Preface

Software engineering can be a delightful profession, but at the same time, software engineering can be a dreadful profession. For me, software engineering is particularly dreadful when I experience confusion.

Unfortunately, in this industry, confusion is a constant state of mind. Our ability to think about software systems and our ability to talk about software systems lacks the maturity of other professions that have developed strong mental models and means of communication. For example, ask yourself: What is a micro service, what is the cloud, what is cloud native, or what is serverless — an endless list of questions. Do you know the answers?

Often we have a fuzzy notion, a vague idea, that we cannot communicate effectively and efficiently.

For me, this is a source of great frustration. I do not want a fuzzy notion of a system, I do not want to probe the system, poke it with a stick, observe and rationalize its behavior until the next poke unexpectedly yet inadvertently invalidates what I think I knew.

I want an accurate and concise mental model to reason confidently about the system.

In this book, we will develop accurate and concise mental models to reason confidently about distributed systems. Additionally, you will learn to construct your own mental models to replace confusion with certainty and hesitance with confidence.

Thinking in distributed systems

There is a distinction between knowing about complex distributed systems and truly understanding complex distributed systems.

The distinction between knowing and understanding is not limited to complex, distributed systems, but can also be seen elsewhere, for example in board games. While it may be easy to learn the basic rules of a game, it takes a much greater investment of time to master the strategies and tactics required for full understanding. This process may take months, years, or even a lifetime.

The game of chess showcases the divide between knowing and understanding perfectly. While it only takes a short amount of time to learn the rules of the game, it takes much longer to truly understand the strategies and tactics necessary to excel at it. This is why chess is often seen as a measure of intelligence, both for humans and machines. Despite its simple setup of a board, six types of pieces, and two players, chess is a remarkably complex and sophisticated game.

The same concept applies to distributed systems. So as we progress through this book, our goal is to not only build a deeper knowledge but also to build a deeper understanding of distributes systems.

AHA! moments

On my own journey to understanding distributed systems, I had many moments of realization, or “AHA!” moments. These moments always left me excited and increased my confidence.

In this book, I hope to share my AHA! moments with you. Some may be obvious to you, while others may be more profound. However, I don't think that I'm the only one who struggled to gain a deeper understanding of these concepts, and I believe that sharing my own experiences, whether they seem obvious or profound, may be helpful to you as well.

Distributed System Incorporated

A distributed system is made up of multiple components that operate concurrently and communicate with each other by sending and receiving messages over a network.

- The overall behavior of a distributed system is a result of the behavior of its individual components and how they interact with each other.
- The overall complexity of a distributed system is a result of the complexity of its components and the complexity of the interactions between them.

As we dive into the topic of distributed systems, I encourage you to picture them as a cooperation situated in an office building.

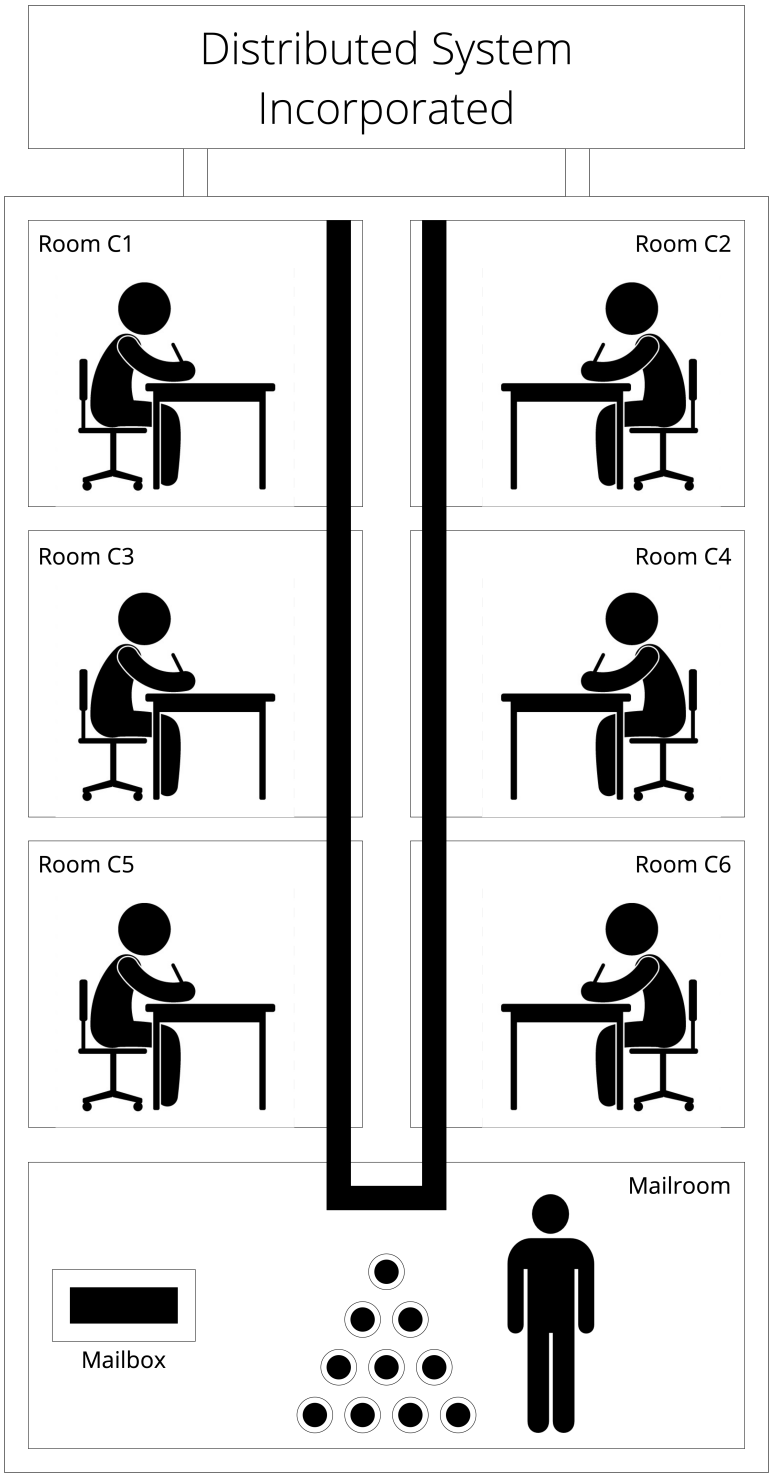


Figure 1. Distributed Systems Incorporated

The building represents the system, with each room within it representing a component. These rooms are connected by pneumatic tubes, which act as the network for communication. The office building is connected to the rest of the world via a mailbox where all incoming and outgoing messages are processed.

I find this mental model helpful because it allows me to think about and talk about distributed systems: It takes an intangible, abstract cyber system and maps it onto a tangible, concrete physical system, while faithfully capturing the core mechanics of the system.

In addition, this model is able to represent a wide range of concerns. For example, what happens to Distributed Systems Incorporated if Bob the mailroom attendant loses messages, duplicates messages, or rearranges messages? This reflects different types of message delivery semantics. What happens when employees take a 15 min break, take vacation, or leave the company? This reflects different types of crashes.

Through these examples, we will analyze the consequences of these actions and explore potential countermeasures.

You may be surprised at how far this model can take you. Try using it to model Kubernetes Inc., Kafka Inc., or a multi-cluster deployment involving multiple buildings and a public mail service like USPS.

Although we won't be covering Kubernetes or Kafka in this book, I am eager to use insights gained from Distributed Systems Inc. to accurately and concisely define the terms *service* and dare I say *microservice* .

Some AHA! Moments

Even without delving deeply into the complexities of distributed systems, we can already begin to have some AHA! moments. Let's preview a few informally and explore them further in future chapters:

AHA! Moment • Simplicity vs Complexity

While each member of our staff may only handle a simple set of tasks and follow a simple set of rules, the resulting behavior of Distributed Systems Inc. is complex. Simple components do not necessarily result in simple systems!

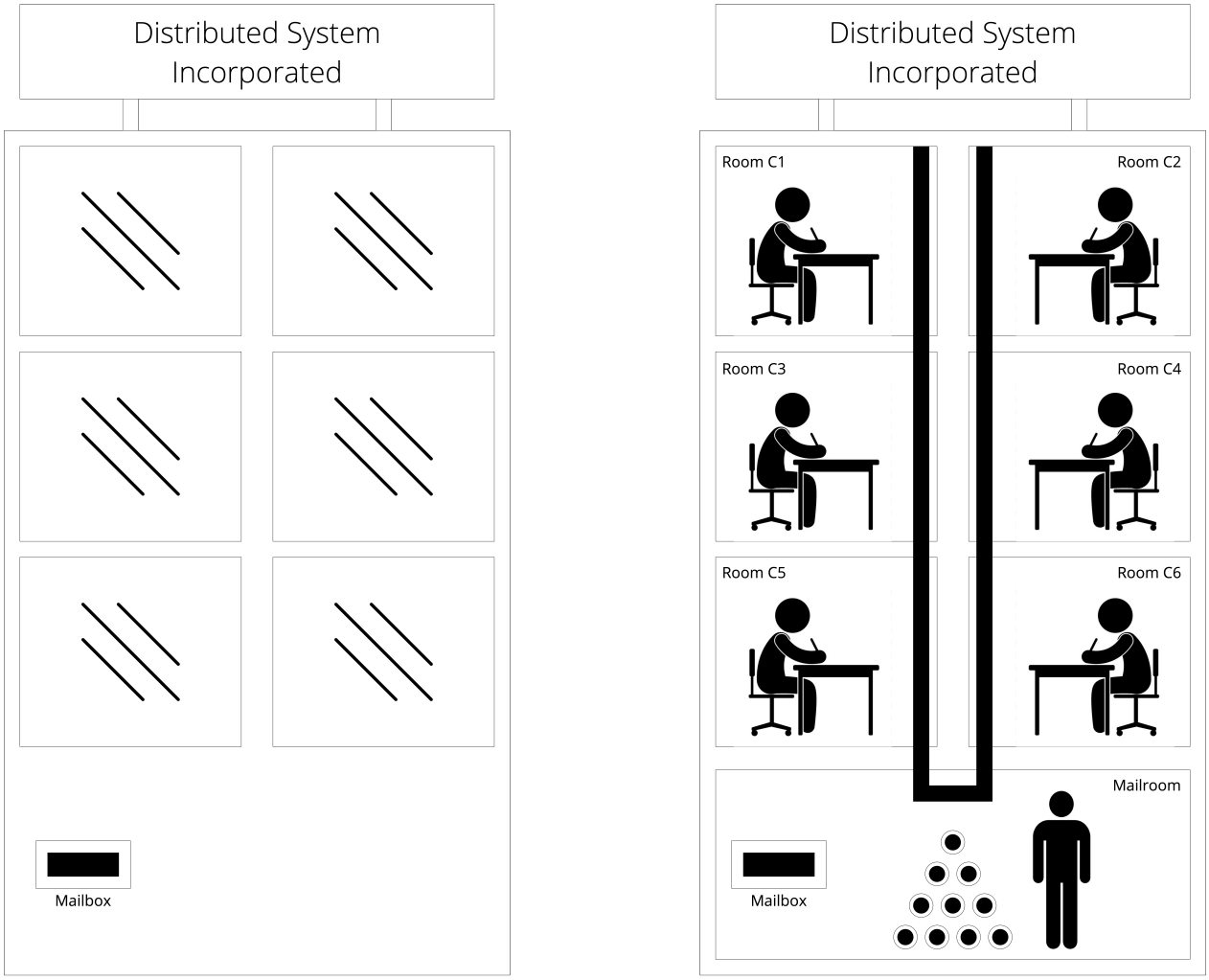
AHA! Moment • Emergent Behavior

Interesting behavior of Distributed System Inc. cannot be traced back to individual employees. For example, one employee cannot be held responsible for the scalability of the company, as there is a limit to how much work they can accomplish in a day. Additionally, one employee cannot be held responsible for the reliability of the company, as they may be absent due to illness or leave the company.

Instead, the interesting behavior of Distributed System Inc. is emergent, resulting from the behavior of individual employees and their interactions. This is the very foundation of the idea to build reliable systems from unreliable components.

AHA! Moment • Changing Perspectives

For me, this was one of my most profound AHA! moments - and also one of the most vulnerable moments in my career. To this day, I am almost embarrassed that it took me so long to realize it.



Black Box Model White Box Model

Figure 2. Black Box vs. White Box, global point of view

We often easily and instinctively think about systems from a black box and a white box perspective. However, moving from a black box to a white box perspective or vice versa is a change in resolution, not a change in perspective.

As shown in Figure 2., whether we are looking at the black box or the white box side, we are considering the system from the perspective of an all-knowing observer, meaning that we have the ability to observe the state of the entire system, **we have a global view**.

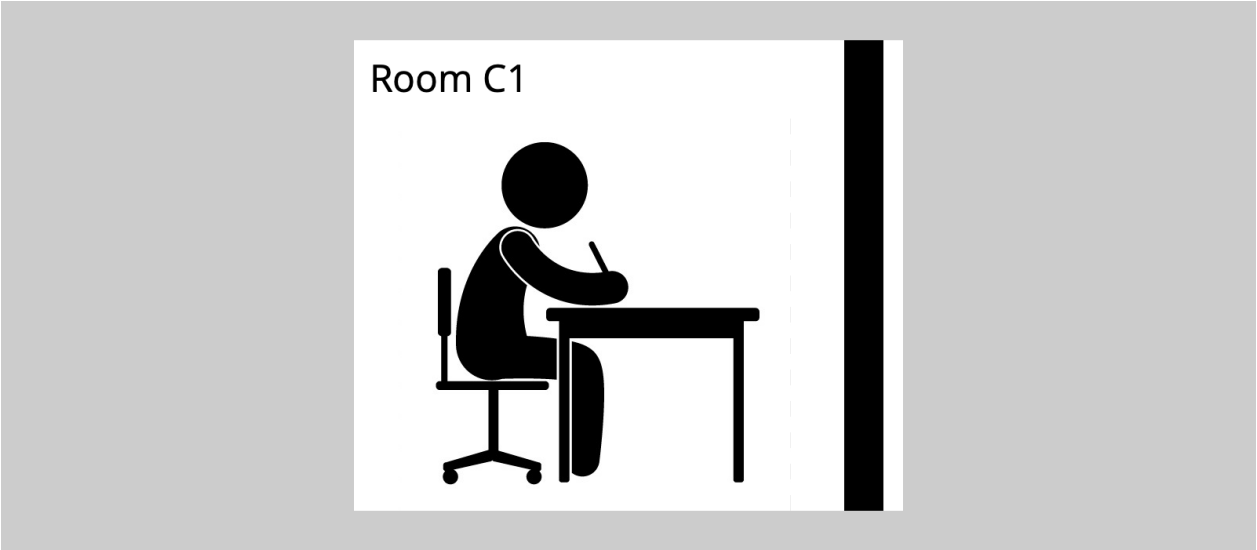


Figure 3. Local point of view

However, Figure 3. illustrates that a component does not have the same luxury as an all-knowing observer. A component can only observe its own state, **giving it a limited, local view.**

As a side note: While the depiction in Figure 3 may seem lonely and depressing, I like to think that Distributed System Incorporated has a positive work culture, strong team dynamics, and the highest levels of employee satisfaction.

On my own journey of understanding and thinking in distributed systems, it took me a long time to change my perspective. In hindsight, I think it would have been beneficial to make this shift at the outset.

With this shift in perspective, we are able to accurately and concisely identify the core challenge of distributed systems: thinking globally while acting locally.

Think globally act locally

The core challenge of distributed systems is to design a global algorithm, while each component of the system implements a local algorithm.

In other words, the challenge is to create a system that functions as a cohesive whole, despite the fact that each component is only aware of and able to access local information.

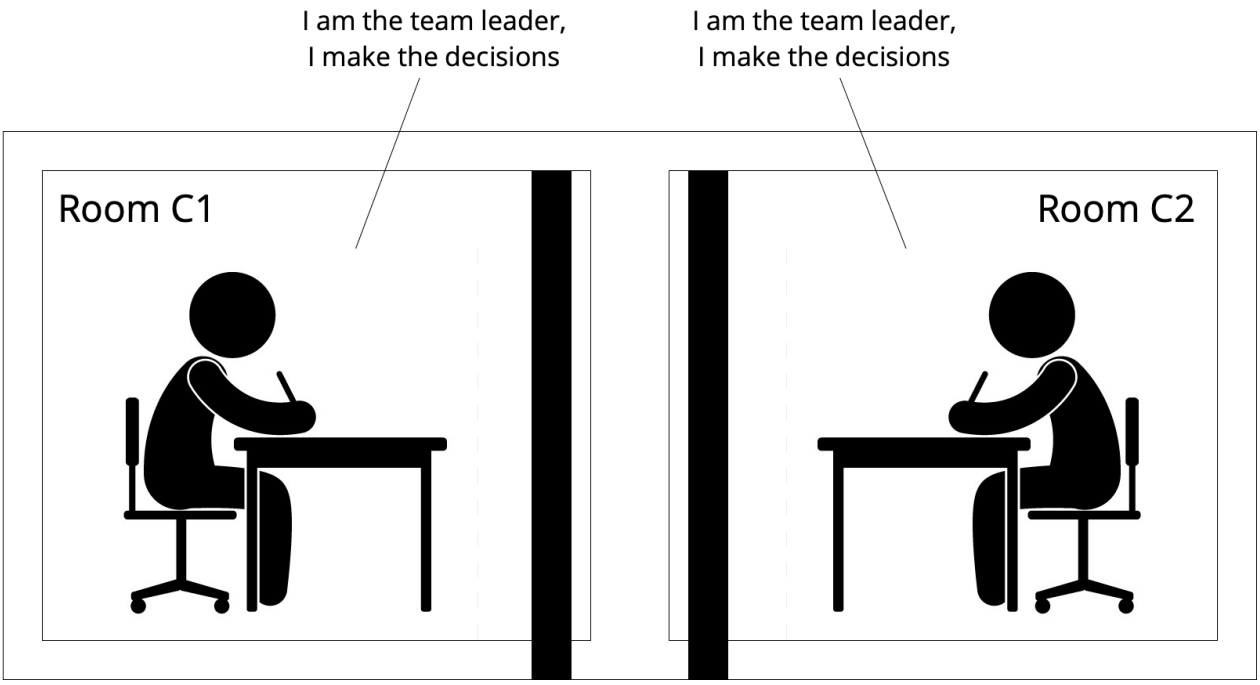


Figure 4. Splitbrain

Figure 4. illustrates a common example of the difficulty in ensuring global correctness with only limited knowledge: Splitbrain. In this scenario, global correctness depends on one person being the leader and making decisions. But how can you ensure that only one person believes they are the team lead?

What are the participants involved in this process, what local knowledge do they possess, and what local steps do they take to ensure global guarantees are met?

Conclusion

In the chapters ahead, we will utilize the analogy of Distributed System Incorporated to map cyber systems to physical systems, making abstract concepts more concrete while still maintaining the core mechanics of the system.

We have briefly touched upon several important concepts related to distributed systems. In future chapters, we will dive deeper into each of these points:

- The complexity of distributed systems: We will explore how simple components can combine to create complex systems and how this complexity arises from emergent behavior.
- Building reliable systems from unreliable components: We will delve into the foundation of emergent behavior and how it allows us to construct reliable systems from unreliable components.
- Thinking globally while acting locally: We will examine the central challenge of distributed systems and the importance of shifting perspective from an all-knowing observer to the limited, local view of an individual component.

I am excited to embark on this journey with you.