Publish / Subscribe Systems

A Summery of:
Eugster, Felber, Guerraoui, Kermerrec:
The Many Faces of Publish / Subscribe

Oliver Hohlfeld, Andre Schröder
January 2007

Definition

"**Subscribers register** their **interest** in an **event** [..] and are subsequently asynchronously **notified** of events generated **by publishers**"

Outline

1. The Publish/Subscribe Paradigm
2. Decoupling
3. Alternative Paradigms
4. Publish-Subscribe Variations
5. Implementation Issues

The Public/Subscribe Paradigm

Motivation

- Large scale **distributed systems**
  - Nodes differ in
    - Location
    - Lifetime
    - Latency
    - ...

  - → **Loosely coupled** form of interaction required
    - Participants operate independent from each other
    - Scalability

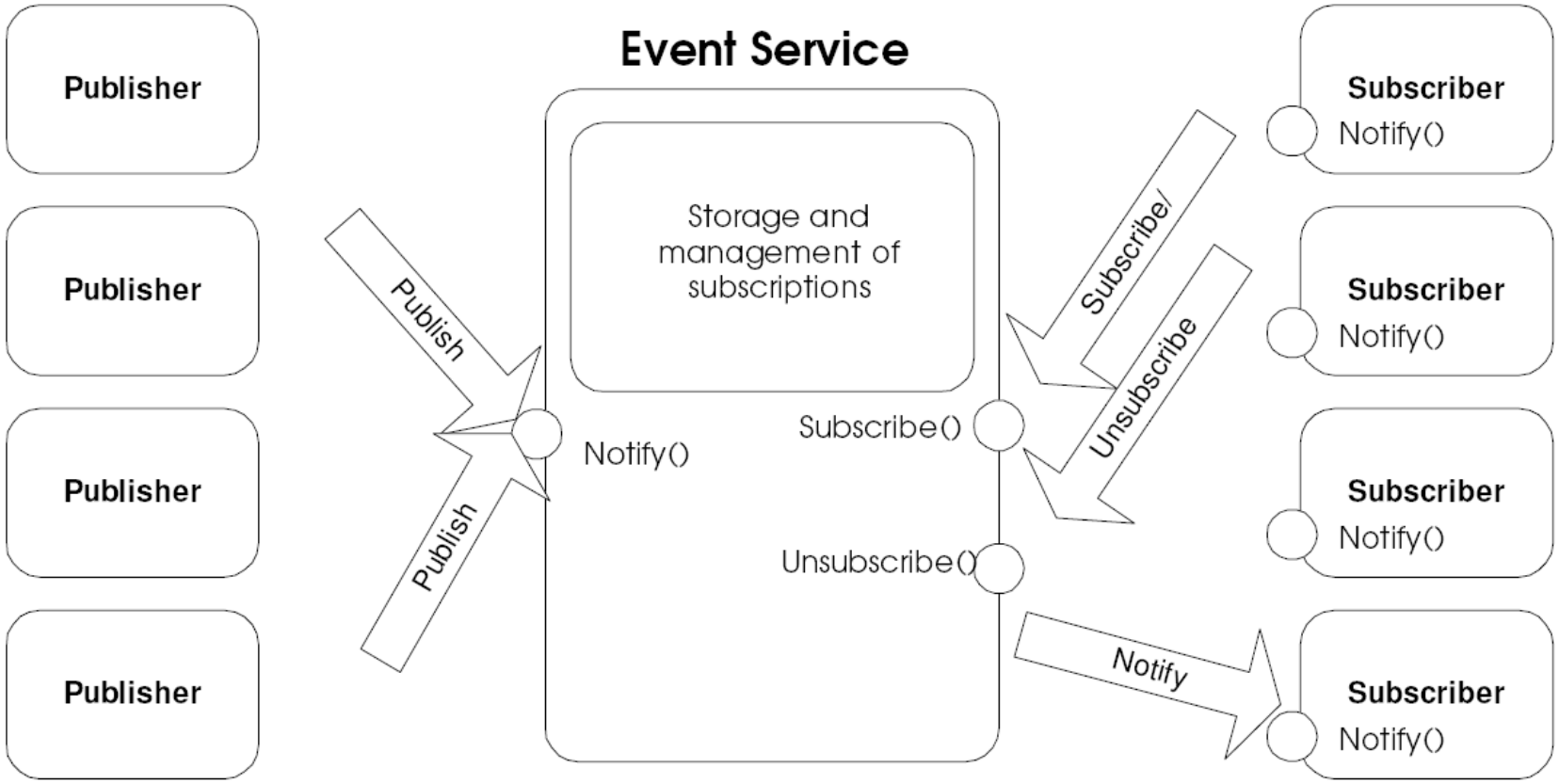
Publish / Subscribe Scheme

1. Subscribers **register interest** in an event, or a pattern of events

2. Producers **publish information**
   - Publisher = Generator of events

3. Subscribers get **asynchronously notified** of events
   - Subscriber = Consumer of events

Basic Interaction Scheme



Event Service

- Ways to describe:
  - Neutral Mediator
  - Layer of indirection
  - Proxy for subscribers

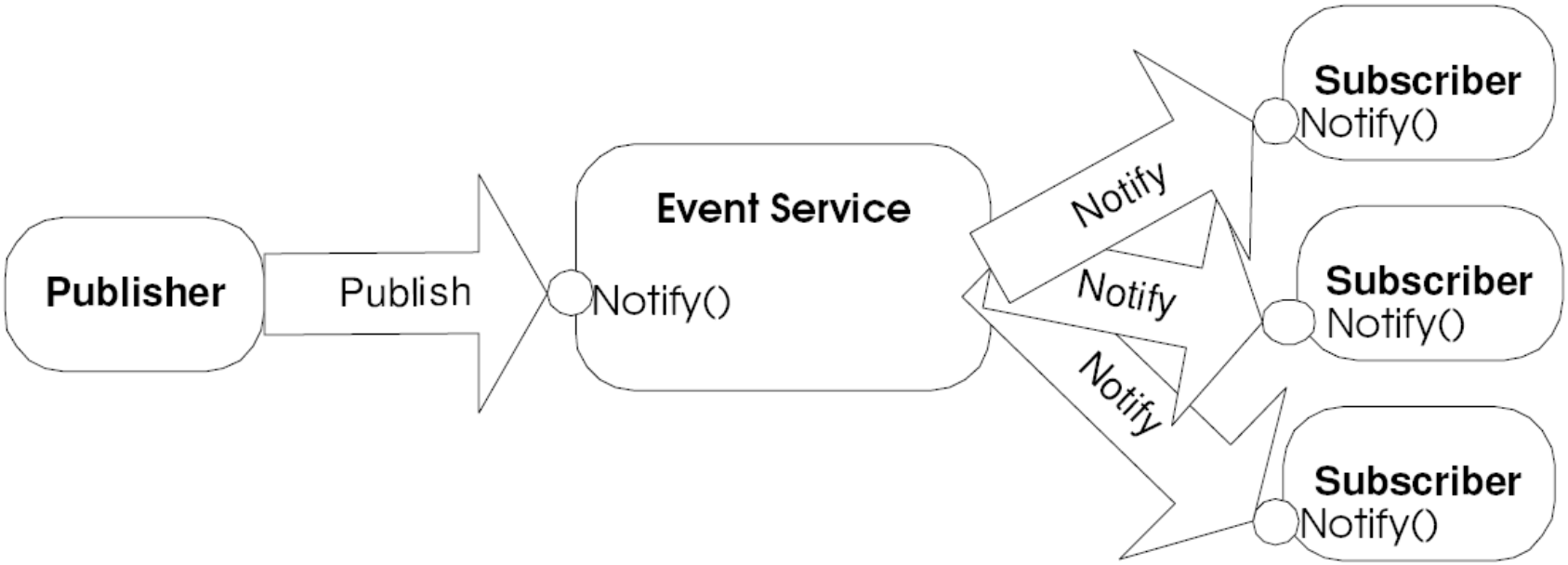
  → decouples between publisher and subscriber

Decoupling

Decoupling

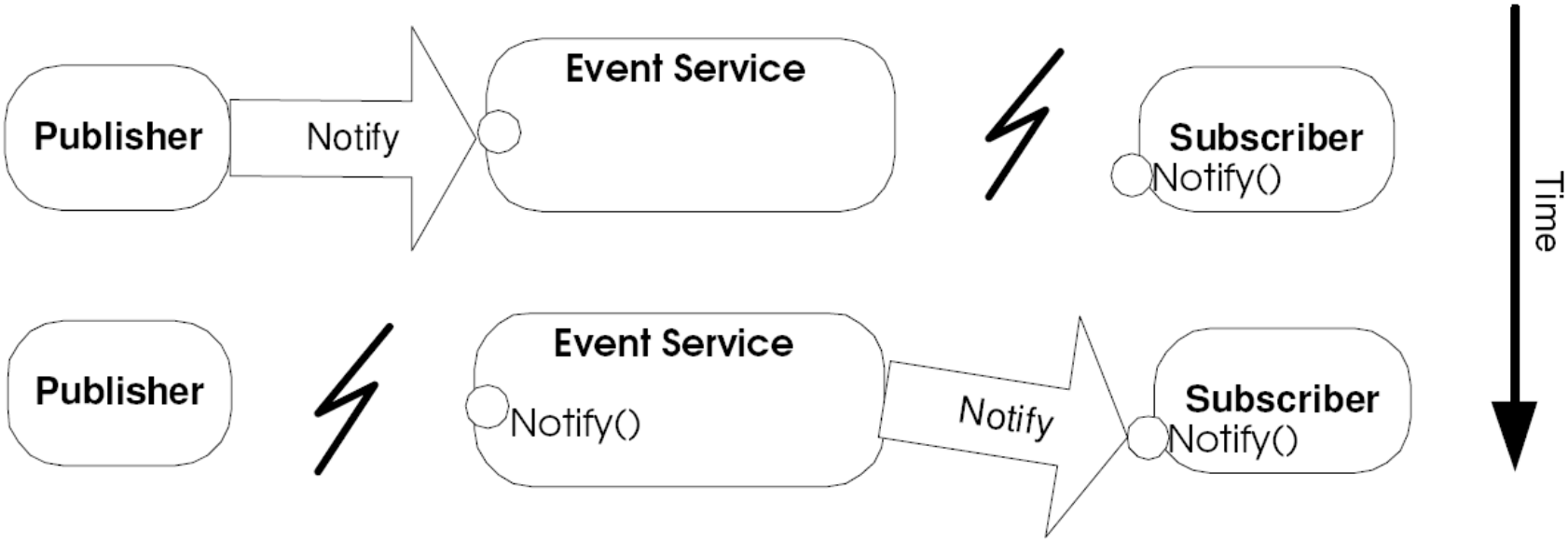
- Main strength of Publish / Subscribe Systems
  - Information generation and consumption independent from each other
- Decoupling in terms of
  - **Time**
  - **Space**
  - **Synchronization**

Space Decoupling



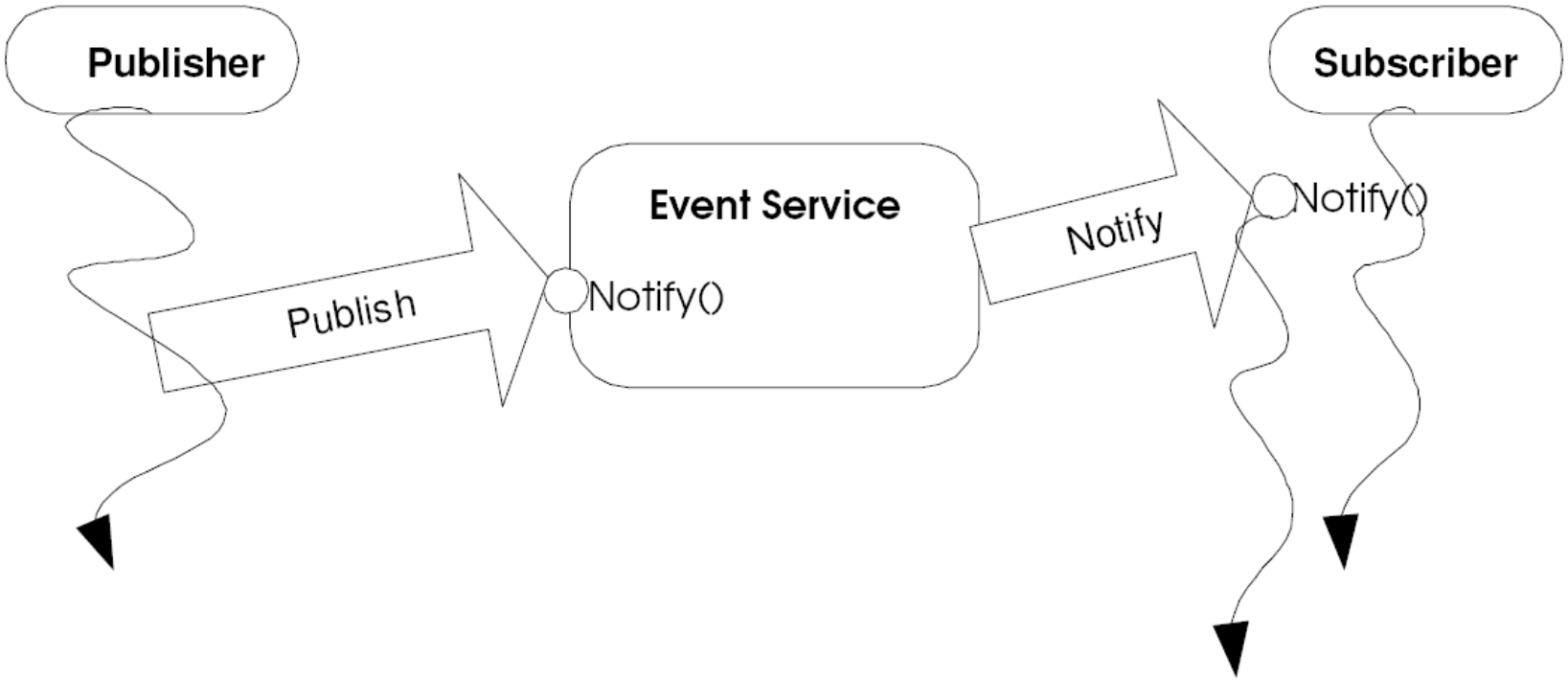
- Interacting parties don't know each other (ref.)
  - **Publisher** does not know its subscribers
  - **Subscriber** does not to know publishers
  - **Both** don't know number of participants

Time Decoupling



- Parties don't need to be active at the same time
  - Publisher can generate events when a subscriber is disconnected.
  - Subscriber can be notified when publisher is disconnected.

Synchronization Decoupling



- Asynchronous notification of subscriber
  - Event services calls back subscriber (when online) ≠ pull of subscriber
  - Not in main flow of control of publisher/subscriber

Decoupling - Summary

- Decoupling in terms of
  - **Time**
  - **Space**
  - **Synchronization**

  - ➔ Increases scalability
    - By removing dependencies

  - ➔ Well adapted to distributed environments that are asynchronously by nature

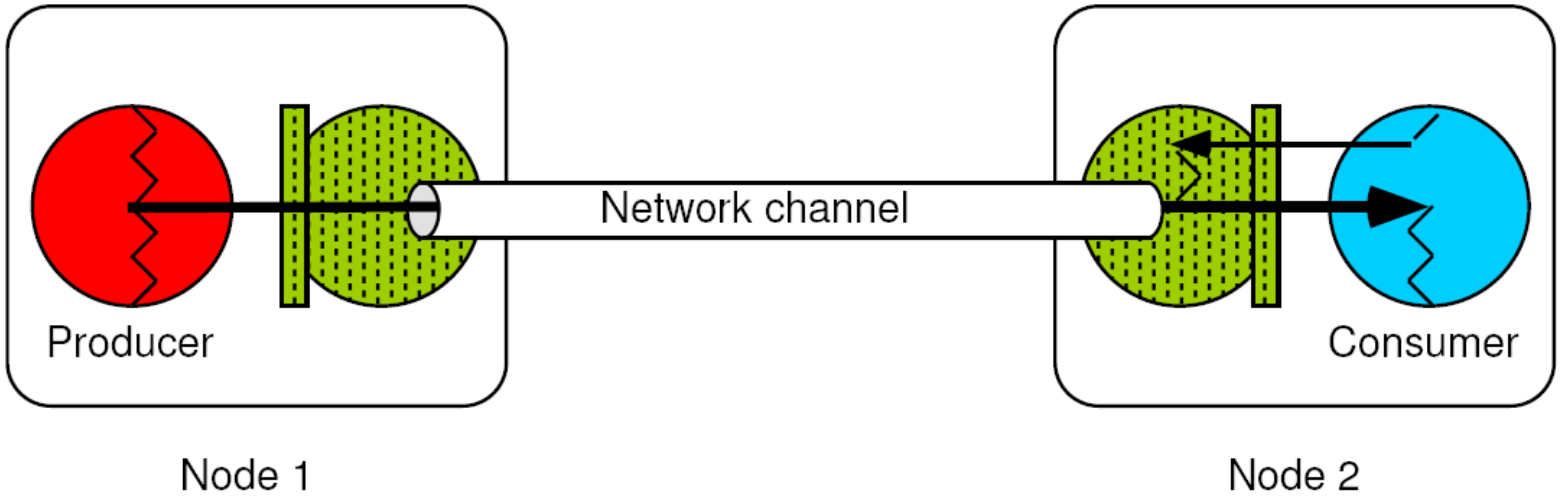
Alternative Paradigms

Alternative Paradigms

1. Message Passing
2. Remote Procedure Call (RPC) / RMI
3. Notifications (Observer Design Pattern)
4. Shared Spaces
5. Message Queuing

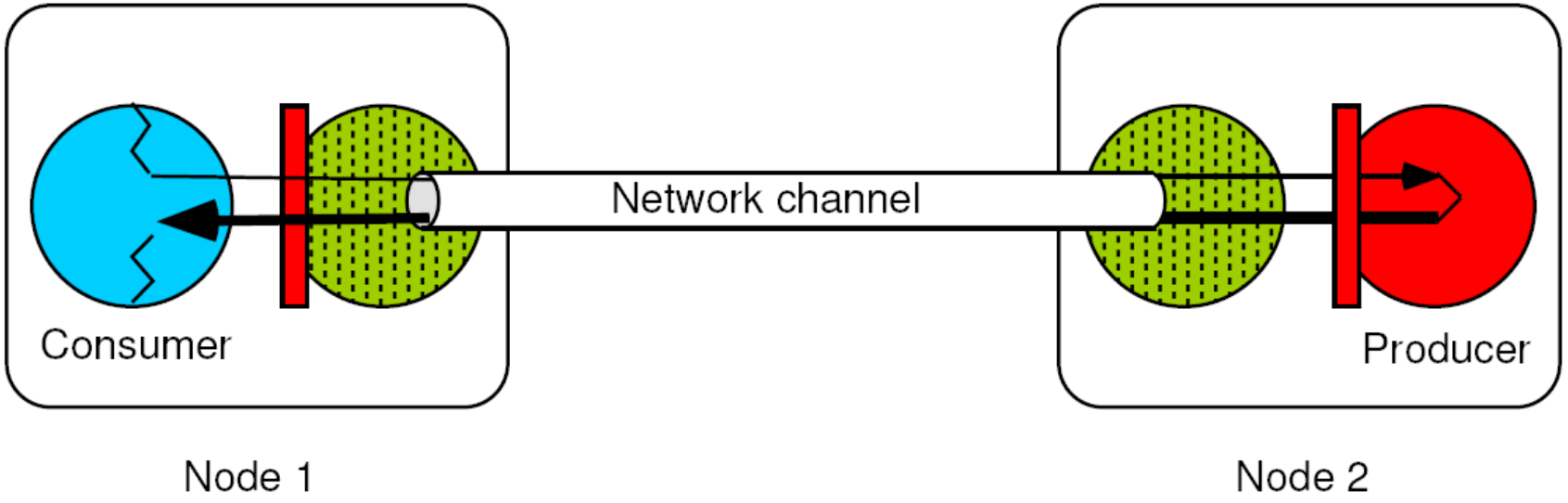
➔ Partially discussed in this course

Message Passing



- Low-level form of communication
  - Participants exchange messages directly
    e.g. by using **sockets**
- Producer&Consumer coupled in time and space:
  - Must be active at the same time
  - Must know each other

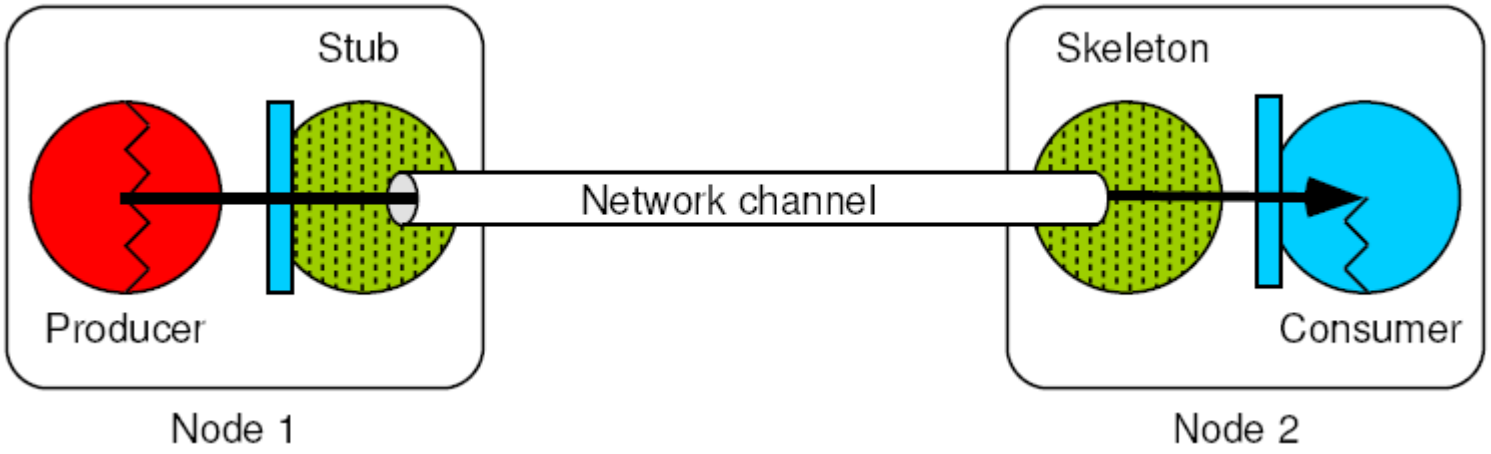
RPC / RMI



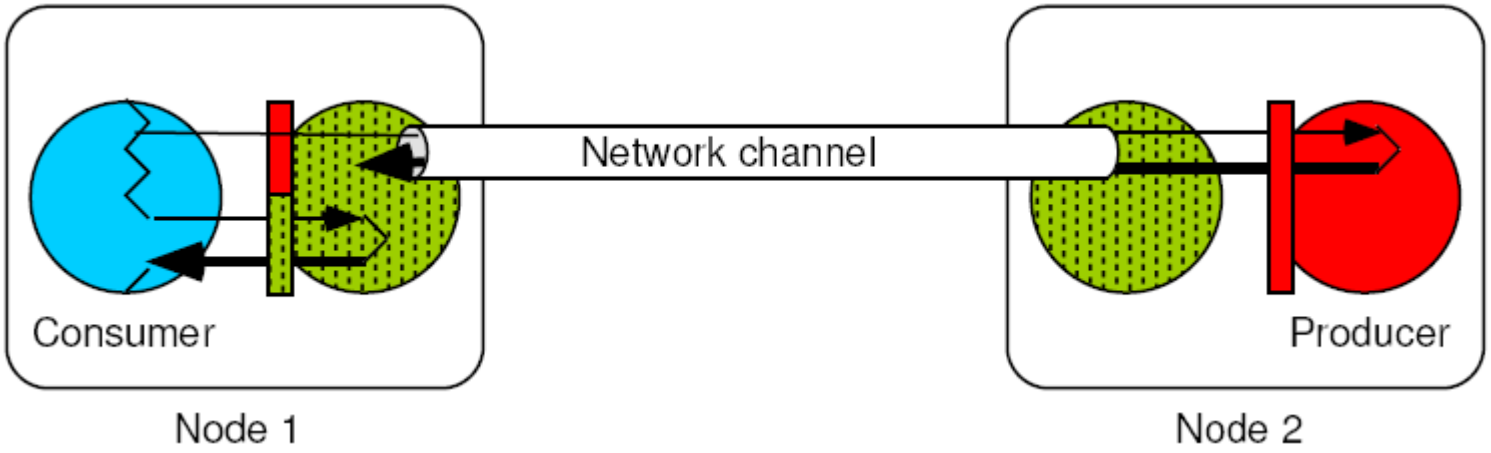
- Basic RPC is synchronous
  - Strong **time** and **synchronization** (consumer) coupling
- Objects hold remote reference
  - **Space** coupling

RPC / RMI (asynchronous)

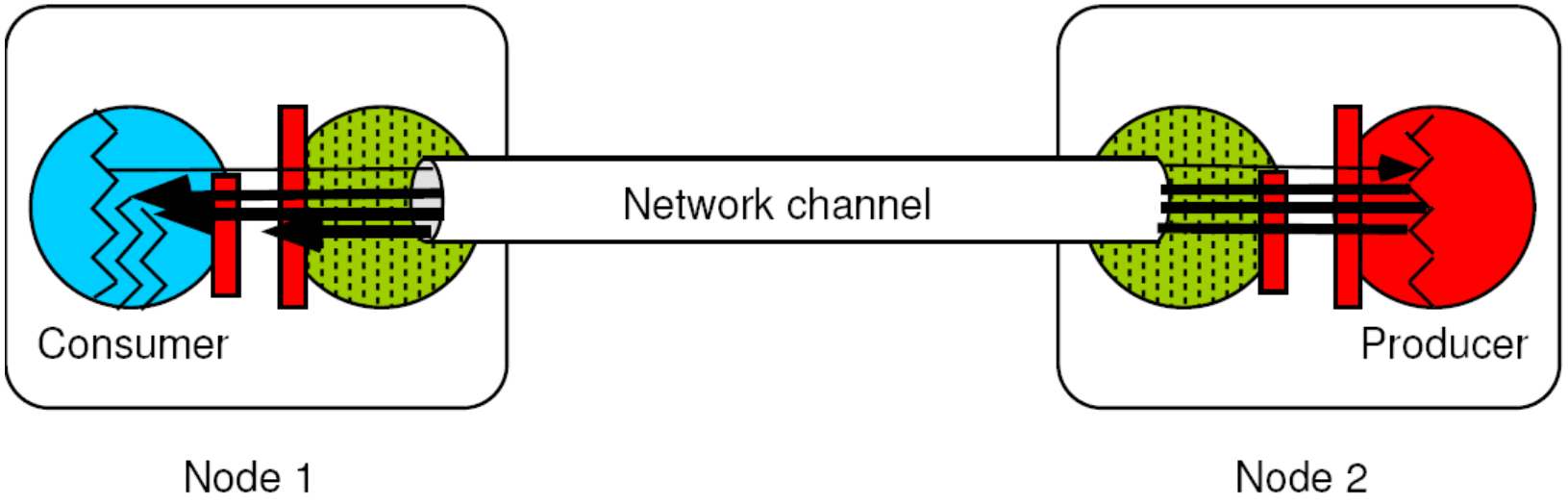
- Without reply (fire & forget)



- With extra-thread (non-blocking)



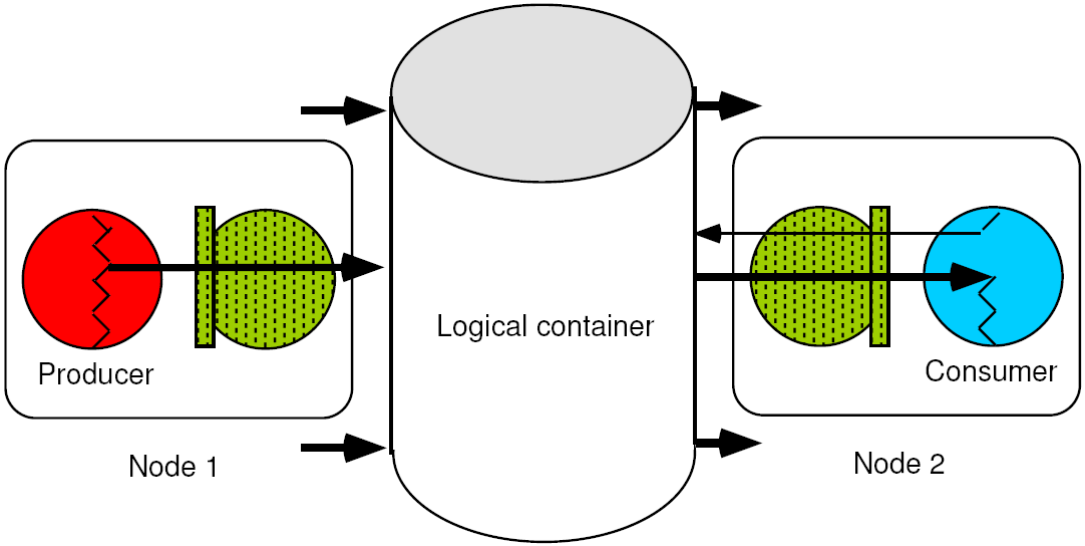
Notifications (Observer Pattern)



- Consumer passes callback reference to Producer

- Time & space coupling
- Synchronization decoupling

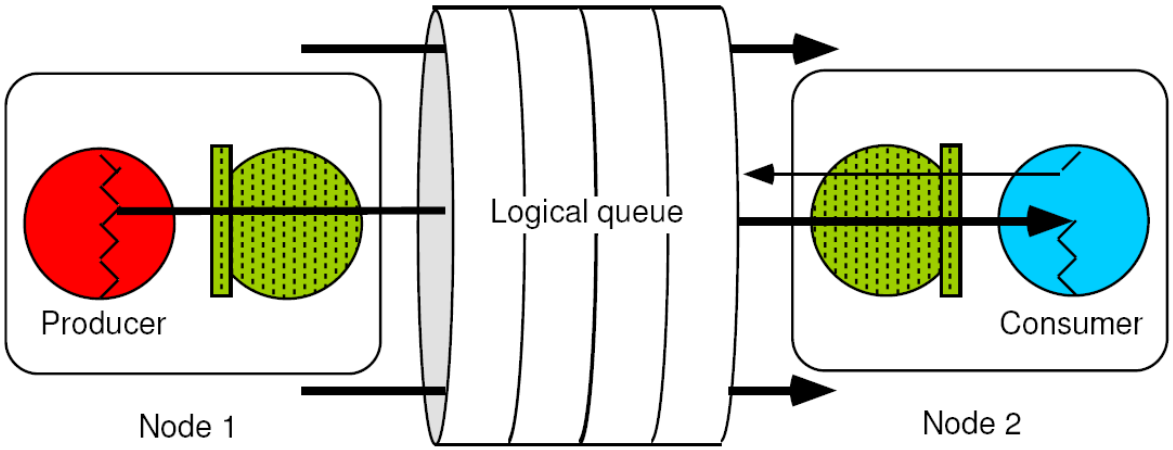
Shared Spaces



- Producer
  - Inserts entities ***asynchronously*** into container
- Consumer
  - Pulls (=reads ***synchronously***) from container
  - ➔ Not event based

- Time & space decoupling

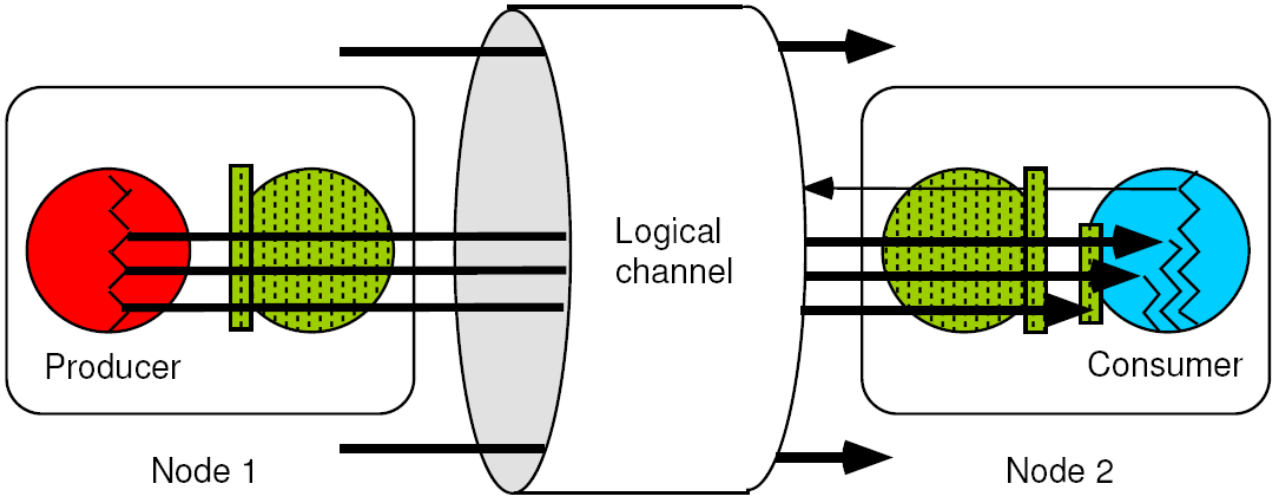
Message Queuing



- Producer
  - Inserts entities ***asynchronously*** into **FIFO**
- Consumer
  - Pulls (=reads ***synchronously***) from **FIFO**
  - ➔ *One-of-n* semantics

- Time & space decoupling

Paradigms - Conclusion



- Only **Publish/Subscribe** provides decoupling in terms of
  - Time
  - Space
  - Synchronization

Publish/Subscribe Variations

Publish/Subscribe Variations

Not all events are of interest.

How to subscribe to ...

- particular events,
- event patterns?

Variants:

- Topic-based
- Content-based
- Type-based

Topic-Based

- Subscribe to **topics**
  - Identified by keywords (strings)
    - platform interoperability
  - Similar to groups
    - Becoming member of an event group
- Improvements
  - Topic hierarchies
    - Subscription also involves subscription to existing sub-topics
  - Wildcards
    - Subscription to set of topics

Topic-Based - Example

- Topic names usually expressed in a **URL-like** notation, e.g.
  - Notify me when something new is published, concerning the DSMWare course

    ➔ *Eurecom/Courses/DSMWare*

Topic-Based - Issues

- *Static scheme with limited expressiveness*
  - Topic = pre-defined criterion
  - Does not say anything about the specific content of an event

Content-Based

- Considers **content of events**, e.g.
  - Internal Attributes of data structures
  - meta-data of events
- ➔ More dynamic than topic based

- Subscription language used, e.g.

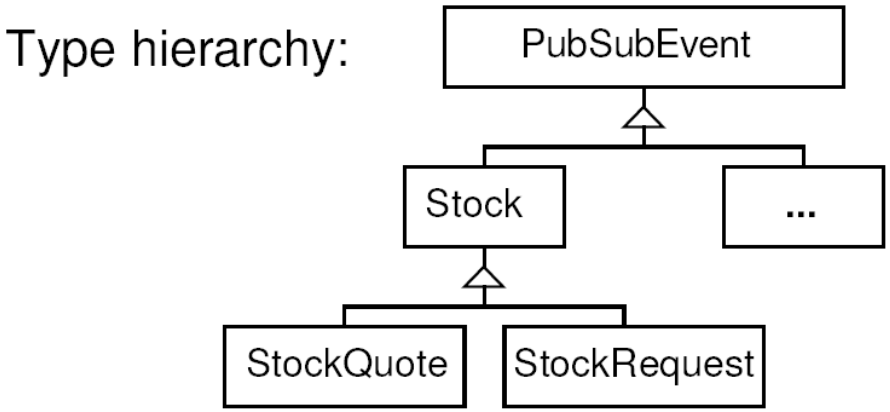
  `"Course=DSMWare and Grade>10"`

Content-Based (cont.)

- Ways to specify patterns:
  - Strings
    - SQL, XPath for XML, ...
  - Template Objects
    - Event for each entity that equals the template object
    - Wildcards via “null”
  - Executable code
    - Consumer provides pattern matcher
    - Not used in practice (due to efficiency, scalability)

Type-Based

- Don't filter according to a name but **type**
  - Type in the notion of programming languages
  - Integrates programming language in middleware
- Ensure **type safety** at compile-time
- Includes **content-based** filtering



Variations - Summary

- Different publish/subscribe variants
  - Different **degrees of expressiveness**
    - High: content-based
    - Low: topic-based
  - Difference **performance**
    - High: topic-based
    - Low: content-based

- Variant selection
  - Limited discrete properties -> static scheme
  - Non-discrete properties (price) -> content-based

Implementation Issues

Implementation Issues

- Aspects of publish/subscribe
  - Events
  - Media
  - Quality of Service
- Tradeoffs
  - Flexibility
  - Reliability
  - Scalability
  - Performance

Events

- Lowest Level
  - **Messages**
    - Explicitly created by application
    - Most have header
    - Text or XML format
- Higher Level
  - **Invocations**
    - Well defined interface & semantics

Media

- Architecture
  - Centralized
  - Distributed
  - Distributed Network of Servers
- Data Transmission
  - Point-to-Point
  - Multicast

Media - Architecture

- **Centralized Architecture**
  - Central storing and forwarding of messages
  - Producers send messages to this entity
  - Events will be forwarded to subscribers afterwards

- Strong requirements in terms of
  - Reliability
  - Data consistency
  - Transaction support
- No need for fast event processing
  - e-Commerce, online-banking

Media - Architecture

- **Distributed Architecture**
  - Decentralized (no central entity)
    - no single point of failure
  - Producers and consumers
    - Both store and forward

- ➔ Strong requirements in terms of
  - Fast and efficient delivery of data
    - Stock exchange, multimedia broadcasting

Media - Architecture

- **Distributed Network of Servers**
  - Combined *centralized* and *decentralized* approach

- Strong requirements in terms of
  - Persistence
  - Reliability
  - High-Availability
  - Routing

Media - Communication Type

- Point-to-point
  - Event is sent to each subscriber separately
  - used in centralized systems
- Multicast
  - Event is sent to a group of subscribers
  - Efficient and scalable for topic-based systems
  - Efficiency in content-based systems is still an open question:
    - How to aggregate subscriptions?

➔ Choice depends on environment and architecture of the system

Quality of Service

- Persistence
- Priorities
- Transactions
- Reliability

Persistence

- Guarantee that messages don't get lost
- Present in centralized systems
  - Messages are stored until subscribers are able to process them
- Not generally offered by distributed systems
  - Producer sends messages directly to all subscribers
  - Subscriber might not be online
    - ➔ Producer needs to keep copy until all subscribers got their message

Priorities

- Deliver waiting messages in order of priority
- Should be considered as **best-effort**
  - Cannot be guaranteed
- Provided by most publish/subscribe systems

- *Example*: Real-time applications
  - Events need immediate reaction
  - Process before other messages

Transactions

- Known from database theory
- Multiple operations are grouped to atomic block
  - Either the entire message sequence is sent / received
  - Or nothing!
- Example: Producer will group semantically-related messages
  - Subscribers should not see a incomplete sequence

Reliability

- Guarantee message delivery
- Centralized systems
  - Keep central copy of messages
  - Use reliable point-to-point communication
  - A failure may only delay the delivery
- Distributed systems
  - Need reliable communication protocols
  - E.g. Reliable Application Layer Multicast
  - Using store&replay

Summary&Conclusion

- publish/subscribe
  - Fits well demands of scalable and loosely-coupled systems
  - Decoupling in
    - Space
    - Time
    - Synchronization
- Tradeoffs in scalability, expressiveness, QoS
  - depends on used architecture and implementation/ protocols

Questions?

References

- 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 http://www.caip.rutgers.edu/~virajb/readinglist/facespublishsubscribe.pdf