Proportional Share

General model and practical implementations

Paolo Valente
ReTiS Lab
SSSUP S. Anna

Subject

- Survey of scheduling algorithms compliant with the *proportional share* paradigm
  - Definition of the proportional share paradigm in a general system model
  - Definition of two proportional share schedulers inside the general system model: WF²Q and WF²Q+
  - Practical implementations for process, network and disk scheduling

Resource sharing 1/2

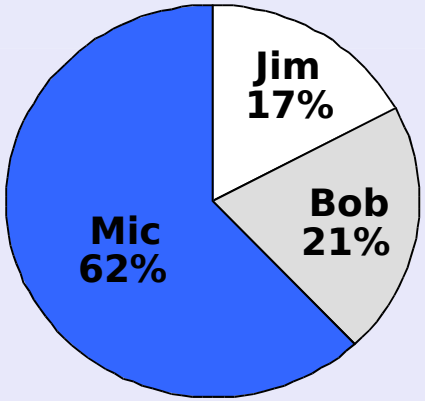
- System: a shared resource providing a certain service to a set of *N* activities ...

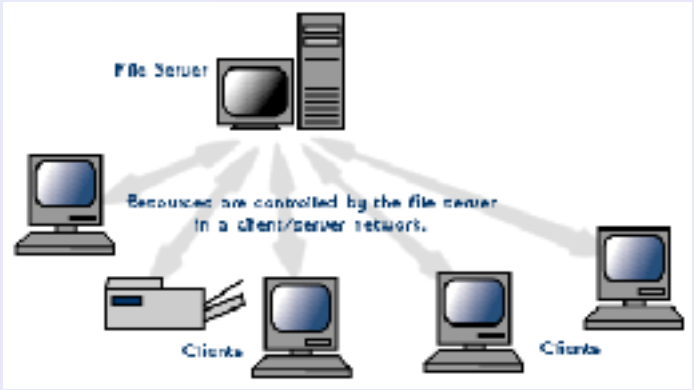
Process scheduling

- Shared Resource: Processor

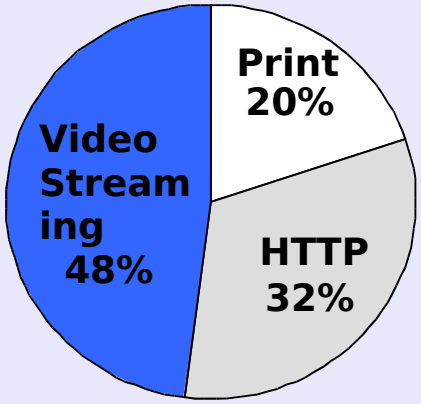
Example 1: Network server

- Per-user CPU-time shares





- Per-service CPU-time shares

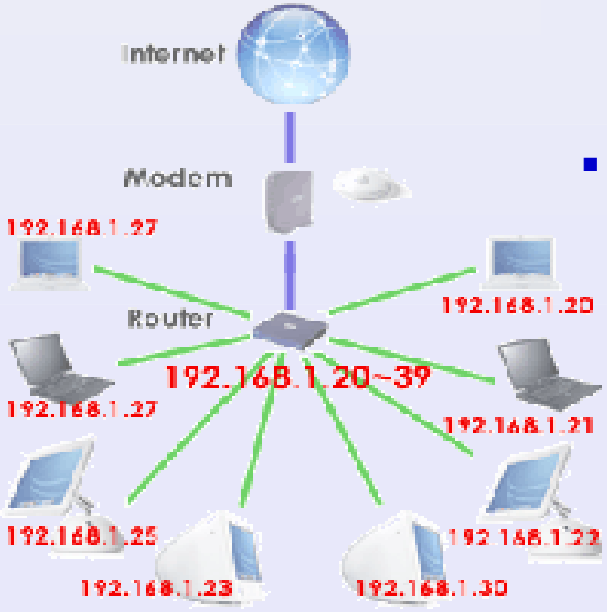


Example 2: Quality of Service guarantees for a mixed application set, on a general purpose system

- E.g.: Multimedia + Batch applications + File transfers

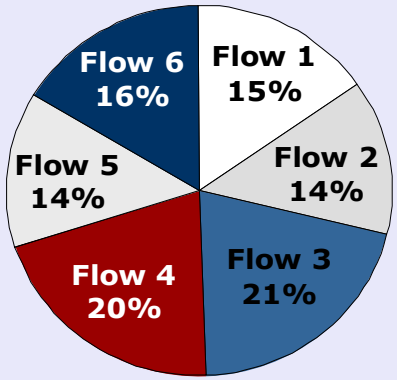
Packet scheduling

- Shared Resource: Transmission Link



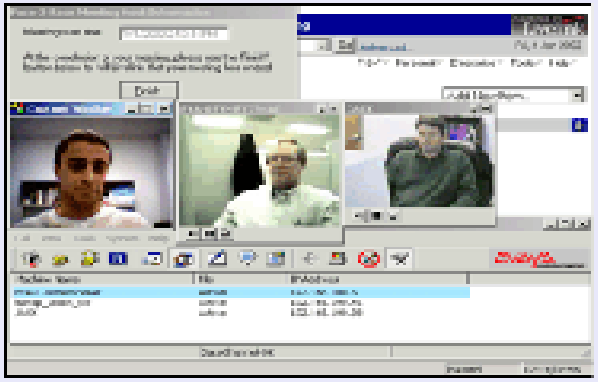
Example 1: Network router

- Per-flow shared-link capacity shares



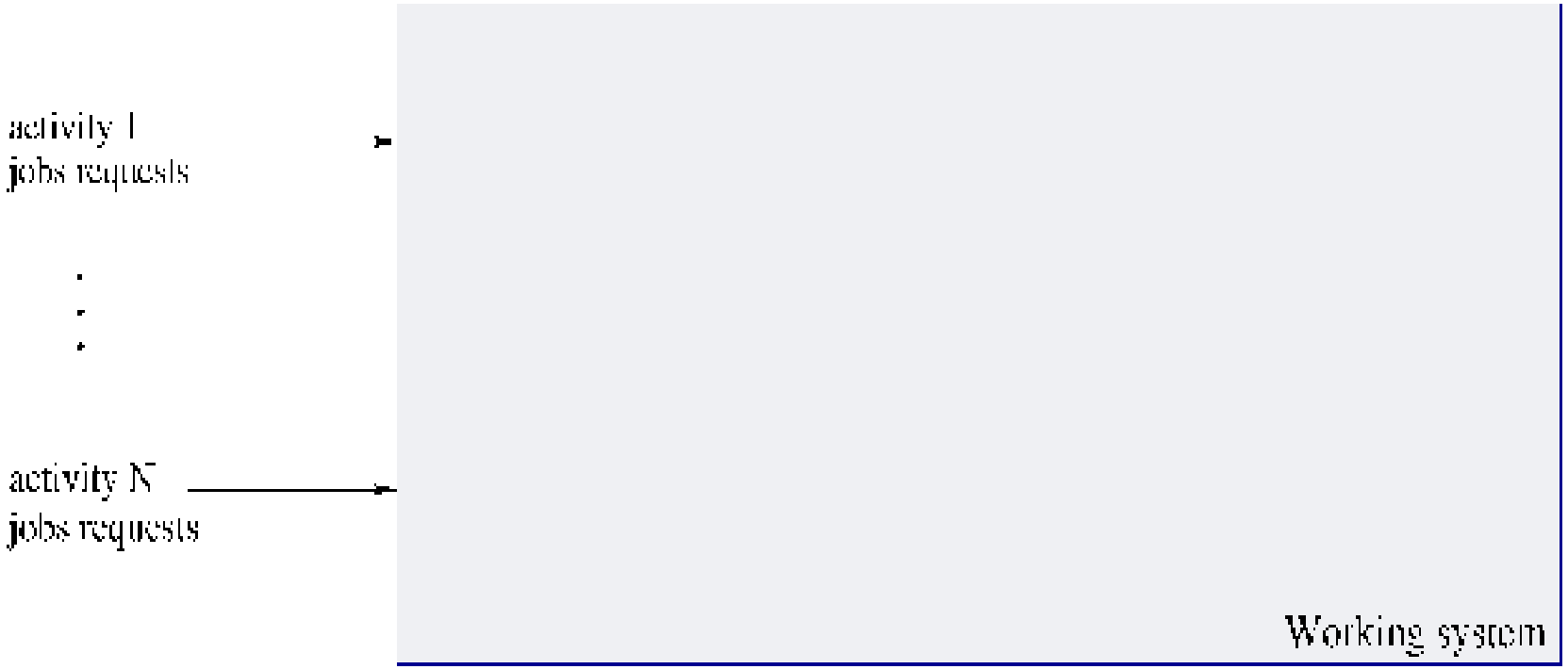
Example 2: Videoconferencing

- Quality of Service guarantees along the network paths between the participants



The working system

A set of *N* activities need the service – *work* – provided by a *working system*. As time passes, activities request work to the system in terms of *jobs* to be accomplished



Resource sharing 2/2

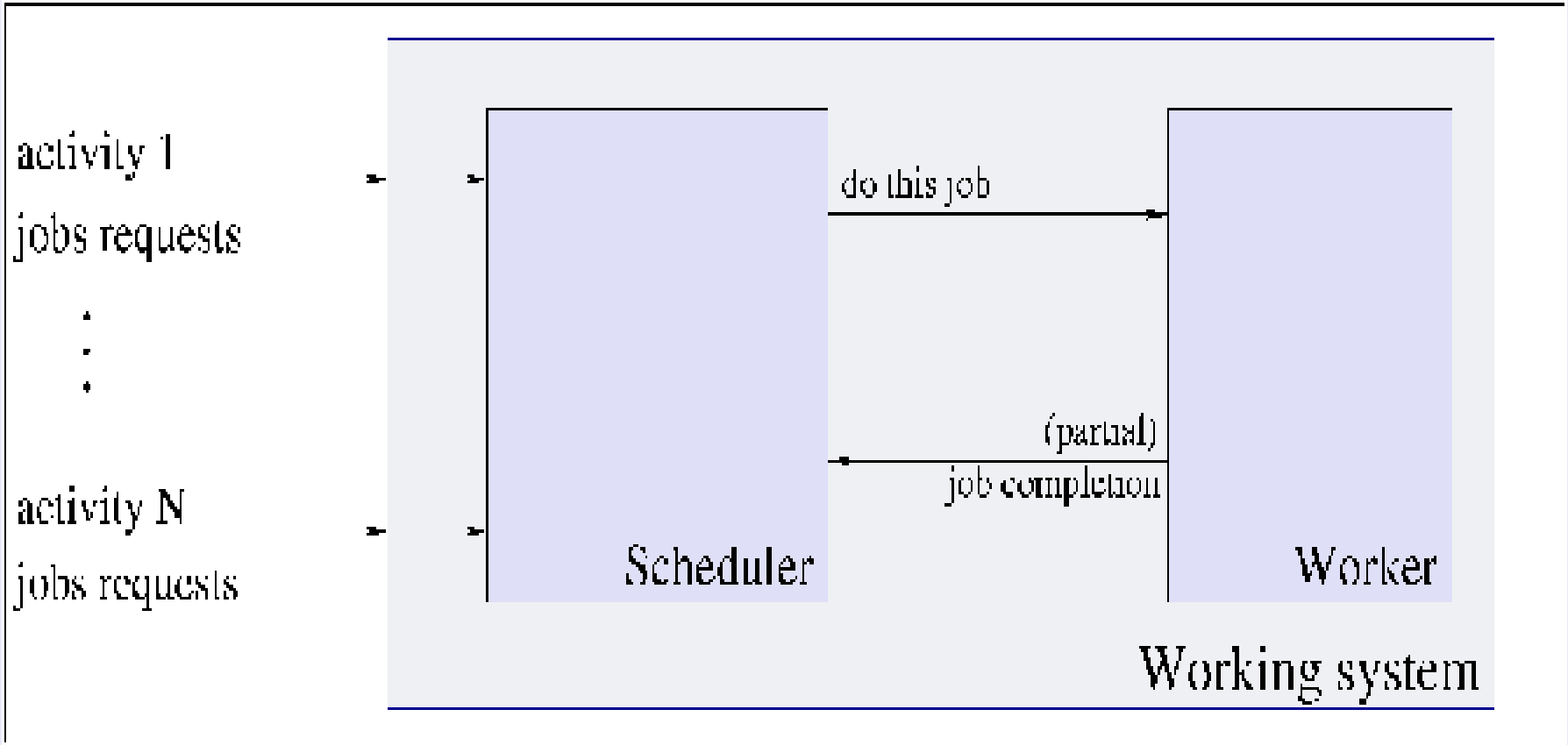
- System: a shared resource providing a certain service to a set of *N* activities
- Constraints:
  - The resource delivers a finite amount of service per time unit
  - The resource can serve a limited number of activities at a time (only one in this survey)
  - Once an activity won the resource, it can not be preempted before a minimum *non preemption interval* has elapsed
- Goal: schedule the resource so as to meet the service requirements of the activities
- Focus: each activity requires a certain *share* of the total service provided by the resource

Assumptions and notations

- No hypothesis on the nature of the work, except that it can be measured
- (*Total*) *System power R(t):* total amount of work per time unit done by the system at time *t*
- Each job has a finite *length* (amount of work needed to accomplish it), known as soon as the job is presented by an activity
- An activity is said *backlogged*, if it has pending jobs (possibly under service)
  - *B(t):* set of all the activities backlogged at time *t*
- How does the system work?

General model

A *worker* (shared resource) actually makes the work requested by the activities, while a scheduler allocates jobs onto the worker



Proportional Share approach

- Weight assignment: each activity *i* has a weight ϕ_i associated with it

- Service distribution: each activity gets a *share* of the total amount of work done by the system *proportional* to its weight

Outline

- A general model for Proportional Share
- Perfect fairness: GPS and WF²Q
- An efficient way to fairness: SP-RPS and WF²Q+
- Three practical implementations
- What is wrong with the schedulers not based on GPS simulation?

Summary

- Perfect fairness: GPS and WF²Q
  - GPS service: the *chimera*
  - GPS benefits
  - Worst-case Fair Weighted Fair Queueing (WF²Q)
  - Implementing WF²Q

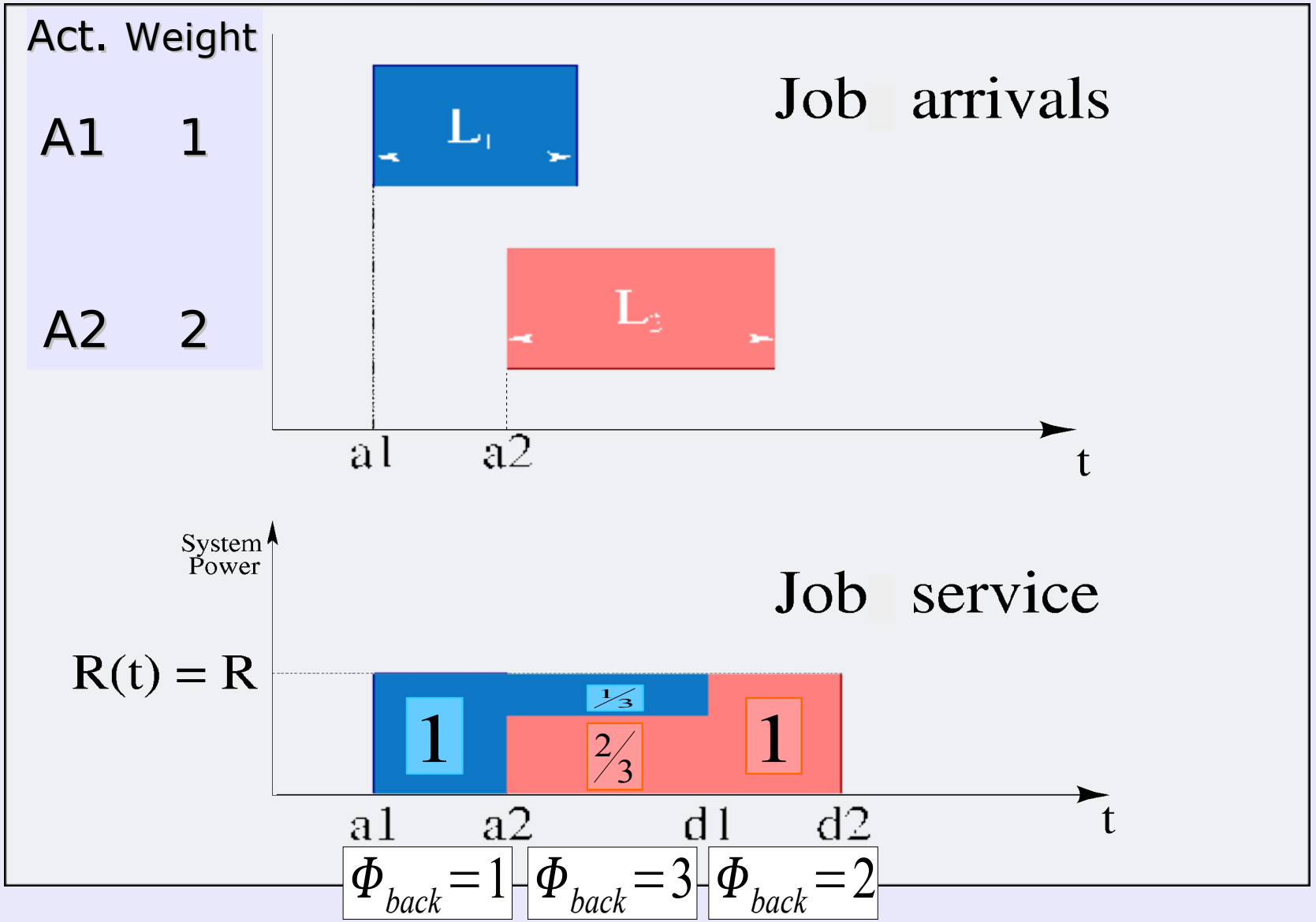
GPS 1/2

- Generalized Processor Sharing (GPS) server [Parekh and Gallager, 1992]:
  - Work-conserving fluid system that serves all backlogged activities simultaneously
- *dW(t)=R(t)·dt*: total amount of work done by the system at time *t*
- The GPS server provides each backlogged activity is with a *share* of the *system power:*

$$dW_i(t) = \frac{\phi_i}{\sum_{j \in B(t)} \phi_j} dW(t) = \frac{\phi_i}{\Phi_{back}(t)} dW(t) \quad \forall i, \forall t$$

instantaneous share: perfect fairness

GPS 2/2


Act. Weight
A1 1
A2 2
Job arrivals
L1
L2
a1
a2
t
System Power
R(t) = R
Job service
1
1/3
2/3
1
a1
a2
d1
d2
t
Φback = 1
Φback = 3
Φback = 2

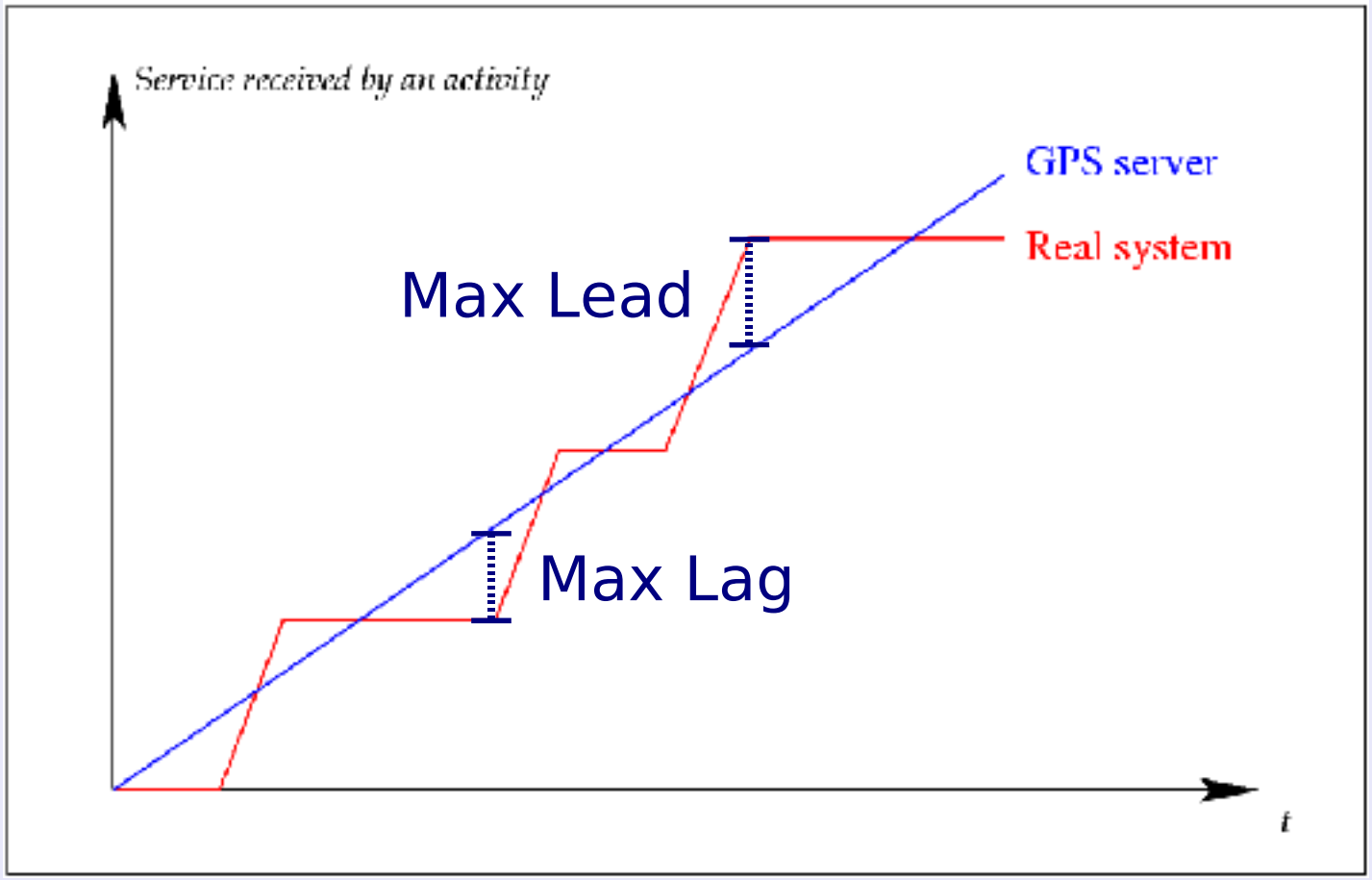
GPS and the real world

- Practical working systems can serve only one job (activity) at a time
- Context switch from one job to another has a finite cost, where possible
  - Minimum *non preemption* time interval
- GPS is not realistic

GPS benefits

- Due to its perfectly fair allocation, the GPS server can be used as a reference system for:
  - Evaluating the fairness of practical schedulers
  - Implementing fair schedulers through on-line simulation of a GPS server
    - internally simulate a GPS server
    - try to stay as close as possible to the service it provides

Evaluating fairness



- Fairness measure: maximum per-activity *deviation* (lead/lag) with respect to the GPS service

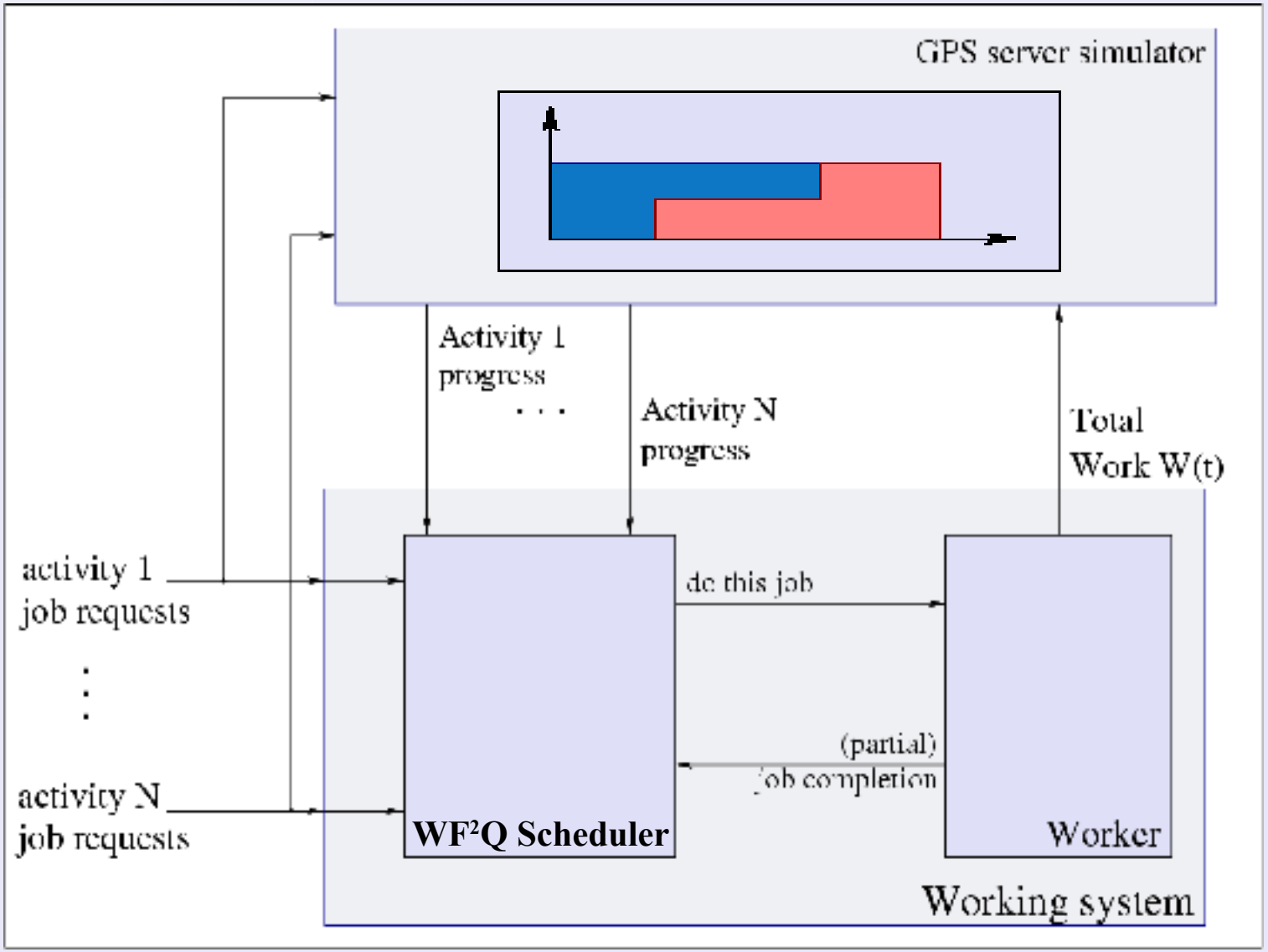
Summary

- Perfect fairness: GPS and WF²Q
  - GPS service: the *chimera*
  - GPS benefits
  - Worst-case Fair Weighted Fair Queueing (WF²Q)
  - Implementing WF²Q

WF²Q

- WF²Q [Bennett and Zhang, 1996] is a very *accurate practical* scheduler
- Schedules jobs so as to reduce the deviation with respect to the GPS service
- Internal simulation of a GPS server ...

WF²Q: GPS simulation



WF²Q policy

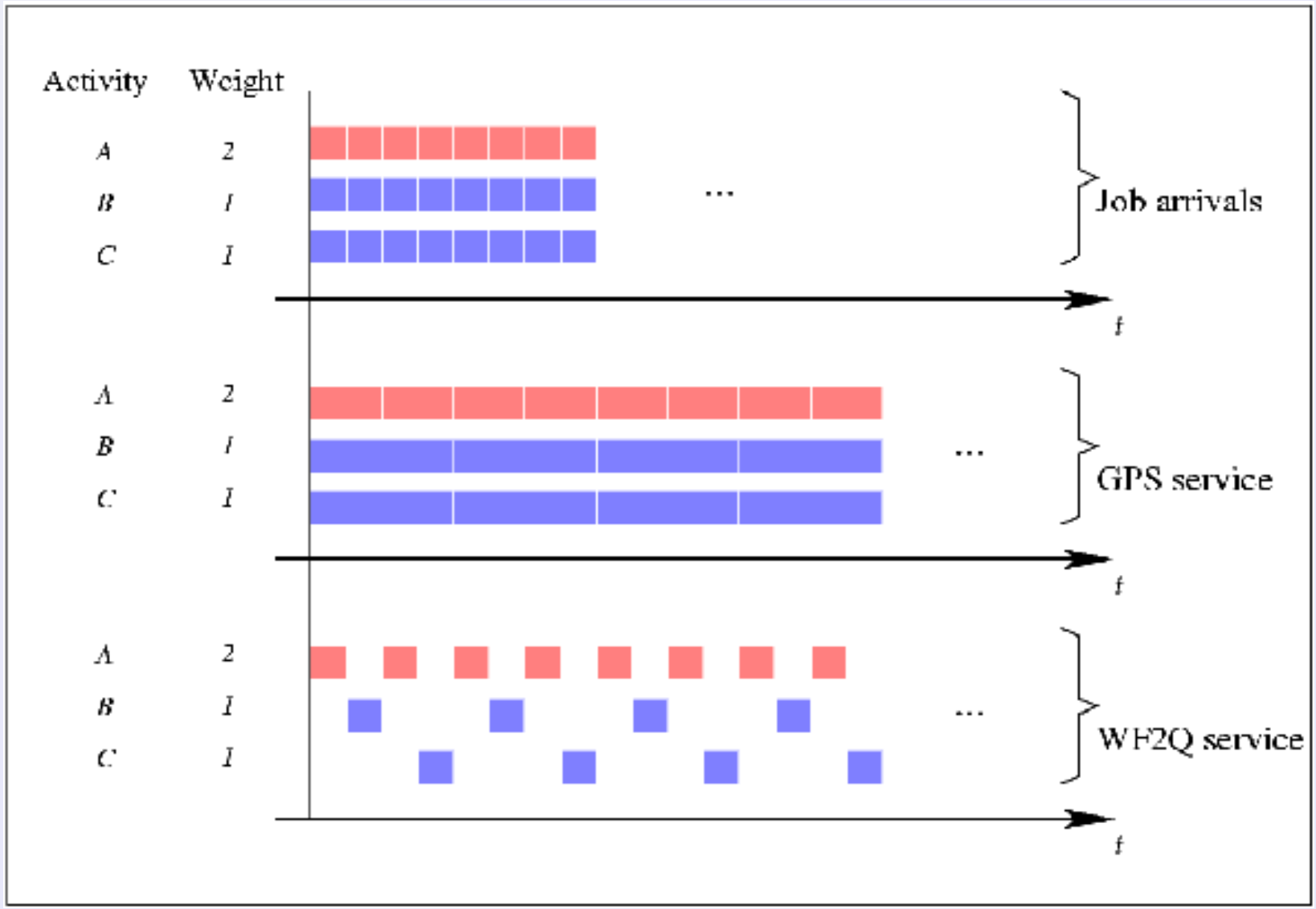
- Policy: each time the worker is ready, choose,

Lead
- among the jobs already started in the GPS

Lag
- the next one that would finish in the GPS

if no other job were to arrive

WF²Q: an example



Optimum service

- Physical constraints: one activity at a time, minimum non preemption interval
  - Per-activity *minimum* deviation
  - Equal to the maximum amount of work provided during the minimum non preemption interval
- *Optimum service*: the discrepancy with respect to the GPS service never exceeds the minimum deviation

- WF^2Q achieves the optimum service

Outline

- Perfect fairness: GPS and WF²Q
  - GPS service: the *chimera*
  - GPS benefits
  - Worst-case Fair Weighted Fair Queueing (WF²Q)
  - Implementing WF²Q

Implementing WF²Q: *P(t)*

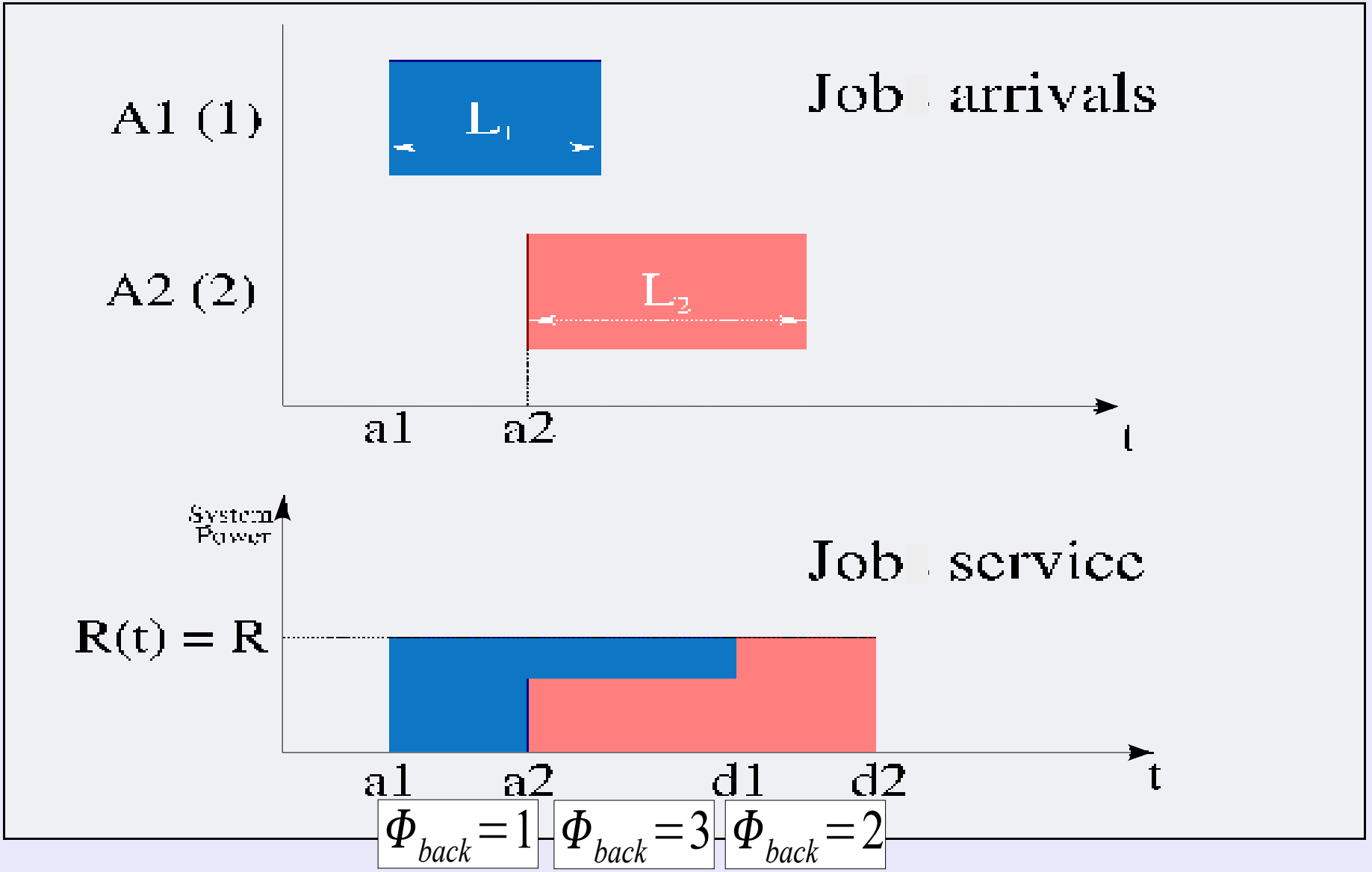
- Job start and finish times in the simulated system are hard to compute: function of *R(t)*, and change in consequence of job arrivals
- Efficient implementation: jobs timestamped using the system potential function:

$$P(t) \equiv \int_0^t \frac{1}{\Phi_{back}(\tau)} dW(\tau)$$

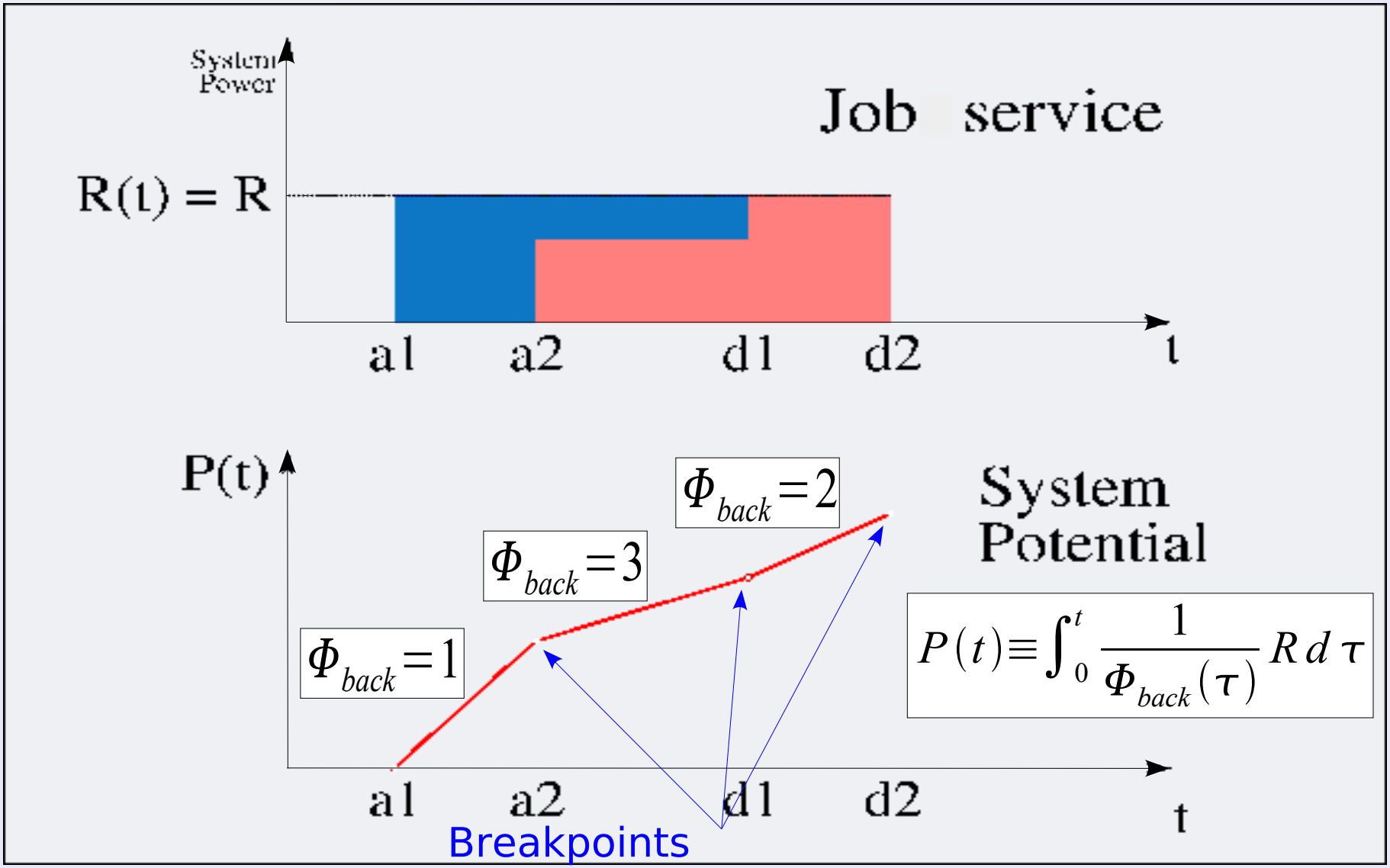
- Since $$dW_i(t) = \frac{\phi_i}{\Phi_{back}(t)} dW(t) = \phi_i dP(t)$$

P(t) measures the amount of *normalized* work received by each backlogged activity in the GPS server

Evolution of *P(t)* 1/3



Evolution of *P(t)* 3/3



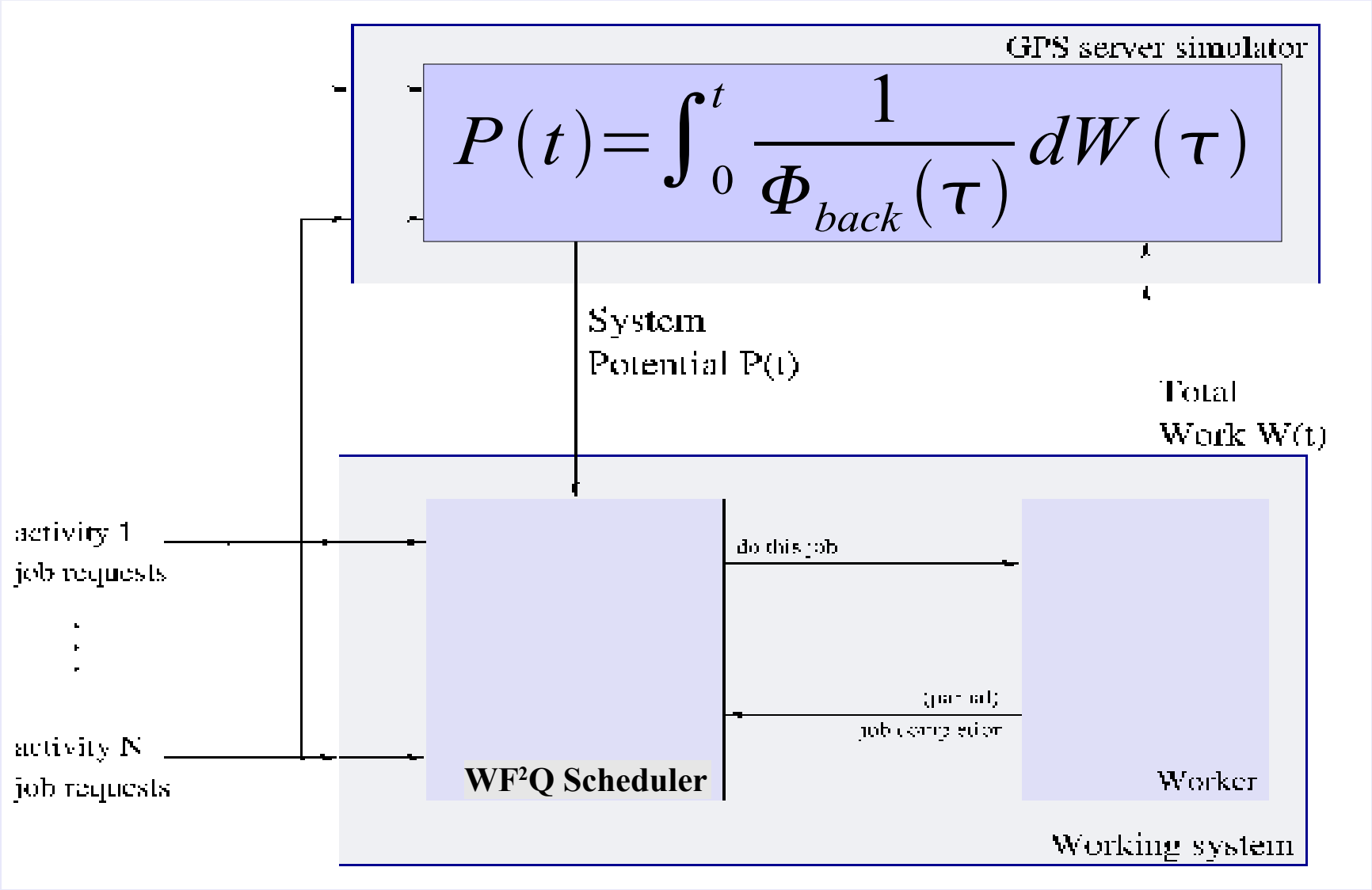
WF²Q implementation

- Jobs efficiently timestamped upon arrivals and easily ordered/served according to timestamps

- WF²Q must know the current value of *P(t)* each time the worker is ready, and each time a new job arrives

- Simulating the GPS server means tracking the system potential function

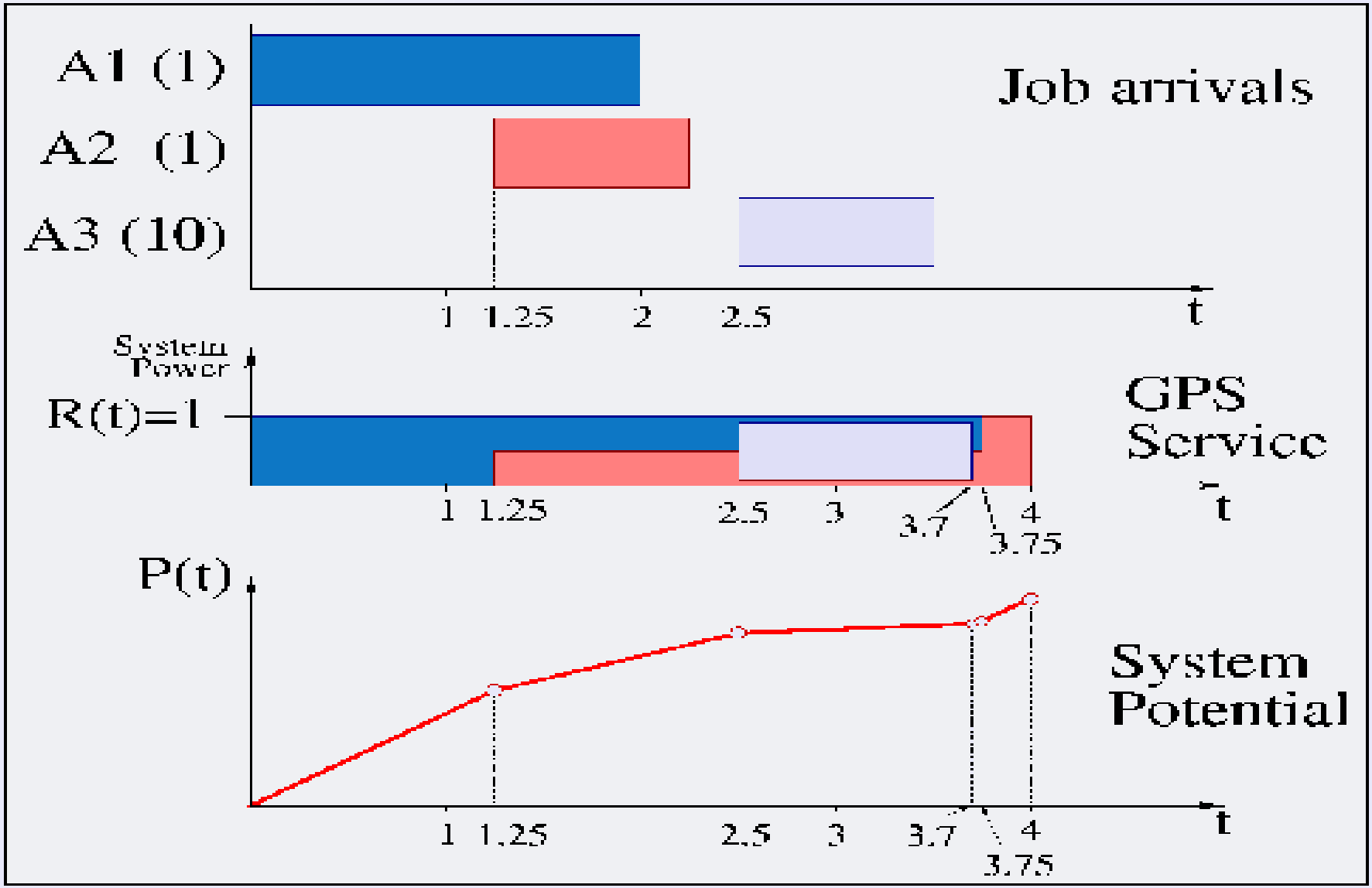
WF²Q: GPS simulation



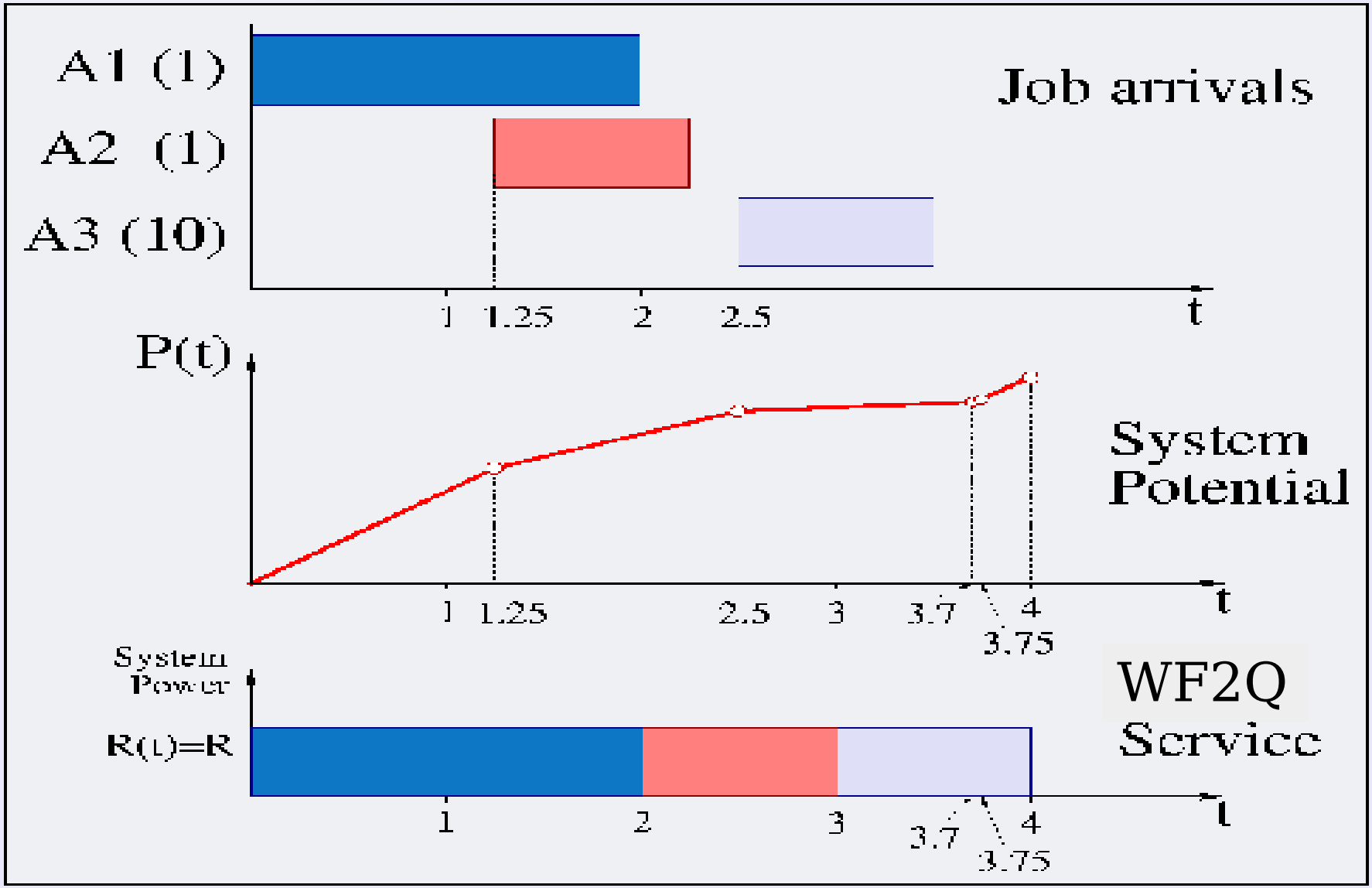
P(t) slope changes often 1/3

- Ordinary situation:
  - Three activities: *A1* with weight *1*, *A2* with weight *1*, *A3* with weight *10*
  - System power is equal to *1* work unit per time unit
  - Overhead issues impose that the system can perform a limited number of operations per time unit
  - Job maximum arrival rate is equal to one job per time unit
  - Job length is an integer multiple of one work unit

P(t) slope changes often 2/3



P(t) slope changes often 3/3



Complexity of GPS simulation

- Since the GPS server serves activities simultaneously, job finish times can be arbitrarily slightly skewed
- *O(N)* departures can occur in an arbitrarily short time interval, e.g. minimum job service time

- Existing algorithms for simulating the GPS server do one step per event (slope changing), hence they have *O(N)* complexity per job service time

Complexity of WF²Q

- Choosing the next job to transmit as a function of the current value of the system potential has *O(logN)* complexity
- But tracking the system potential of the GPS server has *O(N)* complexity per job service time
- WF²Q has *O(N)* complexity per job service time

Alternative algorithms

- This complexity is unfeasible in high speed applications with a high number of competing activities
- Several alternative proportional share schedulers proposed over the last ten years
- Renounce to the optimum service to avoid the burden of the GPS simulation
- Worst-case Fair Weighted Fair Queueing Plus (WF^2Q+) [Bennett and Zhang, 1996] is probably the most accurate one ...

Outline

- A general model for Proportional Share
- Perfect fairness: GPS and WF²Q
- An efficient way to fairness: SP-RPS and WF²Q+
- Three practical implementations
- What is wrong with the schedulers not based on GPS simulation?

Outline

- An efficient way to fairness: SP-RPS & WF²Q+
  - GPS worst-case service: SP-RPS
  - WF²Q+: a scheduler based on SP-RPS
  - Complexity of the SP-RPS simulation

GPS worst-case guarantees 1/2

- GPS distributes service with perfect fairness ...

$$dW_{i,GPS}(t) = \frac{\phi_i}{\Phi_{back}(t)} dW(t) \quad \forall i, \forall t$$

- ... but this type of service is heavy to track

- Worst-case service: all activities backlogged

$$dW_{i,GPS}^{Worst-Case}(t) = \frac{\phi_i}{\sum_{j=1}^{N} \phi_j} dW(t) = \frac{\phi_i}{\Phi_{TOT}} dW(t) \quad \forall i, \forall t$$

worst-case share

GPS worst-case guarantees 2/2

- ... when all activities are backlogged

$$dW_{i,GPS}^{Worst-Case}(t) = \frac{\Phi_i}{\Phi_{TOT}} dW(t) \quad \forall i, \forall t$$

- Activity *i* continuously backlogged during $[t_1, t_2]$

$$W_{i,GPS}^{Worst-case}(t_1, t_2) = \frac{\Phi_i}{\Phi_{TOT}} W(t_1, t_2)$$

- Idea: define a more approximate, but easier to simulate fluid system

SP-RPS

- Starting Potential – Rate Proportional Server (SP-RPS) [Stiliadis and Varma, 1997]
- Fluid work-conserving system guaranteing:

$$W_{i,SP-RPS}(t_1,t_2) \geq \frac{\phi_i}{\Phi_{TOT}} W(t_1,t_2) - \epsilon$$

- No information on the actual work received, except for the minimum amount guaranteed
- Same worst-case guarantees of the GPS server

WF²Q+ 1/2

- Policy: each time the worker is ready, choose,
  - Lead: among the jobs already started according to the minimum work guaranteed in the SP-RPS
  - Lag: the next one that would finish in the SP-RPS

  if no other job were to arrive

- Implementation: SP-RPS system potential, which now represents the minimum amount of normalized service guaranteed

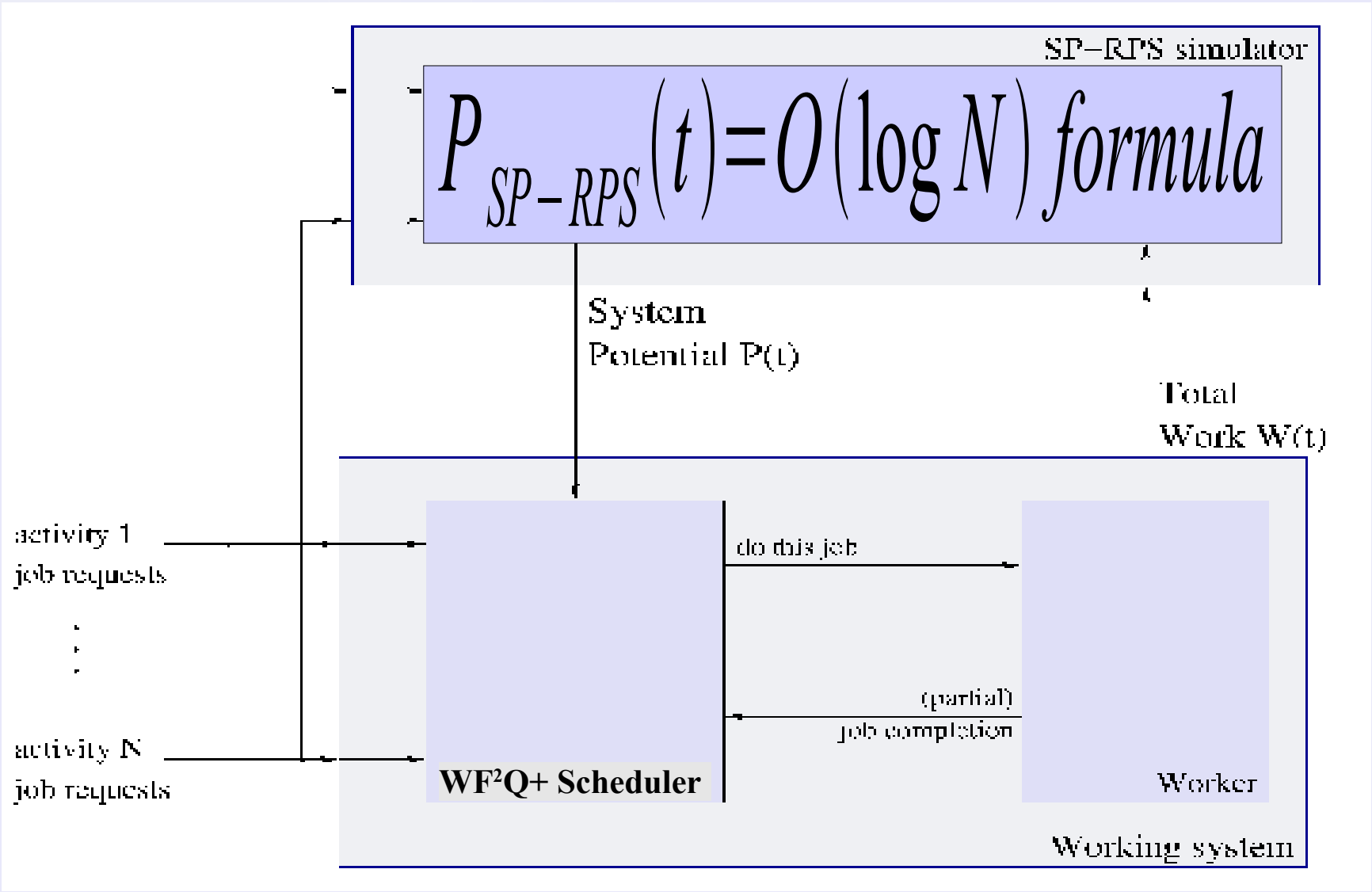
WF²Q+ 2/2

- Jobs efficiently timestamped upon arrivals and easily ordered/served according to timestamps

- WF²Q+ must know the current value of *P(t)* each time the worker is ready, and each time a new job arrives

- Simulating the SP-RPS server means tracking its system potential function

WF²Q+: SP-RPS simulation



Complexity of WF²Q+

- Tracking the system potential of SP-RPS has *O(logN)* complexity per job service time
- Choosing the next eligible job to transmit as a function of the current value of the system potential has *O(logN)* complexity
- WF²Q+ has *O(logN)* complexity per job service time

Outline

- A general model for Proportional Share
- Perfect fairness: GPS and WF^{2}Q
- An efficient way to fairness: SP-RPS and WF^{2}Q+
- Three practical implementations
- What is wrong with the schedulers not based on GPS simulation?

Outline

- Three practical implementations
  - WF²Q+ implementation in Dummynet
  - Efficient Proportional Share Processor Scheduler (EPS²): a proportional share process scheduler
  - Hybrid: achieving fairness and high throughput in disk scheduling

WF²Q+ for Dummynet

- WF²Q+ implemented inside *Dummynet* network emulator, after having enriched WF²Q+ with the *REF(t)* update algorithm
- Efficient *heap* based implementation
- Suitable for network simulations and (non large scale) real traffic scheduling
- Part of latest FreeBSD official releases

EPS² for FreeBSD 1/3

- Can we use job scheduling algorithms for process scheduling in a time sharing operating system?
- In a time sharing operating system, there is no explicit notion of jobs
- Scheduler runs each *ready* process for a certain *time quantum*, then preempts it and assign the processor to another process
- Given a process, we can identify its jobs with its scheduler-assigned *time quanta*, and each job length with its allocated *quantum* duration

EPS² for FreeBSD 2/3

- Problem: processes can voluntarily stop before *quantum* expiration
- An on-line scheduler is not *clairvoyant*
- Extension for correctness:
  - *Guess* the job length to assign timestamps, charge the activity only for the *actual* fraction of the job executed
- Reduce deviation from fair service:
  - Use a job length *predictor*
  - Good results with a *linear predictor* and an *exponential predictor*

EPS² for FreeBSD 3/3

- Efficient Proportional Share Process Scheduler (EPS²)
- FreeBSD monoprocessor non invasive implementation available
- Made some performance tests
  - Test machine: AMD K6 400 Mhz
  - Maximum scheduler overhead on involuntary context switch
    - Standard scheduler: 3.6 usec
    - EPS²: 7.1 usec

Disk scheduling 1/2

- Job: transfer of a sequence of disk sectors
- Before a sector can be read/written the disk head must be on the track the sector belongs to, and the sector must be currently under the head

- Sector access time depends on seek and rotational latency, and it is not negligible with respect to time needed to read/write the sector
- The aggregate throughput is heavily influenced by the locality of the requests

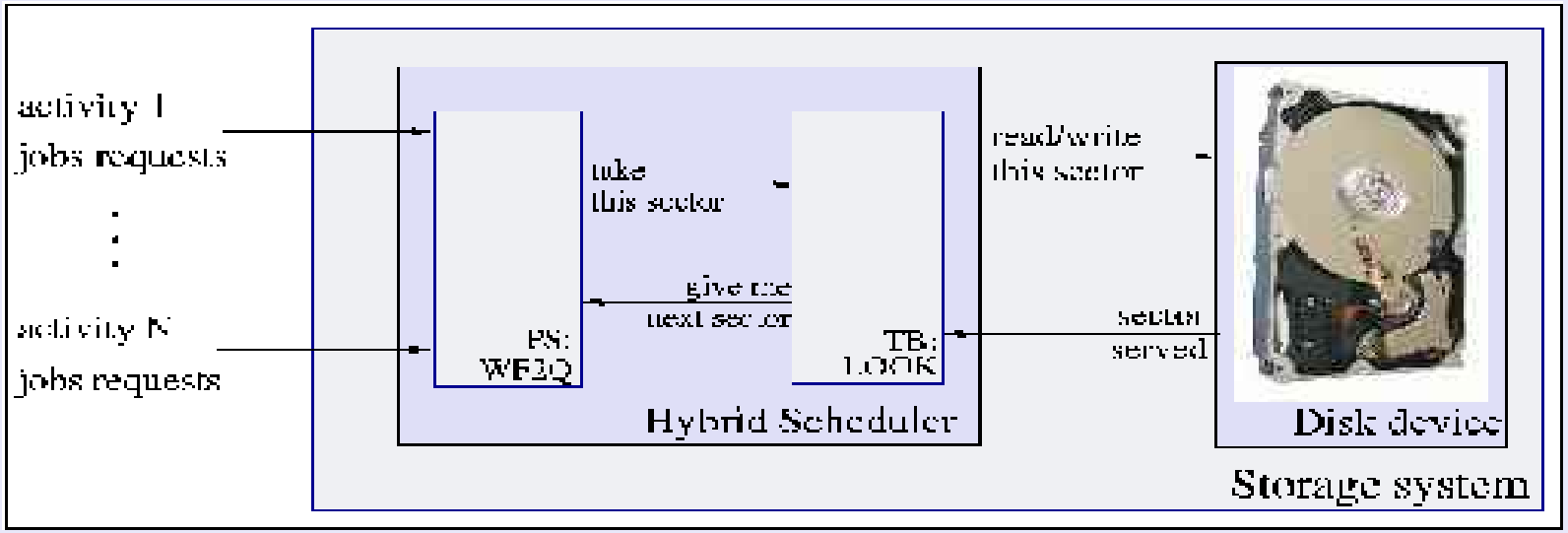
Disk scheduling 2/2

- There are several schedulers for boosting the throughput by reducing seek and/or rotational latencies
- One of the most used is the *elevator* (LOOK) algorithm

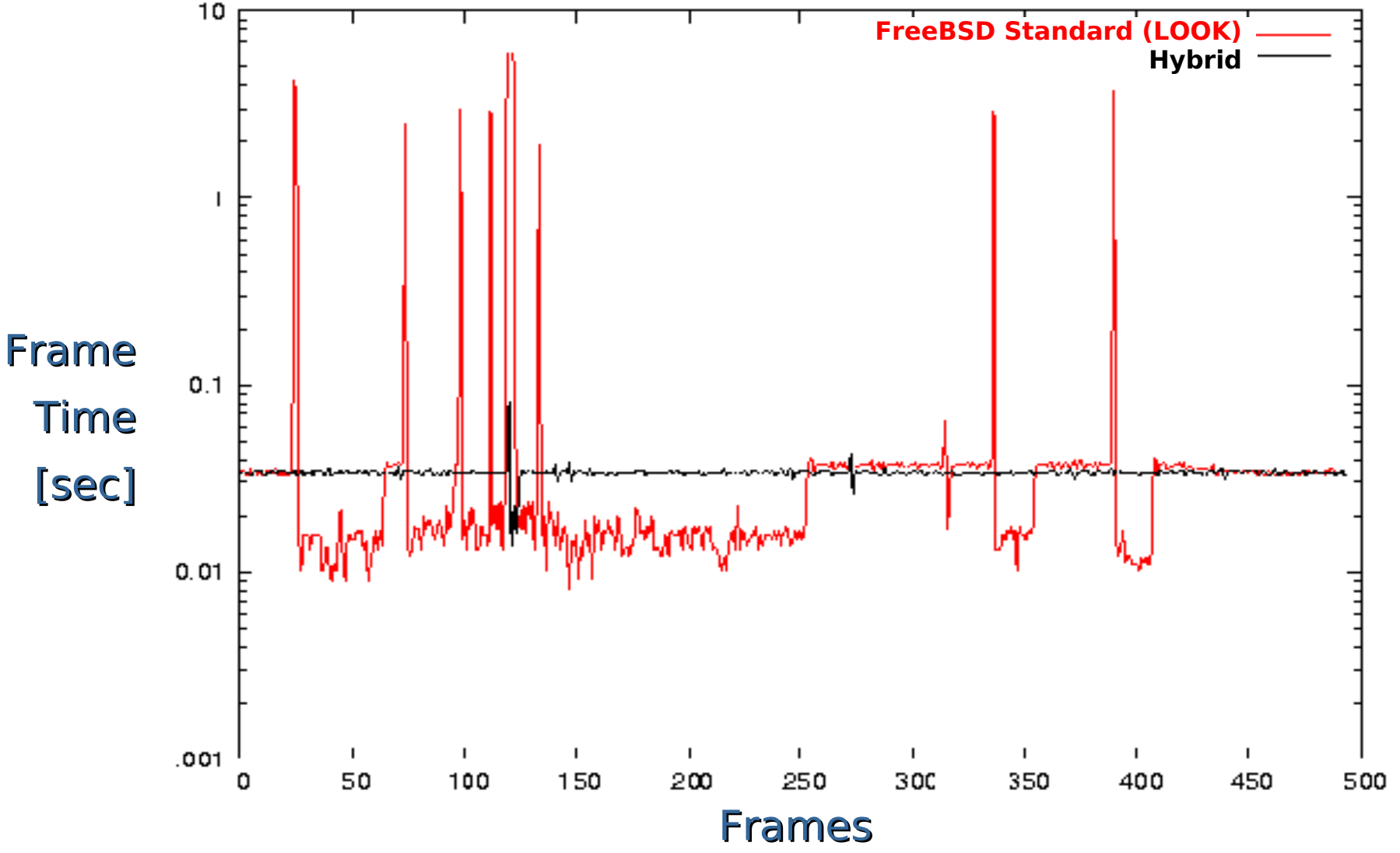
- Throughput boosting algorithms suffer from high variation of the response time
- Pure proportional share algorithms are not concerned with disk idiosyncracies

Hybrid

- Hybrid: tandem of a *proportional share* (PS) module based on WF²Q+, and a *throughput boosting* (TB) module based on the elevator algorithm
  - Requests do not go directly to the disk, but are first *prefetched* and reordered by the throughput booster
  - The extent of the reordering can be configured: trade off between throughput boosting and short term fairness



Experimental results



Outline

- A general model for Proportional Share
- Perfect fairness: GPS and WF^{2}Q
- An efficient way to fairness: SP-RPS and WF^{2}Q+
- Three practical implementations
- What is wrong with the schedulers not based on GPS simulation?

GPS emulation

- To the literature, all practical schedulers, apart from WF²Q, exhibit *O(N)*, or, worse yet, un-bounded deviation with respect to the GPS service
- WF²Q+ has *O(1)* deviation with respect to the *minimum* service guaranteed by the GPS server
- But the actual service provided by the GPS server could be much larger when some activities are idle
- WF²Q+ has *O(N)* deviation from the GPS service

Summary *1/2*

- Proportional Share is a well suited paradigm for distributing the service provided by a shared resource
- An ideal system, the GPS server, is used as a reference system for its perfectly fair service distribution
- There is a scheduling algorithm, called WF^2Q and providing the optimum service, i.e. the minimum possible deviation with respect to the GPS service
- Due to its internal GPS simulation, WF^2Q has been charged for *O(N)* complexity per job service

Summary *2/2*

- An alternative scheduling algorithm, called WF2Q+ and with *O(logN)* complexity per job service has been proposed
- WF^2Q+ can be used to implement efficient, yet accurate schedulers for process, packet and disk scheduling
- All existing schedulers, exept for WF^2Q (and including WF^2Q+) , have *O(N)* or, worse yet, unbounded deviation with respect to the GPS service
- In the end, either we pay *O(N)* complexity, or we settle for *O(N)* deviation
- A new algorithm for simulating the GPS server with *O(logN)* complexity has been recently proposed ...

Any questions?