Parallel and Generic Pipe-and-Filter Architectures with TeeTime

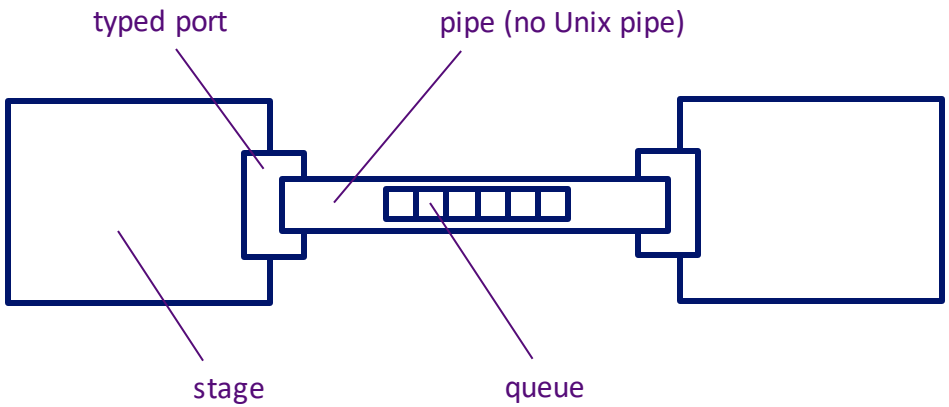
ICSA 2017 – Research Tool Paper

06.04.2017

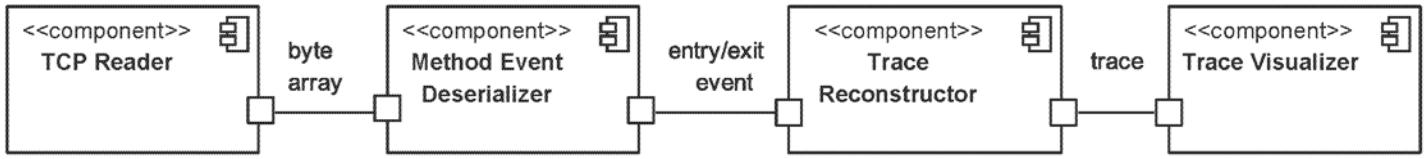
Christian Wulf,
Wilhelm Hasselbring, and
Johannes Ohlemacher

Kiel University, Germany

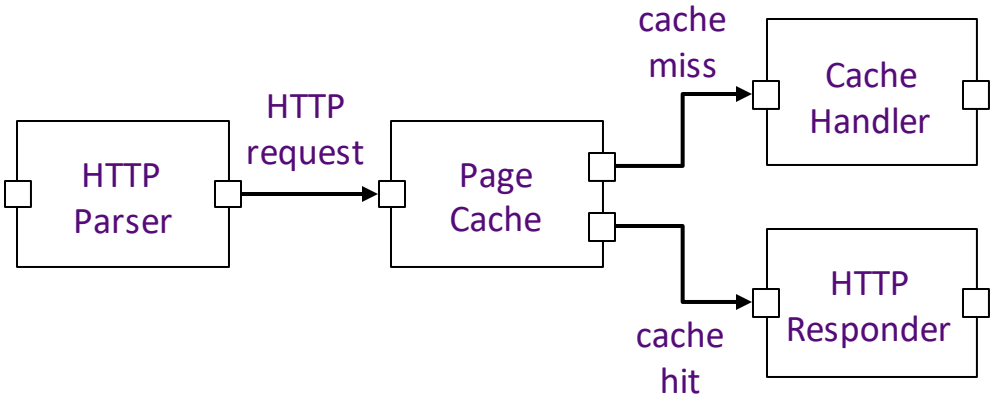
Motivation



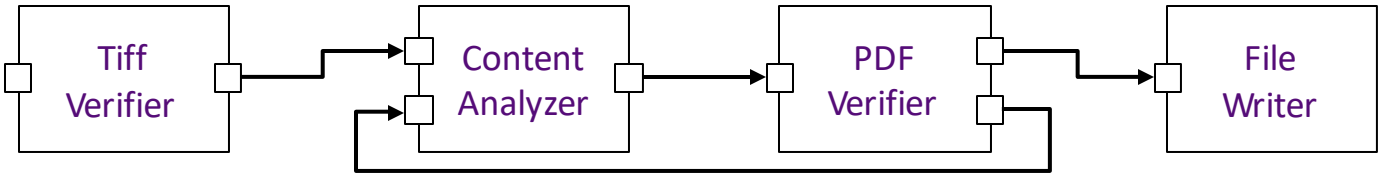
Motivation


<<component>>
TCP Reader
byte
array
<<component>>
Method Event
Deserializer
entry/exit
event
<<component>>
Trace
Reconstructor
trace
<<component>>
Trace Visualizer

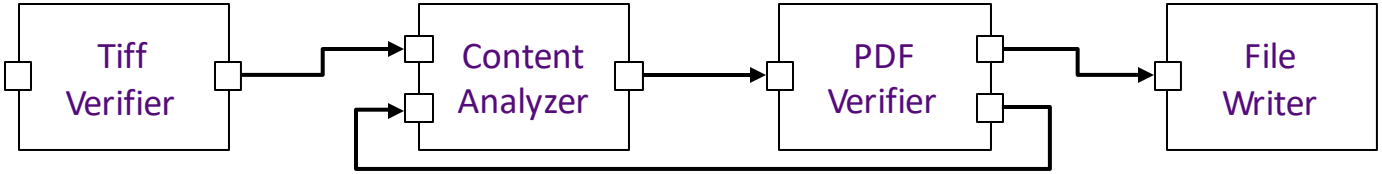
Motivation


HTTP
Parser
HTTP
request
Page
Cache
cache
miss
Cache
Handler
cache
hit
HTTP
Responder

Motivation

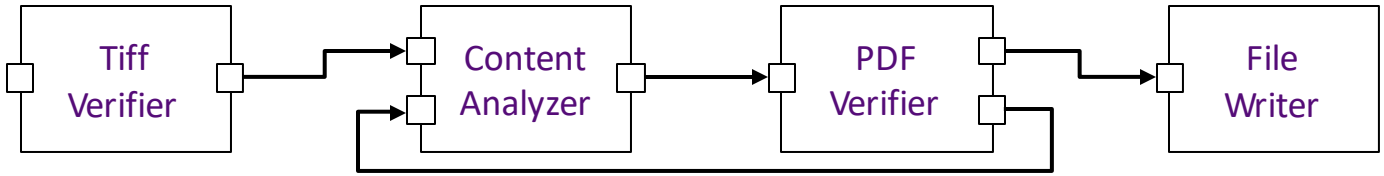


Motivation



- No P&F Framework available
  - StreamIT [Gordon2006]: at most one input/output port per stage
  - FastFlow [Aldinucci2014]: no passive stages
  - Akka [AkkaWebsite]: no type safety at compile time

Motivation

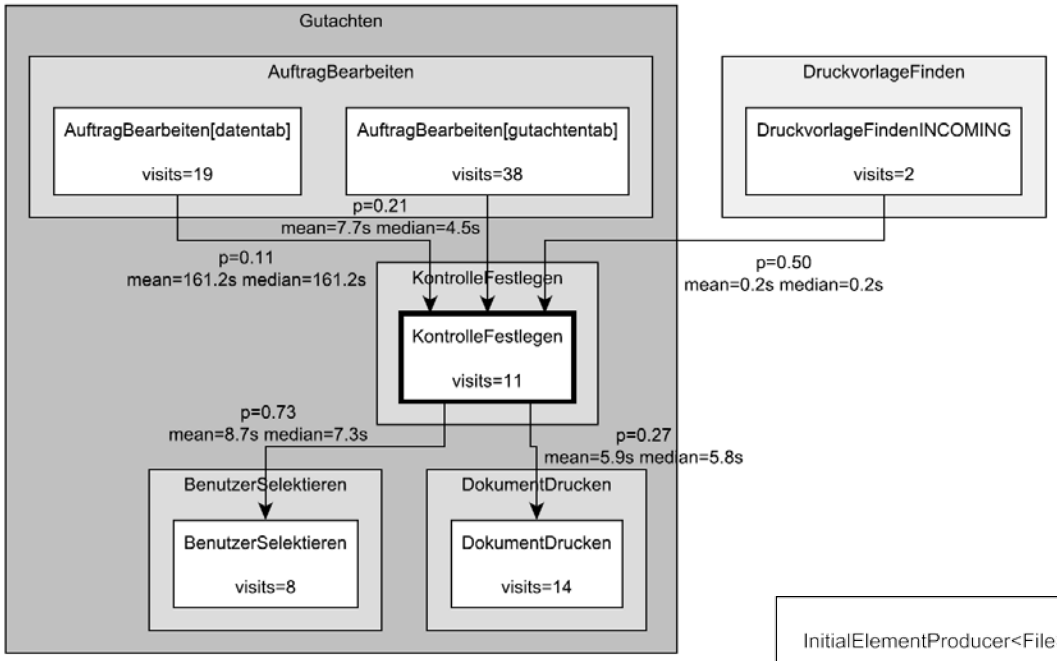


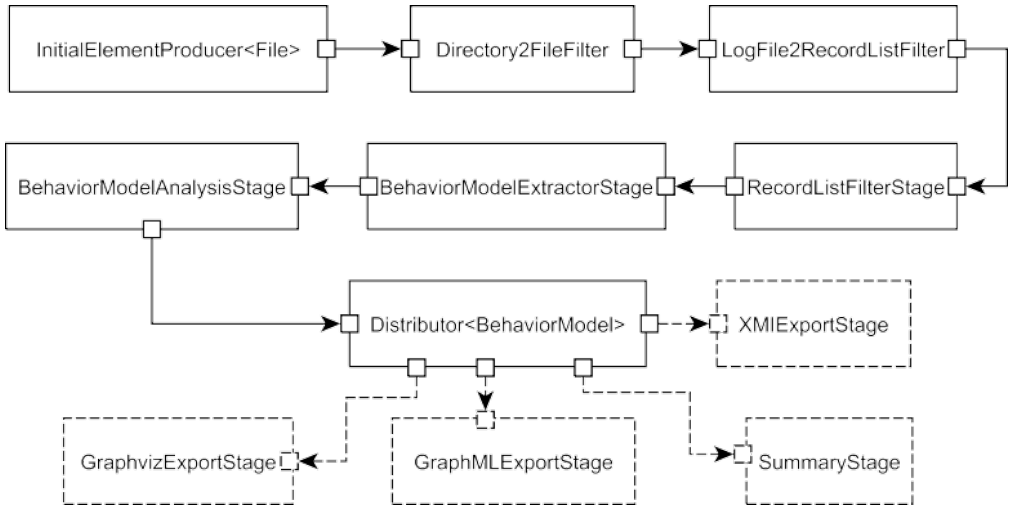
- No P&F Framework available
  - StreamIT [Gordon2006]: at most one input/output port per stage
  - FastFlow [Aldinucci2014]: no passive stages
  - Akka [AkkaWebsite]: no type safety at compile time
- Efficient parallel execution for P&F is still WIP

[Gordon2006, Sugerman2009, Suleman2010]

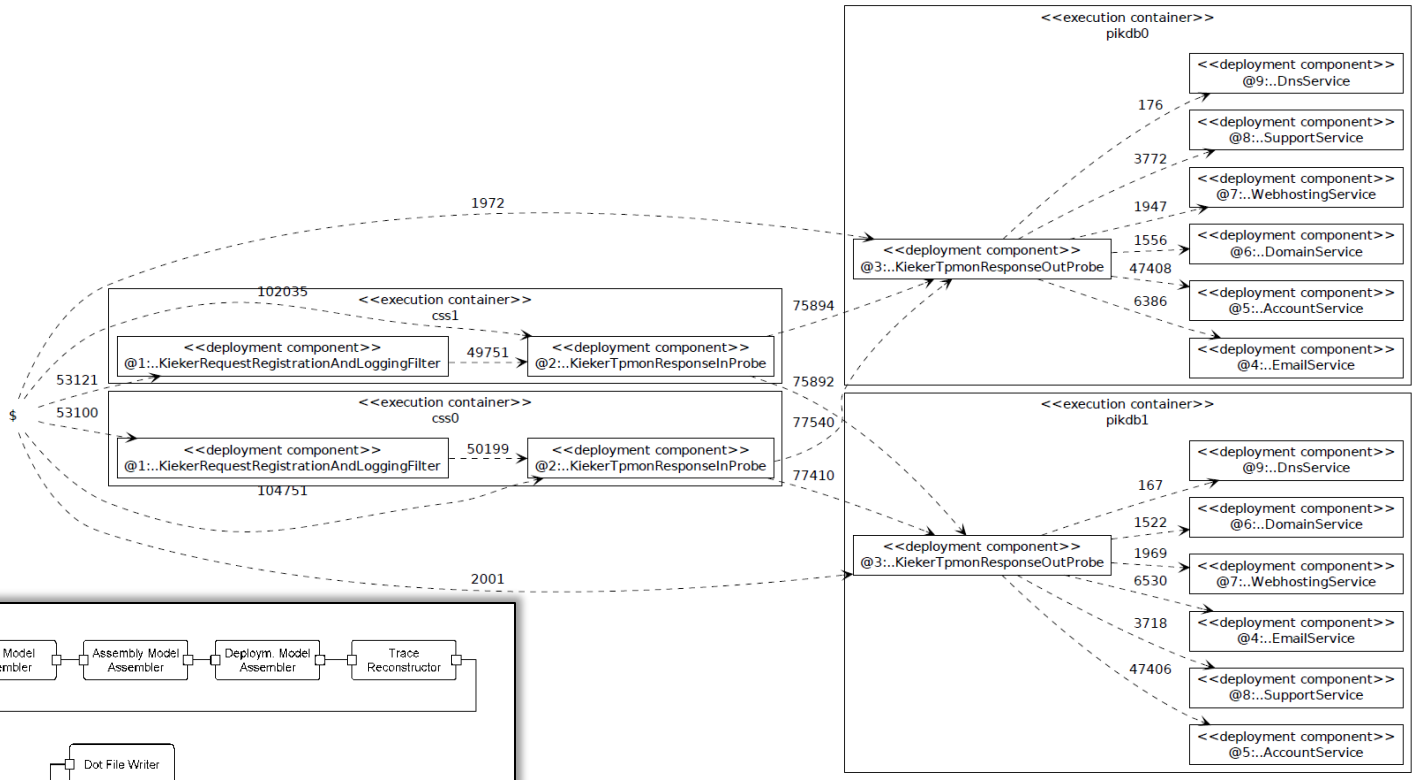
Some Fields of Application

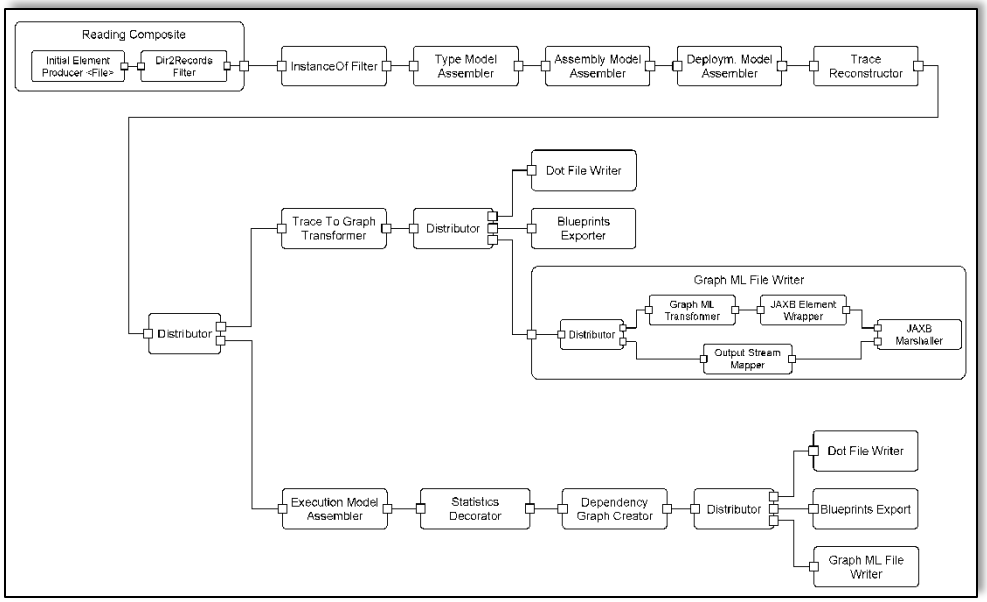
Some Fields of Application





Some Fields of Application





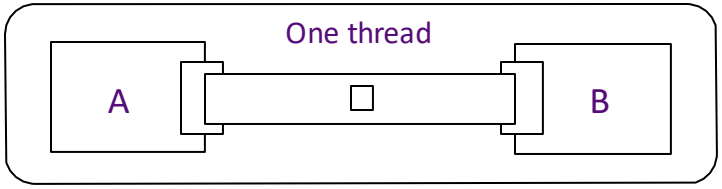
Selected Features

TeeTime

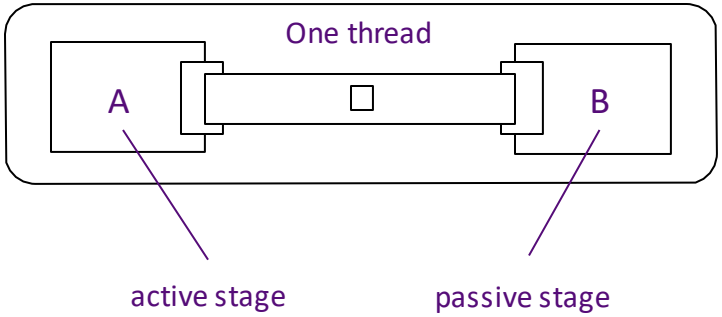
http://teetime-framework.github.io

- More than one input/output port per stage
  => for all P&F manifestations
- High-performance, parallel execution
  => for multi-core systems
- Passive stages
  => for modularization
- Custom pipes and schedulers
  => for experimentations with the P&F style

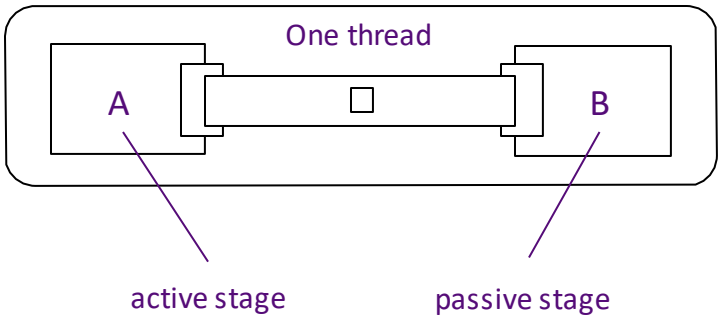
Framework Architecture

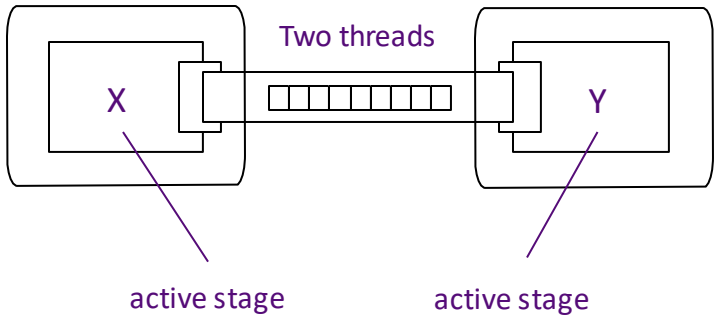


Framework Architecture

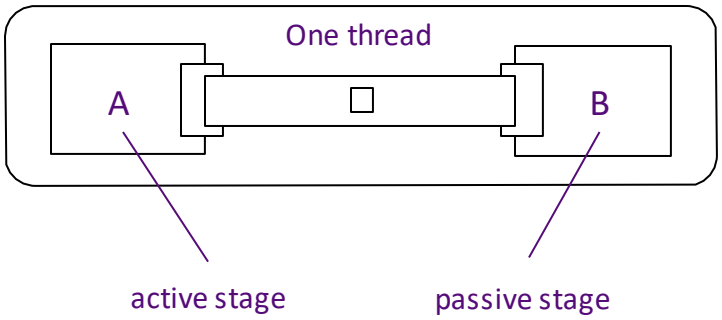


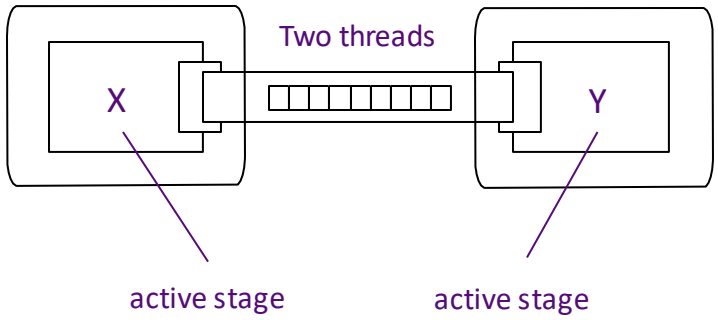
Framework Architecture

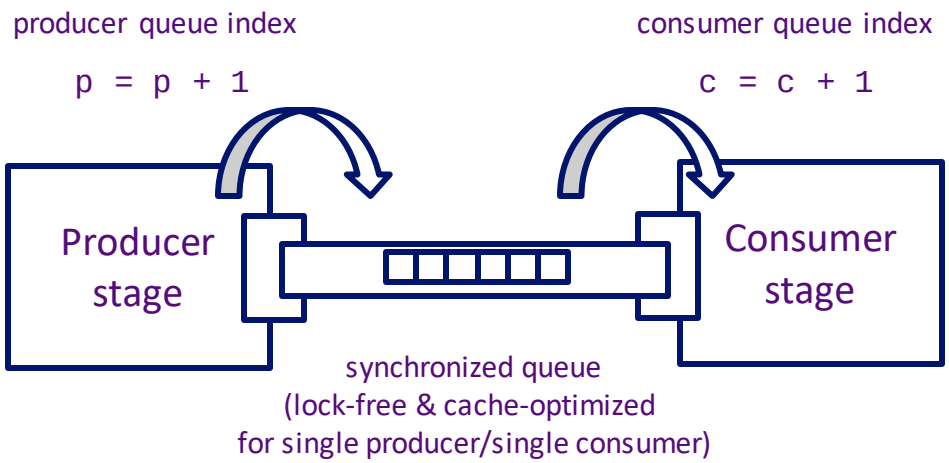




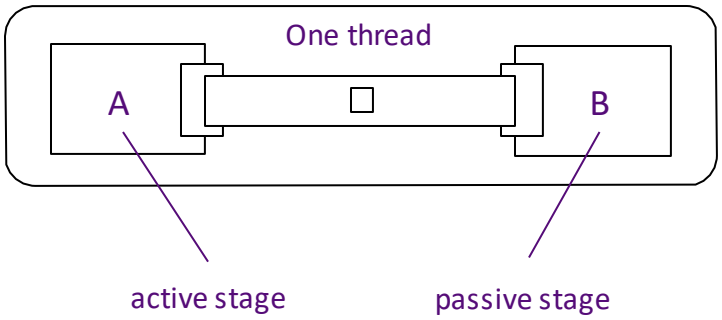
Framework Architecture

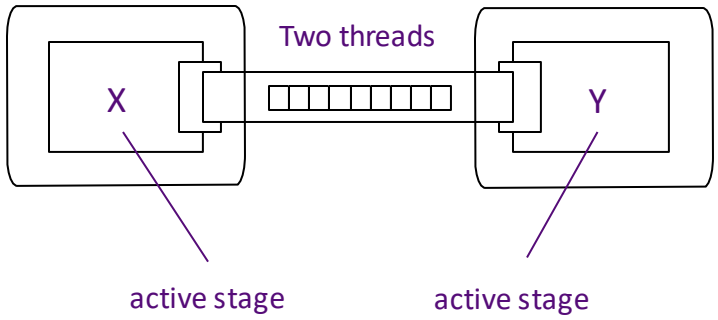


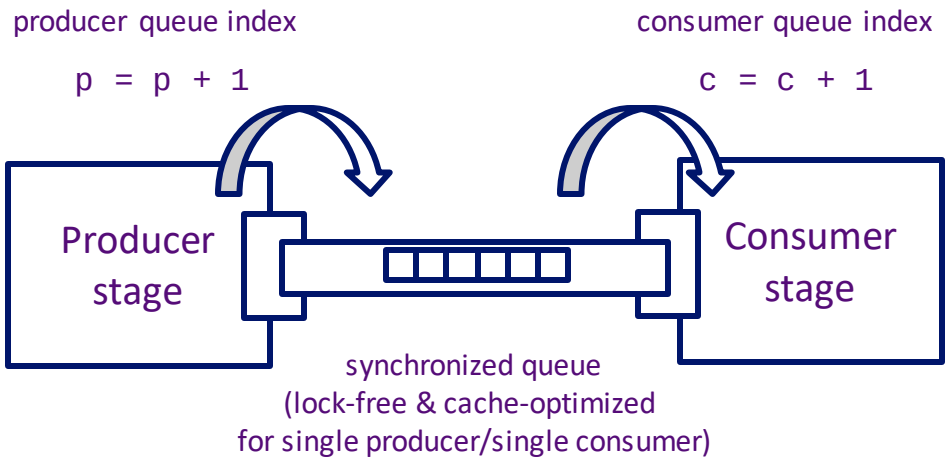




Framework Architecture

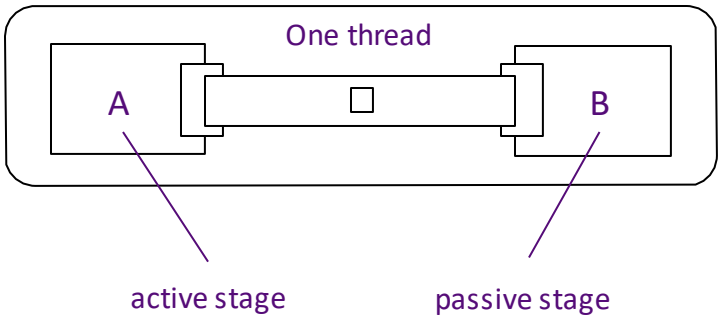


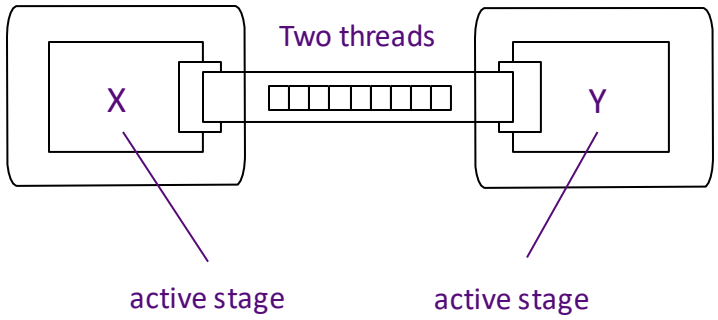


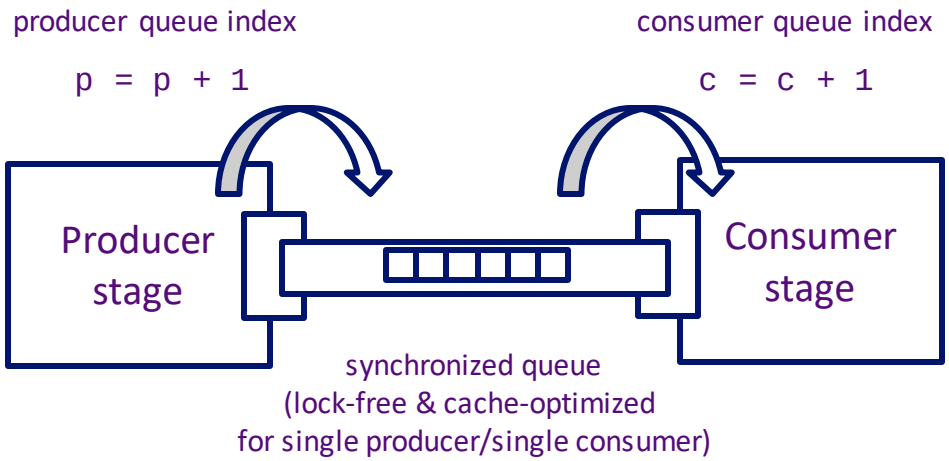


Technology	Throughput
1 Gbit-network	1.0 billion b/s
Sp/Sc queue	6.4 billion b/s

Framework Architecture



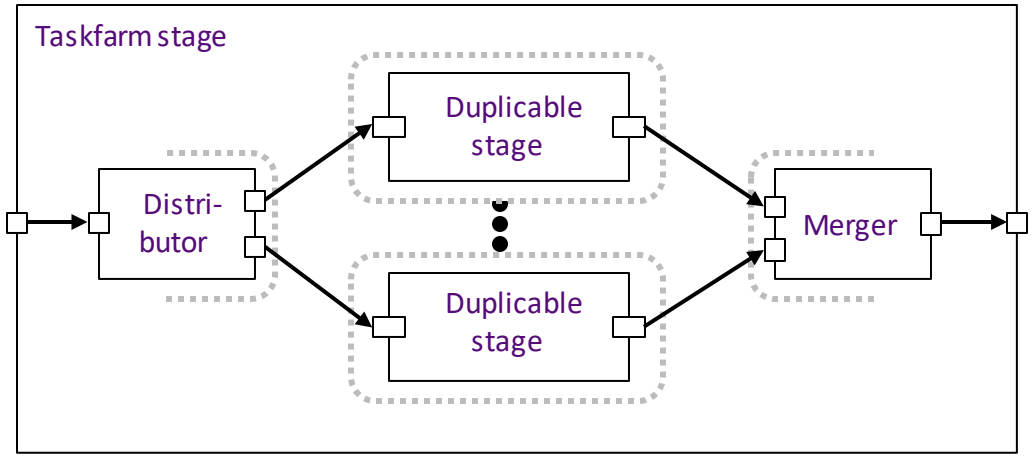




Technology	Throughput
1 Gbit-network	1.0 billion b/s
Sp/Sc queue	6.4 billion b/s

=> much potential for optimization on a single node

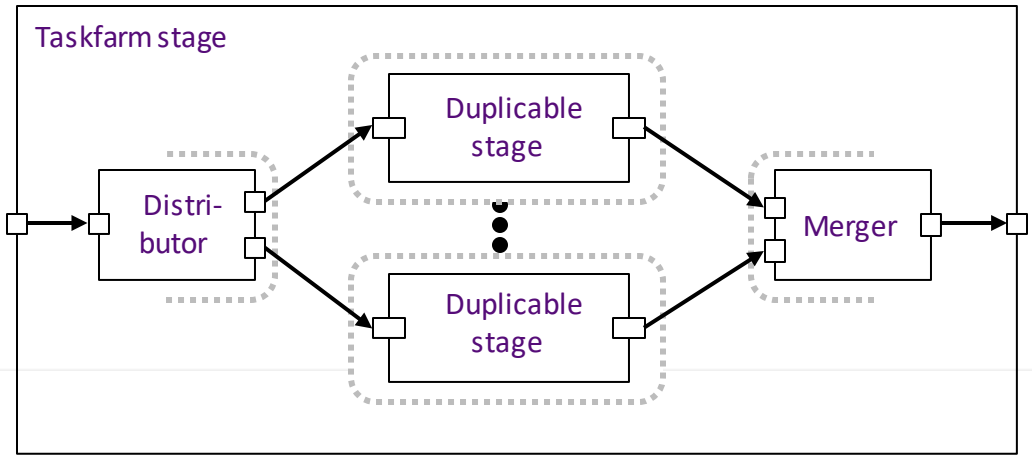
Parallel, Composite Stage



Taskfarm Stage [Wulf2016]

Parallel, Composite Stage

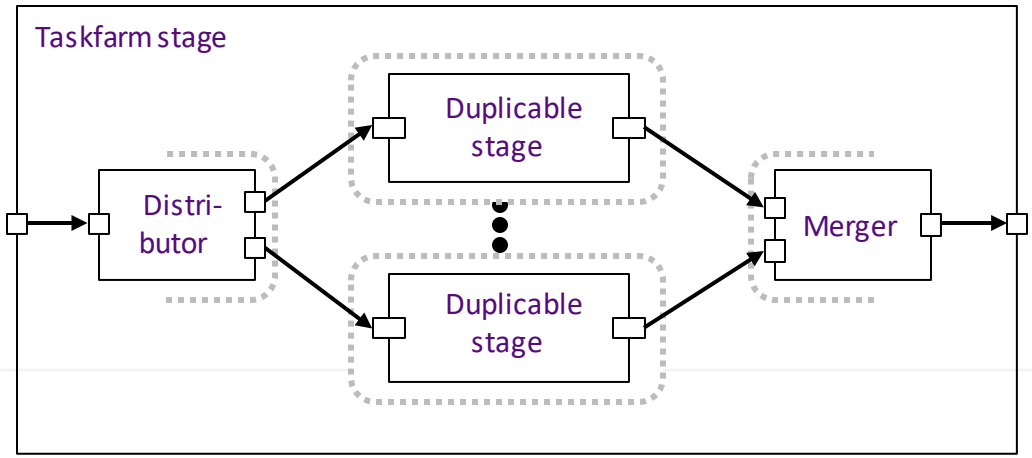

```
 99 class StaticTaskFarmStage<I, O, T extends ITaskFarmDuplicable<I, O>> extends CompositeStage {
100
101     private final Distributor<I> distributor = new Distributor<>();
102     private final Merger<O> merger = new Merger<>();
103
104     protected StaticTaskFarmStage(final T workerStage, final int numberStages, final int pipeCapacity) {
105         for (int i = 0; i < numberStages; i++) {
106             ITaskFarmDuplicable<I, O> duplicatedWorkerStage = workerStage.duplicate();
107
108             connectPorts(this.distributor.getNewOutputPort(), duplicatedWorkerStage.getInputPort(), pipeCapacity);
109             connectPorts(duplicatedWorkerStage.getOutputPort(), this.merger.getNewInputPort(), pipeCapacity);
110
111             duplicatedWorkerStage.getInputPort().getOwningStage().declareActive();
112         }
113
114         if (numberStages > 1) {
115             this.merger.declareActive();
116         }
117     }
118
119     public InputPort<I> getInputPort() {
120         return distributor.getInputPort();
121     }
122
123     public OutputPort<O> getOutputPort() {
124         return merger.getOutputPort();
125     }
126 }
```



Taskfarm Stage [Wulf2016]

Parallel, Composite Stage

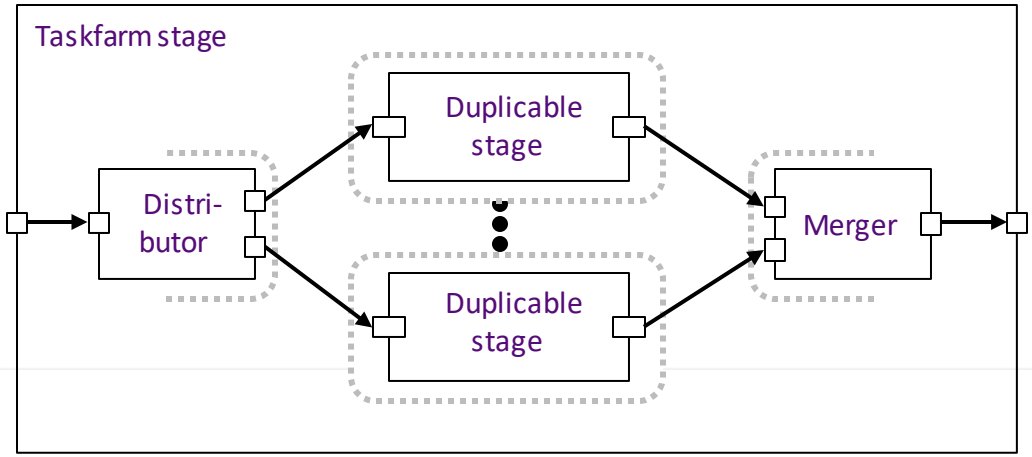

```
99  class StaticTaskFarmStage<I, O, T extends ITaskFarmDuplicable<I, O>> extends CompositeStage {
100
101     private final Distributor<I> distributor = new Distributor<>();
102     private final Merger<O> merger = new Merger<>();
103
104     protected StaticTaskFarmStage(final T workerStage, final int numberStages, final int pipeCapacity) {
105         for (int i = 0; i < numberStages; i++) {
106             ITaskFarmDuplicable<I, O> duplicatedWorkerStage = workerStage.duplicate();
107
108             connectPorts(this.distributor.getNewOutputPort(), duplicatedWorkerStage.getInputPort(), pipeCapacity);
109             connectPorts(duplicatedWorkerStage.getOutputPort(), this.merger.getNewInputPort(), pipeCapacity);
110
111             duplicatedWorkerStage.getInputPort().getOwningStage().declareActive();
112         }
113
114         if (numberStages > 1) {
115             this.merger.declareActive();
116         }
117     }
118
119     public InputPort<I> getInputPort() {
120         return distributor.getInputPort();
121     }
122
123     public OutputPort<O> getOutputPort() {
124         return merger.getOutputPort();
125     }
126 }
```



Taskfarm Stage [Wulf2016]

Parallel, Composite Stage

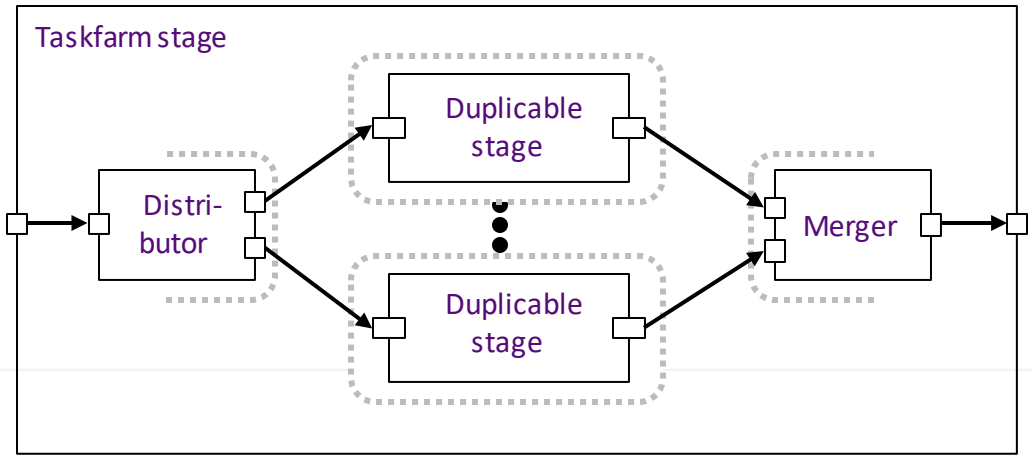

```
99  class StaticTaskFarmStage<I, O, T extends ITaskFarmDuplicable<I, O>> extends CompositeStage {
100
101     private final Distributor<I> distributor = new Distributor<>();
102     private final Merger<O> merger = new Merger<>();
103
104     protected StaticTaskFarmStage(final T workerStage, final int numberStages, final int pipeCapacity) {
105         for (int i = 0; i < numberStages; i++) {
106             ITaskFarmDuplicable<I, O> duplicatedWorkerStage = workerStage.duplicate();
107
108             connectPorts(this.distributor.getNewOutputPort(), duplicatedWorkerStage.getInputPort(), pipeCapacity);
109             connectPorts(duplicatedWorkerStage.getOutputPort(), this.merger.getNewInputPort(), pipeCapacity);
110
111             duplicatedWorkerStage.getInputPort().getOwningStage().declareActive();
112         }
113
114         if (numberStages > 1) {
115             this.merger.declareActive();
116         }
117     }
118
119     public InputPort<I> getInputPort() {
120         return distributor.getInputPort();
121     }
122
123     public OutputPort<O> getOutputPort() {
124         return merger.getOutputPort();
125     }
126 }
```



Taskfarm Stage [Wulf2016]

Parallel, Composite Stage

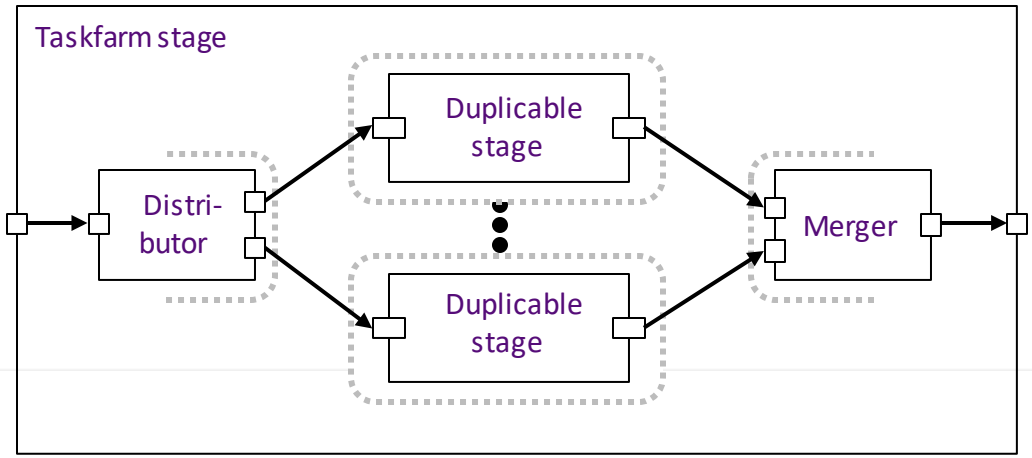

```
99  class StaticTaskFarmStage<I, O, T extends ITaskFarmDuplicable<I, O>> extends CompositeStage {
100
101     private final Distributor<I> distributor = new Distributor<>();
102     private final Merger<O> merger = new Merger<>();
103
104     protected StaticTaskFarmStage(final T workerStage, final int numberStages, final int pipeCapacity) {
105         for (int i = 0; i < numberStages; i++) {
106             ITaskFarmDuplicable<I, O> duplicatedWorkerStage = workerStage.duplicate();
107
108             connectPorts(this.distributor.getNewOutputPort(), duplicatedWorkerStage.getInputPort(), pipeCapacity);
109             connectPorts(duplicatedWorkerStage.getOutputPort(), this.merger.getNewInputPort(), pipeCapacity);
110
111             duplicatedWorkerStage.getInputPort().getOwningStage().declareActive();
112         }
113
114         if (numberStages > 1) {
115             this.merger.declareActive();
116         }
117     }
118
119     public InputPort<I> getInputPort() {
120         return distributor.getInputPort();
121     }
122
123     public OutputPort<O> getOutputPort() {
124         return merger.getOutputPort();
125     }
126 }
```



Taskfarm Stage [Wulf2016]

Parallel, Composite Stage

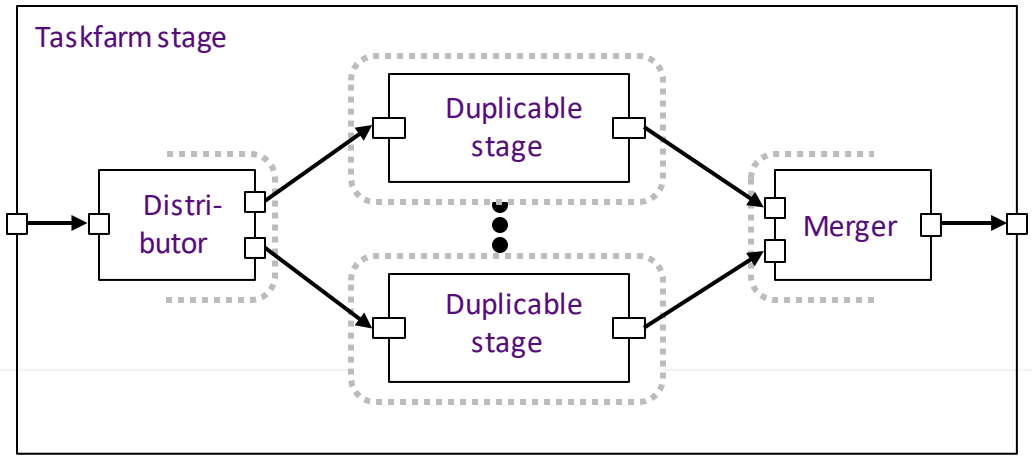

```
99  class StaticTaskFarmStage<I, O, T extends ITaskFarmDuplicable<I, O>> extends CompositeStage {
100
101     private final Distributor<I> distributor = new Distributor<>();
102     private final Merger<O> merger = new Merger<>();
103
104     protected StaticTaskFarmStage(final T workerStage, final int numberStages, final int pipeCapacity) {
105         for (int i = 0; i < numberStages; i++) {
106             ITaskFarmDuplicable<I, O> duplicatedWorkerStage = workerStage.duplicate();
107
108             connectPorts(this.distributor.getNewOutputPort(), duplicatedWorkerStage.getInputPort(), pipeCapacity);
109             connectPorts(duplicatedWorkerStage.getOutputPort(), this.merger.getNewInputPort(), pipeCapacity);
110
111             duplicatedWorkerStage.getInputPort().getOwningStage().declareActive();
112         }
113
114         if (numberStages > 1) {
115             this.merger.declareActive();
116         }
117     }
118
119     public InputPort<I> getInputPort() {
120         return distributor.getInputPort();
121     }
122
123     public OutputPort<O> getOutputPort() {
124         return merger.getOutputPort();
125     }
126 }
```



Taskfarm Stage [Wulf2016]

Parallel, Composite Stage

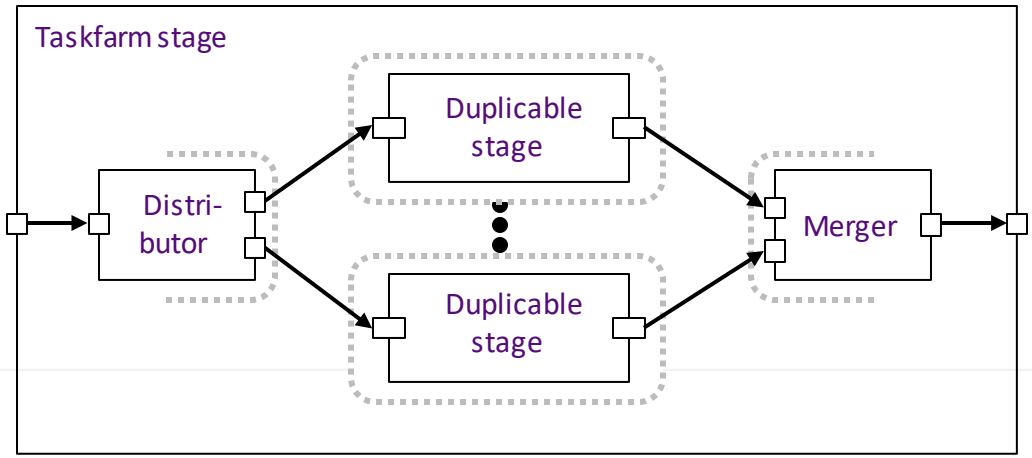

```
99  class StaticTaskFarmStage<I, O, T extends ITaskFarmDuplicable<I, O>> extends CompositeStage {
100
101     private final Distributor<I> distributor = new Distributor<>();
102     private final Merger<O> merger = new Merger<>();
103
104     protected StaticTaskFarmStage(final T workerStage, final int numberStages, final int pipeCapacity) {
105         for (int i = 0; i < numberStages; i++) {
106             ITaskFarmDuplicable<I, O> duplicatedWorkerStage = workerStage.duplicate();
107
108             connectPorts(this.distributor.getNewOutputPort(), duplicatedWorkerStage.getInputPort(), pipeCapacity);
109             connectPorts(duplicatedWorkerStage.getOutputPort(), this.merger.getNewInputPort(), pipeCapacity);
110
111             duplicatedWorkerStage.getInputPort().getOwningStage().declareActive();
112         }
113
114         if (numberStages > 1) {
115             this.merger.declareActive();
116         }
117     }
118
119     public InputPort<I> getInputPort() {
120         return distributor.getInputPort();
121     }
122
123     public OutputPort<O> getOutputPort() {
124         return merger.getOutputPort();
125     }
126 }
```



Taskfarm Stage [Wulf2016]

Parallel, Composite Stage

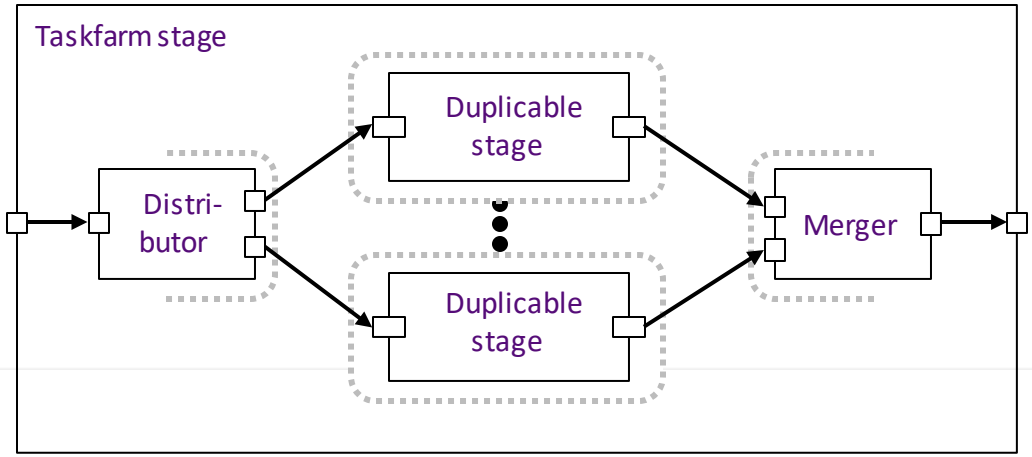

```
99  class StaticTaskFarmStage<I, O, T extends ITaskFarmDuplicable<I, O>> extends CompositeStage {
100
101     private final Distributor<I> distributor = new Distributor<>();
102     private final Merger<O> merger = new Merger<>();
103
104     protected StaticTaskFarmStage(final T workerStage, final int numberStages, final int pipeCapacity) {
105         for (int i = 0; i < numberStages; i++) {
106             ITaskFarmDuplicable<I, O> duplicatedWorkerStage = workerStage.duplicate();
107
108             connectPorts(this.distributor.getNewOutputPort(), duplicatedWorkerStage.getInputPort(), pipeCapacity);
109             connectPorts(duplicatedWorkerStage.getOutputPort(), this.merger.getNewInputPort(), pipeCapacity);
110
111             duplicatedWorkerStage.getInputPort().getOwningStage().declareActive();
112         }
113
114         if (numberStages > 1) {
115             this.merger.declareActive();
116         }
117     }
118
119     public InputPort<I> getInputPort() {
120         return distributor.getInputPort();
121     }
122
123     public OutputPort<O> getOutputPort() {
124         return merger.getOutputPort();
125     }
126 }
```



Taskfarm Stage [Wulf2016]

Parallel, Composite Stage


```
99  class StaticTaskFarmStage<I, O, T extends ITaskFarmDuplicable<I, O>> extends CompositeStage {
100
101     private final Distributor<I> distributor = new Distributor<>();
102     private final Merger<O> merger = new Merger<>();
103
104     protected StaticTaskFarmStage(final T workerStage, final int numberStages, final int pipeCapacity) {
105         for (int i = 0; i < numberStages; i++) {
106             ITaskFarmDuplicable<I, O> duplicatedWorkerStage = workerStage.duplicate();
107
108             connectPorts(this.distributor.getNewOutputPort(), duplicatedWorkerStage.getInputPort(), pipeCapacity);
109             connectPorts(duplicatedWorkerStage.getOutputPort(), this.merger.getNewInputPort(), pipeCapacity);
110
111             duplicatedWorkerStage.getInputPort().getOwningStage().declareActive();
112         }
113
114         if (numberStages > 1) {
115             this.merger.declareActive();
116         }
117     }
118
119     public InputPort<I> getInputPort() {
120         return distributor.getInputPort();
121     }
122
123     public OutputPort<O> getOutputPort() {
124         return merger.getOutputPort();
125     }
126 }
```



Taskfarm Stage [Wulf2016]

Conclusion

- Supports all P&F manifestations
- Tuned for a parallel execution on multi-core systems
- Type safe connection at compile time
- Open for modifications & experimentations

Java [TeeTimeJava]

TeeTime ≡

C++ [TeeTimeC++]

http://teetime-framework.github.io

- Fault-tolerant distributed execution with Akka
- Live visualization for performance diagnoses
- Comparison with other frameworks

Selected References

[AkkaWebsite]

[Aldinucci2014]

[Gordon2006]

[Sugerman2009]

[Suleman2010]

[TeeTimeC++]

[TeeTimeJava]

[Wulf2016]

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 BufferedUnsynchronizedPipeFactory.INSTANCE);
33
34         producer.declareActive();
35         asciiDirReader.declareActive();
36         binaryDirReader.declareActive();
37         merger.declareActive();
38     }
39 }
```

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 BufferedUnsynchronizedPipeFactory.INSTANCE);
33
34         producer.declareActive();
35         asciiDirReader.declareActive();
36         binaryDirReader.declareActive();
37         merger.declareActive();
38     }
39 }
```

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 BufferedUnsynchronizedPipeFactory.INSTANCE);
33
34         producer.declareActive();
35         asciiDirReader.declareActive();
36         binaryDirReader.declareActive();
37         merger.declareActive();
38     }
39 }
```

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 BufferedUnsynchronizedPipeFactory.INSTANCE);
33
34         producer.declareActive();
35         asciiDirReader.declareActive();
36         binaryDirReader.declareActive();
37         merger.declareActive();
38     }
39 }
```

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 BufferedUnsynchronizedPipeFactory.INSTANCE);
33
34         producer.declareActive();
35         asciiDirReader.declareActive();
36         binaryDirReader.declareActive();
37         merger.declareActive();
38     }
39 }
```

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 BufferedUnsynchronizedPipeFactory.INSTANCE);
33
34         producer.declareActive();
35         asciiDirReader.declareActive();
36         binaryDirReader.declareActive();
37         merger.declareActive();
38     }
39 }
```

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 Buff
33
34         producer.dec
35         asciiDirRead
36         binaryDirRea
37         merger.decla
38     }
39 }
```

```
42     public static void main(final String[] args) {
43         Path[] dirs = { Paths.get("a/b/c-ascii"), Paths.get("x/y/z-binary") };
44         LogReaderConfig config = new LogReaderConfig(dirs);
45         Execution<LogReaderConfig> execution = new Execution<>(config);
46         execution.executeNonBlocking();
47         /// ...
48         execution.waitForTermination();
49         records = config.sink.getElements();
50     }
```

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 Buff
33
34         producer.dec
35         asciiDirRead
36         binaryDirRea
37         merger.decla
38     }
39 }
```

```
42     public static void main(final String[] args) {
43         Path[] dirs = { Paths.get("a/b/c-ascii"), Paths.get("x/y/z-binary") };
44         LogReaderConfig config = new LogReaderConfig(dirs);
45         Execution<LogReaderConfig> execution = new Execution<>(config);
46         execution.executeNonBlocking();
47         /// ...
48         execution.waitForTermination();
49         records = config.sink.getElements();
50     }
```

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 Buff
33
34         producer.dec
35         asciiDirRead
36         binaryDirRea
37         merger.decla
38     }
39 }
```

```
42     public static void main(final String[] args) {
43         Path[] dirs = { Paths.get("a/b/c-ascii"), Paths.get("x/y/z-binary") };
44         LogReaderConfig config = new LogReaderConfig(dirs);
45         Execution<LogReaderConfig> execution = new Execution<>(config);
46         execution.executeNonBlocking();
47         /// ...
48         execution.waitForTermination();
49         records = config.sink.getElements();
50     }
```

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 Buff
33
34         producer.dec
35         asciiDirRead
36         binaryDirRea
37         merger.decla
38     }
39 }
```

```
42     public static void main(final String[] args) {
43         Path[] dirs = { Paths.get("a/b/c-ascii"), Paths.get("x/y/z-binary") };
44         LogReaderConfig config = new LogReaderConfig(dirs);
45         Execution<LogReaderConfig> execution = new Execution<>(config);
46         execution.executeNonBlocking();
47         /// ...
48         execution.waitForTermination();
49         records = config.sink.getElements();
50     }
```

Example P&F Configuration


```
14 public class LogReaderConfig extends Configuration {
15
16     final CollectorSink<Record> sink;
17
18     public LogReaderConfig(final List<Path> dirs) {
19         InitialElementProducer<Path> producer = new InitialElementProducer<>(dirs);
20         KiekerLogDirSwitch kiekerLogDirSwitch = new KiekerLogDirSwitch();
21         AsciiLogDirReader asciiDirReader = new AsciiLogDirReader();
22         BinaryLogDirReader binaryDirReader = new BinaryLogDirReader();
23         Merger<Record> merger = new Merger<>();
24         this.sink = new CollectorSink<>();
25
26         connectPorts(producer.getOutputPort(), kiekerLogDirSwitch.getInputPort());
27         connectPorts(kiekerLogDirSwitch.getAsciiPort(), asciiDirReader.getInputPort());
28         connectPorts(kiekerLogDirSwitch.getBinaryPort(), binaryDirReader.getInputPort());
29         connectPorts(asciiDirReader.getOutputPort(), merger.getNewInputPort());
30         connectPorts(binaryDirReader.getOutputPort(), merger.getNewInputPort());
31         connectPorts(merger.getOutputPort(), this.sink.getInputPort(),
32                 Buff
33
34         producer.dec
35         asciiDirRead
36         binaryDirRea
37         merger.decla
38     }
39 }
```

```
42     public static void main(final String[] args) {
43         Path[] dirs = { Paths.get("a/b/c-ascii"), Paths.get("x/y/z-binary") };
44         LogReaderConfig config = new LogReaderConfig(dirs);
45         Execution<LogReaderConfig> execution = new Execution<>(config);
46         execution.executeNonBlocking();
47         /// ...
48         execution.waitForTermination();
49         records = config.sink.getElements();
50     }
```

Example P&F Stage


```
53 class KiekerLogDirSwitch extends AbstractStage {
54     InputPort<Path> inputPort = super.createInputPort();
55     OutputPort<Path> asciiPort = super.createOutputPort();
56     OutputPort<Path> binPort = super.createOutputPort();
57     OutputPort<Path> elsePort = super.createOutputPort();
58
59     @Override
60     protected void execute() {
61         Path path = inputPort.receive();
62         if (path == null) {
63             return;
64         }
65         if (path.endsWith("ascii")) {
66             asciiPort.send(path);
67         } else if (path.endsWith("binary")) {
68             binPort.send(path);
69         } else {
70             elsePort.send(path);
71         }
72     }
73
74     public InputPort<Path> getInputPort() {
75         return inputPort;
76     }
77
78     // getter for each output port
79 }
```

Example P&F Stage


```
53 class KiekerLogDirSwitch extends AbstractStage {
54     InputPort<Path> inputPort = super.createInputPort();
55     OutputPort<Path> asciiPort = super.createOutputPort();
56     OutputPort<Path> binPort = super.createOutputPort();
57     OutputPort<Path> elsePort = super.createOutputPort();
58
59     @Override
60     protected void execute() {
61         Path path = inputPort.receive();
62         if (path == null) {
63             return;
64         }
65         if (path.endsWith("ascii")) {
66             asciiPort.send(path);
67         } else if (path.endsWith("binary")) {
68             binPort.send(path);
69         } else {
70             elsePort.send(path);
71         }
72     }
73
74     public InputPort<Path> getInputPort() {
75         return inputPort;
76     }
77
78     // getter for each output port
79 }
```

Example P&F Stage


```
53  class KiekerLogDirSwitch extends AbstractStage {
54      InputPort<Path> inputPort = super.createInputPort();
55      OutputPort<Path> asciiPort = super.createOutputPort();
56      OutputPort<Path> binPort = super.createOutputPort();
57      OutputPort<Path> elsePort = super.createOutputPort();
58
59      @Override
60      protected void execute() {
61          Path path = inputPort.receive();
62          if (path == null) {
63              return;
64          }
65          if (path.endsWith("ascii")) {
66              asciiPort.send(path);
67          } else if (path.endsWith("binary")) {
68              binPort.send(path);
69          } else {
70              elsePort.send(path);
71          }
72      }
73
74      public InputPort<Path> getInputPort() {
75          return inputPort;
76      }
77
78      // getter for each output port
79  }
```

Example P&F Stage


```
53  class KiekerLogDirSwitch extends AbstractStage {
54      InputPort<Path> inputPort = super.createInputPort();
55      OutputPort<Path> asciiPort = super.createOutputPort();
56      OutputPort<Path> binPort = super.createOutputPort();
57      OutputPort<Path> elsePort = super.createOutputPort();
58
59      @Override
60      protected void execute() {
61          Path path = inputPort.receive();
62          if (path == null) {
63              return;
64          }
65          if (path.endsWith("ascii")) {
66              asciiPort.send(path);
67          } else if (path.endsWith("binary")) {
68              binPort.send(path);
69          } else {
70              elsePort.send(path);
71          }
72      }
73
74      public InputPort<Path> getInputPort() {
75          return inputPort;
76      }
77
78      // getter for each output port
79  }
```

Example P&F Stage


```
53  class KiekerLogDirSwitch extends AbstractStage {
54      InputPort<Path> inputPort = super.createInputPort();
55      OutputPort<Path> asciiPort = super.createOutputPort();
56      OutputPort<Path> binPort = super.createOutputPort();
57      OutputPort<Path> elsePort = super.createOutputPort();
58
59      @Override
60      protected void execute() {
61          Path path = inputPort.receive();
62          if (path == null) {
63              return;
64          }
65          if (path.endsWith("ascii")) {
66              asciiPort.send(path);
67          } else if (path.endsWith("binary")) {
68              binPort.send(path);
69          } else {
70              elsePort.send(path);
71          }
72      }
73
74      public InputPort<Path> getInputPort() {
75          return inputPort;
76      }
77
78      // getter for each output port
79  }
```

Example P&F Stage


```
53  class KiekerLogDirSwitch extends AbstractStage {
54      InputPort<Path> inputPort = super.createInputPort();
55      OutputPort<Path> asciiPort = super.createOutputPort();
56      OutputPort<Path> binPort = super.createOutputPort();
57      OutputPort<Path> elsePort = super.createOutputPort();
58
59      @Override
60      protected void execute() {
61          Path path = inputPort.receive();
62          if (path == null) {
63              return;
64          }
65          if (path.endsWith("ascii")) {
66              asciiPort.send(path);
67          } else if (path.endsWith("binary")) {
68              binPort.send(path);
69          } else {
70              elsePort.send(path);
71          }
72      }
73
74      public InputPort<Path> getInputPort() {
75          return inputPort;
76      }
77
78      // getter for each output port
79  }
```

Example P&F Stage


```
53  class KiekerLogDirSwitch extends AbstractStage {
54      InputPort<Path> inputPort = super.createInputPort();
55      OutputPort<Path> asciiPort = super.createOutputPort();
56      OutputPort<Path> binPort = super.createOutputPort();
57      OutputPort<Path> elsePort = super.createOutputPort();
58
59      @Override
60      protected void execute() {
61          Path path = inputPort.receive();
62          if (path == null) {
63              return;
64          }
65          if (path.endsWith("ascii")) {
66              asciiPort.send(path);
67          } else if (path.endsWith("binary")) {
68              binPort.send(path);
69          } else {
70              elsePort.send(path);
71          }
72      }
73
74      public InputPort<Path> getInputPort() {
75          return inputPort;
76      }
77
78      // getter for each output port
79  }
```