Metode şi Algoritmi de Planificare (MAP)

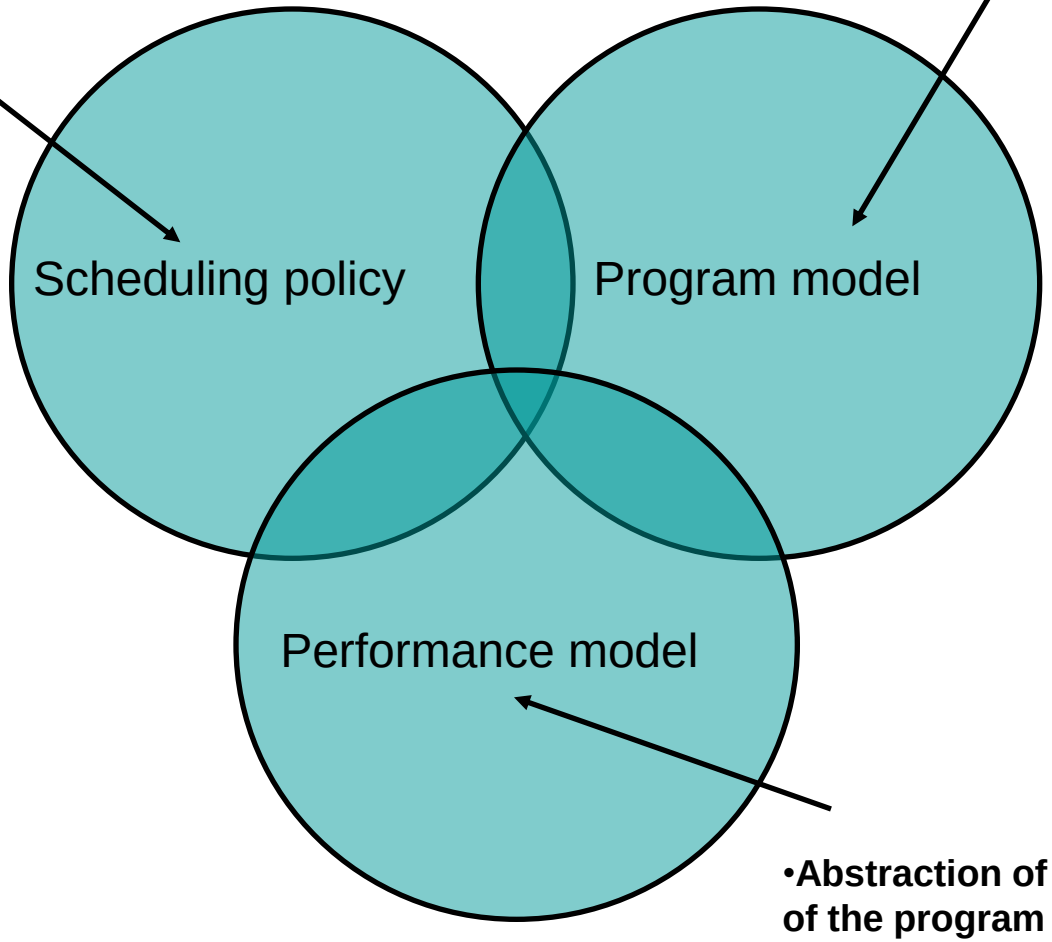
2009-2010

Curs 4

Politici de planificare

Scheduling model

- **Sets of rules used to produce schedules**
  - Description of the performance activity to be optimised by the performance model
- **Abstraction of the programs to be scheduled**

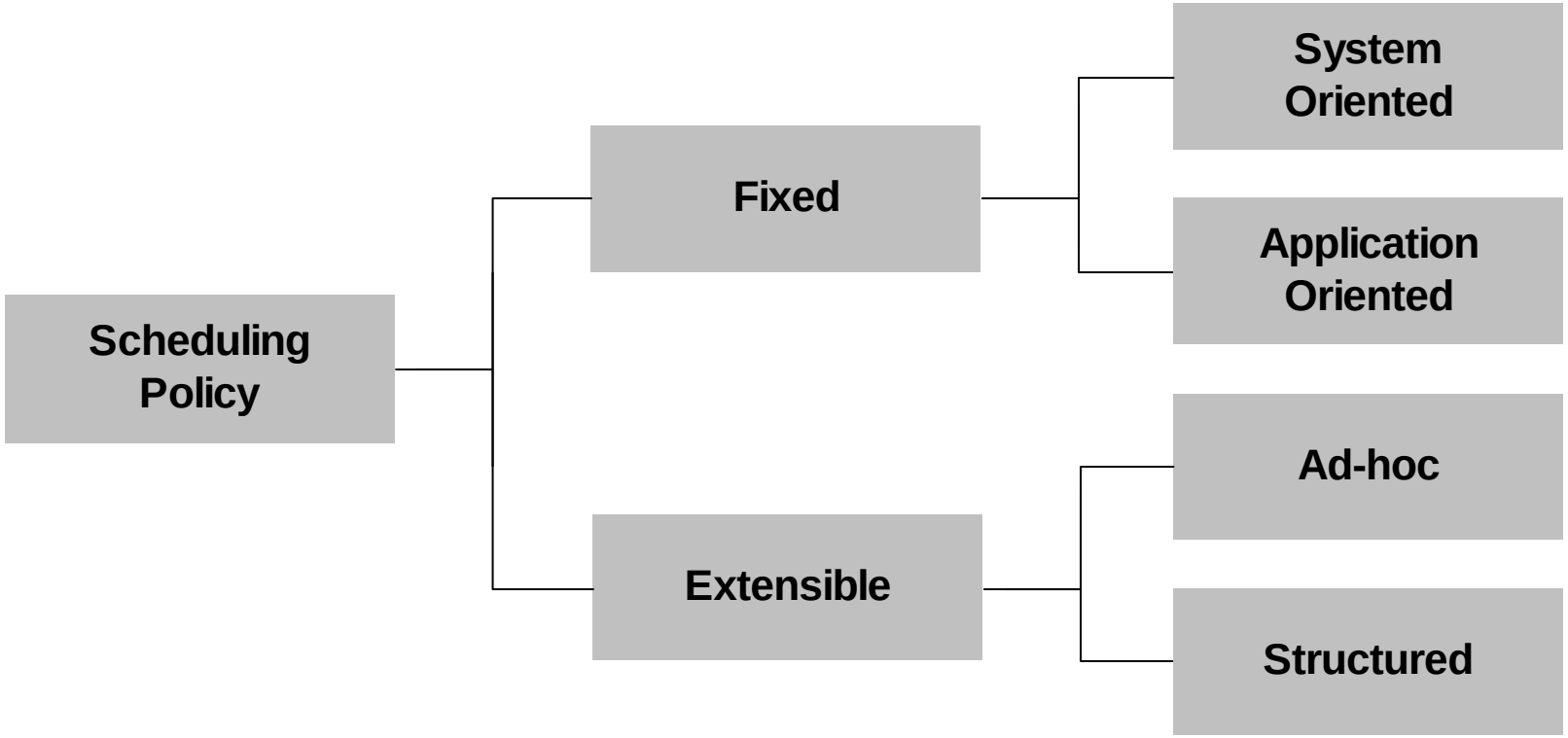


- **Abstraction of the behaviour of the program in the system**

Scheduling - Policies

- Issues
  - Fairness – don’t starve process
  - Priorities – most important first
  - Deadlines – task X must be done by time *t*
  - Optimization – e.g. throughput, response time
- Reality - No universal scheduling policy
  - Many models
  - Determine what to optimize - metrics
  - Select an appropriate one and adjust based on experience

Scheduling Policy



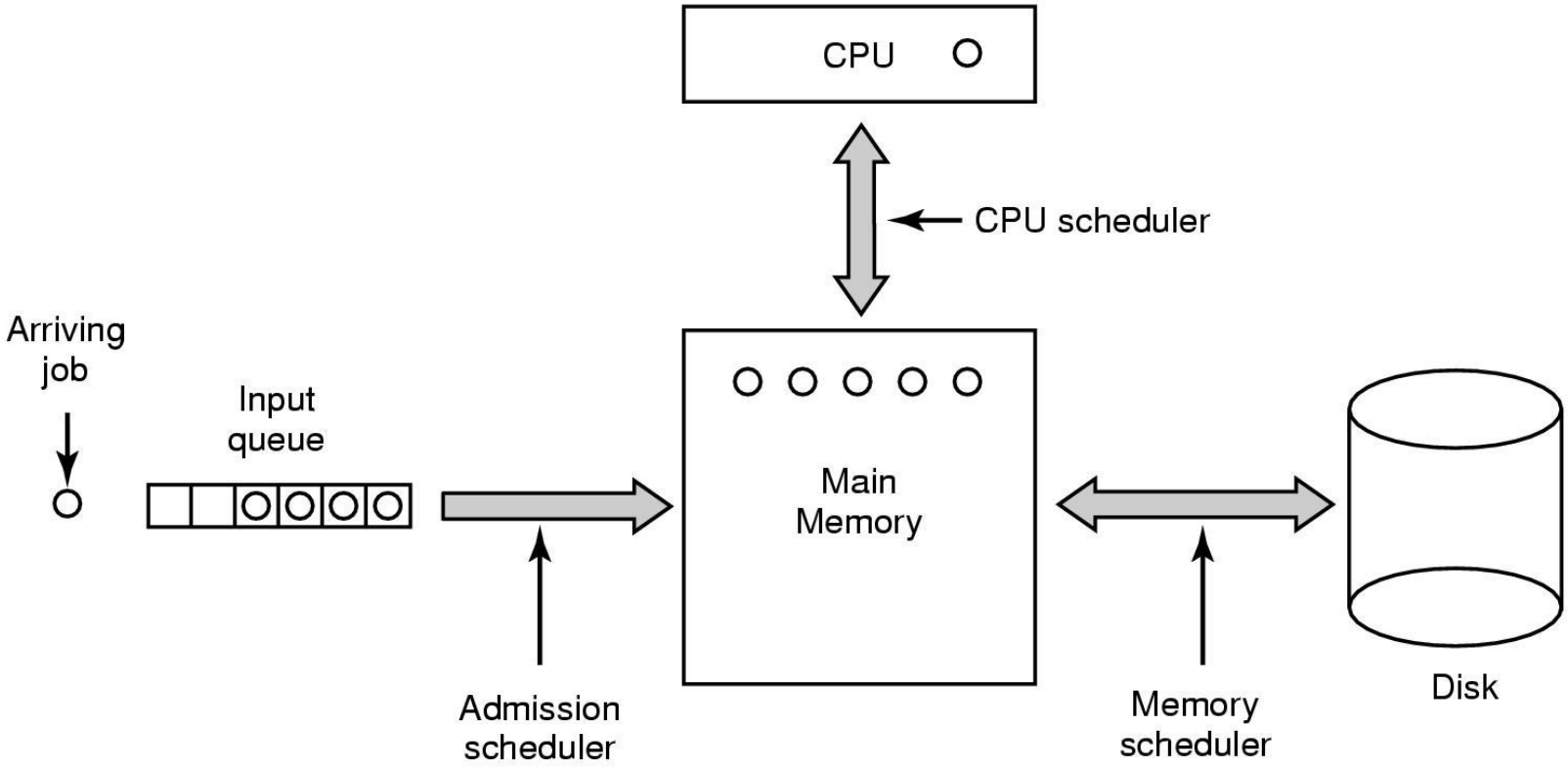
Scheduling Policy

- **Application-centric**
  - **Execution Time** : the time duration spent executing the job
  - **Wait Time** : the time duration spent waiting in the ready queue
  - **Speedup** : the ratio of time spent executing the job on the original platform to time spent executing the job on the Grid.
  - **Turnaround Time** : also called response time. It is defined as the sum of waiting time and executing time.
  - **Job Slowdown** : it is defined as the ratio of the response time of a job to its actual run time.

Scheduling Policy

- **System-centric**
  - **Throughput** : the number of jobs completed in one unit of time, such as per hour or per day.
  - **Utilization** : the percent of time a resource is busy.
  - **Flow Time** : the flow time of a set of jobs is the sum of completion time of all jobs.
  - **Average Application performance**

Batch System Scheduling



Note: Number of Processes in Memory determines the degree of multiprogramming

Scheduling Policies - LIFO

- **Last-In First-out (LIFO)**
  - Newly arrived jobs are placed at head of ready queue
  - Improves response time for newly created threads
- **Problem:**
  - May lead to starvation – early processes may never get CPU

Scheduling Policies - FCFS

- First Come, First Served (FCFS)
  - Easy to implement
  - Non-preemptive
  - Minimizes context switch overhead
  - Favors **compute bound** jobs
  - Short jobs penalized

Scheduling Policies – Round Robin

- Round Robin (RR)
  - Preemptive
  - Ready processes given a **quantum** of time when scheduled
  - Process runs until it blocks or quantum expires
  - Suitable for interactive (timesharing) systems
  - Setting quantum is critical for efficiency

Scheduling Policies - Comparison

- 10 jobs each take 100 seconds - look at when jobs complete
- FCFS – job 1: 100s, job 2: 200s … job 10:1000s
- RR
  - 1 sec quantum
  - Job 1: 991s, job 2 : 992s …
- RR good for short jobs – worse for long jobs

Scheduling Policies – RR example

- Utilization
  - A and B compute bound jobs - 100 ms @ 100% CPU
  - C – 1 ms CPU and 10 ms disk I/O
  - Quantum – 100 ms
    - CPU utilization = (100 + 100 +1)/210 = 95.7 %
  - Quantum – 1 ms (assume 0 overhead)
    - ABCAB*(C-I/O)*ABABAB ….AB
    - CPU utilization = 100%
  - Smaller quantum can improve utilization – but consider overhead

RR: Choice of Time Quantum

- Performance depends on length of the timeslice
  - Context switching isn't a free operation.
  - If timeslice time is set too high
    - attempting to amortize context switch cost, you get FCFS.
    - i.e. processes will finish or block before their slice is up anyway
  - If it's set too low
    - you're spending all of your time context switching between threads.
  - Timeslice frequently set to ~100 milliseconds
  - Context switches typically cost < 1 millisecond

Moral:

Context switch is usually negligible (< 1% per timeslice) unless you context switch too frequently and lose all productivity.

Scheduling Policies - STCF

- STCF – shortest time to completion first (or shortest job first)
  - Can be preemptive
  - Good for throughput
  - Example
    - jobs A – 100, B – 1, C – 2
    - Done B – 1, C – 3 , A – 103
    - Ave = 35, (for RR (1) about 37)
  - Choose the job with the shortest next CPU burst
  - Provably optimal for minimizing average waiting time
- **Problem:**
  - Impossible to know the length of the next CPU burst

Shortest Job First Prediction

- Approximate next CPU-burst duration
  - from the durations of the previous bursts
    - The past can be a good predictor of the future
- No need to remember entire past history
- Use exponential average:
  - t_n duration of the n^{th} CPU burst
  - τ_{n+1} predicted duration of the $(n+1)^{st}$ CPU burst

$$\tau_{n+1} = \alpha\, t_n + (1- \alpha)\, \tau_n$$

where $0 \leq \alpha \leq 1$. α determines the weight placed on past behavior

Scheduling Policies - Priority

- Priority Scheduling
  - Preemptive
  - Process are given priorities and ranked
    - Maybe done with multiple queues - multilevel
  - Highest priority runs next
    - With multilevel queues
      - Select from highest queue
      - Round robin within queue
  - Recalculate priority – many algorithms
    - E.g. increase priority of I/O intensive jobs
    - E.g. favor processes in memory
    - Must still meet system goals – e.g. response time

Scheduling Policy - Problems

- Priority inversion
  - A has high priority, B has lower priority and B acquires a resources that A needs to progress
  - A attempts to get resources, fails and busy waits; B never runs
  - A attempts to get resources, fails and blocks; C (medium) enters system; B never runs
- Priority scheduling can’t be naive

Scheduling Policies - Realtime

- Real Time System – processes have deadlines
  - Deadlines known
  - (usually) preemptive
- Static algorithm – periodic process behavior
  - Rate Monotonic Scheduling (RMS) – priority = 1/period
- Dynamic – aperiodic
  - Earliest Deadline First (EDF) – select process that must complete the soonest

Scheduling – Policies Examples

- Unix – multilevel - many policies and many policy changes over time
- Linux – multilevel – 3 major levels
  - Realtime FIFO
  - Realtime round robin
  - Timesharing
- Win/NT - multilevel
  - Threads scheduled – **fibers** not visible to scheduler
  - Jobs – groups of processes are given quotas that contribute to priorities

Disk scheduling policies

- Each policy assumes
  - a queue of waiting disk requests exists
  - disk requests are entered the queue in random order
- Policies we will consider are:

random	FIFO
PRI	SSTF
SCAN	C-SCAN
N-Step-SCAN	FSCAN

Disk scheduling policies

- Random – Just a benchmark for comparison
- FIFO
  - Next disk request has been in queue the longest
  - Same as random if disk requests are queued randomly (true for many processes)
  - Fair to all processes
- PRI
  - PRIority given to requests, based on process class (interactive, batch, etc.)
  - Scheduling largely outside of disk management control
  - Goal is not to optimize disk use but to meet other objectives
  - Short batch jobs may have higher priority
  - This provides good interactive response time

Disk scheduling policies

- SSTF (Shortest Service Time First)
  - From requests currently in the queue, choose the request that minimizes movement of the arm (read/write head)
  - Always chooses minimum seek time
  - Resolves ties in a fair manner

    (both inward and outward)
  - Doesn’t guarantee minimum total arm movement
  - Starvation possible

Disk scheduling policies

- SCAN
  - Arm moves in one direction only until it reaches the last request in that direction
  - Then the arm reverses and repeats
  - Avoids starvation
- C-SCAN
  - Like SCAN, but in one direction only
  - Then returns arm to the opposite side and repeats
  - Reduces maximum wait in the queue near the disk edge

Disk scheduling policies

- N-step-SCAN
  - Divide queue into N-request segments
  - Use SCAN on each
  - New requests are added to the rear of the queue to form a new N-request segment
  - Reduces maximum waiting time in a high-volume situation
  - Causes head to move more frequently from one cylinder to the next
- FSCAN
  - Like N-step-SCAN but with two queues
  - One queue fills while the other is processed using SCAN

Queuing Systems

- Queues have different limits on the resource requests
  - Number of resources requested
  - Execution time
  - Interactive/Batch jobs
- Jobs are sorted by schedule policy in the queue
  - The highest priority request is the queue head
- If more than one queue can be started, further criteria are needed, such as ***Queue priority***
- If no queue head can be started, the idle resources may be utilized with ***backfilling***

Planning Systems - Replanning

- Requested
  - Start time
  - Estimated run time
- When
  - A new request is submitted
  - A running request ends before it’s estimated end time
- How
  - Delete all non-reservations from schedule
  - Sort non-reservations according to schedule policy
  - Arrange reservations into schedule
  - Insert non-reservations in the schedule at the earliest possible start time

Queuing vs. Planning Systems

	Queuing	Planning
Planning time frame	Present	Present and Future
Submission of resource requests	Insert in queue	Replanning
Assignment of proposed start time	No	All requested
Runtime estimates	Not necessary	Yes
Reservation	Not possible	Yes
Backfilling	Option	Yes

Advanced Planning Functions

- Requesting Resources
- Dynamic Aspects
- Service Level Agreements

Requesting Resources

- Diffuse requests
  - Give a range: “need 32~128 CPUs”
  - Let RMS optimizes: “need as much nodes as possible”
- Negotiation

Dynamic Aspects

- Variable Reservations
  - Make a reservation ASAP
  - Different from reserved jobs:
    - No fix start time
  - Different from non-reserved jobs:
    - Never planed later than its first planned start time
- Resource Reclaiming
  - Replace requested resources at run time
- Automatic Duration Extension
  - Extend the runtime of jobs while they are running
  - How long can it be extended
  - Hoe many time it can be extended

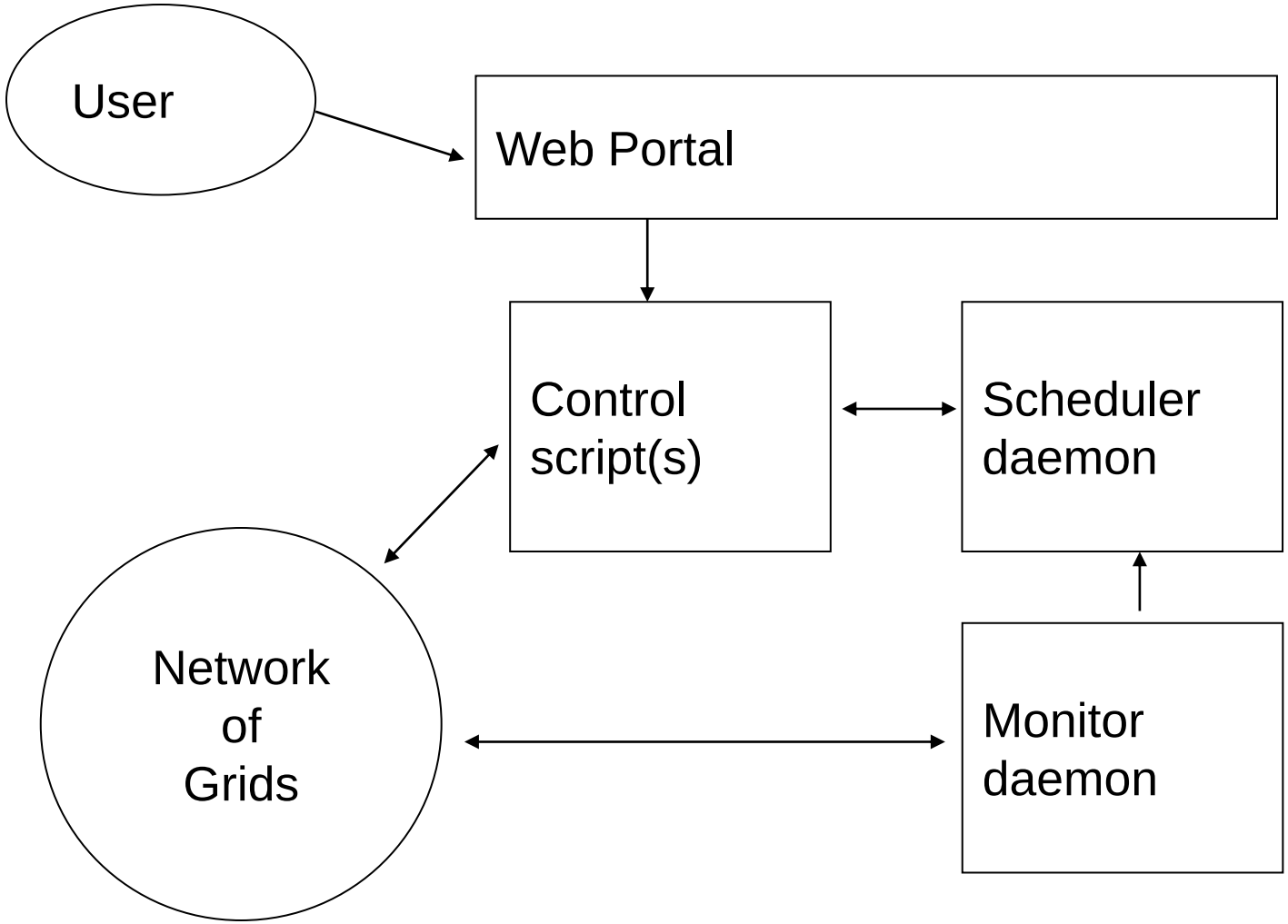
Dynamic Aspects (Cont.)

- Automatic Restart
  - It can utilize short time slots in the scheduling
- Space sharing “Cycle Stealing”
  - Run as a background job to steal resources in a space sharing system (like condor)
- Deployment Servers
  - RMS plans both the requested resources and the time to reconfigure the hardware

Service Level Agreements (SLA)

- SLA has to be considered not only in the scheduling process but also during the runtime
- At runtime the scheduler is not responsible for measuring the fulfillment of the SLA, but to provide all granted resources

Policy-based meta-scheduler implementation



Trends & challenges

- Trends
  - Use of increasingly dynamic information
  - Use of meta-information
  - Scheduling of more real-world programs
  - Restrictions on the program domain
  - Deriving scheduling information from programming language(s)

- Challenges
  - Portability vs. performance
  - Grid-aware programming
  - Scalability, Efficiency, Repeatability
  - Meta-scheduling

Exam's quizzes

- **1.** Ce este o politică de planificare? Care sunt principalele clase de politici de planificare.
- **2.** Descrieţi pe scurt politica Round Robin. Discutaţi avantajele şi dezavantajele acestei politici.
- **3.** Care sunt principalele politici pentru planificarea accesului la disk?
- **4.** Ce înţelegeţi prin politici de meta-planificare?