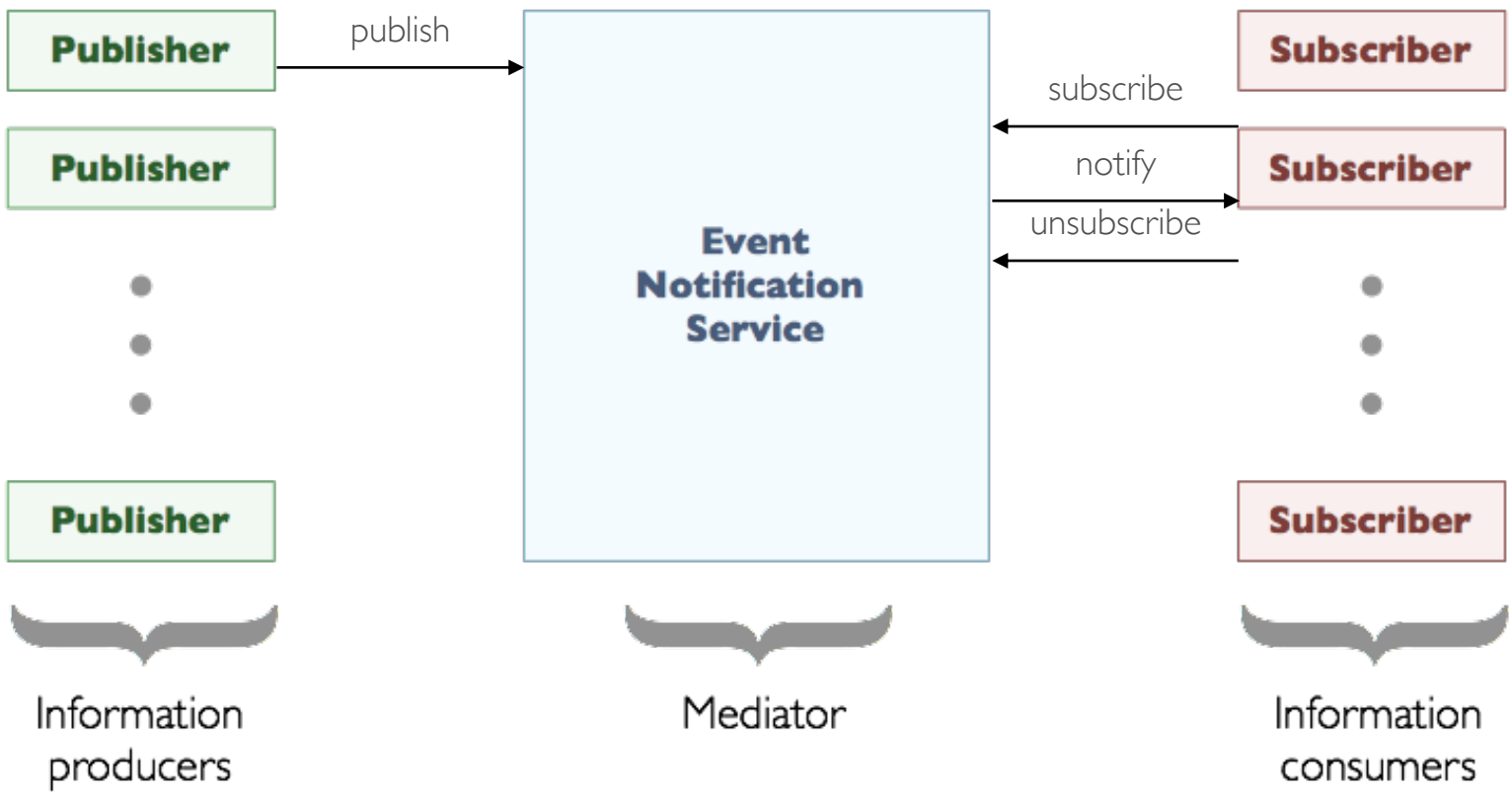
MIDLAB Middleware Laboratory

Università di Roma "La Sapienza"
Dipartimento di Informatica e Sistemistica

Information Dissemination in large scale systems: Publish/Subscribe

Credits: slides provided by Prof. Leonardo Querzoni and Prof. Roberto Baldoni

- The publish/subscribe communication paradigm:
  - **Publishers**: produce data in the form of **events**.
  - **Subscribers**: declare interests on published data with subscriptions.
  - Each **subscription** is a filter on the set of published events.
  - An **Event Notification Service** (ENS) notifies to each subscriber every published event that matches at least one of its subscriptions.



- Publish/subscribe was thought as a comprehensive solution for those problems:
  - **Many-to-many communication model** - Interactions take place in an environment where various information producers and consumers can communicate, all at the same time. Each piece of information can be delivered at the same time to various consumers. Each consumer receives information from various producers.
  - **Space decoupling** - Interacting parties do not need to know each other. Message addressing is based on their content.
  - **Time decoupling** - Interacting parties do not need to be actively participating in the interaction at the same time. Information delivery is mediated through a third party.
  - **Synchronization decoupling** - Information flow from producers to consumers is also mediated, thus synchronization among interacting parties is not needed.
  - **Push/Pull interactions** - both methods are allowed.
- These characteristics make pub/sub perfectly suited for distributed applications relying on document-centric communication.

- Events represent information structured following an *event schema*.
- The event schema is fixed, defined a-priori, and known to all the participants.
- It defines a set of fields or attributes, each constituted by a name and a type. The types allowed depend on the specific implementation, but basic types (like integers, floats, booleans, strings) are usually available.
- Given an event schema, an event is a collection of values, one for each attribute defined in the schema.

- Example: suppose we are dealing with an application whose purpose is to distribute updates about computer-related blogs.

name	type	allowed values
blog_name	string	ANY
address	URL	ANY
genre	enumeration	[hardware, software, peripherals, development]
author	string	ANY
abstract	string	ANY
rating	integer	[1-5]
update_date	date	>1-1-1970 00:00

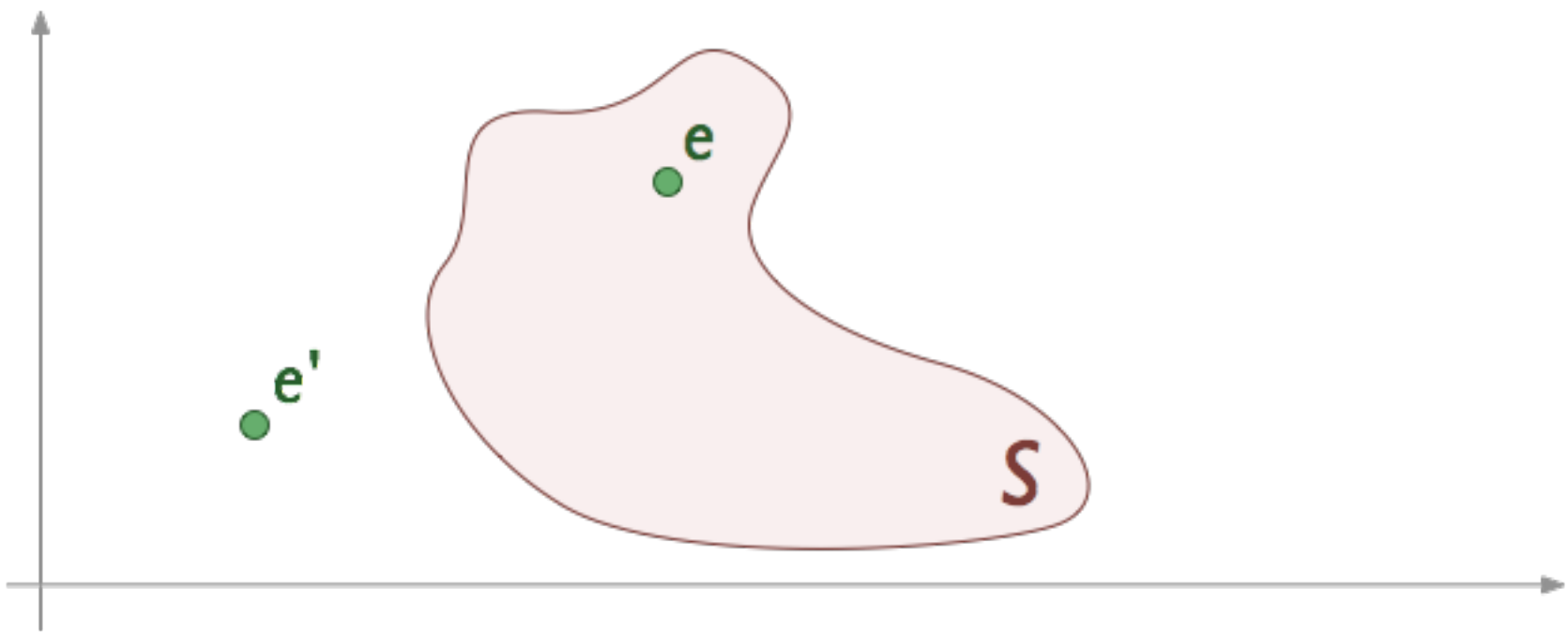
Event Schema

Event

name	value
blog_name	Prad.de
address	http://www.prad.de/en/index.html
genre	peripherals
author	Mark Hansen
abstract	"The review of the new TFT panel..."
rating	4
update_date	26-4-2006 17:58

- Subscribers express their interests in specific events issuing subscriptions.
- A subscription is a *constraint* expressed on the event schema.
- The Event Notification Service will notify an event e to a subscriber *x* only if the values that define the event satisfy the constraint defined by one of the subscriptions *s* issued by x. In this case we say that **e matches *s***.
- Subscriptions can take various forms, depending on the subscription language and model employed by each specific implementation.
- Example: a subscription can be a conjunction of constraints each expressed on a single attribute. Each constraint in this case can be as simple as a >=< operator applied on an integer attribute, or complex as a regular expression applied to a string.

- From an abstract point of view the event schema defines an n-dimensional **event space** (where n is the number of attributes).
- In this space each event e represents a point.
- Each subscription s identifies a subspace.
- An event e matches the subscription s if, and only if, the corresponding point is included in the portion of the event space delimited by s.



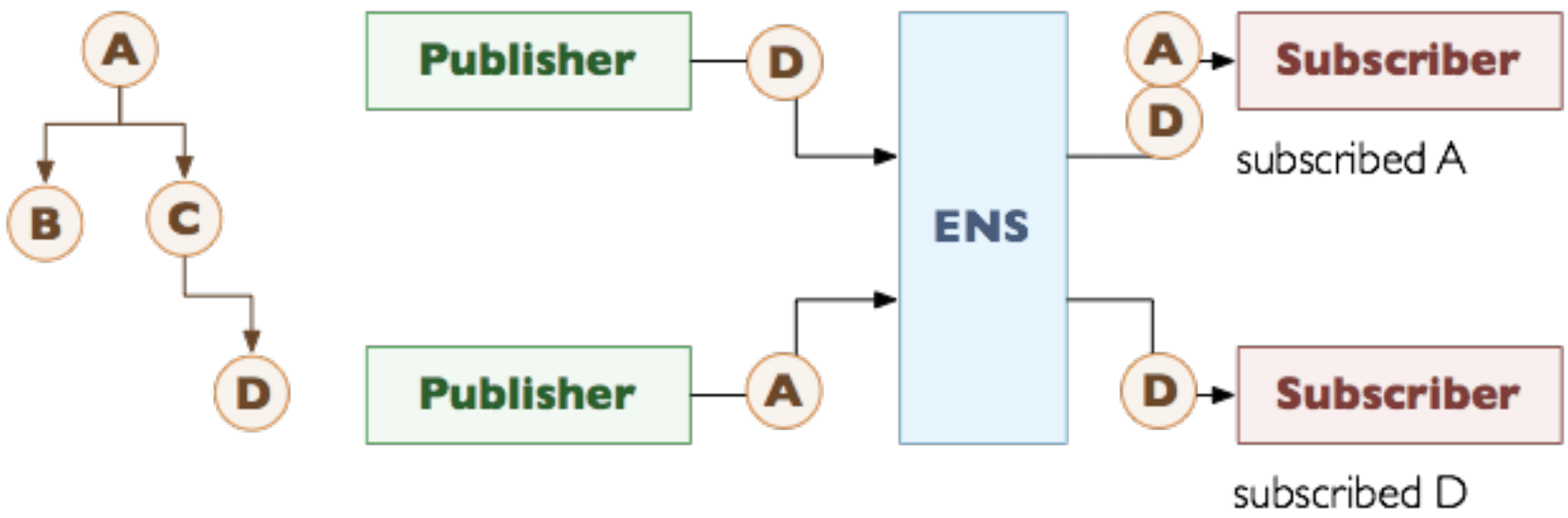
- Depending on the subscription model used we distinguish various flavors of publish/subscribe:
  - Topic-based
  - Hierarchy-based
  - Content-based
  - Type-based
  - Concept-based
  - XML-based
  - .....

- **Topic-based selection**: data published in the system is mostly unstructured, but each event is "tagged" with the identifier of a *topic* it is published in. Subscribers issue subscriptions containing the topics they are interested in.

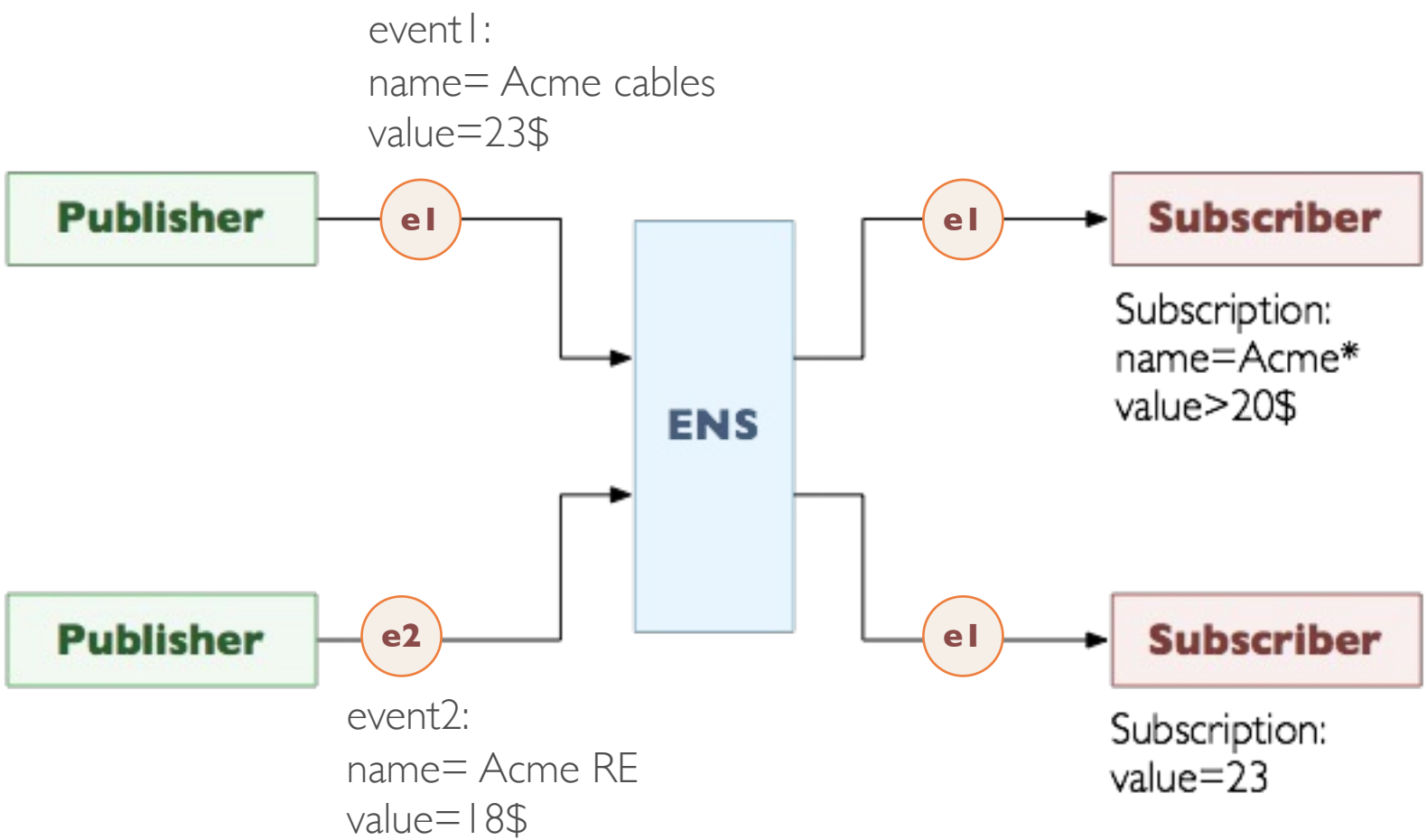
- A topic can be thus represented as a "virtual channel" connecting producers to consumers. For this reason, the problem of data distribution in topic-based publish/subscribe systems is considered quite close to group communications.

- **Hierarchy-based selection**: even in this case each event is "tagged" with the *topic* it is published in, and Subscribers issue subscriptions containing the topics they are interested in.

- Contrarily to the previous model, here topics are organized in a hierarchical structure which express a notion of containment between topics. When a subscriber subscribe a topic, it will receive all the events published in that topic and in all the topics present in the corresponding sub-tree.



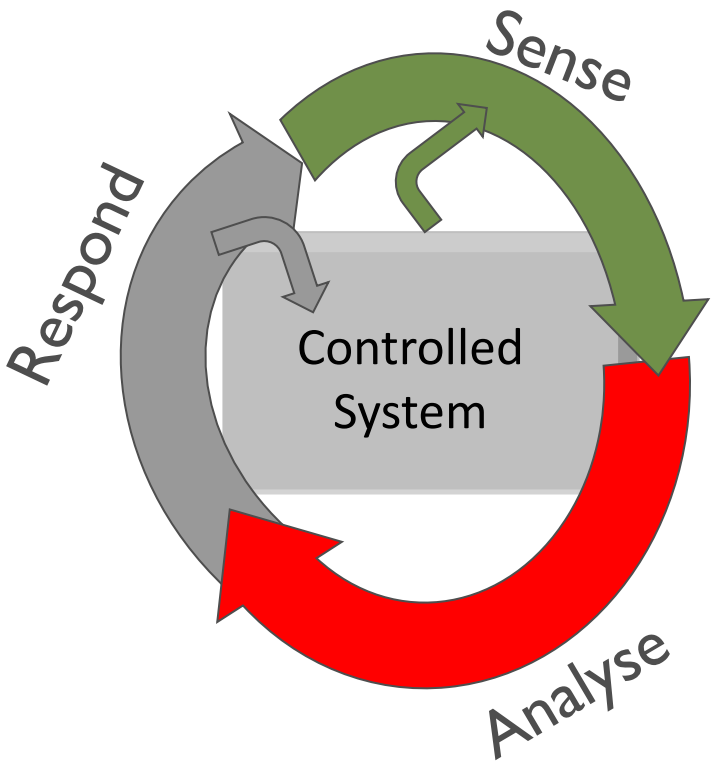
- **Content-based selection**: all the data published in the system is mostly structured. Each subscription can be expressed as a conjunction of constrains expressed on attributes. The Event Notification Service filters out useless events before notifying a subscriber.



Where Publish subscribe is used

- Transport component of EDA architectures
- Data Distribution Services
- Transport component of context aware distributed applications

- **EDA characteristics:** three principal building blocks
  - **Sense:** The sensing block gathers data from within and outside the controlled system
  - **Analyse**: Data are analyzed
  - **Respond**: Analysis used to determine whether appropriate actions are to be timely undertaken in response to what has been sensed (respond block)



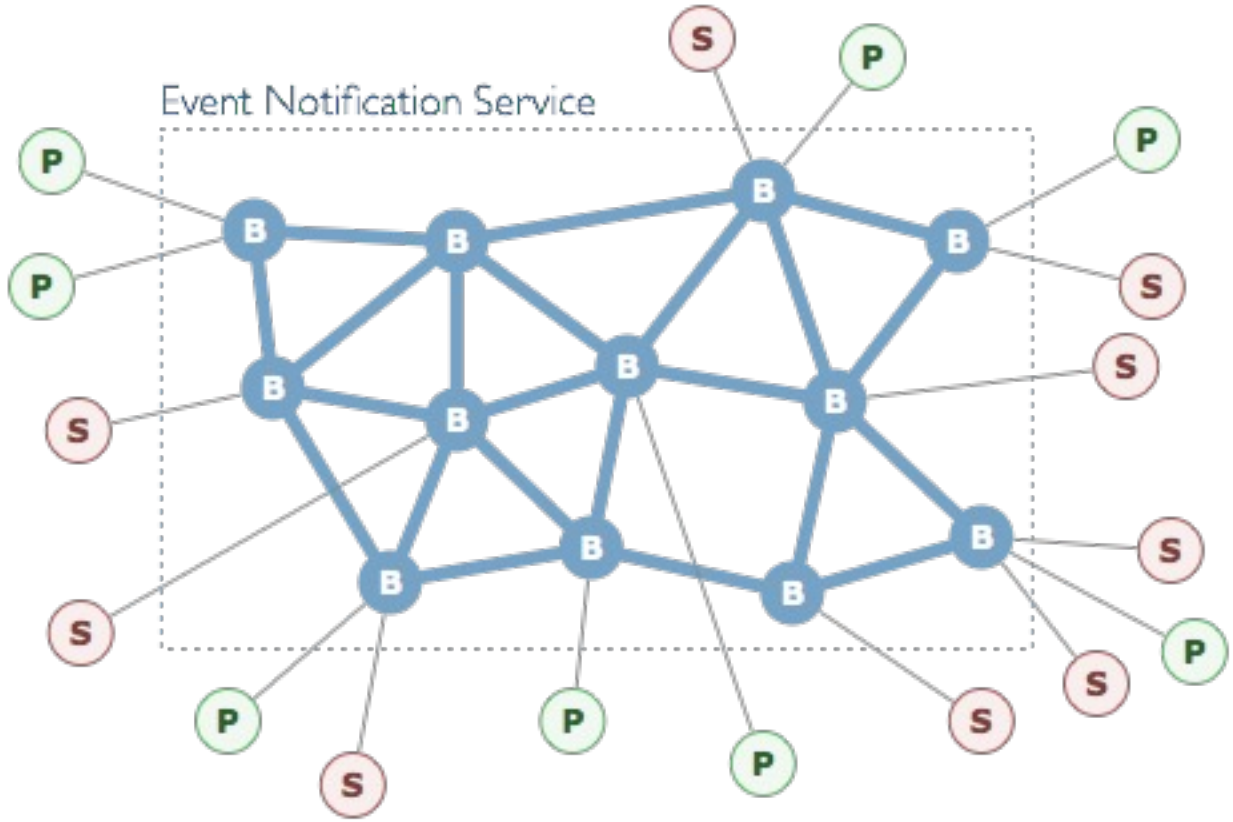
- A control loop is enabled
  - The sensing part obtains data that defines the "real" state
  - The analysis part correlates data in order to determine the current state
  - When reality deviates from that expected then the respond part acts on the system (e.g, sends alerts)

- **EDA Architecture**
  - Event Correlation and Processing
  - Event Distribution
    - Publish subscribe help in doing some pre-processing of the events as well as reducing the network load

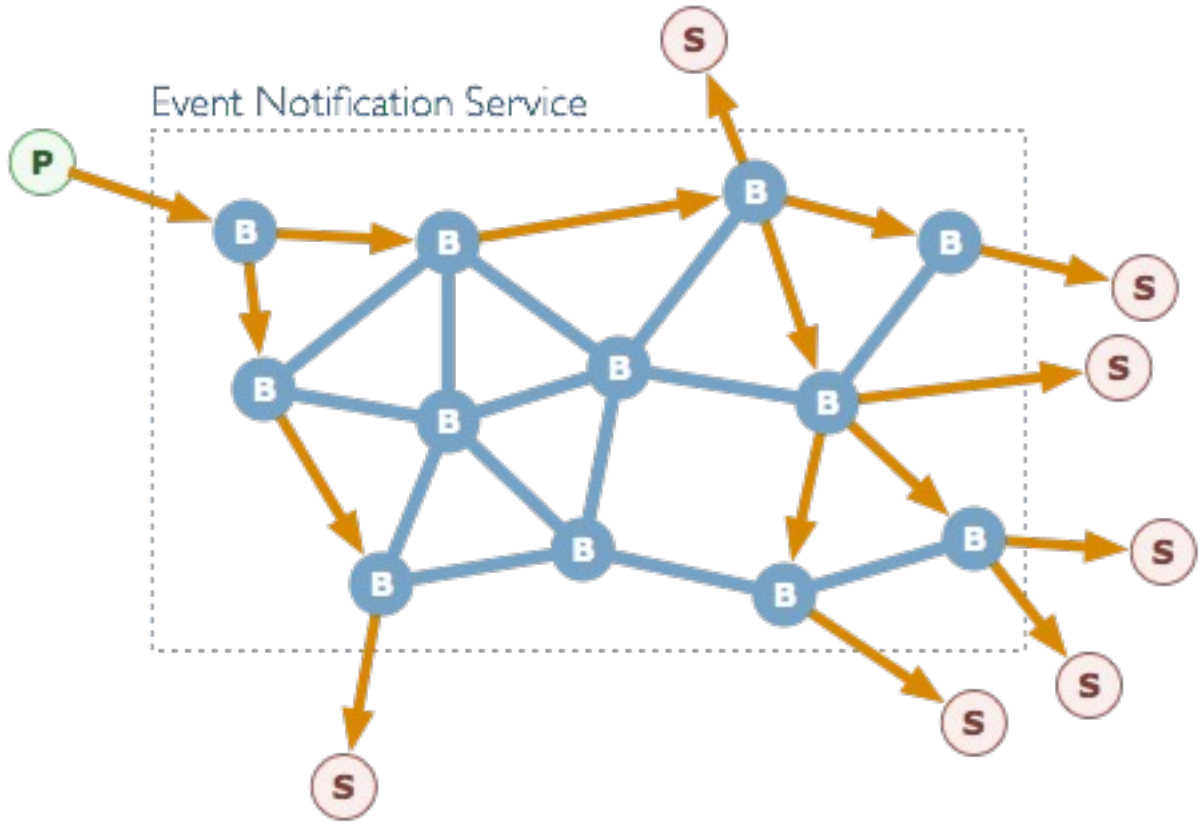
Complex Event Processing

Event Dissemination (Publish-Subscribe)

- The Event Notification Service is usually implemented as a:
  - **Centralized service**: the ENS is implemented on a single server.
  - **Distributed service**: the ENS is constituted by a set of nodes, event brokers, which cooperate to implement the service.
- The latter is usually preferred for large settings where scalability is a fundamental issue.

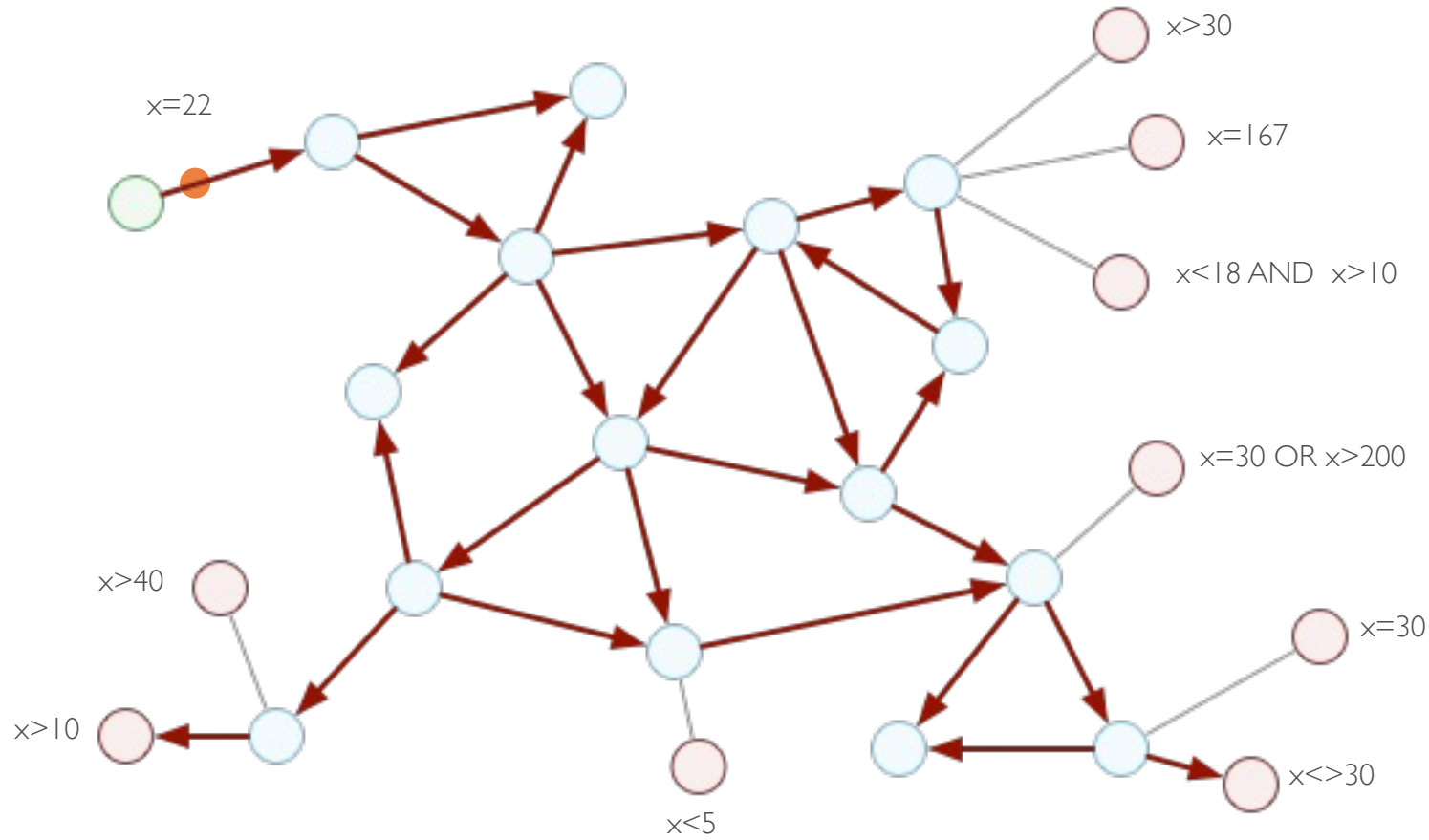


- Modern ENSs are implemented through a set of processes, called *event brokers,* forming an overlay network.
- Each client (publisher or subscriber) accesses the service through a broker that masks the system complexity.

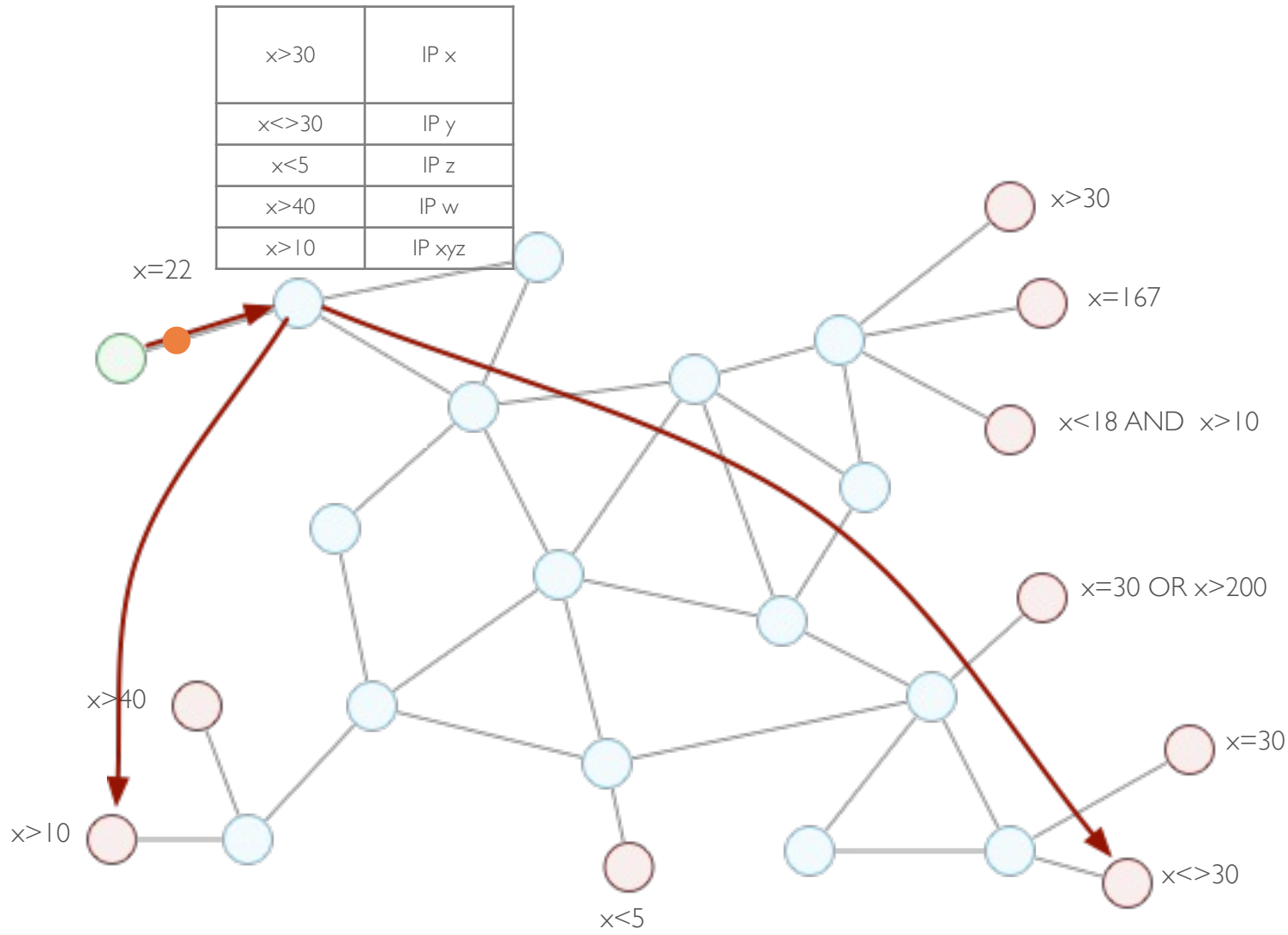


- An **event routing** mechanism routes each event inside the ENS from the broker where it is published to the broker(s) where it must be notified.

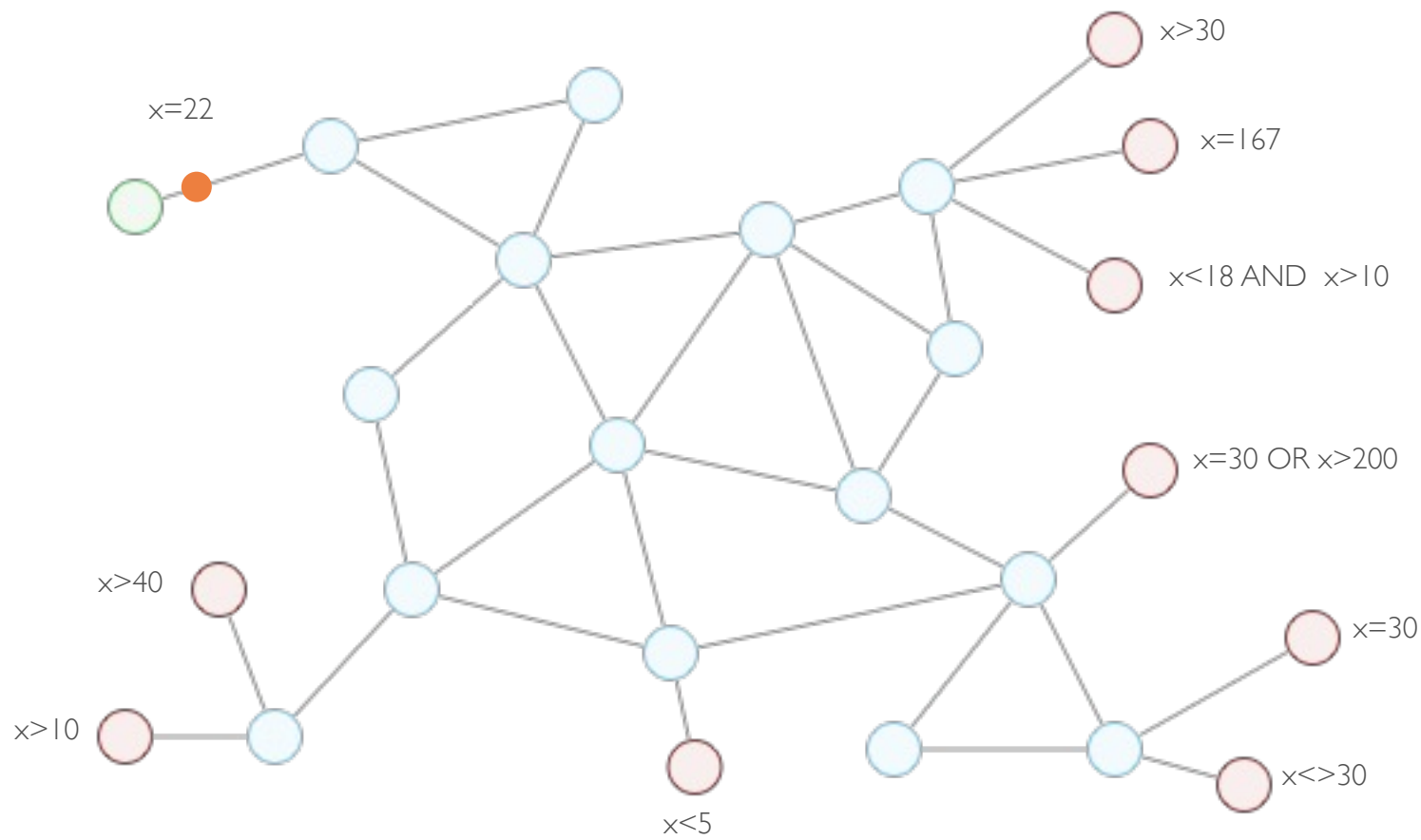
- **Event flooding**: each event is broadcast from the publisher in the whole system.
- The implementation is straightforward but very expensive.
- This solution has the highest message overhead with no memory overhead.



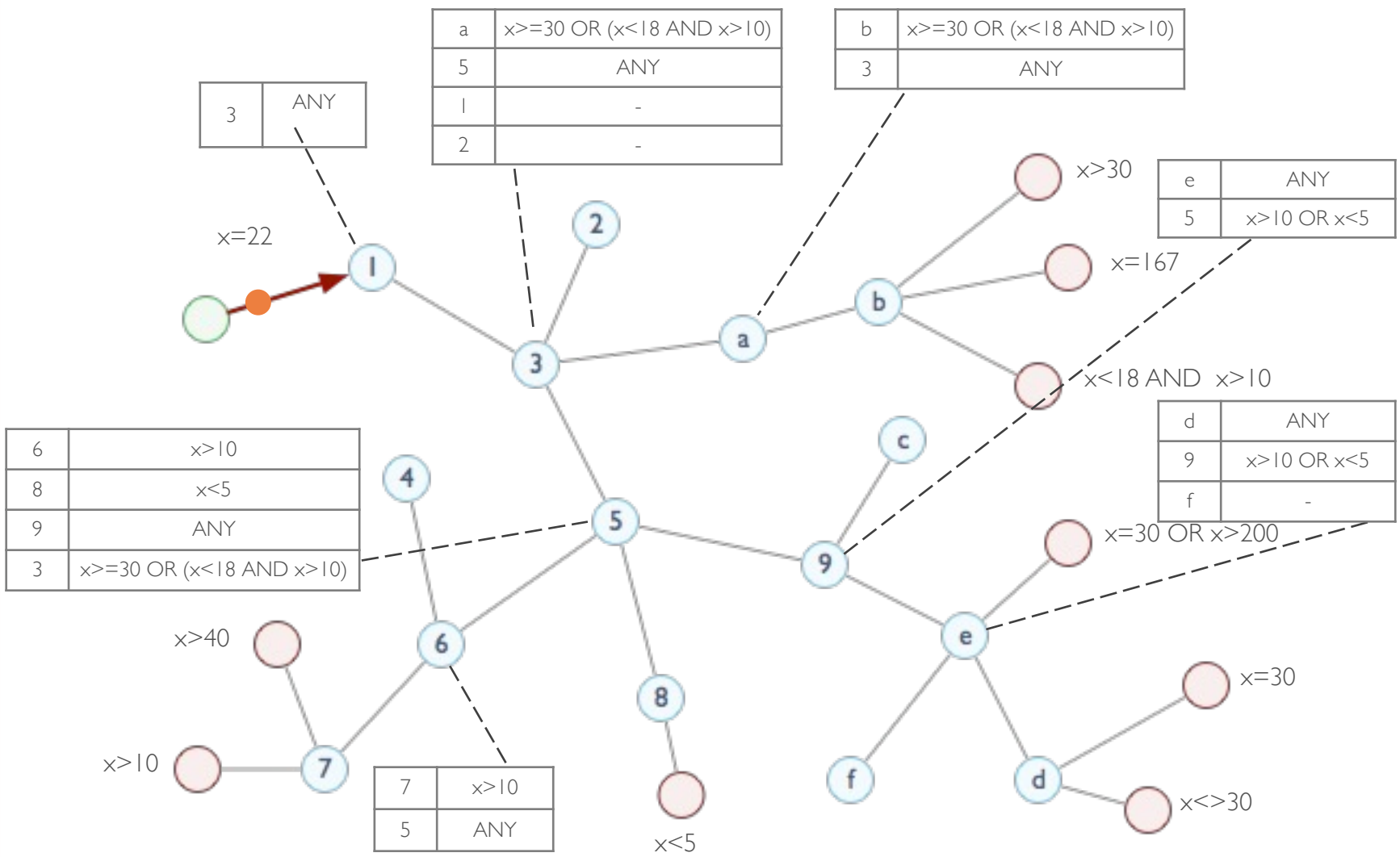
■**Subscription flooding**: each subscription is copied on every broker, in order to build locally complete subscription tables. These tables are then used to locally match events and directly notify interested subscribers. This approach suffers from a large memory overhead, but event diffusion is optimal. It is impractical in applications where subscriptions change frequently.



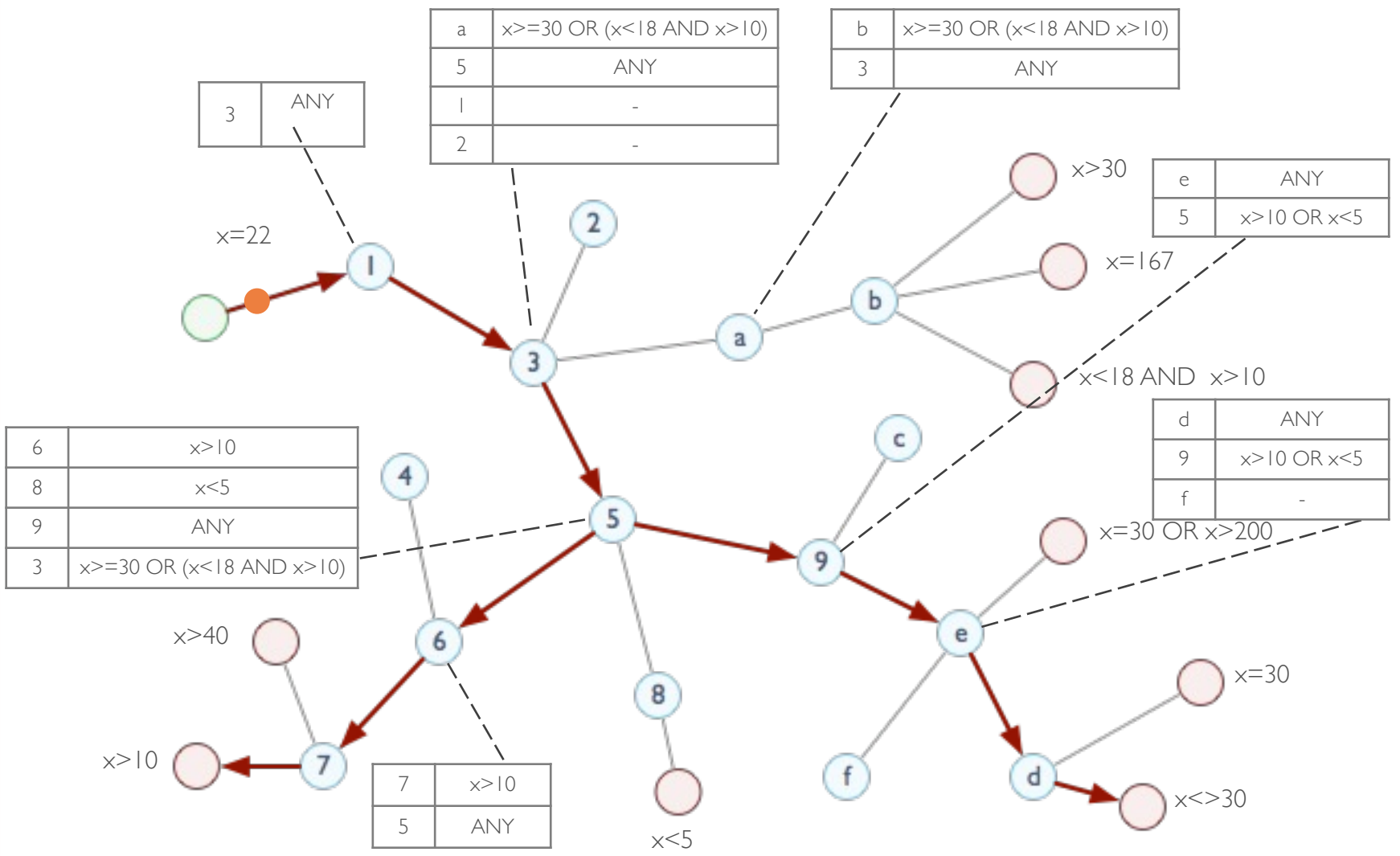
Filter-based routing: subscriptions are partially diffused in the system and used to build *routing tables. These tables, are then exploited during event diffusion to dynamically build a multicast tree that (hopefully) connects the publisher to all, and only, the interested subscribers.*

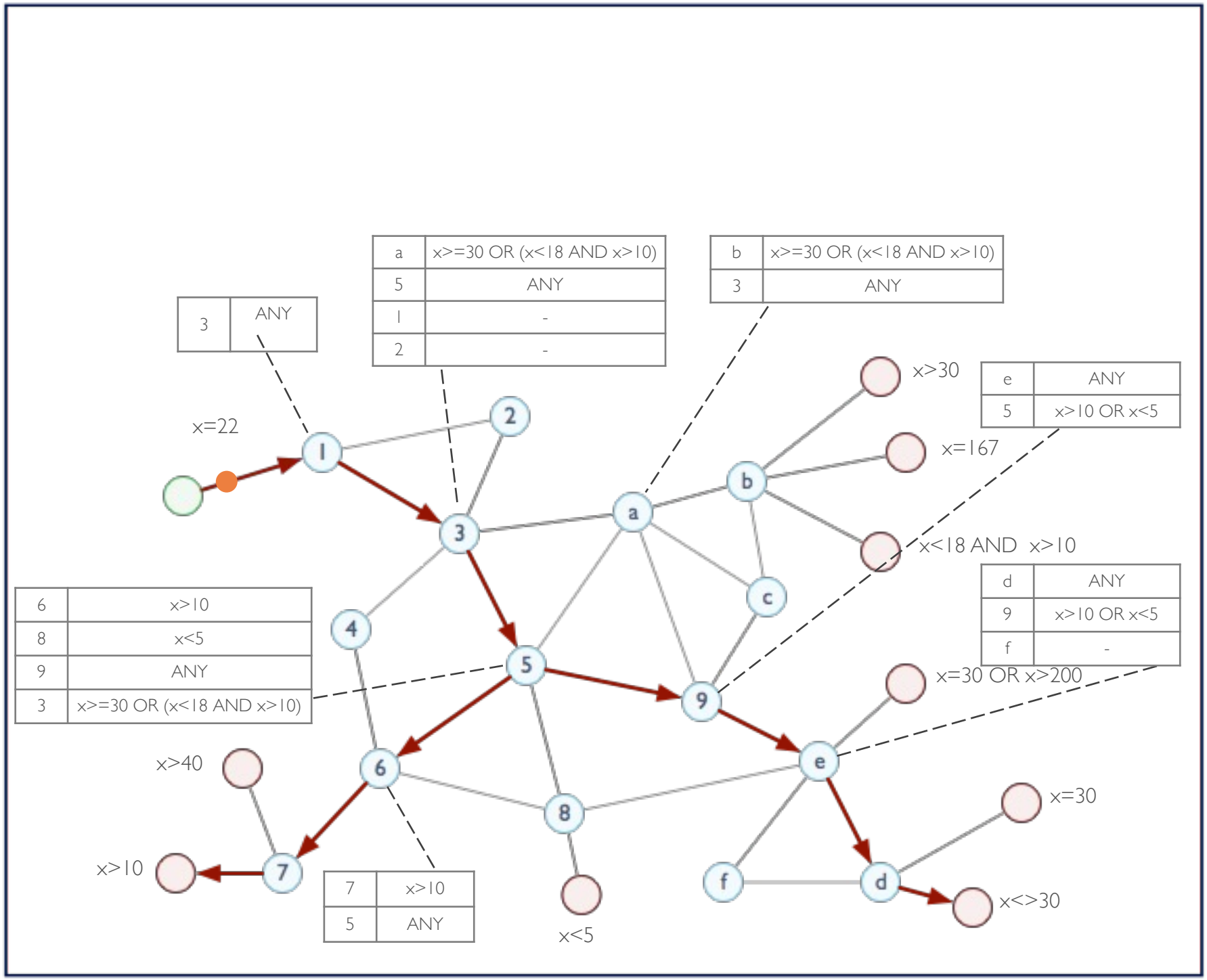


Filter-based routing: subscriptions are partially diffused in the system and used to build *routing tables. These tables, are then exploited during event diffusion to dynamically build a multicast tree that (hopefully) connects the publisher to all, and only, the interested subscribers.*



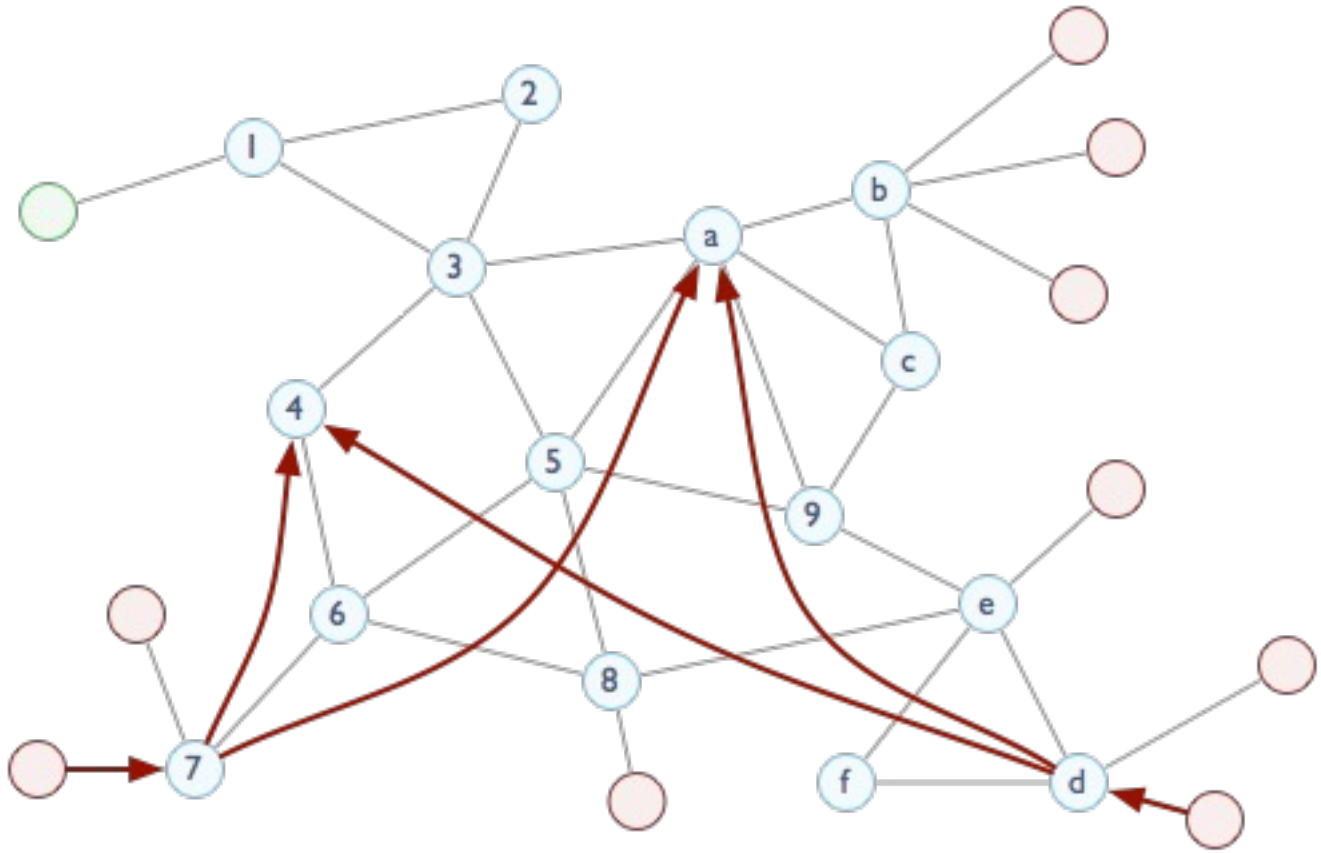
Filter-based routing: subscriptions are partially diffused in the system and used to build *routing tables. These tables, are then exploited during event diffusion to dynamically build a multicast tree that (hopefully) connects the publisher to all, and only, the interested subscribers.*





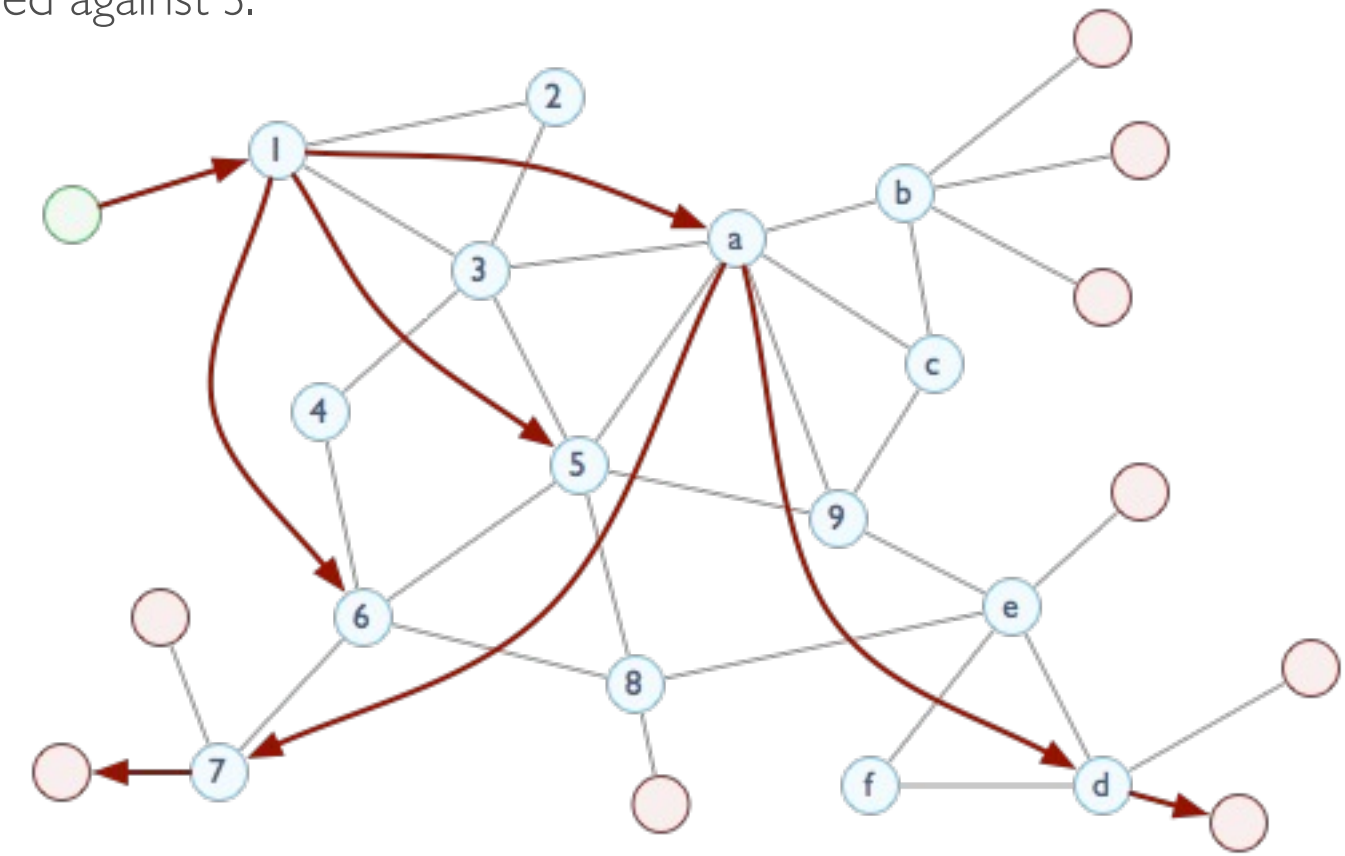
- **Rendez-Vous routing**: it is based on two functions, namely *SN* and *EN*, used to associate respectively subscriptions and events to brokers in the system.
- Given a subscription *s*, *SN(s)* returns a set of nodes which are responsible for storing *s* and forwarding received events matching *s* to all those subscribers that subscribed it.
- Given an event *e*, *EN(e)* returns a set of nodes which must receive *e* to match it against the subscriptions they store.
- Event routing is a two-phases process: first an event *e* is sent to all brokers returned by *EN(e)*, then those brokers match it against the subscriptions they store and notify the corresponding subscribers.
- This approach works only if for each subscription *s* and event *e*, such that *e* matches *s*, the intersection between *EN(e)* and *SN(s)* is not empty (*mapping intersection rule*).

- **Rendez-Vous routing**: example.
- Phase 1: two nodes issue the same subscription *S*.



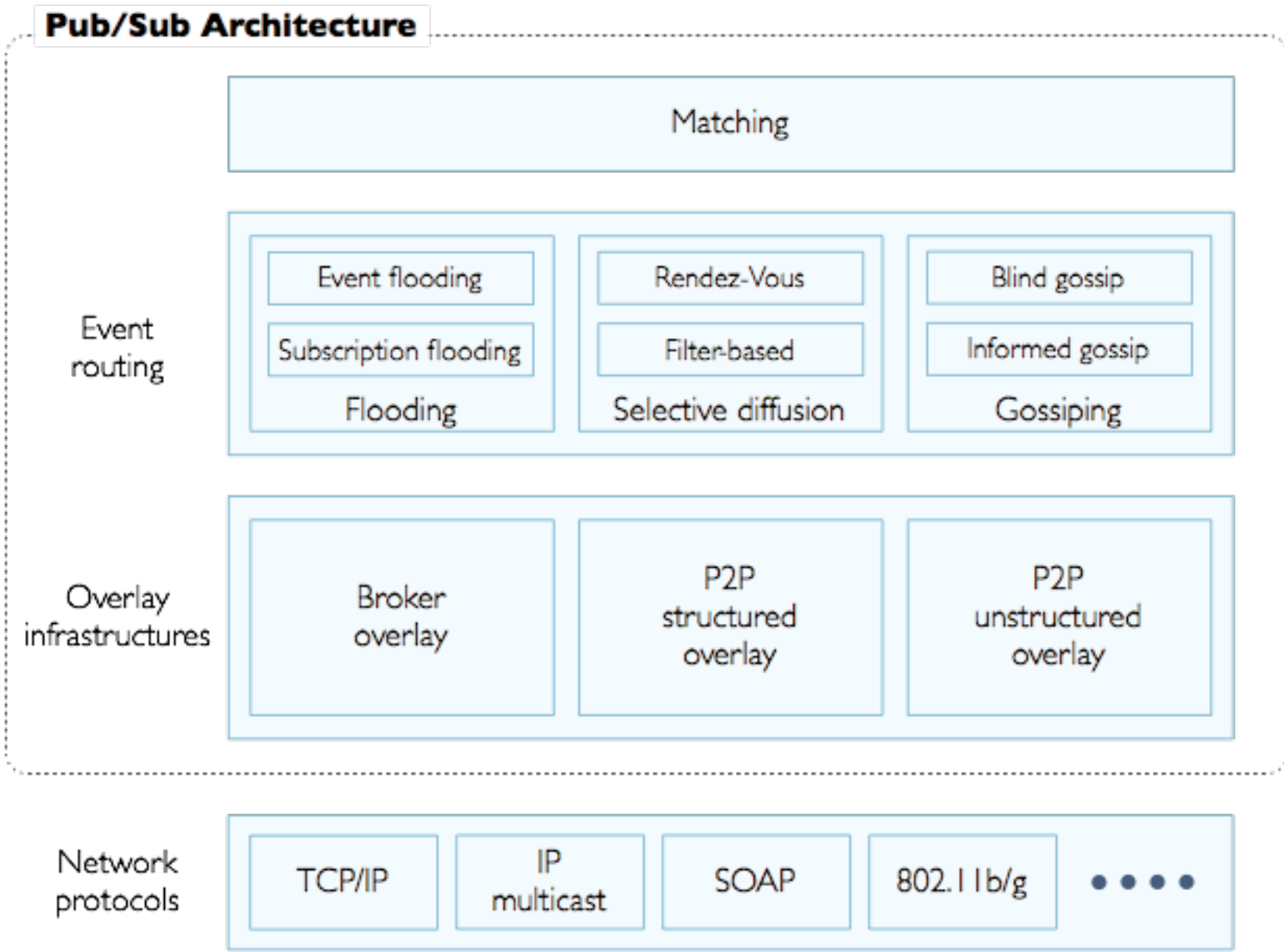
- SN(S) = {4,a}

- **Rendez-Vous routing**: example.
- Phase 1I: an event e matching *S* is routed toward the rendez-vous node where it is matched against *S*.



- EN(e) = {5,6,a}
- Broker **a** is the rendez-vous point between event e and subscription *S*.

- A generic architecture of a publish/subscribe system:



From "Distributed Event Routing in Publish/Subscribe Communication Systems: a survey" R.Baldoni, L. Querzoni, S. Takoma, A. Virgillito