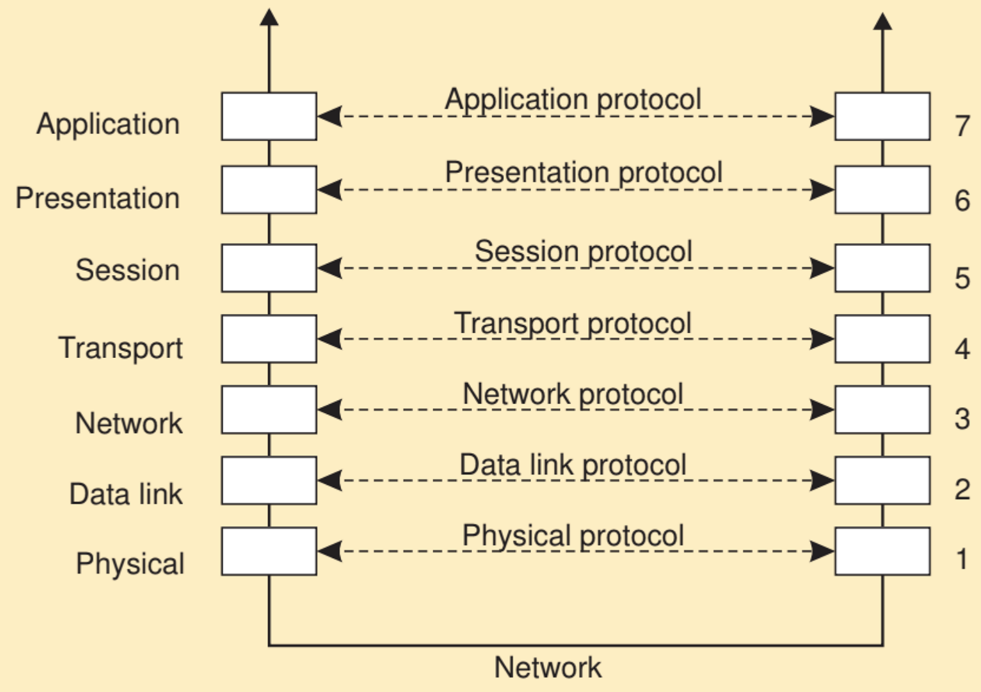
Middleware for Communication and Messaging

CS 237

Distributed Systems Middleware **(with slides from Tanenbaum and Van Steen book , Cambridge Univ and Petri Maaranen)**

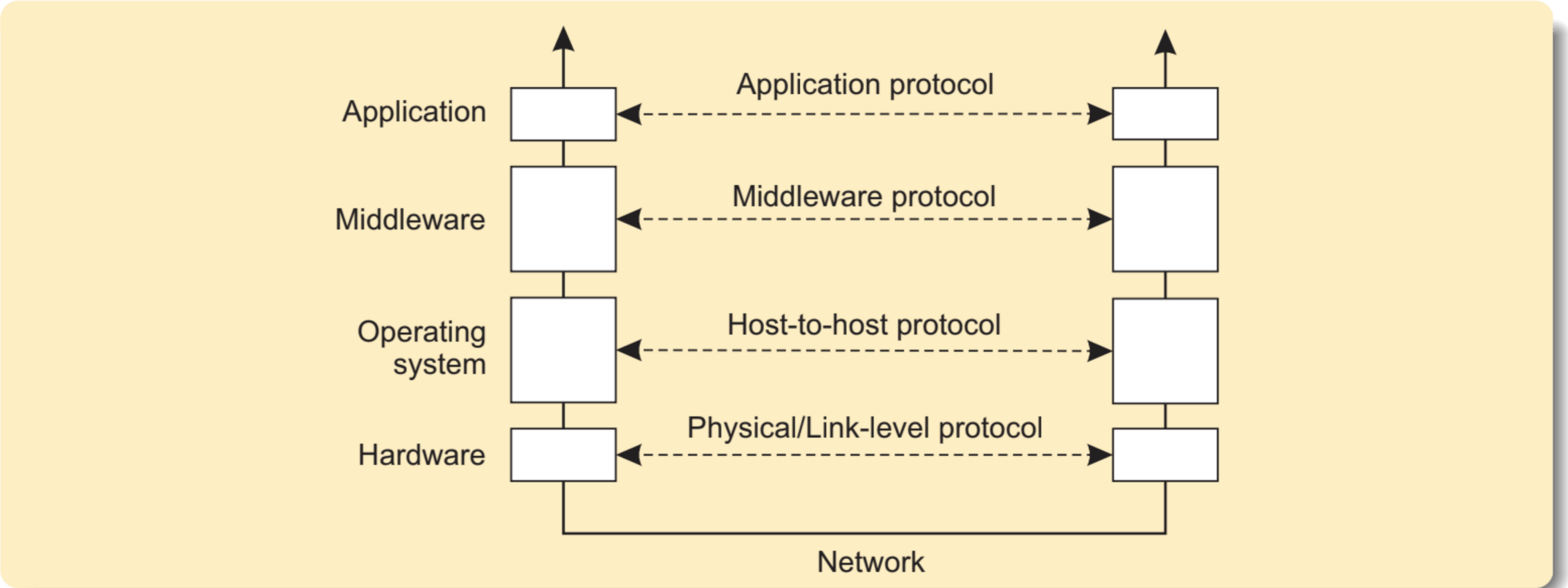
Traditional networking stack



- Physical layer: contains the specification and implementation of bits, and their transmission between sender and receiver
- Data link layer: prescribes the transmission of a series of bits into a frame to allow for error and flow control
- Network layer: describes how packets in a network of computers are to be routed.

The transport layer provides the actual communication facilities for most distributed systems.

For distributed systems

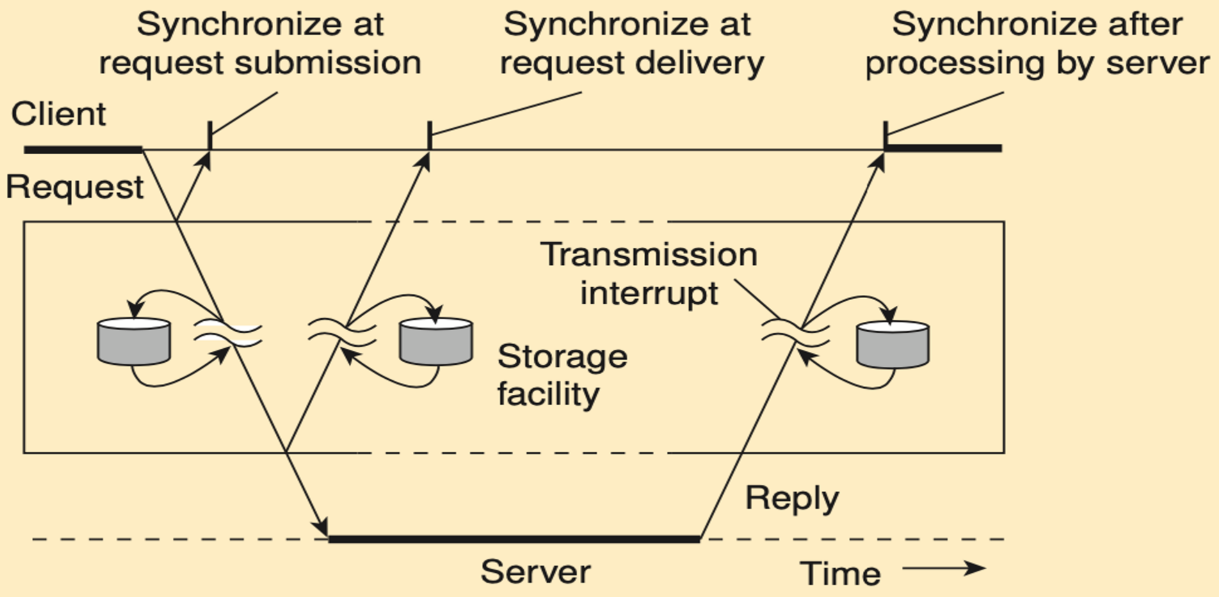


Observation

Middleware is invented to provide common services and protocols that can be used by many different applications

- A rich set of communication protocols
- (Un)marshaling of data, necessary for integrated systems
- Naming protocols, to allow easy sharing of resources
- Security protocols for secure communication
- Scaling mechanisms, such as for replication and caching

Communication Types


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

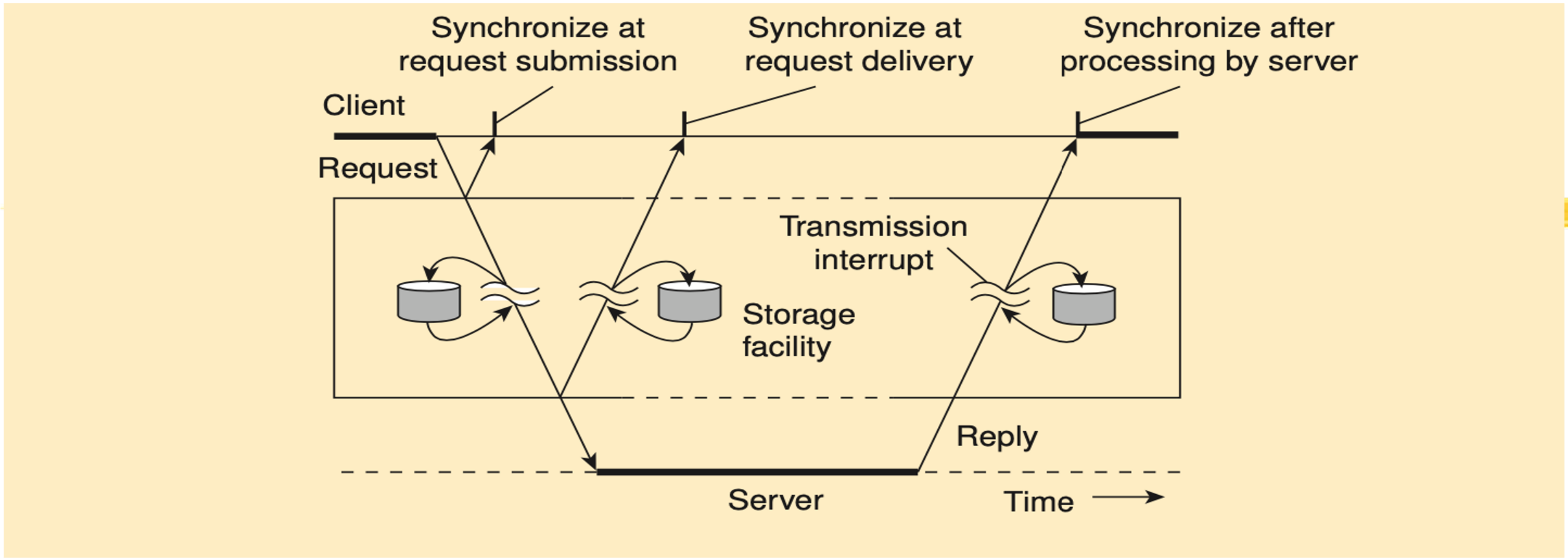
- Transient communication: Comm. server discards message when it cannot be delivered at the next server, or at the receiver.
- Persistent communication: A message is stored at a communication server as long as it takes to deliver it.

When to synchronize?

- At request submission
- At request delivery
- After request processing

Synchronous vs. Asynchronous communication

Communication Types


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

Client/Server computing is generally based on a model of transient synchronous communication:

- Client and server have to be active at time of communication
- Client issues request and blocks until it receives reply
- Server essentially waits only for incoming requests, and subsequently processes them

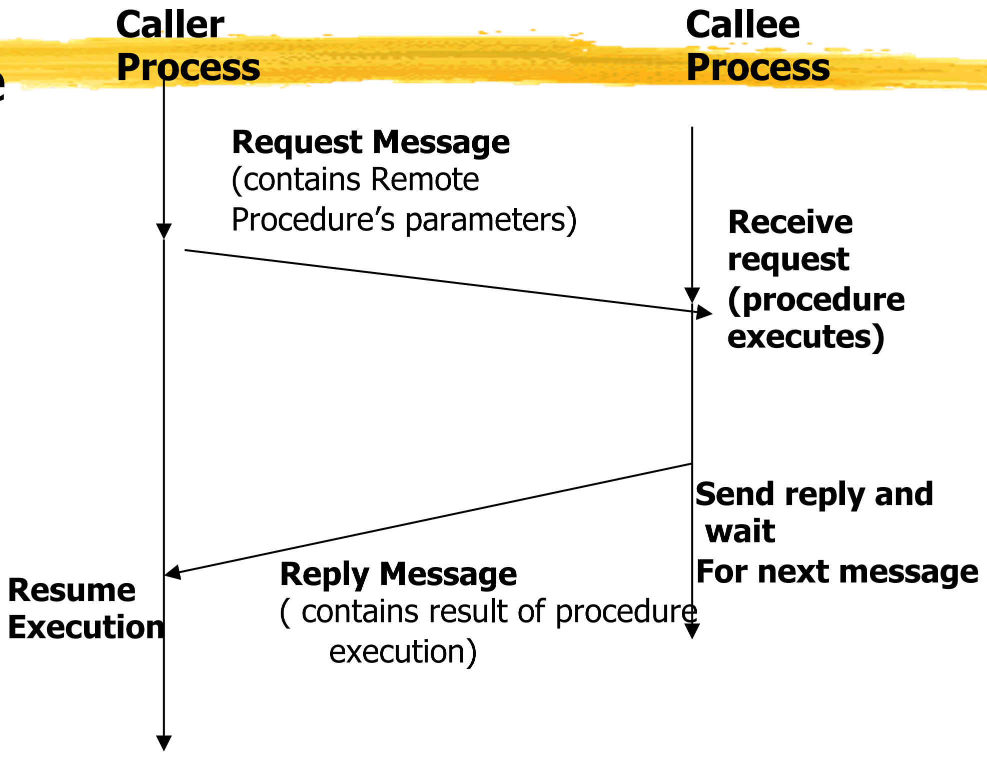
Message-oriented middleware

Aims at high-level persistent asynchronous communication:

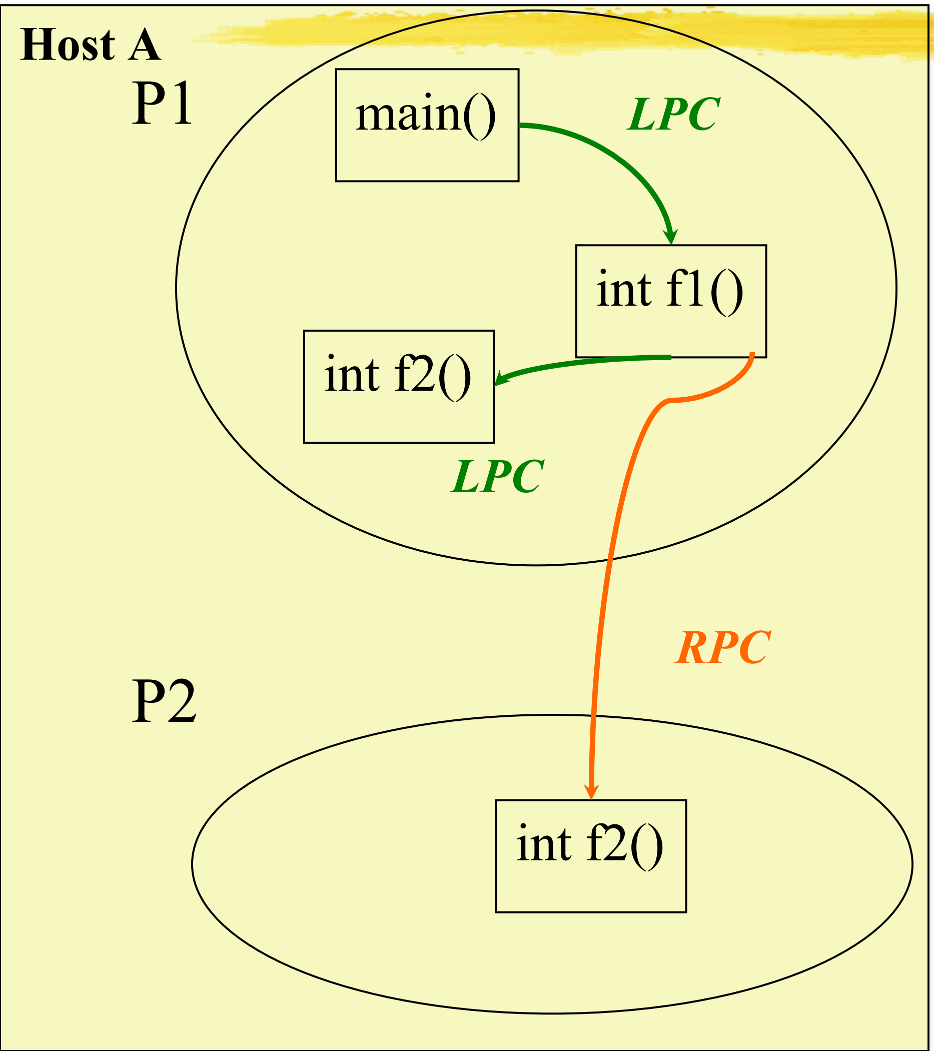
- Processes send each other messages, which are queued
- Sender need not wait for immediate reply, but can do other things
- Middleware often ensures fault tolerance

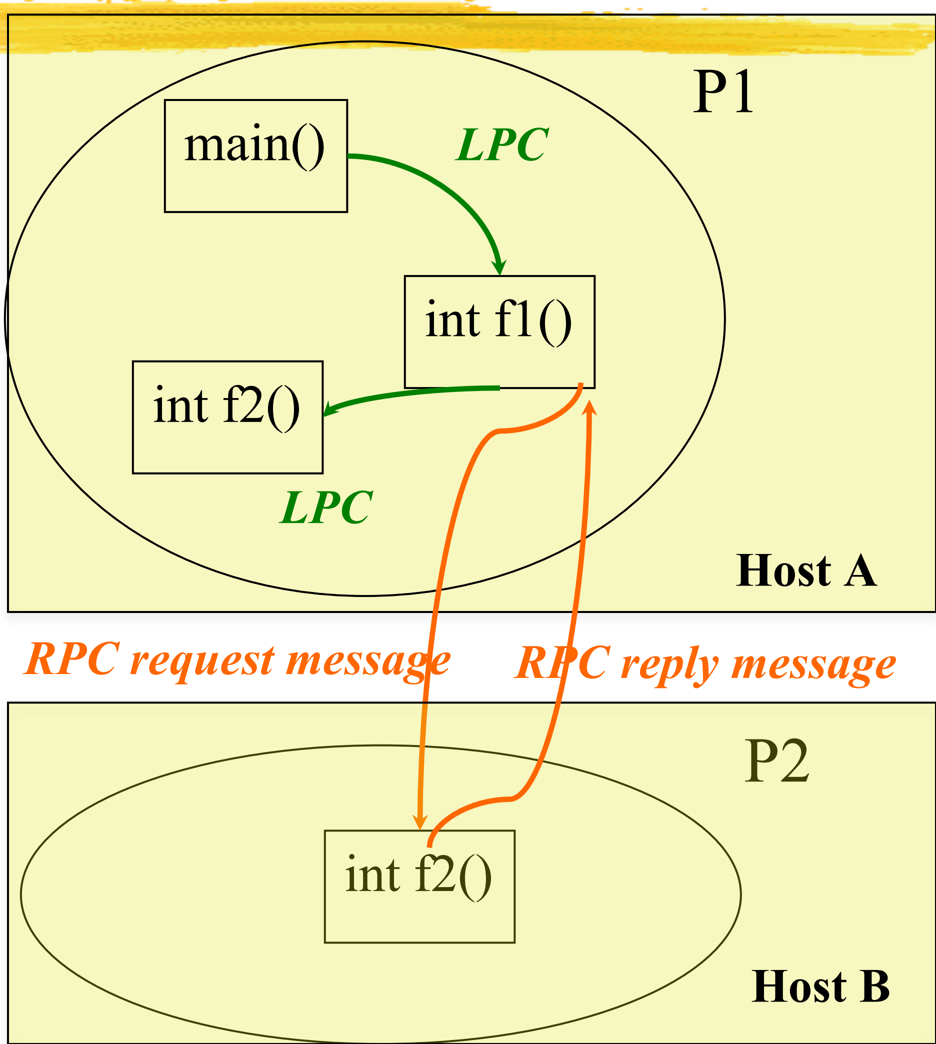
Remote Procedure Calls (RPC)

- Basis of client/server systems
  - 80's - Birrell and Nelson
- General message passing model for execution of remote functionality.
  - Provides programmers with a familiar mechanism for building distributed applications/systems
- Familiar semantics (similar to LPC)
  - Simple syntax, well defined interface, ease of use, generality and IPC between processes on same/different machines.
- It is generally synchronous
- Can be made asynchronous by using multi-threading


Caller Process
Callee Process
Request Message
(contains Remote Procedure's parameters)
Receive request (procedure executes)
Send reply and wait For next message
Reply Message
(contains result of procedure execution)
Resume Execution

LPC vs. RPC


Host A
P1
main()
LPC
int f1()
int f2()
LPC
RPC
P2
int f2()


P1
main()
LPC
int f1()
int f2()
LPC
Host A
RPC request message
RPC reply message
P2
int f2()
Host B

RPC Challenges

Achieving exactly the same semantics for LPC and RPC is hard

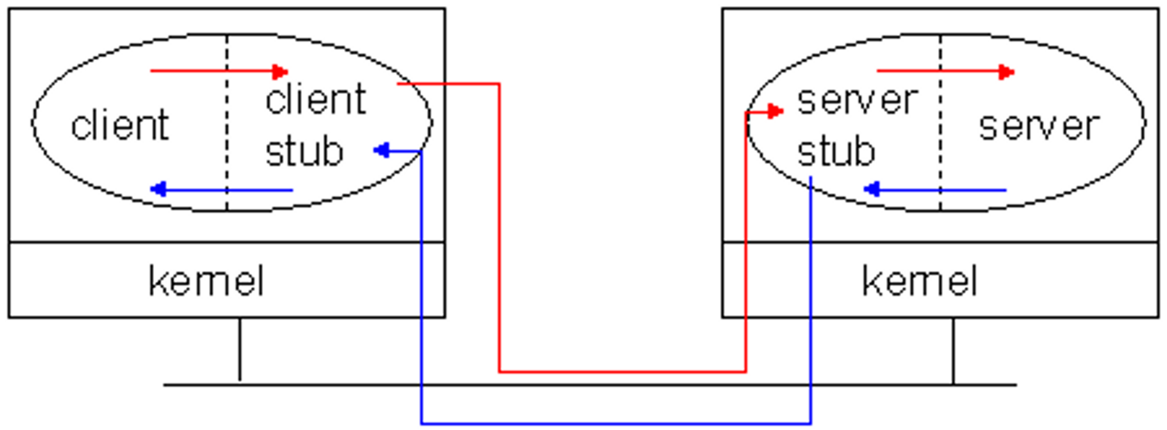
- Disjoint address spaces
- Consumes more time (due to communication delays)
- Failures (hard to guarantee exactly-once semantics)
  - Function may not be executed if
    - Request (call) message is dropped
    - Reply (return) message is dropped
    - Called process fails before executing called function
    - Called process fails after executing called function
    - Hard for caller to distinguish these cases
  - Function may be executed multiple times if request (call) message is duplicated

RPC Needs :Syntactic and Semantic Transparency

- Resolve differences in data representation (CDR)
- Support multi-threaded programming
- Provide good reliability
- Provide independence from transport protocols
- Ensure high degree of security
- Locate required services across networks
- Support a variety of execution semantics
  - At most once semantics (e.g., Java RMI)
  - At least once semantics (e.g., Sun RPC)
  - Maybe, i.e., best-effort (e.g., CORBA)

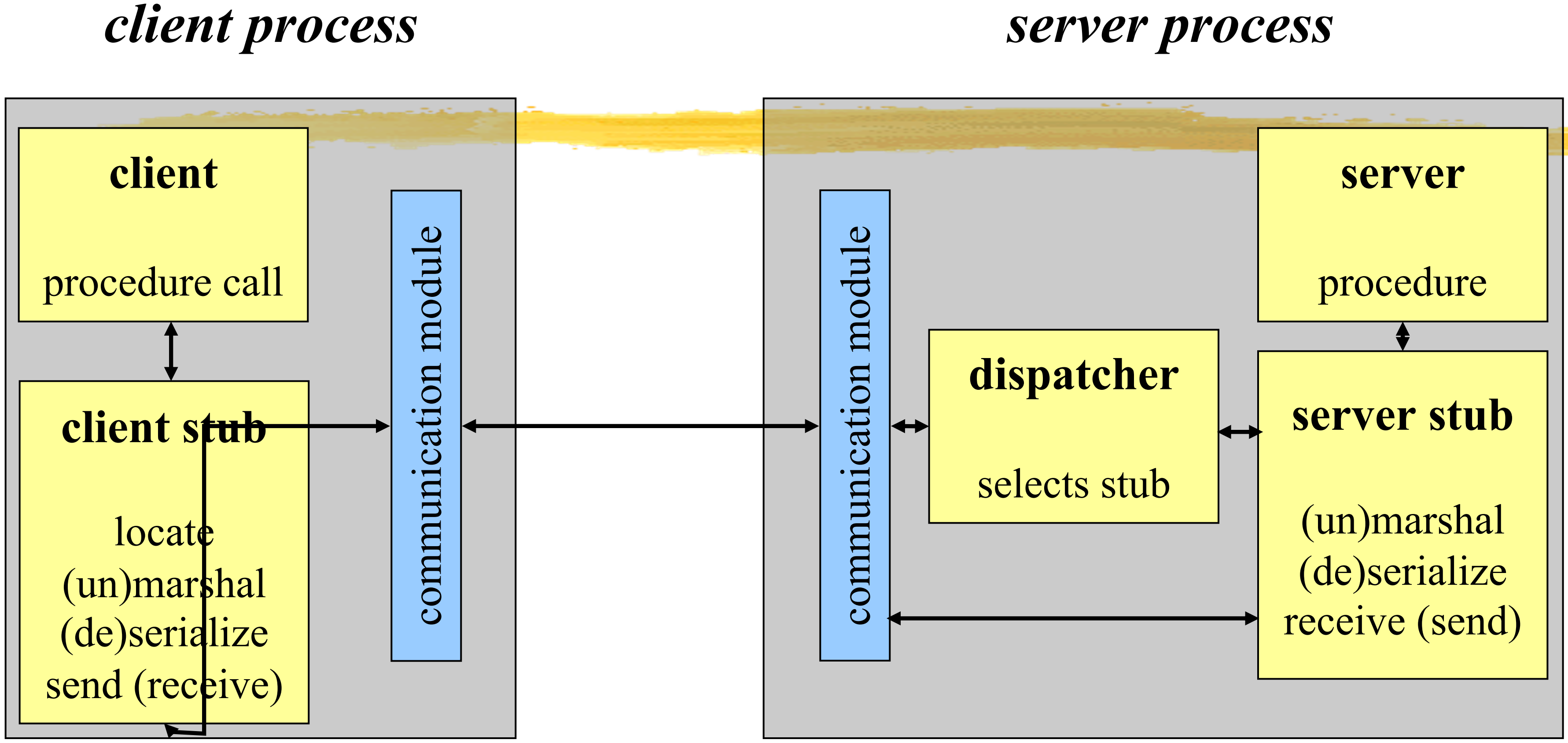
Retransmit request	Filter duplicate requests	Re-execute function or retransmit reply	RPC Semantics
Yes	No	Re-execute	At least once
Yes	Yes	Retransmit	At most once
No	NA	NA	Maybe

Implementing RPC - Mechanism



- Uses the concept of stubs; A perfectly normal LPC abstraction by concealing from programs the interface to the underlying RPC
- Involves the following elements
  - The client, The client stub
  - The RPC runtime
  - The server stub, The server

RPC – How it works II



RPC - Steps

- Client procedure **calls** the client stub in a normal way
- Client stub **builds** a message and **traps** to the kernel
- Kernel **sends** the message to remote kernel
- Remote kernel **gives** the message to server stub
- Server stub **unpacks** parameters and **calls** the server
- Server **computes** results and **returns** it to server stub
- Server stub **packs** results in a message and **traps** to kernel
- Remote kernel **sends** message to client kernel
- Client kernel **gives** message to client stub
- Client stub **unpacks** results and **returns** to client

RPC - Marshalling and Unmarshalling

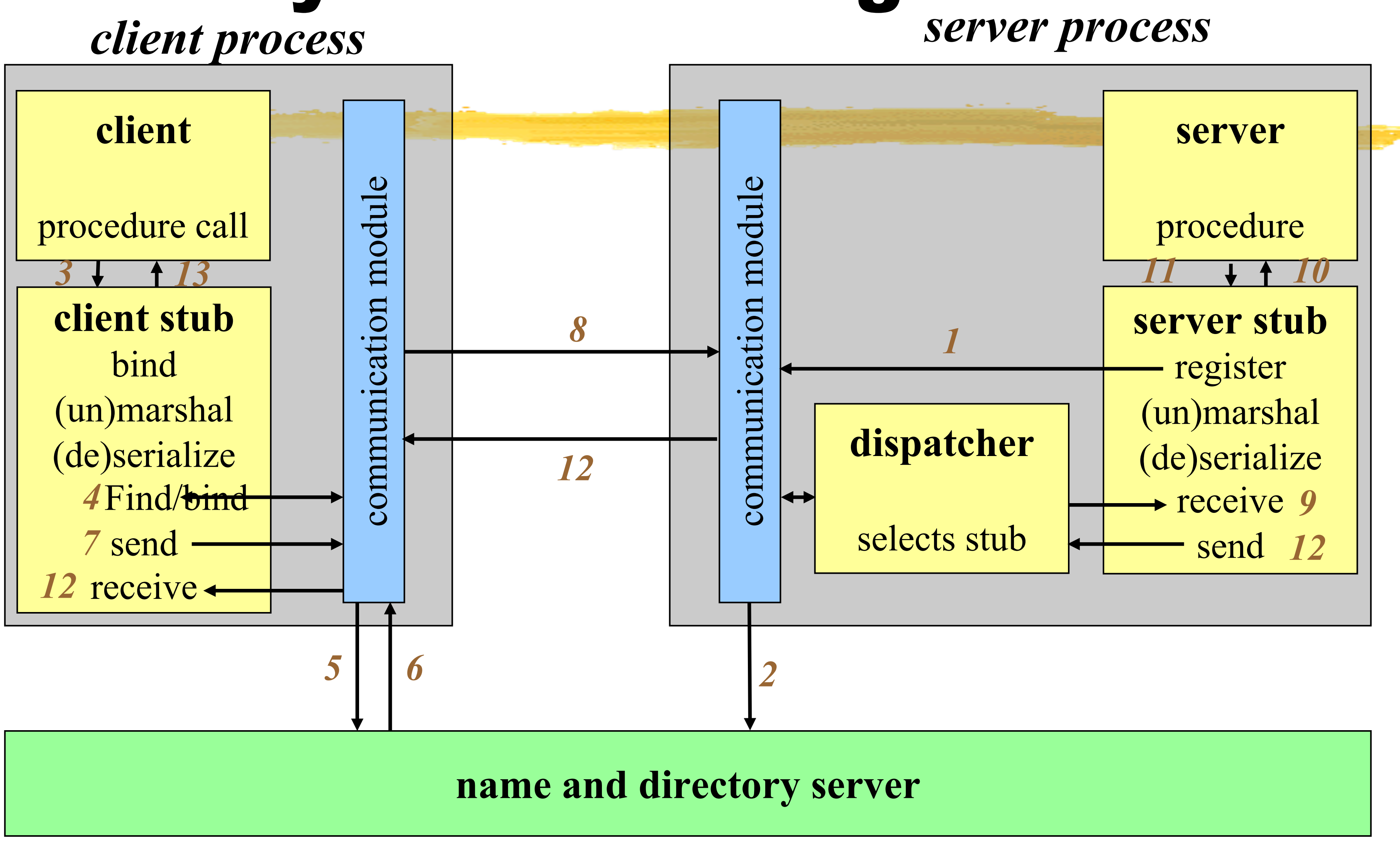
- Different architectures use different ways of representing data
  - **Big endian**: Hex 12-AC-33 stored with 12 in lowest address, then AC in next higher address, then 33 in highest address
    - IBM z, System 360
  - **Little endian**: Hex 12-AC-33 stored with 33 in lowest address, then AC in next higher address, then 12
    - Intel
- Caller (and callee) process uses its own *platform-dependent* way of storing data

- Middleware has a common data representation (CDR)
  - Platform-independent
- Caller process converts arguments into CDR format
  - Called "Marshalling"
- Callee process extracts arguments from message into its own platform-dependent format
  - Called "Unmarshalling"
- Return values are marshalled on callee process and unmarshalled at caller process

RPC - binding

- Static binding
  - hard coded stub
    - Simple, efficient
    - not flexible
  - stub recompilation necessary if the location of the server changes
  - use of redundant servers not possible
- Dynamic binding
  - name and directory server
    - load balancing
  - IDL used for binding
  - flexible
  - redundant servers possible

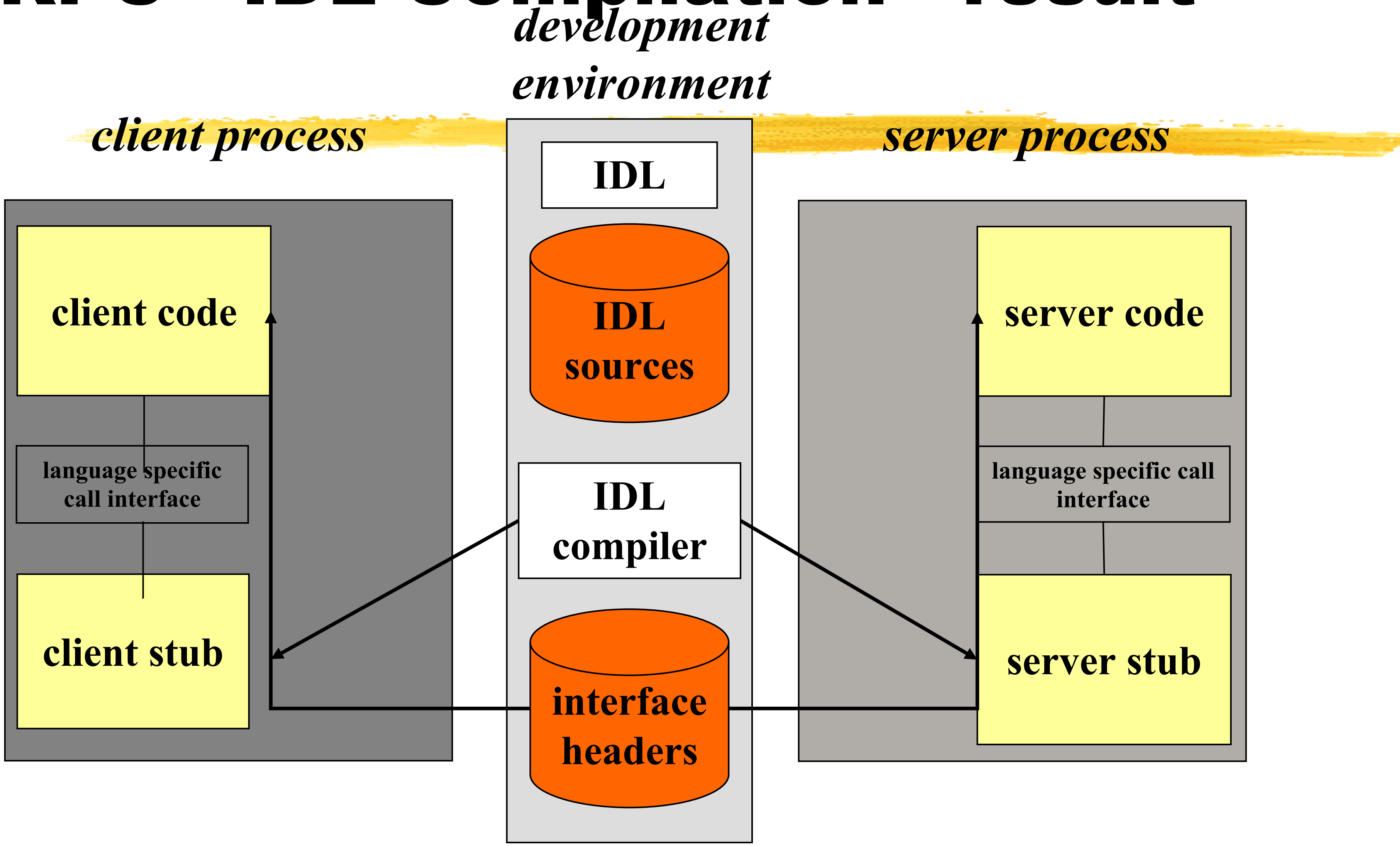
RPC - dynamic binding



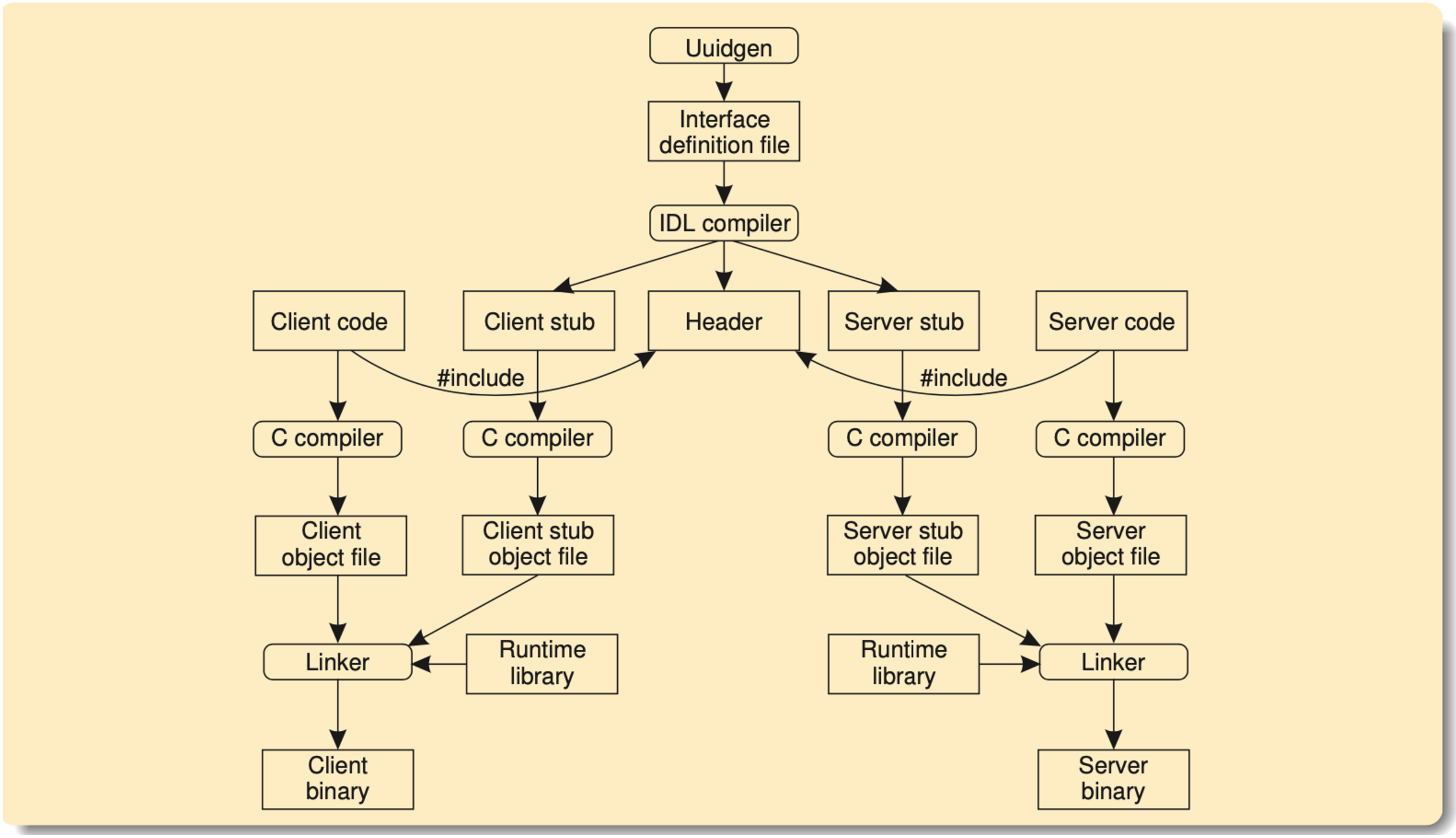
How Stubs are Generated

- Through a compiler
  - e.g. DCE/CORBA IDL – a purely declarative language
  - Defines only types and procedure headers with familiar syntax (usually C)
- It supports
  - Interface definition files (.idl)
  - Attribute configuration files (.acf)
- Uses Familiar programming language data typing
- Extensions for distributed programming are added

RPC - IDL Compilation - result



RPC in Practice...



RPC NG: DCOM & CORBA

- Object models allow services and functionality to be called from distinct processes
- DCOM/COM+(Win2000) and CORBA IIOP extend this to allow calling services and objects on different machines
- More OS features (authentication,resource management,process creation,…) are being moved to distributed objects.

Sample RPC Middleware Products

- JaRPC (NC Laboratories)
  - libraries and development system provides the tools to develop ONC/RPC and extended .rpc Client and Servers in Java
- powerRPC (Netbula)
  - RPC compiler plus a number of library functions. It allows a C/C++ programmer to create powerful ONC RPC compatible client/server and other distributed applications without writing any networking code.
- Oscar Workbench (Premier Software Technologies)
  - An integration tool. OSCAR, the Open Services Catalog and Application Registry is an interface catalog. OSCAR combines tools to blend IT strategies for legacy wrappering with those to exploit new technologies (object oriented, internet).
- NobleNet (Rogue Wave)
  - simplifies the development of **business-critical client/server applications,** and gives developers all the tools needed to distribute these applications across the enterprise. NobleNet RPC automatically generates client/server network code for all program data structures and application programming interfaces (APIs)— reducing development costs and time to market.
- NXTWare TX (eCube Systems)
  - Allows DCE/RPC-based applications to participate in a service-oriented architecture. Now companies can use J2EE, CORBA (IIOP) and SOAP to securely access data and execute transactions from legacy applications. With this product, organizations can leverage their current investment in existing DCE and RPC applications

RPC - Extensions

- conventional RPC: sequential execution of routines
- client blocked until response of server
- asynchronous RPC – non blocking
  - client has two entry points(request and response)
  - server stores result in shared memory
  - client picks it up from there

RPC servers and protocols…

- RPC Messages (call and reply messages)
- Server Implementation
  - Stateful servers
  - Stateless servers
- Communication Protocols
  - Request(R)Protocol
  - Request/Reply(RR) Protocol
  - Request/Reply/Ack(RRA) Protocol

- Idempotent operations - can be repeated multiple times, without any side effects
- Examples (x is server-side variable)
  - x=1;
- Non-examples
  - x=x+1;
  - x=x*2

Some recent views on RPC

- Convenience over correctness?

http://steve.vinoski.net/pdf/IEEE-Convenience_Over_Correctness.pdf

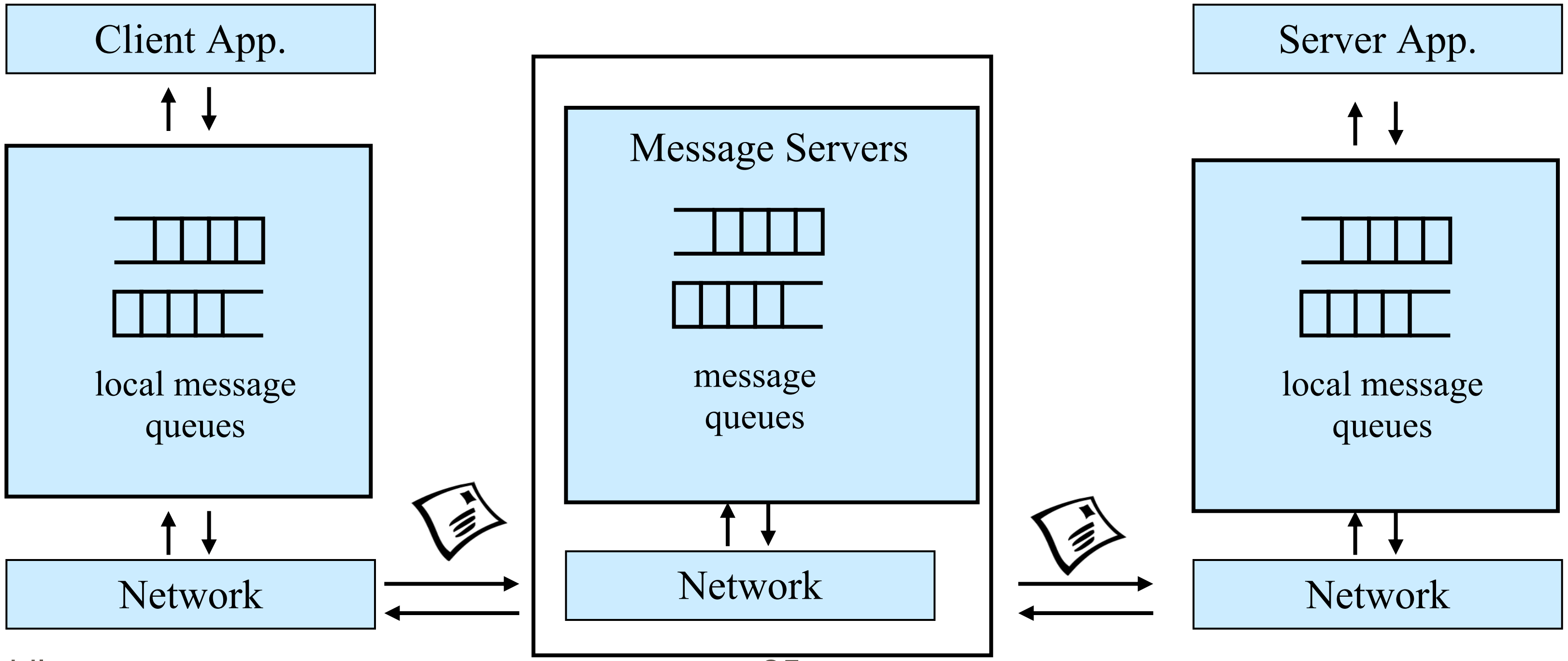
- Time to retire RPC?

Video: Mythbusting Remote Procedure Calls

New messaging formats
-- Google ProtoBuf, Apache Thrift (Facebook)

Message-Oriented Middleware (MOM)

- Software infrastructure to support communication using **messages**
- **Message brokers/ servers** decouple client and server
  - Messages stored in **message queues - asynchronous persistent communication**
- Various assumptions about **message content**
- Developers agnostic to underlying details of OS/network protocols.



cf: www.cl.cam.ac.uk/teaching/0910/ConcDistS/

Properties of MOM

Asynchronous interaction

- Client and server are only **loosely coupled**
- Messages are queued
- Good for application integration

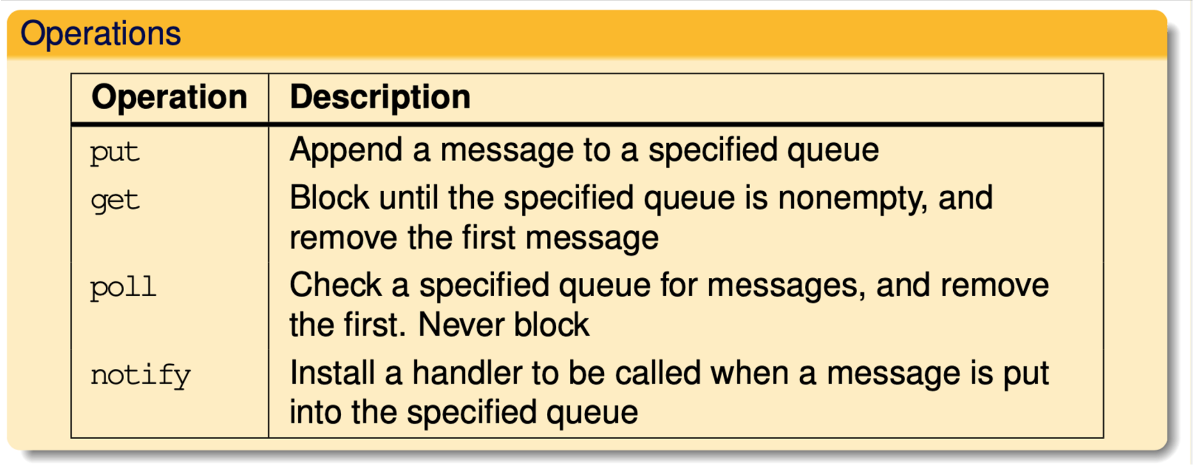
Support for **reliable** delivery service

- Keep queues in persistent storage

Processing of messages by intermediate message server(s)

- May do filtering, transforming, logging, …

Natural for database integration



Operations

Operation	Description
`put`	Append a message to a specified queue
`get`	Block until the specified queue is nonempty, and remove the first message
`poll`	Check a specified queue for messages, and remove the first. Never block
`notify`	Install a handler to be called when a message is put into the specified queue

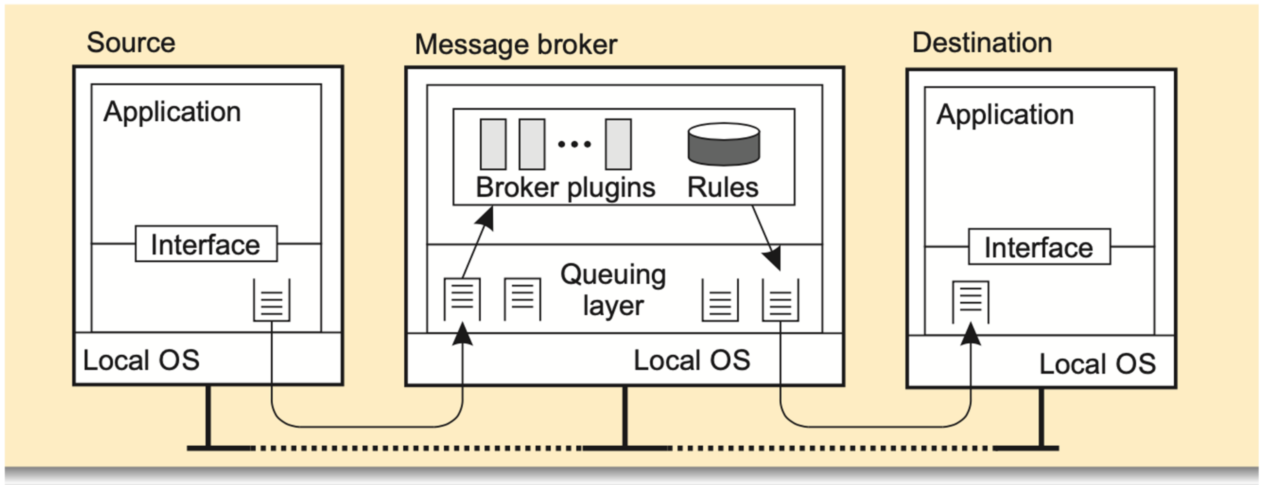
Introducing message brokers

Observation

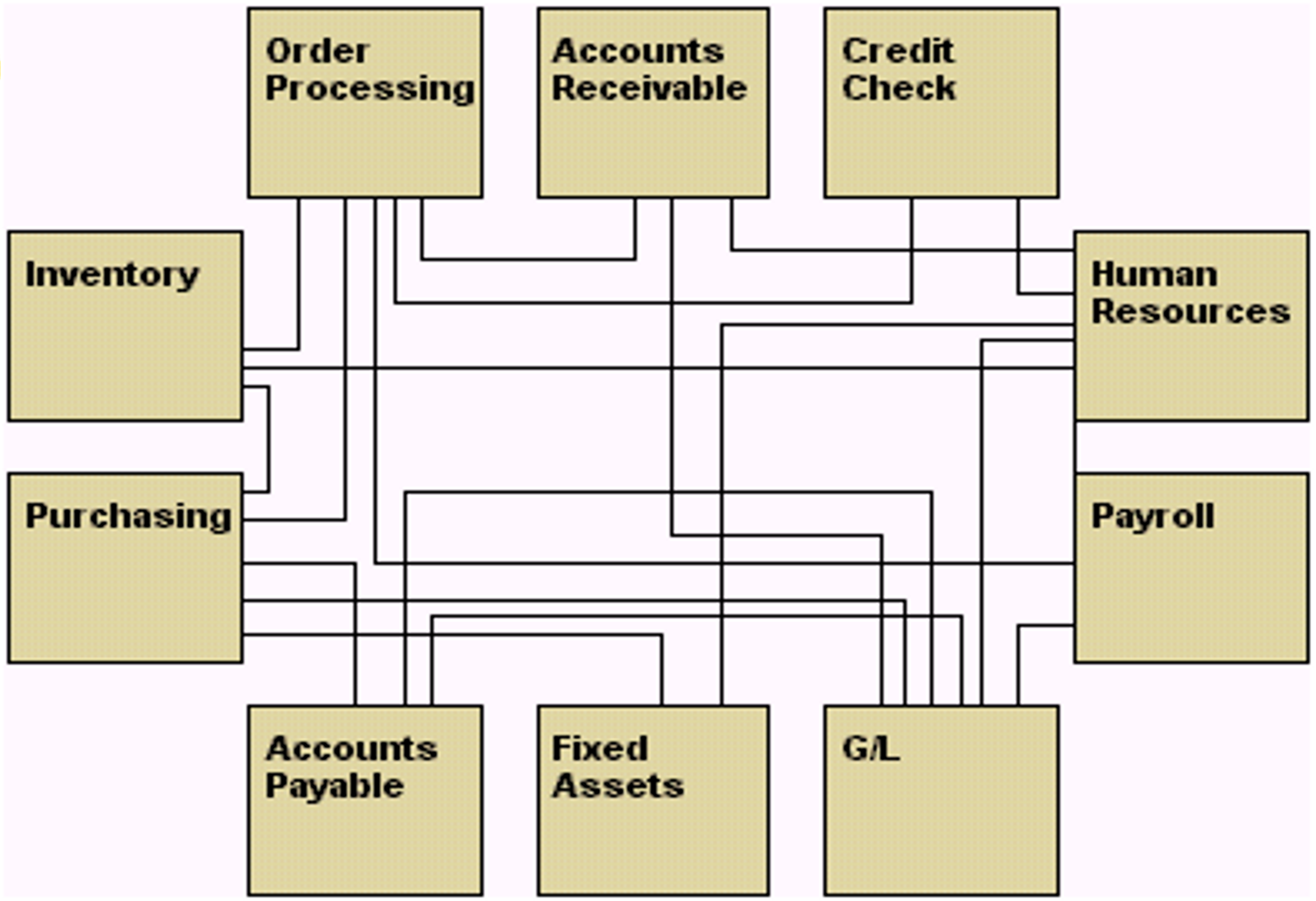
Message queuing systems assume a common messaging protocol: all applications agree on message format (i.e., structure and data representation)

Broker handles application heterogeneity in an MQ system

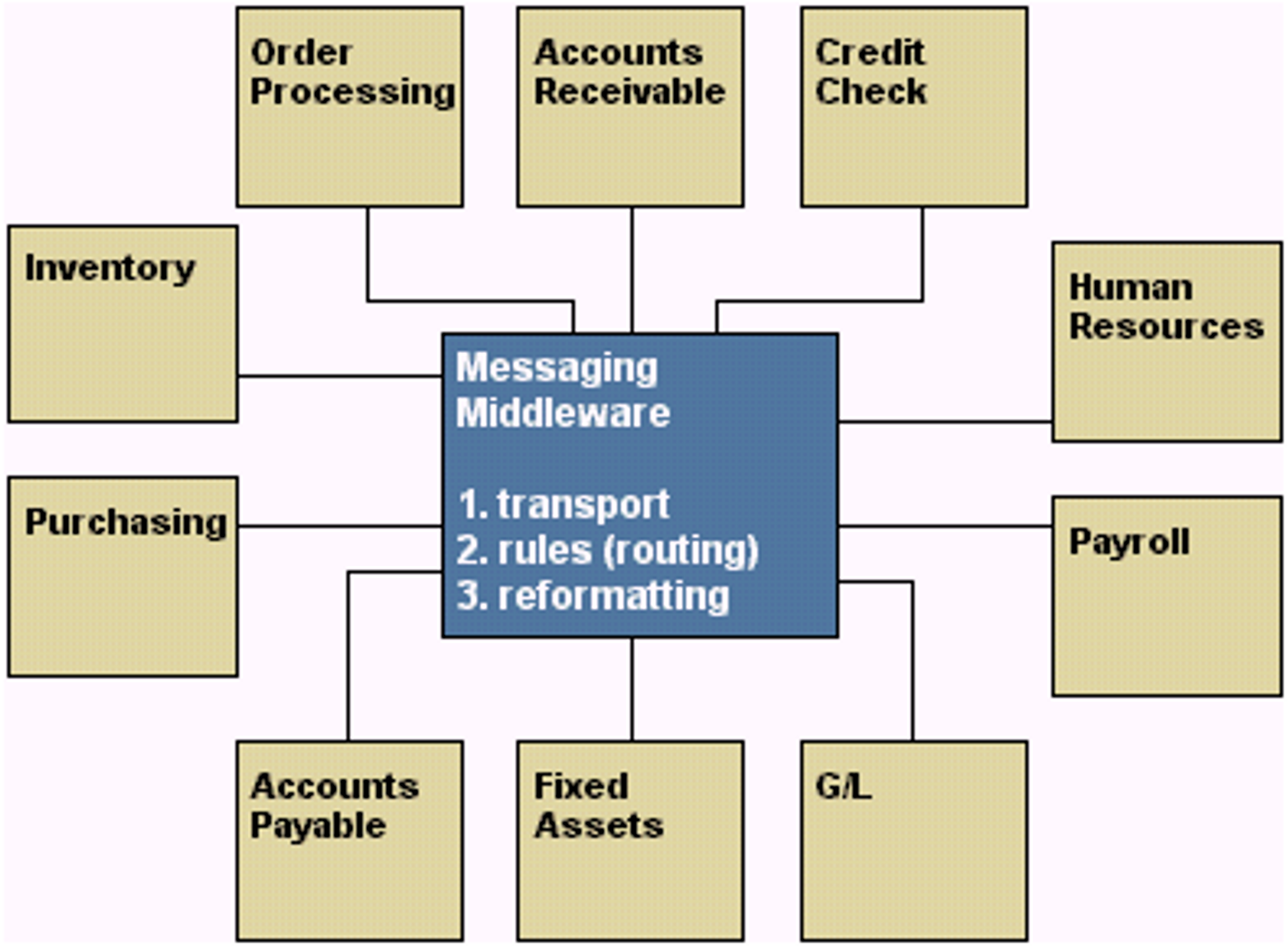
- Transforms incoming messages to target format
- Very often acts as an application gateway
- May provide subject-based routing capabilities (i.e., publish-subscribe capabilities)



From Computer Desktop Encyclopedia
© 2000 The Computer Language Co. Inc.



From Computer Desktop Encyclopedia
© 2000 The Computer Language Co. Inc.



Message-Oriented Middleware Message Brokers

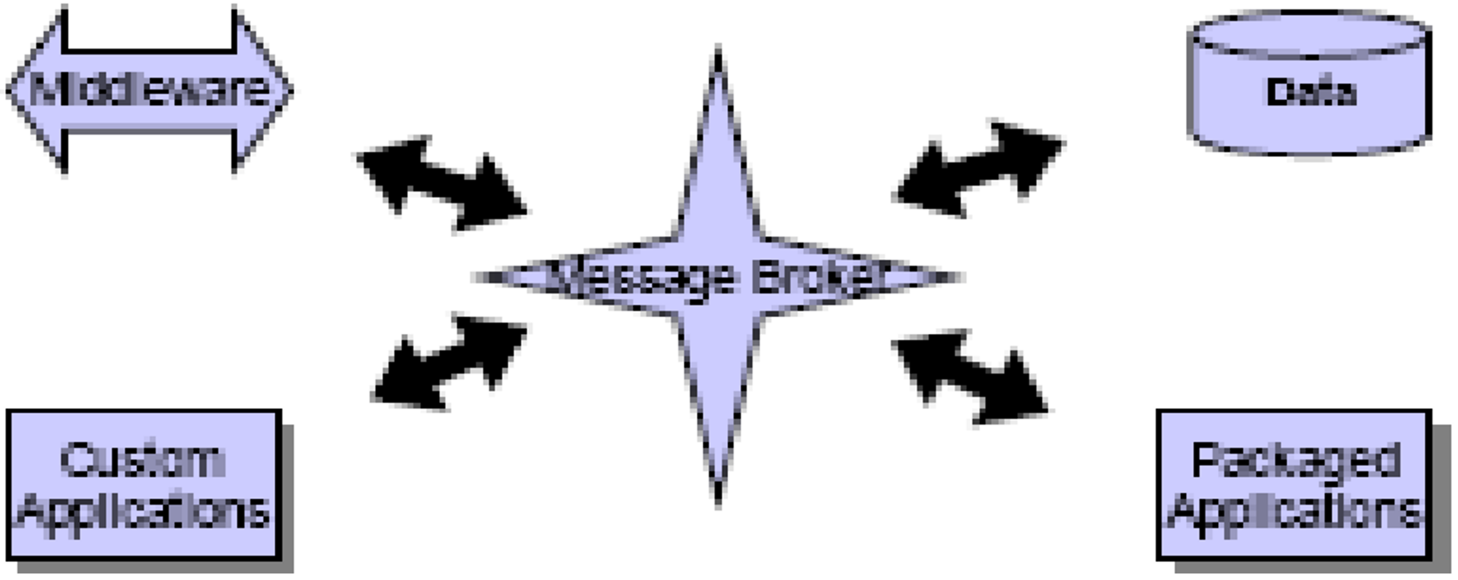
- A message broker is a software system based on asynchronous, store-and-forward messaging.
- It manages interactions between applications and other information resources, utilizing abstraction techniques.
- Simple operation: an application puts (publishes) a message to the broker, another application gets (subscribes to) the message. The applications do not need to be session connected.

(Message Brokers, MQ)

• MQ is fairly fault tolerant in the cases of network or system failure.
• Most MQ software lets the message be declared as persistent or stored to disk during a commit at certain intervals. This allows for recovery on such situations.
• Each MQ product implements the notion of messaging in its own way.
• Widely used commercial examples include IBM's MQSeries and Microsoft's MSMQ.

Advantages of Message Brokers

- They leave systems "where they are" still allowing data to be shared.
- Greater likelihood to be able to automate some manual processes.



Message Brokers

- **Any-to-any**

The ability to connect diverse applications and other information resources

– The consistency of the approach

– Common look-and-feel of all connected resources

- **Many-to-many**

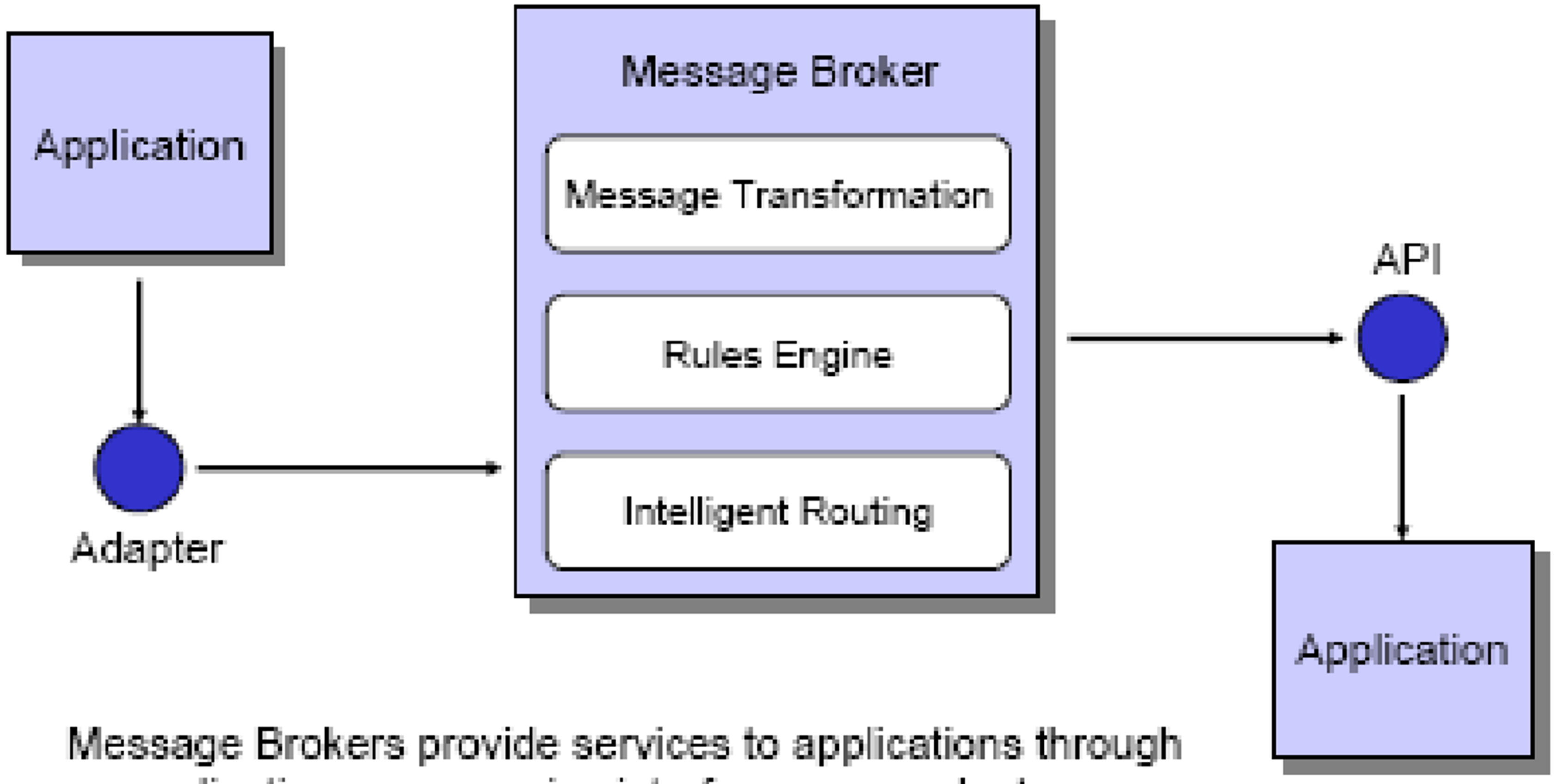
– Once a resource is connected and publishing information, the information is easily reusable by any other application that requires it.

Standard Features of Message Brokers

- Message transformation engines
  - Allow the message broker to alter the way information is presented for each application.
- Intelligent routing capabilities
  - Ability to identify a message, and an ability to route them to appropriate location.
- Rules processing capabilities

– Ability to apply rules to the transformation and routing of information.

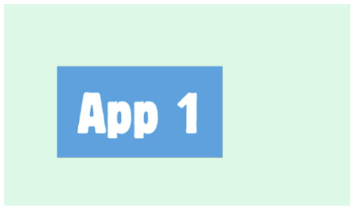
Providing the Services

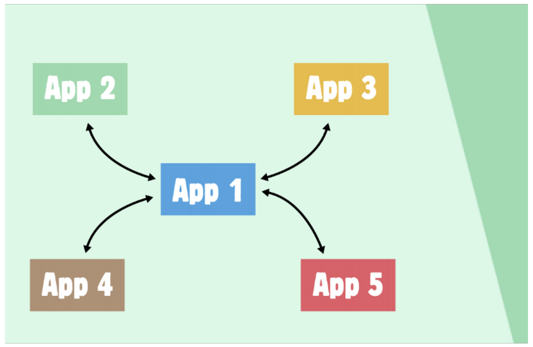


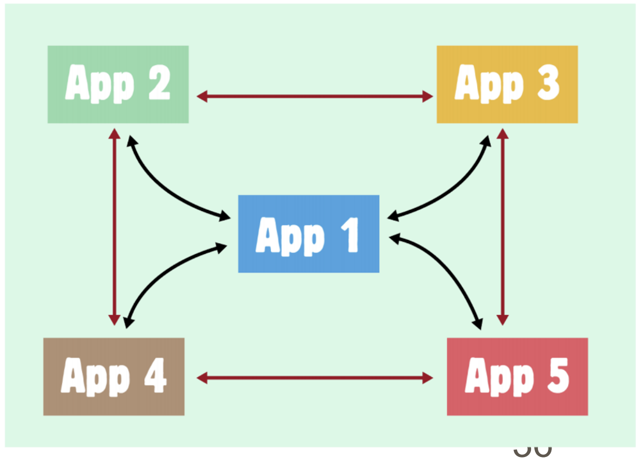
Message Brokers provide services to applications through an application programming interface, or an adapter.

Enterprise Service Buses

- MOMs -> Message brokers -> ESB
- ESB (Enterprise Service Buses)
  - Wikipedia
    - " a software architecture model used for designing and implementing communication between mutually interacting software applications in a service-oriented architecture (SOA)"
  - abstraction layer on top of a messaging system







Vendors

- Adea Solutions[2]: Adea ESB Framework
- ServiceMix[3]: ServiceMix (Apache) Synapse (Apache Incubator)
- BEA: AquaLogic Service Bus
- BIE: Business integration Engine
- Cape Clear Software: Cape Clear 6
- Cordys ESB
- Fiorano Software Inc.Fiorano Software Inc. Fiorano ESB™ 2006
- IBMWebSphere Platform (specifically WebSphere Message Broker or WebSphere ESB)
- IONA Technologies Artix
- iWay Software: iWay Adaptive Framework for SOA
- Microsoft .NET
- Microsoft BizTalk Server
- ObjectWeb
- Celtix (Open Source, LGPL)
- Oracle: Oracle Integration products
- Petals Services Platform: EBM WebSourcing & Fossil E-Commerce (Open Source)
- PolarLake: Integration Suite
- LogicBlaze: ServiceMix ESB (Open Source, Apache Lic.)
- Sonic Software: Sonic ESB
- SymphonySoft Mule (Open Source)
- TIBCO Software
- Virtuoso Universal Server
- webMethods: webMethods Fabric

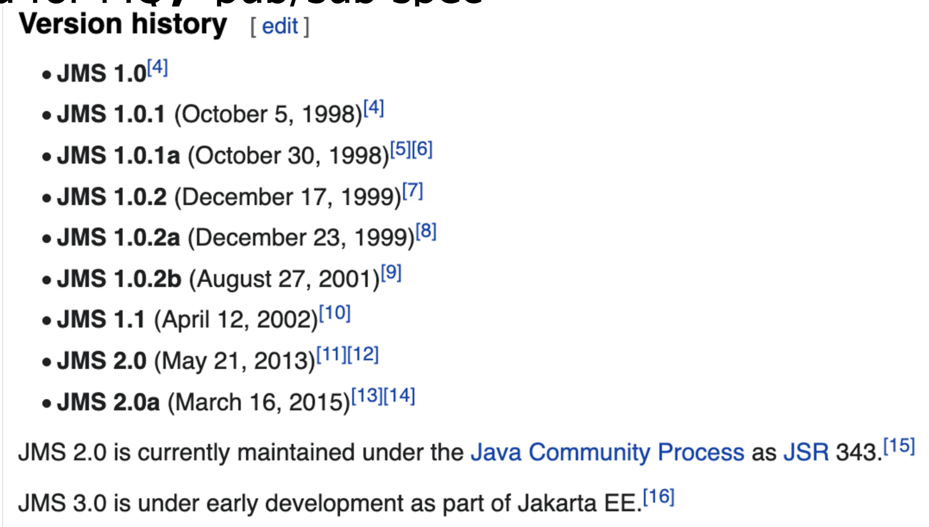
IBM MQSeries

- One-to-one reliable message passing using queues
  - Persistent and non-persistent messages
  - Message priorities, message notification
- **Queue Managers**
  - Responsible for queues
  - Transfer messages from input to output queues
  - Keep routing tables
- **Message Channels**
  - Reliable connections between queue managers
- Messaging API:

MQopen	Open a queue
MQclose	Close a queue
MQput	Put message into opened queue
MQget	Get message from local queue

Java Message Service (JMS)

- Java API **specification** to access MOM implementations
- Two modes of operation *specified*:
  - **Point-to-point** one-to-one communication using queues
  - **Publish/Subscribe** Event-Based Middleware
- **JMS Server** implements JMS API, JMS Clients connect to JMS servers
- Java objects can be serialised to JMS messages
- A JMS interface has been provided for MQ, pub/sub spec



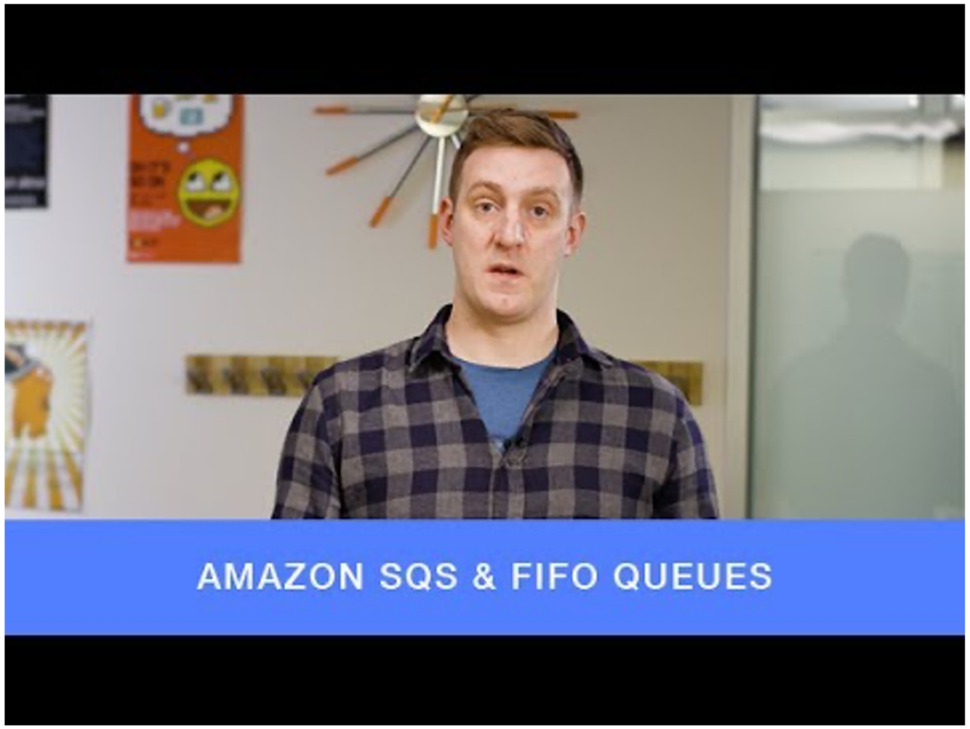
Version history [edit]

- **JMS 1.0**[4]
- **JMS 1.0.1** (October 5, 1998)[4]
- **JMS 1.0.1a** (October 30, 1998)[5][6]
- **JMS 1.0.2** (December 17, 1999)[7]
- **JMS 1.0.2a** (December 23, 1999)[8]
- **JMS 1.0.2b** (August 27, 2001)[9]
- **JMS 1.1** (April 12, 2002)[10]
- **JMS 2.0** (May 21, 2013)[11][12]
- **JMS 2.0a** (March 16, 2015)[13][14]

JMS 2.0 is currently maintained under the Java Community Process as JSR 343.[15]

JMS 3.0 is under early development as part of Jakarta EE.[16]

Amazon Simple Queue Service



https://www.youtube.com/watch?v=XrX7rb6M3jw

Protocol Buffer

lóhika

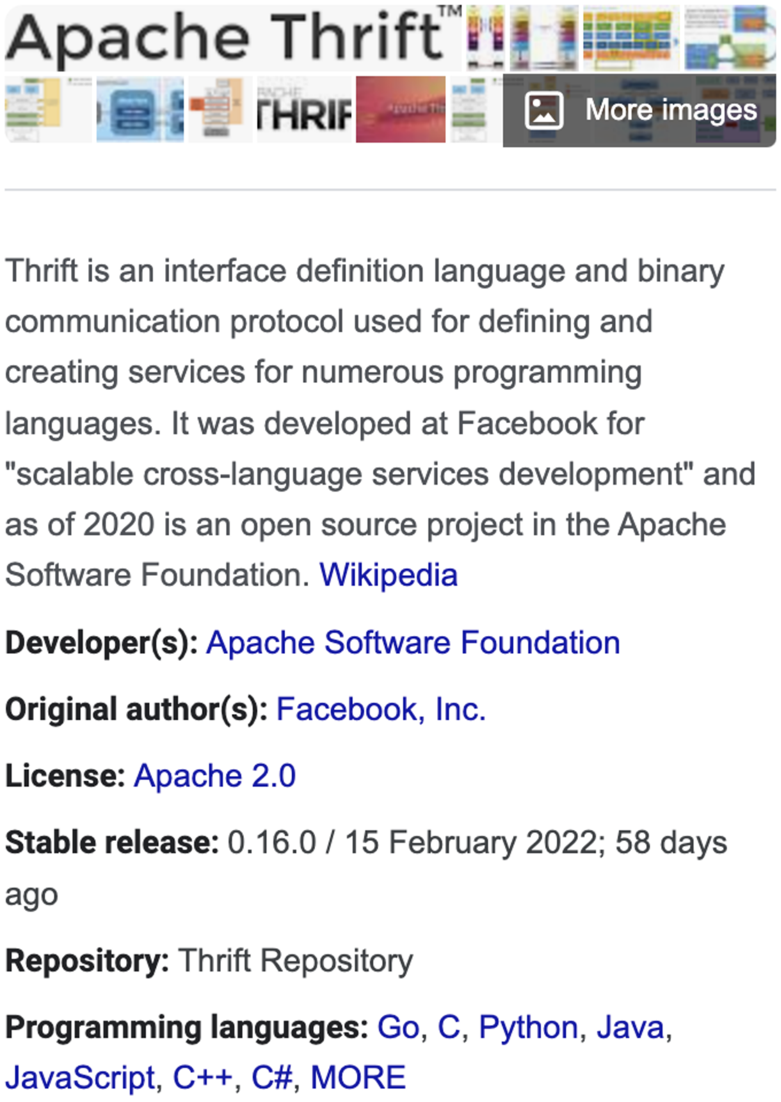
- Designed ~2001 because everything else wasn't that good those days
- Production, proprietary in Google from 2001-2008, open-sourced since 2008
- Battle tested, very stable, well trusted
- Every time you hit a Google page, you're hitting several services and several PB code
- PB is the glue to all Google services
- Official support for four languages: C++, Java, Python, and JavaScript
- Does have a lot of third party support for other languages (of highly variable quality)
- Current Version - 3.20.0 / 25 March 2022; 20 days ago[2]
- BSD License

protobuf

Protocol Buffers - Google's data interchange format

Apache Thrift

- Designed by an X-Googler in 2007
- Developed internally at Facebook, used extensively there
- An open Apache project, hosted in Apache's Inkubator.
- Aims to be the next-generation PB (e.g. more comprehensive featur languages)
- IDL syntax is slightly cleaner than PB. If you know one, then you kno
- Supports: C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages
- Offers a stack for RPC calls



Thrift is an interface definition language and binary communication protocol used for defining and creating services for numerous programming languages. It was developed at Facebook for "scalable cross-language services development" and as of 2020 is an open source project in the Apache Software Foundation. Wikipedia

Developer(s): Apache Software Foundation

Original author(s): Facebook, Inc.

License: Apache 2.0

Stable release: 0.16.0 / 15 February 2022; 58 days ago

Repository: Thrift Repository

Programming languages: Go, C, Python, Java, JavaScript, C++, C#, MORE

Comparing messaging formats

Disadvantages of MOM

- Poor programming abstraction (but has evolved)
  - Rather low-level
  - Request/reply more difficult to achieve, but can be done
- Message formats originally unknown to middleware
  - No type checking (JMS addresses this – implementation?)
- Queue abstraction only gives one-to-one communication
  - Limits scalability (JMS pub/sub – implementation?)

Generalizing communication

- Publish-subscribe systems
  - A form of asynchronous messaging
- Group communication
  - Synchrony of messaging is a critical issue

Publish/Subscribe Communication in distributed systems

Nalini Venkatasubramanian

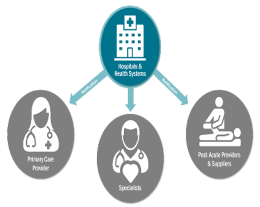
(with slides from Roberto Baldoni, Pascal Felber, Arno Jacobsen, Hojjat Jafarpour etc.)

Notification systems are prevalent

Wide Range of Applications

Communication, Administration, Regulatory Compliance

SCHOOLMESSENGER®



Socialization and Entertainment

NETFLIX

YouTube

Alert, Rescue and Relief operation

WIRELESS EMERGENCY ALERTS

EAS Emergency Alert System

zotALERT get emergency text messages

Transportation,Delivery Service

BUS BULLETIN

Uber

Uber Eats

FedEx®

lyft

Situational Awareness

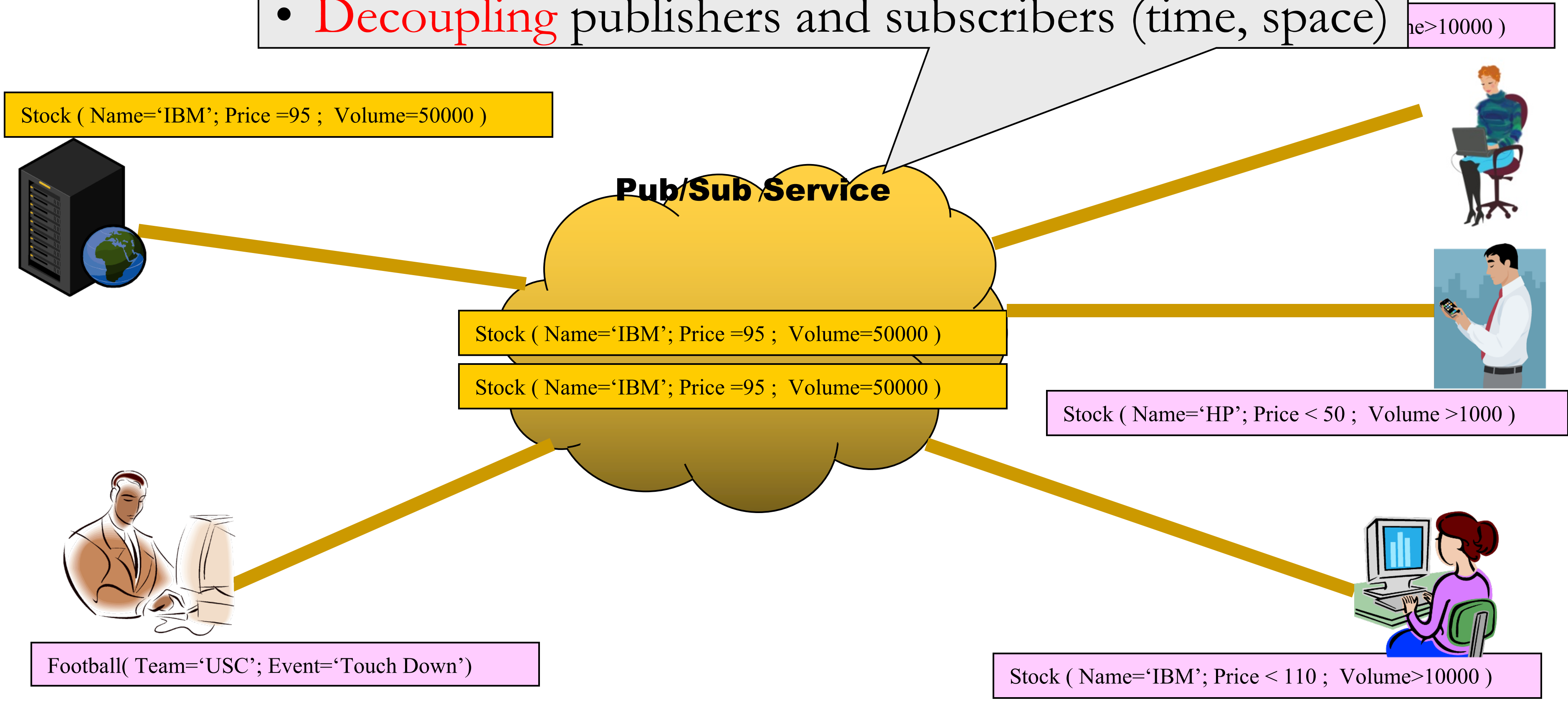
Traffic Notification

At the heart of several such systems are publish subscribe architectures

Publish/Subscribe (pub/sub) systems

- What i

- Asynchronous communication
- Selective dissemination
- Push model, physical separation
- Decoupling publishers and subscribers (time, space)



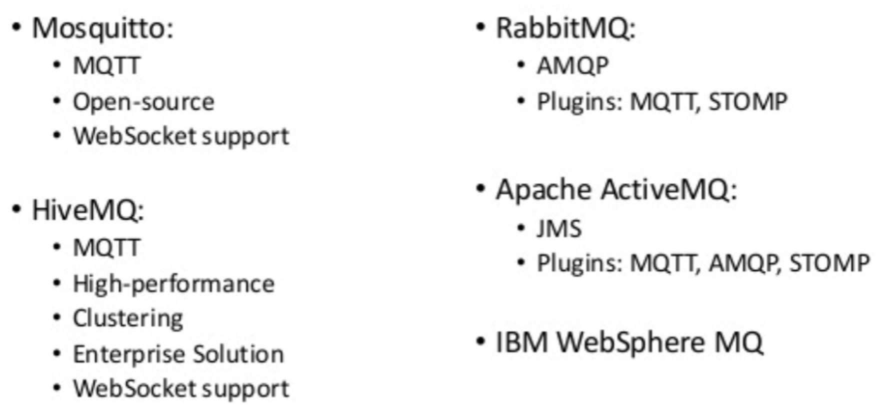
Stock (Name='IBM'; Price =95 ; Volume=50000)
Pub/Sub Service
Stock (Name='IBM'; Price =95 ; Volume=50000)
Stock (Name='IBM'; Price =95 ; Volume=50000)
ne>10000)
Stock (Name='HP'; Price < 50 ; Volume >1000)
Football(Team='USC'; Event='Touch Down')
Stock (Name='IBM'; Price < 110 ; Volume>10000)

Publish/Subscribe (pub/sub) systems

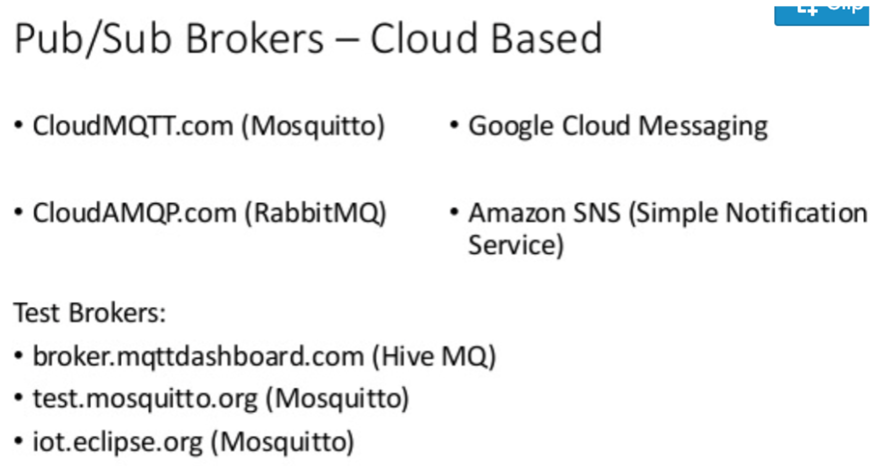
- Applications:
  - Alerting Services
  - Online stock quotes
  - Internet games
  - Sensor networks
  - Location-based services
  - Network management
  - Internet auctions
  - Surveillance and monitoring
  - Business process Management
  - IoT workflows

Sample Pub-Sub Protocols/Brokers

AMQP
XMPP
ZMQ
STOMP
OPC-UA
WAMP
socket.io
JMS
PubSubHubbub
CORBA
AmazonSQS
DCOM
DDS
MQTT



- Mosquitto:
  - MQTT
  - Open-source
  - WebSocket support
- HiveMQ:
  - MQTT
  - High-performance
  - Clustering
  - Enterprise Solution
  - WebSocket support
- RabbitMQ:
  - AMQP
  - Plugins: MQTT, STOMP
- Apache ActiveMQ:
  - JMS
  - Plugins: MQTT, AMQP, STOMP
- IBM WebSphere MQ

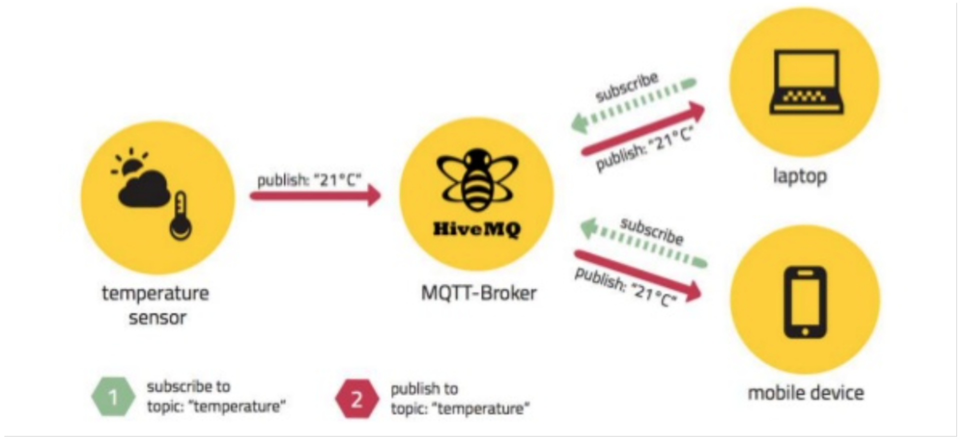


Pub/Sub Brokers – Cloud Based

- CloudMQTT.com (Mosquitto)
- CloudAMQP.com (RabbitMQ)
- Google Cloud Messaging
- Amazon SNS (Simple Notification Service)

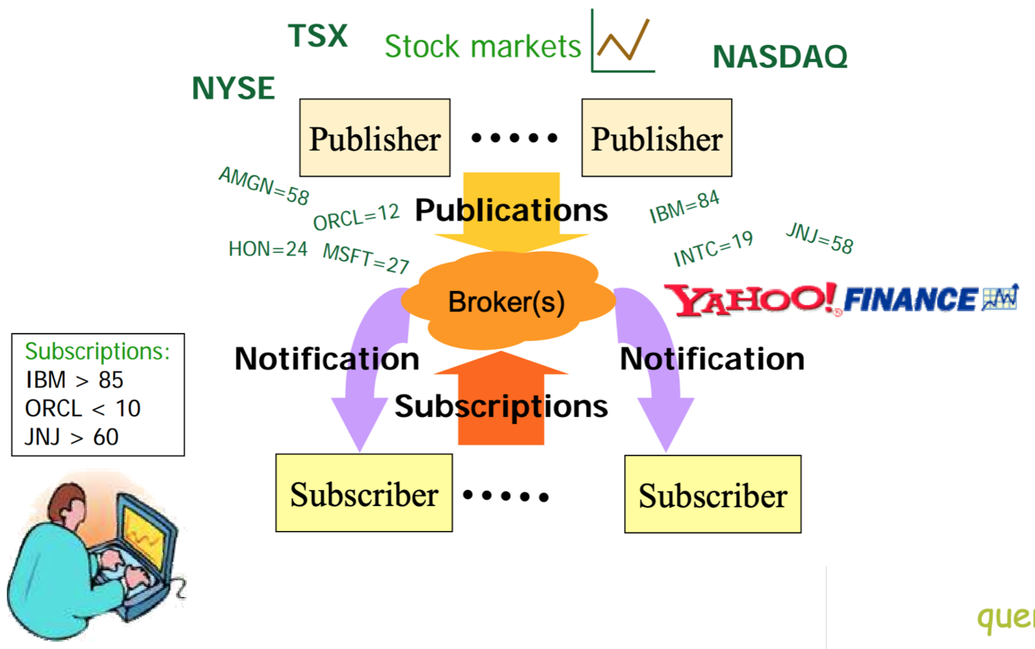
Test Brokers:

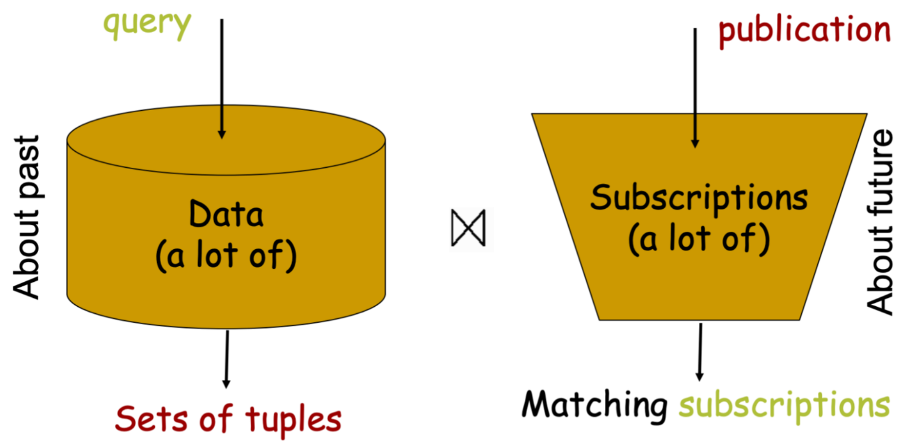
- broker.mqttdashboard.com (Hive MQ)
- test.mosquitto.org (Mosquitto)
- iot.eclipse.org (Mosquitto)



https://www.slideshare.net/PeterHanzlik

Pub/Sub vs. Database Queries





Query and ***subscription*** is very similar.
Set of tuples and ***publication*** is very similar.

MIDDLEWARE SYSTEMS
RESEARCH GROUP

The Pub/Sub Matching Problem

- Given a set of subscriptions, *S*, and a publication, *e*, return all *s* in *S* matched by *e*.
- *e* is referred to as **event** or **publication**
- Splitting hairs
  - *Event* is a state transition of interest in the environment
  - *Publication* is the information about *e* submitted to the publish/subscribe system
- Simple problem statement, widely applicable, and lots of open questions

Matching Problem - Dimensions

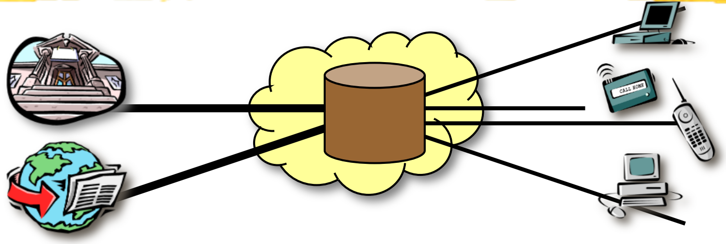
- Text / search strings (information filtering)
- Semi-structured data / queries
  - **attribute-value pairs / attribute-operator-value-predicates**
  - XML, HTML
- Tree-structured data / path expressions
  - XML ./ XPath expressions
- Graph-structured data / graph queries
  - RDF / RDF queries (e.g., SPARQL)
- Regular languages / regular expressions
- Tables / SQL queries

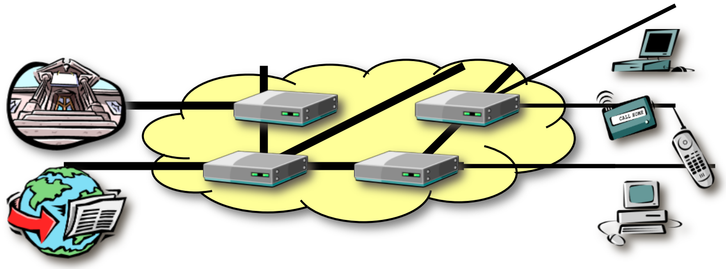
- Different matching semantics
  - **Crisp**
  - Approximate,
  - Similar
  - n-of-m (n of m predicates match)
  - Probability of match

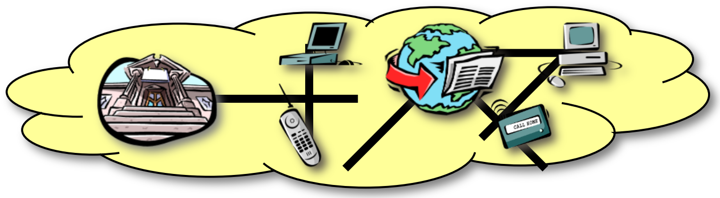
- Centralized and **distributed** instantiation
- Networking architecture
  - Internet (as overlay network)
  - Peer-to-peer style interface (DHT, table-lookup)
  - With mobile publishers, subscribers, brokers
  - Ad hoc network

Publish/subscribe architectures

- **Centralized**
  - Single matching engine
  - Limited scalability
- **Broker overlay**
  - Multiple P/S brokers
  - Participants connected to some broker
  - Events routed through overlay
- **Peer-to-peer**
  - Publishers & subscribers connected in P2P network
  - Participants collectively filter/route events, can be both producer & consumer
  - …….







Types of pub/sub

Two main forms

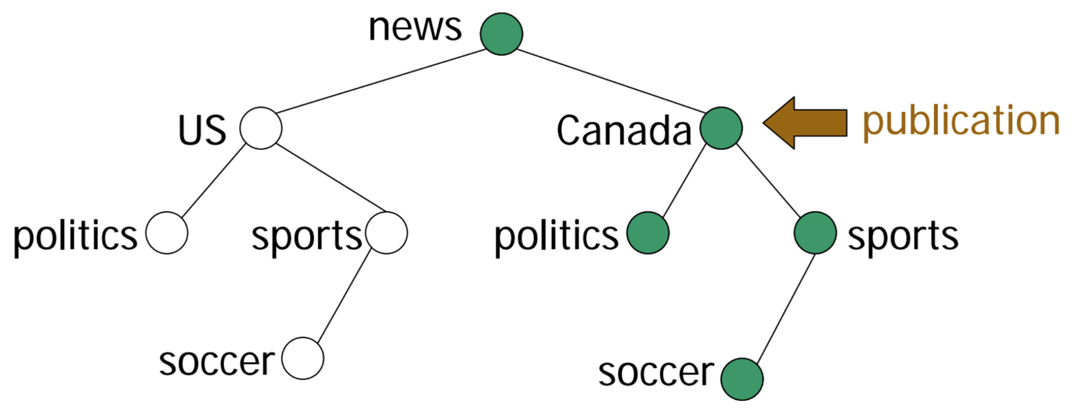
- Topic-Based Publish Subscribe [Oki73]
  - events are divided in topics
  - subscribers subscribe for a single topic
- Content-Based Publish Subscribe [Carz2001]
  - subscriptions are generic SQL-like queries on the event schema

Other forms

- Channel Based Pub/sub
- Type based pub/sub - type as a discriminating attribute, notifications are objects

Topic-Based Pub/Sub

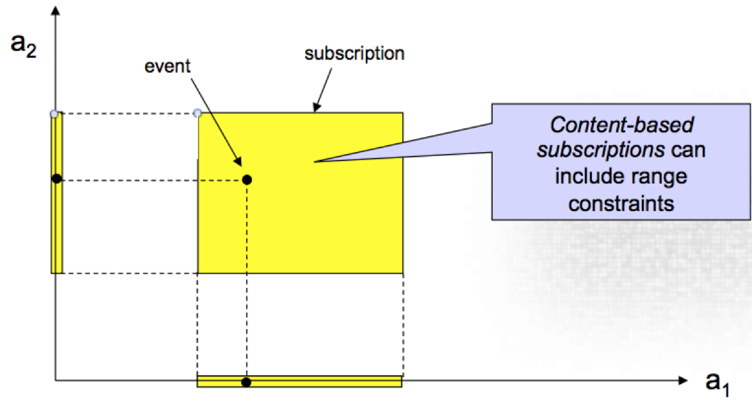
- Event space is divided into topics, corresponding to logical channels
- Topics may be organized in a tree/hierarchy
- Participants subscribe for a topic and publish on a topic
- Receivers for an event are known a priori
- Channel = Group
  - can exploit network-level multicast, group communication
- Limited expressiveness



Content-Based Pub/Sub

- Cannot determine recipients before publication occurs, Receivers calculated for each event being published
- Flexible, general
- Difficult to implement

- Subscriptions and events defined over an n-dimensional *event space* (E.g. `StockName = "ACME" and change < -3)`
  - Subscription: conjunction of constraints

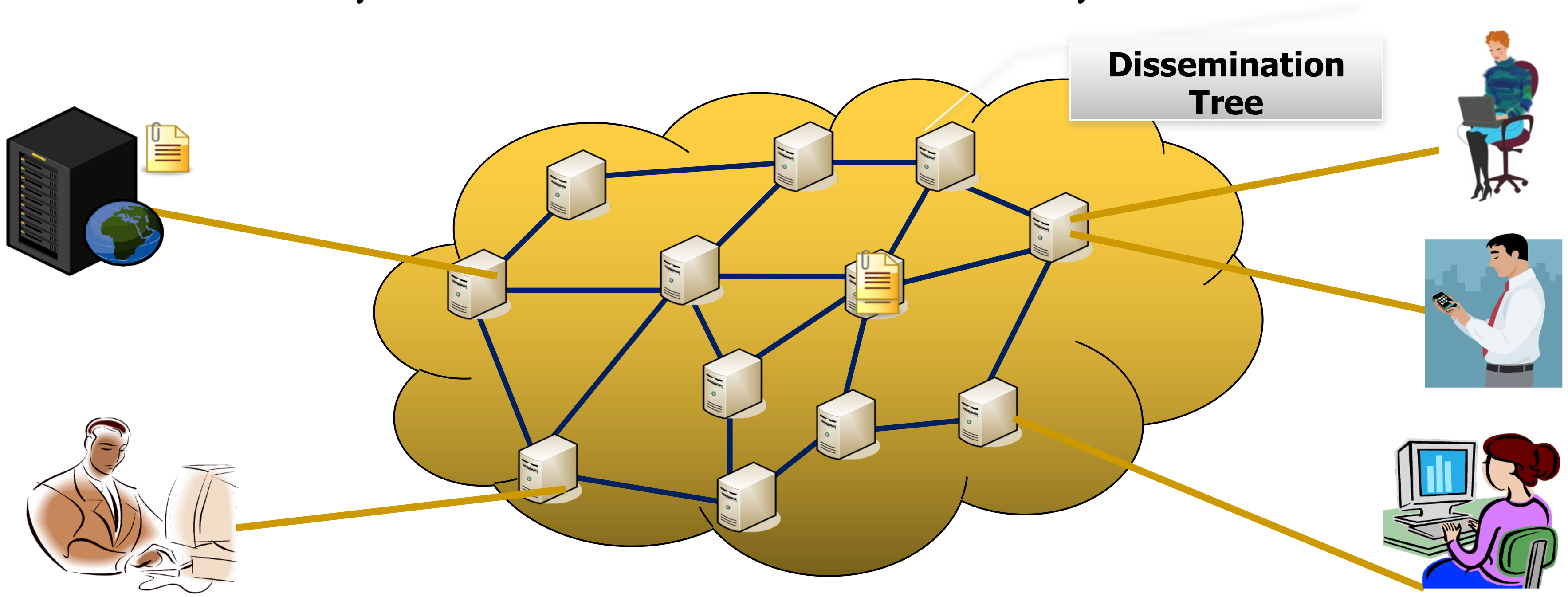


MINEMA Summer School - Klagenfurt (Austria) July 11-15, 2005

- Language and Data model
  - Conjunctive Boolean functions over predicates
  - Predicates are attribute-operator-value triples
    - `[class,=,trigger]`
  - Subscriptions are conjunctions of predicates
    - `[class,=,trigger],[appl,=,payroll],[gid,=,g001]`
  - Publications are sets of attribute-value pairs
    - `[class,trigger],[appl,printer],[gid,g007]`
- Matching semantic
  - A subscription matches if all its predicates are matched

Distributed pub/sub systems

- Broker – based pub/sub
  - A set of brokers forming an overlay
    - Clients use system through brokers
    - Benefits
- Scalability, Fault tolerance, Cost efficiency



Distributed Content-Based Pub/Sub

- Network of publish/subscribe brokers
- Subscriptions & publications are injected into network at closest edge broker
- Routing protocol distributes subscriptions throughout network
- Network routes relevant publications to interested subscribers
- Routing is based on content; it is not based on addresses, which are not available
- Subscriptions may change dynamically

Challenges in distributed pub/sub systems

Broker Responsibility

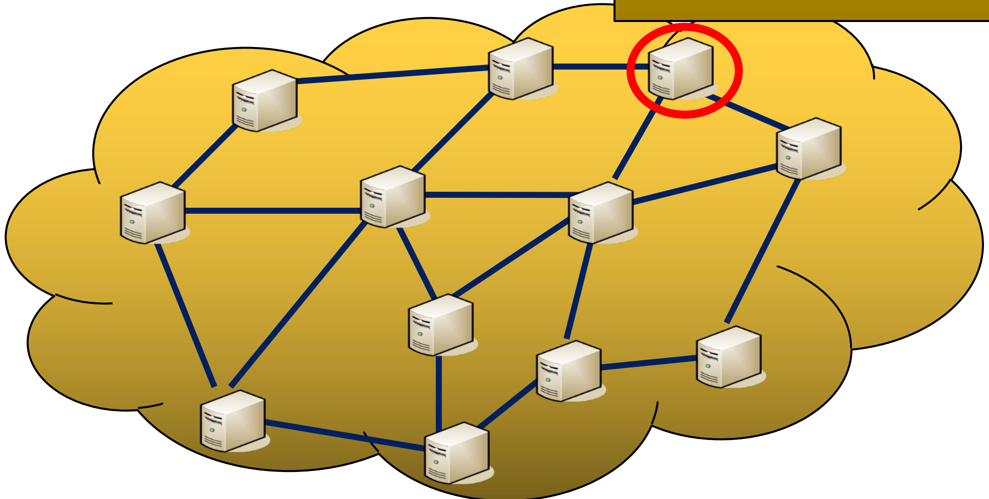
Subscription Management
Matching: Determining the recipients for an event
Routing: Delivering a notification to all the recipients

Broker overlay architecture

- How to form the broker network
- How to route subscriptions and publications

Broker internal operations

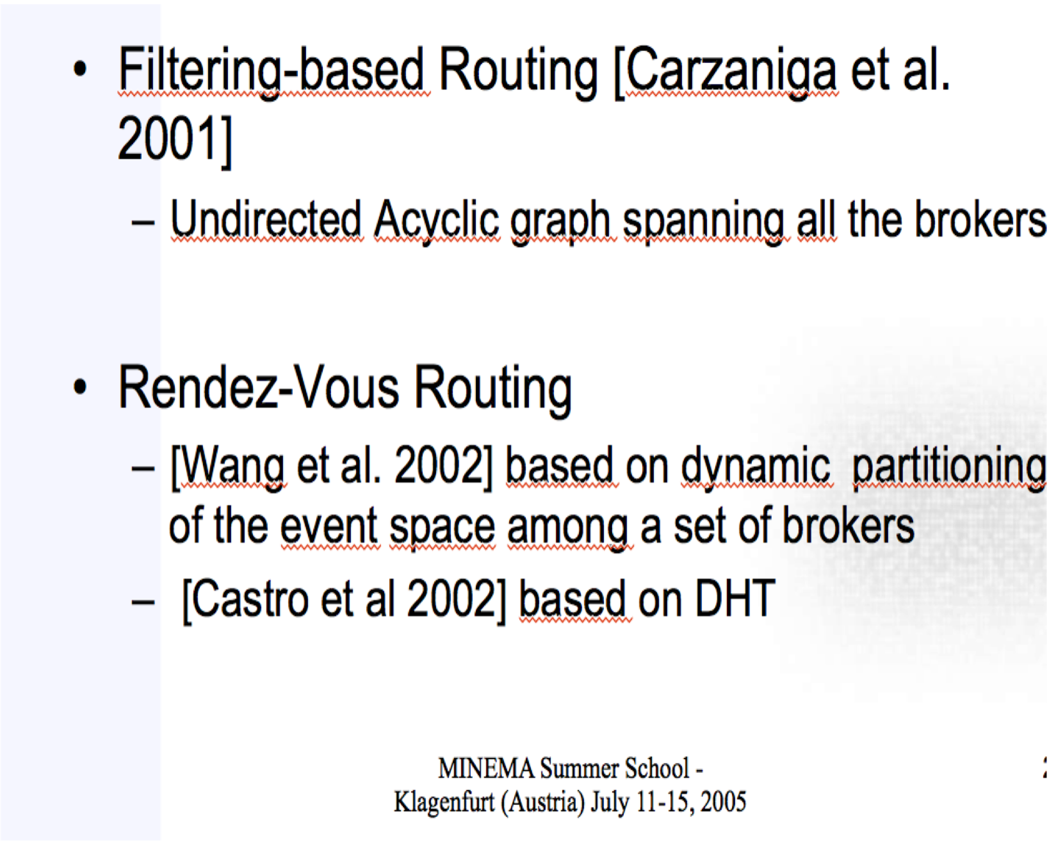
- Subscription management
  - How to store subscriptions in brokers
- Content matching in brokers
  - How to match a publication against subscriptions



EVENT vs SUBSCRIPTION ROUTING

- Extreme solutions
  - Sol 1 (event flooding)
    - flooding of events in the notification event box
    - each subscription stored only in one place within the notification event box
    - Matching operations equal to the number of brokers
  - Sol 2 (subscription flooding)
    - each subscription stored at any place within the notification event box
    - each event matched directly at the broker where the event enters the notification event box

Other routing solutions

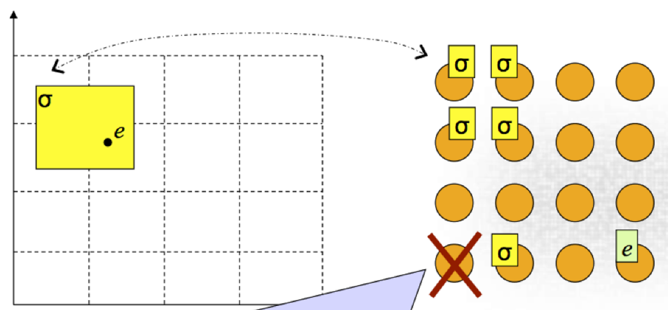


- Filtering-based Routing [Carzaniga et al. 2001]
  - Undirected Acyclic graph spanning all the brokers
- Rendez-Vous Routing
  - [Wang et al. 2002] based on dynamic partitioning of the event space among a set of brokers
  - [Castro et al 2002] based on DHT

MINEMA Summer School -
Klagenfurt (Austria) July 11-15, 2005

- Identify as soon as possible events that are not interesting for any subscriber and arrest their diffusion
- Requires routing info to be maintained at brokers - set of filters (aggregate of subscriptions) that are reachable through that broker

- Each node is responsible for a partition of the event space
  - Storing subscriptions, matching events



Problem: difficult to define mapping functions when the set of nodes changes over time

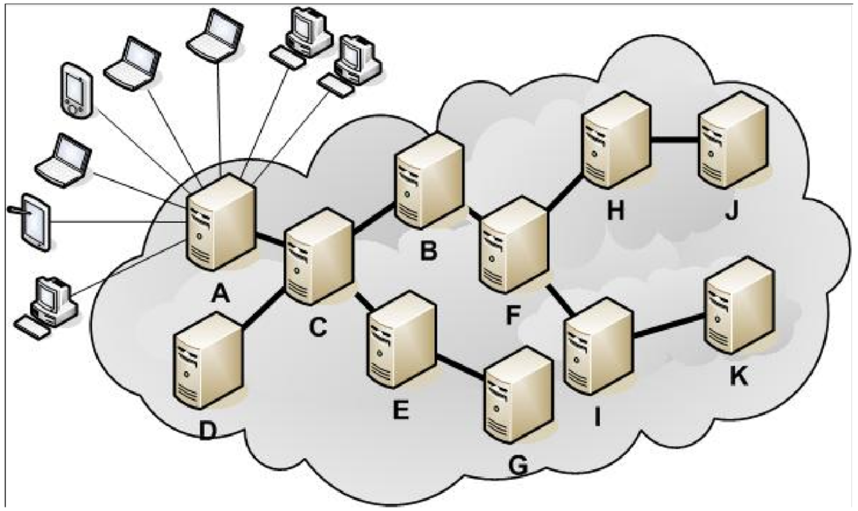
Klagenfurt (Austria) July 11-15, 2005

Major distributed pub/sub approaches

- Tree-based
  - Brokers form a tree overlay [SIENA, PADRES, GRYPHON]
- DHT-based:
  - Brokers form a structured P2P overlay [Meghdoot, Baldoni et al.]
- Channel-based:
  - Multiple multicast groups [Phillip Yu et al.]
- Probabilistic:
  - Unstructured overlay [Picco et al.]

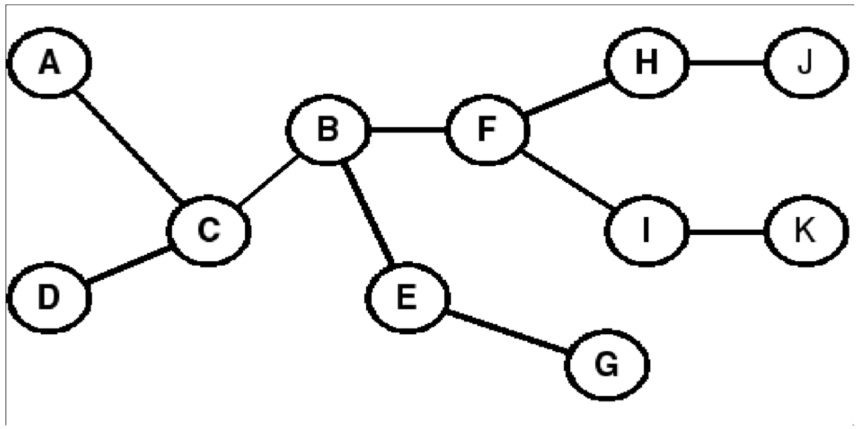
Tree-based

- Brokers form an acyclic graph
- Subscriptions are broadcast to all brokers
- Publications are disseminated along the tree with applying subscriptions as filters



Tree-based

- Subscription dissemination load reduction
  - Subscription Covering
  - Subscription Subsumption
- Publication matching
  - Index selection

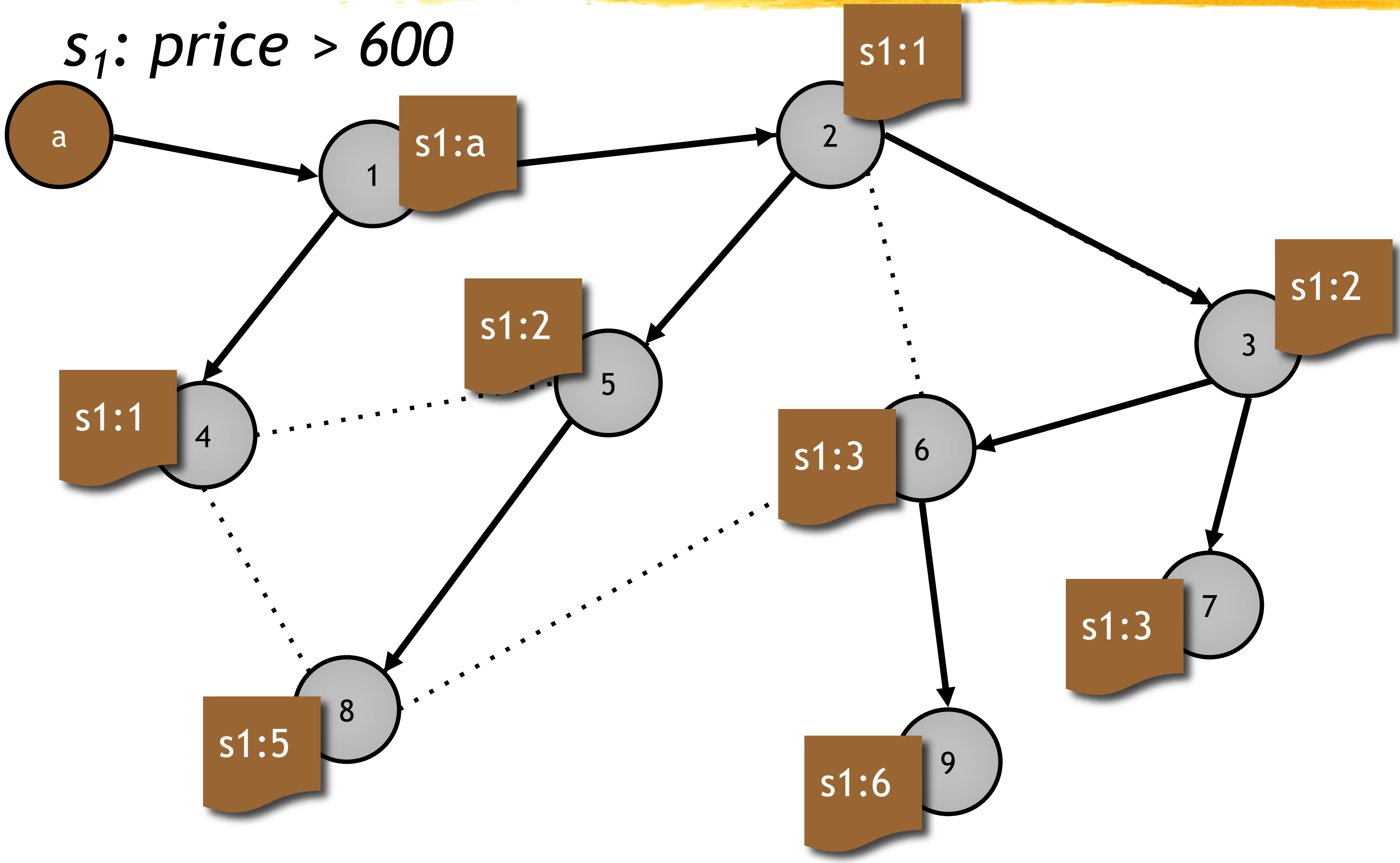


Pub/Sub Sysems: Tib/RV [Oki et al 03]

- Topic Based
- Two level hierarchical architecture of brokers (deamons) on TCP/IP
- Event routing is realized through one diffusion tree per subject
- Each broker knows the entire network topology and current subscription configuration

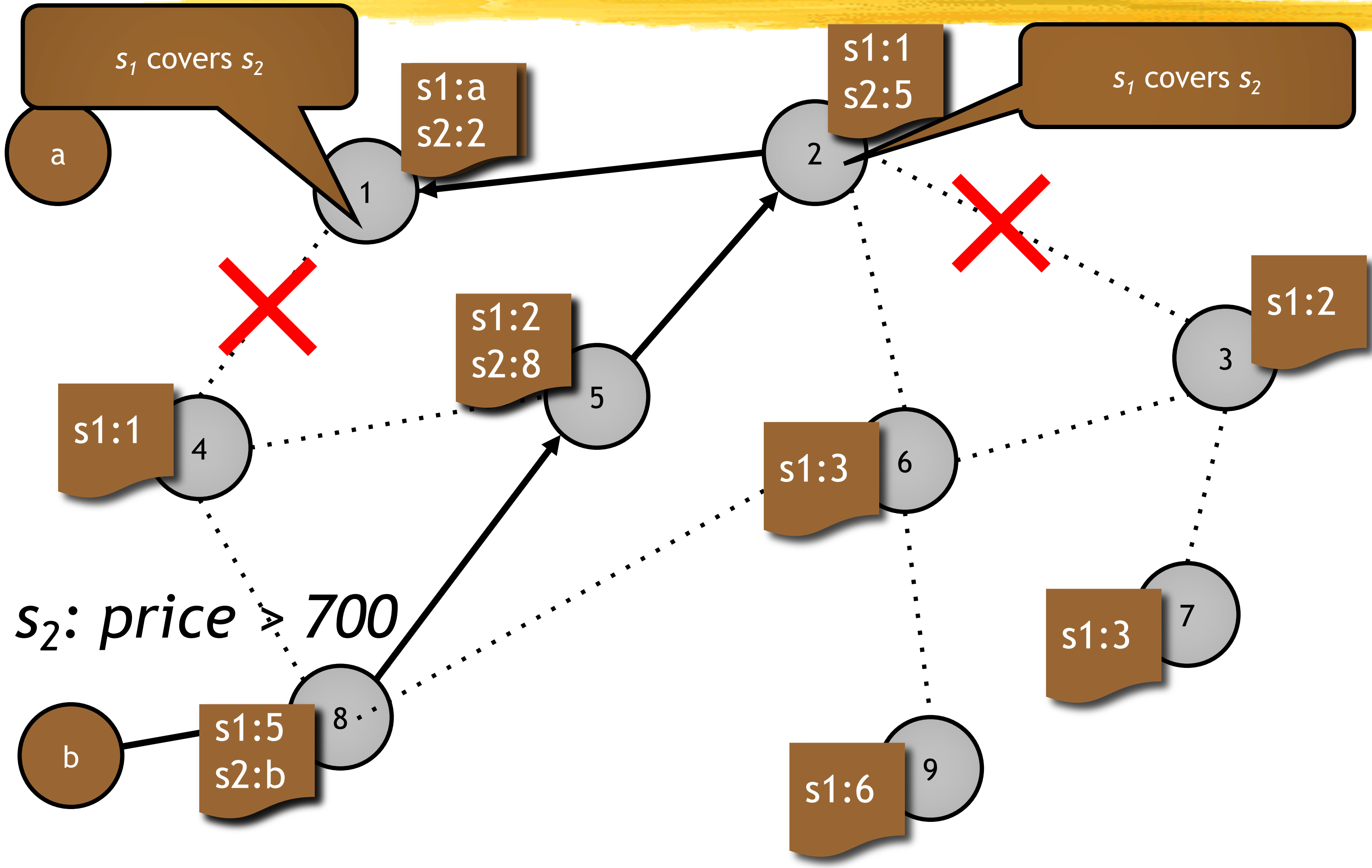
SIENA Filtering-Based Routing
Subscription Forwarding

s_1: price > 600

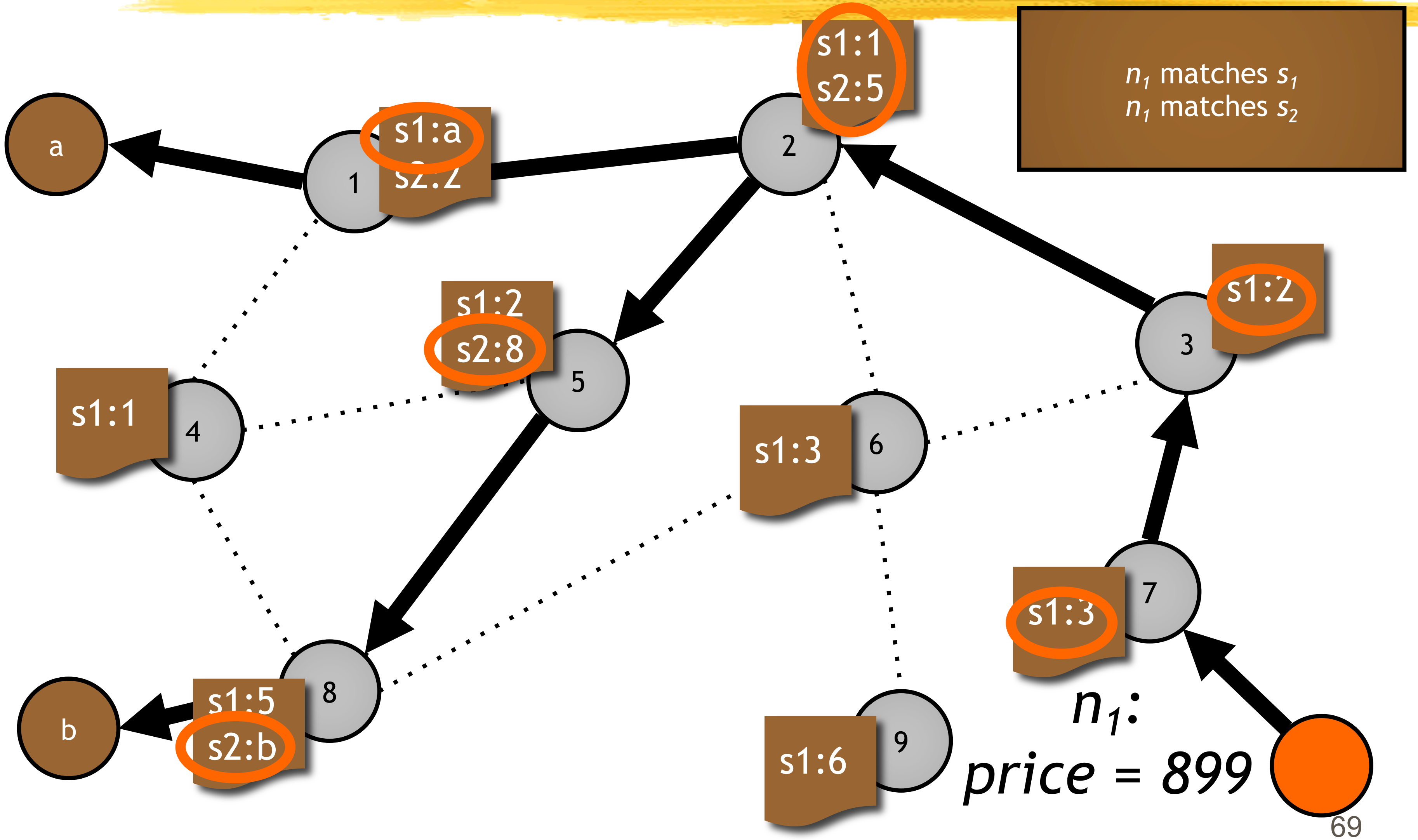


SIENA Filtering-Based Routing

Subscription Merging



SIENA Filtering-Based Routing
Notification Delivery



Pub/Sub systems: Gryphon [IBM 00]

- Content based
- Hierarchical tree from publishers to subscribers
- Filtering-based routing
- Mapping content-based to network level multicast

DHT Based Pub/Sub: SCRIBE [Castro et al. 02]

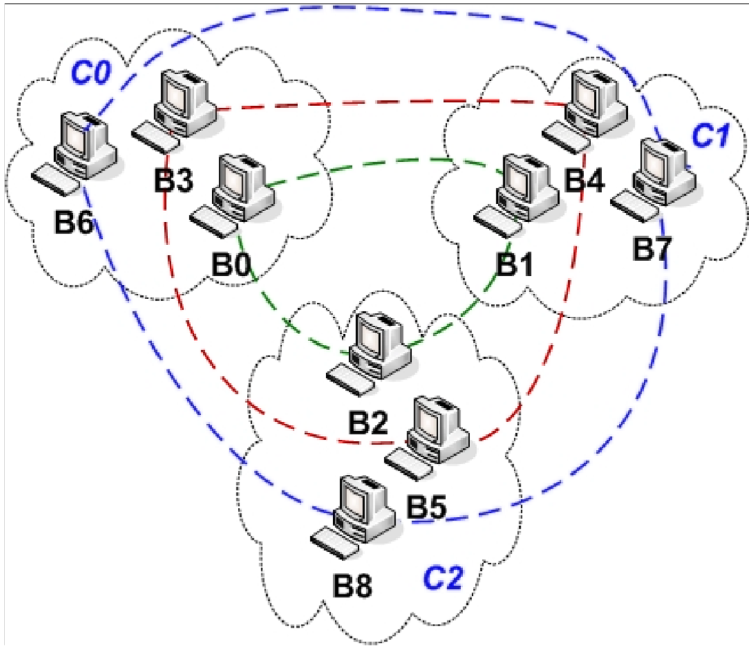
- Topic Based
- Based on DHT (Pastry)
- Rendez-vous event routing
- A random identifier is assigned to each topic
- The pastry node with the identifier closest to the one of the topic becomes responsible for that topic

DHT-based pub/sub MEGHDOOT

- Content Based
- Based on Structured Overlay CAN
- Mapping the subscription language and the event space to CAN space
- Subscription and event Routing exploit CAN routing algorithms

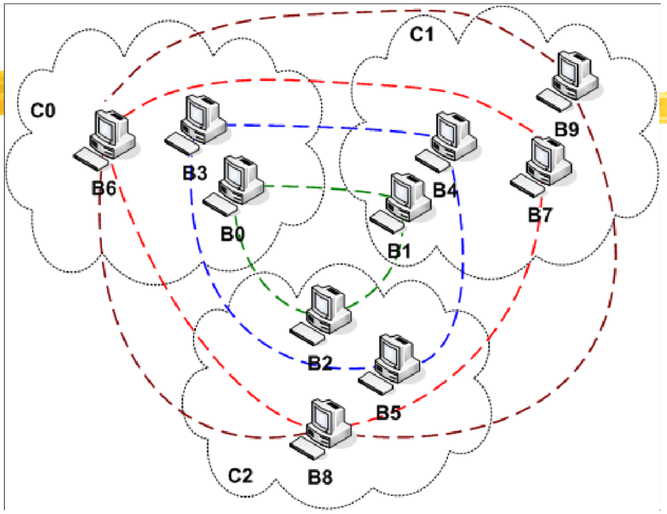
Fault-tolerant Pub/Sub

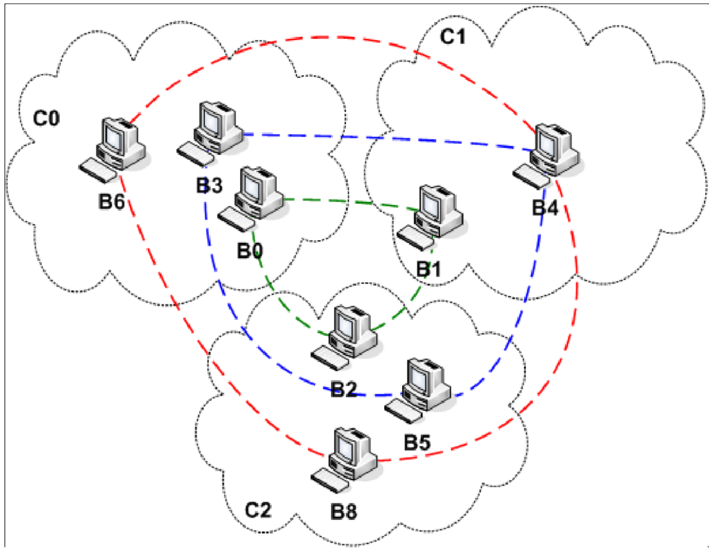
- Brokers are clustered
- Each broker knows all brokers in its own cluster and at least one broker from every other cluster
- Subscriptions are broadcast within clusters
- Every broker maintains subscriptions from brokers in the same cluster
- Subscription aggregation is done based on brokers



Fault-tolerant Pub/Sub

- Broker overlay
  - Join
  - Leave
  - Failure
    - Detection
    - Masking
    - Recovery
- Load Balancing
  - Ring publish load
  - Cluster publish load
  - Cluster subscription load





Messaging Model

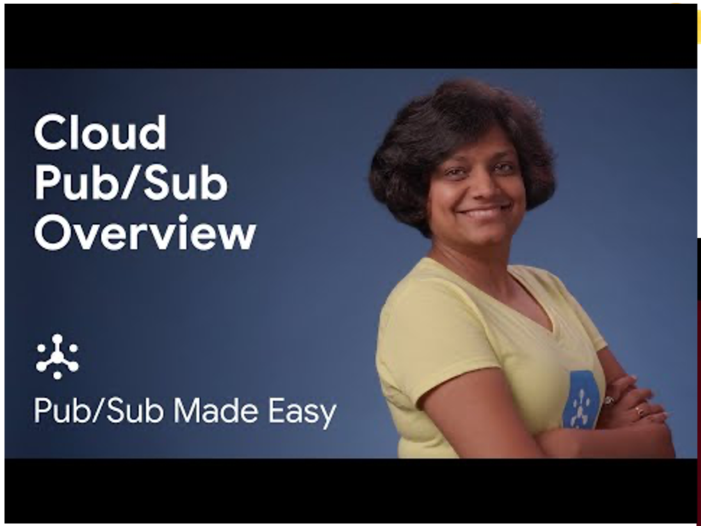
- MQTT: ISO standardized pub/sub protocol
  - TCP/IP
  - Small code footprint, low-bandwidth design
  - Has become a common platform for IoT [1]
- Simple protocol
  - CONNACK
  - SUBSCRIBE(t), SUBACK(t)
  - PUBLISH(t,msg)
  - ...

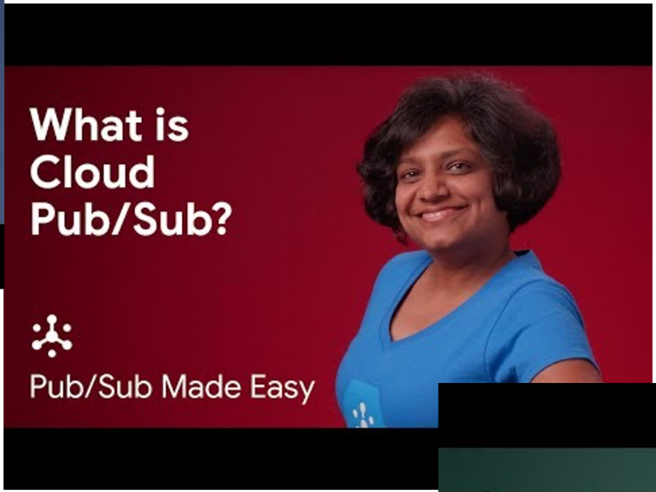
[1] A. Al-Fuqaha, et al. 2015. **Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications** IEEE Commun. Surv. Tutorials, vol. 17, no. 4, pp. 2347–2376, 2015.

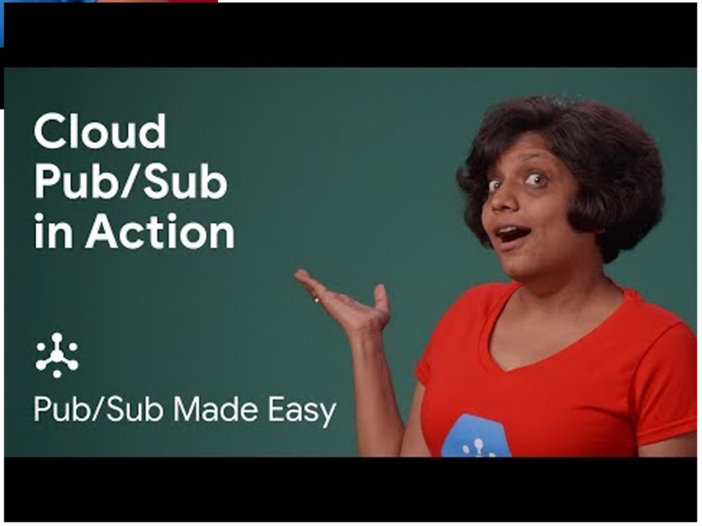
Messaging Model

- Message delivery guarantees (MQTT "QoS")
  - (0) At most once
  - (1) At least once
  - (2) Exactly once
- Maintaining guarantees despite distribution

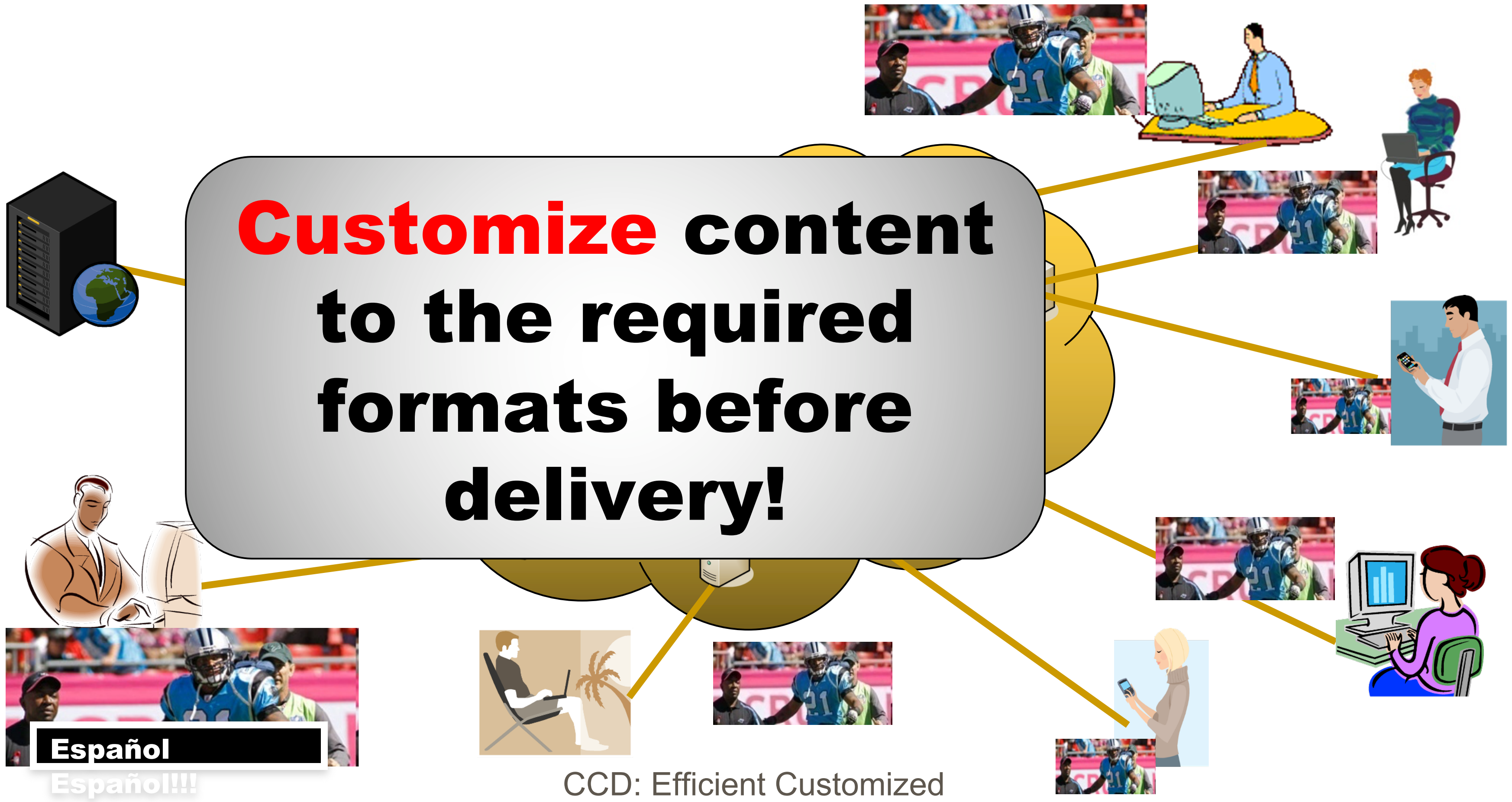
Google Cloud Pub/Sub







Customized content delivery with pub/sub



Customize content to the required formats before delivery!

Motivation

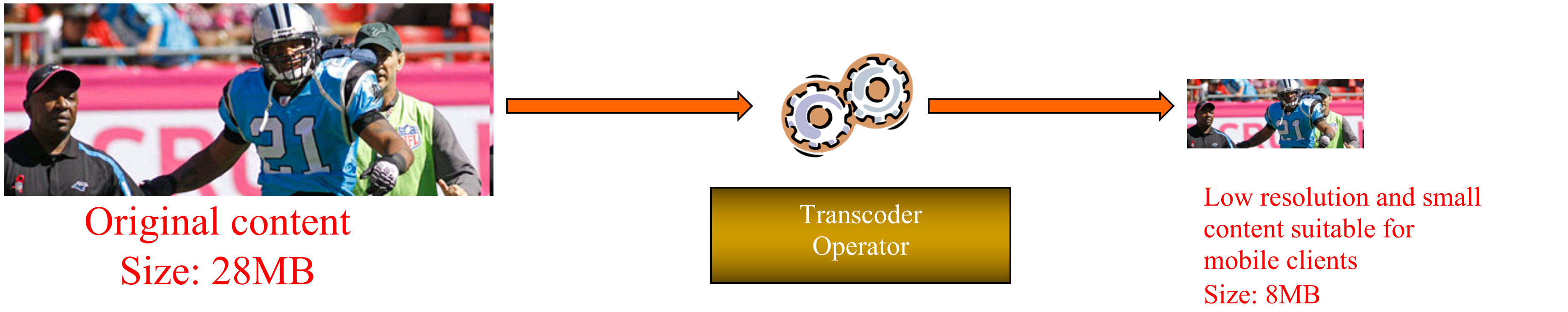
Leveraging pub/sub framework for dissemination of rich content formats, e.g., multimedia content.

Same content format may not be consumable by all subscribers!!!

Challenges: Content customization

How is content customization done?

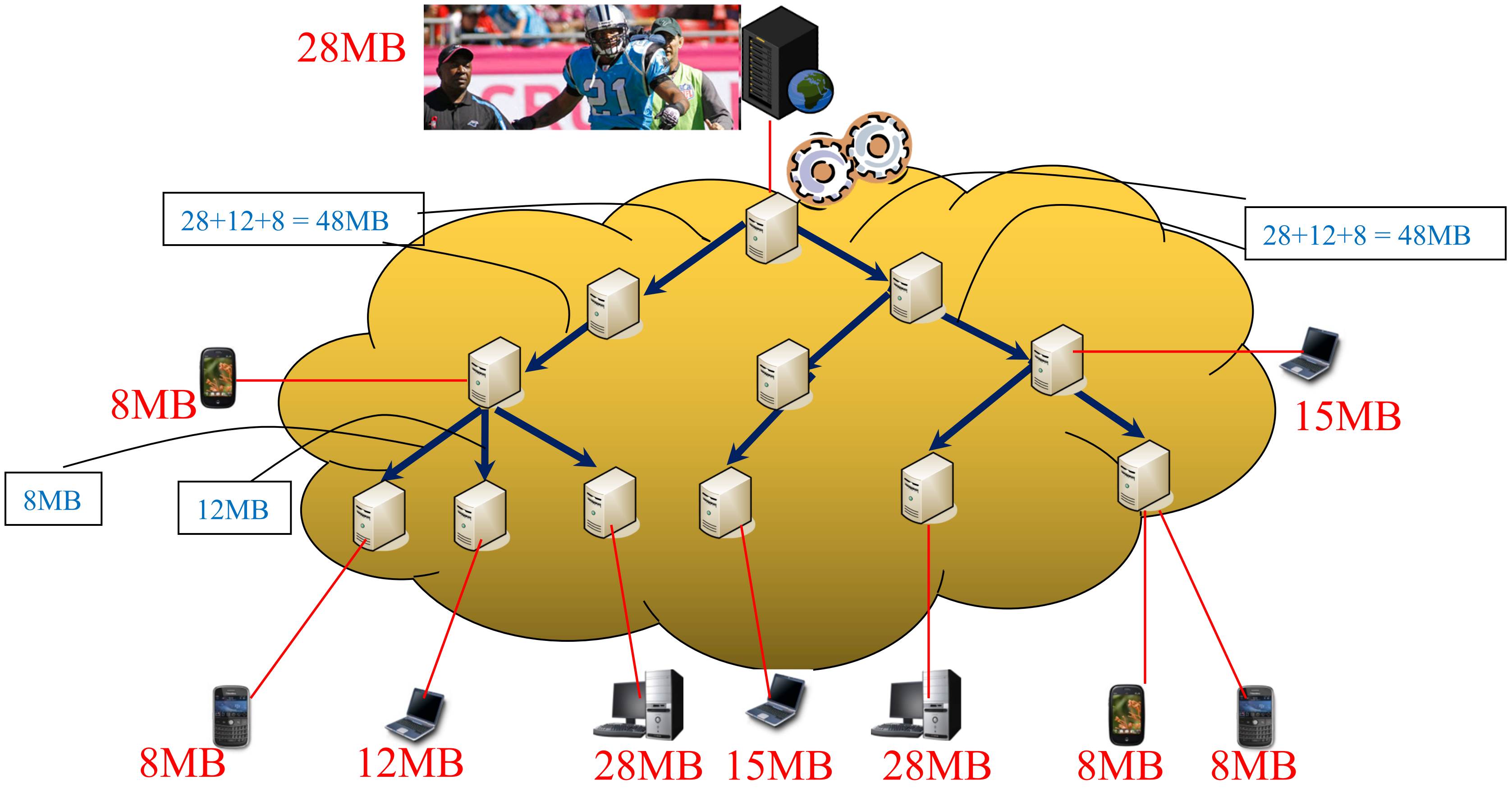
- Through Adaptation operators


Original content
Size: 28MB
Transcoder Operator
Low resolution and small content suitable for mobile clients
Size: 8MB

- How can customization be implemented in a distributed pub/sub system?

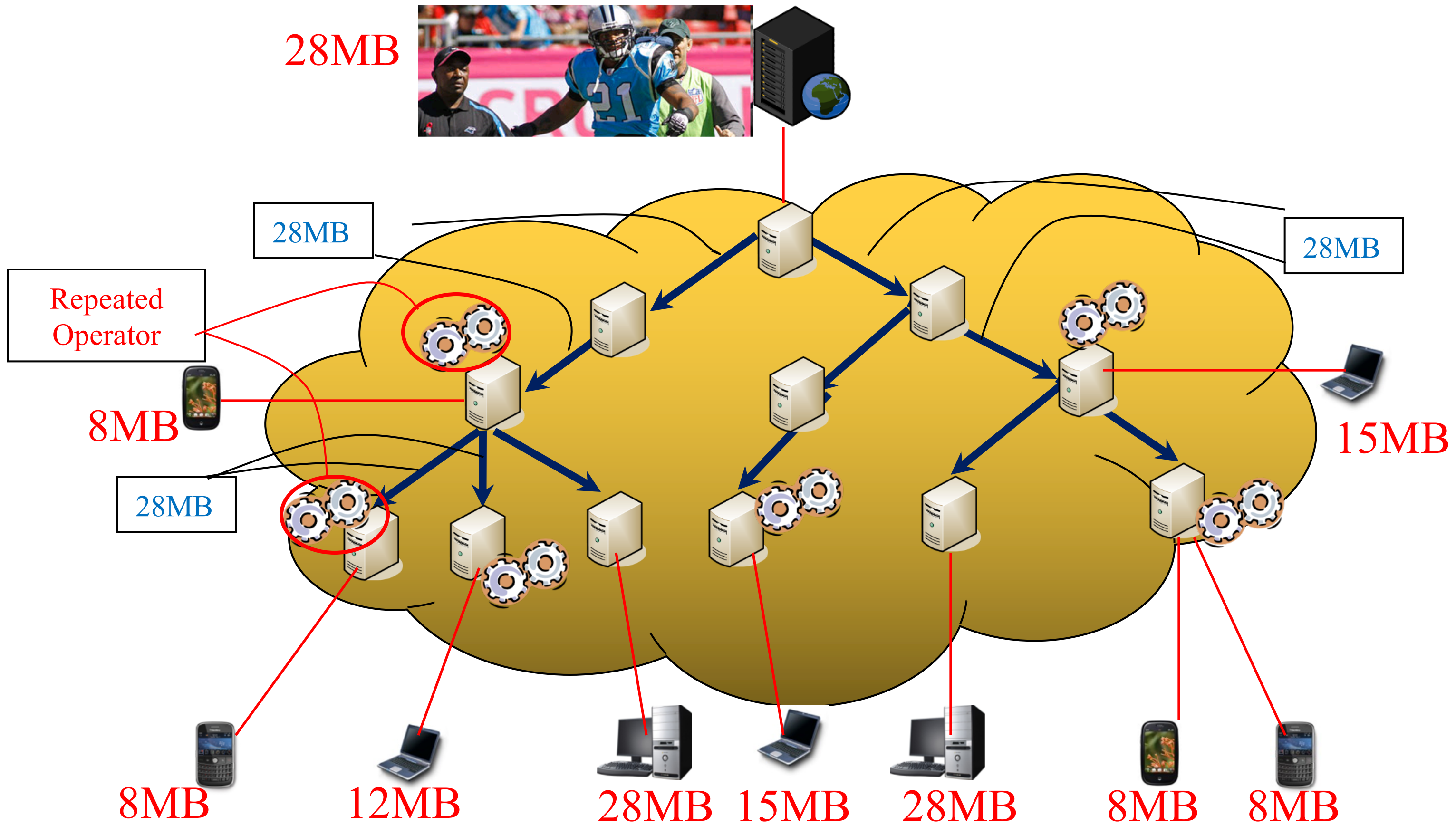
Challenges

Option 1: Perform all the required customizations in the sender broker



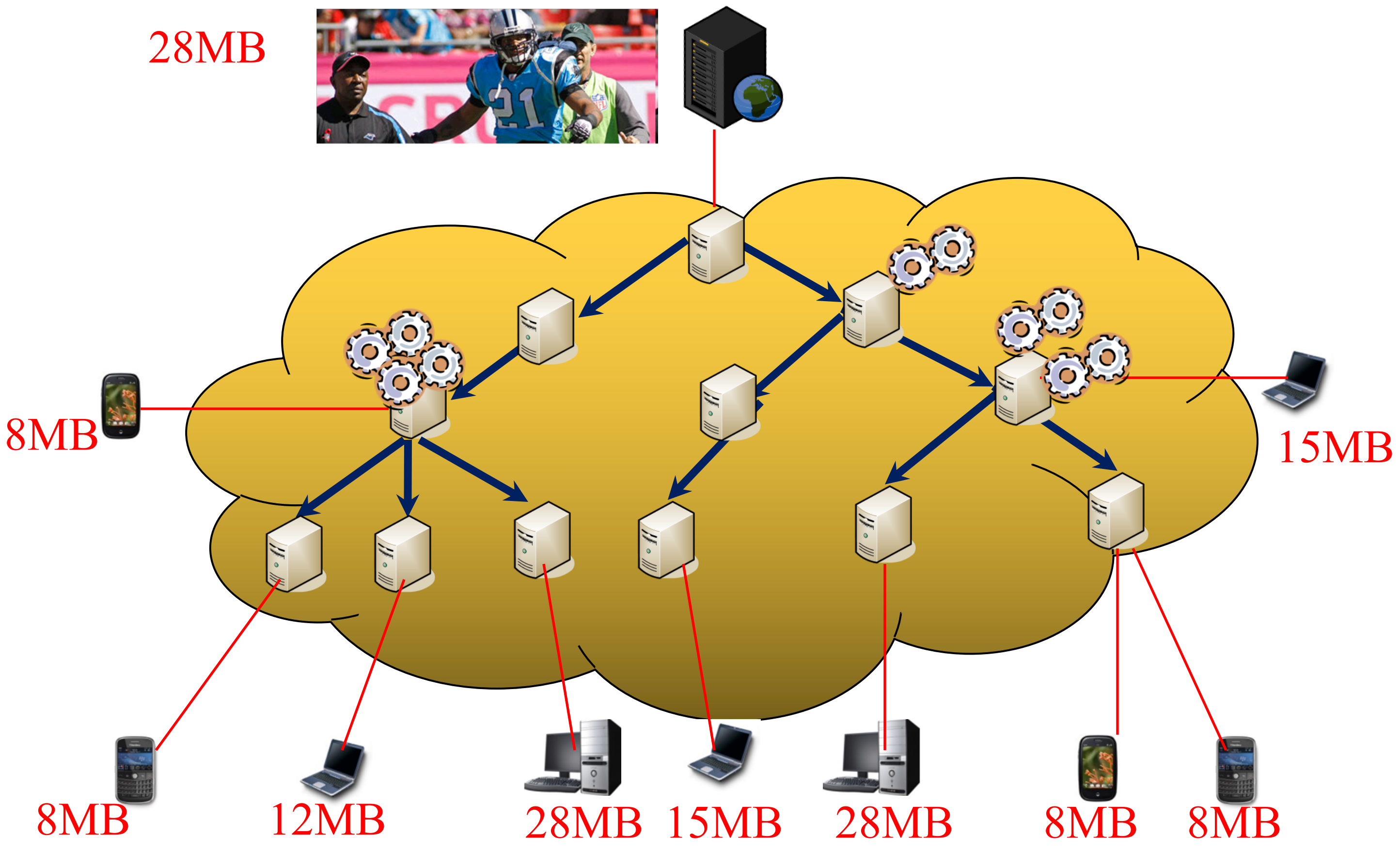
Challenges

- Option 2: Perform all the required customization in the proxy brokers (leaves)

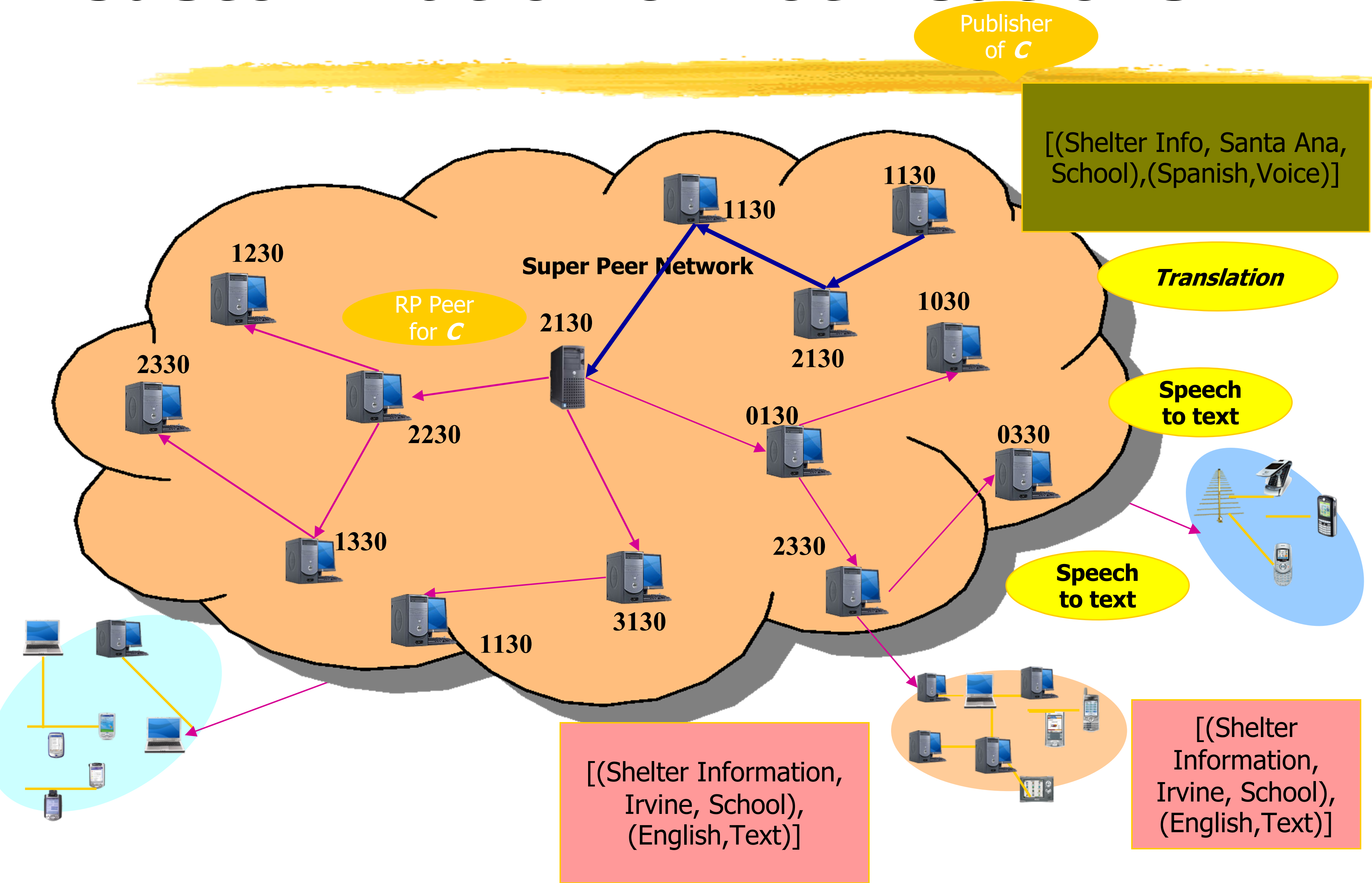


Challenges

- Option 3: Perform all the required customization in the broker overlay network

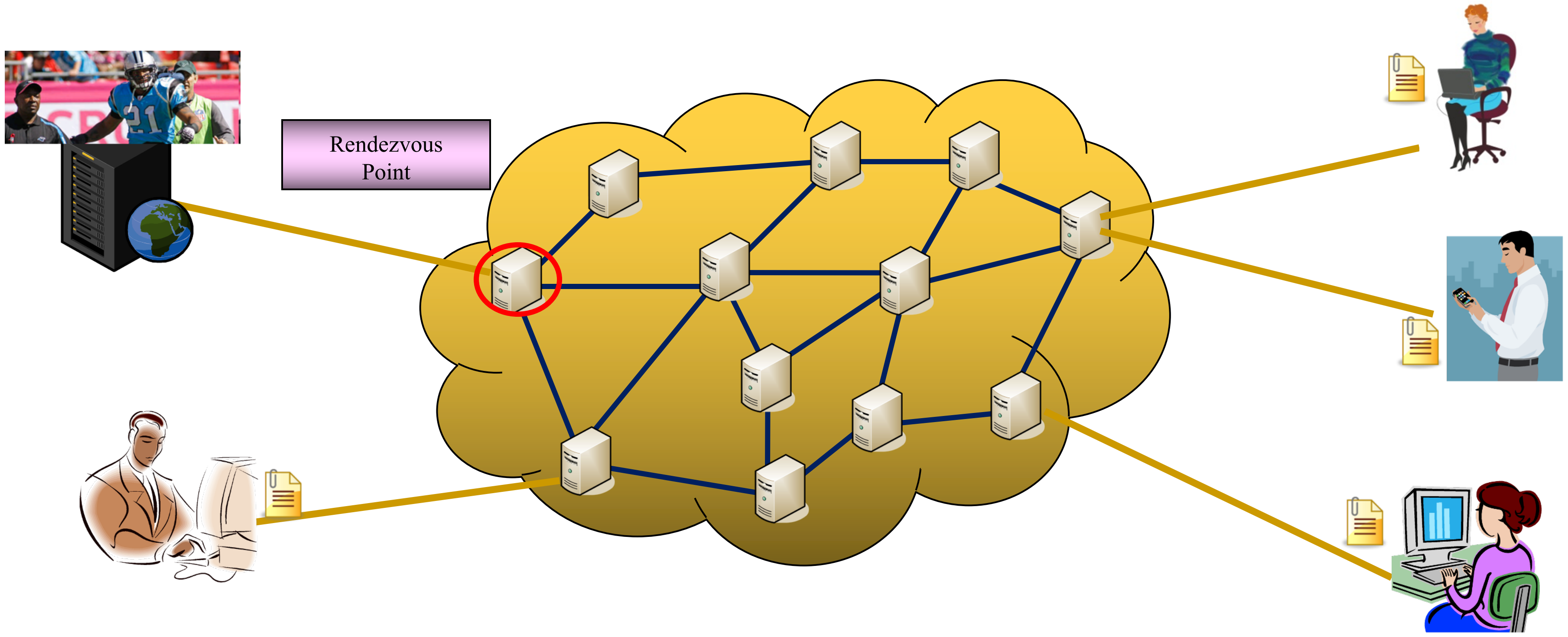


Example: In network customization of notifications


Publisher of *C*
[(Shelter Info, Santa Ana, School),(Spanish,Voice)]
1130
1130
Super Peer Network
1230
RP Peer for *C*
2130
2130
1030
Translation
2330
2230
0130
0330
Speech to text
1330
2330
Speech to text
3130
1130
[(Shelter Information, Irvine, School), (English,Text)]
[(Shelter Information, Irvine, School), (English,Text)]

DHT-based pub/sub

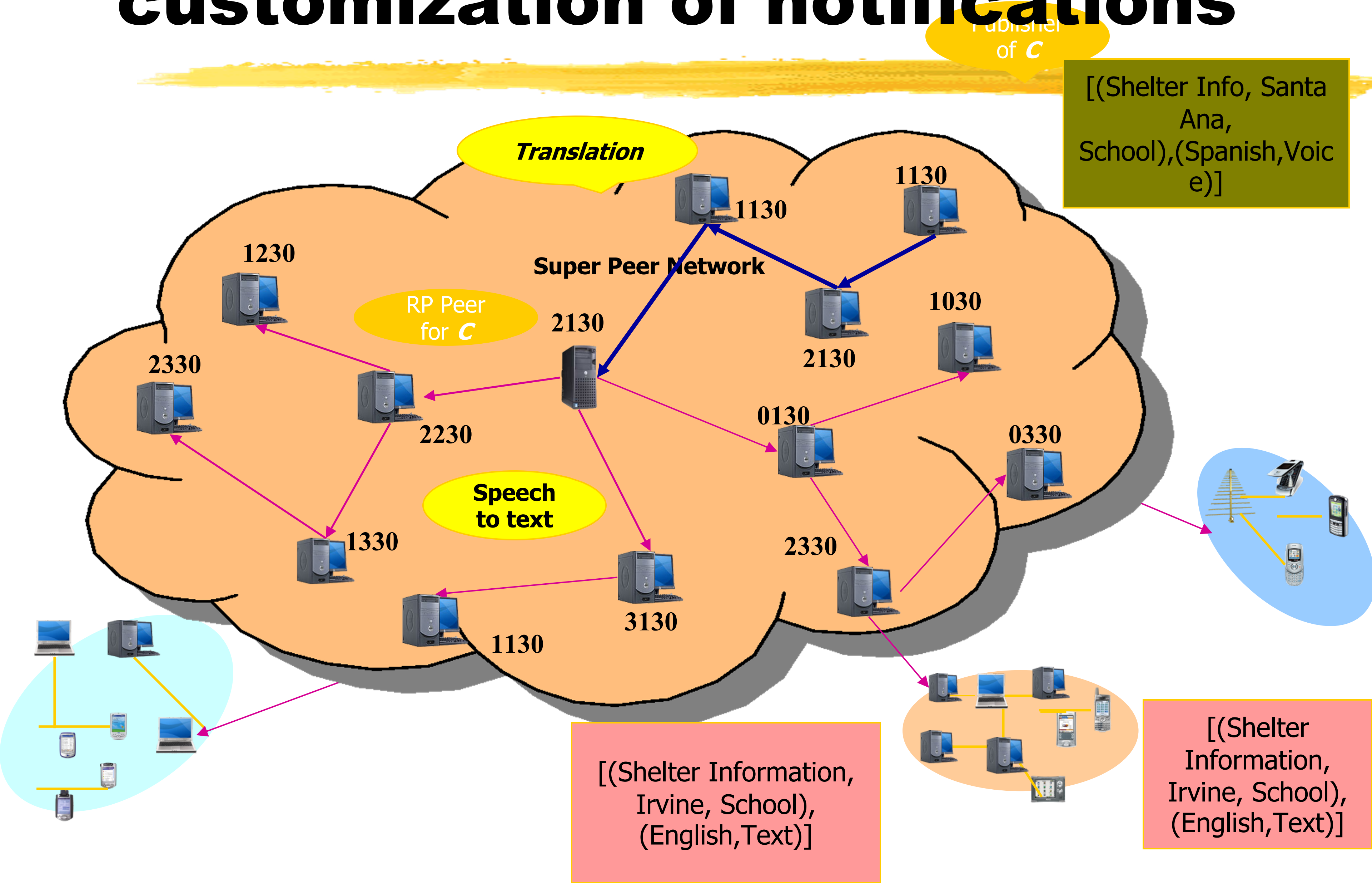
- DHT-based routing schema, e,g. Tapestry



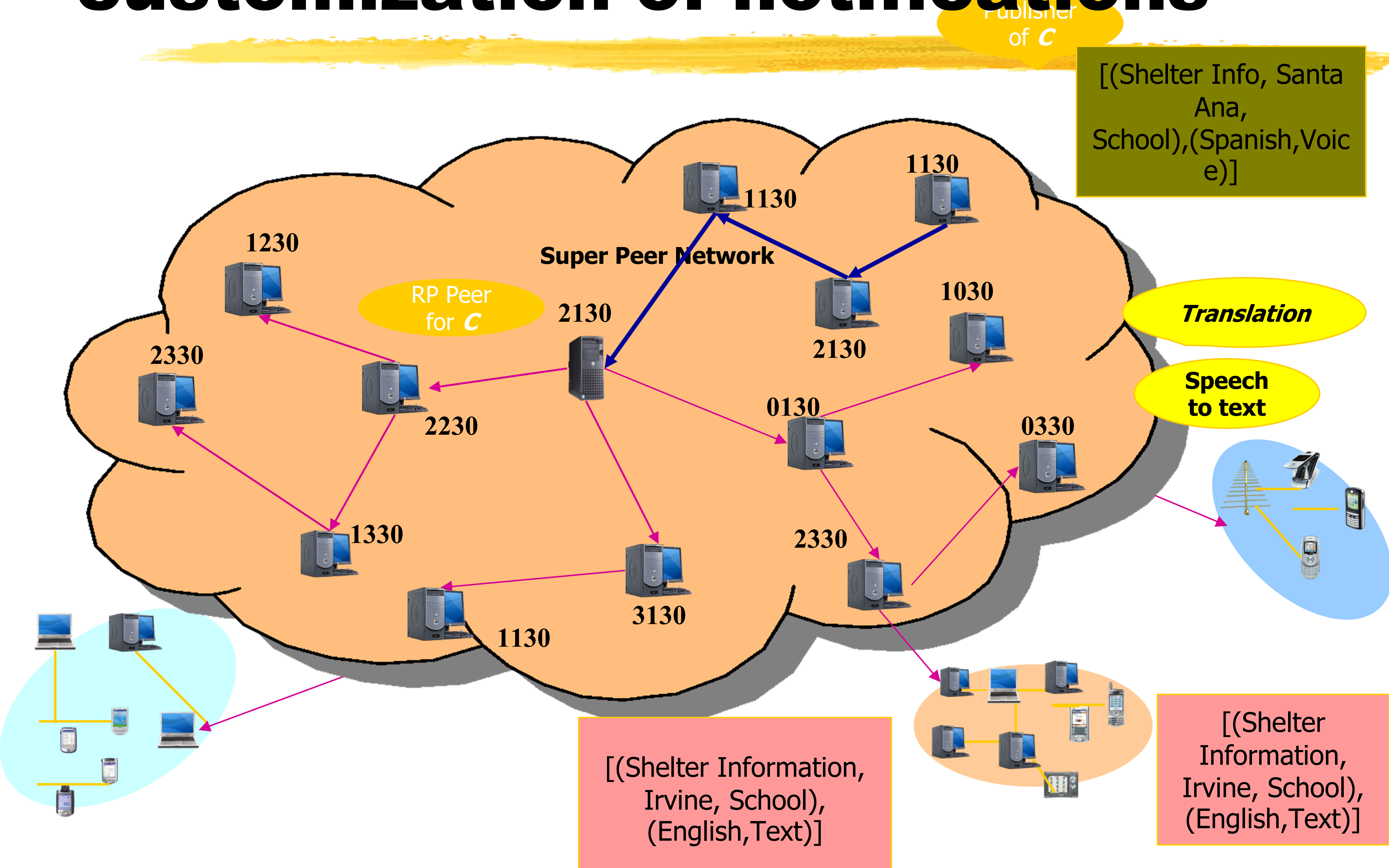
Example using DHT based pub-sub

- Tapestry (DHT-based) pub/sub and routing framework
  - Event space is partitioned among peers
    - Single content matching
  - Each partition is assigned to a peer (RP)
  - Publications and subscriptions are matched in RP
    - All receivers and preferences are detected after matching
  - Content dissemination among matched subscribers are done through a dissemination tree rooted at RP where leaves are subscribers.

Example: In network customization of notifications


Publisher of *C*
[(Shelter Info, Santa Ana, School),(Spanish,Voice)]
Translation
1130
1130
1230
Super Peer Network
RP Peer for *C*
2130
2130
1030
2330
2230
0130
0330
Speech to text
1330
2330
3130
1130
[(Shelter Information, Irvine, School), (English,Text)]
[(Shelter Information, Irvine, School), (English,Text)]

Example: In network customization of notifications

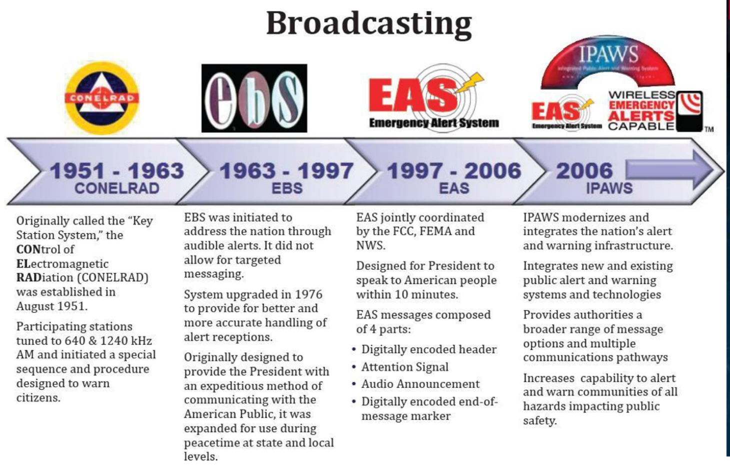

Publisher of *C*
[(Shelter Info, Santa Ana, School),(Spanish,Voice)]
1130
1130
Super Peer Network
1230
RP Peer for *C*
2130
2130
1030
Translation
Speech to text
2330
2230
0130
0330
1330
2330
3130
1130
[(Shelter Information, Irvine, School), (English,Text)]
[(Shelter Information, Irvine, School), (English,Text)]

Videos..

Twitter experiences

https://www.youtube.com/watch?v=zwo3ipH4LZU

Enriched and customized notification systems are needed


Broadcasting

1951 - 1963 CONELRAD

Originally called the "Key Station System," the **CON**trol of **EL**ectromagnetic **RAD**iation (CONELRAD) was established in August 1951.

Participating stations tuned to 640 & 1240 kHz AM and initiated a special sequence and procedure designed to warn citizens.

1963 - 1997 EBS

EBS was initiated to address the nation through audible alerts. It did not allow for targeted messaging.

System upgraded in 1976 to provide for better and more accurate handling of alert receptions.

Originally designed to provide the President with an expeditious method of communicating with the American Public, it was expanded for use during peacetime at state and local levels.

1997 - 2006 EAS

EAS jointly coordinated by the FCC, FEMA and NWS.

Designed for President to speak to American people within 10 minutes.

EAS messages composed of 4 parts:

- Digitally encoded header
- Attention Signal
- Audio Announcement
- Digitally encoded end-of-message marker

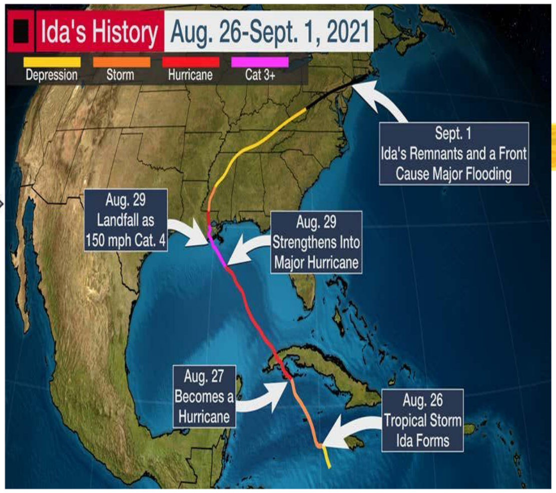
2006 IPAWS

IPAWS modernizes and integrates the nation's alert and warning infrastructure.

Integrates new and existing public alert and warning systems and technologies

Provides authorities a broader range of message options and multiple communications pathways

Increases capability to alert and warn communities of all hazards impacting public safety.


Ida's History Aug. 26-Sept. 1, 2021
Depression
Storm
Hurricane
Cat 3+
Sept. 1 Ida's Remnants and a Front Cause Major Flooding
Aug. 29 Landfall as 150 mph Cat. 4
Aug. 29 Strengthens Into Major Hurricane
Aug. 27 Becomes a Hurricane
Aug. 26 Tropical Storm Ida Forms

BREAKING OVERNIGHT
DEADLY FLASH FLOOD EMERGENCY
IDA'S REMNANTS BATTER MILLIONS ACROSS NORTHEAST
GMA

Subways were shutdown

Most cars were stuck

Buses were able to operate at 90%

Enriched, actionable and customized notifications:

- **Customized** notifications, e.g., nearby traffic and road conditions, etc.
- **Enriched** notifications: maps, pictures, videos, shelter locations, alternate routes, etc.

Population scale and active notification systems:

- Accommodate a **large-scale** workload, many end-users
- Allow users to subscribe and provide **active** data delivery

Next Generation Notification Systems

Big Data Publish Subscribe Systems – Pub/Sub

Paradigms Based on Big Data

"PetaBytes to MegaFolks in Millisecond s"

Big Data Publish Subscribe System = **Big Data Management System** + **Distributed Publish Subscribe System**

Scalable data ingestion and processing

Scalable data delivery

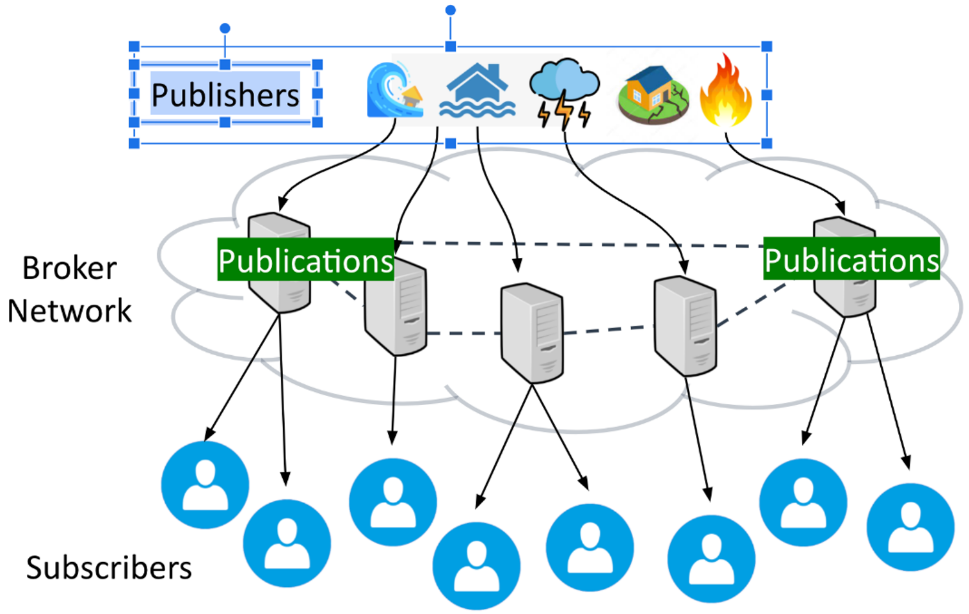
Big Data Pub Sub

Traditional Publish Subscribe Systems

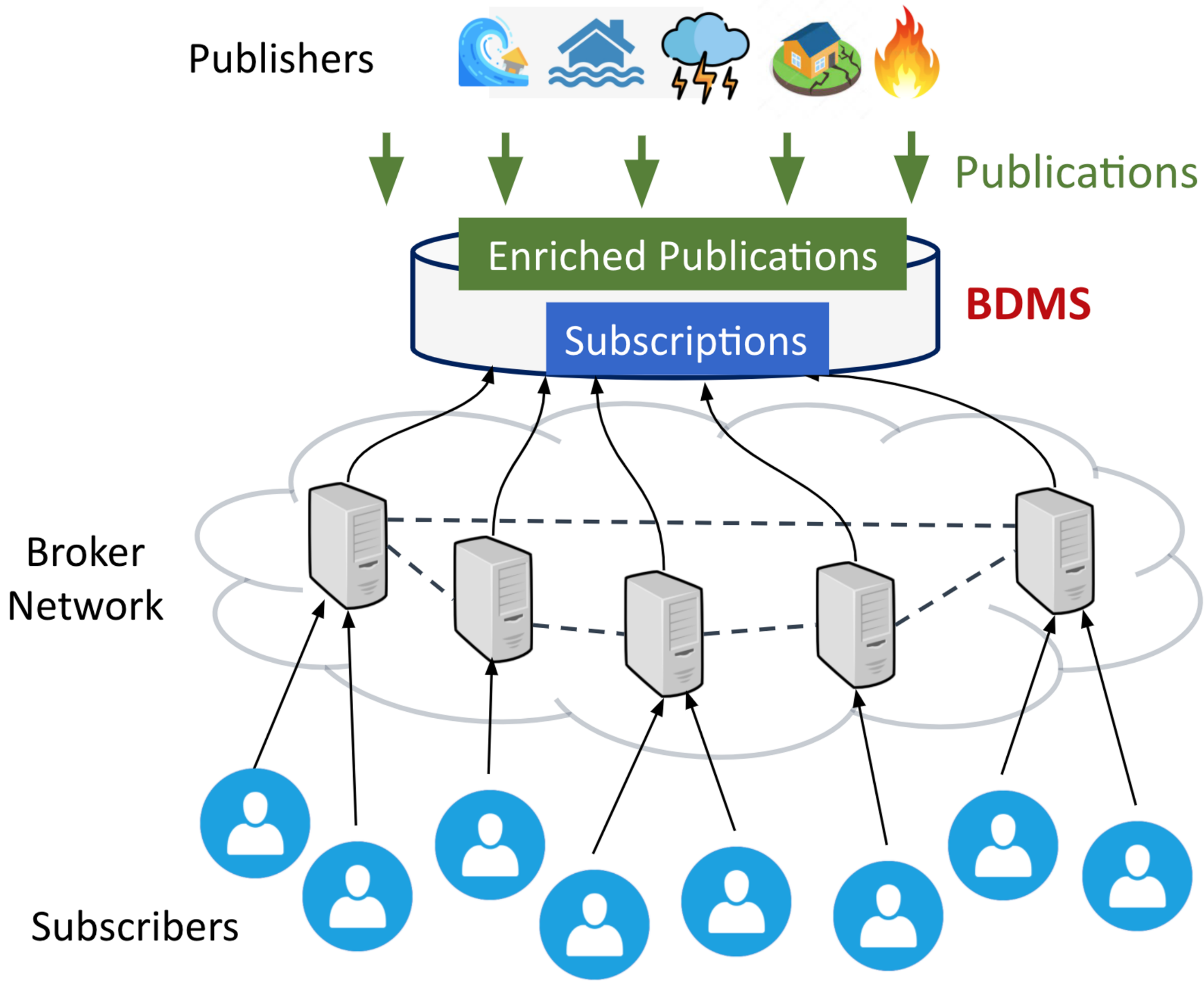
Communication paradigm that decouples data **publishers** and data **subscribers**:

- Entities: publishers, subscribers, brokers, publications, subscriptions
- Types: topic-based, content-based, …
- Architectures: client-server, P2P
- Subscription language:

[class, eq, 'STOCK'] ∧ [symbol, eq, 'YHOO'] ∧ [price, >, 300]



BDPS: Enriched Publications and Subscriptions



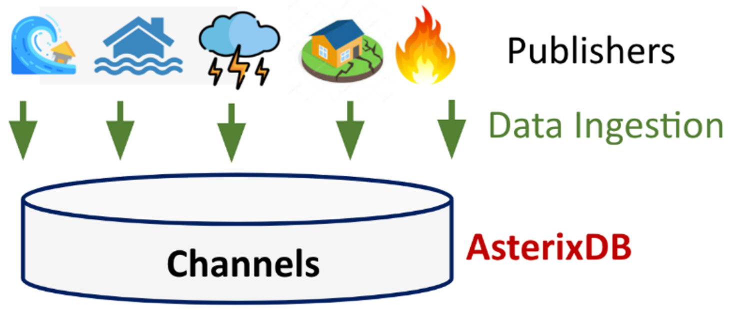
BDPS - How do we enrich and scale at BDMS?

BDMS: AsterixDB, open-source BDMS

Active Toolkit *(A BAD Thesis, Jacobs. Steven, 2018)*
- *Data feeds:* persist data streams into BDMS
- *Repetitive channels*: shared functions which produces individualized results for many users *repetitively*

Data Enrichment
- User defined function (UDF): during ingestion
- Publications and enrichment datasets

Activate Big Data at Scale *(Xikui Wang, 2020)*
- *Dynamic data feeds:* ingest data at scale, adapt to changes in referenced data
- *Continuous channels*: deliver incremental updates



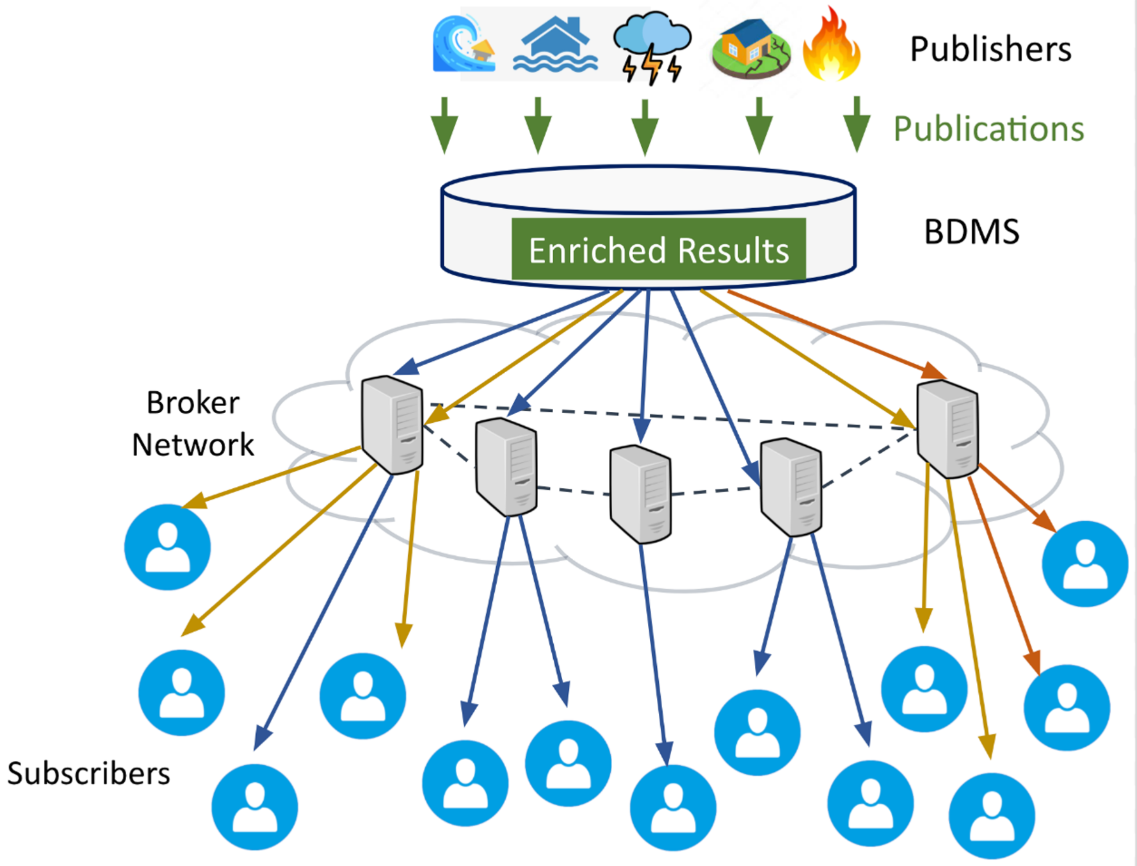
```
create function recentEmergenciesOfType(emergencyType){
  (select r as reports from
  (select value r from EmergencyReports r
   where r.timestamp > current_datetime() –
              day_time_duration("PT10S")) r
    where r.emergencyType = emergencyType)
};
create repetitive channel recentEmergenciesOfTypeChannel
using recentEmergenciesOfType@1 period duration("PT10S");

subscribe to recentEmergenciesOfTypeChannel(“tornado”);
```

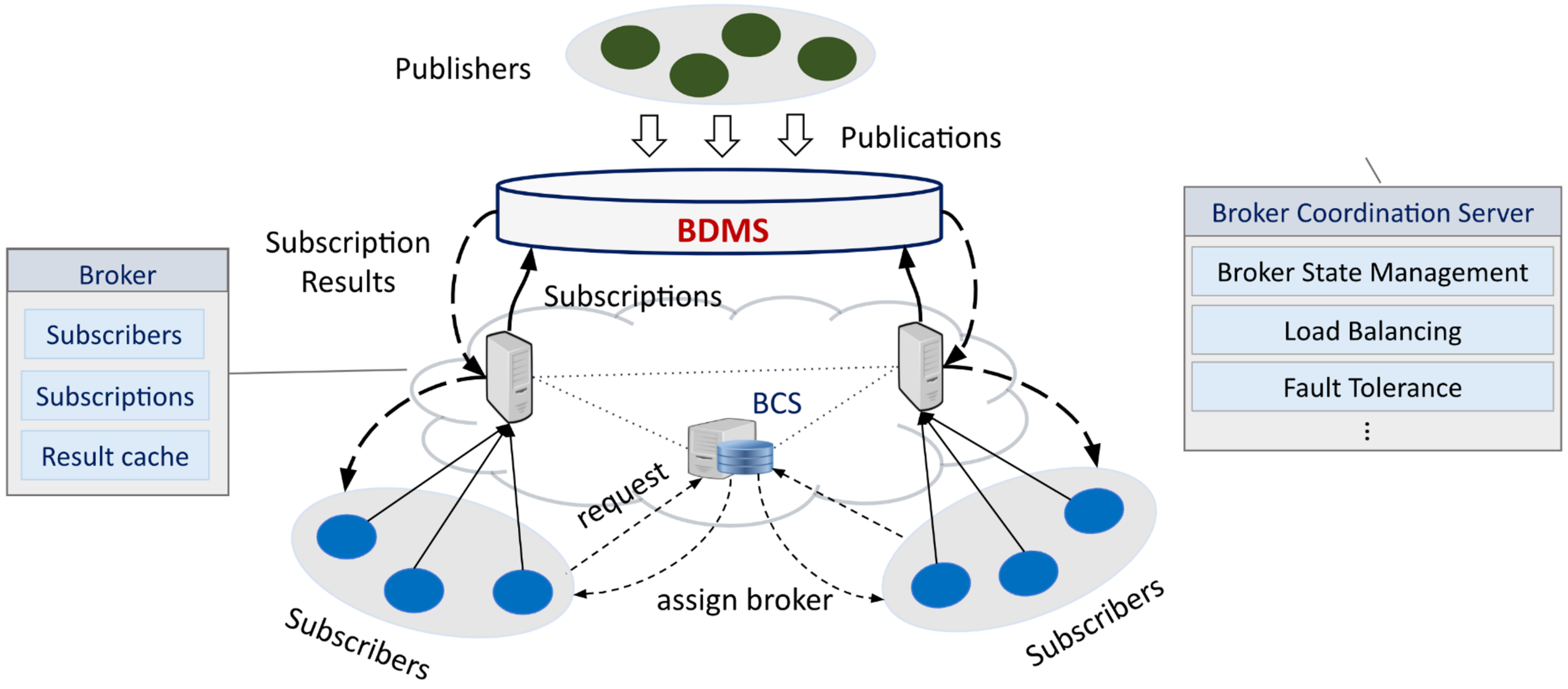
BDPS : Support for scaling in distributed Broker Network

Fan-out Distributed Broker Network

- Front-end Subscriptions
- Back-end Subscriptions
- **Subscription Aggregation**
- **Result Sharing**



BDPS: Information Flow and components



Guest Lecture

Data Distribution Service (DDS) from RTI Inc.

Guest Speaker: Dr. Kyle Benson

Student lecture

Kafka presentation

KSQL and ksqlDB

KSQL

- an open source streaming SQL engine that enables continuous, interactive queries on Apache Kafka
- continuously transforms streams of data -- take existing Apache Kafka® topics and filter, process them to create new derived topics

ksqlDB : integrates traditional database-like lookups on top of these materialized tables of data.

Videos..

Twitter experiences

https://www.youtube.com/watch?v=zwo3ipH4LZU

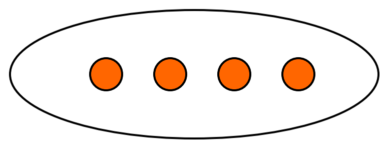
EXTRA SLIDES

Group Communication

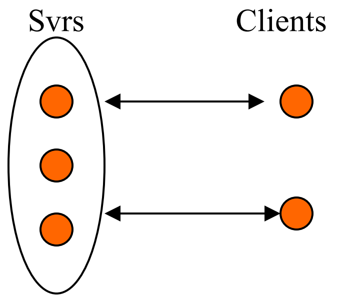
- Communication to a collection of processes – process group
- Group communication can be exploited to provide
  - Simultaneous execution of the same operation in a group of workstations
  - Software installation in multiple workstations
  - Consistent network table management
- Who needs group communication ?
  - Highly available servers
  - Conferencing
  - Cluster management
  - Distributed Logging….

Group communication - Types

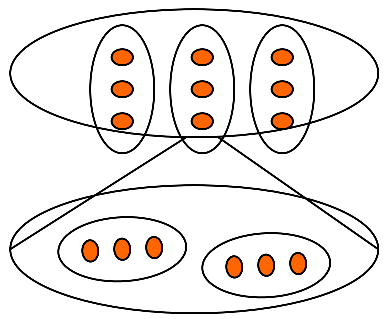
- Peer
  - All members are equal
  - All members send messages to the group
  - All members receive all the messages



- Client-Server
  - Common communication pattern
    - replicated servers
  - Client may or may not care which server answers



- Diffusion group
  - Servers sends to other servers and clients
- Hierarchical
  - Highly and easy scalable



Message Passing Basics

- A system is said to be asynchronous if there is no fixed upper bound on how long it takes a message to be delivered or how much time elapses between consecutive steps
- Point-to-point messages
  - $snd_i(m)$
  - $rcv_i(m,j)$
- Group communication
  - Broadcast
    - one-to-all relationship
  - Multicast
    - one-to-many relationship
    - A variation of broadcast where an object can target its messages to a specified subset of objects

Using Traditional Transport Protocols

- TCP/IP
  - Automatic flow control, reliable delivery, connection service, complexity
    - linear degradation in performance
- Unreliable broadcast/multicast
  - UDP, IP-multicast - assumes h/w support
  - message losses high(30%) during heavy load
    - Reliable IP-multicast very expensive

Group Communication Issues

- Ordering
- Delivery Guarantees
- Membership
- Failure

Ordering Service

- Unordered
- Single-Source FIFO (SSF)
  - For all messages m_1, m_2 and all objects a_i, a_j, if a_i sends m_1 before it sends m_2, then m_2 is not received at a_j before m_1 is
- Totally Ordered
  - For all messages m_1, m_2 and all objects a_i, a_j, if m_1 is received at a_i before m_2 is, the m_2 is not received at a_j before m_1 is
- Causally Ordered
  - For all messages m_1, m_2 and all objects a_i, a_j, if m_1 happens before m_2, then m_2 is not received at a_i before m_1 is

Delivery guarantees

- Agreed Delivery
  - guarantees total order of message delivery and allows a message to be delivered as soon as all of its predecessors in the total order have been delivered.
- Safe Delivery
  - requires in addition, that if a message is delivered by the GC to any of the processes in a configuration, this message has been received and will be delivered to each of the processes in the configuration unless it crashes.

Membership

- Messages addressed to the group are received by all group members
- If processes are added to a group or deleted from it (due to process crash, changes in the network or the user's preference), need to report the change to all active group members, while keeping consistency among them
- Every message is delivered in the context of a certain configuration, which is not always accurate. However, we may want to guarantee
  - Failure atomicity
  - Uniformity
  - Termination

Failure Model

- Failures types
  - Message omission and delay
    - Discover message omission and (usually) recovers lost messages
  - Processor crashes and recoveries
  - Network partitions and re-merges
- Assume that faults do not corrupt messages (or that message corruption can be detected)
- Most systems do not deal with Byzantine behavior
- Faults are detected using an unreliable fault detector, based on a timeout mechanism

Some GC Properties

- Atomic Multicast
  - Message is delivered to all processes or to none at all. May also require that messages are delivered in the same order to all processes.
- Failure Atomicity
  - Failures do not result in incomplete delivery of multicast messages or holes in the causal delivery order
- Uniformity
  - A view change reported to a member is reported to all other members
- Liveness
  - A machine that does not respond to messages sent to it is removed from the local view of the sender within a finite amount of time.

Virtual Synchrony

- Virtual Synchrony
  - Introduced in ISIS, orders group membership changes along with the regular messages
  - Ensures that failures do not result in incomplete delivery of multicast messages or holes in the causal delivery order(failure atomicity)
  - Ensures that, if two processes observe the same two consecutive membership changes, receive the same set of regular multicast messages between the two changes
    - A view change acts as a barrier across which no multicast can pass
  - Does not constrain the behavior of faulty or isolated processes

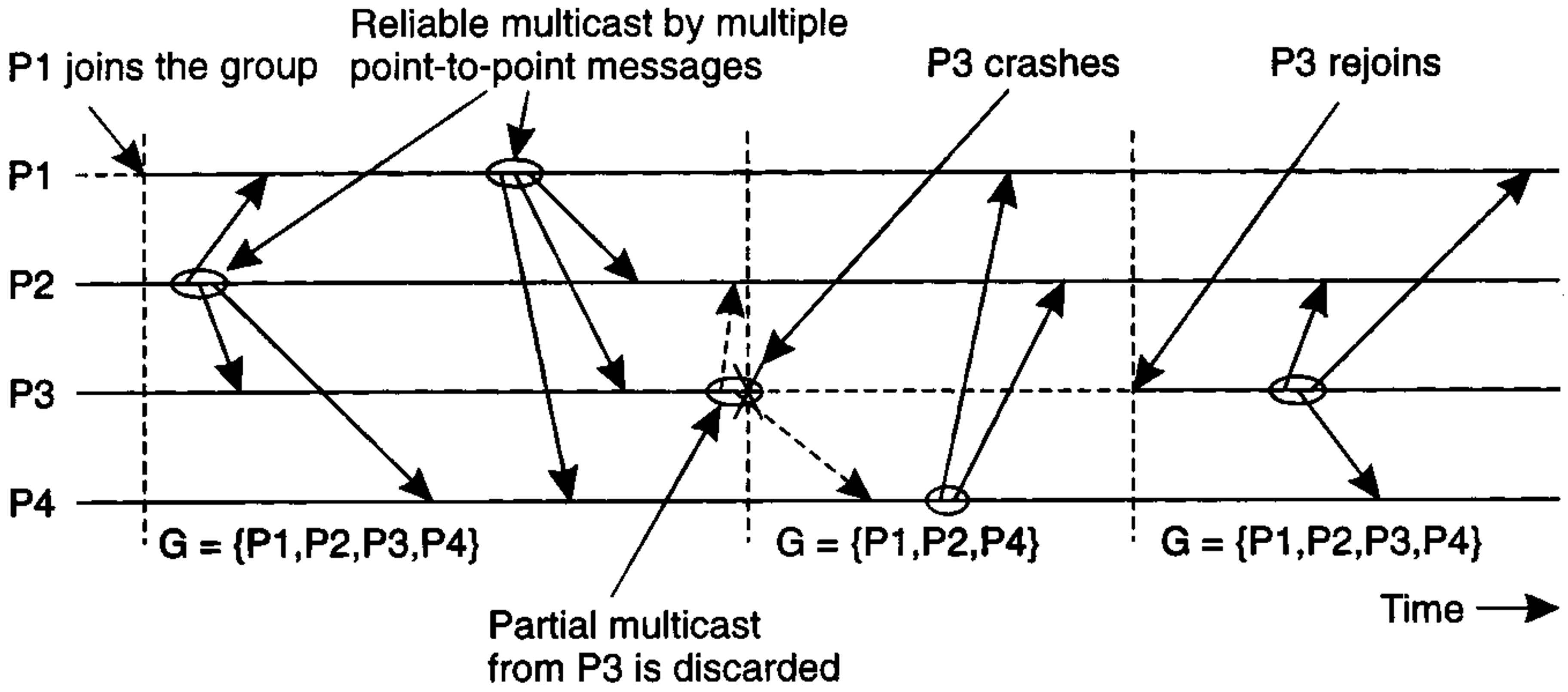


Figure 7-12. The principle of virtual synchronous multicast.

More Interesting GC Properties

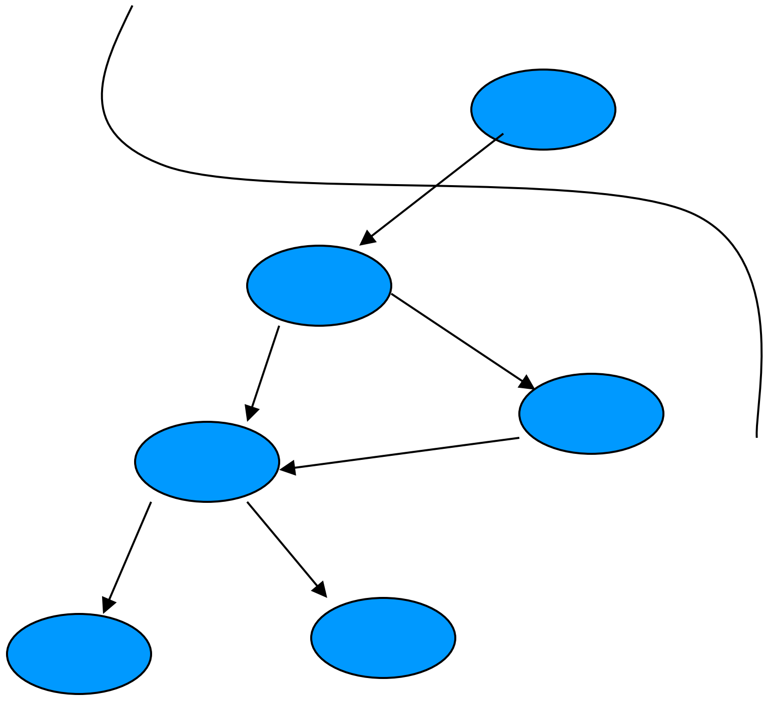
- There exists a mapping *k* from the set of messages appearing in all rcv$_i$(m) for all i, to the set of messages appearing in snd$_i$(m) for all i, such that each message *m* in a rcv() is mapped to a message with the same content appearing in an earlier snd() and:
- Integrity
  - *k* is well defined. *i.e.* every message received was previously sent.
- No Duplicates
  - *k* is one to one. *i.e.* no message is received more than once
- Liveness
  - *k* is onto. *i.e.* every message sent is received

Reliability Service

- A service is reliable (in presence of *f* faults) if exists a partition of the object indices into faulty and non-faulty such that there are at most *f* faulty objects and the mapping of *k* must satisfy:
  - Integrity
  - No Duplicates
    - no message is received more than once at any single object
  - Liveness
    - Non-faulty liveness
      - When restricted to non-faulty objects, *k* is onto. *i.e.* all messages broadcast by a non-faulty object are eventually received by all non-faulty objects
    - Faulty liveness
      - Every message sent by a faulty object is either received by all non-faulty objects or by none of them

Faults and Partitions

- When detecting a processor P from which we did not hear for a certain timeout, we issue a fault message
- When we get a fault message, we adopt it (and issue our copy)
- Problem: maybe P is only slow
- When a partition occurs, we can not always completely determine who received which messages (there is no solution to this problem)



Extended virtual synchrony

- Failures
  - Processes can fail and recover
  - Networks can partition and remerge
- Virtual synchrony handles recovered processes as new processes
  - Can cause inconsistencies with network partitions
- Network partitions are real
  - Gateways, bridges, wireless communication
- Extended VS (introduced in Totem)
  - Does not solve all the problems of recovery in fault-tolerant distributed systems, but avoids inconsistencies

Extended Virtual Synchrony Model

- Network may partition into finite number of components
  - Two or more may merge to form a larger component
- Each membership with a unique identifier is a *configuration.*
    - Membership ensures that all processes in a configuration agree on the membership of that configuration

Regular and Transitional Configurations

- To achieve safe delivery with partitions and remerges, the EVS model defines:
  - Regular Configuration
    - New messages are broadcast and delivered
    - Sufficient for FIFO and causal communication modes
  - Transitional Configuration
    - No new messages are broadcast, only remaining messages from prior regular configuration are delivered.
  - Regular configuration may be followed and preceeded by several transitional configurations.

Configuration change

- Process in a regular or transitional configuration can deliver a configuration change message s.t.
    - Follows delivery of every message in the terminated configuration and precedes delivery of every message in the new configuration.
- Algorithm for determining transitional configuration
  - When a membership change is identified
    - Regular conf members (that are still connected) start exchanging information
    - If another membership change is spotted (e.g. failure cascade), this process is repeated all over again.
    - Upon reaching a decision (on members and messages) – process delivers transitional configuration message to members with agreed list of messages.
    - After delivery of all messages, new configuration is delivered.

Totem

- Provides a Reliable totally ordered multicast service over LAN
- Intended for complex applications in which fault-tolerance and soft real-time performance are critical
  - High throughput and low predictable latency
  - Rapid detection of, and recovery from, faults
  - System wide total ordering of messages
  - Scalable via hierarchical group communication
  - Exploits hardware broadcast to achieve high-performance
- Provides 2 delivery services
  - Agreed
  - Safe
- Use timestamp to ensure total order and sequence numbers to ensure reliable delivery

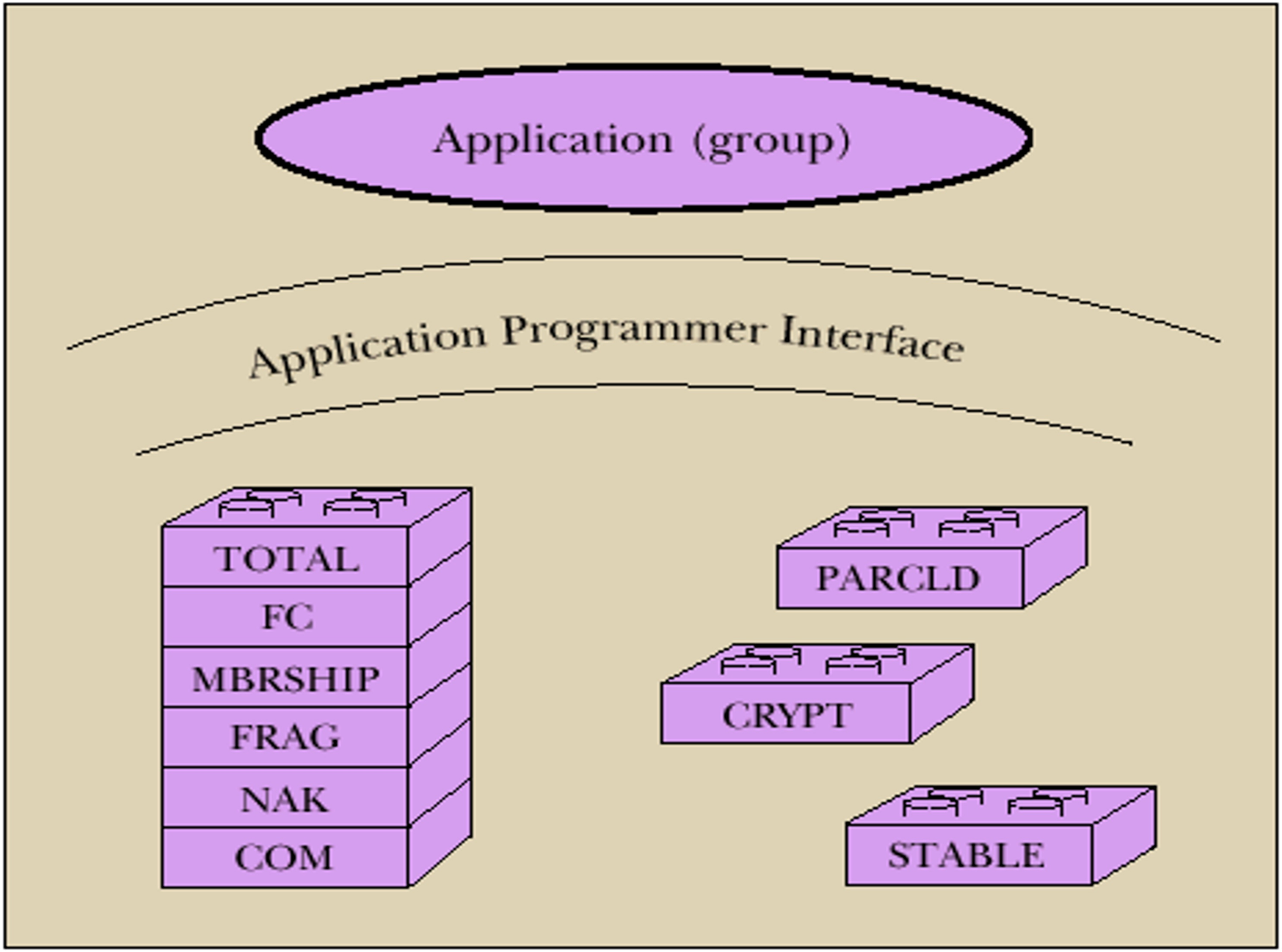
ISIS

- Tightly coupled distributed system developed over loosely coupled processors
- Provides a toolkit mechanism for distributing programming, whereby a DS is built by interconnecting fairly conventional non-distributed programs, using tools drawn from the kit
- Define
  - how to create, join and leave a group
  - group membership
  - virtual synchrony
- Initially point-to-point (TCP/IP)
- Fail-stop failure model

Horus

The Horus Project

- Aims to provide a very flexible environment to configure group of protocols specifically adapted to problems at hand
- Provides efficient support for virtual synchrony
- Replaces point-to-point communication with group communication as the fundamental abstraction, which is provided by stacking protocol modules that have a uniform (upcall, downcall) interface
- Not every sort of protocol blocks make sense
- HCPI - Horus Common Protocol Interface for protocol composition
  - Stability of messages
  - membership
- Electra
  - CORBA-Compliant interface
  - method invocation transformed into multicast


Application (group)
Application Programmer Interface
TOTAL
FC
MBRSHIP
FRAG
NAK
COM
PARCLD
CRYPT
STABLE

Transis

- How different components of a partition network can operate autonomously and then merge operations when they become reconnected ?
- Are different protocols for fast-local and slower-cluster communication needed ?
- A large-scale multicast service designed with the following goals
  - Tackling network partitions and providing tools for recovery from them
  - Meeting needs of large networks through hierarchical communication
  - Exploiting fast-clustered communication using IP-Multicast
- Communication modes
  - FIFO
  - Causal
  - Agreed
  - Safe

Other Challenges

- Secure group communication architecture
- Formal specifications of group communication systems
- Support for CSCW and multimedia applications
- Dynamic Virtual Private Networks
- Next Generations
  - Spread
  - Ensemble
  - MaelStrom, Ricochet - for cloud data centers
- Wireless networks
  - Group based Communication with incomplete spatial coverage
  - Dynamic membership

*VSync - ISIS2 (VS + Paxos) https://www.youtube.com/watch?v=3o81K1olx0Q

Horus

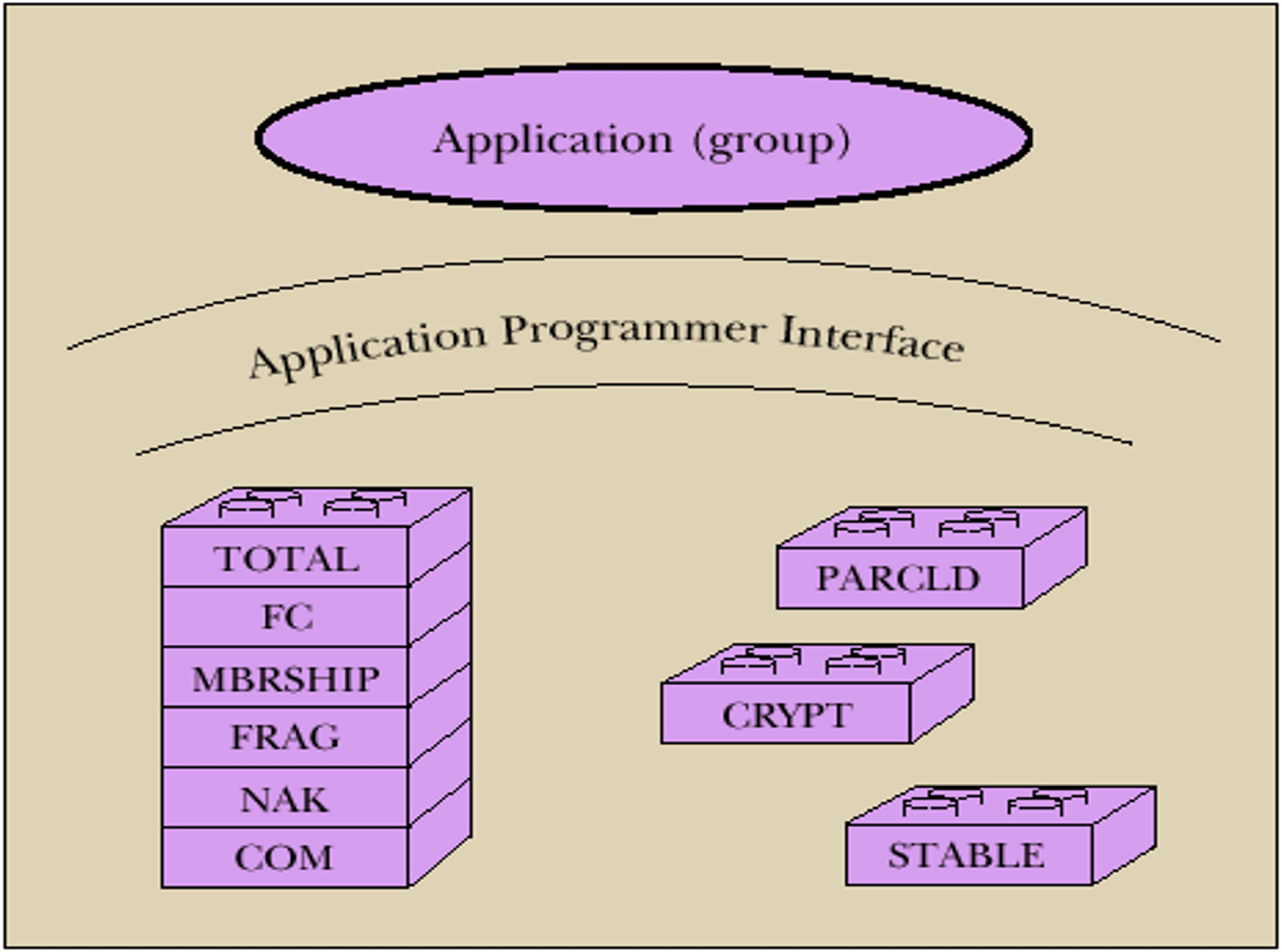
A Flexible Group Communication Subsystem

Horus: A Flexible Group Communication System

- Flexible group communication model to application developers.
  1. System interface
  2. Properties of Protocol Stack
  3. Configuration of Horus
     - Run in userspace
     - Run in OS kernel/microkernel

Architecture

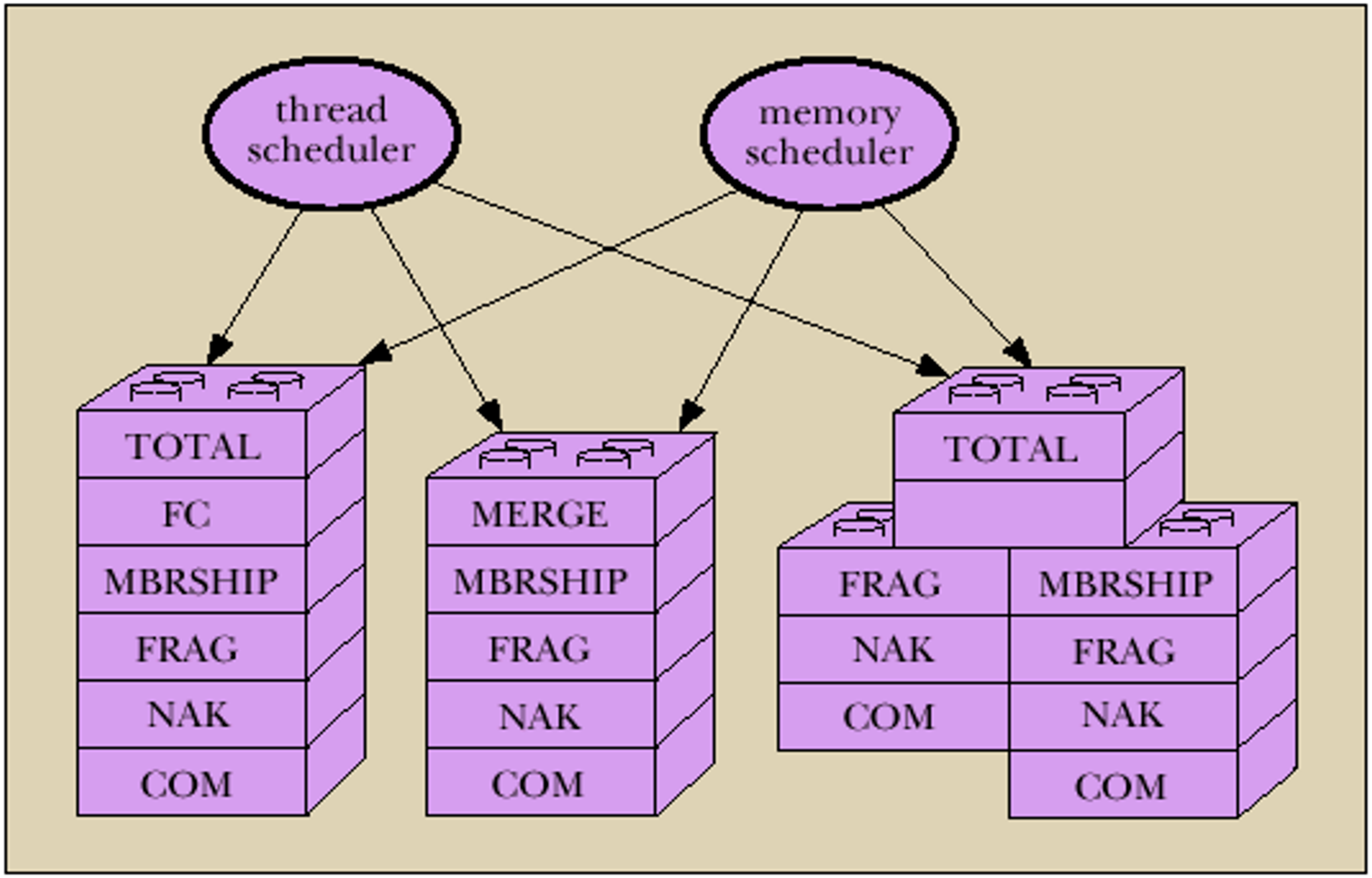
- Central protocol => Lego Blocks
- Each Lego block implements a communication feature.
- Standardized top and bottom interface (HCPI)
  - Allow blocks to communicate
  - A block has entry points for upcall/downcall
  - Upcall=receive mesg, Downcall=send mesg.
- Create new protocol by rearranging blocks.


Application (group)
Application Programmer Interface
TOTAL
FC
MBRSHIP
FRAG
NAK
COM
PARCLD
CRYPT
STABLE

Message_send

- Lookup the entry in topmost block and invokes the function.
- Function adds header
- Message_send is recursively sent down the stack
- Bottommost block invokes a driver to send message.

- Each stack shielded from each other.
- Have own threads and memory scheduler



Endpoints, Group, and Message Objects

- Endpoints
  - Models the communicating entity
  - Have address (used for membership), send and receive messages
- Group
  - Maintain local state on an endpoint.
  - Group address: to which message is sent
  - View: List of destination endpoint addr of accessible group members
- Message
  - Local storage structure
  - Interface includes operation pop/push headers
  - Passed by reference

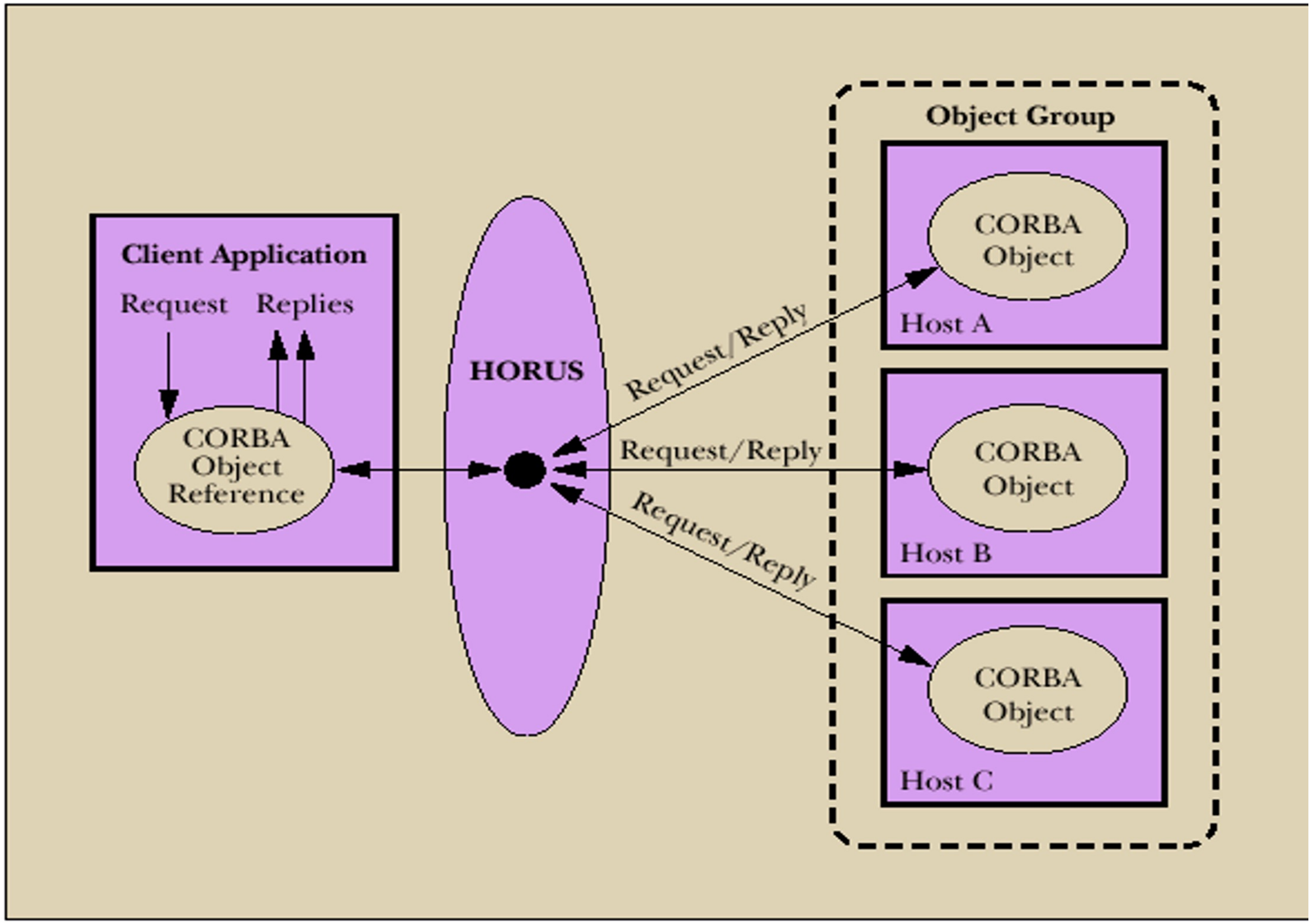


Figure 5. Object group communication in Electra

Transis

A Group Communication Subsystem

Transis : Group Communication System

- Network partitions and recovery tools.
  - Multiple disconnected components in the network operate autonomously.
  - Merge these components upon recovery.
- Hierachical communication structure.
- Fast cluster communication.

Systems that depend on primary component:

- Isis System: Designate 1 component as primary and shuts down non-primary.
  - Period before partition detected, non-primaries can continue to operate.
  - Operations are inconsistent with primary
- Trans/Total System and Amoeba:
  - Allow continued operations
  - Inconsistent Operations may occur in different parts of the system.
  - Don't provide recovery mechanism

Group Service

- Work of the collection of group modules.
- Manager of group messages and group views
- A group module maintains
  - Local View: List of currently connected and operational participants
  - Hidden View: Like local view, indicated the view has failed but may have formed in another part of the system.

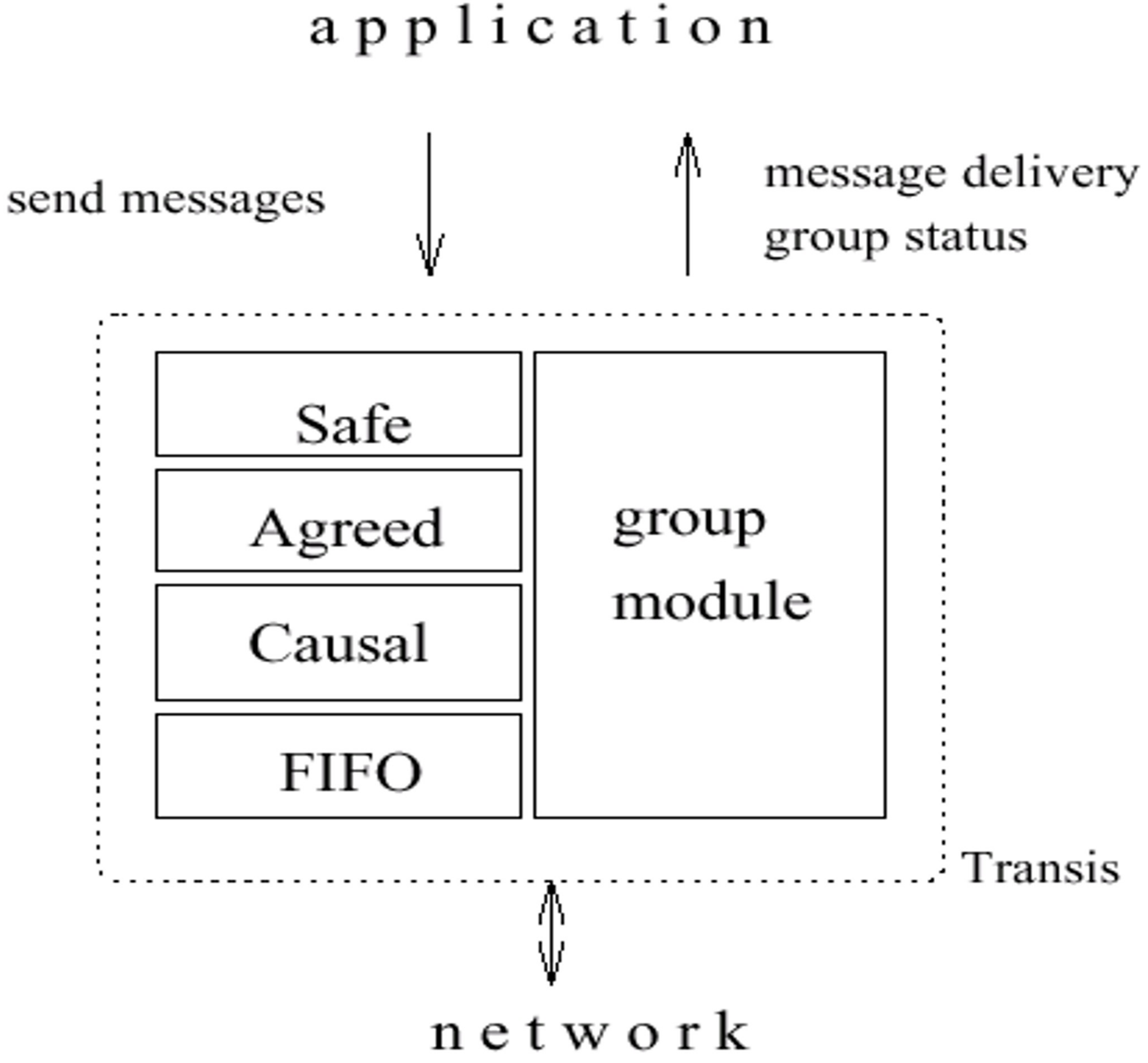


Figure 1: The System Model Structure

Network partition wishlist

1. At least one component of the network should be able to continue making updates.
2. Each machine should know about the update messages that reached all of the other machines before they were disconnected.
3. Upon recovery, only the missing messages should be exchanged to bring the machines back into a consistent state.

Transis supports partition

- Not all applications progress is dependent on a primary component.
- In Transis, local views can be merged efficiently.
  - Representative replays messages upon merging.
- Support recovering a primary component.
  - Non-primary can remain operational and wait to merge with primary
  - Non-primary can generate a new primary if it is lost.
    - Members can totally-order past view changes events. Recover possible loss.
    - Transis report Hidden-views.

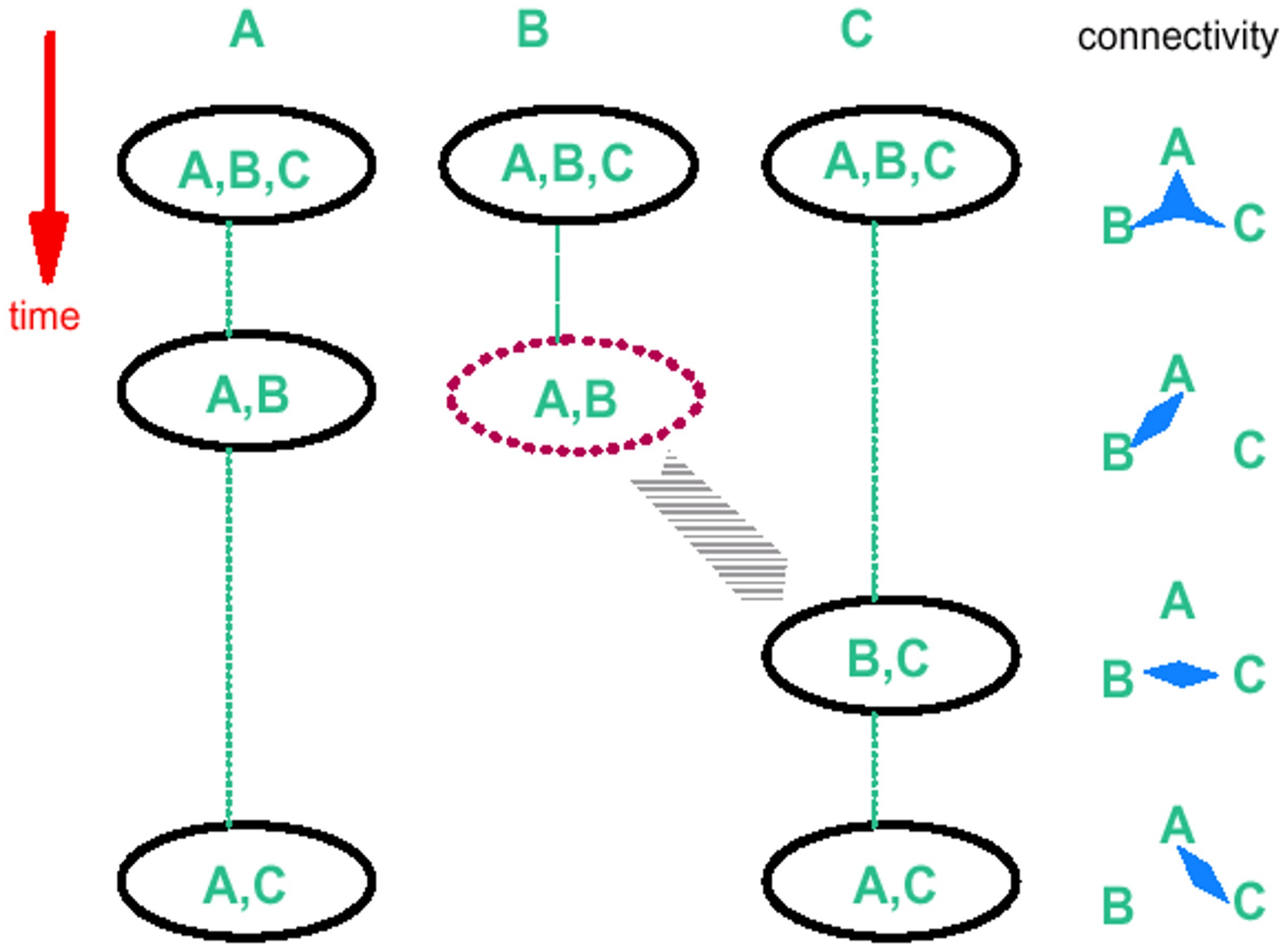


Figure 3: Breaking the symmetry between A and C