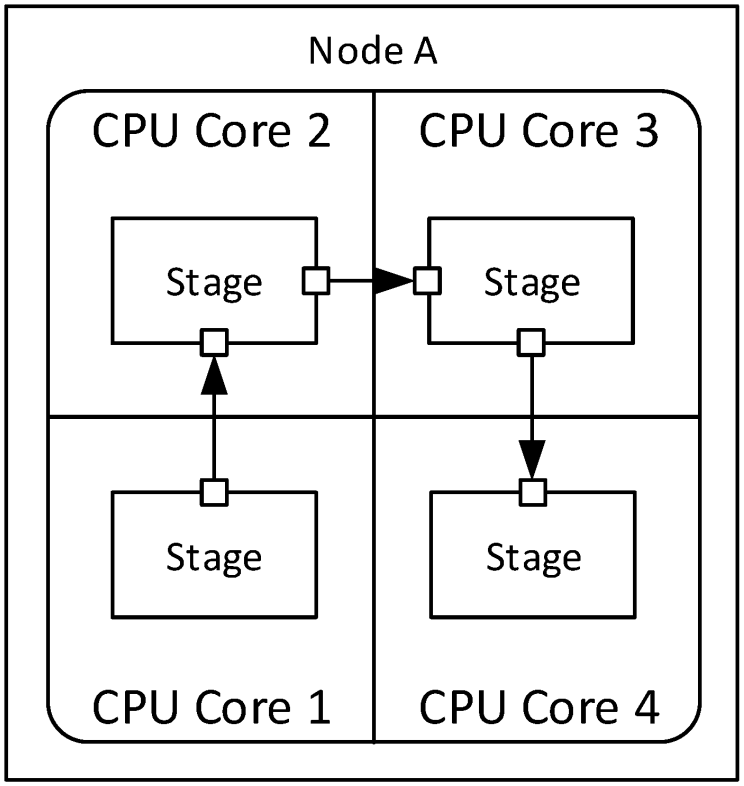
Distributed Pipe-and-Filter Architectures with TeeTime

Master's thesis

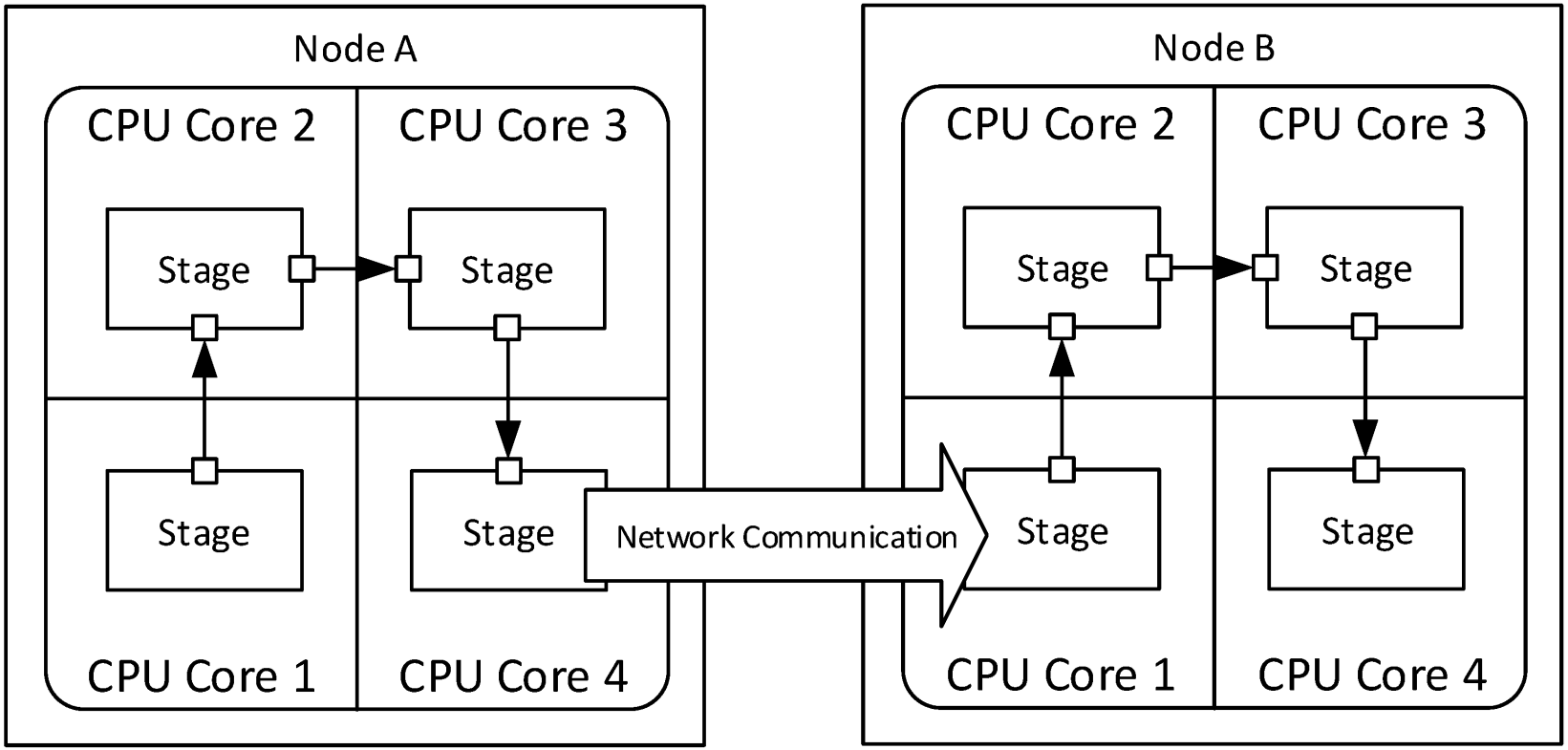
Florian Echternkamp – 21.04.2017

Motivation: Performance



Single node: Parallelization limited by CPU cores

Motivation: Performance



Multiple nodes: parallelization no more limited by CPU

Motivation: Locality

- Data locality
  - Big Data
  - Stages process data close to the corresponding data source

Outline

- Motivation
- Goals
- Foundations
- Developed Approach
- Evaluation
- Related Work
- Conclusion
- Future Work

- Motivation
- Goals
- Foundations
- Developed Approach
- Evaluation
- Related Work
- Conclusion
- Future Work

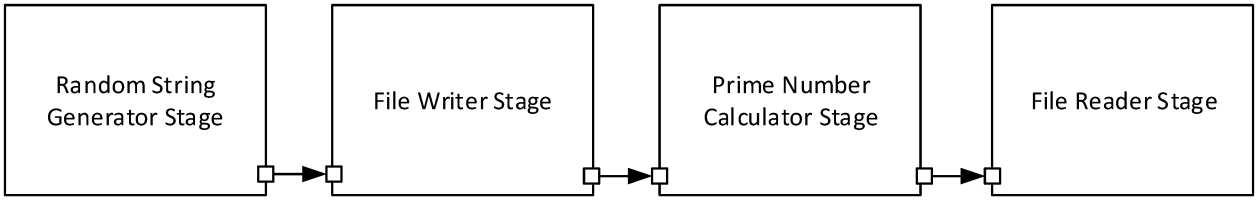
Goals

- G1: Evaluation of Java Frameworks for Distributed Application Development
  - G1.1: Providing Efficient Distributed Communication
  - G1.2: Providing Fault Tolerance
  - G1.3: Providing Remote Deployment and Execution
  - G1.3: Providing Encrypted Data Transmission
- G2: Implementation of a Distributed Pipe-and-Filter Architecture
- G3: Adding Support for Distributed Configurations in the TeeTime DSL
- G4: Evaluation of Our Approach
  - G4.1: Feasibility
  - G4.2: Performance

- Motivation
- Goals
- Foundations
- Developed Approach
- Evaluation
- Related Work
- Conclusion
- Future Work

Foundations & Technologies

- The Pipe-and-Filter Architectural Style [Sommerville 2012]



- The Pipe & Filter Framework TeeTime [Wulf et al. 2014]

TeeTime

- The Actor Model [Agha 1985]
- The Actor Framework Akka [http://akka.io]

akka

Foundations & Technologies

- TeeTime Domain-specific Language (DSL) [Zloch 2016]

- Communication Patterns for Distributed Systems [Silcock and Goscinski 1995]
  - Message Passing
  - Remote Procedure Call
  - Distributed Shared Memory

- Motivation
- Goals
- Foundations
- Developed Approach
- Evaluation
- Related Work
- Conclusion
- Future Work

Evaluation of Java Frameworks

- Build Infrastructure

License	Open Source	Min. JDK Version	Latest activity	Latest release
Apache 2.0	Yes	1.8	12.06.2016	03.03.2016 (2.3.0)

- Features

Communication Pattern	Transport Protocol	Fault Tolerance	Remote Deployment	Custom Serializer	Encryption
Message Passing	TCP, UDP	Supervisor, …	Yes	No	Yes

Evaluation of Java Frameworks

Apache Zookeeper

Apache River

JGroups

Copycat

Apache Storm

Hystrix

Orbit

akka

Atomix

Axon Framework
CQRS Framework for Java

Apache hadoop

Quasar

Apache Spark

JPPF

Evaluation of Java Frameworks

APACHE STORM™

Copycat

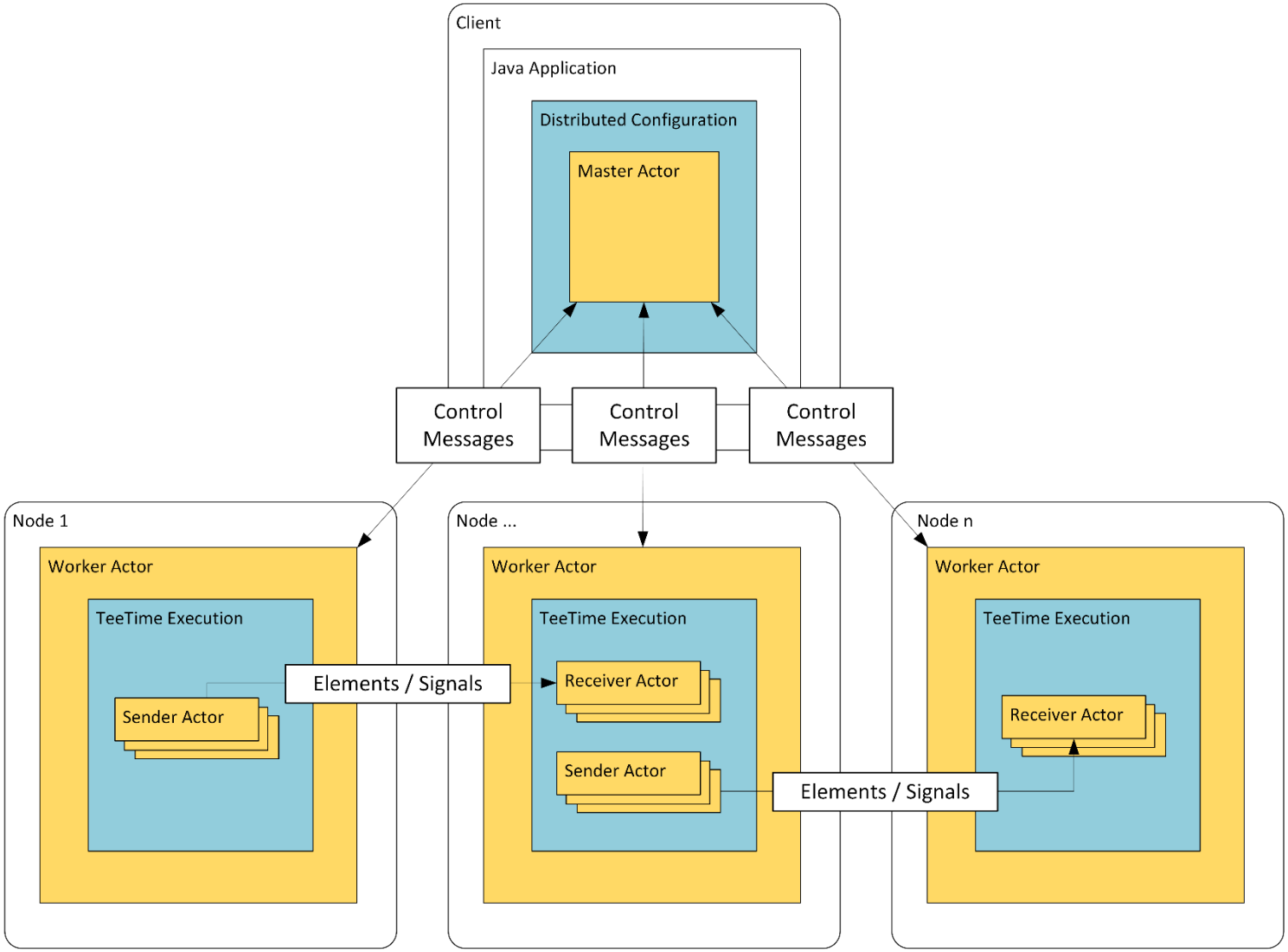
Hystrix

Atomix

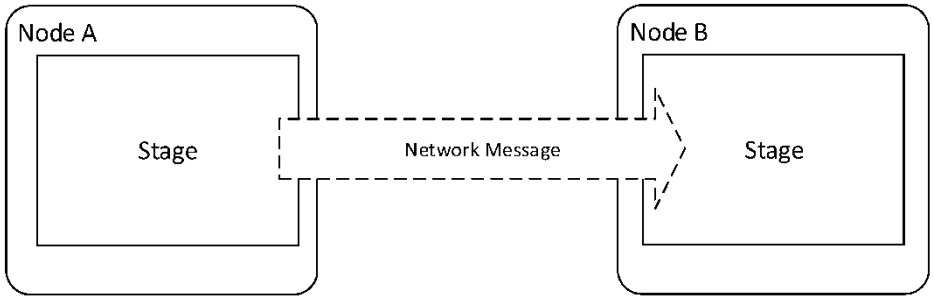
AxonFramework
CQRS FRAMEWORK FOR JAVA

Quasar

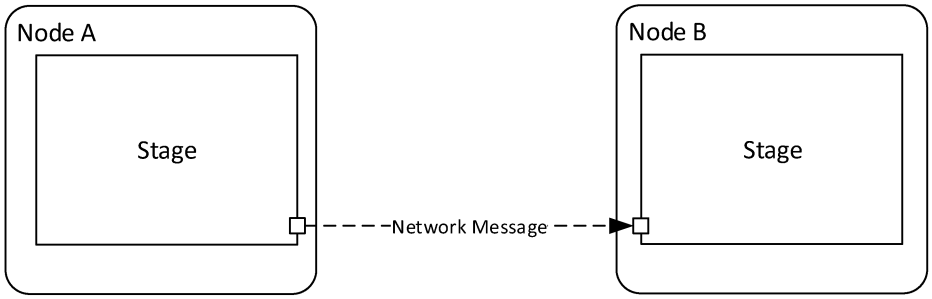
Architecture



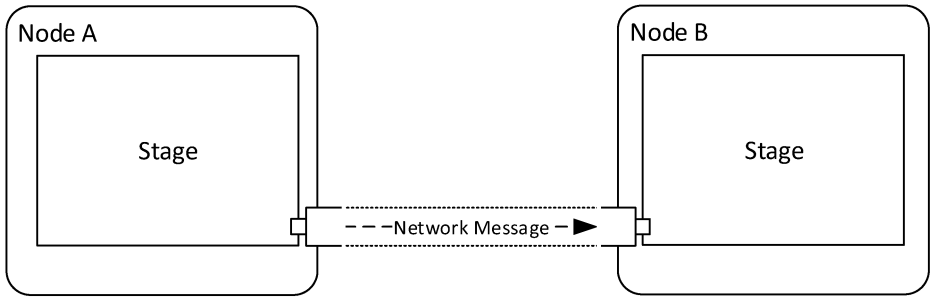
Implementation of Distr. Comm.



Communication via stages

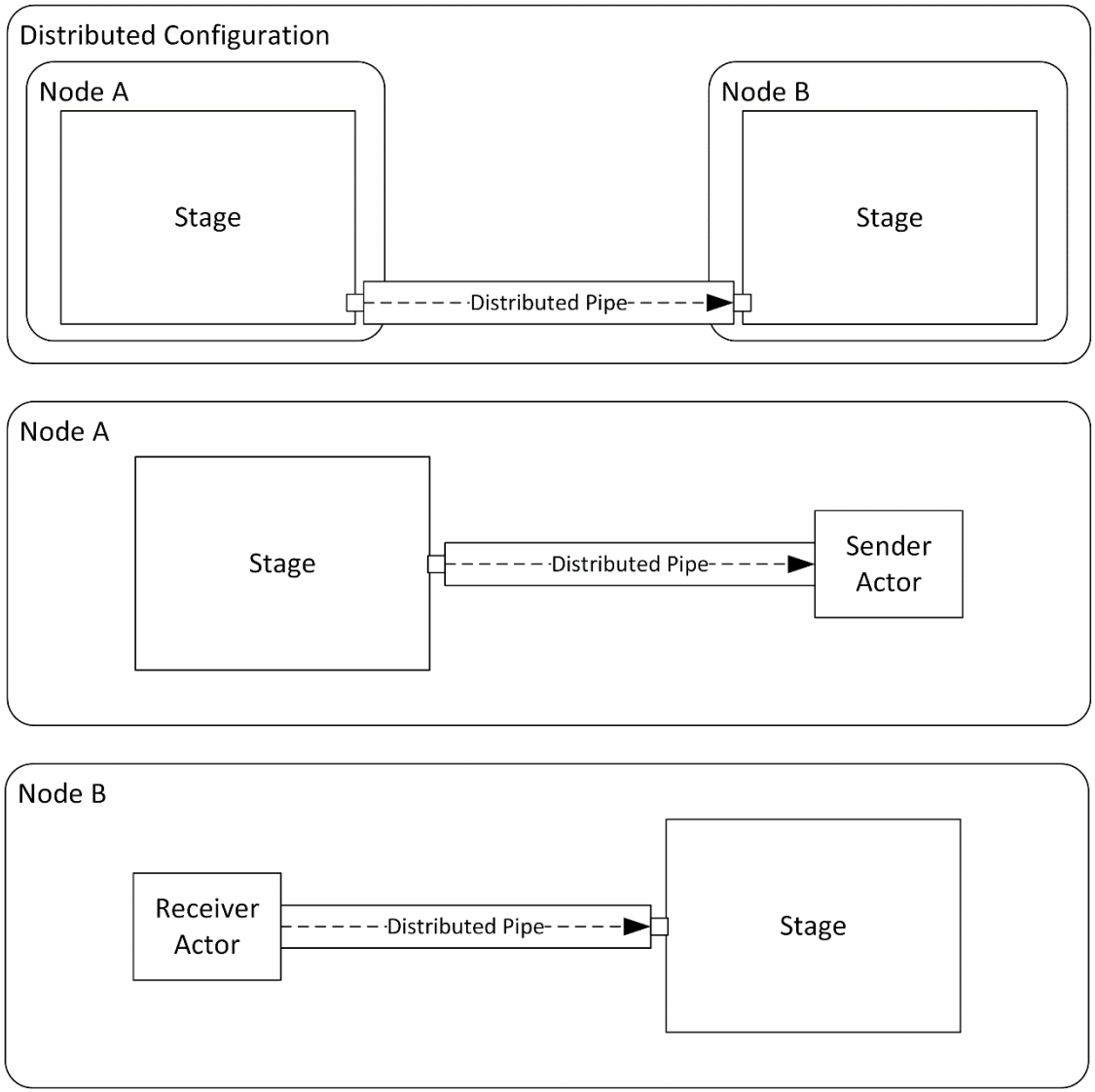


Communication via ports without a pipe



Communication via a pipe

Distributed Pipe



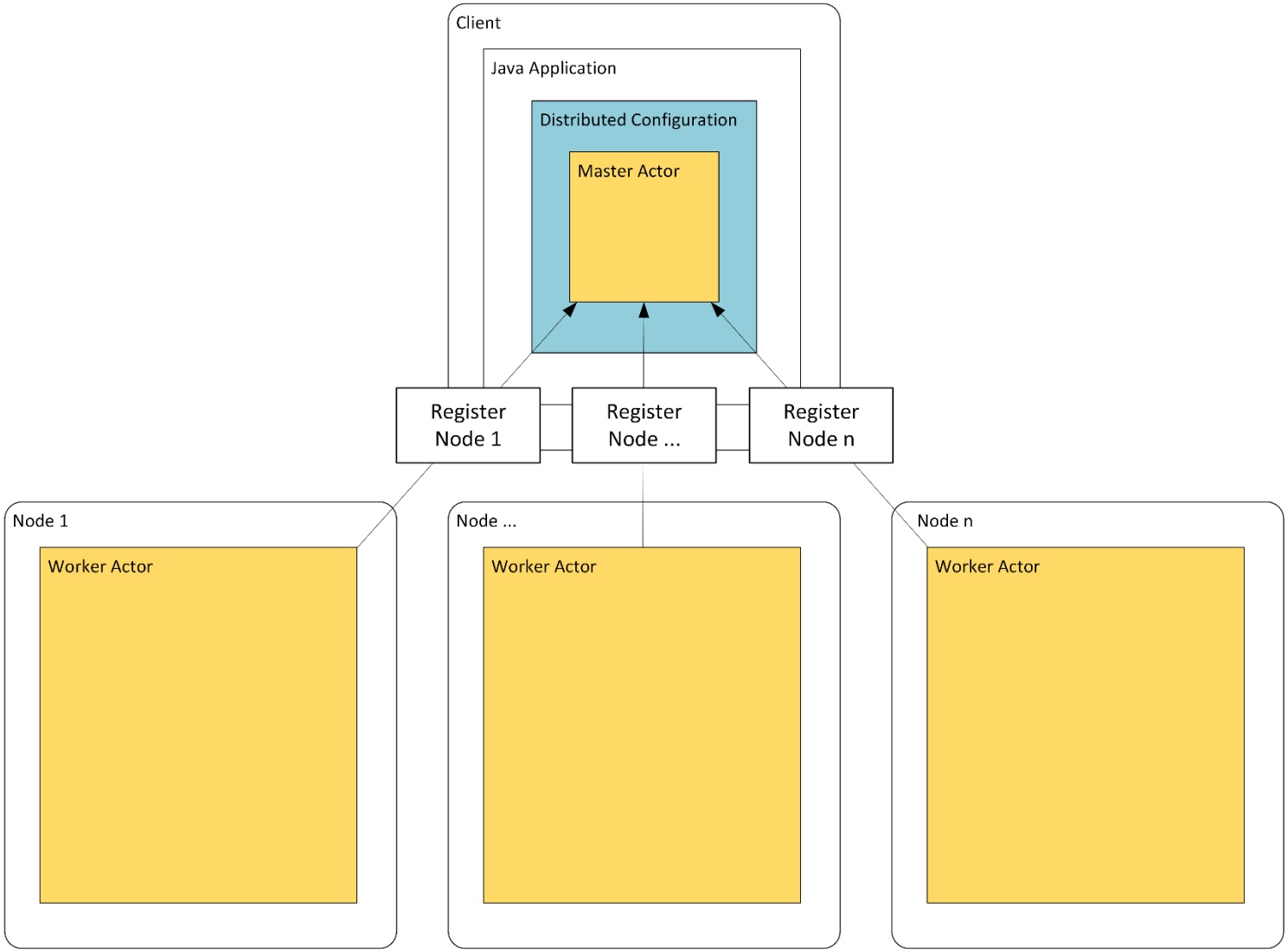
Impl. of Single Configuration


```
1   public class DistributedConfiguration extends AbstractDistributedConfiguration {
2       public DistributedConfiguration(){
3           RandomStringGeneratorStage stage1 = new RandomStringGeneratorStage(...);
4           CPULoadGeneratorStage stage2 = new CPULoadGeneratorStage(...);
5           CPULoadGeneratorStage stage3 = new CPULoadGeneratorStage(...);
6           CPULoadGeneratorStage stage4 = new CPULoadGeneratorStage(...);
7           EndStage stage5 = new EndStage();
8
9           createNodeForStages("Node1", stage1, stage2);
10          createNodeForStages(stage3);
11          createNodeForStages(stage4, stage5);
12
13          connectPorts(stage1.getOutputPort(), stage2.getInputPort());
14          connectPorts(stage2.getOutputPort(), stage3.getInputPort(), TransportProtocol.TCP);
15          connectPorts(stage3.getOutputPort(), stage4.getInputPort());
16          connectPorts(stage4.getOutputPort(), stage5.getInputPort());
17      }
18  }
```

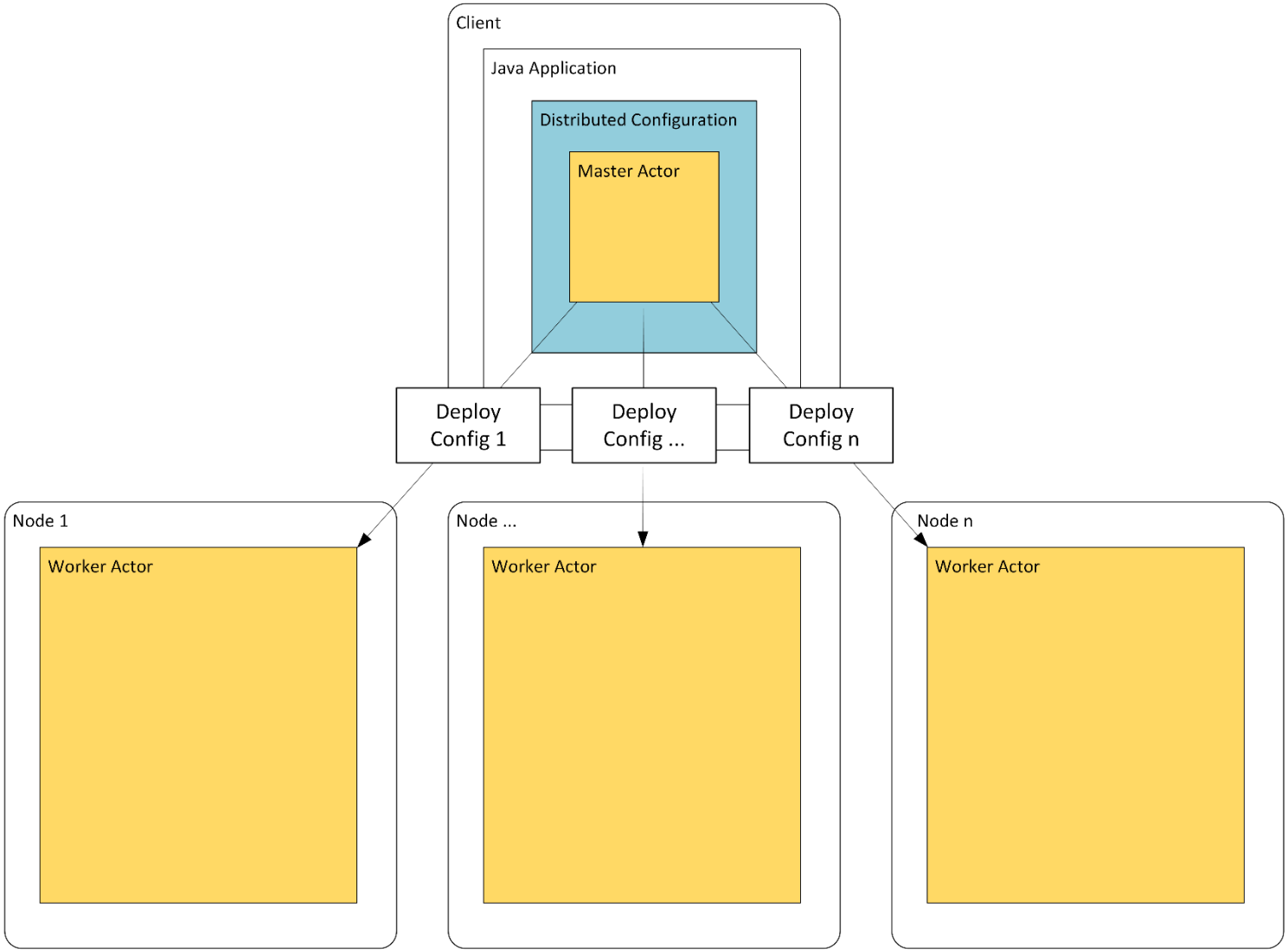
DSL for the Distributed Config


```
1   DistributedConfig(){
2       RandomStringGeneratorStage stage1(...)
3       CPULoadGeneratorStage stage2(...)
4       CPULoadGeneratorStage stage3(...)
5       CPULoadGeneratorStage stage4(...)
6       EndStage stage5()
7
8       Node(Node1) stage1 stage2
9       Node stage3
10      Node stage4 stage5
11
12      stage1->stage2-TCP->stage3->stage4->stage5
13  }
```

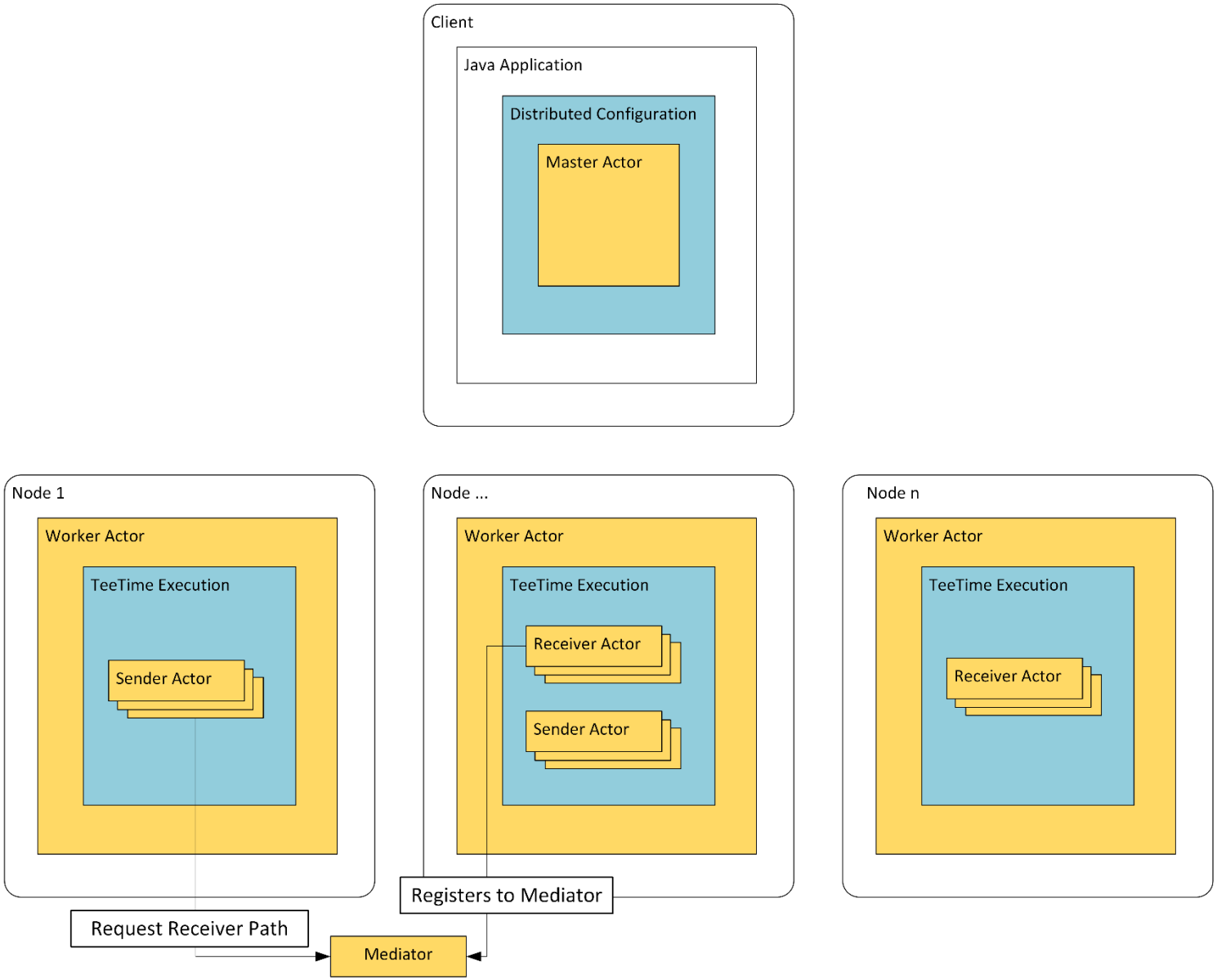
Remote Deployment (1 of 6)



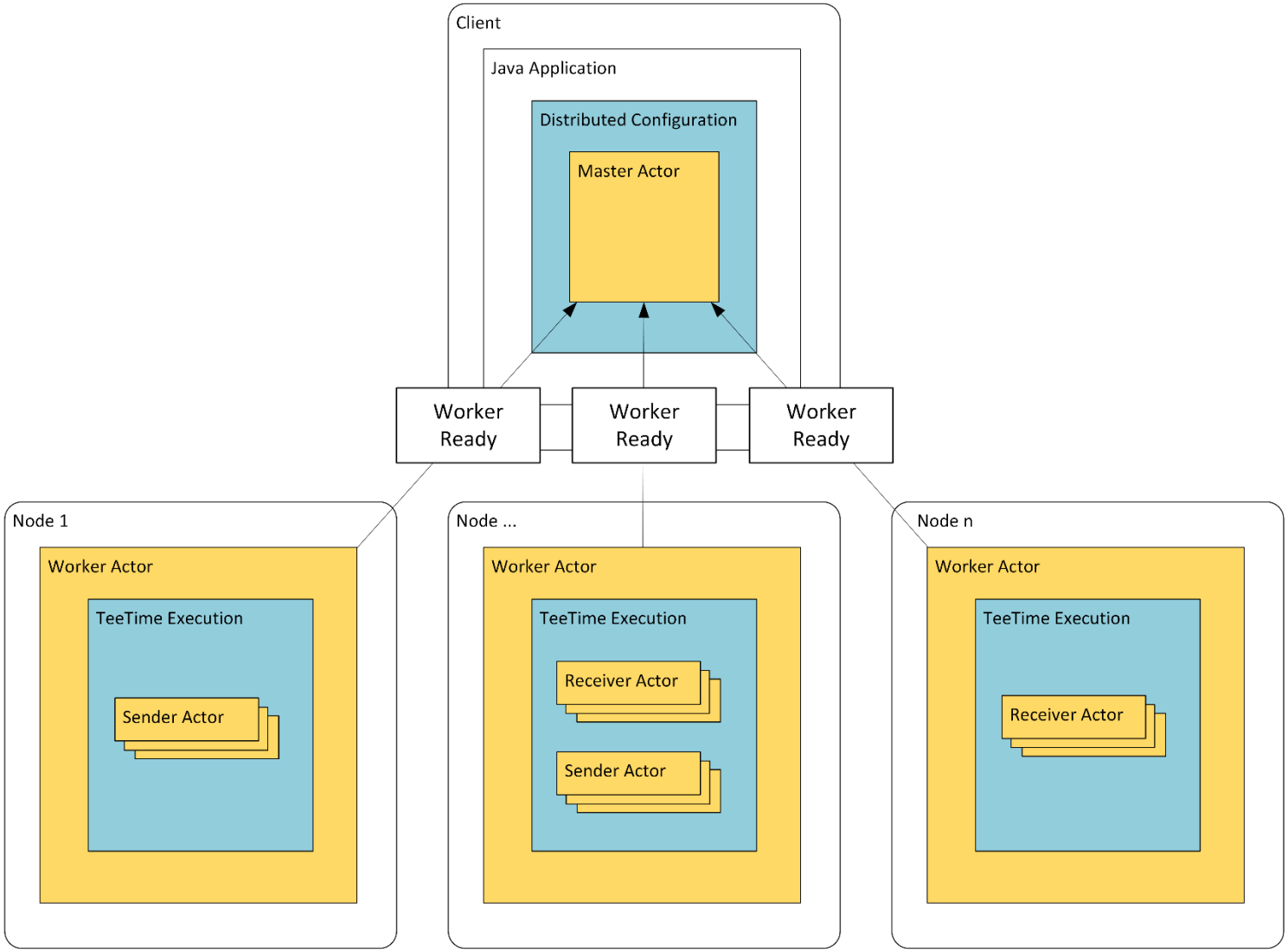
Remote Deployment (2 of 6)



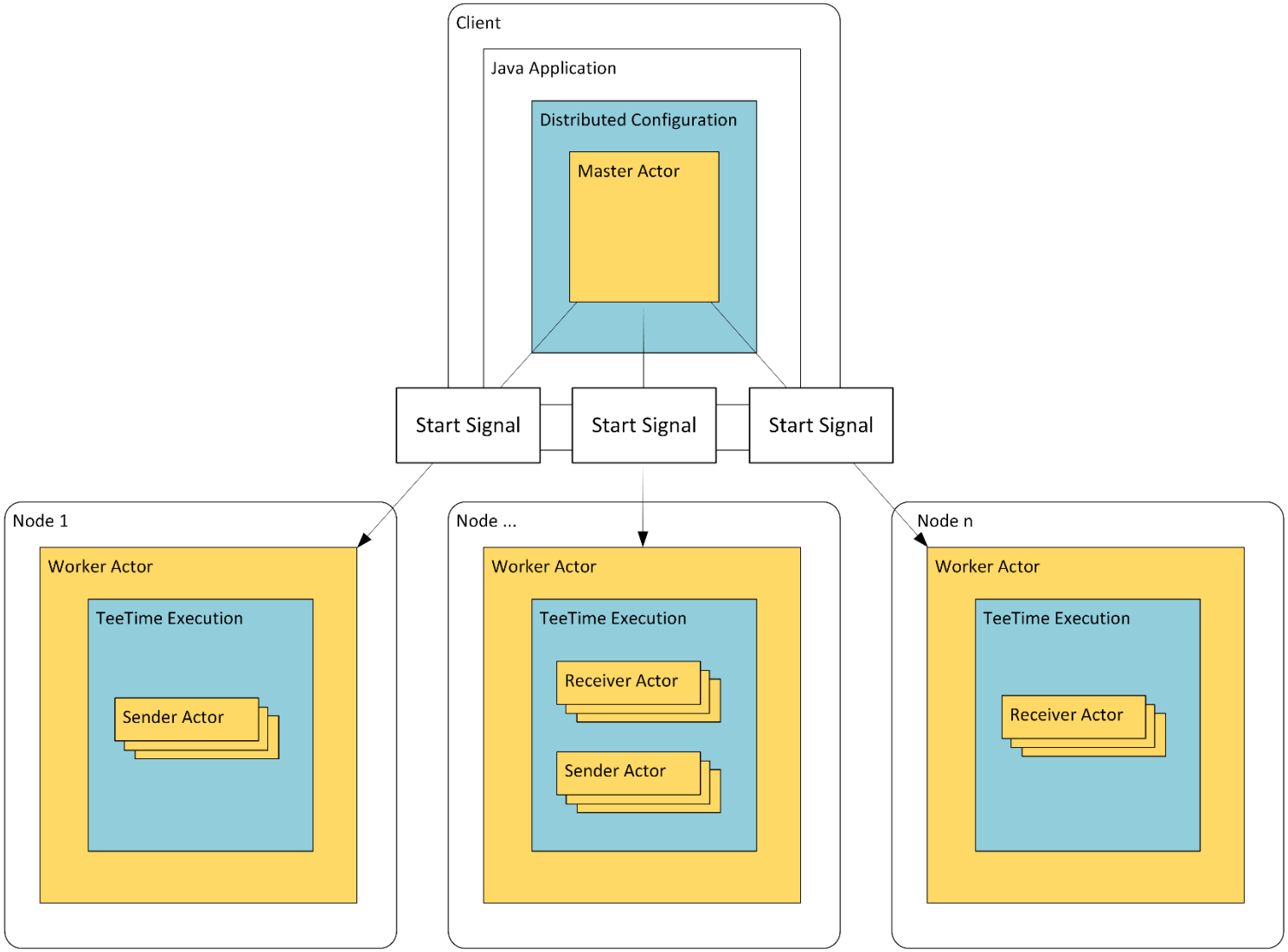
Remote Deployment (3 of 6)



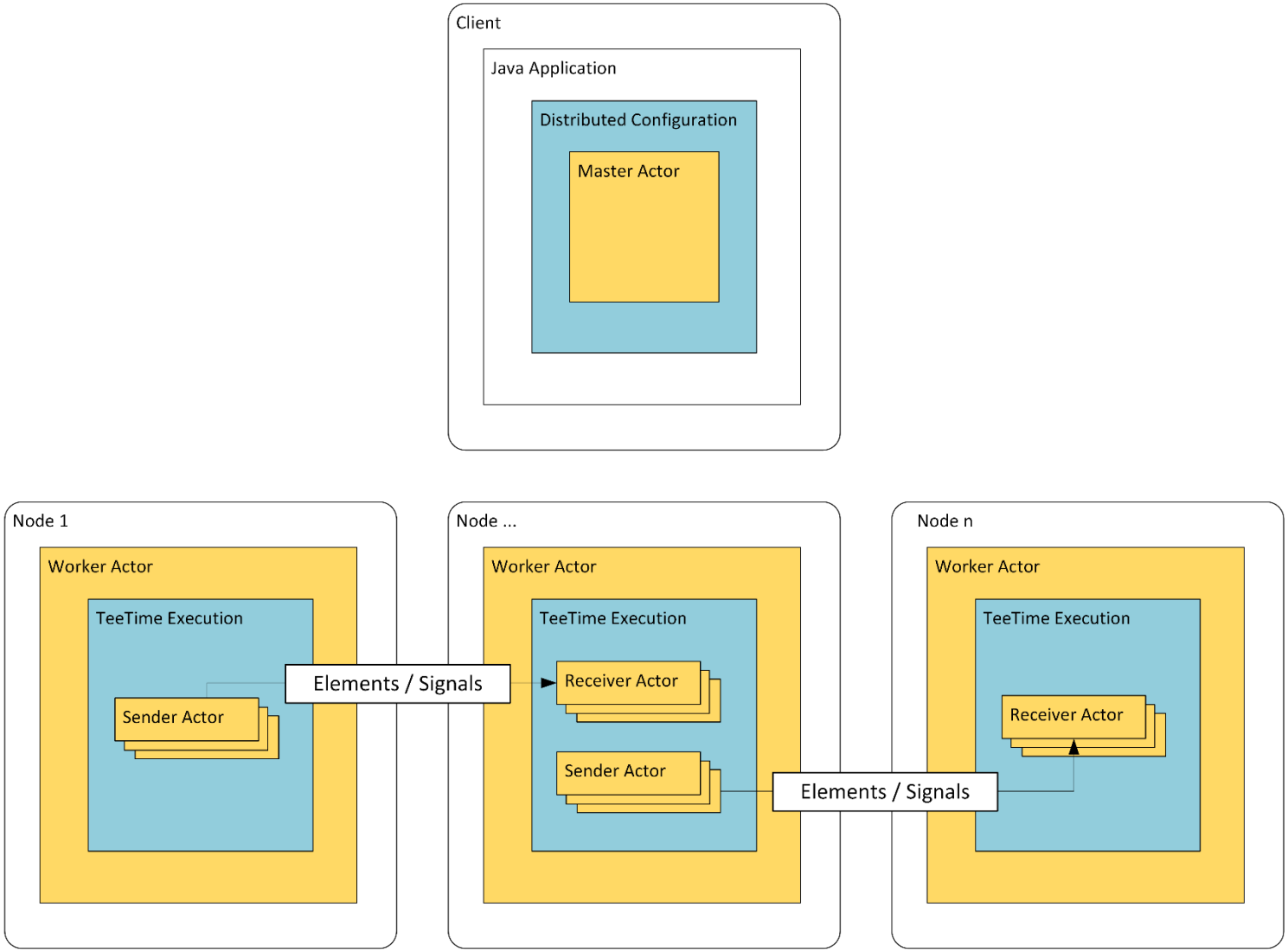
Remote Deployment (4 of 6)



Remote Deployment (5 of 6)



Remote Deployment (6 of 6)

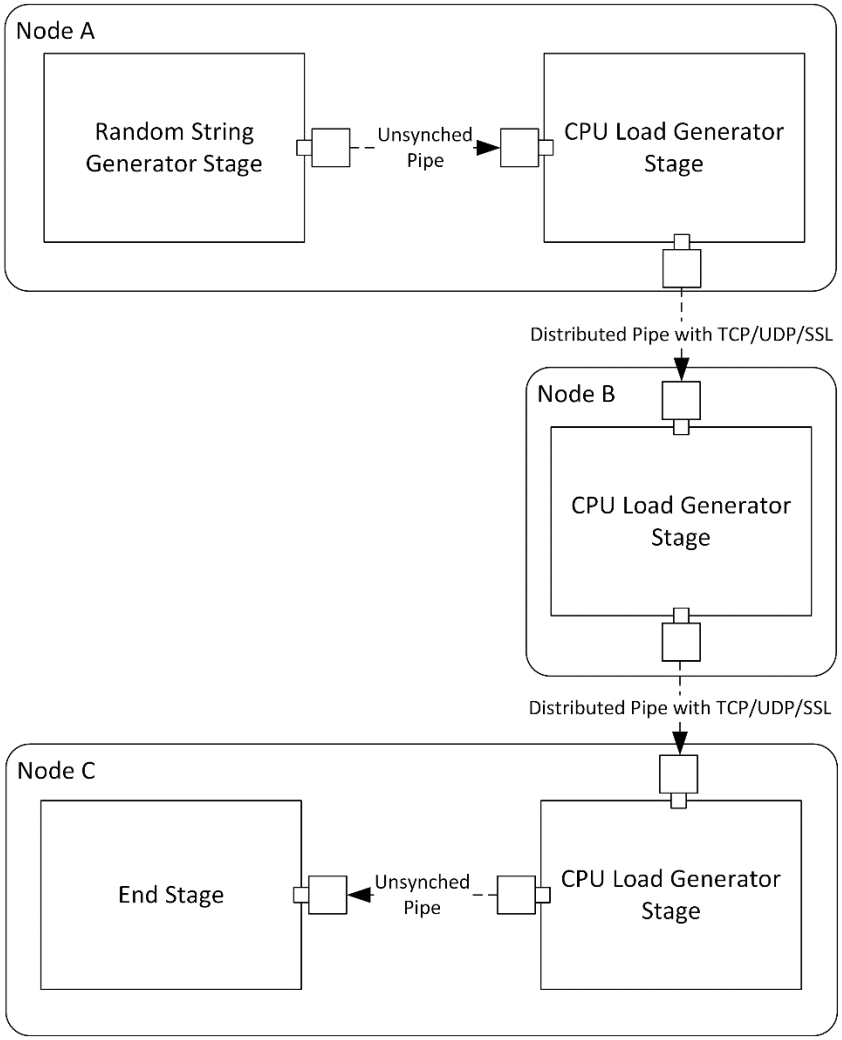


- Motivation
- Goals
- Foundations
- Developed Approach
- Evaluation
- Related Work
- Conclusion
- Future Work

Evaluation of Our Approach

- Feasibility
  - Remote deployment and remote execution
  - Distributed communication
  - Fault tolerance
- Performance
  - Communication overhead
  - Execution time

Feasibility Test Scenario

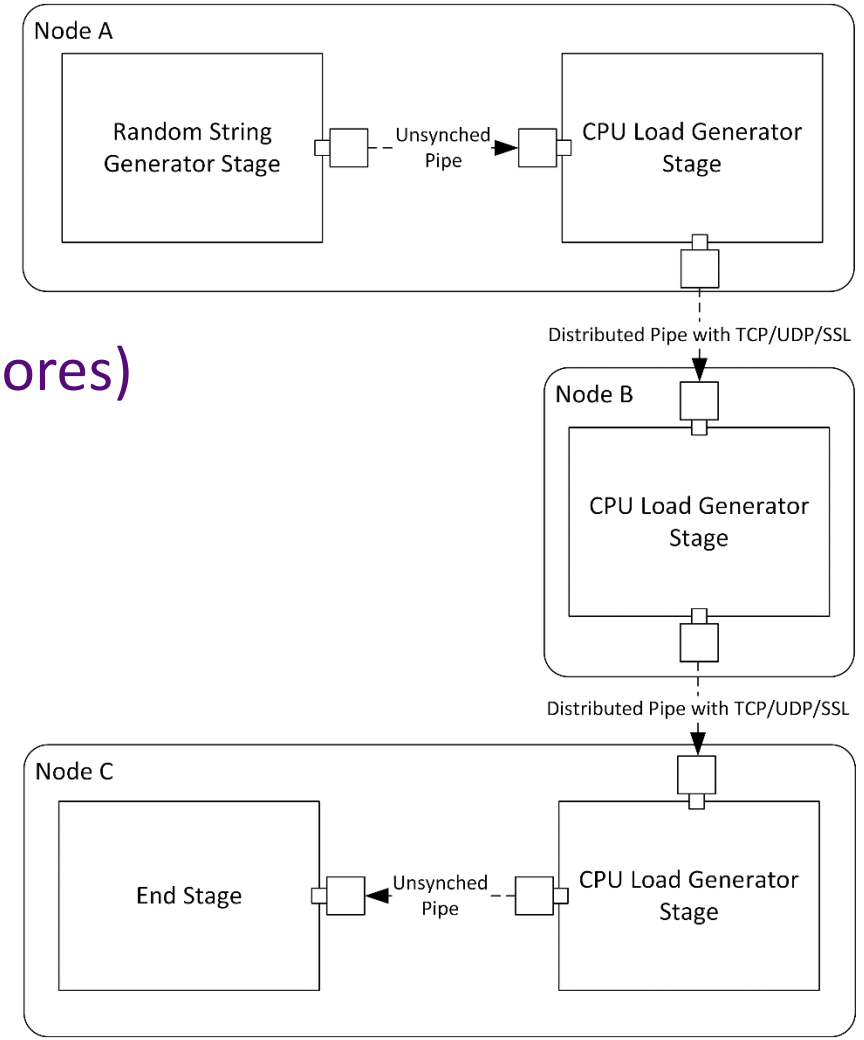


Resulting P&F architecture

Demo time

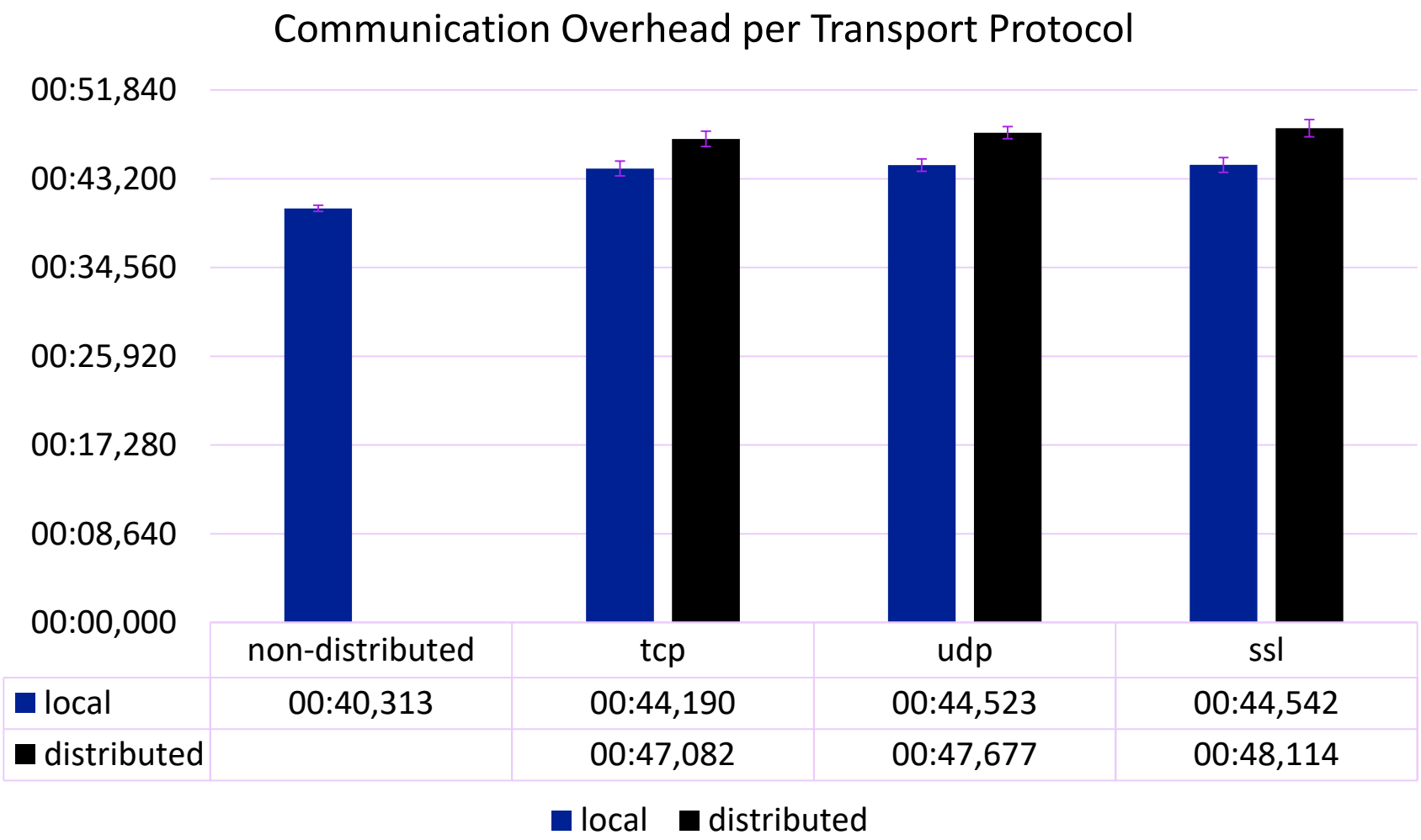
Comm. Overhead Test Scenario

- Test setup
  - 3 Cloud Nodes
    - 2x Intel Xeon (2.8 GHz, 8 cores)
    - 128 GB RAM
  - Software
    - Ubuntu 14.04.5 LTS
    - Java JDK 1.8.121 64bit
  - 10GBit/s network



Resulting P&F architecture

Comm. Overhead Results



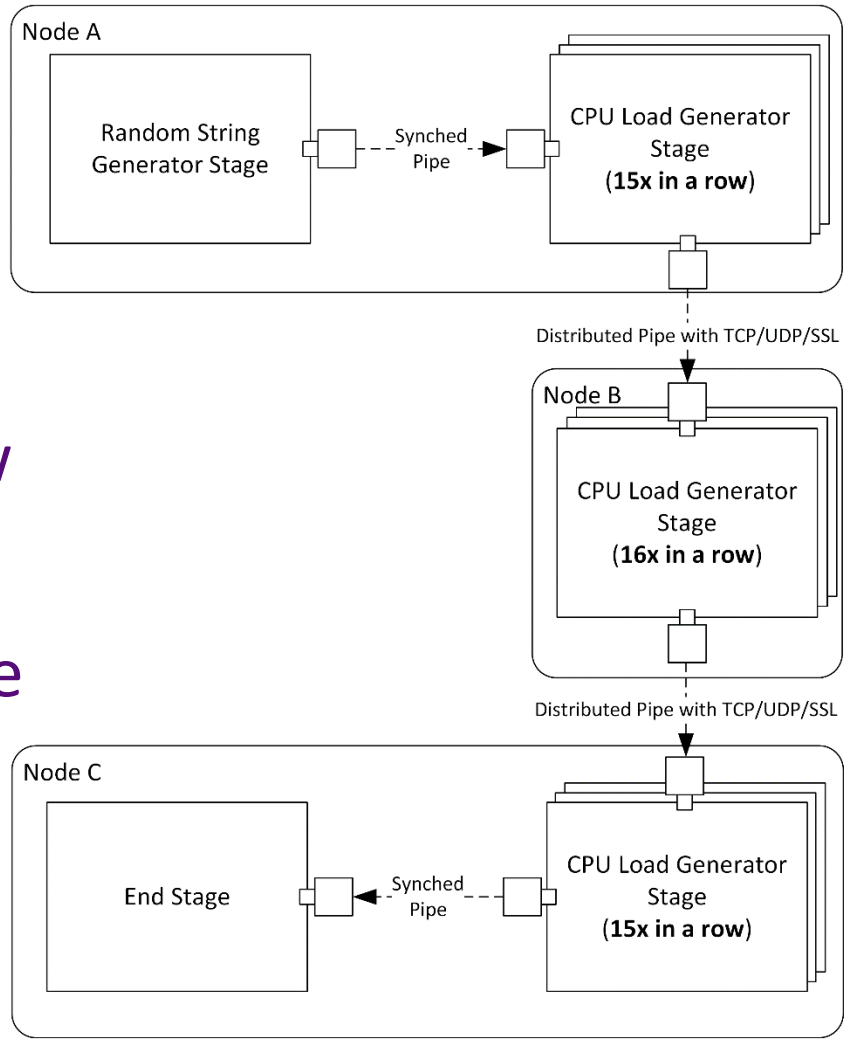
	non-distributed	tcp	udp	ssl
local	00:40,313	00:44,190	00:44,523	00:44,542
distributed		00:47,082	00:47,677	00:48,114

Test objects: 5000x 1 Megabyte large Strings | 20 test iterations

CI 95% <= ±00:00,840

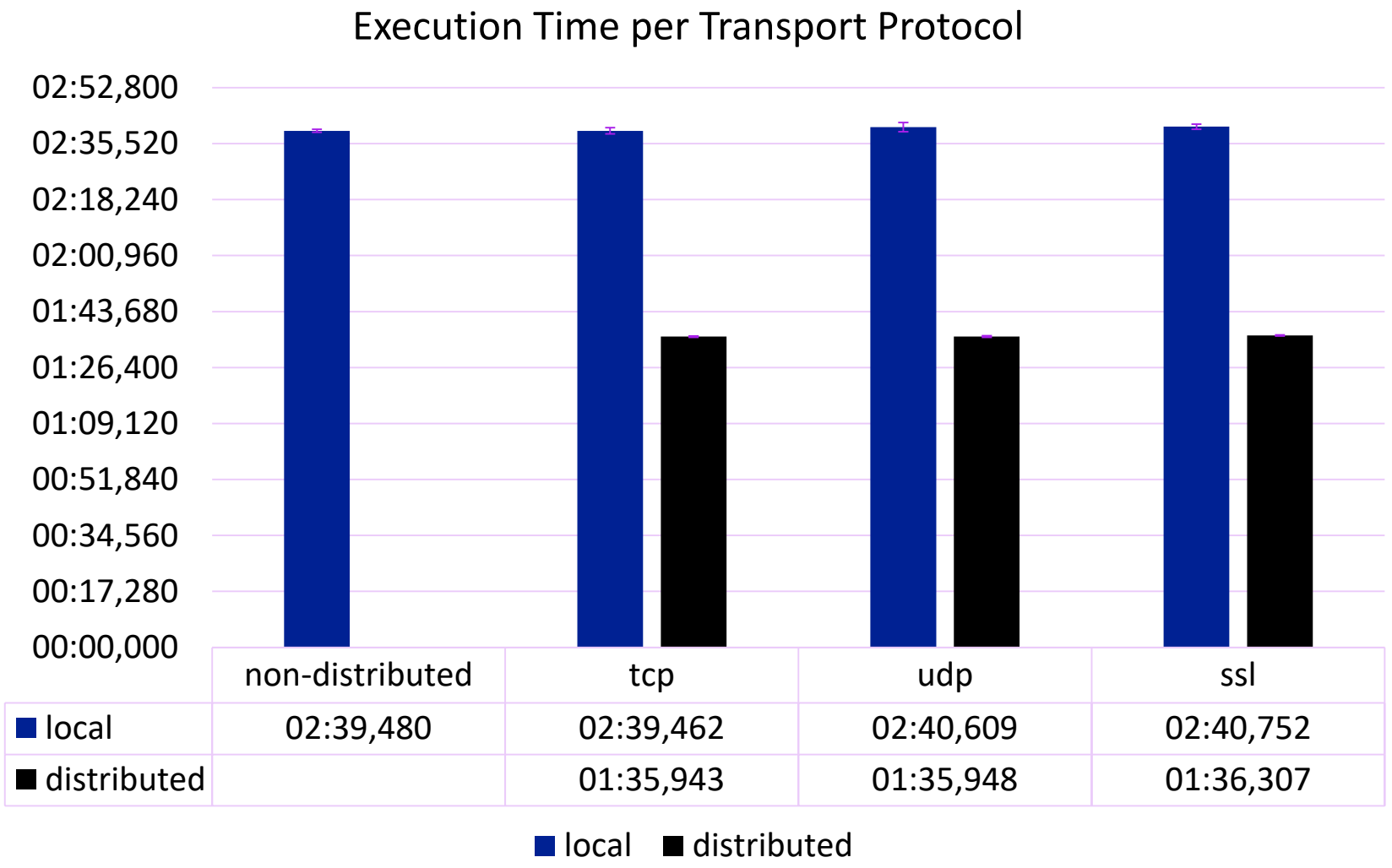
Execution Time Test Scenario

- Configuration Changes
  - Multiple CPU Load Generator Stages in a row
  - Synched pipes
  - Increased CPU load by the factor 10



Resulting P&F architecture

Execution Time Results



	non-distributed	tcp	udp	ssl
local	02:39,480	02:39,462	02:40,609	02:40,752
distributed		01:35,943	01:35,948	01:36,307

Test objects: 1000x 1 Megabyte large Strings | 20 test iterations

Local CI 95% <= ±00:01,386 | Distributed CI 95% <= ±00:00,250

- Motivation
- Goals
- Foundations
- Developed Approach
- Evaluation
- Related Work
- Conclusion
- Future Work

Related Work

- Apache Hadoop[1]: MapReduce on a big problem

- Apache Spark[2] and Storm[3]: only acyclic graphs

- Akka[4]: message box is untyped

[1]http://hadoop.apache.org | [2]http://spark.apache.org | [3]http://storm.apache.org | [4]http://akka.io

- Motivation
- Goals
- Foundations
- Developed Approach
- Evaluation
- Related Work
- Conclusion
- Future Work

Conclusion

- Achieved all implementation goals
  - Distributed communication
    - TCP and UDP
    - Encrypted
  - Remote deployment and remote execution
  - Fault tolerance
  - Distributed configurations in the TeeTime DSL
- Feasibility and performance advantages shown

- Motivation
- Goals
- Foundations
- Developed Approach
- Evaluation
- Related Work
- Conclusion
- Future Work

Future Work

- Access stage attributes via the client
- SSL connection between master and worker
- Providing a master system similar to the remote system
  - No need to embed the execution of the distributed configuration
  - Similar to the non-distributed config

References

- [Agha 1985] G. A. Agha. Actors: a model of concurrent computation in distributed systems.
- Technical report. DTIC Document, 1985.
- [Silcock and Goscinski 1995] J. Silcock and A. Goscinski. Message passing, remote procedure calls and distributed shared memory as communication paradigms for distributed systems. Deakin University, School of Computing and Mathematics, 1995.
- [Sommerville 2012] I. Sommerville. Software engineering. In: Pearson Studium, Mar. 1, 2012. Chapter 6.3.4, pages 200–201.
- [Wulf et al. 2014] C. Wulf, N. C. Ehmke, and W. Hasselbring. Toward a generic and concurrency-aware pipes & filters framework (2014).
- [Zloch 2016] M. Zloch. Development of a domain-specific language for pipe-and-filter configuration builders. Feb. 2016.