/Zöbel95/

4.1 Characteristics of RTOS (Realtime Operating Systems)

4.2 The Kernel

4.3 Synchronisation and priority inheritance

4.4 Driver

What is a RT OS (Realtime Operating System)?

First: **what is an OS**?

DIN 44300:

Die Programme eines digitalen Rechners, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten das digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.

"The programs of a computer which together with the HW form the base of the computer system, in particular control the execution of programs."

ANS(American National Standard):

Software which controls the execution of programs and which may provide scheduling, debugging, input/output control, accounting, compilation, storage assignment, data management and related services.

What is a RT OS (Realtime Operating System)?

First: **what is an OS?** You remember?!

Definitionsversuche

DIN 44300:

Die Programme eines digitalen Rechners, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten das digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.

ANS(American National Standard):

Software which controls the execution of programs and which may provide scheduling, debugging, input/output control, accounting, compilation, storage assignment, data management and related services.

Unterschiede?

BSS1 1.3

/Zöbel95/

4.1 Characteristics of RTOS (Realtime Operating Systems)

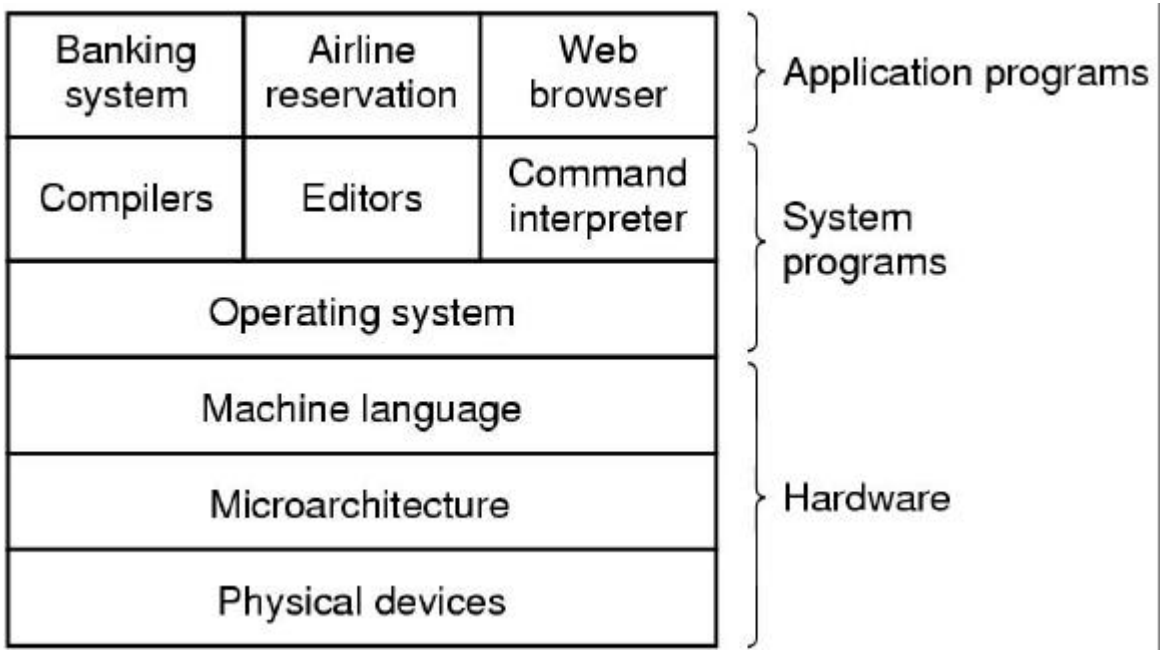
4.2 The Kernel

4.3 Synchronisation and priority inheritance

4.4 Driver

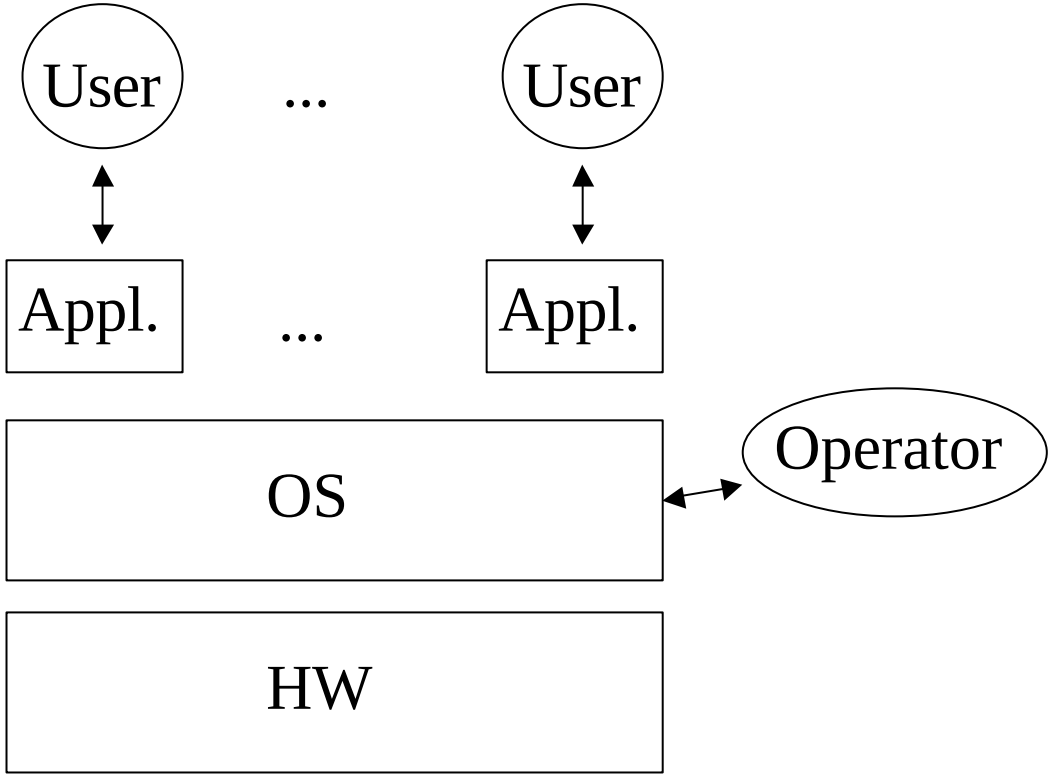
OS (Operating System) is extended machine and resource manager

First: **what is an OS**? /Tanenbaum02/:



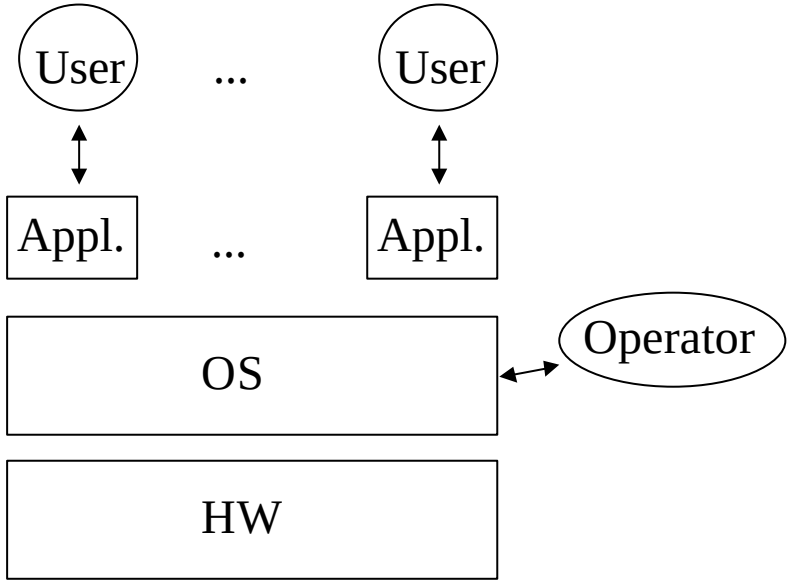
- It is an extended machine
  - Hides the messy details which must be performed
  - Presents user with a virtual machine, easier to use
- It is a resource manager
  - Each program gets time with the resource
  - Each program gets space on the resource

OS (Operating System) is extended machine



OS (Operating System) is extended machine - you remember?

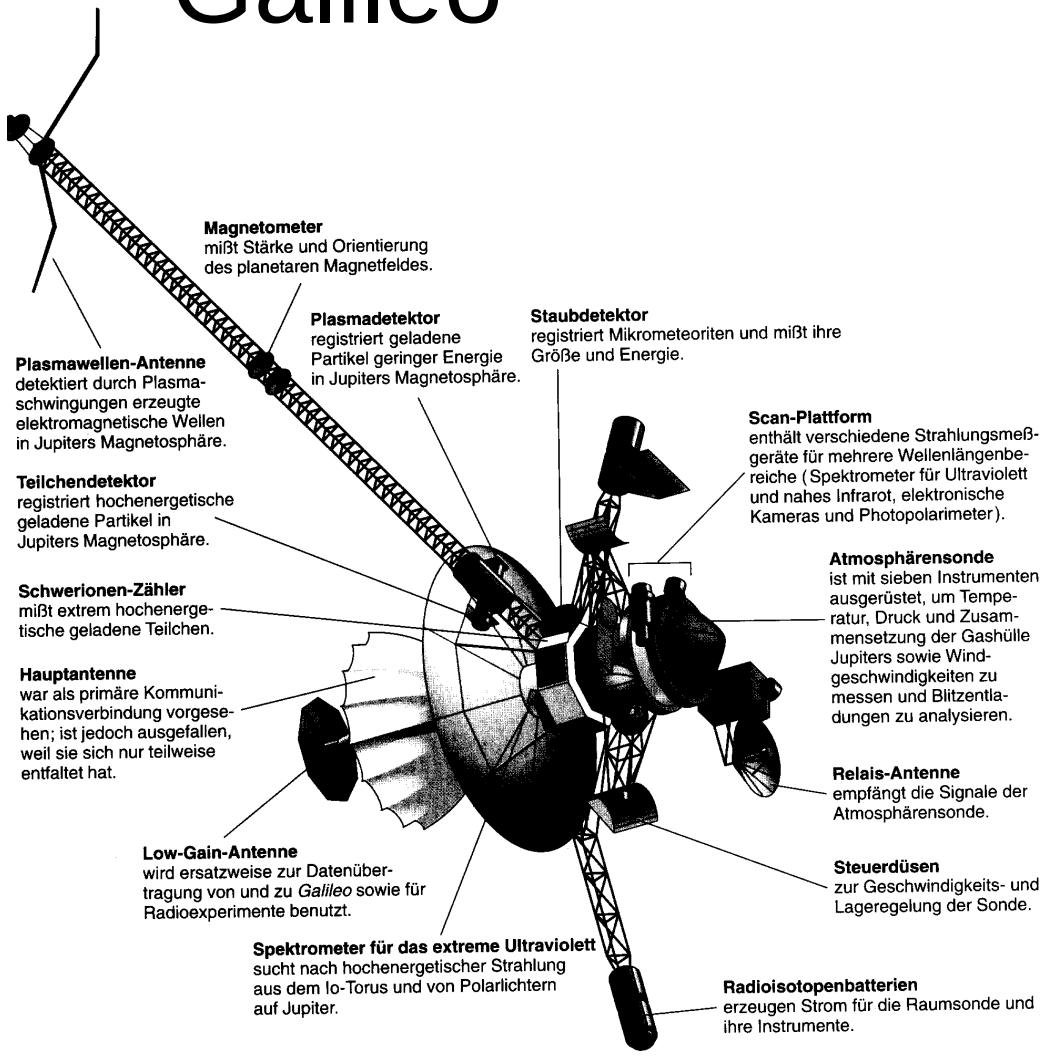
Aufgabe: Schnittstelle SW-HW



Applikationen sollten auf unterschiedlicher HW laufen (z.B. in INTEL-Welt und auf MACs), also HW-unabhängig sein.
Damit müssen HW-abhängige Dienste (z.B. CPU selber, Hauptspeicher, IO-Devices) über das OS ´vermittelt´ werden.
Vgl. ANS: control program.

OS (Operating System) is extended machine - OS can save HW!

Galileo

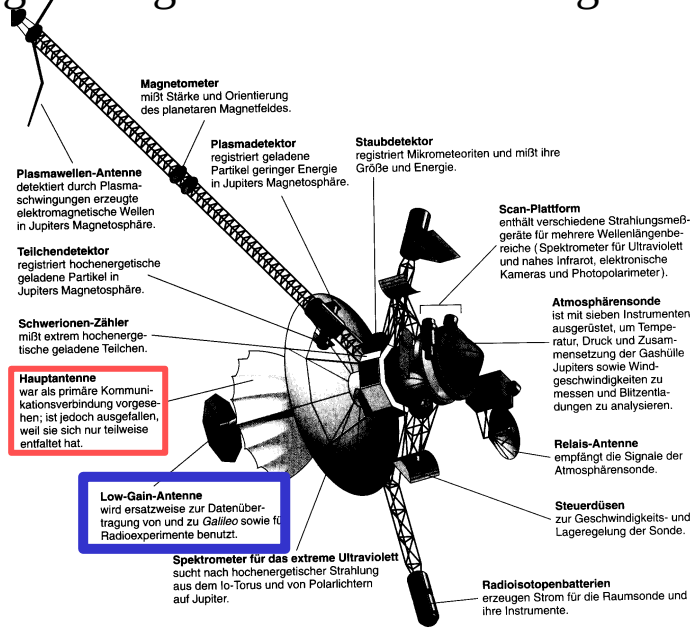


Design:	1976
Start (1.plan):	1/82
Start (2.plan):	5/86
Challenger:	1/86
Start (real):	18.10.1989
Jupiter:	07.12.1995
Io, Ganymed, Kallisto, Europa	
End(Plan):	07.12.1997
Shoemaker-Levy ...	
Killed:	22.09.2003

OS (Operating System) is extended machine - you remember?

HW vs. OS

Entwicklung von HW (insbesondere CPUs) und Betriebssystemen haben sich gegenseitig stark beeinflusst – vgl. z.B. Intel ? MS ´Hochschaukeln´



weiteres Beispiel: **Galileo**

Design:	1976
Start (1.Plan):	1/82
Start (2.Plan):	5/86
Challenger:	1/86
Start (real):	18.10.1989
Jupiter:	07.12.1995

Io, Ganymed, Kallisto, Europa

Ende(Plan):	07.12.1997

Shoemaker-Levy ...

absichtlich verglüht: 22.09.2003

OS (Operating System) is resource manager

- It is a resource manager
  - –Each program gets time with the resource (i.e. CPU itself, see "RT scheduling")
  - –Each program gets space on the resource (on RAM, on disk + contents!)

  and
    - IO devices (keyboard, mouse, screen,...)
    - Communication lines (internal/external: bus, modem, network,...)
    - Other resources as timers, semaphores, monitors,...

  for all corresponding parties (i.e users and processes) in a fair, effective, safe, simple, ... way.

OS (Operating System) is resource manager

Aufgabe: Betriebsmittelverwaltung

Betriebsmittel (Ressourcen), die zum Ablauf einer Applikation erforderlich sind und natürlich auch vom OS selber benötigt werden:

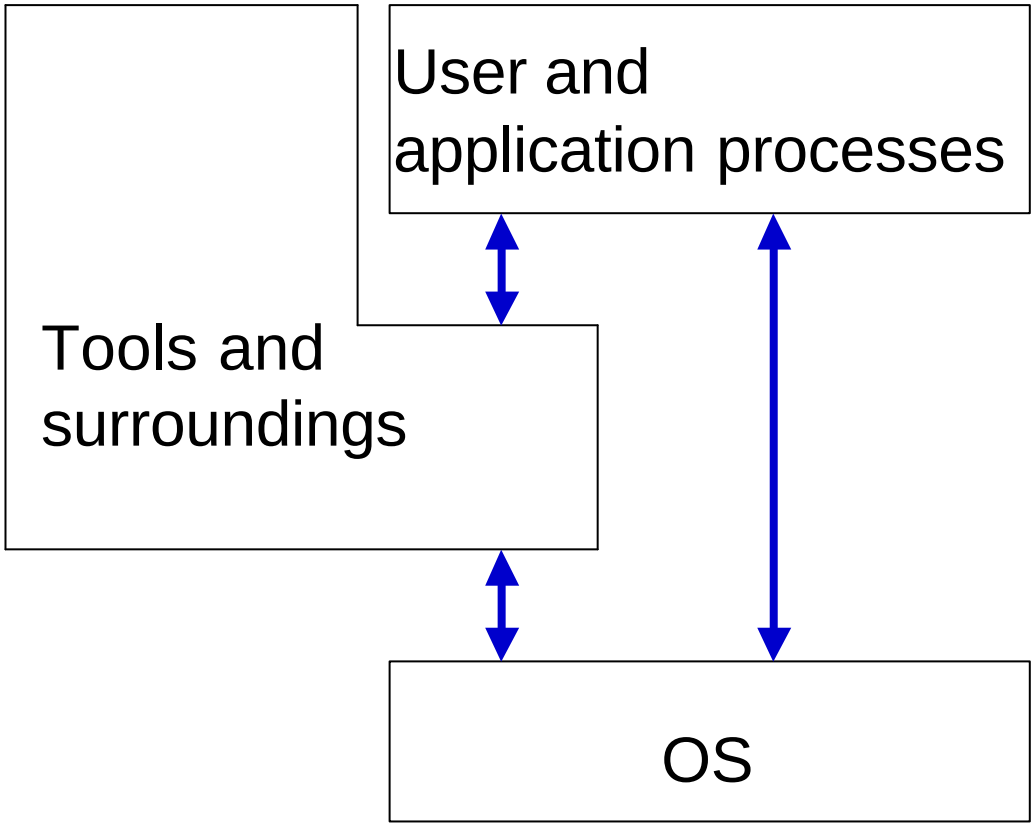
- CPU selber (natürlich!)
- Arbeitsspeicher (mit Inhalt: Programme selber, Daten, Arbeitsbereiche)
- Sekundärspeicher (´Platte´, vgl. MS-DOS: Disk Operating System)
- IO-Devices(Ein- und Ausgabegeräte: Tastatur, Maus, Bildschirm,...)
- Kommunikationsverbindungen (intern und extern: Bus, Modem, Netz,...)
- evtl. Timer, Semaphore, Monitore,...

OS verwaltet diese Ressourcen, entscheidet bei konkurrierenden Anforderungen, wer in welcher Reihenfolge was (Stichwort: Kooperation und Konkurrenz) bekommt (möglichst fair, unter Beachtung Gesamtperformance, ...). Stellt einmalige Ressourcen (s.o., zusätzlich Peripherie) allen zur Verfügung.

You know it already!

BSS1 1.9

A third aspect of OS: tools and surrounding



E.g.:

- tools for program development
  - compiler
  - editor
  - test tools
- GUIs
- communication services
- databases and archives

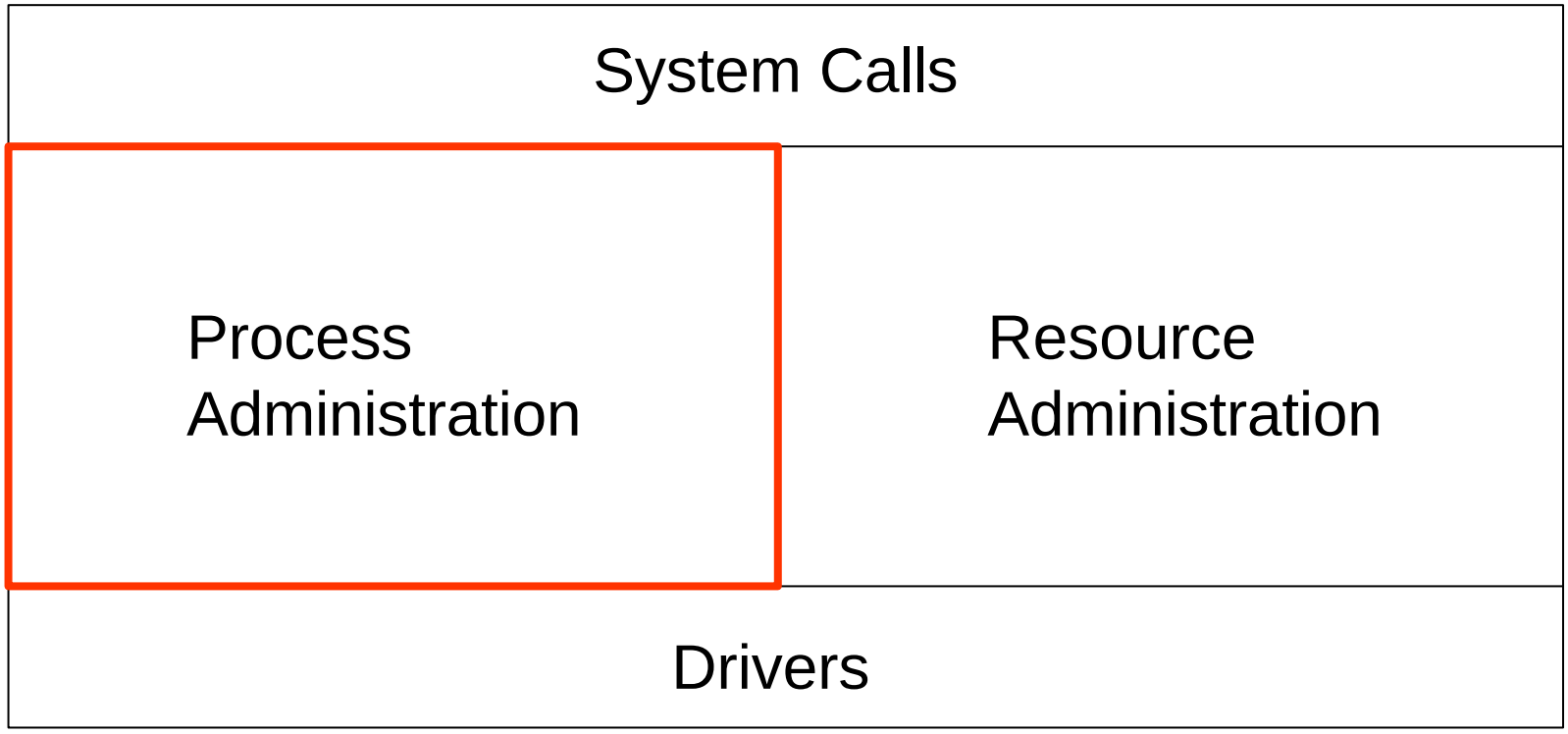
The acceptance of an OS is strongly dependent on such tools!

UNIX "tools" demanded by POSIX

Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

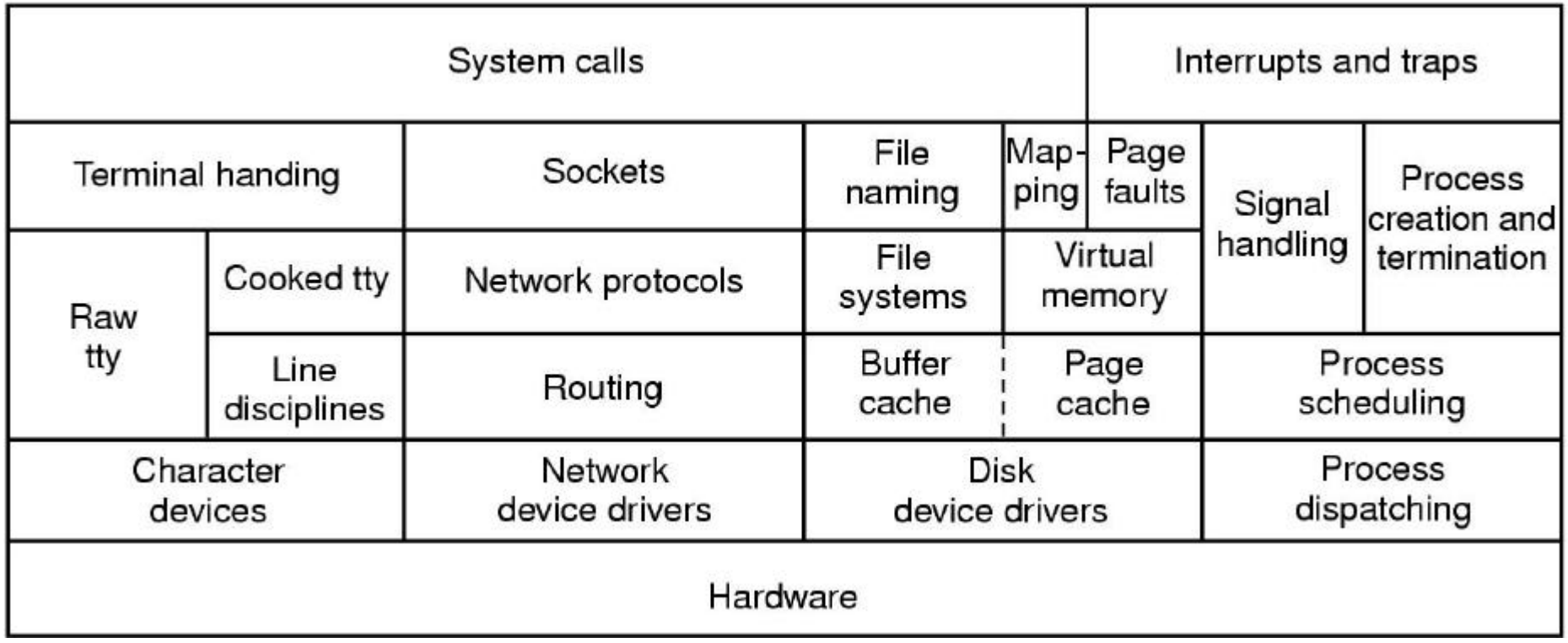
/Tanenbaum02/

OS fine structure



Kernel

OS fine structure



Structure of 4.4BSD kernel

/Tanenbaum02/

RTOS vs. OS

RT demands:

- formulate and control timing conditions
- predictability of system behaviour

are also RTOS demands, so - in addition to "normal" OS tasks - a RTOS must:

- mask interrupts of different urgency
- give user ways to priorise processes (at least with priorities)
- give user timing operations (e.g. alarm clock for processes) with sufficient granularity
- OS kernel (or at least all other processes) should be preemptive
- support asynchronous I/O operations
- support access to disk controller to have predictable access to disk files
- support virtual memory administration (i.e. paging) including blocking of this
- support network access in a predictable way (no ethernet!)

RTOS zoo

RT applications with no OS at all (remember speedometer!) or rudamental OS: real time executives, was e.g. base for VRTX (_32, _a, _mc).

RTOS designed (only) for RT systems (tasks see slide before+disk and memory admin), e.g. iRMX, EUROS, VOCOS.

Extensions of standard OS with RT properties:
MS-DOS based (sic!): RTKernel, AMX.
UNIX based: REAL/IX, SORIX, AIX, LynxOS , embedded Linux.

All purpose (including RT) OS(*): Mach, OSF/1, Solaris.

Somewhere inbetween(*): VxWorks, pSOS, COSMOS, OS-9, OSE, PXR, RMOS, QNX, Windows CE(?), ...

RTOS extensions for UNIX - standardisation efforts

SVID (system V interface definition)
source code of system calls based on UNIX System V

POSIX (portable operating system interface for computer environments)
IEEE (institute of electronical and electronics engineers) defines interface between user processes and a virtual OS (not necessarily UNIX!)

OSF (open software foundation)
based on standardised(!) UNIX: extensions, tools etc. as
OSF/1, OSF/Motif, OSF/DCE

POSIX - see working groups 4, 4a and 21!

Working Groups	Charter
POSIX.O	Open-system architecture
POSIX.1	Posix application interface (API, close to base functions of SVID)
POSIX.2	Shell and command utilities
POSIX.3	Testing and verification methods
POSIX.4	Real-time extensions to POSIX (4a Extensions for threads)
POSIX.5	Ada binding to POSIX. 1
POSIX.6	System security extensions
POSIX.7	System administration
POSIX.8	Transparent file access
POSIX.9	FORTRAN-77 binding to POSIX.1
POSIX.1O	Supercomputing profile
POSIX.11	Transaction processing
POSIX.12	Protocol-independent network access
POSIX.13	Application environment profiles
POSIX. 14	Multiprocessing application environment profiles
POSIX.15	Batch services
POSIX. 16	Language independent POSIX. 1
POSIX.17	Directory/name Services (IEEE 1224.2)
POSIX.18	Basic POSIX system profile
POSIX. 19	FORTRAN-90 binding to POSIX. 1
POSIX.20	Ada binding to POSIX.4
POSIX.21	Distributed real-time

POSIX - working group 4

Working Groups	Charter
POSIX.O	Open-system architecture
POSIX.1	Posix application interface (API, close to base functions of SVID)
...	
POSIX.4	Real-time extensions to POSIX (4a Extensions for threads)
...	
POSIX.21	Distributed real-time

POSIX.4 supports RT abilities of applications

- routines for RT functionalities
- definition of timing constraints

and offers following classes of functions

- synchronisation with semaphores
- priorities for (heavyweight) processes
- common memory for several processes
- message passing between processes
- signaling of asynchronous events
- granting of non displacable memory to processes
- synchronous and asynchronous I/O
- coherent files

POSIX - Scalable (RT)OS, AEPs (Application Environment Profiles)

AEP 1 (Minimal real-time system):

- Typically for embedded systems.
- Only one address space, i.e. one (POSIX) process,
- parallel processing by threads
- minimum HW: CPU, RAM but no MMU
- functionality comparable to (OS) kernel, base for AEP 2..4
- N.B.: developers for embedded systems rare, well paid, ...

AEP 2 (Real-time controller system):

- AEP 1
  - + file interface
  - + serial i/o
  - + exceptions (POSIX signals)
- minimum HW: AEP 1 + at least one serial interface (e.g. RS 232)
- not yet: mass storage (e.g. disk), file system can be realised in RAM

POSIX - Scalable (RT)OS, AEPs (Application Environment Profiles)

AEP 3 (Dedicated real-time system):

- Several (POSIX) processes/address spaces
- Common interface for device drivers
- (Non hierarchical) file system
- RAM resident processes (memory locking) if MMU present
- HW: one or more CPU(s), with or without MMU

AEP 4 (Multi-purpose real-time system):

- support of all RT POSIX extensions
- HW: CPU(s) with MMU, mass storage, network support, terminal

/Zöbel95/

4.1 Characteristics of RTOS (Realtime Operating Systems)

4.2 The Kernel

4.3 Synchronisation and priority inheritance

4.4 Driver

/Zöbel95/

4.1 Characteristics of RTOS (Realtime Operating Systems)

4.2 The Kernel

- 4.2.1 Definition
- 4.2.2 Synchronisation
- 4.2.3 Cross development
- 4.2.4 Latency Time

4.3 Synchronisation and priority inheritance

4.4 Driver

The kernel - inconsistent definitions

Minimum kernel tasks:

- administration of data structures for process description (PCB)
- processing of system calls
- scheduling

by consistent access on state describing data - so first **definition**:

> Kernel is that part of the OS where interrupts are not allowed.

On the other side may a non preemptive kernel delay reactions on external events or system calls unpredictable - hard RT conditions can not be guaranteed. So a RT kernel must be interruptable - contradiction!

Second **definition**:

> Kernel is that part of the OS, which enables (quasi) parallel execution of several processes including functions for synchronisation of and communication between those processes (microkernel).

Signals demanded by POSIX

Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

/Tanenbaum02/

/Zöbel95/

4.1 Characteristics of RTOS (Realtime Operating Systems)

4.2 The Kernel

- 4.2.1 Definition
- 4.2.2 Synchronisation (low level)
- 4.2.3 Cross development
- 4.2.4 Latency Time

4.3 Synchronisation and priority inheritance

4.4 Driver

Kernel and RT capabilities

To fulfill hard RT conditions every process - even an OS one - must be interruptable.
To get this one

1. defines **preemption points** within the kernel with consistent states or
2. makes the complete kernel interruptable except some **critical regions**.

where

1. is slower (e.g. latency time2-5ms vs. 75-250µs, MC68030 33MHz) and
2. is more complicated.

Guarantee that only one process is within a critical region

E.g. by **busy waiting** (also called **spin lock**) using the `test-and-set` instruction:

```
ts (int x, int y) /* in fact both boolean */
{/* copy y on x and set y to FALSE */
}
```

Guarantee that only one process is within a critical region

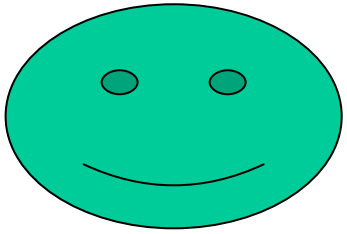
```
ts (int x, int y) /* in fact both boolean */
{/* copy y on x and set y to FALSE */
}
```

```
/* y initialised to TRUE */

lock (int y)
{ int xp;
  /* private variable of corresponding process */
   do
      ts (xp, y);
   while (!xp);
}

unlock (int y)
{ y = TRUE; }
```

Concurrent processes may cause deadlocks

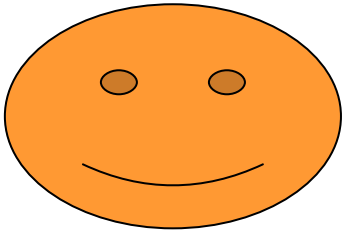
Deadlocks as in FHK:

Student A



lends book a
reads
needs book b
waits

Student B

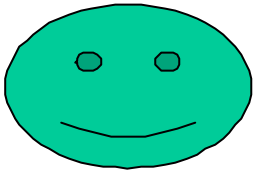


lends book b
reads
needs book a
waits

Sie erinnern Sich (Deadlock-Beispiel)?

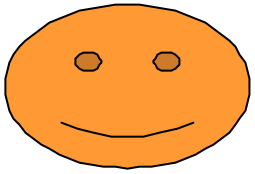
Deadlock – Beispiel FHK

Student A



entleiht Buch a
liest
benötigt Buch b
wartet

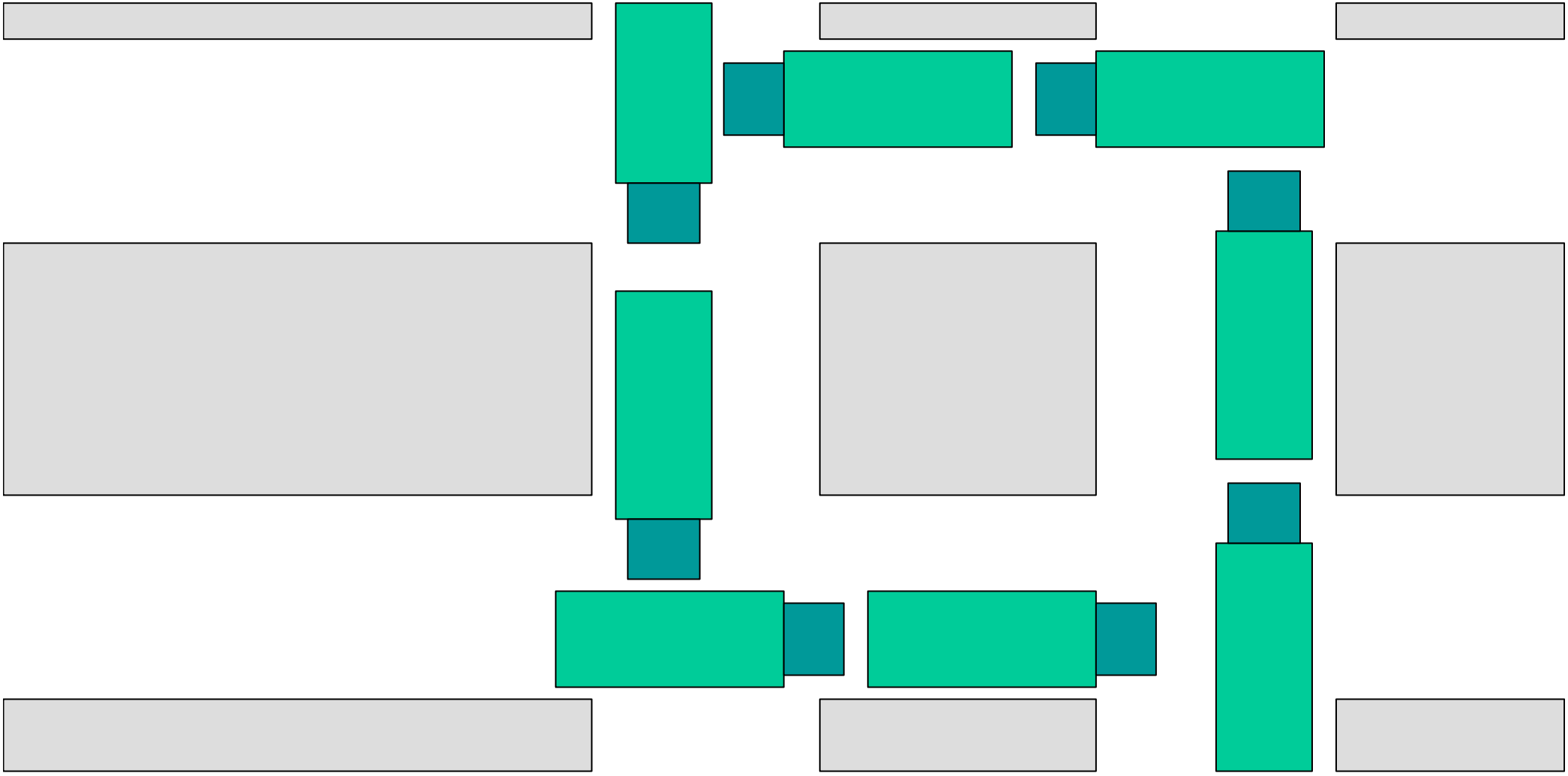
Student B



entleiht Buch b
liest
benötigt Buch a
wartet

Concurrent processes may cause deadlocks

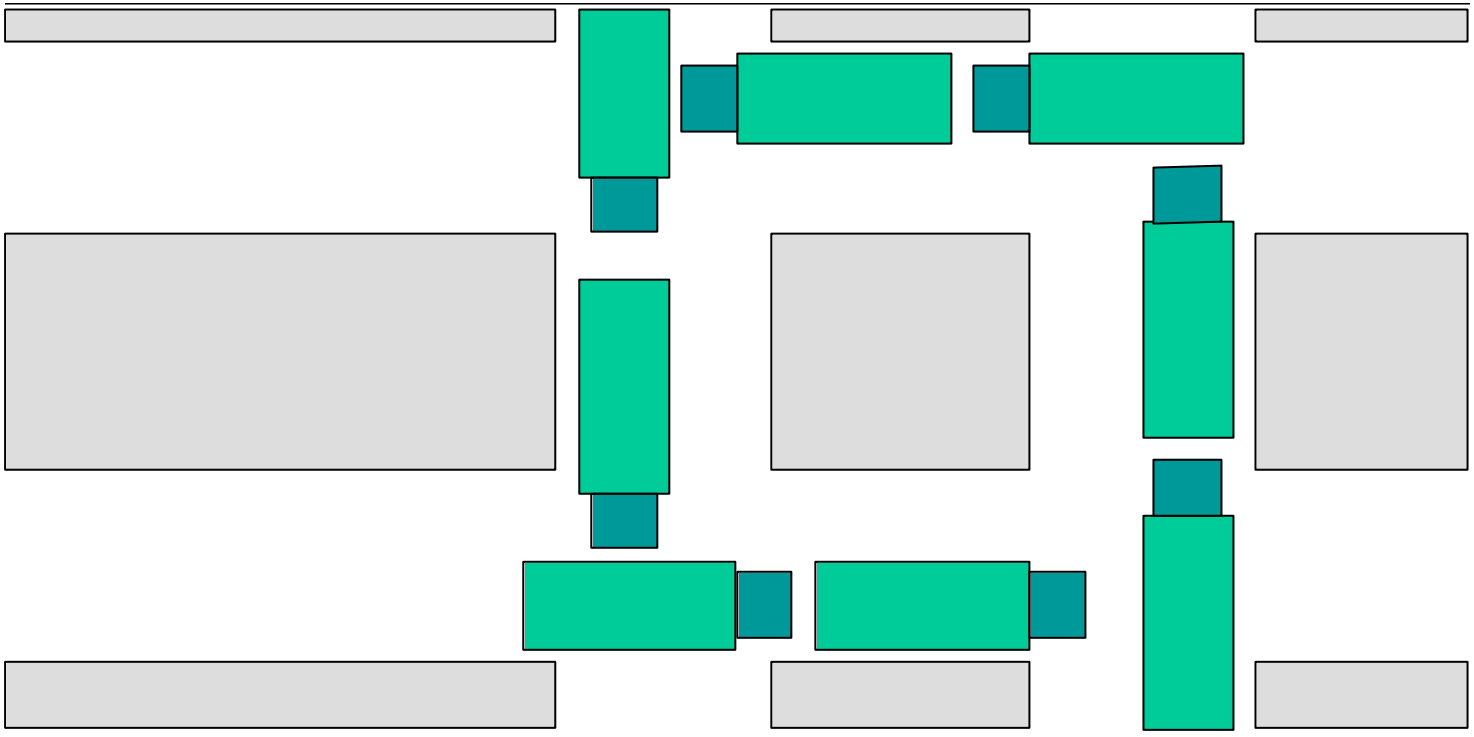
Deadlocks as in the street:



Sie erinnern Sich (Deadlock-Beispiel)?

X

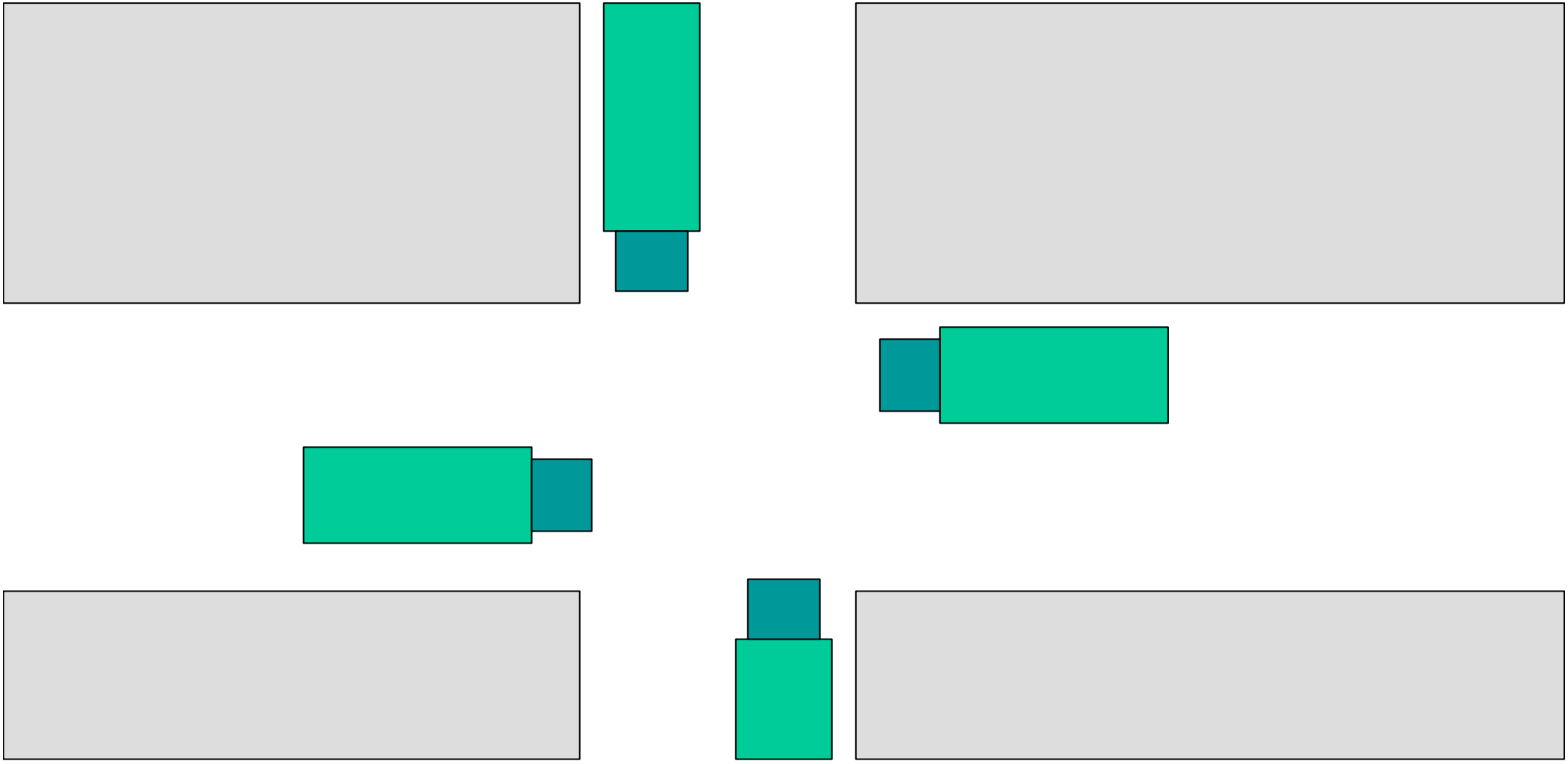
Deadlocks (Systemverklemmungen)



BSS1 5.31

Concurrent processes may cause deadlocks

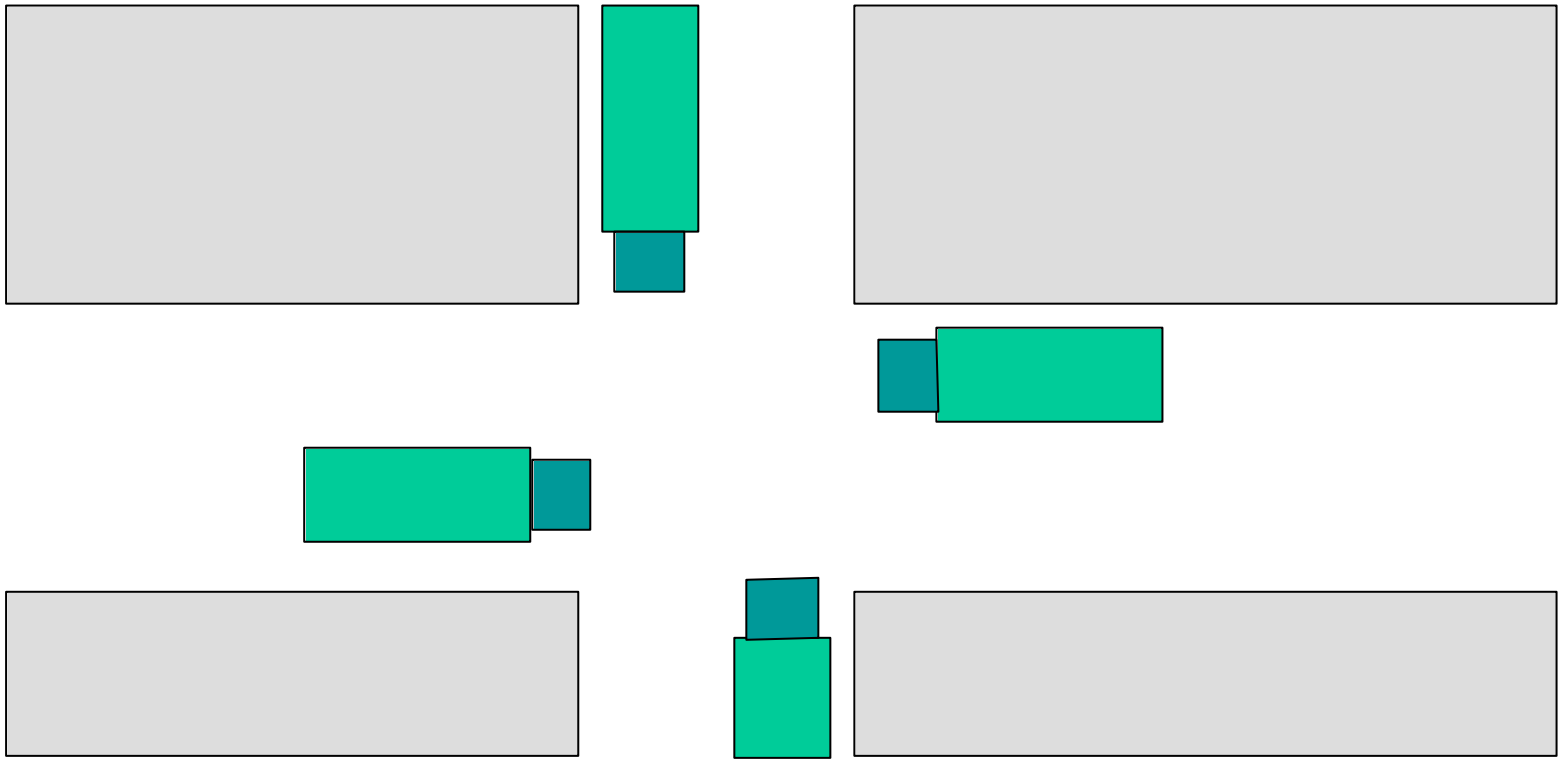
Deadlocks as in the street again:



Sie erinnern Sich (Deadlock-Beispiel)?

X

Deadlocks – noch‘n Beispiel



Concurrent processes may cause deadlocks

E.g deadlock as follows:

- process a gets access to critical region A by calling of lock()
- process b displaces process a
- process b tries to enter critical region A

so one needs additional functionalities disable `di()` and enable interrupt `ei()` for processes on the same processor in addition to `lock()` and `unlock()`.

```
enter (int y)
{ di();
  lock(y);
}
leave (int y)
{ unlock(y);
  ei();
}
```

/Zöbel95/

4.1 Characteristics of RTOS (Realtime Operating Systems)

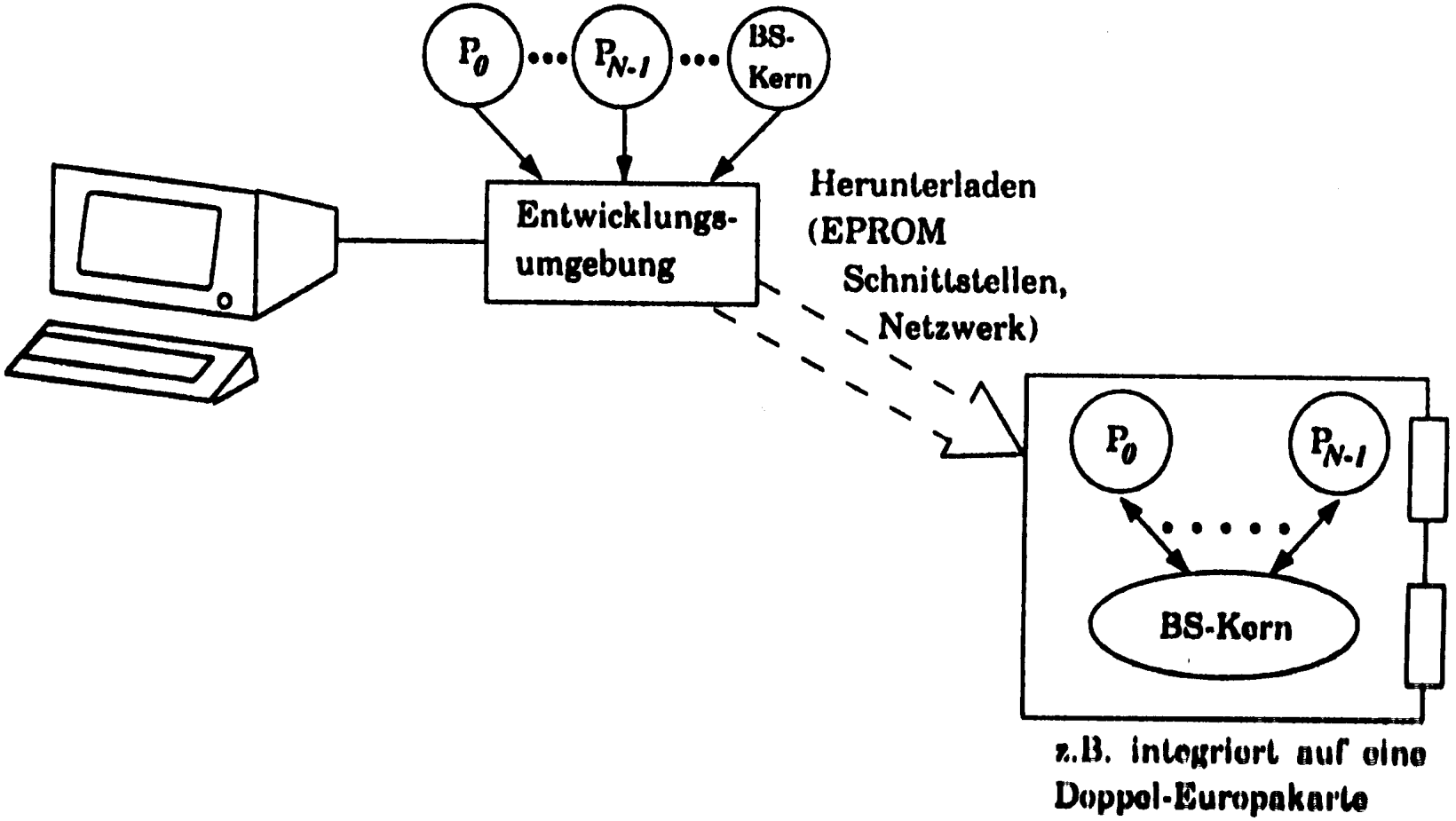
4.2 The Kernel

- 4.2.1 Definition
- 4.2.2 Synchronisation
- 4.2.3 Cross development
- 4.2.4 Latency Time

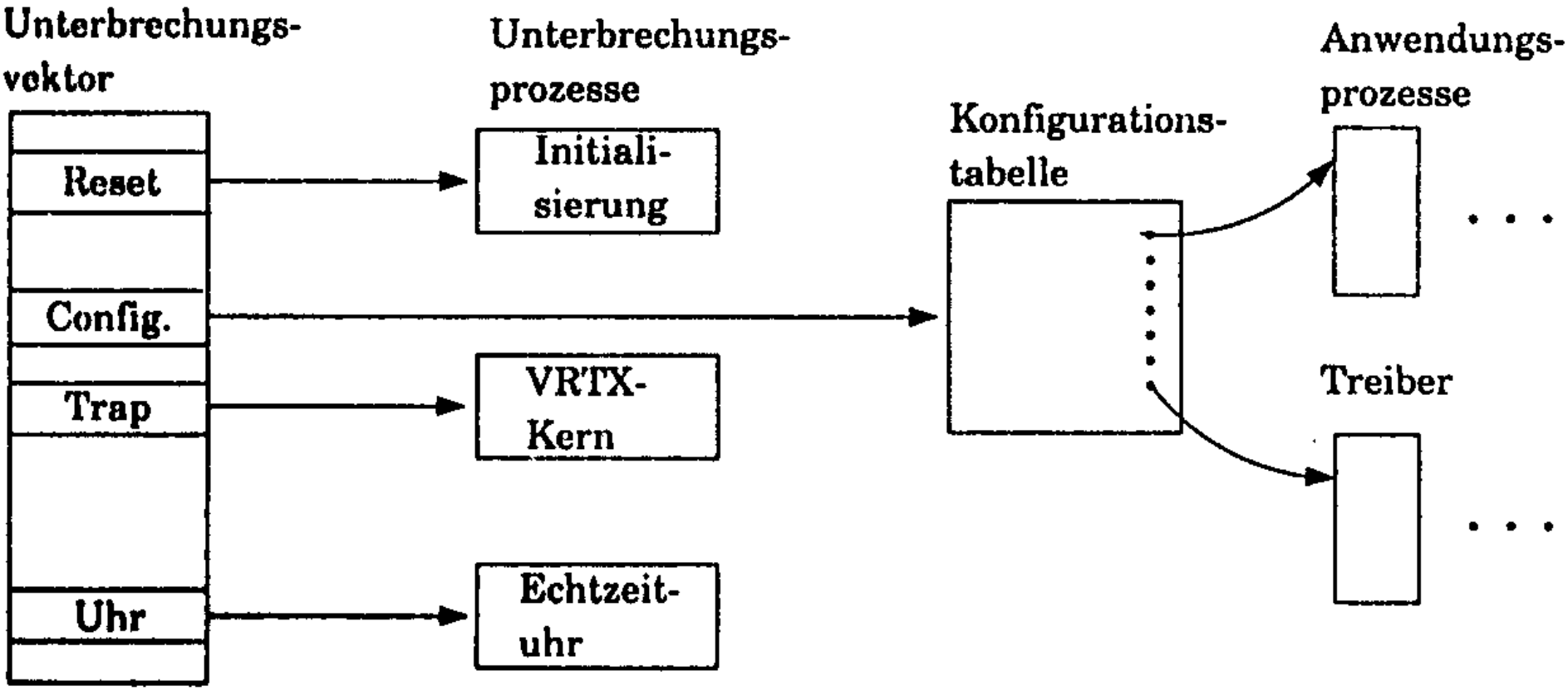
4.3 Synchronisation and priority inheritance

4.4 Driver

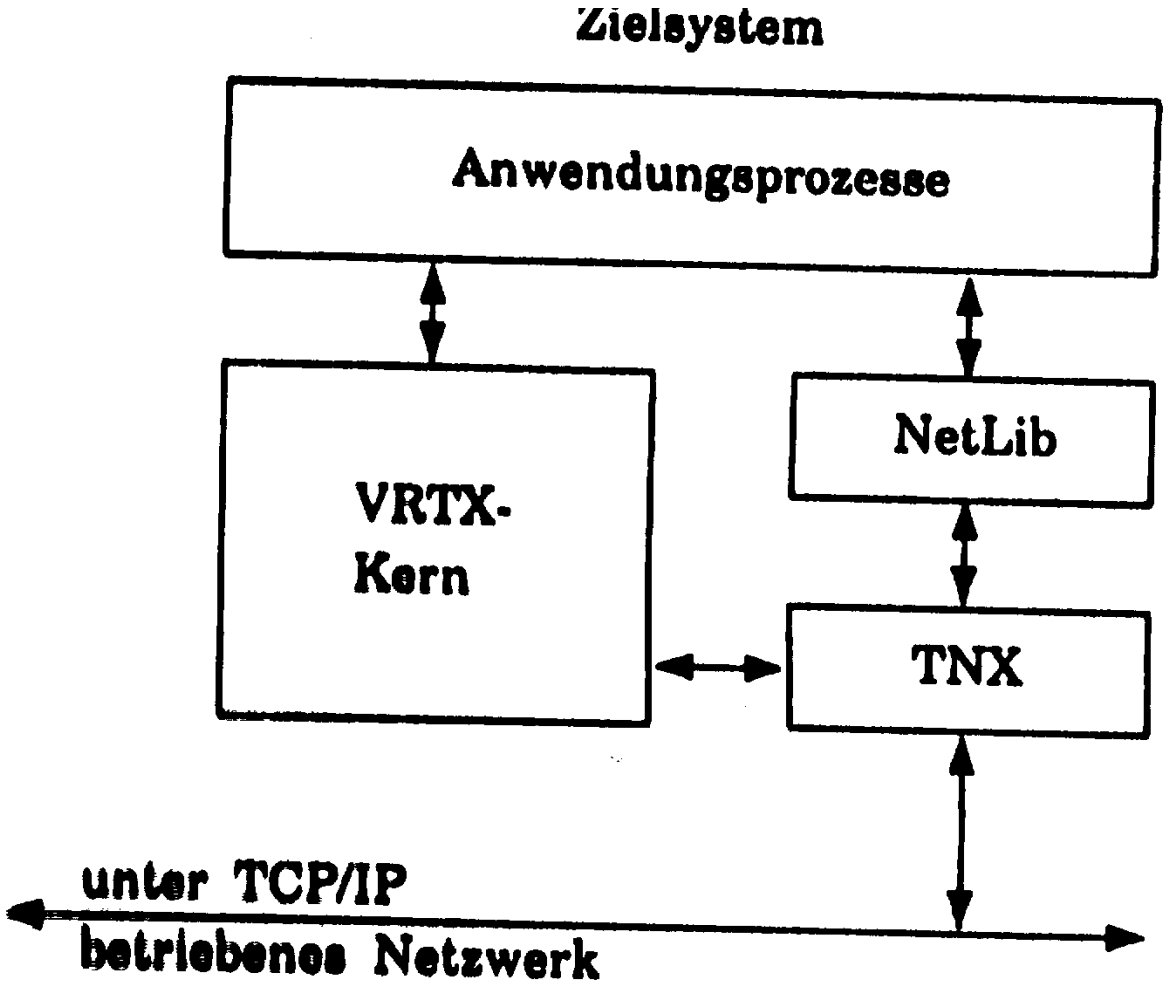
Cross Development - Scheme



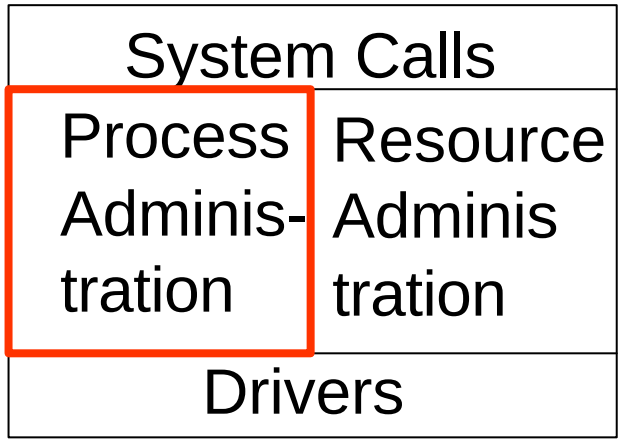
One Cross Development OS - VRTX Structure



Cross Development - Connection Target to Network



Make the kernel small!



One tries - not only in RTOS - to keep the kernel as small as possible:

- microkernel
- nanokernel
- (pikokernel)

"Small is beautiful!", "Strip it down!", ...

milli, mikro, nano, ...

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

/Tanenbaum02/

Small is beautiful

X

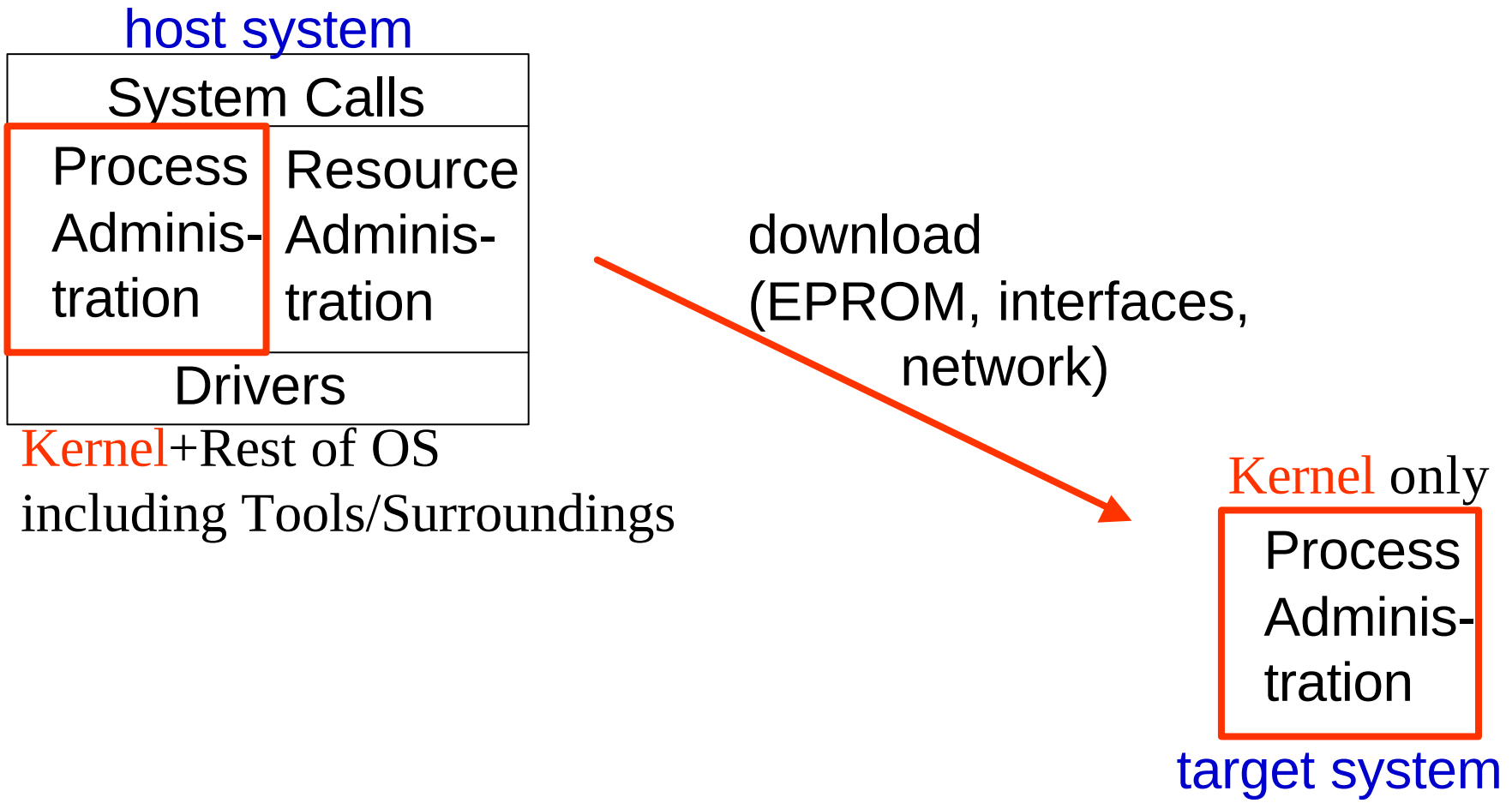
µ- und n-Kernel: Reduzierung auf Kernaufgaben

Beim Bau skalierbarer OS wird versucht, einen ´Kern´ des OS zu definieren, dem je nach Bedarf zusätzliche ´OS-Applikationen´ angehängt werden. Reduzierung beispielsweise auf:

- Schnittstelle SW-HW
- ~~Schnittstelle externe HW~~
- Betriebsmittelverwaltung
- Sicherheit?
- Fehlerbehandlung?
- ~~Multi-Task und –User, Betriebsarten~~
- (Prozesse, Synch und Kommunikation)
- ~~User-Interface~~
- ~~Admin einschl. Kostenabrechnung~~

BSS1 1.16

Cross development with small kernel



/Zöbel95/

4.1 Characteristics of RTOS (Realtime Operating Systems)

4.2 The Kernel

- 4.2.1 Definition
- 4.2.2 Synchronisation
- 4.2.3 Cross development
- 4.2.4 Latency Time

4.3 Synchronisation and priority inheritance

4.4 Driver

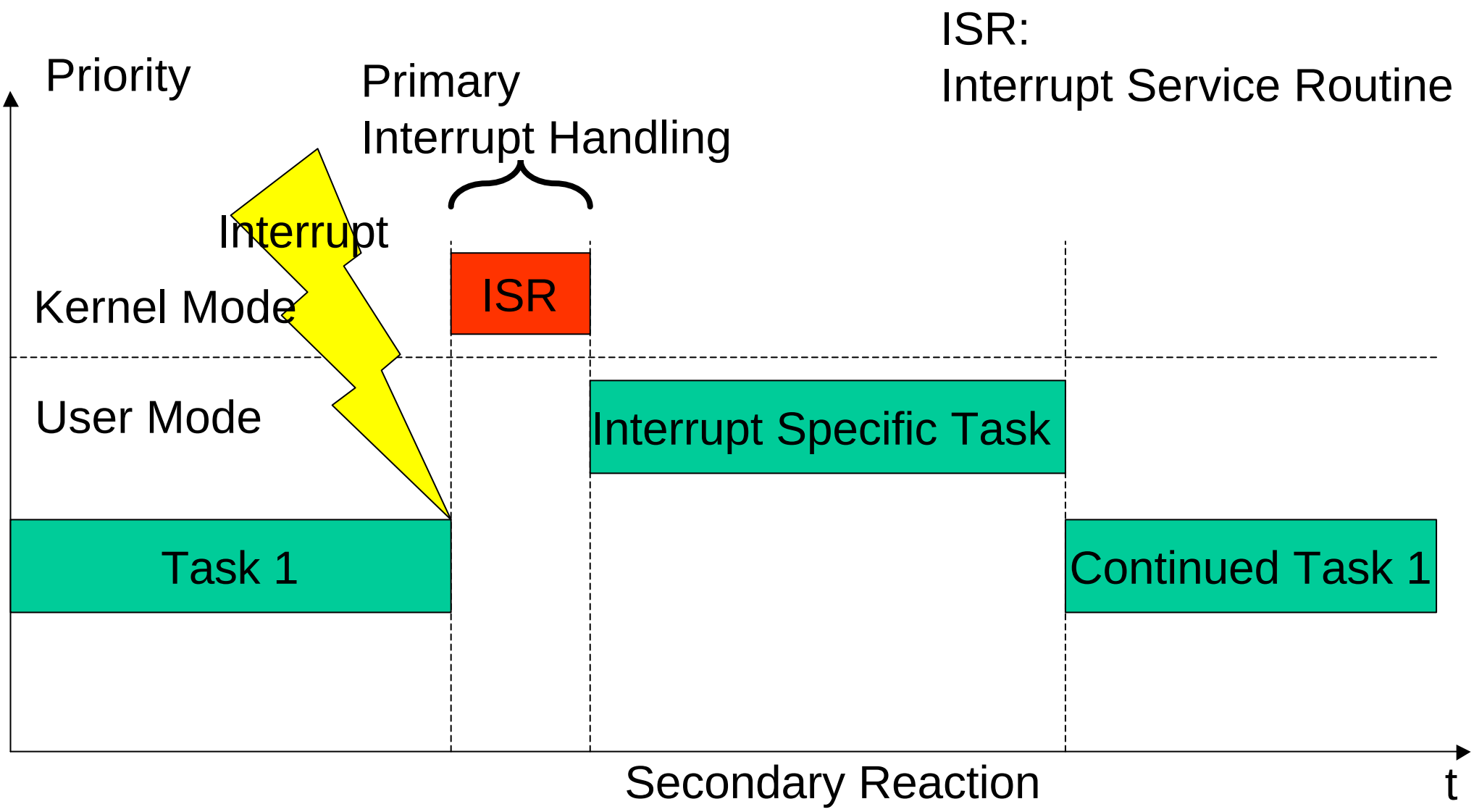
Interrupt task answer time and interrupt latency time

Interrupt latency time is the time between an external event and the execution of the first statement of the corresponding interrupt service routine (ISR).
This is **not** the time of a context switch - even if every manufacturer says so.

A RT analysis has to check, whether the application reacts on time at an external event - the **interrupt task answer time**.
It is dependent of

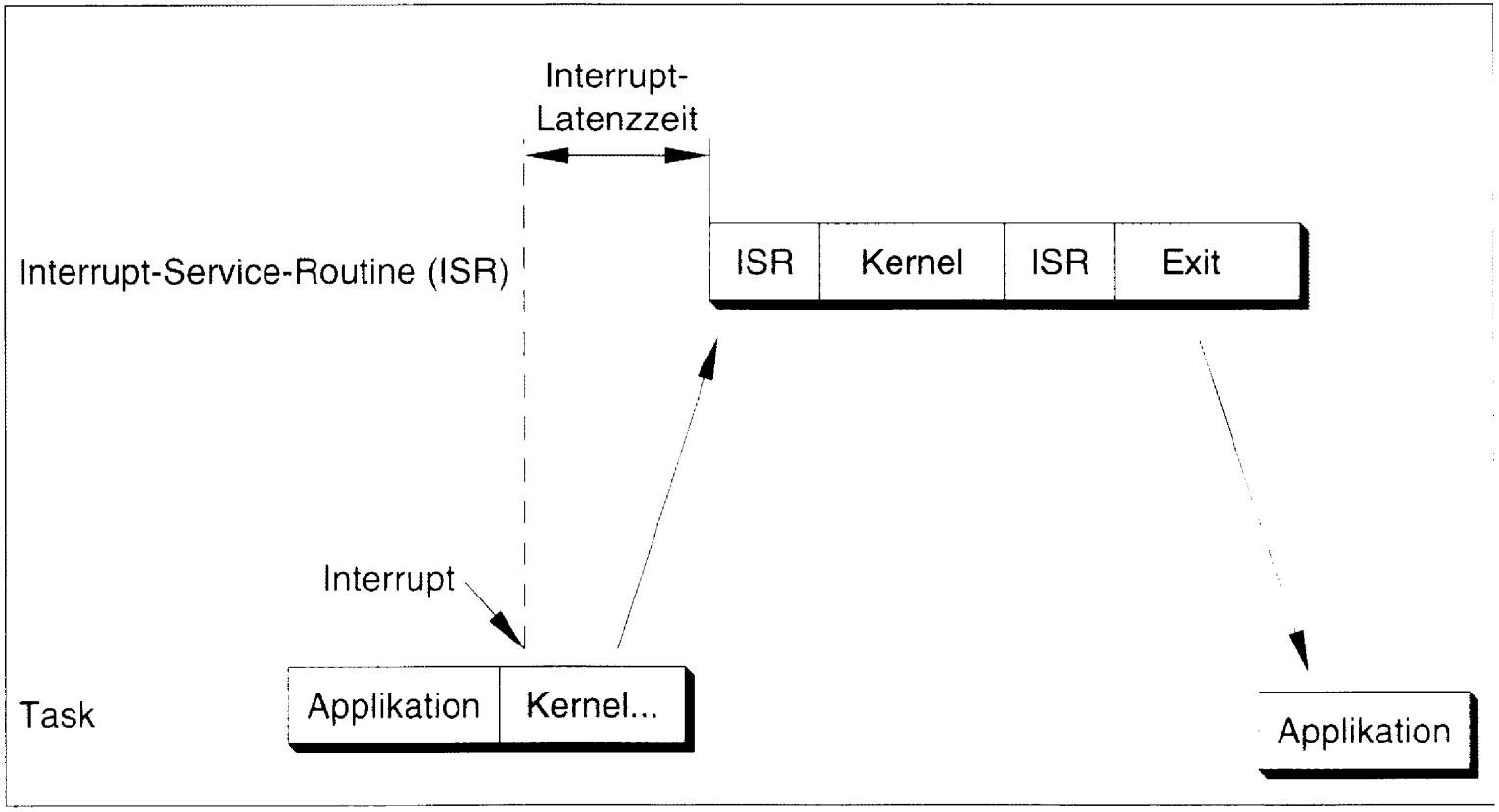
- the switching time (the time of a context switch)
- the interrupt latency time
- whether the kernel is preemptive or not
- whether only heavyweight processes are possible or threads too.

Reactions on external events



Interrupt latency time

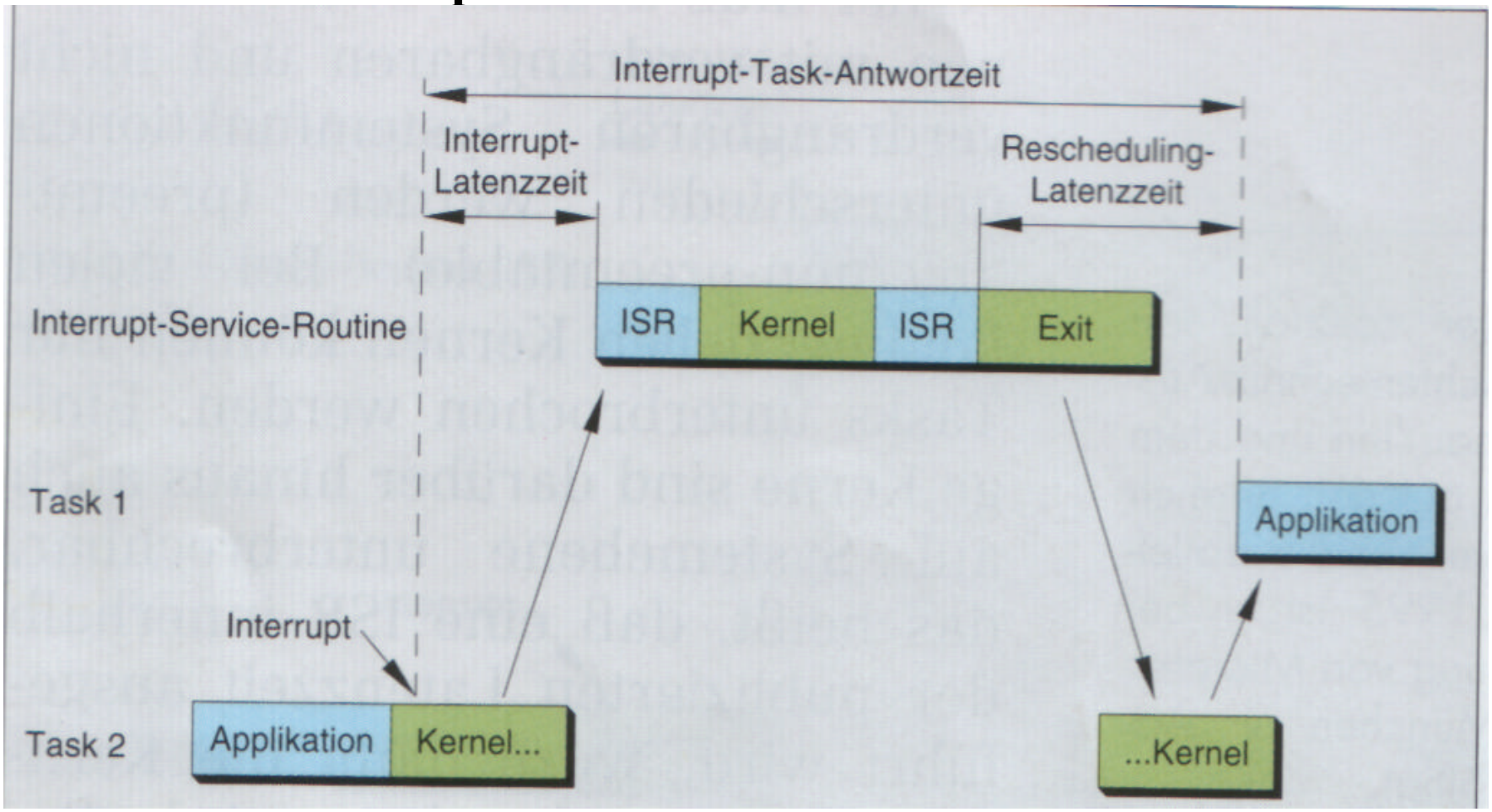
Interrupt latency time is the time between an external event and the execution of the first statement of the corresponding interrupt service routine (ISR).



Interrupt task answer time

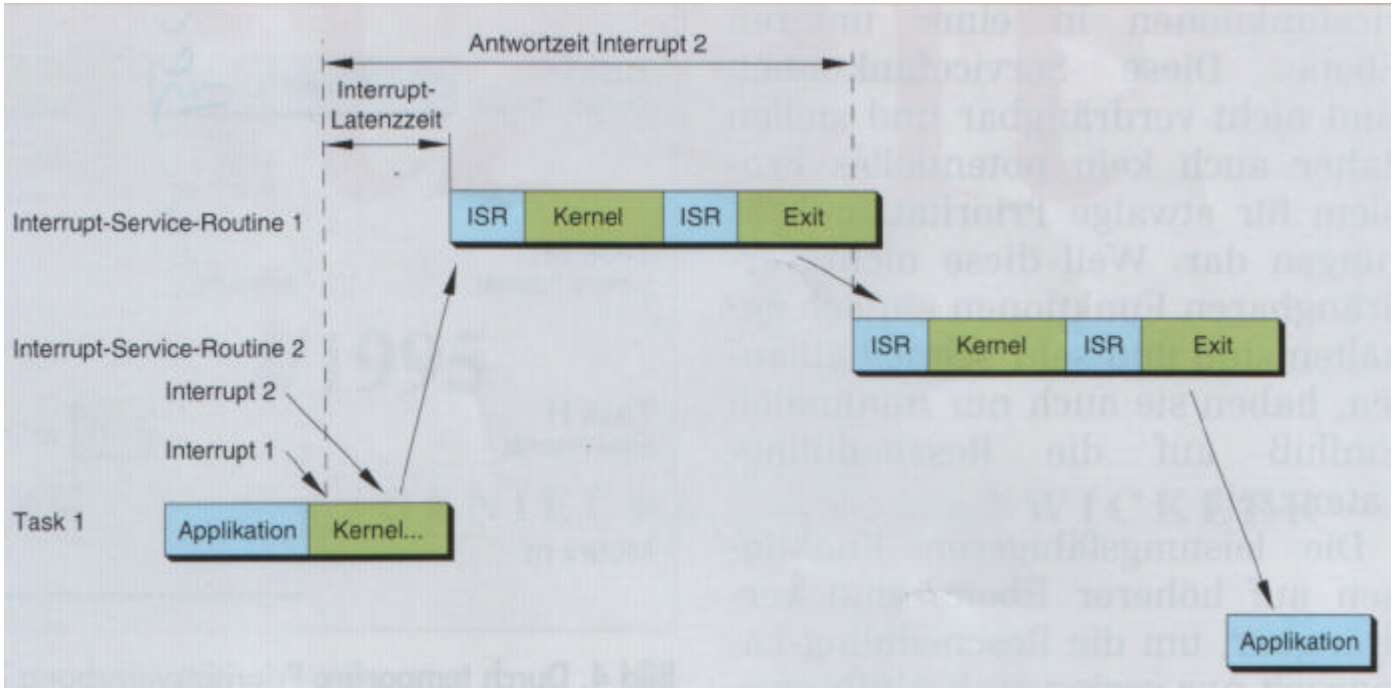
Interrupt task answer time

A RT analysis has to check, whether the application reacts on time on an external event - the **interrupt task answer time**.



Here worst case with a non preemptive kernel.

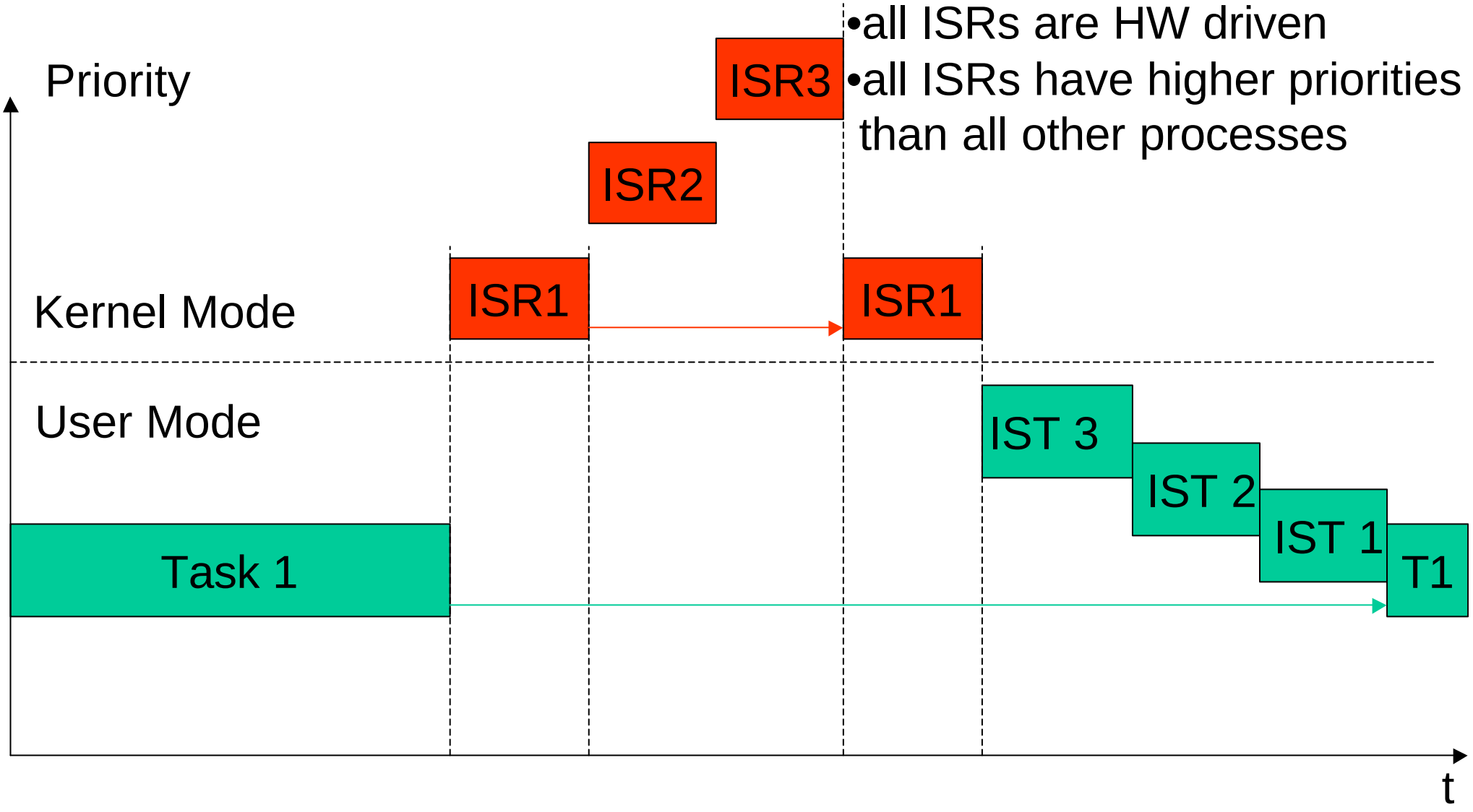
Preemptive kernel, second interrupt



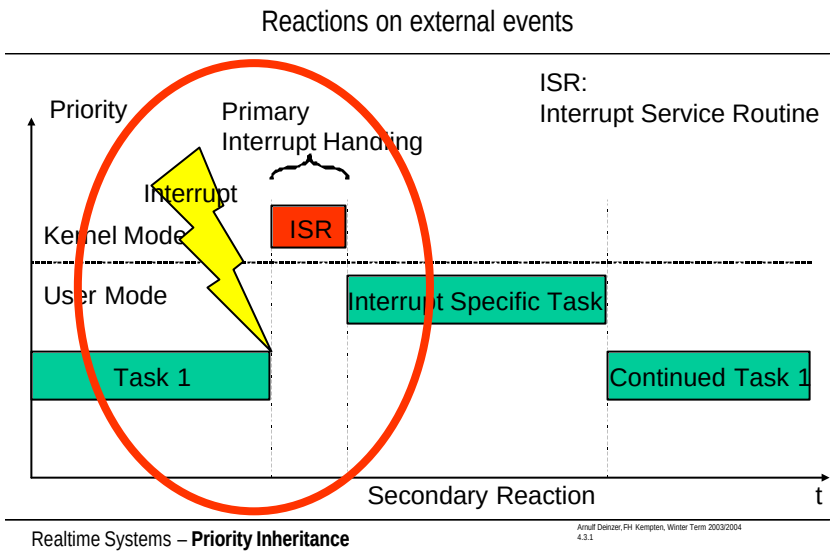
Here is the interrupt latency time only one out of three components of the interrupt task answer time; additional

- time of the first ISR itself and
- possible (more) system calls

Hierarchy of ISRs and Secondary Reactions



Nothing happens in 0 time



Interrupt latency (time)

ISR

Interrupt Specific Task

Task 1

PDLT (Process Dispatch Latency Time)

Nothing happens in 0 time

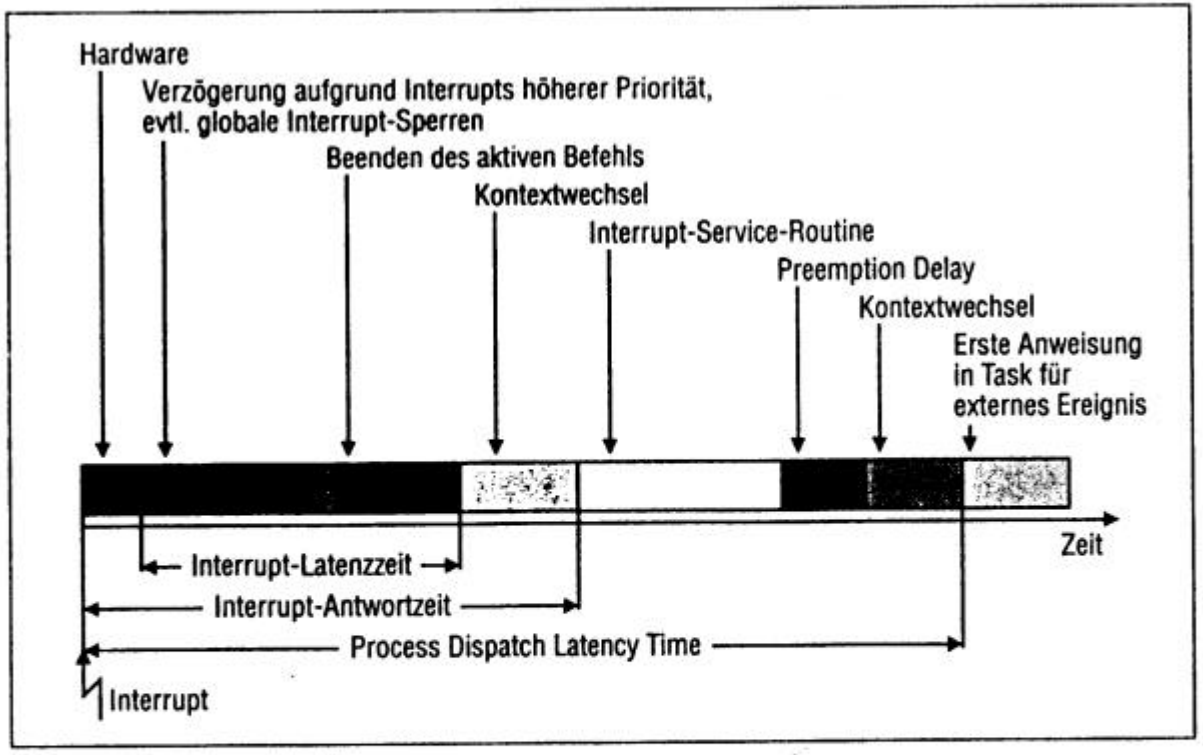
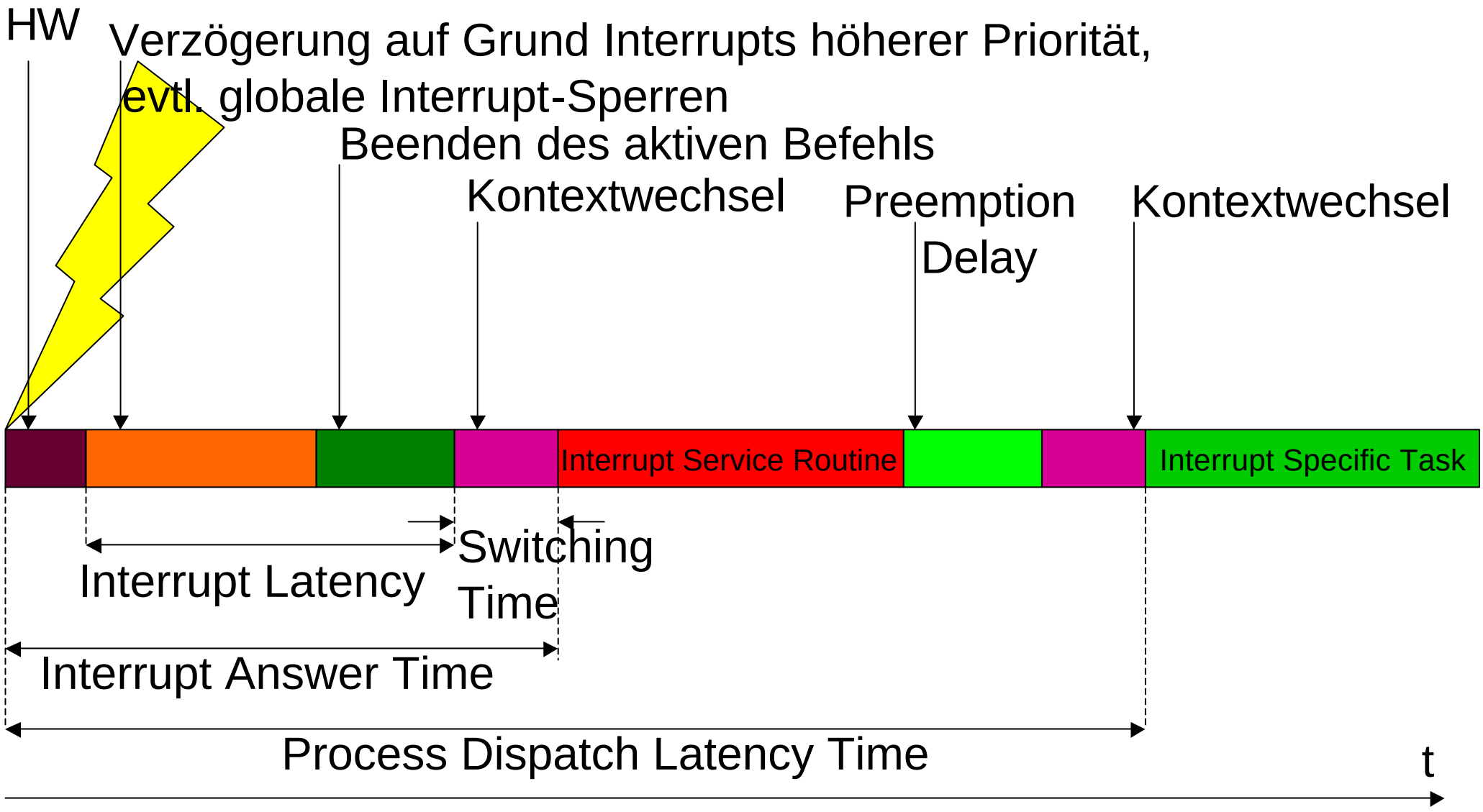


Bild 1. Vom Interrupt bis zur Systemreaktion: Die „Process Dispatch Latency Time" setzt sich aus mehreren Wartezeiten zusammen. Das „Preemption Delay" tritt auf, wenn die zu unterbrechende Task gerade kritischen Systemcode ausführt und diesen erst beenden muß, bevor zu einer anderen Task gewechselt werden kann.

Nothing happens in 0 time

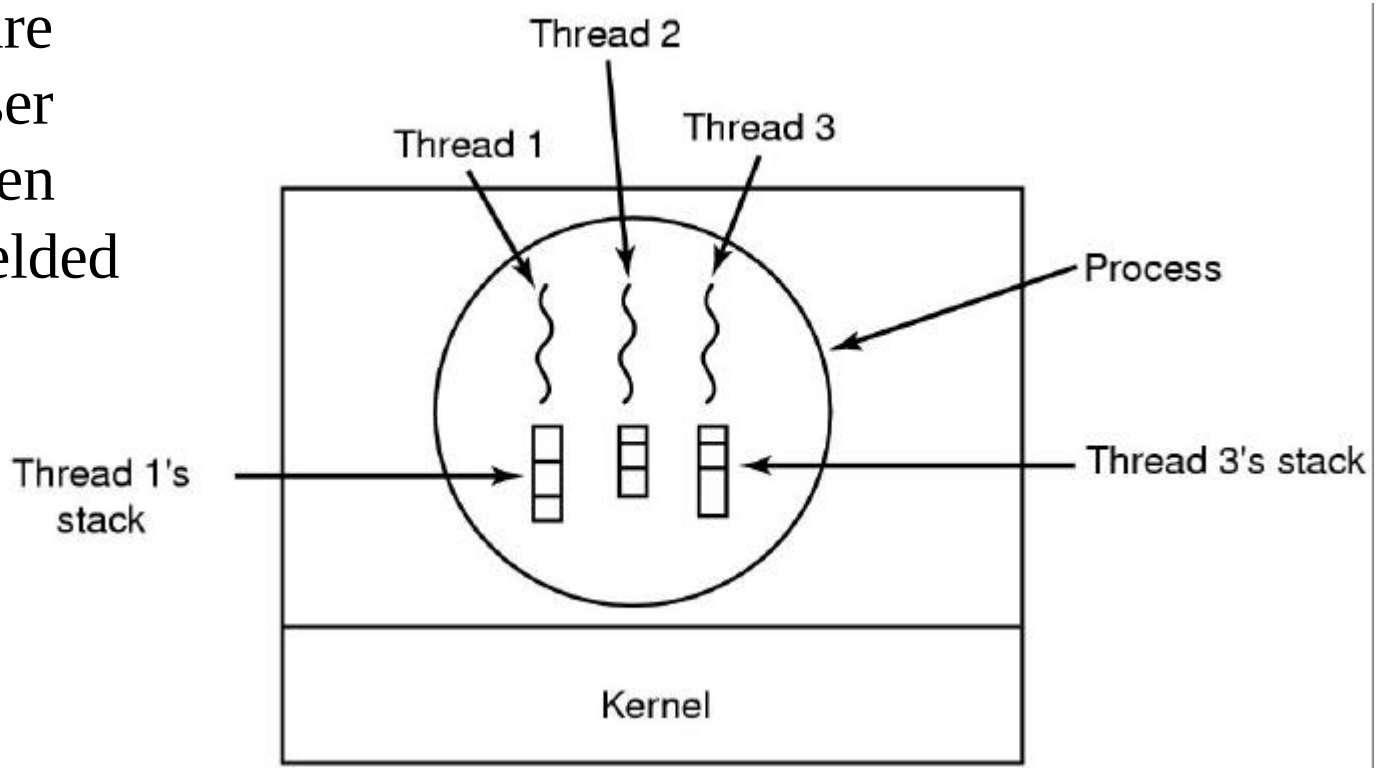


Latency time enhancements by use of threads

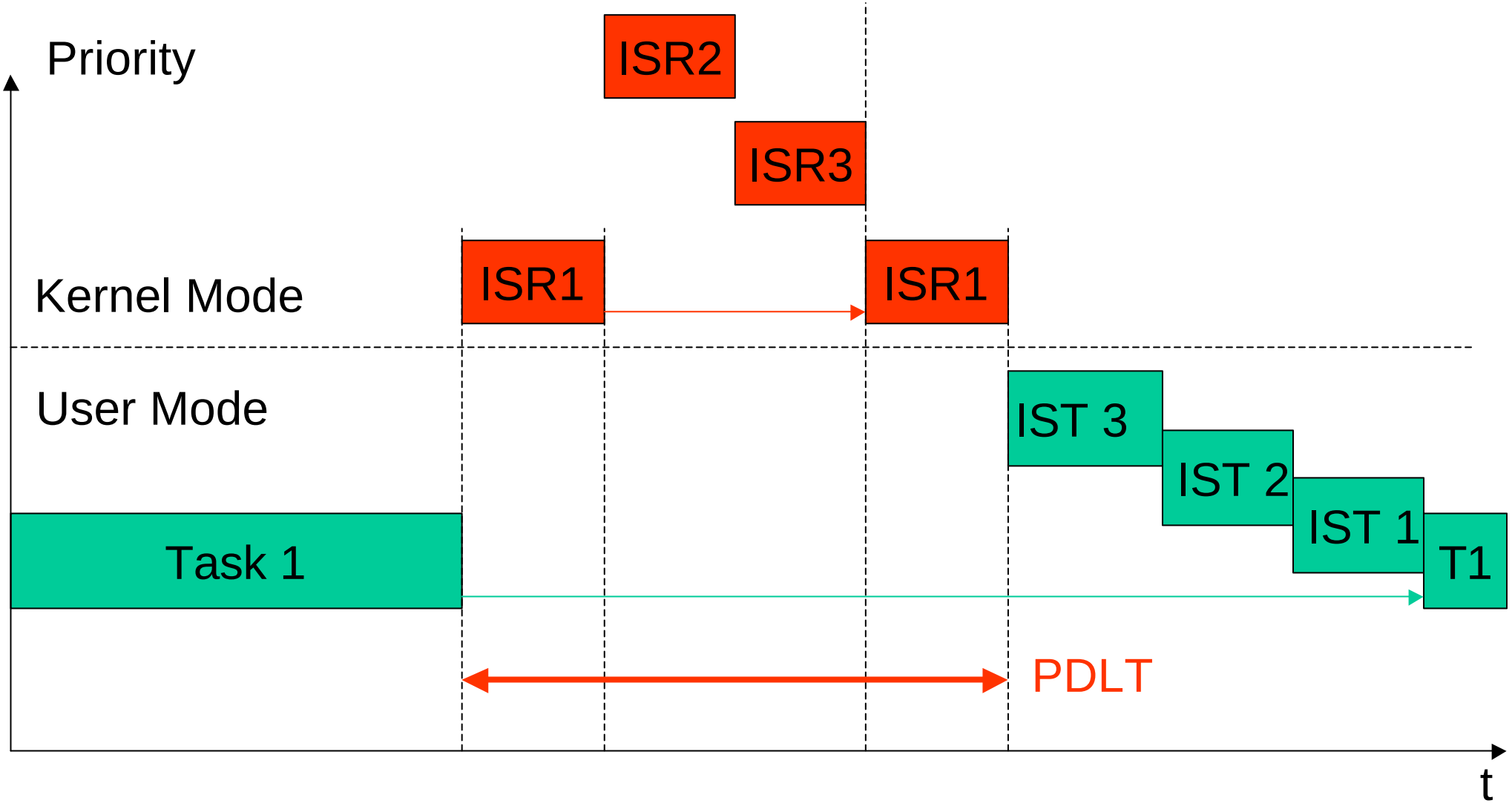
Due to their simplicity and flexibility do threads better fit to RT models than (heavyweight) processes:

- their task switch times are much shorter and
- their variance (i.e. the difference between slowest and fastest) is smaller too

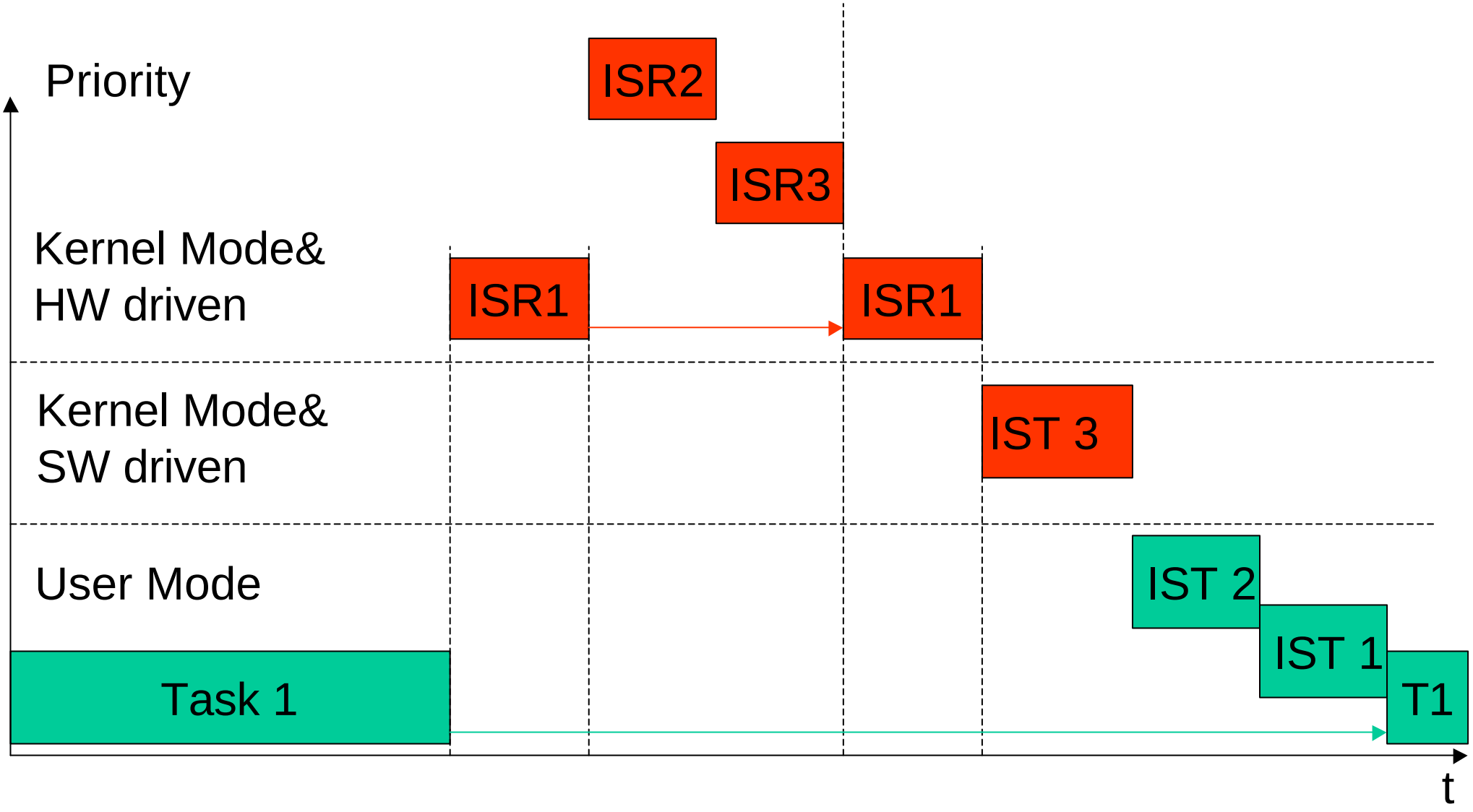
Heavyweight processes are needed if behaviour of user and applications inbetween and to OS have to be shielded due to security reasons.



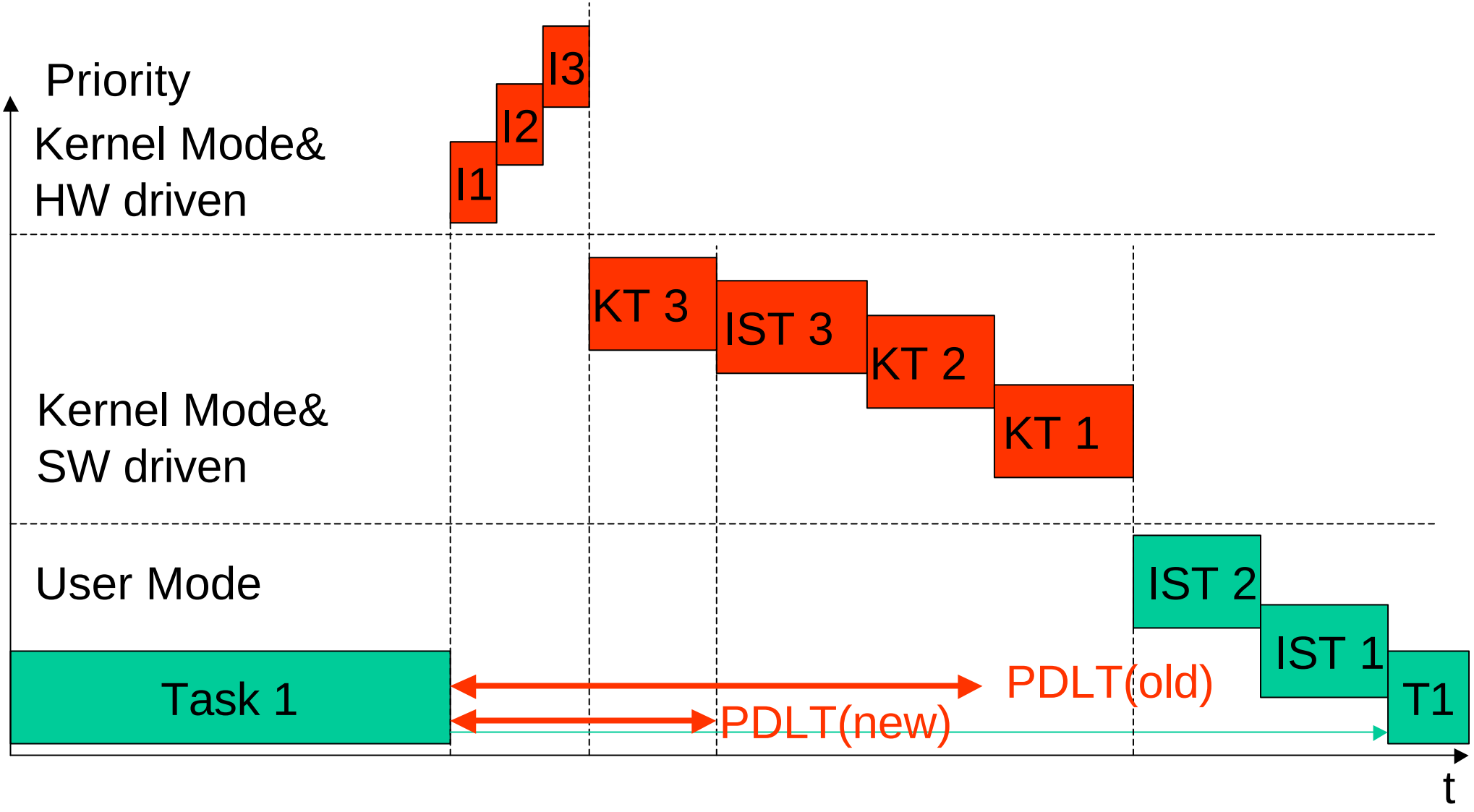
Change of IR Priority does **not** change PDLT...



...even when secondary routines become SVCs...



...but kernel threads may help



RTOS: user has influence on latency times

One of the main features of RTOS (vs. "normal" OS) is, that the user/programmer has influence on the scheduling of processes, e.g. by giving them priorities.

Typical process classes can be

IT: processes started by asynchronous interrupts

RT: processes with hard RT constraints

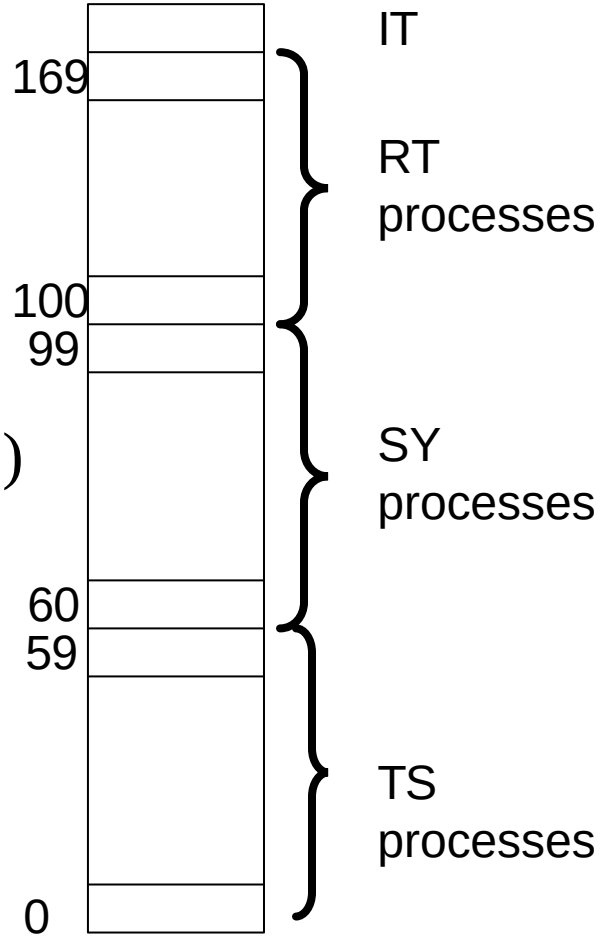
SY: processes of the OS

BE: processes with soft or no RT constraints (best e.)

TS: processes using gaps (time sharing)

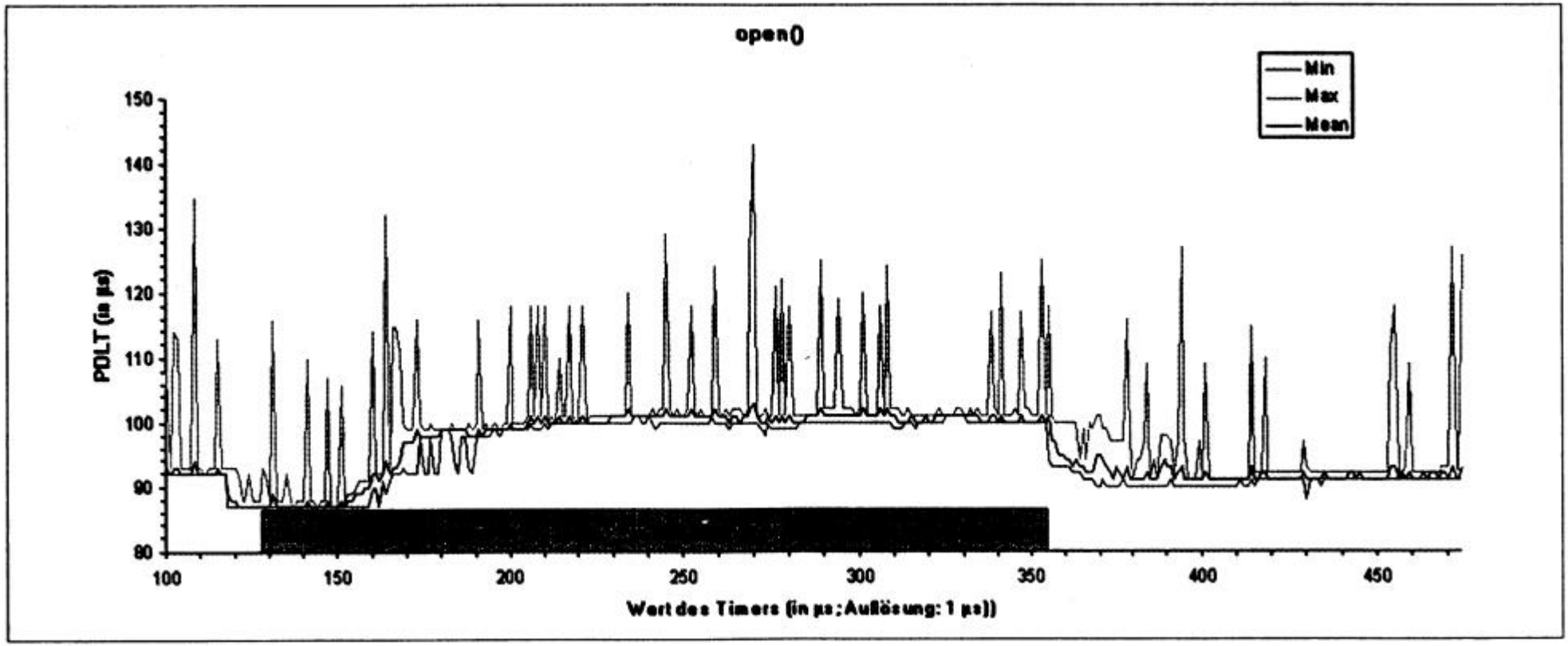
ID: idle

The user shall at least have the chance to give the RT processes priorities.



Process Classes in OS Solaris

RTOS may be predictable - better measure!



/Zöbel95/

4.1 Characteristics of RTOS (Realtime Operating Systems)

4.2 The Kernel

4.3 Synchronisation and priority inheritance

4.4 Driver

Synchronisation with semaphores

Synchronisation methods up to now:
are
- low level (based on `ts()`-statement)
- time consuming (busy wait)
- dangerous (possible deadlocks)

but
process synchronisation is needed often in RT scenarios - so a more comfortable mechanism is needed: **semaphores**.

Guarantee that only one process is within a critical region

```
ts (int x, int y) /* in fact both boolean */
{/* copy y on x and set y to FALSE */
}
```

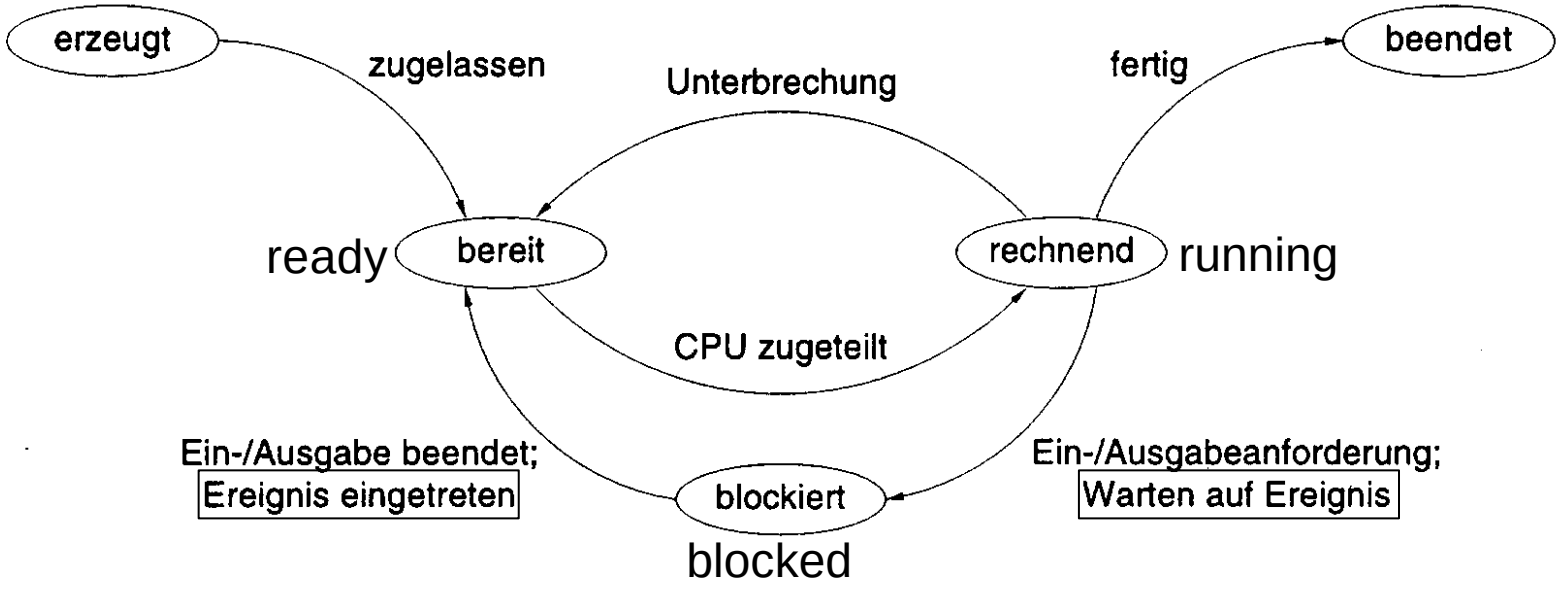
```
/* y initialised to TRUE */
lock (int y)
{ int xp;
  /* private variable of corresponding process */
  do
     ts (xp, y);
  while (!xp);
}
unlock (int y)
{ y = TRUE; }
```


A semaphore s (E. W. Dijkstra 1968) is an abstract data type incremented (only) by operation V (dutch "verlaat", leave) and decremented by operation P (dutch "passeer", enter).

Semaphores: P and V operation

```
P (semaphore s) {
   s=s-1;
   if (s<0) wait_for(s);
}

V (semaphore s) {
   s=s+1;
   if (s<=0) awake(s);
}
```

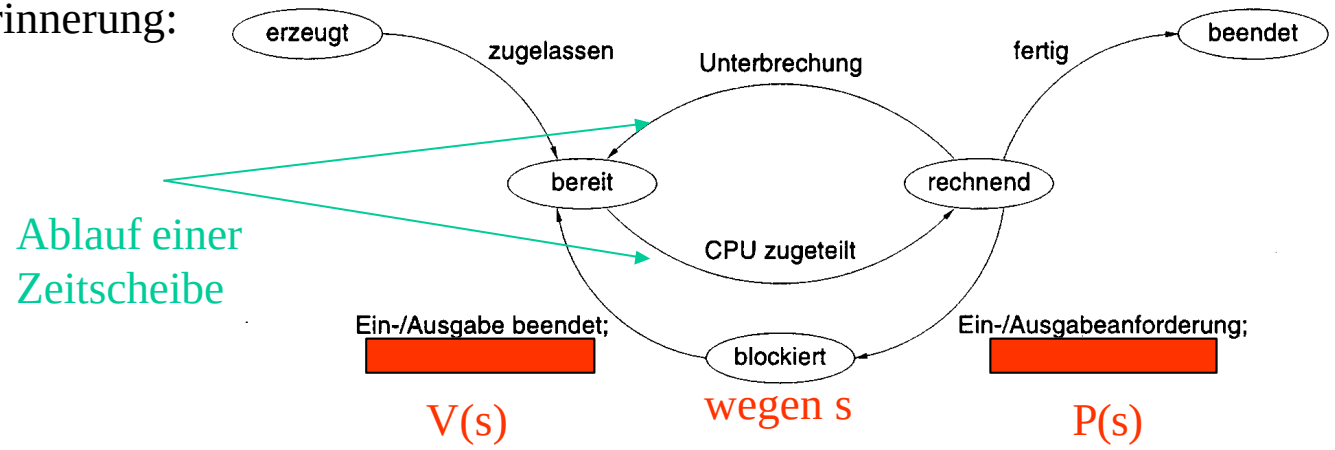
Semaphores: P and V operation and process states



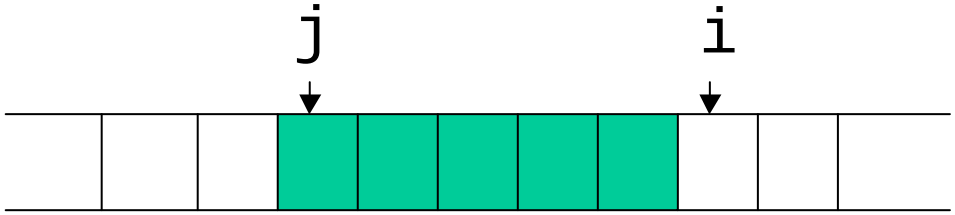
Semaphores: P and V operation and process states

Semaphore und Zustandsübergänge

Erinnerung:

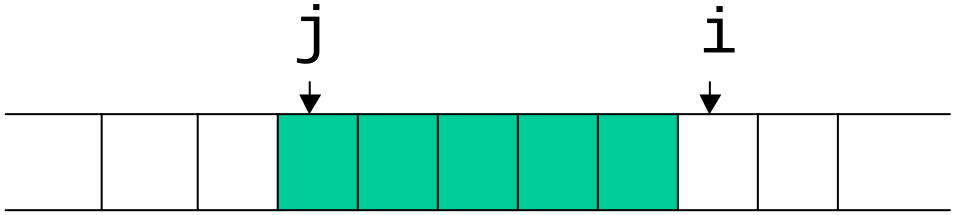


Semaphores: example producer/consumer problem




```
pr() /* producer *
{ while (true)
      { /* produce data/item x */
      P(f);
      /*f>=0 && b<N */
      buffer[i]=x;
      i=(i+1) % N;
      V(b);
      /* f>=0 && b<=N */
   }
}
```

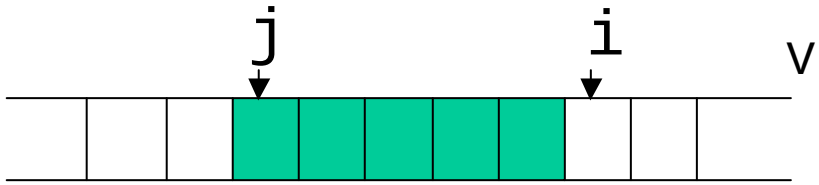
Semaphores: example producer/consumer problem




```
c()  /* consumer *
{ while (true)
      P(b);
      /*b<=0 && f<N */
      y=buffer[j];
      j=(j+1) % N;
      V(f);
      /* b>=0 && f<=N */
      /* consume data/item y */
   }
}
```

Semaphores: example producer/consumer problem

```
P (semaphore s) {
    s=s-1;
    if (s<0) wait_for(s);
}
```

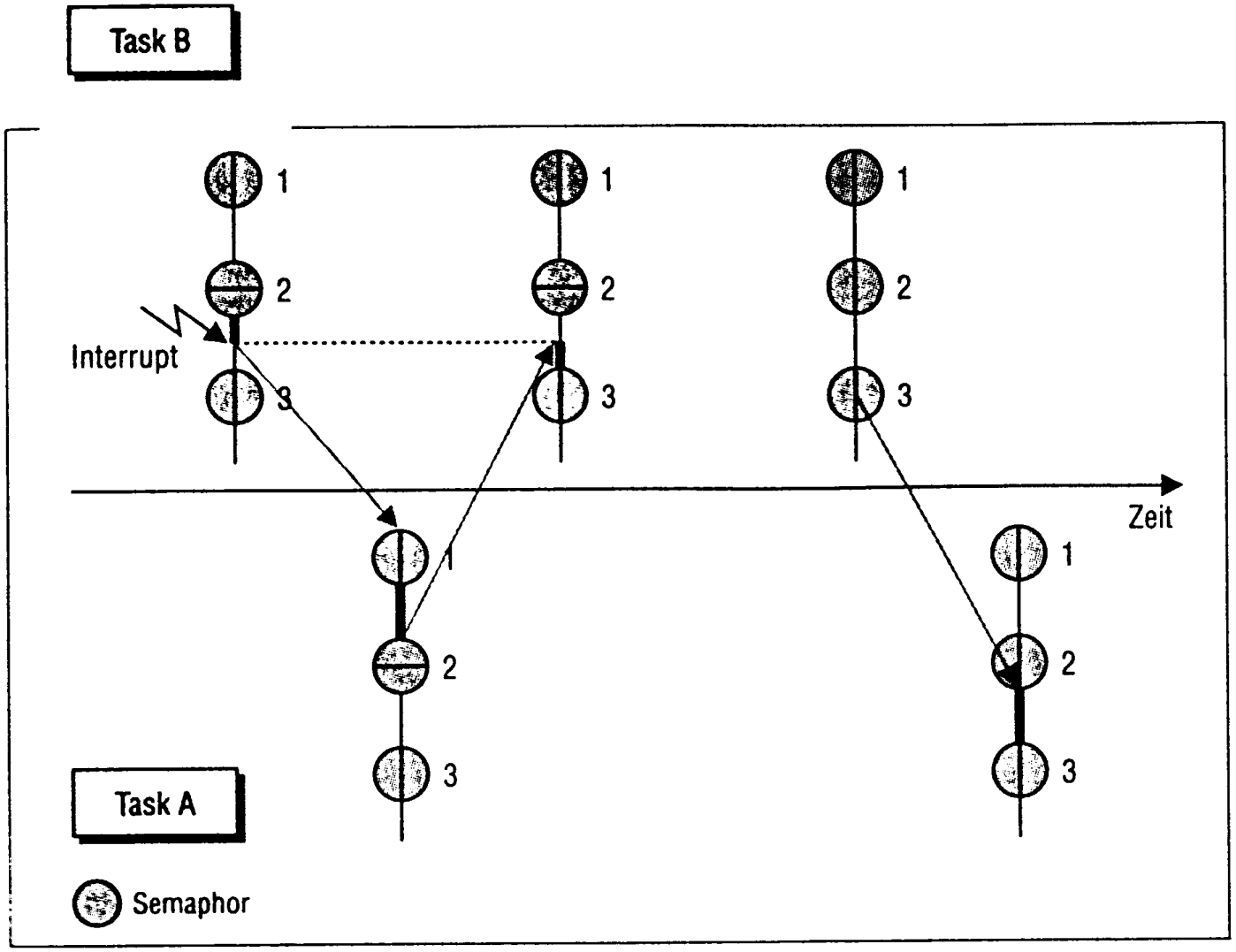



```
V (semaphore s) {
    s=s+1;
    if (s<=0) awake(s);
}
```

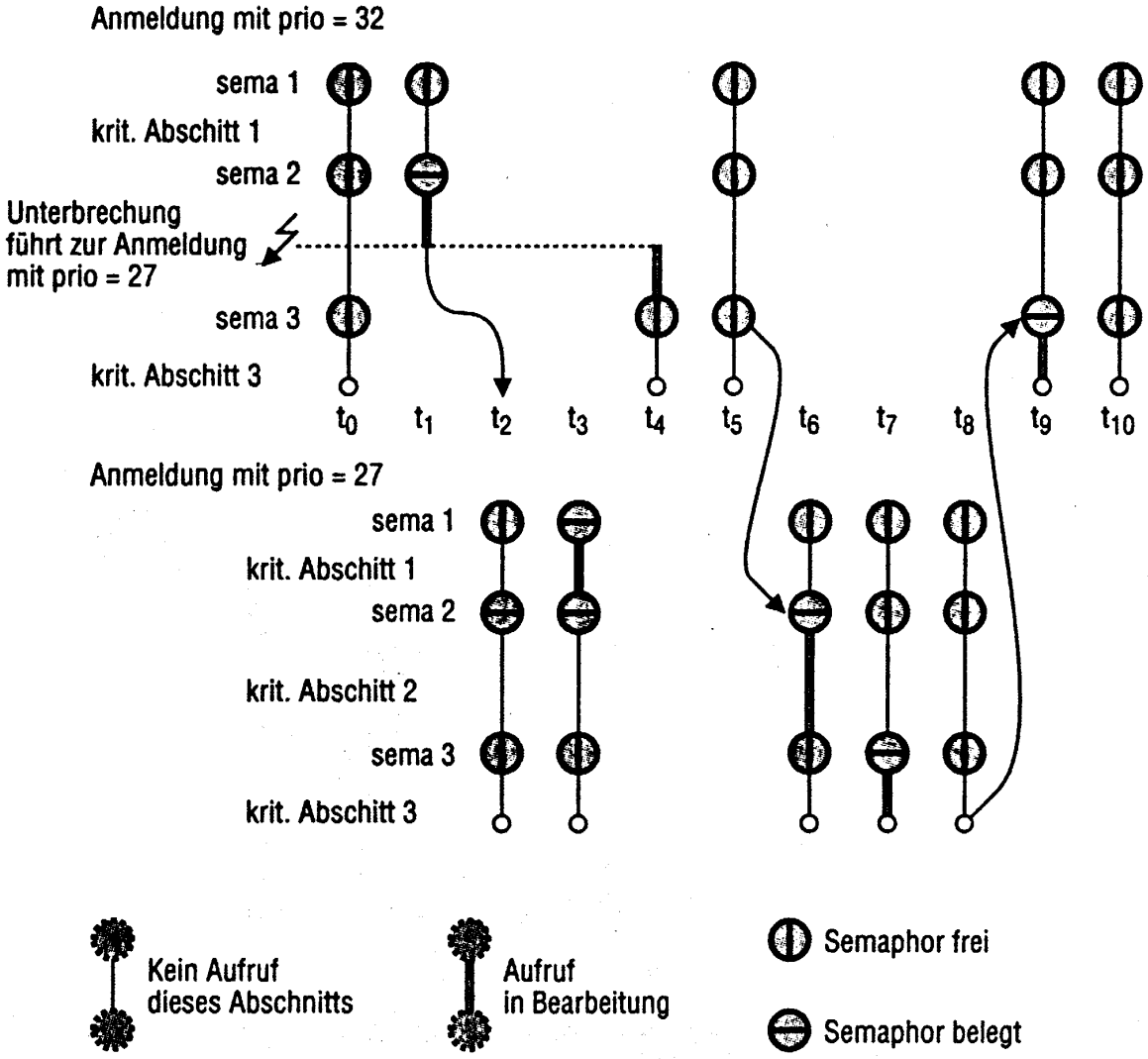
```
pr()   /* producer *
{ while (true)
        { /* produce data/item x */
        P(f);
        /*f>=0 && b<N */
        buffer[i]=x;
        i=(i+1) % N;
        V(b);
        /* f>=0 && b<=N */
    }
}
```

```
c()    /* consumer *
{ while (true)
        P(b);
        /*b<=0 && f<N */
        y=buffer[j];
        j=(j+1) % N;
        V(f);
        /* b>=0 && f<=N */
        /* consume data/item y */
    }
}
```

Synch in the OS: via Semaphores



Synch in the OS: via Semaphores

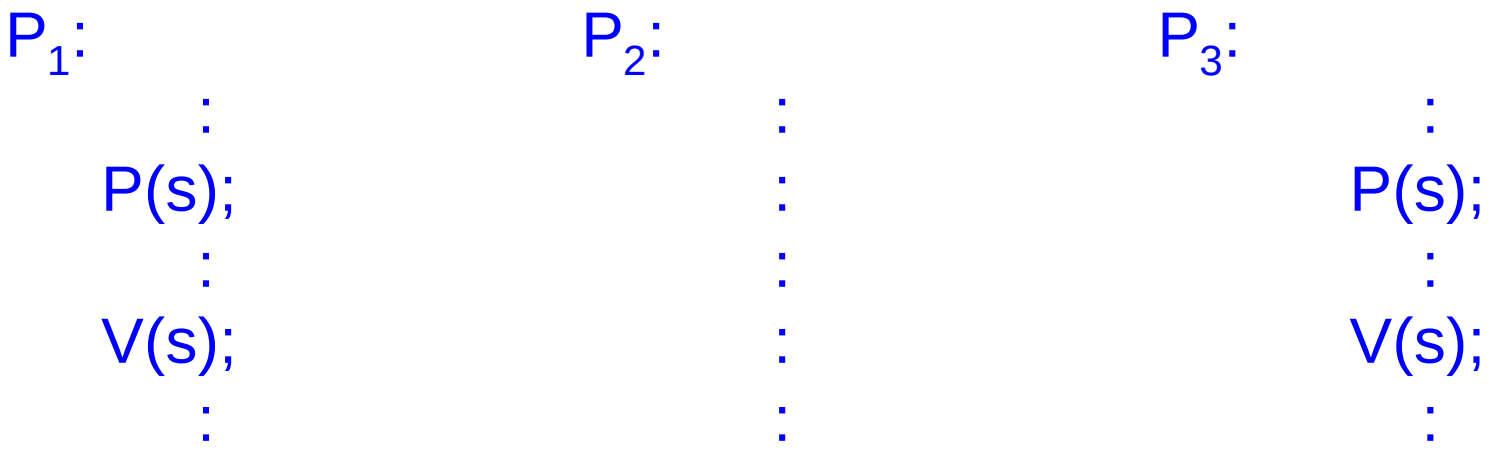


Semaphores: enhancements

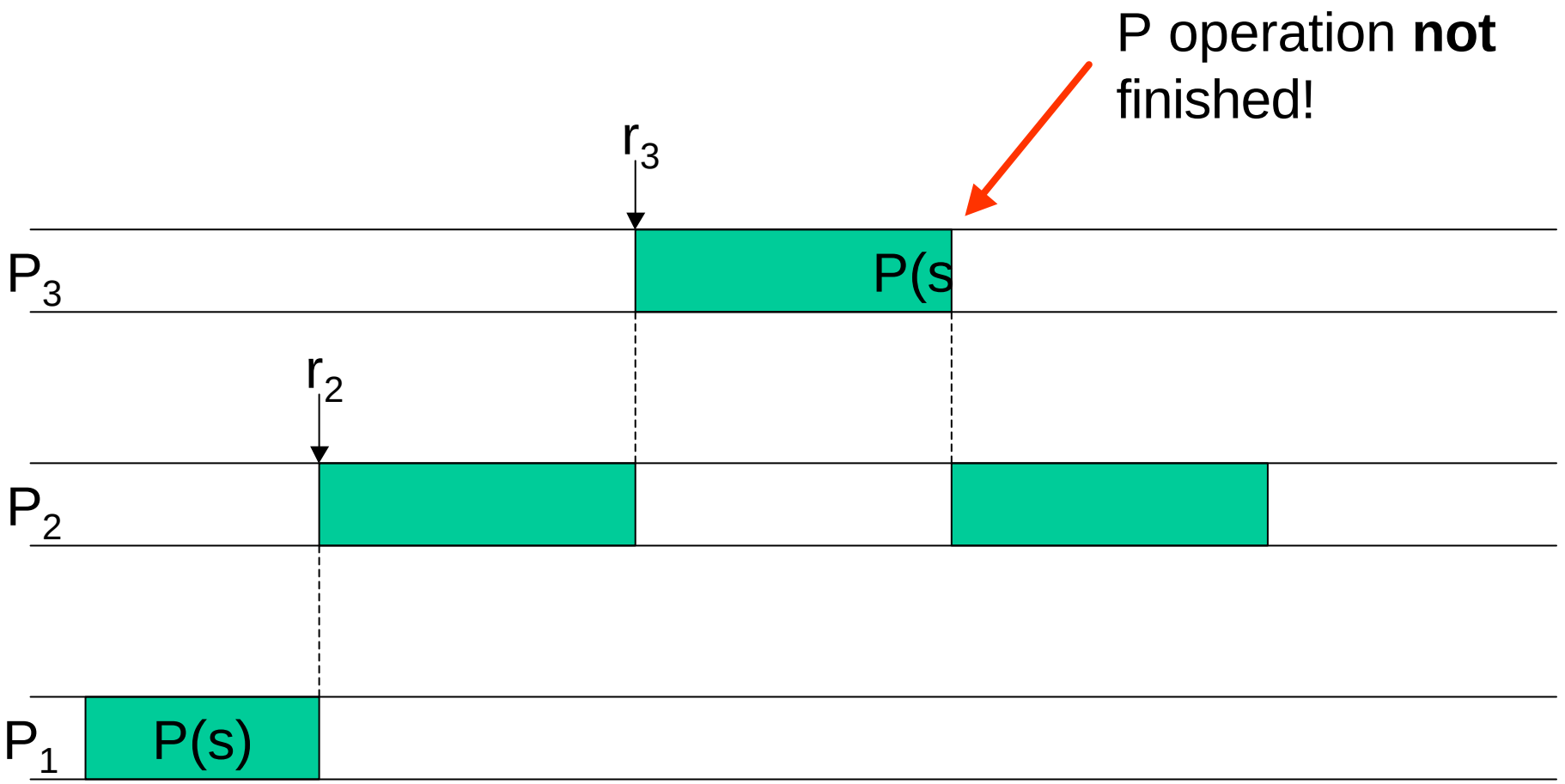
- explicite creation and deletion of semaphores (they are OS resources!)
- unique identification of semaphores and introducing to processes
- specific functional properties of semaphores
  - range of values
    - value 0..? : "normal" semaphore
    - value 0..1 : binary semaphore
    - value 0..N : finite semaphore
  - kind of delay
    - wait immediately
    - show, that resource not available
    - ...
  - organisation of waiting queue
  - decrement, increment other values than 1
  - delay due to P() concurrent to a timing constraint

Example: Priority inversion

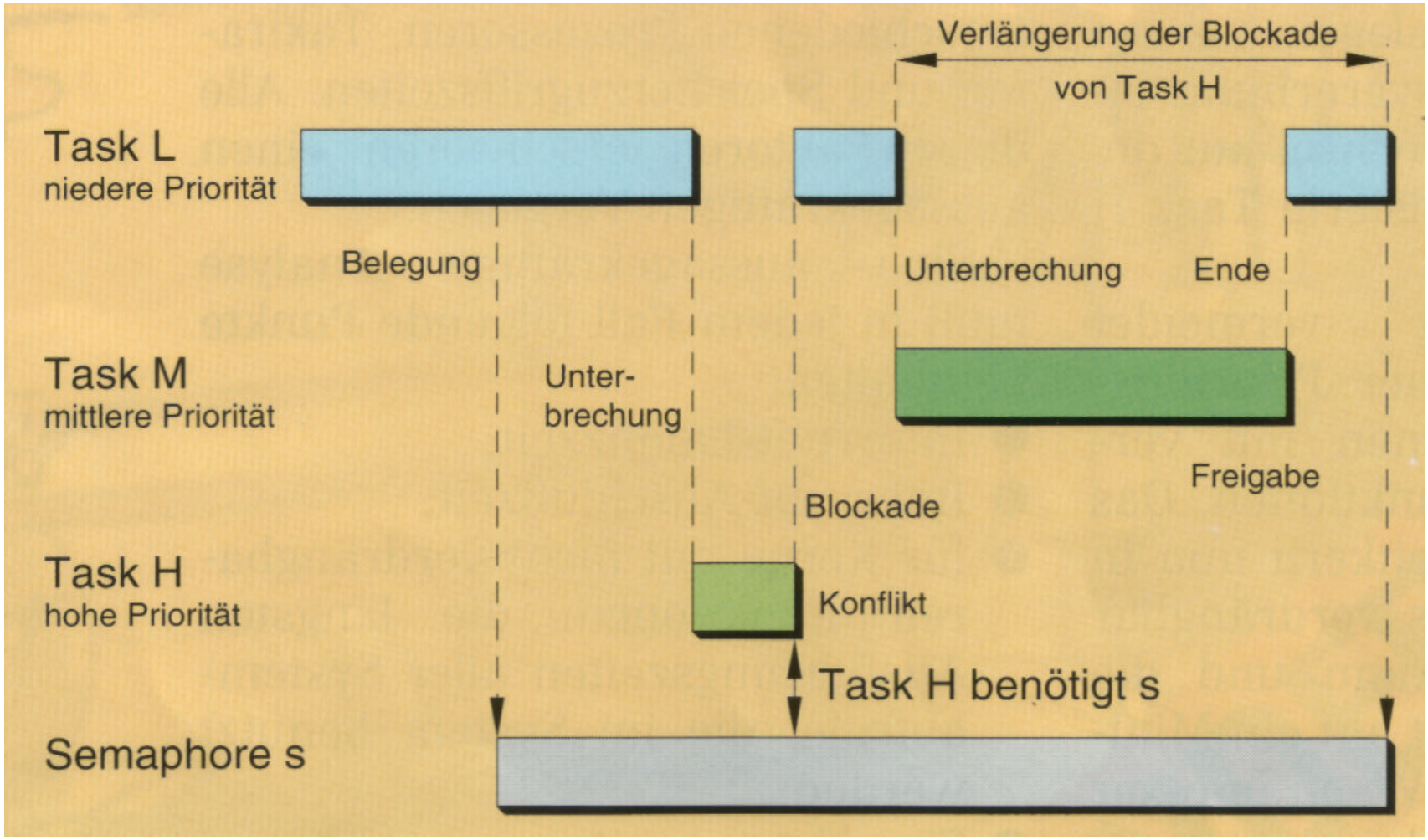
Given: processes P_i (i=1,..,3), their "times" (? e_i, r_i, s_i, d_i) and priorities (P_3 has the highest, P_1 has the lowest). Also the time ? s_1 is given, P_1 needs within the critical region (protected by semaphore s), analogue ? s_3 for P_3. What are the timing constraints for the processes (formulated as in 1.7, e.g. $P(s_2+? e_2 ? d_2 | P_3$ does not run) = 1) if they are structured as follows?



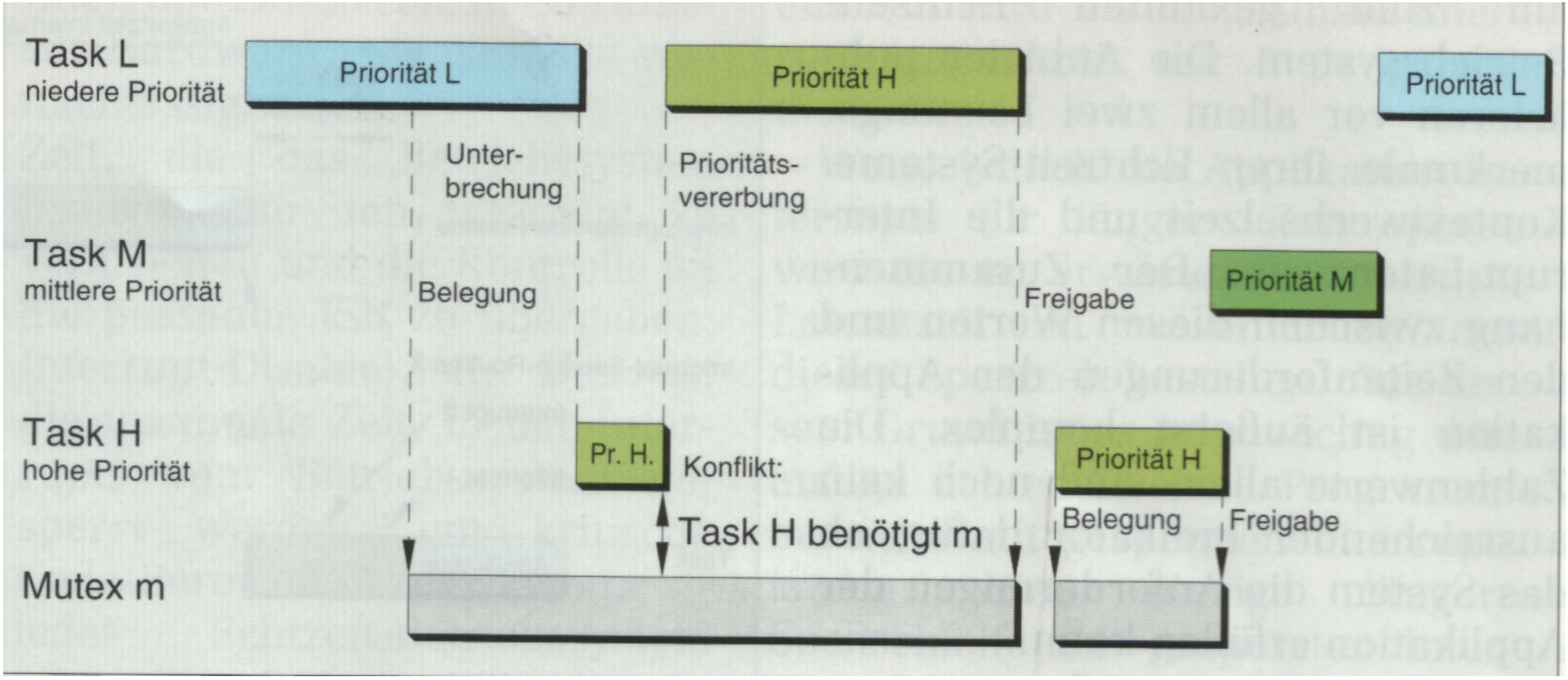
Example: Priority inversion



Priority inversion



Priority inheritance to fight priority inversion



/Zöbel95/

4.1 Characteristics of RTOS (Realtime Operating Systems)

4.2 The Kernel

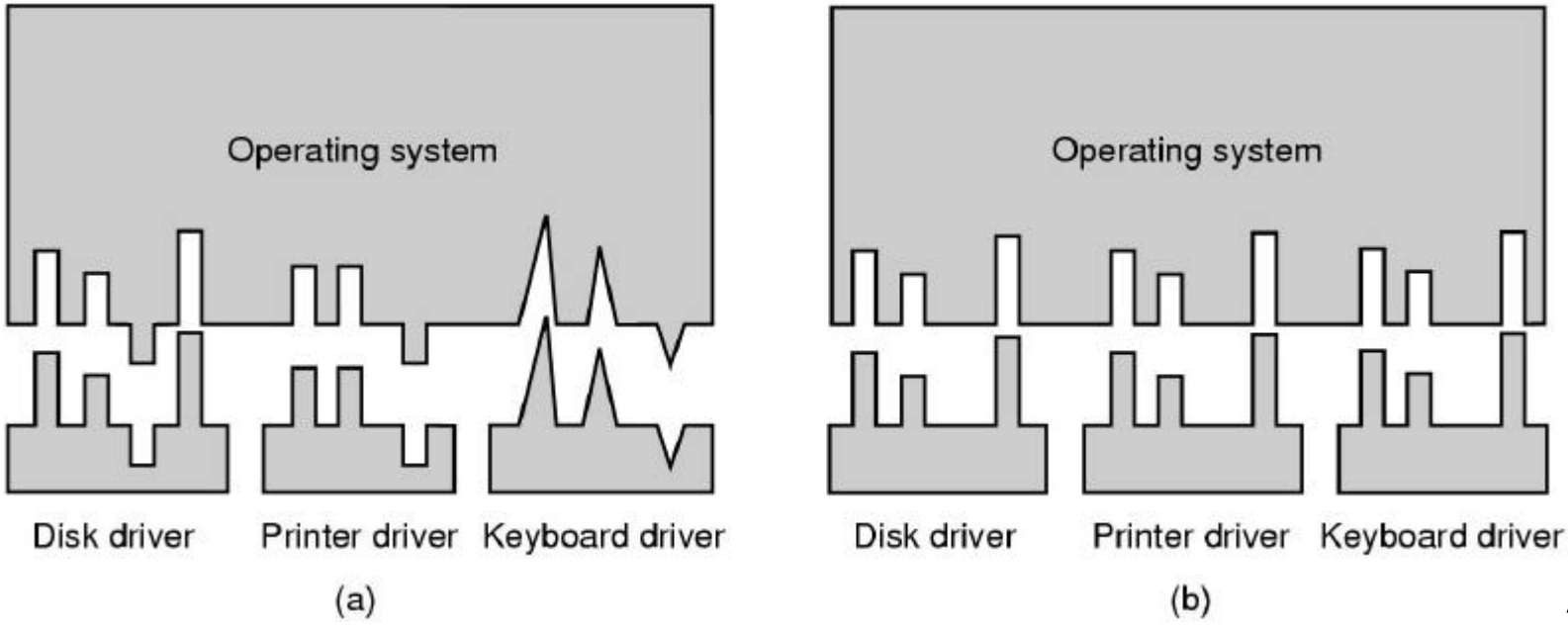
4.3 Synchronisation and priority inheritance

4.4 Driver

Driver

Application programmer: drivers belong to the HW.
OS developer: drivers are integral part of the OS.

History:
main motivation for development of UNIX was standardisation of driver interfaces (see b) below - without: see a))!

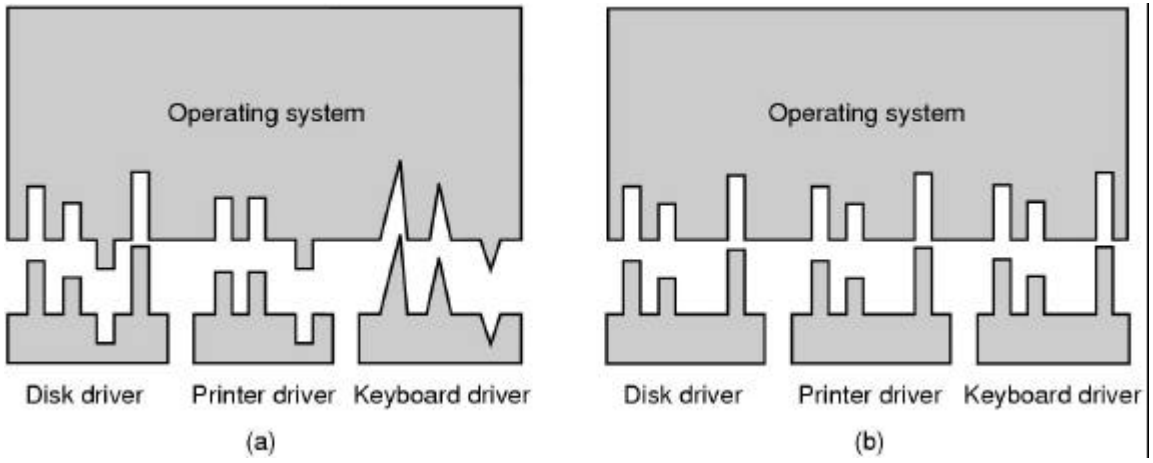


/Tanenbaum02/

Erinnerungen an die Zukunft

Geräteunabhängige Ein-/Ausgabe-Software
Standardisierung lohnt sich!

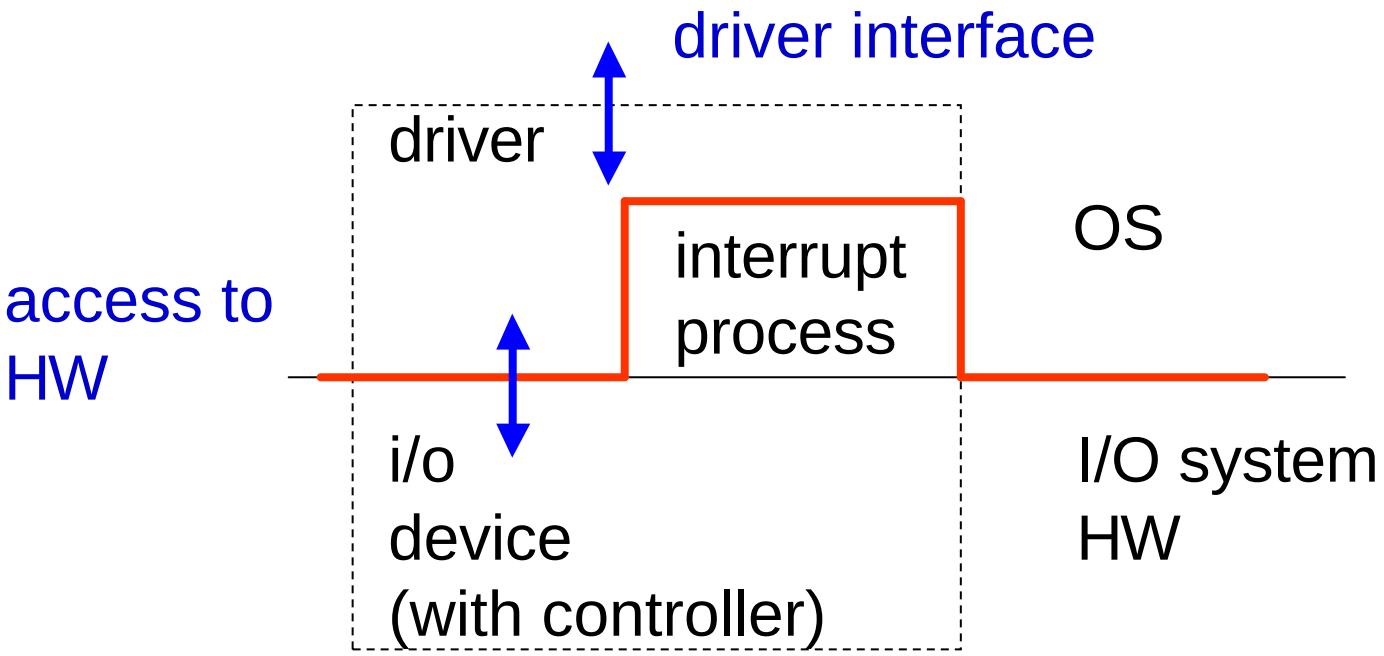
Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicate devices
Providing a device-independent block size



a) Ohne eine Standardschnittstelle für Treiber

b) Mit

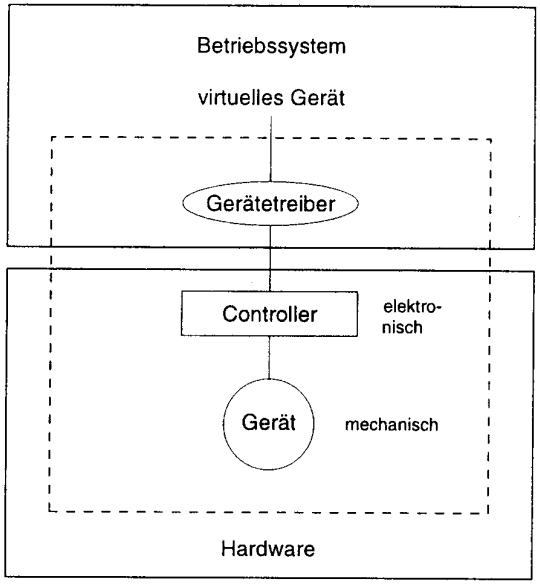
Driver and interrupt process

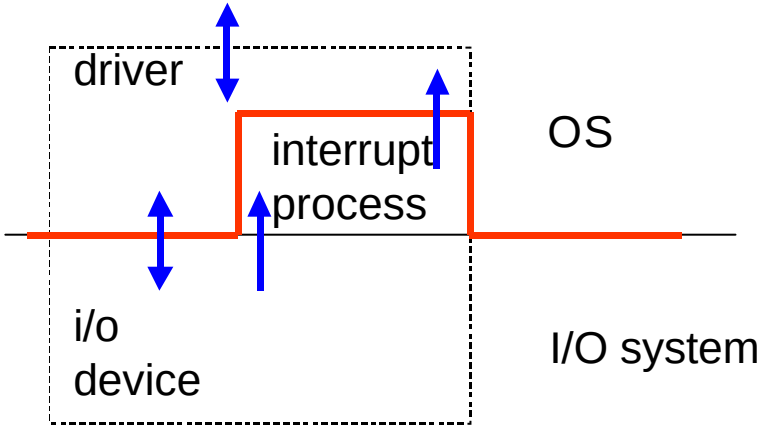


You know it already!

Driver and interrupt process

Interrupt processes plus drivers build lowest SW level for interface periphery.





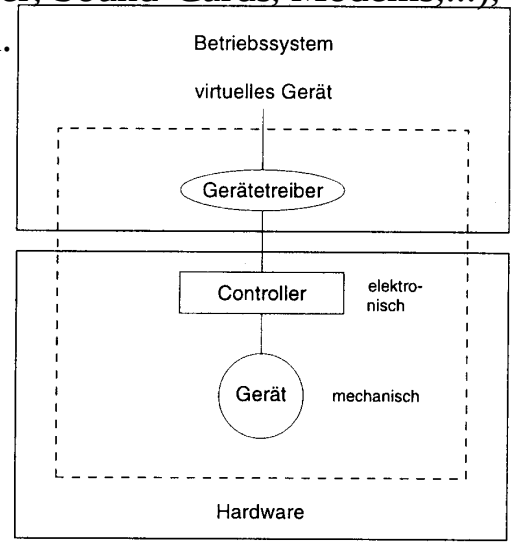
Driver has immediate access to i/o device but all interrupts are masked by the interrupt process (but may be forwarded to the driver).

Driver: part of the SW/HW interface

Aufgabe: Schnittstelle externe HW

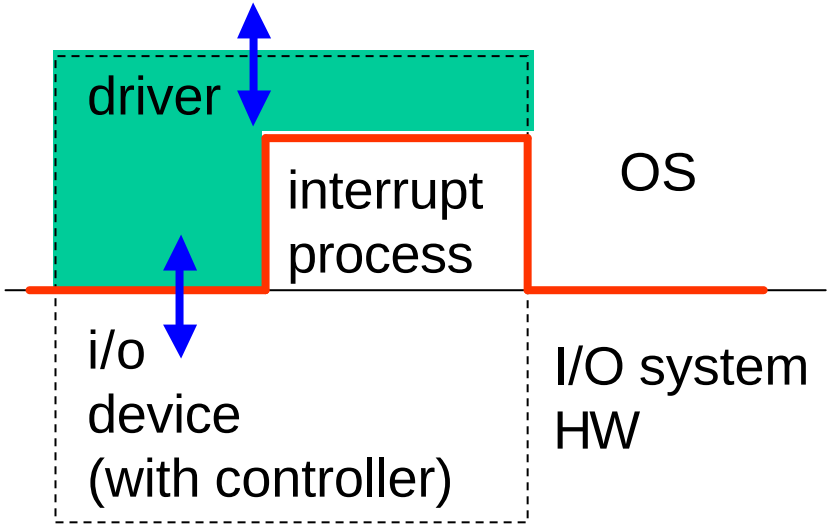
Applikationen sollten auf unterschiedlicher HW mit unterschiedlicher Peripherie laufen (z.B. Drucker, Scanner, Sound-Cards, Modems,...), also (Peripherie-)HW-unabhängig sein. Damit müssen (Peripherie-)HW-abhängige Dienste (s.o.) über das OS ´vermittelt´ werden.

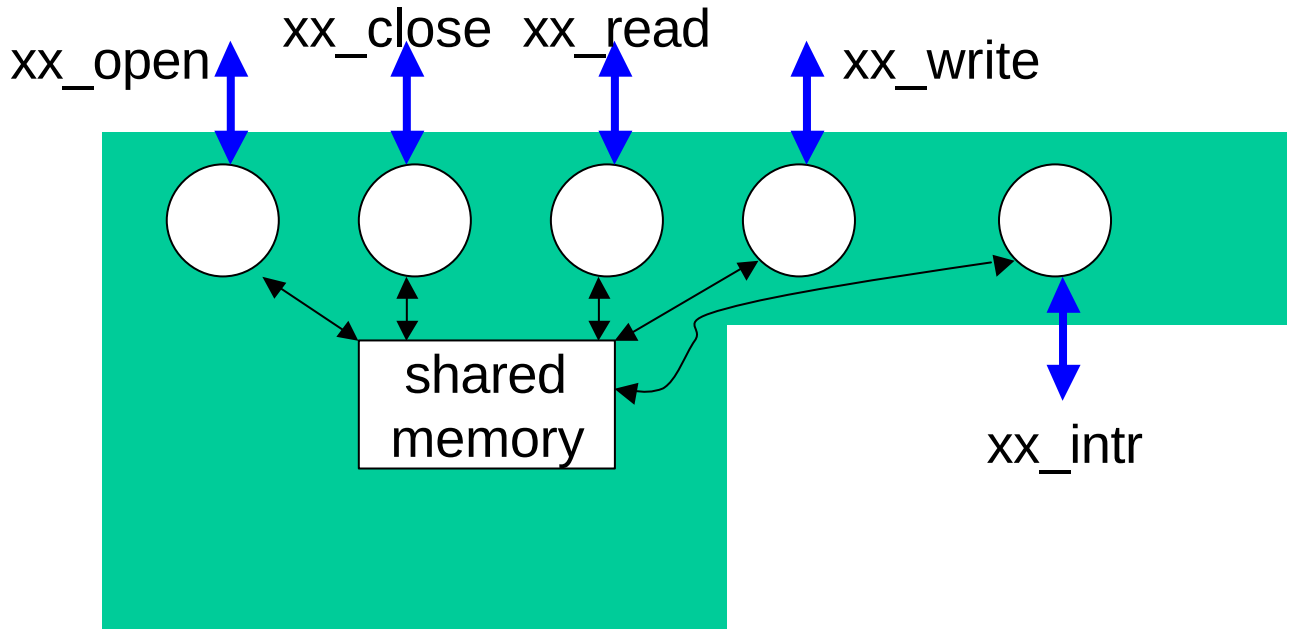
Stichwort: Treiber.



BSS1 1.8

Fine structure of driver - thread based





Example: driver interfaces in Solaris

Loadable driver interface:

```
   _init, _info, _fini
```

Device configuration:

```
   xx_getinfo, xx_identify, xx_probe,
   xx_attach, cc_detach
```

Device access:

```
   xx_open, xx_close, xx_read, xx_write,
   xx_strategy, xx_ioctl, xx_chpoll, xx_mmap,
   xx_print, xx_prop_op
```

What´s missing?

/Zöbel95/

4.1 Characteristics of RTOS (Realtime Operating Systems)

4.2 The Kernel

4.3 Synchronisation and priority inheritance

4.4 Driver

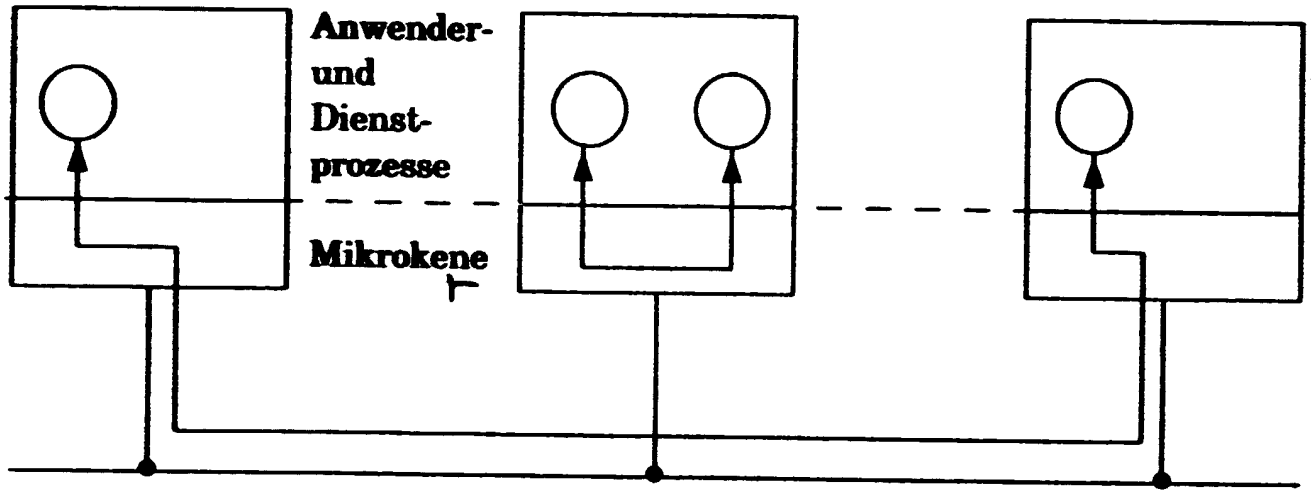
4.5 Distributed OS

Distributed (RT)OS

Due to RT constraints distributed RT systems are not connected via middle-ware but by a (common) distributed (RT)OS:

- distributed system are built from autonomous computers connected by a network
- on any of theses computers there the same micro kernel is installed
- that micro kernel offers IPC functionalities (inter process communication) based on messages
- based on the micro kernel all typical OS functionalities are offered as services - the OS can be adapted to the special needs of the applications

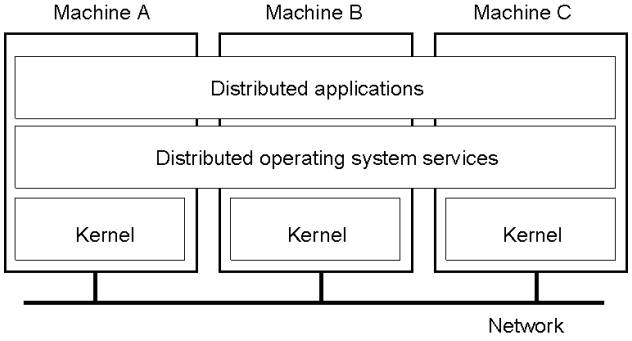
Distributed (RT)OS by micro kernels



The micro kernels realise an abstract distributed machine.

Distributed (RT)OS by micro kernels - you know it already

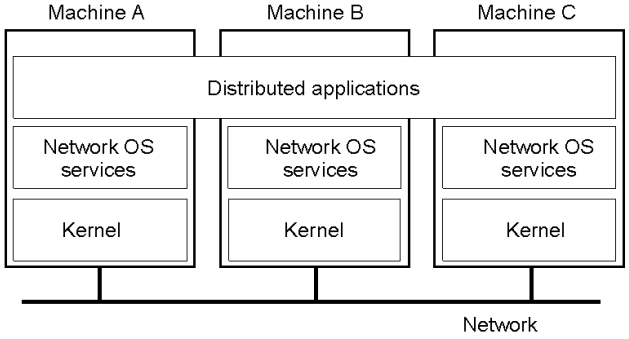
Vom (1-Prozessor-)Kernel zur Middleware



VS 1.2.13

General structure of a multicomputer operating system /Tanenbaum03/

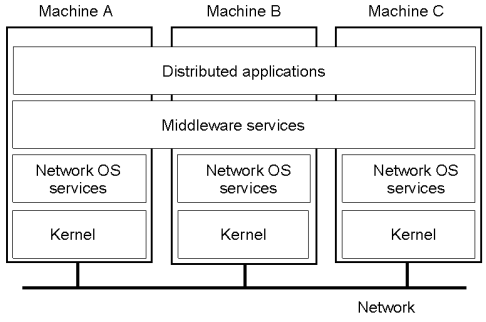
Vom (1-Prozessor-)Kernel zur Middleware



General structure of a network operating system. /Tanenbaum03/

VS 1.2.14

Vom (1-Prozessor-)Kernel zur Middleware

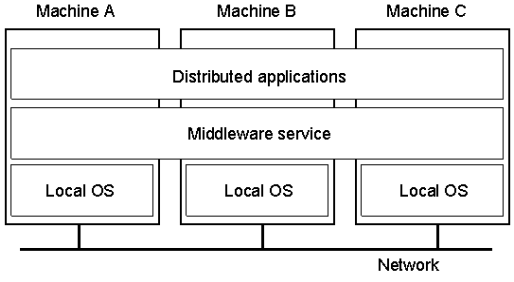


VS 1.2.15

/Tanenbaum03/

General structure of a distributed system as middleware.

Vom (1-Prozessor-)Kernel zur Middleware



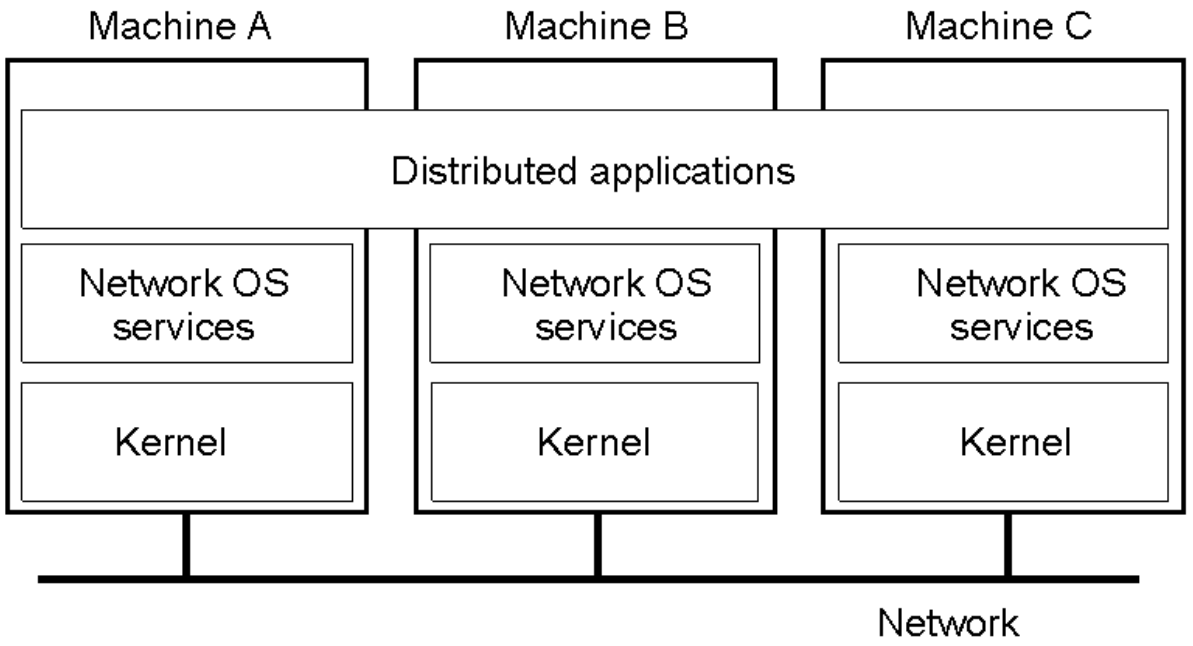
/Tanenbaum03/

A distributed system organized as middleware.
Note that the middleware layer extends over multiple machines.

VS 1.2.16

Distributed (RT)OS by micro kernels - you know it already

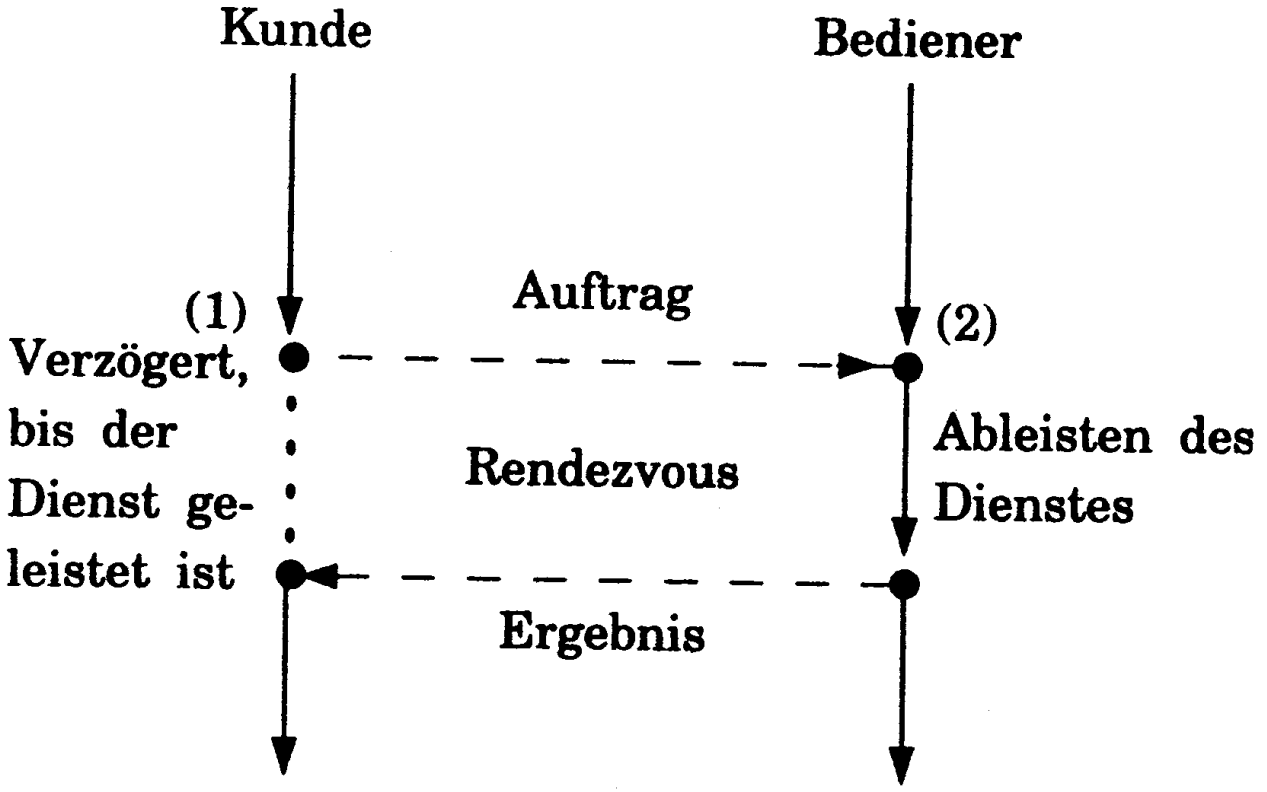
Vom (1-Prozessor-)Kernel zur Middleware



General structure of a network operating system. /Tanenbaum03/

VS 1.2.14

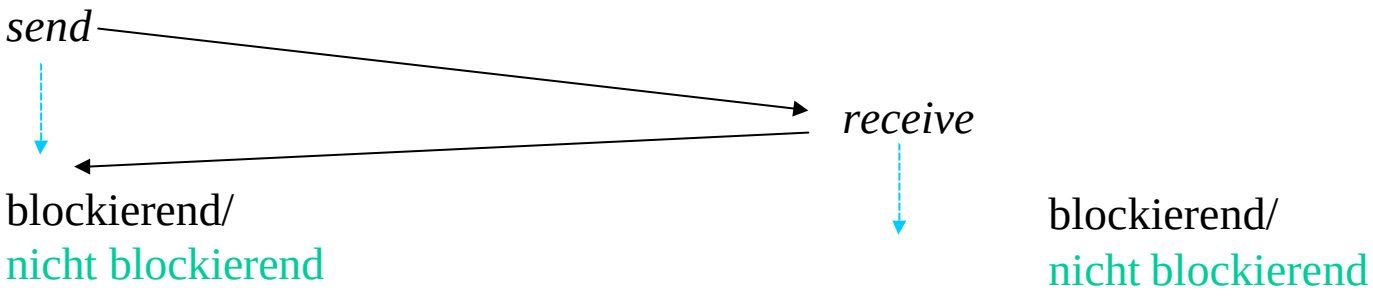
Distributed (RT)OS IPC by rendezvous



Synchronisation between client and server:
(1) client has to wait for server
(2) server has to wait for client tasks

Distributed (RT)OS IPC by rendezvous - you know it already

Noch mal synchron/asynchron - hier bei Kommunikation

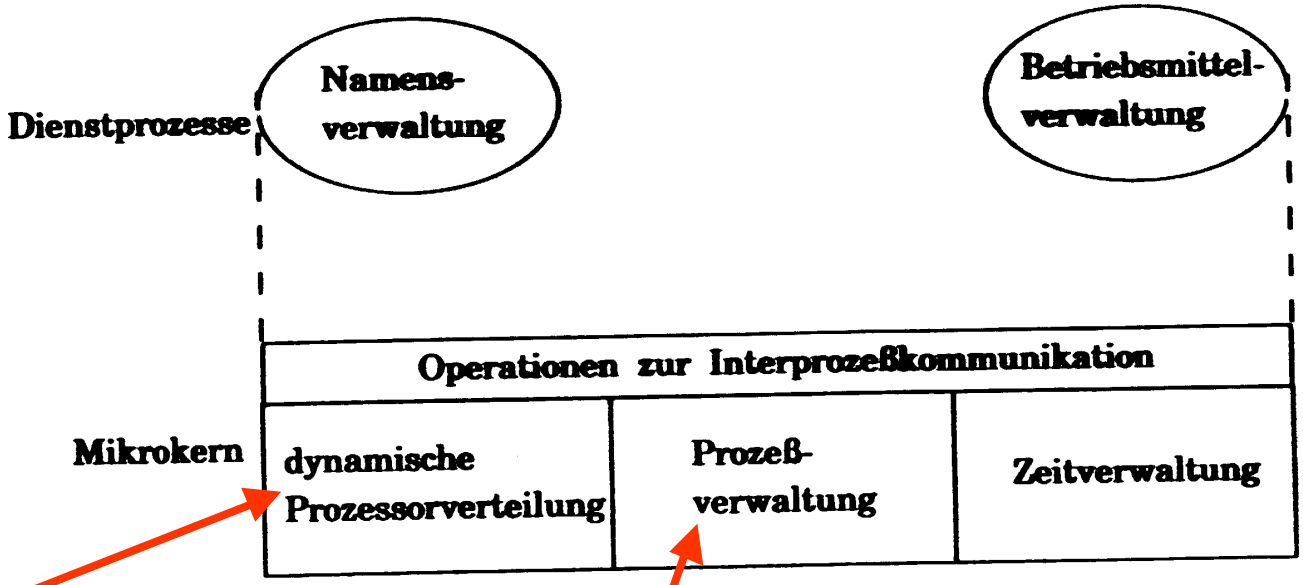


Bei **synchroner Kommunikation**:
- *send* blockierend
- *receive* blockierend

bei **asynchroner Kommunikation**:
- *send* nicht blockierend
- *receive* blockierend (Regelfall) oder nicht blockierend

Tasks of distributed (RT)OS

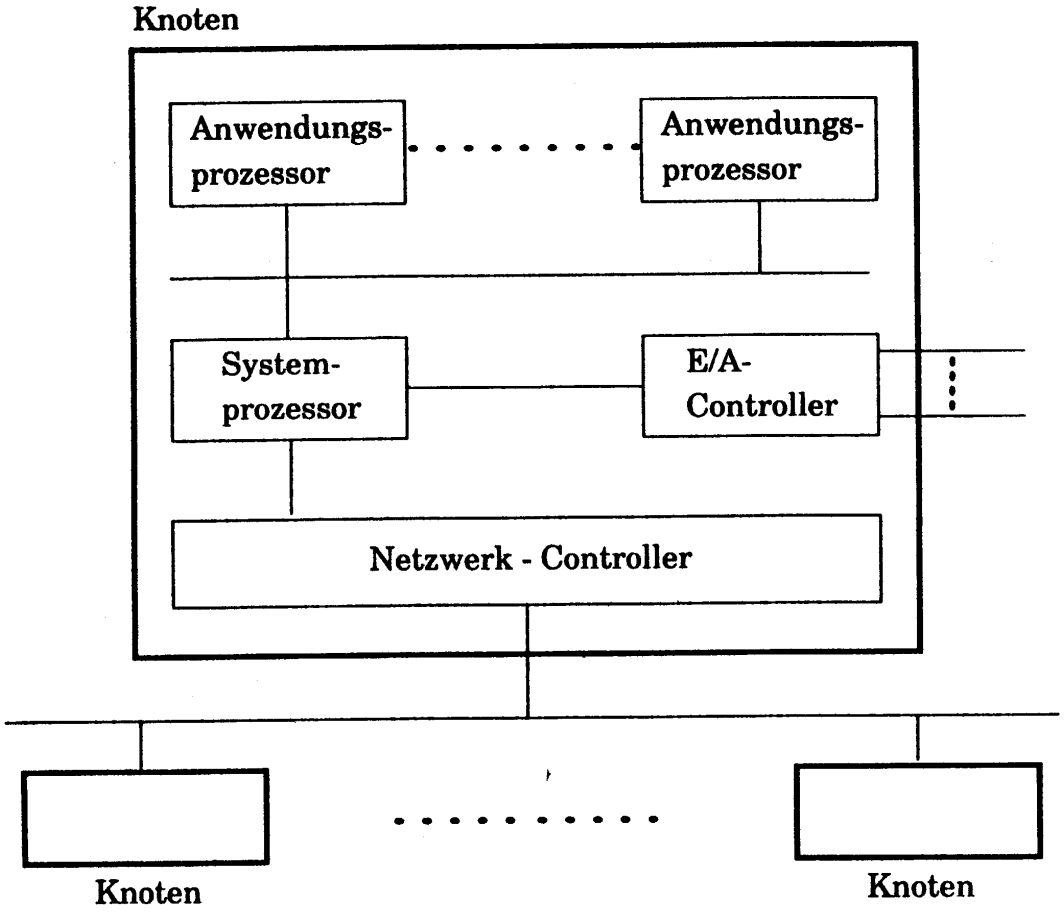
Structure of a micro kernel for distributed RTOS



CPU (one out of many!) administration

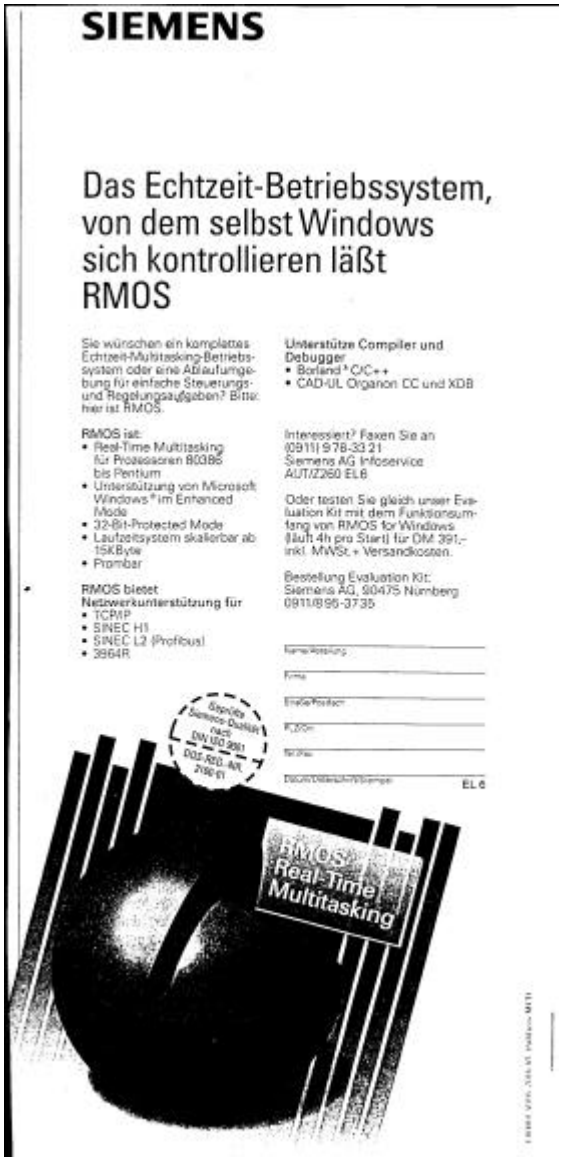
process administration

Distributed (RT)OS : Example Spring Kernel



Structure of SPRINGNET

(RT)OS vs. MS Windows



SIEMENS

Das Echtzeit-Betriebssystem, von dem selbst Windows sich kontrollieren läßt RMOS

Sie wünschen ein komplettes Echtzeit-Multitasking-Betriebssystem oder eine Ablaufumgebung für einfache Steuerungs- und Regelungsaufgaben? Bitte hier ist RMOS.

RMOS ist:
- Real-Time Multitasking für Prozessoren 80386 bis Pentium
- Unterstützung von Microsoft Windows® im Enhanced Mode
- 32-Bit-Protected Mode
- Laufzeitsystem skalierbar ab 15KByte
- Prombar

RMOS bietet Netzwerkunterstützung für
- TCP/IP
- SINEC H1
- SINEC L2 (Profibus)
- 3964R

Unterstützte Compiler und Debugger
- Borland® C/C++
- CAD-UL Organon CC und XDB

Interessiert? Faxen Sie an (0911) 978-3321 Siemens AG Infoservice AUT/Z260 EL6

Oder testen Sie gleich unser Evaluation Kit mit dem Funktionsumfang von RMOS for Windows (läuft 4h pro Start) für DM 39!,- inkl. MWSt. + Versandkosten.

Bestellung Evaluation Kit: Siemens AG, 90475 Nürnberg 0911/895-3735

Name/Abteilung

Firma

Straße/Postfach

PLZ/Ort

Telefon

Datum/Unterschrift/Stempel

EL6

RMOS Real-Time Multitasking