6.033 in the news

Beavers shut down town’s internet for 36 hours after chewing through and stealing cables to build a dam

BY SOPHIE LEWIS
APRIL 26, 2021 / 12:18 PM / CBS NEWS


The Global Internet Is Being Attacked by Sharks, Google Confirms

BY WILL OREMUS
AUG 15, 2014 • 3:23 PM

Sharks’ attraction to undersea fiber-optic cables has been well-documented over the years.


Content providers like Microsoft, Google, Facebook and Amazon now own or lease more than half of the undersea bandwidth

Source: Tel Geography


Lecture #19: Replicated State Machines

high availability + single-copy consistency
our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high.

transactions — which provide **atomicity** and **isolation** — make it easier for us to reason about failures.

our job in lecture is to understand how a system implements these two abstractions. how do our systems guarantee atomicity? how do they guarantee isolation?

**atomicity**: provided by **logging**, which gives better performance than shadow copies* at the cost of some added complexity; **two-phase commit** gives us multi-site atomicity.

**isolation**: provided by **two-phase locking**

* shadow copies are used in some systems.
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data

client | coordinator | A-M server | A-M server

A-amount
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data

| client | coordinator | A-M server | A-M server | coordinator | client |
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data

A=20
ok

A=30
ok
to increase availability, let’s try replicating data on two servers

attempt 1: nothing special, just two copies of the data

result: A=30

result: A=20
to increase availability, let's try replicating data on two servers

attempt 1: nothing special, just two copies of the data

result: $A = 30$
result: $A = 20$

**problem:** replica servers can become inconsistent
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures
to increase availability, let's try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

clients communicate only with C, not with replicas
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

clients communicate only with C, not with replicas

C sends requests to primary server
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have a coordinator in place to help manage failures

**primary** chooses order of operations, decides all non-deterministic values

clients communicate only with **C**, not with replicas

**C** sends requests to **primary** server

**primary** sends requests to **S1**

**S1** (primary)

**S2** (backup)
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

clients communicate only with C, not with replicas

C sends requests to primary server

primary chooses order of operations, decides all non-deterministic values

primary ACKs coordinator only after it’s sure that backup has all updates

primary

C

S1

S2

(typical)

(backup)
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

if primary fails, C switches to backup
C knows how to contact backup server

clients communicate only with C, not with replicas

C sends requests to primary server

primary chooses order of operations, decides all non-deterministic values
primary ACKs coordinator only after it’s sure that backup has all updates

primary

S1

S2

C

backup

(backup)

(secondary)

Katrina LaCurts | lacurts@mit.edu | 6.033 2021
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have a coordinator in place to help manage failures

if **primary** fails, **C** switches to **backup**

**C** knows how to contact backup server

clients communicate only with **C**, not with replicas

**C** sends requests to **primary** server

**S_2** (backup)

(failed)
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

if primary fails, C switches to backup
  C knows how to contact backup server

clients communicate only with C, not with replicas

C sends requests to primary server

S₂ (primary)

(failed)

ideally, S₁ recovers at some point, or we get some other replacement machine, and we go back to having both a primary and a backup. but for the purposes of this example, we’re just concerned about correctly switching over to the backup server
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

for a single transaction, a client would communicate with a single coordinator
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have a coordinator in place to help manage failures

for a single transaction, a client would communicate with a single **coordinator**
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

for a single transaction, a client would communicate with a single coordinator
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have a coordinator in place to help manage failures

for a single transaction, a client would communicate with a single **coordinator**

suppose that all machines remain up, but that there is a **network partition** that effectively splits this network in half
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

suppose that all machines remain up, but that there is a network partition that effectively splits this network in half
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

for a single transaction, a client would communicate with a single coordinator

suppose that all machines remain up, but that there is a network partition that effectively splits this network in half

a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

for a single transaction, a client would communicate with a single coordinator

C₁ keeps using S₁ as primary, with no backup

a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side

suppose that all machines remain up, but that there is a network partition that effectively splits this network in half
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

for a single transaction, a client would communicate with a single coordinator

$C_1$ keeps using $S_1$ as primary, with no backup

a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side

suppose that all machines remain up, but that there is a network partition that effectively splits this network in half
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the primary replica, and have a coordinator in place to help manage failures

for a single transaction, a client would communicate with a single coordinator

a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side

suppose that all machines remain up, but that there is a network partition that effectively splits this network in half

C₁ keeps using S₁ as primary, with no backup

C₂ begins using S₂ as primary, with no backup

Katrina LaCurts | lacurts@mit.edu | 6.033 2021
to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have a coordinator in place to help manage failures

**C**

to increase availability, let’s try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have a coordinator in place to help manage failures

for a single transaction, a client would communicate with a single **coordinator**

**C**

because two different replicas both think that they are the primary replica, our data can become inconsistent
to increase availability, let’s try replicating data on two servers

try 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let's find out)

view server keeps a table that maintains a sequence of views

- S1
- S2
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

**view server** keeps a table that maintains a sequence of views
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

view server keeps a table that maintains a sequence of views

| view server alerts | primary/backups about their roles |

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

C

S1

S2
to increase availability, let’s try replicating data on two servers

try

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

view server keeps a table that maintains a sequence of views

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

**view server** alerts primary/backups about their roles
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

view server alerts primary/backups about their roles

view server keeps a table that maintains a sequence of views

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

```
<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>
```

**view server** keeps a table that maintains a sequence of views

**view server** alerts primary/backups about their roles

**primary** sends updates to, gets ACKs from **backup** (as before)
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

**view server** keeps a table that maintains a sequence of views

- **primary** sends updates to, gets ACKs from **backup** (as before)
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

 coordinators make requests to **view server** to find out which replica is primary

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

**view server** keeps a table that maintains a sequence of views

**primary** sends updates to, gets ACKs from **backup** (as before)
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

coordinators make requests to **view server** to find out which replica is **primary**

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

**view server** keeps a table that maintains a sequence of views

*primary* sends updates to, gets ACKs from **backup** (as before)
to increase availability, let’s try replicating data on two servers

try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

coordinators make requests to view server to find out which replica is primary

view server keeps a table that maintains a sequence of views

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

primary sends updates to, gets ACKs from backup (as before)
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

coordinators make requests to view server to find out which replica is primary

view server keeps a table that maintains a sequence of views

coordinators contact primary (as before)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

primary sends updates to, gets ACKs from backup (as before)
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let's find out)

**coordinators** make requests to **view server** to find out which replica is **primary**

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

**primary** sends updates to, gets ACKs from **backup** (as before)

**view server** keeps a table that maintains a sequence of views
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

**view server** keeps a table that maintains a sequence of views

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

**coordinators** make requests to **view server** to find out which replica is **primary**

**coordinators** contact **primary** (as before)

**primary** sends updates to, **primary** gets ACKs from **backup** (as before)

**primary/backup** ping **view server** so that **view server** can discover failures

**view server** keeps a table that maintains a sequence of views
to increase availability, let’s try replicating data on two servers
attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

coordinators make requests to view server to find out which replica is primary

view server keeps a table that maintains a sequence of views

view # | primary | backup
1     S1        S2

coordinators contact primary (as before)

S1 (primary)

primary/backup ping view server so that view server can discover failures

S2 (backup)

primary sends updates to, gets ACKs from backup (as before)
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if the primary replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

What happens if the primary replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

lack of pings indicates to VS that $S_1$ is down

what happens if the primary replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

```
view # | primary | backup
1     S1     S2 (failed)
2     S2     S1 (backup)
```

lack of pings indicates to VS that S₁ is down

what happens if the primary replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td></td>
</tr>
</tbody>
</table>

what happens if the primary replica fails?

lack of pings indicates to VS that S₁ is down.
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

if \( C \) communicates with \( S_1 \), \( C \) won’t get a response; when \( C \) next asks \( VS \) who the primary is, \( VS \) will respond with \( S_2 \)

lack of pings indicates to \( VS \) that \( S_1 \) is down

what happens if the primary replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

if \( C \) communicates with \( S_1 \), \( C \) won’t get a response; when \( C \) next asks \( VS \) who the primary is, \( VS \) will respond with \( S_2 \)

lack of pings indicates to \( VS \) that \( S_1 \) is down

notice there is no longer a backup. once again, we’d hope to eventually bring \( S_1 \) back online, or find a new machine to act as a backup. but in this example, we’re only interested in safely making \( S_2 \) the new primary.

what happens if the primary replica fails?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if a network partition prevents $S_1$ from communicating with VS?
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if a network partition prevents $S_1$ from communicating with VS?
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if a network partition prevents $S_1$ from communicating with VS?

in a sense, this is the worst possible partition: VS is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except VS
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

view # | primary | backup
-------|---------|---------
1       | S1      | S2      

lack of pings indicates to VS that $S_1$ is down

what happens if a network partition prevents $S_1$ from communicating with VS?

in a sense, this is the worst possible partition: VS is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except VS
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

![Diagram showing view server and two servers, S1 and S2, with VS making S2 primary]

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td></td>
</tr>
</tbody>
</table>

lack of pings indicates to VS that S₁ is down

VS makes S₂ primary

what happens if a network partition prevents S₁ from communicating with VS?

In a sense, this is the worst possible partition: VS is going to presume S₁ has failed (and so switch to using S₂ as a backup), while S₁ can still communicate with everyone *except* VS.
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if a network partition prevents $S_1$ from communicating with VS?

in a sense, this is the worst possible partition: VS is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except VS
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

**Diagram**

- **C**: Central Coordinator
- **S1**: Server 1
- **S2**: Server 2
- **VS**: View Server

<table>
<thead>
<tr>
<th>view</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>S2</td>
</tr>
</tbody>
</table>

lack of pings indicates to VS that S1 is down

VS makes S2 primary

at this stage, VS thinks S2 is primary; S2 and S1 think S1 is primary

what happens if a network partition prevents S1 from communicating with VS?

in a sense, this is the worst possible partition: VS is going to presume S1 has failed (and so switch to using S2 as a backup), while S1 can still communicate with everyone except VS
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

![Diagram of network setup]

what happens if a network partition prevents $S_1$ from communicating with $VS$?

in a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone **except $VS$**
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if a network partition prevents $S_1$ from communicating with $VS$?

in a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td></td>
</tr>
</tbody>
</table>

at this stage, **VS** thinks **S2** is primary; **S2** and **S1** think **S1** is primary

if **S1** receives any requests from **C**, it will behave as primary with **S2** as backup

if **S2** receives any requests from **C**, it will reject them; it believes that it is the backup (and so does not communicate directly with **C**)

**what happens if a network partition prevents **S1** from communicating with **VS**?**

in a sense, this is the worst possible partition: **VS** is going to presume **S1** has failed (and so switch to using **S2** as a backup), while **S1** can still communicate with everyone except **VS**
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

![Diagram showing data flow between servers](image)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>S1</td>
</tr>
</tbody>
</table>

at this stage, **VS** thinks **S2** is primary; **S2** and **S1** think **S1** is primary

if **S1** receives any requests from **C**, it will behave as primary with **S2** as backup

if **S2** receives any requests from **C**, it will reject them; it believes that it is the backup (and so does not communicate directly with **C**)

**new detail**: backups **reject** any requests from coordinators

what happens if a network partition prevents **S1** from communicating with **VS**?

in a sense, this is the worst possible partition: **VS** is going to presume **S1** has failed (and so switch to using **S2** as a backup), while **S1** can still communicate with everyone except **VS**
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if a network partition prevents $S_1$ from communicating with $VS$?

in a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

![Diagram showing network partitions and view server](image)

### What happens if a network partition prevents $S_1$ from communicating with VS?

In a sense, this is the worst possible partition: VS is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except VS.
to increase availability, let’s try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if a network partition prevents $S_1$ from communicating with $VS$?

in a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if a network partition prevents $S_1$ from communicating with $VS$?

in a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$
to increase availability, let’s try replicating data on two servers

try attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if a network partition prevents S₁ from communicating with VS?

in a sense, this is the worst possible partition: VS is going to presume S₁ has failed (and so switch to using S₂ as a backup), while S₁ can still communicate with everyone except VS

at this stage, VS and S₂ think S₂ is primary; S₁ thinks S₁ is primary

if S₁ receives any requests from C, it won’t be able to get an ACK from S₂, and so will reject

if S₂ receives any requests from C, it will respond as the primary (in line with what VS expects)
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

**important rule**: if a machine is primary in view \( n \), it must have been primary or backup in view \( n-1 \) (with the exception of view 1, when we’re just starting)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>S1</td>
</tr>
</tbody>
</table>

once \( S_1 \) is can communicate with \( VS \) again, \( VS \) will respond notifying it that it is *not* in the current view

at this stage, \( VS \) and \( S_2 \) think \( S_2 \) is primary; \( S_1 \) thinks \( S_1 \) is primary

if \( S_1 \) receives any requests from \( C \), it won’t be able to get an ACK from \( S_2 \), and so will reject

if \( S_2 \) receives any requests from \( C \), it will respond as the primary (in line with what \( VS \) expects)

what happens if a network partition prevents \( S_1 \) from communicating with \( VS \)?

in a sense, this is the worst possible partition: \( VS \) is going to presume \( S_1 \) has failed (and so switch to using \( S_2 \) as a backup), while \( S_1 \) can still communicate with everyone except \( VS \)
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

**important rule:** if a machine is primary in view $n$, it must have been primary or backup in view $n-1$ (with the exception of view 1, when we’re just starting)

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$S_1$</td>
<td>$S_2$</td>
</tr>
<tr>
<td>2</td>
<td>$S_2$</td>
<td>$S_1$</td>
</tr>
</tbody>
</table>

Once $S_1$ is can communicate with $VS$ again, $VS$ will respond notifying it that it is *not* in the current view

At this stage, $VS$ and $S_2$ think $S_2$ is primary; $S_1$ thinks $S_1$ is primary

What happens if a network partition prevents $S_1$ from communicating with $VS$?

In a sense, this is the worst possible partition: $VS$ is going to presume $S_1$ has failed (and so switch to using $S_2$ as a backup), while $S_1$ can still communicate with everyone except $VS$

New detail: primaries reject any updates from other replicas

If $S_1$ receives any requests from $C$, it won’t be able to get an ACK from $S_2$, and so will reject

If $S_2$ receives any requests from $C$, it will respond as the primary (in line with what $VS$ expects)

Once $S_1$ is can communicate with $VS$ again, $VS$ will respond notifying it that it is not in the current view
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions (can we? let's find out)
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if VS fails?

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

S1 (primary)

S2 (backup)
to increase availability, let’s try replicating data on two servers

attempt 3: use a view server to determine which replica is primary, in hopes that we can deal with network partitions (can we? let’s find out)

what happens if VS fails?

find out in tomorrow’s recitation :)
coordinators make requests to view server to find out which replica is primary.

view server keeps a table that maintains a sequence of views:

<table>
<thead>
<tr>
<th>view #</th>
<th>primary</th>
<th>backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

coordinators contact primary (as before)

primary sends updates to, gets ACKs from backup (as before)

primary/backup ping view server so that view server can discover failures

view server keeps a table that maintains a sequence of views.

if a machine is primary in view n, it must have been primary or backup in view n-1 (with the exception of view 1, when we're just starting).

backups will reject any requests that they get directly from coordinators; primary will reject any update that comes from a backup.

(both of these events can happen in the case of certain types of failures)
our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high

**transactions** — which provide **atomicity** and **isolation** — make it easier for us to reason about failures

our job in lecture is to understand how a system *implements* these two abstractions. how do our systems guarantee atomicity? how do they guarantee isolation?

**atomicity**: provided by **logging**, which gives better performance than shadow copies* at the cost of some added complexity; **two-phase commit** gives us multi-site atomicity

**isolation**: provided by **two-phase locking**

* shadow copies are used in some systems
replicated state machines (RSMs) provide single-copy consistency: externally, it appears as if there is a single copy of the data, though internally there are replicas.

RSMs use a primary/backup mechanism for replication. The view server ensures that only one replica acts as the primary, and can recruit new backups if servers fail.

to extend this model to handle view-server failures, we need a mechanism to provide distributed consensus; see tomorrow’s recitation.