Communications in Distributed Systems

CS670/ECE 670

Shrideep Pallickara

Computer Science, Colorado State University

Communication is a fundamental primitive in distributed systems

- Data transfers among distributed components
- Access to methods, objects, or services

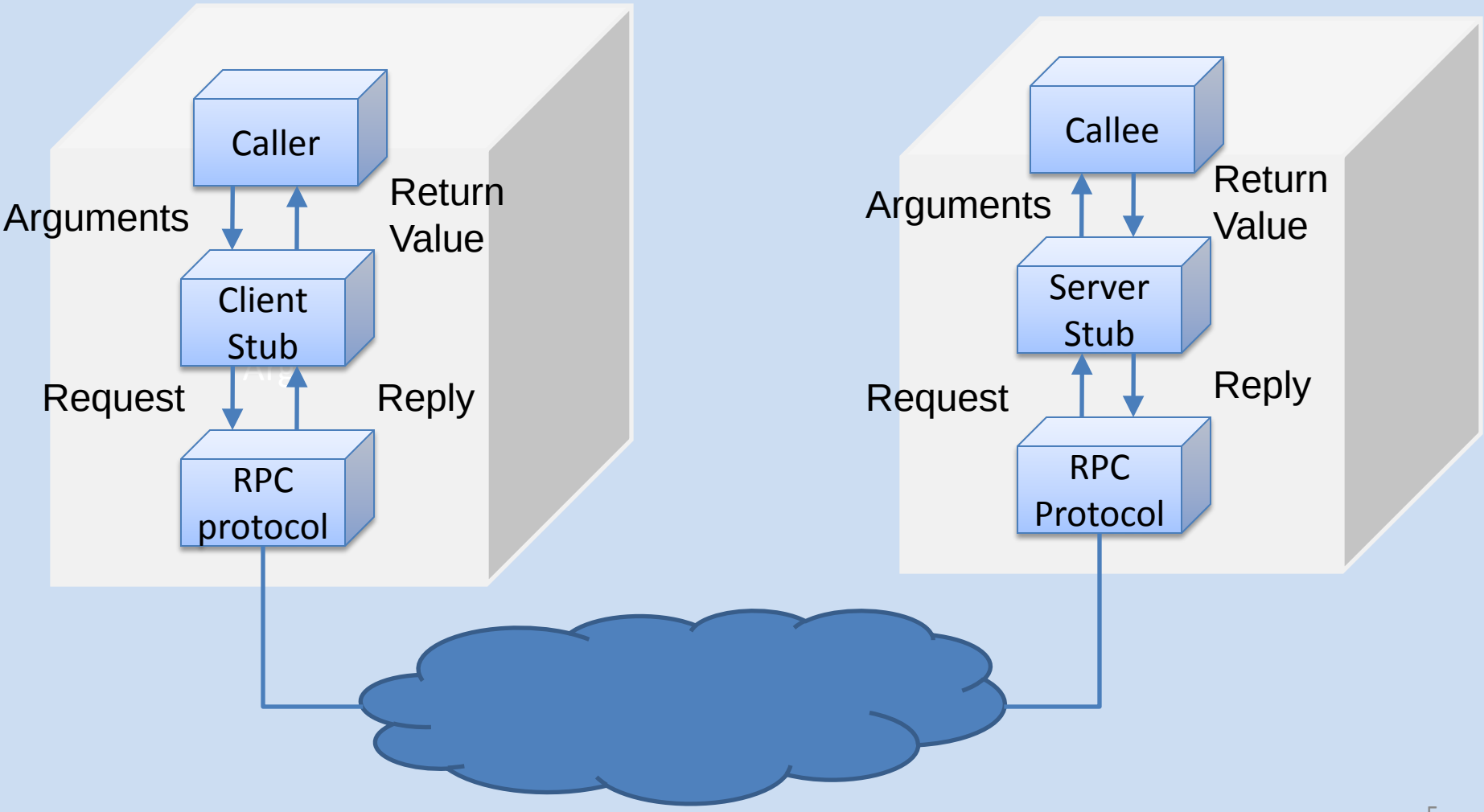
Remote Procedure Calls (RPC)

- Based on the request/reply model
- Based on semantics of a local procedure call
- Application makes CALL into a procedure (which may be local or remote), **and** BLOCKS until call returns
- Origins: RFC 707 (1976). First use Xerox 1981 (Courier)

RPCs are slightly more complicated than local procedure calls

- Network between the Calling process and Called process can **limit** message sizes, **reorder** them or **lose** them
- Computers hosting processes may differ
  - Architectures and data representation formats.

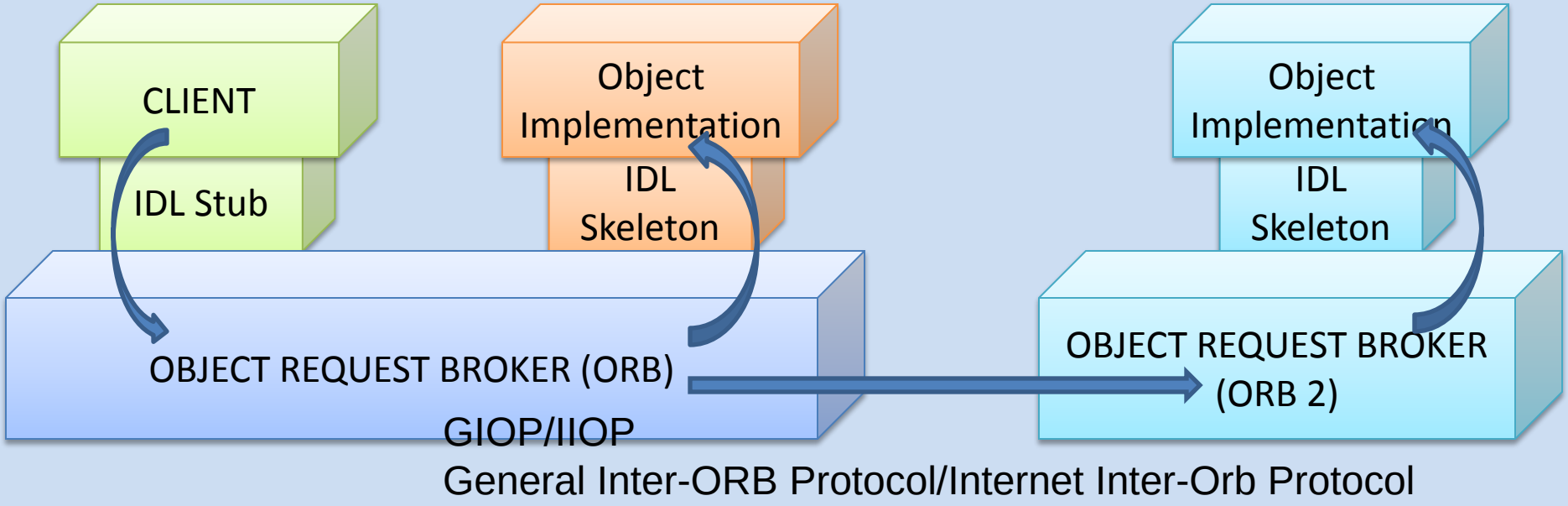
RPC Mechanism



Distributed Objects

- RPC based on distributed objects with an **inheritance** mechanism
- **Create**, **invoke** or **destroy** remote objects, and interact as if they are local
- Data sent over network:
  - **References**: class, object and method
  - Method **arguments**
- CORBA early1990s, RMI mid-late 90s

Distributed Objects in CORBA defined using the Interface Definition Language



Web Services in some sense borrowed some of these concepts

- Used XML to describe services: Web Services Description Language
  - Defined methods and arguments to them
- Added another feature/**problem**
  - Generation of WSDL from actual implementation.

Message queuing systems: The store-and-forward approach

- Based on **asynchronous** communications between the producer and consumer
- Producers **place** messages on to a queue
  - When destination is not available
- Queuing system is responsible for **resending** messages from queue when destination is available
- Examples: Microsoft MQ, Websphere MQ

Message queuing: Problems

- Typically point-to-point communications
- Store-and-forward does not scale particularly well

Multicast is used for one-to-many communications

- Uses datagram packets
- Sender sends packet to multicast address
  - 224.0.0.0 to 239.255.255.255 **Class D**
- Routers make sure packet delivered to all hosts in multicast group
  - Choose points where streams are duplicated
- Pay attention to **TTL** for the datagrams
  - Max number of routers a datagram is allowed to cross

Multicast is not particularly suitable in some situations

- **Packet size** restrictions in Multicast
- Practical considerations
  - Turned **off** at several institutions to curb free-riding

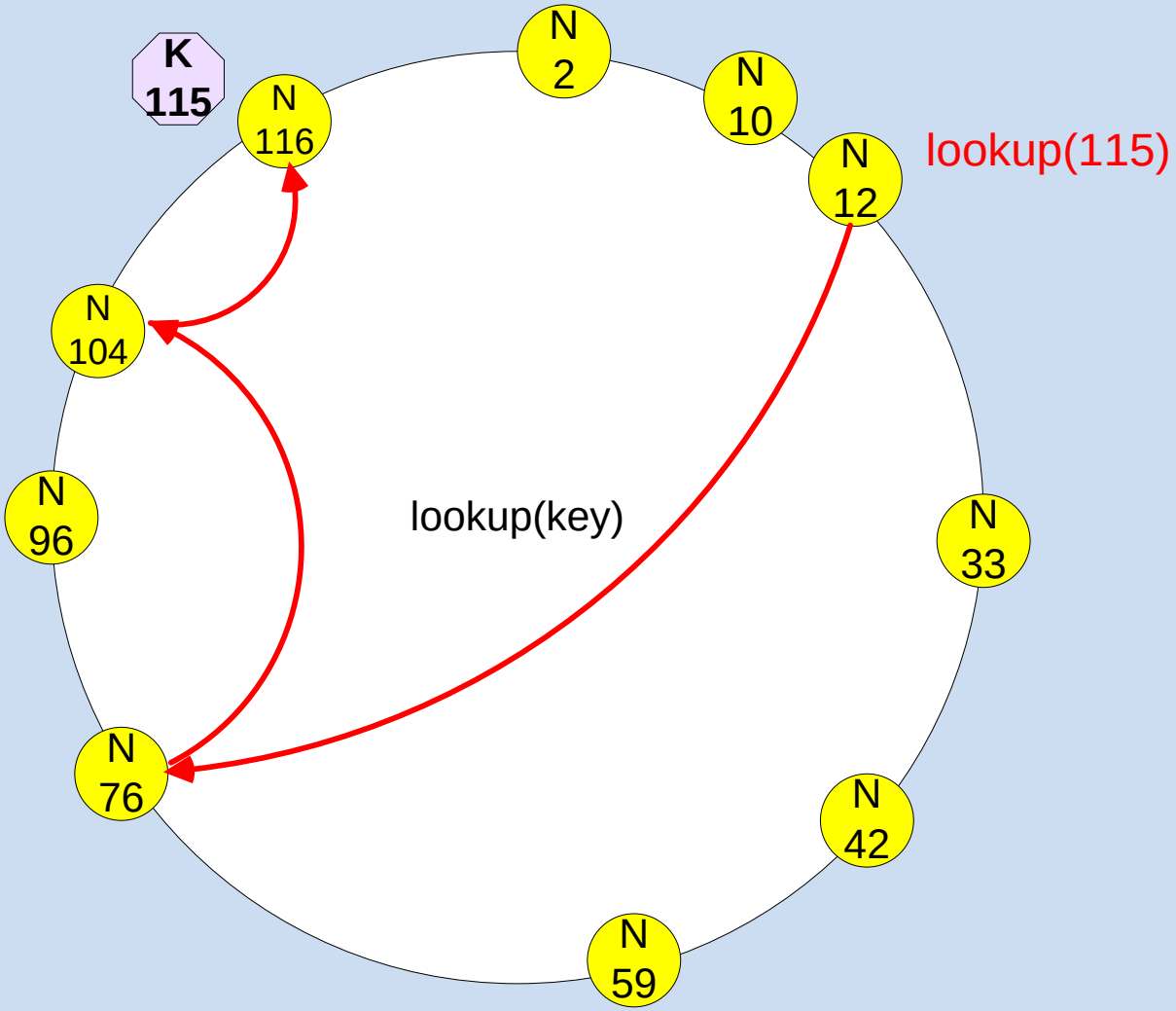
Multicast is not particularly suitable in some situations

- Groups cannot be pre-allocated
  - Consumption patterns change dynamically
- Representing consumption profiles as groups
  - Enormous number of groups – potentially 2^N for N consumers.
  - Eliminating impossible groups would still require millions of groups.

Peer-2-Peer networks use peers (each other) to route data

- Unstructured P2P networks
  - Flooding model: Each node sends data to **all** its neighbors
    - Uses variant of TTL to control the flood
- Structured P2P networks
  - Most commonly used scheme is DHT (Distributed Hash Table)

P2P networks: An example of how basic DHT works



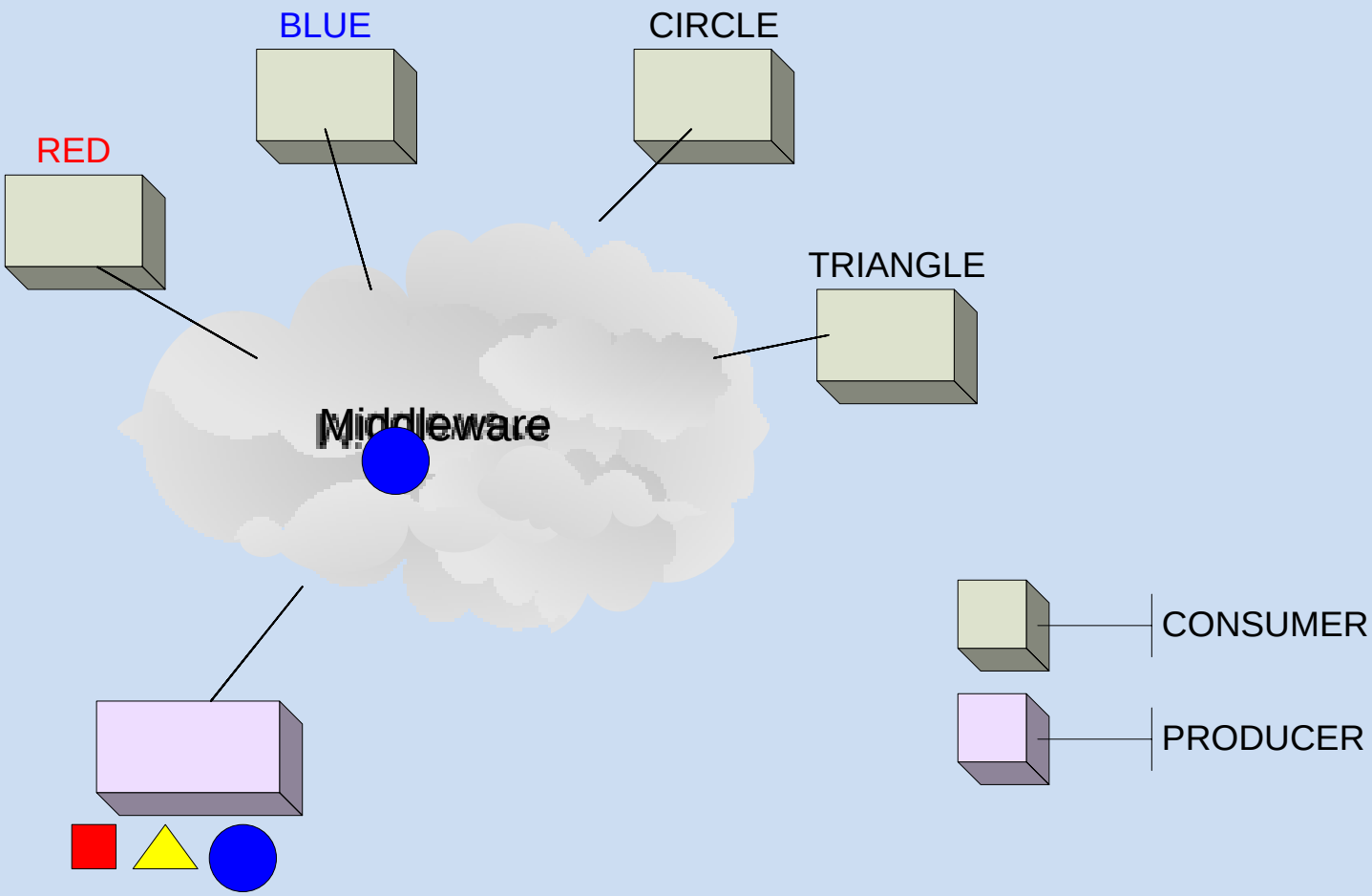
Publish/subscribe is very suitable for scalable content dissemination

- **Demarcates** producer and consumer roles
- **Loosely-coupled**
  - No need for producers and consumers to know each other
- Data dissemination is within the purview of the CDN
  - Comprises a set of software router nodes
    - Logical Overlay

Publish/Subscribe Systems

- Subscriptions
  - Predicates specified by consumers to assert **data of interest** to them
    - Could be “/” separated Strings, <tag, value> tuples, SQL or Regular Expression queries
- Advertisements
  - Describes content descriptors for a stream
- Data
  - Has to be self descriptive with values for content descriptors

Publish/Subscribe Systems: An example



Managing Web Content Delivery: Akamai

- Websites redirect users to **Akamaized URL**s
- IP address associated with client used to select server-farm *closest* to client.
  - Most popular content served up from caches
    - Benefits of caching and network proximity
- Server farms **sync up** with managed websites to track content changes.

Issues in data distribution

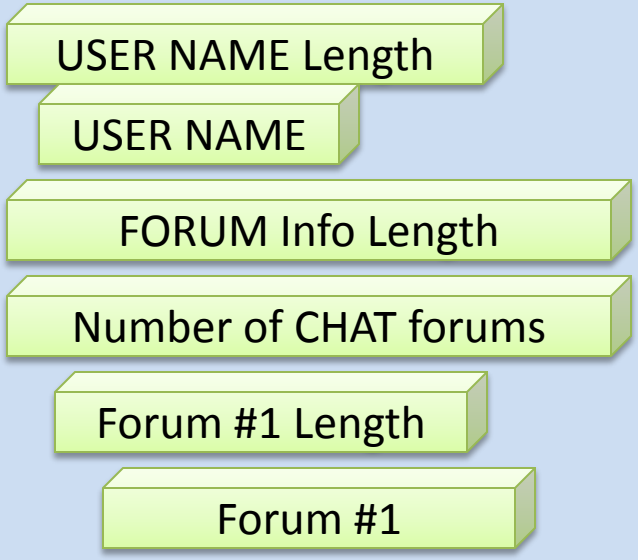
- **Latency** for communications
- Security
  - **Integrity**, **tamper-evidence** & **non-repudiation**
    - Hashing and digital signatures
  - **Confidentiality**
    - Encryption/Decryptions and key distributions
- Coping with **failures** *(availability)*
- Data **volumes**
  - Number of entities and generation rates

Building a Chat Server

- TCP-based communications
- First pass
  - Set up a TCP ServerSocket and accept inbound connections
  - All active connections to the server receive all messages sent over any of the connections.

Pass 2: Manage different forums

- Incorporate support for registering to different chat forums
  - Registration message



Message Types and such …

- Publishing format

  – Message sent to a forum should be sent to all active participants on that forum
- Requests
  – (De)Register (from) to forum
  – Retrieve Forum memberships
  – Relay Messages to OTHER servers

Some questions pertaining to the ability to scale

- What's the maximum number of connections that we can go up to?
- How do your response times behave as the number of clients increase?
  - What is the distribution of response times for different clients?

Advanced Scaling

- Can we use a thread-pool of worker threads to cope with larger number of clients?
  - Fine grained pipelining strategies
  - Non-blocking IO
  - Interleaving IO with processing
  - Lightweight, reliable concurrency constructs
- Can we set up a head-node that forwards connections to a backend cluster?