COMP90015 Distributed Systems

Indirect Communication

Aaron Harwood

School of Computing and Information Systems

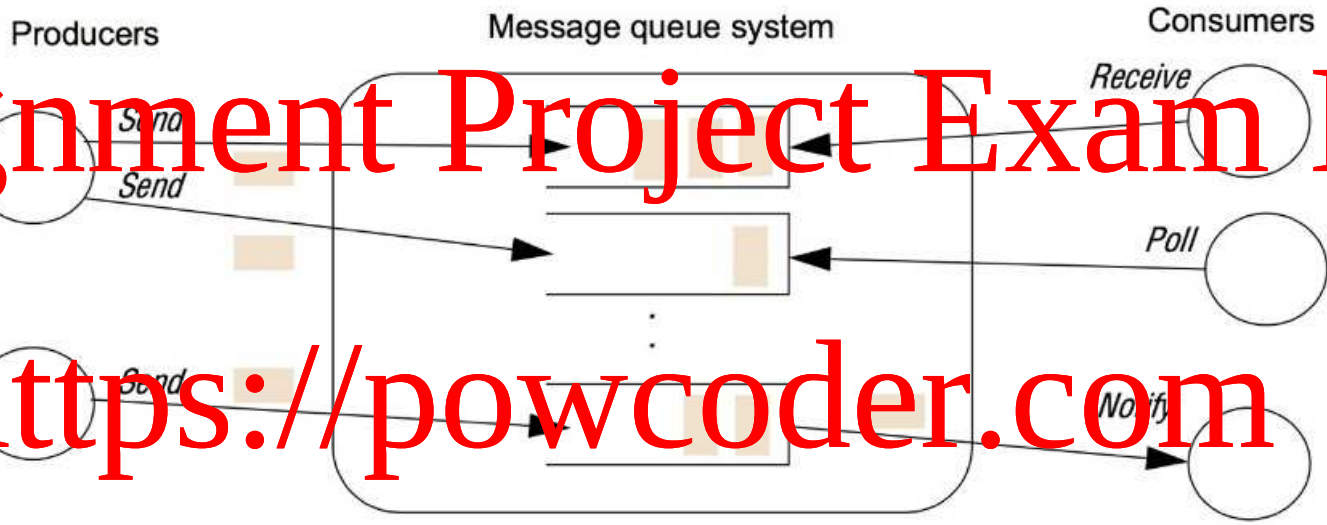
2022 Semester II

1. Message Queues
2. Group Communication
3. Publish/Subscribe
4. Distributed Shared Memory
   - Tuple Spaces
5. Comparison

Indirect Communication

- Whereas direct communication is communication that takes place directly between the communicating processes, *indirect communication* is defined as communication between entities in a distributed system *through an intermediary* with no direct coupling between the sender and the receiver(s).
  - *Space uncoupling* – sender does not know or need to know the identity of the receiver(s)
  - *Time uncoupling* – sender and receiver can have independent lifetimes, they do not need to exist at the same time
- Time uncoupling is not synonymous with asynchronous communication. Asynchronous communication doesn't imply that the receiver has an independent lifetime, in other words we could consider a time coupled asynchronous system.
- Indirect communication paradigms tend to be described in terms of a metaphor that aids in understanding the expectations of the paradigm and for what kinds of distributed applications it is useful:
  - Message Queues
  - Group Communication
  - Publish/Subscribe
  - Shared Memory
  - Tuple Spaces

Message Queues



- Message queues provide a point-to-point service using the concepts of a *message* for data encapsulation and *queue* as an indirection. They are point-to-point in that each message is sent by a single process – *producer* – and received by a single process – *consumer*.
- Since communication uses messages, the message queue paradigm may not be suitable for applications that require streaming data or bulk data transfer.
- Good for distributing units of work to processes and command/control type operations.

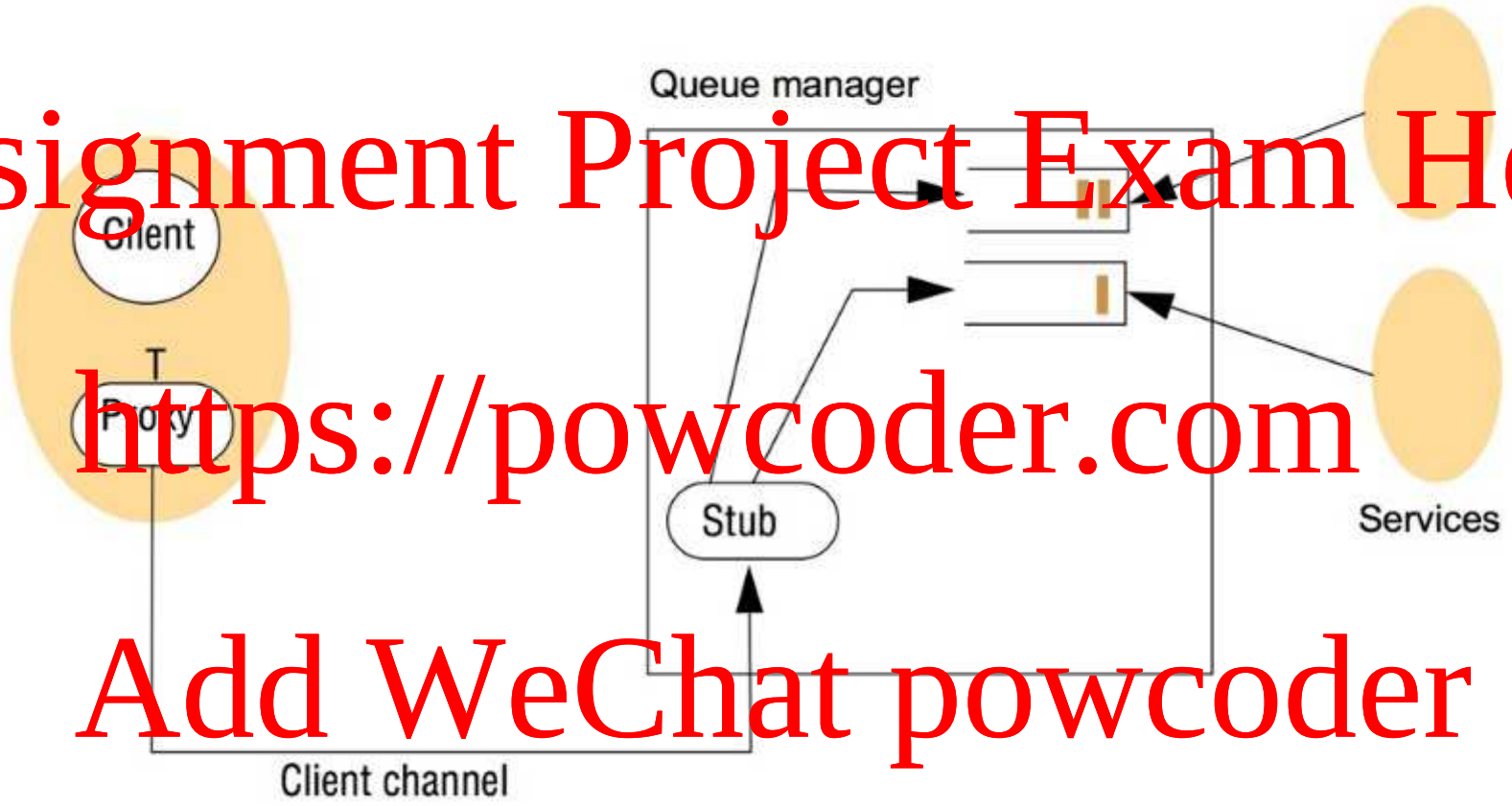
Programming model

Usually the message queue system is expected to provide reliability in that messages are not dropped or lost, and since its a queue, messages are received in the order sent. The API is very much the same as a blocking queue that is used in concurrent programming.

- *send* – producers put a message on a particular queue, may block the sender if the queue has finite capacity.
- *blocking receive* – a consumer waits for at least one message on a queue and then returns.
- *non-blocking receive* – or *poll*, a consumer will check and get a message if there, otherwise it returns without a message.
- *notify* – a signal is sent to the consumer when messages are available on the queue for consumption.

It is useful to consider this API in terms of actual processes and low-level exchange protocols. E.g. the implementation may use TCP for producers and consumers to connect to the queueing system.

Examples



- A message queueing system typically provides a library for the programming to build a client, either a producer or a consumer, and a server implementation that implements the queue manager itself. The server implementation will typically run a process that allows producers and consumers to connect.
- Modern examples include RabbitMQ and ZeroMQ.

Discussion questions

Question (1) Discuss how the message queue paradigm could be used to implement a chat room application.

Group Communication

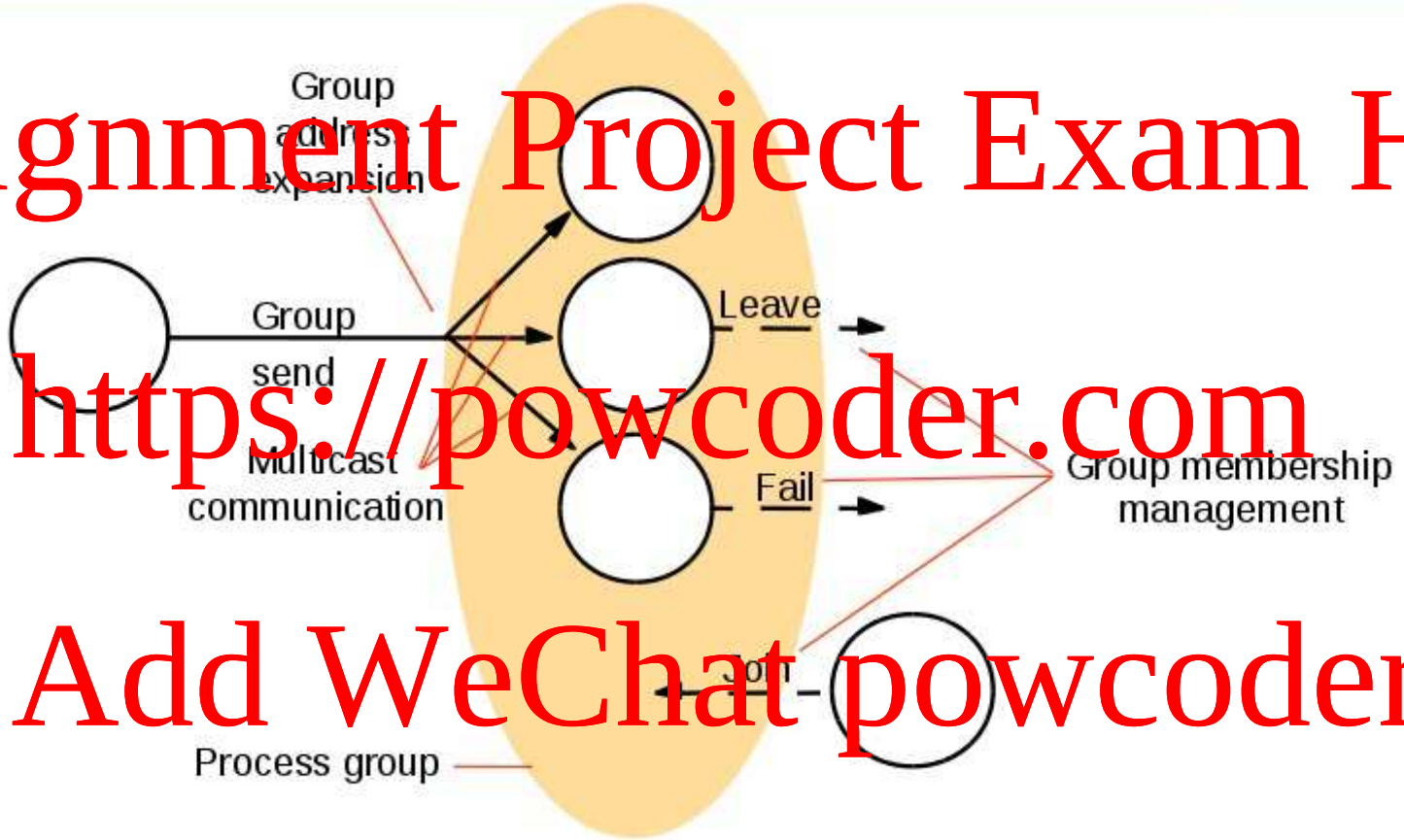
Group communication offers a space uncoupled service whereby a message is sent to a group and then this message is delivered to all members of the group. It provides more than a primitive IP multicast:

- manages group membership
- detects failures and provides reliability and ordering guarantees

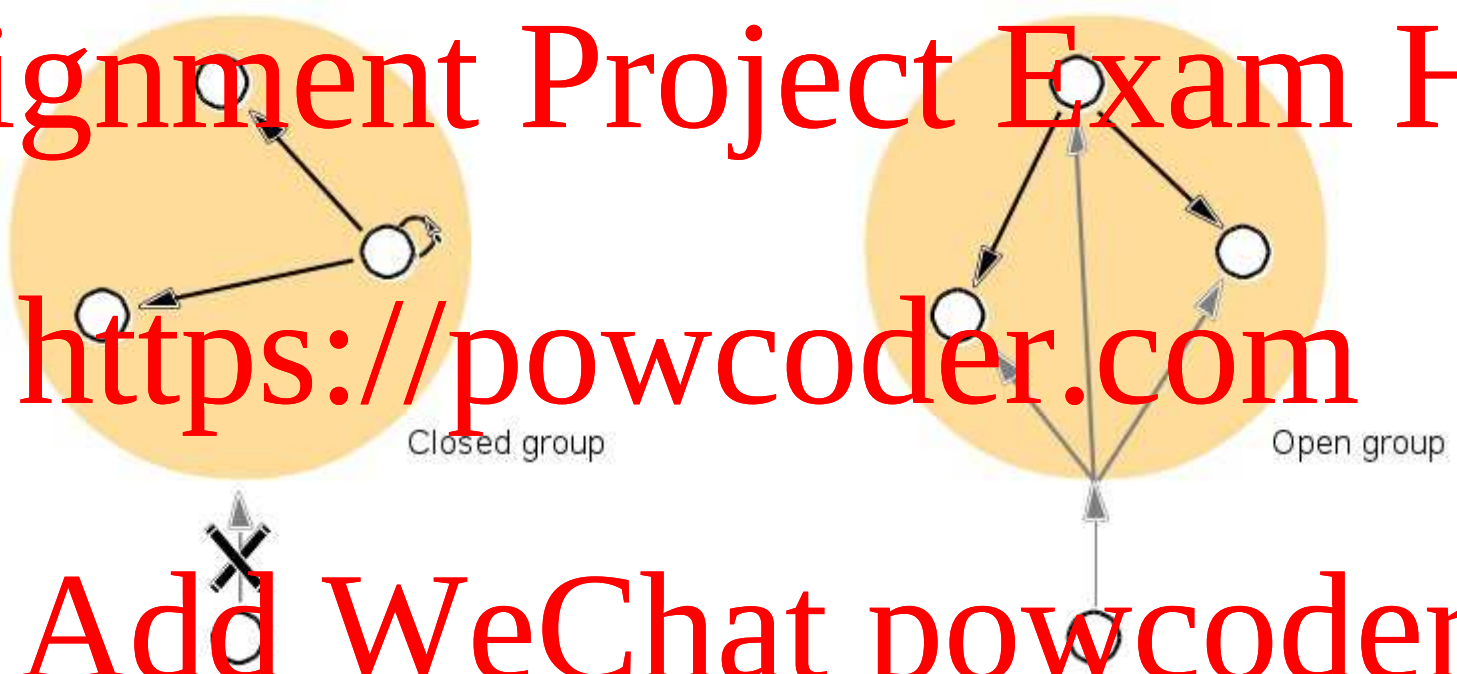
Typical aspects of a Group API:

- group creation
  - create/delete a group
  - list/search available groups
- group membership
  - join/leave a group
  - list members of a group
- multicast to selected members of a group, broadcast to all members

Efficient sending to multiple receivers, instead of multiple independent send operations, is an essential feature of group communication.



Group services



- closed groups only allow group members to multicast to it
- overlapping groups allows entities to be members of multiple groups
- synchronous and asynchronous variations can be considered

Implementation issues

- reliability and ordering in multicast
  - FIFO (first in first out) ordering is concerned with preserving the order from the perspective of a sender process
  - causal ordering, a message that *happens before* another message will be preserved in that order in the delivery at all processes
  - total ordering, if a message is delivered before another message at one process then this is preserved at all processes
- group membership management
  - group members leave and join
  - failed members
  - notifying members of group membership changes
  - changes to the group address

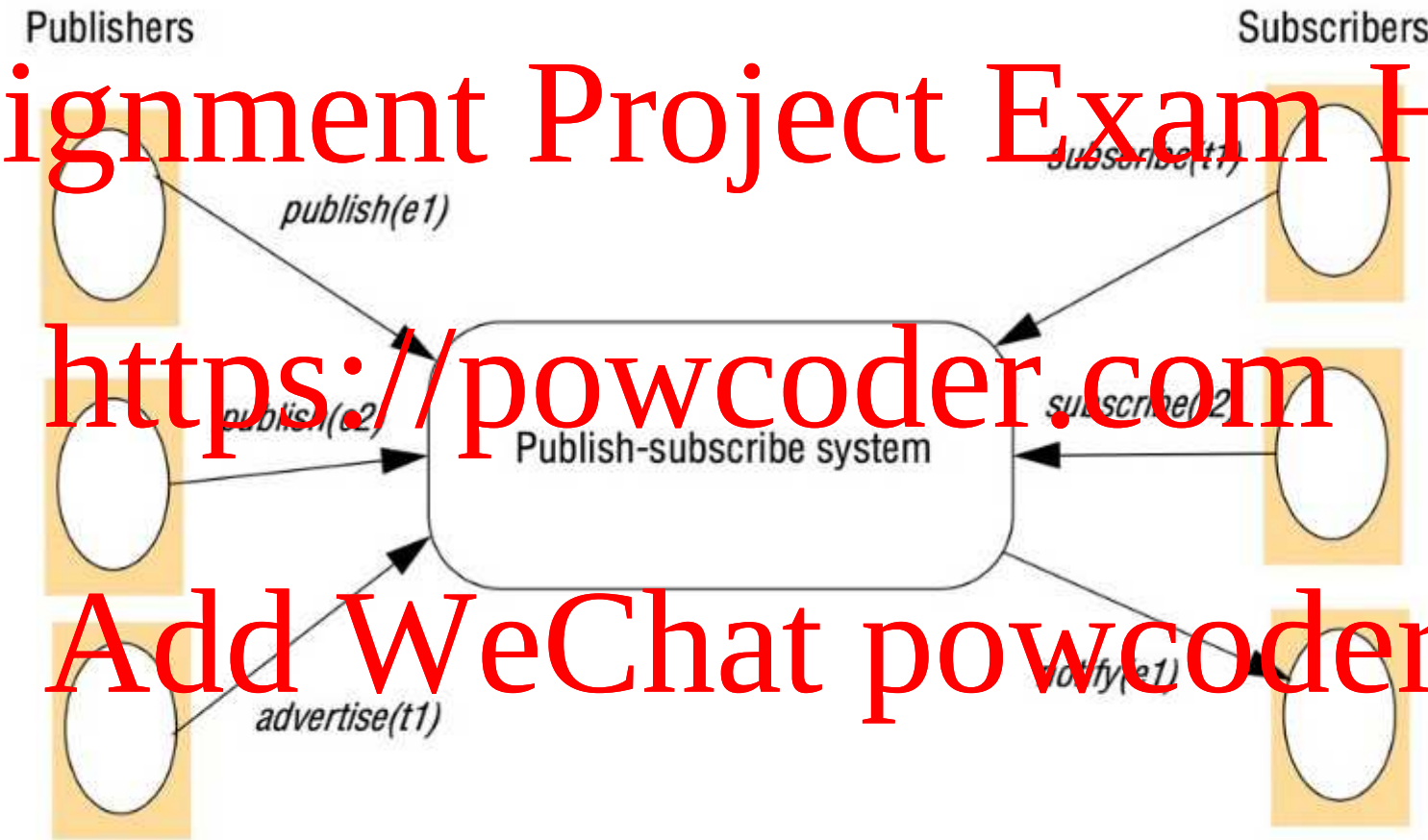
Discussion questions

Question (2): Discuss how the group communication paradigm could be used to implement a chat room application. Would this be better or worse than using the message queue paradigm?

Publish/Subscribe Systems

- *Publish/subscribe* systems are sometimes referred to as *distributed event-based systems*. A publish/subscribe system is a system where *publishers* (event sources) publish structured *events* to an *event service* and *subscribers* express interest in particular events through *subscriptions* which can be arbitrary patterns or query expressions over the structured events.
  - financial information systems
  - live feeds of real-time data, e.g. RSS feeds
  - support for cooperative working, where a number of participants need to be informed of events of shared interest
  - support for ubiquitous computing, including management of events emanating from the ubiquitous infrastructure, e.g. location events
  - a broad set of monitoring applications, including network monitoring in the Internet
- Types of pub-sub systems include:
  - *Channel Based* – Publishers publish to named channels and subscribers subscribe to all events on a named channel.
  - *Type Based* – Subscribers register interest in types of events and notifications occur when particular types of events occur.
  - *Topic Based* – Subscribers register interest in particular topics and notifications occur when any information related to the topic arrives.
  - *Content Based* – This is the most flexible of the schemes. Subscribers can specify interest is particular values or ranges of values for multiple attributes. Notifications are based on matching the attribute specification criteria.
- When and event matches a subscriber's subscription then the system sends a *notification* that contains the event to the subscriber.

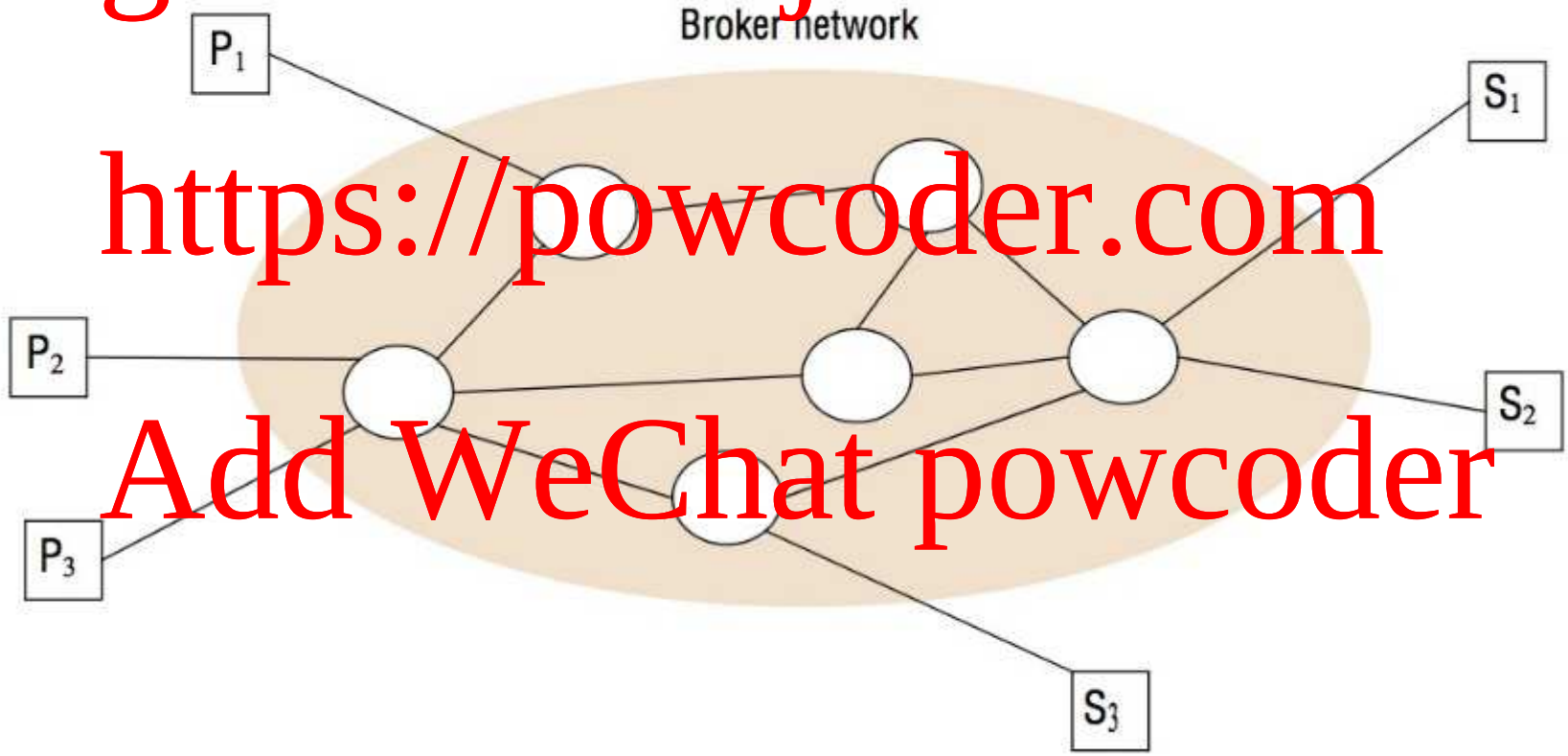
Programming model



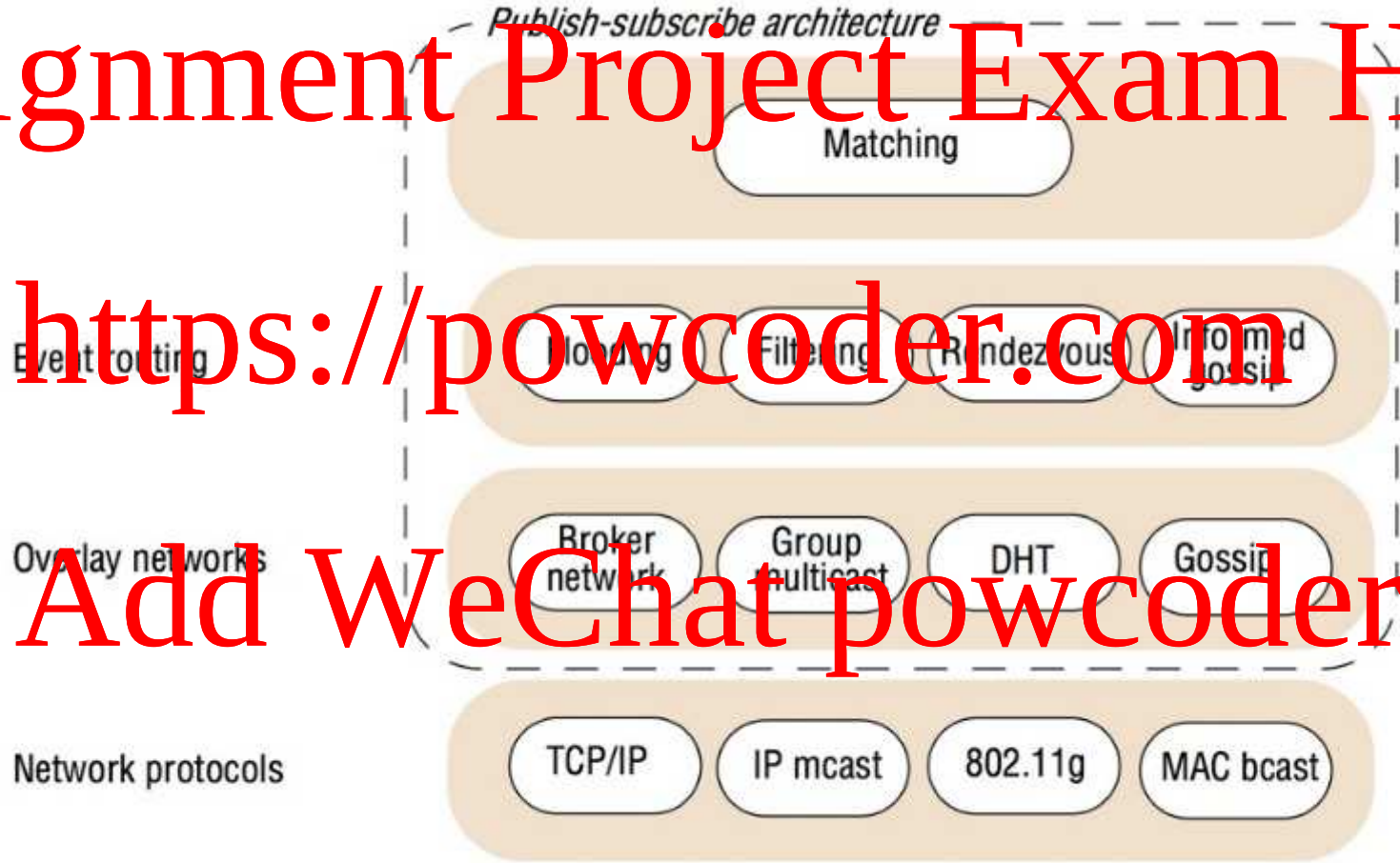
Advertise provides an additional mechanism for publishers to declare the nature of future events, i.e. the types of events of interest that may occur.

Multi-server architecture

The *Broker* exchanges or routes information from publishers to subscribers.



Overall System Architecture



Examples of pub/sub systems

A modern example of a pub/sub system is Apache Kafka. Others are shown below.

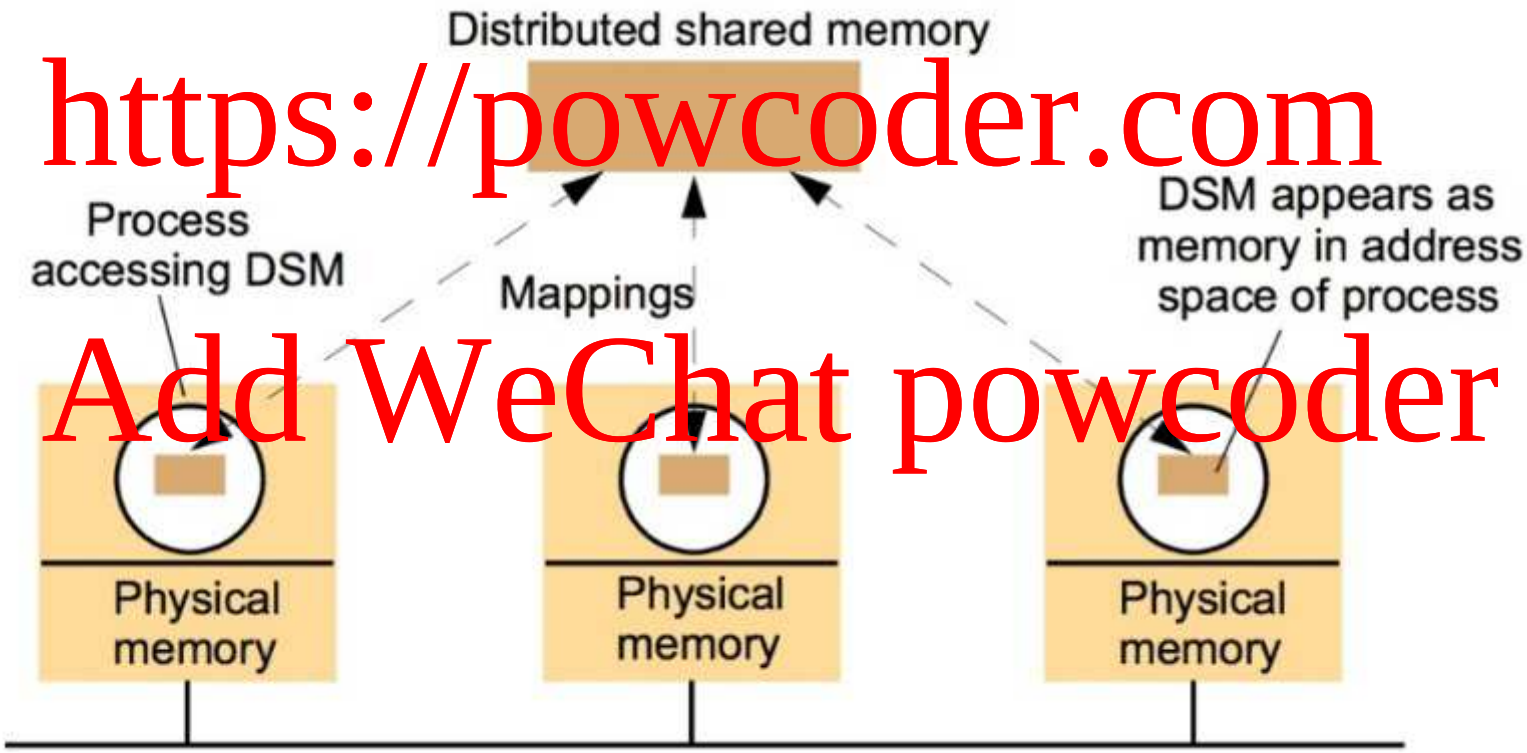
System (and further reading)	*Subscription model*	*Distribution model*	*Event routing*
CORBA Event Service (Chapter 8)	Channel-based	Centralized	-
TIB Rendezvous [Oki *et al.* 1993]	Topic-based	Distributed	Filtering
Scribe [Castro *et al.* 2002b]	Topic-based	Peer-to-peer (DHT)	Rendezvous
TERA [Baldoni *et al.* 2007]	Topic-based	Peer-to-peer	Informed gossip
Siena [Carzaniga *et al.* 2001]	Content-based	Distributed	Filtering
Gryphon [www.research.ibm.com]	Content-based	Distributed	Filtering
Hermes [Pietzuch and Bacon 2002]	Topic- and content-based	Distributed	Rendezvous and filtering
MEDYM [Cao and Singh 2005]	Content-based	Distributed	Flooding
Meghdoot [Gupta *et al.* 2004]	Content-based	Peer-to-peer	Rendezvous
Structure-less CBR [Baldoni *et al.* 2005]	Content-based	Peer-to-peer	Informed gossip

Discussion questions

Question (3): Discuss how the publish/subscribe paradigm could be used to implement a chat room application. Would this be better or worse than the message queue and or group communication paradigm?

Shared memory approaches

Distributed shared memory is an abstraction for sharing data between computers that do not share physical memory. Processes access DSM by reads and updates to what appears to be ordinary memory within their address space.



Tuple Spaces

The tuple space is a more abstract form of shared memory, compared to DSM.

Example: The LighTS interface

Picco, Balzarotti, et. al., "LighTS: A Lightweight, Customizable Tuple Space Supporting Context-Aware Applications"

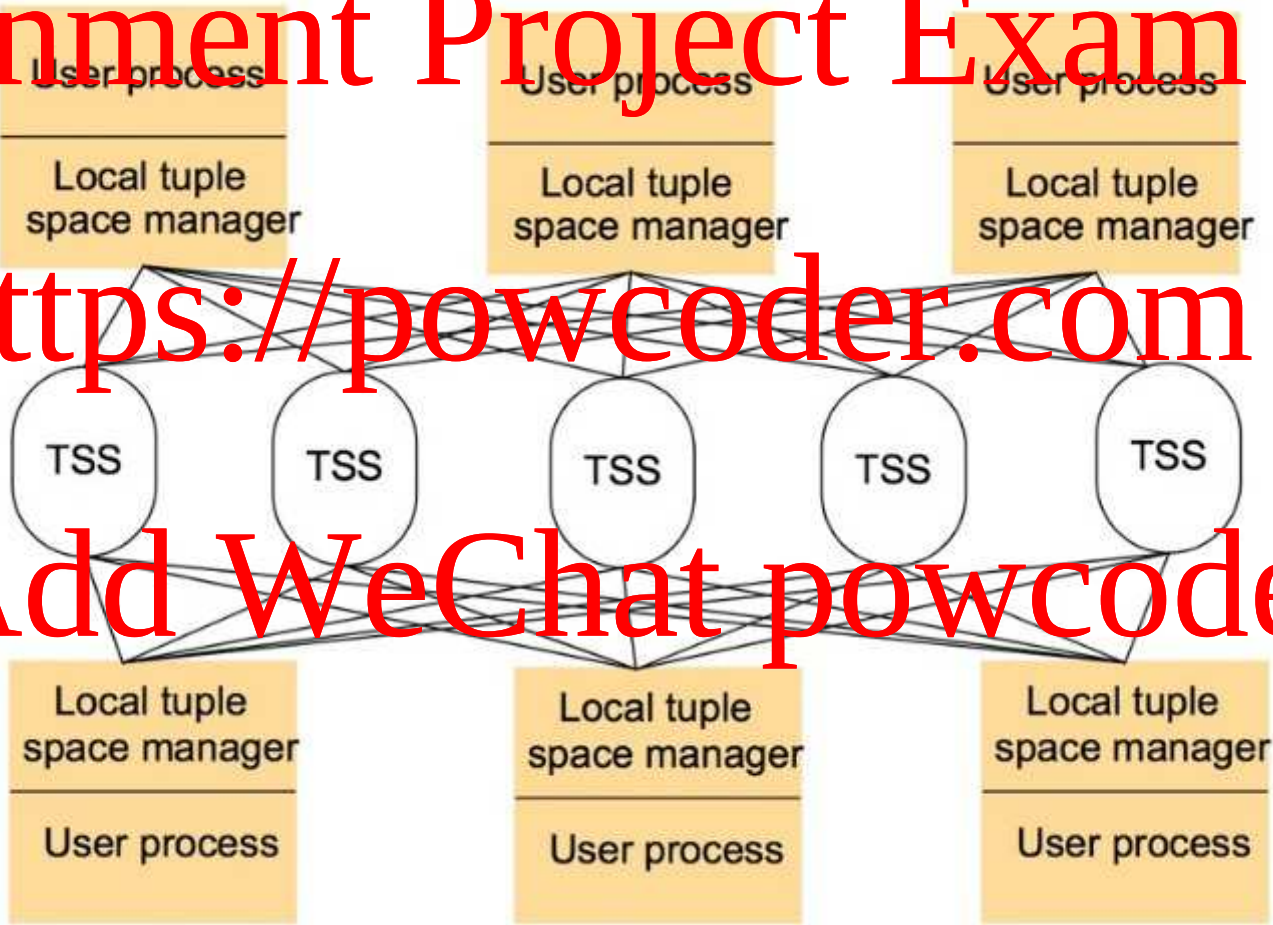

```
public interface ITupleSpace {
  String getName(); // name of tuple space
  void out(ITuple tuple); // put to tuple space
  void outg(ITuple[] tuples); // put to tuple space
  ITuple in(ITuple template); // blocking take
  ITuple inp(ITuple template); // non-blocking take
  ITuple[] ing(ITuple template); // blocking take
  ITuple rd(ITuple template); // blocking read
  ITuple rdp(ITuple template); // non-blocking read
  ITuple[] rdg(ITuple template); // blocking read
  int count(ITuple template); // count tuples
}
public interface ITuple {
  ITuple add(IField field);
  ITuple set(IField field, int index);
  IField get(int index);
  ITuple insertAt(IField field, int index);
  ITuple removeAt(int index);
  IField[] getFields();
  int length();
  boolean matches(ITuple tuple);
}
public interface IField {
  Class getType();
  IField setType(Class classObj);
  boolean matches(IField field);
}
public interface IValuedField extends IField {
  boolean isFormal(); // formal is a wildcard
  java.io.Serializable getValue();
  IValuedField setValue(java.io.Serializable obj);
}
```

Example

```
ITupleSpace ts = new TupleSpace("SAC05");
IField f1 = new Field().setValue("Paolo");
IField f2 = new Field().setValue( new Integer (10));
ITuple t1 = new Tuple().add(f1).add(f2);
ts.out(t1);
```

Example York Linda Kernel

The implementation uses multiple Tuple Space Servers.



Discussion questions

Question (4): Discuss how the tuple space paradigm could be used to implement a chat room application. Compare this to message queue, group communication, and publish/subscribe.

Summary

	Groups	*Publish-subscribe systems*	*Message queues*	*DSM*	*Tuple spaces*
Space-uncoupled	Yes	Yes	Yes	Yes	Yes
Time-uncoupled	Possible	Possible	Yes	Yes	Yes
Style of service	Communication-based	Communication-based	Communication-based	State-based	State-based
Communication pattern	1-to-many	1-to-many	1-to-1	1-to-many	1-1 or 1-to-many
Main intent	Reliable distributed computing	Information dissemination or EAI; mobile and ubiquitous systems	Information dissemination or EAI; commercial transaction processing	Parallel and distributed computation	Parallel and distributed computation; mobile and ubiquitous systems
Scalability	Limited	Possible	Possible	Limited	Limited
Associative	No	Content-based publish-subscribe only	No	No	Yes

Discussion questions

Question (5): Considering all of these indirect communication paradigms, which one, if any, would be more suitable for implementing a video conferencing system like Zoom? If none of them are suitable then what kind of paradigm/metaphor would be more suitable?