# System Architecture Design

*Robert Kausch, Software Architect, iCR*

*9/9/2024*

ICR
Ideas, Commitment, Results

# Architecture

- Architecture is a constant series of tradeoffs
- The goal is to optimize a design for a set of criteria
- There is no one perfect architecture or style
- The architecture of a system is the underlying structure
- Can be thought of as the blueprint or roadmap of the system

# Architecture

ICR
Ideas, Commitment, Results

- Designing software is similar to designing and building a house
- You don’t draw the exterior of the house and start building it
  - This is how we learn to write software!
- You have to work through all of the individual subsystems and components, like the electrical, plumbing, roofing, structural, lighting, etc. before beginning to build
  - These individual disciplines may not know anything about each other, but it is up to the architect to produce a unified plan

# Software Architecture Definition

- The shared understanding that the expert developers have of the system design *(Fowler, 2019)*
- The software architecture of a system represents the design decisions related to overall system structure and behavior. Architecture helps stakeholders understand and analyze how the system will achieve essential qualities such as modifiability, availability, and security. *(Software Architecture | Software Engineering Institute, n.d.)*

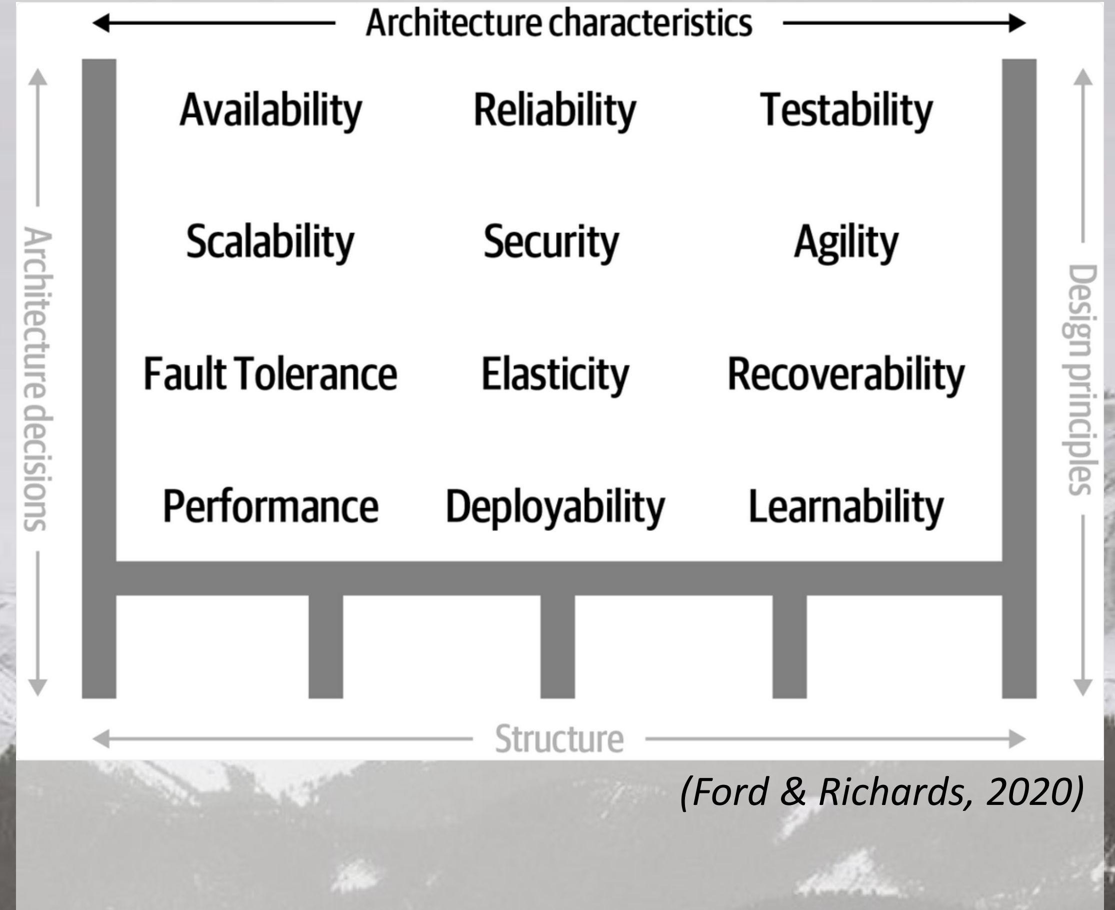
ICR
Ideas, Commitment, Results

# Why do we care about Architecture?

- Properly designing a system's architecture helps achieve target goals
- Informs the various engineering disciplines how to build, validate, operate, and maintain the system
- Exposes key design decisions early in the process
- Informs stakeholders throughout the product lifecycle

# Architecture Characteristics

- There are many styles of architectures
- Each are stronger or weaker for these characteristics
- The chosen style is a trade-off, sacrificing less important characteristics for more important ones
- When designing, understand which characteristics are most important



*(Ford & Richards, 2020)*

iCR
Ideas, Commitment, Results

# Role of a Software Architect

- ***<u>Communicate</u>*** – with customer, team, leadership, other stakeholders
- Model and design systems
- Develop prototype solutions
- Analyze and Perform trade-offs
- Set team standards, e.g.: code quality, security, etc.
- Conduct code, architecture reviews
- Collaborate with, and mentor engineering teams
- Document and disseminate design and decisions
- Differs from development with a broad rather than deep knowledgebase

# Phases of Project Design

## Greenfield

- Starting a new project from scratch
- Not all constraints are well known
- Less common

## Refactoring

- Evolving or adapting an existing system
- Better understanding of constraints after deployment
- Most common type of development

# Front-End Architectural Styles

- Installed Application – *Traditional application installed locally, may not have a separate back-end at all*

# Front-End Architectural Styles

- Installed Application – *Traditional application installed locally, may not have a separate back-end at all*
- Static Website – *a simple website, with mostly static content, interacts with a backend server to provide functionality*

iCR
Ideas, Commitment, Results

# Front-End Architectural Styles

- Installed Application – *Traditional application installed locally, may not have a separate back-end at all*
- Static Website – *a simple website, with mostly static content, interacts with a backend server to provide functionality*
- Web Application – *a multi-page application with dynamic content, interacting with a backend*

# Front-End Architectural Styles

- Installed Application – *Traditional application installed locally, may not have a separate back-end at all*
- Static Website – *a simple website, with mostly static content, interacts with a backend server to provide functionality*
- Web Application – *a multi-page application with dynamic content, interacting with a backend*
- Single Page App (SPA) – *a large and possibly complex application, similar to an installed application, but runs completely in the browser*

# Front-End Architectural Styles

- Installed Application – *Traditional application installed locally, may not have a separate back-end at all*
- Static Website – *a simple website, with mostly static content, interacts with a backend server to provide functionality*
- Web Application – *a multi-page application with dynamic content, interacting with a backend*
- Single Page App (SPA) – *a large and possibly complex application, similar to an installed application, but runs completely in the browser*
- Micro Front End – *independent components that compose either a single page app, or web application*

# Back-end Architectural Styles

- Many styles, broadly categorized as two main types:
  - Monolithic – single deployment of all code
    - Characterized by tightly coupled units of functionality
    - Layered – units of functionality broken into layers, e.g.: 3 tier architecture
    - Distributed Monolith – large interdependent distributed components
  - Distributed – many independent components connected remotely
    - Loosely coupled, independent components, deployed separately
    - Event / Message driven – communication using events / messages
    - Microservices – small components, loosely coupled, single responsibility

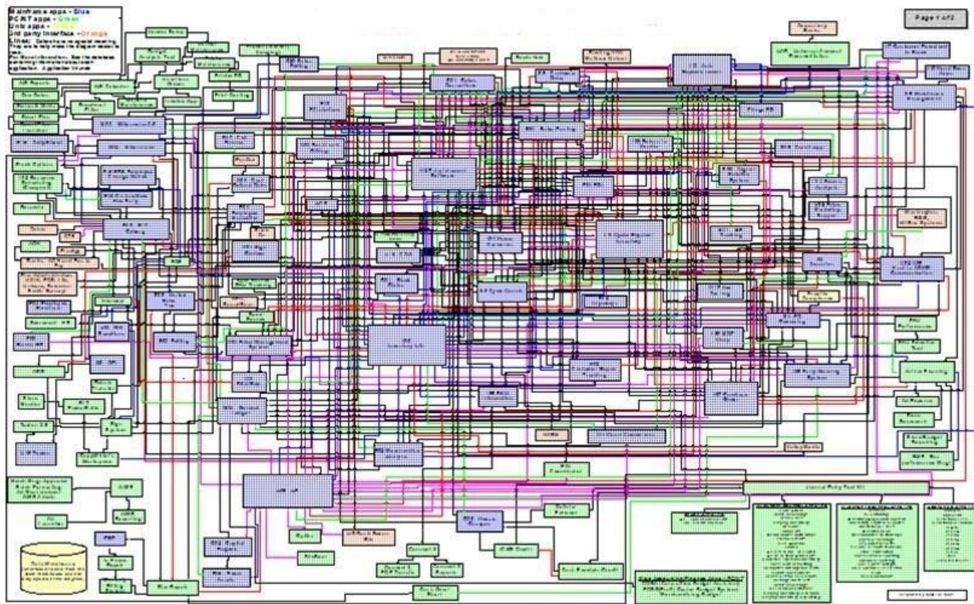
# Architectural Decomposition

- The act of breaking apart an item into smaller pieces
- The human mind is unable to comprehend infinite complexity
- To understand, design, and create complex systems, we use techniques to simplify the problem
- A “Layered Thinking” approach breaks complex problems into layers
- Each layer represents a high-level group of related functionality
- Each layer hides deeper complexity, but can be understood in relation to other high-level layers

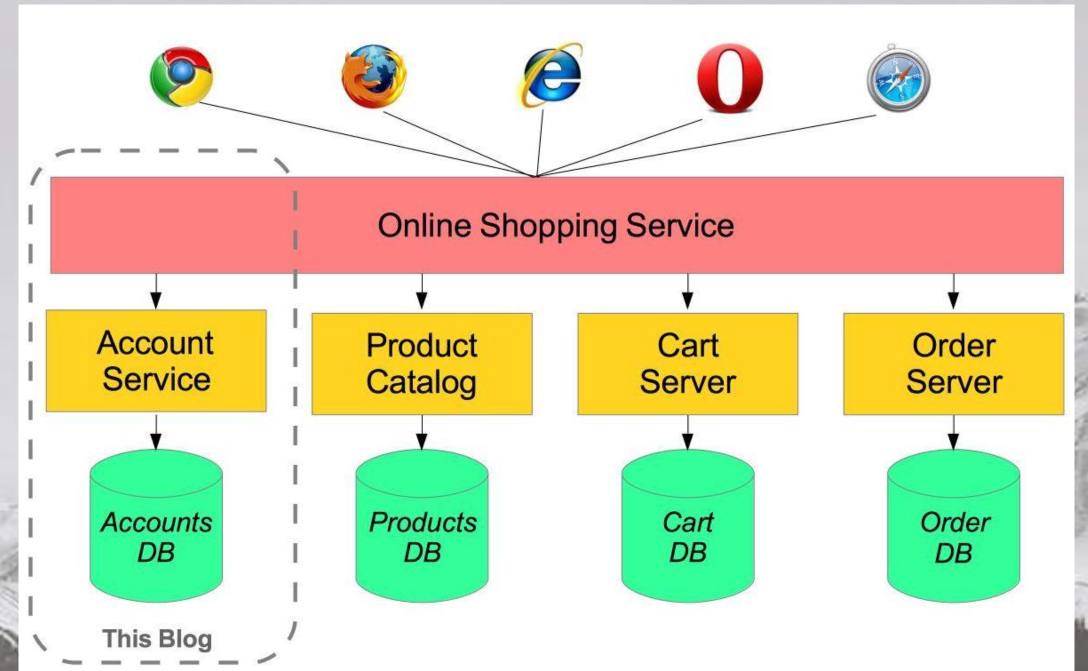
iCR
Ideas, Commitment, Results

# Architectural Decomposition

**Which is easier to understand?**



Jayan, E. (2022, April 17). *How Learning the Software's Architecture can help a Business Analyst?* https://www.linkedin.com/pulse/how-learning-softwares-architecture-can-help-business-eksara-jayan



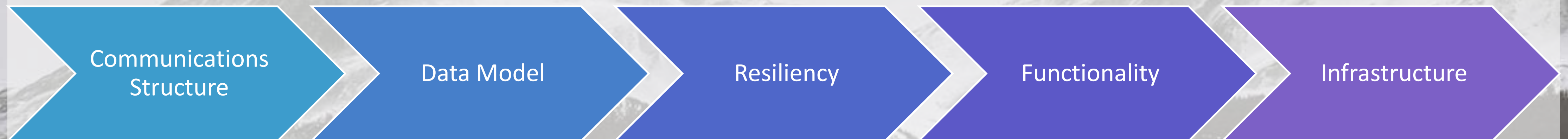

*Microservices with Spring*. (2015, July 14). Microservices With Spring. https://spring.io/blog/2015/07/14/microservices-with-spring

# Example: Autonomous Weather Station

- A customer wants to build an autonomous weather station
- The customer outlines that the system should:
  - Run for long periods without intervention, and accept remote updates
  - Integrate many types of hardware sensors
  - Keep historical data from each sensor
  - Run on off-the-shelf hardware installed in the field
  - Integrate multiple installations into a cohesive network
  - Expose a web interface to stream current readings, and explore historical data
  - Immediately send alerts if extreme readings are detected

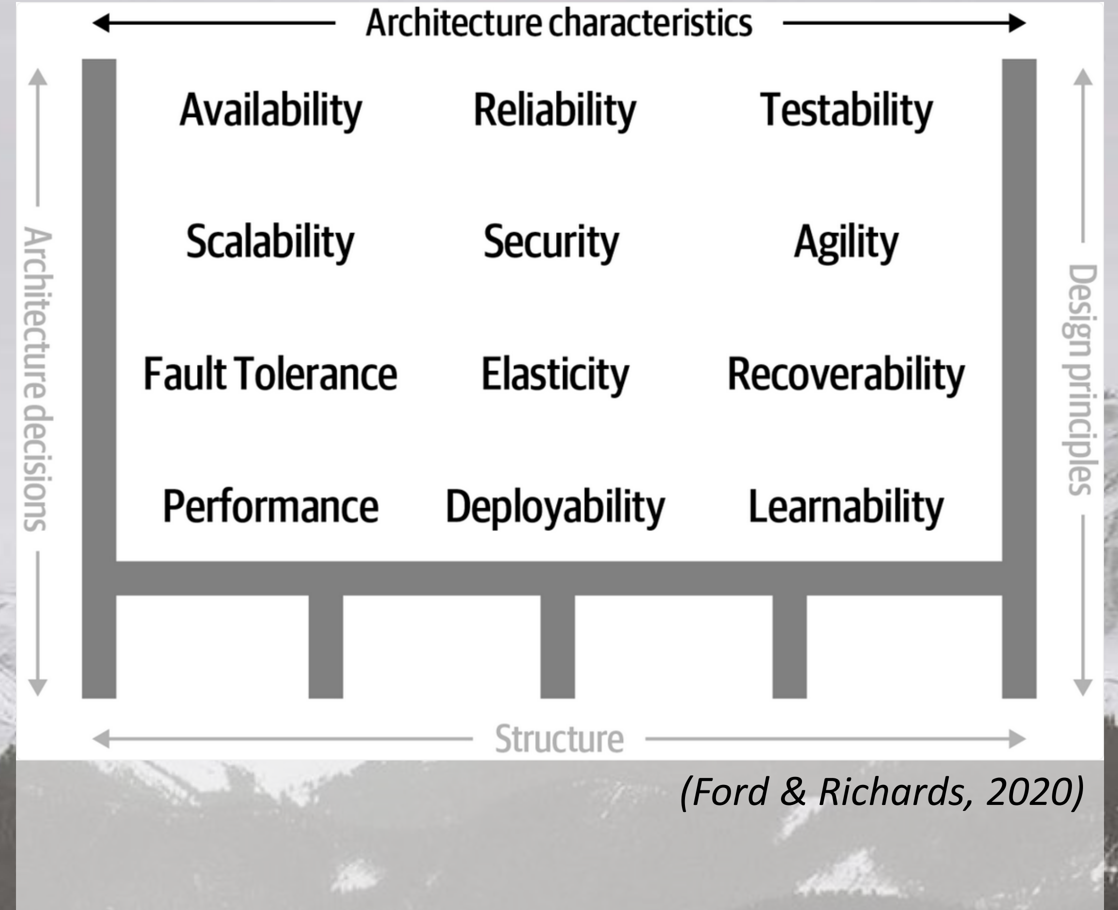
# Where do we start?

- Requirements are vague, but there are important design criteria embedded in them
- First, identify which characteristics are most important
- Then distill concrete criteria, thinking about the system through several lenses:



# Weather Station: Important Criteria

- What is most important?

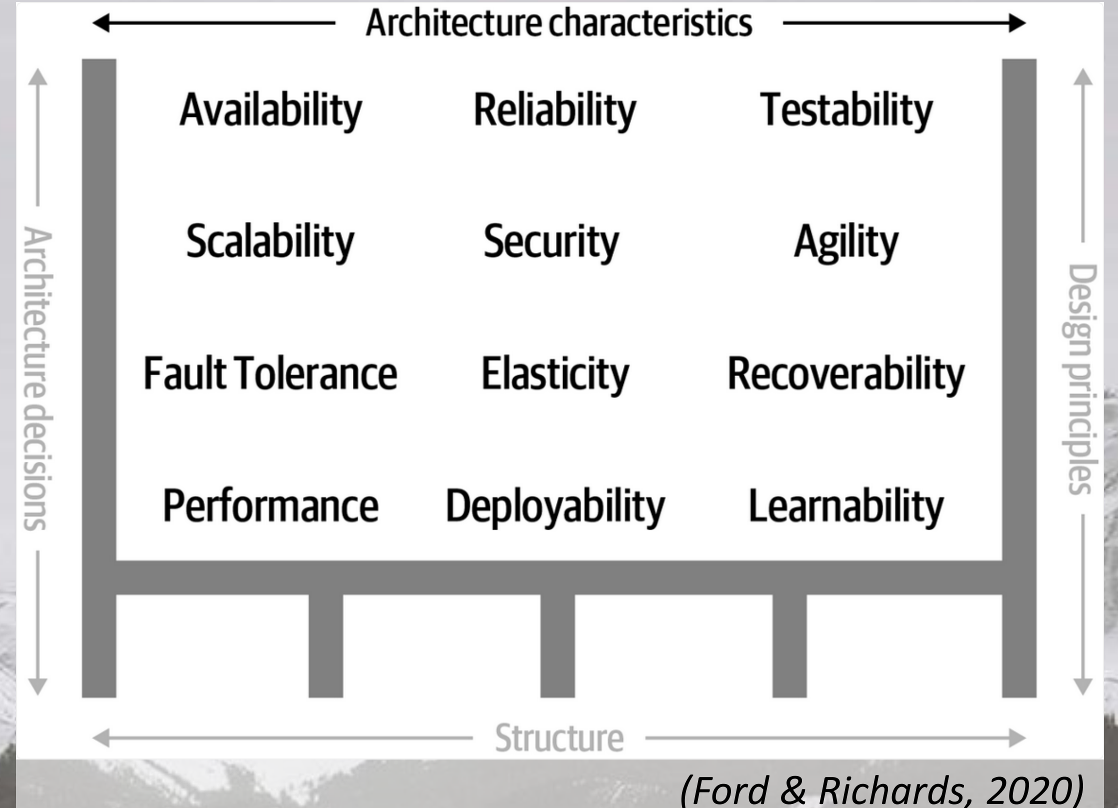



*(Ford & Richards, 2020)*

iCR
Ideas, Commitment, Results

# Weather Station: Important Criteria

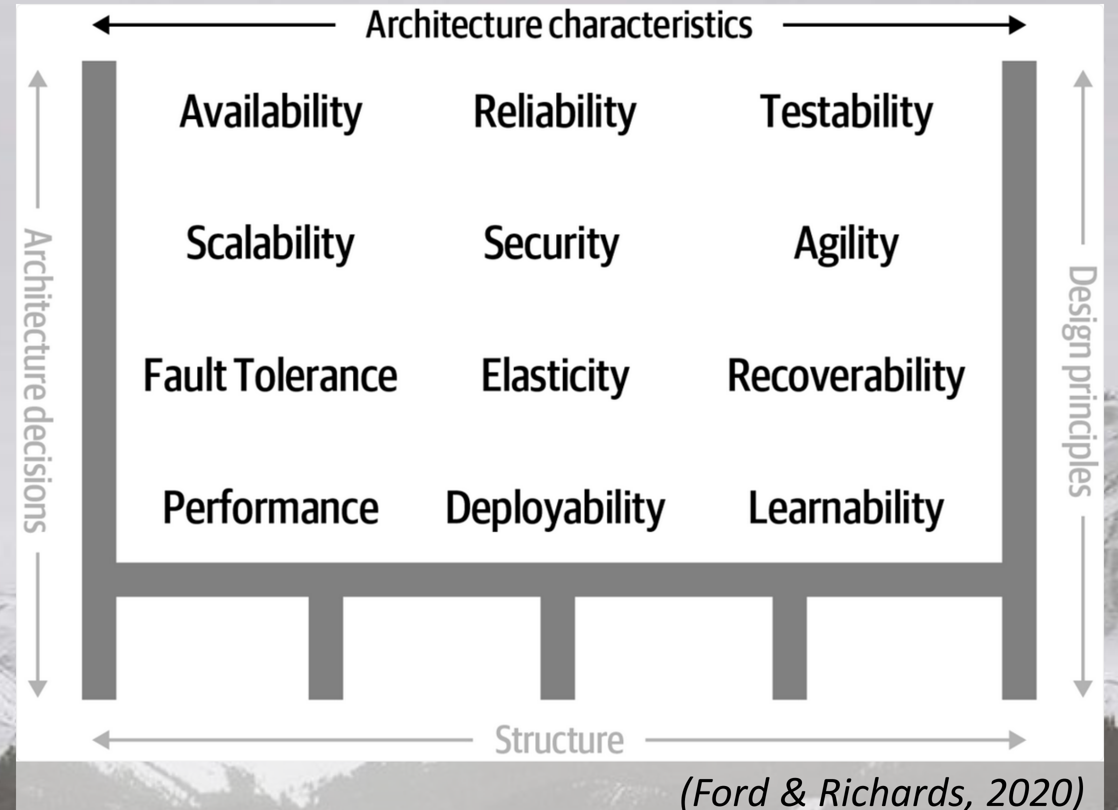
- What is most important?
  - Reliability – sensors should be collecting data all the time




*(Ford & Richards, 2020)*

# Weather Station: Important Criteria

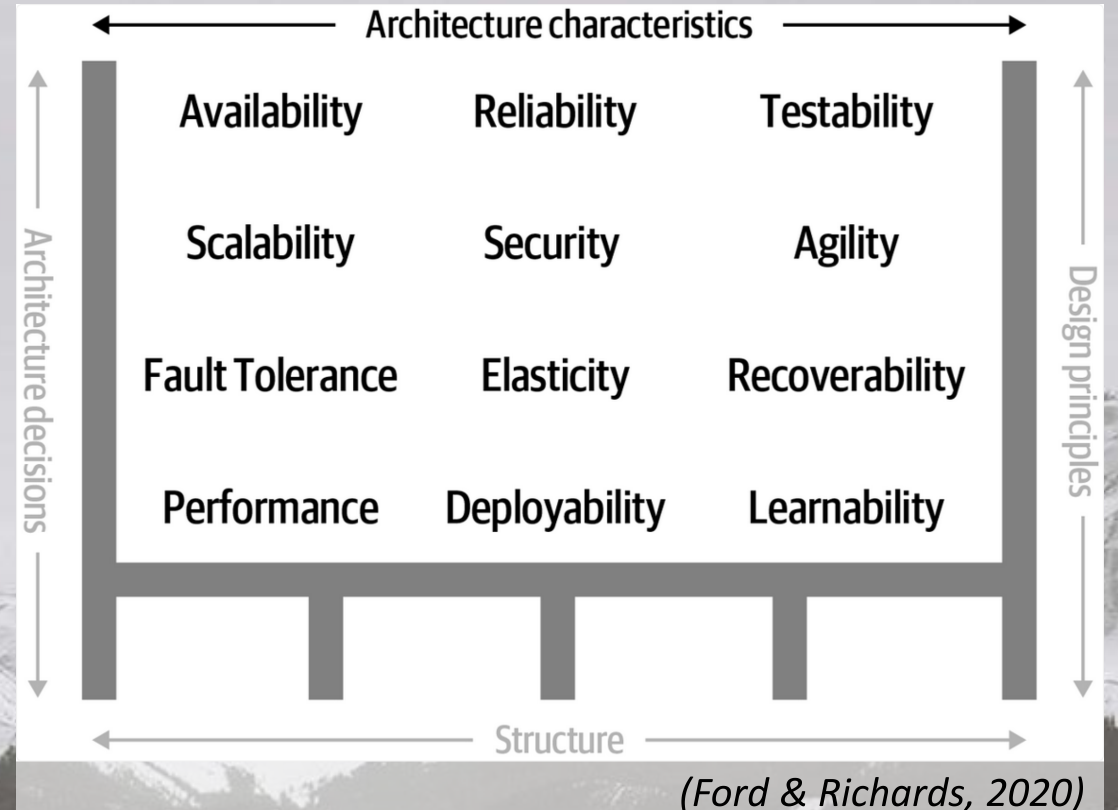
- What is most important?
  - Reliability – sensors should be collecting data all the time
  - Fault Tolerance – Even if parts of the system are unavailable, the sensors should still collect data




*(Ford & Richards, 2020)*

# Weather Station: Important Criteria

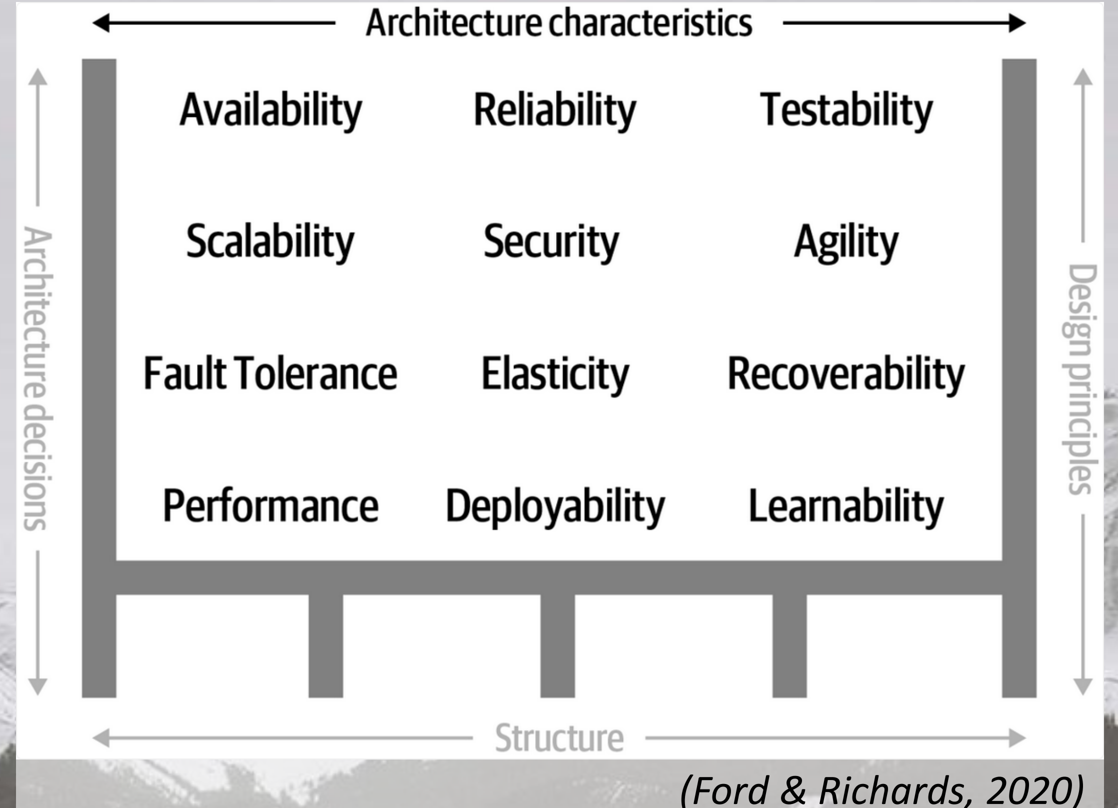
- What is most important?
  - Reliability – sensors should be collecting data all the time
  - Fault Tolerance – Even if parts of the system are unavailable, the sensors should still collect data
  - Deployability – The system must accept remote updates




(Ford & Richards, 2020)

# Weather Station: Important Criteria

- What is most important?
  - Reliability – sensors should be collecting data all the time
  - Fault Tolerance – Even if parts of the system are unavailable, the sensors should still collect data
  - Deployability – The system must accept remote updates
  - Availability – Alerting only works if the system is available




*(Ford & Richards, 2020)*

iCR
Ideas, Commitment, Results

# The Communication Lens

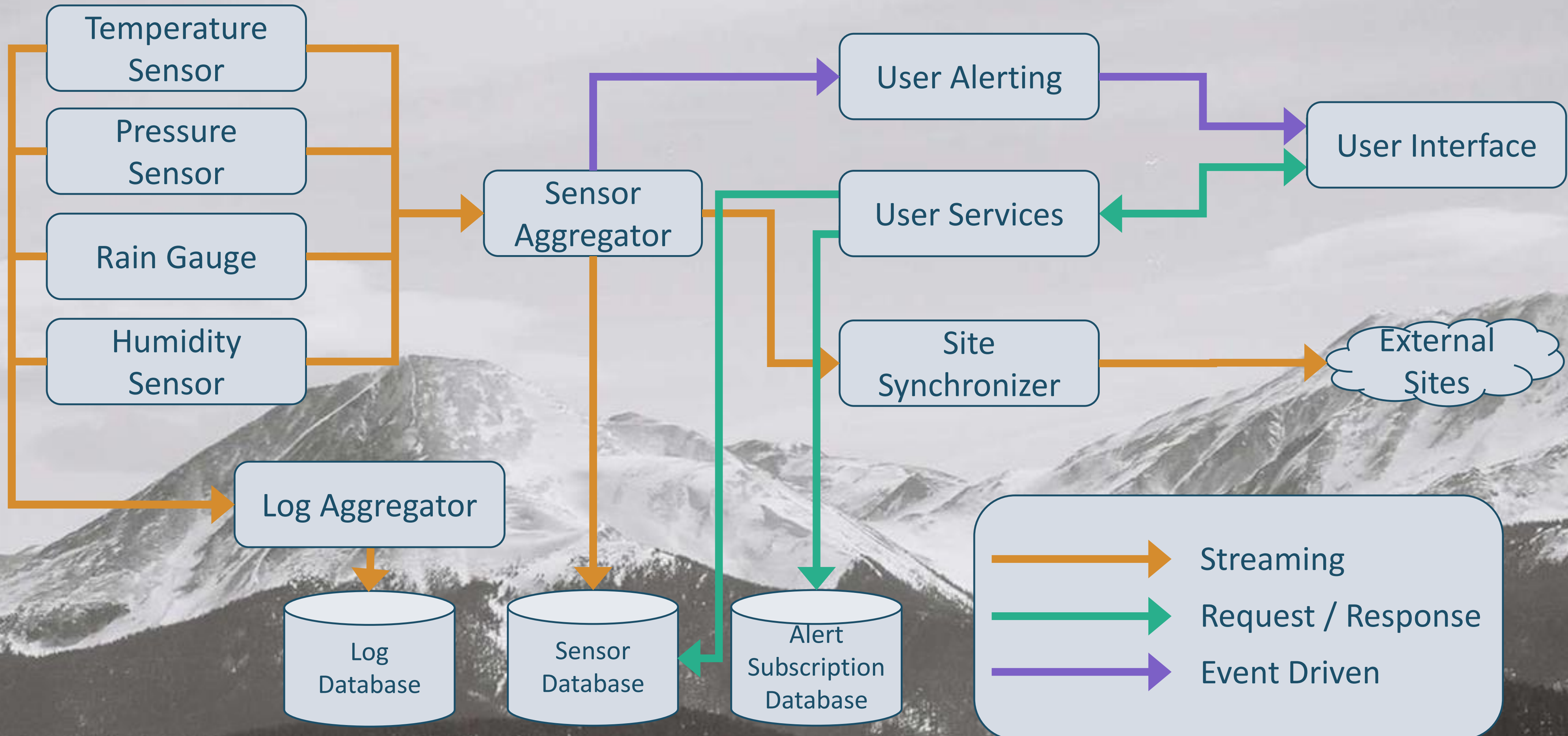
- Asynchronous vs Synchronous
  - Synchronous blocks the sender until the receiver acknowledges receipt
  - Asynchronous allows the sender to perform other tasks after transmission
- Request / Response vs Event Driven vs Real Time Streaming
- What types are present inside the weather station?
  - What parts produce data, and what consumes it?
  - Is data flowing in real time, or by request?
  - Do the senders need to know when data is received?
  - What about between weather stations?

ICR
Ideas, Commitment, Results

# Weather Station Communication

- Sensors
  - Transmitted in real time, asynchronously
- Alerts
  - Event driven, transmitted in real time, asynchronous
- User Interaction
  - Searches, historic queries, request / response, asynchronous
- System Telemetry
  - Health and status transmitted throughout the system, real time, event or streaming
- Multi-site synchronization
  - Transmitted periodically, broadcast asynchronously

# Weather Station Communication Diagram



# The Data Lens

iCR
Ideas, Commitment, Results

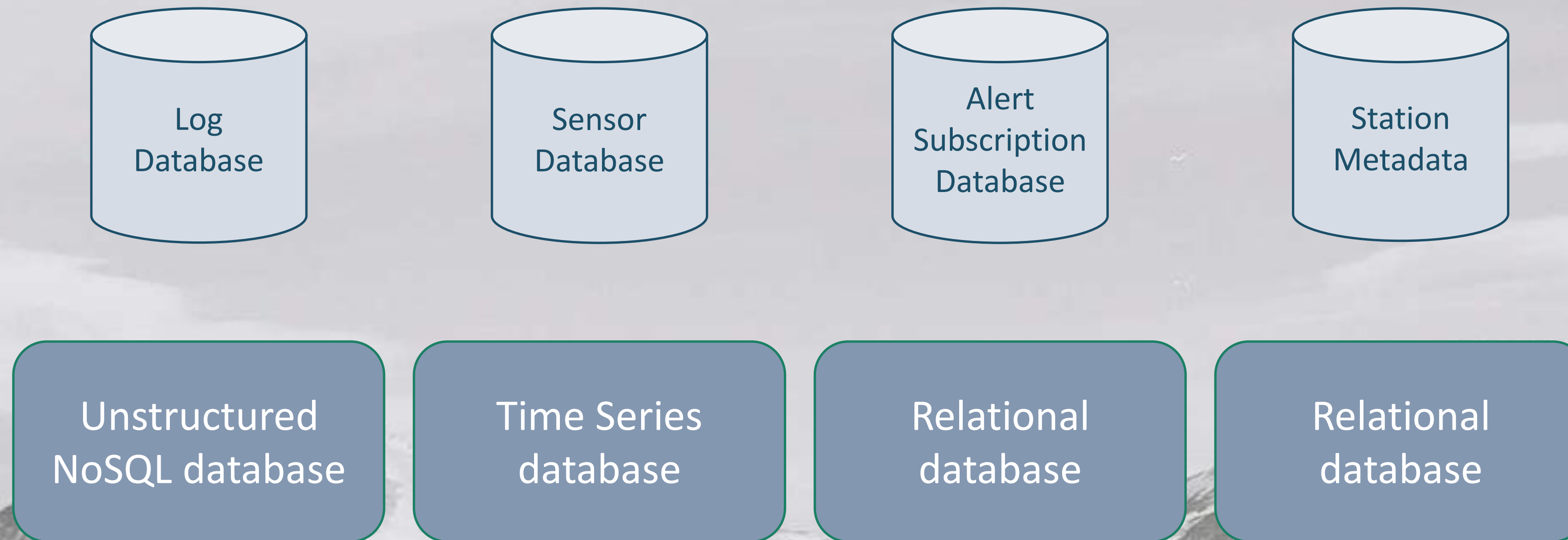
- Types of data storage:
  - Relational, Time Series, Graph, Document, etc.
  - Long-lived vs Temporary
- Retention Requirements:
  - Critical Retention – needs backup and recovery strategy
  - Ephemeral – historic information isn't useful, can be reconstructed easily
- Sensitivity Requirements:
  - Sensitive information should be encrypted (PII, Heath and Financial Records, etc.)
- Look at our target system in terms of Data
  - What data is present in the system?
  - Is all of the data the same type, and does have the same retention requirements?

iCR
Ideas, Commitment, Results

# Weather Station Data

- Sensor Data
  - Time-Series data, events are timestamped, retention not critical
- Sensor Metadata
  - Relational data, outlines type of sensor, installation date, calibration info, etc., retention somewhat critical
- Station Metadata
  - Document data, contains location information, installation date, software manifest, etc., retention somewhat critical
- User Account Data
  - Relational data, contains subscription information, login information, etc., no personally identifiable information, retention not critical

# Weather Station Data Diagram



# Functionality Lens

- Focusing on functionality, or business capabilities, allows us to design the system by defining what it should do
  - Specific business requirements are used to define key system capabilities
- When using a microservice architecture, observe the “single responsibility principle” when defining a component
  - A component should do one and only one thing, and do it well
  - Define responsibility in a single sentence, ideally without using and/or
- Outline the desired functionality of each requirement, then define components

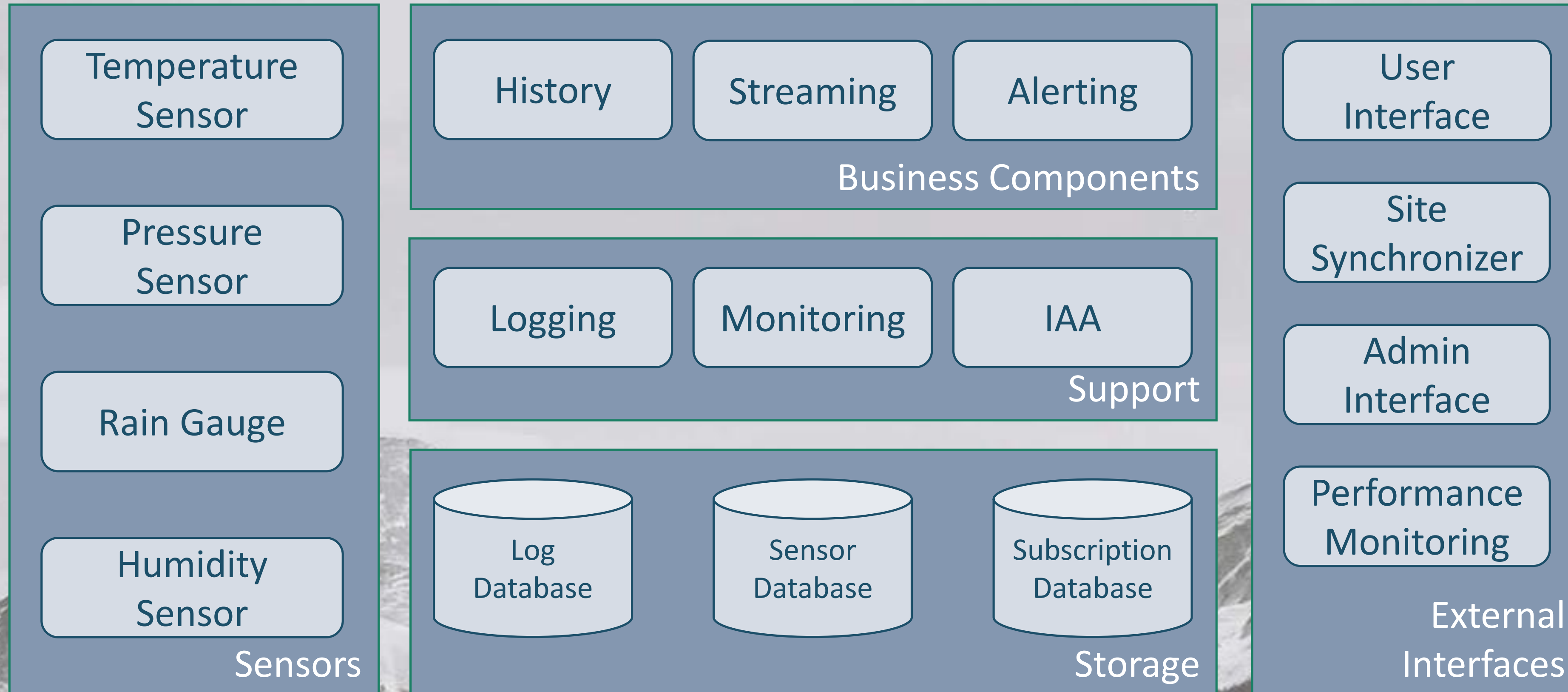
ICR
Ideas, Commitment, Results

# Weather Station Functionality

- Many specific stated requirements, and several implied requirements
- Exploring the requirements, we come up with at least the following:
  - Integrate new sensors automatically
  - Stream real-time data from each sensor, capture into a database
  - Send an alert when a threshold is breached on each sensor
    - Allow end users to define thresholds (implied)
  - Stream the current value for each sensor
  - Permit users to search historic sensor readings by time
  - Replicate sensor data to other stations

# Weather Station Functional Diagram

iCR
Ideas, Commitment, Results



ICR
Ideas, Commitment, Results

# Resiliency Lens

- The architecture can be designed to degrade in case of failure, instead of failing outright
- Strategies to increase reliability
  - Identify single points of failure, and mitigate them
  - Use circuit breaker design pattern to avoid overloads when failure occurs
  - Choose techniques that facilitate automatic recovery and failover
  - Use load balancers / API gateways to enable horizontal scaling
  - Use testing approaches to verify design, e.g.: load testing, chaos testing, testing to failure, etc.

# Resiliency Lens

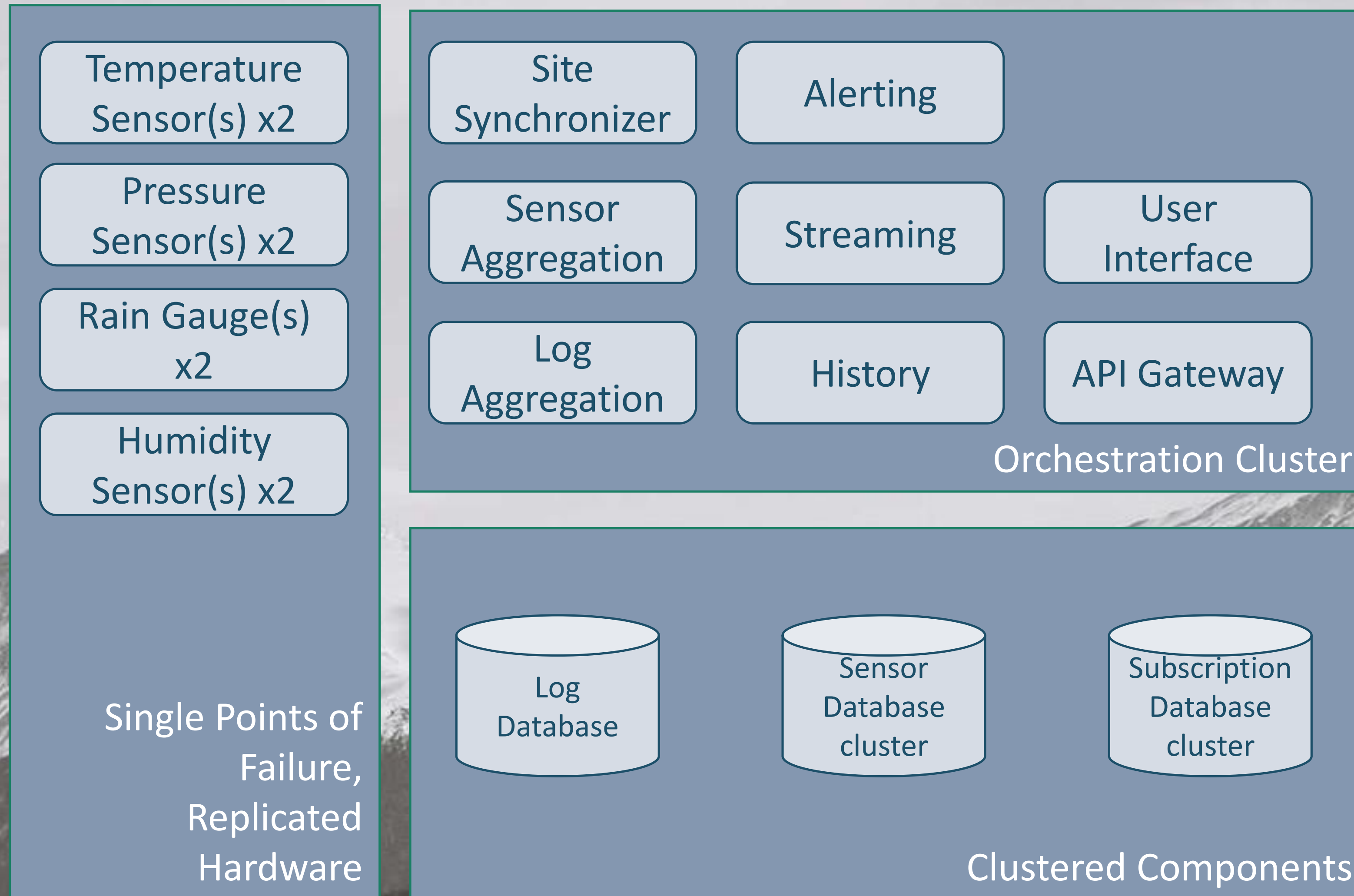
- The customer wants the system to run without manual intervention for “long periods of time”
  - This is an ambiguous requirement, and we should seek clarification on key performance criteria
  - Until we receive that clarification, we can work against the ambiguity
- If the system suffers an anomaly, how can we minimize the impact?
  - Use a risk-based approach to weigh likelihood vs impact of a failure
  - We cannot define all types of failures, this is far too broad of a problem space

iCR
Ideas, Commitment, Results

# Weather Station Resiliency

- A microservice design allows for the use of an orchestration platform
  - Kubernetes, Docker Swarm are some examples
  - The orchestration platform ensures that at least one instance of a given service is always running
  - Can also "horizontally" scale a service if load is too high
- After identification of single points of failure, those can be remediated via duplication
- Key locations of redundancy:
  - Sensors, servers, data storage

# Weather Station Resiliency Diagram



ICR
Ideas, Commitment, Results

# Infrastructure Lens

- When examining a system's design, consider the underlying infrastructure requirements, including:
  - Network requirements, including bandwidth, latency, throughput, and types of communication protocols
  - Processing requirements, including CPU and GPUs
  - Storage requirements, driven by retention policies and data volumes
  - Memory requirements, which includes container needs
  - Cost requirements, driven by size of system, and potential scaling
  - Scalability requirements, driven by anticipated usage loads
- Infrastructure includes on-premises vs cloud hosted solutions

ICR
Ideas, Commitment, Results

# Infrastructure Lens: Cloud vs On-Premises

| Constraint | Cloud | On-Premises |
|---|---|---|
| Cost | Can start small, but increase quickly, can be less predictable | Higher initially, but predictable lifecycle, must include hardware refresh, power, space, cooling, and labor costs |
| Scalability | Extremely scalable with load | Scaling difficult, requires additional hardware |
| Labor | Little infrastructure maintenance, higher up front | High up front, high infrastructure maintenance |
| Deployability | Simple deployments, typically automated | More complex, requires configuration of hardware |
| Maintenance | Lower infrastructure maintenance | Higher infrastructure maintenance, patching, etc. |

Ideas, Commitment, Results

# Weather Station Infrastructure

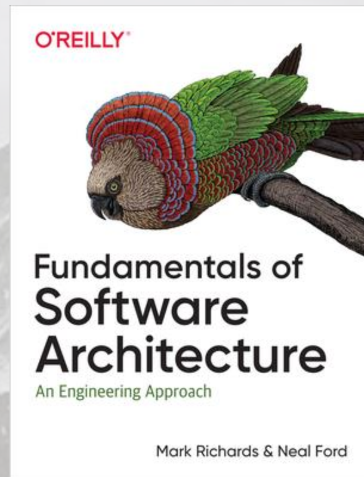
- Due to the nature of the system being tied to sensors, this is likely a good candidate for an on-premises hardware solution
- Consider load on the system for scaling:
  - Sensor load is relatively static
  - User load is dynamic, but not high
- An in-depth analysis of potential usage is critical to this decision point
  - Will yield a usage profile, i.e.: How many users are streaming data, vs performing historic searching, etc.
- This part of the process will yield a comprehensive infrastructure design

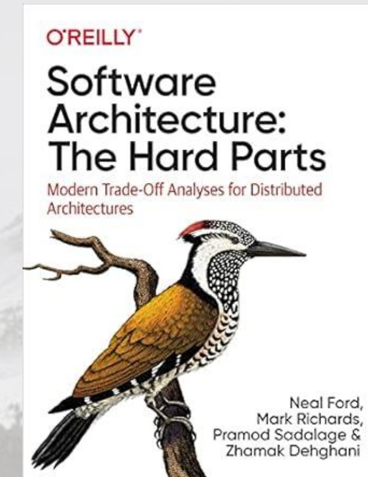
# Weather Station Design: Wrapup

- Through these lenses, we decomposed the system into smaller, focused subsystems and components
- Components are understandable to less technical stakeholders
- This decomposition is a good starting point to estimate cost, schedule, and complexity at a high level
- Each component will be further decomposed by engineering team(s)
- Further decomposition can yield better cost, schedule and complexity estimates

# Resources and Reading

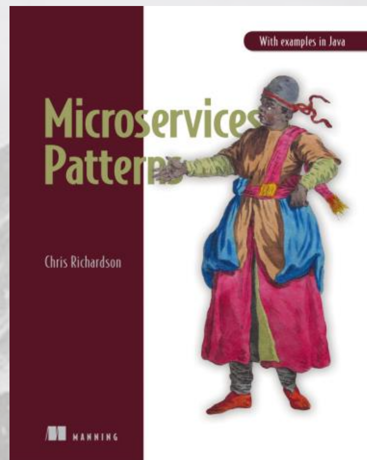
- Fundamentals of Software Architecture: An Engineering Approach*, Ford, N., & Richards, M. (2020)*
- Software Architecture: The Hard Parts, *Ford, N., Richards, M., Sadalage, P., & Dehghani, Z. (2021)*

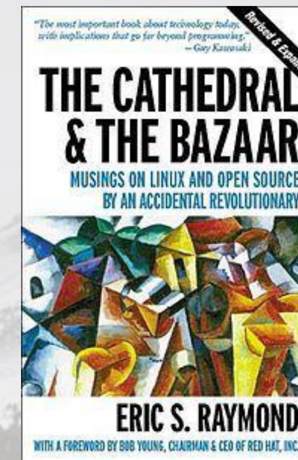


# Resources and Reading

- Microservice Patterns, *Richardson, C. (2018)*
- The Cathedral and the Bazaar, *Raymond, E. (1999)*



Ideas, Commitment, Results

# Resources and Reading

- The 12 Factor App – www.12factor.net
- https://microservices.io
- Software Architecture (Software Engineering Institute), https://www.sei.cmu.edu/our-work/software-architecture/
- Software Architecture Guide, *Fowler, M.*, https://martinfowler.com/architecture/

# Questions?