Section 1 – Software Architecture

Definitions of Software Architecture

- ***Architectural Design: The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.***
  - IEEE Glossary

Definitions of Software Architecture

- ***The Software Architecture of a program or a computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.***
  - ***Software Architecture in Practice, 2nd Edition, Len Bass, Paul Clements, Rick Kazman***

- **Externally visible properties refer to**
  - Provided services
  - Performance characteristics
  - Fault Handling
  - Shared Resource usage
- The point of seeing a system through its external interfaces is to abstract out irrelevant detail and provide enough information for decision making and risk reduction.

Definitions of Software Architecture

- ***Software Architecture is the "structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time".***
  - ***Garlan and Perry, guest editorial to the IEEE transactions on Software Engineering, April 1995***

- According to this definition, architecture evolves and is not necessarily static.

Definitions of Software Architecture

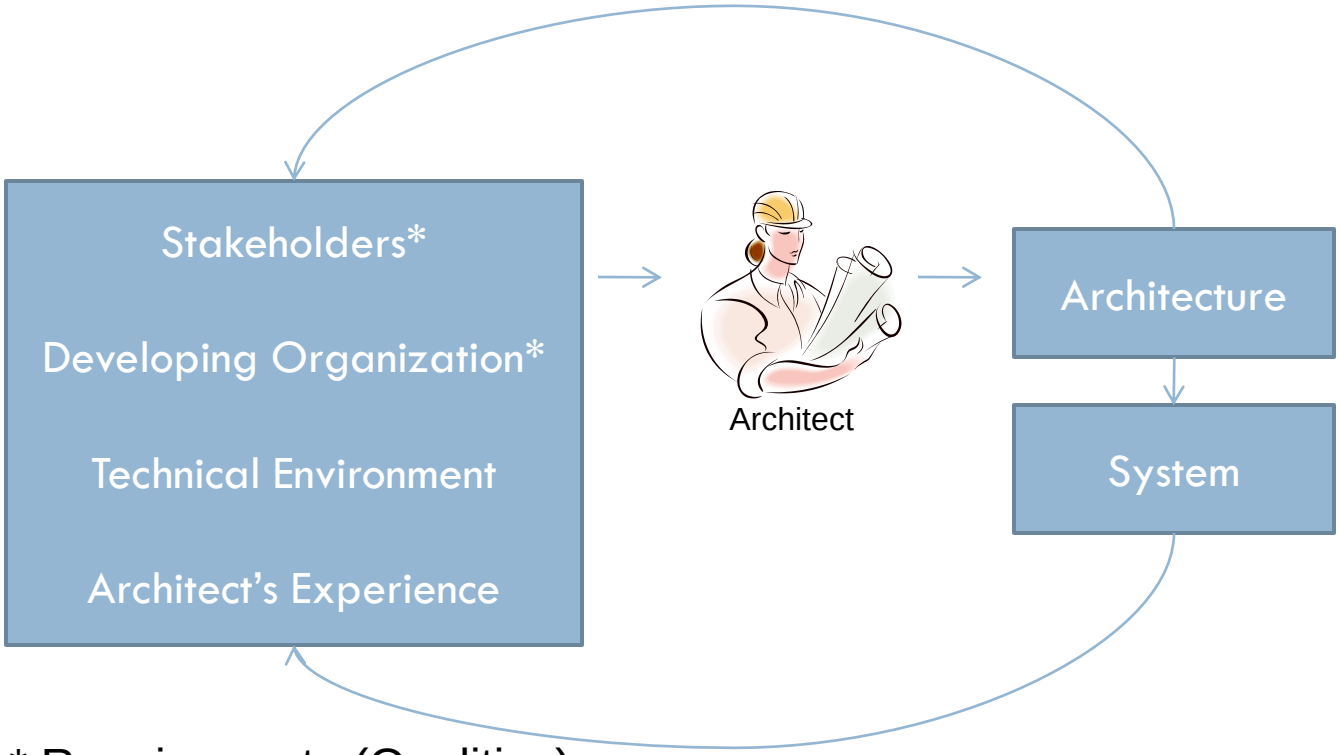
- ***The architecture of a system defines the system in terms of computational components and interaction between those components.***
- ***Components are such things as clients and servers, databases, filters and layers in a hierarchal system.***
- ***Interactions among components at this level of design can be simple and familiar such as procedure call and shared variable access. But they can also be semantically rich as client-server protocols, database accessing protocols, asynchronous event multicast, and piped streams.***
  - David Garlan and Mary Shaw

What determines Software Architecture?

- Is Software Architecture a result and outcome of just customer requirements?

- **Case Study:** In 1620, Sweden was at war with Poland. The Swedish King ordered the construction of a large warship, called the *Vasa*. The Vasa was envisioned to be the most powerful warship of the day, the largest in size, and with two levels of gun ports, something never seen before. The architect, under pressure from the king to deliver the ship on time, extrapolated the single gun port design to build the double gun port ship.
- The ship was built by the ship builders according to the architects design. In 1628, the ship rolled out of the harbor and gave a gun salute on which it rolled over. Water poured in through the gun ports and the ship sank. The report later said, that the ship was “badly proportioned”.

- Software architecture is a result of **technical, business,** and **social** influences.



* Requirements (Qualities)

What makes a good architecture

- The architecture should be the product of a single architect or a small group of architects with an identified leader.
- The architect should have the functional requirements for the system, and a prioritized list of ***quality attributes*** like security or modifiability etc. (We will be studying in detail about quality attributes).
- The architecture should be **well** documented. With at least one static, and one dynamic **view**. (We will be studying about architecture views later)
- The architecture should be circulated and reviewed with stakeholders.
- The architecture should be analyzed for applicable quantitative measures (such as maximum throughput) and formally evaluated for ***quality attributes.***

Structural Rules of Thumb

- ***The architecture should feature well-defined modules whose functional responsibilities are allocated on the principles of information hiding and separation of concerns.***
- The information-hiding modules should include those that encapsulate idiosyncrasies of the computing infrastructure, thus insulating the bulk of the software from change should the infrastructure change.

- ***Each module should have a well defined interface that encapsulates or "hides" changeable aspects (such as implementation strategies and data structure choices) from the other software that uses its facilities.***
- These interfaces should allow their respective development teams to work largely independently of each other.

- ***Quality Attributes should be achieved using well-known architectural tactics specific to each attribute.(we will be studying these later)***

- ***The architecture should never depend on a particular version of a commercial product or tool.***

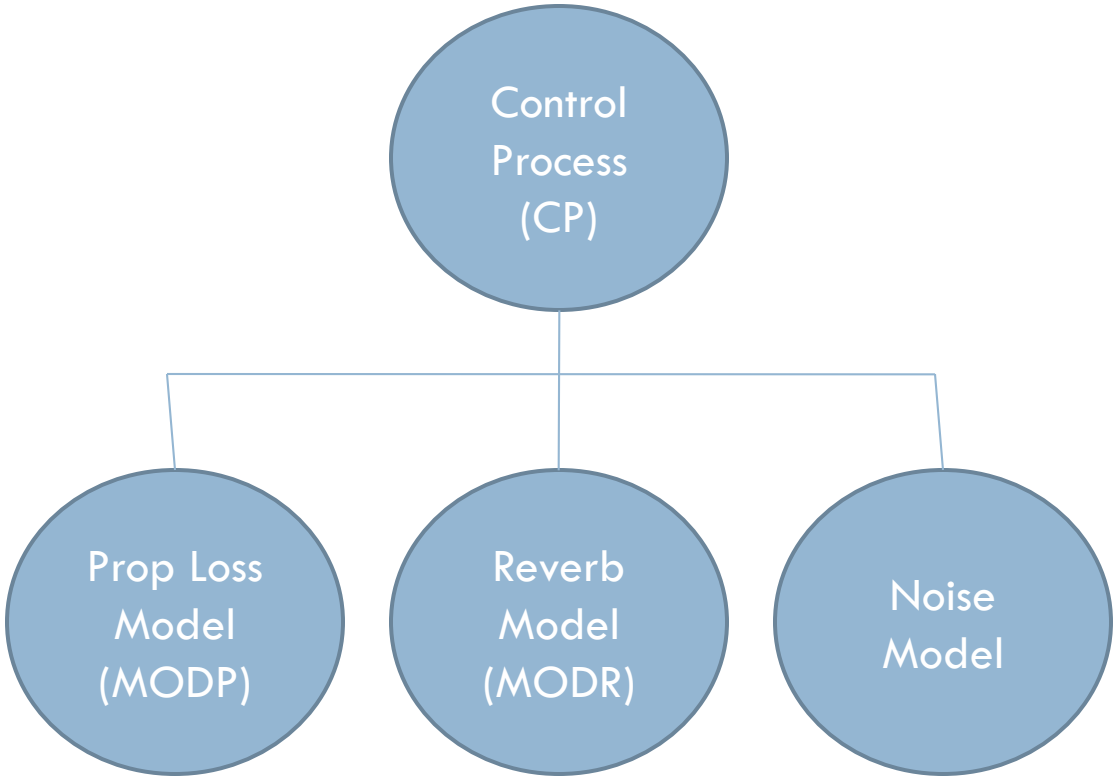
- If it depends upon a particular commercial product, it should be structured such that changing to a different product is straightforward and inexpensive.

- ***Modules that produce data should be separate from modules that consume data.***

- This tends to increase modifiability became changes are often confined to either the production or the consumption side. If new data is added, both sides will have to change, but the separation allows for a staged upgrade.

SA Structure and Views

Take a look at a top level architecture of an underwater acoustic simulation.



- What can we observe from this diagram?
  - The system consists of four elements.
  - Three of the four elements may have something in common because they are positioned next to each other.
  - All elements have some sort of relationship with each other since they are fully connected.

- Does the diagram show a software architecture?
- Lets see what we cannot determine from this diagram..
- ***What is the nature of the elements?***
  - What is the significance of their separation?
  - Are they separate processes?
  - Do they run on separate processors?
  - Do they run at separate times?
  - Are they objects, tasks, modules? Classes? Functions? What are they?

- ***What are the responsibilities of the elements?***
  - What are the functionalities of the elements?
- ***What is the significance of the connections?***
  - Do the elements communicate with each other?
  - Control each other?
  - Send data ?
  - Use each other?
  - Invoke each other?
  - Share some information hiding secrets ?

- ***What is the significance of the layout?***
  - Why is CP on a separate level?
  - Does it call the other three elements?
  - Are the others allowed to call it?

- Answers of all these questions are vital for the implementation team hence need to be discussed, defined and documented.
- Complete definition of ***structures*** and ***views*** are important.

Software Architecture Structures

- Software architectures can be broadly classified into three major categories.
  - ***Module Structures***
    - Answers how a system is to be structured as a set of code units.
  - ***Component and Connector Structures***
    - How is the system to be structured as a set of elements that have runtime behavior (components) and interactions(connectors)
  - ***Allocation Structures***
    - How the system is related to non software structures in its environment (CPUs, file systems, networks, development teams etc).

Structures - Module

- Module based structures include the following
  - **Module**
  - **Uses**
  - **Layered**
  - **Class**

Structures – Module - Decomposition

- Units are related to each other by the ***"is a sub module of"*** relationship.
- Modules are broken down until they are small enough to be understood.
- Modules have associated artifacts such as interface design, code, test plans etc.
- Makes the system ***modifiable.***
- ***Examples: an engine is a sub module of a car. Spell checker is a sub module of MS word. ALU is a sub module of a CPU. Service layer is a sub module of UCP online system.***

Structures – Module -Uses

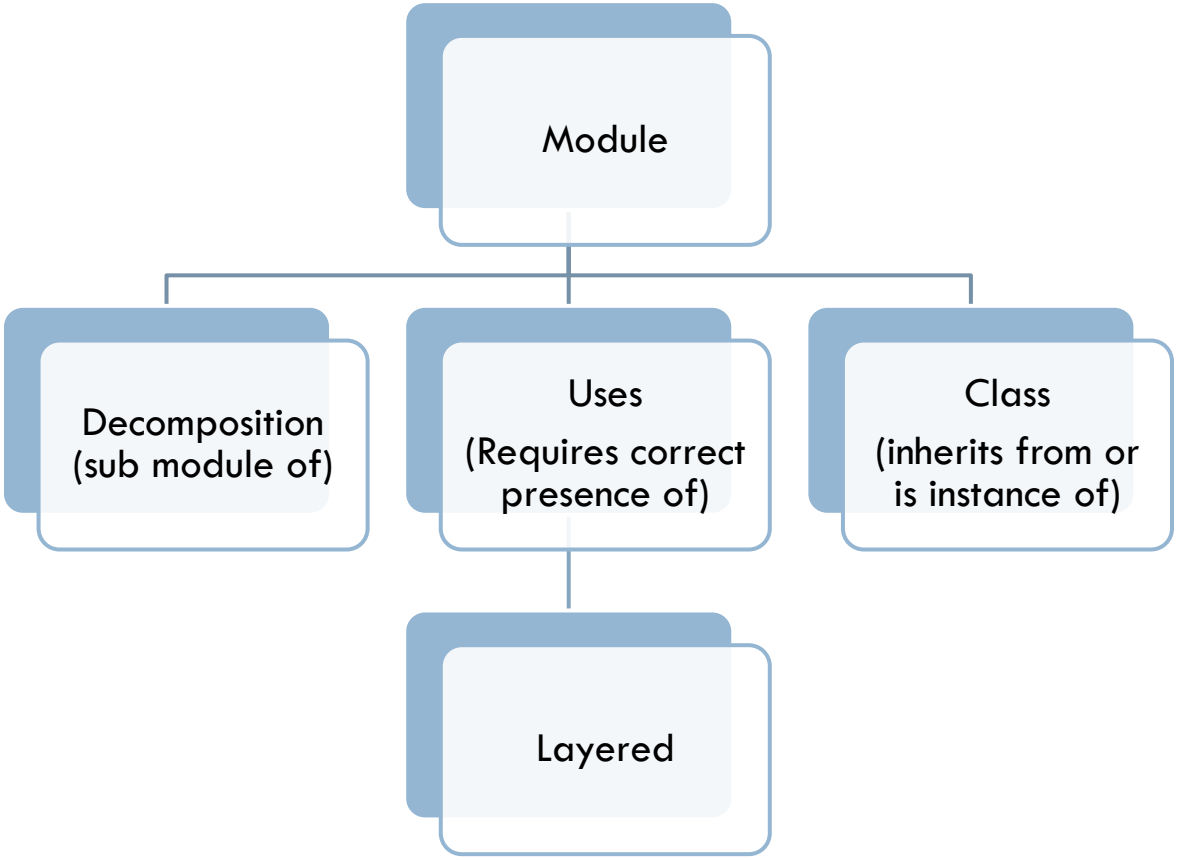
- Another type of division is the “uses” relationship.
- One structure uses the other if the correctness of the first requires the presence of a correct version of the second.
- **My java software uses the java virtual machine.**
- **CPU uses memory for retrieving instructions, operands etc.**
- **Service layer uses database layer to access data.**

Structures – Module -Layered

- When the uses relation are carefully controlled, a system of layers emerges.
- In a strictly layered system, layer n may only use the services of layer n-1.
- **Examples, OS, OSI model.**

Structures – Module -Class

- Inheritance and composition.


Module
Decomposition
(sub module of)
Uses
(Requires correct
presence of)
Class
(inherits from or
is instance of)
Layered

Structures – Component and Connector

- These structures include the following
  - Process or communicating processes
  - Concurrency
  - Shared Data or Repository
  - Client-Server

Structures – C&C - Process

- Deals with dynamic aspect of the running system.
- Units here are ***Processes, threads.***
- Are connected to each other.
- Relationship is: ***Runs concurrently with; may run concurrently with***

Structures – C&C - Concurrency

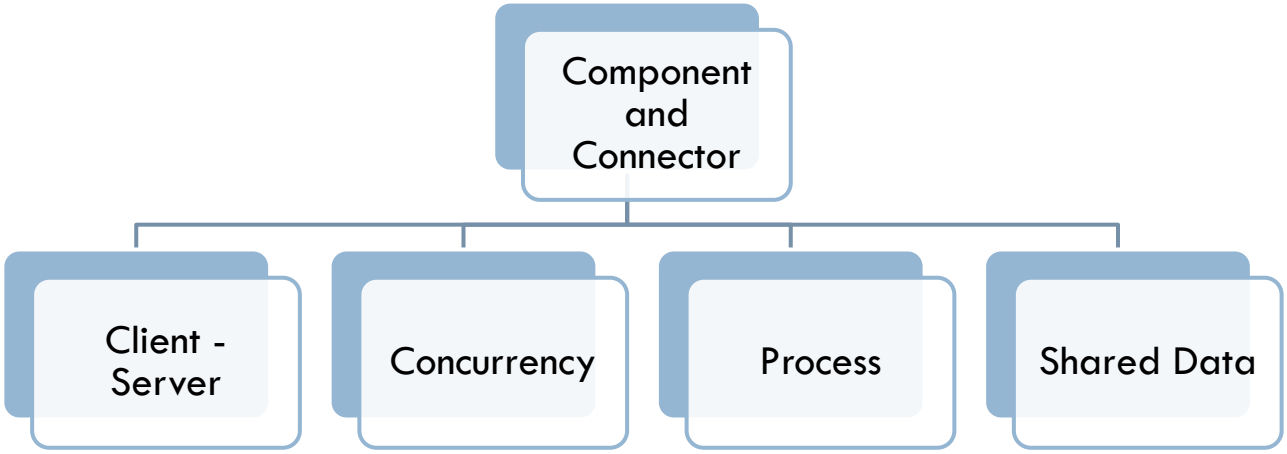
- The C&C structure allows the architect to determine opportunities for parallelism.
- In concurrency, the architect determines units that may run parallel to others in logical threads.
- Identify locations where resource contention may occur.
- Relationships are: ***Runs on the same logical thread***

Structures – C&C – Shared Data or Repository

- This structure comprises components and connectors that
  - Create persistent data
  - Store persistent data
  - Access persistent data
- Relationship is: ***Produces data, consumes data.***

Structures – C&C – Client Server

- The components in this type of structure are
  - Servers
  - Clients
- The connectors are
  - Protocols
  - Messages
- Benefits are ***separation of concerns***, and ***load balancing.***
- Relationships are: ***Communicates with, depends on***


Component and Connector
Client - Server
Concurrency
Process
Shared Data

Structures - Allocation

- Allocation Structures include the following
  - Deployment
  - Implementation
  - Work Assignment

Structures- Allocation - Deployment

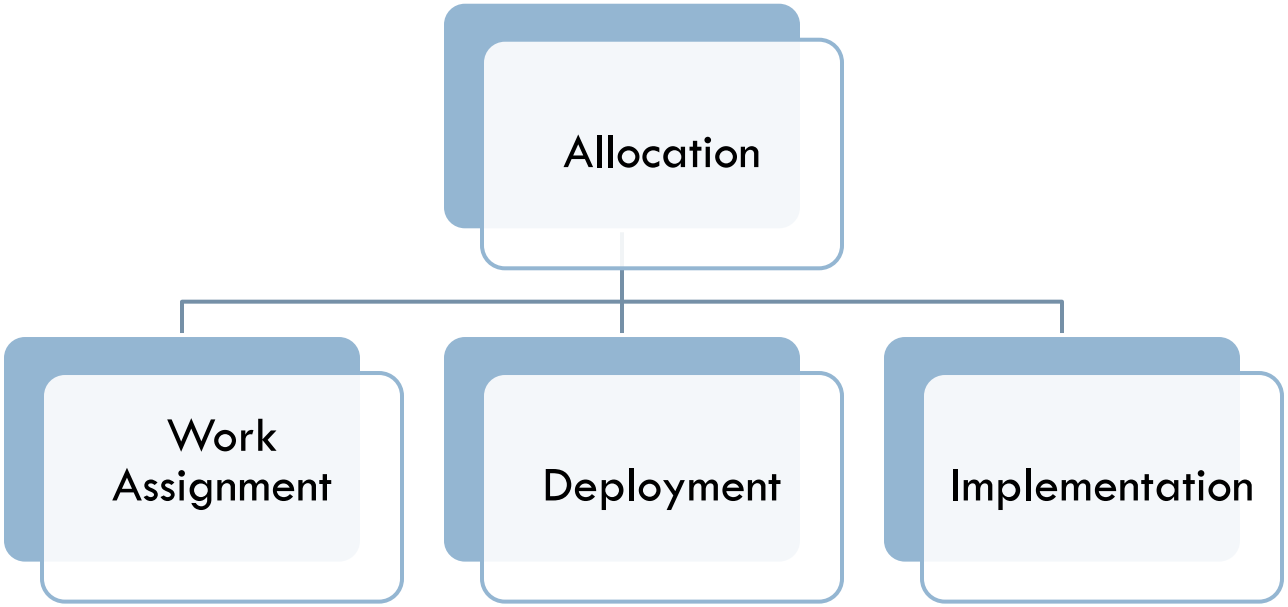
- Shows how a software is assigned to hardware processing and communication elements
- The elements are usually process (from the component and connector view)
- The allocation is to hardware entities such as processors, and communication pathways.
- Relationships are: ***Allocated to, migrates to***

Structures – Allocation - Implementation

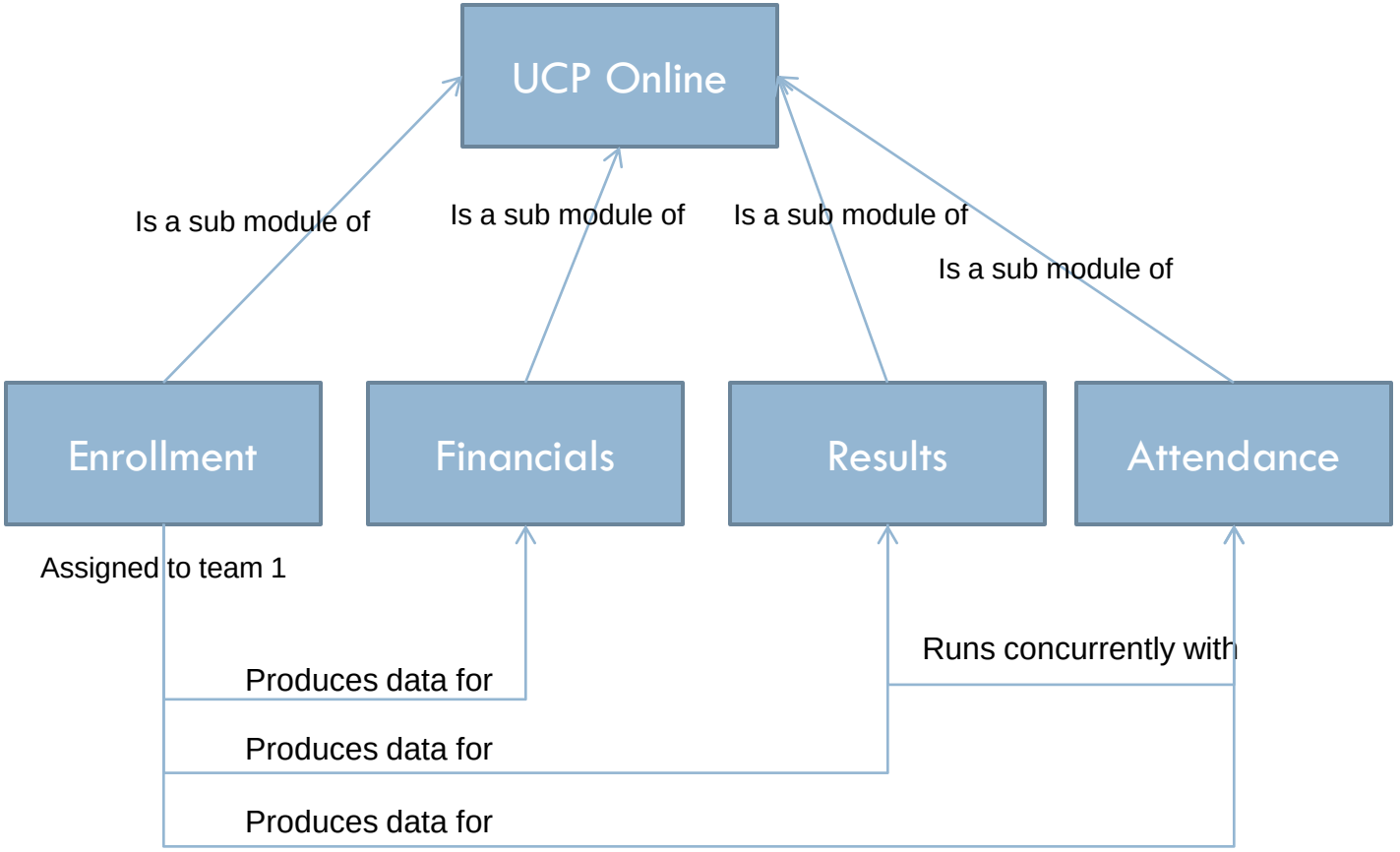
- How software elements(usually modules) are mapped to file structures.
- Relates to configuration control environments.
- Relationship is : ***Stored in***

Structures – Allocation – Work Assignment

- This structure assigns responsibility for developing and integrating modules to appropriate development teams.
- Relationship is: ***Assigned to***


Allocation
Work Assignment
Deployment
Implementation

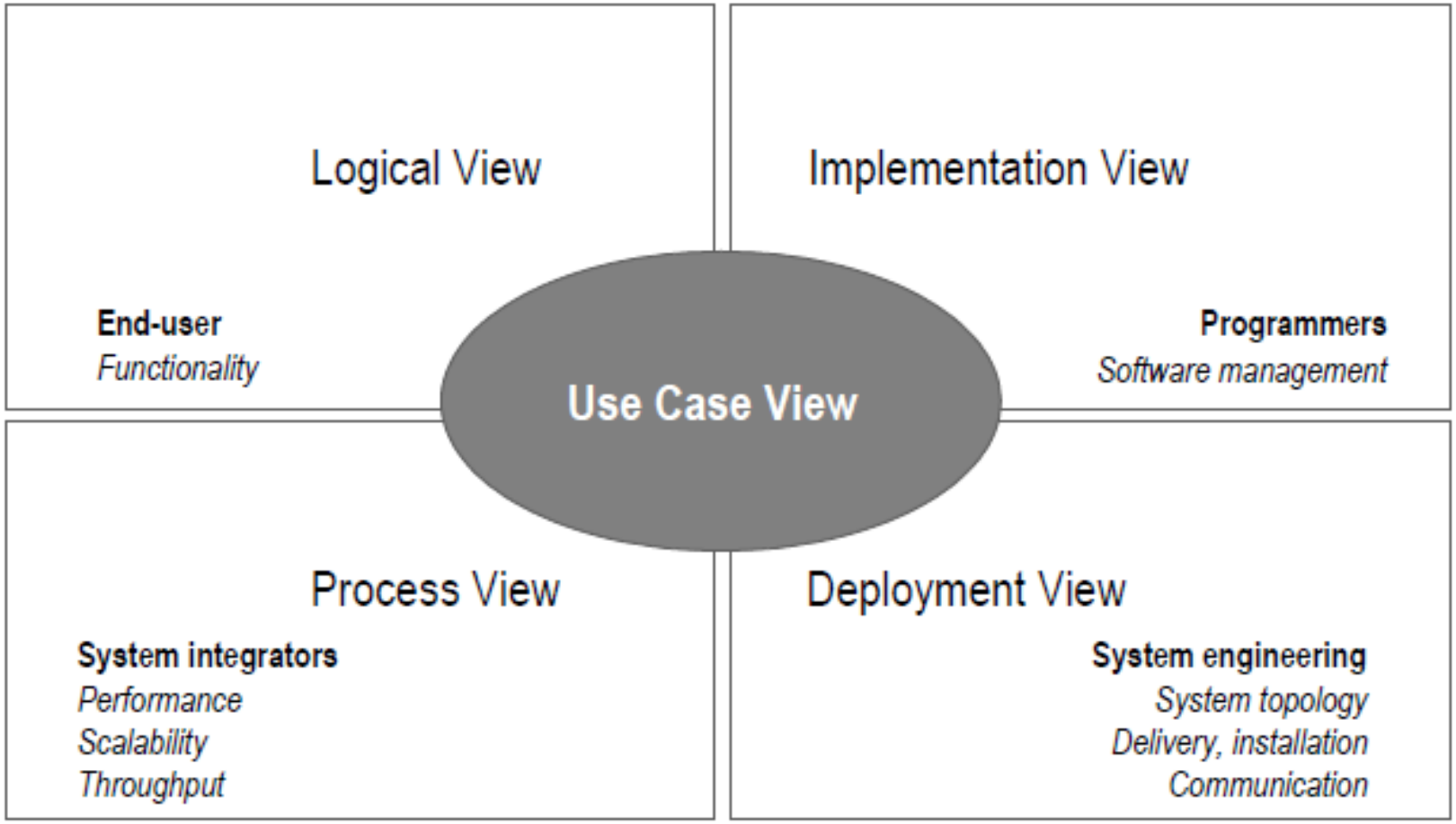
UCP Online

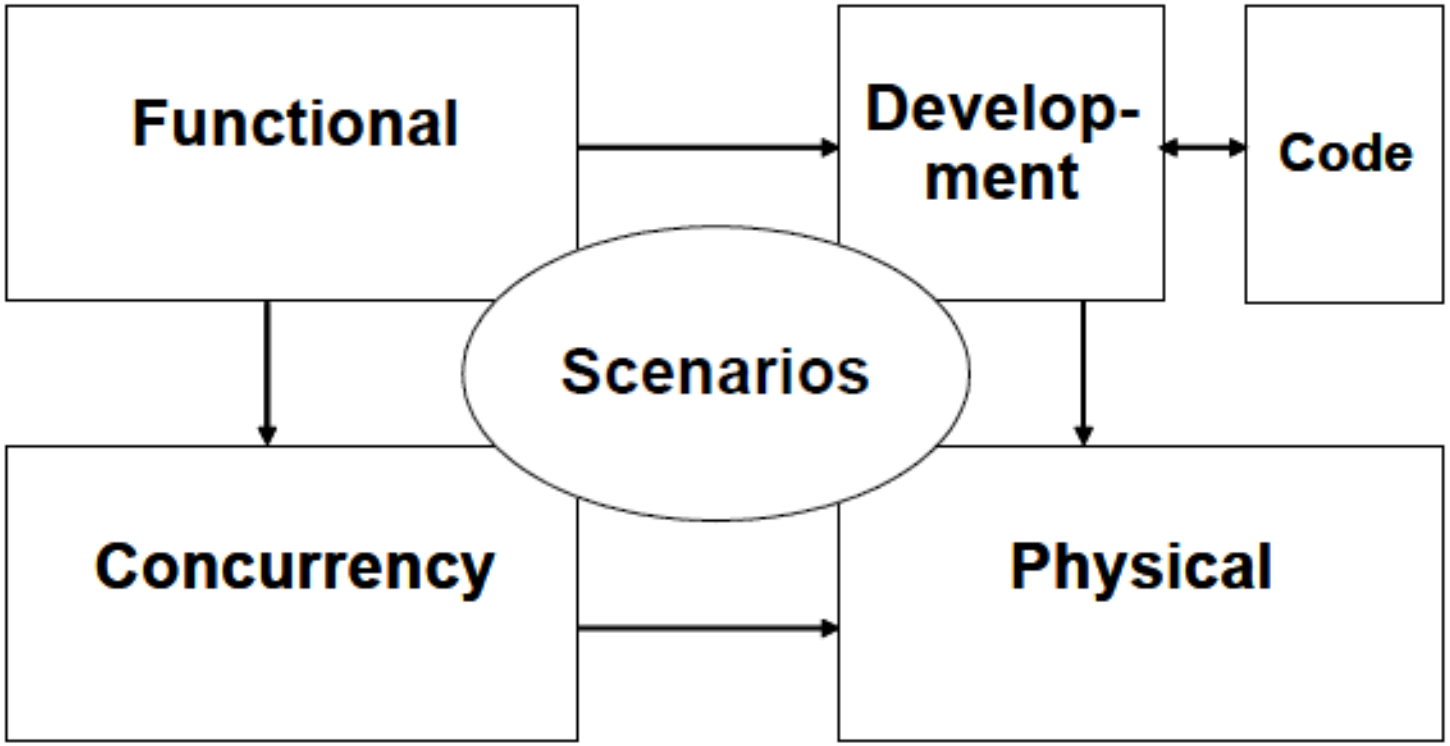


This single diagram will become very complex and difficult to understand if all relations between all modules are shown.

Software Architecture Views

- Views are a subset of the structural elements intended for a certain audience/purpose.


Logical View
End-user
Functionality
Implementation View
Programmers
Software management
Use Case View
Process View
System integrators
Performance
Scalability
Throughput
Deployment View
System engineering
System topology
Delivery, installation
Communication


Functional
Develop-
ment
Code
Scenarios
Concurrency
Physical

View	Components	Users	Rationale
Functional View	functions, key system abstractions, domain elements	domain engineers, product-line designers, end users	functionality, modifiability, product lines/reusability, tool support, work allocation
Code View	classes, objects, procedures, functions, subsystems, layers, modules	programmers, designers, reusers	modifiability/maintainability, portability, subsetability
Development View	files, directories	managers, programmers, configuration managers	managers, programmers, configuration managers
Physical View	CPUs, sensors, storage	hardware engineers, system engineers	system delivery and installation, performance, availability, scalability, security
Concurrency View	processes, threads	performance engineers, integrators, testers	performance, availability

System Quality Attributes

- **Quality Attributes** Scenarios and **Architectural Tactics** are some of the tools available for creating quality architectures.
- There are six major quality attributes
  - **Availability**
  - Modifiability
  - **Performance**
  - **Security**
  - Testability
  - **Usability**

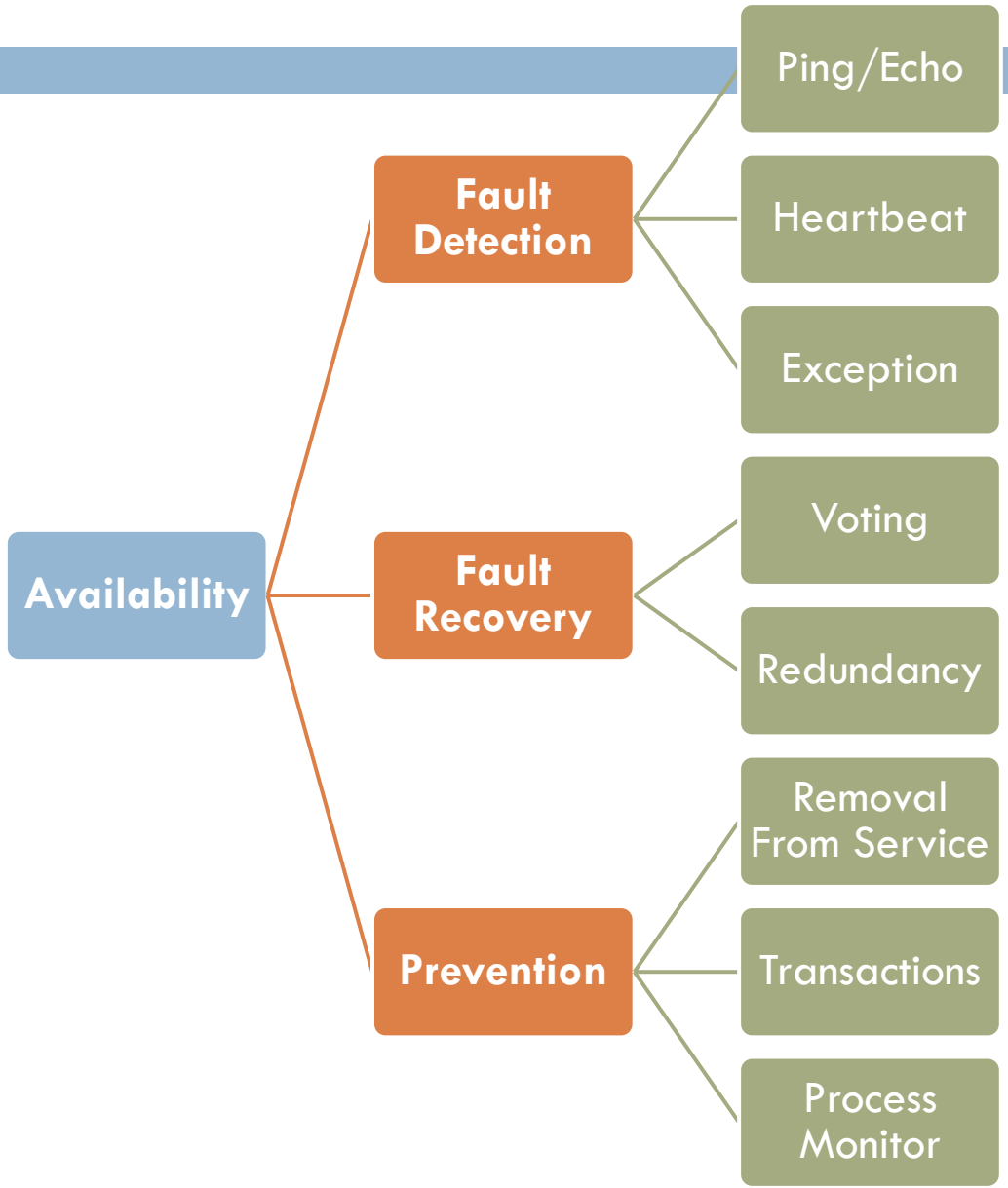
Availability

- Availability is concerned with system failure.
- System failure occurs when the system no longer provides service according to its specification.
- A failure is observable by the users of the system.
- A failure that is masked or hidden is a *fault*.
- Availability = mean time to failure/mean time to failure + mean time to repair.
- The quicker a failure is fixed, the higher the availability.

Availability

Source	Internal to the system, external to the system
Stimulus	Fault, omission, crash, timing, response
Artifact	System's processors, processes, modules, communication channels, persistent storage
Environment	Normal Operation, Degraded Mode
Response	Record it, Notify it, shift to another mode, try automated recovery
Response Measure	Mean time between failures, Time taken for recovery

Availability Tactics



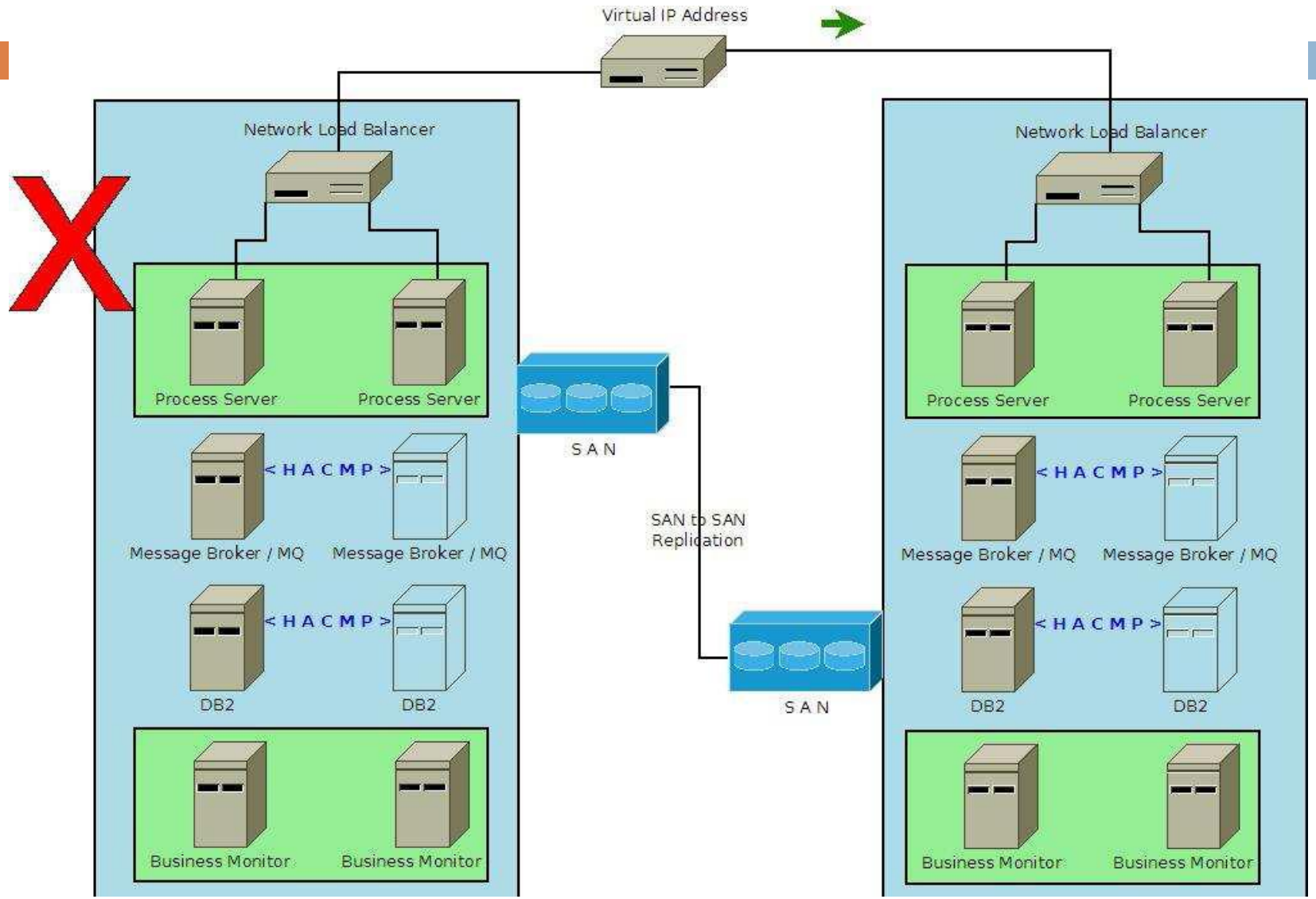
Availability Tactics

- **Fault Detection:**
- The ability of a system to detect failures and faults.
  - **Ping/Echo:** Ping a component and listen to the echo to determine availability.
  - **Heartbeat:** The component notifies some other component that its alive and available.
  - **Exception:** The programming platform/architecture provides a exception throwing/handling mechanism that can be used to detect faults.

Availability Tactics

- **Fault Recovery:**
- The ability of a system to self recover from a fault if it occurs.
  - **Voting:** Redundant processes having the same algorithm. Use voting algorithm to determine correct response.
  - **Redundancy:** Use redundant processes, modules, communication channels.
    - Example, high availability architectures

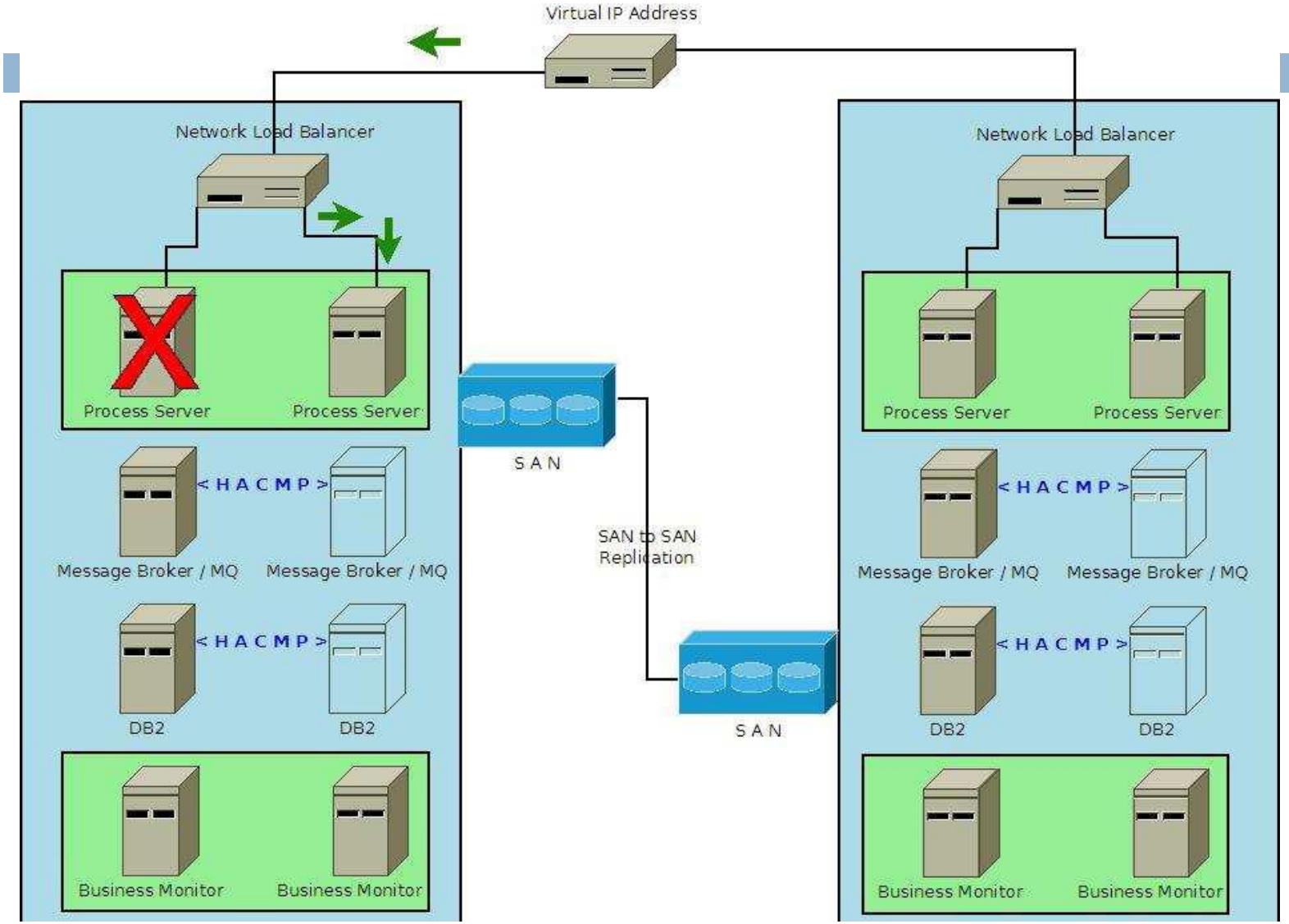
HA Architecture – Failure Example 1



Production Site

Disaster Site

HA Architecture – Failure Example 2



Production Site

Disaster Site

Availability Tactics

- **Fault Prevention:**
- The ability of the system to perform preventive measures before a fault occurs.
  - **Removal from service:** Remove a faulty service, module or component from the system.
    - Example, rebooting system to prevent memory leaks from crashing it.
  - **Transactions:** A transaction is a bundling of steps that can be undone (rolled back) if any step fails.
  - **Process Monitor:** A component responsible for using ping or heart beat to monitor components and remove failed components and use redundant components.

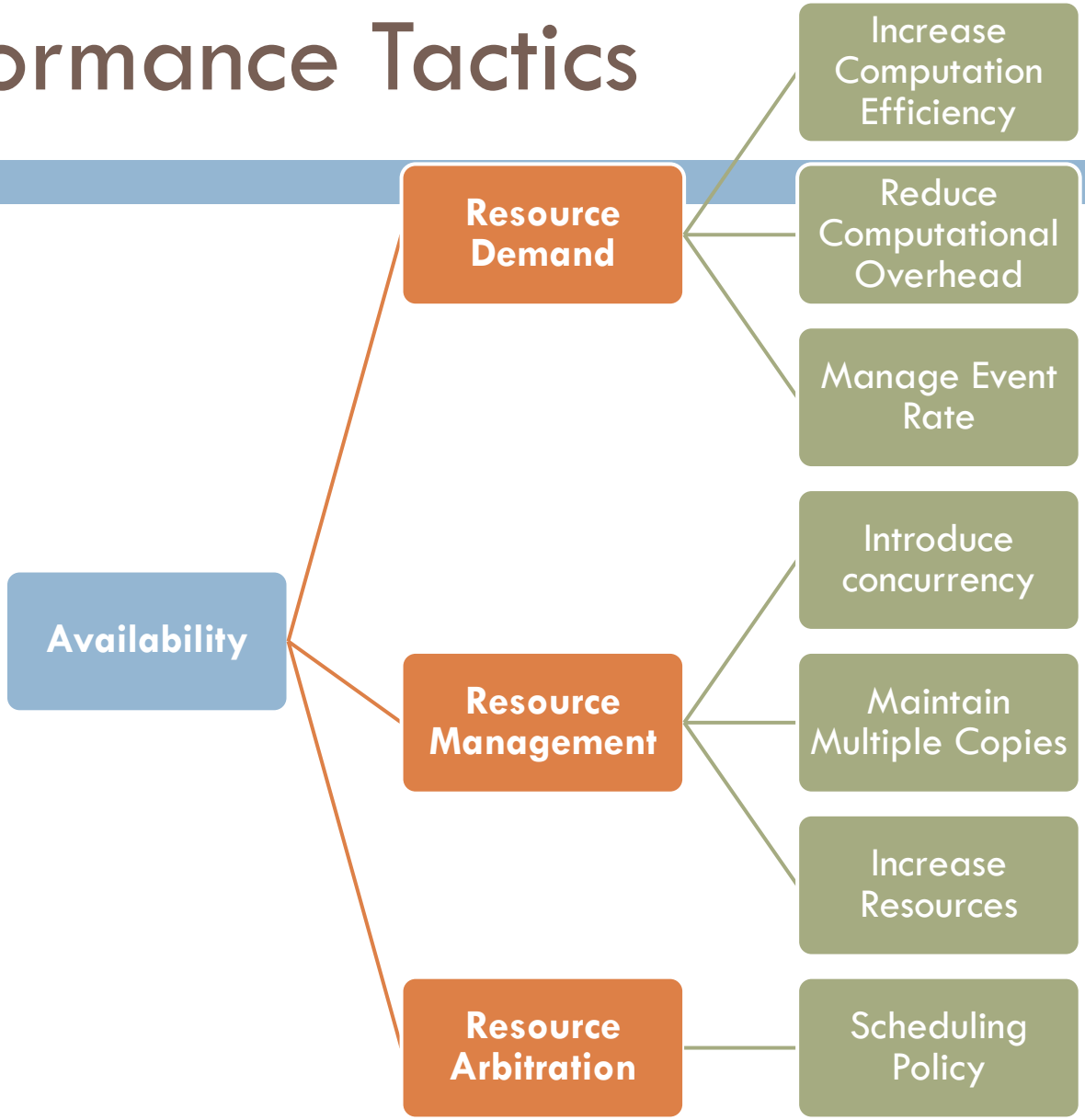
Performance

- Performance is about timing.
- Events (interrupts, messages, requests from users, passage of time) occur and the system must respond to them.
- Performance is the measure of how long it takes the system to response with a correct result when an event occurs.

Performance

Source	Internal to the system, external to the system
Stimulus	Periodic, sporadic or stochastic events arrive.
Artifact	System
Environment	Normal Mode, Overload Mode
Response	Processes stimuli, changes mode of services, records, notifies.
Response Measure	Latency, deadline, throughput, jitter, miss rate, data loss.

Performance Tactics



Performance Tactics

- **Resource Demand:**
- One way of increasing performance is to *reduce* the resources required for processing an event stream.
  - **Increase computational efficiency**
    - Increasing computational efficiency means using better **algorithms** and **data structures** so that the same computation takes less time.
  - **Reduce computational overhead**
    - Remove overheads like *remote procedure calls, intermediaries etc.*
    - *Tradeoff between performance and modifiability.*

Performance Tactics

- **Manage Event Rate**
  - If its possible to reduce event sampling rate.

- **Resource Management**
  - **Introduce Concurrency(of computation)**
    - If requests can be processed in parallel, they should.
  - **Maintain Multiple Copies(of data)**
    - Caching
  - **Increase Resources**
    - More, better hardware.

- **Resource Arbitration**
  - Deals with the scheduling of resources.
    - LIFO
    - FIFO
    - Priority Queue
    - SJF
    - LRU
    - RR

Security

- Security is the measure of the system's ability to resist unauthorized usage while still providing its services to authorized users.
- Goal of unauthorized usage could be to
  - Access data
  - Access services
  - Modify data
  - Deny access to legitimate users

Security

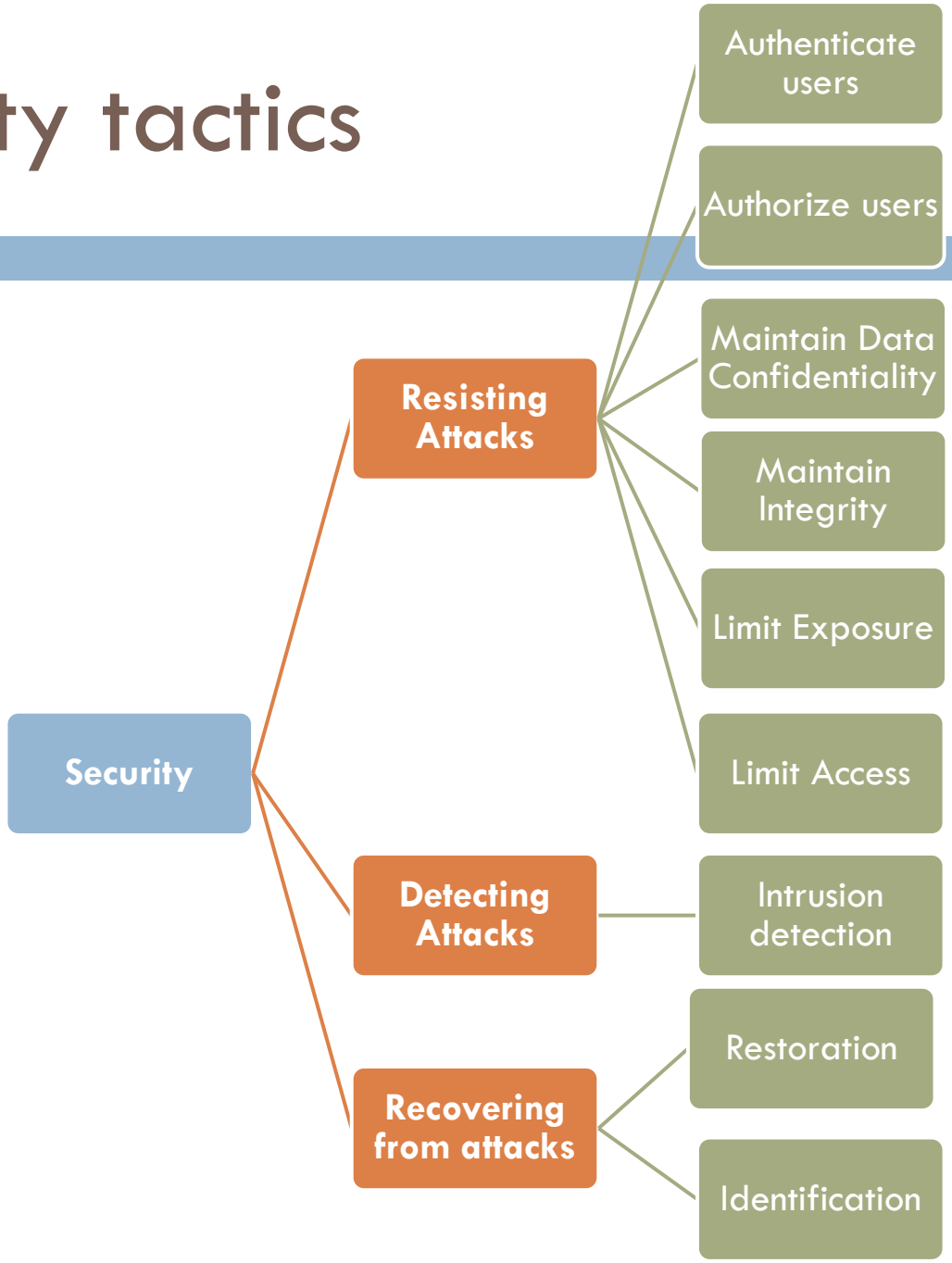
- Security can be characterized as the system providing
  - **Non repudiation:** Any action performed on the system cannot be denied by those who did.
  - **Confidentiality:** Means that services or data are protected from unauthorized access.
  - **Integrity:** Data or services are delivered as intended. E.g. grades have not been changed since the instructor assigned them.
  - **Assurance:** Parties to a transaction(access to data/services) are who they say they are.

- **Availability:** Means that the system will be available to authorized users.
- **Auditing:** To track transactions and activities to a certain level of detail.

Security

Source	Individual or system who is correctly identified, incorrectly identified, internal, external, with access to limited resources, vast resources
Stimulus	Tries to view data, change/delete data, access services, reduce availability of services to other users.
Artifact	System services, data within the system
Environment	Online or offline, connected or disconnected, firewalled or open
Response	Authenticates user, blocks access to data, allows access to data, records access or attempts to access data/services by identity, stores data in unreadable format, recognizes an unexpected demand of services.
Response Measure	Probability of detecting attacks, probability of recovering from attacks etc.

Security tactics



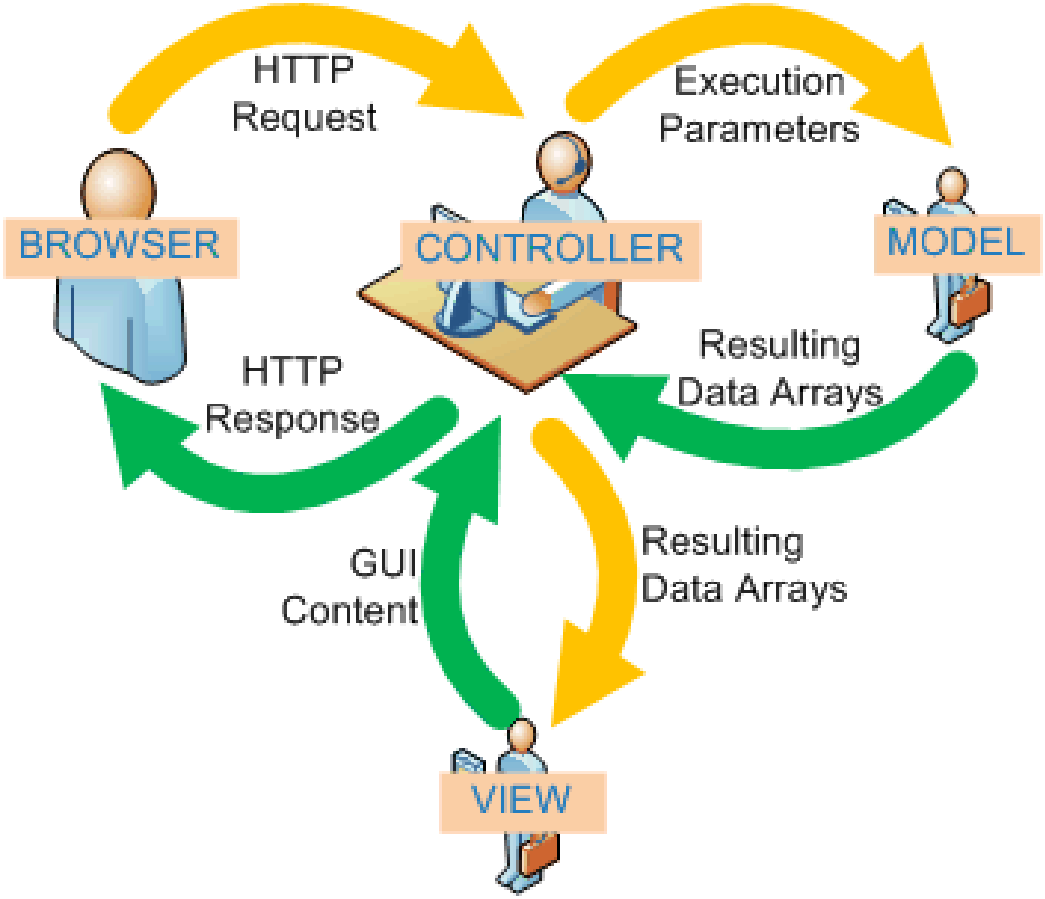
Architectural Styles

Architectural Styles

- In order to design software architectures, software architects use a “architectural styles.
- An architect can use more than one of these styles to model software systems.

- Model View Controller (MVC)
- Pipes and Filters
- Abstract Data Types
- Event Driven Architectures
- Layered Systems
- Blackboard/Repositories
- Peer-to-Peer
- Client Server
- Monolithic

Model View Controller



- The MVC is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller.
- **Model** - The model represents data and the business rules that determine access to and updates of this data. Often the model serves as a software approximation to a real-world process, so simple real-world modeling techniques apply when defining the model.

- **View** -The view displays the contents of a model. It accesses data through the model and specifies how that data should be presented.
- **Controller** - translates interactions with the view into actions to be performed by the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

- **Consequences**
- **Re-use of Model components.** The separation of model and view allows multiple views to use the same model.
- **Easier support for new types of clients.** To support a new type of client, you simply write a view and some controller logic and wire them into the existing enterprise application.
- **Increased design complexity.** This pattern introduces some extra classes due to the separation of model, view, and controller.

Technical Architect

- **The technical architect identifies the tools and technologies that will be used in the project.**
  - Some architectural decisions are made at the project level, for example, choosing Java for all server-side components, where as some decisions are made at the project level, such as which XML parser to use.
- Examples: Operating Systems, hardware, database engines, third party components, web servers, application servers etc.

Technical Architect

- **Recommends (to the project manager)development methodologies and frameworks of the project.**
  - For example: One recommendation could be to use a specific process model, like iterative model.
  - Another recommendation would be to use “use case models” to specify requirements.
  - The decision of when to make prototypes.
- Some of these decisions are standardized at the enterprise level.

Technical Architect

- **The technical architect provides the overall design and structure of the application.**
  - The technical architect is responsible for providing a consistent design or framework which less experienced developers follow.

Technical Architect

- **The technical architect ensures that the project is adequately defined.**
  - Ensures that the requirements, their specifications and the corresponding design are well defined, consistent, complete, correct and unambiguous.
  - Requires the help of the project manager and business analyst.

Technical Architect

- **Ensures that the project design is well documented.**
  - The goal is for the developers to read application design and code without active participation of the architect at all times.

Technical Architect

- **Establishes coding guidelines.**
  - Exception Handling
  - Logging
  - Testing
  - Threading
- Important because every developer has his/her own preferences that reduces consistency.

Technical Architect

- **The technical architect is a mentor for developers**
  - Technical architect is a leader and a mentor than a developer.

Technical Architect

- **The technical architect ensure compliance to rules and standards.**
  - Through code reviews.

Technical Architect

- **The technical architect assists project manager in time, resource and cost estimations.**
  - The manager may not be well versed in technology

Technical Architect

- **Assesses technical competence and recommends positions of developers.**
  - Mistakes in this area can be harmful.