Distributed Systems (CS 543)

Networks, RMI, & Pub/Sub

Dongman Lee

Dept of CS

KAIST

Class Overview

- Introduction
- Networking Issues for Distributed Systems
- Network Basics
- Remote Method Invocation
- Message-based Invocation: Publish/Subscribe

Introduction - Definition

- Definitions
  - A set of machines (computers, phones, routers, switches, gateways, etc.) connected by communication links
  - A communication subsystem
- Why networks?
  - Sharing resources
  - Increasing proximity between people

Introduction - Network Basics

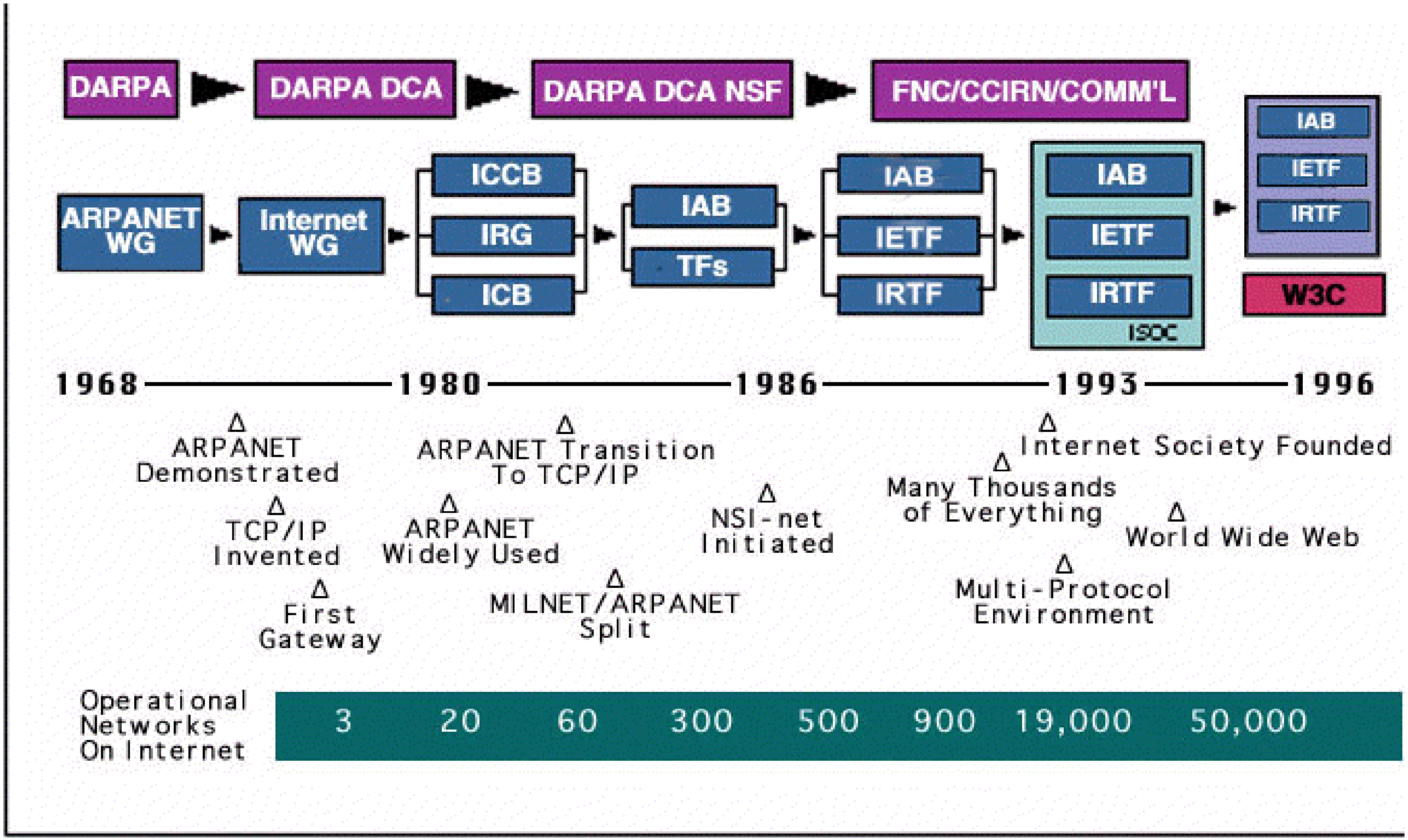
	Example	*Range*	*Bandwidth (Mbps)*	*Latency (ms)*
Wired:				
LAN	Ethernet	1-2 kms	10-1000	1-10
WAN	IP routing	worldwide	0.010-600	100-500
MAN	ATM	250 kms	1-150	10
Internetwork	Internet	worldwide	0.5-600	100-500
Wireless:				
WPAN	Bluetooth (802.15.1)	10 - 30m	0.5-2	5-20
WLAN	WiFi (IEEE 802.11)	0.15-1.5 km	2-54	5-20
WMAN	WiMAX (802.16)	550 km	1.5-20	5-20
WWAN	GSM, 3G phone nets	worldwide	0.01-02	100-500

Introduction - History of Networking

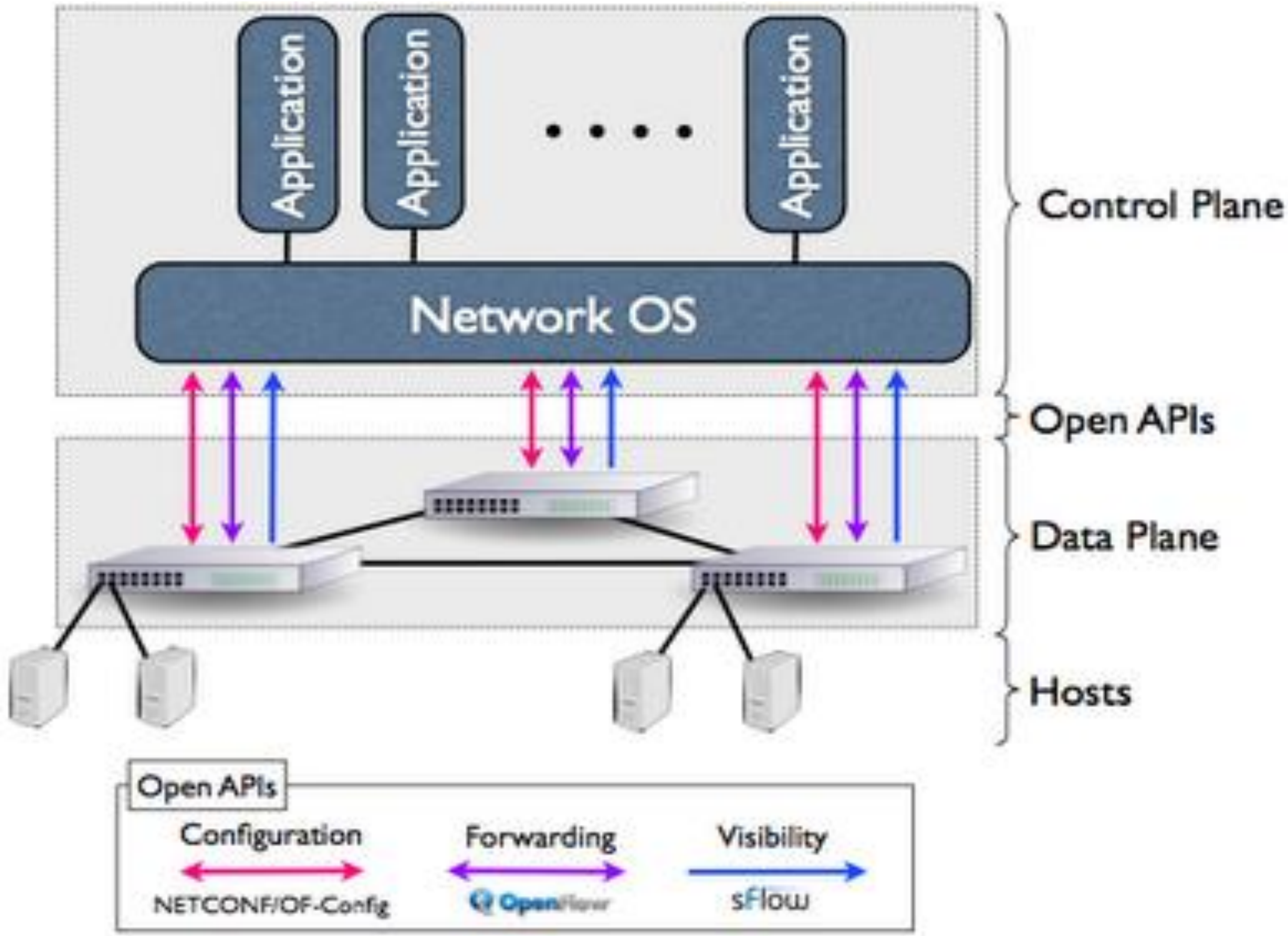
- Initially circuit switching (PSTN)
- 1960’s: packet-switching
  - ARPAnet (1969)
- 1970’s: local area network (LAN)
  - Ethernet
- 1980’s: workstations & PCs
  - proliferation of LANs and WANs
- 1990’s: WWW
  - explosion of Internet nodes; mobile networks
- 2000’s: ad-hoc , sensor networks, PAN
  - Spontaneous; infra-less; energy efficiency
  - Programmable network: software defined network (SDN)

Introduction - History of Internet

- http://www.isoc.org/internet/history



Software Defined Networking



http://blog.sflow.com/2012_05_01_archive.html

Networking Issues for Distributed Systems

- Performance
  - transmission delay is arbitrary but finite
  - transmission time
    - latency + message size/ data transfer rate
  - total system bandwidth
    - LAN vs. WAN
  - local vs. remote operations
- Scalability
  - number of hosts and networks
    - addressing, routing mechanism, domain names
  - Latency
- Security
  - firewall
  - VPN

Networking Issues for Distributed Systems (cont.)

- Reliability
  - hardware medium does not support replication
  - error recovery
    - message lost
    - message duplication
    - message out-of-order
    - message corruption
- Quality of service
  - bandwidth, latency, jitter, and reliability
  - admission control and resource reservation
- Mobility
  - location management
  - hand-off
  - energy

Protocols

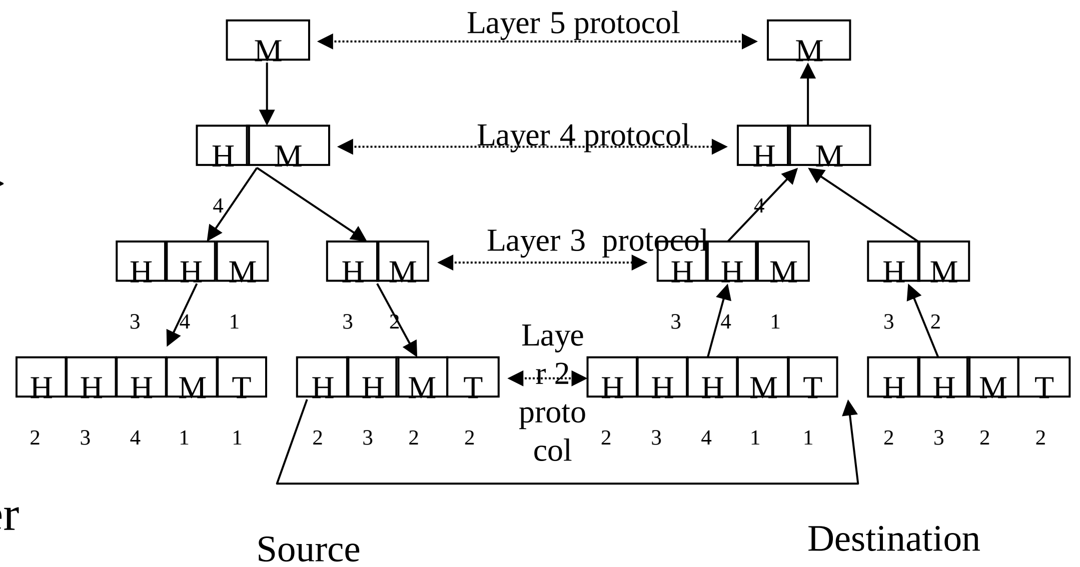
- A protocol specifies
  - rules and formats used for interactions between two parties
    - specification of the sequences of messages
    - specification of the format of the data in the message
- Requirements
  - specifications must be represented in finite states
  - implementation independent

Layered Architecture

- Motivation
  - reducing complexity by modularizing tasks vertically
  - encapsulation by a well-defined service interface to the upper layer
    - layer N provides a service to layer N+1
    - layer N extends the service of layer N-1
  - independence of each layer allows various implementations across layers
- Protocol suites
  - a complete set of protocol layers
  - protocol stack

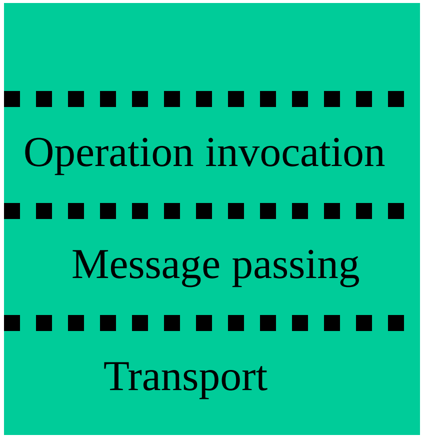
Services in Layered Protocols

- Data transport service types
  - connection-oriented vs. connectionless
  - virtual circuit vs. datagram
  - depends on quality of service (QoS) requirements from the layer above
- Packet assembly
  - message at layer N+1 is fragmented into multiple PDUs at layer N (MTU at layer N+1 > MTU at layer N)
  - layer N at peer: responsible for assembly
- Addressing
  - identification of the peer at layer N
  - conversion into layer N-1 address



Abstraction of Communication Subsystem

- Regards a network as another I/O device
  - no abstraction beyond a transport protocol
    - client and server are wholly responsible for message exchange over a given transport mechanism (e.g. TCP/IP)
    - primitives: write (send) and read (receive)
- Request and reply
  - abstraction of message passing required to execute a procedure at a server
    - primitives: DoOperation, GetRequest, and SendReply
- Remote method invocation
  - hides the separation between a client and a server
    - makes the invocation of a remote method residing at a server the same as that of a local one

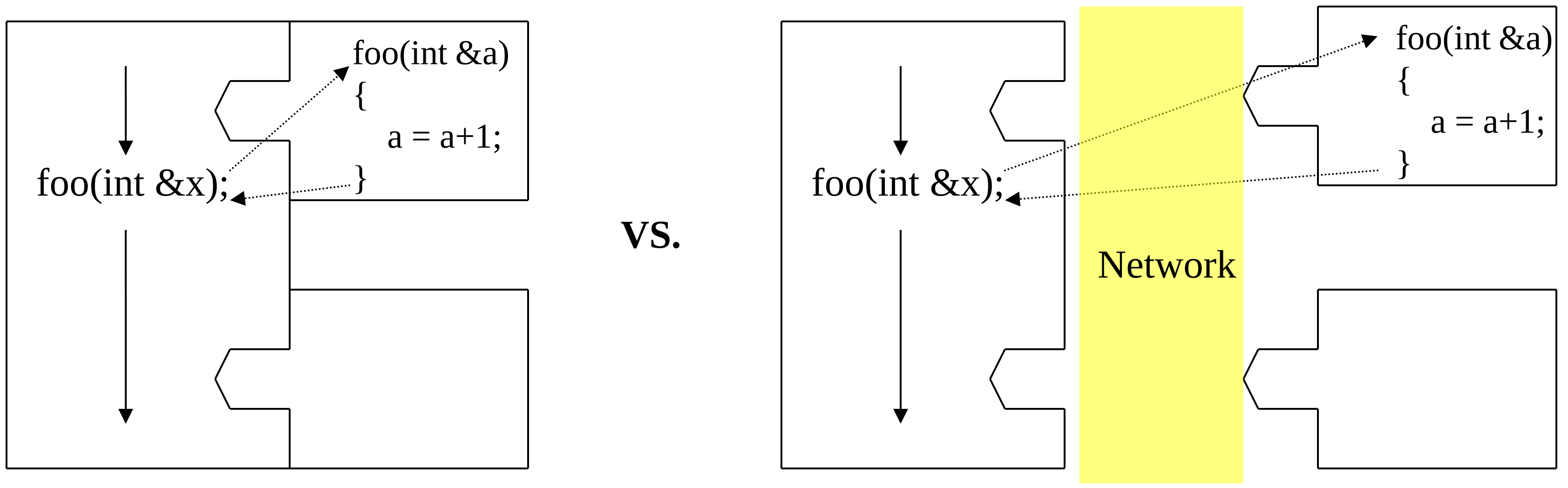


RMI- Motivation

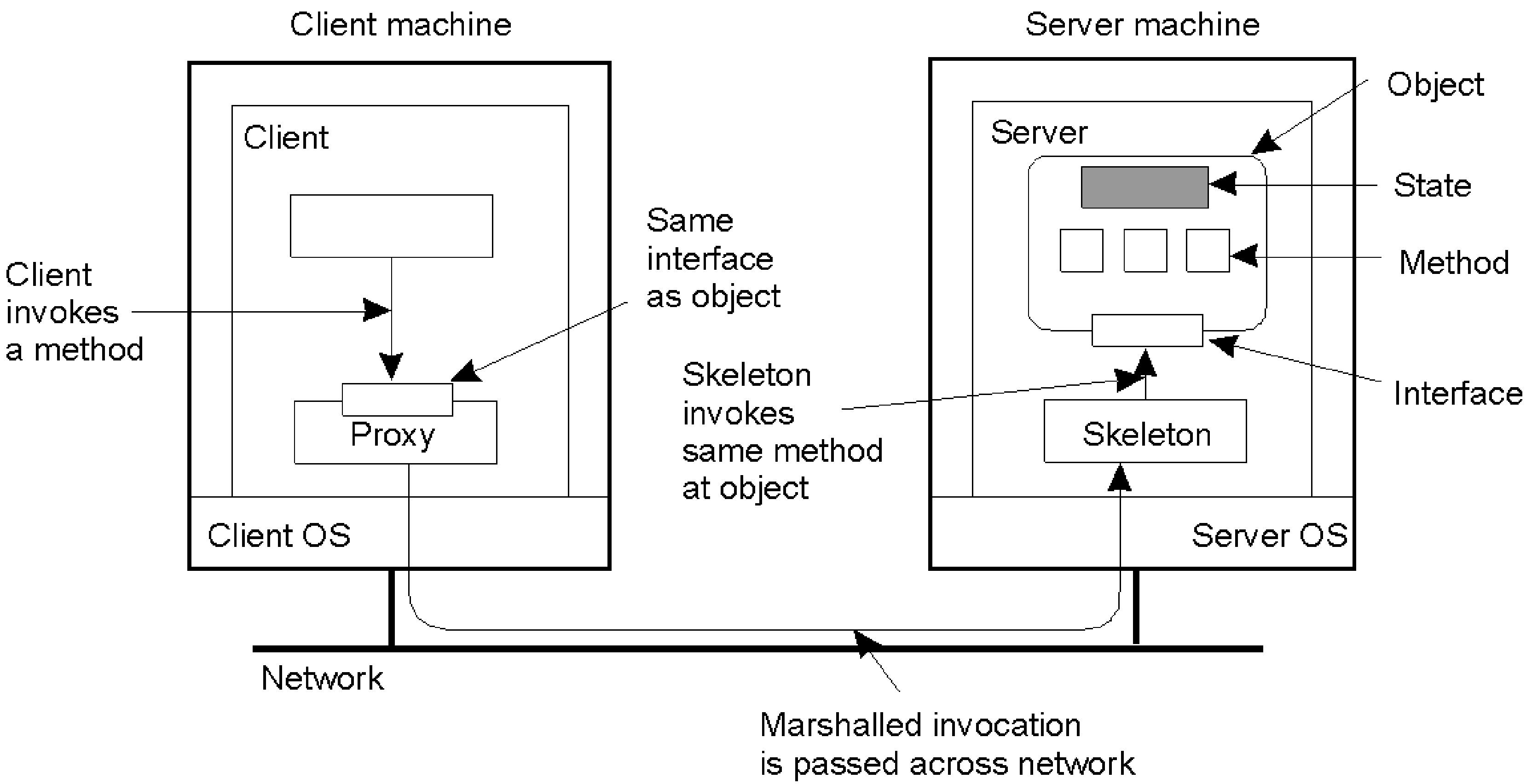
- Construction of distributed applications is a difficult task
  - Heterogeneity
  - Separation
  - Partial failure
- Leverage a local method invocation [Birrell & Nelson]
  - method invocations are a well-known and well-understood mechanism for transfer of control and data within a program
  - the same principle can be extended to the client-server interaction
    - clean and simple semantics
    - efficiency
    - generality

Local Invocation vs. Remote Invocation

- Client and server fail independently
  - extra exception handling
  - return value needs to be overloaded with failure codes from the server
- Global variables and pointers cannot be used across the interface



Remote Invocation Structure



Distributed object method invocation

Design Considerations

- Abstraction of RPC semantics
- Heterogeneity
- Parameter passing
- Binding
- Failure Transparency
- Support for concurrency

Level of Abstraction of RMI Semantics

- Integrate into the language
  - extend language to provide constructs for remote invocation semantics
  - examples: Cedar, Argus, Mercury, Java RMI
- Describe in a special purpose interface definition language
  - interface definition language supports language independence
    - limited representation of parameter types
  - examples:
    - Sun XDR, DCE IDL, Xerox Courier,
    - CORBA IDL, DCOM MIDL

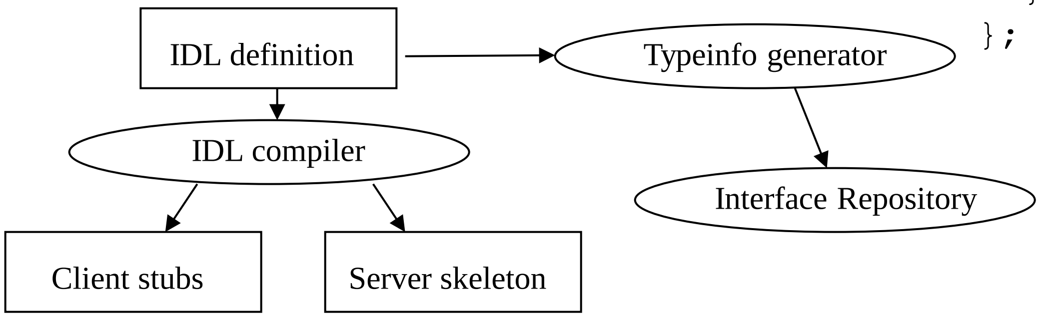
Level of Abstraction of RMI Semantics (cont.)

- IDL provides a distributed object with an abstract way to identify
  - what the server interface is going to be
  - what the interface will support
  - what other interface(s) it will collaborate with
- A client does not need to know how the interface is implemented
  - a client only incorporates the stub code generated from the IDL specification in the client implementation language
  - a client does not get any impact from change(s) in the server implementation as long as the server interface stays the same
  - the server can have multiple implementation of the same interface

Example: CORBA IDL

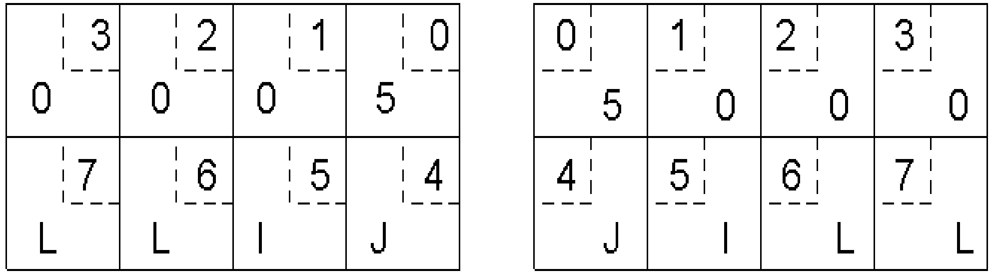
- IDL is the language used to describe the interfaces that client objects can call and object implementations provide
  - C++-like syntax
  - features
    - multiple inheritance
    - attributes
    - exceptions
    - contexts
- CORBA has two IDL compilers
  - stub & skeleton generator
  - typeinfo generator for dynamic invocation

```
module Shopping {
  /* structure of an item */
  struct Item {
    string item_name;
    float  item_price;
  };
  typedef sequence <Iteπm> Basket;
  /* class GoShopping */
  interface GoShopping {
    exception NotEnoughMoney
                                    {float
needed};
    attribute float money;
    void buy (in Basket, out Basket)
                  raises
(NotEnoughMoney);
  };
};
```



Heterogeneity

- Marshalling policy
  - conversion to the common data type agreed by both a sender and a receiver: Sun RPC, CORBA
  - a receiver makes it right (tagging): NCS RPC
  - a sender negotiates with a receiver

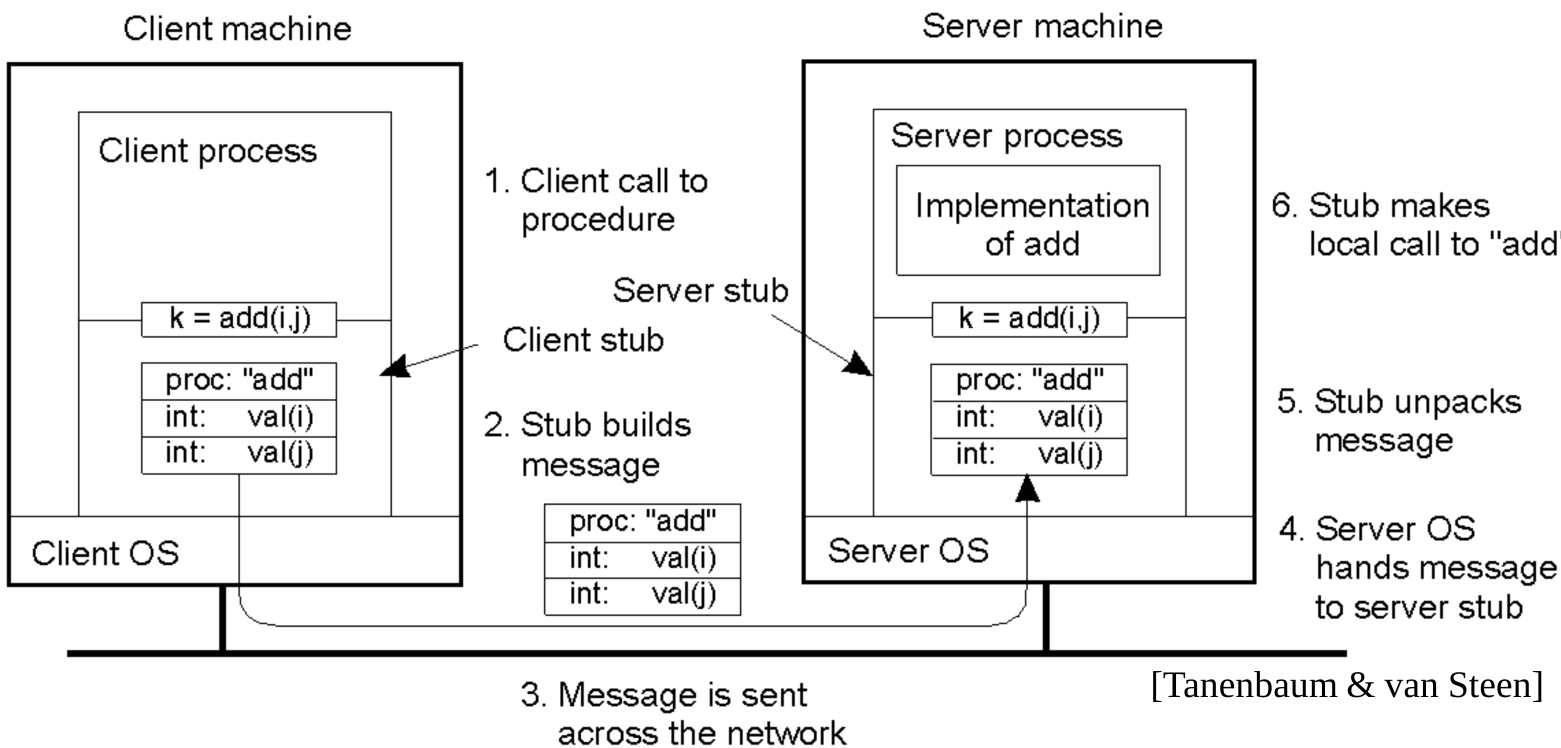


- Accommodate multiple transport protocols

Parameter Passing

- Considerations for parameter passing in RMI
  - direction of parameters
    - input: client -> server
    - output: server -> client
    - input & output: client <-> server
  - call-by-reference
    - use call-by-value-result instead
  - pointer parameters
    - no meaning to pass a pointer to memory location
    - linearize before passing it
  - procedure parameter
    - procedure should be accessible across the network

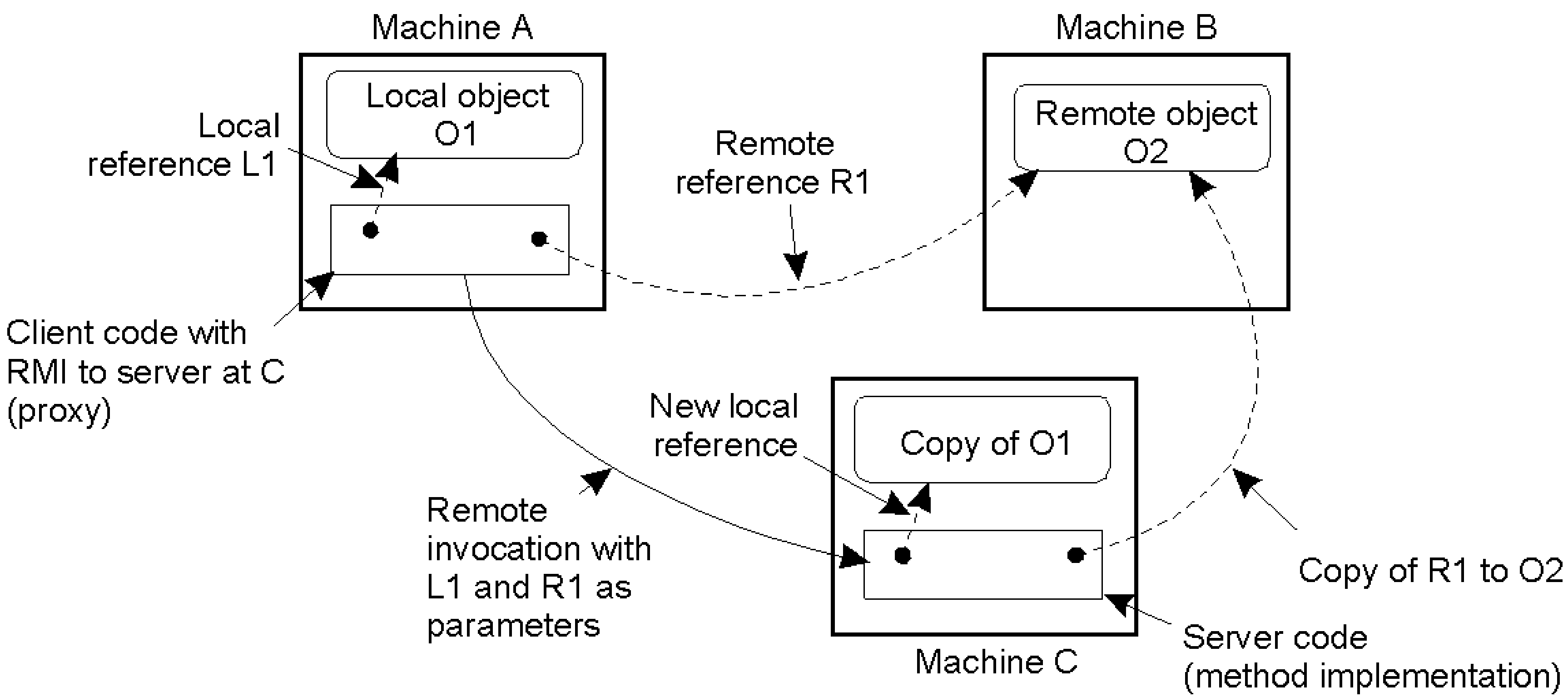
Parameter Passing (cont.)



[Tanenbaum & van Steen]

Parameter Passing (cont.)

- Distributed objects

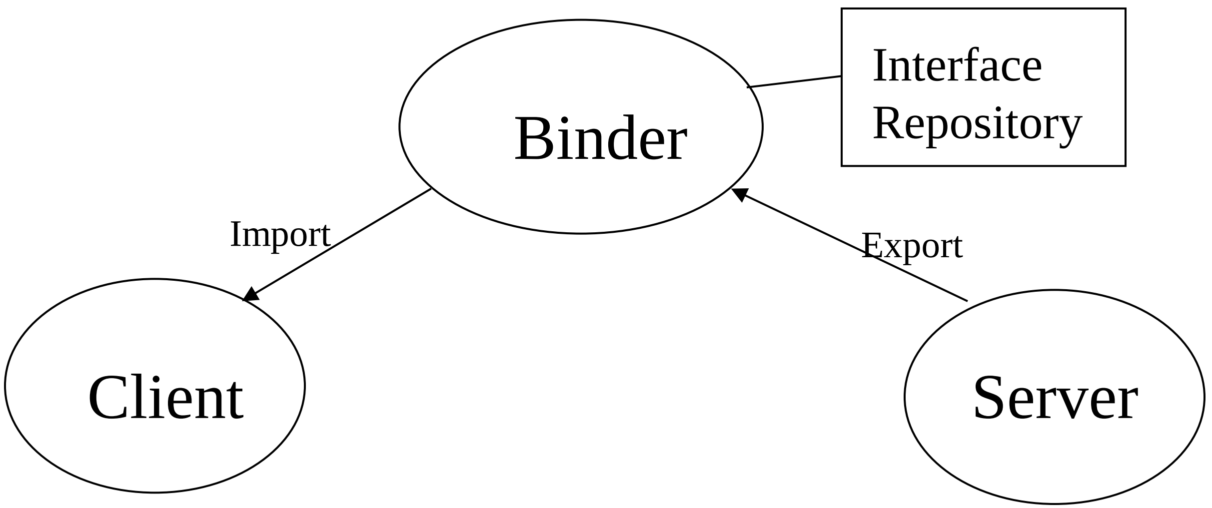


Binding

- Naming
  - mapping from a name to a particular service
    - static linking: most RPCs
    - dynamic linking: NCS RPC
    - procedure variable: Argus, Mercury
  - exportation and importation
    - server exports its service interface
    - client imports exported service
  - interface consists of
    - name: uniquely identifiable
      - version info and location info can be included
    - signature: parameter name & type

Binding (cont.)

- Locating a server
  - via a binder (like a directory service)
    - well-know address
    - run-time notification
    - broadcasting (recruiting)



Failure Transparency

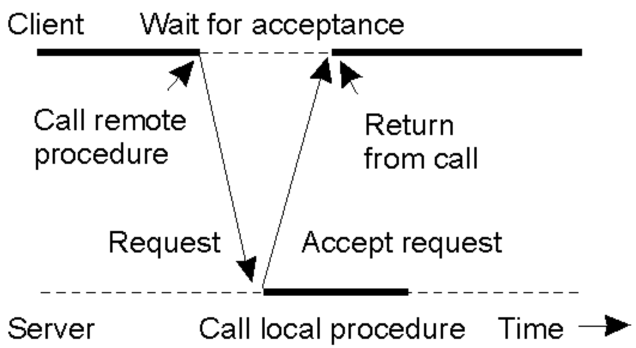
- Ordered delivery
  - request and reply delivered in order (e.g. Argus)
- RPC Execution semantics
  - How many times will a remote procedure be executed by the callee per call?
  - Semantics
    - at-least-once; at-most-once; exactly-once
- At least once
  - when a caller receives the expected result, it implies that the requested call has been executed one or more (at least once) at a callee
  - the callee does not remember multiple executions due to call duplication or retry after failure
  - acceptable if operations are idempotent
  - example: Sun RPC – NFS

Failure Transparency (cont.)

- At-most-once
  - when a caller receives the expected result, it implies that the requested call has been executed exactly once. If not completed, executed zero or one time
  - the callee can detect duplicated requests at the absence of failure but persist through the abnormal termination (failure amnesia)
  - example: NCS RPC
- Exactly-once
  - when a caller receives the expected result, it implies that the requested call has been executed exactly once. If not completed, executed zero times
  - the callee is failure-resilient: logging and two-phase commit
  - example: Argus

Support for Concurrency

- RMI behaves differently from LMI
  - the client does not need to be blocked while the server executes the requested call
  - the client may not need the result
- Approaches
  - Synchronous
    - a process/thread per call (at a client or a server)
  - Asynchronous
    - two types
      - no result
      - delayed result: delayed synchronous



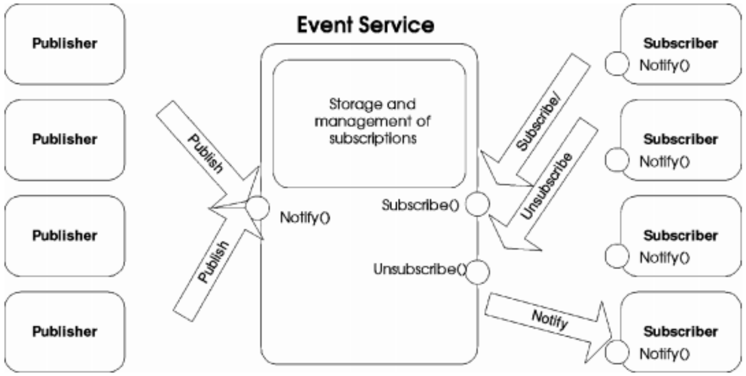
- Examples:
  - Argus, Mercury(Promises), X11 RPC, ANSA REX
  - CORBA
    - oneway

Why Publish/Subscribe?

- Demand for more flexible communication models
  - Individual *point-to-point* and *synchronous* communications lead to rigid and static applications in large-scale distributed systems
  - Increasing Scale of distributed systems
    - The Internet has considerably increased the scale of distributed systems
  - Dynamicity
    - Entities in distributed systems vary their location and behavior throughout the lifetime.

Publish/Subscribe

- Loosely coupled interaction scheme
  - Subscribers
    - Subscribers express their interest in any events, and are subsequently notified of matched events, generated by a publisher
  - Publishers
    - Publishers generate and publish new information denoted by an event
  - Event notification services
    - A neutral mediator between publishers and subscribers which provides storage and management for subscriptions and efficient delivery of events
  - Events
    - An event from a publisher is asynchronously propagated to all subscribers that registered interest in that given event

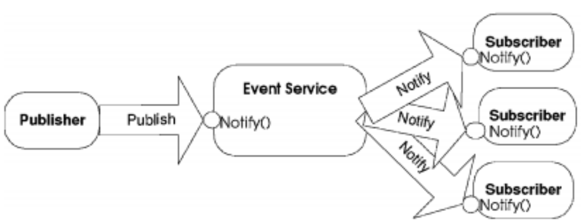


Event Service Decoupling

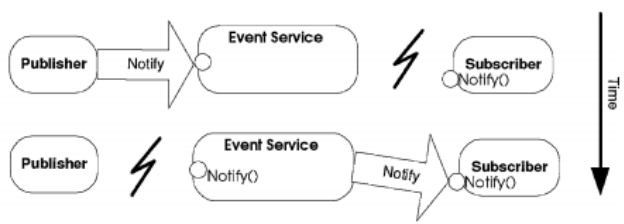
- Event service decoupling
  - Event services provide the full decoupling between publishers and subscribers
  - The full decoupling releases the burden of application designers in large-scale systems
  - The strength of publish/subscribe lies in decoupling:
    - Decoupling increases scalability by removing all explicit dependencies between the interacting participants
    - Dependency-free infrastructure can well adapted to distributed environments that are asynchronous by nature (e.g. Mobile environment)
  - Event service decoupling can be decomposed along the following three dimensions:
    - Time
    - Space
    - Synchronization

Event Service Decoupling (cont.)

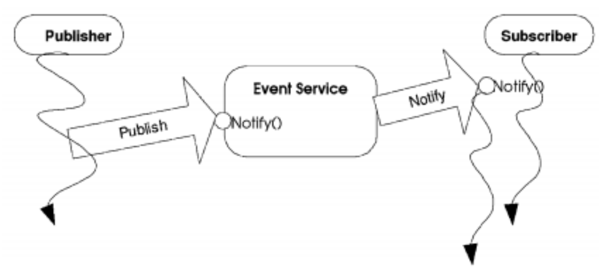
- Decoupling dimensions
  - Space
    - The interacting parties do not need to know each other’s references or number
  - Time
    - The interacting parties do not need to be actively participating in the interaction at the same time
    - Publishers can publish events while the subscribers are disconnected, and vice versa
  - Synchronization
    - Publishers are not blocked while producing events
    - Subscribers can get asynchronously notified of the occurrence of an event through a callback



Space decoupling



Time decoupling



Synchronization decoupling

Publish/Subscribe Variations

- Publish/Subscribe Variations
  - Subscribers are usually interested in particular events or event patterns, and not in all events
  - The different ways of specifying the events of interest have led to several subscription schemes:
    - Topic-based publish/subscribe
    - Content-based publish/subscribe
    - Type-based publish/subscribe

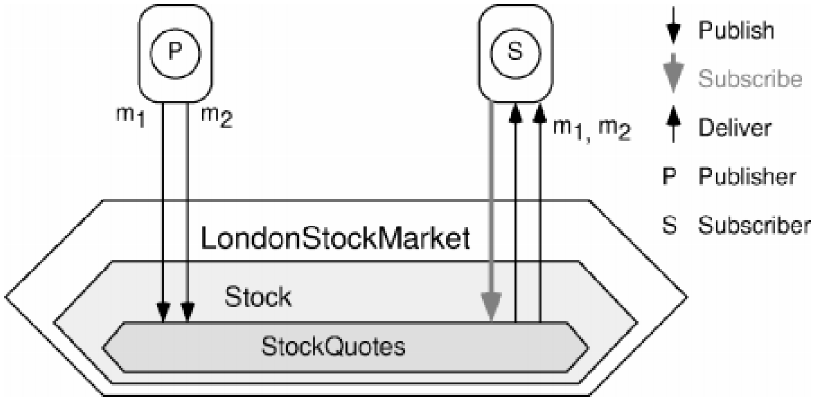
Publish/Subscribe Variations (Cont.)

- Topic-based Publish/Subscribe (*or Subject-based*)
  - The earliest publish/subscribe scheme which extends the notion of *channels*
  - Subscriptions
    - Participants can publish events and subscribe to individual topics, which are identified by *keywords*
    - Every topic is viewed as an event service of its own, identified by a unique name, keyword
  - Similarity to *group communications*
    - Subscribing to a topic T ≈ Becoming a member of a group T
    - Publishing an event on topic T ≈ Broadcasting that event among the members of T
  - Improvement: *Hierarchies*
    - Topics can be organized according to containment relationships (*hierarchical addressing*)
    - A subscription made to a node involves those to all the subtopics

Publish/Subscribe Variations (Cont.)

- Topic-based Publish/Subscribe (*or Subject-based*)
  - Sample code and an interaction example

```
public class StockQuote implements Serializable {
  public String id, company, trader;
  public float price;
  public int amount;
}
public class StockQuoteSubscriber implements Subscriber {
  public void notify(Object o) {
    if (((StockQuote)o).company == 'TELCO' && ((StockQuote)o).price < 100)
      buy();
  }
}
// ...
Topic quotes = EventService.connect("/LondonStockMarket/Stock/StockQuotes");
Subscriber sub = new StockQuoteSubscriber();
quotes.subscribe(sub);
```



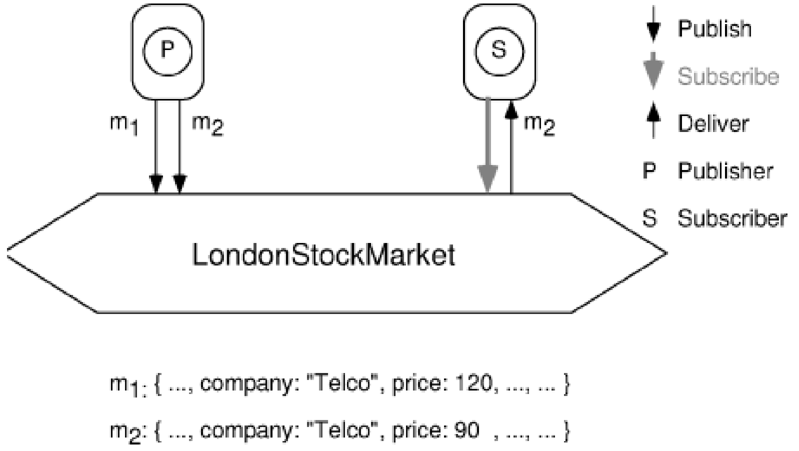
Publish/Subscribe Variations (Cont.)

- Content-based publish/subscribe (*or property-based*)
  - Topic-based scheme offers only limited expressiveness
  - Subscriptions
    - Subscribers subscribe to the actual content of the considered events
    - Events are not classified according to some predefined external criterion (e.g. topic name), but according to the properties of the events themselves
  - Selective subscriptions with *filters*
    - Subscribers specify filters using a subscription language
    - Filters define constraints, usually in the form of name-value pairs of properties and basic comparison operators (e.g., =, <, >, etc.)
    - Constraints can be logically combined (e.g., AND, OR)

Publish/Subscribe Variations (Cont.)

- Content-based publish/subscribe (*or property-based*)
  - Sample code and an interaction example

```
public class StockQuote implements Serializable {
  public String id, company, trader;
  public float price;
  public int amount;
}
public class StockQuoteSubscriber implements Subscriber {
  public void notify(Object o) {
    buy();      // company == 'TELCO' and price < 100
  }
}
// ...
String criteria = ("company == 'TELCO' and price < 100");
Subscriber sub = new StockQuoteSubscriber();
EventService.subscribe(sub, criteria);
```



Publish/Subscribe Variations (Cont.)

- Type-based publish/subscribe
  - Event type
    - Topics usually regroup events that present commonalities not only in content, but also in structure
    - A regrouped events which share the commonalities are said to be of the same *type*
  - Advantages
    - This enables a *closer integration* of the language and the middleware (e.g., an object written in a language has its own structure as well as contents)
    - *Type safety* can be ensured at compile-time by parameterizing the resulting abstraction interface by the type of the corresponding events

Publish/Subscribe Variations (Cont.)

- Type-based publish/subscribe
  - Sample code and an interaction example

```
public class LondonStockMarket implements Serializable {
  public String getId() {...}
}
public class Stock extends LondonStockMarket {
  public String getCompany() {...}
  public String getTrader() {...}
  public int getAmount() {...}
}
public class StockQuote extends Stock {
  public float getPrice() {...}
}
public class StockRequest extends Stock {
  public float getMinPrice() {...}
  public float getMaxPrice() {...}
}
public class StockSubscriber implements Subscriber<StockQuote> {
  public void notify(StockQuote s) {
    if (s.getCompany() == 'TELCO' && s.getPrice() < 100)
      buy();
  }
}
// ...
Subscriber<StockQuote> sub = new StockSubscriber();
EventService.subscribe<StockQuote>(sub);
```