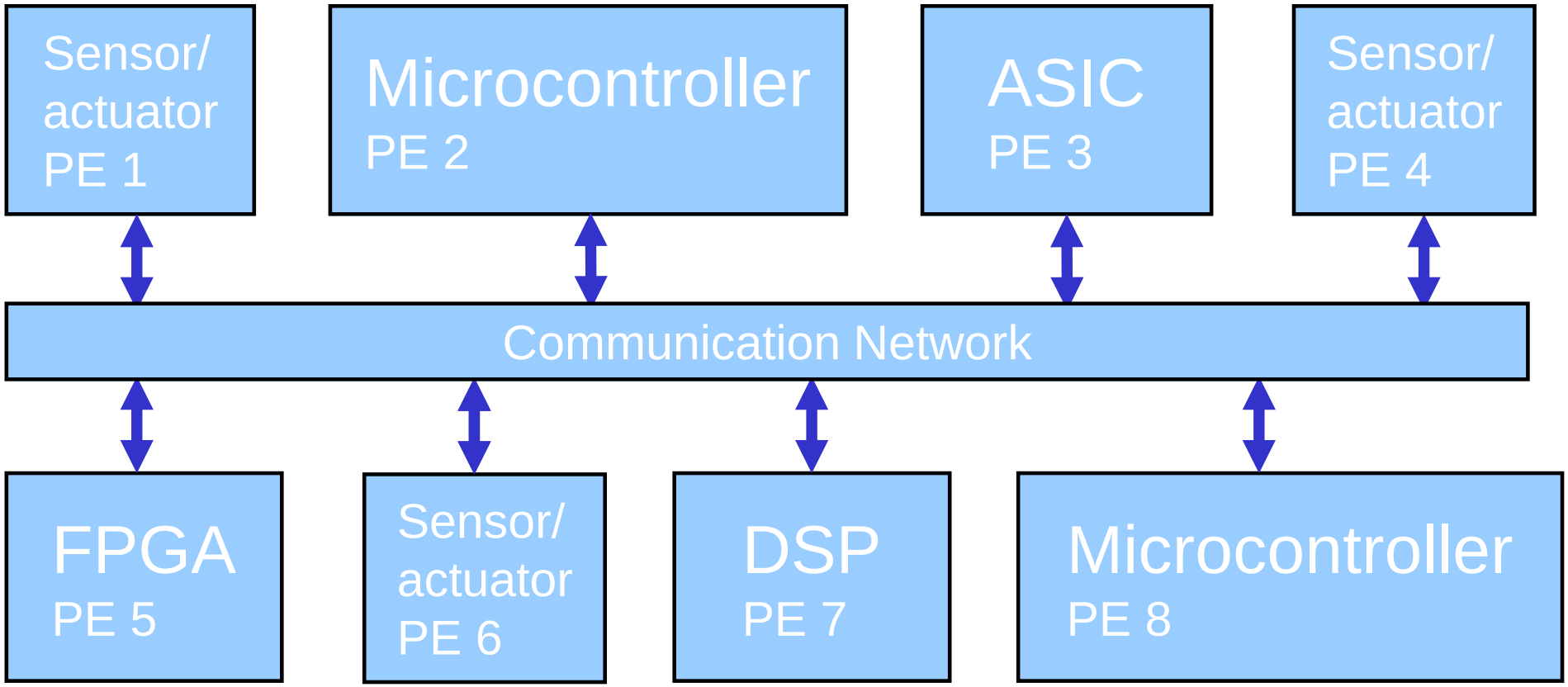
Architecture of Digital Systems II

6 Networks and Distributed Systems

Distributed Embedded Systems

Many embedded applications today require a distributed system architecture: the individual components of the embedded system, called processing elements, are connected over a communication network. Example:



Why Distributed?

There are several reasons for designing a system *distributed*.

1. Increase the computational power of the system:

Instead of having a single CPU executing all computations, the tasks can be spread across several processing units. For example, a DSP performs the time-critical high-throughput numerical computations, and a general-purpose microprocessor takes care of the remaining less computation-intensive system tasks.

2. Devices may be physically separated:

In some applications (e.g., automotive), the processing elements and the devices they are communicating with are physically separated. The processing is distributed if some of the computation power must be put next to some sensor or actuator (to save communication time or bandwidth), while the remaining tasks are performed somewhere else. (Example: engine control).

Why Distributed? (cont'd)

3. Modular design:

Network-based architectures support modular design because they allow for clean component interfaces. When designing systems using existing components (that may themselves include embedded processors, memory, I/O devices) the components are usually integrated into the system using standard network interfaces.

4. Fault tolerance:

Fault tolerance can be built into systems by adding hardware redundancy and software redundancy of some kind and connecting the redundant processing elements over a network.

5. Testing and Diagnosis:

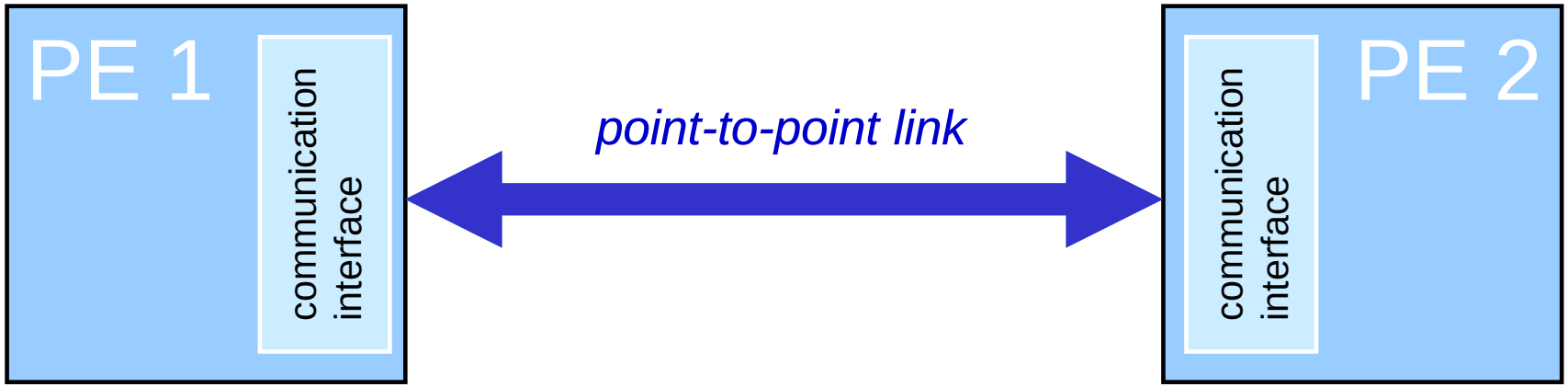
Network-based architectures can make testing and diagnosis easier. For example, one part of the system can be used to generate stimuli and evaluate responses for another part of the system.

Network types

For distributed embedded systems, many network architectures exist. We categorize into three types:

1. Point-to-point communication:

This type of link connects exactly two processing elements. (Easy to design, no communication conflicts).

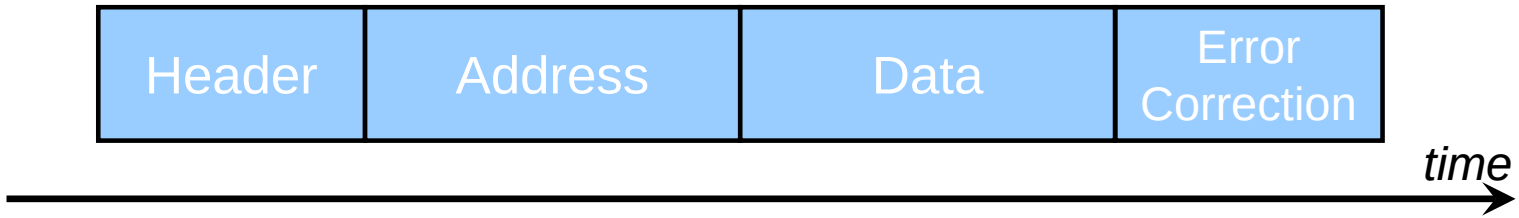


Connections can be simplex (only in one direction), duplex (in both directions) or half-duplex (both directions, but not simultaneously; i.e., sending and receiving are interleaved).

(Network types)

2. Buses:

Buses allow more than two processing elements to communicate. Communication on the bus usually means sending and receiving data packets. A packet typically includes an address for the destination and the data to be transmitted. Often, the packet format also includes a header field containing information for bus arbitration and a field for error detection/correction information.



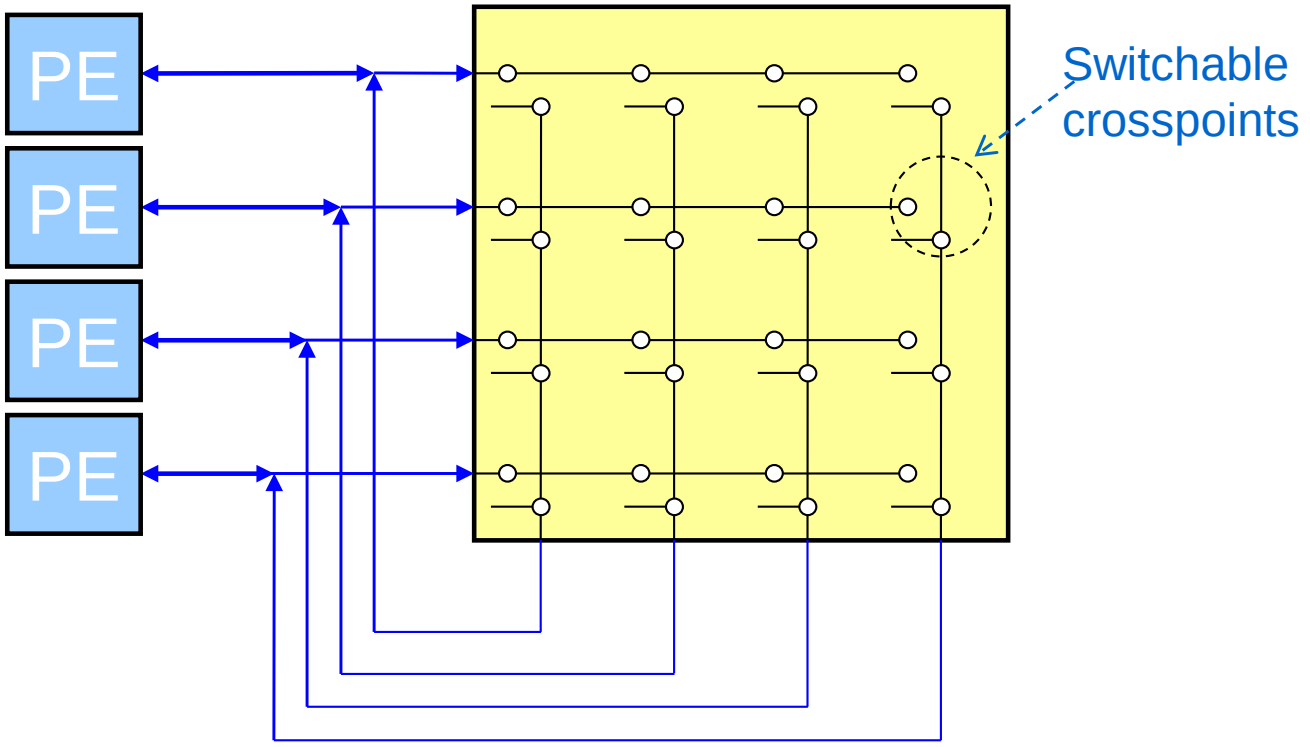
A variety of bus arbitration schemes exist, including fixed-priority arbitration and fair arbitration schemes (like round-robin).

(Network types)

3. Switched networks:

Buses have limited bandwidth, and communication conflicts occur.
More general network topologies exist helping to reduce conflicts.
The highest bandwidth with no communication conflicts is provided by an N x N crossbar, connecting N senders and N receivers:

Example:
4x4 crossbar



Major drawback: expensive. Size grows quadratically with inputs.

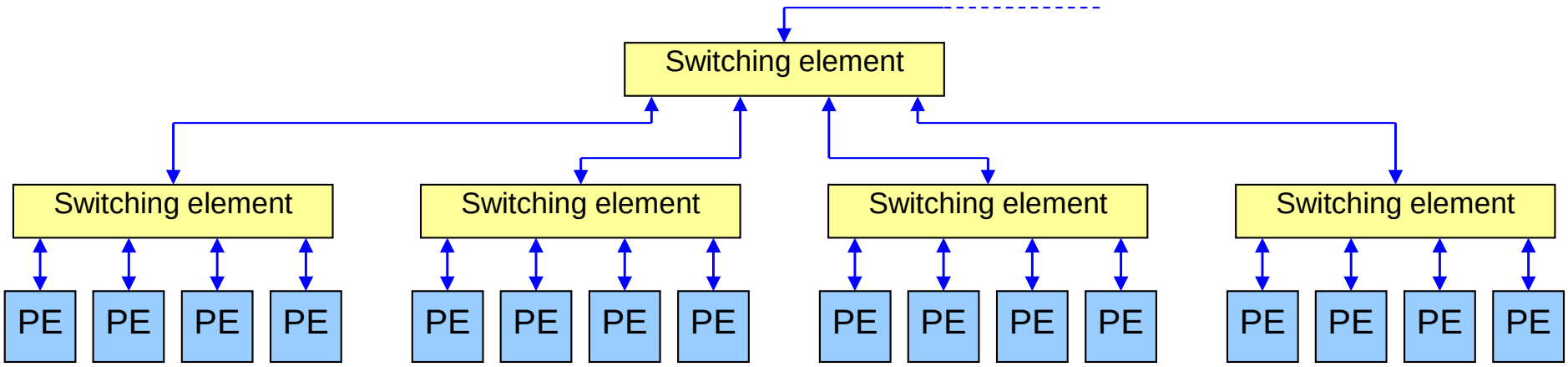
(Network types)

(3. Switched networks)

If a large number of processing elements are to be connected while several simultaneous communications are required then switched networks are used. Compared to a crossbar, this may reduce the number of possible simultaneous connections (increasing conflicts) but it also reduces cost.

The switching elements may contain storage to buffer data packets during conflicts.

Example: A tree-like switched network



Abstract Network Modeling

Communicating over networks is a complex process. There are many aspects that need to be considered, e.g.,

- how to pack data fragments into packets
- how to address the recipient of the data
- how to route the data packet from sender to receiver
- how to represent data packets on the physical bus signals
- how to check for and correct errors
- how to deal with communication conflicts
- ...

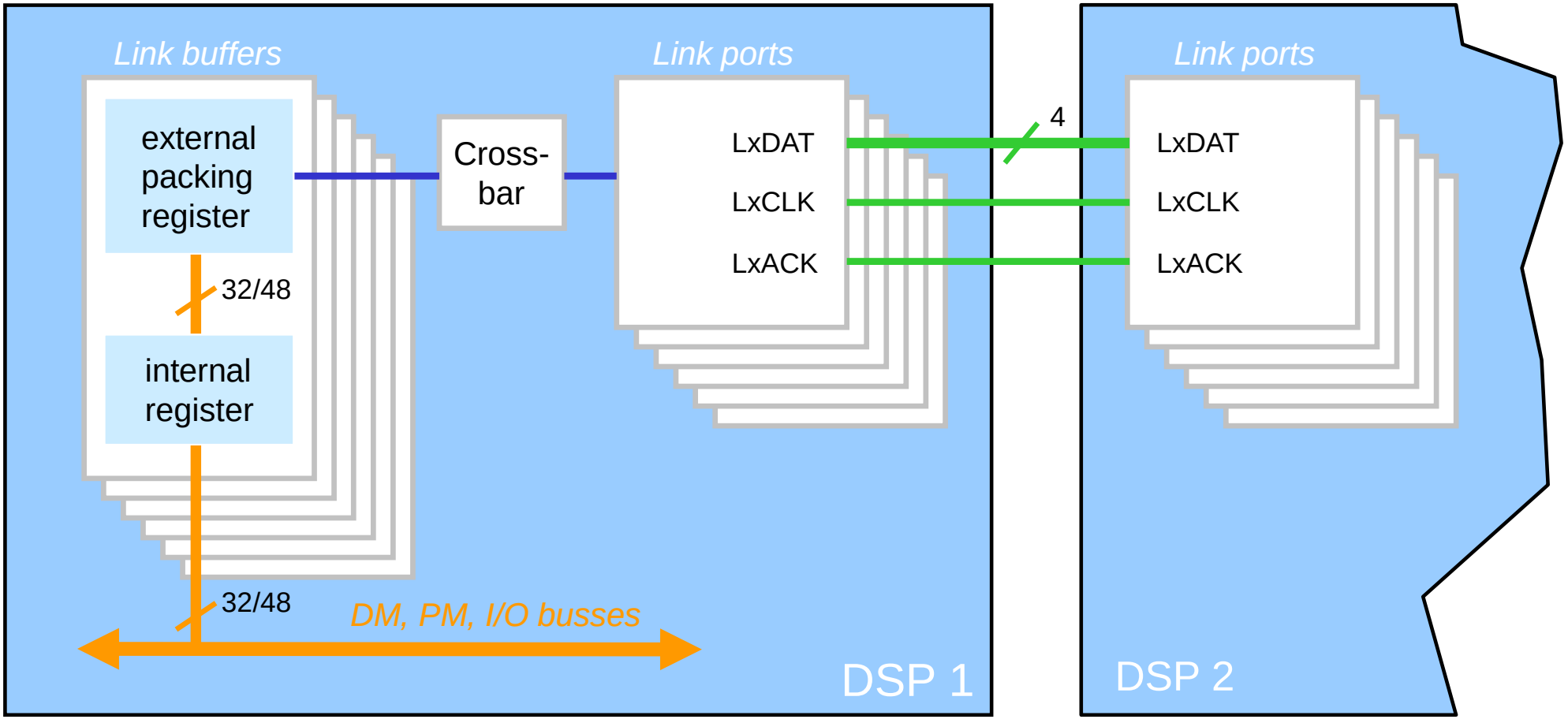
In order to structure these tasks, abstract hierarchical network communication models have been devised. The International Standards Organization (ISO) has developed a seven-layer network model known as the Open Systems Interconnection (OSI) Basic Reference model.

ISO-OSI Seven-Layer Model

Application	Application interface between network and end-user programs
Presentation	Coding of characters, compression, encryption, data exchange formats
Session	End-user service interaction control; establishing and terminating connections
Transport	Connection-oriented services, ensuring that data are delivered in-order and error-free.
Network	Basic end-to-end transmission service; handles multi-hop transmissions
Data Link	Error detection and control across a single link (single hop)
Physical	Physical interface between systems, electrical properties, basic bit exchanging procedures

Network Example 1: SHARC Link Ports

As an example of point-to-point networking, consider the six *link ports* of the SHARC.



Each link port can send four bits *twice* per CPU clock cycle.

(Network Example 1: SHARC Link Ports)

A link port establishes a half-duplex connection. The port must be configured by control register bits to act as a transmitter or receiver. Typically, the communicating processors pass *tokens* between each other. The owner of the token is the sender, the other is the receiver. The tokens are stored in local memory. Care must be taken so that tokens get neither lost nor duplicated.

The link buffers are used for packing/unpacking of data into 4-bit units. The link buffers and link ports can be arbitrarily connected through a crossbar, based on configuration bits in a control register. Link buffers can be accessed over the DM, PM or I/O data busses and they can be targets of DMA transfers.

Networking Example 2: CAN

The CAN bus (Controller Area Network bus) finds widespread use in automotive applications. Developed by Bosch and Intel in 1983, was it originally designed for networking of automotive electronic components. However, today, CAN and some derivatives (e.g., CANopen) are used also in other areas like production automation.

Some properties:

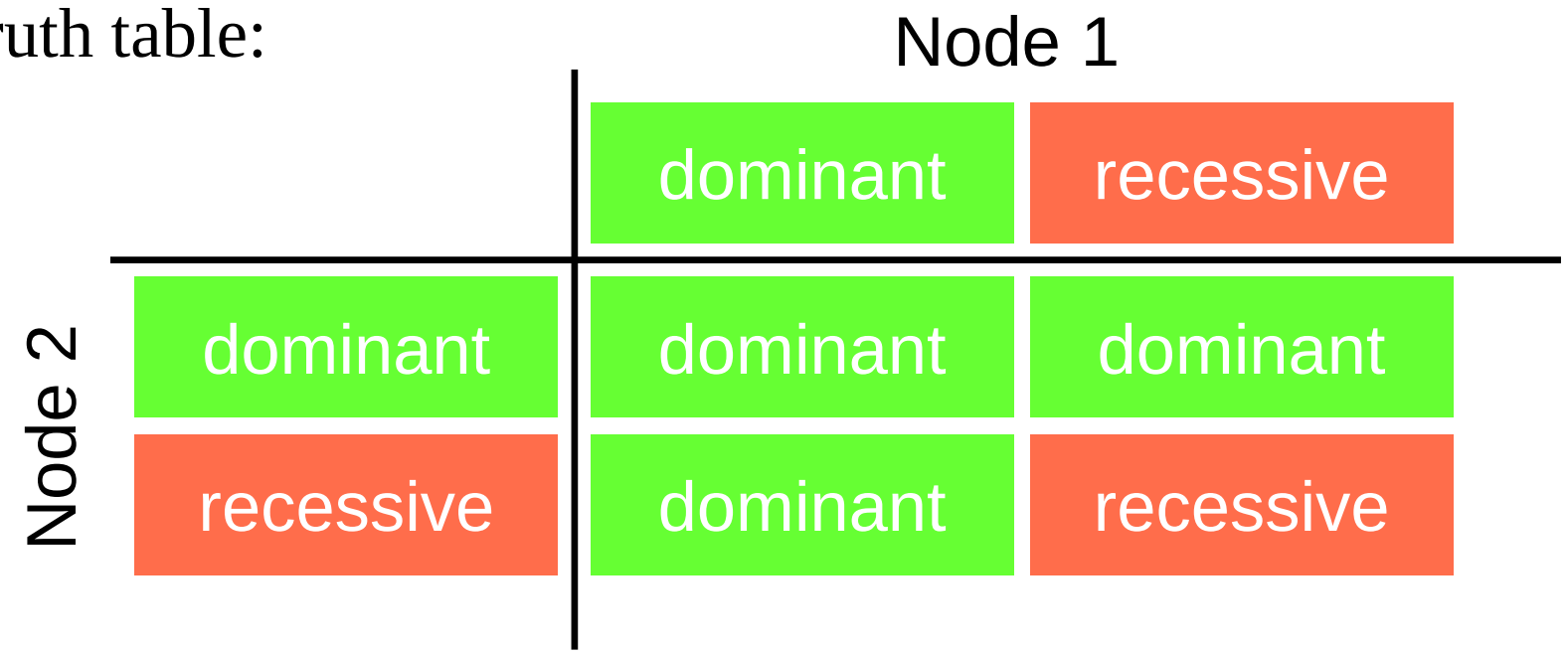
- asynchronous bus
- bit-serial transmission
- up to 1Mb/s over twisted-pair cable of up to 40m in length (or longer distances at slower rates, e.g., 125kb/s over 500m)
- supports broadcasts
- error-checking

Some microcontrollers have built-in support for CAN.

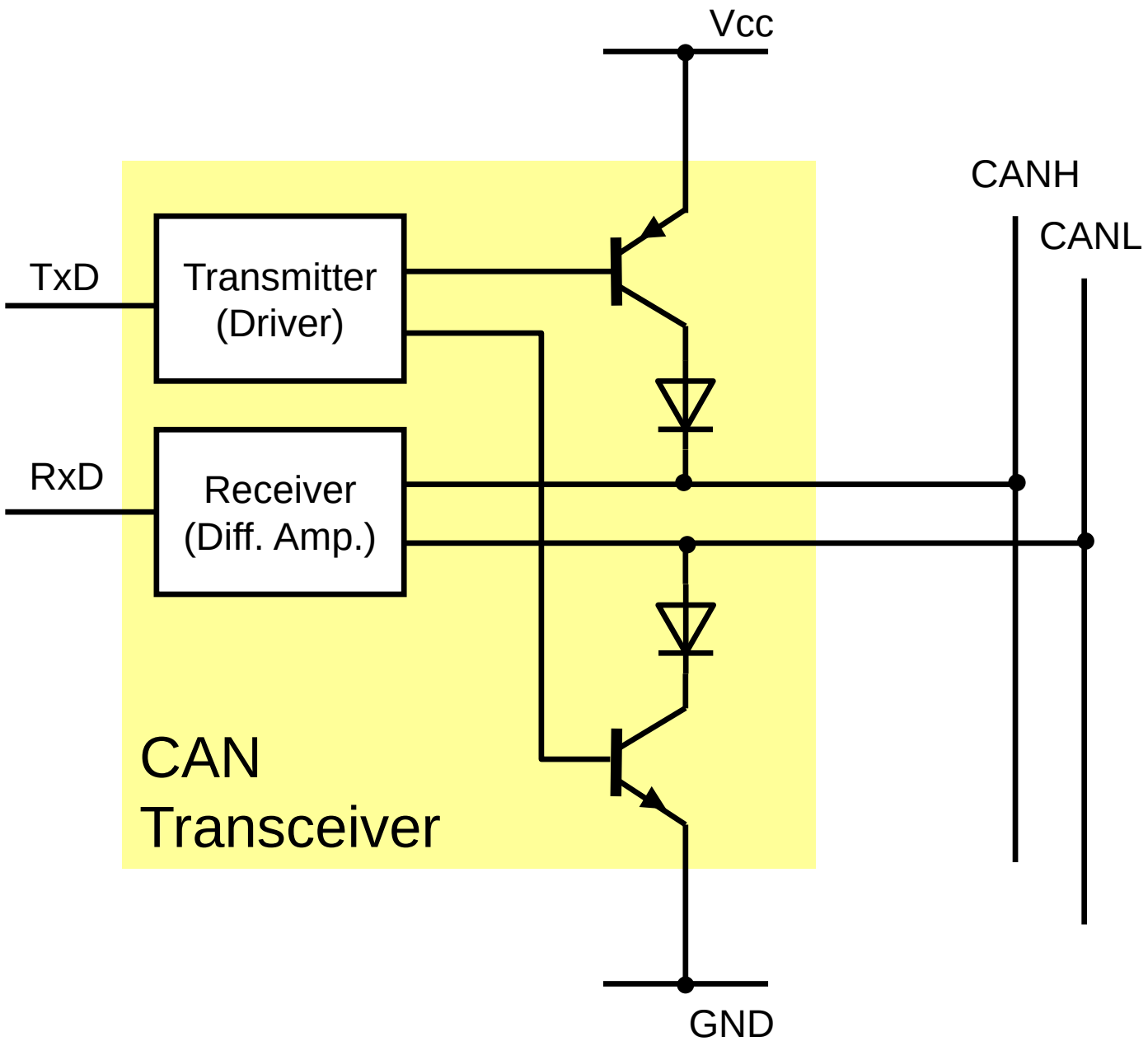
CAN: Physical Layer

Each node in a CAN bus has its own transceiver. The transceiver circuitry connects to the bus in a wired-AND fashion. A logical 1 is called a recessive bit, a logical 0 is called a dominant bit. A node transmitting a dominant bit can override another node transmitting at the same time a recessive bit. (We will see how this is used in arbitration, shortly.)

Truth table:



CAN Transceiver



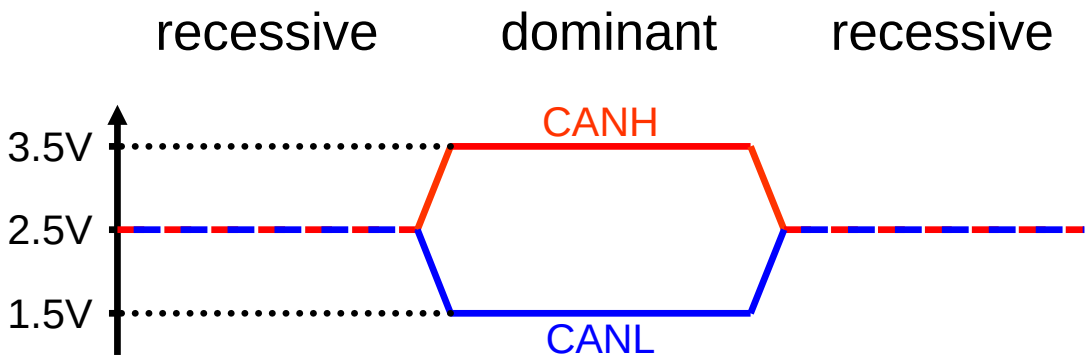
The logical state of the bus is given by the voltage difference on the two signals CANH and CANL. (This reduces effects of electro-magnetic interferences.)

(CAN Transceiver)

Electrical specifications:

	CANH	CANL
Dominant state	3,5V	1,5V
Recessive state	2,5V	2,5V

Example timing diagram:



Noise margins:

CANH – CANL > 0.9V : dominant state; logical zero

CANH – CANL < 0.5V : recessive state; logical one

CAN Packets

Data are sent over a CAN network in packets called *frames*. There are four types of frames:

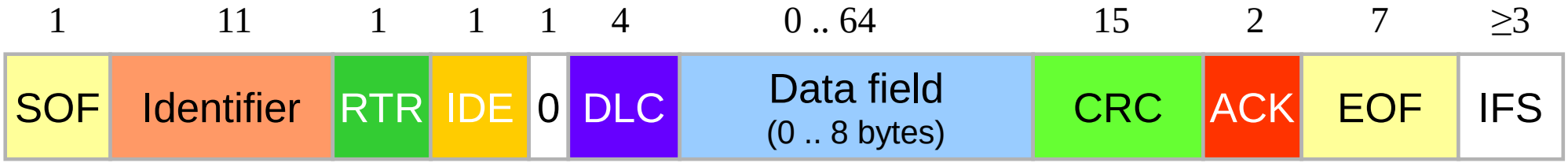
- *Data frame*
- *Remote frame*: a frame requesting the transmission of data as specified by the identifier
- *Error frame*, sent by any node detecting an error
- *Overload frame*: a frame to inject a delay between data and/or remote frames

The data frame is the only frame actually transmitting data. There are two types of data frames:

- *Standard Data Frame*: with an 11-bit identifier
- *Extended Data Frame*: with a 29-bit identifier

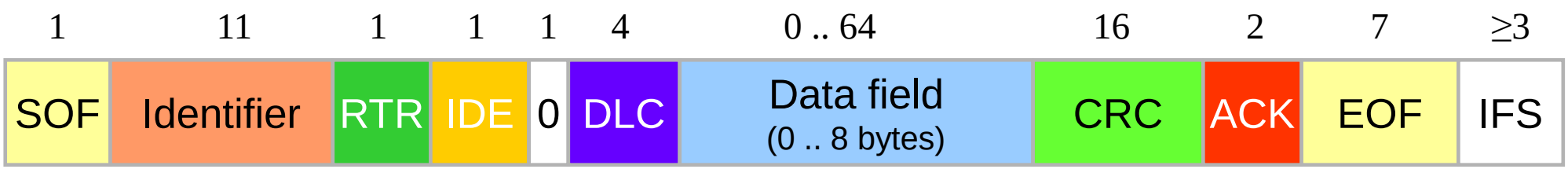
CAN: Standard Data Frame (1)

As an example of a CAN frame, we consider a standard data frame or standard remote frame. The frame consists of a number of fields explained below.



SOF	"Start of frame" (1 bit)
Identifier	An 11-bit address identifying *data objects* within the network (not necessarily CAN nodes!)
RTR	"Remote Transmission Request" (1 bit) If recessive (1): data is requested from the given identifier.
IDE	"Identifier extension" (1 bit) must be dominant (0) in a standard data frame

CAN: Standard Data Frame (2)

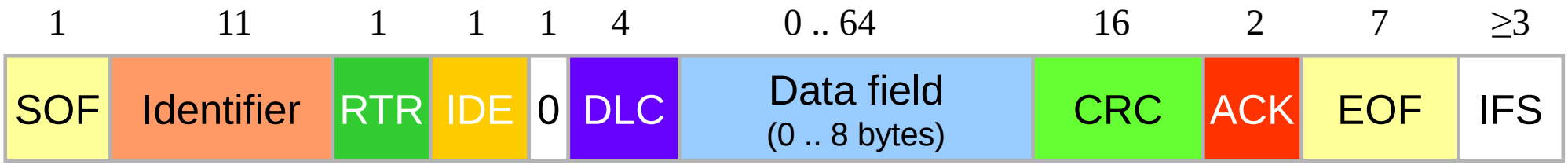


r0 "reserved bit" (1 bit) – must be dominant (0)

DLC "Data Length Code"
defines the number of bytes in the data field (0 .. 8)

CRC "Cyclic Redundancy Check" (16 bits)
consists of a 15-bit CRC checksum followed by a dominant CRC field delimiter.
The recipient of the message computes a CRC checksum on the fly during reception. If the received and the computed sums mismatch the recipient will signal an error.

CAN: Standard Data Frame (3)



ACK "Acknowledge Field"
consists of an Acknowledge Slot (1 bit) and an Acknowledge Delimiter (1 bit, recessive '1')
– if the recipient detects an error it drives the bus to a dominant value during the Acknowledge slot; the sender will then repeat the message. (Note that nodes permanently monitor the bus, even during their own transmissions.)

EOF "End of Frame" (7 bits, recessive)

IFS "Inter-Frame Spacing"
After a message, the bus must be recessive for at least 3 bit times.

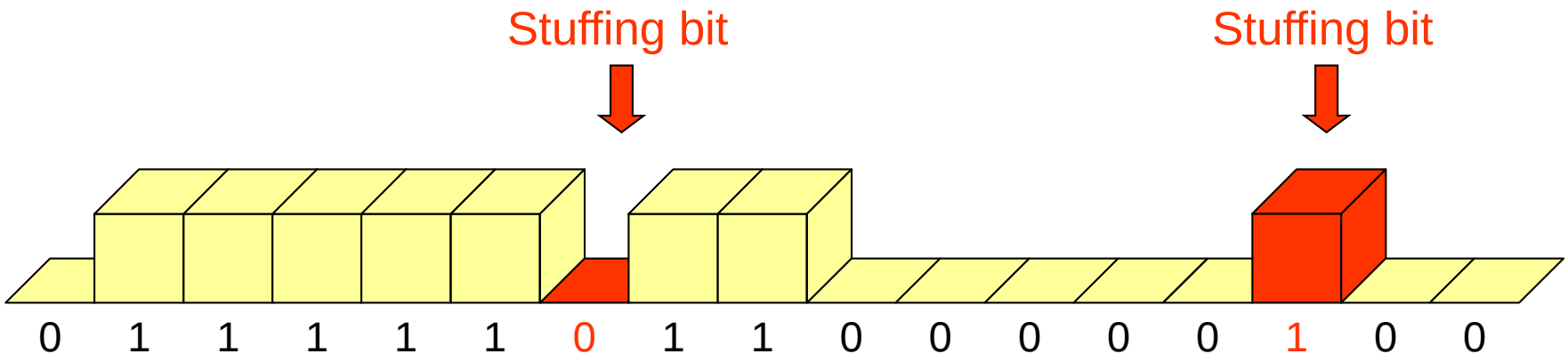
CAN: Node synchronization

CAN is a bit-serial asynchronous protocol – there is no clock line. Every node has its own local bit timing clock. All clocks run at the same rate, however, independently. A node synchronizes its local clock to the received bit stream by detecting rising and falling edges.

If the data packets sent over the network contain long sequences of zeros or ones then a receiver may lose synchrony with the bit stream.

CAN uses a technique called bit stuffing to provide frequent signal edges for synchronization:
After a sequence of 5 consecutive bits of same polarity, a so-called stuffing bit of inverse polarity is inserted into the bit stream. It carries no information but is merely used for synchronization.

CAN: Bit Stuffing Example



The example shows two stuffing bits inserted into the bit stream. The transmitted data excludes the stuffing bits – in this example, the transmitted bits are

0 1 1 1 1 1 1 1 0 0 0 0 0 0 0

CAN: Arbitration

The CAN protocol uses an arbitration method known as Carrier Sense Multiple Access (CSMA) with Bitwise Arbitration.

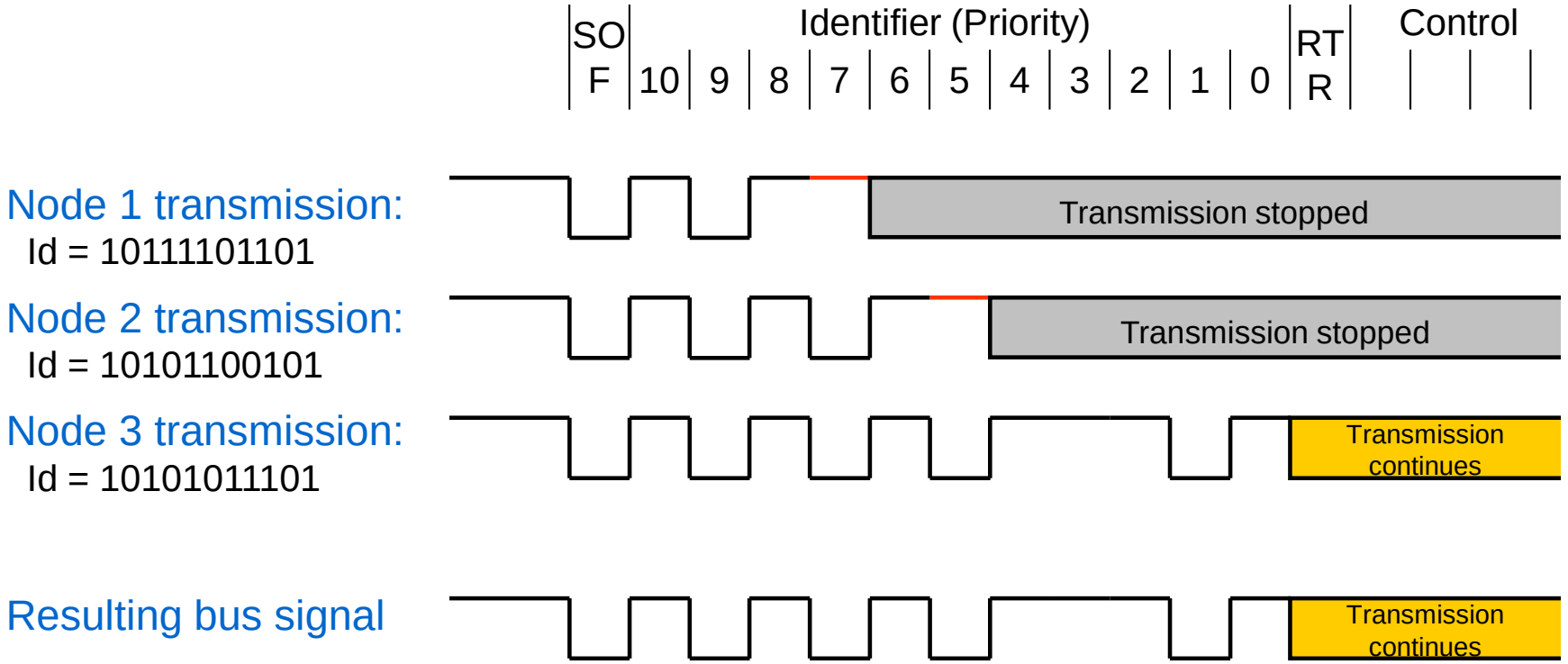
A node may only start a transmission if the bus is idle (i.e., in IFS field). However, network nodes may start a transmission at the same time. All transmitting nodes send their identifier simultaneously. Every node permanently monitors the bus signals. If a node detects a dominant bit while sending a recessive bit, it immediately stops transmitting. When the last bit of the identifier field is transmitted there is only a single transmitter left.

The identifier resembles a priority value. The lower the identifier value, the higher the priority. The all-0 identifier has the highest priority.

Once transmission of a data frame has started, it may not be interrupted by another data frame. (Non-preemptive arbitration.)

CAN Arbitration Example

In this example, three nodes start a transmission simultaneously by sending SOF and an Identifier (Id). Because the MSB of the Id is sent first, the Id directly corresponds to a priority. After the Id field, only node 3 will be left transmitting.



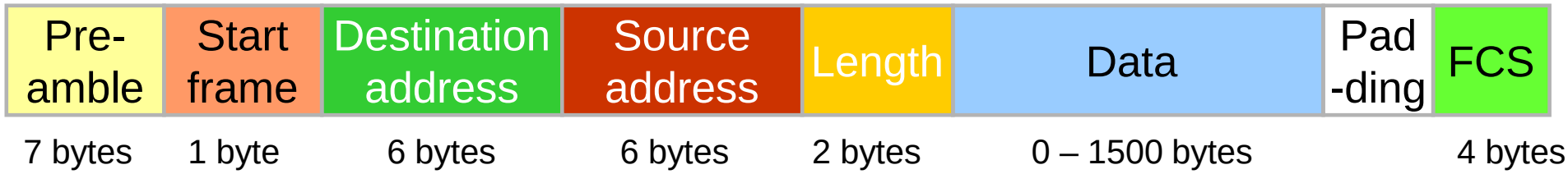
Network Example 3: Ethernet

Ethernet is used widely for Local Area Networks (LAN). Some embedded devices include an Ethernet network interface.
The physical implementation of Ethernet provides a single signal bus. (There are many variants, e.g., 10BASE-T, 100BASE-TX, 1000BASE-T or fiber optic standards).

Ethernet nodes are not synchronized – they may start sending any time if they see a silent bus. Ethernet uses an arbitration scheme known as Carrier Sense Multiple Access with Collision Detection (CSMA/CD).

The maximum length of an Ethernet network is determined by the time needed for collision detection and the minimum length of a packet. In practice, depending on the standard used, Ethernet connections can be up to several hundred meters long.

(Network Example 3): Ethernet Packet Format



The addresses of destination and source are 6-byte MAC addresses uniquely identifying the sending and receiving network nodes.

The length field defines the number of subsequent data bytes.

The "padding" field is used in a collision to extend the packet to the minimum packet length so that collisions can be safely detected by all bus nodes.

The FCS field ("frame check sequence") is a 32-bit CRC checksum.

Ethernet is defined on the physical and the data link layers of the OSI model.

Network-Based Design

Designing distributed embedded systems is a difficult task. Computation and communication must be scheduled and allocated to the respective resources. In low-cost networks, careful scheduling and allocation of the communication is especially important so that the network does not become the bottleneck of the system.

Before we can evaluate the performance of a network-based system we need to analyze the communication, i.e., determine the delays for transmitting messages. If we neglect bus contention or re-transmission due to transmission faults, the basic message delay can be modeled as

$$t_d = t_a + t_t + t_m + t_r$$

t_d	message delay
t_a	network availability delay
t_t	transmitter-side overhead
t_m	network transmission time
t_r	receiver-side overhead

(Network-Based Design)

The transmitter/receiver-side overhead can often be neglected against the other parameters. The *network availability delay* t_a is the time the transmitter needs to wait before the network becomes available so that the message can actually be sent. This time depends on the arbitration scheme used in the network.

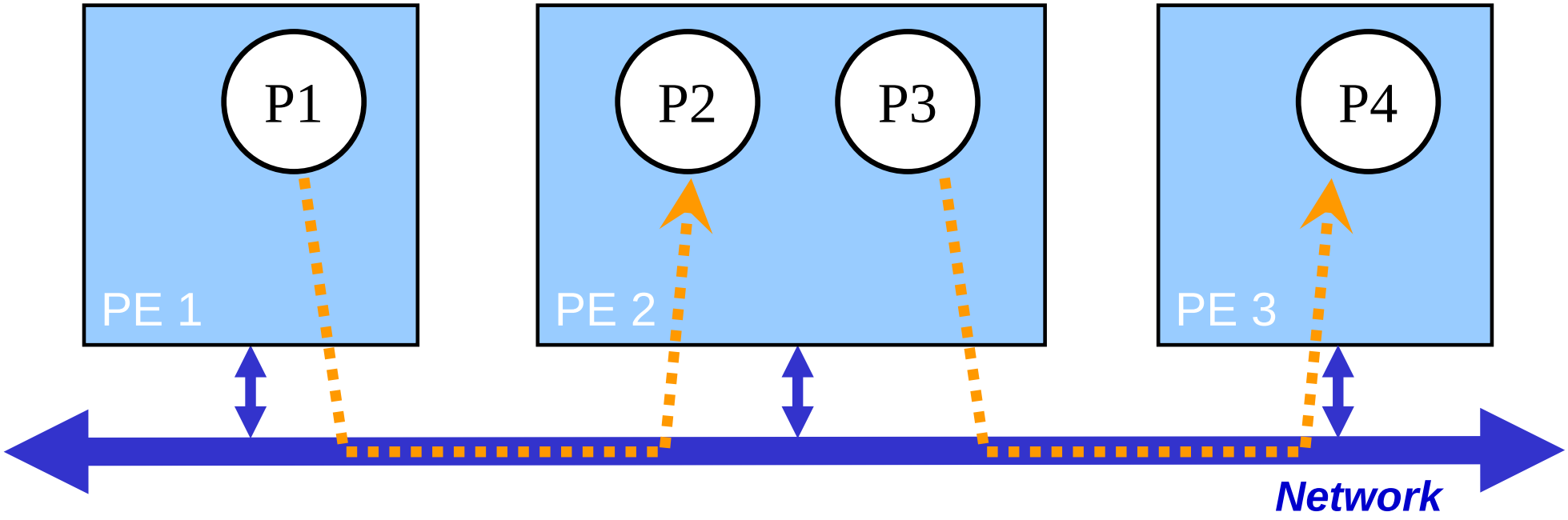
- In fixed-priority arbitration, t_a is unbounded except for the highest-priority network node.
- In fair arbitration (e.g., round-robin), t_a is bounded and depends on the number of devices on the bus.

In practice, network delay analysis is usually much more difficult due to additional factors such as

- data corruption
- message acknowledgement schemes
- multi-hop network designs
- interrelation of computation and communication

System Performance Analysis

Performance analysis of distributed embedded systems is difficult, because computations and communications interfere with each other. Consider the following example:



Four processes run on three processing elements (PEs). P2 needs data from P1, P4 needs data from P3. P2 and P3 run on the same node. The data is sent over the network.

(System Performance Analysis)

(Example continued)
The performance of this system is hard to analyze because of the following dependencies:

- Because P2 needs to wait for P1 to complete, P2's starting time "inherits" any variations in P1's execution time.
- Because P2 and P3 run on the same processor, P2's deadlines as well as its execution time affect the completion time of P3.
- Variations in the completion time of P3 are translated into the starting time of P4.
- Messages from P1 and P3 on the bus can interfere, causing even more variations in completion time.

Performance analysis of complex distributed systems is done using Computer-Aided Design tools that can give accurate lower and upper bounds on the start and completion times of processes.