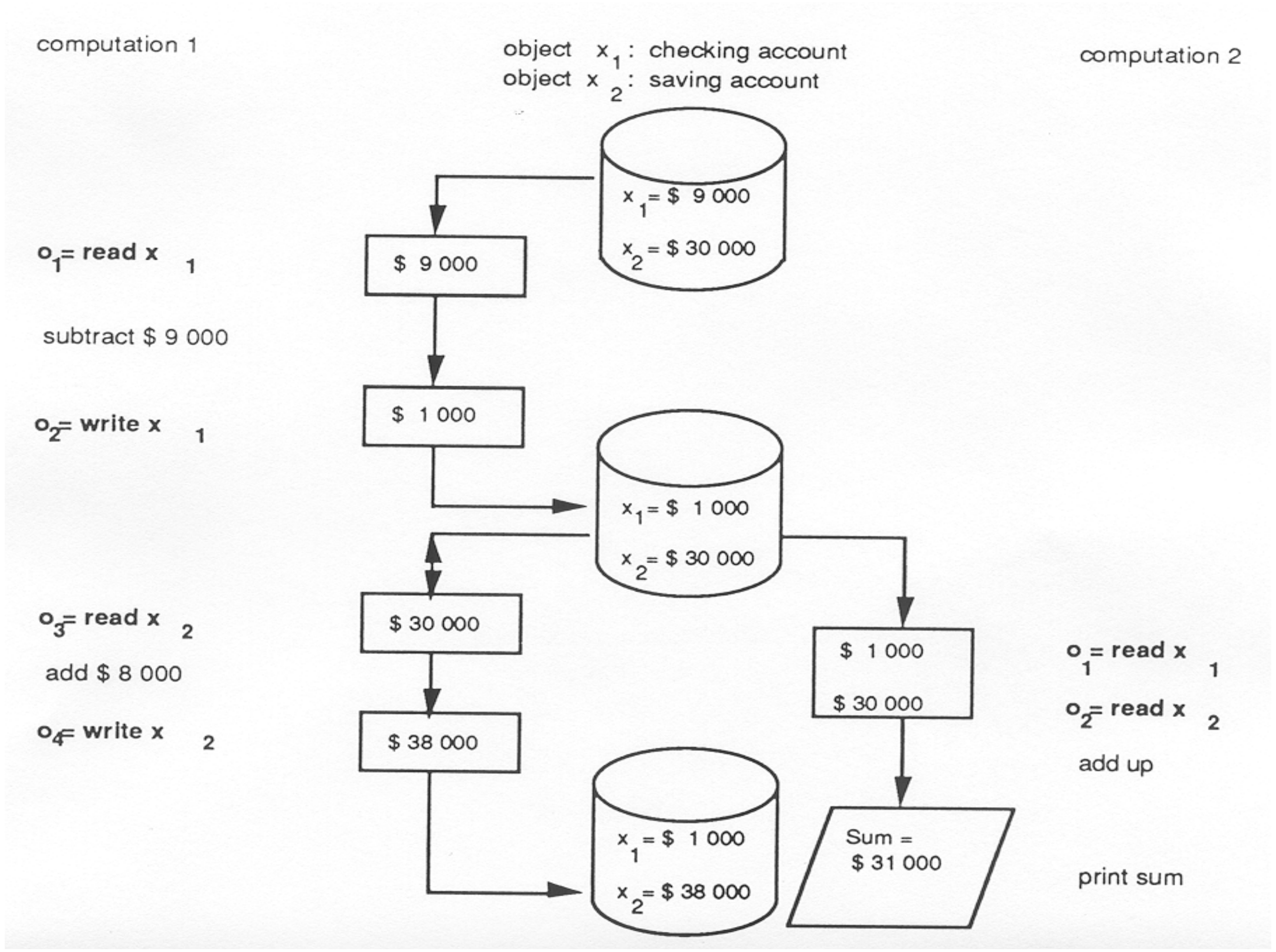
Real-Time (Paradigms) (47)

Memory: Memory Access Protocols

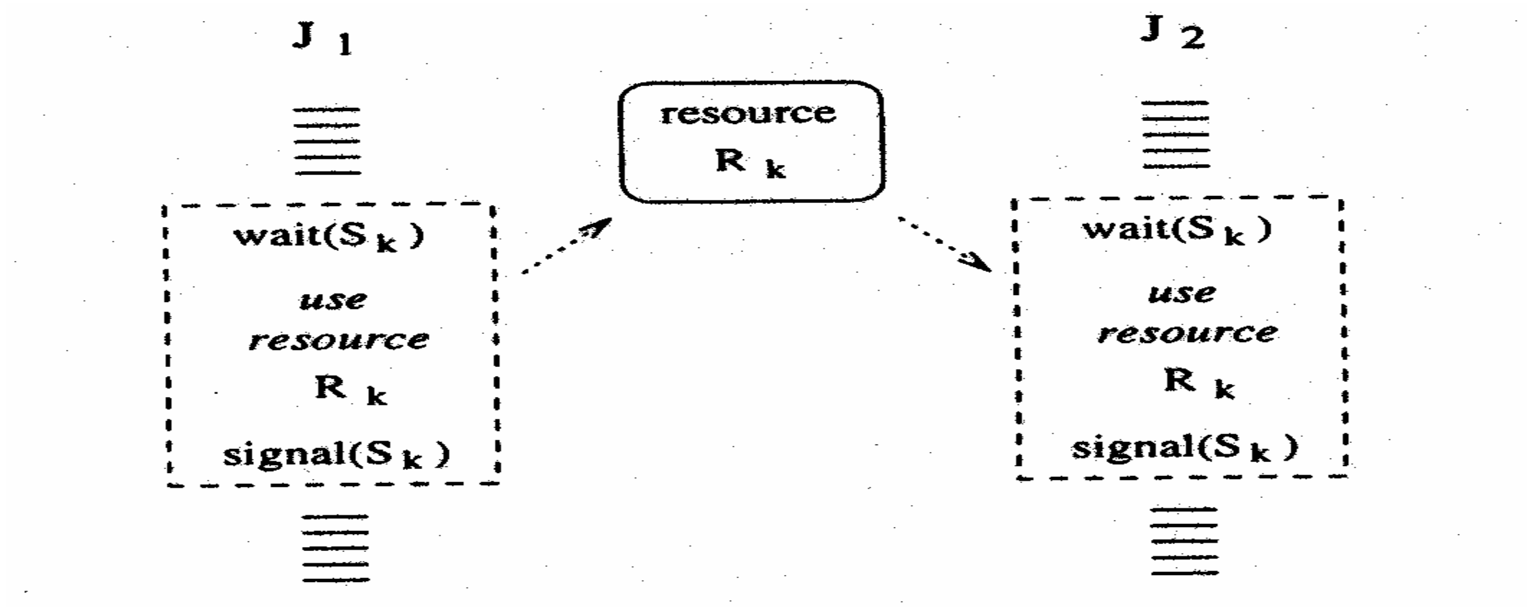
Tasks competing for exclusive memory access (critical sections, semaphores) become interdependent, a common phenomenon especially in distributed systems

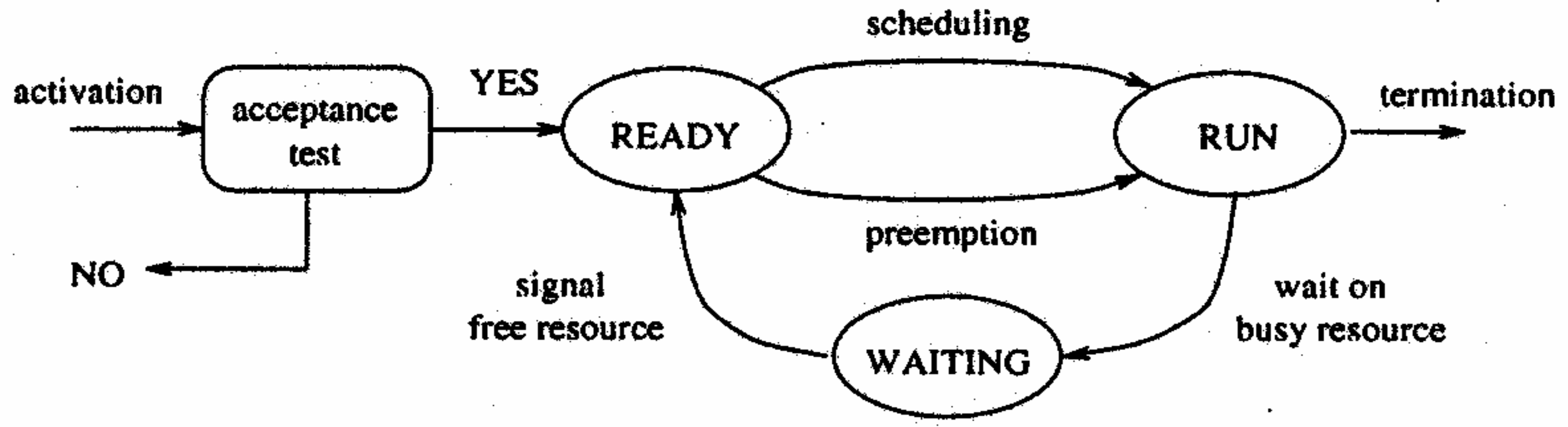
Example:



Real-Time (Paradigms) (47a)

Realizing mutual exclusion by semaphores when accessing an exclusive resource:



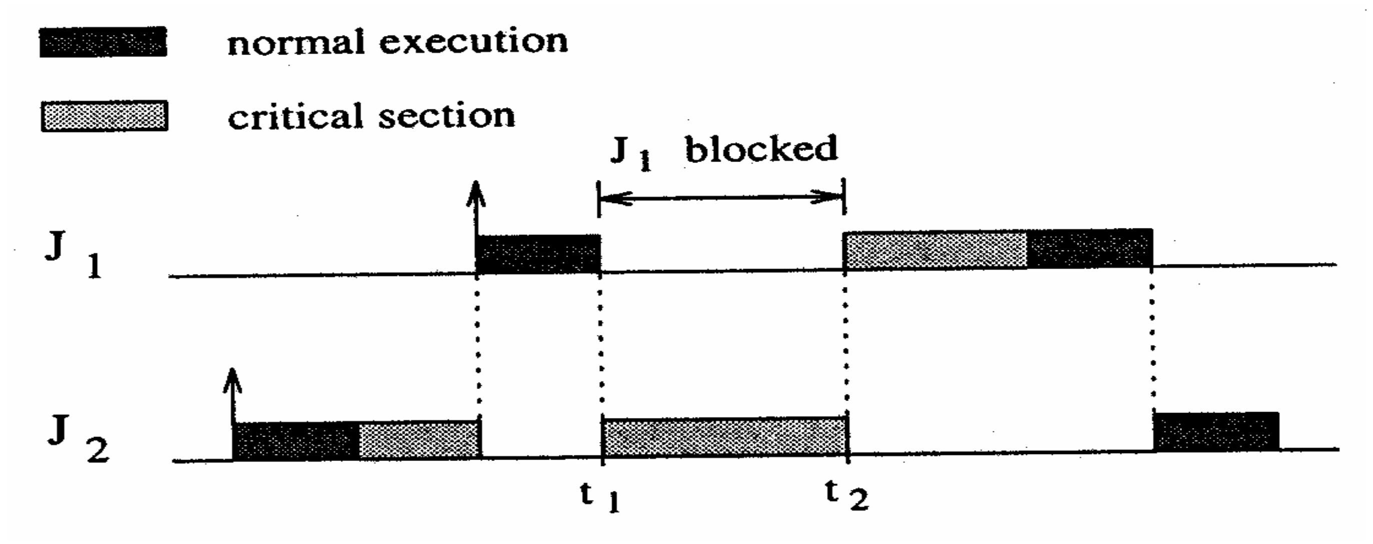


Real-Time (Paradigms) (47a)

---> determining memory access times as part of the overall execution time becomes extremely difficult

Why?

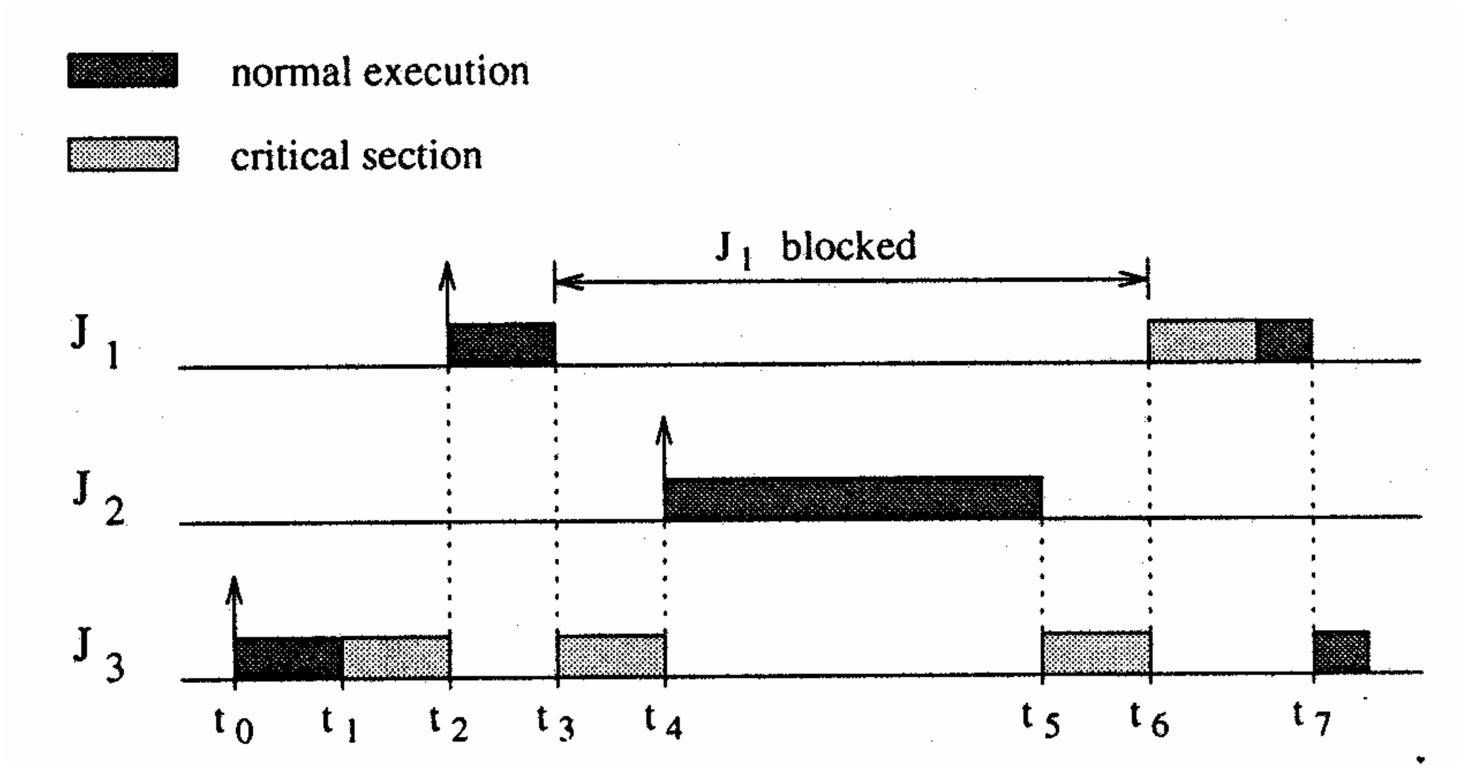
Example of blocking:



–––> We still can compute an upper bound on the execution time

Real-Time (Paradigms) (48)

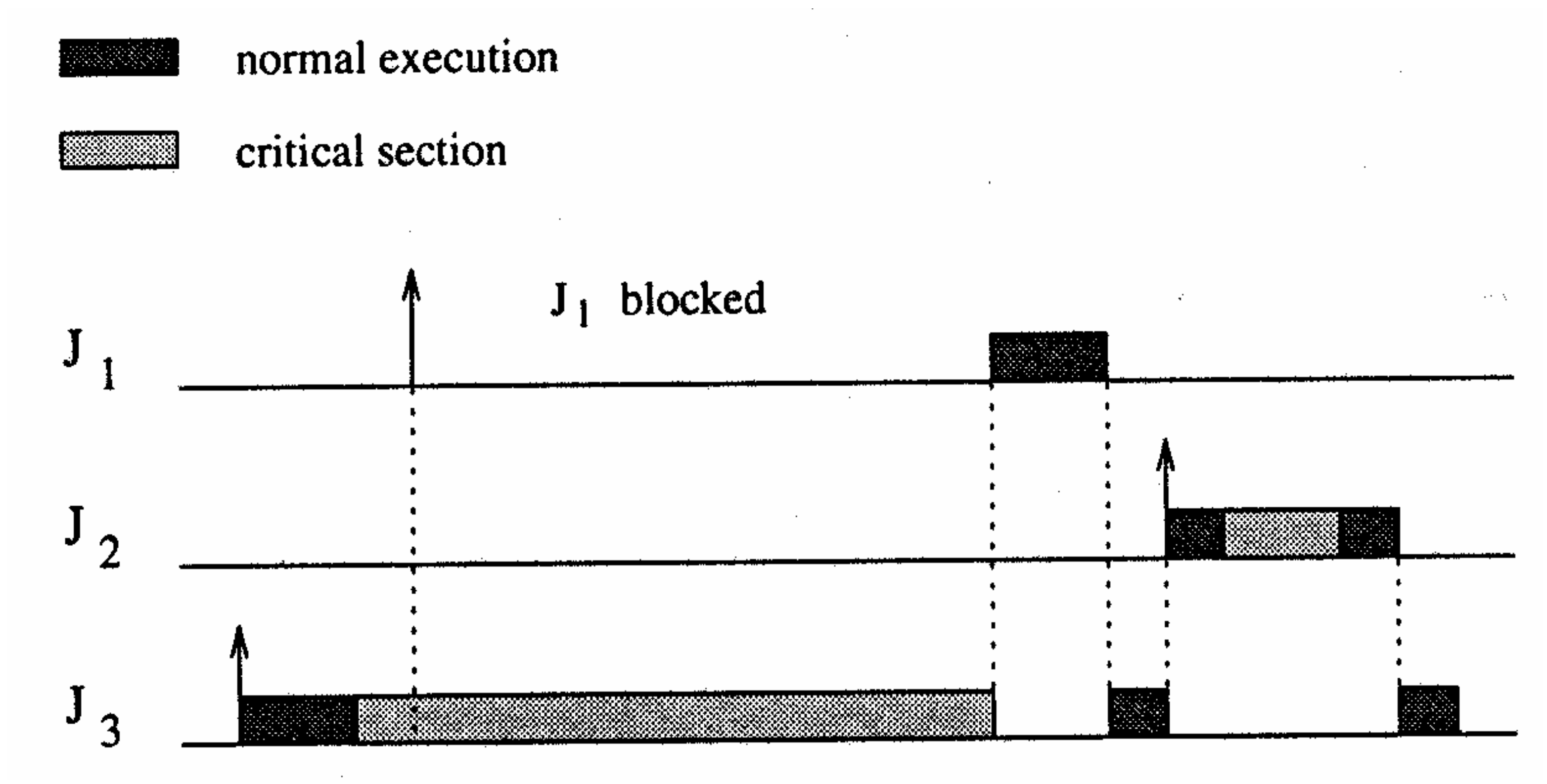
Example of priority inversion:



Effects of priority inversion do affect predictability!

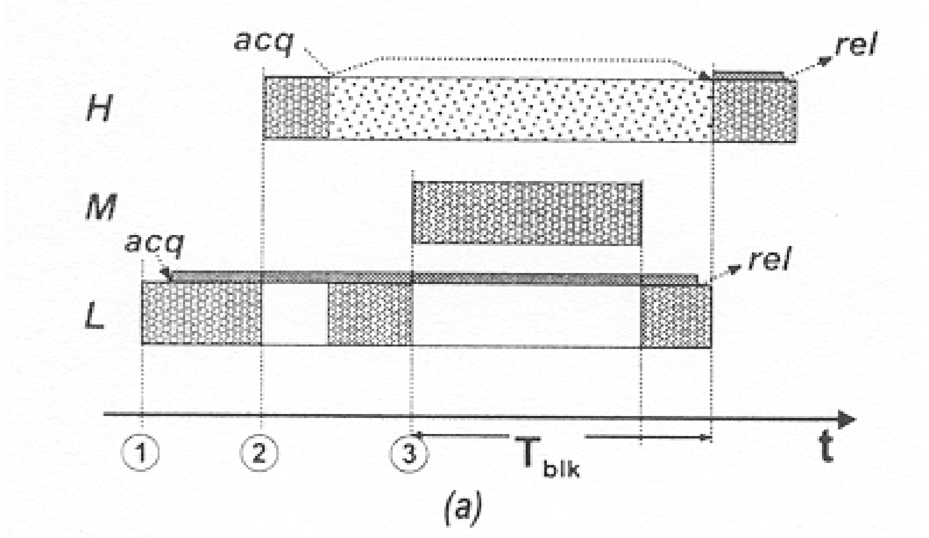
Real-Time (Paradigms) (48a)

Scheduling with non-preemptive critical sections



Real-Time (Paradigms) (49)

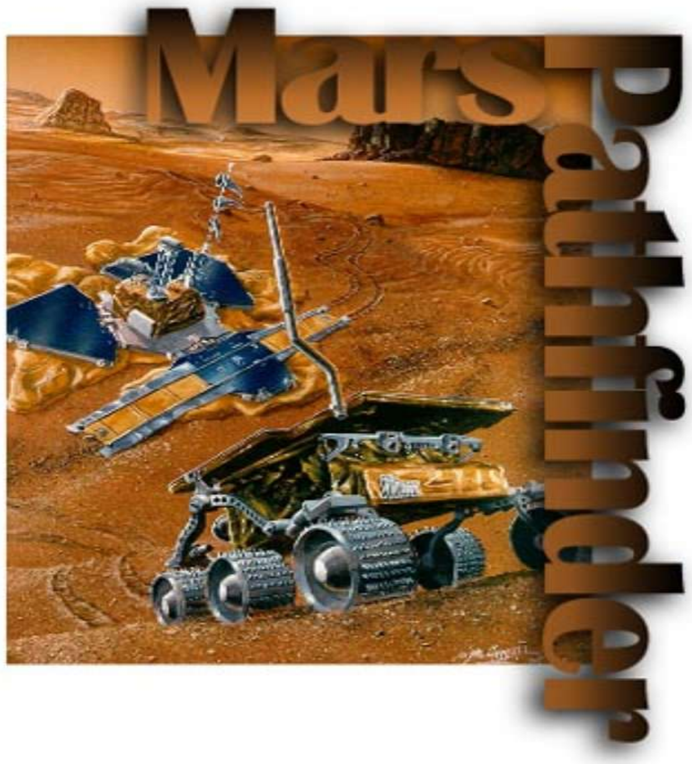
Application Example :



It happened on Mars!

Using *priority inheritance* can prevent priority inversion.

It introduces dynamic priorities defined as follows:
the dynamic priority of a task at time t is the maximum of its initial fixed priority and the priorities of all tasks blocked on account of it at time t.



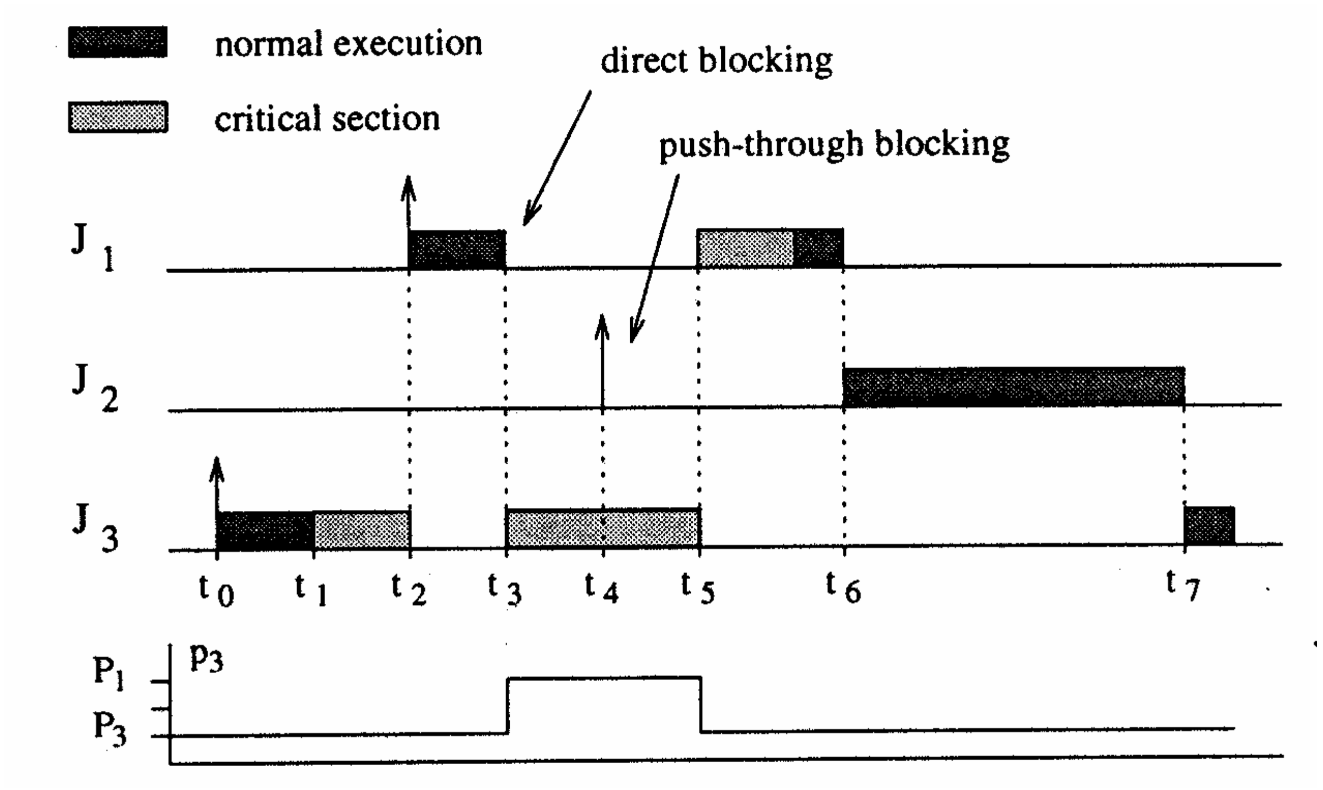
Real-Time (Paradigms) (50a)

Definition of the priority inheritance protocol:

- Jobs are scheduled based on their active priorities. Jobs with the same priority are executed in a First Come First Served discipline.
- When job J_i tries to enter a critical section $z_{i,j}$ and resource $R_{i,j}$ is already held by a lower-priority job, J_i will be blocked. J_i is said to be blocked by the task that holds the resource. Otherwise, J_i enters the critical section $z_{i,j}$.
- When a job J_i is blocked on a semaphore, it transmits its active priority to the job, say J_k, that holds that semaphore. Hence, J_k resumes and executes the rest of its critical section with a priority $p_k = p_i$. J_k is said to *inherit* the priority of J_i. In general, a task inherits the highest priority of the jobs blocked by it.
- When J_k exits a critical section, it unlocks the semaphore, and the highest-priority job, if any, blocked on that semaphore is awakened. Moreover, the active priority of J_k is updated as follows: if no other jobs are blocked by J_k, p_k is set to its nominal priority P_k, otherwise it is set to the highest priority of the jobs blocked by J_k.
- Priority inheritance is transitive; that is, if a job J_3 blocks a job J_2, and J_2 blocks a job J_1, then J_3 inherits the priority of J_1 via J_2.

Real-Time (Paradigms) (50)

Example of a priority inheritance protocol:



Two kinds of blocking can be distinguished:

- *direct blocking*
- *push-through blocking*

Resource Access Protocols (6)

Properties of the priority inheritance protocol :

Lemma 7.3 *If there are n lower-priority jobs that can block a job J_i, then J_i can be blocked for at most the duration of n critical sections (one for each of the n lower-priority jobs), regardless of the number of semaphores used by J_i.*

Lemma 7.4 *If there are m distinct semaphores that can block a job J_i, then J_i can be blocked for at most the duration of m critical sections, one for each of the m semaphores.*

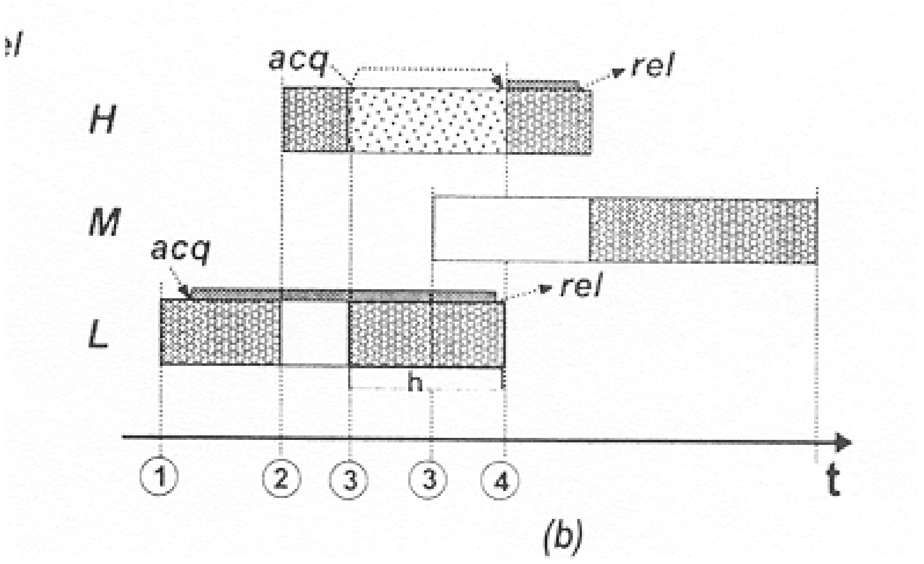
Theorem 7.1 (Sha-Rajkumar-Lehoczky) *Under the Priority Inheritance Protocol, a job J can be blocked for at most the duration of $\min(n, m)$ critical sections, where n is the number of lower-priority jobs that could block J and m is the number of distinct semaphores that can be used to block J.*

Theorem 7.2 *A set of n periodic tasks using the Priority Inheritance Protocol can be scheduled by the Rate-Monotonic algorithm if*

$$\forall i, \quad 1 \leq i \leq n, \quad \sum_{k=1}^{i} \frac{C_k}{T_k} + \frac{B_i}{T_i} \leq i(2^{1/i} - 1). \qquad (7.2)$$

Real-Time (Paradigms) (50a)

The same example using priority inheritance:



- *1. The meteo task (L) runs with priority l, acquires the mutex and publishes*
- *2. The dispatcher task (H) runs with priority h: H preempts L, tries to acquire the mutex and blocks on it, awaiting for the meteo task L, which runs again inheriting H's priority, h.*
- *3. The communications task (M), with priority m, becomes ready for execution: M waits for L, since h (current pri. of L) is higher than m.*
- *4. L finishes and releases the mutex unblocking H, which grabs the processor ($h > m$), acquires the mutex, and runs to completion. Then, M runs.*
- *Conclusion: H had the minimum blocking possible: waiting for L to finish.*

Real-Time (Paradigms) (51)

5. Real-Time Communication

Data flow (communication) in embedded systems :

Sensor --> Controller

Controller --> Actor

Controller --> Display

Control Panel --> Controller

Controller <--> Controller

Major challenge in real-time communication:

The communication delay adds to the computer response time.

---> the delay has to be bounded and to be known, even in the presence of disturbing factors such as other (RT) traffic, variable load, or faults

Different goals ---> different key performance measure

In non-real-time systems: Throughput

In real-time systems: Guarantee (high probability) of delivery of a message within a certain deadline

Real-Time (Paradigms) (52)

Principal service of the Medium Access Control Sublayer:

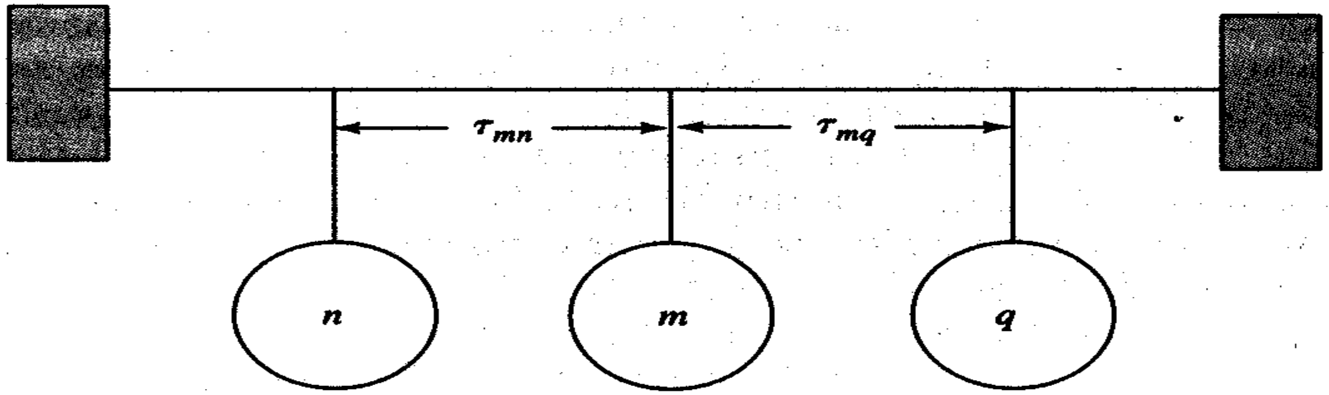
Allocating a single broadcast channel (like Ethernet) among competing users

CSMA protocols

All the nodes can monitor the communication channel and can check whether it is busy or idle (This means carrier sense). If a node observes that the channel is busy, it refrains from sending in order not to interfere (collide).

CSMA is a truly distributed algorithm, with each node deciding when to transmit ---> collisions are still possible and must be detected ---> CSMA/CD protocols like Ethernet. **Are they suitable for Real-Time?**

Example case of a collision in a bus network



There are various types of CSMA, differing in how to compute the time until trying a retransmission.

Real-Time (Paradigms) (53)

Types of CSMA protocols

1-persistent

Behavior:

When a station has data to send, it transmits with a probability of 1 whenever it finds the channel idle. If the channel is busy, the station waits until it becomes idle (greedy approach).

nonpersistent

Behavior:

It differs from 1-persistent w.r.t. the case where the channel is busy.Then, the station deliberately waits a random period of time before sensing the channel again.

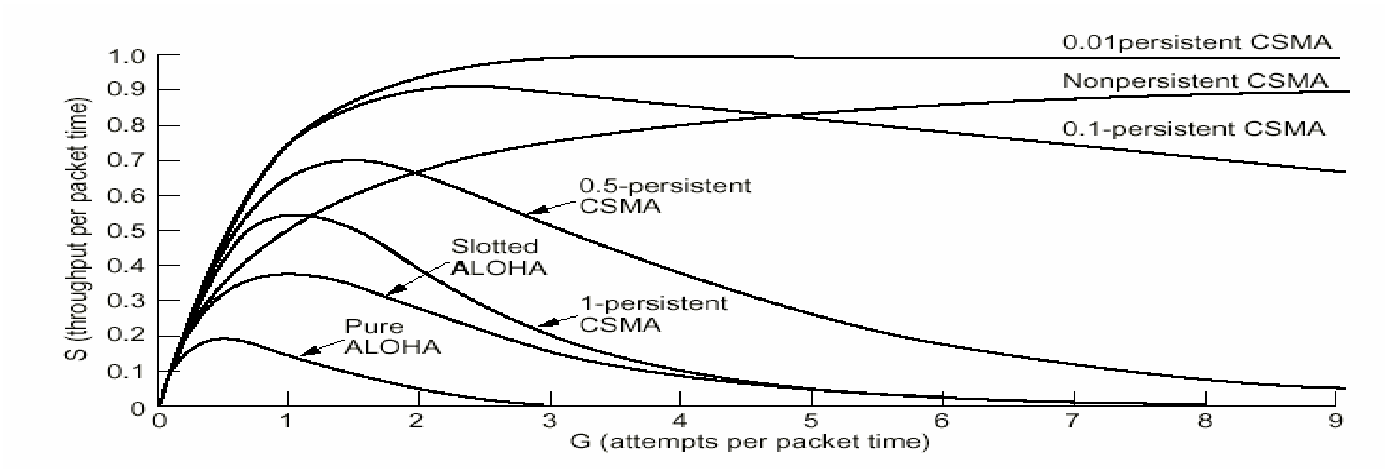
p-persistent (applies to slotted channels)

Behavior:

When a station has data to send, it transmits with a probability of p whenever it finds the channel idle. With a probability of q = 1- p it defers until the next slot and the same procedure iterates. If the channel has become busy meanwhile, the station waits a random time and starts again.If the channel is busy when first sensing it, the station waits until the next slot and repeats the procedure.

Real-Time (Paradigms) (54)

Comparison of channel utilization versus load :



---> CSMA/CD protocols are not suitable for Real-Time because they cannot be time-bounded!

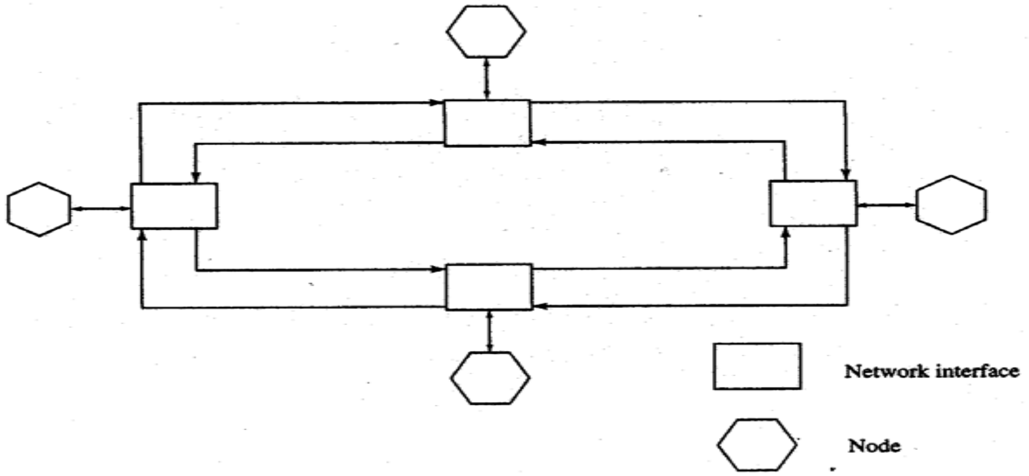
Real-Time (Paradigms) (55)

Examples of real-time communication systems

Timed-Token Protocol

Physical medium: Bus or ring on any medium

Example of a typical ring structure



Basic algorithm for accessing the medium:

- A *token* is a grant of permission to a node to transmit its packets on the network.
- When the token-holding node completes its transmission, it surrenders the token to another node.
- A node is only permitted to transmit on the network if it currently holds the token.
- The token is passed along a logical ring of all participating nodes.

Key parameter TTRT (target token rotation time):

Upper bound for the cycle time of the token which is defined as :
The time it takes for the token to make a complete circuit around the nodes of the ring.

Real-Time (Paradigms) (56)

Basic idea of the protocol:

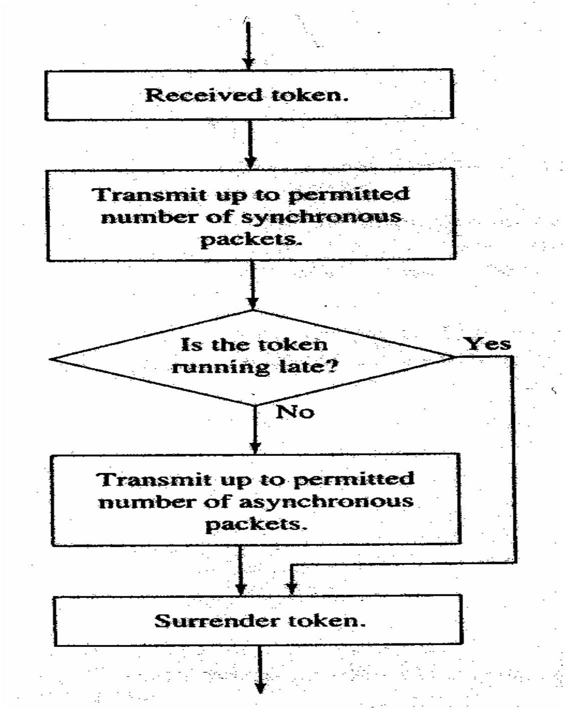
If each node *i* (of overall *n* nodes) holds the token for a fraction f_i of TTRT, with

$$\sum_{i=i}^{n} f_i TTRT \leq TTRT$$

it is guaranteed that each node can send real-time messages up to a certain amount of time ($f_i\ TTRT$) each TTRT time units.
This is a simple protocol, that fulfils hard real-time requirements and is efficient as long as each node has to send only real-time traffic (constant or bounded rate), i..e. the fraction defined above represents only the real-time traffic.

Extension of the protocol to allow for mixed real-time and non-real-time traffic ("synchronous" and "asynchronous packets")

- The token is called "late", if the time since its previous arrival time at the same node (i.e. the actual cycle time) is greater than TTRT.
- The token may become late if the transmission of an asynchronous packet delays the token rotation.


Received token.
Transmit up to permitted number of synchronous packets.
Is the token running late?
Yes
No
Transmit up to permitted number of asynchronous packets.
Surrender token.

Real-Time (Paradigms) (57)

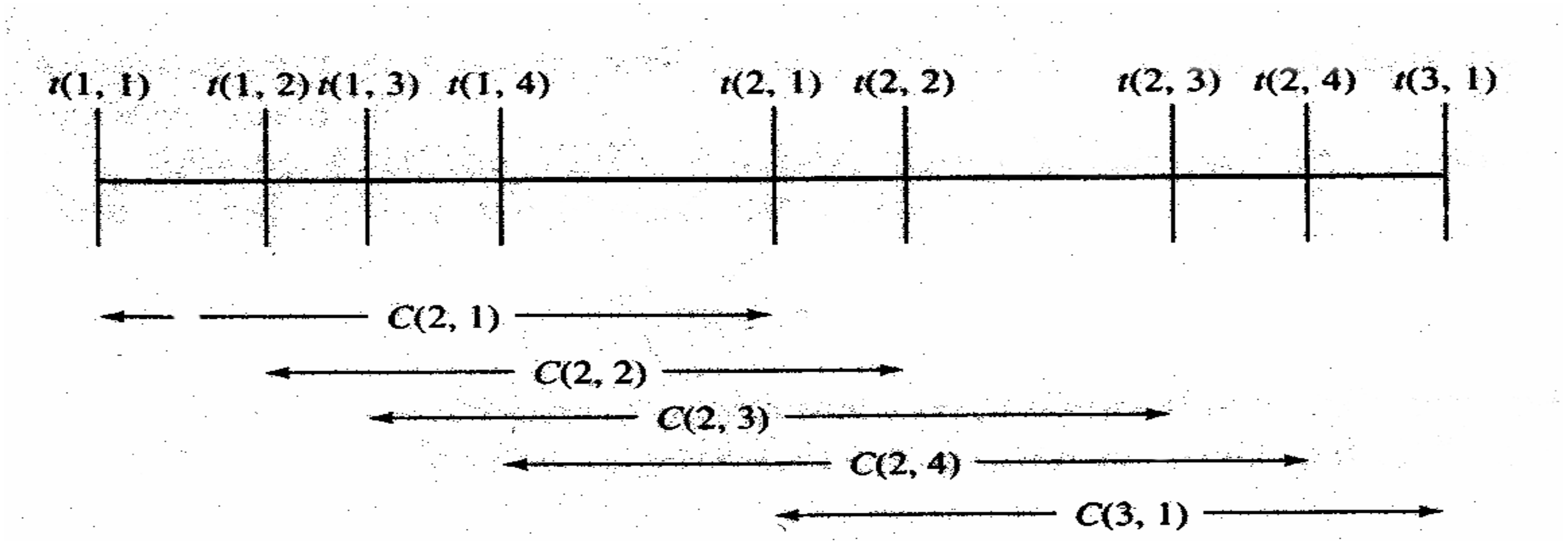
Analysis of the Timed-Token Protocol

Claim: Even with this extension the protocol still provides predictable timing behavior, the token-cycle time is bounded (in contrast to CSMA)

Theorem 1:
In the absence of failures, the maximum cycle time of the token is not greater than twice the TTRT.

Proof: Without loss of generality we ignore overhead for this proof.

Let *(a, b)* the *a*th visit of the token to node *b*, *t(a,b)* the time when the token leaves at *(a,b)*, *C(a,b) = t(a-1,b - t(a, b)* the *a*th token-cycle time as seen by *b*



Real-Time (Paradigms) (58)

S(a, b) the time for transmitting the synchronous packets at *(a, b)*

A(a, b) the time for transmitting the asynchronous packets at *(a, b)*

Case 1: The token was not late during the entire cycle preceding visit *(a, b)*

=> $C(a, b) < TTRT$ ✓

Case 2: The token was always late during the entire cycle preceding visit *(a, b)*

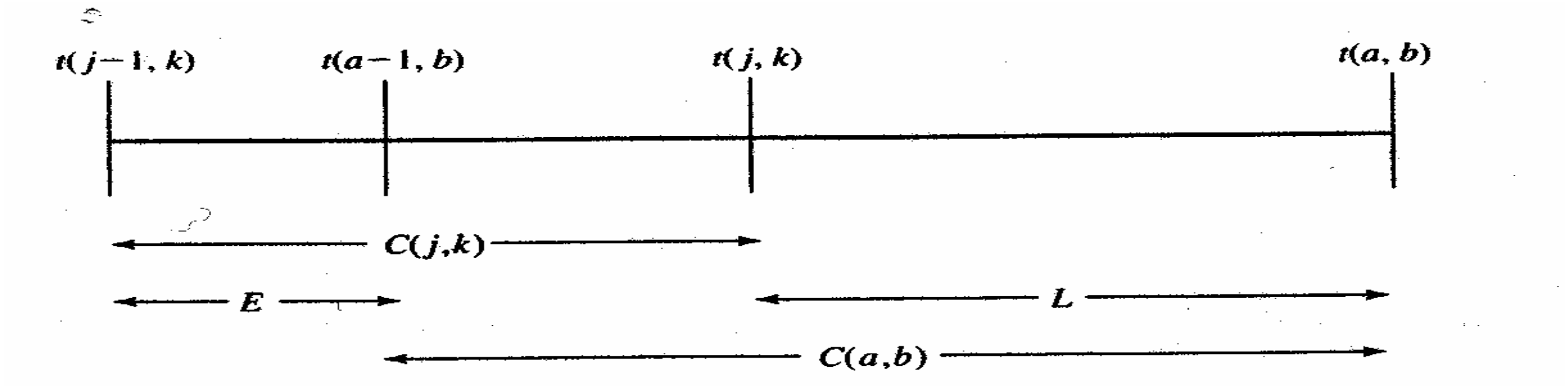
=> during *C(a, b)* only synchronous traffic is transmitted

$$
\begin{aligned}
C(a, b) &= \sum_{x, y = a-1, b+1}^{a, b} S(x, y) \\
&\leq \text{TTRT} \sum_{x, y = a-1, b+1}^{a, b} f_i \\
&\leq \text{TTRT}
\end{aligned}
$$

=> two consecutive cycles cannot have a time greater TTRT ✓

Real-Time (Paradigms) (58b)

Case 3: The token was not late for a part of the cycle preceding *(a, b)*. *(j, k)* was the visit preceding *(a, b)* were the token was not late the last time.



Since the token was early at t(j, k): *C(j, k) <= TTRT*

The token is late over the interval L (only synchronous traffic allowed):

$$C(a,b) = C(j,k) - E + L$$

$$\leq \mathrm{TTRT} - E + \sum_{x,y=j,k}^{a,b} S(x,y)$$

$$\leq \mathrm{TTRT} + \sum_{x,y=j,k}^{a,b} S(x,y)$$

$$\leq TTRT + TTRT \leq 2TTRT \quad ✓ \qquad ■$$

Real-Time (Paradigms) (59)

Impact of TTRT on available bandwidth (Throughput):

Token algorithms incur the following overheads constantly per cycle, regardless of the data volume transmitted:

Medium progagation delay: It takes a certain time for a message to propagate from one node to the next.

Token transmission time: Sending out the token takes some time. Since the token is usually much smaller than a frame that contains information, this overhead is typically very small.

Token capture delay: There is usually some time lag between when a node captures the token and when it begins transmitting.

Network interface latency: At each network interface, the input is retransmitted to the output (except for packets that are removed from the ring). The network interface latency is the time between when a bit is received by the network interface and when it is retransmitted.

Let the overall overhead of one cycle be *O* **-->** the useful time for message transmission per cycle is *TTRT*-*O*. The utilization of the medium is upper-bounded by

$$\Psi = \frac{TTRT - O}{TTRT}$$

--> the throughput is reduced to ΨB (B is the bandwidth bits/time).

→Trade-off:

A smaller TTRT leads to a smaller upper bound of the token delay, but also to a smaller throughput ΨB.

Real-Time (Paradigms) (60)

Analysis of the Timed-Token Protocol

Theorem 2:
The total duration of L consecutive cycles of the token is upper bounded by $(L+1)TTRT$, for L = 1,2,...

Corollary 1: Over any interval of duration I, node n_i will be able to transmit at least:

$$\left\lfloor \frac{I}{TTRT} - 1 \right\rfloor f_i TTRT\, B$$ bits (B is the bandwidth bits/time)

Supporting periodic messages

Corollary 2: If a node n_i wants to send with a period of P_i: $TTRT \leq \frac{P_i}{2}$ (1)

Corollary 3: (Computation of the synchronous quota (fraction of time reserved for synchronous messages c_i))

If node n_i has to send c_i bits per period P_i, then $\left\lfloor \frac{P_i}{TTRT} - 1 \right\rfloor f_i (TTRT - O) B \geq c_i$

This equation can be solved for f_i , the upper bound for c_i .
Both equations are necessary and sufficient conditions for node n_i to be able to transmit c_i bits of real-time data every P_i seconds.

Real-Time (Paradigms) (61)

Initialization of the Timed-Token Protocol

- In the 1. cycle, each node n_i sends out its desired TTRT (desired $P_i/2$ if periodic tasks)
- The smallest requested TTRT is chosen
- The node having requested the smallest TTRT generates the token. If two nodes request the same TTRT, the tie is broken by the node-id
- The token site computes and distributes the fraction f_i
- During the 2. cycle of the token only synchronous packets are permitted
- Now the protocol is in the steady state

Real-Time (Paradigms) (62)

Static TDMA (Time Domain Multiplexed Access)

Physical: Broadcast-bus with copper or fiber medium

Medium access:

Each node is allowed to send messages only during a predetermined time span, called its TDMA slot. The allocation of these slots is determined at system’s design time. During run-time the nodes maintain a global clock and each nodes exactly knows which messages can be expected in the next slot.

Pros:

- Hard real-time capable by construction
- No control messages or bus arbitration required
- A-priori knowledge can be used for fault detection

Cons:

- No flexibility, everything must be known a-priori (static scheduling required)
- Can be inefficient (e.g. all sporadic messages must be mapped to periodic slots)
- raises problems (waste or lack of bandwidth) when number of users varies
- inherently inefficient for most general-purpose computer systems, since data traffic is extremely bursty (1000:1 ratios)

Real-Time (Paradigms) (63)

Implementation of static TDMA : TTP/C (Time-Triggered Protocol)

TTP realizes communication between the nodes of a TTA.

Time-Triggered Architecture

all system actions are initiated by the progression of a globally synchronized time

---> the periodic clock signal is the only control signal in the system

---> all clocks are synchronized

---> every event (observation of the controlled object) is timestamped

---> granularity of global time (clock signals) such that temporal order of any to events can be reestablished from their timestamps

---> a TT systems takes a snapshot (observation) of the world at predetermined points of time

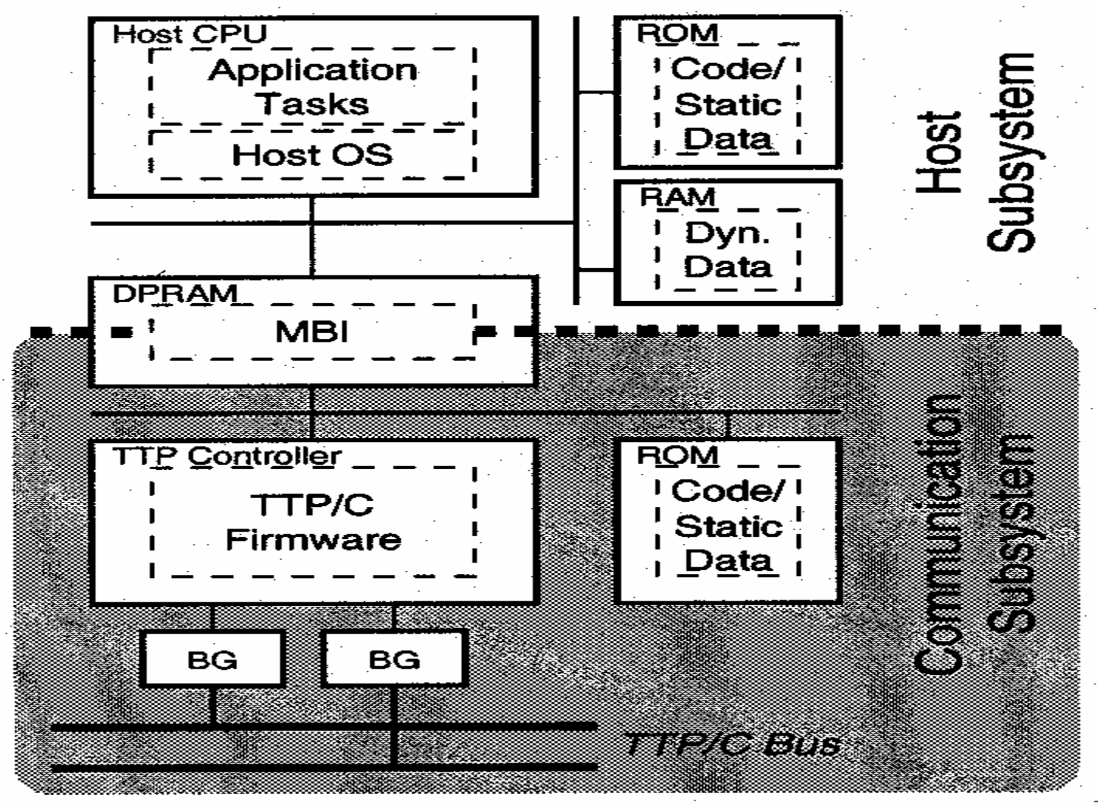
---> messages have a state-message semantics

---> eliminates the problem of dynamic buffer management

---> it is determined (scheduled) a priori, which messages are sent (received) in each time slot

Real-Time (Paradigms) (64)

Structure of a TTA node computer:



Real-Time (Paradigms) (65)

Polled Bus Protocol

Physical: Broadcast-bus with copper or fiber medium

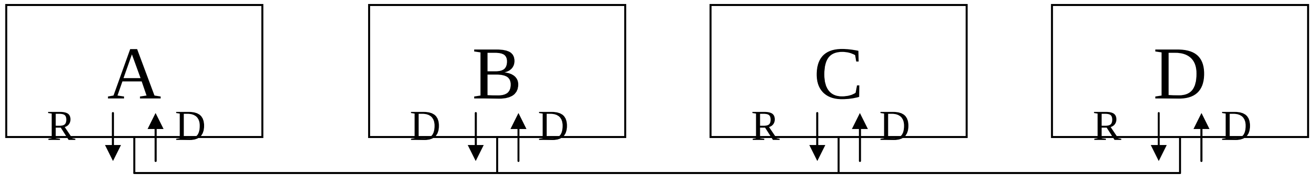
Medium access:

The bus maintains a *busy*-line. This line is used for for priority-based bus arbitration and during transmission for enforcing mutual exclusion on the broadcast medium.

Busy-line: executes wired-OR

- the line has two states *dominant* and *recessive*
- if one node assigns *dominant* to the busy-line, all node perceive *dominant*
- the time on the bus is divided into equally long bit-times (slots)
- one bit-time is long enough to propagate the signal to all participants

  ---> Bit-time is a function of the bus-length



Mutual exclusion:

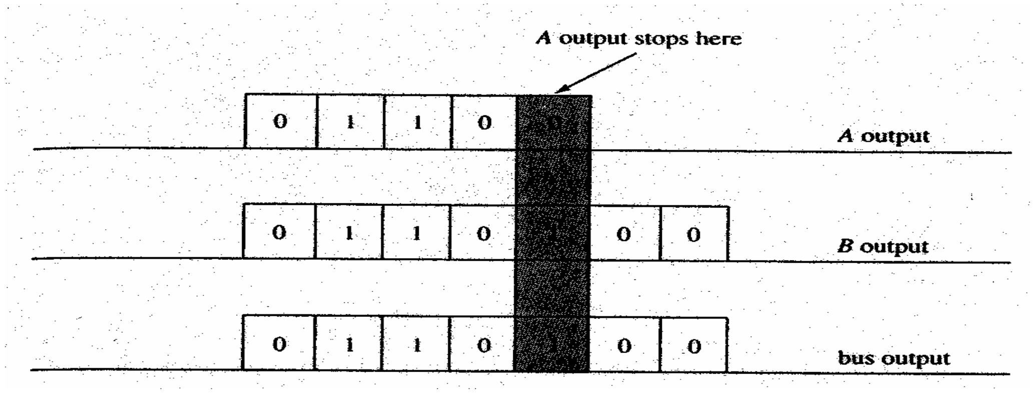
- During a message transmission the sending node keeps the busy signal *dominant*
- As long as a node perceives the busy signal *dominant*, it doesn't start a new transmission.

Real-Time (Paradigms) (66)

Bus-arbitration:

- A node starts sending if it detects no *dominant* signal on the busy-line for one bit-time.
- Then it starts sending a unique bit-string (1 = *dominant*, 0 =*recessive*) of predefined length (the binary coding (poll number) of the messages' priority) on the busy-line, one bit per bit-time
- At the same time it listens to the received value of the busy-line. If it receives a different value that the one it sends, it aborts arbitration
- If all bit-values have been transmitted successfully, it keeps the busy-line *dominant* and sends its message.
- Type CSMA/CD+CR ("Collision Detection and Collision Resolution")

What happens if two nodes A and B are starting arbitration simultaneously?



the message priority must be unique!--->

- Global assignment of priorities for each message and nodes a-priori or dynamically by a priority server
- Locally: Priorities are tuples (Prio, Node-id), only the Node-id is statically assigned and unique

Real-Time (Paradigms) (67)

Other solutions:

- deadline-driven scheme
- combined deadline-driven and priority scheme

Analysis of the Polled Bus Protocol

This Protocol implements a priority-based, non-preemptive resource access

Claim: A message with the highest priority is at most delayed for one message having maximal length.

Proof: a) If the busy-line is *recessive*, the message with the highest priority can be sent immediately as it will win any arbitration phase.

b) If the busy-line is *dominant*, the sending node has to wait for for the remainder of the ongoing transmission (i.e. at most one max. message length) until the busy-line becomes *recessive* again, then see a)

---> This algorithm is acceptable in case of a small bit-time of the bus

Is there any guarantee that can be given for messages with lower priorities?

Each message can be delayed by

a) a message already on the wire.

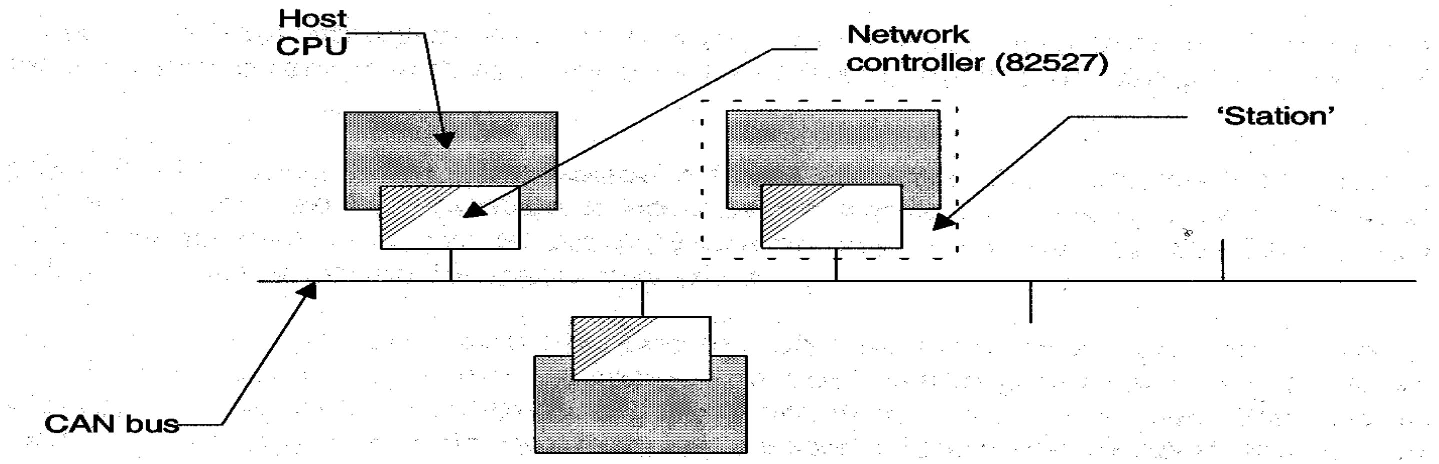
b) by any pending message with a higher priority.

Real-Time (Paradigms) (68)

Implementation of the Polled Bus Protocol on a CAN-Bus

CAN (Control Area Network) is a fieldbus designed by BOSCH. It has the character of a broadcast communication medium where a number of processors are connected to the bus via an controller interface.

CAN Architecture



Features of the CAN-Bus

- Priority (message-identifier) length 11 bit or 29 bit
- Data-length max. 8 bytes
- Bit-time 1µs - 0.1 ms => Bandwidth 1MB/s - 10kB/s
- Bus-Length 30 m - > 1 km

Aim is to bound the worst-case response time (latency) of a given hard real-time message type.

Real-Time (Paradigms) (69)

Theorem: Given that for each message *m* the maximal transmission time C_m and the minimal Period T_m is known, the worst-case response time R_m of a message m (defined as the longest time between the start of a task queuing m and the latest time m arrives at the destination stations) is bounded by:

$$R_m = C_m + w_m$$

where

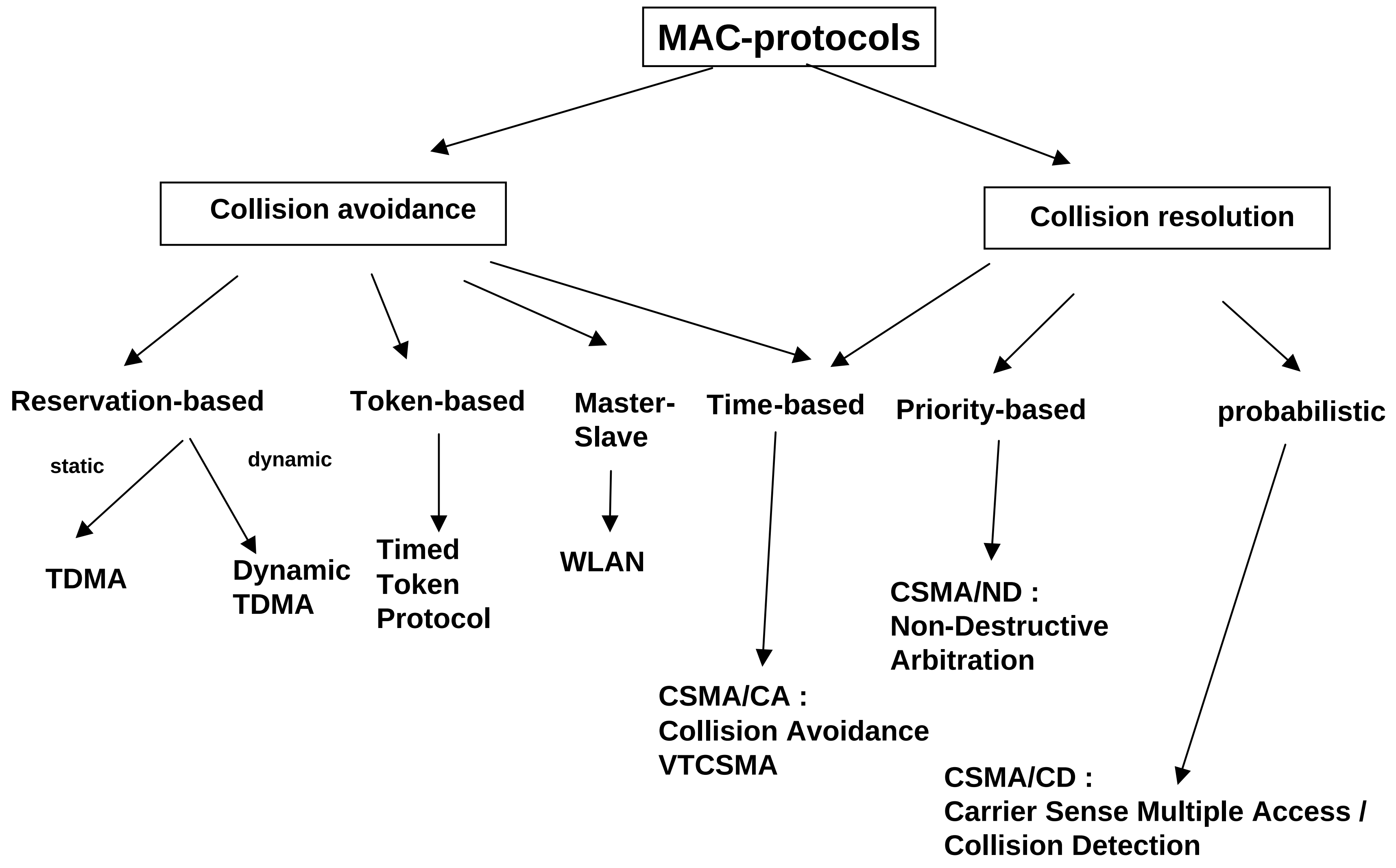
$$w_m = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{w_m}{T_j} + 1 \right\rceil C_j \quad \text{with} \quad B_m = \max_{\forall k \in lp(m)} (C_k)$$

hp(m) is the set of messages with a higher priority than m
lp(m) is the set of messages with a lower priority than m
w_m is the waiting time (for messages with higher priority) that m remains queued before it is transmitted on the medium
B_m is the time that m is blocked by a lower priority message on the medium that started before m became queued

---> For each message a worst case delay can be computed!

Real-Time (Paradigms) (70)

Taxonomy of Medium Access Control - Protocols:


MAC-protocols
Collision avoidance
Collision resolution
Reservation-based
Token-based
Master-Slave
Time-based
Priority-based
probabilistic
static
dynamic
TDMA
Dynamic TDMA
Timed Token Protocol
WLAN
CSMA/CA : Collision Avoidance VTCSMA
CSMA/ND : Non-Destructive Arbitration
CSMA/CD : Carrier Sense Multiple Access / Collision Detection