JSR 352 Expert Group

Working Session

2 March 2012

Agenda

- Review: Readers/Writers
- Discussion: Listeners
- First Look: Concurrency – the Models
- List for Next Meeting

Review: Readers/Writers

- Annotations

```
@ItemReader (or @ItemWriter)
public  class MyReader {
    @Parameter String fileName= null;
    @Open void open(CheckpointInfo chkpt) {…}
    @Close void close() {…}
    @ReadItem {Type} read() {…}
    @GetCheckpointInfo CheckpointInfo getCheckpt() {…}
}

@WriteItem void write({Type} record) {…}
```

Discussion: Readers/Writers

- Usage

```
@Step(name="Postings")
public class PostingsStep {

@ItemReader FileReader reader;
@ItemWriter FileWriter writer;
@ItemProcessor {Type1} process({Type2} rec) {…}

}
```

Defining reader/writer as field allows natural use of DI, e.g. CDI example with qualifier:

```
@Inject @TestMode
            @ItemReader FileReader reader;
```

Discussion: Listeners

- Spring Batch Listeners
  - JobExecutionListener
  - StepExecutionListener
  - ChunkListener
  - ItemReadListener<T>
  - ItemProcessListener<T, S>
  - ItemWriteListener<S>
  - SkipListener<T,S>
  - RepeatListener
  - RetryListener
- WebSphere Batch Listeners
  - JobListener (combines Job/StepExecutionListeners)
  - SkipListener
  - RetryListener
  - CheckpointListener (i.e. ChunkListener)

Discussion: Listeners

- JobExecutionListener

```
@Job(name=“Job1”)
public class MyJob {
        @BeforeJob void before();
        @AfterJob void after();
}
```

- StepExecutionListener

```
@Step(name=“Step1”)
public class MyStep {
        @BeforeStep void before();
        @AfterStep void after();
}
```

How do listeners know identity? I.e. job name and step name.

Job name and current step name could be exposed in runtime “batch context” object.

Discussion: Listeners

- ChunkListener

```
@Step
public class MyStep {
        @BeforeCheckpoint void before();
        @AfterCheckpoint void after();
}
```

- ItemReadListener<T>

```
@Step
public class MyStep {
        @BeforeRead void beforeRead();
        @AfterRead void afterRead({Type} item);
        @OnReadError void onReadError(Exception ex);
}
```

Discussion: Listeners

- ItemProcessListener<T, S>

```
@Step
public class MyStep {
        @BeforeProcess void beforeProcess({Type} item);
        @AfterProcess void afterProcess({Type} item, {Type2} result);
        @OnProcessError void onProcessError(Exception ex, {Type} item);
}
```

- ItemWriteListener<S>

```
@Step
public class MyStep {
        @BeforeWrite void beforeWrite(List<{Type}> items);
        @AfterWrite void afterWrite(List<{Type}> items);
        @OnWriteError void onWriteError(Exception ex, List<{Type}> items);
}
```

Discussion: Listeners

- SkipListener<T,S>

```
@Step
public class MyStep {
        @OnSkipInRead void onSkipInRead(Throwable t);
        @OnSkipInProcess void onSkipInProcess({Type} item, Throwable t);
        @OnSkipInWrite void onSkipInWrite({Type2} result, Throwable t);
}
```

- RepeatListener

```
@Step
public class MyStep {
        @BeforeRepeat void beforeRepeat(RepeatContext context);
        @AfterRepeat void afterRepeat(RepeatContext context,
                                              RepeatStatus result);
        @OpenRepeat void openRepeat(RepeatContext context);
        @OnErrorRepeat void onErrorRepeat(RepeatContext context,
                                                    Throwable e);
        @CloseRepeat closeRepeat(RepeatContext context);
}
```

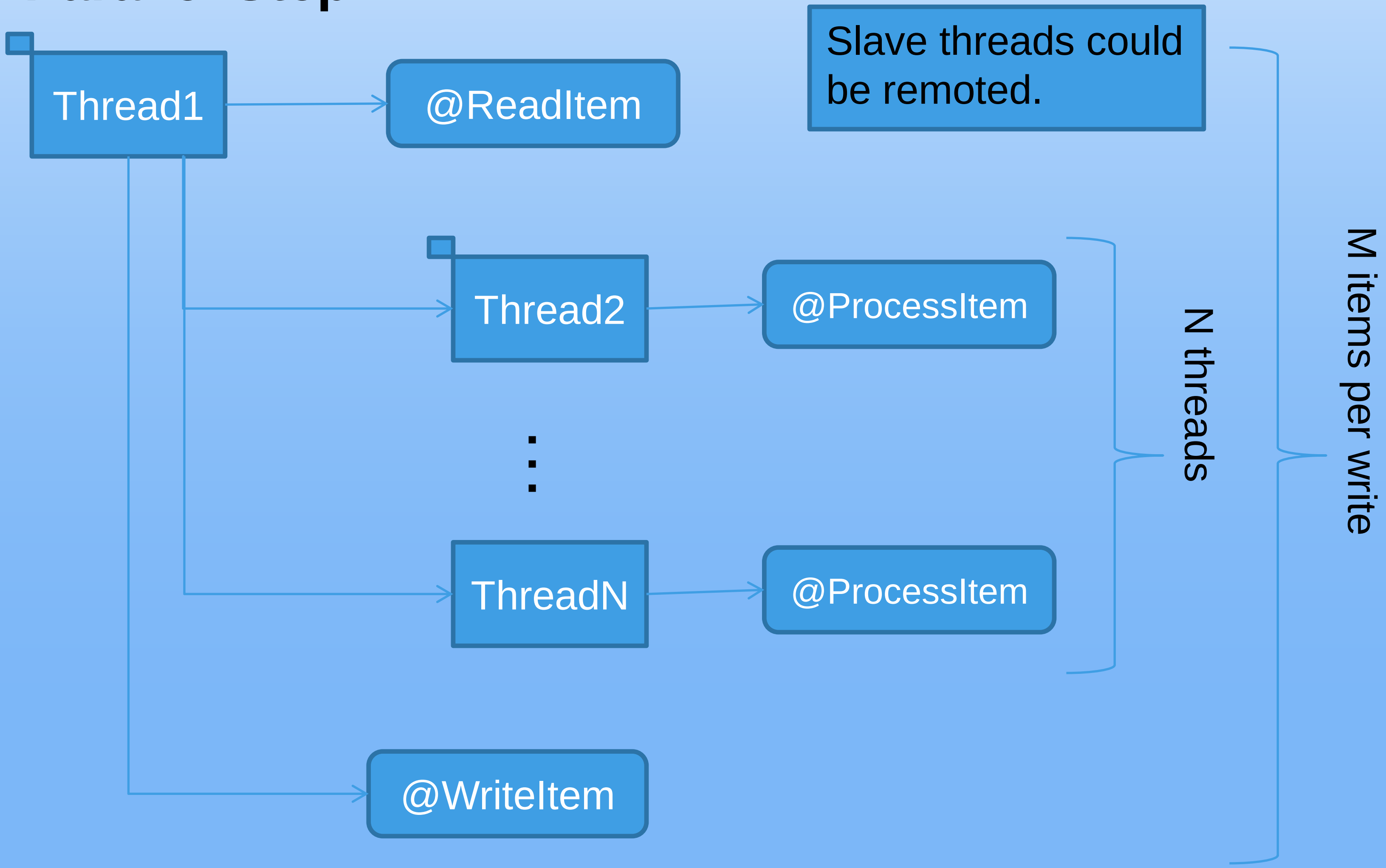
Discussion: Listeners

- RetryListener

```
@Step
public class MyStep {
@OpenRetry void openRetry(RetryContext context,
                                        RetryCallback{Type} callback);
@OnErrorRetry void onErrorRetry(RetryContext context,
                                        RetryCallback{Type} callback, Throwable);
@CloseRetry void closeRetry(RetryContext context,
                              RetryCallback{Type} callback, Throwable e);
}
```

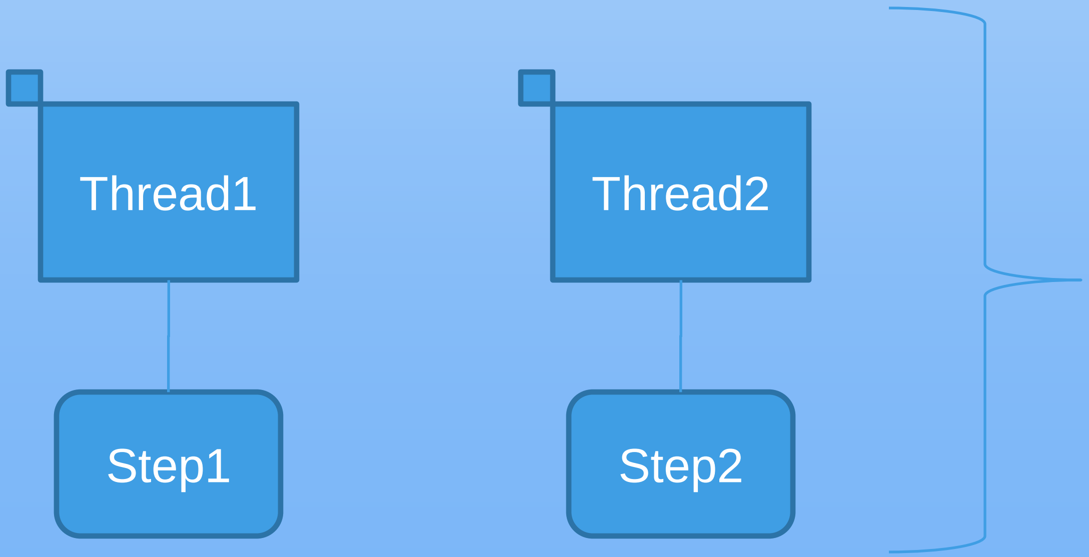
First Look: Concurrency

Parallel Step



First Look: Concurrency

Concurrent Steps, Single JVM

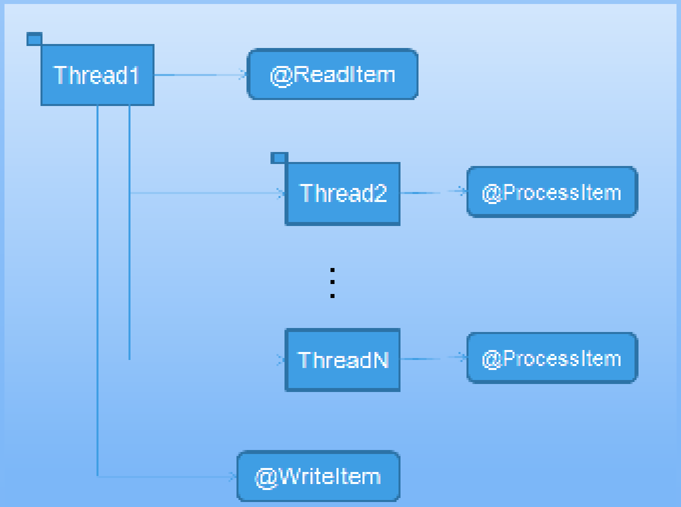


Step1 and Step2 started at same time on separate threads

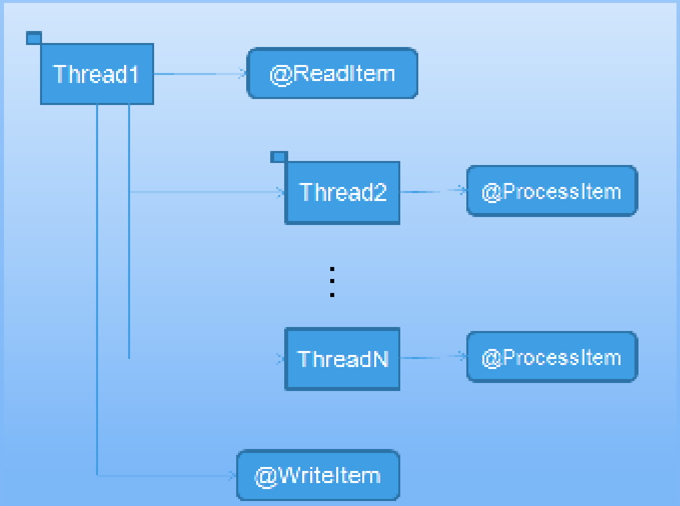
First Look: Concurrency

Concurrent and Parallel can be combined

Step1

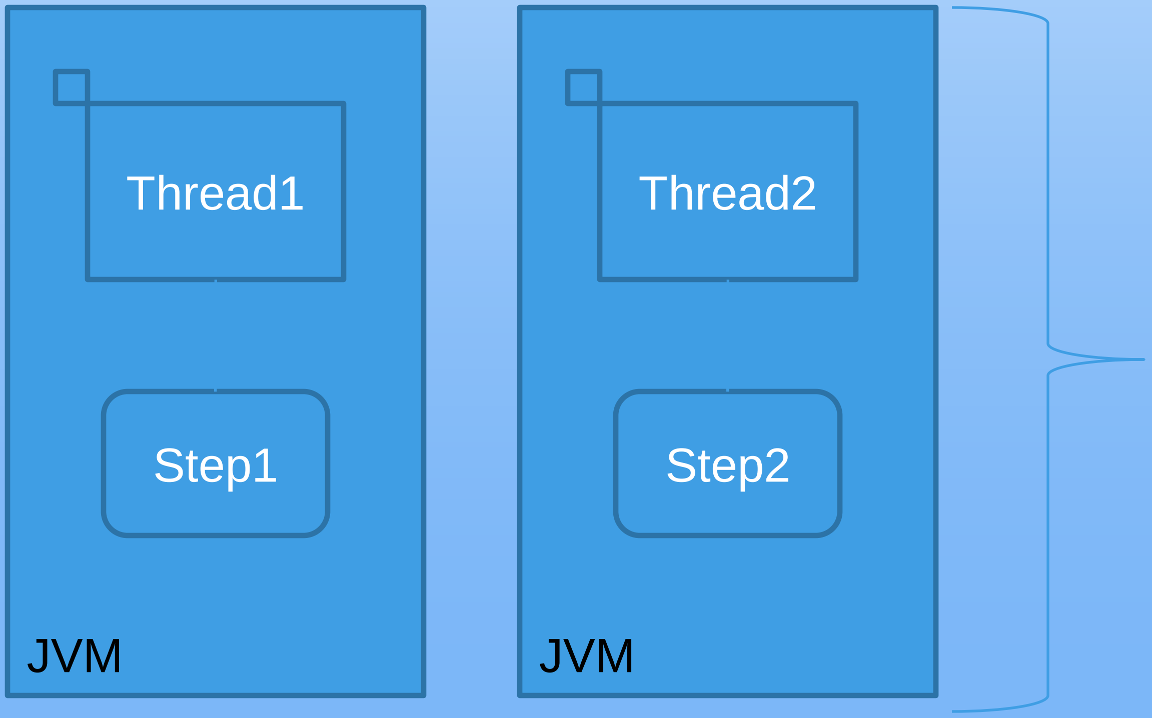


Step2



First Look: Concurrency

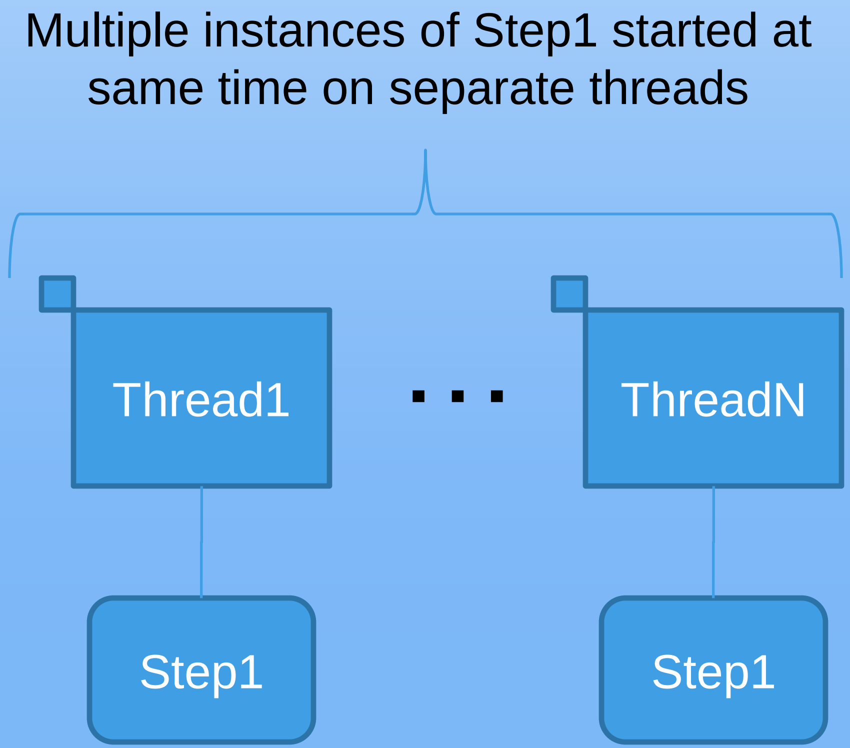
Concurrent Steps, Multiple JVMs



Step1 and Step2 started at same time on separate threads, in different JVMs

First Look: Concurrency

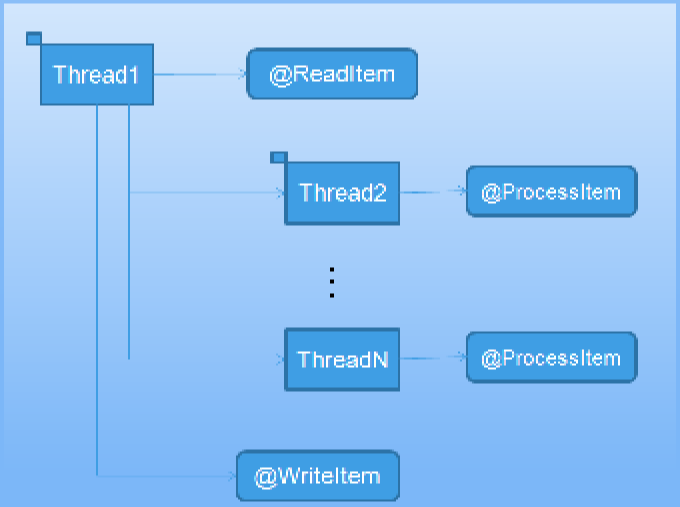
Partitioned Step, Single JVM

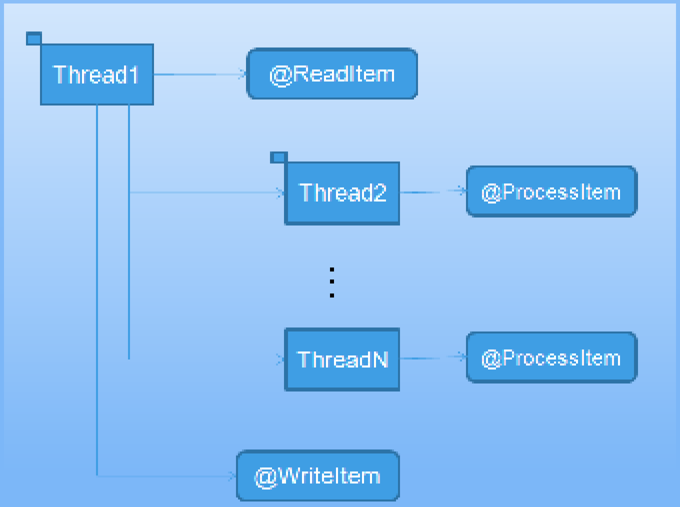

Multiple instances of Step1 started at same time on separate threads
Thread1
. . .
ThreadN
Step1
Step1

First Look: Concurrency

Partitioned and parallel can be combined.

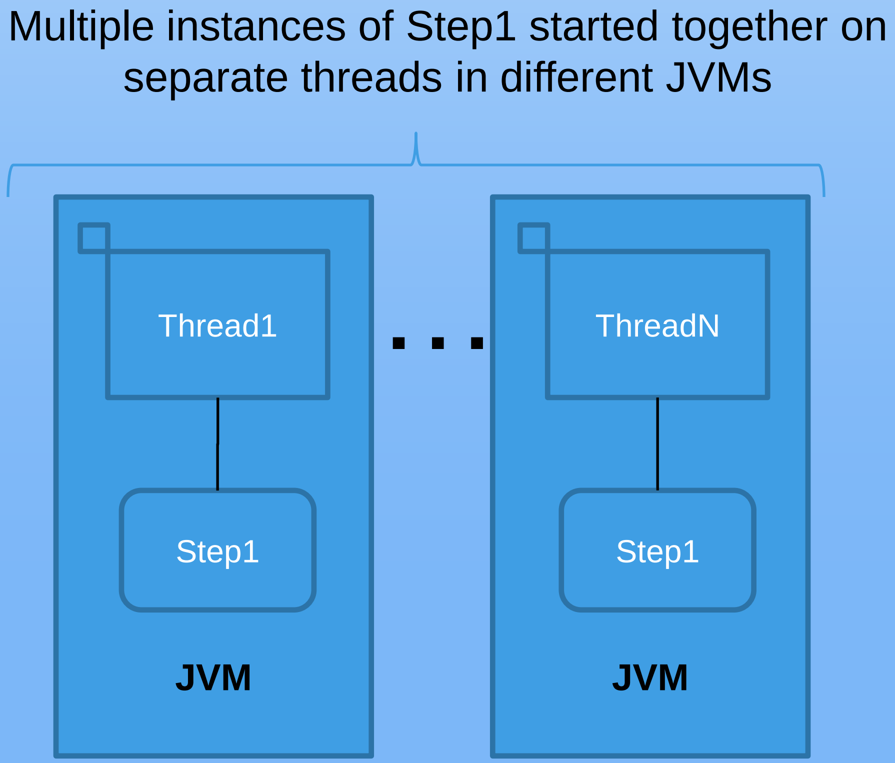
Multiple instances of Step1 started at same time on separate threads





First Look: Concurrency

Partitioned Step, Multiple JVMs

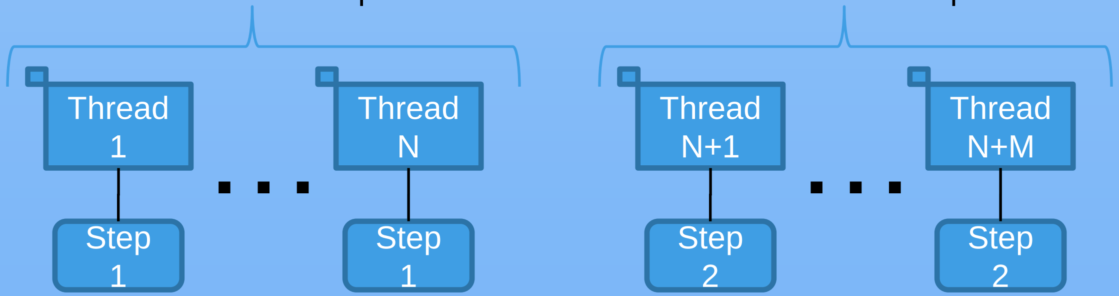


Multiple instances of Step1 started together on separate threads in different JVMs
Thread1
Step1
JVM
. . .
ThreadN
Step1
JVM

First Look: Concurrency

Concurrent Partitioned Steps, Single JVM

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

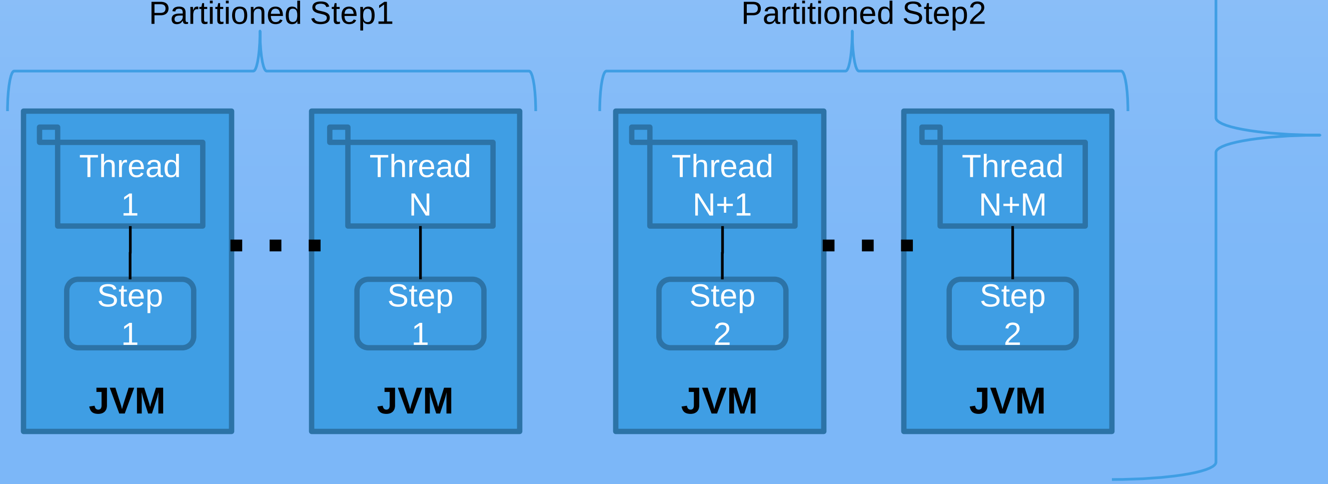


Partitioned Step1 and Partitioned Step2 are started together.

First Look: Concurrency

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.

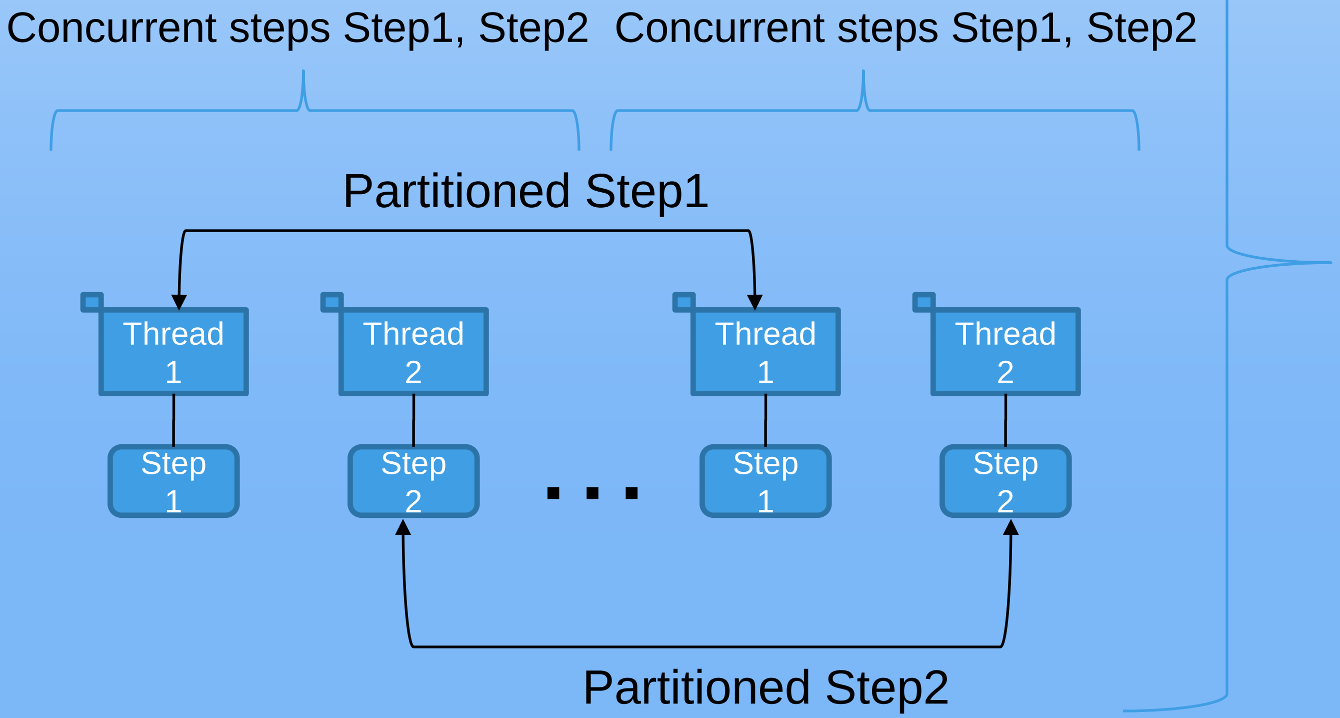


Partitioned Step1 and Partitioned Step2 are started together.

First Look: Concurrency

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.

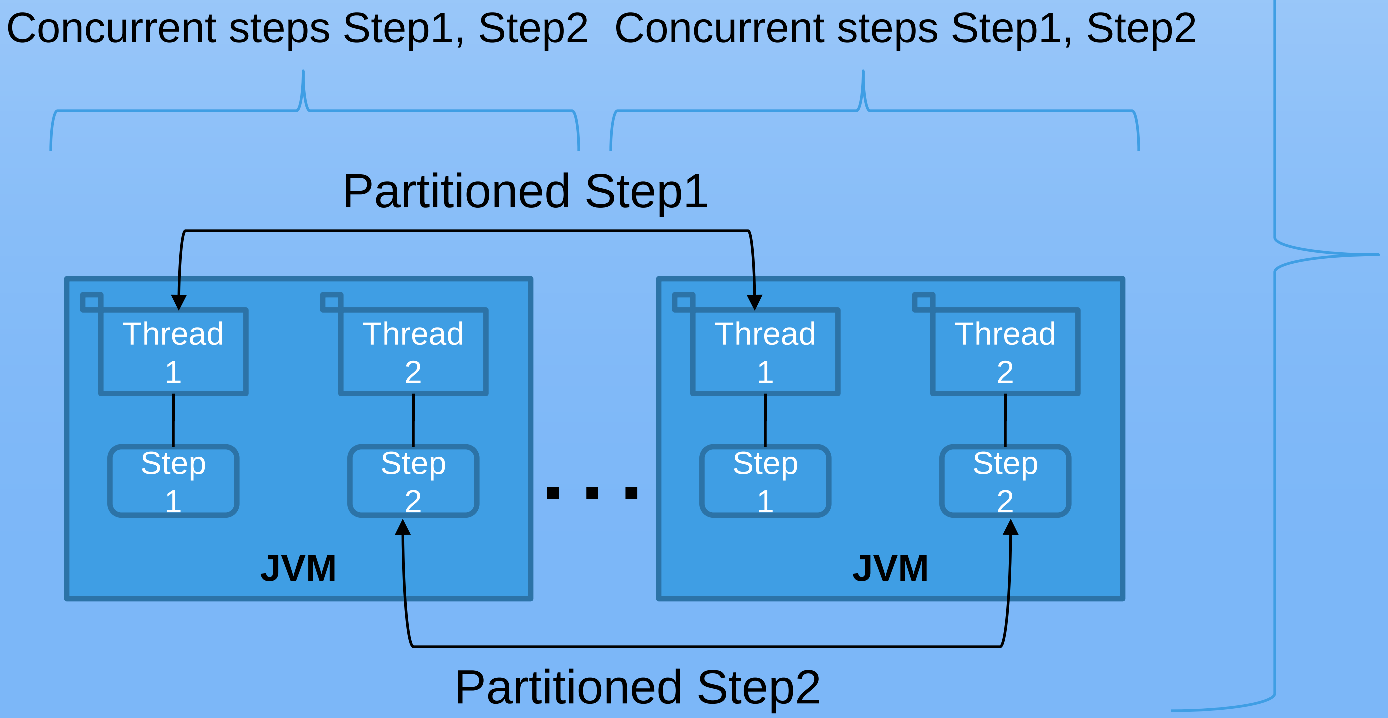


Steps Step1 and Step2 are started together as pairs.

First Look: Concurrency

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.

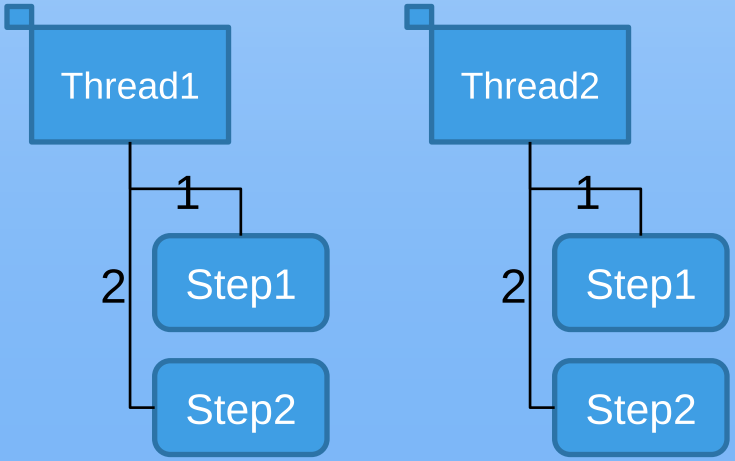


Concurrent steps Step1, Step2
Concurrent steps Step1, Step2
Partitioned Step1
Thread 1
Thread 2
Step 1
Step 2
JVM
Thread 1
Thread 2
Step 1
Step 2
JVM
. . .
Partitioned Step2
Steps Step1 and Step2 are started together as pairs across multiple JVMs.

First Look: Concurrency

Partitioned Sequential Steps, Single JVM

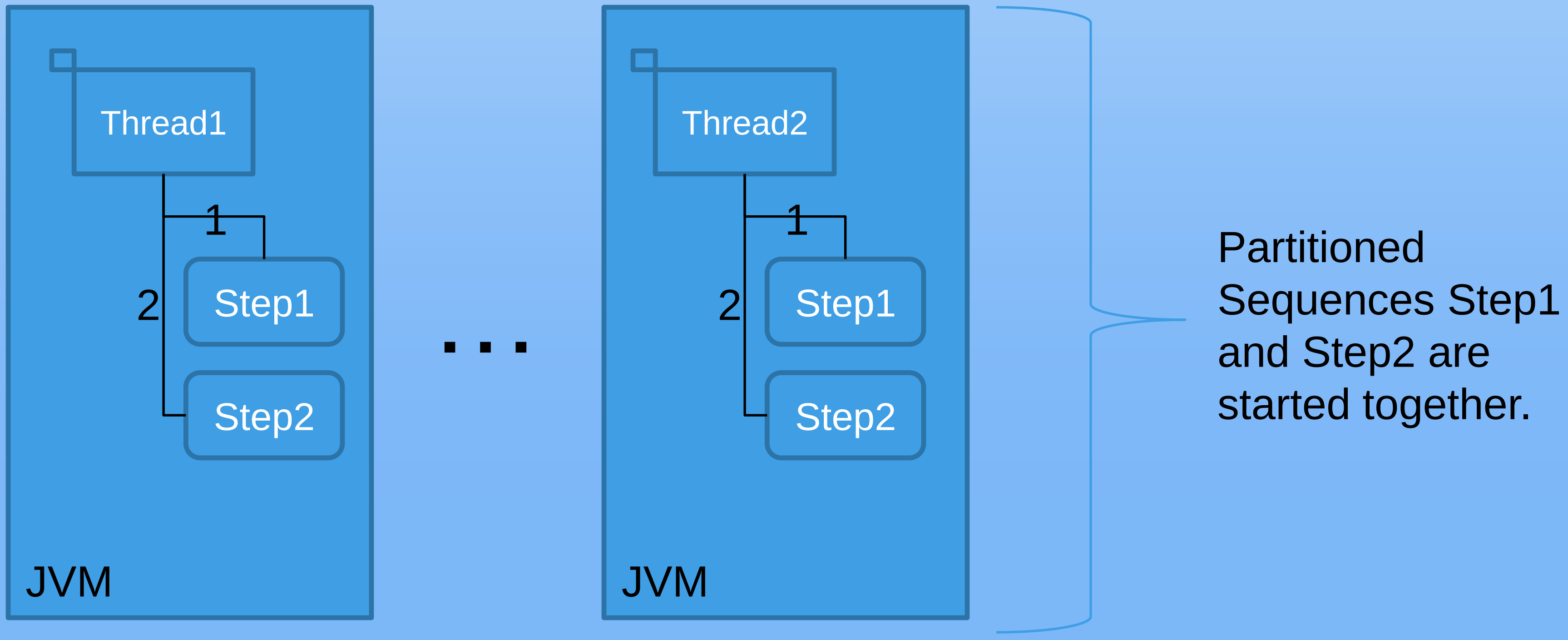
Each threads run the same sequence of steps.



First Look: Concurrency

Partitioned Sequential Steps, Multiple JVMs

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



List for Next Meeting

- Repeat
- Retry
- More on Concurrency
- Future
  - Exit codes
  - Step conditions
  - Execution Context
  - Metrics
  - Java EE