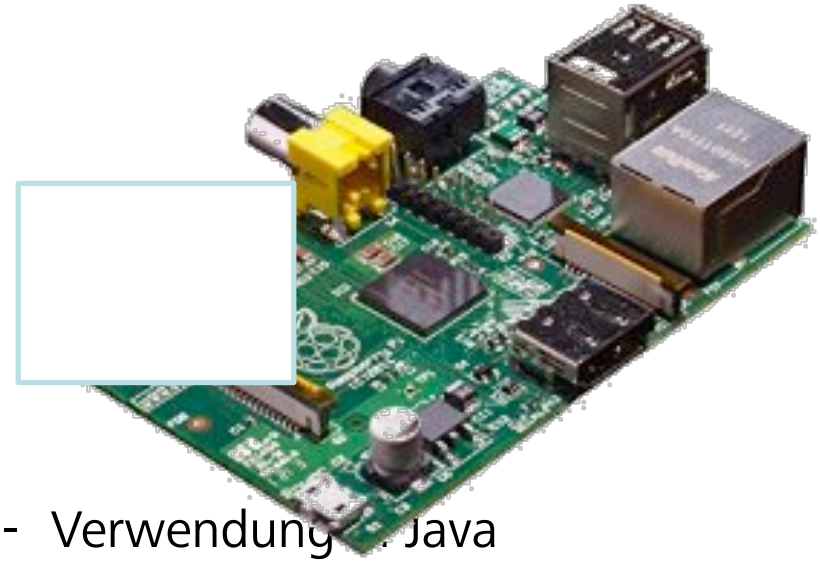
Aufbau



Raspberry Pi

GPIO (General Purpose I/O)

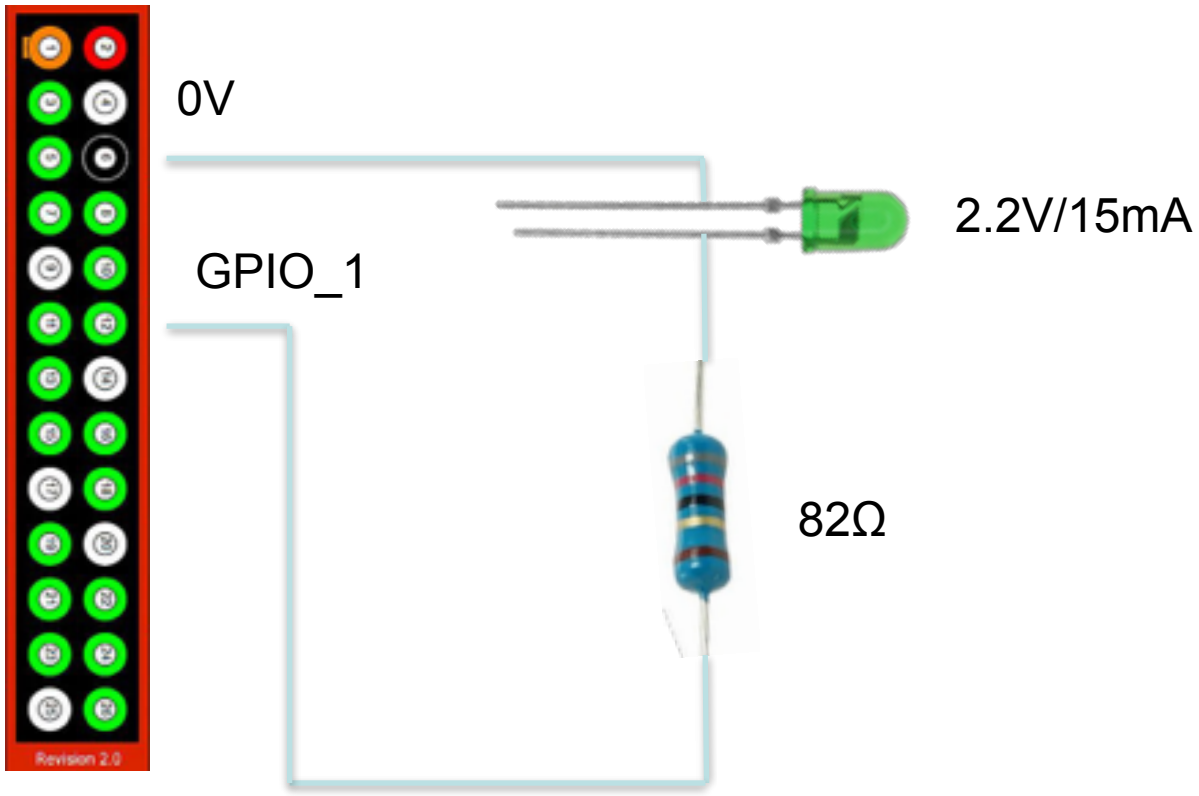
- Bibliothek zur Ansteuerung
  - wiringPi (http://wiringpi.com/)



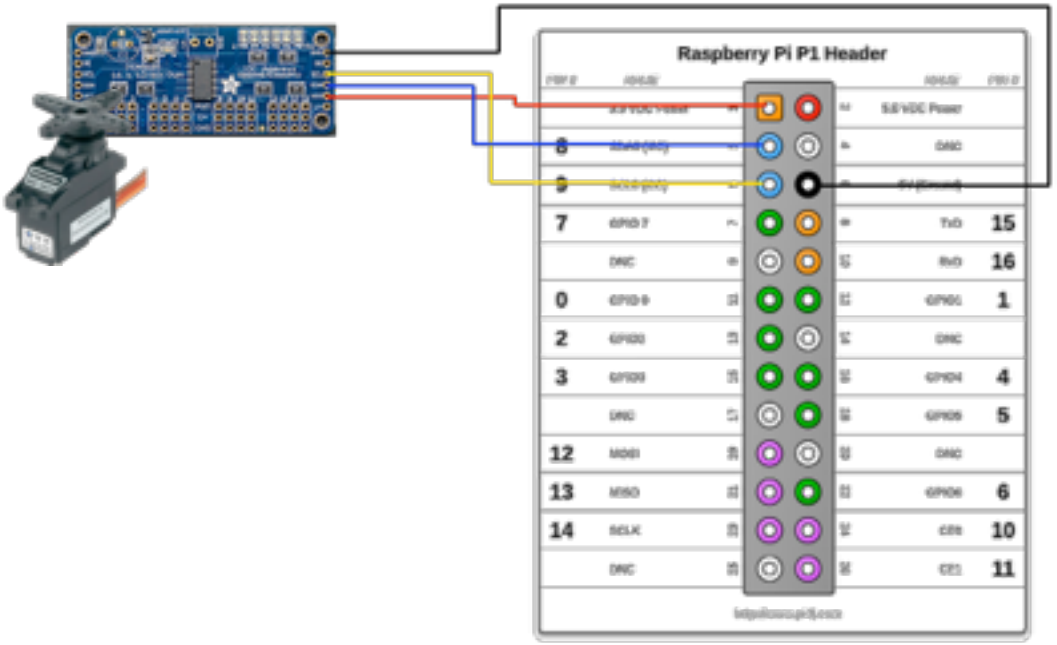
P1: The Main GPIO connector							
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin
		3.3v	1	2	5v		
8	Rv1:0 - Rv2:2	SDA	3	4	5v		
9	Rv1:1 - Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3.3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin

- Verwendung in Java
  - Wrapper: Pi4J (http://pi4j.com/)

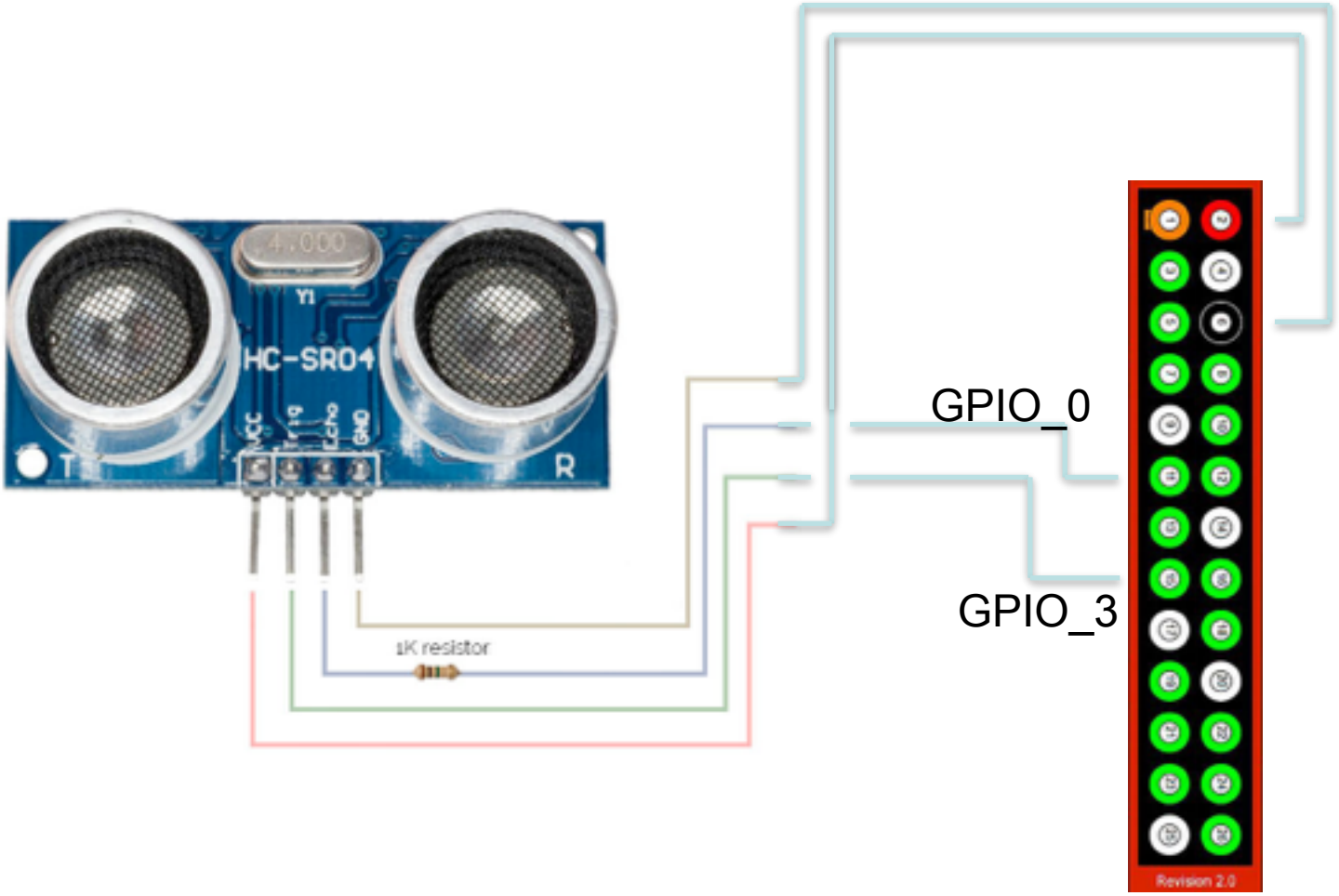
Projekt 1: LED



Projekt 2: Servo



Projekt 3: Ultraschallsensor



Zusammenfassung

- Collections
    - Menge
    - Map
    - Collections-Operationen
- Git
- Demo