A Distributed **Selectors** Runtime System for Java Applications

Group 13

Actors -> Selectors

Actors:

- ➢ All communication is **asynchronous**, difficult for concurrent coordination involving multiple actors.

Selectors:

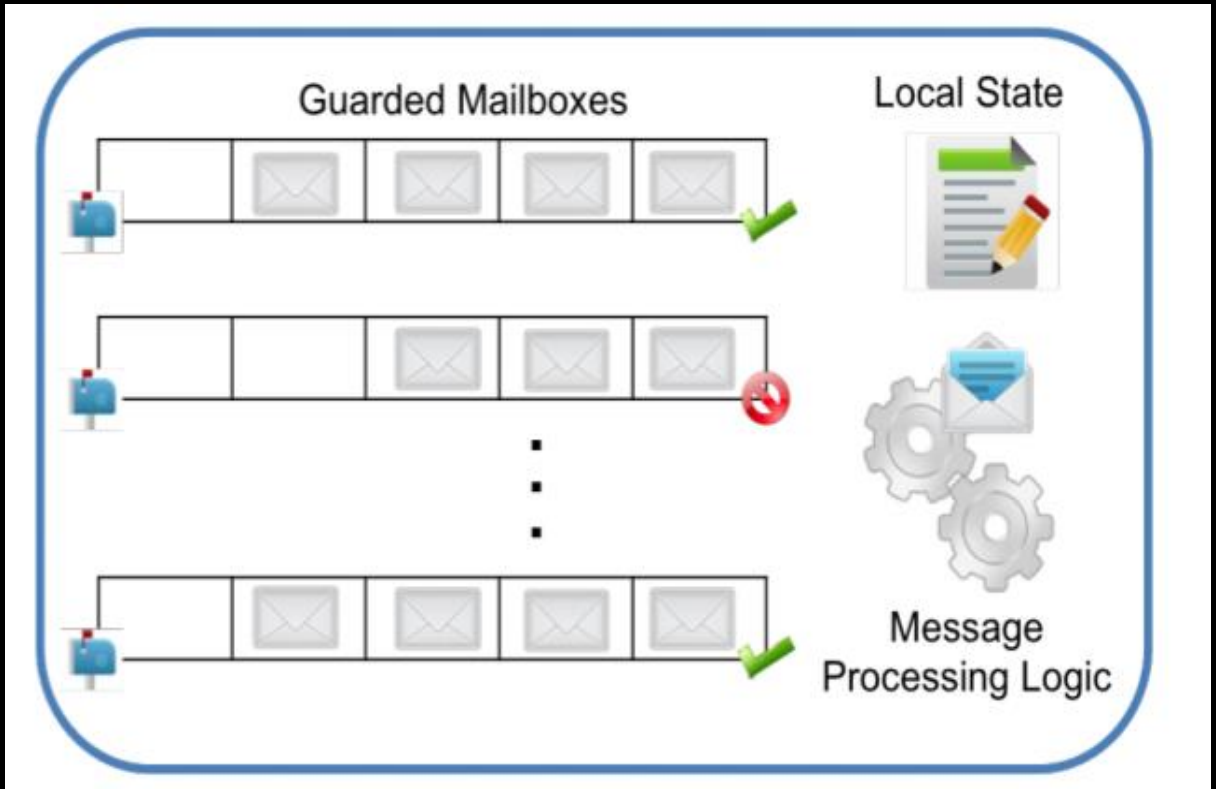
- ✓ An **execution unit** that has the capability to process incoming messages
- ✓ an **extension** of the Actor model

Selectors: Main Features

Selectors:

- ✓ **multiple mailboxes** to receive messages

- ➢ the sender and receiver can **determine where** to put.
- ➢ Process one message from a certain mailbox at a time.



Selectors: Main Features

Selectors:

- ✓ Each mailbox has a **boolean condition** (named **guard**)

- ➢ enable or disable a specific mailbox while processing a message
- ➢ Guard **does not affect** the mailbox's ability to receive messages.
- ➢ Besides modified by the selector, guards can be declared with **explicit expressions** (just like expressions in digital logic circuits)

Selectors: Life Cycle

- ✓ New
  - ➢ not guaranteed to instantiate immediately after creation
  - ➢ An access handle is immediately created and passed to the caller.
- ✓ Run
  - ➢ process messages in mailboxes of higher priority first
  - ➢ rotates between mailboxes with same priority
- ✓ Terminated
  - ➢ **all new operations** requested by this selector are **completed** and **no outgoing messages** remain in the local buffer.

Synchronous Request-Reply Pattern

- ✓ Task
  - ➢ The replier receives messages from the requestor and responds to it after processing messages.
  - ➢ The requestor **can not do anything** until receiving response messages.

Synchronous Request-Reply Pattern

- ✓ Designs for Actors
  - ➢ The sender stalls all other computations until it receives the corresponding reply.
  - ➢ The incoming messages before the response message must be stashed, and unstashed to the mailbox after processing the reply message.
  - ➢ hard to implement efficiently

Synchronous Request-Reply Pattern

- ✓ Solutions for Selectors
  - ➢ Define two **separate mailboxes**, one to receive **regular messages** including all the request messages and another mailbox to receive **only synchronous response messages**.
  - ➢ When a receiver expected to process a synchronous response message, disables the REGULAR mailbox until finish it.

Join Patterns in Streaming

- ✓ Task
  - ➢ Messages from two or more data streams are combined into a single message.
  - ➢ match the data from all sources before processing the messages.

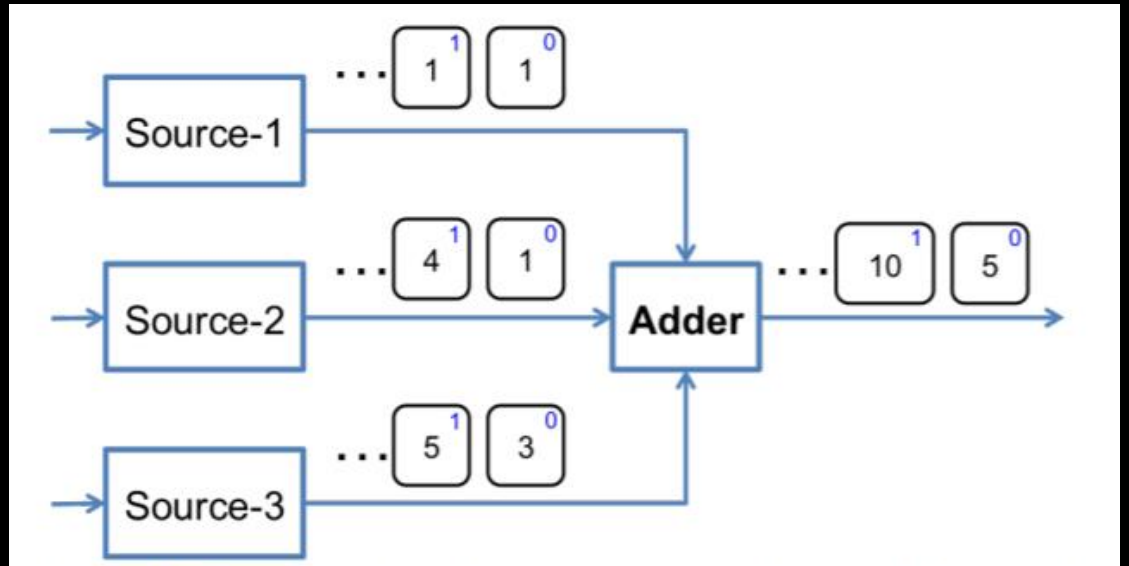


Figure 4: Actor network simulating a join pattern. `Source-1`, `Source-2`, and `Source-3` are producers for data streams. The `Adder` actor aggregates data items from each of the three sources and sums them up.

Join Patterns in Streaming

- ✓ Designs for Actors
  - ➢ The **order** of processing of messages is **not guaranteed** on the sender actors.
  - ➢ Keep a track of all the in-flight messages from various sequence numbers.
  - ➢ Tag messages with source and serial number.

Join Patterns in Streaming

- ✓ Solutions one for Selectors
  - ➢ The number of mailboxes the receiver has is just the number of sources.
  - ➢ All mailboxes are enabled at first. When receiving a message from source i, disable mailbox i.
  - ➢ If matched size reaches the specific number, process these messages and enable all messages.

Join Patterns in Streaming

- ✓ Solutions two for Selectors
  - ➢ Only mailbox 0 is enabled at first.
  - ➢ When receiving a message from source i, disable mailbox i and enable mailbox i+1.
  - ➢ If matched size reaches the specific number, process these messages. Then enable mailbox 0 and repeat.

Producer-Consumer Pattern

- ✓ Task
  - ➢ the producer pushes work into the buffer as work is produced and the consumer pulls work from the buffer when they are ready to execute.
  - ➢ We aim to implement a buffer bound to respond the producer and consumer.

Producer-Consumer Pattern

- ✓ Designs for Actors
  - ➢ When the buffer is empty and the consumer requests work, the consumer is placed in a queue until available.
  - ➢ When producers are ready to produce data and the buffer is full, the producer is placed in a queue until the buffer is empty.
  - ➢ Additional complexity to maintain queues for the available producers and consumers.

Producer-Consumer Pattern

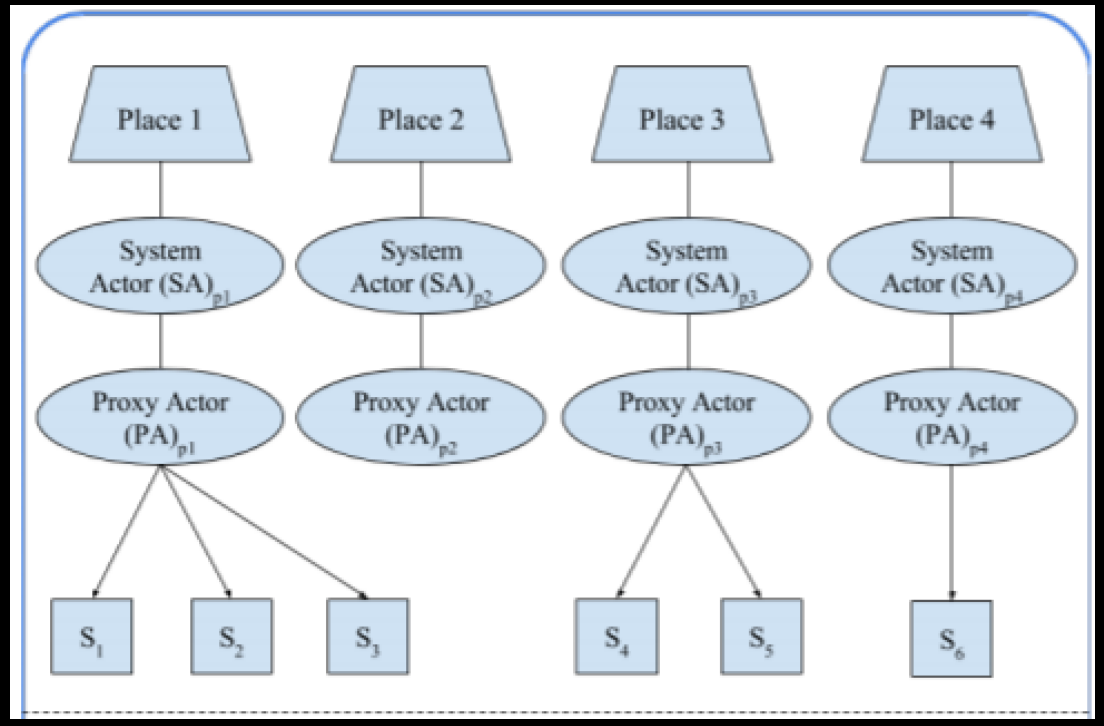
- ✓ Solutions for Selectors
  - ➢ The buffer handles two mailboxes, one for producer and other for consumer.
  - ➢ Simply use the following **explicit expressions.**

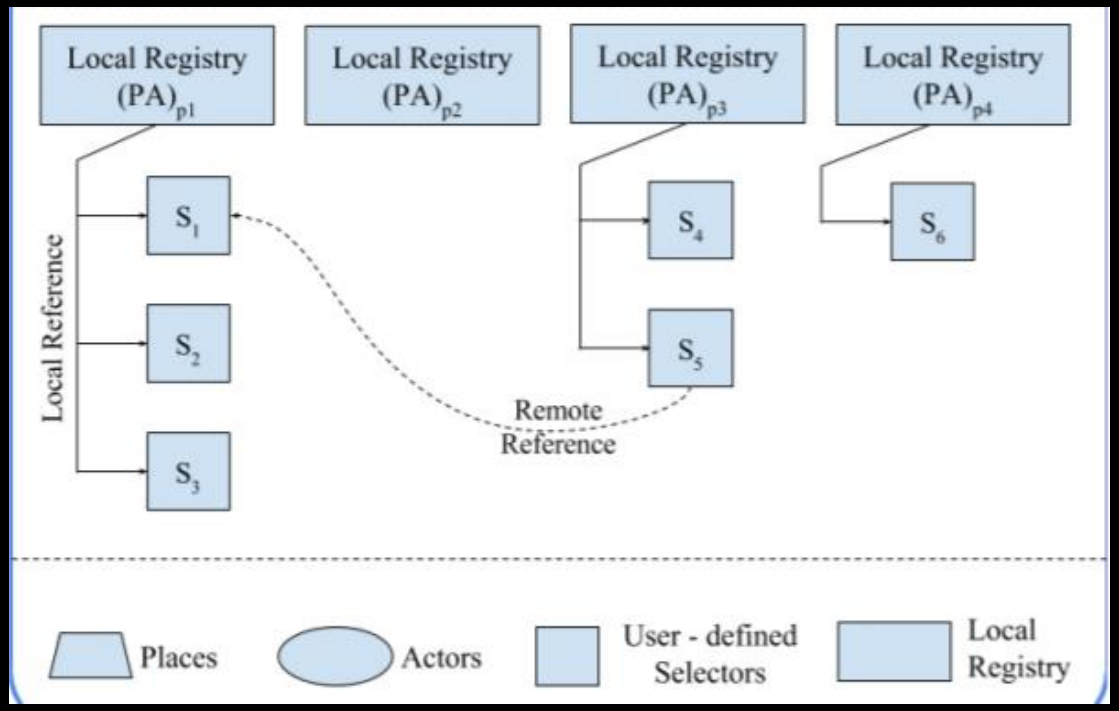
    guard(PRODUCER, dataBuffer .size () < thresholdSize)
    guard(CONSUMER, !dataBuffer .isEmpty ())

Distributed Selector System

- ✓ The structure in one unit (which is called place or node)
  - ➢ One System Actor: maintains the internal state, as well as communicating information with other places.
  - ➢ One Proxy Actor: coordinating the messages between local and remote selectors.
  - ➢ A local registry: maintained by proxy actor, stores the address of selectors located in the same place.

Distributed Selector System

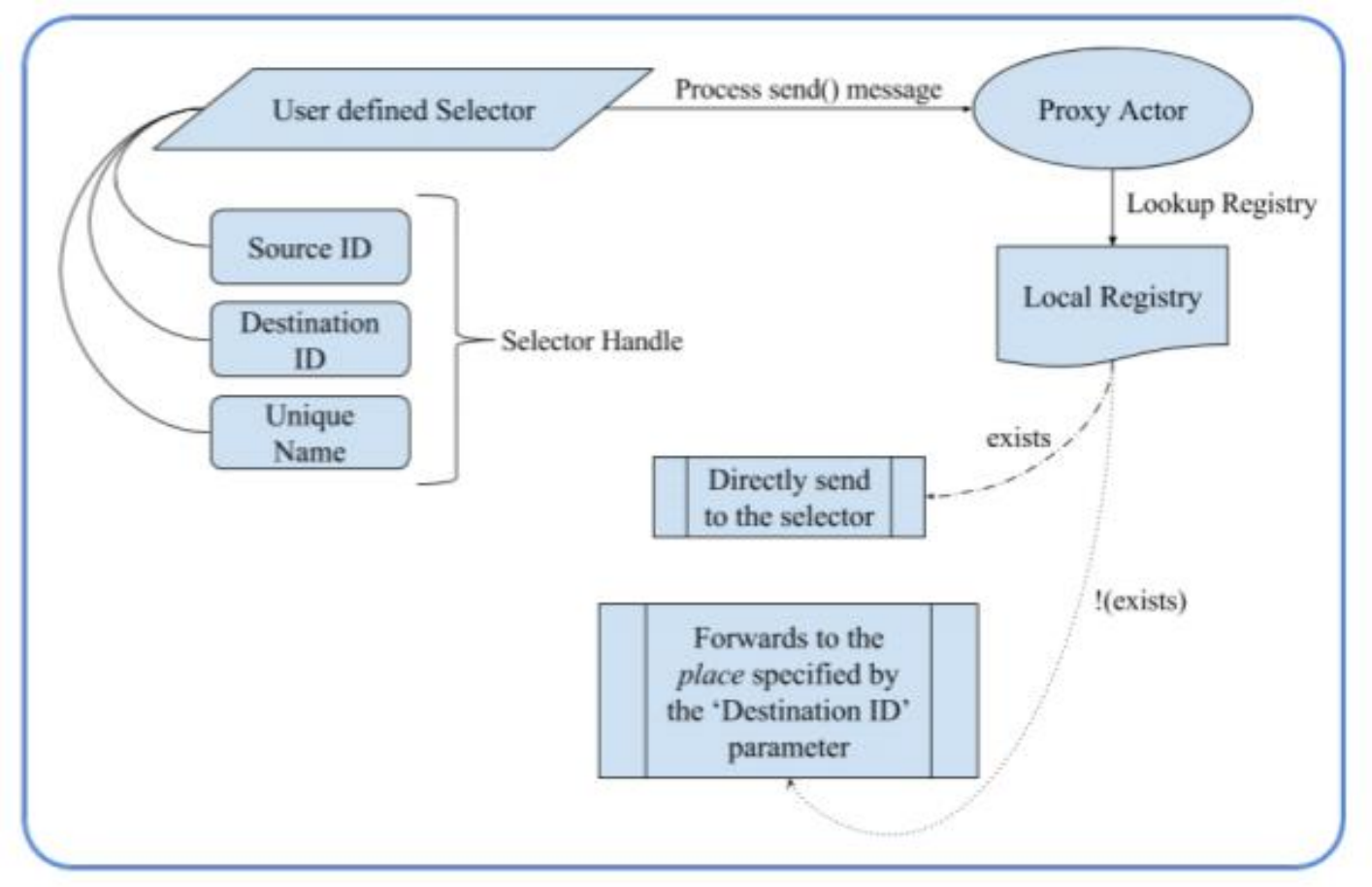




Distributed Selector System

- ✓ When message is passing…
  - ➢ If the destination place matches the local place then the Proxy Actor looks up its local registry to find the selector and forwards the messages.
  - ➢ If the destination place is remote then the Proxy Actor forwards the message to that specified place. The Proxy Actor at the destination place will further use the local registry to forward the message to the specific selector.

Distributed Selector System



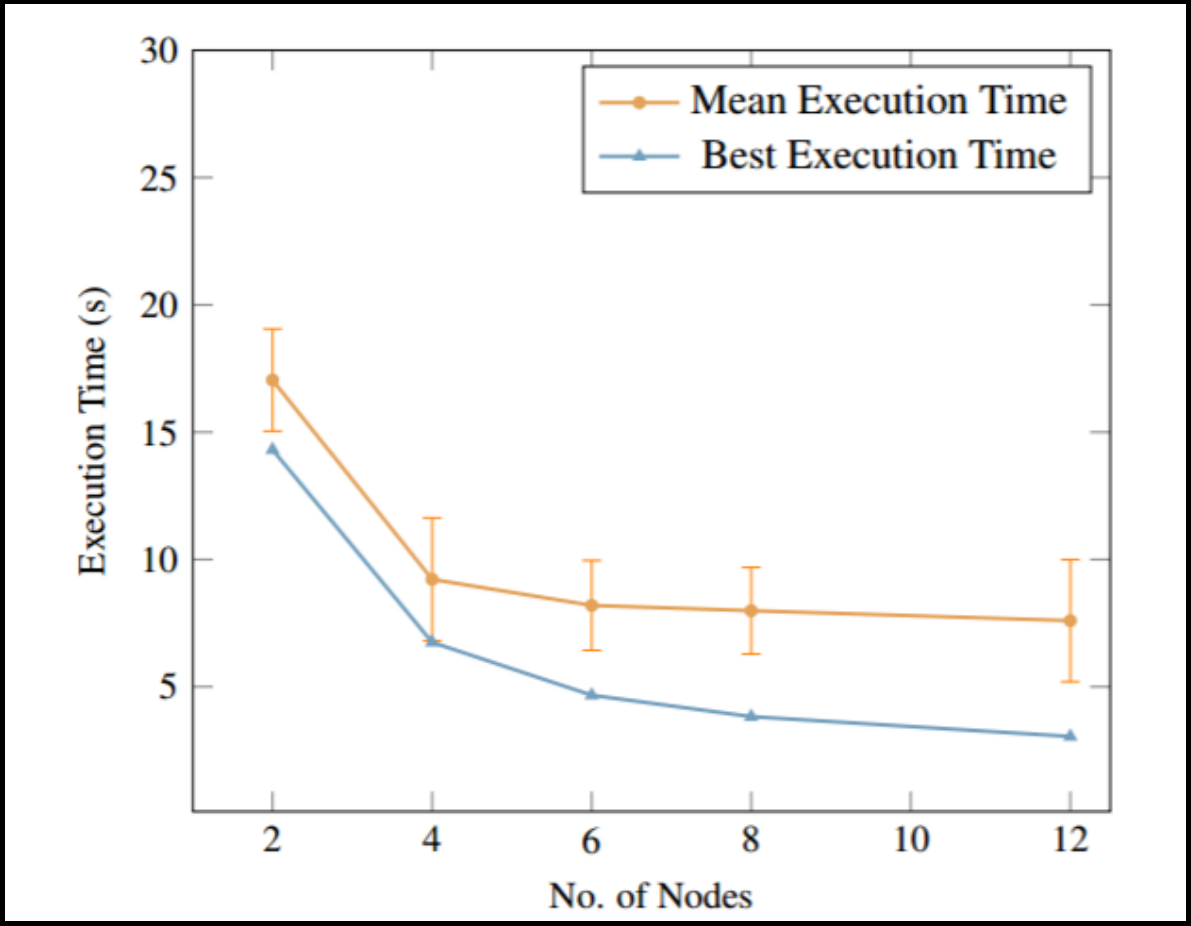
NQueens First K Solutions

- ✓ Solution for Selectors
  - ➢ Each time a worker successfully place a non-attacking queen on the partial solution, the worker reports the partial board back to master.
  - ➢ The master either assigns the partial solution to a worker in a round-robin fashion or records that a valid solution is found.
  - ➢ Exploits the priority feature in our DS implementation, places a higher priority on work items that contain complete partial solutions (i.e. with more safely placed queens).

NQueens First K Solutions

- ✓ Comparison with Actors
  - ➢ With a naive actor-based implementation, a solution limit to allow termination has no effect on the program, and the program has to exhaustively compute the solution space.
  - ➢ Without using priority, depth-first search with a divide-and-conquer style is hard to implement, since messages are processed in their received order.

NQueens First K Solutions



Speedup decreases when number of nodes is more than 4.

Due to Duplication Avoidance

- Master ask each worker to omit those self-duplicate solutions
- Cross-worker duplication cannot be avoided
- More workers, more duplication for the master

Thank You.