technische universität
dortmund

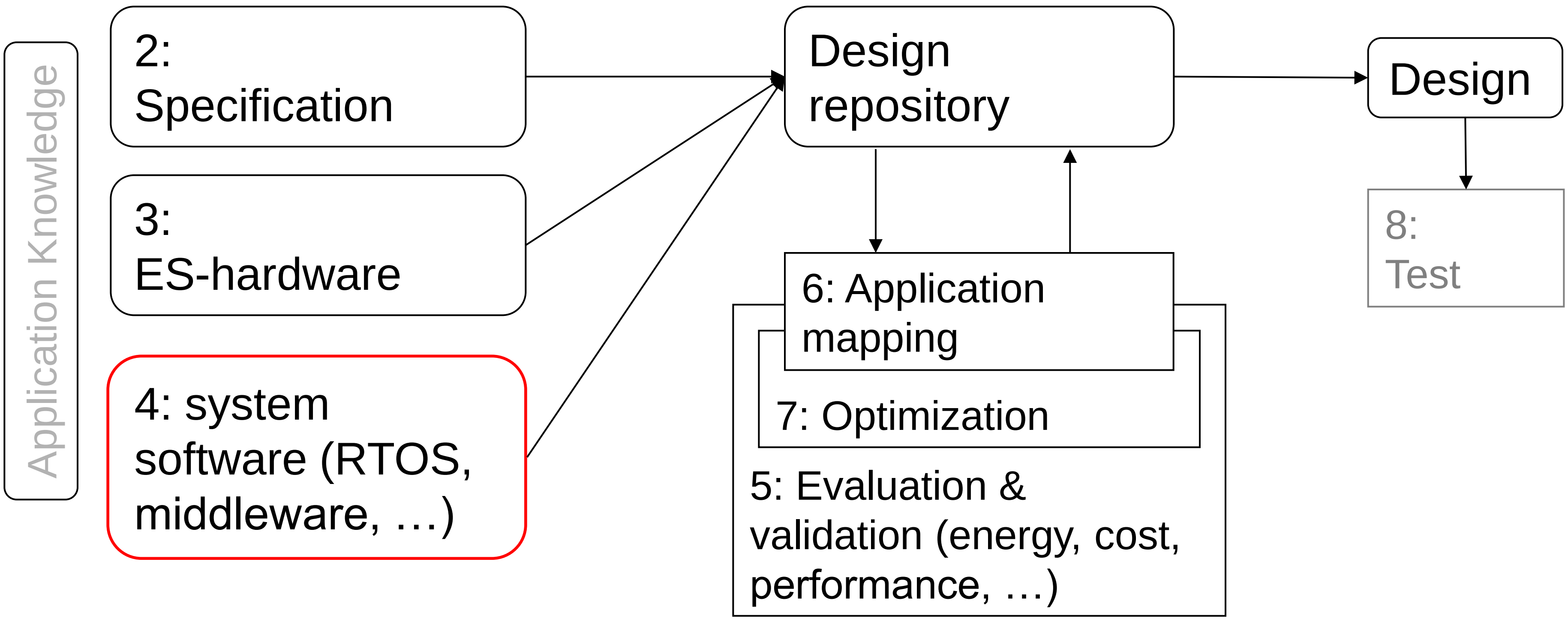
fakultät für
informatik

System Software (2)

Peter Marwedel
TU Dortmund, Informatik 12
Germany

2013年 11 月 26 日

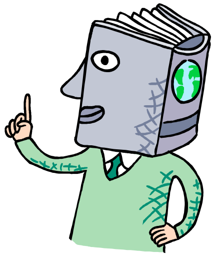
Structure of this course



Numbers denote sequence of chapters

Increasing design complexity + Stringent time-to-market requirements ☞Reuse of components

Reuse requires knowledge from previous designs to be made available in the form of **intellectual property** (IP, for **SW & HW**).



- HW
- Operating systems
- Middleware (Communication libraries, data bases, …)
- ….

Priority Inheritance Protocol (PIP)
☞ Priority Ceiling Protocol (PCP)

The Priority Inheritance Protocol (PIP)

- does not prevent deadlocks
- can lead to chained blocking
  - (Several lower priority tasks can block a higher priority task)
- and has inherent static priorities of tasks

☞ The Priority Ceiling Protocol (PCP)

- avoids multiple blocking
- guarantees that, once a task has entered a critical section, it cannot be blocked by lower priority tasks until its completion.

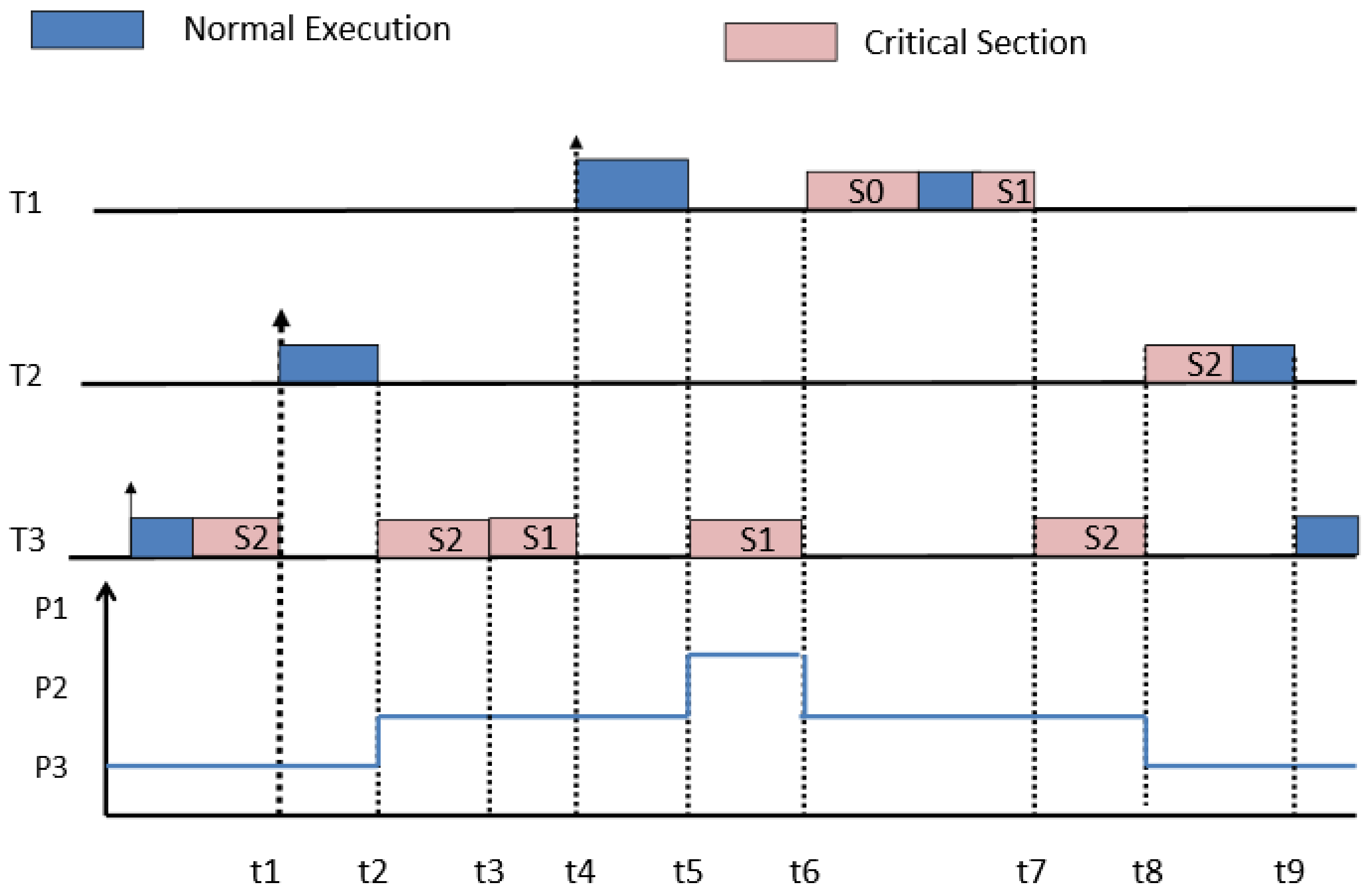
Source: http://www.ida.liu.se/~unmbo/RTS_CUGS_files/Lecture3.pdf

PCP

- A task is not allowed to enter a critical section if there are already locked semaphores which could block it eventually
- Hence, once a task enters a critical section, it can not be blocked by lower priority tasks until its completion.
- This is achieved by assigning priority ceiling.
- Each semaphore S_k is assigned a **priority ceiling** $C(S_k)$.
  It is the priority of the highest priority task that can lock S_k.
  This is a static value.

Source: http://www.ida.liu.se/~unmbo/RTS_CUGS_files/Lecture3.pdf

Priority Ceiling: Example



C(S0=P1) C(S1=P1) C(S2=P2)

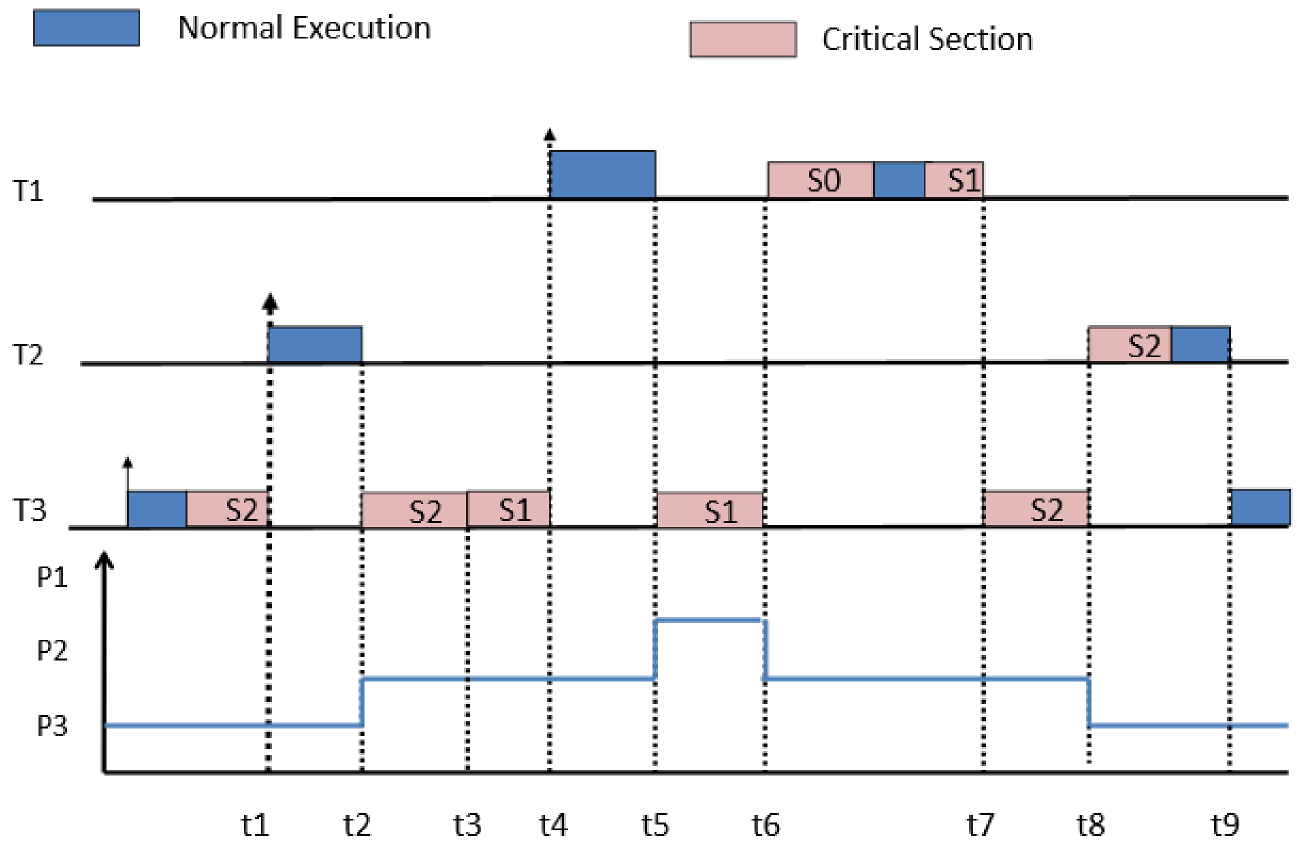
Source: http://www.ida.liu.se/~unmbo/RTS_CUGS_files/Lecture3.pdf

PCP

- Suppose T is running and wants to lock semaphore S_k.
- T is allowed to lock S_k only if priority of T > priority ceiling $C(S^*)$ of the semaphore S^* where:
  - S^* is the semaphore with the highest priority ceiling among all the semaphores which are currently locked by jobs other than T.
  - In this case, T is said to blocked by the semaphore S^* (and the job currently holding S^*)
  - When T gets blocked by S^* then the priority of T is transmitted to the job T that currently holds S^*

Source: http://www.ida.liu.se/ ~unmbo/RTS_CUGS_files/Lecture3.pdf

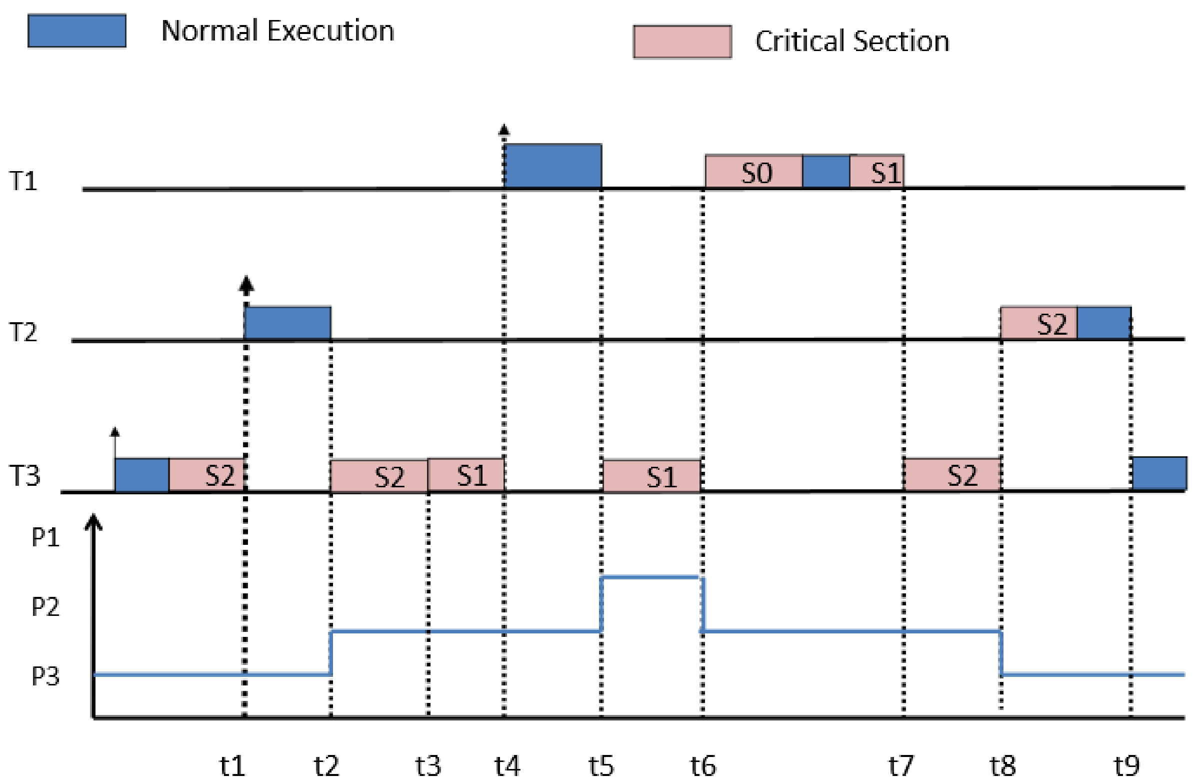
PCP: An Example



t2: T2 can not lock S2. Currently T3 is holding S2 and C(S2) = P2 and the current priority of T2 is also P2

Source: http://www.ida.liu.se/ ~unmbo/RTS_CUGS_files/Lecture3.pdf

PCP: An Example



t5 : T1 can not lock S0. Currently T3 is holding S2 and S1 and C(S1) = T1 and the current priority of T1 is also P1. The (inherited) priority of T3 is now P1

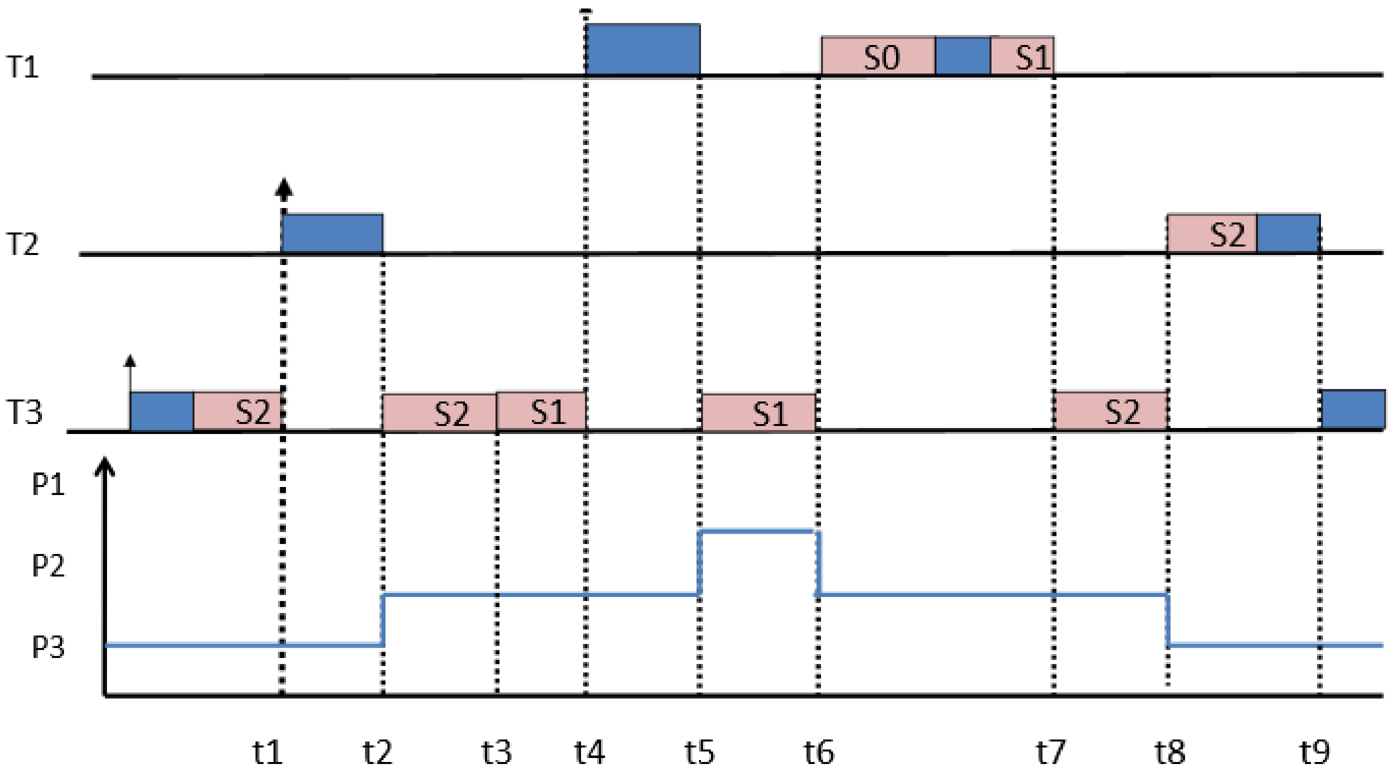
Source: http://www.ida.liu.se/ ~unmbo/RTS_CUGS_files/Lecture3.pdf

PCP

- When T^* leaves a critical section guarded by S^* then it unlocks S^* and the highest priority job, if any, which is blocked by S^* is awakened
- The priority of T^* is set to the highest priority of the job that is blocked by some semaphore that T^* is still holding.
  If none, the priority of T^* is set to be its nominal one.

Source: http://www.ida.liu.se/ ~unmbo/RTS_CUGS_files/Lecture3.pdf

PCP: Example



Source: http://www.ida.liu.se/~unmbo/RTS_CUGS_files/Lecture3.pdf

t6 : T3 unlocks S1. It awakens T1. But T3s (inherited) priority is now only P2 while P1>C(S2) =P2. So T1 preempts T3 and runs to completion.

t7: T3 resumes execution with priority P2

t8 : T3 unlocks S2, goes back to its priority P3. T2 preempts T3, runs to completion

PCP: Example (1)

Task name	T	Priority
A	50	10
B	500	9

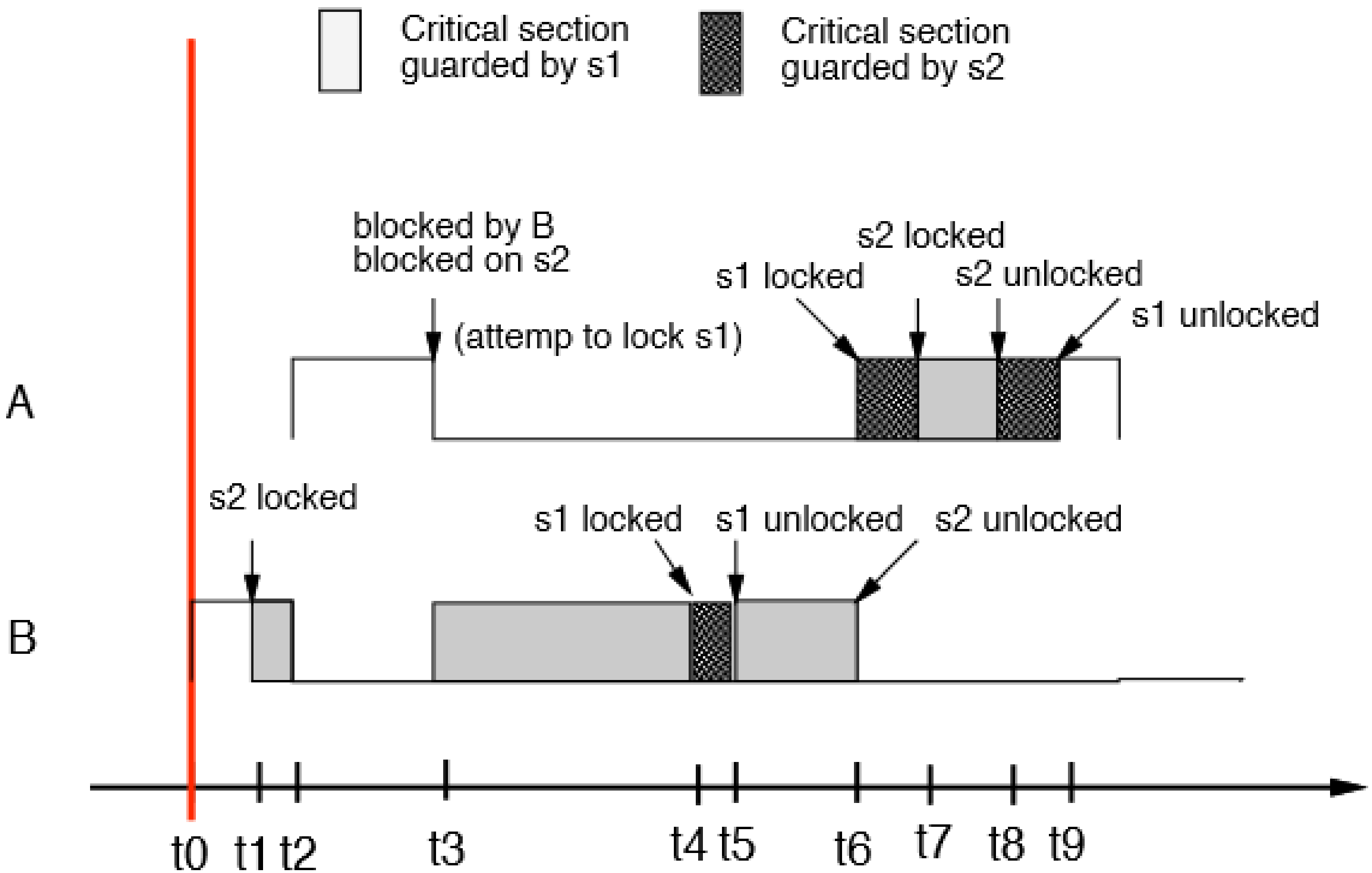
```
Task A          Task B

lock(s1)        lock(s2)
lock(s2)        lock(s1)
...             ...
unlock(s1)      unlock(s1)
unlock(s2)      unlock(s1)
```

$$ceil(s_1) = 10, ceil(s_2) = 10$$

Source: Lund University, course EDA 040,
http://fileadmin.cs.lth.se/cs/Education/EDA040/lecture/RTP-F6b.pdf

PCP: Example (2)



See http://fileadmin.cs.lth.se/cs/Education/EDA040/lecture/RTP-F6b.pdf for detailed explanation

PCP: Properties

- deadlock free (only changing priorities)
- a given task i is delayed at most once by a lower priority task
- the delay is a function of the time taken to execute the critical section
- Certain variants as to when the priority is changed

Extending PCP: Stack Resource Policy (SRP)

- SRP supports dynamic priority scheduling
- SRP blocks the task at the time it attempts to preempt.
- **Preemption level** l_i of task i: decreasing function of deadline (larger deadline ☞ easier to preempt) (Static)
- **Resource ceiling**: of a resource is the highest preemption level from among all tasks that may access that resource (Static)
- **System ceiling**: is the highest resource ceiling of all the resources which are currently blocked (dynamic, changes with resource accesses)

Source: http://www.ida.liu.se/~unmbo/RTS_CUGS_files/Lecture3.pdf

SRP Policy

A task can preempt another task if

- it has the highest priority
- and its preemption level is higher than the system ceiling

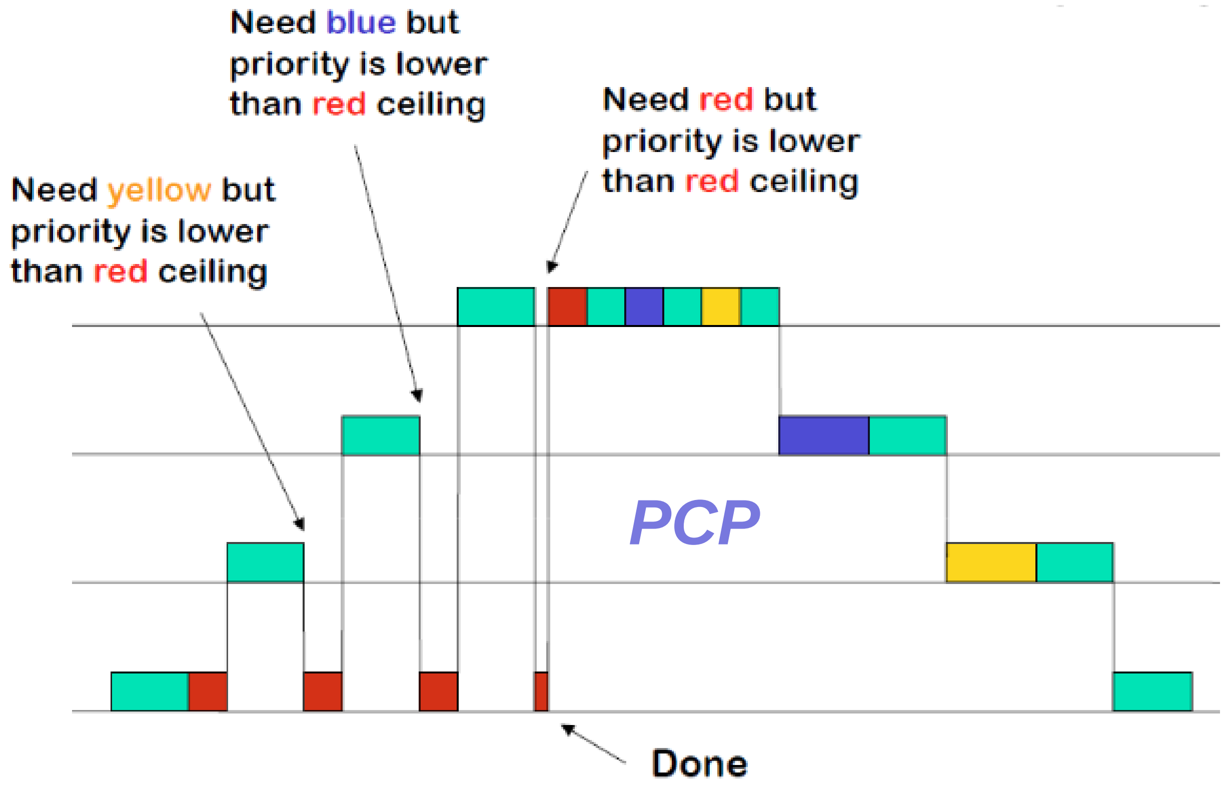
A task is not allowed to start until the resources currently available are sufficient to meet the maximum requirement of every task that could preempt it.

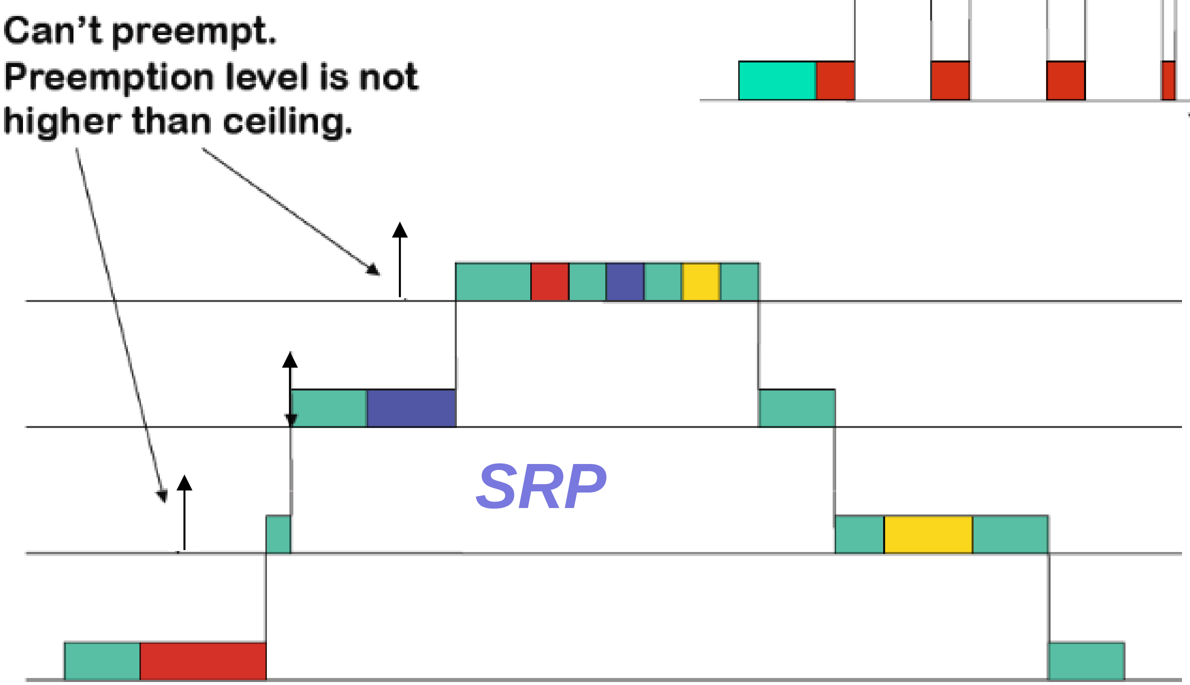
Why ***Stack*** Resource Policy? Tasks cannot be blocked by tasks with lower l_i, can resume only when the task completes.
Tasks on the same l_i can share stack space.
More tasks on the same l_i ☞ higher stack space saving.

Source: http://www.ida.liu.se/~unmbo/RTS_CUGS_files/Lecture3.pdf

SRP vs. PCP

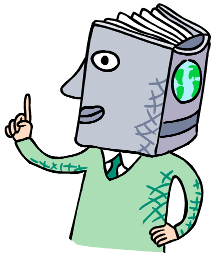
Less preemptions for SRP





Source: http://www.ida.liu.se/ ~unmbo/RTS_CUGS_files/ Lecture3.pdf

Increasing design complexity + Stringent time-to-market requirements ☞Reuse of components



Reuse requires knowledge from previous designs to be made available in the form of **intellectual property** (IP, for **SW & HW**).

- HW
- Operating systems
- Middleware (Communication libraries, data bases, …)
- ….

Models of computation considered in this course

Communication/ local computations	Shared memory	Message passing Synchronous	Asynchronous
Undefined components	Plain text, use cases	(Message) sequence charts	
Communicating finite state machines	StateCharts		SDL
Data flow	Scoreboarding + Tomasulo Algorithm (☞ Comp.Archict.)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, …	
Discrete event (DE) model	VHDL*, Verilog*, SystemC*, …	Only experimental systems, e.g. distributed DE in Ptolemy	
Imperative (Von Neumann) model	C, C++, Java [libraries]	C, C++, Java with libraries CSP, ADA	

* Classification based on semantic model

Pthreads

- **Shared memory model**
- Consists of standard API
  - Originally used for single processor
  - Locks (mutex, read-write locks)

Based on W. Verachtert (IMEC): *Introduction to Parallelism,* tutorial, DATE 2008

PThreads Example

```
threads = (pthread_t *) malloc(n*sizeof(pthread_t));
pthread_attr_init(&pthread_custom_attr);
for (i=0;i<n; i++)
  pthread_create(&threads[i],
&pthread_custom_attr, task, …);
for (i=0;i<n; i++) {
  pthread_mutex_lock(&mutex);
  <receive message>
  pthread_mutex_unlock(&mutex);
}
for (i=0;i<n; i++)
  pthread_join(threads[i], NULL);
```

```
void* task(void *arg) {
  …
  pthread_mutex_lock(&mutex);
  <send message>
  pthread_mutex_unlock(&mutex);
  return NULL
}
```

Based on W. Verachtert (IMEC): *Introduction to Parallelism,* tutorial, DATE 2008

Pthreads

- Consists of standard API
  - Locks (mutex, read-write locks)
  - Condition variables
  - Completely explicit synchronization
  - Synchronization is very hard to program correctly
- Typically supported by a mixture of hardware (shared memory) and software (thread management)
- Exact semantics depends on the memory consistency model
- Support for efficient producer/consumer parallelism relies on murky parts of the model
- Pthreads can be used as back-end for other programming models (e.g. OpenMP)

Based on W. Verachtert (IMEC): *Introduction to Parallelism,* tutorial, DATE 2008

OpenMP

Implementations target shared memory hardware

Parallelism expressed using pragmas

- Parallel loops
  (`#pragma omp for {…}` ;focus: data parallelism)
- Parallel sections
- Reductions

Explicit

- Expression of parallelism (mostly explicit)

Implicit

- Computation partitioning
- Communication
- Synchronization
- Data distribution

Based on W. Verachtert (IMEC): *Introduction to Parallelism,* tutorial, DATE 2008

Lack of control over partitioning can cause problems

Models of computation considered in this course

Communication/ local computations	**Shared memory**	**Message passing Synchronous**	**| Asynchronous**
Undefined components	Plain text, use cases	| (Message) sequence charts	
Communicating finite state machines	StateCharts		SDL
Data flow	(Not useful)°		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, …	
Discrete event (DE) model	VHDL*, Verilog*, SystemC*, …	Only experimental systems, e.g. distributed DE in Ptolemy	
Imperative (Von Neumann) model	C, C++, Java [libraries]	C, C++, Java with libraries CSP, ADA |	

* Classification based on semantic model

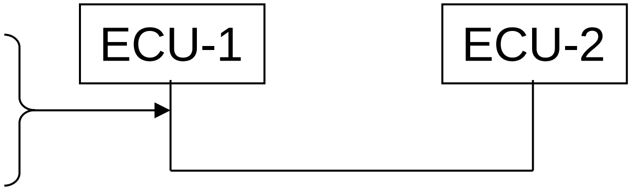
° Somewhat related: Scoreboarding + Tomasulo-Algorithm

OSEK/VDX COM

OSEK/VDX COM

- is a special communication standard for the OSEK automotive OS Standard
- provides an “Interaction Layer” as an API for internal and external communication via a “Network Layer” and a “Data Link” layer (some requirements for these are specified)
- specifies the functionality, it is not an implementation.

© P. Marwedel, 2011

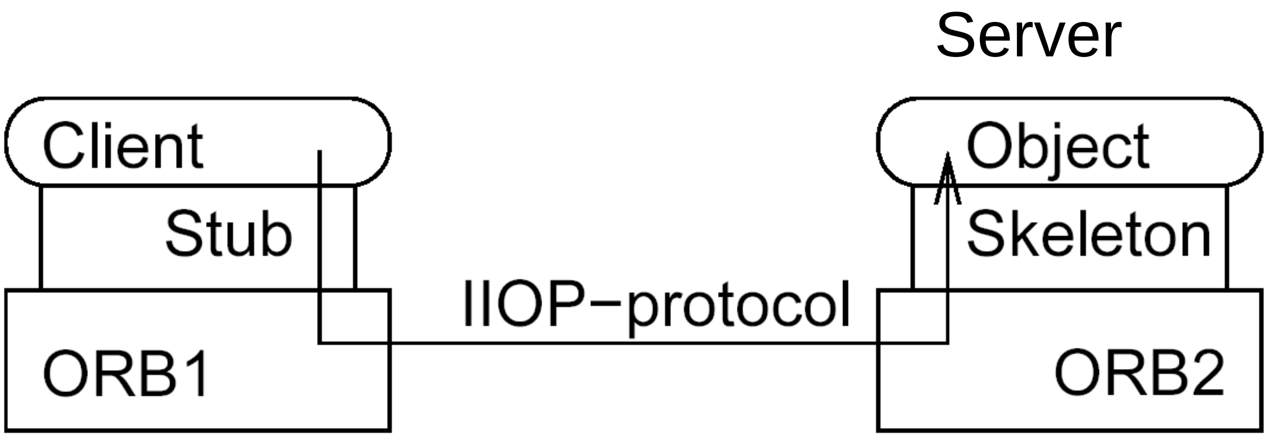


CORBA (Common Object Request Broker Architecture)

Software package for access to remote objects;

Information sent to Object Request Broker (ORB) via local stub.

ORB determines location to be accessed and sends information via the IIOP I/O protocol.

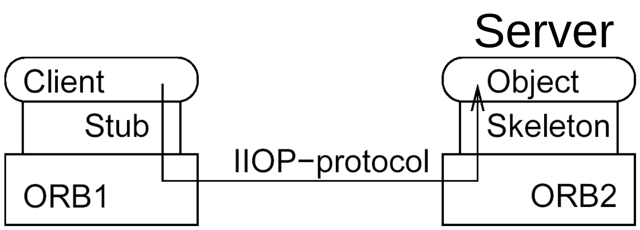


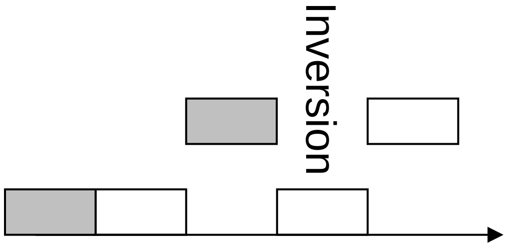
Access times unpredictable.

Real-time (RT-) CORBA

RT-CORBA

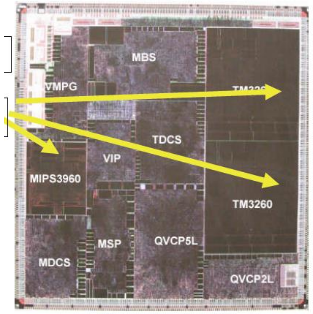
- provides *end-to-end predictability of timeliness in a fixed priority system*.
- *respects thread priorities between client and server for resolving resource contention*,
- provides thread priority management,
- provides priority inheritance,
- bounds latencies of operation invocations,
- provides pools of preexisting threads.





Message passing interface (MPI)

- Asynchronous/synchronous message passing
- Designed for high-performance computing
- Comprehensive, popular library
- Available on a variety of platforms
- Mostly for homogeneous multiprocessing
- Considered for MPSoC programs for ES;
- Includes many copy operations to memory (memory speed ~ communication speed for MPSoCs); Appropriate MPSoC programming tools missing.



http://www.mhpcc.edu/training/workshop/mpi/MAIN.html#Getting_Started

MPI (1)

Sample blocking library call (for C)**:**

- MPI_Send(*buffer,count,type,dest,tag,comm*) where
  - *buffer*: Address of data to be sent
  - *count*: number of data elements to be sent
  - *type*: data type of data to be sent (e.g. MPI_CHAR, MPI_SHORT, MPI_INT, …)
  - *dest*: process id of target process
  - *tag*: message id (for sorting incoming messages)
  - *comm*: communication context = set of processes for which destination field is valid
  - function result indicates success

http://www.mhpcc.edu/training/workshop/mpi/MAIN.html#Getting_Started

MPI (2)

Sample non-blocking library call (for C)**:**

- MPI_Isend(*buffer,count,type,dest,tag,comm,request*) where
  - *buffer … comm:* same as above
  - *request*: unique "request number". "handle" can be used (in a WAIT type routine) to determine completion

http://www.mhpcc.edu/training/workshop/mpi/MAIN.html#Getting_Started

Evaluation

Explicit

- Computation partitioning
- Communication
- Data distribution

Implicit

- Synchronization (implied by communic., explicit possible)
- Expression of parallelism (implied)
- Communication mapping

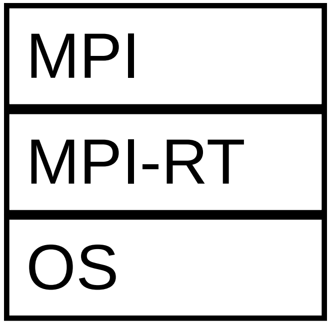
Properties

- Most things are explicit
- Lots of work for the user ("*assembly lang. for parallel prog*.")
- doesn't scale well when # of processors is changed heavily

Based on W. Verachtert (IMEC): *Introduction to Parallelism,* tutorial, DATE 2008

RT-issues for MPI

- MPI/RT: a real-time version of MPI [MPI/RT forum, 2001].
- MPI-RT does not cover issues such as thread creation and termination.
- MPI/RT is conceived as a potential layer between the operating system and standard (non real-time) MPI.



Universal Plug-and-Play (UPnP)

- Extension of the plug-and-play concept
- *Enable emergence of easily connected devices & simplify implementation of networks @ home & corporate environments!*
- Examples: Discover printers, storage space, control switches in homes & offices
- Exchanging data, no code (reduces security hazards)
- Agreement on data formats & protocols
- Classes of predefined devices (printer, mediaserver etc.)
- http://upnp.org


© P. Marwedel, 2012

Devices Profile for Web Services (DPWS)

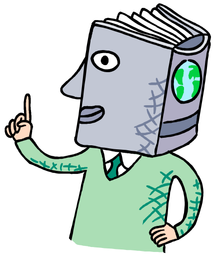
- More general than UPnP
- *… DPWS defines a minimal set of implementation constraints to enable secure Web Service messaging, discovery, description, and eventing on resource-constrained devices. …*
- *DPWS specifies a set of built-in services:*
  - *Discovery services …*
  - *Metadata exchange services…*
  - *Publish/subscribe eventing services…*
- *Lightweight protocol, supporting dynamic discovery, … its application to automation environments is clear.*

Network Communication Protocols
- e.g. JXTA -

- *Open source peer-to-peer protocol specification.*
- *Defined as a set of XML messages that allow any device connected to a network to exchange messages and collaborate independently of the network topology.*
- *.. Can be implemented in any modern computer language.*
- *JXTA peers create a virtual overlay network, allowing a peer to interact with other peers even when some of the peers and resources are behind firewalls and NATs or use different network transports.*

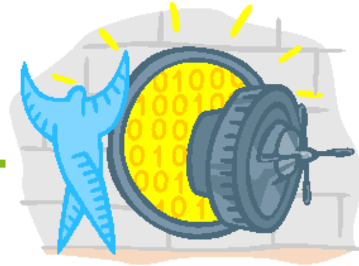
Increasing design complexity + Stringent time-to-market requirements ☞Reuse of components

Reuse requires knowledge from previous designs to be made available in the form of **intellectual property** (IP, for **SW & HW**).



- HW
- Operating systems
- Middleware (Communication libraries, data bases, …)
- ….

Data bases



Goal: store and retrieve persistent information

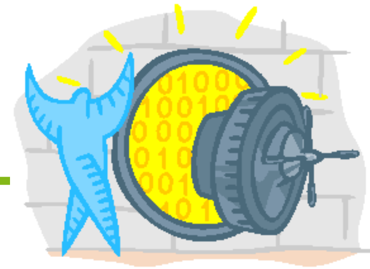
Transaction= sequence of read and write operations

Changes not final until they are committed

Requested ("ACID") properties of transactions

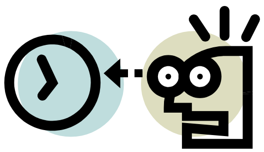
1. **Atomic:** state information as if transaction is either completed or had no effect at all.
2. **Consistent:** Set of values retrieved from several accesses to the data base must be possible in the world modeled.
3. **Isolation:** No user should see intermediate states of transactions
4. **Durability:** results of transactions should be persistent.

Real-time data bases



Problems with implementing real-time data bases:

1. transactions may be aborted various times before they are finally committed.



2. For hard discs, the access times to discs are hardly predictable.

Possible solutions:

1. Main memory data bases
2. Relax ACID requirements

Source: Krishna, Shin, 1997

Summary

- Communication middleware
  - Pthreads
  - OpenMP
  - OSEK/VDX COM
  - CORBA
  - MPI
  - JXTA
  - DPWS
- RT-Data bases (brief)

RESERVE

Priority Ceiling Protocol (PCP)

Restrictions on how we can lock (Wait, EnterMonitor) and unlock (Signal, LeaveMonitor) resources:

- a task must release all resources between invocations
- the computation time that a task *i* needs while holding semaphore s is bounded. $cs_{i,s}$ = the time length of the critical section for task i holding semaphore s
- a (fixed set of) tasks may only lock semaphores from a fixed set of semaphores known a priory. $uses(i)$=the set of semaphores that may be used by task i

L. Sha, R. Rajkumar, J. Lehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization, IEEE Transactions on Computers, Vol. 39, No. 9, 1990

Source: Lund University, course EDA 040, http://fileadmin.cs.lth.se/cs/Education/EDA040/lecture/RTP-F6b.pdf

PCP: the protocol

- The ceiling of a semaphore, $ceil(s)$, is the priority of the highest priority task that uses the semaphore
- $pri(i)$ is the priority of task i
- At run-time:
  - a task i can only lock a semaphore s, if $pri(i)$ > ceilings of all semaphores currently locked by other tasks
  - if $\neg$ ($pri(i)$ > ceilings of all …): task i will be blocked (task i is said to be blocked on the semaphore, S^*, with the highest priority ceiling of all semaphores currently locked by other jobs and task i is said to be blocked by the task that holds S^*)
  - when task i is blocked on S^*, the task currently holding S^* inherits the priority of task i

Source: Lund University, course EDA 040, http://fileadmin.cs.lth.se/cs/Education/EDA040/lecture/RTP-F6b.pdf