Data Dissemination

Contents:

- Some Apps
- Nature of information
- Modes of interaction
- Data delivery options
- MOM
  - Addressing options

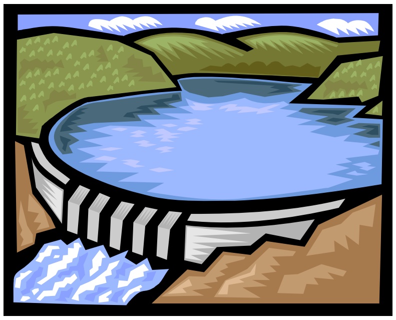
Mariano Cilia / mcilia@gmail.com

Motivated by Examples

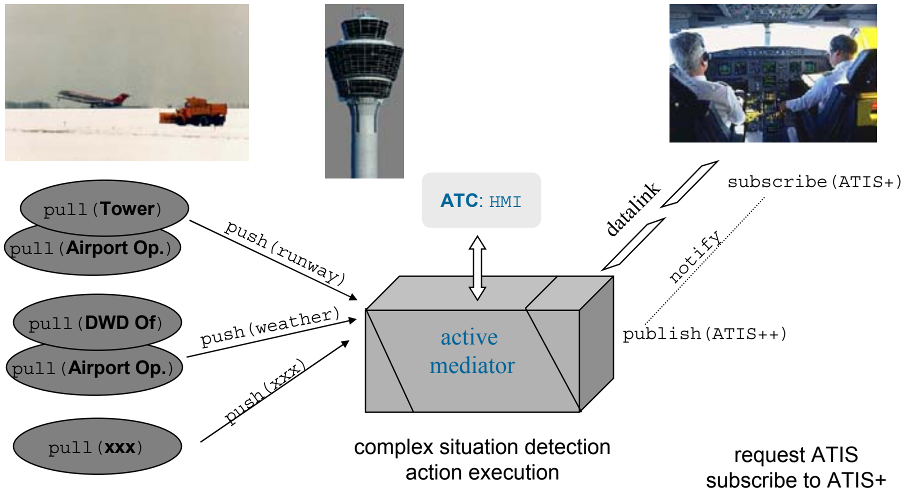
- Traffic management
  - Air-traffic control
  - Emergency vehicles
- Environmental
  - Concentration of chemicals
  - Dangerous emissions
  - Forest fires
  - Flood control

Example – Flood Control

- A combination of happenings (coming from sensor data) determines a situation
- Polling sensors
  - Too frequent
    - Consumption of resources
  - Too infrequent
    - Malfunction



Application scenario: ATIS



publisher and
event detection wrappers

complex situation detection
action execution
repository
configuration management

request ATIS
subscribe to ATIS+
notification

Examples (cont.)

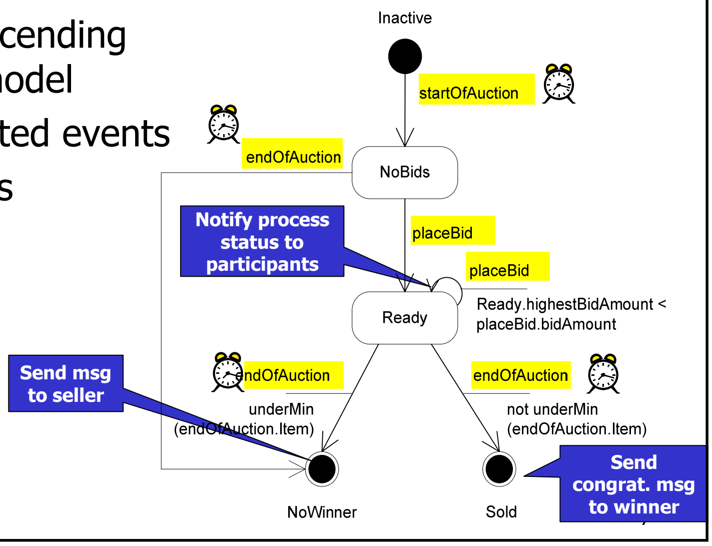
- Plant and reactor control
  - Equipment control
- Defense
  - Missile detection
  - Battlefield monitoring
- Workflow management

Examples (cont.)

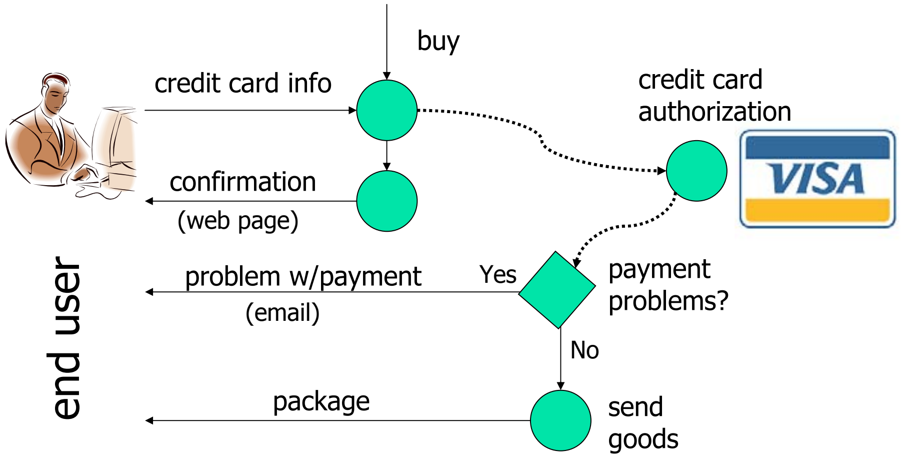
- Commerce
  - Inventory control
  - Supply Chain Management
  - Marketplaces
  - e-Auctions
  - Online shops

Example – e-Auctions

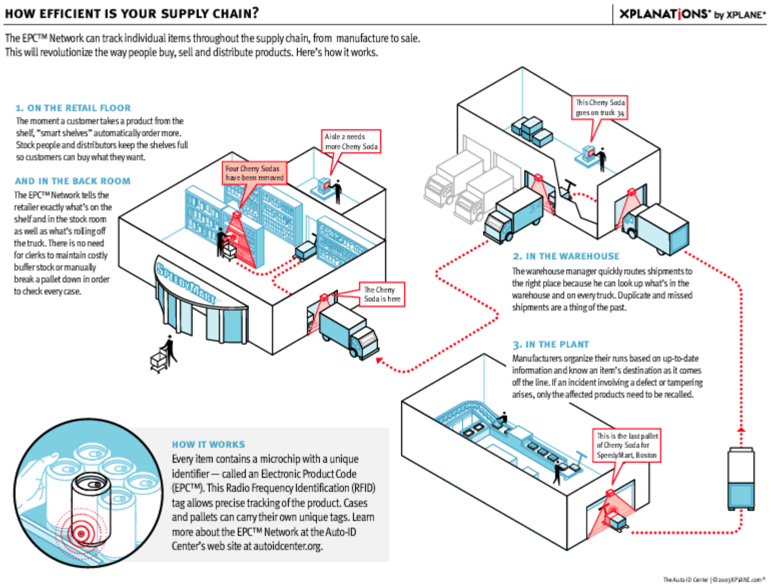
- Simple ascending auction model
- Time-related events
- Conditions


Inactive
startOfAuction
endOfAuction
NoBids
Notify process status to participants
placeBid
placeBid
Ready.highestBidAmount < placeBid.bidAmount
Ready
Send msg to seller
endOfAuction
underMin (endOfAuction.Item)
endOfAuction
not underMin (endOfAuction.Item)
Send congrat. msg to winner
NoWinner
Sold

Example – Online shops



RFID – Supply Chain Mgmt (cont)



HOW EFFICIENT IS YOUR SUPPLY CHAIN?

XPLANATIONS* by XPLANE*

The EPC™ Network can track individual items throughout the supply chain, from manufacture to sale. This will revolutionize the way people buy, sell and distribute products. Here's how it works.

1. ON THE RETAIL FLOOR

The moment a customer takes a product from the shelf, "smart shelves" automatically order more. Stock people and distributors keep the shelves full so customers can buy what they want.

AND IN THE BACK ROOM

The EPC™ Network tells the retailer exactly what's on the shelf and in the stock room as well as what's rolling off the truck. There is no need for clerks to maintain costly buffer stock or manually break a pallet down in order to check every case.

2. IN THE WAREHOUSE

The warehouse manager quickly routes shipments to the right place because he can look up what's in the warehouse and on every truck. Duplicate and missed shipments are a thing of the past.

3. IN THE PLANT

Manufacturers organize their runs based on up-to-date information and know an item's destination as it comes off the line. If an incident involving a defect or tampering arises, only the affected products need to be recalled.

HOW IT WORKS

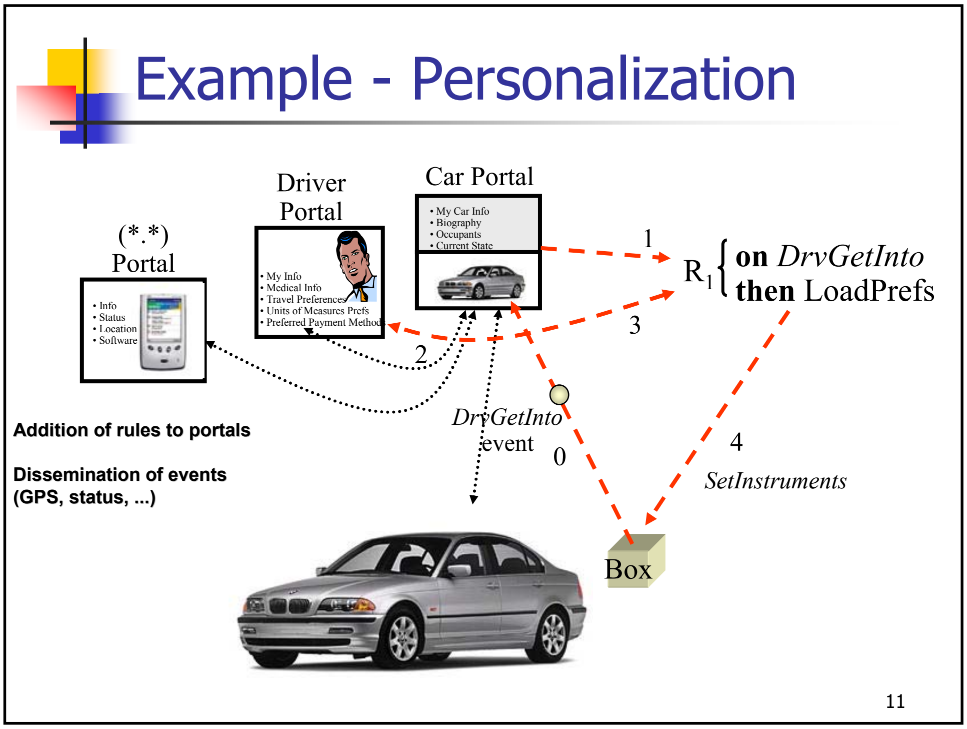
Every item contains a microchip with a unique identifier — called an Electronic Product Code (EPC™). This Radio Frequency Identification (RFID) tag allows precise tracking of the product. Cases and pallets can carry their own unique tags. Learn more about the EPC™ Network at the Auto-ID Center's web site at autoidcenter.org.

Aisle 2 needs more Cherry Soda
Four Cherry Sodas have been removed
SpeedyMart
The Cherry Soda is here
This Cherry Soda goes on truck 34
34
This is the last pallet of Cherry Soda for SpeedyMart, Boston

The Auto-ID Center | ©2003XPLANE.com®

Examples (cont.)

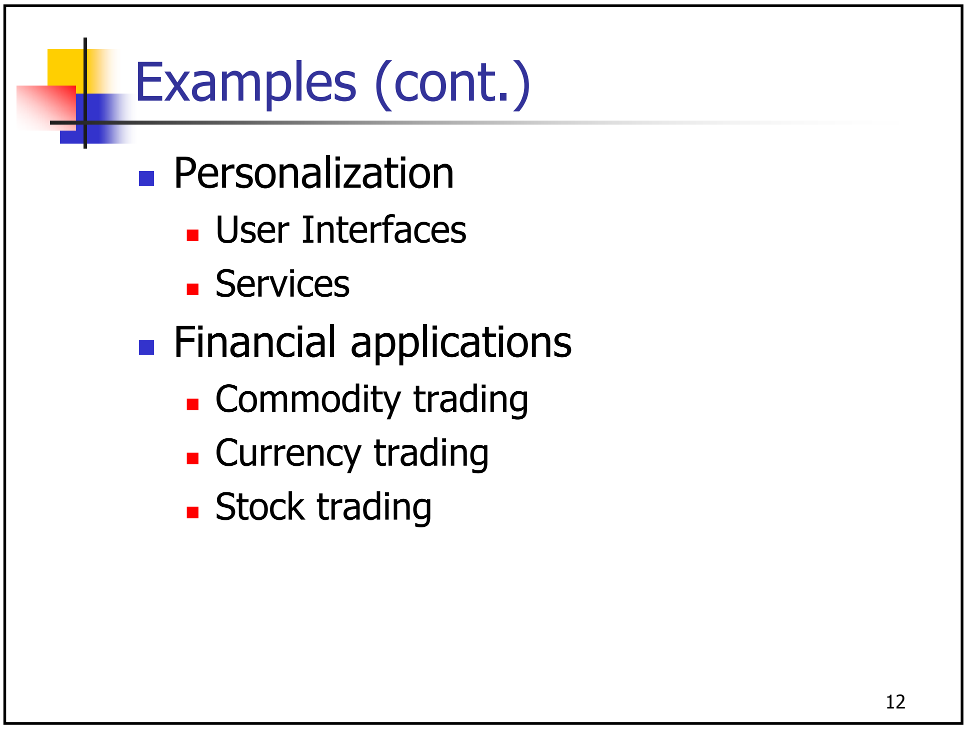
- Personalization
  - User Interfaces
  - Services



Example - Personalization

Addition of rules to portals

Dissemination of events (GPS, status, ...)



Examples (cont.)

- Personalization
  - User Interfaces
  - Services
- Financial applications
  - Commodity trading
  - Currency trading
  - Stock trading

Example – Stock trading

- Sample
  - **ON** stock.Name=IBM
    IF stock.Price<20
    THEN call myBuy()
- High volume

Convergence of Technologies

- Ambient Intelligence and smart devices require continuous monitoring of events
- Miniaturization of sensors, ubiquitous deployment
- Context information for proper interpretation
- (Almost) complete reachability of individuals causes unbounded appetite for information
- Need to filter and interpret large amounts of heterogeneous and short-lived data
- Large distributed systems must detect and correct failures/exceptions (autonomic computing, ESCM, zero latency enterprise)

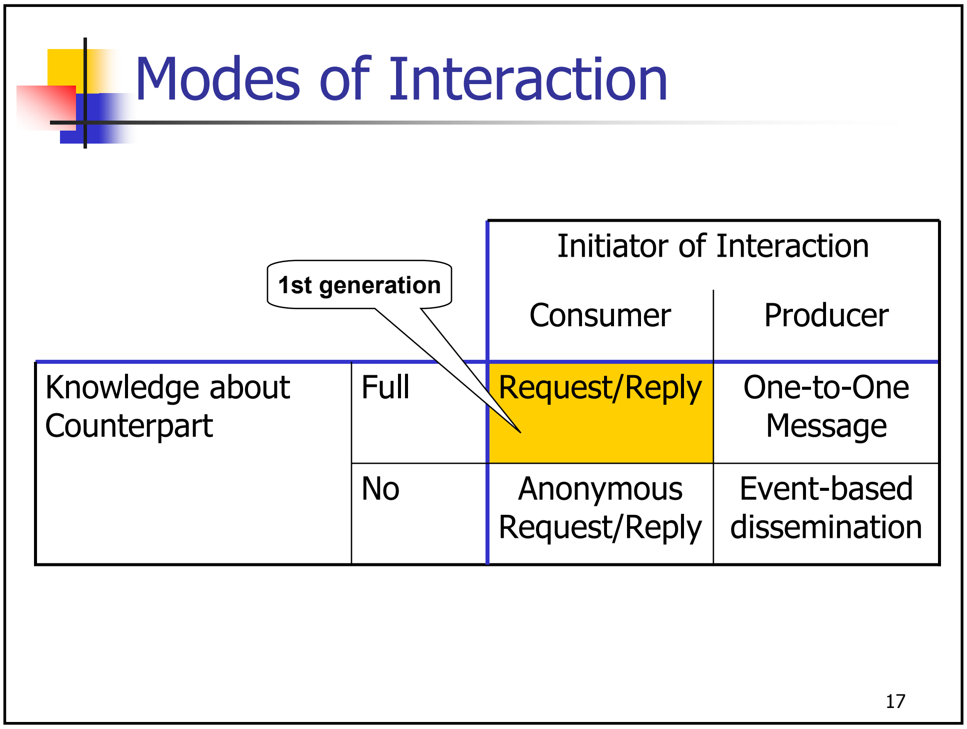
The Nature of Information

- Information *flows* from producer to consumer
  - info-pipes, broadcast disks, event streams, pub/sub
- Static view of information is a simplification
  - data flows into/out of high latency pool (database)

Mechanisms for access to static information (queries) are different from those for accessing flow of information (subscription/filters)

Working Hypothesis

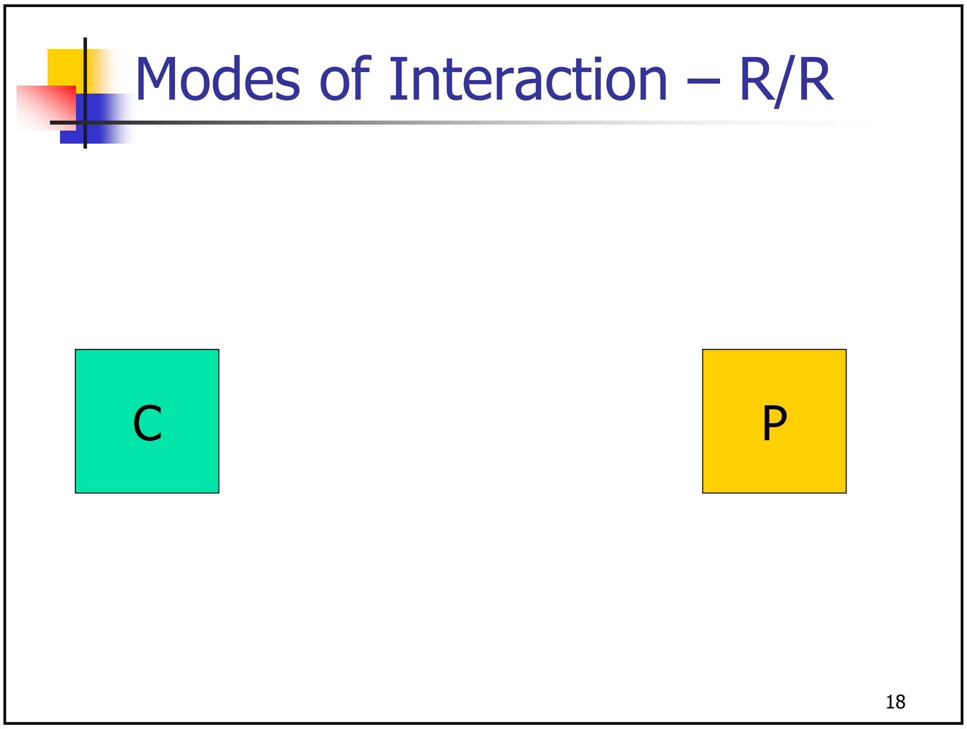
- Desintermediation/Reintermediation
- B2C, B2B, B2B2C, C2C, Portals, Markets (DotComs vs. NewCos), m-commerce …
- First generation e-commerce systems mapped existing applications 1:1 to new medium
- Next generation(s) will be based on flexible integration of services and components
- Flow of tasks and information
- How should (middleware) platforms look?



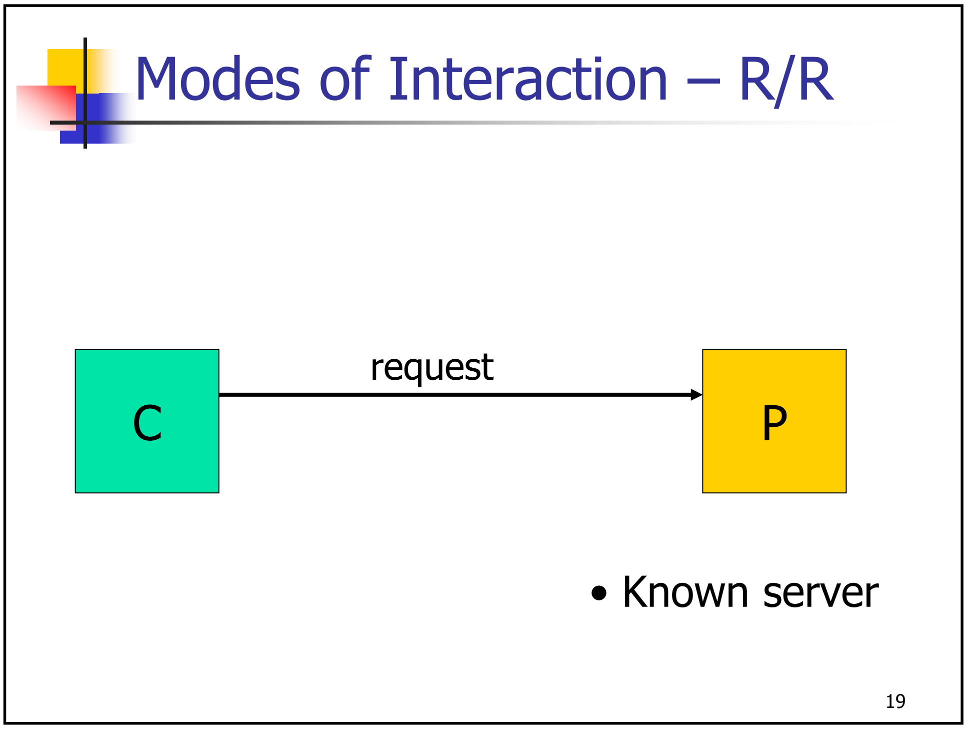
Modes of Interaction

1st generation

		Initiator of Interaction	
		Consumer	Producer
Knowledge about Counterpart	Full	Request/Reply	One-to-One Message
	No	Anonymous Request/Reply	Event-based dissemination



Modes of Interaction – R/R



Modes of Interaction – R/R

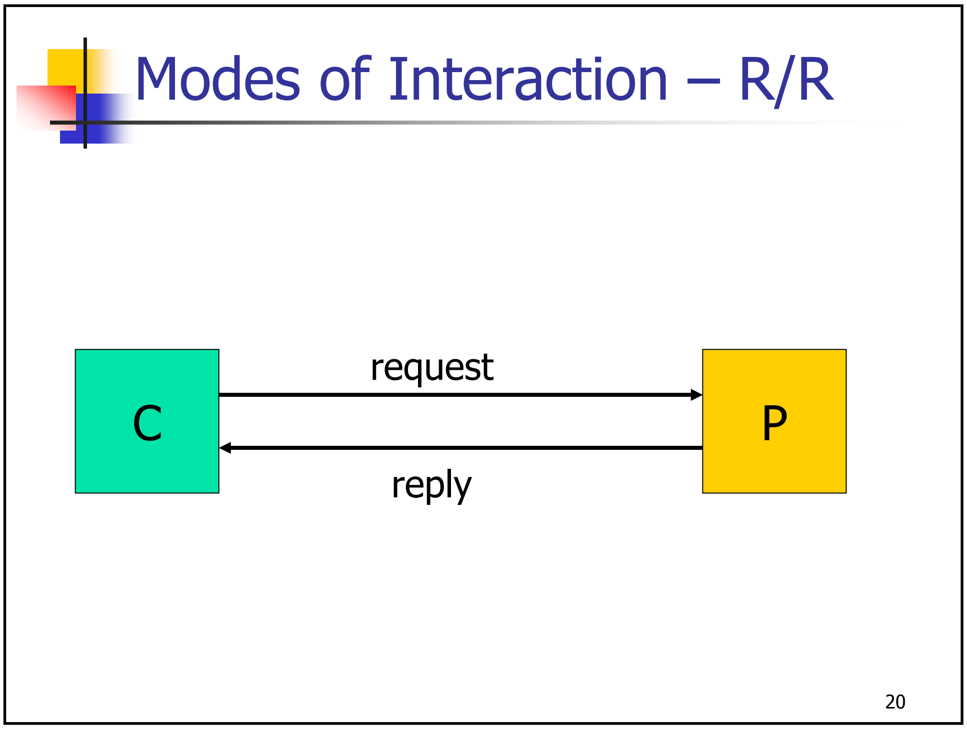
C

request

P

- Known server

19



Modes of Interaction – R/R

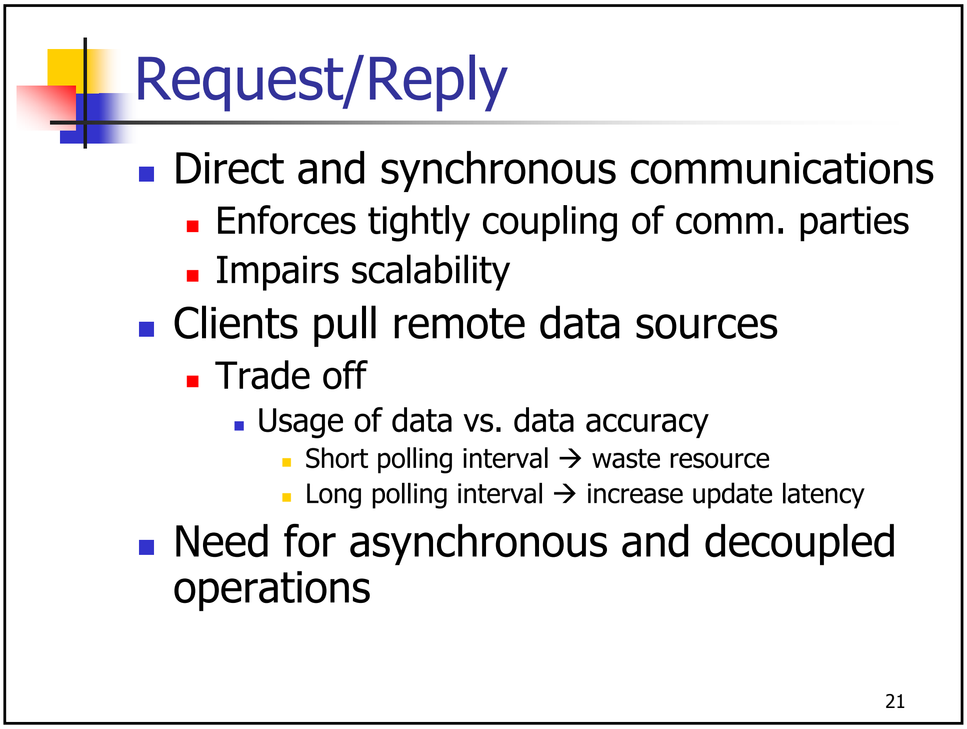
C

request

reply

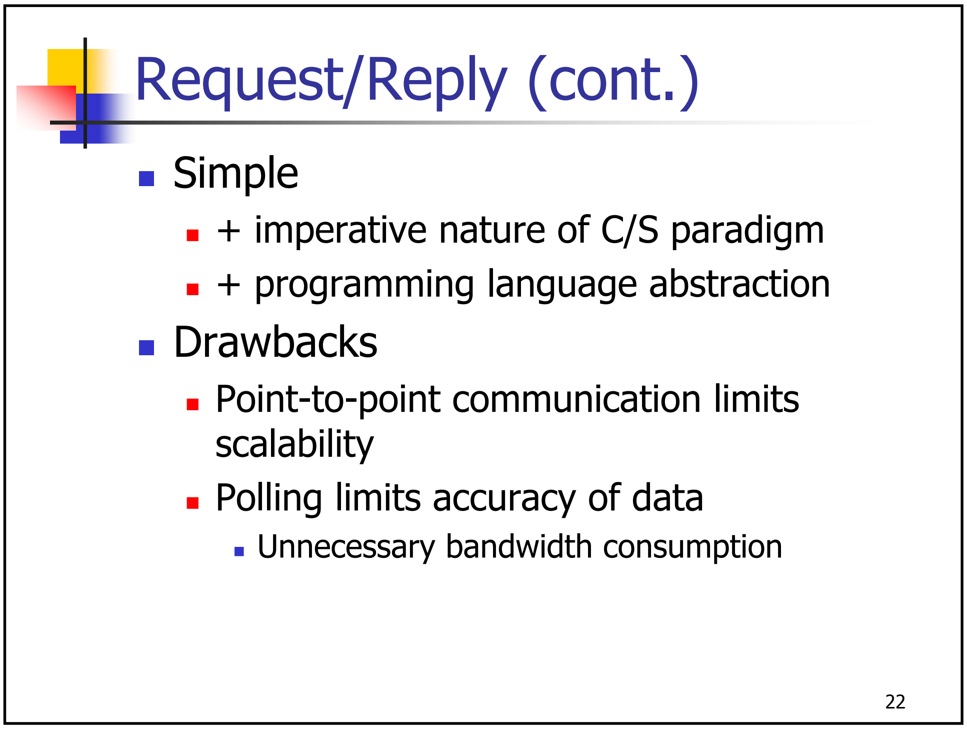
P

20



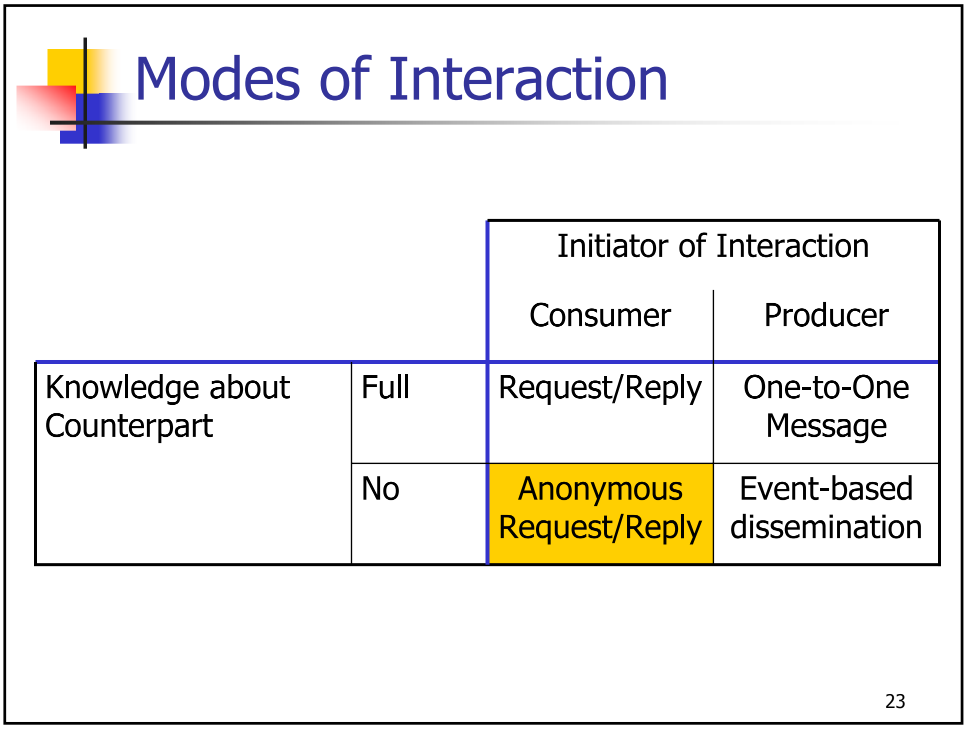
Request/Reply

- Direct and synchronous communications
  - Enforces tightly coupling of comm. parties
  - Impairs scalability
- Clients pull remote data sources
  - Trade off
    - Usage of data vs. data accuracy
      - Short polling interval → waste resource
      - Long polling interval → increase update latency
- Need for asynchronous and decoupled operations



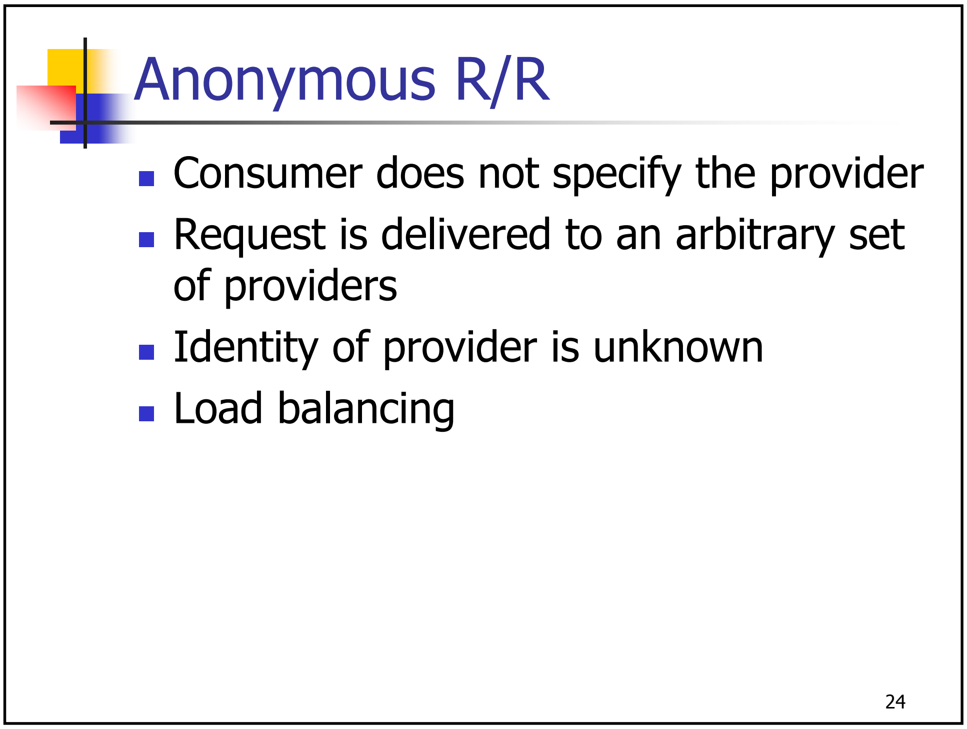
Request/Reply (cont.)

- Simple
  - + imperative nature of C/S paradigm
  - + programming language abstraction
- Drawbacks
  - Point-to-point communication limits scalability
  - Polling limits accuracy of data
    - Unnecessary bandwidth consumption



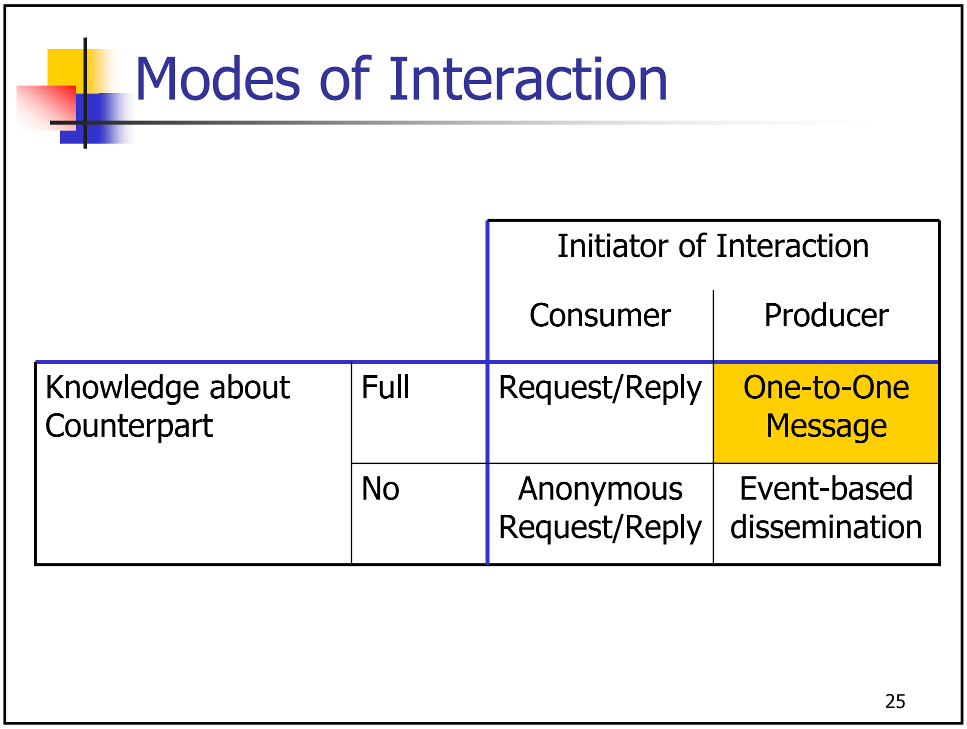
Modes of Interaction

		Initiator of Interaction	
		Consumer	Producer
Knowledge about Counterpart	Full	Request/Reply	One-to-One Message
	No	Anonymous Request/Reply	Event-based dissemination



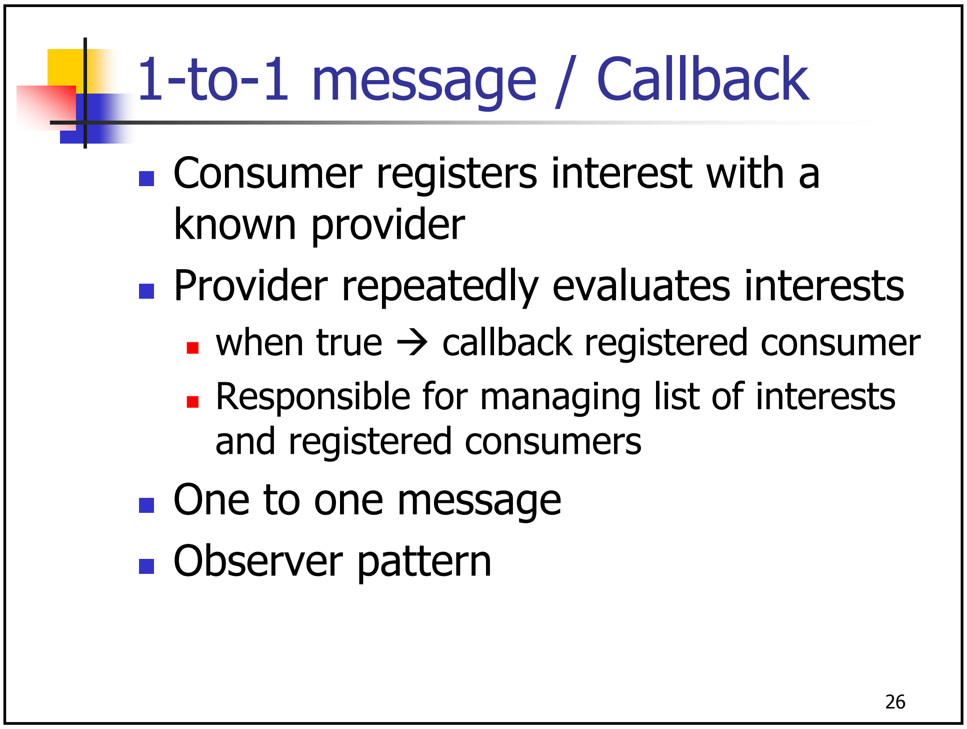
Anonymous R/R

- Consumer does not specify the provider
- Request is delivered to an arbitrary set of providers
- Identity of provider is unknown
- Load balancing



Modes of Interaction

		Initiator of Interaction	
		Consumer	Producer
Knowledge about Counterpart	Full	Request/Reply	One-to-One Message
	No	Anonymous Request/Reply	Event-based dissemination



1-to-1 message / Callback

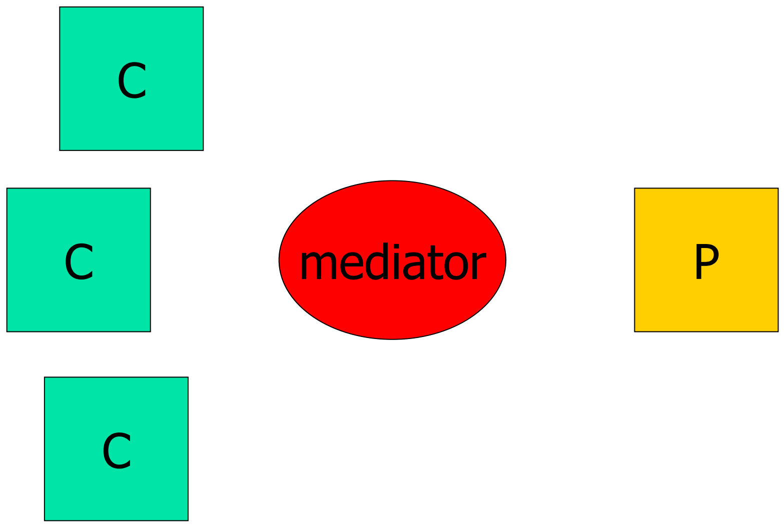
- Consumer registers interest with a known provider
- Provider repeatedly evaluates interests
  - when true → callback registered consumer
  - Responsible for managing list of interests and registered consumers
- One to one message
- Observer pattern

Modes of Interaction

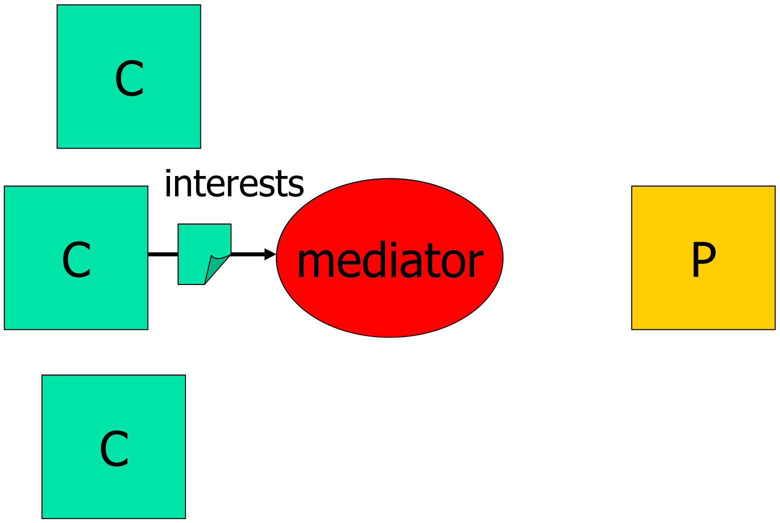
		Initiator of Interaction	
		Consumer	Producer
Knowledge about Counterpart	Full	Request/Reply	One-to-One Message
	No	Anonymous Request/Reply	Event-based dissemination

Needs a mediator

Modes of Interaction – Events



Modes of Interaction – Events



C

C

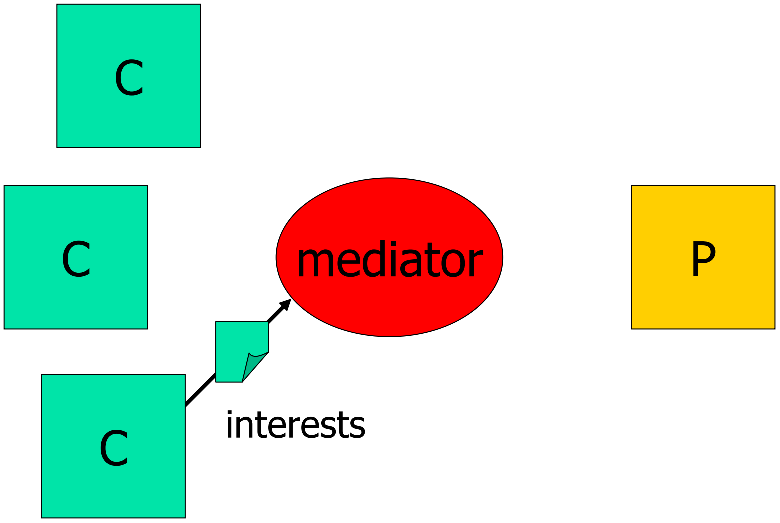
interests

mediator

P

C

Modes of Interaction – Events



C

C

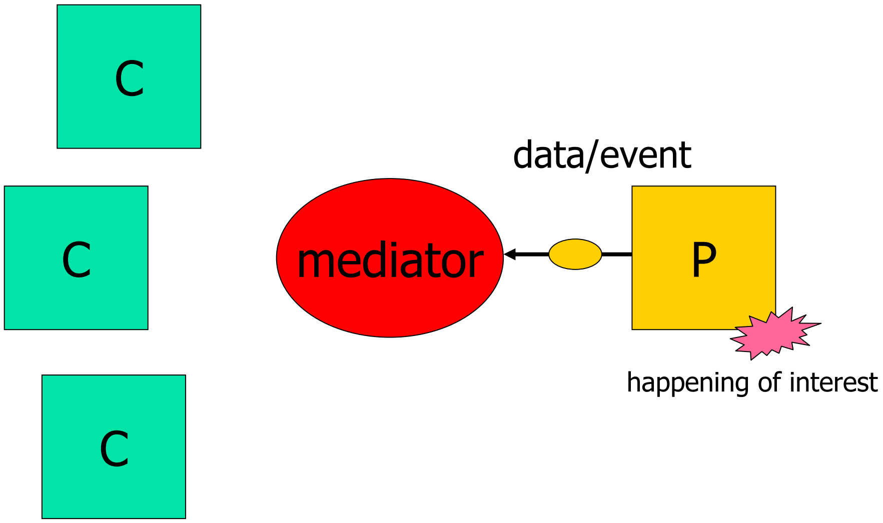
mediator

P

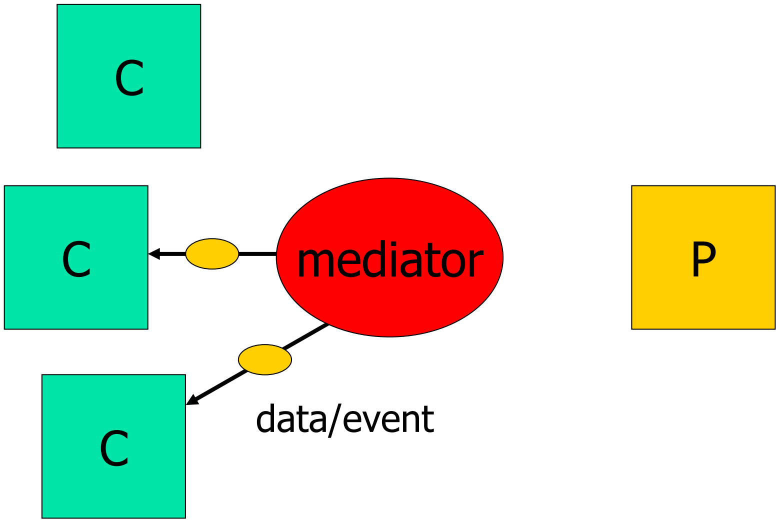
C

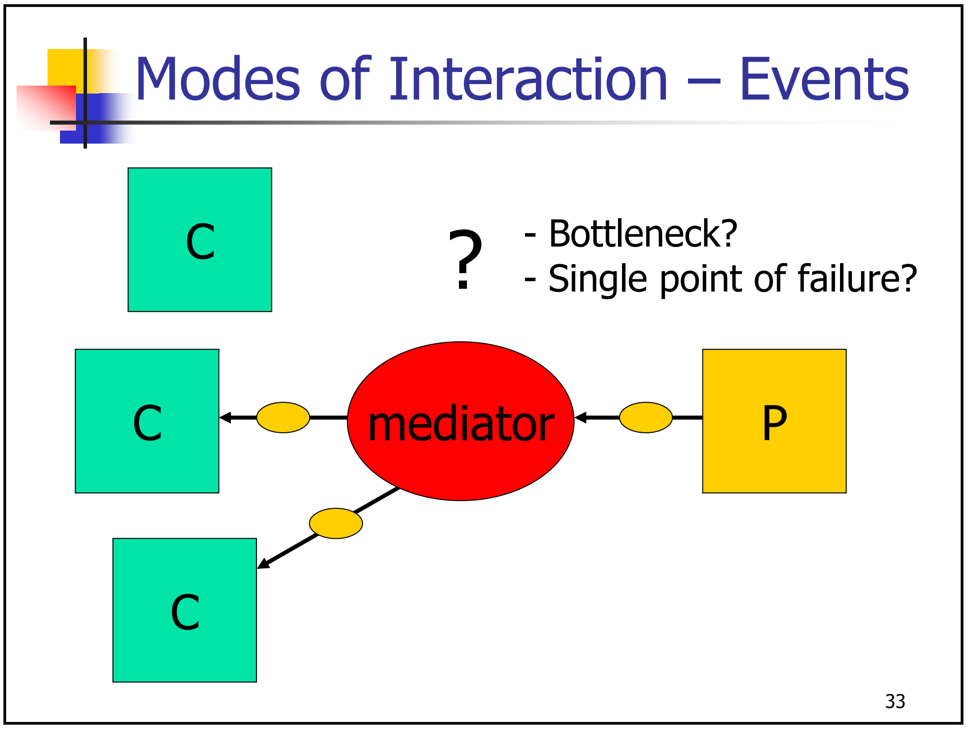
interests

Modes of Interaction – Events



Modes of Interaction – Events

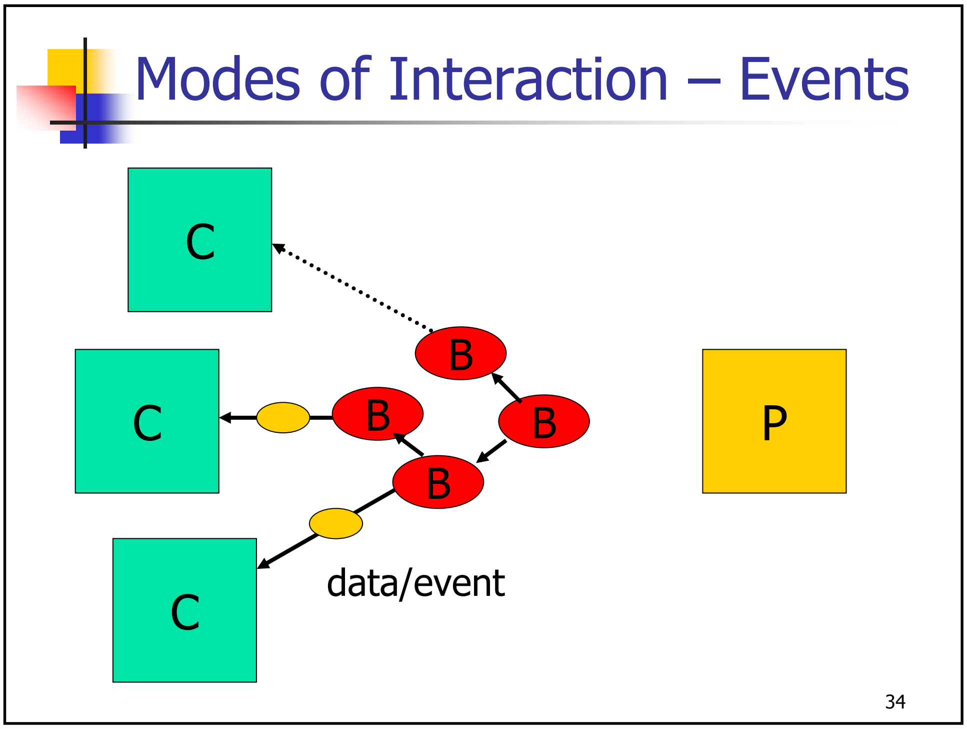




Modes of Interaction – Events

- Bottleneck?
- Single point of failure?

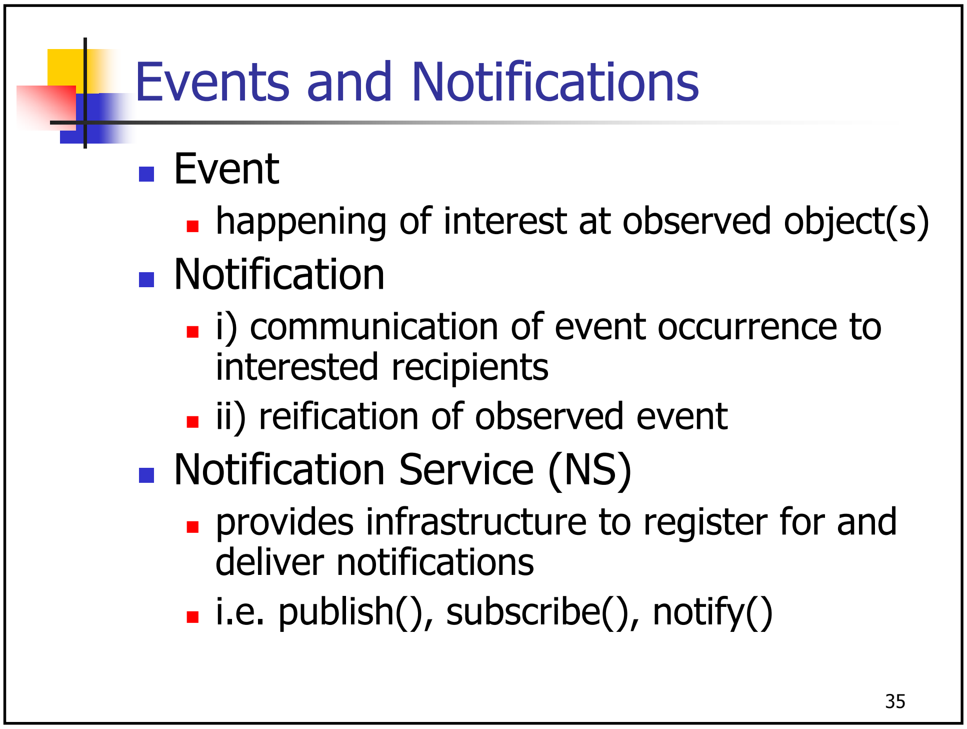
33



Modes of Interaction – Events

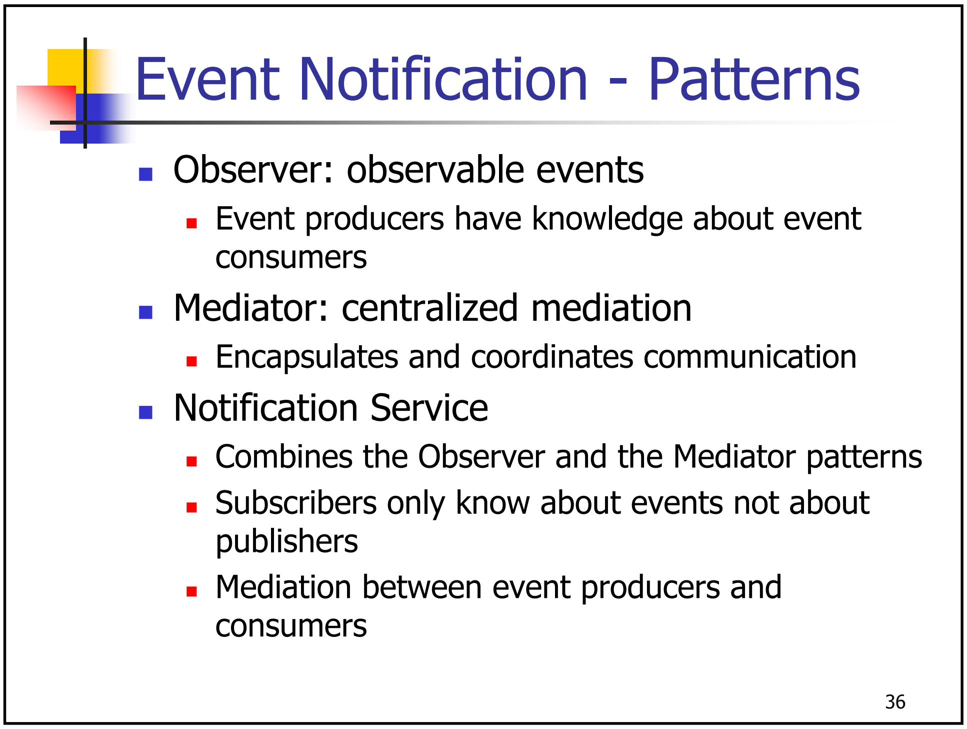
data/event

34



Events and Notifications

- Event
  - happening of interest at observed object(s)
- Notification
  - i) communication of event occurrence to interested recipients
  - ii) reification of observed event
- Notification Service (NS)
  - provides infrastructure to register for and deliver notifications
  - i.e. publish(), subscribe(), notify()



Event Notification - Patterns

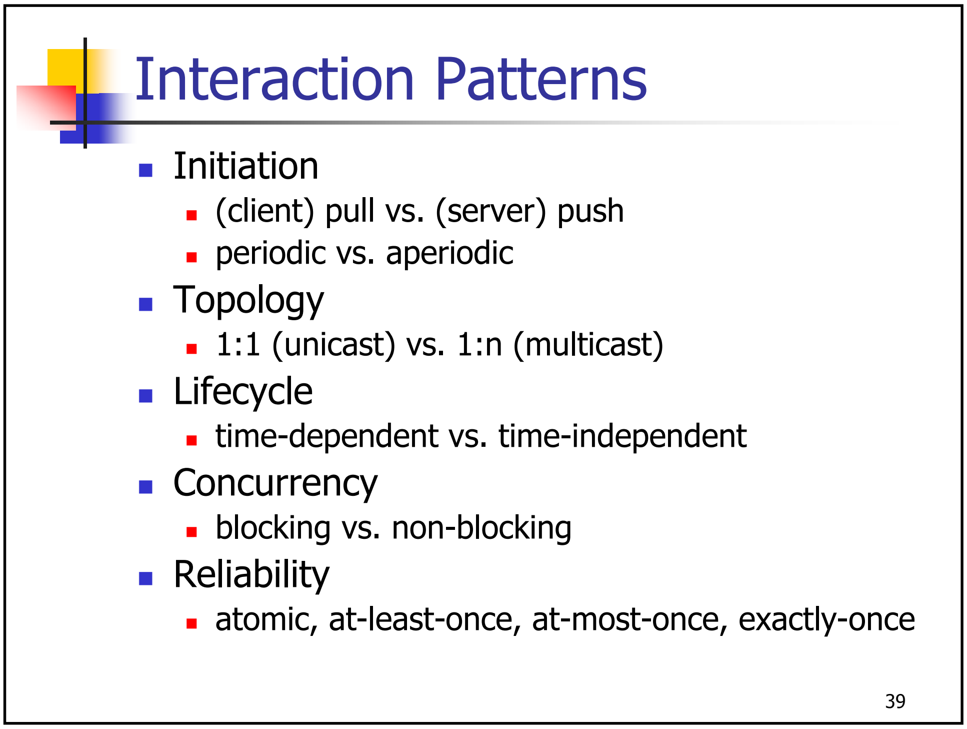
- Observer: observable events
  - Event producers have knowledge about event consumers
- Mediator: centralized mediation
  - Encapsulates and coordinates communication
- Notification Service
  - Combines the Observer and the Mediator patterns
  - Subscribers only know about events not about publishers
  - Mediation between event producers and consumers

Event-based Paradigm

- The (data) producer is the initiator of communication
- Notifications are not addressed to any specific consumer
  - Producers are not aware of consumers
- Consumers issue subscriptions (interests)
  - Consumers are not aware of producers
- Notifications are delivered to consumers if they match with subscriptions
- Flexible!

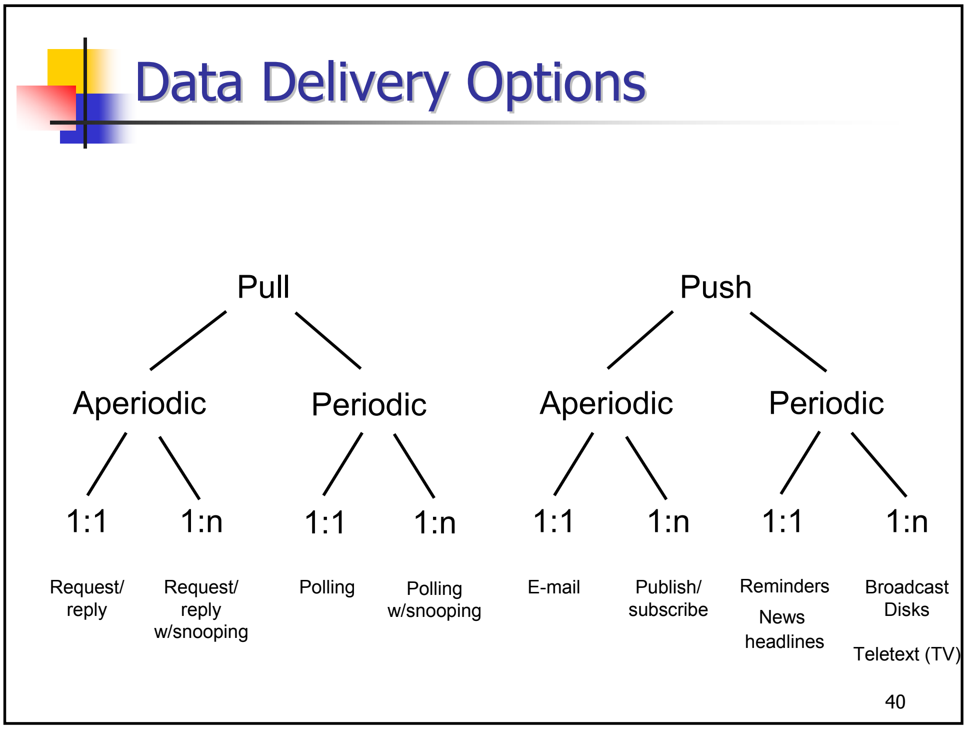
Modes of Interaction

- Influences
  - The architecture of the system
  - The design of the individual processes involved
- Link distributed parts of the system
  - difficult to change afterwards
- MoI determine system’s ability to adapt, evolve and scale
- MoI is confused with the implementation techniques



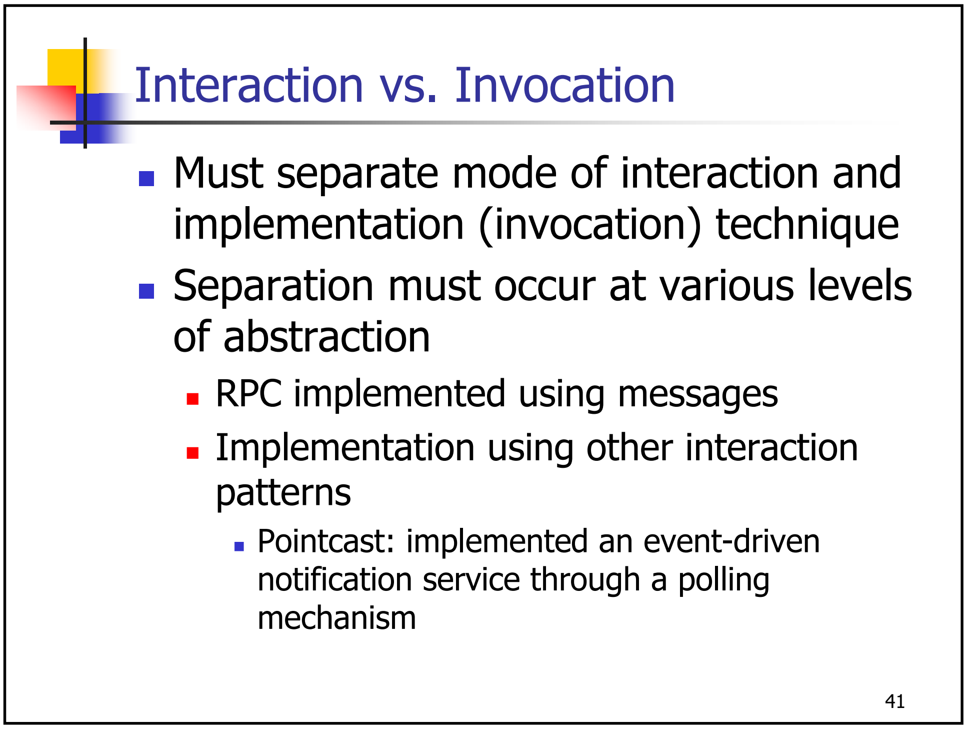
Interaction Patterns

- Initiation
  - (client) pull vs. (server) push
  - periodic vs. aperiodic
- Topology
  - 1:1 (unicast) vs. 1:n (multicast)
- Lifecycle
  - time-dependent vs. time-independent
- Concurrency
  - blocking vs. non-blocking
- Reliability
  - atomic, at-least-once, at-most-once, exactly-once



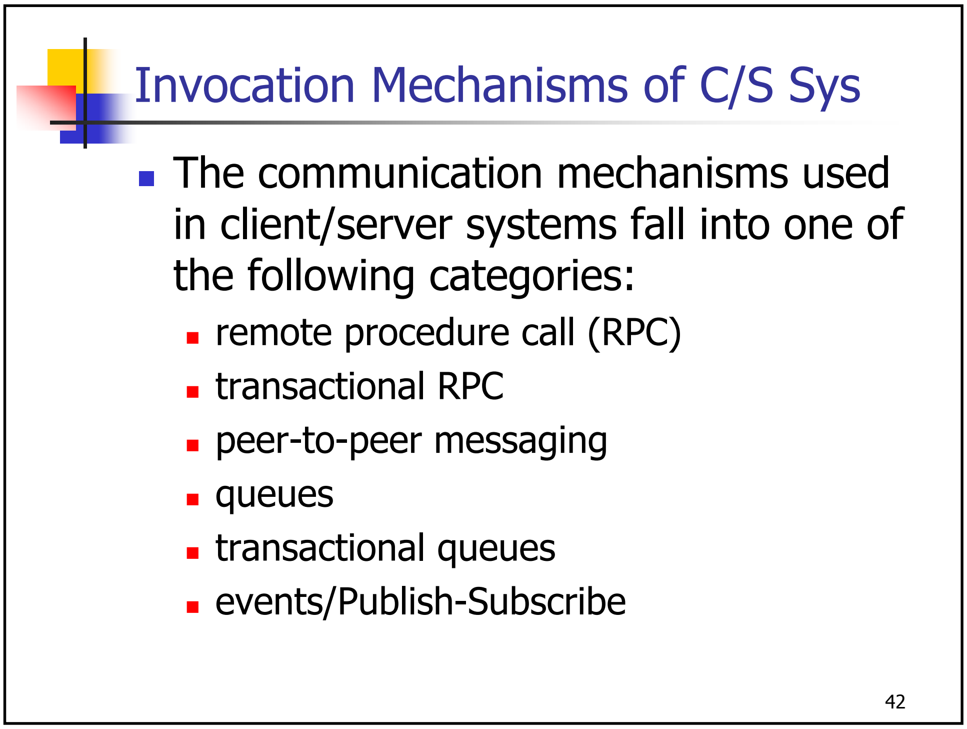
Data Delivery Options

- Pull
  - Aperiodic
    - 1:1
      Request/reply
    - 1:n
      Request/reply w/snooping
  - Periodic
    - 1:1
      Polling
    - 1:n
      Polling w/snooping
- Push
  - Aperiodic
    - 1:1
      E-mail
    - 1:n
      Publish/subscribe
  - Periodic
    - 1:1
      Reminders
      News headlines
    - 1:n
      Broadcast Disks
      Teletext (TV)



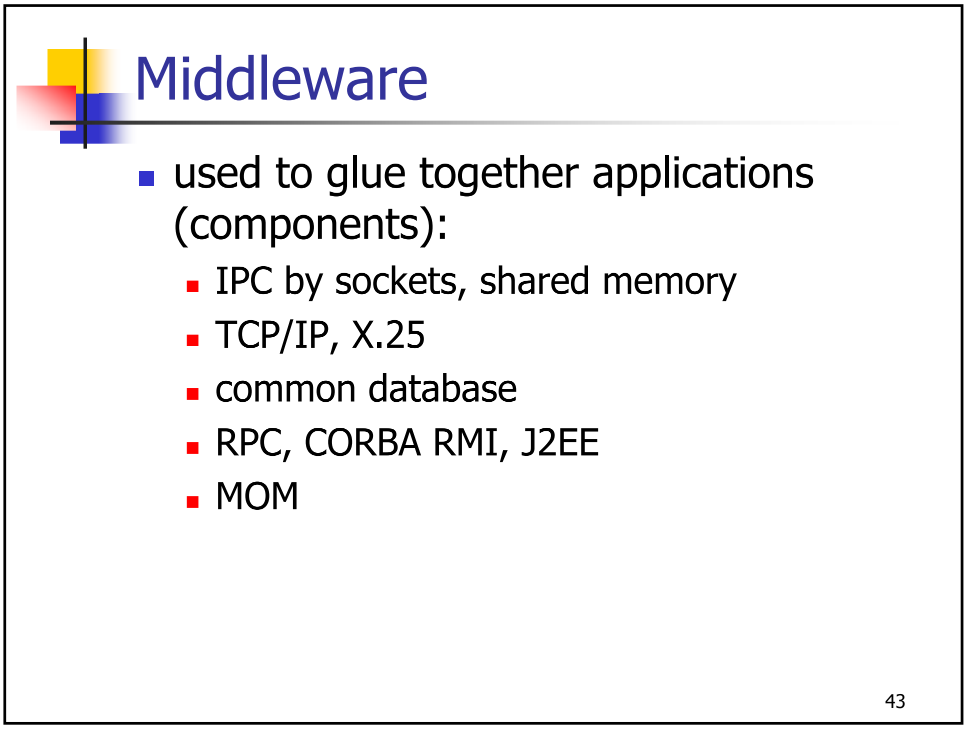
Interaction vs. Invocation

- Must separate mode of interaction and implementation (invocation) technique
- Separation must occur at various levels of abstraction
  - RPC implemented using messages
  - Implementation using other interaction patterns
    - Pointcast: implemented an event-driven notification service through a polling mechanism



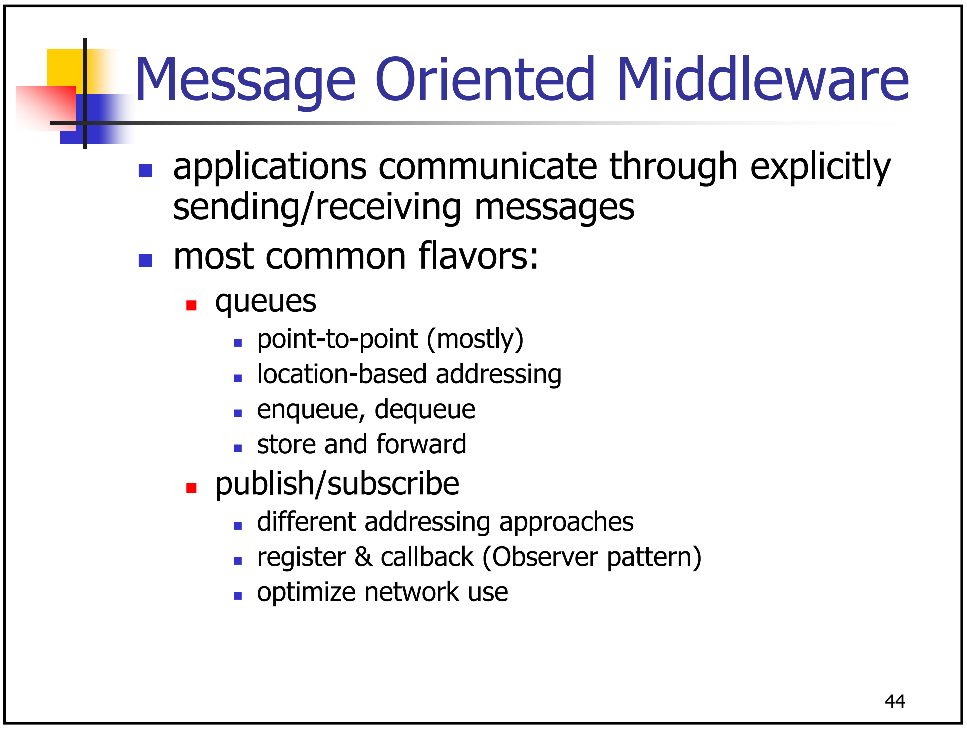
Invocation Mechanisms of C/S Sys

- The communication mechanisms used in client/server systems fall into one of the following categories:
  - remote procedure call (RPC)
  - transactional RPC
  - peer-to-peer messaging
  - queues
  - transactional queues
  - events/Publish-Subscribe



Middleware

- used to glue together applications (components):
  - IPC by sockets, shared memory
  - TCP/IP, X.25
  - common database
  - RPC, CORBA RMI, J2EE
  - MOM



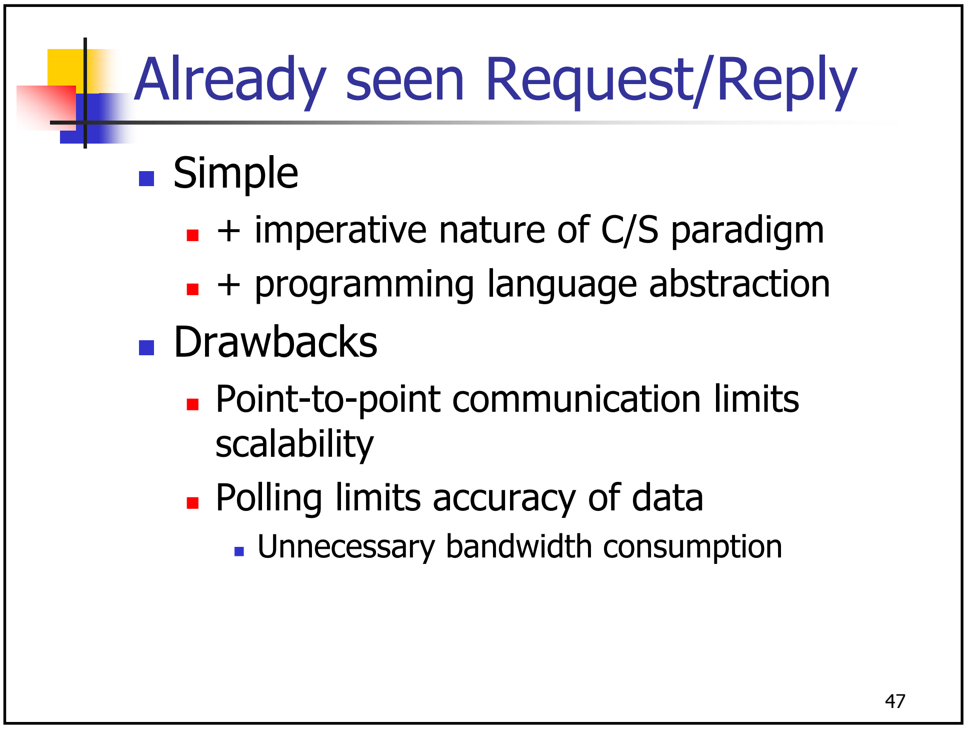
Message Oriented Middleware

- applications communicate through explicitly sending/receiving messages
- most common flavors:
  - queues
    - point-to-point (mostly)
    - location-based addressing
    - enqueue, dequeue
    - store and forward
  - publish/subscribe
    - different addressing approaches
    - register & callback (Observer pattern)
    - optimize network use

MOM (cont.)

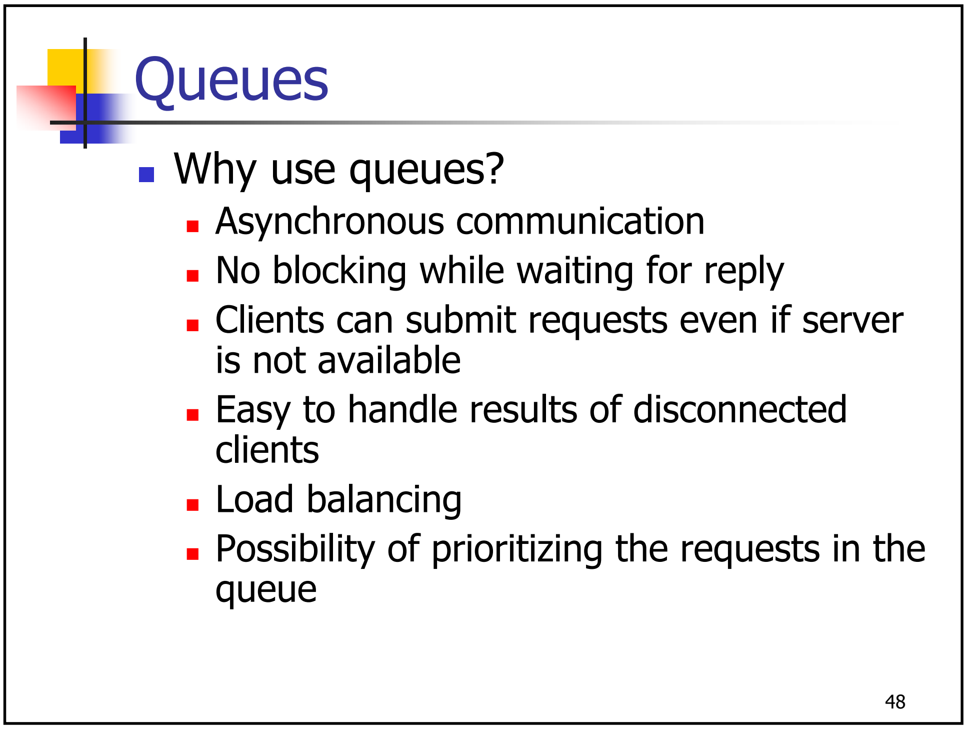
- flexible interaction
  - C/S request/reply, one-way push
  - asynchronous and time-independent
  - 1:1, n:1, 1:n, m:n
  - priorities
- flexible reliability
  - volatile/persistent/transactional queues
  - reliable/certified/transactional pub/sub
- additional services
  - load balancing, naming, security, content transformation

Communication Mechanisms



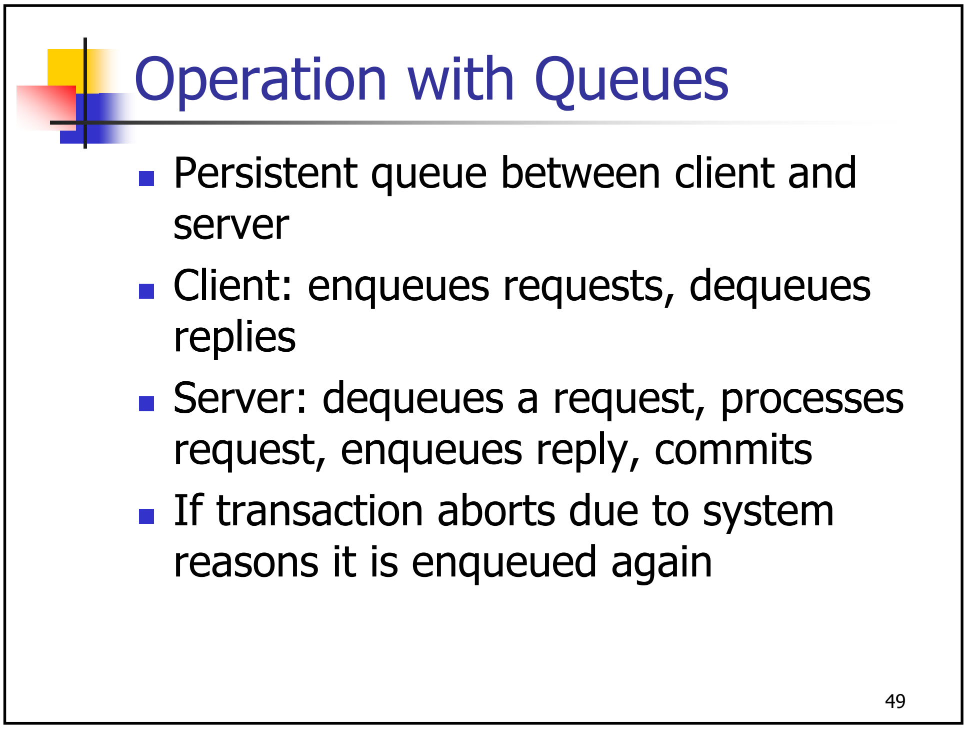
Already seen Request/Reply

- Simple
  - + imperative nature of C/S paradigm
  - + programming language abstraction
- Drawbacks
  - Point-to-point communication limits scalability
  - Polling limits accuracy of data
    - Unnecessary bandwidth consumption



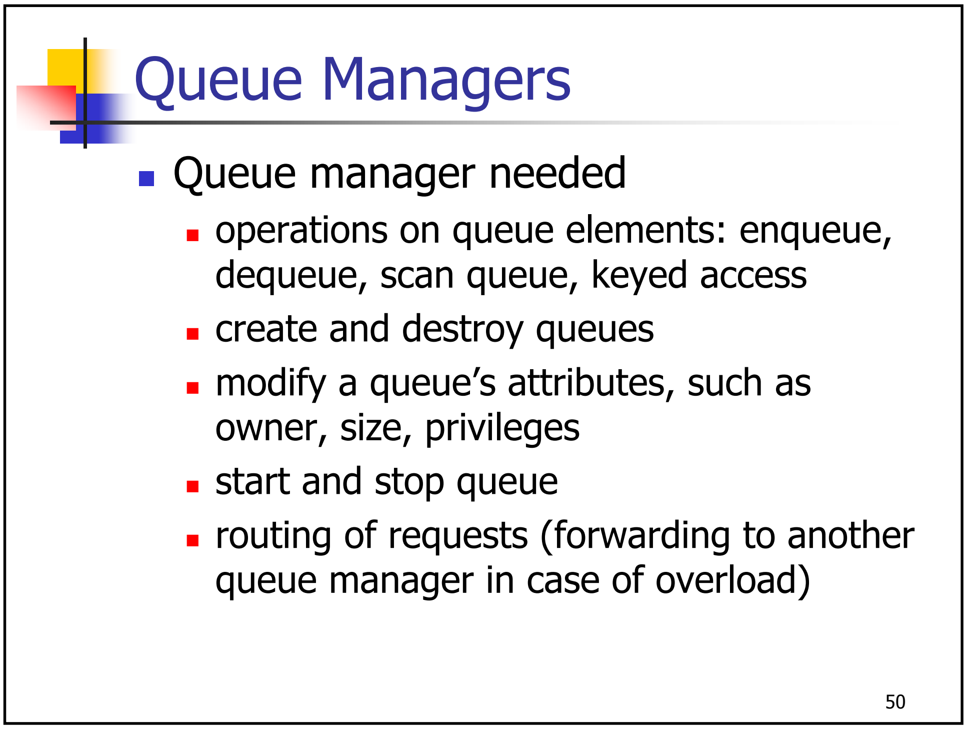
Queues

- Why use queues?
  - Asynchronous communication
  - No blocking while waiting for reply
  - Clients can submit requests even if server is not available
  - Easy to handle results of disconnected clients
  - Load balancing
  - Possibility of prioritizing the requests in the queue



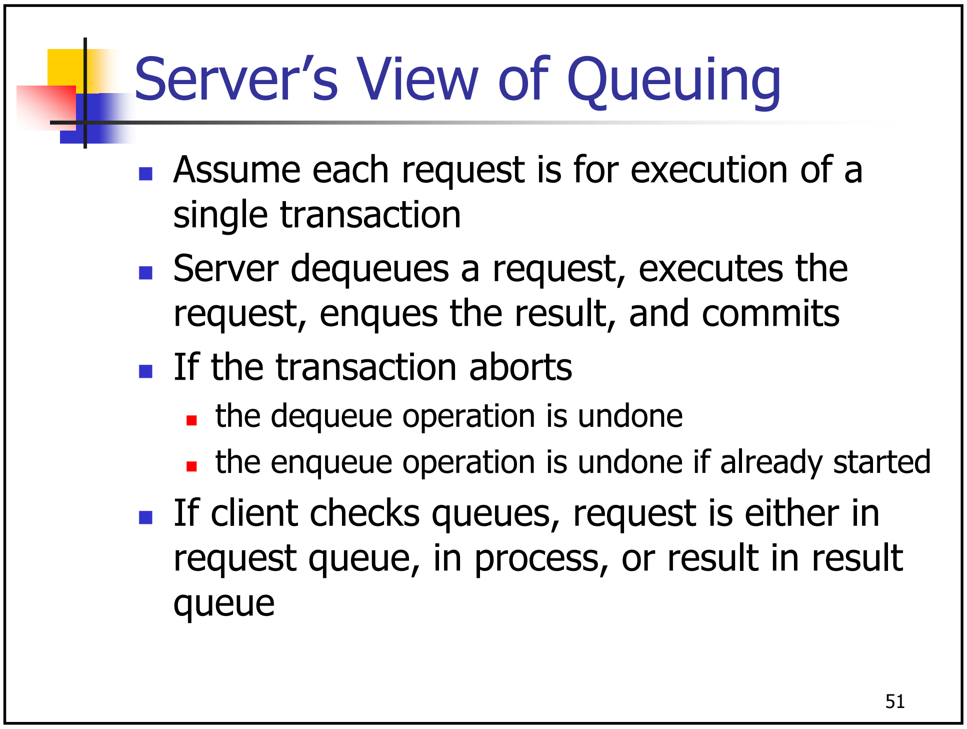
Operation with Queues

- Persistent queue between client and server
- Client: enqueues requests, dequeues replies
- Server: dequeues a request, processes request, enqueues reply, commits
- If transaction aborts due to system reasons it is enqueued again



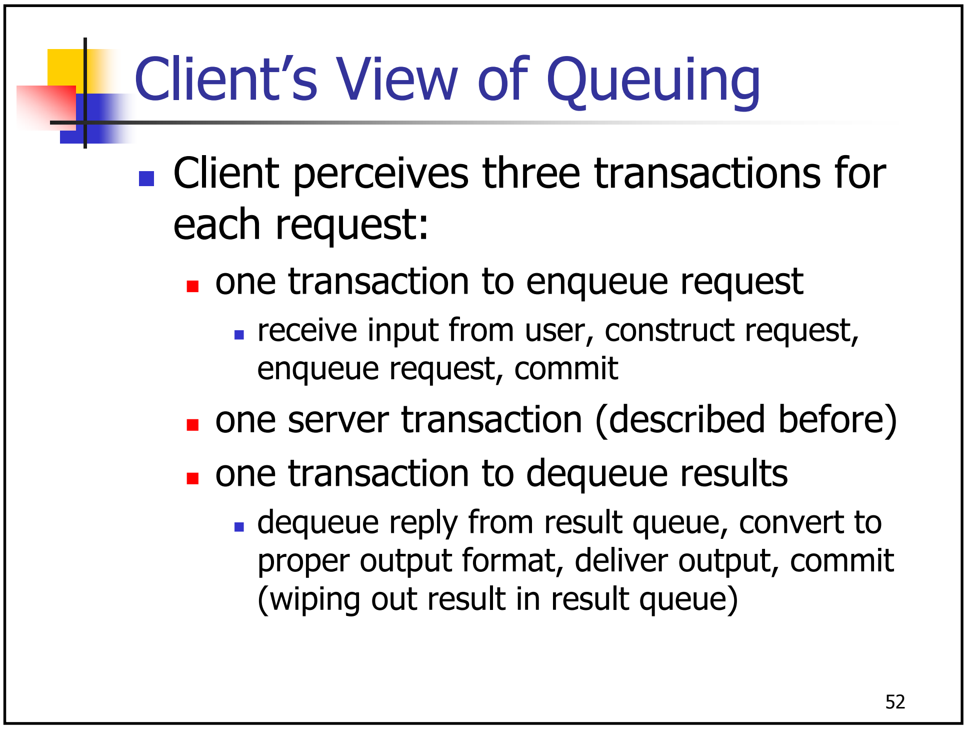
Queue Managers

- Queue manager needed
  - operations on queue elements: enqueue, dequeue, scan queue, keyed access
  - create and destroy queues
  - modify a queue’s attributes, such as owner, size, privileges
  - start and stop queue
  - routing of requests (forwarding to another queue manager in case of overload)



Server’s View of Queuing

- Assume each request is for execution of a single transaction
- Server dequeues a request, executes the request, enques the result, and commits
- If the transaction aborts
  - the dequeue operation is undone
  - the enqueue operation is undone if already started
- If client checks queues, request is either in request queue, in process, or result in result queue



Client’s View of Queuing

- Client perceives three transactions for each request:
  - one transaction to enqueue request
    - receive input from user, construct request, enqueue request, commit
  - one server transaction (described before)
  - one transaction to dequeue results
    - dequeue reply from result queue, convert to proper output format, deliver output, commit (wiping out result in result queue)

Request/Reply with Queues

Client A

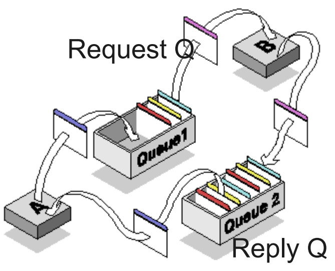
TX 1:
Start
  get input
  construct request
  enqueue request
Commit

TX 3:
Start
  dequeue reply
  decode reply
  process output
Commit

Message queues are good for asynchronous point to point (1:1) messaging

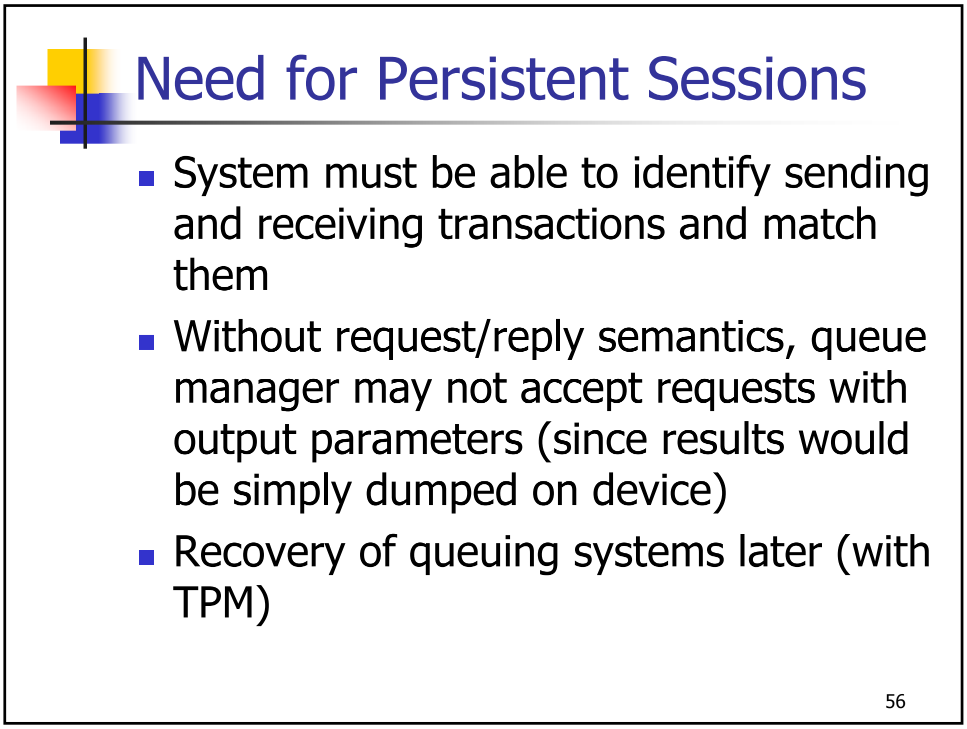
Server B

TX 2:
Start
  dequeue request
  process request
  enqueue reply
Commit



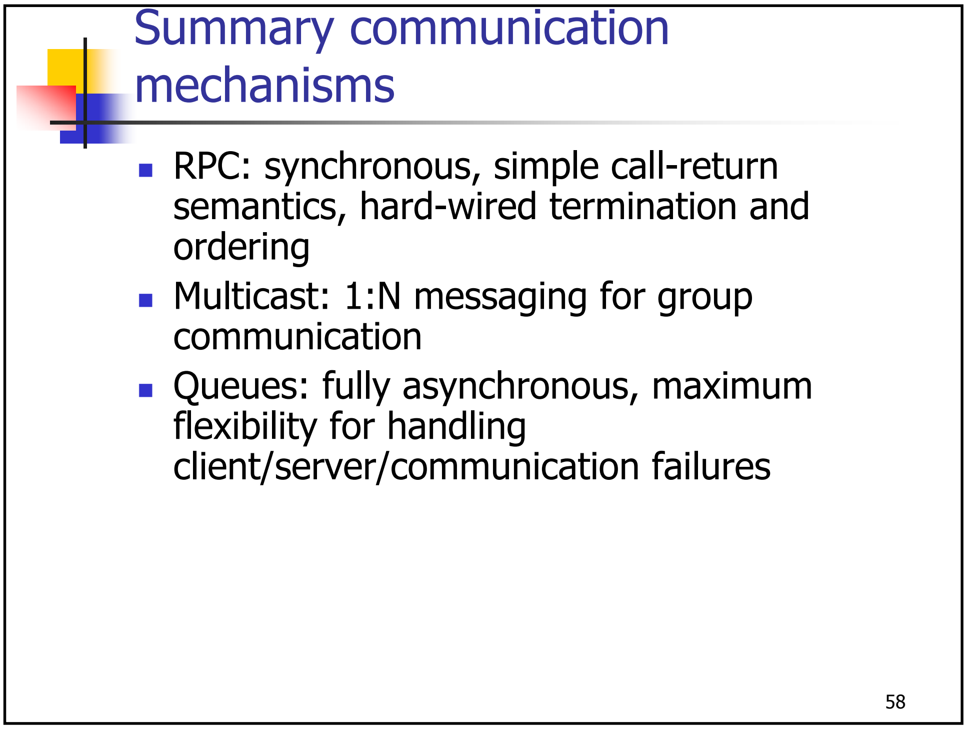
Cost/Benefit of Operating with Queues

- Using queues buys flexibility
  - communication with unavailable clients or servers
  - load balancing across servers
  - easy implementation of priorities
  - easier integration of legacy systems
- Using queues is expensive
  - 3 transactions instead of one
  - transactional queues must be managed by a (specialized) DBMS to guarantee persistence and transaction semantics



Need for Persistent Sessions

- System must be able to identify sending and receiving transactions and match them
- Without request/reply semantics, queue manager may not accept requests with output parameters (since results would be simply dumped on device)
- Recovery of queuing systems later (with TPM)



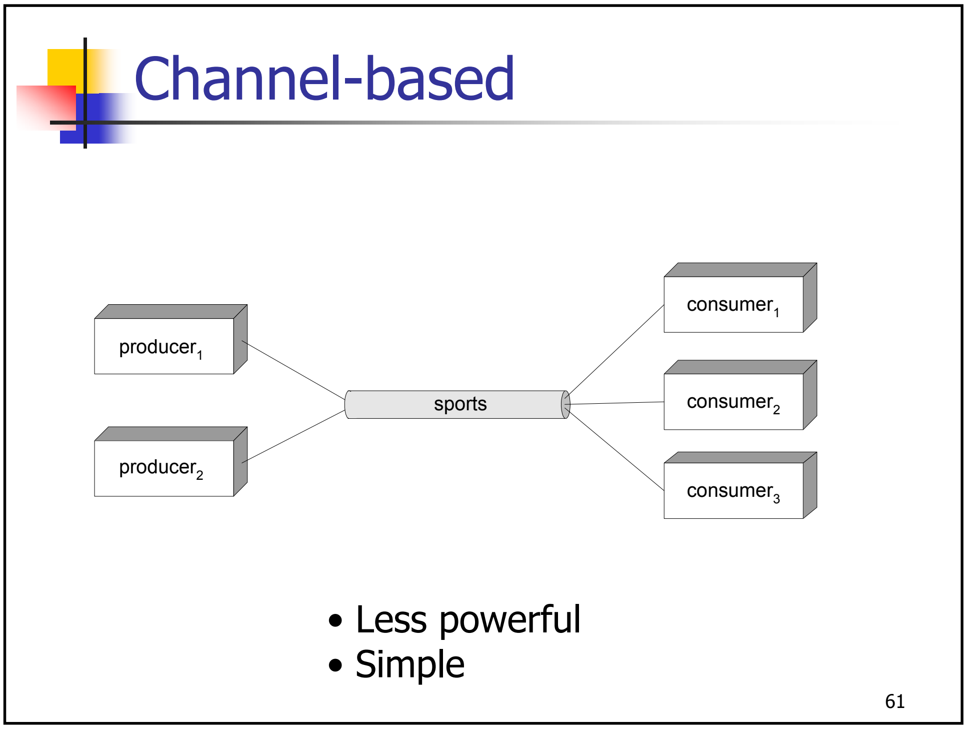
Summary communication mechanisms

- RPC: synchronous, simple call-return semantics, hard-wired termination and ordering
- Multicast: 1:N messaging for group communication
- Queues: fully asynchronous, maximum flexibility for handling client/server/communication failures

Publish/Subscribe

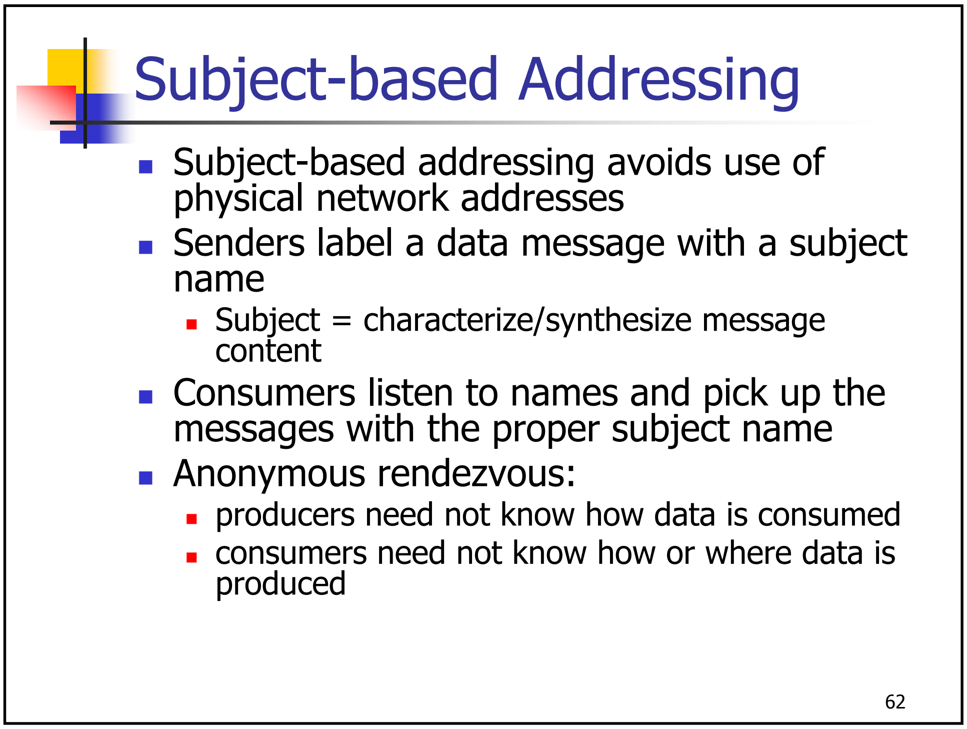
Pub/Sub Notification Service

- Main characteristics
  - decouples producers and consumers
  - anonymous to each other
  - dynamic number of consumers and producers
  - no directory service is needed
- Addressing models
  - Channel-based
  - Subject-based
  - Content-based
  - Concept-based



Channel-based

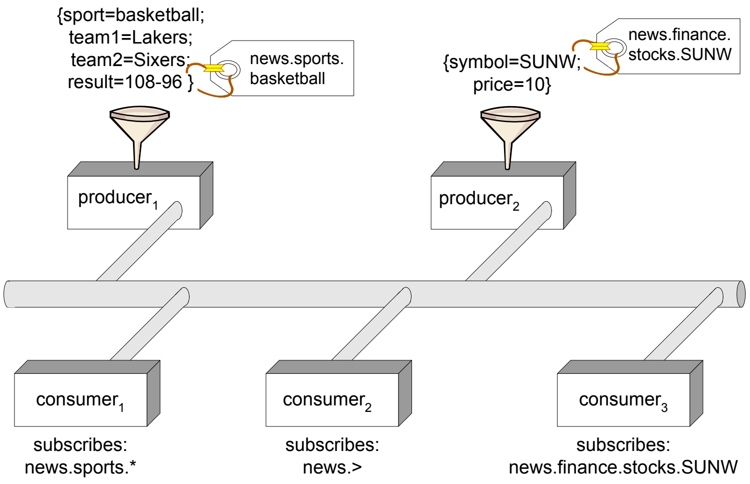
- Less powerful
- Simple



Subject-based Addressing

- Subject-based addressing avoids use of physical network addresses
- Senders label a data message with a subject name
  - Subject = characterize/synthesize message content
- Consumers listen to names and pick up the messages with the proper subject name
- Anonymous rendezvous:
  - producers need not know how data is consumed
  - consumers need not know how or where data is produced

Subject-based Addressing (2)



Subject-based Addressing (3)

- Agreement on subject names
- New subjects can be created dynamically
- Subject names consist of elements (subject name hierarchy)
  - element.subelement.subsubelement
- Wildcards can be used

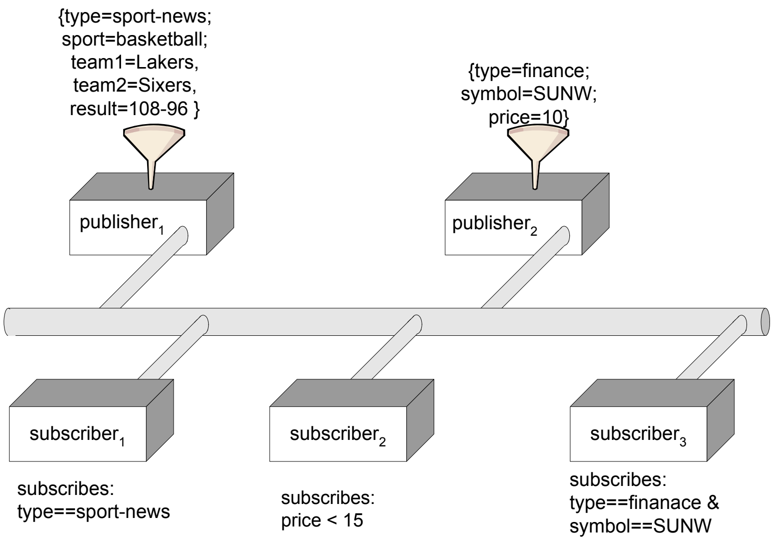
RUN.*	matches	RUN.AWAY
		RUN.home
RUN.>	matches	RUN.AWAY.far

- Difficult to change subject hierarchy

Content-Based Pub/Sub

- A content-based filter F
  - is a predicate on the content of notifications
  - induces the set of matching notifications
- Content-Based filtering is flexible but complex
  - cannot be easily mapped to "IP-Multicast"
- Centralized implementations not scalable to wide-area scenario
  - powerful distributed infrastructure required

Content-Based Pub/Sub



{type=sport-news;
sport=basketball;
team1=Lakers,
team2=Sixers,
result=108-96 }

{type=finance;
symbol=SUNW;
price=10}

subscribes:
type==sport-news

subscribes:
price < 15

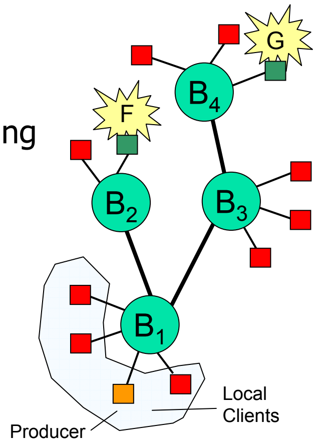
subscribes:
type==finanace &
symbol==SUNW

Problems derived from scale

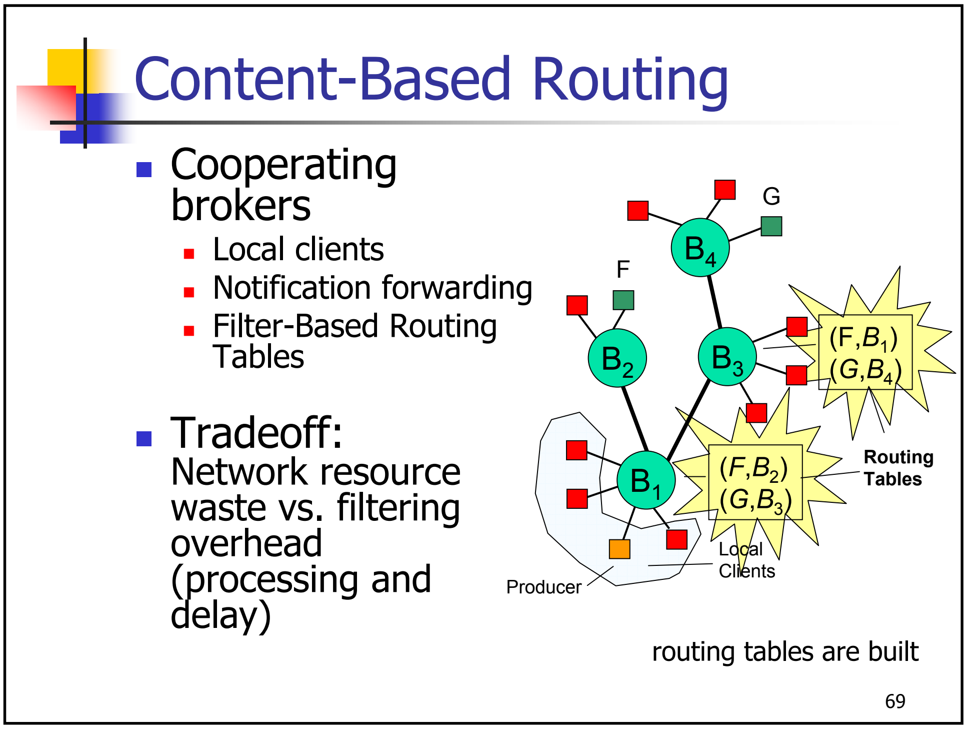
- Flooding of notifications is not an applicable solution
  - need strategies for filter placement to optimize bandwidth and size of routing tables

Content-Based Routing

- Cooperating brokers
  - Local clients
  - Notification forwarding
  - Filter-Based Routing Tables
- Tradeoff: Network resource waste vs. filtering overhead (processing and delay)



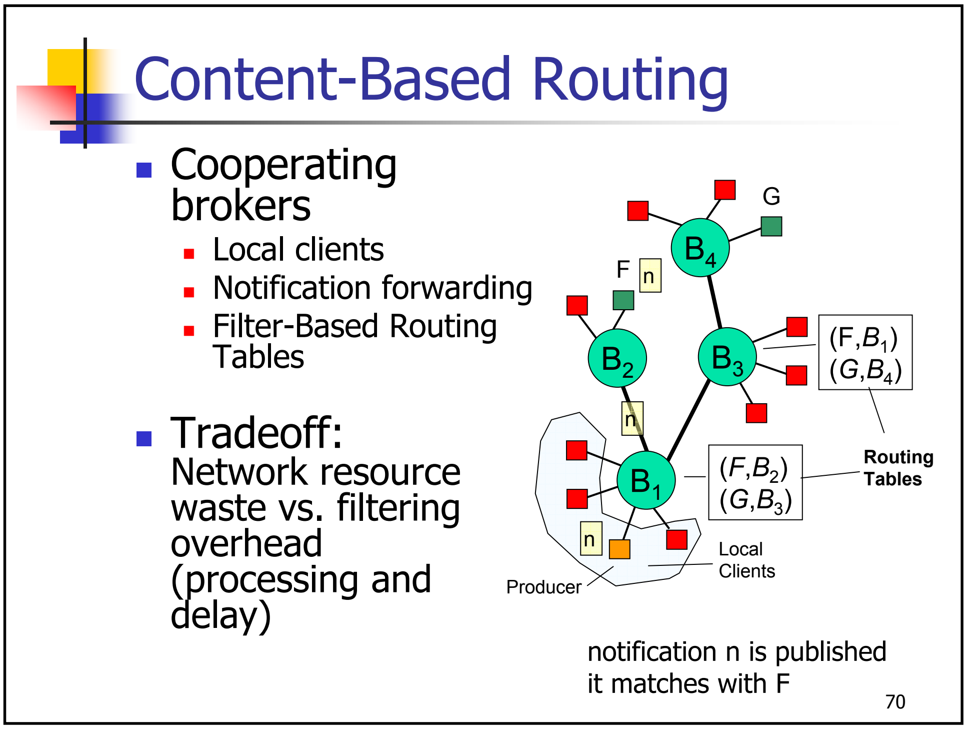
subscriptions: F and G



Content-Based Routing

- Cooperating brokers
  - Local clients
  - Notification forwarding
  - Filter-Based Routing Tables
- Tradeoff: Network resource waste vs. filtering overhead (processing and delay)

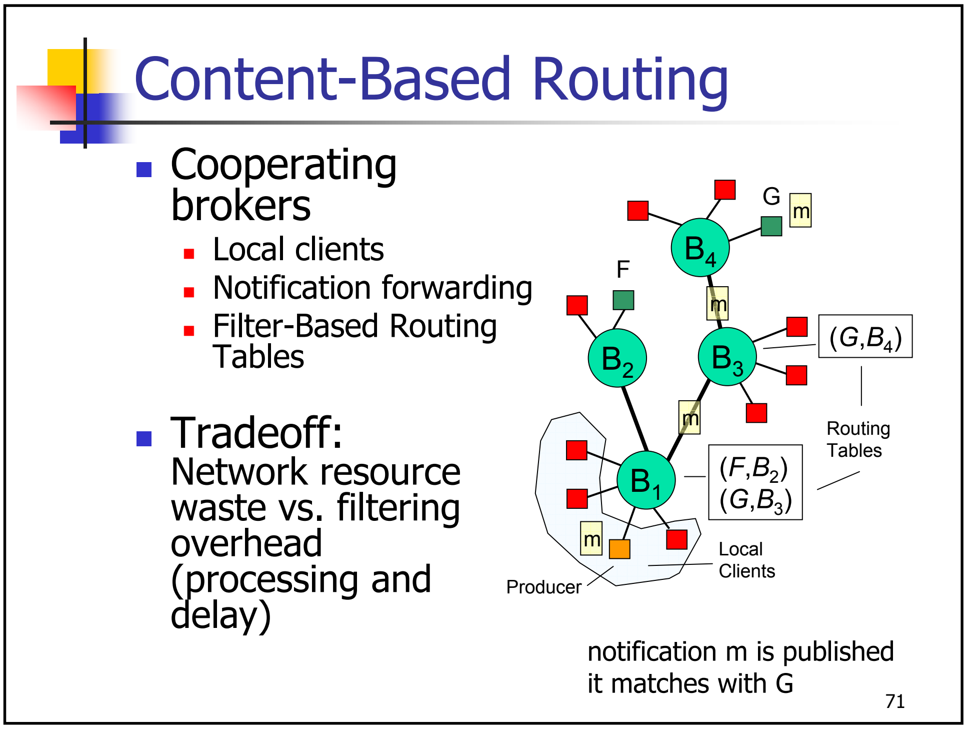
routing tables are built



Content-Based Routing

- Cooperating brokers
  - Local clients
  - Notification forwarding
  - Filter-Based Routing Tables
- Tradeoff: Network resource waste vs. filtering overhead (processing and delay)

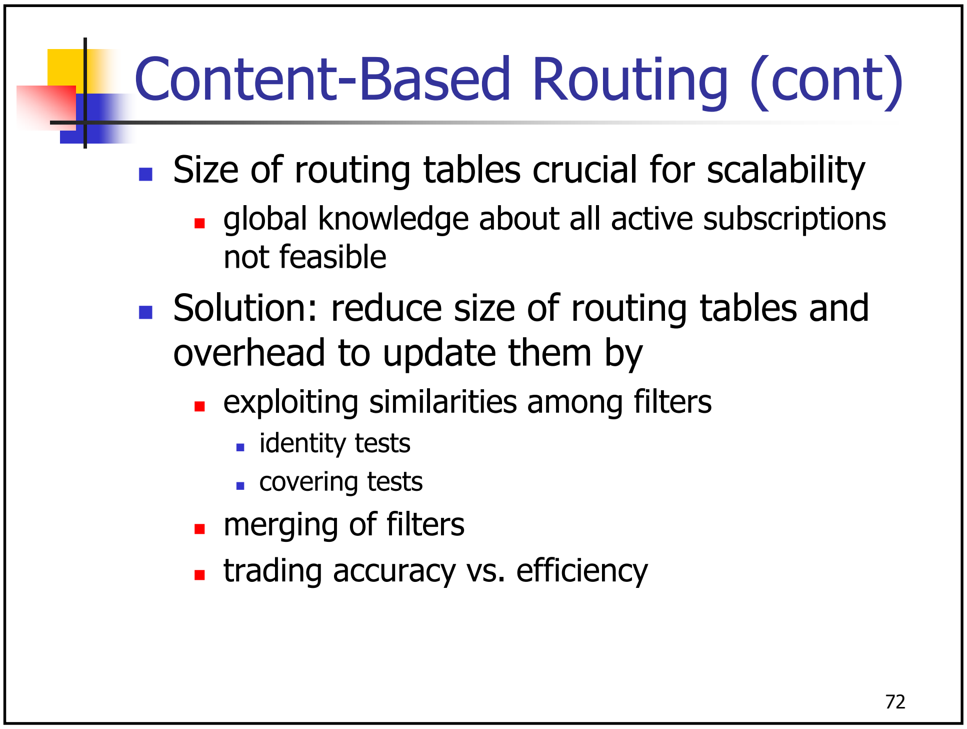
notification n is published
it matches with F



Content-Based Routing

- Cooperating brokers
  - Local clients
  - Notification forwarding
  - Filter-Based Routing Tables
- Tradeoff: Network resource waste vs. filtering overhead (processing and delay)

notification m is published
it matches with G

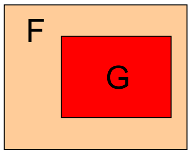


Content-Based Routing (cont)

- Size of routing tables crucial for scalability
  - global knowledge about all active subscriptions not feasible
- Solution: reduce size of routing tables and overhead to update them by
  - exploiting similarities among filters
    - identity tests
    - covering tests
  - merging of filters
  - trading accuracy vs. efficiency

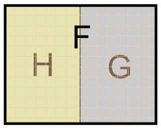
Covering

- Filters can cover each other
- Covering can decrease
  - size of routing tables
  - filter forwarding overhead

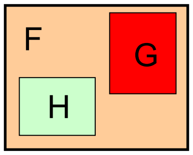


Merging of Filters

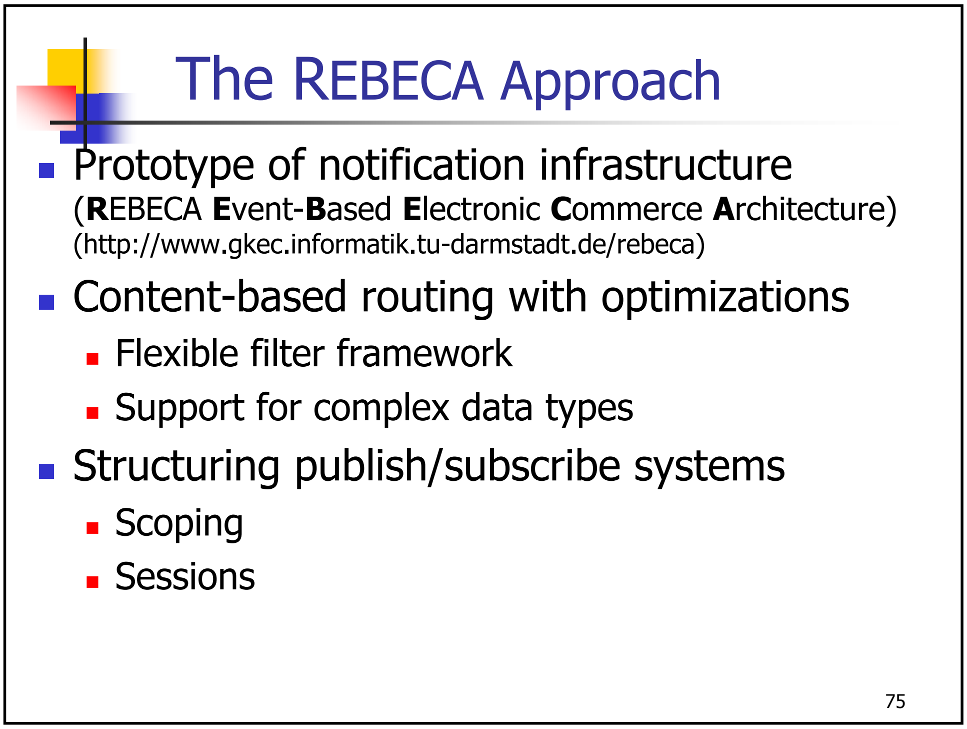
- Filters can be merged
  - *perfect*



  - *Imperfect*

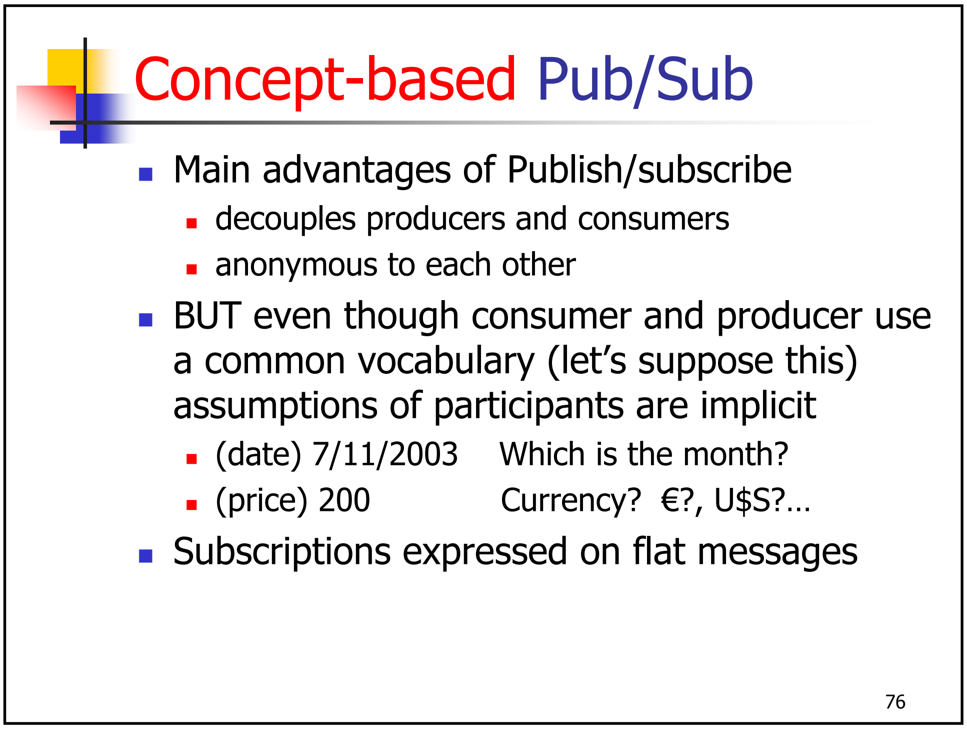


- Merging generates new covers
  - similar benefits as covering



The REBECA Approach

- Prototype of notification infrastructure
  (**R**EBECA **E**vent-**B**ased **E**lectronic **C**ommerce **A**rchitecture)
  (http://www.gkec.informatik.tu-darmstadt.de/rebeca)
- Content-based routing with optimizations
  - Flexible filter framework
  - Support for complex data types
- Structuring publish/subscribe systems
  - Scoping
  - Sessions



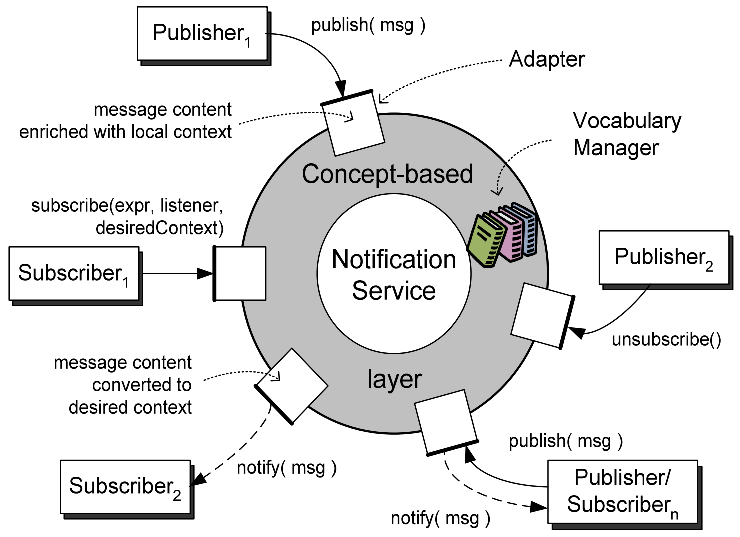
Concept-based Pub/Sub

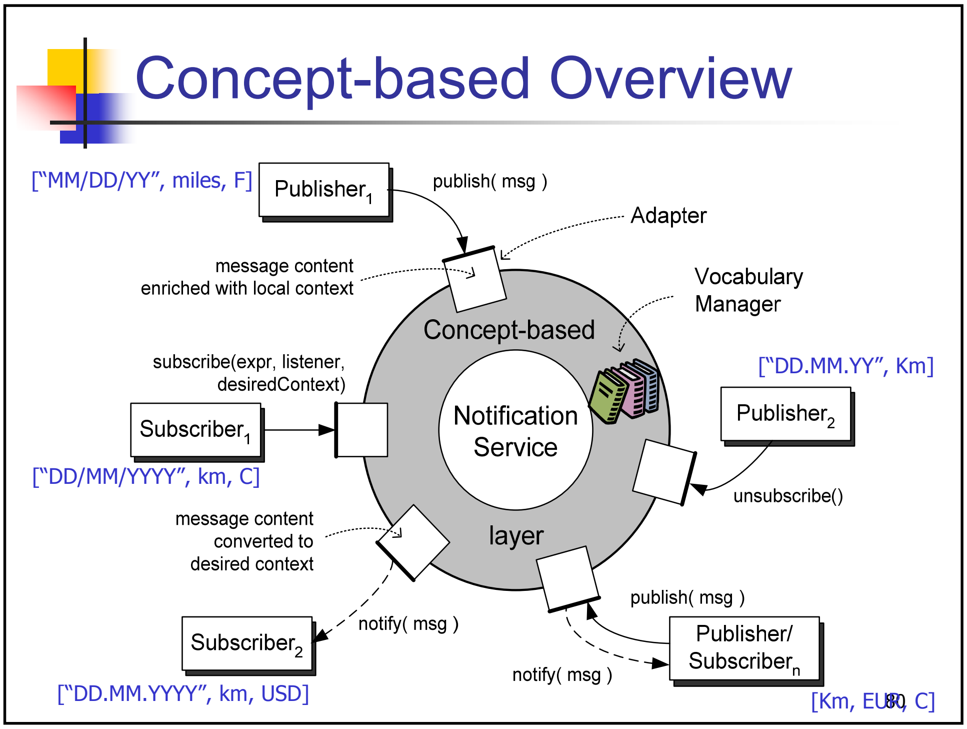
- Main advantages of Publish/subscribe
  - decouples producers and consumers
  - anonymous to each other
- BUT even though consumer and producer use a common vocabulary (let's suppose this) assumptions of participants are implicit
  - (date) 7/11/2003 Which is the month?
  - (price) 200 Currency? €?, U$S?…
- Subscriptions expressed on flat messages

Concept-based Pub/Sub

- Provide a higher level of abstraction to describe the interests of publishers and subscribers
- Events represented using MIX
- Subscribers can specify their assumptions
  - Price < 100 [€]
  - DeliveryDate <= 7/11/2003 [dd/mm/yyyy]
- The notification service delivers ready-to-process events to subscribers
  - No further processing is needed

Concept-based Overview





Concept-based Overview

["MM/DD/YY", miles, F]

Publisher$_1$

publish(msg)

Adapter

message content enriched with local context

Vocabulary Manager

Concept-based

Notification Service

layer

subscribe(expr, listener, desiredContext)

["DD.MM.YY", Km]

Subscriber$_1$

Publisher$_2$

["DD/MM/YYYY", km, C]

unsubscribe()

message content converted to desired context

publish(msg)

notify(msg)

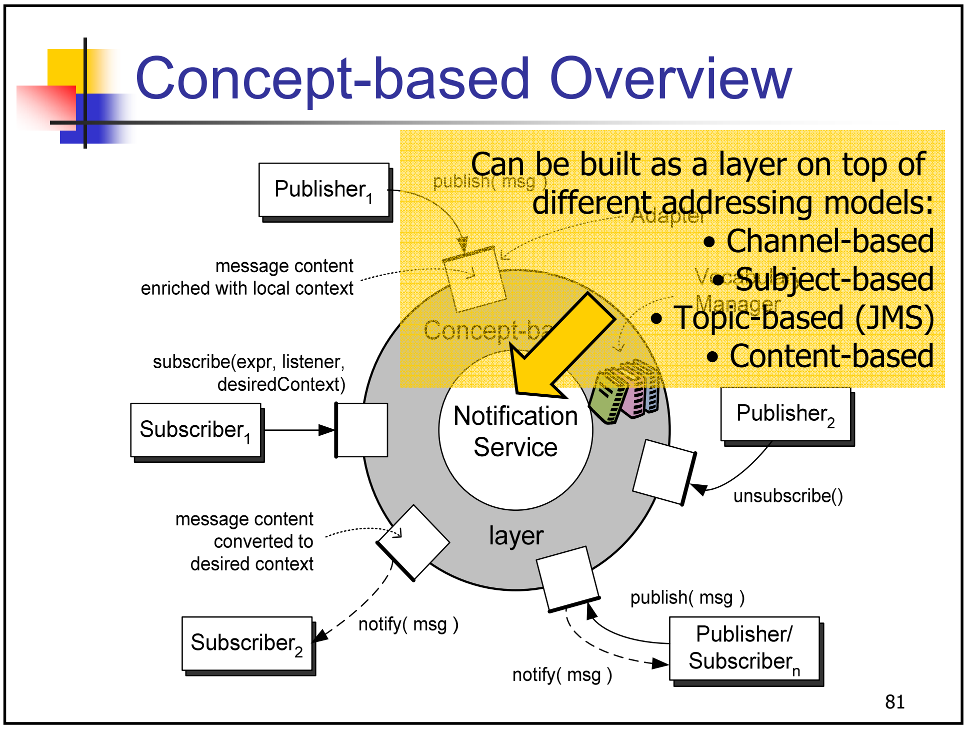
Subscriber$_2$

Publisher/ Subscriber$_n$

notify(msg)

["DD.MM.YYYY", km, USD]

[Km, EU80, C]



Concept-based Overview

Can be built as a layer on top of different addressing models:

- Channel-based
- Subject-based
- Topic-based (JMS)
- Content-based

Publisher$_1$

publish(msg)

Adapter

message content enriched with local context

Vocabulary Manager

Concept-based

Notification Service

layer

subscribe(expr, listener, desiredContext)

Subscriber$_1$

Publisher$_2$

unsubscribe()

message content converted to desired context

publish(msg)

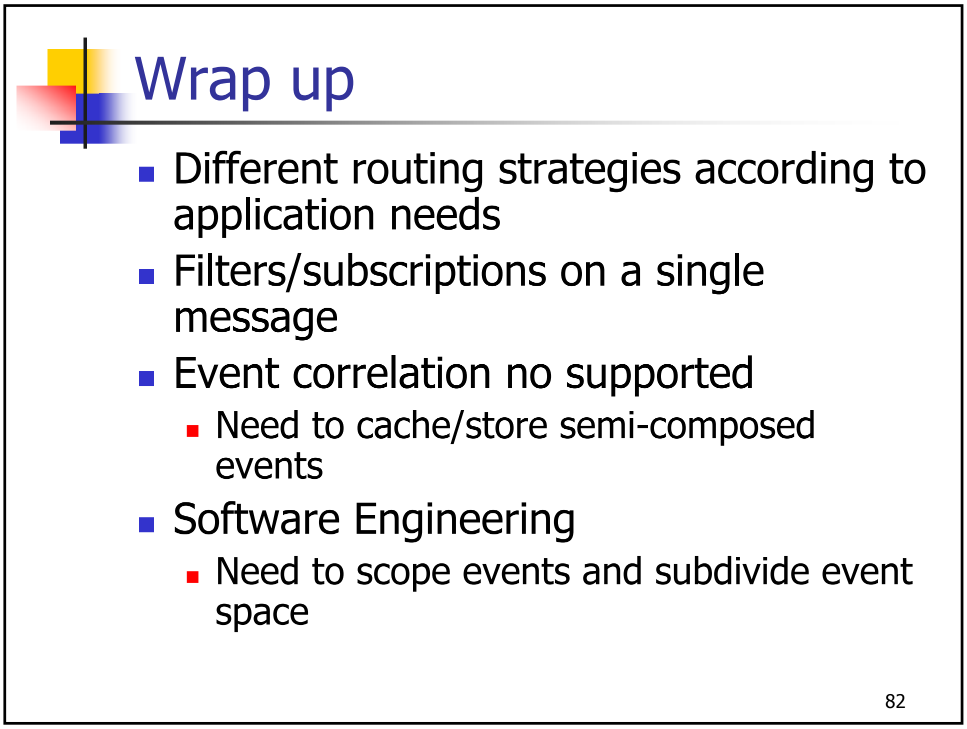
notify(msg)

Subscriber$_2$

Publisher/ Subscriber$_n$

notify(msg)

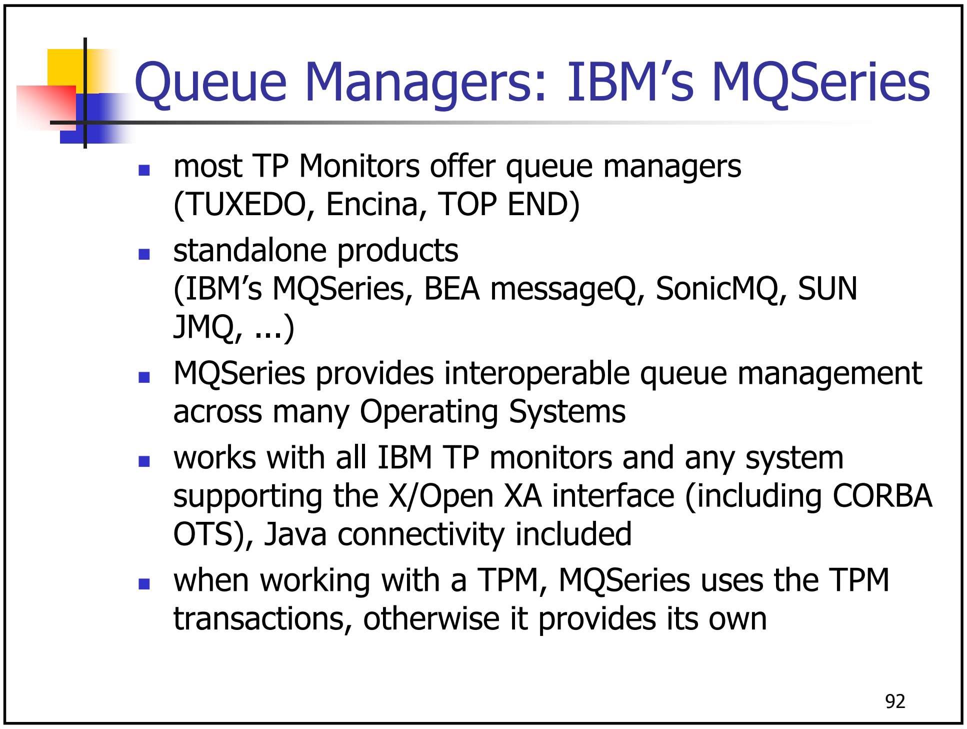
81



Wrap up

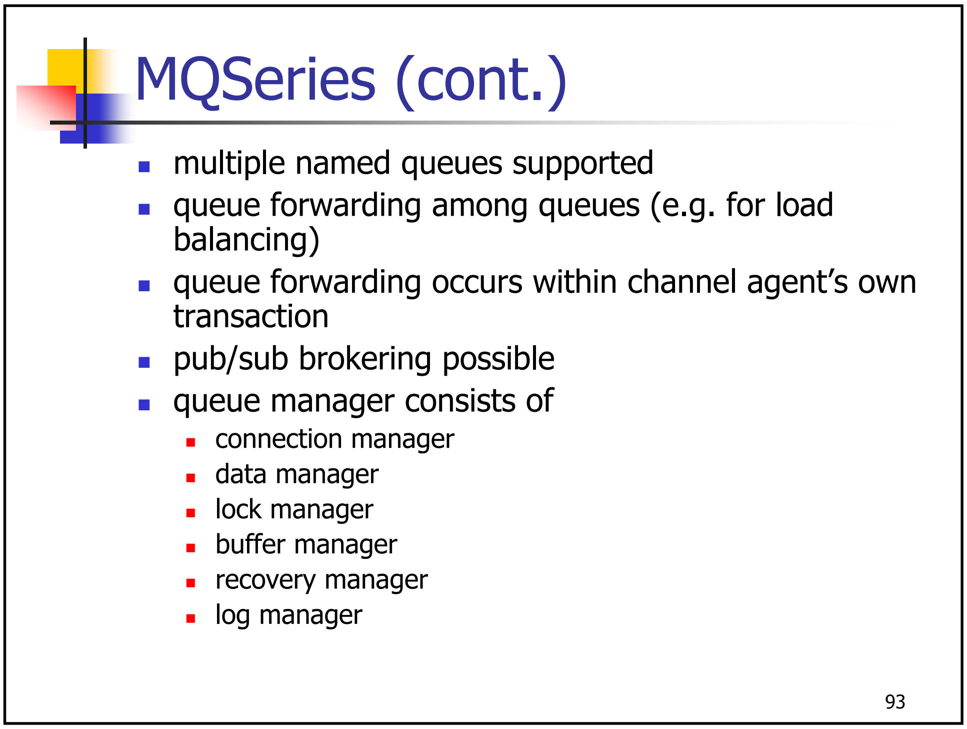
- Different routing strategies according to application needs
- Filters/subscriptions on a single message
- Event correlation no supported
  - Need to cache/store semi-composed events
- Software Engineering
  - Need to scope events and subdivide event space

Data Dissemination Products



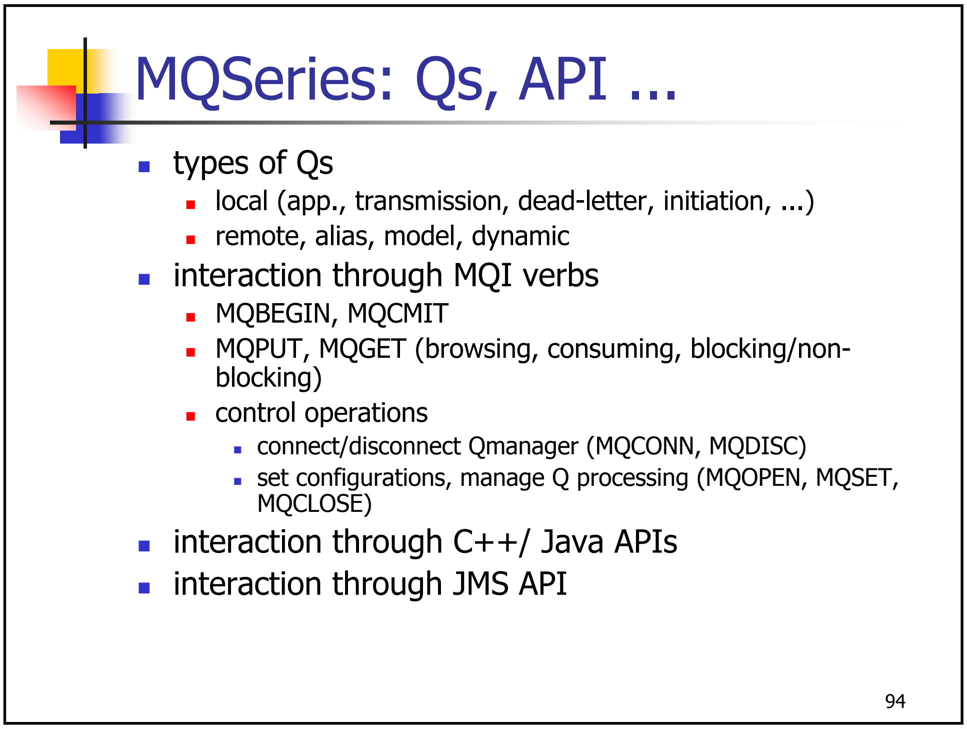
Queue Managers: IBM’s MQSeries

- most TP Monitors offer queue managers (TUXEDO, Encina, TOP END)
- standalone products (IBM’s MQSeries, BEA messageQ, SonicMQ, SUN JMQ, ...)
- MQSeries provides interoperable queue management across many Operating Systems
- works with all IBM TP monitors and any system supporting the X/Open XA interface (including CORBA OTS), Java connectivity included
- when working with a TPM, MQSeries uses the TPM transactions, otherwise it provides its own



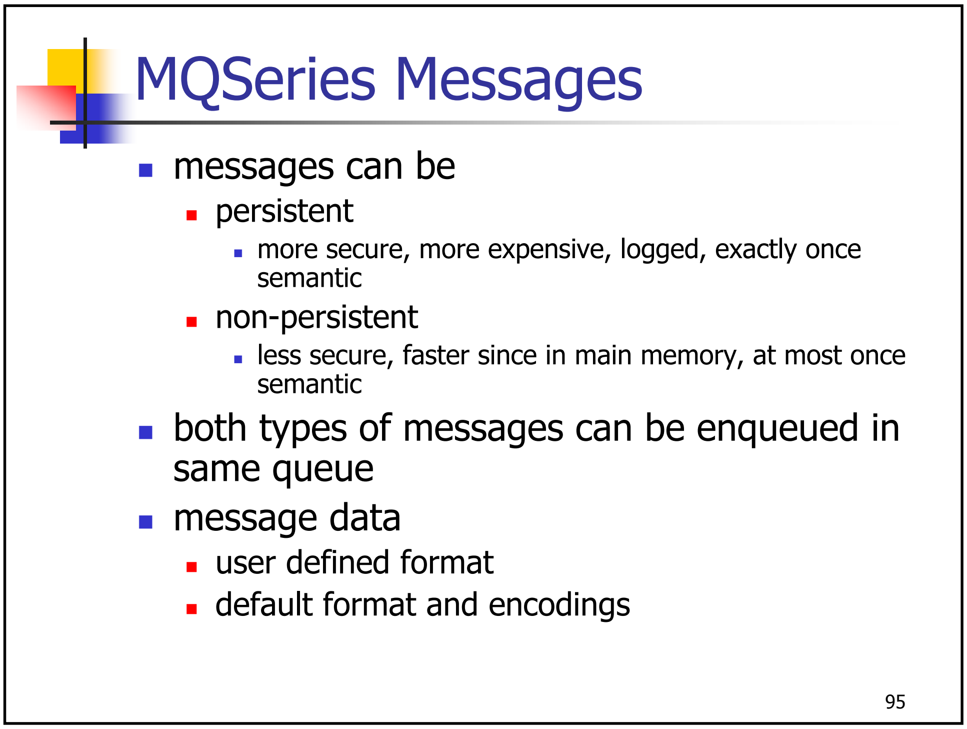
MQSeries (cont.)

- multiple named queues supported
- queue forwarding among queues (e.g. for load balancing)
- queue forwarding occurs within channel agent’s own transaction
- pub/sub brokering possible
- queue manager consists of
  - connection manager
  - data manager
  - lock manager
  - buffer manager
  - recovery manager
  - log manager



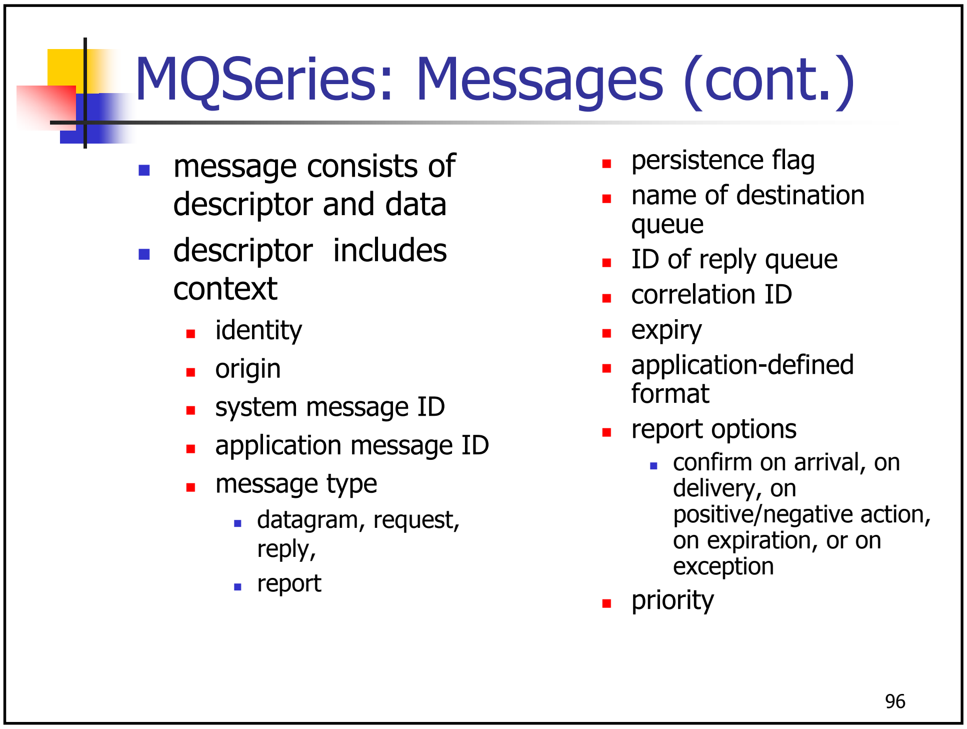
MQSeries: Qs, API ...

- types of Qs
  - local (app., transmission, dead-letter, initiation, ...)
  - remote, alias, model, dynamic
- interaction through MQI verbs
  - MQBEGIN, MQCMIT
  - MQPUT, MQGET (browsing, consuming, blocking/non-blocking)
  - control operations
    - connect/disconnect Qmanager (MQCONN, MQDISC)
    - set configurations, manage Q processing (MQOPEN, MQSET, MQCLOSE)
- interaction through C++/ Java APIs
- interaction through JMS API



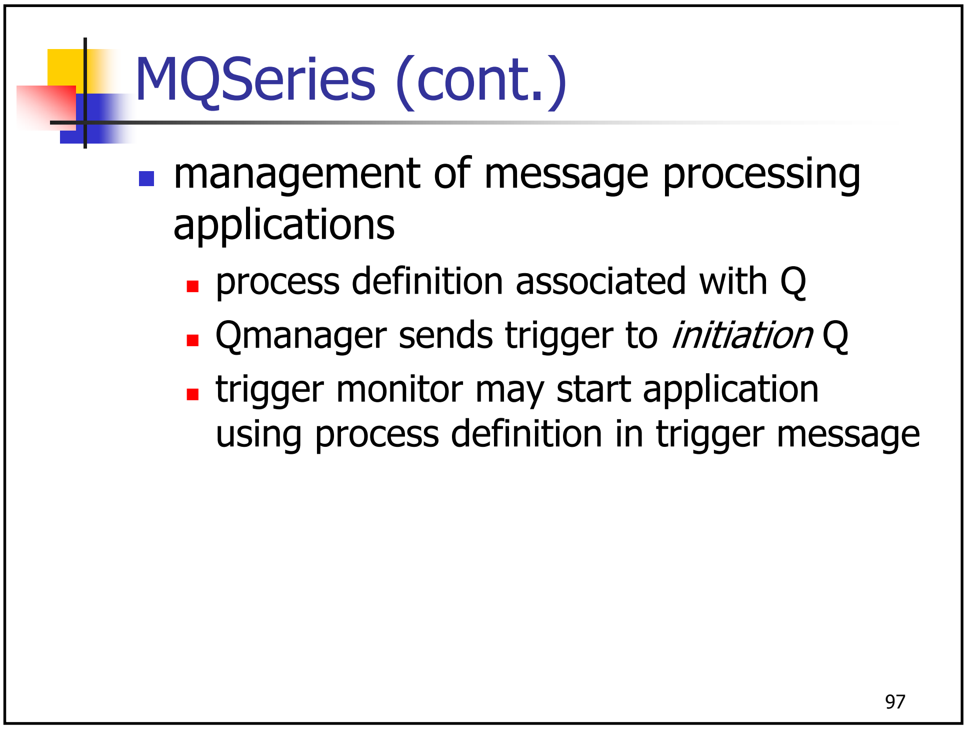
MQSeries Messages

- messages can be
  - persistent
    - more secure, more expensive, logged, exactly once semantic
  - non-persistent
    - less secure, faster since in main memory, at most once semantic
- both types of messages can be enqueued in same queue
- message data
  - user defined format
  - default format and encodings



MQSeries: Messages (cont.)

- message consists of descriptor and data
- descriptor includes context
  - identity
  - origin
  - system message ID
  - application message ID
  - message type
    - datagram, request, reply,
    - report
  - persistence flag
  - name of destination queue
  - ID of reply queue
  - correlation ID
  - expiry
  - application-defined format
  - report options
    - confirm on arrival, on delivery, on positive/negative action, on expiration, or on exception
  - priority



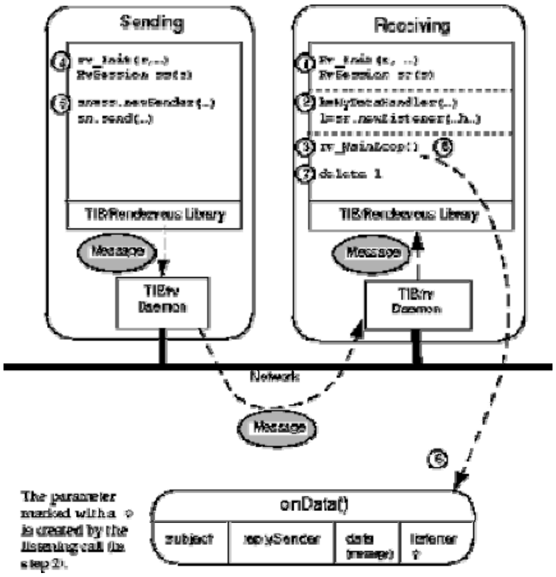
MQSeries (cont.)

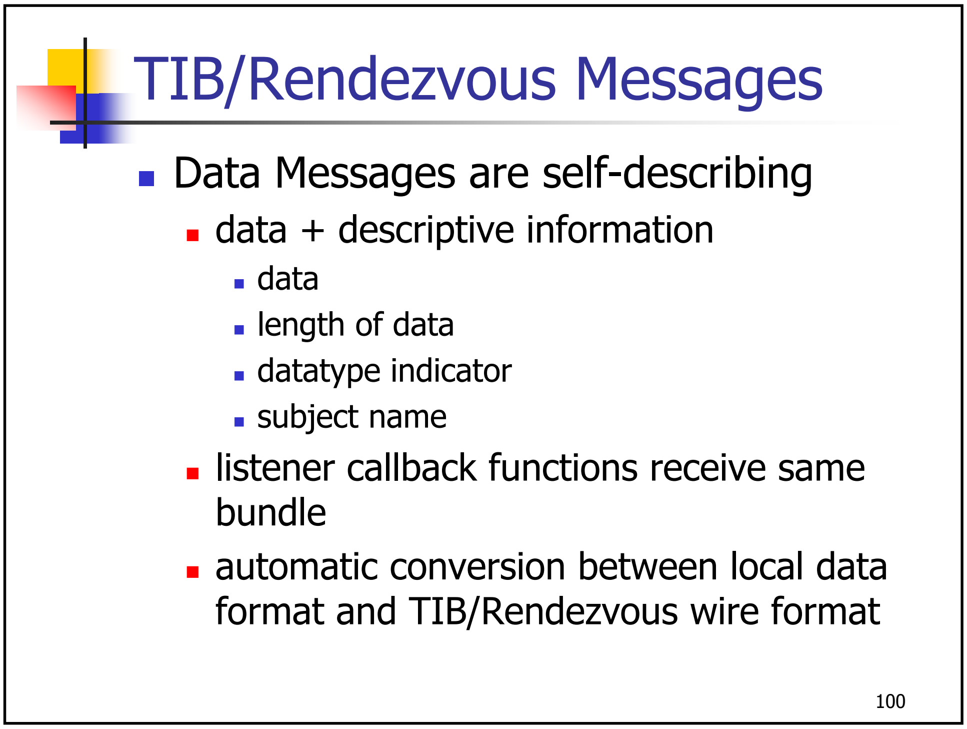
- management of message processing applications
  - process definition associated with Q
  - Qmanager sends trigger to *initiation* Q
  - trigger monitor may start application using process definition in trigger message

TIBCO's TIB/Rendezvous

- Event-driven, publish/subscribe
- Subject-based addressing
- Self-describing messages
- Leverage on broadcast & IP-multicast

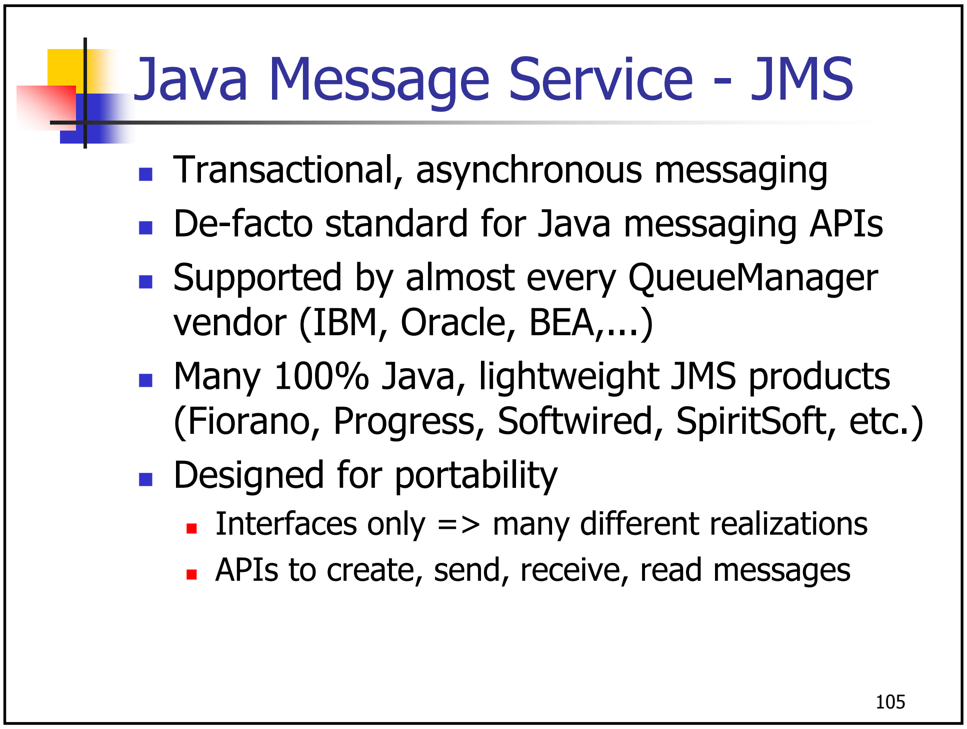
TIB/Rendezvous Architecture





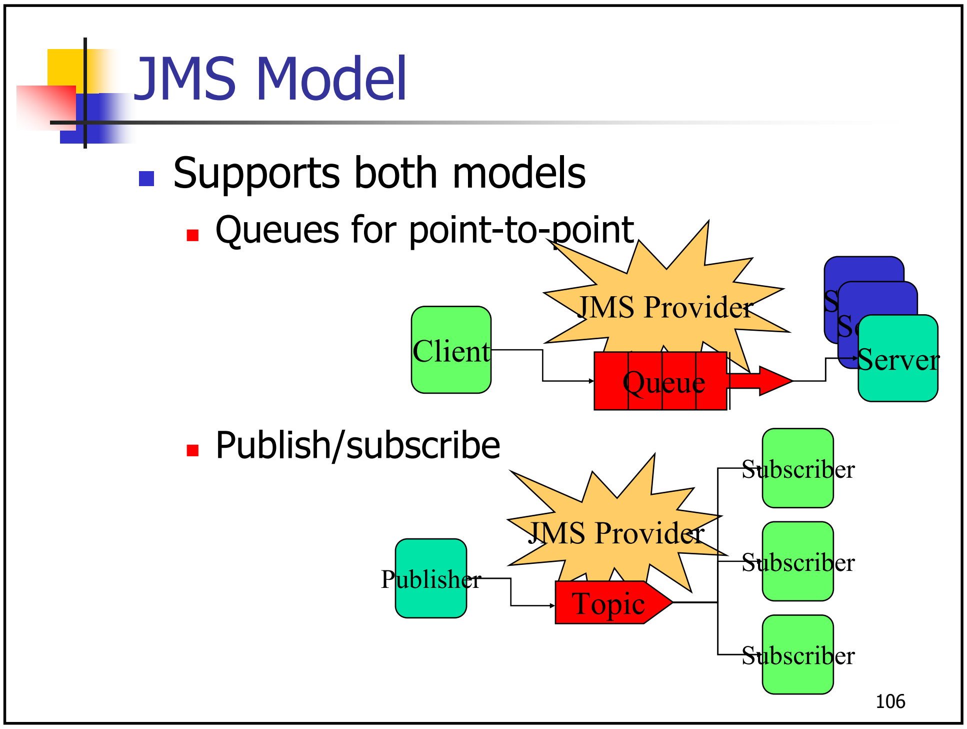
TIB/Rendezvous Messages

- Data Messages are self-describing
  - data + descriptive information
    - data
    - length of data
    - datatype indicator
    - subject name
  - listener callback functions receive same bundle
  - automatic conversion between local data format and TIB/Rendezvous wire format



Java Message Service - JMS

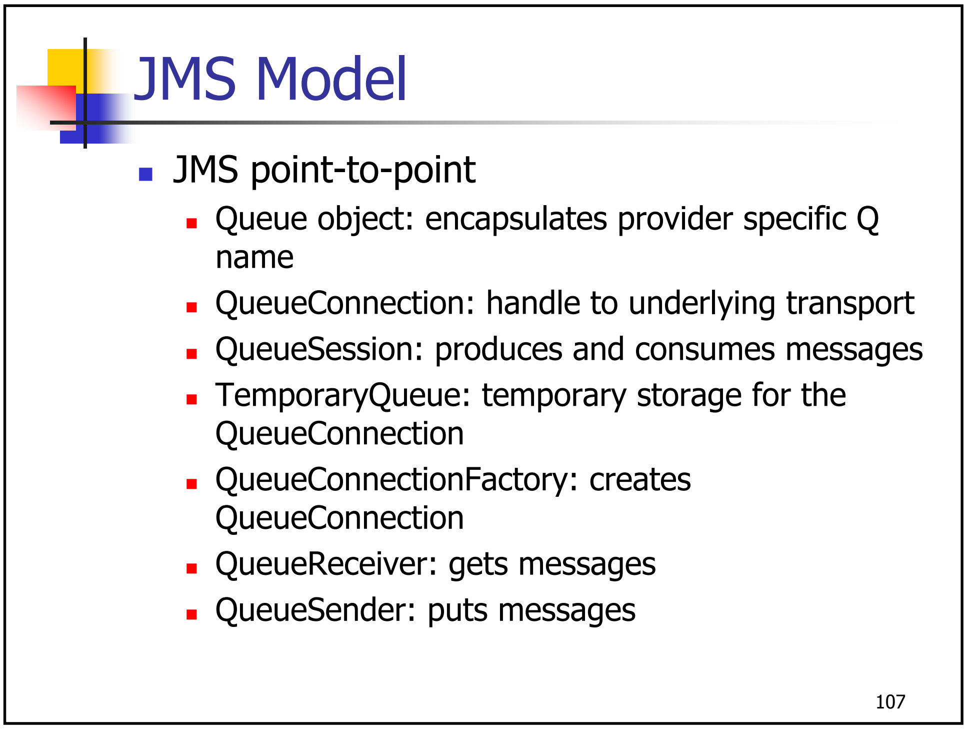
- Transactional, asynchronous messaging
- De-facto standard for Java messaging APIs
- Supported by almost every QueueManager vendor (IBM, Oracle, BEA,...)
- Many 100% Java, lightweight JMS products (Fiorano, Progress, Softwired, SpiritSoft, etc.)
- Designed for portability
  - Interfaces only => many different realizations
  - APIs to create, send, receive, read messages



JMS Model

- Supports both models
  - Queues for point-to-point
  - Publish/subscribe

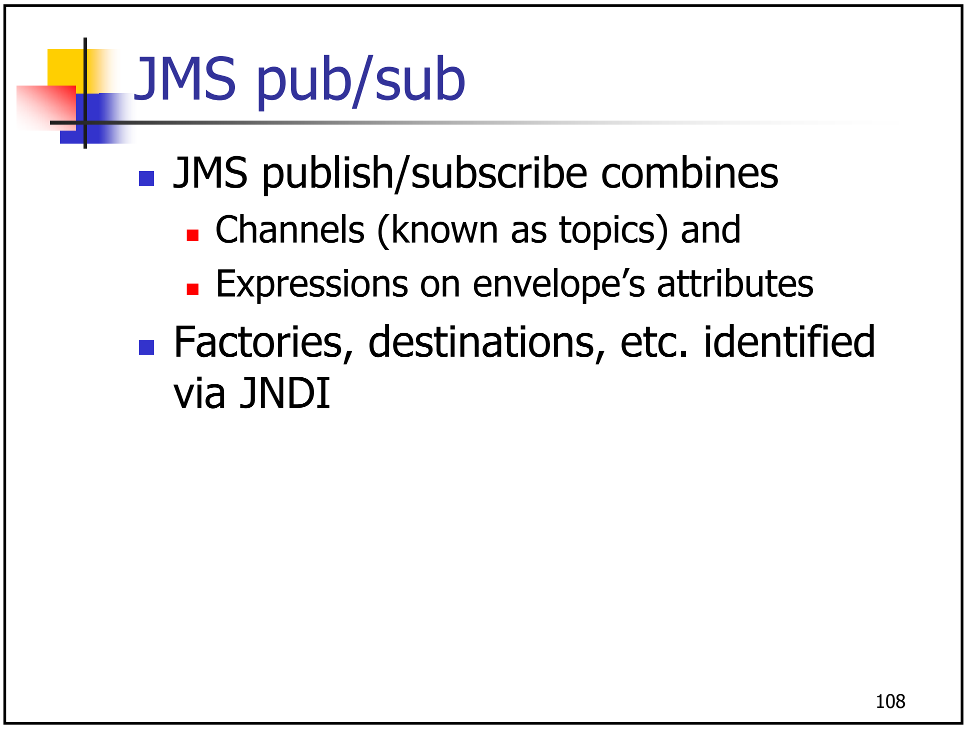
106



JMS Model

- JMS point-to-point
  - Queue object: encapsulates provider specific Q name
  - QueueConnection: handle to underlying transport
  - QueueSession: produces and consumes messages
  - TemporaryQueue: temporary storage for the QueueConnection
  - QueueConnectionFactory: creates QueueConnection
  - QueueReceiver: gets messages
  - QueueSender: puts messages

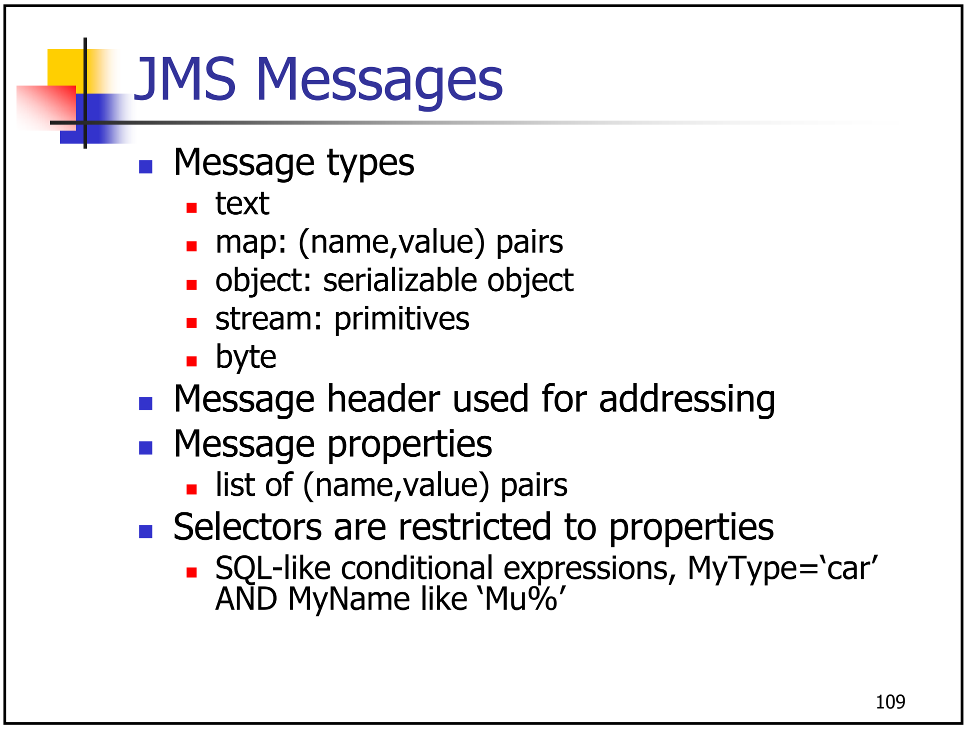
107



JMS pub/sub

- JMS publish/subscribe combines
  - Channels (known as topics) and
  - Expressions on envelope’s attributes
- Factories, destinations, etc. identified via JNDI

108



JMS Messages

- Message types
  - text
  - map: (name,value) pairs
  - object: serializable object
  - stream: primitives
  - byte
- Message header used for addressing
- Message properties
  - list of (name,value) pairs
- Selectors are restricted to properties
  - SQL-like conditional expressions, MyType=‘car’ AND MyName like ‘Mu%’

109

JMS Open Issues

- load balancing
- scalability/availability
- fault tolerance
- error notification
- end-to-end security
- segregation of domains
- simple and flexible deployment configuration
- Many of these issues being addressed by vendors