BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Software Architectures

SECLZG651/SSCLZG653

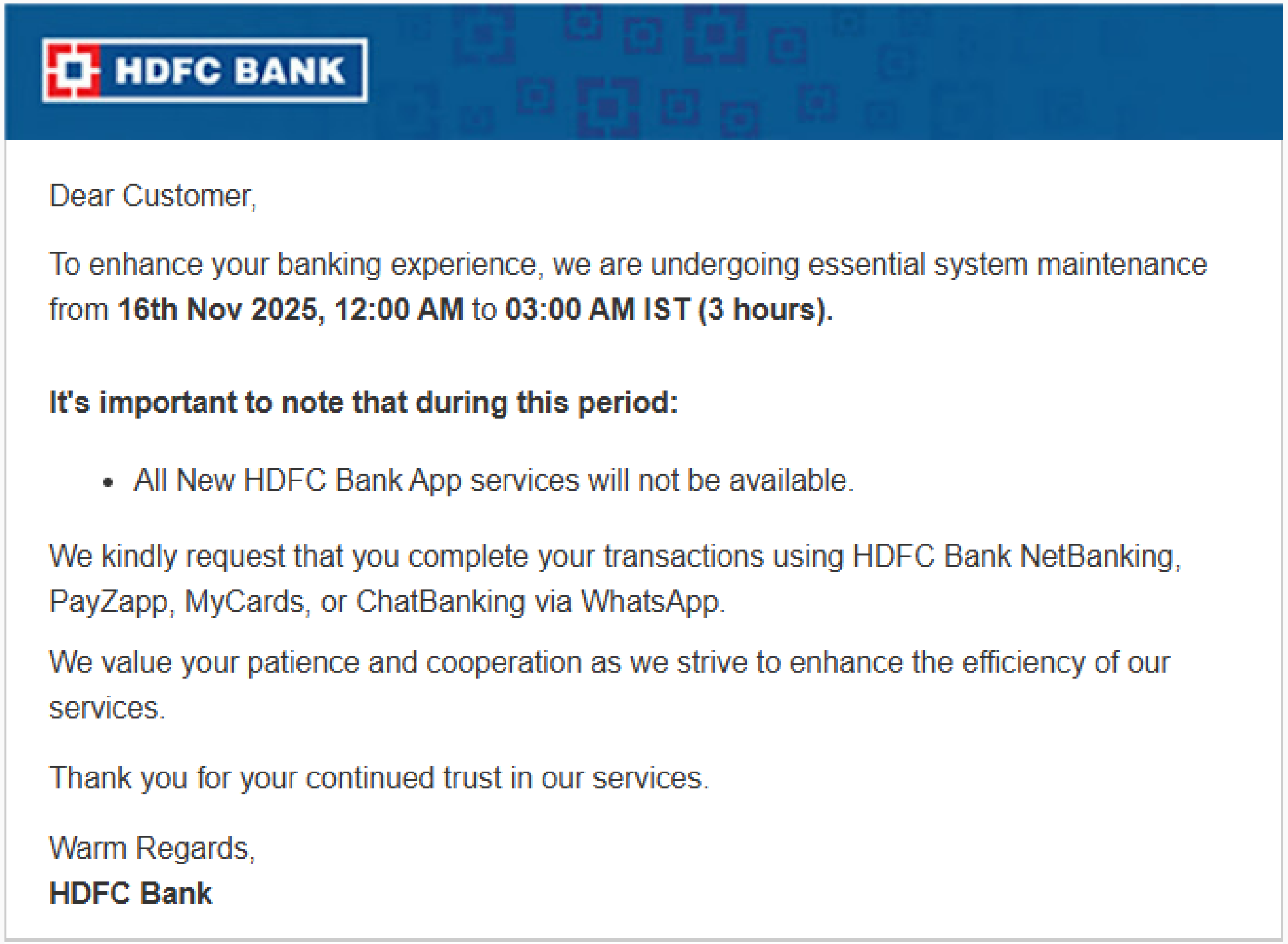
Parthasarathy

SECLZG651/SSCLZG653 – CS#4
Quality Attributes

Agenda for CS #4

1) Recap of CS3
2) Complete Availability and its tactics
3) Modifiability and its tactics
4) Performance and its tactics
5) Security and its tactics
6) Testability and its tactics (Next Class)
7) Interoperability and its tactics (Next Class)
5) Q&A!

Availability Example!



Dear Customer,

To enhance your banking experience, we are undergoing essential system maintenance from **16th Nov 2025, 12:00 AM** to **03:00 AM IST (3 hours).**

It's important to note that during this period:

- All New HDFC Bank App services will not be available.

We kindly request that you complete your transactions using HDFC Bank NetBanking, PayZapp, MyCards, or ChatBanking via WhatsApp.

We value your patience and cooperation as we strive to enhance the efficiency of our services.

Thank you for your continued trust in our services.

Warm Regards,
HDFC Bank

Hardware vs Software Reliability Metrics

- Hardware metrics are not suitable for software since its metrics are based on notion of component failure
- Software failures are often design failures
- Often the system is available after the failure has occurred
- Hardware components can wear out

Software Reliability Metrics

- Reliability metrics are units of measure for system reliability
- System reliability is measured by counting the number of operational failures and relating these to demands made on the system at the time of failure
- A long-term measurement program is required to assess the reliability of critical systems

Time Units

- Raw Execution Time
    - non-stop system
- Calendar Time
    - If the system has regular usage patterns
- Number of Transactions
    - demand type transaction systems

Reliability Metric POFOD

- Probability Of Failure On Demand (POFOD):
  - Likelihood that system will fail when a request is made.
  - E.g., POFOD of 0.001 means that 1 in 1000 requests may result in failure.
- Any failure is important; doesn’t matter how many if the failure > 0
- Relevant for safety-critical systems

Reliability Metric ROCOF & MTTF

- Rate Of Occurrence Of Failure (ROCOF):
  - Frequency of occurrence of failures.
  - E.g., ROCOF of 0.02 means 2 failures are likely in each 100 time units.
- Relevant for transaction processing systems
- Mean Time To Failure (MTTF):
  - Measure of time between failures.
  - E.g., MTTF of 500 means an average of 500 time units passes between failures.
- Relevant for systems with long transactions

Rate of Fault Occurrence

- Reflects rate of failure in the system
- Useful when system has to process a large number of similar requests that are relatively frequent
- Relevant for operating systems and transaction processing systems

Mean Time to Failure

- Measures time between observable system failures
- For stable systems MTTF = 1/ROCOF
- Relevant for systems when individual transactions take lots of processing time (e.g. CAD or WP systems)

Failure Consequences

- When specifying reliability both the number of failures and the consequences of each matter
- Failures with serious consequences are more damaging than those where repair and recovery is straightforward
- In some cases, different reliability specifications may be defined for different failure types

Building Reliability Specification

- For each sub-system analyze consequences of possible system failures
- From system failure analysis partition failure into appropriate classes
- For each class send out the appropriate reliability metric

Failure Class	Example	Metric
Permanent Non-corrupting	ATM fails to operate with any card, must restart to correct	ROCOF = .0001 Time unit = days
Transient Non-corrupting	Magnetic stripe can't be read on undamaged card	POFOD = .0001 Time unit = transactions

Activity [10 min]

Scenario 1:

*You're designing a self-service **online railway booking system** for passengers of all ages and digital literacy levels, including first-time users.*

- Which **usability tactics** would you apply in the architecture/design, and why?

Scenario 2:

You're designing a hospital patient-monitoring system that tracks and alerts about vitals in ICU. Downtime is not acceptable.

- Which **availability tactics** would you apply, and why?

Modifiability and its Tactics

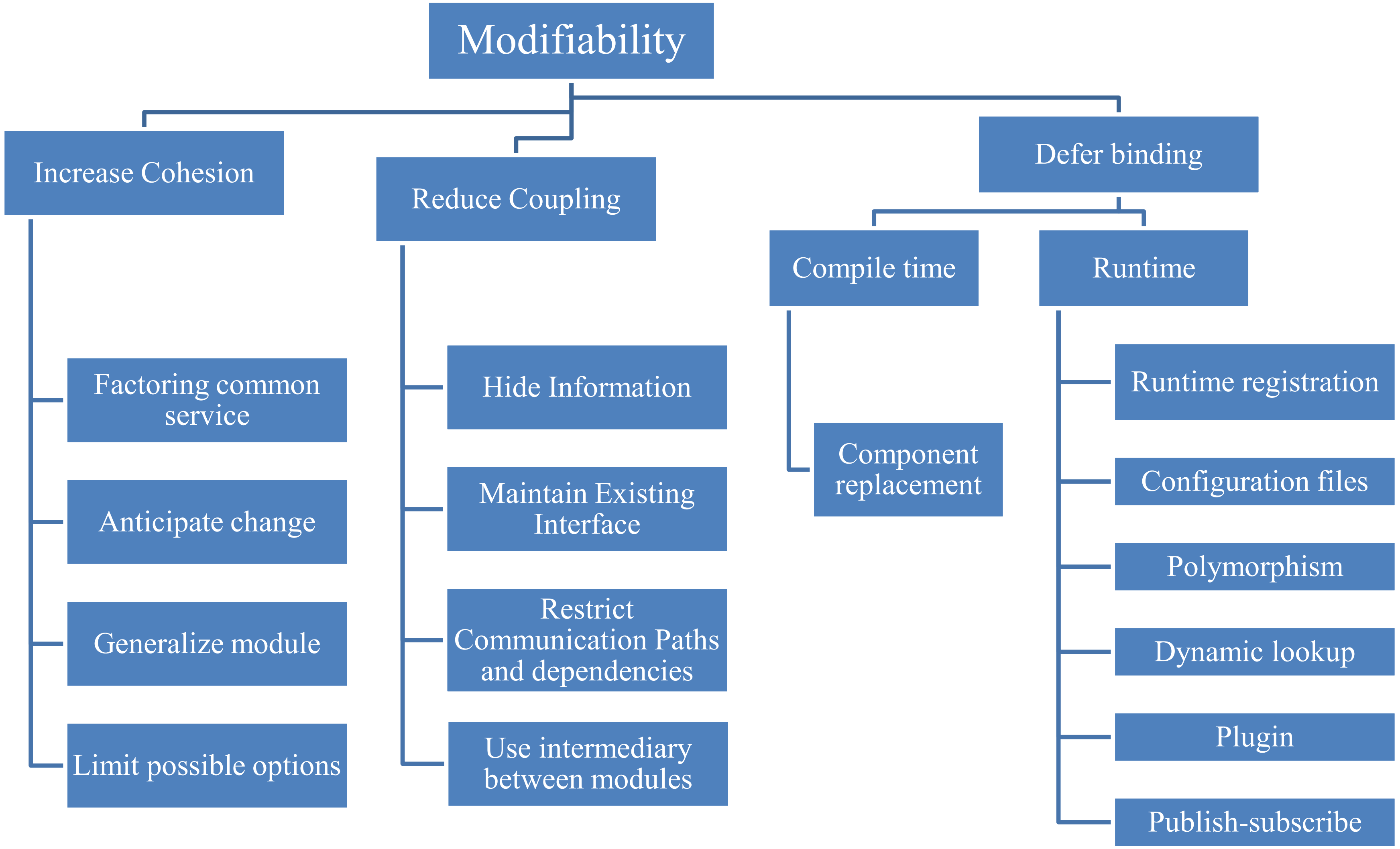
Modifiability

- Ability to Modify the system based on the change in requirement so that
  - the time and cost to implement is optimal
  - Impact of modification such as testing, deployment, and change management is minimal
- When do you want to introduce modifiability?
  - If (cost of modification w/o modifiability mechanism in place) > (cost of modification with modifiability in place)+ Cost of installing the mechanism

Modifiability Scenarios

WHO	STIMULUS	IMPACTED PART	MITIGATING ACTION	MEASURABLE RESPONSE
• Enduser • Developer • SysAdm	They want to modify ➢ Functionality ➢ Add, modify, delete ➢ Quality ➢ Capacity	UI, platform or System • Runtime • Compile time • Design time • Build time	When fault occurs it should do one or more of ✓ Locate (Impact analysis) ✓ Modify ✓ Test ✓ Deploy again	• Volume of the impact of the primary system (number, size) • Cost of modification • Time and effort • Extent of impact to other systems • New defects introduced
Developer	Tries to change UI	Artifact – Code Environment: Design time	Changes made and unit test done	Completed in 4 hours

Modifiability Tactics



Dependency between two modules (B→ A)

Syntax (compile+runtime)

- Data : B uses the type/format of the data created by A
- Service B uses the API signature provided by A

Semantics of A

- Data: Semantics of data created by A should be consistent with the assumption made by B
- Service: Same …..

Sequence

- Data -- data packets created by A should maintain the order as understood by B
- Control– A must execute 5ms before B. Or an API of A can be called only after calling another API

Interface identity

- Handle of A must be consistent with B, if A maintains multiple interfaces

Location of A

- B may assume that A is in-process or in a different process, hardware..

Quality of service/data provided by A

- Data quality produced by A must be > some accuracy for B to work

Existence of A

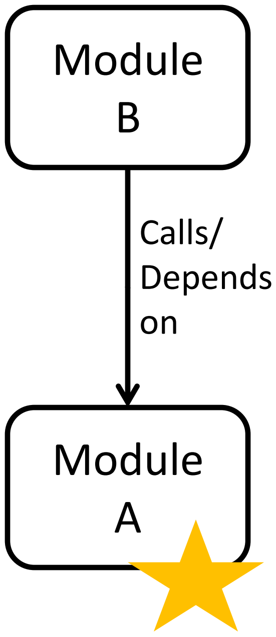
- B may assume that A must exist when B is calling A

Resource behavior of A

- B may assume that both use same memory
- B needs to reserve a resource owned by A

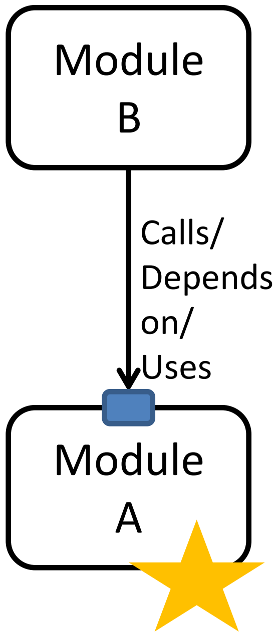
Localize Modifications

1. Factoring common service
   - Common services through a specialized module (only implementing module should be impacted)
     - Heavily used in application framework and middleware
   - Reduce Coupling and increase cohesion
2. Anticipate Expected Changes
   - Quite difficult to anticipate, hence should be coupled with previous one
   - Allow extension points to accommodate changes
3. Generalize the Module
   - Allowing it to perform broader range of functions
   - Externalize configuration parameters (could be described in a language like XML)
     - The module reconfigure itself based on the configurable parameters
   - Externalize business rules
4. Limit Possible options
   - Do not keep too many options for modules that are part of the framework



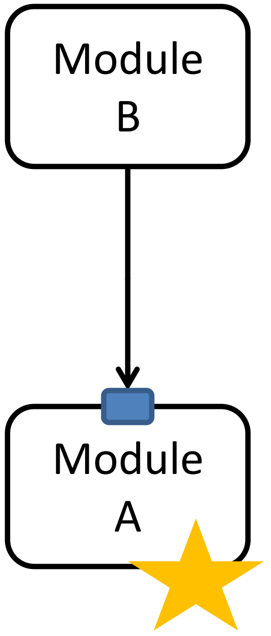
Prevent Ripple Effect Tactics

1. Hide Information (of A)
   - Use interfaces, allow published API based calls only
2. Maintain existing Interface (of A)
   - Add new interfaces if needed
   - Use Wrapper, adapter to maintain same interface
   - Use stub
3. Restrict Communication Paths
   - Too many modules should not depend on A
4. Use an intermediary between B and A
   - Data
     - Repository from which B can read data created by A (blackboard pattern)
     - Publish-subscribe (data flowing through a central hub)
     - MVC pattern
   - Service: Use of design patterns like bridge, mediator, strategy, proxy
   - Identity of A – Use broker pattern which deals with A's identity
   - Location of A – Use naming service to discover A
   - Existence of A- Use factory pattern



Defer Binding Time

1. Runtime registration of A (plug n play)
   - Use of pub-sub
2. Configuration Files
   - To take decisions during startup
3. Polymorphism
   - Late binding of method call
4. Component Replacement (for A)
   - during load time such as classloader
5. Adherence to a defined protocol- runtime binding of independent processes



Design Checklist- Modifiability

Allocation of Responsibilities

- Determine the types of changes that can come due to technical, customer or business
- Determine what sort of additional features are required to handle the change
- Determine which existing features are impacted by the change

Coordination Model

- For those where modifiability is a concern, use techniques to reduce coupling
  - Use publish-subscribe, use enterprise service bus
- Identify
  - which features can change at runtime
  - which devices, communication paths or protocols can change at runtime
- And make sure that such changes have limited impact on the system

Design checklist- Modifiability

Data Model

- For the anticipated changes, decide which data elements will be impacted, and the nature of impact (creation, modification, deletion, persistence, translation)
- Group data elements that are likely to change together
- Design to ensure that changes have minimal impact to the rest of the system

Resource Management

- Determine how addition, deletion or modification of a feature or a quality attribute cause
  - New resources to be used, or affect resource usage
  - Changing of resource usage limits
- Ensure that the resources after modification are sufficient to meet the system requirement
- Write Resource manager module that encapsulates resource usage policies

Design Checklist- Modifiability

Binding

- Determine the latest time at which the anticipated change is required
- Choose a defer binding if possible
- Try to avoid too many binding choices

Choice of Technology

- Evaluate the technology that can handle modifications with least impact (e.g. enterprise service bus)
- Watch for vendor lock-in

Activity

Scenario:

You are working on the ***backend system of an e-commerce platform****. The business team now wants to support a new category of digital products (e.g., e-books, online courses) in addition to physical goods — with different payment and delivery rules.*

What **two architectural tactics** would you apply to ensure the system can be **modified easily** to support this new requirement, without breaking existing functionality?

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Performance and its Tactics

What is Performance?

- Software system’s ability to meet timing requirements when it responds to an event
- Events are
  - interrupts, messages, requests from users or other systems
  - clock events marking the passage of time
- The system, or some element of the system, must **respond to them** in time

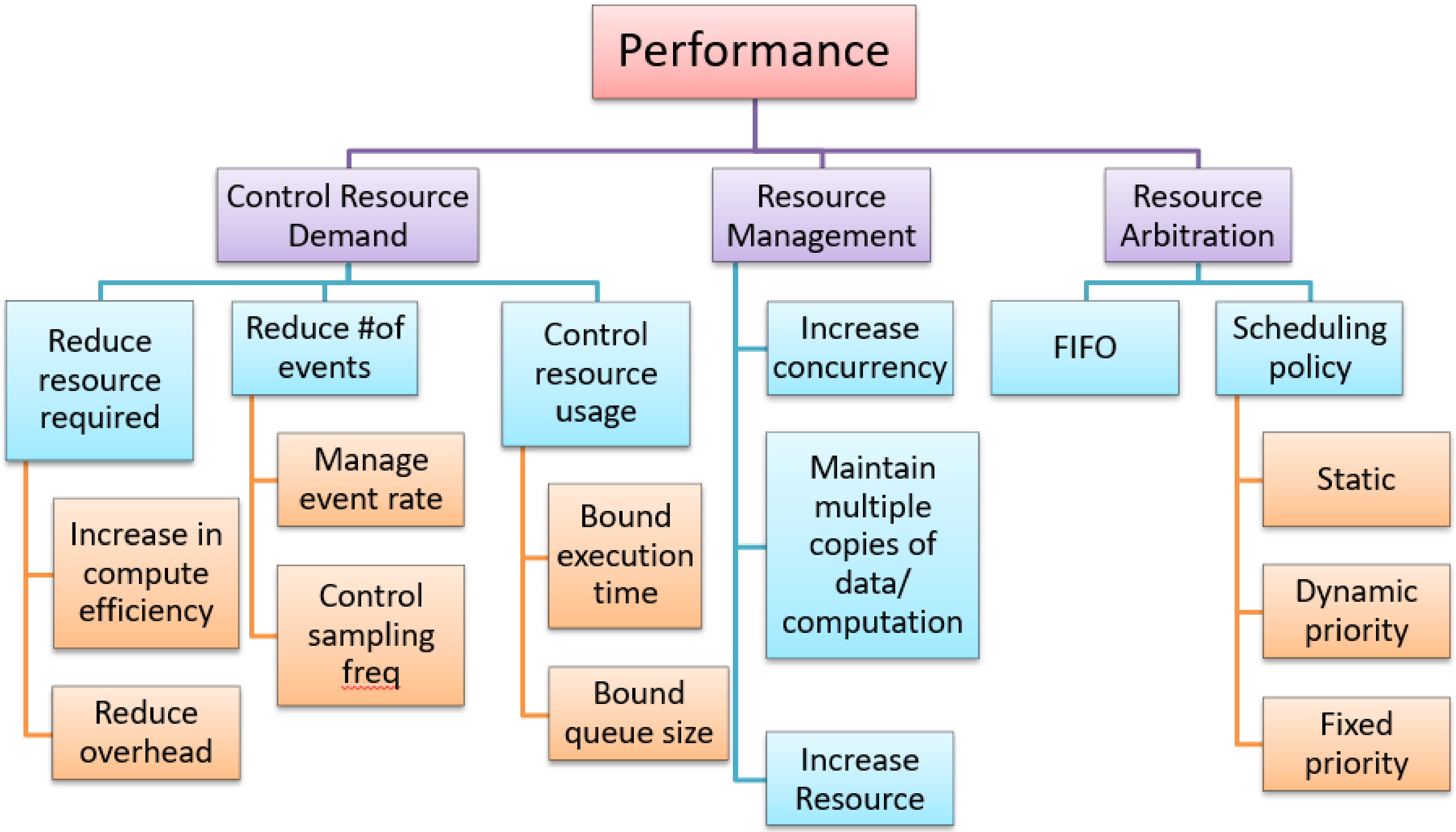
Performance Scenarios

WHO	STIMULUS	IMPACTED PART	MITIGATING ACTION	MEASURABLE RESPONSE
• External • Internal	**• Events that are** • Periodic • Sporadic • Stochastic (with avg 1000 txn/min)	System • Running in normal mode • Overloaded	✓ Processes event in time ✓ Changes the service-level	• Latency or deadline by which it must be processed • Throughput- #of events processed/time • Jitter- variation • Miss rate • Data loss
Users	Submits transactions and expect response in 3s		Transactions processed	Average latency of 2s

Events and Responses

- Periodic- comes at regular intervals (real time systems)
- Stochastic- comes randomly following a probability distribution (eCommerce website)
- Sporadic- keyboard event from human
- Latency- time between arrival of stimulus and system response
- Throughput- number of txn processed/unit of time
- Jitter- allowable variation in latency
- #events not processed

Performance Tactics



Why System fails to Respond?

- Resource Consumption
  - CPU, memory, data store, network communication
  - A buffer may be sequentially accessed in a critical section
  - There may be a workflow of tasks one of which may be choked with request

- Blocking of computation time
  - Resource contention
  - Availability of a resource
  - Deadlock due to dependency of resource

Control Resource Demand

- Increase Computation Efficiency: Improving the algorithms used in performance critical areas
- Reduce Overhead
  - Reduce resource consumption when not needed
    - Use of local objects instead of RMI calls
    - Local interface in EJB 3.0
  - Remove intermediaries (conflicts with modifiability)
- Manage
  - event rate: If you have control, don't sample too many events (e.g. sampling environmental data)
  - sampling time: If you don't have control, sample them at a lower speed, leading to loss of request
- Bound
  - Execution: Decide how much time should be given on an event. E.g. iteration bound on a data-dependent algorithm
  - Queue size: Controls maximum number of queued arrivals

Manage Resources

- Increase Resources(infrastructure)
  - Faster processors, additional processors, additional memory, and faster networks
- Increase Concurrency
  - If possible, process requests in parallel
  - Process different streams of events on different threads
  - Create additional threads to process different sets of activities
- Multiple copies
  - Computations : so that it can be performed faster (client-server), MapReduce computation
  - Data:
    - use of cache for faster access and reduce contention
    - Hadoop maintains data copies to avoid data-transfer and improve data locality

Resource Arbitration

- Resources are scheduled to reduce contention
  - Processors, buffer, network
  - Architect needs to choose the right scheduling strategy
- FIFO
- Fixed Priority
  - Semantic importance
    - Domain specific logic such as request from a privileged class gets higher priority
  - Deadline monotonic (shortest job first)
- Dynamic priority
  - Round robin
  - Earliest deadline first- the job which has earliest deadline to complete
- Static scheduling
  - Also, pre-emptive scheduling policy

Design Checklist for a Quality Attribute

- Allocate responsibility
  - Modules can take care of the required quality requirement
- Manage Data
  - Identify the portion of the data that needs to be managed for this quality attribute
  - Plan for various data design w.r.t. the quality attribute
- Resource Management Planning
  - How infrastructure should be monitored, tuned, deployed to address the quality concern
- Manage Coordination
  - Plan how system elements communicate and coordinate
- Binding

Performance- Design Checklist- Allocate responsibilities

- Identify which features may involve or cause
  - Heavy workload
  - Time-critical response
- Identify which part of the system that's heavily used
- For these, analyze the scenarios that can result in performance bottleneck
- Furthermore--
  - Assign Responsibilities related to threads of control — allocation and de-allocation of threads, maintaining thread pools, and so forth
  - Assign responsibilities that will schedule shared resources or appropriately select, manage performance-related artifacts such as queues, buffers, and caches

Performance- Design Checklist- Manage Data

- Identify the data that’s involved in time critical response requirements, heavily used, massive size that needs to be loaded etc. For those data 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.

Performance- Design Checklist- Manage Coordination

- Look for the possibility of introducing concurrency (and obviously pay attention to thread-safety), event priorization, or scheduling strategy
  - Will this strategy have a significant positive effect on performance? Check
  - Determine whether the choice of threads of control and their associated responsibilities introduces bottlenecks

- Consider appropriate mechanisms for example
  - stateful, stateless, synchronous, asynchronous, guaranteed delivery

Performance Design Checklist- Resource Management

- Determine which resources (CPU, memory) in your system are critical for performance.
  - Ensure they will be monitored and managed under normal and overloaded system operation.
- Plan for mitigating actions early, for instance
  - 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.
- Prioritization of resources and access to resources
  - scheduling and locking strategies
- Deploying additional resources on demand to meet increased loads
  - Typically possible in a Cloud and virtualized scenario

Performance Design checklist- Binding

- 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.

Performance Design Checklist- Technology choice

- Choice of technology is often governed by the organization mandate (enterprise architecture)
- Find out if the chosen technology will let you set and meet 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
- Does your choice of technology introduce excessive overhead?

Thank You for your time & attention !

Contact : parthasarathypd@wilp.bits-pilani.ac.in