On the Use of Online Performance Models in Autonomic Computing

Prof. Daniel Menascé
Department of Computer Science
E-Center for E-Business
George Mason University
Fairfax, VA, USA
Menasce@cs.gmu.edu
www.cs.gmu.edu/faculty/menasce.html

Defining Features of Autonomic Computing Systems

(www.research.ibm.com/autonomic/overview/elements.html)

1. Self-aware
2. Self-configurable and self-reconfigurable
3. Self-optimizing
4. Self-healing
5. Self-protecting
6. Adaptable to the environment
7. Use open standards
8. Transparent use of IT resources to achieve business/personal user goals.

Defining Features of Autonomic Computing Systems

(www.research.ibm.com/autonomic/overview/elements.html)

1. Self-aware
2. Self-configurable and self-reconfigurable
3. Self-optimizing
4. Self-healing
5. Self-protecting
6. Adaptable to the environment
7. Use open standards
8. Transparent use of IT resources to achieve business/personal user goals.

Rest of this talk …

- Novel uses for performance models
- Two examples of self-regulating systems:
  - A three-tiered e-commerce system
  - QoS-aware software components
- Concluding Remarks

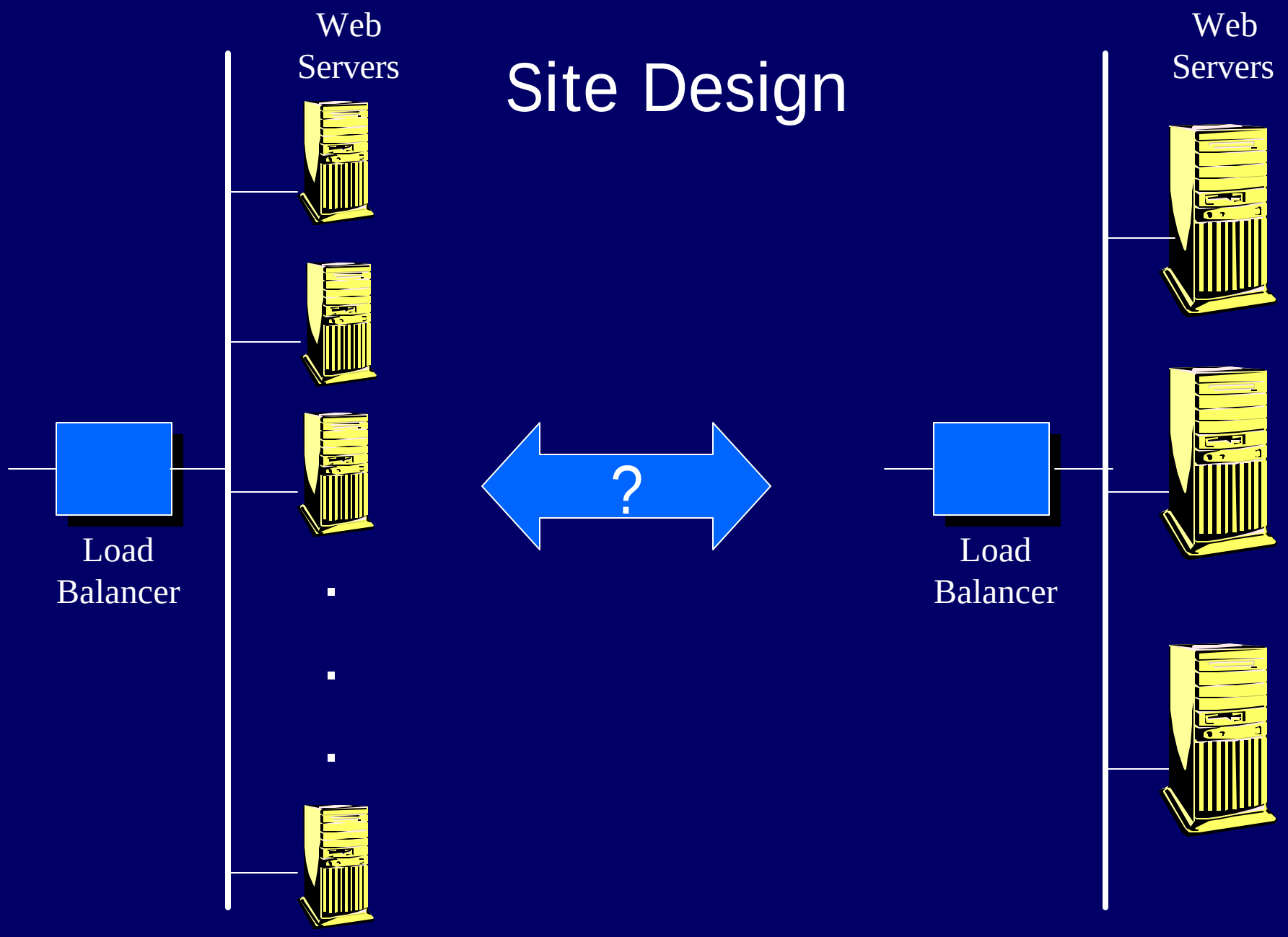
Rest of this talk …

- Novel uses for performance models
- Two examples of self-regulating systems:
  - A three-tiered e-commerce system
  - QoS-aware software components
- Concluding Remarks

What are performance models good for?

- At the design stage:
  - Compare competing design alternatives.
    - A large number of low capacity servers vs. a small number of large capacity servers?

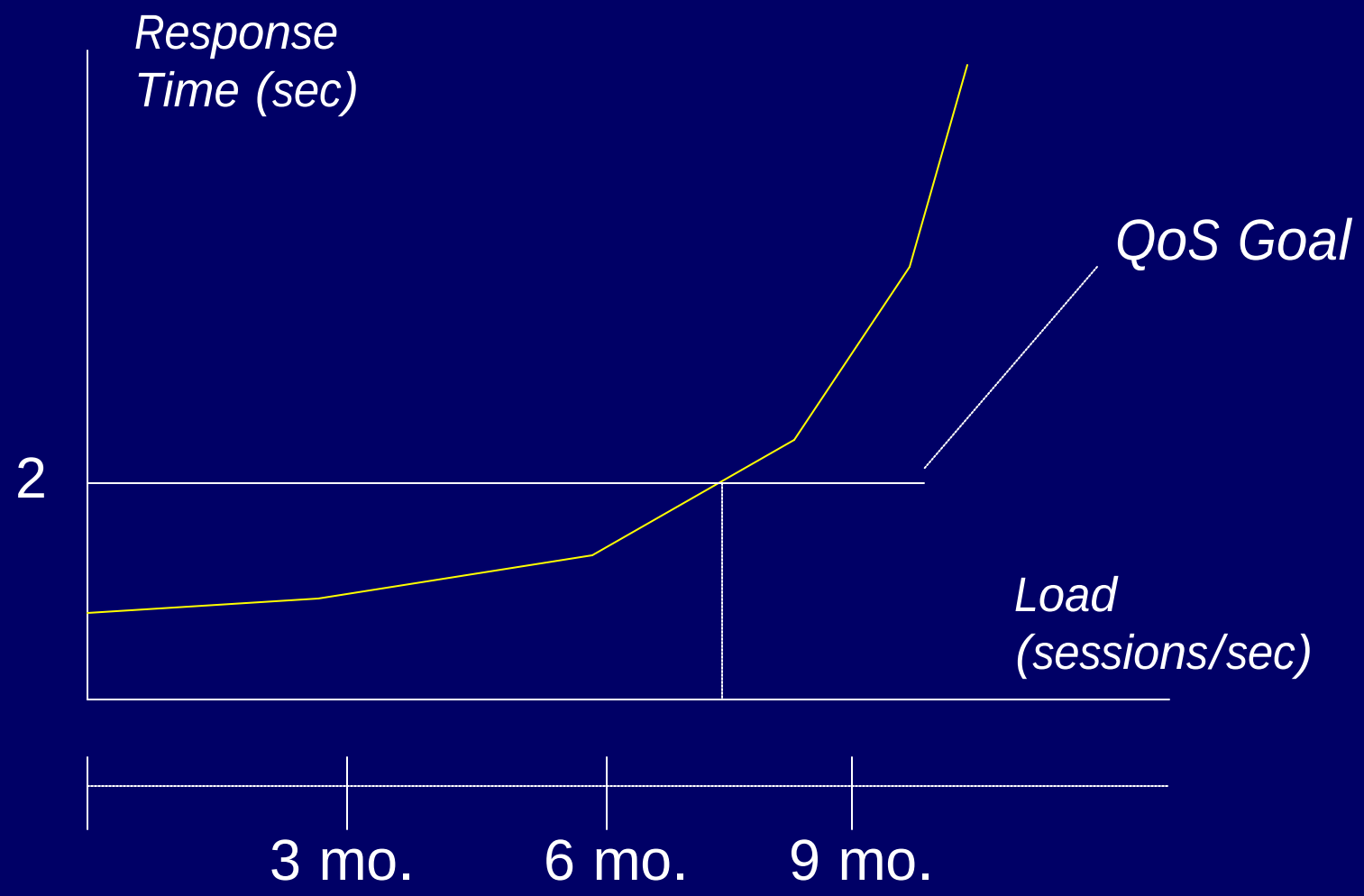
Site Design



What are performance models good for?

- At the design stage:
  - Compare competing design alternatives.
    - A large number of low capacity servers vs. a small number of large capacity servers?
- During production:
  - Medium and long-term (weeks and months):
    - Capacity planning.

Capacity Planning



What are performance models good for?

- At the design stage:
  - Compare competing design alternatives.
    - A large number of low capacity servers vs. a small number of large capacity servers?
- During production:
  - Medium and long-term (weeks and months):
    - Capacity planning.
  - Short-term (minutes):
    - Dynamic reconfiguration.

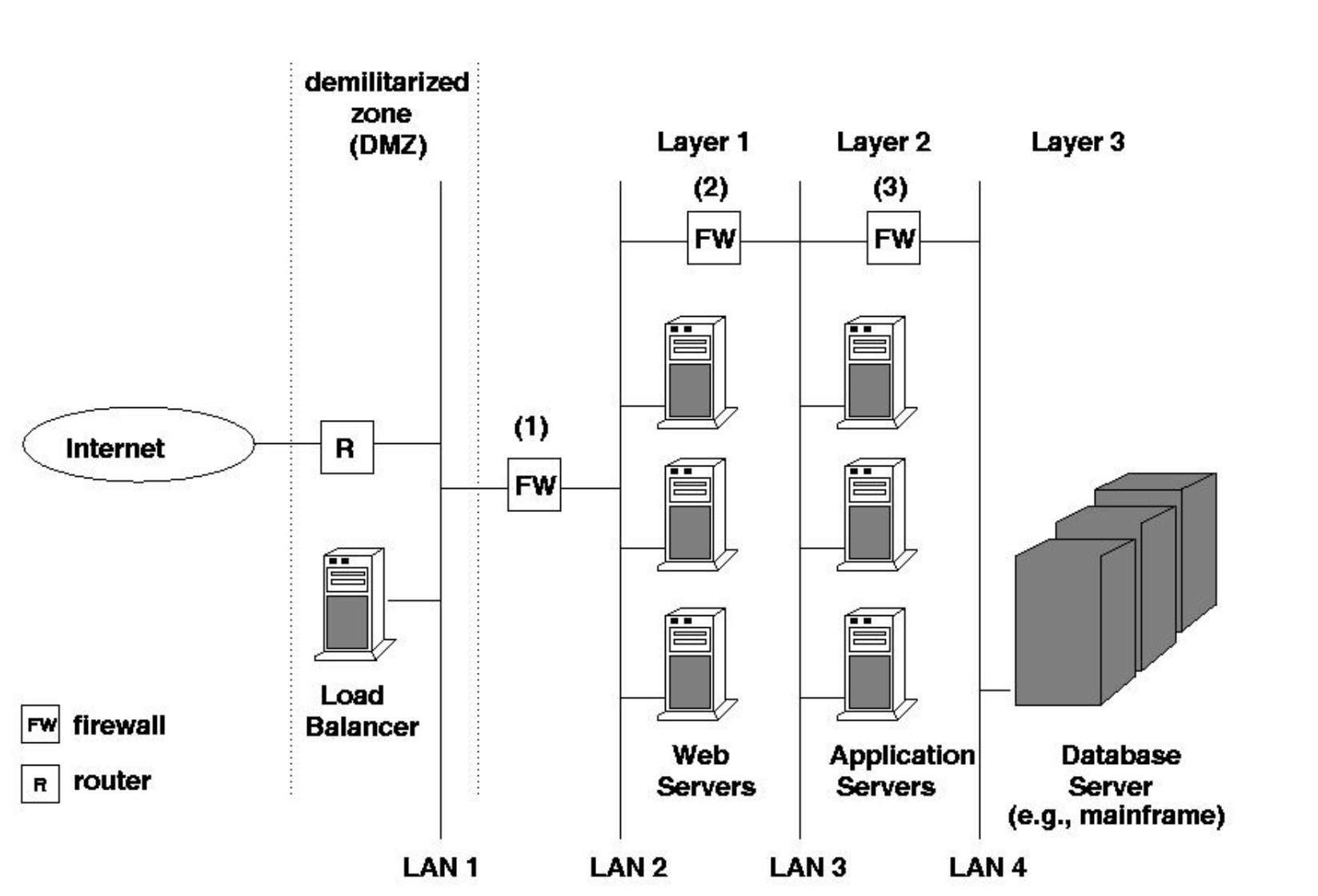
Rest of this talk …

- Novel uses for performance models
- Two examples of self-regulating systems:
  - A three-tiered e-commerce system
  - QoS-aware software components

Automatic QoS Control: Motivation

- Modern computer systems are complex and composed of multiple tiers.

Multi-tier Architecture



Automatic QoS Control: Motivation

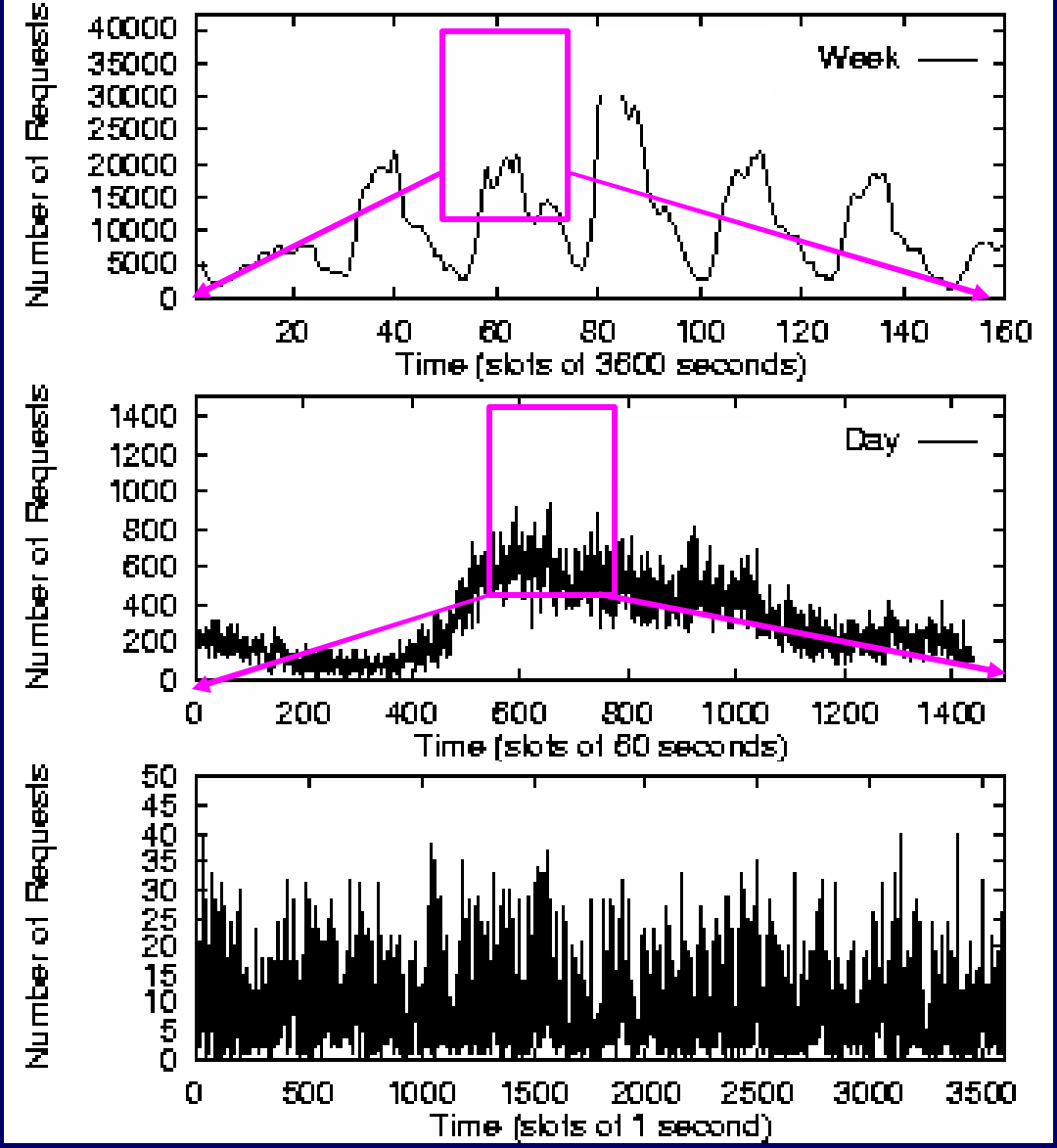
- Modern computer systems are complex and composed of multiple tiers.
- The workload presents short-term variations with high peak-to-average ratios.

Multi-scale time workload variation

3600 sec

60 sec

1 sec

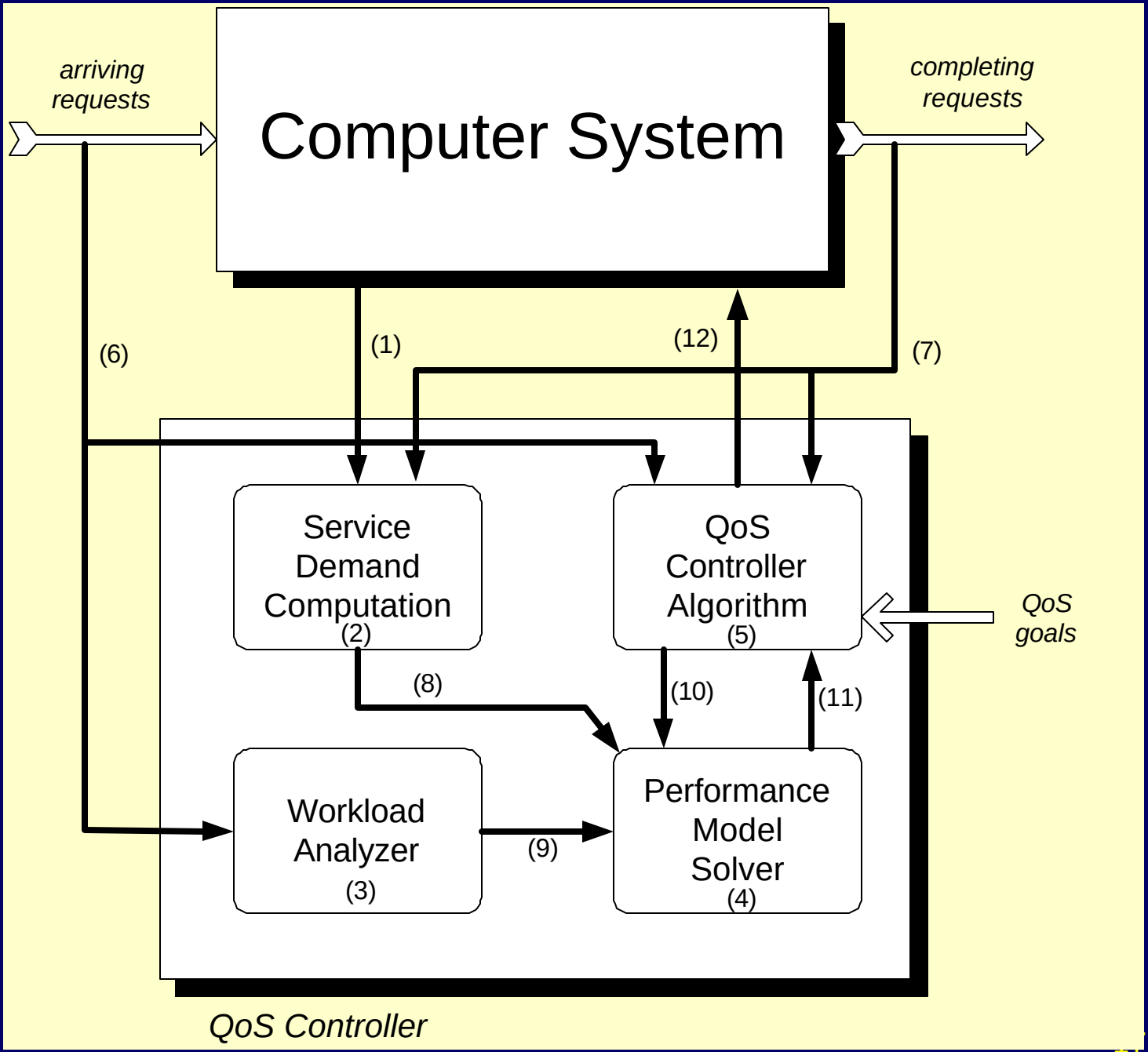


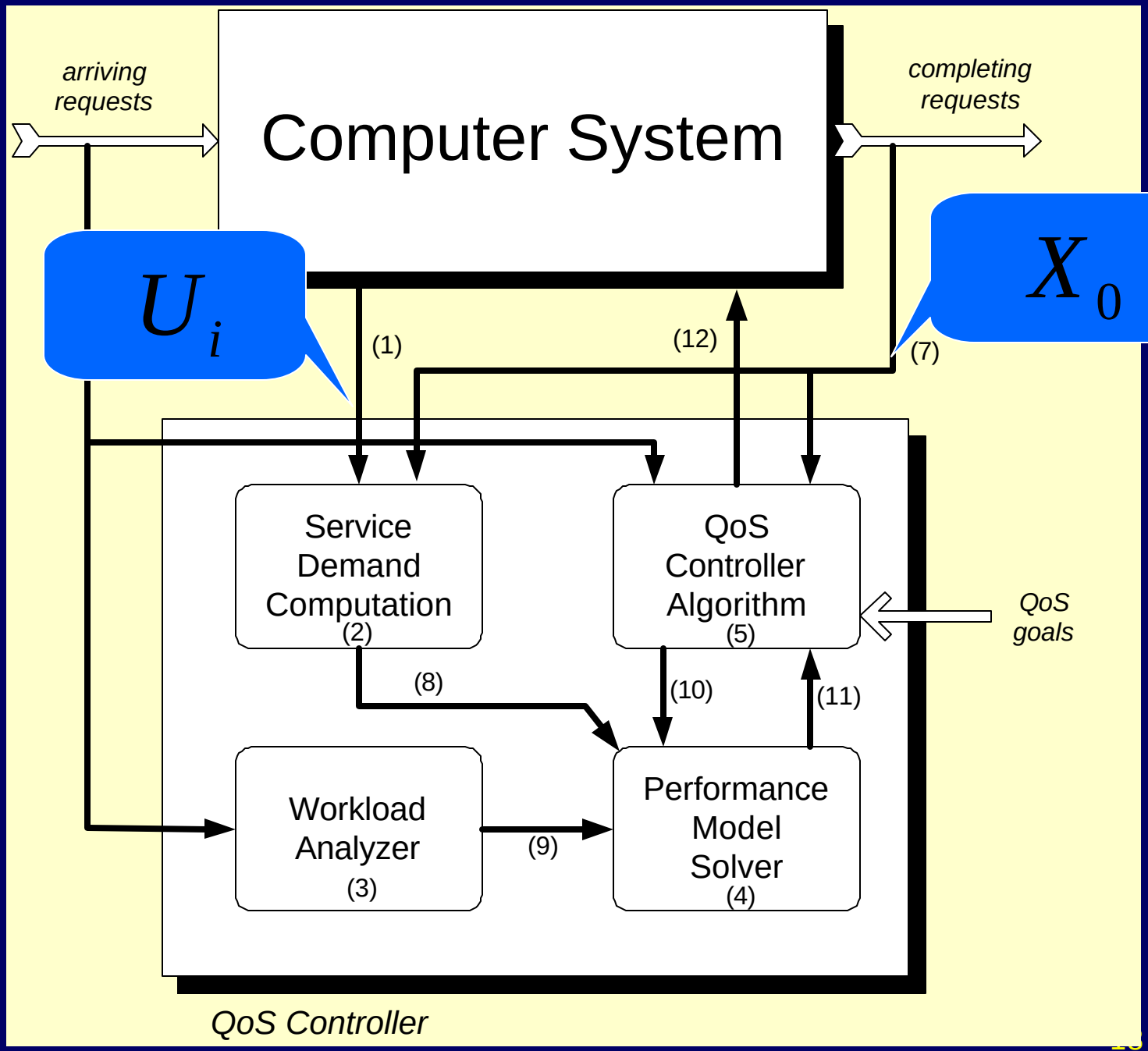
Automatic QoS Control: Motivation

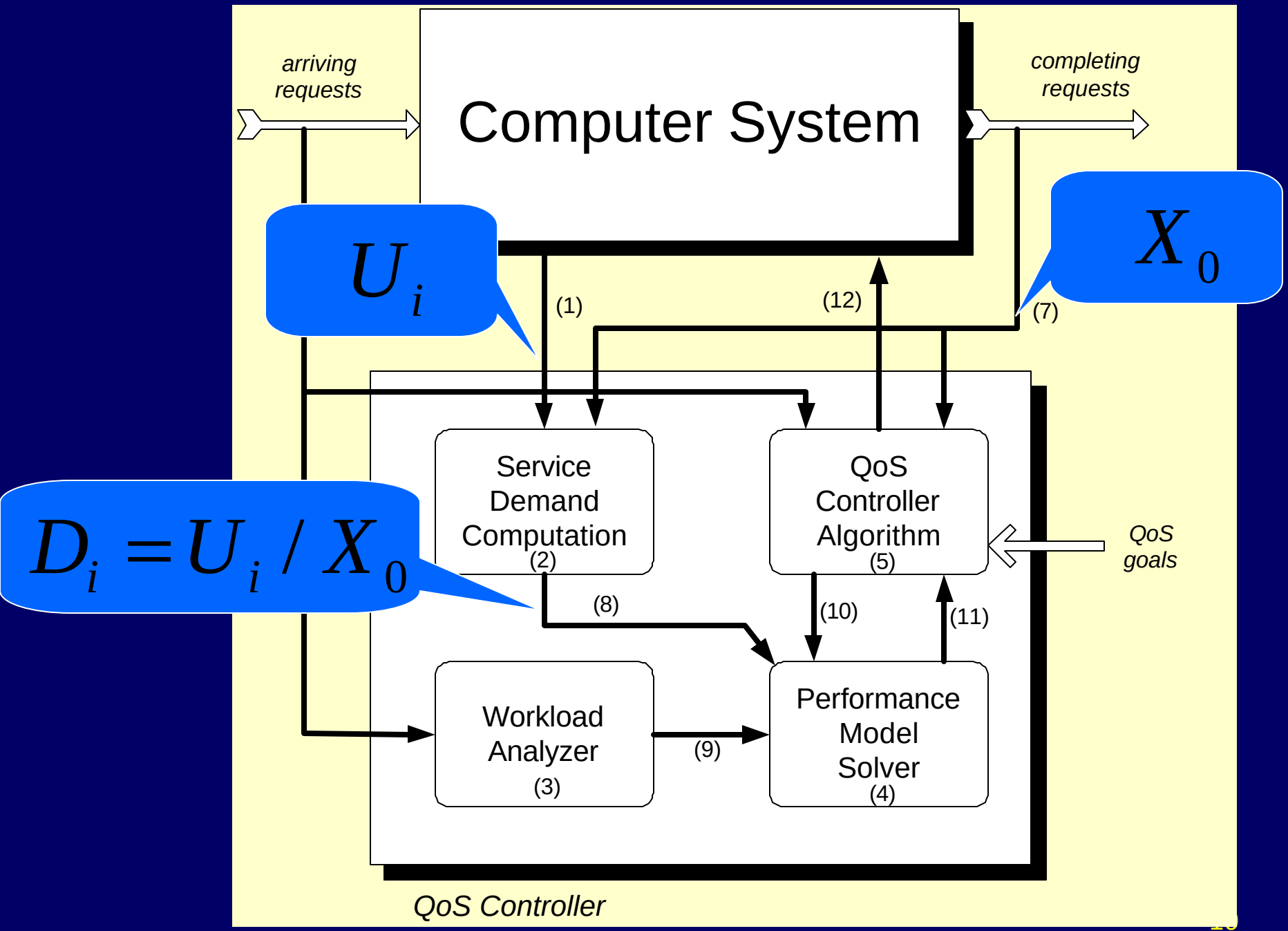
- Modern computer systems are complex and composed of multiple tiers.
- The workload presents short-term variations with high peak-to-average ratios.
- Many software and hardware parameters influence the performance of e-commerce sites.

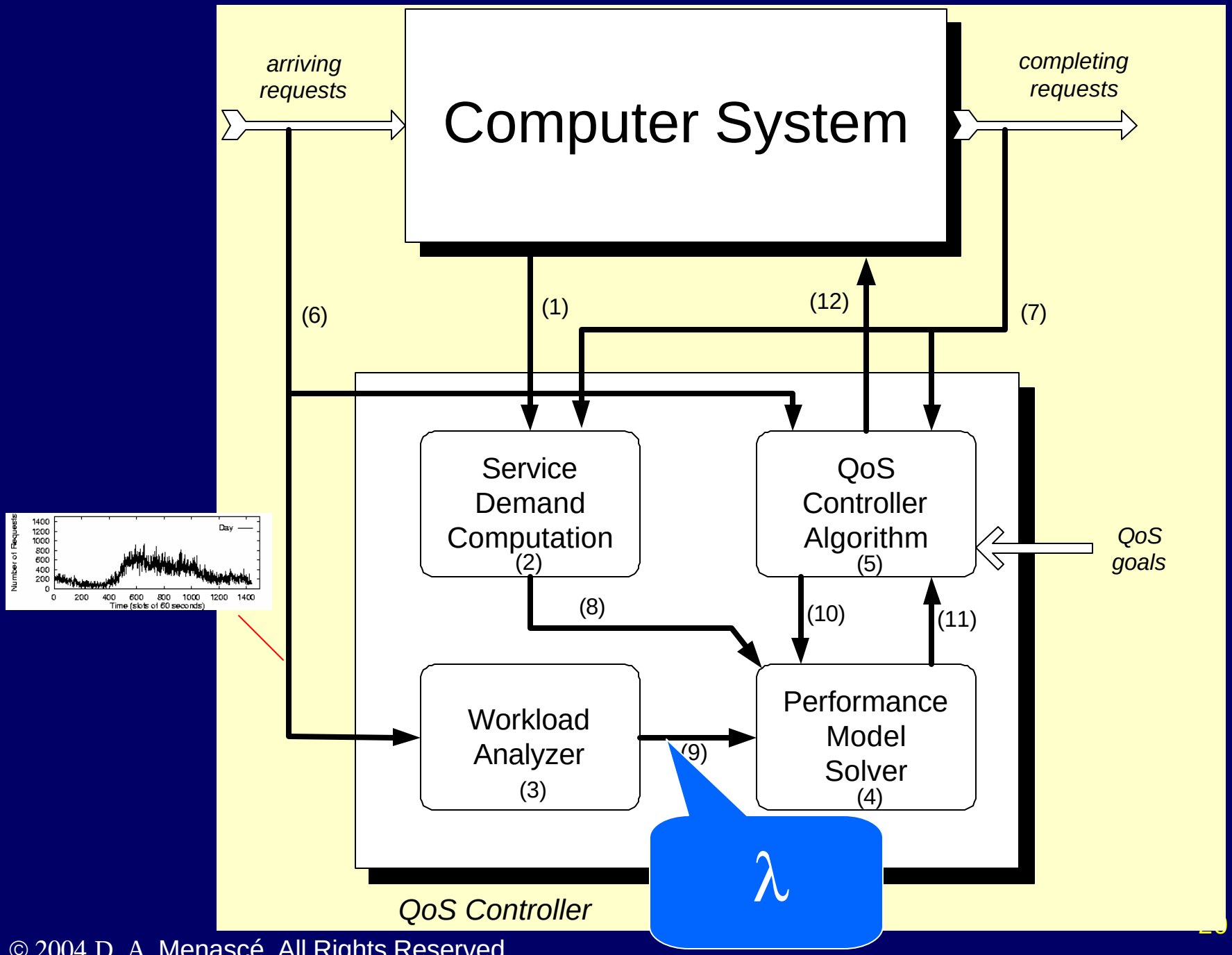
→ Manual reconfiguration is not an option!

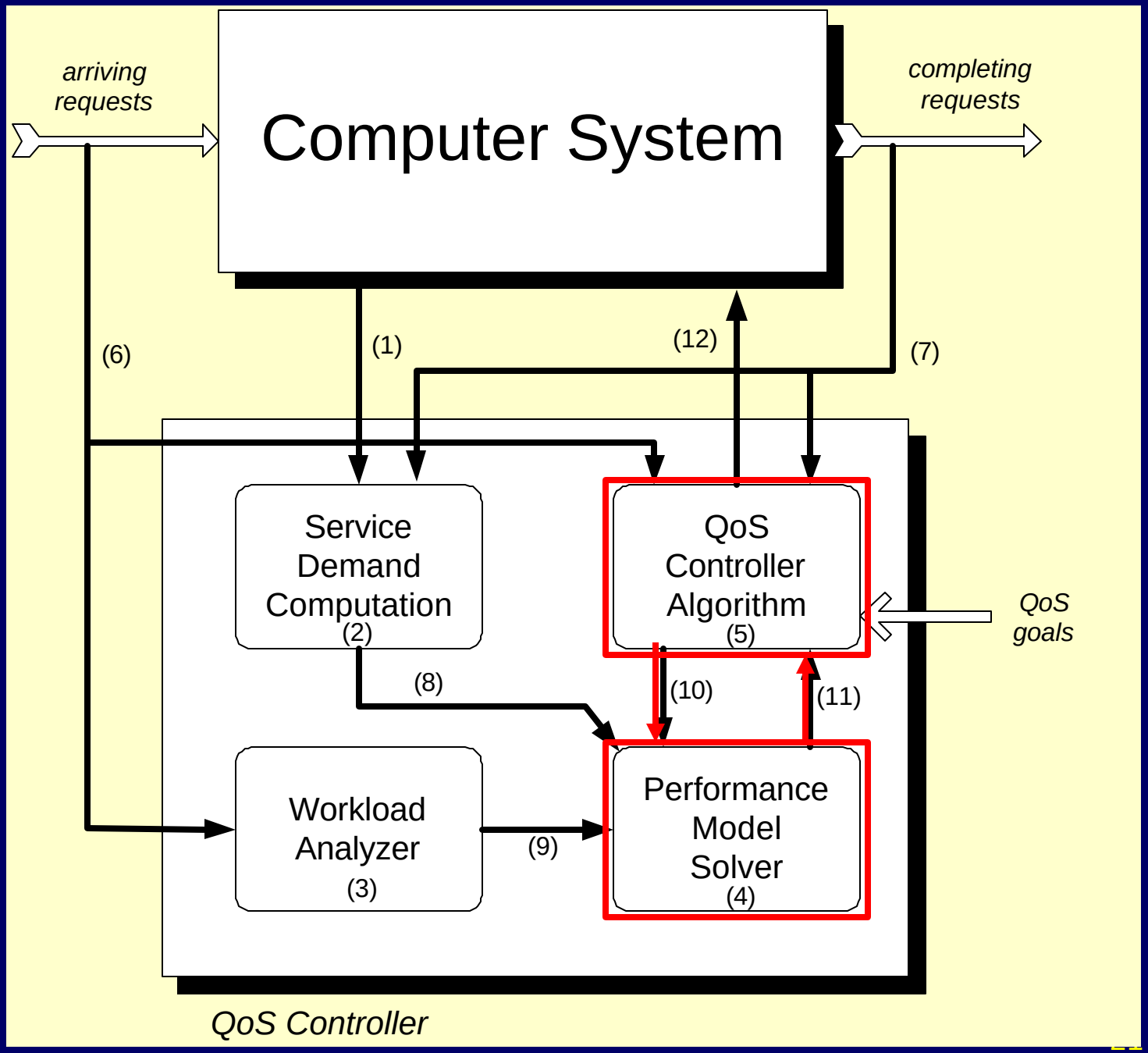
Need self-managing systems.



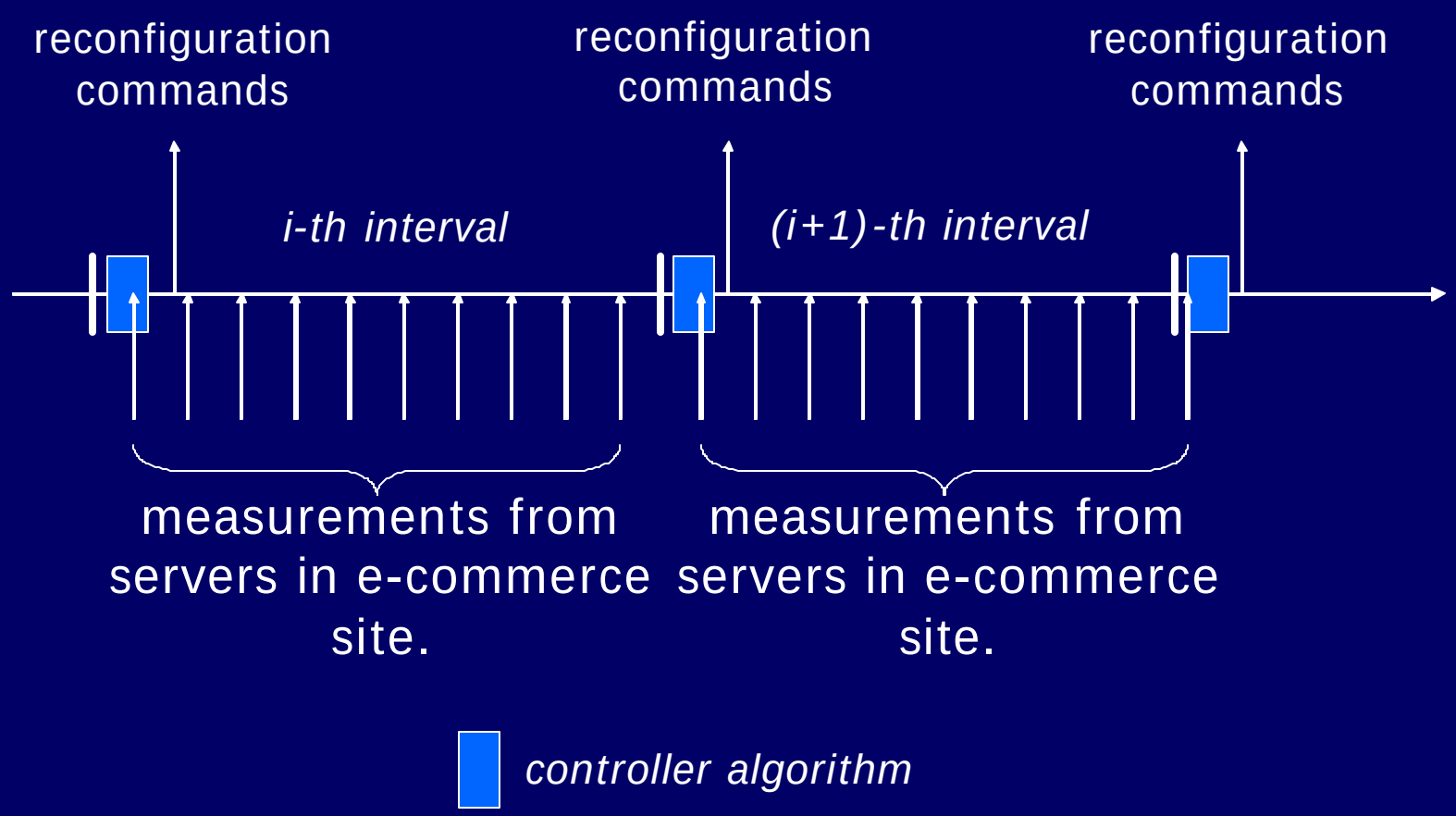








Controller Interval



Combined QoS Metric

$$QoS = w_R \times \Delta QoS_R + w_P \times \Delta QoS_P + w_X \times \Delta QoS_X$$

w_R, w_P, and w_X are relative weights that indicate the relative importance of response time, throughput, and probability of rejection.

QoS Metric

$$QoS = w_R \times \Delta QoS_R + w_P \times \Delta QoS_P + w_X \times \Delta QoS_X$$

w_R, w_P, and w_X are relative weights that indicate the relative importance of response time, throughput, and probability of rejection.

ΔQoS_R, ΔQoS_P, and ΔQoS_X are relative deviations of the response time, throughput, and probability of rejection metrics with respect to their desired levels.

QoS Metric

$$QoS = w_R \times \Delta QoS_R + w_P \times \Delta QoS_P + w_X \times \Delta QoS_X$$

w_R, w_P, and w_X are relative weights that indicate the relative importance of response time, throughput, and probability of rejection.

ΔQoS_R, ΔQoS_P, and ΔQoS_X are relative deviations of the response time, throughput, and probability of rejection metrics with respect to their desired levels.

➡ The QoS metric is a dimensionless number in the interval [-1, 1].

QoS Metric

$$QoS = w_R \times \Delta QoS_R + w_P \times \Delta QoS_P + w_X \times \Delta QoS_X$$

w_R, w_P, and w_X are relative weights that indicate the relative importance of response time, throughput, and probability of rejection.

ΔQoS_R, ΔQoS_P, and ΔQoS_X are relative deviations of the response time, throughput, and probability of rejection metrics with respect to their desired levels.

The QoS metric is a dimensionless number in the interval [-1, 1].

If all metrics meet or exceed their QoS targets, QoS = 0.

Response Time Deviation

$$\Delta QoS_R = \frac{R_{\max} - R_{measured}}{\max(R_{\max}, R_{measured})}$$

- = 0 if the response time meets its target.
- > 0 if the response time exceeds its target.
- < 0 if the response time does not meet its target.

- $$\Delta QoS_R \le 1 - (\sum_{i=1}^{K} D_i) / R_{\max} < 1$$
- $$-1 < -(1 - R_{\max} / R_{measured}) \le \Delta QoS_R$$

Probability of Rejection Deviation

$$\Delta QoS_P = \frac{P_{\max} - P_{measured}}{\max(P_{\max}, P_{measured})}$$

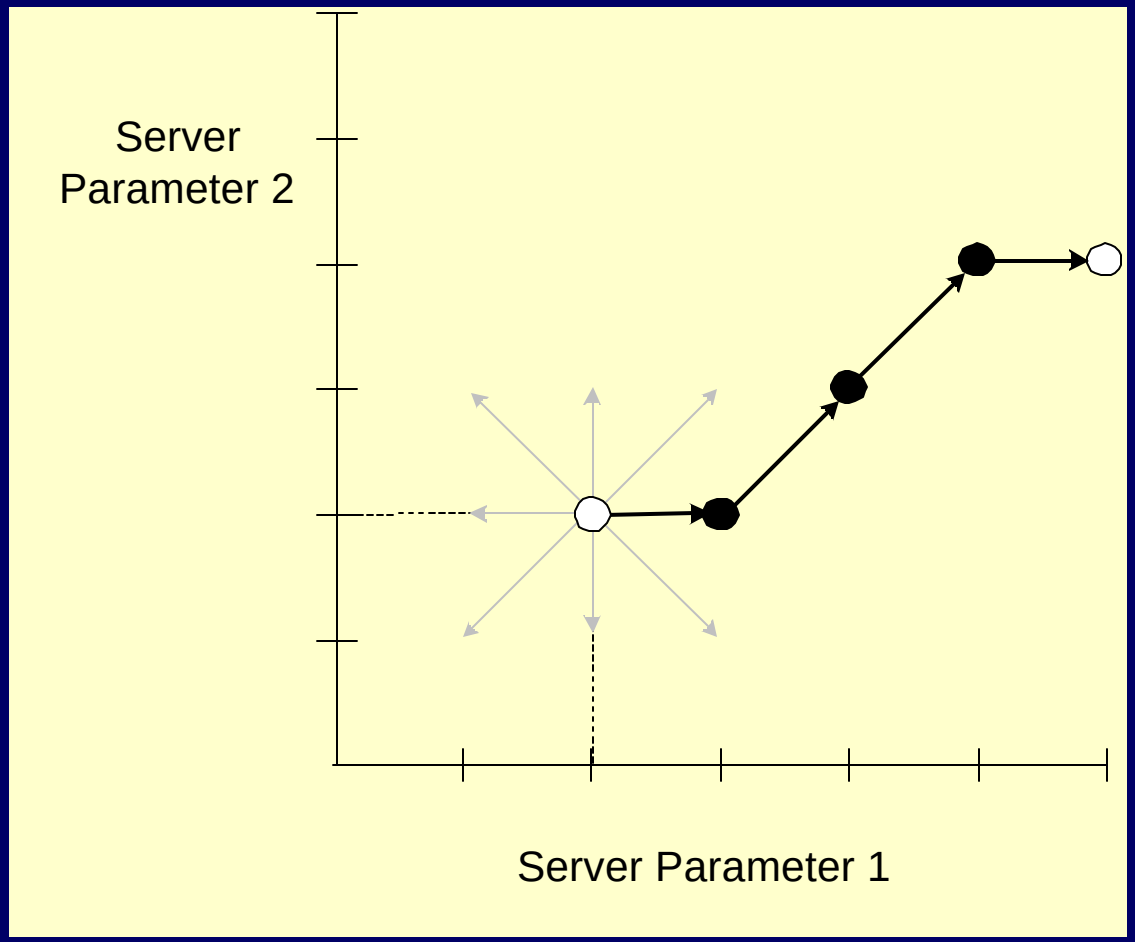
- = 0 if the probability of rejection meets its target.
- > 0 and = 1 if the probability of rejection exceeds its target.
- < 0 and = -1 if the probability of rejection does not meet its target.

Throughput Deviation

$$\Delta QoS_X = \frac{X_{measured} - X^*_{\min}}{\max(X_{measured}, X^*_{\min})}$$

- $X^*_{\min} = \min(l, X_{\min})$
- = 0 if the throughput meets its target.
- > 0 and < 1 if the throughput exceeds its target.
- < 0 and > -1 if the throughput does not meet its target.

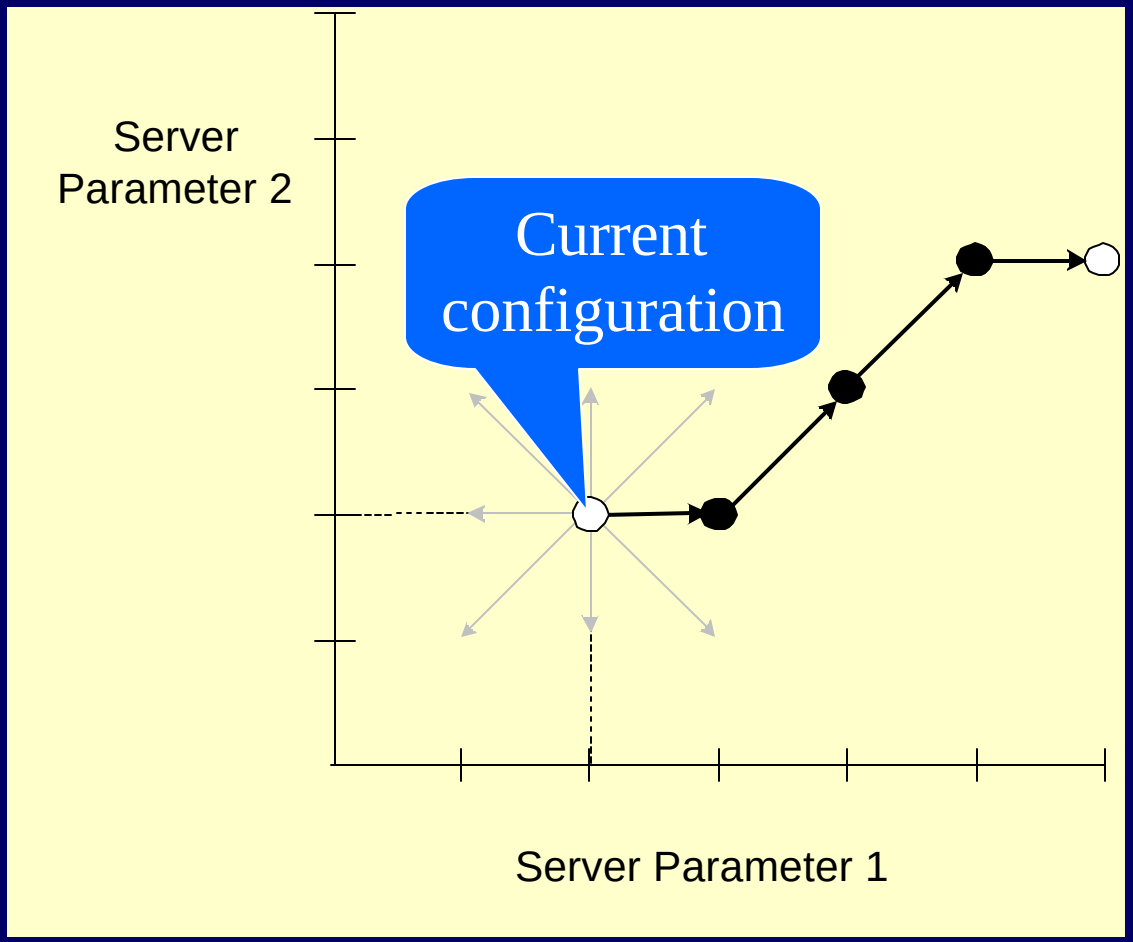
Heuristic Optimization Approach



- The space of configuration points is searched using a combinatorial search technique.
- Each point has a QoS value computed through an analytic performance model.

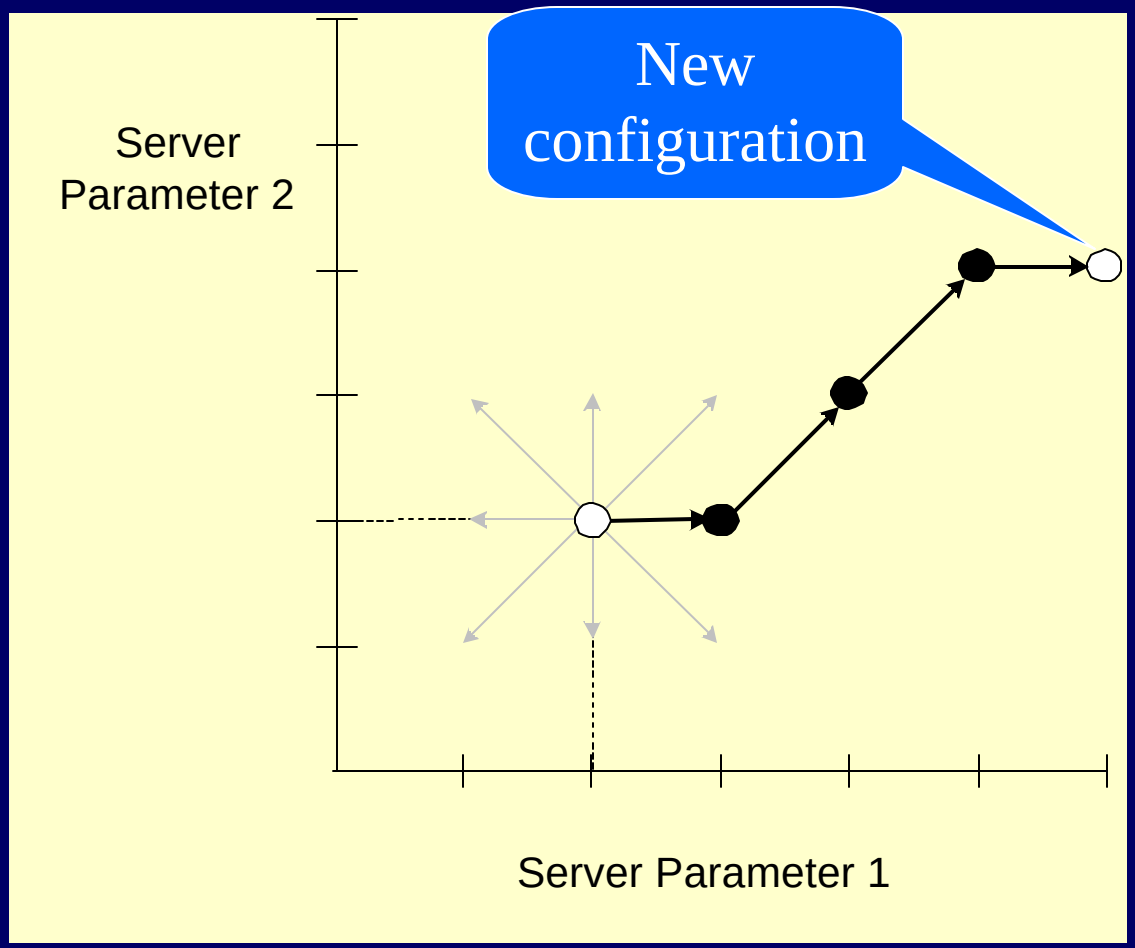
$$QoS = f(\vec{W}, c_1, c_2, \ldots, c_m)$$

Heuristic Optimization Approach



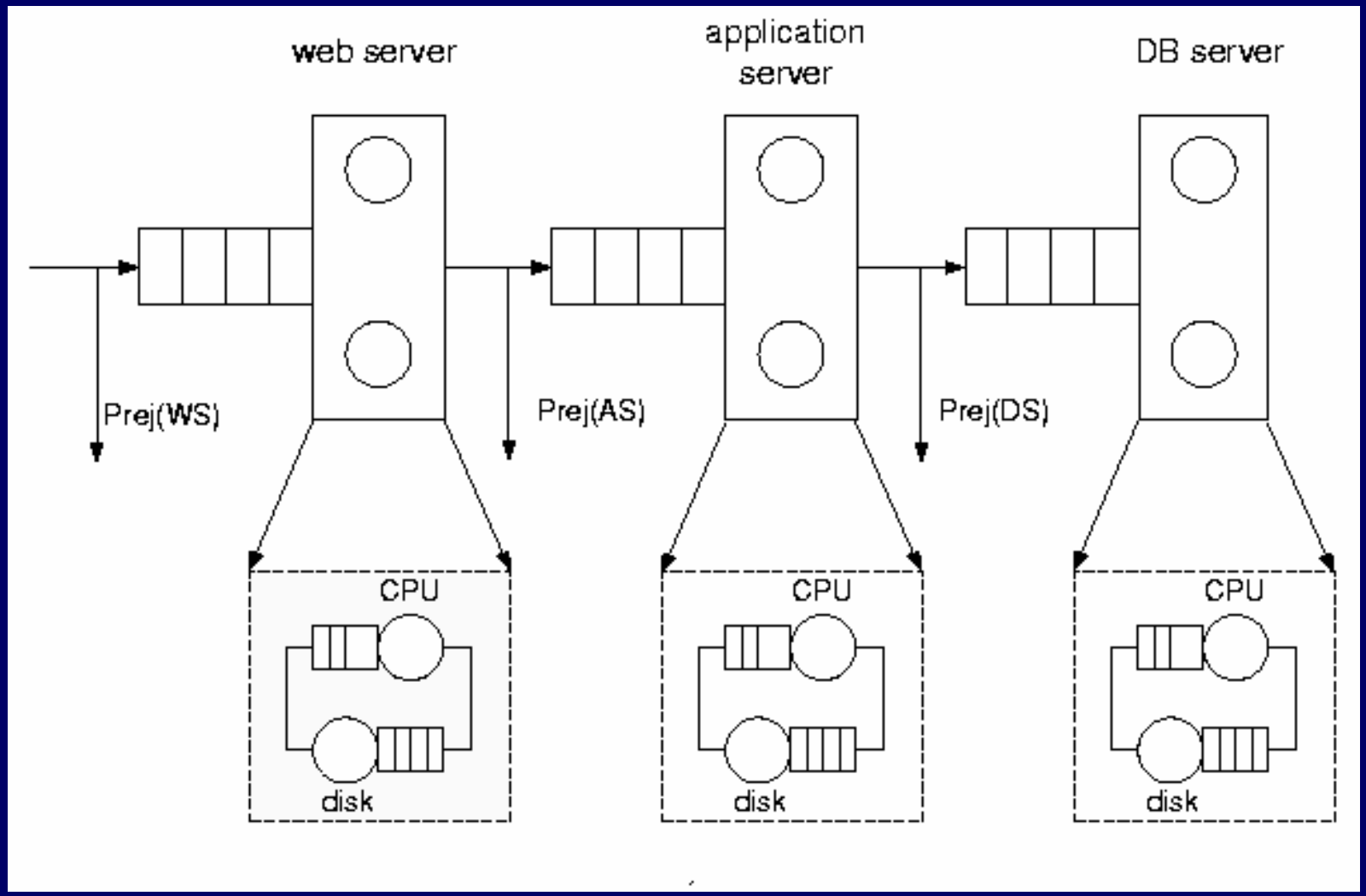
- The space of configuration points is searched using combinatorial search techniques.
- Each point has a QoS value computed through an analytic performance model.

Heuristic Optimization Approach

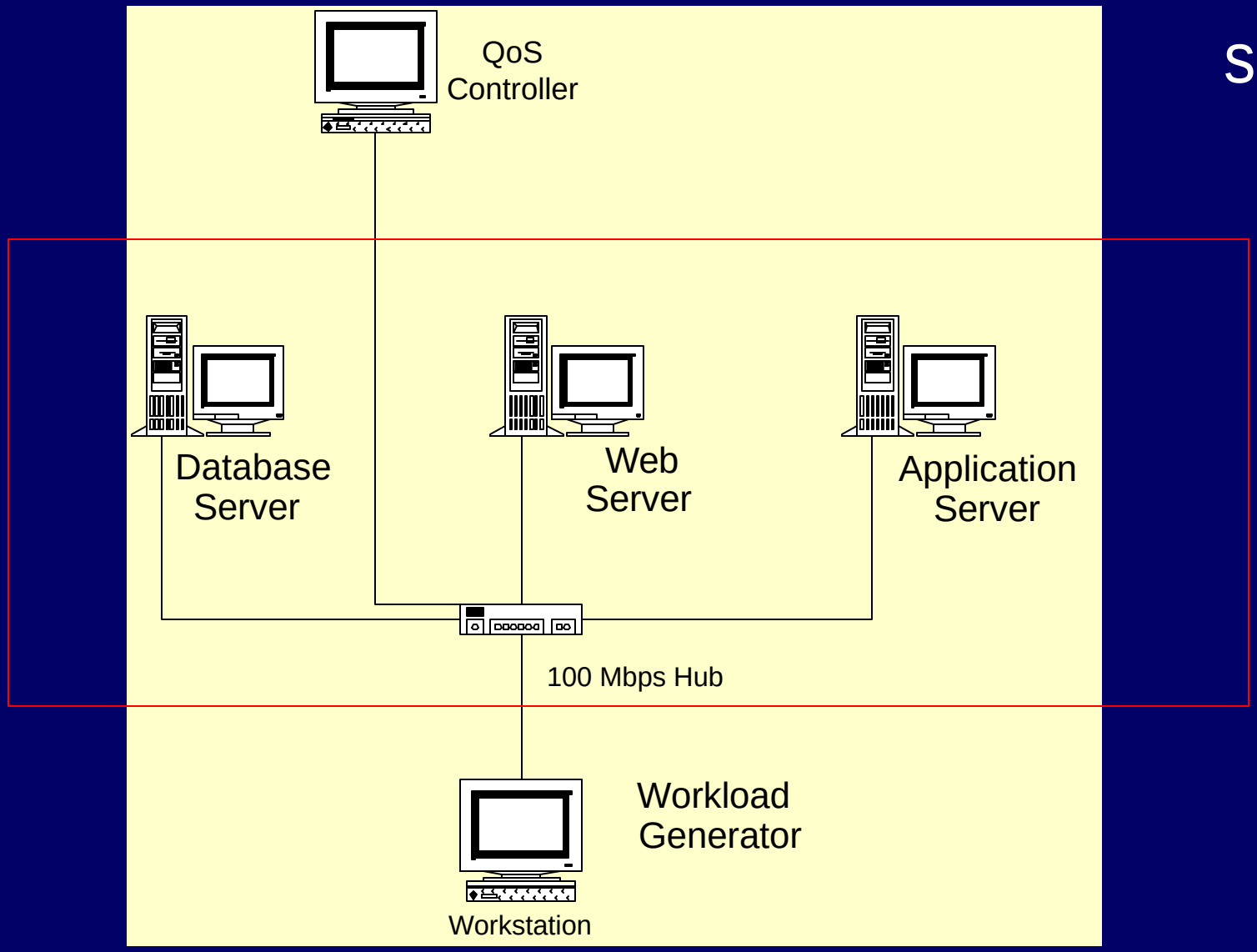


- The space of configuration points is searched using combinatorial search techniques.
- Each point has a QoS value computed through an analytic performance model.

A Queuing Model is Used to Compute QoS Values

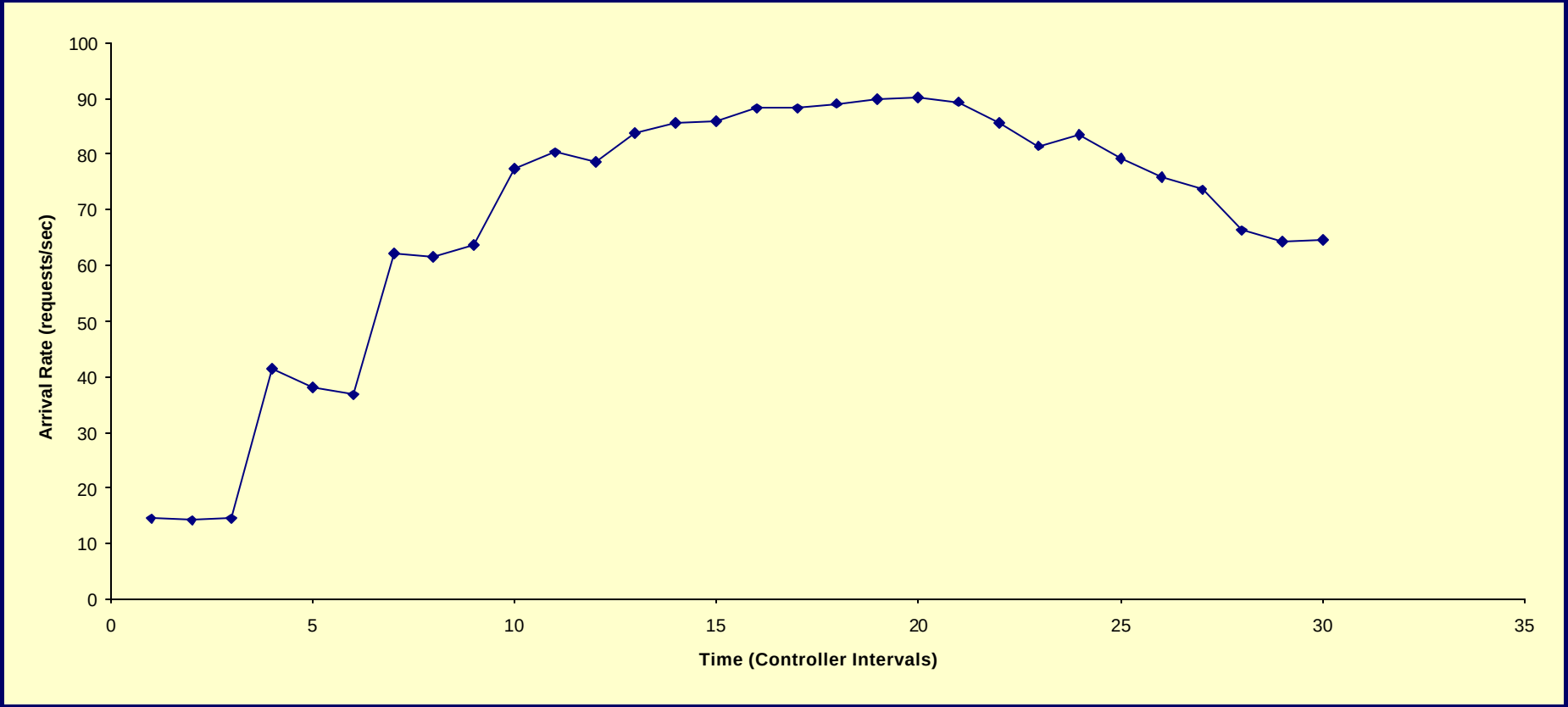


Prototype Configuration

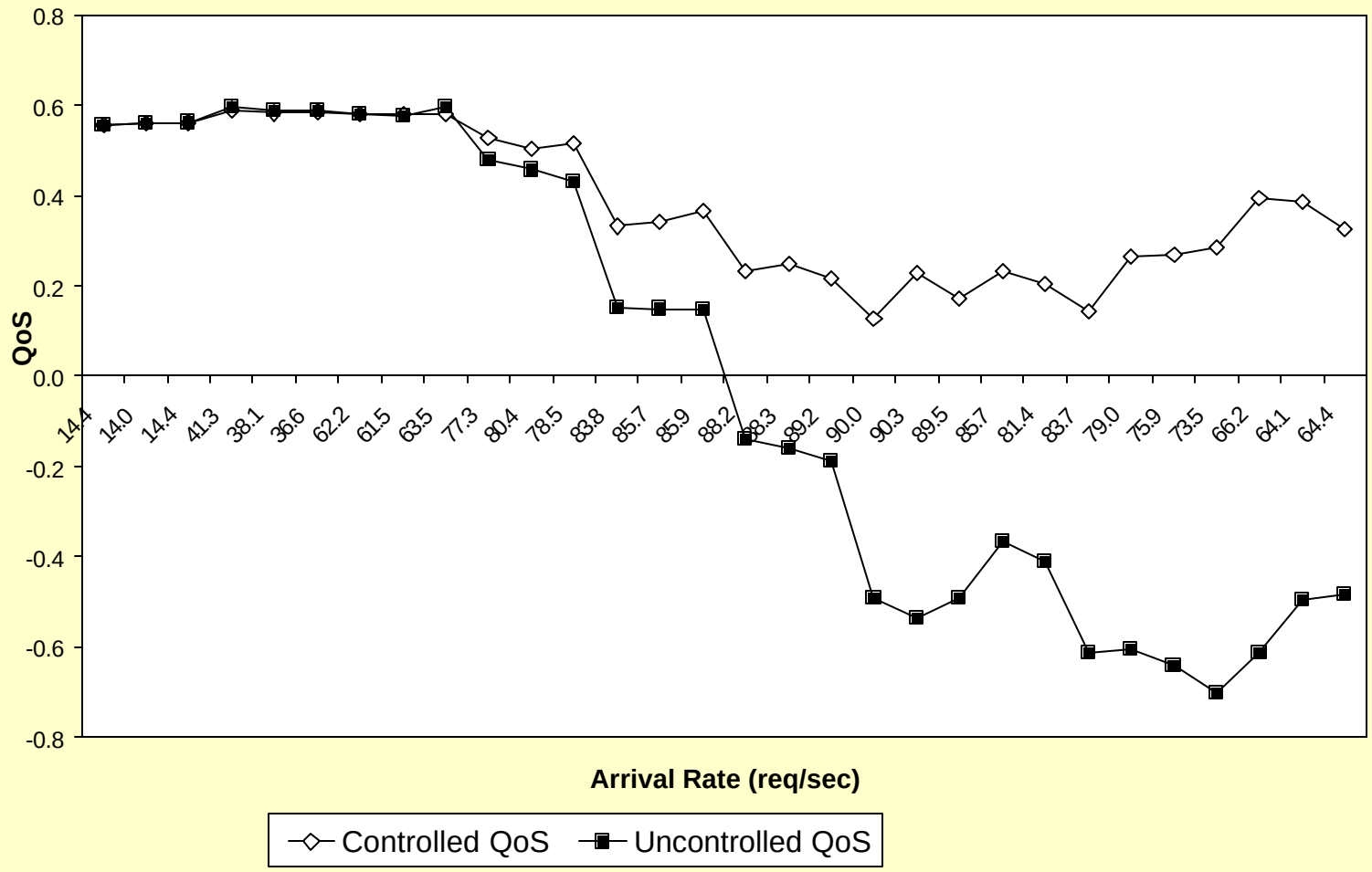


Experiment Results

Arrival rate

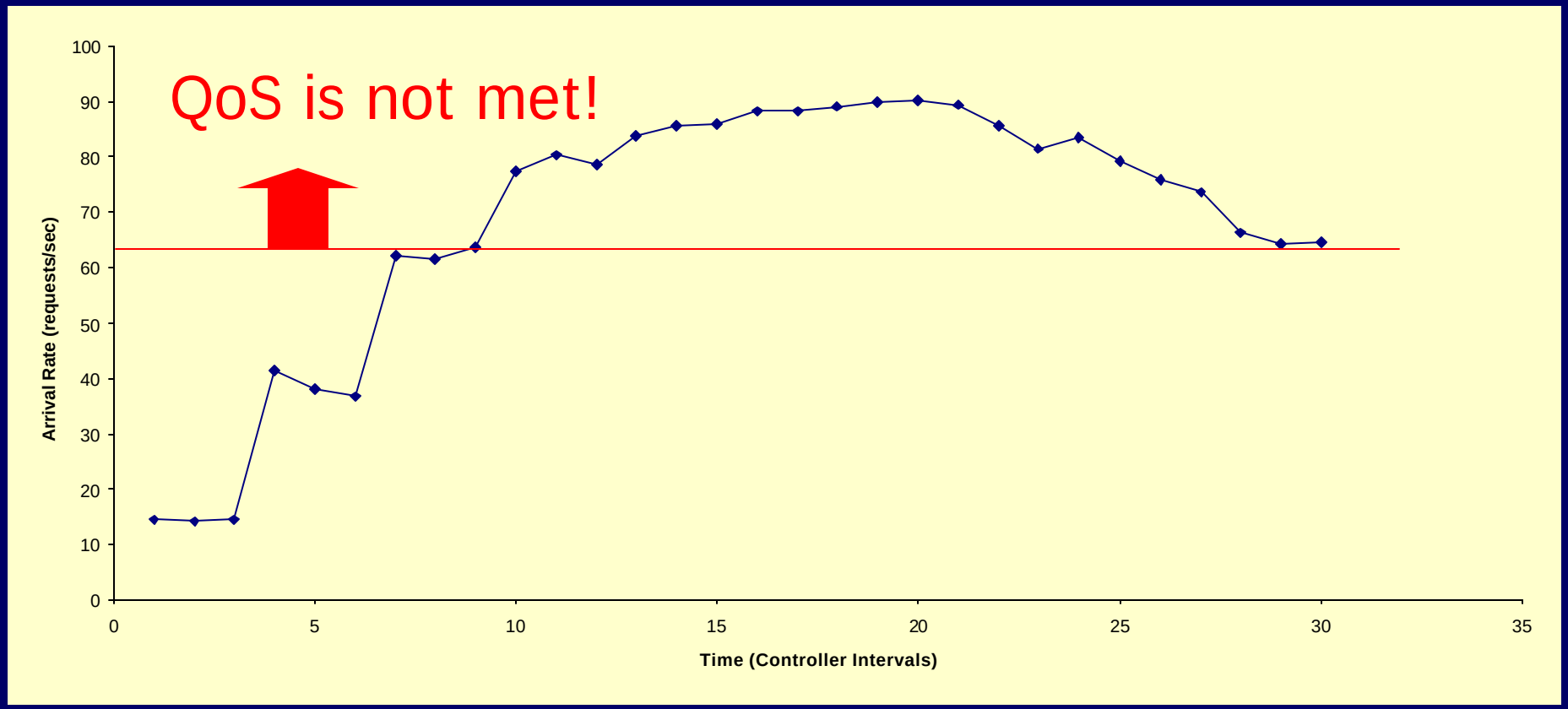


Results of QoS Controller



Experiment Results

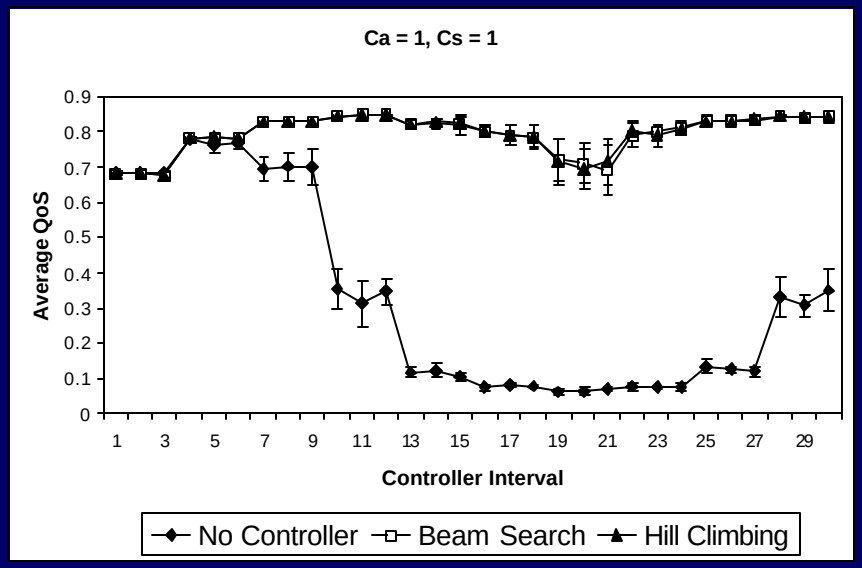
Arrival rate



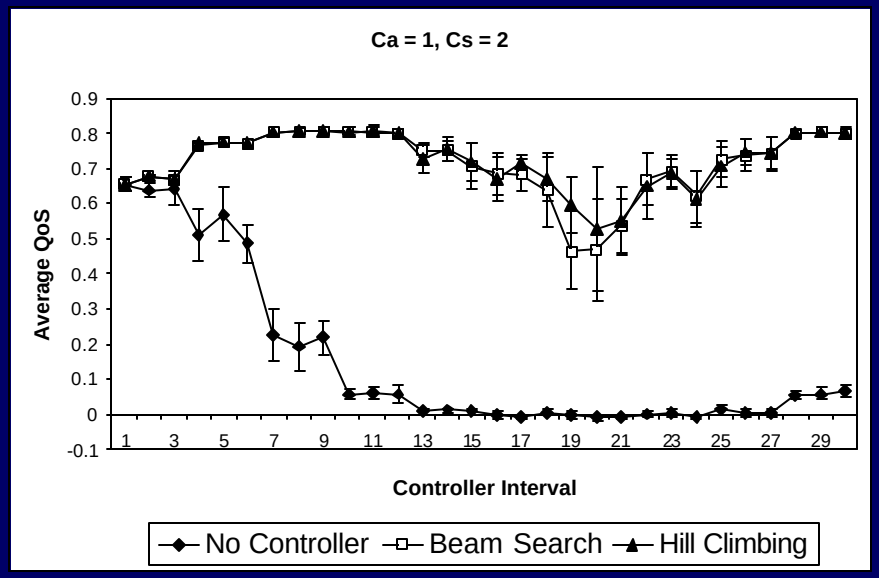
Variable interarrival and service times of requests

- Real workloads exhibit high variability in:
  - Traffic intensity
  - Service demands at various system resources
- Need to investigate the efficiency of the proposed self-managing technique under these conditions
- Consider variability in requests inter-arrival time and requests service times at physical resources (e.g., CPU, disk)

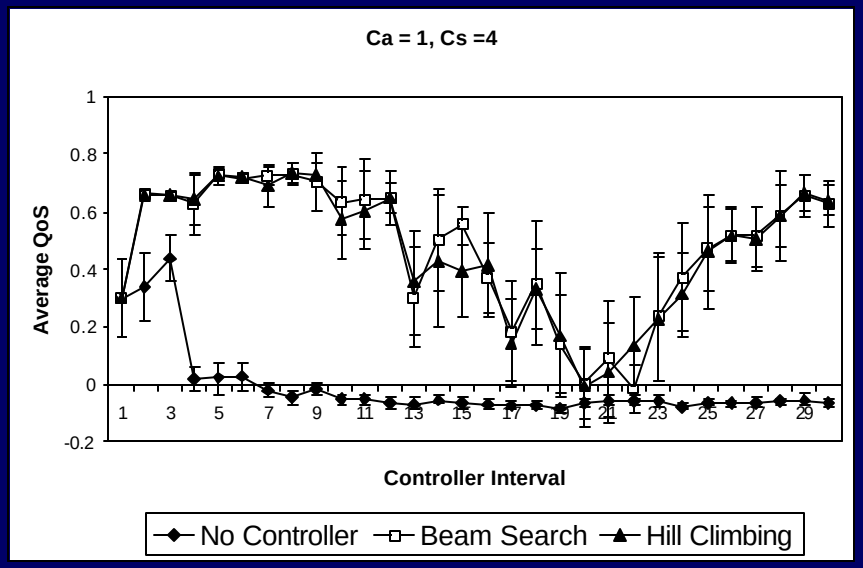
Effect of Varying the COV of the Service Time (Ca = 1)



Cs = 1

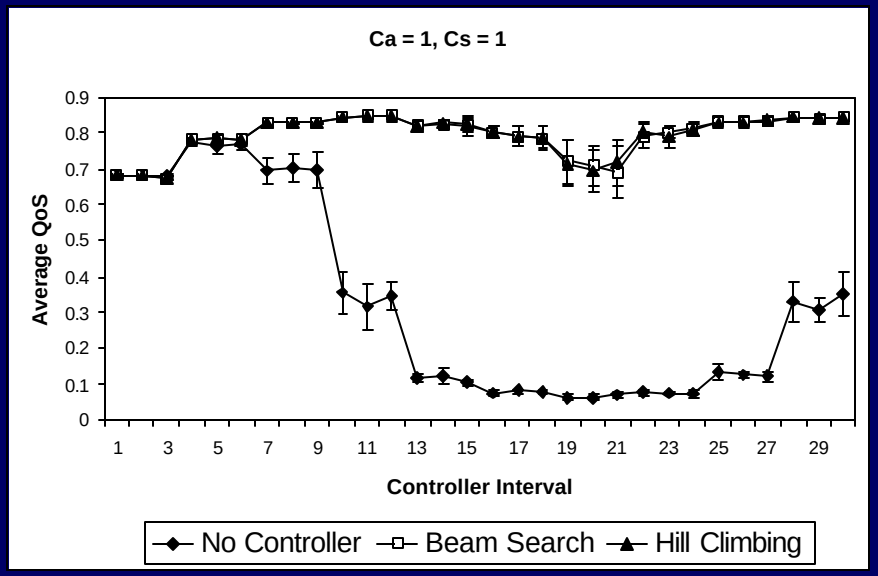


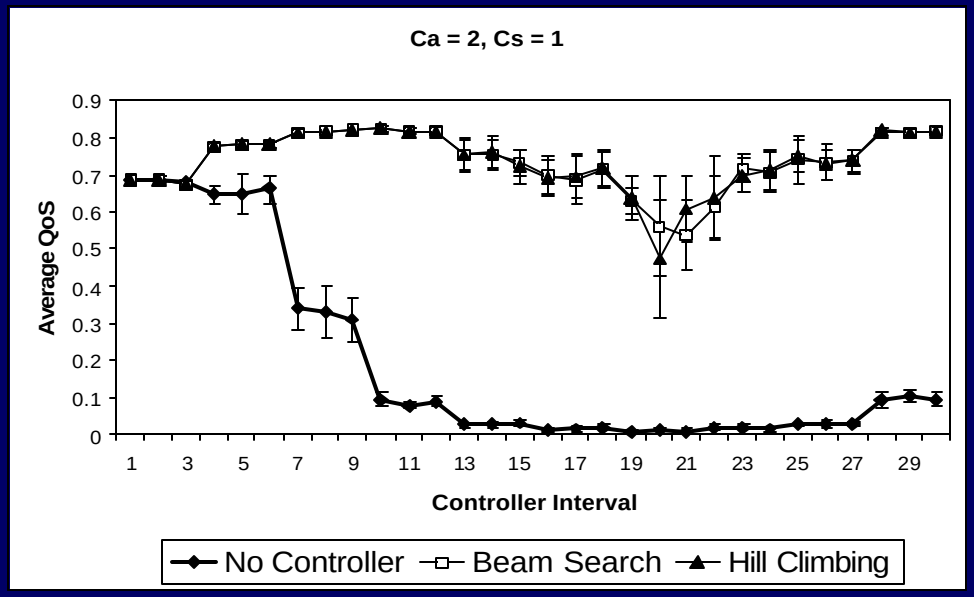
Cs = 2



Cs = 4

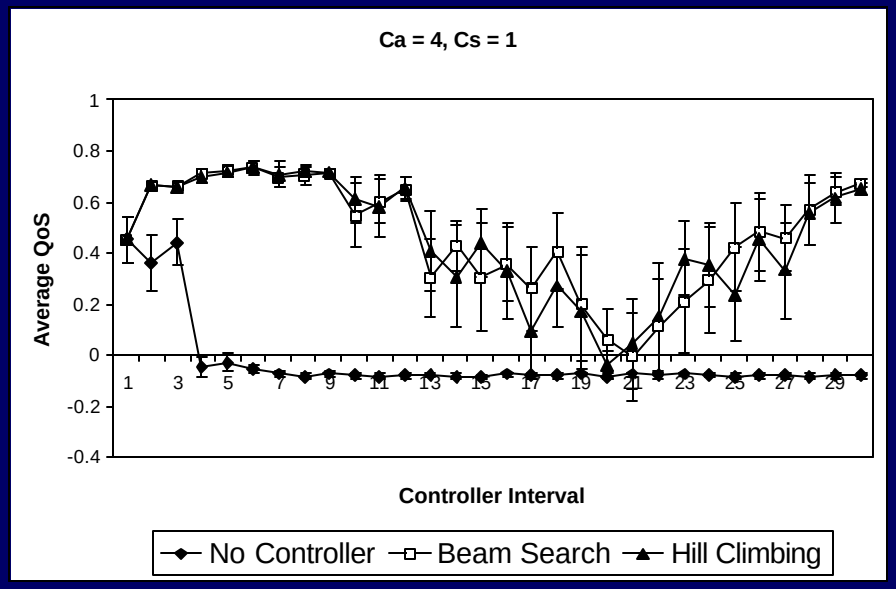
Effect of Varying the COV of the Interarrival Time (Cs = 1)





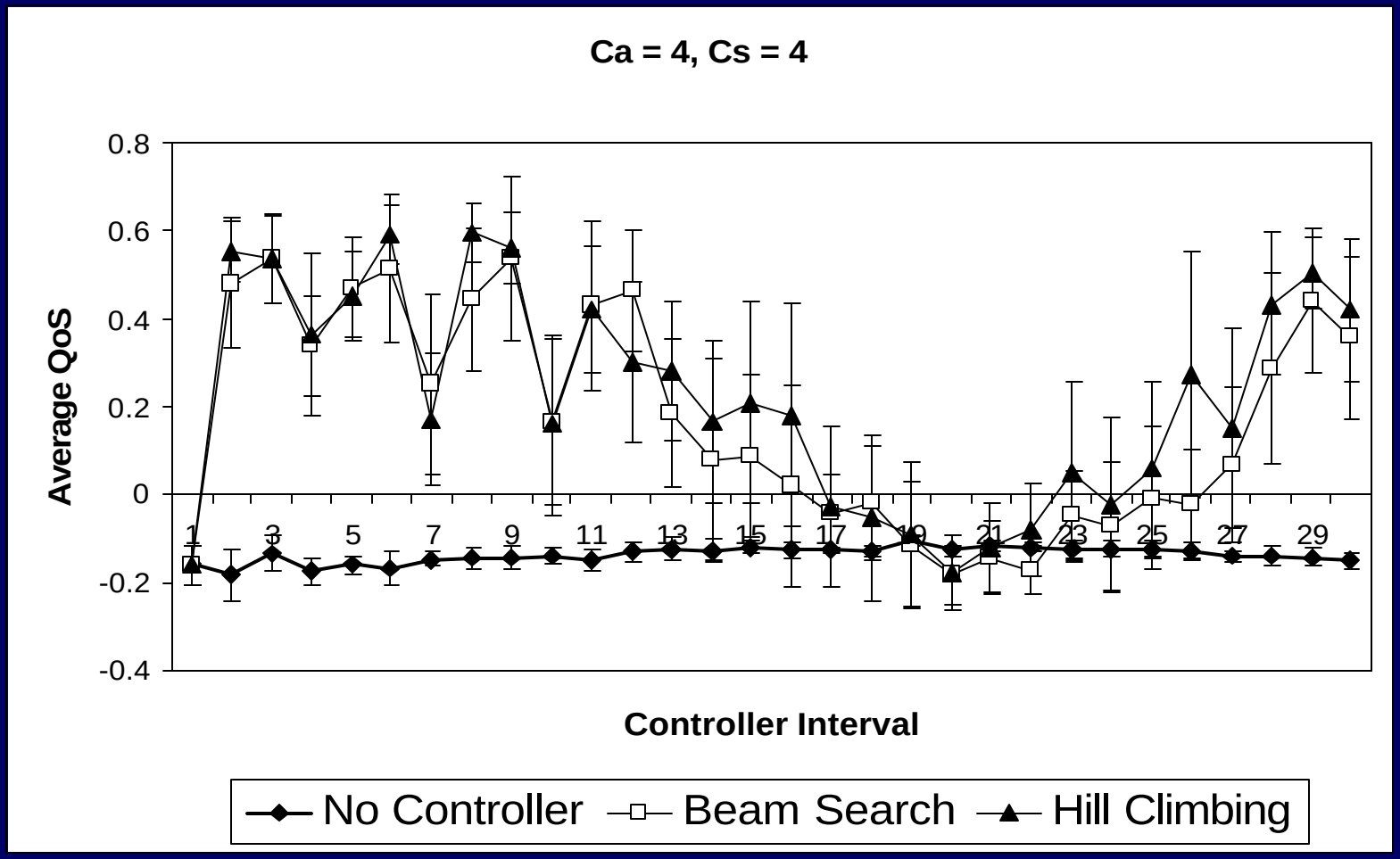
Ca = 1

Ca = 2



Ca = 4

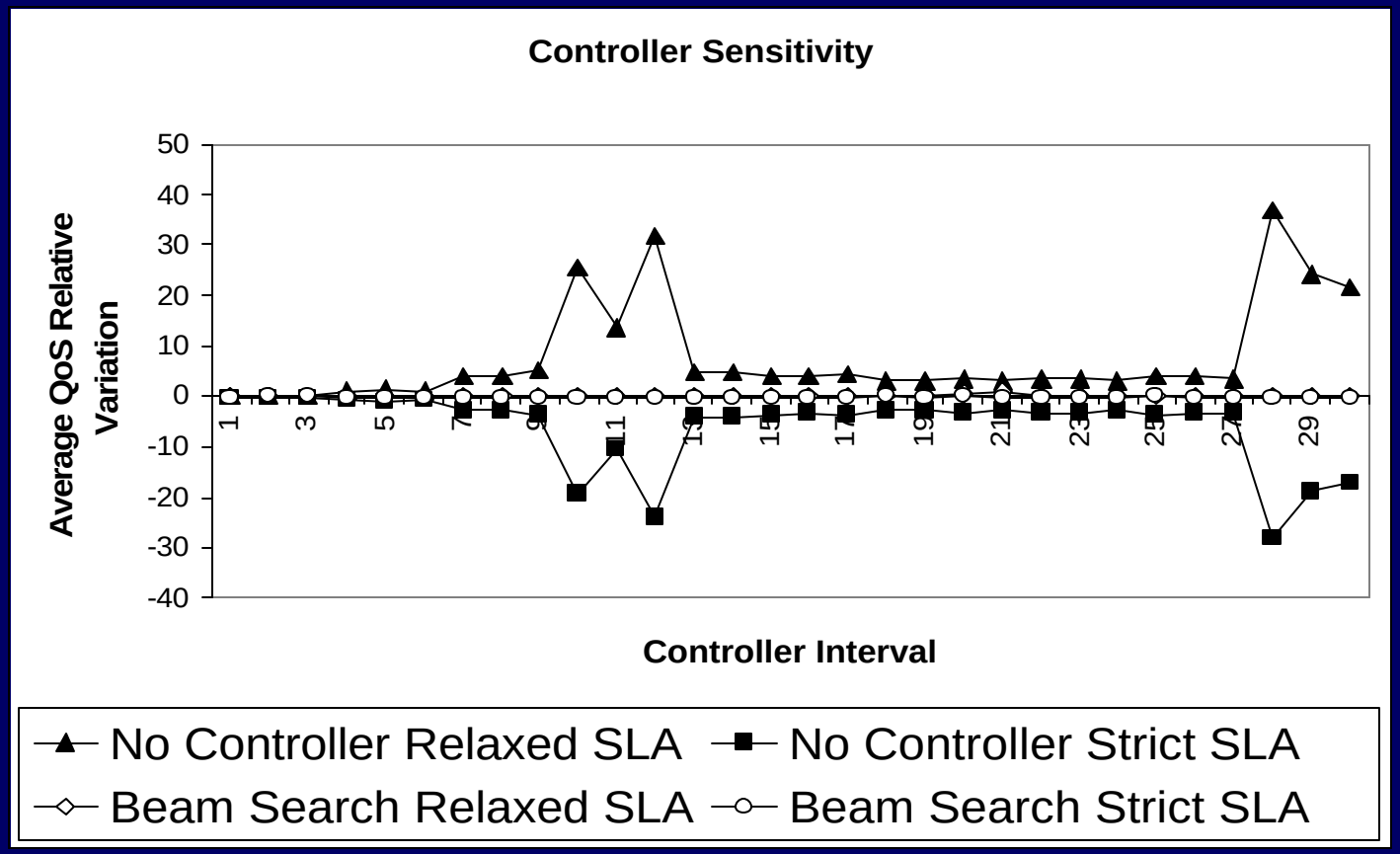
Extreme values for Ca and Cs



Sensitivity of Controller to SLAs

- Need to investigate the controller behavior in the case of a variation in the SLAs
- We ran experiments for stricter and more relaxed SLAs
  - Base: Rmax = 1.2 sec, Xmin = 5 req/sec, Pmax = 0.05
  - Strict: Rmax = 1.0 sec, Xmin = 7 req/sec, Pmax = 0.03
  - Relaxed: Rmax = 1.5 sec, Xmin = 4 req/sec, Pmax = 0.10
- Used Ca = Cs = 2

Sensitivity of Controller to SLAs



Rest of this talk …

- Novel uses for performance models
- Two examples of self-regulating systems:
  - A three-tiered e-commerce system
  - QoS-aware software components
- Concluding Remarks

Characteristics of the new generation of distributed software systems

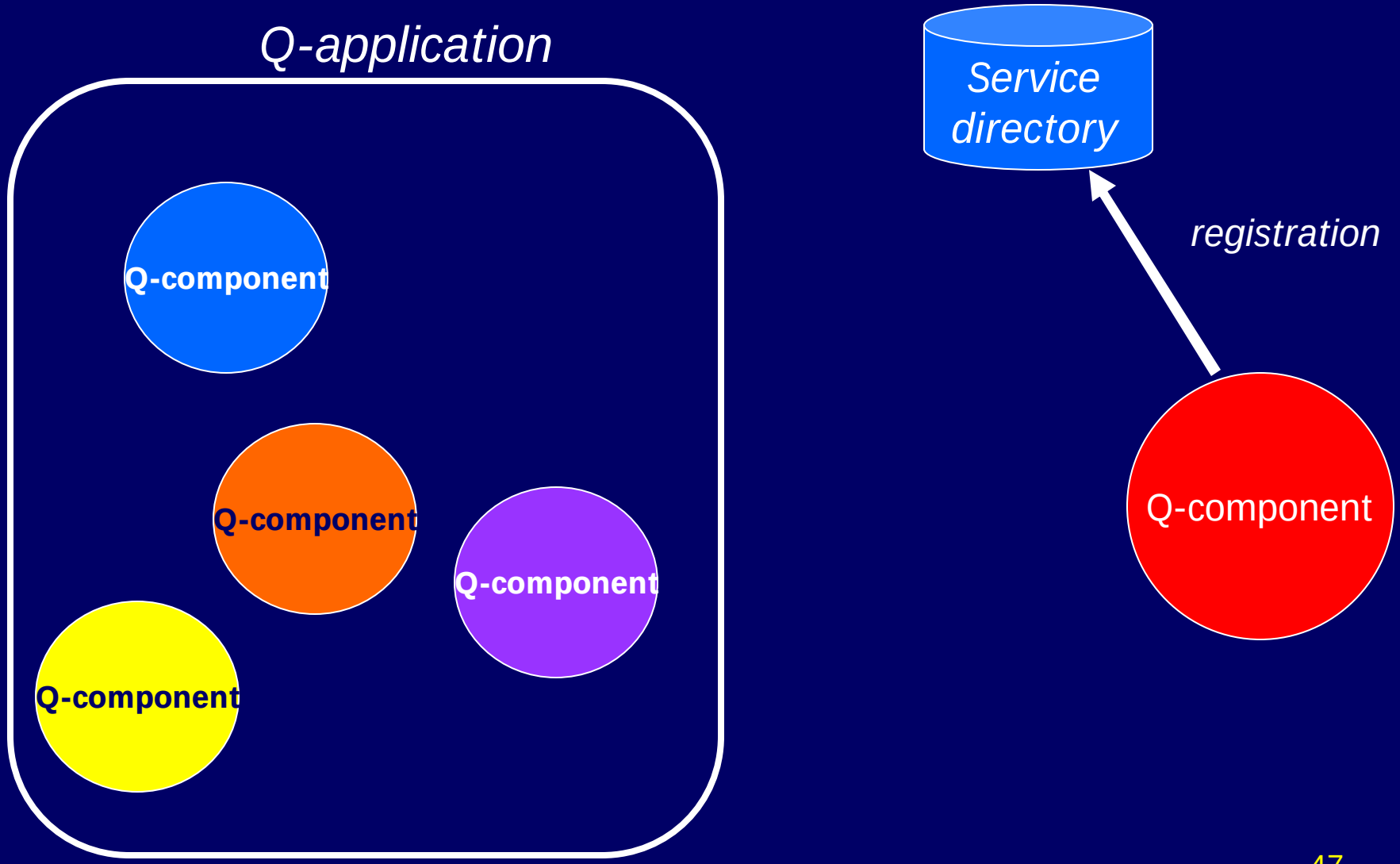
- Highly distributed
- Component-based (for reusability)
- Service-oriented architectures (SOA)
- Unattended operation
- Hostile environments
- Composed of a large number of “replaceable” components discovered at run-time
- Run on a multitude of (unknown and heterogeneous) hardware and network platforms

Requirements of Next Generation of Large Distributed Systems

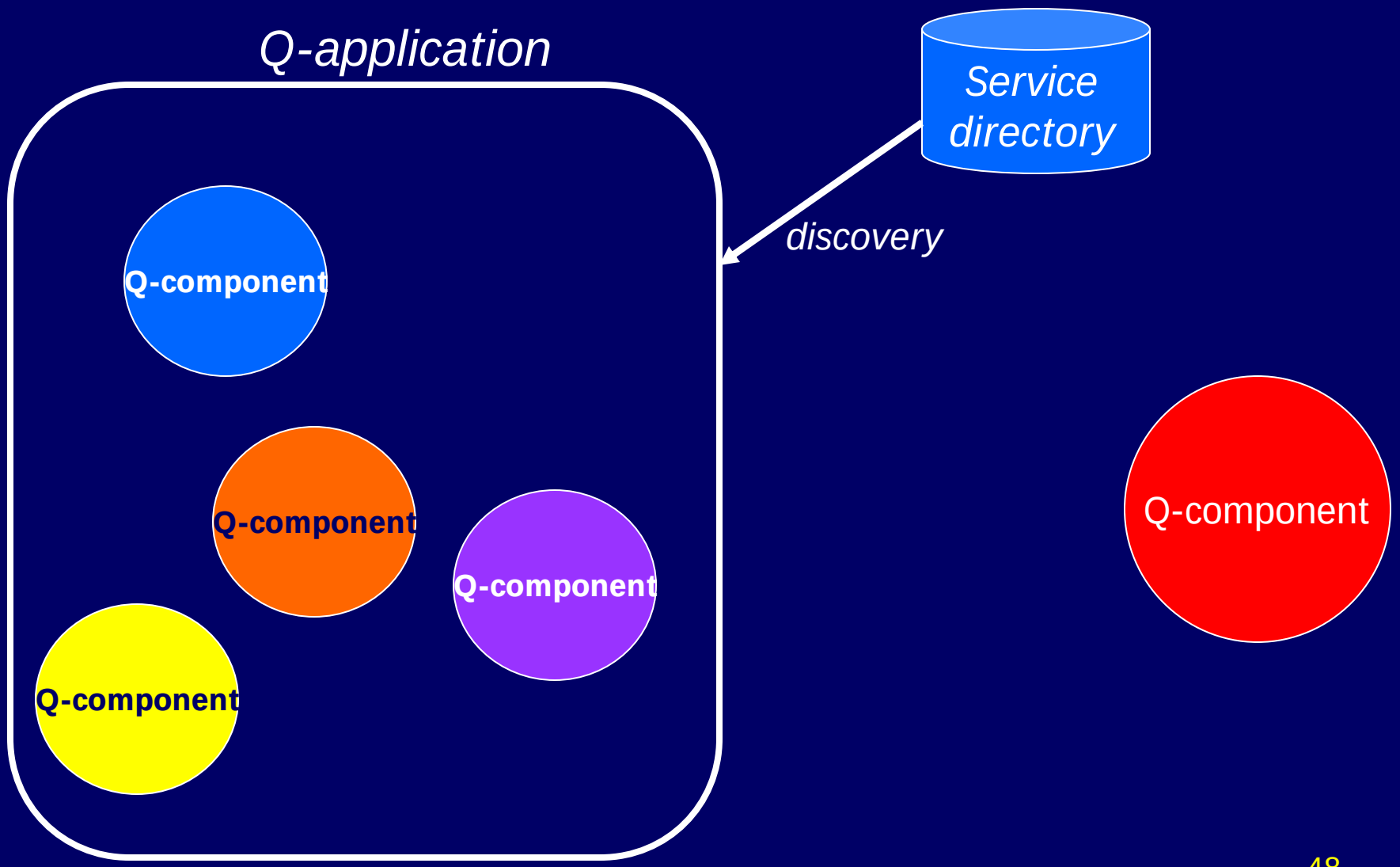
- Adaptable and self-configurable to changes in workload intensity:
  - QoS requirements at the application and component level must be met.
- Adaptable and self-configurable to withstand attacks and failures:
  - Availability and security requirements must be met.

self-configurable, self-optimizing, self-healing, and self-protecting

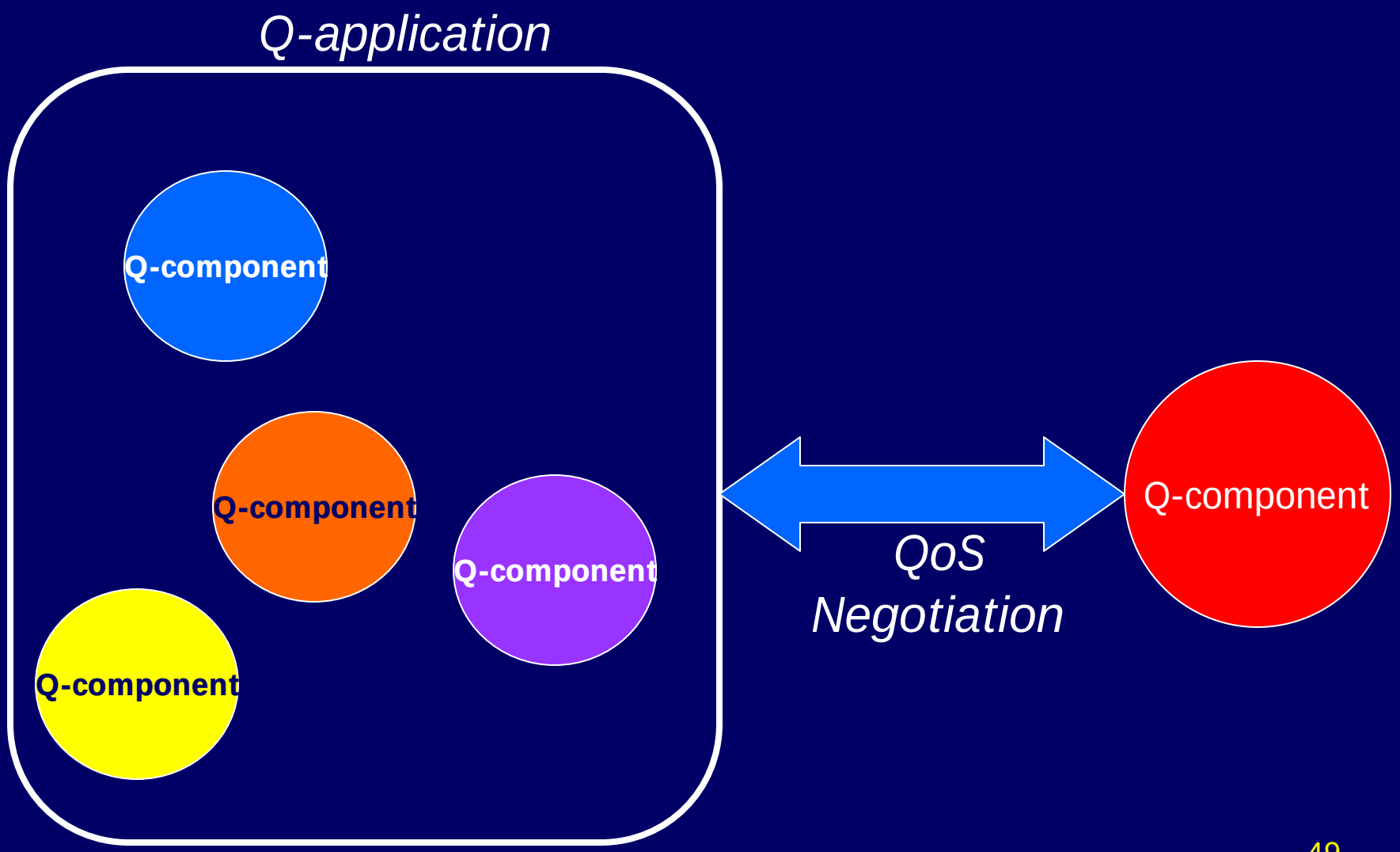
Q-Applications and Q-components



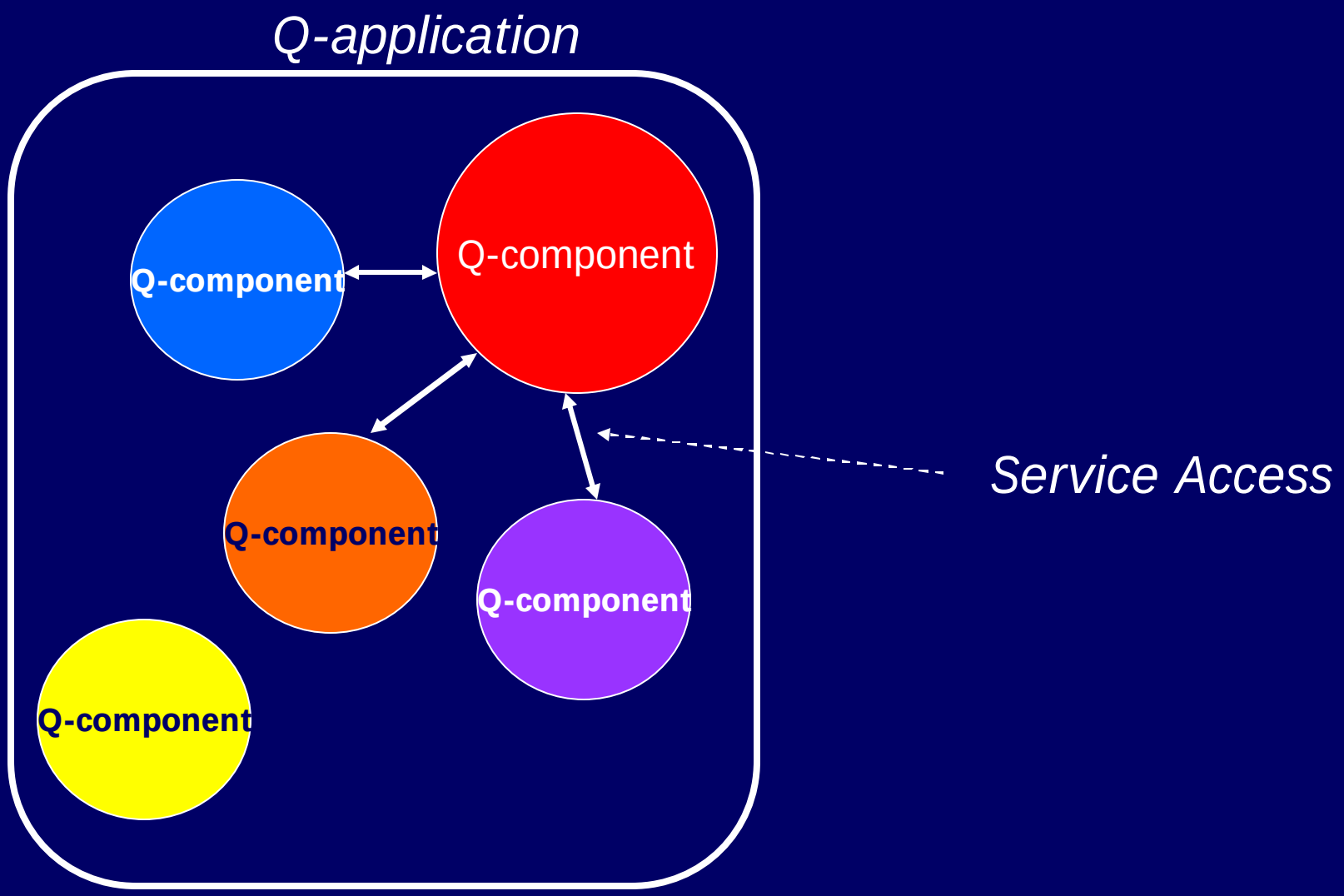
Q-Applications and Q-components



Q-Applications and Q-components



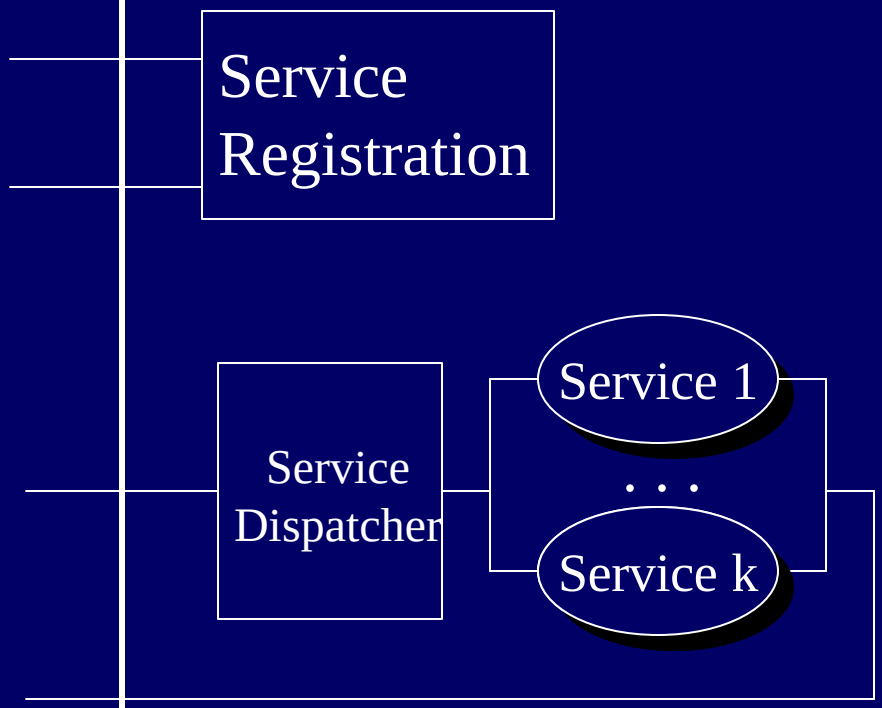
Q-Applications and Q-components



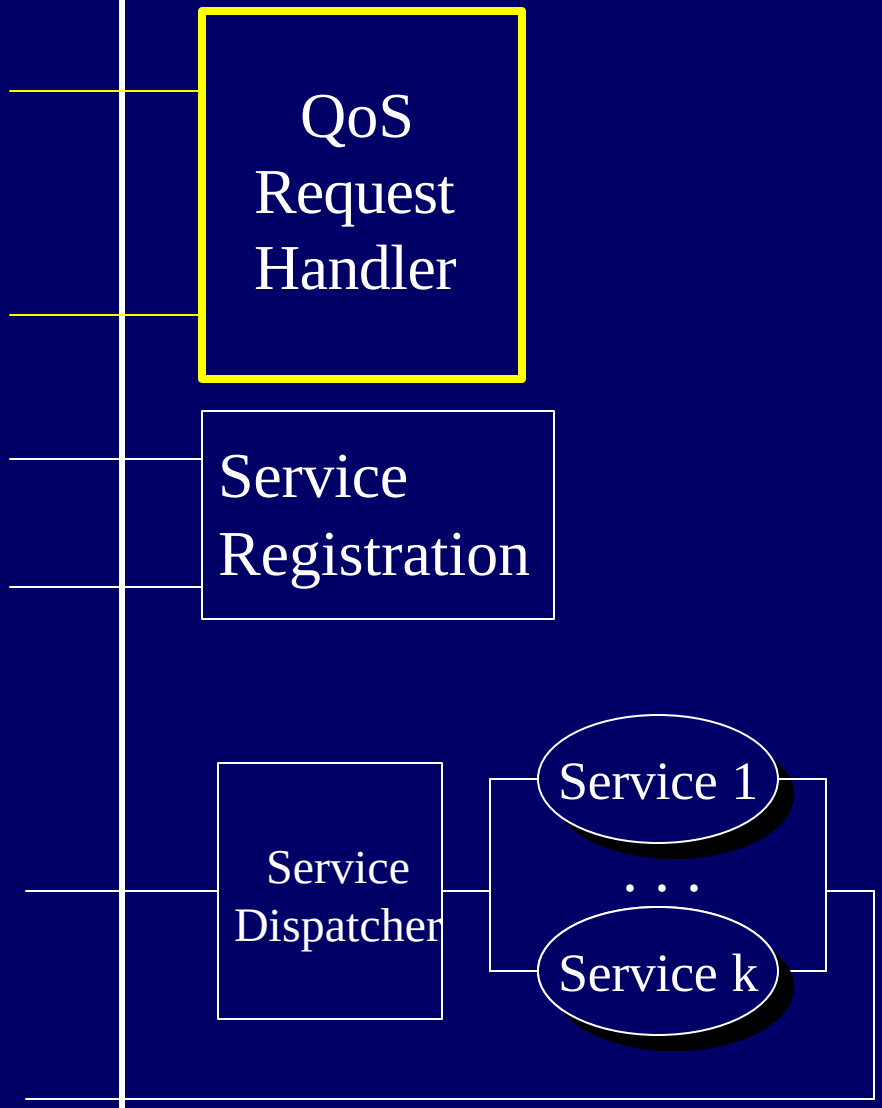
QoS-Aware Software Components: Q-Components

- Engage in QoS Negotiations (accept, reject, counter-offer)
- Provide QoS guarantees for multiple concurrent services
- Maintain a table of QoS commitments
- Service dispatching based on accepted QoS commitments
- Q-components are the building blocks of QoS-aware applications

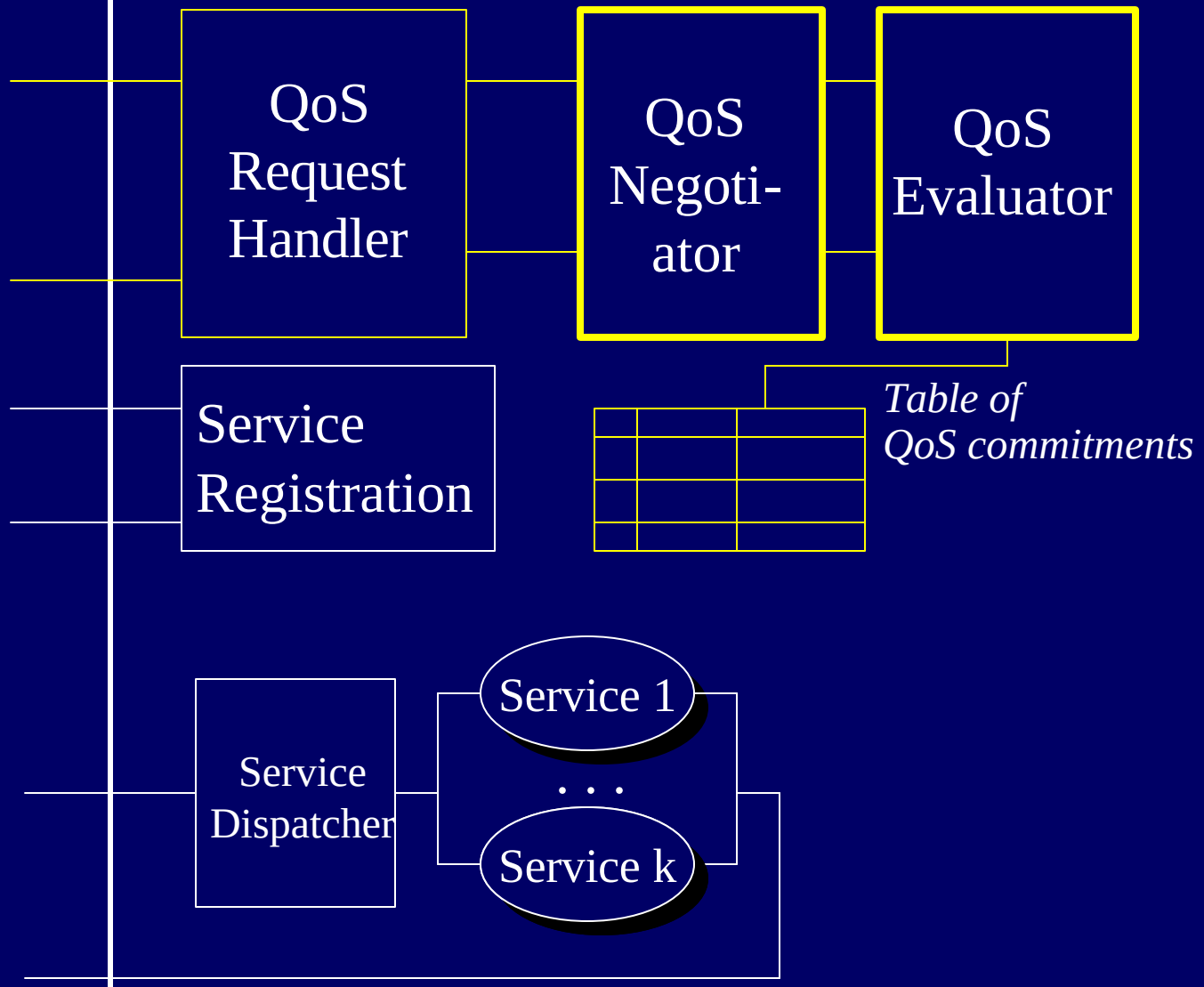
Architecture of a typical software component



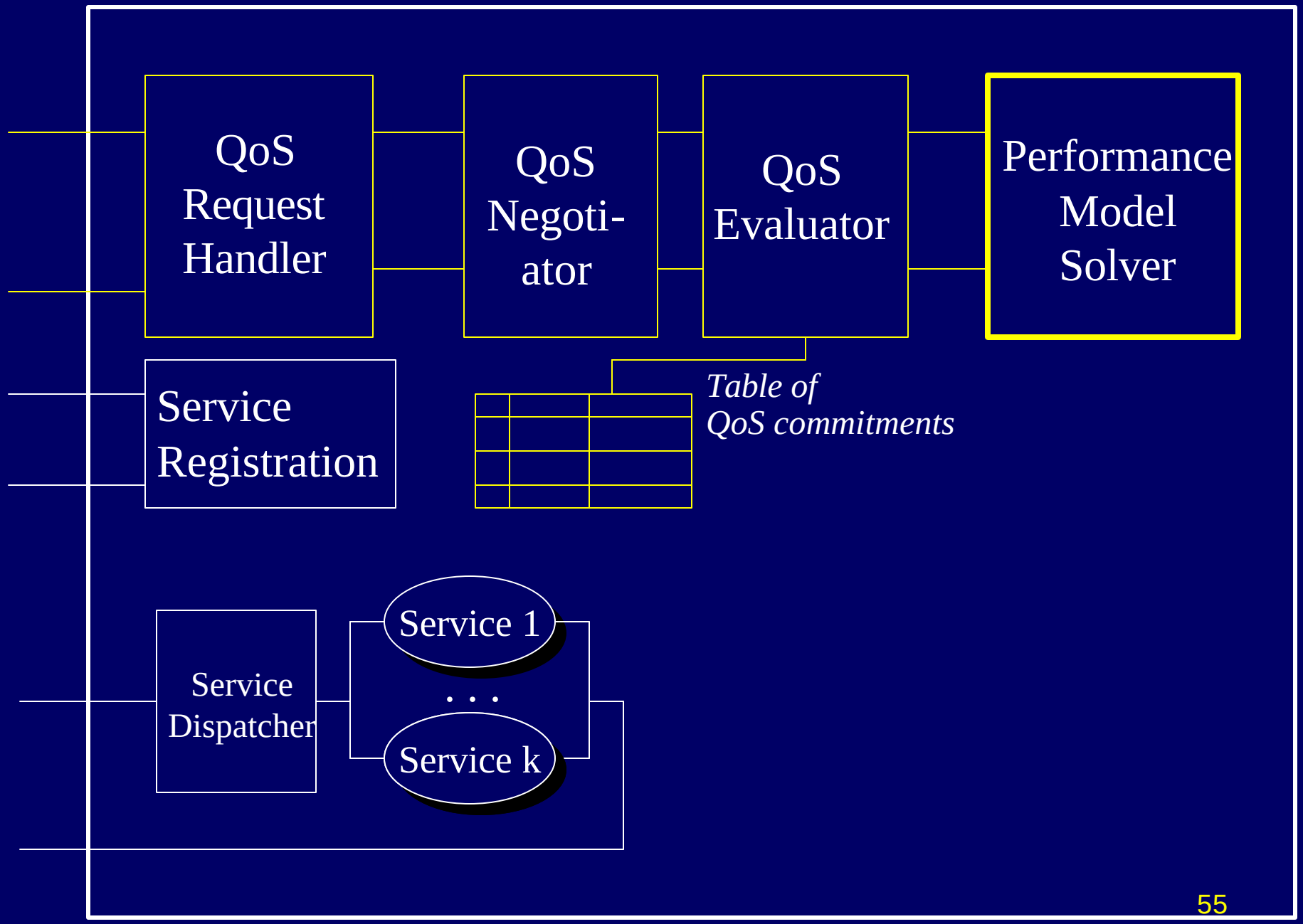
Architecture of a Q-component (QoS Negotiation)



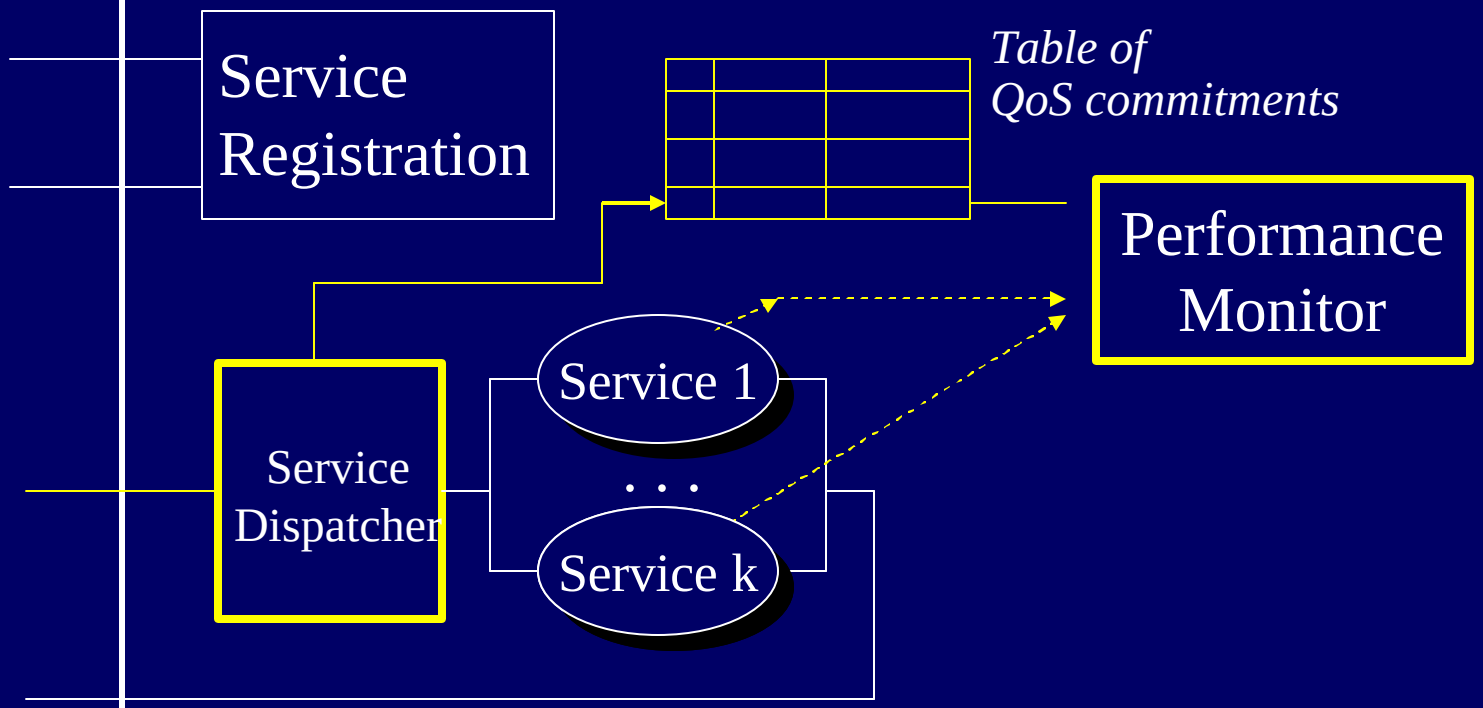
Architecture of a Q-component (QoS Negotiation)



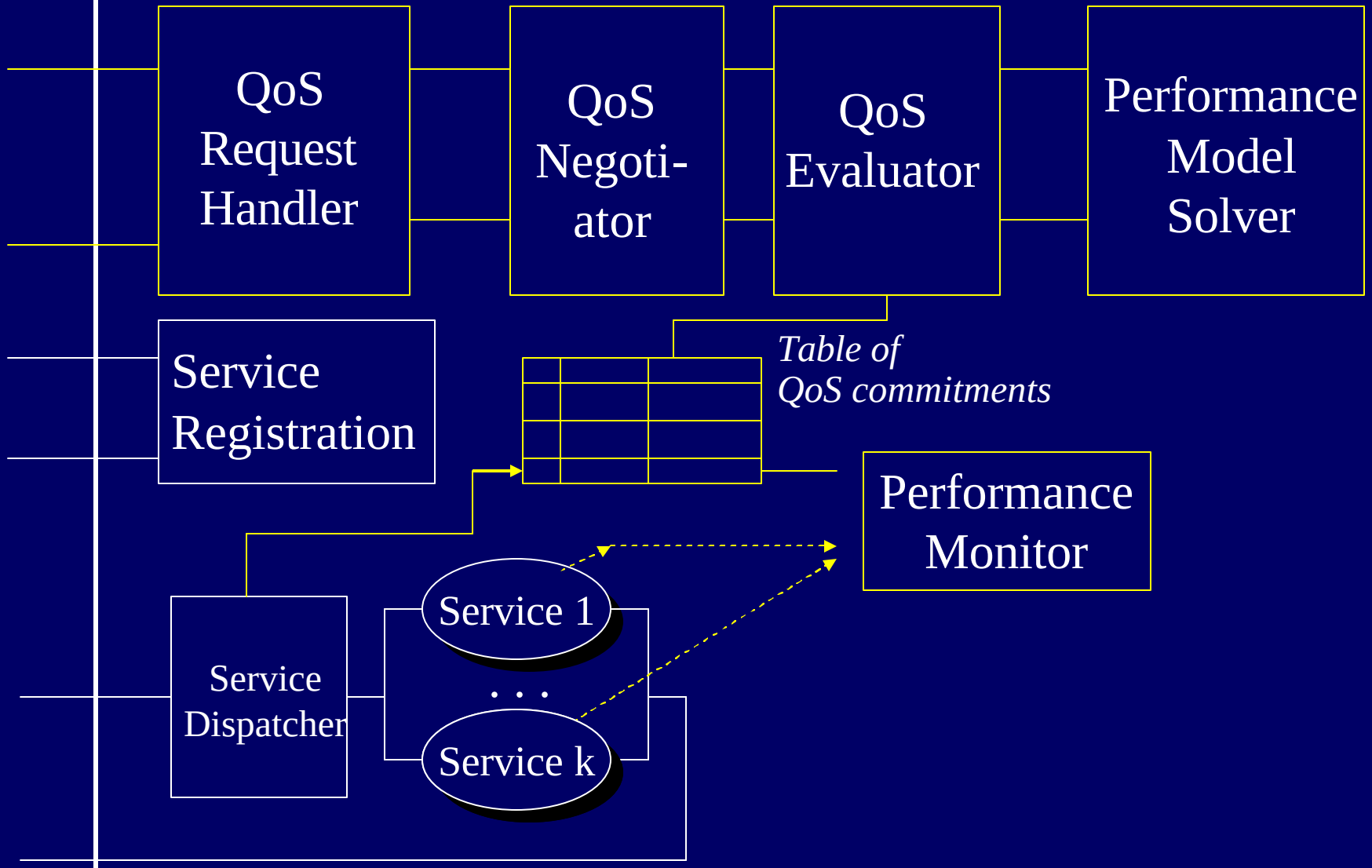
Architecture of a Q-component (QoS Negotiation)



Architecture of a Q-component –Service Requests



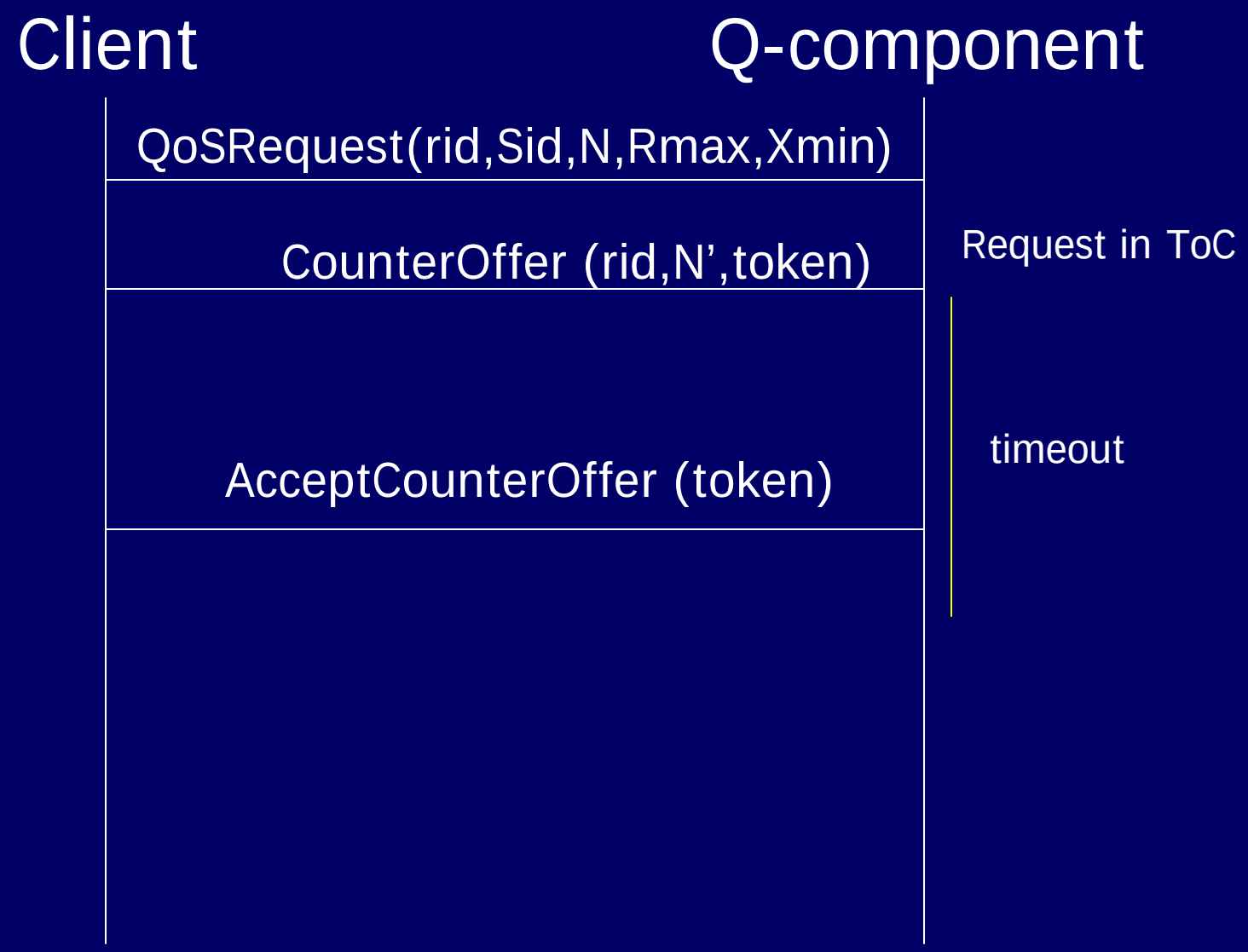
Architecture of a Q-component



Successful QoS Negotiation

Client	Q-component
QoSRequest(rid,Sid,N,Rmax,Xmin)	
Accept(rid,token)	Request in ToC
ServiceReq (...,token)	
ReplyReq (...)	
. . . ServiceReq (...,token)	
ReplyReq (...)	
EndSession (token)	
	Request removed from ToC

On-time Accepted Counteroffer



Expired Accepted Counteroffer

Client | Q-component

QoSRequest(rid,Sid,N,Rmax,Xmin)

CounterOffer (rid,N',token)

Request in ToC

timeout

Request removed from ToC

AcceptCounterOffer (token)

ExpiredCounterOffer (rid)

Rejected Counteroffer

Client | Q-component

QoSRequest(rid,Sid,N,Rmax,Xmin)

CounterOffer (rid,N',token)

Request in ToC

timeout

RejectCounterOffer (token)

Request removed from ToC

Rejected QoS Negotiation

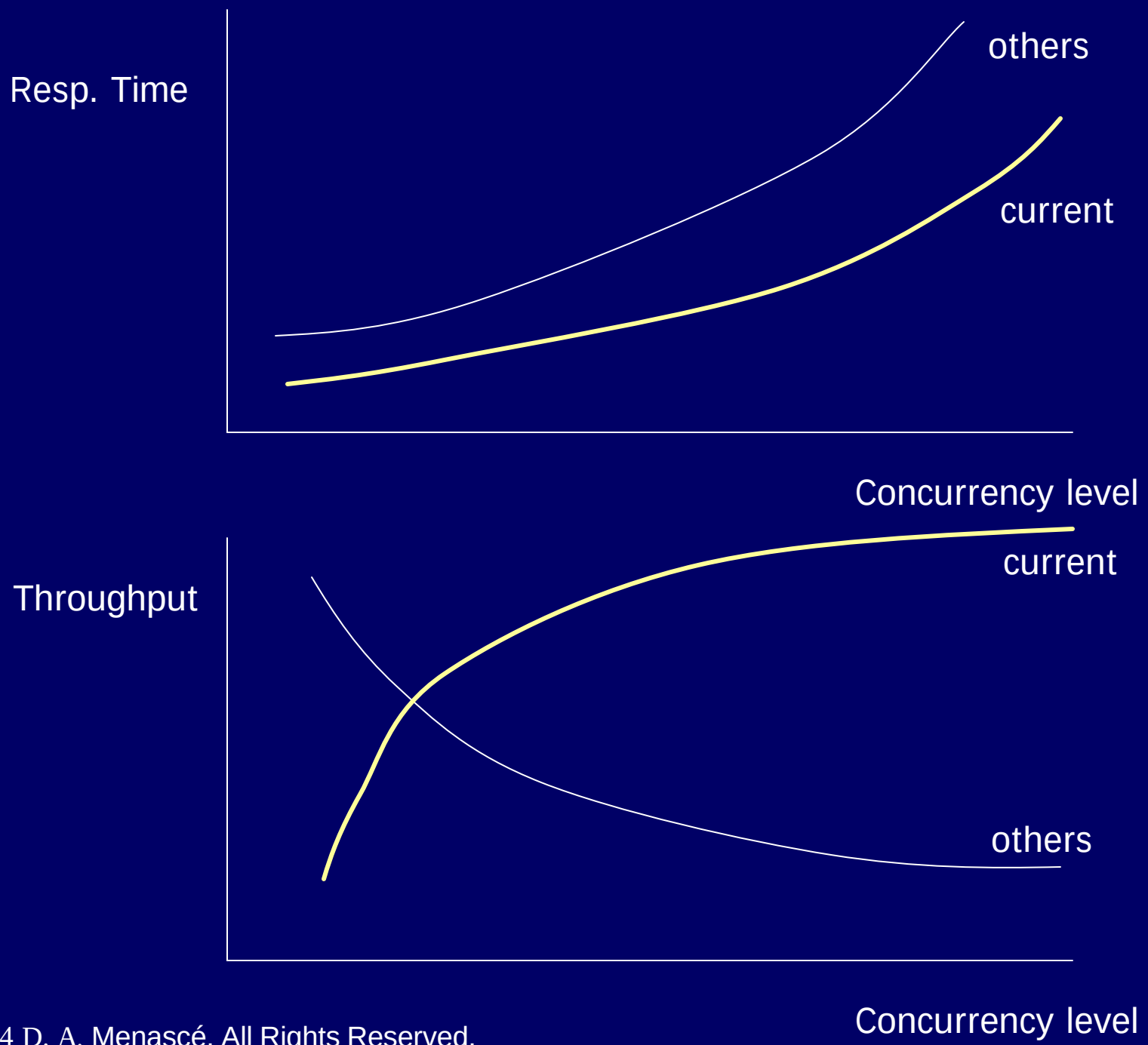
Client | Q-component

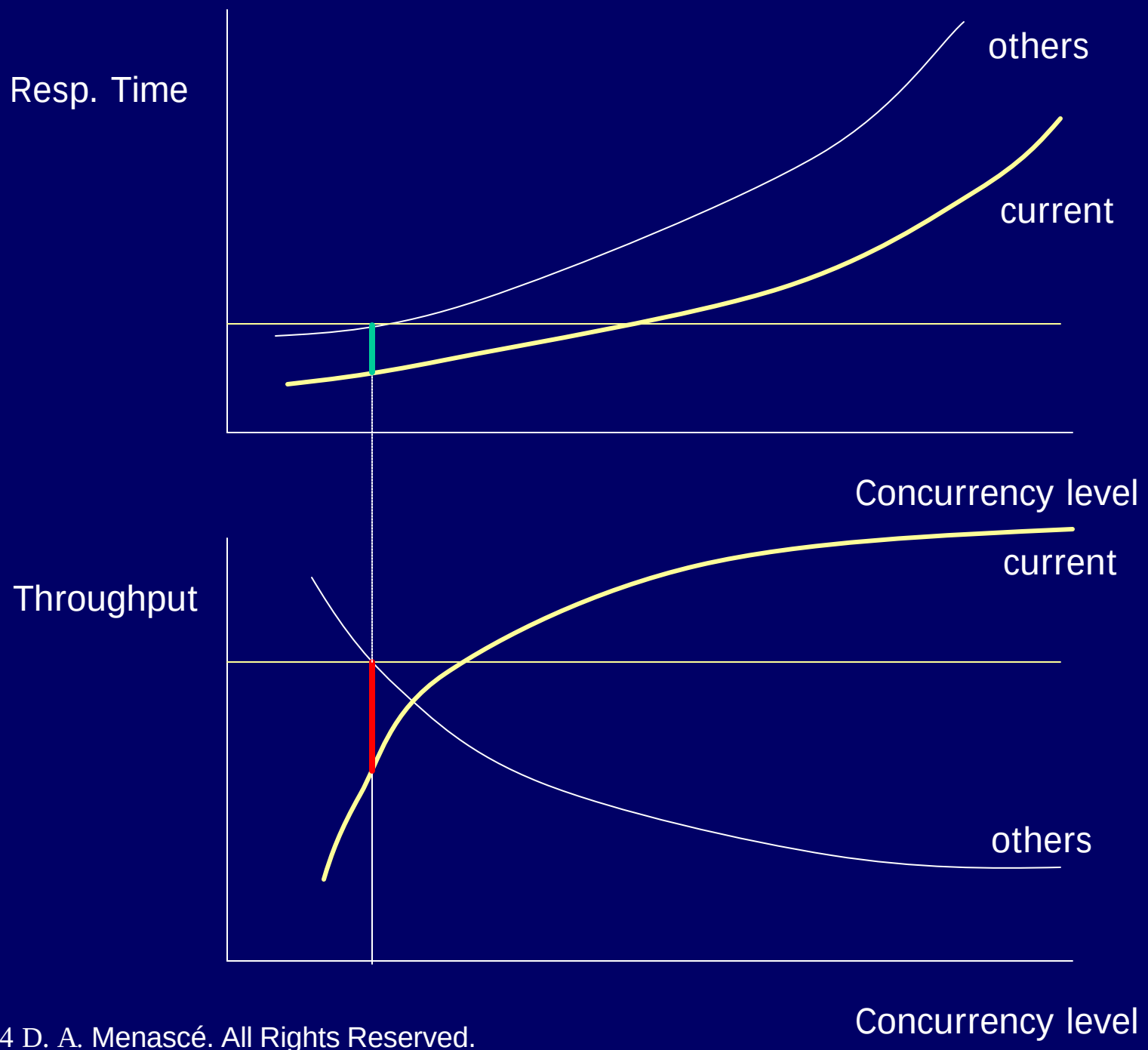
QoSRequest(rid,Sid,N,Rmax,Xmin)

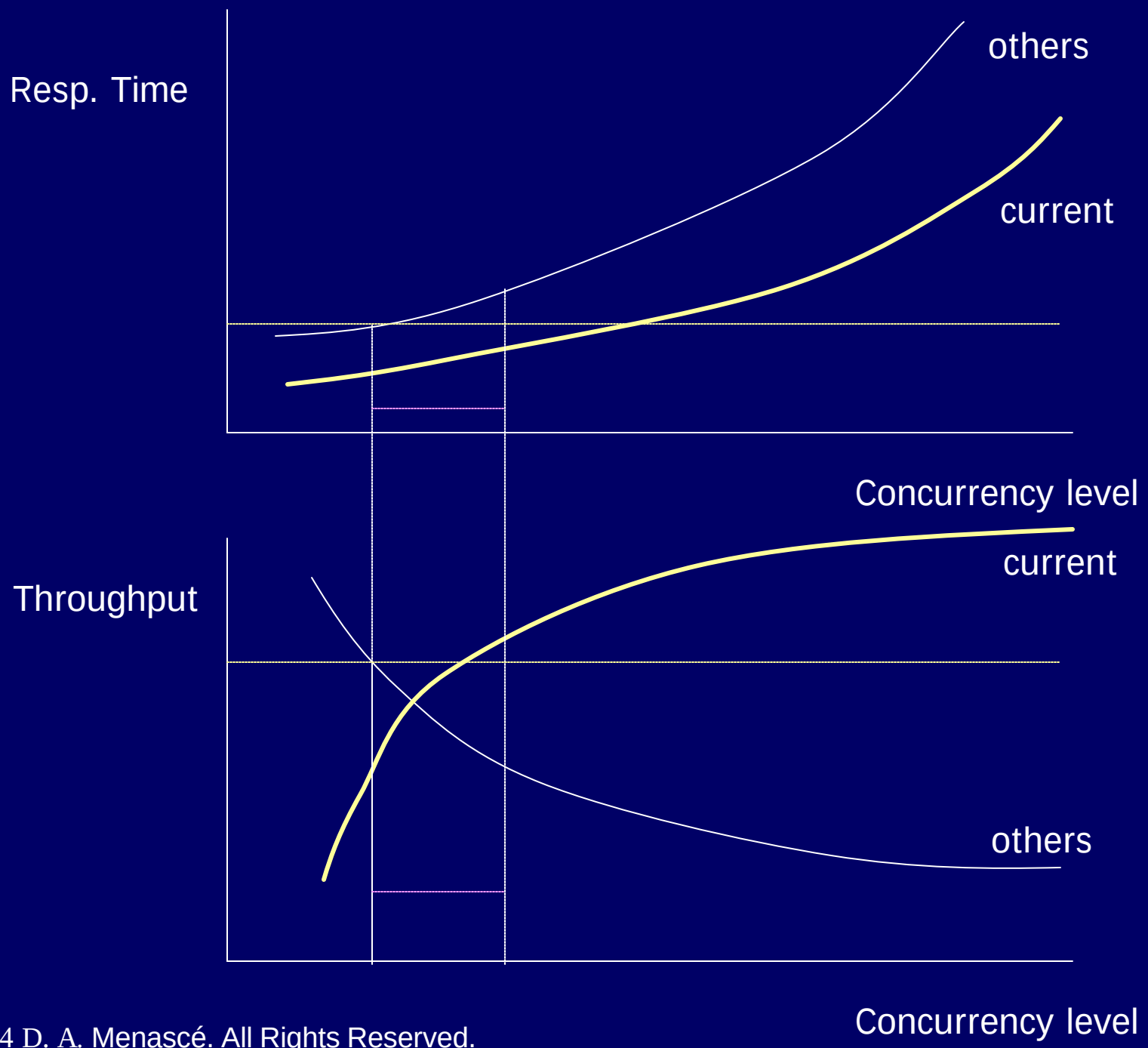
RejectQoSRequest (rid)

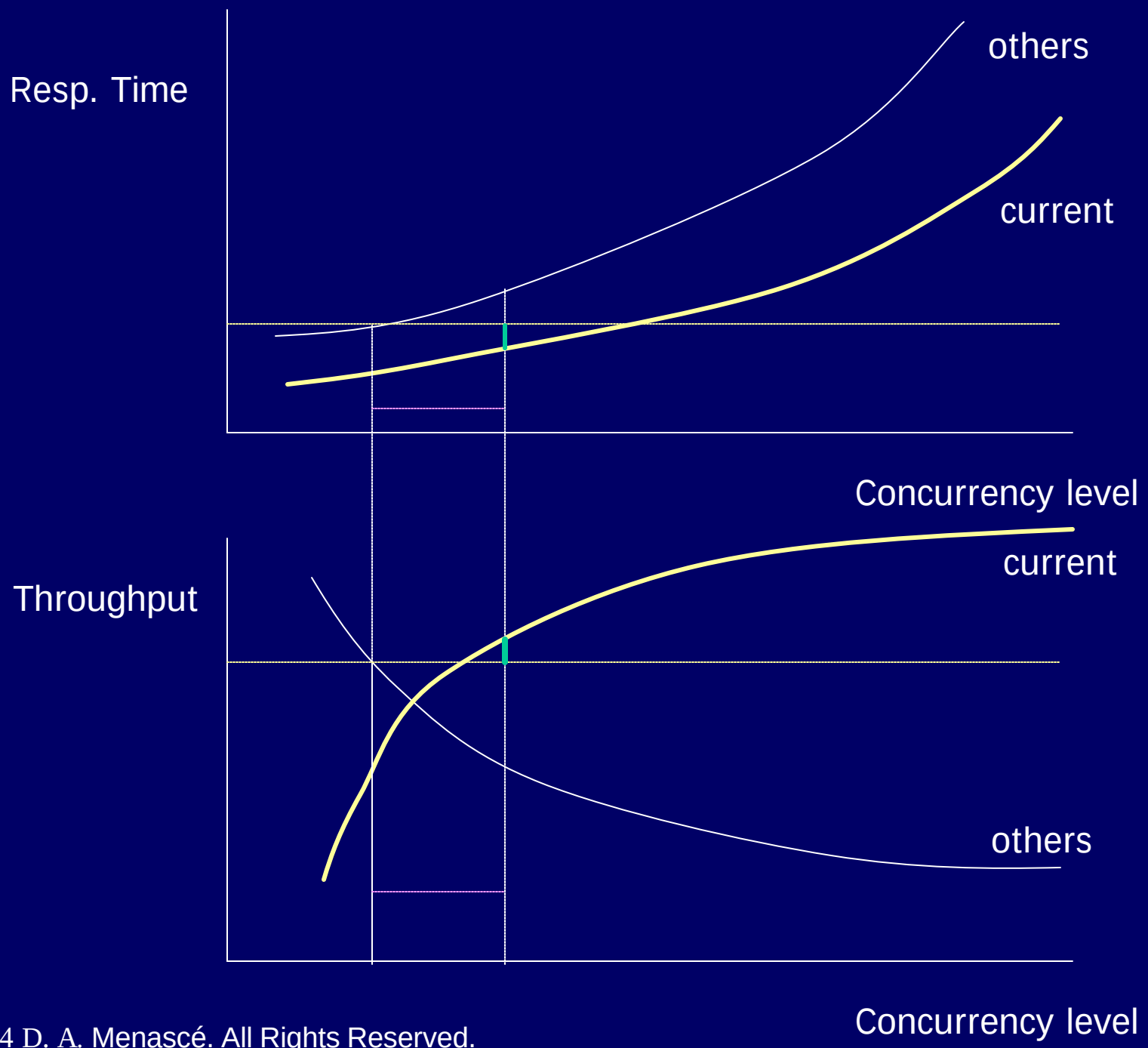
Decision Table for QoS Negotiation

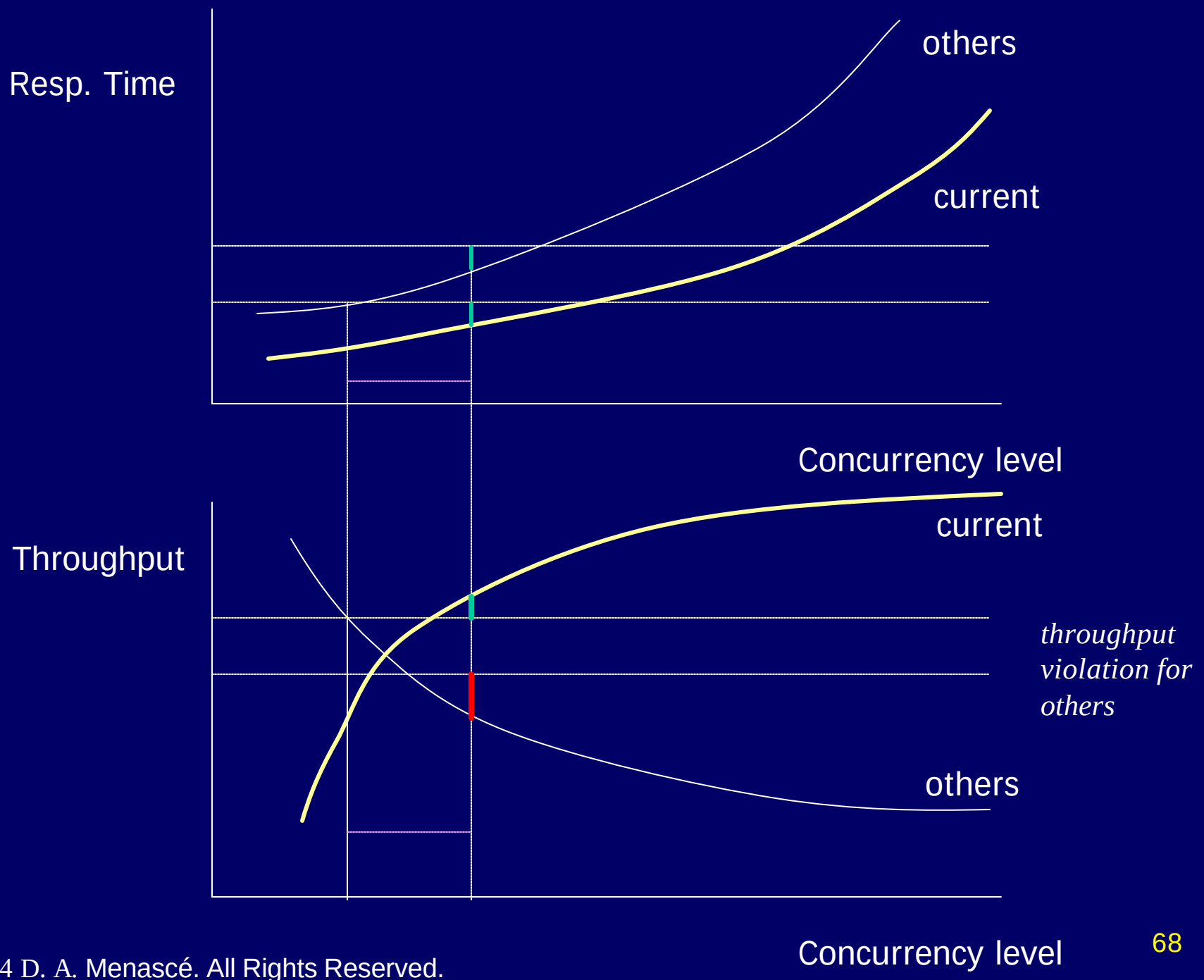
1. Current and other requests are satisfied				Accept
2. Only Current is Violated				
Reason	Remedy	Current	Others	Decision
Only MAXR is violated	Decrease N	OK	OK	Counter Offer
		Not OK: MINX is violated or N=0	OK	Reject
Only MINX is violated	Increase N	OK	OK	Counter Offer
		OK	Not OK	Reject
		Not OK & MAXR is violated	OK	Reject
		Not OK & MAXR is violated	Not OK	Reject
MINX & MAXR are violated	Decreasing N reduces X and increasing N increases R. So, there is no solution.			Reject
3. Current Request and Others are Violated				
Reason	Remedy	Current	Others	Decision
Only MAXR is violated	Decrease N	OK	OK	Counter Offer
		OK	Not OK	Reject
		Not OK: MINX is violated or N=0	OK or not OK	Reject
Only MINX is violated	X could be increased by increasing N. But this would further violate the QoS of other classes.			Reject
MINX & MAXR are violated	Decreasing N reduces X and increasing N increases R. So, there is no solution.			Reject
4. Only Other Requests are Violated				
Reason	Remedy	Current	Others	Decision
Any	Decrease N	OK	OK	Counter Offer
		OK	Not OK: N=1 but others still violated	Reject
		Not OK: MINX violated or N=0	OK or not OK	Reject











Building a Performance Model

New Request: Sid = 3, N = 12

Base Matrix of Service Demands (in msec):

	Service		
	1	2	3
CPU	25	34	20
Disk 1	30	50	24
Disk 2	28	42	31

Table of Commitments (ToC):

Commitment ID	Service ID	N	…
1	2	10	…
2	3	15	…
3	1	8	…
4	1	20	…
5	2	13	…

Matrix of Service Demands (in msec)

	Class					
	1	2	3	4	5	6
CPU	34	20	25	25	34	20
Disk 1	50	24	30	30	50	24
Disk 2	42	31	28	28	42	31

Vector N:	10	15	8	20	13	12

Building a Performance Model

New Request: Sid = 3, N = 12

Base Matrix of Service Demands (in msec):

	Service		
	1	2	3
CPU	25	34	20
Disk 1	30	50	24
Disk 2	28	42	31

Table of Commitments (ToC):

Commitment ID	Service ID	N	…
1	2	10	…
2	3	15	…
3	1	8	…
4	1	20	…
5	2	13	…

Matrix of Service Demands (in msec)

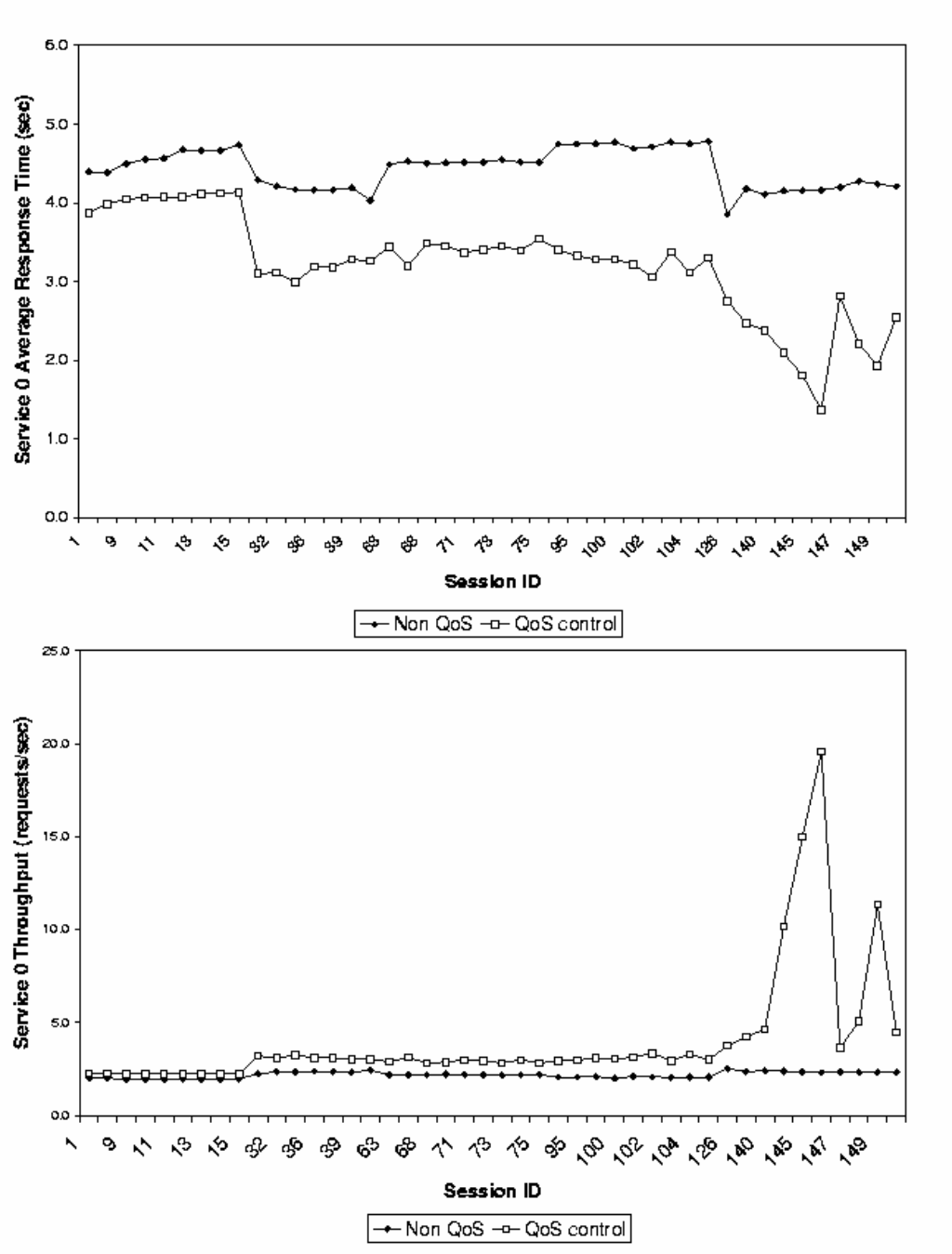
	Class					
	1	2	3	4	5	6
CPU	34	20	25	25	34	20
Disk 1	50	24	30	30	50	24
Disk 2	42	31	28	28	42	31

Vector N:	10	15	8	20	13	12

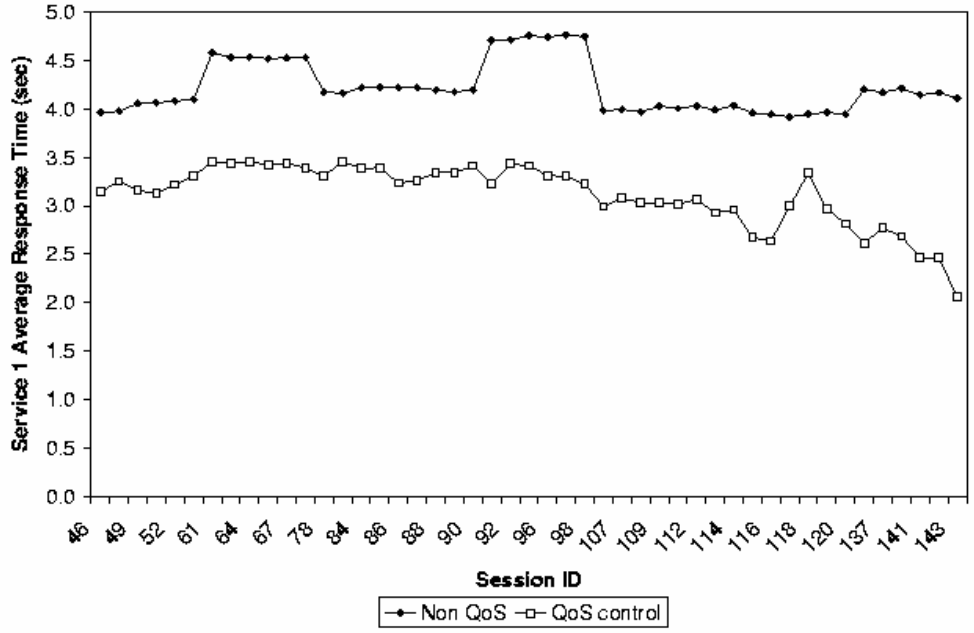
Experimental Evaluation

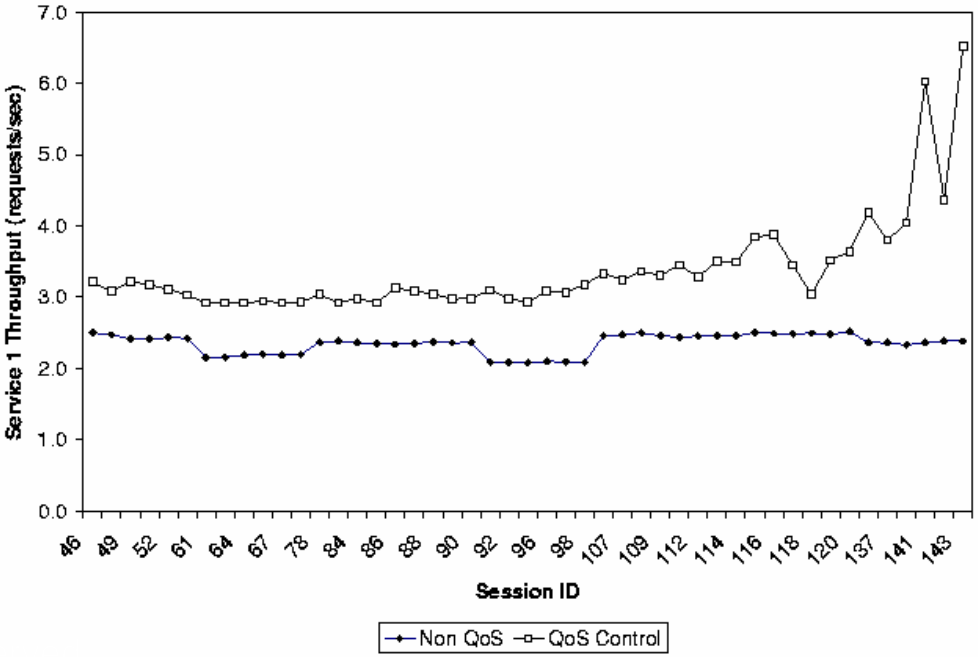
- Built Q-components in Java
- Generated random workload and captured the workload for replay.
- Measured the QoS of the random workload without the Q-component.
- Replayed the random workload against the Q-component and selected a stricter QoS level than obtained without the Q-component
- Ran several experiments and computed average and confidence intervals.

Service 0 Results:

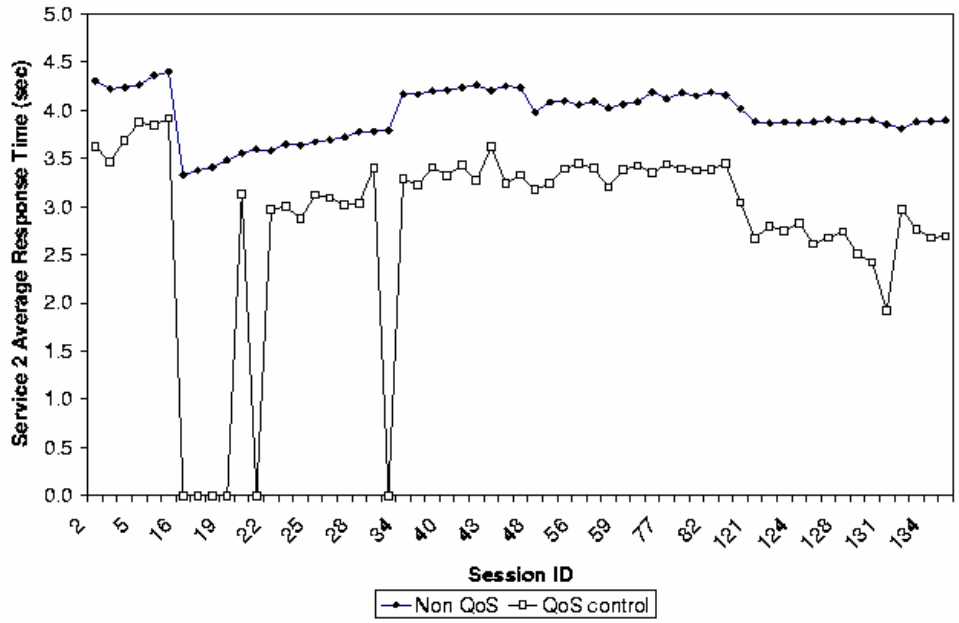


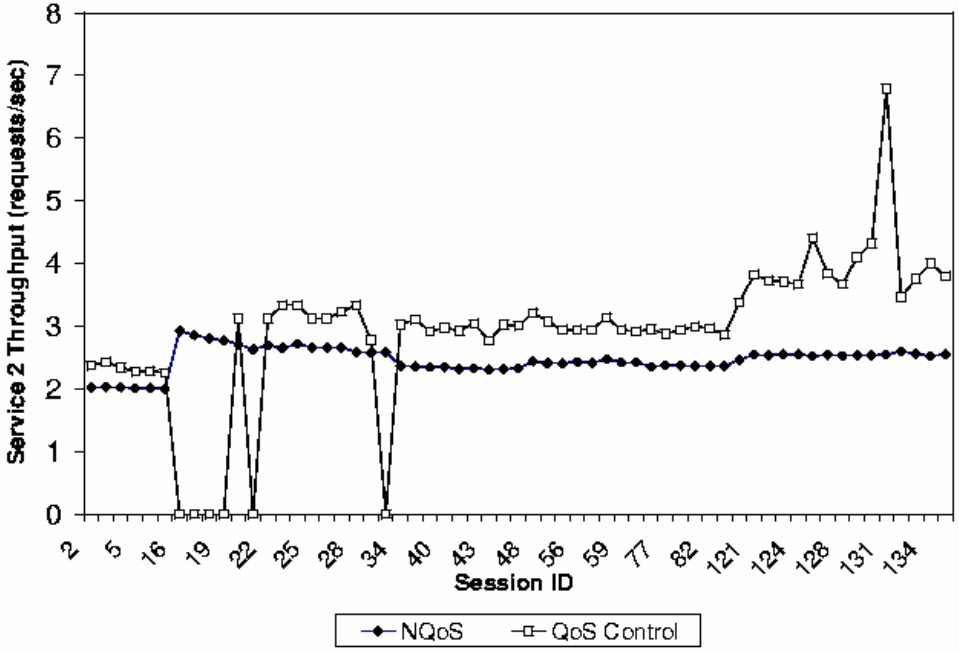
Service 1 Results:





Service 2 Results:



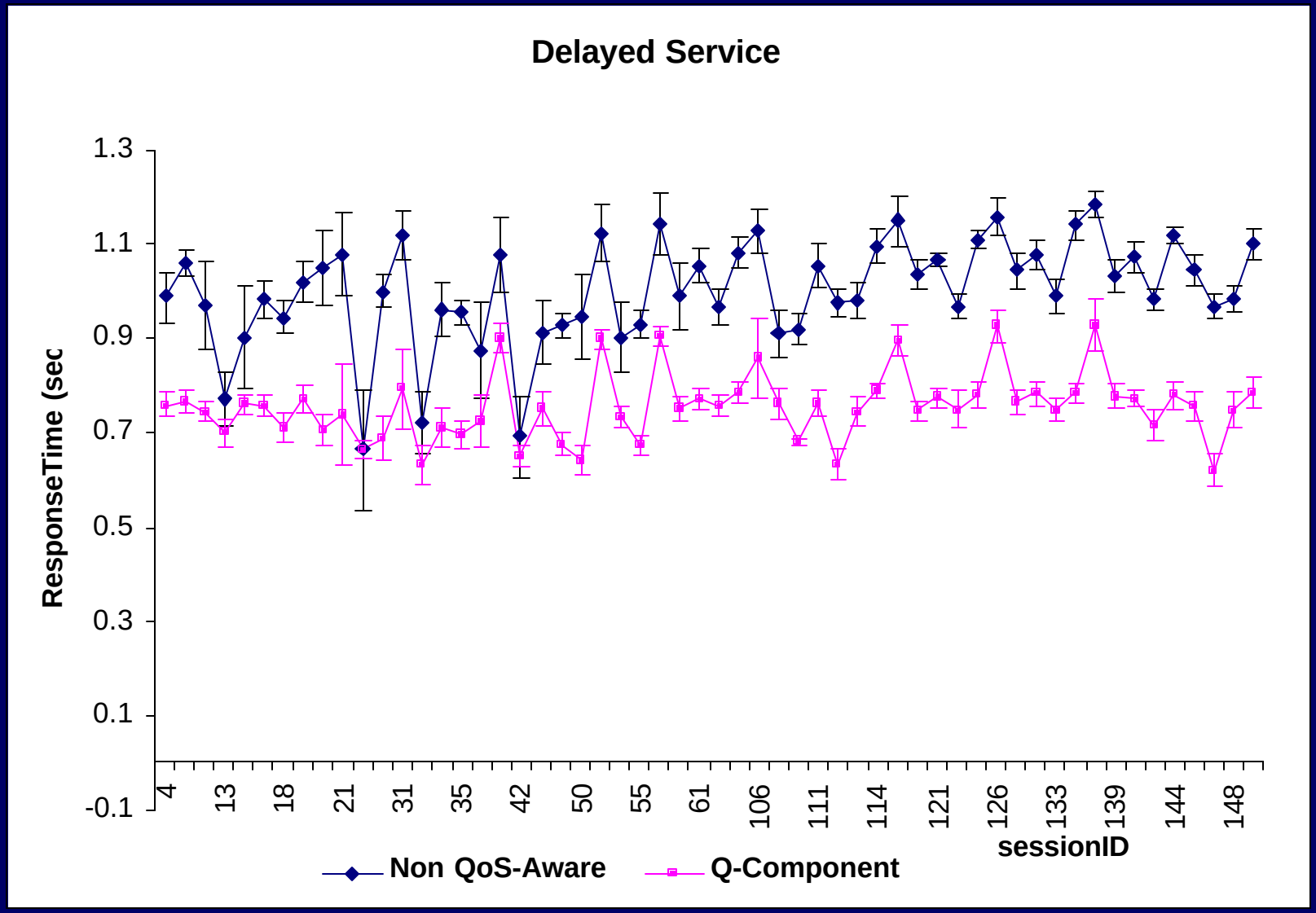


f = 0.0				
Svc No.	No. Dropped Sessions	No. of Sessions	% Drop	% Resp. Time Reduction
0	24	440	5	11
1	19	470	4	9
2	59	590	10	7
Total	102	1500	7	9

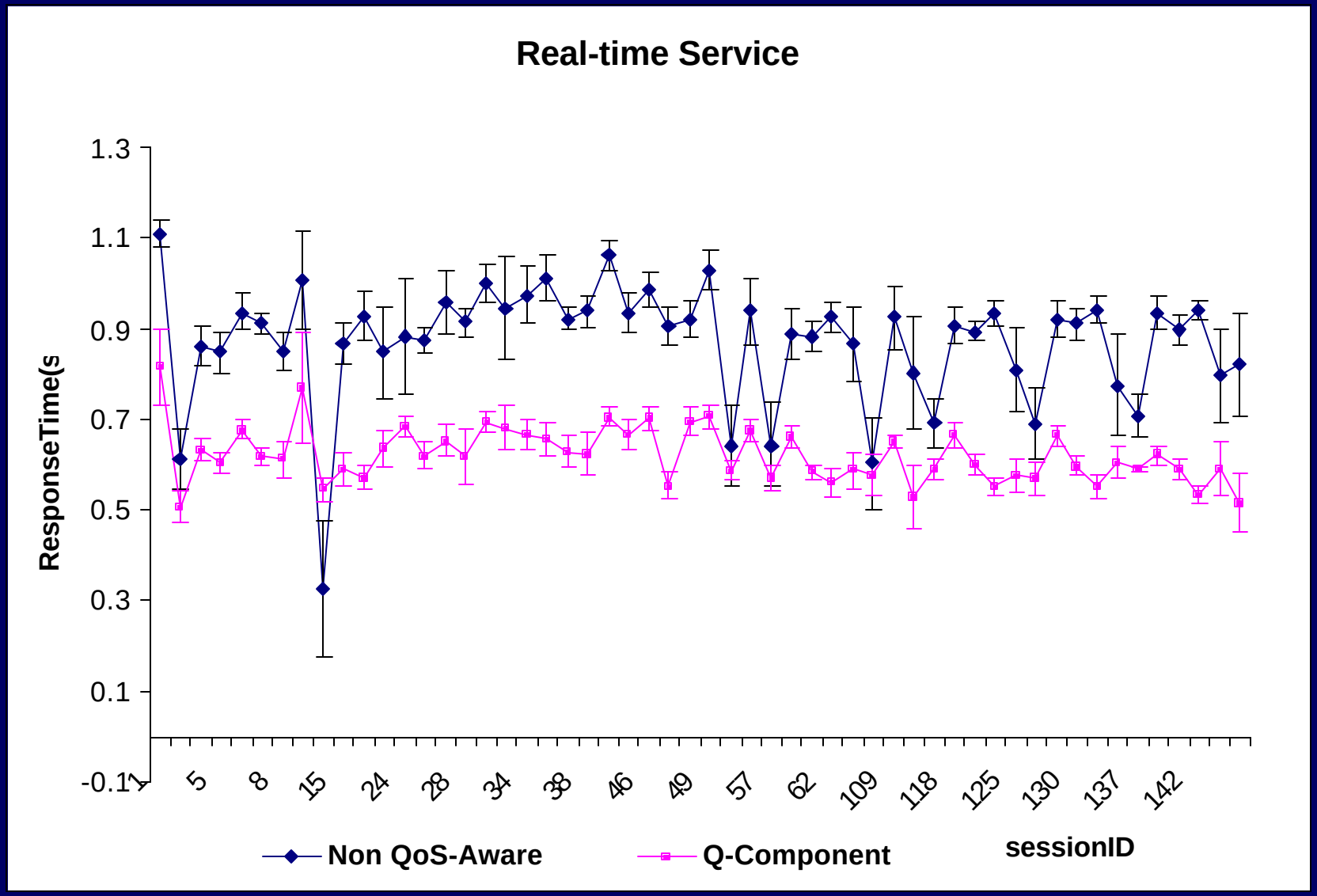
f = 0.10				
Svc No.	No. Dropped Sessions	No. of Sessions	% Drop	% Resp. Time Reduction
0	52	440	12	21
1	66	470	14	16
2	148	590	25	12
Total	266	1500	18	16

f = 0.25				
Svc No.	No. Dropped Sessions	No. of Sessions	% Drop	% Resp. Time Reduction
0	92	440	21	28
1	140	470	30	26
2	263	590	45	20
Total	102	1500	33	24

Stock Quote Service



Stock Quote Service



Summary of Results for Stock Quuotes Application

Requested Response Time Reduction	20.0%
Obtained Response Time Reduction	27.0%
Percent dropped sessions	33.5%
Percent counter offers	35.0%
Percent accepted requests	31.5%

Concluding Remarks

- Performance models can be used to build QoS controllers for complex multi-tiered systems:
  - Controlled system exhibits better QoS values even in case of high variability in request's inter-arrival and service times
  - Short term workload forecasting improves the QoS, especially when the workload intensity gets close to system saturation level
  - Dynamic adjustment of the controller interval length improves the QoS further
  - Even when basic model assumptions are violated, the models are robust enough to track the evolution of the performance metrics as the workload and configuration parameters change.

Concluding Remarks (Cont'd)

- Performance models can be used by software components to make admission control decisions.
  - QoS components should be able to negotiate QoS requests and perform admission control
  - QoS negotiation overhead is small (it did not exceed 10% of the CPU service demand in our experiments).

Bibliography

- "Assessing the Robustness of Self-Managing Computer Systems under Highly Variable Workloads," M. Bennani and D. Menascé, *Proc. International Conf. Autonomic Computing (ICAC-04)*, New York, NY, May 17-18, 2004.
- "A Framework for QoS-Aware Software Components," D. Menascé, H. Ruan ,and H. Gomaa, *Proc. 2004 ACM Workshop on Software and Performance (WOSP'04)*, San Francisco, CA, January 14, 2004.
- "On the Use of Performance Models to Design Self-Managing Systems," D. Menascé and M. Bennani, Proc. 2003 Computer Measurement Group Conference, Dallas, TX, Dec. 71-2, 2003.
- "Automatic QoS Control," *IEEE Internet Computing*, January/February 2003, Vol. 7, No. 1.
- "Preserving QoS of E-commerce Sites Through Self-Tuning: A Performance Model Approach," D. A. Menascé, R. Dodge and D. Barbara, *Proc. 2001 ACM Conference on E-commerce*, Tampa, FL, October 14-17, 2001.

Thanks.

For slides go to:

http://www.cs.gmu.edu/faculty/menasce.html

and click on Talks (after July 16, 2004).