Publish/Subscribe Implementations

by Group 6:
Amirhosein Taherkordi
Omar Mohamed Jama
Daniel Johan H. Nebdal
{amirhost, omarmj, djnebdal}@ifi.uio.no

Course INF5040: Open Distributed Processing
Department of Informatics
University of Oslo

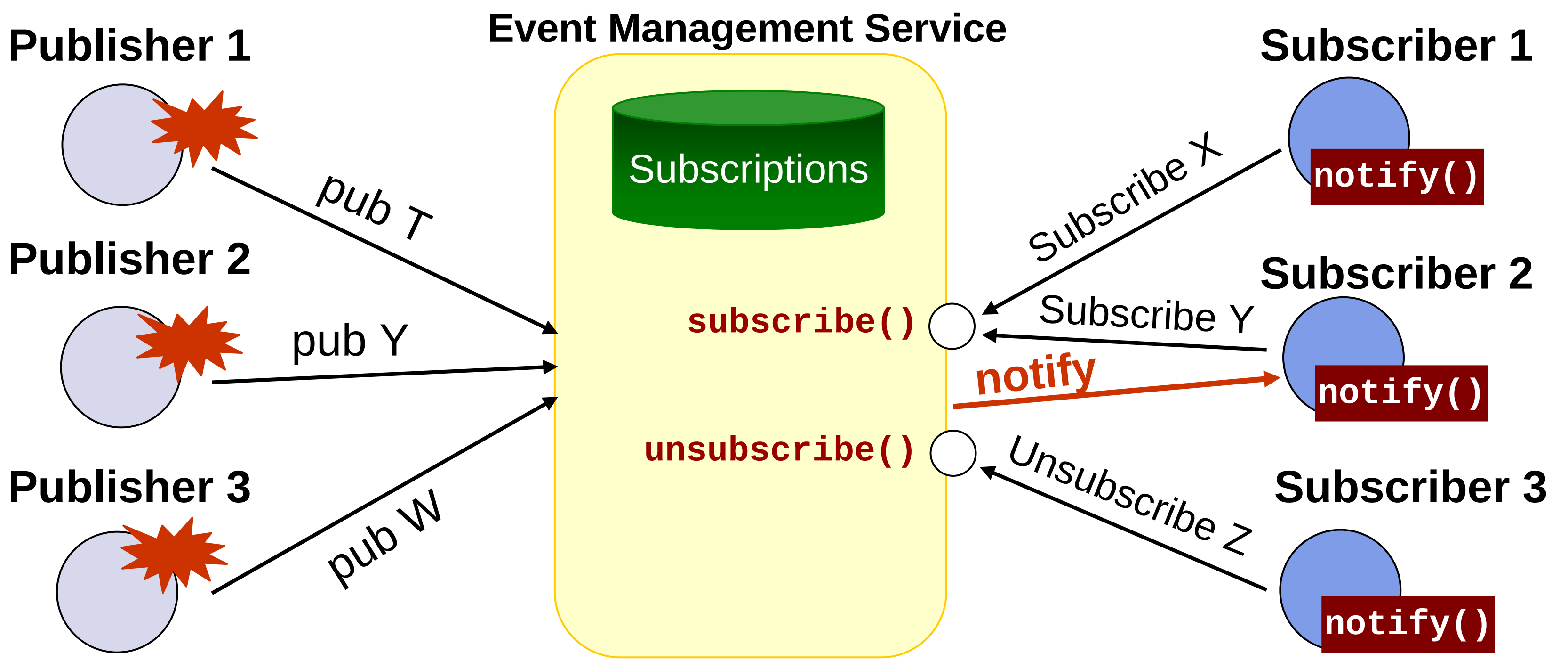
November 22, 2007

Outline

- Introduction
- Mires
- Siena
- Gryphon

Pub/Sub

Introduction



Pub/Sub
Introduction

- The key elements in the Publish/Subscribe paradigm:
  - Notification service
  - Subscription matching service
    - Subject/Group/Canal based
    - Content based
    - Type based
  - Subscriptions data store

Pub/Sub

Event Structure

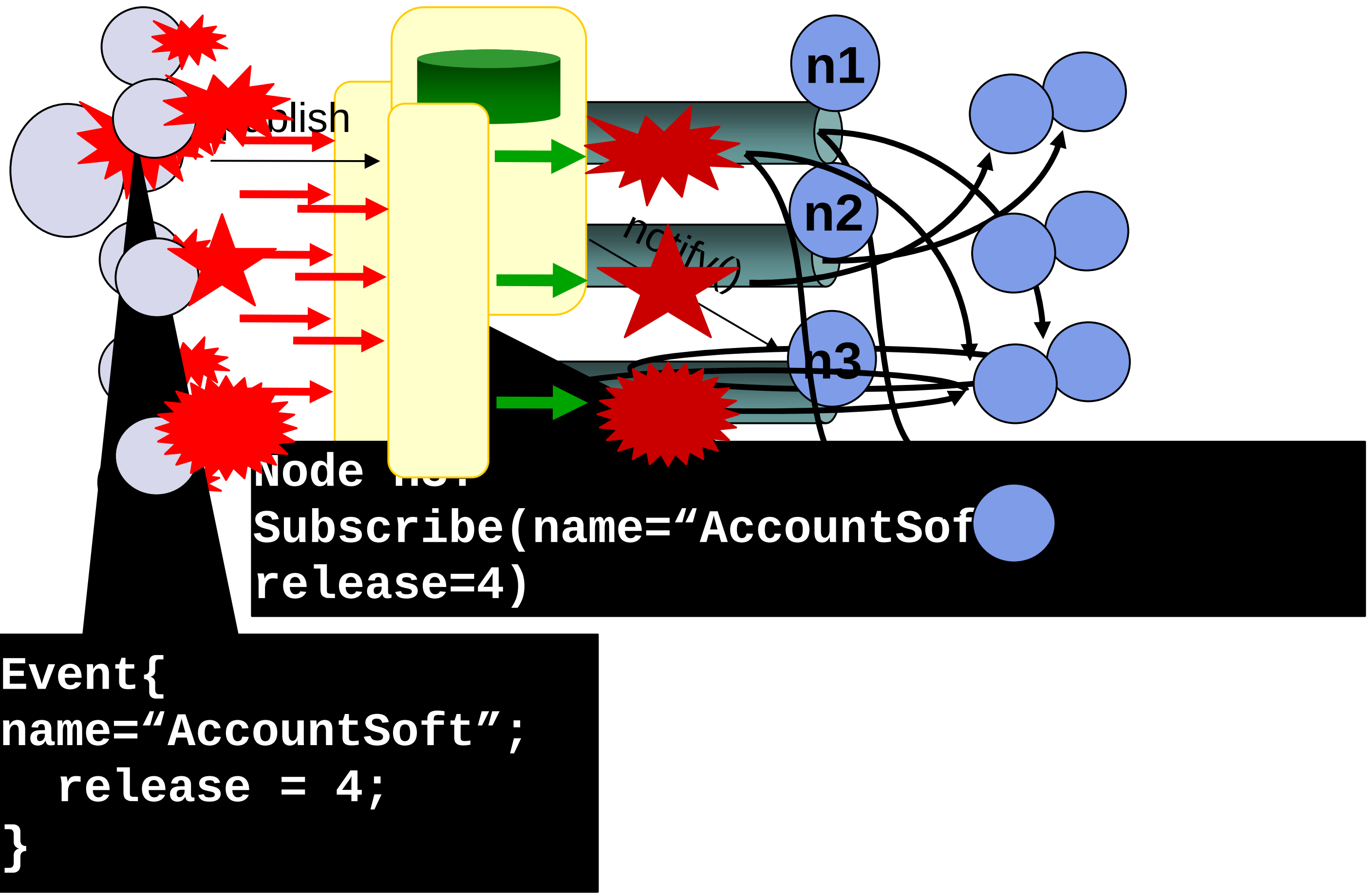
- Touple-based
- Record-based
- Object-based

```
class NewSoftwareRelease:
   public Event {
  public String ProductName;
  public String ProductRelease;
  private String DownloadURL;
  NewSoftwareRelease(String name,
      String Release, String URL);
  public void downloadAndInstall();
}
```

Pub/Sub

Subscription Matching

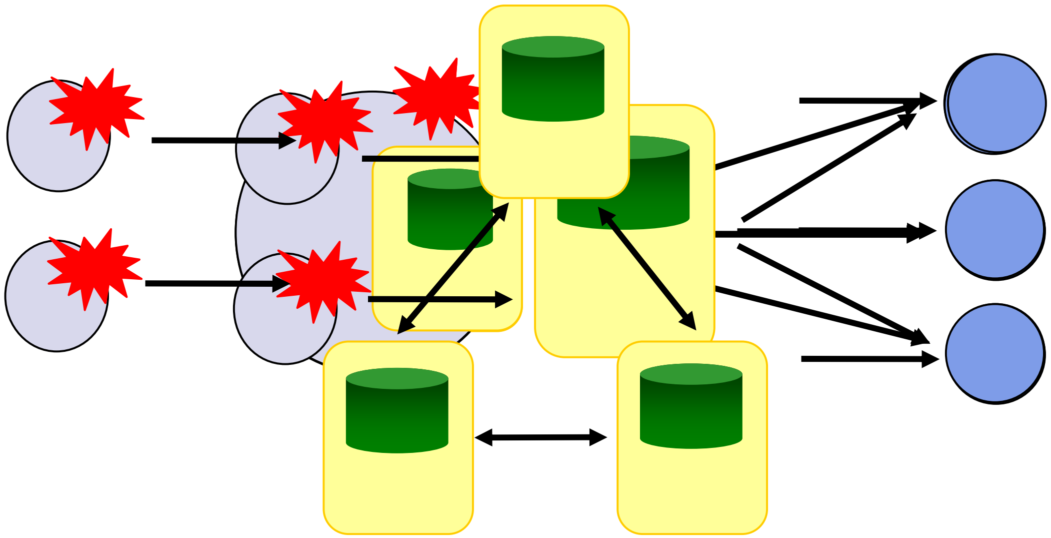
- Canal
- Content
- Type



Pub/Sub

Architectures

- Direct
- Broadcast
- Centralized
- Distributed



Mires
Introduction

- A pub/sub middleware for WSNs

To facilitate development, maintenance, deployment and execution of sensing-based applications

- Why pub/sub?
  - Communication between applications in WSNs is essentially based on events
  - Asynchronous communication
  - Loosely coupled communication

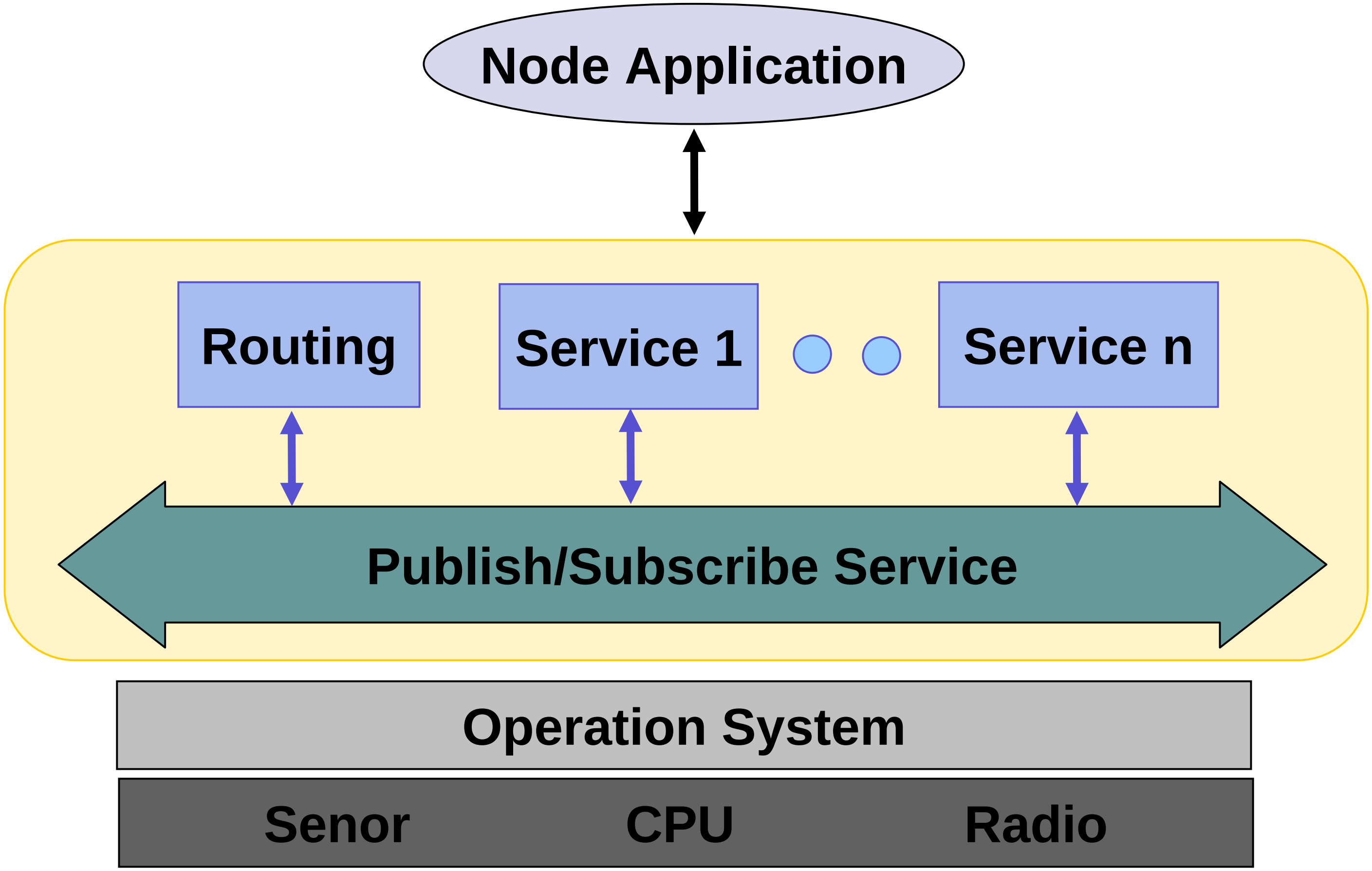
Mires

Introduction – cont.

- Top of TinyOS
- Nodes advertise the types of sensor data they can provide
- Encapsulates the network-level protocols
  - Multi-hop routing protocols
  - Topology control protocols
- Aggregation service: reducing the number of transmission

Mires

Architecture

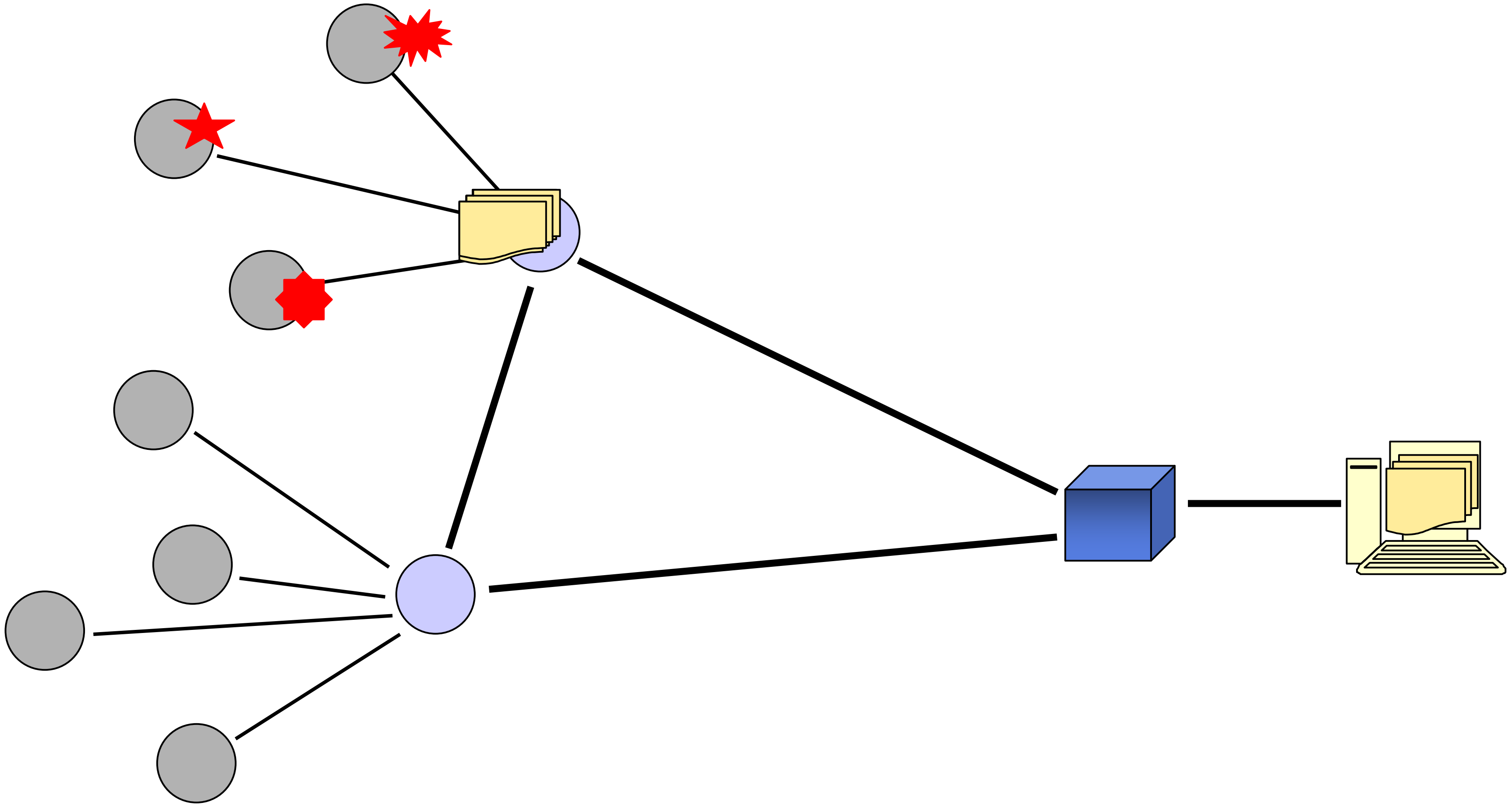


Mires
Comm. Phases

- Communication phases
  1. Advertising sensed data
  2. Routing data to the sink
  3. Subscribing to topics
  4. Start to sensing/processing/routing real data

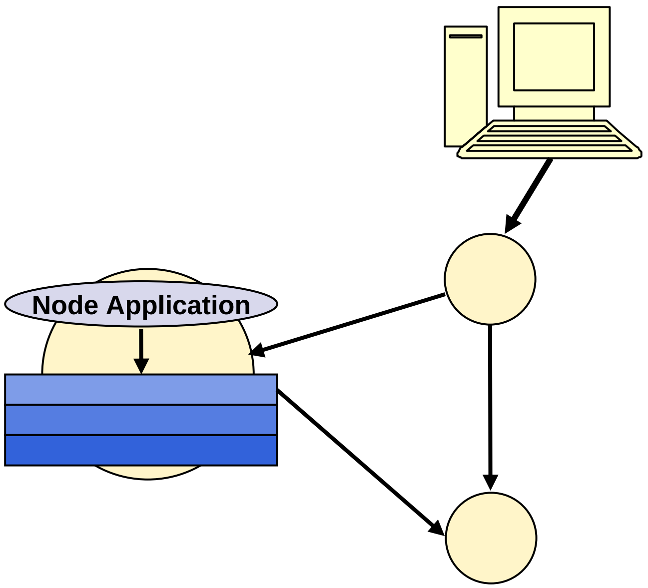
Mires

Comm. Phases



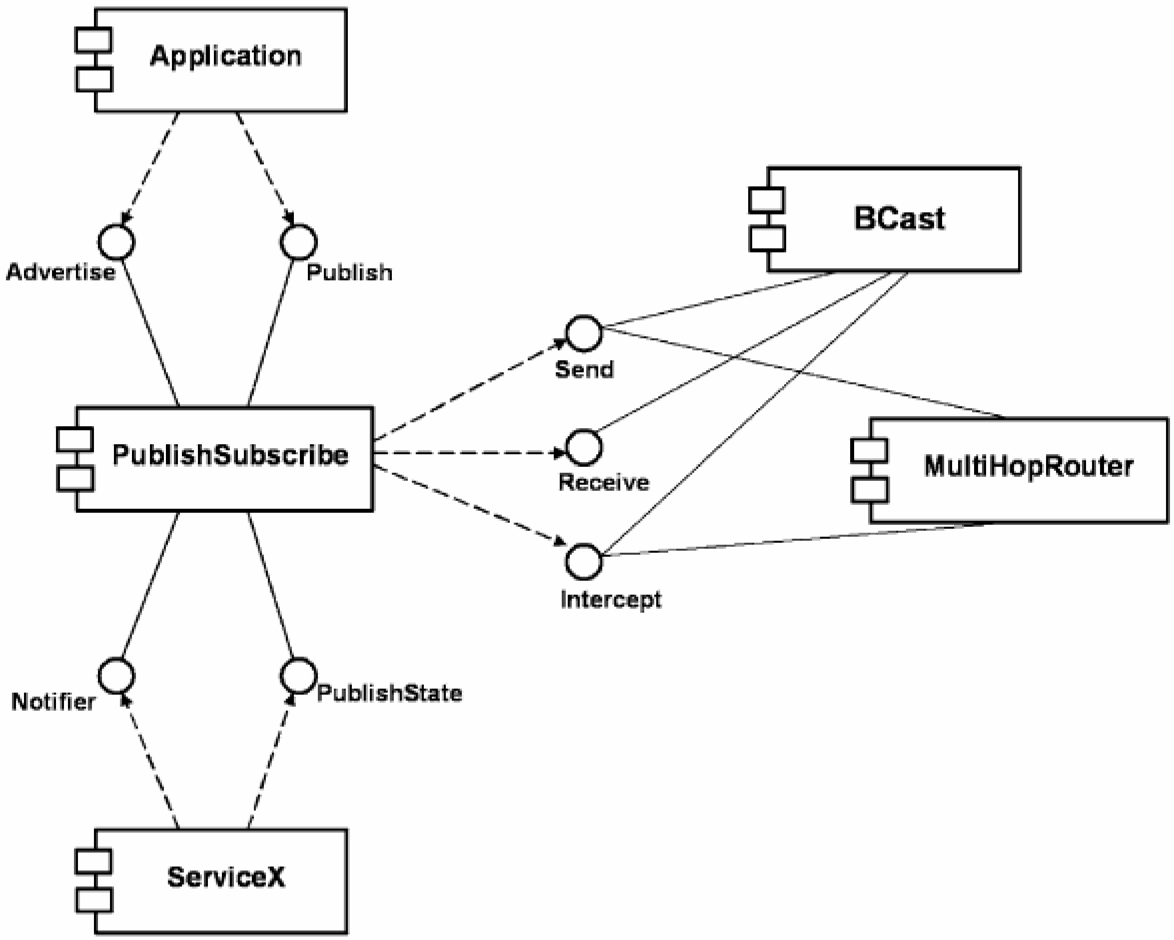
Mires
Events

- topicArrival
  - signals that the node application has submitted data collected from sensors
- stateArrival
  - Similar to topicArrival
  - Data come from network
- topicSetupArrival
  - The subscribe message broadcasted by the user application

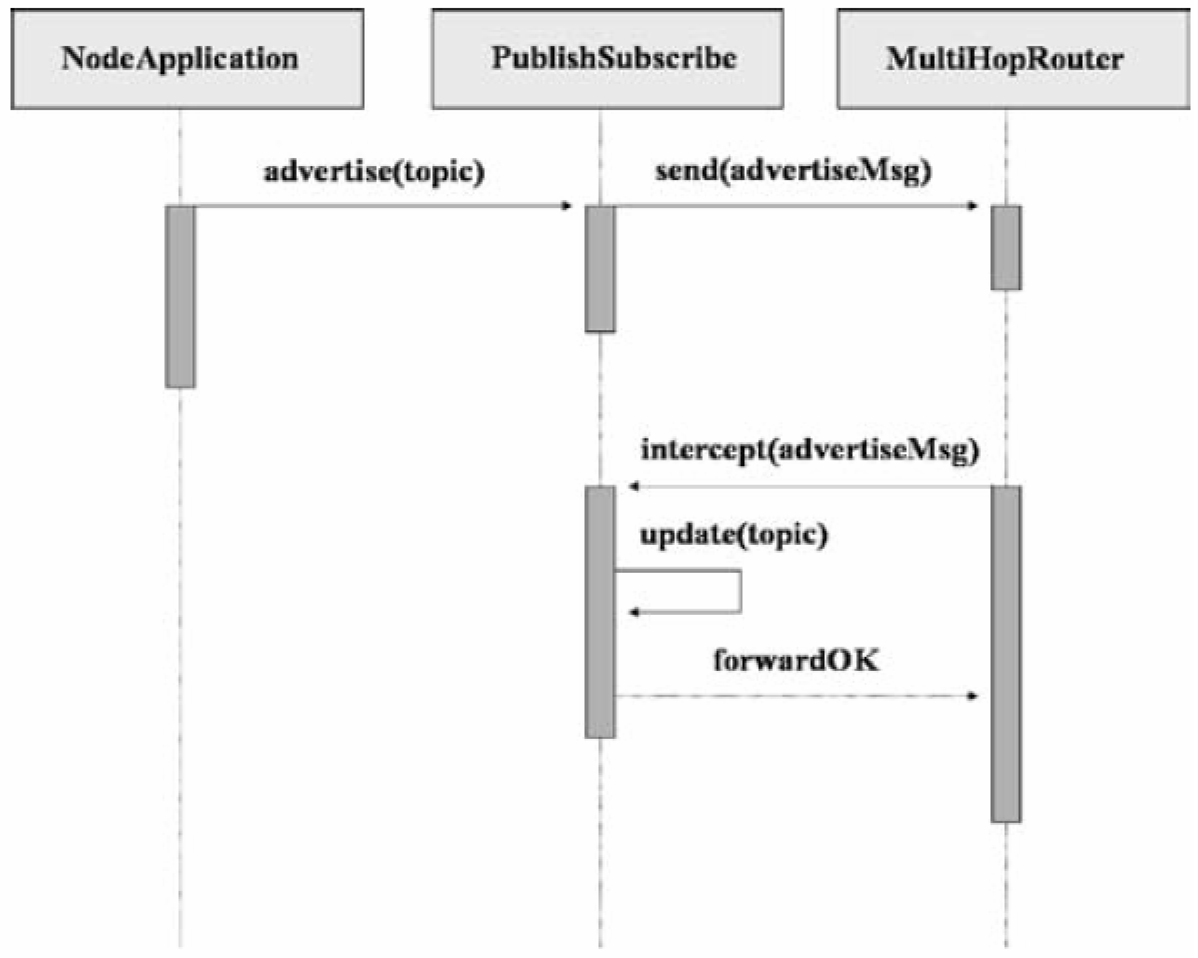


Mires

Components

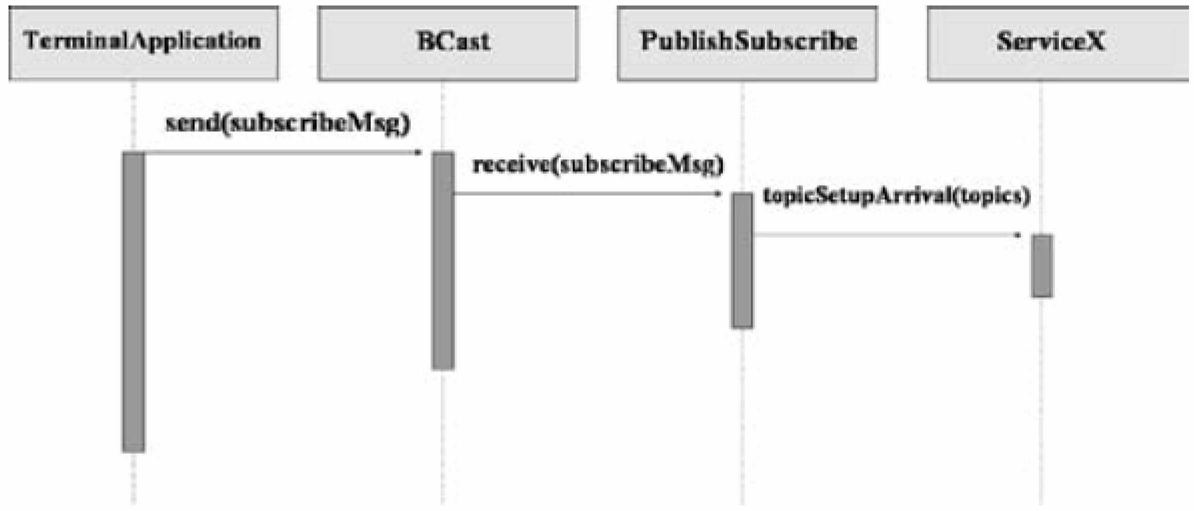


Mires
Topic Advertising

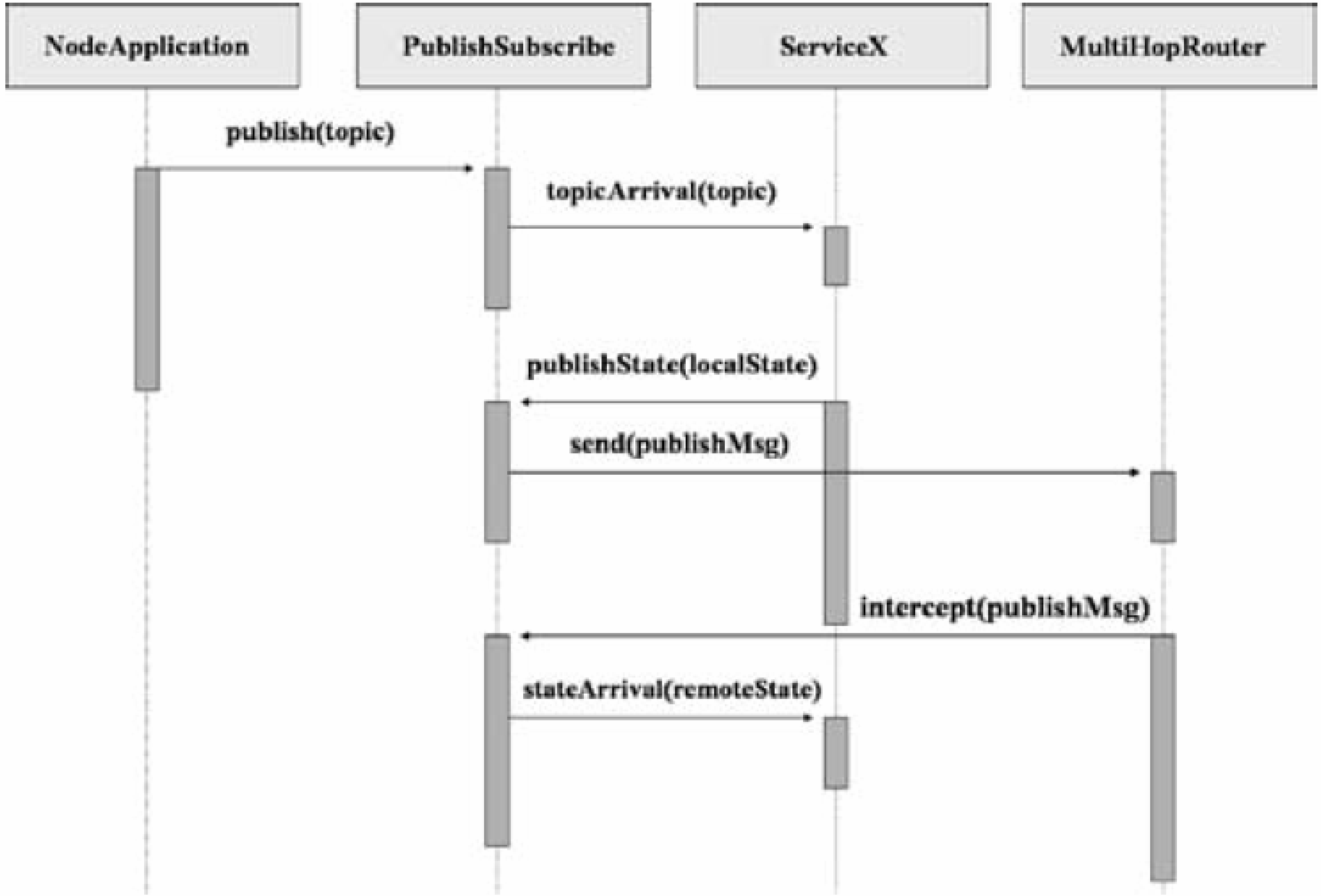


Mires

Topic Subscription



Mires
Data Publishing



Mires

Advantages and Drawbacks

- Simple architecture
- Lightweight components
- Aggregation service for energy saving
- For fixed networks
- Topics of interest are unchangeable
- No subscription language

Siena

- Introduction
- Basic Features
- Architecture
- Routing Strategies

Introduction

- Scalable Internet Event Notification Architecture (Siena)
- Developed at the University of Colorado
- Balances expressiveness with scalability
- Considered best-effort content-based WAN routing

Basic Features

- Event Notification (event):
  - set of attributes
  - each attribute has a type, name and value
  - attribute types belong to predifined set of primitive types found in programming language
  - Example:

    string class=finance/exchange/stock

    time date=Mar 4 11:43:37 MST 1998

    string symbol=DIS

    float change= -4

Basic Features

- Filter (event filter)
  - selects event notification based on criteria
  - specifies a set of attributes and constraint on attribute values
  - each attribute constraint: a tuple
  - specifies a type, a name, a binary predicate operator and an attribute value
  - Example:

    string class>*finance/exchange

    string stock = ”CDE”

    int value > 1.0

Basic Features:

- Patterns
  - is matched against one or more notifications based on both attribute values and their combination

Basic Feature:

- Advertisements
  - Motivation: to inform the event notification service about which kind of notifications will be generated by which objects of interest

Siena Architectures

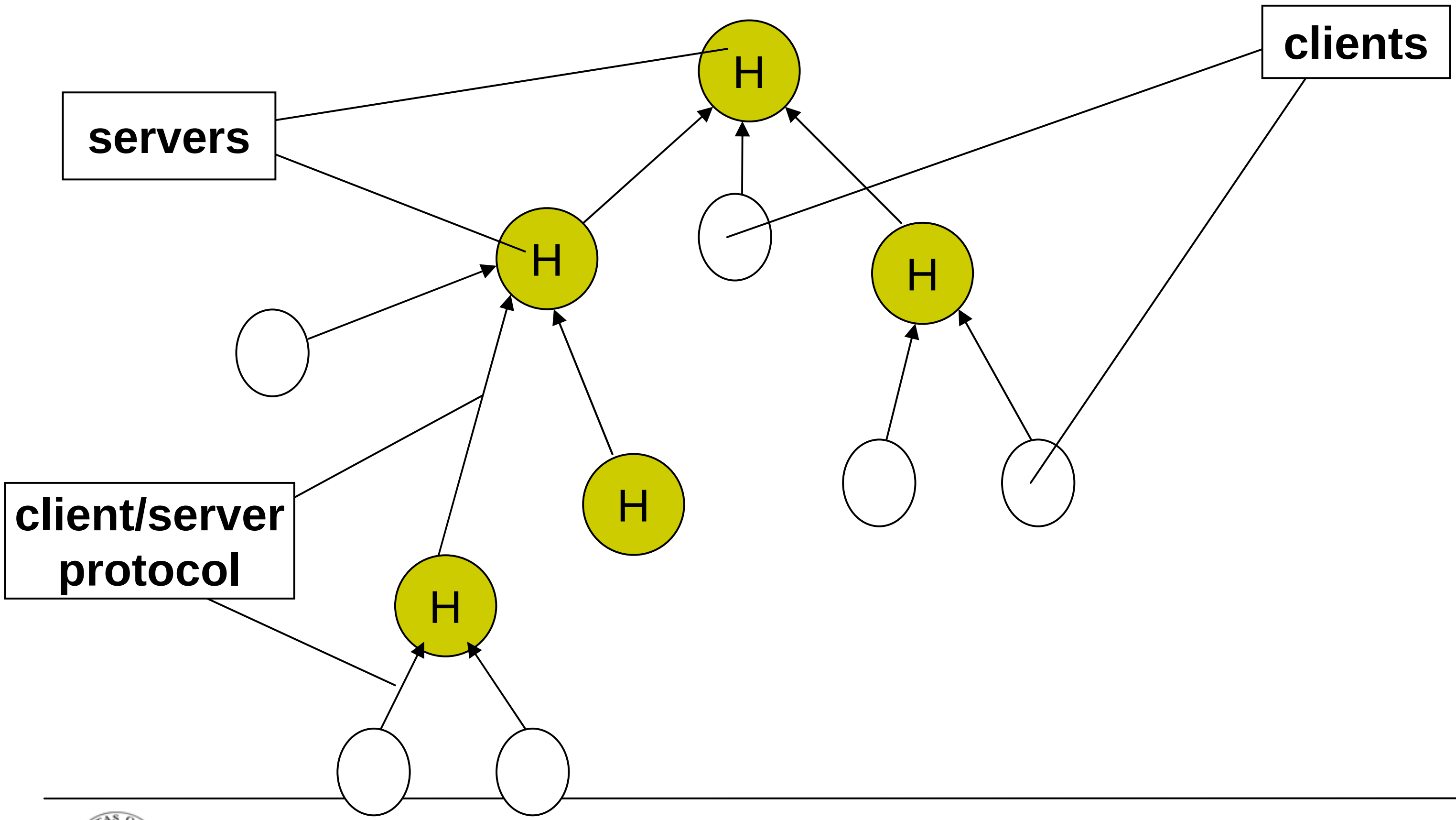
- Server Topology

  Three basic architectures:

  - Hierarchical client/server
  - Acyclic peer-to-peer
  - General peer-to-peer

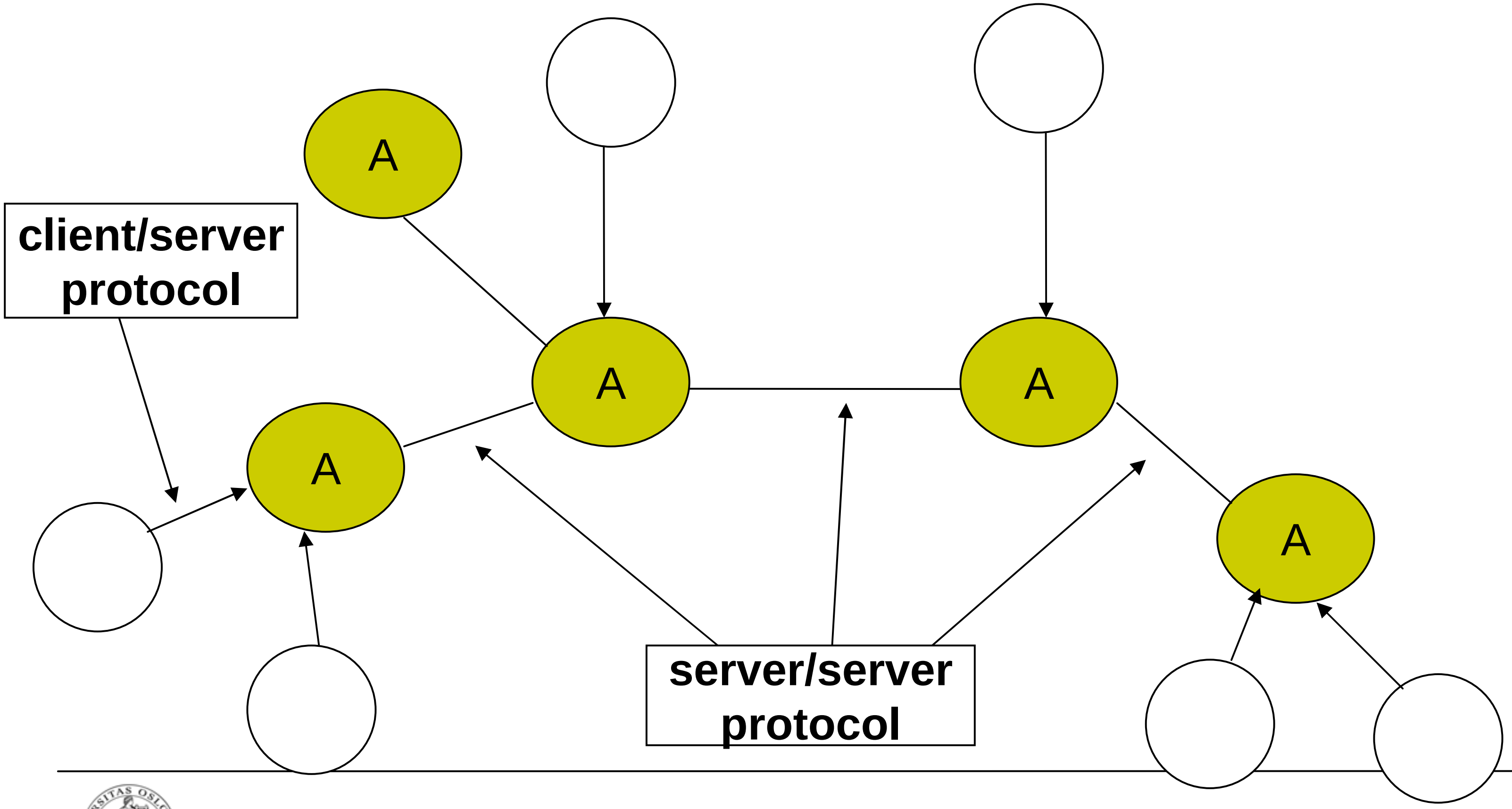
Siena Architecture

- Hierarchical Client/Server Architecture



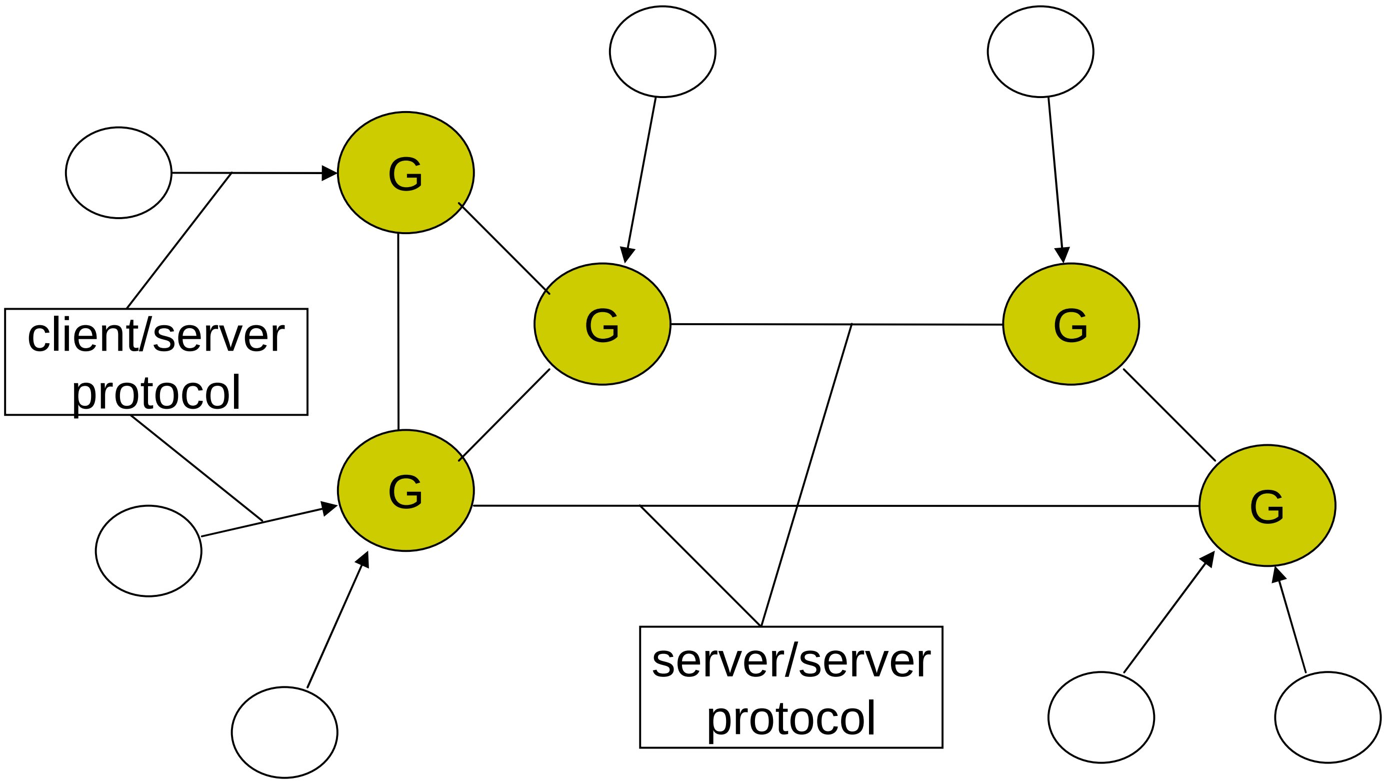
Siena Archicture

- Acyclic Peer-to-Peer Architecture



Siena Architecture

- General Peer-to-Peer Architecture



Routing Algorithms and Processing Strategies

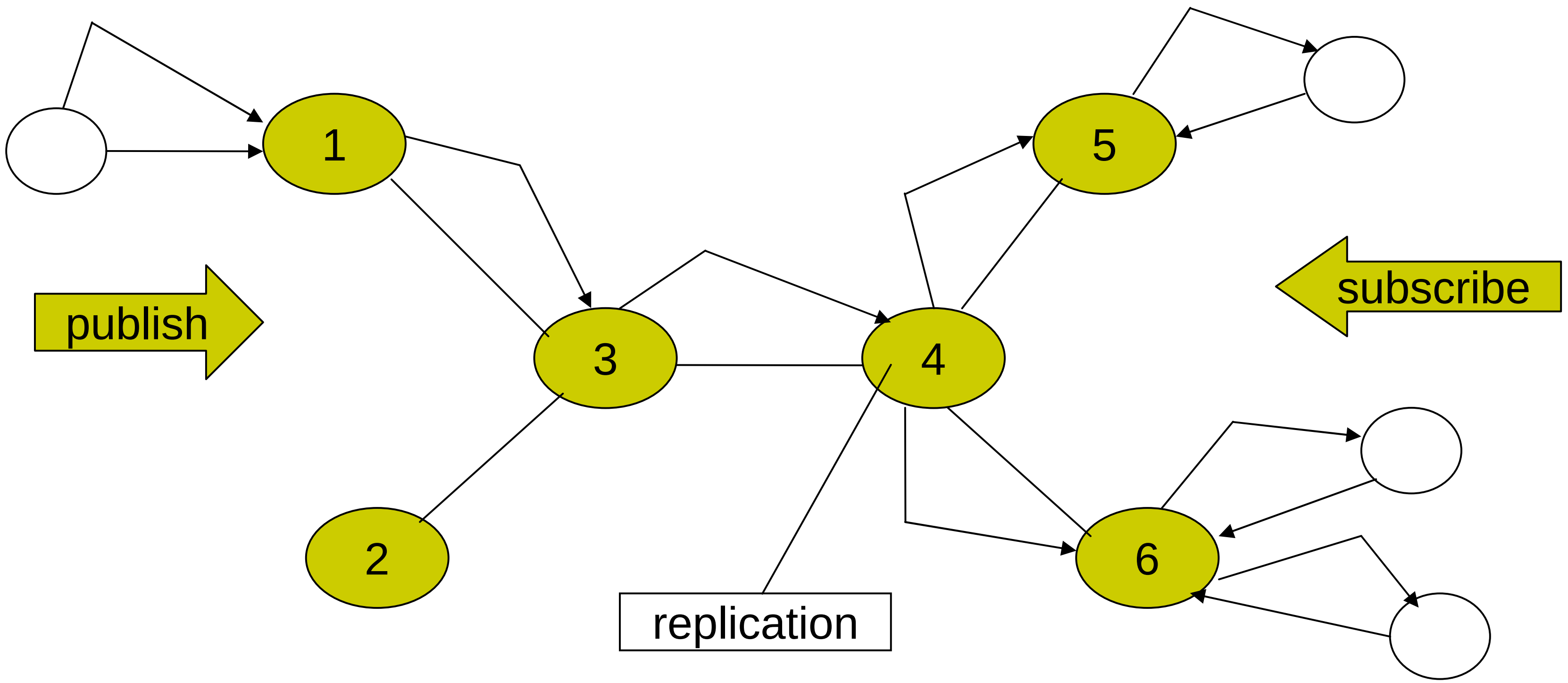
- Devise more efficient routing algorithms
- Principles of IP mutlicast routing protocols employed
- Main idea: to send a notification only toward event servers with clients that're interested in that notification
- Possibly using shortest path
- Same principle applies to patterns of notifications as well

Routing Algorithms and Processing Strategies

- Two generic principles for routing algorithms
  - ***Downstream replication***: a notification should be routed in one copy, and should be replicated only downstream – as close as possible to the parties interested

Routing Algorithms and Processing Strategies

Downstream replication

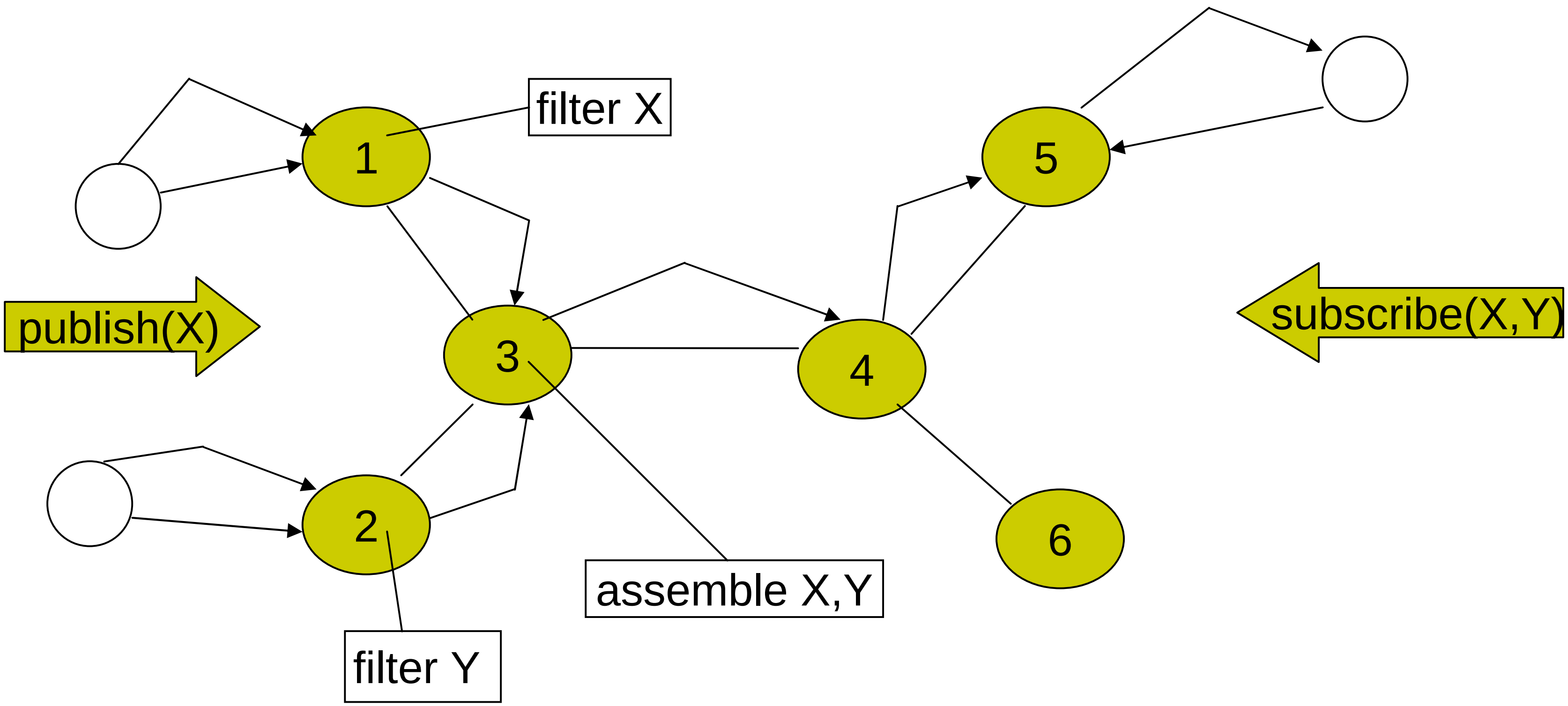


Routing Algorithms and Processing Strategies

- Two generic principles for routing algorithms
  - ***Upstream evaluation***: filters are applied, patterns are assembled upstream as close as possible to the sources of (patterns of) notifications

Routing Algorithms and Processing Strategies

Upstream evaluation



Gryphon

- Yet another Pub-Sub system
- Interesting for its elegant event/subscriber matching algorithm

What does it provide?

- Attribute-based system, like Siena.
- Subscribers give a set of criteria events have to match

- The interesting part:
  Matching an event to the interested subscribers takes less that O(N) time

Attributes vs Groups

- In a group-based system, subscribers ask for events from one or more groups
- Each event belongs to one or more groups.

- This is less powerful than attributes: Attributes can easily be used to emulate groups.
- It is, however, very cheap to implement.

The Gryphon idea

Since attributes can do everything groups can and more, a fast attribute-based solution would be everything we would ever need.

In order to make it work, some assumptions:

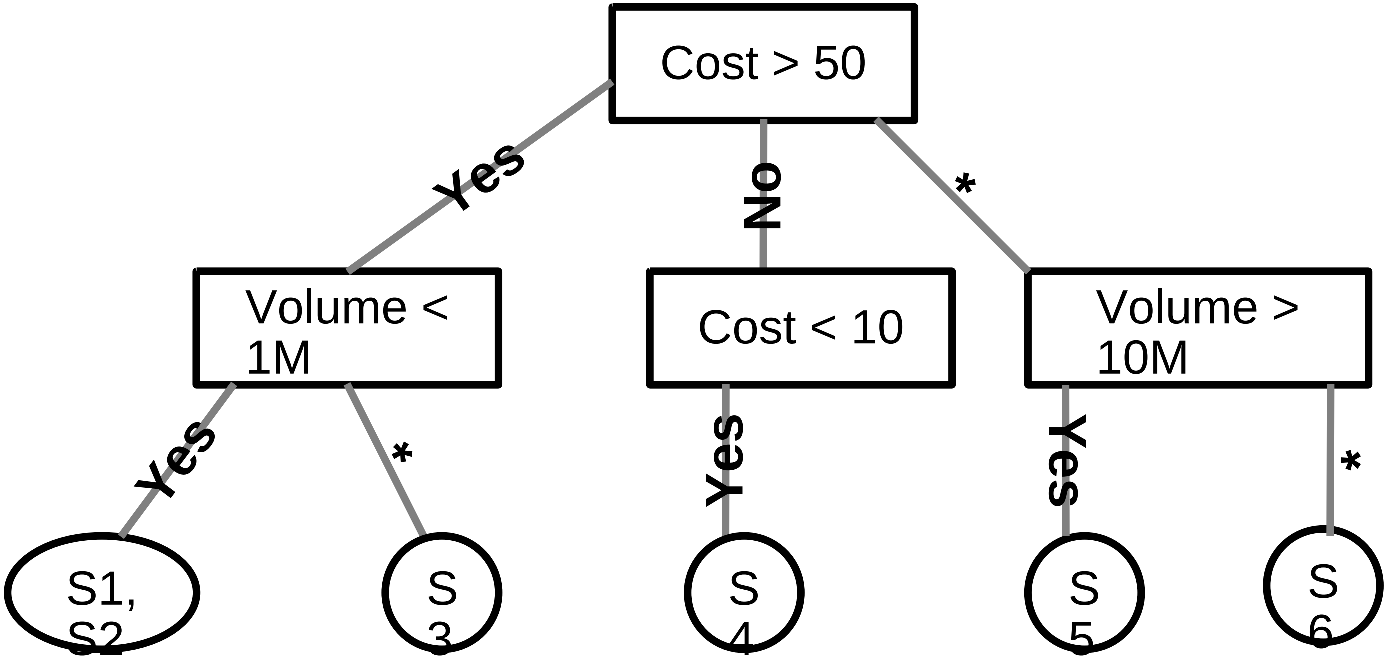
- Events are much more common than subscription changes
- Doing extra work for subscription changes in order to do less per event is worth it.

How does it work

- Tree-based
- Build a tree representing the subscribers at the start
- Add to it when needed

- When an event arrives, follow the matching path(s) in the tree to get to the leaf nodes with subscribers.

The tree



S1 = { cost > 50, volume < 1M }
S3 = { cost > 50 }
S4 = { cost < 10 }
S5 = { volume > 10M}
S6 = { }

Constructing the tree

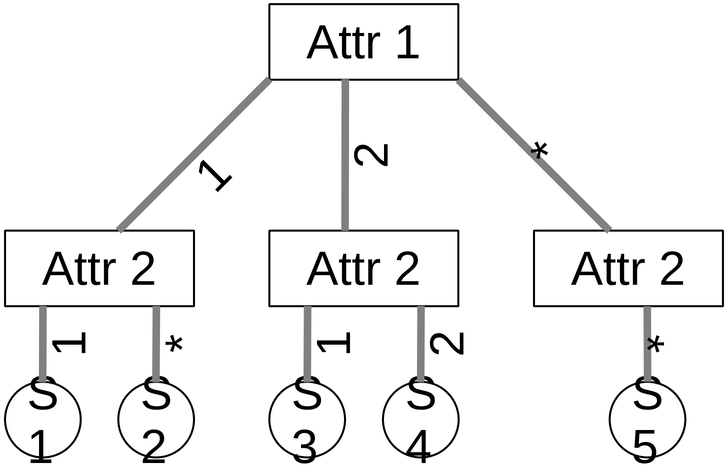
- While a basic assumption is that we can afford a bit more time when changing the tree than per event, it should still scale ok.
- The method used is to add each subscriber separately. In the worst case, this adds as many nodes to the tree as there are attributes – a constant number.
- Thus, it will at most use O(N) time and space.
- Adding another subscriber while running is cheap.

Complexity

- The absolute worst case is that every subscriber adds as many nodes as there are attributes.
- However. The *average case is better (obviously). It depends on assorted factors, but is very generally* $O(N^{1-\lambda})$*, where* $\lambda < 1$*,and usually* > 0*.*

Equality-only tree

- If all tests are for equality, the tree can be simplified to have one level per attribute:



Performance and use

- In 1999, a fairly rough version in java was tested on a P1 MMX, 100MHz.
  With 25000 subscriptions, it used <4ms per event, which leaves time for 250 events/sec.
- Since then, it has been implemented as a full pub/sub system (IBM Gryphon), and integrated into a larger message delivery system.
- As an example, Gryphon was used to deliver all data about results and other events to the audience and press during the 2000 summer olympics (the score boards were among the subscribers).

Questions?

That was all – any questions?