TDDD07 – Real-time Systems

Lecture 5: Distributed Systems II

Simin Nadjm-Tehrani

Real-time Systems Laboratory
Department of Computer and information Science

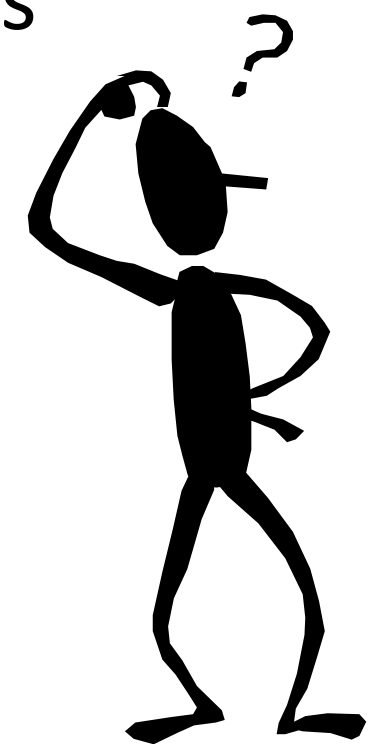
LINKÖPING UNIVERSITY

Overview: Next three lectures

From one CPU to networked CPUs:

- First, from one CPU to multiple CPUs
  - Allocating VMs on multiple CPUs: Cloud
- Next, fully distributed systems
  - fundamental issues with timing and order of events
- Next, hard real-time communication
  - Guaranteed message delivery within a deadline, bandwidth as a resource
- Finally: QoS guarantees instead of timing guarantees, focus on soft RT

Recap from last lecture: Logical clocks



- e is concurrent with g
- g is concurrent with f
- but e is not concurrent with f!

- Comparing the LC values does not tell us if two events are concurrent in the sense of $\prec$
- Vector clocks do more…

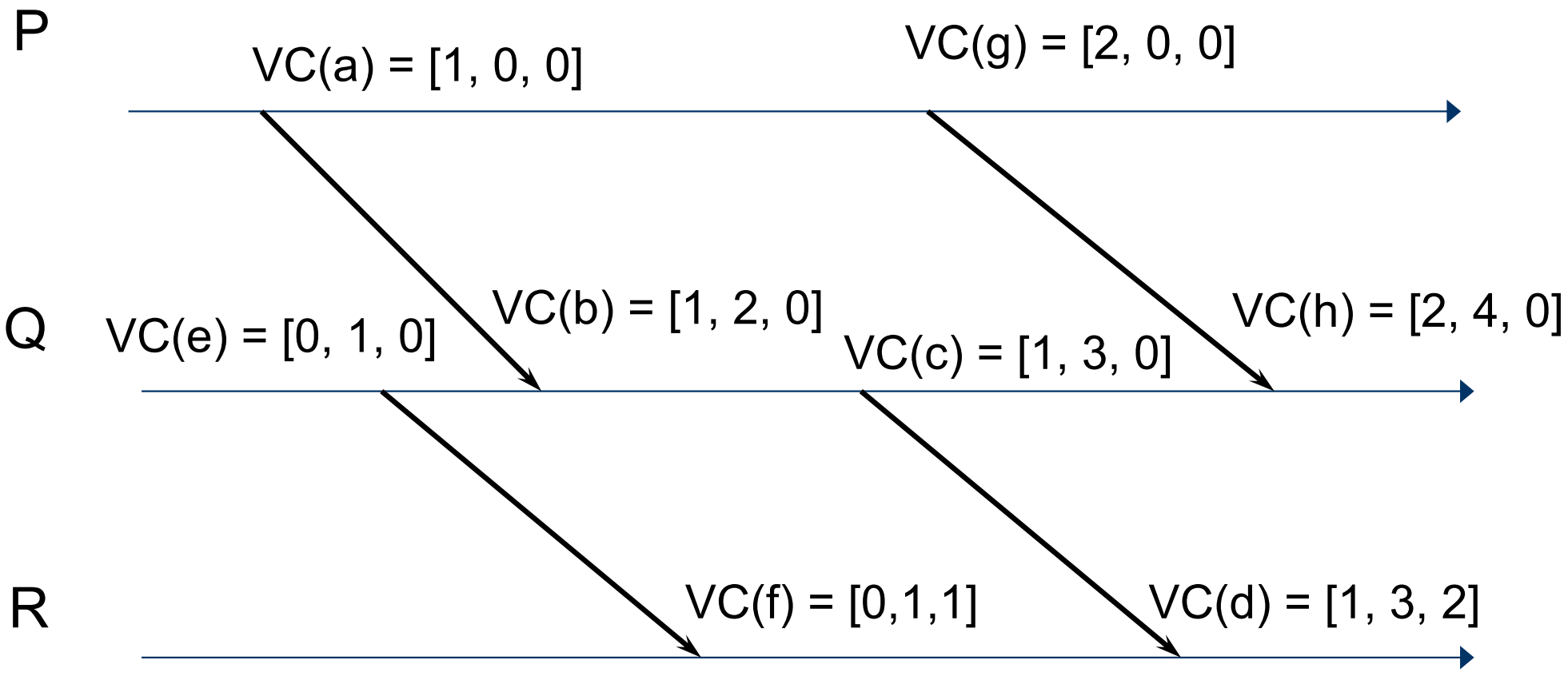
Vector clocks (VC)

- Every node maintains a vector of counted events (one entry for each other node)
- VC for event e, VC(e) = [1,…,n], shows the perceived count of events at nodes 1,…,n
- VC(e)[k] denotes the entry for node k

Implementation of VC

- Rule 1: For each local event increment own entry
- Rule 2: When sending message m, append to m the VC(send(m)) as a timestamp T
- Rule 3: When event x is “receiving a message” at node i,
  - increment own entry: VC(x)[i]:= VC(x)[i]+1
  - For every entry j in the VC: Set the entry to max (T[j], VC(x)[j])

Example (1) revisited with vector clocks



Precedence in VC

- Relation < on vector clocks defined by:

  VC(x) < VC(y) iff
  - For all i: VC(x)[i] ≤ VC(y)[i]
  - For some i: VC(x)[i] < VC(y)[i]

- It follows that event x $\prec$ event y if

  VC(x) < VC(y)

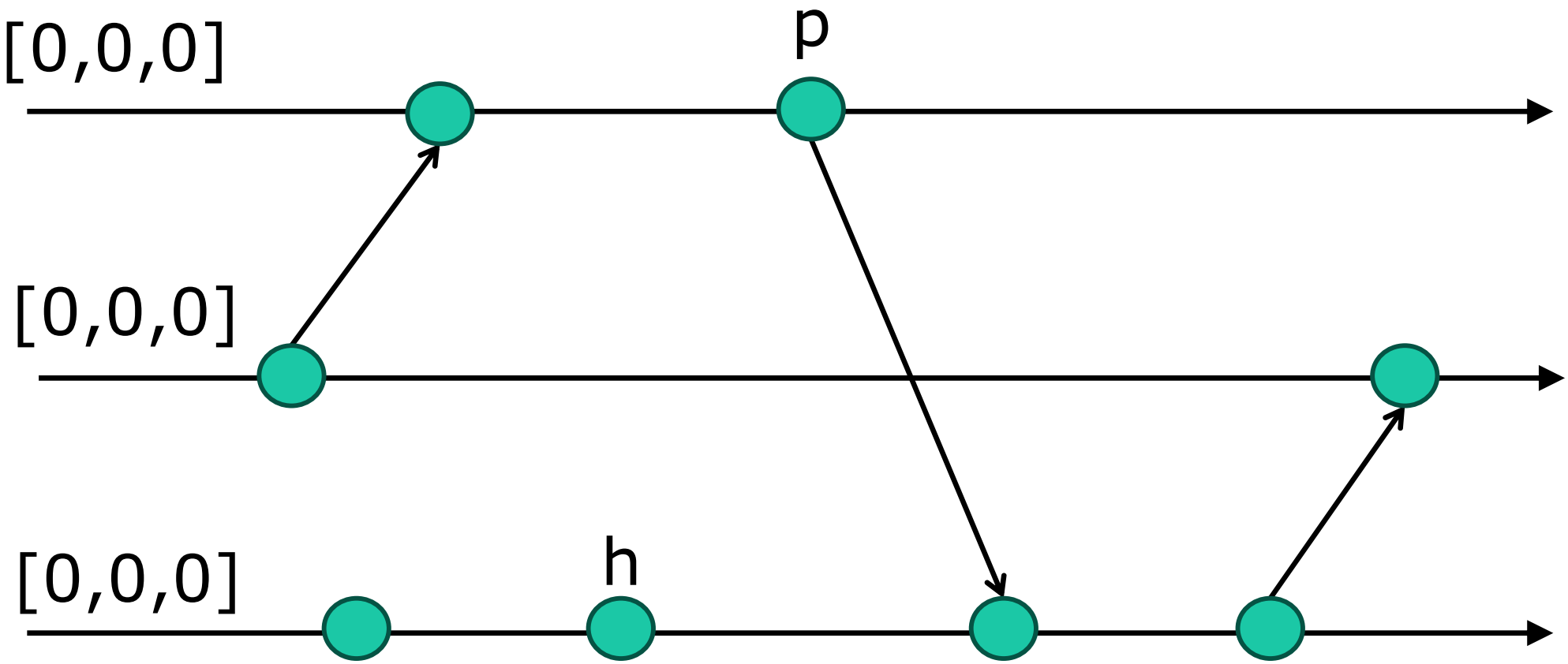
Concurrency and VC

Hence:

- VC(x) < VC(y) **iff** $x \prec y$

- If neither VC(x) < VC(y) nor VC(y) < VC(x) then x and y are concurrent

Exercise: Example (2)



Pros and cons

- Vector clocks are a simple means of capturing "happened before" exactly

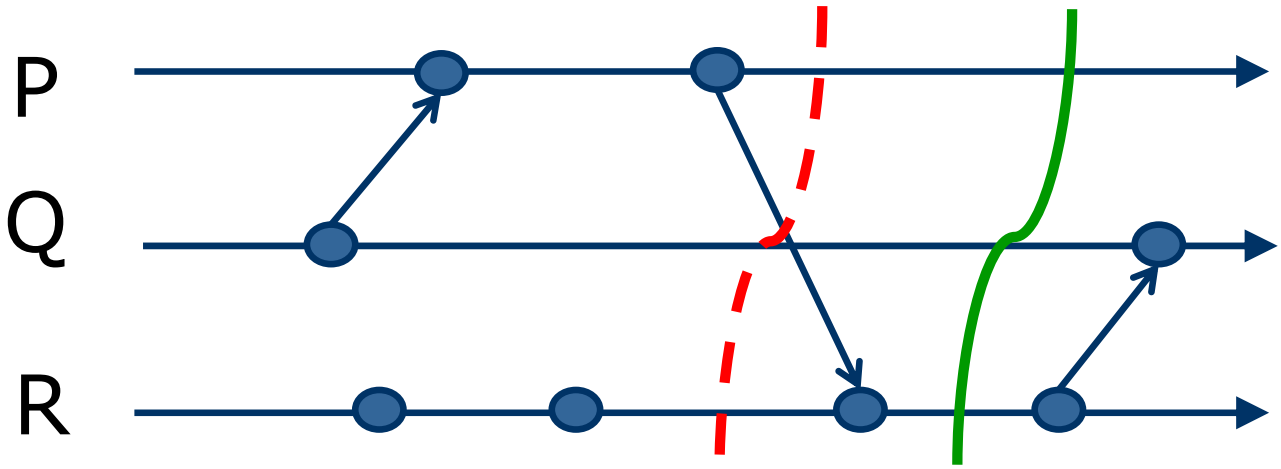
  $VC(x) < VC(y)$ iff $x \prec y$

Recall: $LC(x) < LC(y) \not\rightarrow x \prec y$

- For large systems we have resource issues (bandwidth wasted), and maintainability issues

Distributed snapshot

- Vector clocks help to synchronise at event level
  - Consistent snapshots



- But reasoning about response times and fault management needs quantitative bounds

Hard real-time comunication

Overview: Next three lectures

From one CPU to networked CPUs:

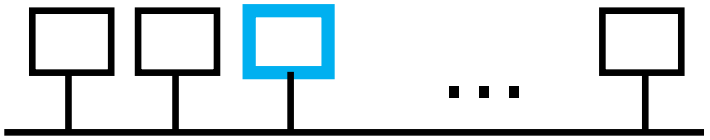
- First, from one CPU to multiple CPUs
  - Allocating VMs on multiple CPUs: Cloud
- Next, fully distributed systems
  - fundamental issues with timing and order of events
- Next, hard real-time communication
  - Guaranteed message delivery within a deadline, bandwidth as a resource
- Finally: QoS guarantees instead of timing guarantees, focus on soft RT

Reading Material

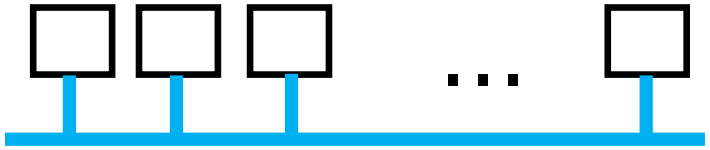
- TTP: Kopetz (2003) with focus on TTP/C, and Poledna (2014)
- CAN: Davis et al. (2007) with a focus on section 3 or Ch. 4.5 in Carlsson et al

Context

- In the scheduling lectures we looked at single processor hard real-time scheduling



- RT communication is about scheduling the communication medium



RT communication in applications

- Vehicle electronics
  - Power train and chassis
  - Infotainment/telematics
  - Body electronics

- A modern car configuration can have between 30 and 150 ECUs, distributed over several buses
- Avionics-specific standards
  - ARINC 429 (70's), AFDX (in Airbus 380)

Message constraints

- Message delivery time bound dictated by *application*
  - So called end-to-end deadlines

- Example: shortly after each driver braking, brake light must know it in order to turn on!

A more fundamental reason

Two interaction models in distributed systems

- Synchronous model
  - Assumes that the rate of computation at different nodes can be related, and there is a bound on maximum message exchange latency

Can use timers and timeouts

- Asynchronous model
  - Has no assumptions on rate of processing in different nodes, or bounds on message latency

Only coordination possible at event level

Real-time message scheduling

- Needed for providing the bound on maximum message delay
- Essential for reasoning about system properties under the synchronous model of distributed systems
  - e.g. proof that a service will be provided despite a single node crash will need bounds on message delay
  - We'll come back to that in dependability lectures…

Bandwidth as a resource

Scheduling messages

	Single Node	Distributed
Resource	CPU	Bandwidth
Scheduled element	Task/process	Message
Demand on resource	WCET & interarrival	Message size & frequency
Performance metric	Deadlines met & Utilisation	Message delay & Throughput

Two approaches

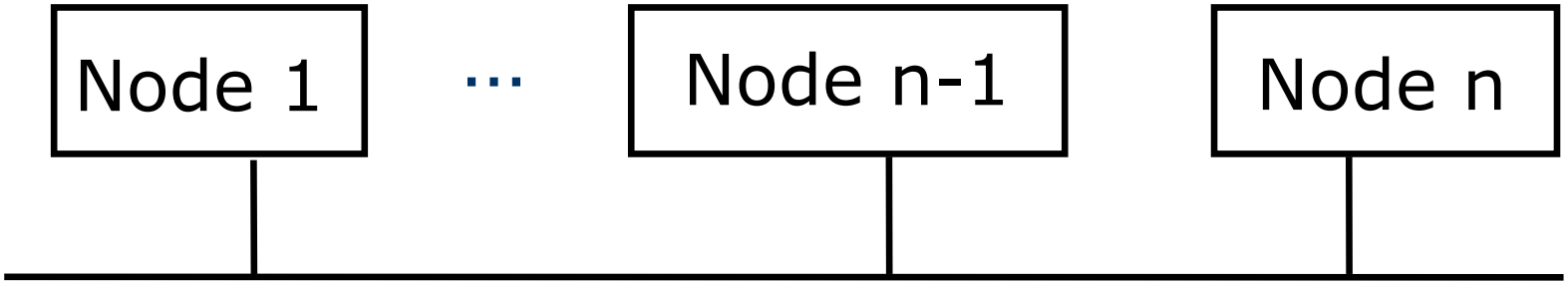
- We will look at two well-known methods for bus scheduling
  - Time triggered (TTP)
  - Event triggered (CAN)

- Used extensively in aerospace and automotive applications respectively

Time-triggered protocol

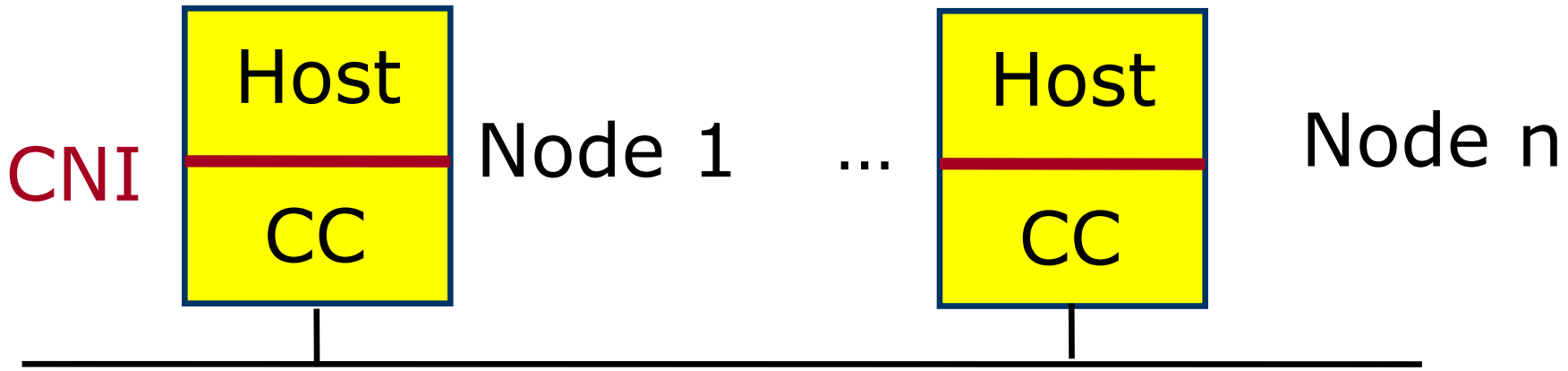
Time-triggered protocol (TTP)

- Origin in research projects in Vienna in early 90´s
  [Kopetz et al.]



- Time division multiple access (TDMA)

Temporal firewall



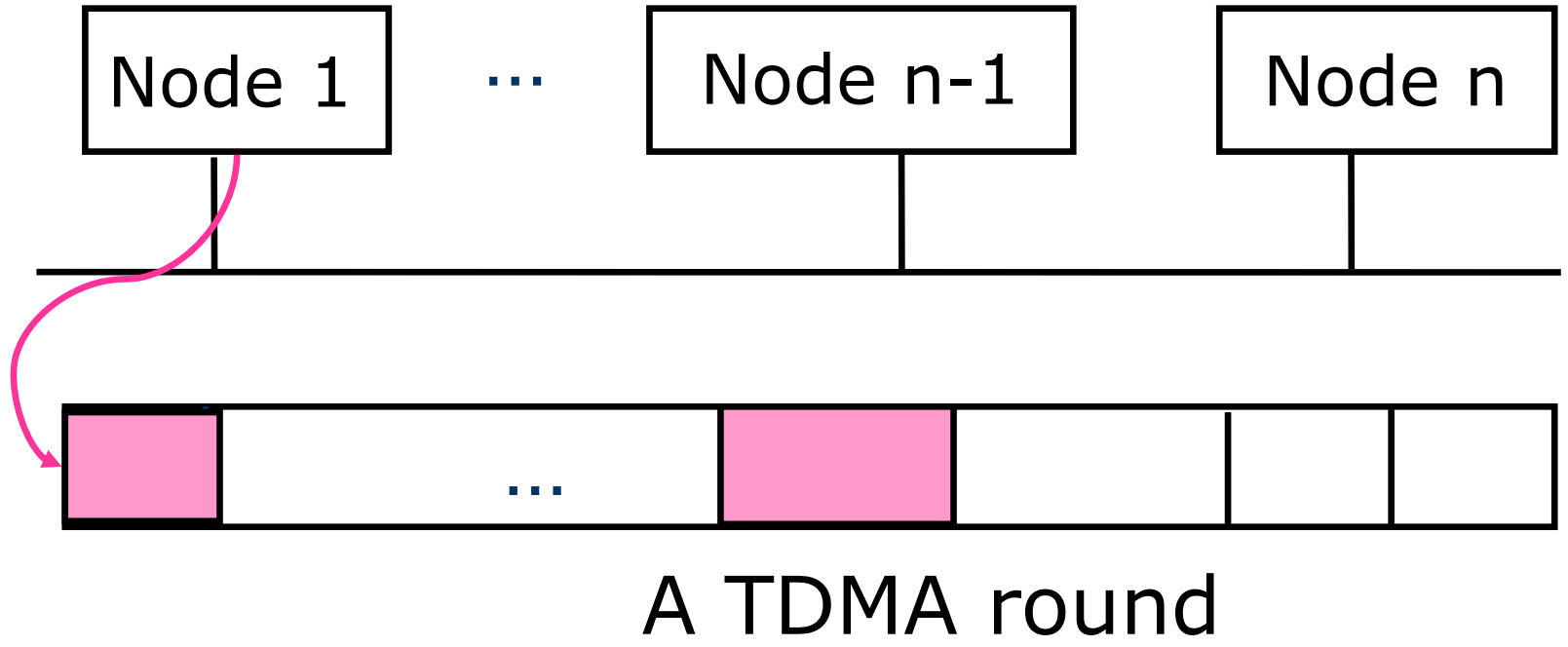
- CC provides temporally accurate state information (via clock synchronisation)
- When the data is temporally not valid, it can no longer be exchanged

Message scheduling

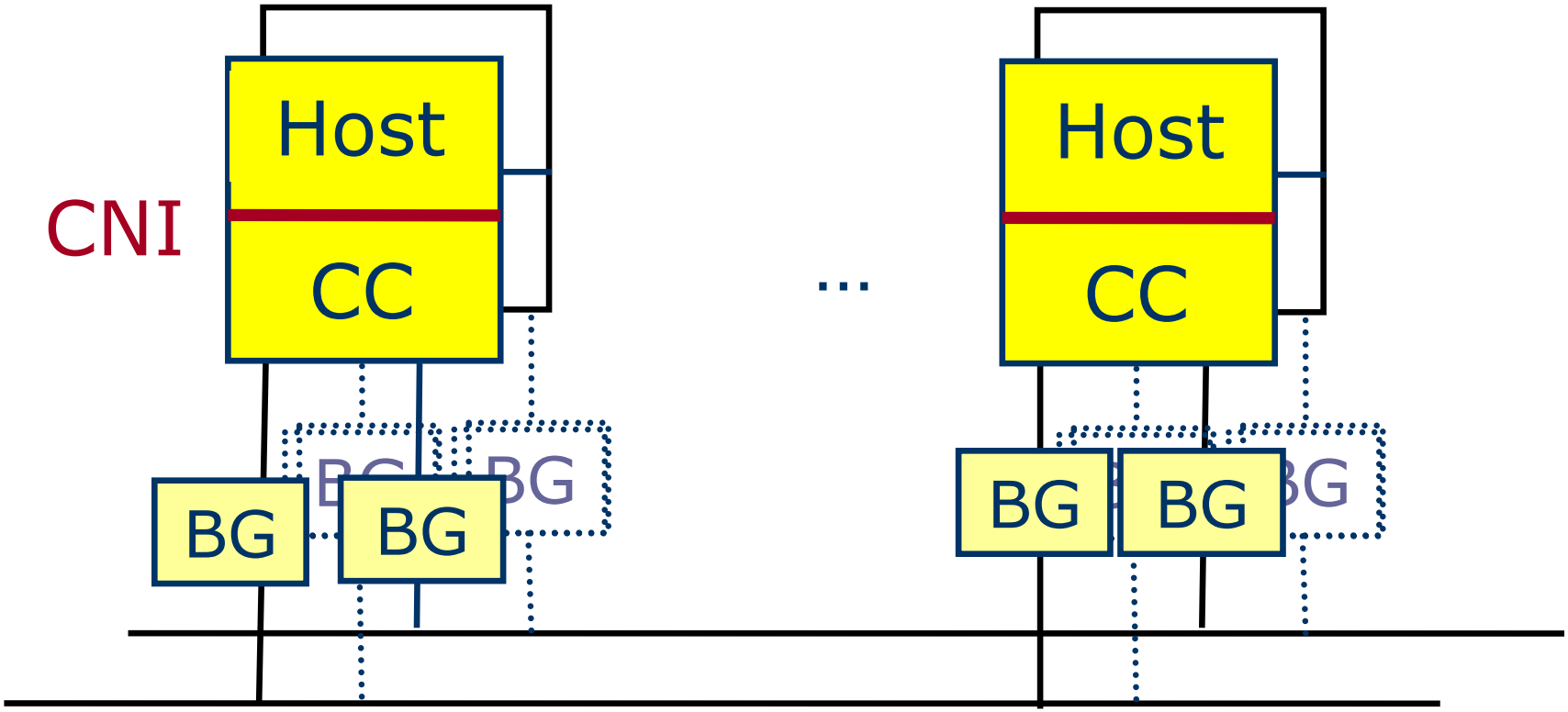
- TDMA round implemented through the MEDL (message description list)
  - The communication system (collection of CC:s and the bus) reads a message from the CNI of sending node at the apriori known fetch instant and places in the CNI of all other nodes at the apriori known delivery instant, replacing the previous value
- No constraints on (local) node CPU scheduling

Communication protocol

- Message Description List (MEDL): allocates a pre-defined slot within which each node can send its (pre-defined) message



TTP error detection



BG: Buss Guardian (stops babbling idiots)

CRC: for corruption in transit

- The major success of the TTP is due to the possibility of detecting additional faults including arbitrary (Byzantine) faults
- We will come back to this later…

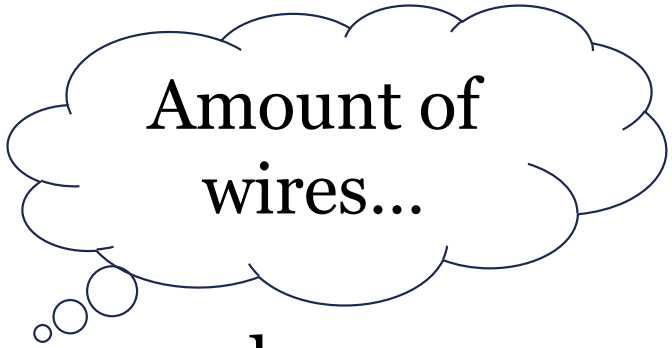
Event-triggered protocol

The CAN bus

- Controller Area Network protocol that was developed for use in all cars built in Europe

- Compulsory for the on-board diagnostics in USA car models from 2008

- Why?
  - Imagine: 2500 signals, 32 ECUs on one bus



Predecessor to CAN (1976)

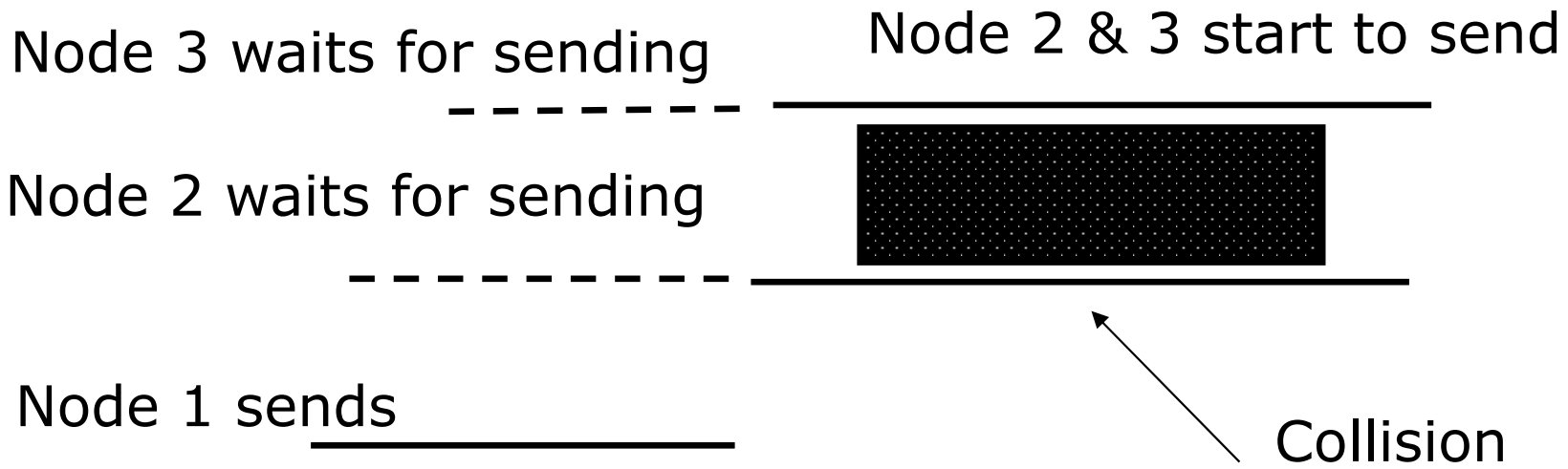
Ethernet:

- Current versions give high bandwidth but time-wise nondeterministic
- CSMA/CD
  - Sense before sending on the medium

    (Carrier Sense: CS)
  - All nodes broadcast to all (Multiple Access: MA)
  - If collision, back off and resend (Collision Detection: CD)

Collisions

- The original Ethernet has high throughput but temporally nondeterministic



Backoff

- The period for waiting after a collision
- Each node waits up to two “slot times” after a collision (random wait)
- If a new collision, the max. backoff interval is doubled
- After 10 attempts the node stops doubling
- After 16 attempts declares an error

Collisions and non-determinism

- Model the network throughput and compute probabilistic guarantees that collisions will not be *too often*
  - Theoretical study: With 100Mbps, sending 1000 messages of 128 bytes per second, there is a 99% probability that there will not be a delay longer than 1 ms due to collisions over ~1140 years

    [www.rti.com Ethernet study]
- If you cannot measure effects of collisions, make collision resolution deterministic!

CAN Protocol

- Developed by Bosch and Intel (1986)
- ISO Standard 1993
- Highest bandwidth 1Mbps, ~40m
- CSMA/CR: broadcast to all nodes
- CR: Collision resolution by bit-wise arbitration plus fixed priorities (deterministic)
- Bus value is bitwise AND of the sent messages

Message priority

- The ID of the frame is located at the beginning
  - initial bits that are inserted into the bus are the ID-bits

- ID also determines the priority of a frame
  - priority of the frame increases as the ID decreases

Bit-wise arbitration

Node 1 sends: 010…

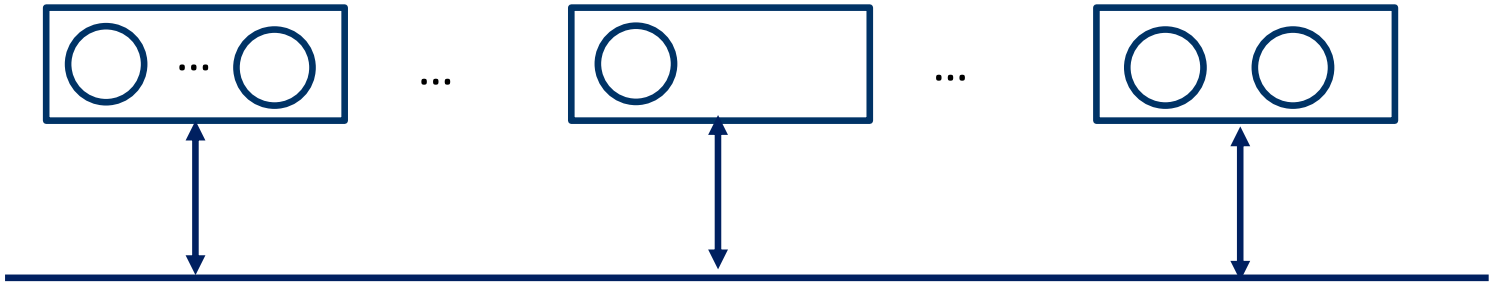
Node 2 sends: 100…

Node 3 sends: 011…

- This is how ID for a message (frame) works as its priority

Note

- Two roles for message ID:
  - Arbitration via priority
  - Processes on every node that receives a message, use the ID to work out whether that message is any use to them or not



Response time analysis

- Scheduling analysis: Is *every* message delivered before its deadline?

Questions?

www.ida.liu.se/~TDDD07

LINKÖPING UNIVERSITY