TDDD25
Distributed Systems

Distributed Real-Time Systems:
Real-Time Communication

Christoph Kessler

IDA
Linköping University
Sweden

2024

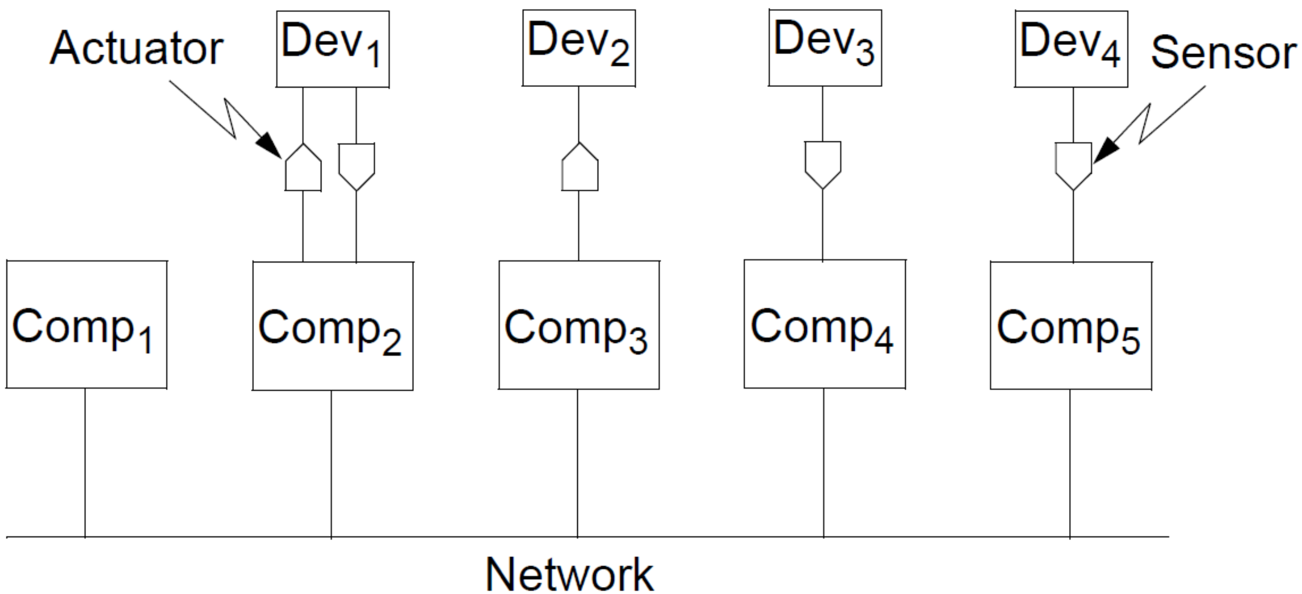
LINKÖPING UNIVERSITY

Agenda

DISTRIBUTED REAL-TIME SYSTEMS

1. What is a Real-Time System?
2. Distributed Real Time Systems
3. Predictability of Real-Time Systems
4. Process Scheduling
5. Static and Dynamic Scheduling
6. Clock Synchronization
7. Universal Time
8. Clock Synchronization Algorithms
9. Real-Time Communication
10. Protocols for Real-Time Communication

Real-Time Communication



- Data flows
  - from sensors and control panels to processors
  - between processors
  - from processors to actuators and displays
- In order to achieve predictability:
  hard real-time systems need communication protocols
  that allow for the communication overhead to be bounded.

Time/Event Triggered Communication

Time-triggered communication

- The sender and receiver agree on a cyclic, time-controlled, conflict-free communication schedule.

  Each message transmission is started at a certain, pre-defined, moment in time; conflicts are avoided per definition.

- Examples:
  - TDMA
  - FlexRay (static phase)

  } Predictable
  Appropriate for real-time

Time/Event Triggered Communication

Event-triggered communication

- Messages can be sent whenever a significant event happened at the sender (task terminated, interrupt, etc.).
  - No pre-defined moments in time for communication
  - Potential conflicts for bus access.
- Examples:
  - Ethernet – is *not* predictable;
  - CAN
  - Token ring
  - FlexRay (dynamic phase)

(CAN, Token ring, FlexRay (dynamic phase)): Predictable
Appropriate for real-time

Ethernet Protocol

Ethernet is a Carrier Sense Multiple Access/Collision Detection (CSMA/CD) protocol.

- On Ethernet, any device can try to send a frame at any time.
- Each device **senses** whether the line is idle and thus available to be used.
  - If it is, the device begins to transmit.
- If two or more devices try to send at the same time, a **collision** occurs and the frames are discarded.
  - Each device then **waits** for a random amount of time and **retries** until successful in getting its transmission sent.

- Ethernet is inherently stochastic. It **cannot provide a known upper bound on transmission time**.
  - → Ethernet is not suitable for real-time applications.

- The above is true for the original "vintage" Ethernet.
  - New, Ethernet based solutions have been proposed (e.g. full-duplex switched Ethernet, that avoids collision) which provide support for predictability and could be applied for real-time communication.

Protocols for Real-Time Communication

- **CAN protocol**
- **Token Ring**
- **TDMA protocol**
- **FlexRay protocol**

- TDMA is mostly suitable for applications with regular data flow (constant rate). It is the most reliable and predictable.
- The CAN protocol provides a higher degree of flexibility in the case of irregular flow.
- FlexRay is a heterogeneous time- and event-triggered protocol
  - potentially combines advantages of TDMA and CAN

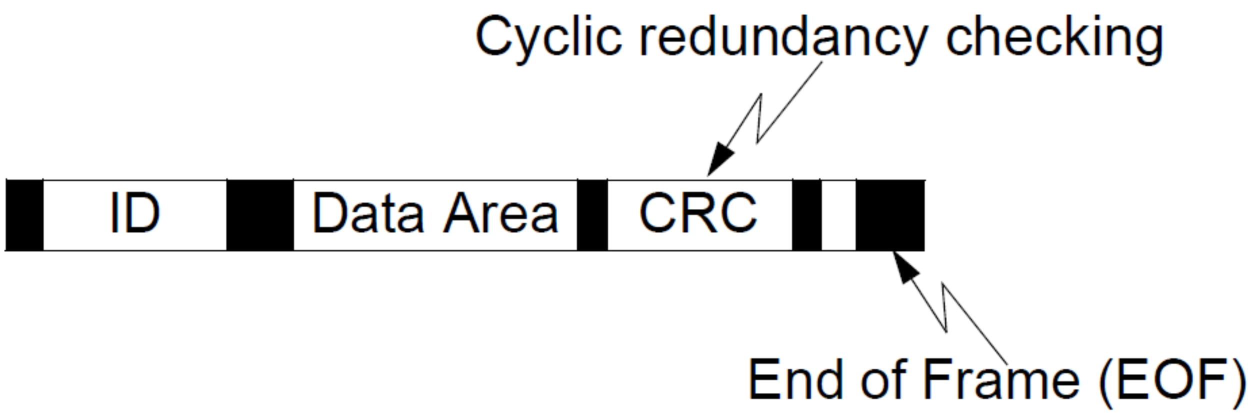
CAN Protocol

CAN = Control Area Network

- CAN is a Carrier Sense Multiple Access/Collision **Avoidance** (CSMA/CA) protocol.
- CAN is widely used in **automotive** applications.
- In the CAN protocol, collisions are avoided by arbitration based on **priorities** assigned to messages.
- CAN communication is based on the transfer of packages of data called **frames**.

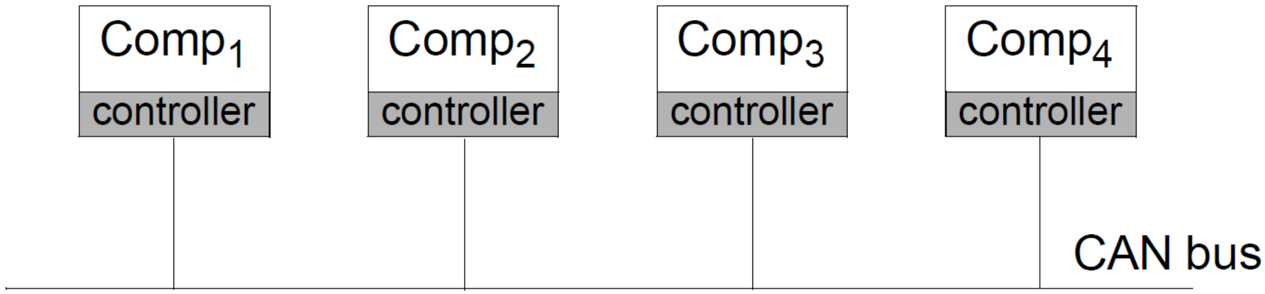
CAN Protocol

A **CAN frame**:



- The **identifier** (**ID**) field is used for two purposes:
  1. To distinguish between different frames.
  2. To assign relative priorities to the frames

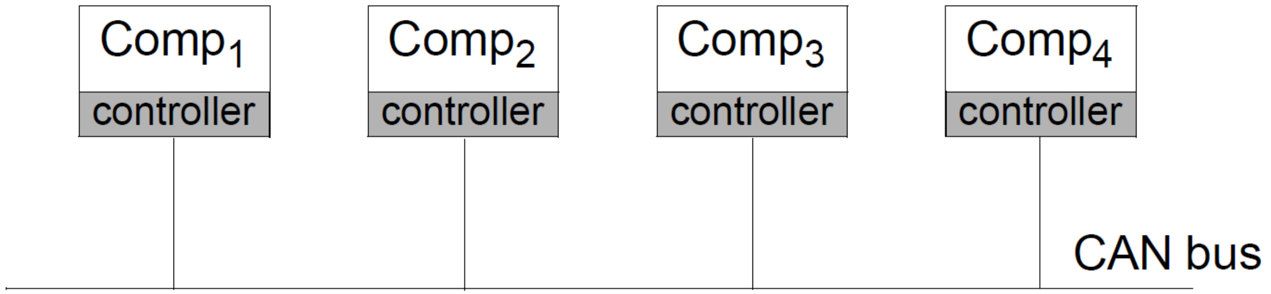
CAN Protocol



- A **CAN controller** is attached to each processor in the system. It ensures that:
  1. The **highest priority frame** (smallest identifier) waiting to be transmitted from the respective processor is entering the arbitration for the bus.
  2. The **arbitration** procedure, performed in cooperation by the controllers, guarantees access to the message with highest priority.

    - The **arbitration** is based on the existence of a **dominant** and a **recessive bit**: 0 is the dominant, 1 is the recessive:
      - If several controllers transmit and **at least one** transmits 0, the bus is at 0;
      - if **all** controllers write 1, the bus is at 1.

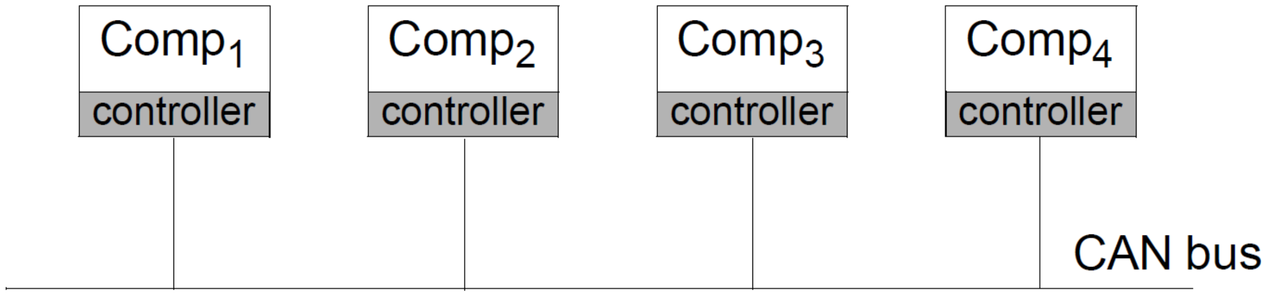
CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - -	EOF

- During **arbitration**, controllers **write** the **frame ID** to the bus, **bit by bit**, **as long as they read from the bus the same value they have written**.
  - Once a controller reads different,
    it continues by writing 1s until it reads **EOF** from the bus.
  - After the **EOF**, nodes which were unsuccessful **retry** with the same frames.

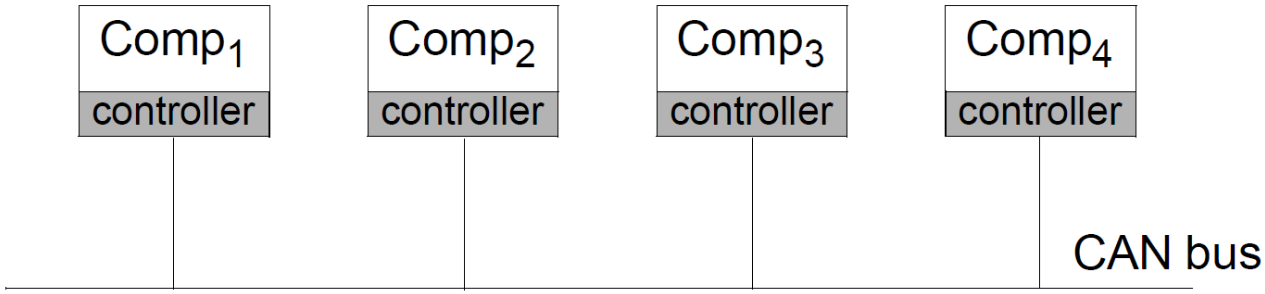
CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - - -	EOF

node 1	0	
node 2	0	
node 3	0	
node 4	0	
CAN bus	0	

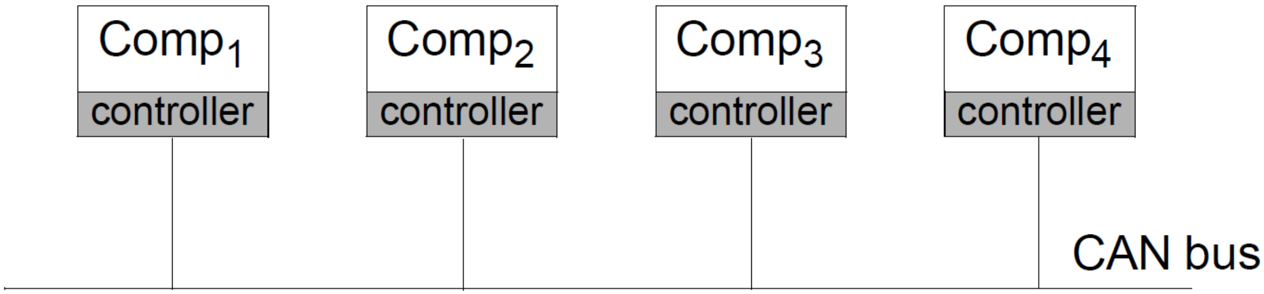
CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - - -	EOF

node 1	0	0	
node 2	0	0	
node 3	0	0	
node 4	0	0	
CAN bus	0	0	

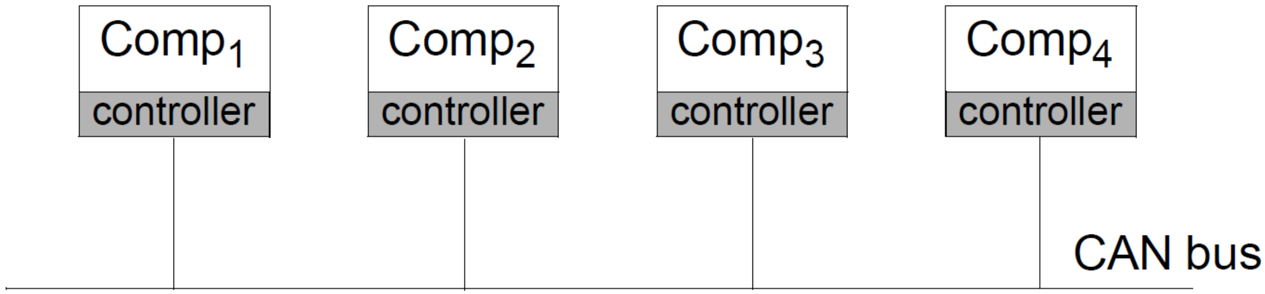
CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - - -	EOF

node 1	0	0	0
node 2	0	0	1
node 3	0	0	0
node 4	0	0	0
CAN bus	0	0	0

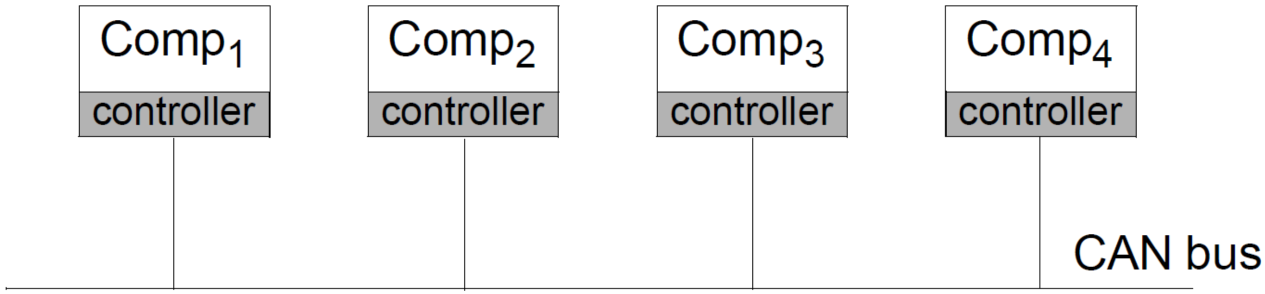
CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - - -	EOF

node 1	0	0	0	0
node 2	0	0	1	1
node 3	0	0	0	0
node 4	0	0	0	0
CAN bus	0	0	0	0

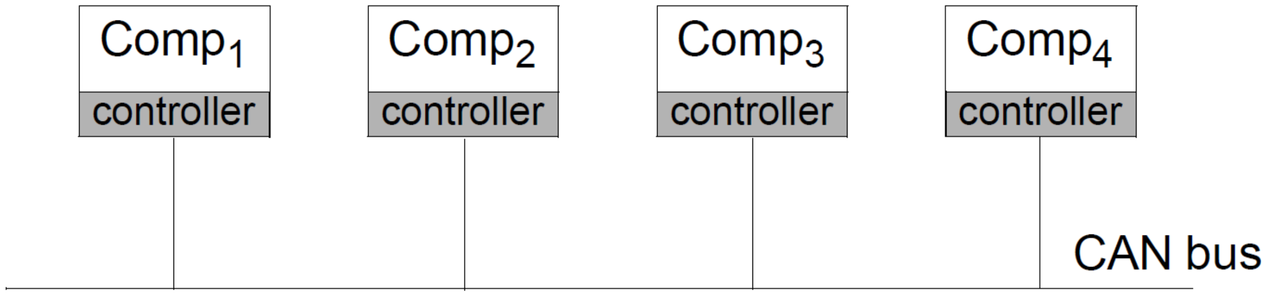
CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - -	EOF

node 1	0	0	0	0	1
node 2	0	0	1	1	1
node 3	0	0	0	0	1
node 4	0	0	0	0	1
CAN bus	0	0	0	0	1

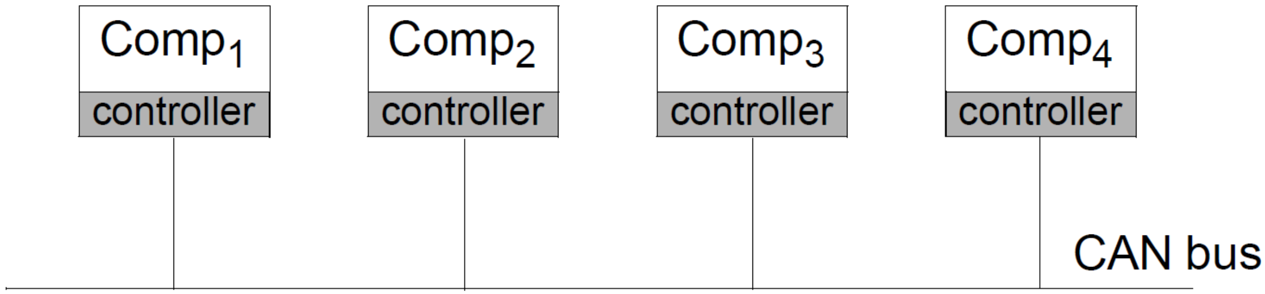
CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - - -	EOF

node 1	0	0	0	0	1	1
node 2	0	0	1	1	1	1
node 3	0	0	0	0	1	0
node 4	0	0	0	0	1	0
CAN bus	0	0	0	0	1	0

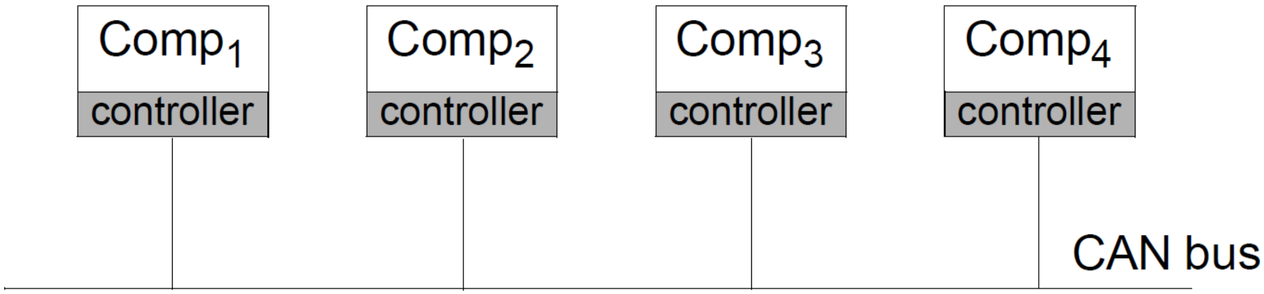
CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - - -	EOF

node 1	0	0	0	0	1	1	1
node 2	0	0	1	1	1	1	1
node 3	0	0	0	0	1	0	1
node 4	0	0	0	0	1	0	1
CAN bus	0	0	0	0	1	0	1

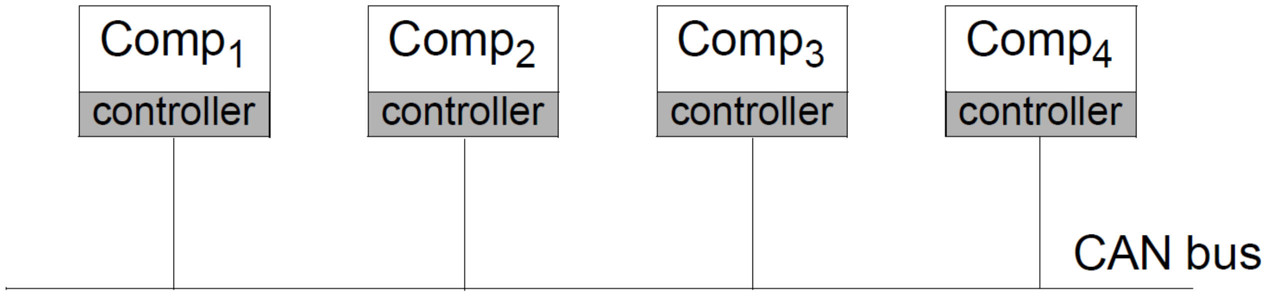
CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - - -	EOF

node 1	0	0	0	0	1	1	1	1
node 2	0	0	1	1	1	1	1	1
node 3	0	0	0	0	1	0	1	0
node 4	0	0	0	0	1	0	1	0
CAN bus	0	0	0	0	1	0	1	0

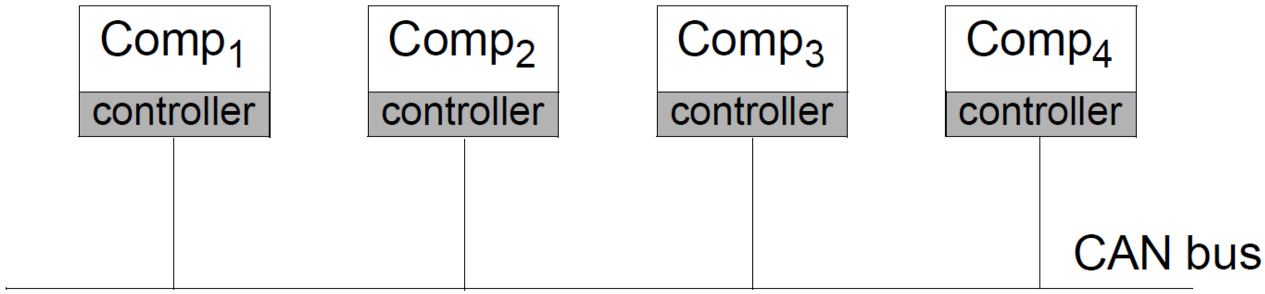
CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - - -	EOF

node 1	0	0	0	0	1	1	1	1	1
node 2	0	0	1	1	1	1	1	1	1
node 3	0	0	0	0	1	0	1	0	1
node 4	0	0	0	0	1	0	1	0	1
CAN bus	0	0	0	0	1	0	1	0	1

CAN Protocol



Frame node_1: 00001100011 | - - - - - - - - - - - - - - - - | EOF

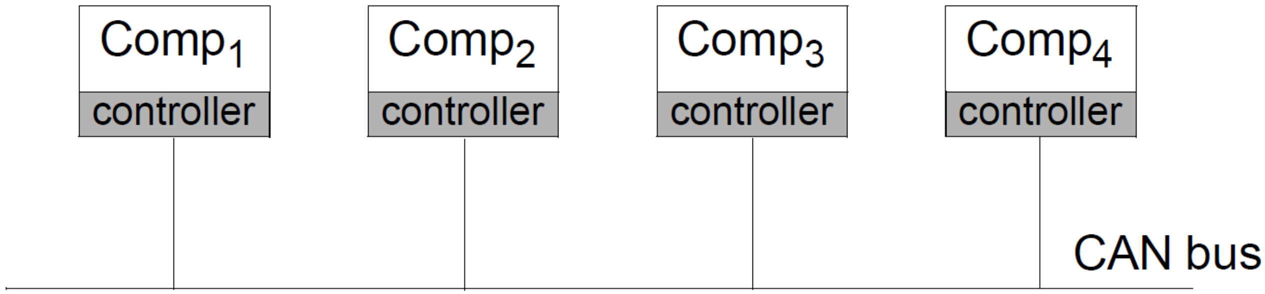
Frame node_2: 00100010101 | - - - - - - - - - - - - - - - - | EOF

Frame node_3: 00001010101 | - - - - - - - - - - - - - - - - | EOF

Frame node_4: 00001010111 | - - - - - - - - - - - - - - - - | EOF

node 1	0	0	0	0	1	1	1	1	1	1
node 2	0	0	1	1	1	1	1	1	1	1
node 3	0	0	0	0	1	0	1	0	1	0
node 4	0	0	0	0	1	0	1	0	1	1
CAN bus	0	0	0	0	1	0	1	0	1	0

CAN Protocol

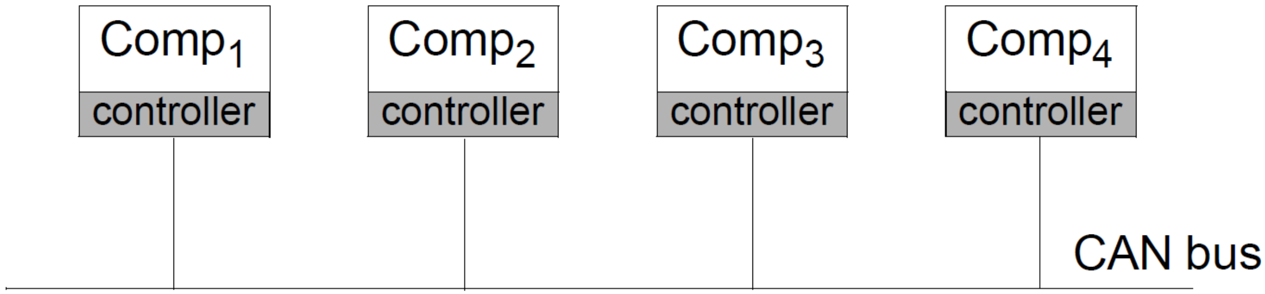


Frame node_1:	00001100011	- - - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - - -	EOF

node 1	0	0	0	0	1	1	1	1	1	1	1
node 2	0	0	1	1	1	1	1	1	1	1	1
node 3	0	0	0	0	1	0	1	0	1	0	1
node 4	0	0	0	0	1	0	1	0	1	1	1
CAN bus	0	0	0	0	1	0	1	0	1	0	1

ID-based arbitration:
Frame node_3 has priority

CAN Protocol



Frame node_1:	00001100011	- - - - - - - - - - - - - - - - -	EOF
Frame node_2:	00100010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_3:	00001010101	- - - - - - - - - - - - - - - - -	EOF
Frame node_4:	00001010111	- - - - - - - - - - - - - - - - -	EOF

node 1	0	0	0	0	1	1	1	1	1	1	1	1 1 1 1 1 1 1 . . .	. 1 1
node 2	0	0	1	1	1	1	1	1	1	1	1	1 1 1 1 1 1 1 . . .	. 1 1
node 3	0	0	0	0	1	0	1	0	1	0	1	0 1 0 0 1 1 0 . . .	EOF
node 4	0	0	0	0	1	0	1	0	1	1	1	1 1 1 1 1 1 1 . . .	. 1 1
CAN bus	0	0	0	0	1	0	1	0	1	0	1	0 1 0 0 1 1 0 . . .	EOF

CAN Protocol

If the following assumptions are fulfilled, message communication times can be **bounded** using techniques similar to those developed for priority based process scheduling:

- Messages are generated periodically, and the **period** is known.
- The **maximum size** of each frame is known.
- The **software overhead** connected to handling of messages is known.

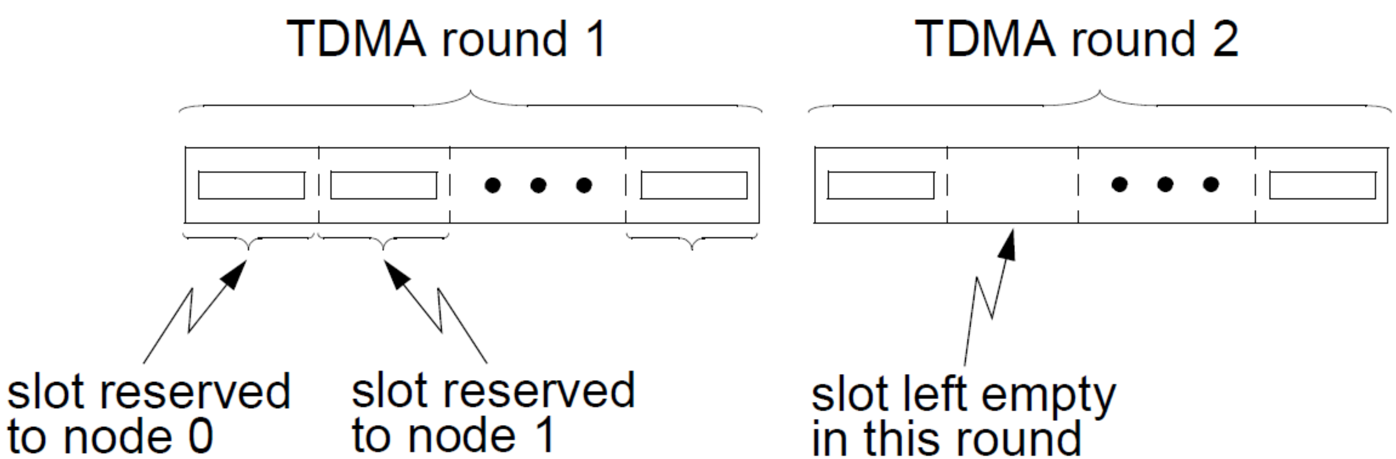
Token Ring

- Nodes are logically organized in a ring,
  on which the right to communicate is continuously passed.
- With a token ring protocol, **maximum bounds on message delay** can be established.
  The following are the essential parameters:
  - The **hold time** T_h:
    the longest time a node needs for communicating one message.
    - Can be derived from communication speed on bus and the maximum bound on the message length.
  - The **full rotation time** T_r:
    the longest time needed for a full rotation over all nodes.
    - $T_r = k*T_h$, where k is the number of nodes.
- Fault tolerance can be a problem:
  if one node fails, the traffic is disrupted.

TDMA Protocol

TDMA = Time Division Multiple Access

- For a system of *N* nodes, the total channel (bus) capacity is statically divided into *N* **slots**. Each slot is assigned to a certain node.
- The sequence of *N* slots is called a **TDMA round**.
  - One node can send one frame in a TDMA round.
  - The frame is placed into the slot assigned to that processor.
  - If no frame is to be sent by a node,
    the slot will stay empty in that round.
- The duration of one TDMA round is the **TDMA period**.



TDMA Protocol

- TDMA practically means a **static partitioning of access time** to the bus.
  - Each node knows in advance when and for how long it is allowed to access the communication line.
- TDMA implies the availability of a **global physical time** base (**clock synchronisation**).
- **Collisions are avoided** as nodes know when they have guaranteed exclusive access to the line.
- Message passing delay is **bounded**.

☺ High degree of predictability

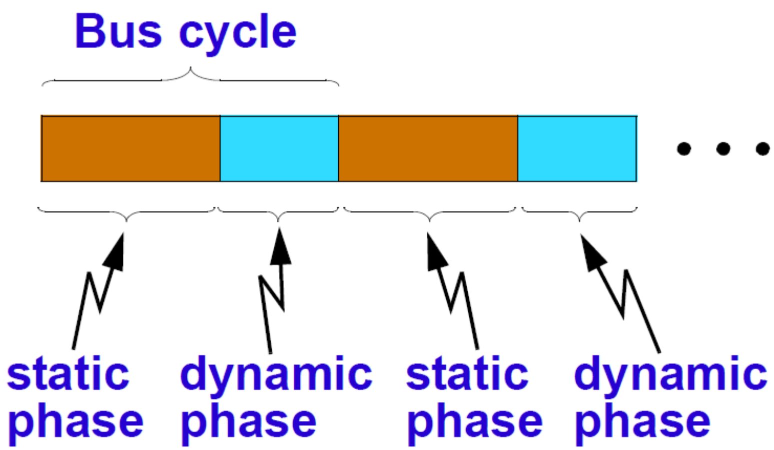
☺ Well-suited for safety-critical applications.

☹ Can lead to poor utilisation of bus bandwidth (e.g., empty slots).

☹ Low degree of flexibility → problems with irregular flows.

Partly resolved by **Dynamic TDMA**, a TDMA variant that dynamically assigns a variable number of time slots to nodes in each frame, based on the current traffic demand of each node.

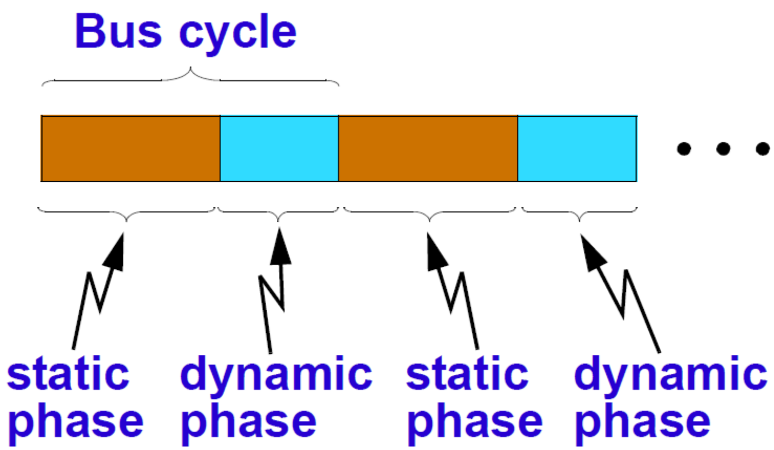
FlexRay
A Heterogeneous Communication Protocol



- FlexRay combines two protocols:
  an event-triggered and a time-triggered.
  - It combines some of the advantages of the two approaches.

- FlexRay has been designed by the "FlexRay consortium" for automotive applications.

FlexRay
A Heterogeneous Communication Protocol



The FlexRay bus cycle is divided into **two phases**

(the length of each phase is fixed by the designer):

- **Static phase**
  - During the static phase the bus works according to a **TDMA** policy
    - The static phase consists of slots assigned to nodes.
- **Dynamic phase**
  - During the dynamic phase, the bus works according to an **event-triggered** protocol, somewhat similar to CAN.

Combining two predictable approaches → FlexRay is **suitable for real-time**.

Some Usage Domains (Selection)

- **Token ring**
  - launched as alternative to Ethernet by IBM 1985, no longer in use
- **TDMA**
  - GSM (2G), DECT, Powerline
- **Dynamic TDMA**
  - Bluetooth
- **CAN**
  - Automotive domain (CAN development started by Robert Bosch AG 1983): on-board networks connecting sensors and ECUs, e.g. for lane-assist, collision avoidance, brake-by-wire, ...
  - Field bus in industrial automation
- **Flexray**
  - Automotive domain, e.g. on-board networks in high-end cars (mostly in Europe)
  - Faster and more reliable but more expensive than CAN
  - For non-safety-critical automotive communication, alternative networks could be considered.

Acknowledgments

- Most of the slide contents is based on a previous version by Petru Eles, IDA, Linköping University.