An Actor-based Software Framework for Scalable Applications

Federico Bergenti, Agostino Poggi, Michele Tomaiuolo

{federico.bergenti,agostino.poggi,michele.tomaiuolo}@unipr.it

Research Goals

- Develop a software framework that
  - Is suitable for implementing concurrent and distributed applications
  - Ensures efficient execution of applications
- Experiment the software framework in
  - Distributed information management
  - Agent-based modelling and simulation

Actor Model

- No shared state
- Asynchronous message passing
- Mailboxes to buffer incoming messages
- Dynamic behaviors
- A *very old* model, but multi-core processors made it useful again!

Actor Programming Models

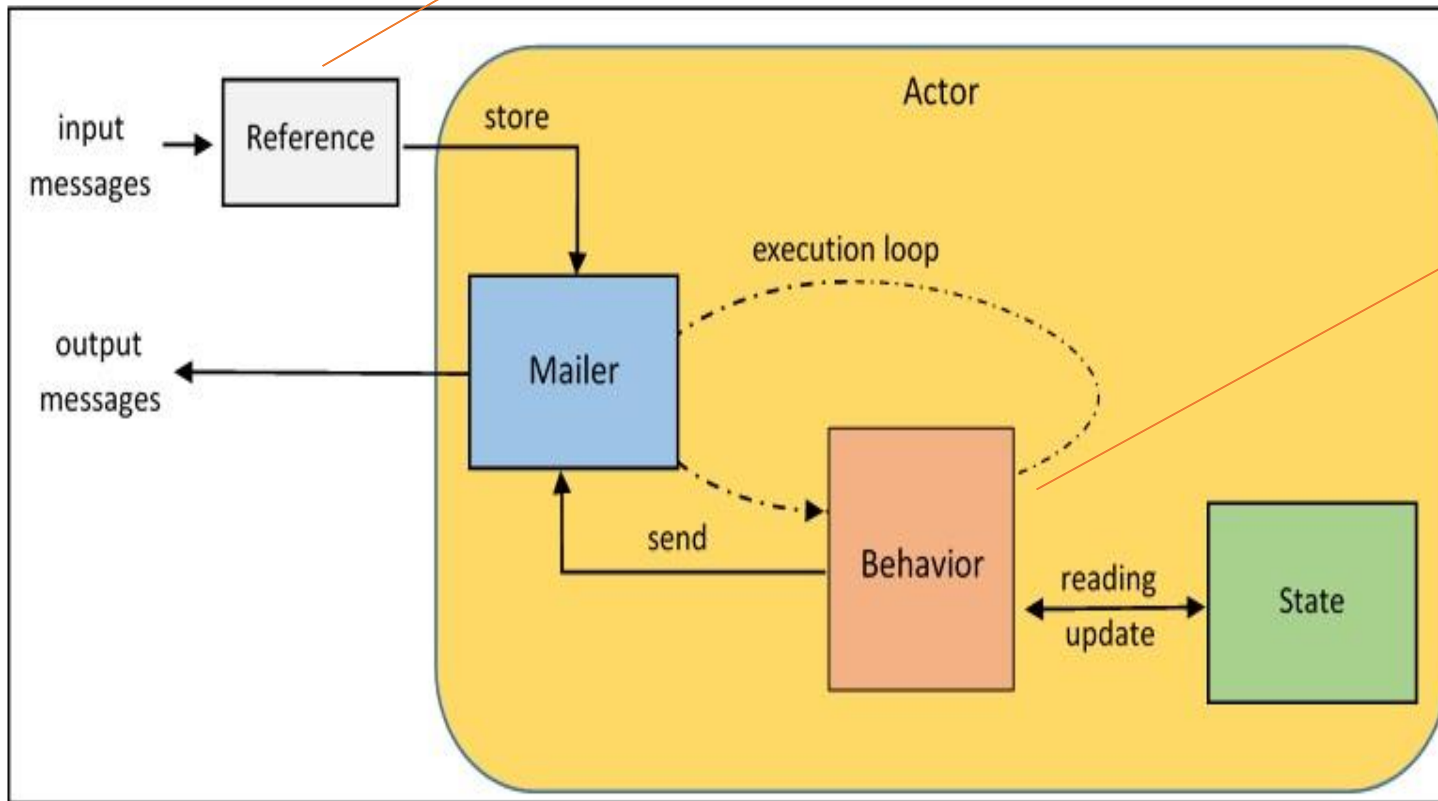
- Thread based
    - Performance overhead
  - Light-weight thread (e.g., Kilim)
      - Low level programming
- Event based
    - Complex code
  - Continuation (e.g., Salsa)
    - Fairness problems
- Mixed (e.g., Scala)
    - Use of specific primitives
    - Fairness problems

CoDE Features

- *CoDE (Concurrent Development Environment)* is a software framework based on the actor model
- Uses an event-based programming model
  - Actor behaviors define the event (message) handlers
  - Runtime components propagate messages and activate appropriate handlers
- Offers a flexible implementation model for applications
  - Applications are defined through extensible sets of *replaceable* runtime behaviors
- Makes the development of applications simple
  - Application developers must only write the code of actor behaviors and configure runtime components

Actor Architecture

- Queues messages in mailboxes
- Acts as the actor address

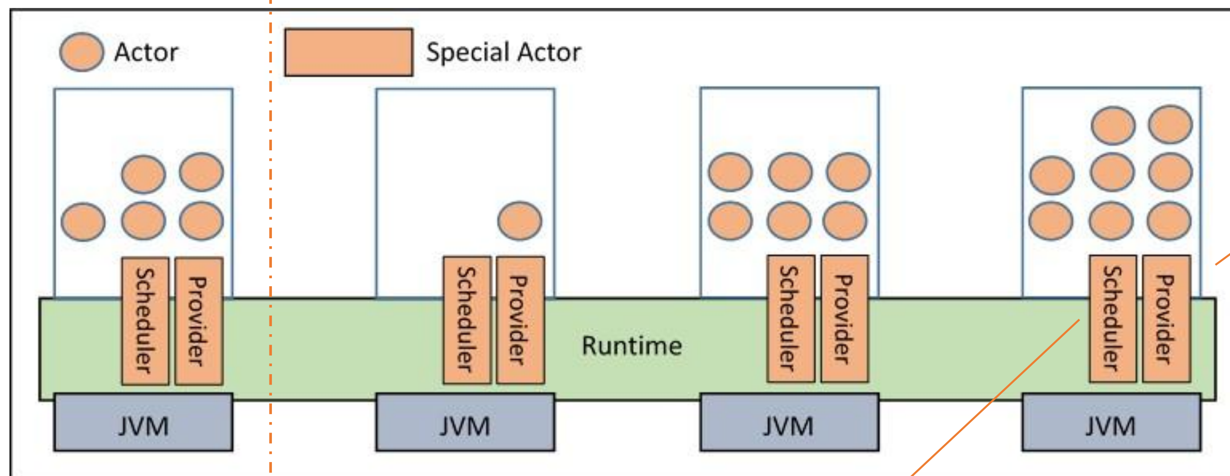


Sends messages
Creates new actors
Updates its local state
Changes its behavior
Kills itself

Application Architecture

- Acts as container of a set of actors
- Supports a transparent communication between local actors and remote actors
- Enhances their functionalities through a set of services

Actor Space



Supports actors with additional services

- Drives the execution of actors
- Usually defines the execution profile

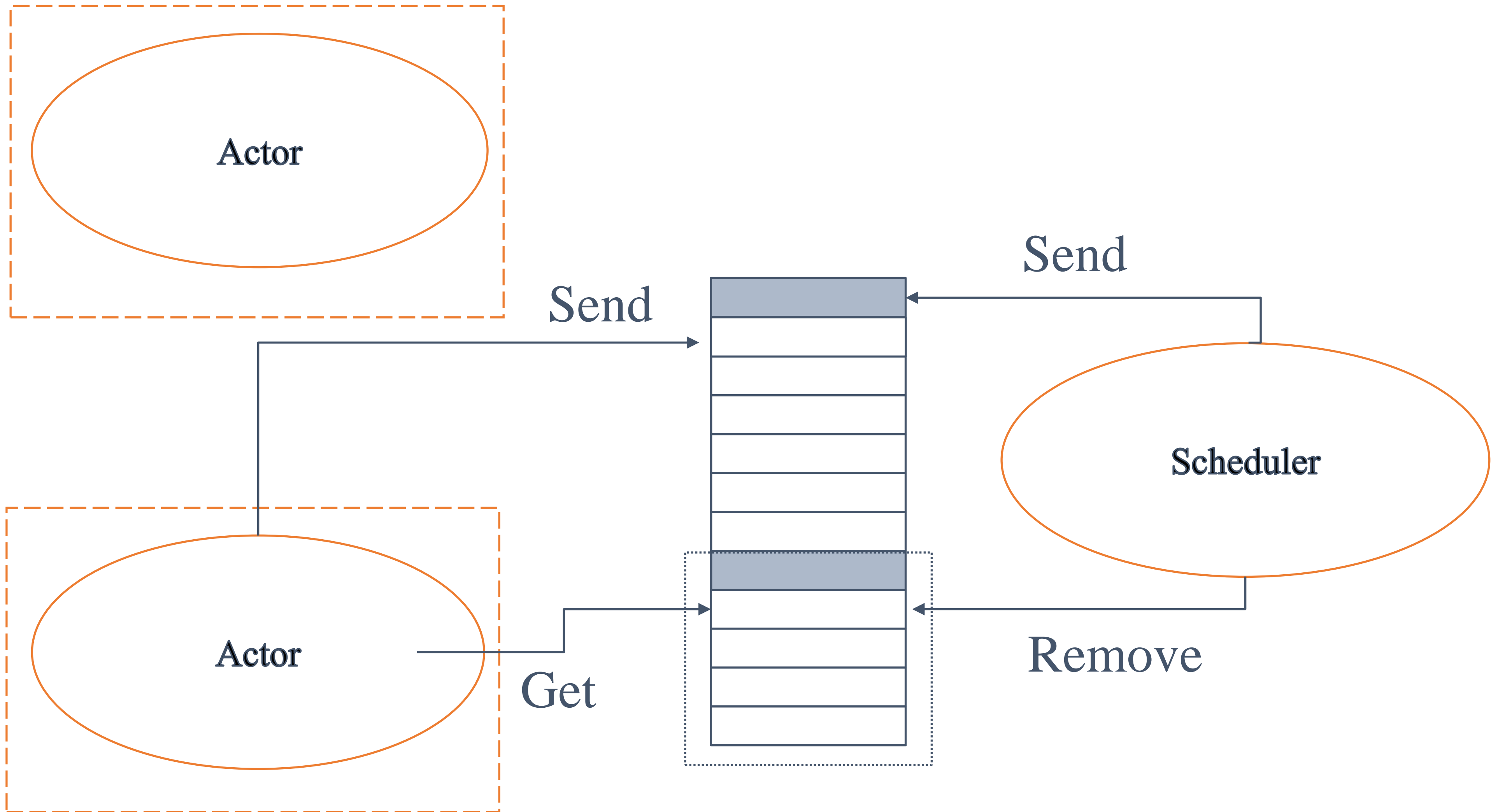
Actor Space Services

- A means to
  - Access services provided by legacy systems
  - Avoid the need of extending the set of primitive actions that characterize an actor
- Are provided by the *service provider actor*
- Current implementation includes
  - Broadcasting
  - Mobility
  - Naming
  - Remote creation
  - Topic-based publish-subscribe

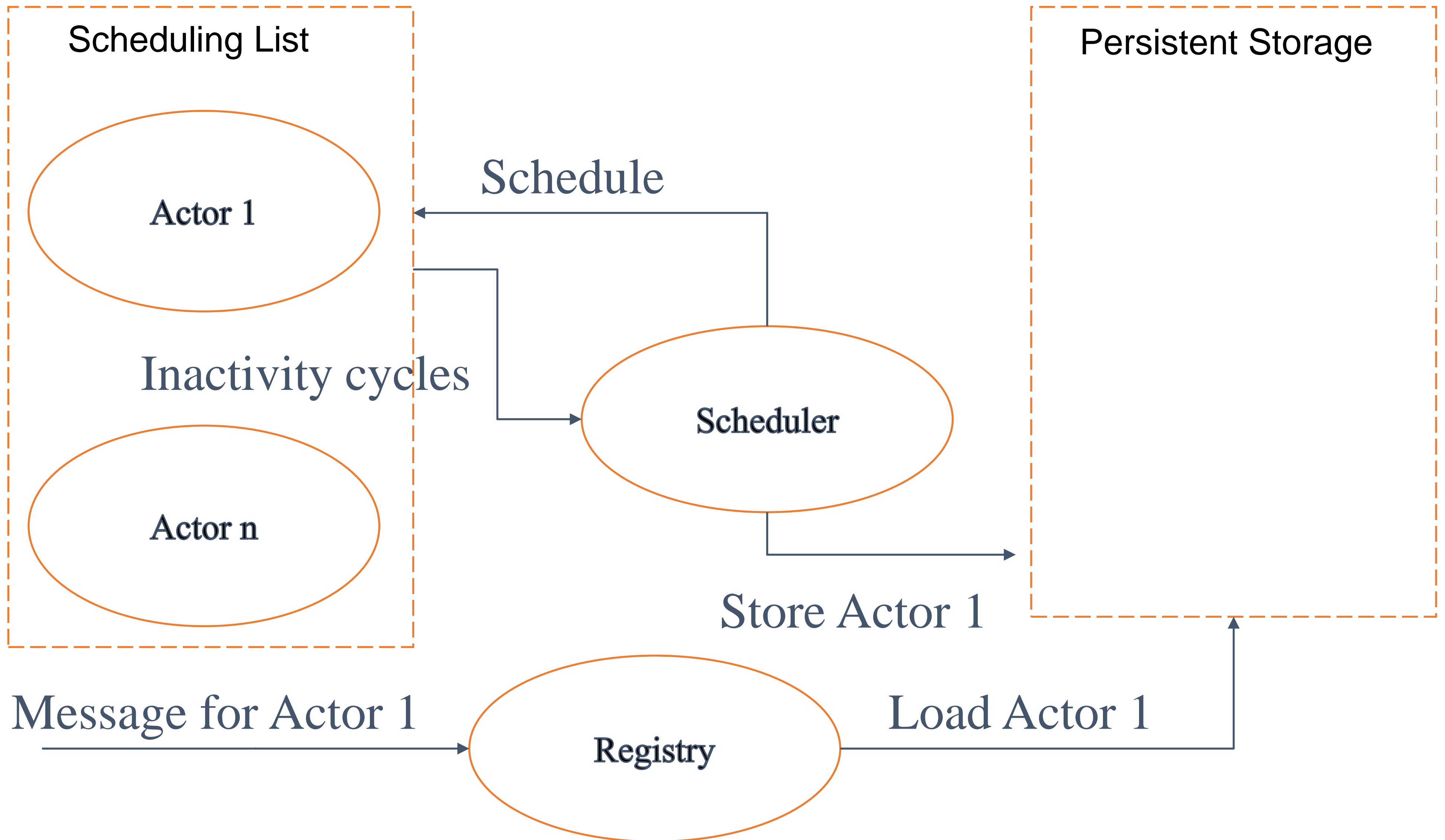
Execution Profiles

- Active
  - Each actor has its own execution thread
- Passive
  - All actors share an execution thread
- Shared
  - All actors share an execution thread
  - Their mailboxes share the same message queue
- Resizable
  - Inactive actors can be moved to a persistent storage
- Hybrid
  - Some actors can have their own execution thread

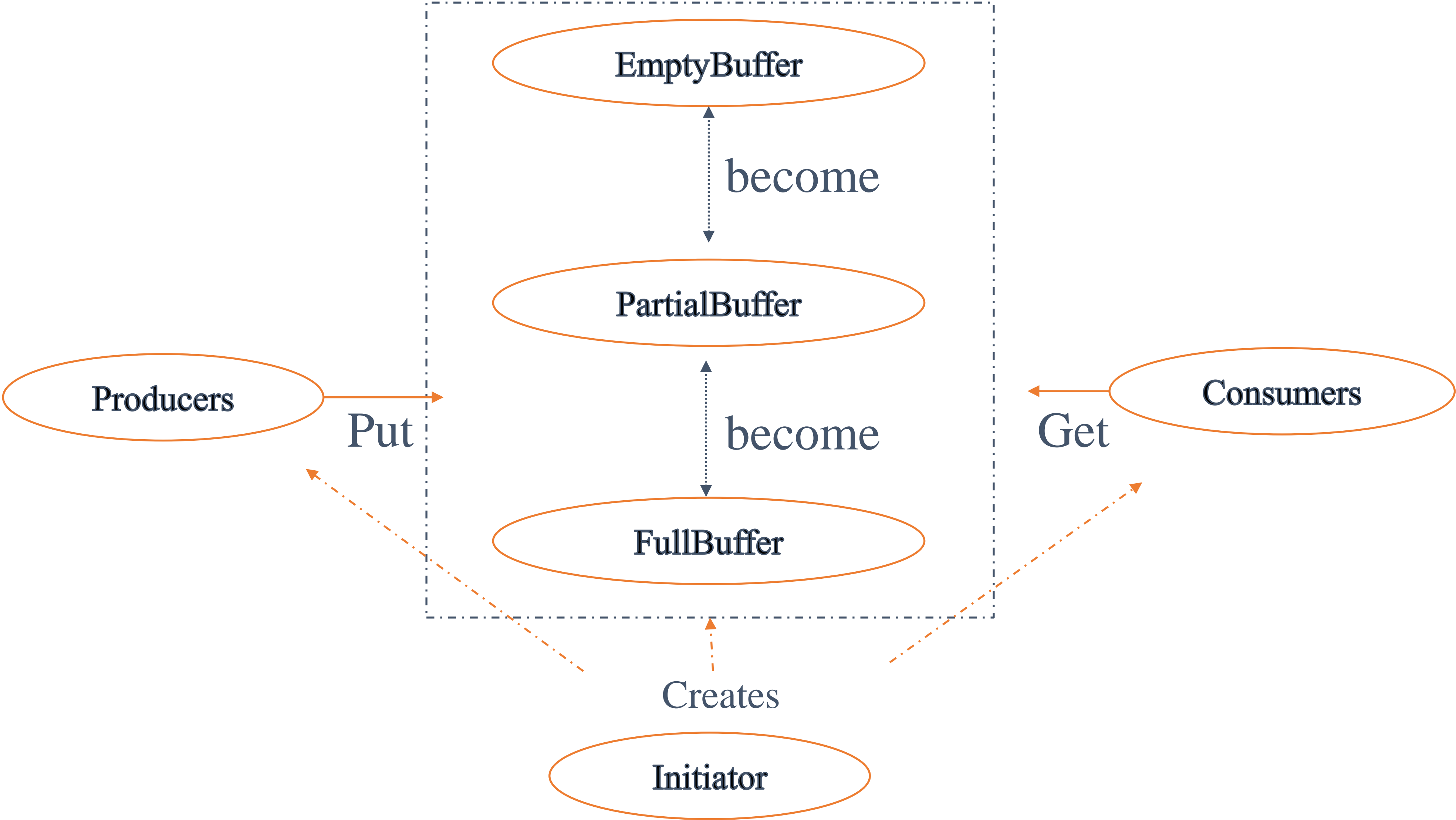
Shared Profile



Resizable Profile



Bounded Buffer Example (1/4)



Bounded Buffer Example (2/4)

```
public final class EmptyBuffer extends Behavior {
  public List<Case> initialize(final Object[] v) {
    BufferState s = new BufferState();

    s.setCapacity((Integer) v[0]);
    setState(s);

    return initialize();
  }

  public List<Case> initialize() {
    ArrayList<Case> l = new ArrayList<>();

    l.add(new PutCase());
    l.add(new Killer());

    return l;
  }
}
```

Initializes the behavior when the actor is created

Initializes the behavior when the actor moves to a new one

Bounded Buffer Example (3/4)

```
public final class GetCase extends Case {
  GetCase() {
    super(new MessagePattern(
        MessagePattern.CONTENT, new IsInstance(Get.class)));
  }

public void process(final Message m) {
    BufferState s = (BufferState) getState();

    send(m, s.remove());

    if (s.size() == 0) {
      become(EmptyBuffer.class);
    }
    else if (getBehavior().equals(FullBuffer.class.getName())) {
      become(PartialBuffer.class);
    }
  }
}
```

Sets the message pattern

Processes the message

Bounded Buffer Example (4/4)

```
public static void main(final String[] v) {
  final long lifetime  = 1000;
  final int size       = 10;
  final int producers  = 10;
  final int consumers  = 10;

  Configuration c = Controller.INSTANCE.getConfiguration();

  c.setScheduler(ThreadPoolScheduler.class.getName());

  c.setArguments(
      Initiator.class.getName(),
      new Object[] {lifetime, size, producers, consumers});

  Controller.INSTANCE.run();
}
```

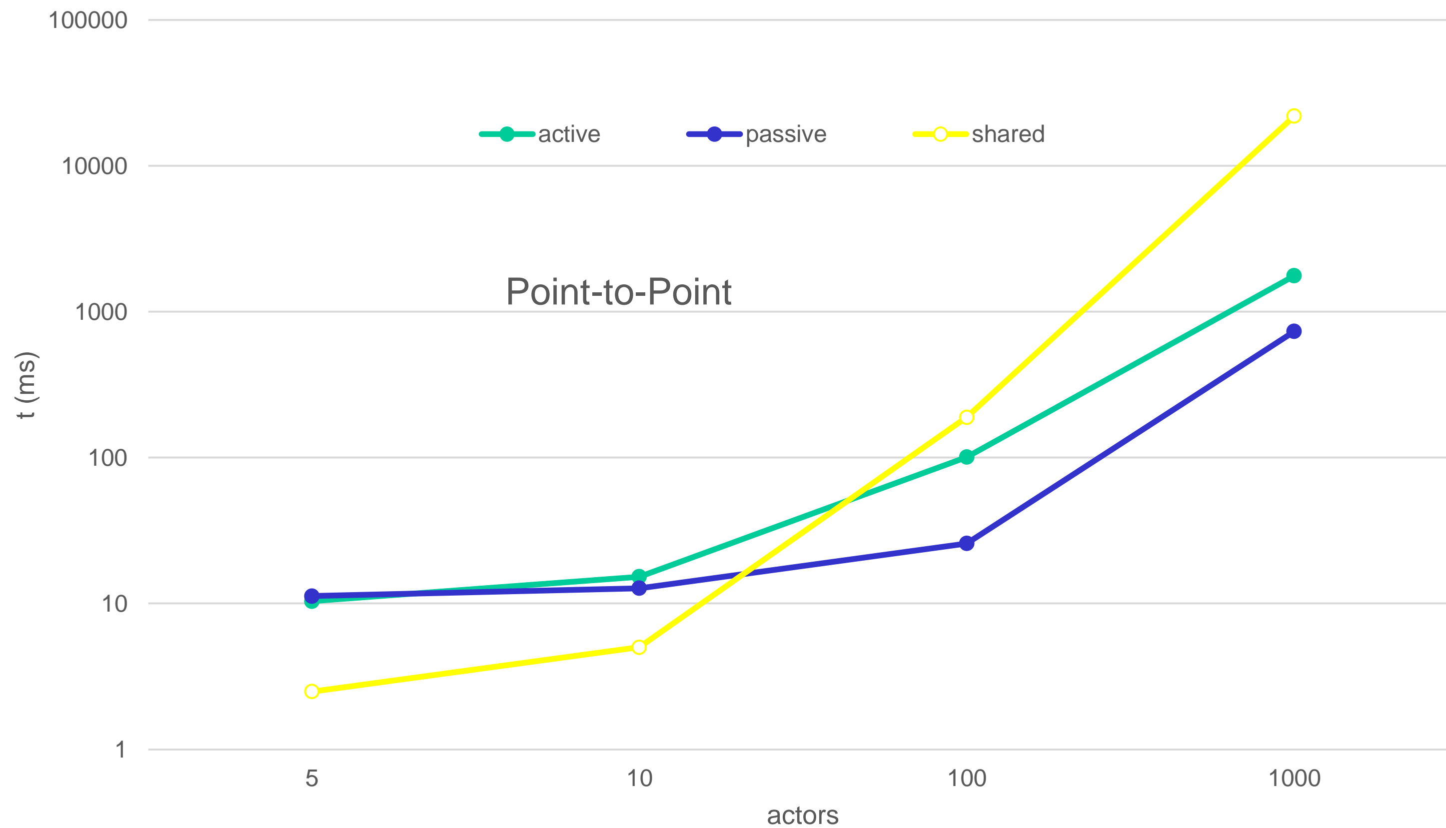
Sets the scheduler

Sets the initialization arguments of the scheduler

Starts the execution

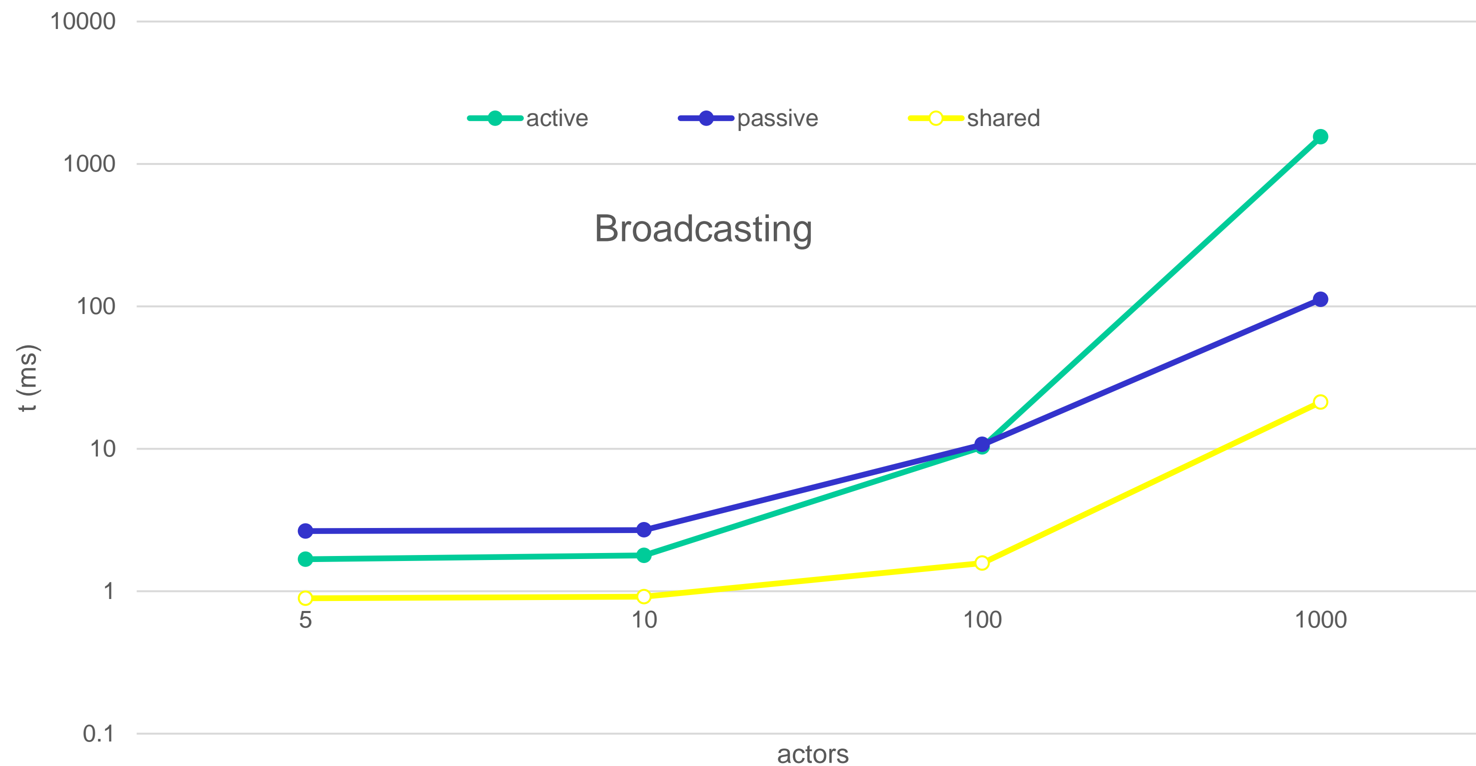
Performance Comparison (1/3)

Intel Core 2 - 2.90GHz - 16 GB RAM
Windows 8 64bit - Java 7 - 8 GB heap size



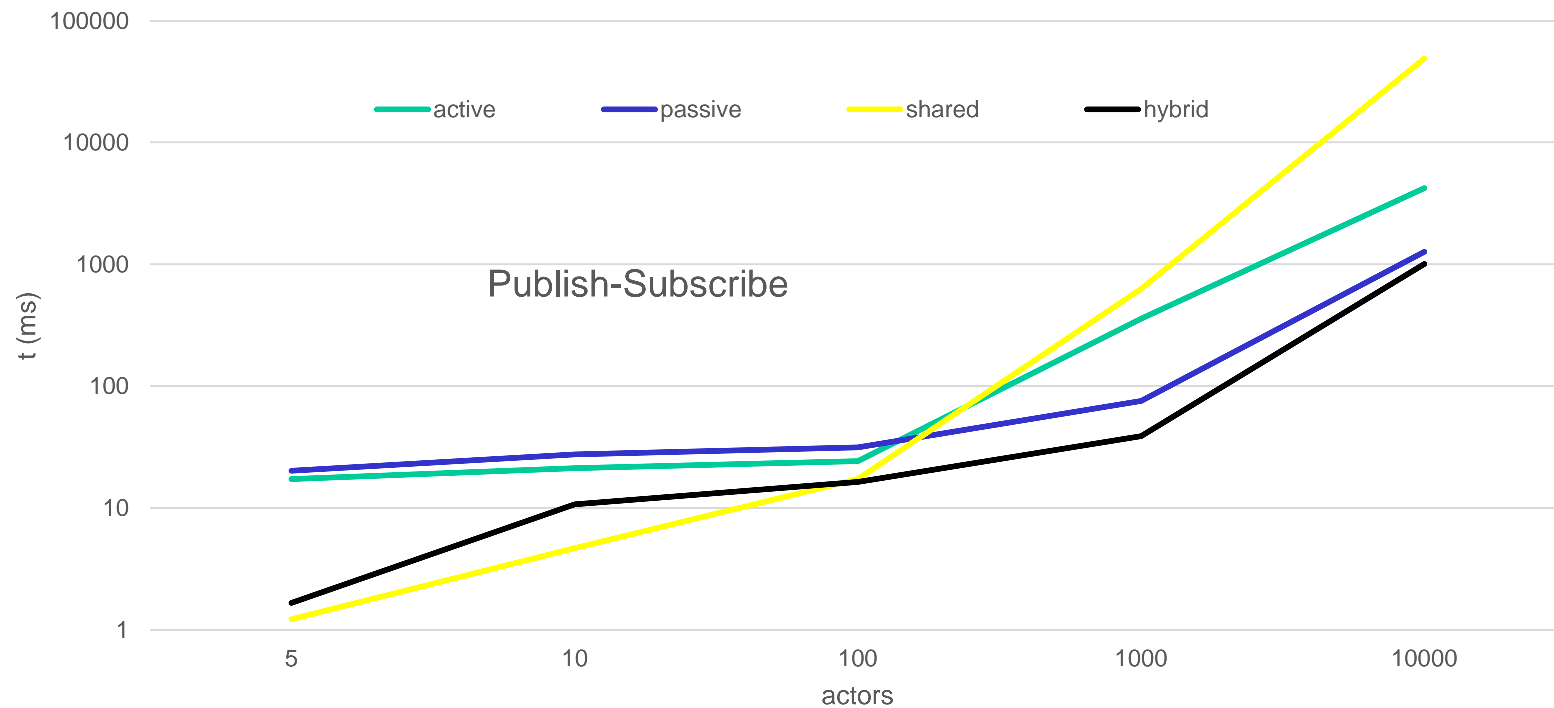
Performance Comparison (2/3)

Intel Core 2 - 2.90GHz - 16 GB RAM
Windows 8 64bit - Java 7 - 8 GB heap size



Performance Comparison (3/3)

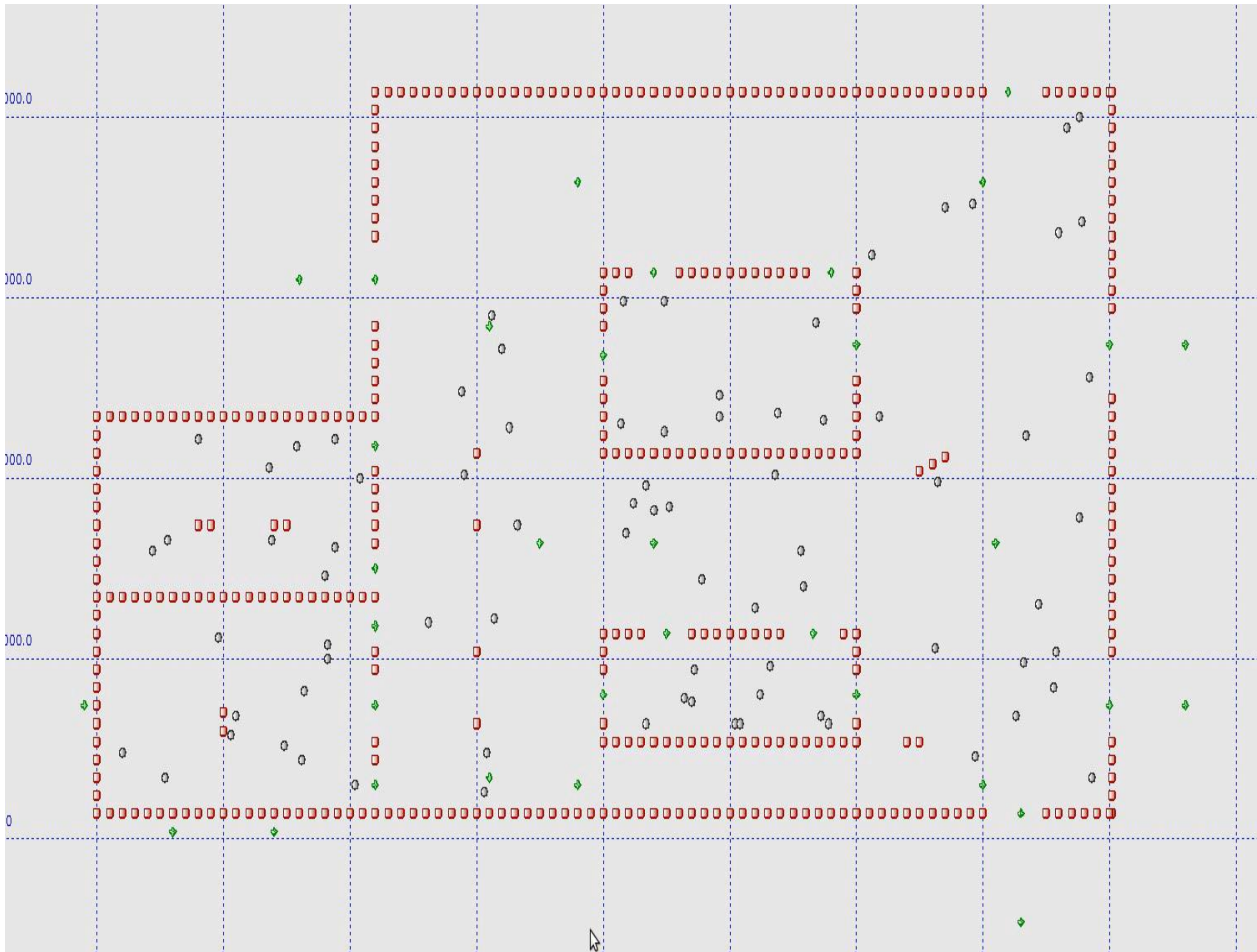
Intel Core 2 - 2.90GHz - 16 GB RAM
Windows 8 64bit - Java 7 - 8 GB heap size



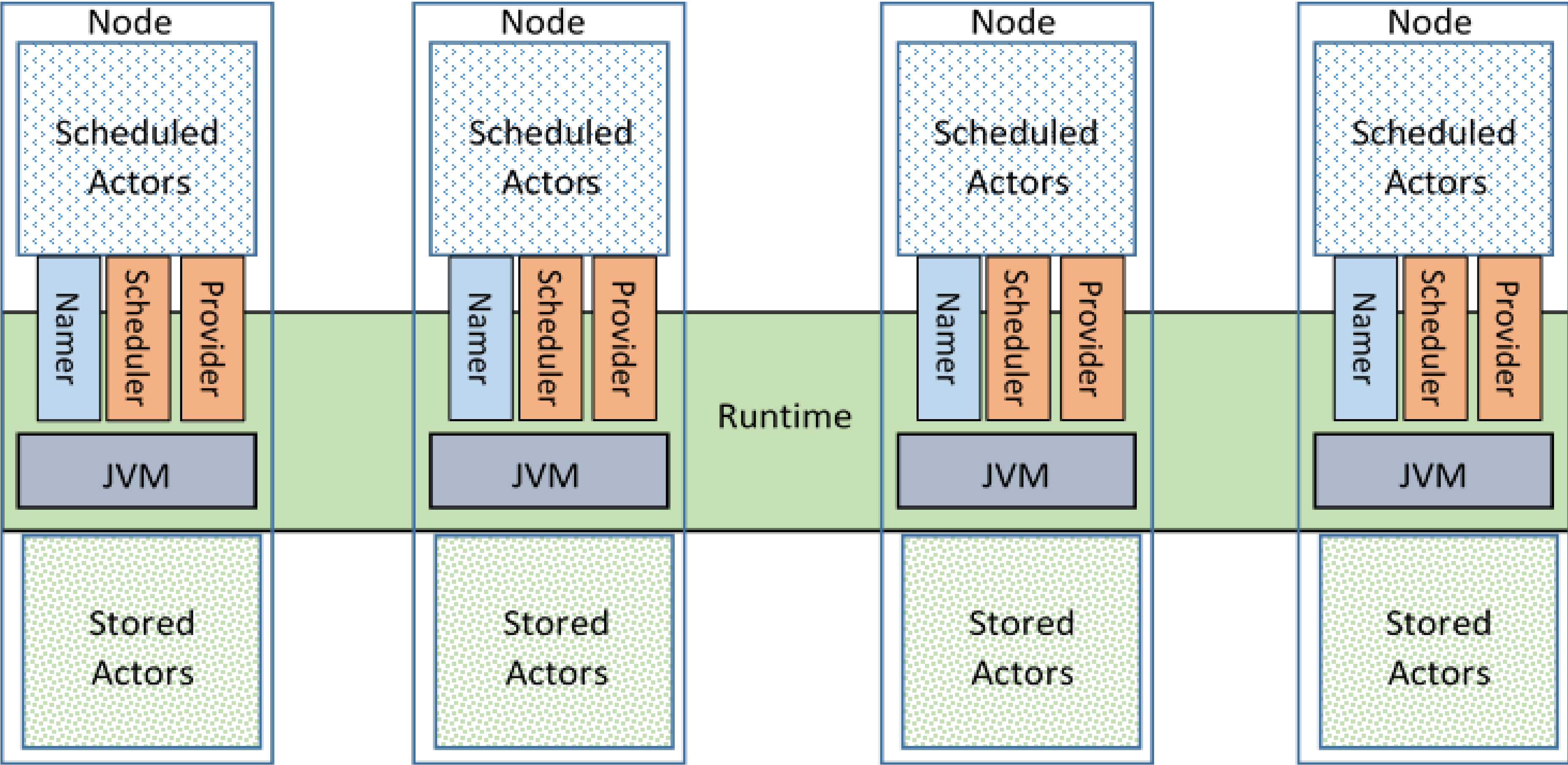
Experimentation

- Distributed information management
  - Peer-to-peer systems for the sharing of information
  - Distributed filtering of information
  - Data mining
- Agent-based modeling and simulation
  - Crowds simulation
  - Social networks and computer networks simulation and analysis

Crowds Simulation



Social Network Simulation and Analysis



Future Work

- Continue the experimentation and validation of the software framework
  - Smart city modeling and simulation
- Improve the of performance of distributed simulations by reducing the overhead due to
  - The synchronization of the actor space
  - The propagation of information between different actor spaces
- Support for a more *intelligent* interaction between actors

An Actor-based Software Framework for Scalable Applications

Federico Bergenti, Agostino Poggi, Michele Tomaiuolo

{federico.bergenti,agostino.poggi,michele.tomaiuolo}@unipr.it