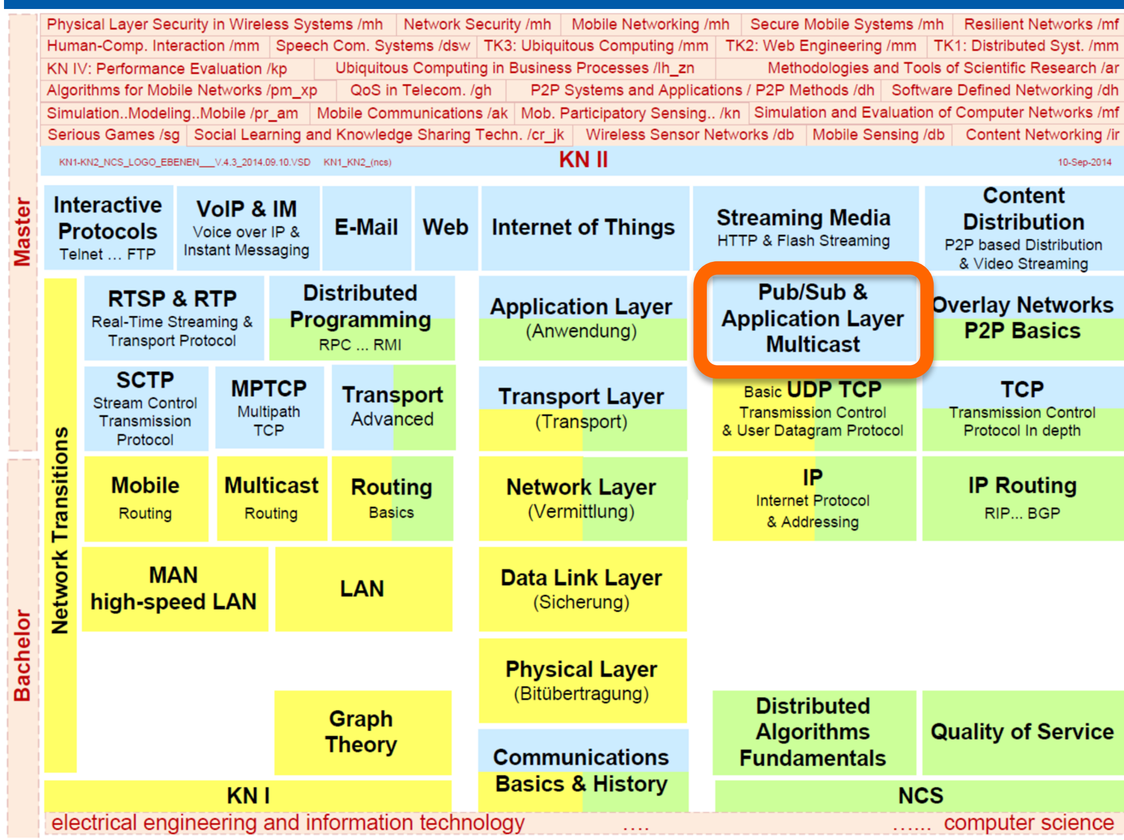
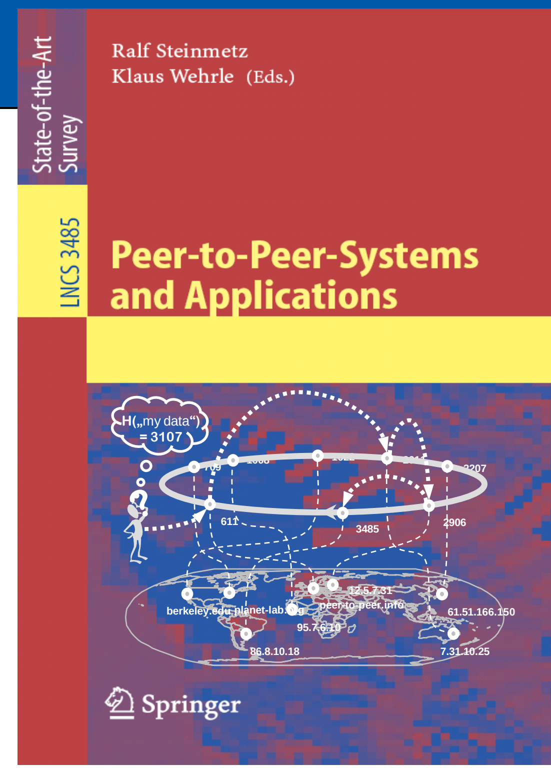
Communication Networks II

Application Layer Multicast - ALM

(including Event-based Systems,
Publish – Subscribe Pub / Sub)



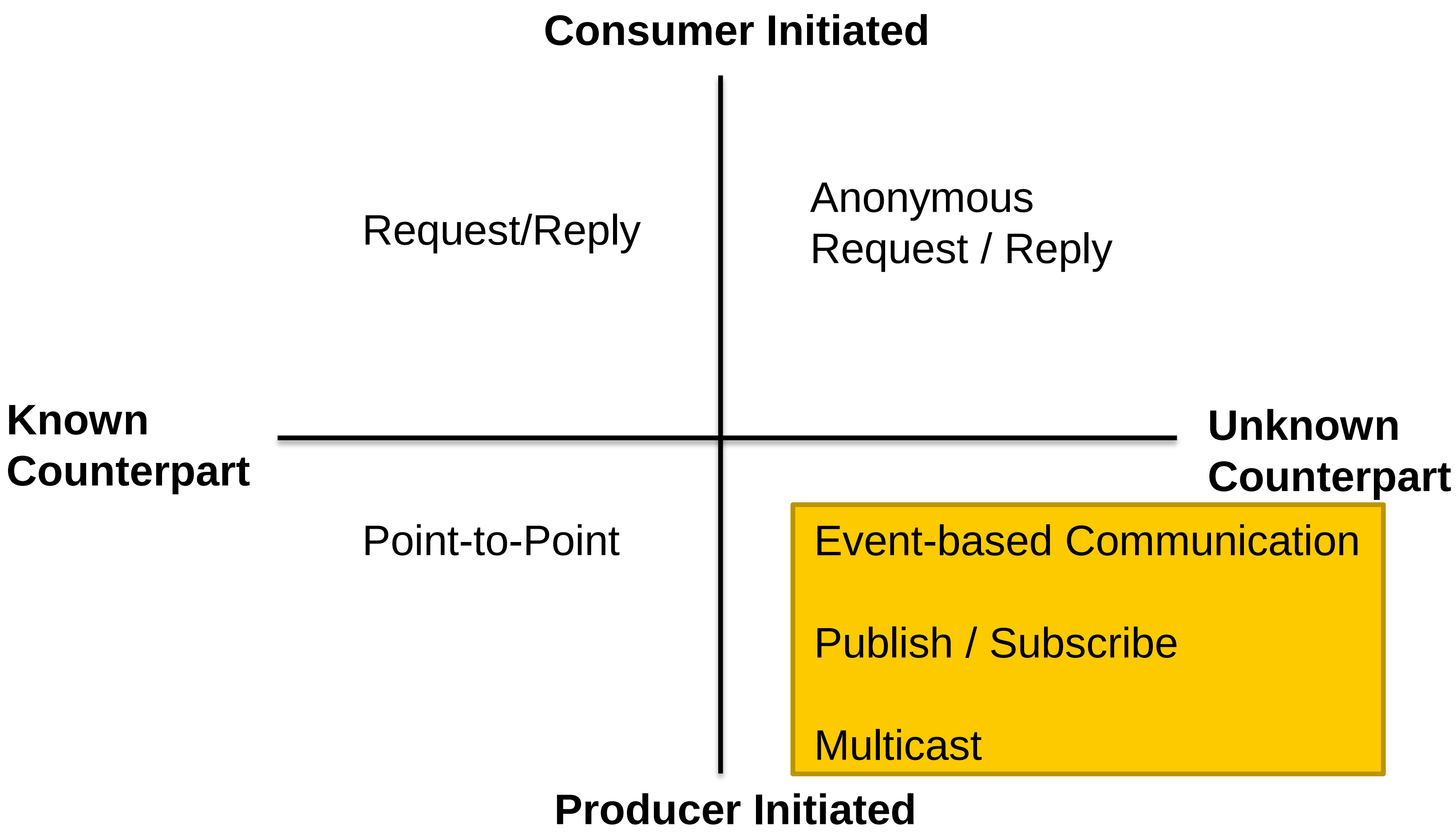


Prof. Dr.-Ing. **Ralf Steinmetz**
KOM - Multimedia Communications Lab

© author(s) of these slides including research results from the KOM research network and TU Darmstadt; otherwise it is specified at the respective slide
Template Teaching v.3.4

Overview

1 Introduction - Basic Types of Interaction



Source: I. Petrov, Lecture: Middleware, 7. Event Based Systems and Publish/Subscribe Notification, WS 2011/12

Event-based Systems becoming more and more popular

- Smart environments producing streams of events
  - RFID tag detection and sensor readings
  - Images from surveillance cameras
  - Vehicle count/identification
  - Position/context data
  - Online social networks Twitter, Facebook

Stream Processing requires new Approach

- Data in database
  - Stationary
  - *Pull* = Query
  - Easy to control

- Streaming data
  - Flowing
  - *Push* = Filter & Aggregate
  - Harder to control

2.1 What is an Event?

- An event is a meaningful change of state (Buchmann et al.)
  - Based on a model of reality
  - Interest (what is meaningful) is determined by the producers and consumers

- Producers can advertise the events they detect
- Consumers subscribe to events of interest

- Connection is made by notification service
- Time is an integral part of the definition of state

- Since time advances, two observations of the same value are two distinct events
  - This makes it easy to deal with status events

Source: I. Petrov, Lecture: Middleware, 7. Event Based Systems and Publish/Subscribe Notification, WS 2011/12

Event and Event Objects

Events have a representation

- Event representation is generally known as an event object

Minimally, an event object has

- An identifier
- A type
- A timestamp
- Optional attributes

Events are typically represented as tuples

- E.g.: Temp, 090221112345, 24.6

As in a database schema, the attributes of an event object carry their specific semantics

Design of event types can have far-reaching effects

Source: I. Petrov, Lecture: Middleware, 7. Event Based Systems and Publish/Subscribe Notification, WS 2011/12

Type of Event: Simple or Complex

- Note: by introducing time as an integral dimension of event characterization, status events and change events are treated uniformly

Simple event:

- any discrete event that is directly detectable and not the result of a composition
- Change event: detected change of state
- Status event: meaningful observation that the state has not changed between distinct detections

Complex event:

- any event that is produced by composing two or more simple events through
  - operators of an event algebra and/or
    enriching an event with external information
- Complex events are all events that
  - were obtained through the application of
    - aggregation, composition or derivation (see next slide)

Source: I. Petrov, Lecture: Middleware, 7. Event Based Systems and Publish/Subscribe Notification, WS 2011/12

Types of Compositions

Aggregation

- the application of relational aggregation operators,
- such as max, min, sum, count or avg

Composition

- combination of two or more events (simple or complex) through the use of an operator of the event algebra.
- Participating events can be heterogeneous

Derivation

- derived events are typically events of higher level of abstraction.
- In the derivation process we apply domain semantics to define new events based on observed events.
  - Example:
    - 5 simple temperature-reading events taken every 5 minutes and
    - showing that the temperature is monotonically increasing lead to an air-conditioner-failed event
    - Assumes:
      - Temperature readings are ambient temperature readings,
      - uses domain knowledge about location of sensor(s),
      - fact that AC is on, etc.

Source: I. Petrov, Lecture: Middleware, 7. Event Based Systems and Publish/Subscribe Notification, WS 2011/12

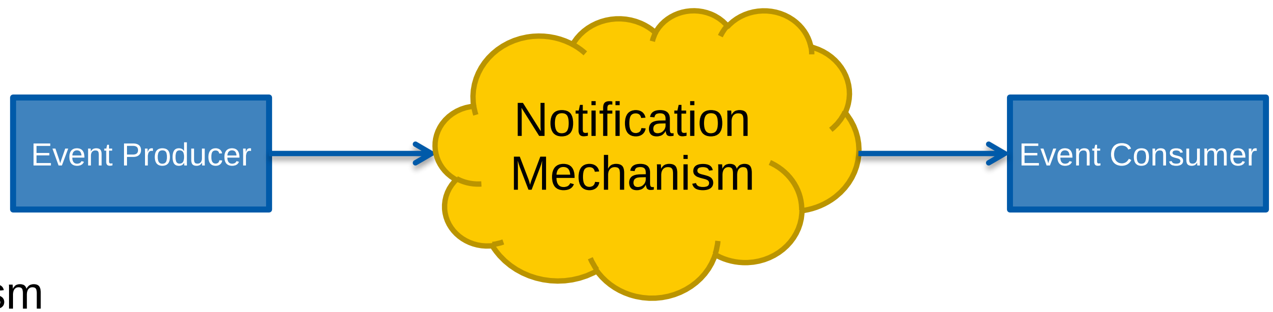
2.3 Event Driven Architecture

Goals:

- Provide agility
- Provide flexibility
- Support celerity

Components:

- Event Producer
- Notification Mechanism
- Event Consumer



Properties

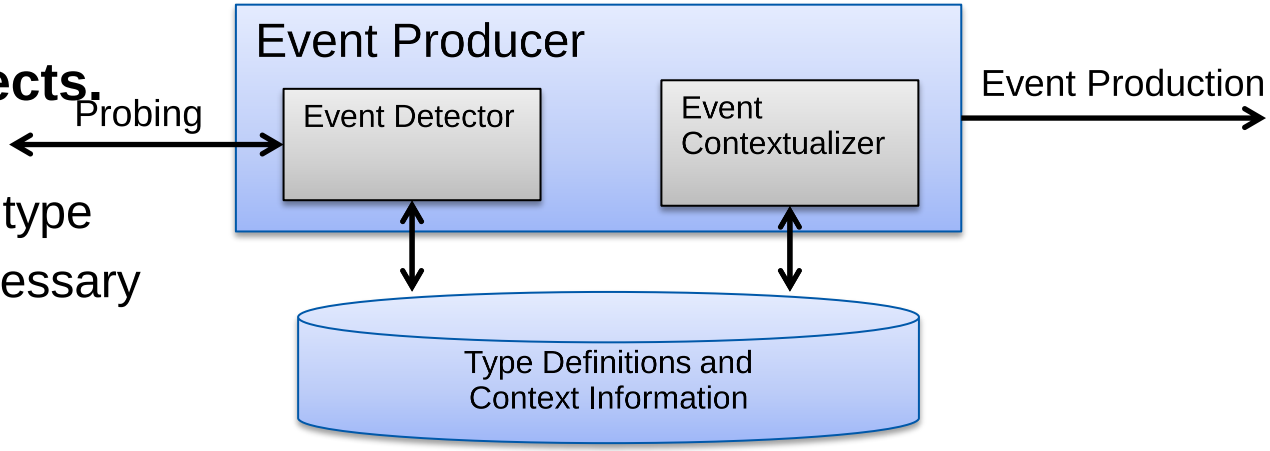
- Event producers need not be aware of who will eventually consume the events
- Event producers and consumers should be decoupled in space and time
- Event consumers are notified as soon as possible about events of interest
- Reporting of current events as they happen
  - No store and forward
- Pushing notifications of events from the producer to the consumer
  - Events are packaged as notifications and delivered to consumer
- Responding immediately to recognized events

Source: I. Petrov, Lecture: Middleware, 7. Event Based Systems and Publish/Subscribe Notification, WS 2011/12

Event Producers

Detect events and produce event objects.

- Their structure is defined by the event type
- and contains the necessary event parameters



Event parameters are instantiated by

- the event detection process and
- the event contextualization process.

The event detection process

- typically will probe the environment.

The event contextualization process

- may rely on external data sources,
- type and context information may be held locally.

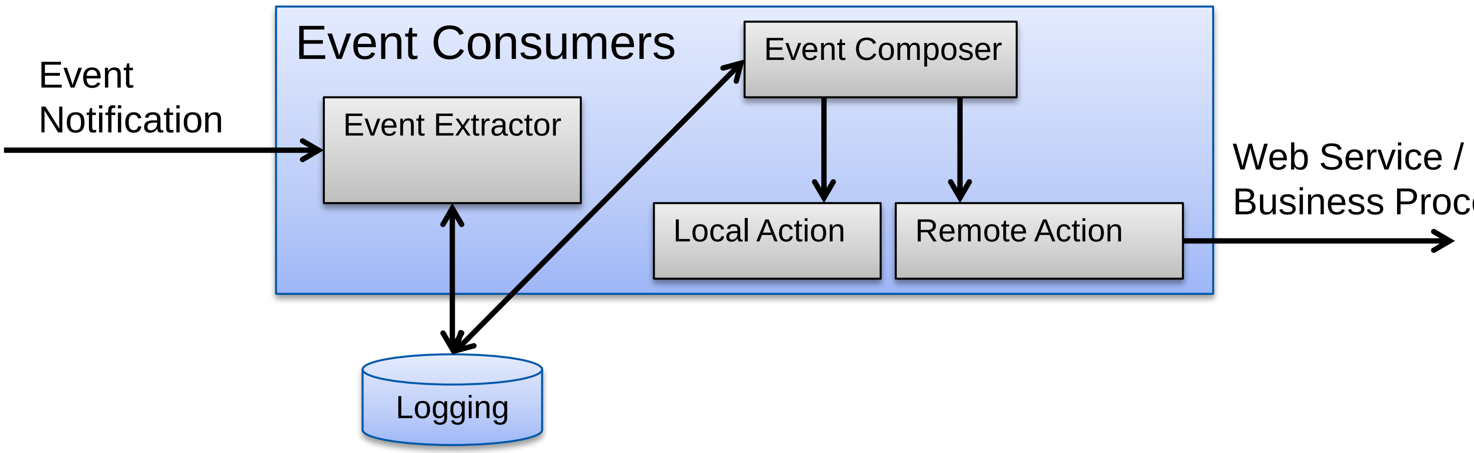
Source: I. Petrov, Lecture: Middleware, 7. Event Based Systems and Publish/Subscribe Notification, WS 2011/12

Event Consumers

- Receive event notifications from the notification mechanism/service.
- Unpack the event notification,
- extract the event object and
- execute an action in response to the received event.

The response may be a

- local action,
- the invocation of a (remote) service or business process,
- an event composition or
- storage of the event for logging.

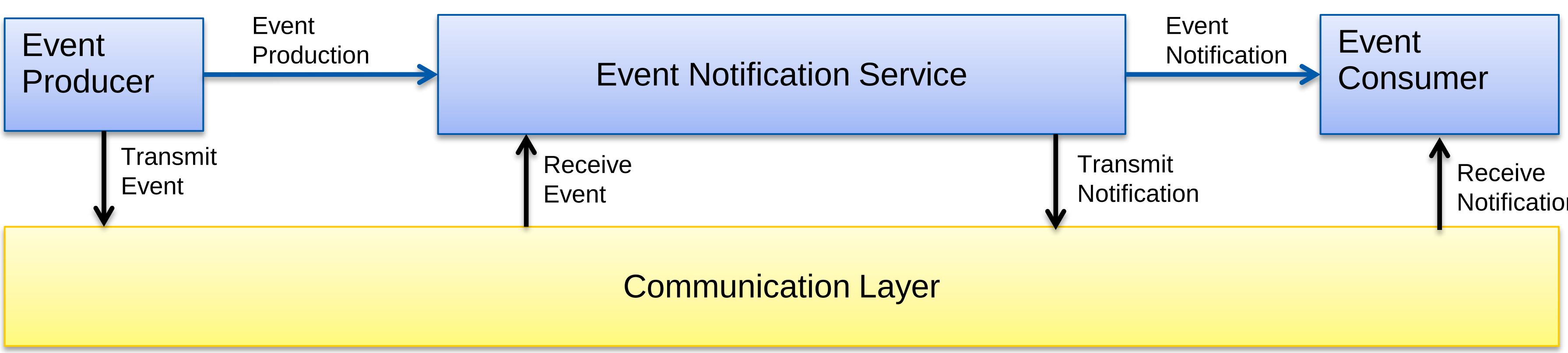


Source: I. Petrov, Lecture: Middleware, 7. Event Based Systems and Publish/Subscribe Notification, WS 2011/12

Event Notification Service

Key questions:

- How are producers and consumers brought together?
- Does the channel deliver all messages or does it filter?
- If filtering is done, on what criteria and where are the filters placed?
- Are events only routed by the notification mechanism or are they transformed?
- If transformations are applied, where are they applied and what are they?



Source: I. Petrov, Lecture: Middleware, 7. Event Based Systems and Publish/Subscribe Notification, WS 2011/12

What's "wrong" with RPC and other similar mechanisms?

They are based on Request/Reply approach:

- Client
  - has initiative, client "pulls" data from server
- Peer
  - "locked" in handshake
- Bad if:
  - Lots of data
  - many receivers which come and go
  - information not generated very often

Push approach

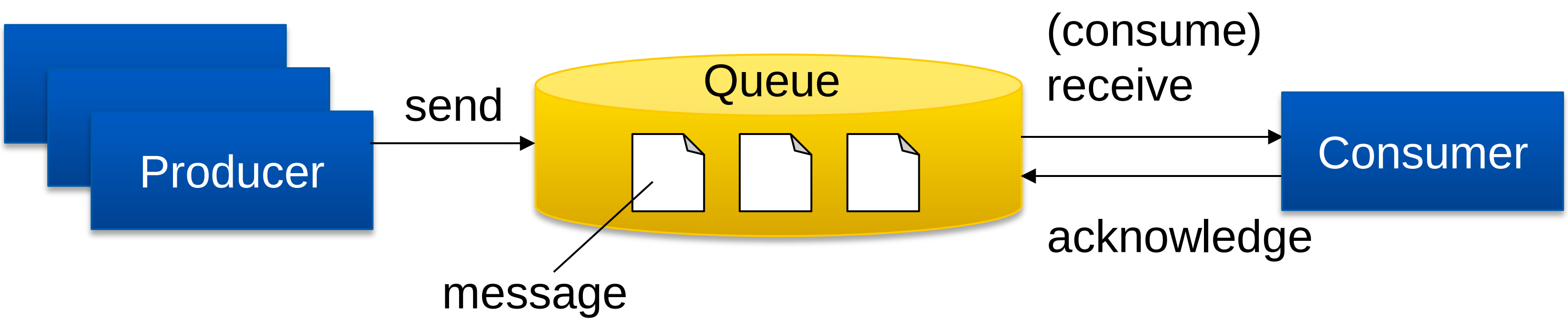
- Idea:
  - Peer with data "pushes" it to interested parties
- Producer:
  - Immediate information delivery ("publish")
- Consumer:
  - Initial "subscribe" for event-type/channel (= info-category)
- Consumer receives events
  - that match the subscription asynchronously as they are generated by producer

Such systems are also called publish/subscribe

Pub / Sub systems are widely used:

- Twitter
- Facebook
- Google Alerts

Messaging Domain: Point-to-Point



Each message has only one consumer

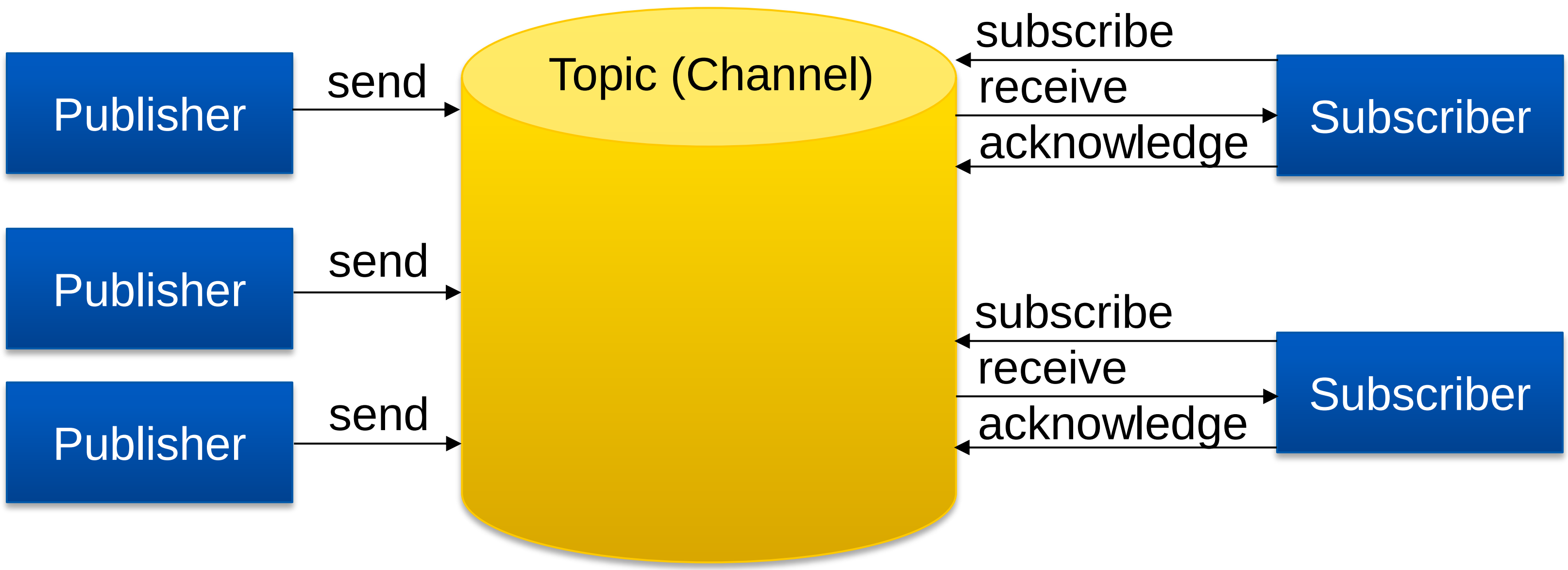
Receiver acknowledges successful processing of message

No timing dependencies between sender and receiver

Queue stores message (persistent), until

- It is read by a receiver
- The message expires (Leases)

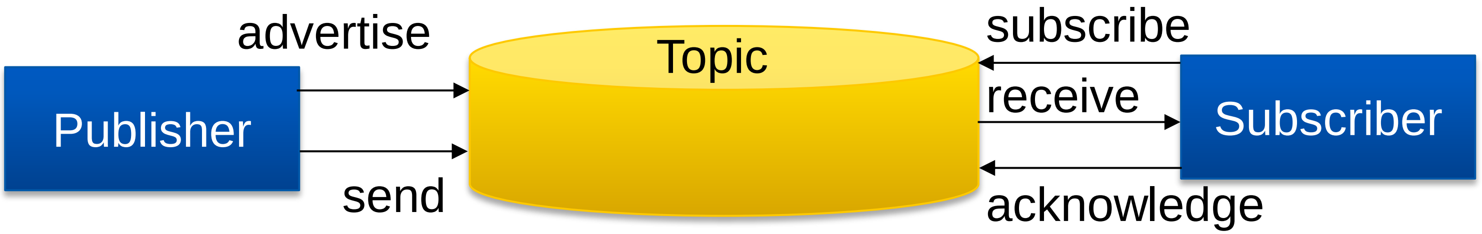
Messaging Domain: Publish/Subscribe



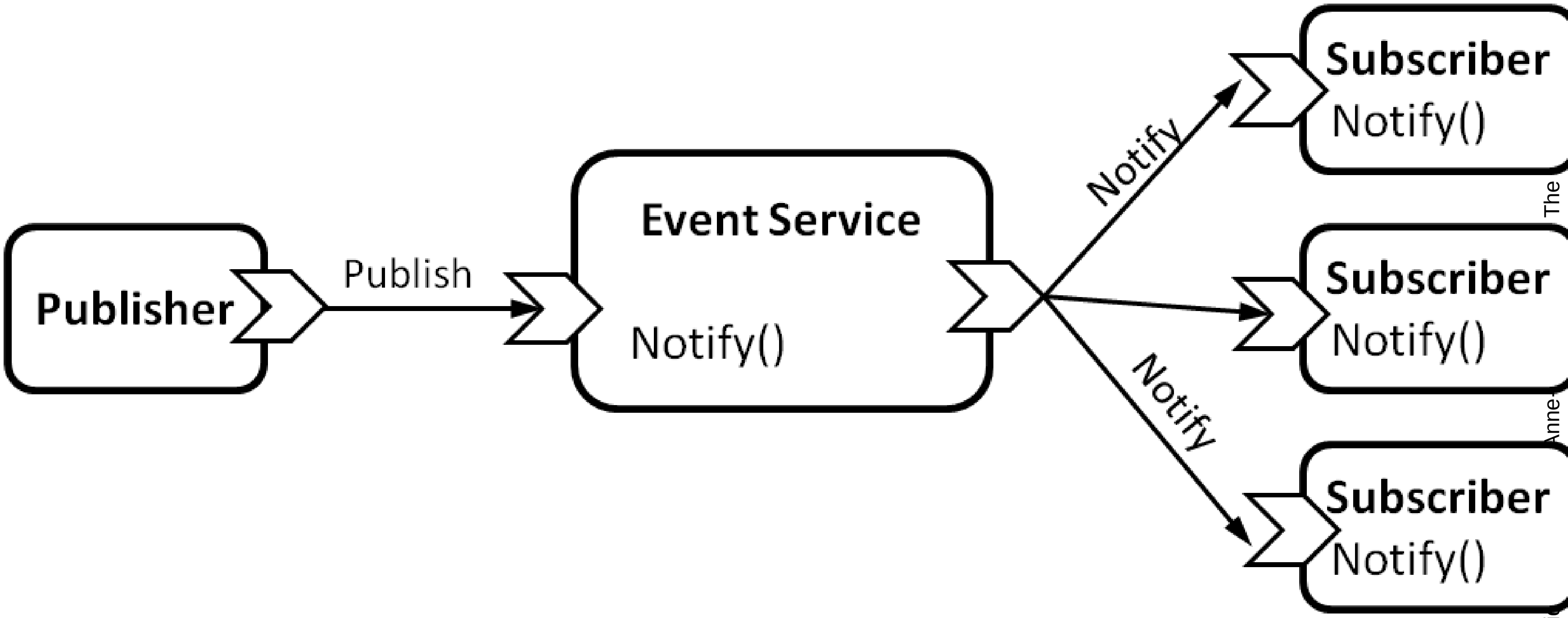
- Interested parties can subscribe to a channel (topic)
- Applications post messages explicitly to specific channels
- Each message may have multiple receivers
- Timing dependency between publishers and subscribers

Messaging Domain: Advertisements

- Publisher advertises topic before publishing
- Subscribers can get a list of advertised topics
- Avoids problem of subscriber having to figure out which topics are available for subscription



Basic Principle Space Decoupling

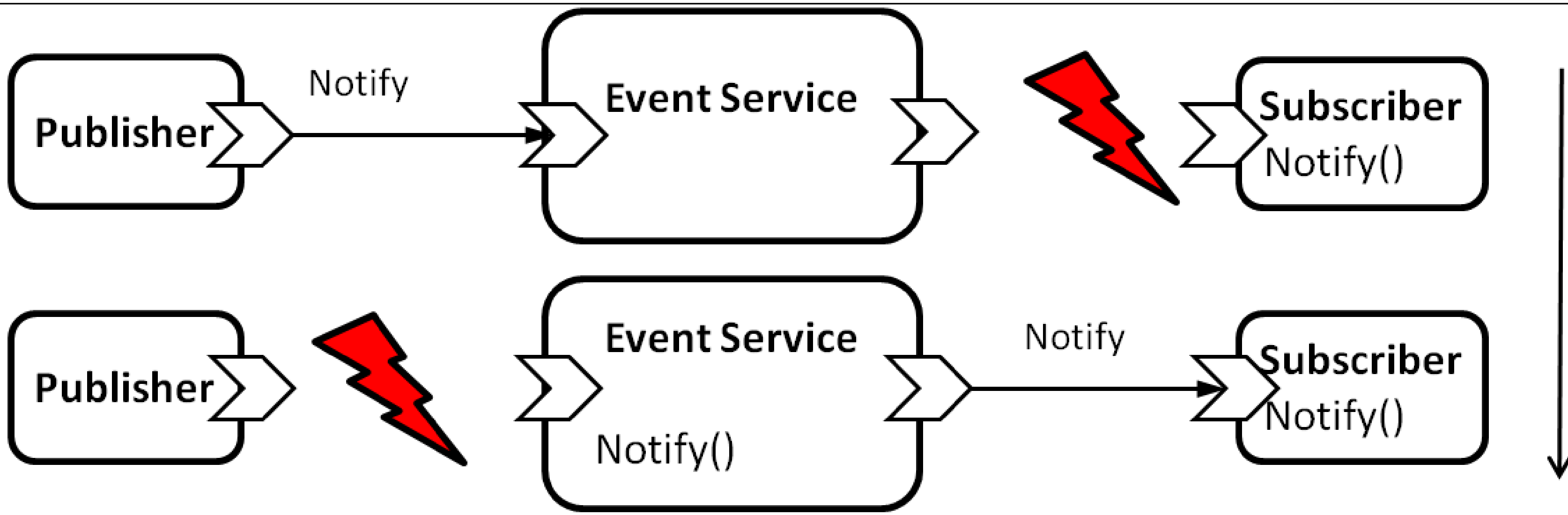


Space decoupling

- The interacting parties do not need to know each other
- Producers publish messages through an event service
- Subscribers indirectly receive messages from event service
- One-to-many communication patterns possible

Source: Patrick Eugster, Pascal Felber, Rachi… Anne… The
Publish/Subscribe. ACM Computing Surveys, Number: 2, Volume 35, June 2003, pages 114-131

Basic Principles Time Decoupling



Time decoupling

- The interacting parties do not need to be actively participating in the interaction at the same time

Reducing space and time dependencies greatly reduces the need for coordination between systems

Source: Patrick Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec: The Many Faces of Publish/Subscribe. ACM Computing Surveys, Number. 2, Volume 35, June 2003, pages 114-131

Publish/Subscribe vs. Event-based Systems

Commonalities:

- Both decouple sender and receiver from space and time

Differences:

- Events vs. messages

- Event-based systems are a class of systems
- Pub-Sub is a paradigm

- Event-based systems process events
  by aggregating, filtering, and composing events
  → derive complex events
- Pub/Sub just delivers messages to subscribers

- In Event-based systems the consumer does not need to register necessarily
- In the Pub / Sub paradigm a subscription is necessary

Network Multicast

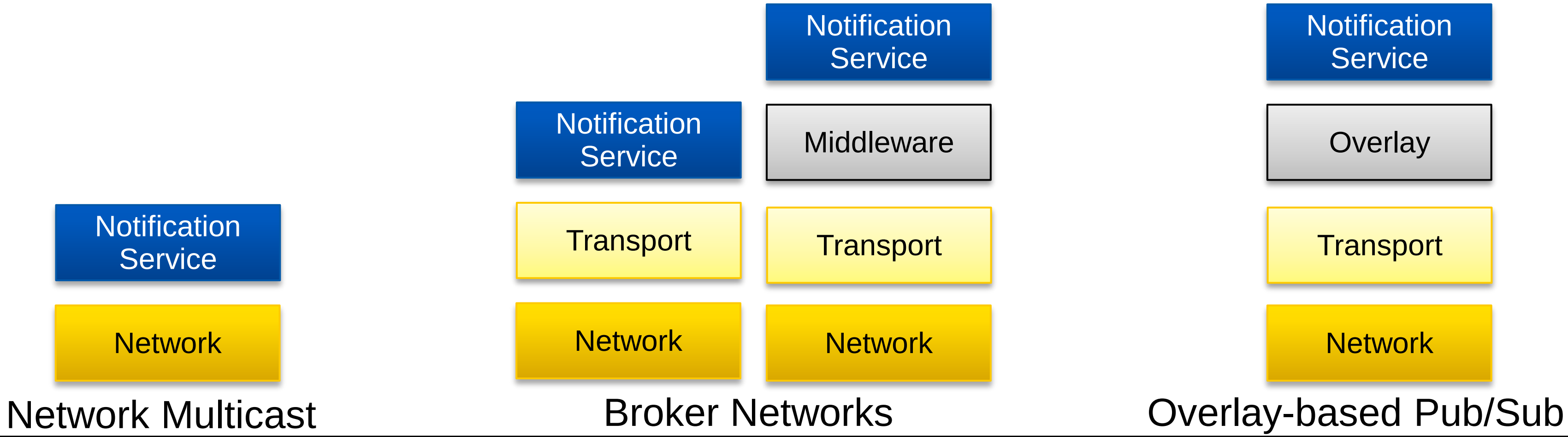
- Use of multicast networking facilities (also at data link level)

Broker Networks

- Based on transport level connections between nodes
- Hierarchical (Decision tree from publisher to subscribers)
- Undirected Acyclic graph spanning all brokers

(Structured) Overlay

- DHT (abstracting from physical nodes)



Subscription Model

- Topic-based / Channel-based
- Content-based
- Type-based

Routing

- Filter-based
- Rendezvous-based

Topology

- Centralized
- Decentralized
  - Broker-based
  - DHT-based
  - Rendezvous-based

3.3 Subscription Model

Topic-based subscription

- Messages sent to a well-known topic
- Recipients are known a-priori
- Subscribers subscribe to topics
- Topics are typically expressed as strings
- Limited expressiveness
- Many efficient implementations exist

Content-based subscription

- Subscriptions are matched against the content of the message
- Subscribers describe their interest as filter expressions
- Cannot determine recipients before publication
- More flexible / expressive / general
- Difficult to implement efficiently

Special case: Subject-based subscription

- Special case of content-based subscription
- Well-known subject in messages
- Subscriptions matched against the subject
- Subject typically strings or key-value-pairs

Subscription Semantics

Covering Relations

- Attribute Filter: Filter covers Notification (= message)

$$\phi \subset_f^n \alpha :\Leftrightarrow \phi \quad \text{covers} \quad \alpha$$

$$\phi \subset_f^n \alpha :\Leftrightarrow \phi.name = \alpha.name \wedge \phi.type = \alpha.type \wedge \phi.match(\alpha.value, \phi.value)$$

- Subscription: Subscription covers Notification

$$s \subset_S^N n :\Leftrightarrow s \text{ covers } n$$

$$N_S(s) \subseteq N;\; n \in N_S(s) :\Leftrightarrow s \subset_S^N n$$

$$N_S(s) = \{n \in N : \forall \phi \in s : \exists \alpha \in n : \phi \subset_f^n \alpha\}$$

- Examples:

String event=alarm	$\subset_S^N$	String event=alarm
		Time date=02:40:03
String event=alarm	$\not\subset_S^N$	String event=alarm
Integer level>3		Time date=02:40:03

Covering Relations

- Advertisement: Advertisement covers Notification

$$a \subset_A^N n :\Leftrightarrow a \text{ covers } n$$

$$N_A(s) \subseteq N; \quad n \in N_A(a) :\Leftrightarrow a \subset_A^N n$$

$$N_A(a) = \{n \in N : (\forall \alpha \in n : \exists \phi \in a : \phi.name = \alpha.name)$$

$$\wedge (\forall \alpha \in n : \forall \phi \in a : \phi.name = \alpha.name \Rightarrow \phi \subset_f^n \alpha)\}$$

- Advertisement covers Subscription

$$a \subset_A^S s :\Leftrightarrow N_A(a) \cap N_S(s) \neq \emptyset$$

- “a is relevant for s”

Covering Relations: Examples

String event=alarm Time date any Integer level>0	$\subset_A^S$	String event=alarm Integer level>3
String event=alarm Time date any Integer level>0	$\not\subset_A^S$	String event=alarm Integer level>3 String user any
String event=alarm Time date any Integer level>0	$\subset_A^S$	Integer level>5

Subscription-based event service

- Service delivers notification n to party X iff
  - X subscribes s
  - $s \subset_S^N n$

Advertisement-based event service

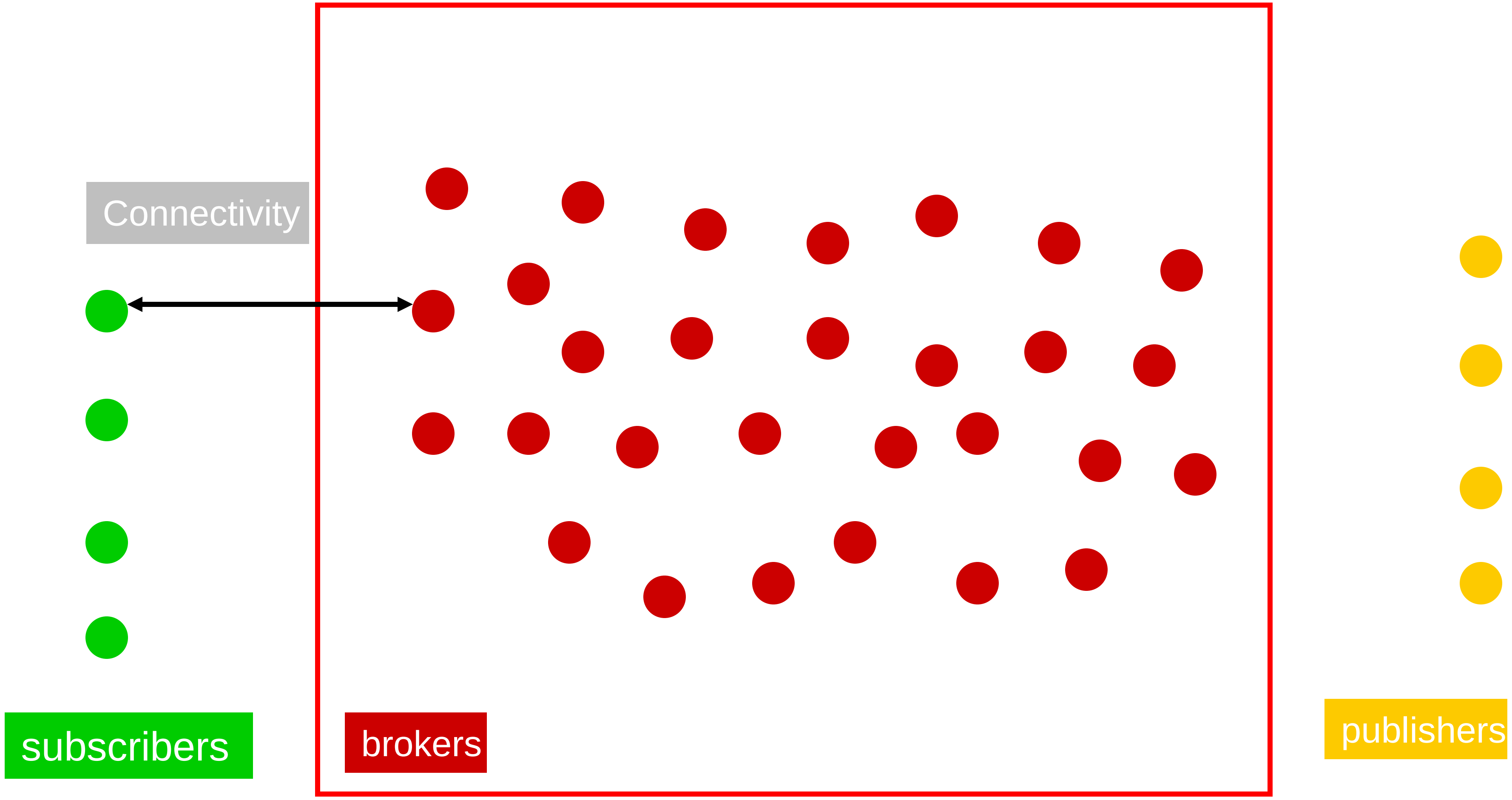
- Service delivers notification n posted by object Y to party X iff
  - Y advertises a
  - X subscribes s
  - $a \subset_A^S s$
  - $s \subset_S^N n$

Two extreme solutions

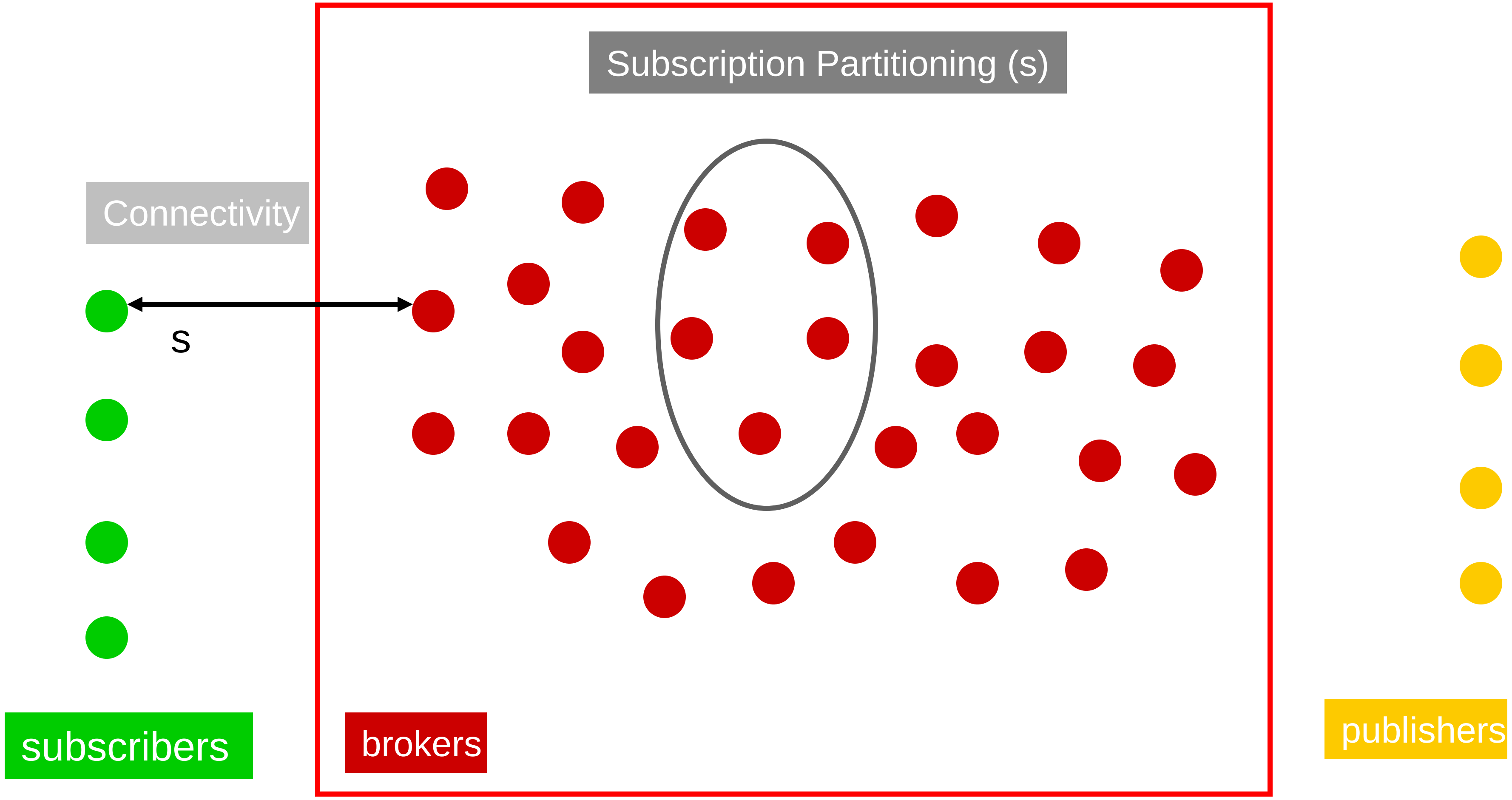
- Event flooding
  - flooding of events in the notification event box
  - each subscription stored only in one place within the notification component
  - matching operations equal to the number of brokers

- Subscription flooding:
  - each subscription stored at any place within the notification component
  - each event matched directly at the broker where the event enters the notification event box

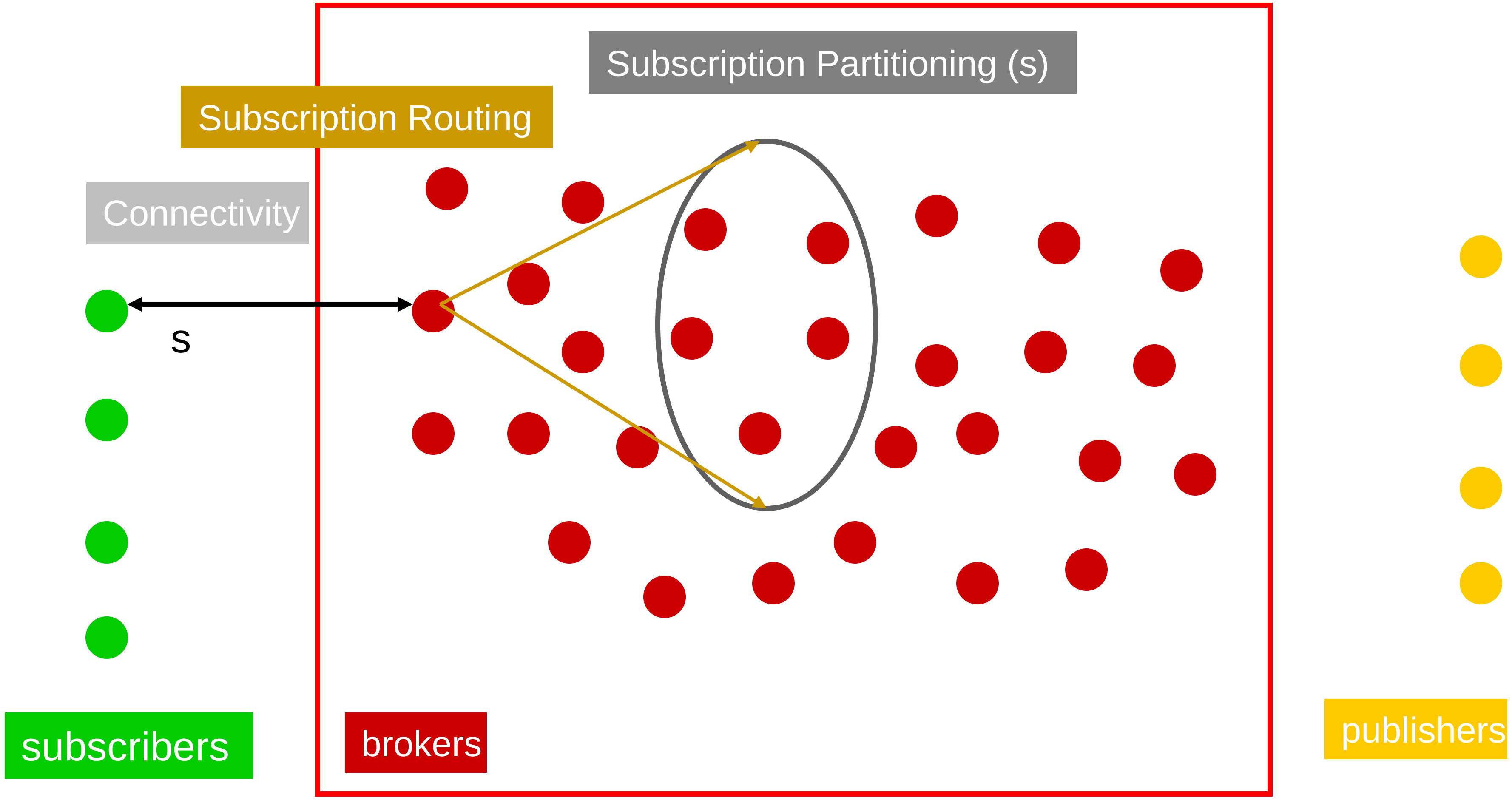
Step 1: Handling Subscriptions



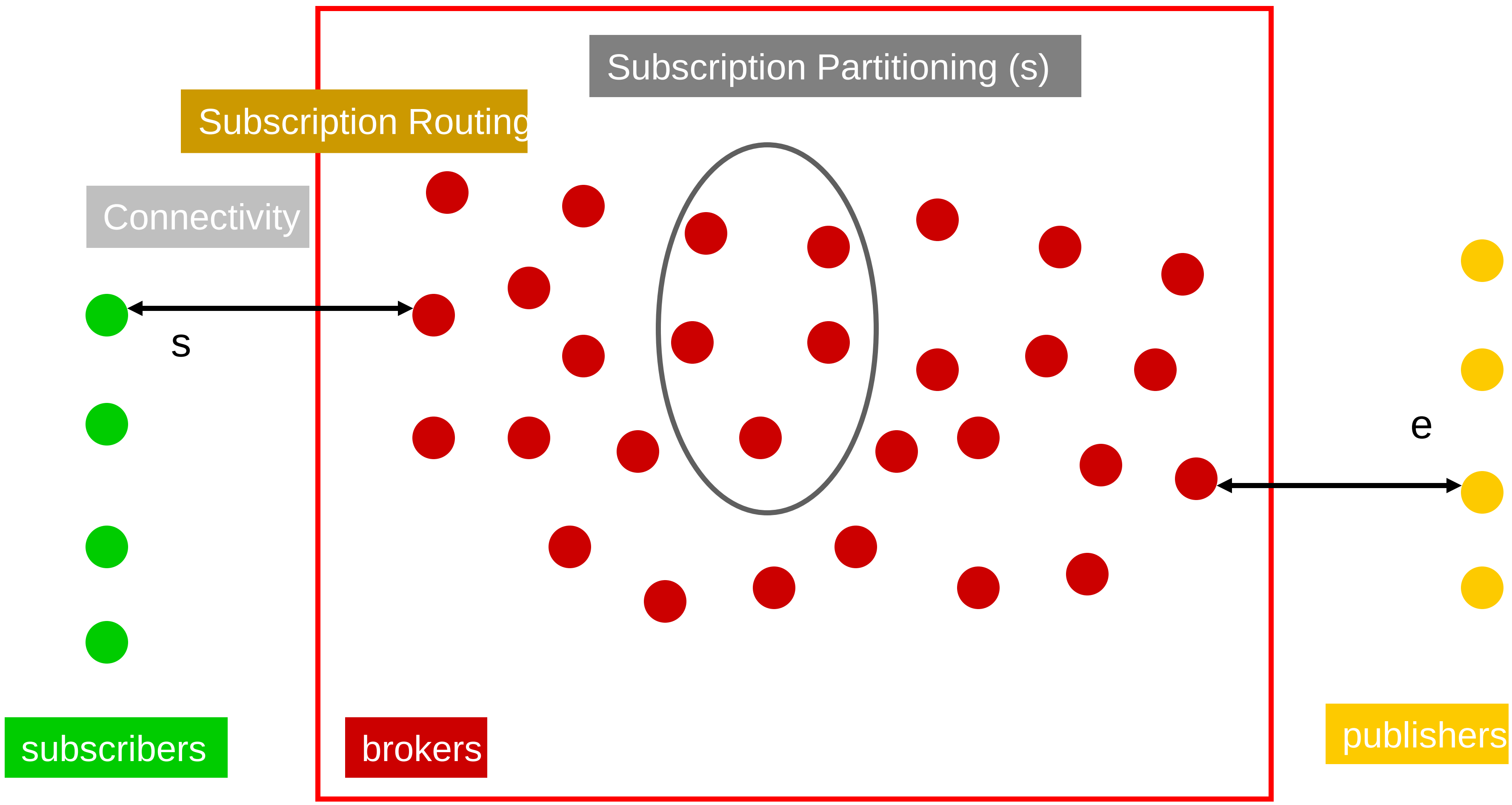
Step 2: Subscription Partitioning: Where to store *s*?



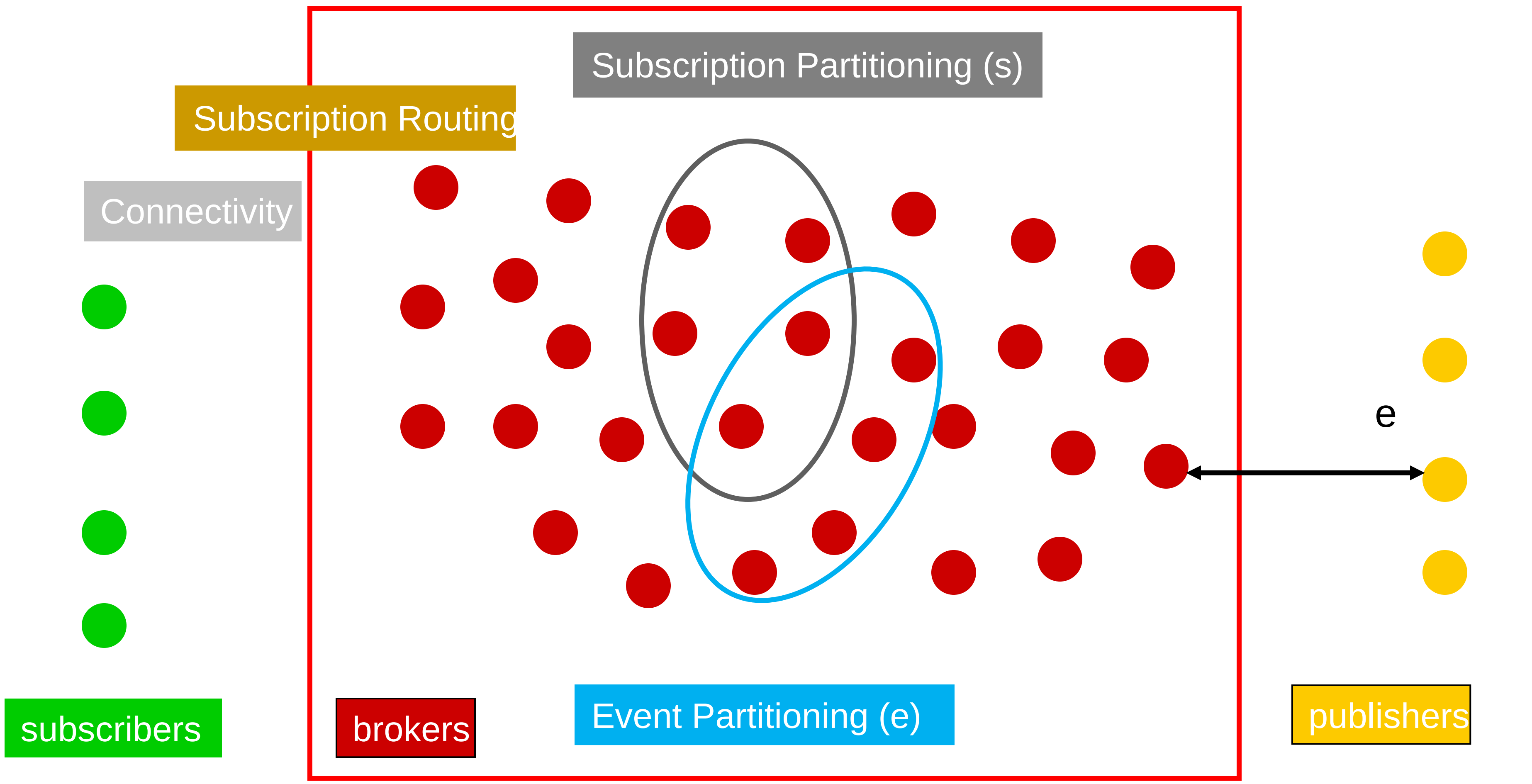
Step 3: Subscription Routing: How to bring *s* there?



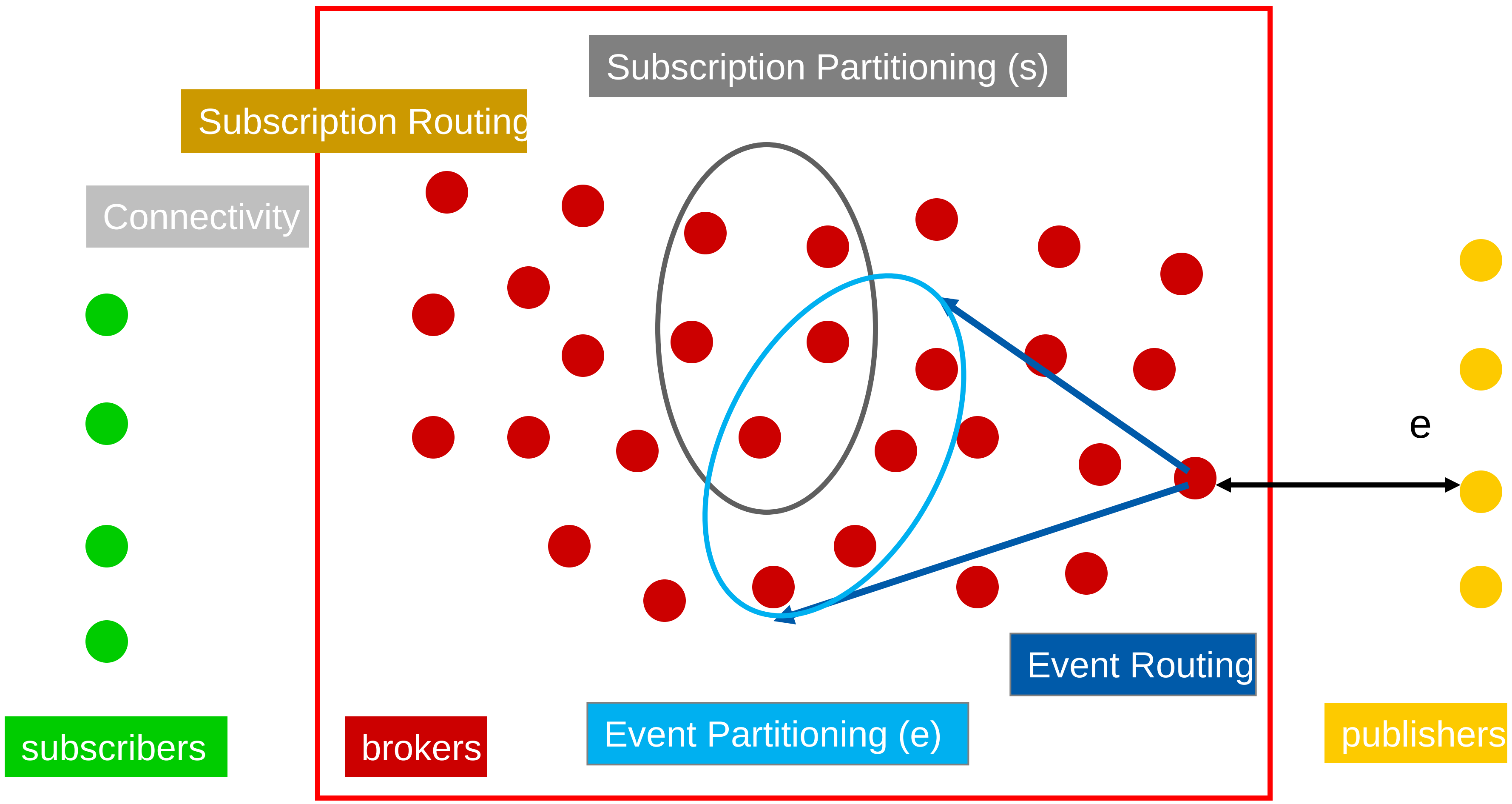
Step 4: Handling Events



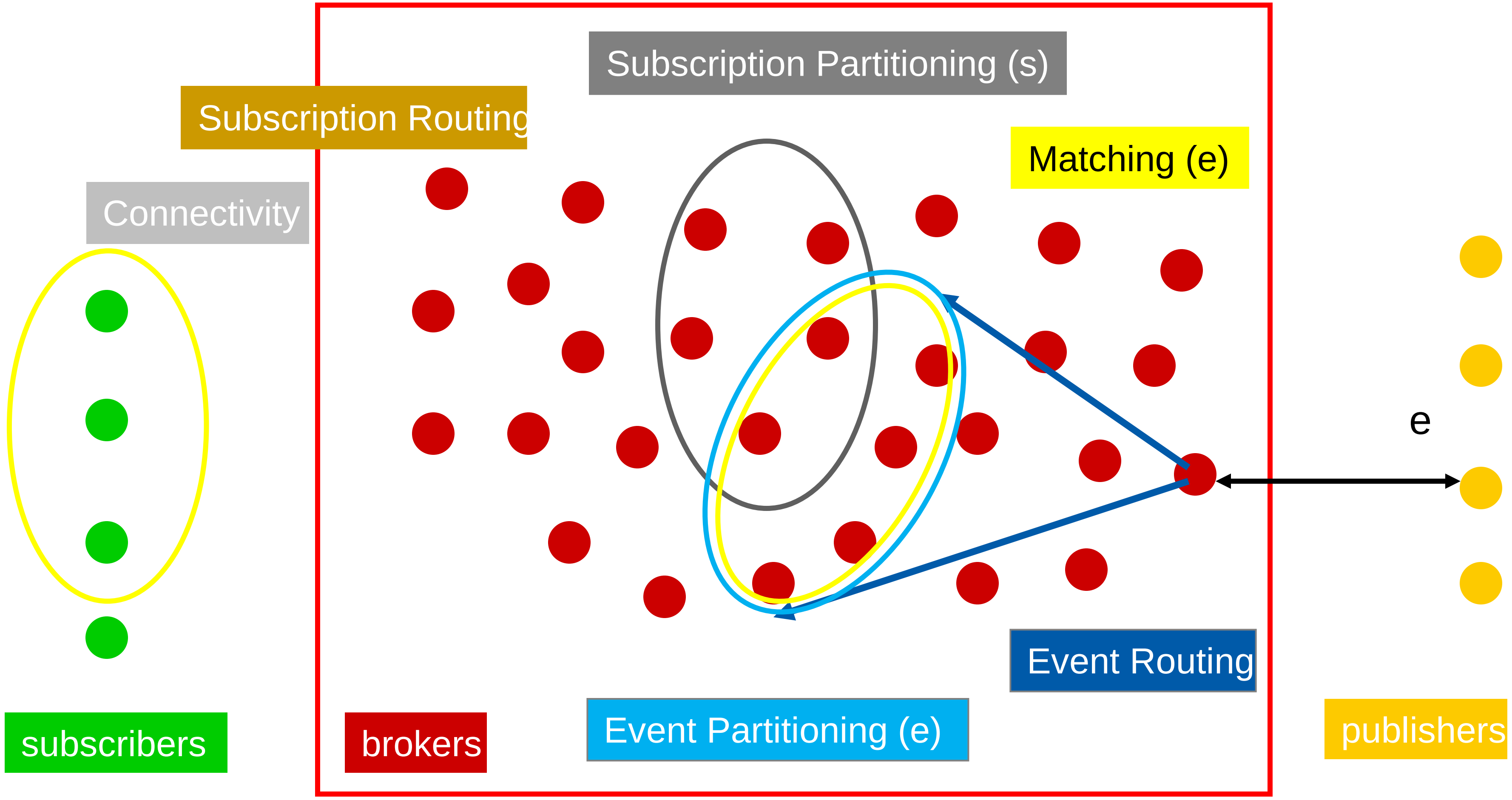
Step 5: Event Partitioning: Where to match *e?*



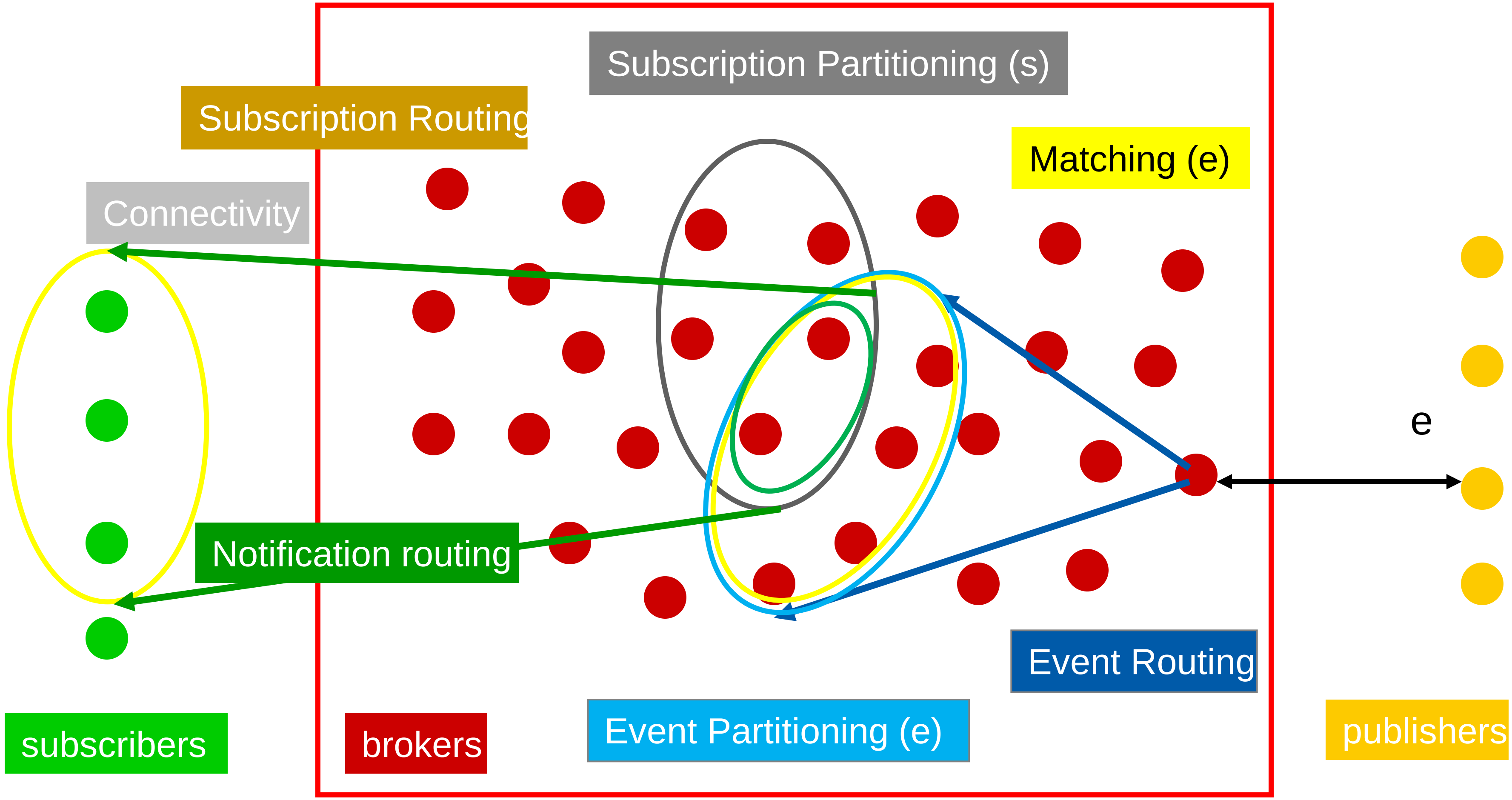
Step 6: Event Routing: How to bring *e* there?



Step 7: Event Matching: Define the set of recipients of *e*



Step 8: Notification Routing: Bring *e* to the subscribers?



3.5 Filtering-based Routing

Identifying as soon as possible events that are not interesting for any subscriber and arrest their diffusion

Creating «diffusion paths» that lead to subscribers for the event

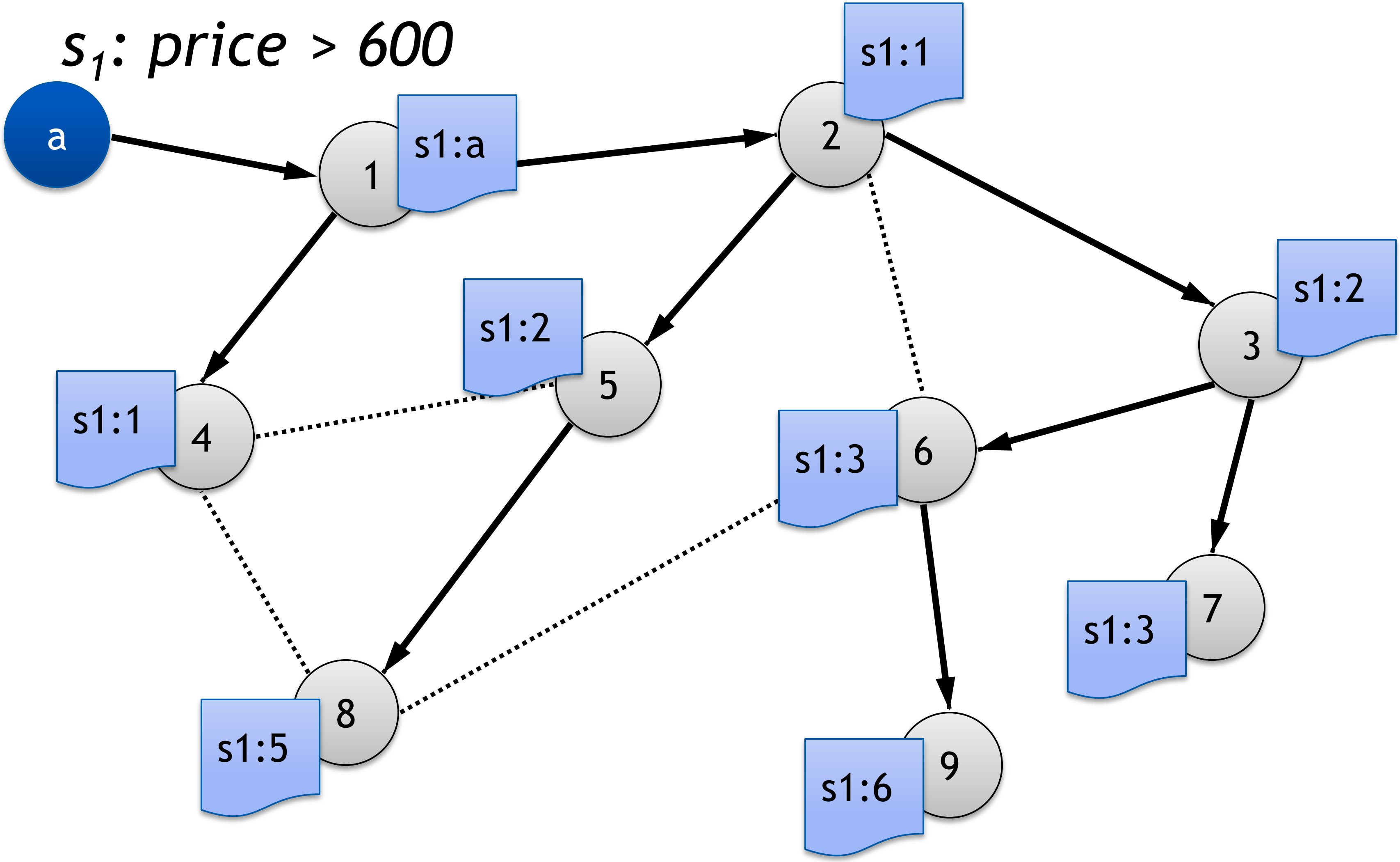
Construction of a diffusion path requires routing info to be maintained at brokers

Routing information consists of a set of filters (aggregate of subscriptions) that are reachable through that broker

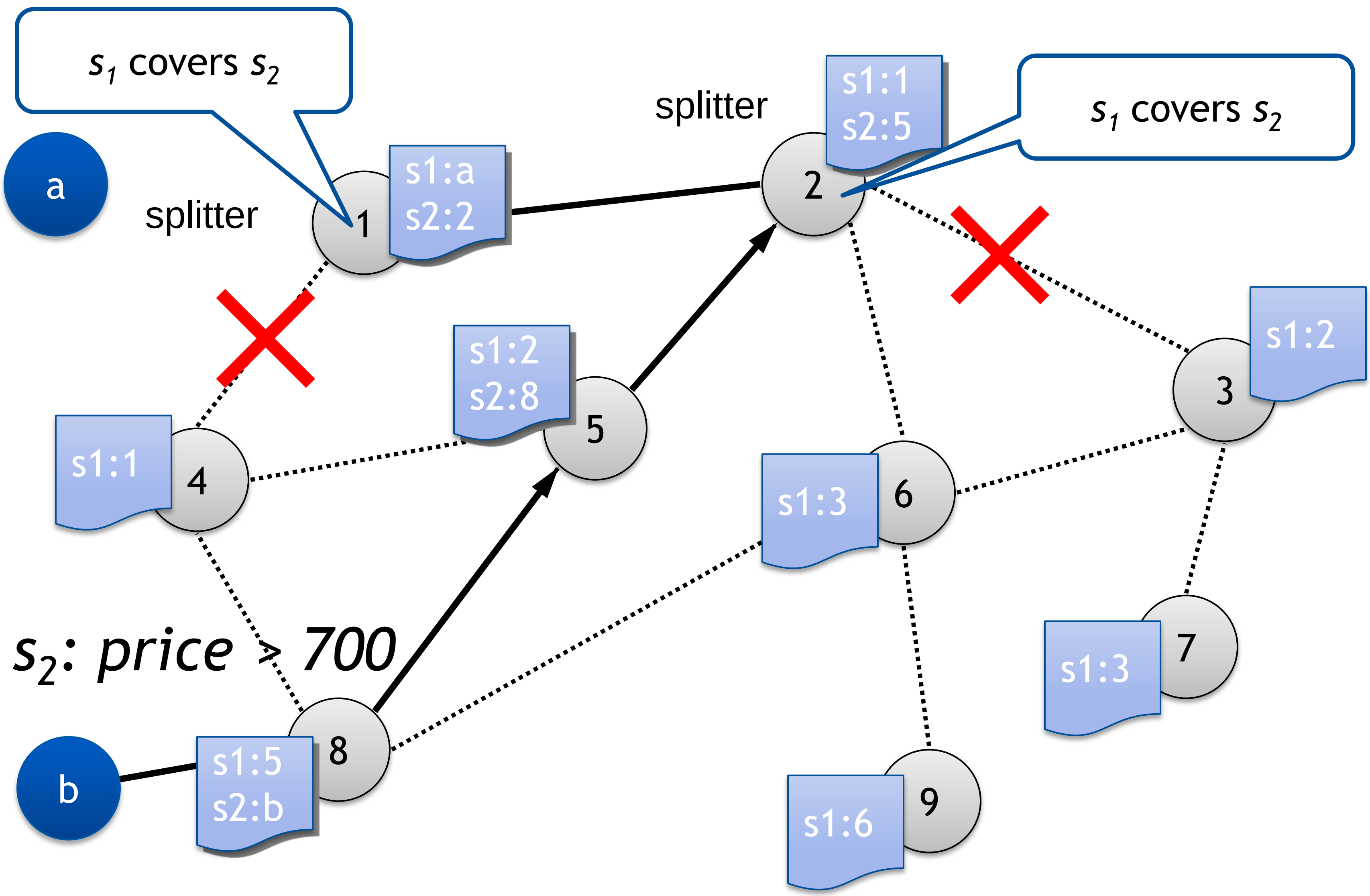
Three main phases:

- Subscription forwarding: subscriptions (filters) are propagated to every broker leaving state along the path
- Matching notification follow (backwards) the path set by subscriptions
- Reducing subscription propagation
  - Avoiding subscription propagation when a filter including the subscription has been already forwarded

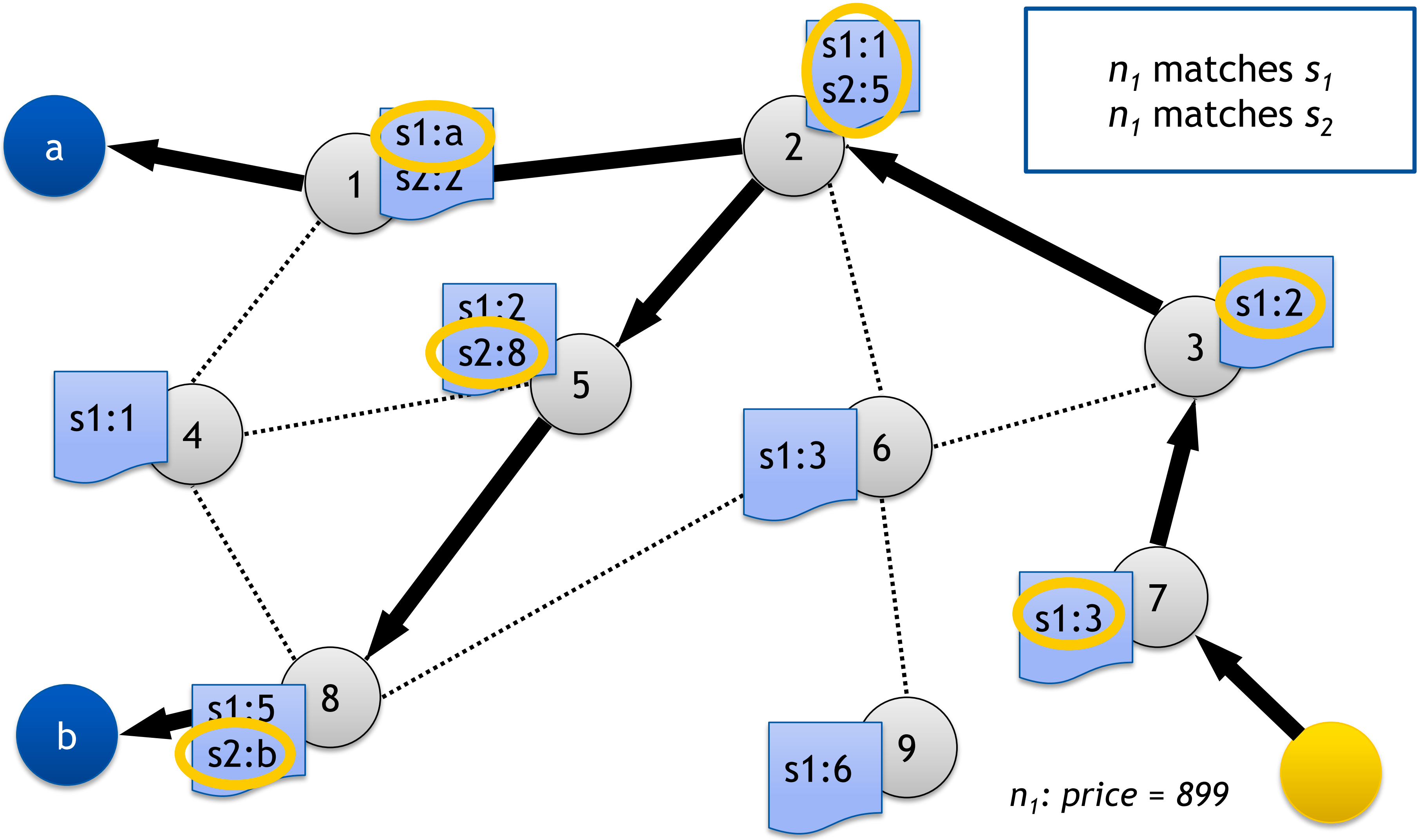
SIENA Filtering-Based Routing
Subscription Forwarding



SIENA Filtering-Based Routing
Subscription Merging



SIENA Filtering-Based Routing
Notification Delivery



Filtering based routing

Tradeoff between subscription redundancy and subscription flooding

The more the brokers aware of any subscription the earlier notifications that do not match any subscription are filtered out

Tradeoff between expressiveness of subscription language and scalability

- Expressiveness complexity brings to flood filters
- Limited complexity better scalability

Each event of the event space is owned by one or more (reliability reasons) brokers

Three functions:

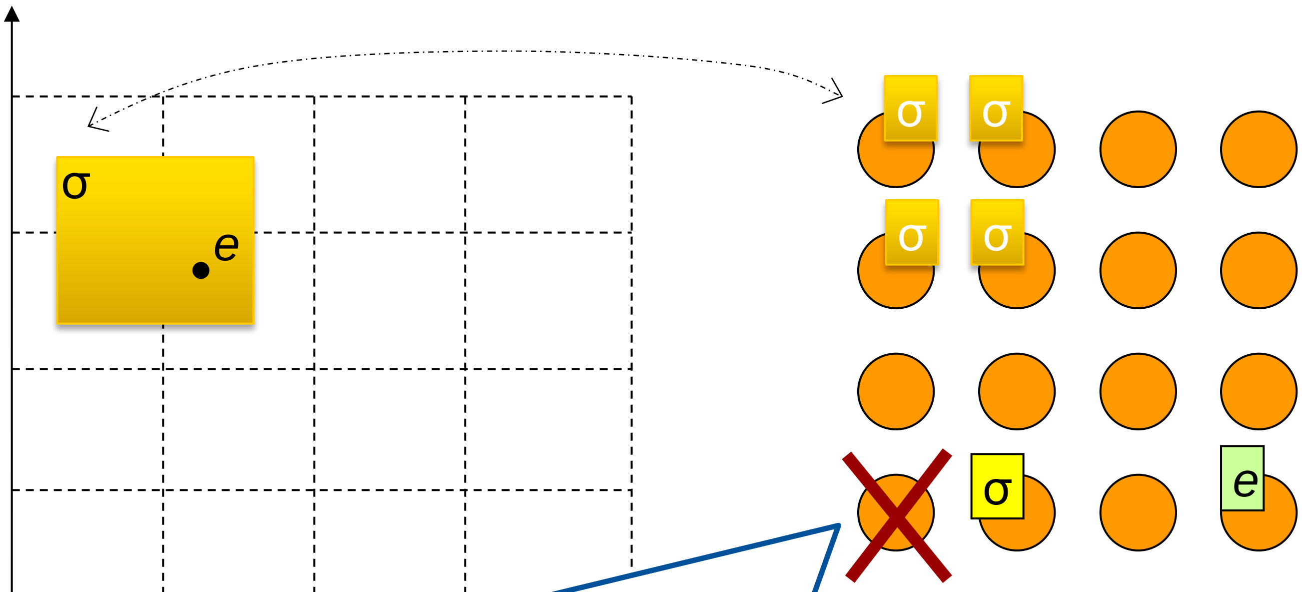
- SN(σ): $\Sigma \rightarrow 2^N$ (rendez-vous nodes of σ)
- EN(e): $\Omega \rightarrow 2^N$ (rendez-vous nodes of e)
- **If** $e \in$ SN(σ) **then** SN(σ) $\cap$ EN(e) $\neq \varnothing$ (intersection rule)

Main characteristics:

- Controlled subscriptions distribution
- Controlled matching
- Better load balance

Each node is responsible for a partition of the event space

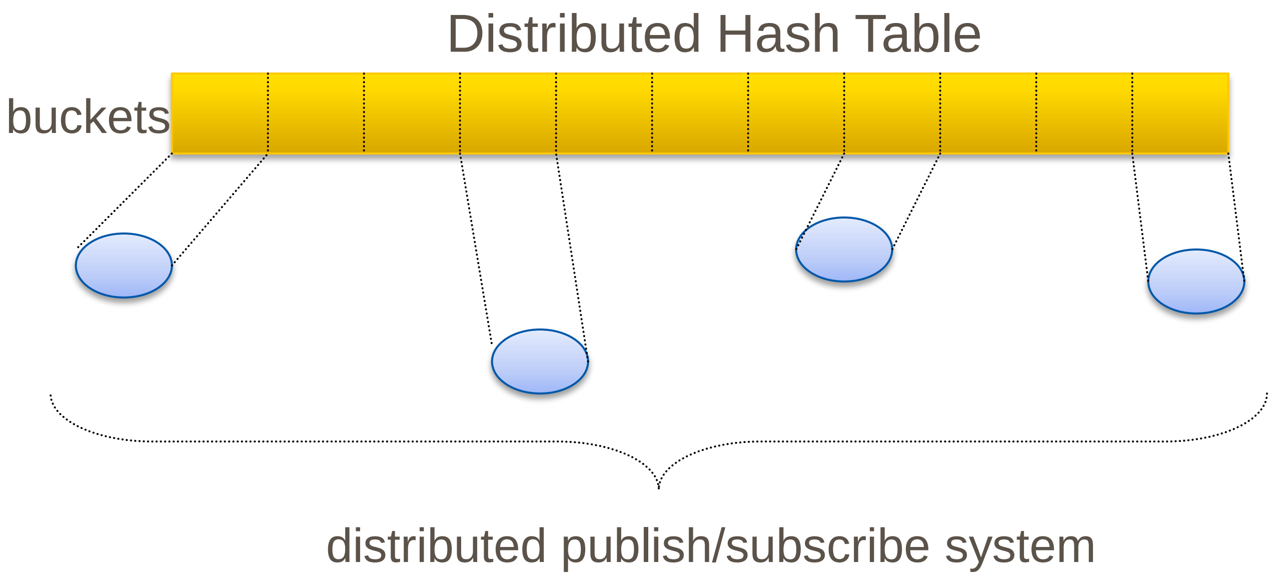
- Storing subscriptions, matching events



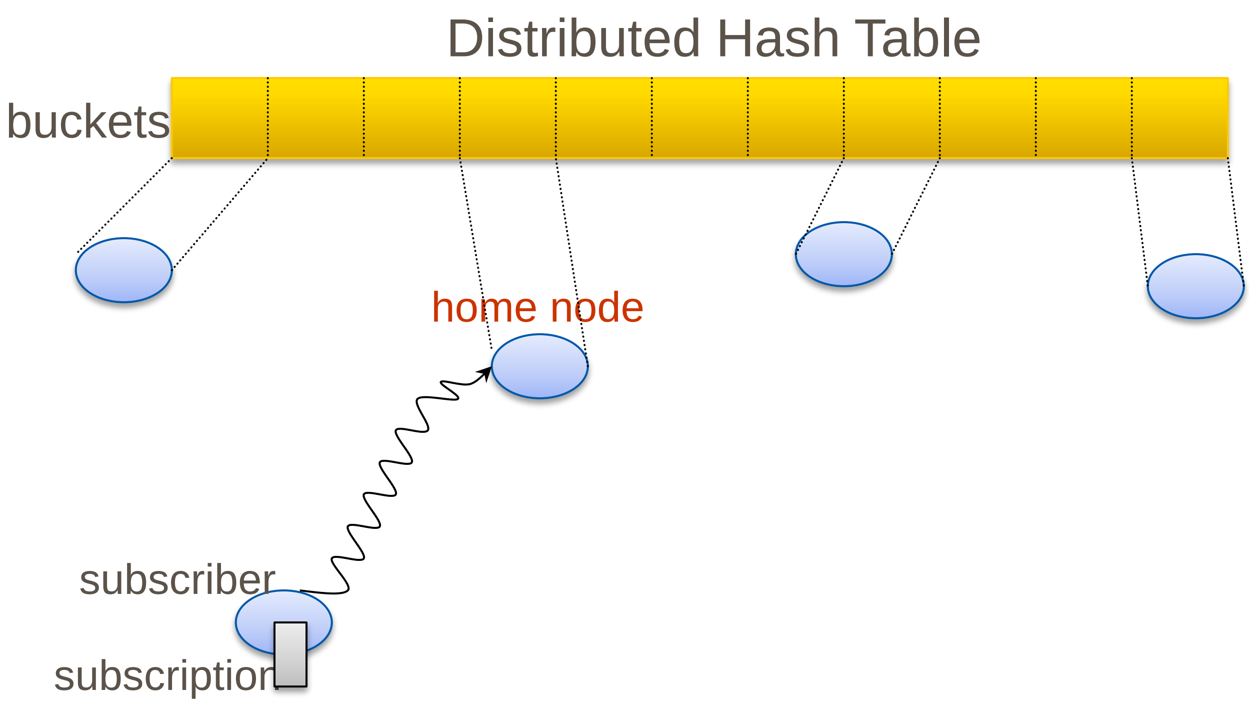
Problem: difficult to define mapping functions when the set of nodes changes over time

Rendez-Vous Routing based on a DHT

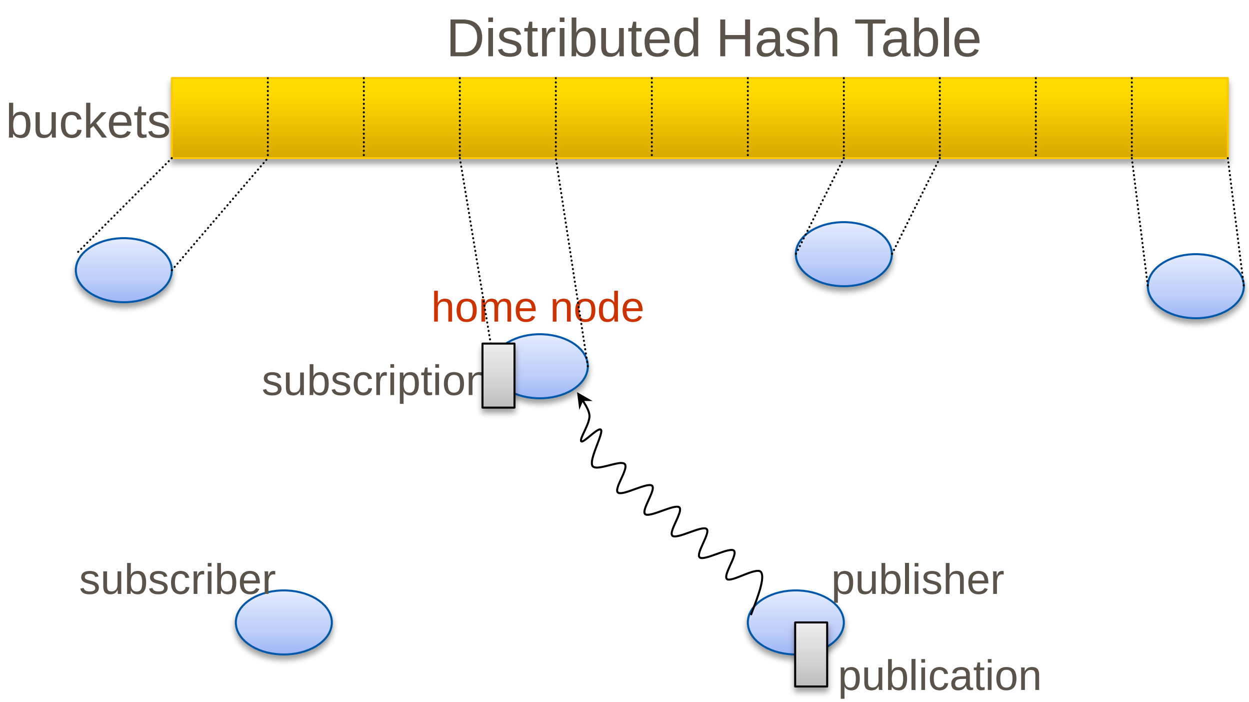
- **A matching publisher & subscriber must come up with the same hash keys based on the content**



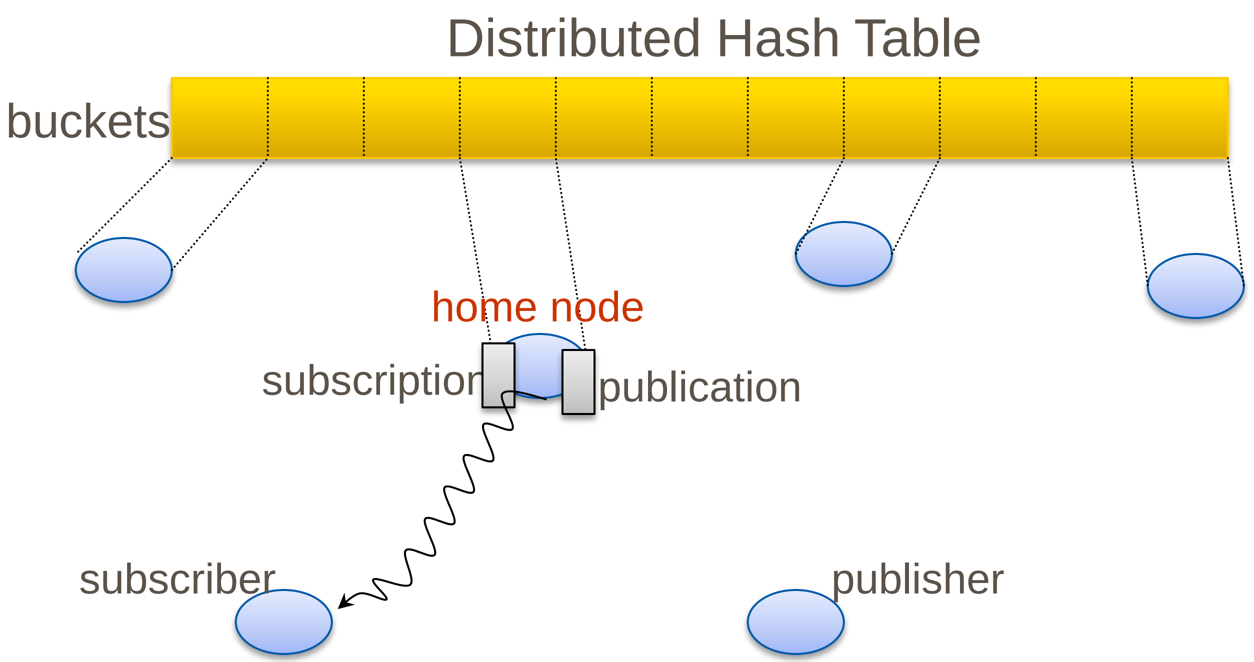
A matching publisher & subscriber must come up with the same hash keys based on the content



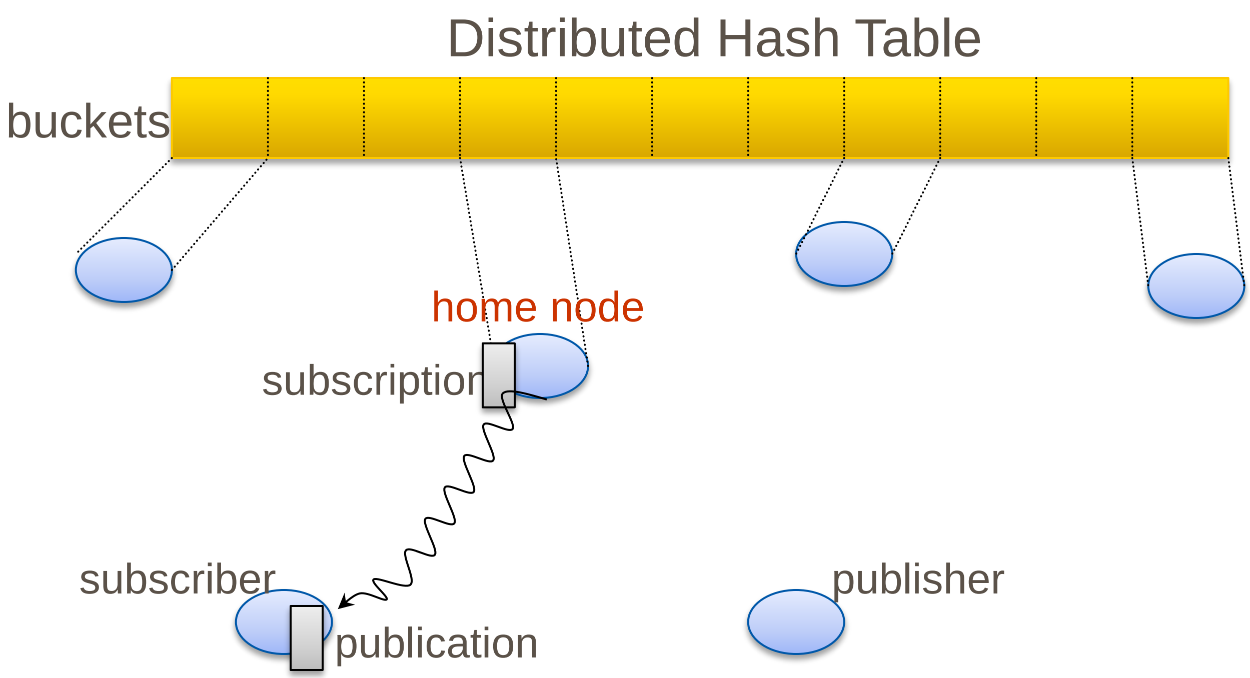
A matching publisher & subscriber must come up with the same hash keys based on the content



- A matching publisher & subscriber must come up with the same hash keys based on the content



A matching publisher & subscriber must come up with the same hash keys based on the content



Rendez Vous Routing

Problem:

- bottleneck at hash bucket nodes
  - Structuring subscribers as a multicast diffusion tree
- Restrictions on the subscription language:
  - Mapping between multidimensional multi-typed subscriptions to the uni or bi-dimensional space of a structured overlay is not straightforward (e.g., string manipulation)

Strong points:

- Handle inherently dynamic changes in the overlay

TECHNISCHE UNIVERSITÄT DARMSTADT

Motivation – Why Multicast?

Paradigm shift in the distribution model of TV

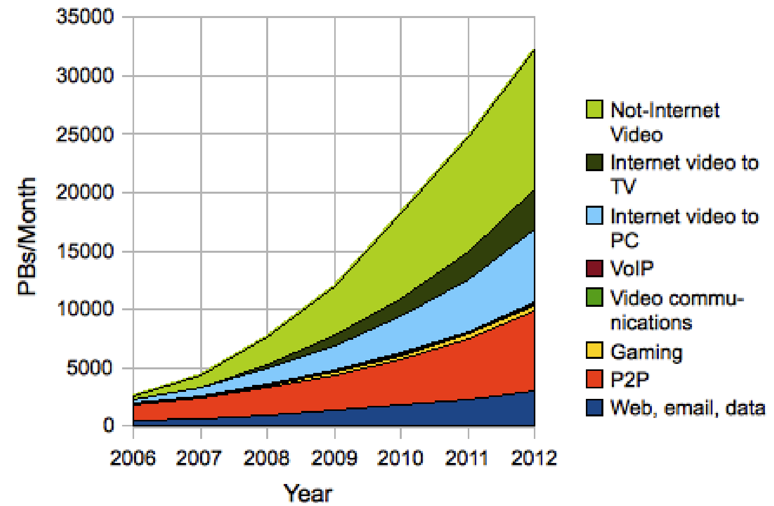
- From convenient broadcast transmission
- To a many-to-many, user-centered transmission
  - “I find”, “I select”, “I schedule”, “I interact”
  - Real-time streaming of TV program
  - Video-on-demand (Youtube)
- Users can choose between huge pool of content from an endless pool of producers → all using multicast

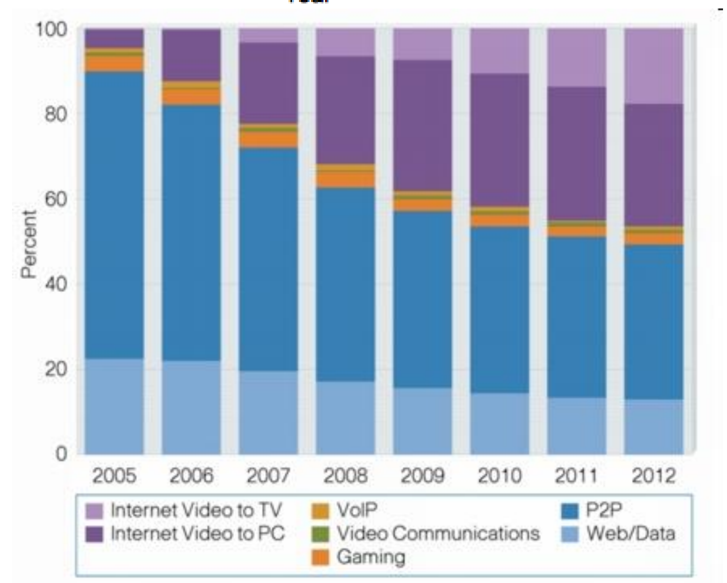
Content providers started using P2P-based solutions

- Deal with huge amount of bandwidth
- And costs (E.g. Youtube 1Mio US$ per day)

IP-based TV / VoD / streaming is a growing market

Distribution of Global Internet Traffic





Data source: Cisco visual networking index, 2009

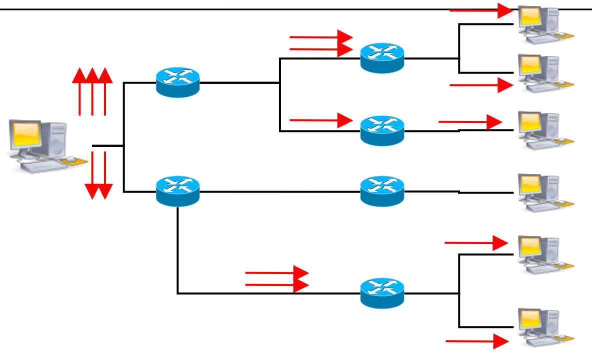
Applications using Multicast

Application	Sender	Group Size	Membership Change Rate	Data Rate	Time Bounded	Type
Voice Conferencing	Multiple	Small	Low	Low	Highly	Live
Video Conferencing	Multiple	Small	Low	Very high	Highly	Live
Multiplayer Gaming	Multiple	Small	Low	Medium	Highly	Live
Personal audio / video broadcast	Single	Small	Low	High	Medium	Live or on-demand
Whiteboard	Multiple	Small	Low	Low	Some delay Acceptable	Live
IPTV	Multiple	Large	High	High	Some delay Acceptable	Live or on-demand
Internet Radio	Multiple	Large	High	Medium	Some delay	Live or on-demand
Collaborative document editing	Multiple	Small	Low	Low	Low delay Acceptable	Live

John F. Buford, Networking and Applications

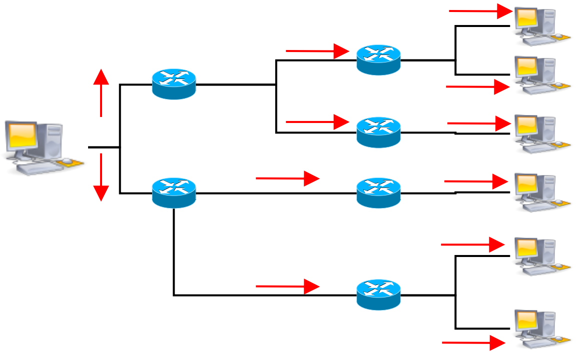
Unicast

- Point-to-point communication
- Data delivered from sender to specific receiver
- Replicated unicast necessary to send N copies



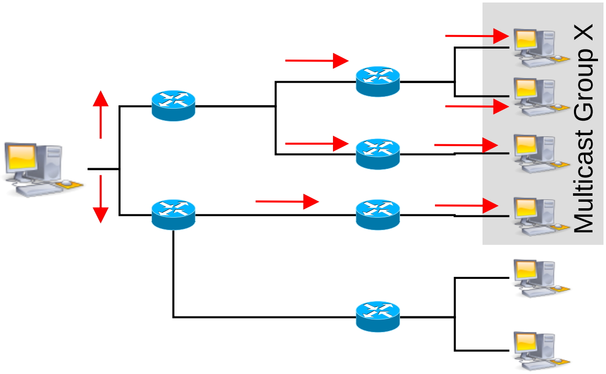
Broadcast

- Point-to-multipoint transmission
- Indiscriminate transmission of data
- Associated with traditional radio / tv transmission
- Every connected receiver gets the data



Multicast

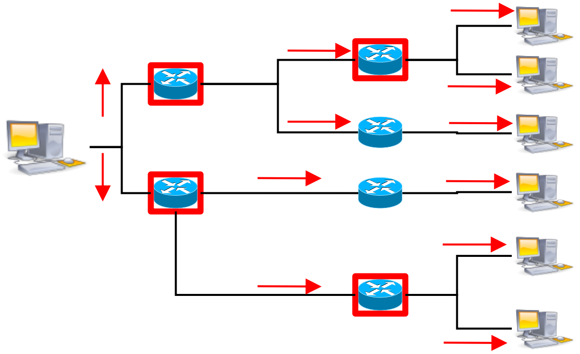
- Multipoint-to-multipoint transmission
- Mostly designed for IP multicast
  - Data replicated at IP level by routers



IP Multicast vs. Overlay Multicast vs. Peercast

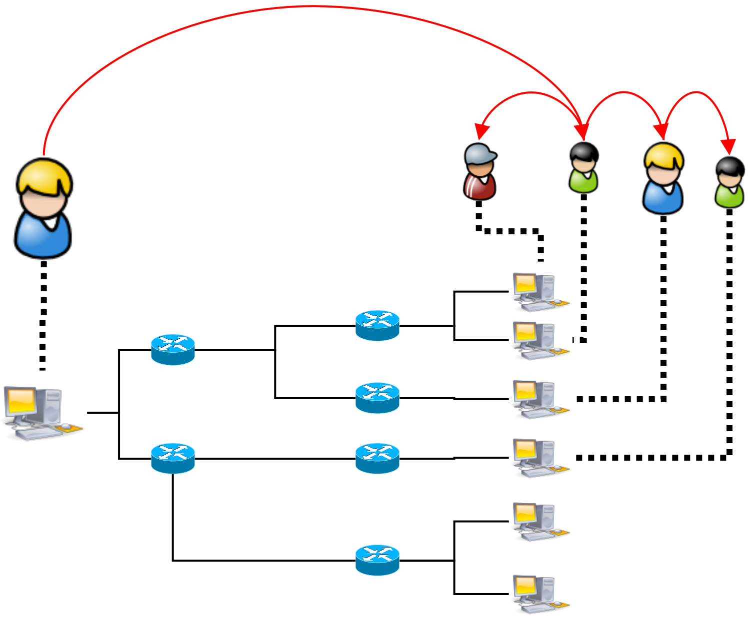
IP Multicast

- Multicast service at IP level
- Data is duplicated at routers
- Also called native multicast



Overlay Multicast / Application Layer Multicast

- Multicast at application layer / overlay
- Group handling, routing, and tree construction done at overlay level



Peercast

- Multicasting, broadcasting, or unicasting using a P2P network

Why not using IP multicast?

Scalability to number of groups

- Routers maintain per-group state
- Aggregation of multicast addresses is complicated

Supporting higher level functionality is difficult

- IP Multicast: best-effort multi-point delivery service
- End systems responsible for handling higher level functionality
- Reliability and congestion control for IP Multicast complicated

Inter-domain routing is hard

Deployment is difficult and slow

- ISP’s reluctant to turn on IP Multicast

→Other multicast solution necessary!

→Independent from autonomous systems

→Easy do deploy!

4.2 Application Layer Multicast

Multicast functionality integrated into application/overlay layer

Packets are replicated at application layer

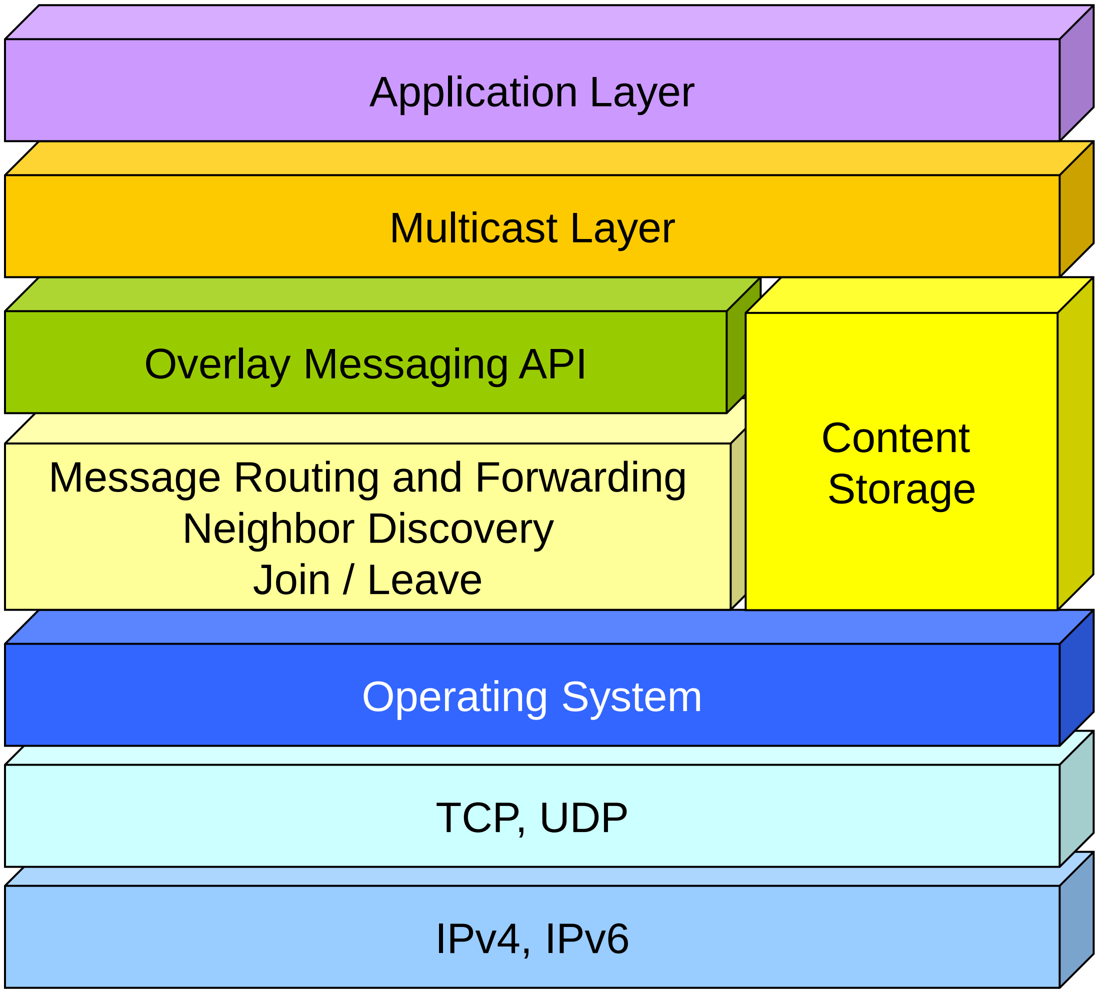
General functions:
- Group creation and management
- Message/content dissemination
- Routing
- Access control and accounting

Easier to deploy

Scalability to number of sessions in the network
- Routers do not maintain per-group state
- End systems do, but they participate in very few groups

Potentially simplifies support for higher level functionality
- Leverage computation and storage on end systems
- Leverage solutions for unicast, congestion control, and reliability
  - For example, for buffering packets, transcoding, ACK aggregation

4.3 ALM Integration into Protocol Stack



4.4 Design Considerations for ALM

Quality of the data delivery path

- Link stress
- Stretch (relative delay penalty)
- Responsiveness

Control overhead

Dissemination overhead

Robustness of the overlay

- Error recover latency at packet loss
- Error recover latency at node failure
- (Average) loss rate per node

Scalability

- Max number of multicast groups
- Maximum group size

Security

Reliability and Availability

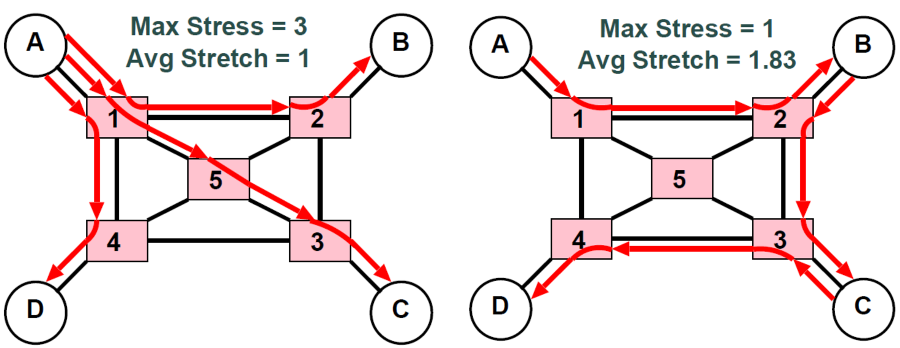
Other important metrics for comparing systems (especially for streaming)

- Startup latency
- Join latency

Design differs in

- Overlay structure
- Message routing
- Tree management

Examples showing link stress:



A Comparative Study of Application Layer Multicast Protocols, Suman Banerjee et al.

4.5 Classification of ALM Systems

Group management / signaling topology

- How to manage groups?
- Which topology to use for signaling?
  - Tree vs. Mesh vs. Hybrid
  - How many trees exist? What is the tree used for?
    - Shared single tree vs. Source specific tree (multi tree approach)

Message/Content Dissemination

- Who is initiating transmission?
  - Push vs. Pull
- Which topology to use for content dissemination?
  - Tree vs. Mesh vs. Hybrid

End system-based vs. Proxy-based

- Dedicated entities (proxies) provided by overlay?
  - Stabilize overlay in term of high peer dynamics

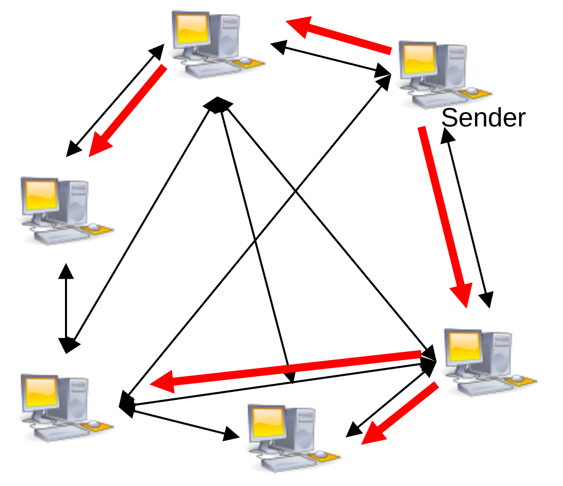
Group Management Signaling Paradigms

Basic questions

- How is signaling traffic transferred?
- How is the streaming/multicast topology created?

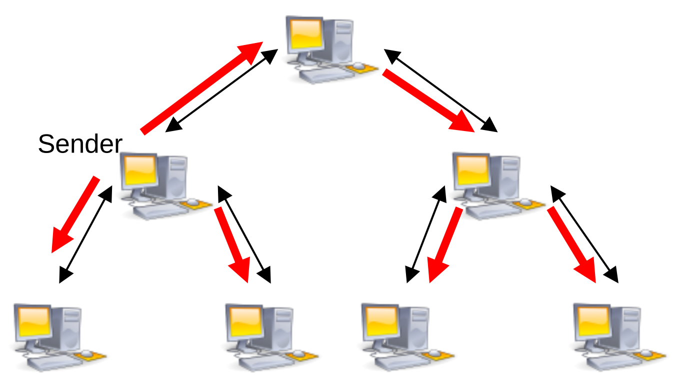
„Mesh-first, tree-based“

- Create an explicit signaling „mesh“ (overlay)
- Streaming
  - in separate streaming overlay
  - along selected links of signaling mesh



„Tree-first“

- Plain (streaming-) neighbor selection
- Signaling along same links
- Only for single „channel“ topologies



Message/Content Dissemination Paradigms

Pull-based streaming

- Streams are divided into chunks
- Each peer requests the chunks it needs
- Simple example:
  - BitTorrent, rarest first exchanged for request in order
    - naive file-sharing like distributed download of stream
    - (+) very robust
    - (-) slow, high delays, significant signaling overhead (request each packet, but little compared to stream)
    - „OK“ for Video-on-Demand (VoD)

Pushed (subscription-based) streaming

- Explicit construction of streaming topology (explicit parent <-> child relations)
- Parent forwards packet to child(ren) on reception
- (+) little overhead, small delays
- (-) less robust (topology repair on node failure/departure)

Different Types of Multicast Trees

Shared tree / Single tree approach

- A single (minimum) spanning (Steiner-) tree connects all participants
- Acyclic graph → simplifies routing
- But:
  - Bandwidth of leaves not used
  - High load on interior nodes
  - Sensitive to partitioning

Source specific trees / Multi-tree approach

- (Shortest path) Trees are created for each potential source
- Acyclic graph → simplifies routing
- Improved fairness of resource sharing
- Improved performance due to avoiding of bottlenecks
- But:
  - Increase overhead for multiple tree construction

Mesh-first, tree-based ALM, streaming along signaling topology

- Scribe

Mesh-first, tree-based ALM, streaming along signaling topology

- CAN-multicast

Mesh-first, tree-based ALM, separate streaming topology

- ESM / Narada

Tree-first

- Banana-Tree

4.7 Scribe – based on Pastry

Scalable application-level multicast infrastructure

- Built on top of the Pastry overlay
- Best-effort message dissemination
- No in-order delivery guaranteed

Multicast groups: a “Scribe group” per session

- Unique group-id; multicast tree to disseminate messages
- Root of tree = rendezvous point
- GroupID = hash of group’s textual name concatenated with it’s creator’s name

A Scribe node may

- create a group
- join a group
- be the root of a multicast tree or
- act as a multicast source

Original paper:

- “The Design of a Large-Scale Event Notification Infrastructure“
- Rowstron, Kermarrec, Castro, Druschel (MSR, Rice University; 2001)

Scribe - API

create (credentials, group-id)

- create a group with the group-id

join (credentials, group-id, message-handler)

- join a group with group-id
- Published messages for the group are passed to the message handler

leave (credentials, group-id)

- leave a group with group-id

multicast (credentials, group-id, message)

- publish the message within the group with group-id

Creating a group:

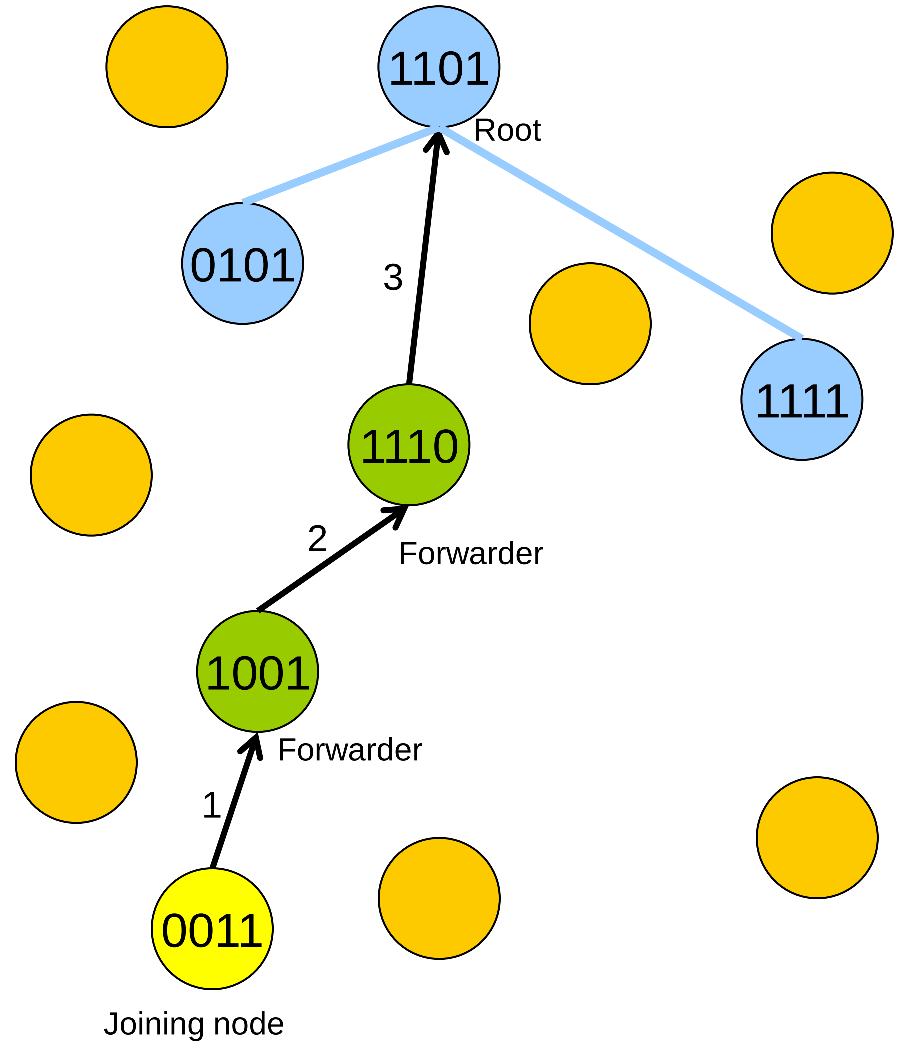
- Send a CREATE message with the group-id as the key
- Pastry delivers message to root (key)
- This node becomes the rendezvous point
- deliver method checks and stores credentials and also updates the list of groups

Scribe - Join Group

Send JOIN message with group-id "0011" as key 0011

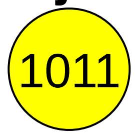
Pastry routes to rendezvous point

- If intermediate node is forwarder:
  - Add the node as its child
  - Do not further forward message
- If intermediate node is not a forwarder:
  - Creates child table for the group, and adds the node
  - Forward JOIN towards the rendezvous point
- If node is root:
  - Terminate JOIN message from the child
  - Creates child table for the group, and adds the node



Scribe - Join Group Cont

Node 1011 wants to join the multicast group

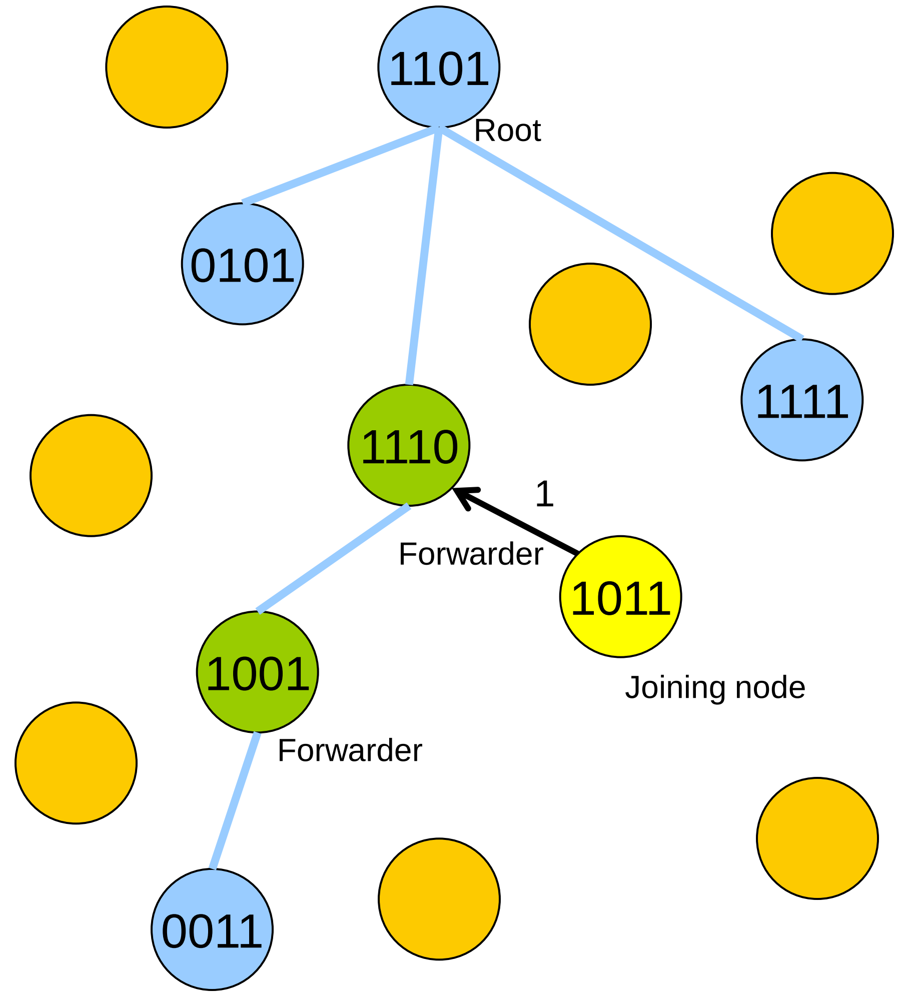


Joining node

- sends join message to node 1110

Node 1110 is already forwarder for the group

- Add joining node to child list
- Terminate join request



Scribe - Message Multicast

Multicast a message to the group

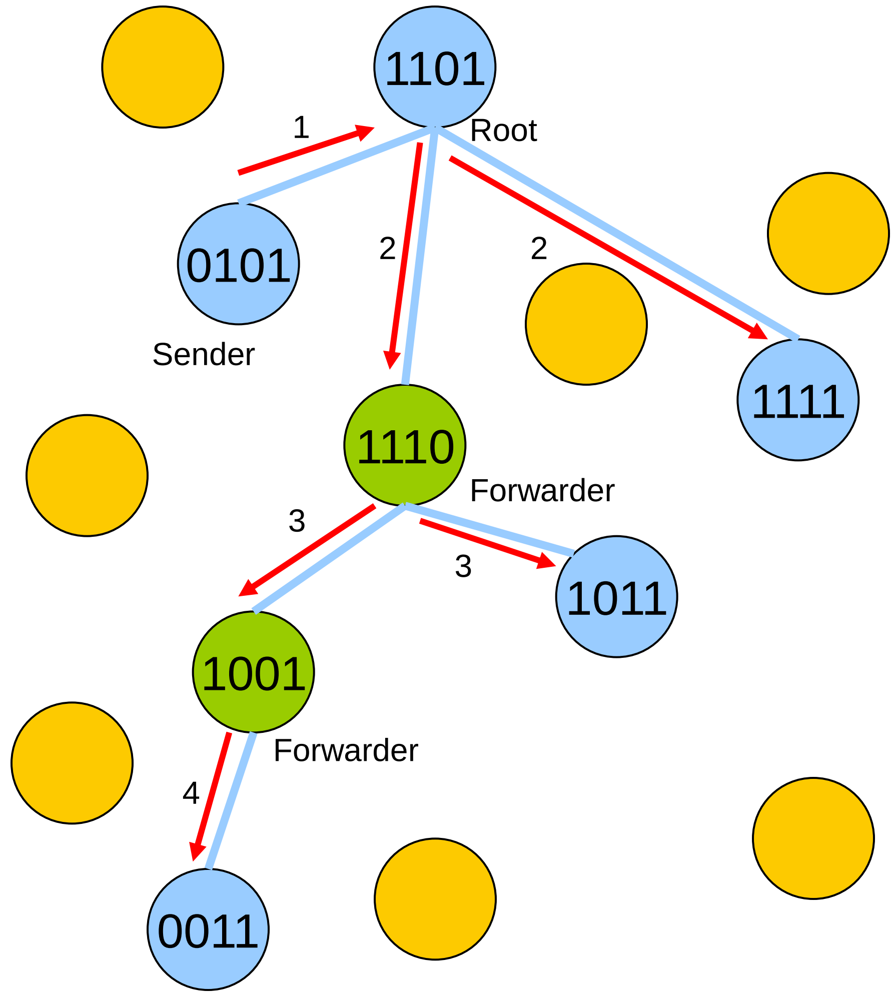
- Scribe node sends MULTICAST message to the rendezvous point
- A node caches the IP address of the rendezvous point
  - so that it does not need Pastry for subsequent messages

Single multicast tree for each group

- Access control for a message is performed at the rendezvous point

Root of multicast-tree

- disseminates message to its children

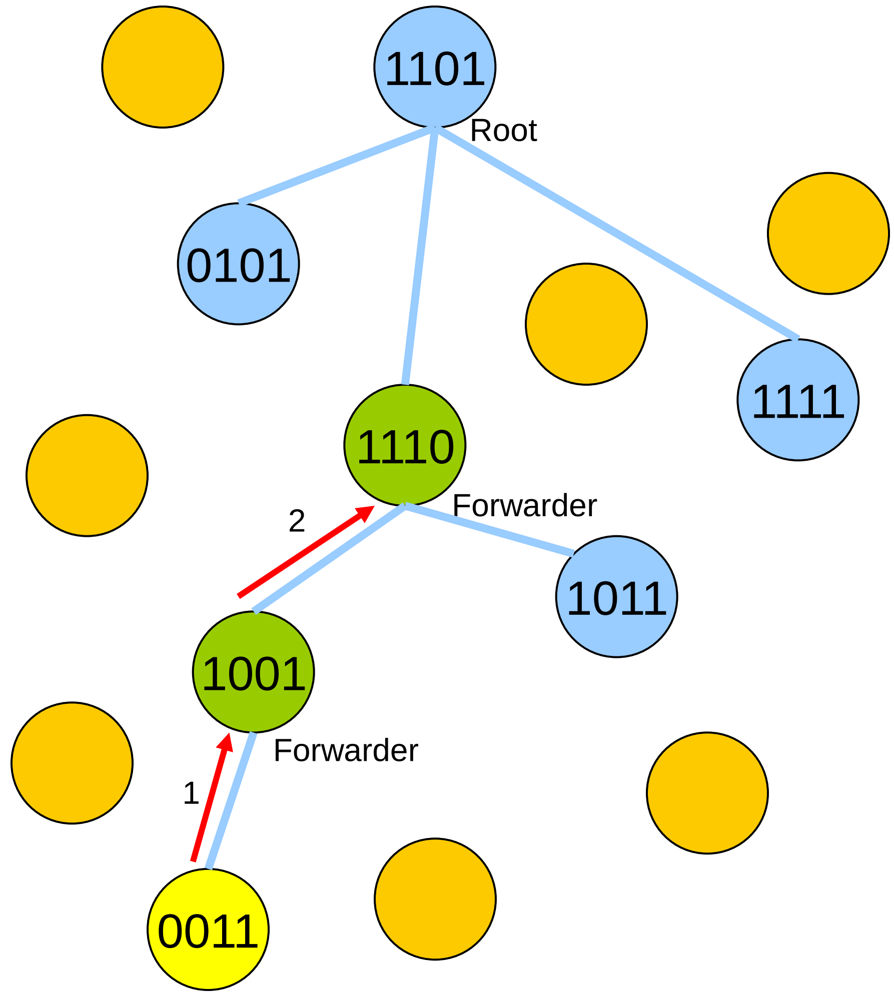


Scribe - Leave Group

Scribe node records locally that it left the group

If the node has no children in its table

- it sends a LEAVE message to its parent
- The message travels recursively up the multicast tree
- The message stops at a node which has children after removing the departing node



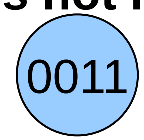
Scribe - Multicast Tree Repair

Broken link detection and repair

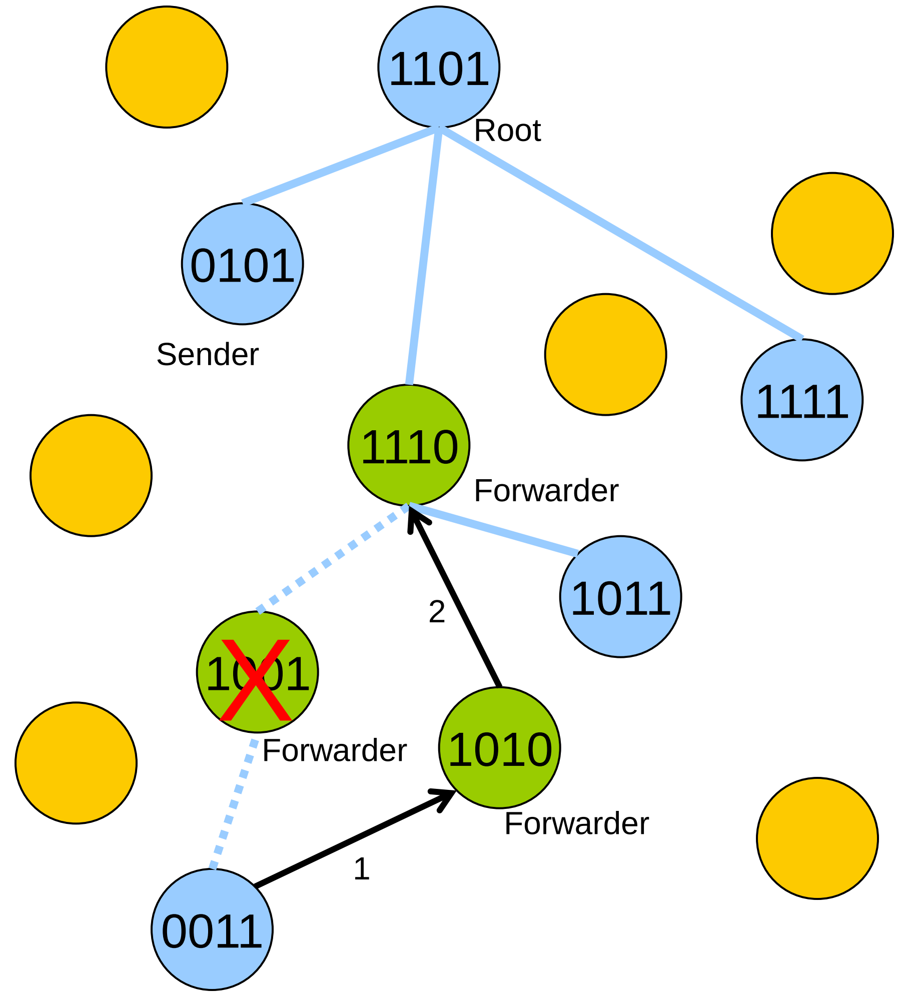
Non-leaf nodes

- send heartbeat message to children
- Multicast messages serve as implicit heartbeat

If child does not receive heartbeat message



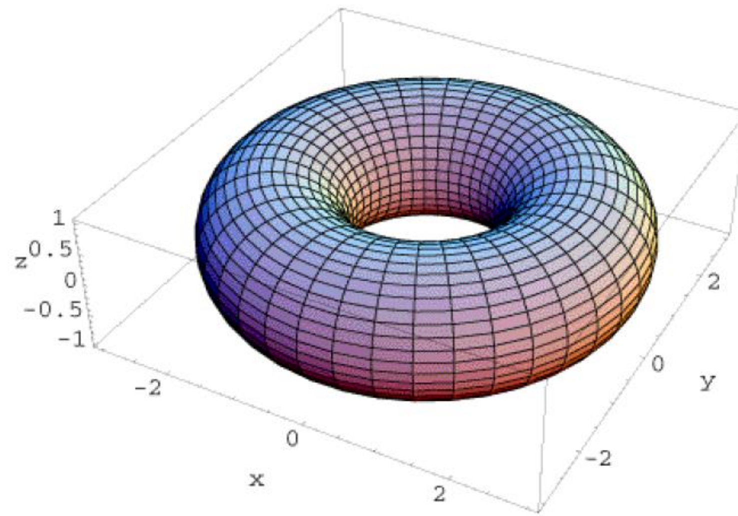
- child assumes that the parent has failed
- finds a new route
  - by sending a JOIN message to another node towards the group ID



4.8 Multicast based on CAN

CAN = Content Addressable Network

- Node form d-dimensional Cartesian space
- Begin and end of space are connected
  - 2D → Torus



Multicast group members form a "mini" CAN consisting of CAN nodes

Group ID is hashed on the coordinate space (x,y)

Node responsible for (x,y) becomes bootstrap node for new "mini" CAN

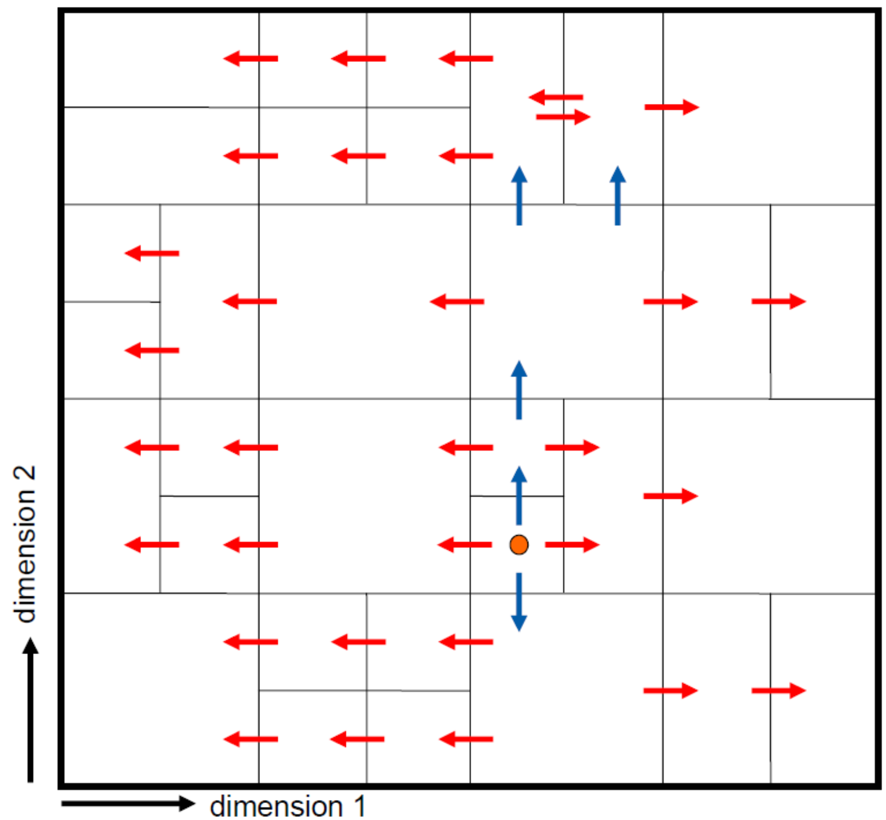
Group members follow usual CAN construction process

CAN - Multicast

TECHNISCHE UNIVERSITÄT DARMSTADT

Multicast Forwarding in CAN DHT

- Source sends message to all neighbors
  - If a neighbor receives the message along dimension i,
    it forwards the message to …
    - …all neighbors along the ith dimension (simple forwarding along dimension i)
    - …all neighbors, whose zones are neighboring in dimension 1…(i-1)
  - Stop forwarding
    - after packet traversed half of the address space along dimension I



Advantage:

- Packets are not received multiple times
  - if name space is well (equally) partitioned
- Avoids loops during forwarding

TECHNISCHE UNIVERSITÄT DARMSTADT

End System Multicast - ESM

- Goal:
  to implement ALM for small, sparse groups
- Introduces Narada protocol for group management

Properties:

- Self-Organization in a fully distributed fashion
- Overlay Efficiency: Multicast tree must be efficient
- Adaptive to Network Dynamics, Improve overlay by underlay metrics

Original paper:

- A Case for End System Multicast, Yang-hua Chu et al., 2002

Narada - Mesh and Tree Construction

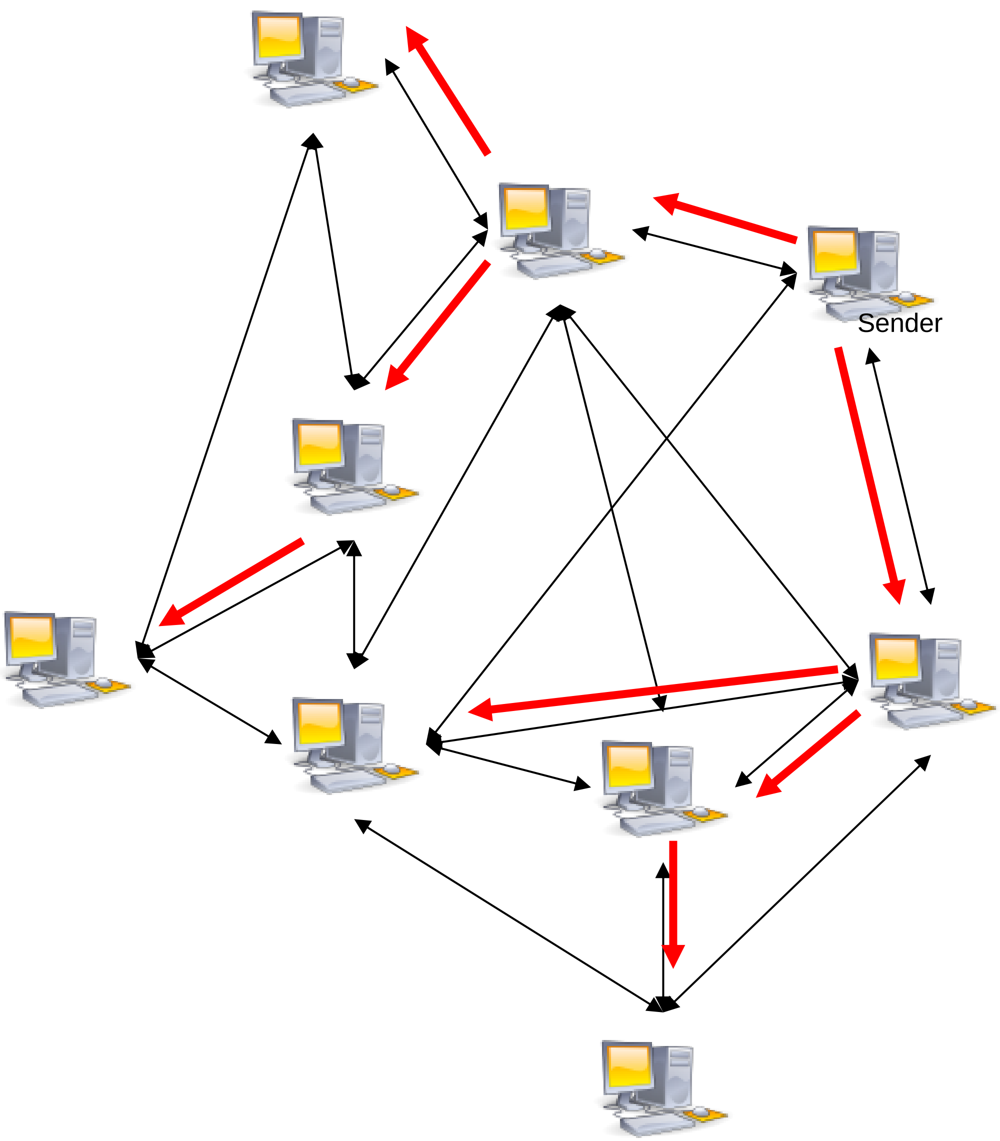
Two step protocol for tree construction:

Step 1

- Create "mesh":
  - Subset of complete graph
    - includes all group members
    - may have cycles
- Optimize mesh
  - by adding and removing links
    - According to delay and bandwidth utilization

Step 2

- Build spanning tree per source within the mesh
- Constructed using
  - reverse shortest path spanning tree algorithm
- Small delay from source to receivers



Narada - Mesh Maintenance

Mesh maintenance is shared jointly among group members

- Each peers stores information about all group members
  - i.e.
    - increases robustness of overlay
    - Increases overhead (only small and midsized groups are considered)

Refresh messages sent periodically by each peer k

- With monotonically increasing sequence number s_k

Each peer i stores information for every member k in the group

- The k´s member IP address
- The last sequence number peer i
  - knows that member k has issued
- The local time when peer i first received the sequence number s_k

- If peer i has not received any update within T seconds
  → assume node is not available
  → perform maintenance

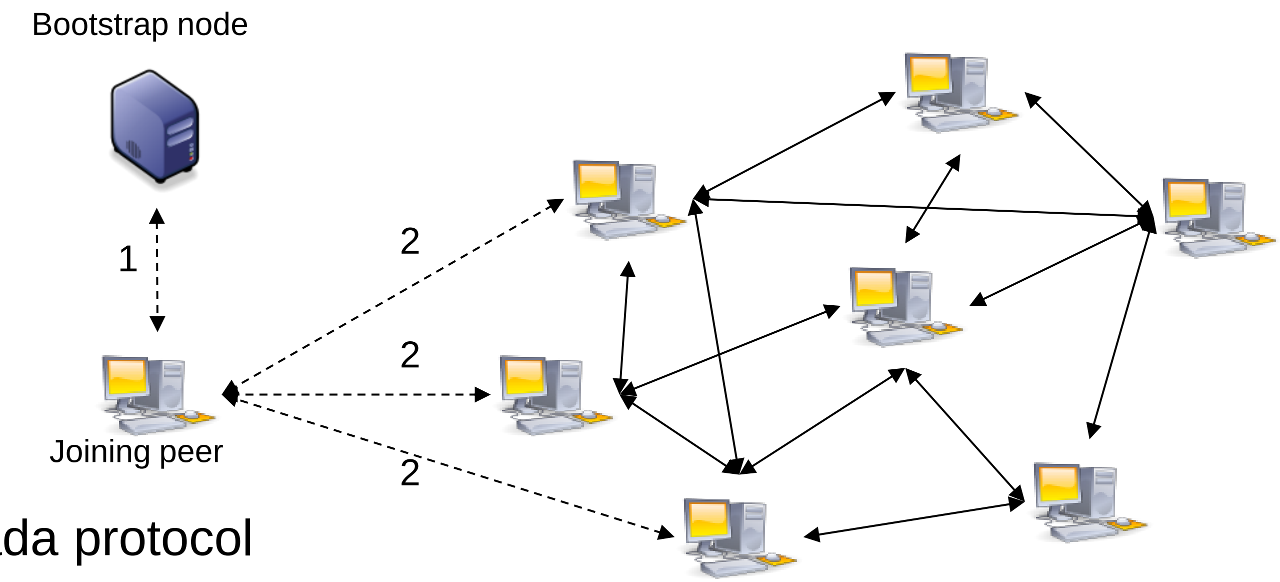
Peers periodically exchange information about existing peers in the group

Narada - Join

Peers decides which group to join

Peer contracts bootstrap node

- for list of neighbor nodes within group
- But, bootstrapping not covered by Narada protocol



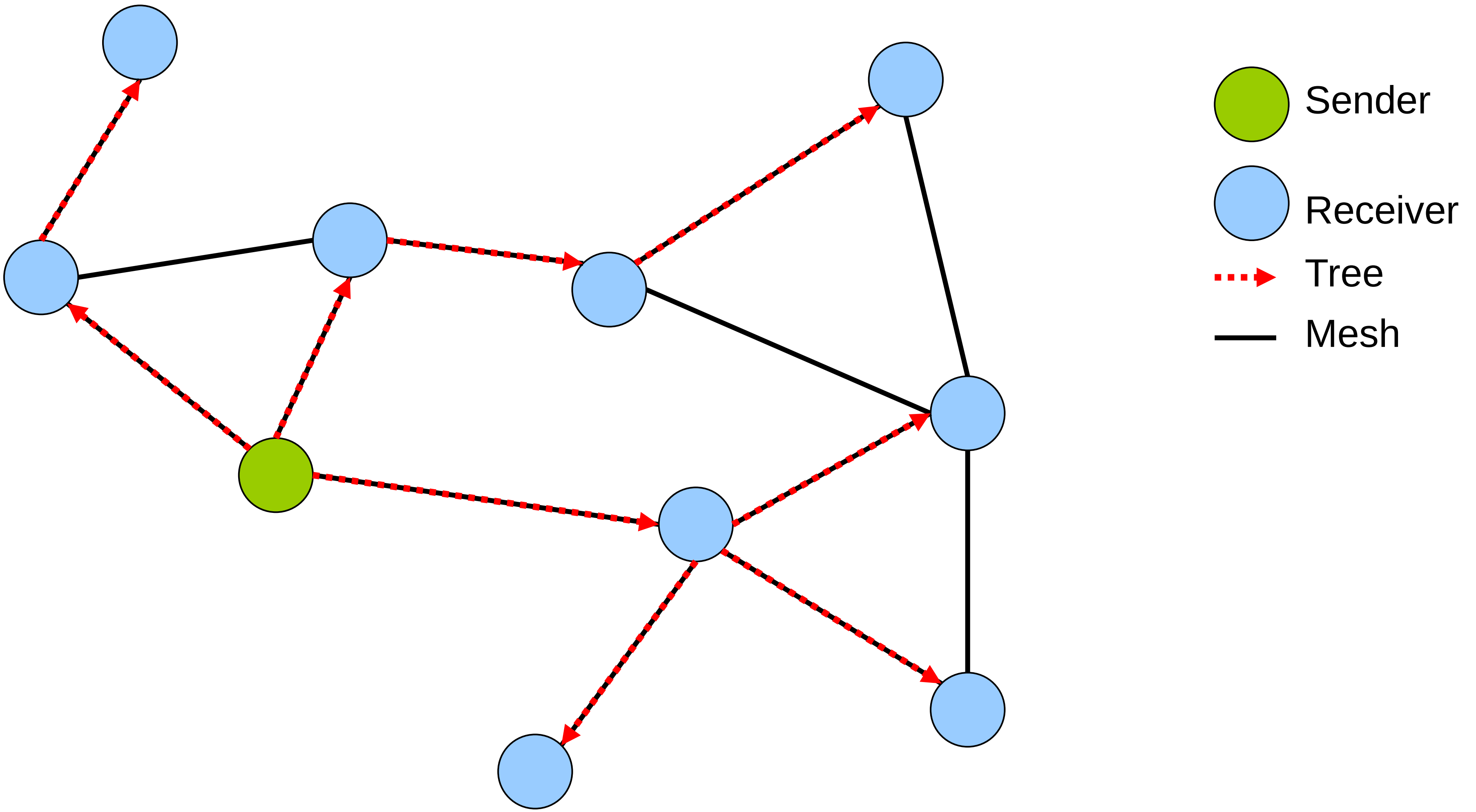
Joining peer contacts

- a subset of the nodes from the neighbor list

Peers exchange information

- about neighbors and merge information

Narada – Overlay Structure



Narada - Optimizing Mesh Quality

Members periodically probe other members at random

New link added if

- utility gain of adding link greater than “add threshold"
  - Based on number of members to which routing delay improves
    - i.e. how significant the improvement in delay to each member is

Members periodically monitor existing links

Existing link dropped if:

- cost of dropping link < drop threshold
  - Based on number of members to which routing delay increases, per neighbor

Add/Drop thresholds are functions of:

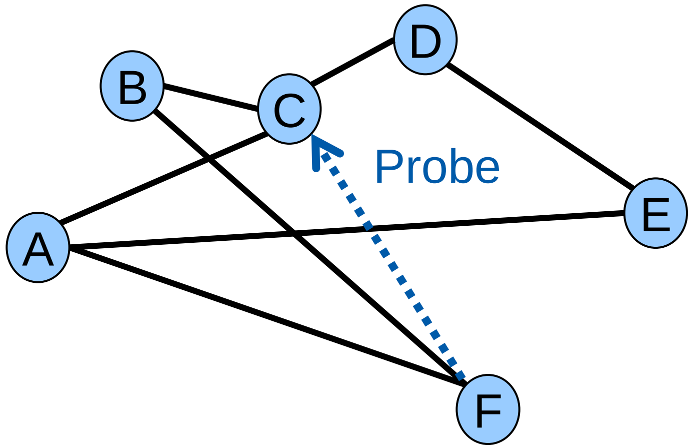
- Member’s estimation of group size
- Current and maximum degree of member in the mesh

Stability

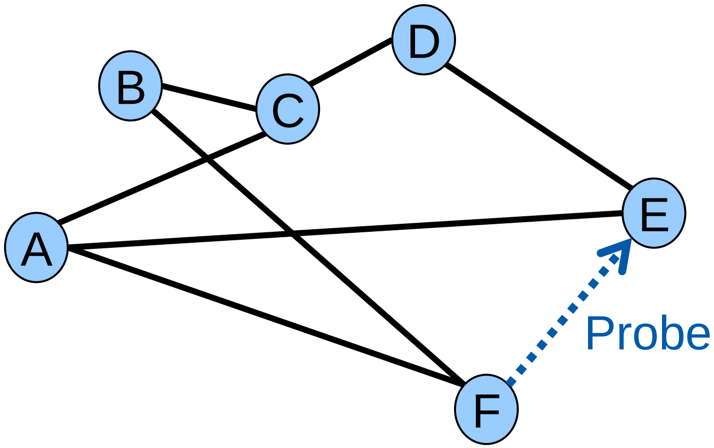
- A dropped link will not be immediately re-added

Partition Avoidance

- A partition of the mesh is unlikely to be caused as a result of any single link being dropped



Delay improves to C, D
but marginally.
Do not add link!



Delay improves to D, E
and significantly.
Add link!

University of Michigan

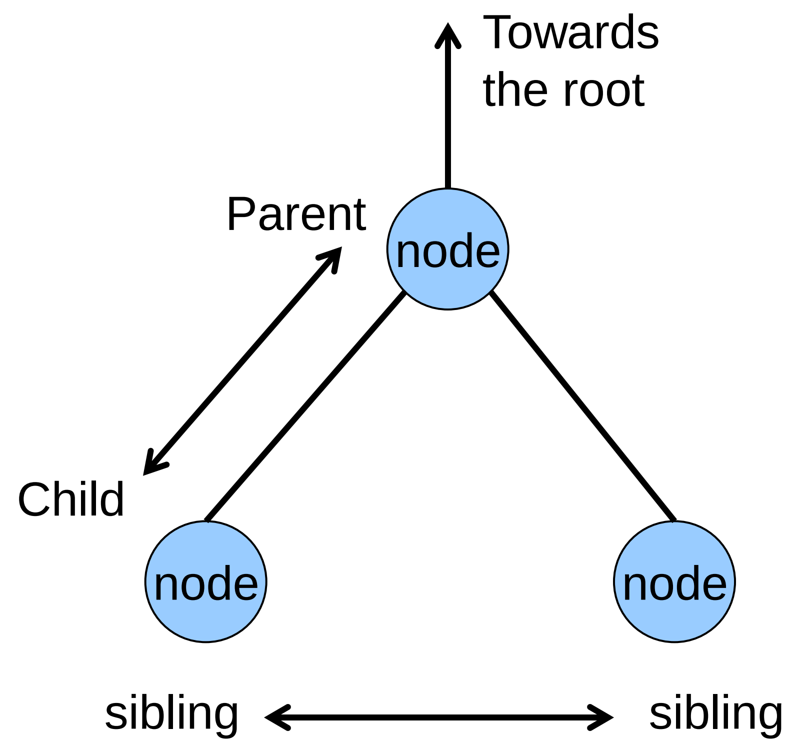
- David Helder and Sugih Jamin
  - (Zattoo) in 2000

Goal:

- tree-first creation of a tree-based overlay multicast

Main approach

- Create a tree starting at a root
- Join nodes at arbitrary node
- Perform only local changes to adapt the tree
- Next node on path to root is „parent“
  - (parents forward stream to children)
- Children of same parent are „siblings“



Banana Tree - Main Functionality

Existence of a bootstrap protocol is assumed

A host joins a group

- by becoming the child of a node currently in the tree
  - (e.g., the root node)
- A node that joins an empty multicast group is the new root node

Any node can multicast

- To multicast, a node sends the packet(s) to its neighbors
- On reception of a packet, each node forwards it to all other neighbors

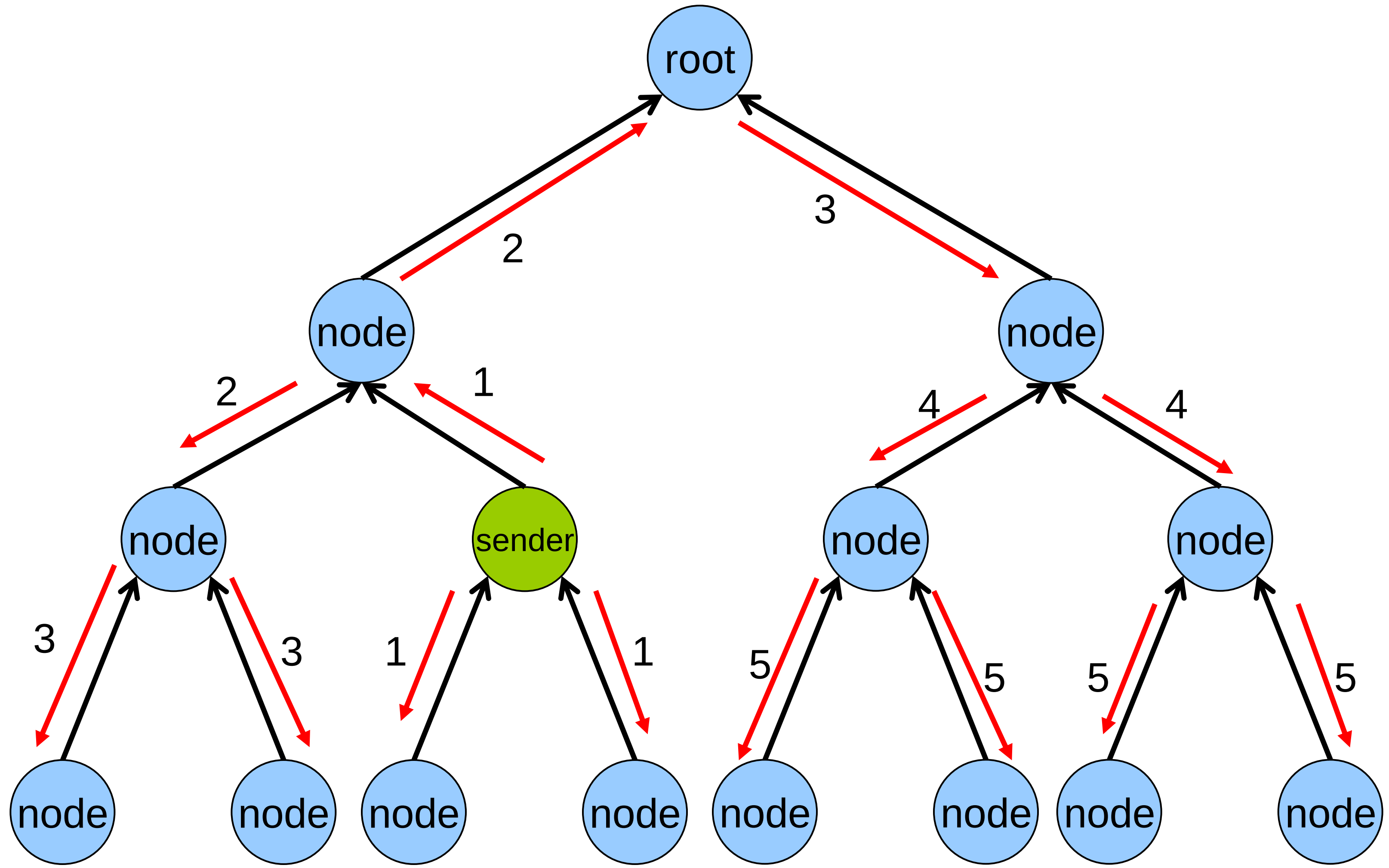
In case of a departing parent

- the tree partitions and children reconnect to root

Nothing is done on failure of child(ren)

- successors will automatically reconnect to root

Banana Tree - Multicast

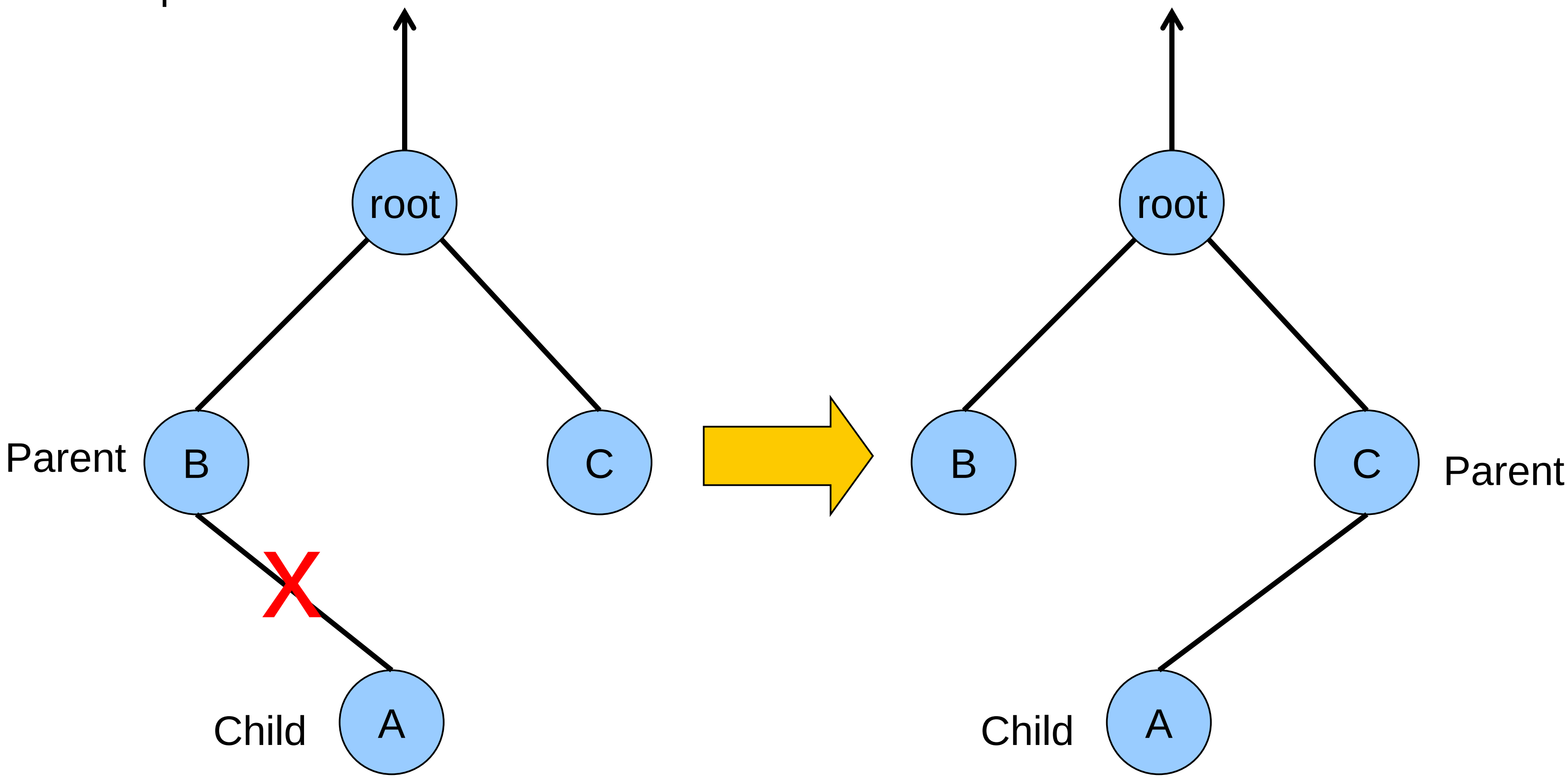


Banana Tree - Changing Parents in the Group

TECHNISCHE UNIVERSITÄT DARMSTADT

Changing the parent nodes is called “switching”

- Optimize resource usage within the system
- Example: node A switches to node C



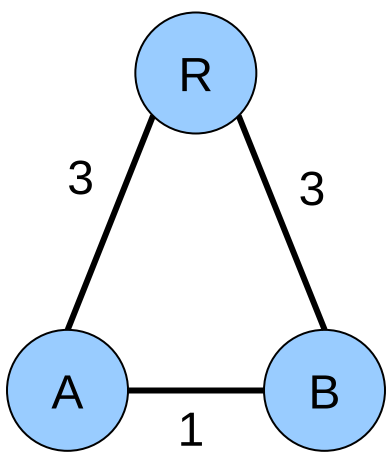
Banana Tree - Optimizing the Tree

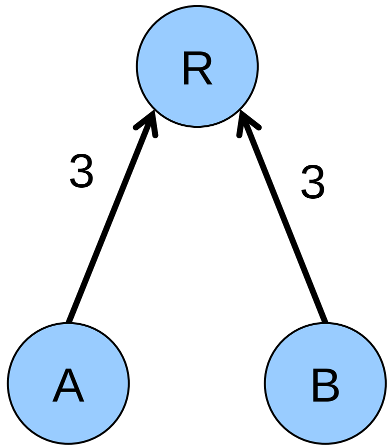
Why would a node switch parents?

So far the tree evolves purely by lack of bandwidth

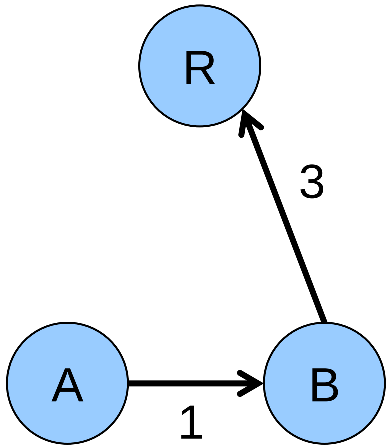
- Connect to node (root)
- If bandwidth depleted, child switches down

Additional switches of parents to optimize the tree for low cost.





Tree costs = 6



Tree costs = 4

Banana Tree - Switching Policies

Parents send information about siblings regularly

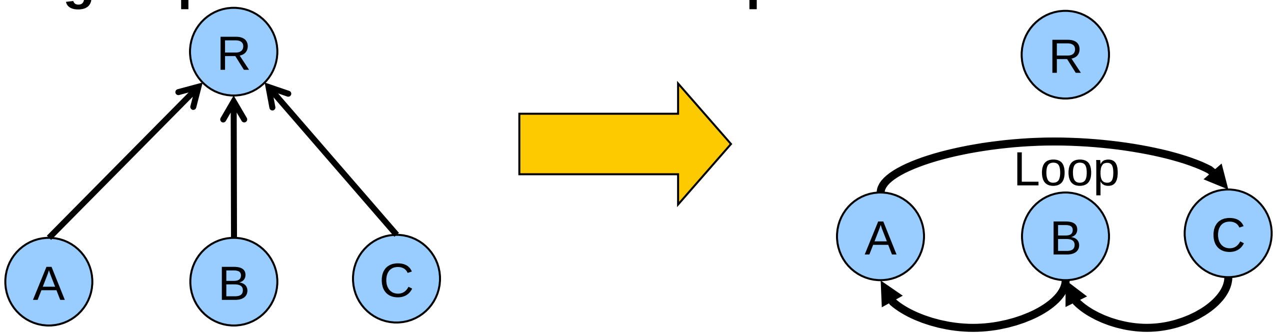
Siblings ping each other to determine distance

To switch, a switching request is sent to “potential parent”

Potential parent only accepts if

- it is not switching itself at the same time
- always reject while switching itself → avoid loops

Switching request includes current parent information



4.11 Some existing ALM Systems - Comparison

Protocol	**Scribe**	**ESM**	**CAN**	**Banana Tree**
Group size	Large groups	Small groups	Large groups	Large groups
Approach	▪ Multicast tree is built on top of Pastry. ▪ Uses reverse path forwarding to build a multicast tree per group. ▪ Each group is identified by the groupID. ▪ Multicast message propagates through the multicast spanning three.	▪ Multicast group members ▪ self-organize into an overlay structure. ▪ End hosts (peers) periodically exchange group membership and routing information, ▪ build a mesh based on end-to-end measurements, ▪ and run a distributed distance vector protocol to construct a multicast delivery tree.	CAN-based multicast has two steps: ▪ (1) the members of the group first form a group specific overlay; ▪ (2) multicasting is achieved by flooding over the overlay, creating a separate overlay per multicast group. ▪ Multicast message is broadcast within each overlay.	▪ Create a tree starting at a root ▪ Join nodes at arbitrary node ▪ Perform only local changes to adapt the tree ▪ Next node on path to root is „parent“ (parents forward stream to children) ▪ Children of same parent are „siblings“

John F. Buford, Networking and Applications