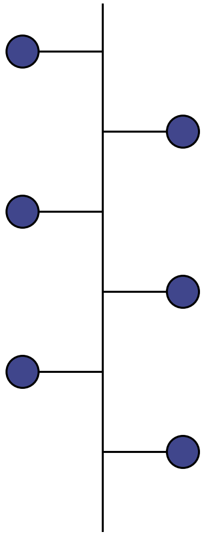
TechGI IV - ComDis

ComDis

Introduction to
Communication Networks
and **Dis**tributed Systems

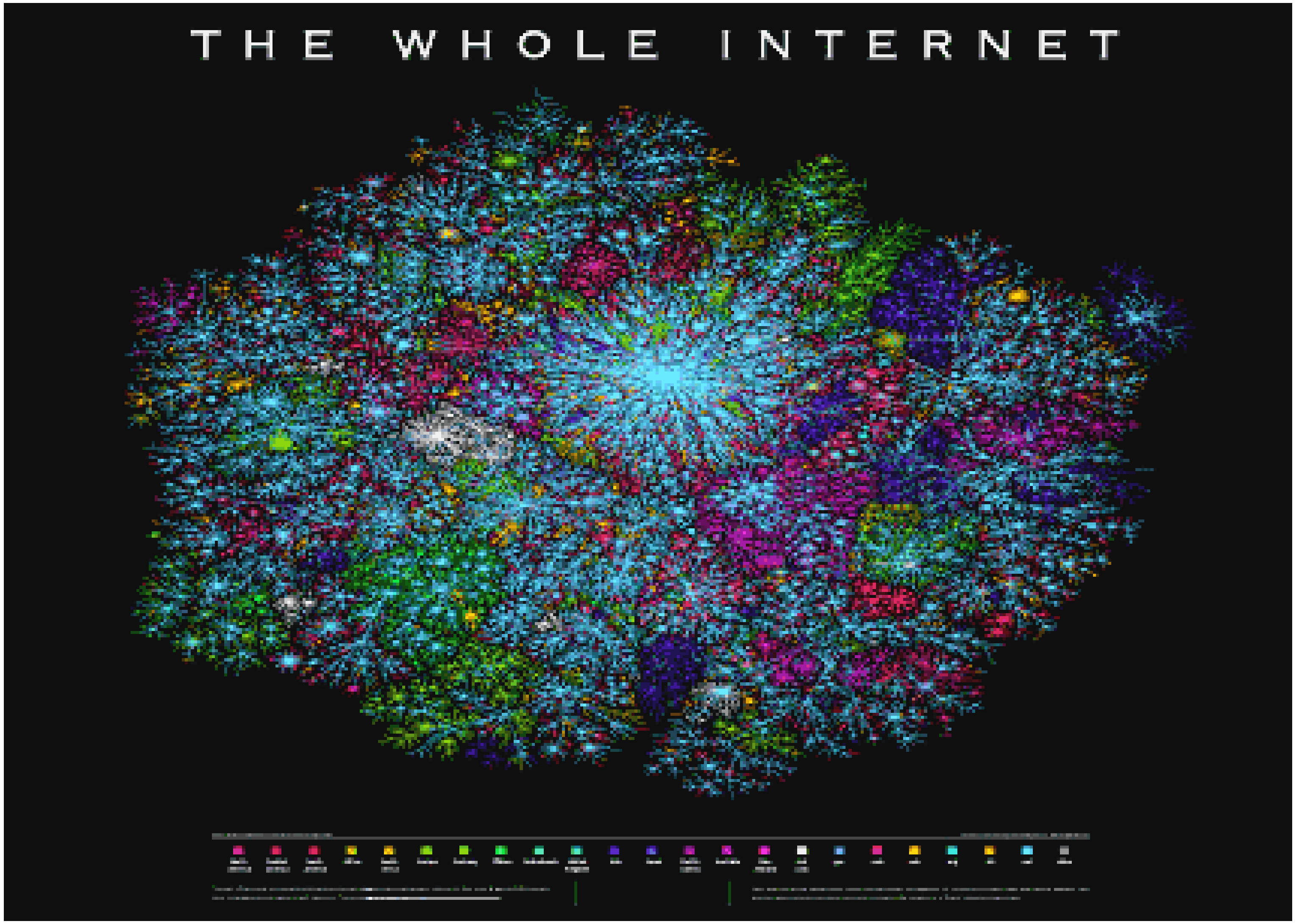


Unit 2: Reference Models and beyond…

Prof. Dr.-Ing. Adam Wolisz

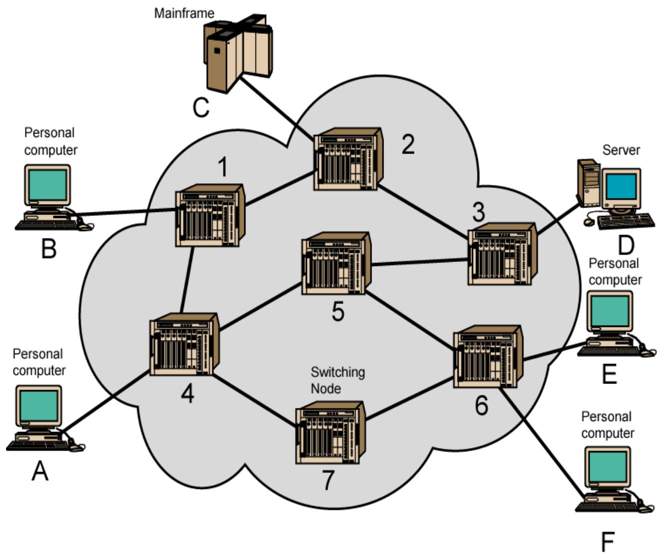
TKN **Telecommunication Networks Group**

Internet maps

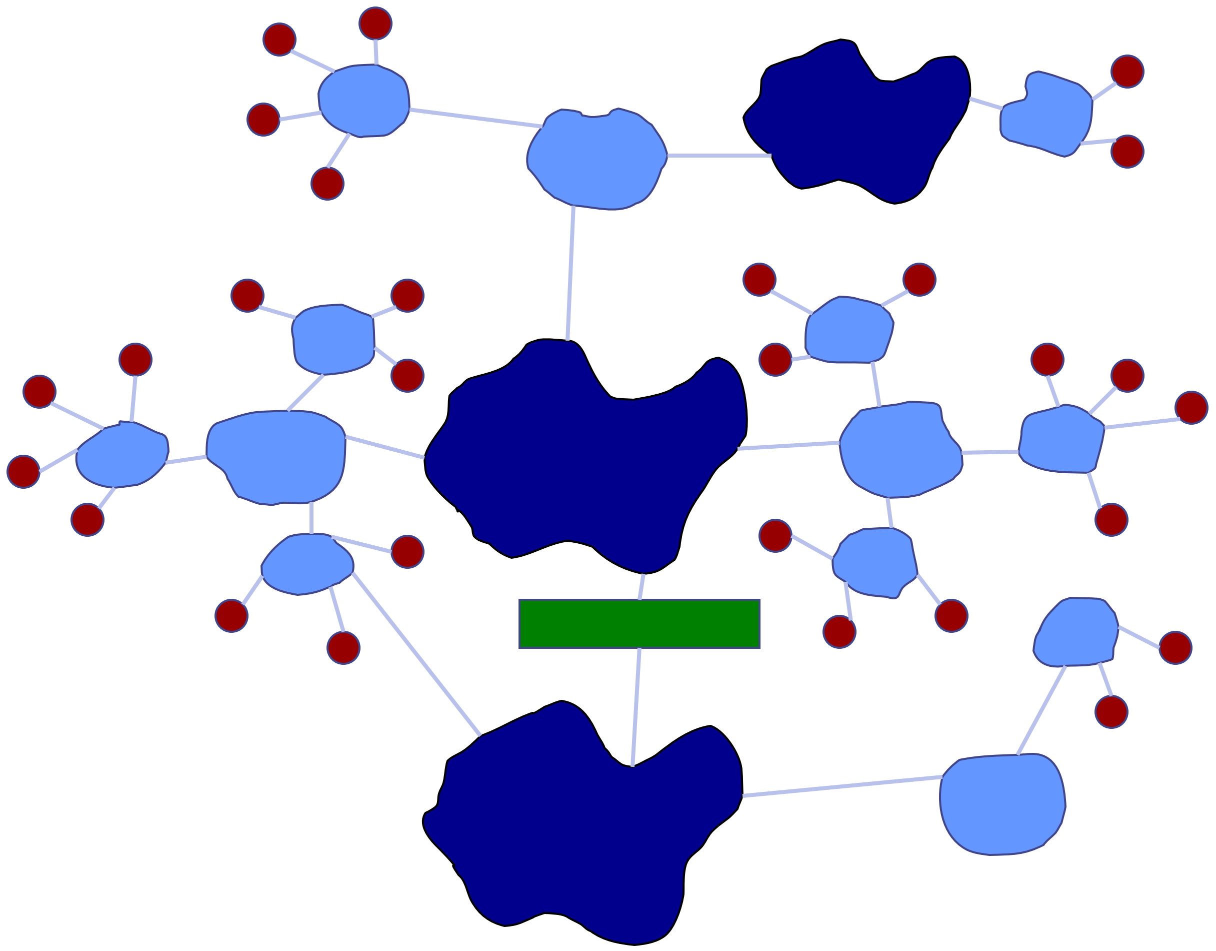


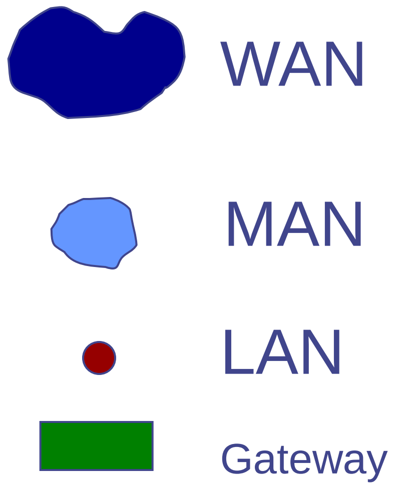
Large Networks need structure! Why?

- Scaling
  - Remember: each switch knows route to each destination…
  - Hierarchy usually simplifies a lot…
- Locality
  - Close hosts are clustered, Local networks
- Heterogeneity
  - Different applications (e.g. control, sensing) have requirements best served by NNN
  - Multiple technologies used….
- Administration
  - Who sets the rules for usage ???



Internet: Interoperability vs. Heterogeneity





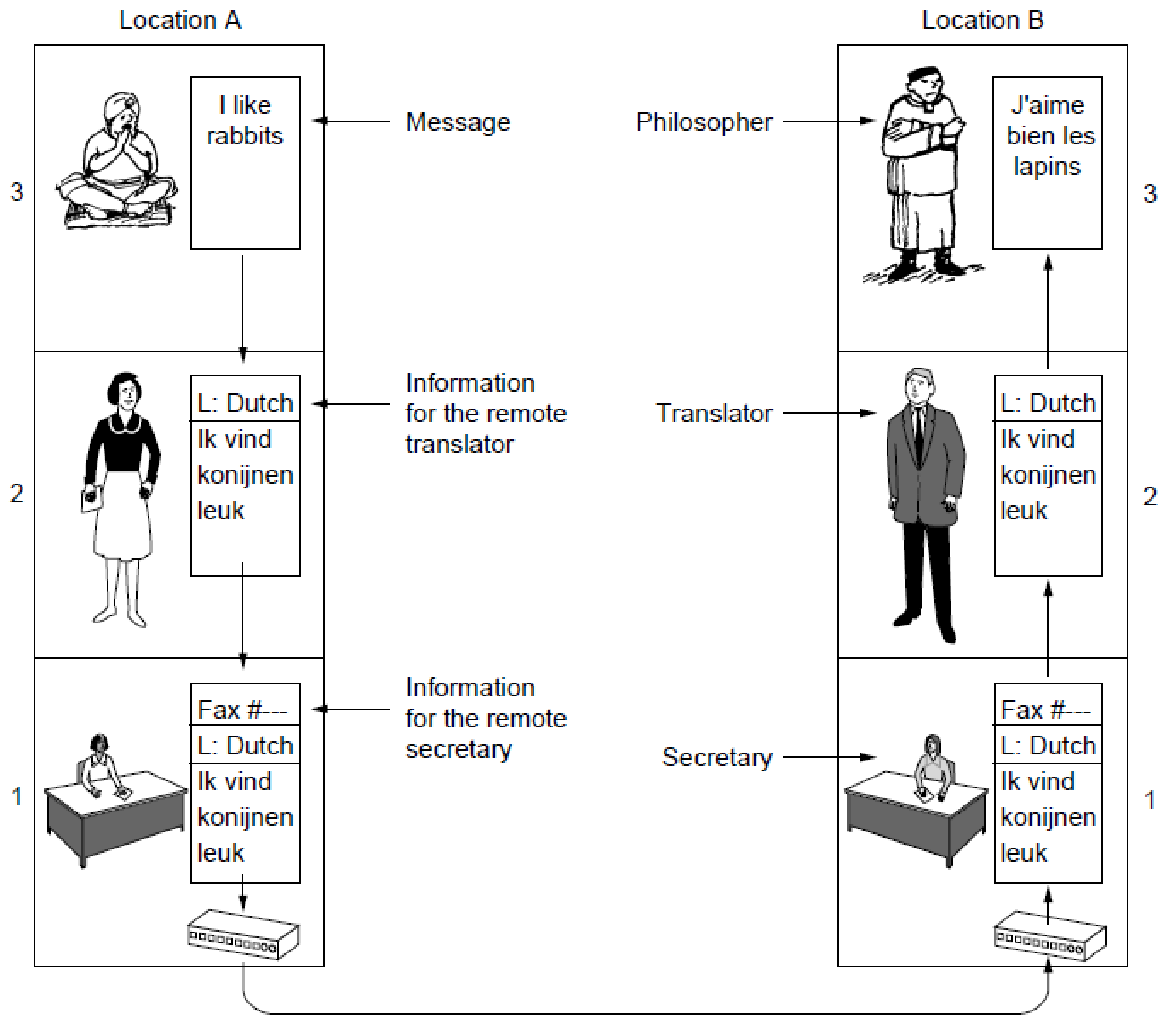
See:The Brief History of the Internet

Do you remember the philosophers…

Philosophers

Assistants

Office clerks



Computer Networks, Fifth Edition by Andrew Tanenbaum and David Wetherall, © Pearson Education-Prentice Hall, 2011

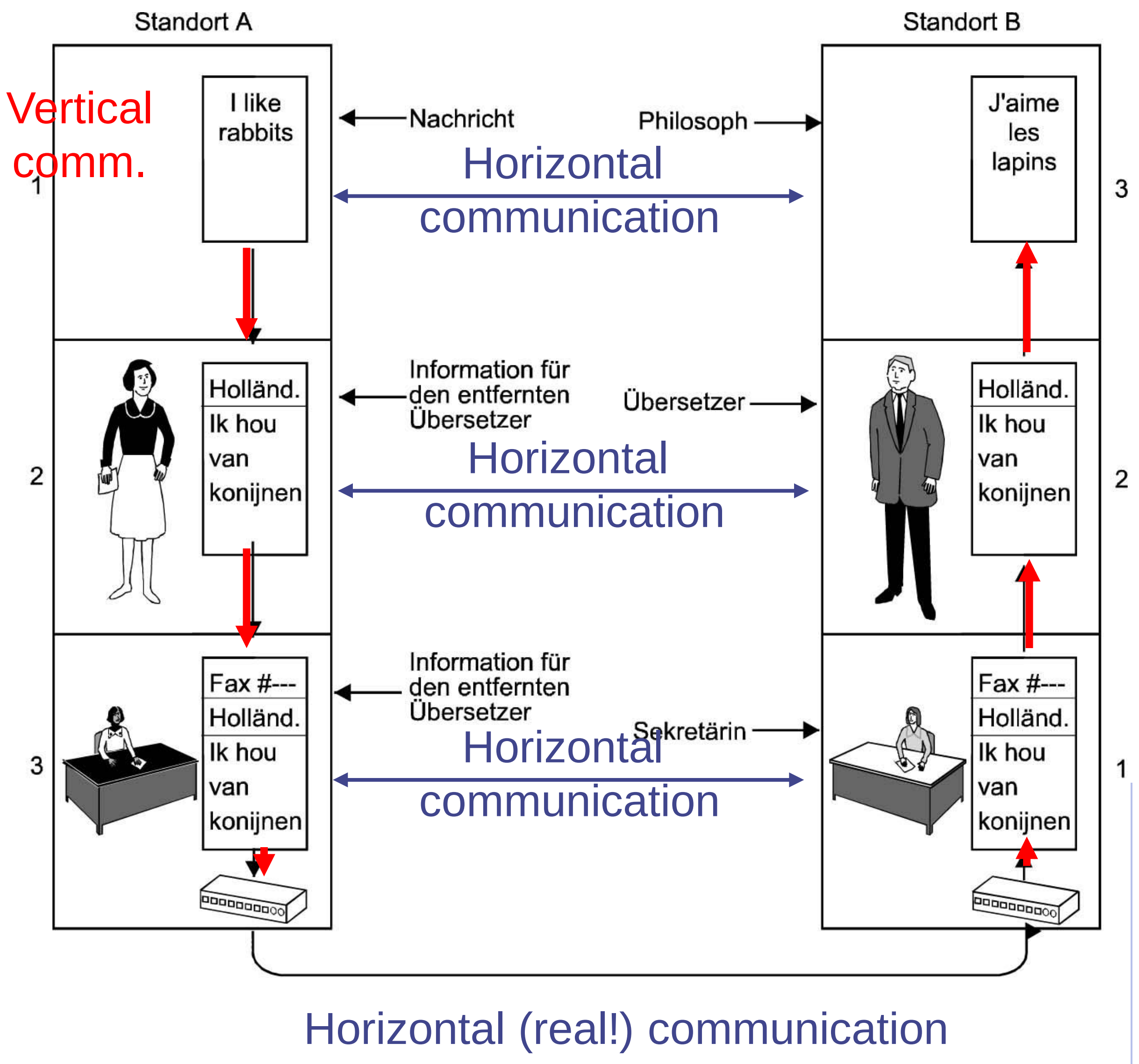
The reference model

- To keep complexity of communication systems tractable, division in subsystems with clearly assigned responsibilities is necessary – layering
- Each layer (and the communication system as a whole) offers a particular service
  - Services become more abstract and more powerful the higher up in the layering hierarchy
- To provide a service, a layer has to be distributed over remote devices
- Remote parts of a layer use a protocol to cooperate
  - Make use of service of the underlying layer to exchange data
  - Protocol is a horizontal relationship, service a vertical relationship
- Layers/protocols are arranged as a (protocol) stack
  - One atop the other, only using services from directly beneath

  $\Rightarrow$Strict layering

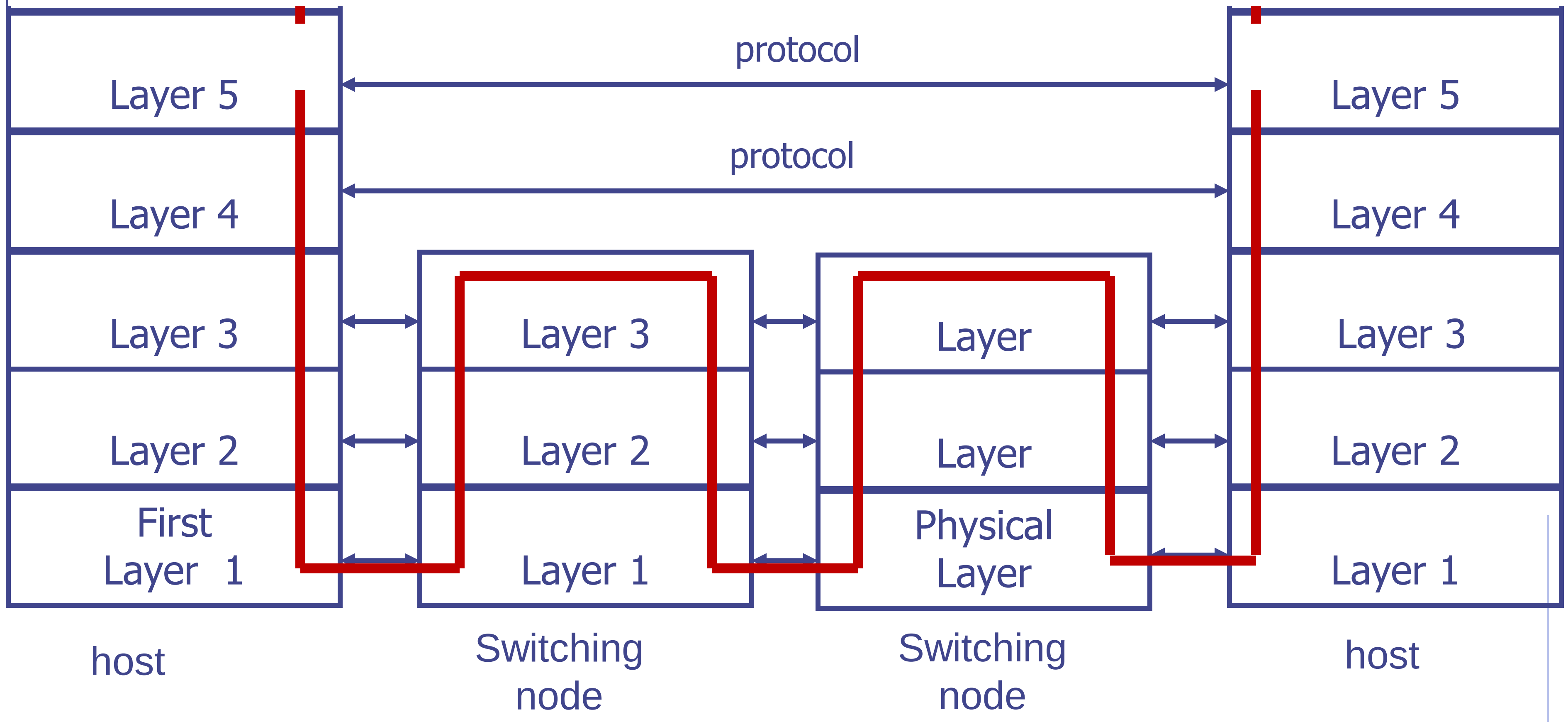
Analogy: Nested layers as nested translations

- Vertical vs. horizontal communication
  - Vertical: always real
  - Horizontal: may be real or virtual



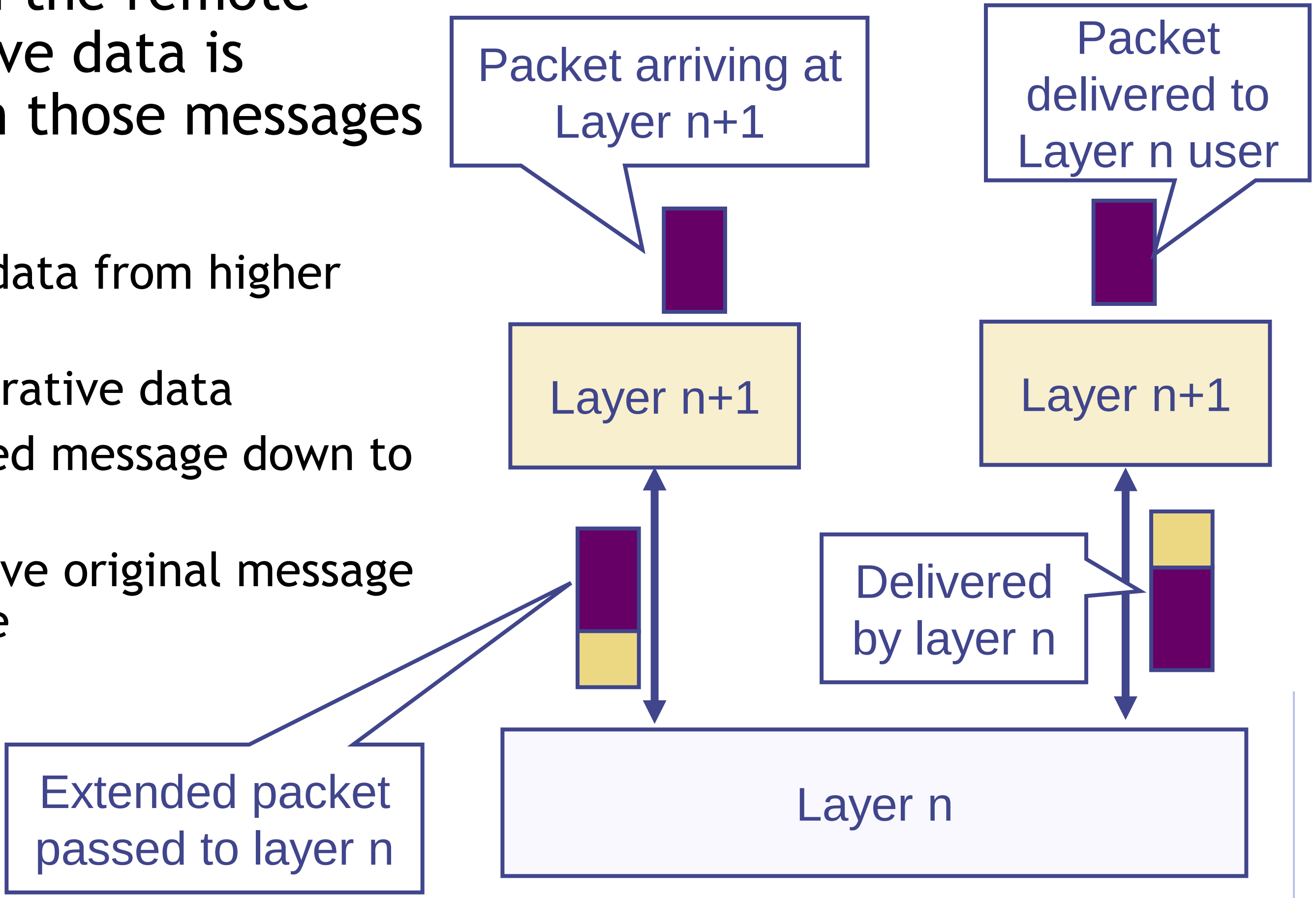
Multi-layer Architecture

- Number of Layers, and { services, naming and addressing conventions) / Layer
- Functions to be executed in each layer
- Protocols: (host- to – host, node –to – node, Host-to-node)



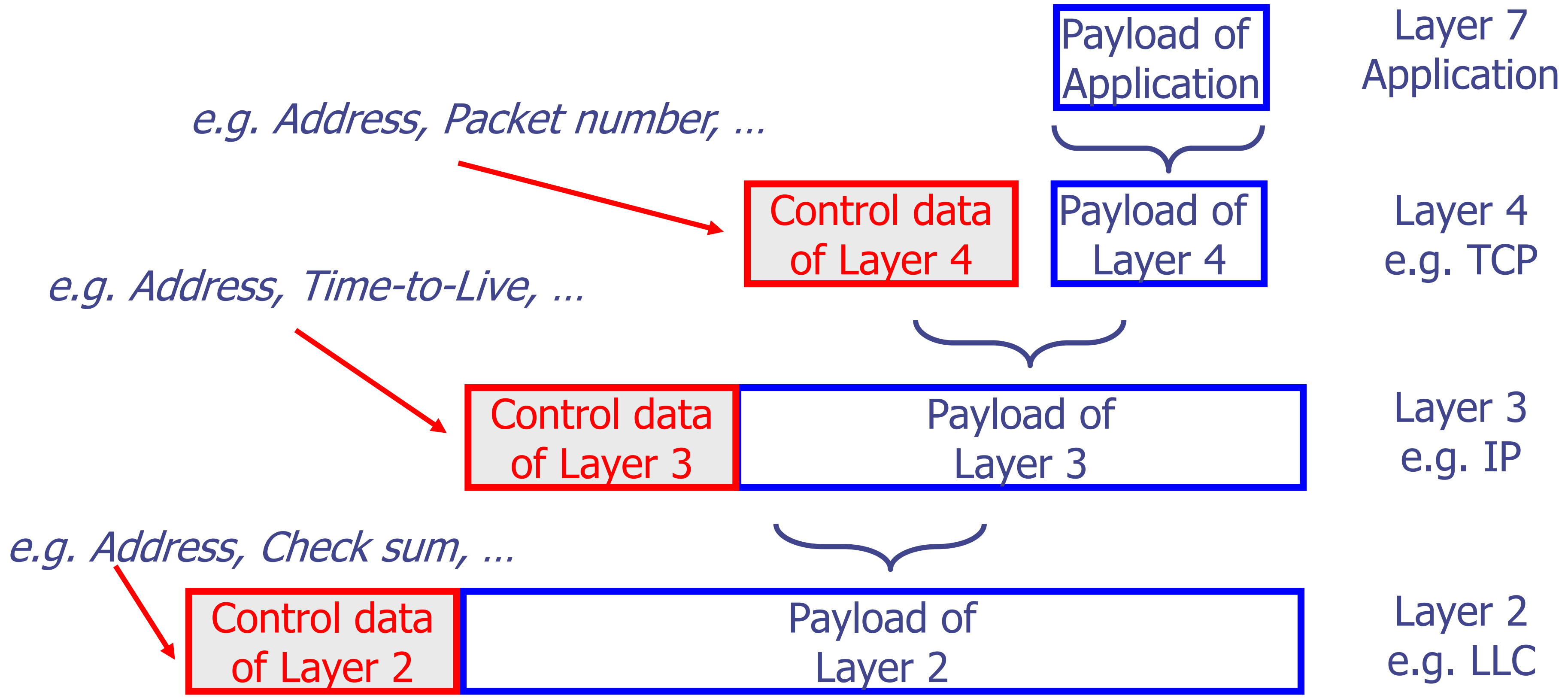
Protocols and messages

- When using lower-layer services to communicate with the remote peer, administrative data is usually included in those messages
- Typical example
  - Protocol receives data from higher layer
  - Adds own administrative data
  - Passes the extended message down to the lower layer
  - Receiver will receive original message plus administrative data
- Encapsulating
  - Header or trailer


Packet arriving at Layer n+1
Packet delivered to Layer n user
Layer n+1
Layer n+1
Delivered by layer n
Extended packet passed to layer n
Layer n

Embedding messages

- Messages from upper layers are used as payload for messages in lower layers



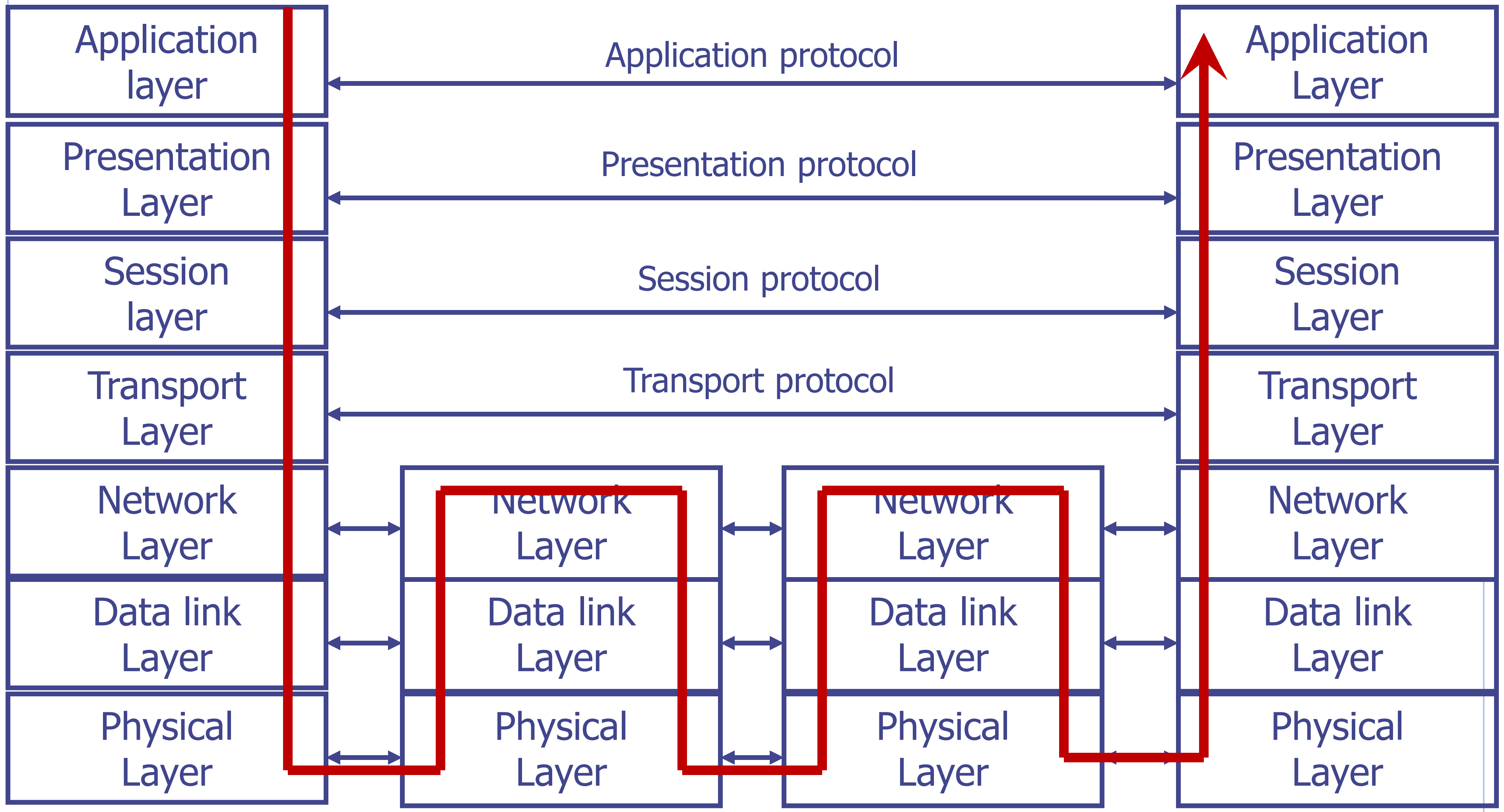
How to structure functions/layers ?

- Many functions have to be realized
- Not each function Is necessery in each Layer..
- How to actually assign them into layers to obtain a real, working communication system?
  - This is the role of a specific reference model
- Two main reference models exist
  - ISO/OSI reference model (International Standards Organization Open Systems Interconnection)
  - TCP/IP reference model (by IETF – Internet Engineering Taskforce)

ISO/OSI reference model

- Basic design principles
  - One layer per abstraction of the “set of duties”
  - Choose layer boundaries such that information flow across the boundary is minimized (minimize inter-layer interaction)
  - Enough layers to keep separate things separate, few enough to keep architecture manageable
- Result: 7-layer model
  - Not strictly speaking an architecture, because
  - Precise interfaces are not specified (nor protocol details !)
  - Only general duties of each layer are defined

ISO/OSI model



7 layers in brief

- Physical layer: Transmit raw bits over a physical medium
- Data Link layer: Provide a (more or less) error-free transmission service for data frames - also over a shared medium!
- Network layer: Solve the forwarding and routing problem for a network- bring data to a **desired host**
- **Transport layer**: Provide (possibly reliable, in order) end-to-end communication, overload protection, fragmentation to **processes**

  "Bringing data from process A to B with sufficient quality"
- Session layer: Group communication into sessions which can be synchronized, checkpointed, …
- Presentation layer: Ensure that syntax and semantic of data is uniform between all types of terminals
- Application layer: Actual application, e.g., protocols to transport Web pages

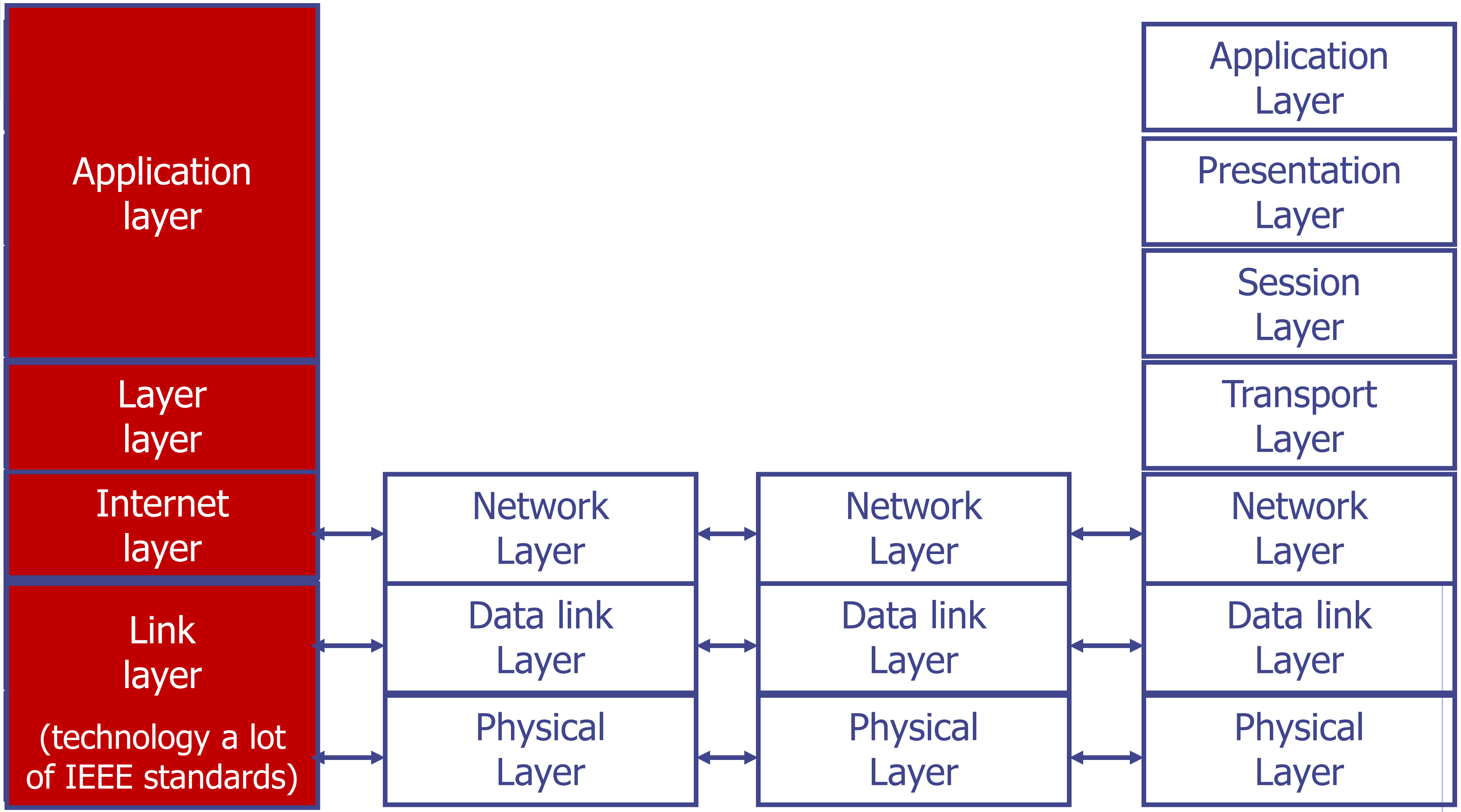
TCP/IP reference model

- Historically based on ARPANET, evolving to the Internet
  - Started out as little university networks, which had to be interconnected
- Some generic rules & principles
  - Internet connects **networks**
    - Minimum functionality assumed (just unreliable packet delivery!)
    - Internet layer (IP): packet switching, addressing, routing & forwarding

      ➔ ***Internet over everything***
  - End-to-end
    - Any functionality should be pushed to the instance needing it!
  - Fate sharing
- In effect only two layers really defined… Internet and Transport Layer - Lower and higher layers not really defined

  ➔ *Anything over internet*
- New Applications do NOT need any changes in the NETWORK!
  - Compare with the telephone network!!!

TCP/IP model (in red)



Some example protocols

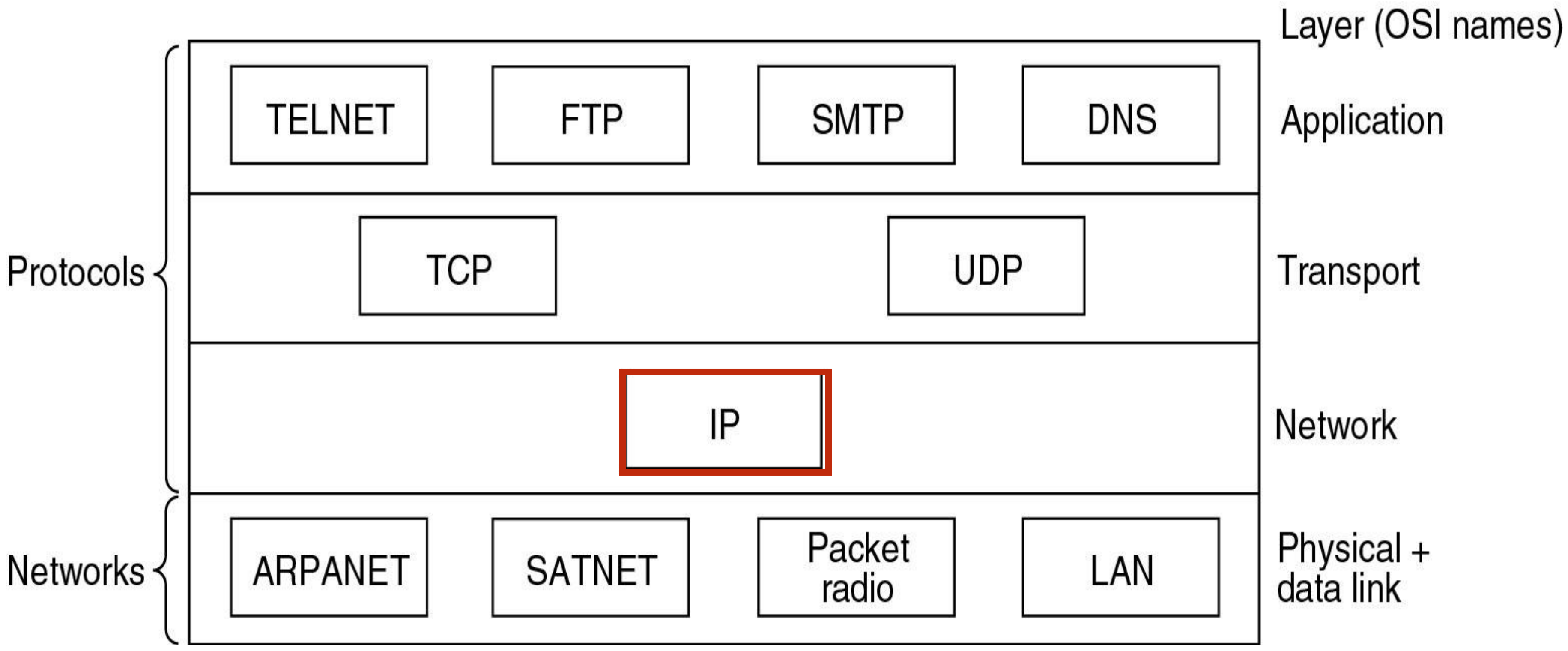
- A communication architectures needs standard protocols in addition to a layering structure
- And: some generic rules & principles which are not really a protocol but needed nonetheless
  - Example principle: end-to-end
  - Example rule: Naming & addressing scheme
- Popular protocols of the 5-layer reference model
  - Data link layer: Ethernet & CSMA/CD
  - Network layer: Internet Protocol (IP)
  - Transport layer: Transmission Control Protocol (TCP)

Standardization

- To build large networks, standardization is necessary
- Traditional organization from a telecommunication/ telephony background
  - Well established, world-wide, relatively slow “time to market”
- Internet
  - Mostly centered around the Internet Engineering Task Force (IETF) with associated bodies (Internet Architectural Board, Internet Research Task Force, Internet Engineering Steering Group)
  - Consensus oriented, focus on working implementations
  - Hope is quick time to market, but has slowed down considerably in recent years
- Manufacturer bodies

TCP/IP – Suite of protocols

- Over time, suite of protocols evolved around core TCP/IP protocols



So-called “**hourglass model**”: Thin waist of the protocol stack at IP, above the technological layers

Naming & addressing in the TPC/IP stack

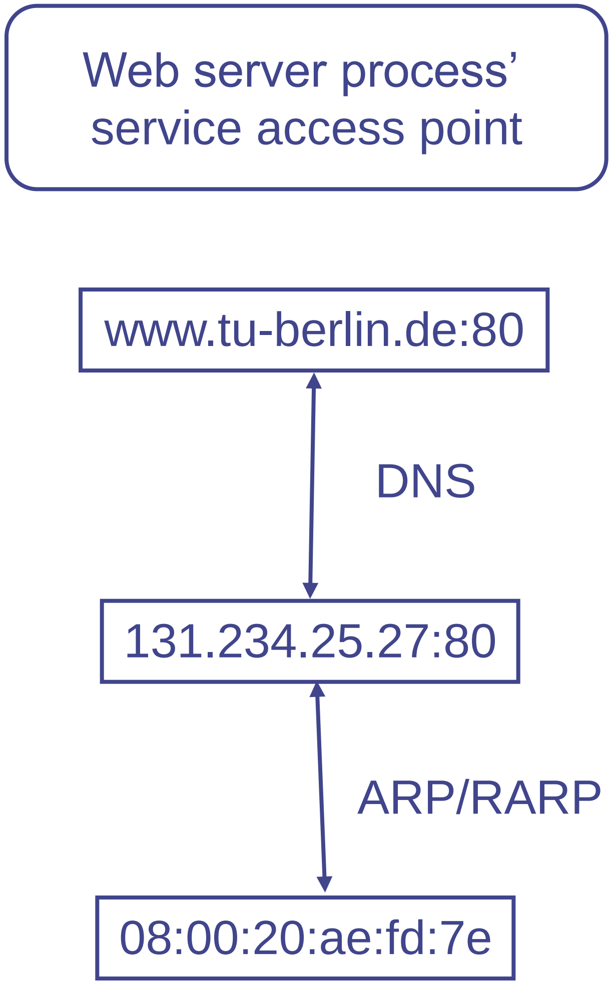
- Names: Data to identify an entity exist on different levels
  - Alphanumerical names for machines: www.tu-berlin.de

- Address: Data how/where to find an entity
  - Address of a network device in an IP network: An IP address
    - IPv4: 32 bits, structured into 4x8 bits
    - Example: 131.234.20.99 (dotted decimal notation)
  - Address of a network: Some of the initial bits of an IP address

- Address of a networked device in the LOCAL AREA (IEEE 802 standardized) network...
  - 48 bits, hexadecimal notation, example: 08:00:20:ae:fd:7e

Mapping

- Needed: Mapping from name to address
  ➔ Realized by separate protocols

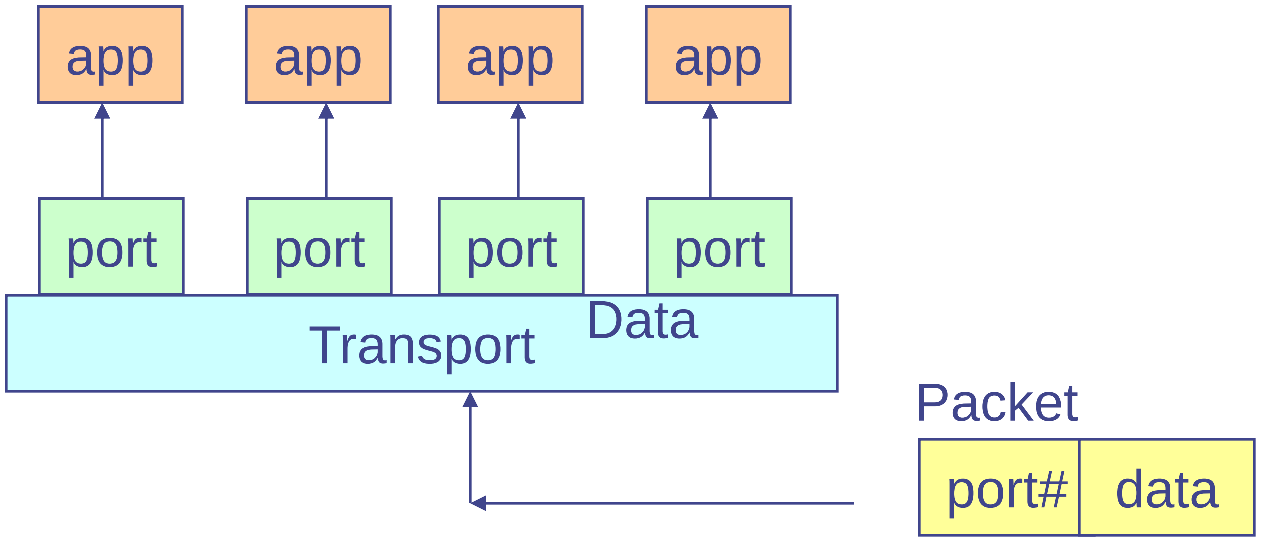


  - From alphanumerical name to IP address: Domain Name System (DNS)
  - Often also useful: Mapping from IP address to MAC name/address: Address Resolution Protocol (ARP)

Understanding Ports

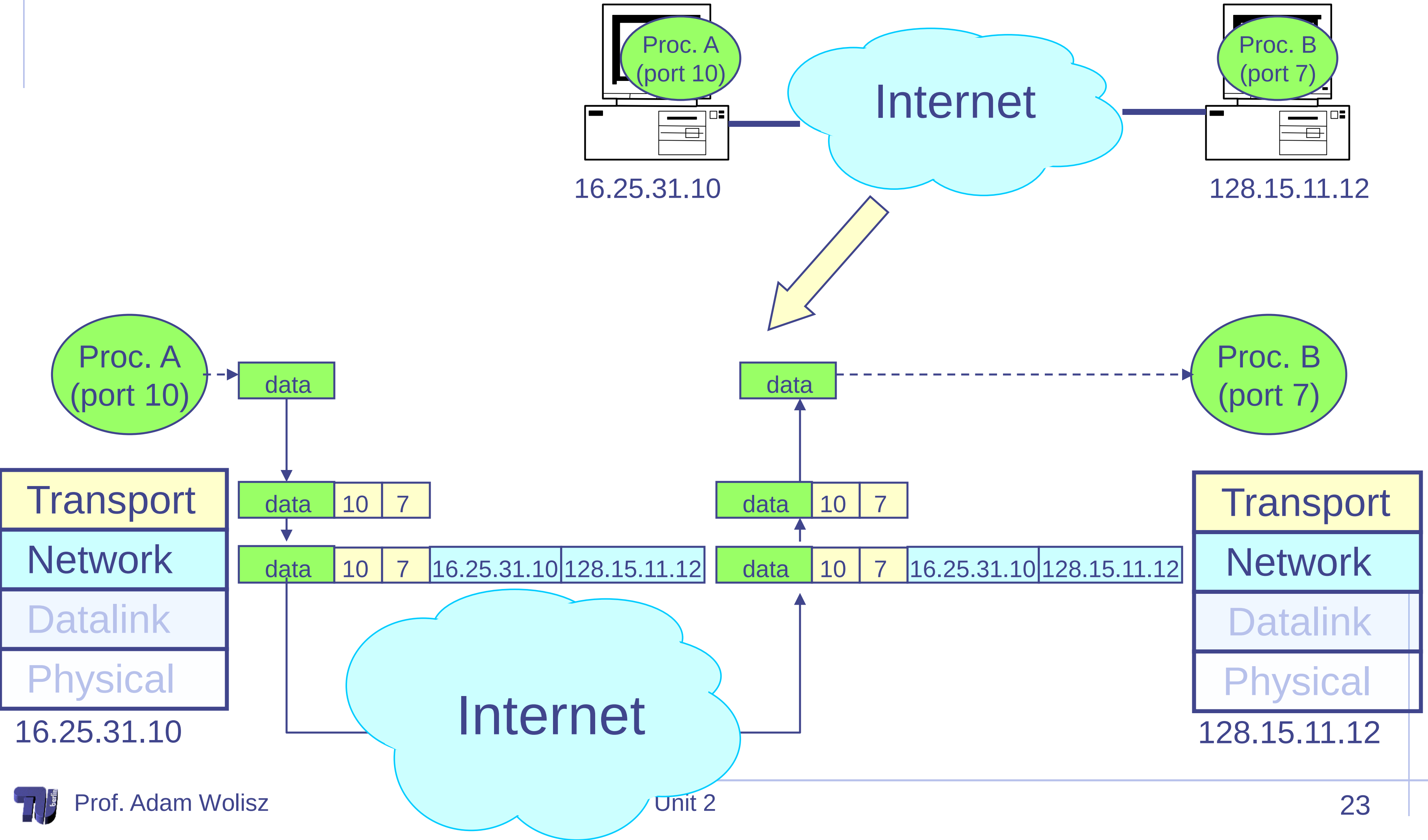
[Buya]

- Port is represented by a positive (16-bit) integer value
- Some ports have been reserved to support common/well known services: http 80/tcp; ftp 21/tcp; telnet 23/tcp; smtp 25/tcp;
- User level process/services generally use port number value >= 1024



End-to-End Layering View

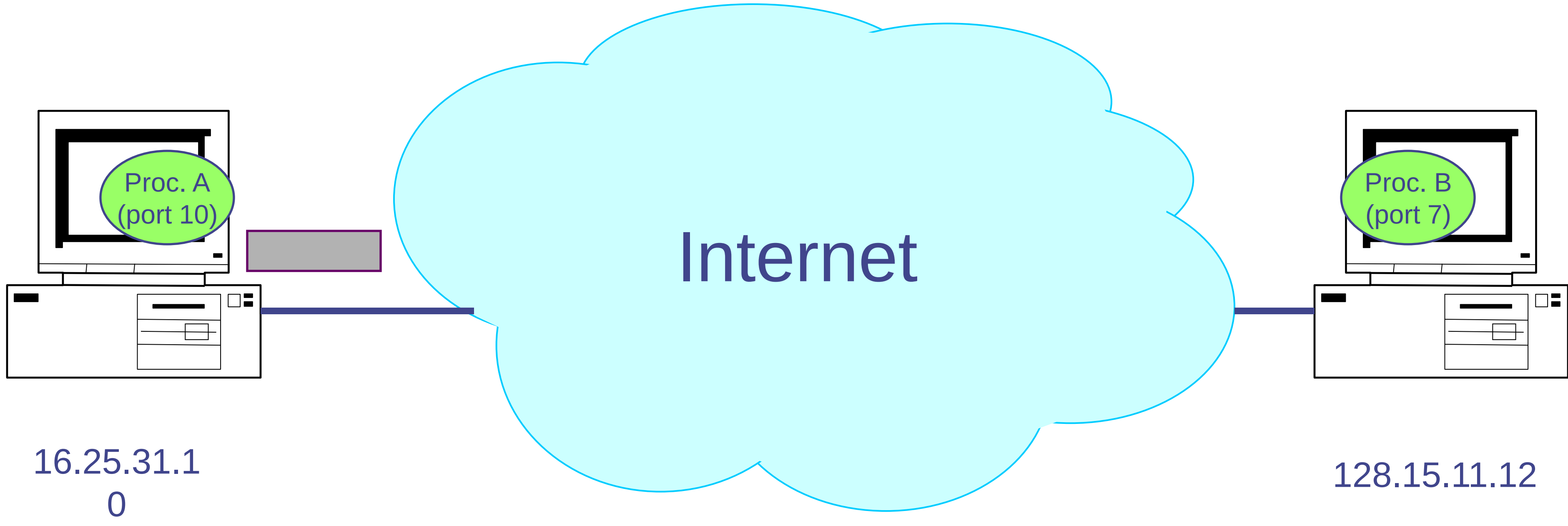
Stoica



Internet End-to-End View

[Stoica]

- Process A sends a packet to process B

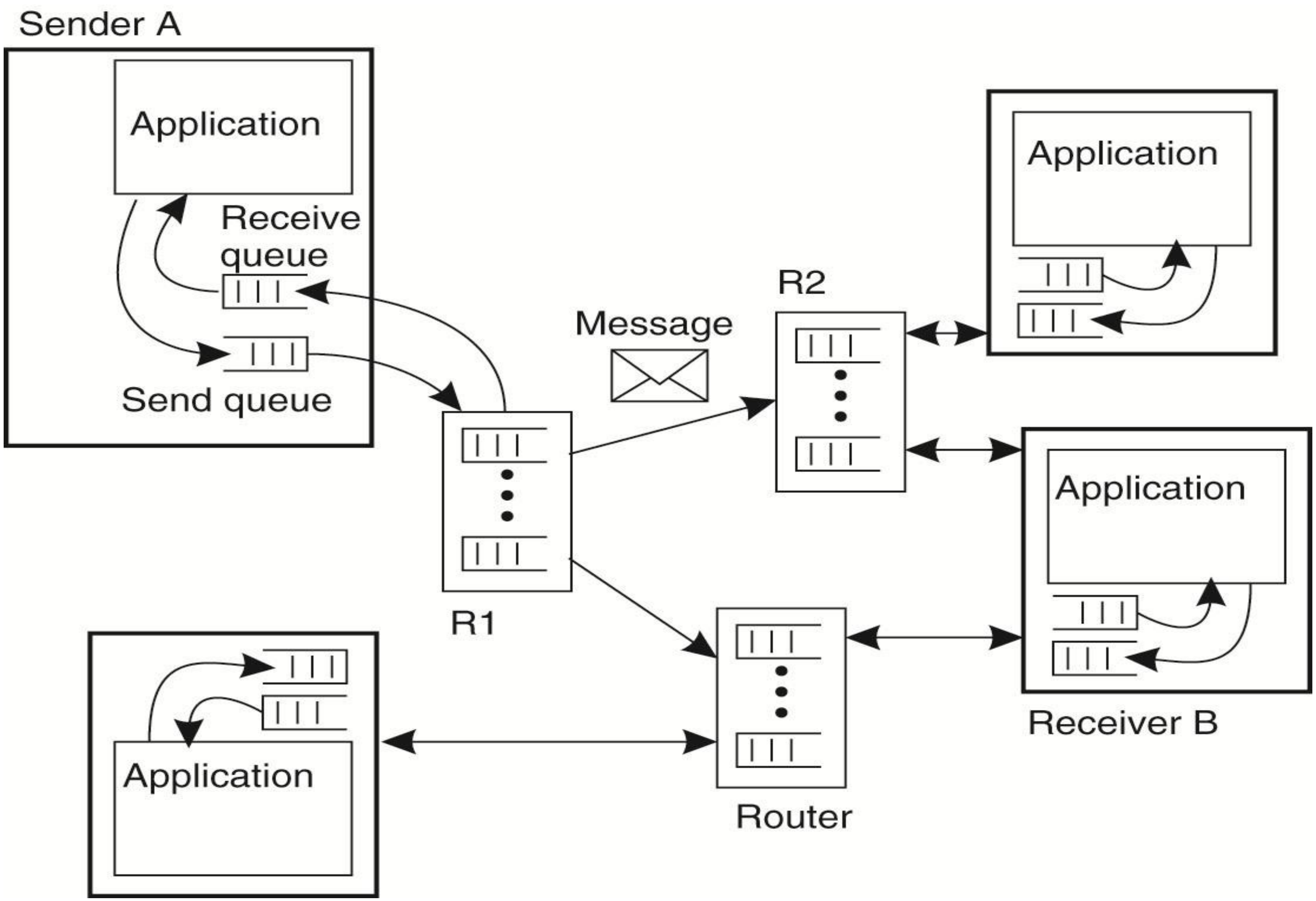


IP Address:

A four-part "number" used by *Network Layer* to route a packet from one computer to another

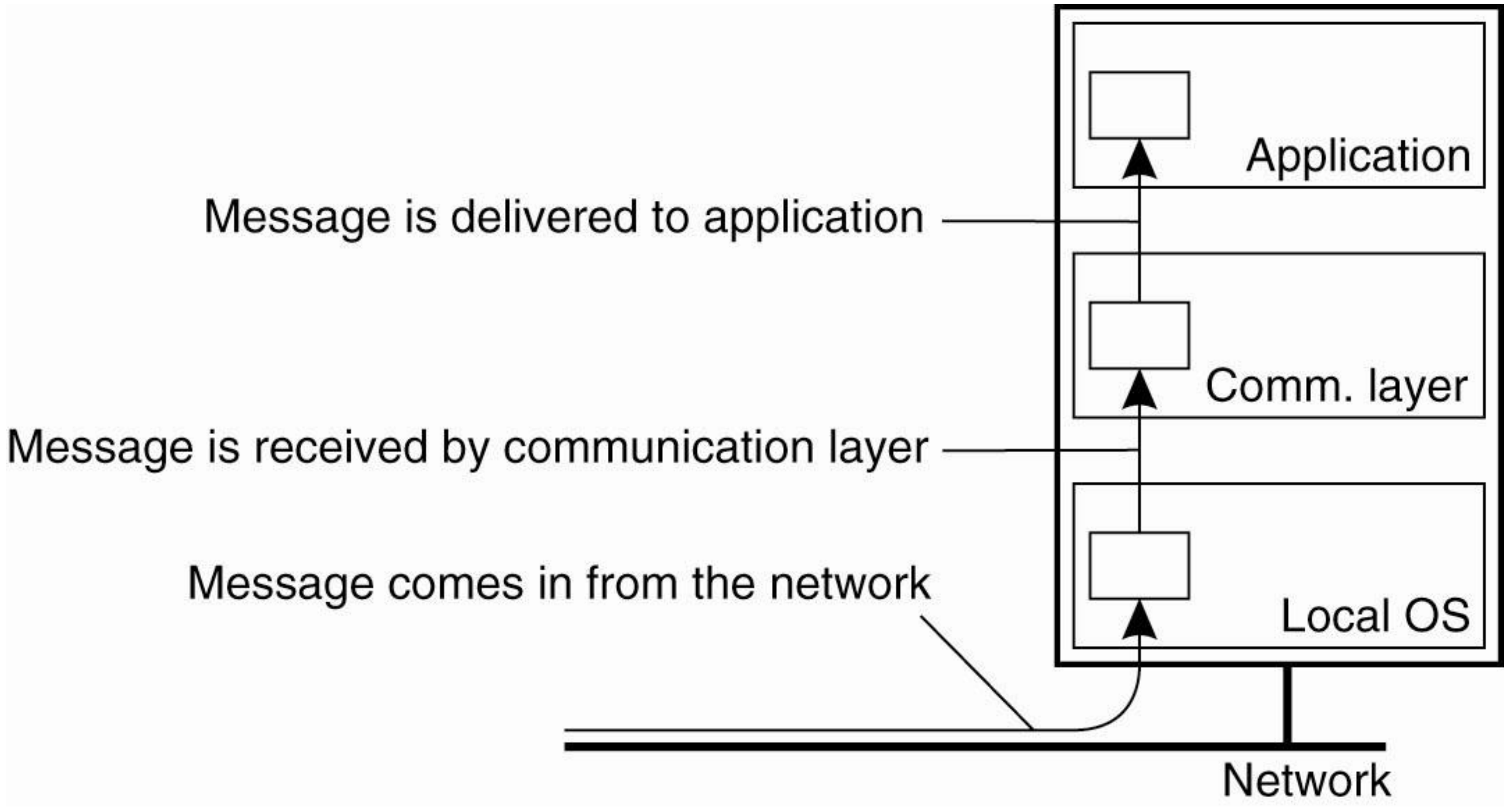
General Architecture of a Message System

[Tanenbaum]



Message Receipt vs. Message Delivery [Tanenbaum]

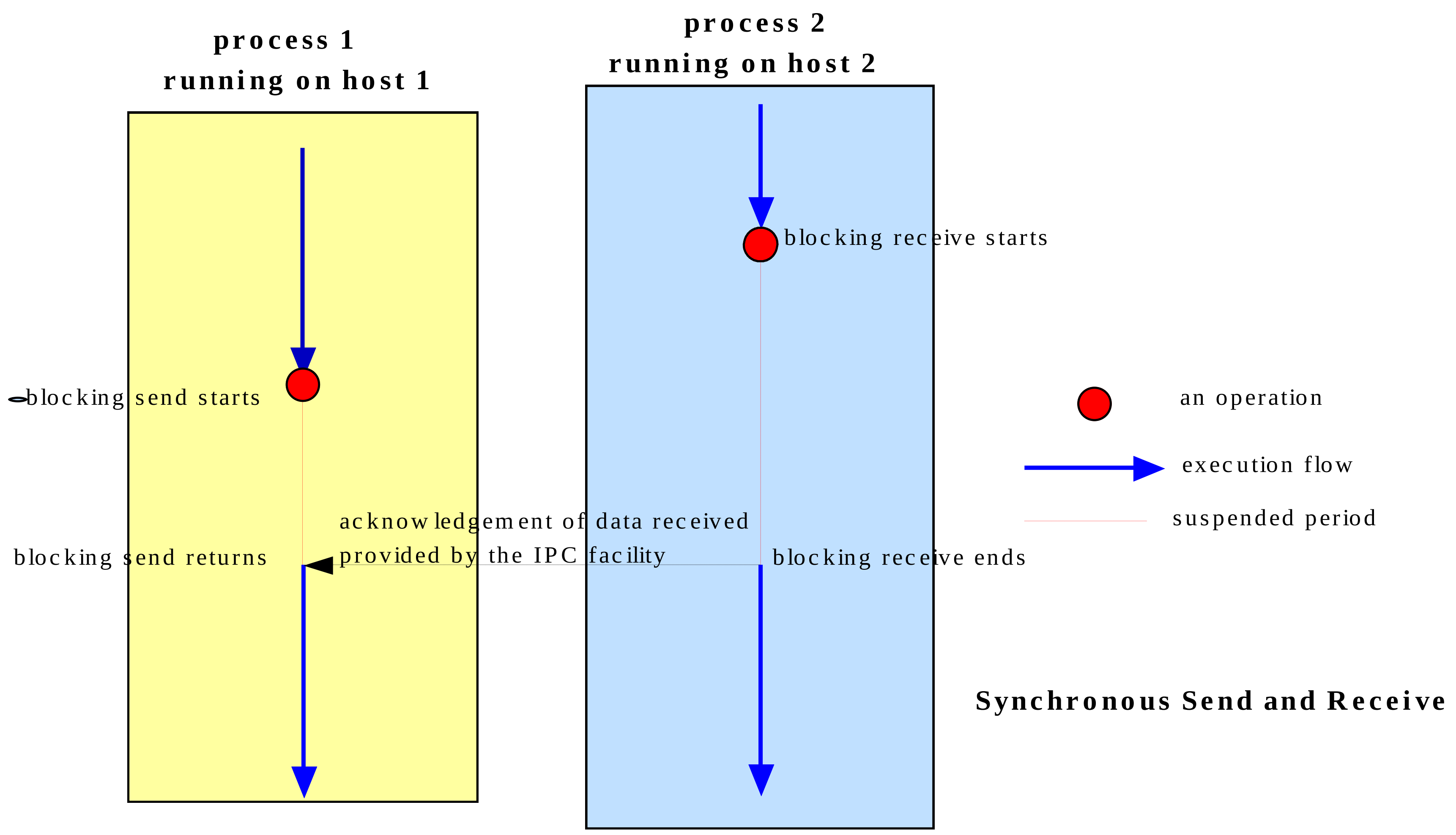
- Figure 8-12. The logical organization of a distributed system to distinguish between message receipt and message delivery.


Application
Message is delivered to application
Comm. layer
Message is received by communication layer
Message comes in from the network
Local OS
Network

Synchronous Interaction

- Blocking send
  - Blocks until message is transmitted
  - Blocks until message acknowledged
- Blocking receive
  - Waits for message to be received
- Known upper/lower bounds on execution speeds, message transmission delays and clock drift rates
- More difficult to build, conceptually simpler model
  - use Queue (for waiting)
  - *send* and *receive* are blocking

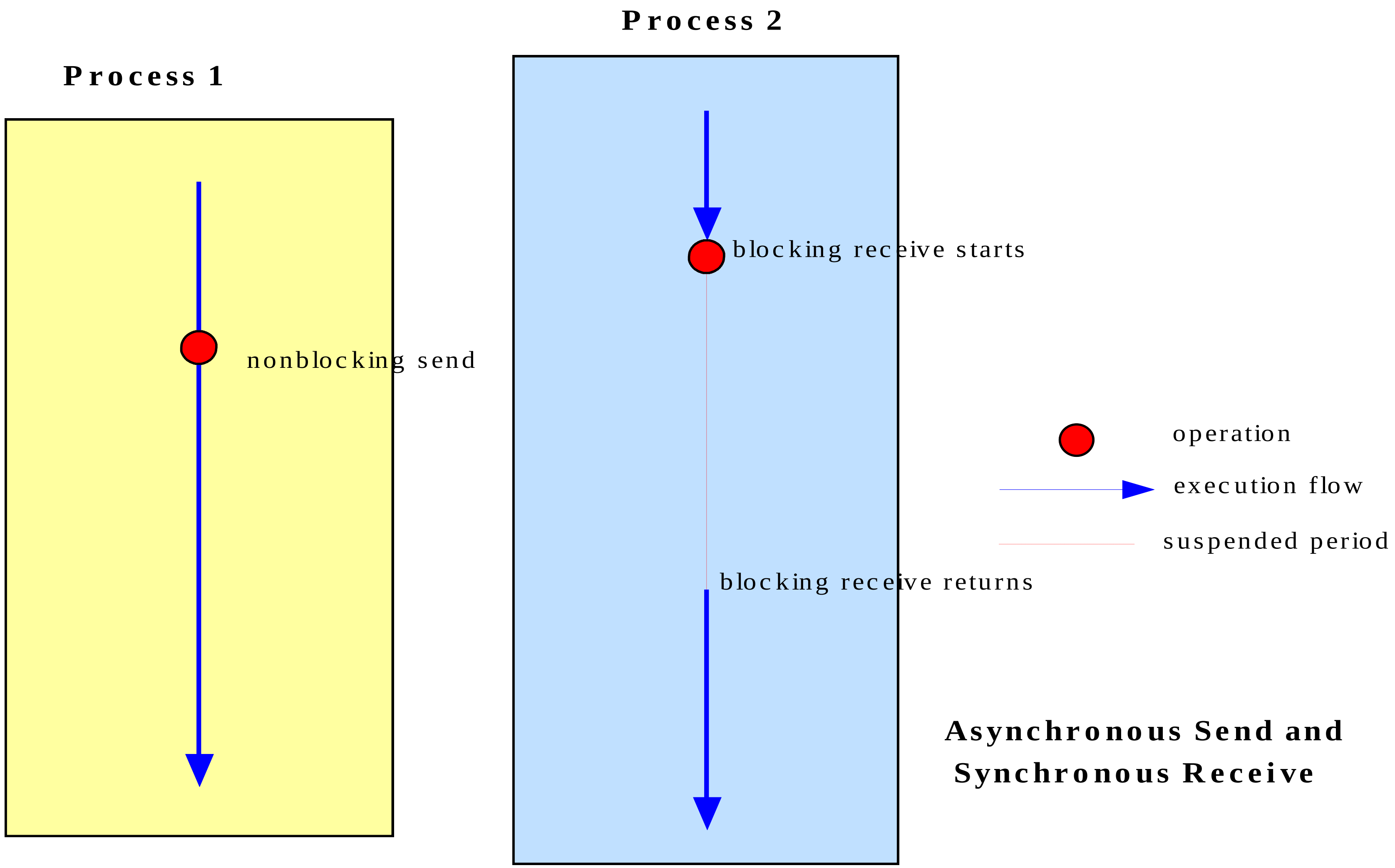
Synchronous send and receive



Asynchronous Interaction

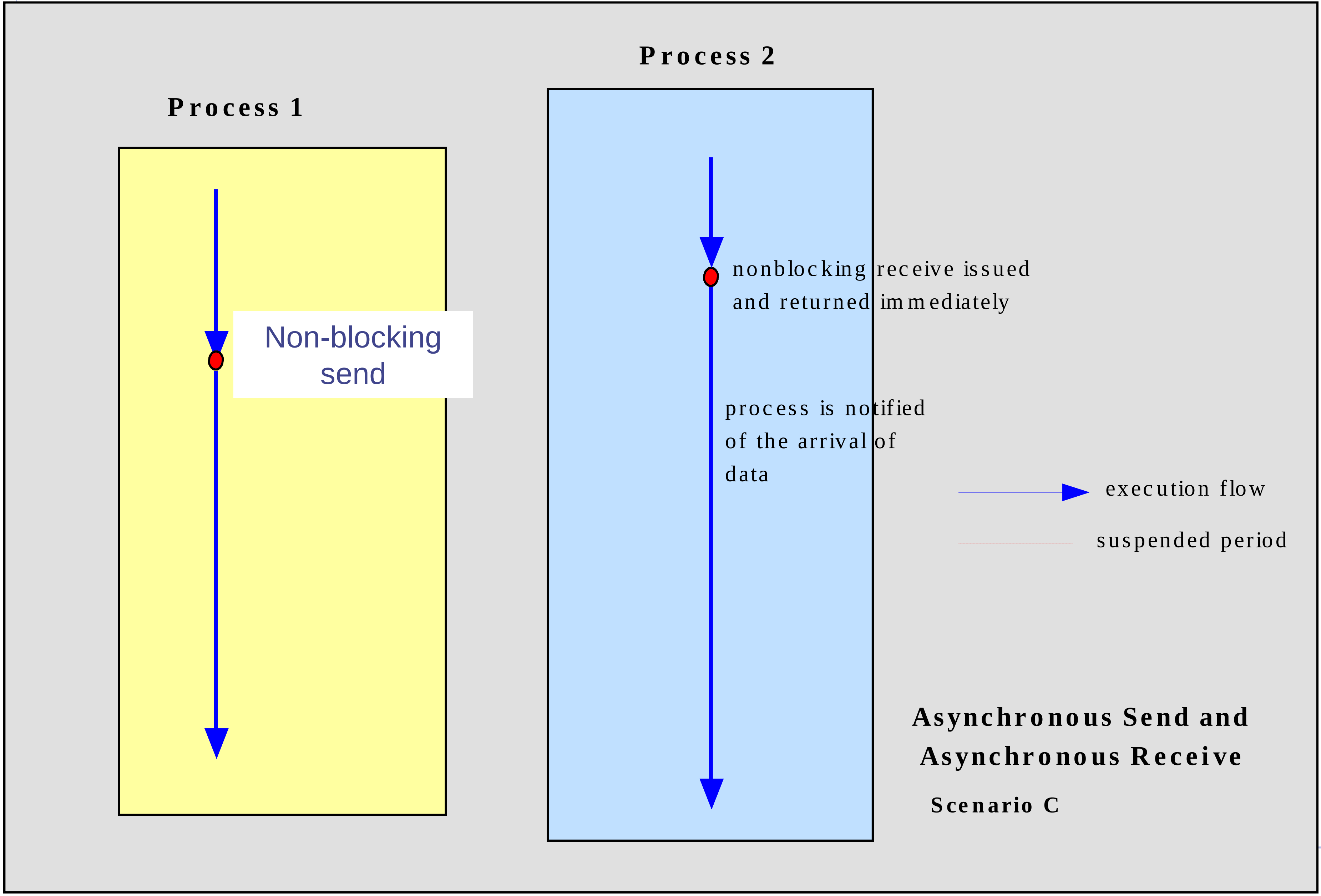
- Non-blocking send: sending process continues as soon message is queued.
- Blocking or non-blocking receive:
  - Blocking:
    - Timeout.
    - Threads.
  - Non-blocking: proceeds while waiting for message.
    - Message is queued upon arrival.
    - Process needs to poll or be interrupted.
- Arbitrary processes execution speeds, message transmission delays and clock drift rates
- Some problems impossible to solve (e.g. agreement)

Asynchronous send and synchronous receive



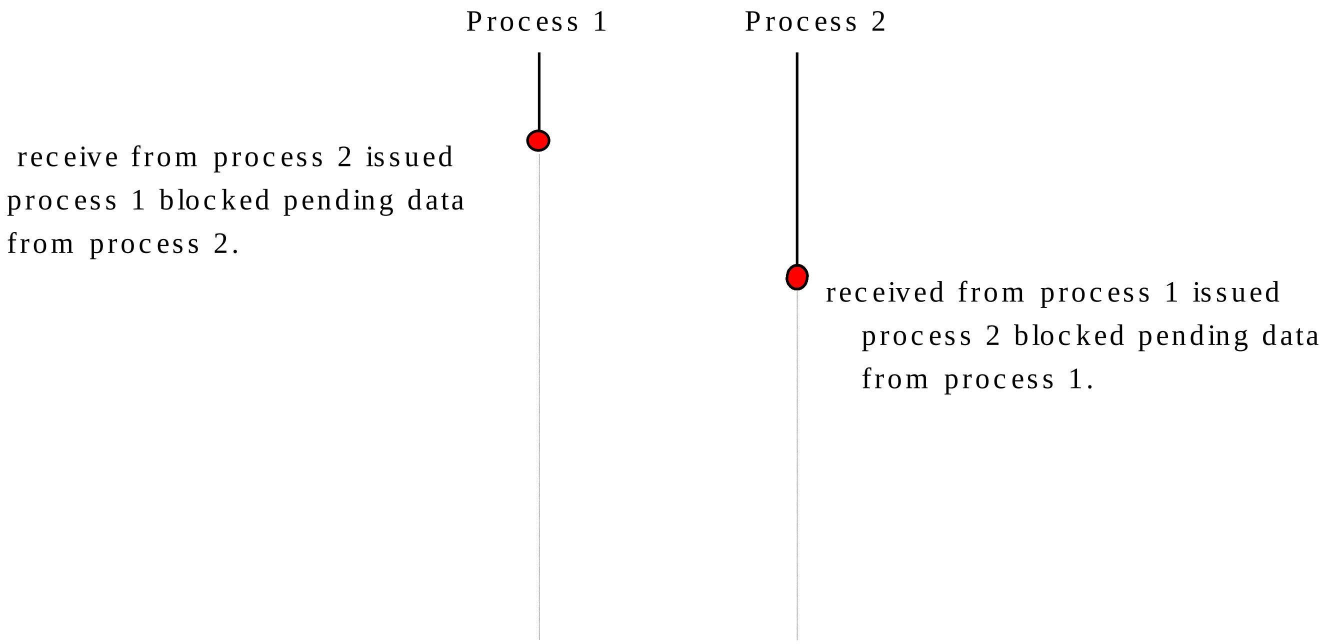
Asynchronous Send and Synchronous Receive

Asynchronous send and Asynchronous receive



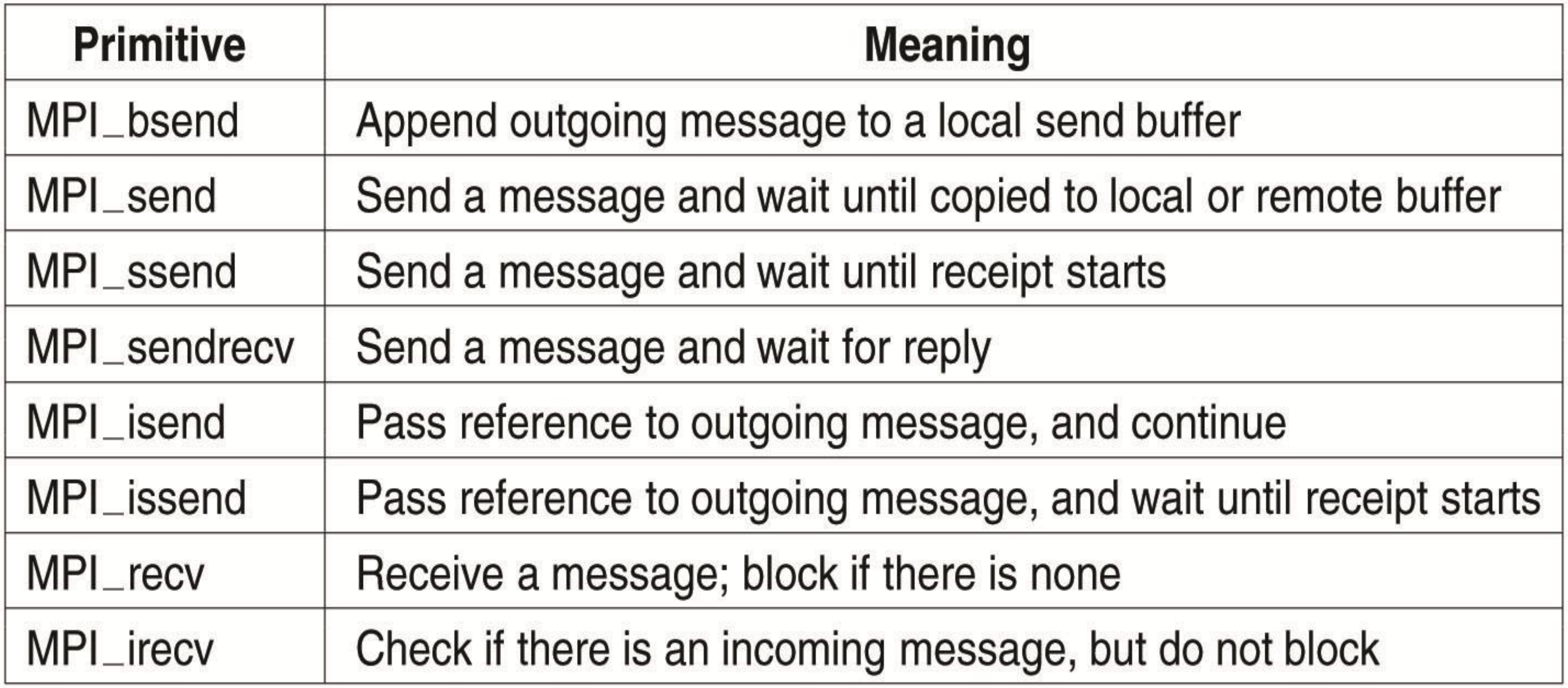
Blocking, deadlock, and timeouts

- **Blocking operations issued in the wrong sequence can cause *deadlocks*.**
- **Deadlocks should be avoided. Alternatively, *timeout* can be used to detect deadlocks.**


Process 1
Process 2
receive from process 2 issued
process 1 blocked pending data
from process 2.
received from process 1 issued
process 2 blocked pending data
from process 1.

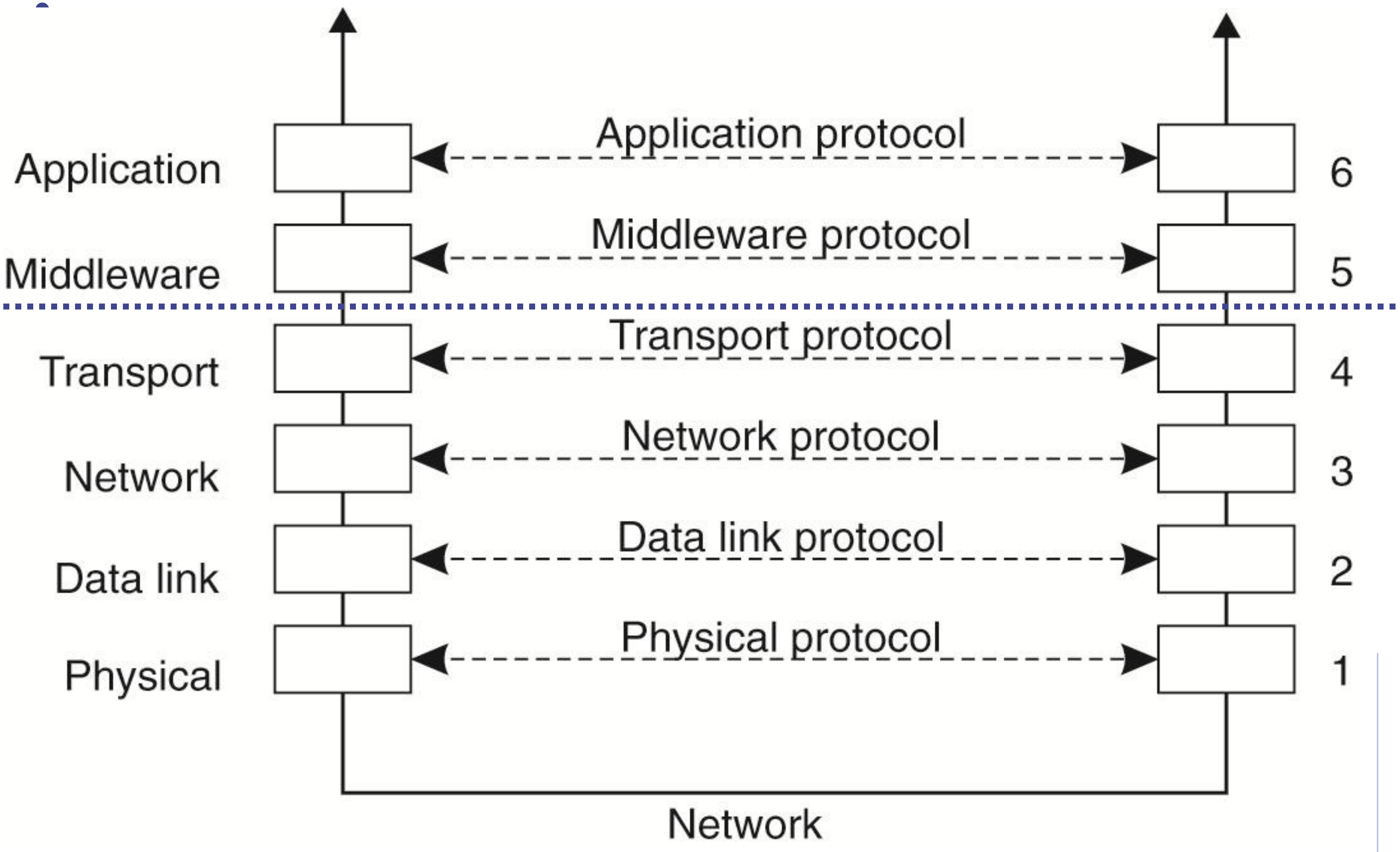
Intuitive message-passing primitives

[Tannenbaum]



Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

An adapted model communication model [Tannenbaum]

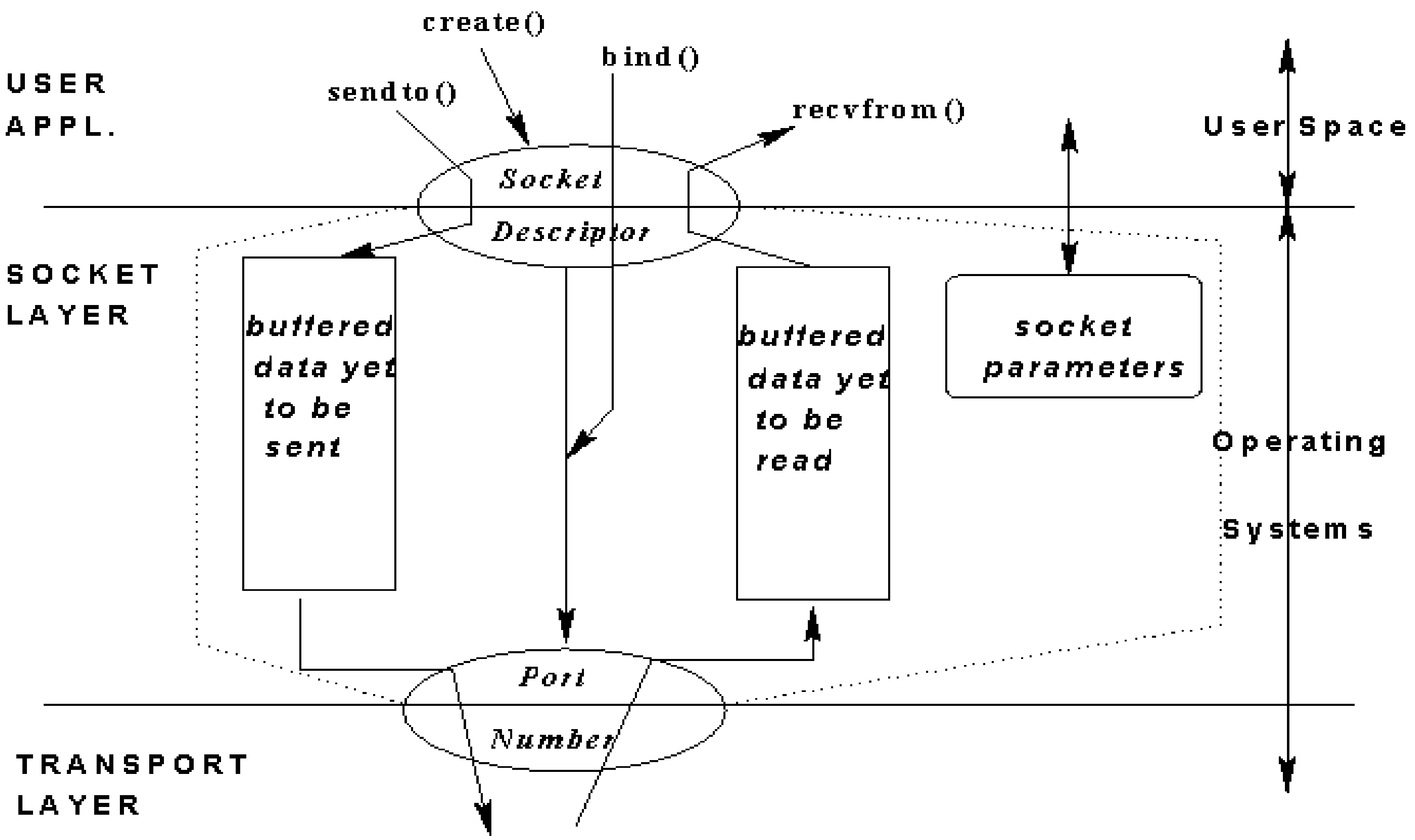


Sockets: the Transport Layer API

- Sockets provide an API (Application Programming Interface) for programming networks at the transport layer.
- A socket is an endpoint of a two-way communication link between two processes located on the same machine - or located on different machines connected by a network.
- Network communication using Sockets is similar to file I/O
  - Socket handle is treated like file handle.
  - The streams used in file I/O operation are also applicable to socket-based I/O
- Socket-based communication is programming language independent!
- The success of this API is based on its abstraction of all possible used network protocols/underlying network topology.
  - A socket is bound to a port number so that the transport protocol can identify the application that data destined to be sent.

Socket Layer

[wonkwang]



Connection-oriented vs. connection-less service [Karl]

- Recall telephony vs. postal service
  - Service can require a preliminary setup phase, e.g., to determine receiver ! ***connection-oriented service***
    - Three phases: connect, data exchange, release connection
  - Alternative: Invocation of a service primitive can happen at any time, with all necessary information provided in the invocation ! ***connection-less service***
  - Note: This distinction does ***NOT*** depend on circuit or packet switching – connection-oriented services can be implemented on top of packet switching (and vice versa, even though a bit awkward)
- Connection-oriented services must provide primitives to handle connection
  - CONNECT – setup a connection to the communication partner
  - LISTEN – wait for incoming connection requests
  - INCOMING_CONN – indicate an incoming connection request
  - ACCEPT – accept a connection
  - DISCONNECT – terminate a connection

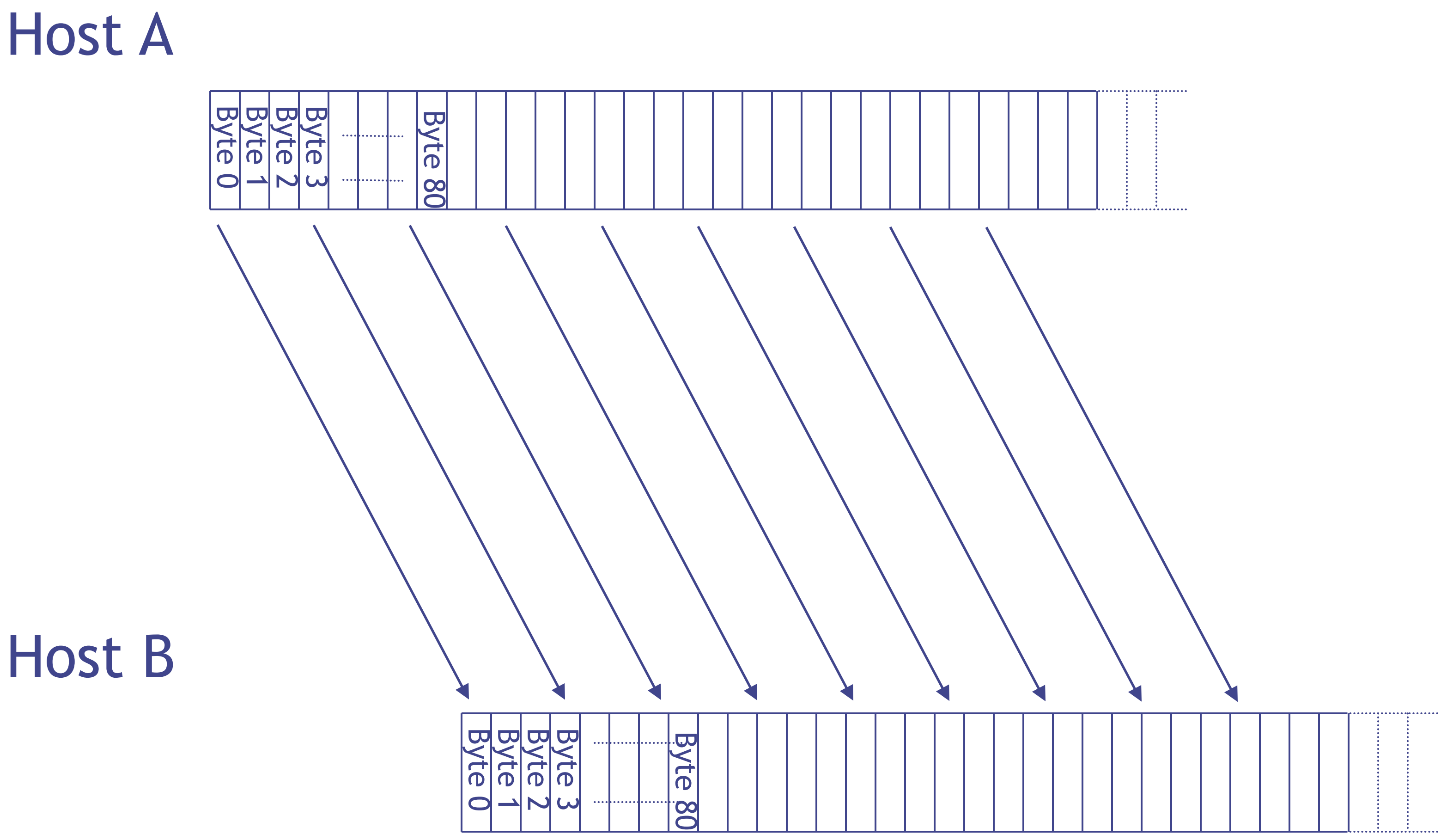
Typical examples of services

- ***Datagram service***
  - Unit of data are messages (limited length)
  - Correct, but not necessarily complete or in order –
  - Connection-less (*e.g. supported by UDP*)
  - Usually insecure/not dependable, not confirmed

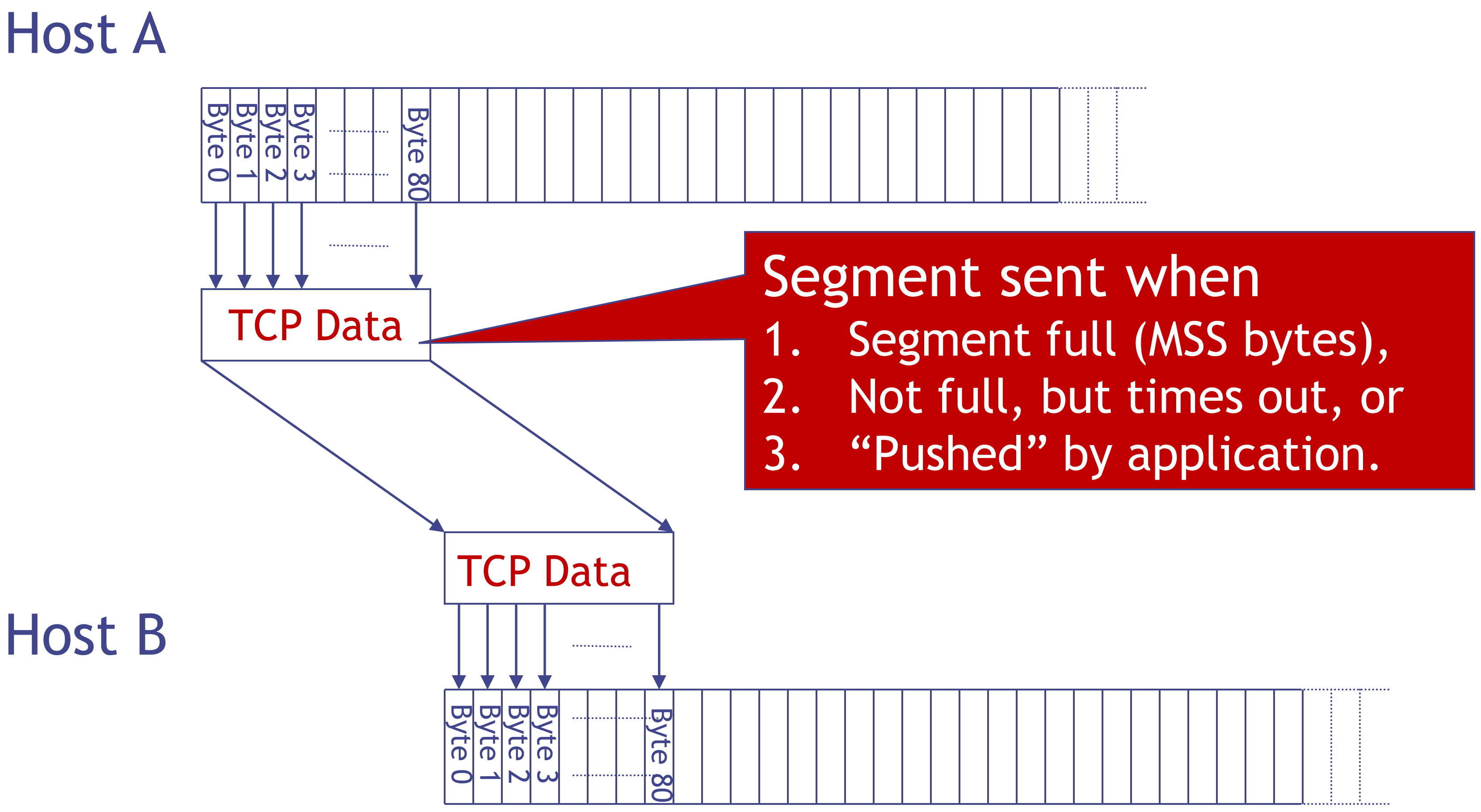
    Application must provide its own reliability“

- ***Reliable byte stream***
  - Byte stream
  - Correct, complete, in order, confirmed - Processes have some guarantee that messages will be delivered.
    - Possible to build reliability atop unreliable service (E2E argument).
  - Sometimes, but not always secure/dependable
  - Connection-oriented (*e.g. Supported by TCP*)

TCP supports a "stream of bytes" service



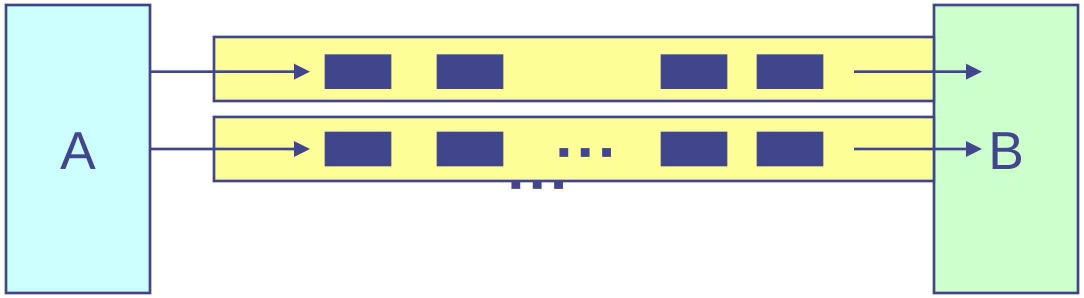
…which is emulated using TCP "segments"



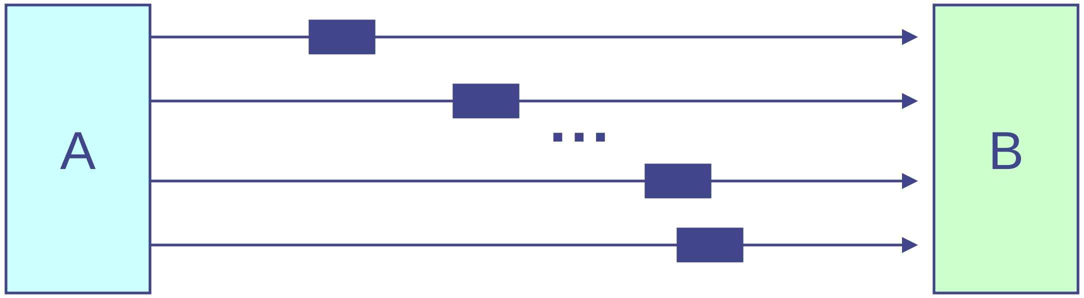
Segment sent when
1. Segment full (MSS bytes),
2. Not full, but times out, or
3. "Pushed" by application.

TCP vs. UDP Communication

[Buya]

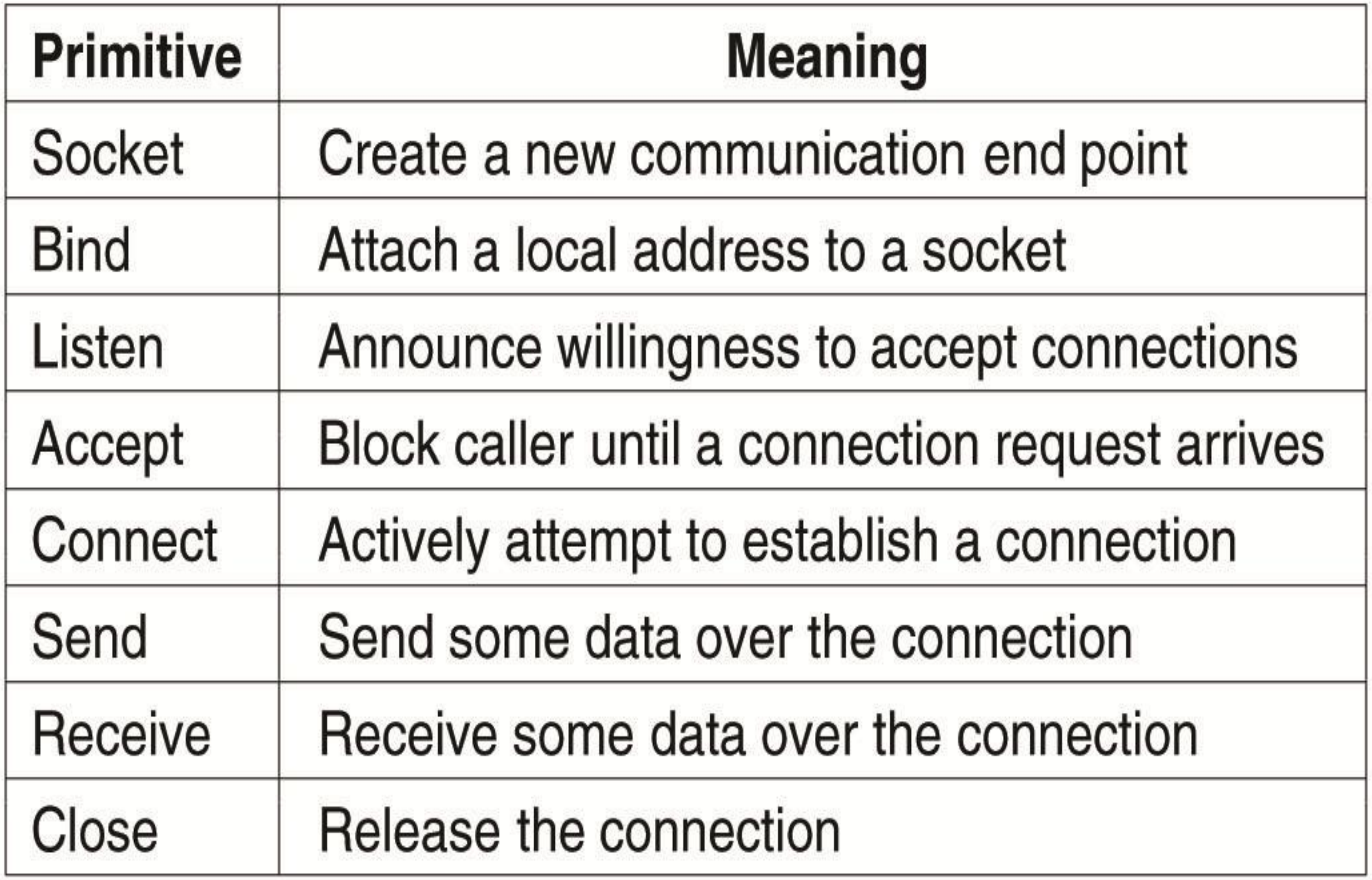


- Connection-Oriented Communication



- Connectionless Communication

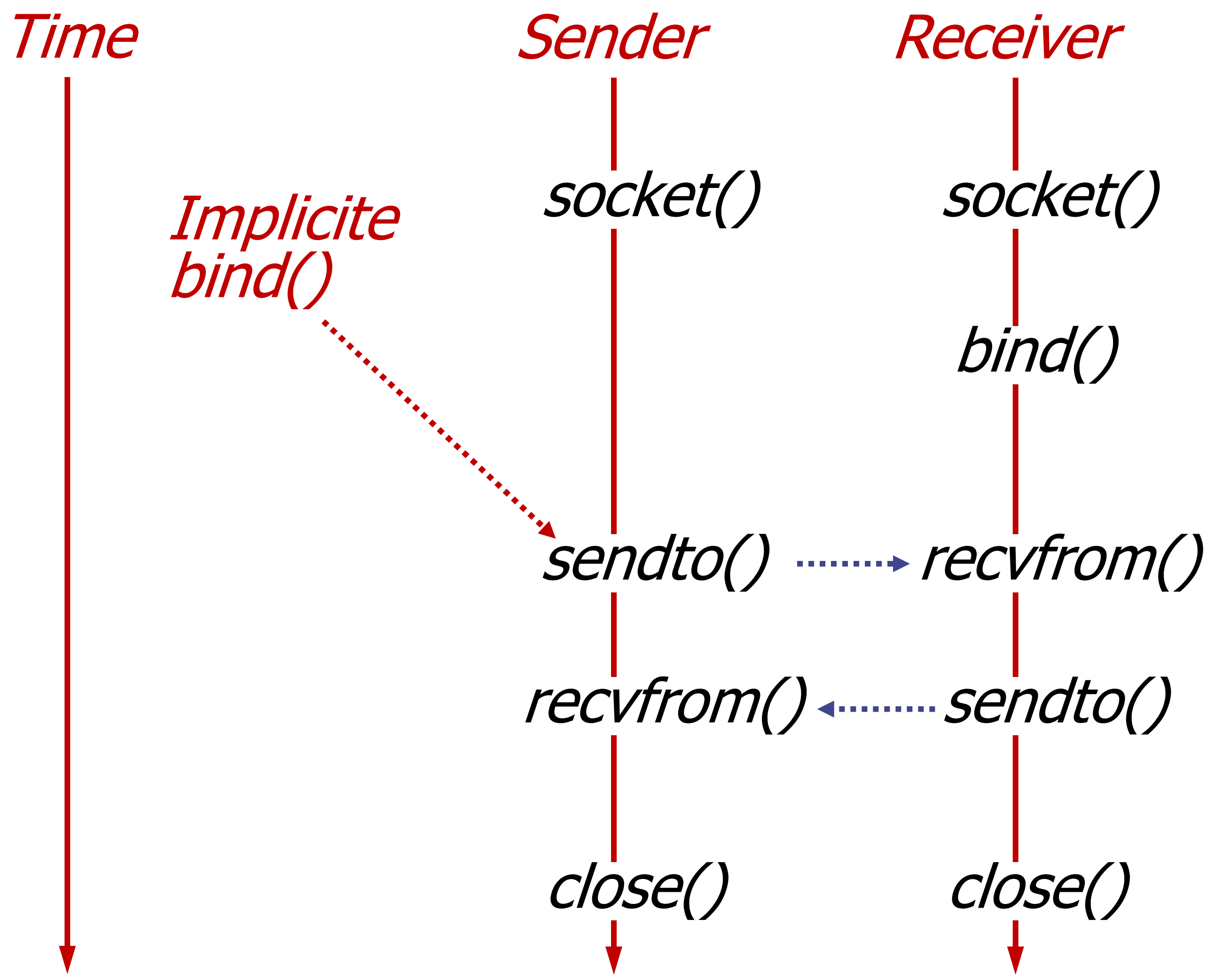
Berkeley Sockets



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

A VERY GOOD source of information about sockets is the Beej´s Guide
http://beej.us/guide/bgnet/ (legal free download!)

Datagram sockets (see MPGI 4)



Datagram communication over sockets

[Karl]

- Simplest possible service: unreliable datagrams

Sender

- `int s = socket (…);`
-
```
sendto (s,
      buffer,
      datasize,
      0,
      to_addr,
      addr_length);
```
- `to_addr` and `addr_length` specify the destination

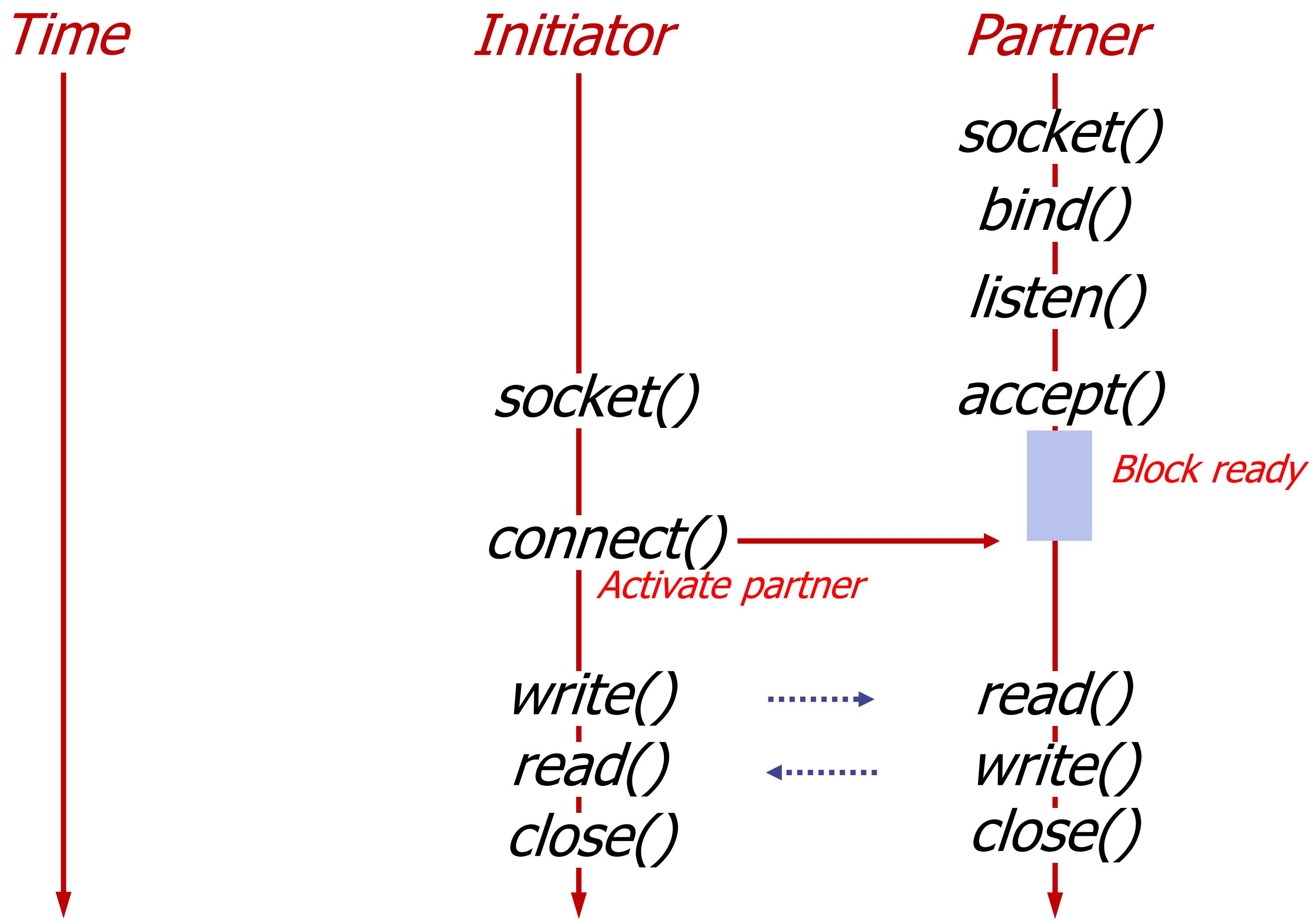
Receiver

- `int s = socket (…);`
- `bind (s, local_addr, …);`
-
```
recv (s,
      buffer,
      max_buff_length,
      0);
```
- Will wait until data is available on socket `s` and put the data into `buffer`

Java API for Datagram Comms [wonkwang]

- Class *DatagramSocket*
  - *socket constructor* (returns free port if no arg)
  - *send DatagramPacket*, non-blocking
  - *receive DatagramPacket*, blocking
  - *setSoTimeout* (receive blocks for time *T* and throws *InterruptedIOException*)
  - *connect*
  - *close DatagramSocket*
  - throws *SocketException* if port unknown or in use

Stream sockets



Byte streams over a connection-oriented socket [Karl]

- For reliable byte streams, sockets have to be connected first
- Receiver has to accept connection

Initiator *(client)*

- `int s =` **`socket`** `(…);`
- **`connect`** `(s, destination_addr, addr_length);`
- **`send`** `(s,buffer, datasize, 0);`
- Arbitrary `recv()/send()`
- **`close`** `(s);`

- Connected sockets use a `send` without address information

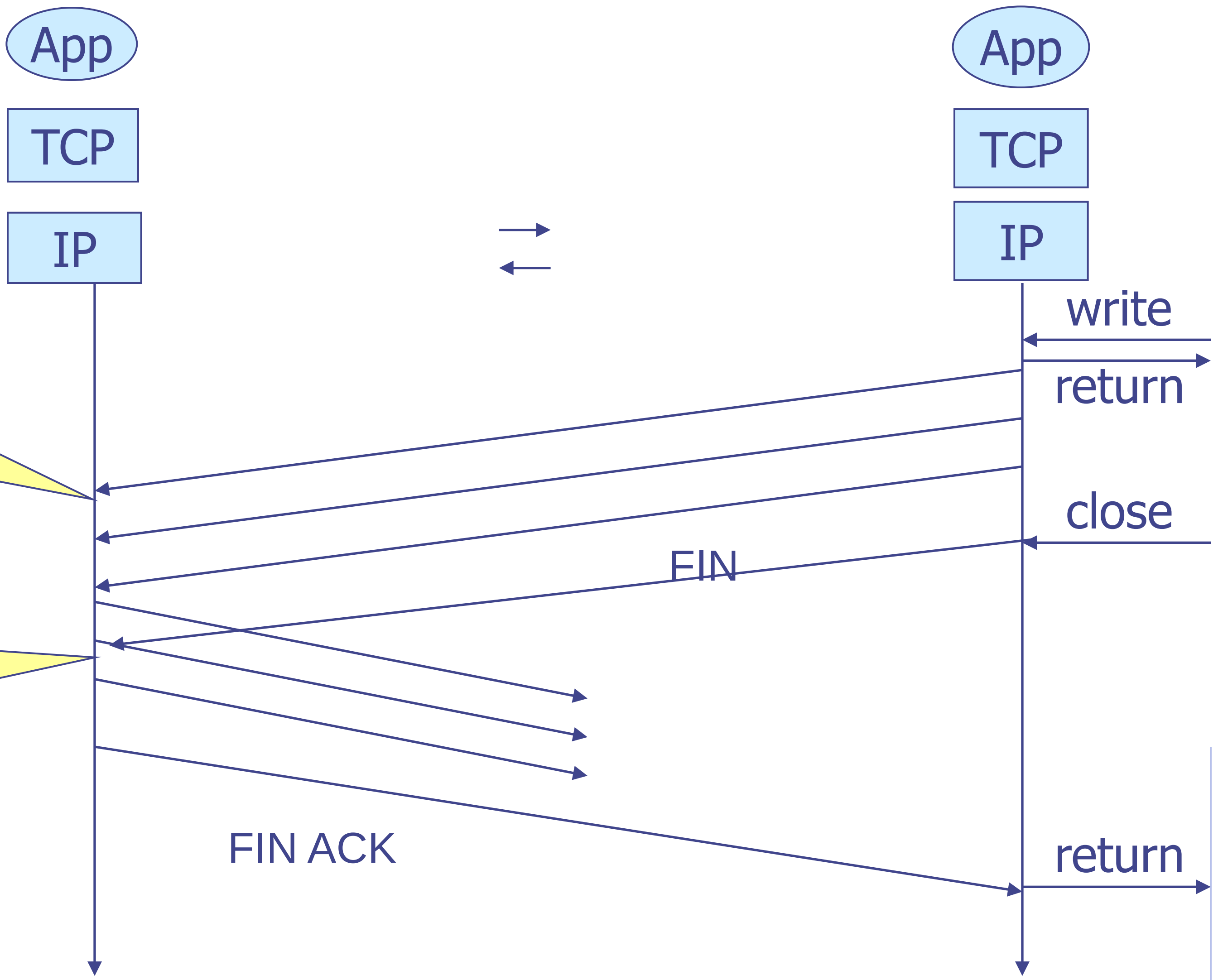
Partner *(server)*

- `int s =` **`socket`** `(…);`
- **`bind`** `(s, local_addr, …);`
- **`listen`** `(s, …);`
- `int newsock =` **`accept`** `(s, *remote_addr, …);`
- **`recv`** `(newsock, buffer, max_buff_length, 0);`
- Arbitrary `recv()/send()`
- **`close`** `(newsock);`

Java API for Data Stream Communications

- Class *ServerSocket*
  - *socket constructor* (for listening at a server port)
  - *getInputStream, getOutputStream*
  - *DataInputStream, DataOutputStream*
    (automatic marshaling/unmarshaling)
  - *close* to close a socket
  - raises *UnknownHost, IOException*, etc

Reliability of sockets



TCP data packets are acknowledged according to the receivers´s policy

The FIN is not acknowledged until all data are received by the end-system

Service primitives – correctness requirements [Karl]

- For both message and byte stream oriented (set of) service primitives, correctness requirements are important
  - ***Completeness***: All data that is sent is eventually delivered
  - ***Correctness***: If data is delivered, its is correct, i.e., the data that has been actually sent
    - Messages are not modified, original version is delivered
    - Byte sequence is free of errors
  - ***In order***: Byte sequence/sequence of messages is delivered in the order it has been sent
  - ***Dependable***: secure, available, …
  - ***Confirmed***: Reception of data is acknowledged to the sender

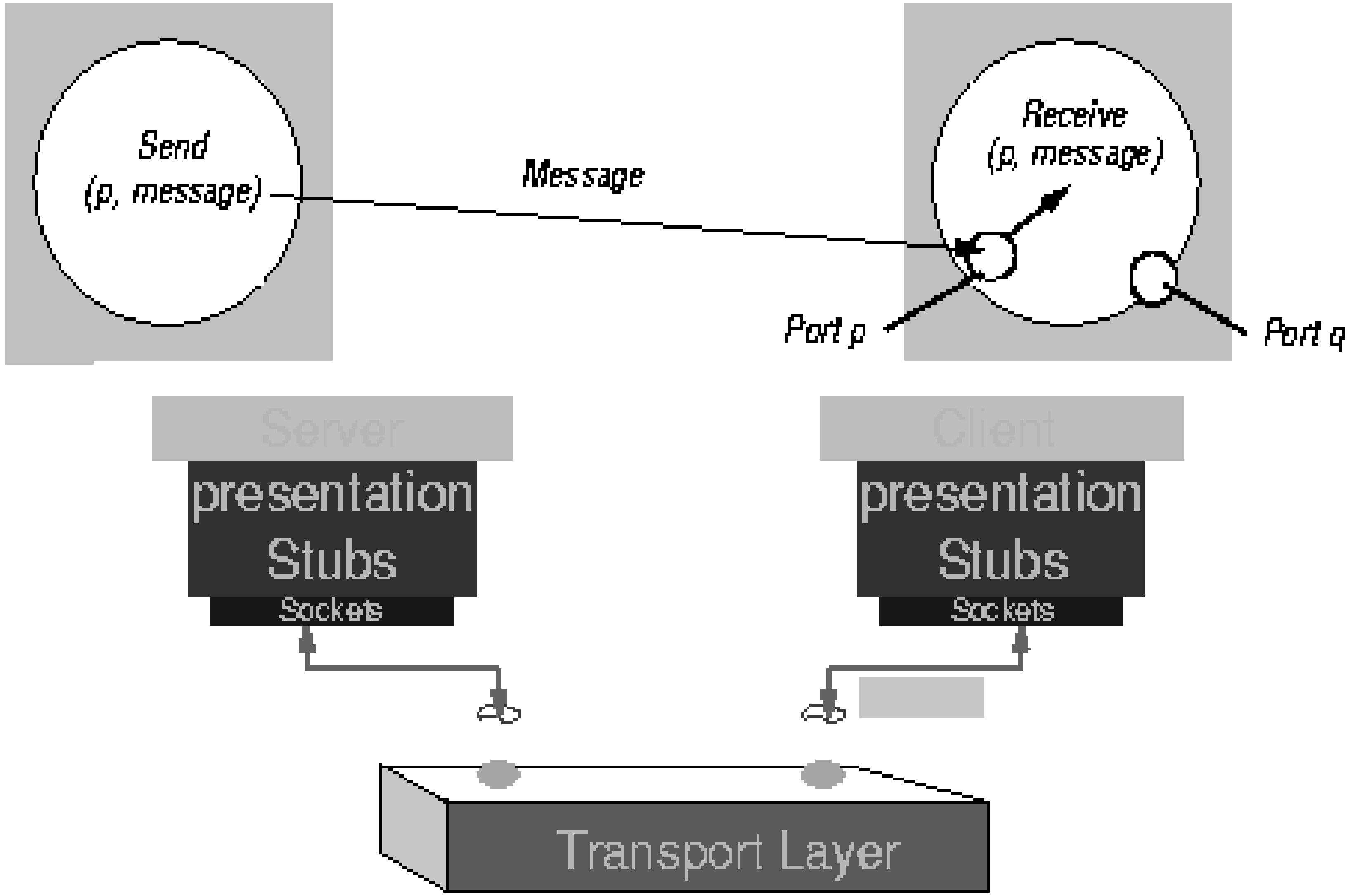
- Not all requirements are always necessary

Streams of Bits/bytes can be transmitted: so what?

How do we know what is the INFORMATION inside?

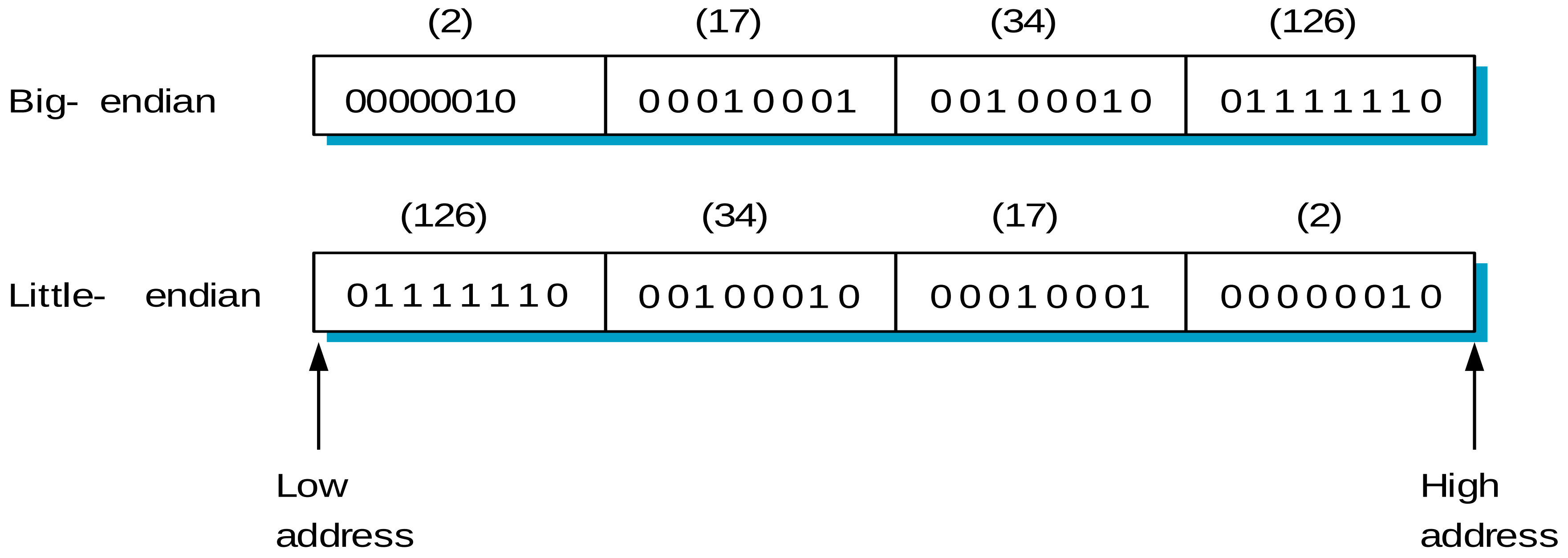
Inter-Process Communication

[wonkwang, modified]



Simple example

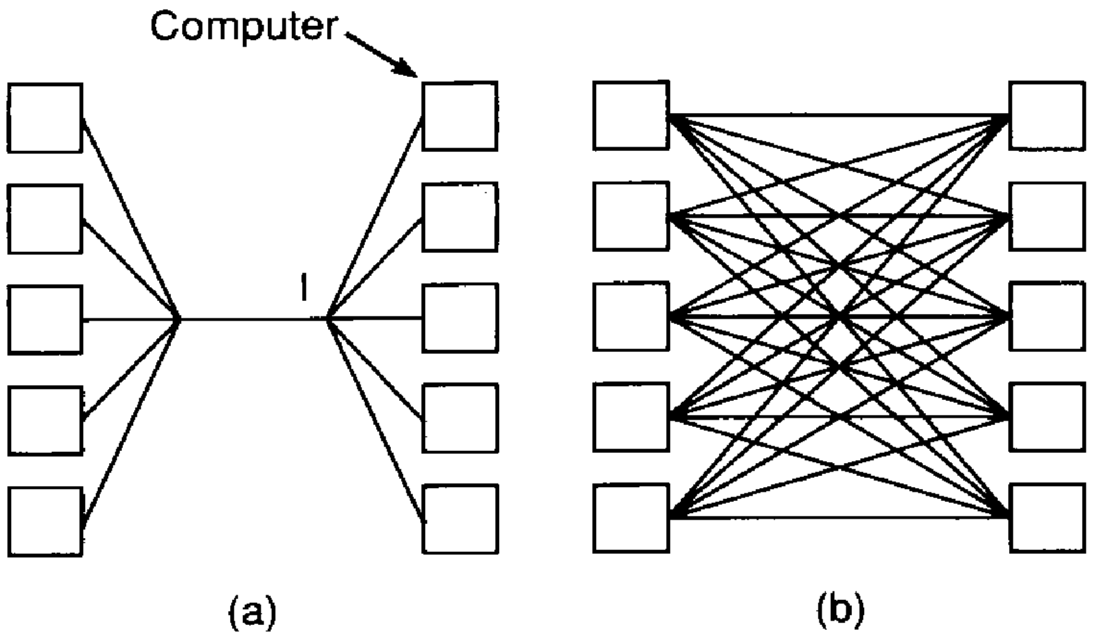
- Representation of base types
  - floating point: IEEE 754 versus non-standard
  - integer: big-endian versus little-endian (e.g., 34,677,374)



– on a 680x0 CPU, the 32 bit integer number 255 is stored as:
00000000 00000000 00000000 11111111
but an Intel 80x86-CPU stores this as:
11111111 00000000 00000000 00000000

Taxonomy

- Data types
  - base types (e.g., ints, floats); must convert
  - flat types (e.g., structures, arrays); must pack
  - complex types (e.g., pointers); must linearize

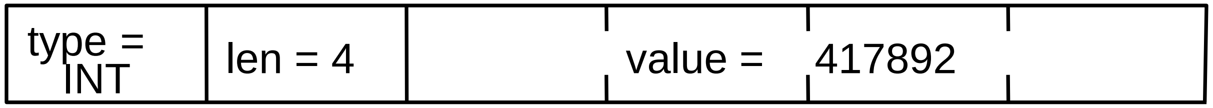


- Conversion Strategy
  - canonical intermediate form
  - receiver-makes-right (an *N* x *N* solution)

Data Conversion

- Two different types of rules are needed:
  - Abstract syntax: a station must define what datatypes are to be transmitted
  - Transfer syntax: it must be defined how these datatypes are transmitted, i.e. which representation has to be used.

  Tagged versus untagged data



Abstract Syntax Notation.1 - ASN.1

- Each transmitted data value belongs to an associated data type.
- For the lower layers of the OSI-RM, only a fixed set of data types is needed (frame formats), for applications with their complex data types ASN.1 provides rules for the definition and usage of data types.
- ASN.1 distinguishes between a data type (as the set of all possible values of this type) and values of this type (e.g. ‘1’ is a value of data type Integer).
- Basic ideas of ASN.1:
  - Every data type has a globally unique name (type identifier)
  - Every data type is stored in a library with its name and a description of its structure (written in ASN.1)
  - A value is transmitted with its type identifier and some additional information (e.g. length of a string).

Definition of Datatypes using ASN.1 (1)

- A data type definition is called „abstract syntax“; it uses a Pascal-like syntax.
- Lexical rules:
  - Lowercase letters and uppercase letters are different
  - A type identifier must start with an uppercase letter
  - Keywords are written in uppercase letters
- ASN.1 offers some predefined simple types:
  - BOOLEAN (Values: True, False)
  - INTEGER (natural numbers without upper bound)
  - ENUMERATED (association between identifier and Integer value)
  - REAL (floating point values without upper or lower bound)
  - BIT STRING (unbounded sequence of bits)
  - OCTET STRING (unbounded sequence of bytes/ octets)
  - NULL (special value denoting absence of a value)
  - OBJECT IDENTIFIER (denoting type names or other ASN.1-objects)

Definition of Datatypes using ASN.1 (2)

- Examples:
  - MonthsPerYear ::= INTEGER
    MonthsPerYear ::= INTEGER (1..12)
    Answer ::= ENUMERATED (correct(0), wrong(1),noAnswer(3))
- With the following type constructors new types can be built from existing ones:
  - SET: the order of transmission of the elements of a set is not specified. The number of elements is unbounded, their types can differ
  - SET OF: like SET, but all elements are of the same type.
  - SEQUENCE: the elements of a sequence are transmitted in the defined order. They can be of different types. The number of elements is unbounded.
  - SEQUENCE OF: like SEQUENCE, but all elements are of the same type
  - CHOICE: the type of a given value is chosen from a list of types (like a Pascal variant record)
  - ANY: unspecified type

ASN.1 Transfer Syntax (1)

- Some coding rules (the „transfer syntax“) specifies how a value of a given type is transmitted. A value to be transmitted is coded in four parts:
  - identification (type field or tags)
  - length of data field in bytes
  - data field
  - termination flag, if length is unknown.
- The coding of data depends on their type:
  - integer numbers are transmitted in High-Endian Two’s complement representation, using the minimal number of bytes: numbers smaller 128 are encoded in one byte, numbers smaller than 32767 are encoded in two bytes, ...
  - Booleans: 0 is false, every value not equal 0 is true.
  - for a sequence type first a type identification of the sequence itself is transmitted, followed by each member of the sequence.
  - Similar rules apply to the transfer of set types

We can now move MEANINGFULL Messages…

but how to use them?

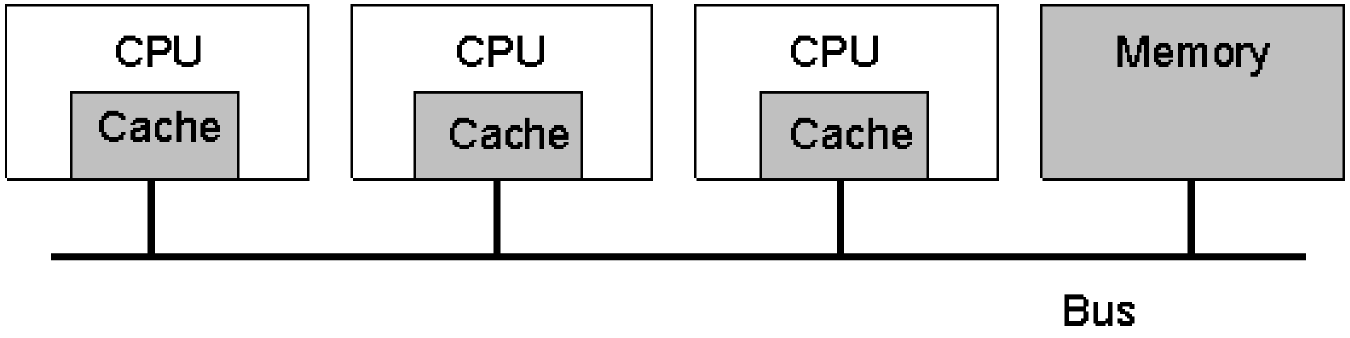
A Distributed System:

- *A distributed computing system consists of multiple* ***autonomous*** *processors that do not share primary memory but* ***cooperate*** *by sending messages over a communication network.*

Henri Bal/ Colouris

Multiprocessors vs. Distributed systems

- A bus-based multiprocessor.

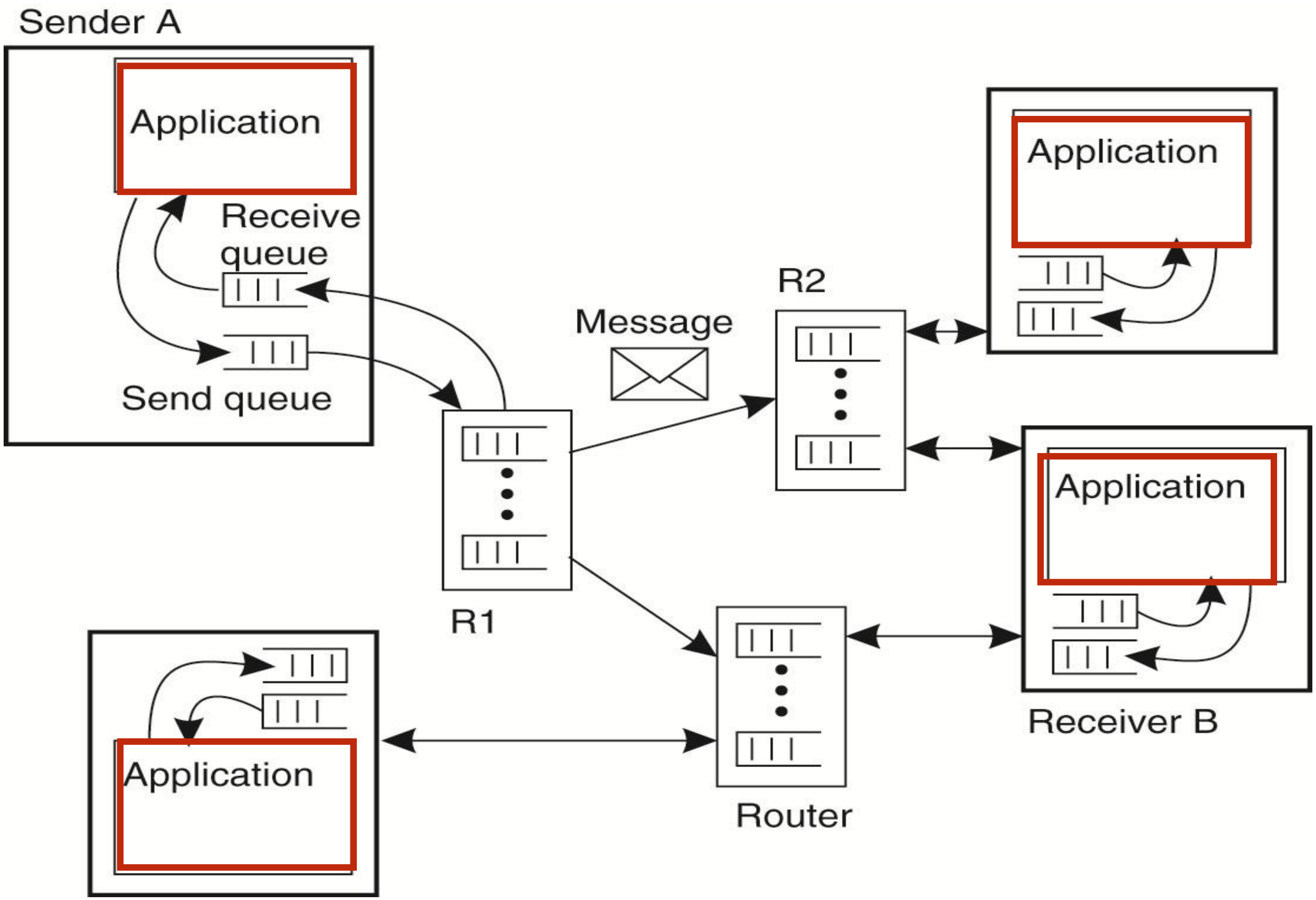


Typical features of a multiprocessor:

- Physical access to a common memory
- Resources under same management
- Very fast (bus, switching matrix) based local communication

A Distributed System

[Tanenbaum]



A Distributed System:

- *A distributed computing system consists of multiple **autonomous** processors that do not share primary memory but **cooperate** by sending messages over a communication network.*

  Henri Bal/ Colouris

- A distributed system is one in which the **failure** of a computer which you didn‘t even know existed can render your own computer unusable.

  Leslie Lamport

==

- A distributed system is a collection of **independent** computers that **appears** to its users as a **single coherent** system.

  A. S. Tannenbaum

Two important features of Distributed Systems

- **Autonomy**
  - A distributed system consists of ***autonomous,*** independent entities (***usually*** cooperative ones, ***sometimes*** antagonistic ones)
  - Each individual entity is – typically – a full-fledged, operational system of its own
  - Individual entities might follow local policies, be subject to local constraints…
- **Transparency**
  - The fact that the distributed systems is indeed a conglomerate of different (simpler) systems is of no interest and should be of no concern to the user

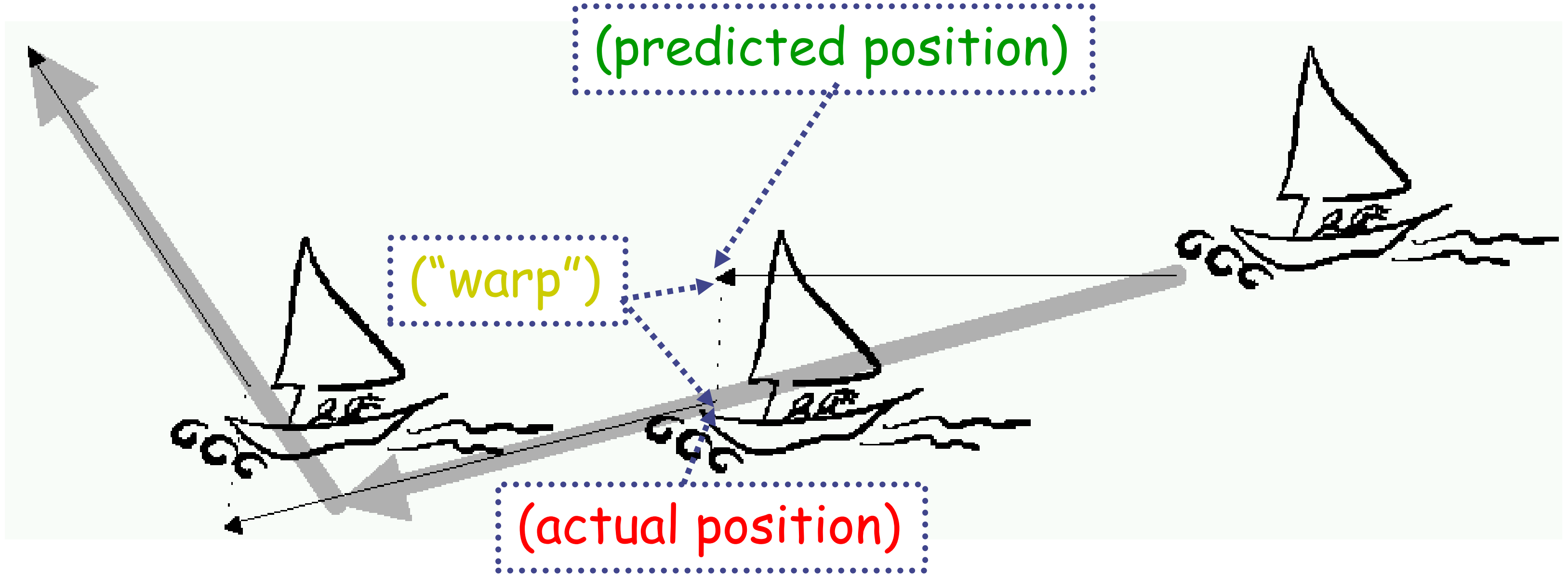
’

A dead Man that Shoots… [Mouve, Mannheim, 2005]

- Consider an action game: player A shoots player B.
  - Each of the players has a computer with a local copy of the game
  - Application of player A creates a bullet entity with a certain heading and velocity. It will then transmit the state of that bullet entity.
  - Upon receiving the state of the bullet, remote applications will start to check whether any entity that they control is hit by the bullet.
- There is network delay between the time A transmits the initial state of the bullet and the time B receives the state.
  - This network delay may be much larger (in the order of 100ms or more) than the amount of time that the bullet needs to hit its target!
  - During this time player B might take actions,
    - shoot at another player C, even though he’s already dead.
    - move so that the bullet would not hit him (from his point of view!!!)
- Players A, B, and C may therefore disagree about whether a hit has been scored on B or not!

Dead Reckoning

- Based on ocean navigation techniques
- Predict position based on last known position plus direction
  - Can also only send updates when deviates past a threshold



- When prediction differs, get "warping" or "rubber-banding" effect

The Distributed Stock Exchange…

- Consider a distributed stock exchange…
  - Brokers are distributed all over the world.
  - Each of the brokers runs a computer program
  - Each of the brokers sets his program to:
    Buy stocks of Company HOPE if they fall under XXX $
- The central unit (located at Wall street, NY) defines the value of the HOPE stocks and announces them …
- Due to different transmission delays Broker John – at Wall Street – will always react quicker than Paul in Frankfurt , or Wendong in Peking, or….
- But the relative advantage of Paul vs Wendong (or vice versa!) might dramatically change depending on traffic conditions….(e.g. local soccer league…)
  - What about Paul causing artificial load on Wendong´s computer by sending him numerous superfluous requests from a computer of his girl friend?

Authorizing information exchange

- Smart phones – or Body Area Networks.
- People would like to exchange information locally…
  - With whom? Not with EVERYBODY in your proximity…
- An (older) simple idea:
  accelerometers in “wrist watches” for detecting shaking hands… as a criterion for :
  - exchange of contact data between the smart phones?
  - Giving mutual access to the time planner?
  - Some other data?

Screenshot: The handshake

[M.Eyrich. TKN]

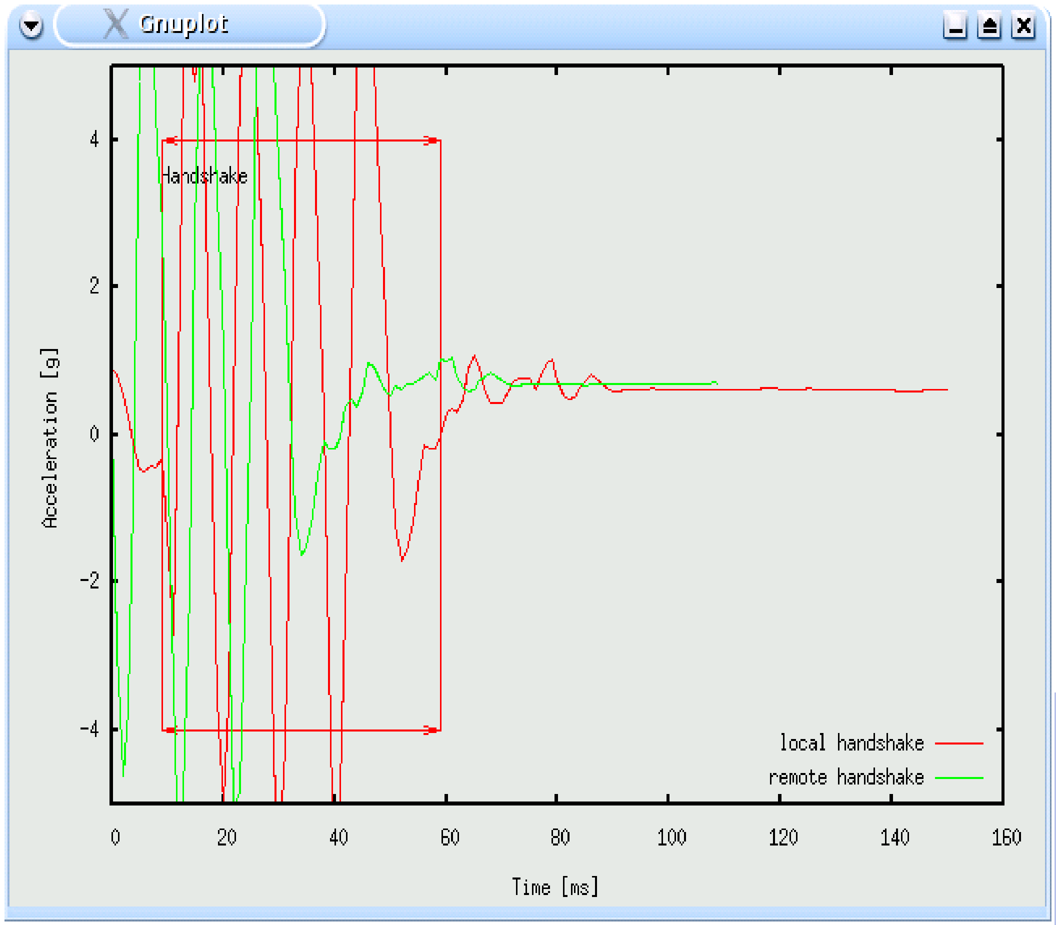
Handshake data: Red curve- Local; Green curve – Partner. (Shift of the curves is given by local optical alignment)

- Red square:

  The data section providing sufficient match for assuming a joint handshake.

- But: Each of the participant might have a DIFFERENT decision Algorithm...

- ➔**Decisions about data export and data acceptance might be contradictory!!!**



Transparency in a Distributed System [Tannenbaum]

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Note: There is a difference between the FEATURE, MECHANISM and POLICIES
Take openness and scaling into account…➔

Different forms of transparency in a distributed system (ISO, 1995)

Design Challenges: Openness

[J. Hou, Illinois]

- Degree to which new resource-sharing services can be added & used.
- Requires *publication* of interfaces for access to shared resources.
- Requires *uniform communication* mechanism.
- Conformance of each component to the published standard must be tested and verified.

Design Challenges: Scalability

[J. Hou, Illinois]

- A system is said to be scalable if it will remain effective when there is a significant increase in the number of **resources** and **users**:
  - Controlling the cost of resources
  - Controlling the performance loss
  - Preventing software resources running out (e.g., IP addresses)

Characteristics of decentralized algorithms

- No machine has complete information about the system state.
- Machines make decisions based only on local information.
- Failure of one machine does not ruin the algorithm.
- There is no implicit assumption that a global clock (i.e. precise common understanding of time!) exists.

Pitfalls when Developing Distributed Systems

- The network is reliable.
- The network is secure.
- The network is homogeneous.
- The topology does not change.
- **Latency is zero.**
- **Bandwidth (= bit rate!) is infinite.**
- Transport cost is zero.
- There is one administrator.