Telecooperation Lab
Prof. Dr. Max Mühlhäuser

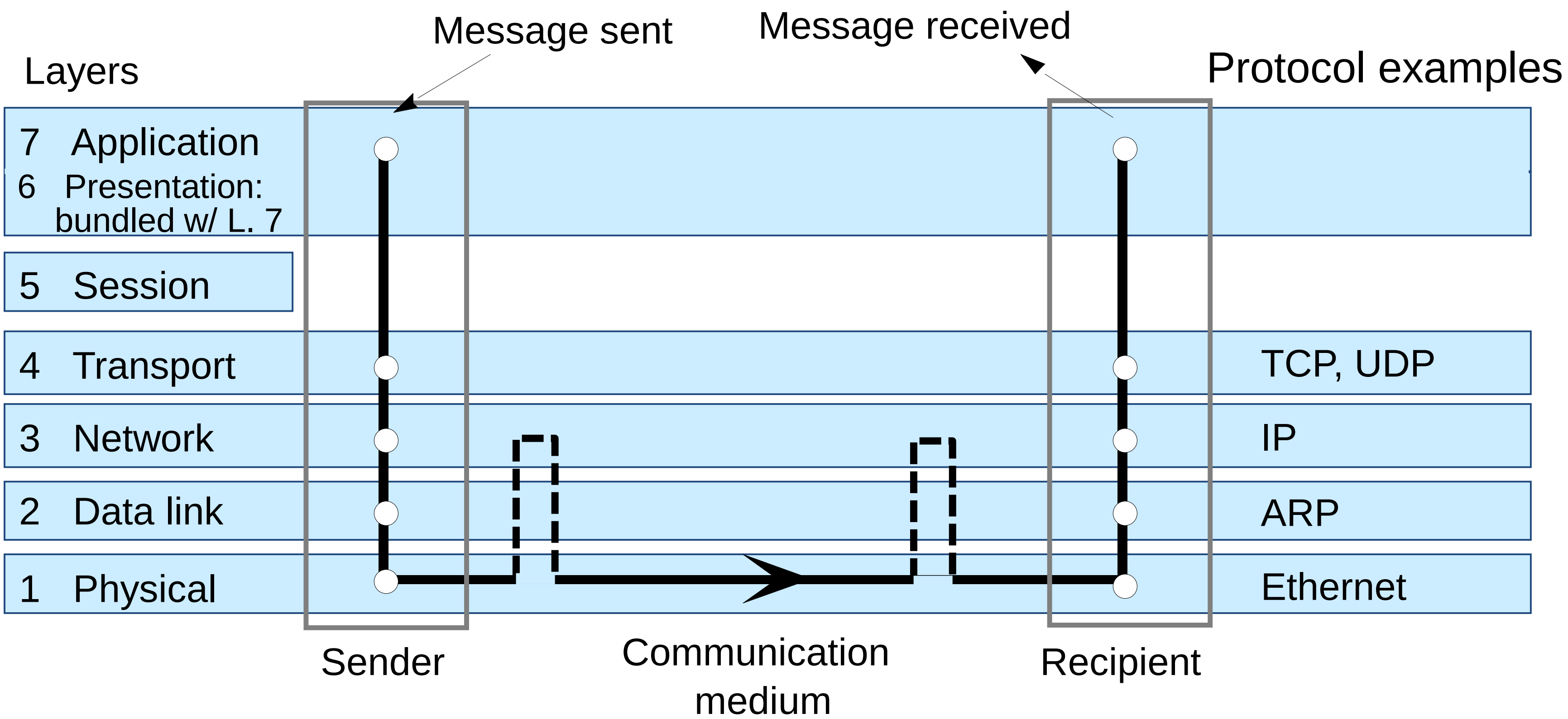
TK1: Distributed Systems - Programming & Algorithms

Chapter 2: Distributed Programming

Section 1: Mainstream Paradigms

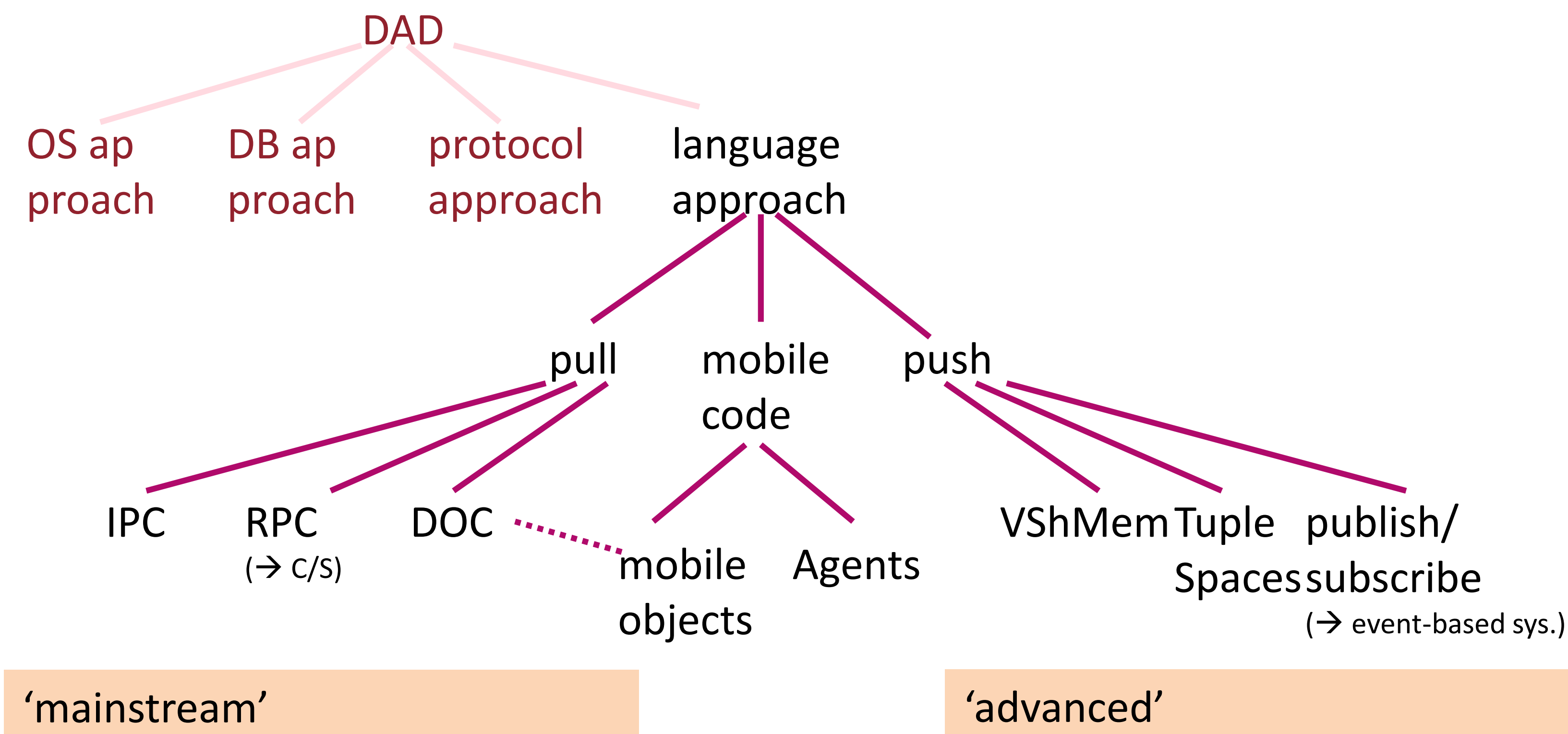
Lecturer: **Prof. Dr. Max Mühlhäuser, Dr. Immanuel Schweizer, Dr. Benedikt Schmidt**

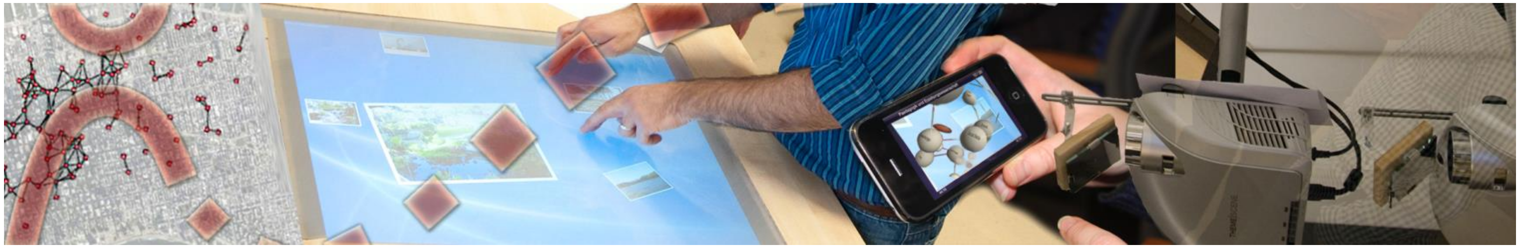
Internet Layer Architecture



Pragmatic Taxonomy

all in one, we get the following taxonomy for distributed application development (DAD):





2.1: MAINSTREAM PARADIGMS

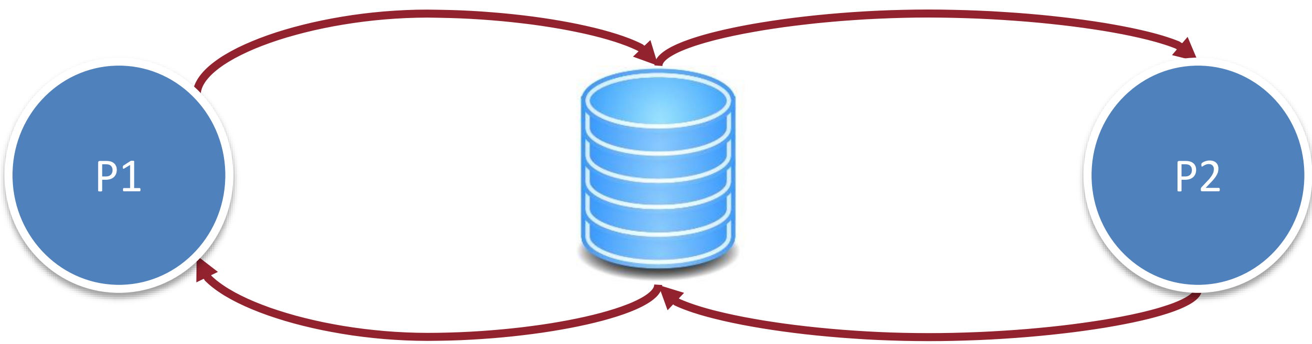
(1) IPC: Interprocess Communication

Interprocess Communication (IPC)

IPC = **Interprocess Communication** = a way of exchanging data between programs running at the same time

Interprocess Communication (IPC)

IPC = **Interprocess Communication** = a way of exchanging data between programs running at the same time

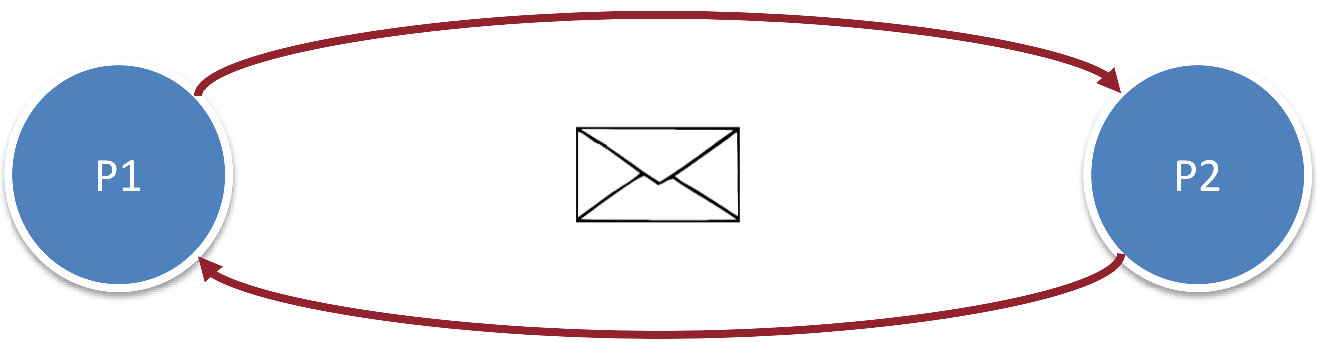


Shared Memory:

- Memory that can be simultaneously accessed by multiple programs
- Implicit communication

Interprocess Communication (IPC)

IPC = **Interprocess Communication** = a way of exchanging data between programs running at the same time



Message passing:

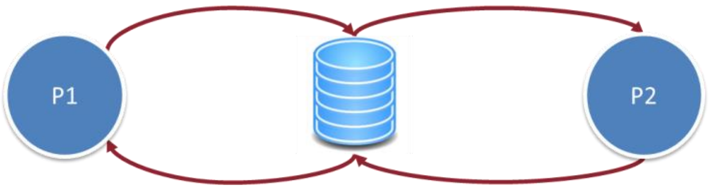
- Sending messages between programs / processes
- Explicit communication

Interprocess Communication (IPC)

IPC = **<u>I</u>nter<u>p</u>rocess <u>C</u>ommunication** = a way of exchanging data between programs running at the same time

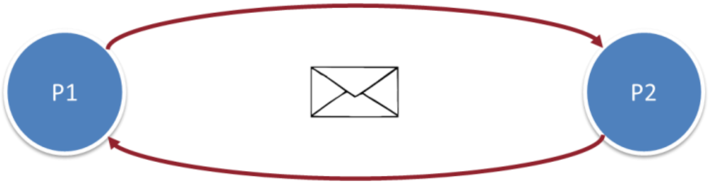
Shared Memory:

- Memory that can be simultaneously accessed by multiple programs
- Implicit communication



Message passing:

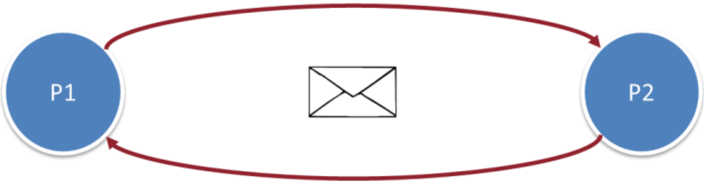
- Sending messages between programs / processes
- Explicit communication



Interprocess Communication (IPC)

Message passing:

- Sending messages between programs / processes
- Explicit communication



Socket:

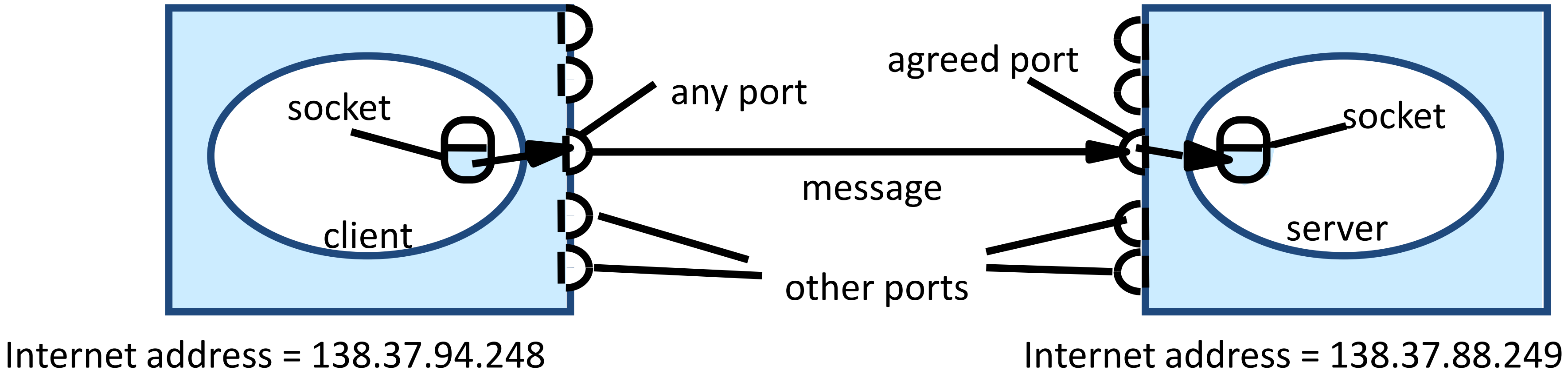
- „the API for the Internet“
- Internet Sockets
  - Connection-less (UDP) / Connection-oriented (TCP)
- Layer 4

Message Queues:

- Persistent queues
- De-coupling
- Layer 7

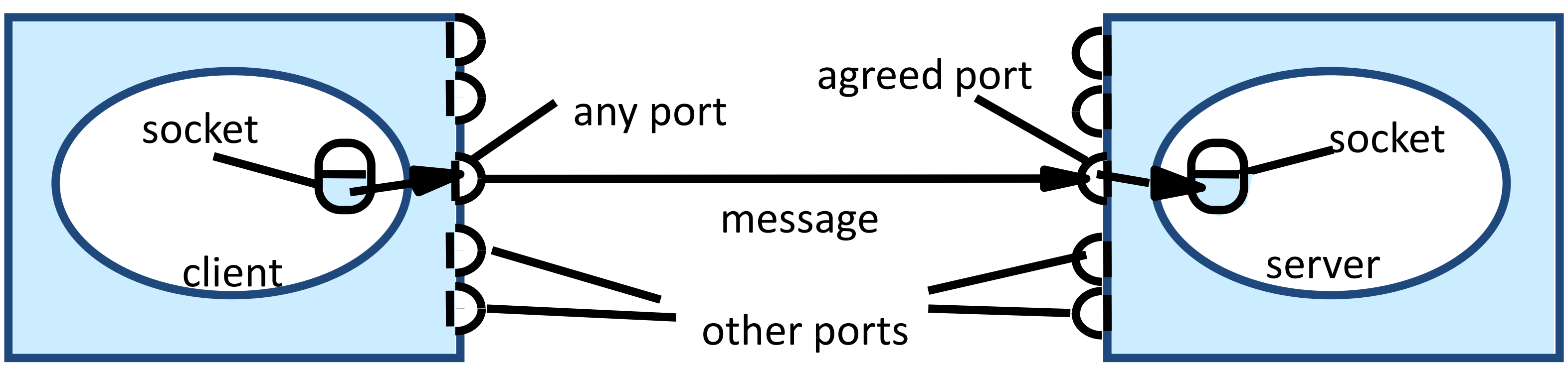
Sockets

- Messages are sent from/to **socket**
  - Socket address = (Internet address | port number)
  - Layer 4 protocol: Transport Layer
- A (unicast) port has one receiver but can have many senders
  - How to identify a connection if several connections may 'end' at a server's socket?
  - Dirty trick: pair (local socket; remote socket) is used → need further tricks for programming
- Different 'agreed' Port numbers identify different 'officially known' service types
  - Official assignments maintained by IANA; e.g., FTP (21), SSH (22), etc.



Sockets

- Purpose: End-to-end communication
- Datagrams vs. Byte Streams
  - Connection-less vs. connection-oriented
- Reliability
- Flow Control
- Congestion Control
- Multiplexing



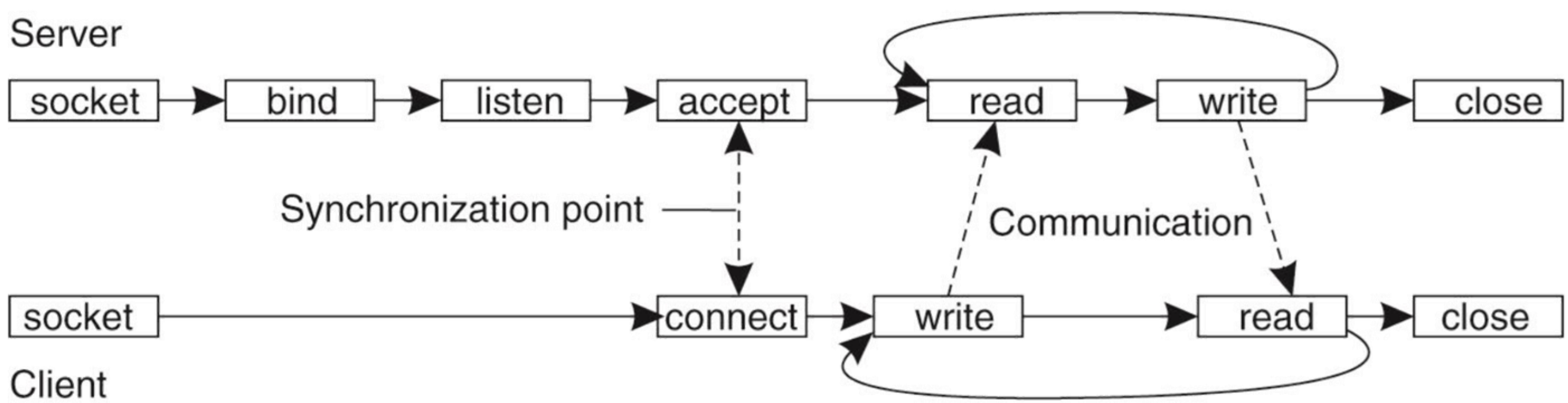
Internet address = 138.37.94.248

Internet address = 138.37.88.249

Sockets

Example: The Berkeley socket interface, which has been adopted by all Posix systems, as well as Windows.

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection



Synchronization

Synchronous

- sender and receiver synchronize (wait for each other) at every message exchange

Asynchronous

- **send** is non-blocking, sending process continues while transmission is in progress
- **receive** is blocking (wait for message to arrive) or non-blocking (interrupt/callback)
  - Blocking receive has no disadvantages when multithreading available
    - one thread waits for message, other thread(s) continue(s)
    - well … : multiple receivers & comm. patterns: what if ‚this receive‘ becomes obsolete?
  - Non-blocking operations appear more efficient, but extra complexity needed in receiving process to associate received message with request

Reliability

Reliable communication is defined in terms of validity and integrity

- Validity: any message in the outgoing message buffer is eventually delivered to the incoming message buffer
  - At-least-once
- Integrity: the message received is identical to the one sent, and no messages are delivered twice
  - At-most-once

User Datagram Protocol (UDP)

- UDP transmits datagrams **without acknowledgement** or **retries**

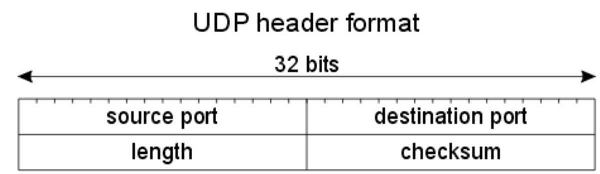
- Sockets API for UDP
  - *sender* sends from any socket, specifies destination in datagram
  - *receiver* creates socket and binds it to *(local address, local port)*

```
DatagramSocket sock = new DatagramSocket();
byte requestBuffer[] = args[1].getBytes();
InetAddress addr =
   InetAddress.getByName(args[0]);
DatagramPacket request = new
   DatagramPacket(requestBuffer,
   requestBuffer.length, addr, SERVER_PORT);
sock.send(request);
sock.close();
```

```
DatagramSocket sock = new
   DatagramSocket(SERVER_PORT);
byte[] buffer = new byte[256];
for(;;) {
   DatagramPacket request = new
   DatagramPacket(buffer, buffer.length);
   sock.receive(request);
   }
```

Characteristics of UDP

- **Message size**:
  - 65,535 byte: bigger messages are truncated!
  - Limits given by IP layer: 65,507 bytes (8 byte UDP Header; 20 byte IP header)



- **Synchronization**:
  - *sends* are non-blocking
  - *receive* gets msg from local queue, msg discarded if socket unbound
    - Receive is blocking, timeout can be specified (setSoTimeout)
- **Receive from any**: *receive* receives msg from any origin
  - Sender address returned in datagram
- No congestion or flow control

Characteristics of UDP

- **Reliability**
  - Integrity: Datagram header and payload are protected by a **checksum**
    - Note: checksum *not* used for correcting errors!
  - Validity: No
- **Omission failures**: messages may be dropped occasionally, because of checksum error or no buffer space at sender or receiver
- **Ordering**: messages can sometimes be delivered out of sender order

- Advantages of UDP
  - Neither sender nor receiver need to keep state about the connection (beside the bind on the receiver side)
  - No transmission of extra messages
  - Low latency

- Usage examples: DNS, VoIP, RTP

Transmission Control Protocpl (TCP)

- TCP abstraction: **stream of bytes** („pipe“)
  - To which data may be written and from which data may be read
  - Bi-directional

```
Socket sock = new Socket(args[0], SERVER_PORT);
// connect & accept hidden behind io operations
   for creation of Input / Output Streams
DataInputStream in = new
   DataInputStream(sock.getInputStream());
DataOutputStream out = new
   DataOutputStream(sock.getOutputStream());
// send & receive hidden behind io operations for
   read / write
out.writeUTF(args[1]);
```

```
ServerSocket listenSock = new
   ServerSocket(SERVER_PORT);
for(;;) {
   Socket sock = listenSock.accept();
   //recognize  dirty trick: ServerSocket plus Socket for
   each individual ‚connection‘
   DataInputStream in = new
   DataInputStream(sock.getInputStream());
   DataOutputStream out = new
   DataOutputStream(sock.getOutputStream());
   String data = in.readUTF();
   sock.close();
}
```

Characteristics of TCP

- The stream abstraction hides
  - **Message sizes**: automatic blocking, fragmentation, reassembly
    - Along with Internet, TCP ‚outruled‘ other connection-oriented L4-protocols in the 1980s
    - All others kept messages intact (despite fragmentation/reassembly ‚on the way‘)
    - TCP: you may send `[abc] [def] [ghi]` but receive `[abcdefg] [hi]`
  - **Lost messages**
    - TCP uses an acknowledgement scheme (again, different from all other protocols: byte-numbers!)
    - Sender retransmits message if no ack received before timeout
  - **Message duplication and ordering**
    - Message identifiers in each IP packet facilitate the detection of duplicates and reordering of messages
  - **Message destinations**
    - Pair of communicating processes establish a connection (*connect/accept*) before they can communicate over a stream
  - **Flow control**
    - Fast writer is blocked until reader has consumed sufficient data
  - **Congestion control**
    - Sender reduces traffic, if congestion is detected

TCP/IP

- Reliability of TCP
  - Integrity:
    - Corrupt packets are detected & rejected using checksums
    - Duplicate packets are detected & rejected using sequence numbers
    - (no “integrity” of original messages, see last slide)
  - Validity:
    - TCP uses timeouts and retransmissions to handle lost packets

- Limitation: If a connection breaks before an explicit *close* operation
  - Processes cannot distinguish between network failure and failure of the remote process
  - Communicating processes cannot tell whether the messages they sent recently have been received or not

- Usage examples: HTTP, FTP, Telnet, SMTP

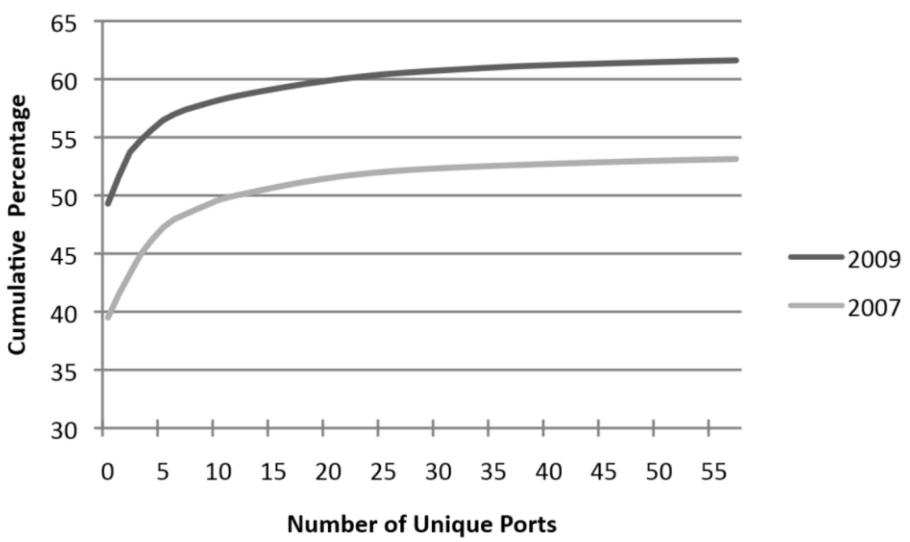
TCP/IP problems

- TCP implementation must break down byte stream to messages
  - When to send a message?
    - Send when MSS (max. segment size) reached? - only suitable for large quantities of data
    - Interactive apps may send single keystrokes and expect low latency!
  - old implementation: send data no later than timeout T (T=0.5 sec)
- Today: **Nagle's algorithm**. Better in fast networks (, but feature interaction with TCP delayed ACKs)
  - Nagle's algorithm:
    - Buffer all user data if unacknowledged data is outstanding
    - Send all data if everything is ACK'd or have a full packet (MSS) size worth of data to send
  - For certain apps: use explicit flush op. or disable Nagle (setTcpNoDelay)
- Today, many RPC-like interactions are also based on TCP/IP
  - However, message-based traffic is a bad match

Internet Transport Today

Mostly TCP and UDP

- Ratio UDP / TCP ~ 0.1-0.2
- Consolidation of content



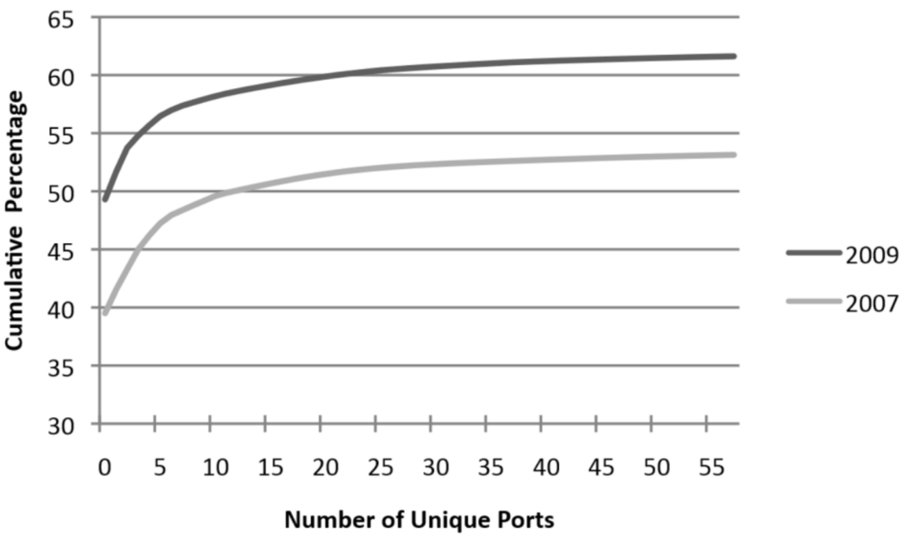
- Growth of UDP (Real-time Applications & Streaming)
- Decreasing number of protocols and ports
- Largest growth not in transit carriers, but large CDNs, hosting, and consumer networks

Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. 2010. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM 2010 conference* (SIGCOMM '10). ACM, New York, NY, USA, 75-86

Internet Transport Today

Mostly TCP and UDP

- Growth of UDP (Real-time Applications & Streaming)
  - UDP has no congestion control
- Consolidation of content
  - TCP is not multi-homed



Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. 2010. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM 2010 conference* (SIGCOMM '10). ACM, New York, NY, USA, 75-86

Stream Control Transmission Protocol (SCTP)

- Stream Control Transmission Protocol (SCTP) …
  - is message-oriented like UDP
  - ensures reliable, in-sequence transport of messages
  - supports congestion control like TCP
- Features:
  - Flexible message-oriented streaming
    - Each message in a stream is assigned a sequence number for ordering
    - Receiver can enable/disable **ordering** and/or **reliability**
    - Sender can enable/disable **fragmentation** and/or **Nagle**
  - **Multi-streaming**: multiple independent message streams can be multiplexed into a single connection
    - Independent message ordering in different streams
    - Application: e.g., Web browser sends text in one stream, images in another
  - **Multi-homing**: both endpoints can have multiple IP addresses for failover

Datagram Congestion Control Protocol (DCCP)

- For applications requiring unreliable, timely delivery
- UDP has no congestion control -> senders can „flood“ the network
- DCCP is roughly
  - TCP without bytestream semantics and reliability
  - UDP with congestion control, handshakes, ACKs (optional)
- Features
  - Message transport unreliable, separate header/data checksums
  - Connection-oriented
    - Connection setup and teardown for middlebox (NAT…) traversal
  - Reliable ACKs
    - Gives knowledge about congestion along ACK path
  - Feature negotiation mechanism for congestion control (CC):
    - CC scheme not fixed, instead defined by Congestion Control ID (CCID)

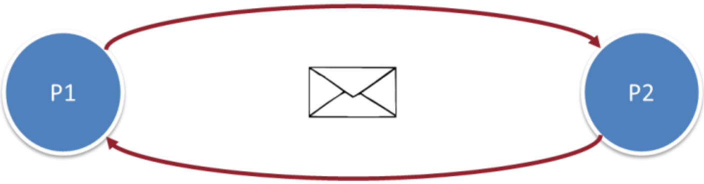
Comparison of Transport-layer Protocols

Feature Name	UDP	UDP Lite	TCP	SCTP	DCCP
Connection oriented	-	-	Yes	Yes	Yes
Reliable transport	-	-	Yes	Yes	-
Unreliable transport	Yes	Yes	-	Yes	Yes
Preserve message boundary	Yes	Yes	-	Yes	Yes
Ordered delivery	-	-	Yes	Yes	-
Unordered delivery	Yes	Yes	-	Yes	Yes
Data checksum	Yes	Yes	Yes	Yes	Yes
Partial checksum	-	Yes	-	-	Yes
Path MTU	-	-	Yes	Yes	Yes
Congestion control	-	-	Yes	Yes	Yes
Multiple streams	-	-	-	Yes	-
Multi-homing	-	-	-	Yes	-
Bundling / Nagle	-	-	Yes	Yes	-

Interprocess Communication (IPC)

Message passing:

- Sending messages between programs / processes
- Explicit communication



Socket:

- „the API for the Internet“
- Internet Sockets
  - Connection-less (UDP) / Connection-oriented (TCP)
- Layer 4

Message Queues:

- Persistent queues
- De-coupling
- Layer 7

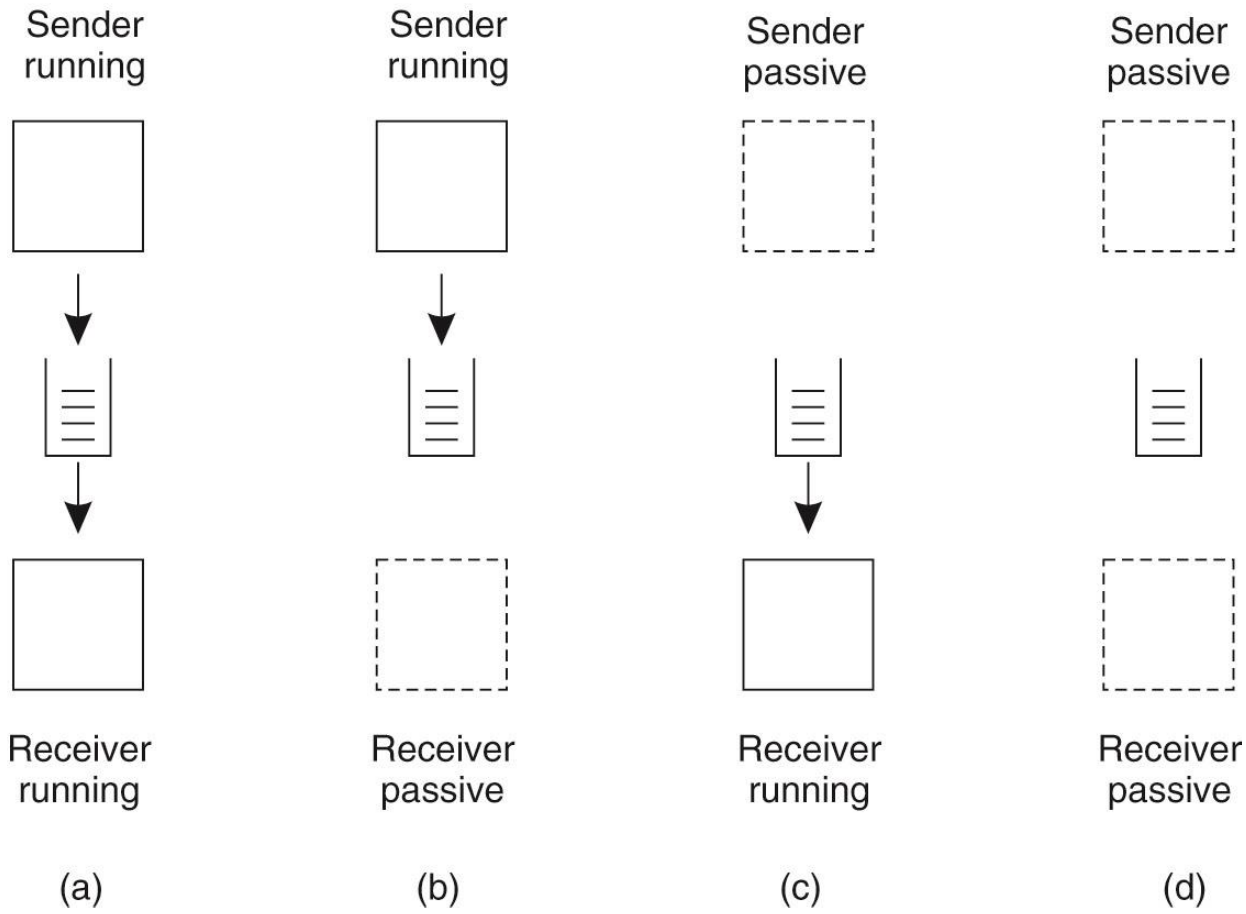
Message Queues

Essence:

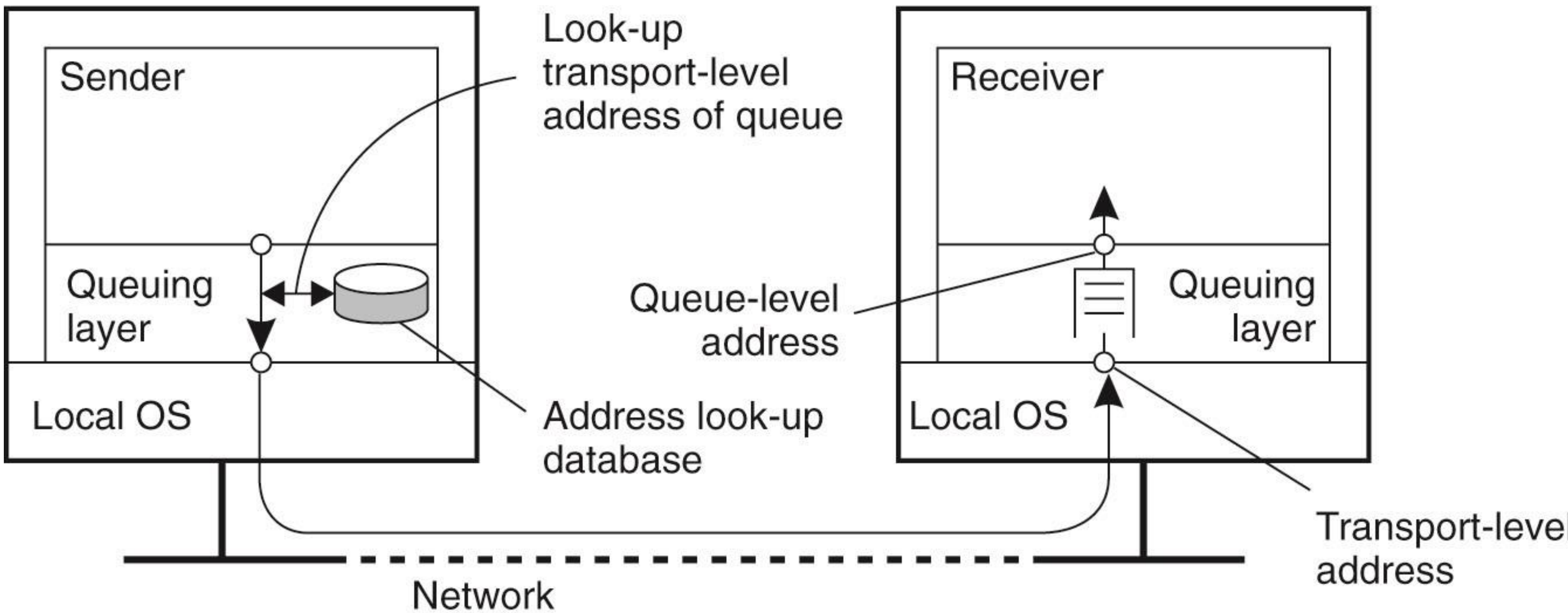
Asynchronous persistent communication through support of middleware-level queues. Queues correspond to buffers at communication servers

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

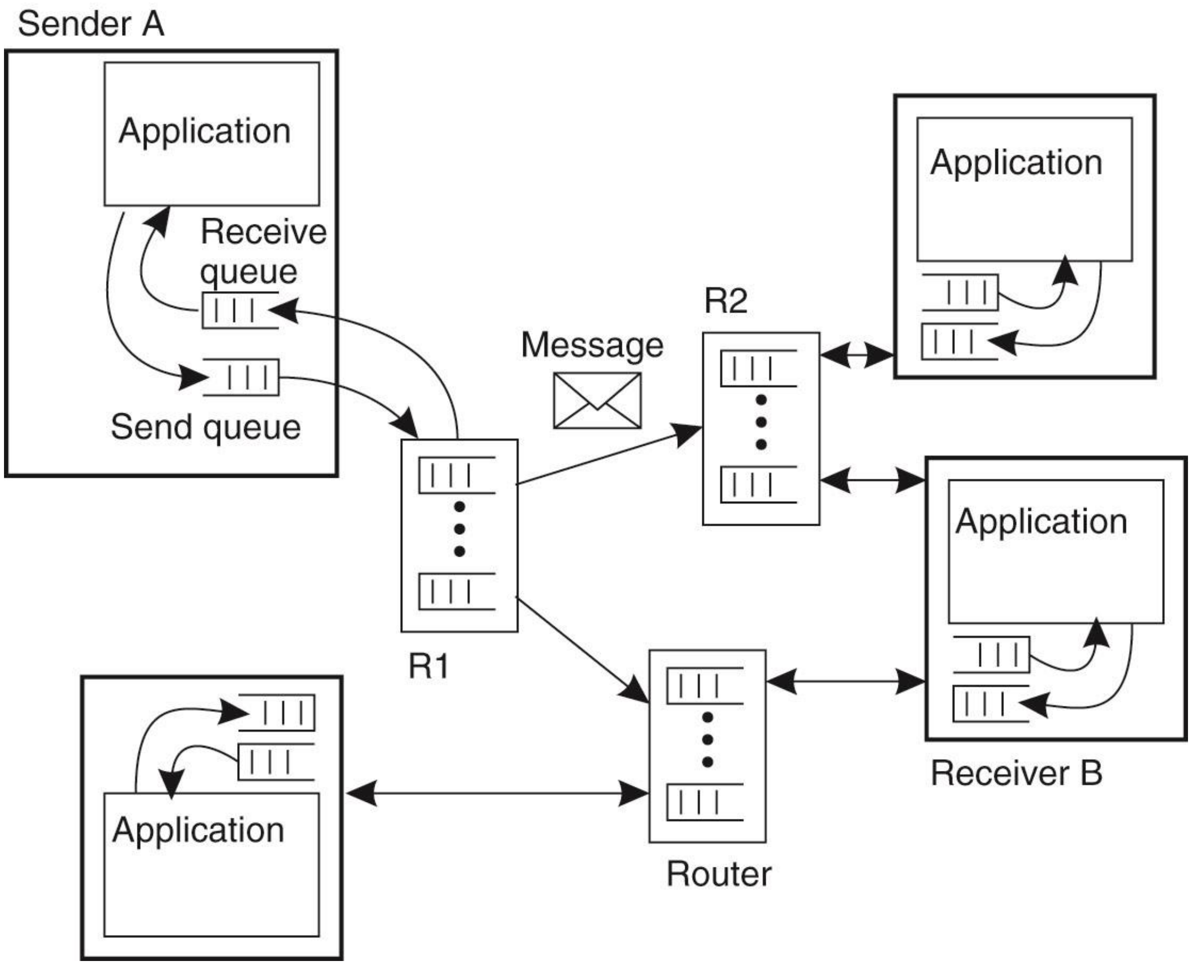
Message Queues



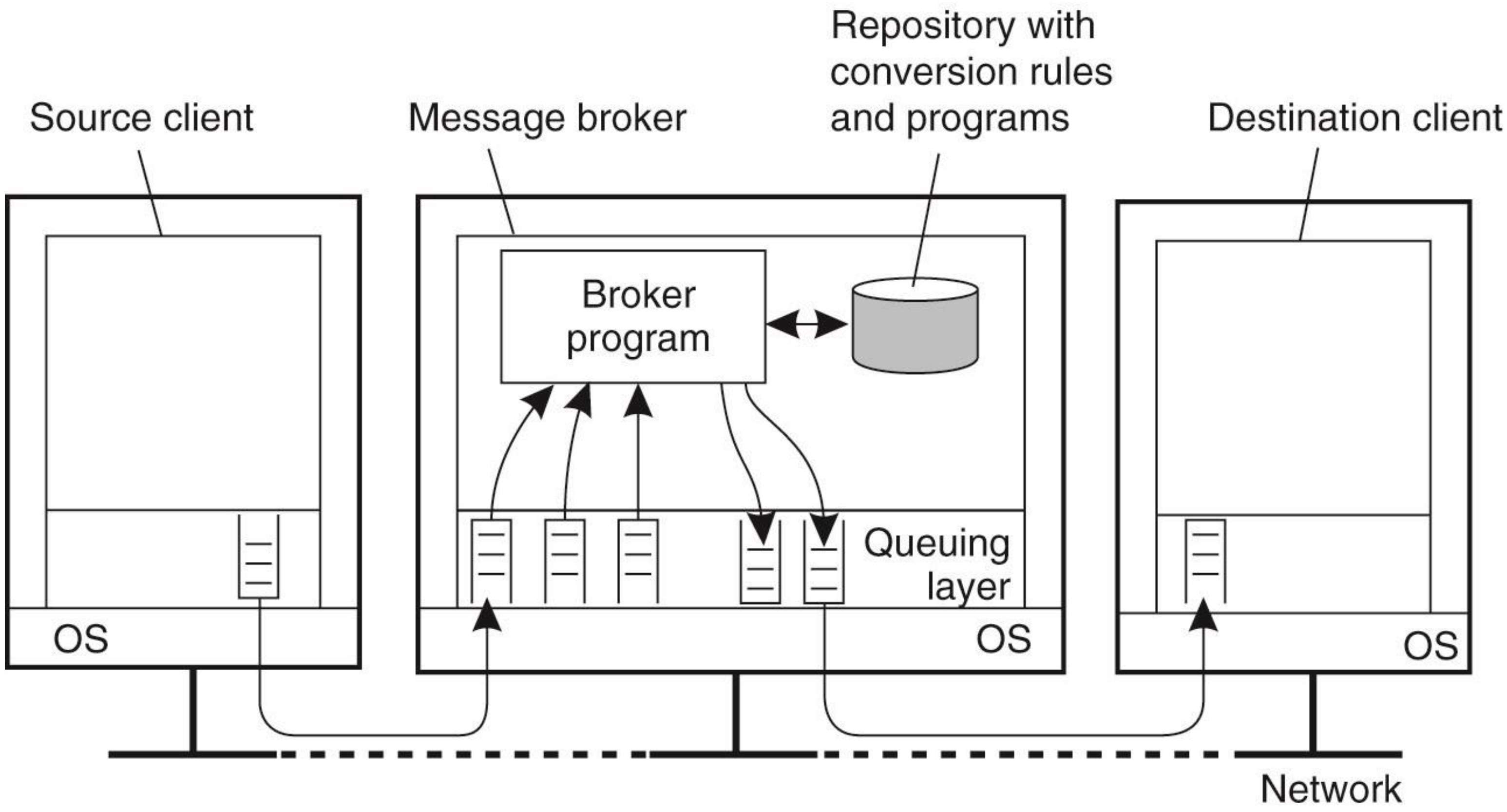
Message Queues



Message Queues

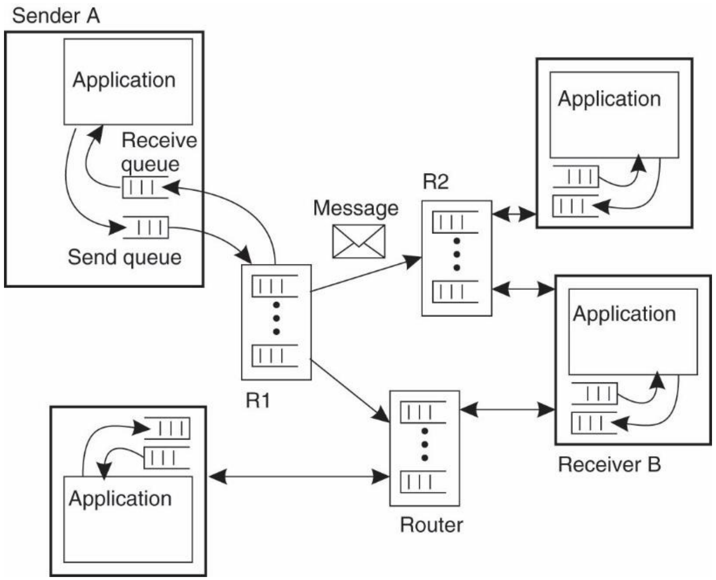


Message Broker



Message Queues

- Persistent Queues
  - Not small buffers
- De-coupling
  - Sender and / or Receiver can be offline
- Layer 7
  - Make use of Sockets etc. to connect
- Message Brokers
  - Non-homogenous environments
  - Translate between systems



Advanced Message Queuing Protocol (AMQP)

Essence:

AMQP defines only the protocol to transfer messages between two nodes without any assumptions. The queues between different nodes are called distribution nodes.

Primitive	Meaning
Open	Opens a connection by expressing capabilities
Close	Terminating a connection
Attach	Initiate a new link to receive or send messages
Detach	Tear down a link
Transfer	Sent a message (unidirectional)
Flow	Control the flow on a given link
Settled	Agree on a common state for a transfer
Disposition	Communicate changes in state or settlement
Session	Group different links into bidirectional, sequential conversations

Advanced Message Queuing Protocol (AMQP)

Essence:

AMQP defines only the protocol to transfer messages between two nodes without any assumptions. The queues between different nodes are called distribution nodes.

Distribution Nodes define:

- Standard outcomes for transfers (e.g., accept, reject)
- Indicating competing- and non-competing-consumers (Move, Copy)
- Create nodes on-demand
- Apply filters to refine messages

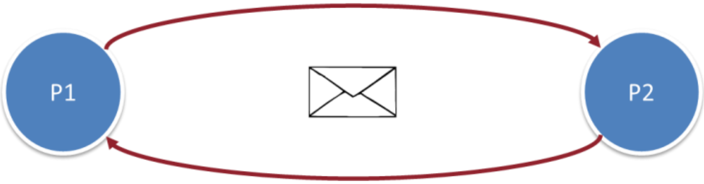
AMQP defines a simple framework, but can be further extended

Example: Noisemap Gamification

Interprocess Communication (IPC)

Message passing:

- Sending messages between programs / processes
- Explicit communication



Socket:

- „the API for the Internet“
- Internet Sockets
  - Connection-less (UDP) / Connection-oriented (TCP)
- Layer 4

Message Queues:

- Persistent queues
- De-coupling
- Layer 7