Foundations of Software Engineering

Lecture 8: Software Architecture II

Christian Kästner

(with slides by Ivan Ruchkin)

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

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

institute for SOFTWARE RESEARCH

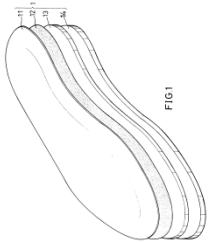
Levels of abstraction

- 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)

institute for SOFTWARE RESEARCH

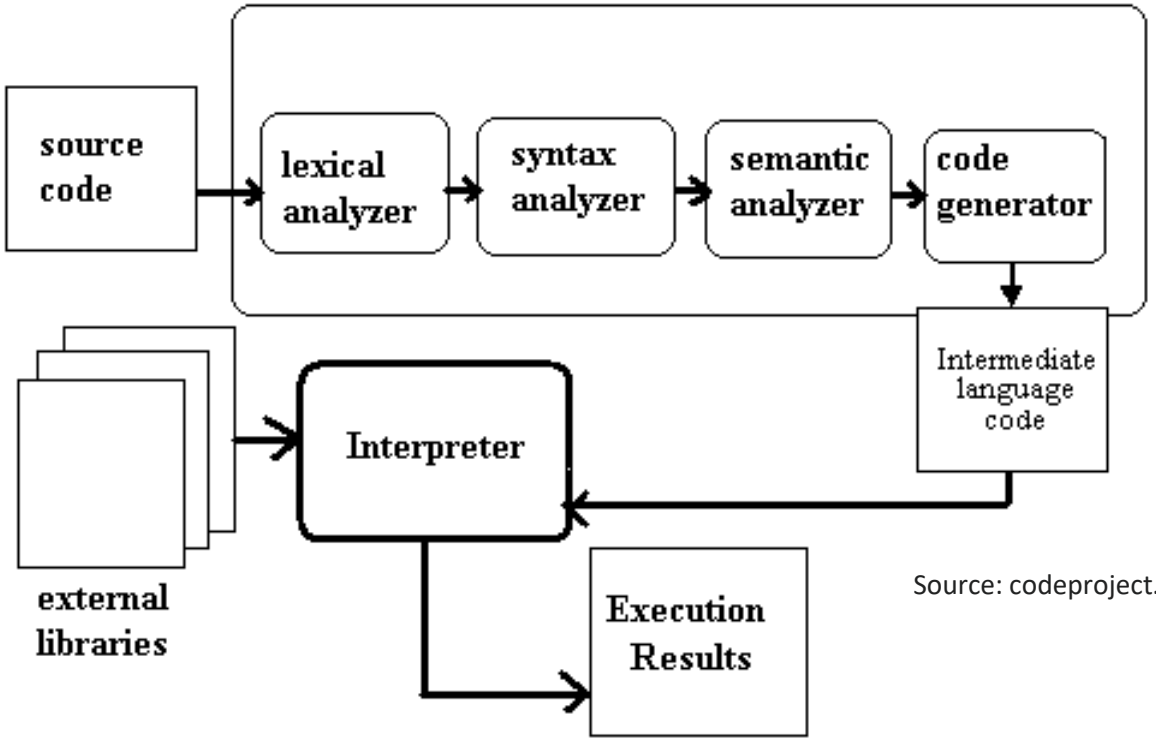
Architectural Styles and Tactics

Architectural style (pattern)

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

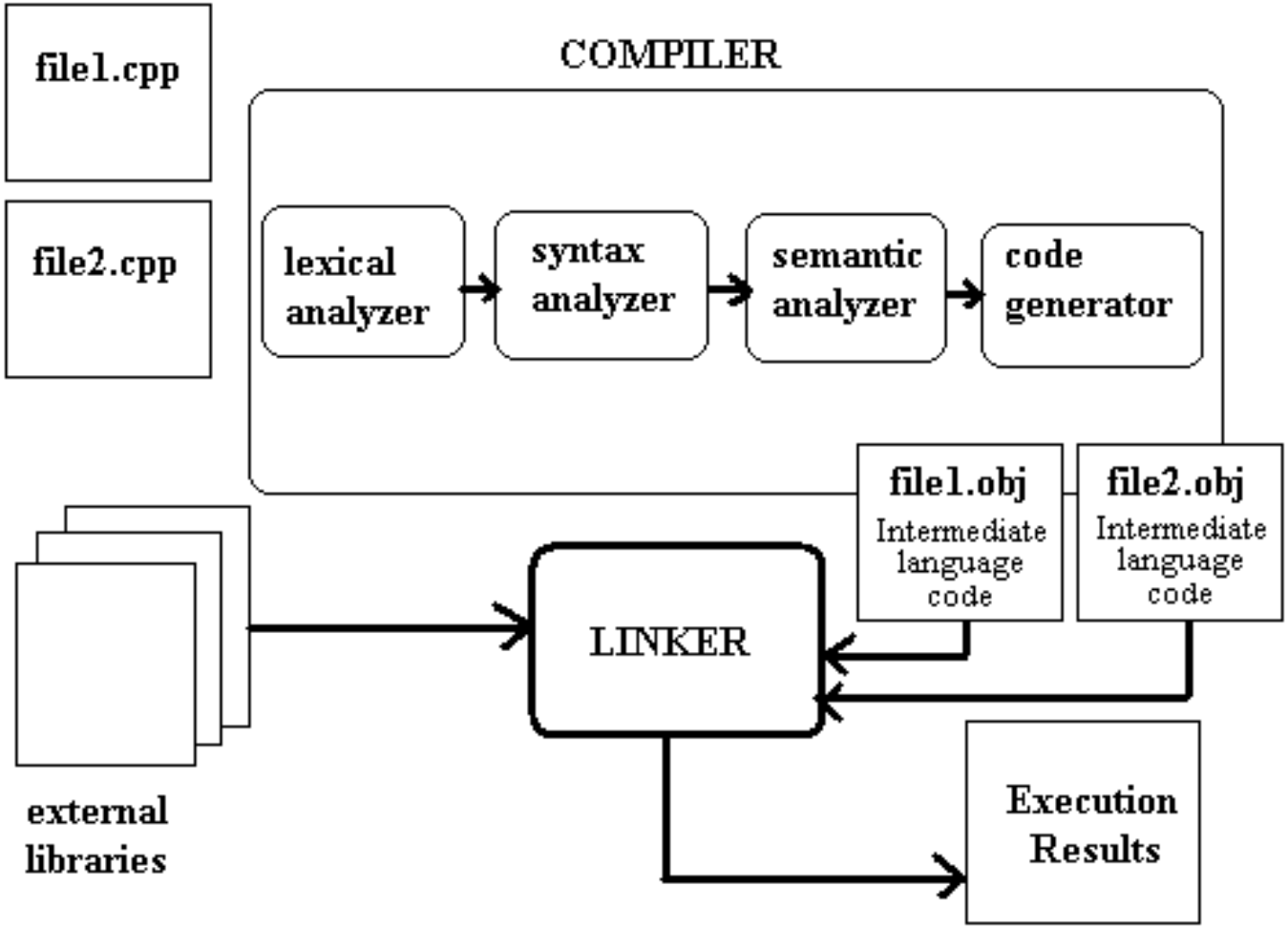
Architectural style (pattern)

- Broad principle of system organization
- See reading



Source: codeproject.org

Architectural style (pattern)

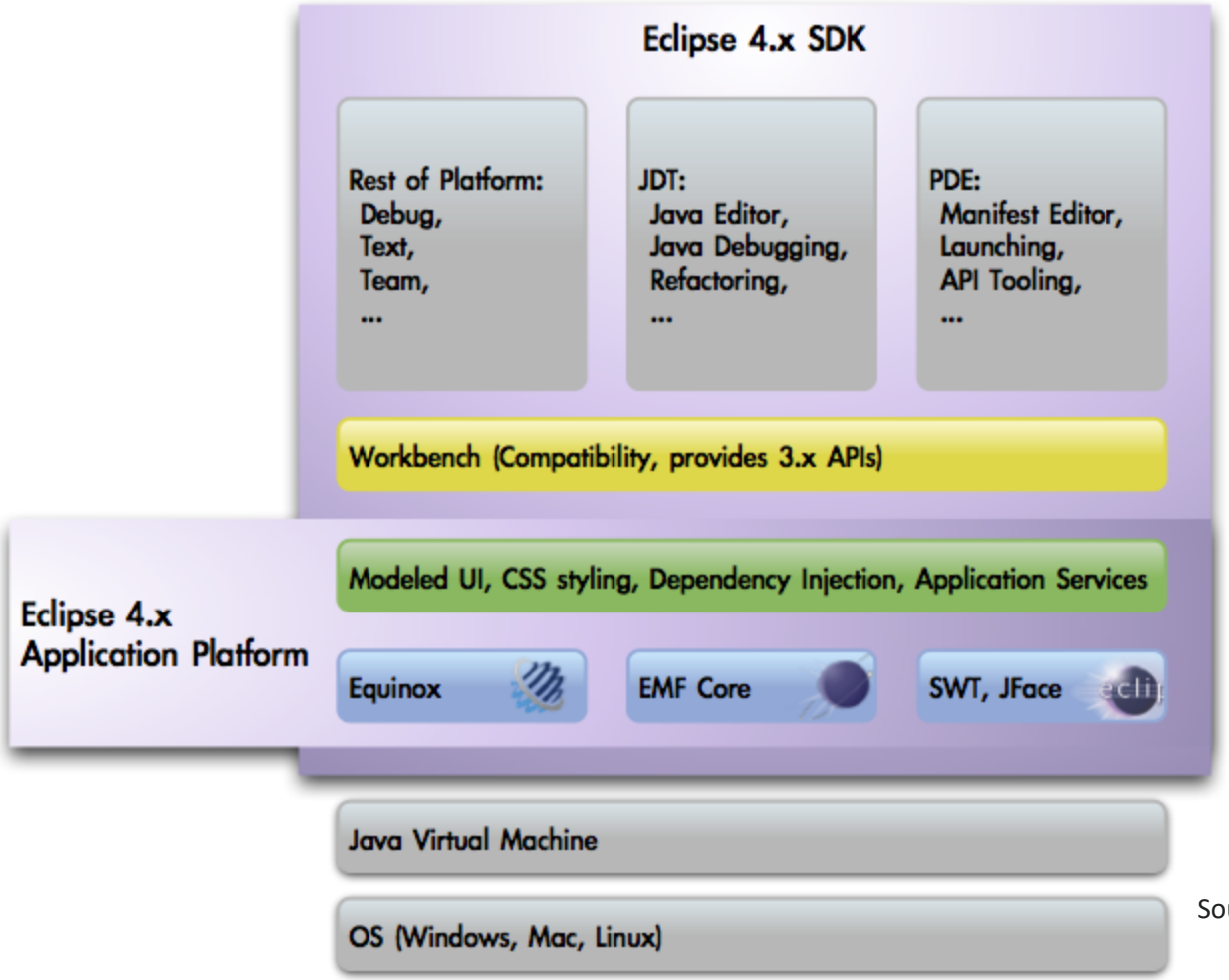


Source: codeproject.org

Client-server pattern

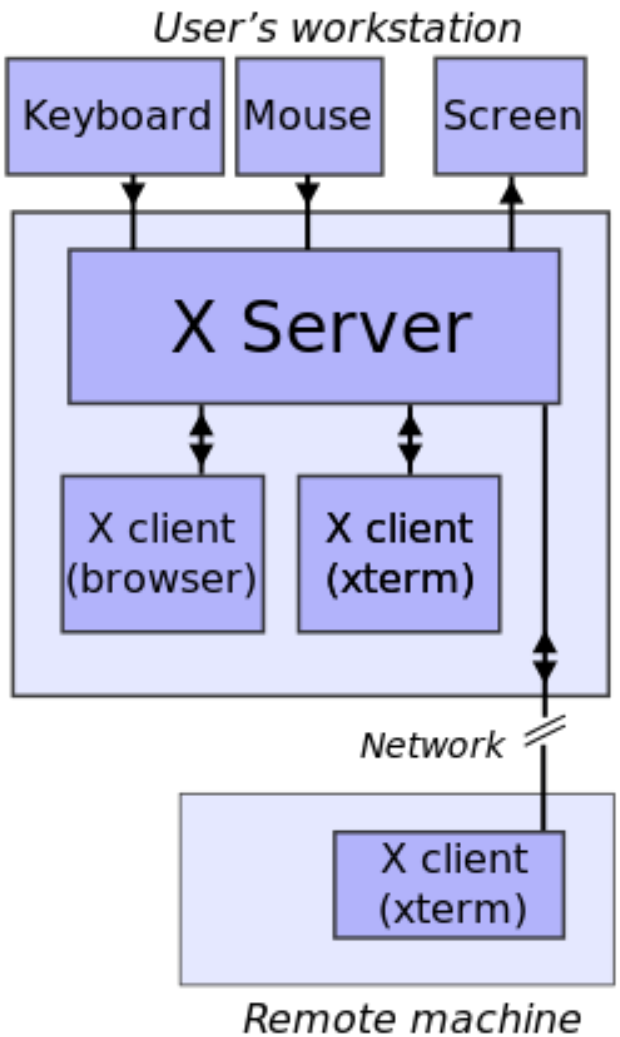
- Separation of clients and servers
  - Servers provide services; known and "stable"
  - Clients request services; come and go
- Varieties: synchronous/asynchronous
- Impact on security, performance, scalability
- Examples: TCP, HTTP, X11

Layered system



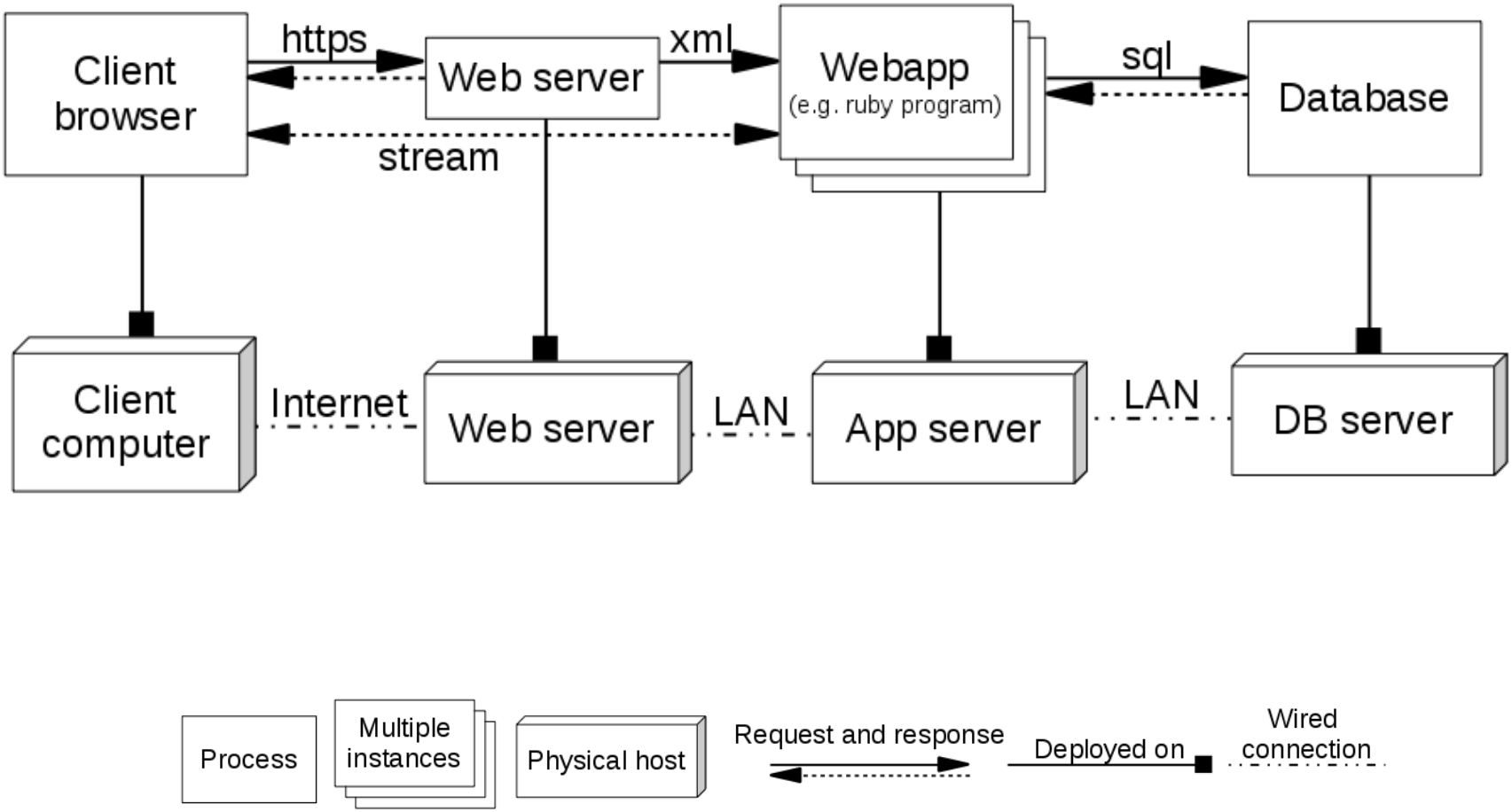
Source: eclipse.org

Client-server style



Source: wikimedia commons

Tiered architecture



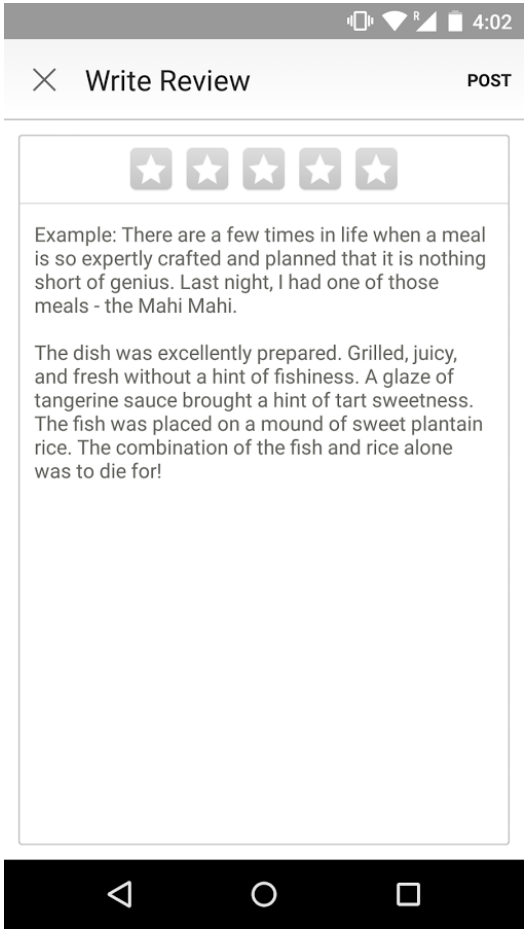
Client

Server

Database

Where to validate user input?

Example: Yelp App



Services?

Architectural Styles

- Pipes and Filters
- Object-Oriented Organization, Services
- Event-Based, Implicit Invocation
- Layered System
- Repositories
- …

Architectural Style

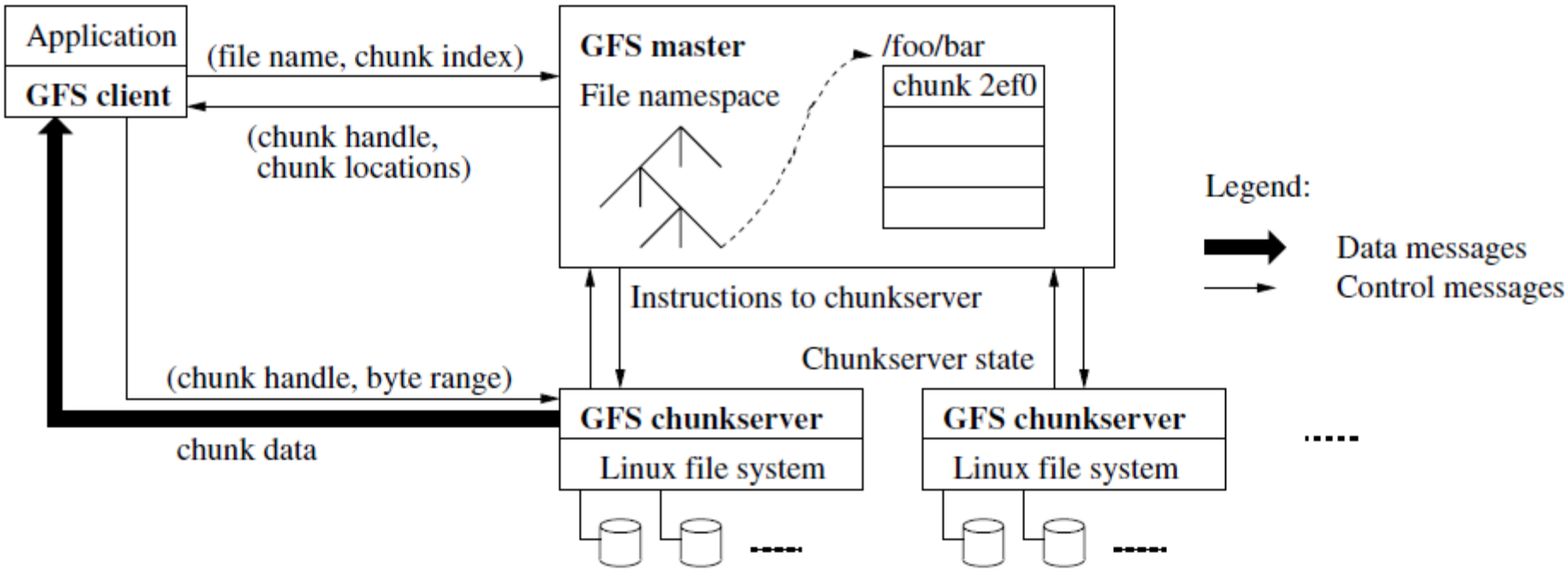
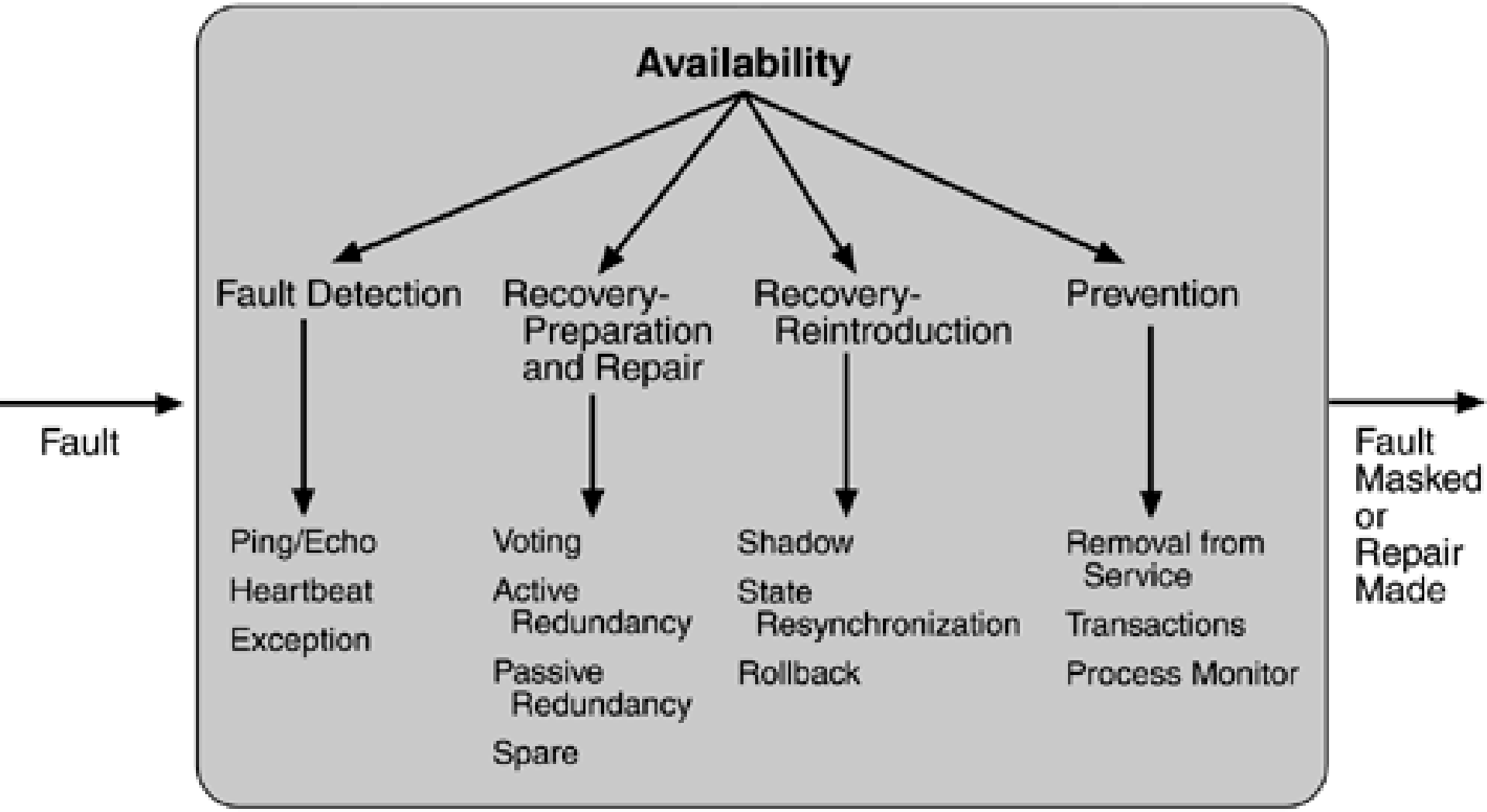


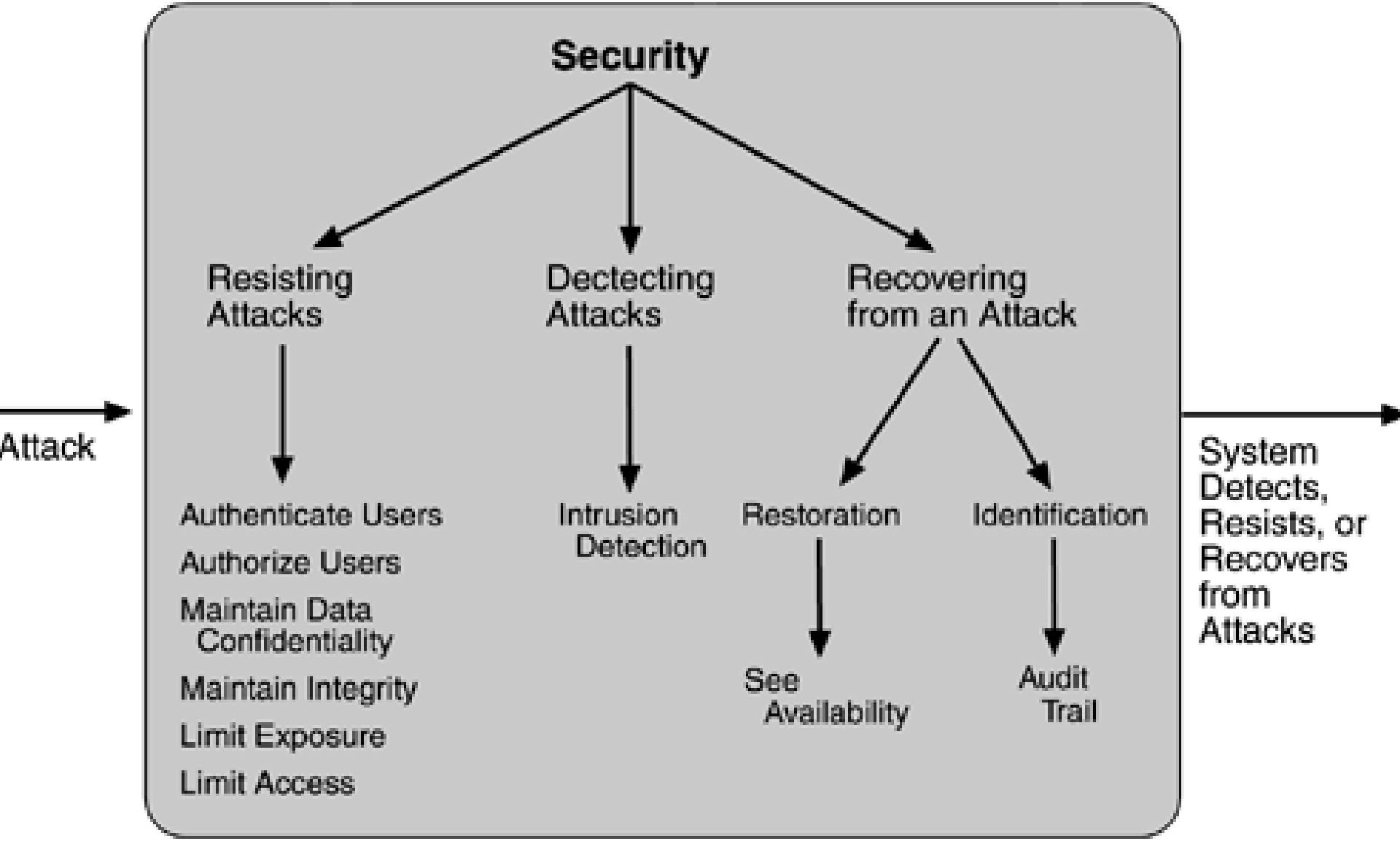
Figure 1: GFS Architecture

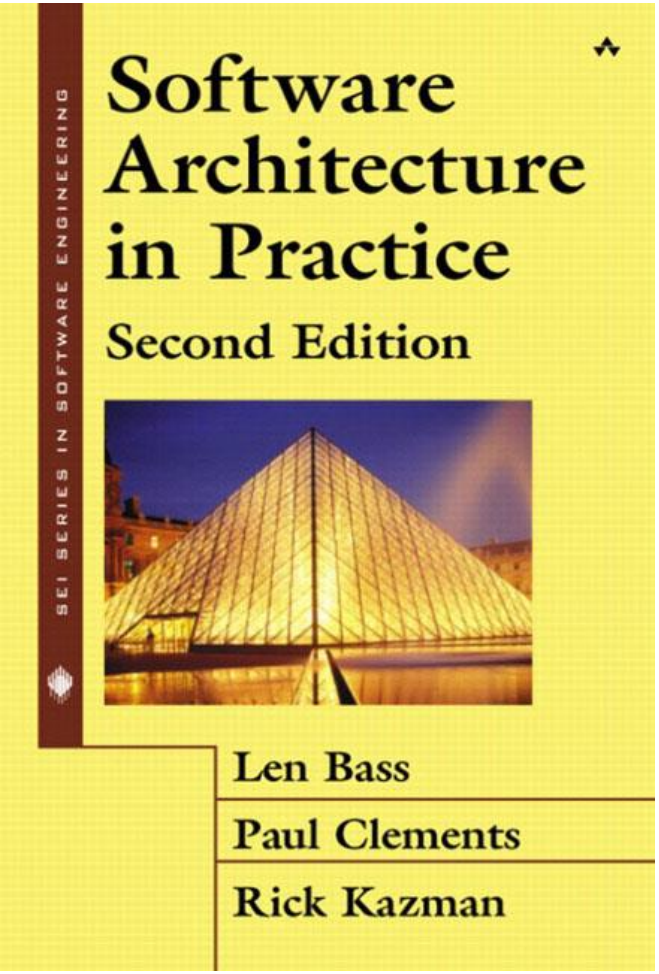
Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." *ACM SIGOPS operating systems review*. Vol. 37. No. 5. ACM, 2003.

Tactics

- Architectural techniques to achieve qualities
- 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


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


Security
Resisting Attacks
Dectecting Attacks
Recovering from an Attack
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



Many tactics described in Chapter 5

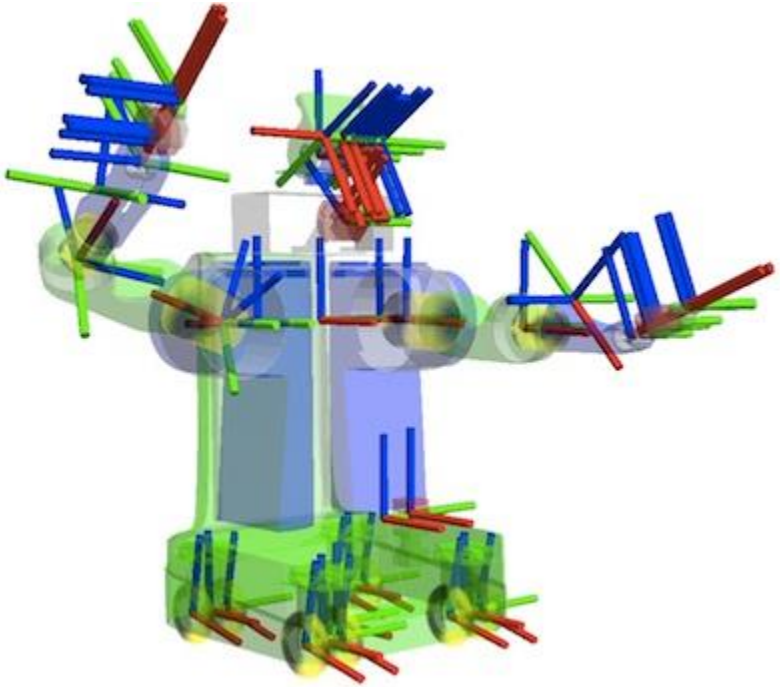
Brief high-level descriptions (about 1 paragraph per tactic)

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

Case Study: ROS

ROS

- "Robot Operating System", open source
- The philosophical goals of ROS can be summarized as:
  - Peer-to-peer
  - Tools-based
  - Multi-lingual
  - Thin
  - Free and Open-Source



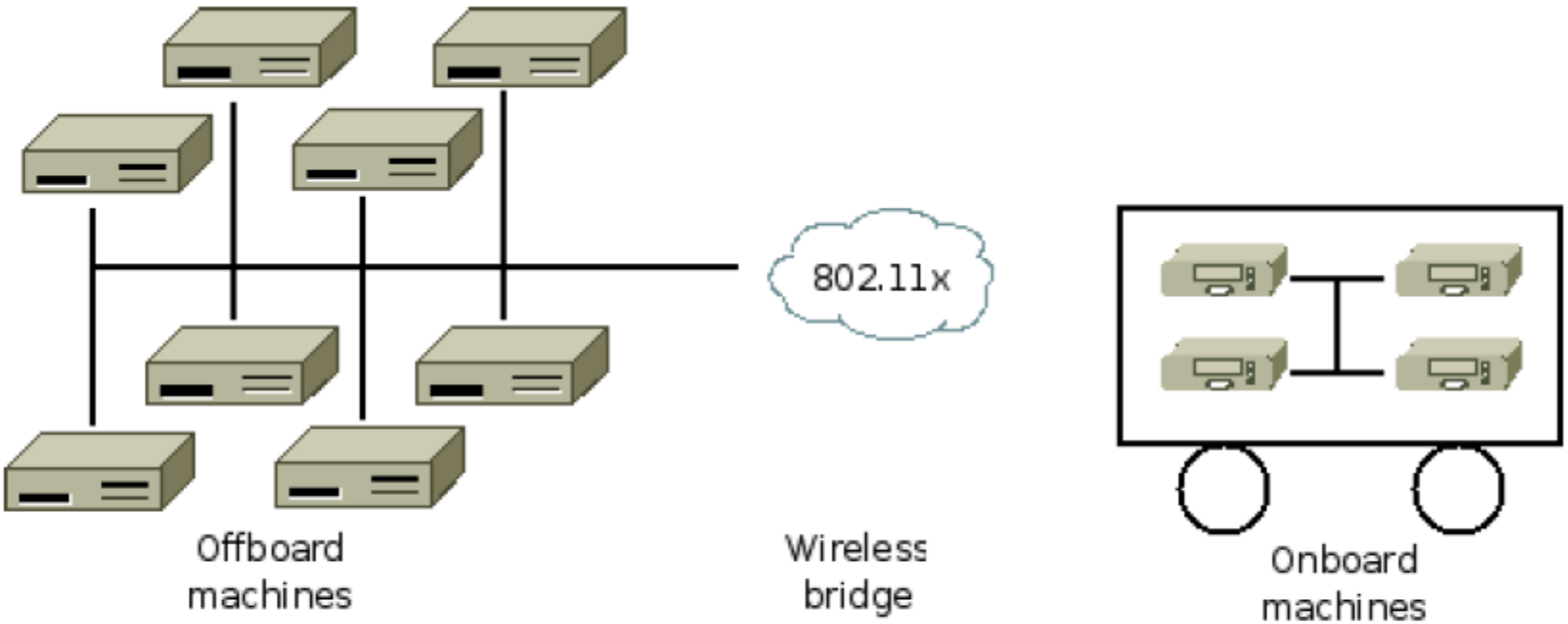


Fig. 1. A typical ROS network configuration

Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." *ICRA workshop on open source software*. Vol. 3. No. 3.2. 2009.

Quality Goals?

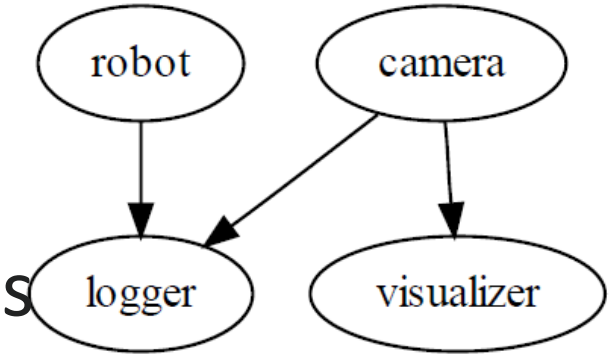
" A Distributed, Modular Design"

- users can use as much or as little of ROS as they desire
- modularity of ROS allows you to pick and choose which parts are useful for you and which parts you'd rather implement yourself
- large community of user-contributed packages (3000 packages)

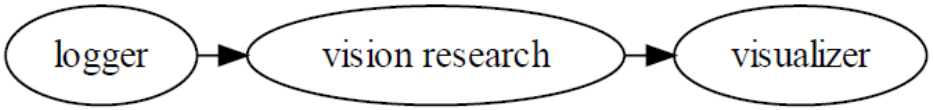
Architectural Style?

- Pipes and Filters
- Object-Oriented Organization, Services
- Event-Based, Implicit Invocation
- Layered System
- Repositories
- …

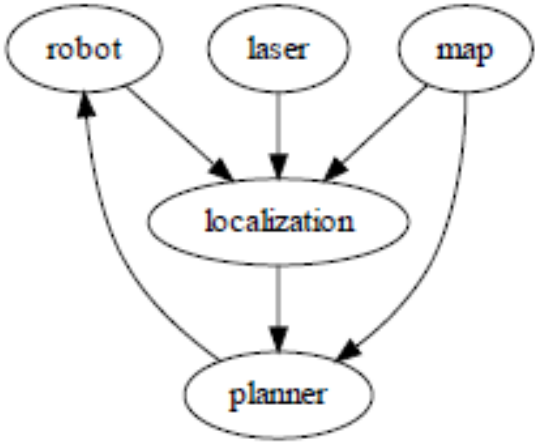
ROS Communication Infrastructure

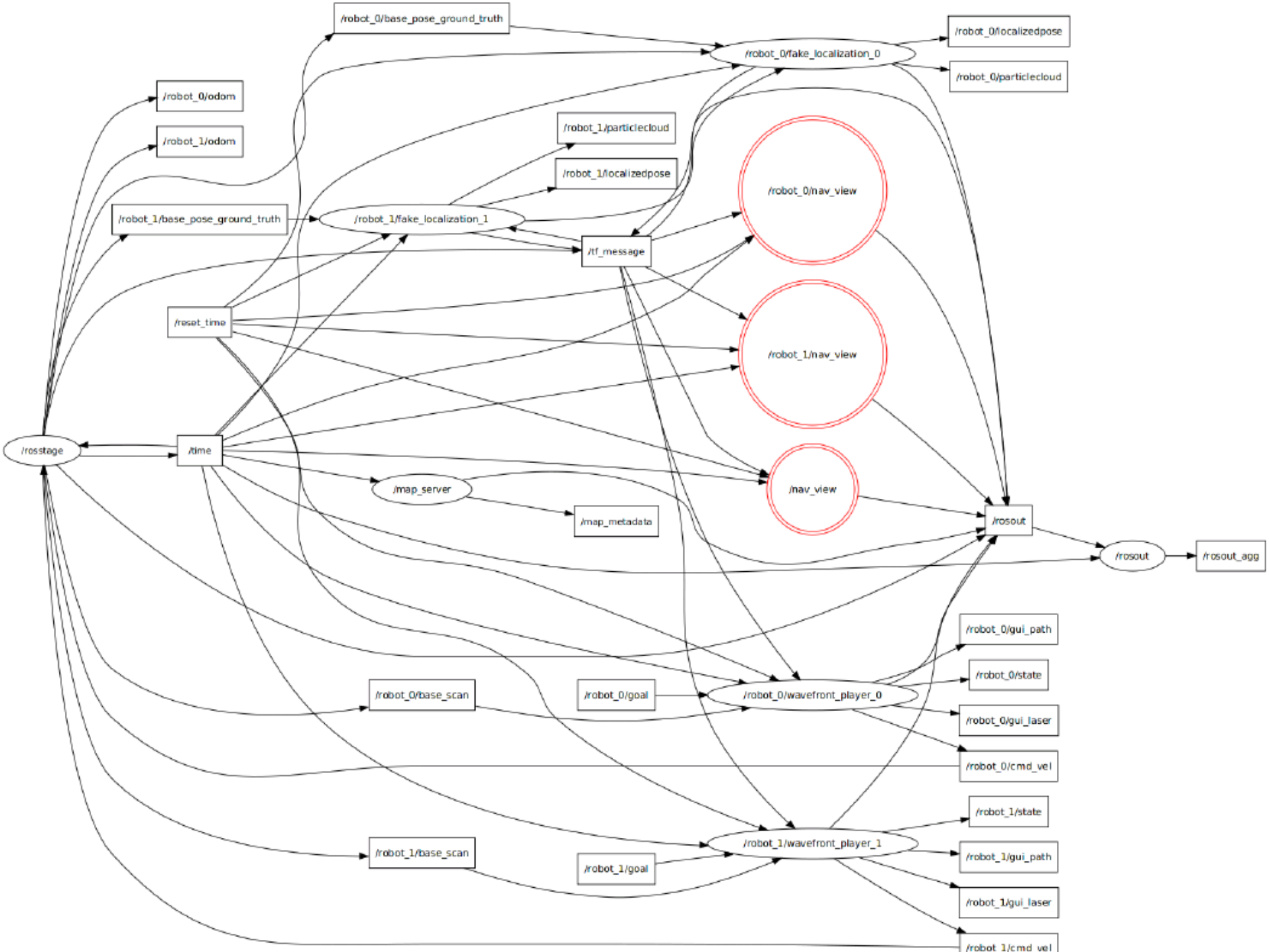


- Message Passing
  - Publish/subscribe for channels
  - Messages interfaces through IDL (cross-language)



- Recording and Playback of Messages
- Remote procedure calls
- Share configuration through global key-value store




/robot_0/base_pose_ground_truth
/robot_0/fake_localization_0
/robot_0/localizedpose
/robot_0/particlecloud
/robot_0/odom
/robot_1/odom
/robot_1/particlecloud
/robot_1/localizedpose
/robot_0/nav_view
/robot_1/base_pose_ground_truth
/robot_1/fake_localization_1
/tf_message
/reset_time
/robot_1/nav_view
/rosstage
/time
/map_server
/nav_view
/map_metadata
/rosout
/rosout
/rosout_agg
/robot_0/gui_path
/robot_0/state
/robot_0/base_scan
/robot_0/goal
/robot_0/wavefront_player_0
/robot_0/gui_laser
/robot_0/cmd_vel
/robot_1/state
/robot_1/base_scan
/robot_1/wavefront_player_1
/robot_1/goal
/robot_1/gui_path
/robot_1/gui_laser
/robot_1/cmd_vel

Tradeoff discussion

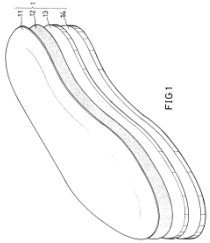
- Decoupling
- Reuse, Extensibility
- Reliability
- Understandability
- Performance
- Community contributions

Architecture Design Process

What is architecture?

Architecture as
structures and relations
(the actual system)

Architecture as
documentation
(representations of the system)



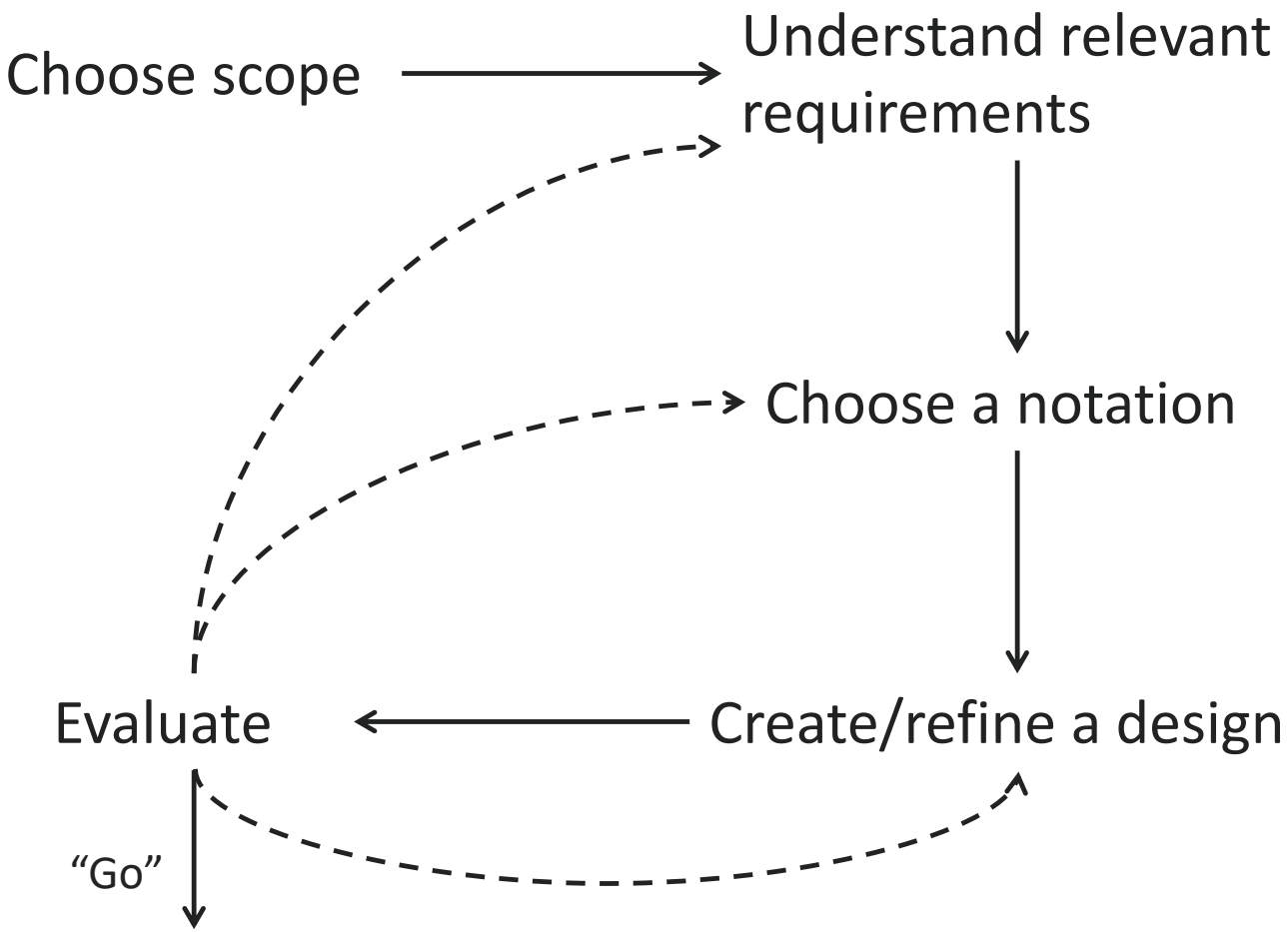
Architecture as process
(activities around the other two)

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

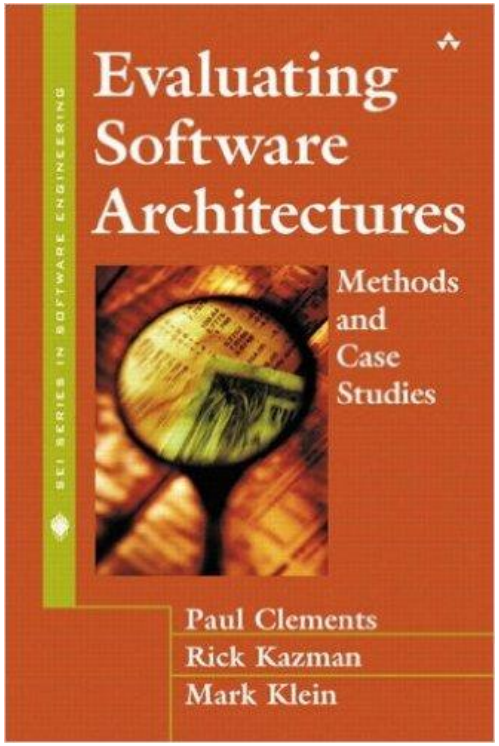
- Software architecture is *design*

  "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

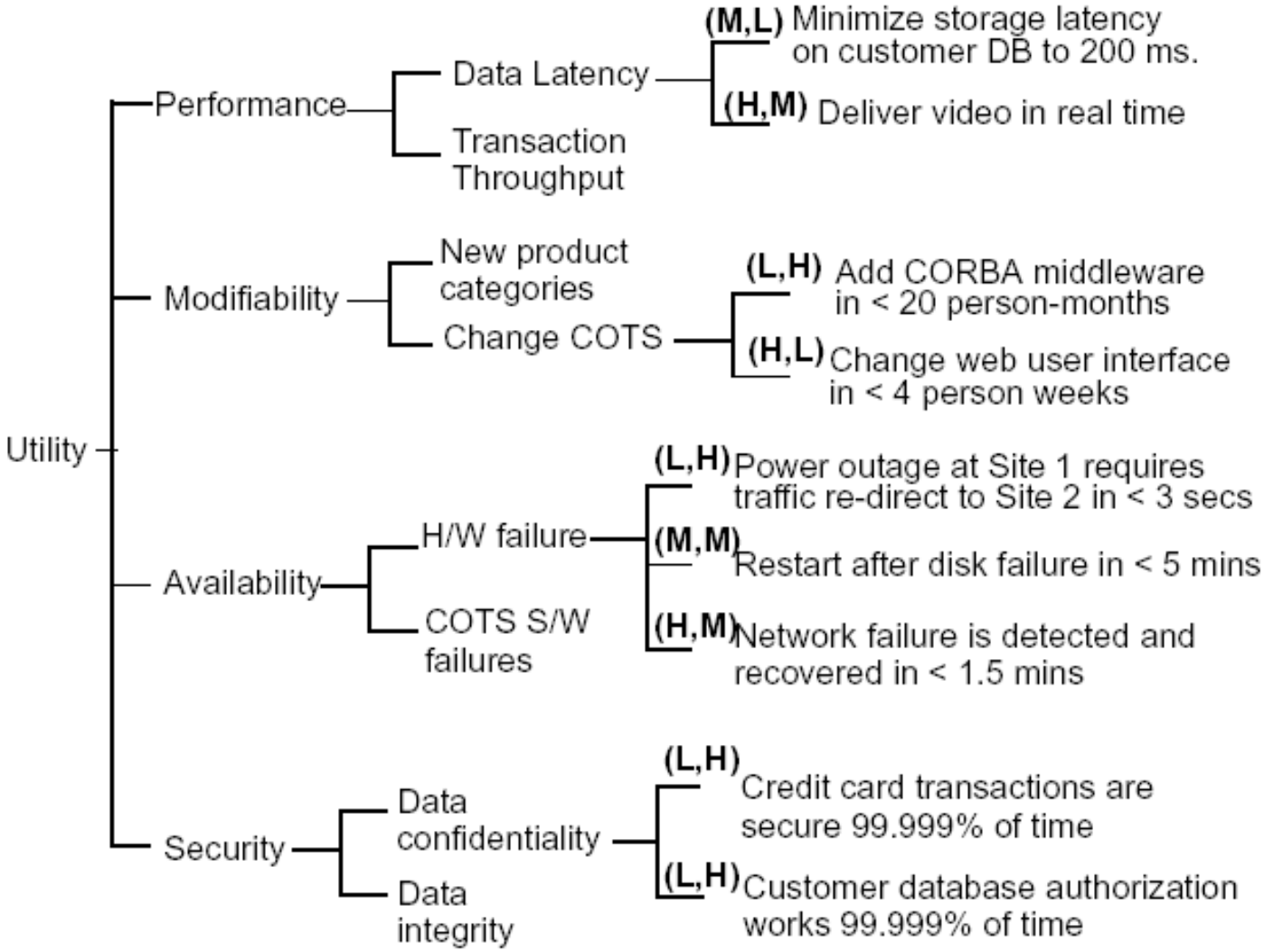
- A decision is a step in the process
  - **Record rationale!** (not just diagrams)
  - Tradeoffs
  - Backtracking

Architecture evaluation

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

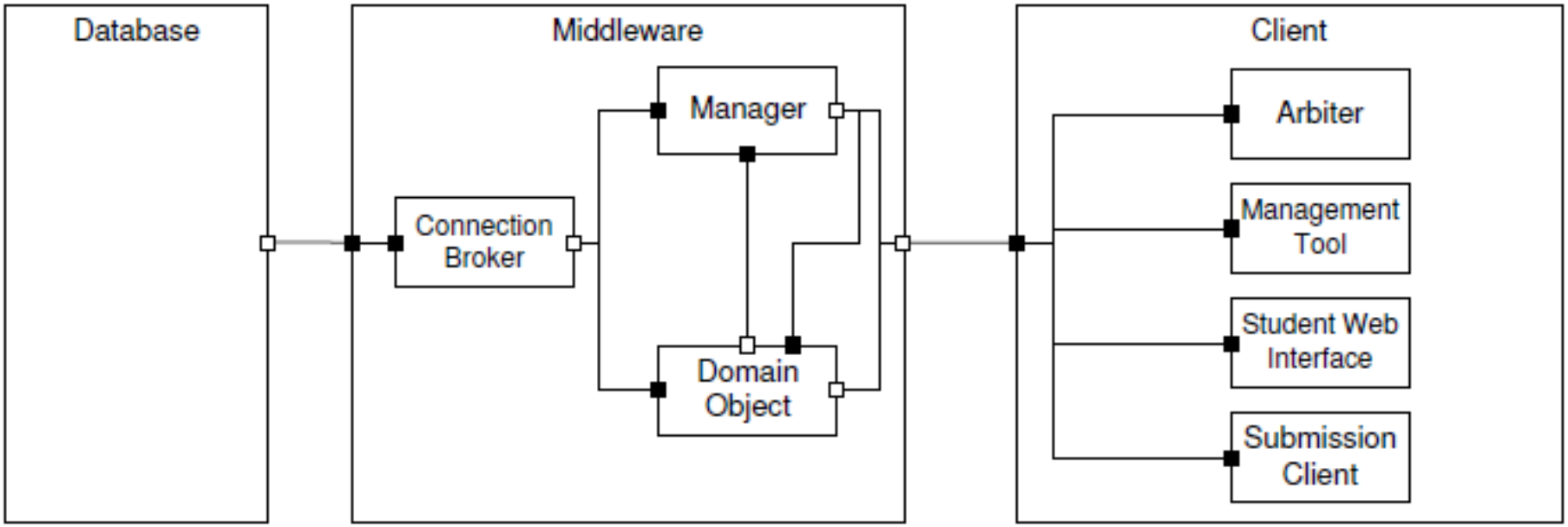


Source:sei.cmu.edu

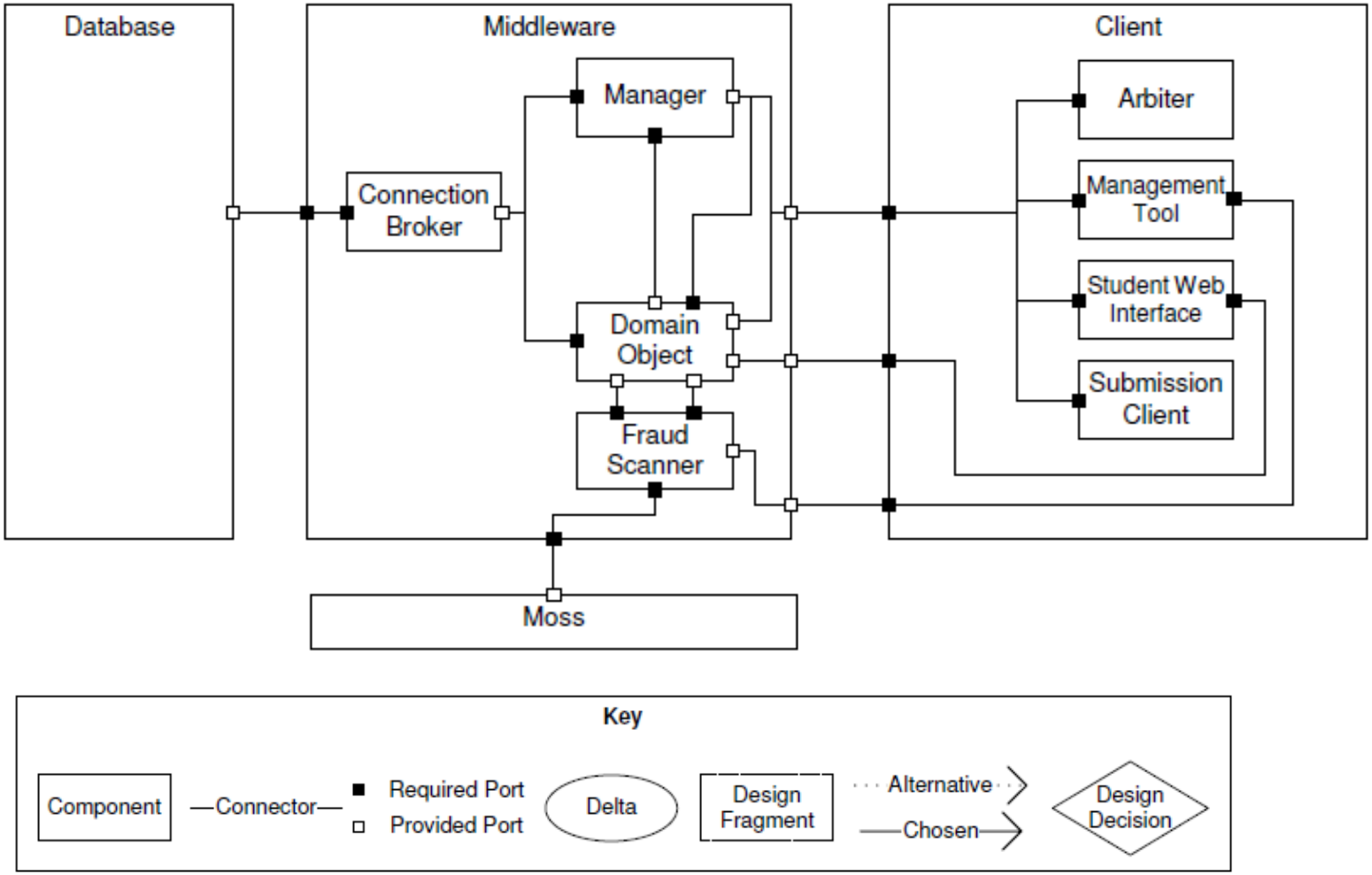


Source:arnon.me

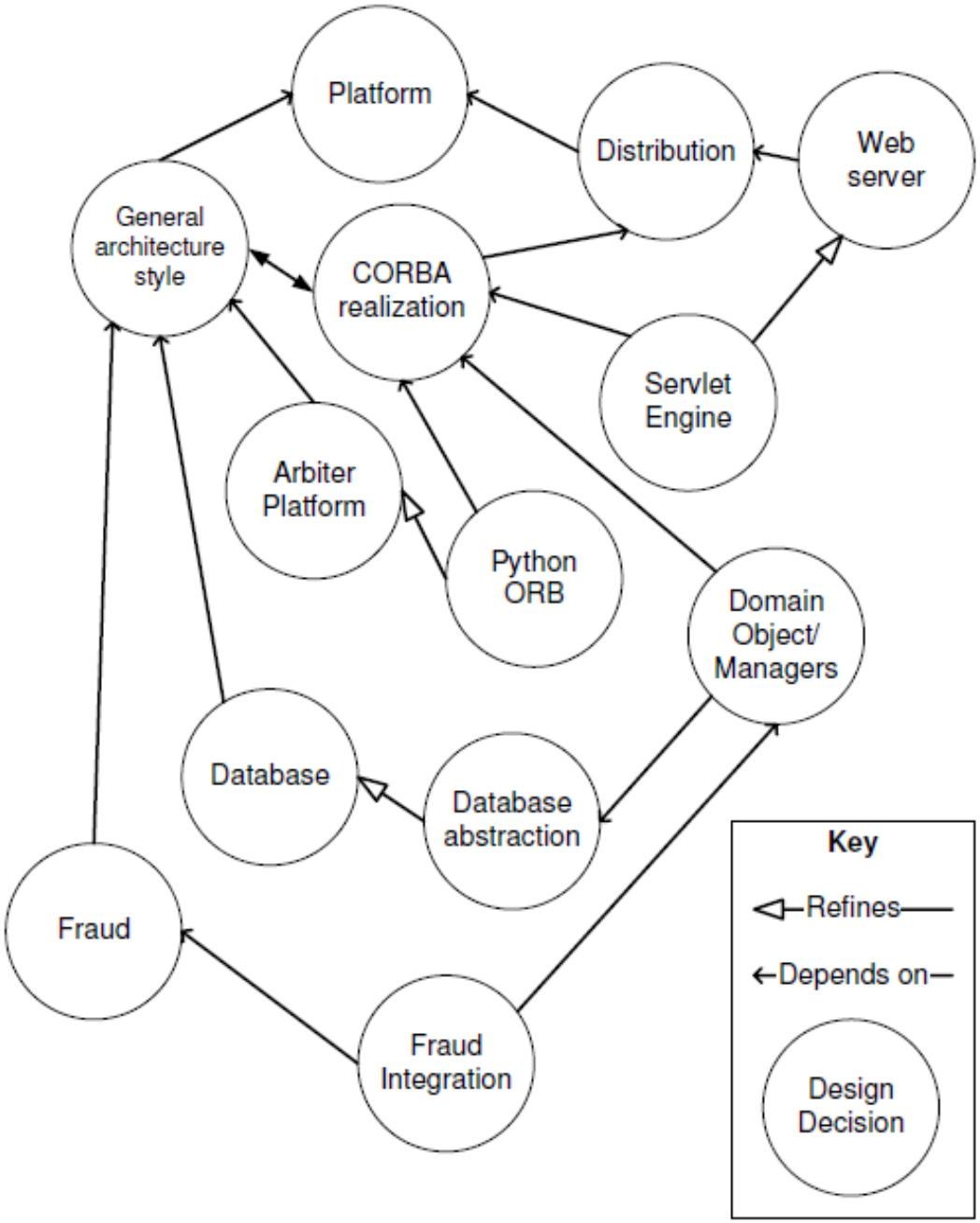
Athena – code review system (original design)



Athena – code review system



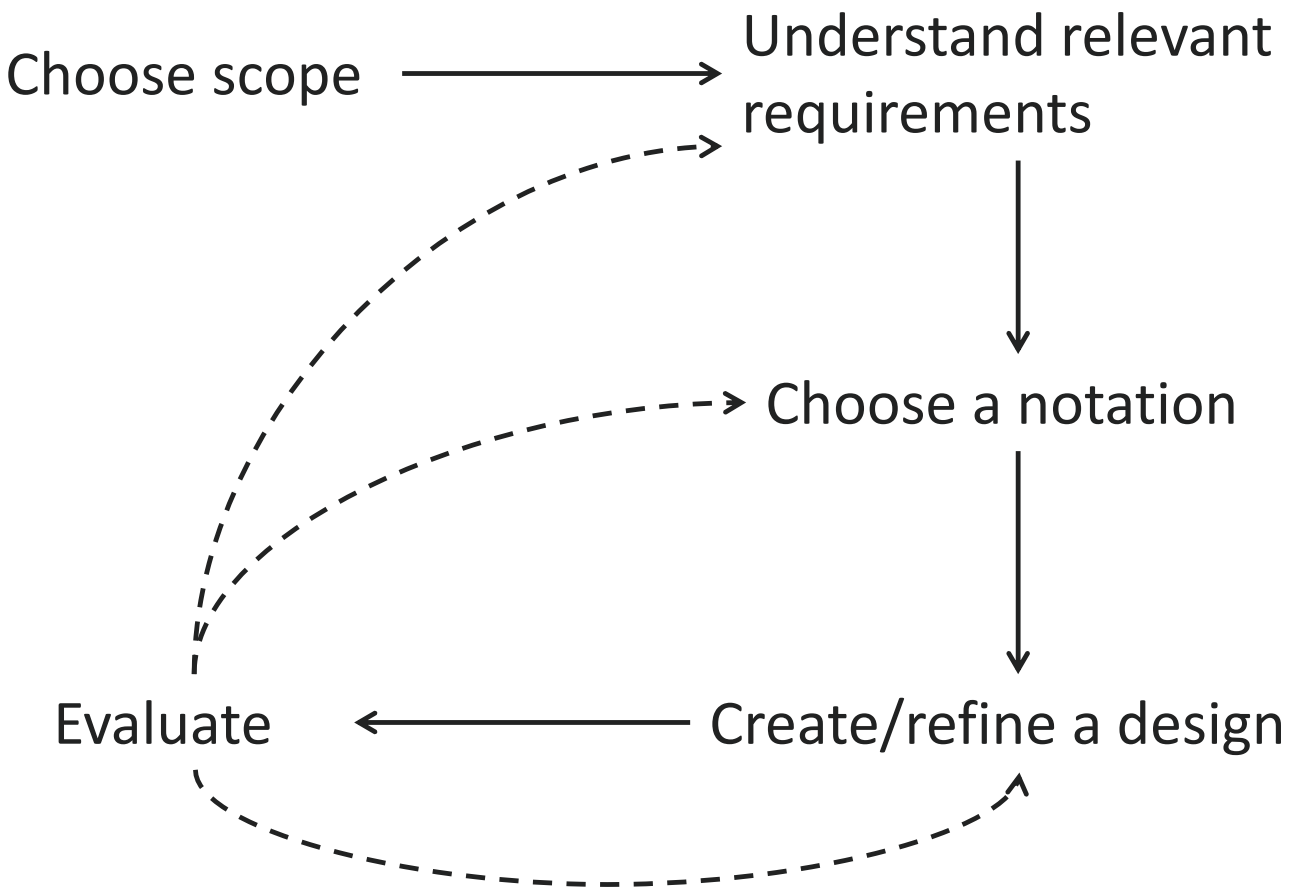
Source: Jansen and Bosch 2005



Source: Jansen and Bosch 2005

institute for SOFTWARE RESEARCH

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

Learning Goals

- Architectural patterns provide common abstractions of system structures with known tradeoffs
- Tactics provide common strategies to achieve quality goals, often useful during design to explicitly discuss alternatives
- Architecture evaluation provides early quality assurance

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.