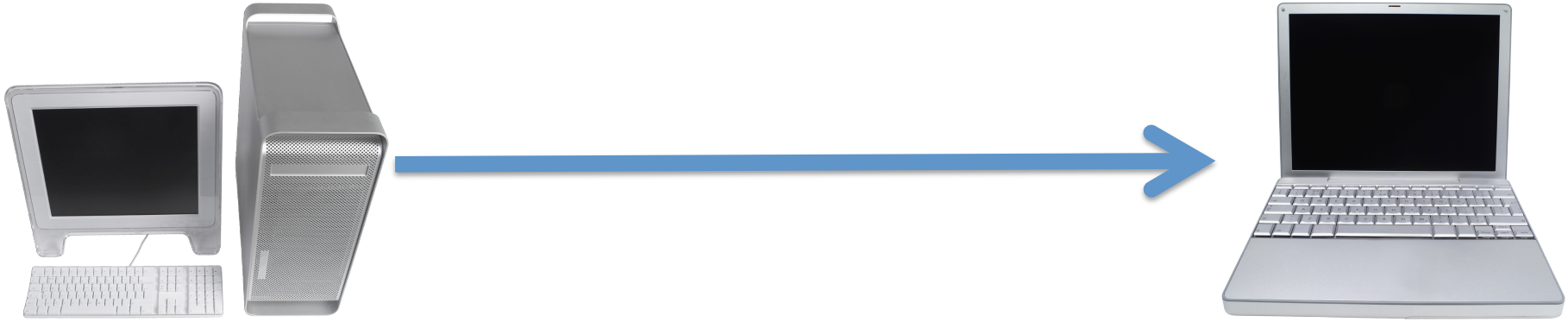
Architecture, interaction design

Agile roles

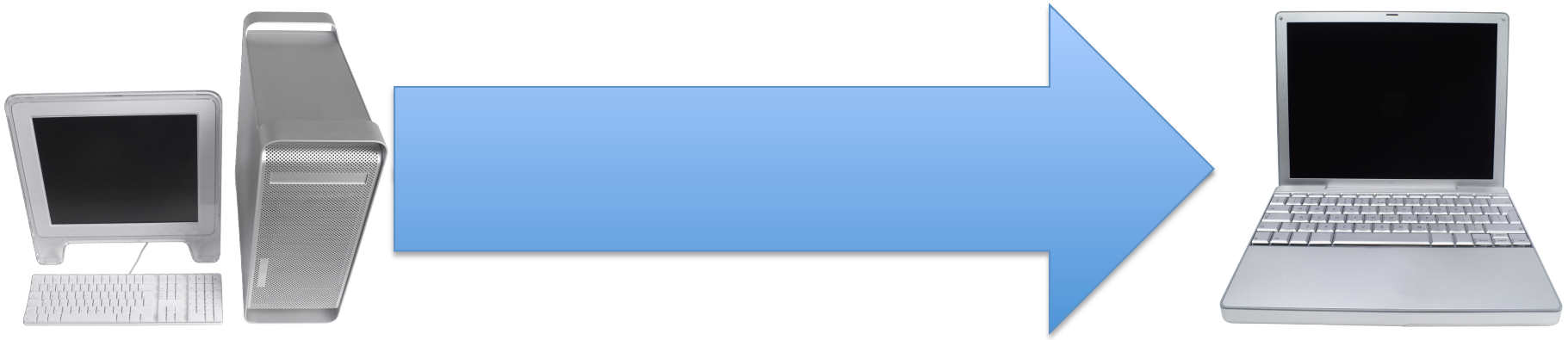
- Architect
  - A team member (programmer) who designs interfaces and data structures
- Interaction (protocol) designer
  - A team member (programmer) who designs message flows over the internal and external interfaces

What we want to do



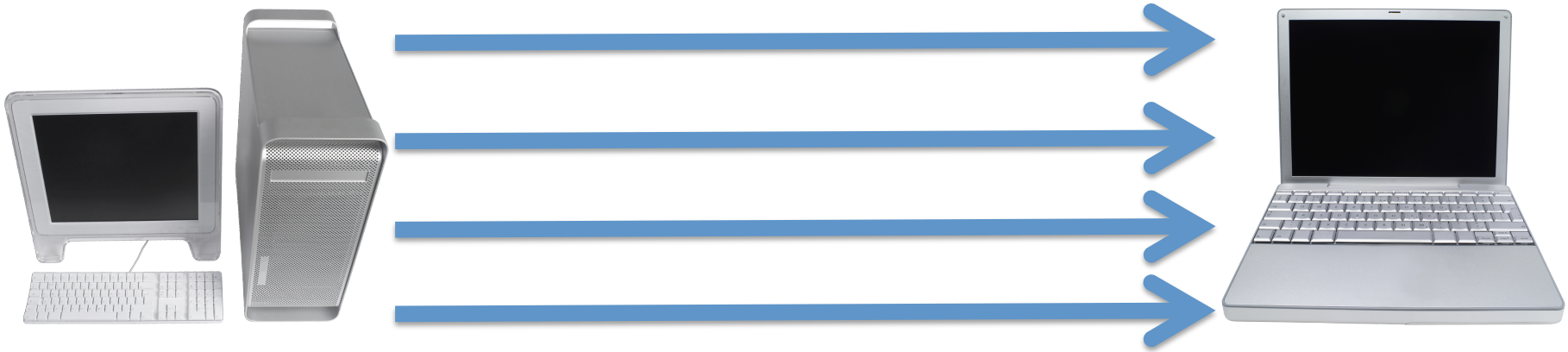
- Get data from one node to another
- In a network of a set of nodes/computers/ software component
- Taking requirements into account

All at once



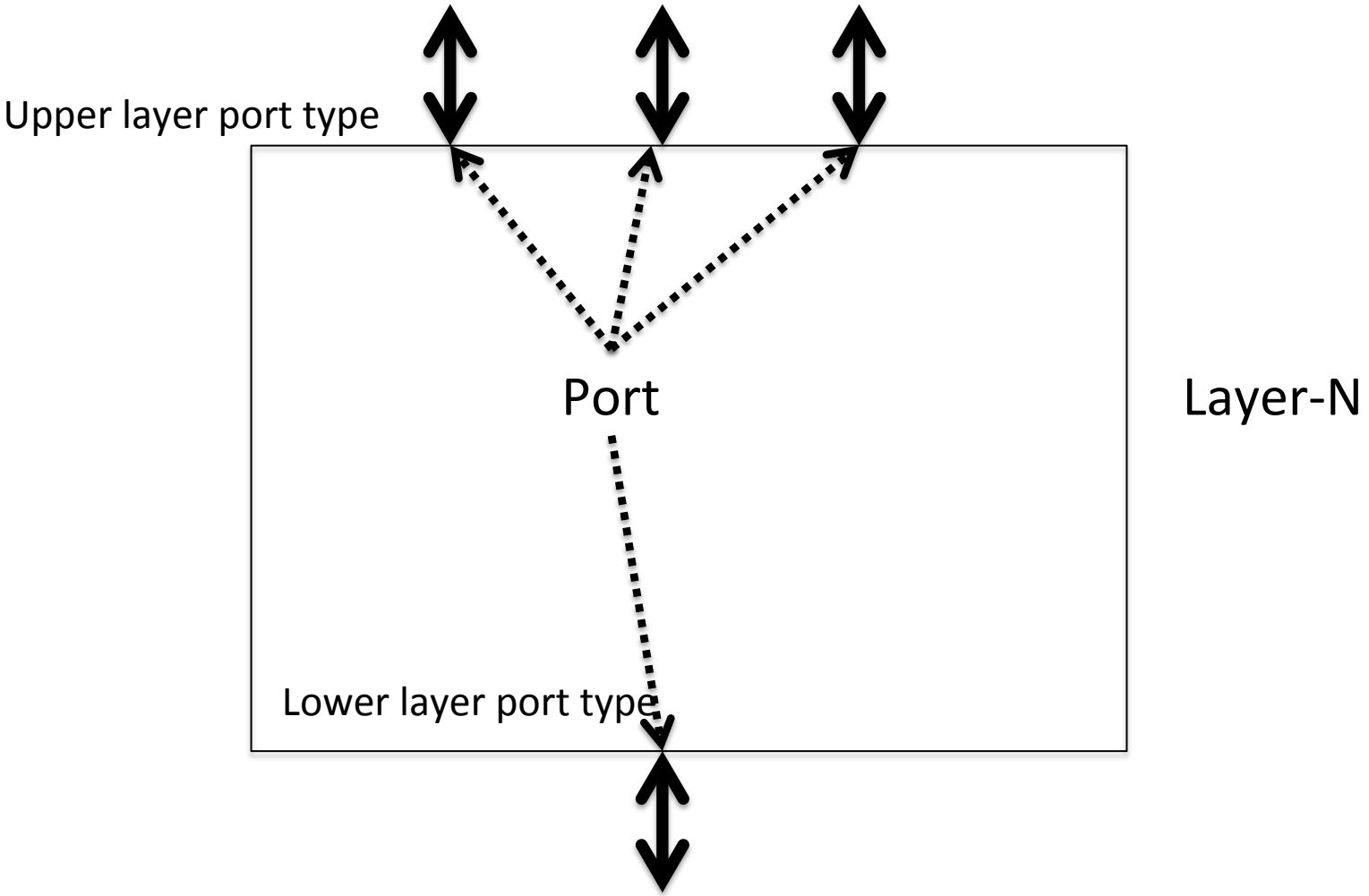
- Sending all pieces of data in one message
- Complex data structure
- Simple protocol
- Used in software, computer protocols and higher layer network protocols

In chunks



- Transfer each data item in a separate message
- Simple data structure
- Complex protocol
- Used in telecommunications networks

Network node model



Network node model

- Analogy between network node and software component
  - Layer vs. tier
  - Port type vs. interface
  - Message vs. function declaration
  - Message format vs. formal parameter list
  - Ingress port activation vs. function called or return from a call
  - Egress port activation vs. calling a function or returning a value
  - PDU vs. actual parameter value

Network node model

- Each node has
  - A set of port types
  - A set of ports
- Each port type has
  - Message types
- Each message type has
  - Message format
- Each port has
  - A message queue (may be of length 1)
  - The queue may be shared

Input for architects and designers

- Requirements from:
  - Textual protocol specification
  - Customer stories – explained by domain experts

Elements of a protocol specification

- What to look for in the requirements?
  - Services of the system, module, component
  - Assumptions about the execution environment
  - Messages
  - Message formats
  - Behavior rules

Environment

- What to look for?
  - Is it reliable? → If not, then connection oriented protocol must be used and data must be protected with a checksum
  - Is is secure? → If not, then security and policy extensions may be necessary
- Differences between network node and software component
  - In software the environment is reliable, in communications it is nondeterministic
  - In communications the frequency of environmental events is higher

Service

- Input for both architects and designers
- What to look for?
  - Node types → This defines the network architecture
  - Is it reliable? → If yes, then flow control must be adopted
  - Is it synchronous? → If not, then message identification is necessary
  - Is it symmetric? → If yes, then P2P, if not, then client-server or SOA
  - Is it between neighbors? → If not, then addressing is necessary

Functional decomposition

- Breaking down top level functions to functions of smaller granularity
  - Higher level functions: services of components
  - Leaf-level functions: operational functions – executing constraints
- The ordering of lower level functions is irrelevant for the decomposition

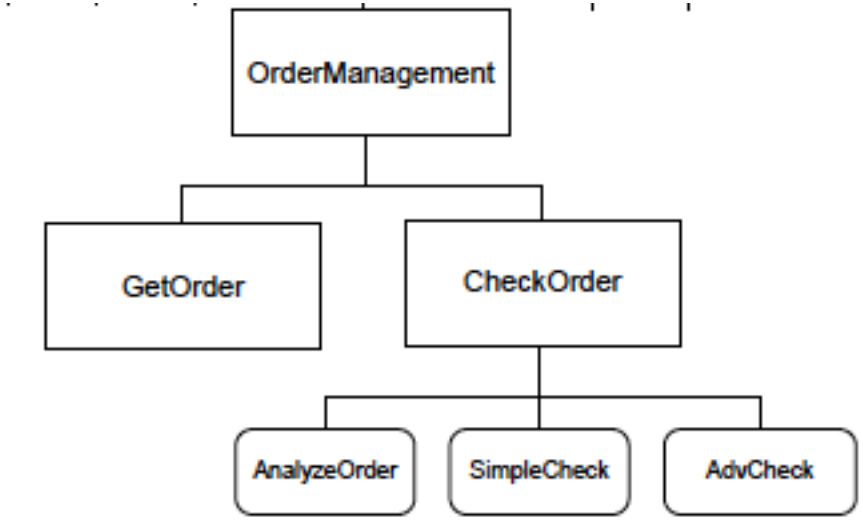


Fig. from Weske: Business Process Management – Concepts, Languages, Architectures, Springer, 2007

Service in networking environment

- Specific to telecommunications architecture
- Service is subdivided into:
  - Data plane (Forwarding plane)
  - Control plane
  - Management plane
- Often different protocols implement the planes

Service in networking environment

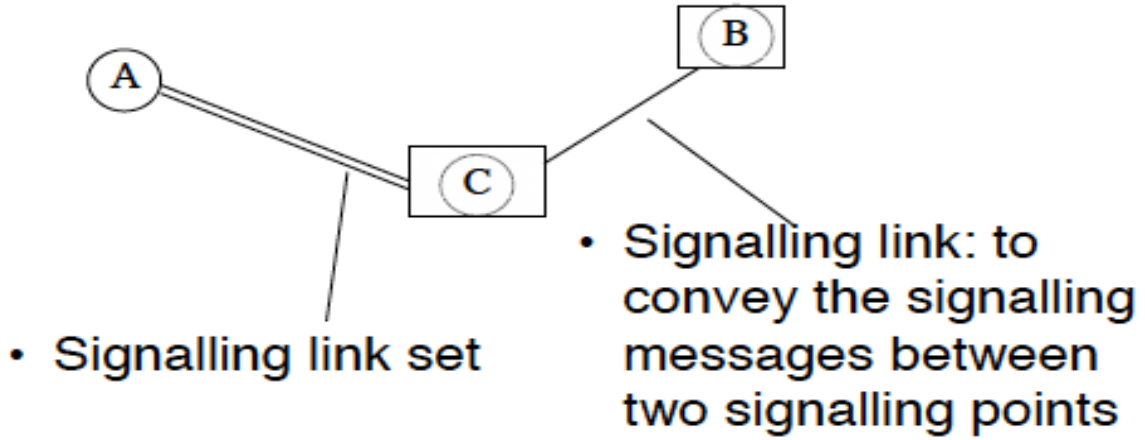
- Data plane
  - Forwards incoming requests, user traffic
  - Parses protocol headers
  - Queuing, QoS, filtering
- Control plane
  - Learns what the node should do with incoming requests -- control traffic
  - Maintains resource state and exchanges status information between peers
- Management plane
  - Controls and observes node status

Service examples

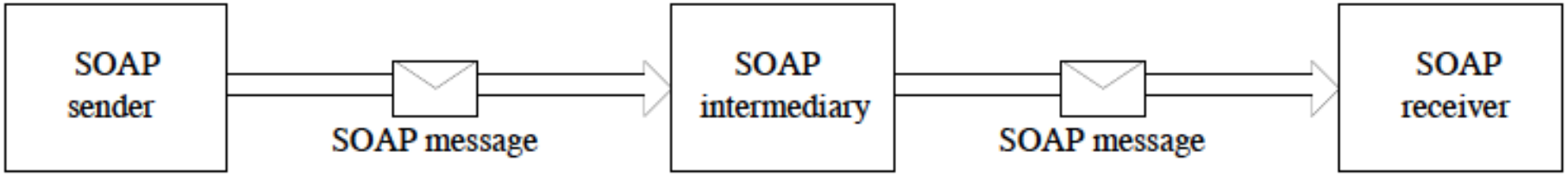
- SS7 MTP2
  - Historic control plane protocol for ISDN networks
  - Service: reliable transfer and delivery of signaling information between neighboring signaling nodes
- SOAP
  - Core protocol of XML web services (data plane, application layer)
  - Service: transfer of XML documents between neighboring SOAP nodes over a network

Network architecture example

- Network nodes may be on the same computer, even in the same software component
- SS7 MTP2



- SOAP

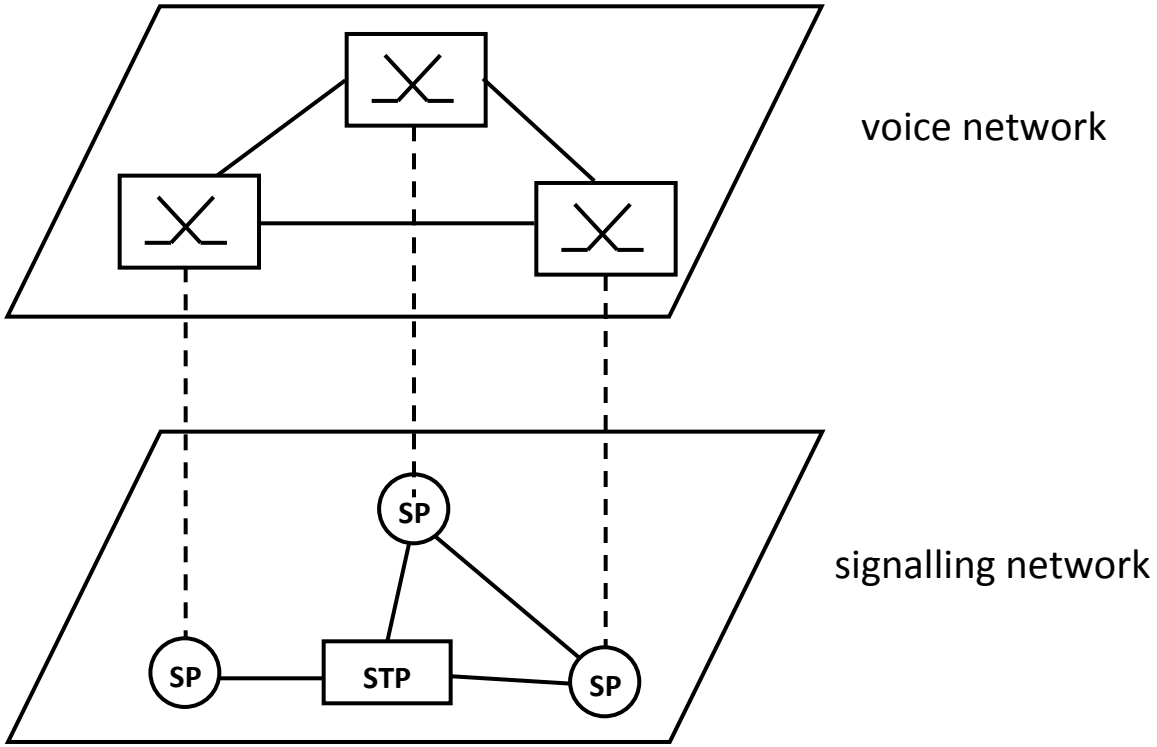


Multilayer and multitier

- Multilayer architecture
  - In communication networks
  - Usually between symmetric peers
  - Service is functionally decomposed into layers
    - Each layer has a small, well-defined task
    - Each layer has a unique language, a small set of functions
    - Higher layers are transparent to lower layers
    - Each layer has an overlay network
    - Layer protocols are components and hence replaceable

Multilayer and multitier

- Overlay network

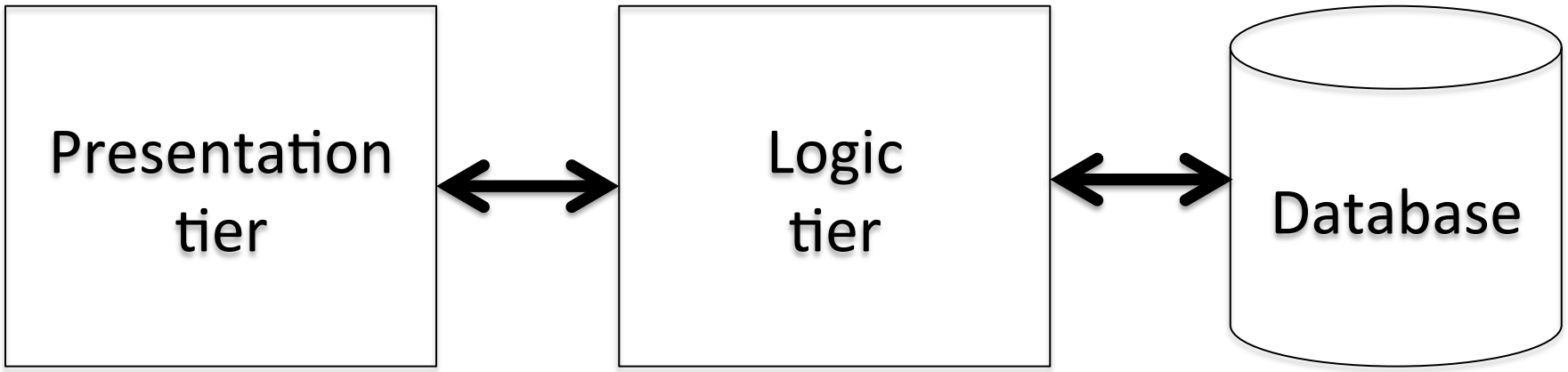


Multilayer and multitier

- Multitier (or N-tier) architecture
  - In software systems
  - Usually in client-server communication
  - Usually each tier is a separate network/ infrastructure node
    - Internal data representation
    - Unbounded number of data access functions
    - Data is not encapsulated but translated on tier borders

Multilayer and multitier

- Three-tier architecture



UI application

Application server

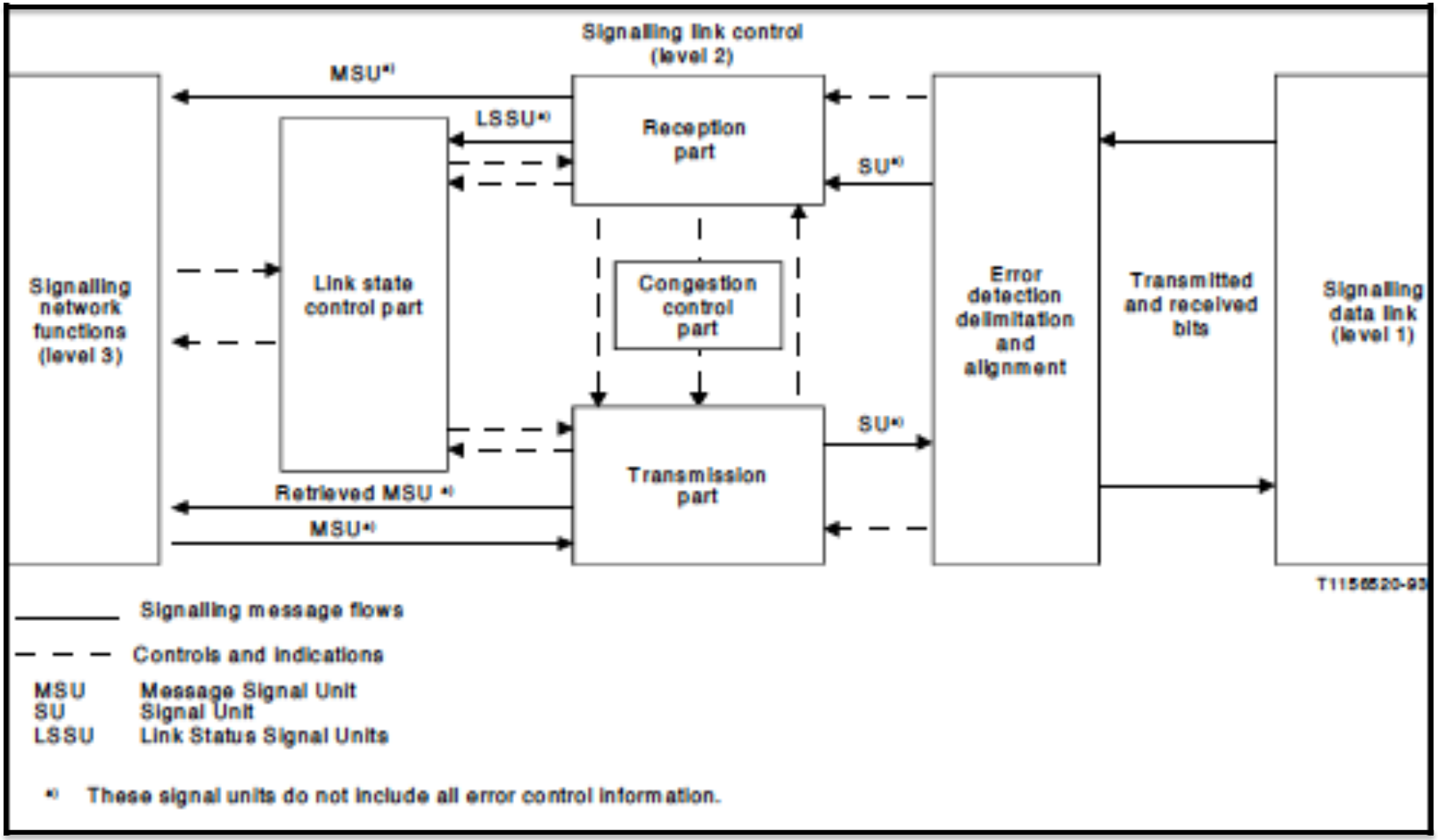
Database server

Multilayer and multitier

- Layer/tier elements
  - Client/transmitter
  - Server/receiver
  - Encoder-decoder → data serialization
  - Control, management
- If P2P, same transmitter and receiver in all nodes
- If client-server or SOA, unique transmitter and receiver in each node

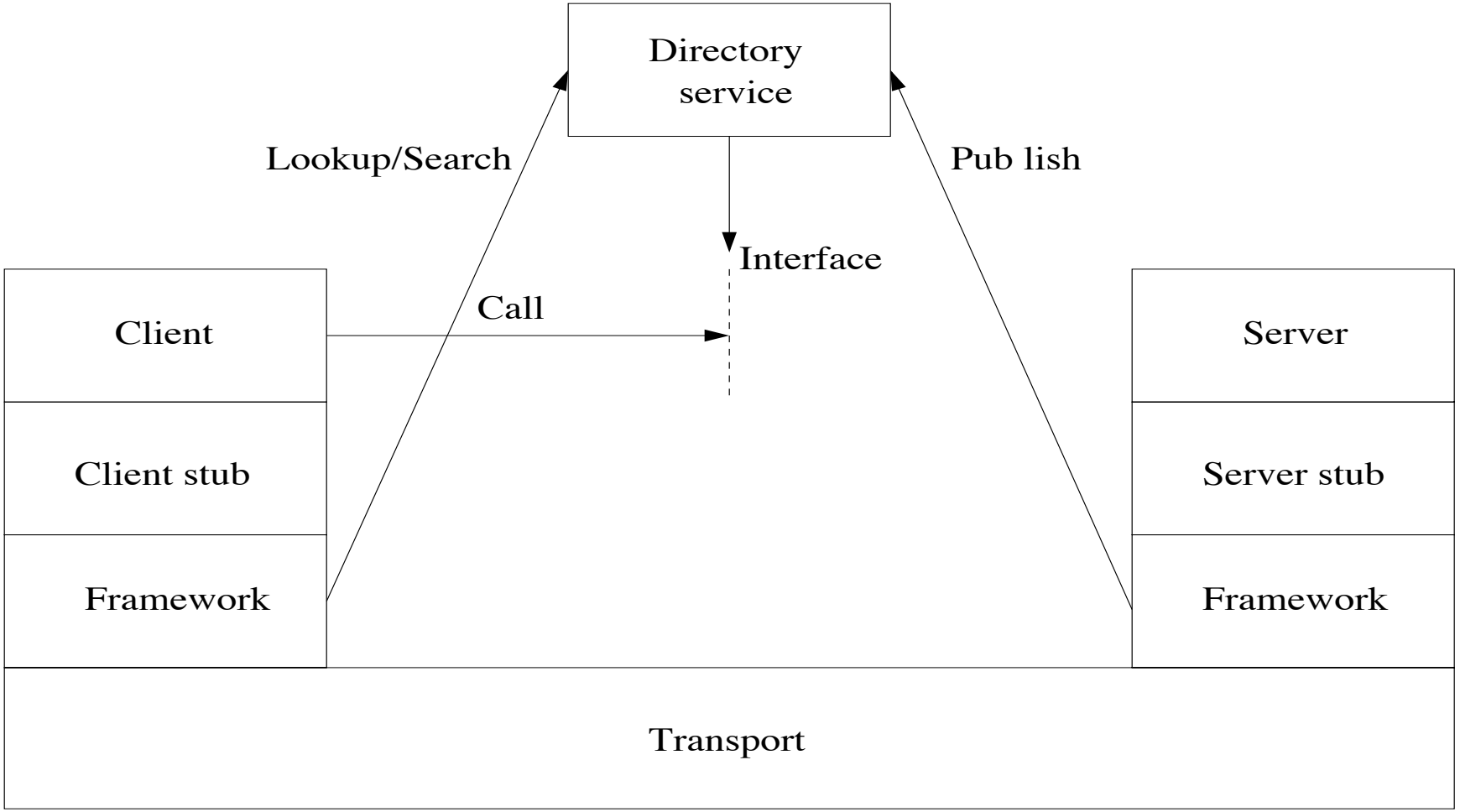
Example

- Internal structure of a telecom protocol



Example

- The CORBA distributed communications model



Message type

- Task of the architect
- Message = function identifier on an interface
- What to decide?
  - Synchronous or asynchronous
  - One-way, request-response, solicitation-acknowledgement, notification
  - What errors can occur?
- Functional decomposition: subservices → layered model

Message type

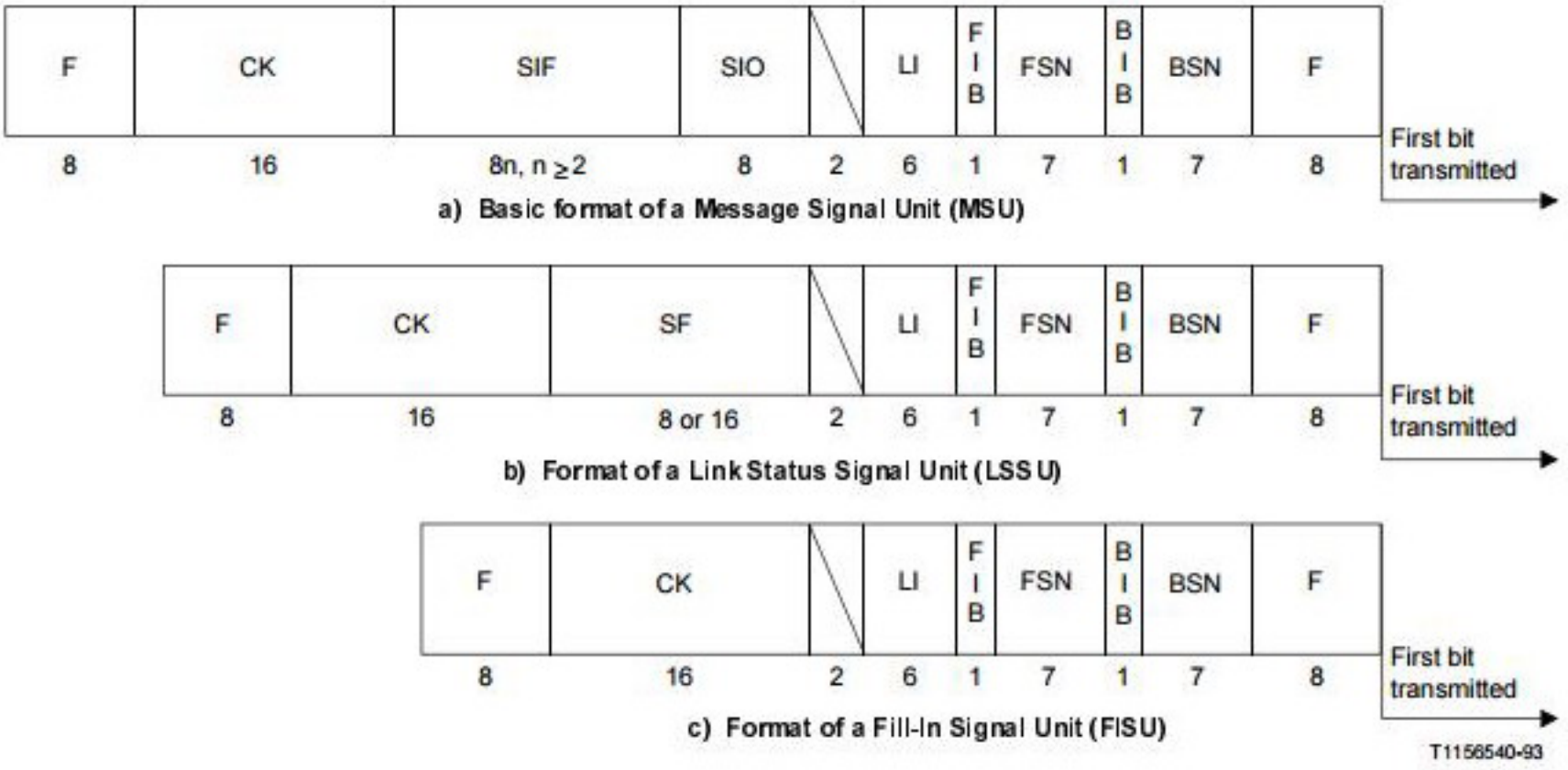
- How many message types should be defined?
  - As less as possible
  - All message types that appear on a public interface must be supported throughout the life cycle of the product – that can be expensive
  - Alternative: optional headers as key-value pairs

Message format

- Task of the architect
- Message format = function signature or return type
- What fields to include?
  - User data
  - Header data
    - Message number if asynchronous
    - Sequence numbers and fault and acknowledgement request indicators if reliable
    - Address if network of similar nodes exists

Example

- MTP2 message types and message format



Example

- SOAP message type and format

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding”>
  <soap:Header>
    ... control data
  </soap:Header>
  <soap:Body>
    ... user data
    <soap:Fault>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Message queue

- Each port of a component is associated with a queue
- Queue types:
  - Single FIFO – ordering of arrivals are lost
  - Multiple (per port) FIFO – which input queue to consider next
  - Typed message pool – which type of message to look for in the queue
  - General message pool – reception may depend on message parameters (TTCN-3 – later)
- A delaying network link itself is a queue

Behavior modeling

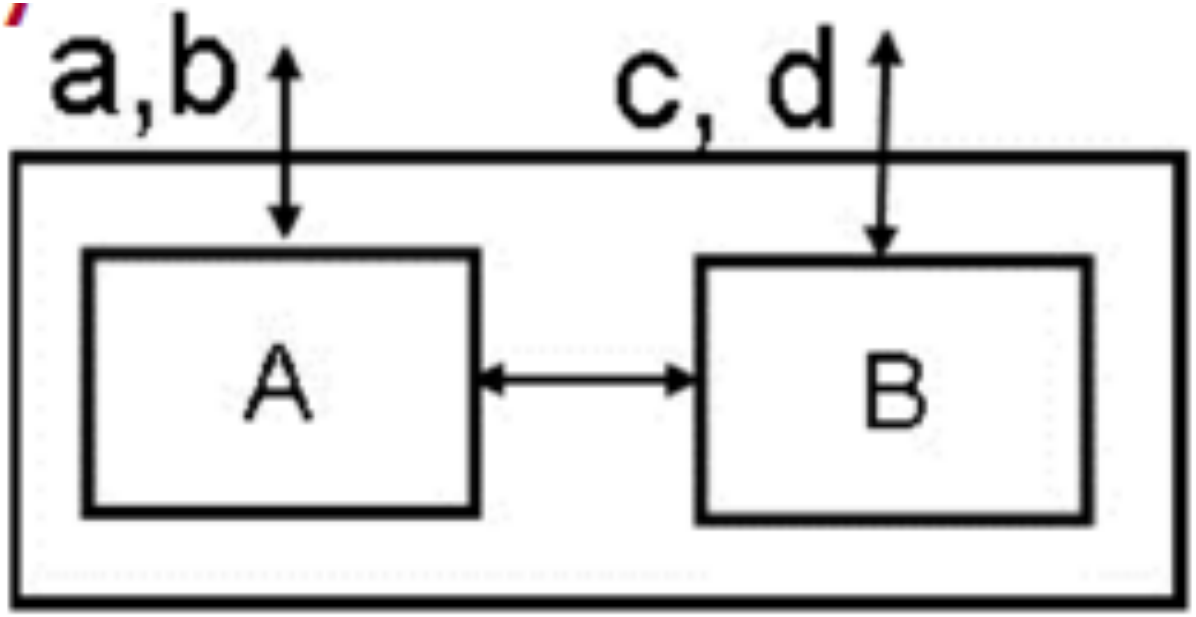
- Mealy finite state machine
  - Set of input events
    - Incoming port activations
    - Internal timeouts – not observable from the outside
  - Set of output events
    - Outgoing port activations
  - Set of states
    - Variables dedicated to remember I/O event history
    - Not observable from the outside
  - State transition rules
    - What is the next state and the output event upon an input event in a certain state

Problems from the environment

- Transport medium is not reliable
  - modeled as a non-deterministic function
- Messages may be
  - delayed
  - lost, corrupted
  - inserted, duplicated
  - reordered

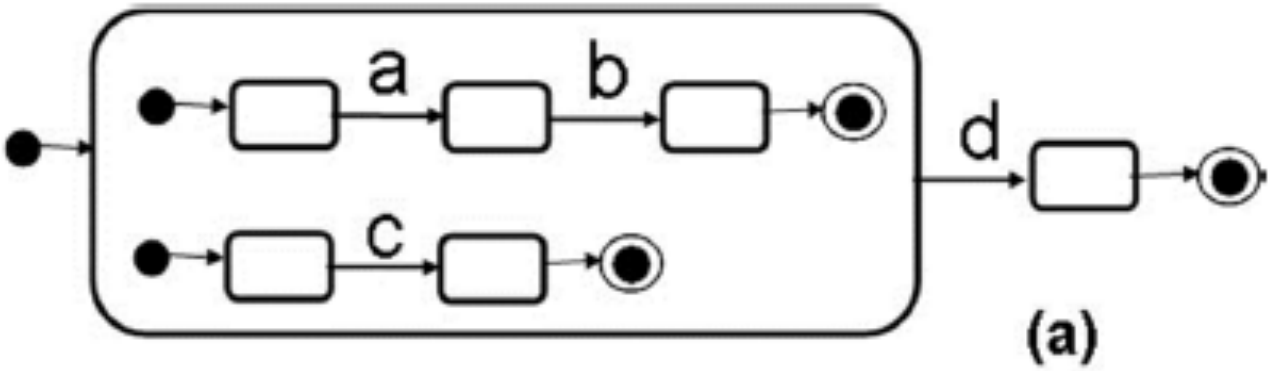
Example

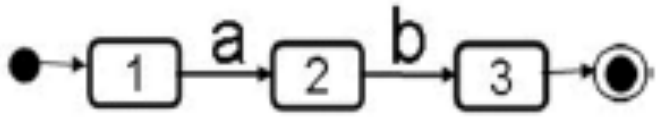
- Protocol layer with two ports
- Components A and B may be local or distributed

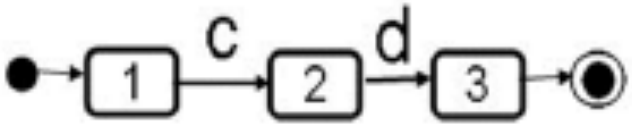


Example

- A behavior expressed with state machine

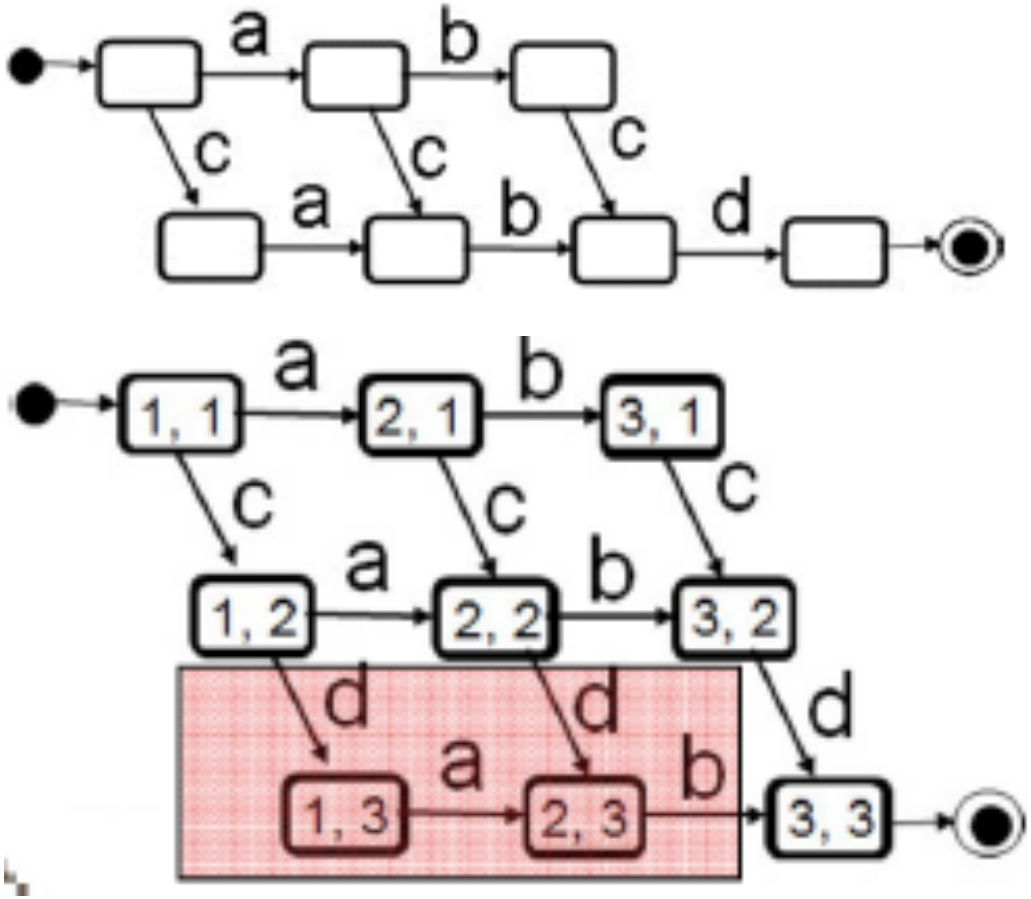






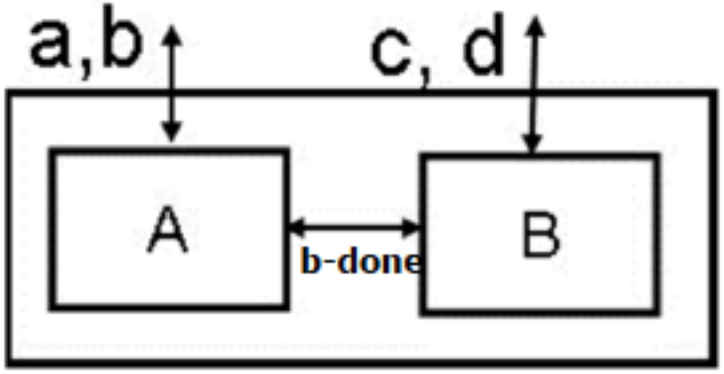
Example

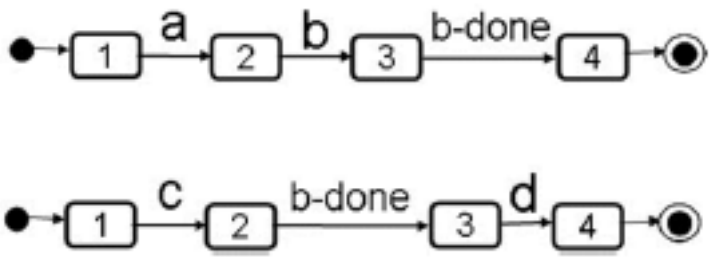
- Desired and invalid behaviors



Example

- How to fix?
  - Rendezvous communication
  - Avoids cross-over of messages over the interface
- If the synchronization message is lost, the system deadlocks, hence must be protected with timers





Synchronous, asynchronous

- Rendezvous communication
  - Synchronous
  - Remote procedure call
  - Caller must wait for the called – return
  - Queue of one element
- Message passing communication
  - Non-blocking
  - Multiple messages can be on route – queuing
  - Callback instead of return type
  - Message identification necessary

Addressing

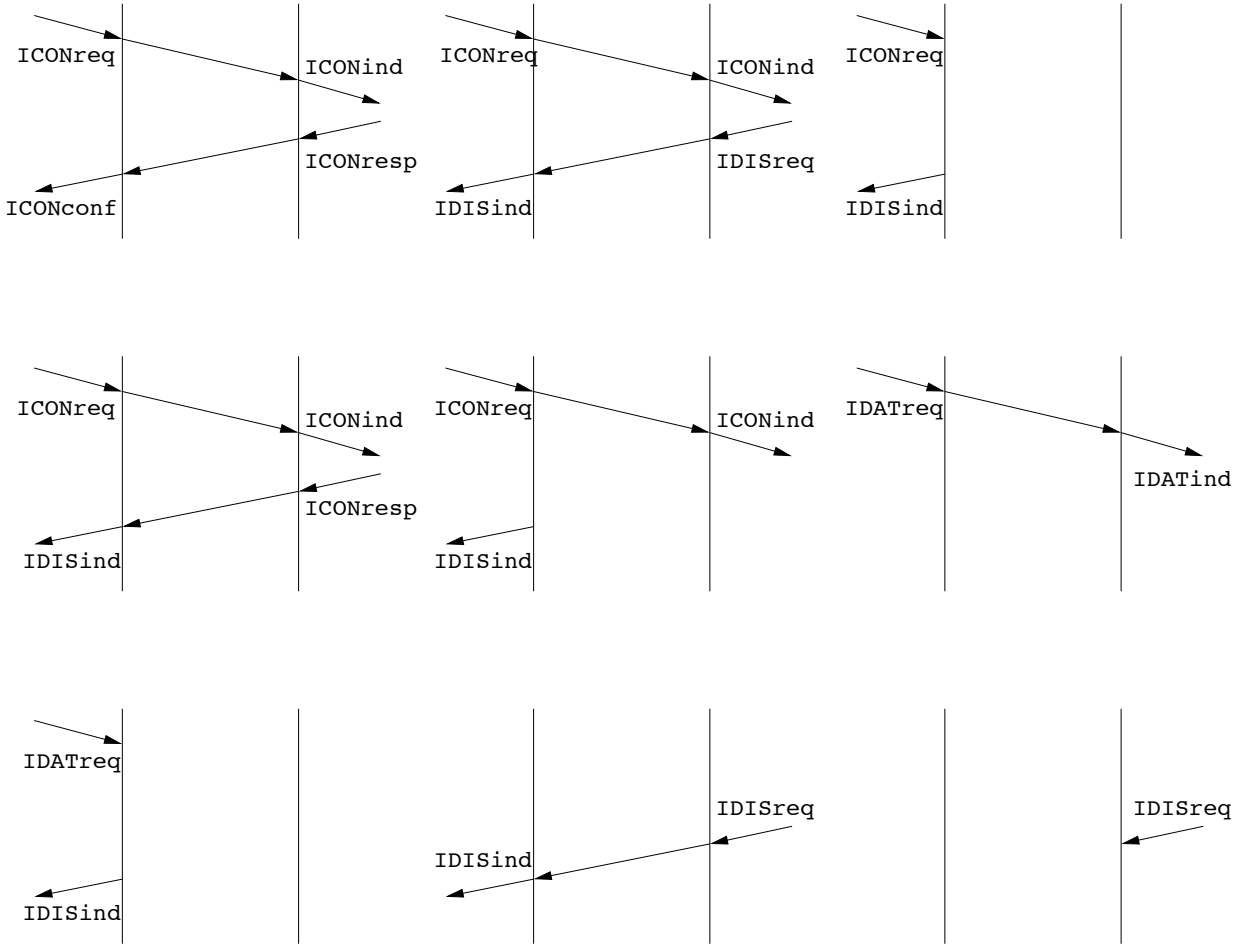
- Endpoint addressing is necessary if the communication is not P2P
  - Communication is asynchronous
    - The request P2P session is terminated once delivered, and the peer must know where to send the callback message
  - Next-hop routing is used
    - Over a link layer service or WS-Addressing
    - Each node along the path examines the destination endpoint address, and forwards the message to a node closer to the destination

Connectionless, connection oriented

- If messages must arrive exactly or at least once or in order a P2P connection must be set up between the two nodes
- Connectionless
  - Example: HTTP
- Connection oriented
  - Connection/session id
  - Tracking delivery of messages
  - Extra message types for connection setup, termination and management

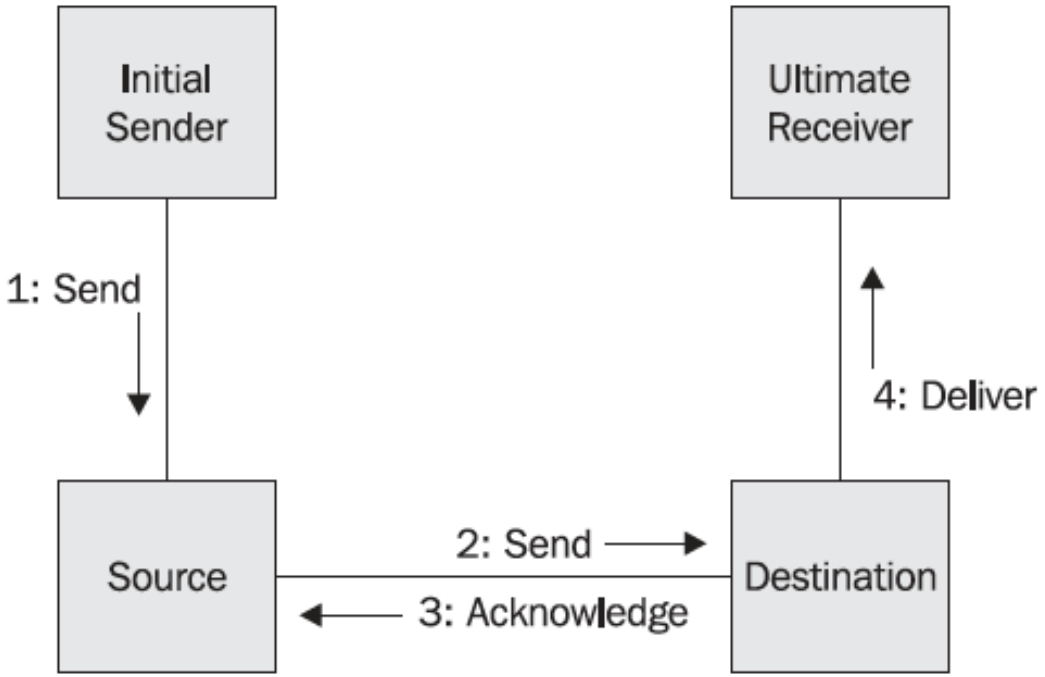
Example

- Connection orientation with primitives



Example

- SOAP is unreliable: WS-ReliableMessaging extension

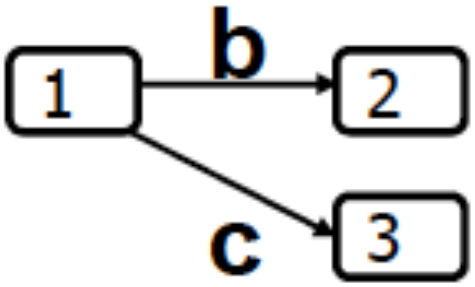


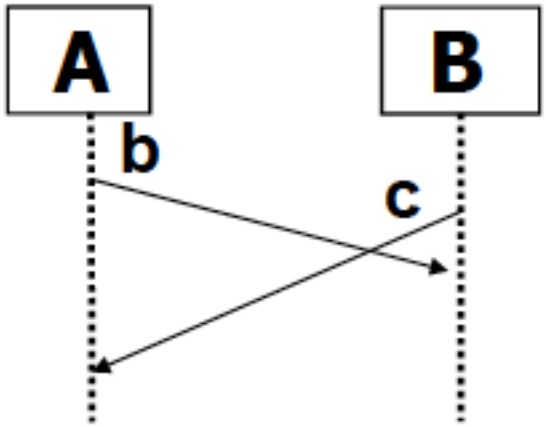
Example

- SOAP connection control
  1. CreateSequence (s→d)
  2. CreateSequenceResponse(id) (d→s)
  3. Sequence(id, msgId) (s→d)
  4. SequenceAcknowledgement – optional (d→s)
  5. AckRequested (s→d)
  6. TerminateSequence(id) (s→d)
- Connections and messages have id

Racing messages

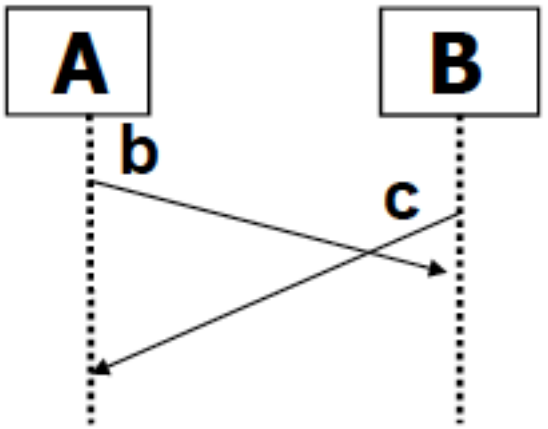
- If the source of *b* and *c* is the same component, the ordering is a local choice
- If different component initiate *b* and *c*, then those are competing





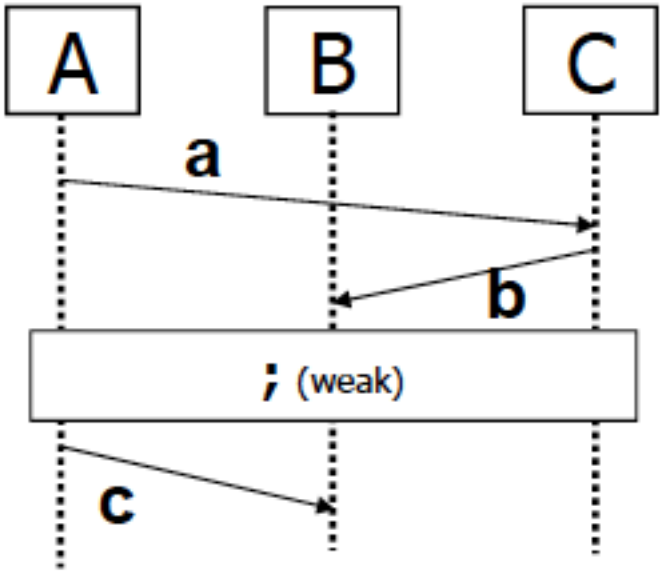
Racing messages

- Both components will be in different states
- Possible solutions:
  - Coordination protocol
  - Priority for one side
  - Non-specified reception
    - Undefined behavior
    - Error state
    - Message is ignored



Racing messages

- Two messages sent from a component arrive at different ports of another component
- Ordeding enforced in A
- That has no influence on B
- Coordination is necessary
- Problem a b is a loop
  - When does the loop end?

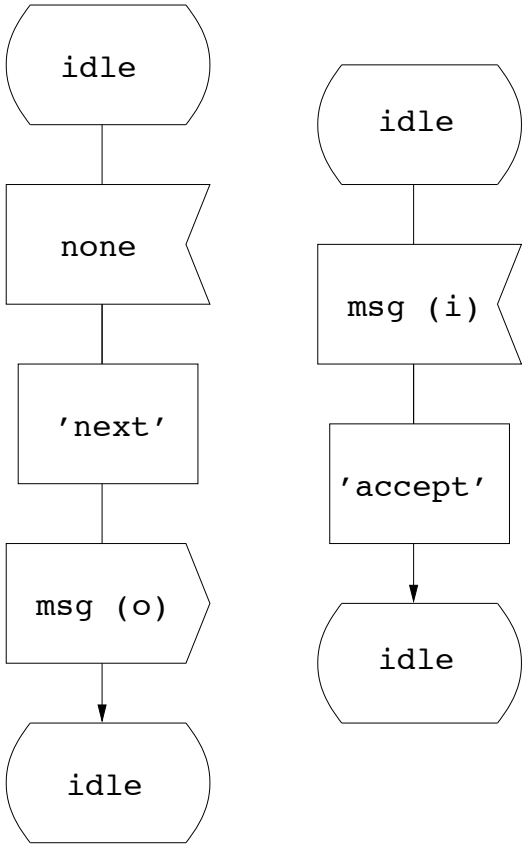


Flow control

- The sender produces data faster than the receiver could process them
- Protects against: message deletion, insertion, duplication and reordering, racing conditions, deadlock
- Elements:
  - Sequence numbers
  - Control messages
  - Timers

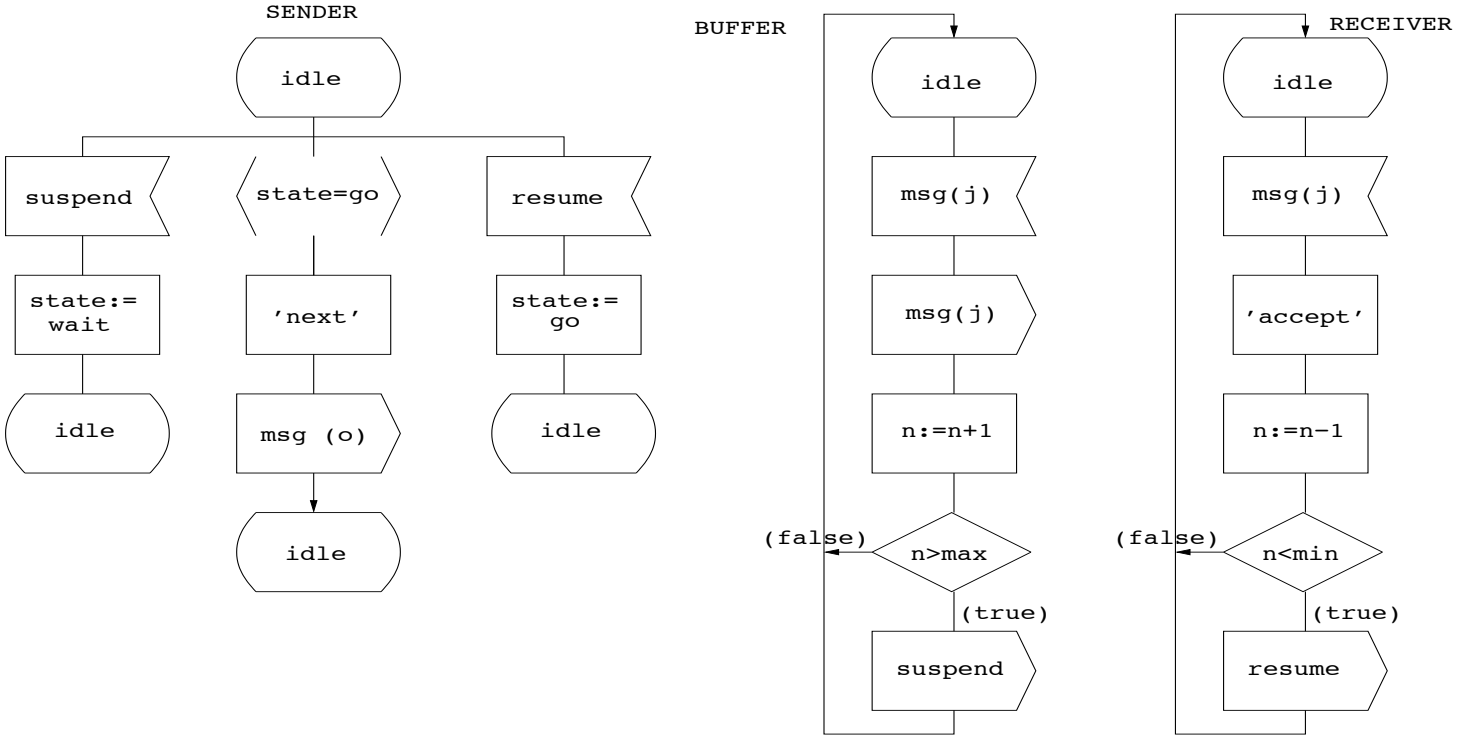
No flow control

- The receiver process must be faster than the sender



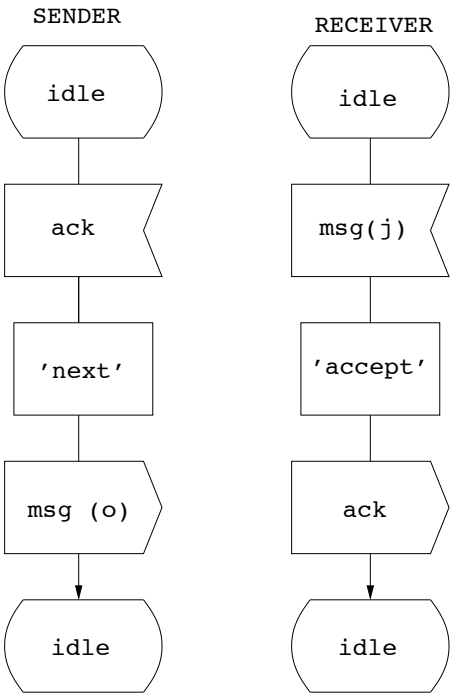
On/off

- The receiver sends suspend if its buffer is full and resume if it is empty
- Problem: loss or delay of control message



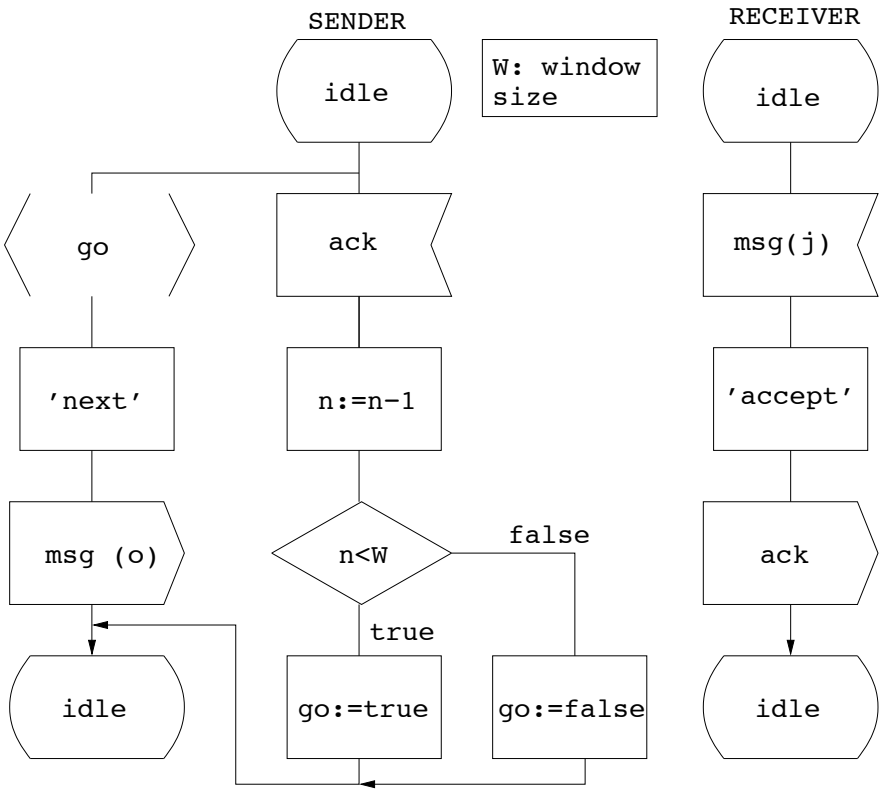
Stop and wait

- The receiver acknowledges each message, the sender must wait for the acknowledgement
- Deadlock if a control message is lost



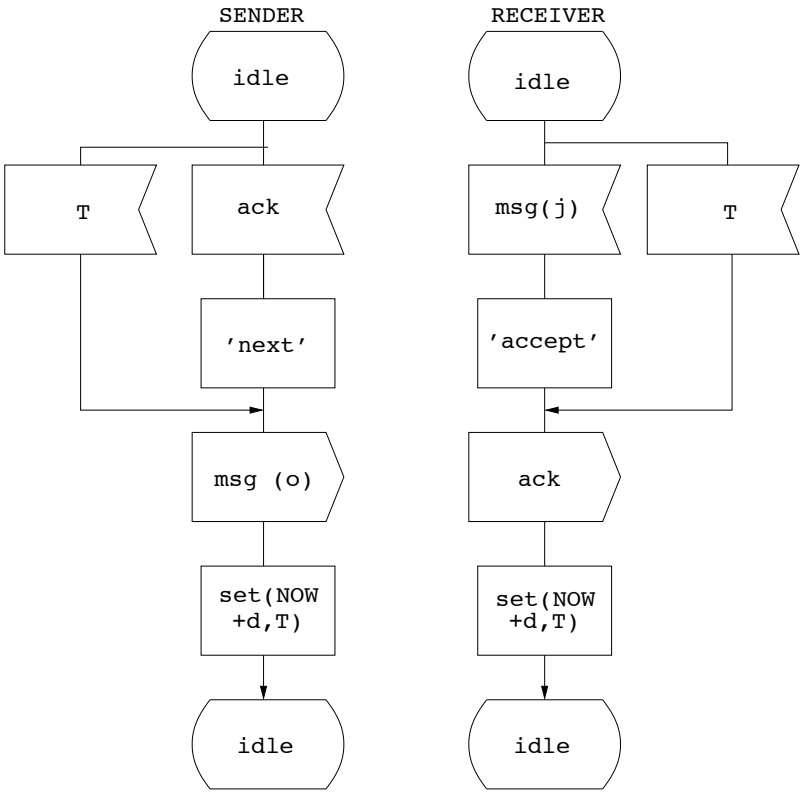
Windowing

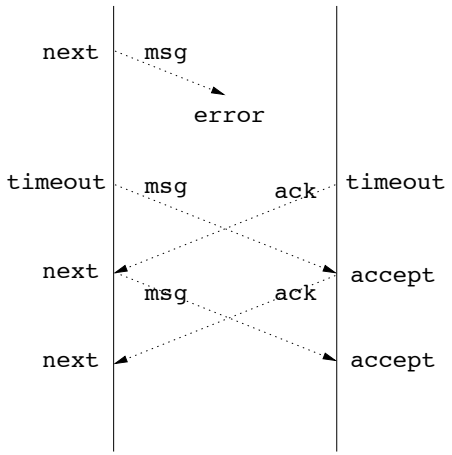
- The receiver acknowledges each message, the sender knows the buffer size of the receiver
- Problem: lost, inserted reordered, duplicated messages



Timeout

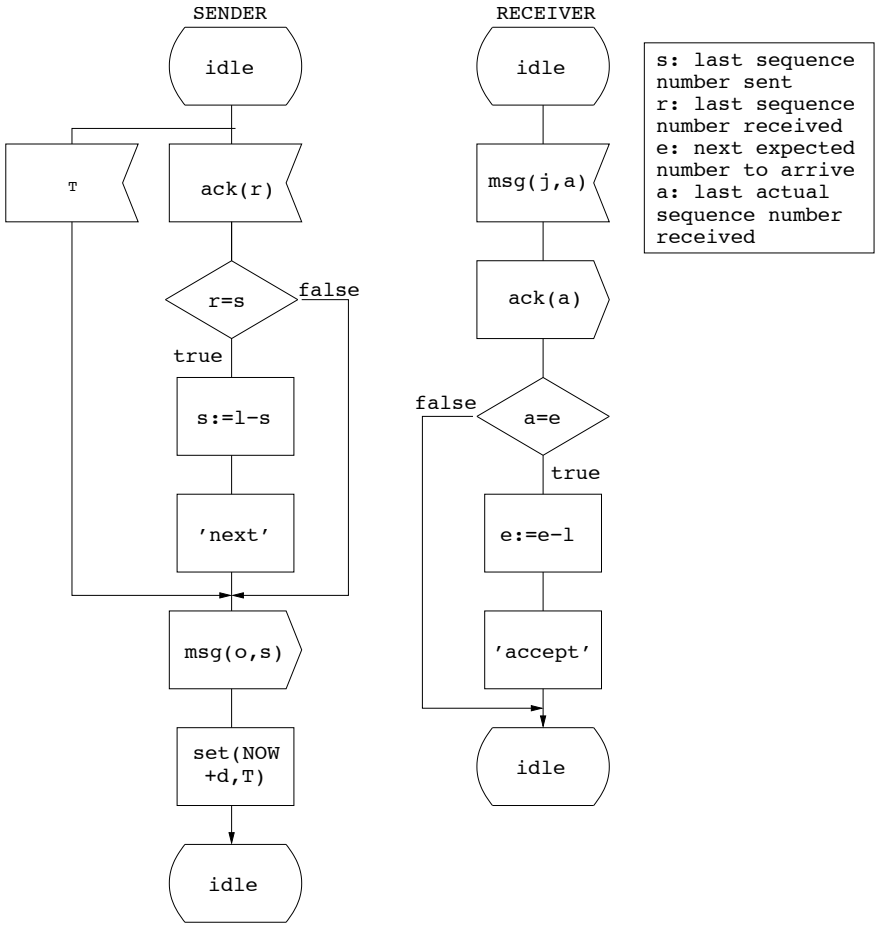
- Protection against message loss
- Only one of the processes may initiate the retransmission





Sequence numbering

- Provides retransmission of lost messages



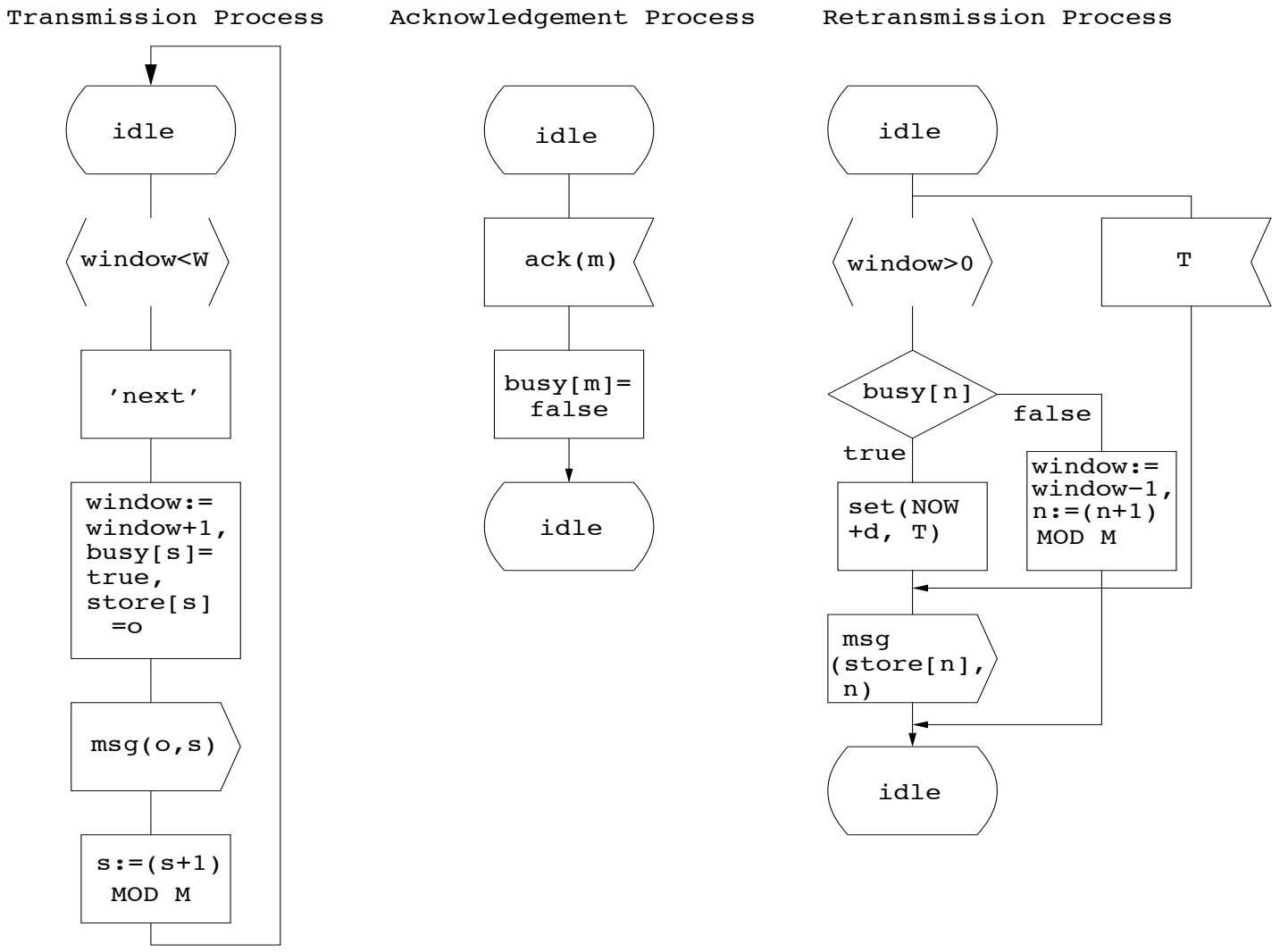
Sliding window

- Provides the restransmission of lost messages
- Elements:
  - Larger sequence numbers, total M
  - Window size: W <= M/2
  - Sent but not yet ackd messages are stored on the sender side
  - Received but not yet accepted messages are stored on the receiver side
  - Receiver side buffer

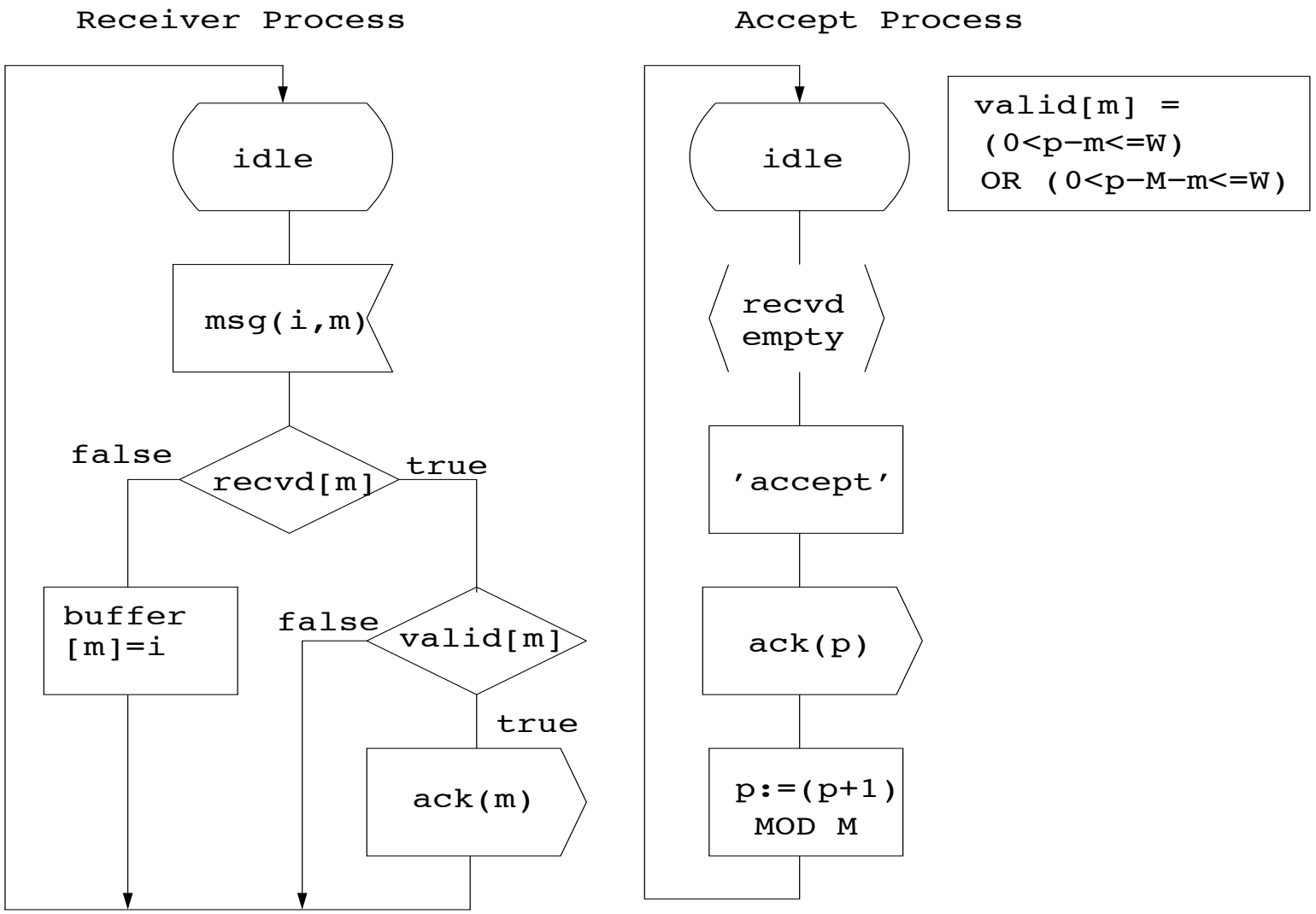
Sliding window

- The window:
  - Next message to be sent (s)
  - Number of unacknowledged messages (window)
  - Oldest unacknowledged message (n)
  - Latest unacknowledged message (m)
  - Next message to accept (p)

Sliding window

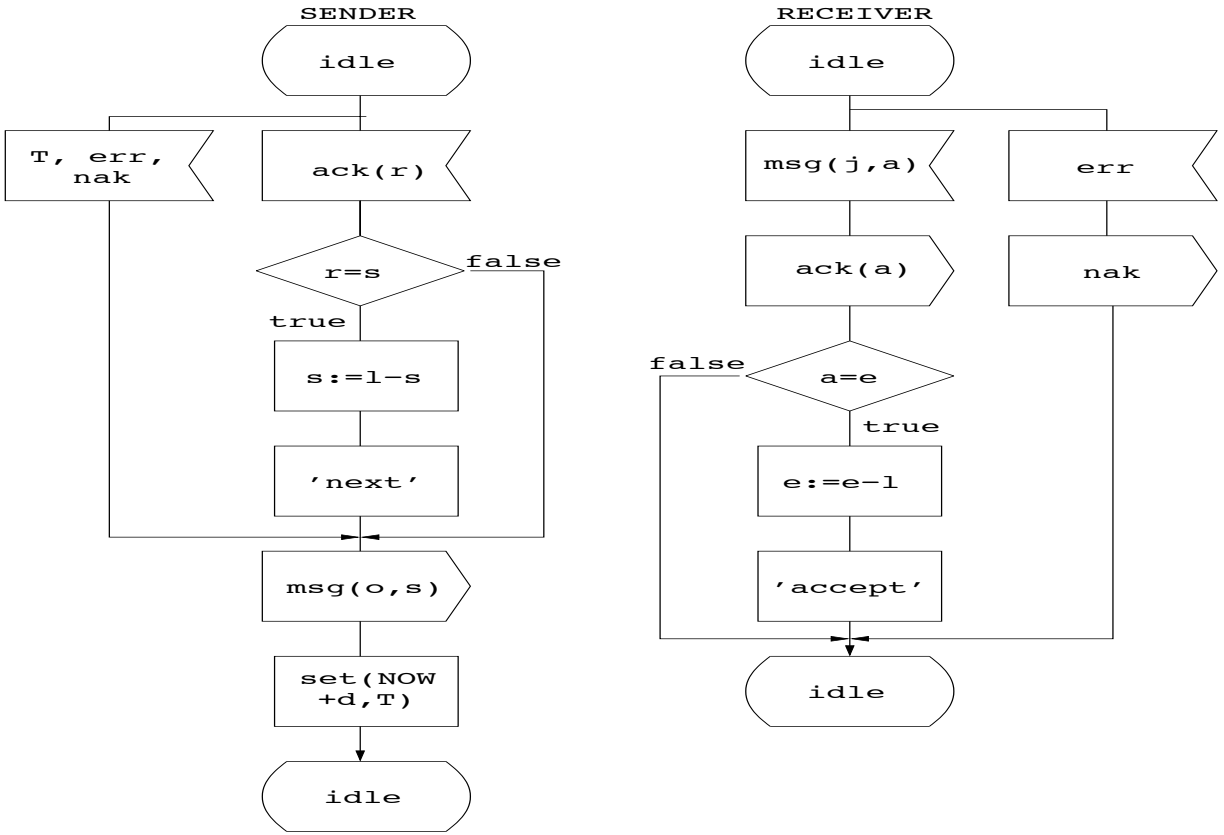


Sliding window

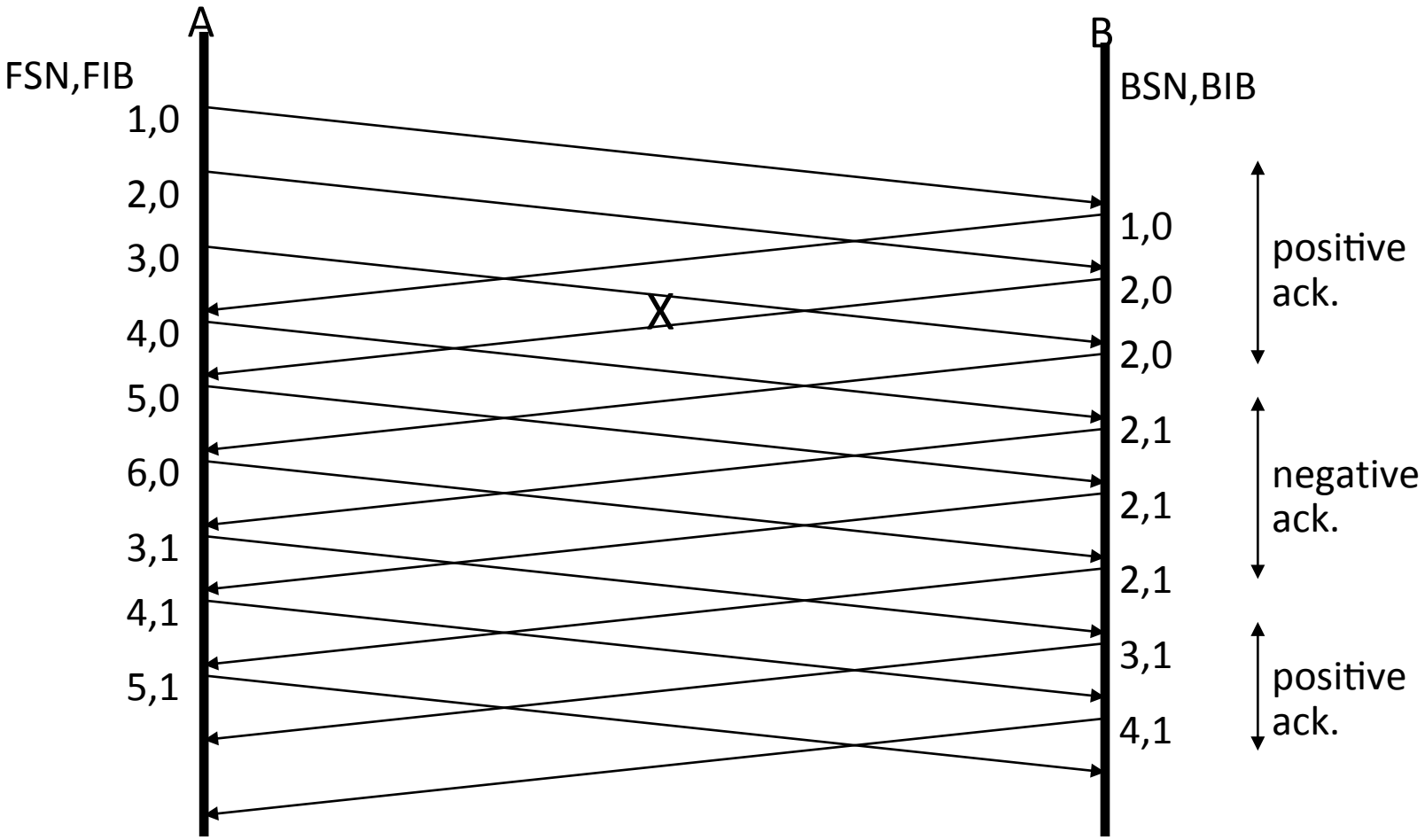


Negative acknowledgement

- The receiver forces the retransmission of damaged messages instead of sender timeouts
- Block acknowledgement



Example


A
B
FSN,FIB
BSN,BIB
1,0
2,0
3,0
4,0
5,0
6,0
3,1
4,1
5,1
X
1,0
2,0
2,0
2,1
2,1
2,1
3,1
4,1
positive ack.
negative ack.
positive ack.

Policy

- Rules for participating in communication sessions
- Must be agreed upon before the communication takes place
- Policy consists of:
  - Subject
  - Domain
  - Rules, assertions
    - Required, optional, observed etc.
  - Decision points
- Examples:
  - In networking: SNMP
  - In software: Java Security Policy or WS-Policy