Achieving Qualities - Tactics

Part -1

Reference

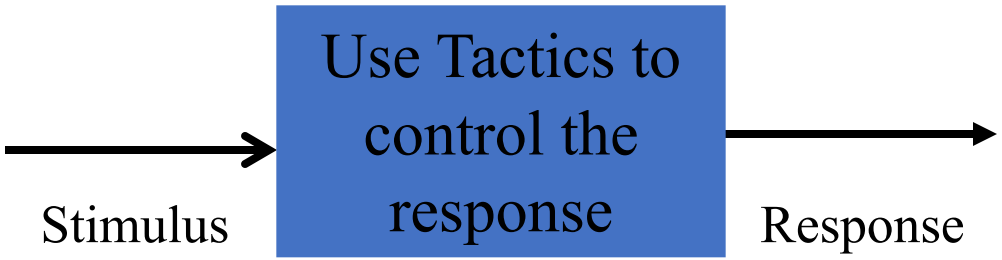
Software Architecture in Practice

Bass, Clements, Kazman

3rd Edition

What are tactics?

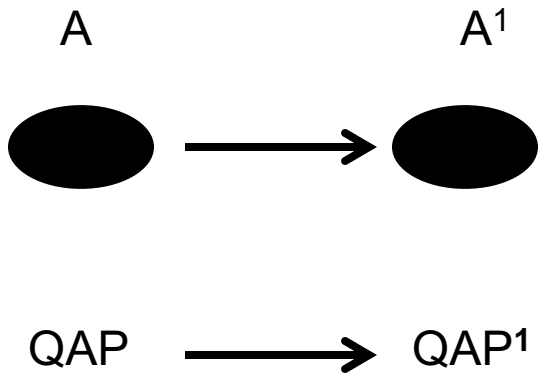
- Design is a collection of decisions
  - Some achieve functionality
  - Other control quality
- Tactics are design options for the architect to achieve quality



What is a tactic?

- An action carefully planned to achieve a specific end
- "An architectural tactic is a means of satisfying a quality attribute response measure ..through architectural design decisions"
- Patterns package tactics!

Architectural Tactics



An architectural tactic moves from one architecture to another where a parameter of the quality attribute model moves in a known direction.

Informal Characterization of Tactics

- “An architectural pattern is a collection of components and interactions to resolve multiple conflicting forces” – Buschmann [1996]

- An architectural tactic is a transformation that will move in the direction of resolving a single quality attribute force.

Architectural Tactics

- Quality requirements drive design
- Quality attribute parameters differentiate one design from another
- How does one achieve quality
- Tactics!

Quality Attributes

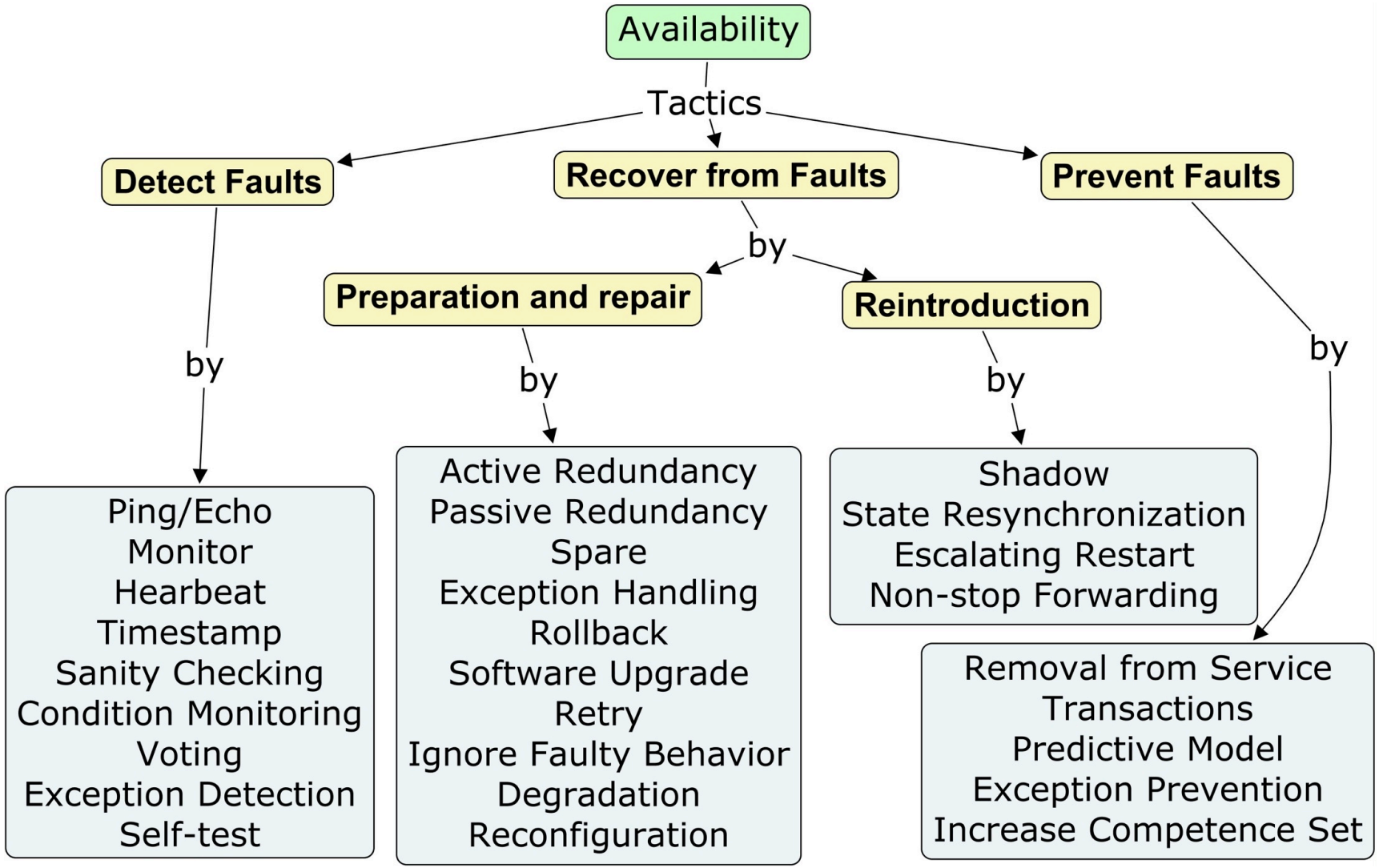
- Availability
- Modifiability
- Performance
- Security
- Testability
- Usability

Availability

- Should be there when needed
- It is about reliability and recovery
- “Availability refers to the ability of a system to mask or repair faults such that the cumulative service outage period does not exceed a required values over a specified period of time”

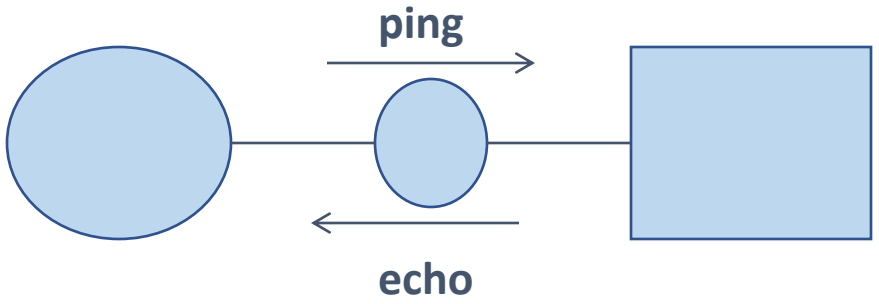
  …Bass, Clements, Kazman

- Linked to
  - Security: DOS attacks
  - Performance: if the response is very slow..


Availability
Tactics
Detect Faults
Recover from Faults
Prevent Faults
by
Preparation and repair
Reintroduction
by
by
by
by
Ping/Echo
Monitor
Hearbeat
Timestamp
Sanity Checking
Condition Monitoring
Voting
Exception Detection
Self-test
Active Redundancy
Passive Redundancy
Spare
Exception Handling
Rollback
Software Upgrade
Retry
Ignore Faulty Behavior
Degradation
Reconfiguration
Shadow
State Resynchronization
Escalating Restart
Non-stop Forwarding
Removal from Service
Transactions
Predictive Model
Exception Prevention
Increase Competence Set

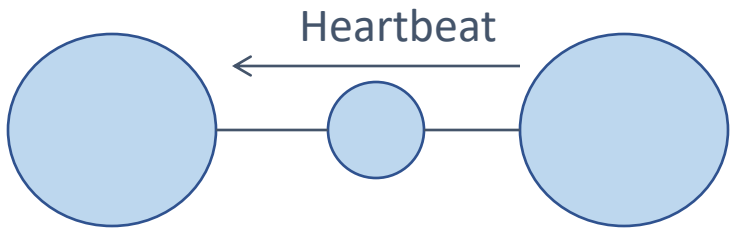
Fault Detection (availability..)

Ping/Echo(availability..fault detection..)



- One component issues a ping and expects to receive back an echo, within a predefined time , from the computer under scrutiny

Heartbeats
(availability..fault detection..)

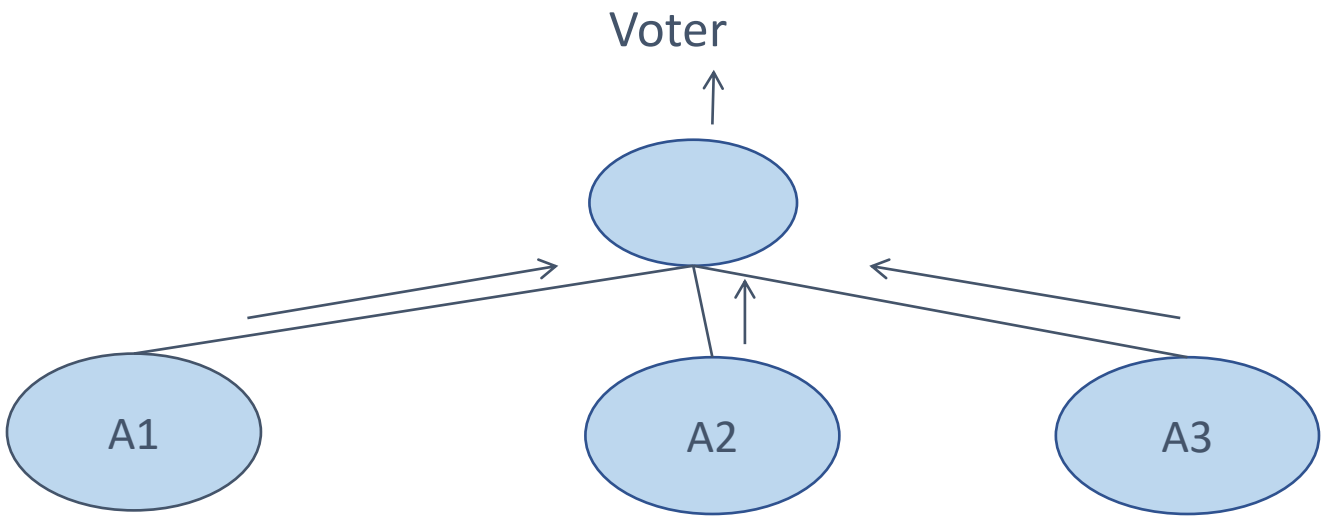


- In this case one component emits a heartbeat message periodically and another component listens for it

Exception Detection
(availability..fault detection..)

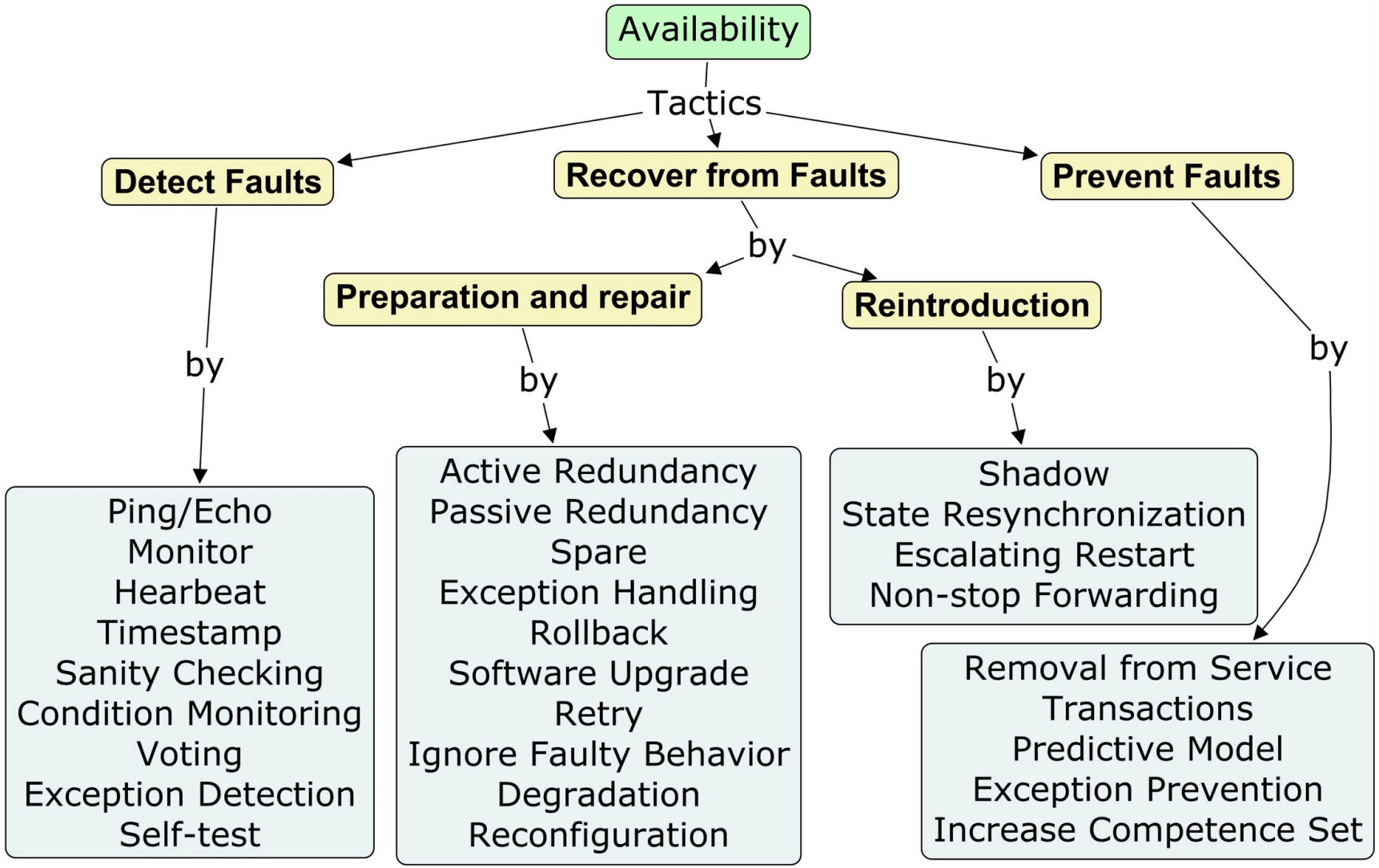
When a fault occurs an exception is raised by the faulting process itself.

Voting(availability..fault detection..)



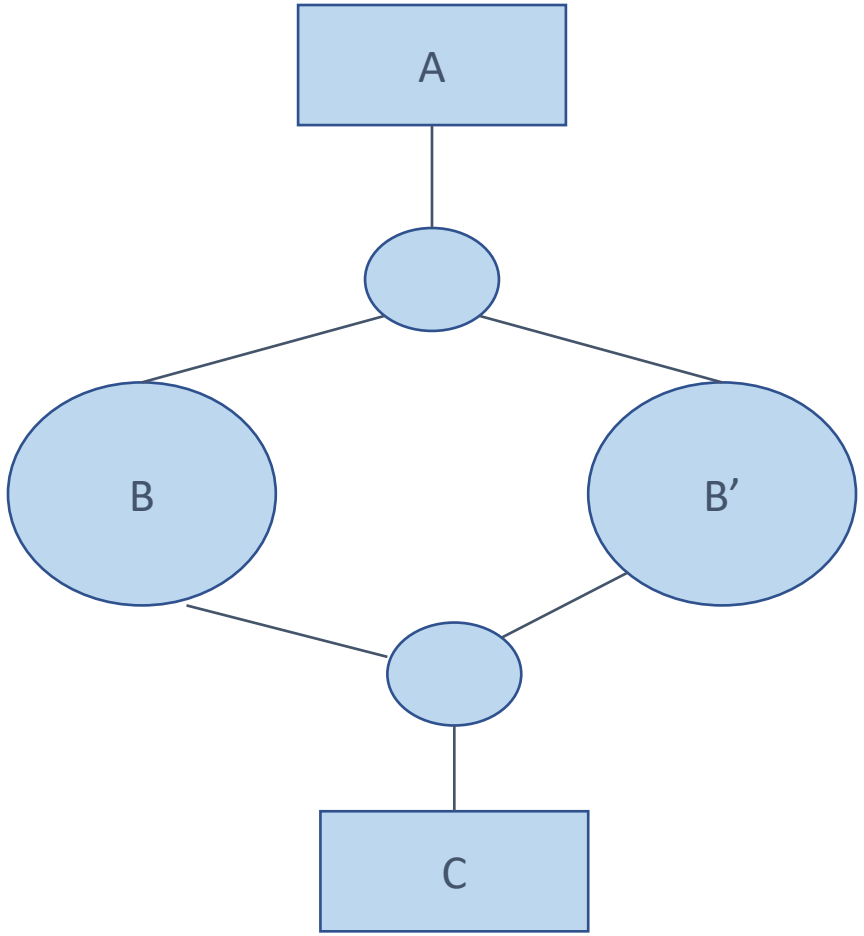
- Execute the same functionality in more than one component and take majority voting
- Popular in hardware

Recovering from Faults


Availability
Tactics
Detect Faults
Recover from Faults
Prevent Faults
by
Preparation and repair
Reintroduction
by
by
by
by
Ping/Echo
Monitor
Hearbeat
Timestamp
Sanity Checking
Condition Monitoring
Voting
Exception Detection
Self-test
Active Redundancy
Passive Redundancy
Spare
Exception Handling
Rollback
Software Upgrade
Retry
Ignore Faulty Behavior
Degradation
Reconfiguration
Shadow
State Resynchronization
Escalating Restart
Non-stop Forwarding
Removal from Service
Transactions
Predictive Model
Exception Prevention
Increase Competence Set

Active Redundancy
(availability..recover..prep & repair..)

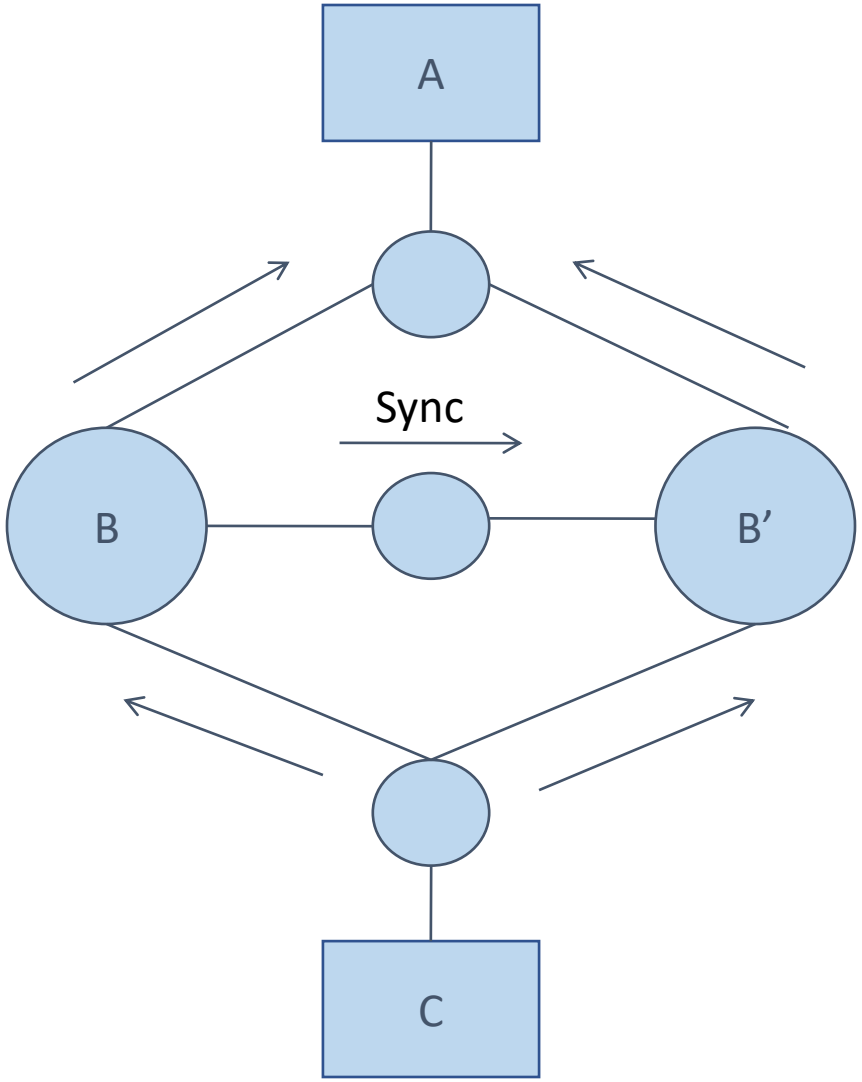
- Run the same functionality on several components.
- All components respond to events in parallel



Passive Redundancy

(availability..recover from faults..prep and repair..)

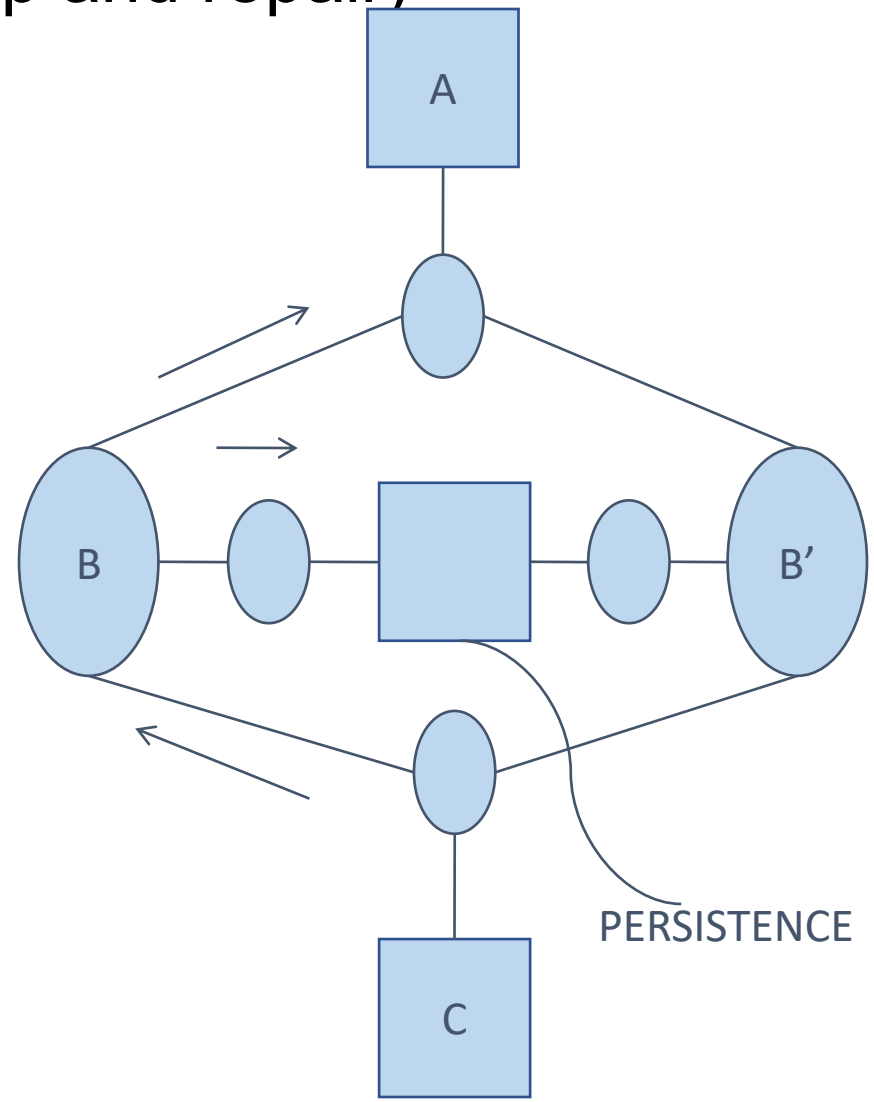
- One component (the primary) responds to events and informs the other components (the secondary) of state updates it must make periodically



Spare

(availability..recover from faults..prep and repair)

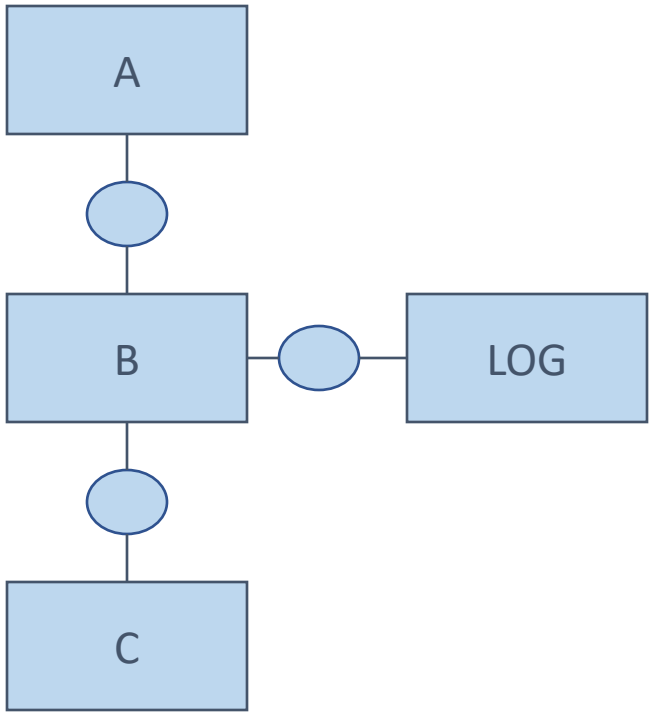
- standby component configured to replace the failed component
- on a fault, a persistence component ensures that the spare is sufficiently fresh before resuming services



Roll Back

(availlability..recover from faults..prep & repair..)

- First create a checkpoint (copy of a consistent state created periodically or in a response to some event)
- When the System fails, state gets inconsistent
- Restore the system using a previous check point of a consistent state and a log of the transactions that since then



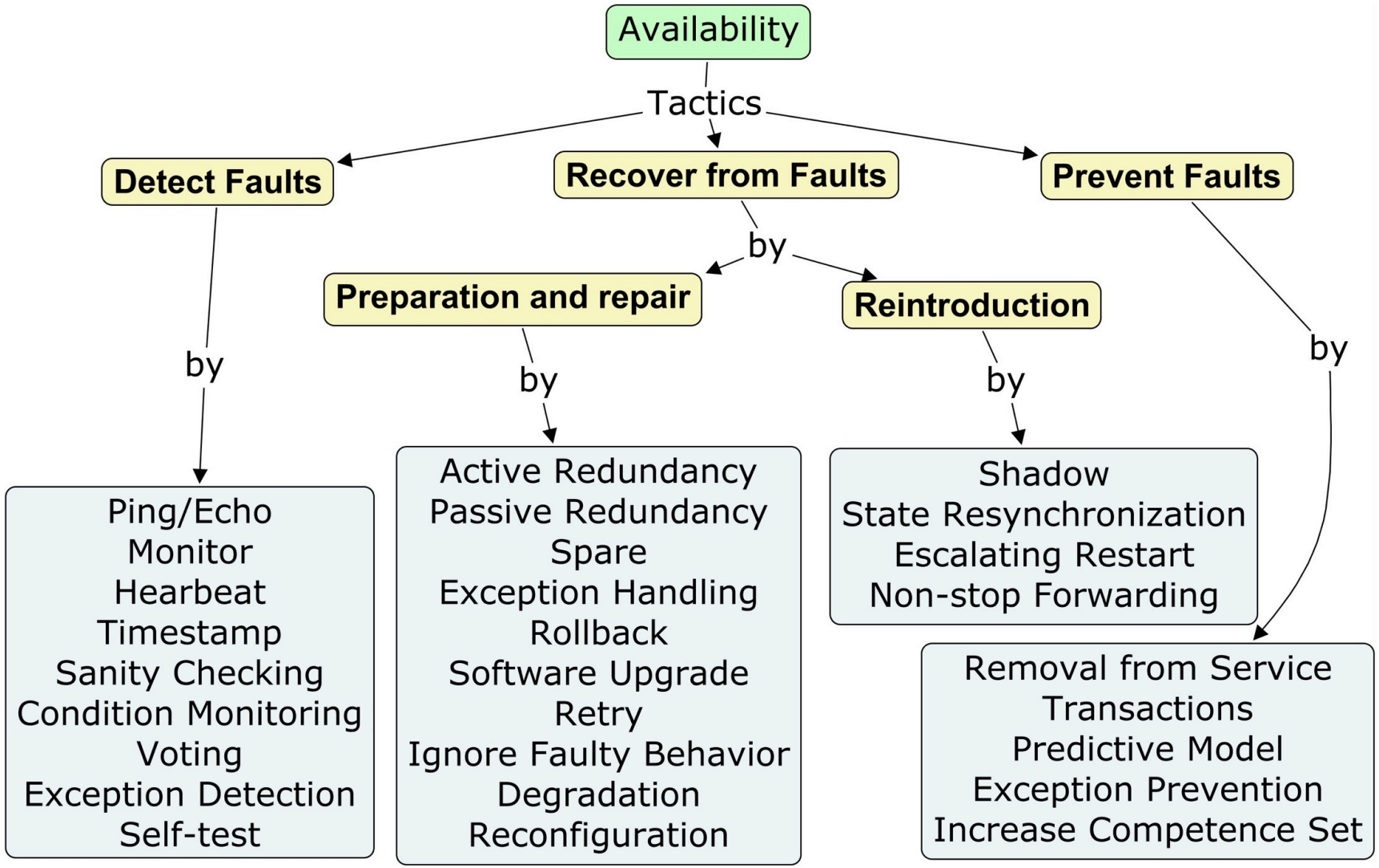
Shadow Operation
(availability..recover from faults..reintroduce..)

- Run a previously failed component in “shadow mode” for a while to make sure it is working properly before resuming its services

State Resynchronization
(availability..recover from faults..reintroduce..)

- Part of Passive redundancy when the components need to be synchronized. How often should they be synchronized and in what size are the issues dealt with in this tactic

Fault Prevention (availability..)

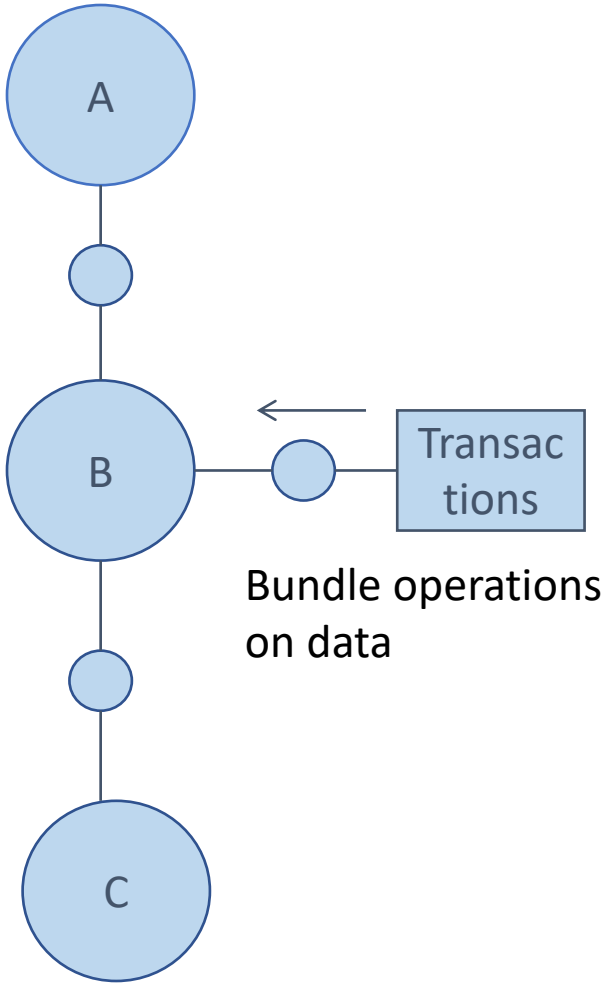

Availability
Tactics
Detect Faults
Recover from Faults
Prevent Faults
by
Preparation and repair
Reintroduction
by
by
by
by
Ping/Echo
Monitor
Hearbeat
Timestamp
Sanity Checking
Condition Monitoring
Voting
Exception Detection
Self-test
Active Redundancy
Passive Redundancy
Spare
Exception Handling
Rollback
Software Upgrade
Retry
Ignore Faulty Behavior
Degradation
Reconfiguration
Shadow
State Resynchronization
Escalating Restart
Non-stop Forwarding
Removal from Service
Transactions
Predictive Model
Exception Prevention
Increase Competence Set

Removal from service
(availability..prevent faults..)

- A component is temporarily removed from service and undergoes some activities to prevent anticipated failures
- E.g.: rebooting a component to prevent memory leaks

Transactions
(availability..prevent faults..)

- Bundle several sequential step such that the bundle can be undone together
- Prevents any data to be affected if a component fails
- All or none



Home Work

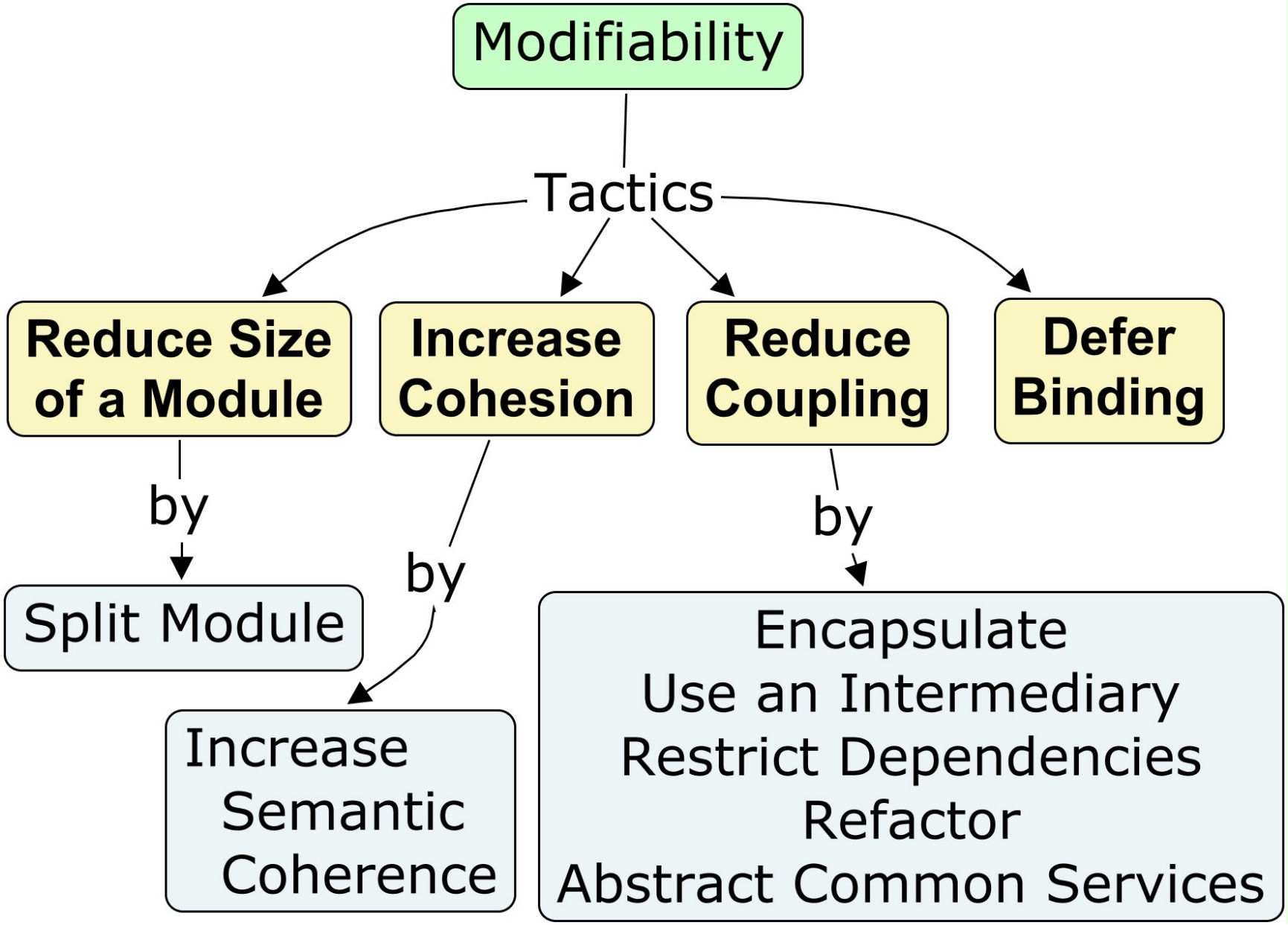
- Think of tactics for Availability

Tactics – Part 2

Modifiability

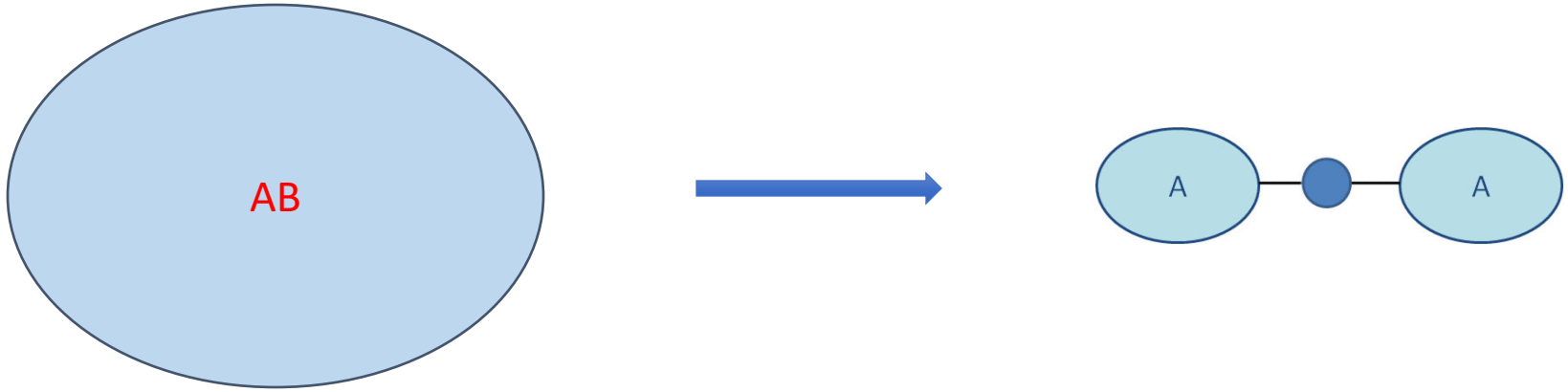
Modifiability

- A large percentage of software cost occurs after release
- Ability to make changes
  - To fix defects
  - Add new features
  - Retire old features
  - New technologies, paltforms
- Cost and risk in making these changes


Modifiability
Tactics
Reduce Size of a Module
Increase Cohesion
Reduce Coupling
Defer Binding
by
by
by
Split Module
Increase Semantic Coherence
Encapsulate
Use an Intermediary
Restrict Dependencies
Refactor
Abstract Common Services

Split Module (modifiability..reduce size..)

- If the module is very big, break it into smaller modules

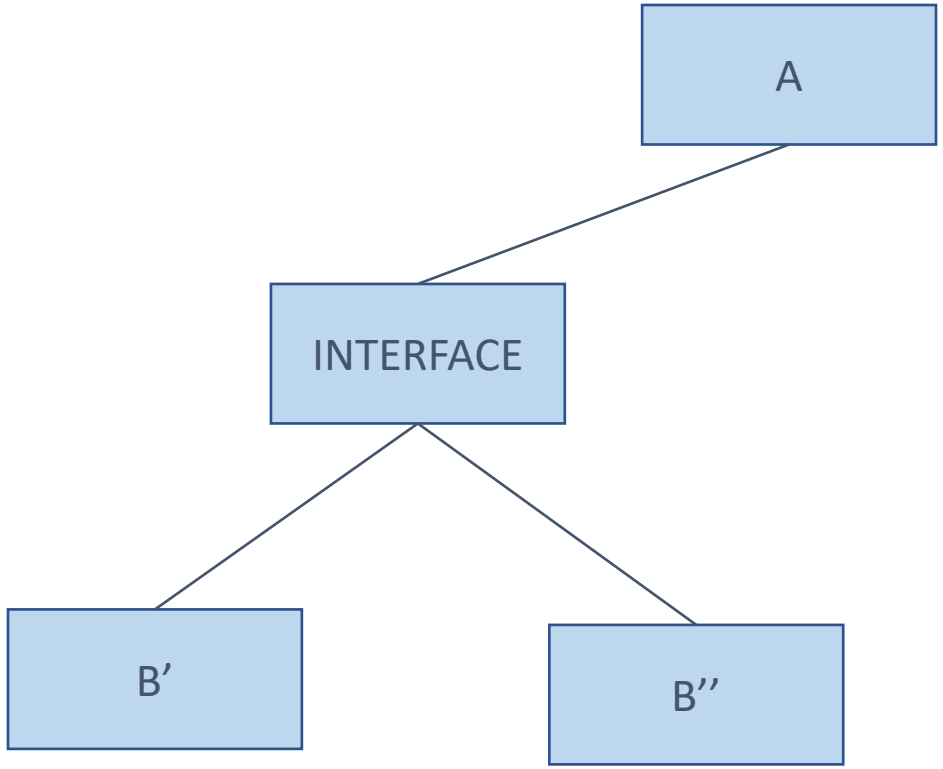


Increase Semantic Coherence
(modifiability..increase coherence..)

- Move some responsibilities from one module to another
- What responsibilities should stay together can be based on the semantics – similar functions stay together

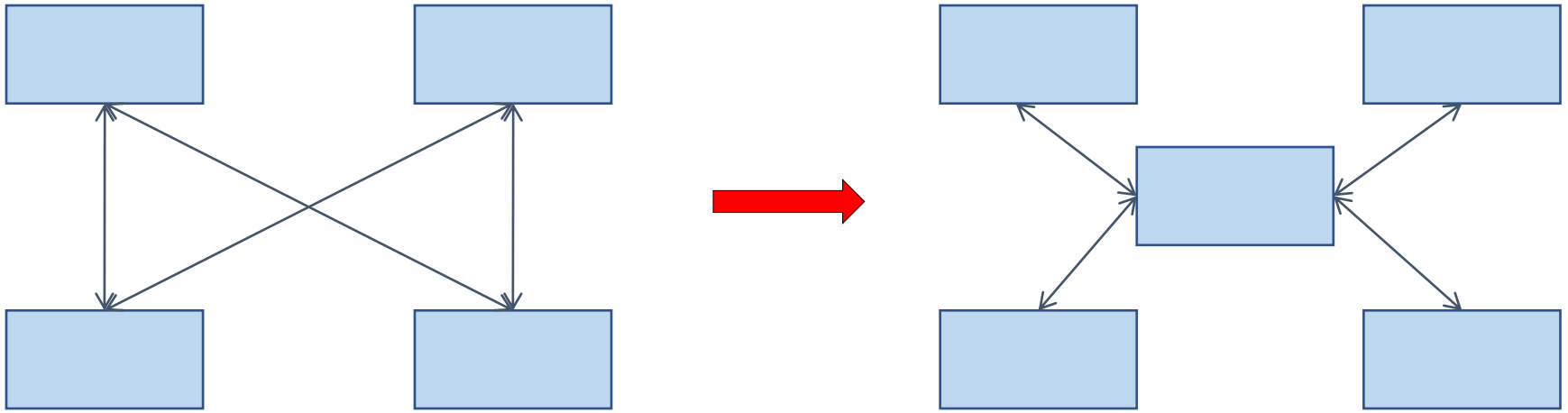
Encapsulate
(modifiability..reduce coupling..)

- If module A uses services of module B, make it available through an interface I
- This will allow A to be unchanged when module B is - but I needs to be maintained



Different implementations of A

Use an intermediary
(modifiability..reduce coupling..)



- If the responsibility of module A requires B to be executed, an intermediary can break this dependency
- Publish-subscribe: publisher need not know who is subscribing, the intermediary handles this

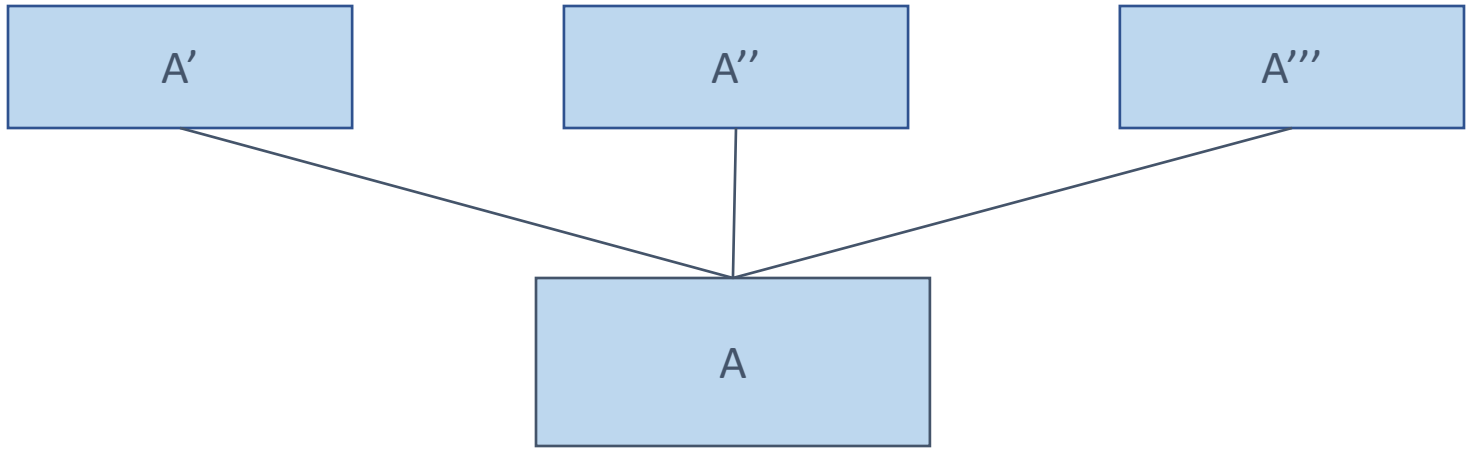
Restrict Dependencies
(modifiability..reduce coupling..)

- Reduce the data production/consumption between modules - sharing data causes ripple effects
- Seen in layered architectures

Refactor (modifiability..reduce coupling..)

- If the same service is being provide by two modules in two different situations, refactor the code and make them one module

Abstract common services (modifiability..reduce coupling..)



- If the services are quite the same but similar, implement a more generalized service

Defer Binding Time
(modifiability..)

- Reduce time to deploy changes
- Non developers to make changes.
- System administrator to make settings or provide input that affects the behavior.
- Examples:
  - Compile time: parameters
  - Boot time: configuration files
  - Runtime: dynamic lookup

Home Work

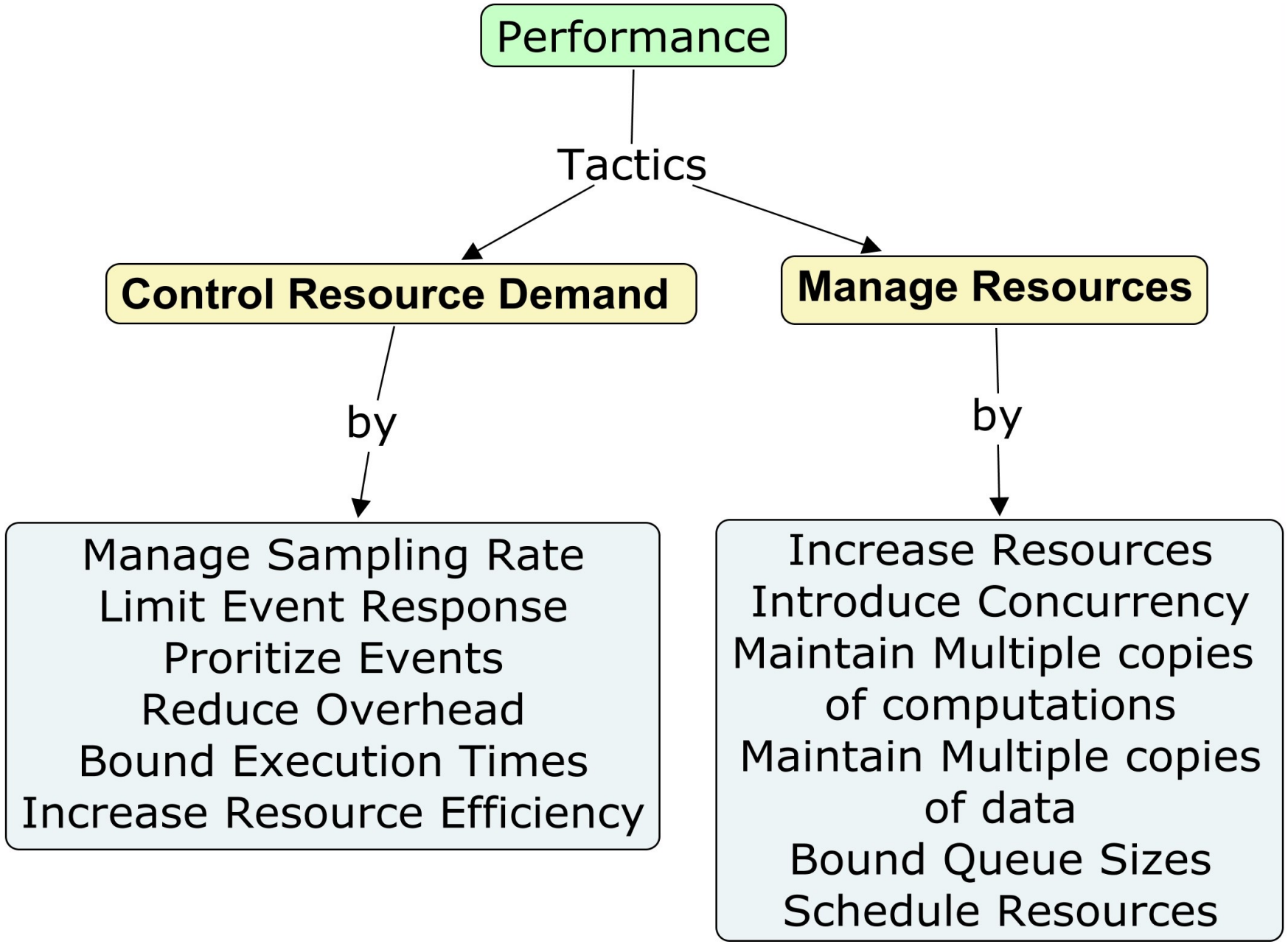
- Mull over Modifiability tactics – look for examples and some new tactics? ☺

Tactics – Part 3

Performance

Performance

- it is about response to an event
- Measured in time
- Connected to scalability
  - Increasing capacity without sacrificing time response

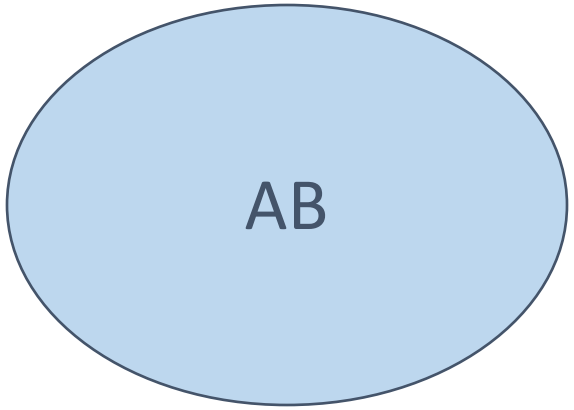

Performance
Tactics
Control Resource Demand
Manage Resources
by
by
Manage Sampling Rate
Limit Event Response
Proritize Events
Reduce Overhead
Bound Execution Times
Increase Resource Efficiency
Increase Resources
Introduce Concurrency
Maintain Multiple copies
of computations
Maintain Multiple copies
of data
Bound Queue Sizes
Schedule Resources

Control Resource Demand
(performance..)

Increase Resource efficiency (performance.. control demand..)

- Improving the efficiency of algorithms used in critical areas will increase performance of the system

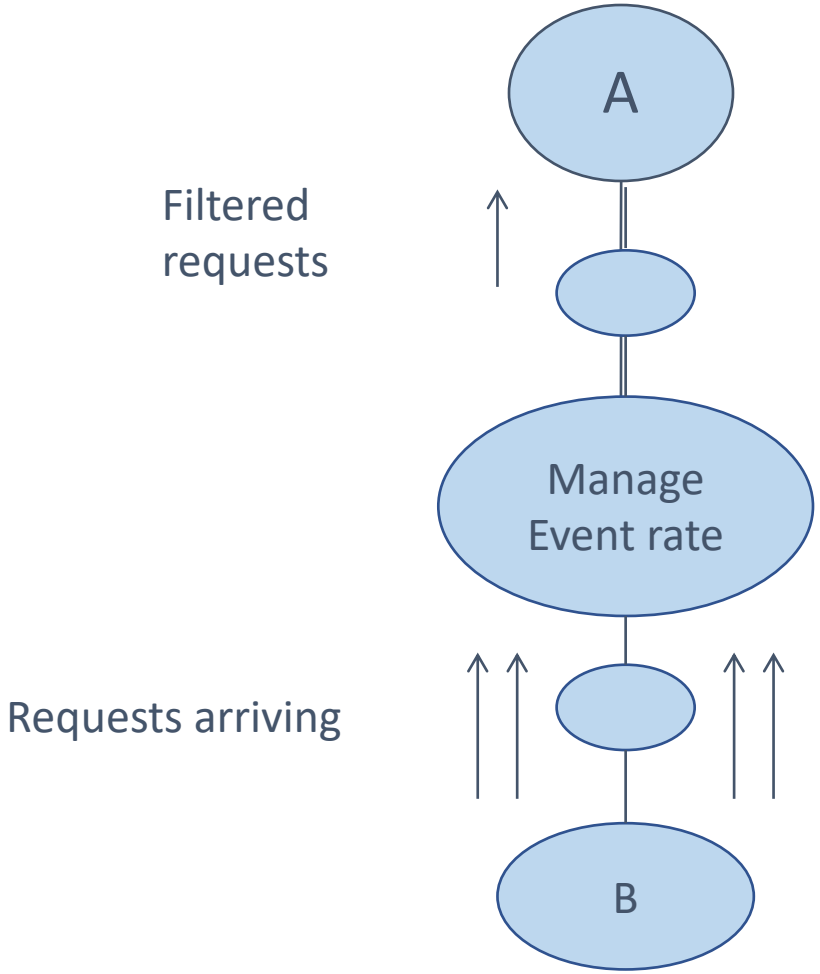
Reduce overhead (performance..control demand..)



- Using more components for the same job increases latency as they need to share data - communication overheads

Limit Event Rate (performance..control demand..)

- Don’t process request beyond a certain number – drop them!
- A separate component or a policy can manage the arrival of events



Manage Sampling Rate
(performance..control demand..)

- Don’t process all members of the queue!
- Signal processing – sample at a lower frequency

Bound Execution Time (performance..control demand..)

- Place a limit on how much execution time is needed to respond to an event
- ‘good enough’

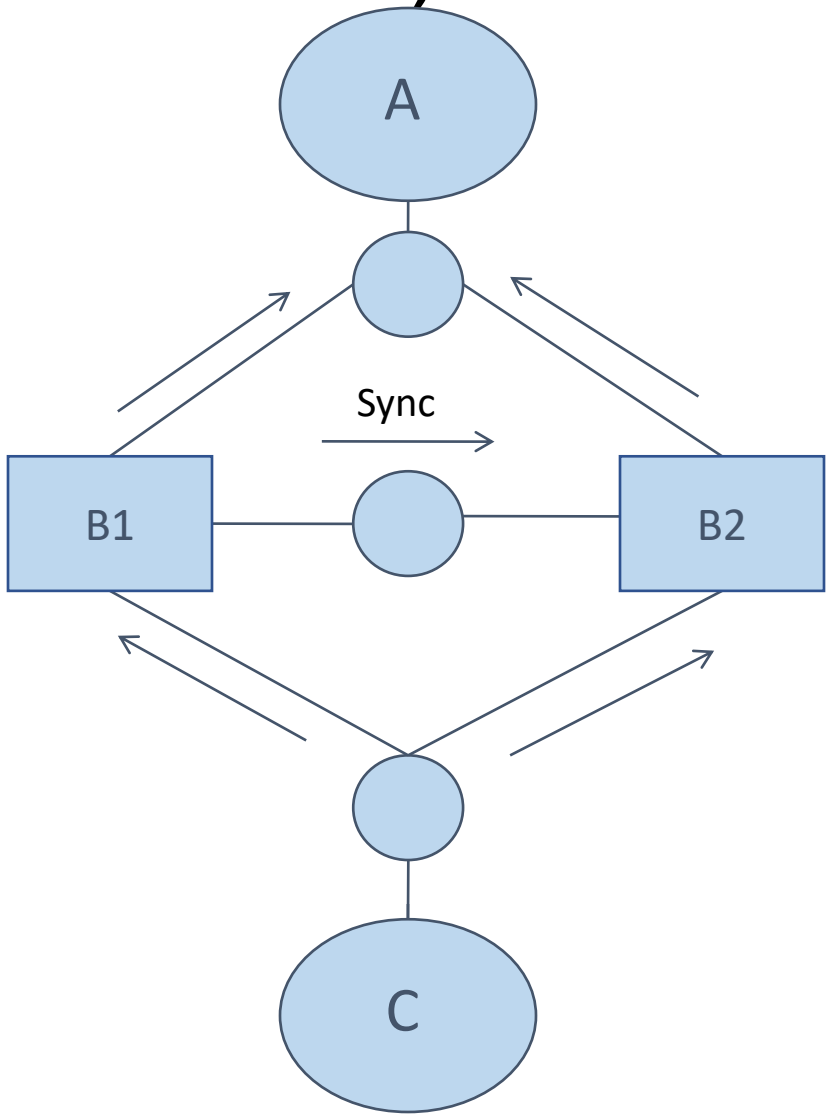
Manage Resources (performance)..

Increase Resources (performance..manage resources)

- Faster processors
- More processors
- Faster networks
- Additional memory ..
- All have the potential to reduce latency

Concurrency (performance..manage resources)

- Create new threads/processes for different activities



Maintain Multiple Copies of Data (performance..manage resources)

- Caching
- Keep multiple copies of data at different levels to reduce contention on the central server

Maintain Multiple copies of Computation (performance..manage resources)

- Load Balancer!!

Schedule Resources (performance..manage resources)

- Scheduling is all about assigning priorities
- Whenever there is a contention(of resources) one can schedule
- FIFO – same priority for all
- Fixed Priority – can decide what is important

Home Work

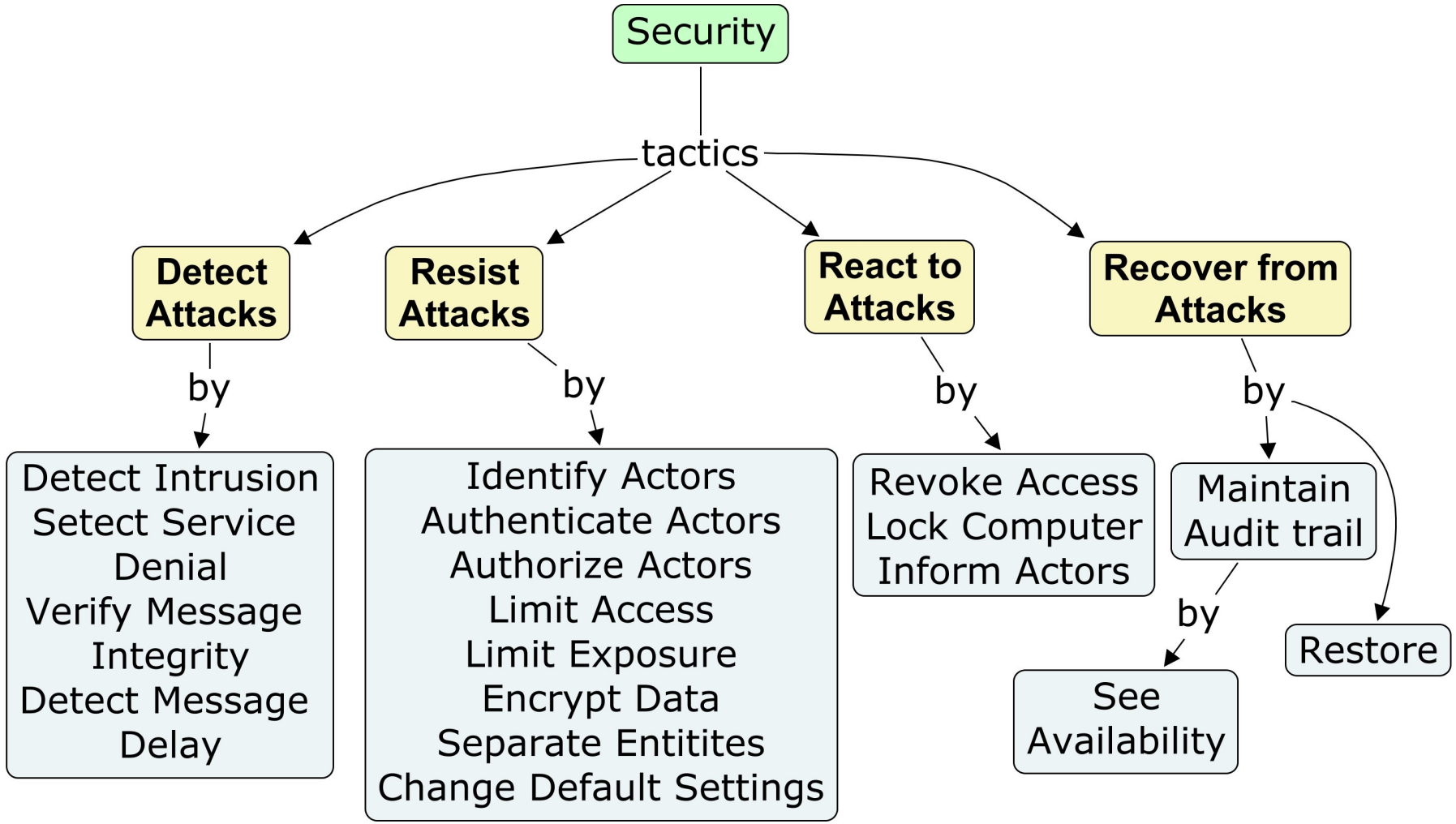
- Performance issues are fun!!
- Think of how you tackled performance in your experience and see what tactics has been applied(could be applied)

Tactics – Part 4

Security

Security

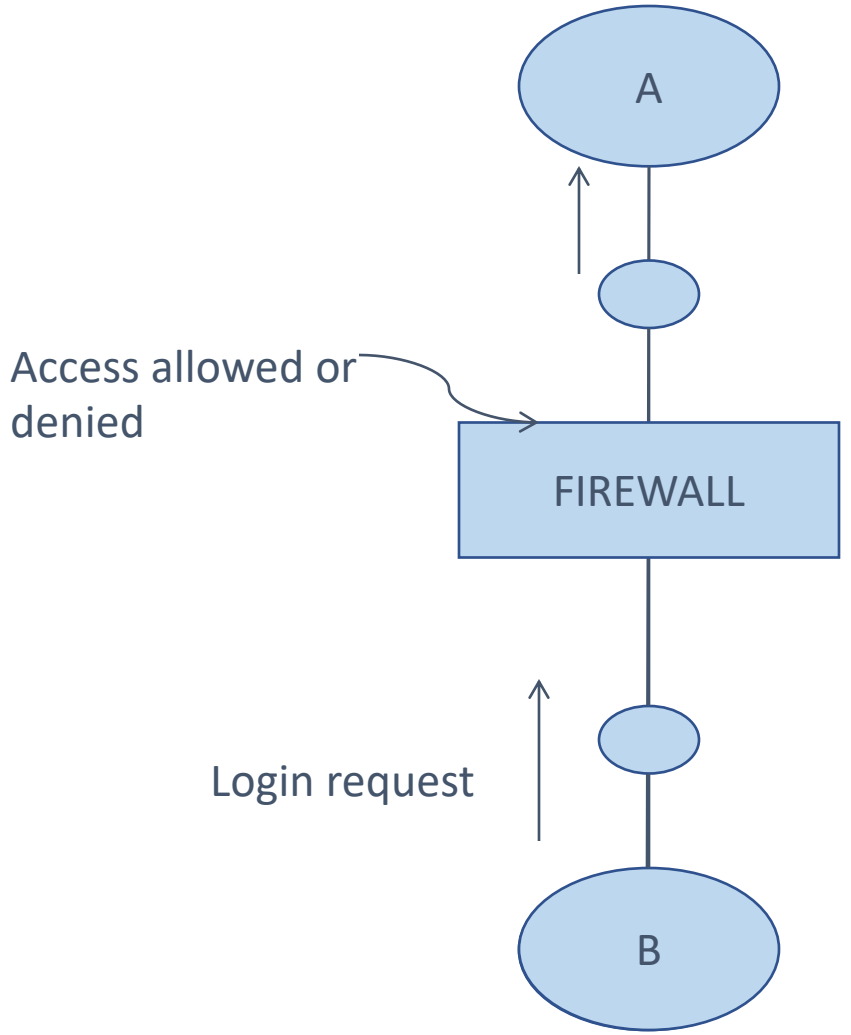
- Prevent unauthorized access
- Permit authorized access
- Attack
  - Unauthorized attempt to access data or services
  - Attempt to deny services to authorized users


Security
tactics
Detect Attacks
Resist Attacks
React to Attacks
Recover from Attacks
by
by
by
by
Detect Intrusion
Setect Service Denial
Verify Message Integrity
Detect Message Delay
Identify Actors
Authenticate Actors
Authorize Actors
Limit Access
Limit Exposure
Encrypt Data
Separate Entitites
Change Default Settings
Revoke Access
Lock Computer
Inform Actors
Maintain Audit trail
by
See Availability
Restore

RESISTING ATTACKS

Authenticate Actors (security..resist attacks..)

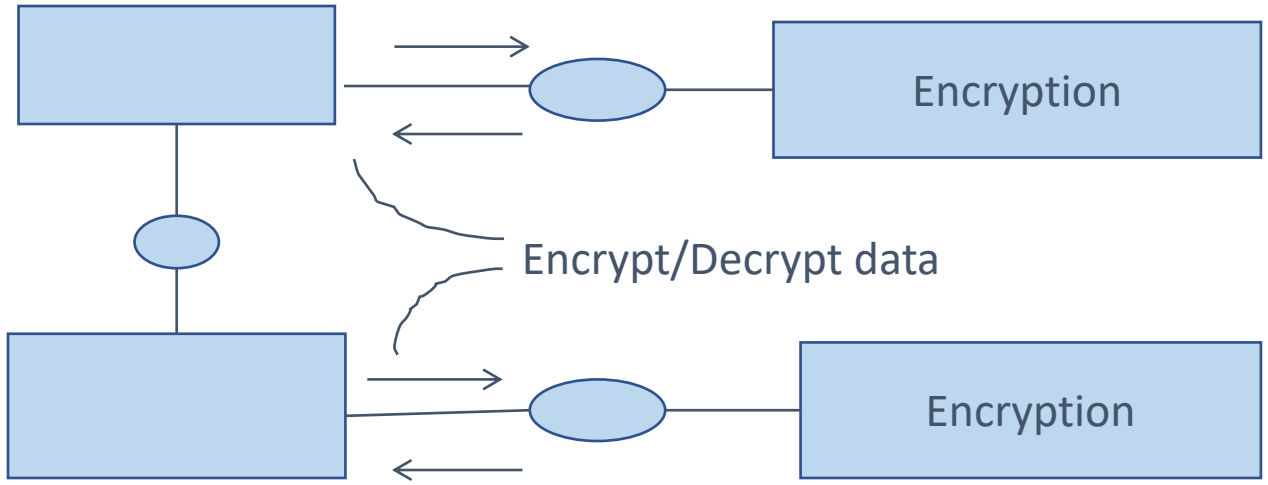
- Make sure the user is who he claims to be
- Passwords, digital signatures, biometric identification …
- A separate component can be used to achieve this



Authorize Actors (security..resist attacks..)

- Ensure that the right user has the right access to data or service
- Normally done by providing some access control patterns – classes, roles, lists..

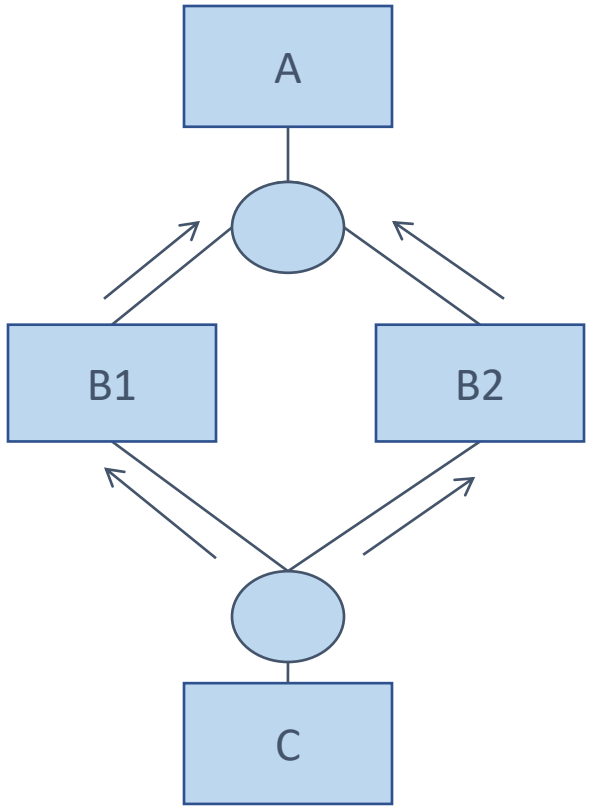
Encrypt Data (security..resist attacks..)



- To ensure no eavesdropping (data confidentiality)
- When sending data over publicly accessed communication links, encrypt
- Decrypt at receiver end

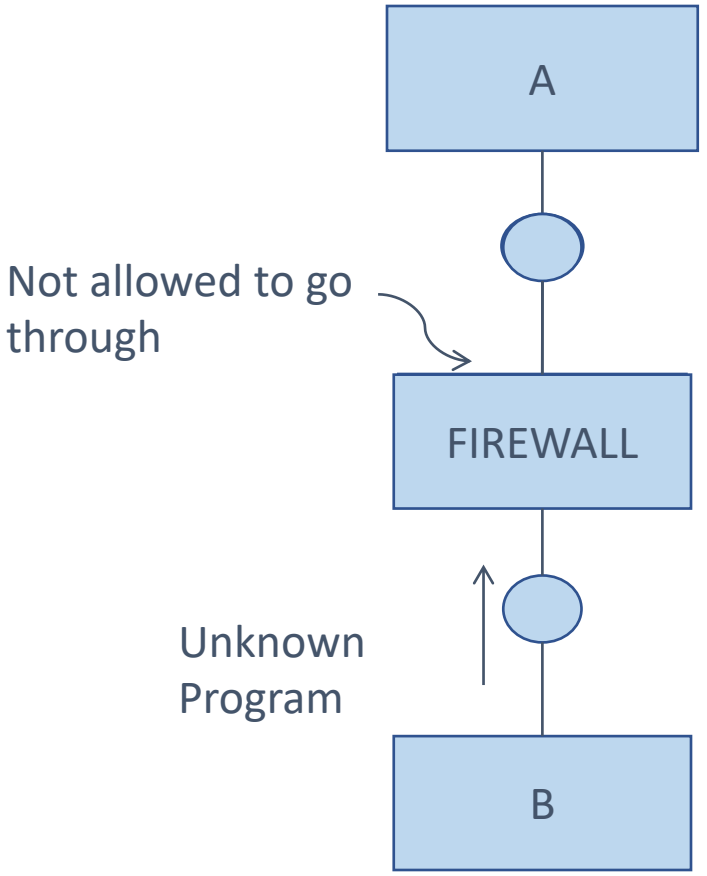
Limit Exposure (security..resist attacks..)

- Attacks normally exploit a single weakness in the system
- Split the system to run on different hosts
- If one host is compromised services corresponding to that host only will go down



Limit Access (security..resist attacks..)

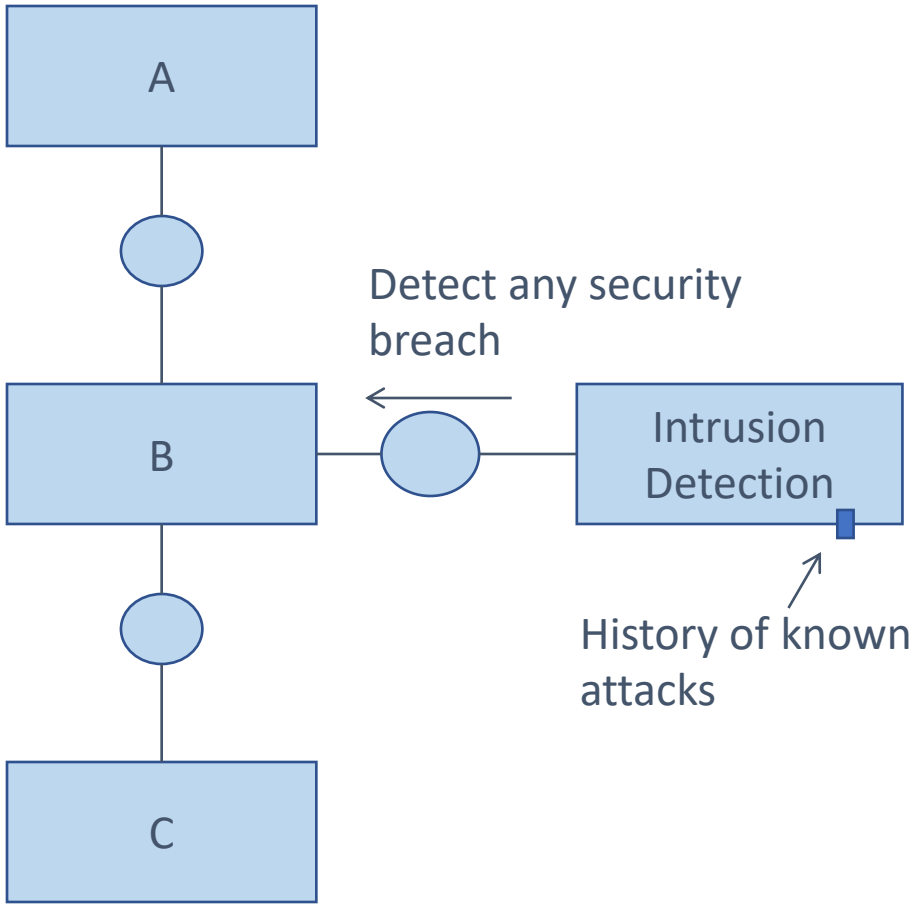
- Firewalls restrict messages/access from unknown sources



Security
(Detect Attacks..)

Detect Intrusion (security..detect attacks)

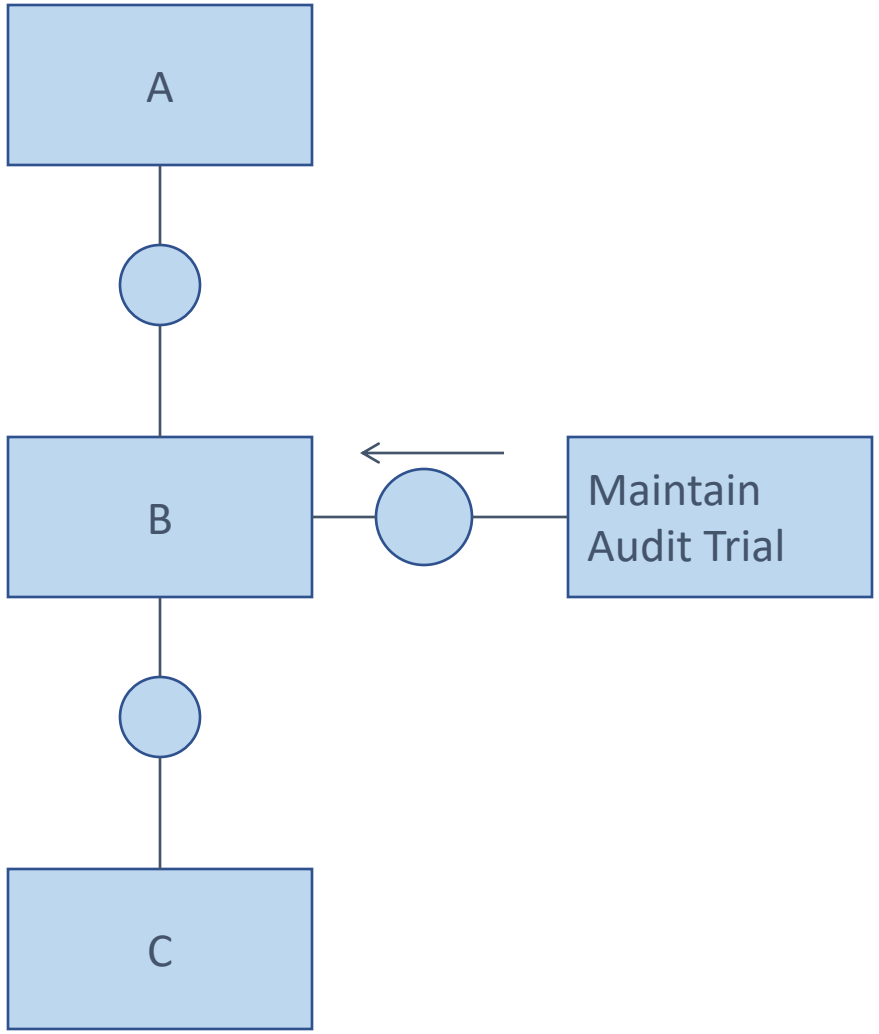
- Keep a record of known attacks and the associated traffic patterns
- Detect an attack by comparing the traffic patterns to the known attacks



Recovering from Attacks (security..)

Maintain Audit Trail (security..recover from attacks..)

- Every change is logged, what and who – an audit trail
- Can be use to trace the actions of the attacker and help establish his identity
- Audit information can also be used to support system recovery



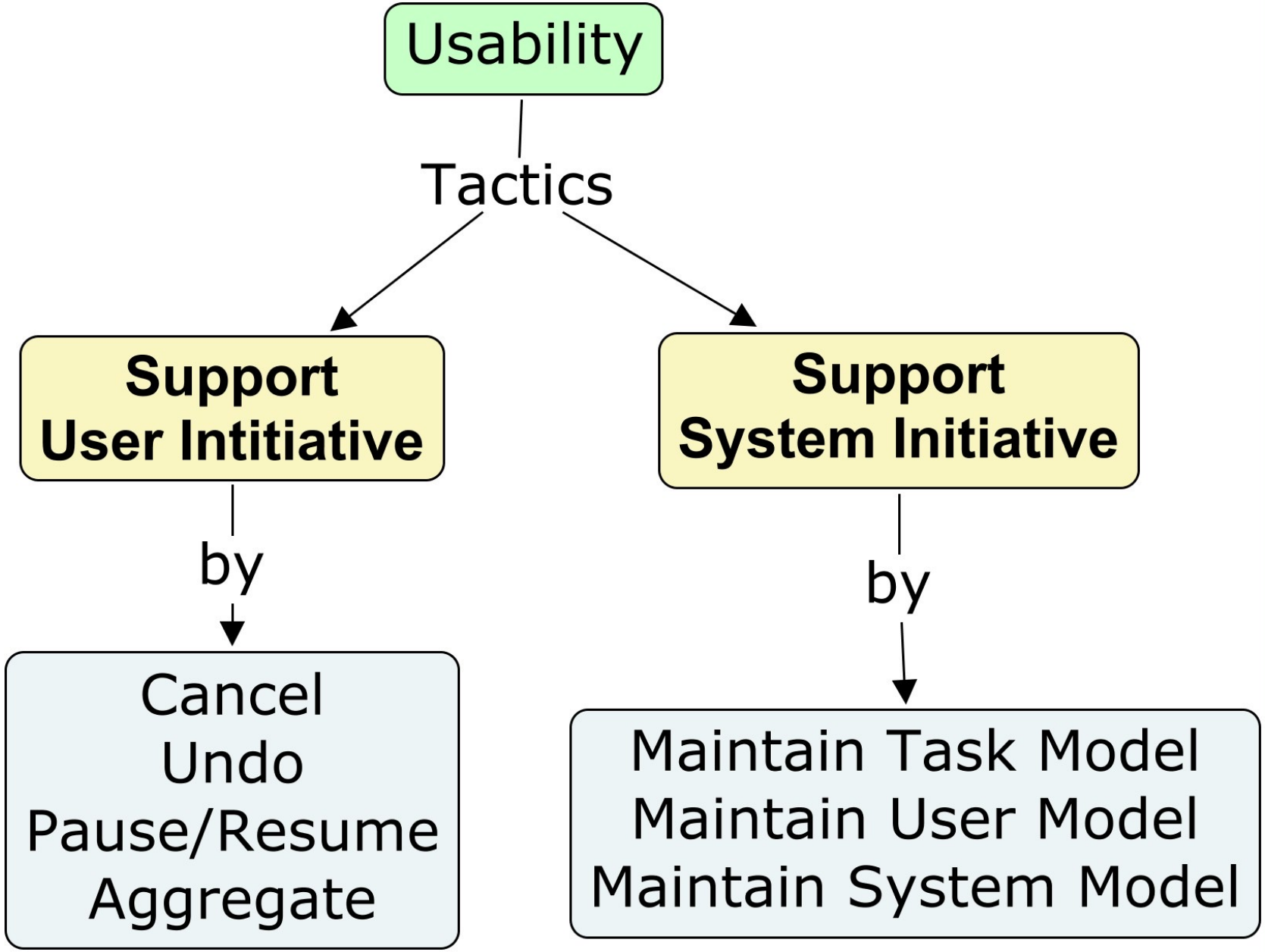
Home Work

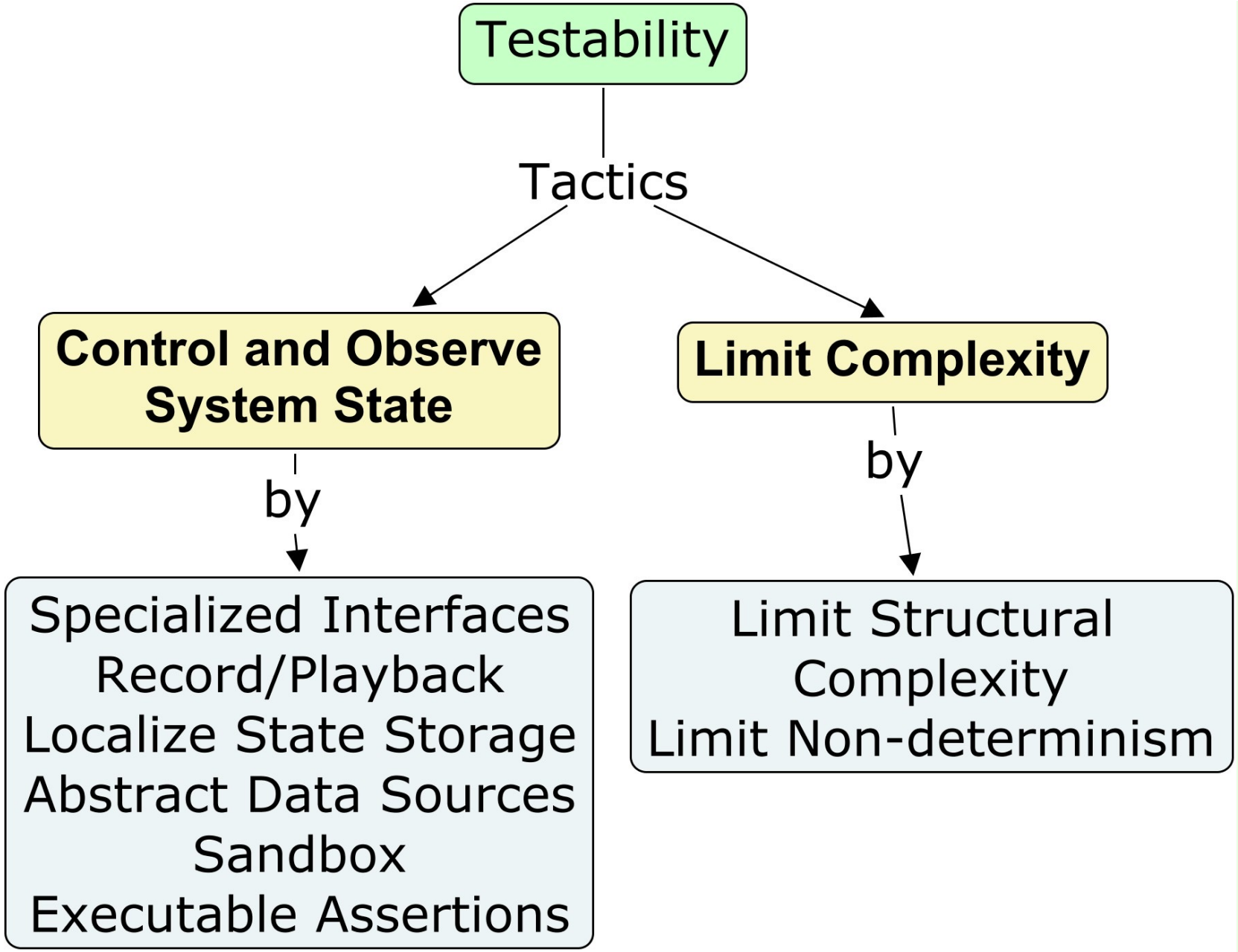
- Security has become very very prime
- Look for mechanisms that you encounter in this space and see how to classify them

Tactics – Part 5

Usability

Testability


Usability
Tactics
Support
User Intitiative
Support
System Initiative
by
by
Cancel
Undo
Pause/Resume
Aggregate
Maintain Task Model
Maintain User Model
Maintain System Model


Testability
Tactics
Control and Observe System State
by
Specialized Interfaces
Record/Playback
Localize State Storage
Abstract Data Sources
Sandbox
Executable Assertions
Limit Complexity
by
Limit Structural Complexity
Limit Non-determinism

Thank You