JSR 352 Expert Group

Working Session

14 March 2012

Agenda

- Discussion: Parallel Annotations
- Discussion: Job Context
- List for Next Meeting

Discussion: Parallel Annotations

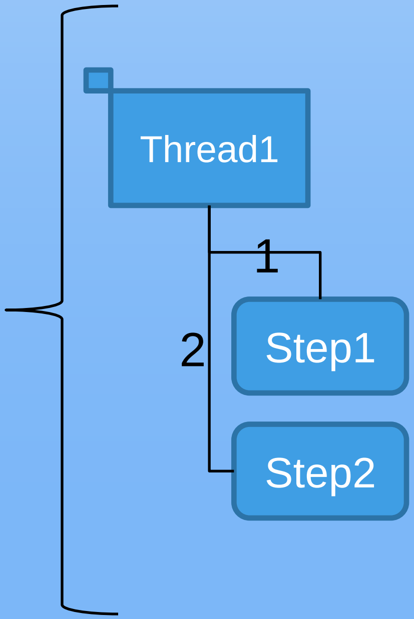
- Decide which models to support
  - Partitioned, Concurrent, Pipeline
- RI/TCK will support single JVM parallelization, but spec must accommodate multi-JVM
- Annotations must support
  - Flow control specification (i.e. which style of parallelization, which steps, etc)
  - Parallel decision/communication needs (e.g. partitioning, inter-job communication, etc)

Discussion: Parallel Annotations

First: Sequential Steps, Single JVM

Threads run a sequence of steps in a specified order.

Need a way to specify execution order



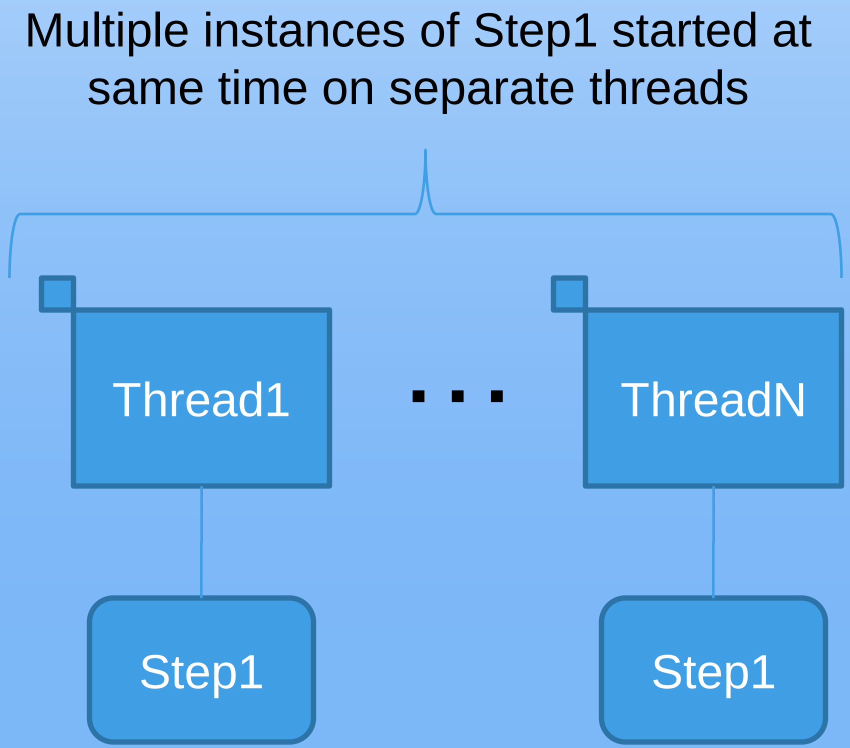
Discussion: Parallel Annotations

Sequential Steps, Single JVM

```
@Job(name="Job1")
@ExecutionOrder=({“Step1”,”Step2”})
public class MyJob {
        @Step(name="Step1") MyStep myStep;
        @Step(name="Step2") MyOtherStep myOtherStep;
}
```

Discussion: Parallel Annotations

Partitioned Step, Single JVM


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

Discussion: Parallel Annotations

Partitioned Step, Single JVM

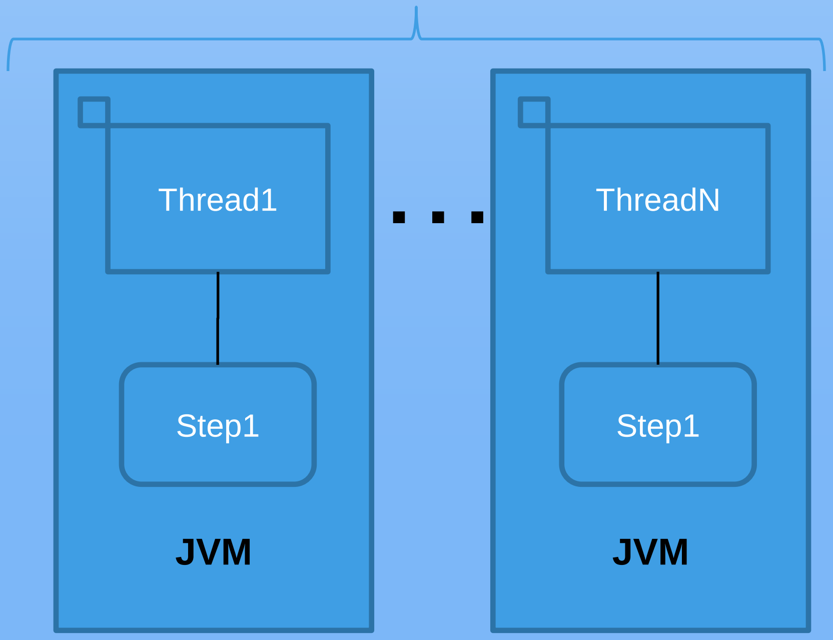
```
@Job(name="Job1")
public class MyJob {
        @Step(name="Step1")
                @Parallel(partition=true) MyStep myStep;
}
```

Note: there will be an annotation that deals with “partitioning rules”
e.g. how many partitions, unique parameter values for each

Discussion: Parallel Annotations

Partitioned Step, Multiple JVMs

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



Discussion: Parallel Annotations

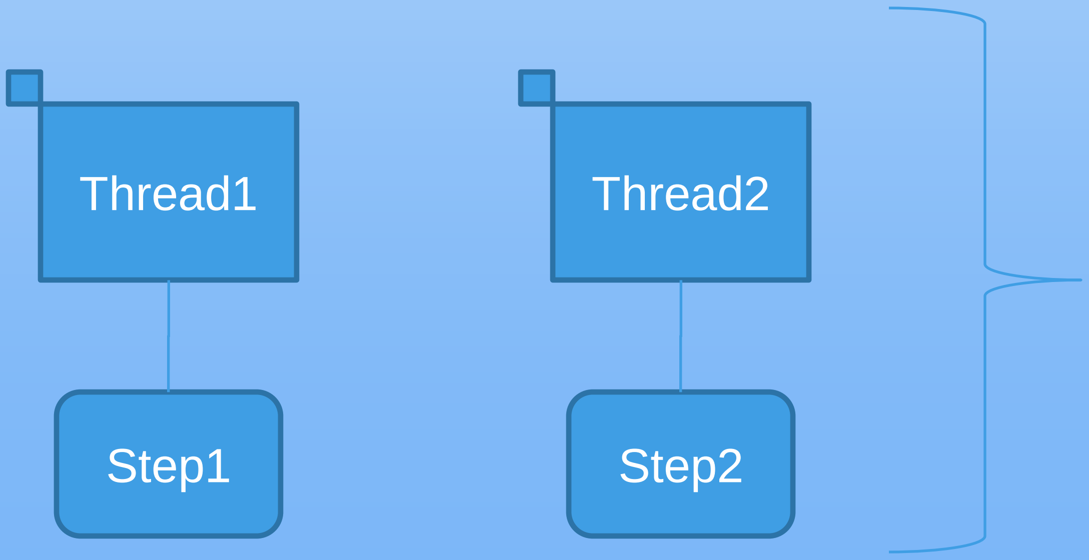
Partitioned Step, Multiple JVMs

```
@Job(name="Job1")
public class MyJob {
        @Step(name="Step1")
                @Parallel(partition=true,jvms=JVMs.MULTIPLE)
                        MyStep myStep;
}
```

Note: there will be a batch container plug-in that deals with partition distribution

Discussion: Parallel Annotations

Concurrent Steps, Single JVM



Step1 and Step2 started at same time on separate threads

Discussion: Parallel Annotations

Concurrent Steps, Single JVM

Presence of multiple steps requires use of @ExecutionOrder annotation.

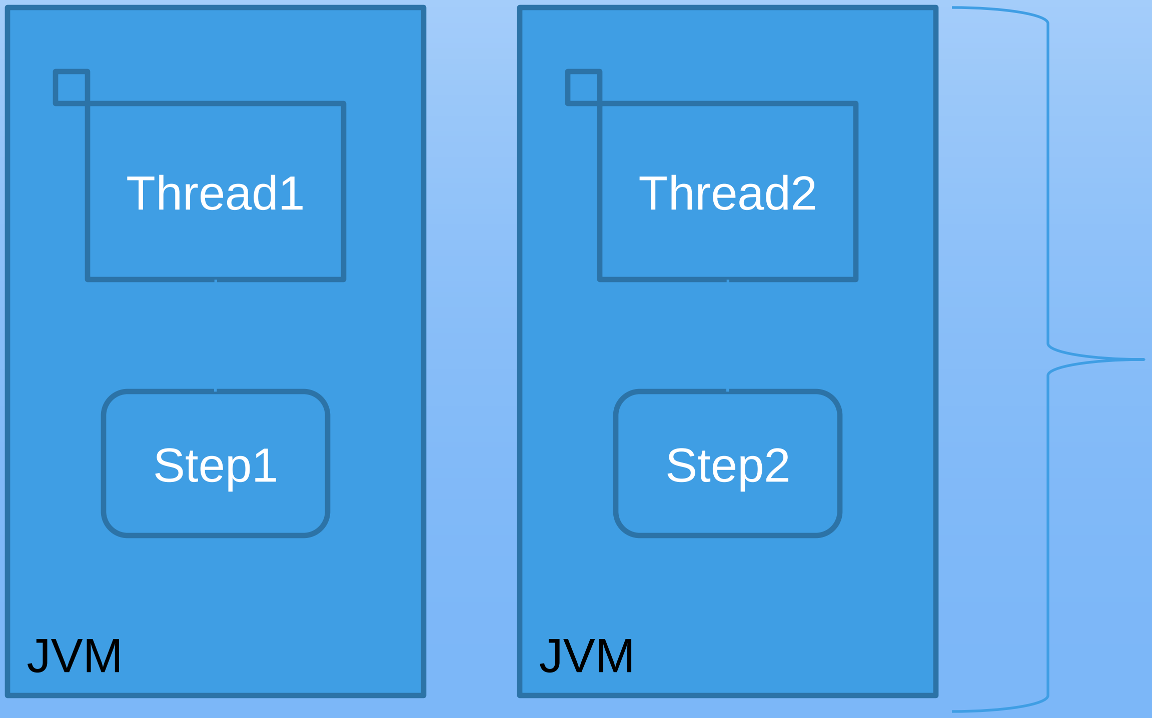
```
@Job(name="Job1")
@ExecutionOrder=(“Group1")
public class MyJob {
        @Step(name="Step1") MyStep myStep;
        @Step(name="Step2") MyOtherStep myOtherStep;
        @StepGroup(name=“Group1”) String[]= {“Step1”,”Step2”}

}
```

Note: A StepGroup is concurrent steps, single JVM by default.

Discussion: Parallel Annotations

Concurrent Steps, Multiple JVMs



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

Discussion: Parallel Annotations

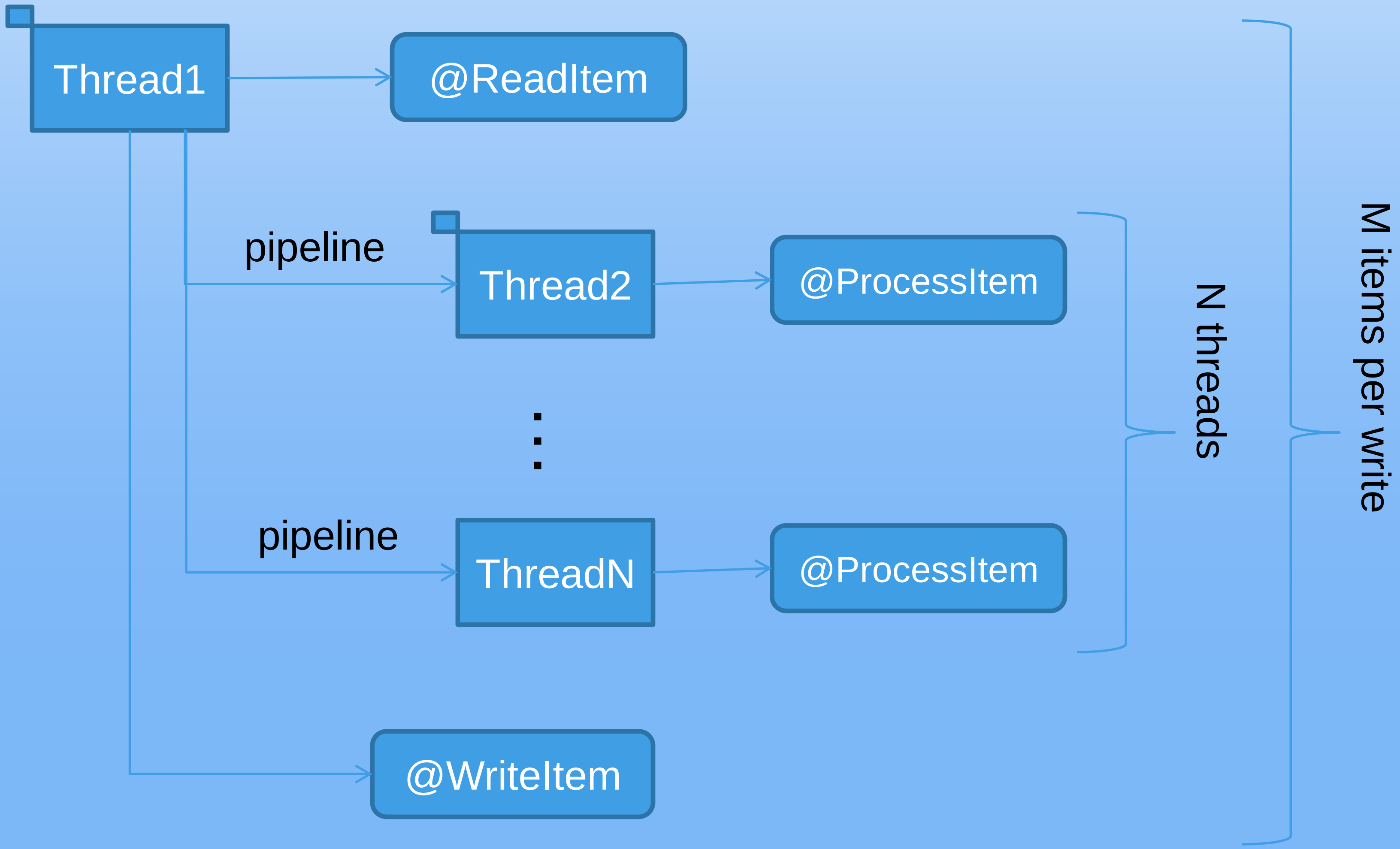
Concurrent Steps, Multiple JVMs

```
@Job(name="Job1")
@ExecutionOrder=( “Group1”)
public class MyJob {
        @Step(name="Step1") MyStep myStep;
        @Step(name="Step2") MyOtherStep myOtherStep;
        @StepGroup(name=“Group1”)
                @Parallel(concurrent=true,jvms=JVMs.MULTIPLE)
                        String[]= {“Step1”,”Step2”}

}
```

Discussion: Parallel Annotations

Pipeline Step, Single JVM



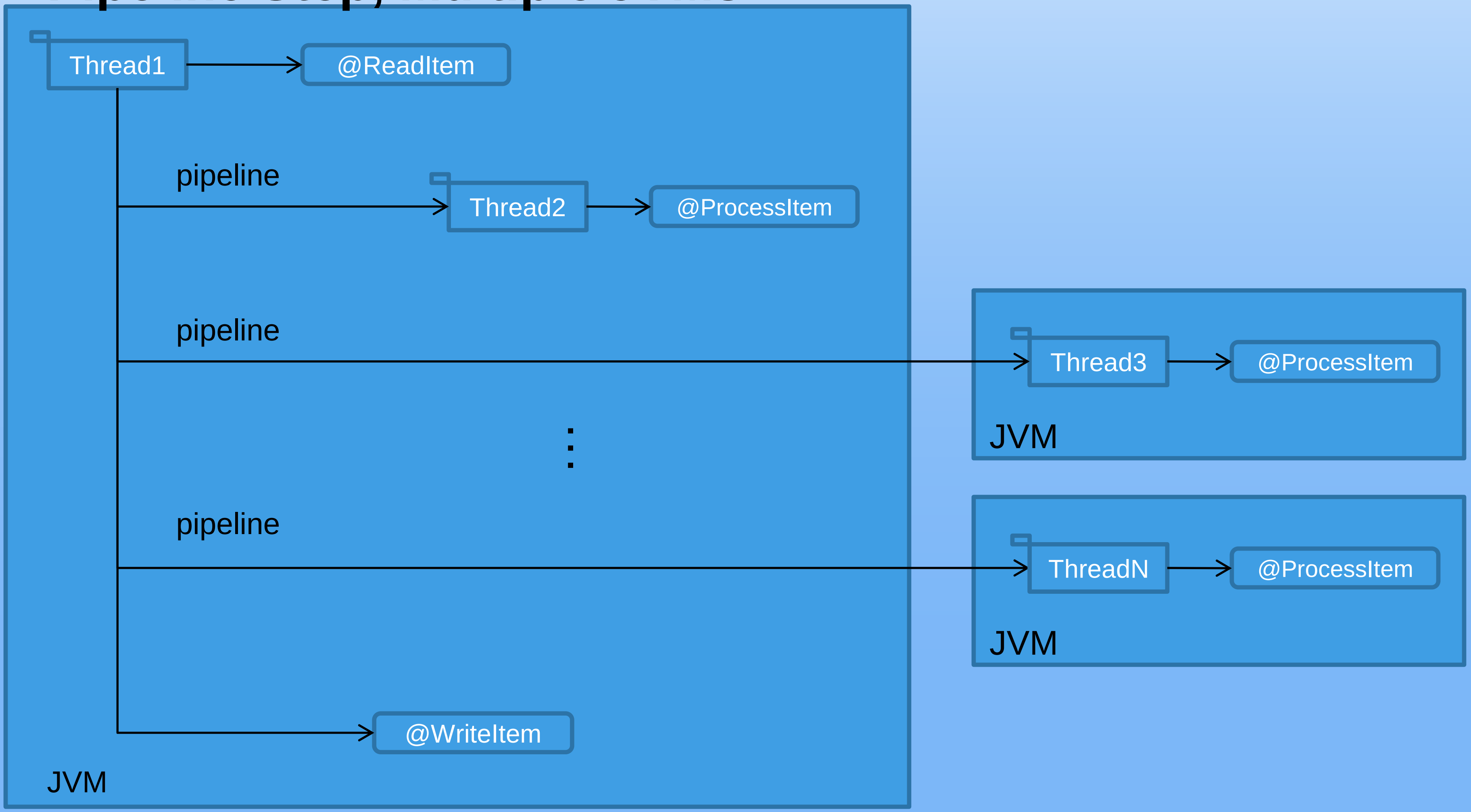
Discussion: Parallel Annotations

Pipeline Step, Single JVM

```
@Job(name="Job1")
public class MyJob {
        @Step(name="Step1")
                @Parallel(pipeline=true) MyStep myStep;
}
```

Discussion: Parallel Annotations

Pipeline Step, Multiple JVMs



Discussion: Parallel Annotations

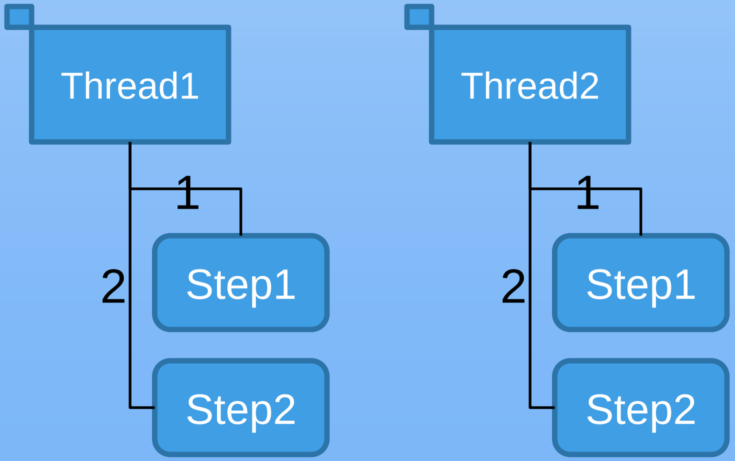
Pipeline Step, Multiple JVMs

```
@Job(name="Job1")
public class MyJob {
        @Step(name="Step1")
                @Parallel(pipeline=true,jvms=JVMs.MULTIPLE)
                        MyStep myStep;
}
```

Discussion: Parallel Annotations

Partitioned Sequential Steps, Single JVM

Each threads run the same sequence of steps.



Discussion: Parallel Annotations

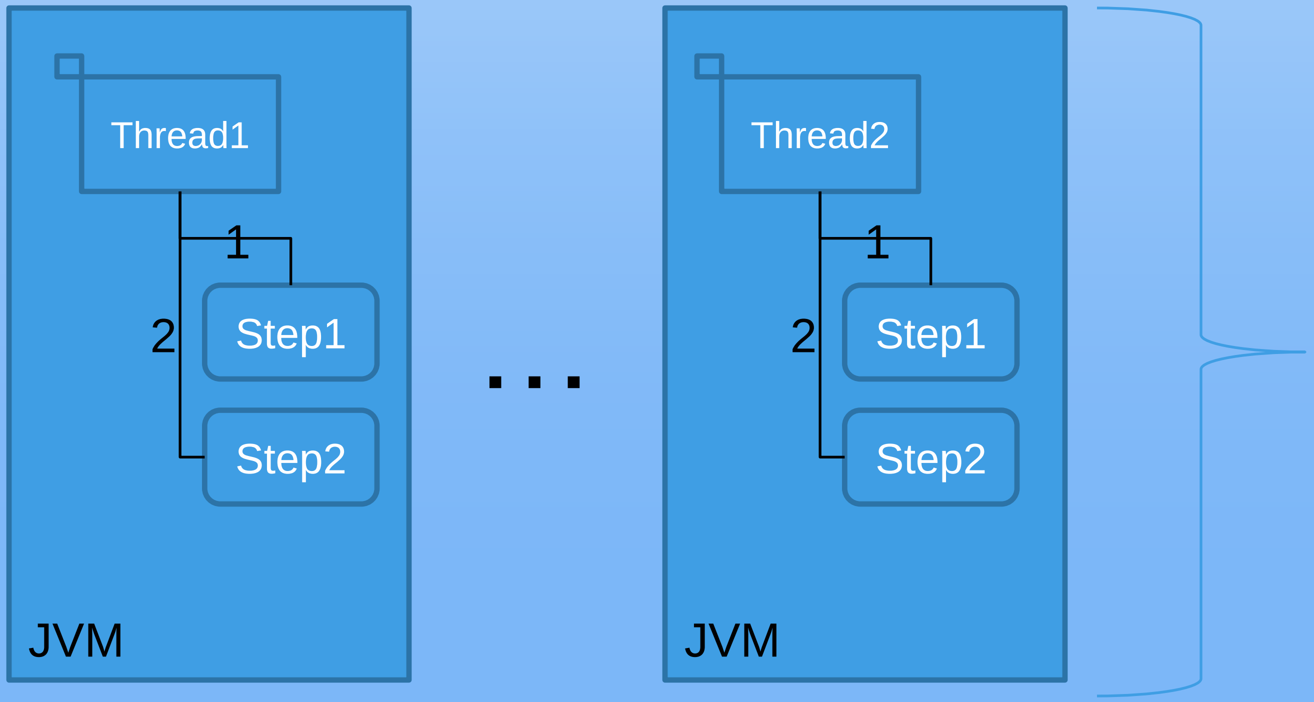
Partitioned Sequential Steps, Single JVM

```
@Job(name="Job1")
@ExecutionOrder=(“Group1”)
public class MyJob {
        @Step(name="Step1") MyStep myStep;
        @Step(name="Step2") MyOtherStep myOtherStep;
        @StepGroup(name=“Group1”)
                @Parallel(partition=true) String[]={“Step1”,”Step2”}
}
```

Discussion: Parallel Annotations

Partitioned Sequential Steps, Multiple JVMs

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



Partitioned Sequences Step1 and Step2 are started together.

Discussion: Parallel Annotations

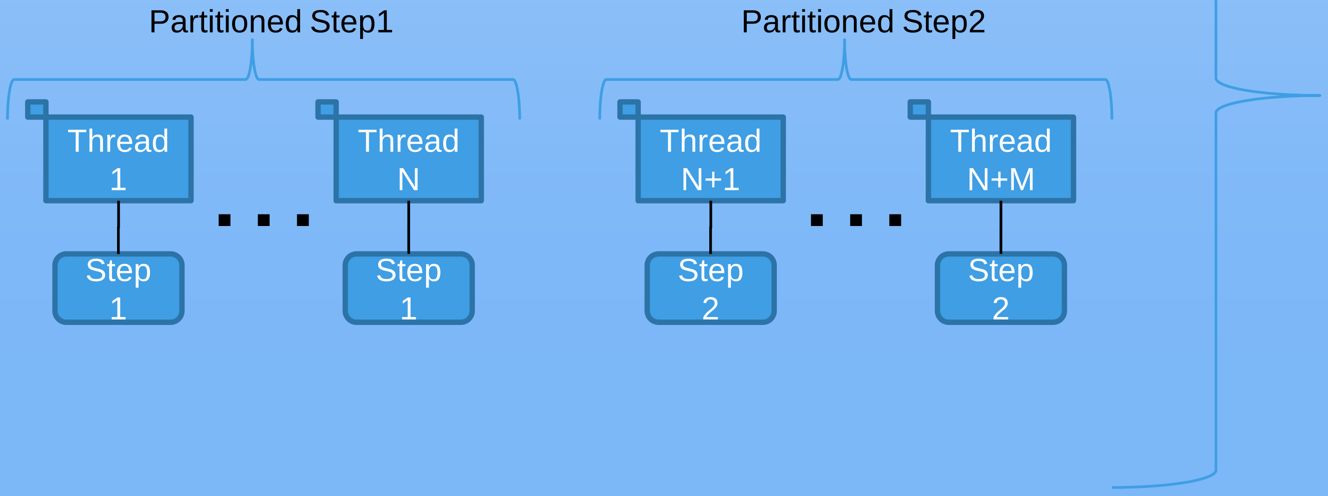
Partitioned Sequential Steps, Multiple JVMs

```
@Job(name="Job1")
@ExecutionOrder=(“Group1”)
public class MyJob {
        @Step(name="Step1") MyStep myStep;
        @Step(name="Step2") MyOtherStep myOtherStep;
        @StepGroup(name=“Group1”)
                @Parallel(partition=true,jvms=JVMs.MULTIPLE)
                        String[]={“Step1”,”Step2”}
}
```

Discussion: Parallel Annotations

Concurrent Partitioned Steps, Single JVM

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



Partitioned Step1 and Partitioned Step2 are started together.

Discussion: Parallel Annotations

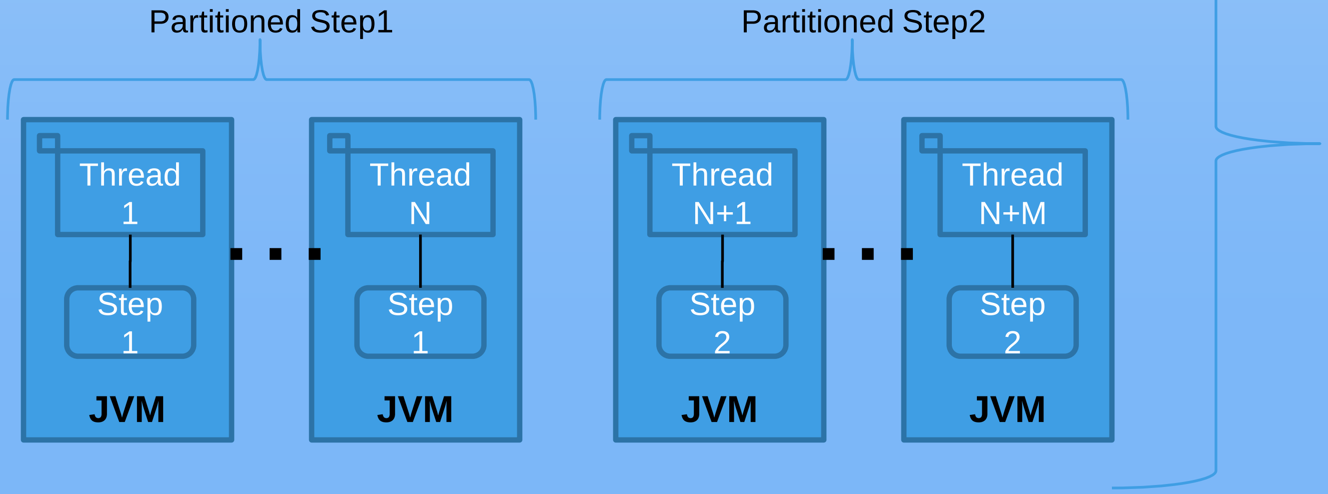
Concurrent Partitioned Steps, Single JVM

```
@Job(name="Job1")
@ExecutionOrder=("Group1")
public class MyJob {
        @Step(name="Step1")
                @Parallel(partition=true) MyStep myStep;
        @Step(name="Step2")
                @Parallel(partition=true) MyOtherStep myOtherStep;
        @StepGroup(name="Group1")
                @Parallel(concurrent=true)
                        String[]={"Step1","Step2"}
}
```

Discussion: Parallel Annotations

Concurrent Partitioned Steps, Multiple JVMs

Each partitioned step runs 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.

Discussion: Parallel Annotations

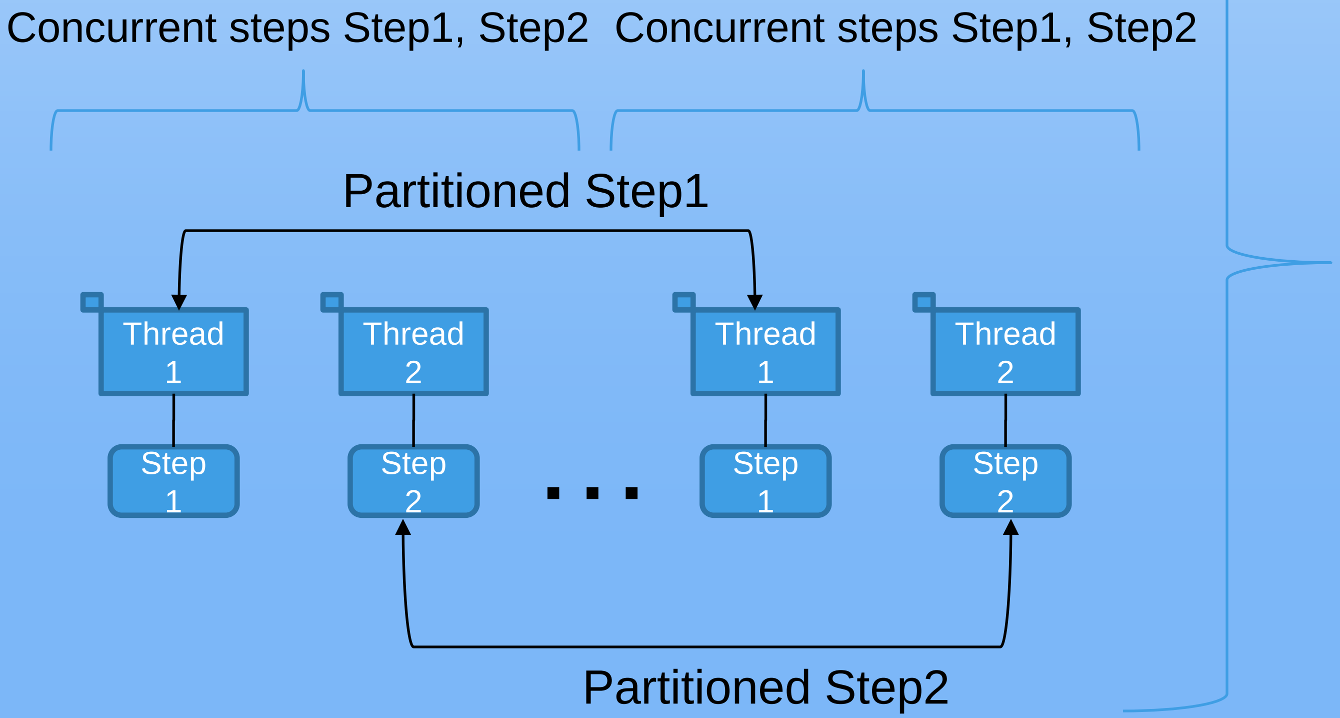
Concurrent Partitioned Steps, Multiple JVMs

```
@Job(name="Job1")
@ExecutionOrder=("Group1")
public class MyJob {
        @Step(name="Step1")
                @Parallel(partition=true,jvms=JVMs.MULTIPLE)
                        MyStep myStep;
        @Step(name="Step2")
                @Parallel(partition=true,jvms=JVMs.MULTIPLE)
                        MyOtherStep myOtherStep;
        @StepGroup(name="Group1")
                @Parallel(concurrent=true)
                        String[]={"Step1","Step2"}
}
```

Discussion: Parallel Annotations

Partitioned Concurrent Steps, Single JVM

Each group of concurrent steps runs partitioned in the same JVM. The number of threads per step is the same.



Steps Step1 and Step2 are started together as a group.

Discussion: Parallel Annotations

Partitioned Concurrent Steps, Single JVM

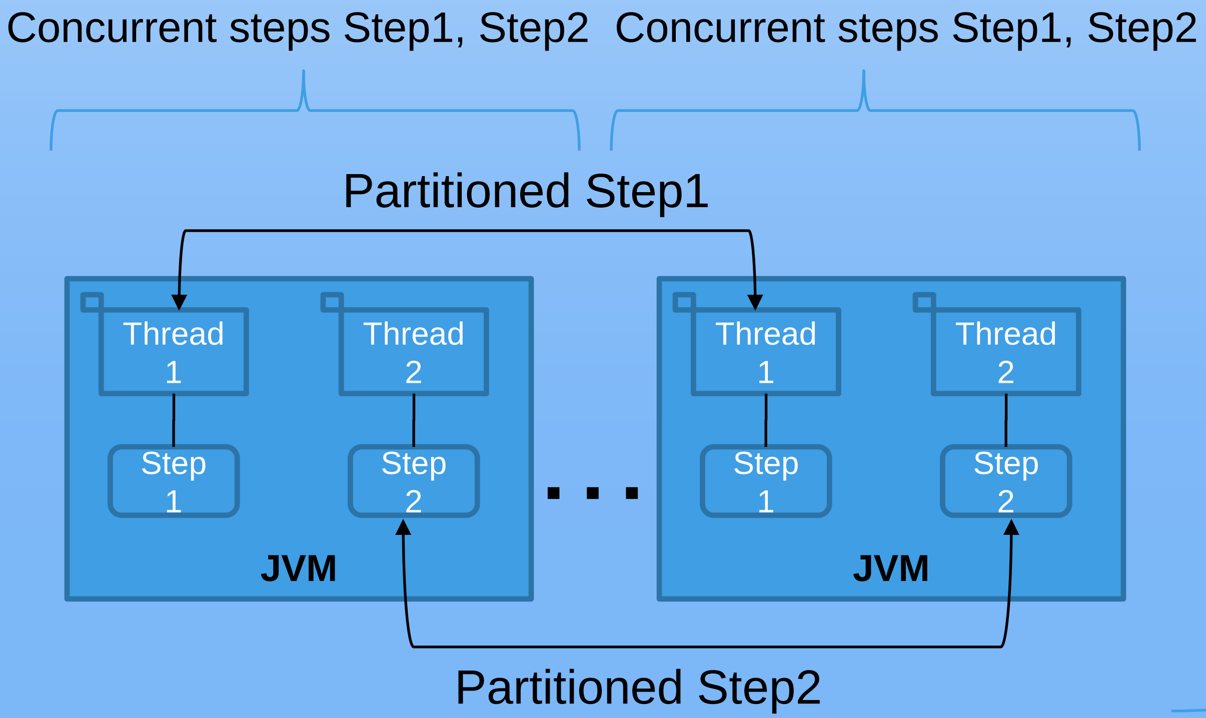
```
@Job(name="Job1")
@ExecutionOrder=(“Group1”)
public class MyJob {
        @Step(name="Step1")  MyStep myStep;
        @Step(name="Step2")  MyOtherStep myOtherStep;
        @StepGroup(name=“Group1”)
                **@Parallel(partition=true,concurrent=true)**
                        String[]={“Step1”,”Step2”}
}
```

Discussion: Parallel Annotations

Partitioned Concurrent Steps, Multiple JVMs

Each group of concurrent steps runs partitioned across multiple JVMs. The key requirement is same-JVM proximity for the concurrent steps, and then partitioning of each step group.

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



Discussion: Parallel Annotations

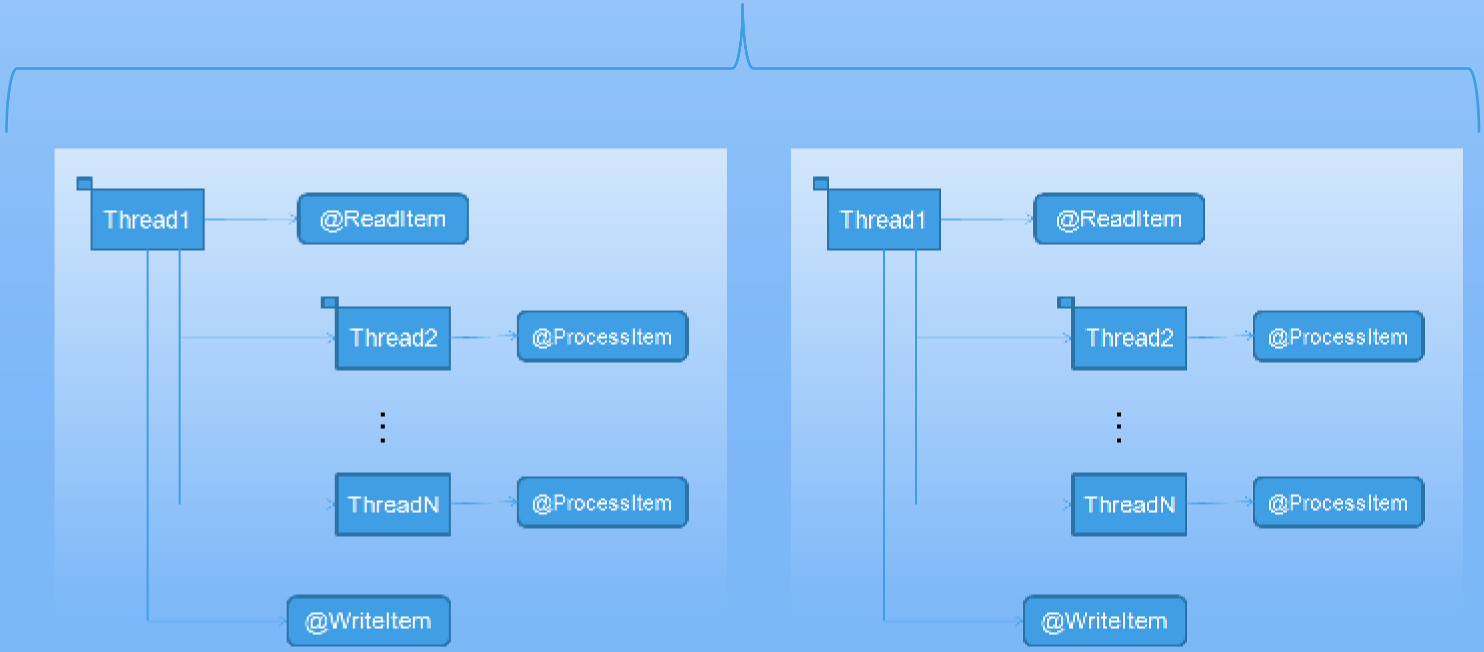
Partitioned Concurrent Steps, Multiple JVMs

```
@Job(name="Job1")
@ExecutionOrder=(“Group1”)
public class MyJob {
            @Step(name="Step1")  MyStep myStep;
            @Step(name="Step2")  MyOtherStep myOtherStep;
            @StepGroup(name=“Group1”)
            @Parallel(partition=true,concurrent=true,
                                jvms=JVMs.MULTIPLE)
                                            String[]={“Step1”,”Step2”}
}
```

Discussion: Parallel Annotations

Partitioned pipelined Step

Multiple instances of Step1 started at
same time on separate threads



Discussion: Parallel Annotations

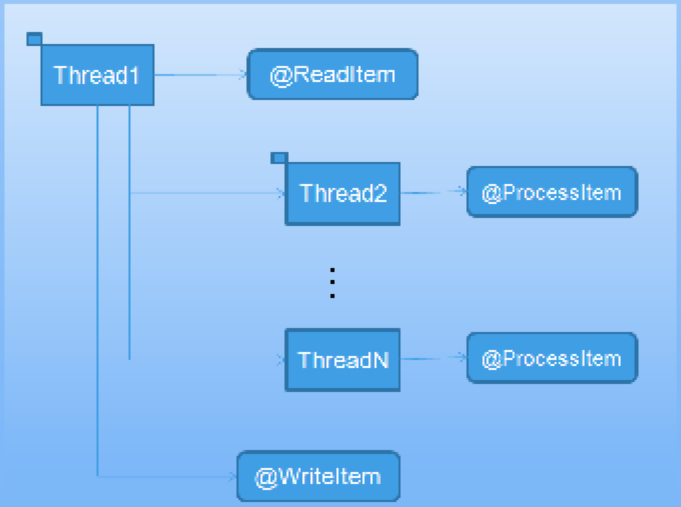
Partitioned pipelined Step

```
@Job(name="Job1")
public class MyJob {
        @Step(name="Step1")
                @Parallel(partition=true,pipeline=true)
                        MyStep myStep;
}
```

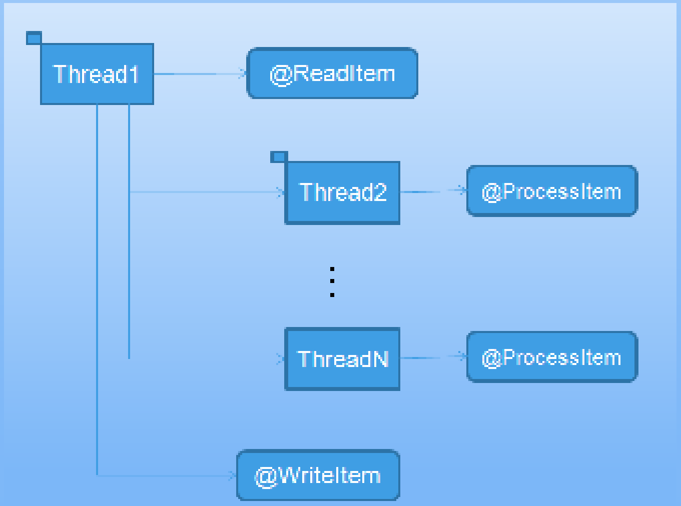
Discussion: Parallel Annotations

Concurrent Pipelined Step

Step1



Step2



Discussion: Parallel Annotations

Concurrent Pipelined Step

```
@Job(name="Job1")
@ExecutionOrder(“Group1”)
public class MyJob {
        @Step(name="Step1")
                @Parallel(pipeline=true)
                        MyStep myStep;
        @Step(name="Step2")
                @Parallel(pipeline=true)
                        MyOtherStep myOtherStep;
        @StepGroup(name=“Group1")
                @Parallel(concurrent=true)
                        String[]={“Step1”,”Step2”}
}
```

Discussion: Job Context

- Runtime object that communicates state of current job execution
- Injected by Runtime via annotation
- Holds following information:
  - Job
    - name, parameters
    - End state, return code
    - Metrics
    - Transient and persistent “properties” bags
  - Per step
    - Name, parameters
    - End state, return code
    - Metrics
    - Transient and persistent “properties” bags

Discussion: Job Context

```
package jsr352.example;
import javax.batch.runtime.JobContext;
@Job(name="Job1")
public class MyJob {
        @Context JobContext jobCtx;
        @Step(name="Step1") MyStep myStep;
        @BeforeJob void begin() {
                System.out.println("Job name=",jobCtx.getJobName());
        }
}
```

Discussion: Job Context

```
package jsr352.example;
import javax.batch.runtime.JobContext;
import javax.batch.runtime.StepContext;
@Job(name=“Job1”)
@ExecutionOrder({“Step1”,”Step2”})
public class MyJob {
        @Context JobContext jobCtx;
        @Step(name=“Step1”) MyStep myStep;
        @Step(name=“Step2”) MyOtherStep myOtherStep;
        @BeforeJob void begin() {
                StepContext stepCtx= jobCtx.getCurrentStepContext();
                System.out.println(“Step name=“,jobCtx.getStepName());
        }
}
```

List for Next Meeting

- Partition and Parallel Communication Annotations
- Future
  - Exit codes
  - Step conditions
  - Metrics
  - Java EE