7. Operating System Support

7.1 Real-time Operation

## 7.2 Scheduling Algorithms

7.1   Real-time Operation

Management of processes in operating systems

Logical resources (i.e., processes) are mapped onto physical resources (e.g., central processing unit (CPU) and memory).

A process can have four states:

- **running**: it is allocated to a processor
- **ready**: it holds all operating resources except the processor
- **blocked**: it is waiting for the occurrence of an event (e.g., the termination of an output to disk)
- **inactive**: the process is not assigned to an application program.
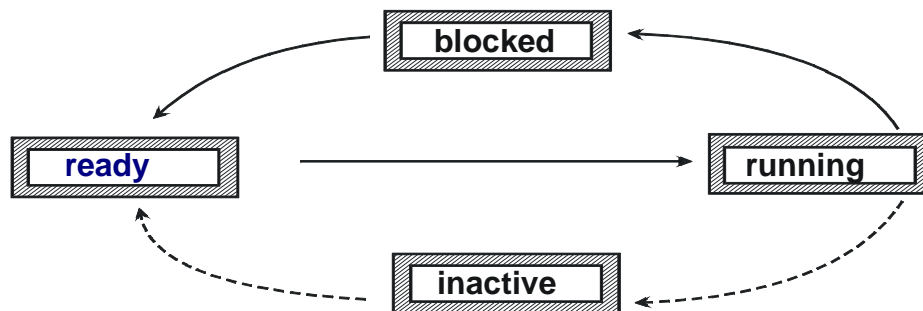
Scheduler and Dispatcher

The **scheduler** decides which of the *inactive* processes will be moved to the *ready* state.

The **dispatcher** decides which process to move next from state *ready* to state *running*.

In conventional operating systems, **neither** scheduler **nor** dispatcher **are capable of real-time processing.**

Requirements for Multimedia

Processing of continuous data streams

The data to be processed appears in **periodic** time-intervals.

Typical operations on multimedia data are:
- Creation and playout of audio and video packets,
- Transmission of audio and video packets,
- Compression and decompression of audio and video packets.

Real-time requirements:
- Processing must be completed by a specific time (**deadline**)
- Processing of a multimedia data packet takes approximately the same amount of resources in each time period.

Light-weight Processes (threads)

Concept

- Concurrent (parallel) activities are carried out **within one address space.**
- Each concurrent activity is called a **thread.**
- Data transfer between threads is efficient (just a copy operation).
- A "context switch" (i.e., a switch between different threads) is much more efficient than a switch between normal, heavy-weight operating system processes.

But:

There is **no access protection** between threads provided by the operating system (same address space for all threads)!

Reservation of Resources

Pessimistic (deterministic) reservation

- Takes the worst case into consideration
- Reserves too many resources most of the time and thus leads to sub-optimal use of the resources

Optimistic (stochastic) reservation

- Reservation is made for the expected value (mean value) of the processing resources
- Leads to a good utilization of the resources
- But can lead to an overload of the resources (blocking or sub-optimal execution of a process)
- Role of a run-time monitor:
  - Monitors resource usage
  - In case of overload initiates suitable action, such as scaling (QoS adaptation) or blocking of the process and withdrawal of the resource.

Real-time Process Specification for Multimedia

Traditional systems usually have

- *either* a scheduling mechanism for time-sharing applications
- *or* a scheduling mechanism for real-time applications

In multimedia systems, a scheduler for both types of applications is required.

real-time applications	time-sharing applications (best effort)
scheduler and dispatcher	
CPU	
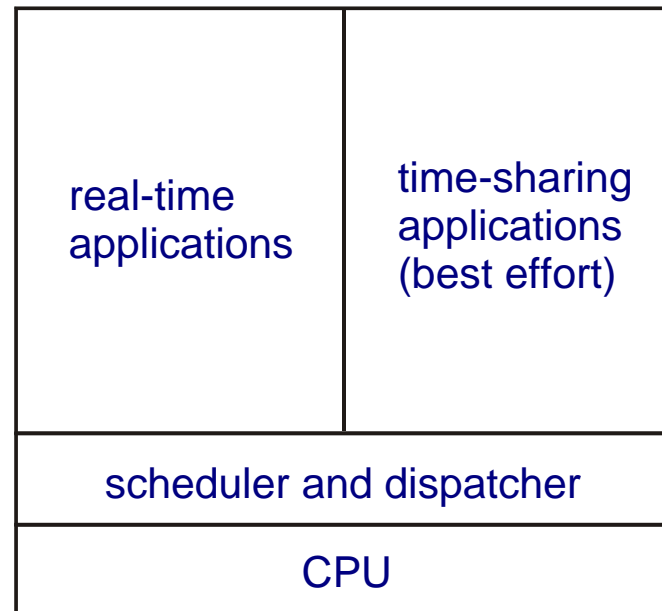
Real-time Applications vs. Multimedia Applications

Data in traditional real-time applications (e.g., in process automation and control) typically have **hard** real-time requirements.

Multimedia data is generally intended for presentation to the human being as the data sink. This means:

- **Rare violations of deadlines are acceptable.** For example, a macro block decoding failure in an MPEG video that is caused by late arrival of the data packet can be tolerated; the decoder repeats the corresponding macro block of the previous frame.

- In traditional real-time systems, operations are not always periodic. Multimedia data is periodic. Scheduling and dispatching algorithms can take advantage of the periodicity. **Resource requirements are easier to predict.**

Process Scheduling Goals

The main goal of process scheduling and dispatching is to ensure that **all** deadlines of **all** processes are satisfied. Additional optimization goals are:
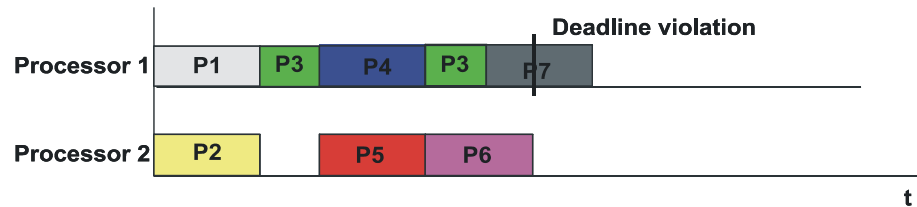
- A high average utilization of the resources, leading to high throughput
- fast computation of the resource allocation (an efficient algorithm).

The allocation does not necessarily have to be **optimal**. Complexity theory tells us that the computation of optimal schedules is NP-complete.

Scheduling Problem: An Example

Deadline

P1
P2
P3
P4
P5
P6
P7

t
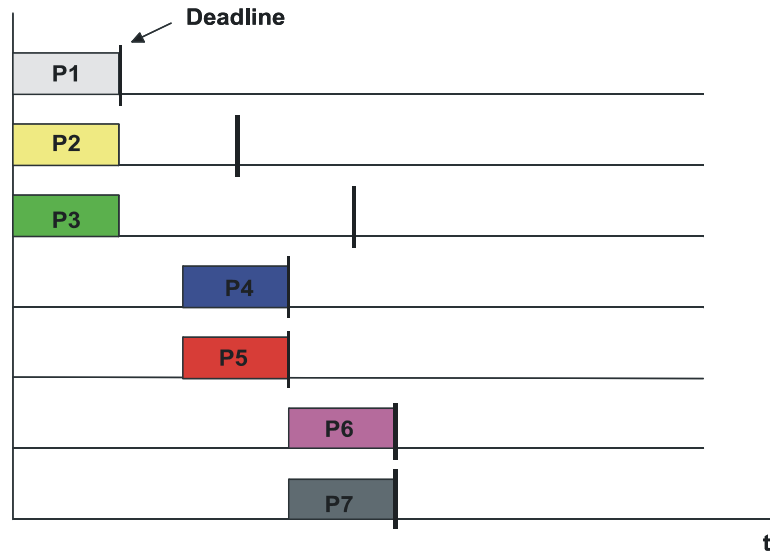
Allocation of the processor according to laxity, with "preemption"

Deadline violation

Processor 1 | P1 | P3 | P4 | P3 | P7

Processor 2 | P2 | P5 | P6

t

Improved allocation of the processor (non-algorithmic)

Processor 1 | P1 | P2 | P4 | P6

Processor 2 | P2 | P3 | P5 | P7

t

7.2 Scheduling Algorithms

Requirements

- All deadlines must be met.

- Resource utilization should be high.

- Best-effort requests should not starve.

- The algorithm should be efficient, i.e., not use too much CPU time itself.

The scheduler can take advantage of the **periodicity** of continuous data streams.

Preemptive vs. Non-preemptive Scheduling
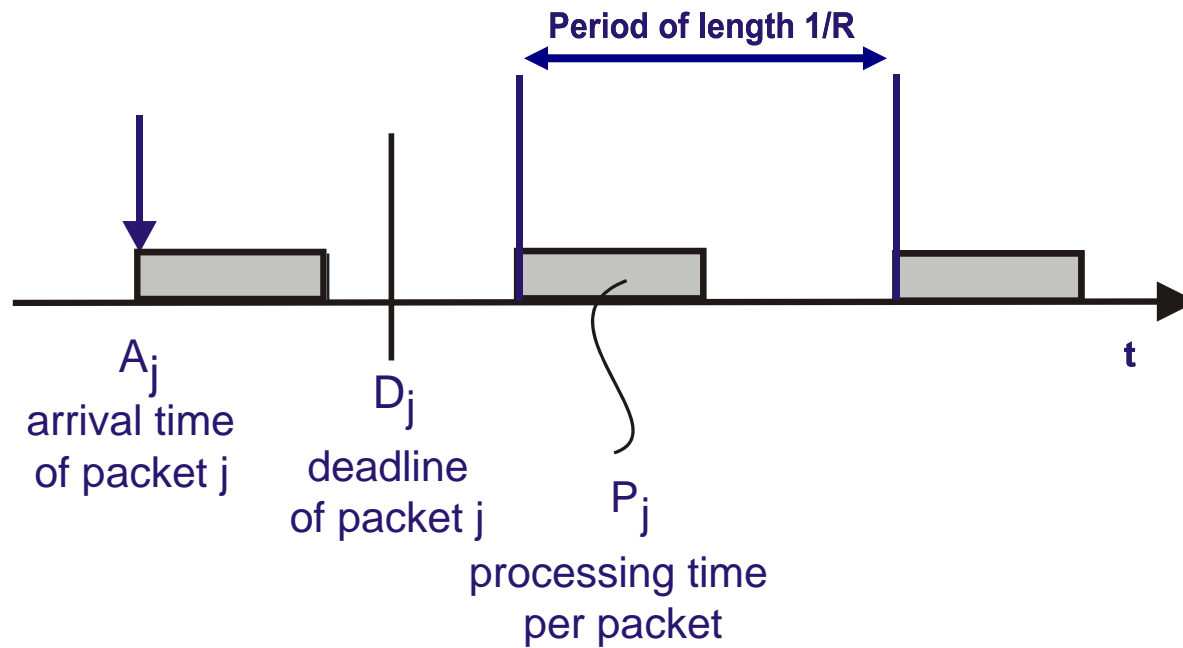
Preemptive scheduling

- If a process with higher priority requests a resource, the currently active (running) process is moved to the READY state, the high-priority process gets the CPU, i.e., is moved to the RUNNING state.
- There are many more process switches, process management overhead is high.

Non-preemptive scheduling

- Active processes are not interrupted by high-priority requests.
- Process switching occurs less frequently, overhead is lower.

Non-preemptive scheduling generally performs well for processes with short execution times.

Model of a Periodic Data Stream

Period of length 1/R

A_j
arrival time
of packet j

D_j
deadline
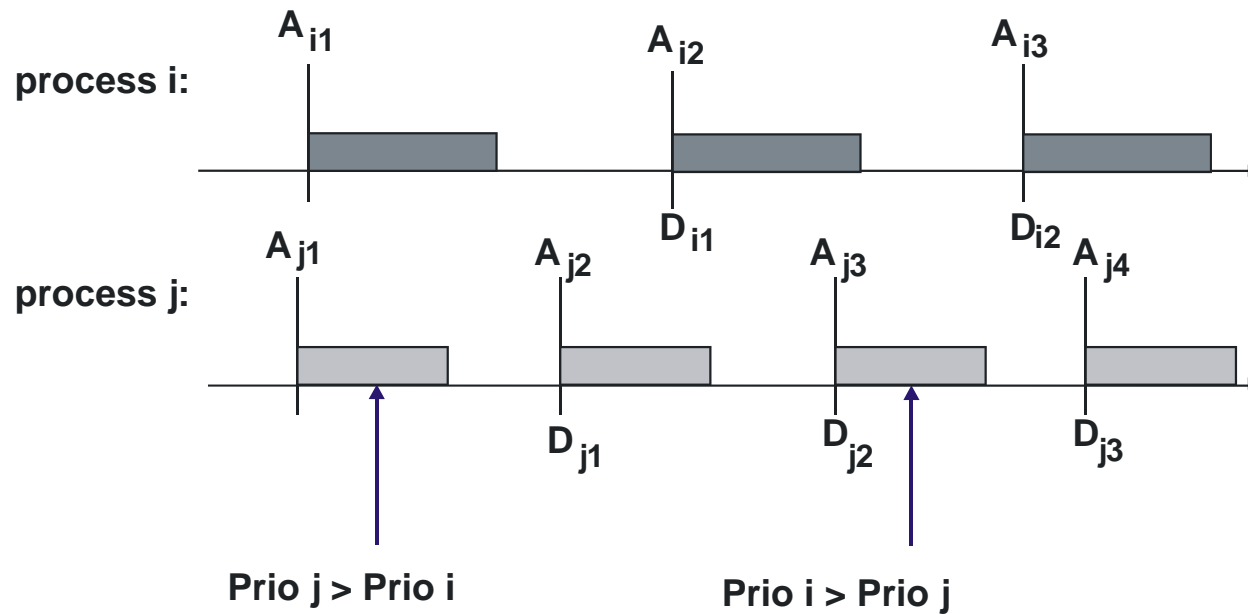of packet j

P_j
processing time
per packet

- R: Arrival rate
- P_j: Processing time
- D_j: Deadline for the termination of the process

These three parameters control the scheduling algorithm.

Algorithm 1: Earliest Deadline First (EDF)

The process with the **earliest deadline** is assigned the highest priority.

process i:

A_{i1} A_{i2} A_{i3}

D_{i1} D_{i2}

process j:

A_{j1} A_{j2} A_{j3} A_{j4}

D_{j1} D_{j2} D_{j3}

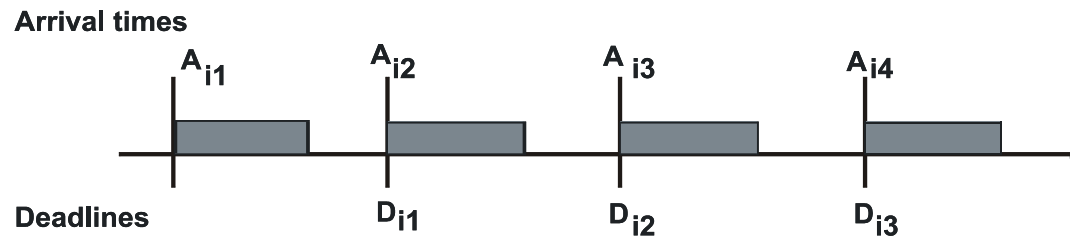**Prio j > Prio i**   **Prio i > Prio j**

Process priorities vary over time.

One can show that the EDF algorithm always finds a valid schedule if there is one.

Resources can be used up to 100%.

EDF Scheduling

In most cases of **periodic** scheduling, the deadline of a request is identical to the end of the period since the data buffer is needed again.

Arrival times

A_{i1} A_{i2} A_{i3} A_{i4}

Deadlines D_{i1} D_{i2} D_{i3}

Performance

Preemptive scheduling (Liu /Layland, 1973):

- maximum possible throughput:
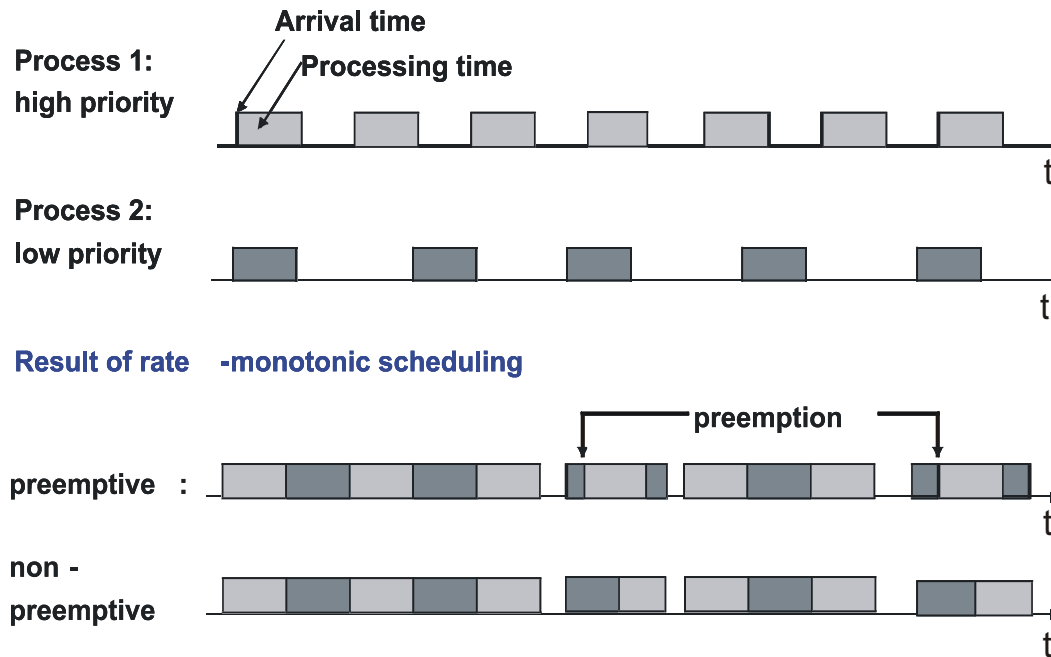
$$\sum_{all\ streams\ i} R_i P_i \leq 1$$

- delay of a packet <= $1/R_i$

Non-preemptive scheduling (Nagarajan/Vogt, 1992)

- same throughput as above
- delay of a packet <= $1/R_i + P_i$

Algorithm 2: Rate-Monotonic Scheduling (RM)

The process with the **highest packet rate** is assigned the highest priority.

Arrival time

Processing time

Process 1:
high priority

t

Process 2:
low priority

t

Result of rate   -monotonic scheduling

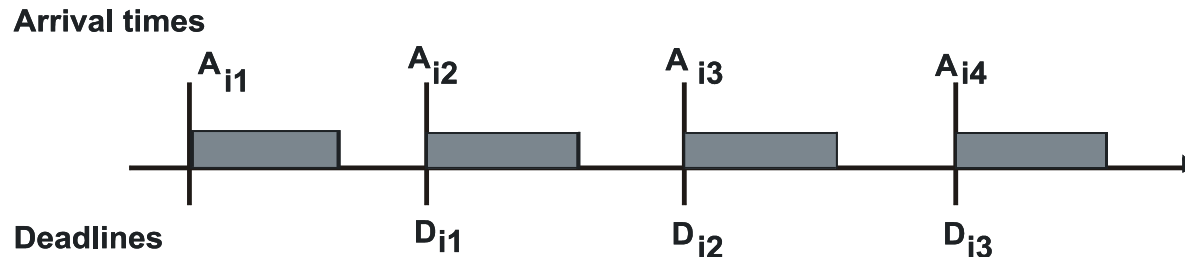preemption

preemptive   :

t

non -
preemptive

t

While streams are running priorities do not change; only when a new stream starts or when a stream ends, priorities are re-computed.

RM is an algorithm for **periodic** processes only.

Rate-Monotonic Scheduling

Again, we set the deadline to the end of the period:

Arrival times

Deadlines

Performance

Preemptive scheduling (Liu/Layland, 1973):

- maximum possible throughput:

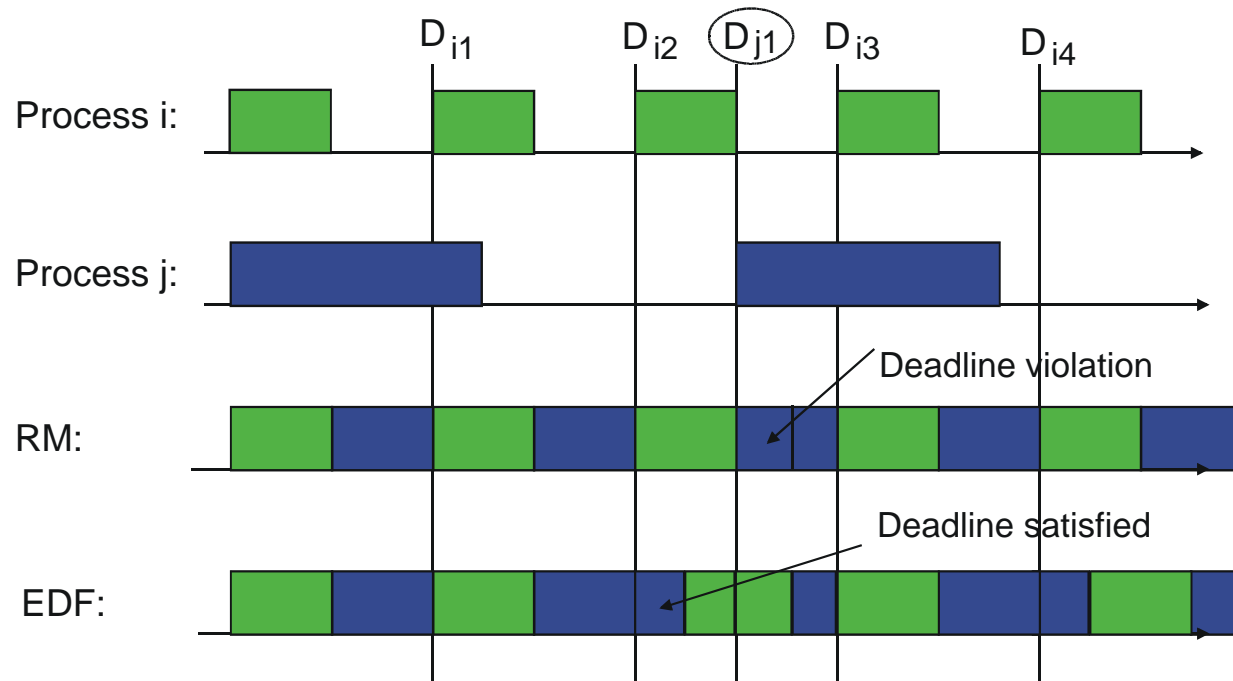$$\sum_{all\ streams\ i} R_i P_i \leq \ln 2$$

- delay of a packet <= $1/R_i$

Non-preemptive scheduling (Nagarajan/Vogt, 1992):

- highly sophisticated computation
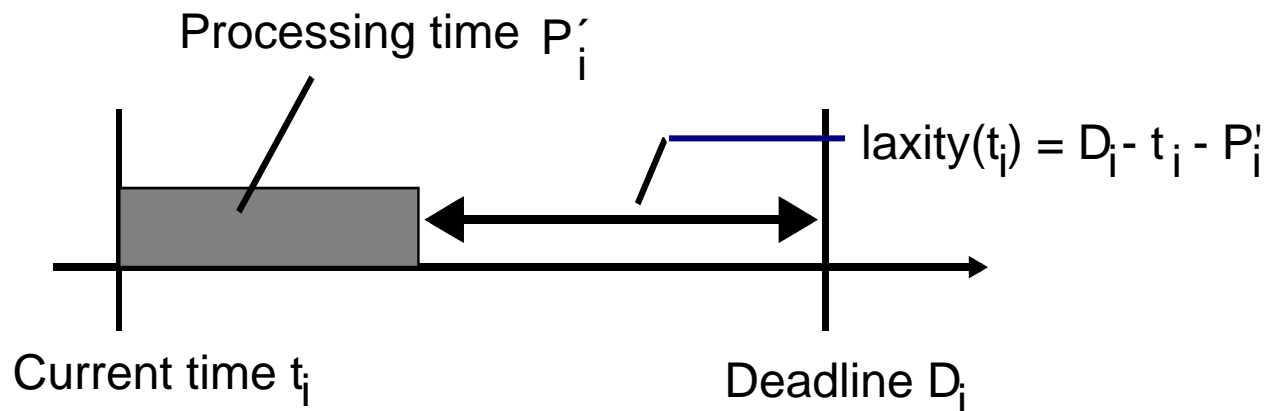- guaranteed throughput is much less

Examples for RM and EDF

With preemption

Other Scheduling Algorithms

Scheduling according to laxity

Processing time P_i'

$$\text{laxity}(t_i) = D_i - t_i - P_i'$$

Current time t_i Deadline D_i

- **Laxity:** maximum allowed waiting time until processing begins
- The process with the smallest laxity is assigned the highest priority.
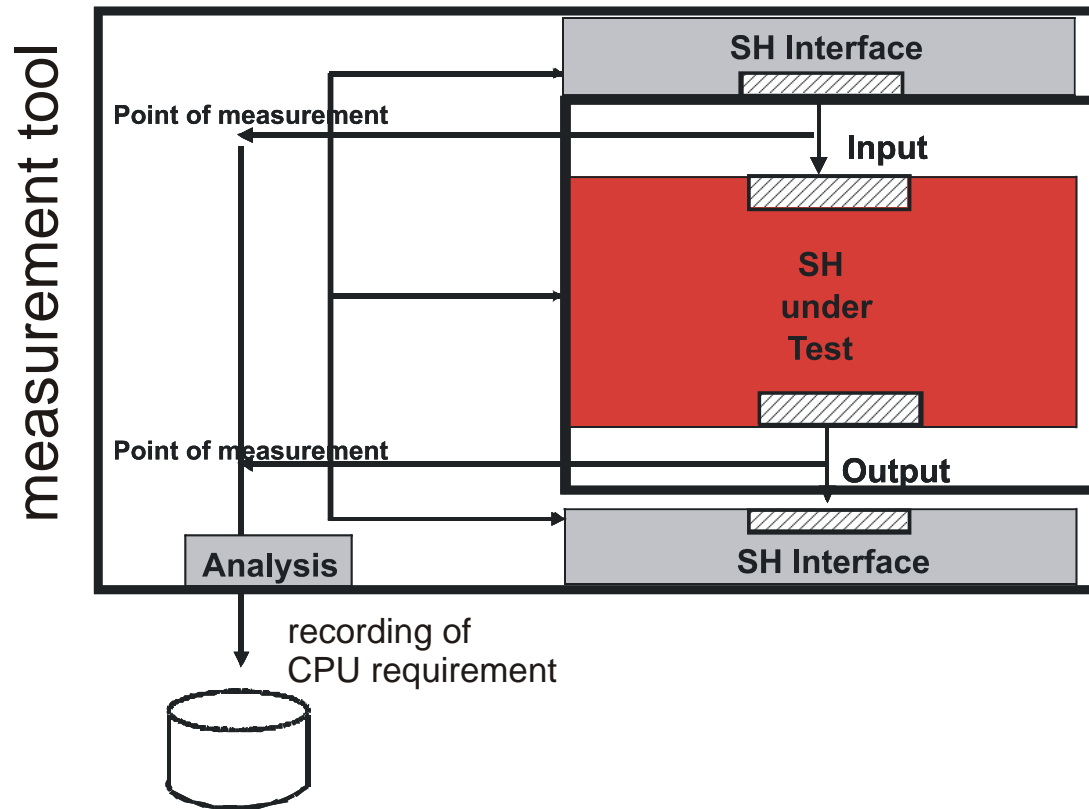- Priorities must be updated continuously (large computational overhead).

Determination of Processing Times

Problem

- For most scheduling algorithms, the actual processing time per packet on the current CPU must be known!

- An analytic computation is hardly possible.

A pragmatic approach: measurement and standardization.

Measurement of Processing Time

measurement tool

SH Interface

Point of measurement

Input

SH under Test

Point of measurement

Output

Analysis

SH Interface

recording of
CPU requirement

SH = Stream Handler