Defining Architecture

Candidate Architecture

Candidate architecture is a proposed addition or change to the architecture.

Baseline architecture

A *baseline architecture* describes the existing system

In the future Candidate architecture becomes the baseline from which new candidate architectures can be created and tested.

Architectural Spikes

An *architectural spike* is a test implementation of a small part of the application's overall design or architecture.

The purpose is to analyze a technical aspect of a specific piece of the solution to choose between potential designs and implementation strategies, or sometimes to estimate implementation timescales.

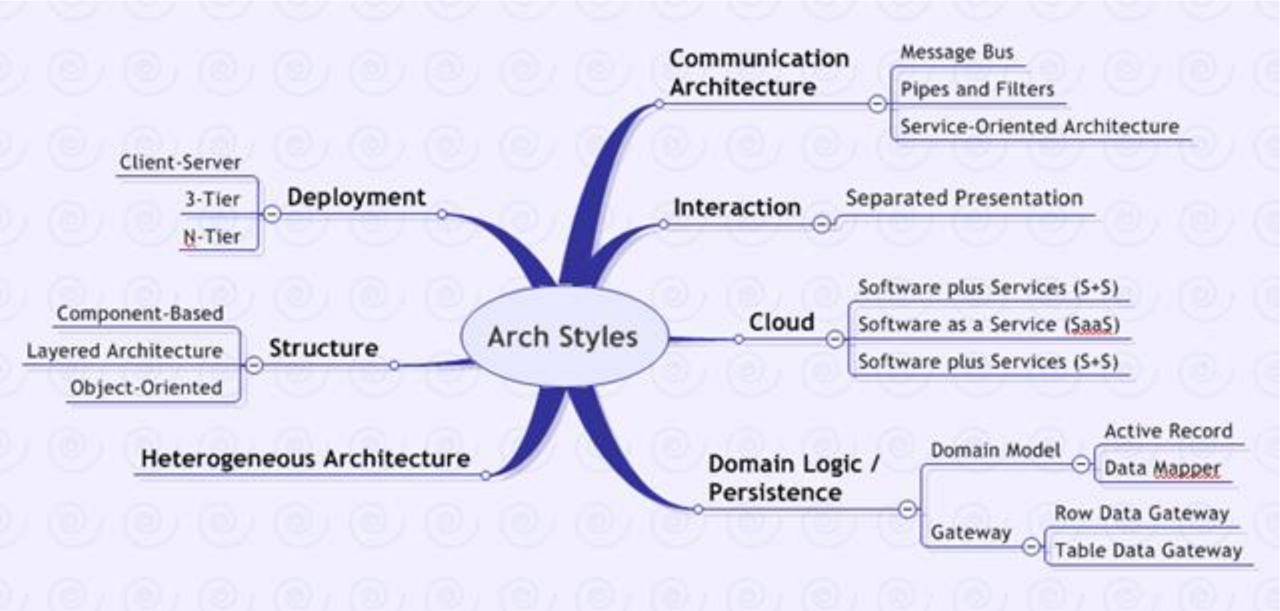
Reference Architecture

A ***reference architecture*** provides a proven template solution for an architecture for a particular domain.

Reference architecture are often

harvested from previous projects.

Architectural Style/ Pattern



You can think of architectural styles as a collection of principles that shape or govern the design of your application.

Design Pattern

- **Architectural** Styles
  - Provides their own design language
  - Focus on large grain components
- Design **Patterns**
  - utilize UML as the design language
  - Focus on objects
- **Architectural** Styles
  - tend to solve system wide problems
  - System wide analysis
- Design **Patterns**
  - tend to solve small, specific problems
  - Can easily be translated into code

Architectural Tactics

Architectural-strategy is a collection of tactics

An architectural *tactic is a fine-grained design approach* used to achieve a quality attribute response.

You should not try to build your architecture in a single iteration. Each iteration should add more detail.

Attribute-Driven Design Method

©2006 Sean Kenney http://www.seankenney.com

A recursive decomposition process where, at each stage, tactics and architectural patterns are chosen to satisfy a set of quality scenarios and then functionality is allocated to instantiate the module types provided by the patterns.

Attribute-Driven Design (ADD) Method

ADD Inputs

1. Quality attribute scenarios
2. Functional requirements
3. Constraints

ADD Outputs

1. Several levels of a module decomposition
2. Interconnection and coordination mechanisms
3. Application of patterns and tactics to specific modules
4. Explicit achievement of quality attribute requirements
5. NOT detailed interfaces

Attribute-Driven Design Method

1 Confirm there is sufficient requirements information.

2 Choose part of the system to decompose.

3 Prioritize requirements and identify architectural drivers.

4 Choose patterns, tactics for the module that satisfies the drivers.

5 Instantiate architectural elements and allocate functionality

6 Define interfaces for instantiated elements

7 Verify and refine requirements

Iterative

Building Upon QAW results

1. List of architectural drivers
2. The scenarios
3. A prioritized list of scenarios
4. Refined scenarios.

Attribute-Driven Design Method

1. **Choose part of the system to decompose.**
2. Prioritize requirements and identify architectural drivers.
3. Choose patterns, tactics for the module that satisfies the drivers.
4. Instantiate architectural elements and allocate functionality
5. Define interfaces for instantiated elements
6. Verify and refine requirements

Iterative

Choose Part of the System to Decompose

(COURTESY BRICKARTIST.COM)

Just starting out?

- Choose the “part” is the whole system

Otherwise

- choose a part identified from an earlier iteration

How to choose?

1. **Risk**. Design the high-risk pieces first.
2. **Progress and hand-off**. Design the low-risk (i.e., simple) pieces quickly, to begin implementation.
3. **Importance**. Design the important pieces (in terms of business context) first.
4. **Depth first**. Choose a part of the system and “drive” its design to completion
5. **Breadth first**. Make sure there are no major unknowns lurking at the high levels.
6. **Prototype building**. Design enough (and in the right areas) to build a prototype early on.

Attribute-Driven Design Method

1. Confirm there is sufficient requirements information.
2. Choose part of the system to decompose.
3. **Prioritize requirements and identify architectural drivers.**
4. Choose patterns, tactics for the module that satisfies the drivers.
5. Instantiate architectural elements and allocate functionality
6. Define interfaces for instantiated elements
7. Verify and refine requirements

Iterative

Prioritize requirements and identify architectural drivers

Architectural drivers “shape” the architecture.

1. *Functional* requirements
2. *Quality* requirements
3. Constraints

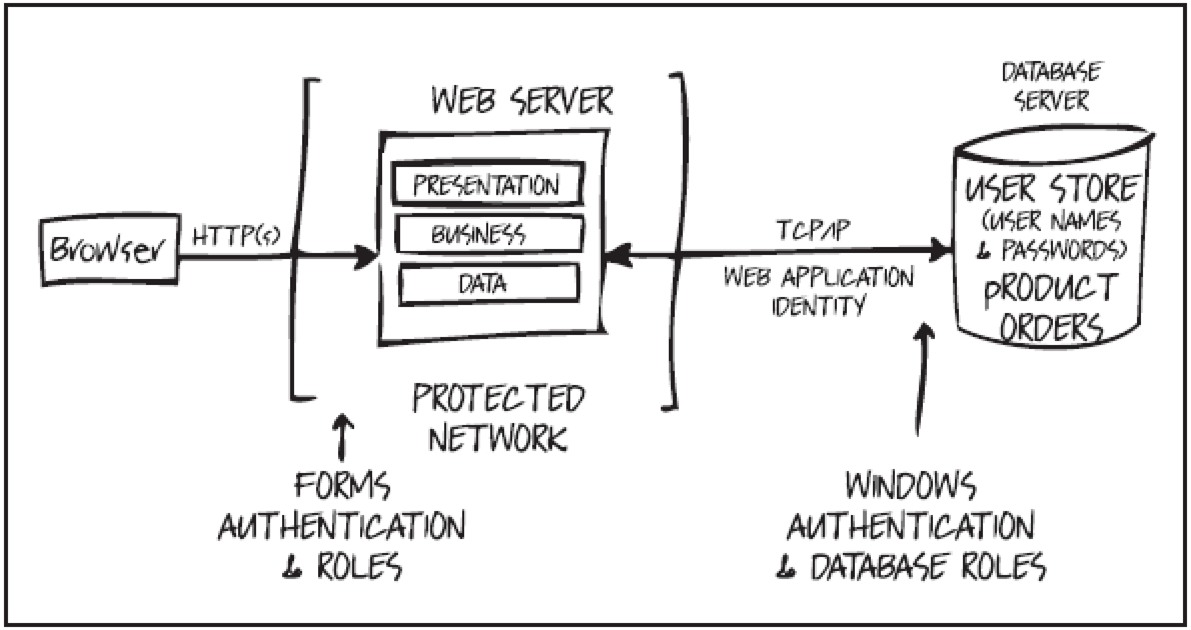
Output of QAW

Attribute-Driven Design Method

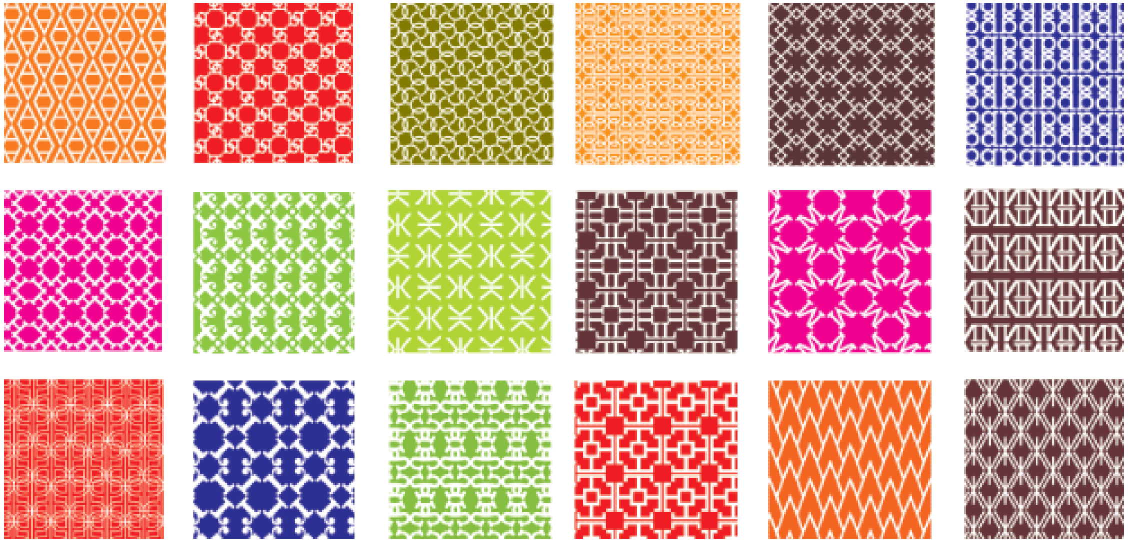
1. Confirm there is sufficient requirements information.
2. Choose part of the system to decompose.
3. Prioritize requirements and identify architectural drivers.
4. **Choose patterns, tactics for the module that satisfies drivers.**
5. Instantiate architectural elements and allocate functionality
6. Define interfaces for instantiated elements
7. Verify and refine requirements

Iterative

Whiteboard Your Architecture



Choose patterns, tactics for the module that satisfies the drivers



Choose design concept – patterns, styles, tactics -- that satisfies the architectural drivers associated with the part of the system we’ve chosen to decompose.

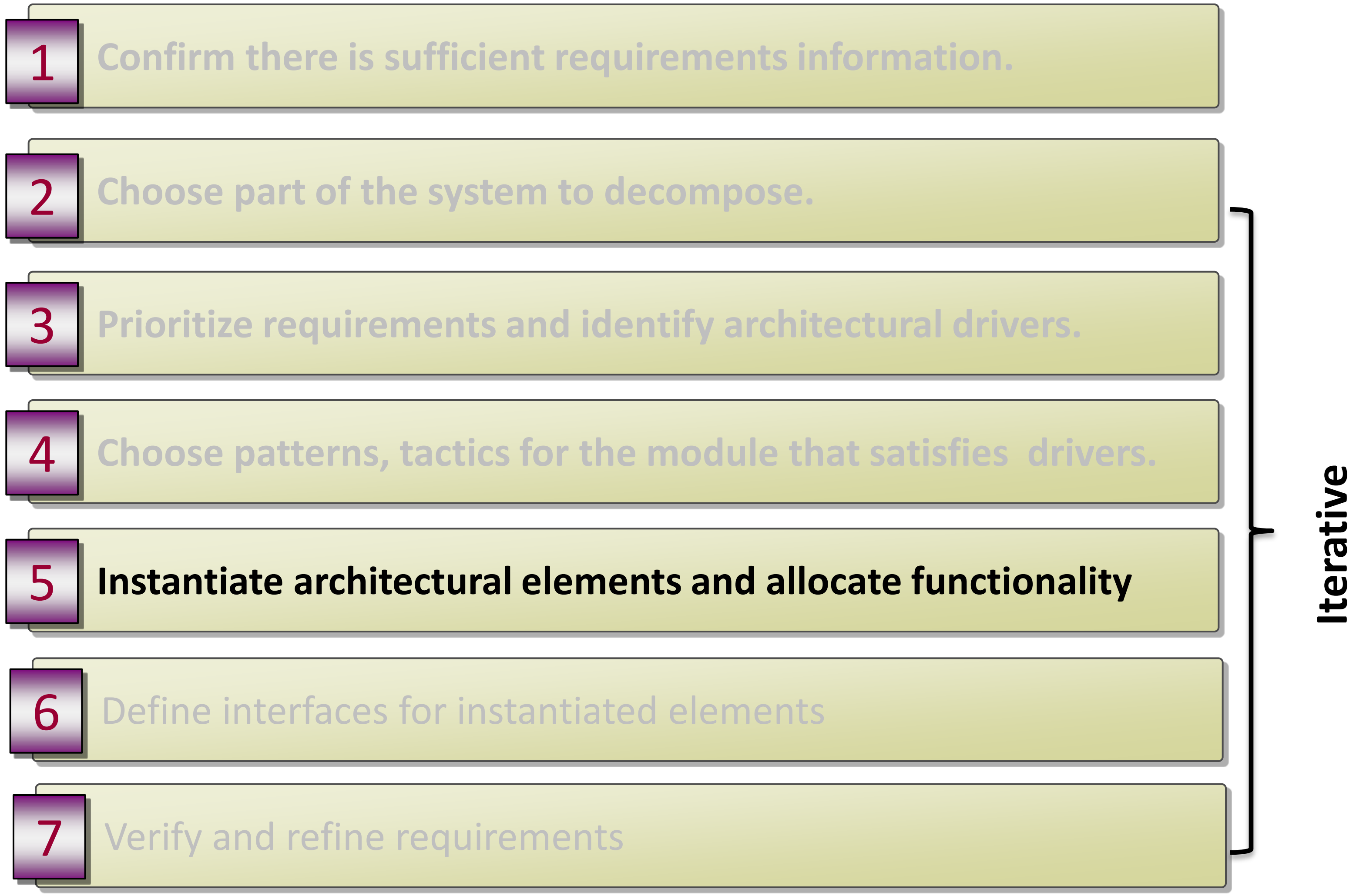
Choose patterns, tactics for the module that satisfies the drivers

1. Start by trying to apply an architectural pattern.
   - E.g. client-server

2. If necessary, apply a combination of patterns.
   - E.g., layered client-server

3. If necessary, augment the pattern(s) with tactics.
   - E.g., layered client-server with ping-echo interaction

Attribute-Driven Design Method



1 Confirm there is sufficient requirements information.

2 Choose part of the system to decompose.

3 Prioritize requirements and identify architectural drivers.

4 Choose patterns, tactics for the module that satisfies drivers.

5 **Instantiate architectural elements and allocate functionality**

6 Define interfaces for instantiated elements

7 Verify and refine requirements

Iterative

Instantiate architectural elements and allocate functionality

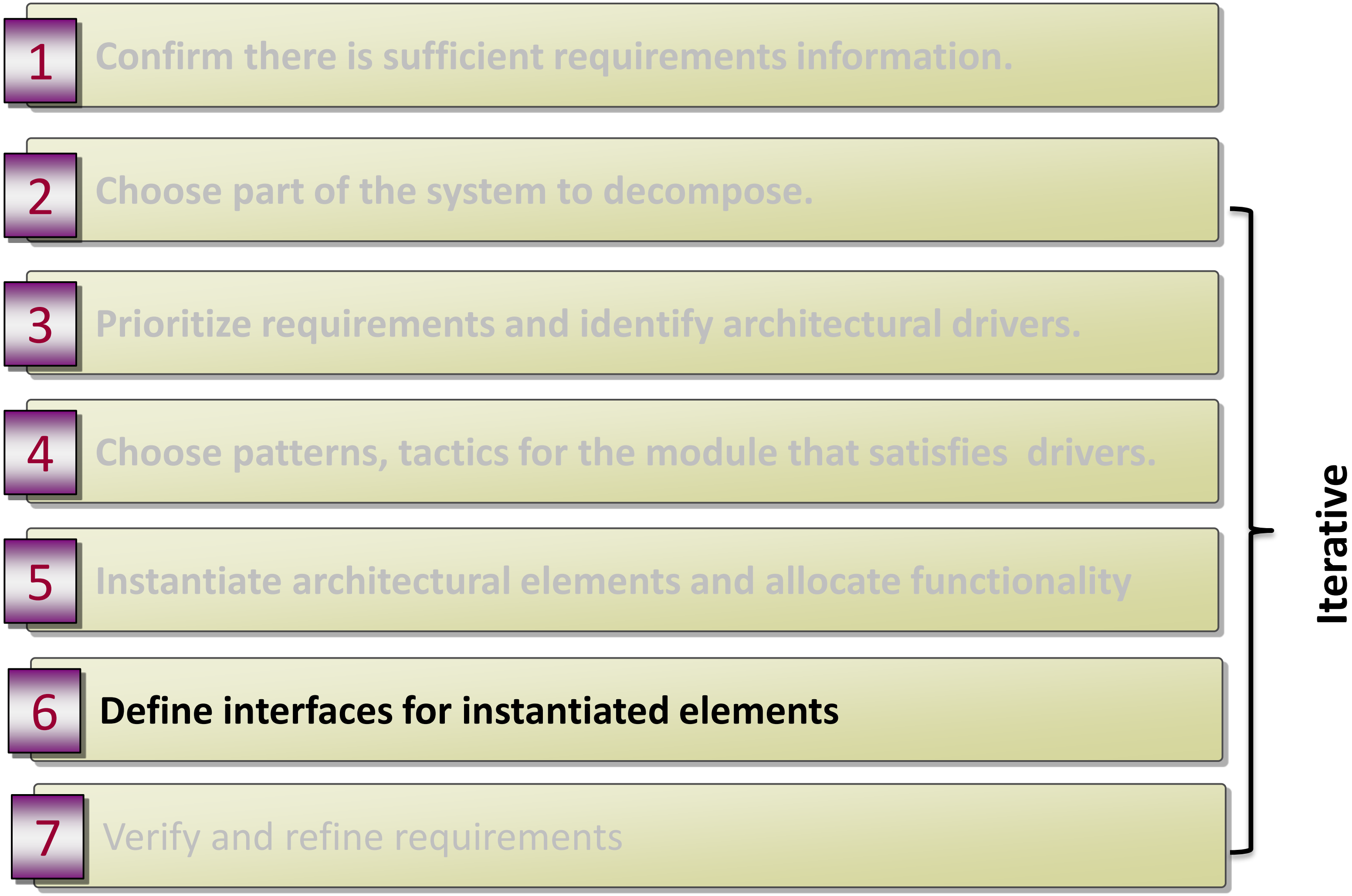
Patterns define the types of elements but not a specific number.

1. A layered pattern doesn’t tell you how many layers
2. A pipe-and-filter pattern doesn’t tell you how many pipes and filters
3. A shared data pattern doesn’t tell you how many data repositories and data accessors

Instantiate architectural elements and allocate functionality

The architect now *applies the chosen patterns to* define a new set of elements that conform to it. Functionality is allocated to the instantiated elements.

Attribute-Driven Design Method



1. Confirm there is sufficient requirements information.
2. Choose part of the system to decompose.
3. Prioritize requirements and identify architectural drivers.
4. Choose patterns, tactics for the module that satisfies drivers.
5. Instantiate architectural elements and allocate functionality
6. **Define interfaces for instantiated elements**
7. Verify and refine requirements

Define interfaces for instantiated elements

The interface for each instantiated element is identified. At this point, interfaces need not be as detailed as a signature, but they document what elements are need, what they can use, and on what they can depend.

Attribute-Driven Design Method

1 Confirm there is sufficient requirements information.

2 Choose part of the system to decompose.

3 Prioritize requirements and identify architectural drivers.

4 Choose patterns, tactics for the module that satisfies drivers.

5 Instantiate architectural elements and allocate functionality

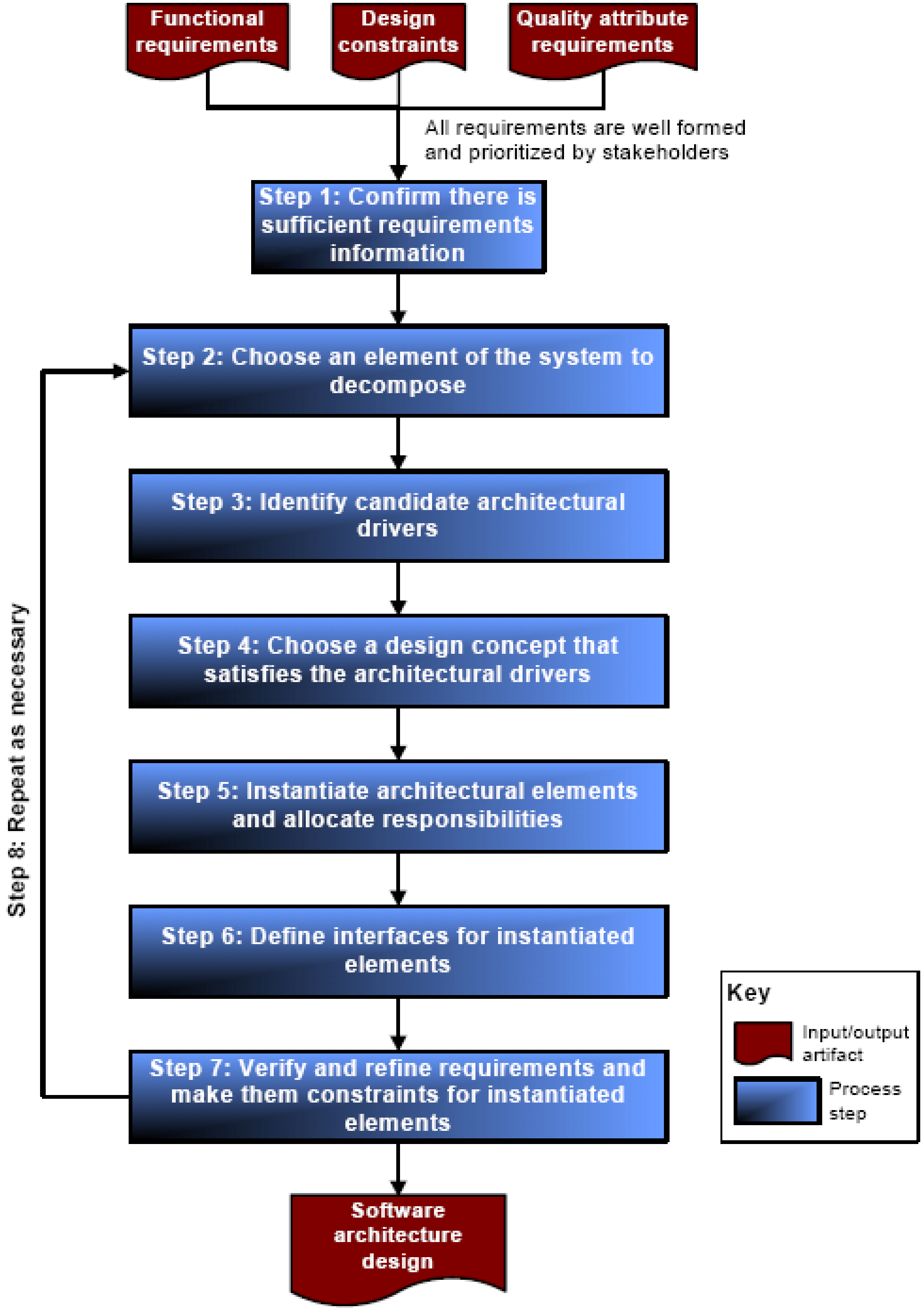
6 Define interfaces for instantiated elements

7 **Verify and refine requirements**

Iterative

Verify and refine

Verify the responsibilities (Functional, Quality, Constraints) of the parent module are decomposed and assigned into the child modules. This step verifies that nothing important was forgotten.



Functional requirements

Design constraints

Quality attribute requirements

All requirements are well formed and prioritized by stakeholders

Step 1: Confirm there is sufficient requirements information

Step 2: Choose an element of the system to decompose

Step 3: Identify candidate architectural drivers

Step 4: Choose a design concept that satisfies the architectural drivers

Step 5: Instantiate architectural elements and allocate responsibilities

Step 6: Define interfaces for instantiated elements

Step 7: Verify and refine requirements and make them constraints for instantiated elements

Step 8: Repeat as necessary

Software architecture design

Key

Input/output artifact

Process step

Appendix

Creating the Architecture

Patterns and tactics represent conceptual tools in the architect’s “tool bag.”

But tools aren’t enough. A method for using the tools is required.

An architect – like a carpenter -- has to know how to use the tools to build something.

Creating the Architecture

ADD is a step-by-step method for systematically producing the first architectural designs for a system. Positioned in the life cycle after requirements analysis.

When using ADD, on the other hand, tactics and a structured set of steps provided design guidance for the creation and nature of each tier. In
this way, each architectural structure is created via an engineering process, rather than simply
being adopted out of habit, intuition, or experience.

Example: Travel Booking System

- Web-based travel booking system, accessible by anyone, but travel agents pay for access and get more functionality
- Provide ability to search for, book, and pay for flights
- Provide ability to search for, book, and pay for hotels
- Security is crucial
- Performance is important
- Must be able to add new airlines and hotels easily

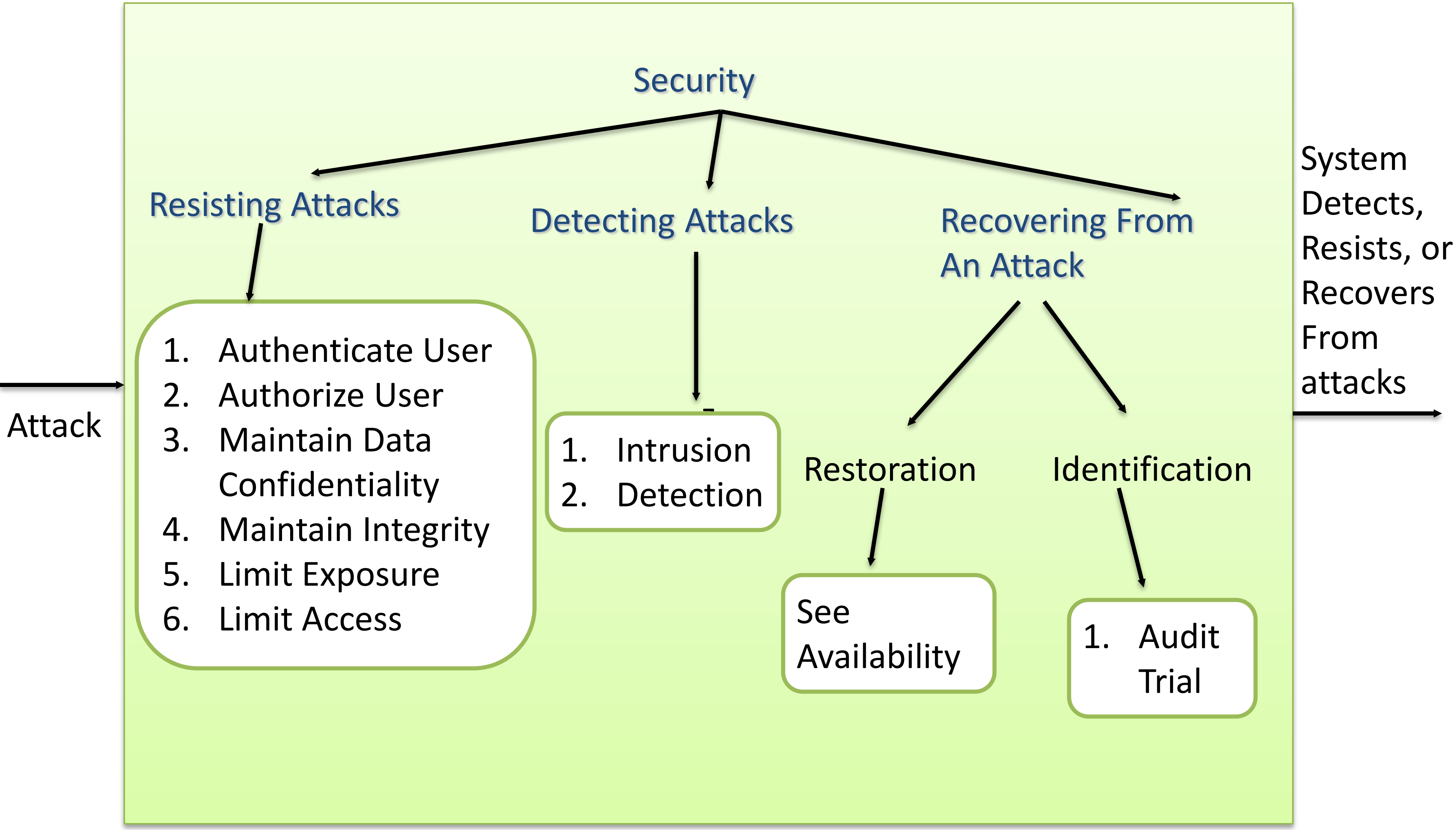
Architectural Drivers

- Resist attacks (communications channels, data stores)
- Allow specialized access to users
- Travel agents must get response within 5 seconds
- Must be possible to add/remove hotels/airlines

Security Quality Attribute Scenario

Source Someone of unknown identity who is external and not authorised with access to limited resources
Stimulus tries to access
Artifact customer data (identity/financial details)
Environment to the system while it is on-line and running normally
Response The response is to block access to the data and record the access attempts
Response Measure with 100% probability of detecting the attack, 100% probability of denying access, and 50% probability of identifying the location of the individual.

Security Tactics Hierarchy



Attack

Security

Resisting Attacks

1. Authenticate User
2. Authorize User
3. Maintain Data Confidentiality
4. Maintain Integrity
5. Limit Exposure
6. Limit Access

Detecting Attacks

1. Intrusion
2. Detection

Recovering From An Attack

Restoration

See Availability

Identification

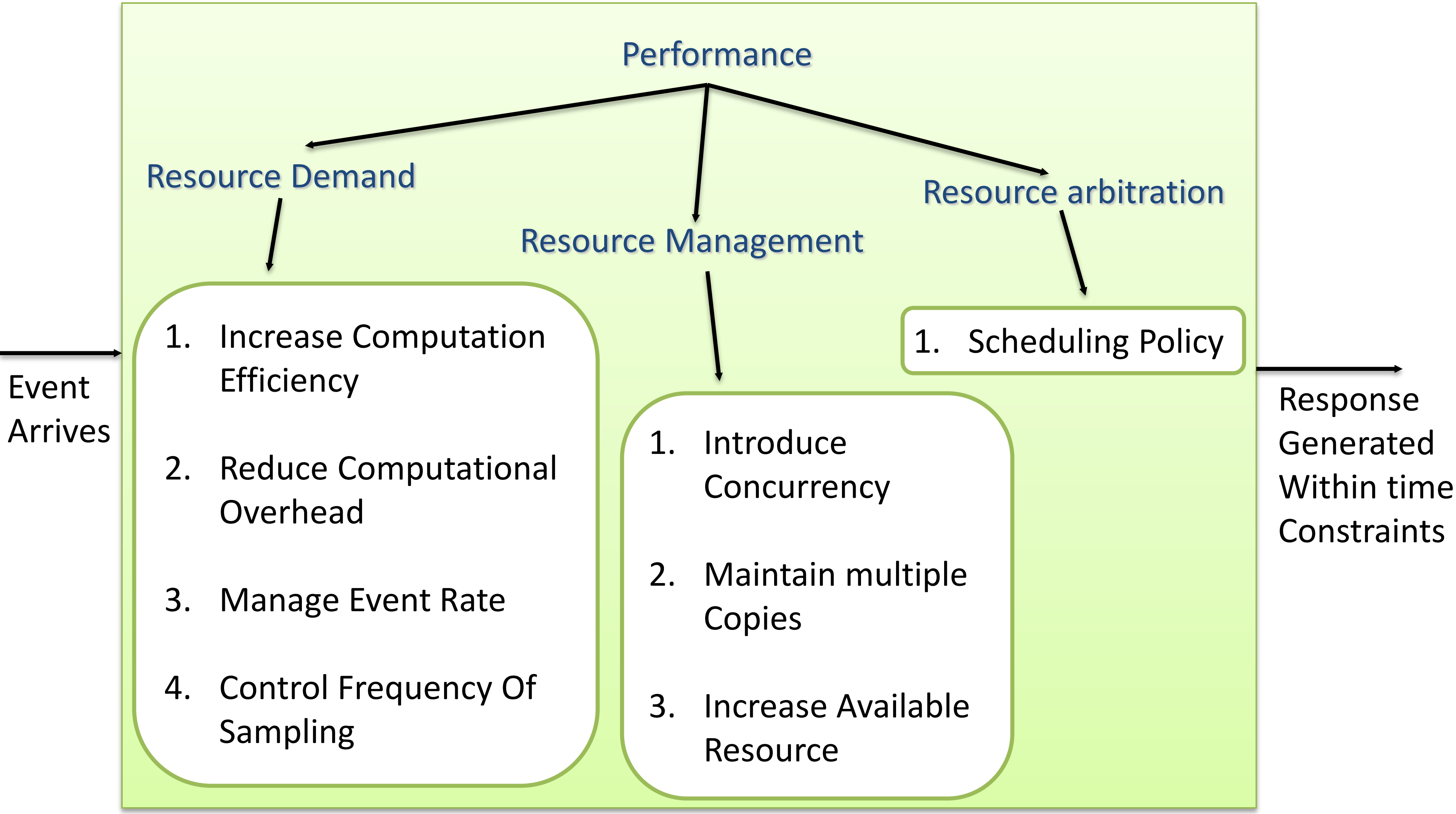
1. Audit Trial

System Detects, Resists, or Recovers From attacks

Security Tactics

- limit access—firewall
- maintain data confidentiality —encryption for communication (performance issue)
- authenticate users—must distinguish travel agents from others
- authorise users—travel agents have different access than others

Performance Tactics Hierarchy



Event Arrives

Performance

Resource Demand

1. Increase Computation Efficiency
2. Reduce Computational Overhead
3. Manage Event Rate
4. Control Frequency Of Sampling

Resource Management

1. Introduce Concurrency
2. Maintain multiple Copies
3. Increase Available Resource

Resource arbitration

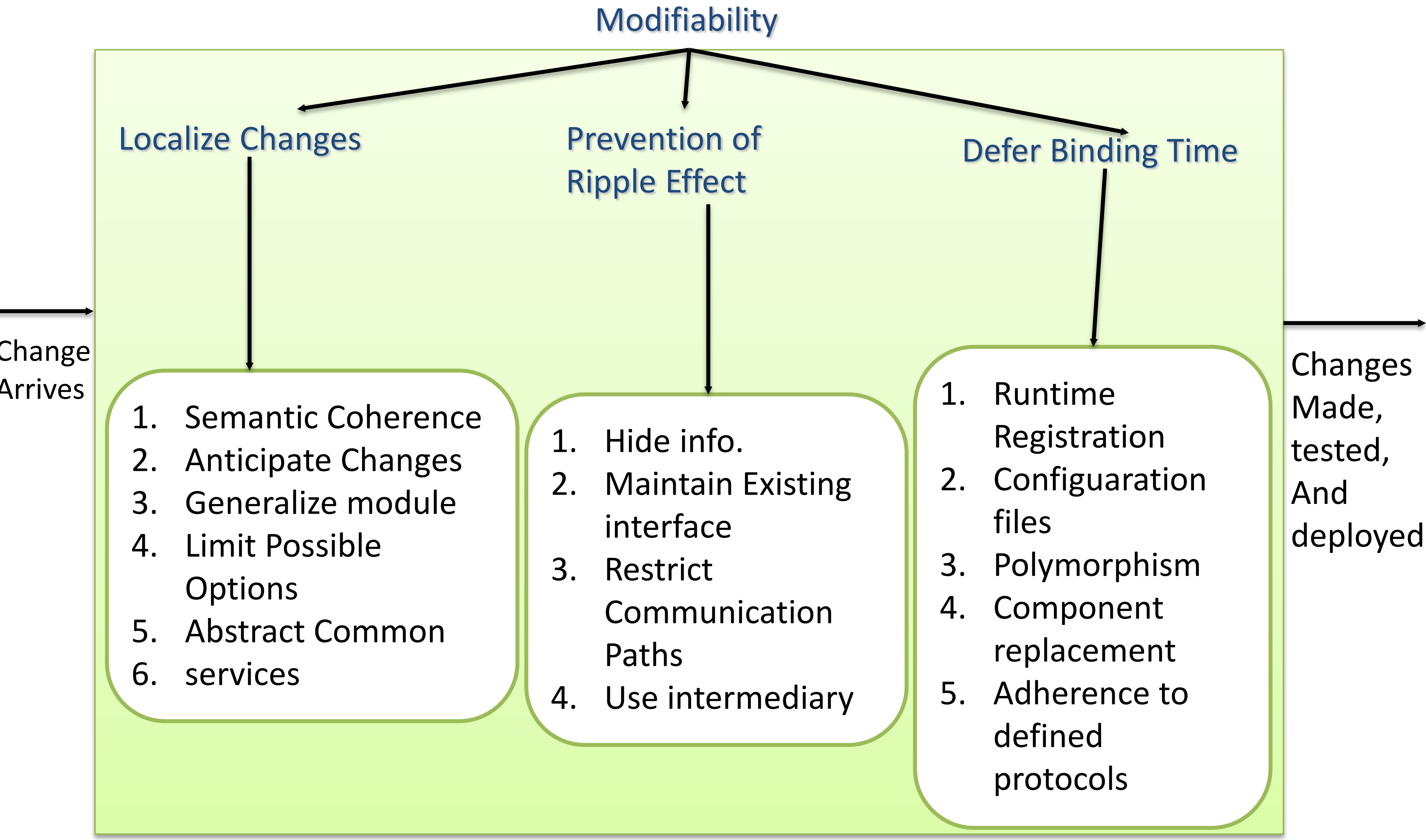
1. Scheduling Policy

Response Generated Within time Constraints

Performance

- *communicating processes pattern*
- introduce concurrency— process requests in parallel
- increase available resources— more processors
- maintain multiple copies —information that doesn’t change much
- scheduling policy —travel agents have priority

Modifiability Tactics Hierarchy



Change Arrives

Modifiability

Localize Changes

1. Semantic Coherence
2. Anticipate Changes
3. Generalize module
4. Limit Possible Options
5. Abstract Common
6. services

Prevention of Ripple Effect

1. Hide info.
2. Maintain Existing interface
3. Restrict Communication Paths
4. Use intermediary

Defer Binding Time

1. Runtime Registration
2. Configuaration files
3. Polymorphism
4. Component replacement
5. Adherence to defined protocols

Changes Made, tested, And deployed

Modifiability

- n-*tier pattern*
- *blackboard pattern —place to record what airlines and hotels* are participating at any time
- semantic coherence—airlines and hotels (at least)
- information hiding —exactly which airlines and which hotels (and which travel agents)
- runtime registration, adherence to defined protocols —central registry to look up airlines, hotels

Architecture 🚫 is structure

Architecture does involve structure, decomposition, and interfaces. Architecture also involves behavior.