Akka Remoting with Akka.Remote

Akka.NET

Petabridge

How This Course Works

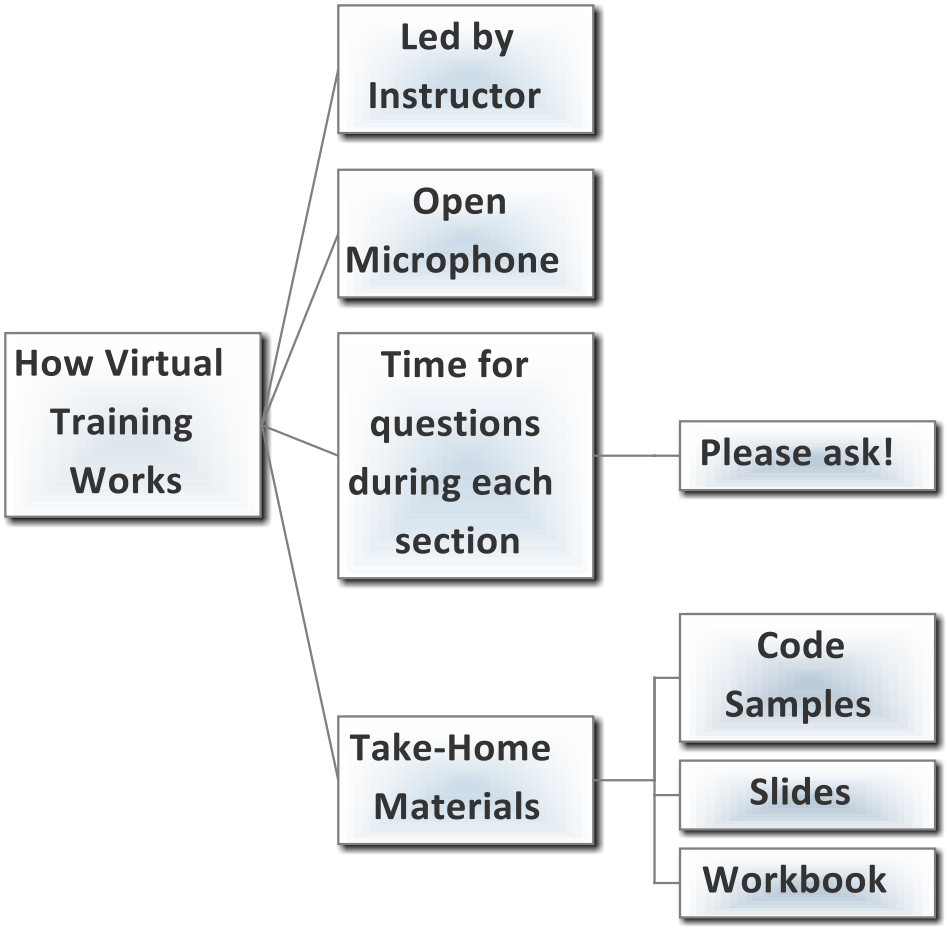
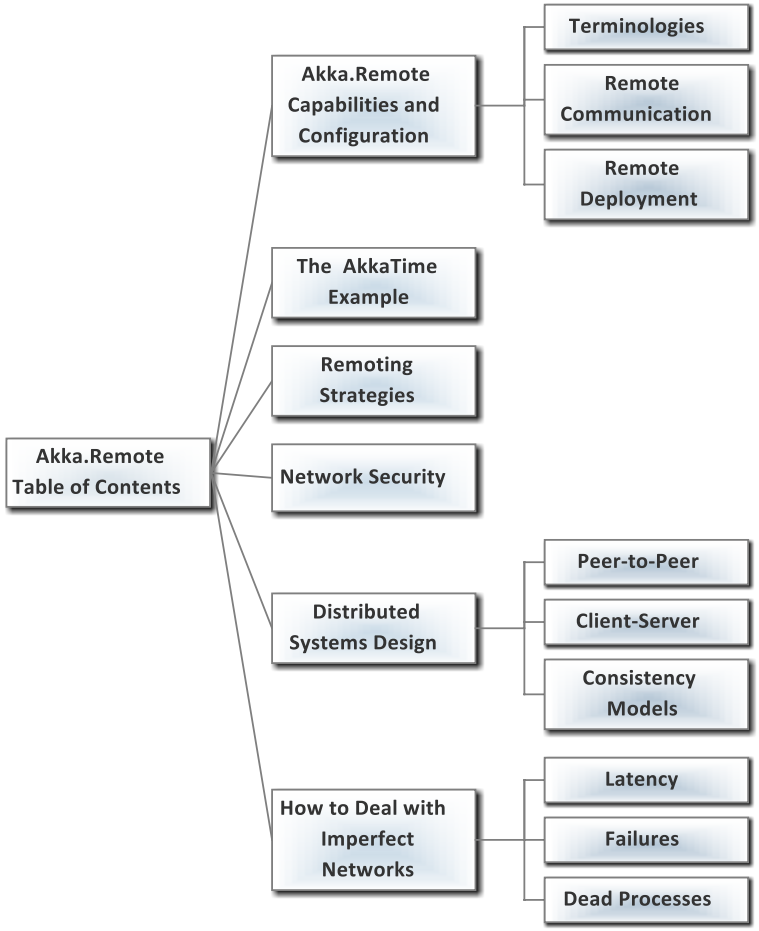


Table of Contents



Akka.Remote Capabilities and Configuration

Akka.NET's Extensibility Model

Akka.Remote Brings the Network to Akka.NET

A Trivial Akka.Remote Sample

CODE SAMPLE

Sample Configuration

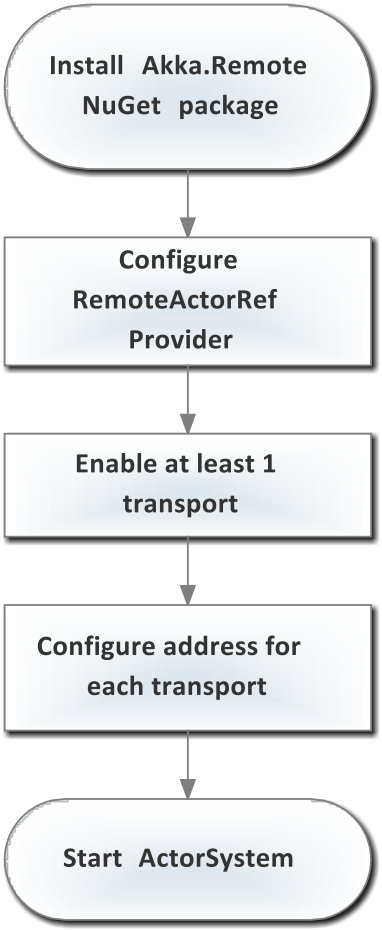
Server

```
akka {
   actor.provider = "Akka.Remote.RemoteActorRefProvider, Akka.Remote"
    remote {
        helios.tcp {
        port = 8081
        hostname = localhost
        }
    }
}
```

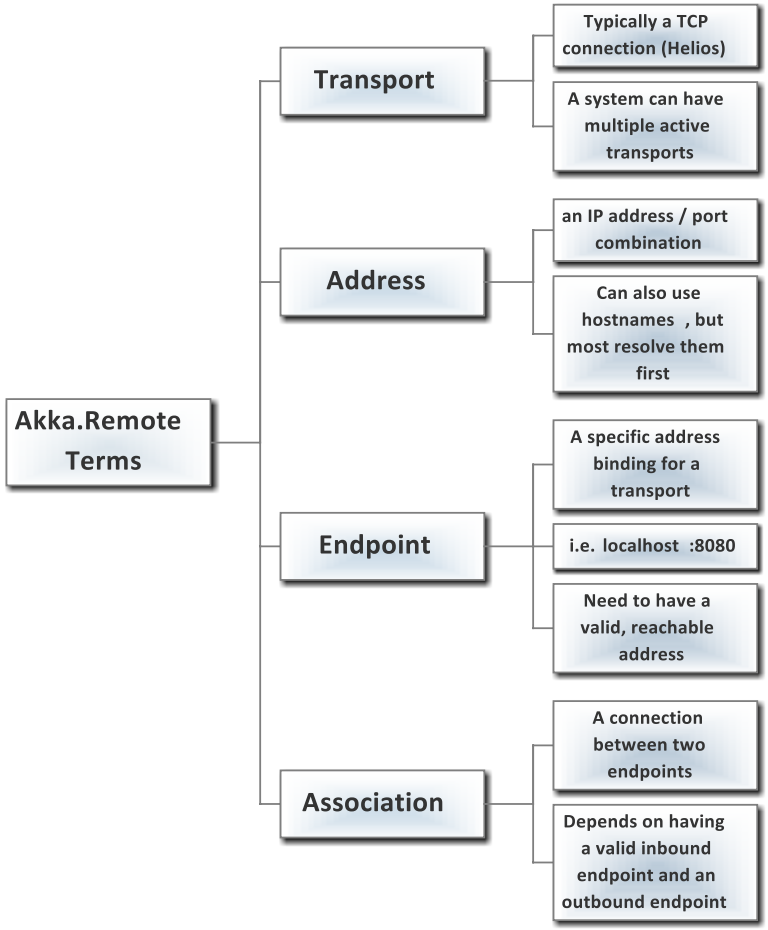
Client

```
akka {
   actor.provider = "Akka.Remote.RemoteActorRefProvider, Akka.Remote"
    remote {
        helios.tcp {
        port = 0
        hostname = localhost
        }
    }
}
```

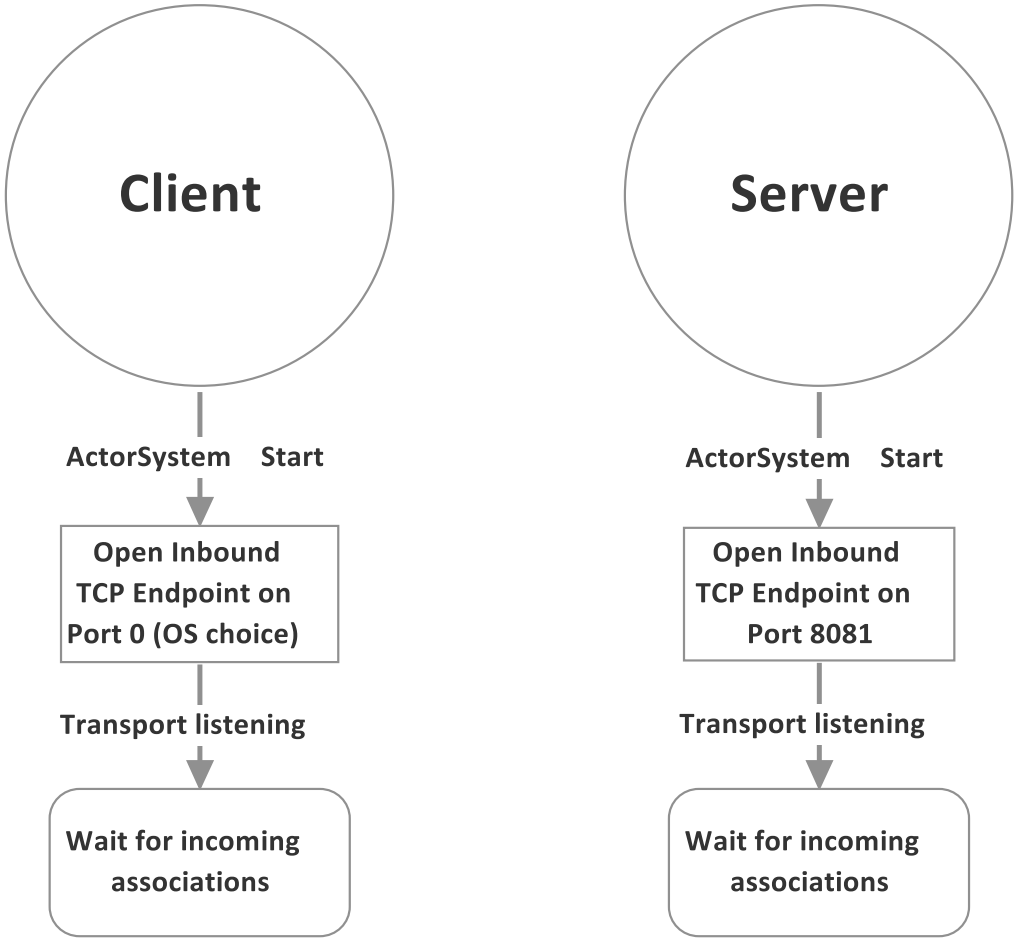
How to Enable Akka.Remote



Transports, Endpoints, and Associations



Initial State



The Magic of Port Zero

- Tells the OS to provide a random, but free port
- Allows us to generate unique addresses for each client instance
- Many clients, same

Local Address

All parts form an " ActorPath "

Protocol

akka://MySystem /user/ myActor1 /myActor2

ActorSystem

Path

Remote Address

All parts form an " ActorPath "

Protocol

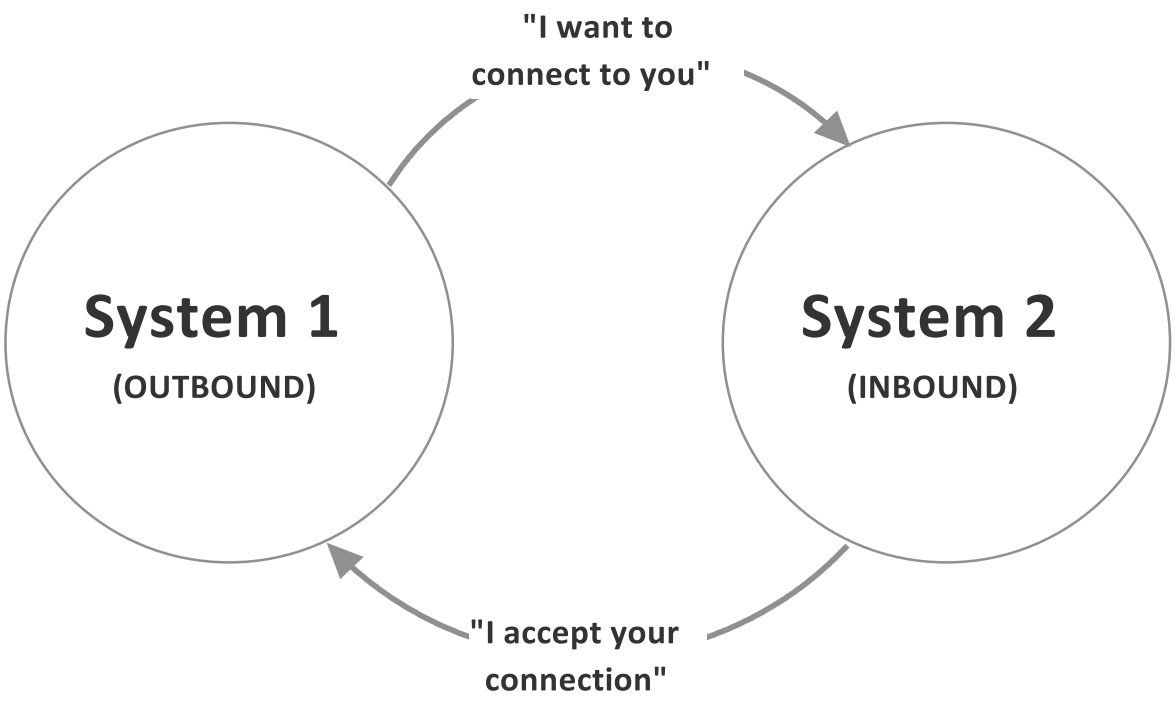
Address

akka.tcp://MySystem@localhost:9001/user/actorName1

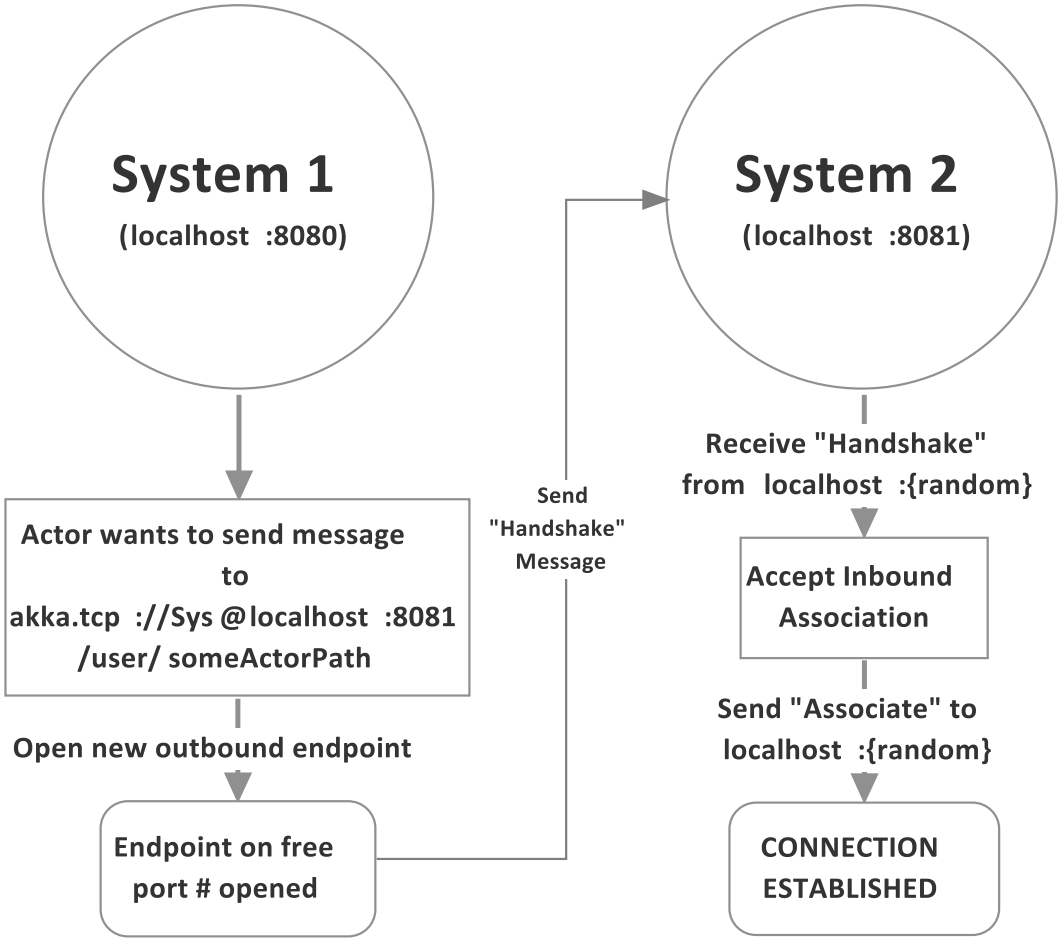
ActorSystem

Path

Inbound vs. Outbound Associations

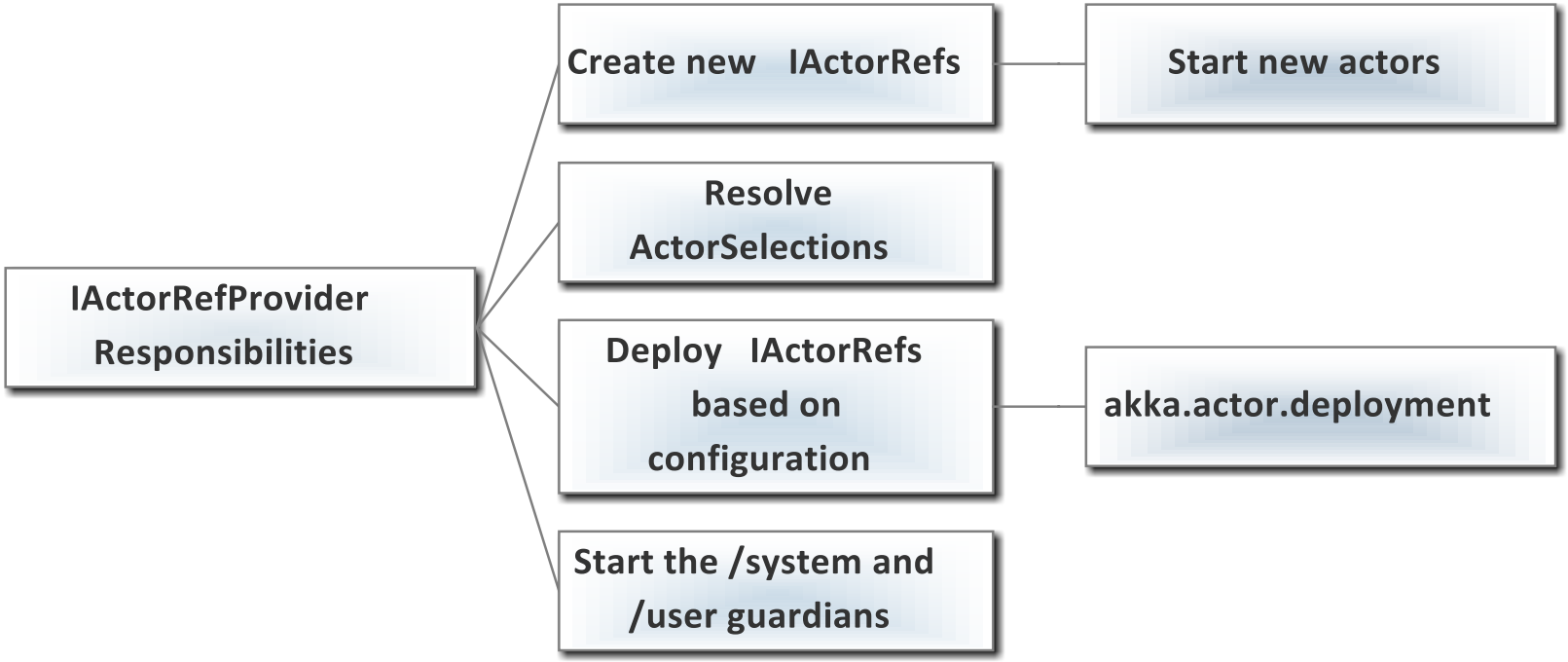


Associating

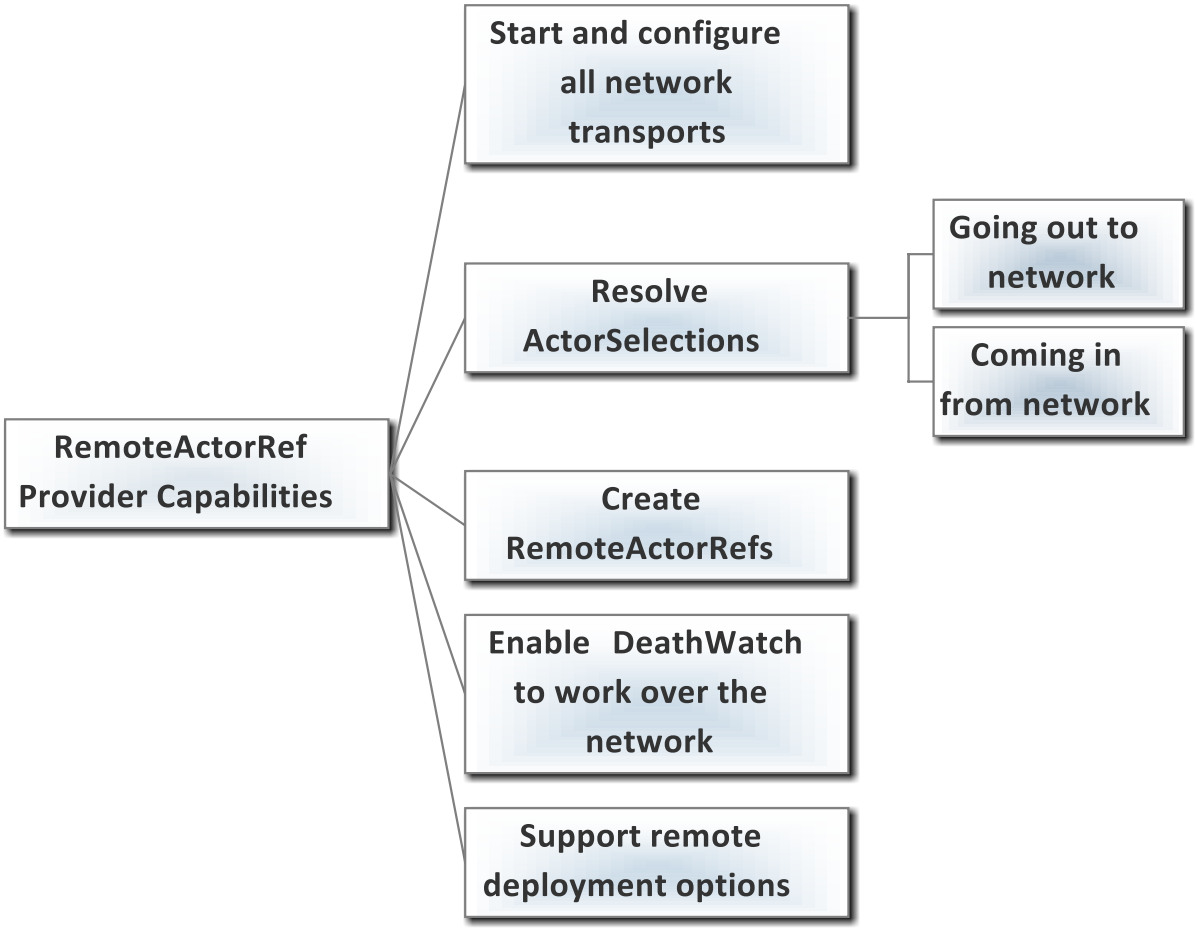


What makes this all possible?

IActorRefProviders!



RemoteActorRefProvider



Two Ways to Initiate a Remote Connection

- Remote ActorSelection
- Remote Deployment

Remote ActorSelection Sample

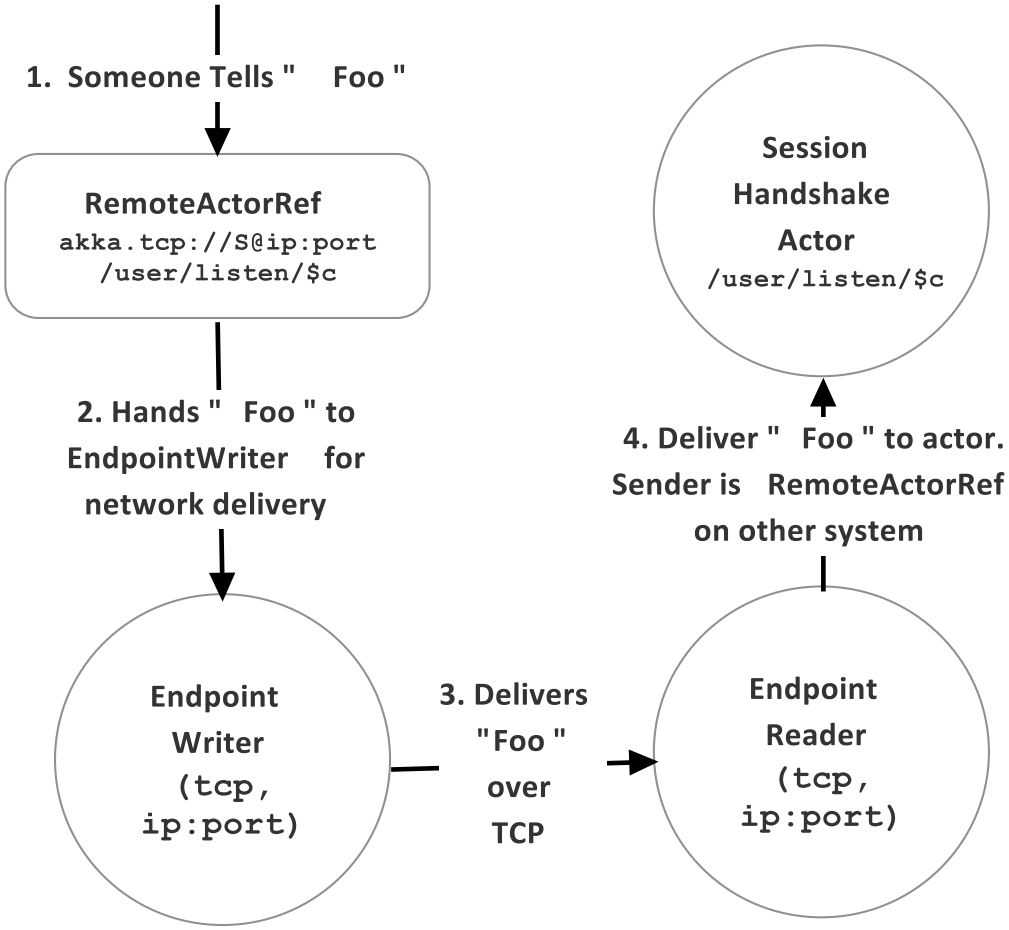
CODE SAMPLE

Replies from Remote Hosts: RemoteActorRef

CODE SAMPLE

RemoteActorRef and Location Transparency

How RemoteActorRef Actually Works



Remote Actor Deployment

"Deploying" an Actor

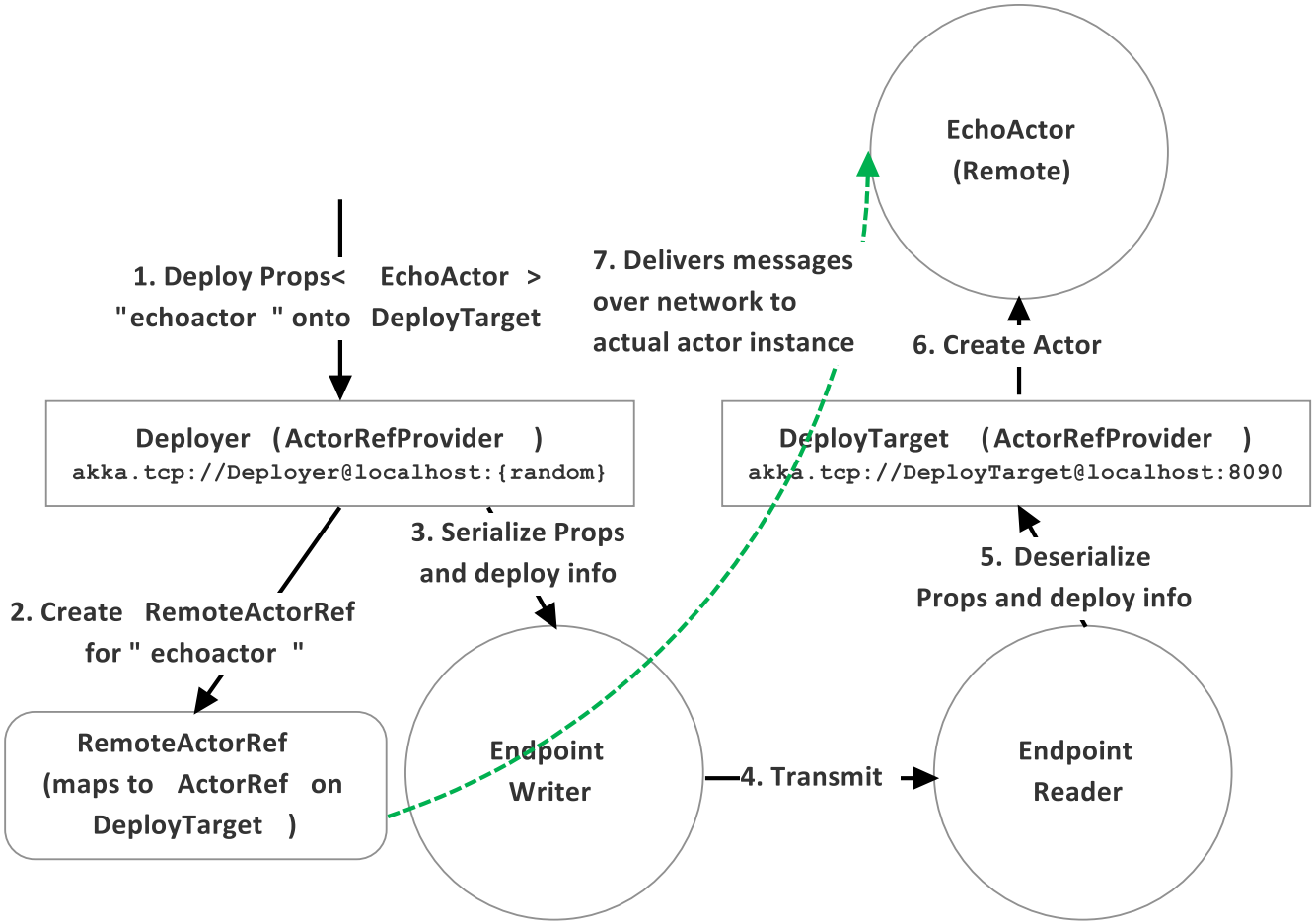
- Creating an actor instance with configured properties
- akka.actor.deployment
- Getting an IActorRef to actor

Location of Actor on Network is a Deployment Detail

Remote Actor Deployment

CODE SAMPLE

How Remote Deployment Actually Works

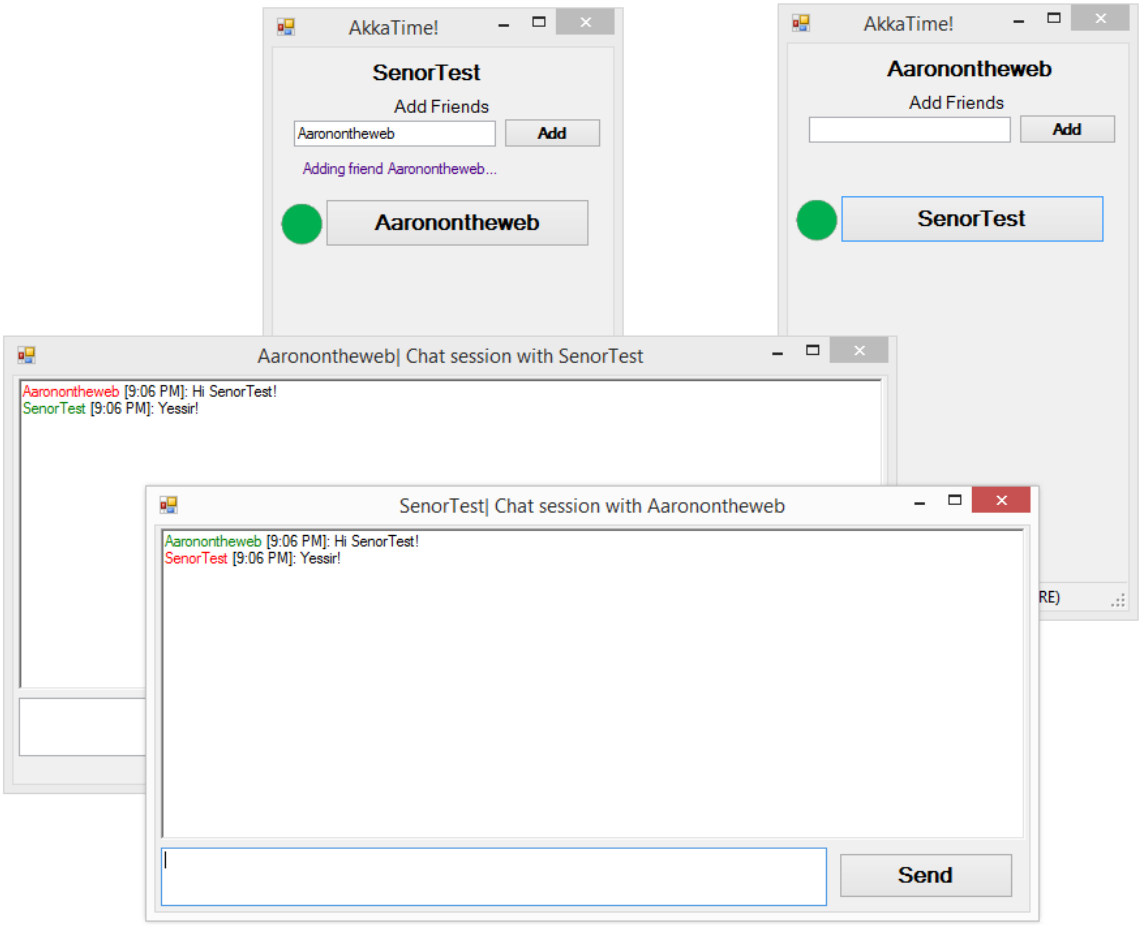


Important Things to Know about Remote Deployment

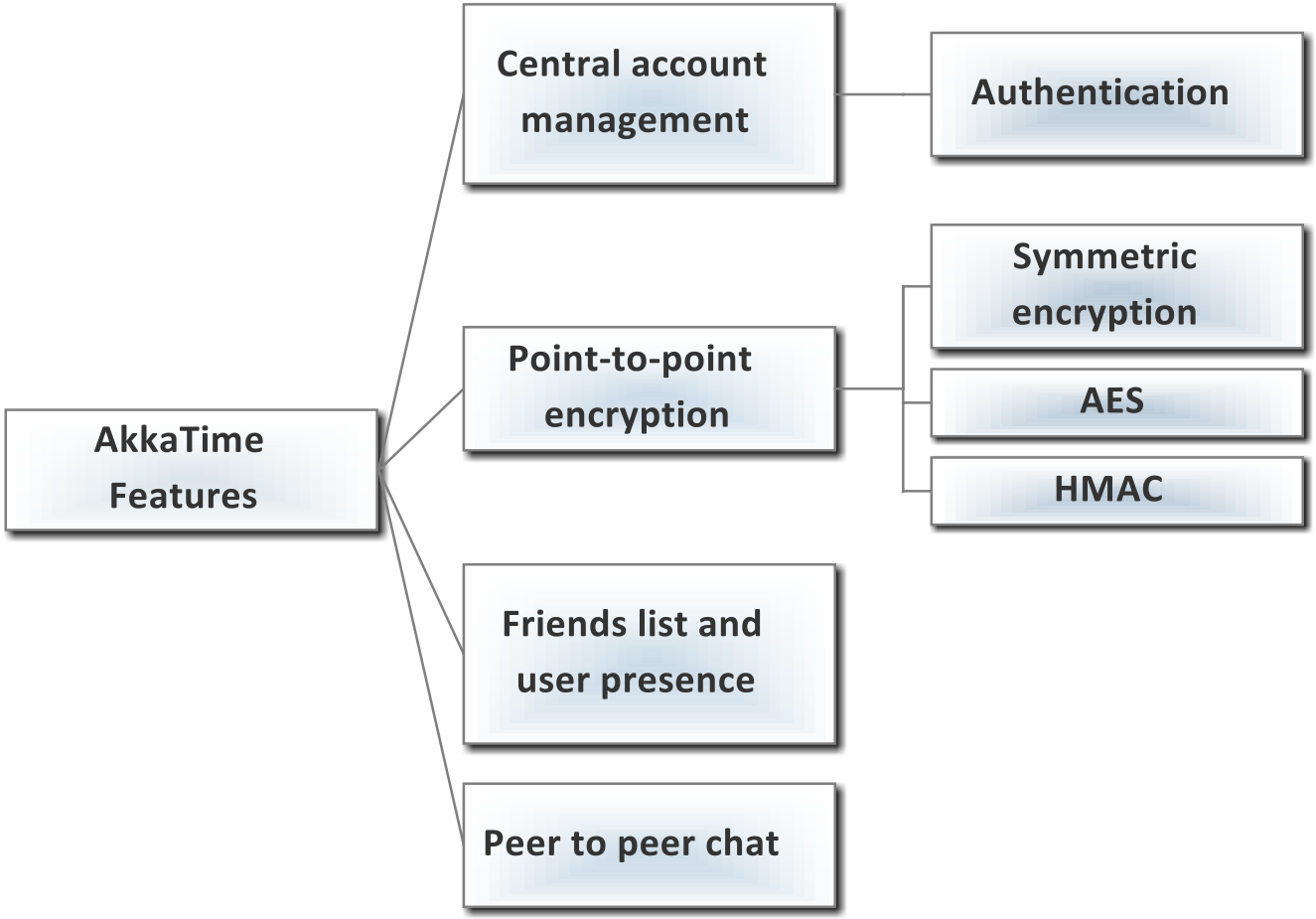
Remote Actor Deployments

- Remote actor names determined by deploying system
- Actor and message types must be present on both deployer and deployee
- All constructor arguments for remote-deployed actors must be serializable

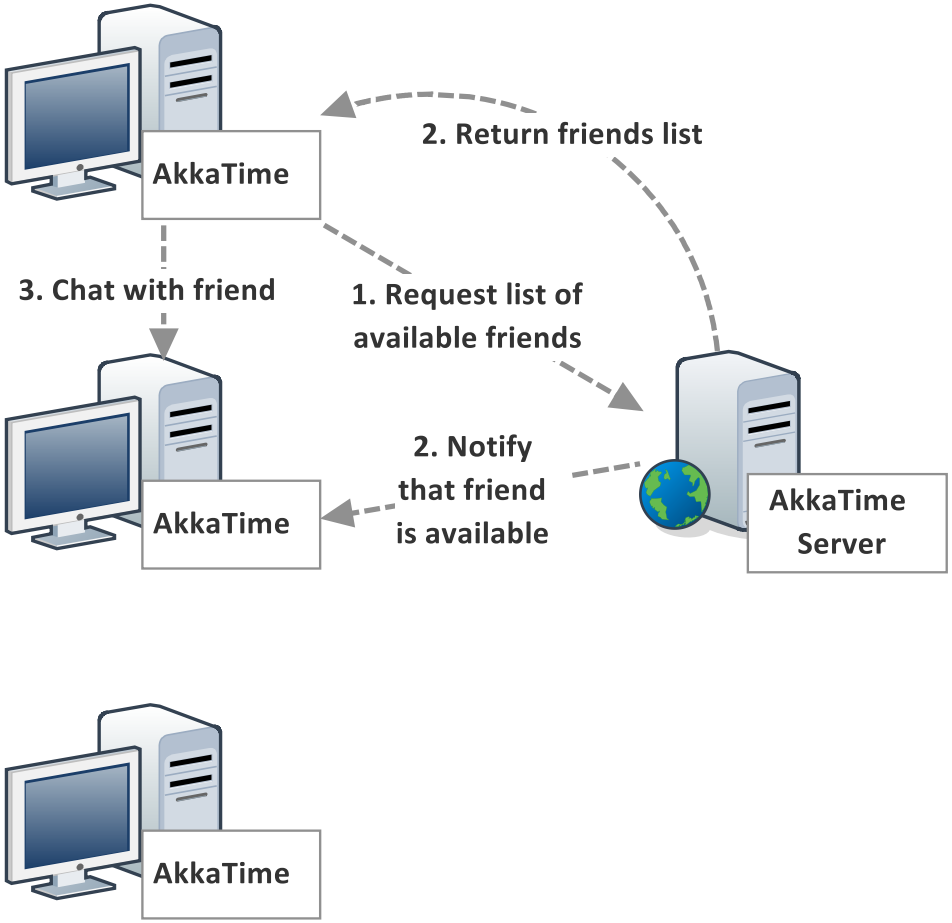
Code Sample: AkkaTime



AkkaTime Features

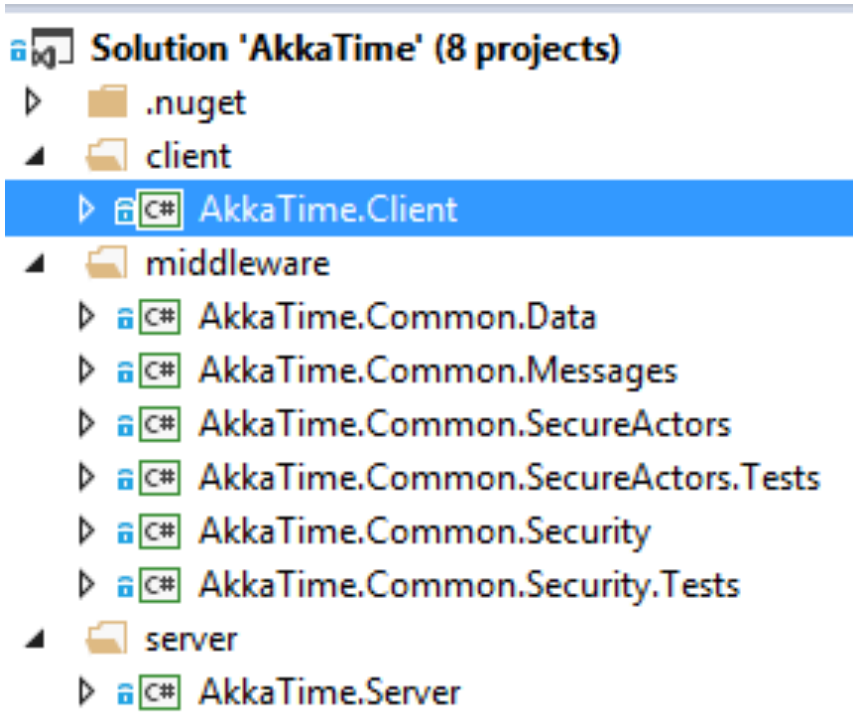


AkkaTime Architecture



AkkaTime Solution

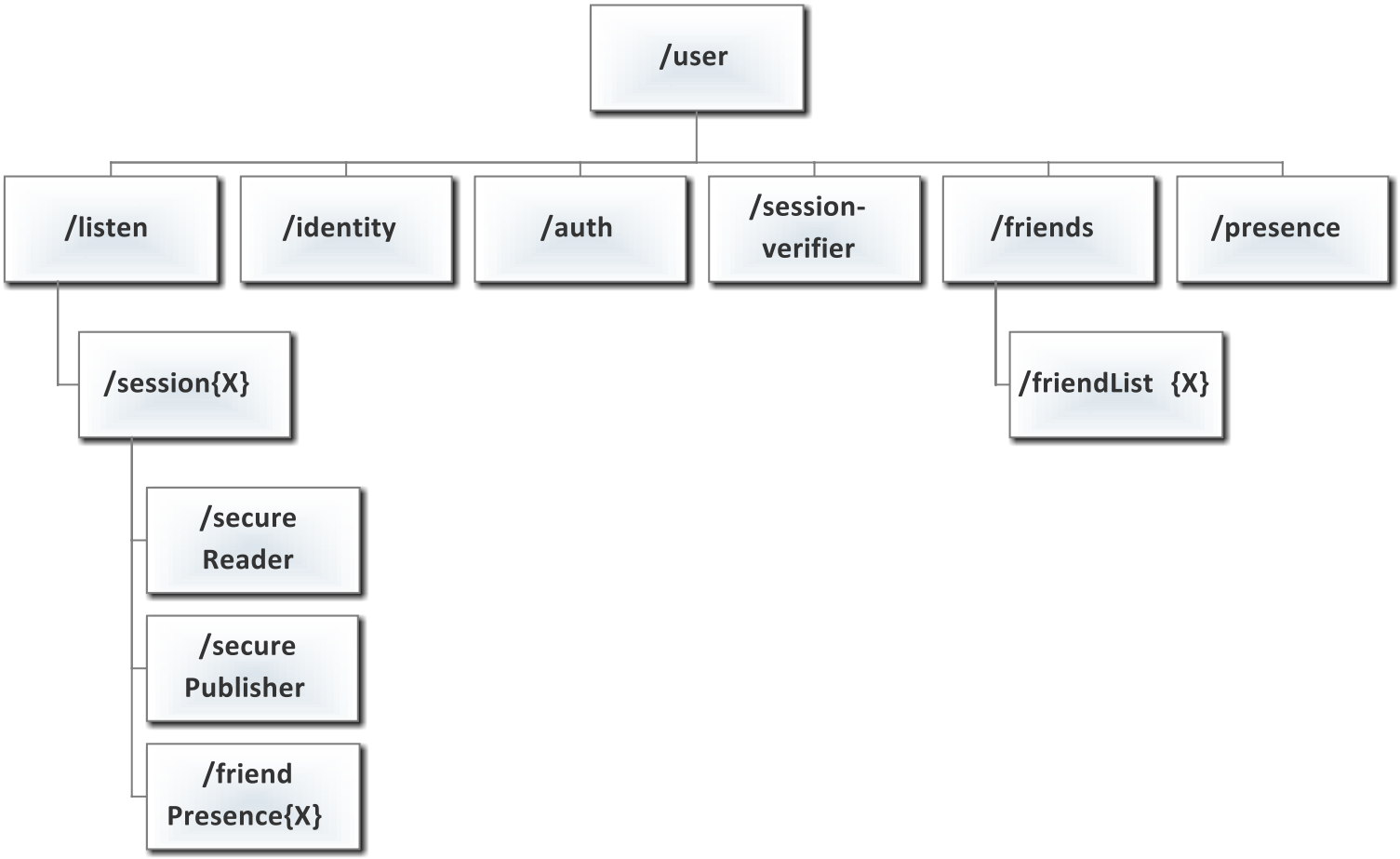
- AkkaTime.Client
- AkkaTime.Server
- AkkaTime.Common



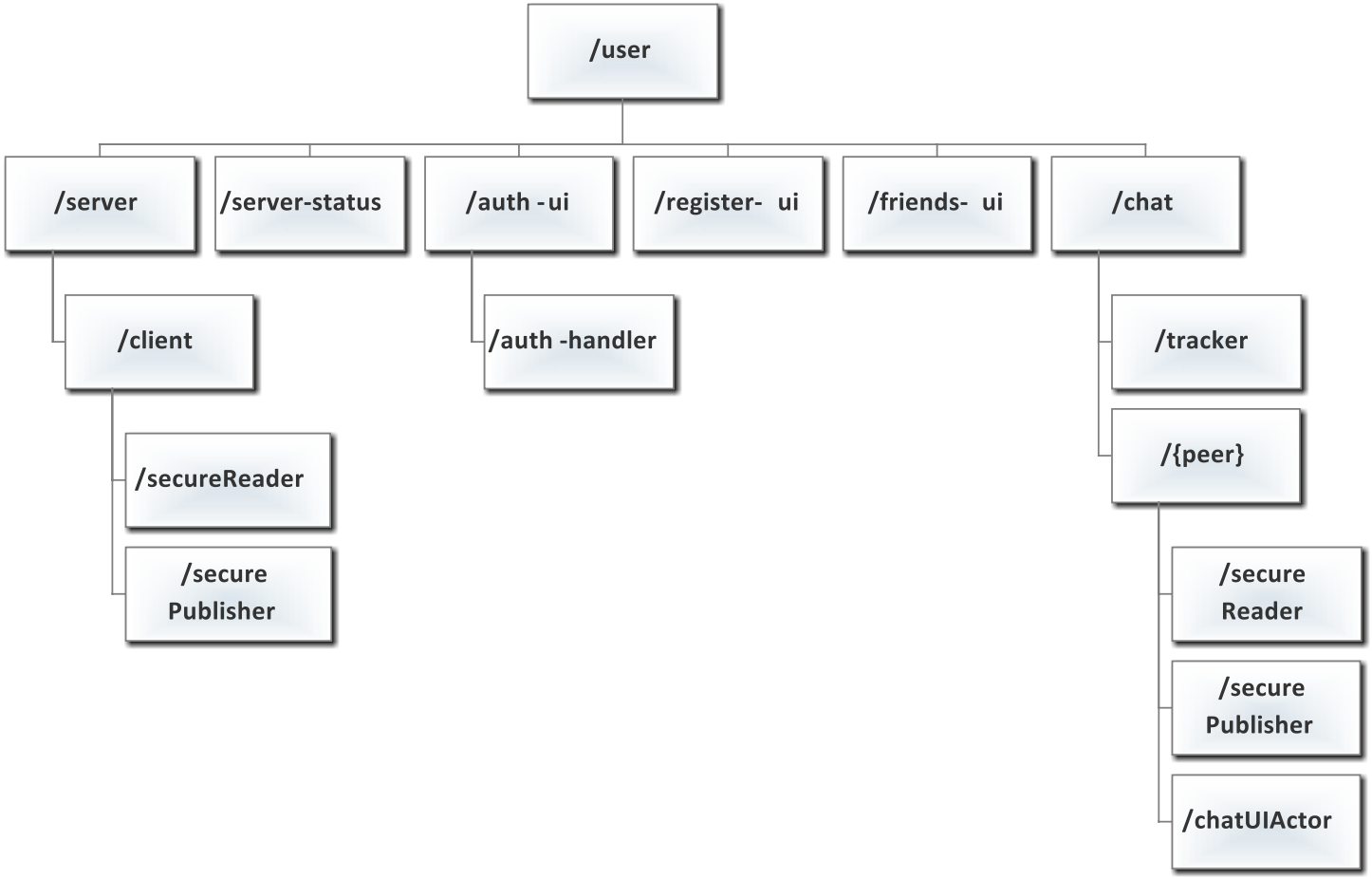
Reused Actor & Message Types

CODE SAMPLE

AkkaTime.Server Actor Hierarchy



AkkaTime.Client Actor Hierarchy

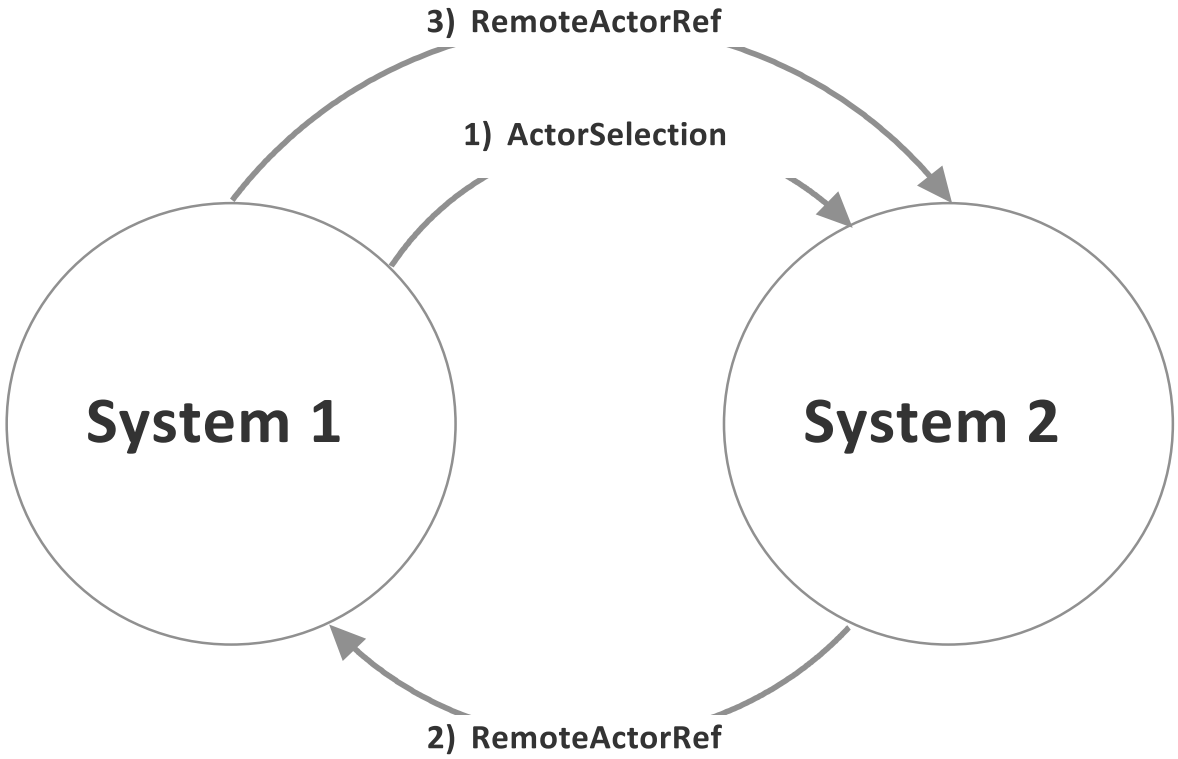


Remote Actor Design Patterns & Best Practices

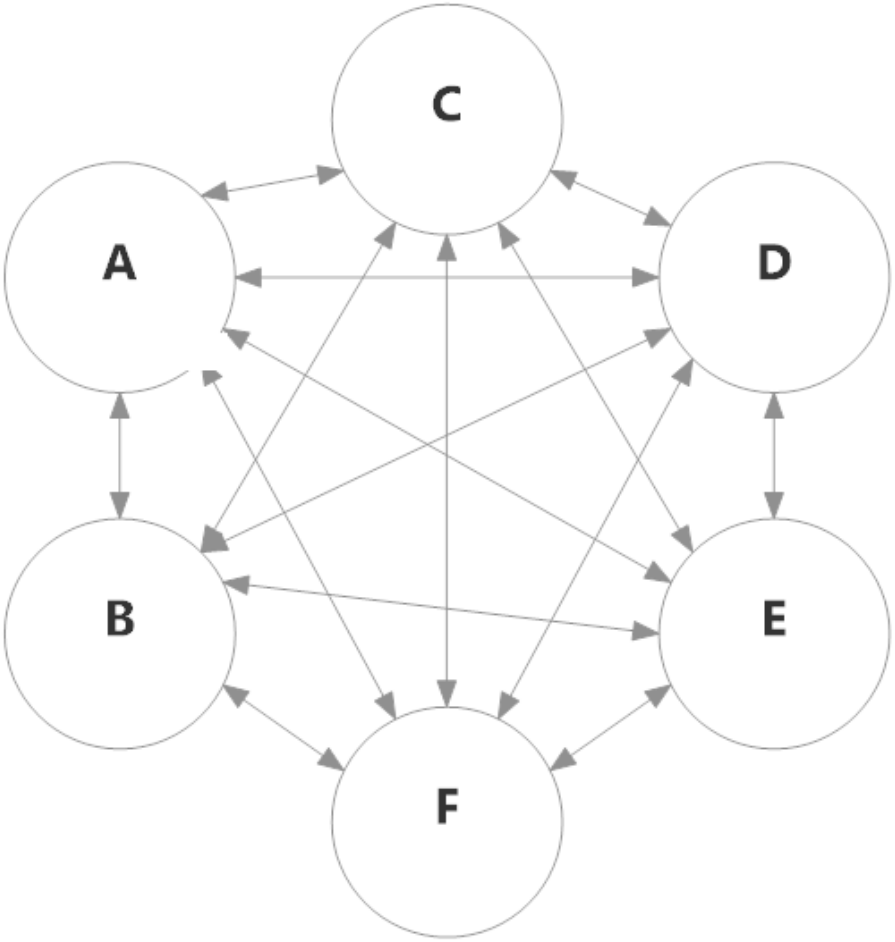
Designing Remote Actor Hierarchies

- Lookup-based hierarchies
  - Actors on remote processes already exist
- Deployment-based hierarchies
  - One process actively deploys actors onto other

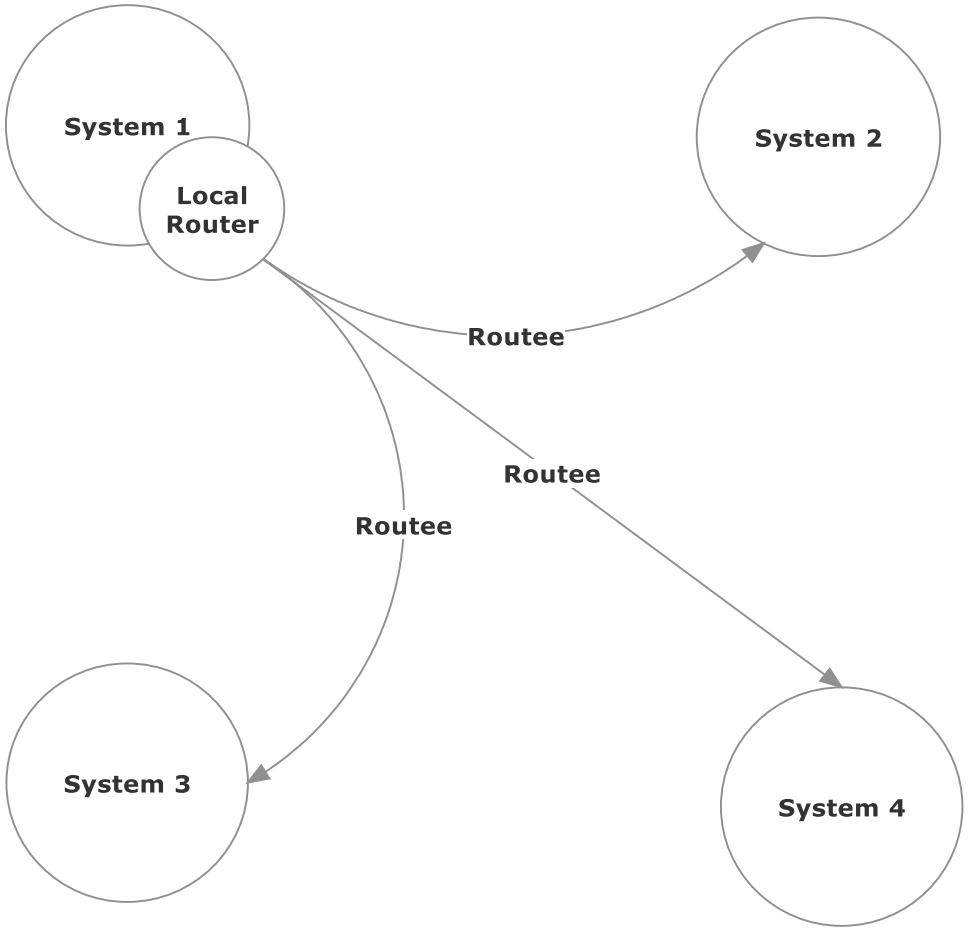
Remote Lookup



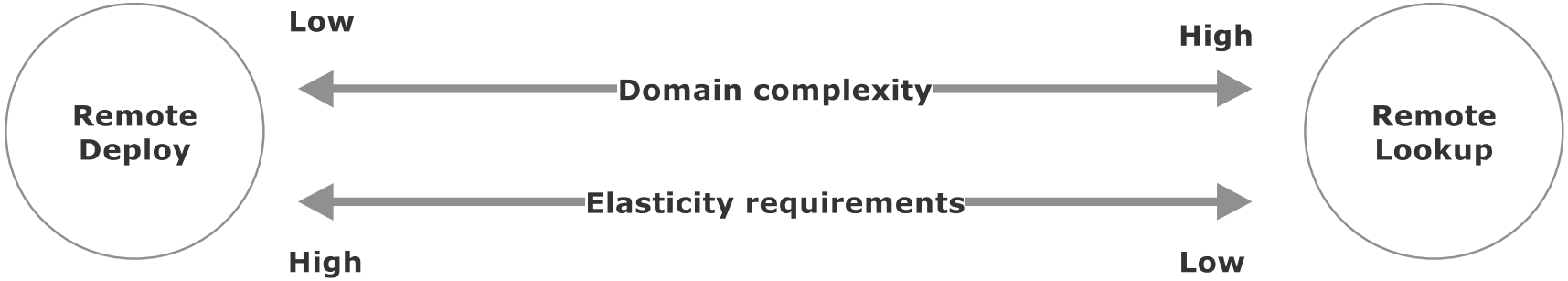
Remote Lookup



Remote Deploy



Choosing


Remote Deploy
Low
High
Domain complexity
Elasticity requirements
High
Low
Remote Lookup

Serialization

Sharing Message and Actor Types

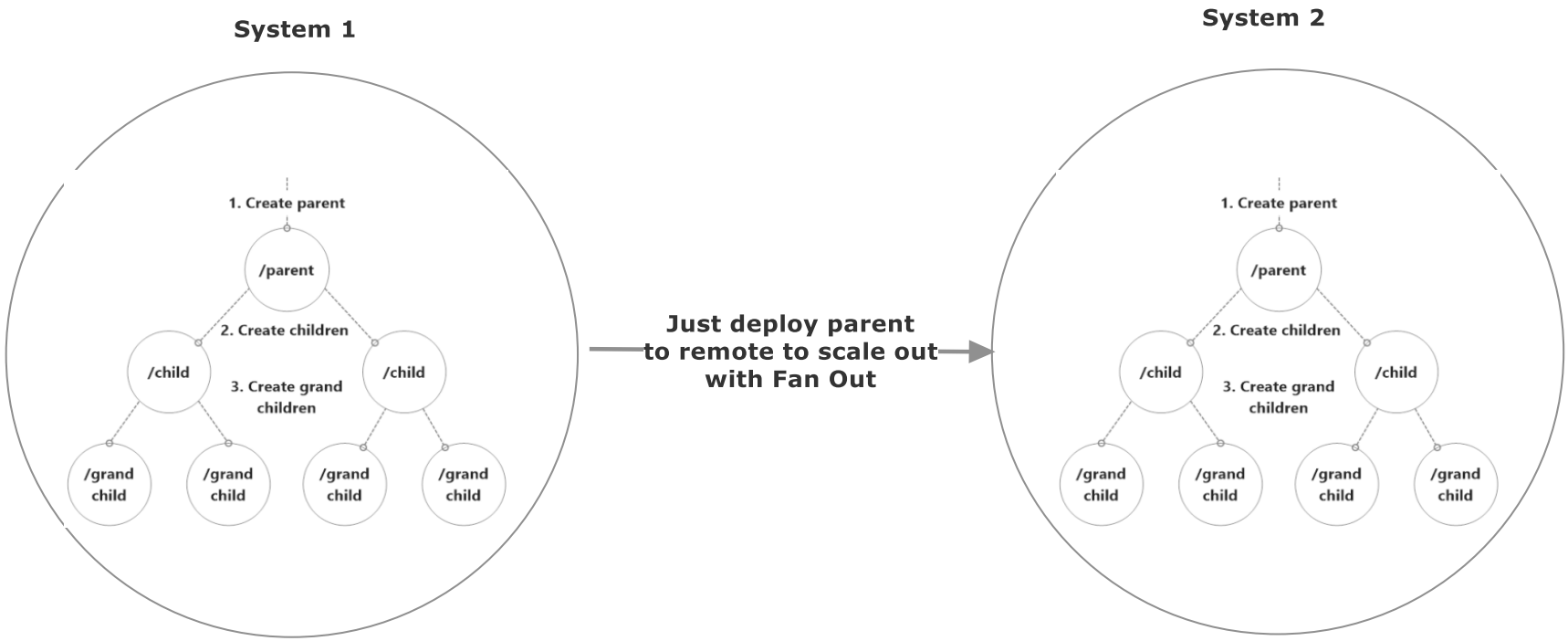
```
kaTime.Client.ActorPaths                    ServerSupervisor

defi

Addr
onne

oint

nnection to the server

ervisor = new ActorMetaData("server");

he server

oint = new ActorMetaData("client", ServerSupervisor);

 publishing encrypted / decrypted messages to the server

lisher = new ActorMetaData("publisher", ClientEndpoint);
```

Solution Explorer

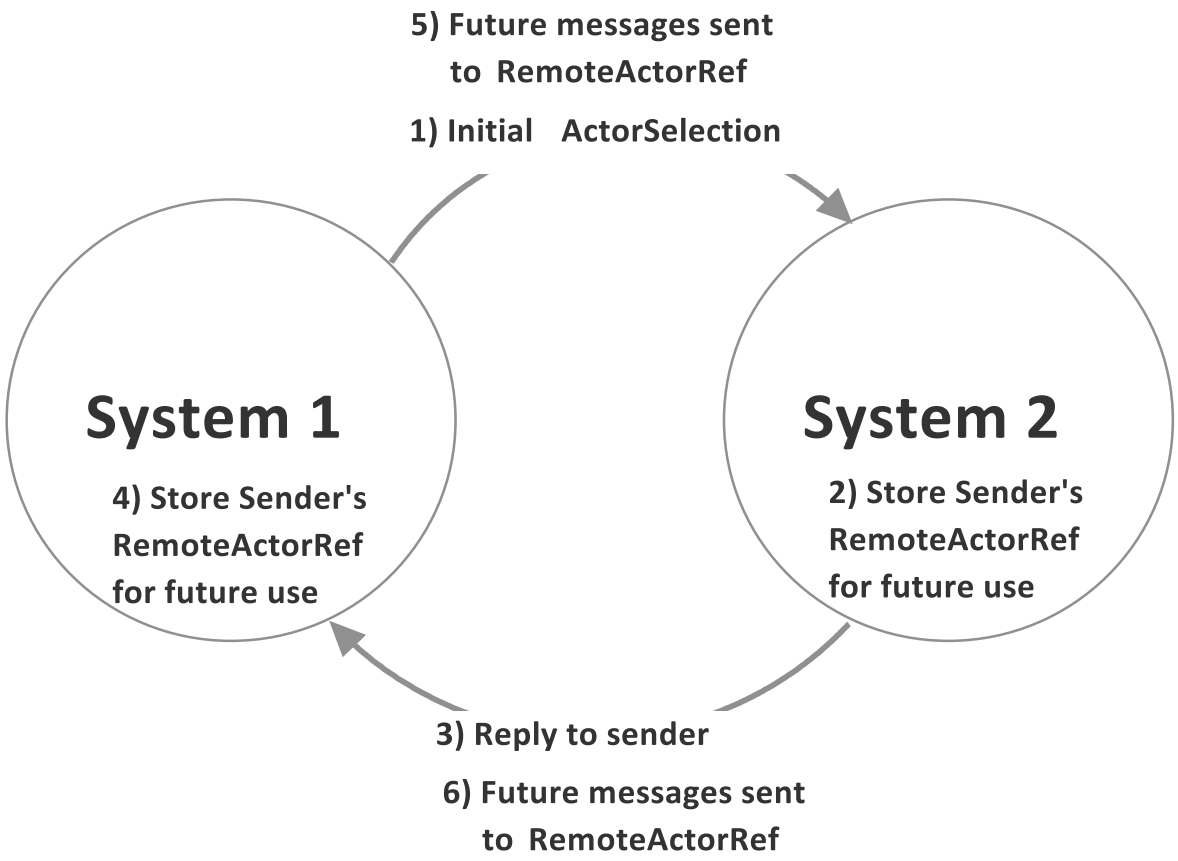
Search Solution Explorer (Ctrl+;)

- middleware
  - AkkaTime.Common.Data
  - AkkaTime.Common.Messages
    - Properties
    - References
    - Activity
    - Authentication
    - Chat
    - Handshake
    - Identity
    - Secure
    - Social
    - ActorMetaData.cs
    - packages.config
  - AkkaTime.Common.SecureActors
    - Properties
    - References
    - AutoAuthenticationApprover.cs
    - ISessionManager.cs
    - ISessionStarter.cs
    - LocalOnlyForwarder.cs
    - packages.config
    - SecureActorRefExtensions.cs
    - SecurePublisher.cs
    - SessionHandshakeActor.cs
    - SessionManagerBase.cs
    - SessionManagerWrapper.cs
    - TwoWayEncryptionActor.cs
  - AkkaTime.Common.SecureActors.Tests
  - AkkaTime.Common.Security
  - AkkaTime.Common.Security.Tests

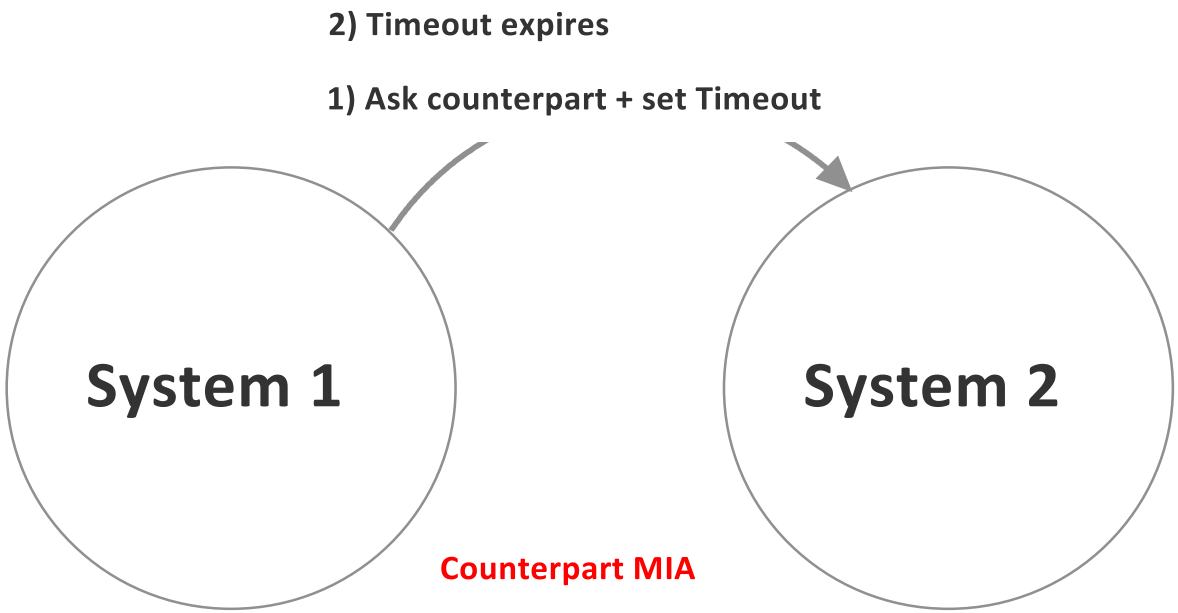
Remote Fan Out



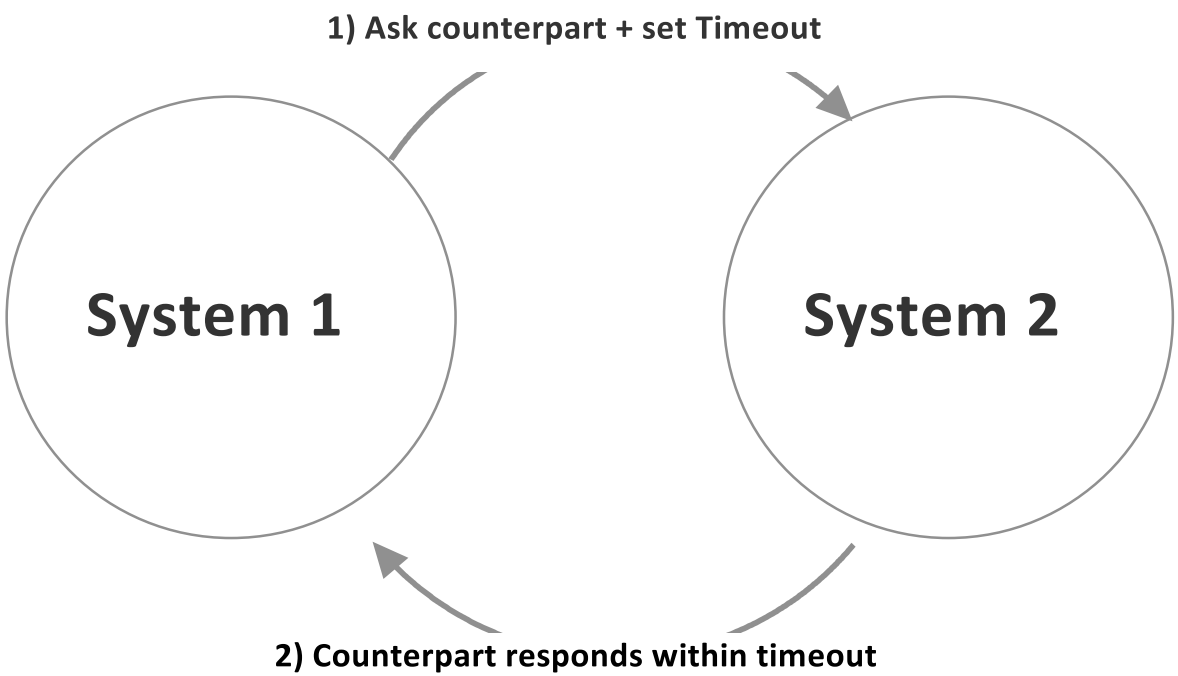
Reply to Sender



Verify Counterparts (fail)

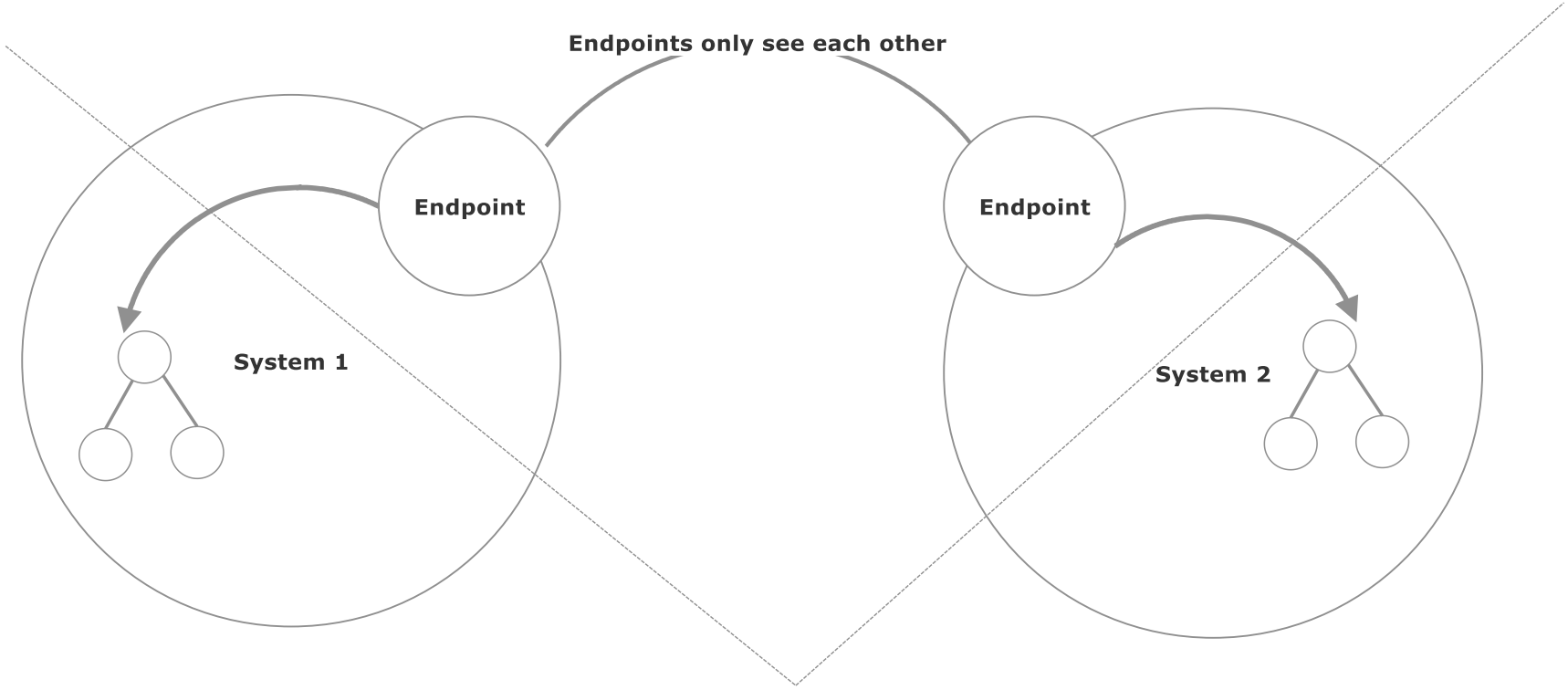


Verify Counterparts (success)



Counterpart present + listening

Limit Assumptions about Remote Systems



Akka.Remote Security

Coming Soon

- TLS
- Secure Cookie

What You Can Do Today

- VPNs
- Two-way Encryption
- Restrict to Local Senders

Two-way Encryption

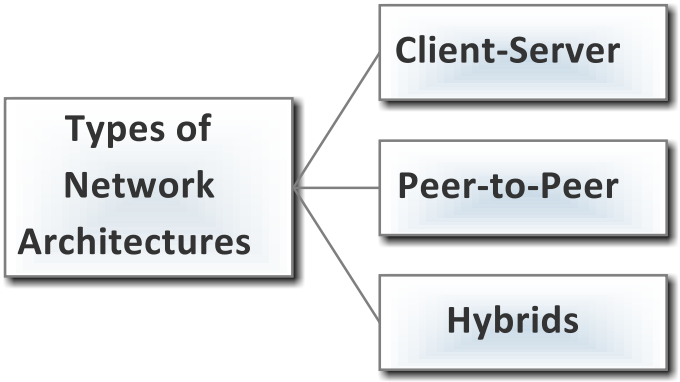
CODE SAMPLE

Restrict to Local Senders

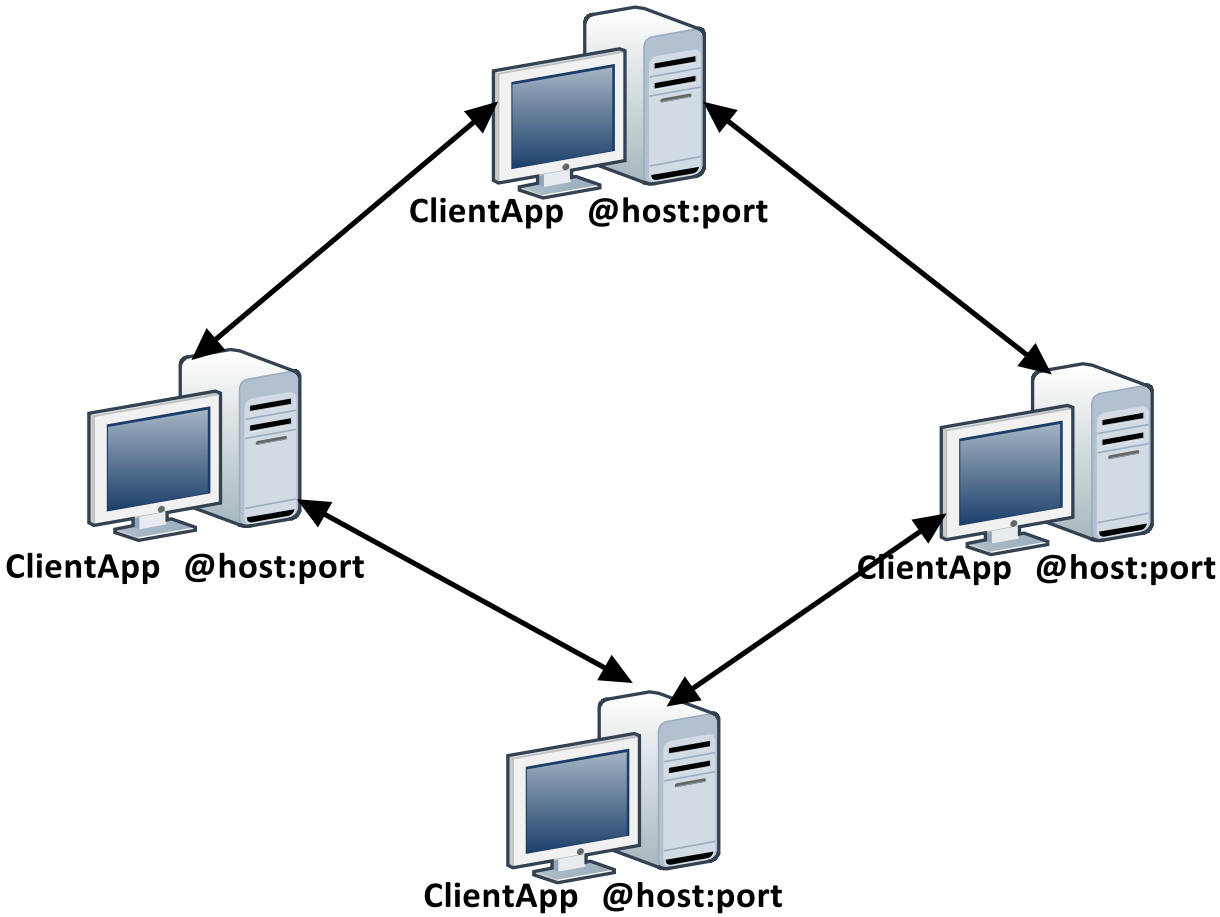
CODE SAMPLE

Akka.Remote Distributed Architectures

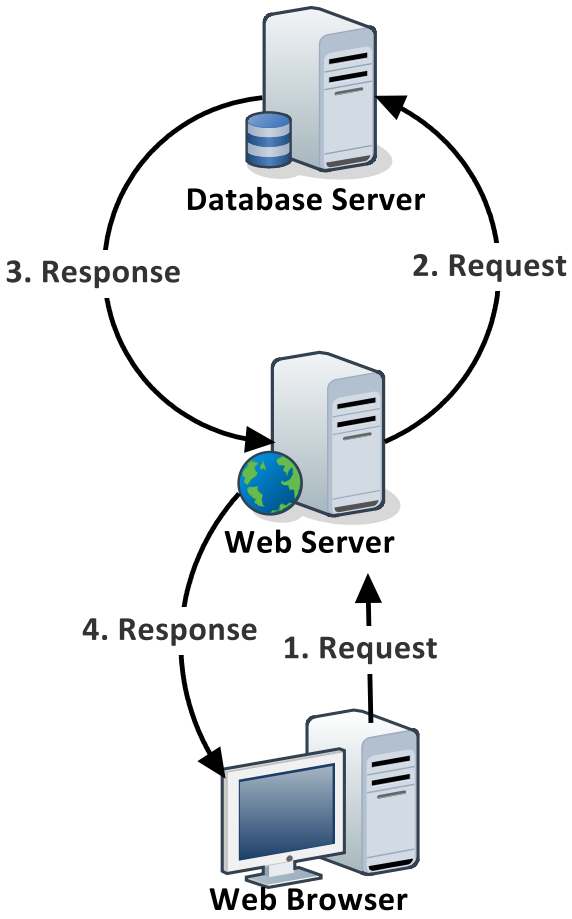
Types of Distributed Architectures



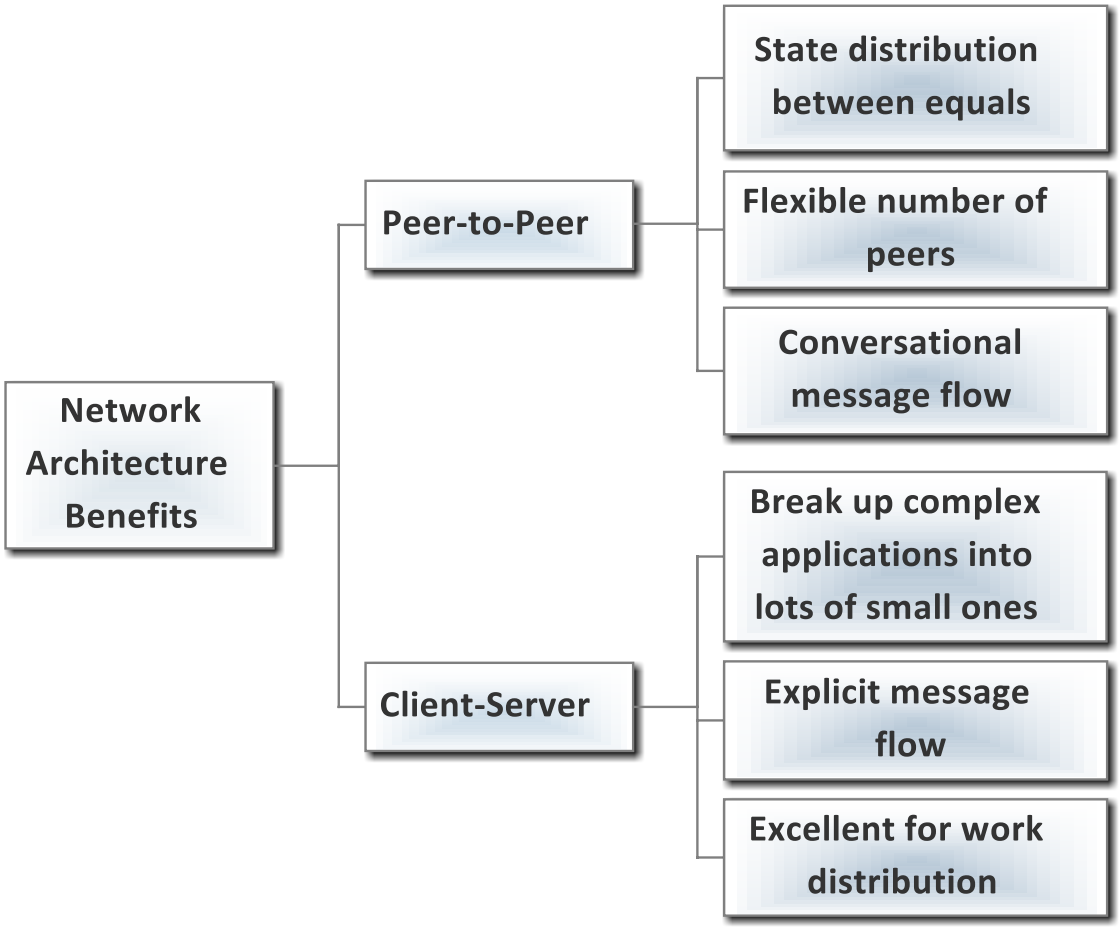
Peer-to-Peer


ClientApp @host:port
ClientApp @host:port
ClientApp @host:port
ClientApp @host:port

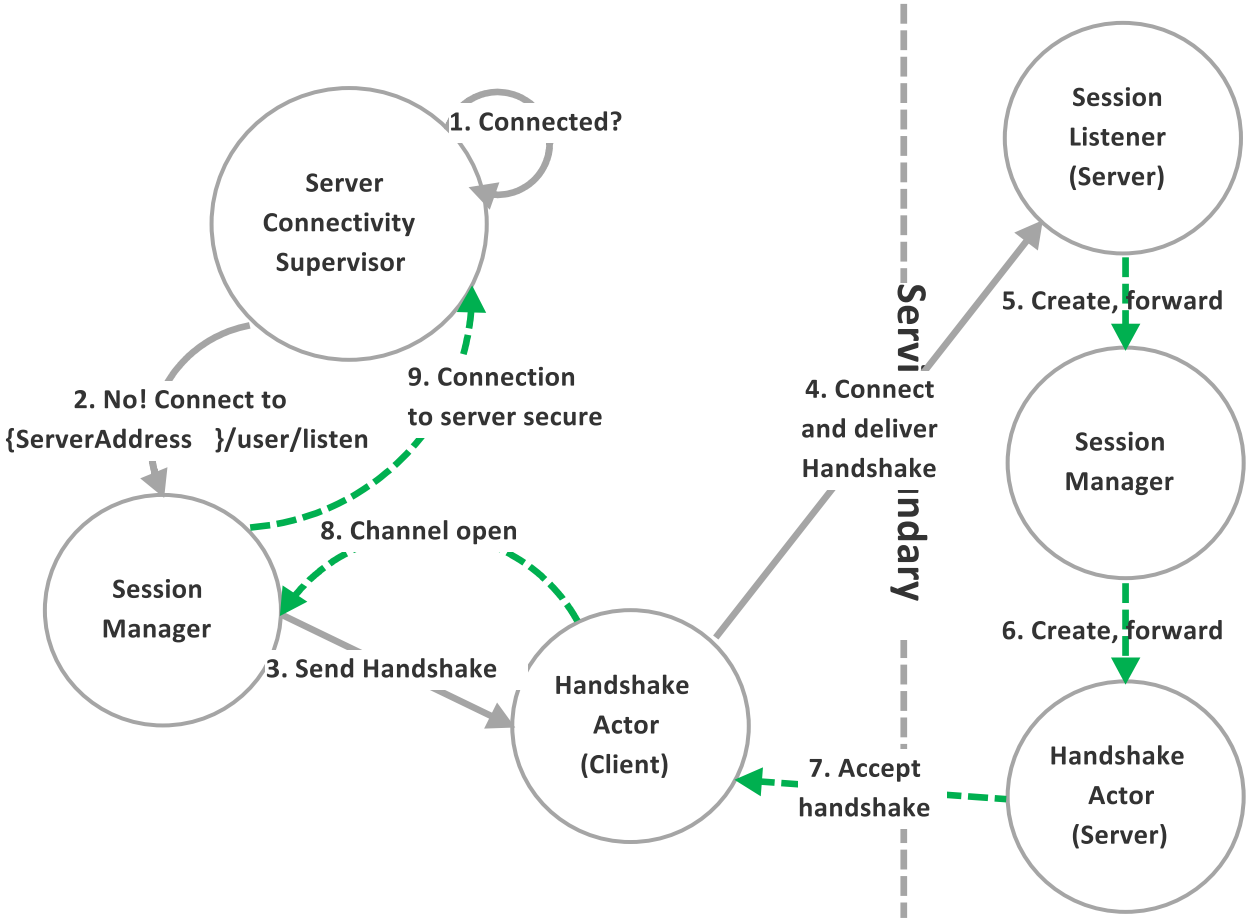
Client-Server



Architecture Tradeoffs



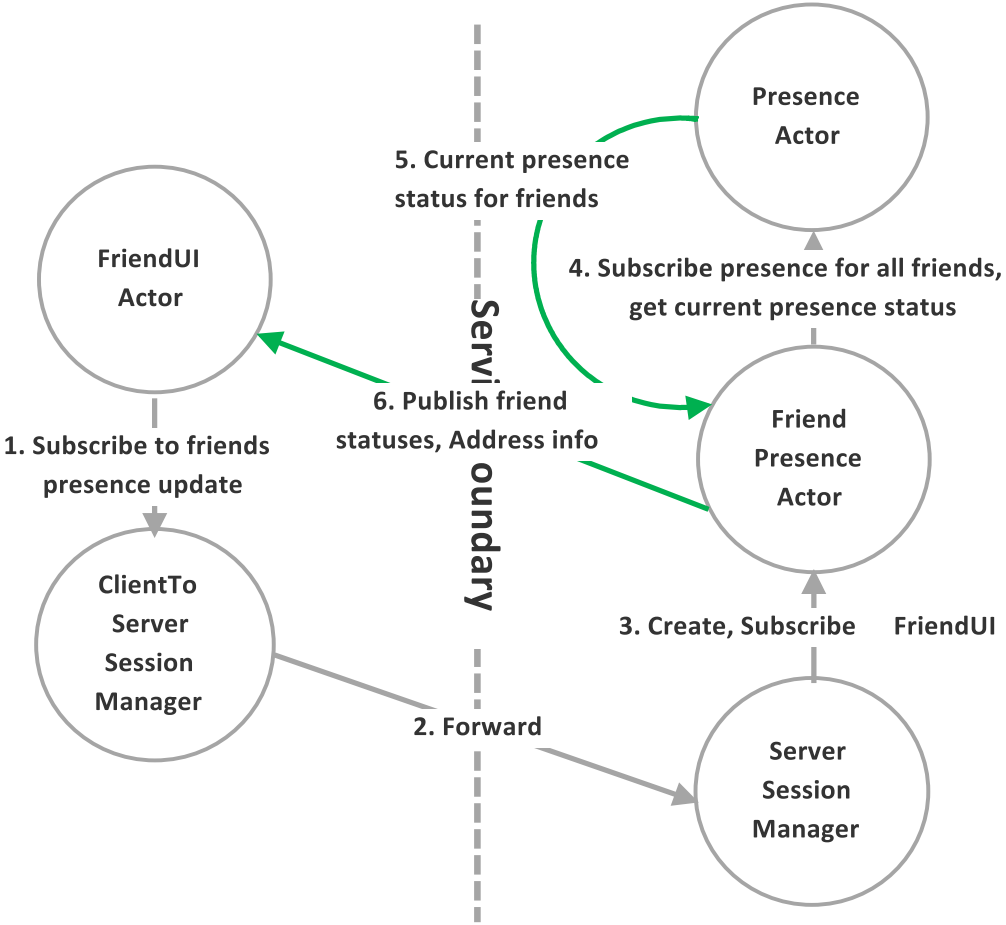
AkkaTime Client-Server



AkkaTime Client-Server

CODE SAMPLE

AkkaTime Peer-to-Peer



AkkaTime Peer-to-Peer

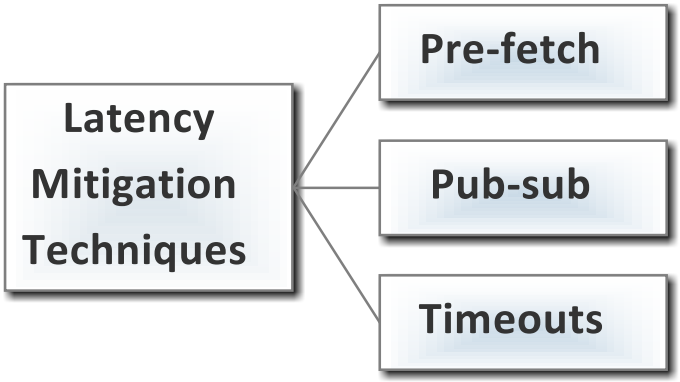
CODE SAMPLE

Realities of the Network

Networks are Imperfect

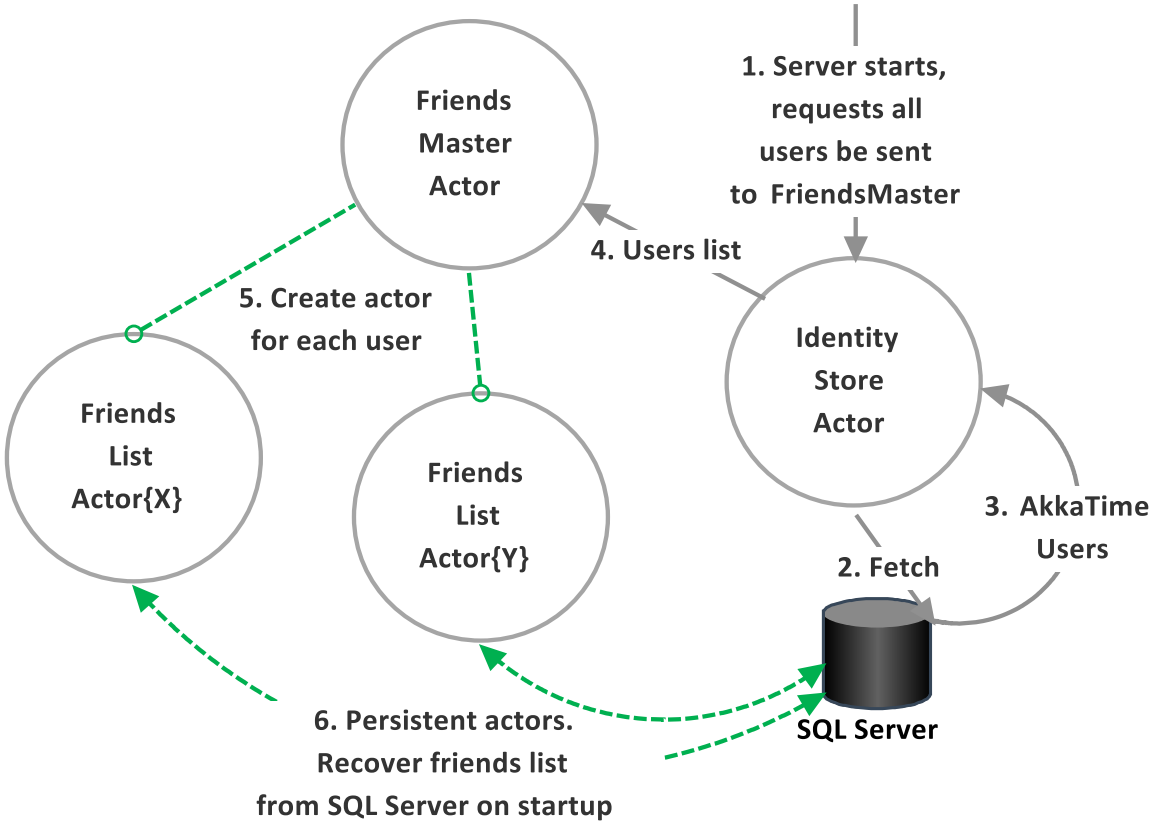
- Latency
- Failures
- Dead Processes

Latency Mitigation Techniques



Pre-fetch

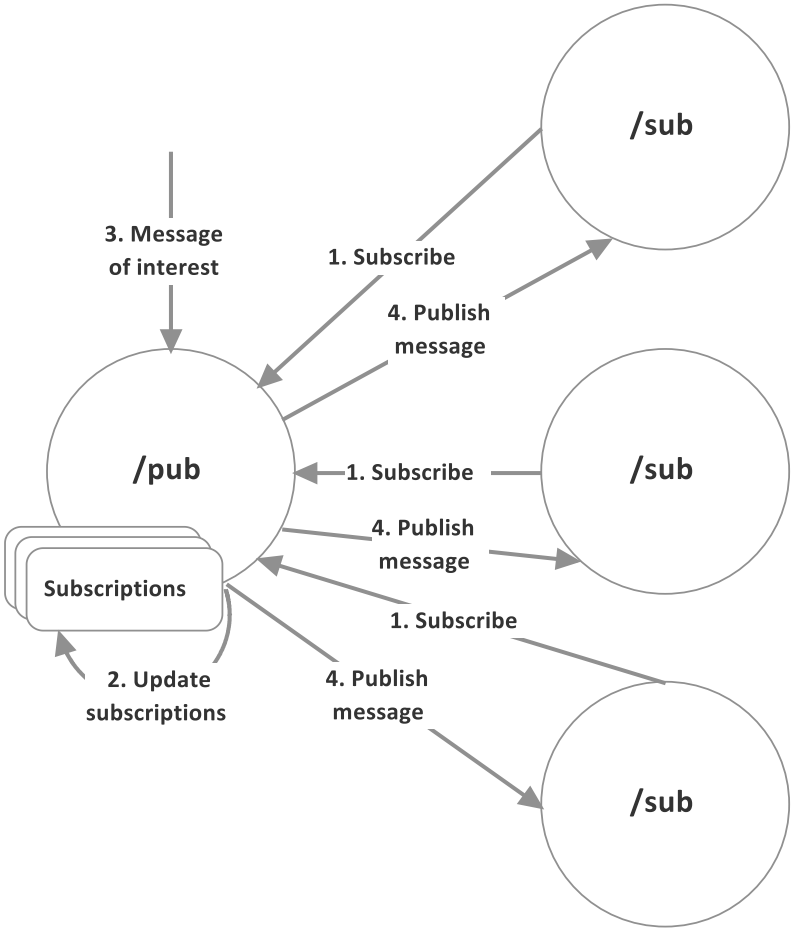
Fetch all friends data from SQL Server and pre-build actor hierarchy before any user ever connects.



Pre-fetch

CODE SAMPLE

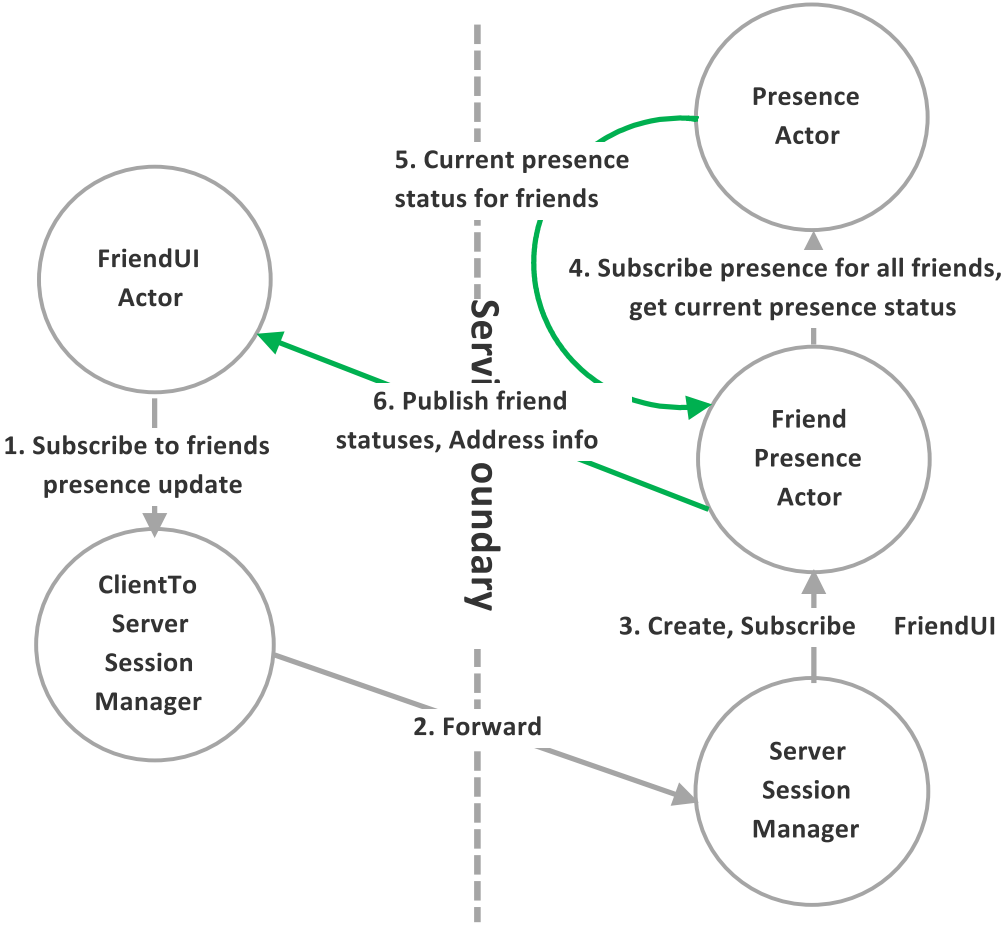
Pub-sub


/sub
3. Message
of interest
1. Subscribe
4. Publish
message
/pub
1. Subscribe
/sub
4. Publish
message
Subscriptions
1. Subscribe
2. Update
subscriptions
4. Publish
message
/sub

Pub-sub

CODE SAMPLE

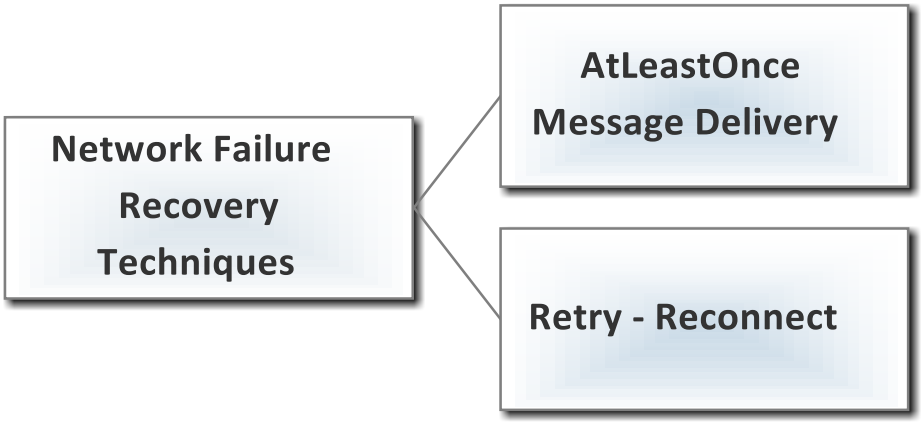
AkkaTime Network Pub-sub



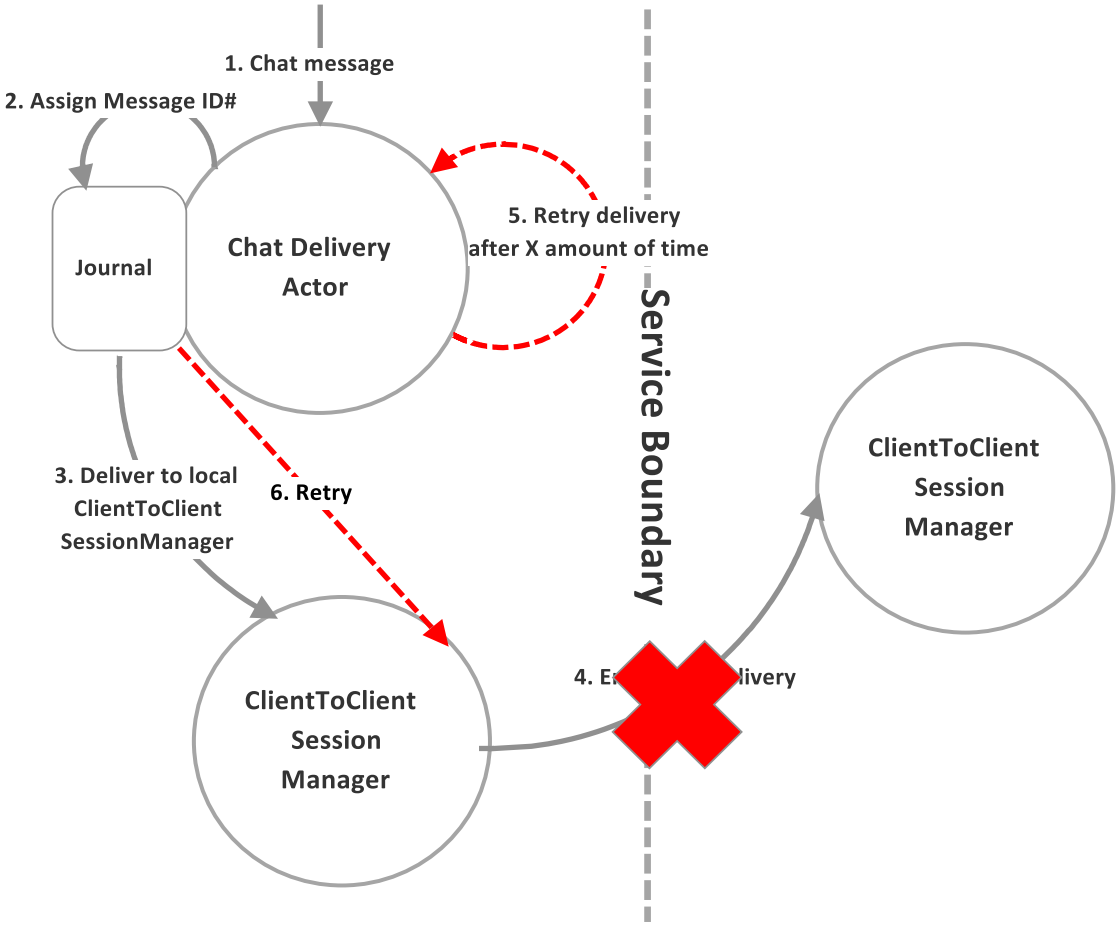
Timeout

CODE SAMPLE

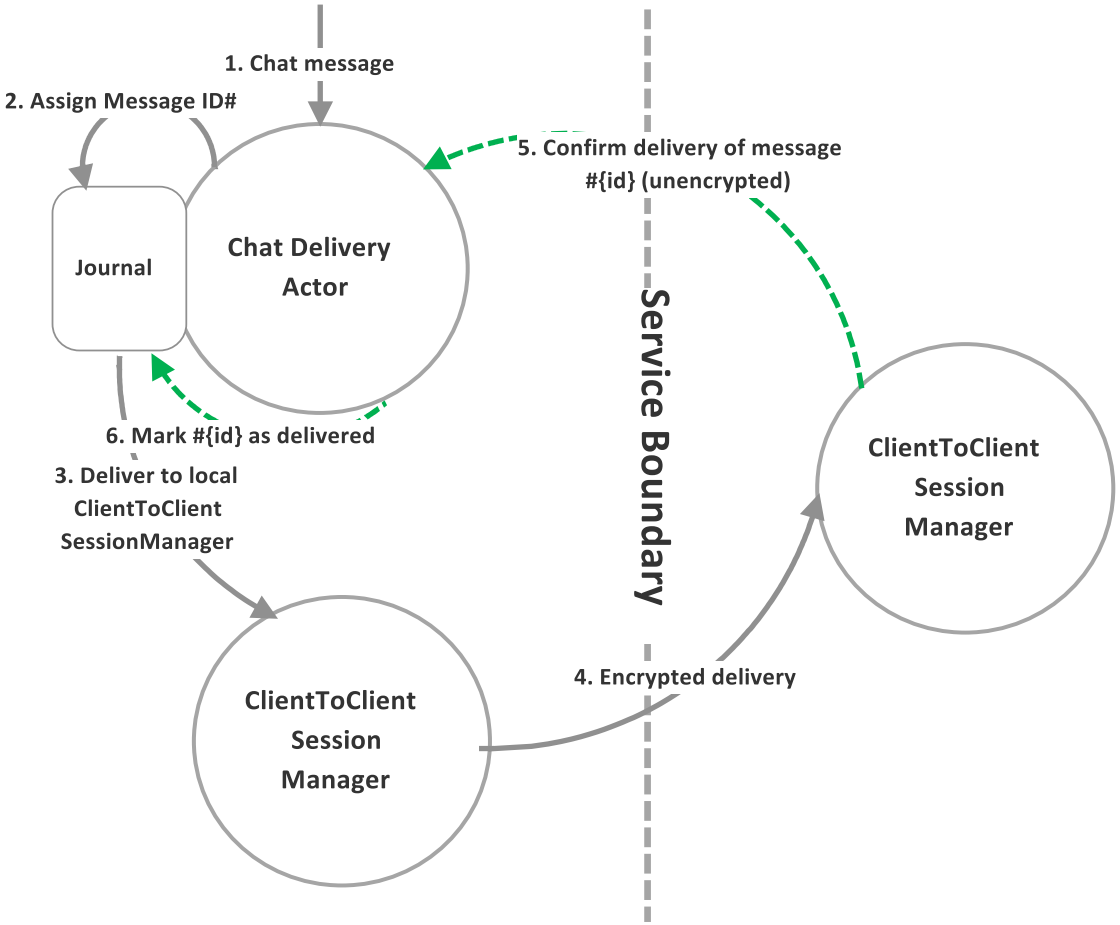
Network Failures



AtLeastOnceMessageDelivery



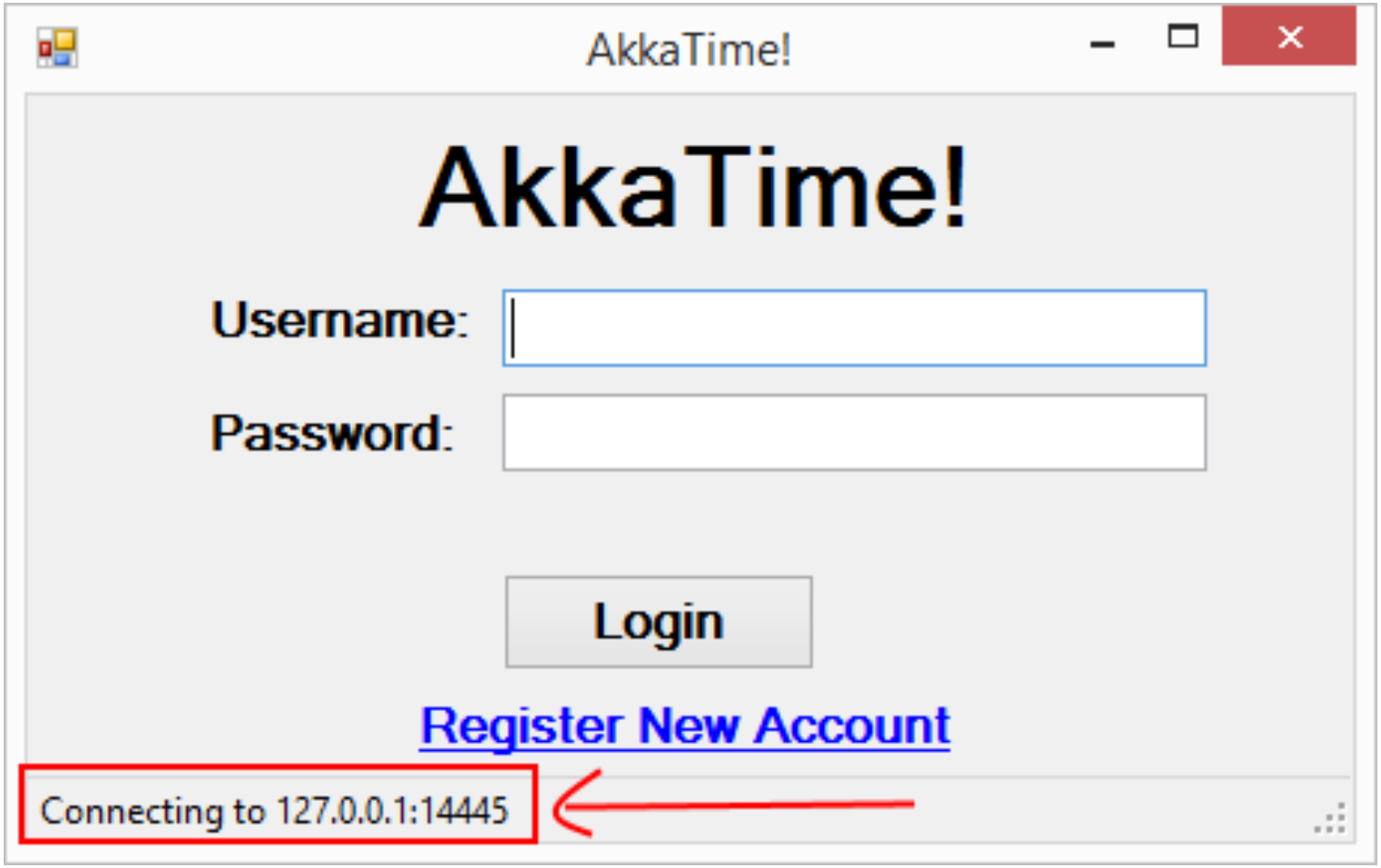
AtLeastOnceMessageDelivery



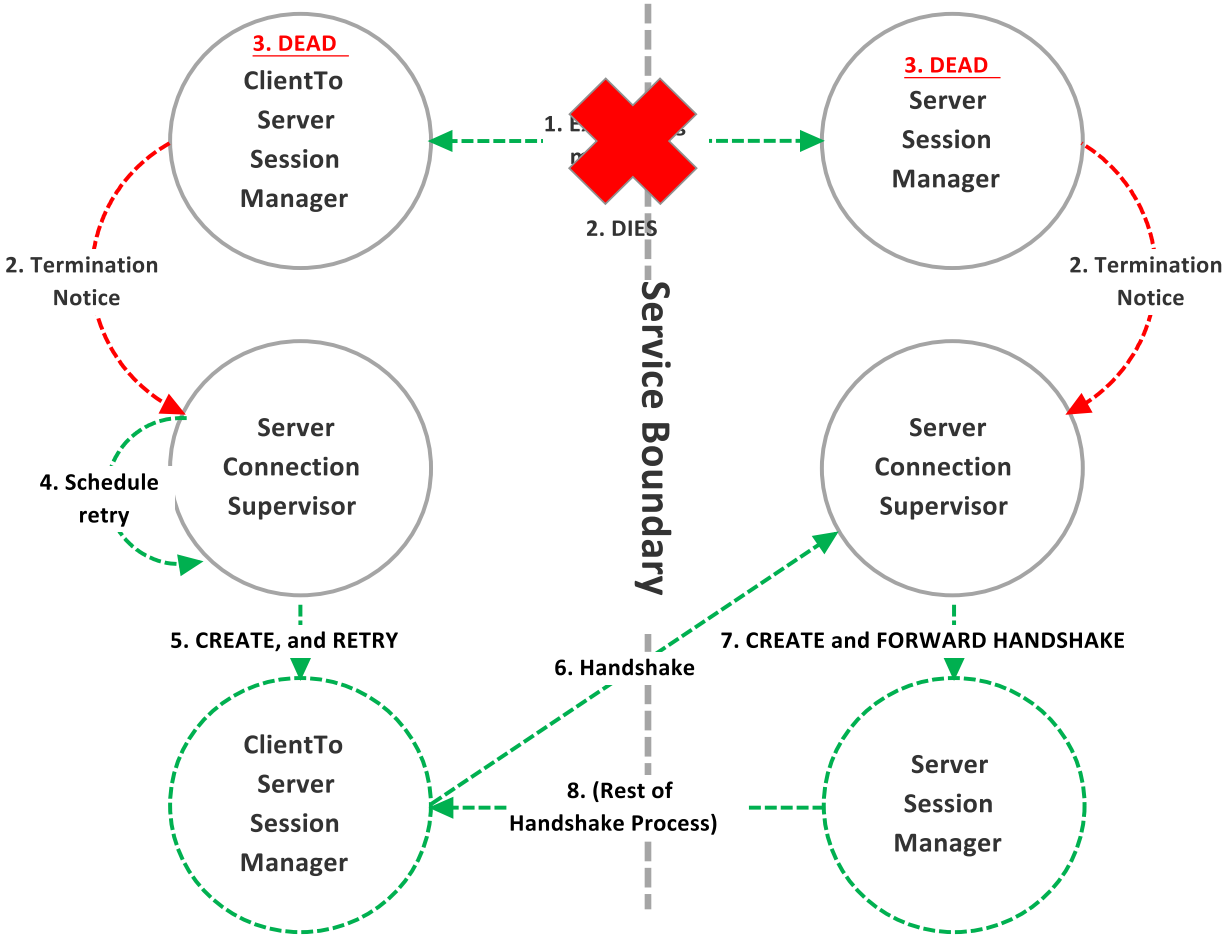
AtLeastOnceMessageDelivery

CODE SAMPLE

Retry / Reconnect



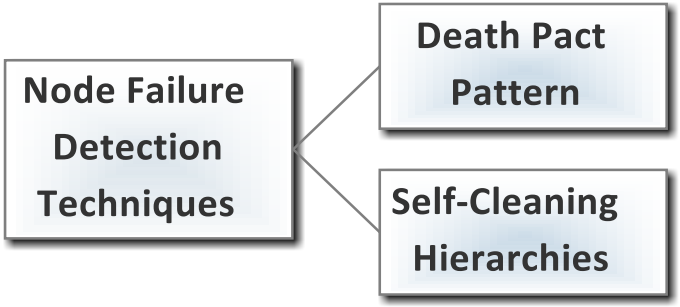
Retry / Reconnect



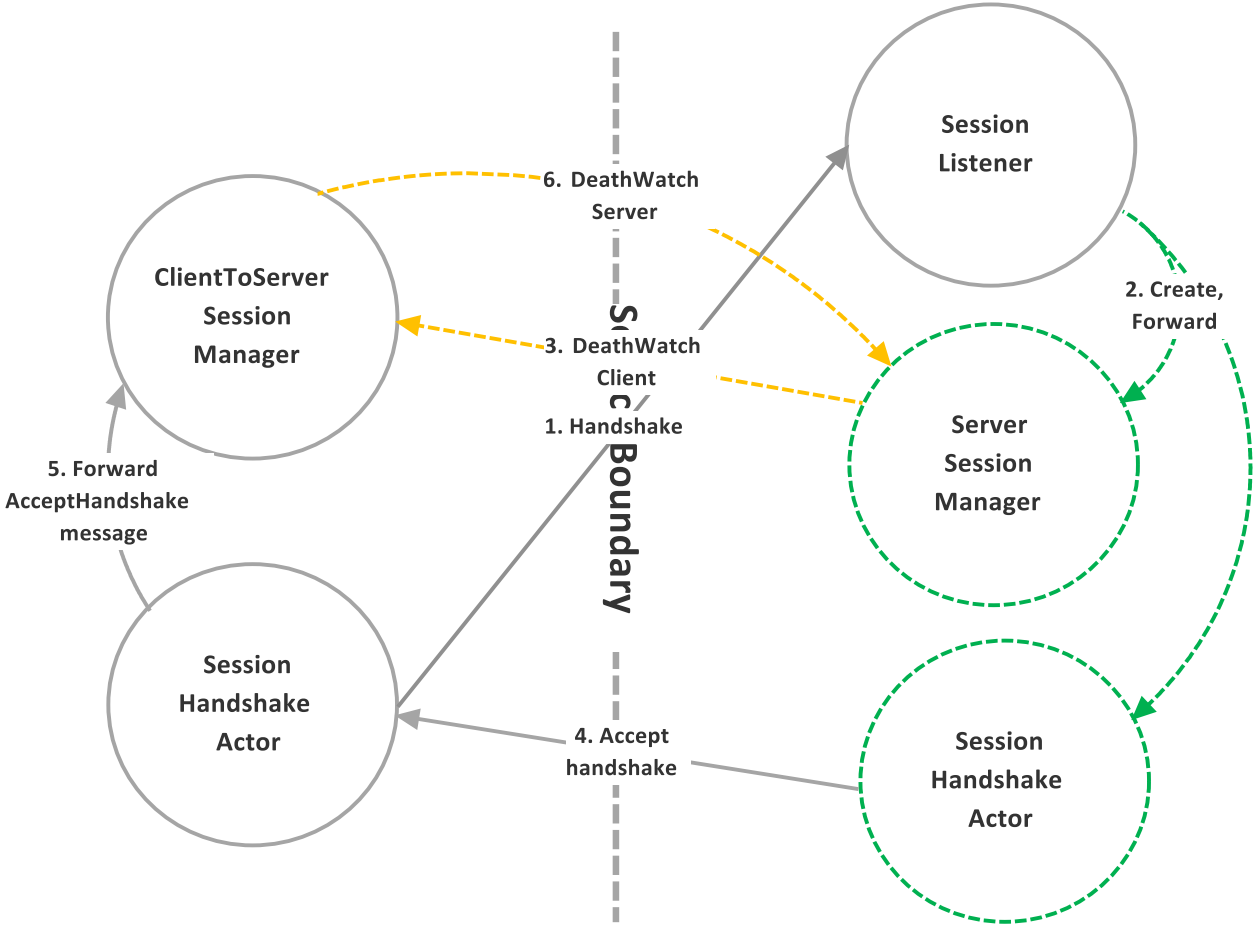
Retry / Reconnect

CODE SAMPLE

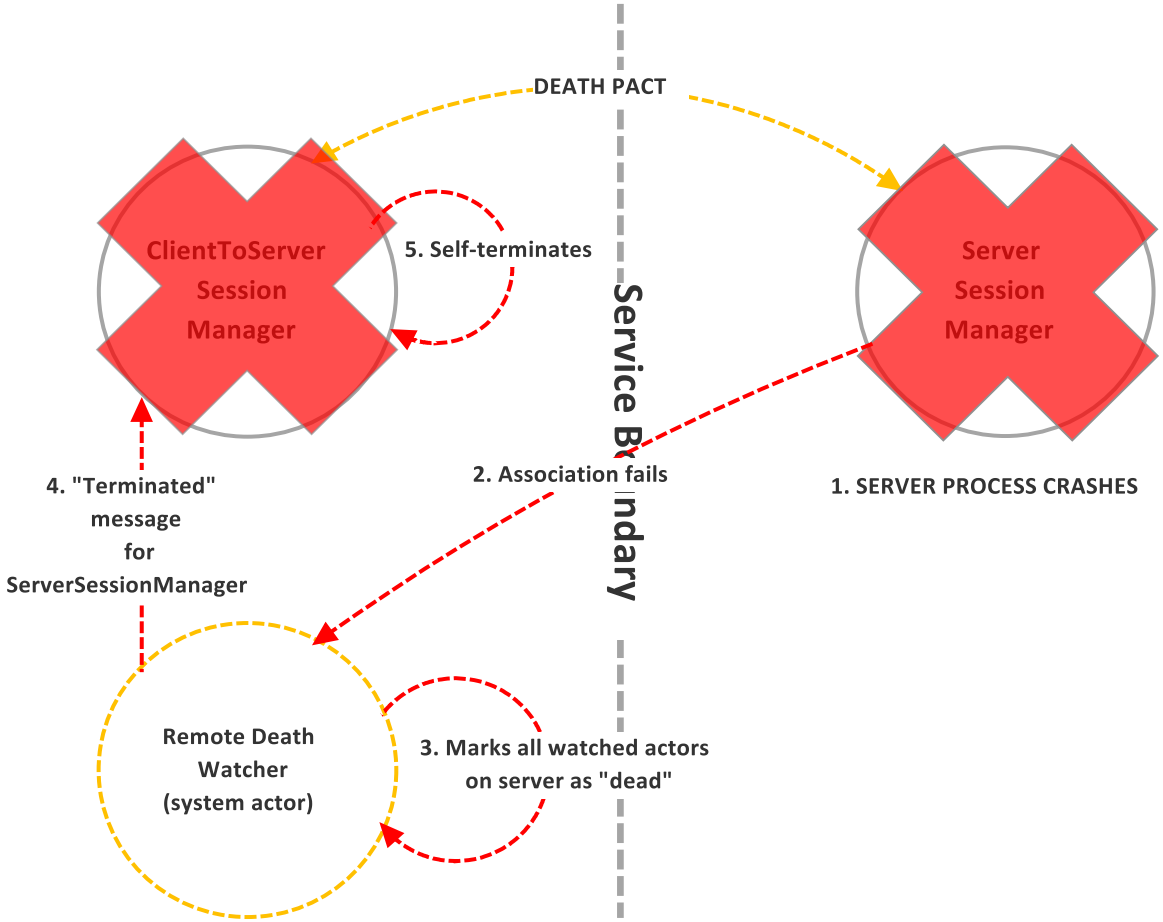
Dead Processes



Death Pact Setup



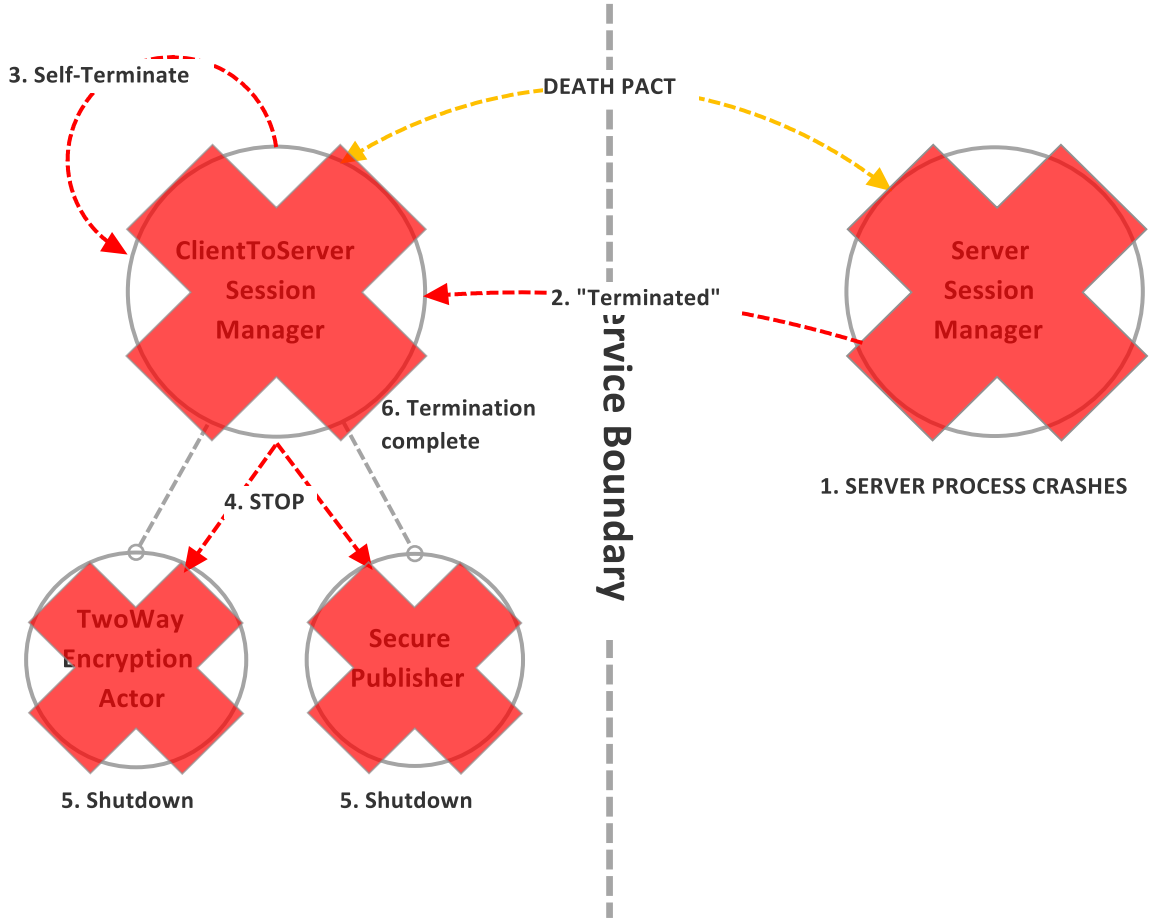
Death Pact Invocation



Death Pact

CODE SAMPLE

Self-Cleaning Hierarchy



Self-Cleaning Hierarchy

CODE SAMPLE

Other Trainings

Akka.NET Design Patterns

Akka.NET Clustering