Challenges of Offline and Batch Processing

Dave Syer, SpringSource

http://www.springsource.com

spring source

Java

Sun microsystems

Learn about the challenges of increasing throughput in crucial high-volume business environments, and how a range of patterns in the Enterprise Java™ space has emerged to help address the challenges.

GOAL

Agenda

- Offline processing real-life examples
- Available tools, frameworks and platforms
- Definition of a batch, and comparison with event-driven system
- Patterns of resilience and failure
- Challenges for asynchronous offline processing
- Patterns of concurrency and scalability
- Event-driven batch patterns

Offline processing examples

- Close of business processing
  - Order processing
  - Business reporting
  - Account reconciliation
- Import/export handling
  - Instrument/position import
  - Trade/allocation export
  - Data warehouse synchronization
- Large-scale output jobs
  - Loyalty scheme emails
  - Bank statements

Agenda

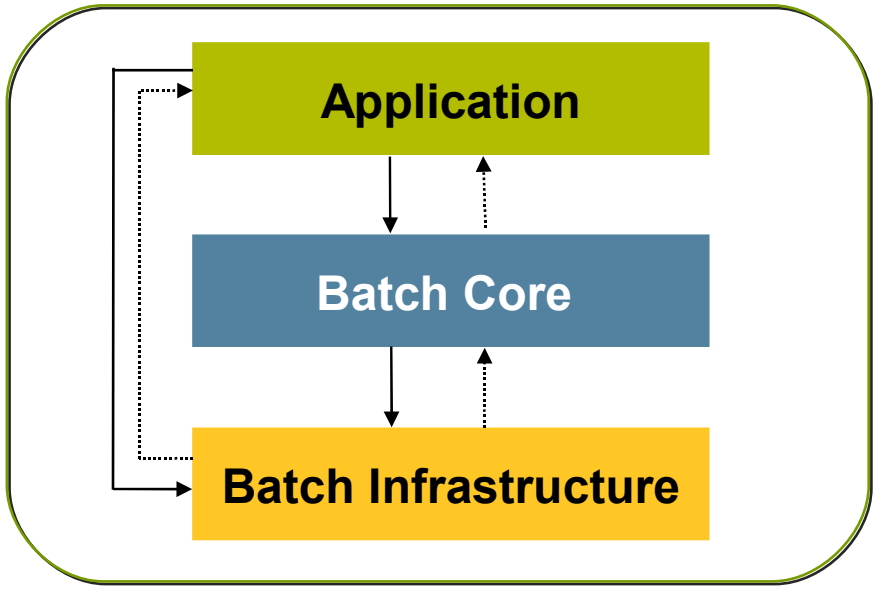
- Offline processing real-life examples
- Available tools, frameworks and platforms
- Definition of a batch, and comparison with event-driven system
- Patterns of resilience and failure
- Challenges for asynchronous offline processing
- Patterns of concurrency and scalability
- Event-driven batch patterns

Available tools, frameworks and platforms

- Good set of standard libraries and integration tools
  - JDBC, JTA
  - nio
  - JMS
  - RMI, web services, JAX-WS etc.
- Higher level frameworks
  - Spring Batch, Spring Web Services, Spring Integration, Apache Camel, etc.
- ESB
  - Mule, BEA AquaLogic, IONA Artix, etc.
- XTP and grid computing
  - Gigaspaces, IBM ObjectGrid, Oracle Coherence, GridGain, Appistry etc.

Spring Batch

- The Spring programming model applied to batch processing
- Don't write code that doesn't make you money
- Download http://www.springframework.org/spring-batch



Agenda

- Offline processing real-life examples
- Available tools, frameworks and platforms
- Definition of a batch, and comparison with event-driven system
- Patterns of resilience and failure
- Challenges for asynchronous offline processing
- Patterns of concurrency and scalability
- Event-driven batch patterns

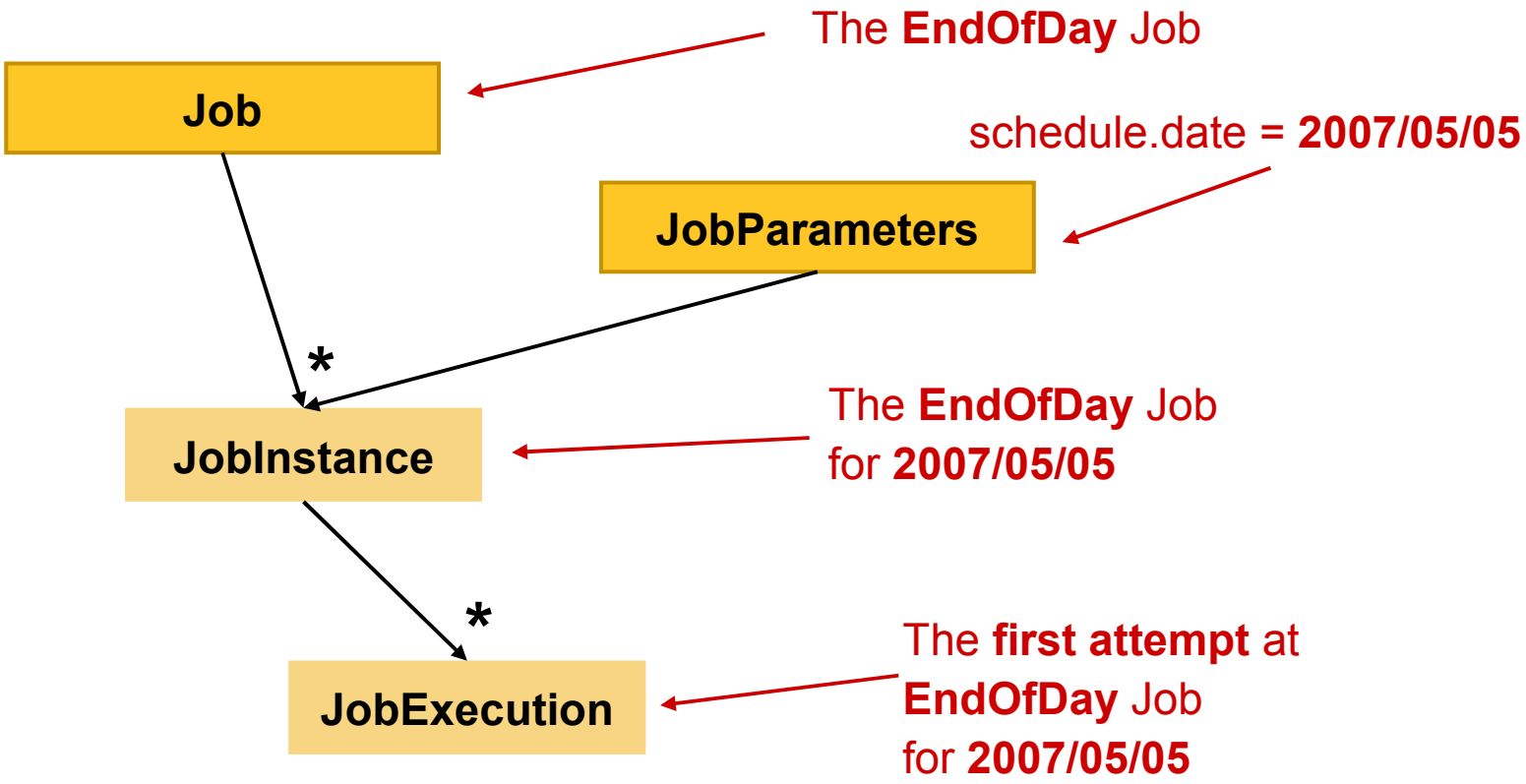
Definition of a Batch

- Batched input - group of items
- Traceable
- Boundary
  - Start
  - End
  - Status
- Optimization of transaction boundaries
  - It's expensive to process one message at a time
  - If you don't have to wait for another message, you might as well process a few at once

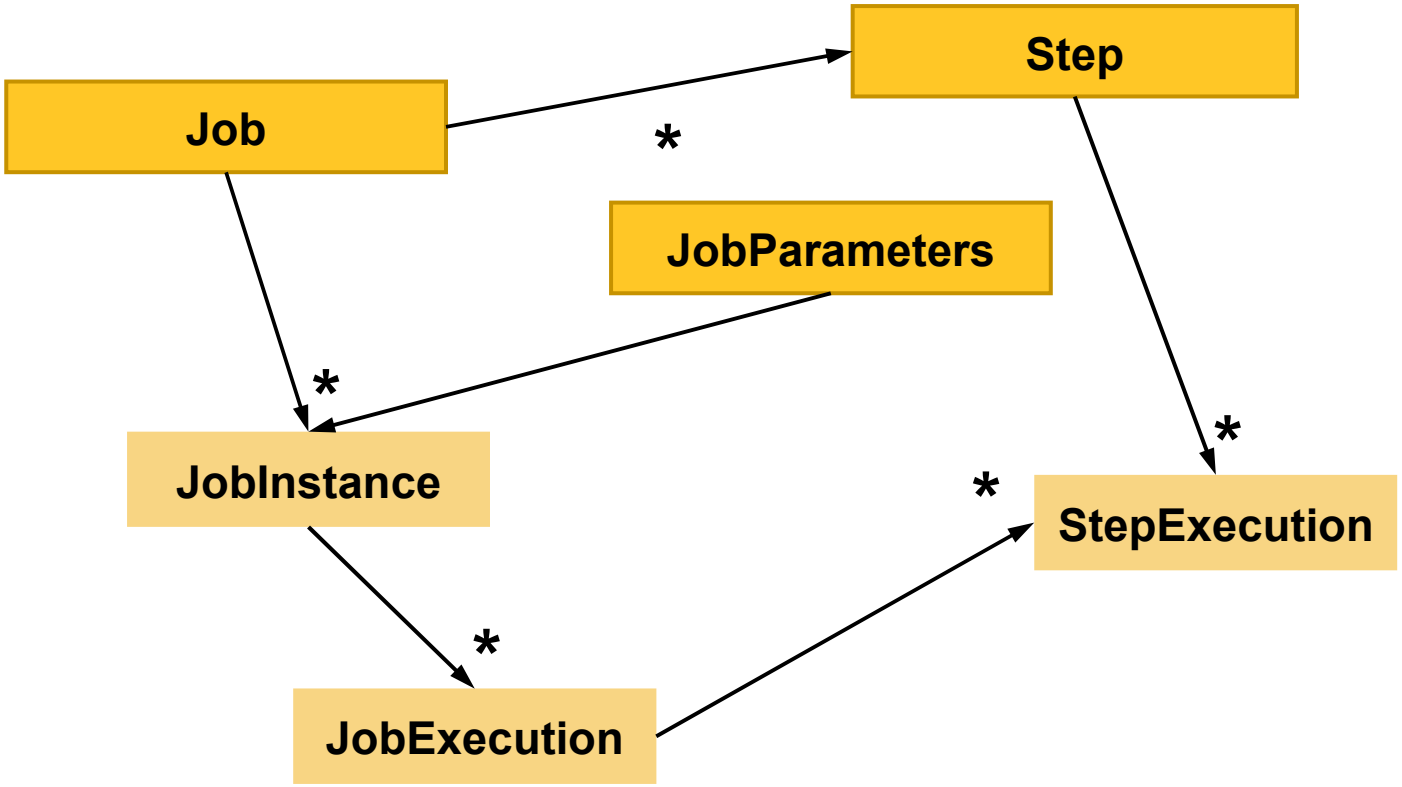
Batch Domain Ubiquitous Language

- Batch infrastructure adds optimization and low level abstractions
- The batch domain adds some value to a plain business process by introducing new concepts:
  - A **job** has an identity – not just a stream of bytes
  - A **job** has **steps**
  - A **job** can be restarted after a failure – a new **execution**
  - Each **execution** has a start time, stop time, status
  - The job has a **status**
  - Each **execution** can tell us how many **items** were processed, how many commits, rollbacks, skips

Job Configuration and Execution

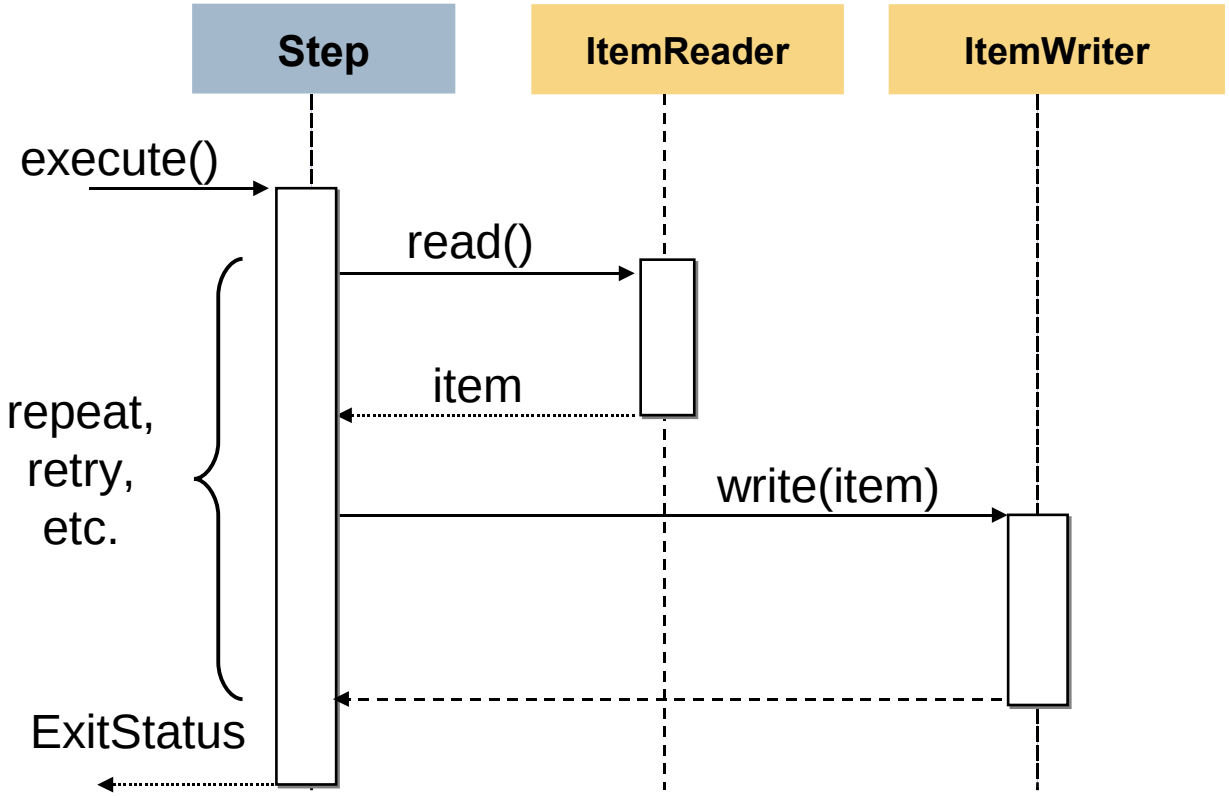


Job and Step


Step
Job
*
JobParameters
*
*
JobInstance
*
StepExecution
*
JobExecution

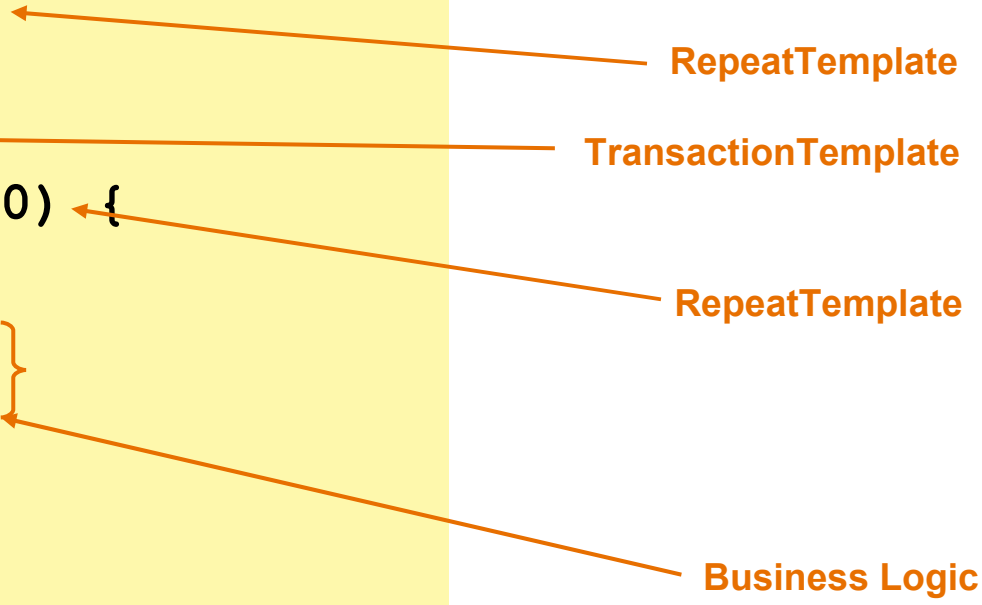
Item-Oriented Processing

- Input-output can be grouped together = Item-Oriented Processing



Item-Oriented Pseudo Code

```
REPEAT(while more input) {

      TX {
            REPEAT(size=500) {

                  input;
                  output;

            }
      }

}
```



Add Batch Domain Concepts

```
JOB {
   STEP {
       REPEAT(while more input) {
              TX {
                     REPEAT(size=500) {
                            input;
                            output;
                     }
              }
       }
   }
}
```

Item-Oriented Processing: Configuration

```
<bean id="loading"
  class="org.sfw...step.SimpleStepFactoryBean">
  <property name="itemReader">
    <bean class="org.sfw..file.FlatFileItemReader">
      <property name="resource" ref="${input.file}" />
    </bean>
  </property>
  <property name="itemWriter">
    <bean class="com...writer.TradeWriter">
      <property name="dao" ref="tradeDao" />
    </bean>
  </property>
</bean>
```

Step configuration

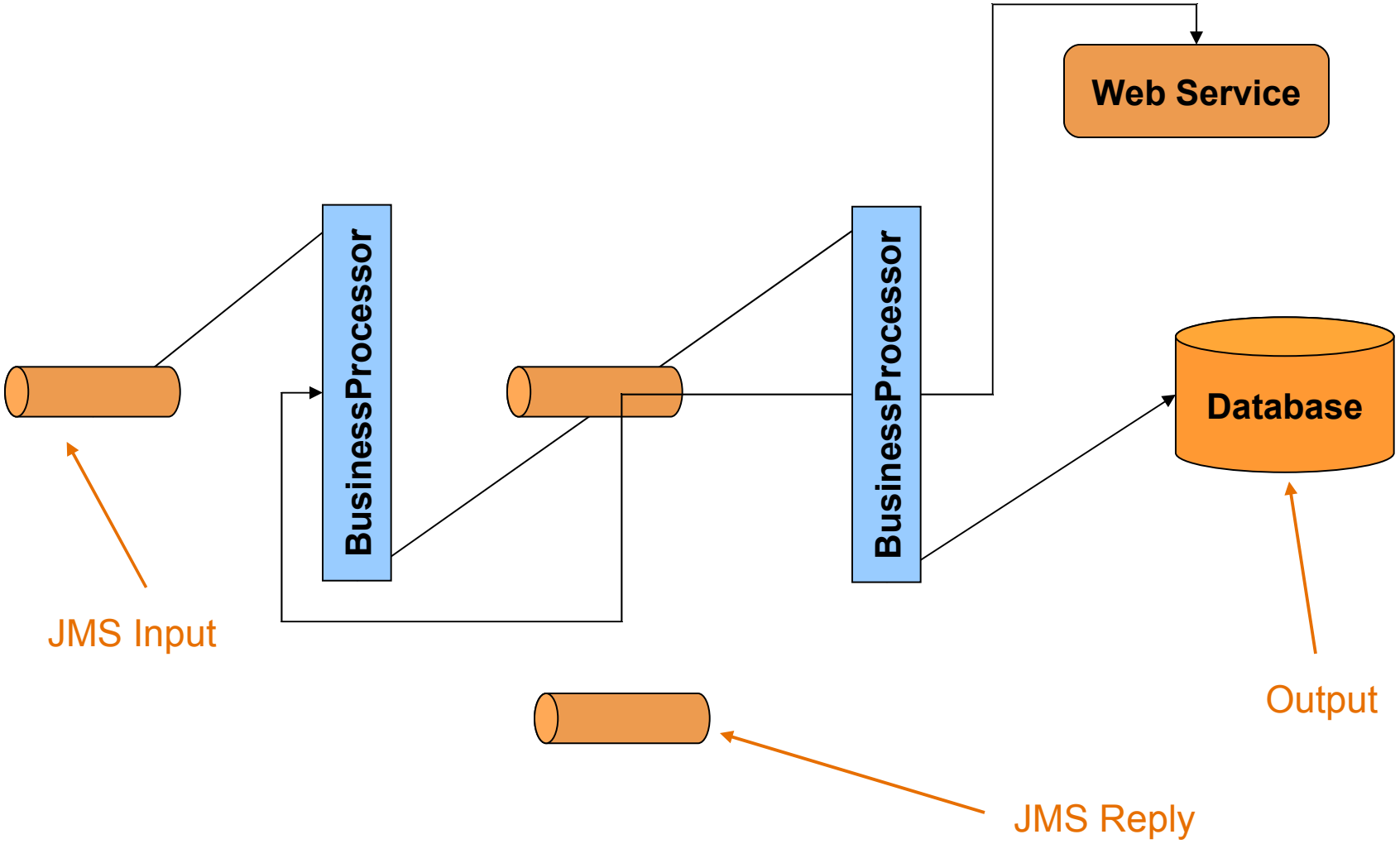
Requires reader and writer

Custom writer re-using online DAO

Event-driven Architecture

- Continuous and concurrent
- Straight through processing
- Sometimes perceived as the opposite of batch processing
  - Is that true?
  - Where are the boundaries?
  - How can they be blurred (technical and business reasons)?
- Analyse concurrent and asynchronous processing problems and solutions
- Leads to Event-driven Batch Processing

Message-driven Pipeline



Agenda

- Offline processing real-life examples
- Available tools, frameworks and platforms
- Definition of a batch, and comparison with event-driven system
- Patterns of resilience and failure
- Challenges for asynchronous offline processing
- Patterns of concurrency and scalability
- Event-driven batch patterns

Processing the Same File Twice...

ComputerWeekly.com

Article from ComputerWeekly.com 4 May 2000

The Department of Social Security has suffered from two major IT failures over the past year. What were the failures, and what action is being taken? *Tony Collins* reports.

Failure 2: Overpayment by BACS

In an unrelated incident, about 112,000 claimants of income support received double their expected payments by automated credit transfer, directly to their bank accounts, when an EDS "Autobacs" batch file was accidentally processed twice.

An irony of the problem was that, having scrapped the Debt Accounting and Management System which was designed to collect overpayment debt, the Department, a few months later, went on to accidentally overpay 112,000 Income Support claimants.

The Department has repeatedly rejected Computer Weekly's requests for a detailed explanation of what went wrong. However staff say the problem began with a bug in a DSS batch programme, which was not intercepted by an EDS "Autobacs" system that collects and reconciles the batch programmes.

Keeping Batch Under Control…

ComputerWeekly.com

Article from ComputerWeekly.com 13 April 2000

Emergency procedures fail the stock exchange

Tony Collins

The failure of systems at the London Stock Exchange last week was due initially to a software bug - as yet still unidentified - but compounded by weaknesses in emergency escalation procedures, *Computer Weekly* has learned.

The problem began with a bug in a non-critical overnight trading systems programme that purges old message logs and the previous day's market data. The batch programme usually takes about an hour to run. In the early hours of Wednesday it took four hours.

This was not a disastrous problem in itself, but all of the exchange's 300 overnight batch programmes must run one after another, and not in tandem. This time, while the first batch programme was still running, a second unrelated batch programme started, for reasons that are not yet clear.

This caused a set of problems that had not been predicted, Chris Broad, the exchange's head of service development, said.

With the two programmes running in tandem, rather than sequentially, data was corrupted. Information from the previous day's trading became mixed with that being prepared for the coming day.

One of the main lessons to be learned from the incident appears to relate to the escalation procedures that involve the system operators and developers. Escalation procedures define the actions that computer operators must take to cope with a potential emergency.

Patterns of Resilience and Failure

- Partial failure and restart
  - Resource management and transaction synchronization
  - Start limit
  - Skip limit
- Automatic retry
  - Retry remote call
  - Retry item processing (transactional)
  - Retry entire job
- Failed item cache
- Flush Policy
  - Binary search for failed item
  - Eager commit on failure

Partial Failure, Retry and Restart

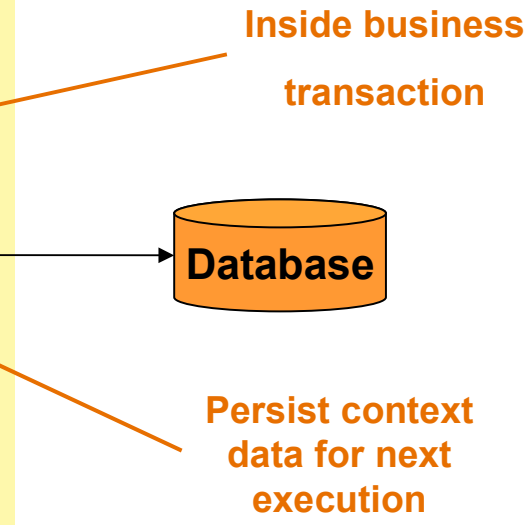
- Stuff happens:
  - Item fails
  - Job fails
- Failures can be
  - Transient – try again and see if you succeed
  - Skippable – ignore it and maybe come back to it later
  - Fatal – need manual intervention
- Mark a job execution as FAILED
- When it restarts, pick up where you left off
- All framework concerns: not business logic

Resource Management Responsibilities

- Open resource
- Close resource at end of step
- Close resource on failure
- Synchronize file with transaction – rollback resets file pointer
- Synchronize cumulative state for restart
  - File pointer, cursor position, processed keys, etc.
  - Statistics: number of items processed, etc.
- Other special considerations
  - Flush policy
  - Footers, checksums, aggregation

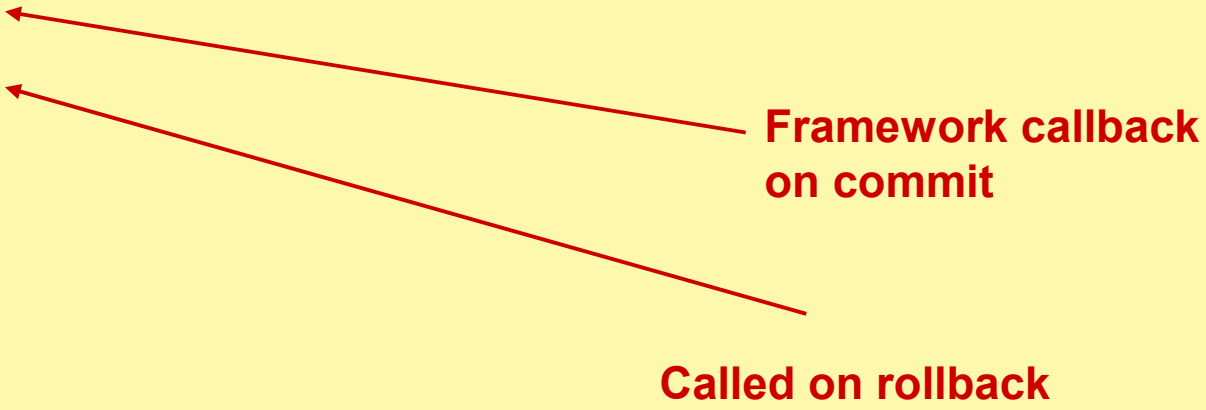
Partial Failure: Piggyback the Business Transaction

```
JOB {
   STEP {
       REPEAT(while more input) {
              TX {
                     REPEAT(size=500) {
                            input;
                            output;
                     }
                     FLUSH and UPDATE;
              }
       }
   }
}
```



Flushing: ItemWriter

```
public interface ItemWriter {

  void write(Object item);
  void flush();
  void clear();

}
```



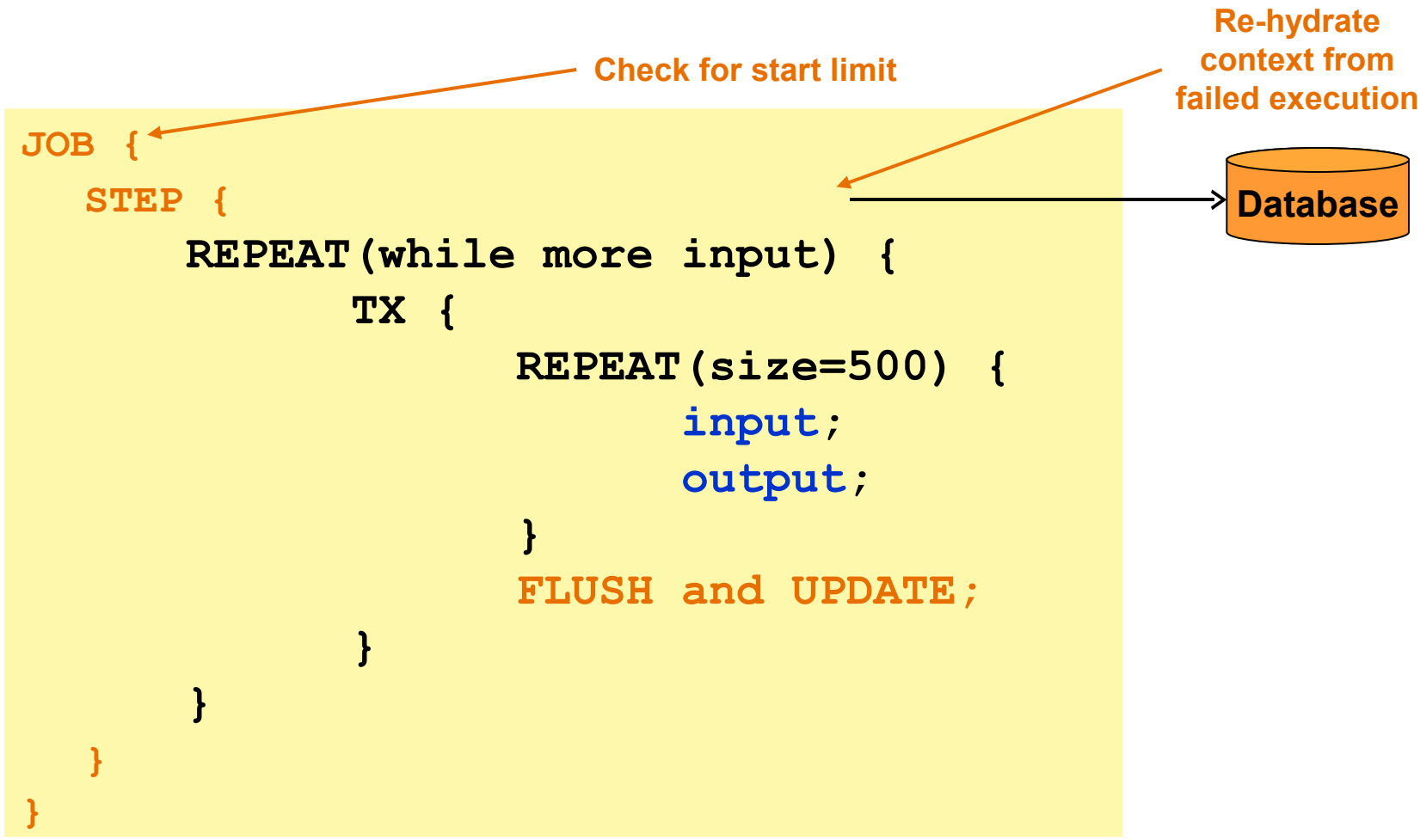
Restartability: ItemStream

```
public interface ItemStream {

  void open(ExecutionContext context);
  void close(ExecutionContext context);
  void update(ExecutionContext context);

}
```

Context from previous (failed) execution

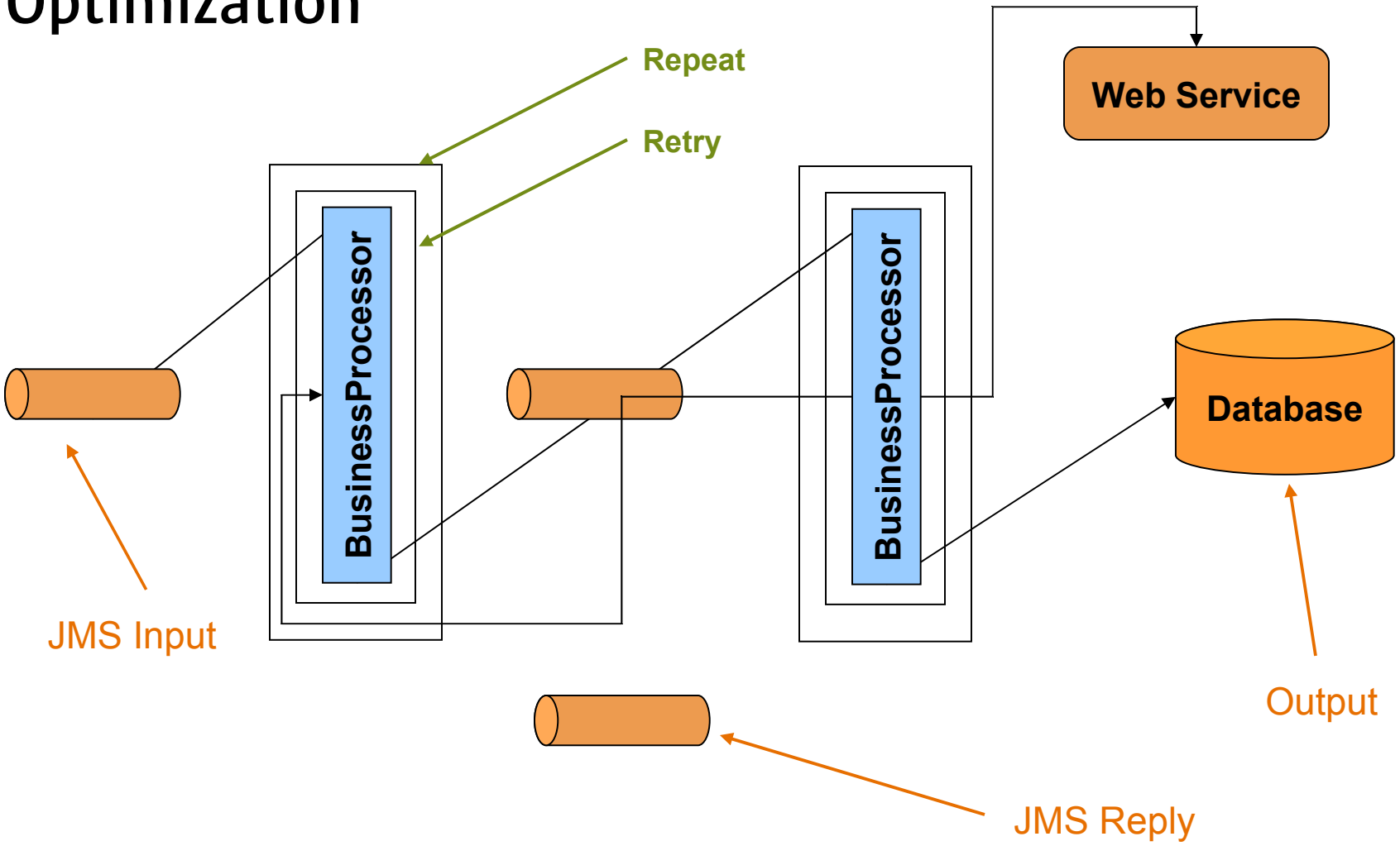
Framework callback before commit

Restart and Start Limit




```
JOB {
   STEP {
       REPEAT(while more input) {
              TX {
                     REPEAT(size=500) {
                            input;
                            output;
                     }
                     FLUSH and UPDATE;
              }
       }
   }
}
```

Automatic Retry: Message-driven Pipeline Optimization



Node 2: Business Logic

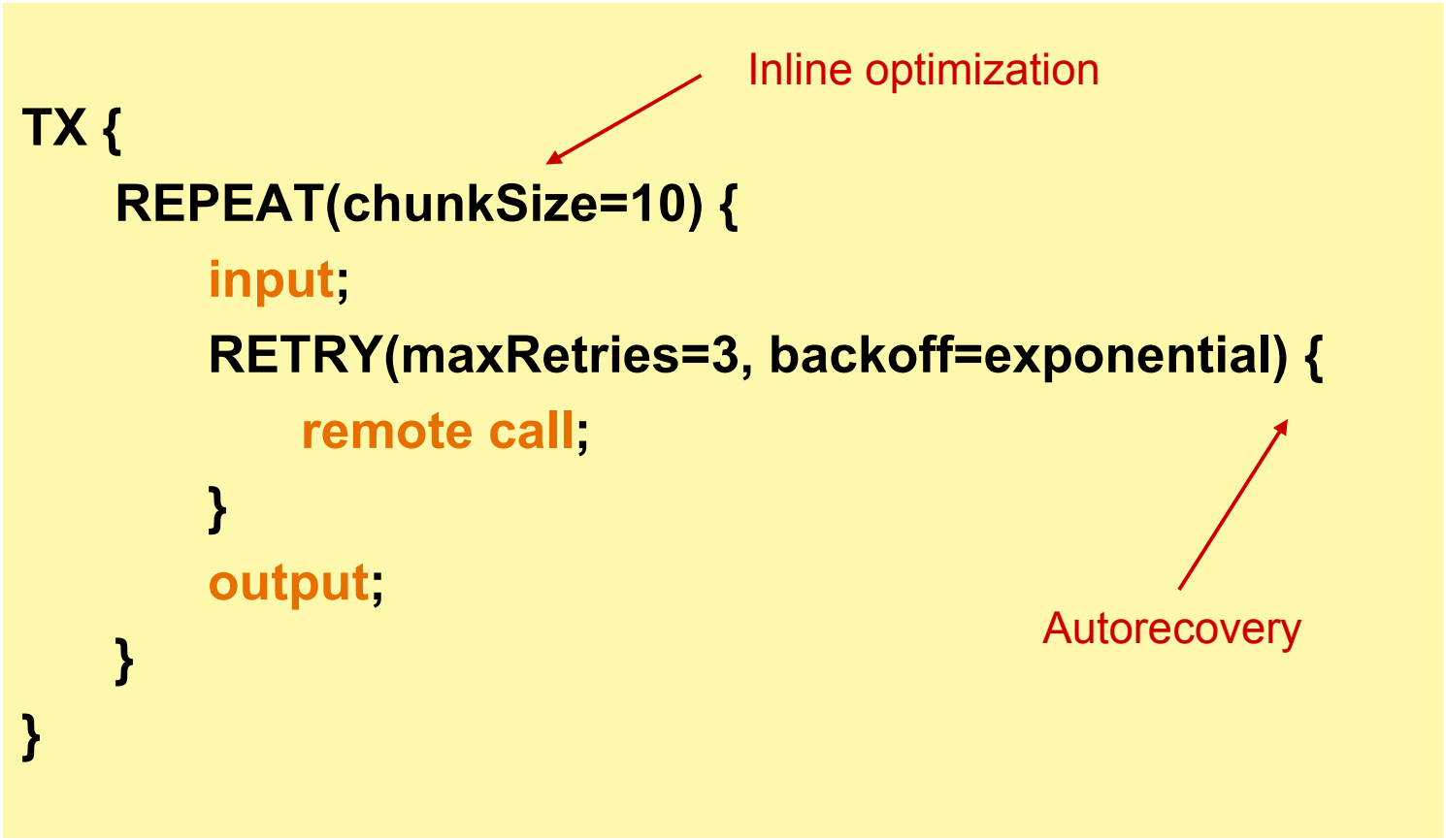
```
input;
remote call;
output;
```

Node 2: Add Transaction Boundary

```
TX {
    input;
    remote call;
    output;
}
```

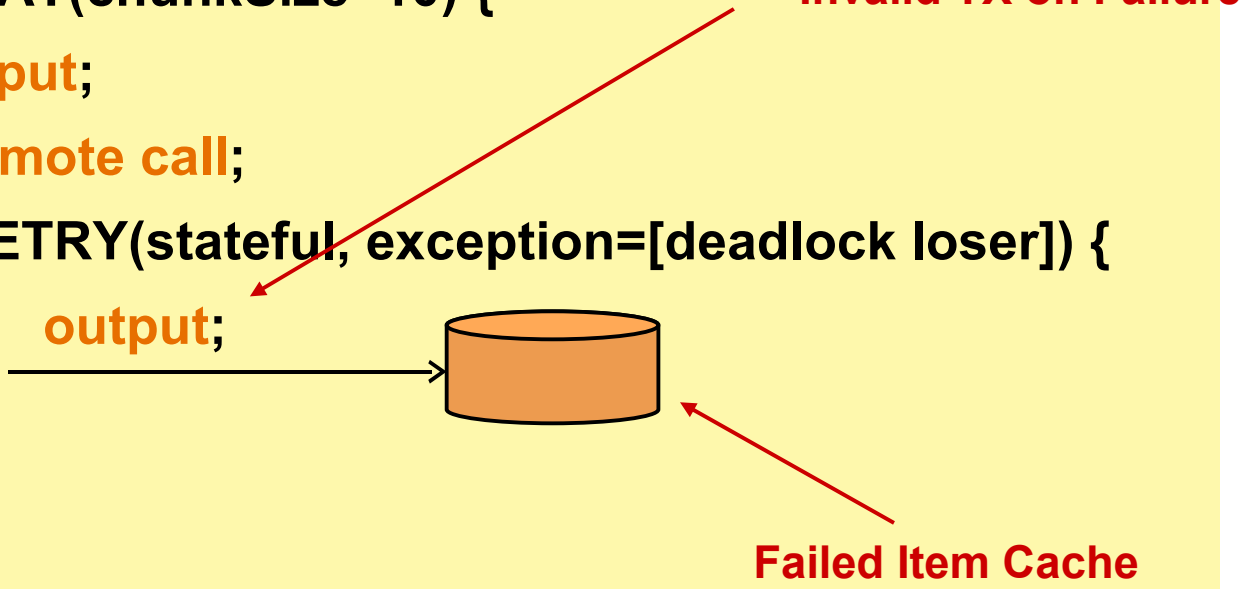
Normal Spring declarative transaction boundary

Node 2: Optimized With Batch Decorations



Node 2: With Stateful Retry on Output

```
TX {
    REPEAT(chunkSize=10) {
        input;
        remote call;
        RETRY(stateful, exception=[deadlock loser]) {
            output;
        }
    }
}
```



Flush Policy

- Flushing allows optimization
- First problem
  - If we do not flush manually, transaction manager handles automatically
  - …but exception comes from inside transaction manager, so cannot be caught and analysed by Step
  - Solution: flush manually on chunk boundaries and catch exception
- Second problem
  - Errors cannot be associated with individual item
  - Two alternatives
    - Binary search through chunk looking for failure = O(logN)
    - Aggressively flush after each item = O(N)
- HibernateAwareItemWriter, BatchSqlUpdateItemWriter

Agenda

- Offline processing real-life examples
- Available tools, frameworks and platforms
- Definition of a batch, and comparison with event-driven system
- Patterns of resilience and failure
- Challenges for asynchronous offline processing
- Patterns of concurrency and scalability
- Event-driven batch patterns

Challenges for Asynchronous Processing

- Improve throughput
  - Amdahl's Law
- Failure patterns
  - What is a rollback?
  - Tracing failed items, or failed chunks
  - What should happen when a job restarts after a failure?
- Manageability
  - What are you launching?
  - What are its dependencies?
  - What are you monitoring?

Agenda

- Offline processing real-life examples
- Available tools, frameworks and platforms
- Definition of a batch, and comparison with event-driven system
- Patterns of resilience and failure
- Challenges for asynchronous offline processing
- Patterns of concurrency and scalability
- Event-driven batch patterns

Patterns of Concurrency and Scalability

- Process Indicator
- Item-oriented middleware
- Chunk-oriented middleware
- Compute and data grids
- Partitioning

Process Indicator

- Driving Query:

```
Select ID from transaction
where status = 'unsent';
```

- Pre-job:

ID	Details	Status
1	adshfag	unsent
2	asdfljkh	unsent
3	elkjrthl	unsent
4	oliusbohu	unsent
5	profiusep	unsent

- This job puts each record in a file and marks is as “sent” inside the business transaction (typically on read)

Process Indicator: Implementation

```
public class StagingItemReader implements ItemReader {

  public Object read() {

    Long id = getNextId();

    jdbcTemplate.update("UPDATE TRANSACTION " +
      "SET STATUS='sent' where ID=?", id);

    return id;

  }

}
```

Buffers ids in transaction

If business transaction fails this change rolls back

Process Indicator

- On the 3rd record we hit a data contention condition and fail
- The driving query filters out the records that were already processed:

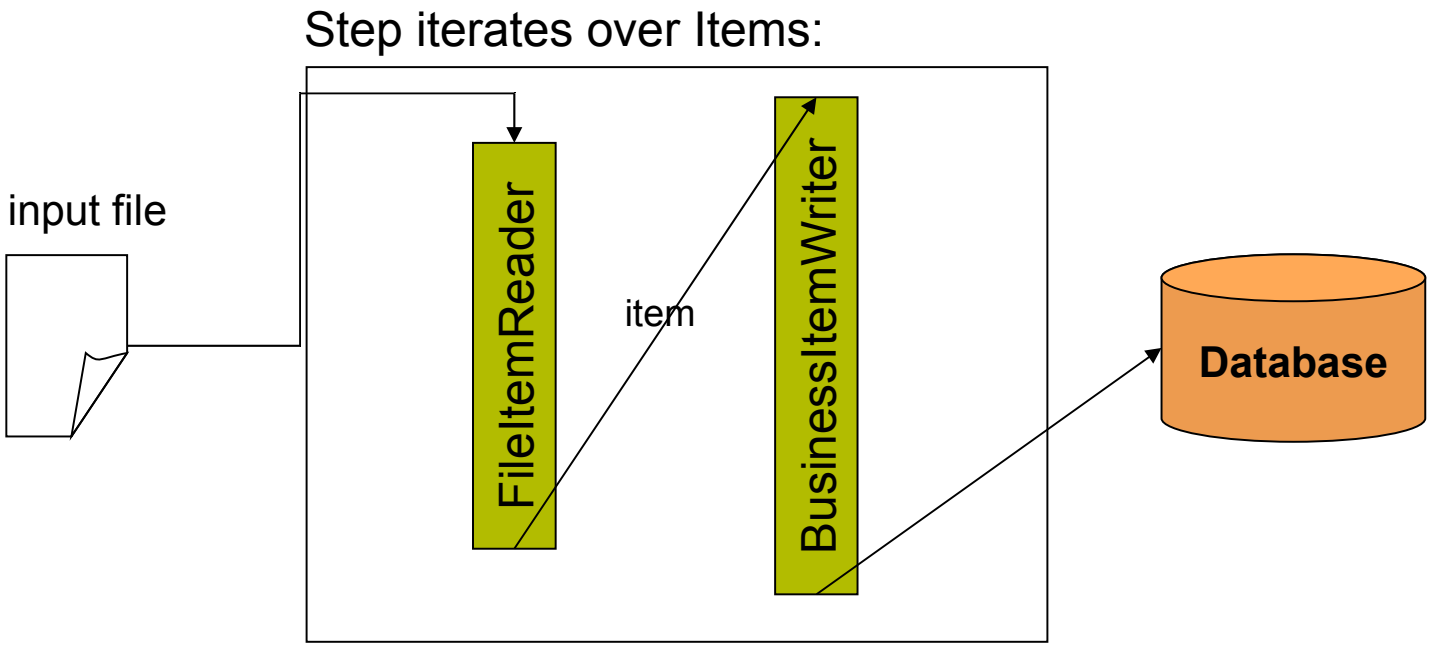
ID	Details	Status
1	adshfag	sent
2	asdfljkh	sent
3	elkjrthl	unsent
4	oliusbohu	unsent
5	profiusep	unsent

Start Here →

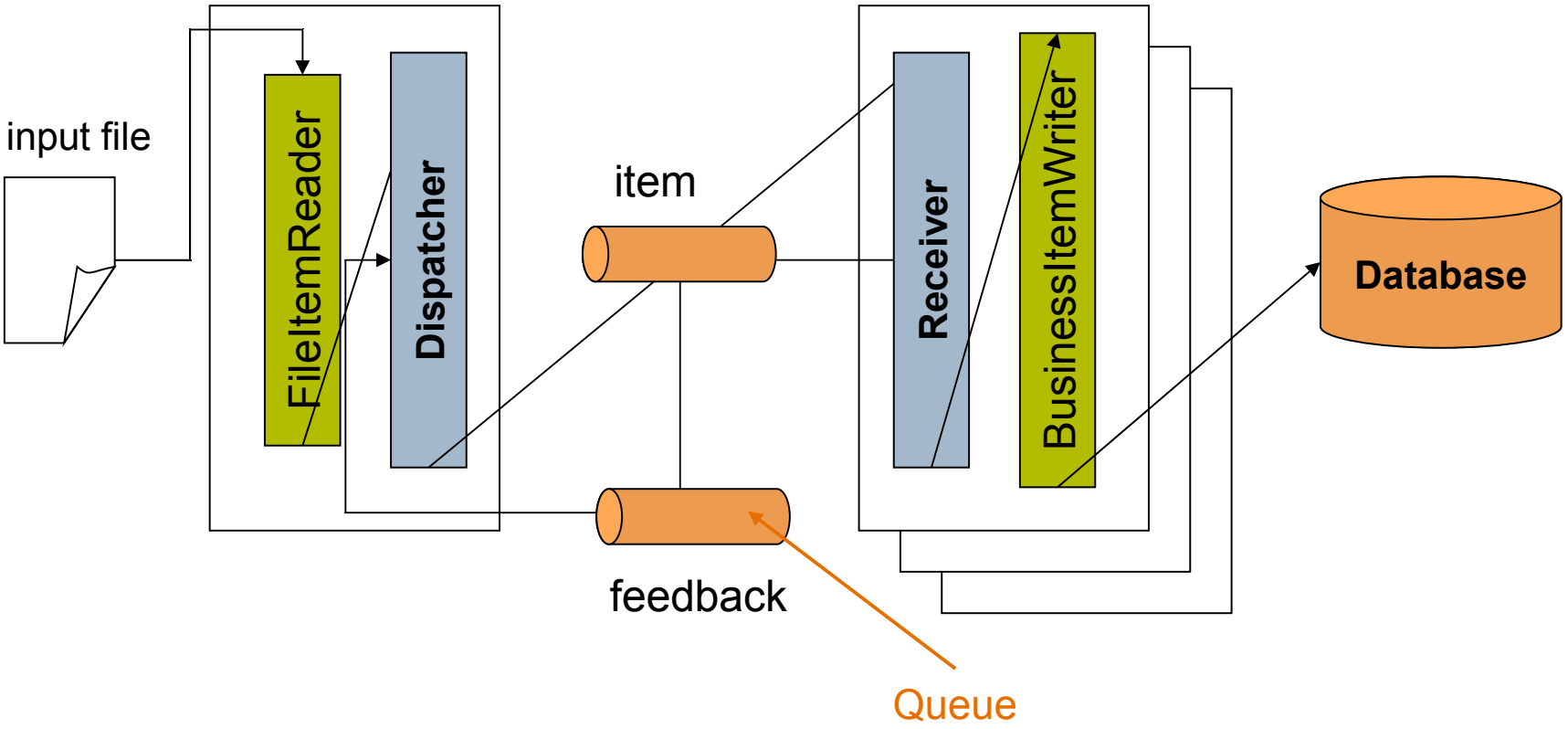
- The top two records are excluded from the result set because they no longer meet the condition of:

```
where status = 'unsent';
```

Item-Oriented Processing (Recap)



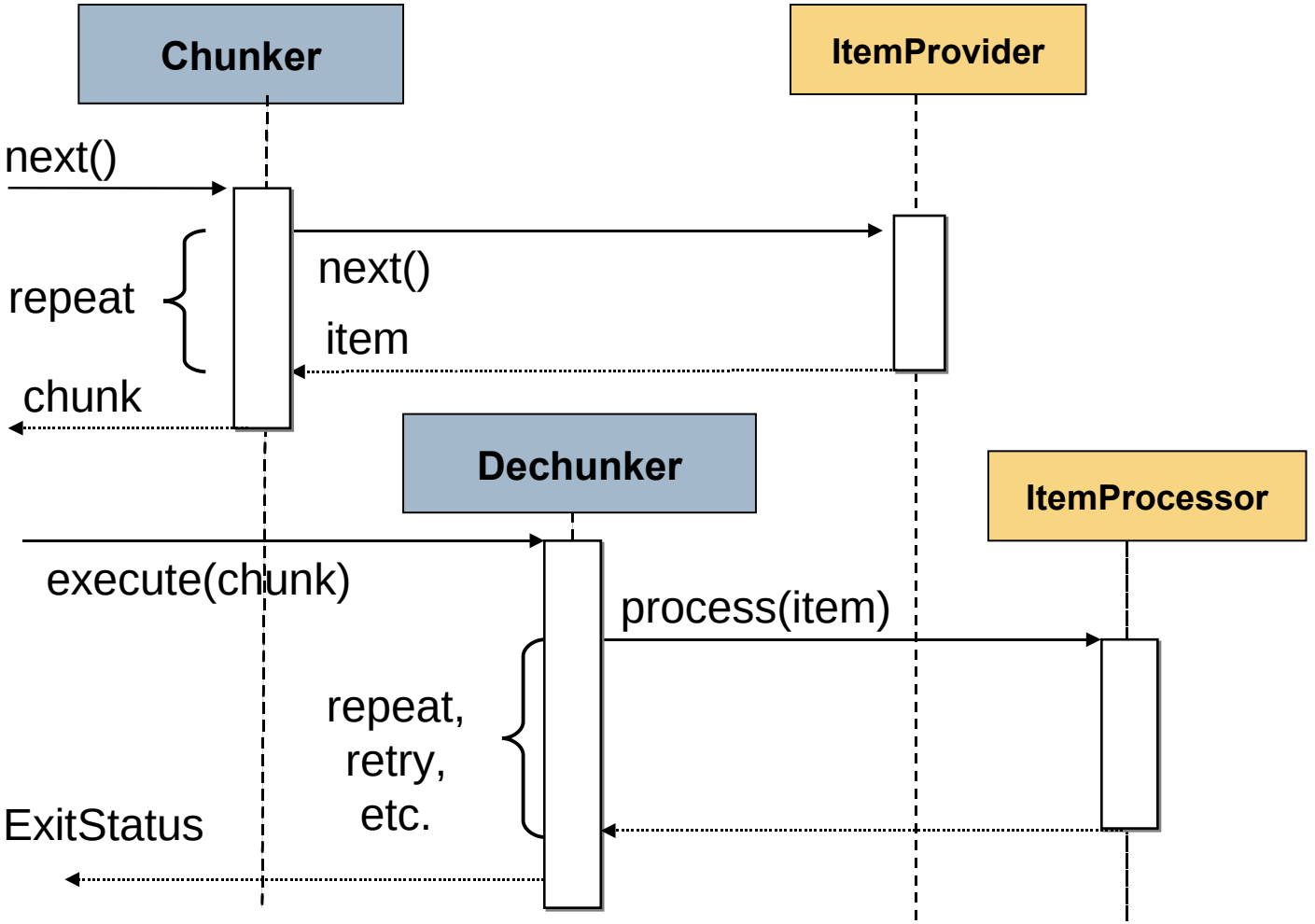
Item-Oriented Middleware



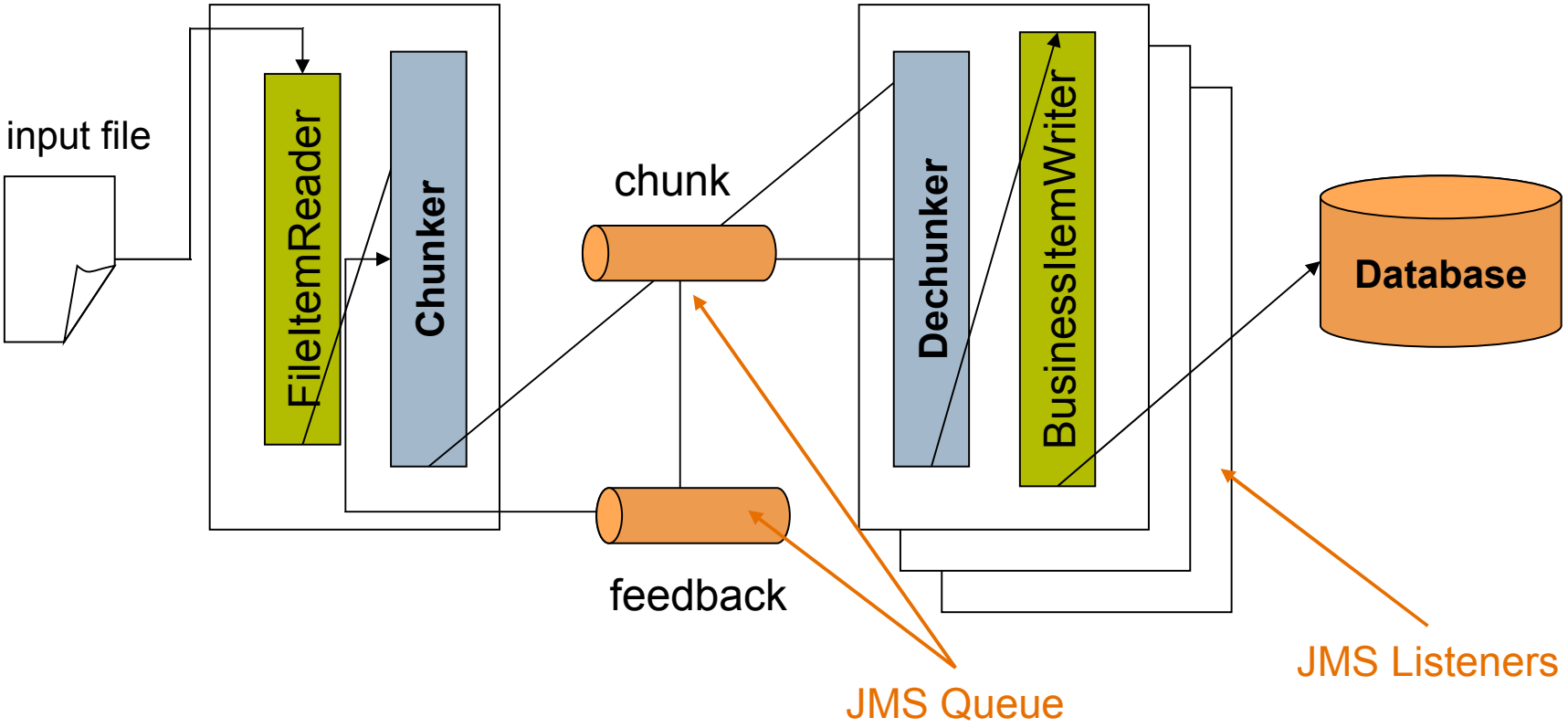
Chunk-Oriented Pseudo Code

```
REPEAT(while more input) {
   chunk = ACCUMULATE(size=500) {
      input;
   }
   RETRY {
      TX {
         for (item : chunk) { output; }
      }
   }
}
```

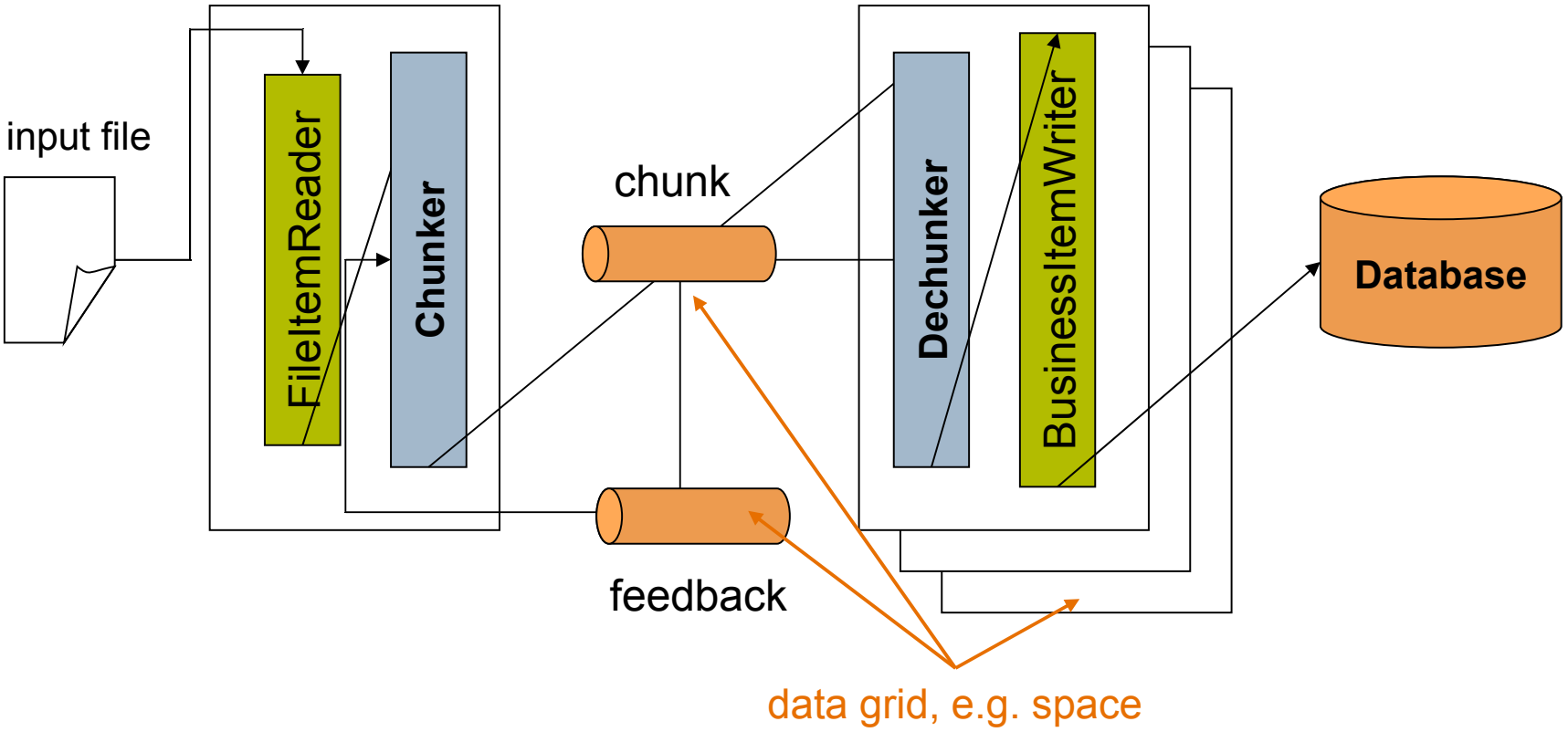
Chunk-Oriented Processing



Chunk-Oriented Middleware

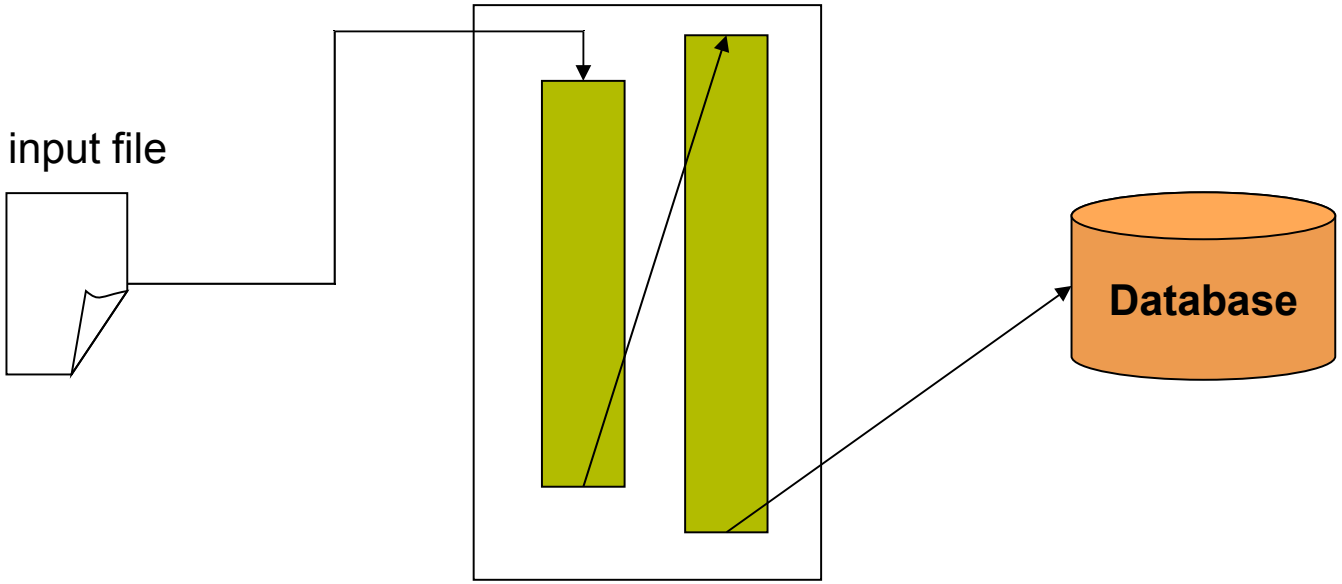


Grid-based Computing



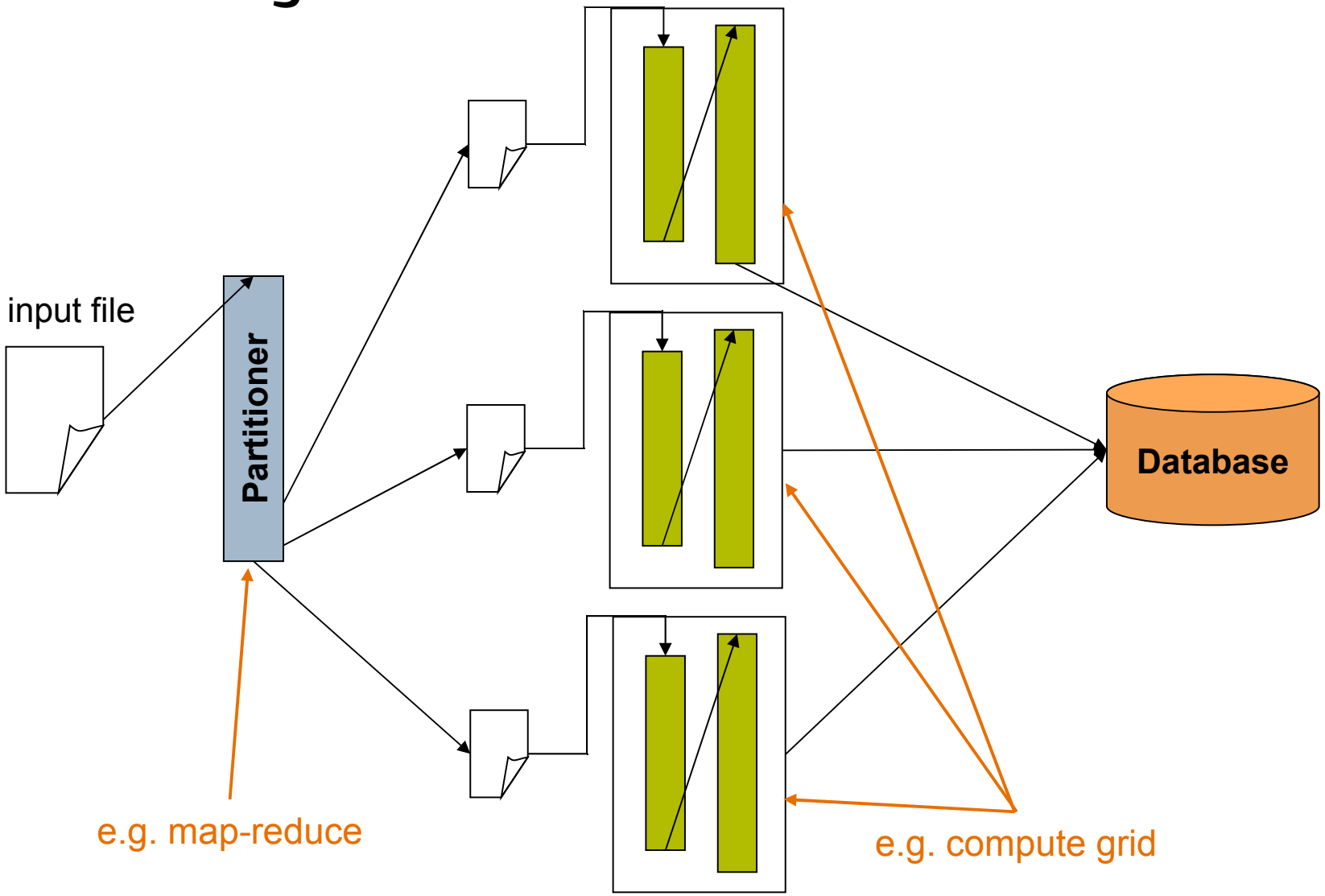
Partitioning

- Take a job…



- …and split it up…

Partitioning

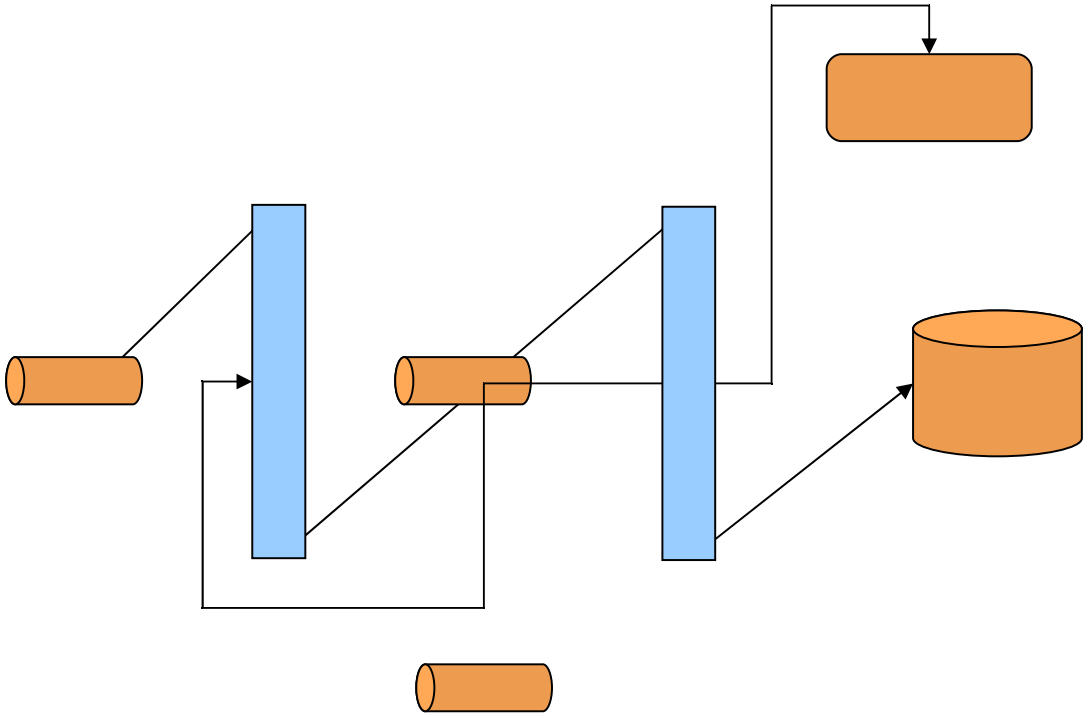


Agenda

- Offline processing real-life examples
- Available tools, frameworks and platforms
- Definition of a batch, and comparison with event-driven system
- Patterns of resilience and failure
- Challenges for asynchronous offline processing
- Patterns of concurrency and scalability
- Event-driven batch patterns

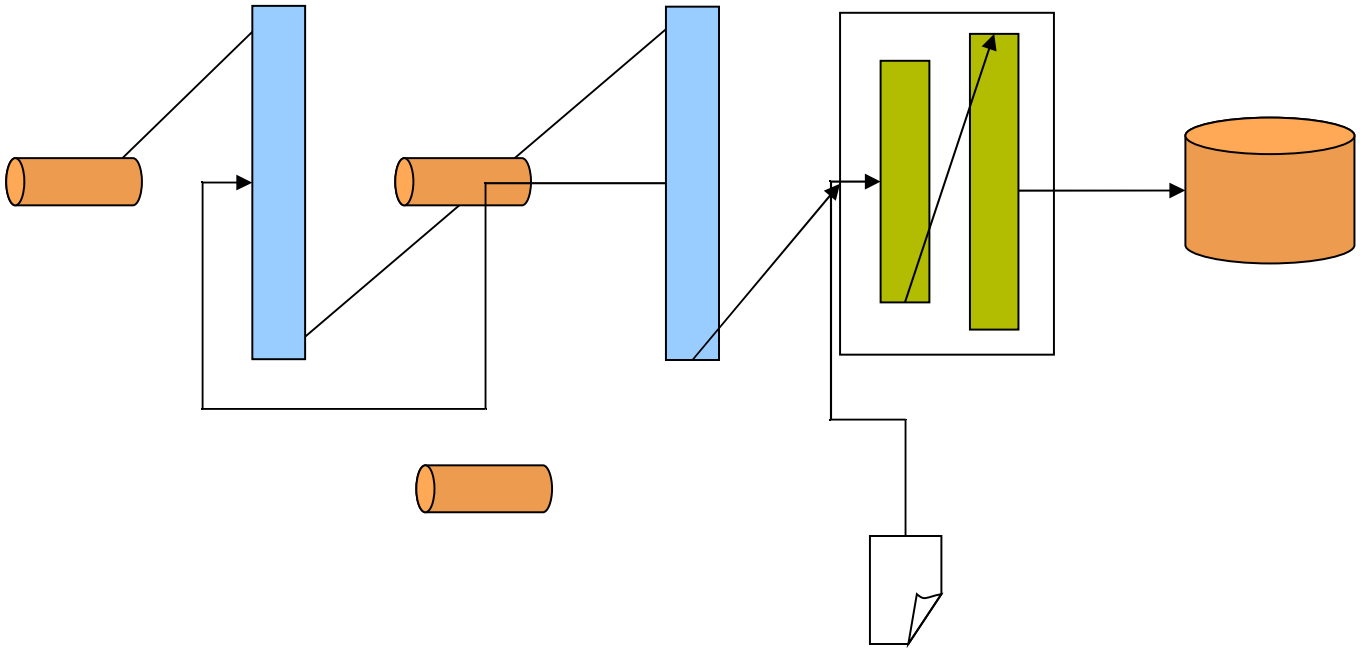
Event-driven Batch

- Recap: start with “normal” message pipeline



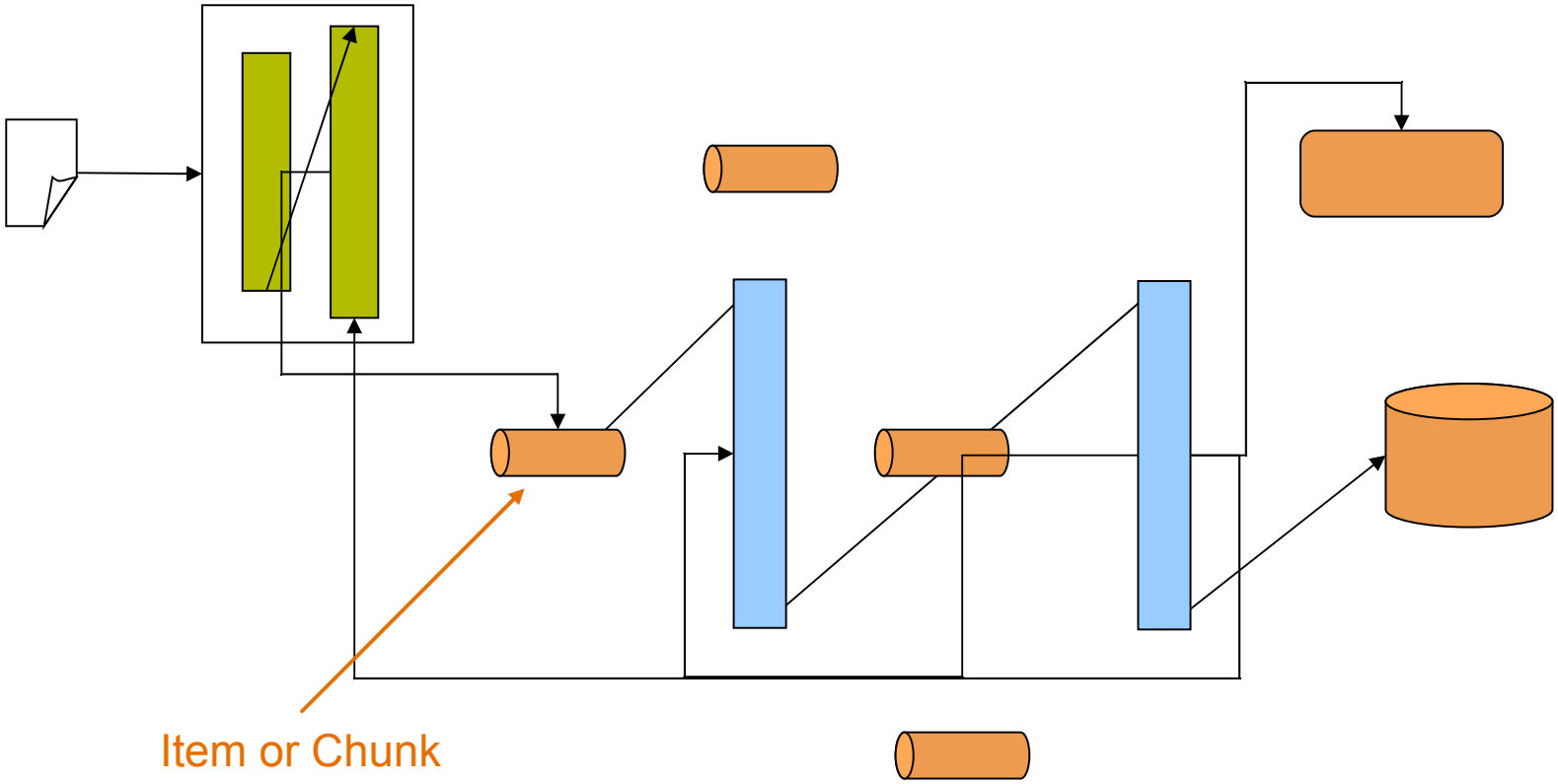
Pattern: Message Trigger

- Event in “normal” message pipeline triggers batch job



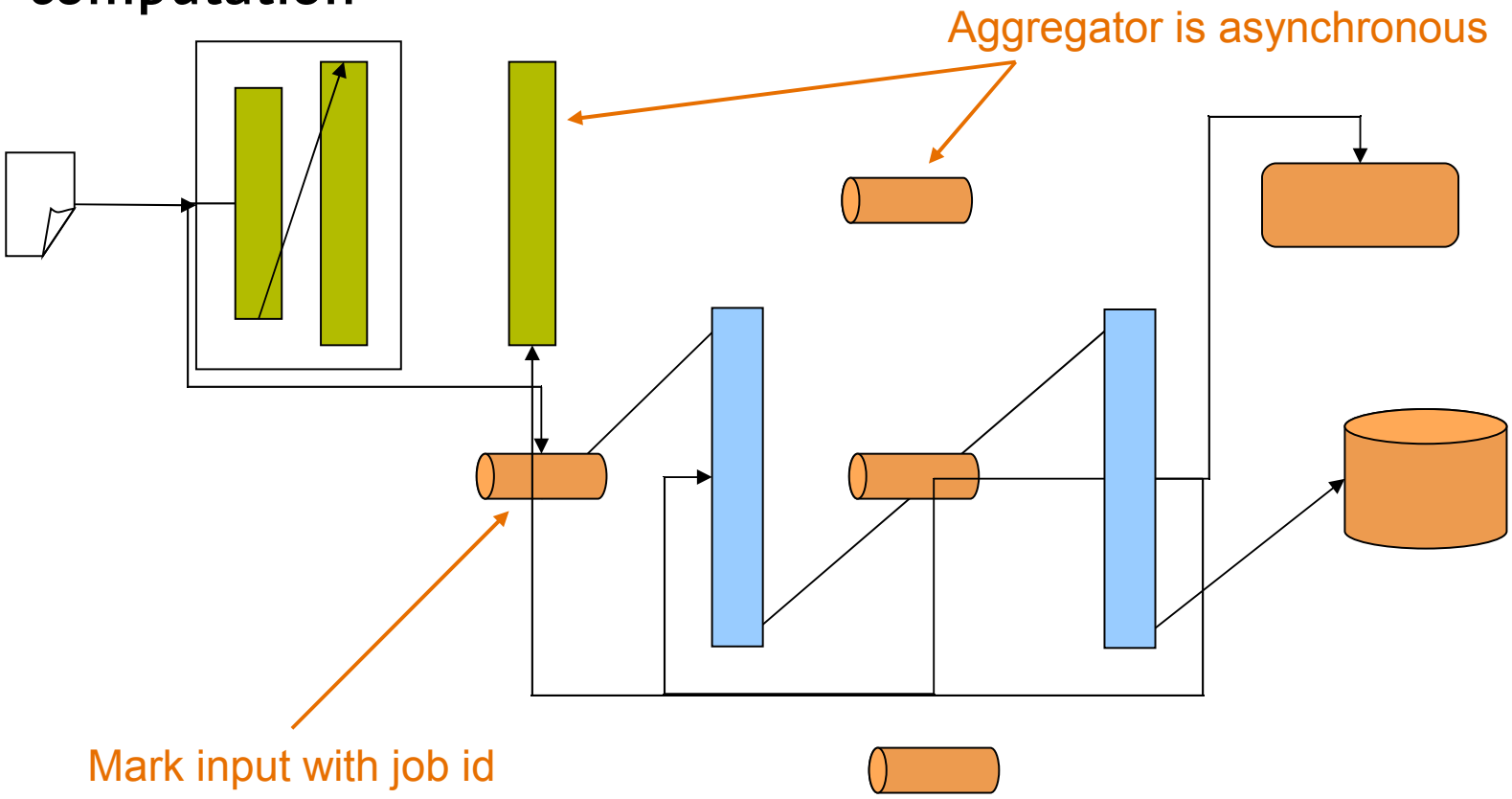
Pattern: Embedded Messaging

- Batch job implemented as message-oriented process



Pattern: Asynchronous Aggregator

- Batch job as emergent process from message- or grid-based computation



Summary

- Offline processing real-life examples
- Available tools, frameworks and platforms
- Definition of a batch, and comparison with event-driven system
- Patterns of resilience and failure
- Challenges for asynchronous offline processing
- Patterns of concurrency and scalability
- Event-driven batch patterns

THANK YOU

Dave Syer, SpringSource.

Challenges of Offline Processing.