Foundations of Software Engineering

Lecture 8: Architectural Patterns, Tactics, and Evaluation

Claire Le Goues

institute for SOFTWARE RESEARCH

Learning Goals

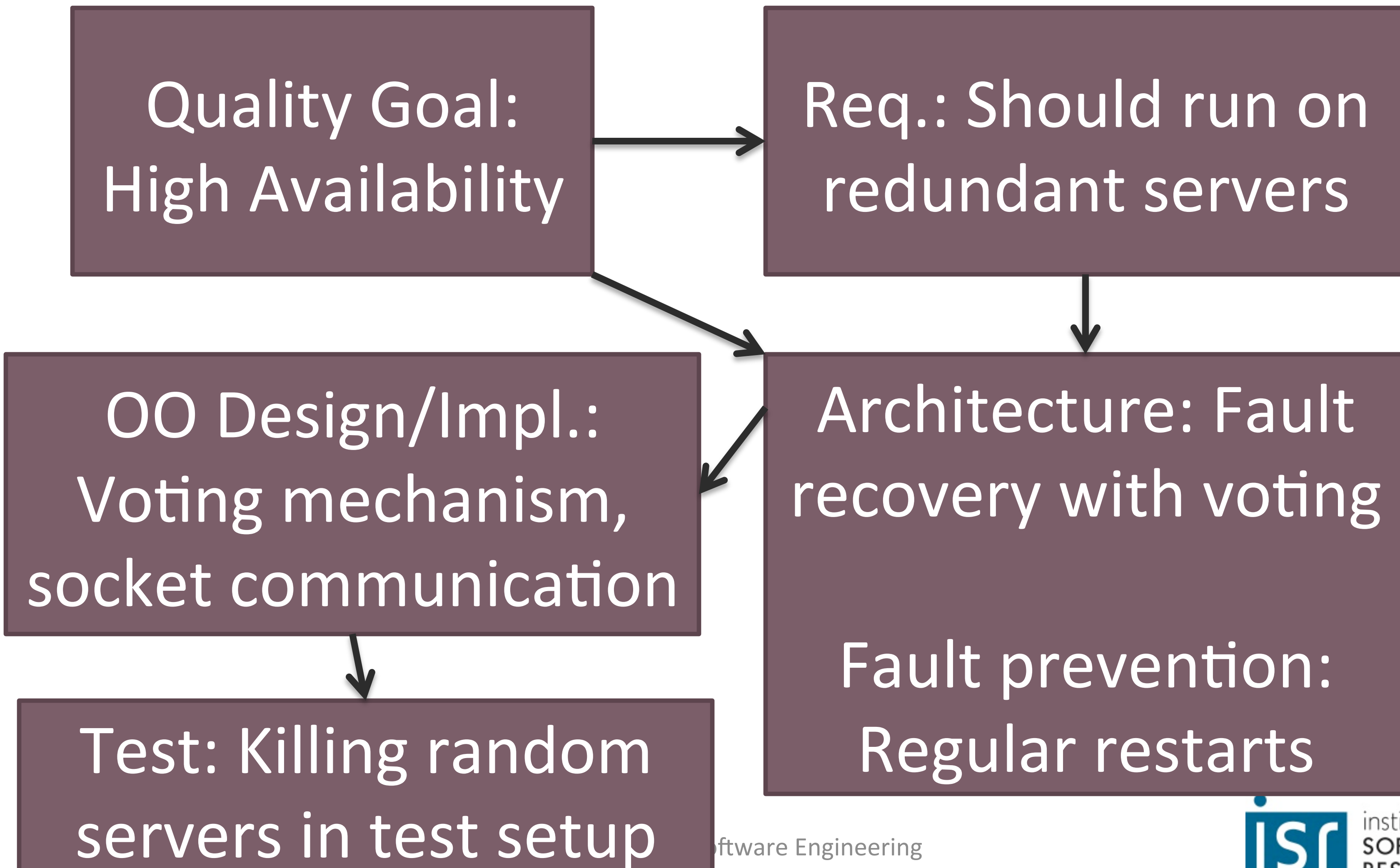
- Understand key parts of architectural process
- Use architectural styles and tactics for design decisions
- Make justified architectural decisions for new systems and within existing systems
- Review a proposed architecture

Traceability - Definition

"The ability to interrelate any uniquely identifiable software engineering artifact to any other, maintain required links over time, and use the resulting network to answer questions of both the software product and it's development process" – CoEST

Traceability in Requirements?

Traceability



Traceability Compliance

- Traceability required in some domains (avionics)
  - Why does X piece of code exist?
- "Enable verification of the absence of undocumented source code and verification of the complete implementation of the low-level requirements"
- Link to specifications and test procedures

Traceability and Architecture

- Architecture links quality attributes to the high-level and low-level system design
- Ensures quality attributes often not even visible in code
- Cost, effort, discipline needed to create and maintain.
  - Often incomplete, incorrect, outdated
- Developers hate it, and often do not understand the need.
  - "Unnecessary evil"

Case Study: The Google File System

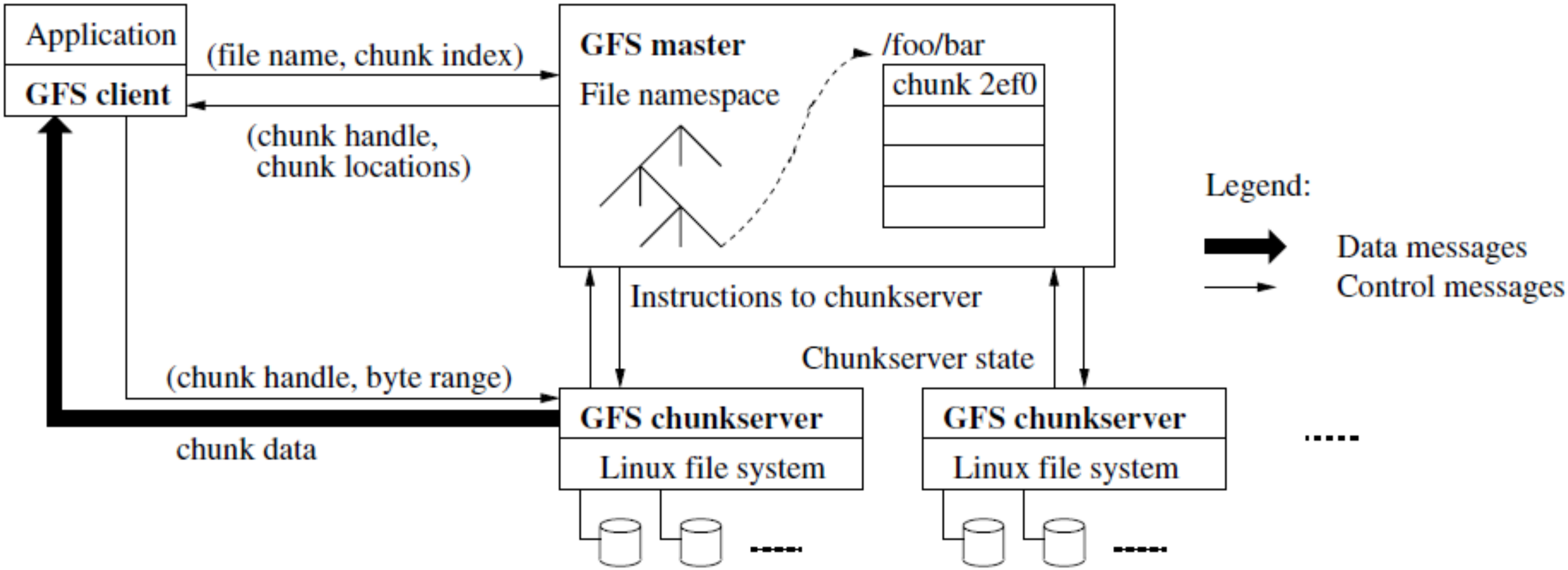


Figure 1: GFS Architecture

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." *ACM SIGOPS operating systems review*. Vol. 37. No. 5. ACM, 2003.

Assumptions

- The system is built from many inexpensive commodity components that often fail.
- The system stores a modest number of large files.
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads.
- The workloads also have many large, sequential writes that append data to files.
- The system must efficiently implement well-defined semantics for multiple clients that concurrently append to the same file.
- High sustained bandwidth is more important than low latency.

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." *ACM SIGOPS operating systems review*. Vol. 37. No. 5. ACM, 2003.

Qualities:
Scalability
Reliability
Performance
Cost

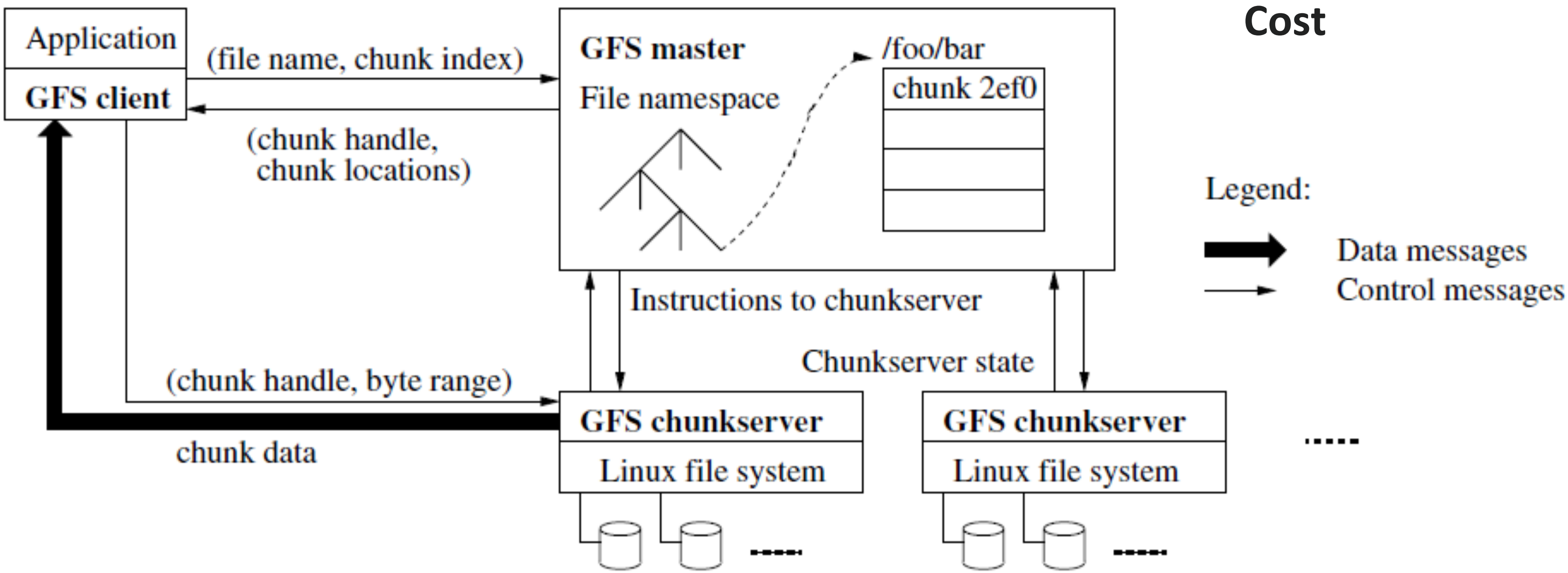


Figure 1: GFS Architecture

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." *ACM SIGOPS operating systems review*. Vol. 37. No. 5. ACM, 2003.

Questions

1. What are the most important quality attributes in the design?
2. How are those quality attributes realized in the design?

Qualities:
Scalability
Reliability
Performance
Cost

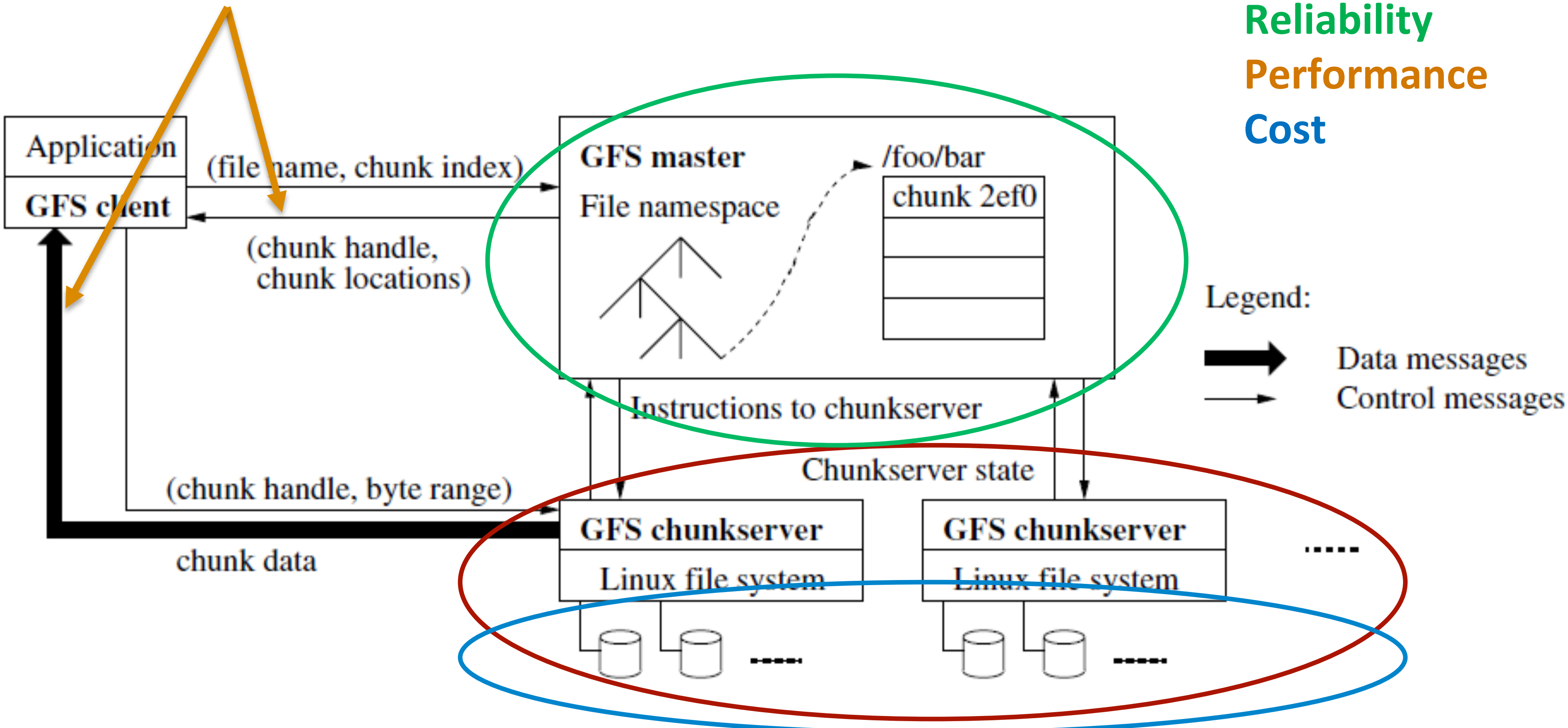


Figure 1: GFS Architecture

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." *ACM SIGOPS operating systems review*. Vol. 37. No. 5. ACM, 2003.

Exercise

For the Google File System, create a physical architecture view that addresses a relevant quality attribute

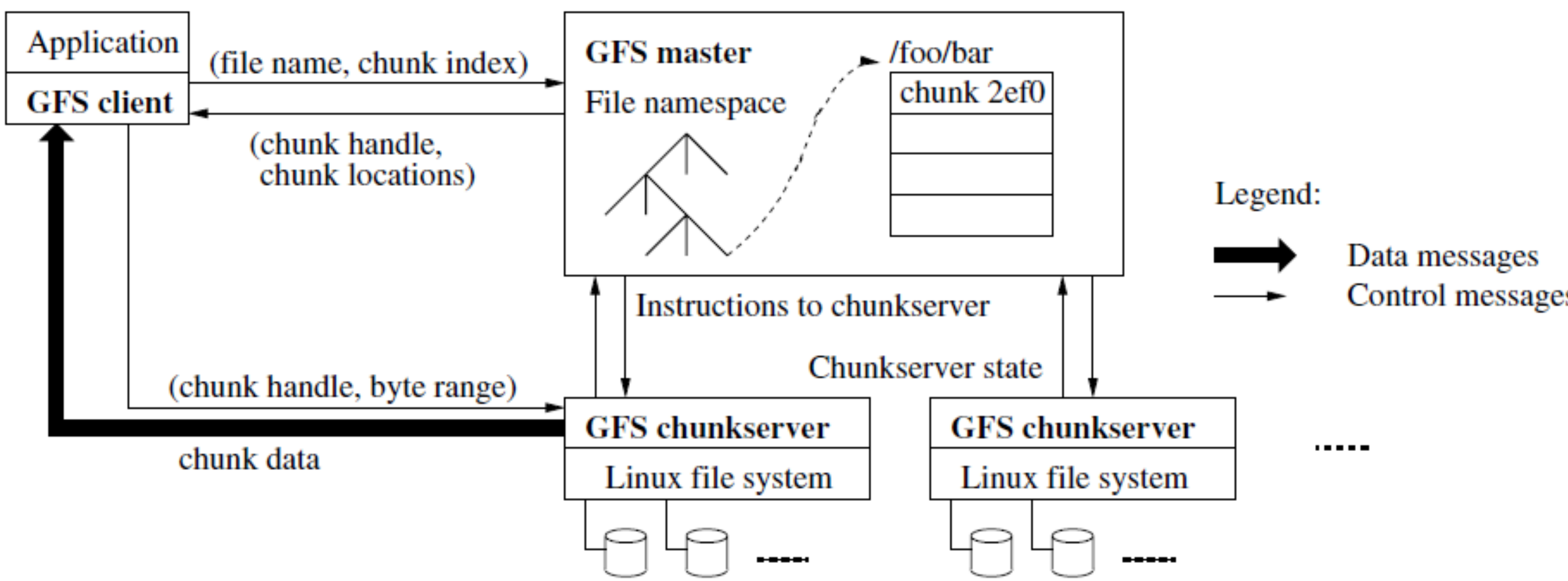
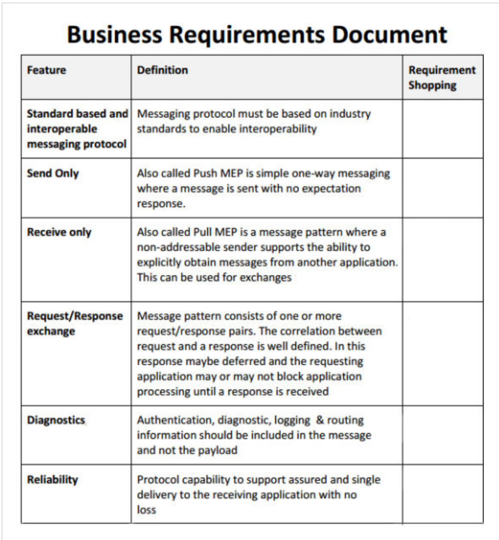


Figure 1: GFS Architecture

So far in course



Business Requirements Document

Feature	Definition	Requirement Shopping
Standard based and interoperable messaging protocol	Messaging protocol must be based on industry standards to enable interoperability	
Send Only	Also called Push MEP is simple one-way messaging where a message is sent with no expectation response.	
Receive only	Also called Pull MEP is a message pattern where a non-addressable sender supports the ability to explicitly obtain messages from another application. This can be used for exchanges	
Request/Response exchange	Message pattern consists of one or more request/response pairs. The correlation between request and a response is well defined. In this response maybe deferred and the requesting application may or may not block application processing until a response is received	
Diagnostics	Authentication, diagnostic, logging & routing information should be included in the message and not the payload	
Reliability	Protocol capability to support assured and single delivery to the receiving application with no loss	

Requirements

Implementation

Architecture

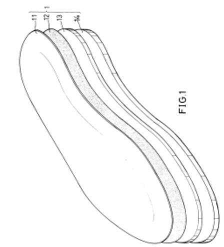
Levels of abstraction

- Requirements
  - high-level “what” needs to be done
- Architecture (High-level design)
  - high-level “how”, mid-level “what”
- OO-Design (Low-level design, e.g. design patterns)
  - mid-level “how”, low-level “what”
- Code
  - low-level “how”

What is architecture?

Architecture as structures and relations
(the actual system)

Architecture as documentation
(representations of the system)



Architecture as process
(activities around the other two)

Architectural Styles and Tactics

Architectural style (pattern)

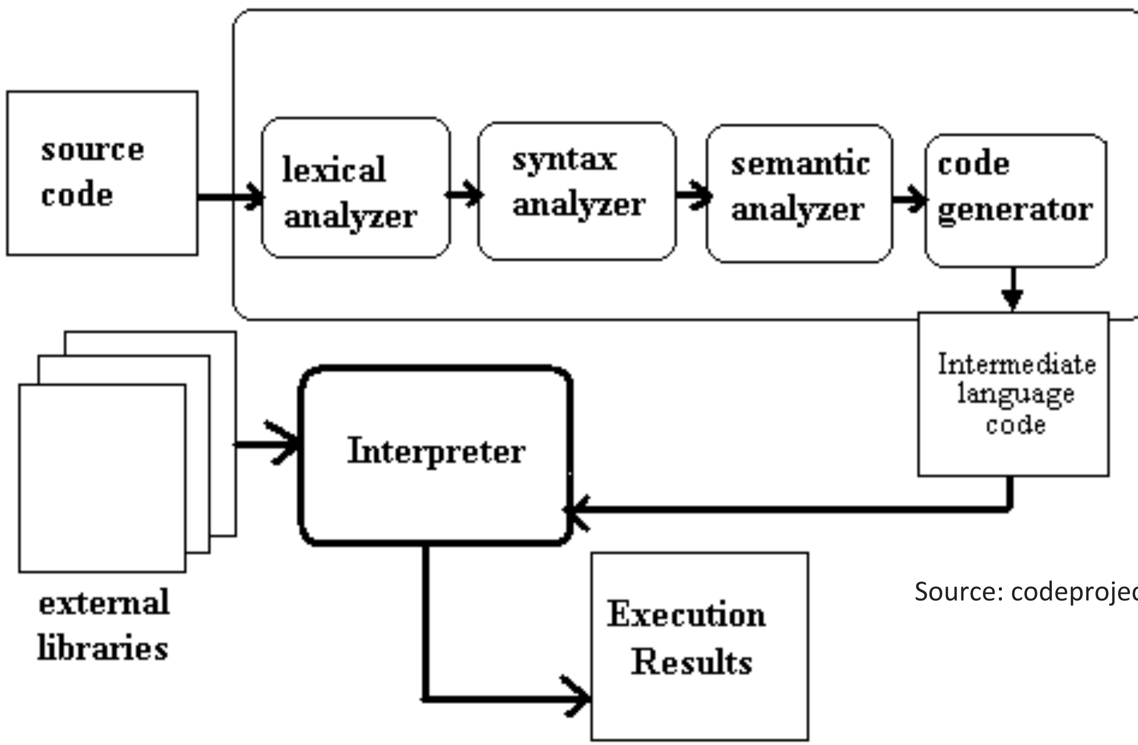
- Broad principle of system organization
- Describes computational model
  - E.g., pipe and filter, call-return, publish-subscribe, layered, services
- Related to one of common view types
  - Static, dynamic, physical

Example Architectural Patterns

- System organization
  - Repository model
  - Client-server model
  - Layered model
- Modular decomposition
  - Object oriented
  - Function-oriented pipelining
- Control styles
  - Centralized control
  - Event-driven systems

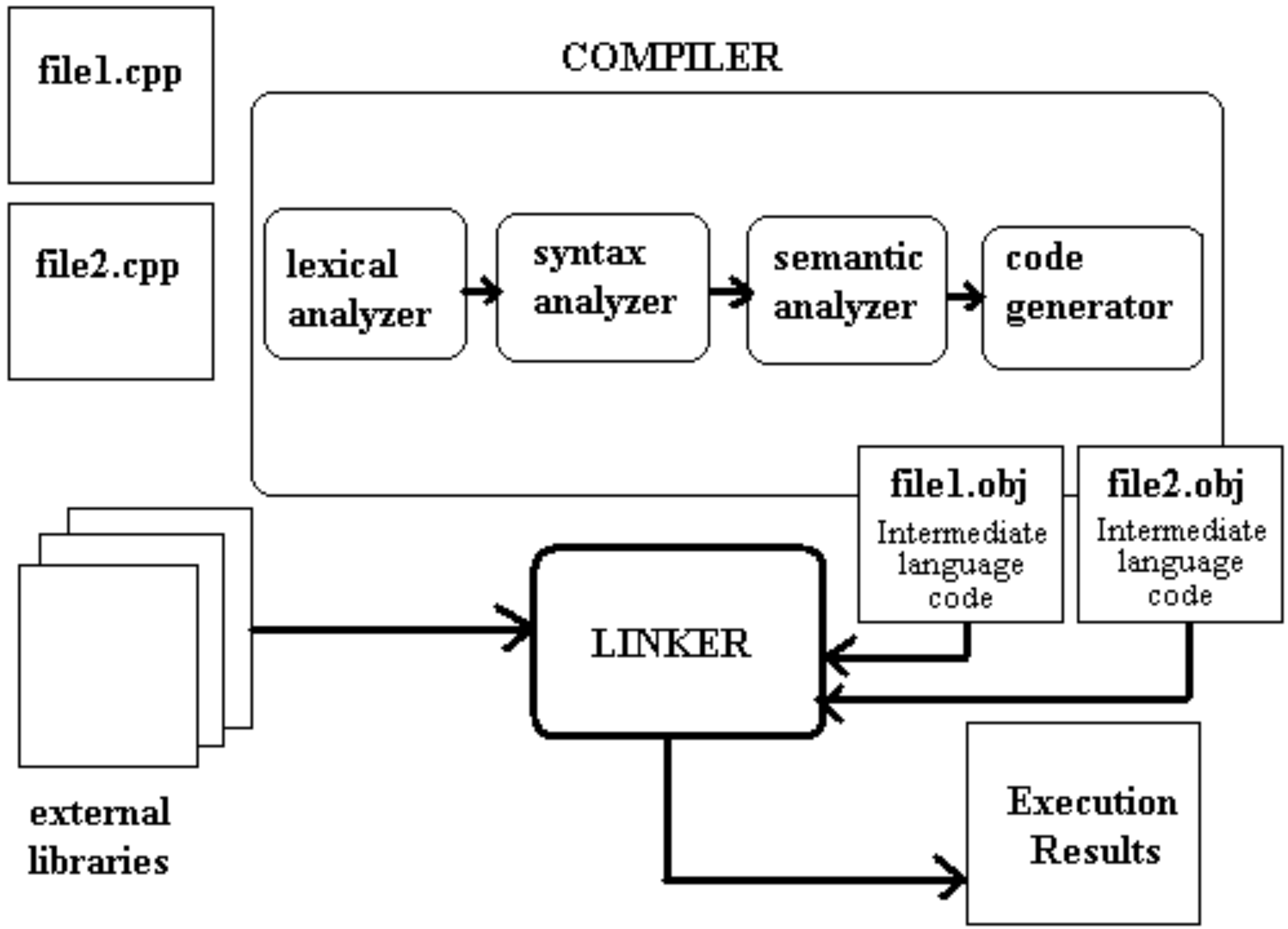
Architectural style (pattern)

- Broad principle of system organization
- See reading



Source: codeproject.org

Architectural style (pattern)



Source: codeproject.org

Client-server pattern

- Separation of clients and servers
  - Servers provide services; known and "stable"
  - Clients request services; come and go
- Varieties: synchronous/asynchronous
- Impact on security, performance, scalability
- Examples: TCP, HTTP, X11

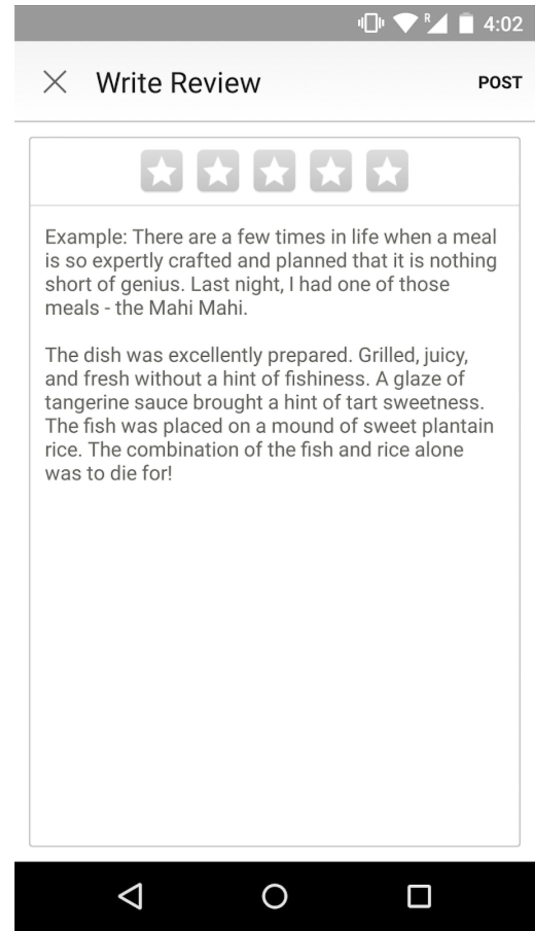
Client

Server

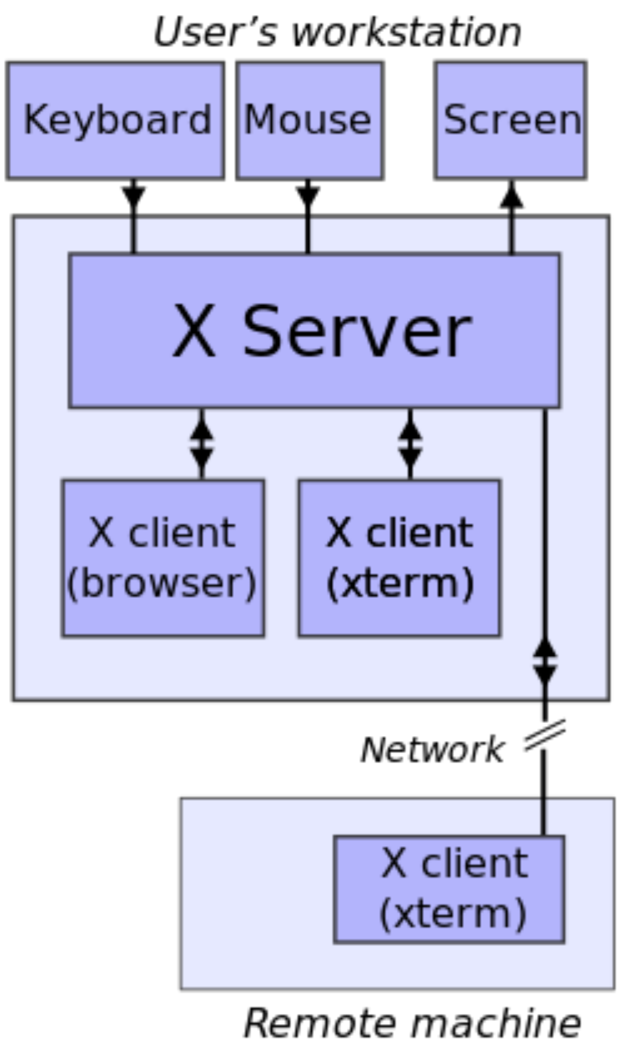
Database

Where to validate user input?

Example: Yelp App

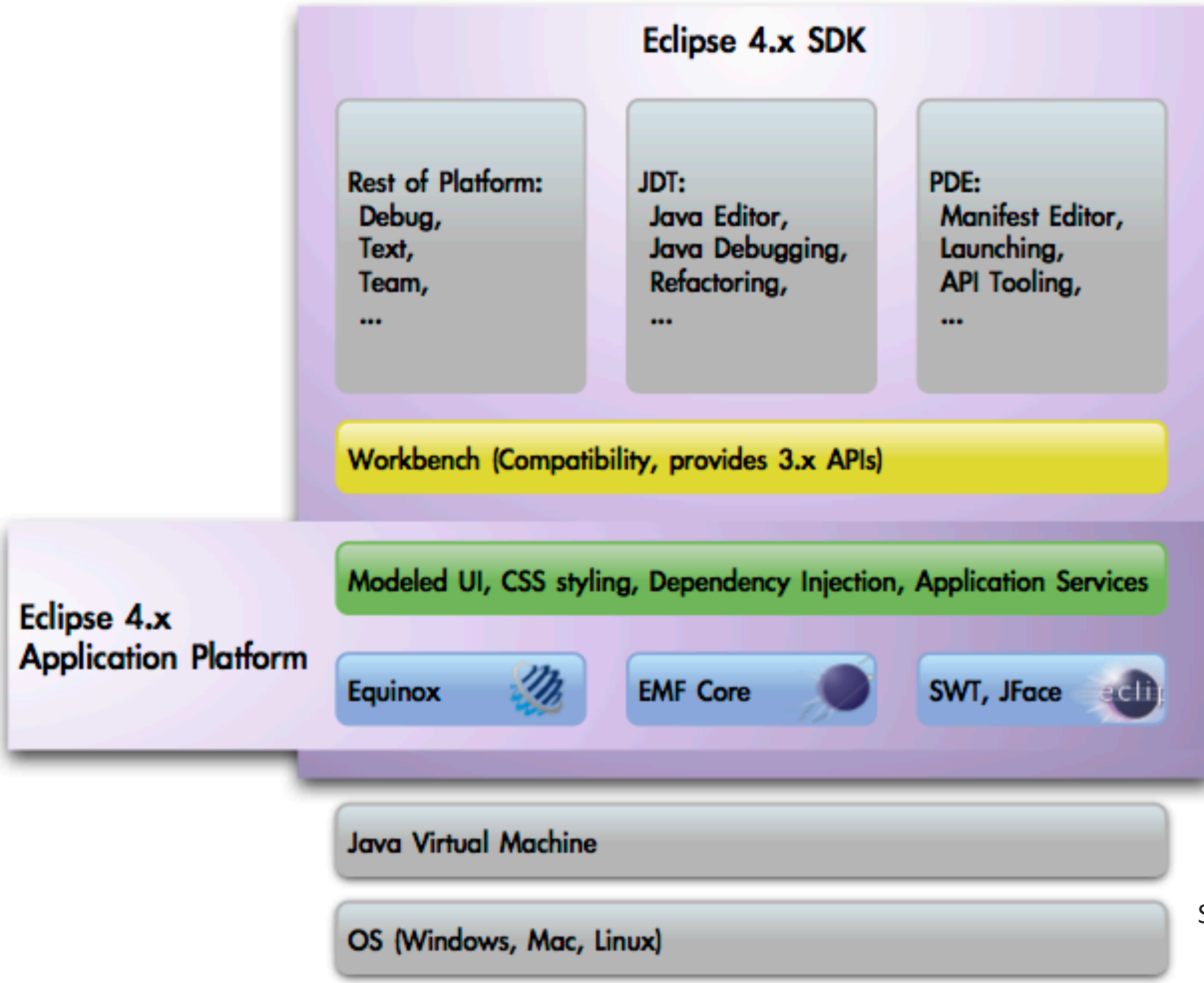


Client-server style



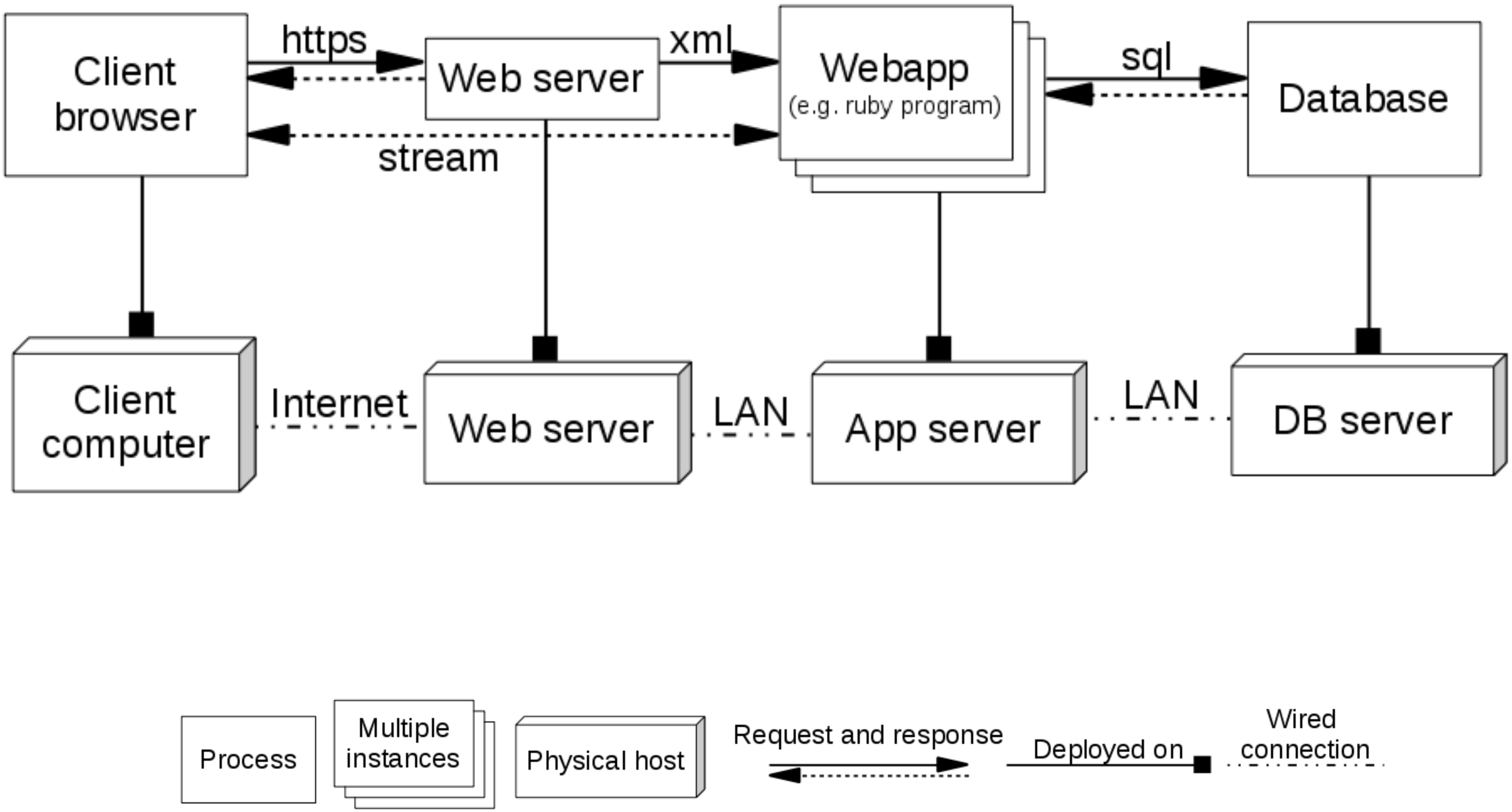
Source: wikimedia commons

Layered system



Source: eclipse.org

Tiered architecture



institute for SOFTWARE RESEARCH

Architectural Style?

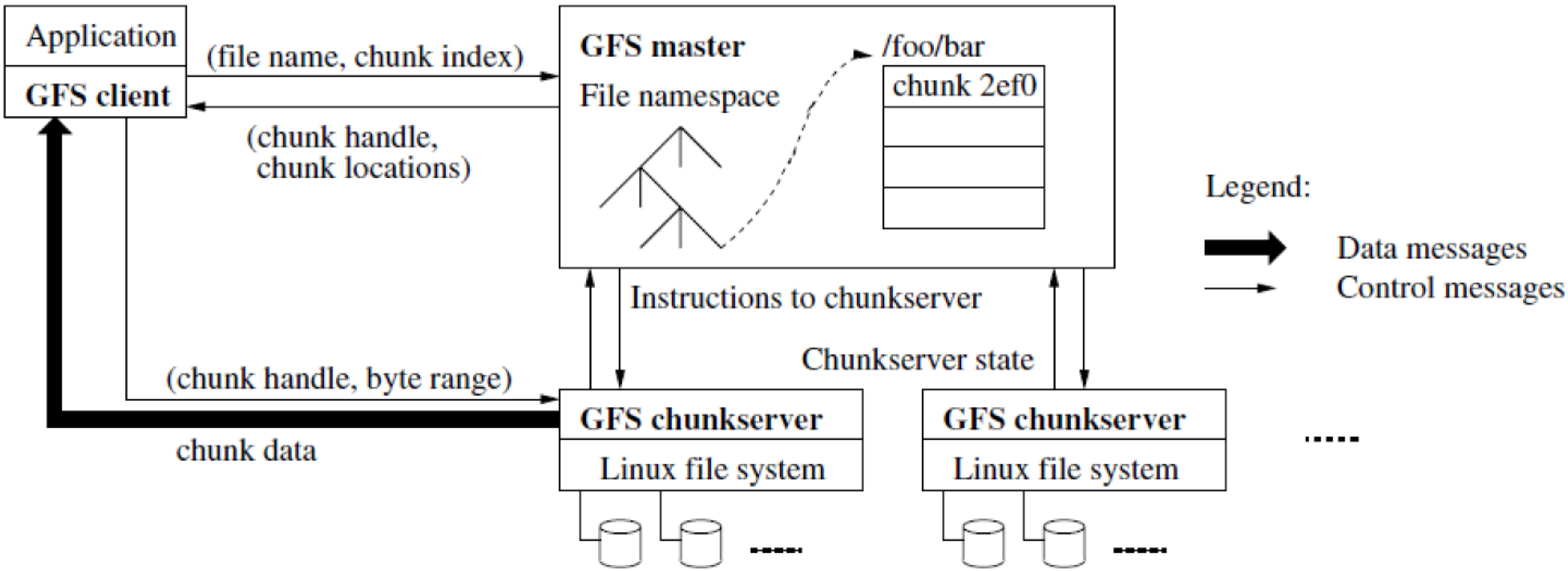


Figure 1: GFS Architecture

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." *ACM SIGOPS operating systems review*. Vol. 37. No. 5. ACM, 2003.

Tactics

- Architectural techniques to achieve qualities
  - More tied to specific context and quality
- Smaller scope than architectural patterns
  - Problem solved by patterns: "How do I structure my (sub)system?"
  - Problem solved by tactics: "How do I get better at quality X?"
- Collection of common strategies and known solutions
  - Resemble OO design patterns

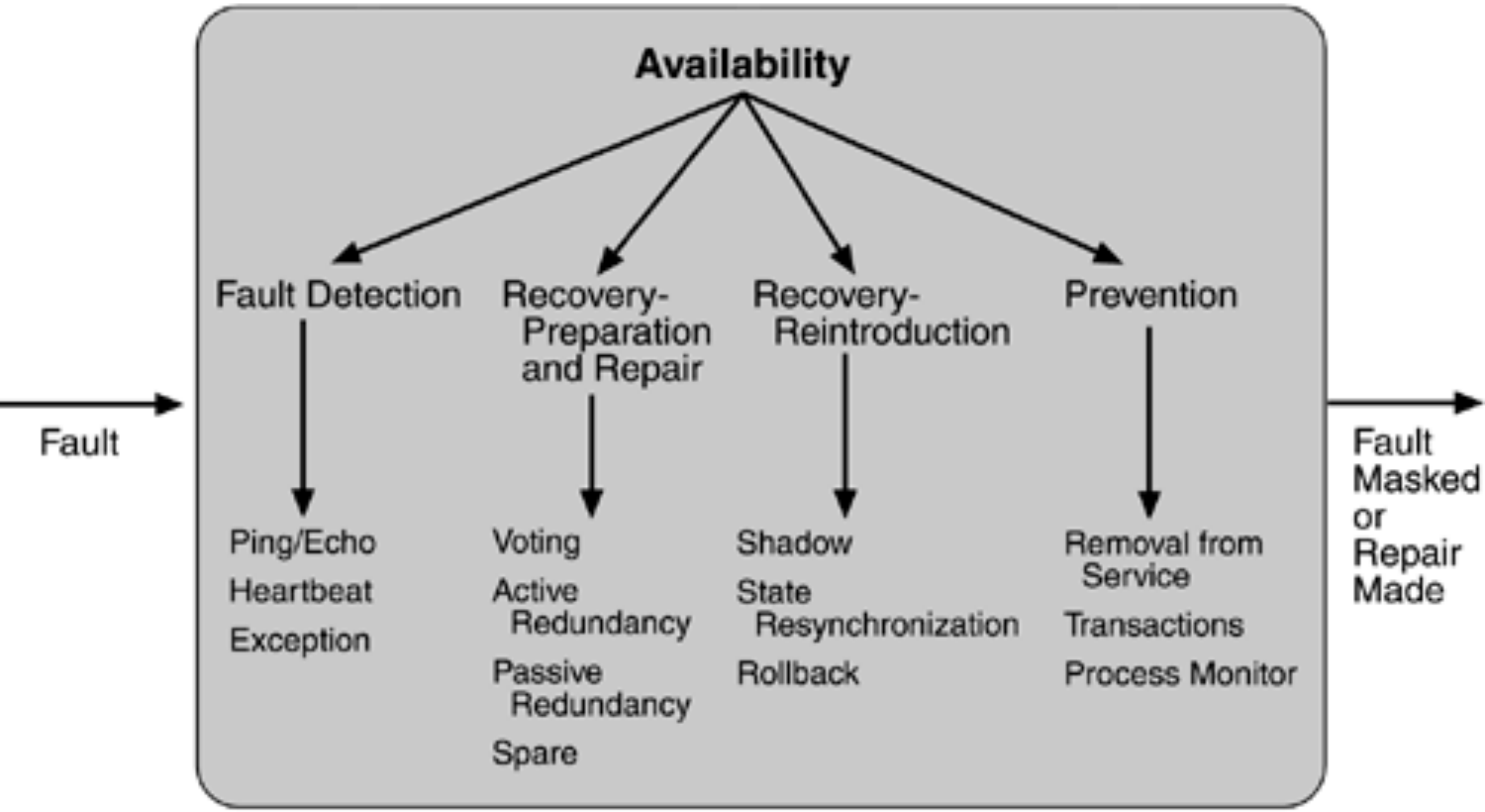
Prioritization
Concurrency
Nondeterminism
Encryption
Separation
Undo
Authentication
Timestamps
Sanitation
Redundancy
Authorization
Synchronization
Voting
Cohesion
Transactions
Sandboxing
Coupling
Auditing

Example Tactic Description: Record/playback

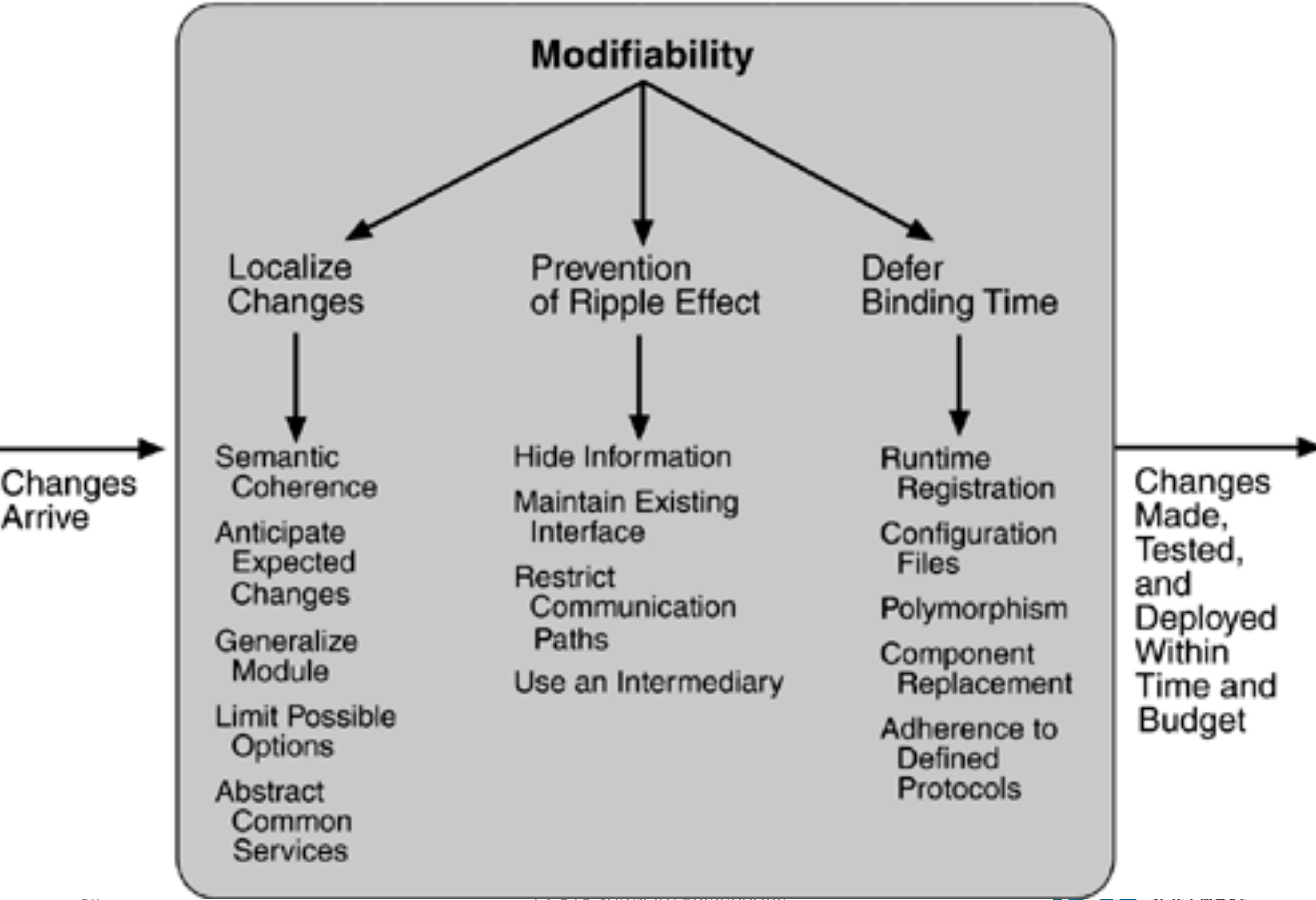
- Record/playback refers to both capturing information crossing an interface and using it as input into the test harness. The information crossing an interface during normal operation is saved in some repository and represents output from one component and input to another. Recording this information allows test input for one of the components to be generated and test output for later comparison to be saved.

Example Tactic Description: Built-in monitors

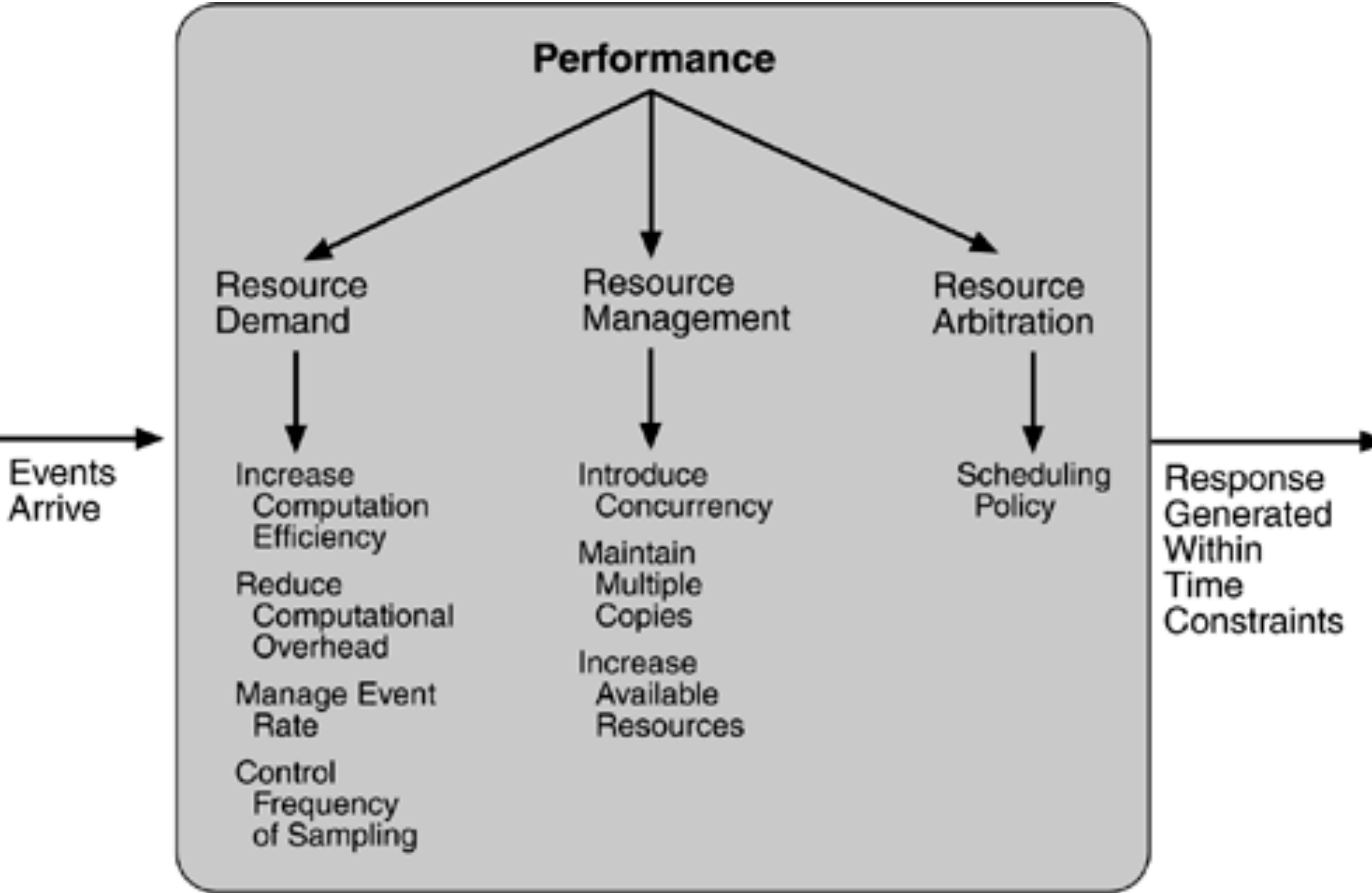
- The component can maintain state, performance load, capacity, security, or other information accessible through an interface. This interface can be a permanent interface of the component or it can be introduced temporarily via an instrumentation technique such as aspect-oriented programming or preprocessor macros. A common technique is to record events when monitoring states have been activated. Monitoring states can actually increase the testing effort since tests may have to be repeated with the monitoring turned off. Increased visibility into the activities of the component usually more than outweigh the cost of the additional testing.

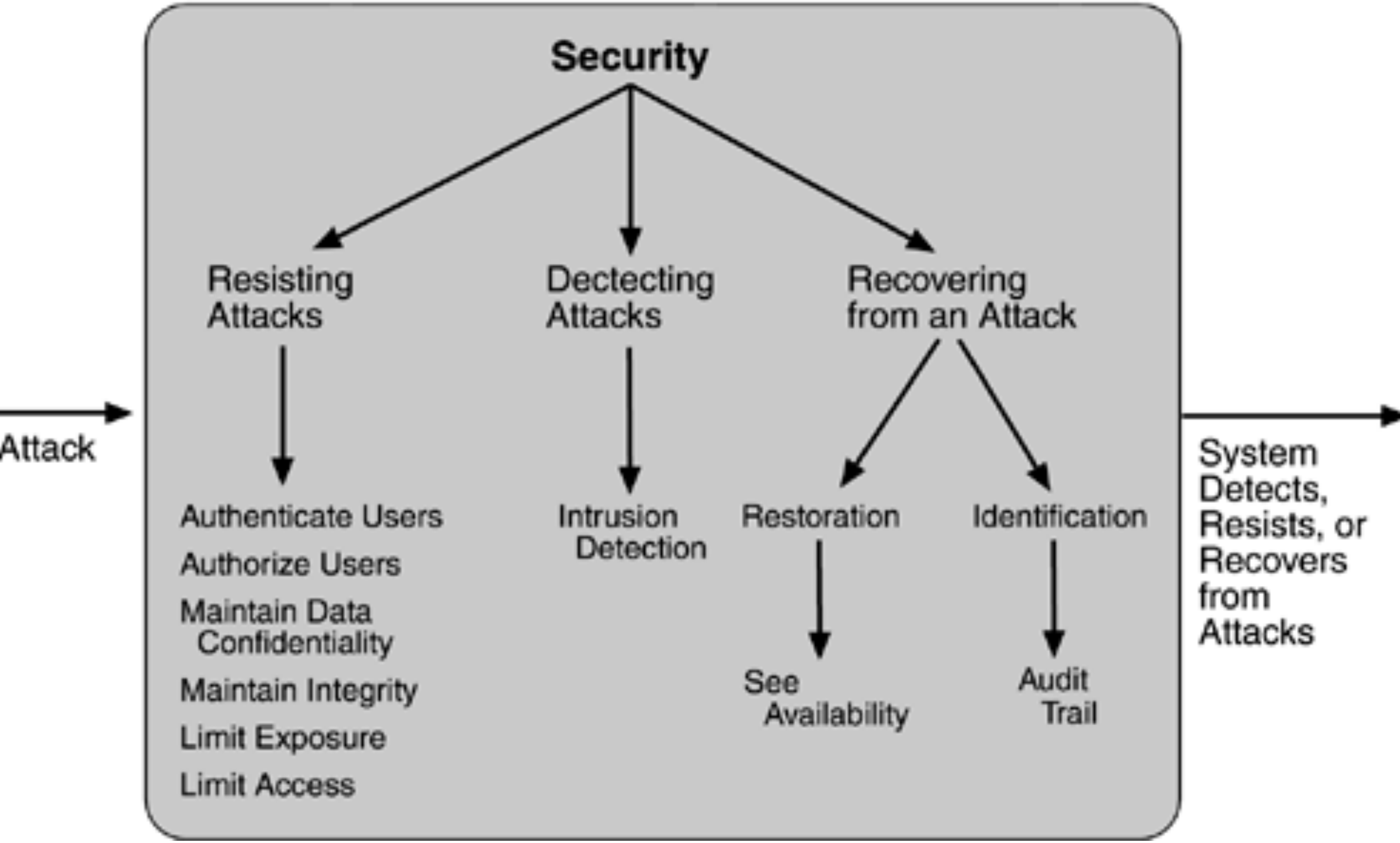


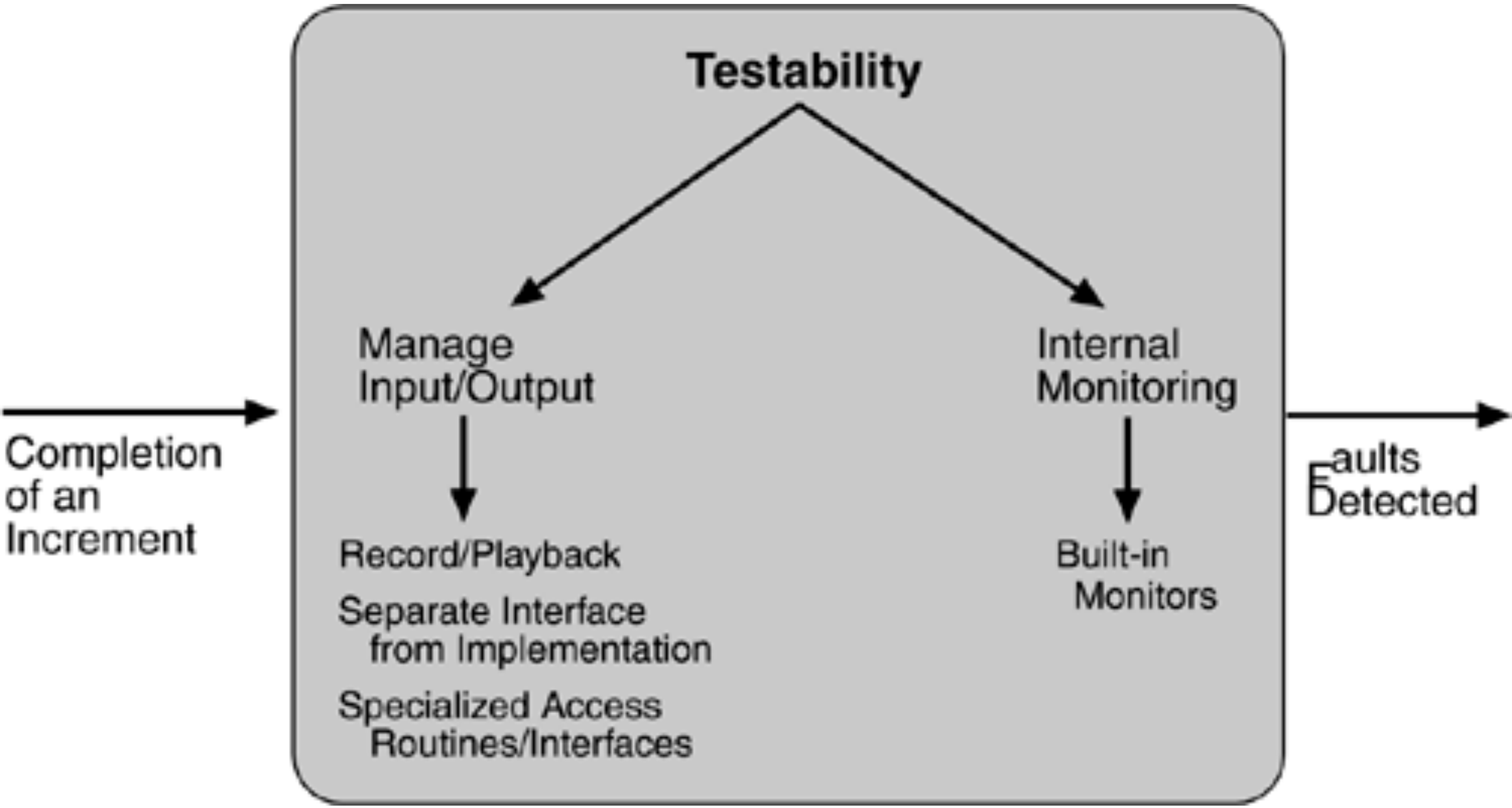
Availability
Fault Detection
Recovery-Preparation and Repair
Recovery-Reintroduction
Prevention
Fault
Ping/Echo
Heartbeat
Exception
Voting
Active Redundancy
Passive Redundancy
Spare
Shadow
State Resynchronization
Rollback
Removal from Service
Transactions
Process Monitor
Fault Masked or Repair Made

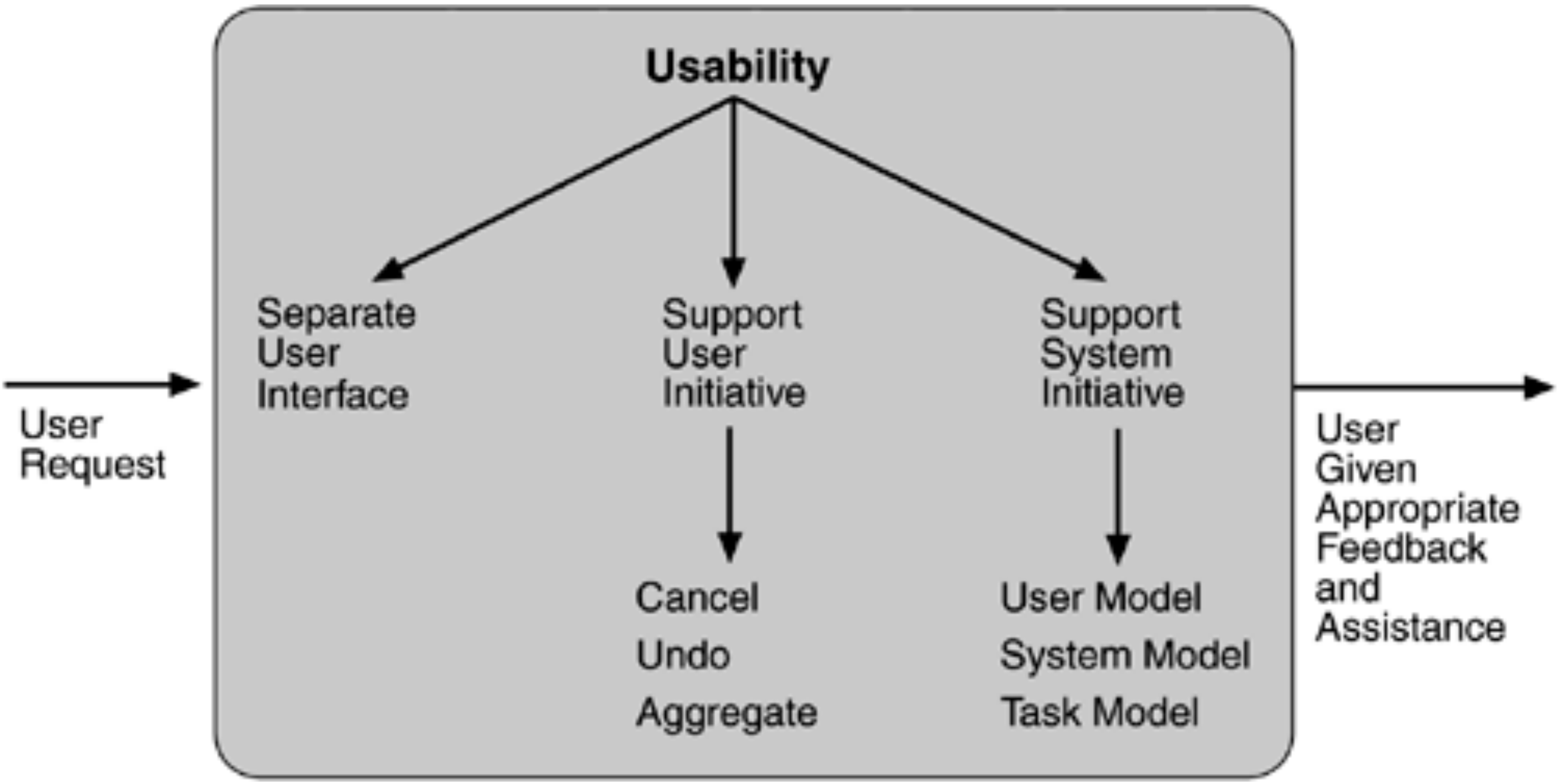


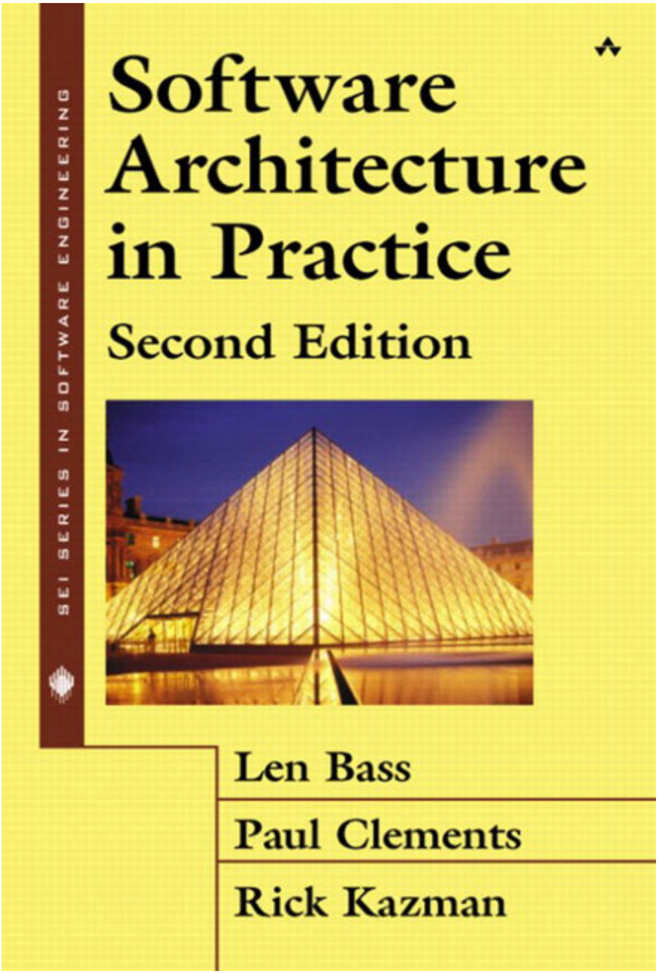
Modifiability
Localize Changes
Prevention of Ripple Effect
Defer Binding Time
Changes Arrive
Semantic Coherence
Anticipate Expected Changes
Generalize Module
Limit Possible Options
Abstract Common Services
Hide Information
Maintain Existing Interface
Restrict Communication Paths
Use an Intermediary
Runtime Registration
Configuration Files
Polymorphism
Component Replacement
Adherence to Defined Protocols
Changes Made, Tested, and Deployed Within Time and Budget


Performance
Resource Demand
Resource Management
Resource Arbitration
Events Arrive
Increase Computation Efficiency
Reduce Computational Overhead
Manage Event Rate
Control Frequency of Sampling
Introduce Concurrency
Maintain Multiple Copies
Increase Available Resources
Scheduling Policy
Response Generated Within Time Constraints


Security
Attack
Resisting Attacks
Dectecting Attacks
Recovering from an Attack
Authenticate Users
Authorize Users
Maintain Data Confidentiality
Maintain Integrity
Limit Exposure
Limit Access
Intrusion Detection
Restoration
Identification
See Availability
Audit Trail
System Detects, Resists, or Recovers from Attacks


Testability
Manage Input/Output
Internal Monitoring
Completion of an Increment
Faults Detected
Record/Playback
Separate Interface from Implementation
Specialized Access Routines/Interfaces
Built-in Monitors


Usability
Separate User Interface
Support User Initiative
Support System Initiative
User Request
Cancel
Undo
Aggregate
User Model
System Model
Task Model
User Given Appropriate Feedback and Assistance



Many tactics described in Chapter 5

Brief high-level descriptions (about 1 paragraph per tactic)

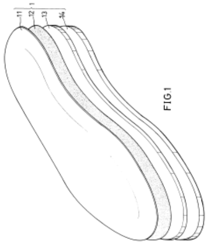
Second and more detailed third edition available as ebook through CMU library.

Architecture Design Process

What is architecture?

Architecture as
structures and relations
(the actual system)

Architecture as
documentation
(representations of the system)



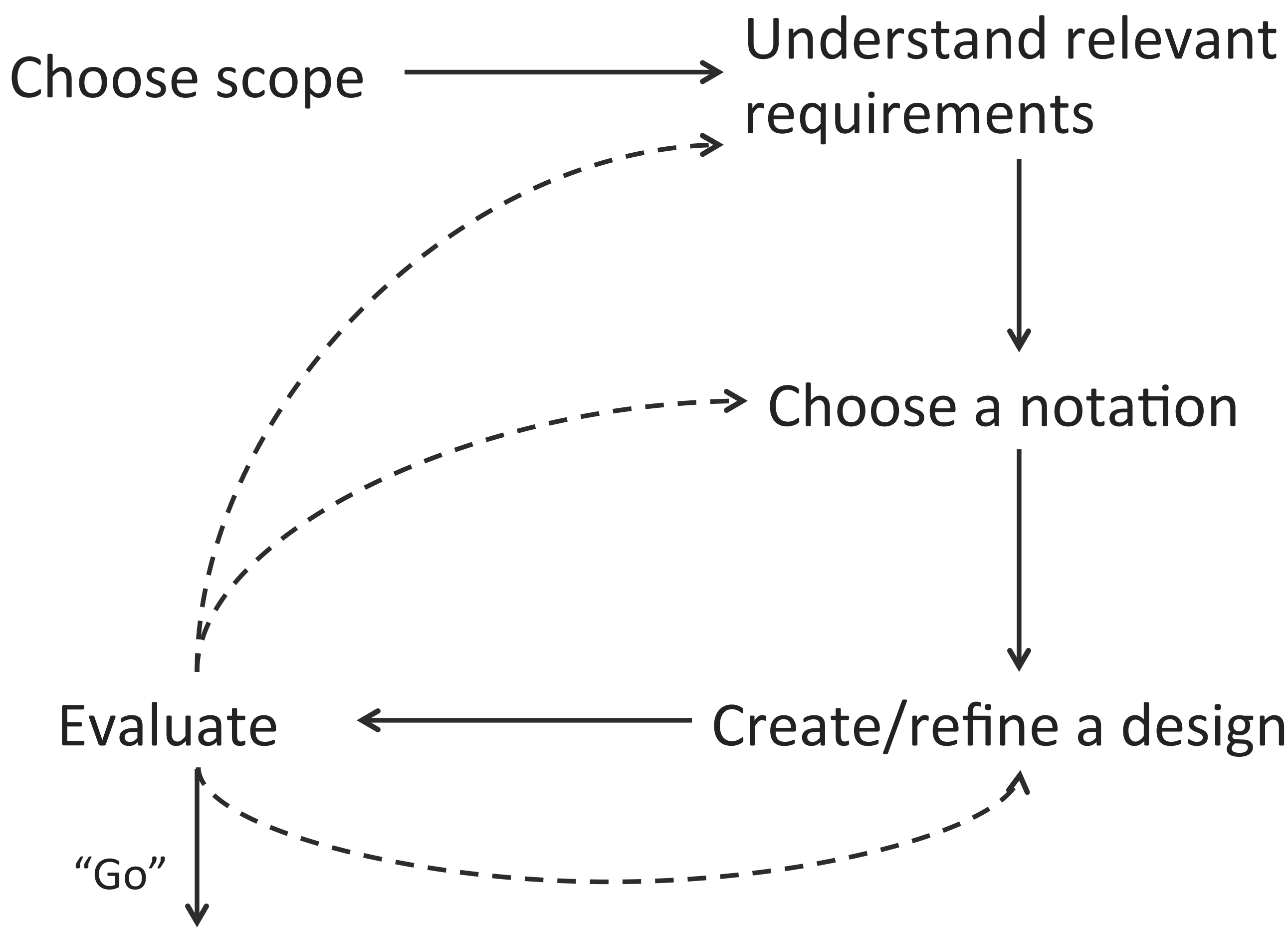
Architecture as process
(activities around the other two)

institute for SOFTWARE RESEARCH

Architecture design process

- Choose part or whole system to focus on
- Understand relevant requirements
- Choose a notation
  - Type of view, vocabulary of elements
- Create a design
  - Patterns, tactics
- Evaluate
- Go vs no-go
  - Issues feed back into process

Architecture design process



Source: ACDM, ADD

Architectural decisions

- Heart of architecture – deciding which path to go
- Involve tradeoff analysis
- Representing the alternatives clearly – half of work

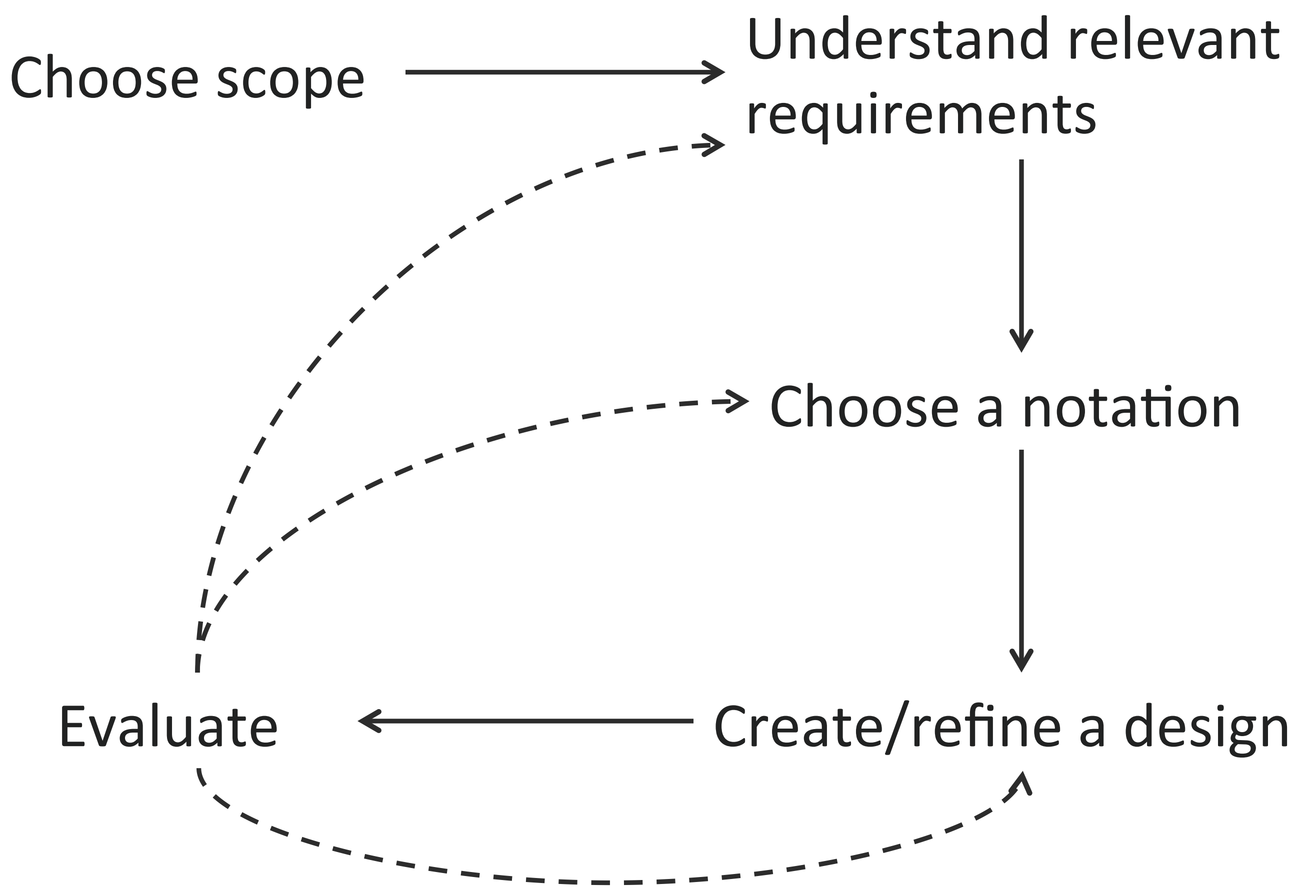
Architectural decisions

- Software architecture is *design*

  "Engineering design is […] a decision-making process (often iterative), in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective" – ABET

- A decision is a step in the process
  - **Record rationale!** (not just diagrams)
  - Tradeoffs
  - Backtracking

Architecture design process



Source: ACDM, ADD

Architectural decisions

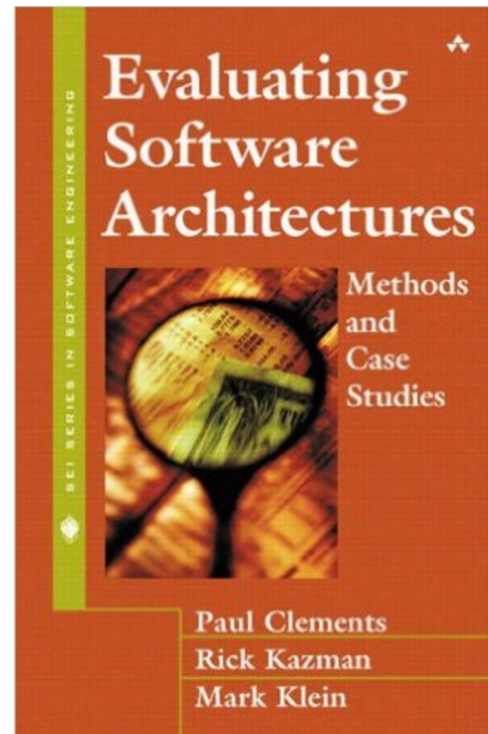
- Software architecture is *design*

  "Engineering design is […] a decision-making process (often iterative), in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective" – ABET

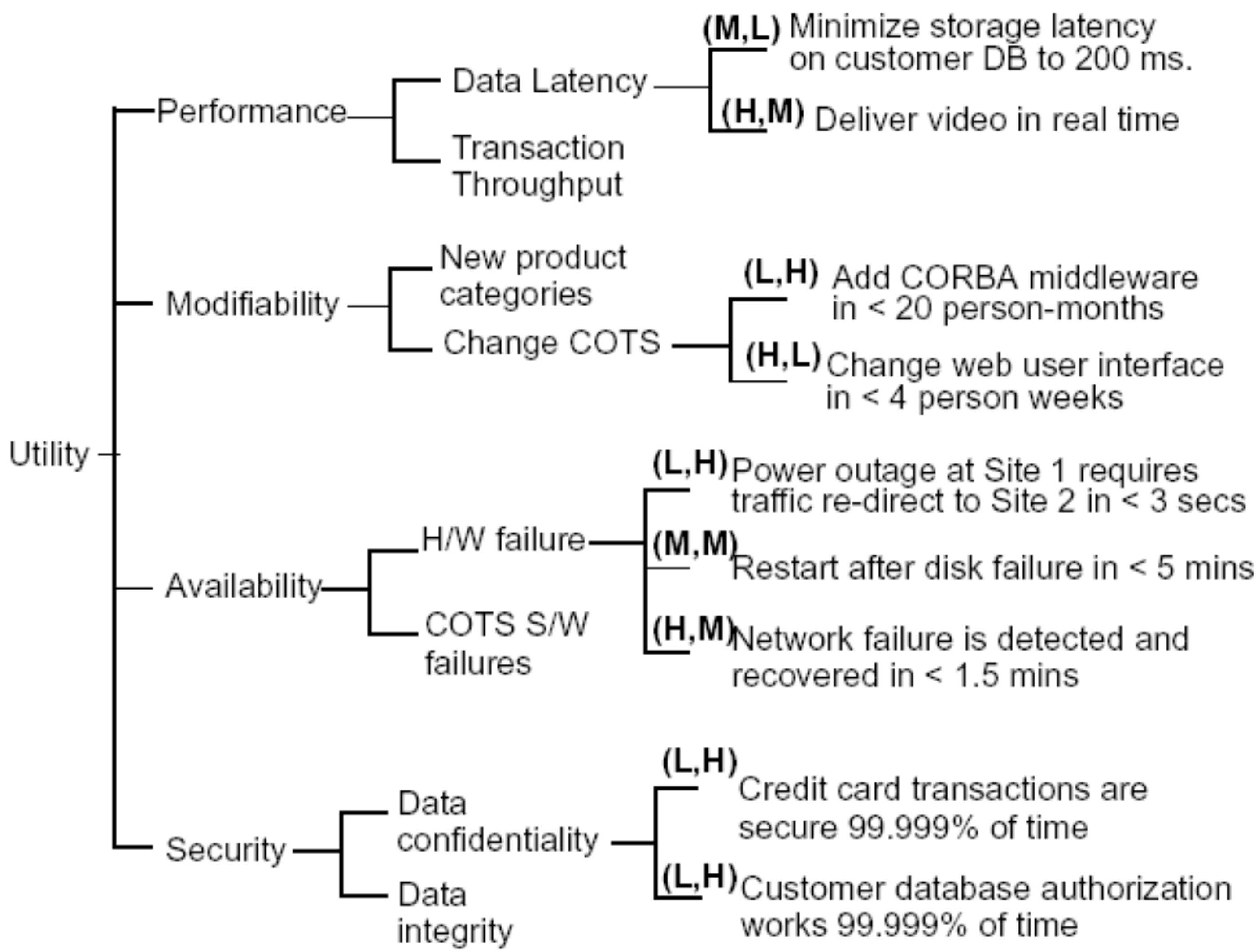
- A decision is a step in the process
  - **Record rationale!** (not just diagrams)
  - Tradeoffs
  - Backtracking

Architecture evaluation

- Goal: does the architecture satisfy requirements?
- ATAM – Architecture Tradeoff Analysis Method
  - Present requirements
  - Present architecture
  - Analyze architecture
  - Present results – risks and non-risks

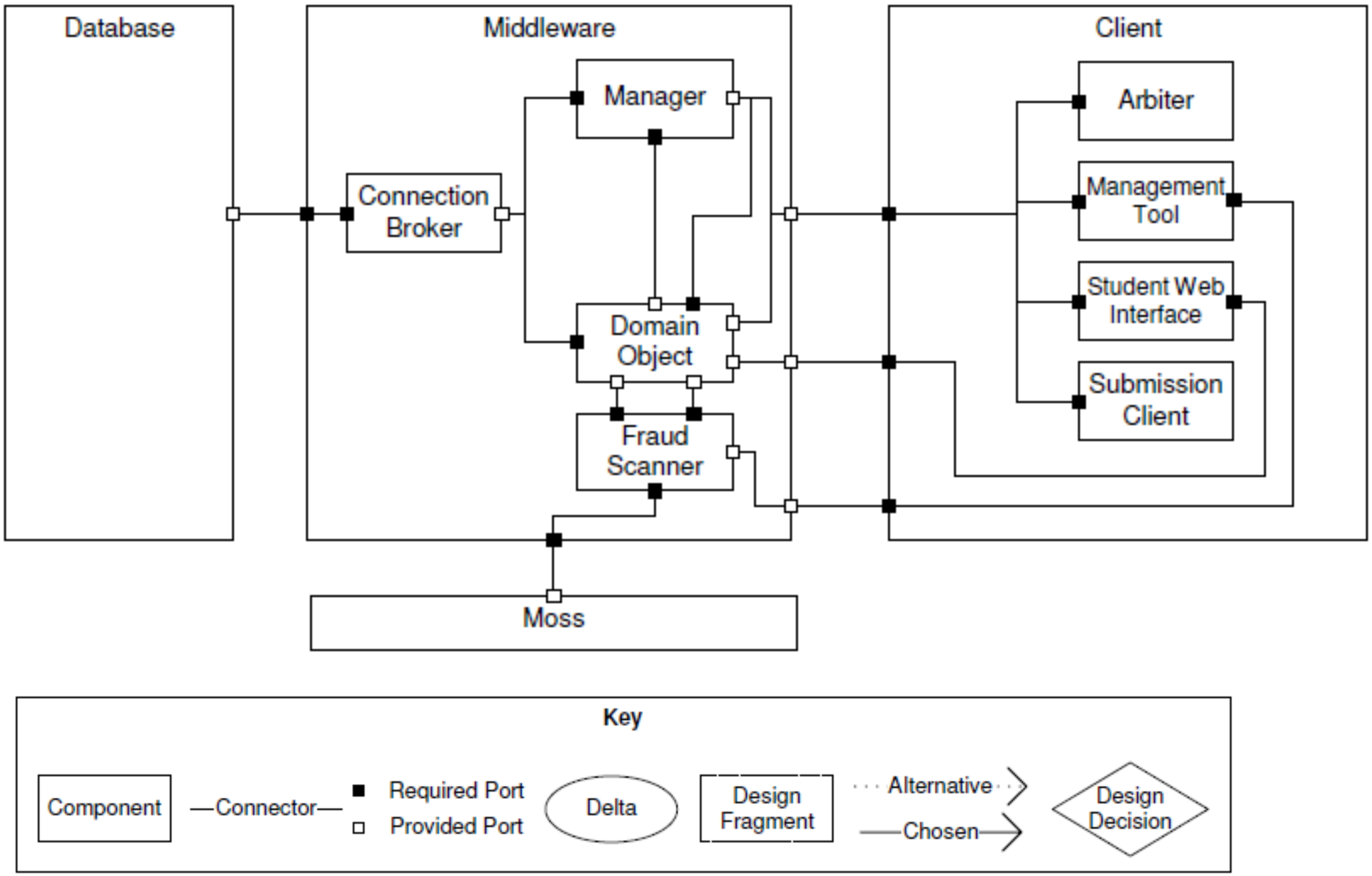


Source:sei.cmu.edu

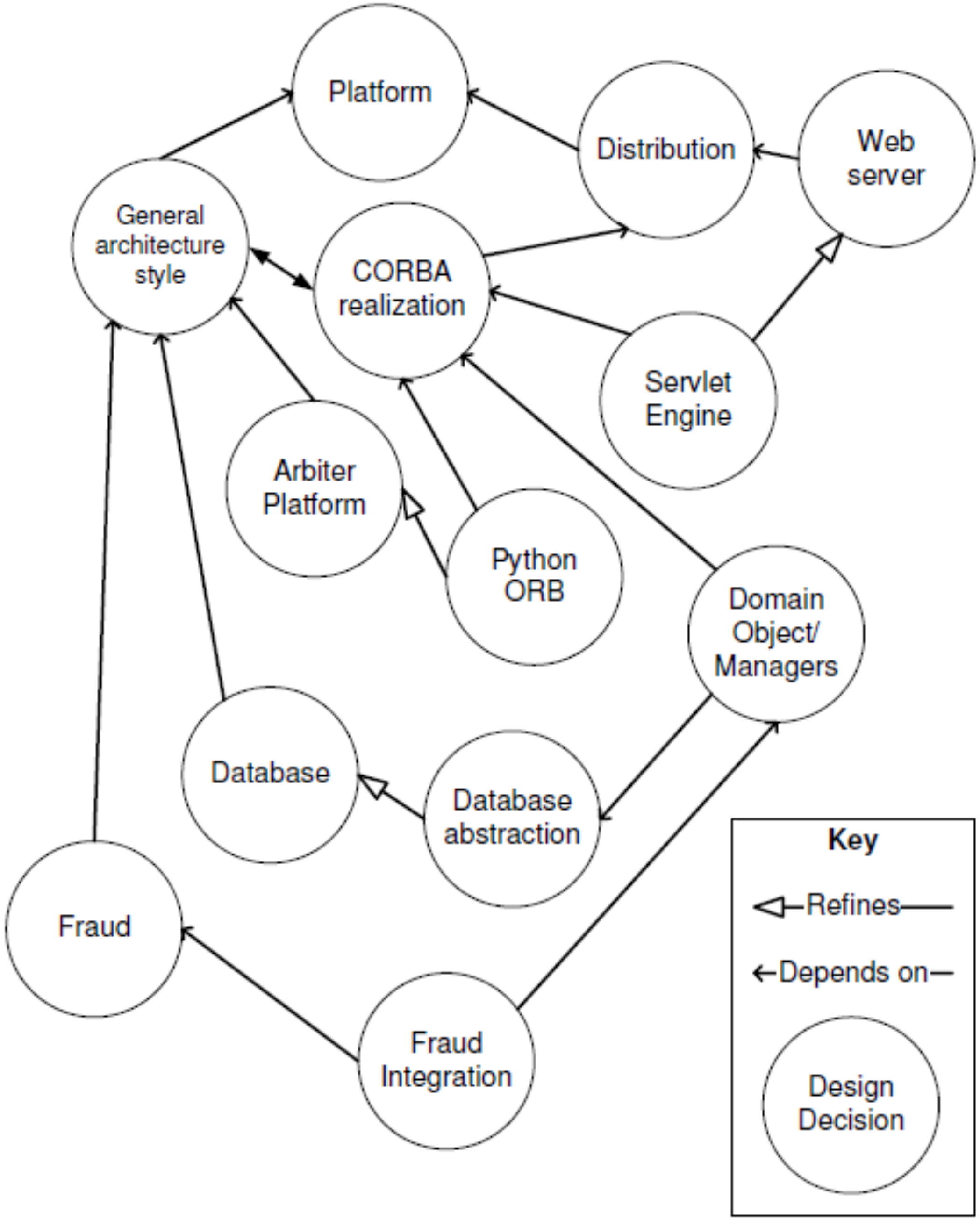


Source:arnon.me

Athena – code review system

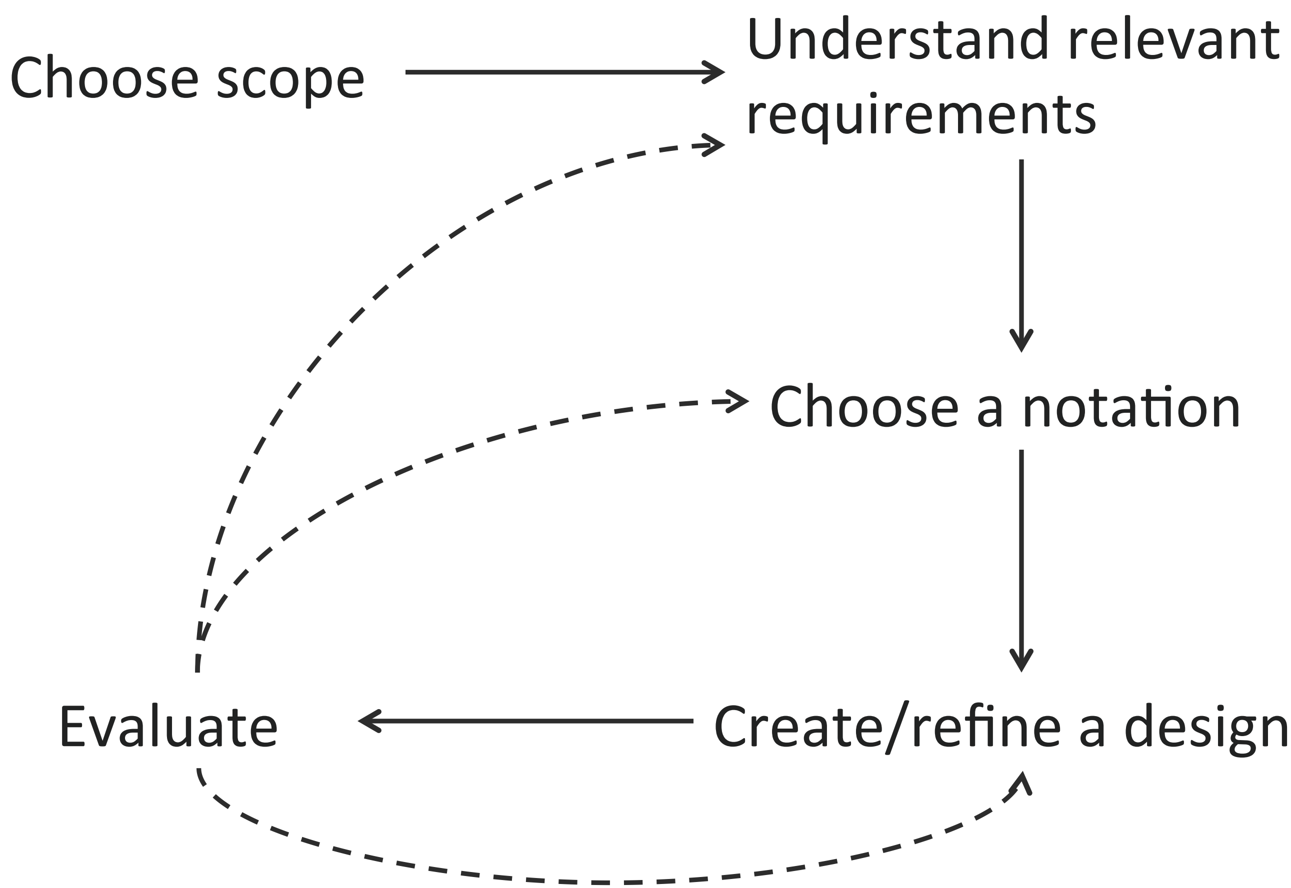


Source: Jansen and Bosch 2005



Source: Jansen and Bosch 2005

Architecture design process



Source: ACDM, ADD

Challenges of architecting

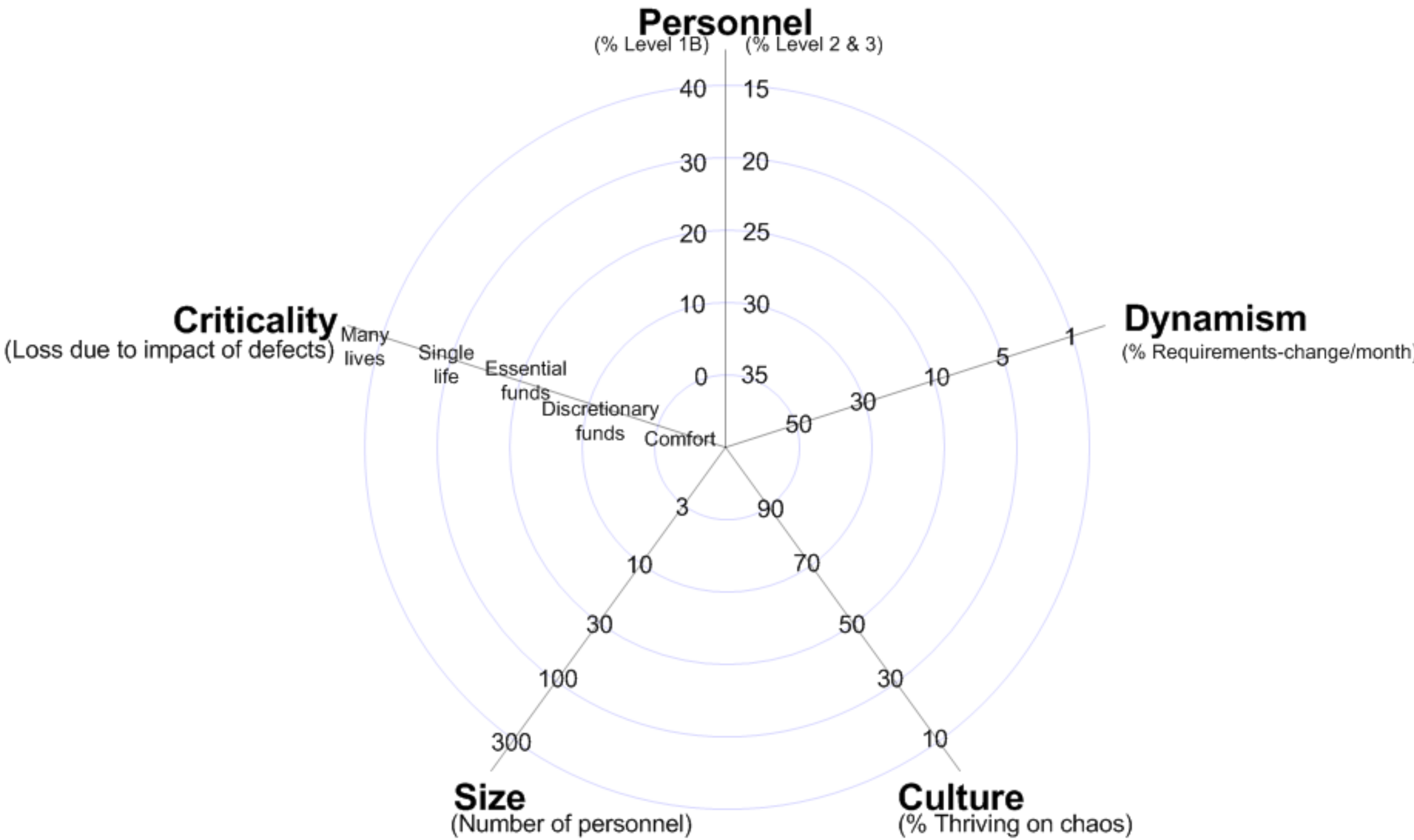
- Describe the system that is not built yet
- Domain knowledge is essential
- Huge space of options
- Heavily reliant on judgment

How much architecture?

- Design and document when needed, based on risk
- When:
  - Beginning
  - Whenever circumstances change
- Agile

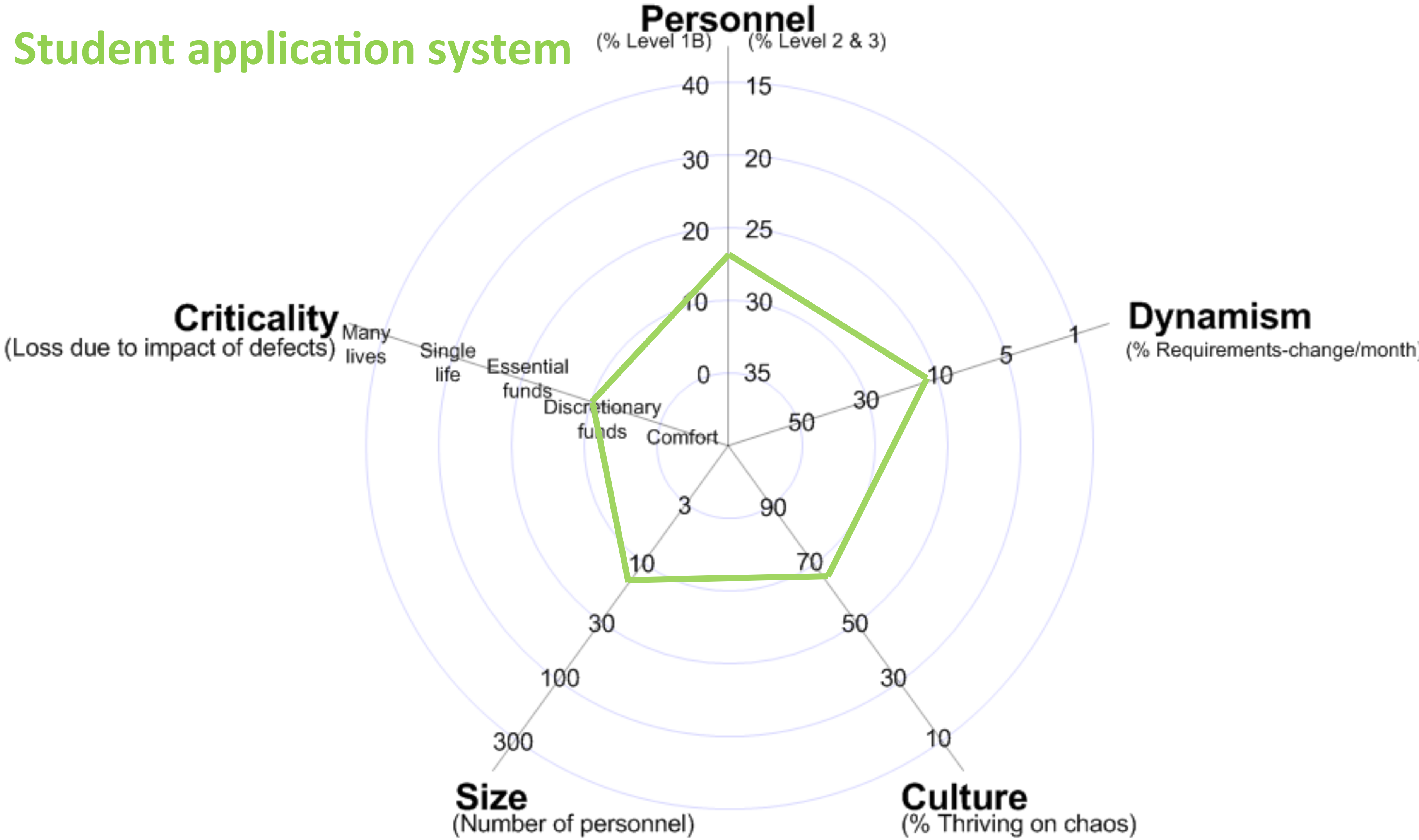
How much architecture?

- YAGNI
- Risk
- When to start:
  - Before implementation
  - Circumstances change
- When to stop:
  - Well-defined, requirements addressed, passes evaluation



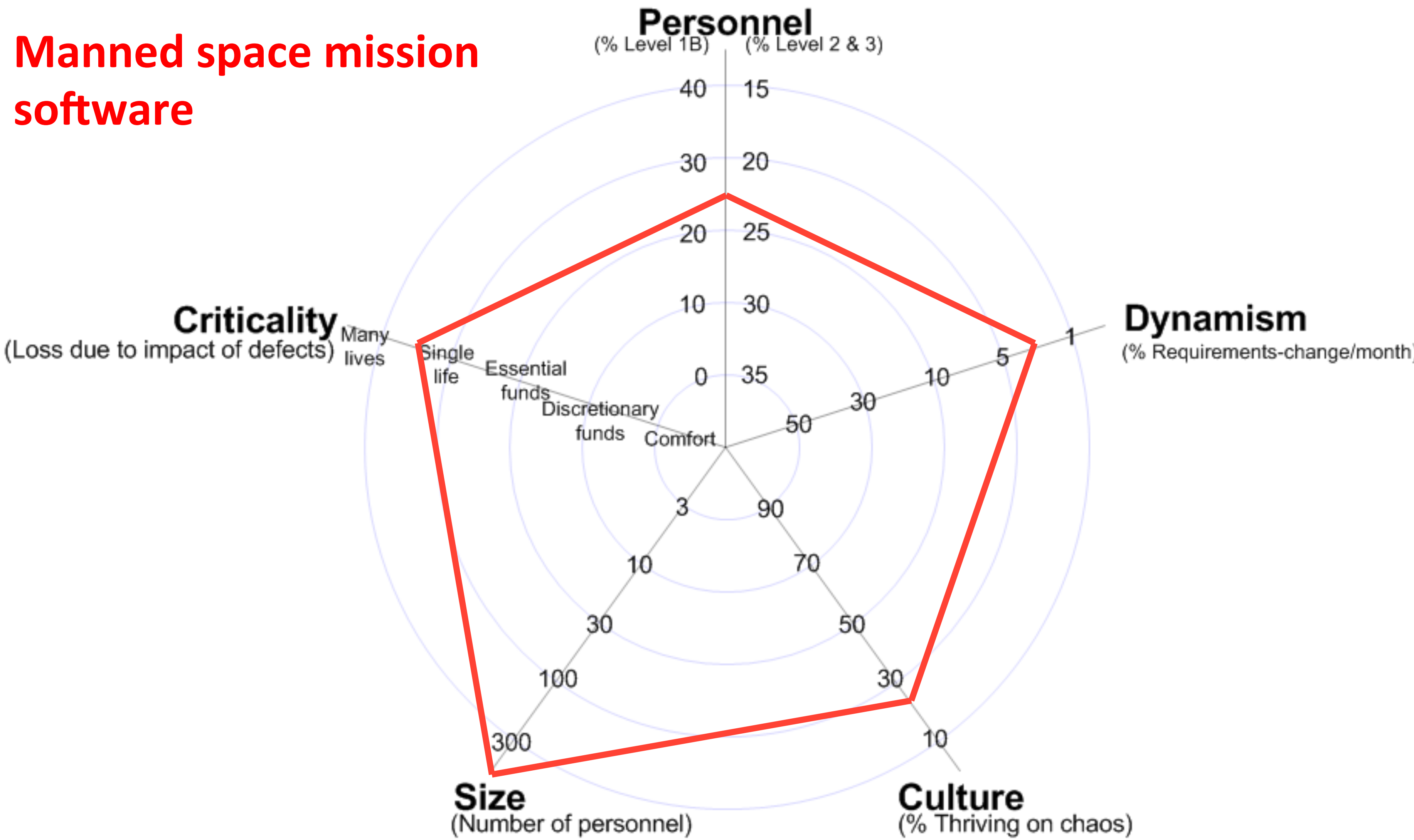
Source: Boehm and Turner 2003

Student application system



Source: Boehm and Turner 2003

Manned space mission software



Source: Boehm and Turner 2003

Challenges of architecting

- Describe the system that is not built yet
- Domain knowledge is essential
- Huge space of options
- Heavily reliant on judgment

Summary

Architecture as
structures and relations

- Patterns
- Tactics

Architecture as
documentation

- Views
- Rationale

Architecture as process

- Decisions
- Evaluation
- Reconstruction
- Agile

References

- Bass, Clements, Kazman. Software Architecture in Practice, 2013.
- Lattanze. Architecting Software Intensive Systems: a Practitioner's Guide, 2009.
- Clements, Bachmann, Bass, Garlan, Ivers, Little, Merson, Nord, Stafford. Documenting Software Architectures: Views and Beyond, 2010.
- Boehm and Turner. Balancing Agility and Discipline: A Guide for the Perplexed, 2003.
- Jansen and Bosch. Software Architecture as a Set of Architectural Design Decisions, WICSA 2005.

institute for SOFTWARE RESEARCH

Further Readings

- Bass, Clements, Kazman. Software Architecture in Practice, 2013.
- Lattanze. Architecting Software Intensive Systems: a Practitioner's Guide, 2009.
- Clements, Bachmann, Bass, Garlan, Ivers, Little, Merson, Nord, Stafford. Documenting Software Architectures: Views and Beyond, 2010.
- Boehm and Turner. Balancing Agility and Discipline: A Guide for the Perplexed, 2003.
- Jansen and Bosch. Software Architecture as a Set of Architectural Design Decisions, WICSA 2005.

Levels of abstraction

- Requirements
  – high-level "what" needs to be done
- Architecture (High-level design)
  – high-level "how", mid-level "what"
- OO-Design (Low-level design, e.g. design patterns)
  – mid-level "how", low-level "what"
- Code
  – low-level "how"

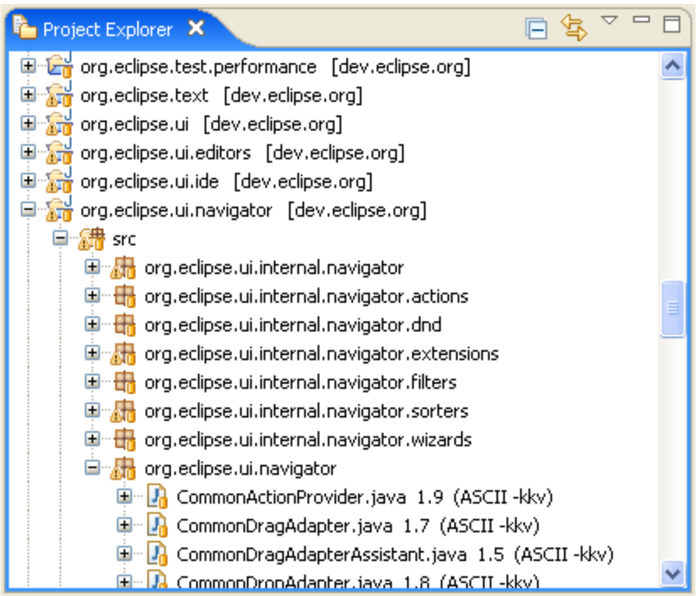
Architecture reconstruction

- Goal: describe architecture of an existing system given its source code
- Difficulty: level of abstraction in programming language is too low
- Process:
  - Iterative
  - Interpretive
  - Interactive

Reconstruction steps

- Extract *raw views*
  - Tool assistance, static & dynamic analysis
- Construct a database
  - Aggregate large volumes of data
- View fusion
  - Synthesize a hypothetical view
- Check for violations

Architecture reconstruction

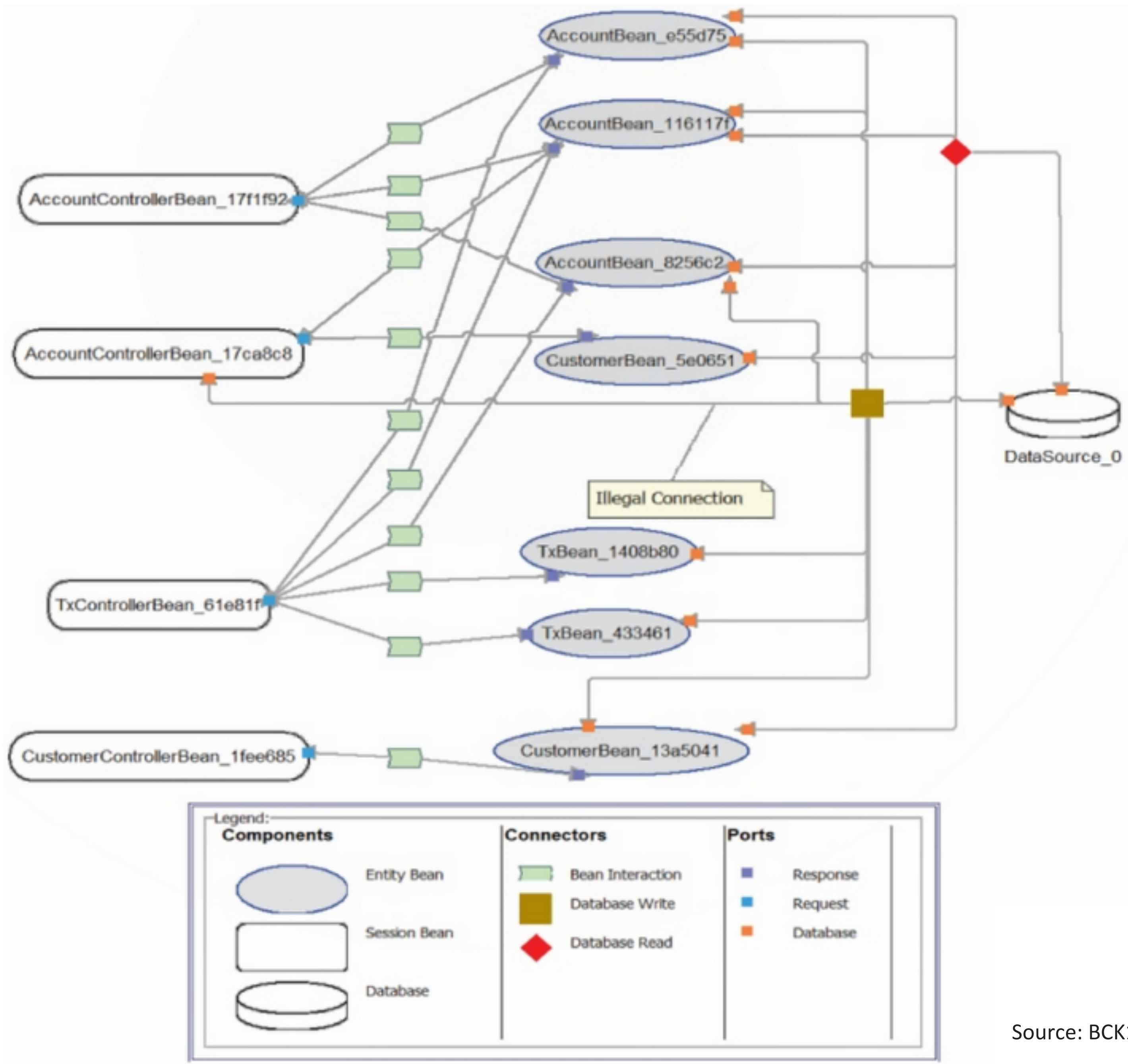



```
weblogic.application.utils.StateMachineDriver.nextState(StateMachineDriver.java:26)
>
####<Dec 29, 2006 2:14:24 PM IST> <Notice> <Log Management> <svaidyan02> <xbusServer>
<[ACTIVE] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'> <<WLS
Kernel>> <> <> <1167381864275> <BEA-170027> <The server initialized the domain log
broadcaster successfully. Log messages will now be broadcasted to the domain log.>
####<Dec 29, 2006 2:14:24 PM IST> <Notice> <WebLogicServer> <svaidyan02> <xbusServer> <Main
Thread> <<WLS Kernel>> <> <> <1167381864976> <BEA-000365> <Server state changed to ADMIN>
####<Dec 29, 2006 2:14:24 PM IST> <Notice> <WebLogicServer> <svaidyan02> <xbusServer> <Main
Thread> <<WLS Kernel>> <> <> <1167381864996> <BEA-000365> <Server state changed to RESUMING>
####<Dec 29, 2006 2:14:28 PM IST> <Notice> <Security> <svaidyan02> <xbusServer> <[STANDBY]
ExecuteThread: '5' for queue: 'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1167381868541> <BEA-090171> <Loading the identity certificate and private key stored under
the alias DemoIdentity from the jks keystore file
C:\bea2613a\WEBLOG~1\server\lib\DemoIdentity.jks.>
####<Dec 29, 2006 2:14:29 PM IST> <Notice> <Security> <svaidyan02> <xbusServer> <[STANDBY]
ExecuteThread: '5' for queue: 'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1167381869643> <BEA-090169> <Loading trusted certificates from the jks keystore file
C:\bea2613a\WEBLOG~1\server\lib\DemoTrust.jks.>
####<Dec 29, 2006 2:14:29 PM IST> <Notice> <Security> <svaidyan02> <xbusServer> <[STANDBY]
ExecuteThread: '5' for queue: 'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1167381869713> <BEA-090169> <Loading trusted certificates from the jks keystore file
C:\bea2613a\JROCKI~1\jre\lib\security\cacerts.>
####<Dec 29, 2006 2:15:32 PM IST> <Warning> <Server> <svaidyan02> <xbusServer>
<DynamicSSLListenThread[DefaultSecure[1]]> <<WLS Kernel>> <> <> <1167381932743> <BEA-002611>
<Hostname "svaidyan02.apac.bea.com", maps to multiple IP addresses: 192.168.1.5,
172.22.56.120>
####<Dec 29, 2006 2:15:32 PM IST> <Notice> <Server> <svaidyan02> <xbusServer> <[STANDBY]
ExecuteThread: '5' for queue: 'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1167381932753> <BEA-002613> <Channel "Default[2]" is now listening on 127.0.0.1:7021 for
```

- Iterative
- Interpretive
- Interactive

Reconstruction steps

- Extract *raw views*
  - Tool assistance, static & dynamic analysis
- Construct a database
  - Aggregate large volumes of data
- View fusion
  - Synthesize a hypothetical view
- Check for violations



Source: BCK13