Chapter 6: System Models

To effectively reason about distributed systems, we use **system models**. A system model is a set of assumptions that define what can and cannot happen within a system. It simplifies reality by encoding expectations about three core components:

1. Communication links
2. Processes
3. Timing

These models allow us to analyze and verify the correctness of distributed algorithms without getting lost in the complexity of the underlying implementation technologies.

Communication Link Models

These models define the assumed behavior of the network connections between processes.

- **Fair-loss Link**:
  - Assumes that messages may be lost or duplicated.
  - However, it guarantees that if a sender continuously retransmits a message, it will *eventually* be delivered.
- **Reliable Link**:
  - A stronger model that assumes a message is delivered *exactly once*, without any loss or duplication.
  - A reliable link can be implemented on top of a fair-loss link, for example, by using sequence numbers to de-duplicate messages at the receiver. TCP is an example of a protocol that implements a reliable link.

- **Authenticated Reliable Link**:
  - Builds on the reliable link model.
  - It makes the same assumptions but adds the guarantee that the receiver can authenticate the sender's identity. This can be implemented using a protocol like TLS on top of TCP.

Process Failure Models

These models describe how processes are expected to behave, particularly concerning failures.

- **Arbitrary-fault Model (Byzantine Model)**:
  - The most extreme model, it assumes a process can deviate from its algorithm in any way.
  - This includes crashing, sending incorrect data, or acting maliciously due to bugs or attacks.
  - A key theoretical result is that a system can tolerate up to **1/3** of its processes being Byzantine faulty and still operate correctly.
  - This model is typically used for safety-critical systems (e.g., airplane engines, nuclear power plants) and decentralized systems where no single entity has full control (e.g., Bitcoin).
- **Crash-recovery Model**:
  - Assumes that a process follows its algorithm correctly but can crash at any time.
  - After a crash, the process can restart, but it loses its in-memory state.
- **Crash-stop Model**:

  - A simpler model that assumes a process follows its algorithm correctly but, if it crashes, it **never comes back online**.
  - While unrealistic for software crashes, this model is useful for representing unrecoverable hardware faults and generally simplifies the design of algorithms.

Timing Models

These models define the assumptions made about the time it takes for messages to be delivered and for processes to execute operations.

- **Synchronous Model**:
  - Assumes that sending a message or executing an operation will never take more than a known, fixed amount of time.
  - This model is **not very realistic** for most distributed systems, where network latency can be unpredictable and processes can be slowed by garbage collection cycles or page faults.
- **Asynchronous Model**:
  - Assumes that sending a message or executing an operation can take an **unbounded** amount of time.
  - While this model is simpler to reason about, many distributed computing problems are impossible to solve under this assumption, as an algorithm could get stuck waiting forever.
- **Partially Synchronous Model**:
  - A realistic middle ground that assumes the system behaves synchronously *most of the time*, but allows for periods of asynchronous

behavior (e.g., high network latency).
  - This model is generally considered representative enough of real-world systems.

The Book's Assumed Model

For the most part, the algorithms presented in this book will assume a system model with:

- **Fair-loss links**
- **Crash-recovery processes**
- **Partial synchrony**

It is important to remember that all models are abstractions of reality. They are useful tools for reasoning, but they don't capture every real-world nuance. As you learn about algorithms, it is helpful to question the models' assumptions and consider how those algorithms might break in practice.