AkkaProf

a Profiler for Akka Actors in Parallel and Distributed Applications

Andrea Rosà*, Lydia Y. Chenˆ, and Walter Binder*

*Università della Svizzera italiana (USI), Faculty of Informatics, Lugano, Switzerland
ˆIBM Research Lab Zurich, Rüschlikon, Switzerland

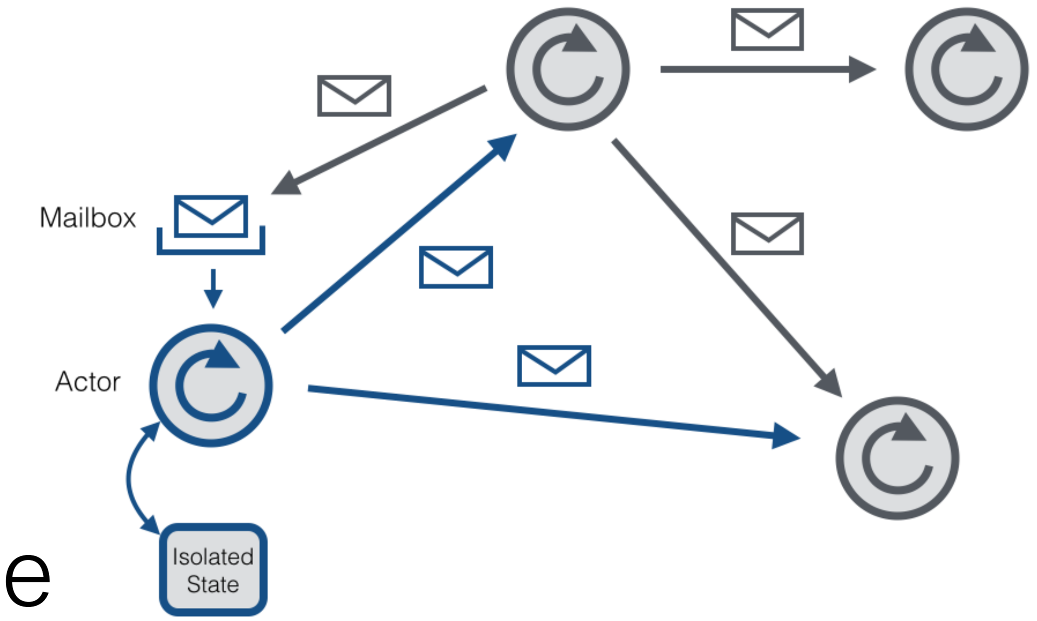
APLAS 2016
November 21st, 2016
Hanoi, Vietnam

AkkaProf

- A **profiler** for Akka actors
  - Based on **bytecode instrumentation**
    - Platform-independent profiling
  - Centered on:
    - actor **utilization**
    - **communication** between actors

Actors

- Atomic concurrent entities
  - Cannot share state
  - Can communicate only via asynchronous messages
  - Execute computations in response to a message
  - Message type dictates executed computation



Actors in practice

- Many implementations for Java, C++, Python, .NET, Haskell, …
- On the JVM: Akka is the most used one
- Existing general-purpose profilers cannot recognize actors
- Existing actor profilers do not measure computations

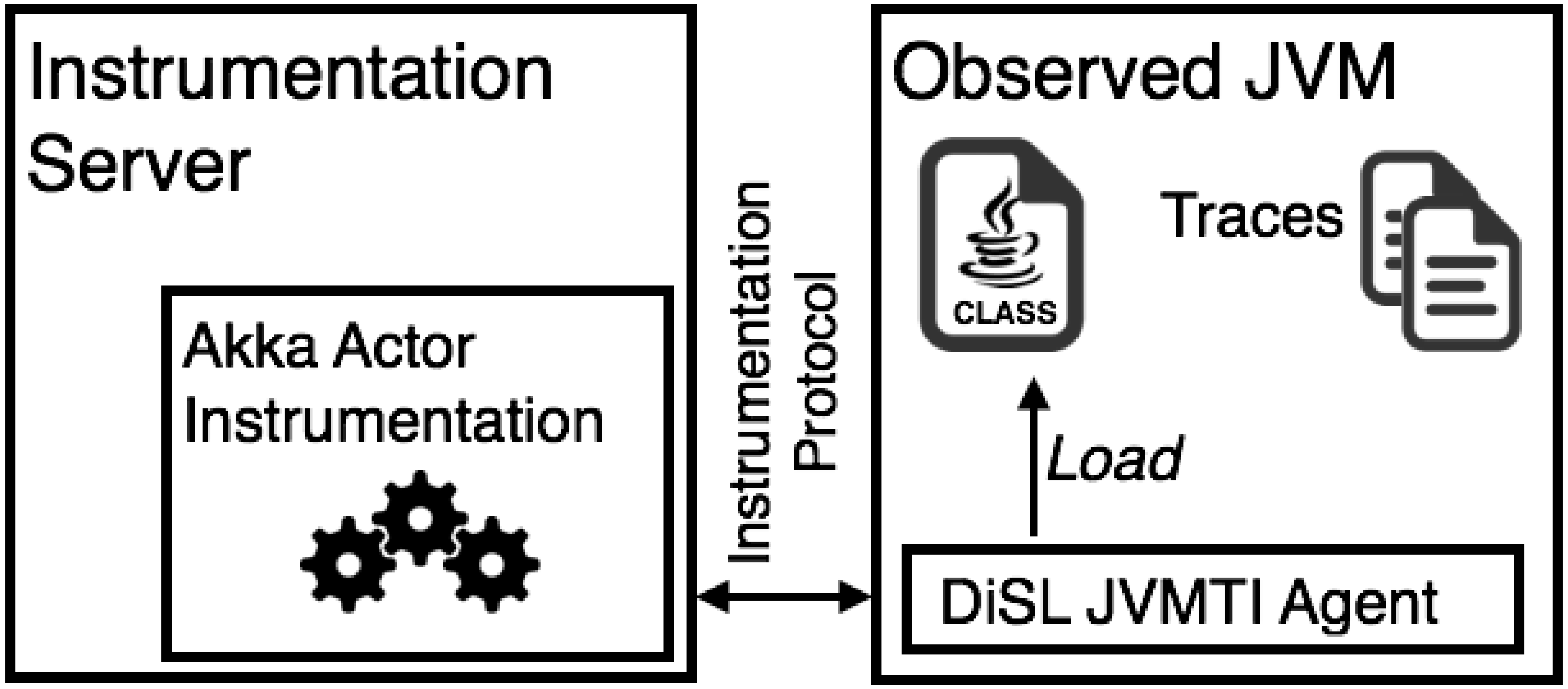
Metrics

- Utilization
  - Executed computations
  - Initialization cost

  } Bytecode count
- Communication
  - Messages sent
  - Messages received
- All metrics are platform-independent

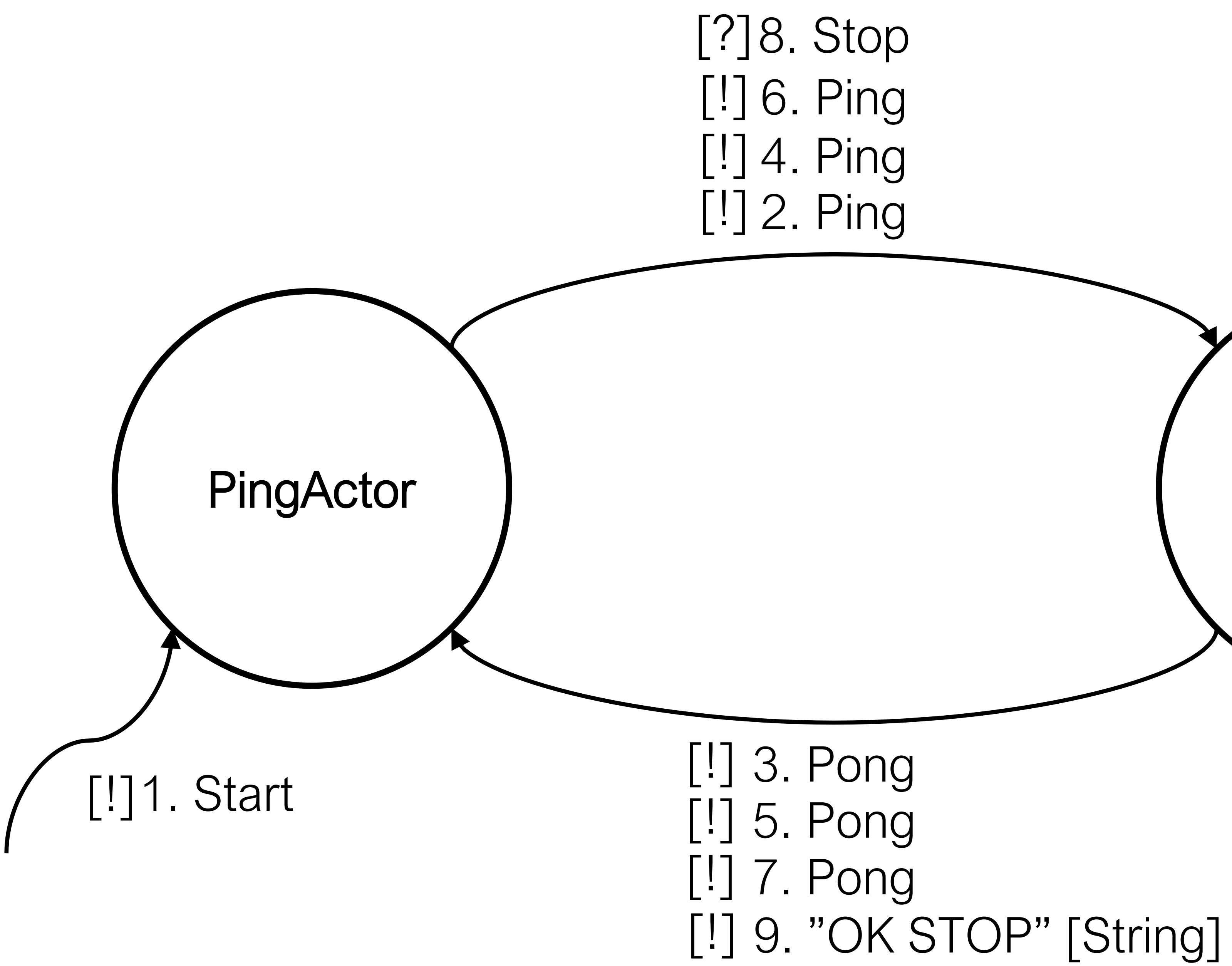
Architecture

- Relies on the DiSL bytecode instrumentation framework [1]
  - Guarantees full bytecode coverage



[1] L. Marek et al., *DiSL: A Domain-specific Language for Bytecode Instrumentation*. AOSD'12

Demo: pingpong



```
! tell
? ask
```

Evaluation

- Use cases:
  1. Savina benchmark suite
     - Goal: analyze actor utilization
  2. Signal/Collect framework
     - Goal: analyze load balancing
  3. Spark and Flink frameworks
     - Goal: analyze communication

Use case: Spark/Flink

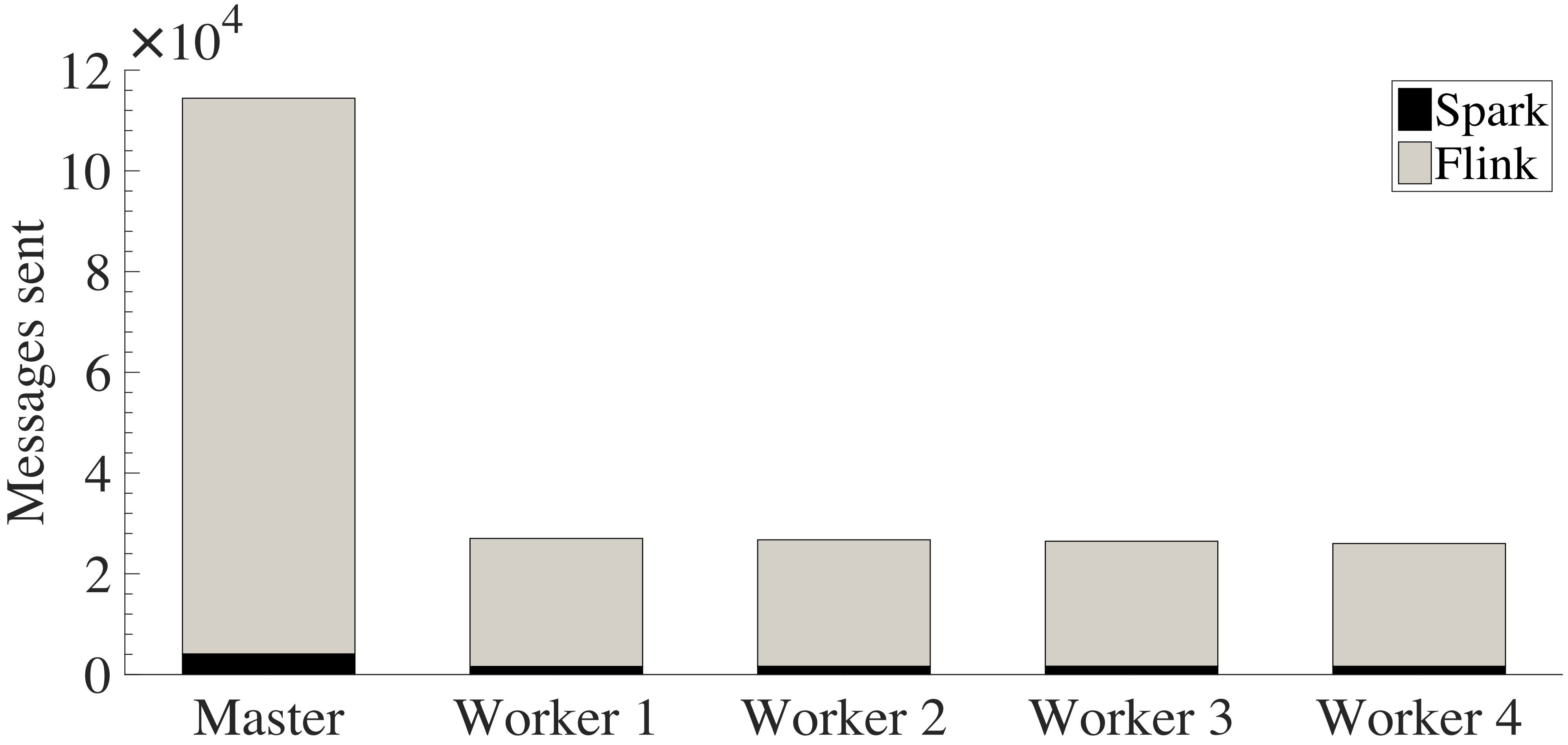
- Apache Spark [2] and Apache Flink [3]
    - Computing frameworks for big-data, machine learning, graphs, streaming, etc.
    - Master/slave architecture
    - Actors handle communication between master and workers
- Goal:
    - Compare communication between workers

[2] M. Zaharia et al., *Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing*. NSDI'12

[3] Apache Flink. https://flink.apache.org.

Use case: Spark/Flink

Kmeans on 10M points



- Great difference between Spark and Flink:
  - Worker: ~1.6k (Spark), ~25k (Flink)
  - Master: ~4.1 k (Spark), ~110k (Flink)
- Kmeans run faster in Spark (up to 7x faster)

Discussion

- Limitation of bytecode count:
  - Cannot track code without bytecode representation (e.g., native methods)
    - Cannot track VM activities (e.g., garbage collection)
  - Bytecodes of different complexity are represented with the same unit
  - Susceptible to dynamic optimizations

Discussion

- Future work:
  - Machine instruction count
    - Platform-specific
    - Perturbed by instrumentation (unlike bytecode count)
  - Network traffic
    - Platform-specific
  - Message flow between actors

Thank you for the attention

- AkkaProf demo: http://www.inf.usi.ch/phd/rosaa/ws/AkkaProf.html
- DiSL: https://disl.ow2.org
- Contact details:

  Andrea Rosà
  andrea.rosa@usi.ch
  http://www.inf.usi.ch/phd/rosaa