The E1 Distributed Operating System Project

Anton Burtsev, Leonid Ryzhyk
<antonb,leonidr>@cse.unsw.edu.au

October 18, 2004

Goals

E1 Goals

- Provide efficient access to the resources of computer network.
- Implement a convenient programming model, isolating software developers from the intrinsic complexity of asynchronous distributed environment.

Distributed Objects

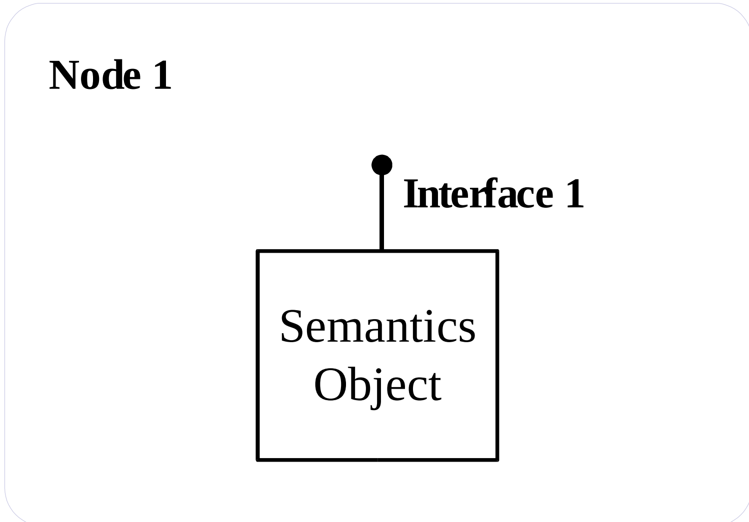
Distributed Object

$$\text{Distributed Object} = \bigcup_{\text{nodes}} (\text{Semantics Object} + \text{Replication Object})$$

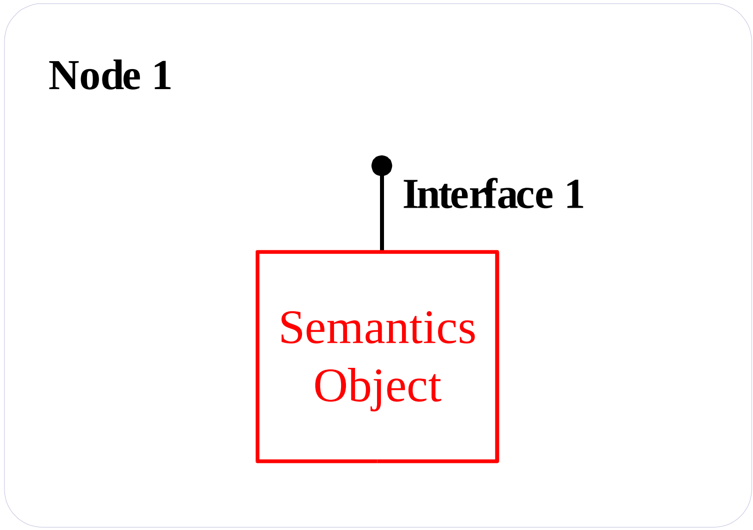
Trivial Case

When distributed object is used
only in one node:

Distributed Object = Semantics Object



Semantics Object

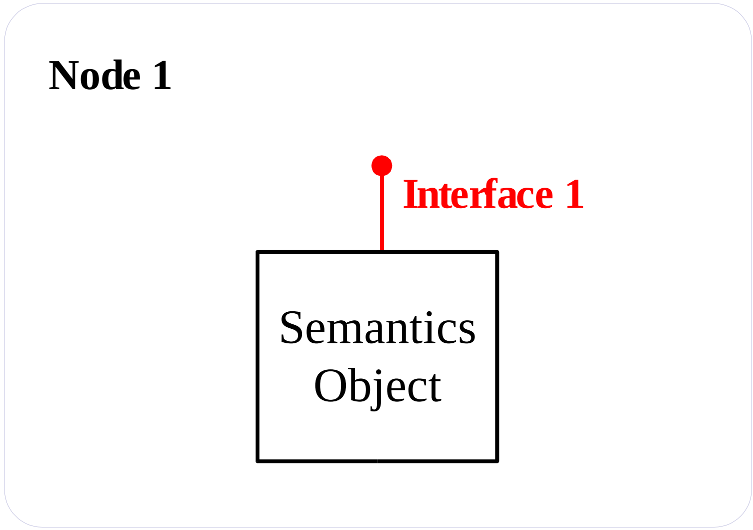


Semantics object is much like a C++ object, it:

- Stores object state
- Implements local object functionality

but ...

Semantics Object



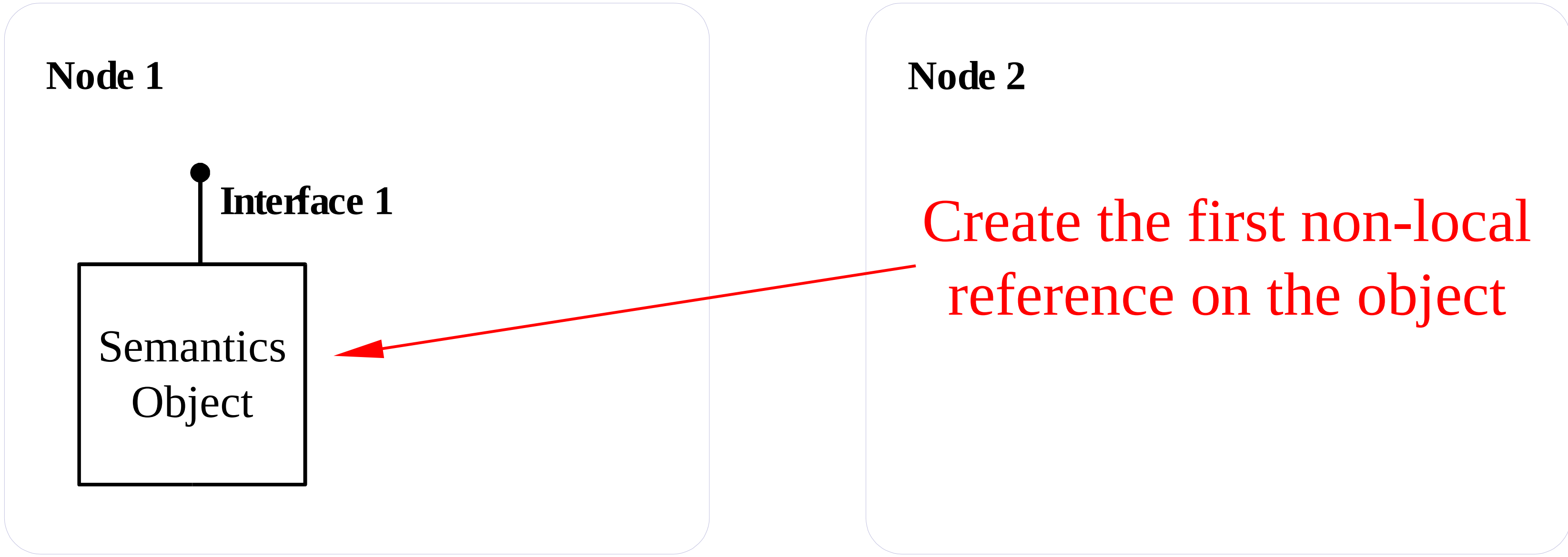
Semantics object is much like a C++ object, it:

- Stores object data
- Implements local object functionality

but it's accessible only via object interfaces

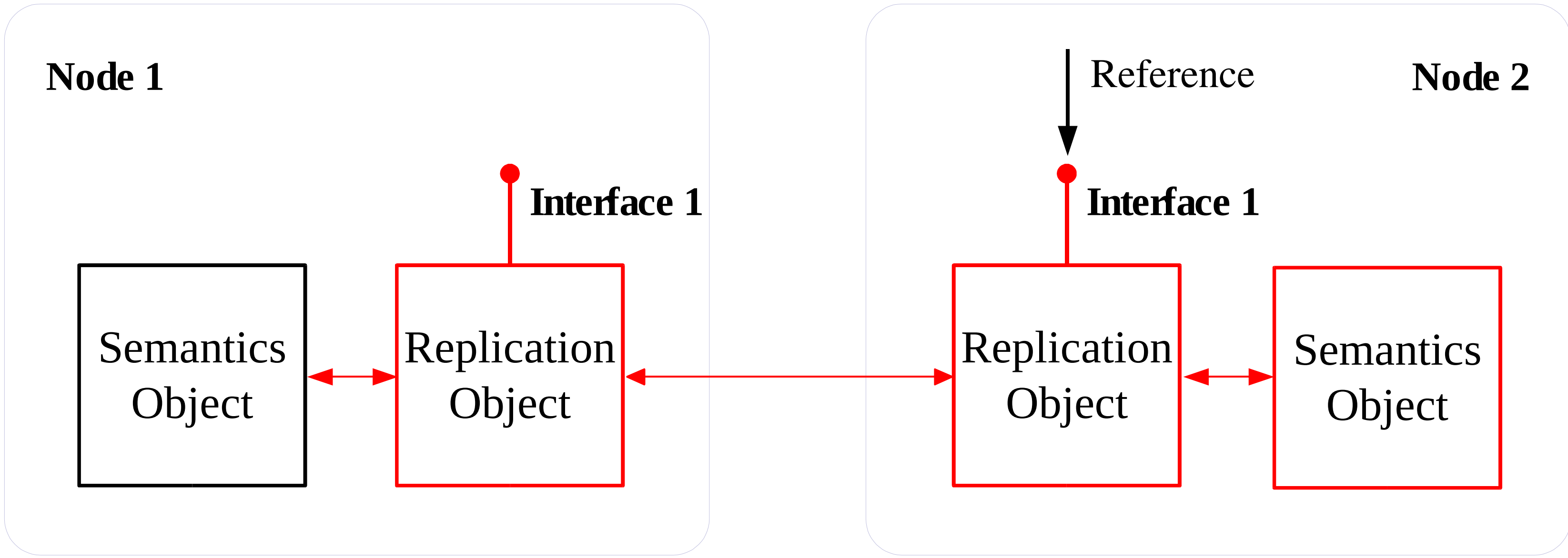
First Reference

Creation of the first non-local reference initiates creation of replication objects

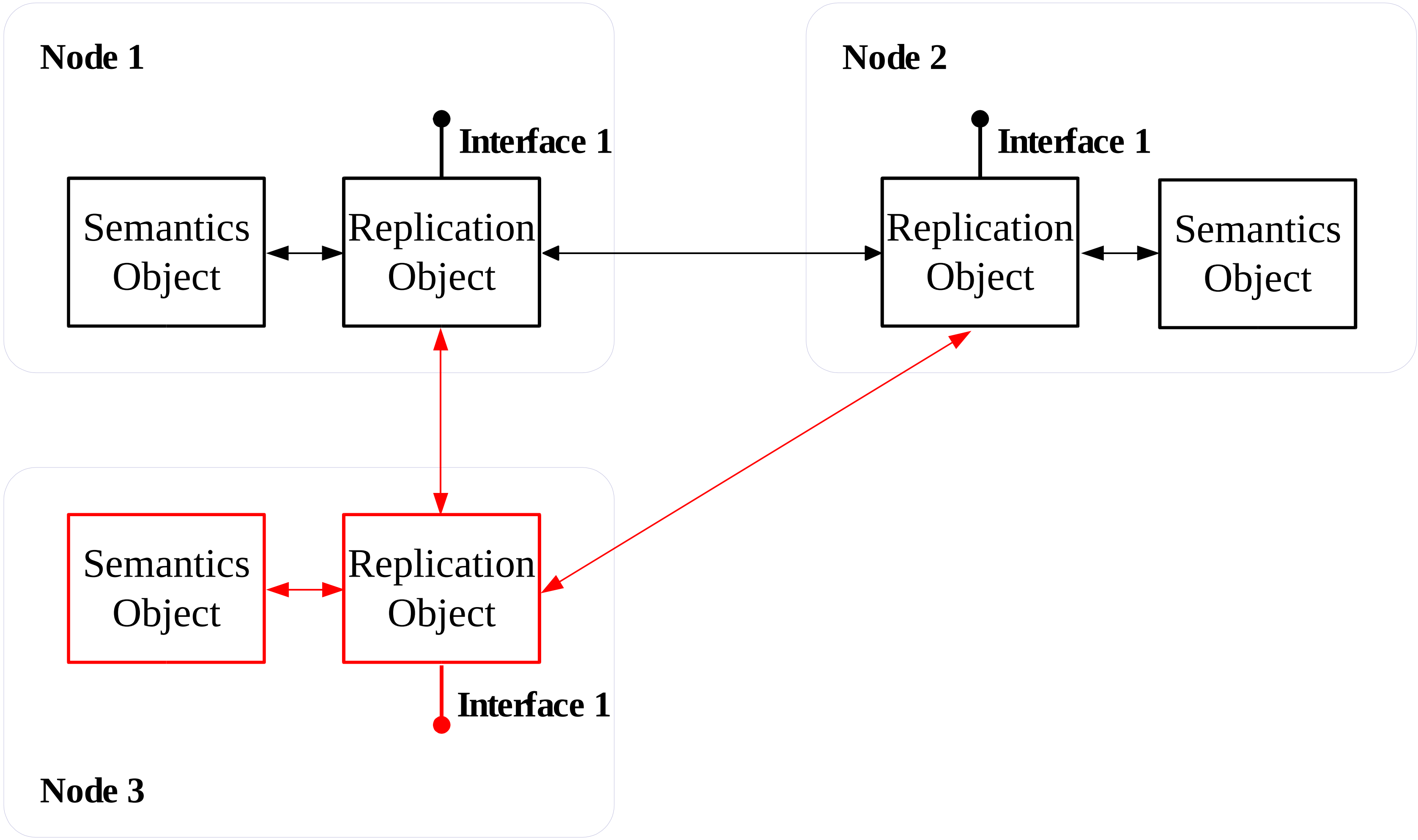


Replication Objects

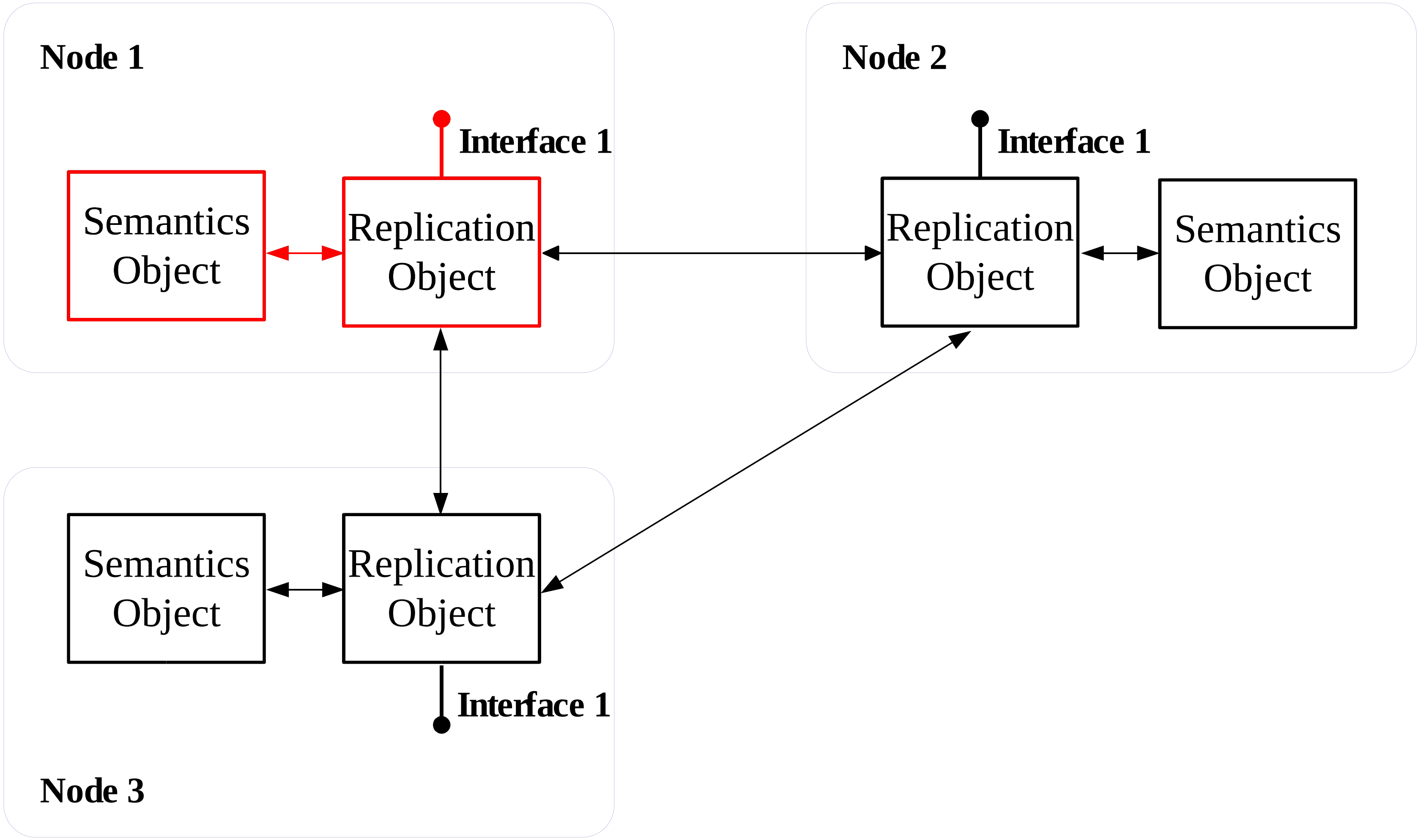
A pair of replication and semantics objects is created in each node



More Nodes...

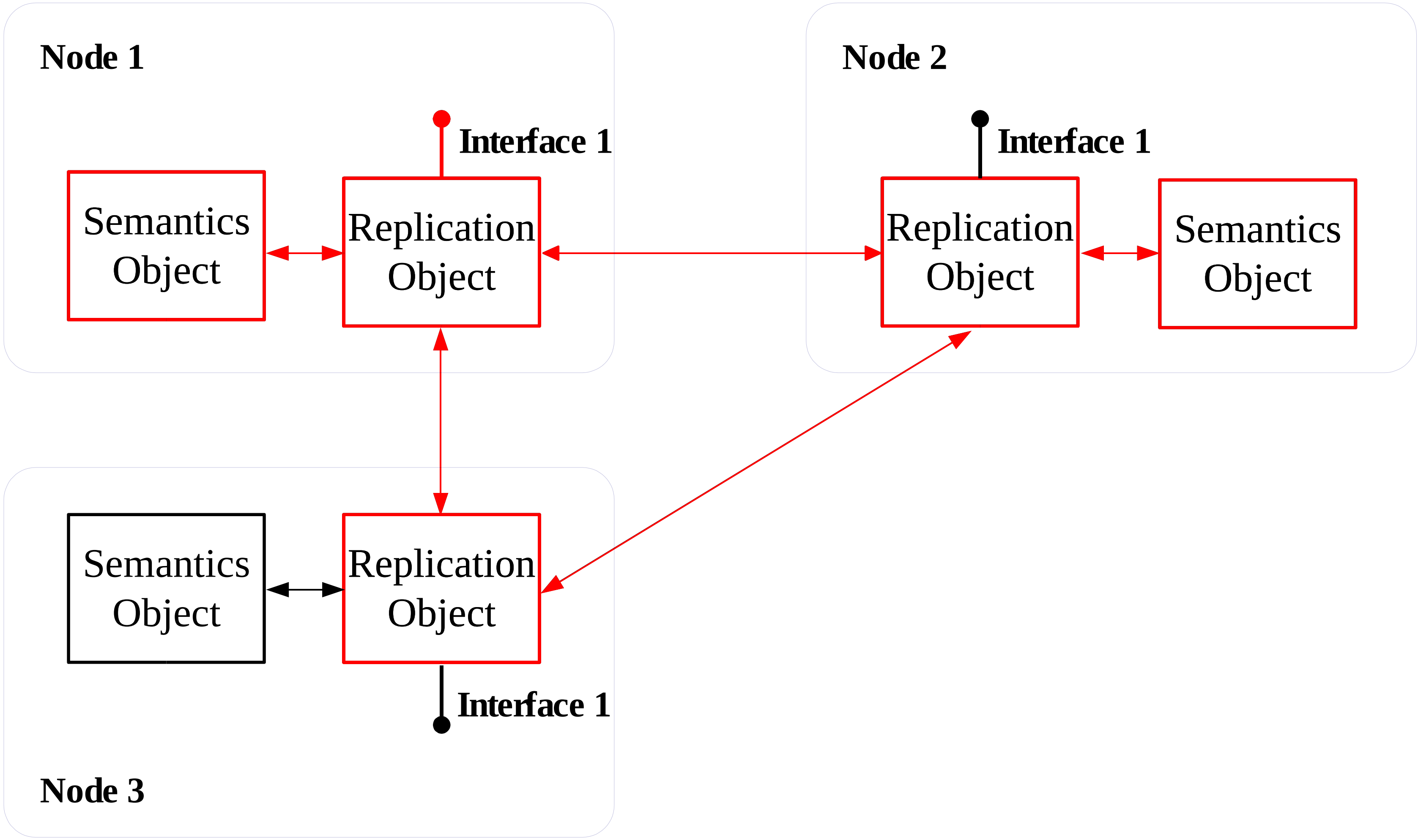

Node 1
Interface 1
Semantics Object
Replication Object
Node 2
Interface 1
Replication Object
Semantics Object
Semantics Object
Replication Object
Interface 1
Node 3

Object Invocations



A. Burtsev, L. Ryzhyk. October 2004

Object Invocations



Advantages

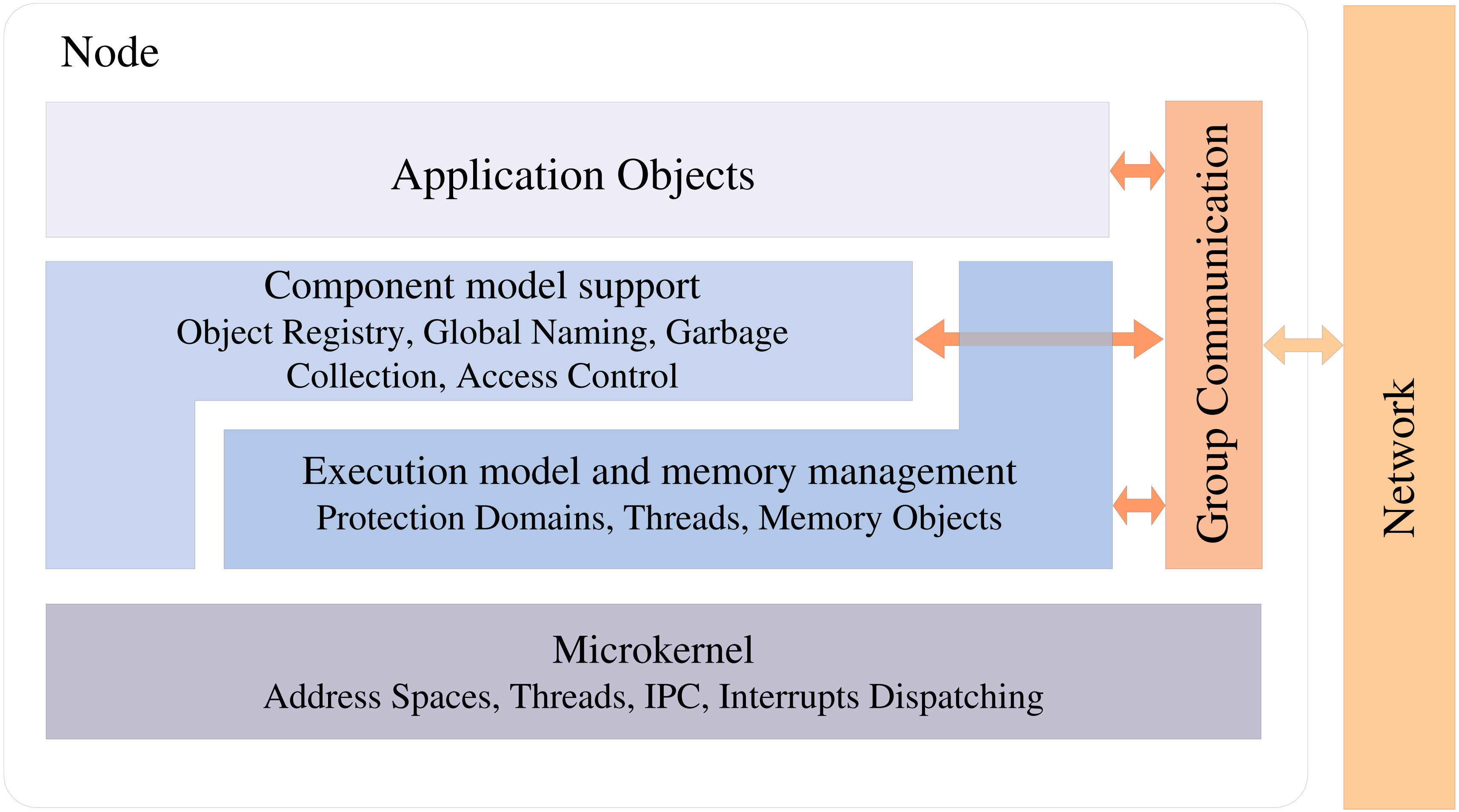
- Effective separation of object semantics and replication strategy
- No imposed restrictions on the replication strategies.

... Results

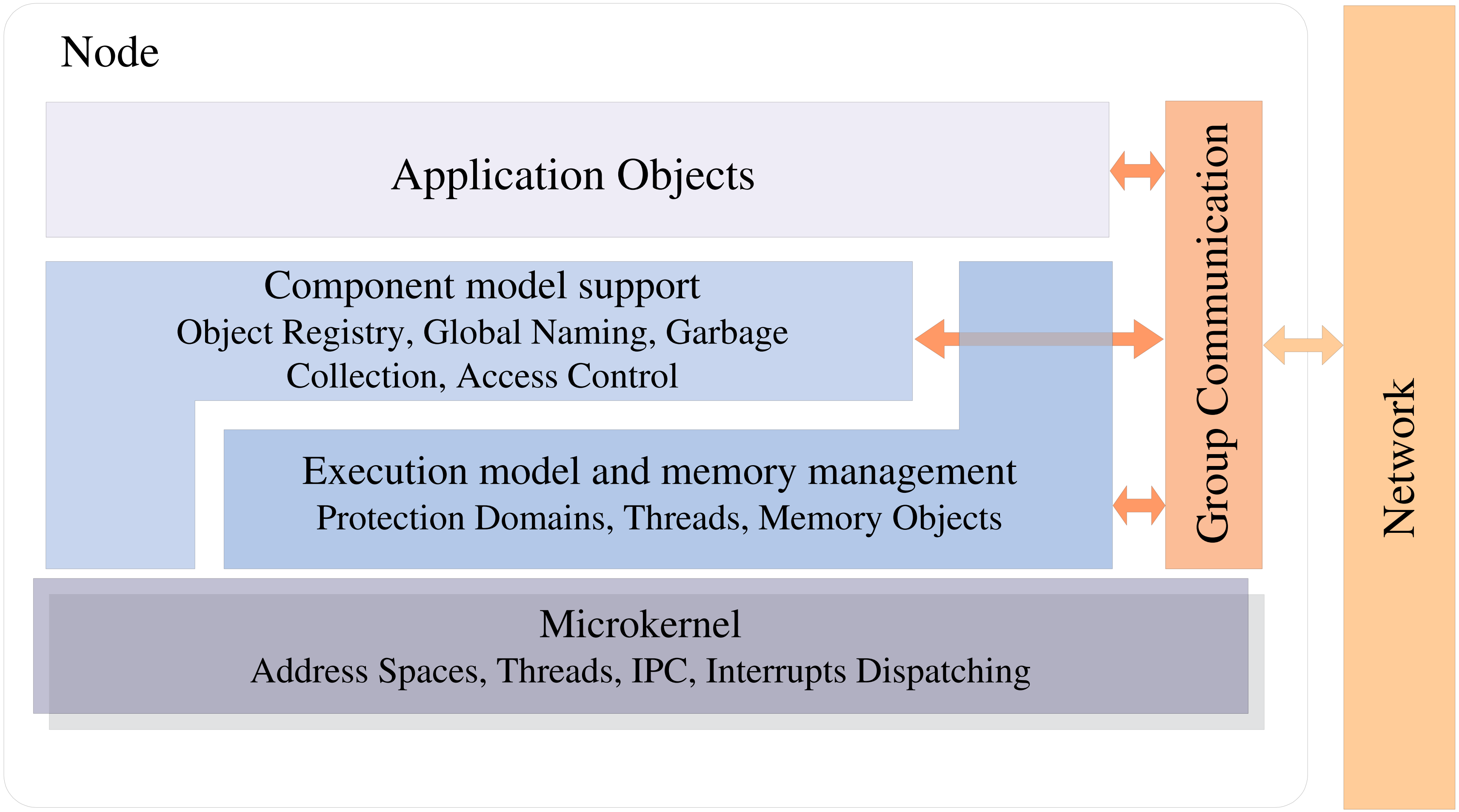
- Object functionality can be implemented separately from its replication strategy
- It's possible to select most efficient replication strategy for each object.

System Architecture

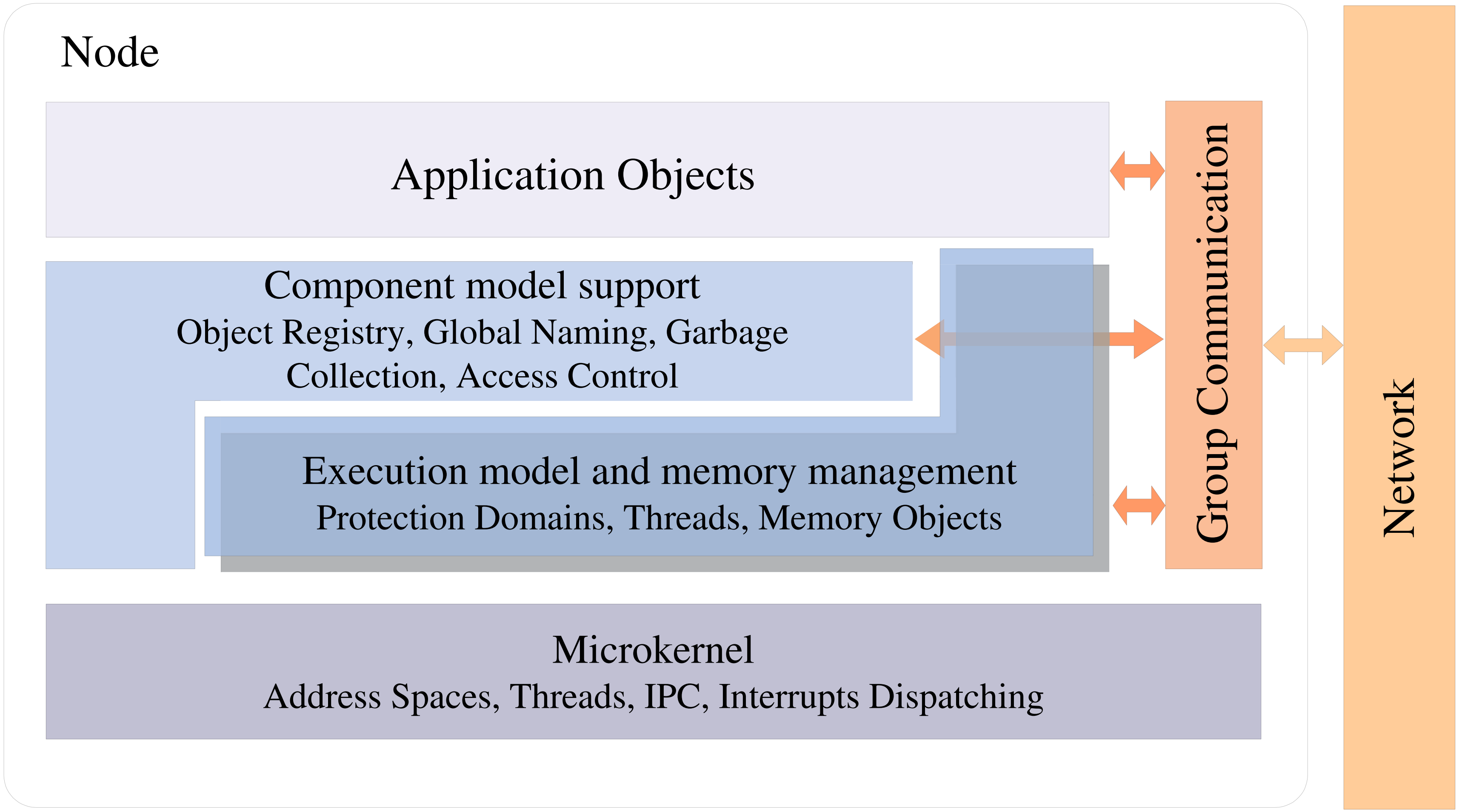
Generalized Architecture



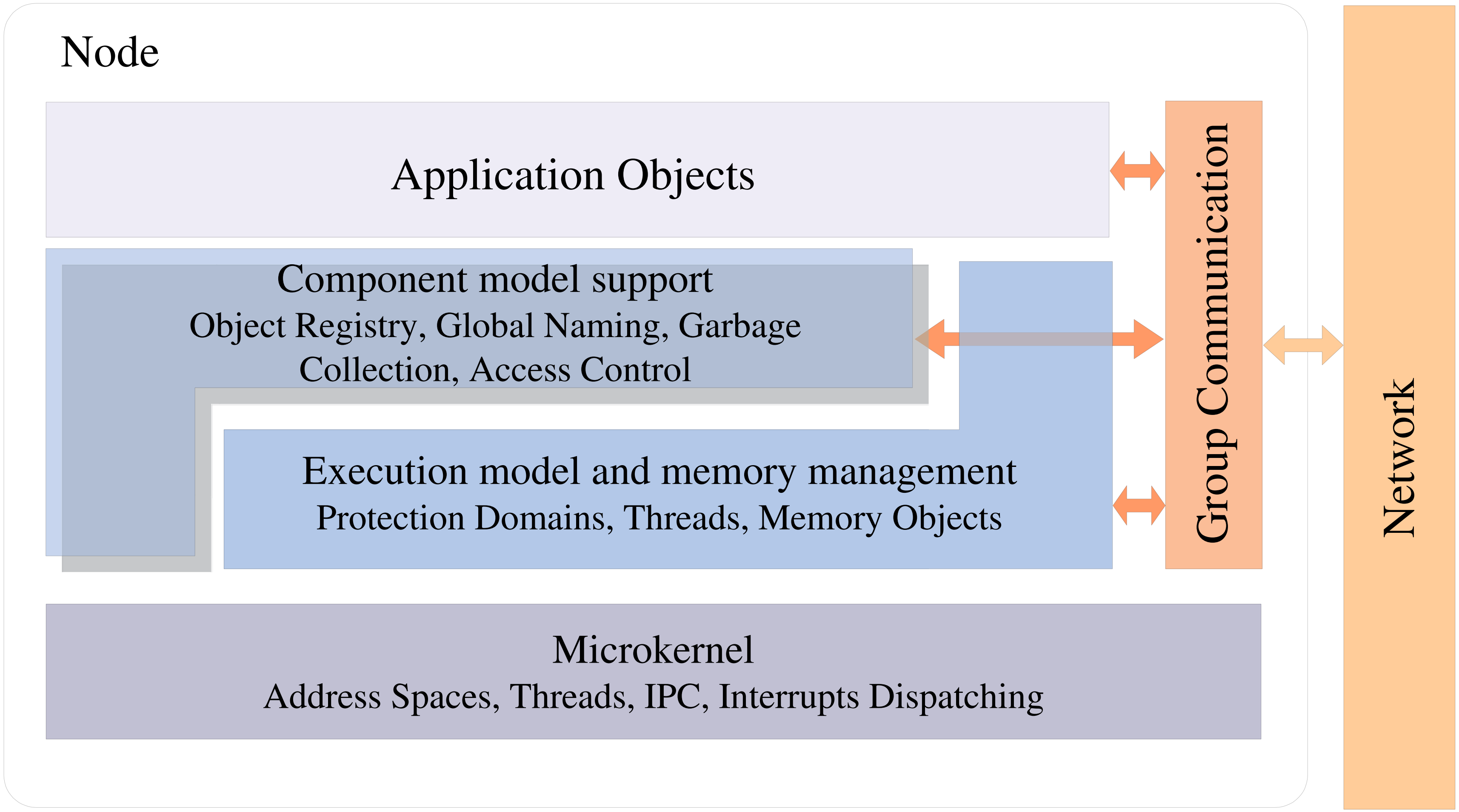
Generalized Architecture



Generalized Architecture



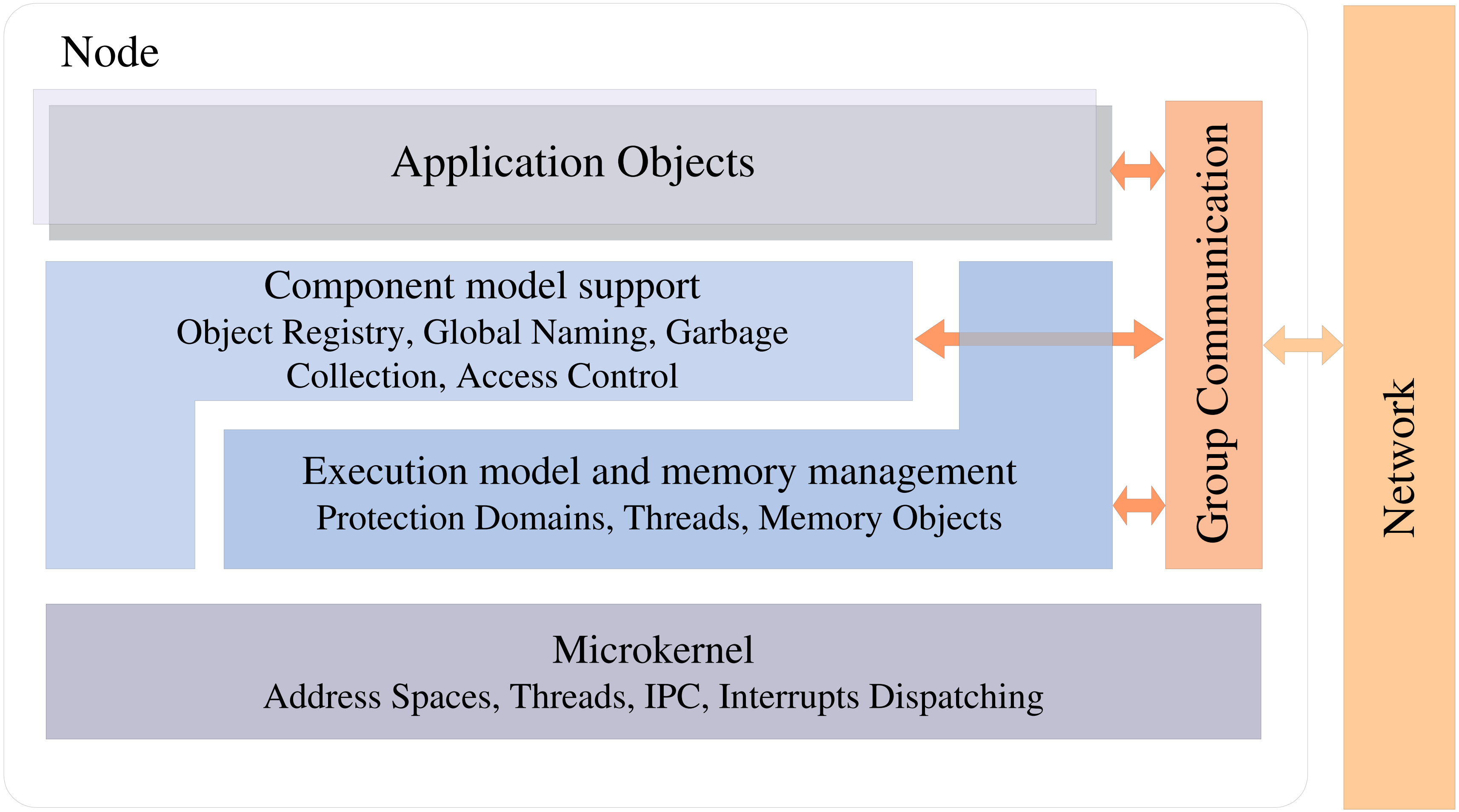
Generalized Architecture



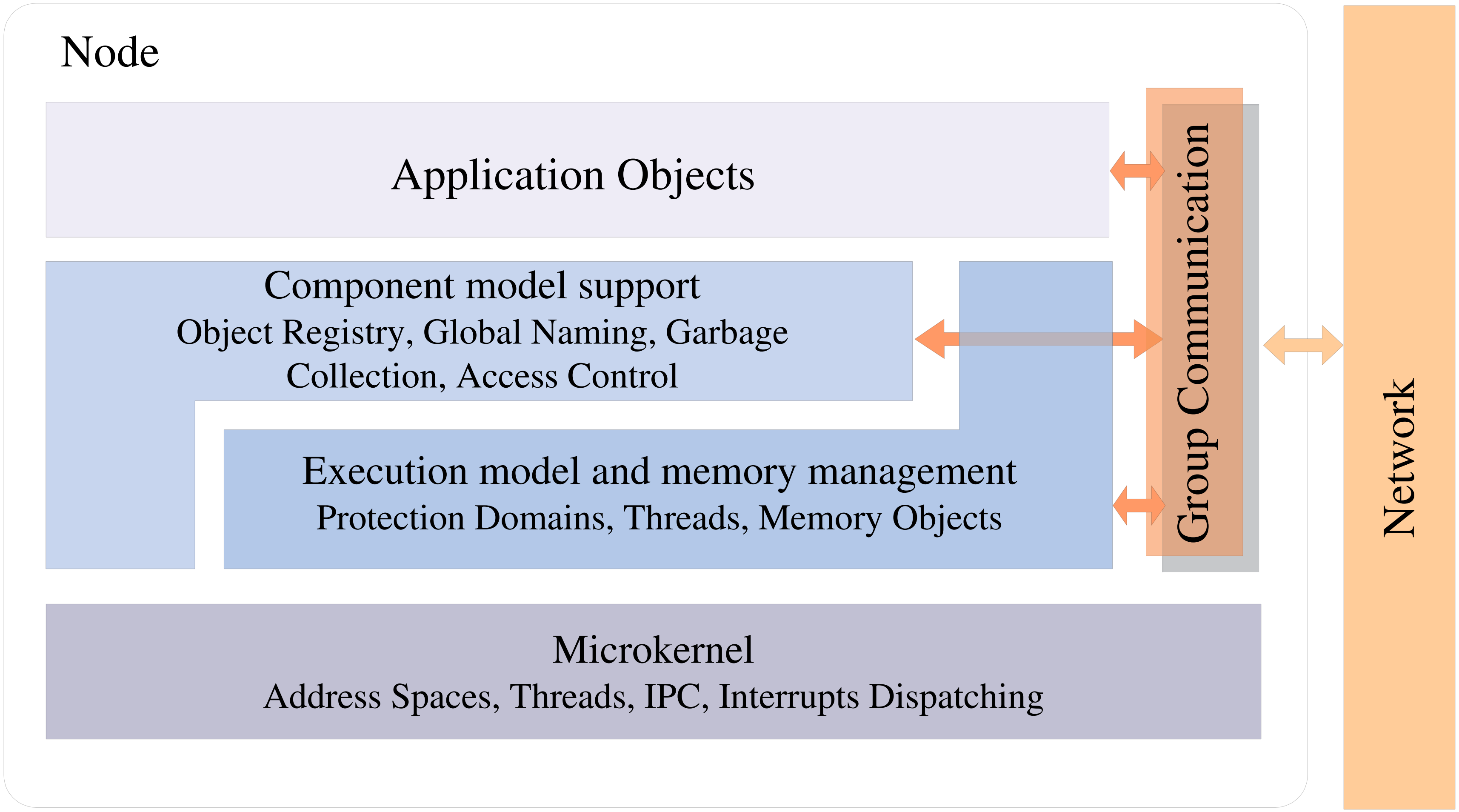
Component Services

- Lifecycle control
  - Object Registry
  - Distributed Garbage Collection System
- Global Naming Service
- Access Control Server

Generalized Architecture



Generalized Architecture



Replication

The Goal

Allow developers to define distributed behaviour of applications without implementing distributed algorithms.

- This is in contrast to DSM and RMI -based operating systems, which try to make distribution completely transparent by sacrificing efficiency.

Defining Distributed Behaviour

In E1 distributed behaviour of an object is defined by:

1. Selecting replication strategy for the object (possibly, at run time)
2. Adjusting replication strategy parameters:
   - consistency criteria;
   - required level of redundancy;
   - object topology (replica placement) ...

Other Replication Strategy Parameters

- timing properties;
- failure detection strategies;
- failure-handling policies;
- handling of network fragmentations;
- network protocol selection;
- etc.

Consistency Criteria

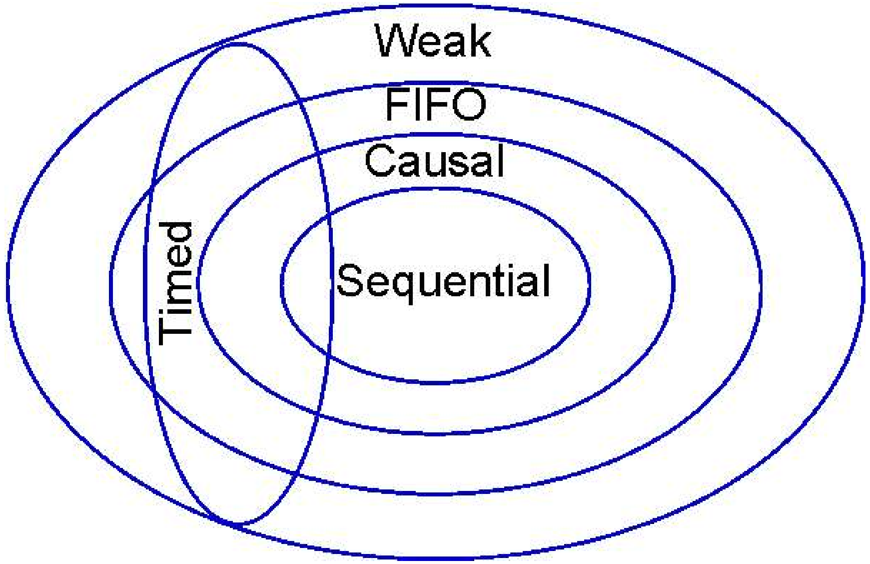
Strict consistency

- Sequential

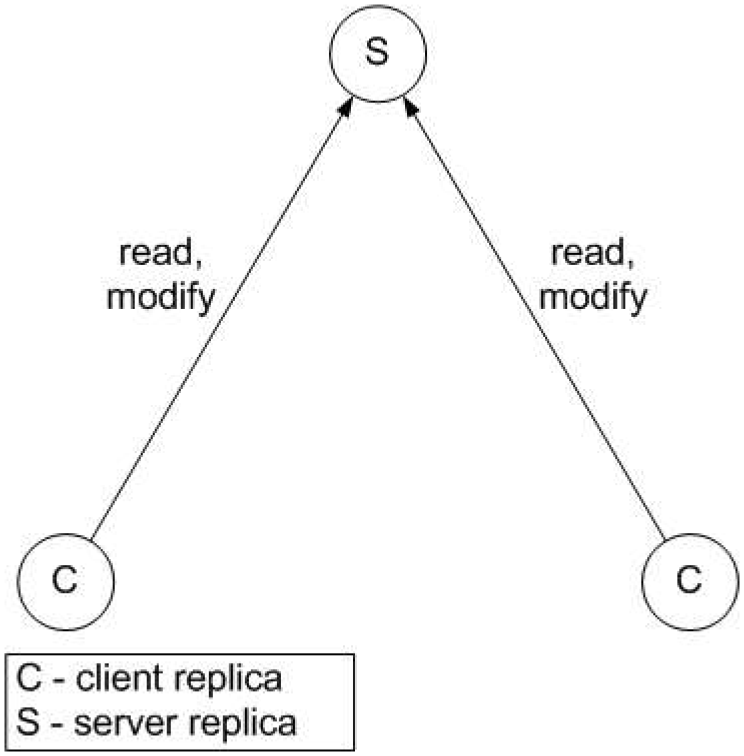
Relaxed consistency

- Causal
- FIFO
- Weak

Timed consistency



Client/Server Replication



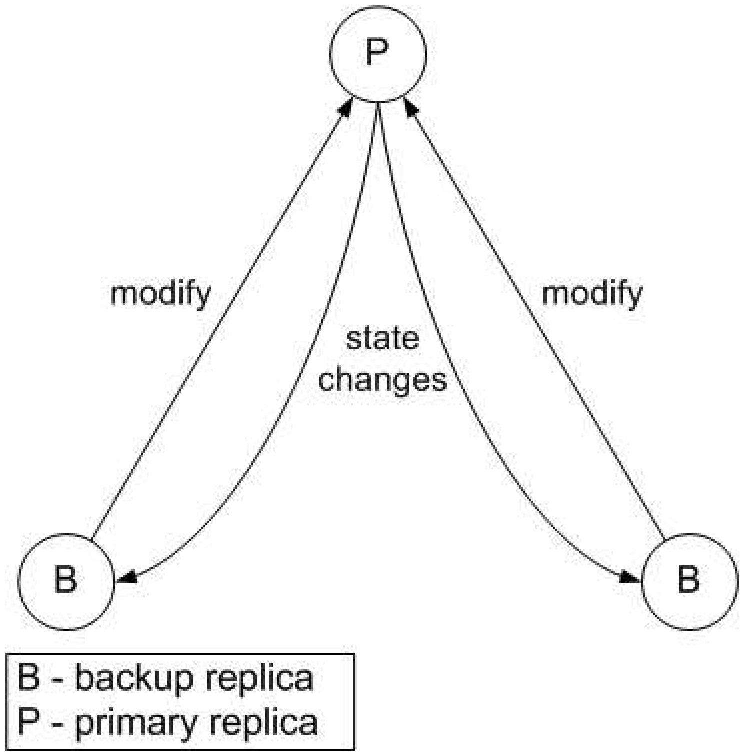
Pros:

+ universal

Cons:

- inefficient
- unreliable

Passive Replication



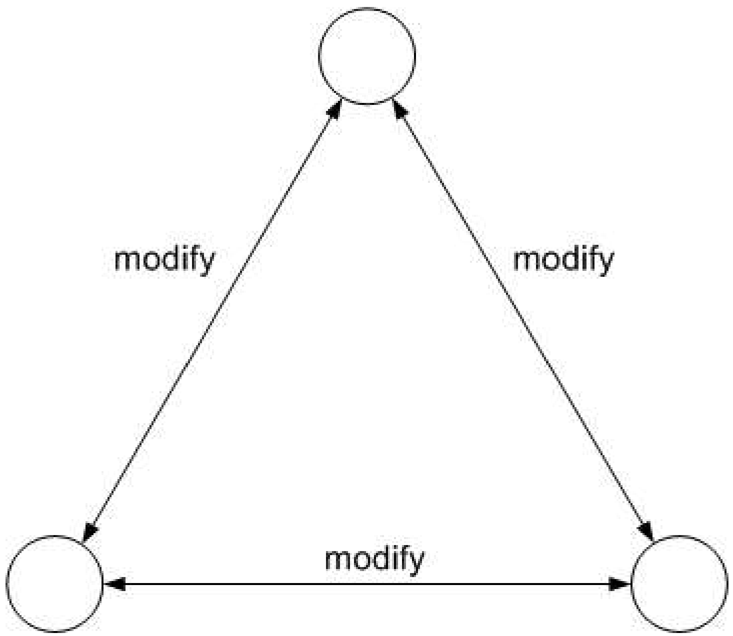
Pros:

+ reliability
+ reads are local

Cons:

- updates are not local

Active Replication



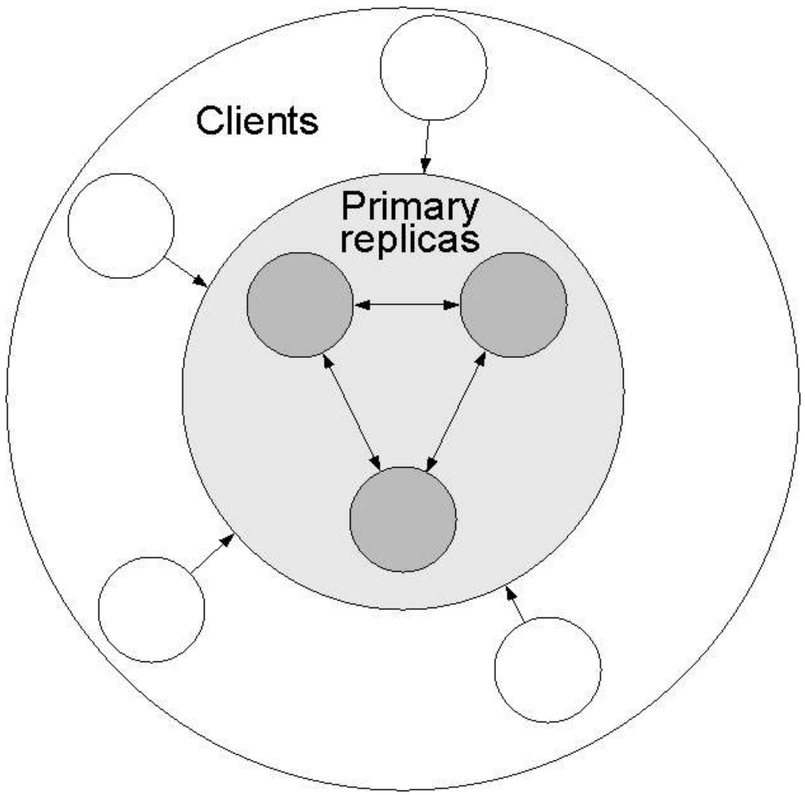
Pros:

+ reliability
+ both reads and updates are local when allowed by consistency criteria

Cons:

- requires deterministic behaviour
- recursive invocations are difficult

Example of a Real Strategy



A small set of active primary replicas with many stateless client replicas connected to them.

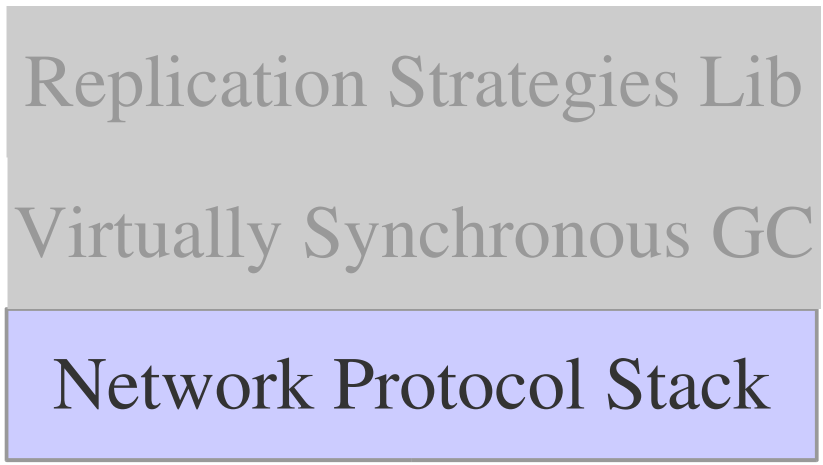
The Replication Strategies Framework

Replication Strategies Library

Virtually Synchronous Group Communication

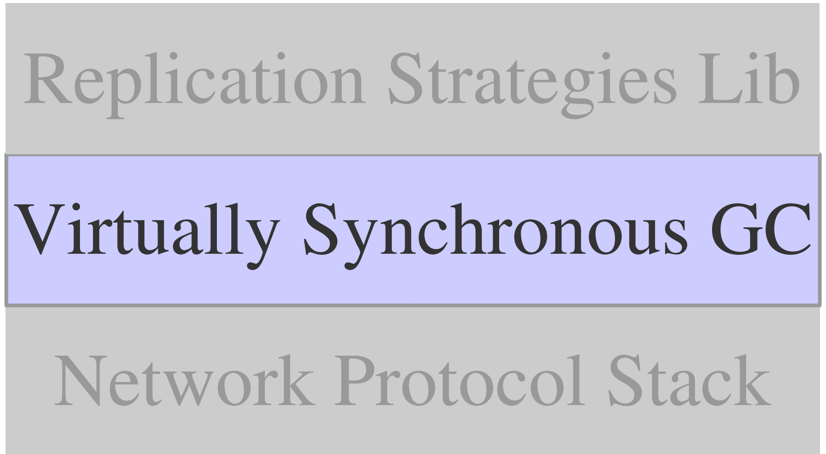
Network Protocol Stack

Network Protocol stack



- Network protocol layer provides at least unreliable unicast primitive.
- However, more advanced primitives, e.g. unreliable multicast or reliable unicast can also be available.

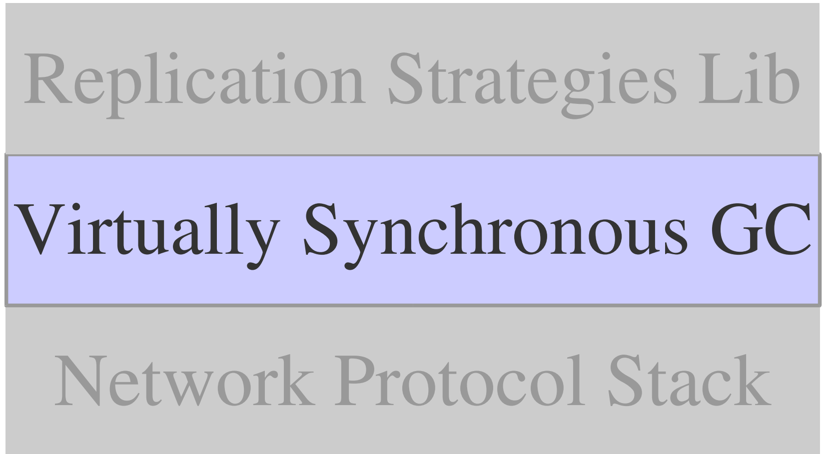
Virtually Synchronous GC



<u>Implements two types of services:</u>

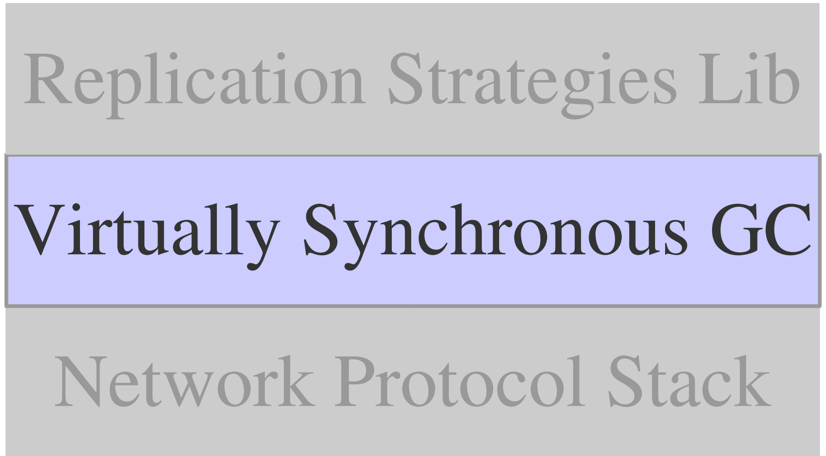
- Group membership service.
- Reliable unicast & multicast message delivery services with various ordering guarantees.

Group Membership Service



Detects crashed and recovered object replicas using unreliable failure detector and delivers consistent *views* of the group of replicas to all its members.

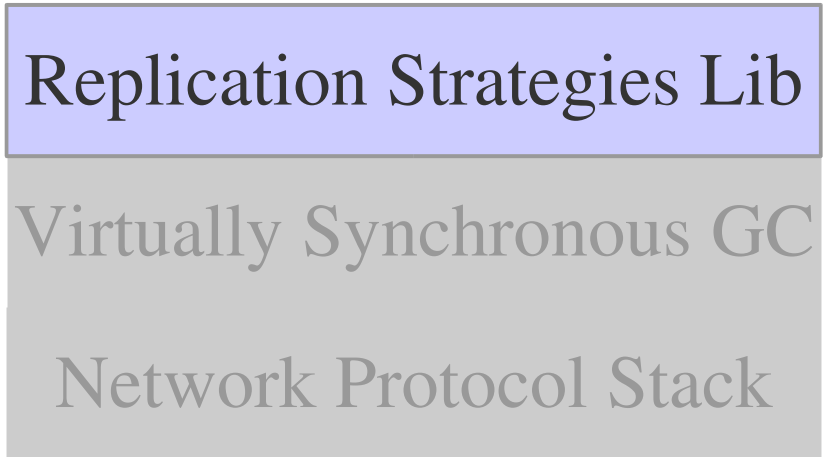
Message Delivery Service



<u>Message ordering properties:</u>

- FIFO multicast
- Causal multicast
- Totally ordered multicast

Replication Strategies Library



Requirements

- Completeness

  For virtually any object a replication strategy providing "good" performance can be found in the library.

- Customizability

  The developer can further fine-tune application performance by adjusting the replication strategy parameters.

- Extensibility

  New replication strategies can be easily developed by reusing existing components and design patterns.

Problem 1

A classical software engineering problem: given a number of related algorithms, construct a framework for unified development, evaluation, utilization and modification of these and similar algorithms.

For example, a similar problem has been successfully solved in the domain of group communication systems (Horus, Transis).

Problem 2

Under what conditions can two replicated objects (with different replication strategies) interact without breaking the consistency of each other?

- Completely ignored in previous research.

Serialization Interface

- Any non-trivial replication strategy involves operations requiring serialization/deserialization of object state:
  - Creation of a new replica;
  - Migration of existing replica to a new node;
  - State synchronization (passive replication).
- Objects are required to provide a serialization interface.

Serialization Interface

- Serialization can be cumbersome.
- Languages like Java and C# support automatic serialization based on RTTI.

- Problem: Implement automatic serialization for objects written in C/C++

Object State Components

- Dynamically allocated data;
- Static data (global variables);
- References to other objects;

Serializing References

References to other objects can be easily serialized by the operating system

Object State Components

- Dynamically allocated data;
- Static data (global variables);
- References to other objects;

Serializing Dynamic Data

We allow each object to have a separate private heap. Serialization of object dynamic data is then reduced to serialization of the heap.

- Possible only in single address space.
- Memory overhead.
  - Should be acceptable for medium-grained objects.
  - For small objects manual serialization is not difficult.

Object State Components

- Dynamically allocated data;
- Static data (global variables);
- References to other objects;

Serializing Static Data

- We allow each object to have a separate copy of writable data segment for each module it depends on.
- It is allocated from the heap on object creation and is serialized/deserialized together with the heap.
- Problem: we have to switch data segment when crossing module and object boundaries.

Serializing Static Data

We adopt Mungi approach with one modification.

- In Mungi all function pointers including C++ virtual method pointers are extended with *global pointer* field.
- In E1 this would require storing copies of all virtual tables in object heap.
- Instead, we store global pointer together with virtual table pointer in the object header.

Thank you