

# Hands On Spring Batch



Mardi 21 Février 2012

## Olivier Bazoud



# Intervenant

**Olivier Bazoud**, @obazoud, Ekino (FullSIX Group)

Architecte technique

Java EE / Spring, Spring Batch / Node.js / NoSQL

Spring User Group France

Co auteur de « Spring Batch in Action »

Cette présentation est issue d'un refactoring de présentations

« Spring Batch » données avec **Julien Jakubowski** (@jak78 / OCTO Technology)

# Ce qui vous attend

- Quelques principes des batchs
- Introduction à Spring Batch

Dessines moi un batch...



# Introduction



# Batch : de quoi parle-t-on ?

Batch processing = suite de traitements sur ensemble de données...



<http://www.flickr.com/photos/burnblue/308441464/>



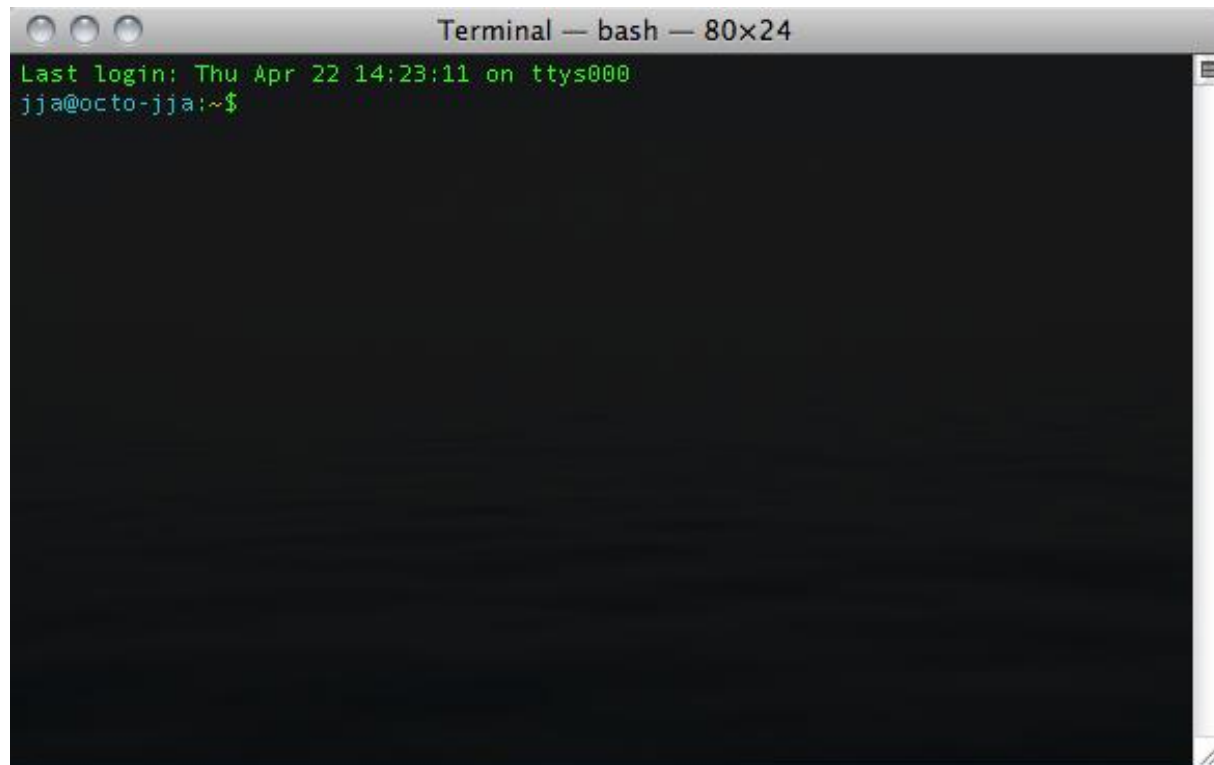
# Batch : de quoi parle-t-on ?

... potentiellement grands volumes...



# Batch : de quoi parle-t-on ?

... sans intervention d'un utilisateur humain – pas d'interface homme-machine

A screenshot of a terminal window titled "Terminal — bash — 80x24". The window has a dark background with green text. The first line shows the login message: "Last login: Thu Apr 22 14:23:11 on ttys000". The second line shows the prompt "jja@octo-jja:~\$" followed by a dollar sign "\$" indicating the shell is ready for input.

```
Terminal — bash — 80x24
Last login: Thu Apr 22 14:23:11 on ttys000
jja@octo-jja:~$
```

# Batch : de quoi parle-t-on ?

Exemples:

- Import flat / XML dans une base de données
- Intégration de flux financiers dans un SI
- ...

Un batch **n'est pas** un scheduler:

- Cron, Quartz, \$U...
- Mais un scheduler peut le lancer



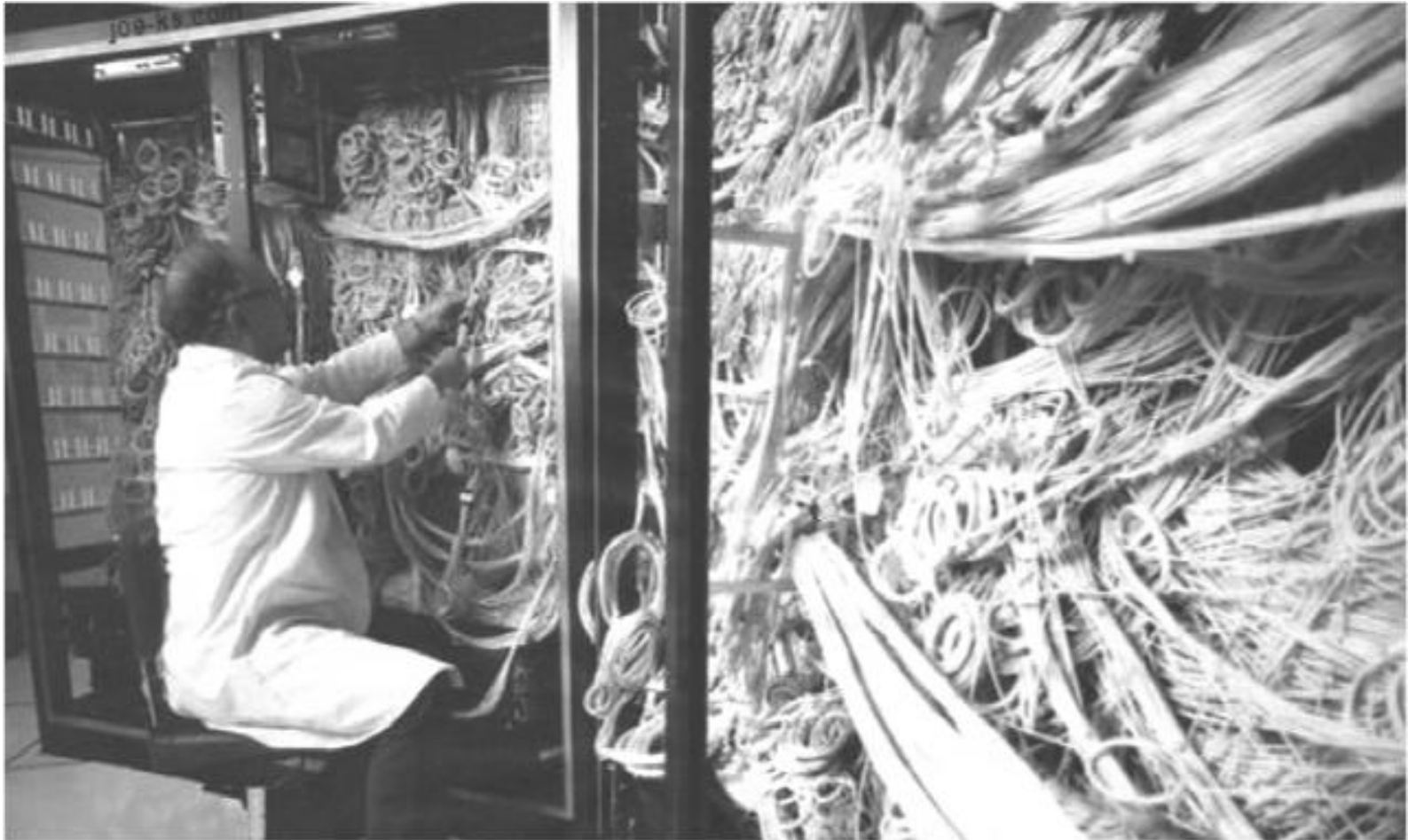
# Problèmes récurrents

- Fiabilité

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
  at org.jdom.ContentList.ensureCapacity(ContentList.java:355)
  at org.jdom.ContentList.add(ContentList.java:234)
  at org.jdom.ContentList.add(ContentList.java:131)
  at java.util.AbstractList.add(AbstractList.java:91)
  at org.jdom.Element.addContent(Element.java:811)
  at org.jdom.DefaultJDOMFactory.addContent(DefaultJDOMFactory.java:180)
  at org.jdom.input.SAXHandler.flushCharacters(SAXHandler.java:693)
  at org.jdom.input.SAXHandler.flushCharacters(SAXHandler.java:660)
  at org.jdom.input.SAXHandler.endElement(SAXHandler.java:716)
  at com.sun.org.apache.xerces.internal.parsers.AbstractSAXParser.endElement(AbstractSAXParser.java:642)
  at com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl.scanEndElement(XMLDocumentFragmentScannerImpl.java:1789)
  at com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl$FragmentCachingReader.parse(XMLDocumentFragmentScannerImpl.java:5434)
  at com.sun.org.apache.xerces.internal.impl.XMLDocumentScannerImpl.next(XMLDocumentScannerImpl.java:605)
  at com.sun.org.apache.xerces.internal.impl.XMLNSDocumentScannerImpl.next(XMLNSDocumentScannerImpl.java:117)
  at com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl.scanDocument(XMLDocumentFragmentScannerImpl.java:308)
  at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(XML11Configuration.java:884)
  at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(XML11Configuration.java:872)
  at com.sun.org.apache.xerces.internal.parsers.XMLParser.parse(XMLParser.java:107)
  at com.sun.org.apache.xerces.internal.parsers.AbstractSAXParser.parse(AbstractSAXParser.java:1241)
  at com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser.parse(SAXParserImpl.java:638)
  at org.jdom.input.SAXBuilder.build(SAXBuilder.java:489)
  at org.jdom.input.SAXBuilder.build(SAXBuilder.java:928)
  at fr.sug.springbatch.plainolddbbatch.PlainOldBatch.run(PlainOldBatch.java:72)
  at fr.sug.springbatch.plainolddbbatch.PlainOldBatch.main(PlainOldBatch.java:37)
```

# Problèmes récurrents

- Maintenabilité



# Problèmes récurrents

- Réinvention de la roue... carrée



# Spring Batch propose...

- Un cadre
- Un vocabulaire (domain language)
- Traitement par lots
- Gestion des transactions
- **Spring** dans ses batchs
- Livré avec:

Parallélisme   Reprise sur erreur   Spring Batch Admin

Partitionnement   Gestion des flows

Spring Batch

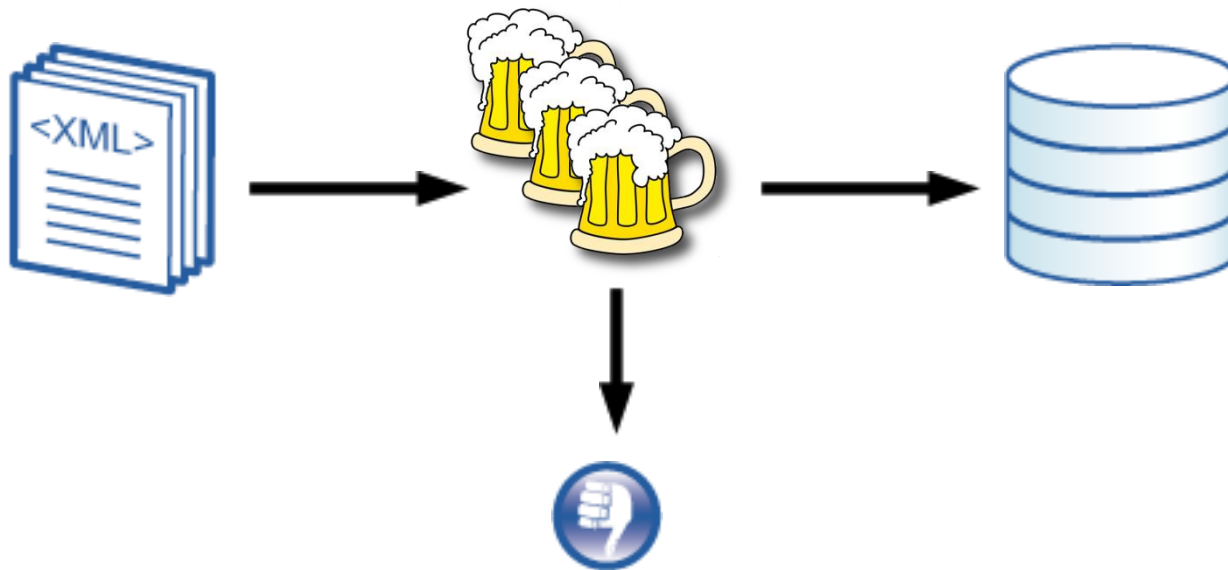


# Notion de bases



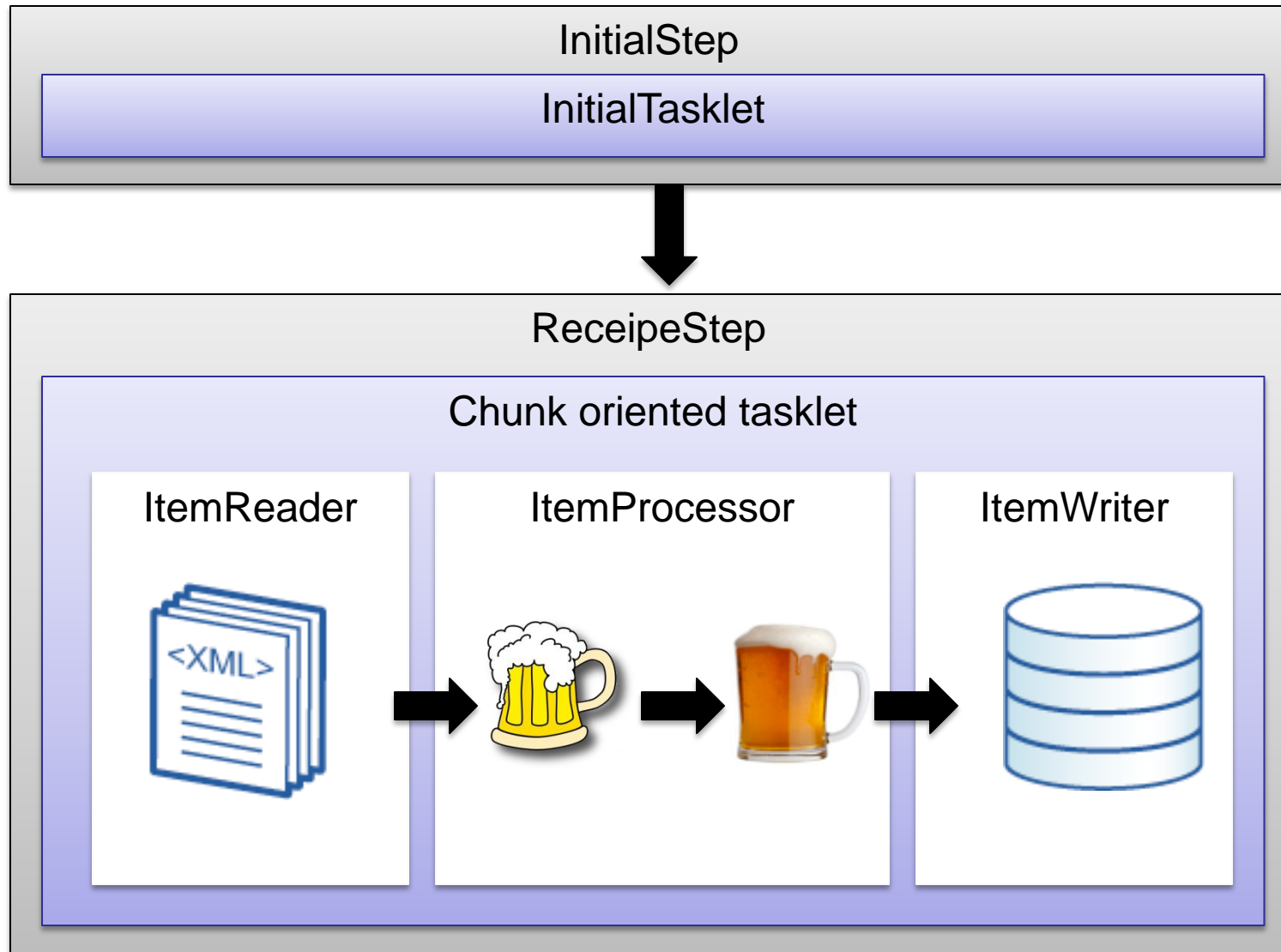
# Exemple avec « Spring Batch »

- Ecrire la date de début du batch
- Lire le fichier XML de recettes, au format BeerXML
- Filtrer certaines recettes
- Ecrire en base de données



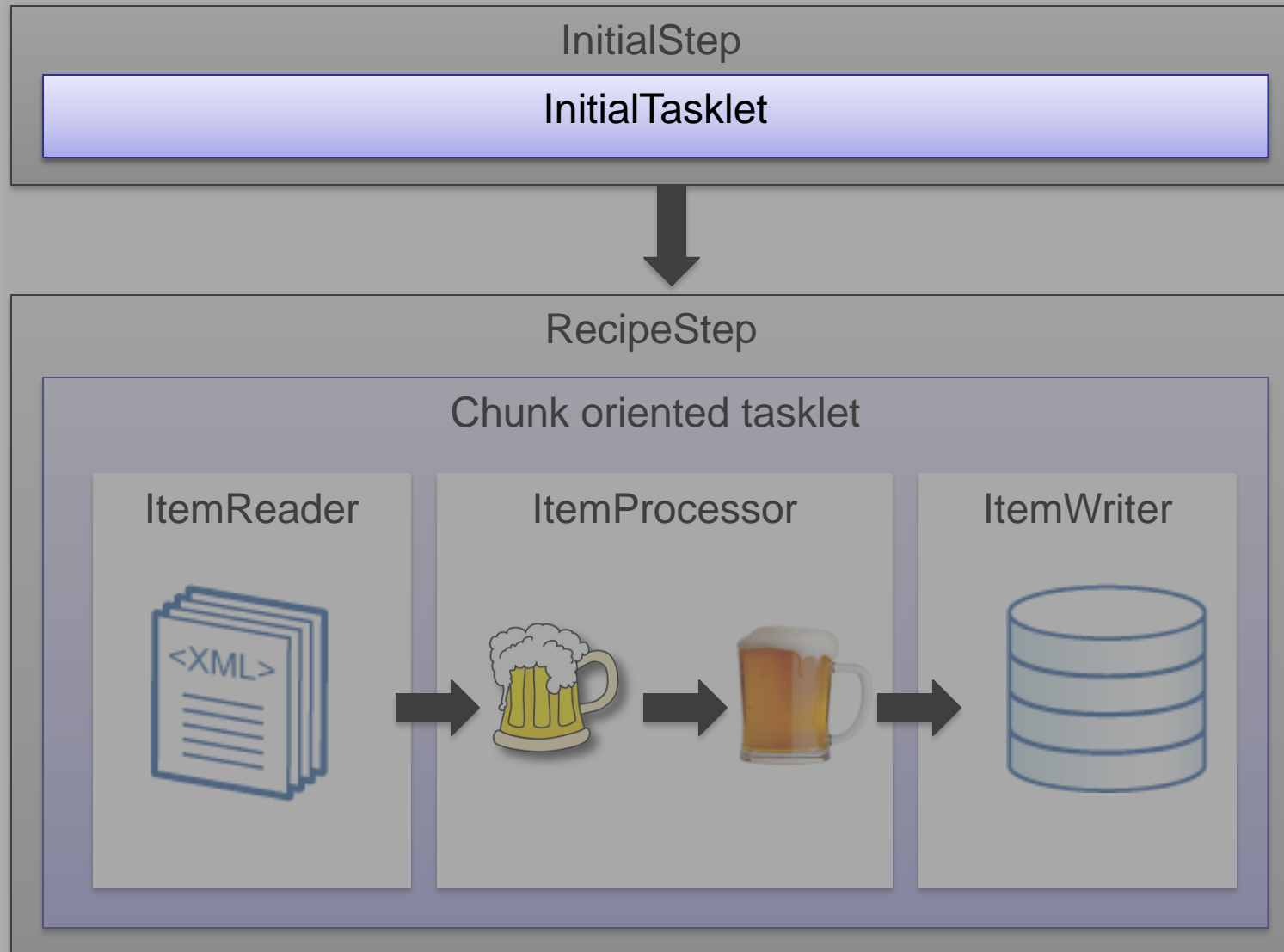


# Schéma du batch





# Tasklet



# Tasklet

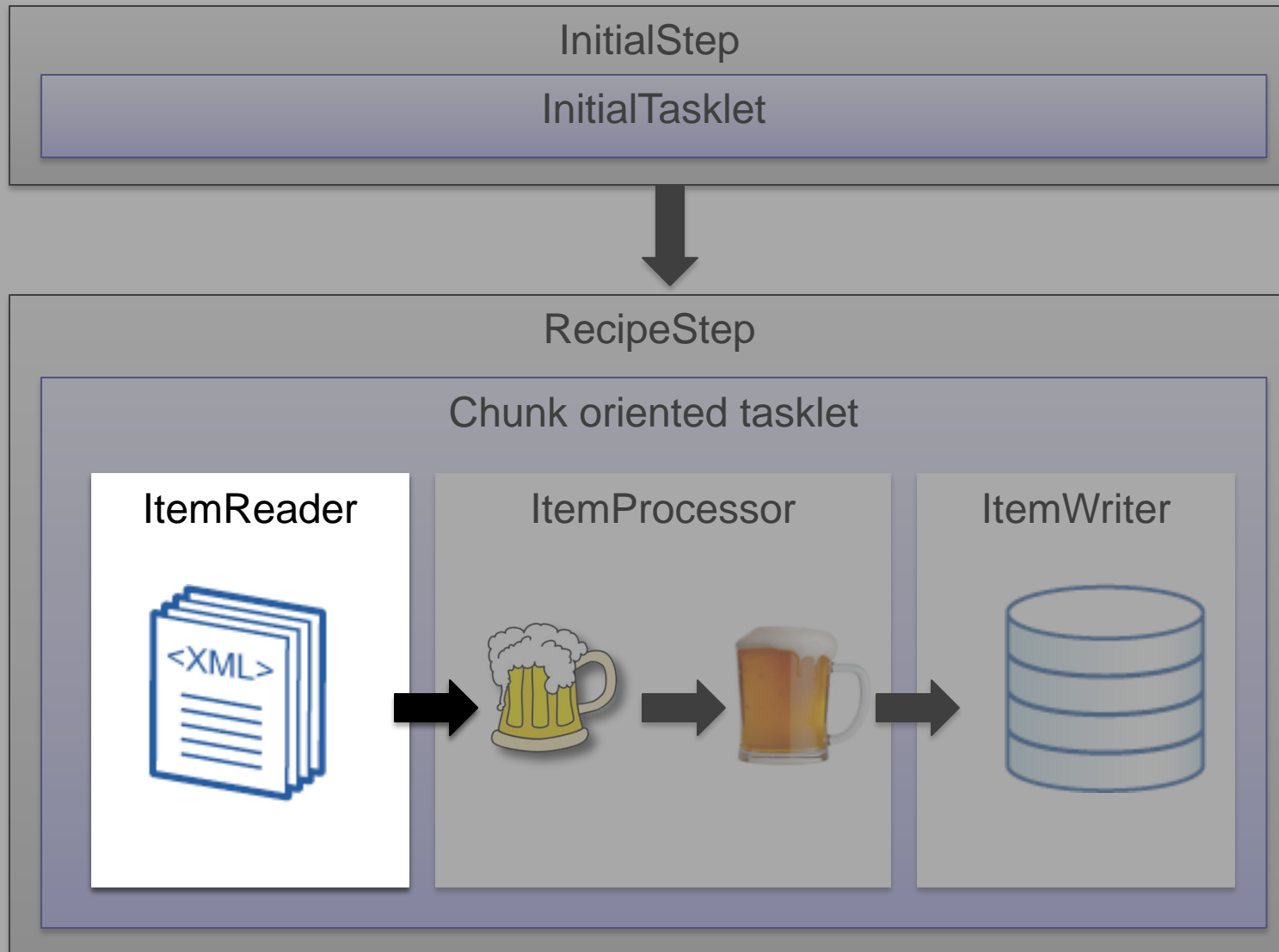
- Besoin
  - Effectuer une tâche unitaire
- Exemples
  - Supprimer un répertoire et son contenu
  - Unzip d'un fichier
  - Appel d'une procédure stockée
  - Appel d'un web service

# Morceaux de code

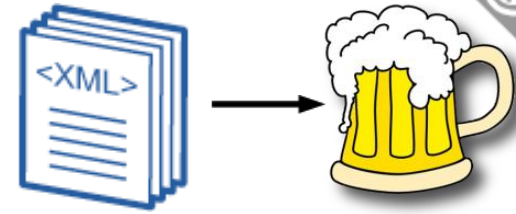
```
@Component
public class InitialTasklet implements Tasklet {
    // logger
    private static final Logger LOG = Logger.getLogger(InitialTasklet.class);
    private static final FastDateFormat DATE_FORMAT =
        FastDateFormat.getInstance("yyyy/MM/dd HH:mm:ss");

    @Override
    public RepeatStatus execute(StepContribution contribution, ChunkContext chunkContext) {
        LOG.info("Initializing at " + DATE_FORMAT.format(new Date()) + ".");
        return null;
    }
}
```

# ItemReader



# ItemReader



- Besoin
  - Lire le XML de recettes
- ItemReader
  - Fournit des items en entrée

```
public interface ItemReader<T> {
```

```
    * Reads a piece of input data and advance to the next one. Implementations  
    T read() throws Exception, UnexpectedInputException, ParseException;
```

```
}
```

```
<xml>
```

```
123;AB;
```

```
456;CD;
```

```
SELECT ... FROM ...
```



# Morceaux de code



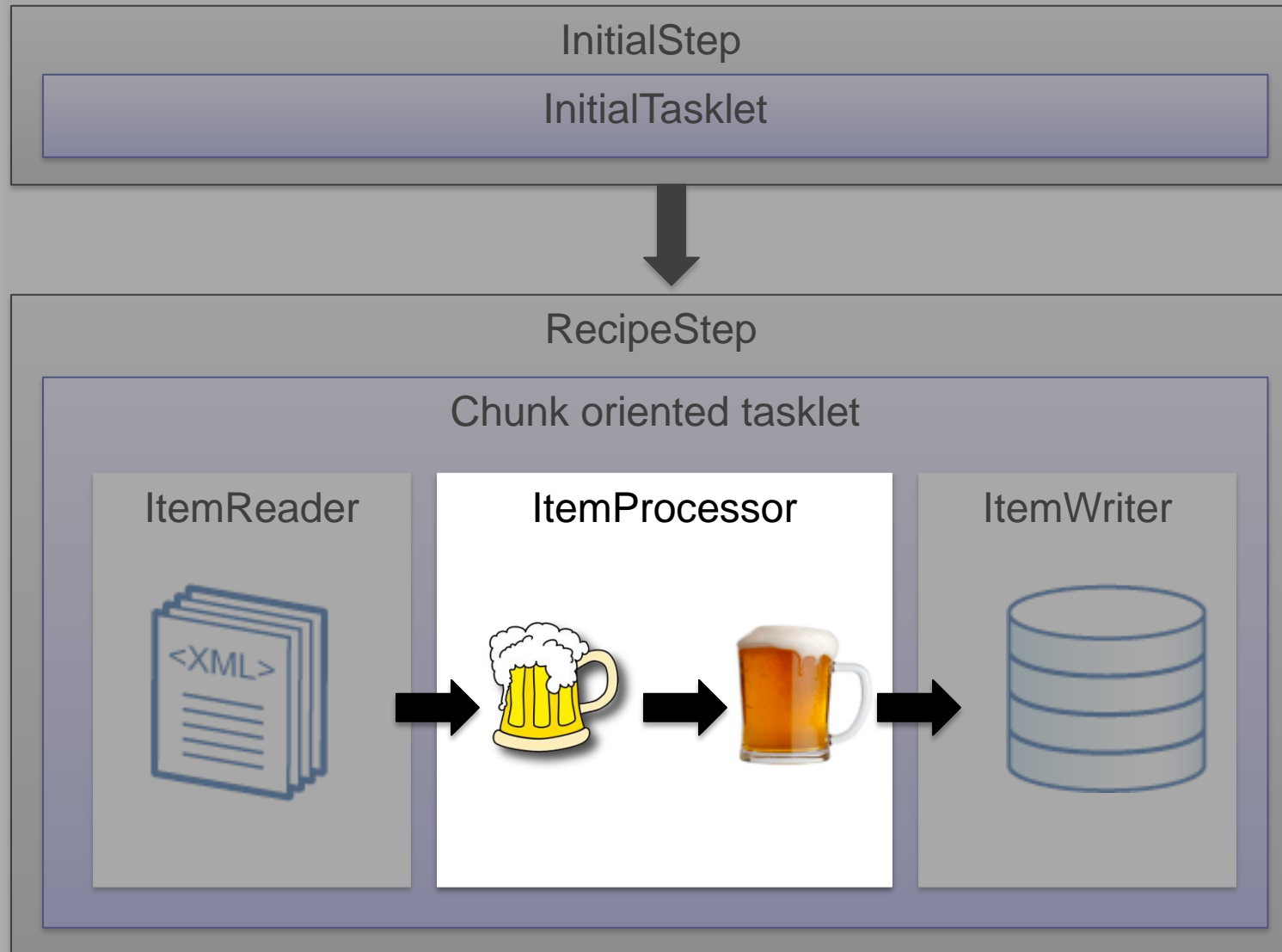
<!-- Le reader a pour responsabilité de lire un élément.

Ici : un objet recipe. Notre reader utilise une implémentation de lecture XML déjà fournie par Spring Batch.

-->

```
<bean id="recipesReader"
      class="org.springframework.batch.item.xml.StaxEventItemReader"
      scope="step">
  <property name="resource" value="#{jobParameters[recipes]}" />
  <property name="fragmentRootElementName" value="RECIPE" />
  <property name="unmarshaller" ref="recipeMarshaller" />
</bean>
```

# ItemProcessor





# ItemProcessor



- Besoin
  - Transforme, valide et / ou filtre une recette
- ItemProcessor
  - Transforme un item en un autre
  - Filtrer ou rejeter un item
  - Emplacement pour les « règles métier »

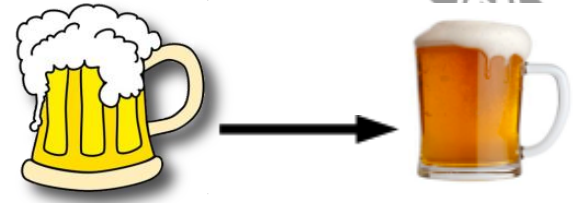
```
public interface ItemProcessor<I, O> {
```

```
    * Process the provided item, returning a potentially modified or new item for continued
```

```
    O process(I item) throws Exception;
```

```
}
```

# Morceaux de code



```
<!-- Le processor a pour responsabilité de, potentiellement :
```

- transformer un objet avant traitement
- rejeter un objet ou non

```
Notre processeur est un composite, c.a.d qu'il chaîne 2 implémentations de processor.
```

```
-->
```

```
<bean id="recipesProcessor"
```

```
  class="org.springframework.batch.item.support.CompositeItemProcessor">
```

```
    <property name="delegates">
```

```
      <list>
```

```
        <bean class="fr.sug.springbatch.example.processor.IdProcessor" />
```

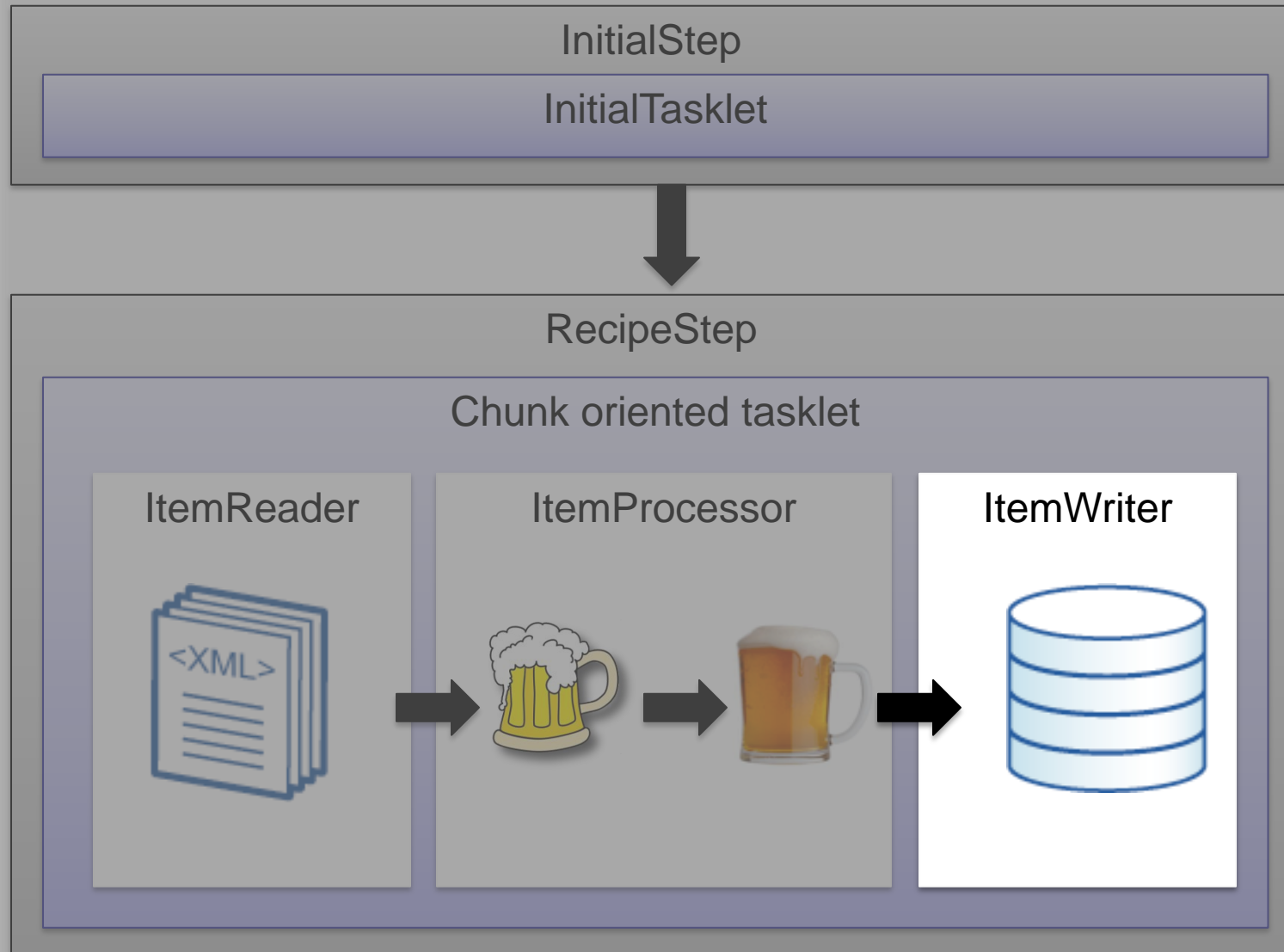
```
        <bean class="fr.sug.springbatch.example.processor.RecipeProcessor" />
```

```
      </list>
```

```
    </property>
```

```
</bean>
```

# ItemWriter



# ItemWriter



- Besoin
  - Décharge les bières dans une base SQL
- ItemWriter
  - Ecrit les items

```
public interface ItemWriter<T> {
```

```
    * Process the supplied data element. Will not be called with any null items
```

```
    void write(List<? extends T> items) throws Exception;
```

```
}
```

```
<xml>
```

```
INSERT... INTO ...
```

```
123;AB;
```

```
456;CD;
```



# Morceaux de code



fullsix

`<!-- Le writer a pour responsabilité d'écrire un lot (chunk) d'objets.  
Il s'agit ici d'une implémentation custom. de writer`

`-->`

`<bean id="recipesWriter"`

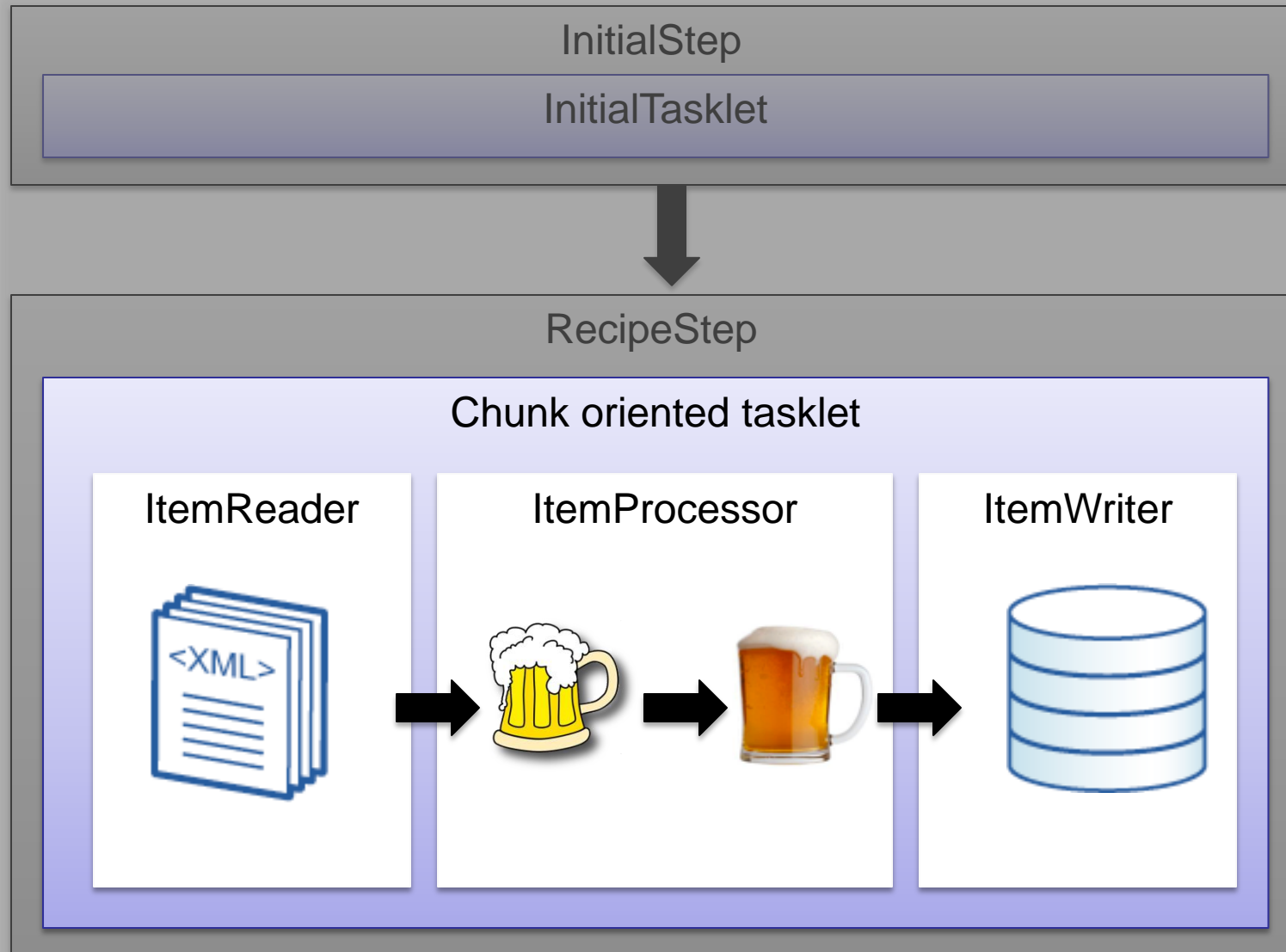
`class="fr.sug.springbatch.example.writer.RecipeItemWriter">`

`<property name="dataSource" ref="dataSource" />`

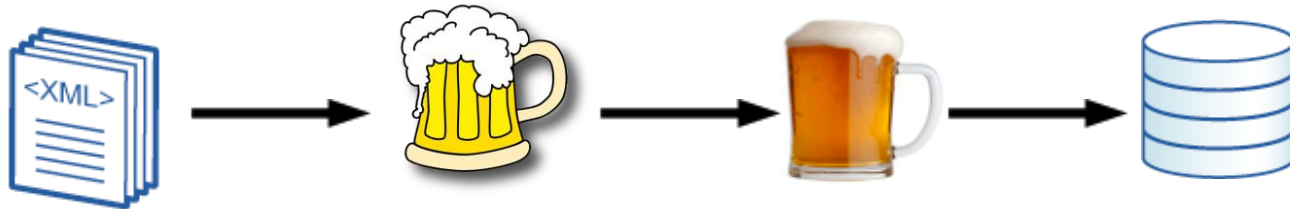
`<property name="sqlQueries" ref="sqlQueries" />`

`</bean>`

# Chunk



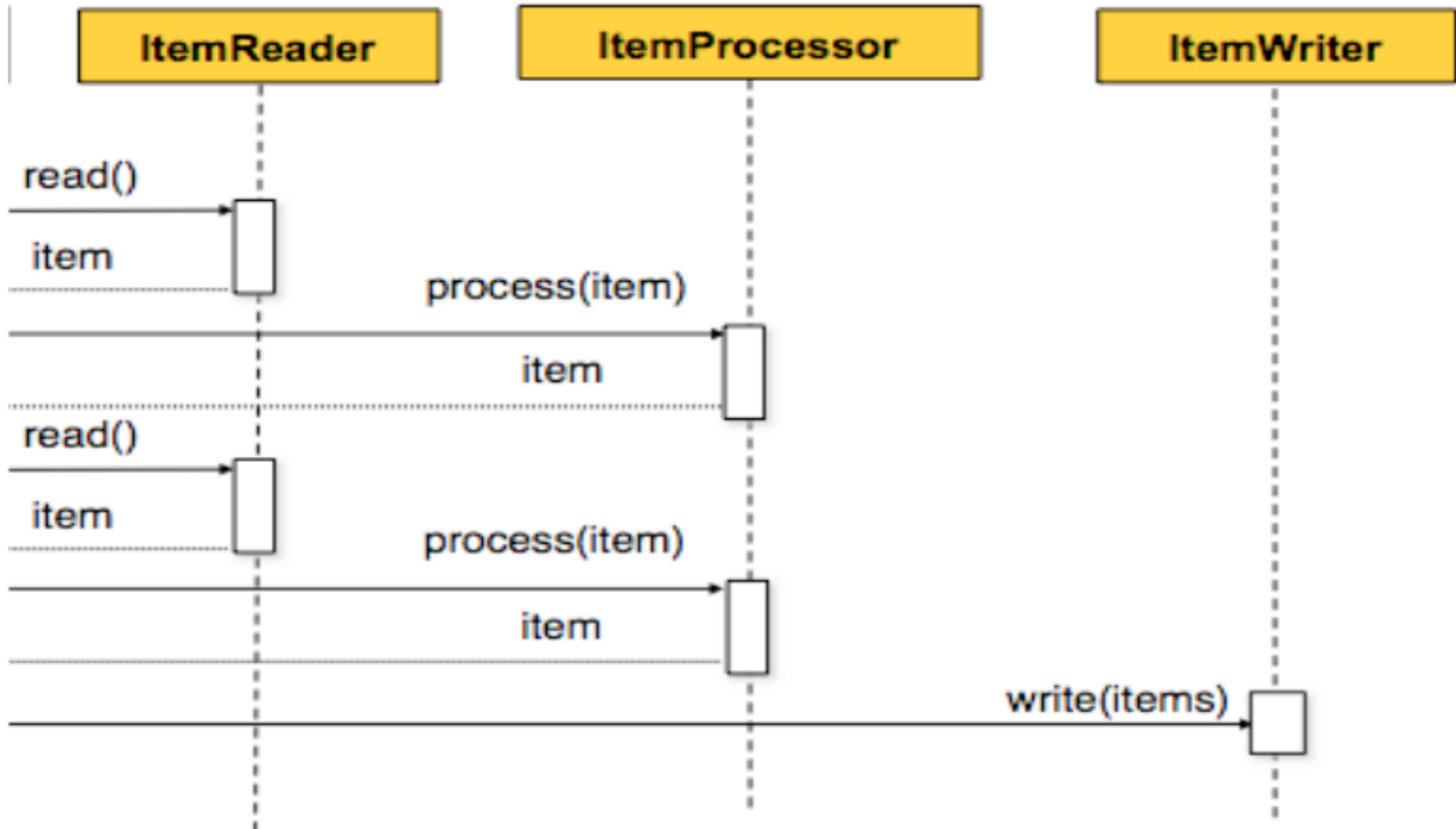
# Chunk



- **Besoin**
  - Lire, transformer et écrire
- **Chunk**
  - Lire et transformer les données successivement
  - Ecrire le lot de données
  - Le commit-interval définit la taille du lot (différent de la taille du fichier)
  - Gestion de la transaction : Commit/Rollback



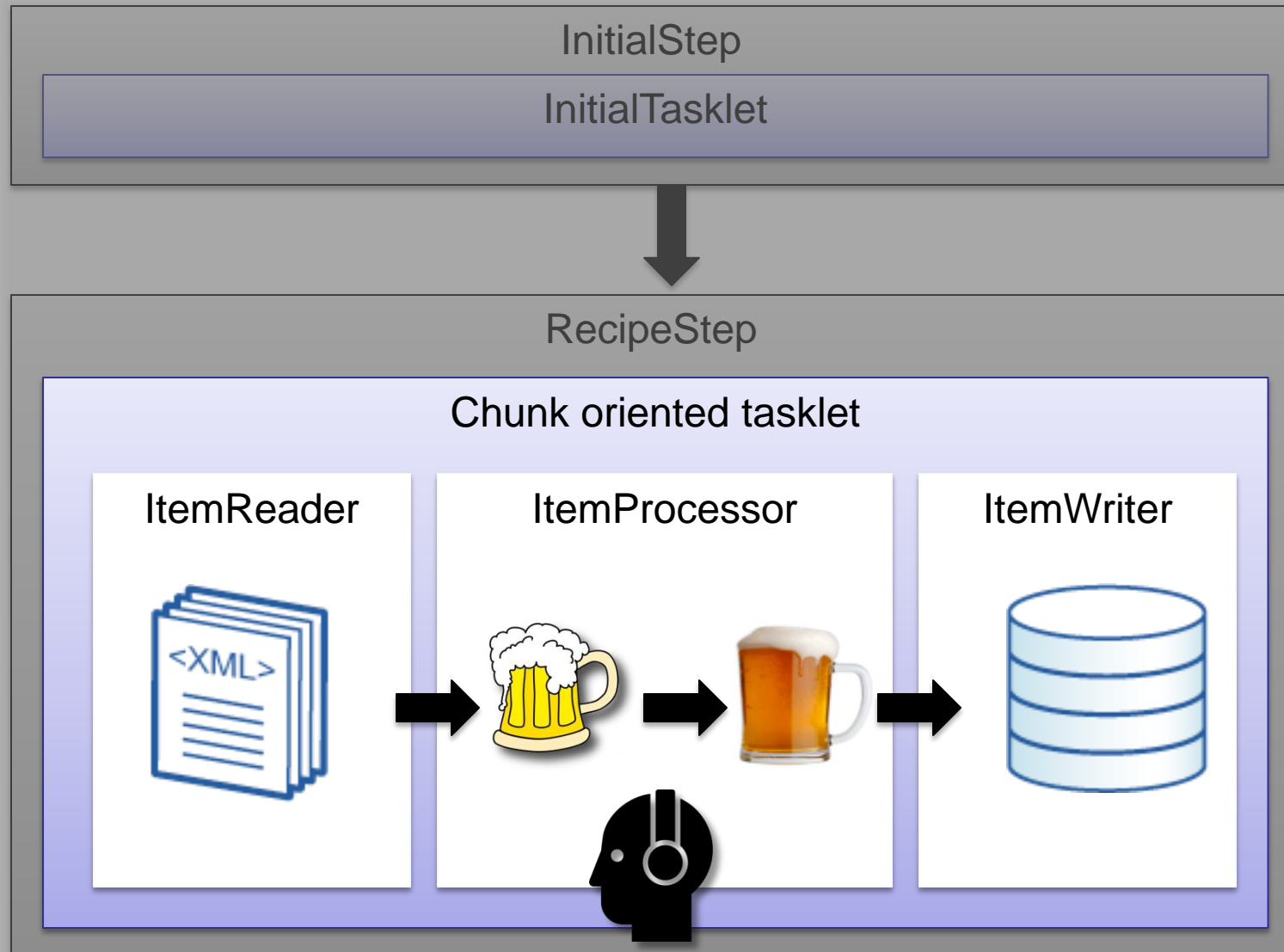
# Chunk



# Morceaux de code

```
<batch:chunk reader="recipesReader" writer="recipesWriter"  
             processor="recipesProcessor"  
             commit-interval="${job.commit.interval}">  
</batch:chunk>
```

# Listener

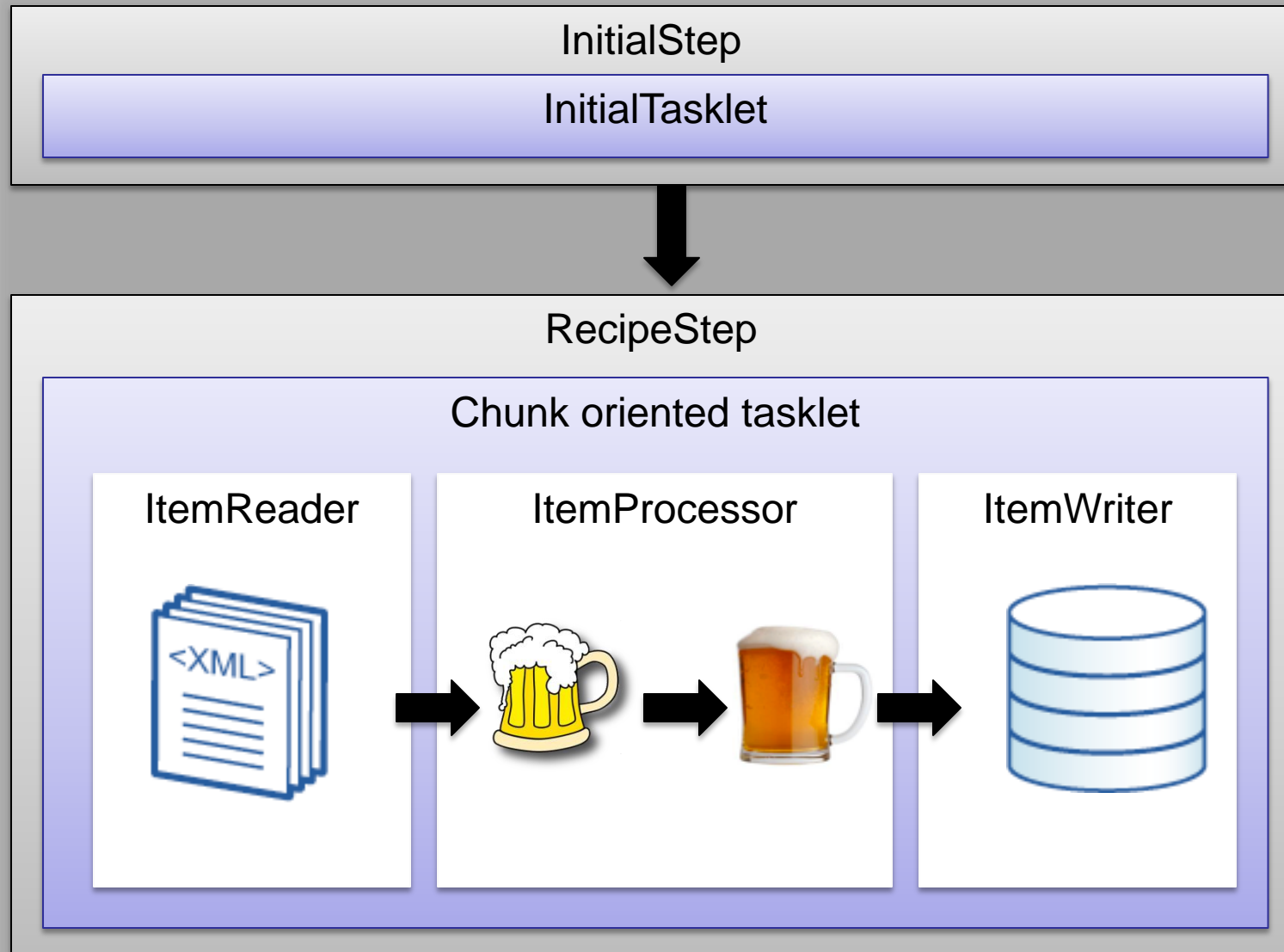


# Morceaux de code



```
<batch:chunk reader="recipesReader" writer="recipesWriter"
             processor="recipesProcessor"
             commit-interval="${job.commit.interval}">
  <batch:streams>
    <batch:stream ref="recipeExcludeWriter" />
  </batch:streams>
</batch:chunk>
<batch:listeners>
  <batch:listener ref="stepRecipeExecutionListener" />
</batch:listeners>
```

# Step

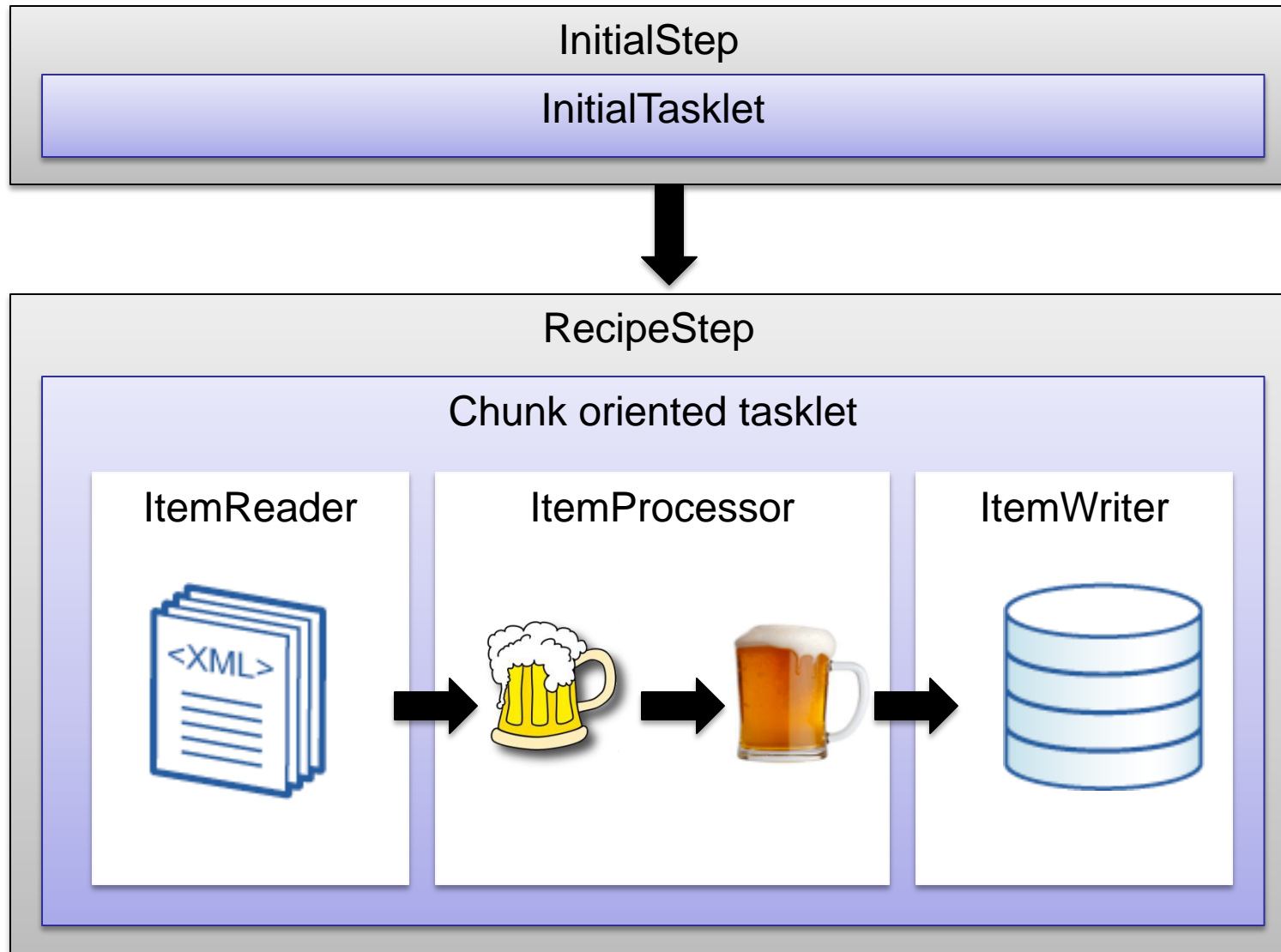


# Step

- Besoin
  - Etape dans le processus du batch
  - Contrôle le workflow

```
<batch:step id="initialStep">  
    <batch:tasklet ref="initialTasklet" />  
    <batch:next on="*" to="recipeStep" />  
</batch:step>  
<batch:step id="recipeStep">  
    <batch:tasklet>  
</batch:step>
```

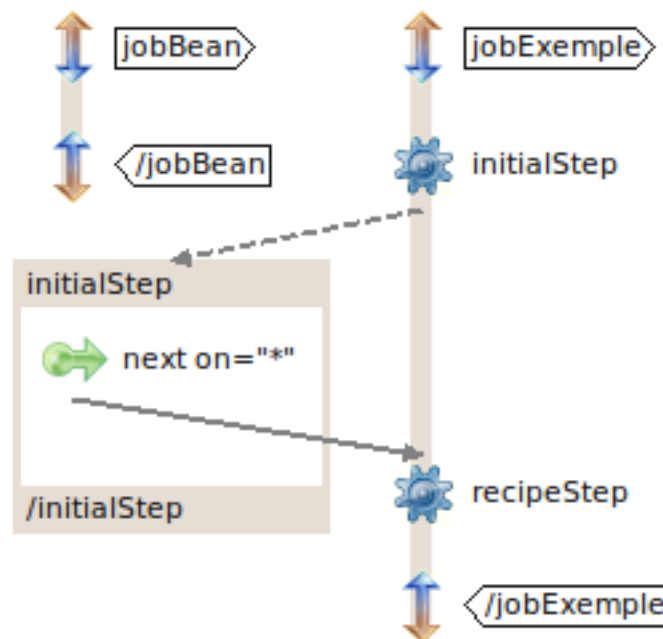
# Job





# Job

- Besoin
  - Décrire les étapes du batch
  - Composé d'un ou plusieurs steps



# Configuration

```
<!-- Notre Job est composé de 2 steps. -->
<batch:job id="jobExemple" parent="jobBean">
  <!-- Le premier step est un exemple de Tasklet.
  Il ne fait qu'afficher l'heure. -->
  <batch:step id="initialStep">
    <batch:tasklet ref="initialTasklet" />
    <batch:next on="*" to="recipeStep" />
  </batch:step>
  <!-- Le second step est un exemple de Tasklet de type chunk.
  Il met en oeuvre un traitement par lots en branchant
  le reader, le processor et le writer ensemble.
  -->
  <batch:step id="recipeStep">
    <batch:tasklet>
      <batch:chunk reader="recipesReader" writer="recipesWriter"
        processor="recipesProcessor"
        commit-interval="${job.commit.interval}">
        <batch:streams>
          <batch:stream ref="recipeExcludeWriter" />
        </batch:streams>
      </batch:chunk>
      <batch:listeners>
        <batch:listener ref="stepRecipeExecutionListener" />
      </batch:listeners>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

# Lancer un job directement

```
java $JAVA_OPTS -classpath $(echo ./lib/*.jar . | sed 's/ /:/g') \  
  org.springframework.batch.core.launch.support.CommandLineJobRunner \  
  fr/sug/springbatch/example/batch-context.xml jobExemple \  
  recipes=file:///data/projects/fr-sug-spring-batch/dirtybatch/data/recipes.xml \  
  time=$(date +%Y%m%d-%k%M%S)
```

# Tests

- Tests unitaires facilités via le découpage Spring Batch
  - writers, processors etc...
- Tests d'intégration facilités par Spring
  - @RunWith
  - Step, Job

**Code coverage****99.4%**

100.0% line coverage ▼

97.1% branch coverage ▲

6 tests

514 ms ▲

**Test success****100.0%**

0 failures

0 errors

## Advanced Spring Batch



# To infinity and beyond



# Un batch plus robuste



# Reprise sur erreur

- Spring Batch out-of-the-box propose:
  - Sauter les erreurs non bloquantes (skip)
  - Recommencer un traitement (retry)
  - Déterminer si le batch est fini (completion)
  - Redémarrer un batch (restart)

# Reprise sur erreur





# Reprise sur erreur: Skip

```
0001;ABC;DEF;  
0002;ABC;DEF;  
000zxjgxdjghjsdfkud  
0004;ABC;DEF;
```

- Ne pas arrêter le batch si la lecture/process/écriture échoue
- Personnaliser les cas de « skip »
- Ecouter les cas de « skip »

# Reprise sur erreur: Skip

```
<step id="stepA">
  <tasklet>
    <chunk reader="itemReader" writer="itemWriter"
      commit-interval="100" skip-limit="10">
      <skippable-exception-classes>
        <include class="org.springframework.batch.item.file.FlatFileParseException"/>
      </skippable-exception-classes>
    </chunk>
  </tasklet>
</step>
```

- Si FlatFileException, Spring Batch skip l'item
- Skip de 10 items max
- Au-delà la step « failed »
- Include/Exclude possible

# Reprise sur erreur: Skip

- Un listener permet de traiter les items écartés

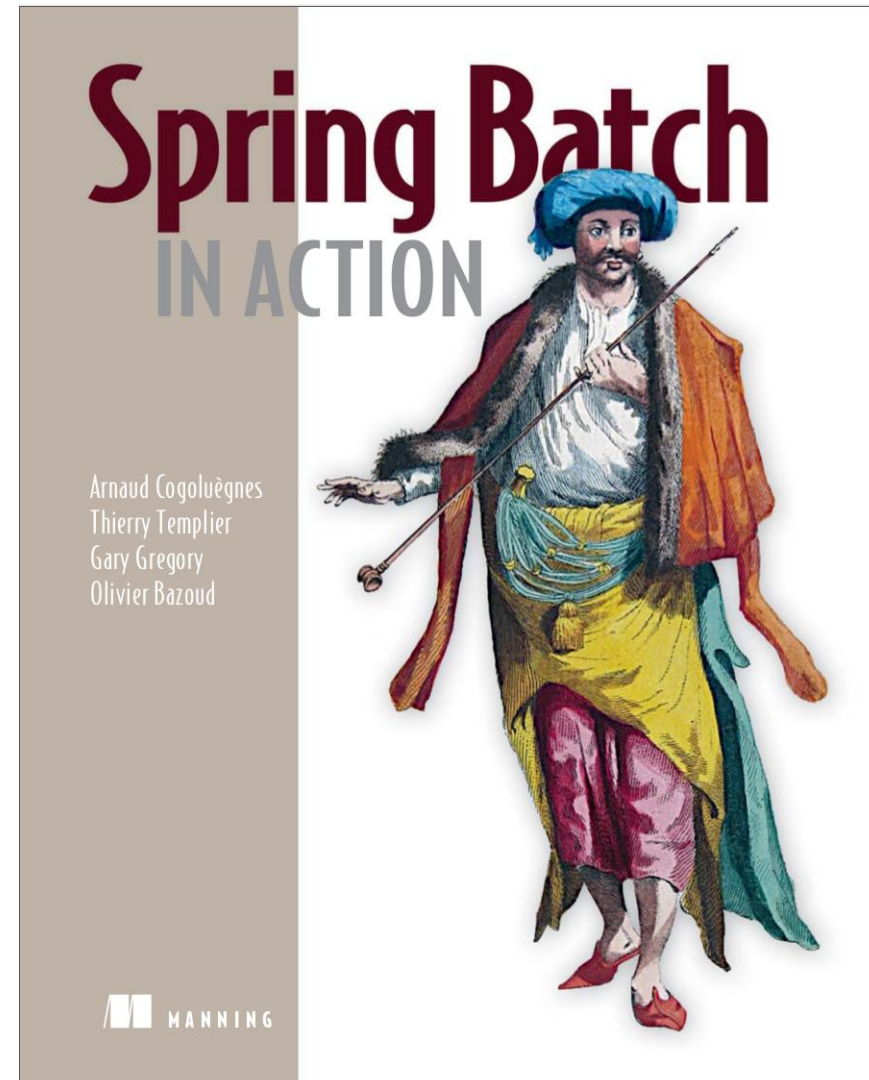
```
public interface SkipListener<T,S> extends StepListener {  
    void onSkipInRead(Throwable t);  
    void onSkipInWrite(S item, Throwable t);  
    void onSkipInProcess(T item, Throwable t);  
}
```

- Les annotations existent aussi
  - @OnSkipInRead
  - @OnSkipInWrite
  - @OnSkipInProcess

# Spring Batch in Action

<http://www.manning.com/templier>

- Arnaud Cogoluègnes
- Thierry Templier
- Gary Grégory
- Olivier Bazoud



# Des questions... avant la suite ?

