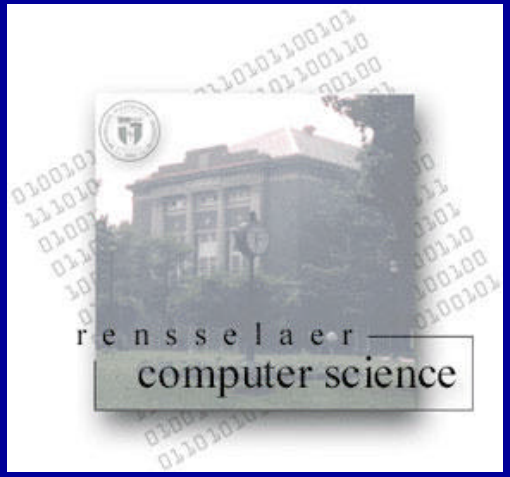
SALSA: Language and Architecture for Widely Distributed Actor Systems.



Carlos Varela, cvarela@cs.rpi.edu
Abe Stephens, stepha@cs.rpi.edu
Department of Computer Science
Rensselaer Polytechnic Institute
Troy NY, USA
http://www.cs.rpi.edu/wwc/SALSA/

AAMAS 2002, University of Bologna Italy.

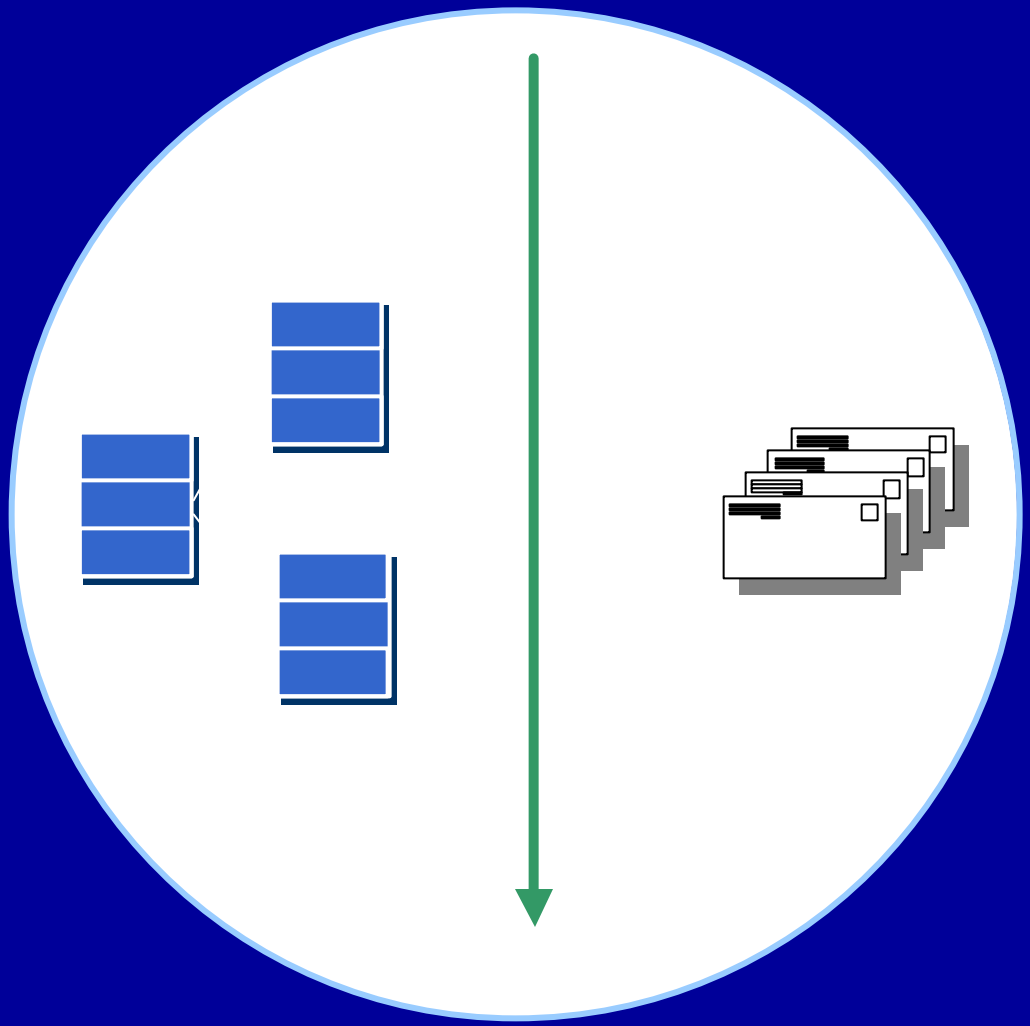
Actor Fundamentals

- Introduced by C. Hewitt (77), further refined and developed by G. Agha et al (85-present)
- An Actor encapsulates a thread of execution with a collection of objects.
- Only the actor's thread may access its objects directly and change their state.
- Provides for implicit object synchronization.

Actor Fundamentals

- Actors communicate by sending messages to each other.
- Messages are sent asynchronously.
- Messages are not necessarily processed in the order they are sent or received.

Actor Implementation



http://www.cs.rpi.edu/wwc/salsa

Worldwide Computing

- Distributed computing over the Internet.
- Access to *large number* of processors *offsets* slow communication and reliability issues.
- Seeks to create a platform for many applications.

World-Wide Computer

- Worldwide Computing platform implementation.
- Provides a runtime middleware for actors.
- Includes support for naming services.
- Message sending protocol.
- Support for actor migration.

Remote Message Sending Protocol

- Messages between actors are sent using RMSP.
- RMSP is implemented using Java Object Serialization.
- Protocol used for both message sending and actor migration.
- When an actor migrates, its location changes but its name does not.

WWC Theaters

- Theater programs provide execution location for actors.
- Provide a layer beneath actors for message passing.
- Example location:

rmsp://wwc.cs.rpi.edu/calendarInstance10

Theater address and port.

Actor location.

Environmental Actors

- Theaters may provide *environmental actors*.
- Perform actions specific to the theater and are not mobile.
- Include standard input and standard output actors.

Universal Naming

- Consists of *human readable* names.
- Provides location transparency to actors.
- Name to location mappings efficiently updated as actors migrate.

Universal Actor Naming

- UAN servers provide mapping between static names and dynamic locations.
  - Example:

uan://wwc.cs.rpi.edu/stepha/calendar

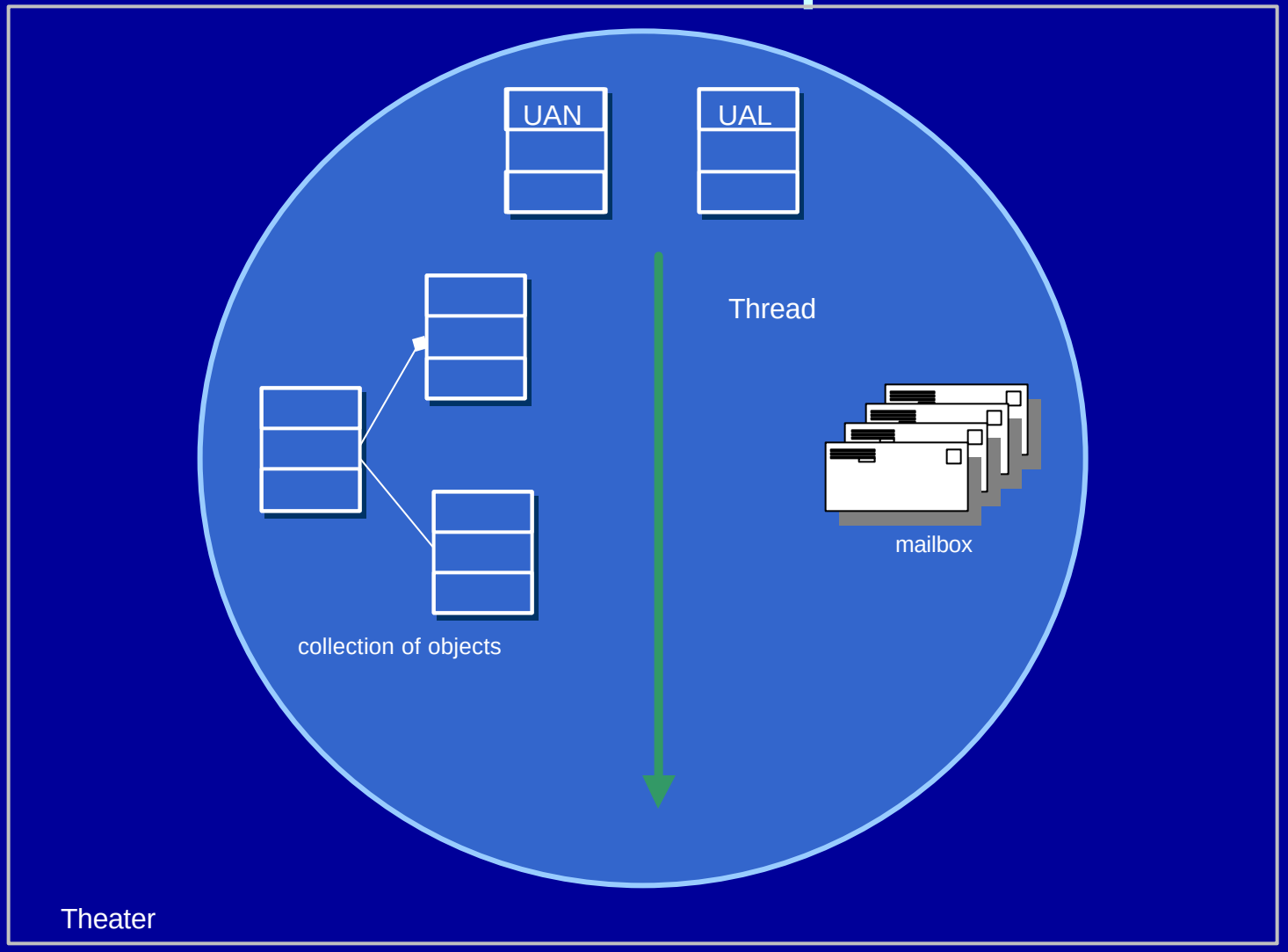
Name server address and port.

Actor name.

Universal Actors

- Universal Actors extend the actor model by associating a location and a universal name with the actor.
- Universal Actors may migrate between theaters and update the name server.

Universal Actor Implementation



http://www.cs.rpi.edu/wwc/salsa

Simple Actor Language System and Architecture

- SALSA is an actor oriented programming language.
- Supports Universal Naming (UAN & UAL).
- Primitives for
  - Message sending.
  - Migration.
  - Coordination.
- Closely tied to WWC platform.

SALSA Basics

- Programmers define *behaviors* for actors.
- Messages are sent asynchronously.
- Messages are modeled as potential method invocations.
- Continuation primitives are used for coordination.

Message Sending

```
TravelAgent a = new TravelAgent();

a<-book( flight );
```

Remote Message Sending

- Obtain a remote actor reference by name.

```
TravelAgent a = new TravelAgent();
a<-getReferenceByName(“uan://myhost/ta”) @
  a<-printItinerary();
```

- Obtain a remote actor reference by location.

```
a<-getReferenceByLocation(“rmsp://myhost/agent1”) @
    a<-printItinerary();
```

Migration

- Creating a new Actor an migrating it to a remote theater.

```
TravelAgent a = new TravelAgent();

a<-bind( “uan://myhost/ta”, “rmsp://myhost/agent1” ) @
    a<-book( flight );
```

- Obtaining a remote actor reference and migrating it.

```
a<-getReferenceByName(“uan://myhost/ta”) @
    a<-migrate( “rmsp://yourhost/travel” ) @
    a<-printItinerary();
```

Token Passing Continuation

- Insures that each message in the expression is sent after the previous message has been processed. It also allows that the return value of one message invocation may be used as an argument for a later invocation in the expression.
  - Example:

    ```
    a1<-m1() @ a2<-m2( token );
    ```

    Send `m1` to `a1` and then after m1 finishes, send the result with `m2` to `a2`.

Join Continuation

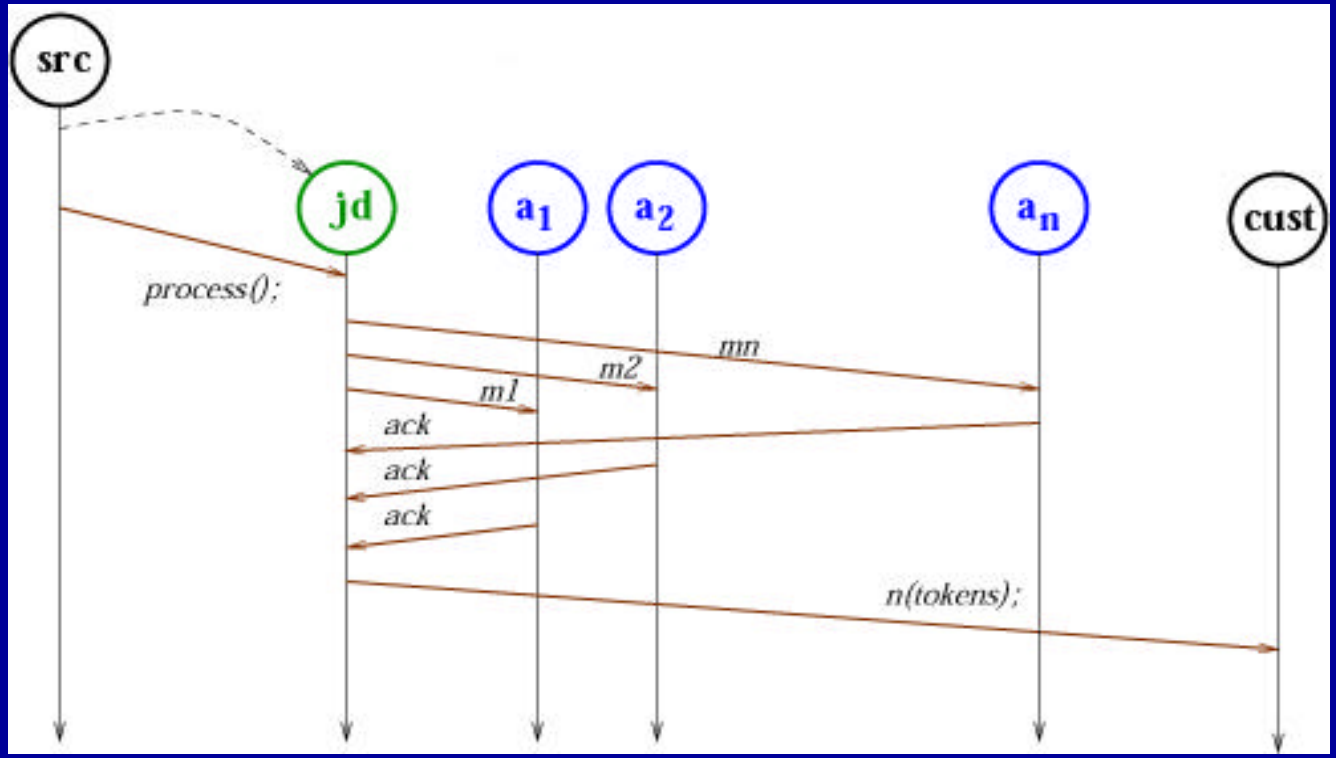
- Provides a mechanism for synchronizing the processing of a set of messages.
- Set of results is sent along as a *token*.
  - Example:

```
Actor[] actors = { searcher0, searcher1,
                   searcher2, searcher3 };
join( actors<-find( phrase ) ) @
  resultActor<-output( token );
```

Send the `find( phrase )` message to each actor in `actors[]` then after all have completed send the result to `resultActor` with an `output( … )` message.

Acknowledged Multicast

```
join( a1<-m1(), a2<-m2, a3<-m3(), … ) @
  cust<-n(token);
```



http://www.cs.rpi.edu/wwc/salsa

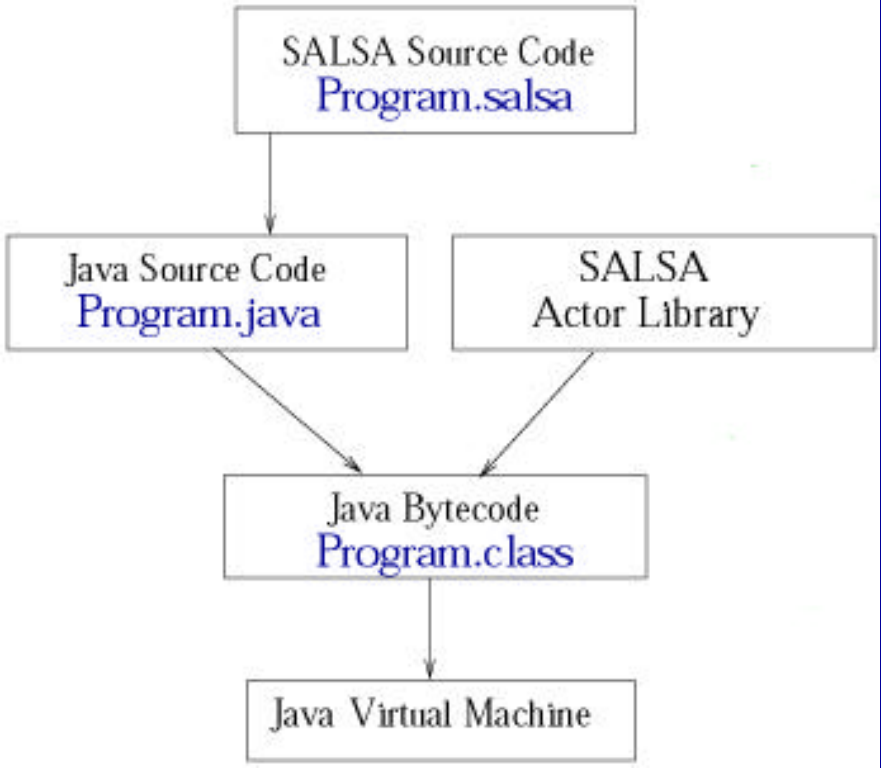
Lines of Code Comparison

	Java	Foundry	SALSA
Acknowledged Multicast	168	100	31

First Class Continuation

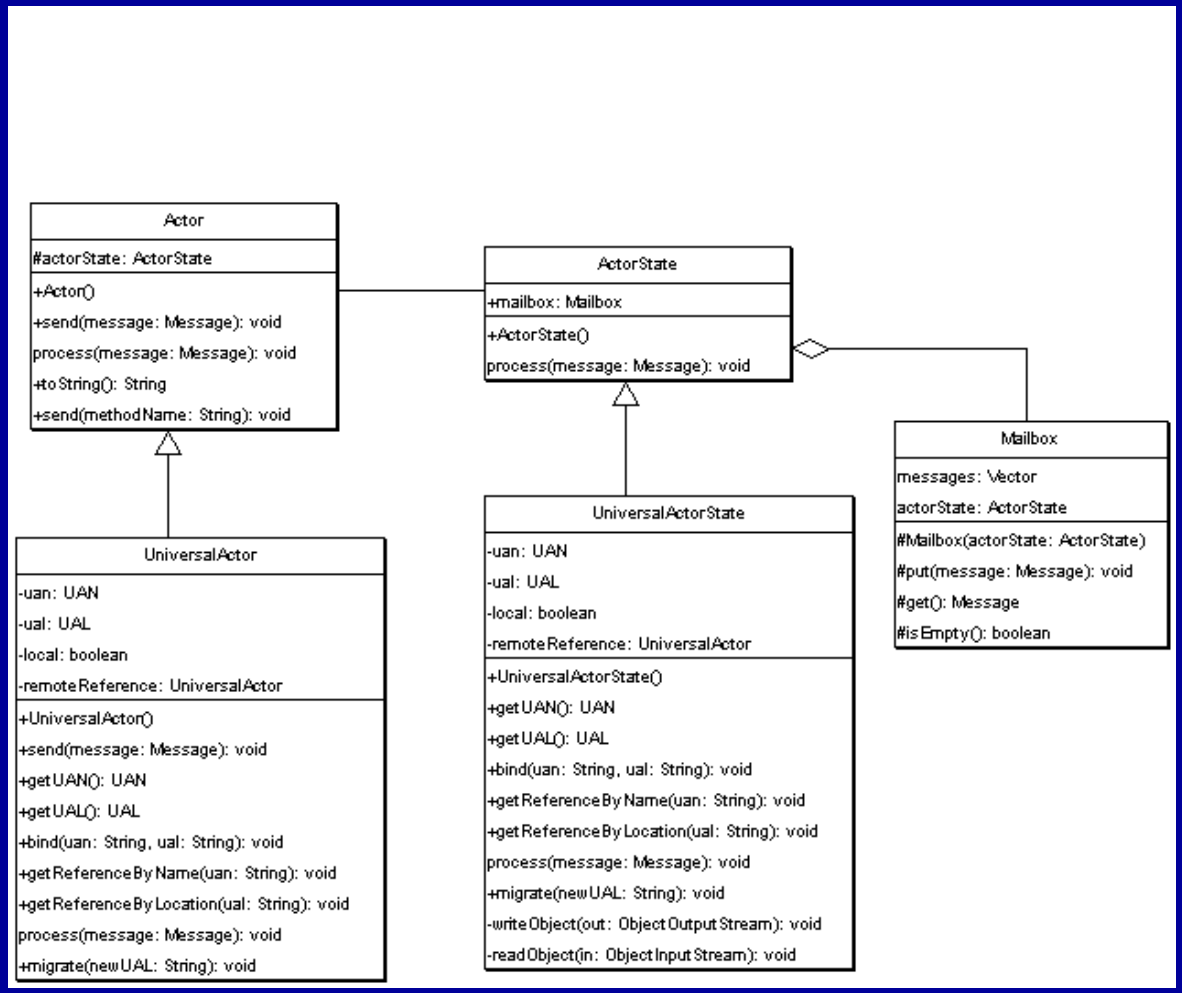
- Enable actors to delegate computation of a third party independently of the processing context.
- Unimplemented in current release.

SALSA and Java



- SALSA source files are compiled into Java source files before being compiled into Java byte code.
- SALSA programs may take full advantage of Java API.

SALSA Language Package



http://www.cs.rpi.edu/wwc/salsa

Hello World Example

```
module demo;

behavior HelloWorld {

   void act( String[] argv ) {

      standardOutput<-print( "Hello" ) @
      standardOutput<-print( "World!" );

   }

}
```

Hello World Example

- The `act( String[] args )` message handler is similar to the `main(…)` method in Java and is used to bootstrap SALSA programs.

Migration Example

```
module demo;

behavior Migrate {

   void print() {

      standardOutput<-println( "Migrate actor just migrated here." );
   }

   void act( String[] args ) {

      if (args.length != 3) {
         standardOutput<-println( “Usage: java migration.Migrate “ +
                                  “<uan> <ual1> <ual2>" );

         return;
      }

      bind( args[0], args[1] ) @
         print() @
         migrate( args[2] ) @
         print();

   }

};
```

Migration Example

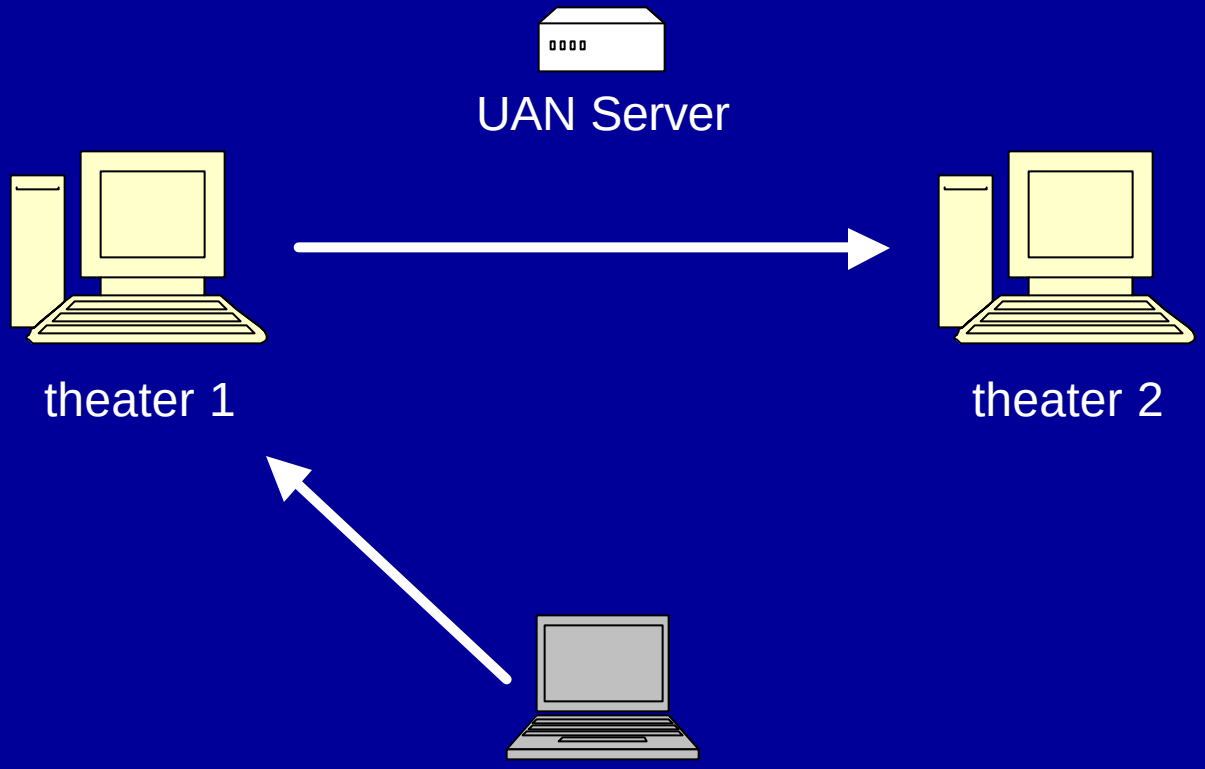
- The program must be bound to a valid name and location.
- After binding the actor sends the `print` message to itself before migrating to the second theater and sending the message again.

Compilation

```
$ java SALSACompiler demo/Migrate.SALSA
SALSA Compiler Version 0.3:  Reading from file demo/Migrate.SALSA . . .
SALSA Compiler Version 0.3:  SALSA program parsed successfully.
SALSA Compiler Version 0.3:  SALSA program compiled successfully.
$ javac demo/Migrate.java
$ java demo.Migrate
Usage: java migration.Migrate <uan> <ual> <ual>
$
```

- Compile Migrate.SALSA file into Migrate.java.
- Compile Migrate.java file into Migrate.class.
- Execute Migrate

Migration Example



The actor will print "Migrate actor just migrated here." at theater 1 then theater 2.

World Migrating Agent Example

Host	Location	OS/JVM	Processor
yangtze.cs.uiuc.edu	Urbana IL, USA	Solaris 2.5.1 JDK 1.1.6	Ultra 2
Vulcain.ecoledoc.lip6.fr	Paris, France	Linux 2.2.5 JDK 1.2pre2	Pentium II 350Mhz
Solar.isr.co.jp	Tokyo, Japan	Solaris 2.6 JDK 1.1.6	Sparc 20

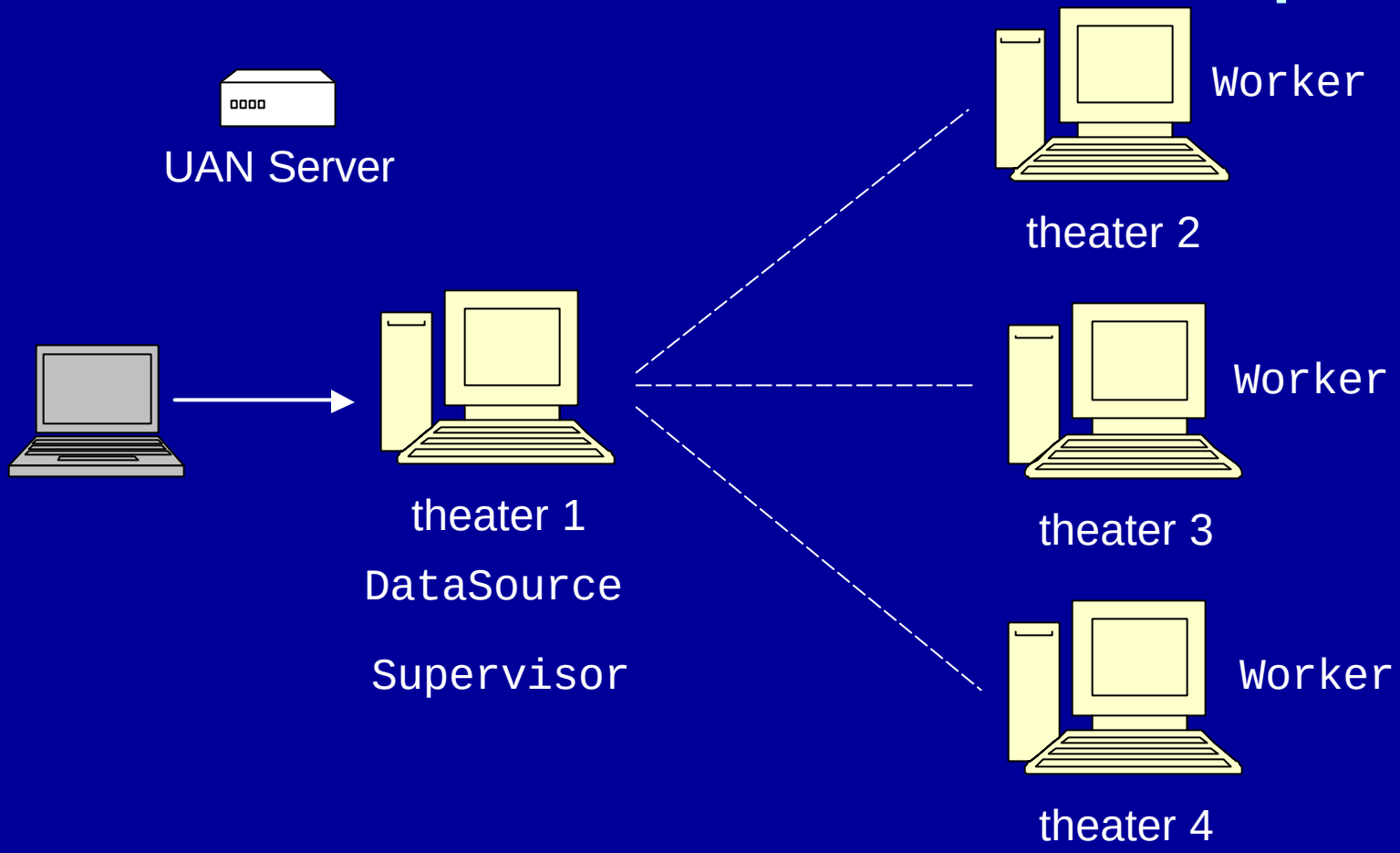
Local actor creation	386*u*s
Local message sending LAN message sending WAN message sending	148 *u*s 30-60 ms 2-3 s
LAN minimal actor migration LAN 100Kb actor migration WAN minimal actor migration WAN 100Kb actor migration	150-160 ms 240-250 ms 3-7 s 25-30 s

http://www.cs.rpi.edu/wwc/salsa

Mean Calculation Example

- `DataSource` actor assigns tasks to `Worker` actors.
- `Worker` actors on remote theaters calculate result and send it to a `Supervisor` actor.
- Coordinates between many `Worker` actors.

Mean Calculation Example



Web Search Example

- `Manager` actor multicasts search queries between distributed `Indexer` actors.
- Mobile `Indexer` actors create word lists from web sites.