Programação em Sistemas Distribuídos

MEI-MI-MSI

2018/19

3. Models of Distributed Computing

Prof. António Casimiro

Distributed Computing Models

Main models, neither exhaustive nor air-tight

Ciências ULisboa

- Client-Server (RPC, RMI, WWW) (Cliente-Servidor)
- Distributed Objects (Objectos Distribuídos)
- Distributed Shared Memory (DSM, Tuples) (Memória Partilhada Distribuída)
- **Distributed Atomic Transactions (Transacções Atómicas Distribuídas)**
- Message-oriented (Message Queue, Publish/Subscribe) (Orientado para mensagens, Fila de Mensagens, Editor/Assinante)
- Stream (Corrente)
- Group-Oriented (Orientado para Grupos)
- Peer-to-peer (Inter-pares)

Distributed Atomic Transactions

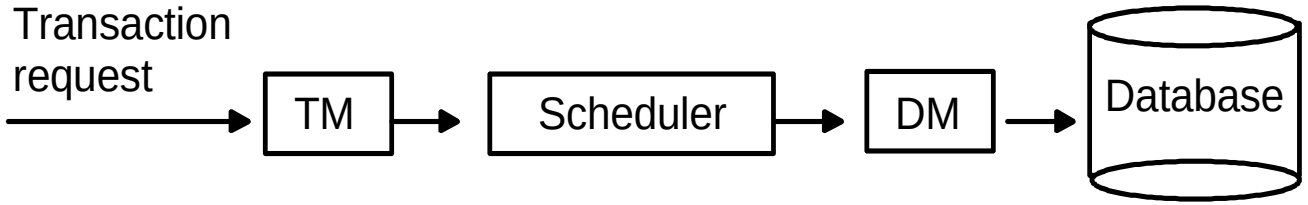
Atomic transactions

Ciências ULisboa

- Atomic action
  - Indivisible computation
- Sequential transaction
  - Technique to render a sequence of operations (e.g. atomic actions) indivisible
- ACID properties
  - Atomicity: undo, redo
  - Consistency: DB remains correct before and after an atomic transaction
  - Isolation (serializability): intermediate results are hidden
  - Durability: transaction effects are not lost

Transaction system architecture

- **Transaction Manager**, TM
  - Interface with the application. Supervises the execution of an atomic transaction, sequentially executing the necessary actions, assigning transaction IDs and invoking the DM through the SCh
- **Scheduler**, SCh
  - Responsible for concurrency control (**locks**)
- **Data Manager**, DM
  - Manages the DB, or the local partition of the DB, performing read and write operations and recovering from failures



Architectural aspects of transactional systems

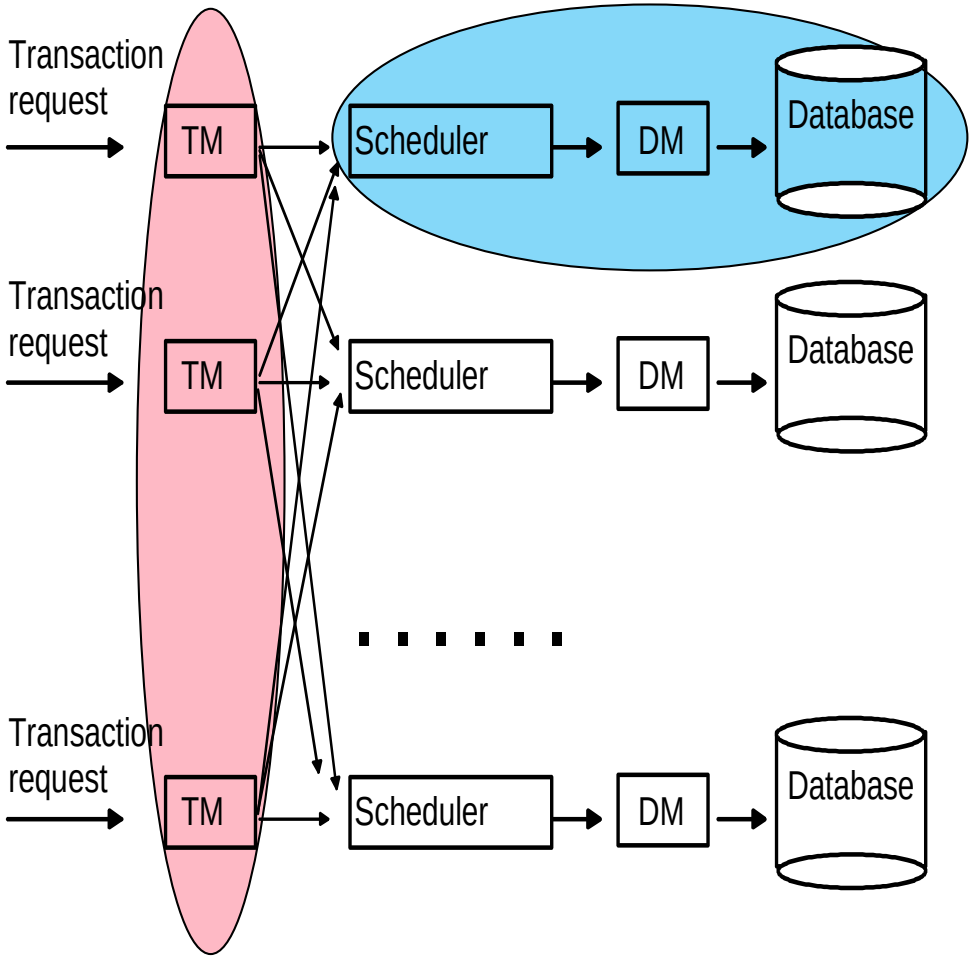
Ciências ULisboa

- Memory hierarchy
  - Volatile (RAM)
  - Non-volatile (disk)
  - Stable (e.g. RAID disks)
- Failures (at each memory level)

	Solution:
– **Abort**: undo of atomic transaction	Redo
– **Crash**: loss of volatile memory	Redo
– **Failure**: loss of non-volatile memory	Backup
– **Catastrophe**: loss of non-volatile and stable memory	Get pissed

Distributed and replicated DBs

- Execution of distributed atomic transaction
  - TMs issue sub-transactions to the appropriate nodes
  - Sub-transactions executed locally
  - Locks applied locally
- Advantages
  - DB performance and availability improve with fragmentation and replication
  - But network must be fast!!
- Consistency vs. Availability
  - Pessimistic vs optimistic concurrency control
  - Strong vs weak consistency



Building Applications with Distributed Atomic Transactions

Revisiting atomic transactions
Basics

Ciências ULisboa

- Based on data operations
  - Replicated or non-replicated data
  - Item - atomic read/write logical unit
  - ***writeset*** - items written by the atomic transaction
  - ***readset*** - items read by the atomic transaction
- Behavior
  - Language: ***begin*** transaction; ***end*** transaction
  - Execution: ***start***, ***commit***, ***abort***
  - Set of atomic actions in closed circuit
- Semantics
  - Exactly once
- Recovery
  - Abort, roll-back

Revisiting atomic transactions
Concurrent transactions

Ciências ULisboa

- Several transactions accessing same data concurrently
- **Problem**: How to achieve serializability?
- **Solution**: concurrency control
  - Two-phase lock (2PL)
  - Acquire, release lock (before and after data access)
  - One write, multiple reads
- **Problem** : 2PL leads to cascading aborts
  - Example. T1 writes data and releases lock; T2 reads data; T1 aborts; T2 also needs to abort
- **Solution** : Strict two-phase lock
  - No exclusive lock is released before the transaction commits
  - Intermediate results not visible before transaction commits

Revisiting atomic transactions
Concurrent transactions

Ciências ULisboa

- **Problem**: deadlock with 2PL
  - Cycles of atomic transactions mutually waiting for locks to be released
- **Solutions**:
  - Prevention:
    - Acquire locks in a orderly manner
    - Acquire all locks at once (hard to do - use only one lock…)
  - Avoidance:
    - Verify if acquiring a lock might lead to deadlock
  - Detection:
    - Detect possible deadlock situations (periodically check for waiting cycles, use timeouts) and abort to break deadlock

Distributed transactions

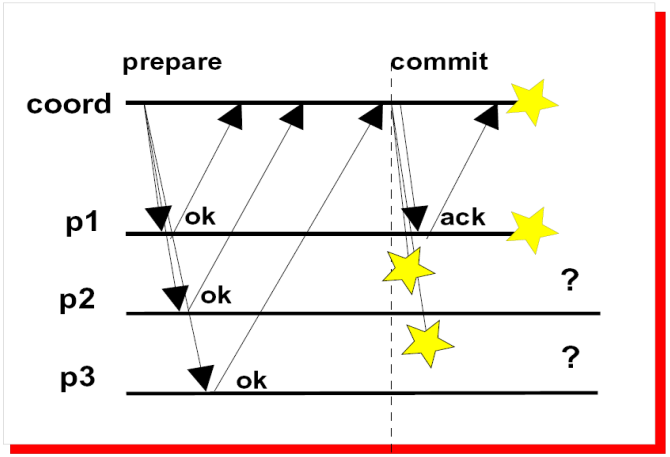
Ciências ULisboa

- Atomicity extended to actions performed in several nodes

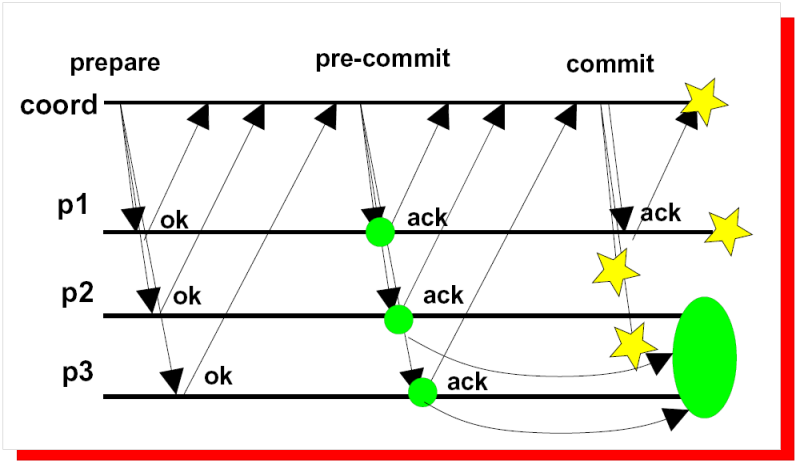
- **Problem**: partial failure of nodes and partitions, leading to imperfect commit
- **Solution**:
  - Distributed commit
    - Two-phase commit (2PC)
    - Three-phase commit (3PC)

Atomic commit protocols

- Two-phase commit
  - **Problem**: may block when the coordinator fails
  - Example:
    - Pi reply OK;
    - Coordinator sends COMMIT to some participants, and then fails;
    - Those participants also fail.



- Three-phase commit
  - **Advantage**: no blocking
    - Provided that majority of nodes is still alive
  - **Problem**: slower



Distributed & replicated transactions

Ciências ULisboa

- Serializability must be extended to actions performed in several nodes, and in several replicas

- **Problem**: conflicting updates in different replicas
- **Solution:** implement strong consistency
  - One-copy serializability (1C-SR)
  - Replication management algorithms:
    - 'read-one/write-all‘: assures 1C-SR, but blocks with partitions
    - 'read-one/write-all-available': 1C-SR within one partition
    - ‘quorums or primary partition’: 1C-SR despite partitions

Distributed Computing Models

Main models, neither exhaustive nor air-tight

Ciências ULisboa

- Client-Server (RPC, RMI, WWW) (Cliente-Servidor)
- Distributed Objects (Objectos Distribuídos)
- Distributed Shared Memory (DSM, Tuples) (Memória Partilhada Distribuída)
- Distributed Atomic Transactions (Transacções Atómicas Distribuídas)
- **Message-oriented (Message Queue, Publish/Subscribe) (Orientado para mensagens, Fila de Mensagens, Editor/Assinante)**
- Stream (Corrente)
- Group-Oriented (Orientado para Grupos)
- Peer-to-peer (Inter-pares)

Message-oriented models

Ciências ULisboa

- Abstraction allowing indirect exchange of messages amongst processes, through mediators or brokers
- Consumers are not directly addressed by producers, both resort to broker
- Processes can be producers and/or consumers
- **Broker**:
  - Node or set of distributed nodes that coordinate among themselves, whose purpose is to dispatch produced messages to interested consumers (and possibly only to them)
  - Message broker or message bus can be stateful or stateless
- **Space** and (potentially) **time decoupling**
  - Producer and consumer may not have to be simultaneously active (if broker is stateful, i.e. persistent)
  - Producer and consumer do not have to know each other

Message-oriented models
Applications

Ciências ULisboa

- Broadcasting of live data
  - RSS feeds (news reports, stock exchanges, …)
  - Satellites' data, etc.
- Monitoring
  - The web: issue subscriptions for pages' updates, etc.
  - The network/system: track requests with specific IPs, etc.
  - Critical infrastructure metrology
- Cooperation
  - Bio-scientists sharing results, (publications) creating a global compendium with notifications to interested researchers…
- Interconnection of heterogeneous applications
  - Message abstraction as connector
- Integration of legacy systems and software
  - Black-box applications into new distributed-systems platforms
- Gaming
  - Multi-player computer games

Message-oriented

Message Queues

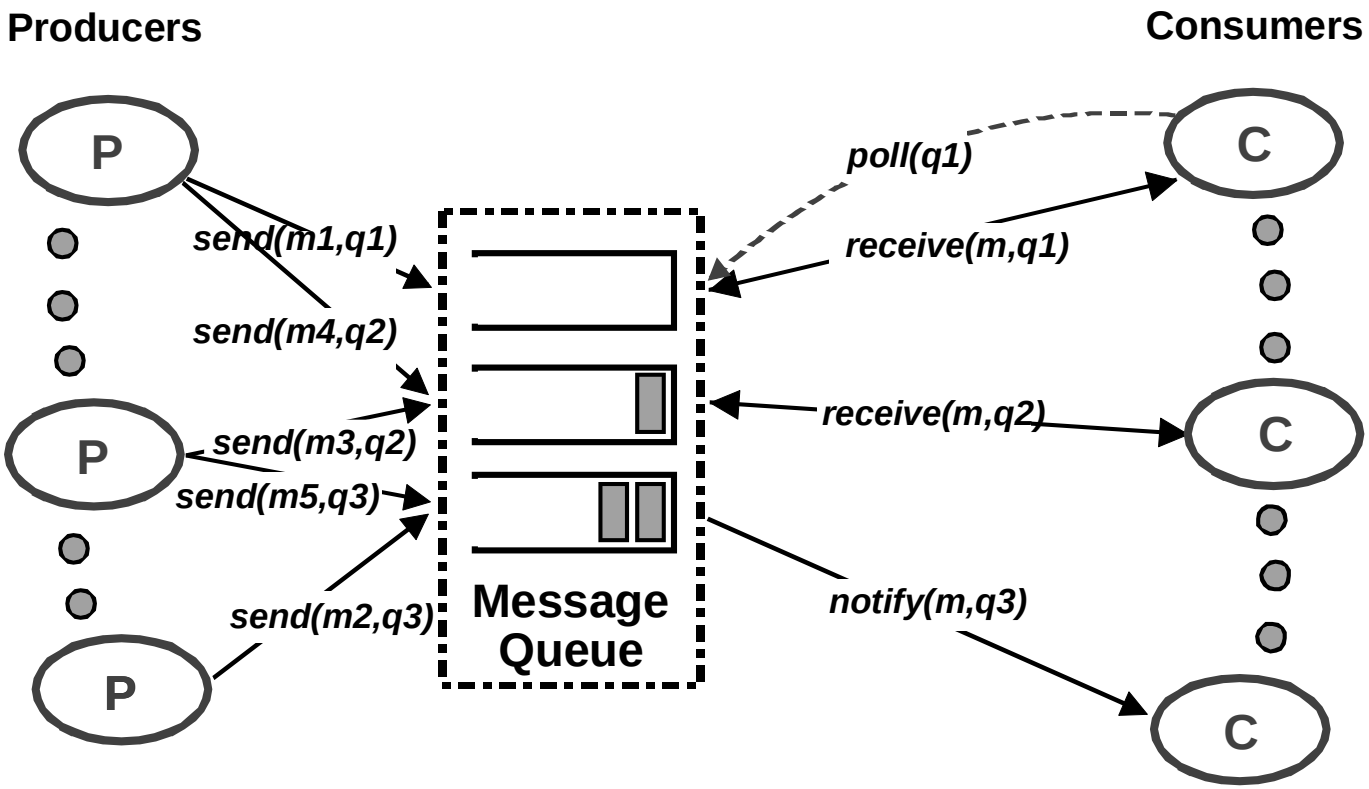
(ALSO CALLED “MESSAGE-ORIENTED MIDDLEWARE”, MOM, OR POINT-TO-POINT MESSAGE MODEL)

Message-oriented models
Message Queues

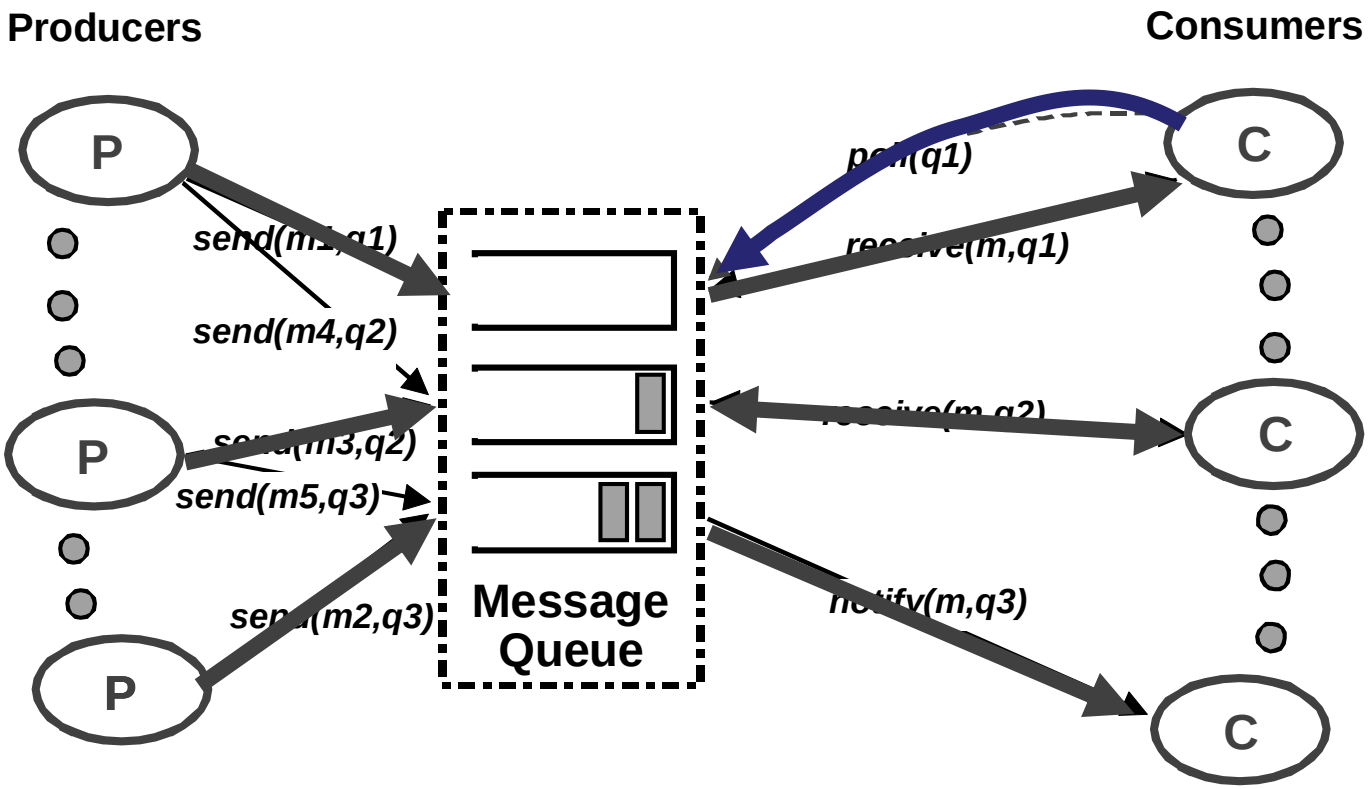
Ciências ULisboa

- **Time and space decoupling**
  - Producer and consumer do not have to be simultaneously active
  - Producer and consumer do not have to know each other
- **Synchronization decoupling**
  - Not provided, since consumers synchronously pull messages
- **Message delivery**
  - Stateful message queue broker (persistent)
- **Addressing based on:**
  - Mailboxes
  - Many-to-one
- **Reliable delivery service**
  - Queues are kept in persistent storage
- **Brokers may do filtering, transforming, logging**
  - To deal with heterogeneity in underlying data representations

Message Queue
Information flow



Message Queue
Information flow



Message Queue
Generic architecture

Ciências ULisboa

- **filtering policies**
  - According to the queuing discipline: FIFO, priority, select

Message matching

  - Supports checking a message against consumption rules, to determine whether or not to dispatch it to the consumer
- **forwarding strategies**
  - According to routing type: ptp, many-to-1, 1-to-many

Message Routing

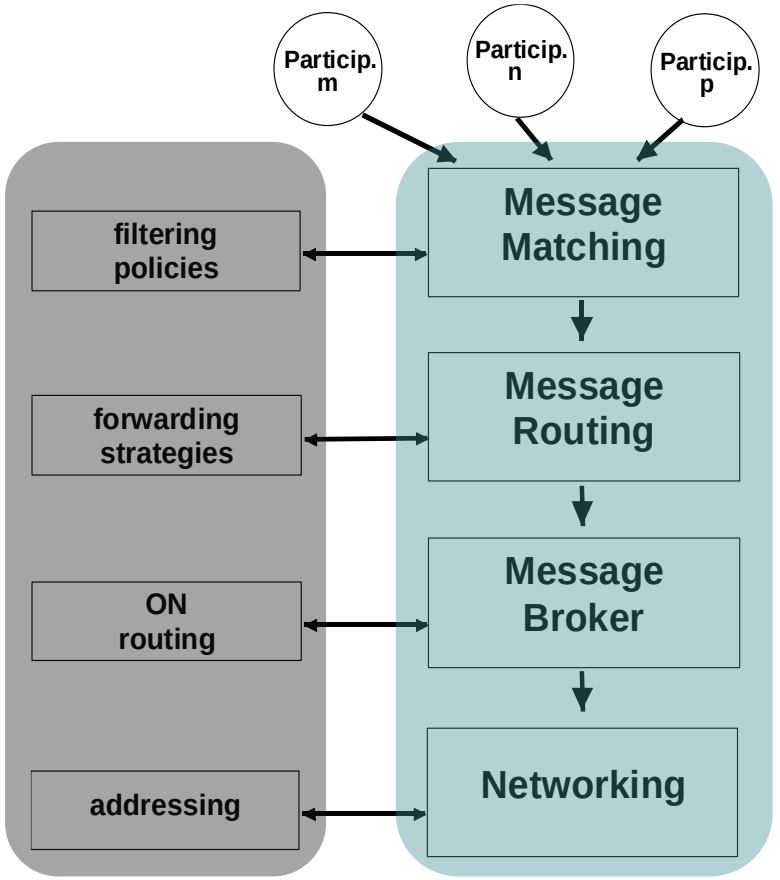
  - Supports delivery of a message from producer, amongst brokers, to intended destination consumer queue(s)
- **ON routing**
  - According to brokerage type and topology

Message Broker

  - Supports intermediation between producers and consumers, managing storage and persistence, transformation, transactions
- **addressing**
  - According to network capabilities

Networking

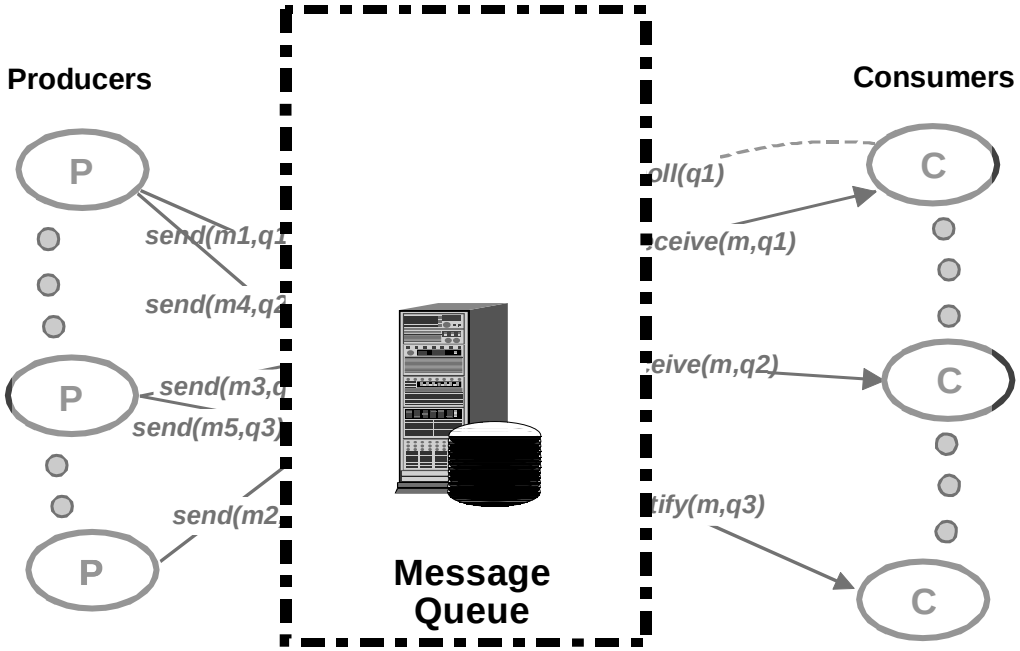
  - Supports unreliable or reliable message transmission services



Message Queue Message Broker
Architectural alternatives

Ciências ULisboa

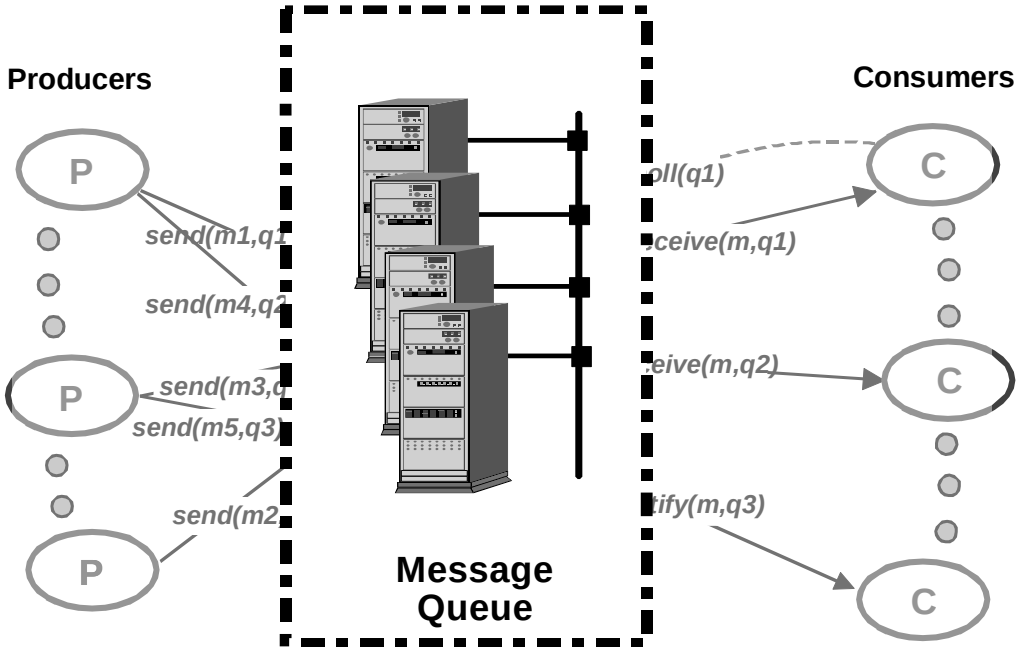
- Centralized
  - Single queue mgt engine
  - Persistent
  - Limited scalability



Message Queue Message Broker
Architectural alternatives

Ciências ULisboa

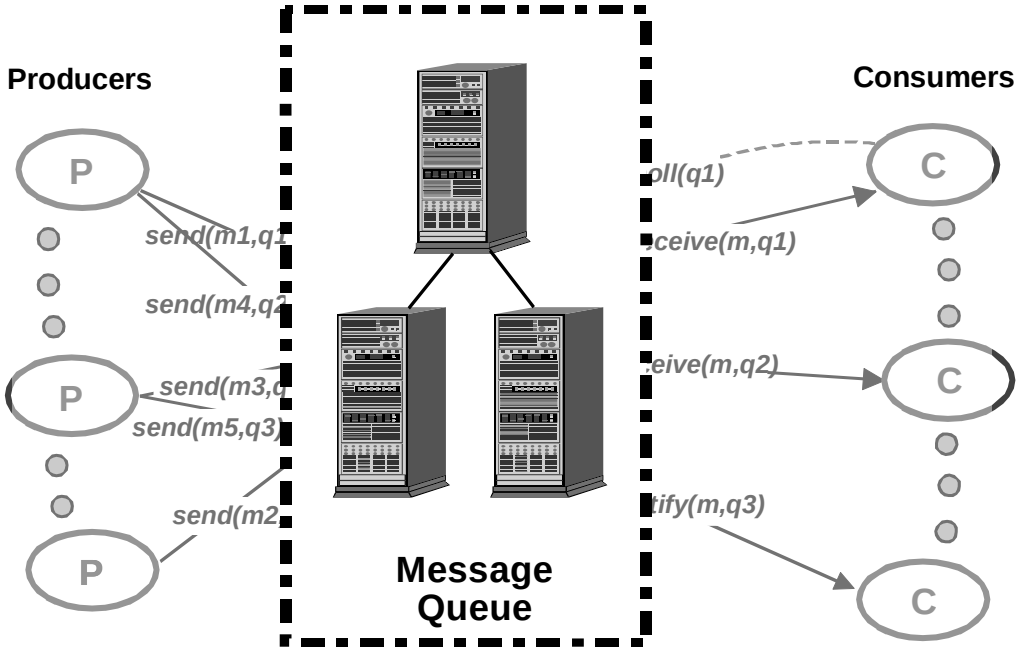
- Centralized
  - Single queue mgt engine
  - Persistent
  - Limited scalability
  - **Expandable in parallel bus**



Message Queue Message Broker
Architectural alternatives

Ciências ULisboa

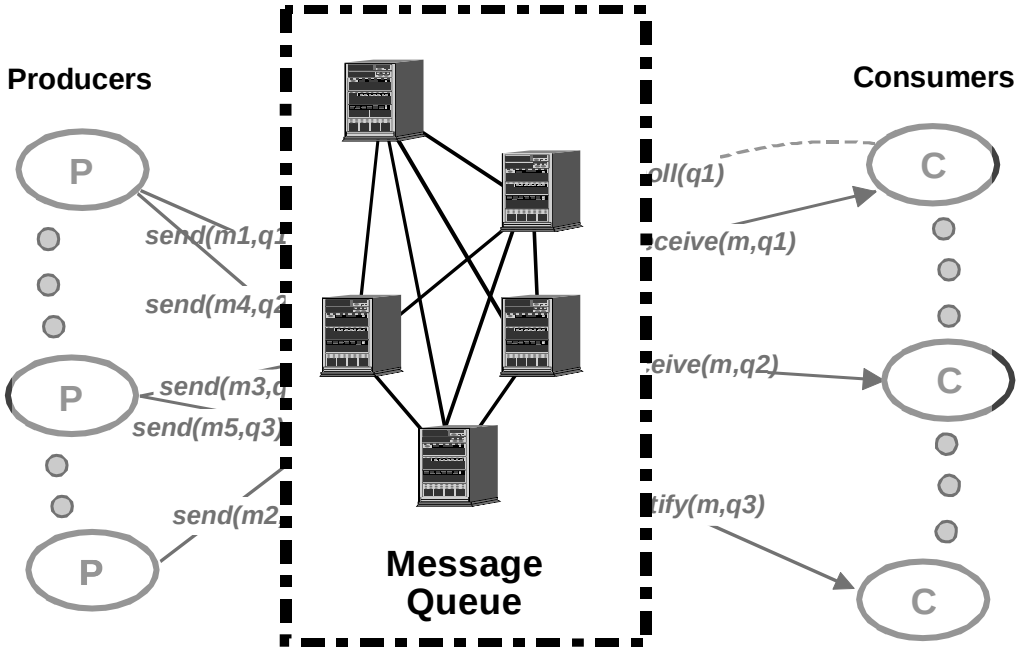
- Centralized
  - Single queue mgt engine
  - Persistent
  - Limited scalability
  - **Expandable in parallel bus**
- Broker network
  - Multiple M/Q brokers



Message Queue Message Broker
Architectural alternatives

Ciências ULisboa

- Centralized
  - Single queue mgt engine
  - Persistent
  - Limited scalability
  - **Expandable in parallel bus**
- Broker network
  - Multiple M/Q brokers
  - Several structures: tree, mesh, …
  - Participants connect to some broker depending on structure
  - Messages routed through overlay network



Building Message-Oriented Applications

Message Queues

Message Queue
Generic Interface

Ciências ULisboa

- ***send(m,q)***
  - Producer sends a message *m* to a destination queue *m*
- ***receive(q)***
  - Consumer will block on queue *q* until an appropriate message is available
  - Depending on ordering discipline, is normally message at the head (FIFO)
- ***poll(q)***
  - Consumer checks status of the queue *q*
  - Returns a message if available, or a ‘not-available’ indication otherwise
- ***notify(q)***
  - Issues an event notification when a message is available in the queue

Message Queue Implementation
Message delivery

Ciências ULisboa

- Different strategies for message delivery
  - Blocking receive
  - Non-blocking receive
  - Notification (upcall)
- Order
  - Standard delivery is reliable FIFO
  - Many systems have provisions for priority based overtaking
  - Others allow consumers to make select operations based on message properties
- Hybridization
  - Several systems compound message queuing with publish/subscribe (e.g., JMS, Java Messaging Service; AMQP, Advanced Message Queuing Protocol)

Message-oriented

Message Bus
(or Publish/Subscribe)

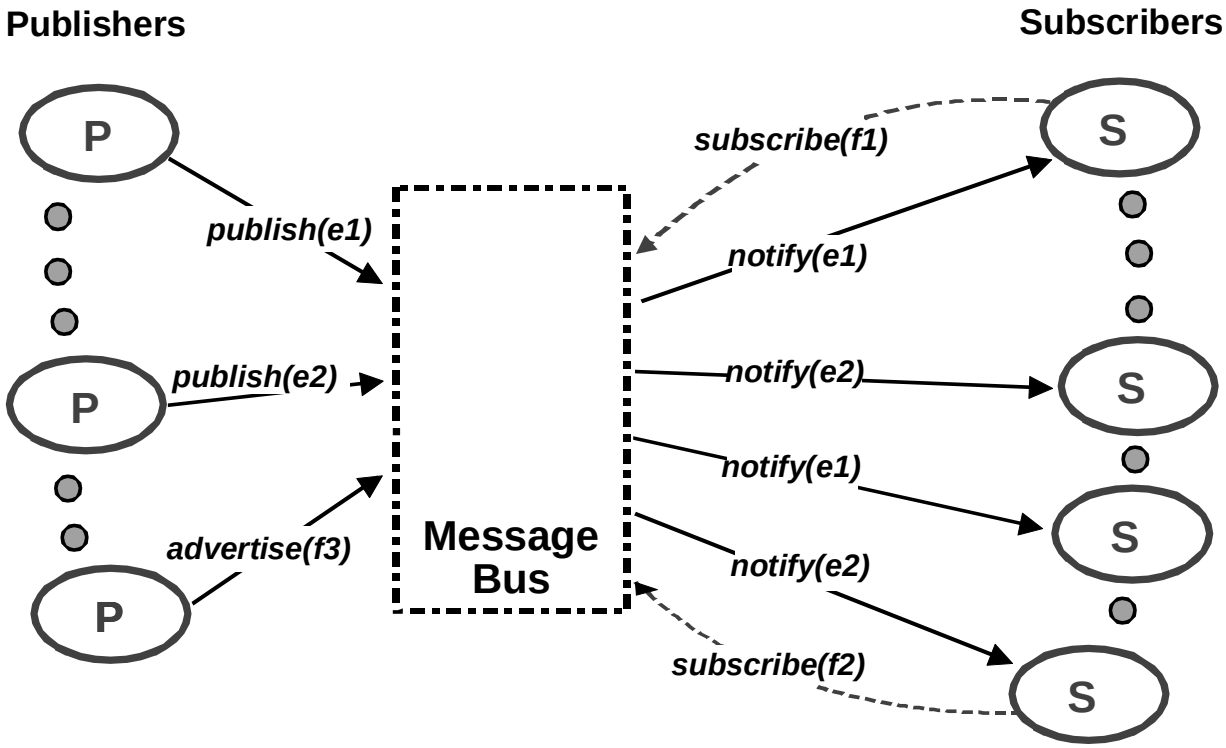
(ALSO CALLED EVENT BUS, DISTRIBUTED EVENT-BASED, EVENT NOTIFICATION)

Message-oriented models
Message Bus or Publish/Subscribe

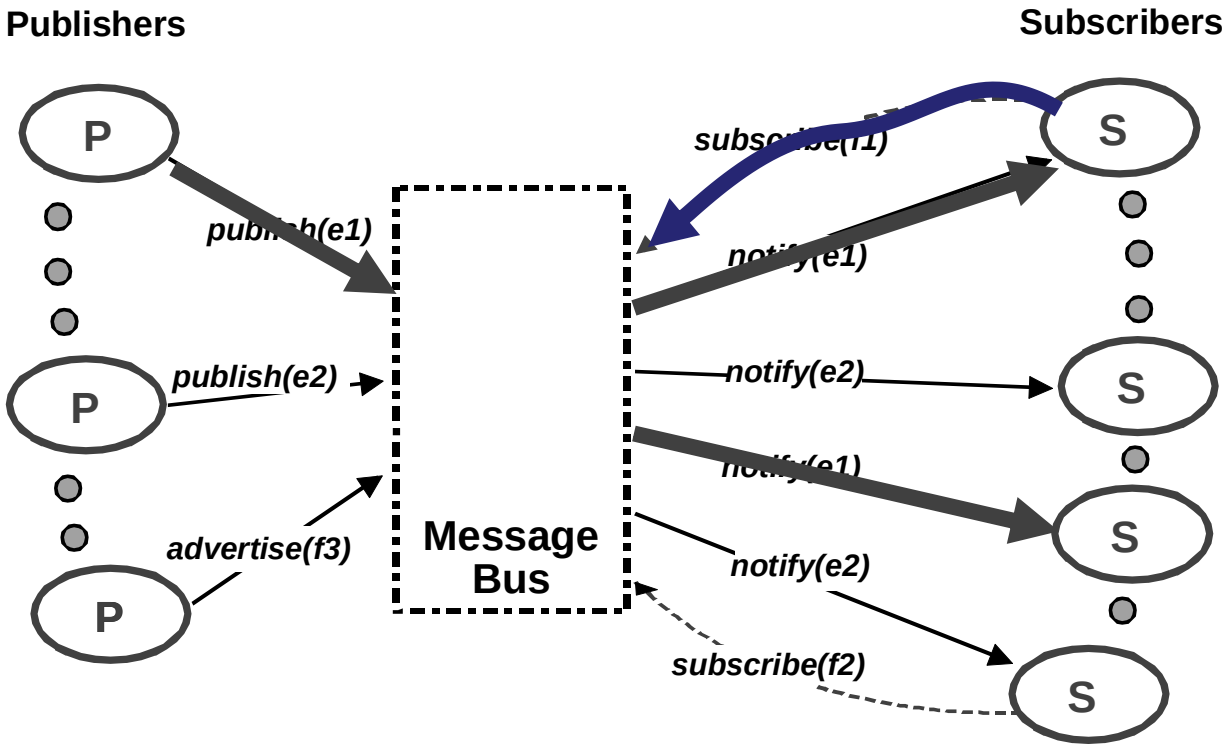
Ciências ULisboa

- **Space and (potentially) time decoupling**
  - Producer and consumer do not have to know each other
  - Producer and consumer are normally simultaneously active (stateless broker)
  - Producer and consumer may not have to be simultaneously active (stateful broker)
- **Synchronization decoupling**
  - Publishers are not blocked while producing events, and subscribers get asynchronously notified of the occurrence of an event
- **Event delivery**
  - Stateful or stateless message dissemination broker
  - Several vehicles: central broker, network multicast channels, broker overlay network, structured peer-to-peer
- **Addressing based on:**
  - Topic, content, type
- **Scalability, in high numbers:**
  - (Concurrent) events; live subscriptions; participants (publishers or subscribers); geographical distance of participants and resources

Publish/subscribe
Information flow



Publish/subscribe
Information flow



Publish/subscribe
Generic architecture

Ciências ULisboa

- **filtering policies**
  - According to the addressing type

Event matching

  - Supports checking an event against a subscription, to determine whether or not to dispatch it to a subscriber
- **dissemination strategies**
  - According to routing type: flooding, filter-based, gossip

Event Routing

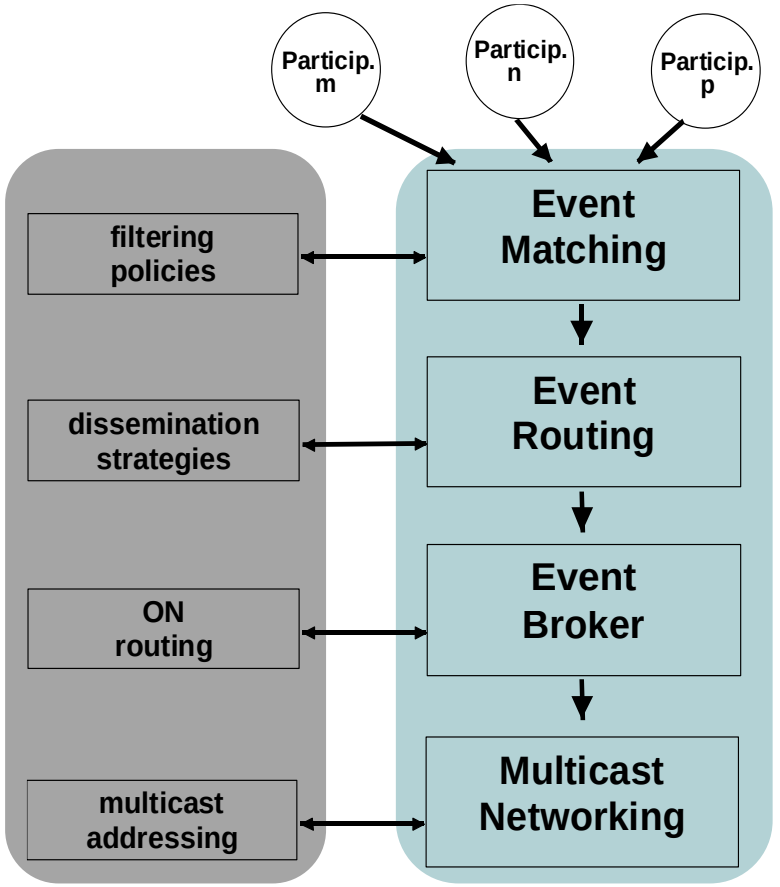
  - Supports delivering an event to all the subscribers that issued a matching subscription before the publication
- **ON routing**
  - According to brokerage type

Event Broker

  - Supports intermediation between publishers and subscribers, managing subscriptions and advertisements, storage
- **multicast addressing**
  - According to network capabilities

Multicast Networking

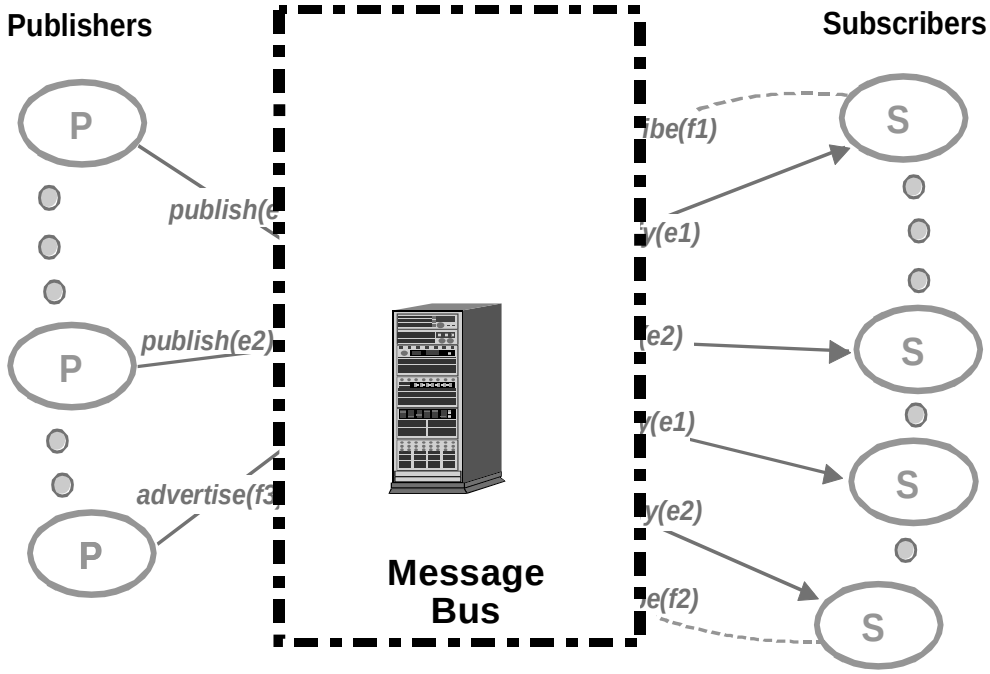
  - Supports unreliable message multicast services



Pub/Sub Message Broker
Architectural alternatives

Ciências ULisboa

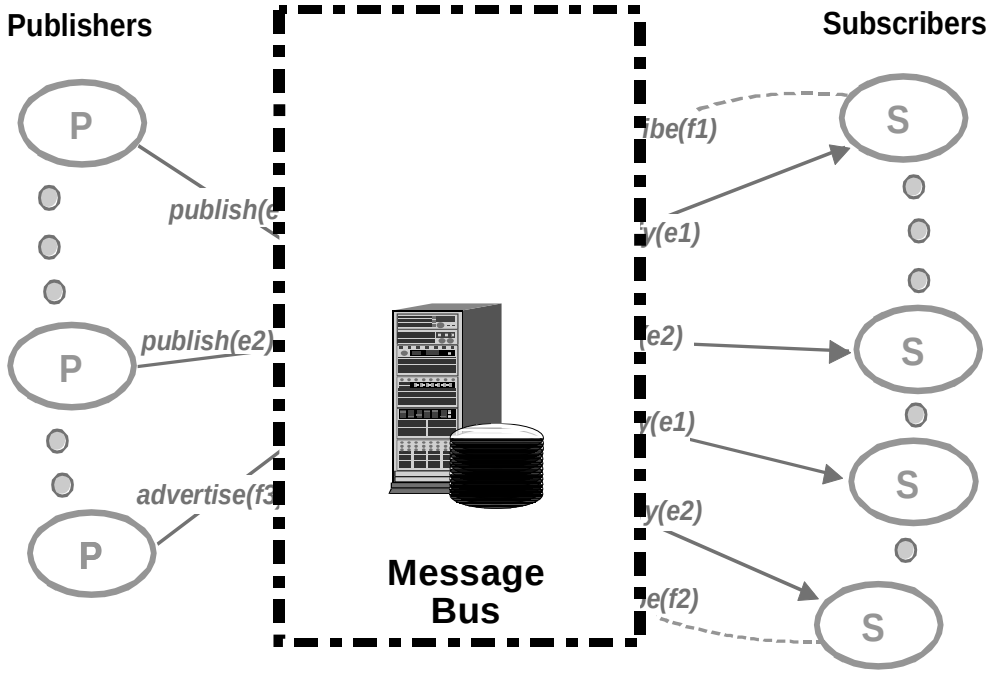
- Centralized
  - Single matching engine
  - Limited scalability



Pub/Sub Message Broker
Architectural alternatives

Ciências ULisboa

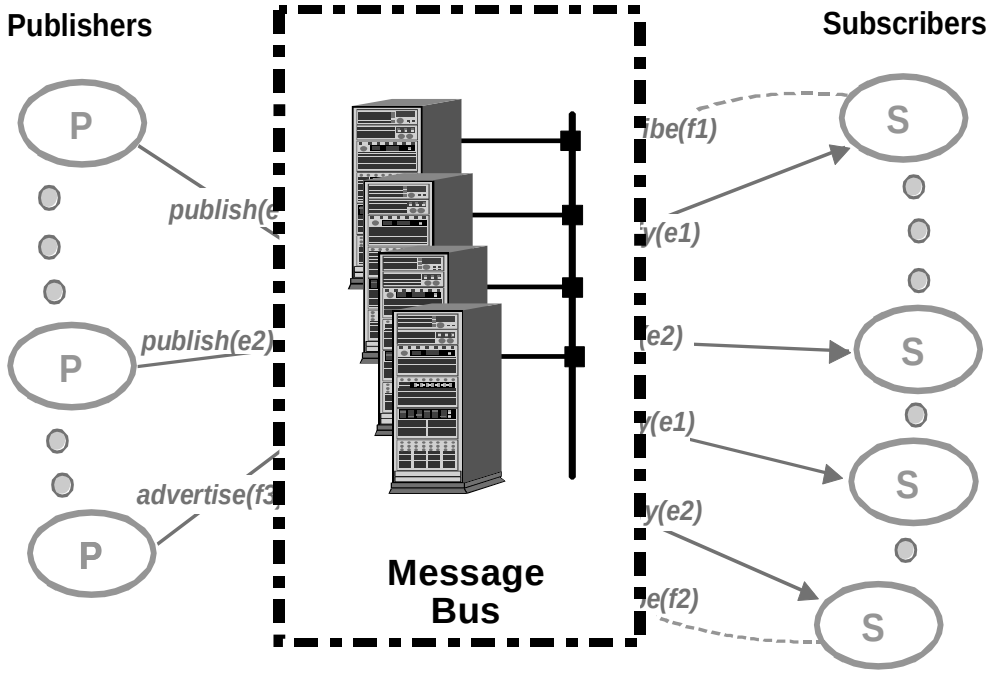
- Centralized
  - Single matching engine
  - Limited scalability
  - Possibly persistent



Pub/Sub Message Broker
Architectural alternatives

Ciências ULisboa

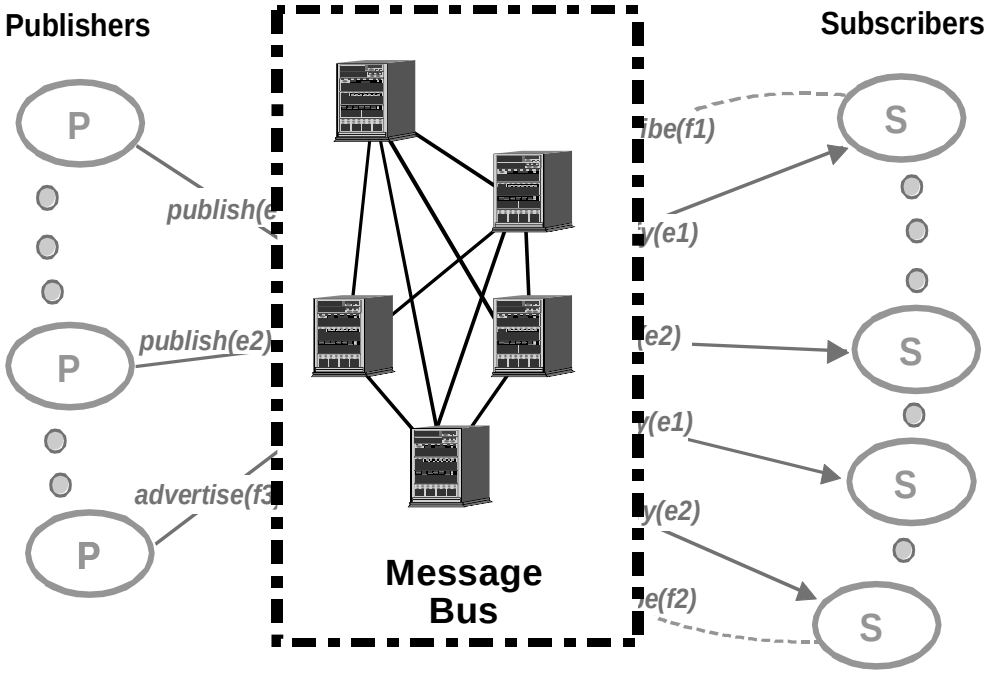
- Centralized
  - Single matching engine
  - Limited scalability
  - Possibly persistent
  - **Local replication for F/T**



Pub/Sub Message Broker
Architectural alternatives

Ciências ULisboa

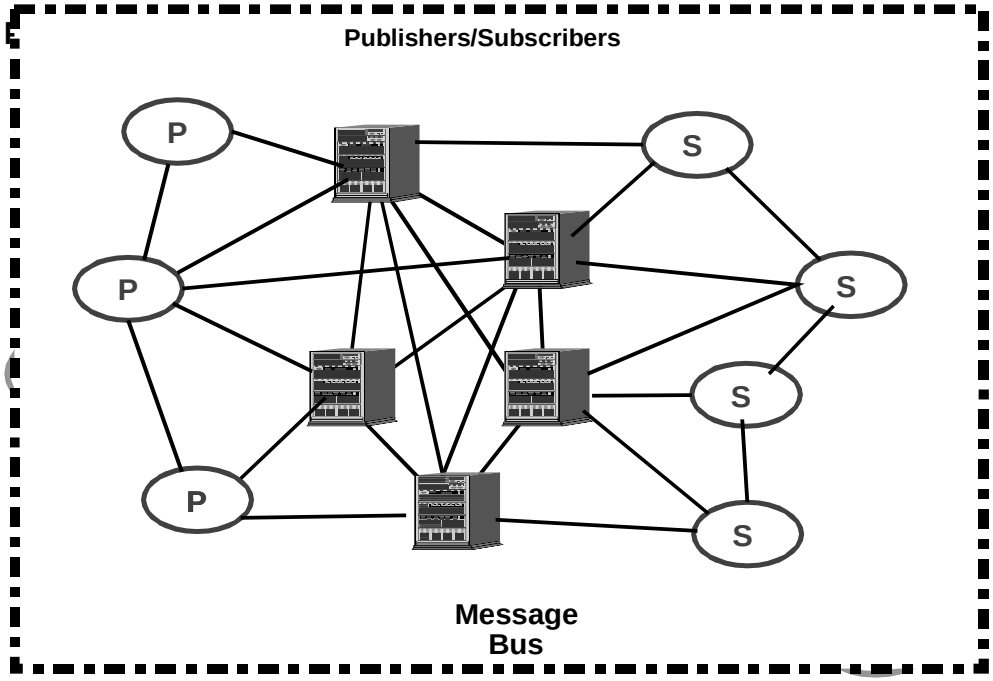
- Centralized
  - Single matching engine
  - Limited scalability
  - Possibly persistent
  - **Local replication for F/T**
- Broker network
  - Multiple P/S brokers
  - Participants connect to some broker
  - Events routed through overlay
  - F/T by redundant paths and/or broker replication



Pub/Sub Message Broker
Architectural alternatives

Ciências ULisboa

- Centralized
  - Single matching engine
  - Limited scalability
  - Possibly persistent
  - **Local replication for F/T**
- Broker network
  - Multiple P/S brokers
  - Participants connect to some broker
  - Events routed through overlay
  - F/T by redundant paths and/or broker replication
- Peer-to-peer network
  - Publishers & subscribers connected in P2P network
  - Participants collectively filter/route events, can be both producer & consumer



Publish/subscribe
Addressing and routing

Ciências ULisboa

- Multicast addressing
  - Take advantage of MAC and IPmcast, map on group addressing
- Flooding alternatives
  - Propagate events to all nodes, filter at consume points (does not scale)
  - Propagate subscription back to producers, filter at produce points (routing becomes more intelligent; non-subscribed events immediately filtered out)
- Filter-based routing
  - Filter non-useful routes, inside broker overlay network (events are forwarded only through path leading to interested subscribers)
  - Non-subscribed events filtered out asap

Building Message-Oriented Applications

Publish/Subscribe

Publish/Subscribe
Generic Interface

Ciências ULisboa

- ***advertise(f)***
  - Publishers may advertise the nature of their future events (pattern *f*)
  - Pattern may be on a topic, content, type
  - Event routing may adjust to the expected flows of events, subscribers may learn of new types of information available
- ***subscribe(f)***
  - Subscribers register interest in events with pattern *f*, with message bus
- ***publish(e)***
  - Publishers generate event *e* for the message bus to propagate
- ***notify(e)***
  - Message bus notifies subscribers of event *e* conforming to their interest (push mode)
- ***read(e,f)***
  - Subscriber interacts with message bus in pull or bulletin board mode, interested in the next event with pattern *f*
- ***unsubscribe(f)***
  - Subscribers remove registration of interest in events with pattern *f*

Publish/subscribe
Subscription model

Ciências ULisboa

- **Channel-based**
  - Subscribe & publish to a channel
  - Subscribers receive all events published in channel
  - Ex.: OMG CORBA Event Service, …
- **Topic-based (a.k.a. subject-based)**
  - Subscribe & publish to topics and topic hierarchies
  - Ex.: TIBCO, WS Notifications, …
- **Type-based**
  - Subscribe & publish to typed objects
  - Filters events according to their type, instead of topic name
  - Closer integration of language, type safety
  - Ex.: OMG Data Dissemination Service (partially), …
- **Content-based**
  - Subscribe & publish to content of messages
  - Ex.: Gryphon, JMS metadata, PADRES ESB

Message Bus Implementation
Event delivery

Ciências ULisboa

Two main strategies for event/message delivery

1. **Information push (notification)**
   - Consumers register interest in messages (channel, topic, content)
   - Messages are immediately forwarded to them
   - If broker is persistent, consumers may not be always active, and messages are delivered when they sign back on
2. **Information pull (bulletin board)**
   - Consumers do not have to be always active
   - Broker is persistent, consumers periodically check new events and read messages in which topic they are interested

Message Bus Implementation
Dependability

- Single broker is single point of failure
- Centralized broker can be replicated locally (e.g. same data centre)
- Broker network offers wide-area solutions
  - Geographical separation
  - Natural path redundancy
  - Possible redundancy by explicit broker replication
- Replica management overhead
- Delivery guarantees
  - Event delivery may or not be reliable and ordered, or fulfil R/T requirements