SOFTENG 325

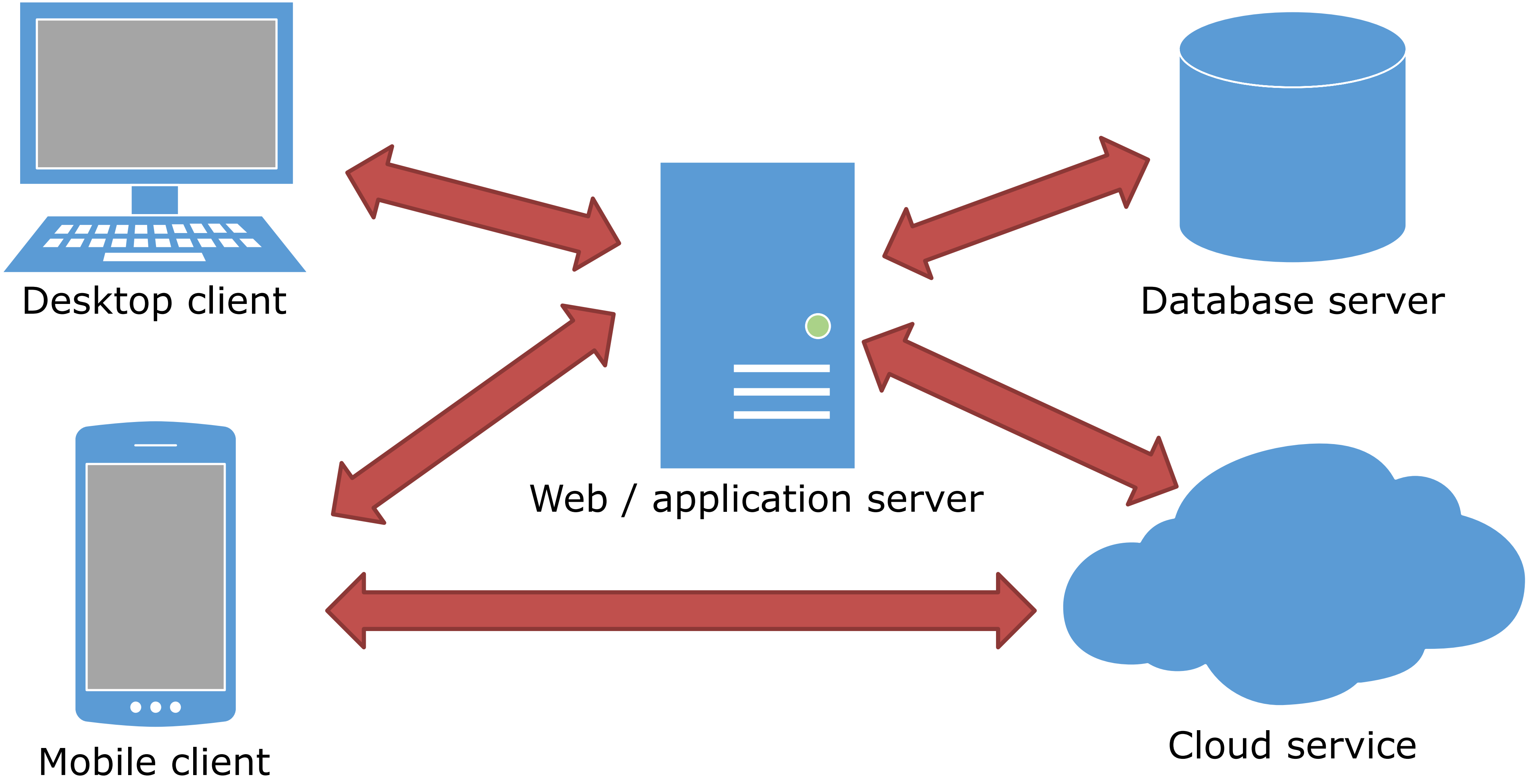
Software Architecture

Andrew Meads

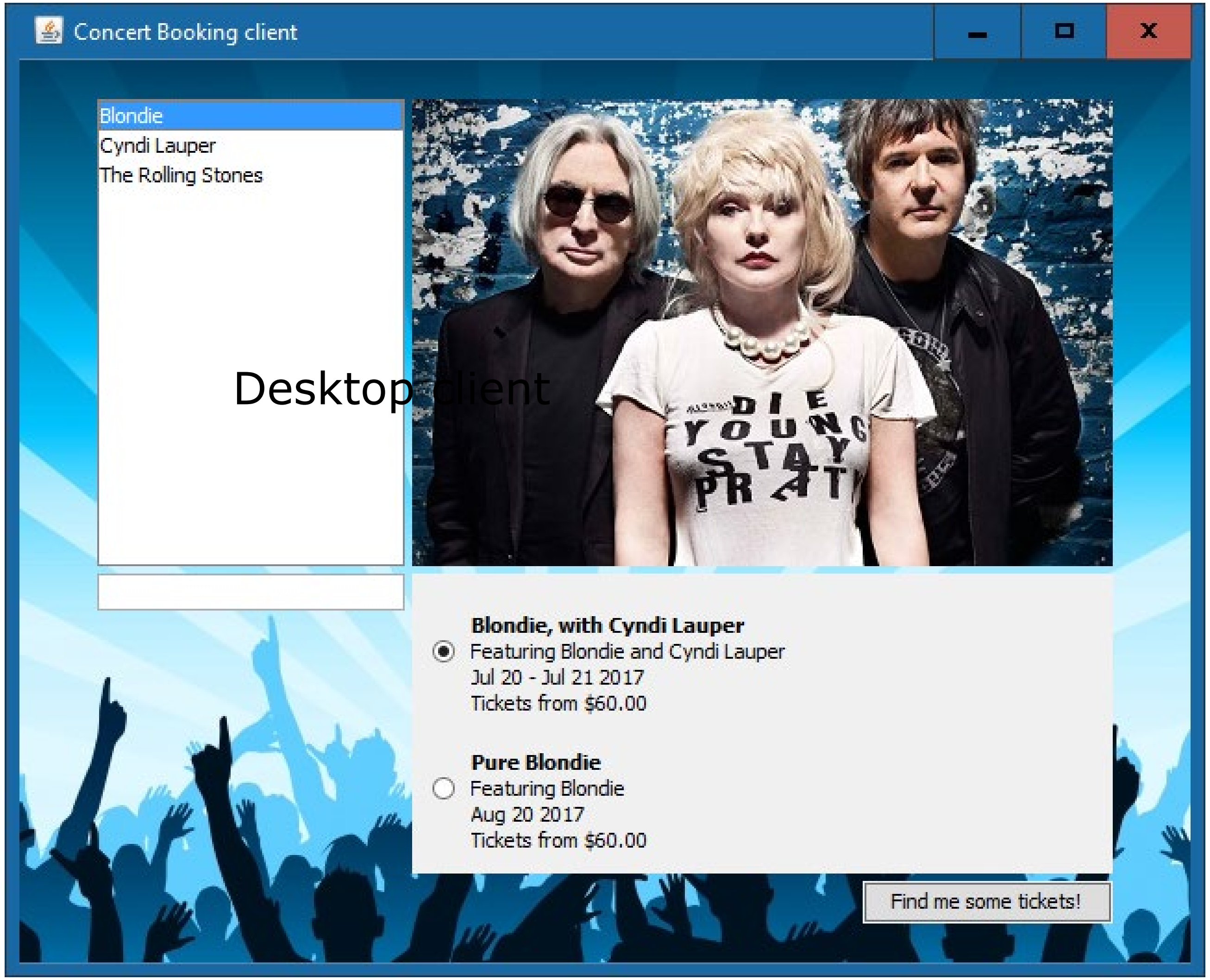
SOFTENG 325 – Software Architecture

Distributed Systems

Distributed systems



Assignment one



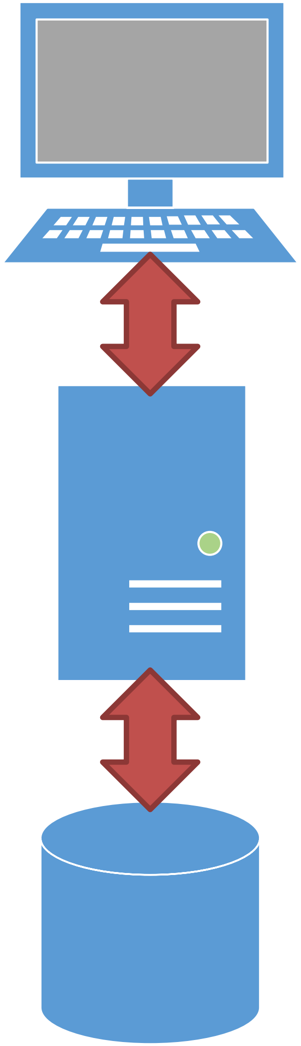
Concert Booking client

Blondie
Cyndi Lauper
The Rolling Stones

● **Blondie, with Cyndi Lauper**
Featuring Blondie and Cyndi Lauper
Jul 20 - Jul 21 2017
Tickets from $60.00

○ **Pure Blondie**
Featuring Blondie
Aug 20 2017
Tickets from $60.00

Desktop client



JavaFX / Swing / Web application

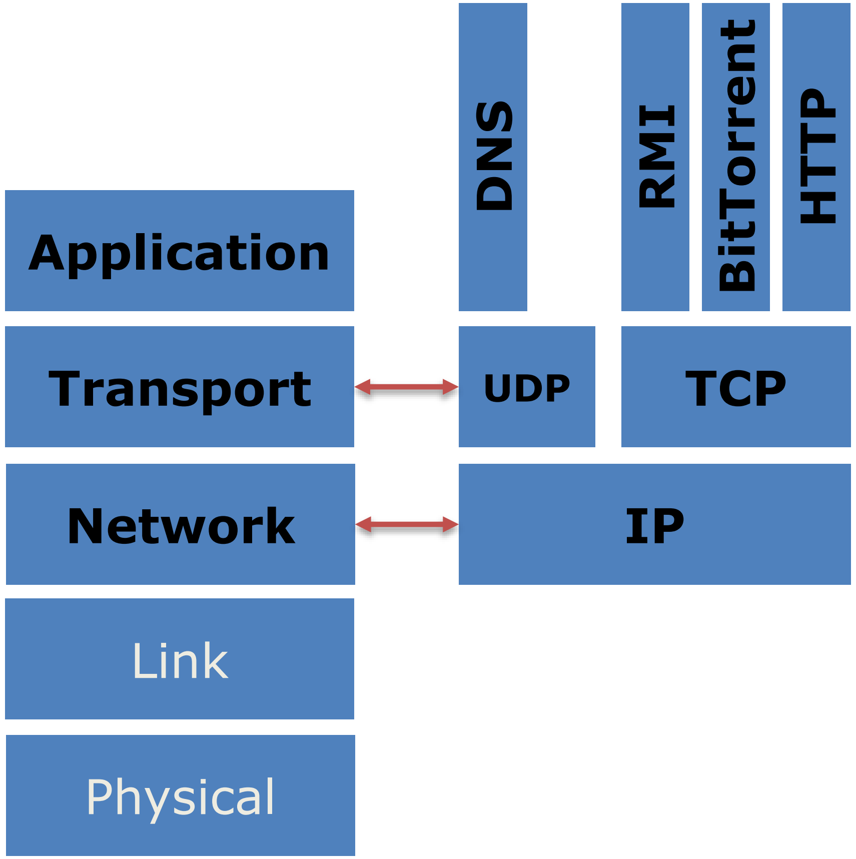
Servlet container, hosting a REST service with ORM

Relational database server

Networking infrastructure

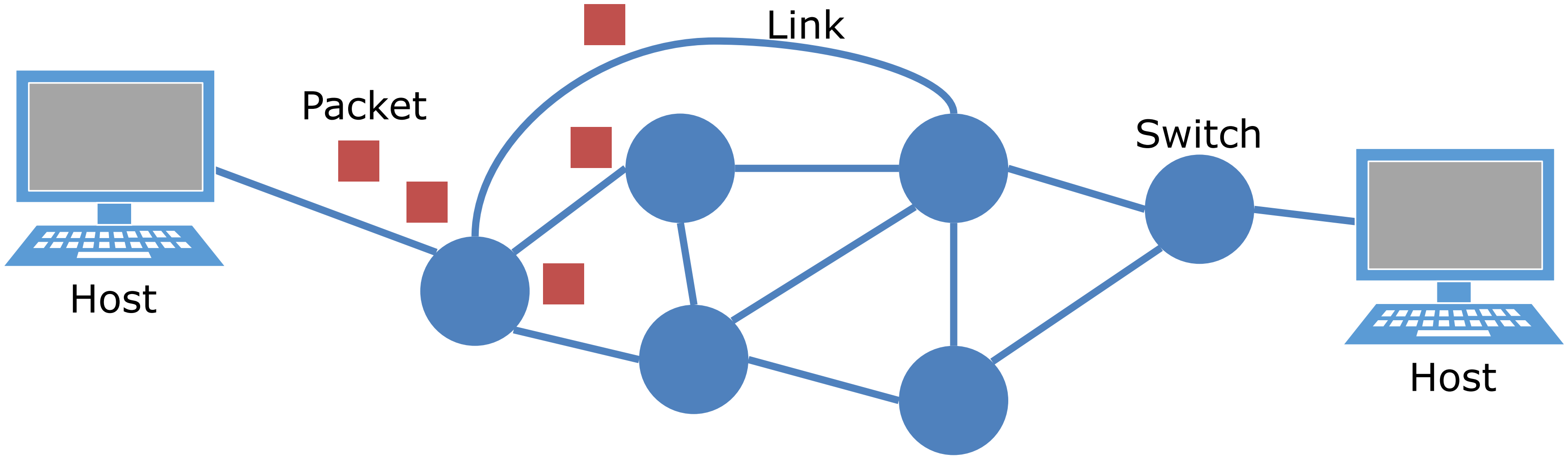
- Networking infrastructure exhibits the following characteristics:
  - Computers and links can fail independently
  - Switches have finite space for storing packets
  - Individual links vary in terms of bandwidth capacity
  - Data can be corrupted during transmission
  - Switches store routing tables that they dynamically update based on knowledge of congested links and failed switches

Network protocols



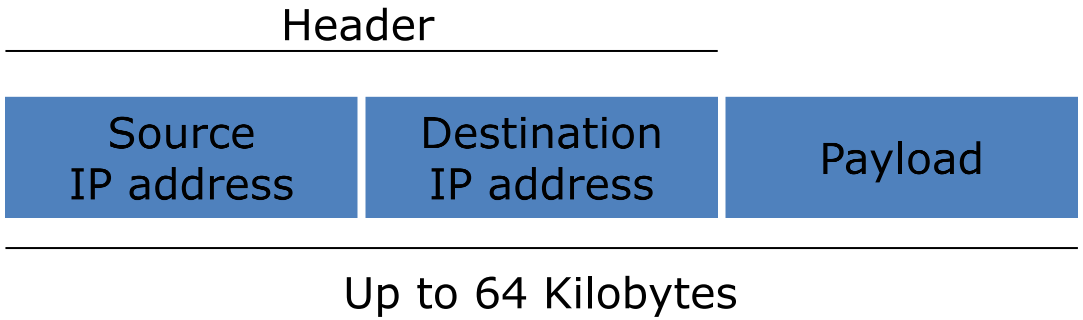
- Network protocols are organised into layers
  - Application layer protocols address the needs of particular applications
  - Transport protocols provide for process-to-process communication
  - The network layer provides a packet delivery service between host machines
- Higher-level protocols use the services of the layer directly beneath them
  - A layer depends on the interface of its underlying layer and not its implementation

Internet Protocol (IP)



The Internet protocol moves packets through the network to their destination – identified by the packet's destination address.

Packet structure



Header

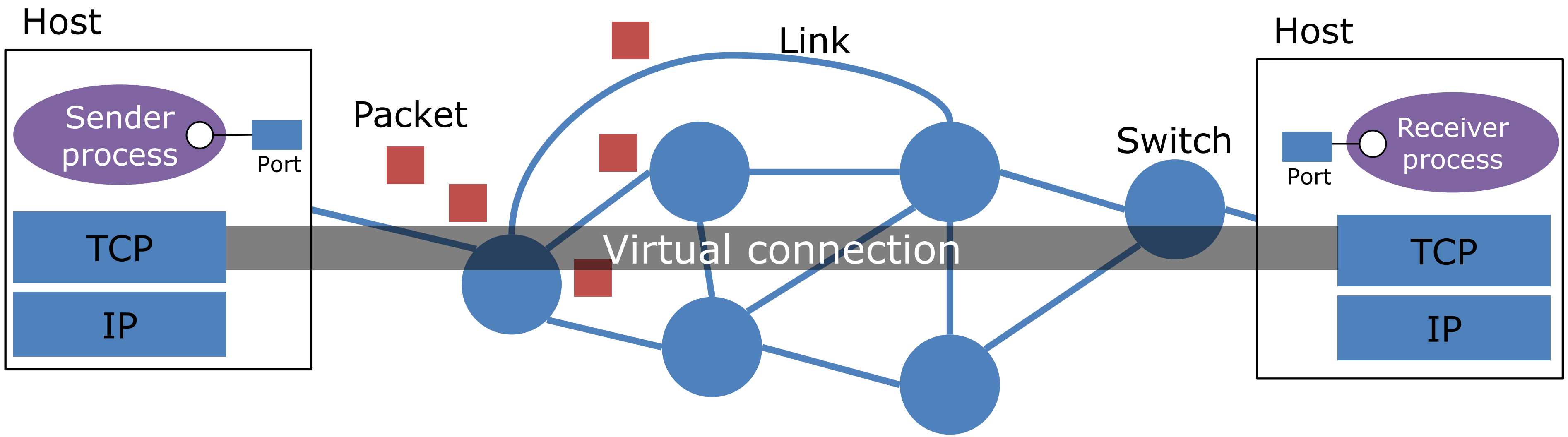
Source IP address

Destination IP address

Payload

Up to 64 Kilobytes

Transmission Control Protocol (TCP)



TCP (Transmission Control Protocol) is a transport protocol that establishes a **virtual connection** between a pair of processes – a bi-directional **stream** abstraction that hides several network characteristics:

- **Message sizes / boundaries:** The application simply reads and writes data from / to the stream – the TCP layer decides how much data to accumulate in the sender before it creates packet(s) and passes them down to the IP layer
- **Message destinations:** Once a stream has been established, the “connected” processes can use the stream without knowledge of ports and IP addresses
- **Lost messages**
- **Message duplication and ordering**
- **Flow control**

Class exercise – TCP

- How might you develop a TCP-like protocol that provides **reliable** and **ordered** communication over a virtual connection?

- Consider using the following …

ACK #1

#1

111111

Log

Timer

Acknowledgment packet

Sequence number

Checksum

Buffer

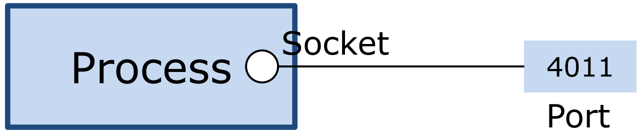
TCP

Issue	TCP behaviour
Validity	Lost packets are detected and resent
Integrity	A mandatory checksum is used to transform a corrupt packet into a lost packet
Ordering	Transmitted data is processed so that once received, it is delivered in the order in which it was sent; each packet has a sequence number
Blocking	The Sender can be blocked inserting data into an output stream; the receiver blocks if the input stream has insufficient data

TCP with Java

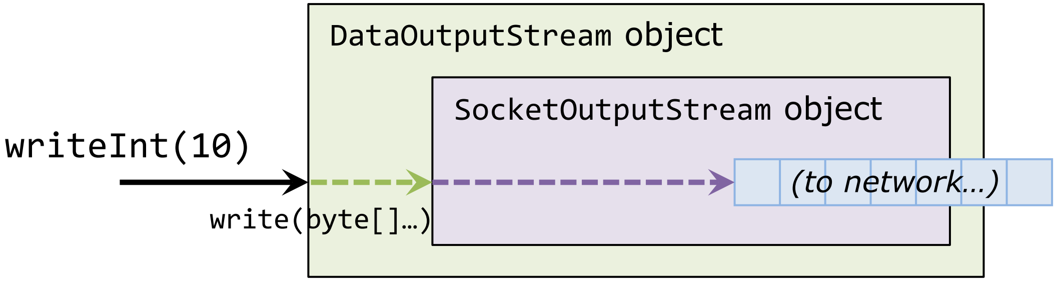
Using sockets

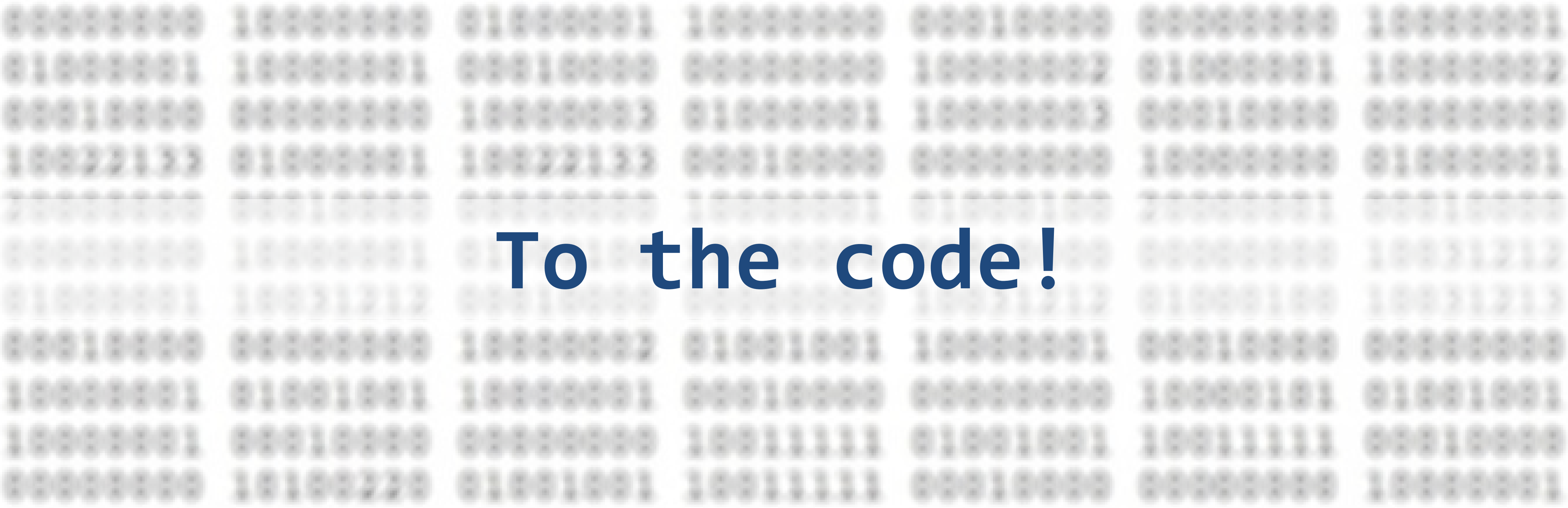
- To bind a process to a port number, a Socket is used
- For TCP, Java provides classes `Socket` and `ServerSocket`
  - These provide methods for establishing connections and acquiring I/O streams



Preparing data

- Data is ultimately sent in byte form, but it is convenient to work with meaningful data types
- Classes `DataInputStream` and `DataOutputStream` are useful for working with primitive data types



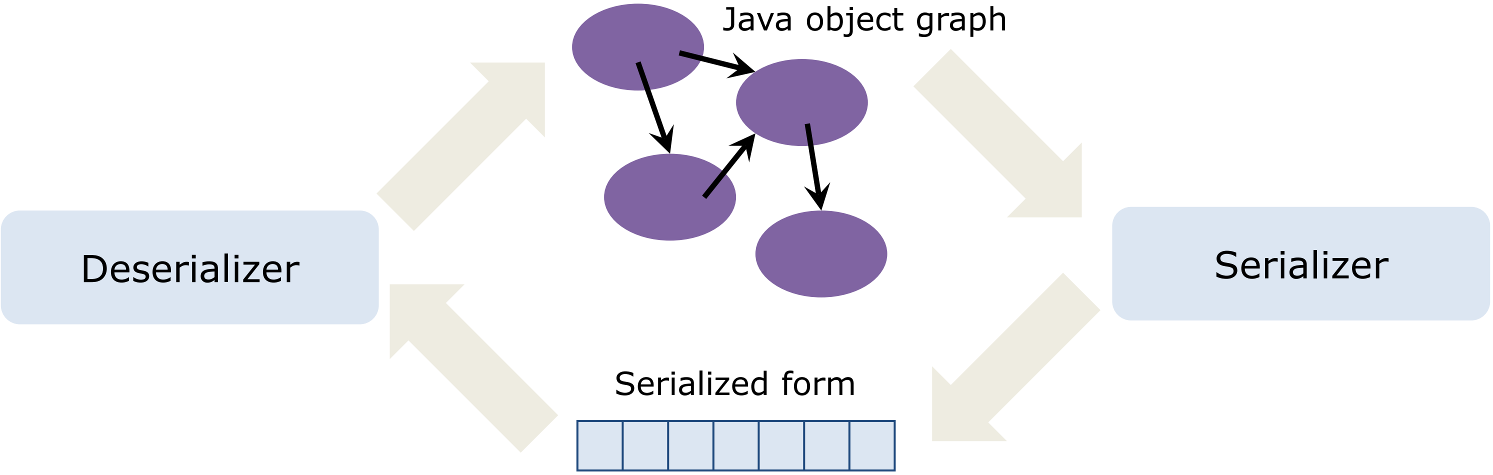


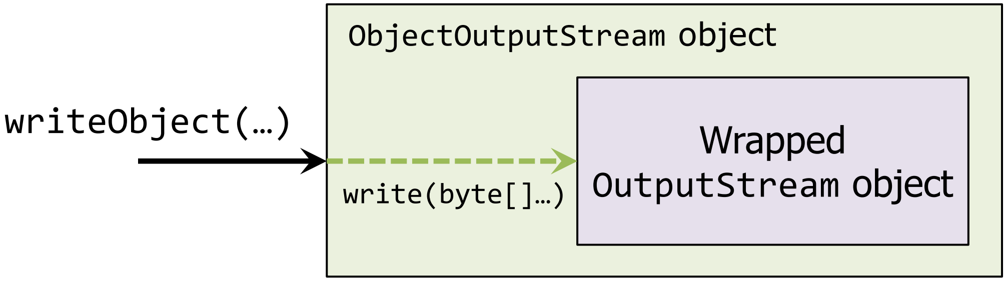
To the code!

SOFTENG 325 – Software Architecture

Java serialization

Java serialization

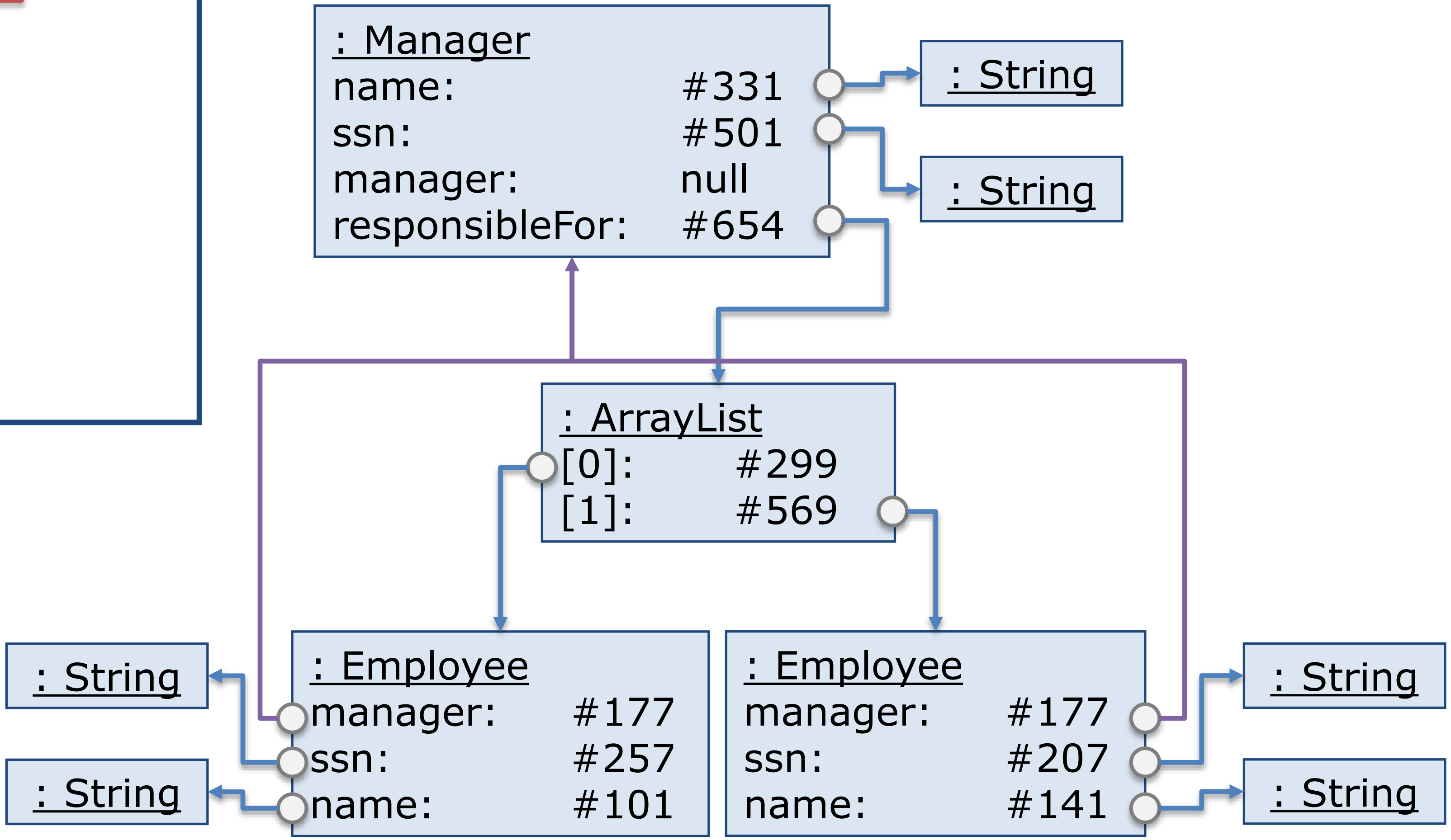




Java serialization

```
public class Employee implements Serializable {

    protected String name;
    protected String ssn;
    protected Manager manager;
    …
}

public class Manager extends Employee {

    private List<Employee> responsibleFor;
    …
}
```

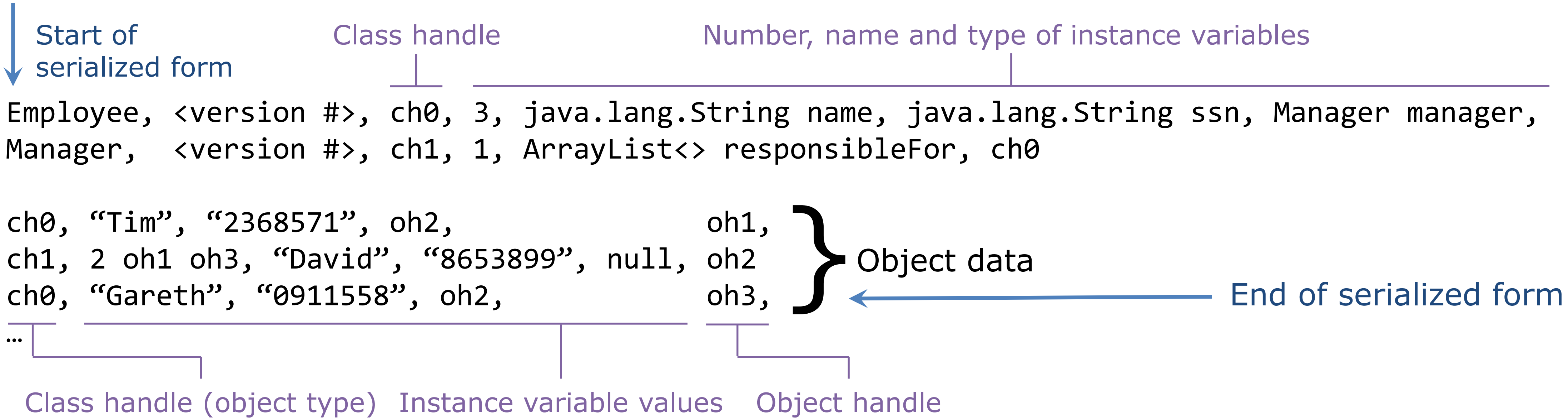
Address	Contents
#101	String object
#141	String object
#177	Manager object
#207	String object
#257	String object
#299	Employee object
#331	String object
#501	String object
#569	Employee object
#654	ArrayList



Java serialization

```
Manager mgr = new Manager("David", "8653899");
Employee e1 = new Employee("Tim", "2368571", mgr);
Employee e2 = new Employee("Gareth", "0911558", mgr);
```

- When serializing an object O, the following information is written out in binary form:
  - O's state - the values of O's instance variables
  - The state of all objects – once only – that are reachable from O
  - A description of each class, and its superclasses, of objects being written



Start of serialized form

Class handle

Number, name and type of instance variables

```
Employee, <version #>, ch0, 3, java.lang.String name, java.lang.String ssn, Manager manager,
Manager,  <version #>, ch1, 1, ArrayList<> responsibleFor, ch0

ch0, “Tim”, “2368571”, oh2,                 oh1,
ch1, 2 oh1 oh3, “David”, “8653899”, null,   oh2
ch0, “Gareth”, “0911558”, oh2,              oh3,
…
```

Object data

End of serialized form

Class handle (object type)

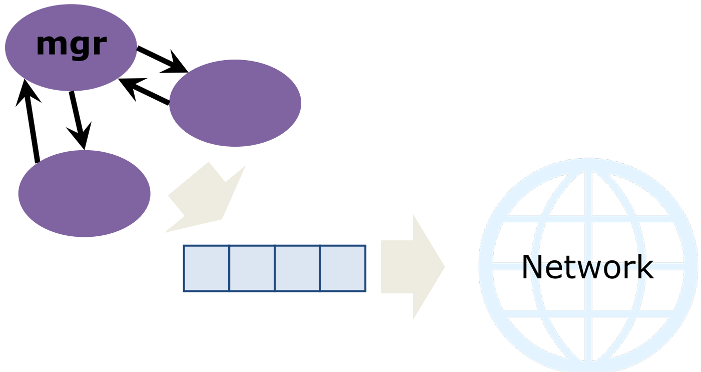
Instance variable values

Object handle

Applications of serialization

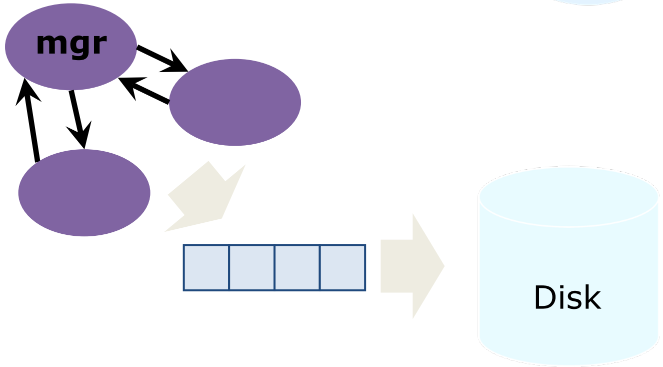
#1 Sending an object structure over a network connection

```
Manager mgr = …;
Socket socket = …;

ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
out.writeObject(mgr);
```



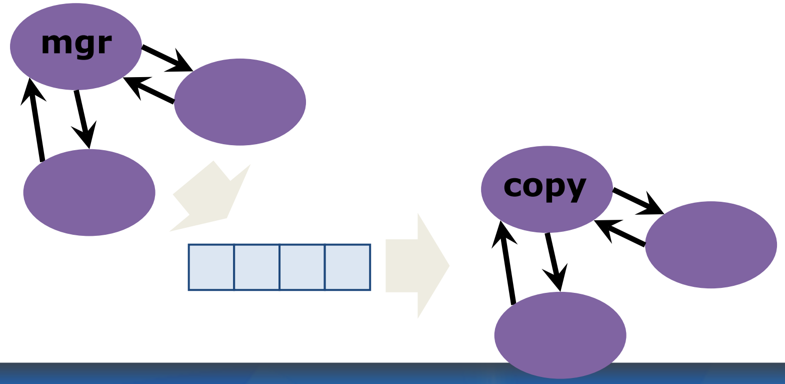
#2 Persisting an object graph to disk

```
Manager mgr = …;

OutputStream file = new FileOutputStream("employees.ser");
ObjectOutputStream out = new ObjectOutputStream(file);
out.writeObject(mgr);
```



#3 Making a deep copy of an object graph in memory

```
Manager mgr = …;

ByteArrayOutputStream bos = new ByteArrayOutputStream();
ObjectOutputStream out = new ObjectOutputStream(bos);
out.writeObject(mgr);

ByteArrayInputStream bis = new ByteArrayInputStream(bos.toByteArray());
ObjectInputStream in = new ObjectInputStream(bis);
Employee copy = (Employee) in.readObject();
```



What have we learned today?

- Packet switched networks – protocols like IP are unreliable and can lead to packets being dropped, corrupted, arriving out of sender order and duplicated
- Protocol layering
  - A layer exposes an interface to the layer above
  - Layers hide their implementations; one implementation can be substituted for another
  - Higher layers can add reliability to unreliable lower layers
    - TCP makes use of acknowledgment packets, checksums and sequence numbers to mask unreliability of the IP layer
- Java's API for TCP, including sockets and stream classes for converting data to/from byte representation, which is necessary for transmitting data over a network