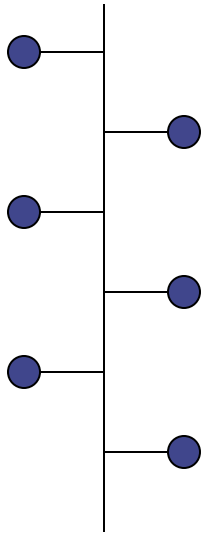
TechGI IV - ComDiS

ComDis

Introduction to
Communication Networks
and **Dis**tributed Systems



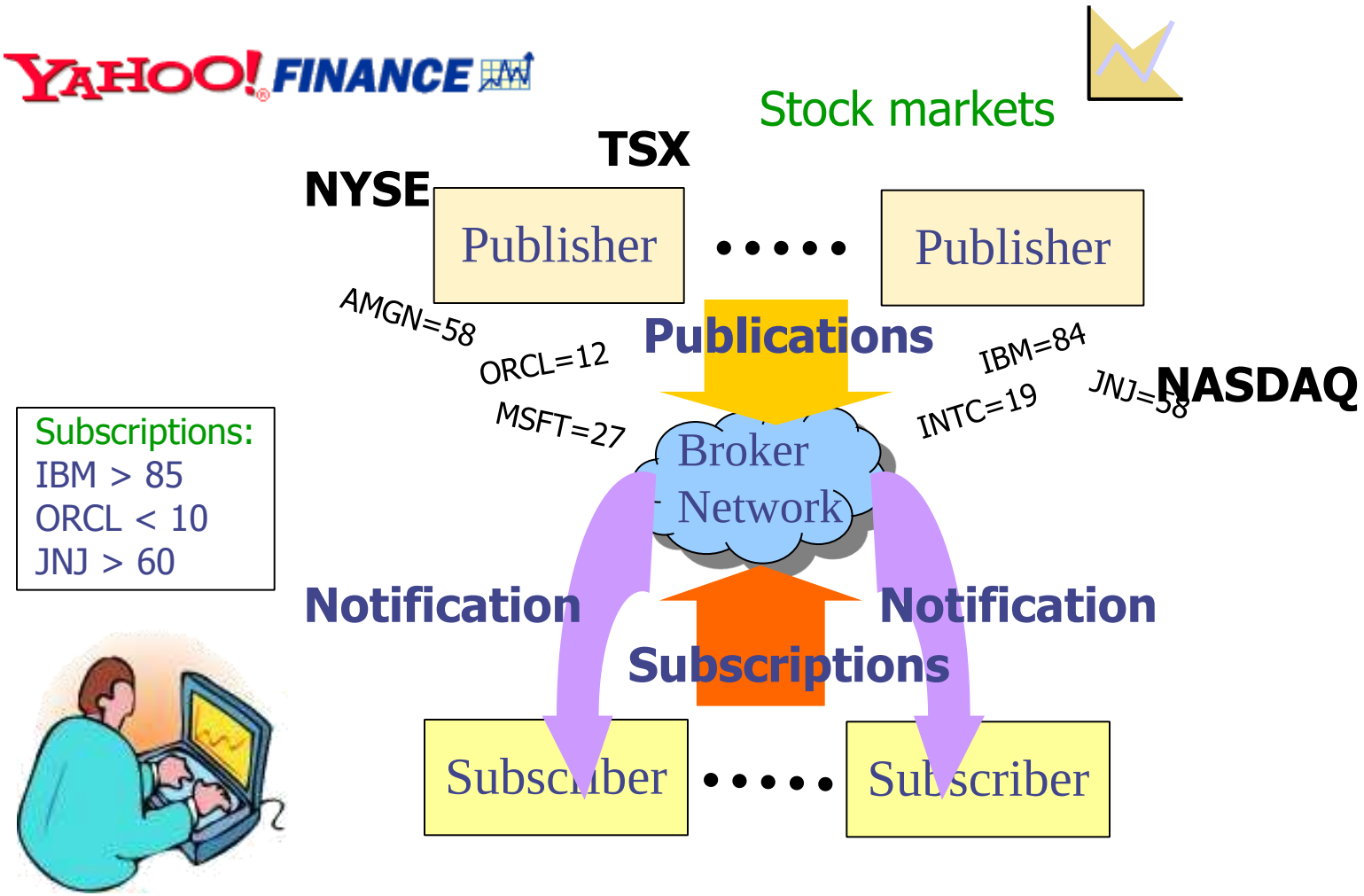
Complex Communication Patterns

Prof. Dr.-Ing. Adam Wolisz

TKN Telecommunication Networks Group

- **Advanced Communication architectures:**
  - **Publish subscribe**
  - **(Virtual) Shared memory**

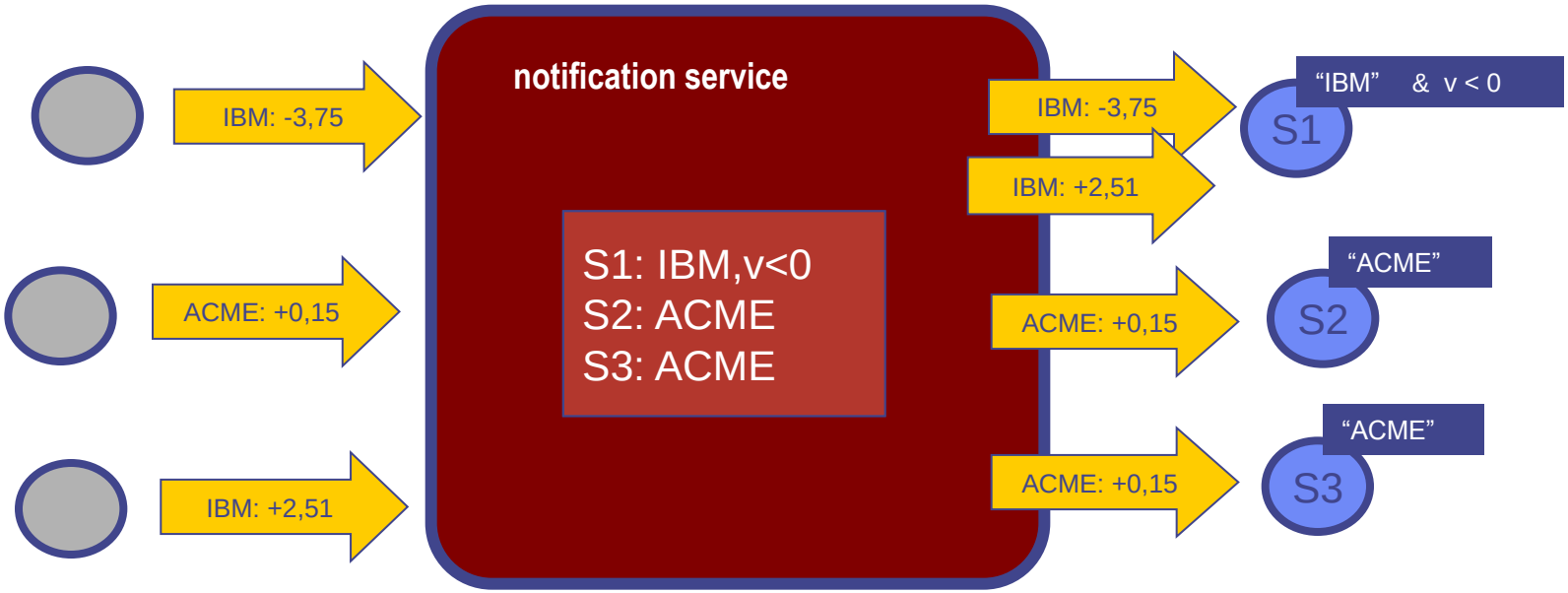
Publish/Subscribe Model



Introduction The Publish/Subscribe Paradigm [baldoni]

- Publish/Subscribe (pub/sub): a powerful abstraction for building distributed applications
  - Message-based, anonymous communication
  - Participants are decoupled
    - in space: no need to be connected or even know each other
    - in flow: no need to be synchronized
    - in time: no need to be up at the same time
- Many names for pub/sub systems: notification service, data distribution service
- Applications
  - Stock information delivery
  - Auction system
  - Air traffic control
  - .etc…

Subscription Models



Topic-Based [Oki et al. 93]:
- events are divided in topics
- subscribers subscribe for a single topic

Type-Based [Eugster 2001]:
- notifications are objects
- type is the discriminating attribute

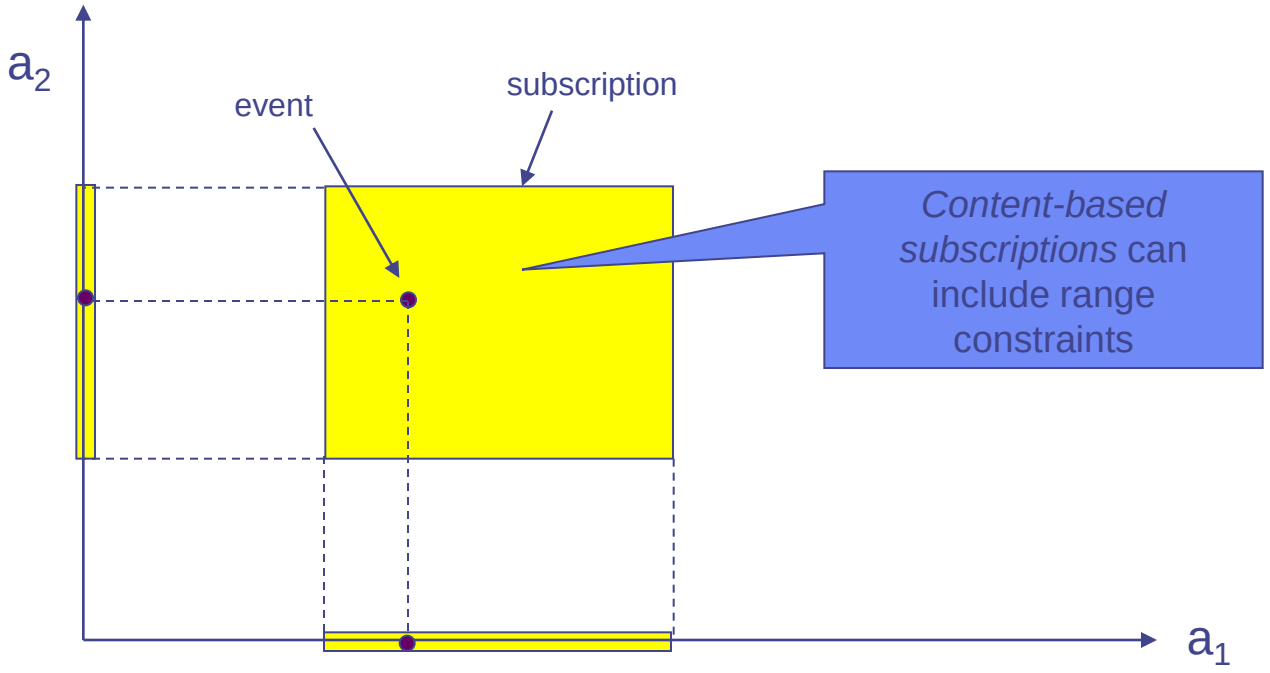
Content-Based [Carzaniga et al 2001]:
- subscriptions are generic queries SQL-like on the event schema

Pub/Sub Variants: Topic-based

- Event space is divided in topics, corresponding to logical channels
- Participants subscribe for a topic and publish on a topic
- Receivers for an event are known a priori
- Channel = Group
  - Therefore often exploit network-level multicast
  - Group communication

Content Based pub-sub: event schema

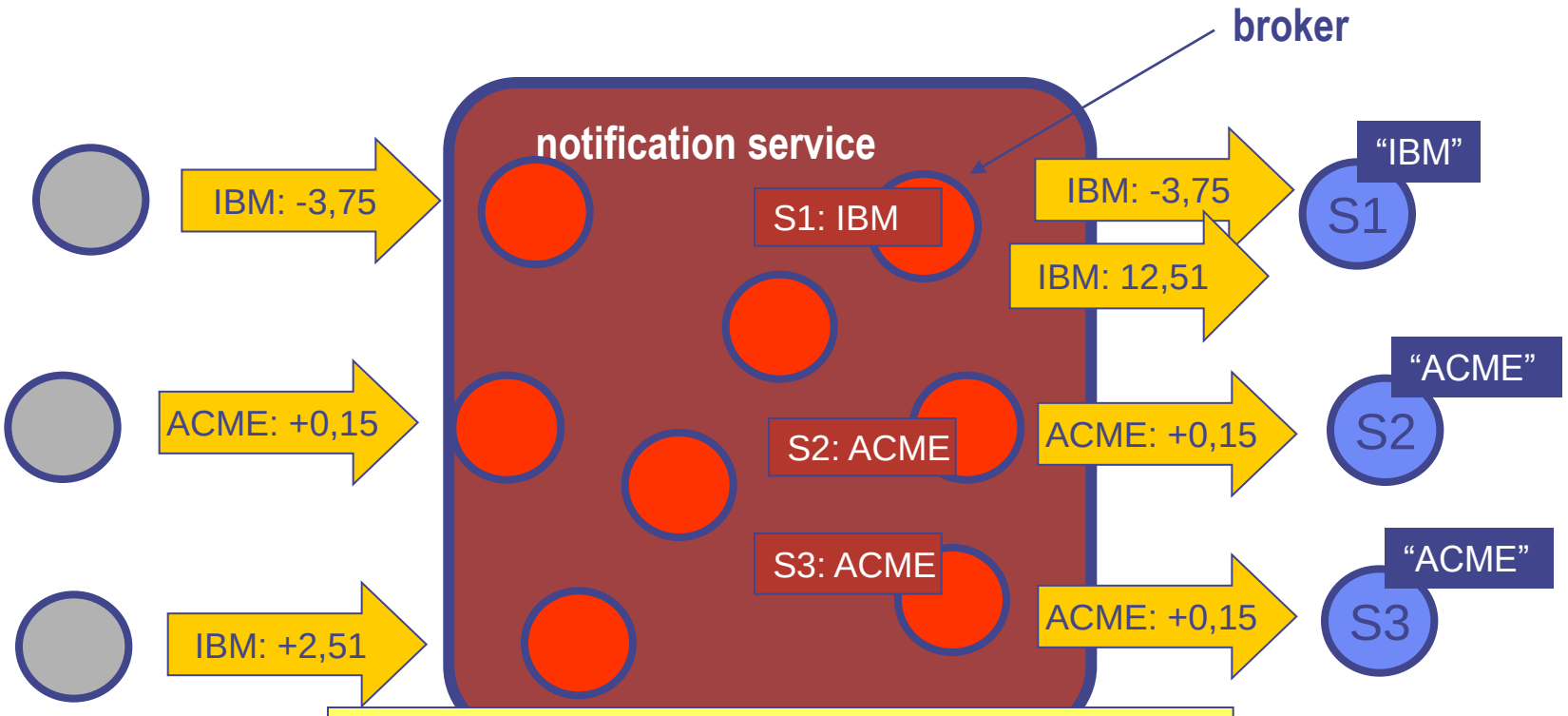
- Subscriptions and events defined over an n-dimensional *event space (*E.g. `StockName = "ACME" and change < -3`)
  - Subscription: conjunction of constraints



Topic vs. Content

- Topic-based pub/sub
  - Recipients are known a-priori
  - Many efficient implementations exist: e.g. tib
  - Limited expressiveness
- Content-based pub/sub
  - Cannot determine recipients before publication
  - More flexible
  - More general
  - Much more difficult to implement efficiently

Distributed Notification Service



Matching

Determining the recipients for an event

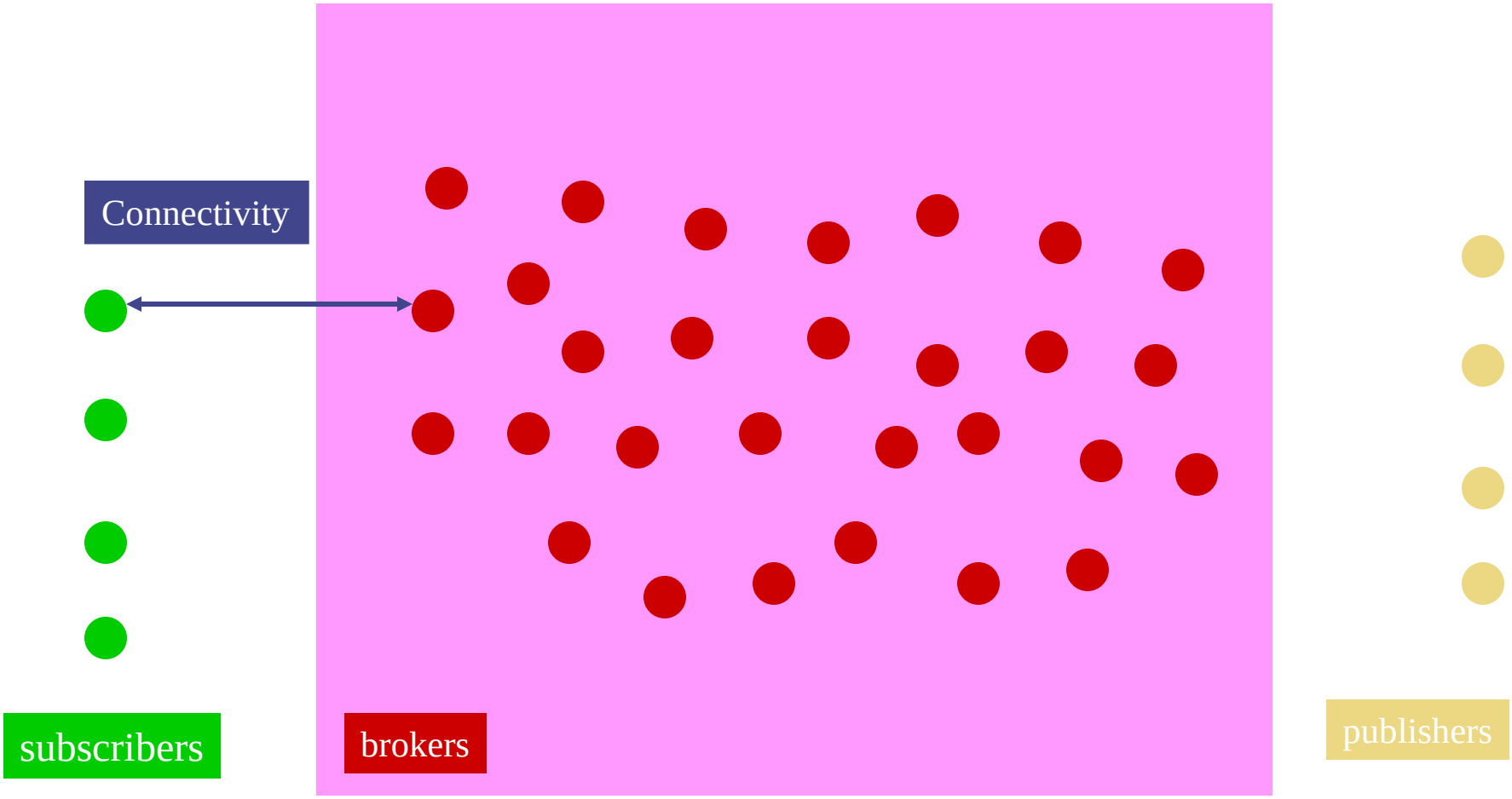
Routing

Delivering a notification to all the recipients

EVENT vs. SUBSCRIPTION ROUTING

- Extreme solutions
- Sol 1 (event flooding)
  - flooding of events in the notification event box
  - each subscription stored only in one place within the notification event box
  - Matching operations equal to the number of brokers
- Sol 2 (subscription flooding)
  - each subscription stored at any place within the notification event box
  - each event matched directly at the broker where the event enters the notification event box

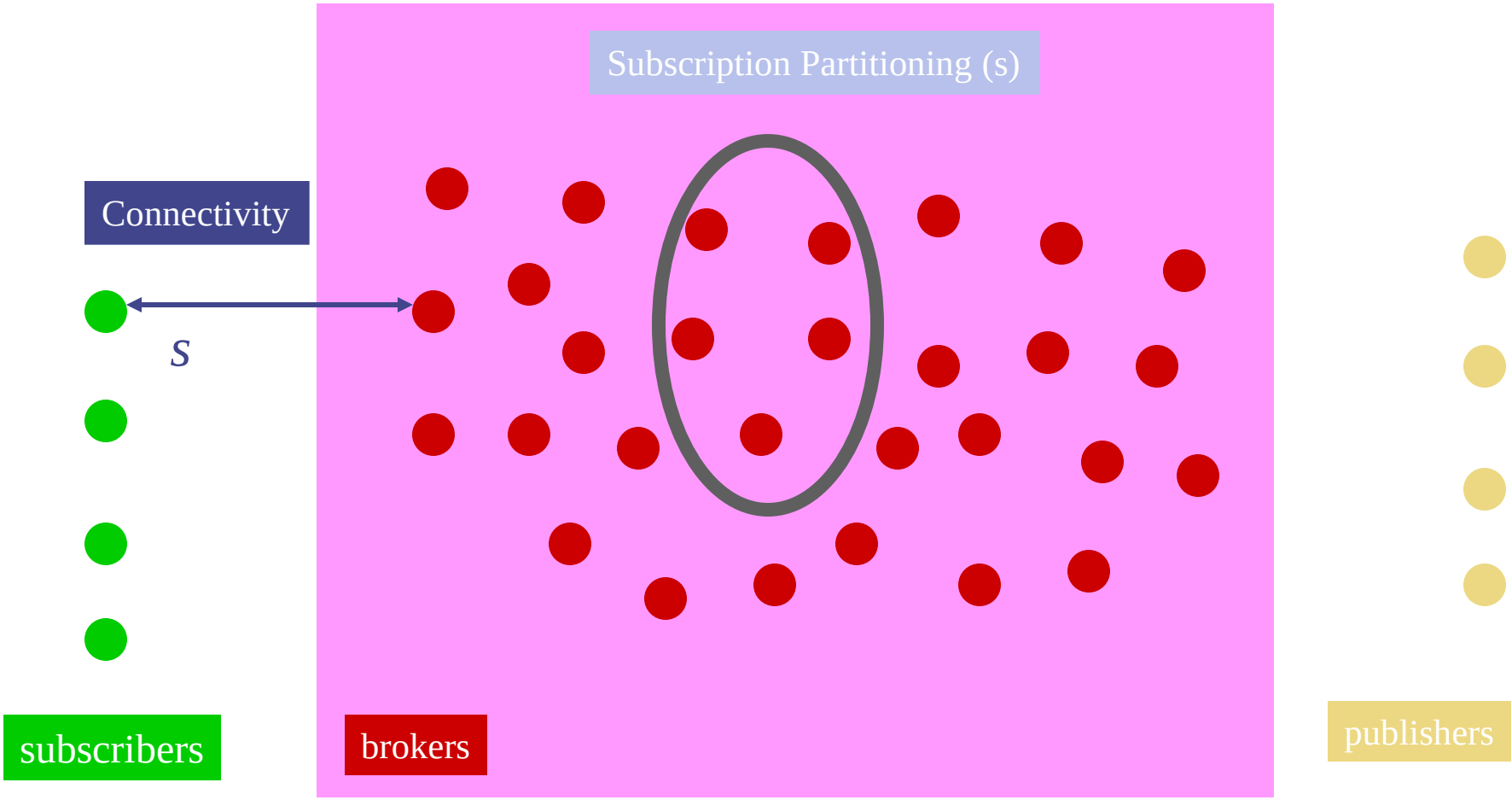
Intermediate routing solutions



Handling subscriptions

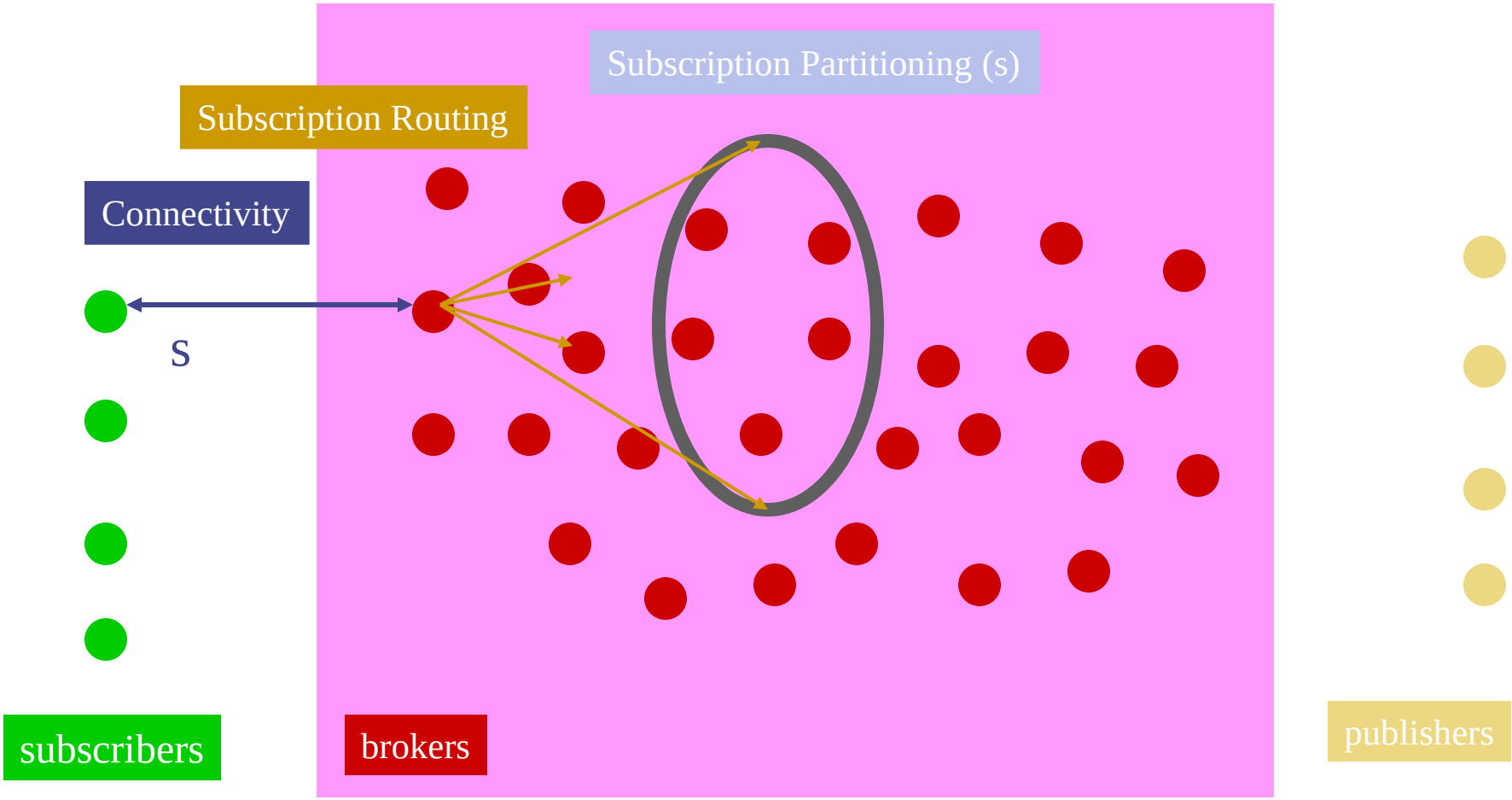
Intermediate routing solutions

Handling subscriptions: "subscription partitioning" i.e., "where to store *s*"



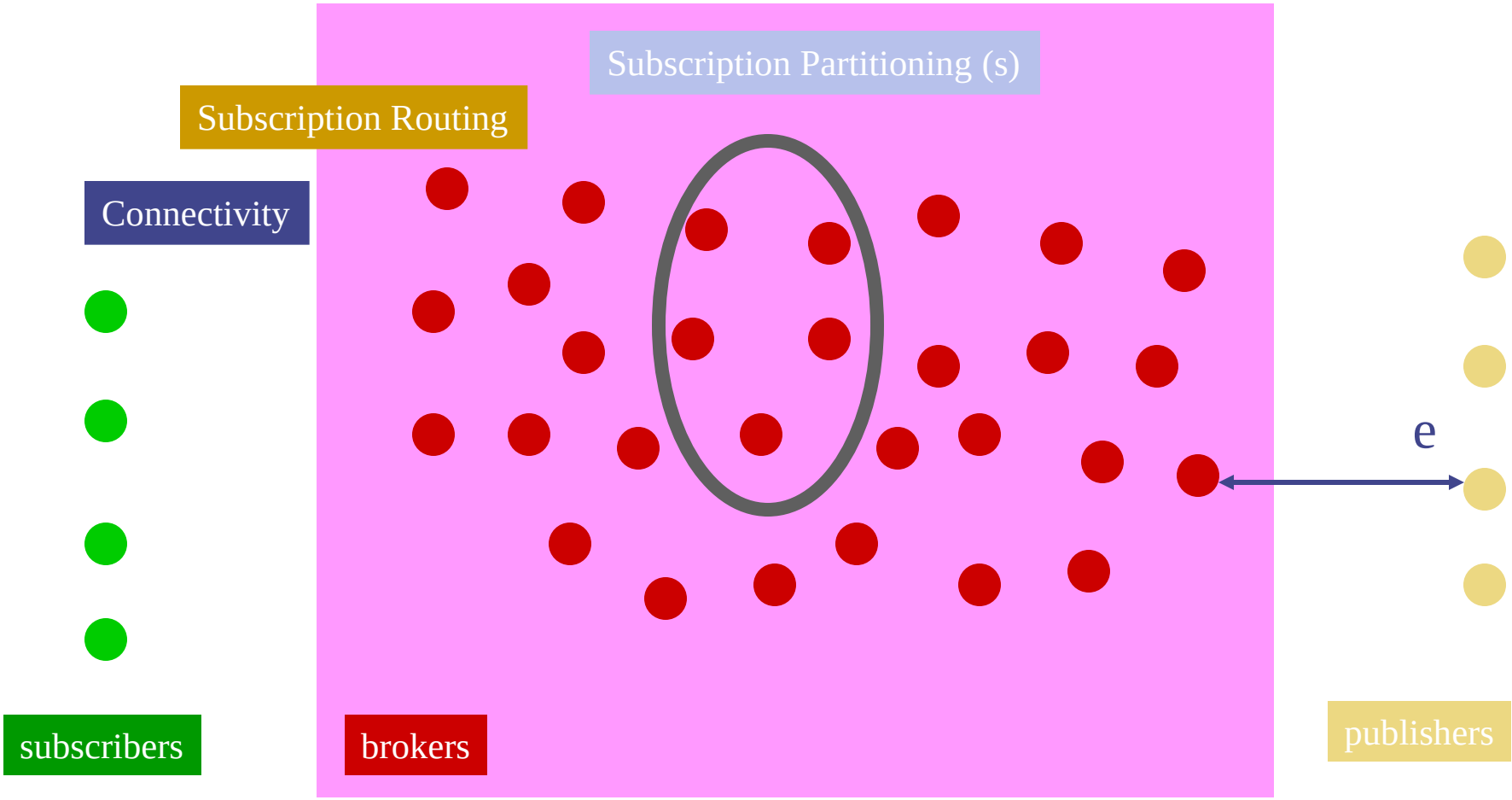
Intermediate routing solutions

Handling subscriptions: "subscription routing" i.e., "how to bring *s* there"



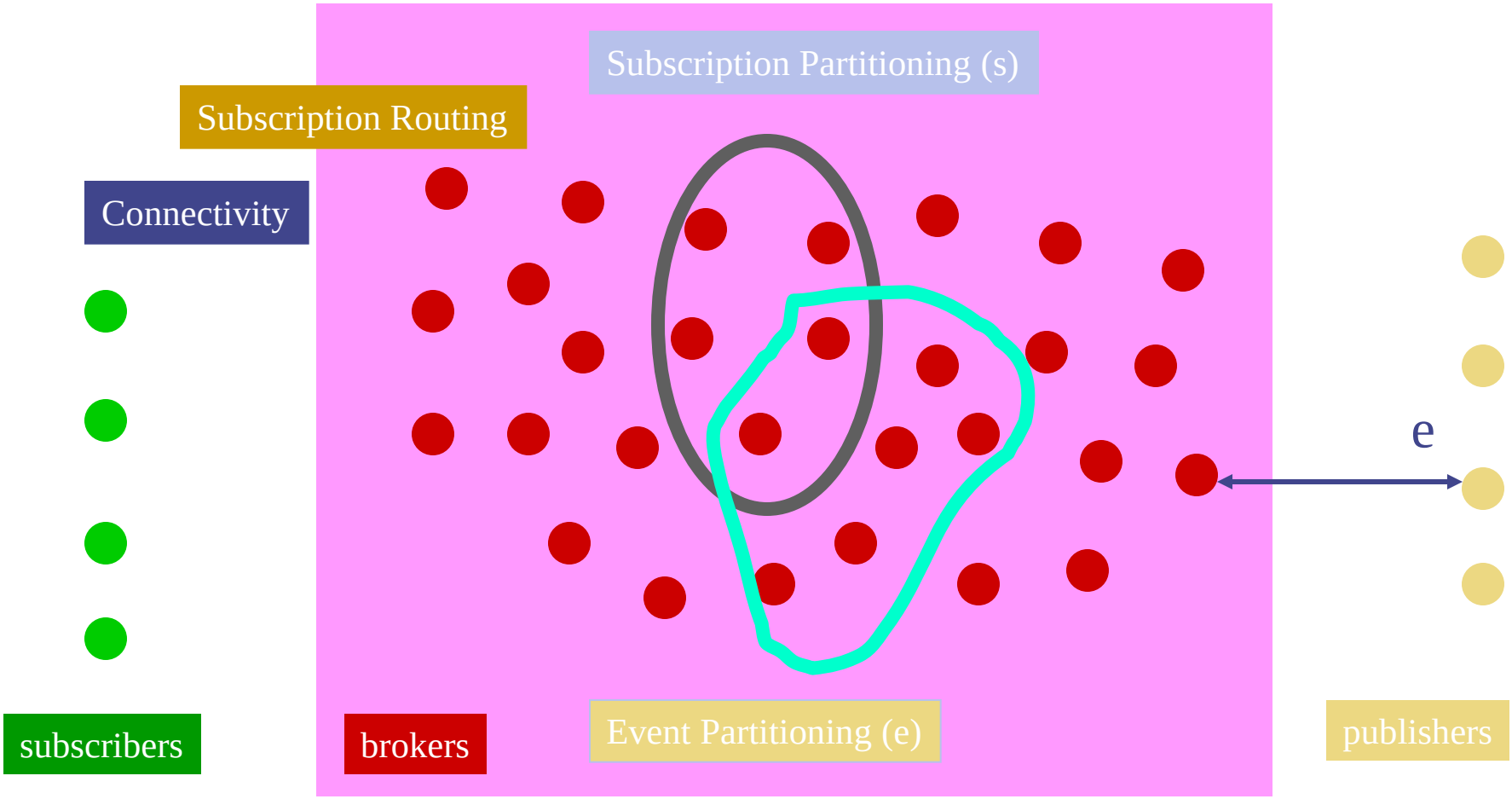
Intermediate routing solutions

Handling events



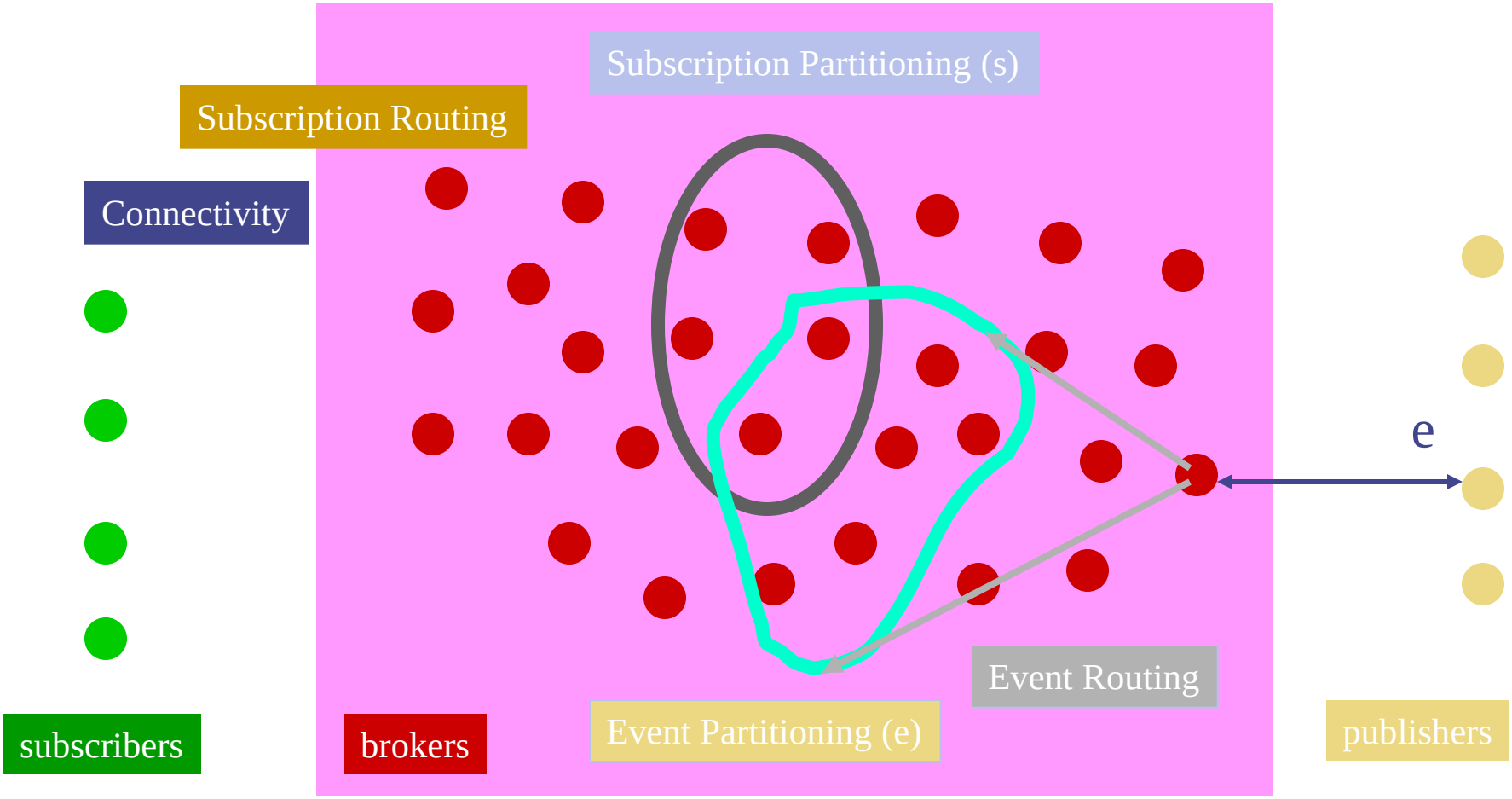
Intermediate routing solutions

Handling events: "Event Partitioning" i.e., "where to match *e*"



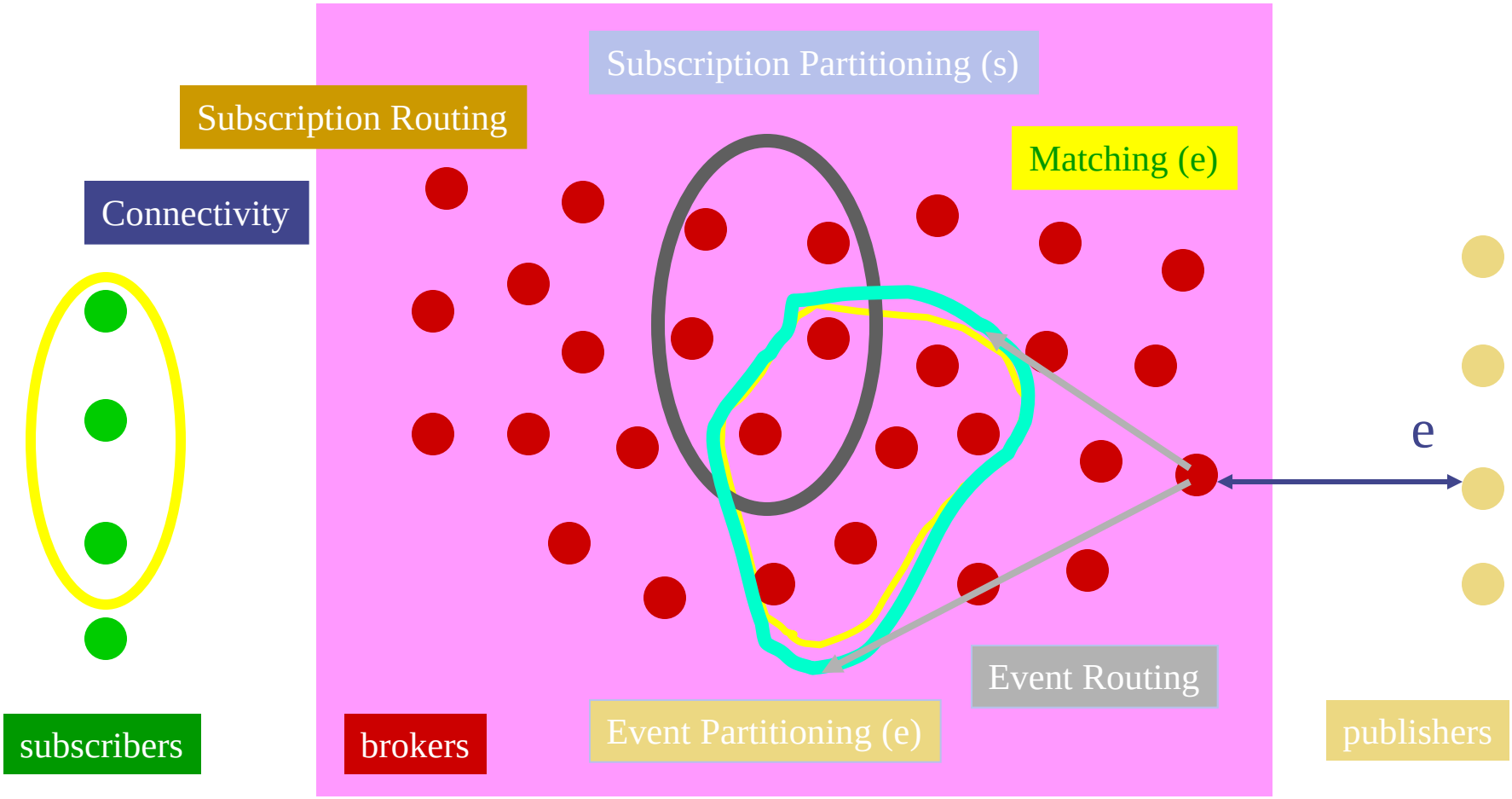
Intermediate routing solutions

Handling events: "Event Routing" i.e., "How to bring *e* there"



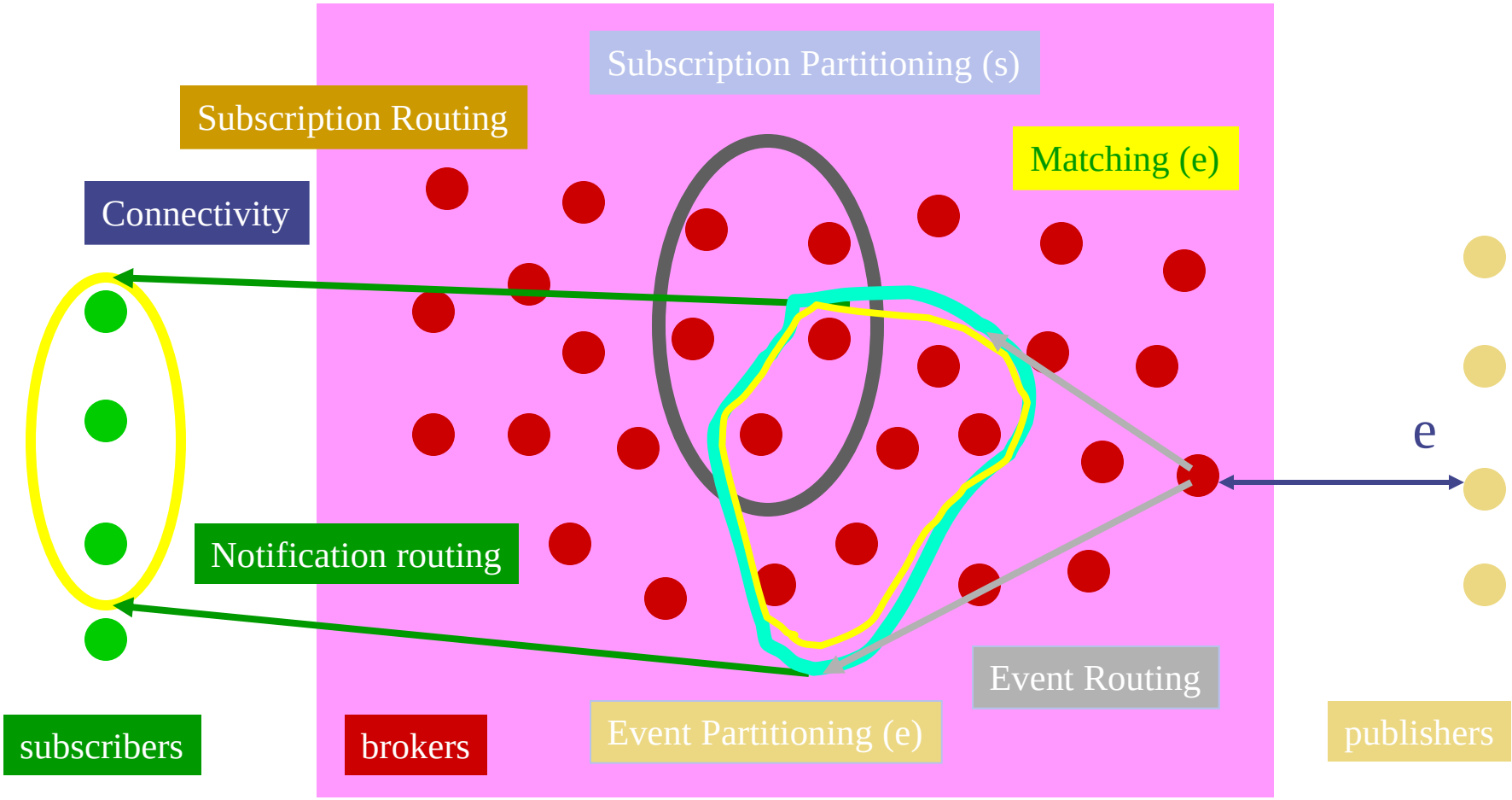
Intermediate routing solutions

Handling events: "Event Matching" i.e., "define the set of recipients of *e*"

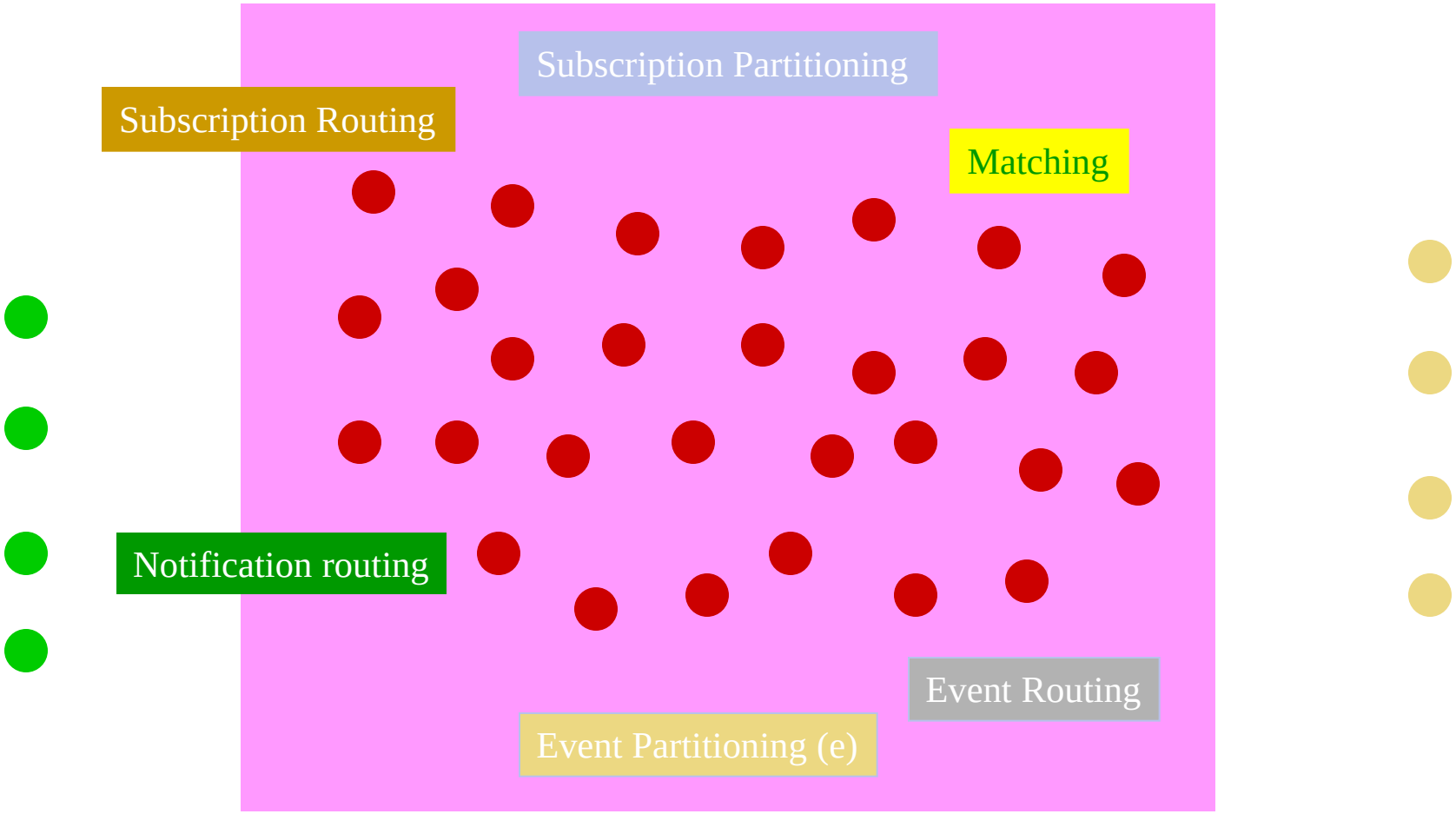


Intermediate routing solutions

Handling events: "Notification Routing" i.e., "bring *e* to the destinations"

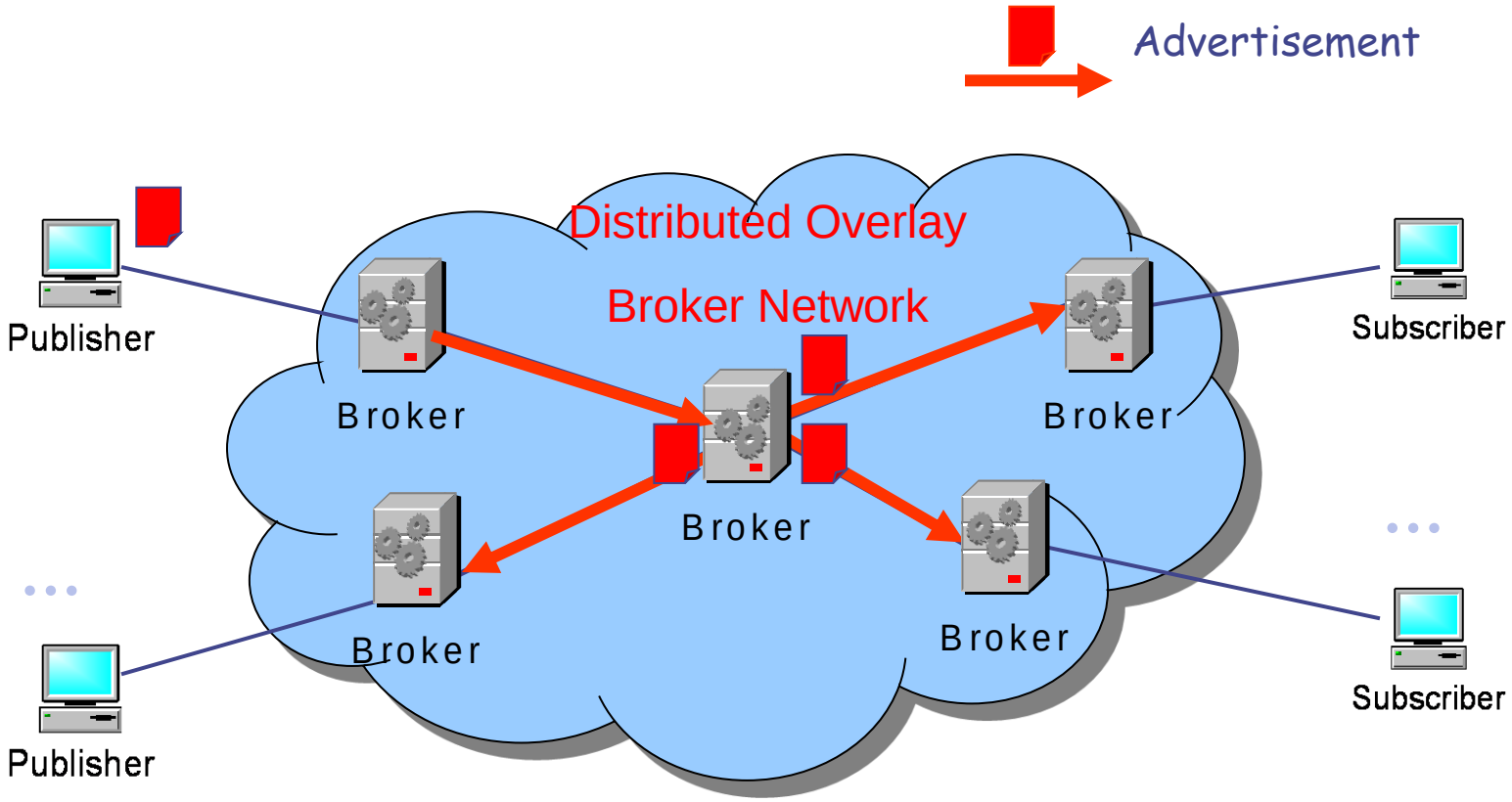


Intermediate routing solutions



Content-based Routing - example PADRES

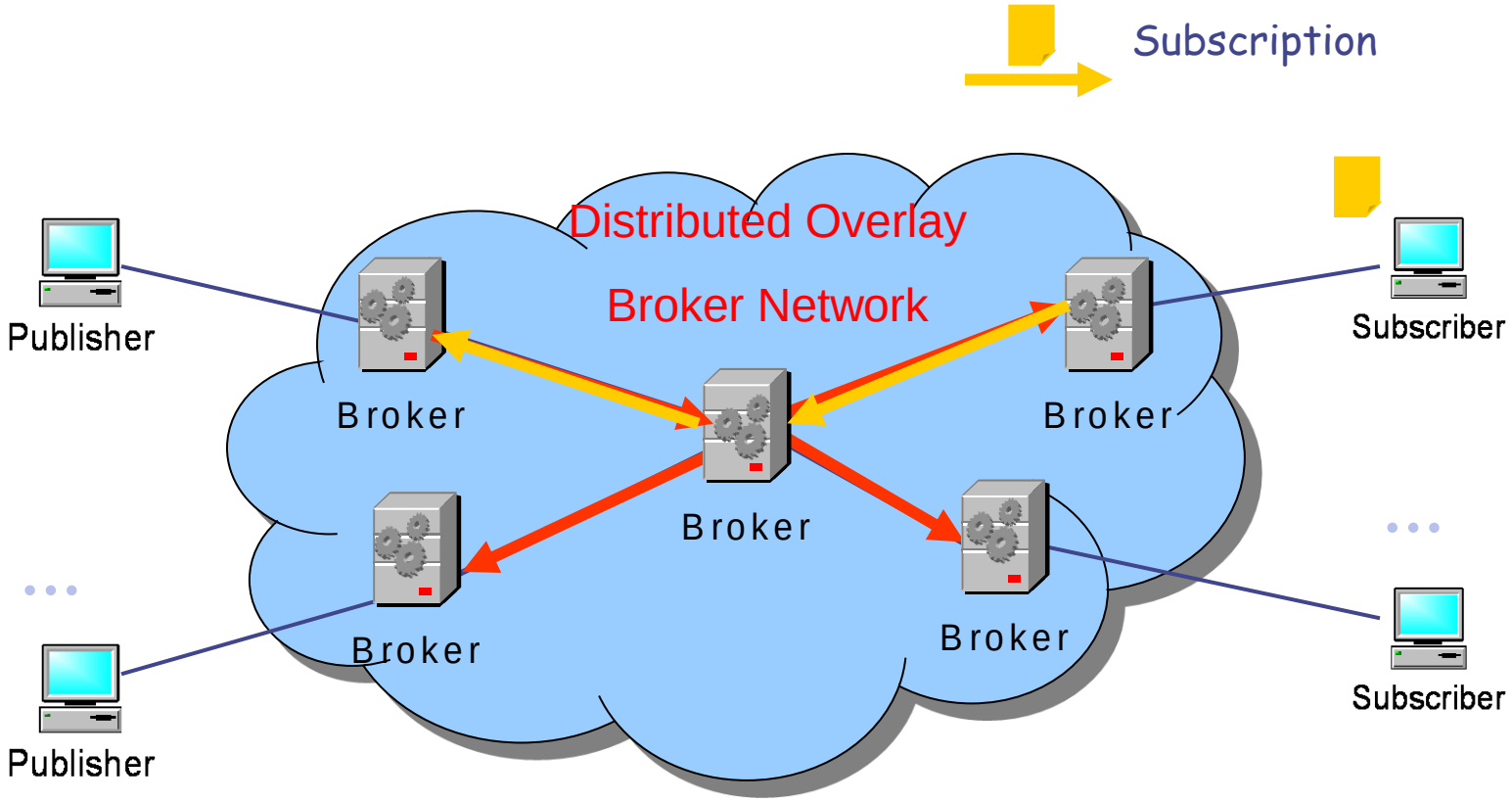
- Advertising



***Adopted from SIENA and REBECA**

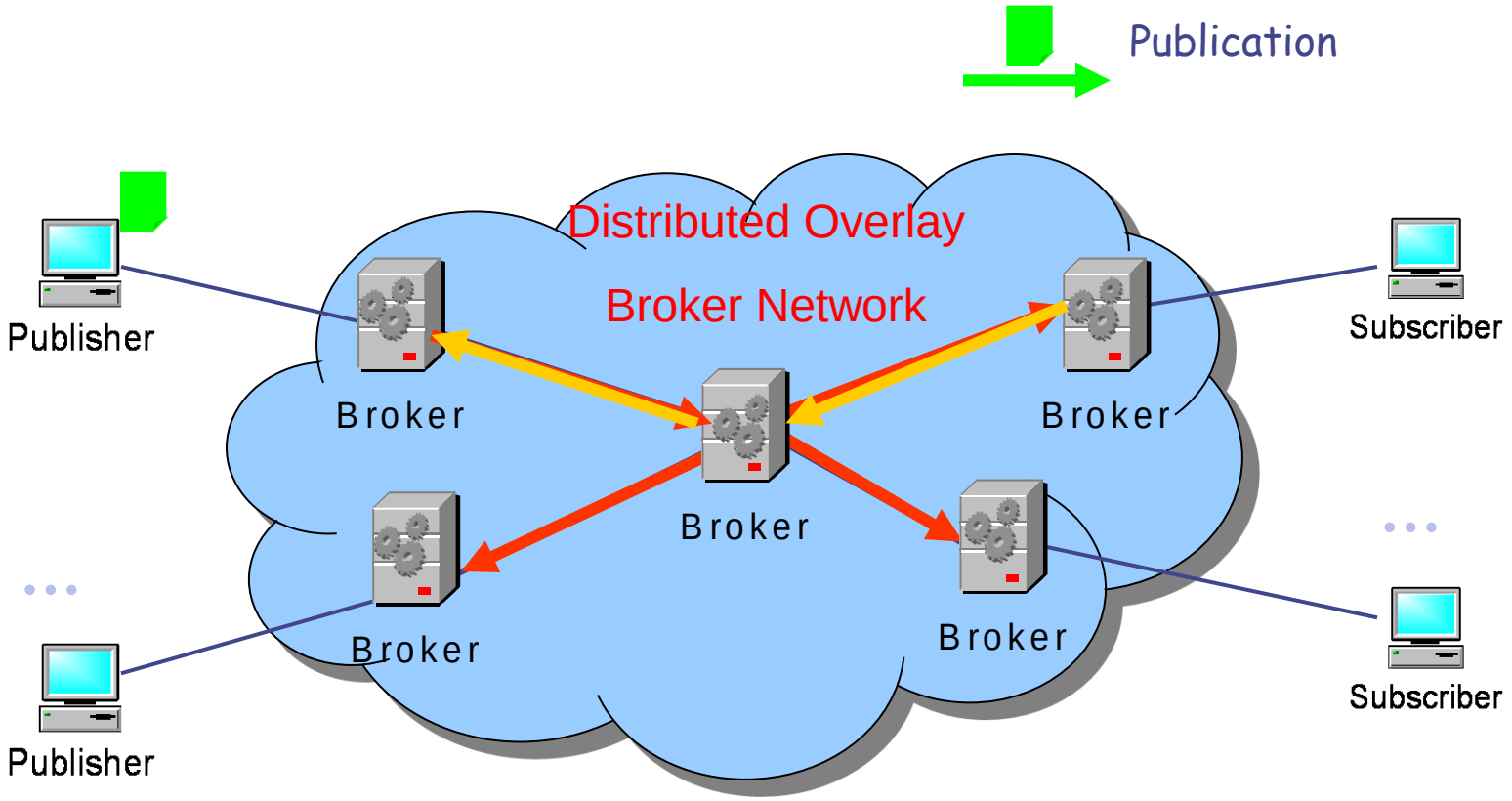
Content-based Routing - example: PADRES

- Subscribing



Content-based Routing - example: PADRES

- Publishing



***Adopted from SIENA and REBECA**

Distributed Shared Memory

[neumann]

- Another abstraction for communication….
- Motivated by development of shared-memory multiprocessors which do share memory.
- Abstraction used for sharing data among processes running on machines that do **not** share memory.
- Processes think they read from and write to a “virtual shared memory”.
- Primitives: read and write.
- OS ensures that all processes see all updates.
  - Happens transparently to processes.

DSM and MP

[neumann]

- DSM is an abstraction!
  - Gives programmers the flavor of a centralized memory system, which is a well-known programming environment.
  - No need to worry about communication and synchronization.
- But, it is implemented atop MP.
  - No physically shared memory.
  - OS takes care of required communication.
- For performance, DSM caches data locally.
  - More efficient access (locality).
  - But, must keep caches consistent.
  - Caching of pages for of page-based DSM.
- Issues:
  - Page size.
  - Consistency mechanism.

The distributed shared memory abstraction [Colouris]