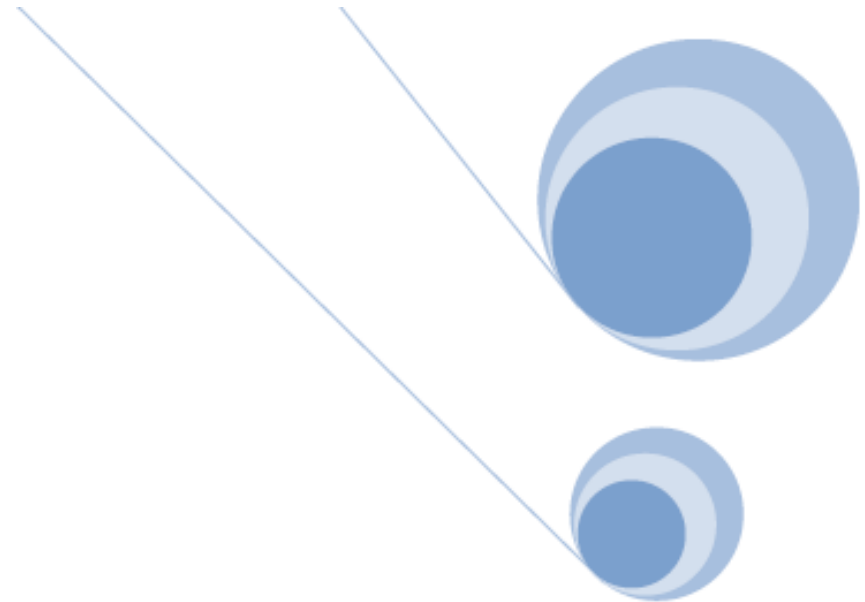


# Batch Frameworks für Java



**Spring Batch - Reference Documentation**



**Batch Applications for  
the Java Platform**

- Die in diesem Seminar verwendete Werkzeuge und Frameworks sind Open Source
  - LPGL Lizenzmodell
- Dies ist ein Programmier-Seminar
  - Damit werden die Inhalte durch Übungen vertieft und verinnerlicht
  - Musterbeispiele werden zur Verfügung gestellt
  - Diese können am Ende des Seminars als ZIP-Datei kopiert werden
    - USB-Stick oder ähnliches
- Dokumentation und Ressourcen stehen auch im Internet zur Verfügung
- Konventionen
  - Befehle werden in `Courier-Schriftart` dargestellt
  - Dateinamen werden in *`Courier-Schriftart`* dargestellt
  - Links werden in `unterstrichener Courier-Schriftart` dargestellt

© Javacream

Javacream

Dr. Rainer Sawitzki

Alois-Gilg-Weg 6

81373 München

**Alle Rechte, einschließlich derjenigen des auszugsweisen Abdrucks, der fotomechanischen und elektronischen Wiedergabe vorbehalten.**

Java Batch	6
Batch Spezifikation	24
Programmierung	32
Beispiele	61

1

## **JAVA BATCH**

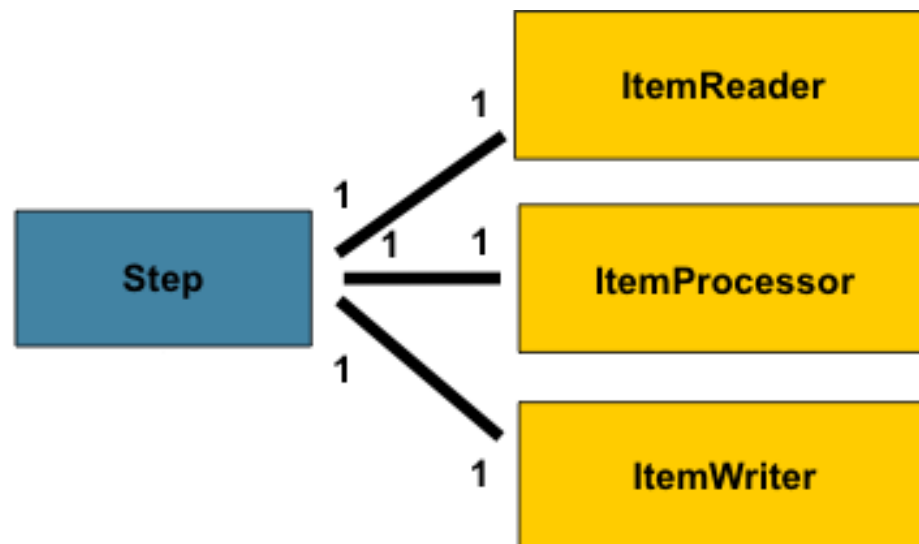
## 1.1

# MERKMALE EINES BATCH-PROZESSES

- Kein Benutzer-Interaktion notwendig
  - Scheduled Ausführung
  - Event- oder Message-gesteuert
- Massendatenverarbeitung
- Typischerweise Parallelisierbar
  - Multithreading
  - Multi-Processing
  - Partitionierung der Eingangsdaten
- Automatisierte Fehlerbehandlung
  - Retry-Mechanismen
- Ausführliche Logging- und Monitoring-Werkzeuge
  - Laufende Jobs
  - Erfolgreiche und abgebrochene Läufe
  - Nachvollziehbare Fehlermeldungen



- Durch das Fehlen einer Benutzer-Interaktion lassen sich Batches extrem gut abstrahieren
  - Ein Batch-Lauf führt einzelne Steps aus
  - Step-Sequenz
    - Ein `ItemReader` liefert Eingabedaten
    - Diese werden von einem `ItemProcessor` verarbeitet
    - Die Ergebnisse werden von einem `ItemWriter` geschrieben

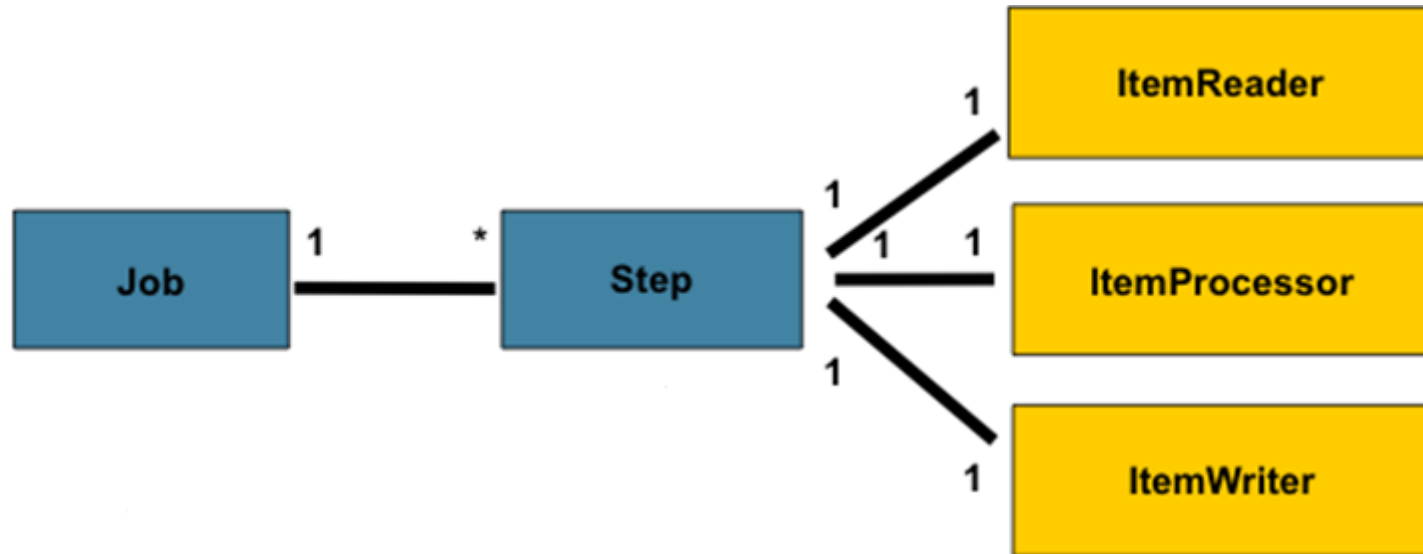


- Conversion
  - Umwandlung der Eingabe-Daten in ein normalisiertes Format
- Validation
- Extract
  - Selektion aus einer Datenquelle in eine Ausgabe schreiben
- Extract/Update
- Processing and Upating
- Output/Format
- Basic Application Shell für Business Logic

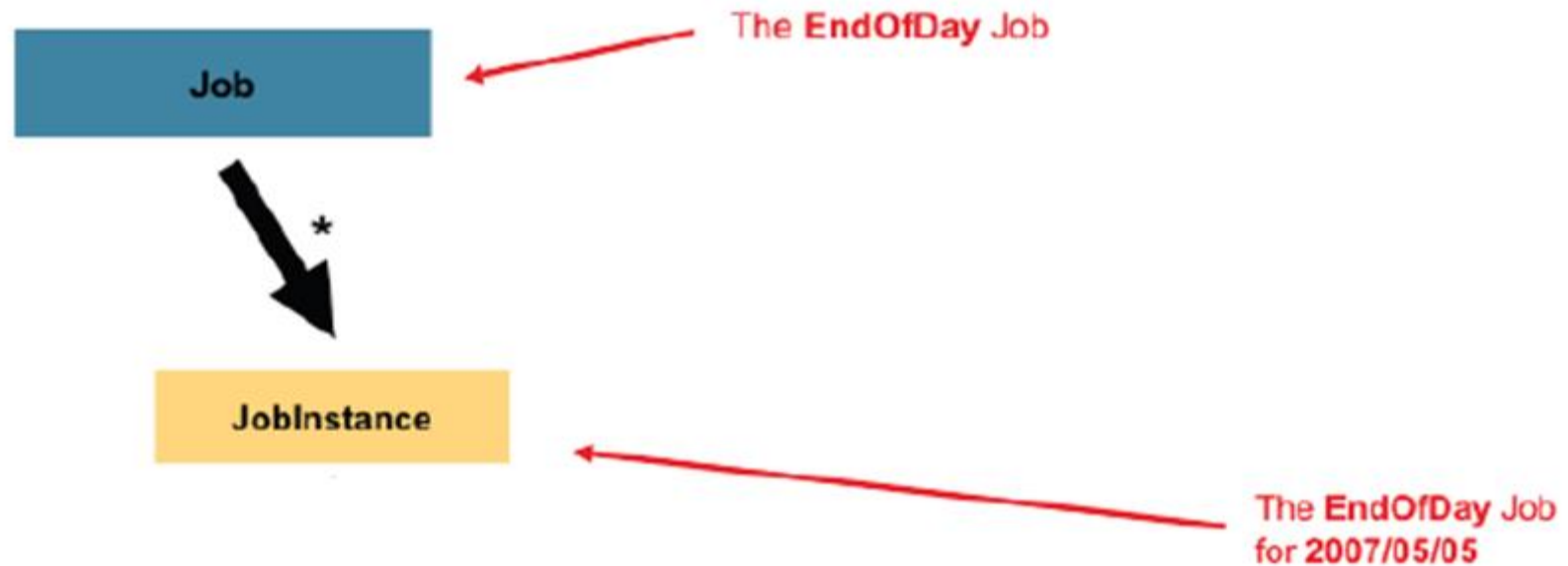
- Normal Processing, Single Process
  - Single commit am Ende des Laufs
- Konkurrierende Batch-Läufe
  - Locking-Mechanismen notwendig (optimistic, pessimistic)
- Parallel Processing
- Partitionierung
  - Parallelisierung auf partitionierten Datenbeständen

## 1.2 **JOBS**

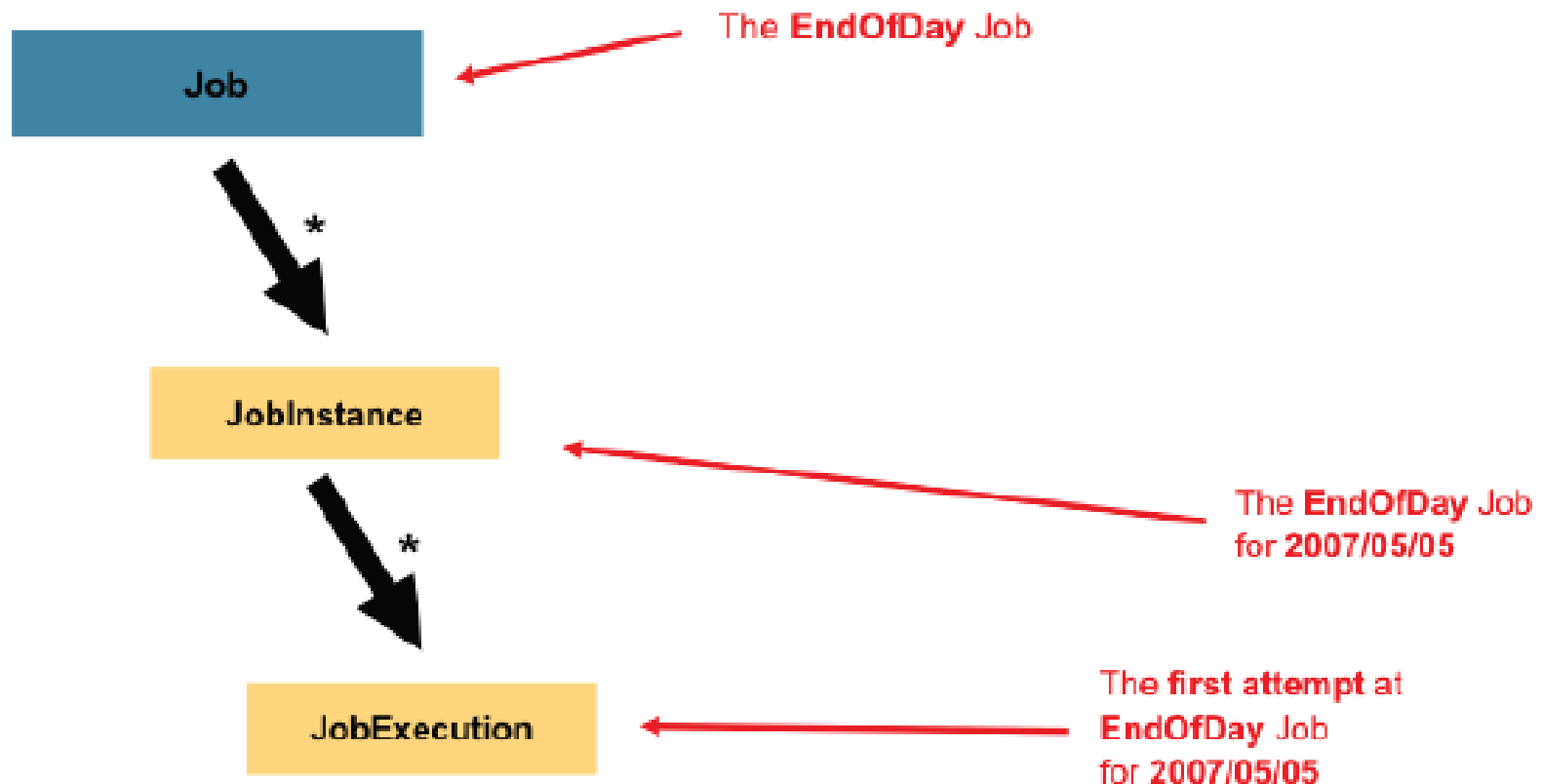
- Ein Job besteht aus mehreren aufeinanderfolgenden Steps



- Ein Job kann wie eine Klasse instanziiert werden



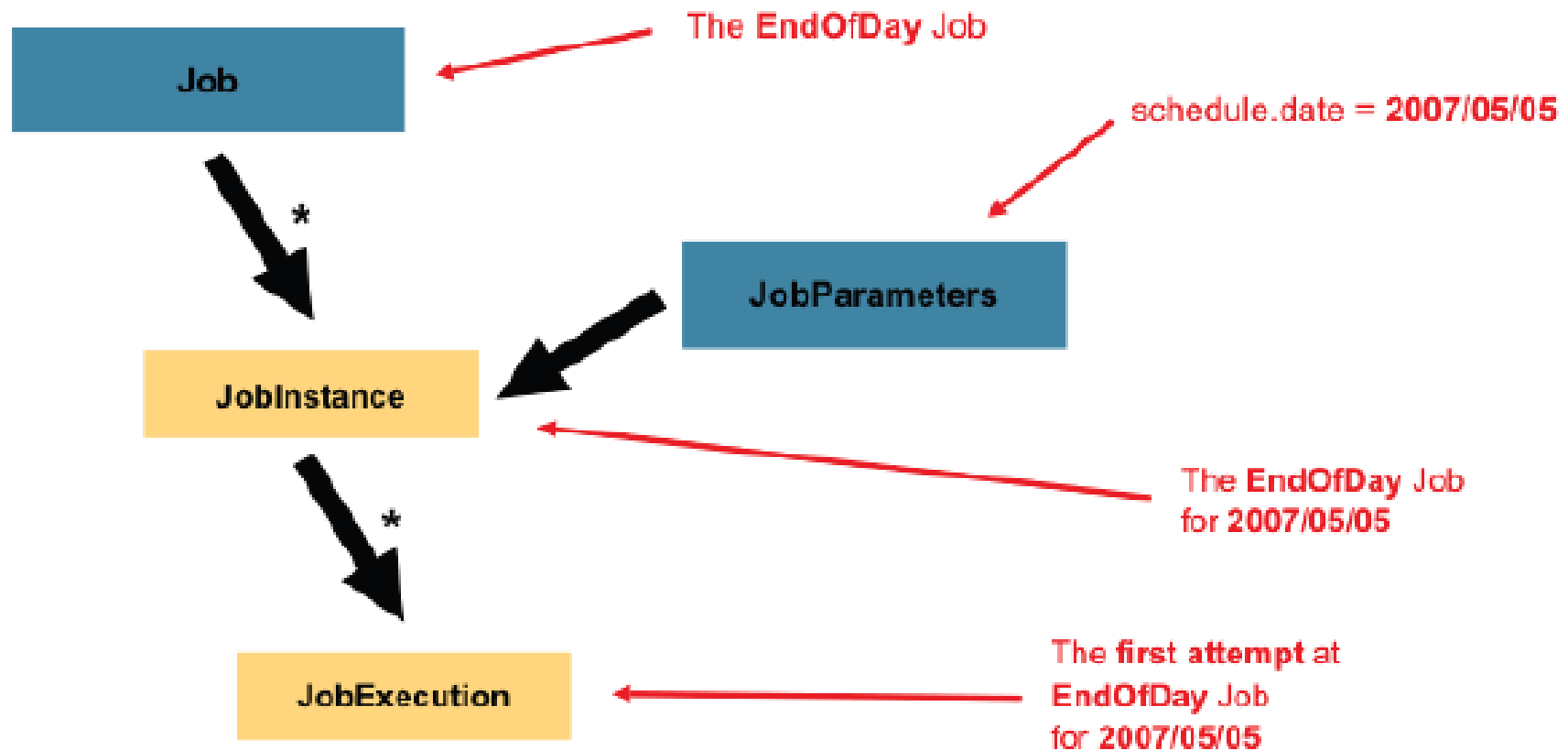
- Eine Job-Instanz kann potenziell mehrfach ausgeführt werden
  - Jede Job-Instanz aggregiert Job-Executions



- `status`
- `start`
- `end`
- `exitStatus`
- `createTime`
- `lastUpdated`
- `executionContext`
- `failureExceptions`

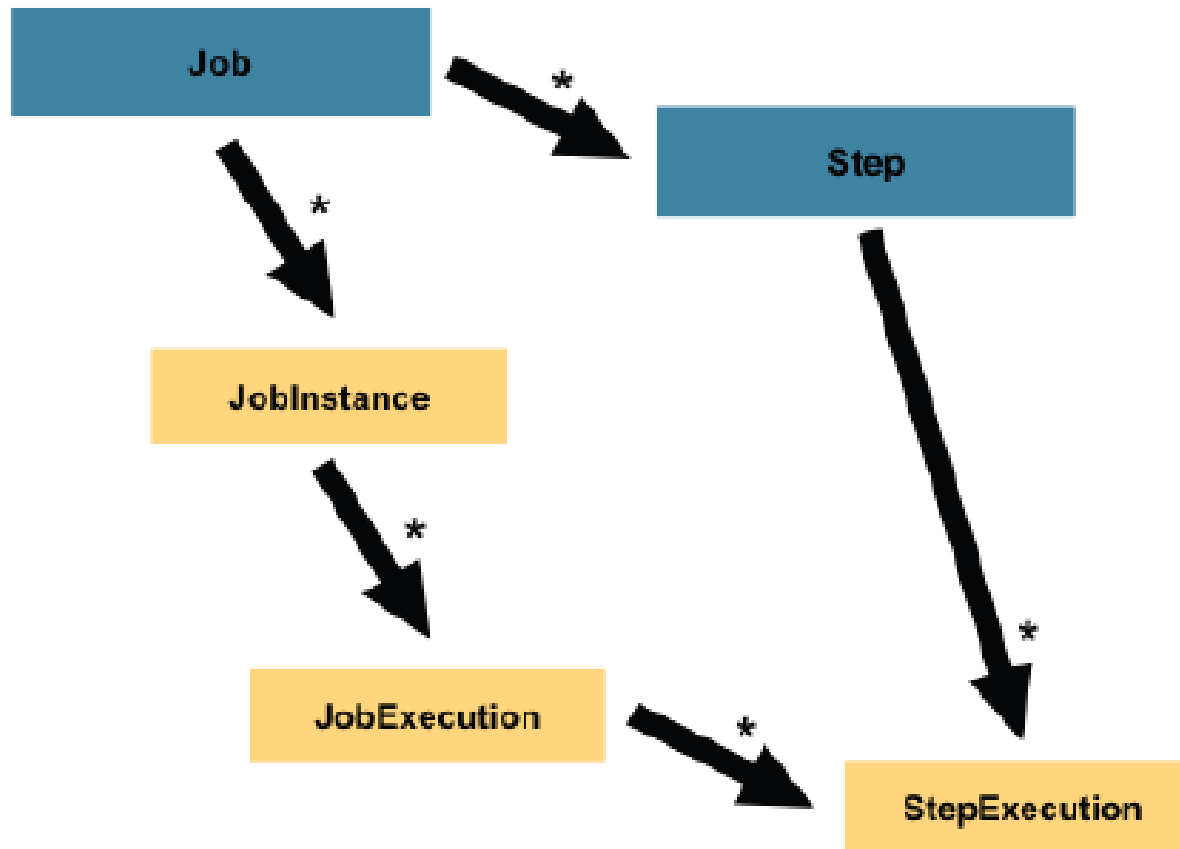


- Bei der Instanziierung eines Jobs werden JobParameter angegeben



## 1.3

# STEPS



- status
- startTime
- endTime
- exitStatus
- executionContext
- readCount
- writeCount
- commitCount
- rollbackCount
- readSkipCount
- processSkipCount
- filterCount
- writeSkipCount

- Step
  - Validierung
  - Formatierung
  - Sortieren
  - Aufsplitten
  - Merge
- Job
  - Parallelisierung
  - Zusammenführen

1.4

## **READER, PROCESSOR, WRITER**

- File-driven
- Database-driven
- Message-driven
  - Einsatz eines Messaging-Systems notwendig
  - Aber auch
    - Remote Aufruf
    - Eingang einer Mail
    - ...

2

## **BATCH SPEZIFIKATION**



2.1

## **SPRING BATCH UND JSR 352**

- Subprojekt der Spring-Community
  - Mit der üblich hohen Qualität an Ressourcen
    - Dokumentation
    - Community
    - Kommerzieller Support über VMWare
- Komplette integriert in die übrige Spring-Infrastruktur
  - Spring Core als Dependency Injection und AOP-Framework
  - Einfache Benutzung zusätzlicher Module
    - Spring Integration
    - Spring Data
- Stand 2014: Version: 3.x

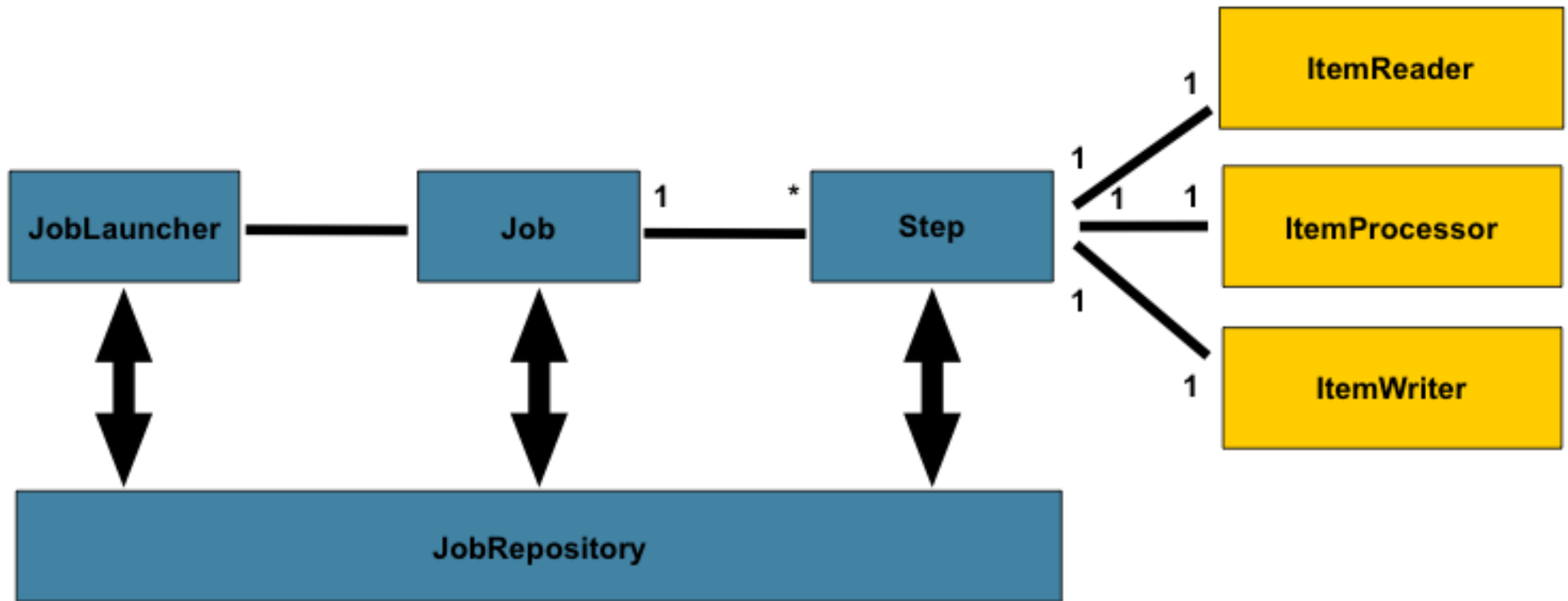
- Als Java Specification Request eine Spezifikation
  - Damit ein einheitliches API
    - Unter starkem Einfluss von Spring Batch definiert
      - Teile der Dokumentation sind identisch
  - Unabhängige Provider-Auswahl
    - Spring Batch ist ein Provider des JSR 352
      - Die Unterschiede zwischen Spring Batch und JSR 352 liegen im Detail
- Spezifikation in der Version 1.0 final seit 2013
  - Maintenance Release 2014

- JBatch
  - Verfügbar für die Java Standard-Edition
  - Auch als JEE-Anwendung verfügbar
- Bestandteil der JEE
  - Seit der Version 6
  - Damit ist jeder Applikationsserver ein Batch-Provider
  - Der Batch-Prozess hat Zugriff auf die Infrastruktur des Applikationsservers

2.2

## **LAUFZEITUMGEBUNG**

- Der JobLauncher ist verantwortlich für das Starten von Jobs
- Das JobRepository enthält die Informationen zu laufenden Jobs
  - Zustand
  - JobParameter
  - Job-Executions



3

## PROGRAMMIERUNG



## 3.1

# JOB SPECIFICATION LANGUAGE

- Jobs werden im XML-Format definiert
  - Ein Job ist Graphen-orientiert aufgebaut
  - Durchaus Ähnlichkeiten mit einem Business Process!

```
<job id="footballJob">  
  <step id="playerload" next="gameLoad"/>  
  <step id="gameLoad" next="playerSummarization"/>  
  <step id="playerSummarization"/>  
</job>
```

3.2

## **SPRING BATCH API**

```
<beans:beans
xmlns="http://www.springframework.org/schema/batch"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/batch
http://www.springframework.org/schema/batch/spring-batch-
2.2.xsd">
<job id="ioSampleJob">
<step id="step1">
<tasklet>
<chunk reader="itemReader" writer="itemWriter" commit-
interval="2"/>
</tasklet>
</step>
</job>
</beans:beans>
```

- Die Job-Definition ist eine normale Spring-Konfiguration
- Damit stehen sofort alle Features des Spring-Frameworks zur Verfügung
  - Context & Dependency Injection
  - AOP
  - Infrastruktur
    - DataSources
    - Transaktionsmanager

- Beispiel: Spring Tools Suite



3.3

## **JSR 352/JBATCH**

```
<?xml version="1.0" encoding="UTF-8"?>
<job id="myJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
version="1.0">
    <step id="myStep" >
        <chunk item-count="3">
            <reader ref="myItemReader"/>
            <processor ref="myItemProcessor"/>
            <writer ref="myItemWriter"/>
        </chunk>
    </step>
</job>
```

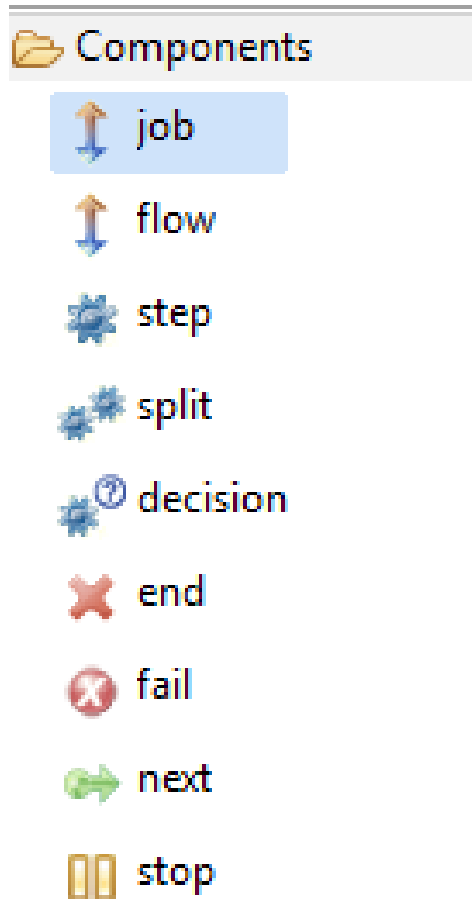


- In der Batch-Spezifikation ist die Verwendung eines CDI-Frameworks nicht explizit angegeben
- In der Praxis wird jedoch das Java-CDI-Framework benutzt
  - `@javax.inject.Inject`
  - `@javax.inject.Named`

3.4

## **BEFEHLSSATZ**

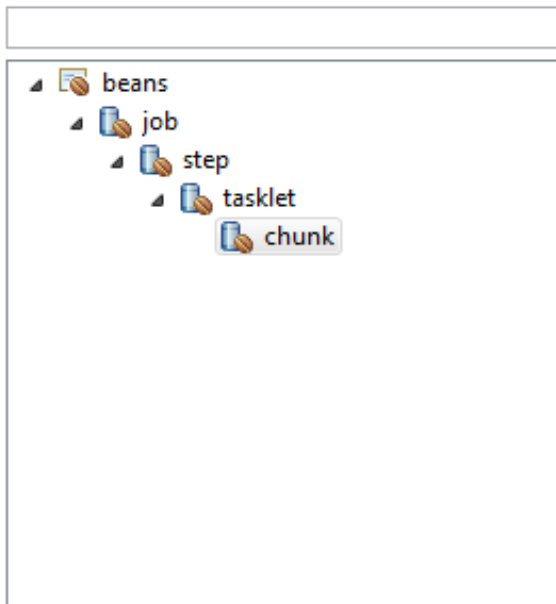
- Die Palette des STS zeigt die in Spring Batch verfügbaren Befehle



- Ein Step selbst besteht
  - aus einem weiteren Sub-Job
  - Einem Tasklet (Spring Batch) oder einem Batchlet (JSR 352)
    - Diese dienen zur Ausführung einer beliebigen Sequenz von Anweisungen
  - Einem Chunk

## Batch Overview

Select an element to edit its details.



- Durchlaufen eine festgelegte Sequenz
  - Reader
  - Processor
  - Writer
- Das Einlesen und Schreiben der Daten erfolgt blockweise
  - Bei Bedarf
    - Transaktionell
    - Mit Restart/Retry
- Chunks sind ein großer Unterschied zwischen Batch- und Business-Prozessen

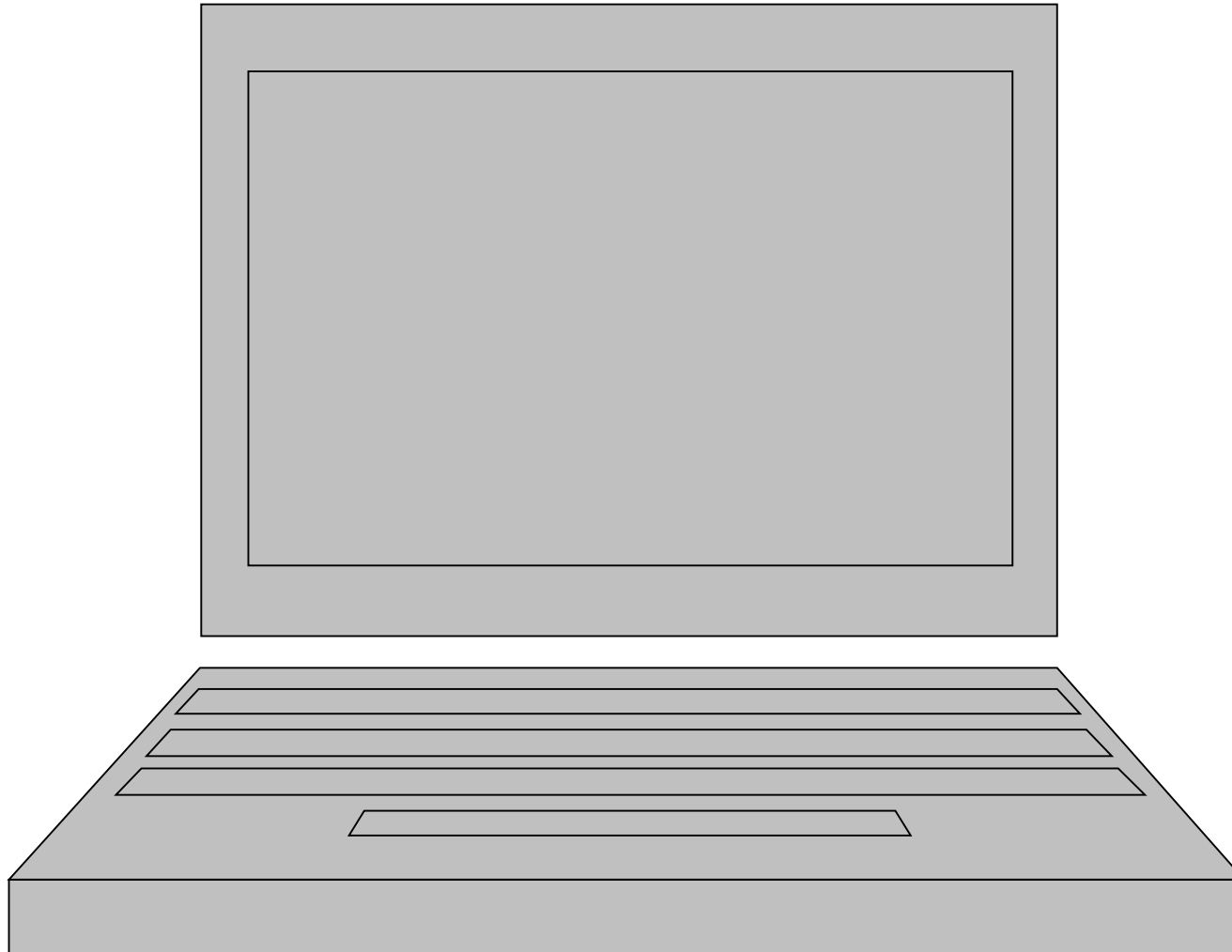
3.5

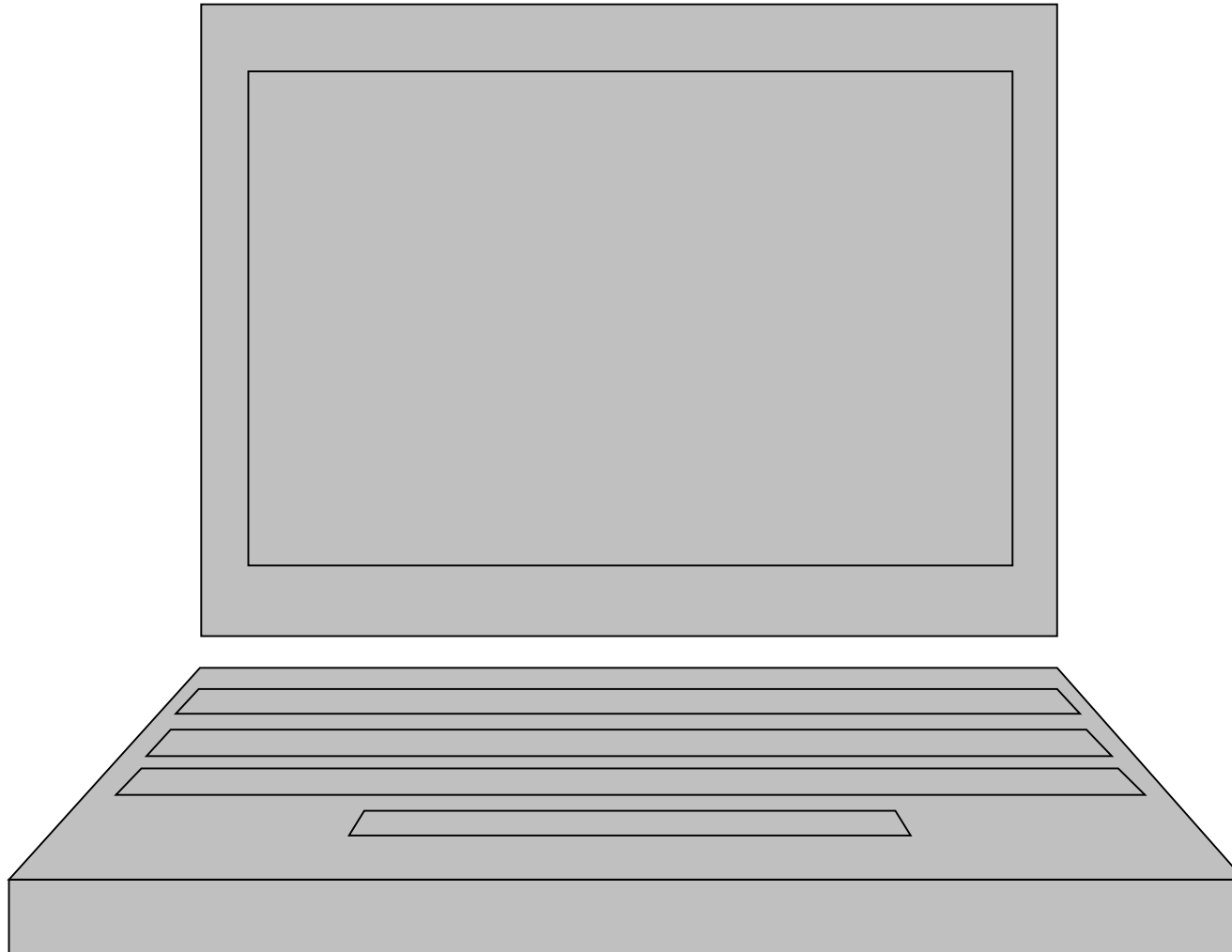
## **JAVA API**

- Schnittstellen für
  - `ItemReader`
  - `ItemWriter`
  - `ItemProcessor`
- Gleich benannt für Spring Batch und JSR 352
  - Nur unterschiedliche Packages

- Der Job-Lauf kann durch Listener detailliert verfolgt werden
  - JobListener
  - StepListener
  - RepeatListener
  - ...







3.6

## TESTEN MIT SPRING BATCH

```
import org.springframework.batch.test.JobLauncherTestUtils;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "/simple-job-launcher-
context.xml", "/jobs/simpleJob.xml", "/job-runner-
context.xml" })

public class JobStepFunctionalTests {
    @Autowired
    private JobLauncherTestUtils jobLauncherTestUtils;
    @Test
    public void testJobLaunch() throws Exception {
        jobLauncherTestUtils.launchJob();
        //Assertions
    }
}
```

```
import javax.batch.operations.JobOperator;

public class ChunkSimpleTest {

    @Test
    public void testChunkSimple() throws Exception {

        JobOperator jobOperator =
BatchRuntime.getJobOperator();

        //Assertions

    }

}
```

3.7

## **JOB-STATE**

- Persistente Key-Value-Paare
  - eine persistente Map
  - Jede Java-Anwendung als Bestandteil einer Job-Ausführung können Daten im `ExecutionContext` ablegen
    - Und stehen damit beispielsweise nach einem Restart zur Verfügung

3.8

## **ABLAUFSTEUERUNG**



- `decision`
- `flow`
- `split`
- `fail`
- `next`
- `stop`

- Chunk-Konfiguration

```
<chunk reader="itemGenerator"  
        writer="itemWriter"  
        commit-interval="1"  
        retry-limit="3">  
</chunk>
```

- Job-Konfiguration

```
<job xmlns=http://www.springframework.org/schema/batch  
id="restartSampleJob"  
restartable="true" >
```

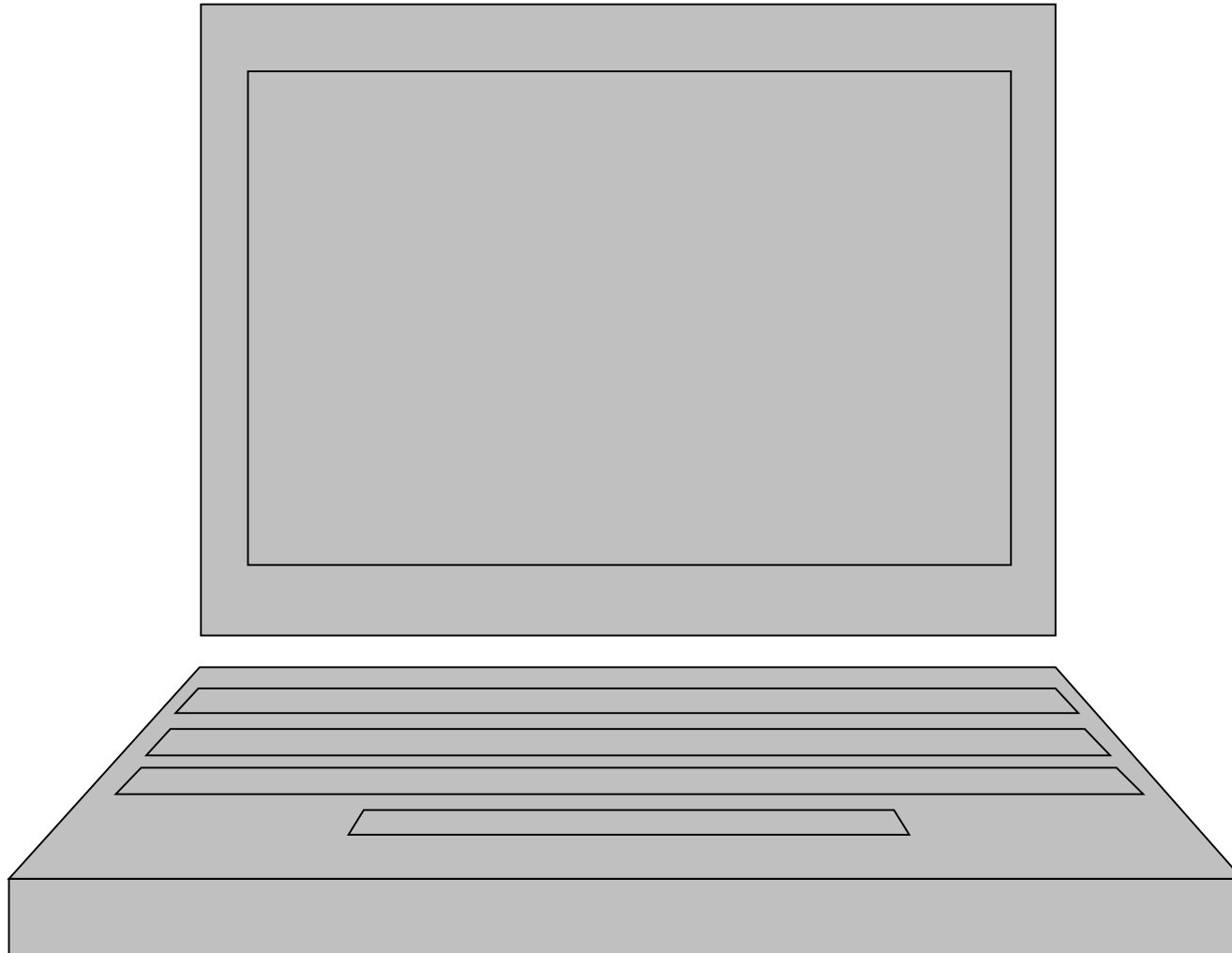
- Die Eingangsdaten werden durch einen Partitioner in verschiedene disjunkte Partitionen getrennt
  - Strategien
    - Fixed and Even Breakup
    - Aufteilung via Schlüsselspalte
    - Partitionierungstabelle
    - Wertetabelle
    - Aufteilung via Database View
    - Processing Indicator
    - Table to Flat File
    - Hashing Column
- Ähnlichkeiten zum Map/Reduce-Algorithmus vorhanden
  - Auch die Abfrage einer komplexen NoSQL-Datenbank erfolgt nach Batch-Algorithmen

4

## BEISPIELE

4.1

## **SPRING BATCH**



4.2

## **JSR 352**



