Module 2-CS 02
L05 Performance

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Harvinder S Jabbal
SSZG653 Software Architectures

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Module 2-CS 02
L05 Performance

Chapter Outline

innovate achieve lead

What is Performance?

Performance General Scenario

Tactics for Performance

A Design Checklist for Performance

Summary

What is Performance?

Performance is about time and the software system's ability to meet timing requirements.

When events occur – interrupts, messages, requests from users or other systems, or clock events marking the passage of time – the system, or some element of the system, must respond to them in time.

Characterizing the events that can occur (and when they can occur) and the system or element's time-based response to those events is the essence is discussing performance.

Performance General Scenario

innovate achieve lead

Portion of Scenario	Possible Values
Source	Internal or external to the system
Stimulus	Arrival of a periodic, sporadic, or stochastic event
Artifact	System or one or more components in the system.
Environment	Operational mode: normal, emergency, peak load, overload.
Response	Process events, change level of service
Response Measure	Latency, deadline, throughput, jitter, miss rate

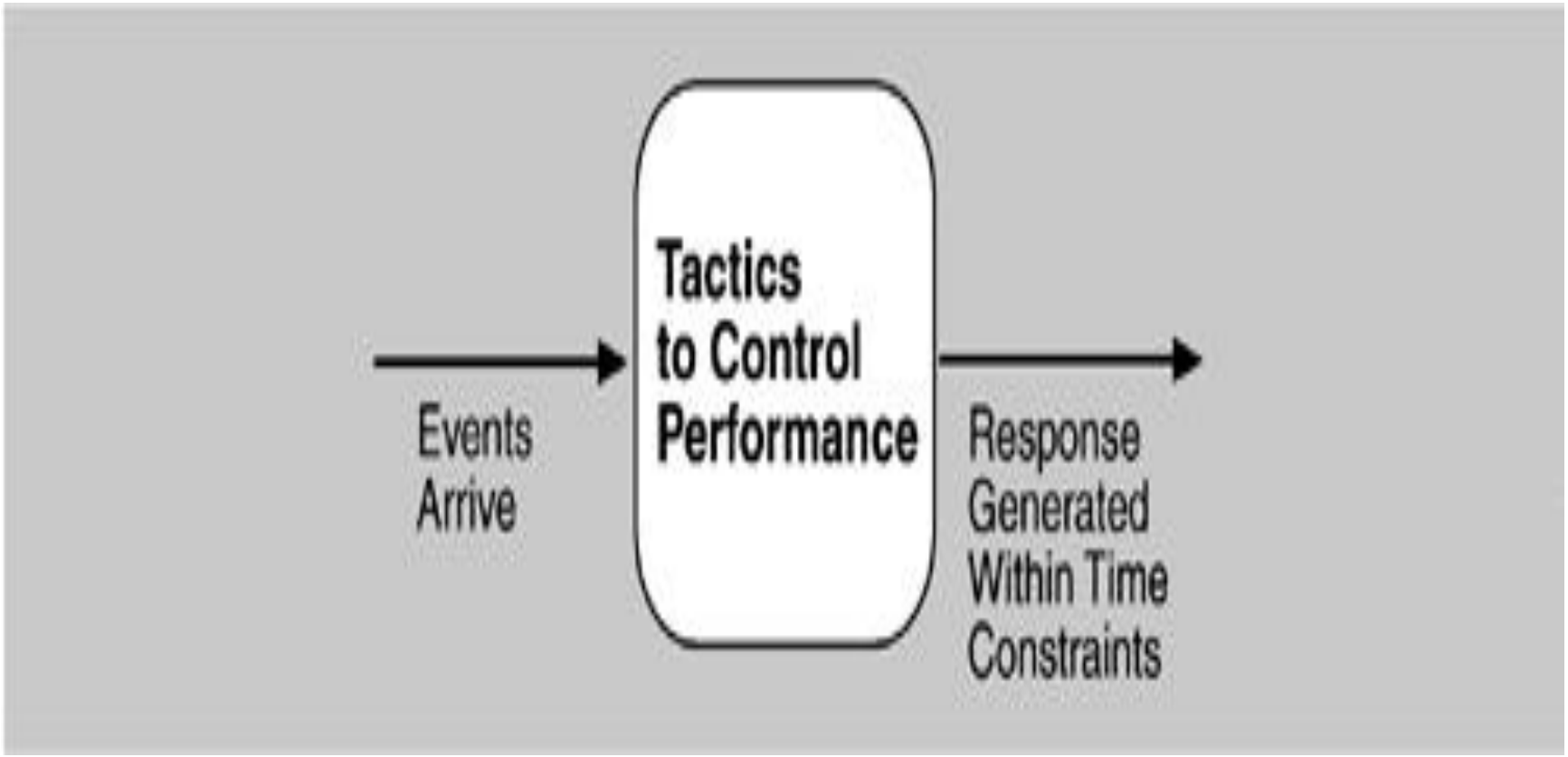
Sample Concrete Performance Scenario

Users initiate transactions under normal operations. The system processes the transactions with an average latency of two seconds.

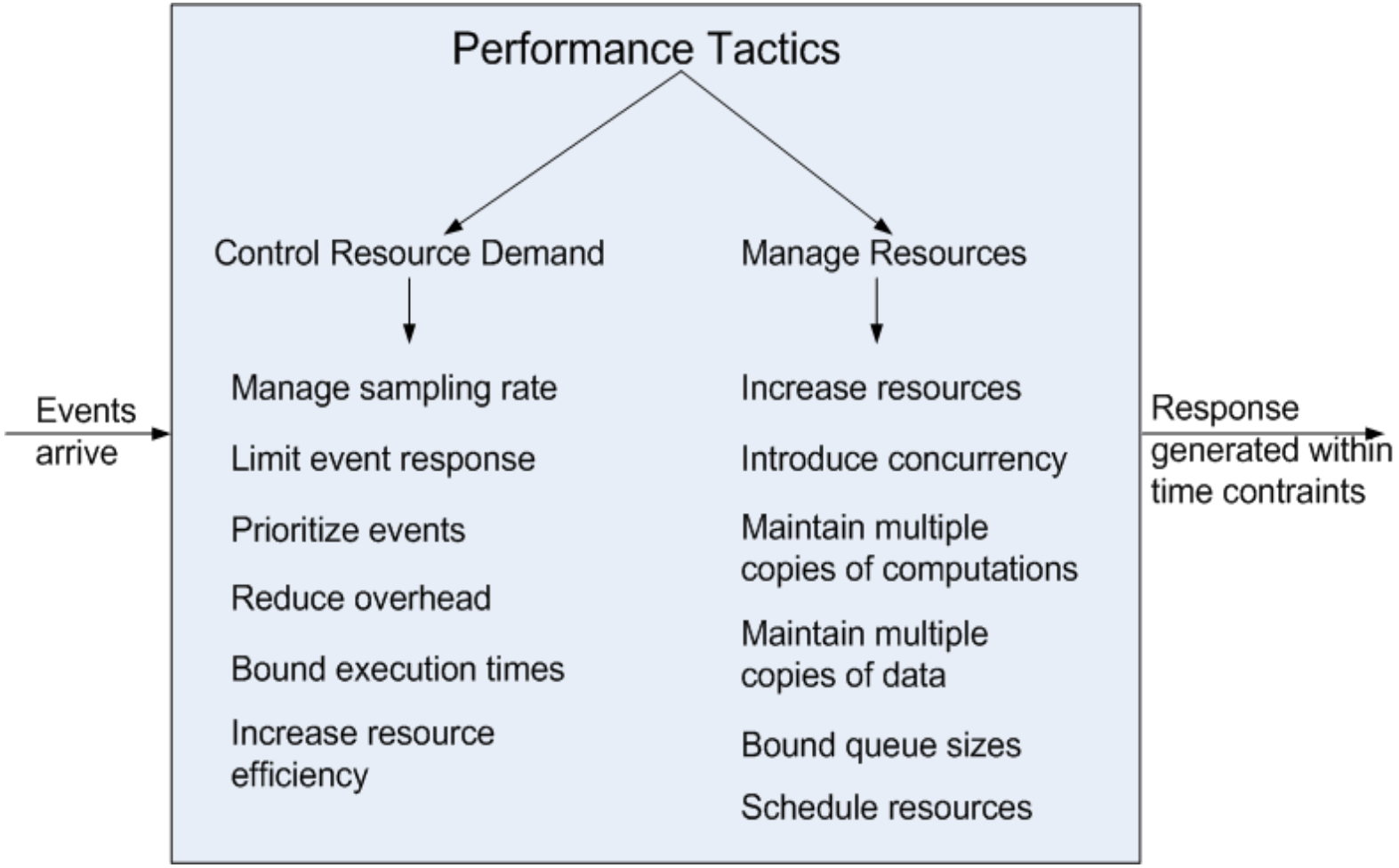
Goal of Performance Tactics

Tactics to control Performance have as their goal to generate a response to an event arriving at the system within some time-based constraint.

Goal of Performance Tactics



Performance Tactics



Control Resource Demand

Manage Sampling Rate: If it is possible to reduce the sampling frequency at which a stream of data is captured, then demand can be reduced, typically with some loss of fidelity.

Limit Event Response: process events only up to a set maximum rate, thereby ensuring more predictable processing when the events are actually processed.

Prioritize Events: If not all events are equally important, you can impose a priority scheme that ranks events according to how important it is to service them.

Control Resource Demand

Reduce Overhead: The use of intermediaries (important for modifiability) increases the resources consumed in processing an event stream; removing them improves latency.

Bound Execution Times: Place a limit on how much execution time is used to respond to an event.

Increase Resource Efficiency: Improving the algorithms used in critical areas will decrease latency.

Manage Resources

Increase Resources: Faster processors, additional processors, additional memory, and faster networks all have the potential for reducing latency.

Increase Concurrency: If requests can be processed in parallel, the blocked time can be reduced. Concurrency can be introduced by processing different streams of events on different threads or by creating additional threads to process different sets of activities.

Maintain Multiple Copies of Computations: The purpose of replicas is to reduce the contention that would occur if all computations took place on a single server.

Manage Resources

Maintain Multiple Copies of Data: keeping copies of data (possibly one a subset of the other) on storage with different access speeds.

Bound Queue Sizes: control the maximum number of queued arrivals and consequently the resources used to process the arrivals.

Schedule Resources: When there is contention for a resource, the resource must be scheduled.

Design Checklist for Performance

Allocation of Responsi-bilities	Determine the system's responsibilities that will involve heavy loading, have time-critical response requirements, are heavily used, or impact portions of the system where heavy loads or time critical events occur. For those responsibilities, identify • processing requirements of each responsibility and determine whether they may cause bottlenecks • additional responsibilities to recognize and process requests appropriately including • Responsibilities that result from a thread of control crossing process or processor boundaries. • Responsibilities to manage the threads of control — allocation and de-allocation of threads, maintaining thread pools, and so forth. • Responsibilities for scheduling shared resources or managing performance-related artifacts such as queues, buffers, and caches. For the responsibilities and resources you identified, ensure that the required performance response can be met (perhaps by building a performance model to help in the evaluation).

Design Checklist for Performance

Coordination Model	Determine the elements of the system that must coordinate with each other—directly or indirectly—and choose communication and coordination mechanisms that . supports any introduced concurrency (for example, is it thread-safe?), event priorization, or scheduling strategy . ensures that the required performance response can be delivered . can capture periodic, stochastic, or sporadic event arrivals, as needed . have the appropriate properties of the communication mechanisms, for example, stateful, stateless, synchronous, asynchronous, guaranteed delivery, throughput, or latency.

Design Checklist for Performance

Data Model	Determine those portions of the data model that will be heavily loaded, have time critical response requirements, are heavily used, or impact portions of the system where heavy loads or time critical events occur. For those data abstractions, determine • whether maintaining multiple copies of key data would benefit performance • partitioning data would benefit performance • whether reducing the processing requirements for the creation, initialization, persistence, manipulation, translation, or destruction of the enumerated data abstractions is possible • whether adding resources to reduce bottlenecks for the creation, initialization, persistence, manipulation, translation, or destruction of the enumerated data abstractions is feasible.

Design Checklist for Performance

Mapping Among Architectural Elements	Where heavy network loading will occur, determine whether co-locating some components will reduce loading and improve overall efficiency. Ensure that components with heavy computation requirements are assigned to processors with the most processing capacity. Determine where introducing concurrency (that is, allocating a piece of functionality to two or more copies of a component running simultaneously) is feasible and has a significant positive effect on performance. Determine whether the choice of threads of control and their associated responsibilities introduces bottlenecks.

Design Checklist for Performance

Resource Management	Determine which resources in your system are critical for performance. For these resources ensure they will be monitored and managed under normal and overloaded system operation. For example . system elements that need to be aware of, and manage, time and other performance-critical resources . process/thread models . prioritization of resources and access to resources . scheduling and locking strategies . deploying additional resources on demand to meet increased loads

Design Checklist for Performance

Binding Time	For each element that will be bound after compile time, determine the . time necessary to complete the binding . additional overhead introduced by using the late binding mechanism Ensure that these values do not pose unacceptable performance penalties on the system.

Design Checklist for Performance

Choice of Technology	Will your choice of technology let you set and meet hard real time deadlines? Do you know its characteristics under load and its limits? Does your choice of technology give you the ability to set . scheduling policy . priorities . policies for reducing demand . allocation of portions of the technology to processors . other performance related parameters Does your choice of technology introduce excessive overhead for heavily used operations?

Summary

Performance is about the management of system resources in the face of particular types of demand to achieve acceptable timing behavior.

Performance can be measured in terms of throughput and latency for both interactive and embedded real time systems.

Performance can be improved by reducing demand or by managing resources more appropriately.