Proactive gossip framework

```
// active thread
do forever
    wait(T time units)
    q = SelectPeer()
    push S to q
    pull S_q from q
    S = Update(S,S_q)

// passive thread
do forever
    (p,S_p) = pull * from *
    push S to p
    S = Update(S,S_p)
```

- To instantiate the framework, need to define
  - Local state ***S***
  - Method **SelectPeer()**
  - Style of interaction
    - **push-pull**
    - **push**
    - **pull**
  - Method **Update()**

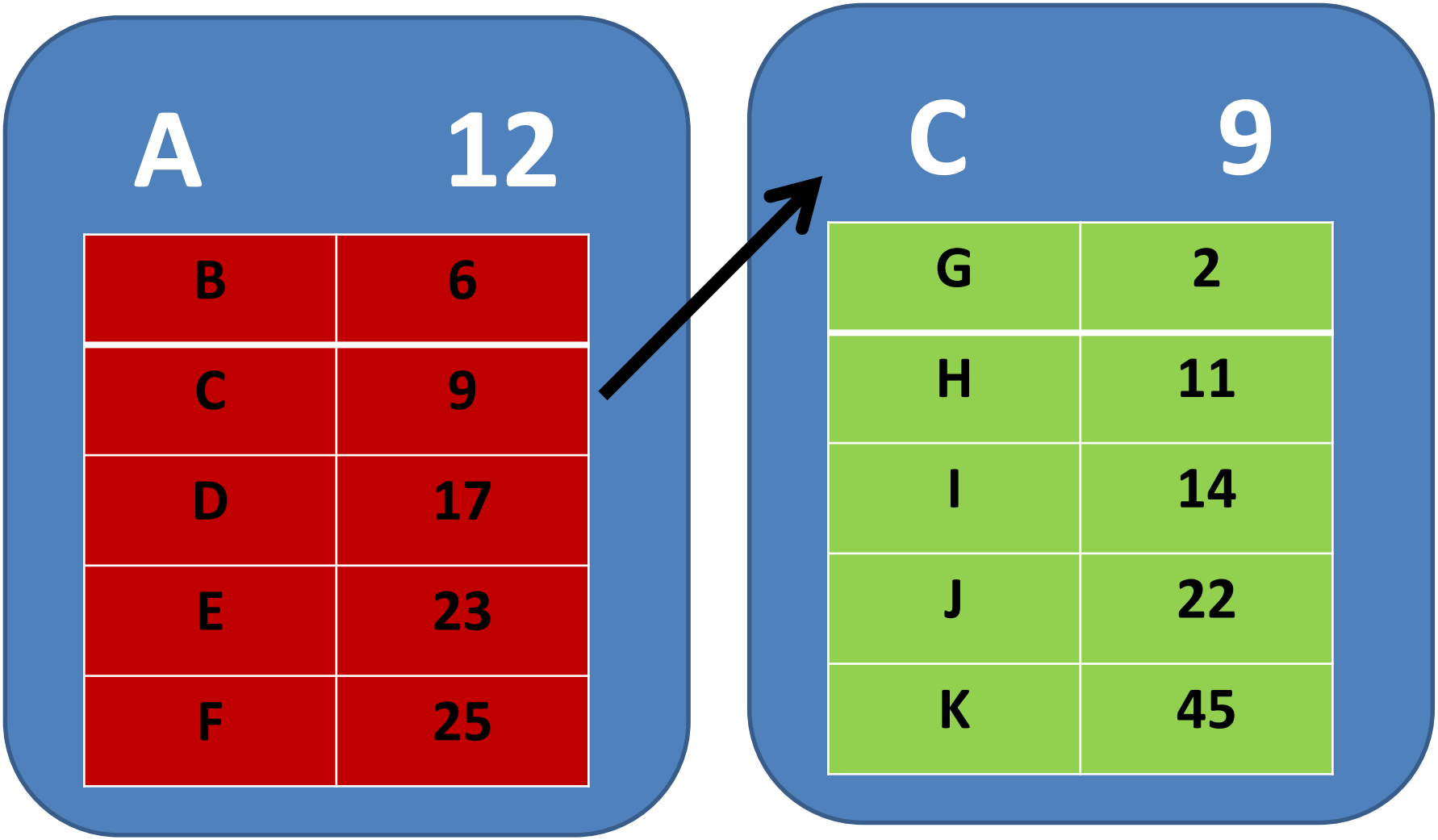
Topology creation

- Can we use gossip to build and maintain any network topology ?
  - **Ideas?**
  - E.g.: Rings, Toruses, Proximity networks, Small-World networks/DHTs etc
- High level intuition:
  - Local state ***S***: Current neighbor set
  - Method **Update()**: Ranking function defined according to desired topology (e.g., Small-World)
  - Method **SelectPeer()**: first (adjusted according to the ranking function) neighbor
  - Style of interaction: push-pull

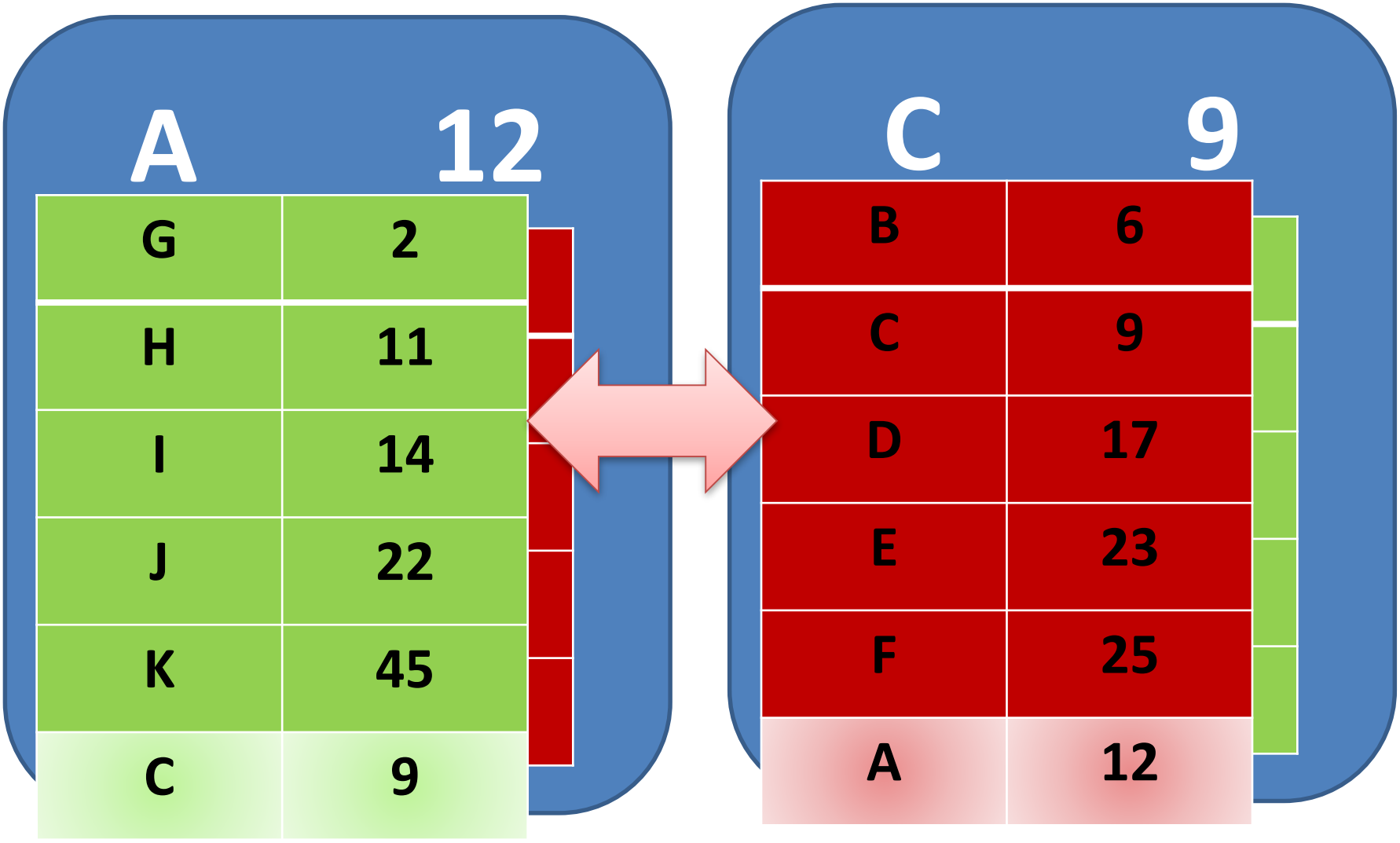
T-man

- Jelasity et al. **”T-Man: Gossip-based Overlay Topology Management”**
  - T-man is a protocol that can construct and maintain **any topology** with the help of a **ranking method**
  - The ranking method **orders any set of nodes** according to their desirability to be neighbors of a given node
  - For example based on **distance function** (ring, tree, mesh)

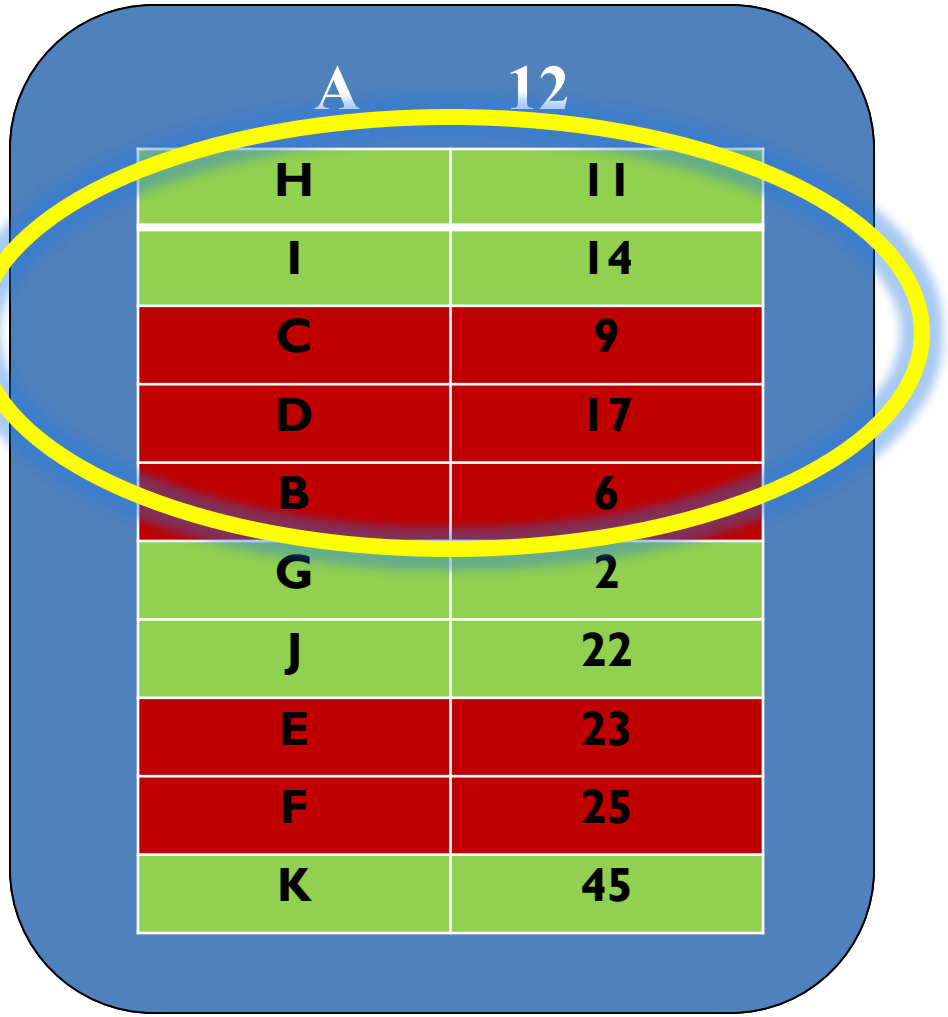
Intuition Behind the Idea



Exchange of Views



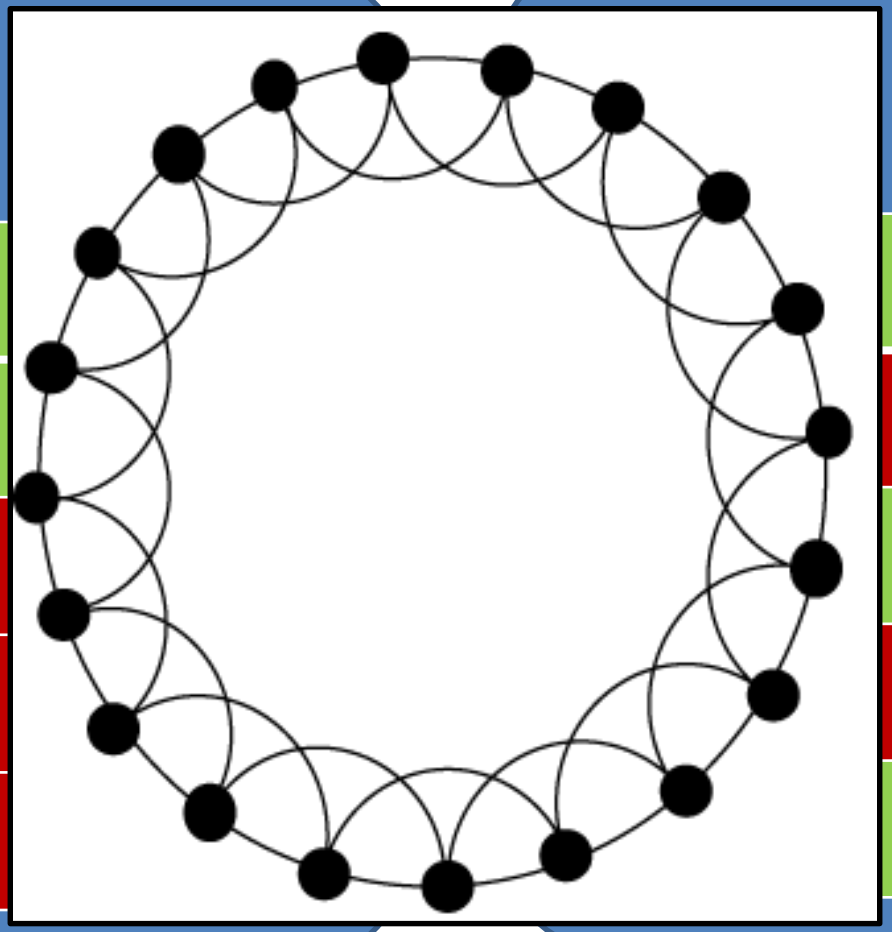
Exchange of Views



- Preference function:
  - distance between mine and their *id*s
    - Keep *k* nodes, with min distance
- **Can we get into trouble?**
- A fix:
  - Need to include "random links" for better convergence speed and to avoid disconnection

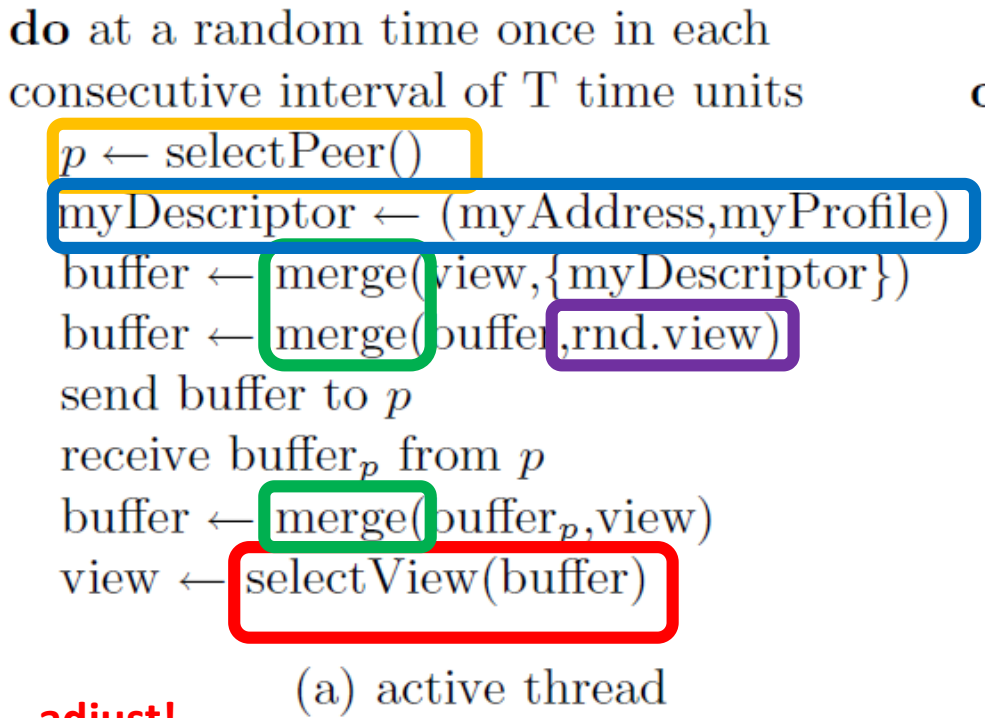
To what Topology Do we converge?

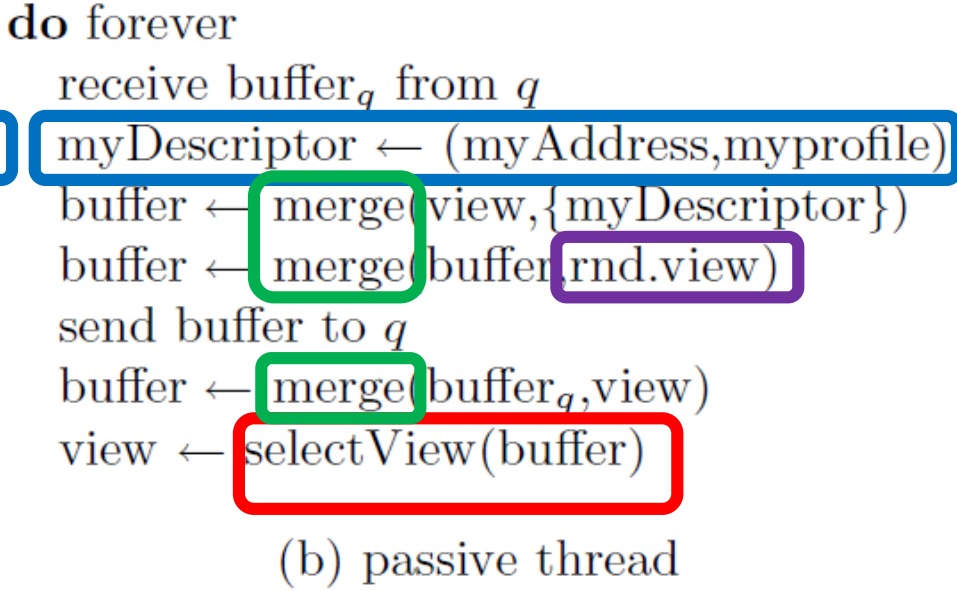
A	
H	
I	
C	
D	
B	



	9
	2
	6
	11
	12
	14

T-Man Protocol





adjust!

selectPeer: (**try to contact the "most useful" peer!**) selects the first node from the view adjusted according to ranking order (be careful: ranking function might not always be exactly the same as in selectview, e.g., think of raking function of preferring furthest away IDs…)

- myDescriptor: contains address and application dependent information
- merge: implements set operation merge (i.e., what do we know after the gossip round?)

selectView: applies the **ranking function to order the elements** in the buffer and returns the first c elements of the buffer according to ranking order.

- Rnd.view: provides a random sample of the nodes from the entire network (e.g., using Cyclon)

T-Man: protocol

- Why random view is necessary?
  - Very important in **large diameter topologies** (e.g., ring)
    - **Speeds up** the discovery of "right neighborhood"
    - Acts as **long range links** to speed up the convergence
  - Helps **merging** disconnected clusters

Torus Example

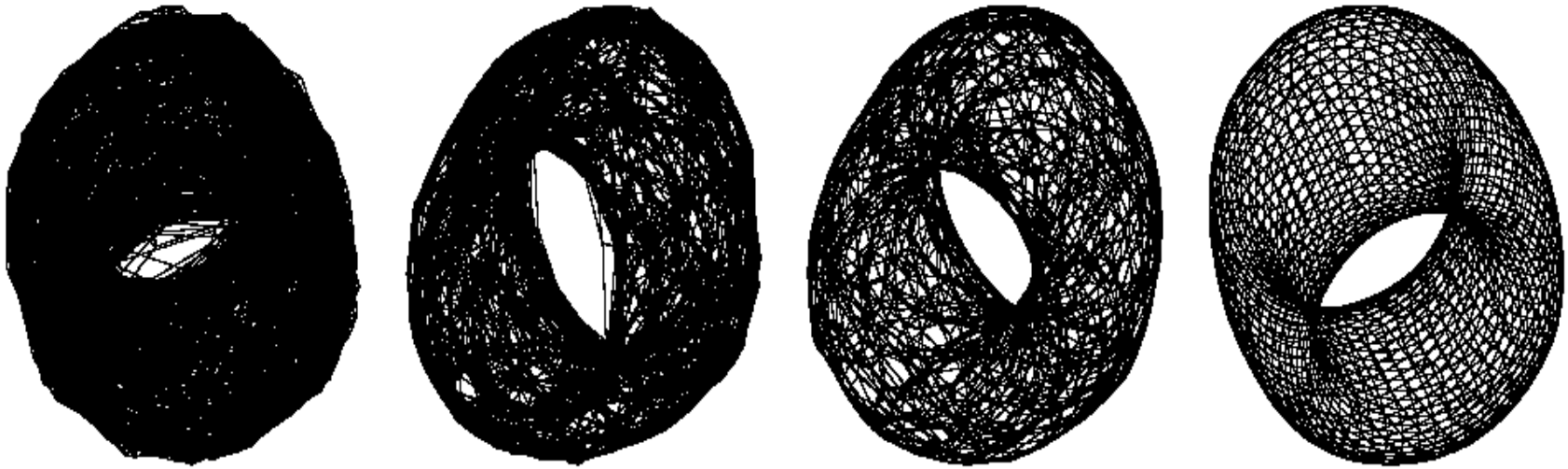
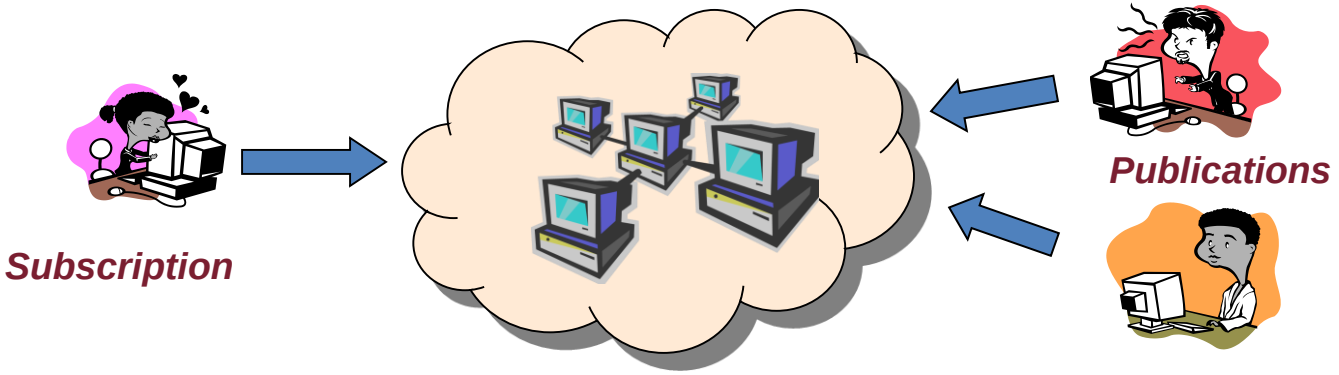


Figure 2. Illustrative example of constructing a torus over $50 \times 50 = 2500$ nodes, starting from a uniform random topology with $c = 20$. For clarity, only the nearest 4 neighbors (out of 20) of each node are displayed.

Publish/Subscribe Systems

- **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.
- A **Publish/Subscribe System** notifies to each subscriber every published event that matches at least one of its subscriptions.
  - Centralized Pub/Sub service
  - **Distributed/Decentralized Pub/Sub service**

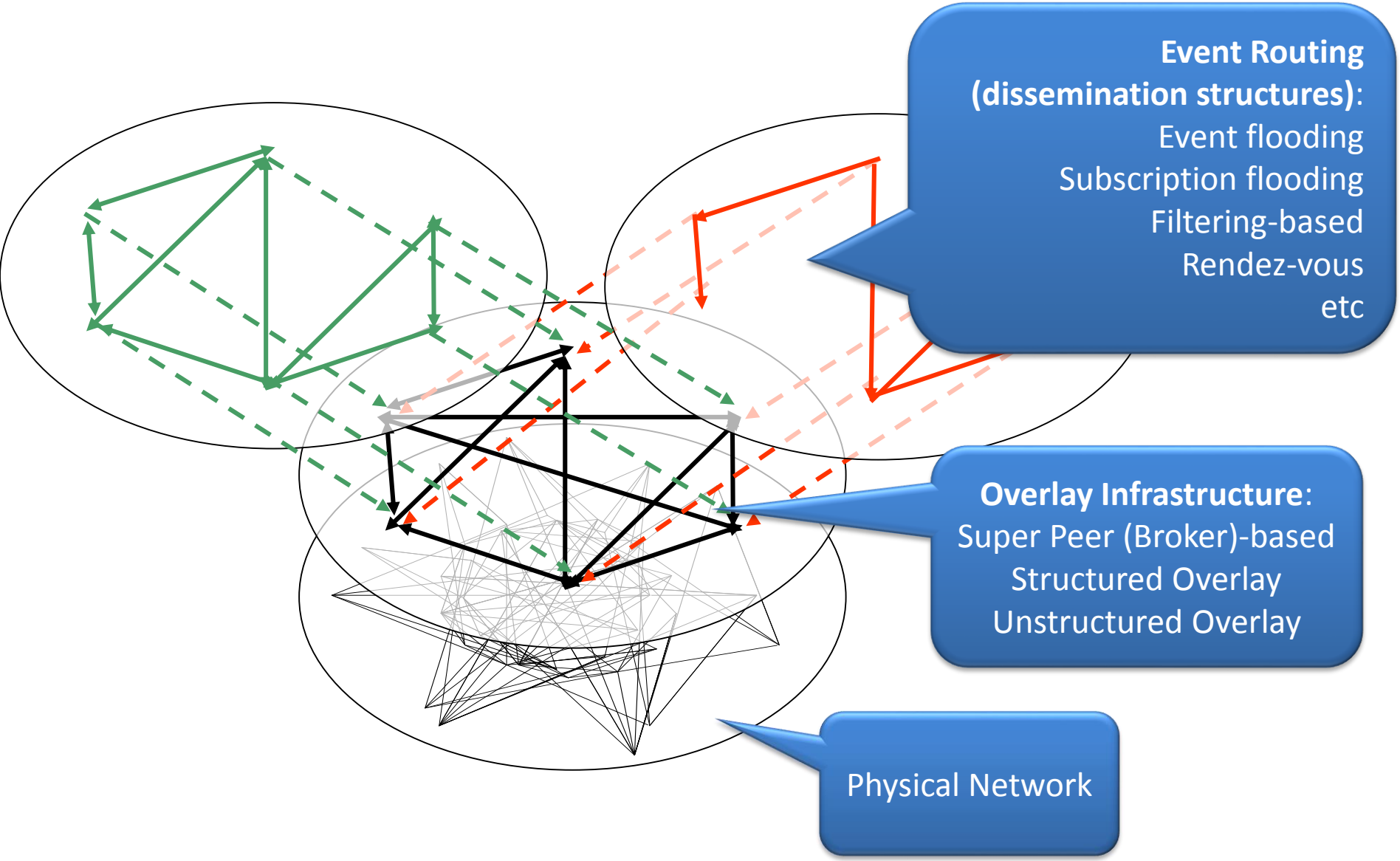
Pub/Sub as a Distributed Service



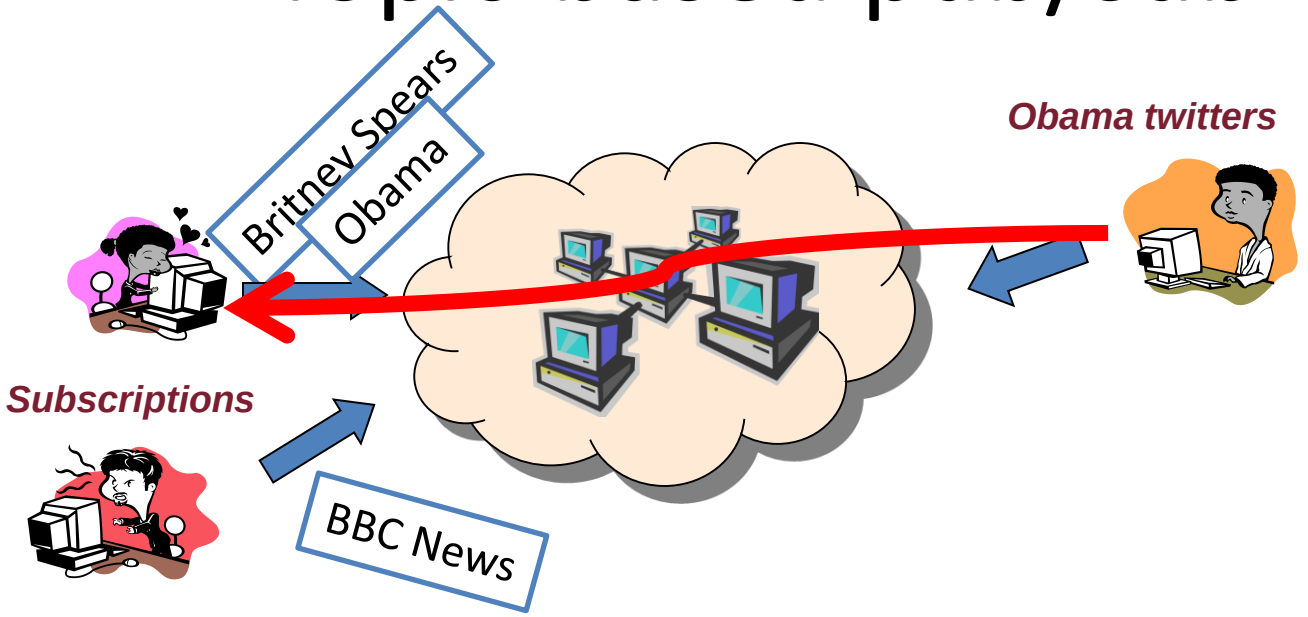
- Set of nodes **distributed** over the **communication network**

- **Network performs routing** such that
  - Publications “meet” subscriptions
  - Publications delivered to appropriate subscribers

Pub/Sub architecture



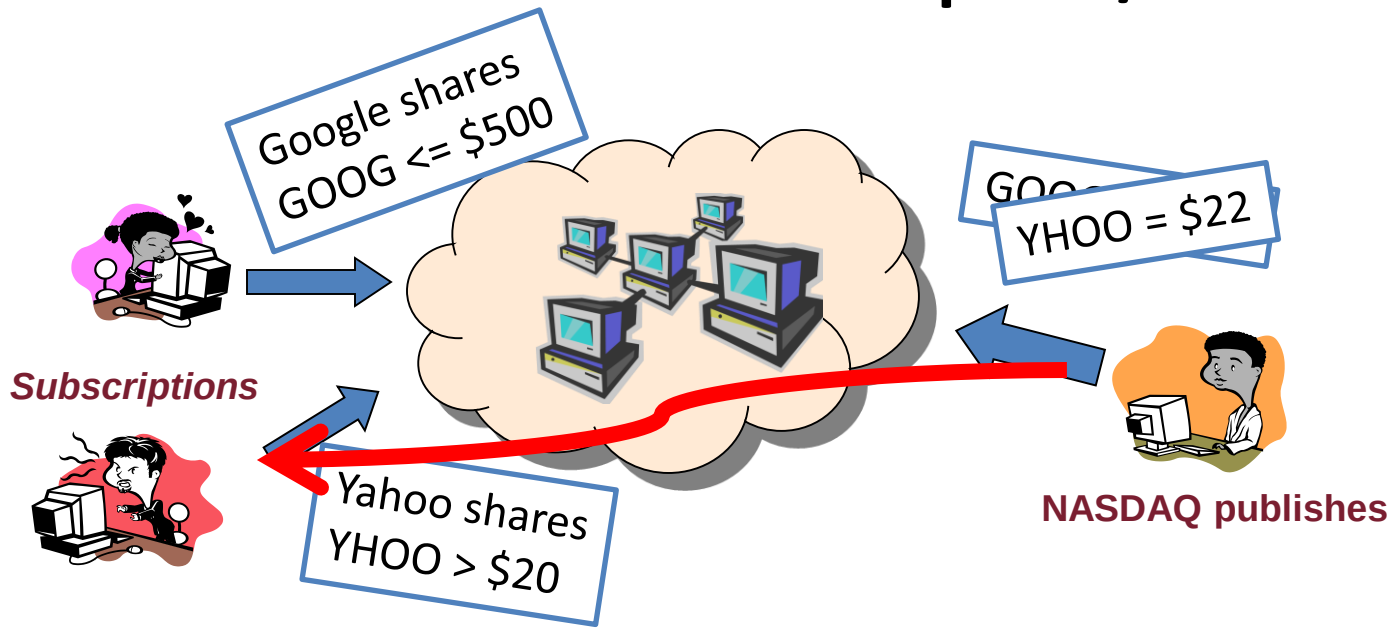
Topic-based pub/sub



Twitter Topic
Obama
Britney Spears
BBC News
DowningStreet

- The most popular pub/sub
- ☹ Limited expressiveness
  - Subscriber interested in a subset of events for a topic receive all the events on that topic
  - Extension to hierarchical-model
    - Topic A can be a subtopic of a more general B

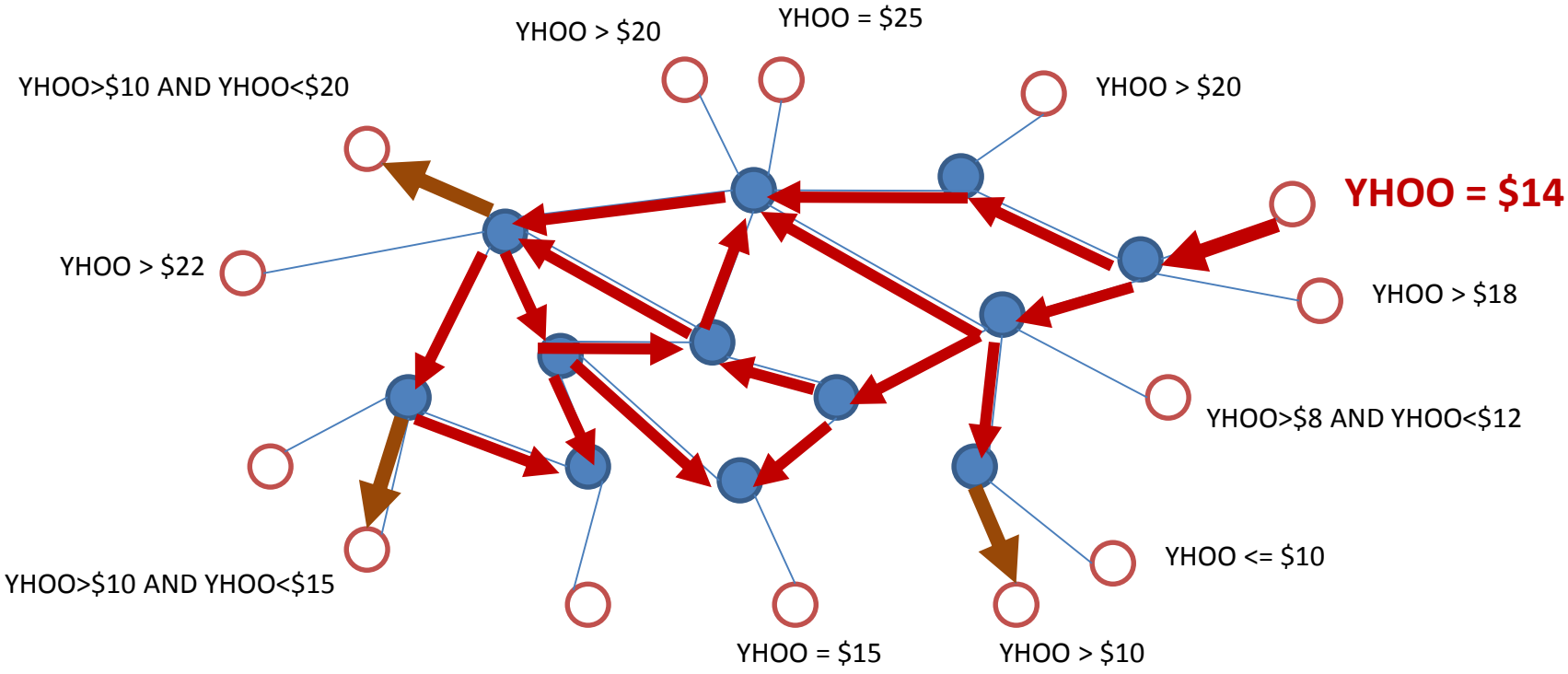
Content-based pub/sub



- The data published in the system is mostly structured.
- The content-based pub/sub system filters out useless events before notifying a subscriber.

Naïve solution: **Event Flooding**

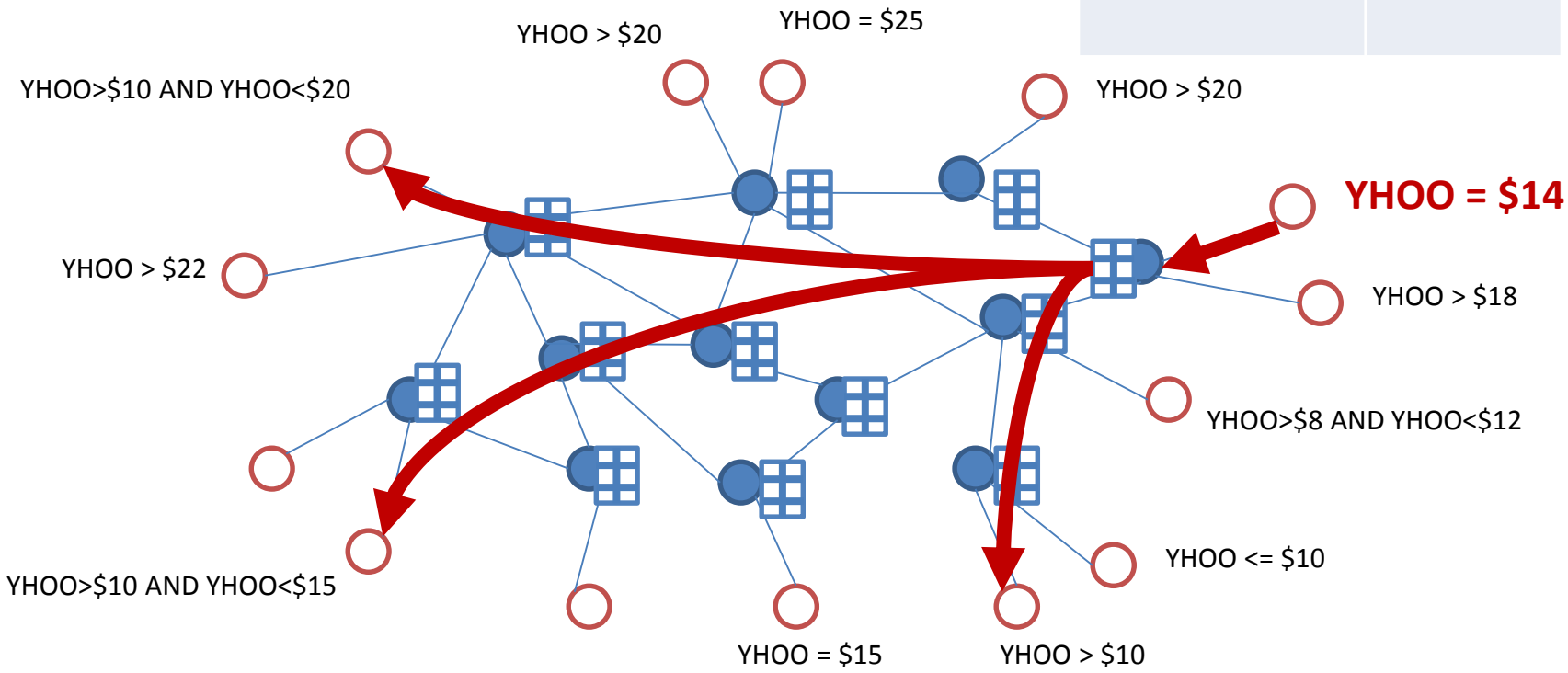
- **Each event is broadcasted from the publisher in the whole** system.
- The implementation is straightforward but very expensive.
- This solution has the highest message overhead with no memory overhead.



Naïve solution: **Subscription Flooding**

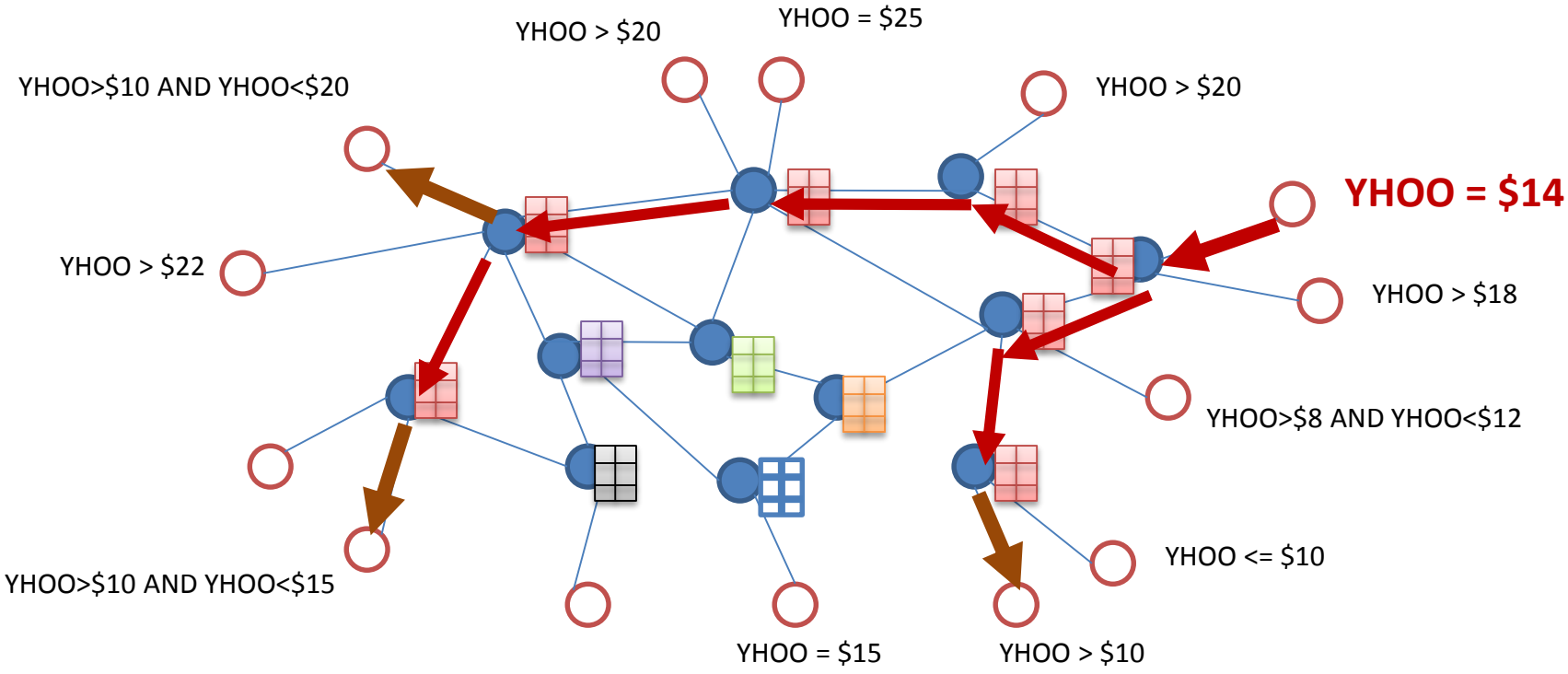
- Each subscription is copied on every node (broker);
- Every node (broker) maintains complete subscription tables, which are used to locally match events and directly notify interested subscribers.
- The approach suffers from a large memory overhead, but event diffusion is optimal.
- Impractical in applications where subscriptions change frequently.

Subscription	Node
YHOO>$22	IP x.x.x.x
YHOO<=$10	IP y.y.y.y
YHOO>$18	IP z.z.z.z
…	…



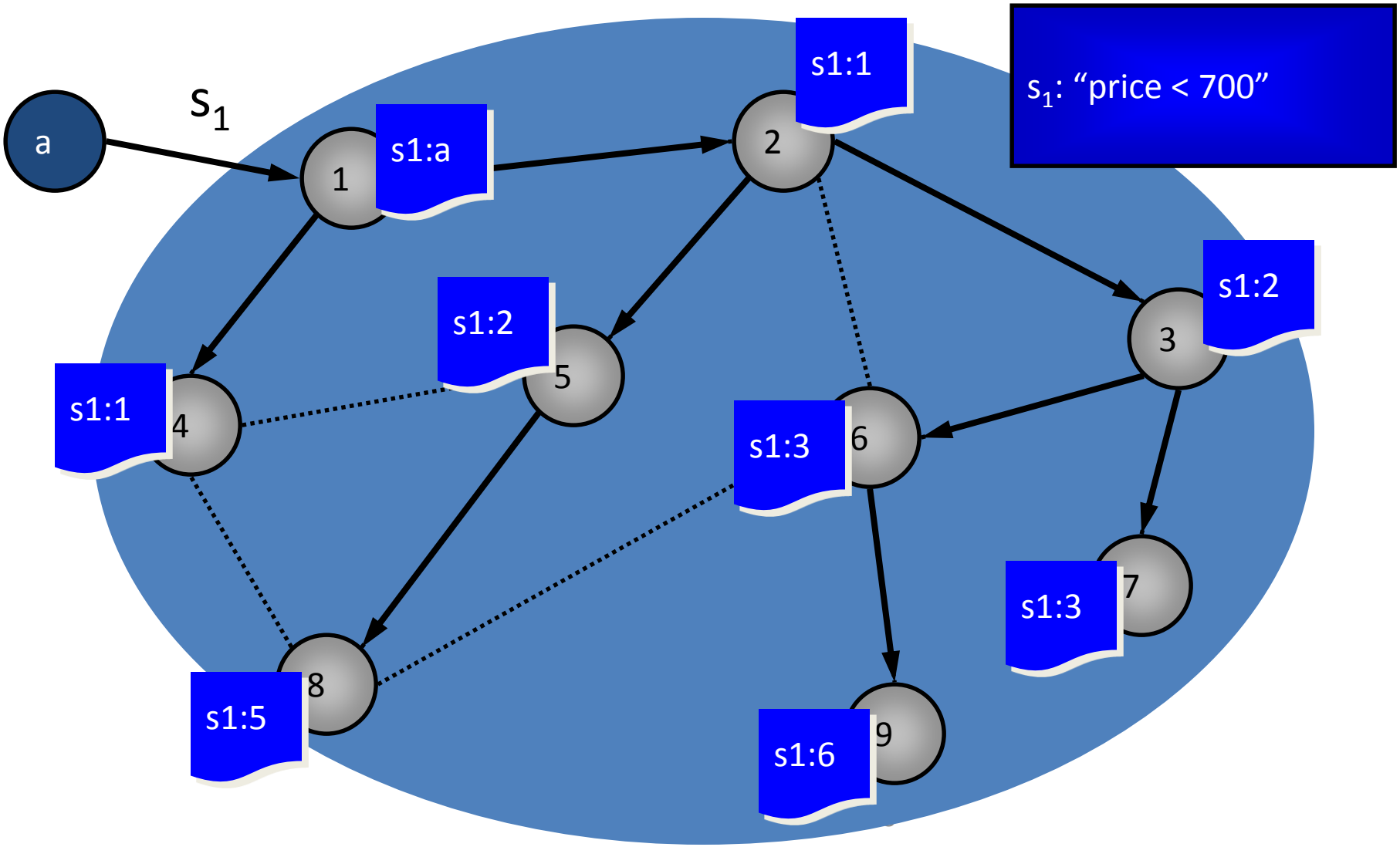
Filter Based Routing

- 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.
- E.g., SIENA
  - Carzaniga et al. "**Achieving scalability and expressiveness in an Internet-scale event notification service**", in ACM Symposium on Principles of Distributed Computing, 2000



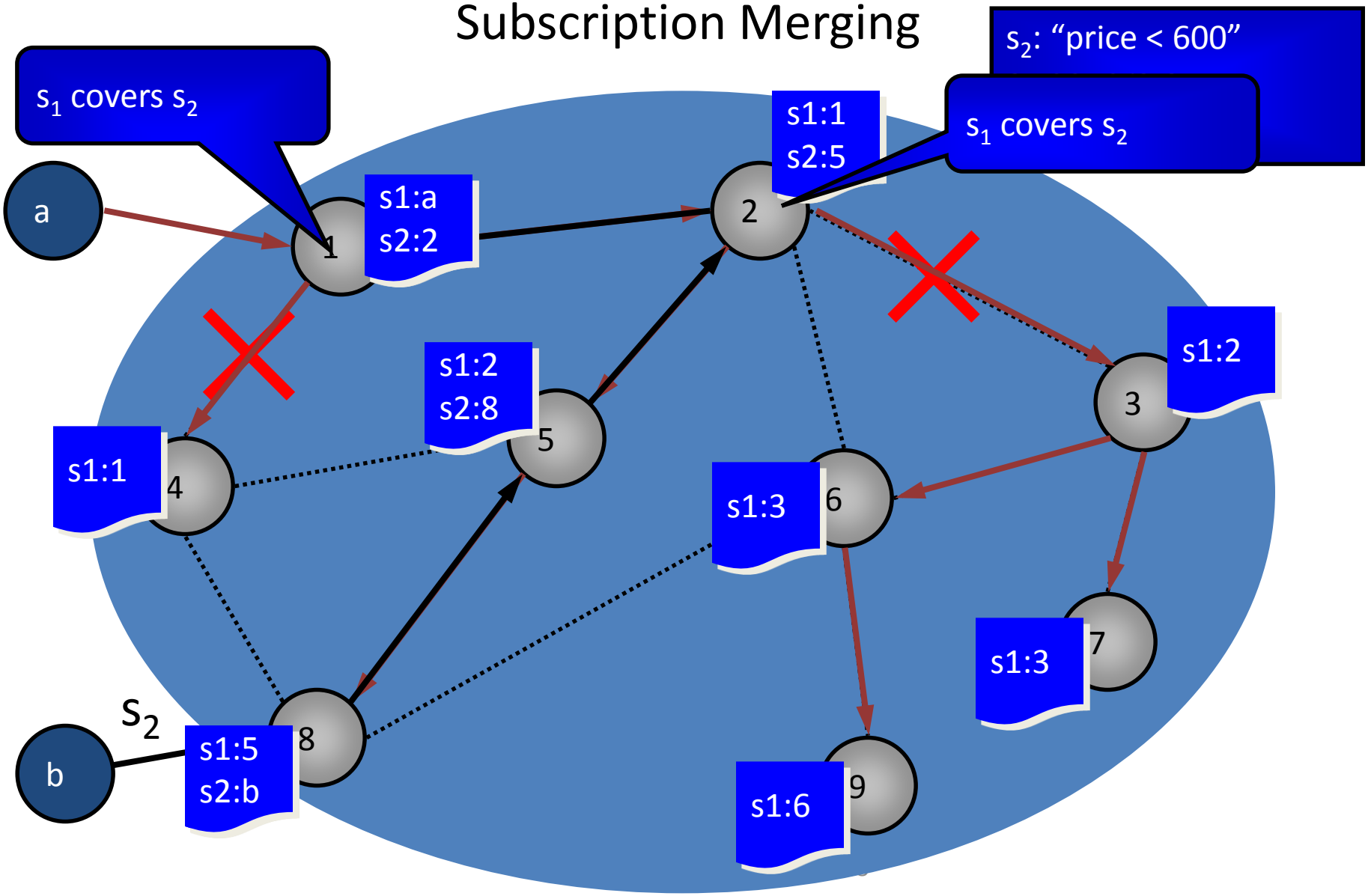
SIENA Content-Based Routing

Subscription Forwarding



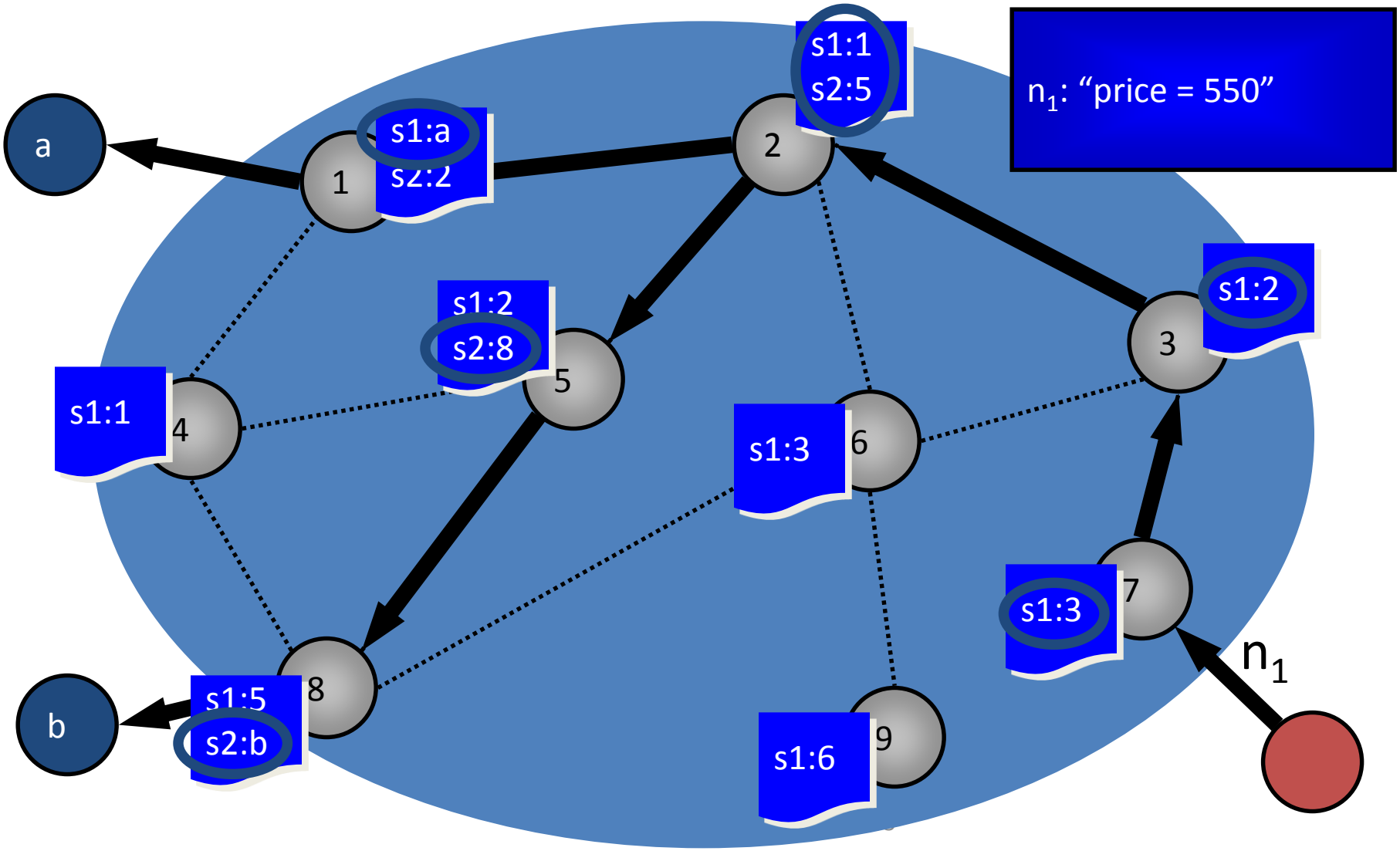
SIENA Content-Based Routing

Subscription Merging



SIENA Content-Based Routing

Notification Delivery

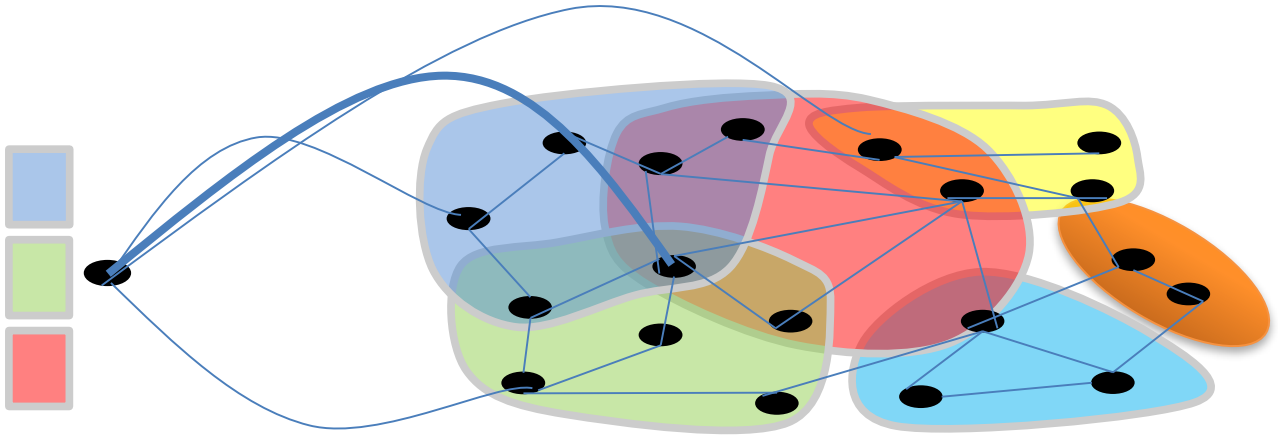


Implications of SIENA's Design

- **Notifications** can be very **frequent**
- **Subscriptions** should be relatively **infrequent**
- There should be some **similar subscriptions**
- The similar subscriptions should come from the **same part of the network**
- **More issues:**
  - **Might not be possible to reuse trees** when we have **topic-based pub/sub**
  - Forwarders might not be happy to keep on forwarding if they are not interested themselves (**spam**)

Overlay-Per-Topic Pub/Sub

- Overlay is **topic-connected** if the subgraph induced by every topic is connected
- Nodes often have **multiple interests**
  - What could be the problem?
- Construct a topic-connected overlay using as few links as possible (small average degree)
  - Cluster similar peers together

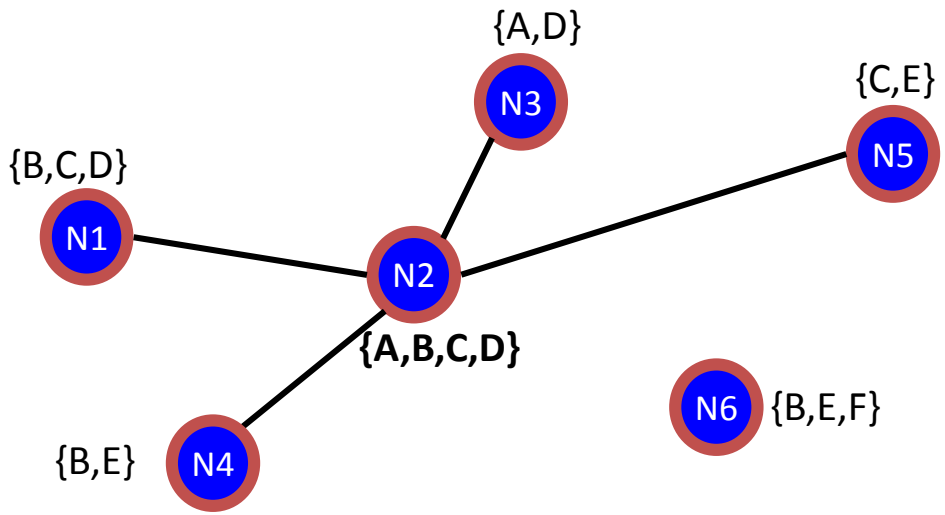


SpiderCast

- Chockler et al. “Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication”, 2007
  - **Topic-based** pub/sub

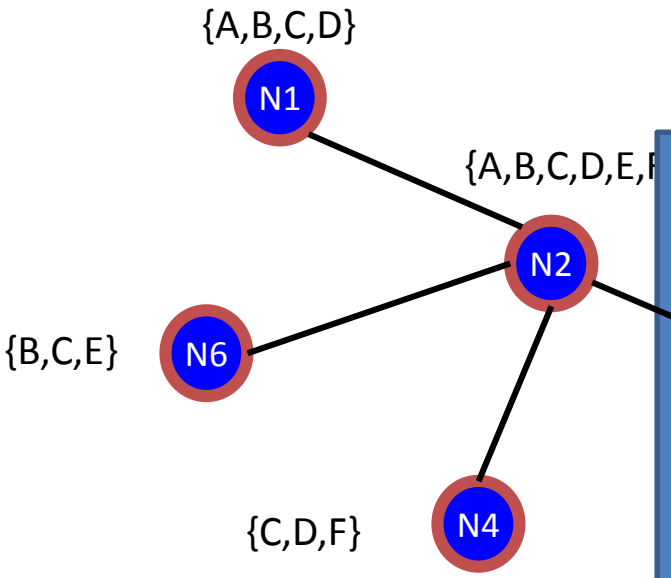
Links of SpiderCast

- **Greedy links**
  - Each node uses the interest views to create interest-aware links
  - Bias by interest similarity reduces the node degree



Interest-Aware (Greedy) Links

- Greedy Coverage Algorithm
  - Bias also towards less covered topics
  - The greedy coverage heuristic selects a neighbor that minimizes the number of topics which are not yet (**Kg**) covered.
- **In practice often greedy links only are sufficient to ensure connected components**



Input: $interest_i$, $view_i$={$interest_j$: ∀j}, k>0
while (∃t ∈ $interest_i$: [t is not k-covered && ∃j∉$neighbors_i$: t ∈ $interest_j$]) **do**
    find j ∉$neighbors_i$ **such that** $interest_j$ ∩ $interest_i$ **covers the biggest not yet covered portion of** $interest_i$
    connect to j;
    $neighbors_i$ = $neighbors_i$ ∪ {j};

Why Topics Are Connected?

- **What can go wrong if we have only Greedy links?**
- **Assuring that each topic is connected**
  - if every node interested in T establishes at least 3 random links to other nodes which are also interested in T, then all the nodes interested in topic T will form a connected component with high probability. (According to theory of k-regular random graphs)

– **Additional 2nd type of links: Random Links**
  - The random coverage heuristic randomly selects a node whose addition as a neighbor would reduce the number of topics that are not yet covered by ***Kr*** random links.

The SpiderCast Protocol

- Each node n tries to cover **K** times each of the topics in which it is interested.
  - That is, for each topic **t** in which **n** is interested, **n** tries to maintain connections to **K** other nodes that are also interested in topic **t**.
- Two types of links: **Random (Kr)**and **Greedy (Kg)**
  - **K = Kr+Kg**
- Can be constructed by gossip protocols (remember Cyclon and T-man!)

SpiderCast: pros and cons

- ☺ Efficient when there is a lot of user subscription correlation in the system
- ☹ Unscalable when the correlation is low
  - Node degree grows linearly with the number of node subscriptions
- ☹ Publishers should also be interested in the topic (need to participate in the overlay)

TERA

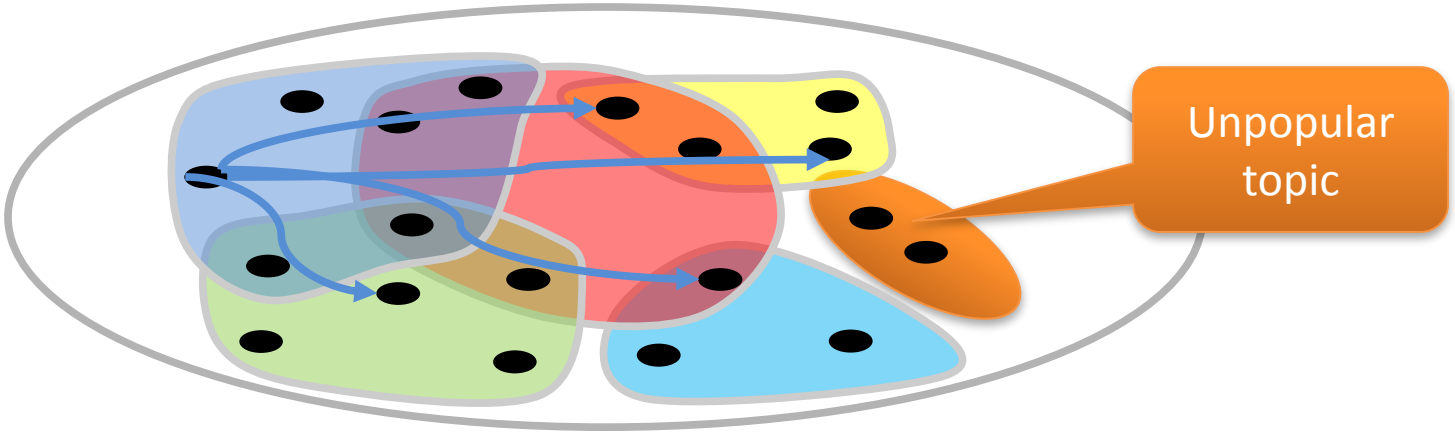
- Baldoni et al. “TERA: topic-based event routing for peer-to-peer architectures”, 2007
  - Publishers do not have to be interested in the topic they are publishing
  - Similar to SpiderCast
    - Interest Clustering
    - Inner-Cluster Dissemination
    - **Outer-Cluster Dissemination**

TERA

- Baldoni et al. “TERA: topic-based event routing for peer-to-peer architectures”, 2007
  - Publishers do not have to be interested in the topic they are publishing
  - Similar to SpiderCast
    - Interest Clustering
    - Inner-Cluster Dissemination
    - **Outer-Cluster Dissemination**

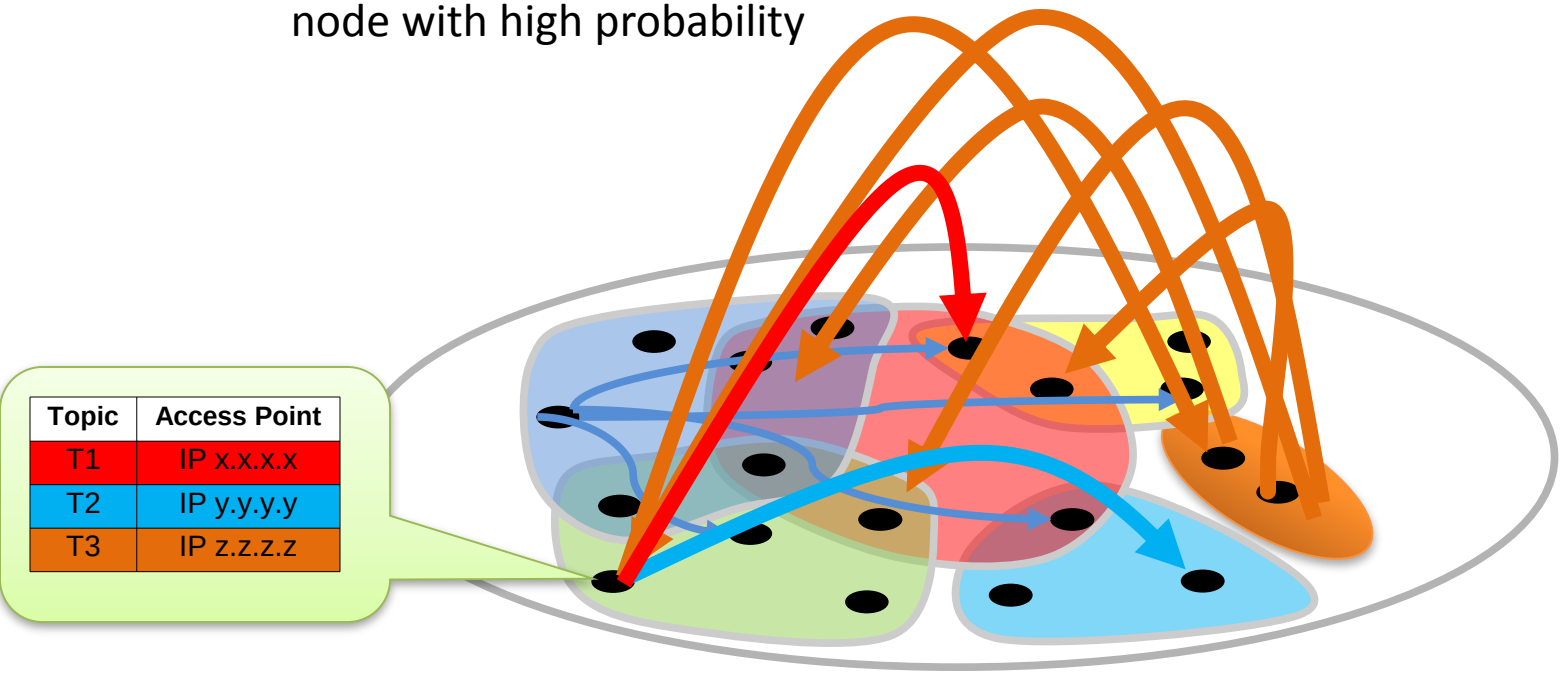
System Overview

- Interest Clustering
  - Instantiates a dedicated overlay network per topic
- Inner-Cluster Dissemination
  - Uses a simple flooding scheme (which can be replaced by more sophisticated routing algorithms, if required)
- How to find a specific cluster in a decentralized fashion (Outer-Cluster Dissemination)?
  - **Random walks** in the network
  - What is the probability of finding?
  - What if a cluster (topic) is very unpopular?



Outer Cluster Routing

- Solution:
  - Every topic advertizes itself on the random nodes in the network (via gossiping)
  - BUT (!) with the **probability inversely proportional to the size of the cluster** (topic population)
  - Every node keeps **Access Point Table** with pointers to **access points** to different clusters (topics)
  - When a random walk is performed one is assured to find a required access node with high probability



Outer Cluster Routing: an Example

- Populating APT:
  - Node X contacts node Y
- Outer Cluster Publishing

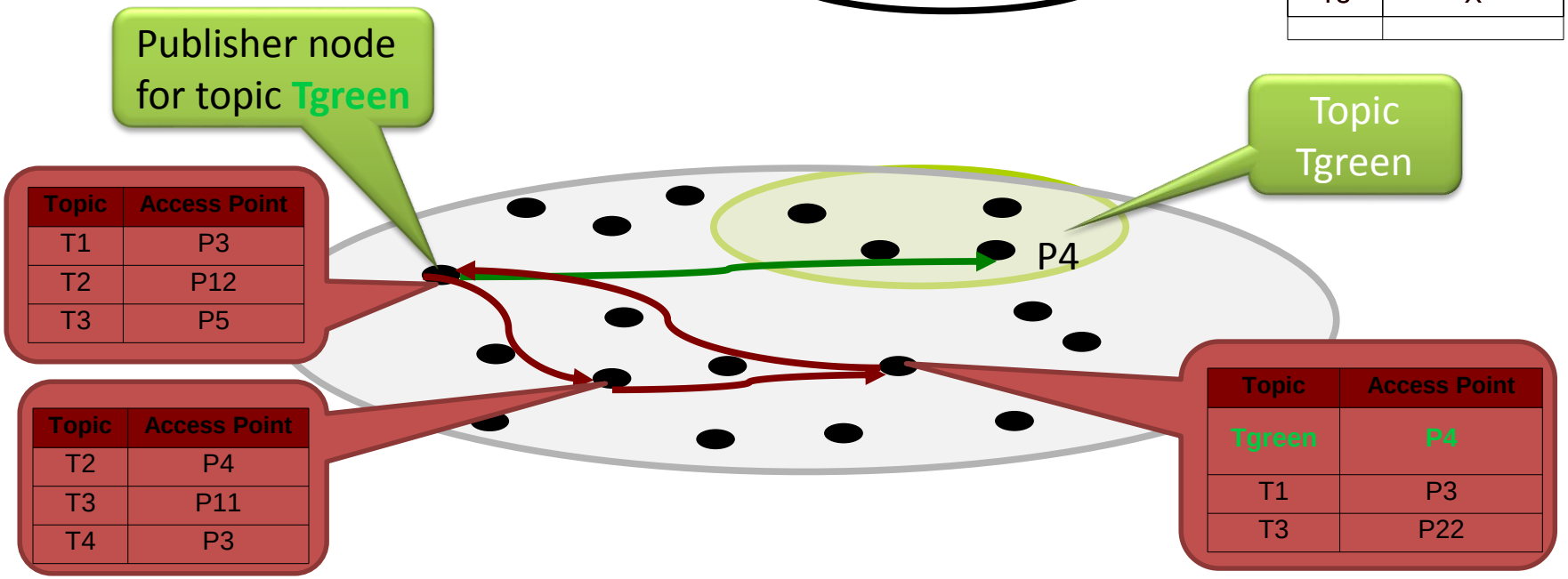
Subscriptions of node X

Topic	Popularity
T2	456
T3	123
T4	678
T5	15

APT in node Y

Topic	Access Point
T1	Q
T2	R

Topic	Access Point
T1	Q
T2	X
T3	X



Issues with Tera

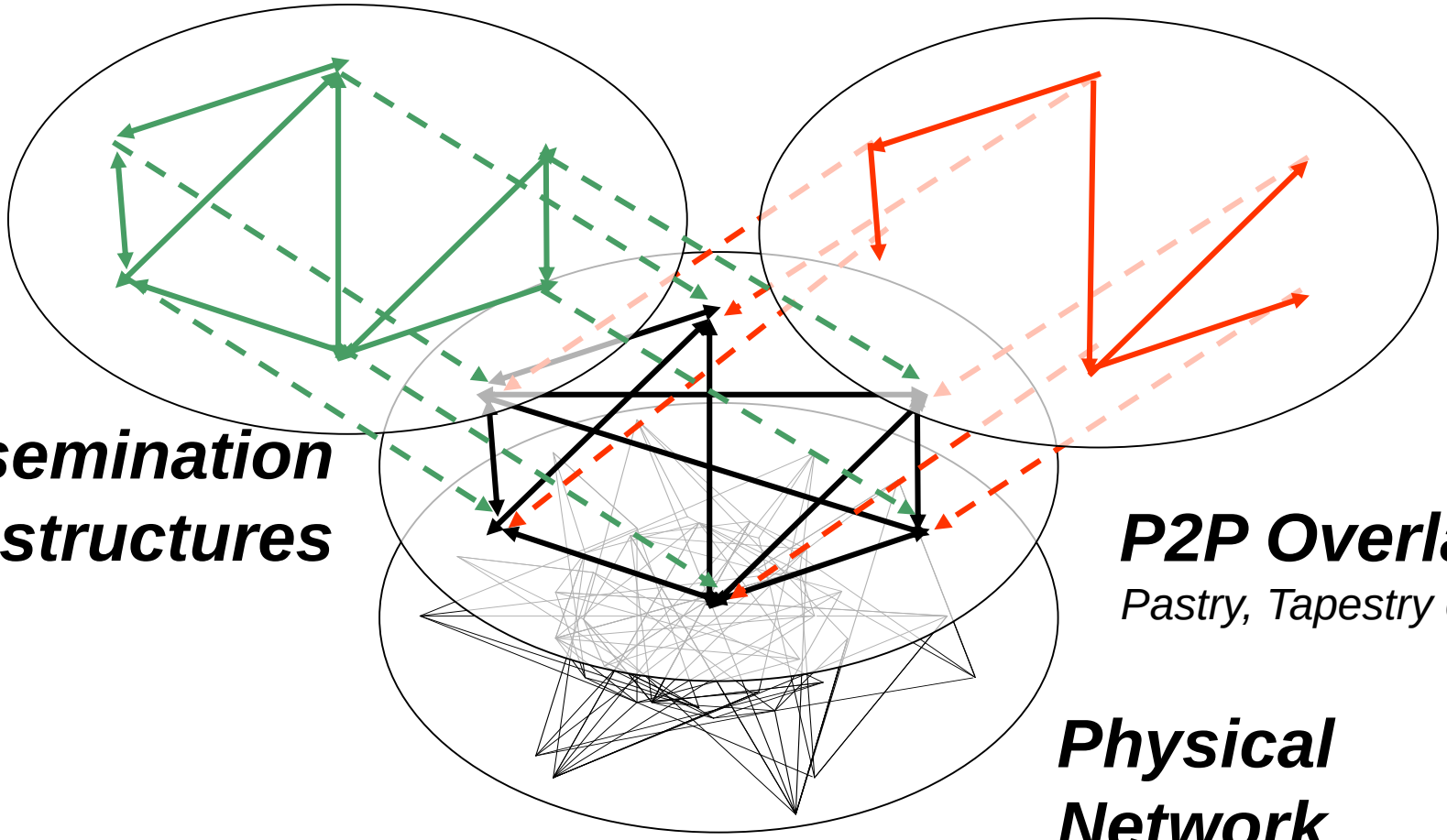
- **Scalability problem**
  - A node has to become member of as many topic overlays as the number of the topics it subscribes to.
- **Reliability problem**
  - If the APT size is small, the lookups may easily fail. If the size of APT is big, the maintenance overhead increases, and entries may become stale.

Rendezvous based pub/sub

- Castro et al. “**SCRIBE: A large-scale and decentralized application-level multicast infrastructure”, 2002**

- Zhuang et al. “**Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination”, 2001**

Structured P2P based Pub/Sub Systems



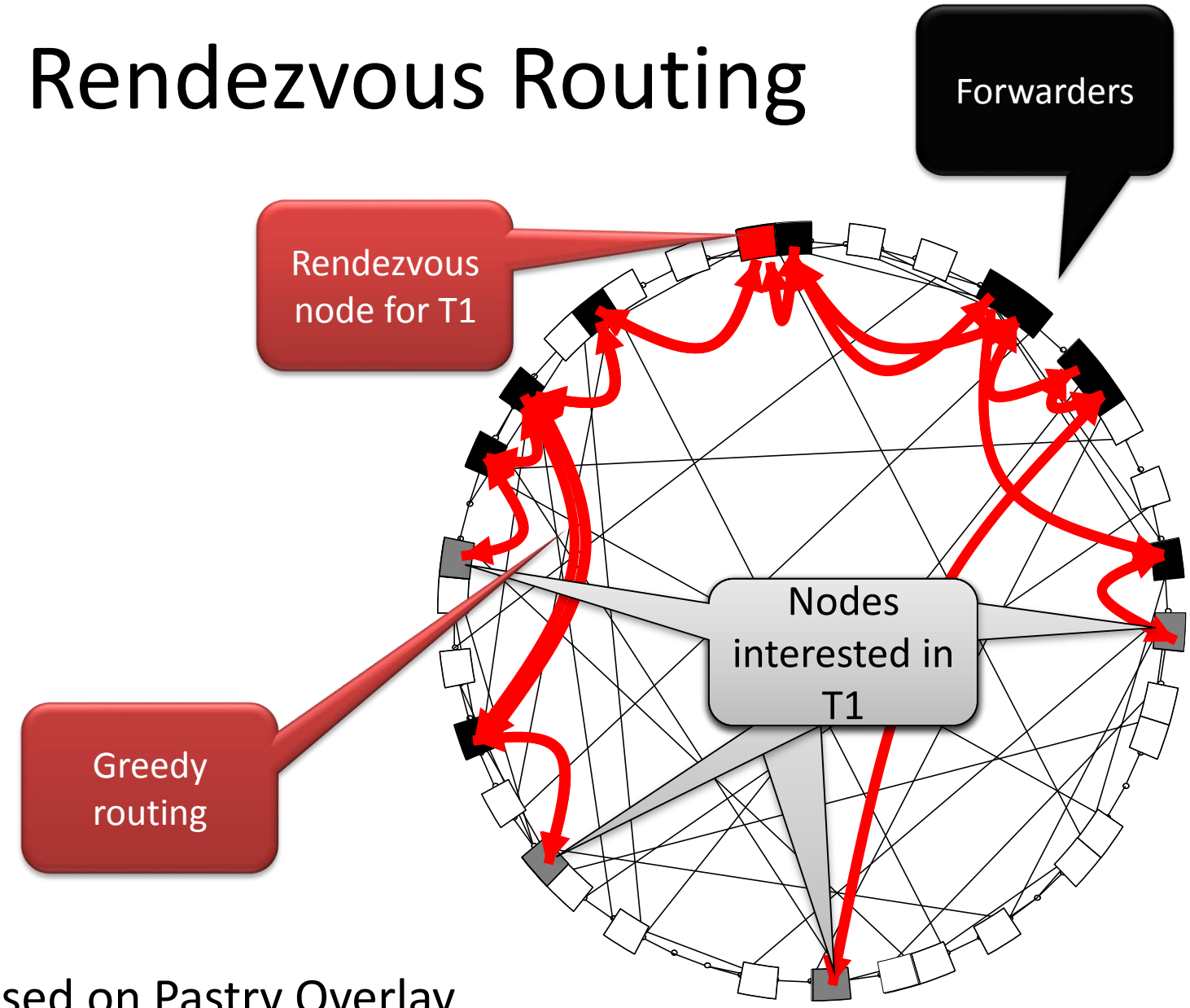
Dissemination structures

P2P Overlay

Pastry, Tapestry etc.

Physical Network

Scribe: Rendezvous Routing



- Scribe is based on Pastry Overlay