Foundations of Software Engineering

Lecture 8: Intr. to Software Architecture

Christian Kästner

institute for SOFTWARE RESEARCH

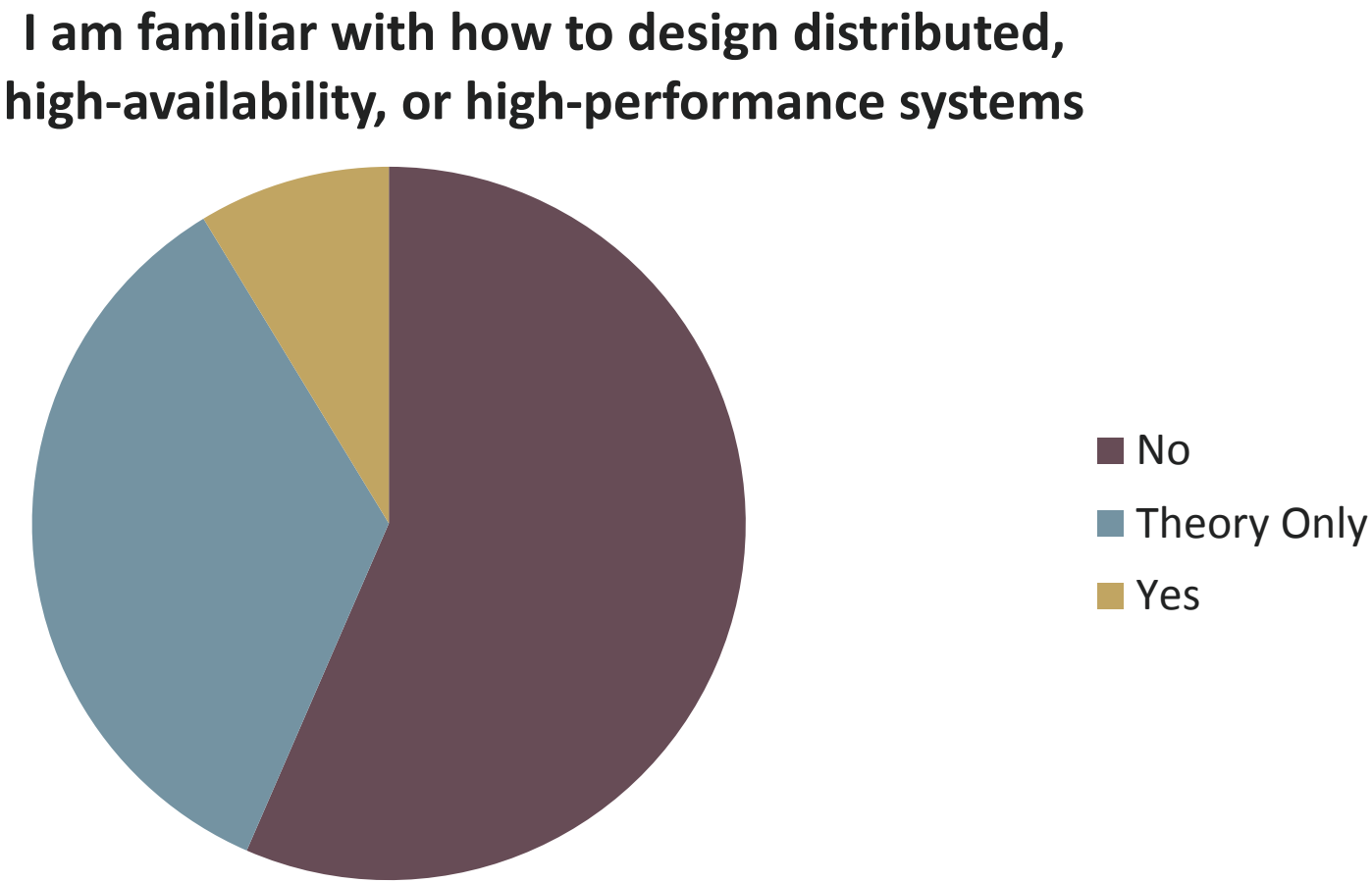
Administrativa

- Homework 1 due tonight
- Teamwork assessment survey
- Homework 2 out now

Learning Goals

- Understand the abstraction level of architectural reasoning
- Approach software architecture with quality attributes in mind
- Distinguish software architecture from (object-oriented) software design

About You


I am familiar with how to design distributed, high-availability, or high-performance systems
No
Theory Only
Yes

Quality Requirements, now what?

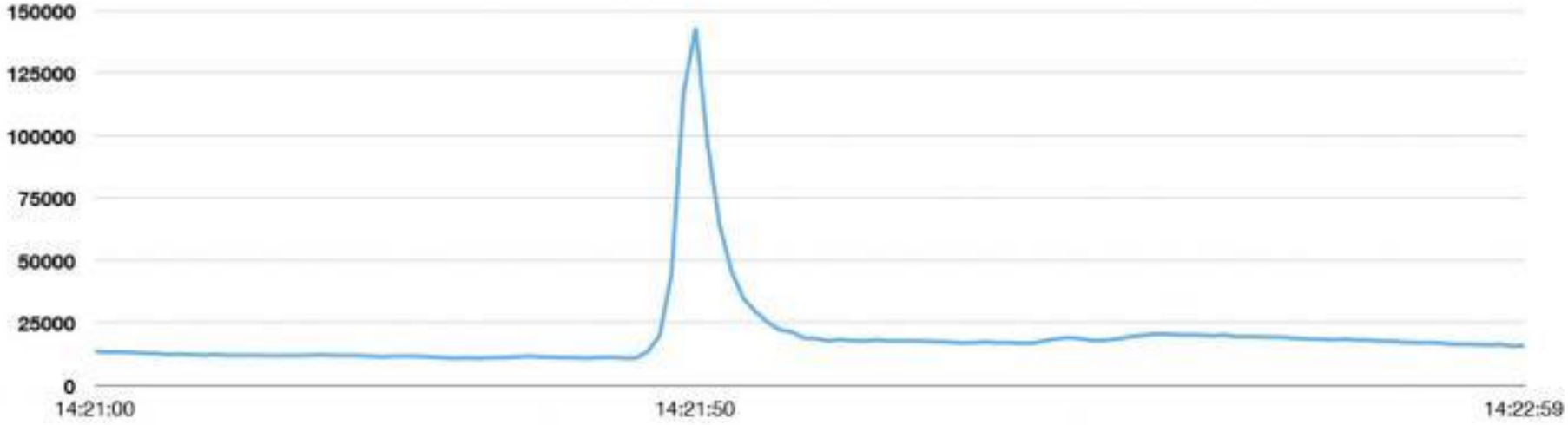
- "should be highly available"
- "should answer quickly, accuracy is less relevant"
- "needs to be extensible"
- "should efficiently use hardware resources"

214 Review

- Design process (analysis, design, implementation)
- Design goals (cohesion, coupling, information hiding, design for reuse, …)
- Design patterns (what they are, for what they are useful, how they are described)
- Frameworks and libraries (reuse strategies)

Case Study:
Architecture Changes at Twitter


血湧き、
天空の城
ラピュタ



On Saturday, August 2 2013 in Japan (11:21:50 p.m. Japan Standard Time), people watched an airing of Castle in the Sky, and at one moment they took to Twitter so much that we hit a one-second peak of 143,199 Tweets per second.

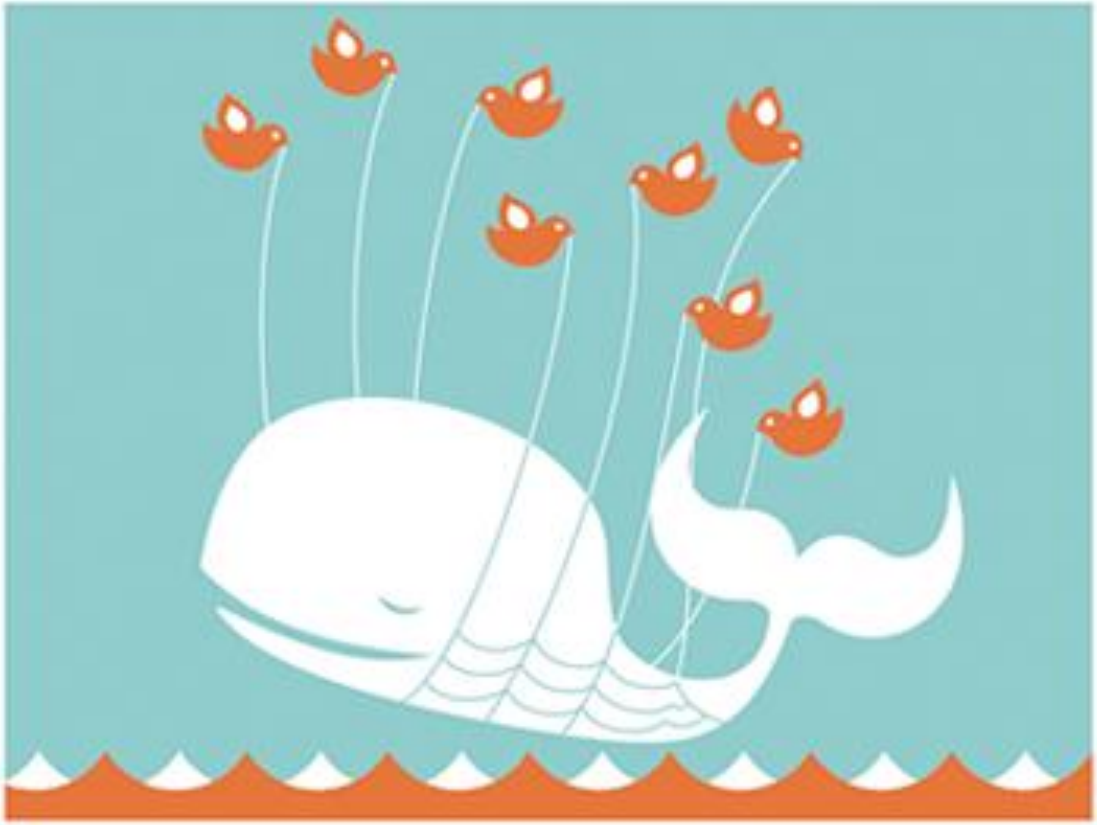
institute for SOFTWARE RESEARCH

twitter

Home Public Timeline Help

Twitter is over capacity.

Too many tweets! Please wait a moment and try again.

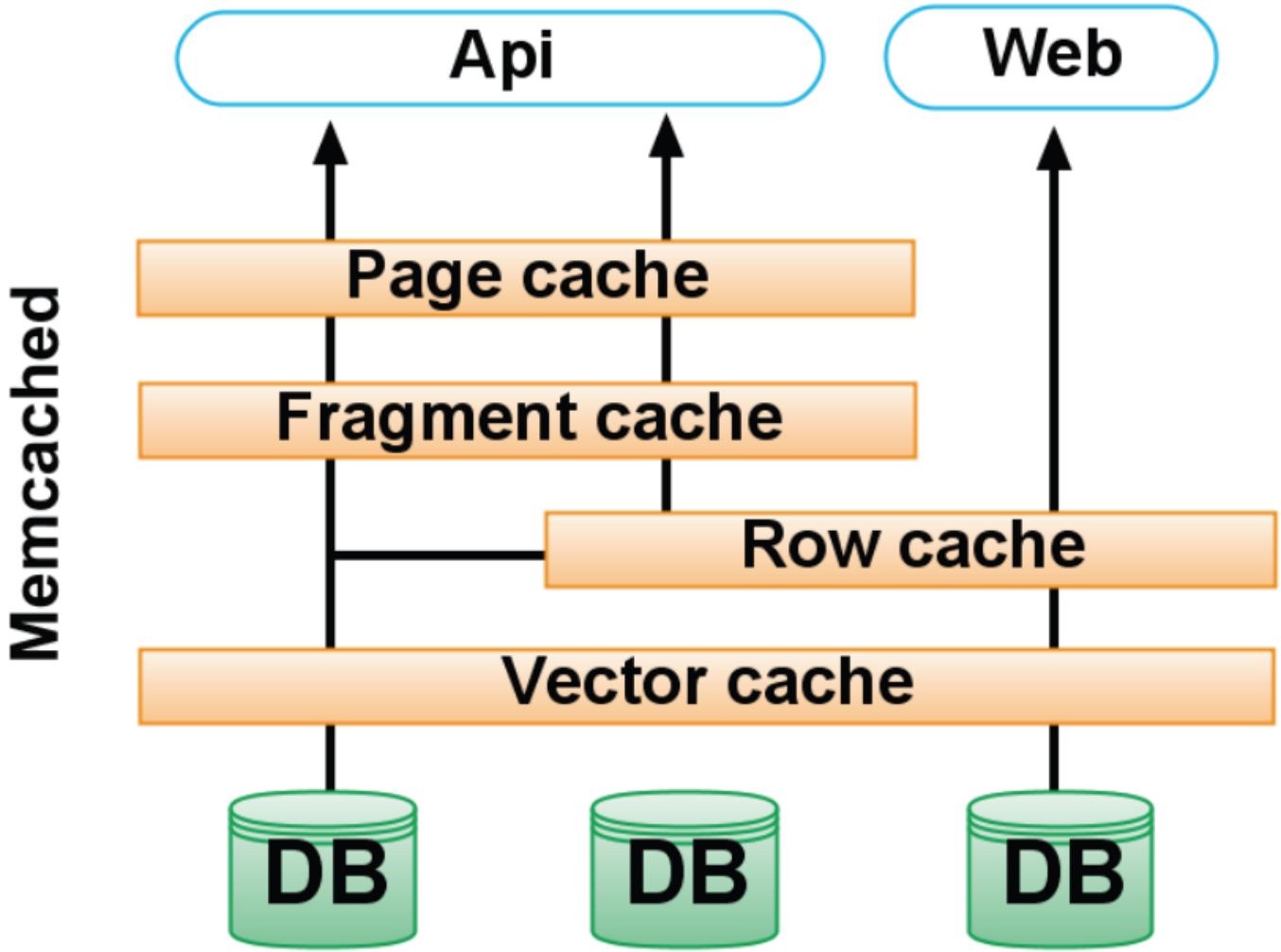


© 2009 Twitter About Us Contact Blog Status API Help Jobs TOS Privacy

Twitter Search (2010)

- Real-time search engine
- Personalized search (depending on searcher's social graph and dynamically evolving resonance signals)
- Scalable to 1000 tweets/sec, 12000 queries/sec
- Latency – time to index a tweet < 0.5 sec
- Latency – time before tweet can be found < 10 sec
- Remove near-duplicate tweets

Caching



Decision to Rearchitect Twitter

"After that experience, we determined we **needed to step back**. We then determined we needed to **re-architect** the site to support the continued growth of Twitter and to keep it running smoothly."

Inspecting the State of Engineering

- Running one of the world's largest Ruby on Rails installations
- 200 engineers
- Monolithic: managing raw database, memcache, rendering the site, and presenting the public APIs in one codebase
- Increasingly difficult to understand system; organizationally challenging to manage and parallelize engineering teams
- Reached the limit of throughput on our storage systems (MySQL); read and write hot spots throughout our databases
- Throwing machines at the problem; low throughput per machine (CPU + RAM limit, network not saturated)
- Optimization corner: trading off code readability vs performance

Redesign Goals

- Improve median latency; lower outliers
- Reduce number of machines 10x
- Isolate failures
- "We wanted cleaner boundaries with “related” logic being in one place"
  - encapsulation and modularity at the systems level (rather than at the class, module, or package level)
- Quicker release of new features
  - "run small and empowered engineering teams that could make local decisions and ship user-facing changes, independent of other teams"

performance

reliability

maintainability

modifiability

JVM vs Ruby VM

- Rails servers capabile of 200-300 requests / sec / host
- Experience with Scala on the JVM; level of trust
- Rewrite for JVM allowed 10-20k requests / sec / host

Programming Model

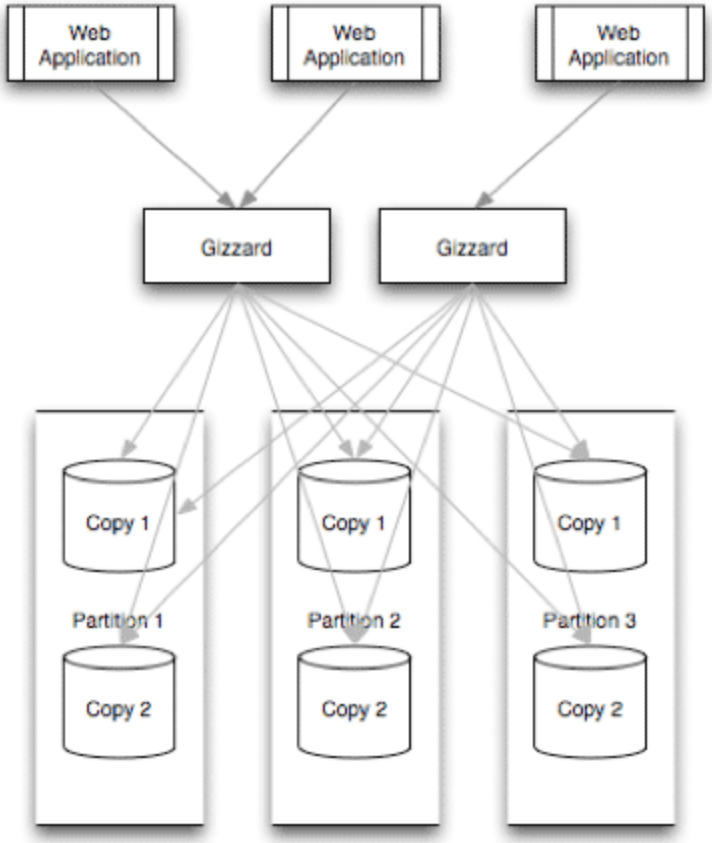
- Ruby model: Concurrency at process level; request queued to be handled by one process
- Twitter response aggregated from several services – additive response times
- *"As we started to decompose the system into services, each team took slightly different approaches. For example, the failure semantics from clients to services didn't interact well: we had no consistent back-pressure mechanism for servers to signal back to clients and we experienced "thundering herds" from clients aggressively retrying latent services."*
- Goal: Single and uniform way of thinking about concurrency
  - Implemented in a library for RPC (Finagle), connection pooling, failover strategies and load balancing

Independent Systems

- *" In our monolithic world, we either needed experts who understood the entire codebase or clear owners at the module or class level. Sadly, the codebase was getting too large to have global experts and, in practice, having clear owners at the module or class level wasn't working. Our codebase was becoming harder to maintain, and teams constantly spent time going on "archeology digs" to understand certain functionality. Or we'd organize "whale hunting expeditions" to try to understand large scale failures that occurred."*
- From monolithic system to multiple services
  - Agree on RPC interfaces, develop system internals independently
  - Self-contained teams

Storage

- Single-master MySQL database bottleneck despite more modular code
- Temporal clustering
  - Short-term solution
  - Skewed load balance
  - One machine + replications every 3 weeks
- Move to distributed database (Glizzard on MySQL) with "roughly sortable" ids
- Stability over features – using older MySQL version

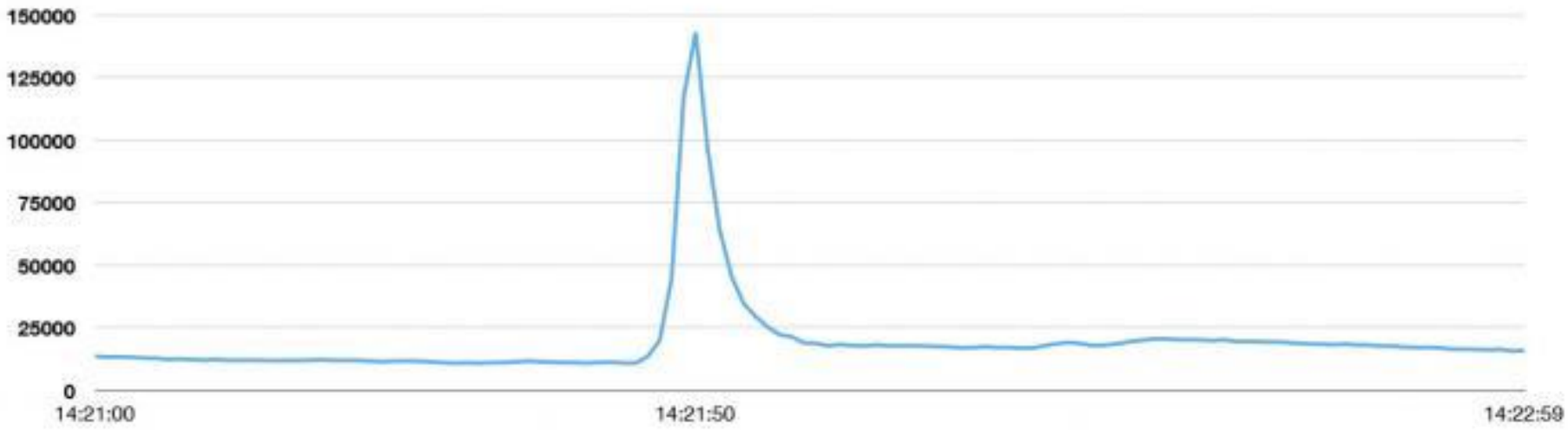


Data-Driven Decisions

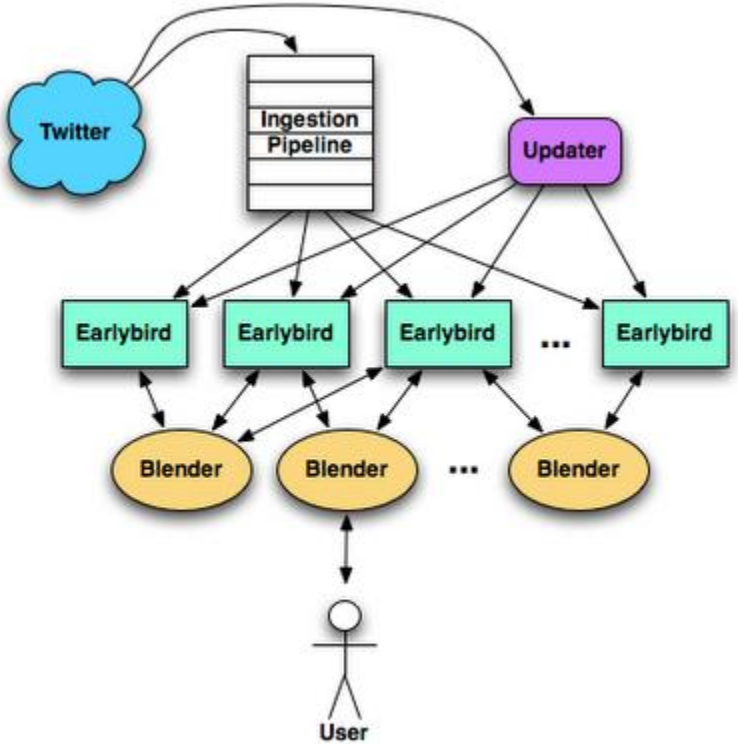
- Many small independent services, number growing
- Own dynamic analysis tool on top of RPC framework
- Framework to configure large numbers of machines
  - Including facility to expose feature to parts of users only

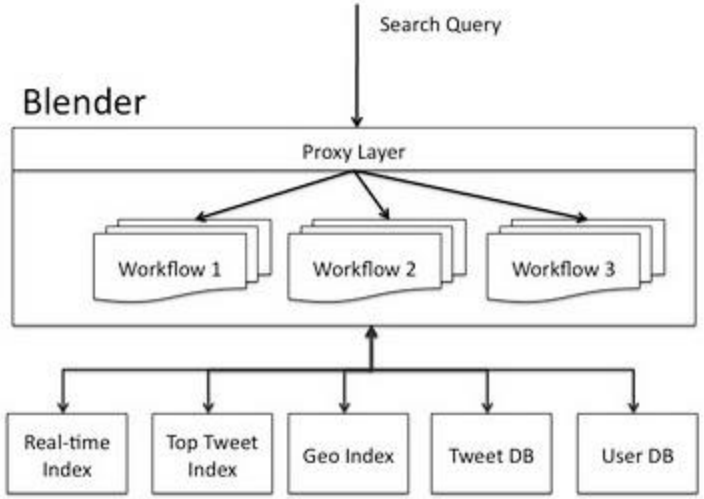
Outcome: Rearchitecting Twitter

"This re-architecture has not only made the service more **resilient when traffic spikes** to record highs, but also provides a more **flexible** platform on which to **build more features faster**, including synchronizing direct messages across devices, Twitter cards that allow Tweets to become richer and contain more content, and a rich search experience that includes stories and users."



On Saturday, August 3 in Japan, people watched an airing of Castle in the Sky, and at one moment they took to Twitter so much that we hit a one-second peak of 143,199 Tweets per second.


Twitter
Ingestion Pipeline
Updater
Earlybird
Earlybird
Earlybird
...
Earlybird
Blender
Blender
...
Blender
User

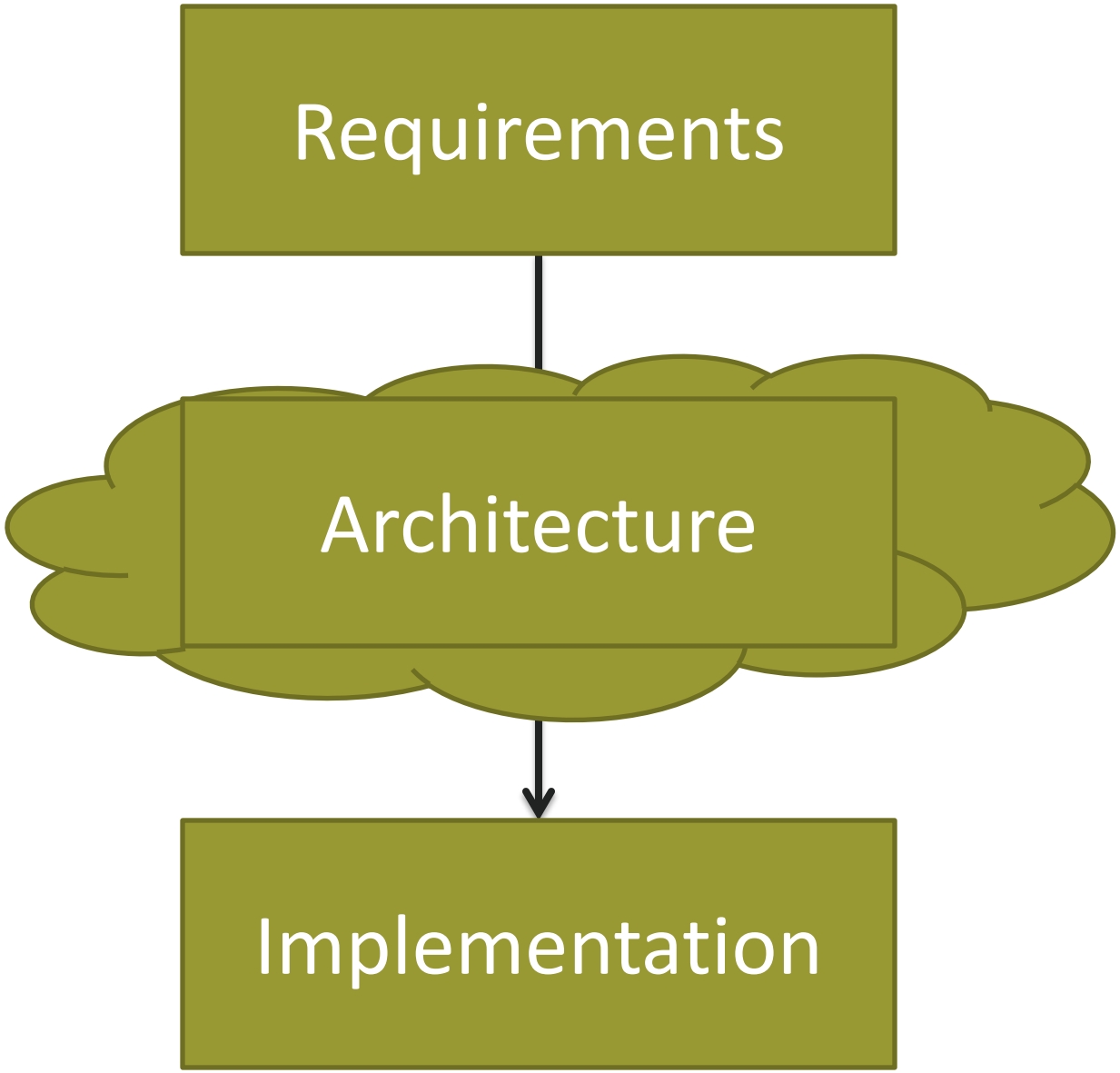

Search Query
Blender
Proxy Layer
Workflow 1
Workflow 2
Workflow 3
Real-time Index
Top Tweet Index
Geo Index
Tweet DB
User DB

Did the original architect make poor decisions?

Key Insights: Twitter Case Study

- Architectural decisions affect entire systems, not only individual modules
- Abstract, different abstractions for different scenarios
- Reason about quality attributes early
- Make architectural decisions explicit

Software Architecture


Requirements
Architecture
Implementation

Beyond functional correctness

- Quality matters, eg.,
  - Availability
  - Modifiability, portability
  - Performance, scalability
  - Security
  - Testability
  - Usability
  - Cost to build, cost to operate

Software Architecture

"The software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both."

[Clements et al. 2010]

Design vs. Architecture

Design Questions

- How do I add a menu item in Eclipse?
- How can I make it easy to add menu items in Eclipse?
- What lock protects this data?
- How does Google rank pages?
- What encoder should I use for secure communication?
- What is the interface between objects?

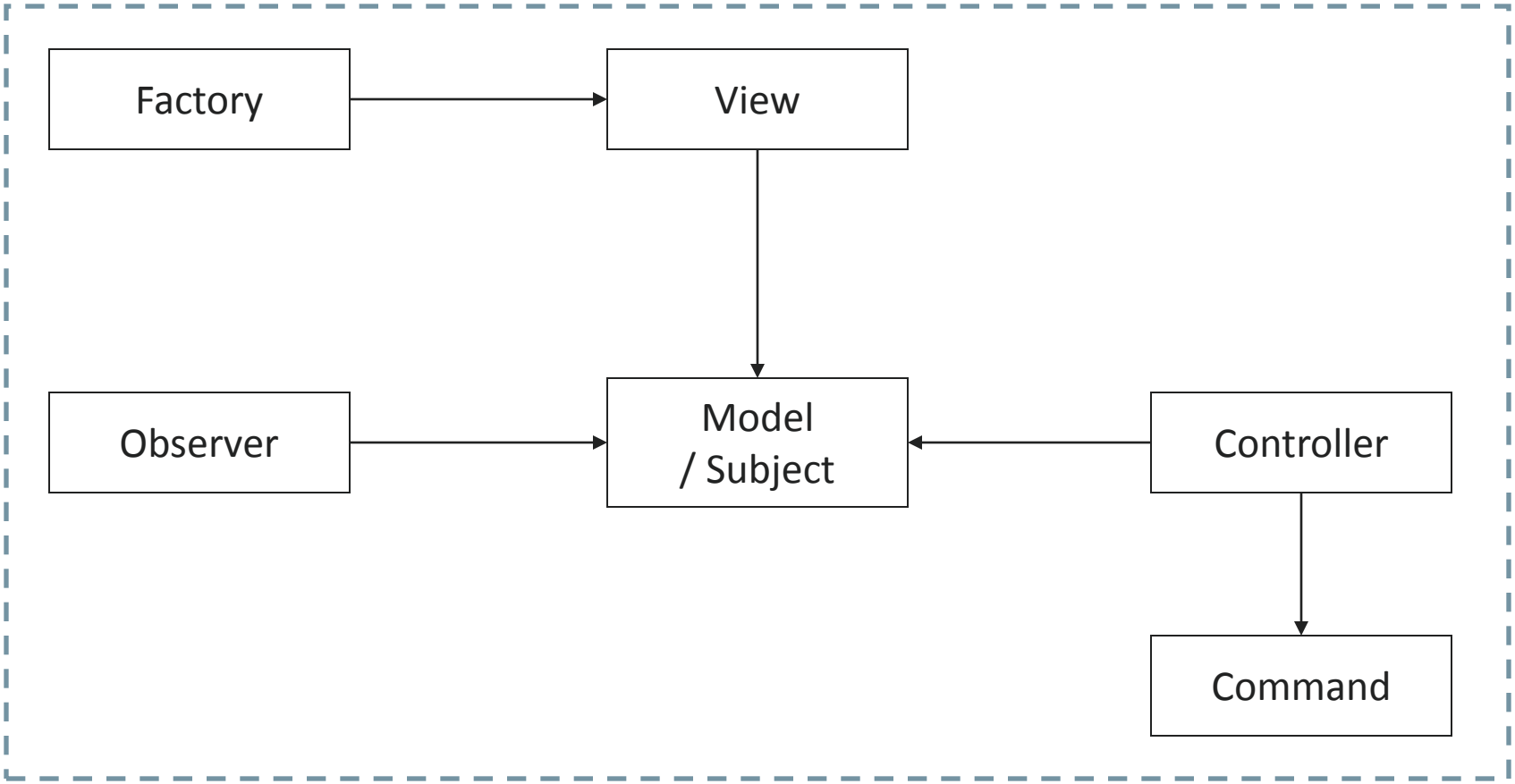
Architectural Questions

- How do I extend Eclipse with a plugin?
- What threads exist and how do they coordinate?
- How does Google scale to billions of hits per day?
- Where should I put my firewalls?
- What is the interface between subsystems?

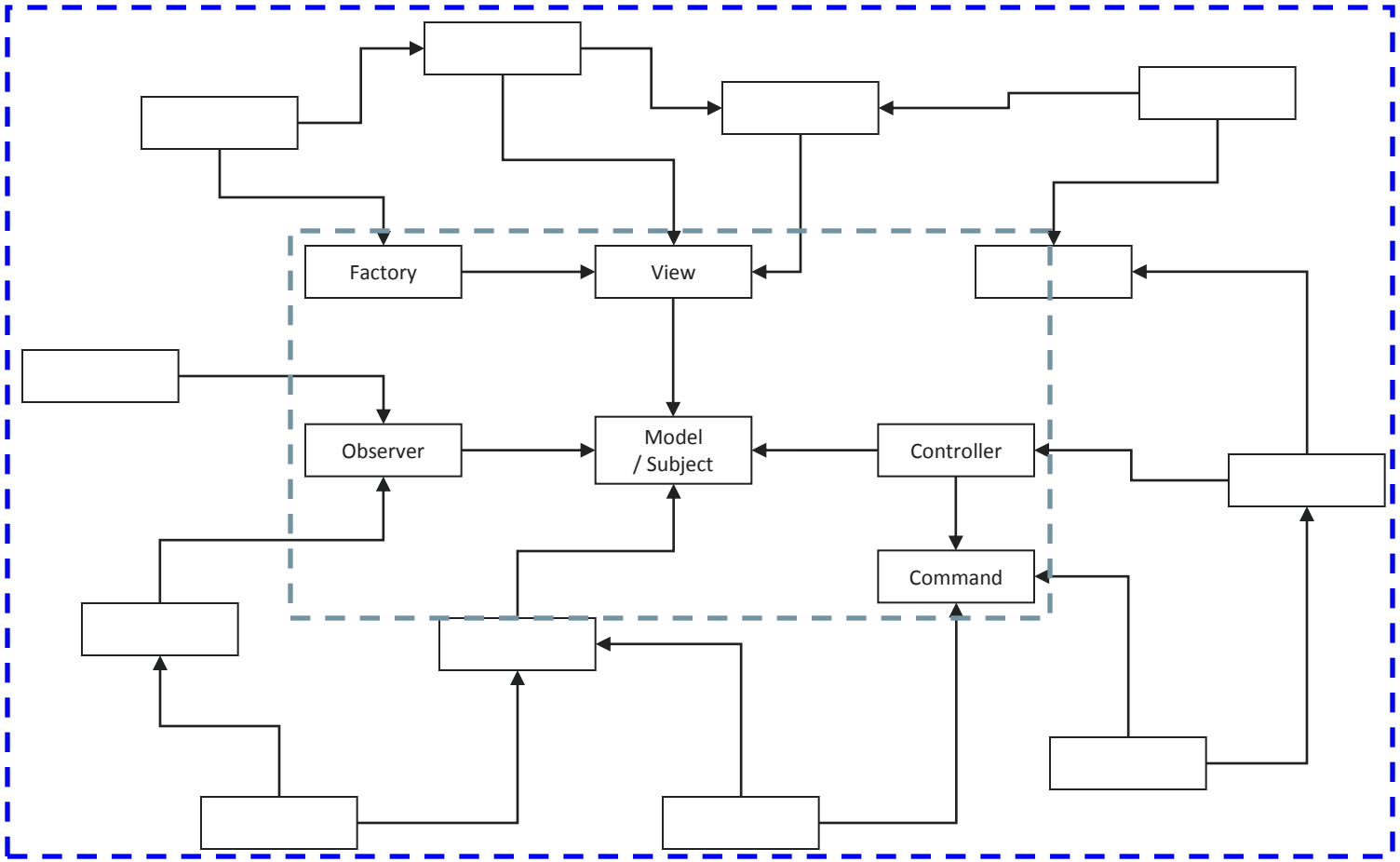
Objects

Model

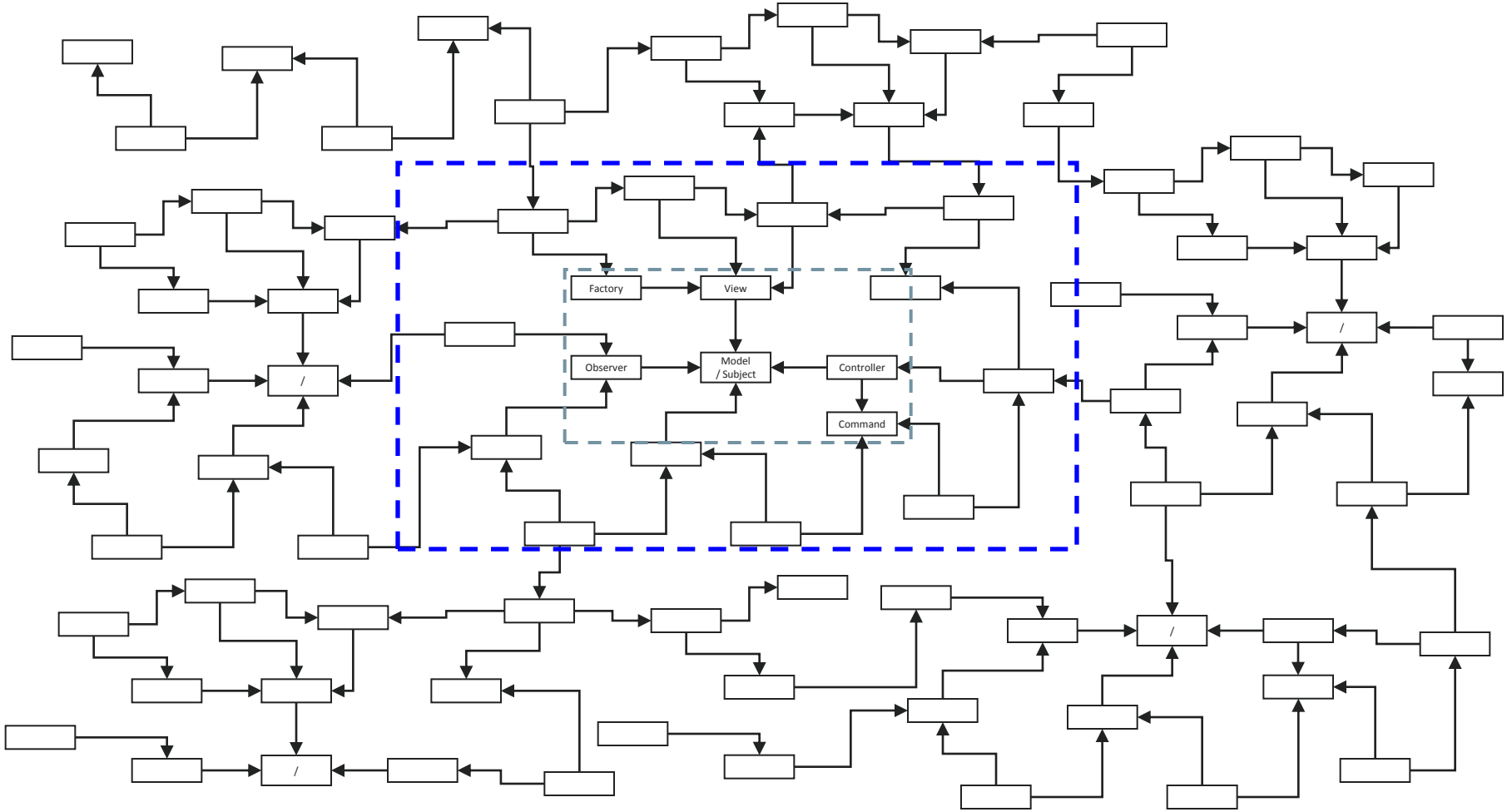
Design Patterns



Design Patterns

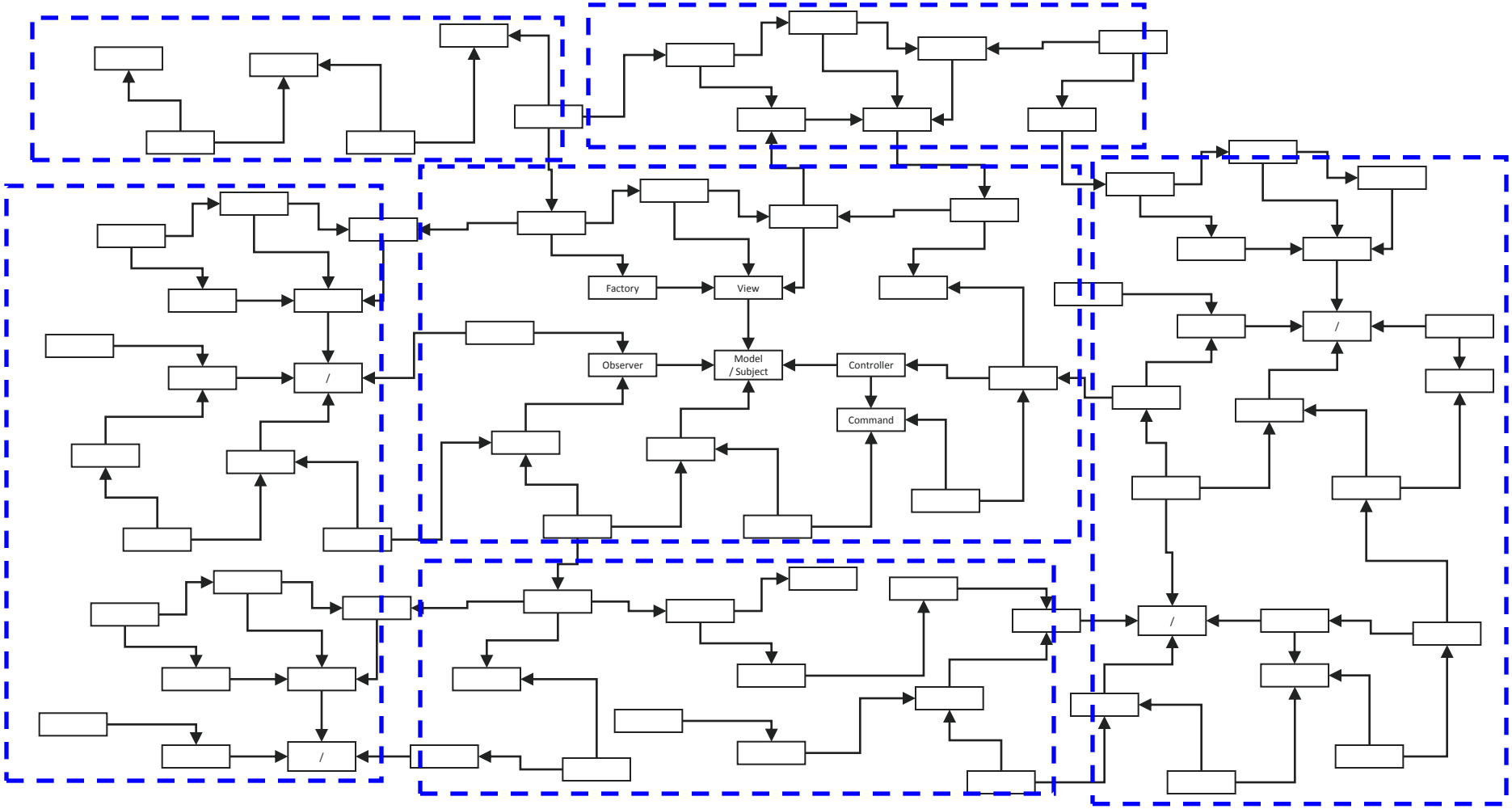


Design Patterns



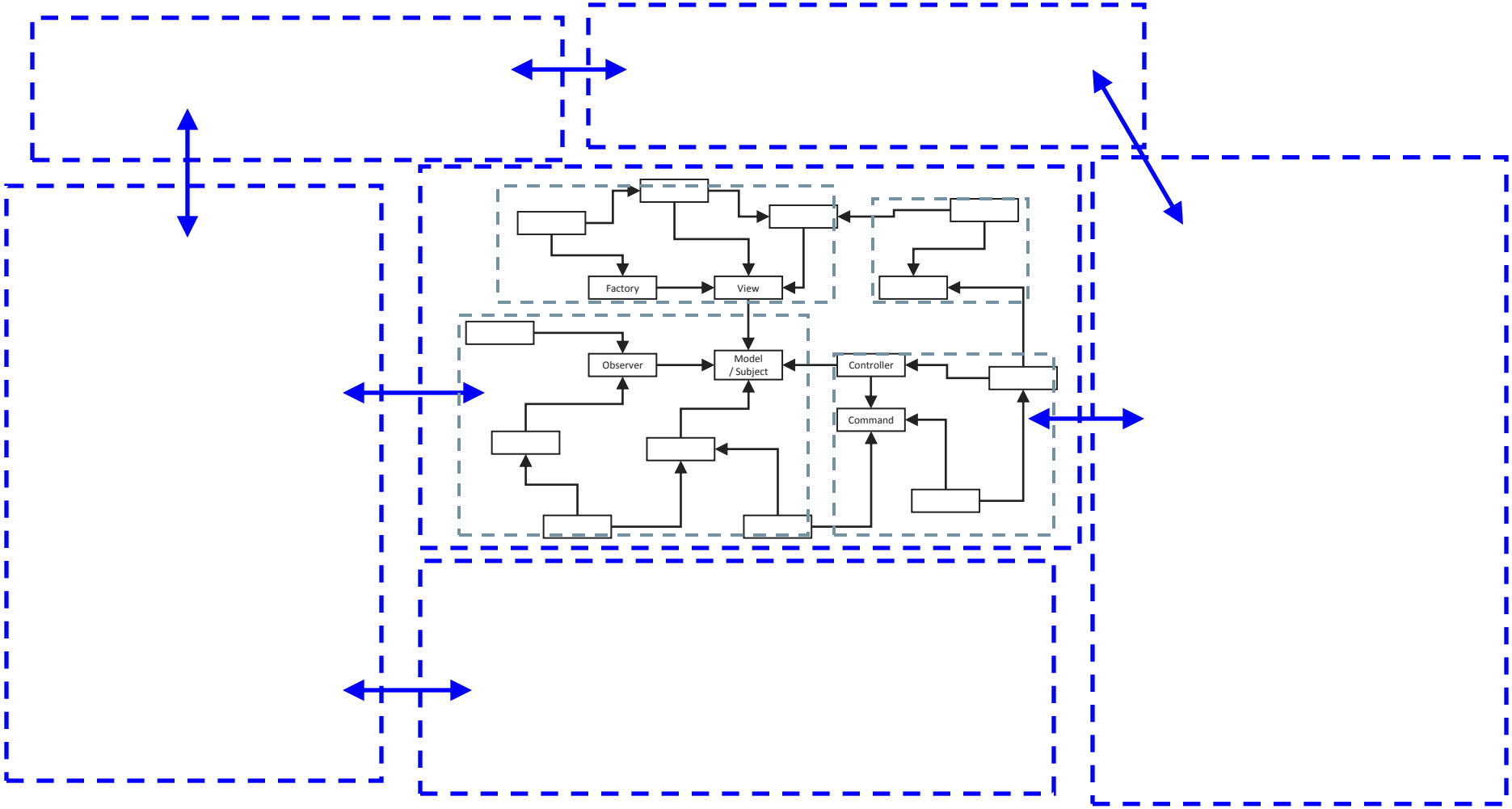
institute for SOFTWARE RESEARCH

Architecture



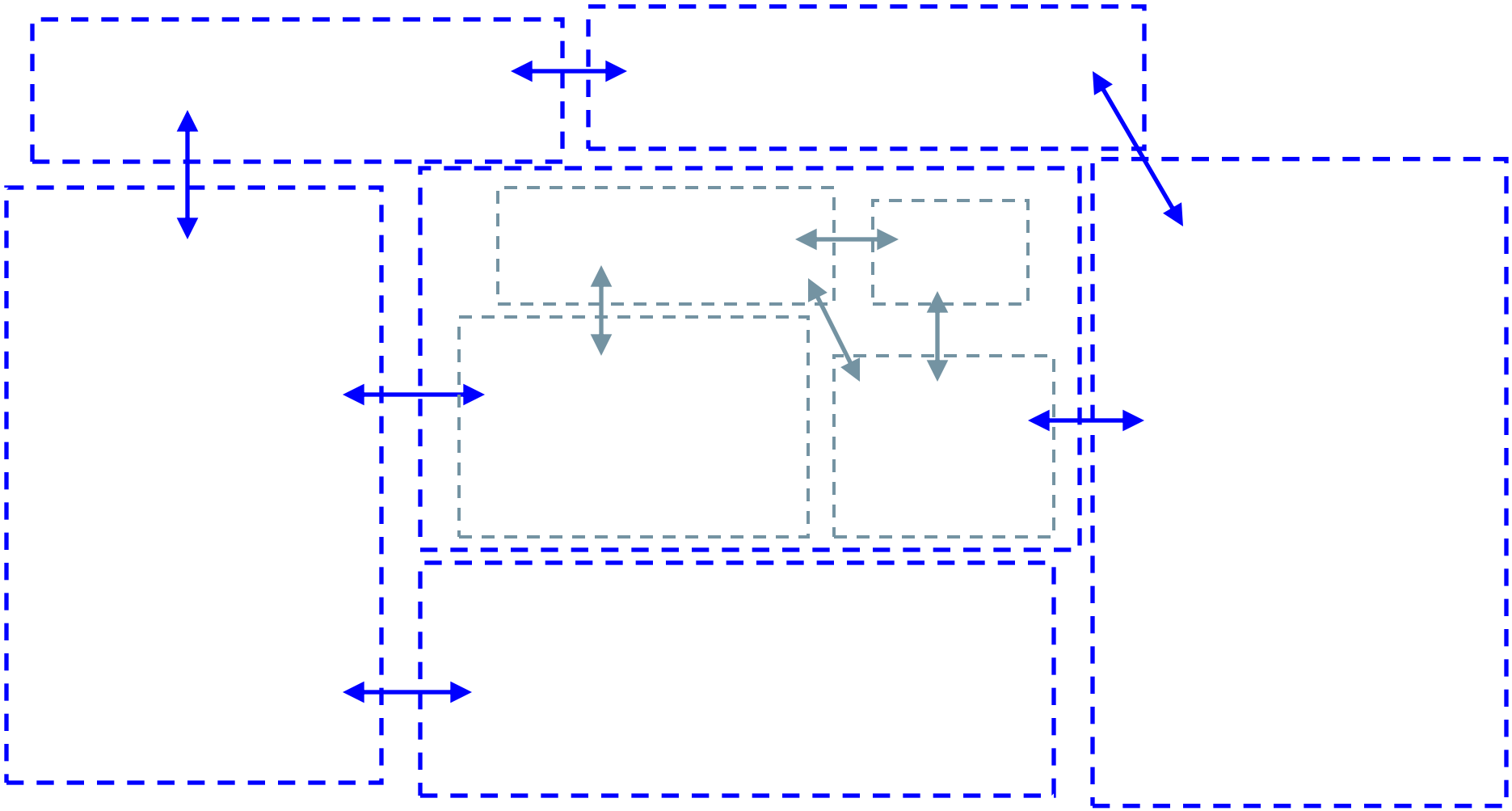
institute for SOFTWARE RESEARCH

Architecture



institute for SOFTWARE RESEARCH

Architecture



institute for SOFTWARE RESEARCH

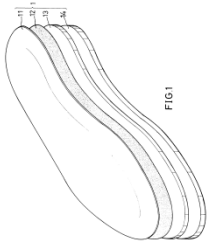
Levels of Abstraction

- 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”

Architecture Disentangled

Architecture as
structures and relations
(the actual system)

Architecture as
documentation
(representations of the system)



Architecture as (design) process
(activities around the other two)

institute for SOFTWARE RESEARCH

Why Architecture

Benefits of Architecture [BCK03]

- Aids in communication with stakeholders
  - Shows them "how" at a level they can understand, raising questions about whether it meets their needs
- Defines constraints on implementation
  - Design decisions form "load-bearing walls" of application
- Dictates organizational structure
  - Teams work on different components
- Inhibits or enables quality attributes
  - Similar to design patterns
- Supports predicting cost, quality, and schedule
  - Typically by predicting information for each component
- Aids in software evolution
  - Reason about cost, design, and effect of changes
- Aids in prototyping
  - Can implement architectural skeleton early

Architectural Drivers

- Functional requirements
  - What the system is supposed to do
- Quality attributes
  - How the system does what it does
- Technical constraints
  - Design decisions you have to work around
    - e.g. Google has 4 approved programming languages
    - e.g. Must use platform X to interoperate with legacy code
- Business constraints
  - e.g. must deliver the product in time for Christmas
- Quality attributes typically drive the architecture
  - But allocation of functionality, technical and business constraints cannot be ignored

Case Study: The Google File System

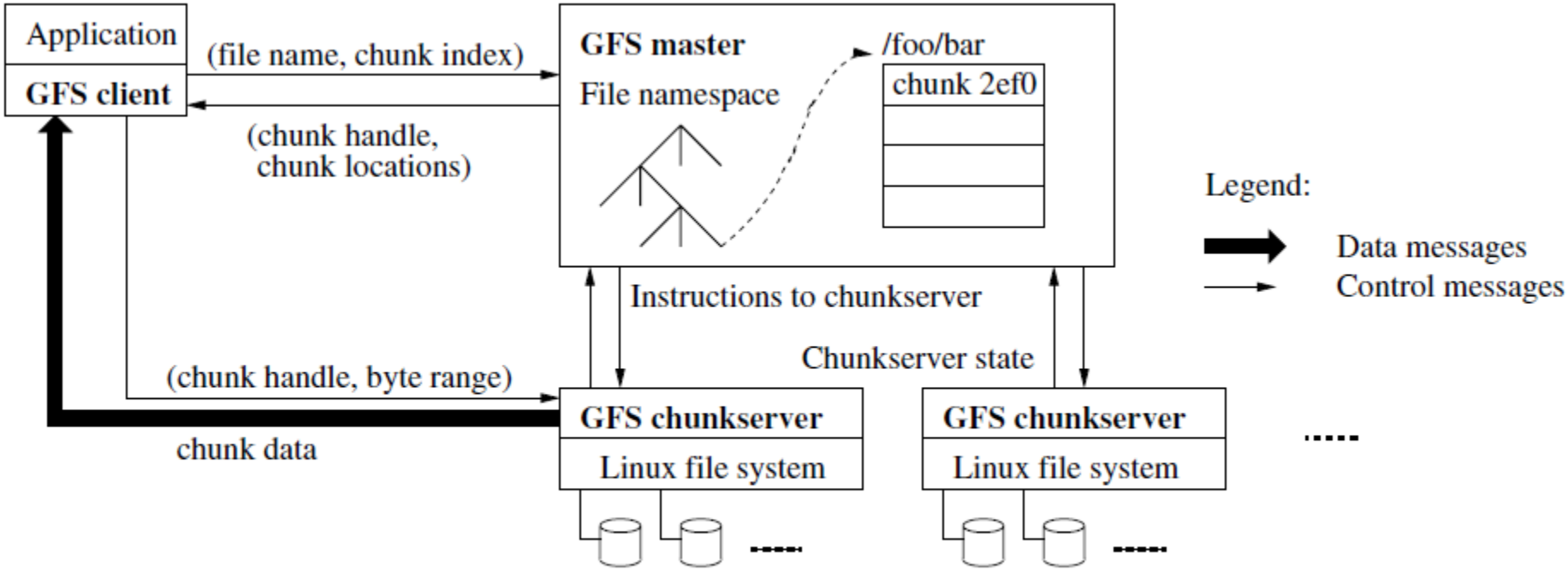


Figure 1: GFS Architecture

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

Questions

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

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.

Case Study: Robotics Controller

Summary: Architecture

- High-level design of parts and their connections
- Linked to requirements, especially quality requirements
- Abstracts from implementation specifics

Further Readings

- Sommerville. Software Engineering. Edition 7/8, Chapters 11-13
- Bass, Clements, and Kazman. Software Architecture in Practice. Addison-Wesley, 2003.
- Fairbanks. Just Enough Software Architecture. Marshall & Brainerd, 2010.
- Taylor, Medvidovic, and Dashofy. Software Architecture: Foundations, Theory, and Practice. Wiley, 2009.