JSR 352 Expert Group

Working Session

7 March 2012

Agenda

- Review: Lifecycle Events vs Listeners
- Review: ~~Concurrency~~ Parallel Models
- Discussion: Repeat, Retry, Skip
- List for Next Meeting

Review: Lifecycle Events vs Listeners

From public mailing list, Proposed lifecycle events per batch artifact:

@Job
- @BeginJob
- @EndJob

@Step
- @BeginStep
- @Process or @ProcessItem
- @EndStep

@ItemReader
- @Open
- @ReadItem
- @Close
- @GetCheckpointInfo

@ItemWriter
- @Open
- @WriteItem
- @Close
- @GetCheckpointInfo

Review: Lifecycle Events vs Listeners

Proposed listeners:

- JobListener
- StepListener
- CheckpointListener
- SkipListener
- RetryListener

Jury still out on these listeners;

- ItemReadListener
- ItemProcessListener
- ItemWriteListener
- RepeatListener

Review: Lifecycle Events vs Listeners

Listeners and their subordinate annotations:

- @JobListener
  - @BeforeJob
  - @AfterJob
- @StepListener
  - @BeforeJob
  - @AfterJob
- @CheckpointListener
  - @BeforeCheckpoint
  - @AfterCheckpoint
- @SkipListener
  - @OnSkipInRead
  - @OnSkipInProcess
  - @OnSkipInWrite
- @RetryListener
  - @BeforeRetry
  - @AfterRetry

Review: Lifecycle Events vs Listeners

Example JobListener:

```
@JobListener
// Count and record jobs
public class CountJobs {
    @BeforeJob public void before() {...}
    @AfterJob public void after() {...}
}
```

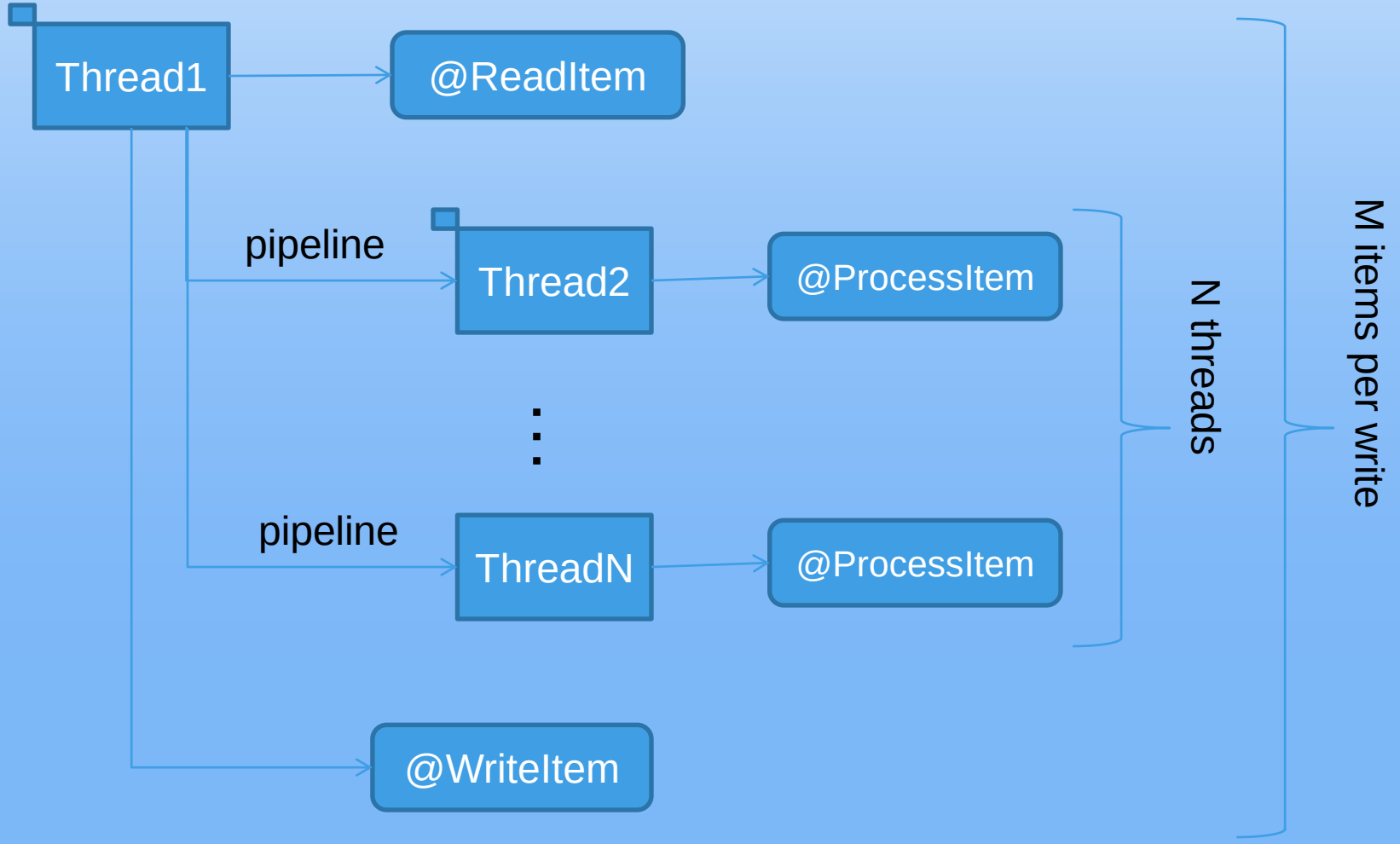
Attach @JobListener to job:

```
@Job(name="Job1")
public class TestJob {
    @JobListener CountJobs jobCounter;
    ....
}
```

The same construction pattern would apply to each listener. Any listener can be attached to any artifact - i.e. job, step, reader, writer.

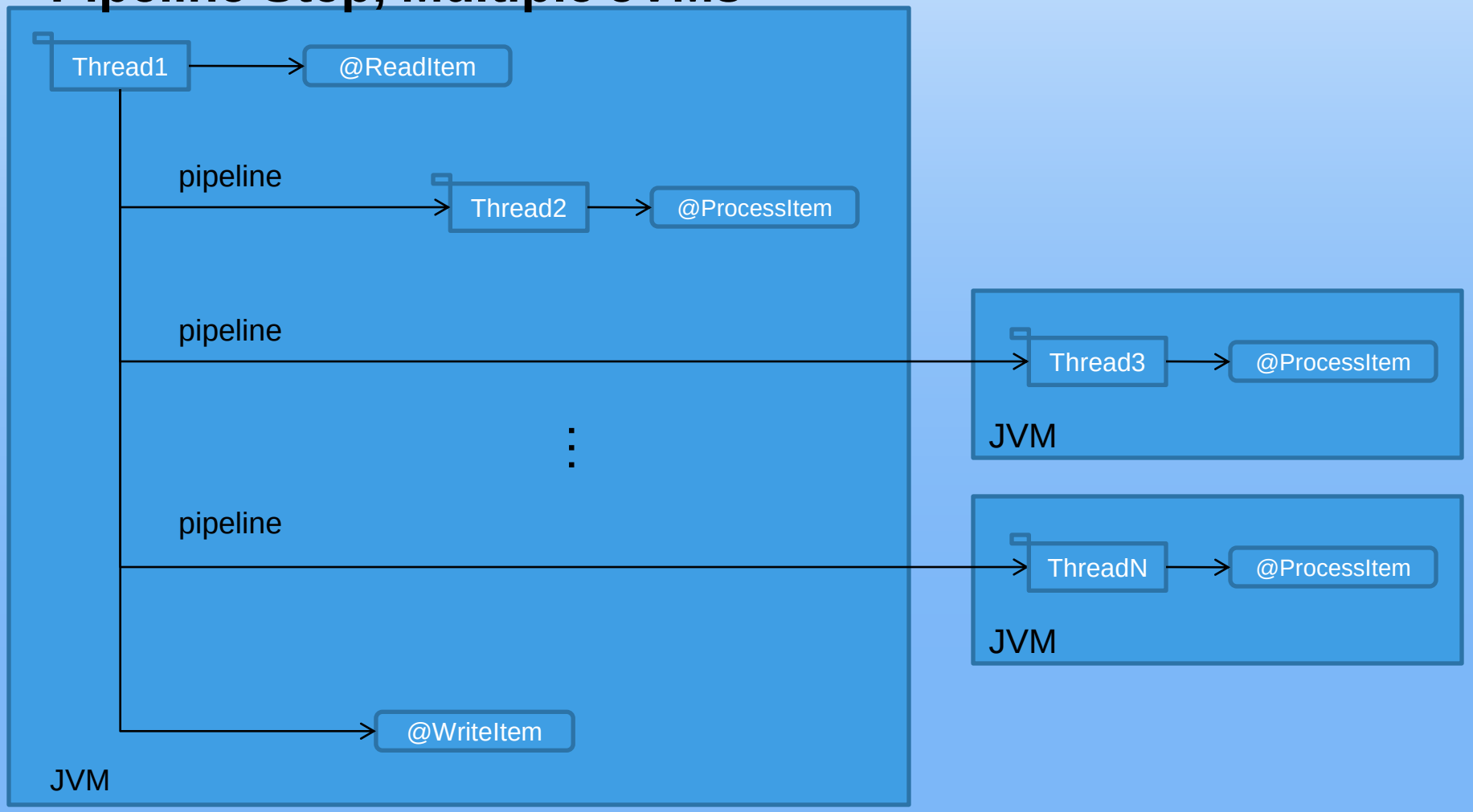
Review: Parallel Models

Pipeline Step, Single JVM



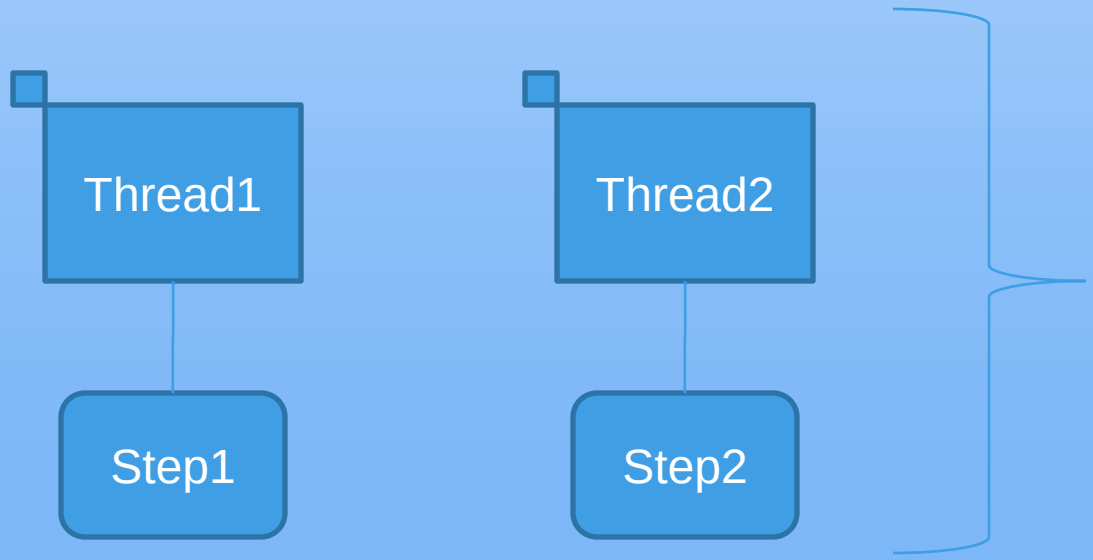
Review: Parallel Models

Pipeline Step, Multiple JVMs



Review: Parallel Models

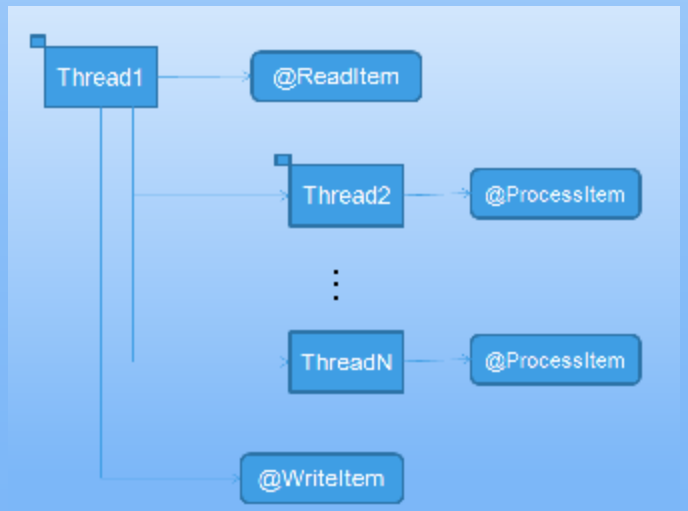
Concurrent Steps, Single JVM



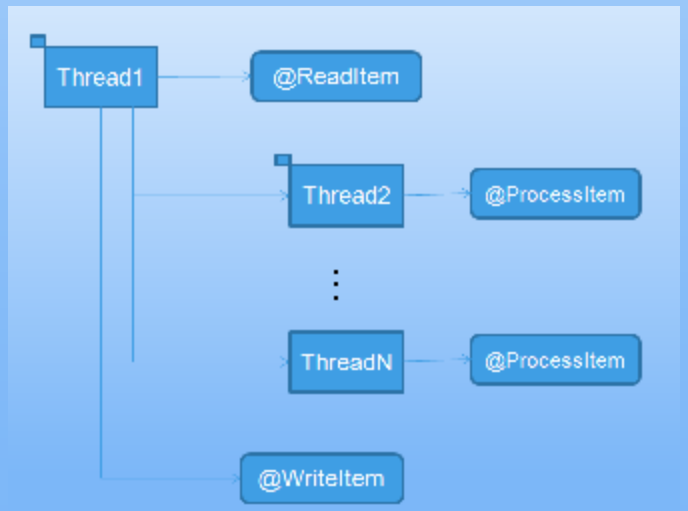
Review: Parallel Models

Concurrent and Pipeline can be combined

Step1

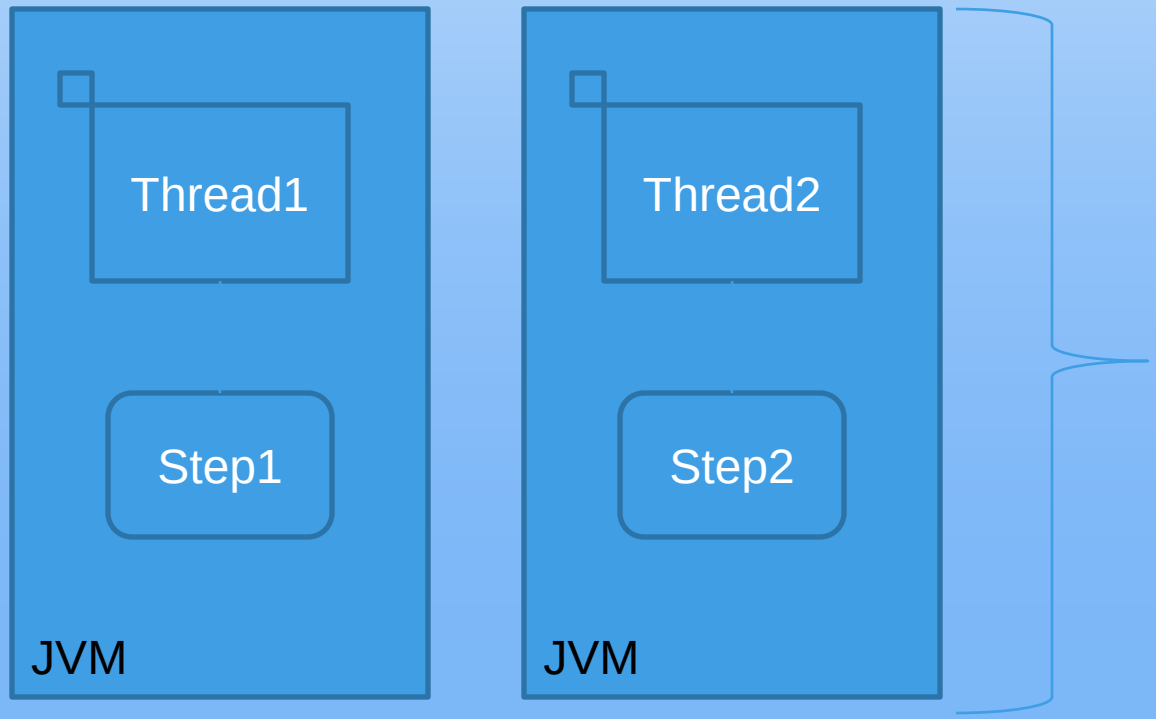


Step2



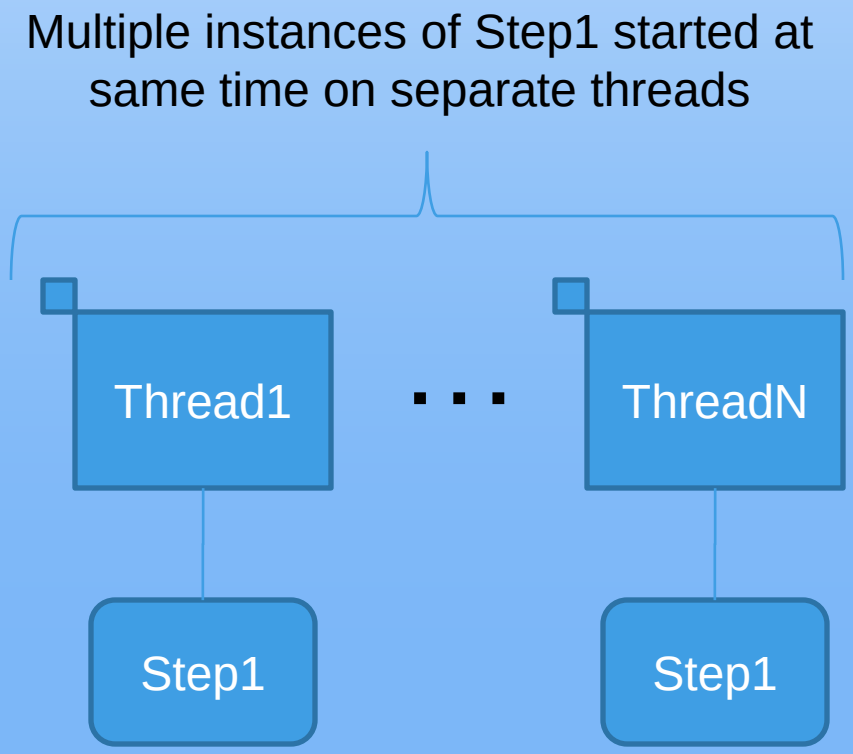
Review: Parallel Models

Concurrent Steps, Multiple JVMs



Review: Parallel Models

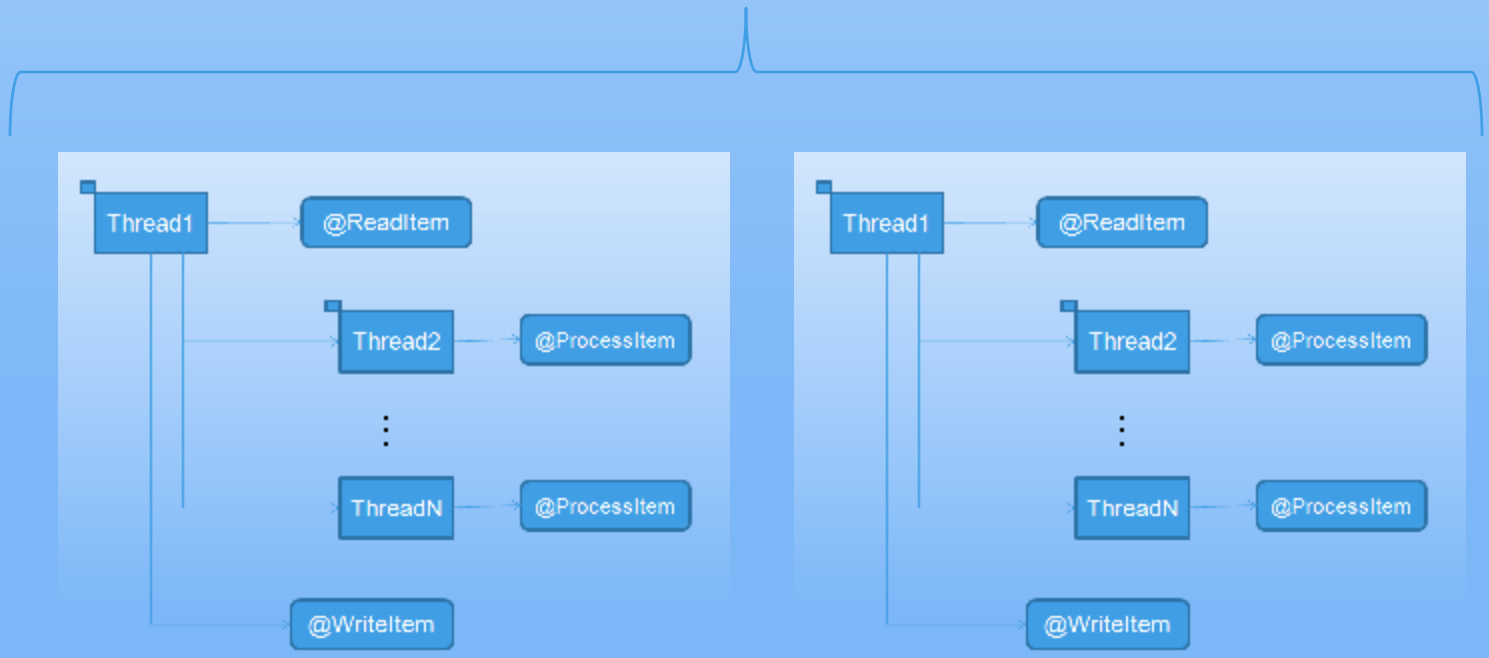
Partitioned Step, Single JVM



Review: Parallel Models

Partitioned and pipeline can be combined.

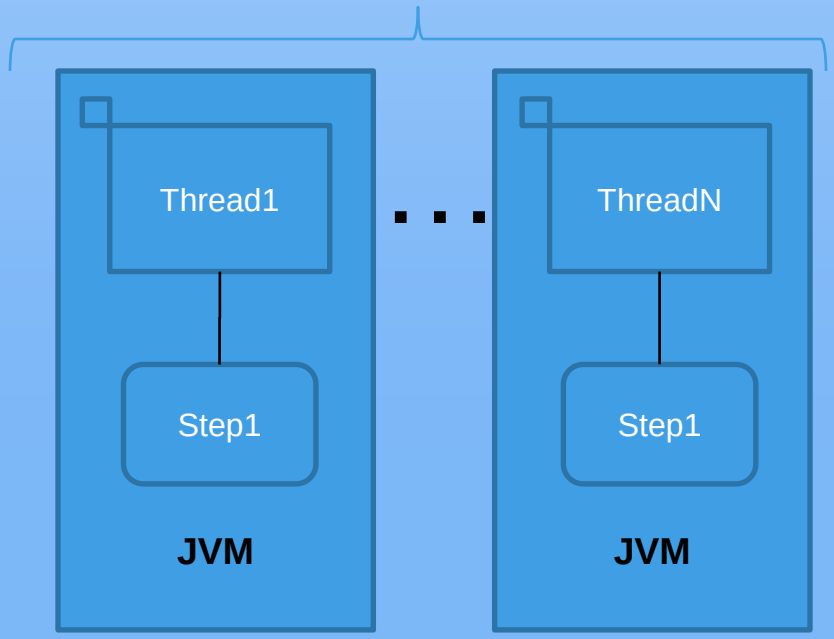
Multiple instances of Step1 started at same time on separate threads



Review: Parallel Models

Partitioned Step, Multiple JVMs

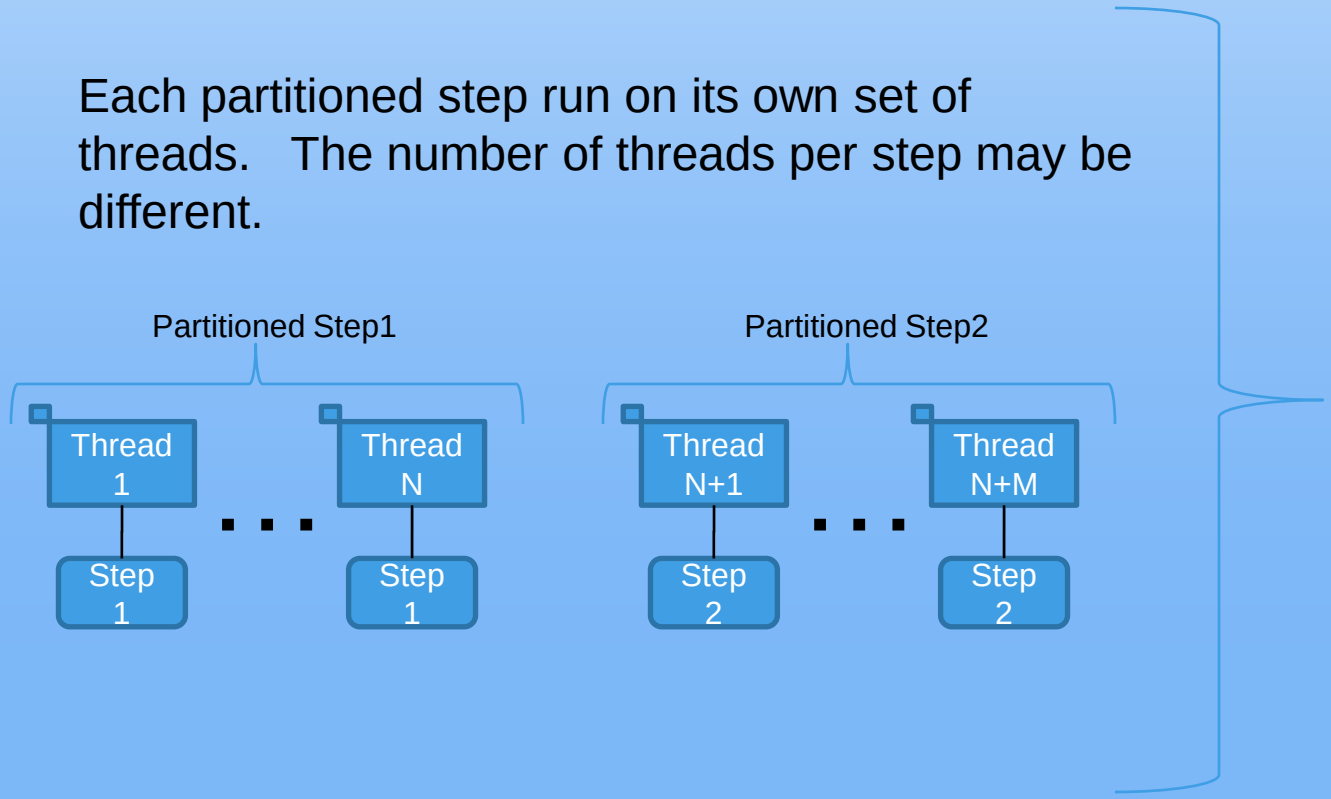
Multiple instances of Step1 started together on separate threads in different JVMs



Review: Parallel Models

Concurrent Partitioned Steps, Single JVM

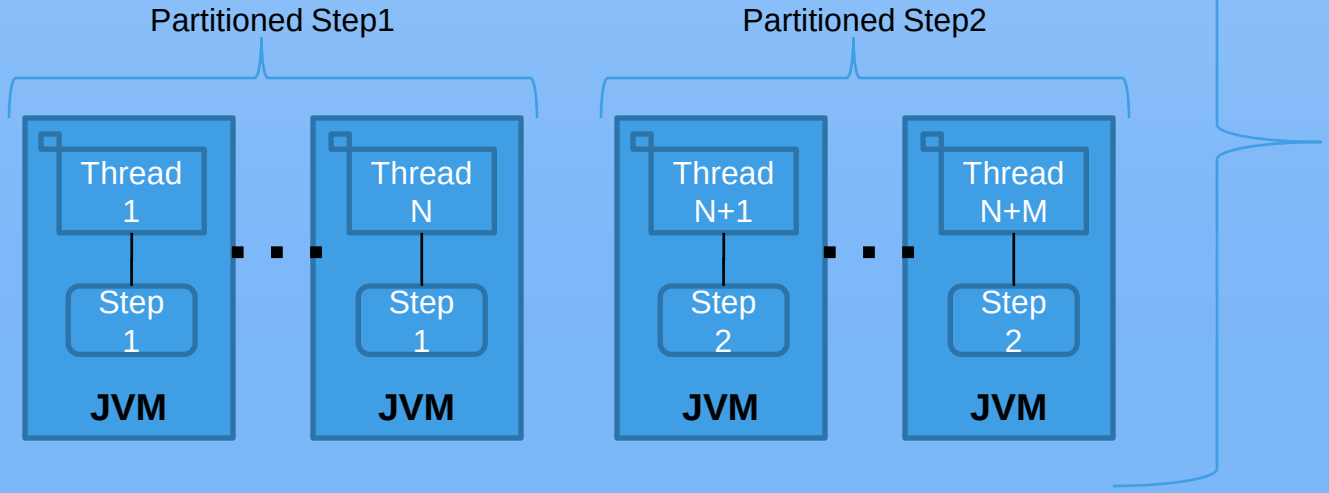
Each partitioned step run on its own set of threads. The number of threads per step may be different.



Review: Parallel Models

Concurrent Partitioned Steps, Multiple JVMs

Each partitioned step run on its own set of threads. The number of threads per step may be different. Threads are in different JVMs.

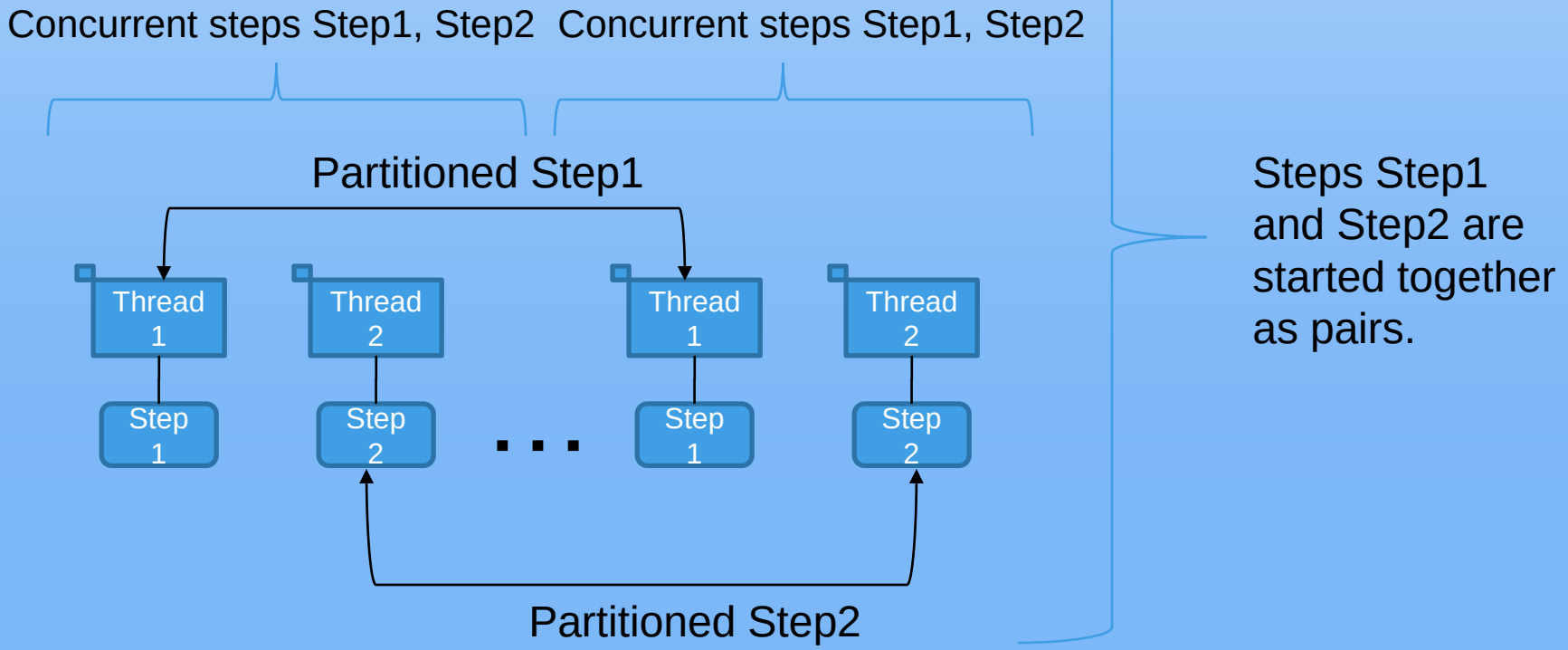


Partitioned Step1 and Partitioned Step2 are started together.

Review: Parallel Models

Partitioned Concurrent Steps, Single JVM

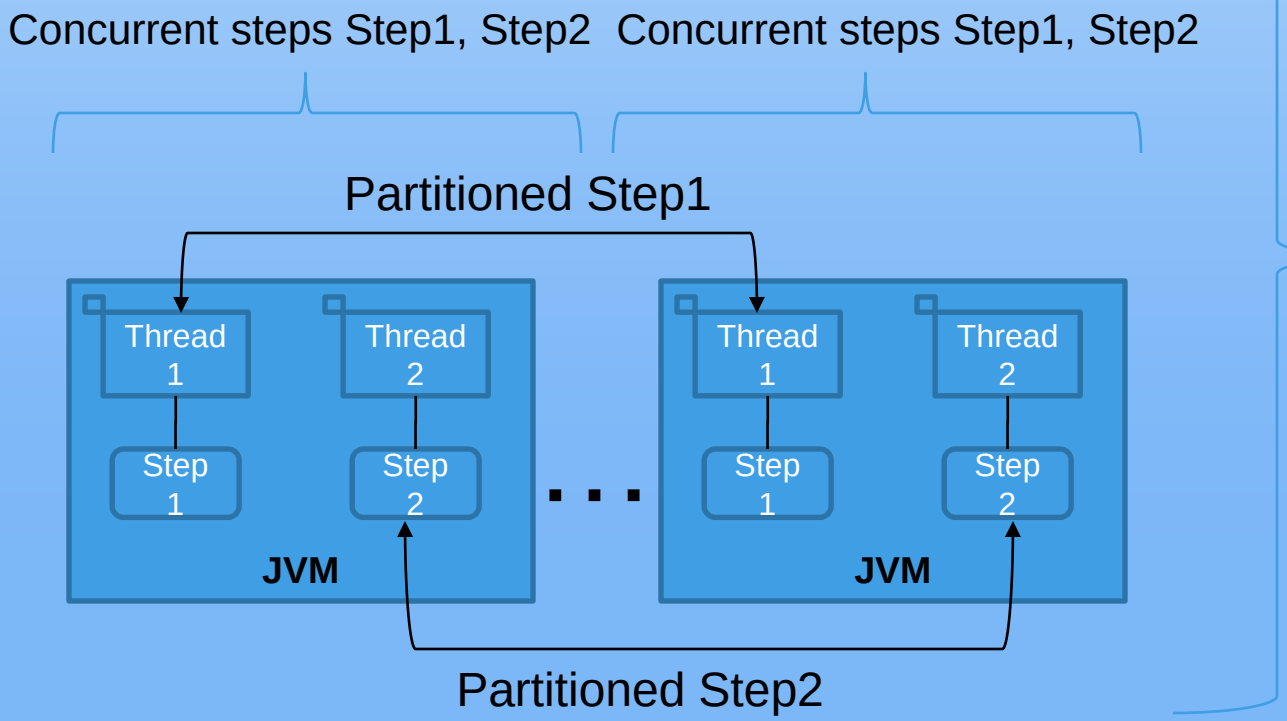
Each pair of concurrent steps is run in partitioned in the same JVM. The number of threads per step may be different.



Review: Parallel Models

Partitioned Concurrent Steps, Multiple JVMs

Each pair of concurrent steps is run in partitioned across multiple JVMs. The key requirement is same-JVM proximity for the concurrent steps, and then partitioning of the concurrent pairs.

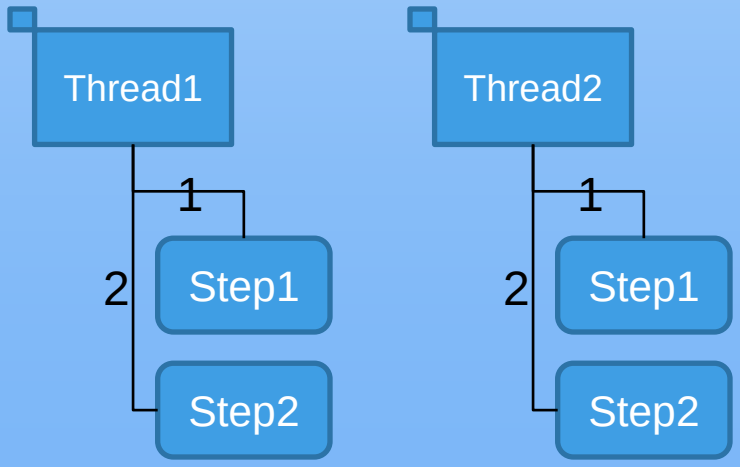


Steps Step1 and Step2 are started together as pairs across multiple JVMs.

Review: Parallel Models

Partitioned Sequential Steps, Single JVM

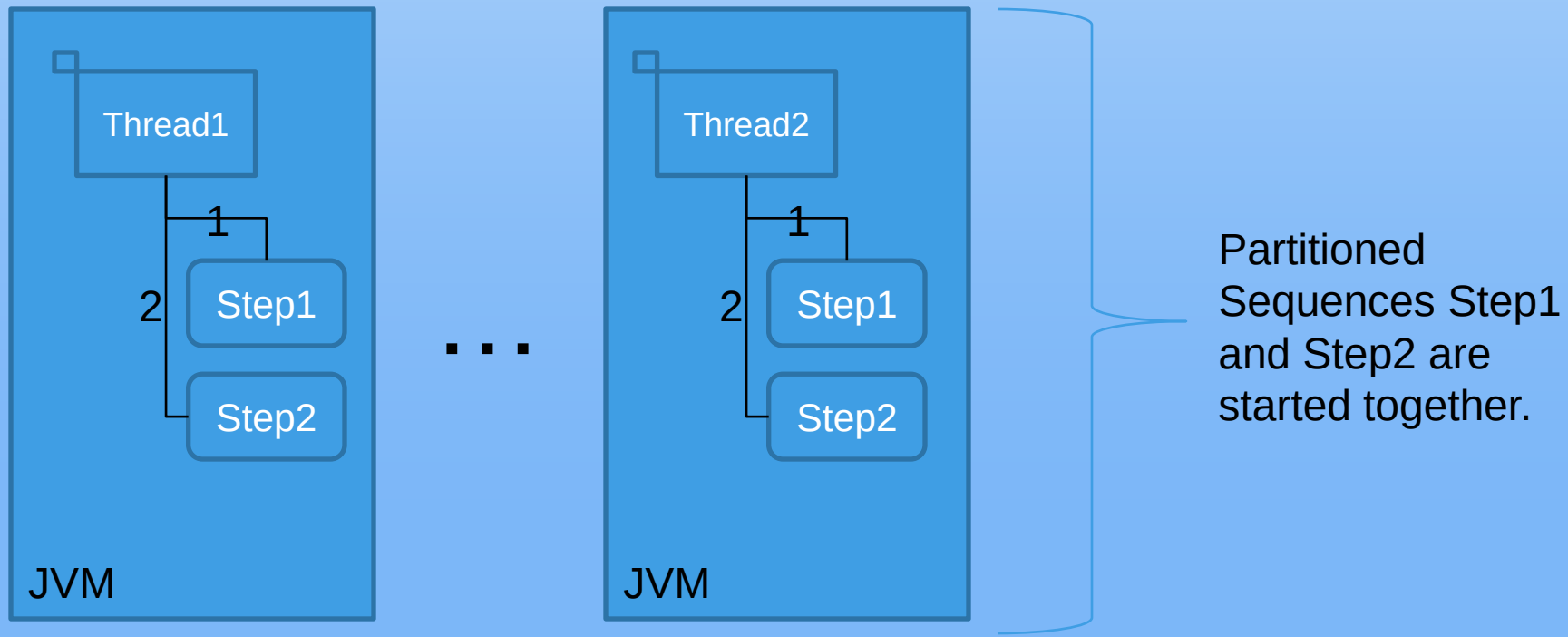
Each threads run the same sequence of steps.



Review: Parallel Models

Partitioned Sequential Steps, Multiple JVMs

Each threads run the same sequence of steps , each sequence runs in separate JVMs



Discussion: Repeat

- What is repeatable?
  - Item processing iteration

    e.g. @ItemProcessor <S> process(<T> item) until no more items or terminating exception

  - User-defined iteration?

    e.g. @Process RepeatStatus process()

Discussion: Repeat

- @ItemProcessor is transactional

  e.g. each checkpoint (chunk) is a transaction

- Should @Process be
  - (optionally) transactional?

    e.g. @Process(transactional=true) process()

  - allow checkpoints?

    e.g.

```
@Process RepeatStatus process() {
      return RepeatStatus.CHECKPOINT_AND_CONTINUE;
}
```

Discussion: Repeat

- Do we need a Repeat Listener?

```
@RepeatListener
public class MyRepeatListener {
   @BeforeRepeat void before(RepeatContext ctx) {}
   @AfterRepeat void after(RepeatContext ctx, RepeatStatus s) {}
   @OpenRepeat void open(RepeatContext ctx) {}
   @CloseRepeat void close(RepeatContext ctx) {}
   @OnError void onError(RepeatContext ctx, Throwable t) {}
}

@Job(name="Job1")
public class MyJob {
   @Step SomeStep step1;
   @RepeatListener MyRepeatListener listener;
   …
}
```

Discussion: Retry

- Needs
  - Step-level callback
  - Configurable retryable exceptions
  - Optional retry limit
  - Ability to backout (rollback)
- Step-level Callback – e.g.

```
@Step public MyStep {
    @BeforeRetry RetryStatus OnRetry(RetryContext ctx) {}
    @ItemProcessor <S> process(<T> item) {}
    @AfterRetry RetryStatus AfterRetry(RetryContext ctx) {}
}
```

Discussion: Retry

- Configurable Retryable Exceptions – e.g.

```
@Step
@Retry(@Exceptions(include={},exclude={}))
public MyStep {
    @BeforeRetry RetryStatus OnRetry(RetryContext ctx) {}
    @ItemProcessor <S> process(<T> item) {}
    @AfterRetry RetryStatus AfterRetry(RetryContext ctx) {}
}
```

Discussion: Retry

- Optional retry limit – e.g.

```
@Step
@Retry(limit=3)
public MyStep {
    @BeforeRetry RetryStatus OnRetry(RetryContext ctx) {}
    @ItemProcessor <S> process(<T> item) {}
    @AfterRetry RetryStatus AfterRetry(RetryContext ctx) {}
}
```

Discussion: Retry

- Ability to Backout (rollback)

```
@Step
@Retry(limit=3,backout=true)
public MyStep {
    @BeforeRetry RetryStatus OnRetry(RetryContext ctx) {}
    @ItemProcessor <S> process(<T> item) {}
    @AfterRetry RetryStatus AfterRetry(RetryContext ctx) {}
}
```

Discussion: Retry

- Do we need a Retry Listener?

```
@RetryListener
public class MyRetryListener {
   @OpenRetry void open(RetryContext ctx) {}
   @CloseRetry void close(RetryContext ctx) {}
   @OnError void onError(RetryContext ctx, Throwable t) {}
}

@Job(name="Job1")
public class MyJob {
   @Step SomeStep step1;
   @RetryListener MyRetryListener listener;
   …
}
```

Discussion: Skip

- Needs
  - Ability to define skippable conditions (just reads and writes?)
  - Step-level callback

```
@Step
@Skip(@Exceptions(include={},exclude={},limit=n)
public MyStep {
    @OnSkipInRead SkipStatus onSkip(Throwable t)
    @OnSkipInWrite SkipStatus onSkip(Throwable t, <T> Item)
}
```

Discussion: Skip

- Do we need a Skip Listener?

```
@SkipListener
public class MySkipListener {
   @OnSkipInRead void skipRead(Throwable t) {}
   @OnSkipInProcess void skipProcess(Throwable t, <T> item) {}
   @OnSkipInWrite void skipWrite(Throwable t, <S> item) {}
}

@Job(name="Job1")
public class MyJob {
   @Step SomeStep step1;
   @SkipListener MySkipListener listener;
   …
}
```

List for Next Meeting

- Parallel Annotations
- Future
  - Exit codes
  - Step conditions
  - Execution Context
  - Metrics
  - Java EE