



Multimedia Systems

WS 2009/2010

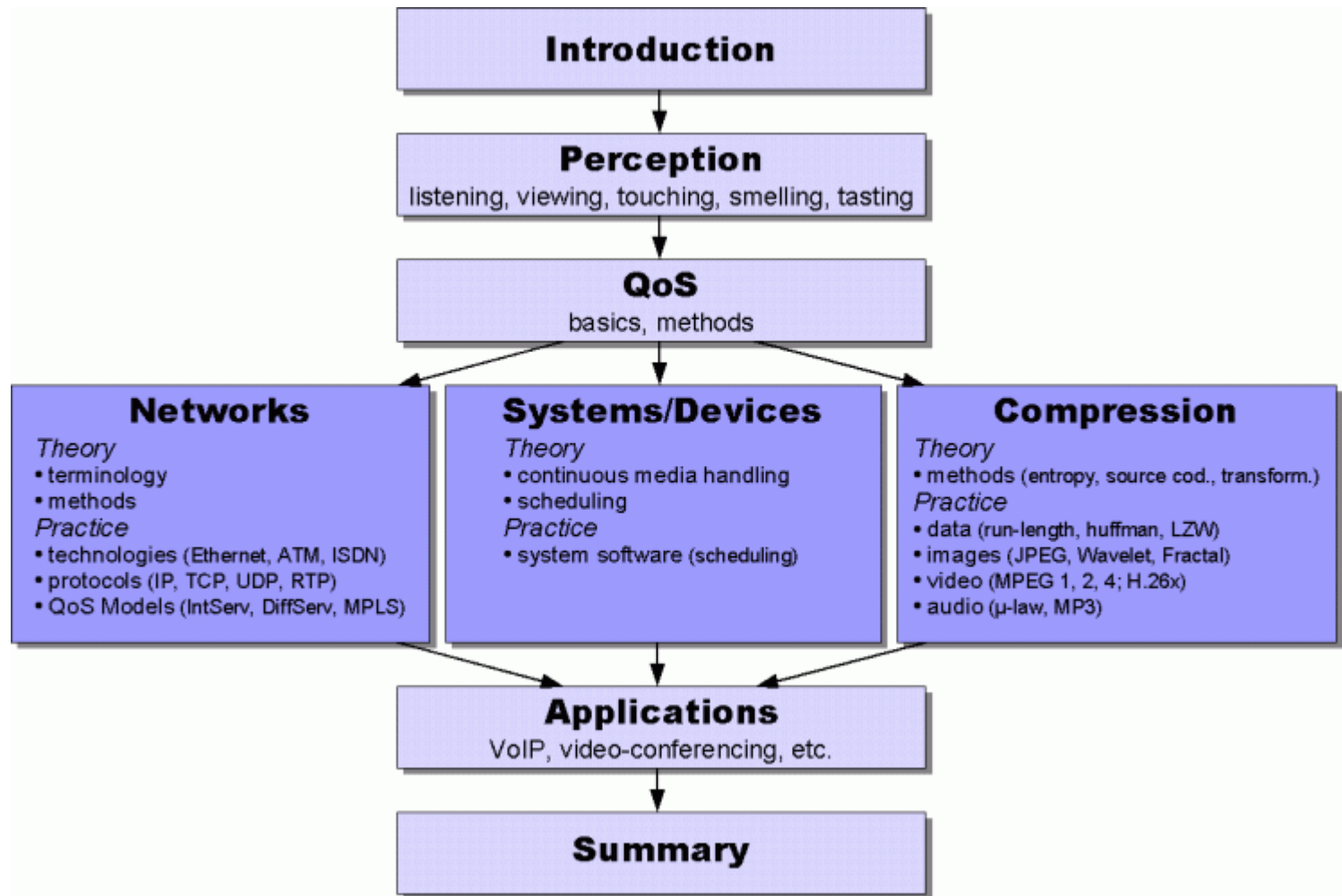
System Architecture

Prof. Dr. Paul Müller

University of Kaiserslautern, Germany
Integrated Communication Systems Lab

Email: pmueller@informatik.uni-kl.de

Sitemap

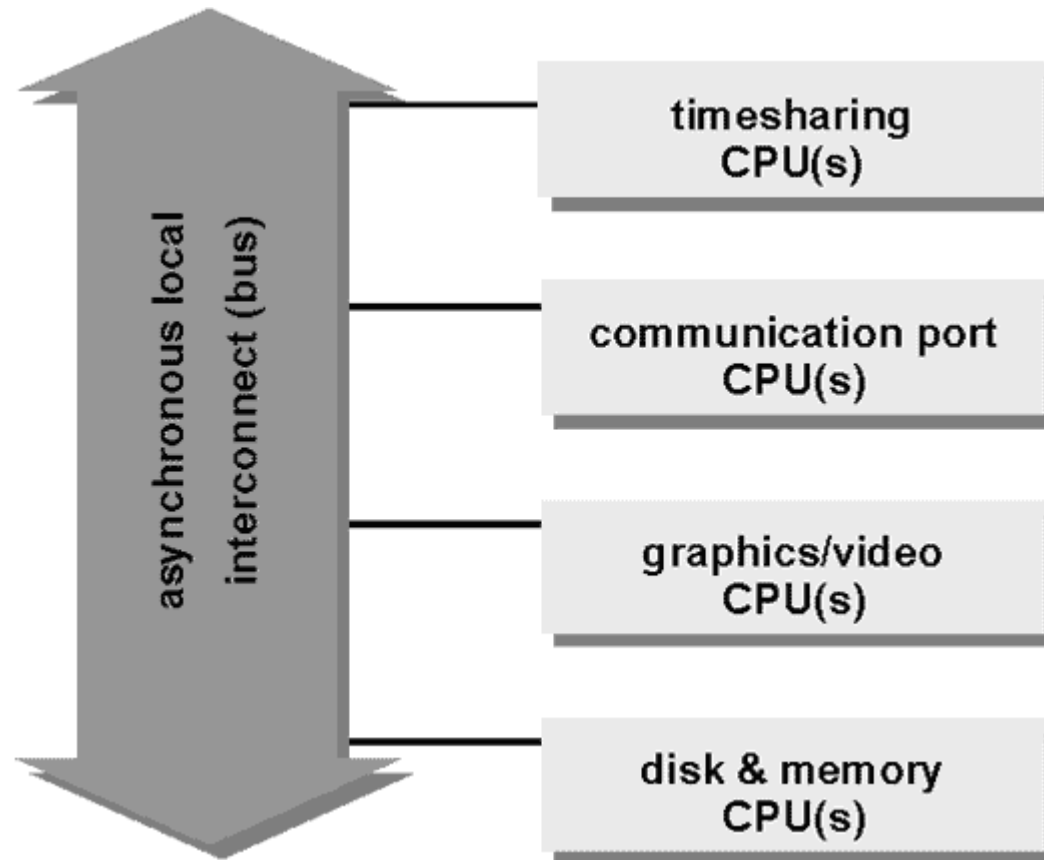


6.1. Hardware

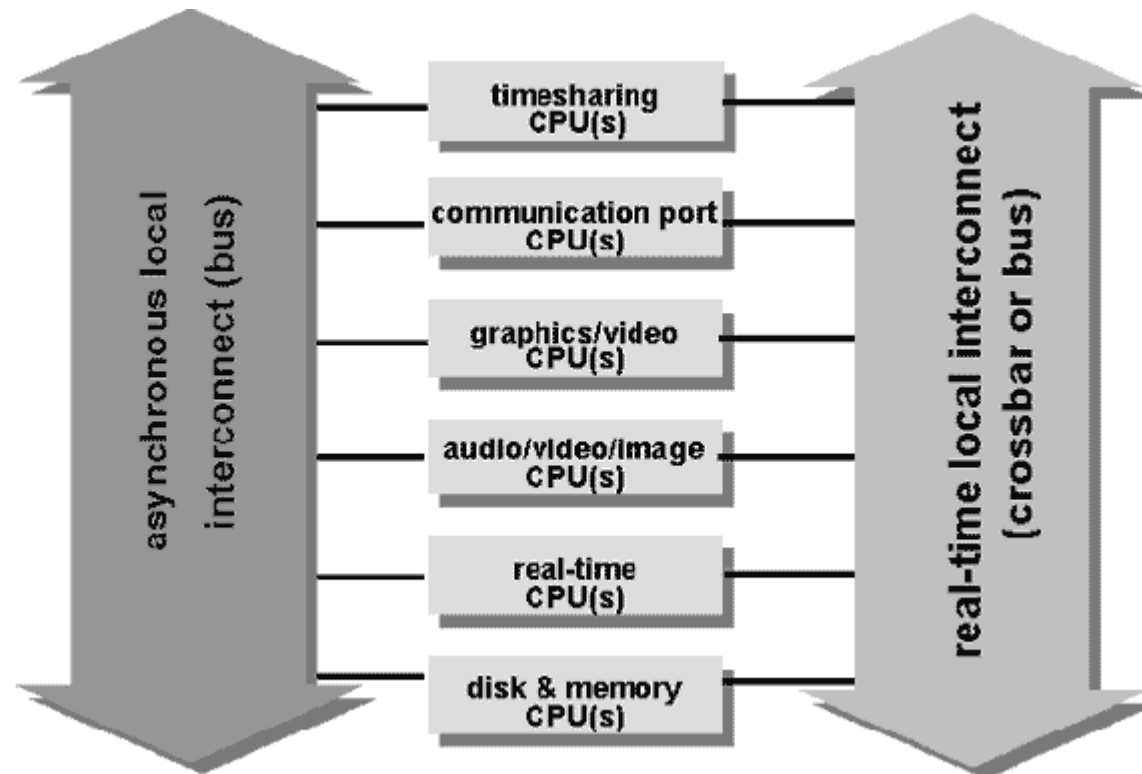
- **Workstation architecture**

- Processors:
 - CPU for processing discrete (and continuous) media
 - Signal processors
 - Processors dedicated to graphics, audio or video data processing
- Storage:
 - Hierarchy of several levels
 - Processor / Cache
 - RAM
 - Harddisks, tapes, etc.
- Input/Output
- Communication adapter
- Busses and interfaces

Workstation Architecture: Today



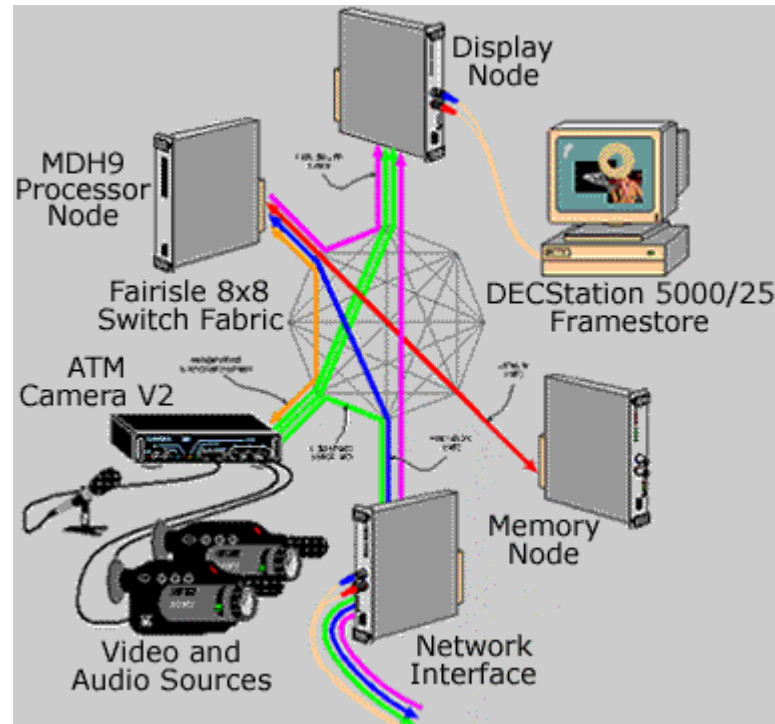
Workstation Architecture in the Future



Workstation Architecture: Switch Based

- **Network as interconnect for computer components:**
 - No bus
 - Internal network
 - No translation between multiplexing techniques used on bus and on external network
- **ATM switch**
 - Internal interconnection of computer system components
 - Directly connect components to high speed ATM network
 - Integrates external and internal network
 - ‘Everything uses ATM cells’
 - Same protocol techniques

Workstation Architecture: Desk Area Network (1)

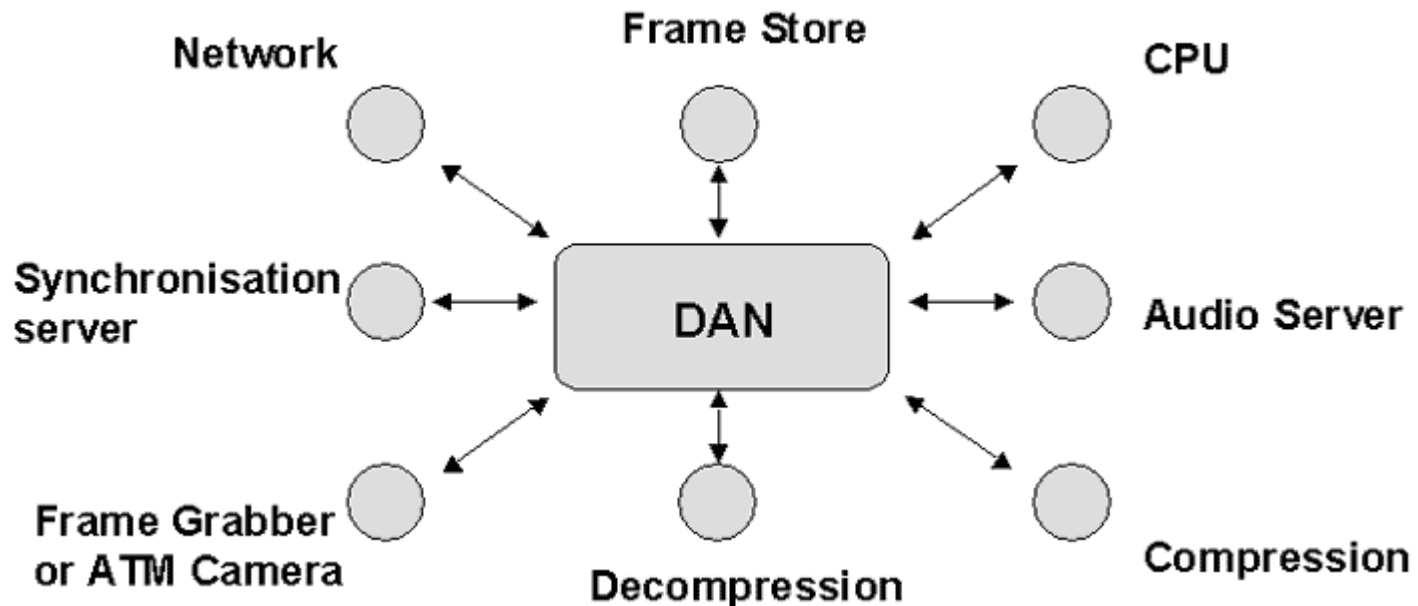


University of Cambridge U.K.

<http://www.cl.cam.ac.uk/Research/SRG/bluebook/10/dev/dev.html>

Workstation Architecture: Desk Area Network (2)

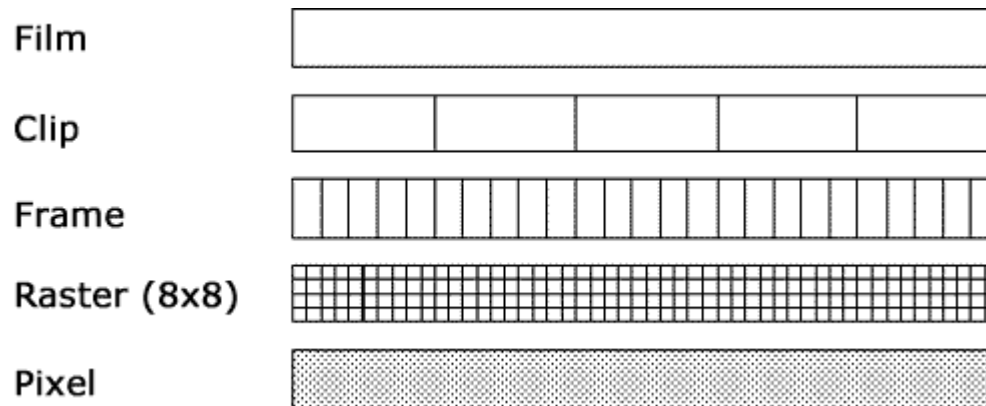
- **DAN (Desk Area Network) as multimedia workstation:**
 - Switch used to deliver AV streams directly to relevant devices



6.2. System Software

- **The system software provides the environment for all processes and applications**
- **Today's widespread system software was designed to handle discrete data only**
 - Therefore Continuous Media is viewed as periodical and discrete data
 - Logical Data Units (LDUs) of different granularity

Example:

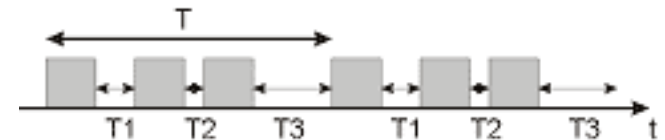


- closed LDUs: known length and number (e.g. film on a hard disk)
- open LDUs: unknown length or number (e.g. input of a camera)

Properties of Continuous Media

- **Periodicity**

- strongly periodic, fixed inter arrival time
- weakly periodic, variation of inter arrival time is fixed
- not periodic



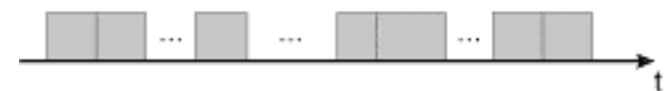
- **Joining**

- joined packages, no gap between packages (strongly periodic)
- not joined



- **Size**

- strongly regular size, all packages have the same size
- weakly regular size, variation of package size is fixed
- irregular



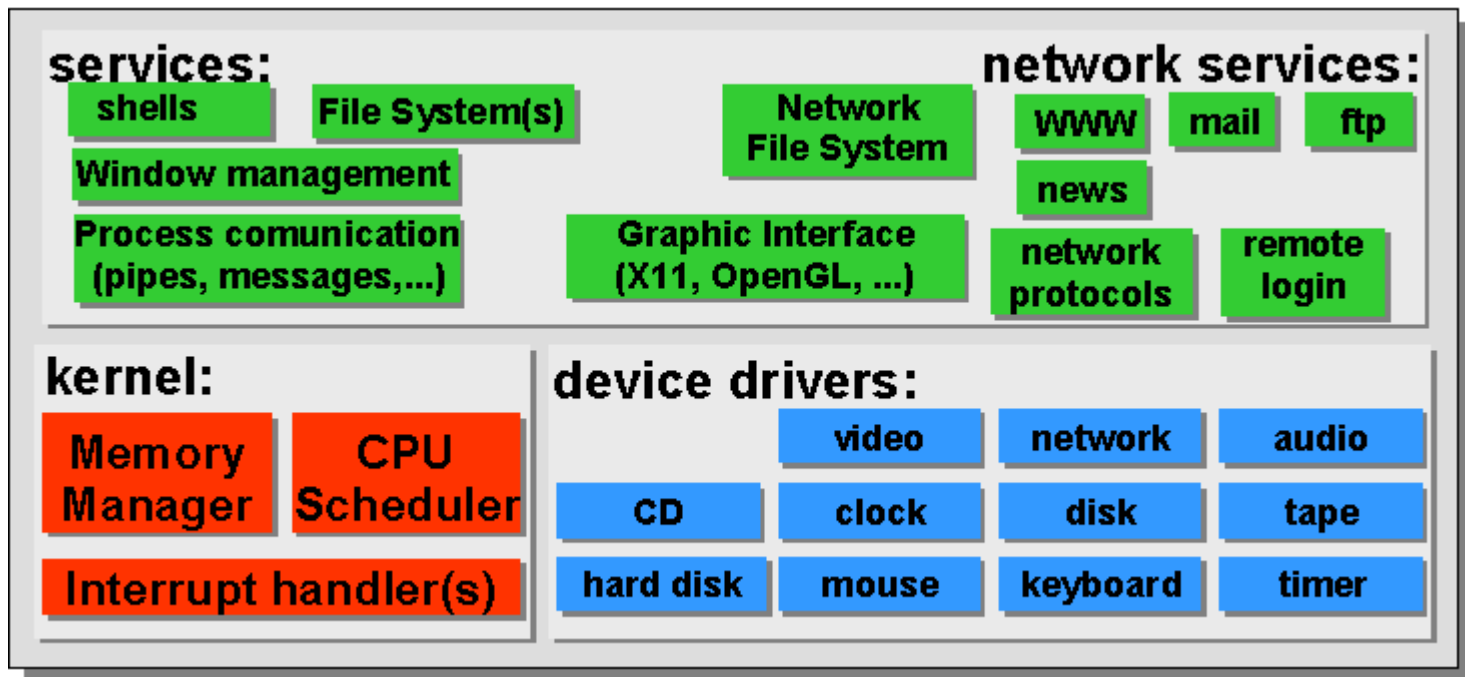
System Software Tasks

- **Typical tasks:**
 - Provide an interface to the rest of the world for processes
 - Access to I/O devices
 - Abstraction of hardware
 - Inter-Process communication and synchronization
 - Operation of hardware
 - Prepare data for hardware components
 - Control the hardware and respond to interrupts
 - Organize the storage of data on hard disks, tapes, etc.
 - Management of access to system resources (memory, CPUs, storage media, network, other I/O devices)
 - When to access
 - Who is allowed to access
 - Reservation of resources
 - More ...
- **System software should fulfill all tasks with respect to the time requirements of continuous media processing**

Examples: System Software Tasks

Applications

(Video Conference, CD player, word processor, ...)

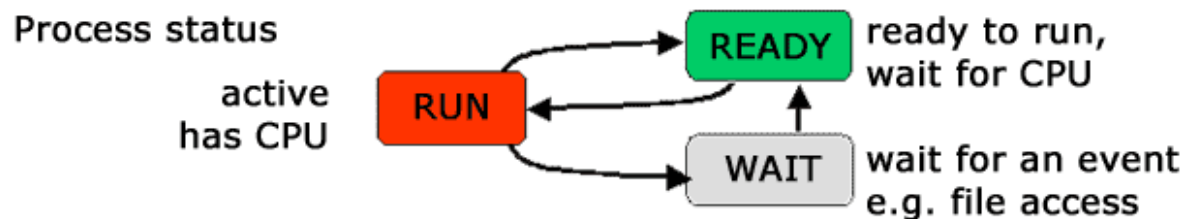


Hardware

(memory, CPU(s), system bus, devices)

Management of CPU access

- **Multitasking allows several processes (applications or services) to run concurrently**
 - several processes compete for resources
 - Assign a CPU for a short time (e.g. 10 ms) slices to a process
- **Process scheduling:**



- **Components:**
 - Scheduler: planning order of processes
 - Dispatcher: assign a new process to a CPU



Process Scheduling (1)

- **Processes are assigned to a CPU according to a specific scheduling strategy**
 - The strategy should be fair
 - Each process gains access to a CPU sooner or later
 - Priorities define how often, how soon and how long a process is assigned to a CPU
 - Different strategies could be applied to classes of processes
 - Further details [lecture System Software](#)



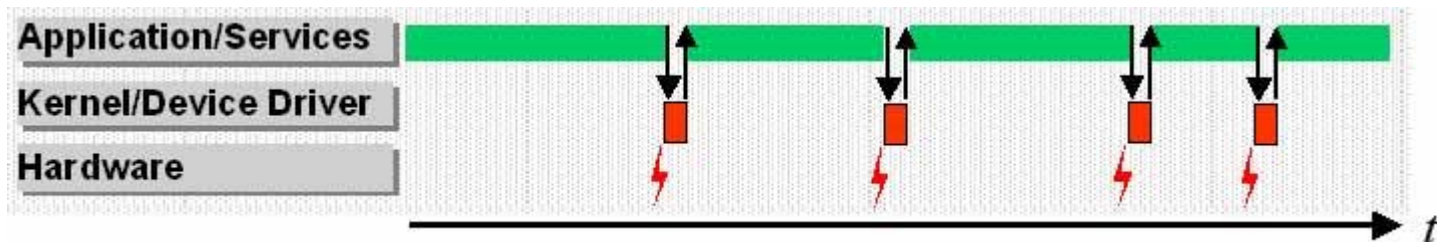
Process Scheduling (2)

- **Example scheduling in Windows NT:**
 - Scheduling priorities depend on process + thread priorities
 - Processes of the modes **IDLE / NORMAL / HIGH** are scheduled by a fair strategy
 - Process priorities are raised and lowered dynamically to implement fairness
 - **REALTIME** modes processes are assigned to the CPU in order of precedence
 - A **REALTIME** process may block the whole system
 - This mode is usually used for few system processes only

Thread \ Process	IDLE	NORMAL	HIGH	REALTIME
TIME CRITICAL	15	15	15	31
HIGHEST	6	10	15	26
NORMAL	4	8	12	24
LOWEST	2	6	11	22
IDLE	1	1	1	16

Interrupts / Deferred Procedure Calls (DPCs)

- **Interrupts are handled by the kernel and device drivers:**
 - about 1000 Interrupts per second must be handled, i.e. about 10 within a typical time slice
 - the utilization of a time slice depends on
 - The number of interrupts
 - The required processing time for each interrupt
 - DPCs are routines executed within the kernel, i.e. with high priority
 - hardware interrupt handlers queue DPCs to finish work associated with the interrupt
 - reduces the interrupt latency
 - increases thread scheduling latency for user threads
- principle of interrupts:

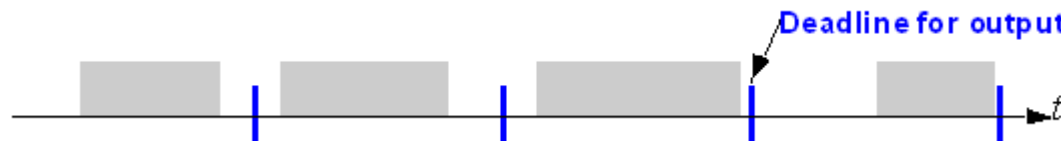


Problems of Continuous Media processing

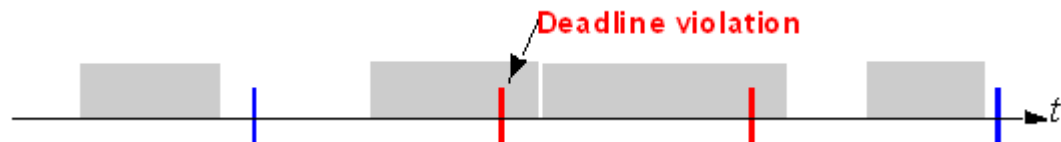
- Irregular behavior of applications makes periodical processing of continuous media difficult, especially for real-time communication.
- Consider the arrival, processing and output time of an LDU:
 - Ideal: regular arrival and constant processing time, leads to regular output



- Acceptable: limited variation of arrival and processing time



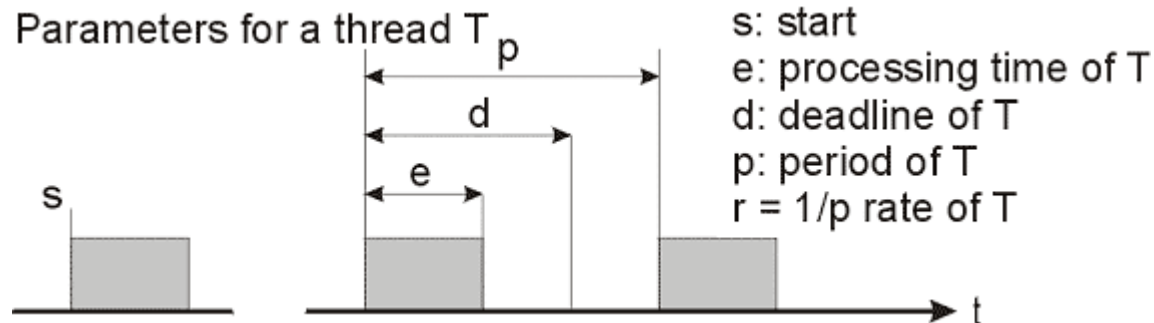
- Not acceptable: increased delay or jitter leads to deadline violation



Real-Time Systems

- Real-time processing means, it can be guaranteed that the processing results are available within a given time interval.
- Properties of Real-Time systems:
 - The processing time for all system components must be predictable
 - Reservation of all types of resources must be possible
 - Short response time for time critical processes
 - The system must be stable at high load
- Two types of real-time guarantees:
 - Hard real-time guarantees means the real-time requirements must be fulfilled. Typically used for controlling industrial processes or electronic controls in cars or airplanes.
 - Soft real-time guarantees means the real-time requirements should be fulfilled. Typically used for multimedia systems.

Real-Time Scheduling Principles



- **Components for real-time scheduling:**
 - CPU-Broker: test if a new real-time process could be accepted, based on scheduling rules
 - Scheduler: determine order of processes, e.g. by priorities
 - Dispatcher: assign process to CPU
 - Problem: parameters often not known in advance, but measuring parameters at run-time requires restart of CPU-Broker no guarantees
- **Distinguish**
 - Realtime processes (RT-Processes) with explicit time requirements
 - Timesharing processes (TS-Processes) having best effort requirements

Earliest Deadline First (EDF)

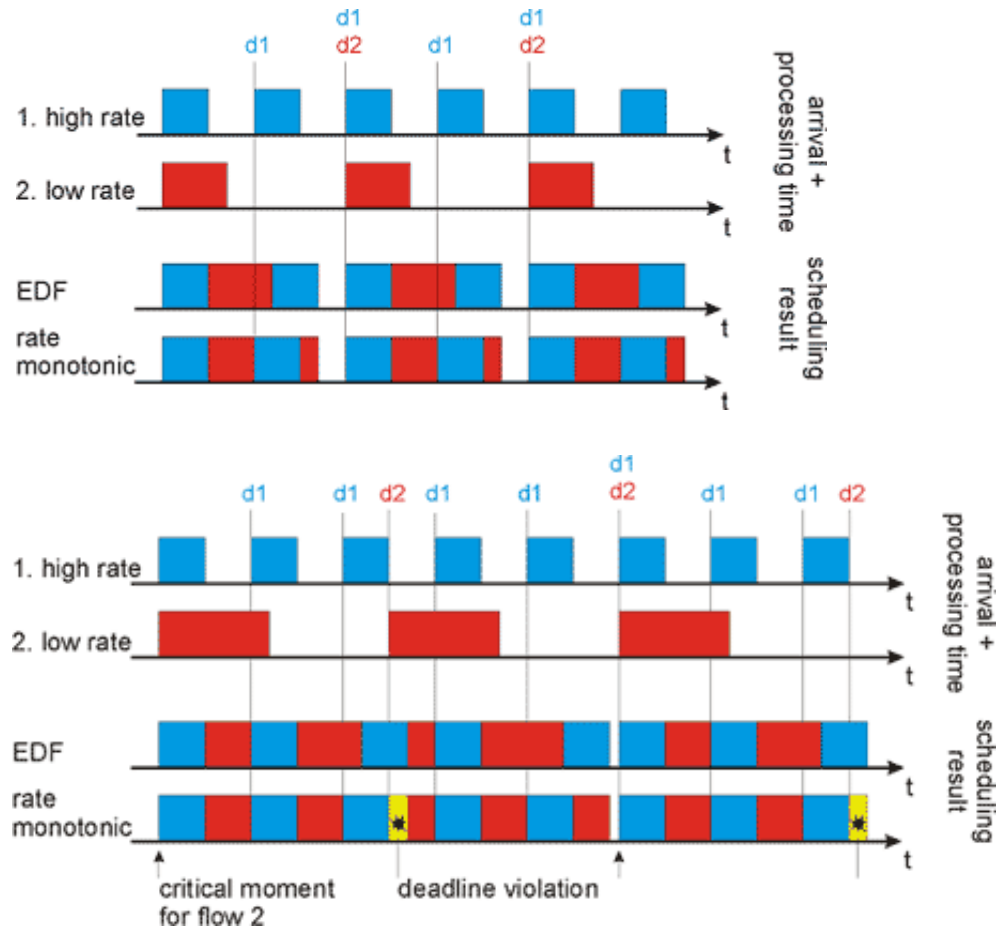
- **From a set of processes select the process with the earliest deadline**
 - processes must be preemptive
 - a process must be interrupted immediately when a new process with an earlier deadline changes to status "Ready".
 - EDF is an optimal dynamic method
 - optimal means, if there is a schedule for a set of RT-processes to meet their deadline, then EDF will find it
 - EDF is dynamic, i.e. there is no fixed schedule plan for all processes EDF could be used even with varying e and d
 - If the processing time of all processes are known, then it is possible to detect if a process will not keep its deadline. Those processes may be removed/skipped to avoid overload
- **Another scheduling strategy**
 - Shortest Job First (SJF)
 - May be static or dynamic
 - Guarantees that as many processes as possible keep their deadline

Rate Monotonic Planning

- **Process priorities are related to the rate, the higher the rate the higher the priority**
 - Rate Monotonic Planning is an optimal static method. Static means a process plan is made only once.
 - Necessary assumptions:
 - All real-time processes are periodic
 - Deadline equals the beginning of the next period ($d = p$)
 - Processes are independent to each other
 - e is fixed
 - Processes are preemptive
 - It is guaranteed that a process always keeps its deadline if this is true in a period after a critical moment
 - The critical moment (or worst case) for a process is when at the beginning of its period all other processes with a higher priority also start their period
- **Another scheduling strategy**
 - Monotonic planning according to difference between deadline and arrival time
 - Could be used if the deadline is earlier than the beginning of the next period



Examples



<http://www.icsy.de>



Scheduling Extensions

- **Support of non periodic processes**
 - Add virtual LDUs to create a periodic behavior
 - Useful if a process is nearly periodic
 - Inefficient resource usage
 - Reserve a CPU budget for non periodic processes; the budget is refreshed periodically
- **Administrative mechanisms**
 - Early recognition of deadline violations
 - Easy to implement for EDF
 - Stop some low priority processes if deadlines are violated continuously
 - Measurement of thread parameters during run time
 - Requires a renew of the scheduling plan even for static strategies

Dispatcher (1)

- **Generation of dispatching tables**

- A dynamic scheduler determines only the next (few) table entries
- The scheduler may produce significant overhead
- A static scheduler determines one table that is valid for each period also called reservation table

- **Meta scheduler (priority scheduling)**

- Use the standard system scheduler and dispatcher
- Additionally use a meta scheduler to plan RT-Processes + wildcards for TS-Processes
- A meta dispatcher changes the priorities of the RT-Processes according to the meta scheduler
 - Starting a RT-Process: increase its priority to a high level
 - Stopping a RT-Process: decrease its priority to a low level
 - Start+Stop of a RT-Process requires a context switch to the meta dispatcher process first => causes significant overhead
 - TS-Processes may run on an average priority level and may be controlled by the standard scheduler

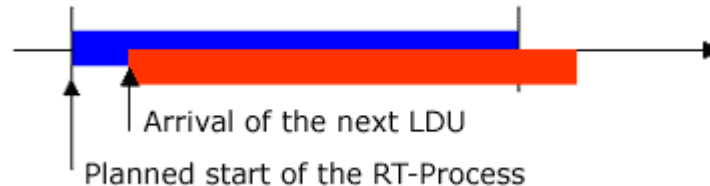
...	...
t_n	Process 330
t_{n+1}	Process 331, 332
t_{n+2}	A TS-Process
...	...

t_0	Process 330
t_1	Process 331, 332
t_2	Process 330
t_3	A TS-Process

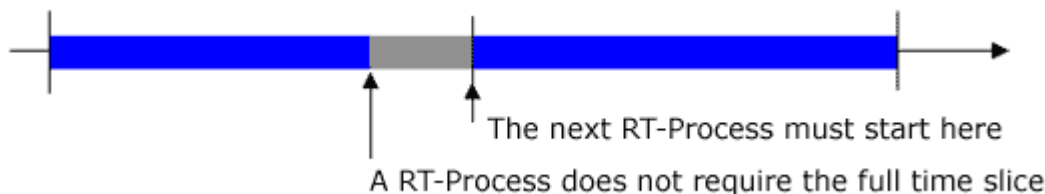
An arrow points from the 'A TS-Process' entry in the top table to the 'A TS-Process' entry in the bottom table.

Dispatcher (2)

- Problems with static dispatching tables** The processing time e must be known exactly
 - What happens if a RT-Process starts at t_n but the next LDU is available at $t_n + \varepsilon$?



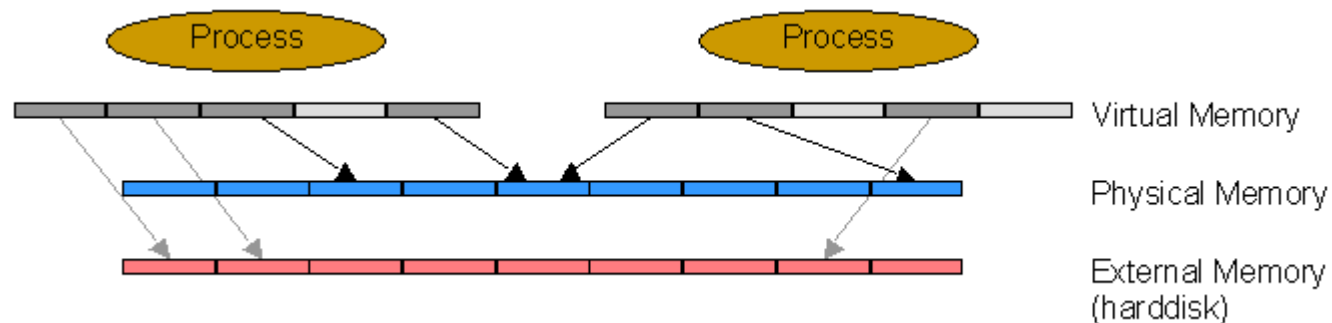
- additional buffering of LDUs may be necessary
- If a RT-Process does not require its full time slice and the next process is also a RT-Process, then the gap between those two processes can not be used efficiently



- LDUs can not be processed as soon as they are available, this leads to additional delay. Particularly when several RT-Processes are involved in handling one media stream.

Management of Memory access

- **Memory management principle**



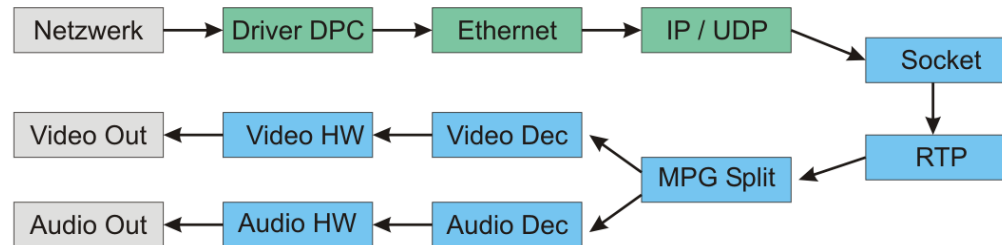
- The whole memory is divided into pages
- Each process has its own virtual address space
- Virtual addresses are mapped to physical addresses by Memory Management Unit (MMU)
- The physical memory is extended by external memory
 - Seldom used pages are written to harddisk in order to increase the available physical memory
 - If external memory is accessed then the MMU causes an interrupt and the system software has to load the page into physical memory

Aspects for Multimedia Systems

- **All media data and program code must be in physical memory**
 - Some systems allow direct access to physical memory
 - Usually pages can be locked to avoid copying to external devices
 - If too many pages are locked then the system performance is reduced significantly
- **Use shared memory to avoid copying of data**
 - Synchronization for memory access is required
 - Use double buffering (or multiple buffers) for reader/writer relations in order to support parallel processing of different LDUs
- **Avoid allocation of memory at run time**
 - Memory allocation is a complex task and may increase the processing time unnecessarily
 - The time need for memory allocation is (in general) unpredictable

Buffer Management

- Example for components involved in real time media processing



**Processing
Context**

Hardware

Kernel

Application

- **Buffer management techniques**

- Copying data
 - Each module has its own buffers
 - Data is copied at the interfaces of a module
- Offset management
 - There is one big buffer of shared memory with offsets pointing to segments for each module
 - The sum of the maximum required buffers of all modules must be known
- Distribution and collection
 - Each modul has its own buffers of shared memory
 - A table of pointers is exchanged at the interfaces (may be shared memory)
 - When sending data the last component has to collect the data, i.e. copy it into a single PDU



Management of Disk Access

- **Today single harddisks are fast enough to deliver few media streams in time**
- **A throughput of 100 Mbyte/s is (theoretically) sufficient to deliver 200 MPEG-2 videostreams of 4 Mbit/s**
- **The problem:**
 - Random access to different media streams prevents sequential reading of data
 - The access time becomes the bottleneck
- **The solution:**
 - The bandwidth of streamed media is much smaller than the throughput of the harddisk and there are no hard delay requirements when there is only a human to machine interaction
 - Buffer large data segments to realize a smooth streaming from memory only, this reduces the number of disk accesses

Increasing Harddisk Throughput (1)

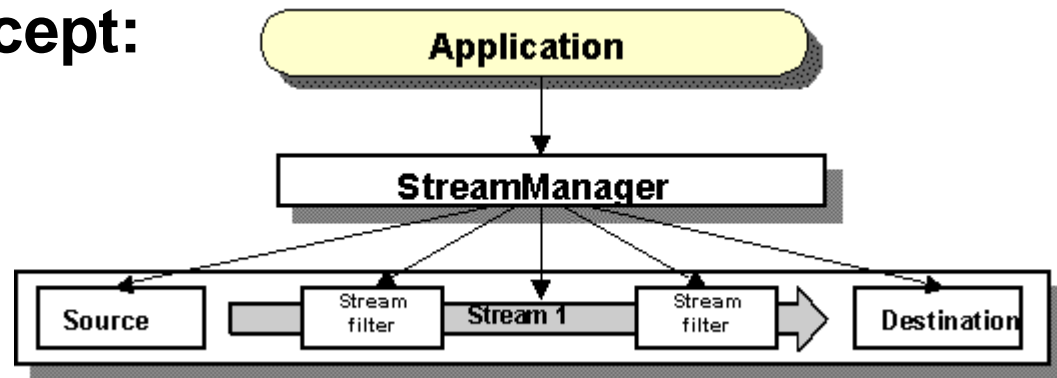
- **RAID = Redundant Arrays of Inexpensive Disks aims to increase throughput and/or reliability RAID-0:**
 - Data may be distributed over several harddisks
- **RAID-1:**
 - All data is written twice, i.e. there is a mirror for each harddisk
- **RAID-2:**
 - Hamming-Codes are written instead of complete mirrors
 - Higher efficiency than RAID-1 but more complex because of Hamming-Code calculation
- **RAID-3:**
 - The bits are distributed over several harddisks using only one additional harddisk for a parity bit
 - Since the harddisk controller is able to detect corrupted disks, the parity bit can replace any bit

Increasing Harddisk Throughput (2)

- **RAID-4:**
 - Calculates a parity block for so called stripes
 - A stripe consists of one or more stripe units, each unit is written to one disk, so that a stripe is distributed over several disks. Except for small data units which are smaller than a stripe unit.
 - The throughput of dedicated disks for parity data is the throughput bottleneck
- **RAID-5:**
 - Like RAID-4 but the parity blocks are also distributed over several disks
- **RAID-6:**
 - Like Raid-5 but a more sophisticated parity calculation (Reed-Solomon) can handle the drop out of two harddisks
- **Parallel access of several disks (RAID-3 to 6) improves the overall throughput**

6.3. Application Software

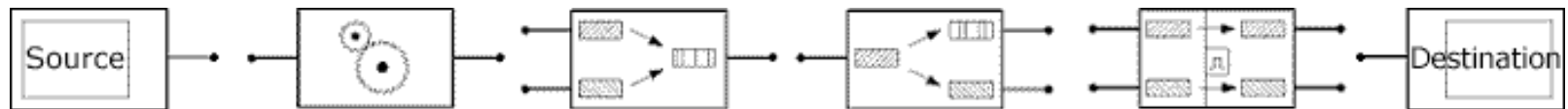
- **How to handle continuous media within applications? Requirements:**
 - Periodic processing of data
 - Control the presentation only
 - Use of optimized modules/libraries for media processing, e.g. coding/decoding
 - Use external components for media processing, which is controlled by the application only
- **Stream concept:**



Stream Concept

• Realization

- Special threads within the application context or even special processes
 - Using threads avoids inter process communication with the „main“ application (e.g. DirectShow from Microsoft)
- Using processes to apply real-time schedulers
- There are different types of stream-filters



• Advantages

- A well defined stream-filter interface enables adequate buffering strategies
- Simple reuse of software components
- Easy handling for applications

• Disadvantages

- If special data processing is required the application programmer must supply a new stream-filter