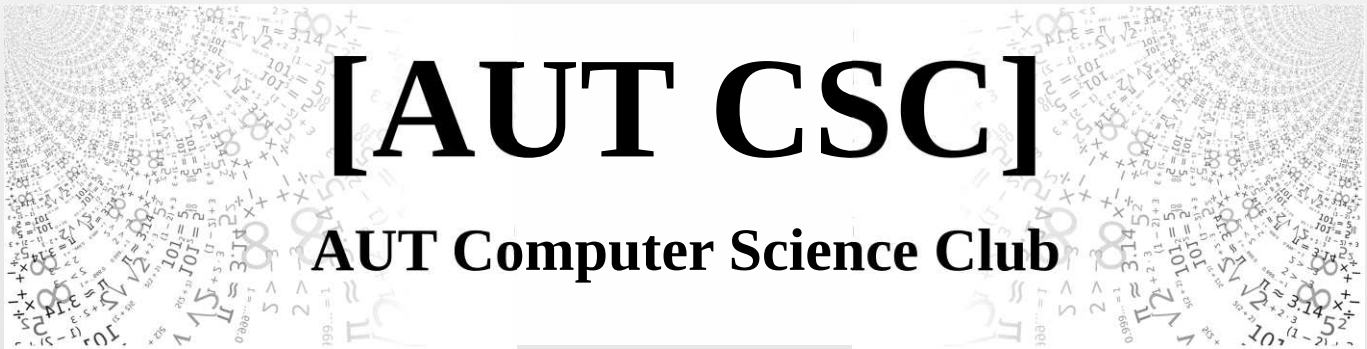
The Two Generals Problem

Or: Why distributed systems are Nintendo Hard



Koz Ross

March 16, 2017

Overview

Introduction to distributed systems

The Two Generals Problem

Consequences

Questions

Introduction to distributed systems

What is a distributed system?

A system is *distributed* if it has:

What is a distributed system?

A system is *distributed* if it has:

- Multiple processing units (so we can do things ‘at the same time’)

What is a distributed system?

A system is *distributed* if it has:

- Multiple processing units (so we can do things ‘at the same time’)
- Independent failure (one unit failing shouldn’t bring down the whole system)

What is a distributed system?

A system is *distributed* if it has:

- Multiple processing units (so we can do things ‘at the same time’)
- Independent failure (one unit failing shouldn’t bring down the whole system)
- Unreliable communication (information about other units is limited and uncertain, connections between units can fail unpredictably)

What is a distributed system?

A system is *distributed* if it has:

- Multiple processing units (so we can do things ‘at the same time’)
- Independent failure (one unit failing shouldn’t bring down the whole system)
- Unreliable communication (information about other units is limited and uncertain, connections between units can fail unpredictably)

Systems which are not distributed are called otherwise, *singular* (or *centralized* in some contexts).

Examples of distributed systems

Example 1: Social networks (like Facebook, Twitter, etc)

Examples of distributed systems

Example 1: Social networks (like Facebook, Twitter, etc)

- Multiple processing units (client apps, web browsers, servers)

Examples of distributed systems

Example 1: Social networks (like Facebook, Twitter, etc)

- Multiple processing units (client apps, web browsers, servers)
- Independent failure (if your computer catches fire, everyone else can still tell each other what they had for breakfast today)

Examples of distributed systems

Example 1: Social networks (like Facebook, Twitter, etc)

- Multiple processing units (client apps, web browsers, servers)
- Independent failure (if your computer catches fire, everyone else can still tell each other what they had for breakfast today)
- Unreliable communication (timelines can go out of sync, trending posts can vary, updates might be lost or take a long time)

Examples of distributed systems

Example 2: Torrents (legitimate or otherwise)

Examples of distributed systems

Example 2: Torrents (legitimate or otherwise)

- Multiple processing units (everyone can seed or download different things, or parts of them, independently, at the same time)

Examples of distributed systems

Example 2: Torrents (legitimate or otherwise)

- Multiple processing units (everyone can seed or download different things, or parts of them, independently, at the same time)
- Independent failure (if a single peer disconnects or runs slowly, everyone else can still download or seed)

Examples of distributed systems

Example 2: Torrents (legitimate or otherwise)

- Multiple processing units (everyone can seed or download different things, or parts of them, independently, at the same time)
- Independent failure (if a single peer disconnects or runs slowly, everyone else can still download or seed)
- Unreliable communication (peers randomly join and leave, network failure to particular peers, slow connections. . .)

Examples of distributed systems

Example 3: Human society (at any scale)

Examples of distributed systems

Example 3: Human society (at any scale)

- Multiple processing units (me teaching a class at 10am happens whether or not you care or show up)

Examples of distributed systems

Example 3: Human society (at any scale)

- Multiple processing units (me teaching a class at 10am happens whether or not you care or show up)
- Independent failure (society doesn't stop because one person gets sick or dies)

Examples of distributed systems

Example 3: Human society (at any scale)

- Multiple processing units (me teaching a class at 10am happens whether or not you care or show up)
- Independent failure (society doesn't stop because one person gets sick or dies)
- Unreliable communication (every sitcom ever. . .)

Examples of distributed systems

Example 3: Human society (at any scale)

- Multiple processing units (me teaching a class at 10am happens whether or not you care or show up)
- Independent failure (society doesn't stop because one person gets sick or dies)
- Unreliable communication (every sitcom ever. . .)

This shows that distributed systems don't just apply to computing!

Why do we care?

Why do we care?

- Distributed systems are *useful*

Why do we care?

- Distributed systems are *useful*
- Distributed systems are *everywhere*

Why do we care?

- Distributed systems are *useful*
- Distributed systems are *everywhere*
- Distributed systems are *weird* (and thus, Nintendo Hard)

Why do we care?

- Distributed systems are *useful*
- Distributed systems are *everywhere*
- Distributed systems are *weird* (and thus, Nintendo Hard)

Whatever you plan to do, distributed systems, their creation, maintenance and use *will* be *your* problem.

Why do we care?

- Distributed systems are *useful*
- Distributed systems are *everywhere*
- Distributed systems are *weird* (and thus, Nintendo Hard)

Whatever you plan to do, distributed systems, their creation, maintenance and use *will* be *your* problem. Understanding their weirdness is essential to making them behave and do what you want.

Why do we care?

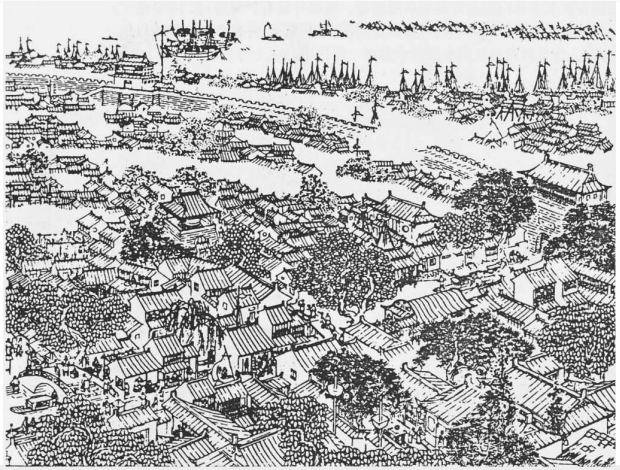
- Distributed systems are *useful*
- Distributed systems are *everywhere*
- Distributed systems are *weird* (and thus, Nintendo Hard)

Whatever you plan to do, distributed systems, their creation, maintenance and use *will* be *your* problem. Understanding their weirdness is essential to making them behave and do what you want.

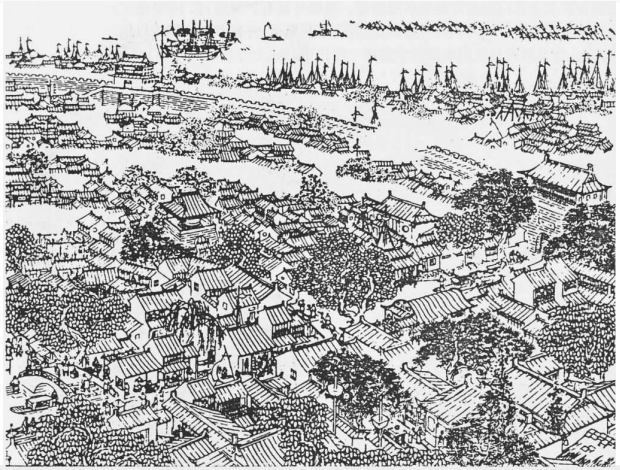
The alternative to understanding distributed systems is *awful* (look at AWS and Github, and that's just recently!).

The Two Generals Problem

Problem overview

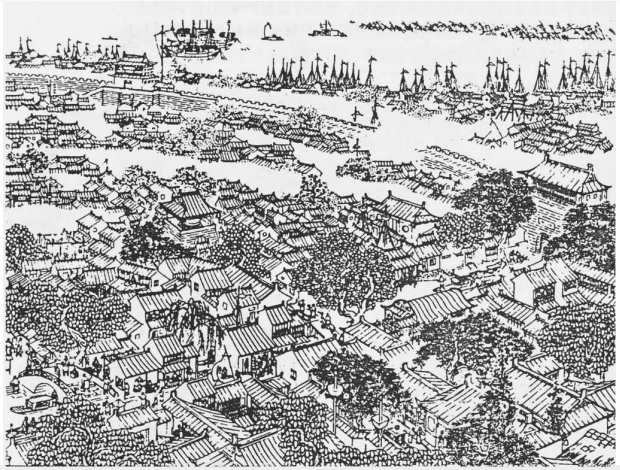


Problem overview

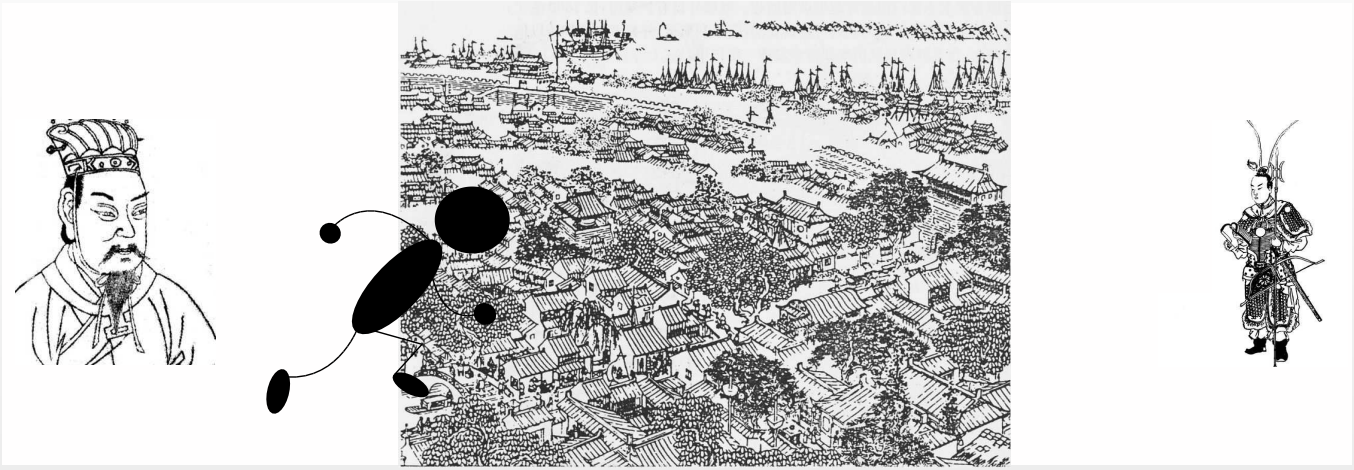

Both generals *must* attack at the same time if they want to take the city.

Problem overview



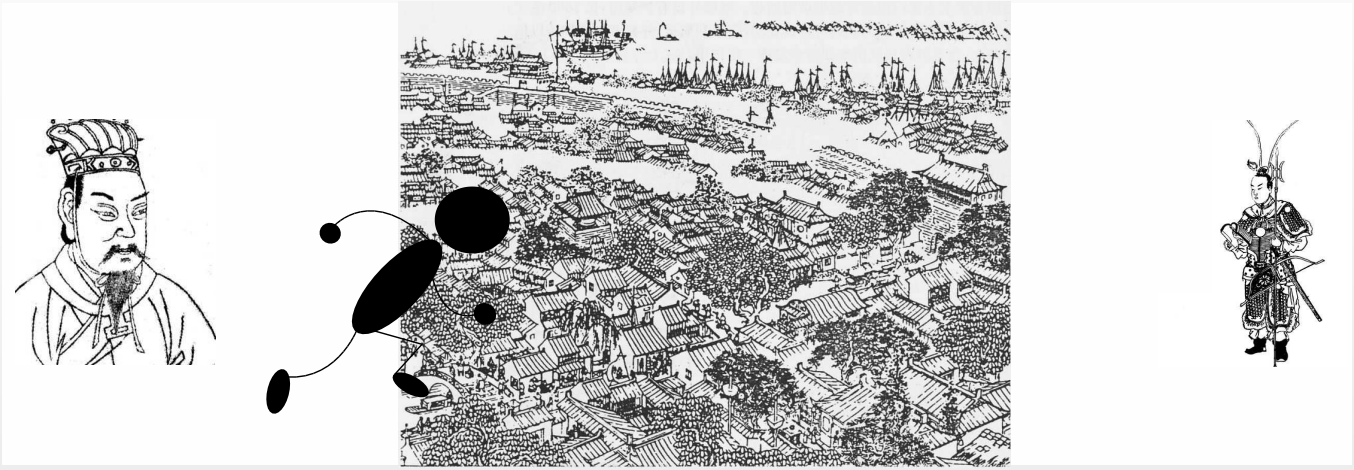
Both generals *must* attack at the same time if they want to take the city. However, they haven't agreed on a time before they set up camp.

Problem overview



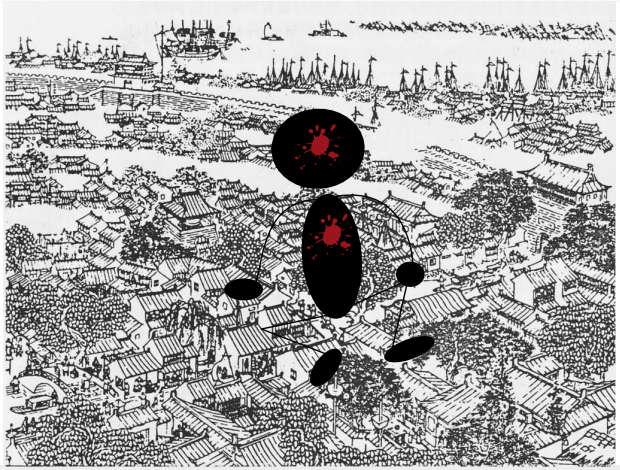
The only way the generals can communicate with each other is by sending messengers between their camps.

Problem overview



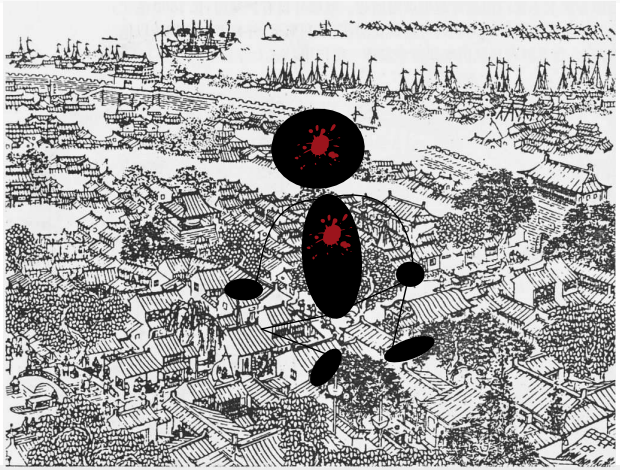
The only way the generals can communicate with each other is by sending messengers between their camps. In order for the messengers to get there in a timely manner, they must run through the city.

Problem overview



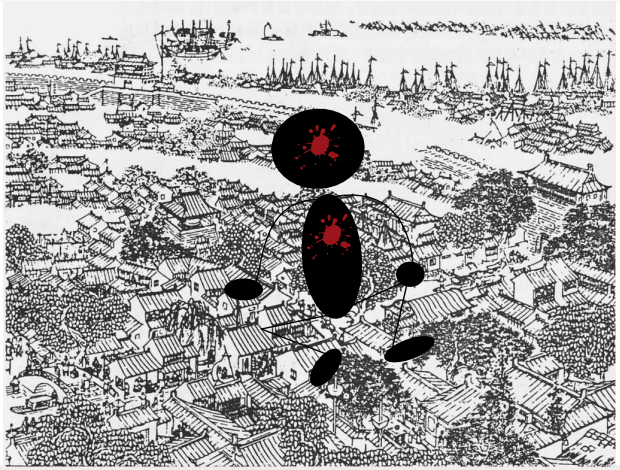
Some messengers won’t make it.

Problem overview



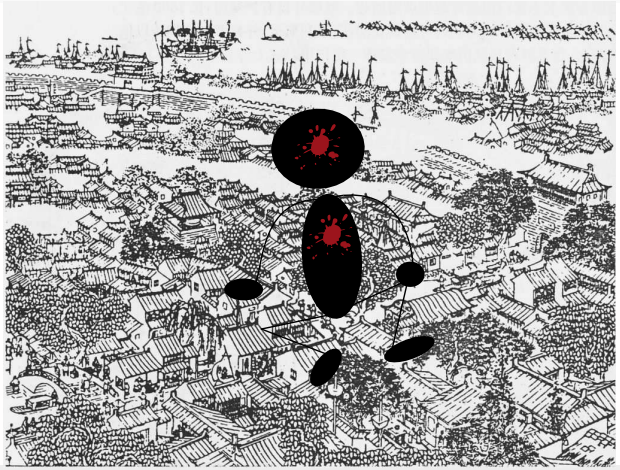
Some messengers won't make it. Thus, neither general knows which of their messages got delivered, or what the chance of any given message getting delivered is.

Problem overview



Question: Is it possible for both generals to agree on an attack time with *total* certainty?

Problem overview


Question: Is it possible for both generals to agree on an attack time with *total* certainty? *No.*

Preliminaries

A *message* is some $x \in \mathbb{N}$.

Preliminaries

A *message* is some $x \in \mathbb{N}$. An *outbox* is a list of sent messages, in the order they were sent.

Preliminaries

A *message* is some $x \in \mathbb{N}$. An *outbox* is a list of sent messages, in the order they were sent.

Each general G has an outbox $\mathrm{out}(G)$, and a *decision* $d(G)$, where $d(G) \in \mathbb{N} \cup \{\mathrm{undef}\}$.

Preliminaries

A *message* is some $x \in \mathbb{N}$. An *outbox* is a list of sent messages, in the order they were sent.

Each general G has an outbox $\mathrm{out}(G)$, and a *decision* $d(G)$, where $d(G) \in \mathbb{N} \cup \{\mathrm{undef}\}$. Initially, any general G has $\mathrm{out}(G) = \emptyset$ and $d(G) = \mathrm{undef}$.

Preliminaries

A *message* is some $x \in \mathbb{N}$. An *outbox* is a list of sent messages, in the order they were sent.

Each general G has an outbox $\mathrm{out}(G)$, and a *decision* $d(G)$, where $d(G) \in \mathbb{N} \cup \{\mathrm{undef}\}$. Initially, any general G has $\mathrm{out}(G) = \emptyset$ and $d(G) = \mathrm{undef}$. Each general is only aware of their own outbox and decision at any given time.

Preliminaries

A *message* is some $x \in \mathbb{N}$. An *outbox* is a list of sent messages, in the order they were sent.

Each general G has an outbox $\mathrm{out}(G)$, and a *decision* $d(G)$, where $d(G) \in \mathbb{N} \cup \{\mathrm{undef}\}$. Initially, any general G has $\mathrm{out}(G) = \emptyset$ and $d(G) = \mathrm{undef}$. Each general is only aware of their own outbox and decision at any given time.

We also define a *success probability* $P \in (0, 1)$.

Preliminaries

A *message* is some $x \in \mathbb{N}$. An *outbox* is a list of sent messages, in the order they were sent.

Each general G has an outbox $\mathrm{out}(G)$, and a *decision* $d(G)$, where $d(G) \in \mathbb{N} \cup \{\mathrm{undef}\}$. Initially, any general G has $\mathrm{out}(G) = \emptyset$ and $d(G) = \mathrm{undef}$. Each general is only aware of their own outbox and decision at any given time.

We also define a *success probability* $P \in (0, 1)$. This can change as messages get sent.

Preliminaries

A *message* is some $x \in \mathbb{N}$. An *outbox* is a list of sent messages, in the order they were sent.

Each general G has an outbox $\mathrm{out}(G)$, and a *decision* $d(G)$, where $d(G) \in \mathbb{N} \cup \{\mathrm{undef}\}$. Initially, any general G has $\mathrm{out}(G) = \emptyset$ and $d(G) = \mathrm{undef}$. Each general is only aware of their own outbox and decision at any given time.

We also define a *success probability* $P \in (0, 1)$. This can change as messages get sent. Neither of the generals know anything about P beyond these two facts.

Preliminaries

A *message* is some $x \in \mathbb{N}$. An *outbox* is a list of sent messages, in the order they were sent.

Each general G has an outbox $\mathrm{out}(G)$, and a *decision* $d(G)$, where $d(G) \in \mathbb{N} \cup \{\mathrm{undef}\}$. Initially, any general G has $\mathrm{out}(G) = \emptyset$ and $d(G) = \mathrm{undef}$. Each general is only aware of their own outbox and decision at any given time.

We also define a *success probability* $P \in (0, 1)$. This can change as messages get sent. Neither of the generals know anything about P beyond these two facts.

We will refer to our generals as C and L (for no particular reason whatsoever).

Preliminaries

A general G can send a message m to another general G'. To do this, we add m to the end of $\mathrm{out}(G)$.

Preliminaries

A general G can send a message m to another general G'. To do this, we add m to the end of $\mathrm{out}(G)$. Then, we generate a random $p \in (0, 1)$ and compare it to P: if $p \leq P$, then the message *arrives*, otherwise, it is *lost*.

Preliminaries

A general G can send a message m to another general G'. To do this, we add m to the end of $\mathrm{out}(G)$. Then, we generate a random $p \in (0, 1)$ and compare it to P: if $p \leq P$, then the message *arrives*, otherwise, it is *lost*.

If m arrives, we set $d(G') = m$ (the generals are genuine and don't want to sabotage each other).

Preliminaries

A general G can send a message m to another general G'. To do this, we add m to the end of $\mathrm{out}(G)$. Then, we generate a random $p \in (0, 1)$ and compare it to P: if $p \leq P$, then the message *arrives*, otherwise, it is *lost*.

If m arrives, we set $d(G') = m$ (the generals are genuine and don't want to sabotage each other).

We say that our generals have *reached agreement* if:

- $d(C) \neq \mathrm{undef}$ and $d(L) \neq \mathrm{undef}$; and
- $d(C) = d(L)$

Problem statement

Theorem. *There is no* $\mathrm{out}(C), \mathrm{out}(L)$ *such that* C, L *are guaranteed to reach agreement with probability 1.*

Problem statement

Theorem. *There is no* $\mathrm{out}(C), \mathrm{out}(L)$ *such that* C, L *are guaranteed to reach agreement with probability 1.*

Note that this does *not* claim that we cannot *ever* reach agreement — only that we can't be certain that we will, given only the outboxes of both generals.

Proof

Observation. *If* $\mathrm{out}(C) = \mathrm{out}(L) = \emptyset$*, then we cannot have reached agreement.*

Proof

Observation. *If* $\mathrm{out}(C) = \mathrm{out}(L) = \emptyset$*, then we cannot have reached agreement.*

This is intentional: if nobody's sent any messages, clearly nobody's made any decisions, and thus, nobody's agreed to anything.

Proof

Proof. Suppose for the sake of a contradiction that there exist some $\mathrm{out}(C), \mathrm{out}(L)$ such that we can guarantee C, L reaching agreement with probability 1. Consider some message m in either outbox, as well as P at the time m was sent.

Proof

Proof. Suppose for the sake of a contradiction that there exist some $\mathrm{out}(C), \mathrm{out}(L)$ such that we can guarantee C, L reaching agreement with probability 1. Consider some message m in either outbox, as well as P at the time m was sent.

As $P \neq 1$, the probability that m was lost is $1 - P \neq 0$.

Proof

Proof. Suppose for the sake of a contradiction that there exist some $\mathrm{out}(C), \mathrm{out}(L)$ such that we can guarantee C, L reaching agreement with probability 1. Consider some message m in either outbox, as well as P at the time m was sent.

As $P \neq 1$, the probability that m was lost is $1 - P \neq 0$. As after sending each message in $\mathrm{out}(C), \mathrm{out}(L)$, we are guaranteed to reach agreement with probability 1, m arriving cannot have been essential for reaching agreement.

Proof

Proof. Suppose for the sake of a contradiction that there exist some $\mathrm{out}(C), \mathrm{out}(L)$ such that we can guarantee C, L reaching agreement with probability 1. Consider some message m in either outbox, as well as P at the time m was sent.

As $P \neq 1$, the probability that m was lost is $1 - P \neq 0$. As after sending each message in $\mathrm{out}(C), \mathrm{out}(L)$, we are guaranteed to reach agreement with probability 1, m arriving cannot have been essential for reaching agreement. Thus, even if we don't send m, we will still reach agreement with probability 1.

Proof

Proof. Suppose for the sake of a contradiction that there exist some $\mathrm{out}(C), \mathrm{out}(L)$ such that we can guarantee C, L reaching agreement with probability 1. Consider some message m in either outbox, as well as P at the time m was sent.

As $P \neq 1$, the probability that m was lost is $1 - P \neq 0$. As after sending each message in $\mathrm{out}(C), \mathrm{out}(L)$, we are guaranteed to reach agreement with probability 1, m arriving cannot have been essential for reaching agreement. Thus, even if we don't send m, we will still reach agreement with probability 1.

As m is an arbitrary message, it follows that we can reach agreement when $\mathrm{out}(C) = \mathrm{out}(L) = \emptyset$.

Proof

Proof. Suppose for the sake of a contradiction that there exist some $\mathrm{out}(C), \mathrm{out}(L)$ such that we can guarantee C, L reaching agreement with probability 1. Consider some message m in either outbox, as well as P at the time m was sent.

As $P \neq 1$, the probability that m was lost is $1 - P \neq 0$. As after sending each message in $\mathrm{out}(C), \mathrm{out}(L)$, we are guaranteed to reach agreement with probability 1, m arriving cannot have been essential for reaching agreement. Thus, even if we don't send m, we will still reach agreement with probability 1.

As m is an arbitrary message, it follows that we can reach agreement when $\mathrm{out}(C) = \mathrm{out}(L) = \emptyset$. However, this is a contradiction, as by definition, this is impossible. Thus, no such $\mathrm{out}(C), \mathrm{out}(L)$ can exist. □

Consequences

For computer scientists

- Whenever we need *any* agreement on state between components in a distributed system, we are going to have a bad time.

For computer scientists

- Whenever we need *any* agreement on state between components in a distributed system, we are going to have a bad time.
- Even something as simple as *shared clocks* become an issue!

For computer scientists

- Whenever we need *any* agreement on state between components in a distributed system, we are going to have a bad time.
- Even something as simple as *shared clocks* become an issue!
- Directly leads to two famous results: the FLP impossibility theorem (Fischer, Lynch and Paterson, 1985) and the CAP theorem (Brewer, 1998).

For computer scientists

- Whenever we need *any* agreement on state between components in a distributed system, we are going to have a bad time.
- Even something as simple as *shared clocks* become an issue!
- Directly leads to two famous results: the FLP impossibility theorem (Fischer, Lynch and Paterson, 1985) and the CAP theorem (Brewer, 1998).
- A lot of work on singular systems is next-to-worthless for distributed ones.

For computer scientists

- Whenever we need *any* agreement on state between components in a distributed system, we are going to have a bad time.
- Even something as simple as *shared clocks* become an issue!
- Directly leads to two famous results: the FLP impossibility theorem (Fischer, Lynch and Paterson, 1985) and the CAP theorem (Brewer, 1998).
- A lot of work on singular systems is next-to-worthless for distributed ones.
- There are lots of ways to obtain coherency and agreement in distributed systems, but *none* will be as perfect as a singular one (although you might not care in some cases).

For practitioners

- When we need agreement or synchronization, there *will* be serious costs that have to be considered.

For practitioners

- When we need agreement or synchronization, there *will* be serious costs that have to be considered.
- If you see (or need to be involved with) any distributed system with heavy amounts of global information that must be kept coherent, *run*.

For practitioners

- When we need agreement or synchronization, there *will* be serious costs that have to be considered.
- If you see (or need to be involved with) any distributed system with heavy amounts of global information that must be kept coherent, *run*.
- Know your tradeoffs — if you can avoid needing a lot of synchronization or information sharing, definitely avoid it, or at least limit its impact.

For practitioners

- When we need agreement or synchronization, there *will* be serious costs that have to be considered.
- If you see (or need to be involved with) any distributed system with heavy amounts of global information that must be kept coherent, *run*.
- Know your tradeoffs — if you can avoid needing a lot of synchronization or information sharing, definitely avoid it, or at least limit its impact.
- Understand what the computer scientists have said about distributed systems — it's not nearly as theoretical as you think, and it might save your job one day.

A final thought

> *"You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done."*
>
> Leslie Lamport

Questions