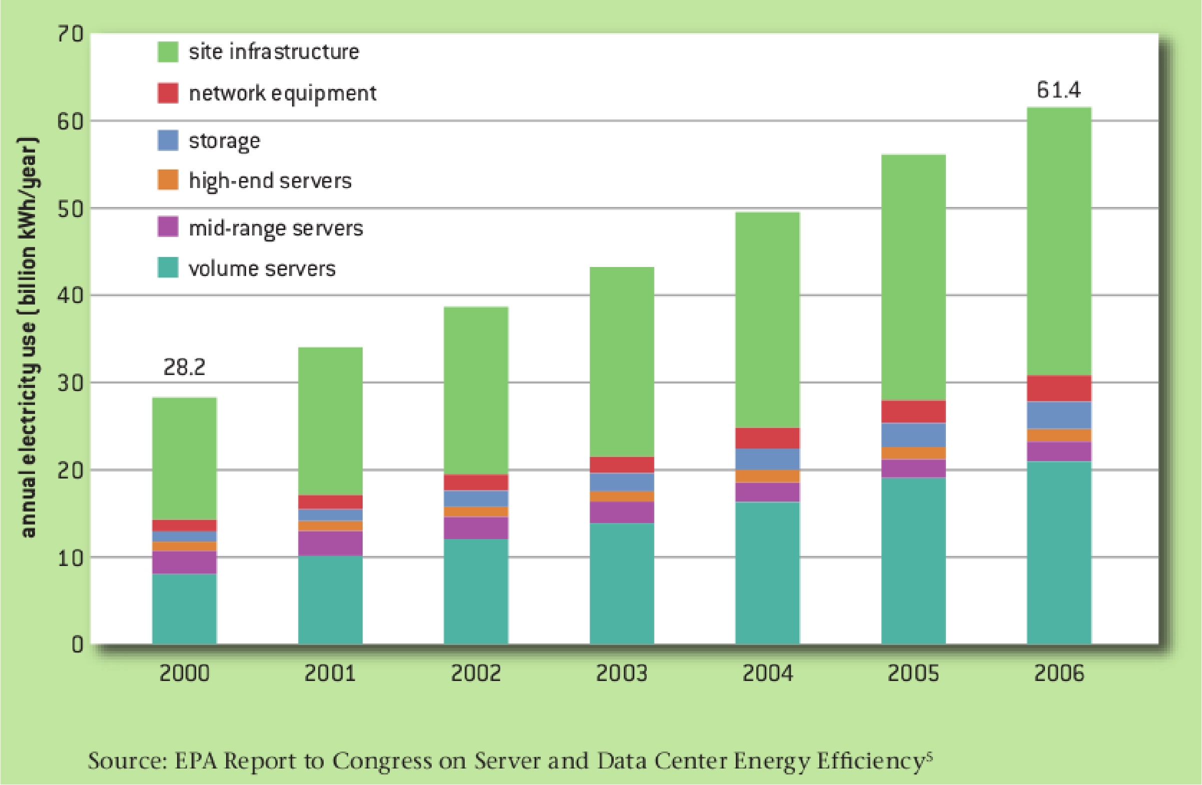
Optimizing the Energy Consumption of Large-Scale Applications

Moreno Marzolla

Dept. of Computer Science and Engineering
University of Bologna
marzolla@cs.unibo.it
http://www.moreno.marzolla.name/

Some Facts / 1

- The electricity bill accounts for a substantial fraction of the total operational costs of large IT infrastructures
- Servers are responsible for a large fraction of the total power consumed



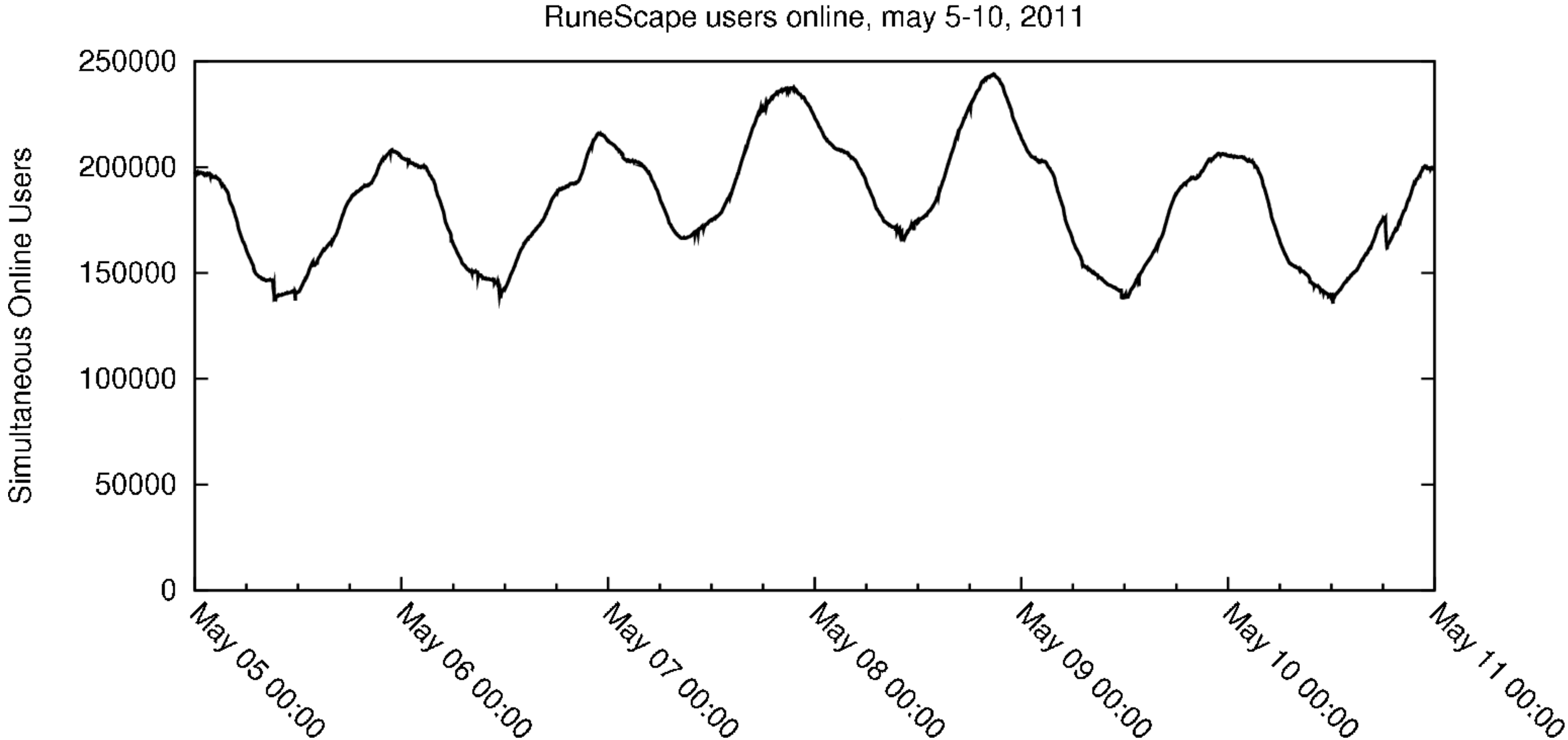
Source: *Toward energy efficient computing*, ACM Queue, http://queue.acm.org/detail.cfm?id=1730791

Some Facts / 2

- Online Data Intensive (OLDI) applications play a major role in widely used services
  - Online gaming
  - Search engine
  - Online advertisement
- OLDI applications
  - are driven by user-generated queries
  - usually have strict response time requirements

Example
www.runescape.com

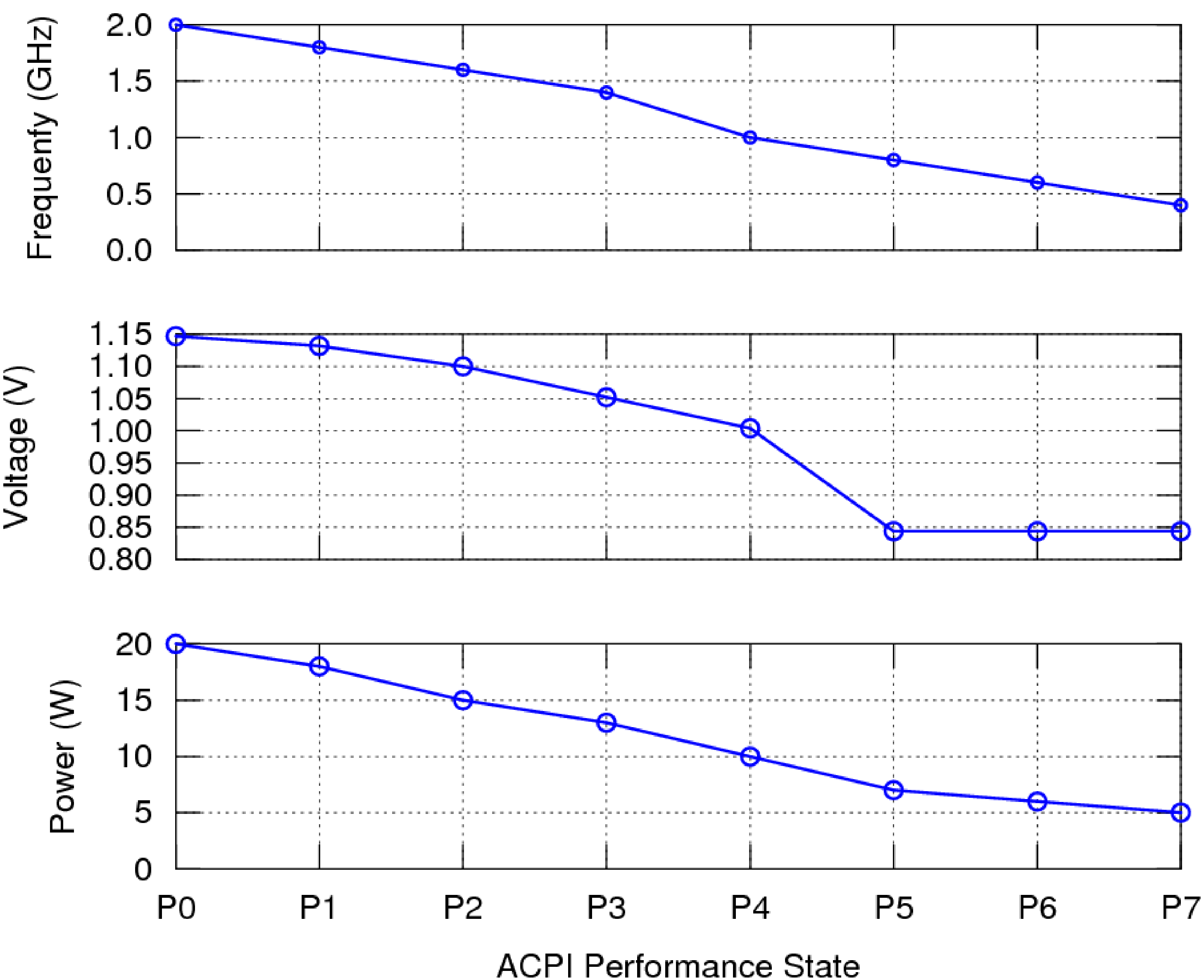
- Massive Multiplayer Online Game with a large user base
- Workload (number of players) exhibits periodic fluctuations



Reducing the energy consumption of OLDI applications

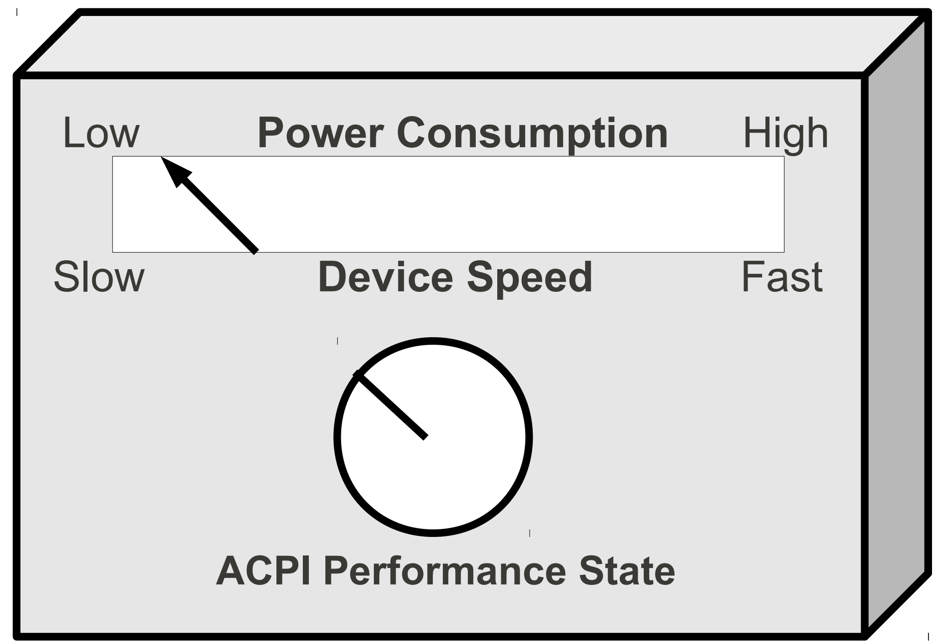
- Ideally, switch unused servers off when workload is low; bring them back when needed
- Given an observed workload W, compute the minimum service capacity which is "just enough" to keep the response time below R_{max}
- Problems
  - How to solve the optimization problem above?
  - Wear and tear of devices
  - Introduces delays (device startup/shutdown, application reconfiguration...) that the application might not tolerate

Frequency, voltage and power consumption for the 2 GHz VIA C7-M Processor



Introducing SAWYER
QoS-Aware Energy Manager

- Use ACPI performance states of devices to tune the power consumption / response time tradeoff
- Use a Queueing Network performance model to quickly estimate the correct ACPI state of each device such that the overall system response time is less than a given threshold R_{max}



Problem formulation

- Given:
  - K devices
  - L_k number of ACPI states supported by device k
  - $E_{k,s}$ power consumption of device k in ACPI state s
  - $RSP_{k,s}$ relative speed of device k in state s
  - R_{max} maximum allowed mean response time

$$\textbf{Minimize:} \quad E(\mathbf{S}) = \sum_{k=1}^{K} E_{k,S_k}$$

$$\textbf{Subject to:} \quad R(\mathbf{S}) \leq R_{\max}$$

$$S_k \in \{1, \ldots, L_k\}, \text{ for all } k \in \{1, \ldots, K\}$$

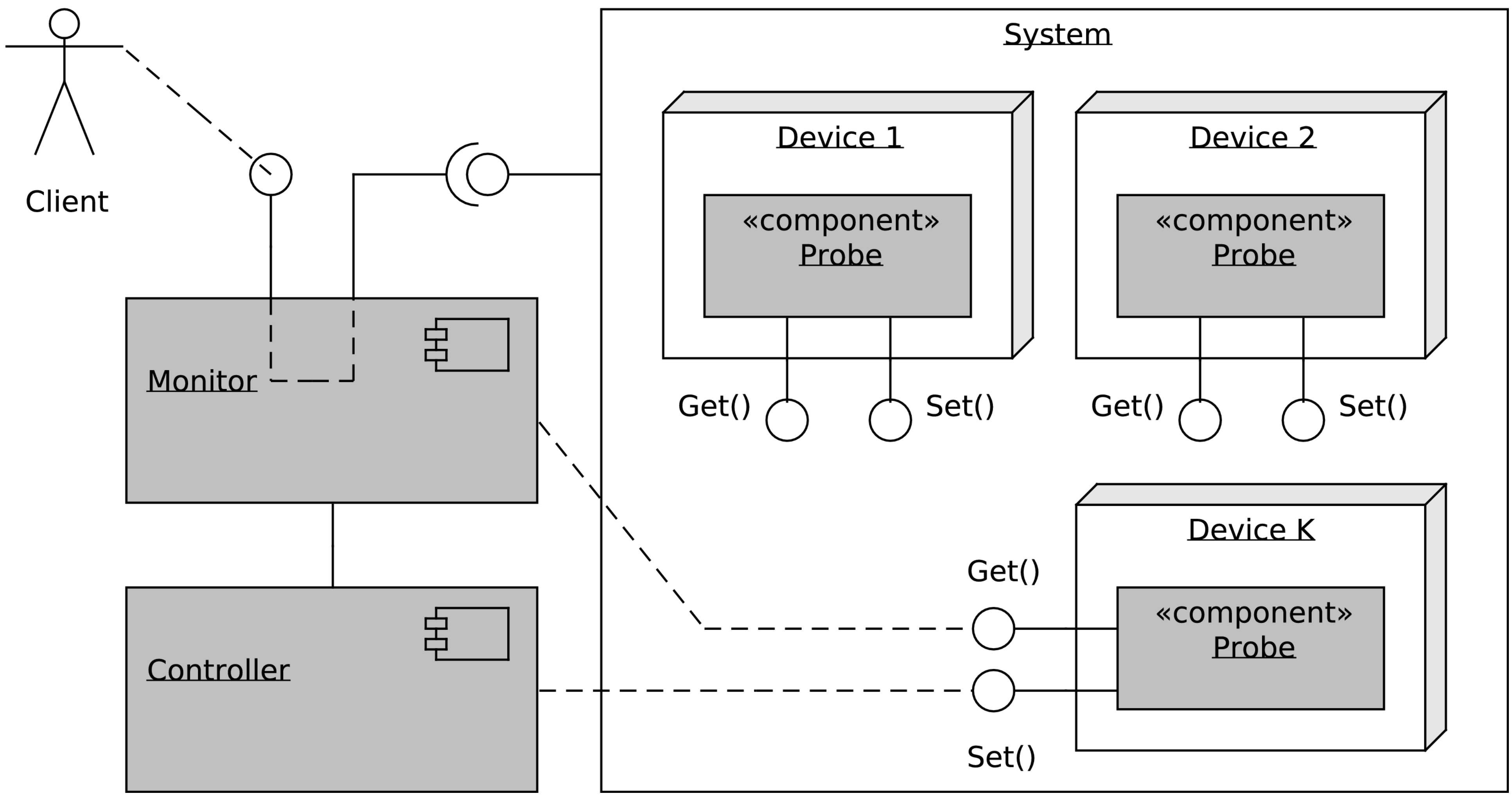
SAWYER: main idea

- Monitor the mean system response time R
- Every Δt:
- If R "too high"
  - Identify bottleneck device(s)
  - Speed up bottleneck devices by switching them to faster ACPI states
- If R "too low"
  - Identify non-bottleneck device(s)
  - Slow down non-bottlenecks by switching them to slower ACPI states
- Avoid trial and error!

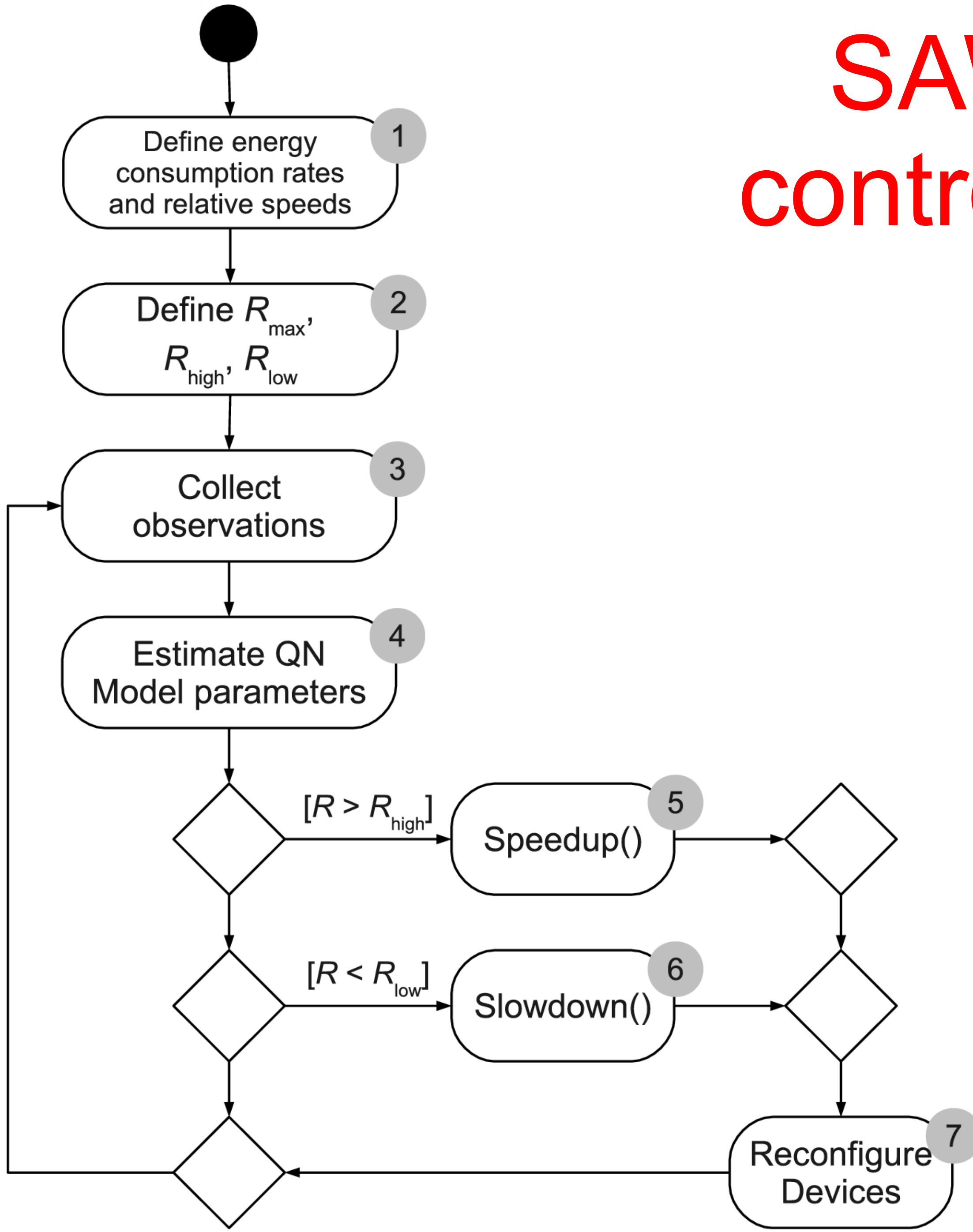
Queueing Network Model

- We model the system as a (closed) Queueing Network (QN); each device is modeled as a queueing center
- The QN parameters can be derived from measurements taken by software probes on the running system
- The QN is used to estimate the system response time with any ACPI setting of the devices

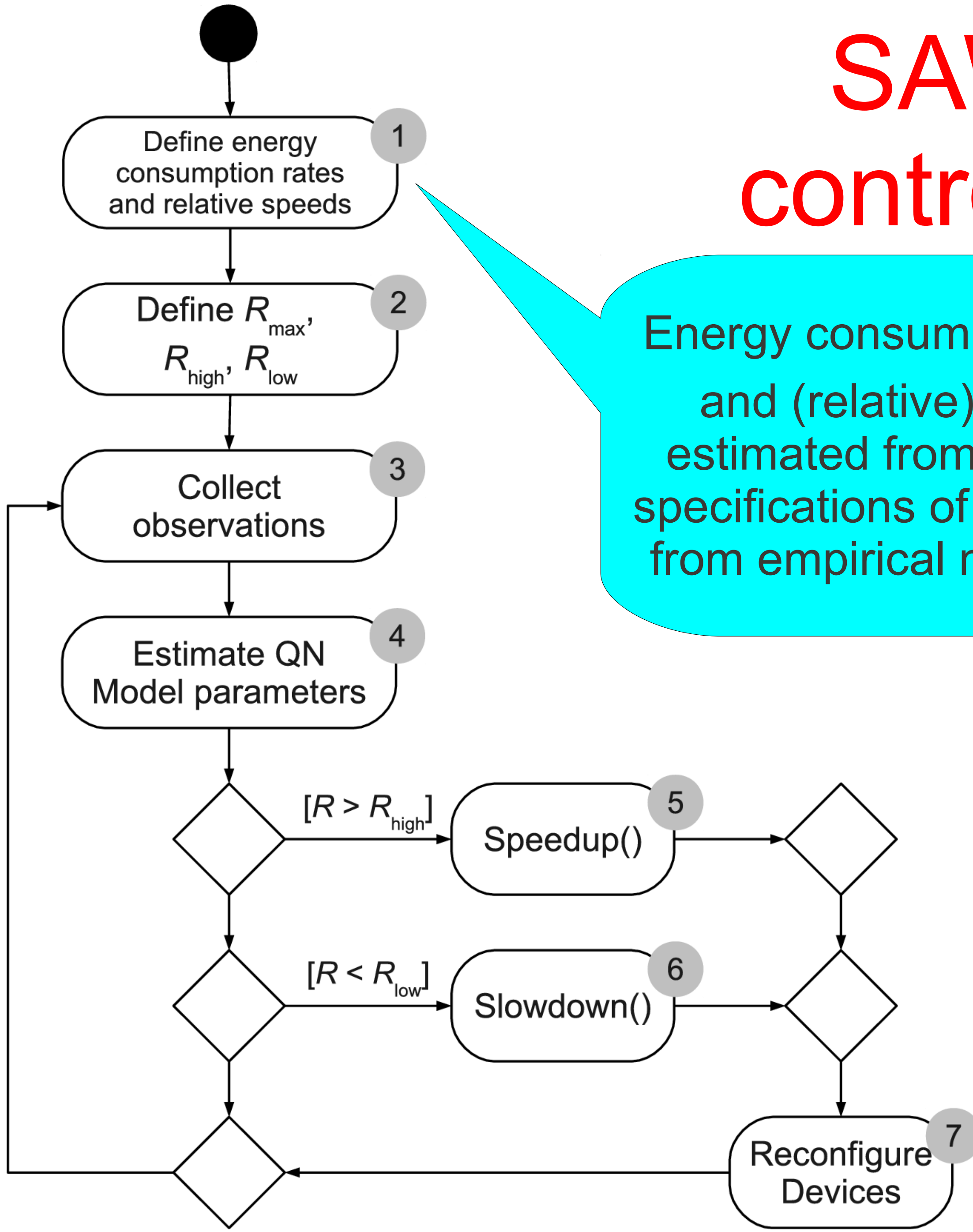
System Architecture



SAWYER control loop

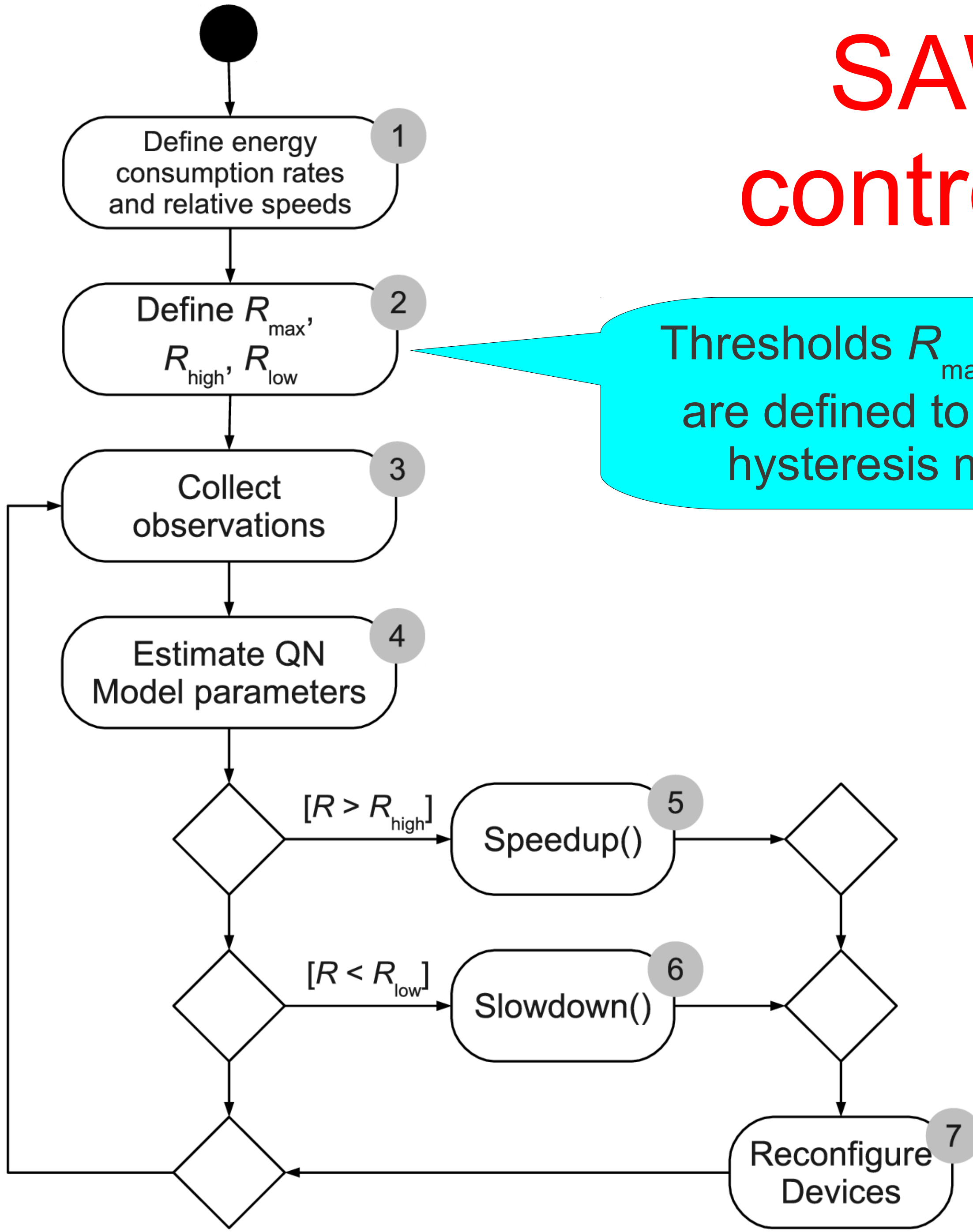


SAWYER control loop



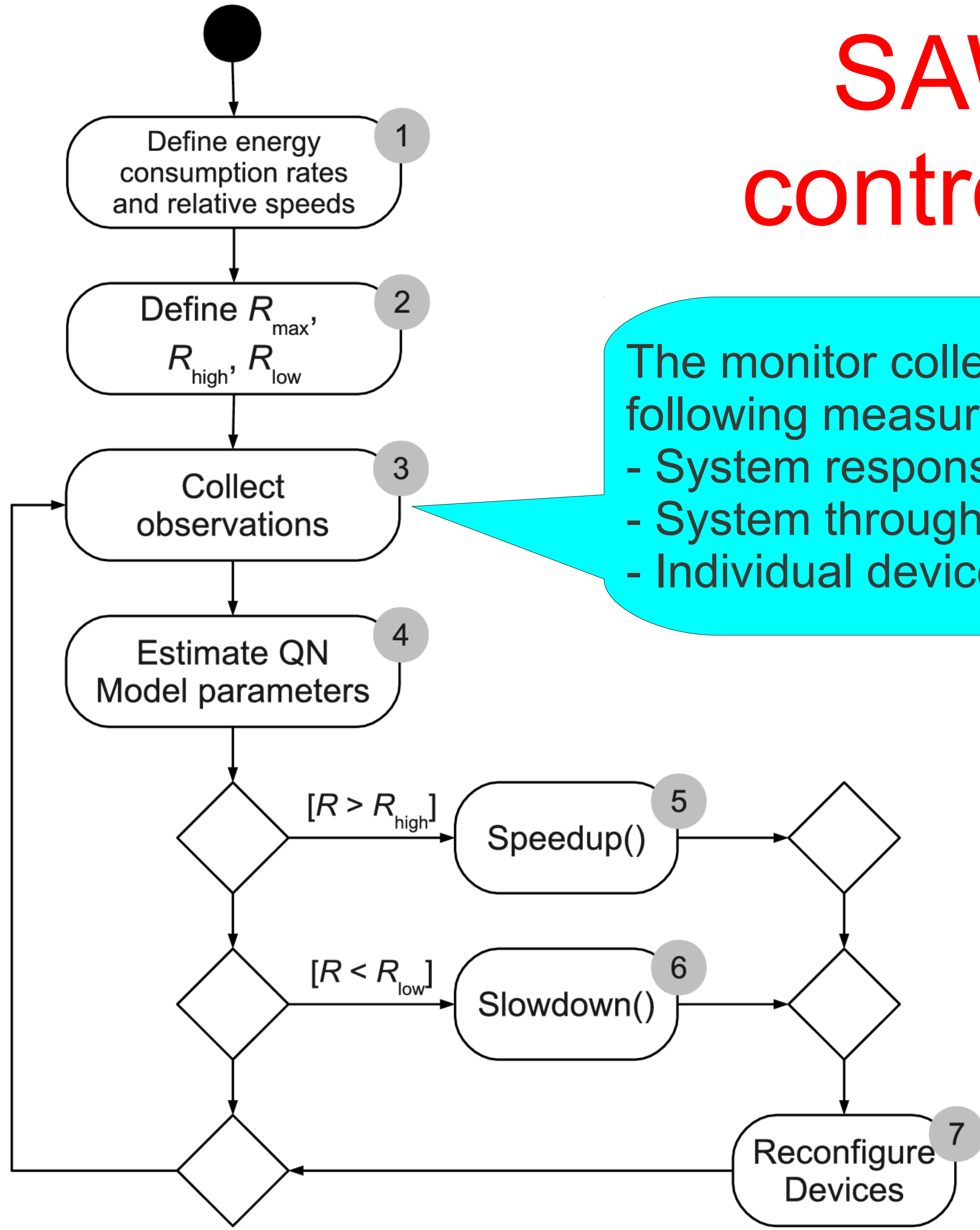
Energy consumption rates $E_{k,\,s}$ and (relative) speeds are estimated from the technical specifications of the devices, or from empirical measurements

SAWYER control loop



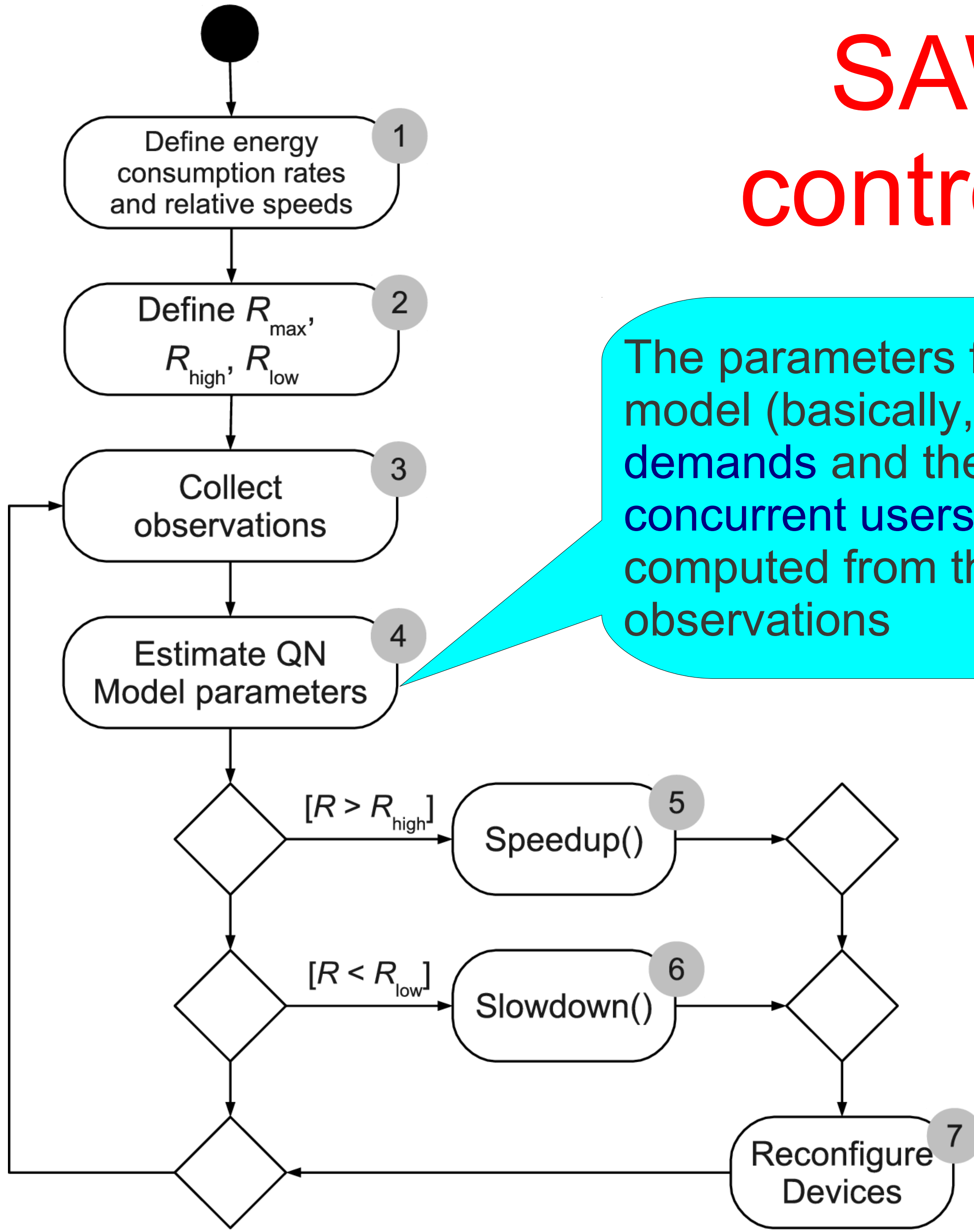
Thresholds $R_{max} > R_{high} > R_{low}$ are defined to implement a hysteresis mechanism

SAWYER control loop



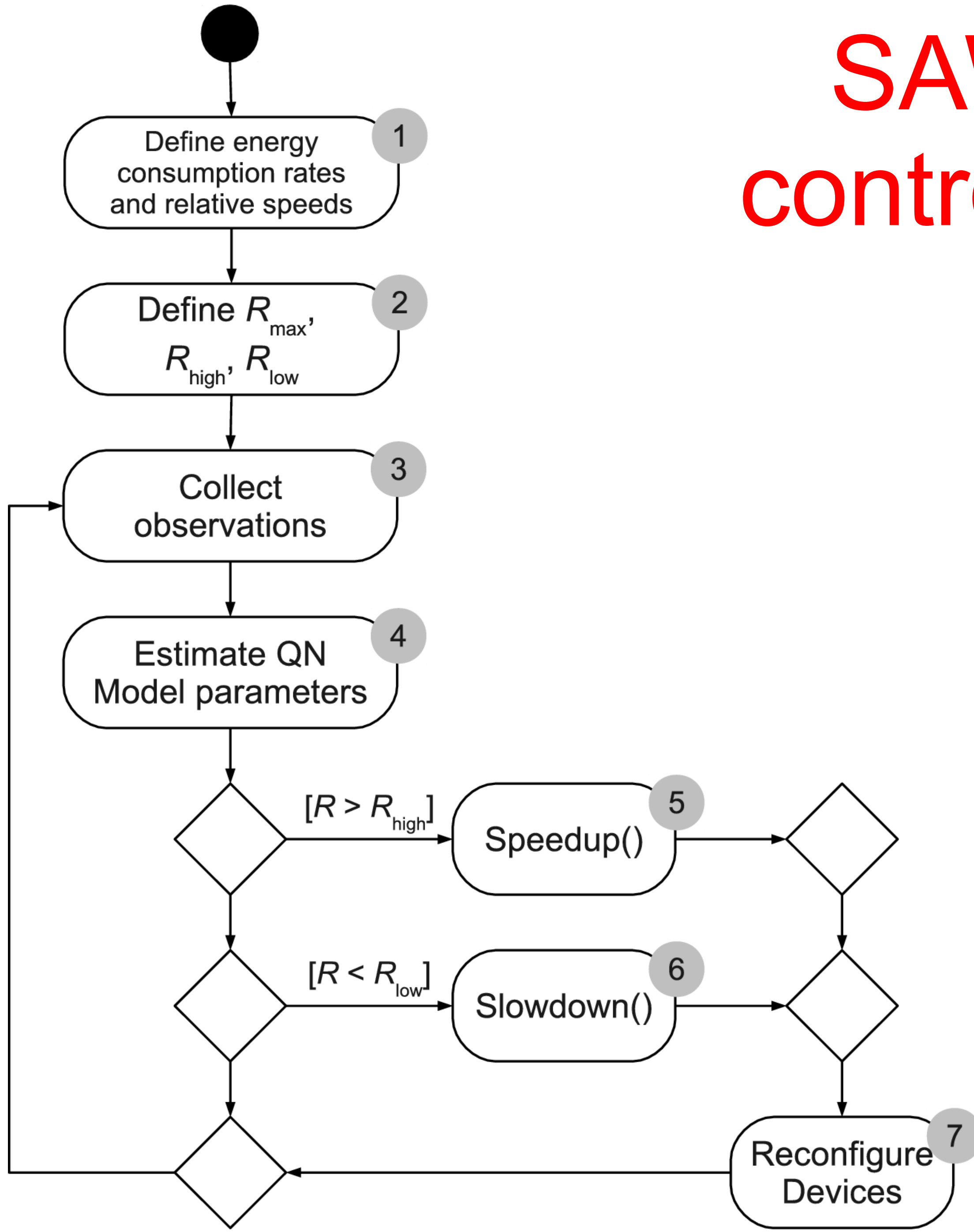
The monitor collects the following measures:
- System response time
- System throughput
- Individual device utilizations

SAWYER control loop



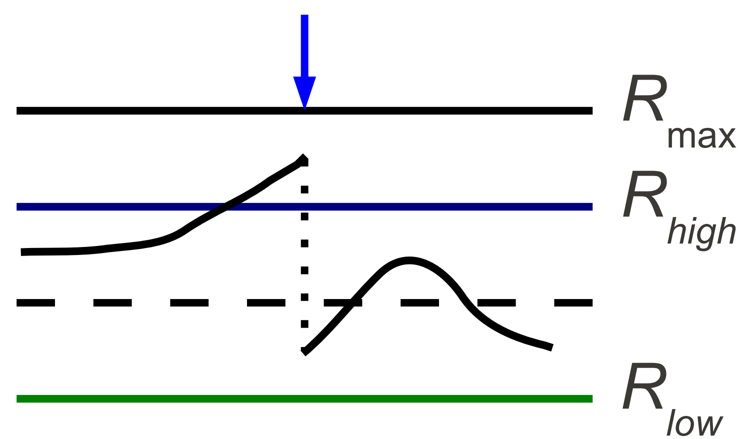
The parameters for the QN model (basically, the service demands and the number of concurrent users) are computed from the collected observations

SAWYER control loop



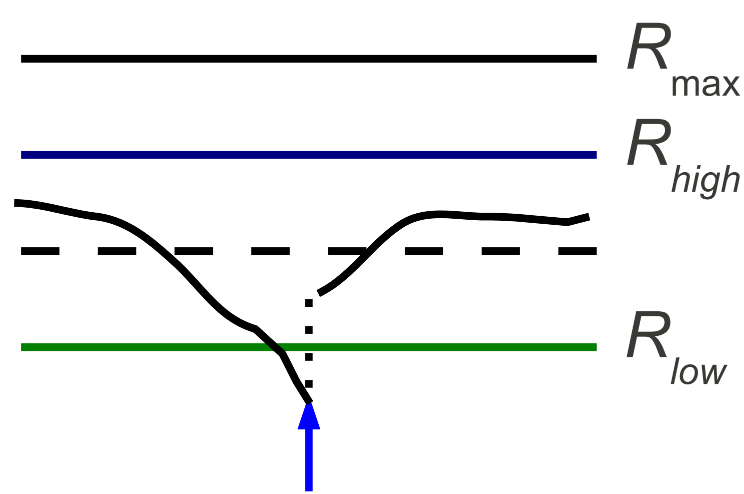
If $R > R_{high} \rightarrow$ Speedup

1. Consider the set S of all devices which can be switched to a faster ACPI state
   - If S is empty, stop
2. Select the component k in S with maximum ratio Service Demand / Power Consumption
3. Switch k to the next faster ACPI state
4. Estimate the new system response time R
   - If $R < (R_{high} + R_{low}) / 2$ stop
   - Otherwise, go to step 1

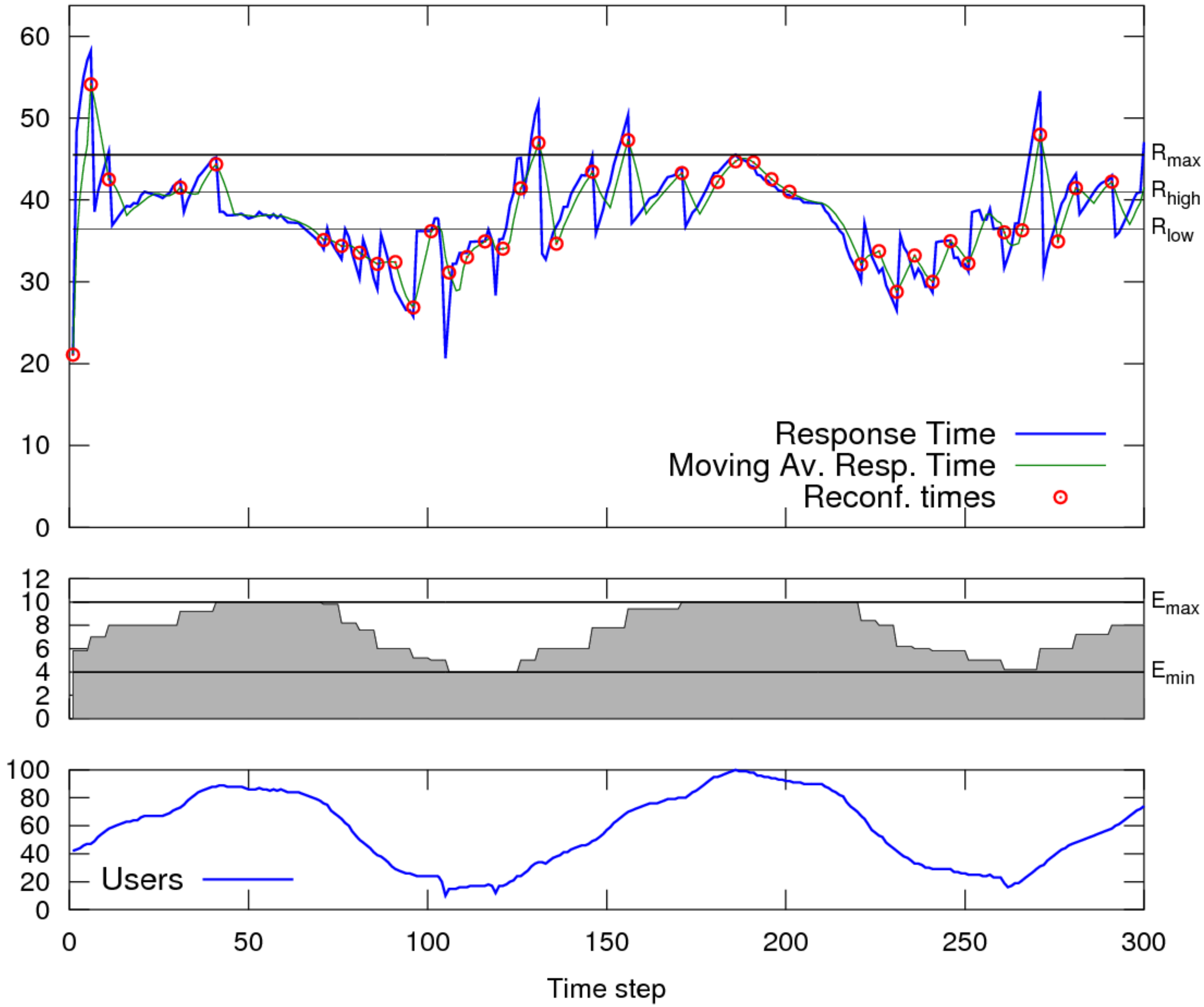


If $R < R_{low} \rightarrow$ Slowdown

1. Let S be the set of devices which can be slowed down
   - If *S* is empty, stop
2. Select the device *k* in *S* with minimum ratio
   New Service Demand / New Power Consumption
   - *New Demand = service demand of device k when switched to the next slower ACPI state*
3. Switch device *k* to the next slower ACPI state
4. Estimate new system response time *R*
   - If $R > (R_{high} + R_{low}) / 2$
     rollback and remove *k* from *S*
5. Go to step 2.

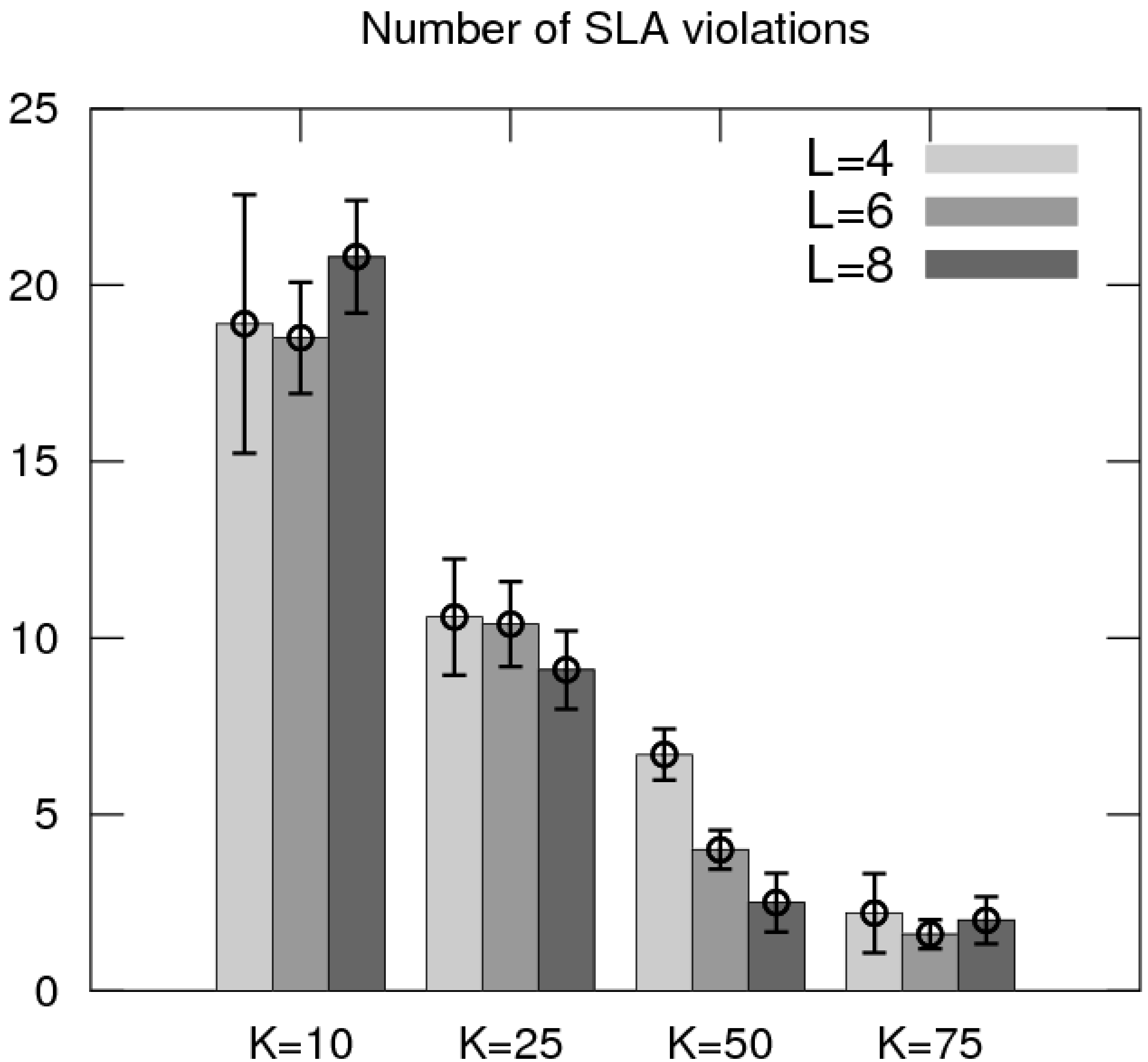


Example

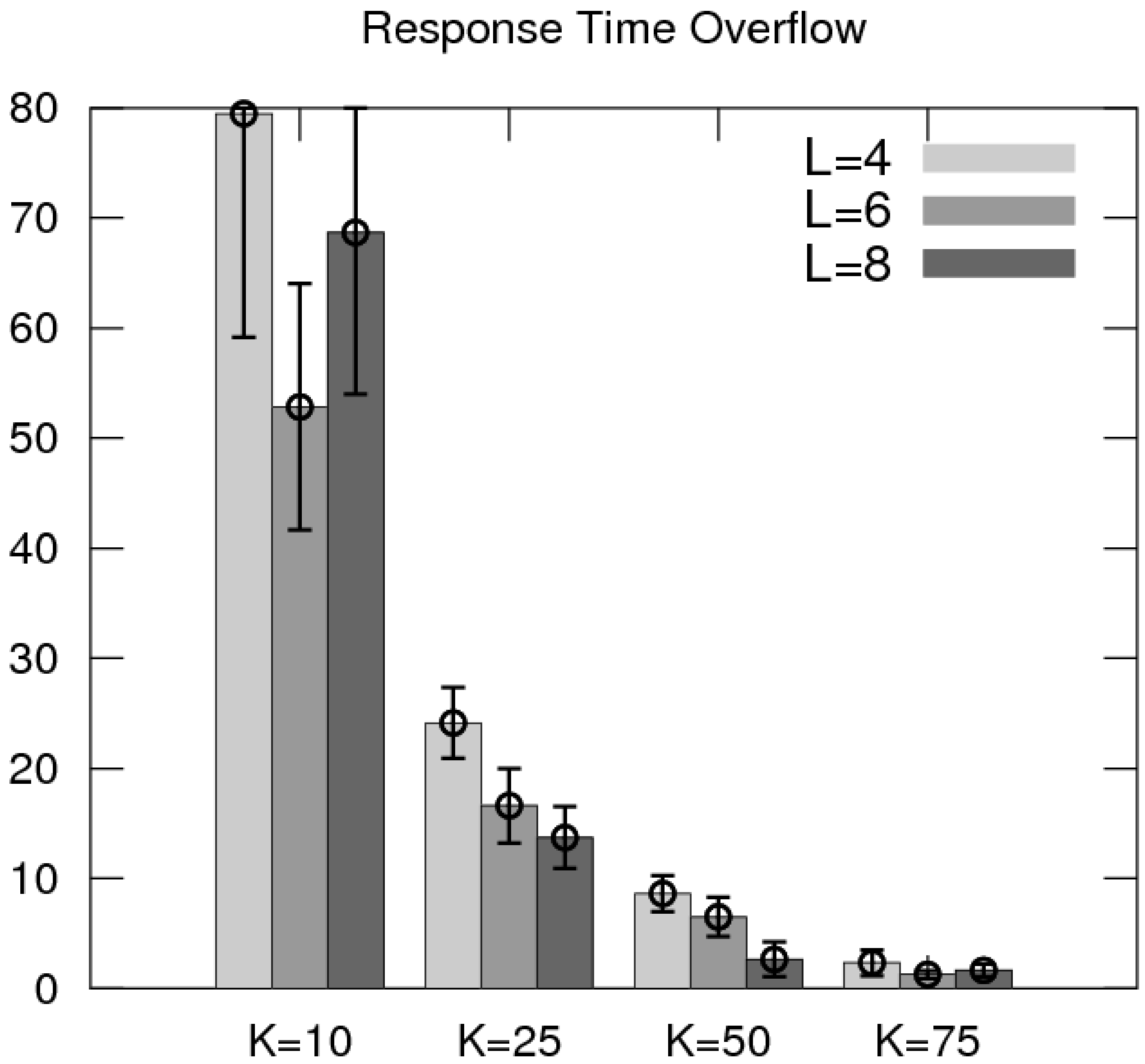


60
50
40
30
20
10
0
R_{max}
R_{high}
R_{low}
Response Time
Moving Av. Resp. Time
Reconf. times
12
10
8
6
4
2
0
E_{max}
E_{min}
100
80
60
40
20
0
Users
0
50
100
150
200
250
300
Time step

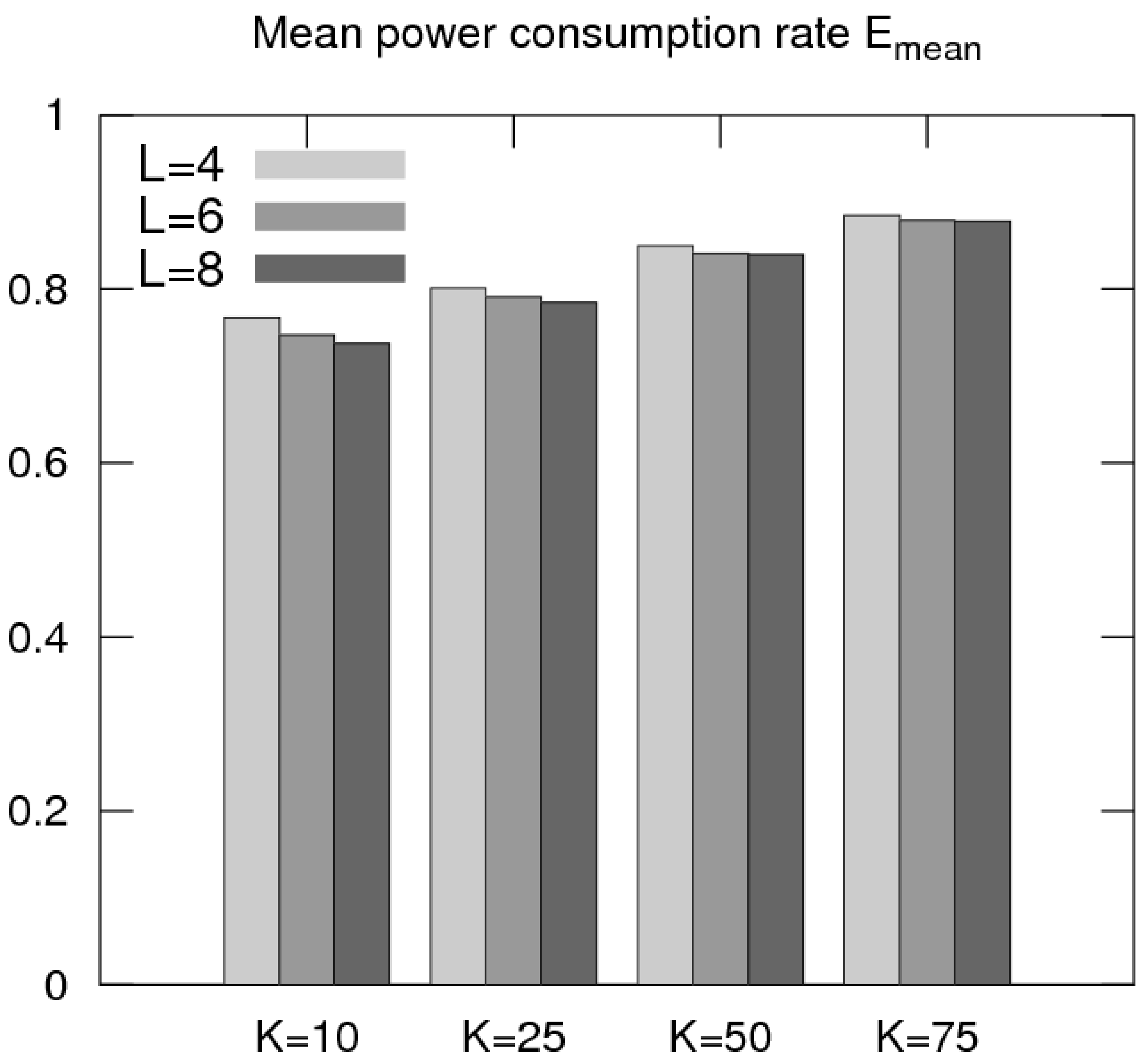
Performance Evaluation



Performance Evaluation



Performance Evaluation



Conclusions and future works

- Preliminary results are promising
  - Efficient
  - Requires no knowledge of the application internals
  - Requires no modifications of the application internals
- TODO
  - Is SAWYER optimal?
  - Evaluate SAWYER on a real application
  - Is it possible to implement a fully decentralized SAWYER?

The answer is “yes, mostly”