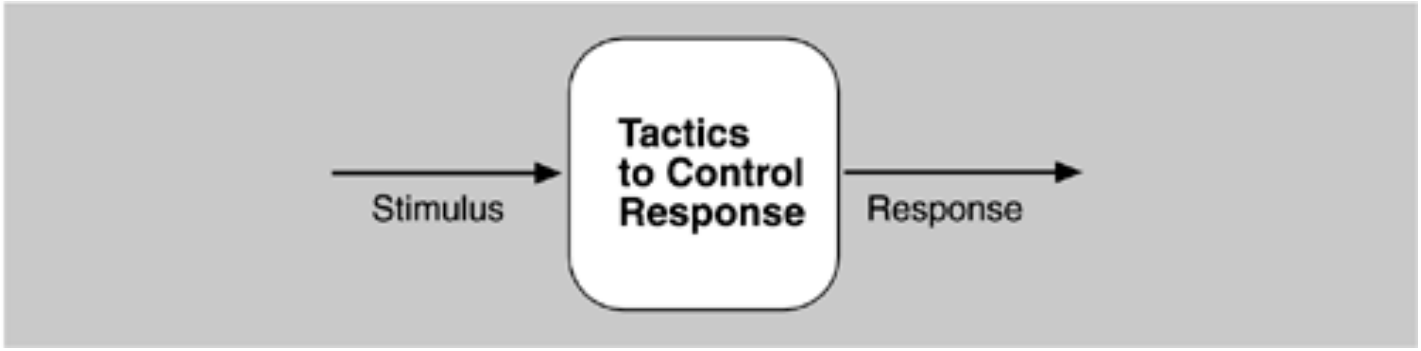
Quality Attribute Tactics

Software Architecture
3rd Year – Semester 1
Lecture 13

What are Tactics?

- An architectural tactic is a means of satisfying a quality attribute response measure by manipulating some aspect of a quality attribute model through architectural decisions



- A planned way to achieve quality goals

More on Tactics…

- A tactic is a design decision that influences the control of a quality attribute response
- A collection of tactics an Architectural Strategy
- Architects usually choose architectural patterns to realize some tactic. But patterns inevitably implement multiple tactics. This can make architectural analysis more difficult.

Achieving Quality Goals

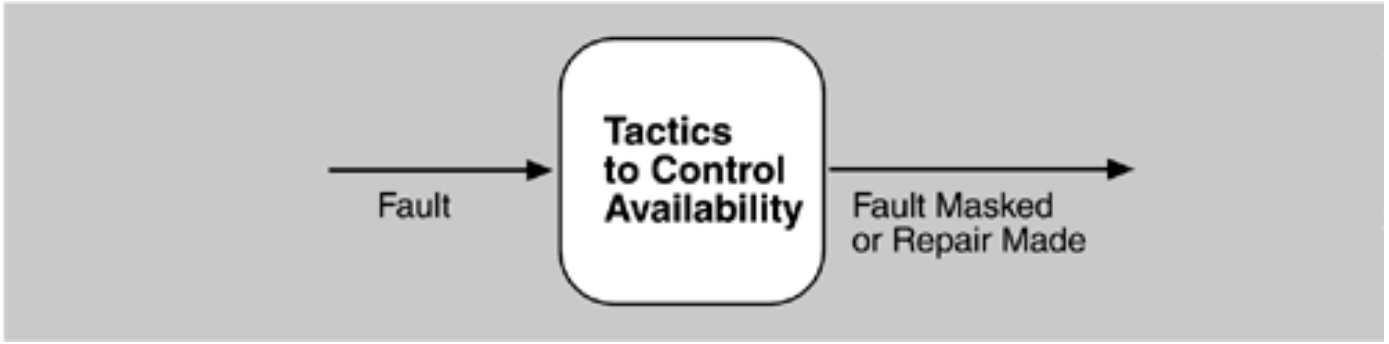
- For each quality, we need to find or formulate tactics to achieve that quality
- Each tactic is a design option for the architect
  - E.g. Increasing Availability: One of the tactics is introducing redundancy. This is one option the architect has to increase availability, but not the only one. Usually achieving high availability through redundancy implies a need for synchronization

Applying Tactics

- Tactics can refine other tactics:
  - We identified redundancy as a tactic. As such, it can be refined into redundancy of data (in a database system) or redundancy of computation (in an embedded control system). Both types are also tactics. There are further refinements that a designer can employ to make each type of redundancy more concrete. For each quality attribute that we discuss, we organize the tactics as a hierarchy.
- Patterns package tactics:
  - There can be multiple tactics to achieve a single goal . For example availability.
  - Availability tactics package include Redundancy tactic and a Synchronization tactic

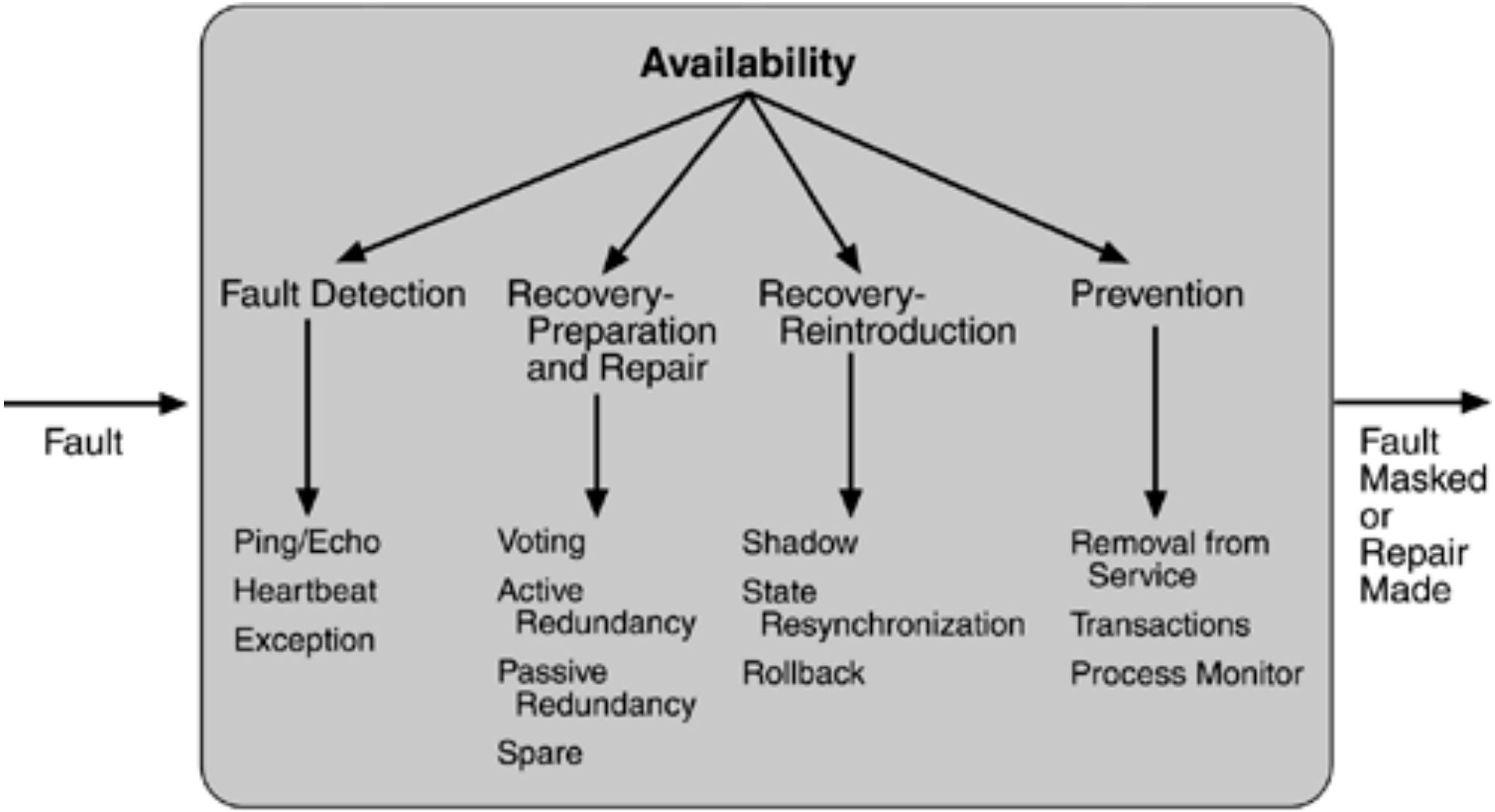
Availability Tactics

- A fault (or combination of faults) has the potential to cause a failure and would affect availability.



- Availability Tactics will keep faults from becoming failures or at least bound the effects of the fault and make repair possible.
  - Fault Detection
  - Fault Recovery
  - Fault Prevention

Availability Tactics Framework



Availability Tactics in Detail

- Fault Detection
  - Ping/echo: send the component a ping and wait for an echo. If not received, notify the fault correction system.
    - Are you alive?
  - Heartbeat: have the component emit a heartbeat periodically, and have another component listen. If not received, notify the fault correction system.
    - I am alive…
  - Exceptions: trigger an exception handler when a fault occurs (most suitable for within the process/component)
    - I'm dead

Availability Tactics in Detail

- Fault Recovery (Preparation & Repair)
  - Voting: have redundant processors and have the processes vote for the answer. If one is different, fail it. No downtime.
    - Majority rules or Preferred Component priority
  - Active redundancy: all redundant components respond to all events, output is taken from the first to respond
  - Passive redundancy: only the master responds to events, but the backup's state is updated so it can take over whenever necessary. Downtime is seconds.
  - Spare: Keep a standby spare component to replace many different failed components. Downtime is minutes.
    - Here you make checkpoints of the system state periodically and log all state changes

Availability Tactics in Detail

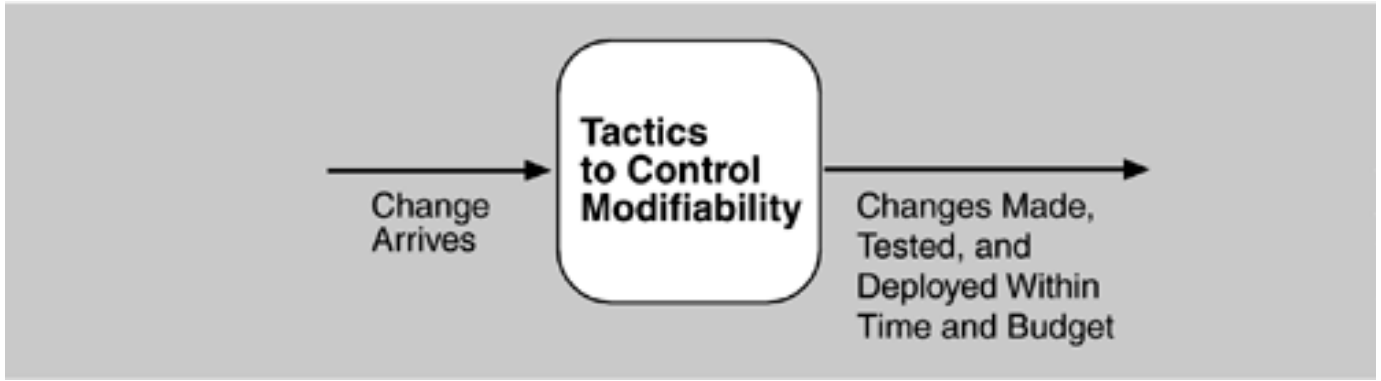
- Fault Recovery (Reintroduction)
  - Shadow operation: have a previously failed component mimic the backup for a while to verify correct operation.
  - State resynchronization: when a failed component is returned to service, its state needs to be resynchronized with the backup.
  - Checkpoint/rollback: record consistent states, and when a fault occurs, restore to the checkpoint.

Availability Tactics in Detail

- Fault Prevention
  - Removal from surface: take a component down periodically to prevent anticipated failures, e.g. reboots to prevent memory leaks from leading to failure.
  - Transactions: bundle sequential steps into chunks that can be undone all at once in case of a fault on an intermediate step.
  - Process monitor: monitor for faulting processes, and kill and restart when a fault is detected.

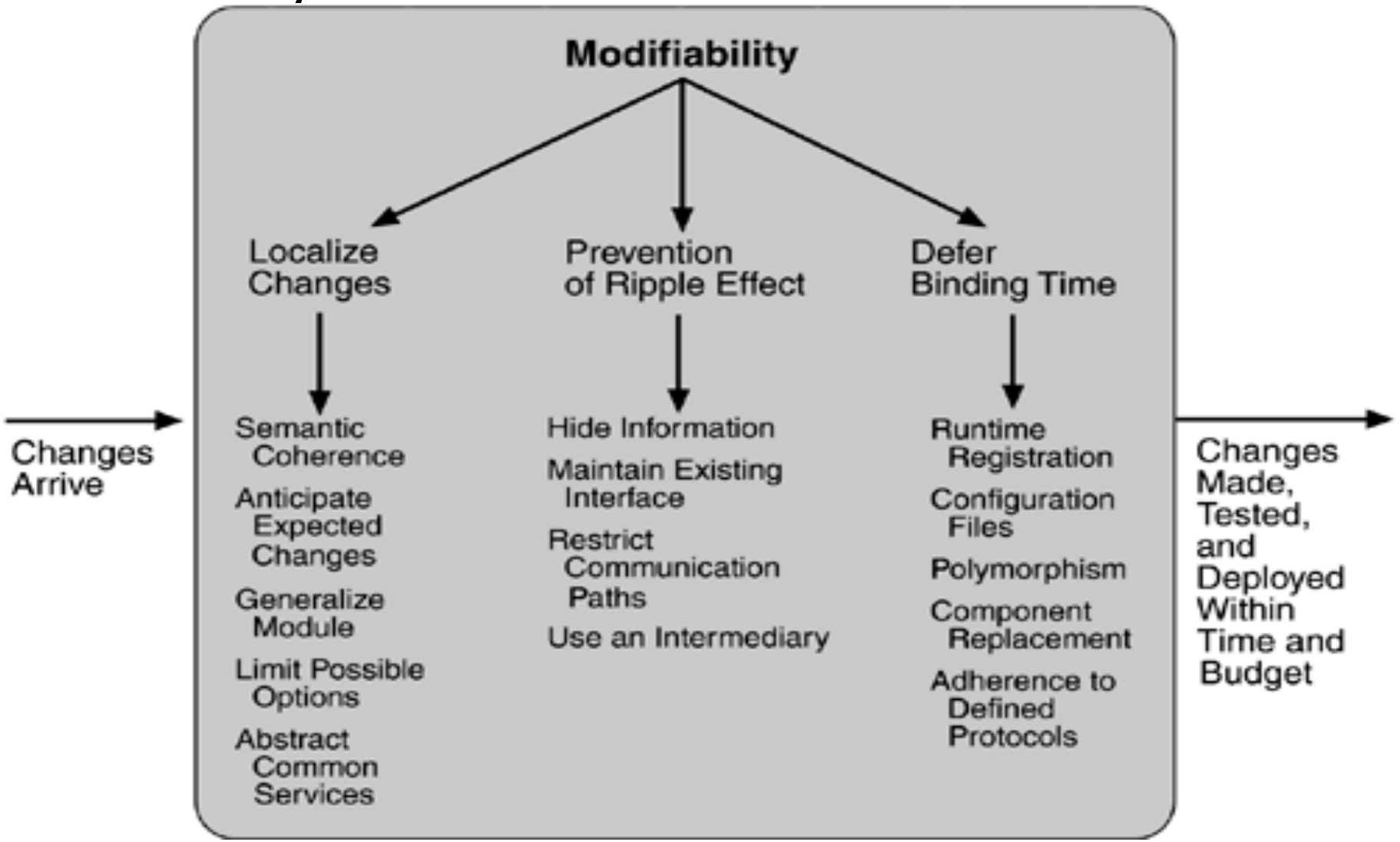
Modifiability Tactics

- Goal is to control the time and cost to implement, test, and deploy changes.



- Tactics for modifiability is organized according to their goals
  - Localize modifications: Reducing the number of modules affected
  - Prevent ripple effects: Limiting the modification to localized modules
  - Defer binding time: Controlling deployment time and cost

Modifiability Tactics Framework



Modifiability Tactics in Detail

- Goal: Reduce the number of modules affected by a change
- Localize Modifications
  - Maintain semantic coherence — keep things that are related together
  - Anticipate expected changes — keep things that are likely to change in one place
  - Generalize the module — making a module more general allows it to compute a broader range of functions based on input
  - Limit possible options — don’t allow much change

Modifiability Tactics in Detail

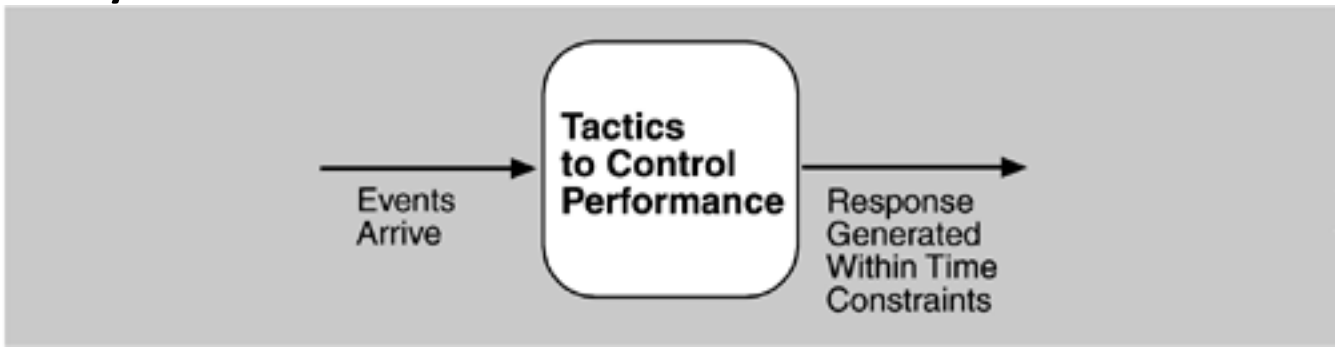
- Goal: Limited modifications of the module due to changed on the dependent modules
- Prevent Ripple Effect
  - Hide information — less visible information means fewer things to be dependent on
  - Maintain existing interfaces — don’t change interfaces, use patterns such as an adapter
  - Restrict communication paths — don’t allow data to flow through too many modules (reduce consumptions and production)
  - Use an intermediary — break dependency chain (not for semantic changes), use patterns such as Façade, Mediator, Delegate, Proxy

Modifiability Tactics in Detail

- Goal: Control deployment time and cost, allow deployment or operational time changes
- Differ Binding Time
  - Runtime registration — have participants identify themselves after the system has begun operation (plug & play)
  - Configuration files — initiation time
  - Polymorphism — what method names mean determined at runtime
  - Component replacement — runtime elements can be changed at load time
  - Adherence to defined protocols — runtime binding of independent processes

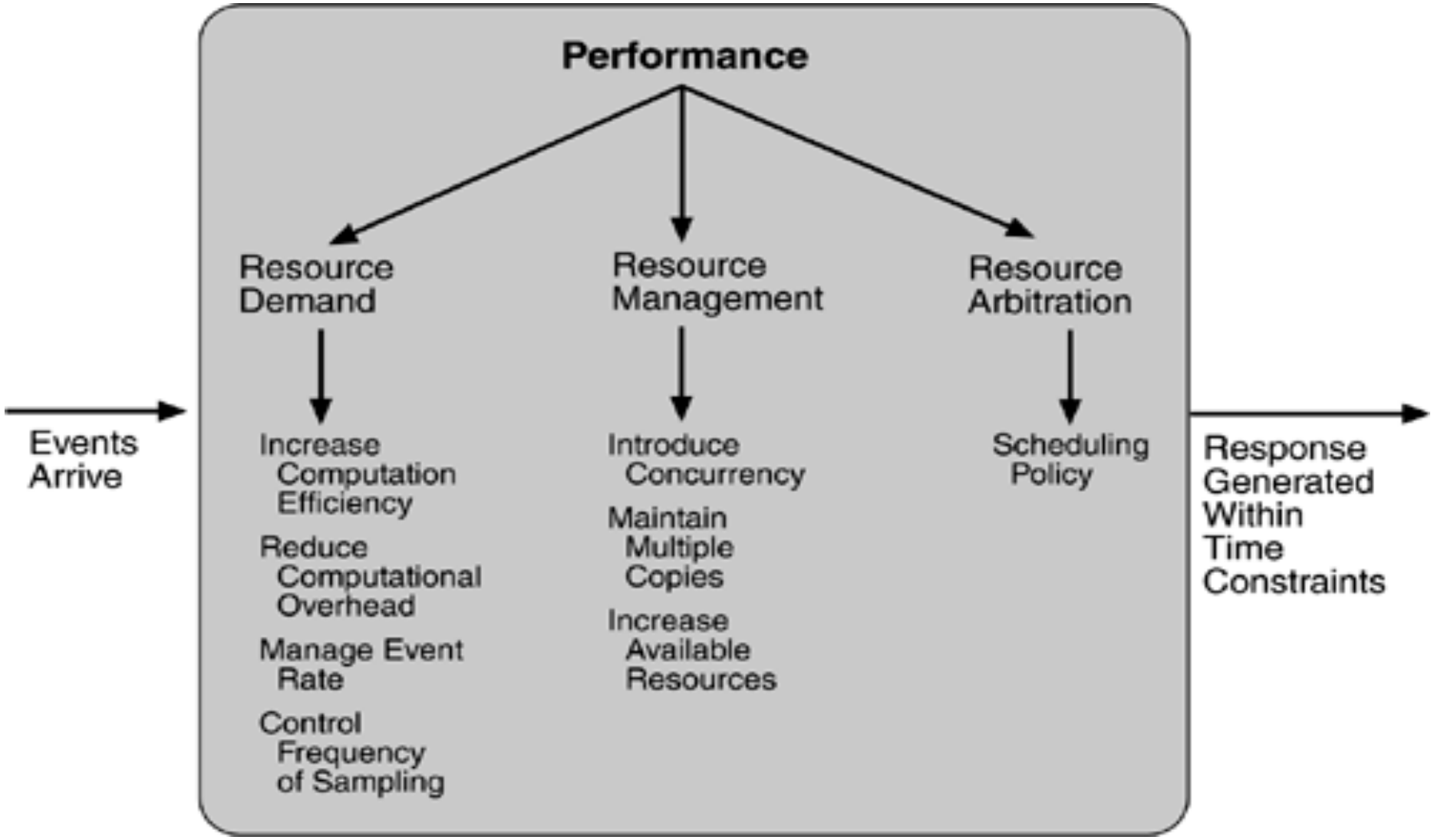
Performance Tactics

- The goal of performance tactics is to generate a response to an event arriving at the system within some time constraint.



- Main contributing factors are…
  - Resource Consumption – e.g. CPU, Disk I/O, etc…
  - Blocked Time – Availability and Dependencies of the Resources, Multi Process Priorities or Policies

Performance Tactics Framework



Performance Tactics in Detail

- Resource Demand
  - Increase computational efficiency – More efficient Algorithms. E.g. Bubble sort Vs. Quick Sort
  - Reduce computational overhead – e.g. without calculating a value in multiple places calculate once and refer to the value (final static double pi = 22/7)
  - Control Frequency of Sampling – Reduce the sampling frequencies (e.g. listeners, watchers)

Performance Tactics in Detail

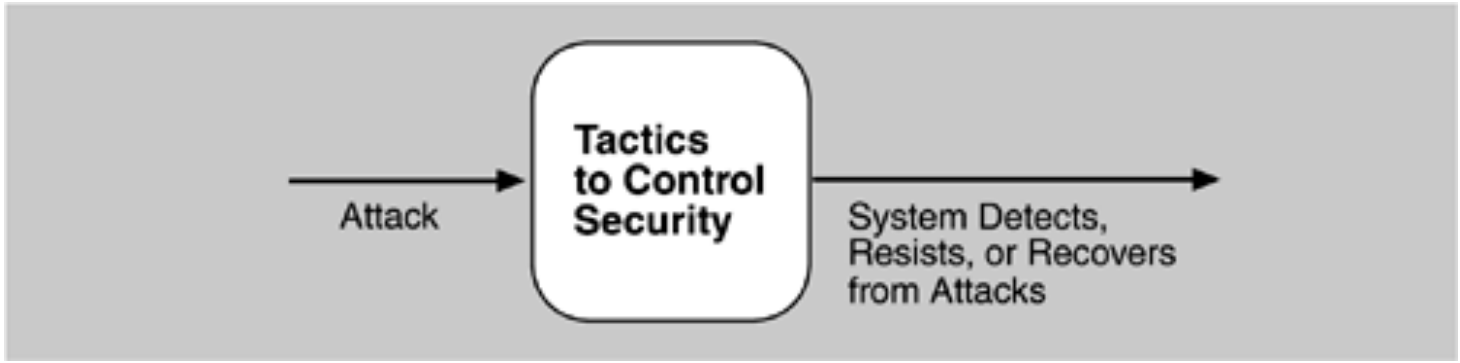
- Resource Management
  - Introduce concurrency - If requests can be processed in parallel , the blocked time can be reduced
  - Maintain multiple copies of either data or computations – cache should be consistent and synchronized.
  - Increase available resources – faster or additional processors, memory or networks

Performance Tactics in Detail

- Resource Arbitration
  - Scheduling Policy – When there is a contention for a resource (Disk IO, Network, etc…) processes should be scheduled
    - Optimal Resource Usage
    - Maximize Throughput
    - Ensure fairness
    - Prevent starvation
    - Scheduling Policies – FIFO, Fixed, Round Robin, etc…

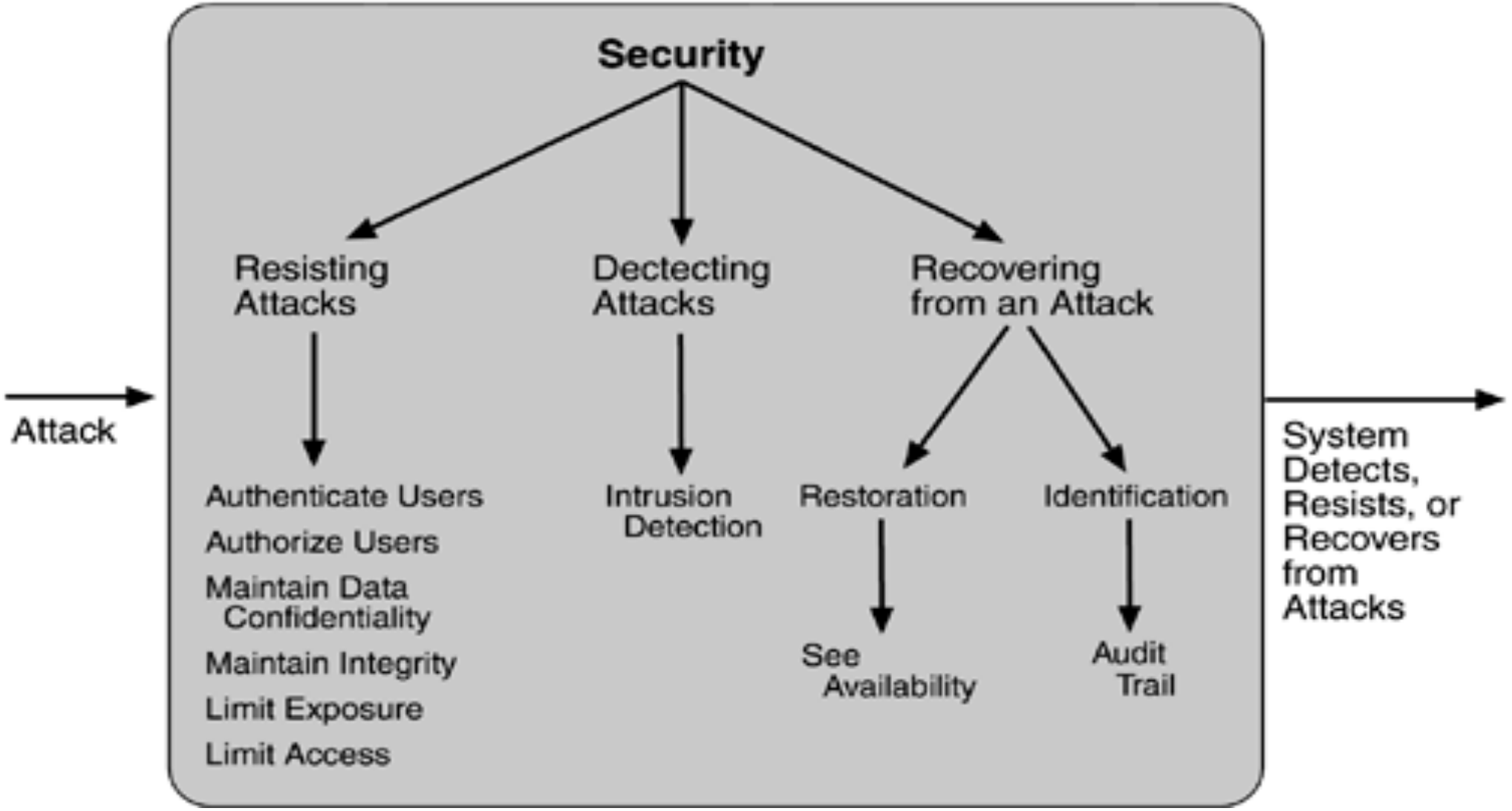
Security Tactics

- Goal is to ensure Non-repudiation, Confidentiality, Integrity and Assurance.



- Main concerns are…
  - Resisting Attacks
  - Detecting Attacks
  - Recovery from Attacks

Security Tactics Framework



Security Tactics in Detail

- Resisting Attacks
  - Authenticate users – Ensuring that a user is actually who it's purport to be
  - Authorize users – Ensuring that an authenticated user has the right to access
  - Maintain data confidentiality – Data should be protected from unauthorized access — encrypt data and communication links (VPN, SSL)
  - Limit exposure – avoid single point of failure, limited services per host
  - Limit access — firewall, DMZ

Security Tactics in Detail

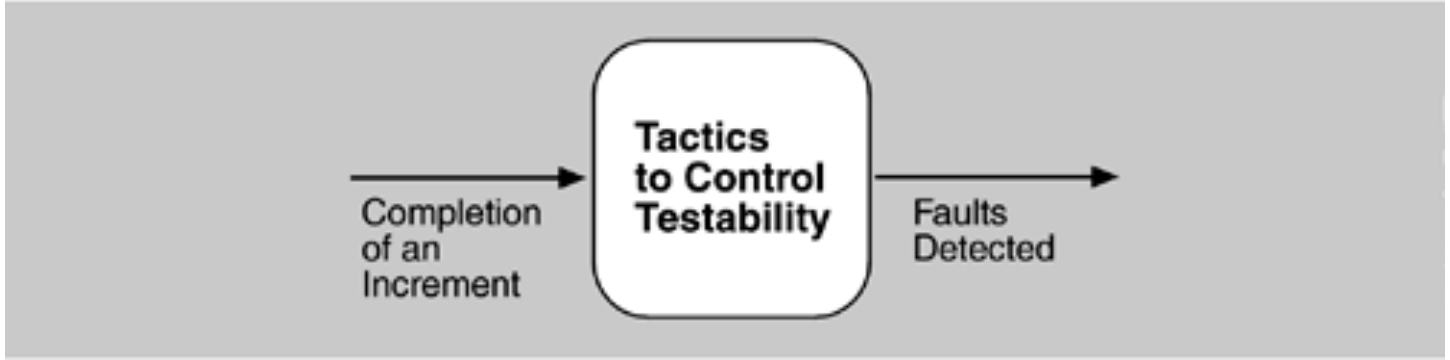
- Detecting Attacks
  - Intrusion Detection – Historical Statistics can be compared with current
    - Server Traffic
    - Resource Usage
    - Monitoring Networks

Security Tactics in Detail

- Recovery from Attacks
  - Restoration
    - Availability tactics can be used with special attention
    - Maintaining redundant copies of system & data
  - Identification (To identify an attacker)
    - Maintain an audit trail (can it be attacked?)
    - Can be used to trace the actions of an attacker
    - So it becomes an attacked target, need to be maintained in a trusted environment

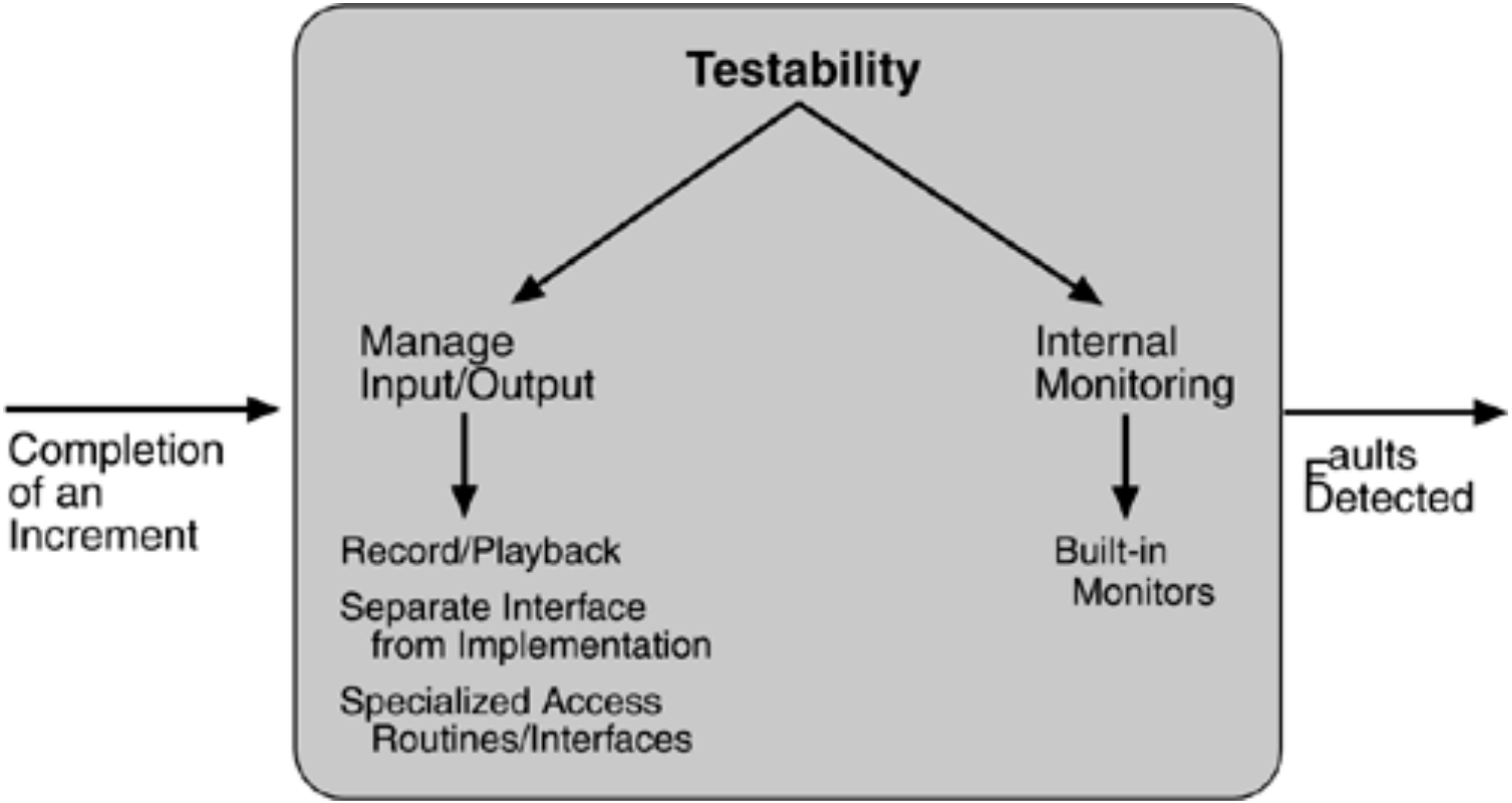
Testability Tactics

- Goal is to allow easier testing when an increment of software development is completed



- The main focus is runtime testing. This requires that input be provided to the software and output be captured
  - Input / Output
  - Internal Monitoring

Testability Tactics Framework



Testability Tactics in Detail

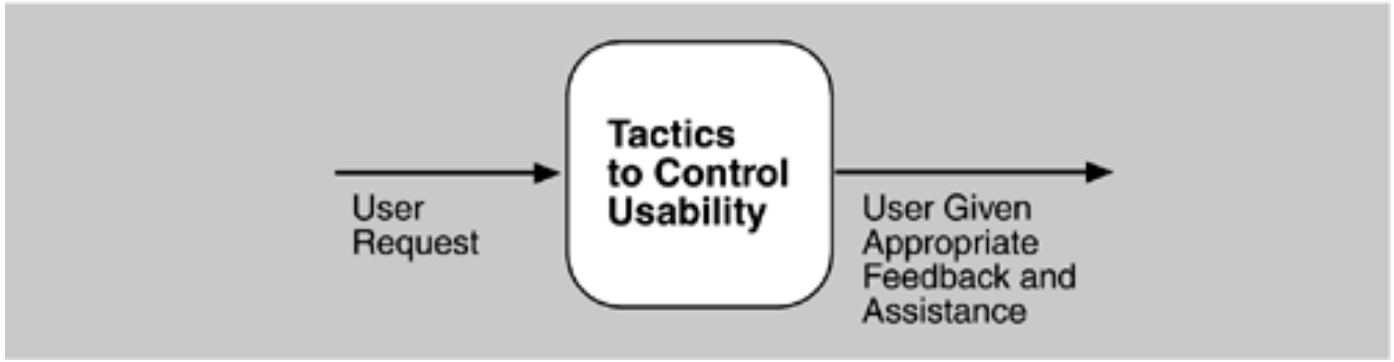
- Input / Output
  - Record/playback – capturing information crossing an interface using it as input into the test harness
  - Separate interface from implementation - Stubbing implementations allows the remainder of the system to be tested in absence of the component being stubbed (mock testing)
  - Specialize access routes/interfaces – hierarchy of test interfaces (have to be cautious on exposing unwanted interfaces)

Testability Tactics in Detail

- Internal Monitoring
  - Built-in monitors
    - This interface can be a permanent interface of the component or it can be introduced temporarily.
    - Would help to test other Runtime Quality Attributes such as Performance, Availability, Security, etc…

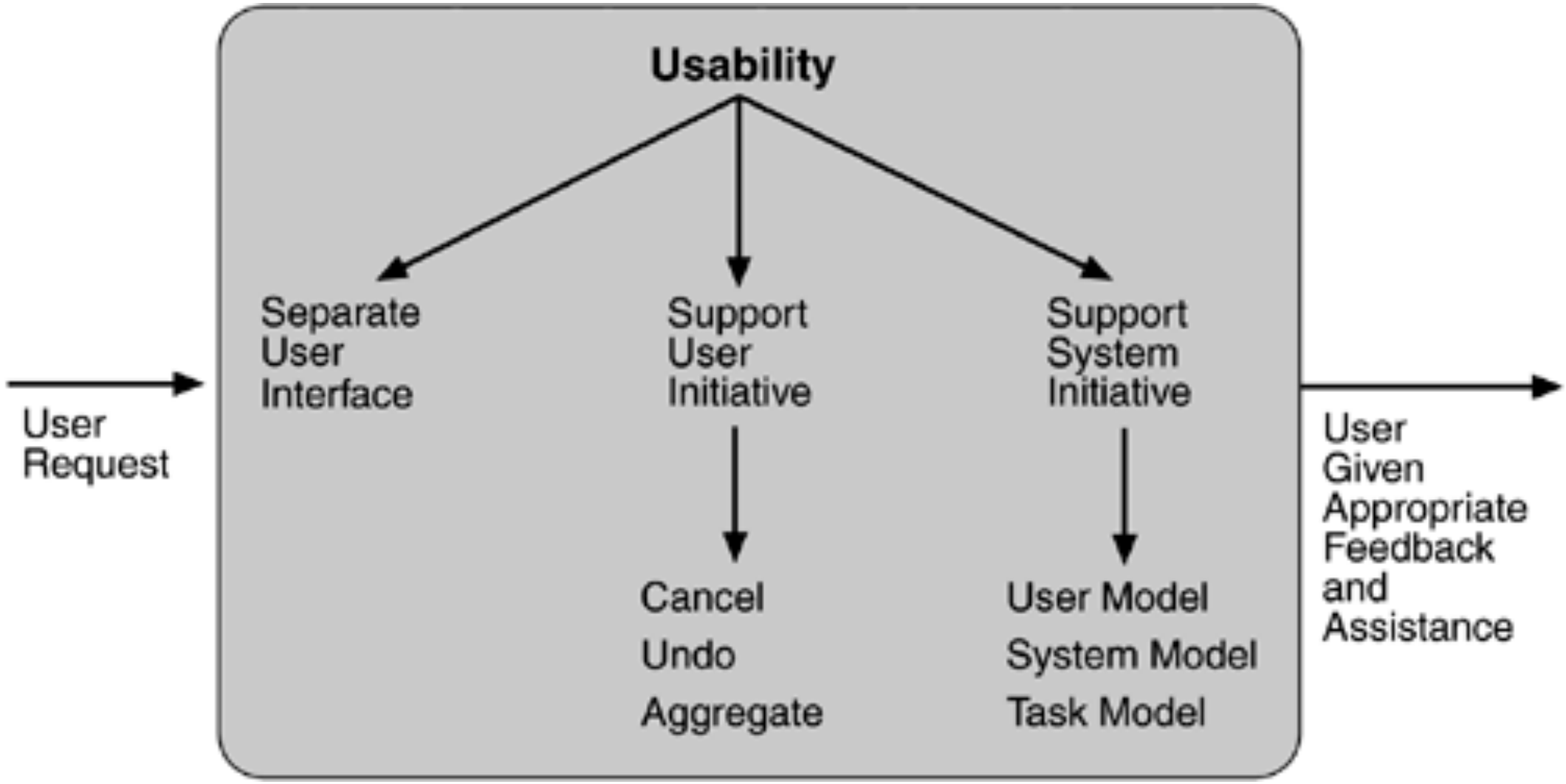
Usability Tactics

- Goal is to provide an easy way to the user to accomplish a desired task and the kind of support the system provides to the user



- The main areas are…
  - Runtime Tactics
  - Design Time Tactics

Usability Tactics Framework



Usability Tactics in Detail

- Runtime Tactics
  - User Initiatives – Operations to help the user such as Pause Operation, Cancel, Undo, etc…
  - System Initiatives – Provide feedback, suggestions, notifications, conformations, etc…
  - Mixes Initiatives – When user hits “Upload” the system gives a Progress Bar

Usability Tactics in Detail

- Design Time Tactics
  - Separate the user interface from the rest of the application
    - Separate User Interface Components – MVC
    - Maintain Semantic Coherence – Localizing expected changes.

References

- http://proquestcombo.safaribooksonline.com/0321154959
- http://www.ece.ubc.ca/~matei/EECE417/BASS/ch05.html

Exercise (offline)

- Refer a few existing software systems (e.g. your Group Case Study) and identify what are the important quality attributes for that system
  - Identify about 4-5 Quality Attributes
- Suggest suitable tactics for implementing the identified Quality Attributes
  - Provide justification for your answers
- Cross check how the system actually improve the identified quality attribute with what you have suggested
  - If the system does not follow the tactic you suggested research on why? Hint Trade-Offs