1. Reliable, Scalable, and Maintainable Applications

- Types of complex systems
  - data-intensive
    - data is its primary challenge
      - amount of data
      - complexity of data
      - speed at which it is changing
  - compute-intensive
    - CPU cycles are the bottleneck
- Standard building blocks
  - Databases
    - Store data so that they, or another application, can find it again later
  - Caches
    - Remember the result of an expensive operation, to speed up
  - Search indexes
    - Allow users to search data by keyword or filter it in various ways
  - Stream processing
    - Send a message to another process, to be handled asynchronously
  - Batch processing
    - Periodically crunch a large amount of accumulated data
- Reliability
  - making systems work correctly, even when faults occur
    - ⚠️ We can only tolerate certain types of faults - so "fault-tolerant" or "resilient" - is slightly misleading term
  - Fault
    - One component of the system deviating from its spec
  - Failure
    - The system as a whole stops providing the required service to the user
  - It is impossible to reduce the probability of a fault to zero
    - therefore it is usually best to design fault-tolerance mechanisms that prevent faults from causing failures
      - ⚠️ Good idea: Randomly killing individual processes without warning!
  - Some faults can't be tolerated and should be prevented - e.g. security faults
  - Faults
    - Hardware Faults
      - Usually just need to add hardware redundancy
      - There is a move toward systems that can tolerate the loss of entire machines
        - by using software fault-tolerance techniques
          - in preference or in addition to hardware redundancy
    - Software Errors
    - Human Errors
      - leading cause of outages
- Scalability
  - having strategies for keeping performance good, even when load increases
  - Describing Load
    - few numbers which we call load parameters
      - requests per second to a web server
      - the ratio of reads to writes in a database
      - the number of simultaneously active users in a chat room
      - the hit rate on a cache
      - etc.
    - Twitter example
      - Simple schema
      - Fan out schema
      - Hybrid schema
  - Describing Performance
    - In batch processing we think about - throughput
      - the number of records we can process per second
      - the total time it takes to run a job on a dataset of a certain size
    - In online systems - response time
      - the time between a client sending a request and receiving a response
    - Latency and response time
      - Response time
        - what the client sees
      - Latency
        - duration that a request is waiting to be handled
    - Distribution of values that you can measure
      - Usually it is better to use percentiles
      - ⚠️ the customers with the slowest requests are often those who have the most data on their accounts because they have made many purchases—that is, they're the most valuable customers
        - Amazon has also observed that a 100 ms increase in response time reduces sales by 1%, and others report that a 1-second slowdown reduces a customer satisfaction metric by 16%
      - percentiles are often used in service level objectives (SLOs) and service level agreements (SLAs)
    - 🚨 head-of-line blocking
      - Queueing delays often account for a large part of the response time at high percen- tiles
      - Due to this effect, it is important to measure response times on the client side!
      - ⚠️ When generating load artificially in order to test the scalability of a system, the load-generating client needs to keep sending requests independently of the response time
    - tail latency amplification
      - if an end-user request requires multiple back-end calls (some of which can be slow)
  - Approaches for Coping with Load
    - scaling up (vertical scaling)
    - scaling out (horizontal scaling)
    - elastic systems
      - Automatically scaling systems
      - can be useful if load is highly unpredictable
    - There is no one-size-fits-all scalable architecture
      - Depends on load parameters
        - which operations will be common and which will be rare
- Maintainability
  - making life better for the engineer- ing and operations teams who need to work with the system
  - Operability: Making Life Easy for Operations
    - Make it easy for operations teams to keep the system running smoothly
  - Simplicity: Managing Complexity
    - Make it easy for new engineers to understand the system, by removing as much complexity as possible from the system
    - Accidental complexity
      - if it is not inherent in the problem that the software solves (as seen by the users) but arises only from the implementation
      - Best tool to remove complexity is - abstraction
  - Evolvability: Making Change Easy
    - Make it easy for engineers to make changes to the system in the future, adapting it for unanticipated use cases as requirements change
      - Also known as extensibility, modifiability, or plasticity
    - Tools and patterns
      - TDD
      - Refactoring
      - and higher level approaches