Attention Please

- Start your research project ASAP!
- For each homework assignment, you have to print your submission.
- Midterm-exam will be scheduled on April 4 in class.
- There're about 10 research areas in project.
- Sample papers have been given.
- Each group has to select one area and each area will be assigned to no more than 2 groups.
- So, please build your team and select your topic ASAP!
- Each group has no more than 2 students.

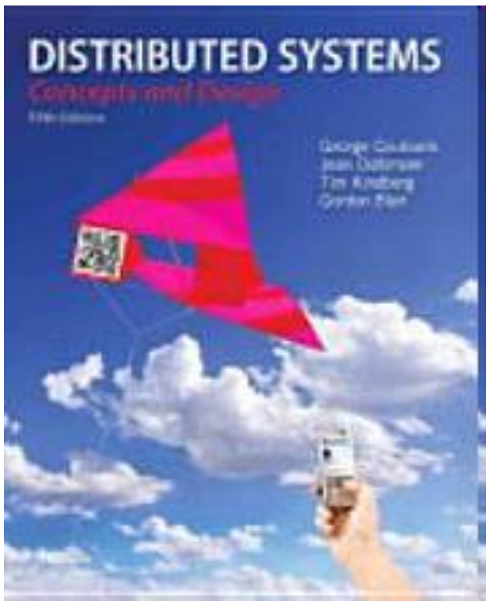
Paper Reading Assignment Is Available Online!

- All papers in blue color are important ones and must be read if you choose that area.

- Every student needs to read this paper before Jan. 29 (only the first 4 pages and Section 4 and Section 5)

L.W. Lee, P. Scheuermann and R. Vingralek, “File assignment in parallel I/O systems with Minimal Variance of Service Time,” *IEEE Transactions on Computers*, 2000.

Slides for Chapter 4: Interprocess Communication



From **Coulouris, Dollimore and Kindberg**

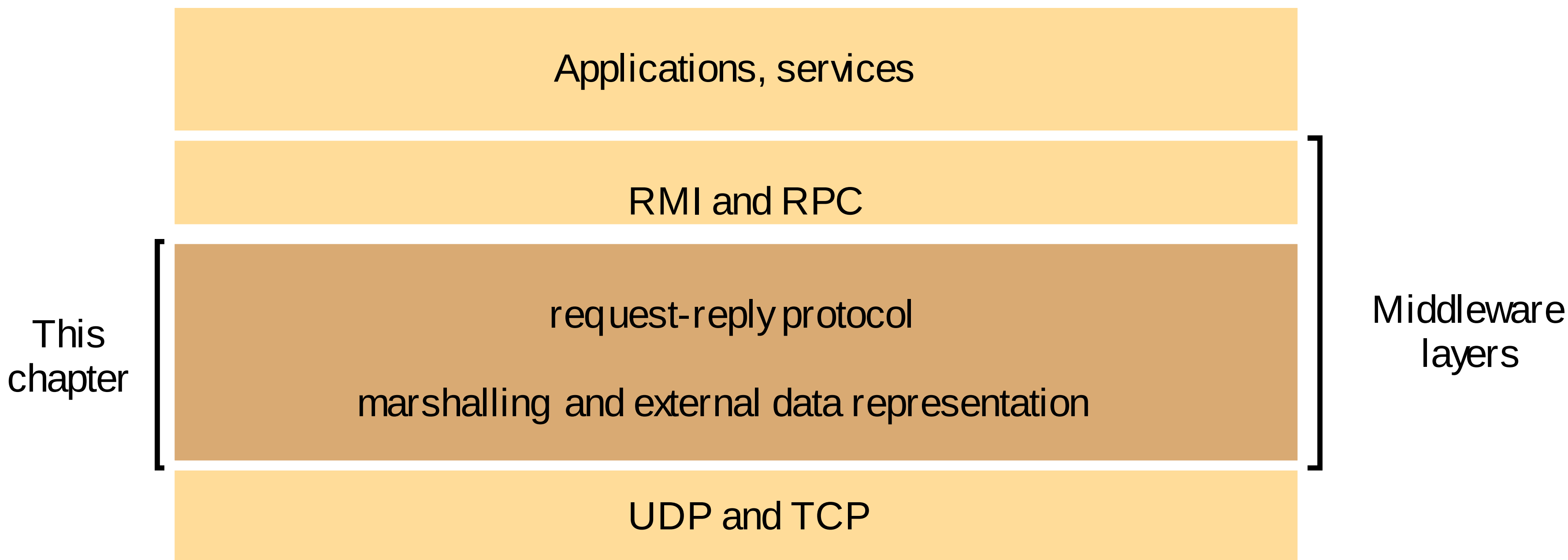
Distributed Systems: Concepts and Design

Edition 5, © Addison-Wesley 2011

Objectives of This Chapter:

- To study the general characteristics of interprocess communication and the particular characteristics of both datagram and stream communication in the Internet.
- To be able to write Java applications that use the Internet protocols and Java serialization.
- To be aware of the design issues for Request-Reply protocols and how collections of data objects may be represented in messages (RMI and language integration are left until Chapter 5).
- To be able to use the Java API to IP multicast and to consider the main options for reliability and ordering in group communication.

Middleware layers



Important Concepts

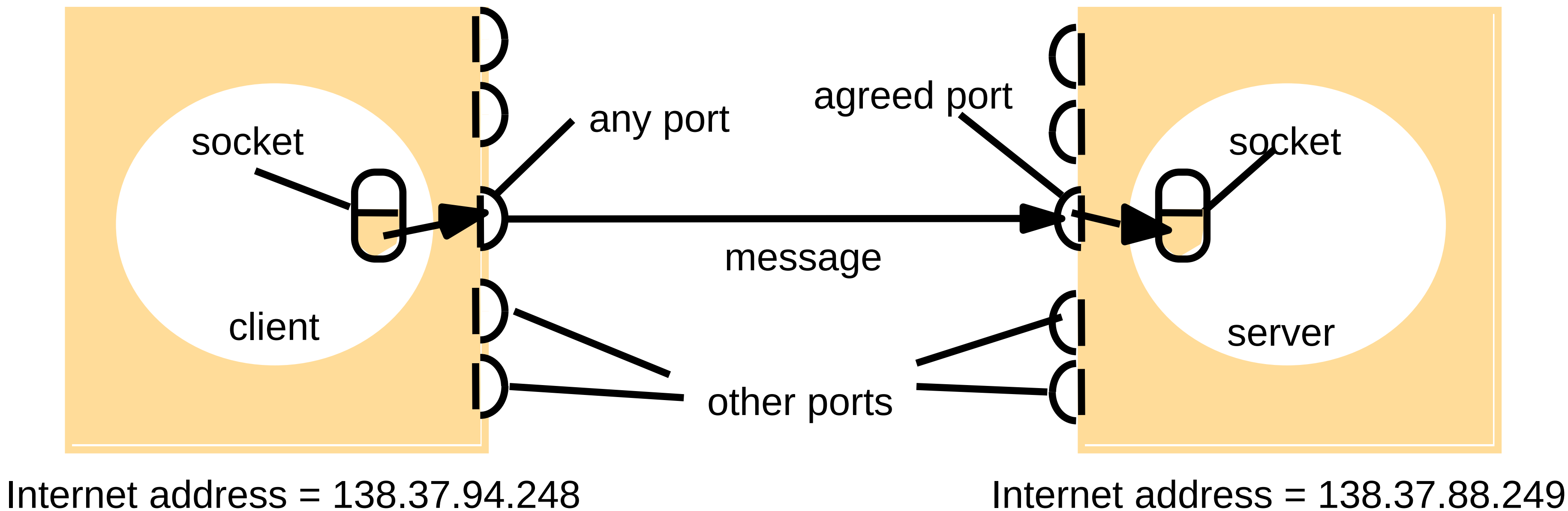
- Port – A message destination within a computer, specified as an integer.
- UDP (User Datagram Protocol)– Using UDP, programs on networked computers can send short messages sometimes known as datagrams (using Datagram Sockets) to one another. No guarantee reliability or ordering.
- TCP (Transmission Control Protocol)– Provides reliable, in-order delivery of a stream of bytes, making it suitable for applications like file transfer and e-mail.
- Multicast – The delivery of information to a group of destinations simultaneously using the most efficient strategy to deliver the messages over each link of the network only once.
- Broadcast – Transmitting a packet that will be received (conceptually) by every device on the network. In practice, the scope of the broadcast is limited to a broadcast domain.

API for Internet Protocols (1): IPC characteristics

- synchronous and asynchronous communication
  - blocking send: waits until the corresponding receive is issued
  - non-blocking send: sends and moves on
  - blocking receive: waits until the msg is received
  - non-blocking receive: if the msg is not here, moves on
  - synchronous: blocking send and blocking receive
  - asynchronous: non-blocking send and blocking or non-blocking receive
- Message Destination
  - IP address + port: one receiver, many senders
  - Location transparency
    - name server or binder: translate service to location
    - OS (e.g. Mach): provides location-independent identifier mapping to lower-level addresses
  - send directly to processes (e.g. V System)
  - multicast to a group of processes (e.g. Chorous)
- Reliability: in terms of validity and integrity
- Ordering: messages are delivered in sender order

API for the Internet Protocols (2): Sockets and ports

- programming abstraction for UDP/TCP
- originated from BSD UNIX



Socket Properties

- For a process to receive messages, its socket must be bound to a local port on one of the Internet addresses of the computer on which it runs.
- Messages sent to a particular port of an Internet address can be only received by a process that has a socket associated with the particular port number on that Internet address.
- Same socket can be used both for sending and receiving messages.
- Processes can use multiple ports to receive messages.
- Ports cannot be shared between processes for receiving messages.
- Any number of processes can send messages to the same port.
- Each socket is associated with a single protocol (UDP or TCP).

API for Internet Protocols (3): UDP Datagram

- message size: up to 2^{16} bytes, usually restrict to 8Kbytes
- blocking: non-blocking send, blocking receive
- timeouts: timeout on blocking receive
- receive from any: doesn't specify sender origin (possible to specify a particular host for send and receive)
- failure model:
  - Data Corruption: checksum can be used to detect data corruption
  - Omission failures: buffers full, corruption, dropping
  - Order: messages might be delivered out of order
- use of UDP
  - DNS
  - less overhead: no state information, extra messages, latency due to start up

API for Internet Protocols (6): TCP stream

- **Message sizes:** There is no limit on data size applications can use.
- **Lost messages:** TCP uses an acknowledgment scheme unlike UDP. If acknowledgments are not received the messages are retransmitted.
- **Flow control:** TCP protocol attempts to match the speed of the process that reads the message and writes to the stream.
- **Message duplication or ordering:** Message identifiers are associated with IP packets to enable recipient to detect and reject duplicates and reorder messages in case messages arrive out of order.
- **Message destinations:** The communicating processes establish a connection before communicating. The connection involves a connect request from the client to the server followed by an accept request from the server to the client.

TCP Stream

Steps involved in establishing a TCP stream socket:

- **Client:**
  - Create a socket specifying the server address and port
  - Read and write data using the stream associated with the socket
- **Server:**
  - Create a listening socket bound to a server port
  - Wait for clients to request a connection (Listening socket maintains a queue of incoming connection requests)
  - Server accepts a connection and creates a new stream socket for the server to communication with the client retaining the original listening socket at the server port for listening to incoming connections. A pair of sockets in client and server are connected by a pair of streams, one in each direction. A socket has an input stream and an output stream.

TCP Communication Issues

When an application closes a socket, the data in the output buffer is sent to the other end with an indication that the stream is broken. No further communication is possible.

TCP communication issues:

- There should a pre-agreed format for the data sent over the socket
- Blocking is possible at both ends
- If the process supports threads, it is recommended that a thread is assigned to each connection so that other clients will not be blocked.

Failure Model:

- TCP streams use checksum to detect and reject corrupt packets and sequence numbers to detect and reject duplicates
- Timeouts and retransmission is used to deal with lost packets
- Under severe congestion TCP streams declare the connections to be broken hence does not provide reliable communication
- When communication is broken the processes cannot distinguish between network failure and process crash
- Communicating process cannot definitely say whether the messages sent recently were received

Use of TCP: HTTP, FTP, Telnet, SMTP

Important concepts

- *External data representation:* Agreed standard for representing data structures and primitive data

- *Marshalling:* Process of converting the data to the form suitable for transmission

- *Unmarshalling:* Process of disassembling the data at the receiver

External Data Representation (1):

- different ways to represent int, float, char... (internally)
- byte ordering for integers
  - big-endian: most significant byte first
  - small-endian: least significant byte first
- standard external data representation
  - marshal before sending, unmarshal before receiving
- send in sender's format and indicates what format, receivers translate if necessary
- External data representation
  - SUN's External data representation (XDR)
  - CORBA's Common Data Representation (CDR)
  - Java's object serialization
  - ASCII (XML, HTTP)

External Data Representation (2): CDR

- Primitive types (15): short, long ...
  - support both big-endian and little-endian
  - transmitted in sender's ordering and the ordering is specified
  - receiver translates if needed
- Constructed types

Type	*Representation*
sequence	length (unsigned long) fol owed by elements in order
string	length (unsigned long) followed by characters in order (can also can have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

External Data Representation (3):

- CORBA IDL (interface definition language) compiler generates marshalling and unmarshalling routines
- Struct with string, string, unsigned long

```
Struct Person {
string name;
string place;
Unsigned long year;
};
```

index in sequence of bytes	← 4 bytes →	*notes on representation*
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h___"	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20-23	"on__"	
24–27	1934	*unsigned long*

The flattened form represents a *Person* struct with value: {'Smith', 'London', 1934}

External Data Representation (4): Java serialization

- serialization and deserialization are automatic in arguments and return values of Remote Method Interface (RMI)
- flattened to be transmitted or stored on the disk
  - write class information, types and names of instance variables
  - new classes, recursively write class information, types, names...
  - each class has a handle, for subsequent references
  - values are in Universal Transfer Format (UTF)

External Data Representation (5): Java serialization

```
public class Person implements Serializable {
        private String name;
         private String place;
         private int year;

         public Person(String aName, String aPlace, int aYear){
            name = aName;
            place = aPlace;
            year = aYear;
         }
}
```

Serialized values				*Explanation*
Person	8-byte version number		h0	*class name, version number*
3	int year	java.lang.String name:	java.lang.String place:	*number, type and name of instance variables*
1934	5 Smith	6 London	h1	*values of instance variables*

The true serialized form contains additional type markers; h0 and h1 are handles to other objects

External Data Representation (6)

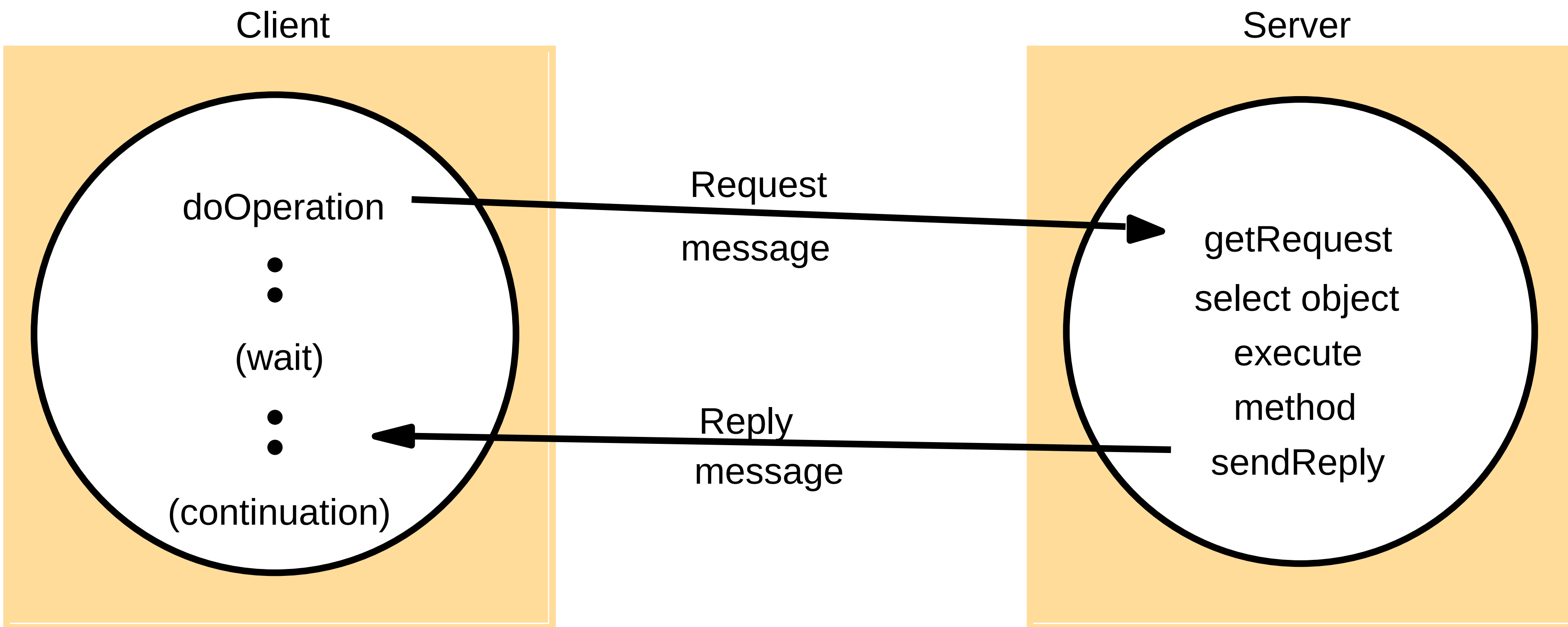
- references to other objects
  - other objects are serialized
  - references are serialized as handles
  - each object is written only once
  - second or subsequent occurrence of the object is written as a handle
- reflection
  - ask the properties (name, types, methods) of a class
  - help serialization and deserialization

External Data Representation (7): Remote object reference

- call methods on a remote object
  - unique reference in the distributed system
  - Reference = IP address + port + process creation time + local object # in a process + interface
  - Port + process creation time -> unique process
  - Address can be derived from the reference
  - Objects usually don't move; is there a problem if the remote object moves?

32 bits	*32 bits*	*32 bits*	*32 bits*	
Internet address	port number	time	object number	interface of remote object

Client-server communication (1)



- Synchronous: client waits for a reply
- Asynchronous: client doesn’t wait for a reply

Client-server communication (2)

public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments)

sends a request message to the remote object and returns the reply.
The arguments specify the remote object, the method to be invoked and the arguments of that method.

public byte[] getRequest ();

acquires a client request via the server port.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

sends the reply message reply to the client at its Internet address and port.

Client-server communication (3)

- Client-server communication normally uses the synchronous request-reply communication paradigm
- Involves *send* and *receive* operations
- TCP or UPD can be used - TCP involves additional overheads
  - redundant acknowledgements
  - needs two additional messages for establishing connection
  - flow control is not needed since the number of arguments and results are limited

Client-server communication (4): Request-reply message structure

messageType	*int (0=Request, 1= Reply)*
requestId	*int*
objectReference	*RemoteObjectRef*
methodId	*int or Method*
arguments	*array of bytes*

Why requestID?

Client-server communication (5)

- Failure model
  - UDP: could be out of order, lost...
  - process can fail...
- not getting a reply
  - timeout and retry
- duplicate request messages on the server
  - How does the server find out?
- *idempotent* operation: can be performed repeatedly with the same effect as performing once.
  - idempotent examples?
  - non-idempotent examples?
- history of replies (for servers)
  - retransmission without re-execution
  - how far back if we assume the client only makes one request at a time?

Drawbacks of UDP-based request-reply protocol

- It’s difficult to decide on an appropriate size for buffer.
- Limited length of datagrams.
- It needs to implement multi-packet protocols

Client-server communication (6)

- using TCP increase reliability and also cost
- HTTP uses TCP
  - one connection per request-reply
  - HTTP 1.1 uses "persistent connection"
    - multiple request-reply
    - closed by the server or client at any time
    - closed by the server after timeout on idle time
  - Marshal messages into ASCII text strings
  - resources are tagged with MIME (Multipurpose Internet Mail Extensions) types: test/plain, image/gif...
  - content-encoding specifies compression algorithm

Client-server communication (7): HTTP methods

- GET: return the file, results of a cgi program, …
- HEAD: same as GET, but no data returned
- POST: transmit data from client to the program at url
- PUT: store data at url
- DELETE: delete resource at url
- OPTIONS: server provides a list of valid methods
- TRACE: server sends back the request

Client-server communication (8): HTTP request/reply format

method	*URL or pathname*	*HTTP version*	*headers*	*message body*
GET	//www.dcs.qmw.ac.uk/index.html	HTTP/ 1.1		

- Headers: latest modification time, acceptable content type, authorization credentials

HTTP version	*status code*	*reason*	*headers*	*message body*
HTTP/1.1	200	OK		resource data

- Headers: authentication challenge for the client

Group communication (1)

- multicast
- useful for:
  - fault tolerance based on replicated services
    - requests multicast to servers, some may fail, the client will be served
  - discovering services
    - multicast to find out who has the services
  - better performance through replicated data
    - multicast updates
  - event notification
    - new items arrived, advertising services

Group communication (2): IP multicast

- class D addresses, first four bits are 1110 in IPv4
- UDP
- Join a group via socket binding to the multicast address
- messages arriving on a host deliver them to all local sockets in the group
- multicast routers: route messages to out-going links that have members
- multicast address allocation
  - permanent
  - temporary:
    - no central registry, use (time to live) TTL to limit the # of hops, hence distance
    - tools like sd (session directory) can help manage multicast addresses and find new ones

Group communication (3): Reliability and ordering

- UDP-level reliability: missing, out-of-order...
- Effects on
  - fault tolerance based on replicated services
    - ordering of the requests might be important, servers can be inconsistent with one another
  - discovering services
    - not too problematic
  - better performance through replicated data
    - loss and out-of-order updates could yield inconsistent data, sometimes this may be tolerable
  - event notification
    - not too problematic

Assignment1 (Chapter 4)

- 4.5