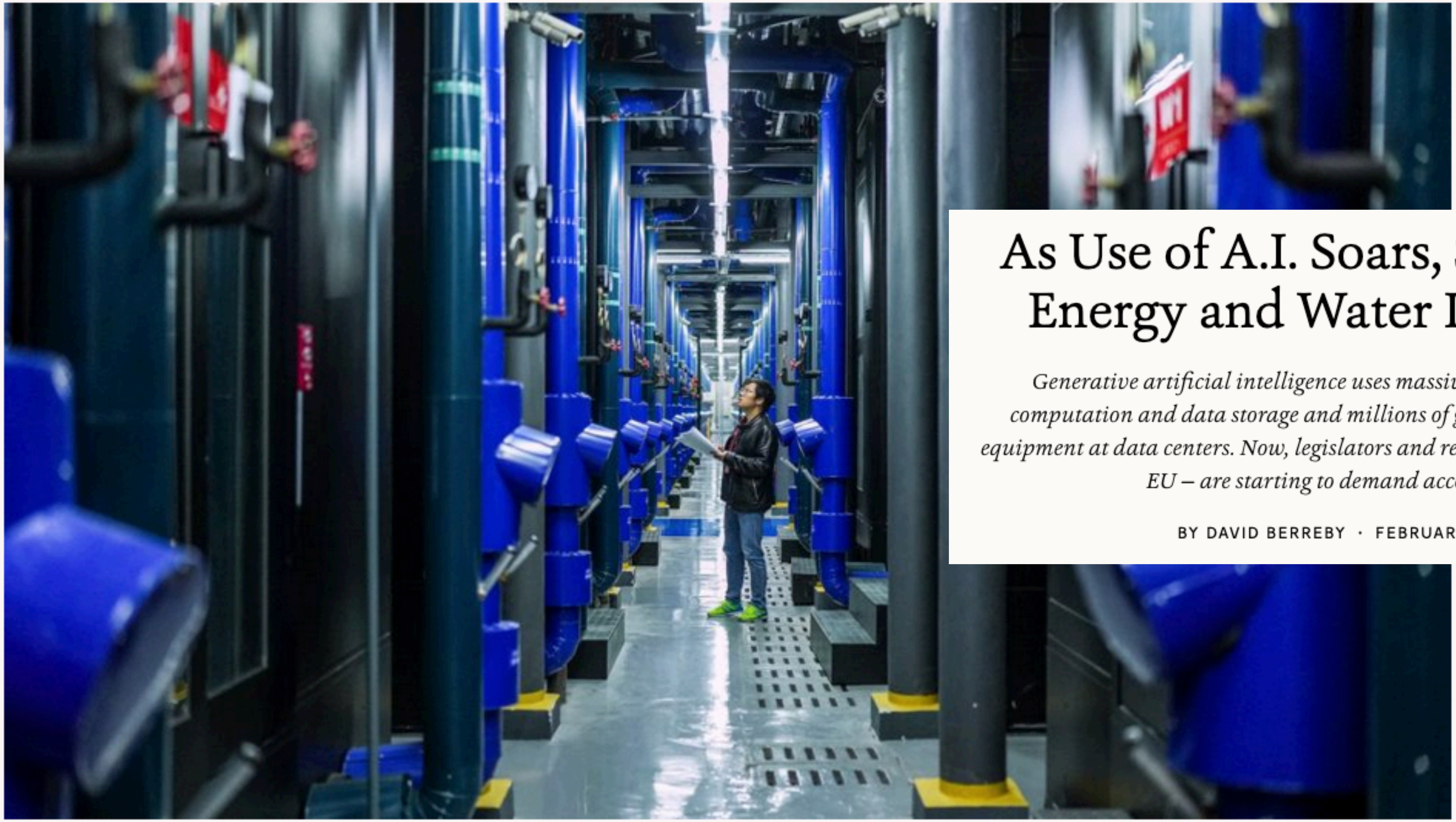
6.1800 Spring 2024

Lecture #20: Replicated State Machines

high availability + single-copy consistency

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024

6.1800 in the news



As Use of A.I. Soars, So Does the Energy and Water It Requires

Generative artificial intelligence uses massive amounts of energy for computation and data storage and millions of gallons of water to cool the equipment at data centers. Now, legislators and regulators – in the U.S. and the EU – are starting to demand accountability.

BY DAVID BERREBY · FEBRUARY 6, 2024

Inside the Guian Data Center of China Unicom, which uses artificial intelligence in its operations. TAO LIANG / XINHUA VIA GETTY IMAGES

https://e360.yale.edu/features/artificial-intelligence-climate-energy-emissions

6.1800 in the news

SpaceX's Starship Kicked Up a Dust Cloud, Leaving Texans With a Mess

Residents of Port Isabel said that their city was covered in grime following SpaceX's rocket launch on Thursday. The city said there was no "immediate concern for people's health."

Give this article

The SpaceX Starship test flight caused dust and debris to travel miles from the launch site in Boca Chica, Texas, on Tuesday. Abraham Pineda-Jacome/EPA, via Shutterstock

https://www.nytimes.com/2023/04/21/us/spacex-rocket-dust-texas.html

6.1800 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

https://www.cbsnews.com/news/beavers-shut-down-internet-tumbler-ridge-british-columbia-canada-chewing-stealing-cables-dam/

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.

https://slate.com/technology/2014/08/shark-attacks-threaten-google-s-undersea-internet-cables-video.html

6.1800 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

https://www.cbsnews.com/news/beavers-shut-down-internet-tumbler-ridge-british-columbia-canada-chewing-stealing-cables-dam/

how does the physical infrastructure of our systems impact the environment?

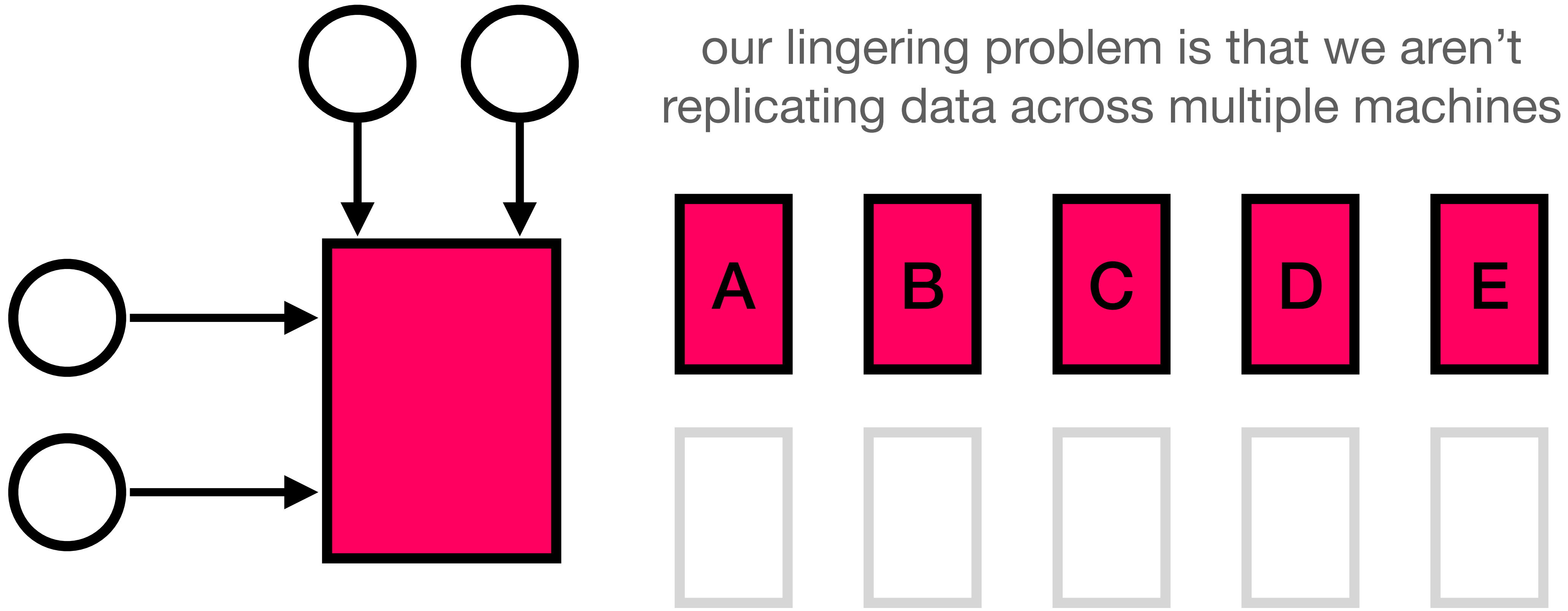
when is it harmful? can it be helpful?

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.

https://slate.com/technology/2014/08/shark-attacks-threaten-google-s-undersea-internet-cables-video.html 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



our lingering problem is that we aren't replicating data across multiple machines

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?

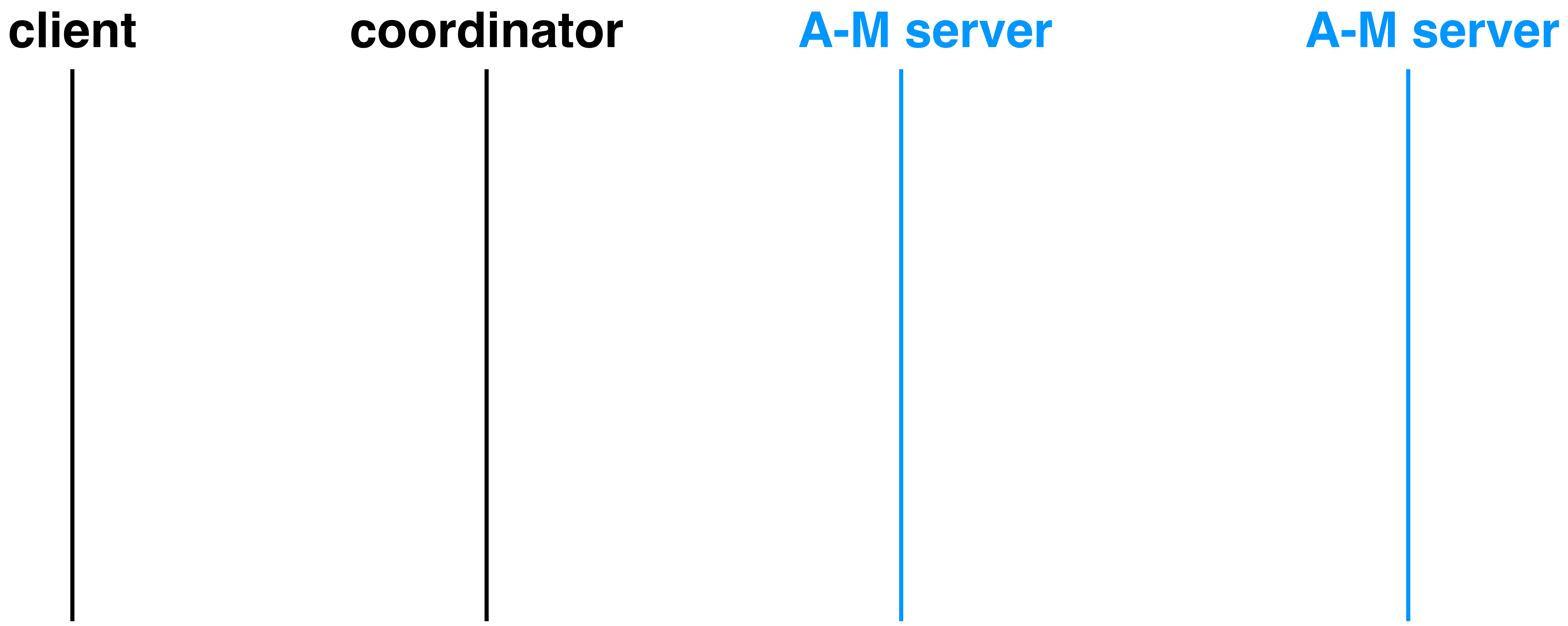
atomicity: provided by **logging**, which gives better performance than shadow copies* at the cost of some added complexity; **two-phase commit** gives us multi-site atomicity

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

* shadow copies *are* used in some systems

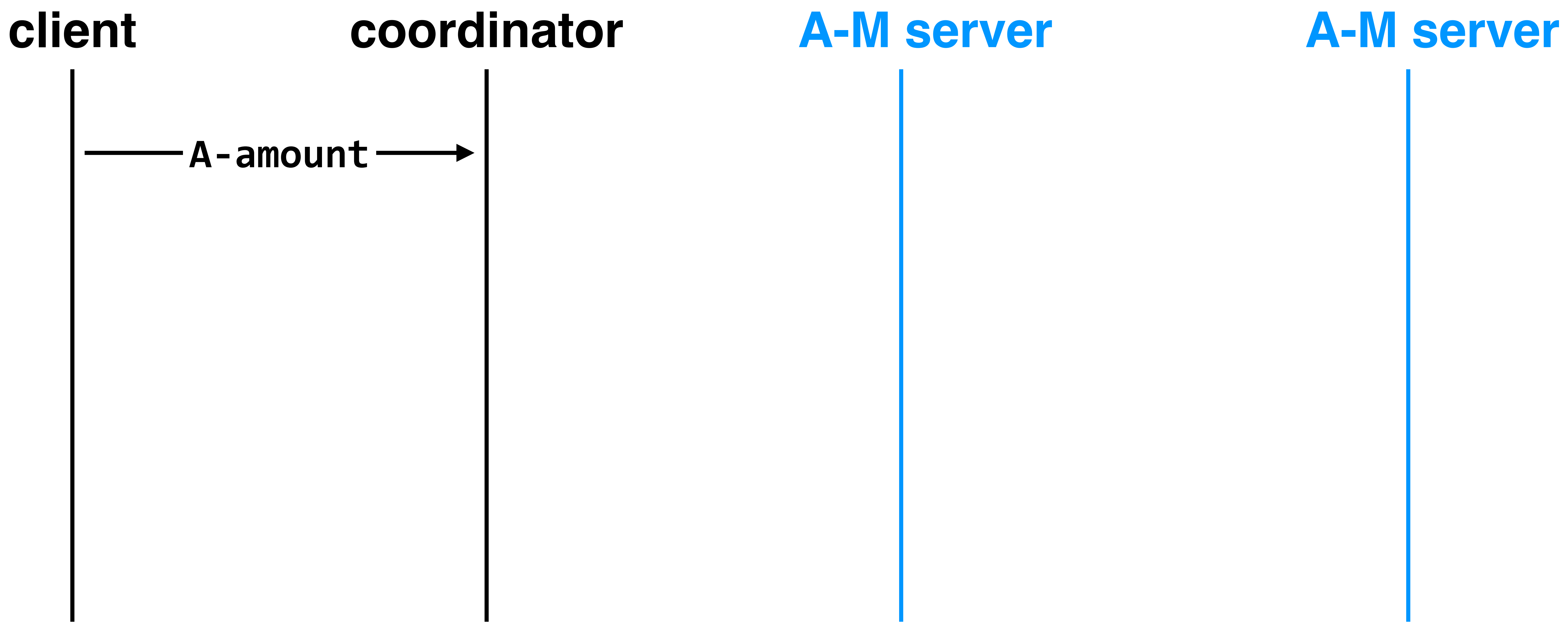
to increase availability, let's try replicating data on two servers

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



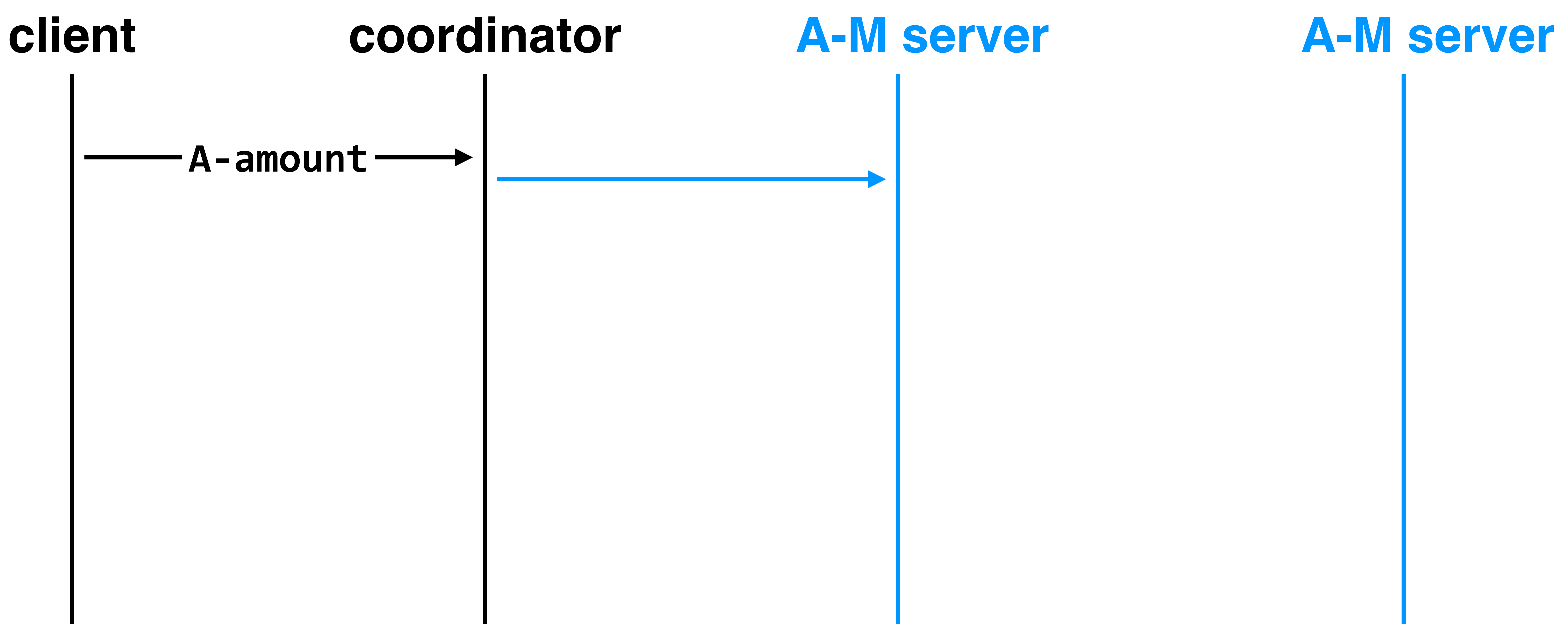
to increase availability, let's try replicating data on two servers

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



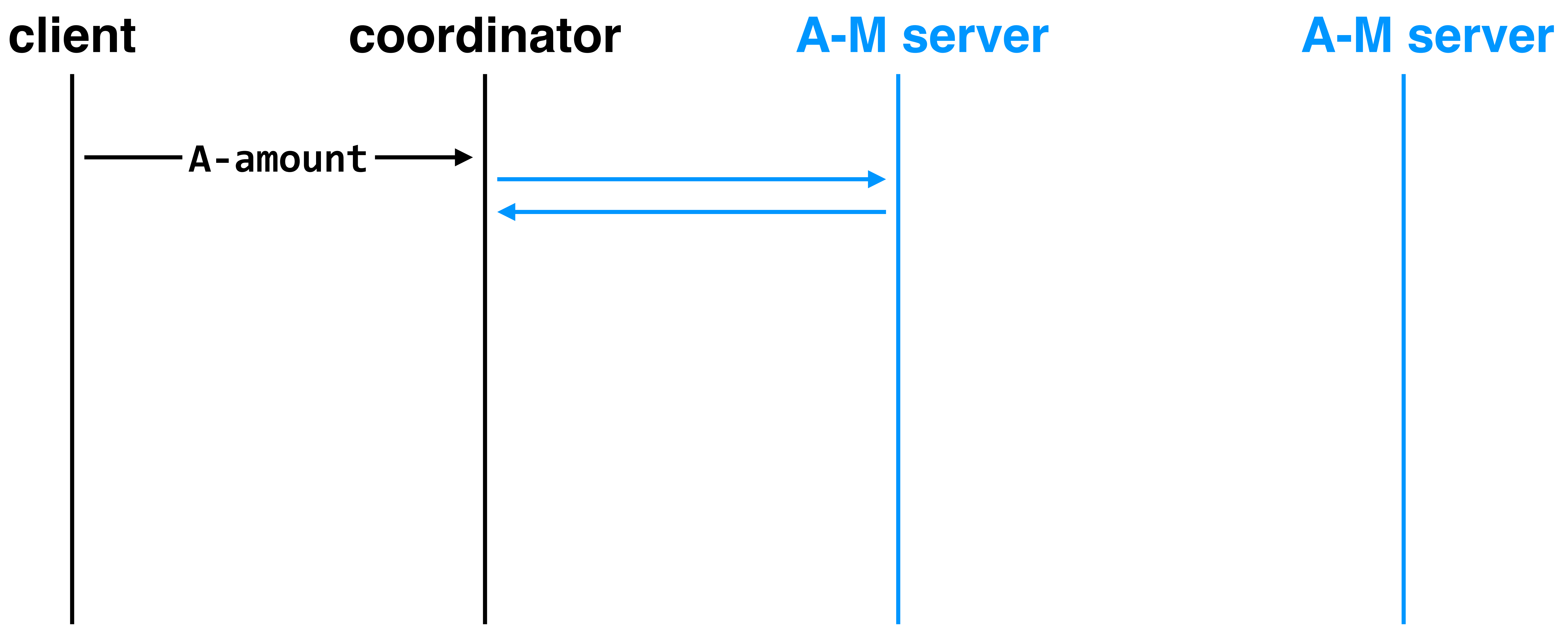
to increase availability, let's try replicating data on two servers

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



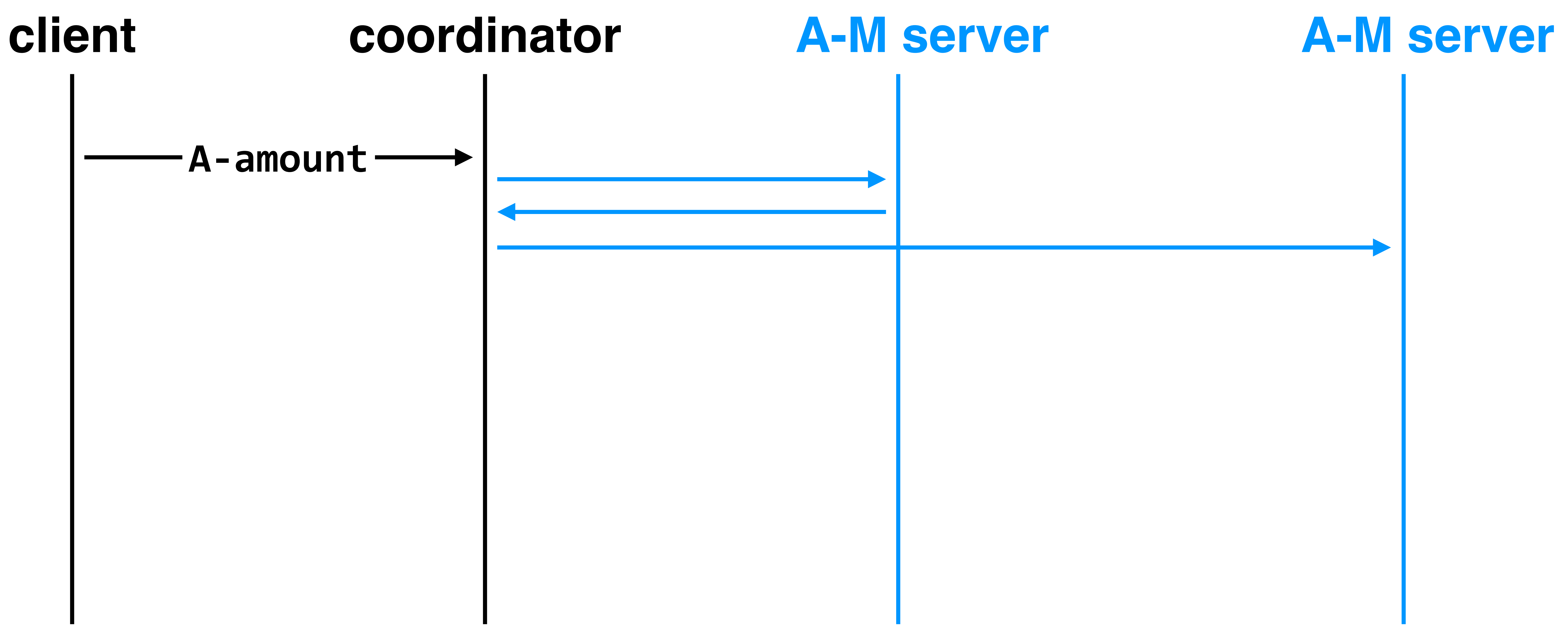
to increase availability, let's try replicating data on two servers

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



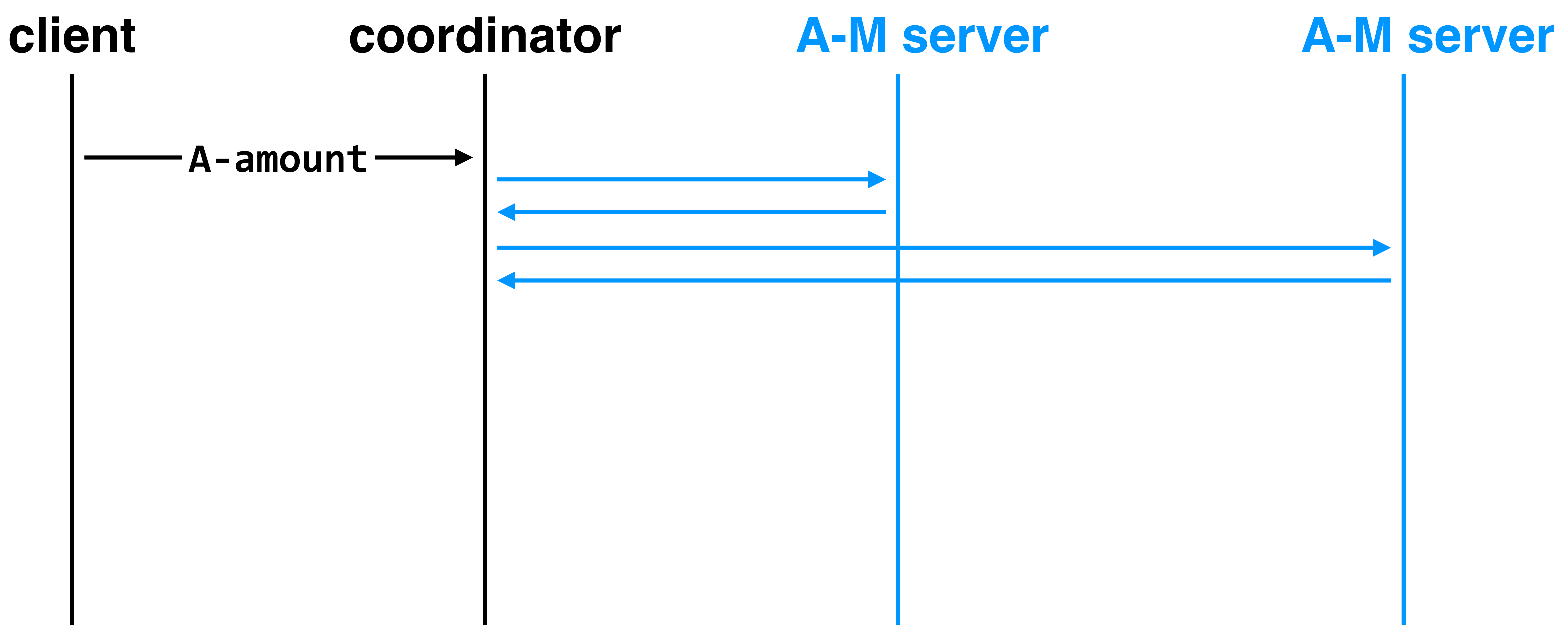
to increase availability, let's try replicating data on two servers

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



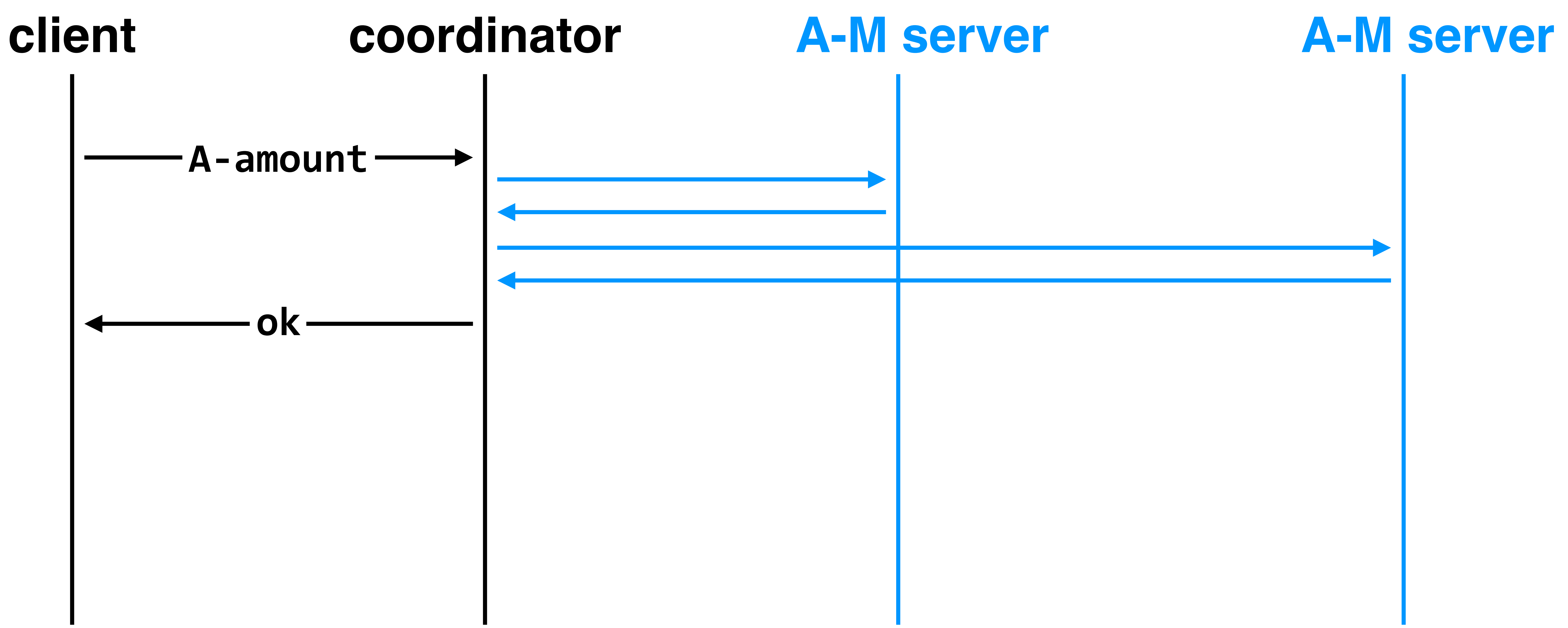
to increase availability, let's try replicating data on two servers

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



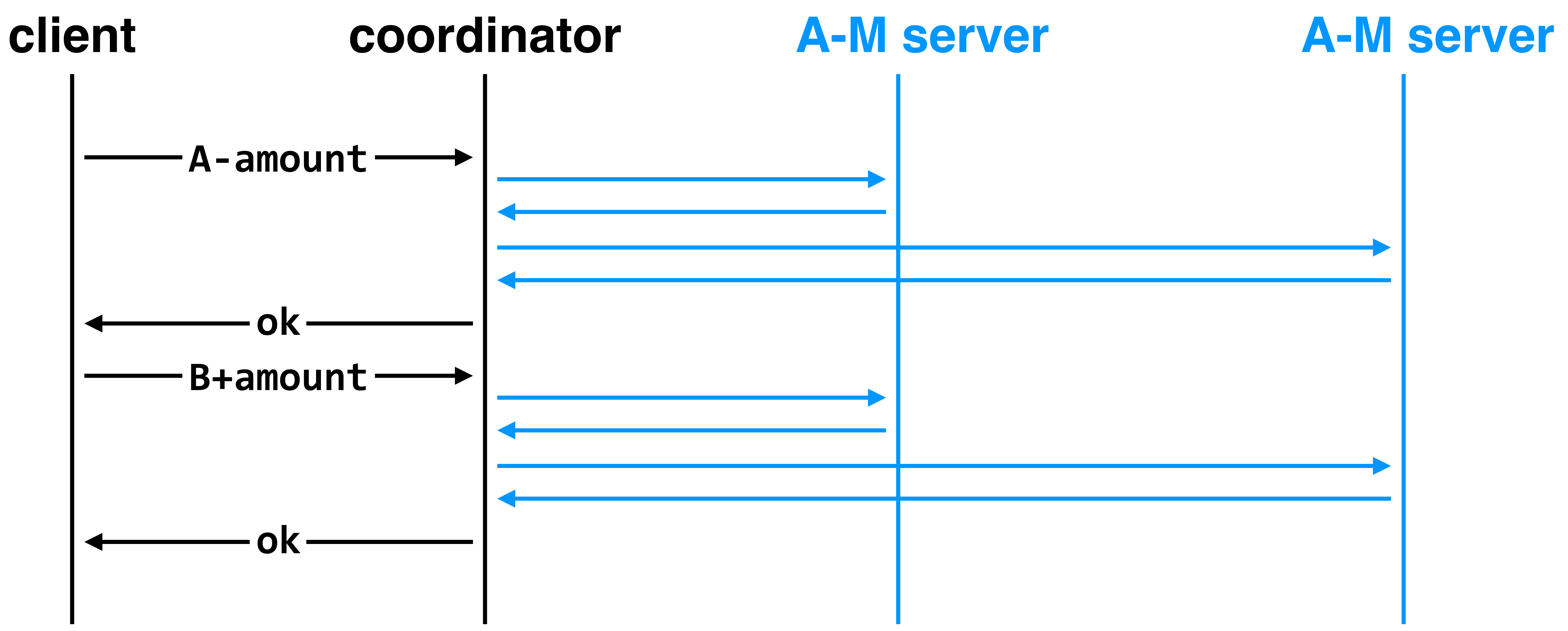
to increase availability, let's try replicating data on two servers

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



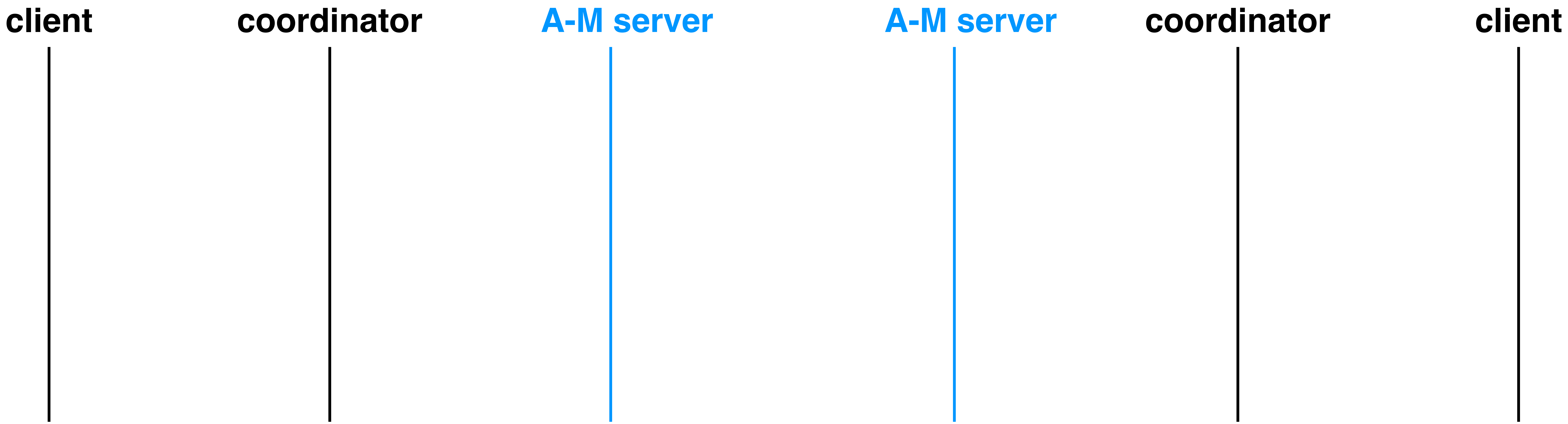
to increase availability, let's try replicating data on two servers

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



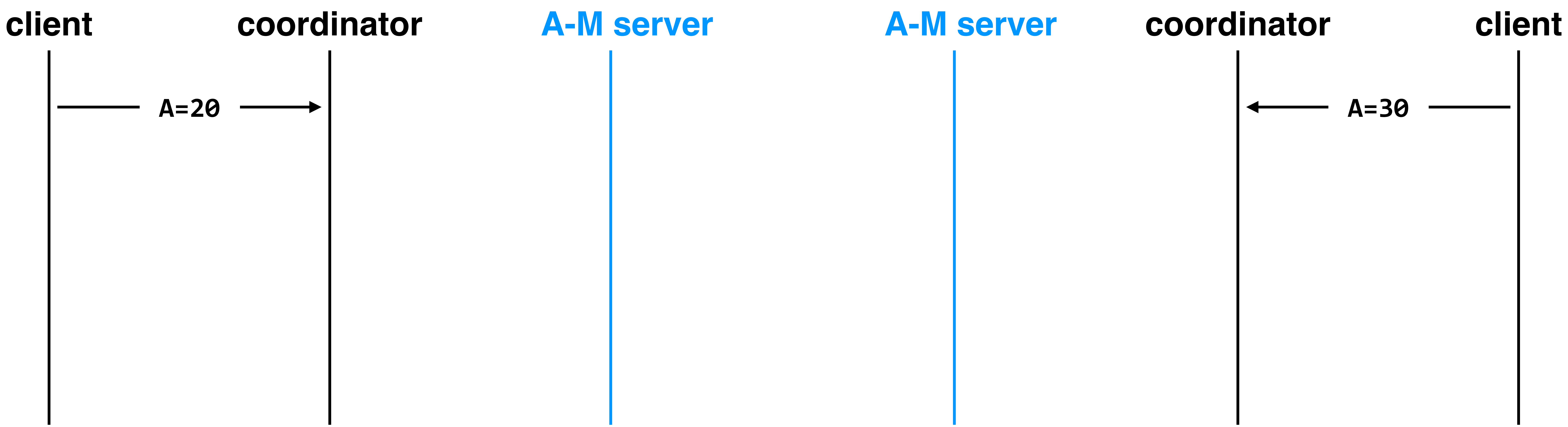
to increase availability, let's try replicating data on two servers

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



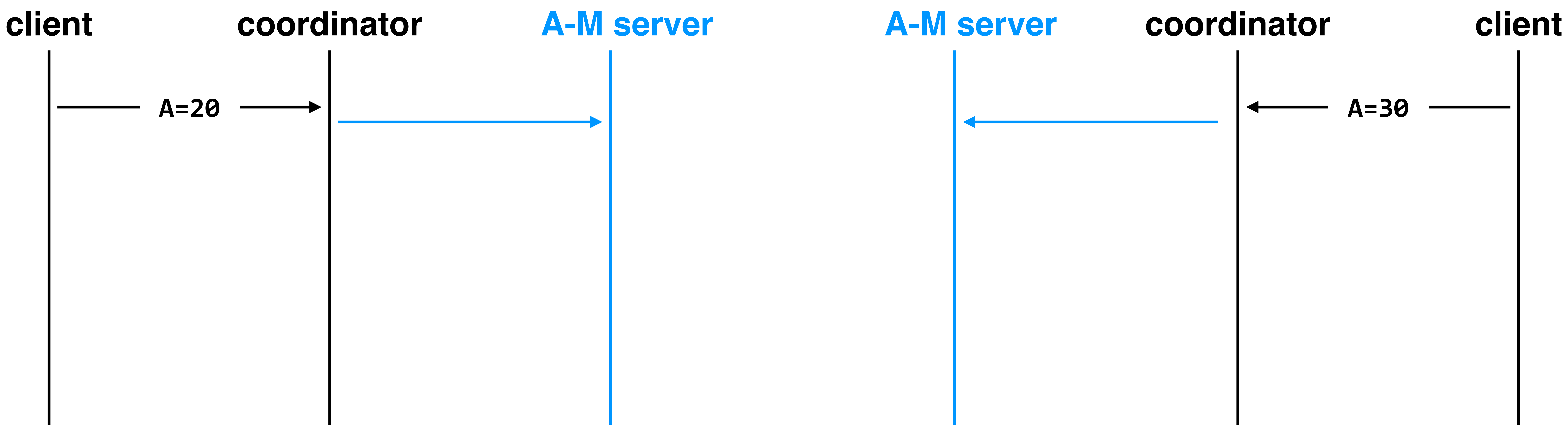
to increase availability, let's try replicating data on two servers

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



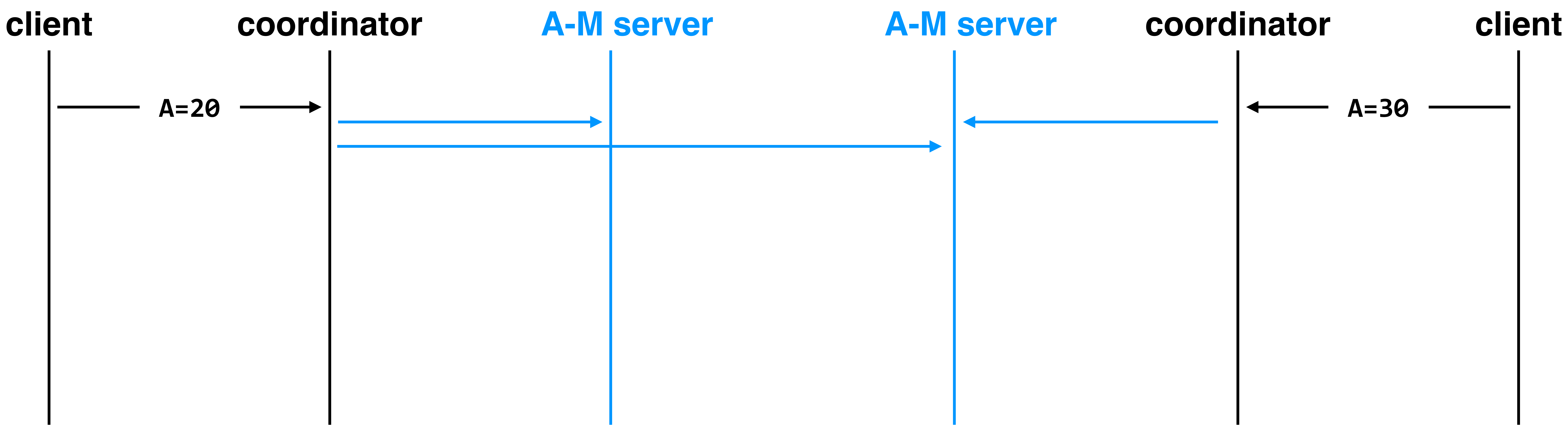
to increase availability, let's try replicating data on two servers

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



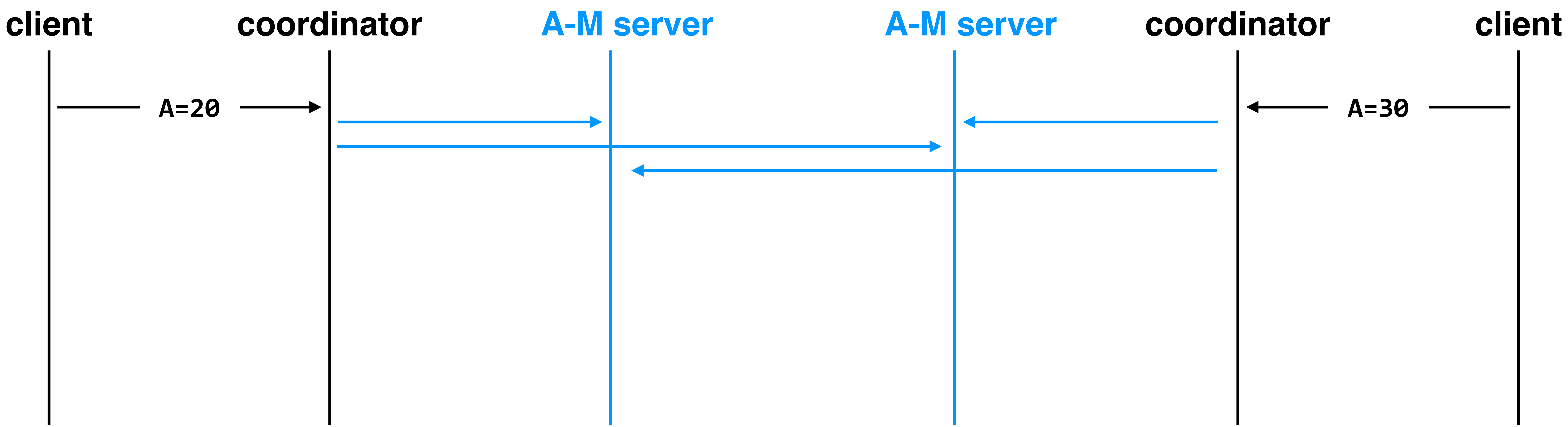
to increase availability, let’s try replicating data on two servers

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



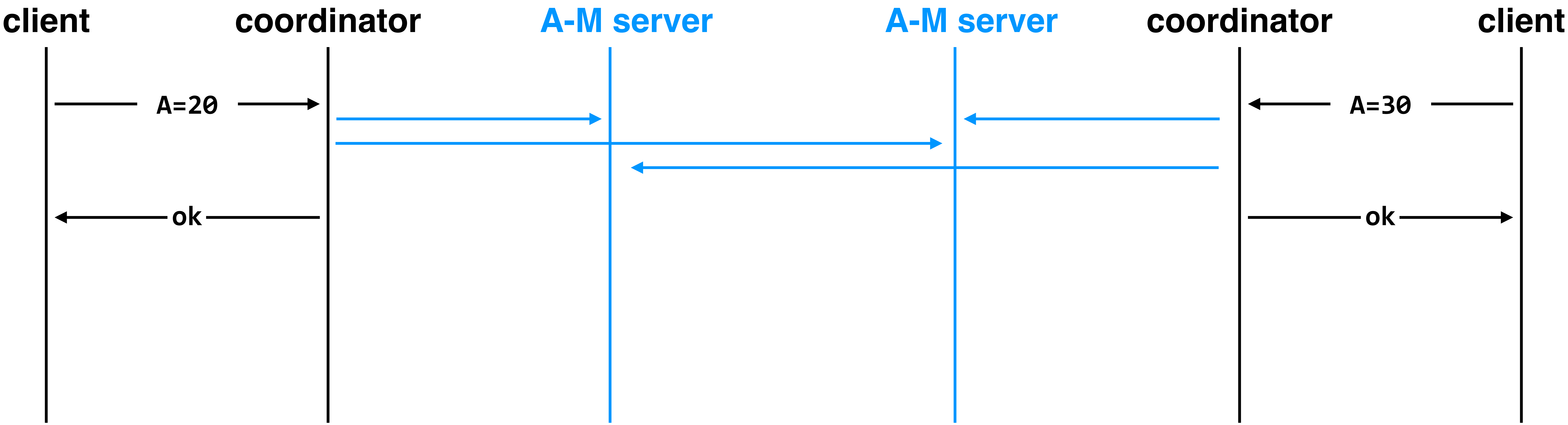
to increase availability, let's try replicating data on two servers

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



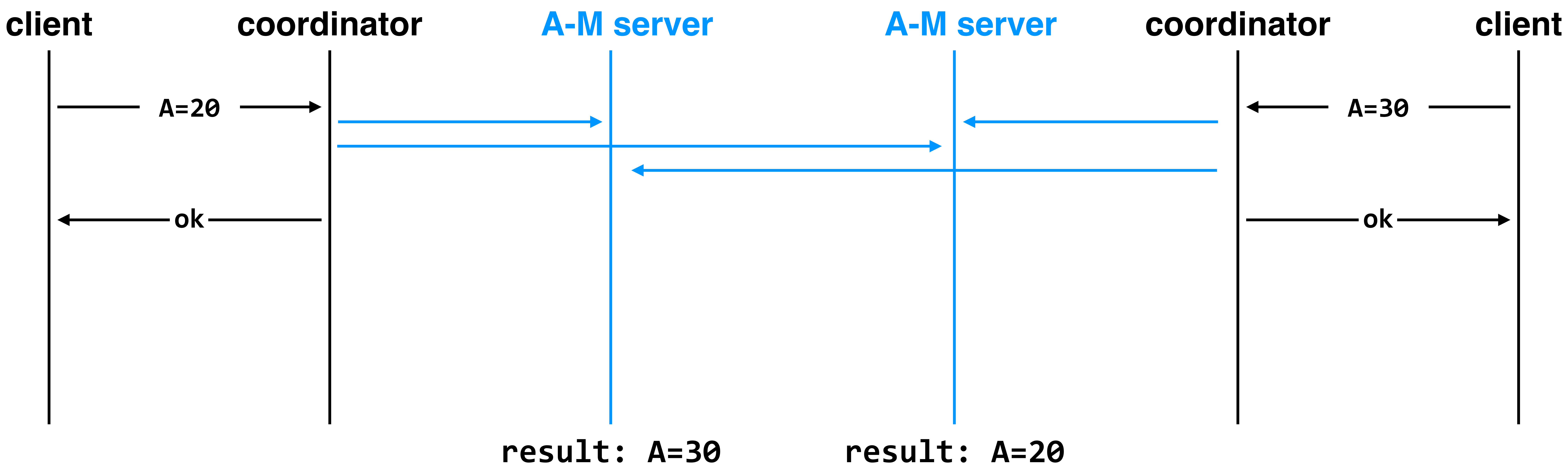
to increase availability, let's try replicating data on two servers

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



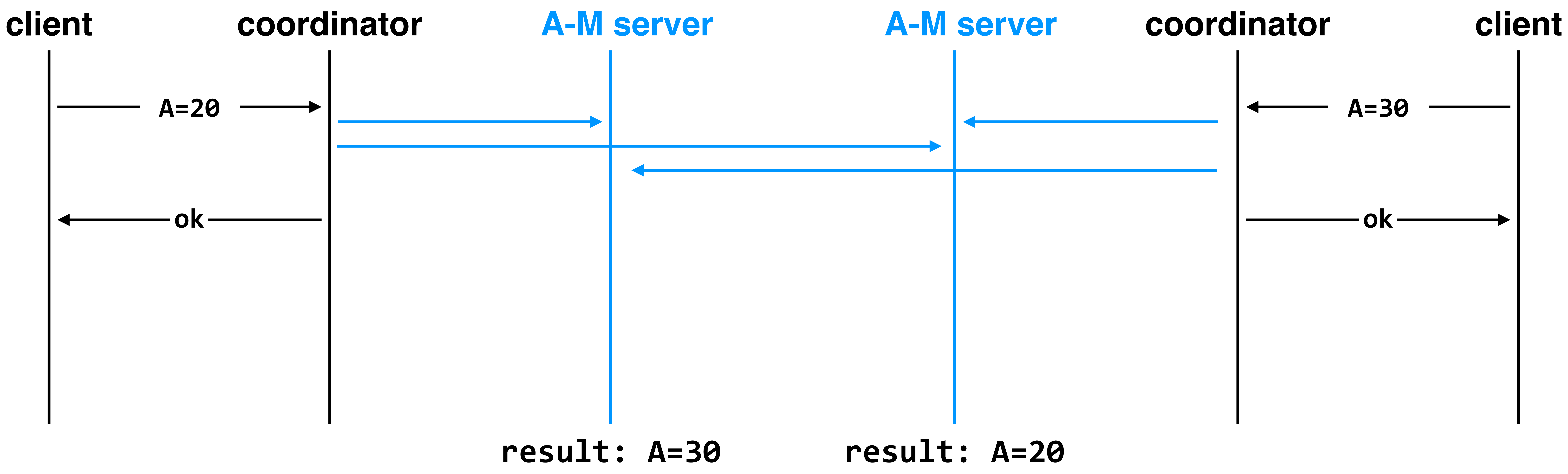
to increase availability, let's try replicating data on two servers

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



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

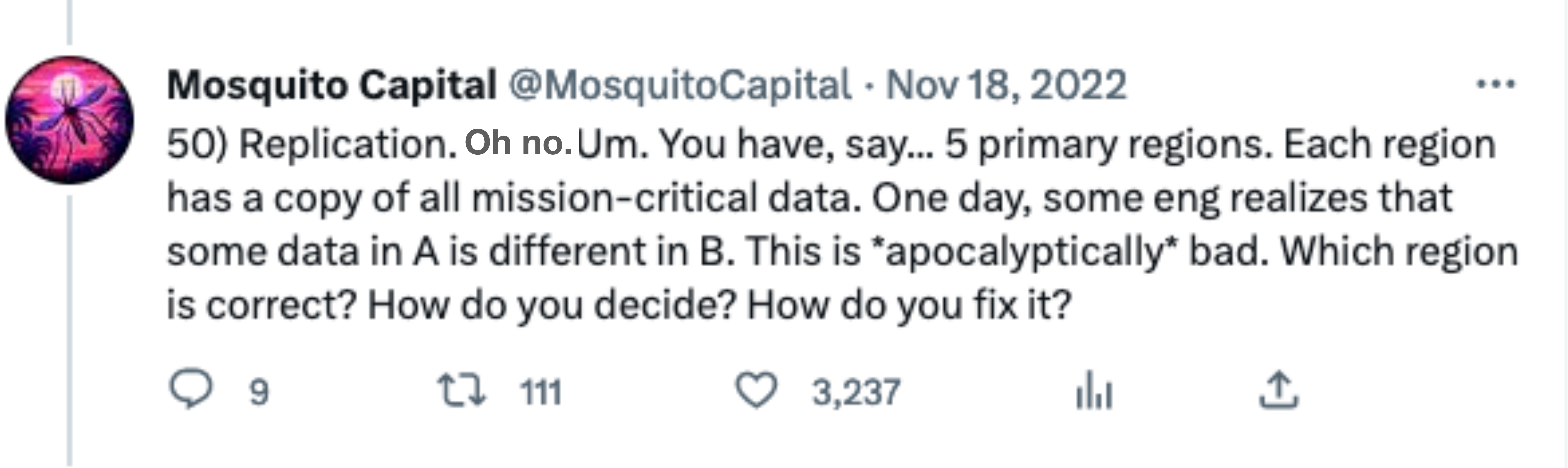
attempt 1: nothing special, just two copies of the data



problem: replica servers can become inconsistent

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

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



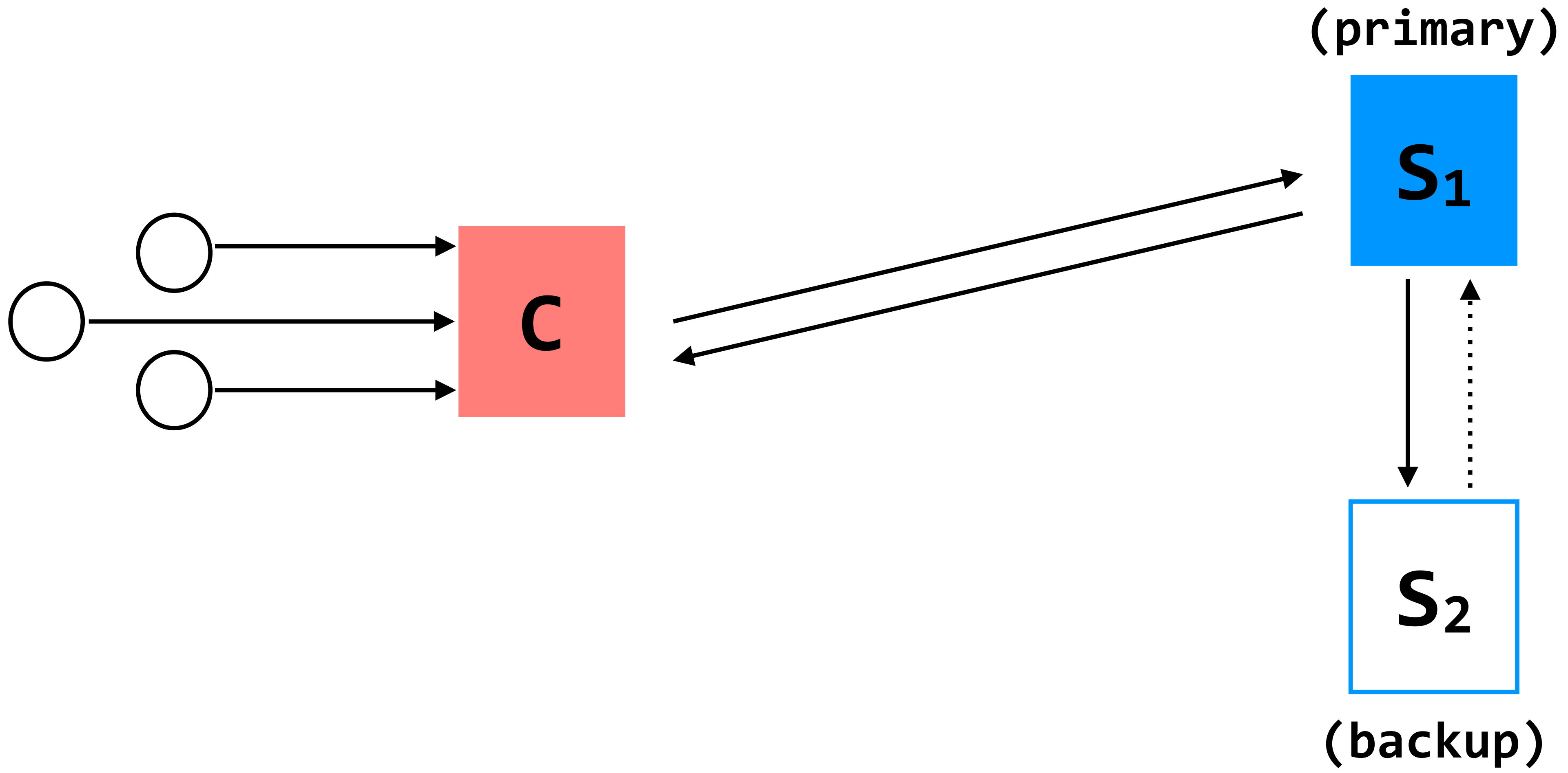
Mosquito Capital @MosquitoCapital · Nov 18, 2022

50) Replication. **Oh no.** Um. You have, say... 5 primary regions. Each region has a copy of all mission-critical data. One day, some eng realizes that some data in A is different in B. This is *apocalyptically* bad. Which region is correct? How do you decide? How do you fix it?

9 111 3,237

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

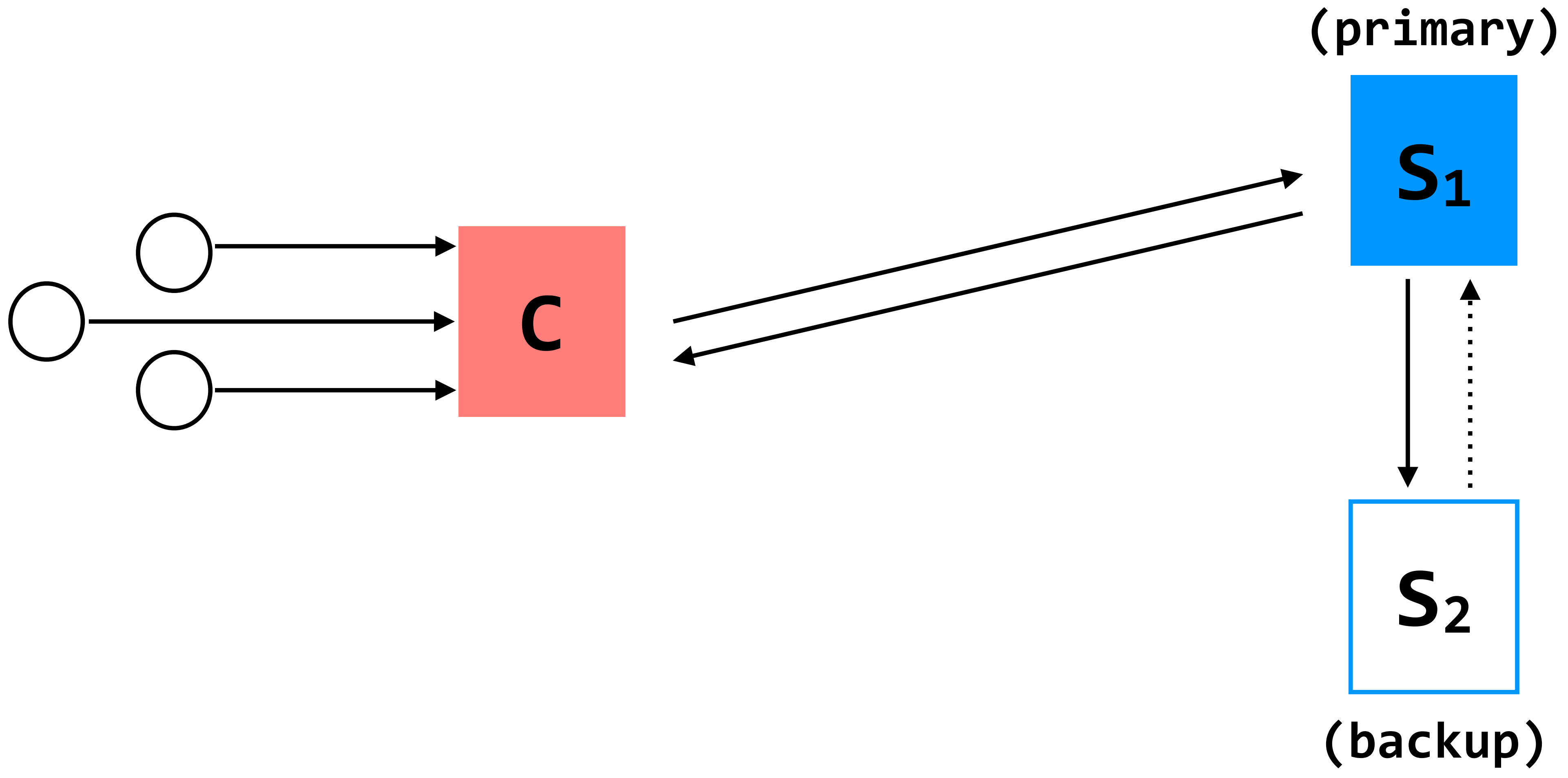
attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures



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

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

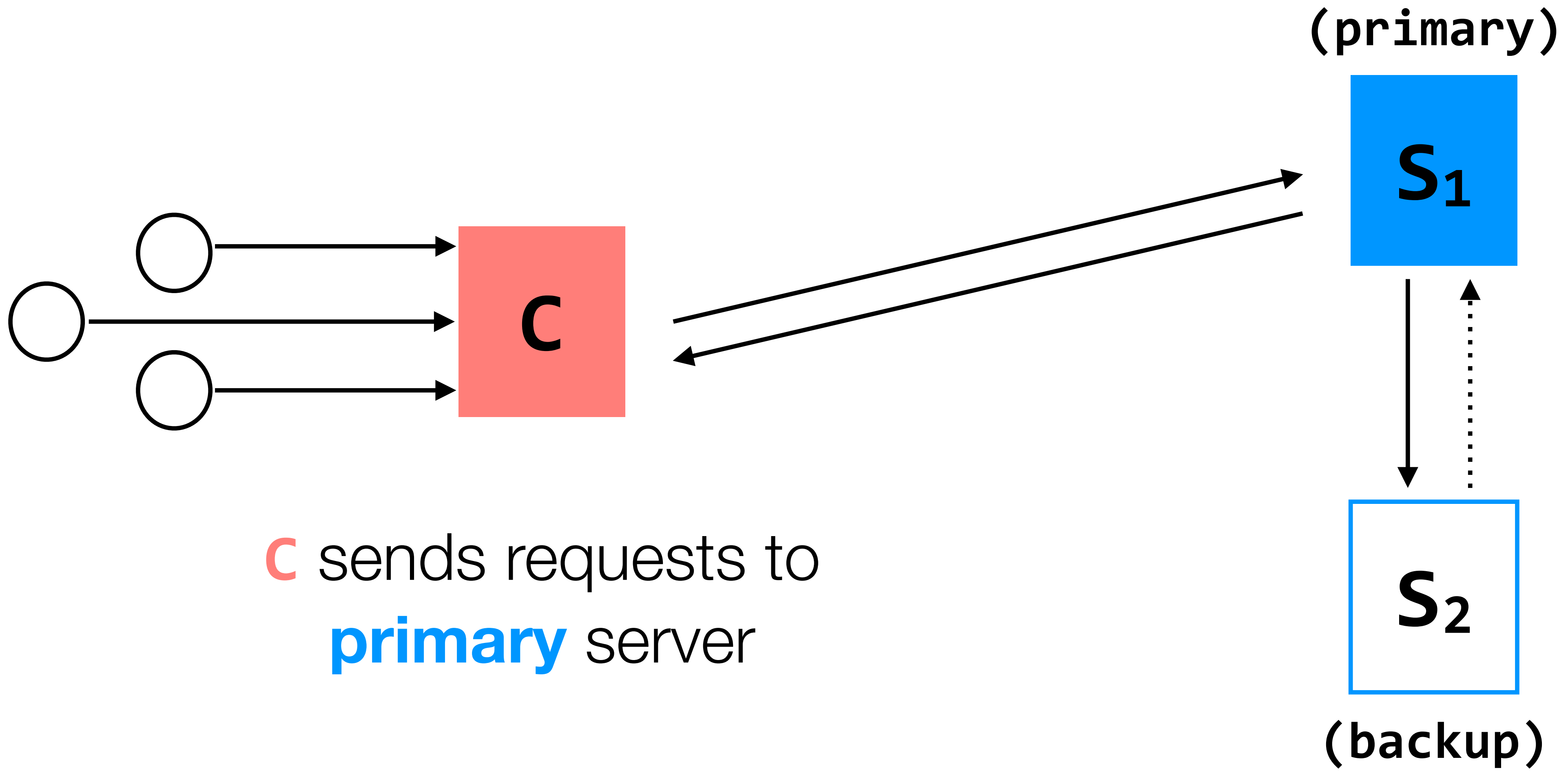
clients communicate only with C, not with replicas



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

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

clients communicate only with C, not with replicas

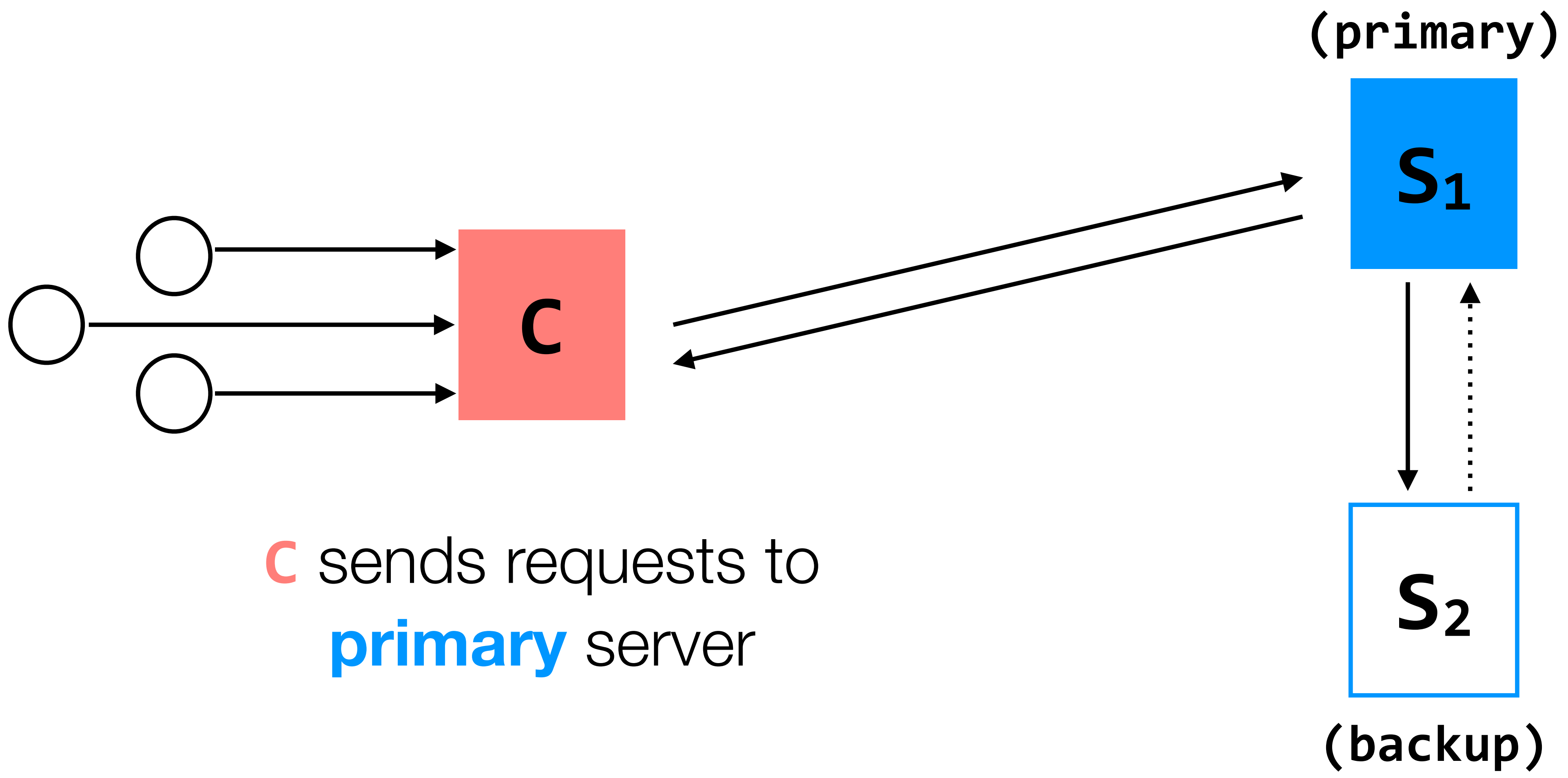


C sends requests to **primary** server

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

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

clients communicate only with C, not with replicas



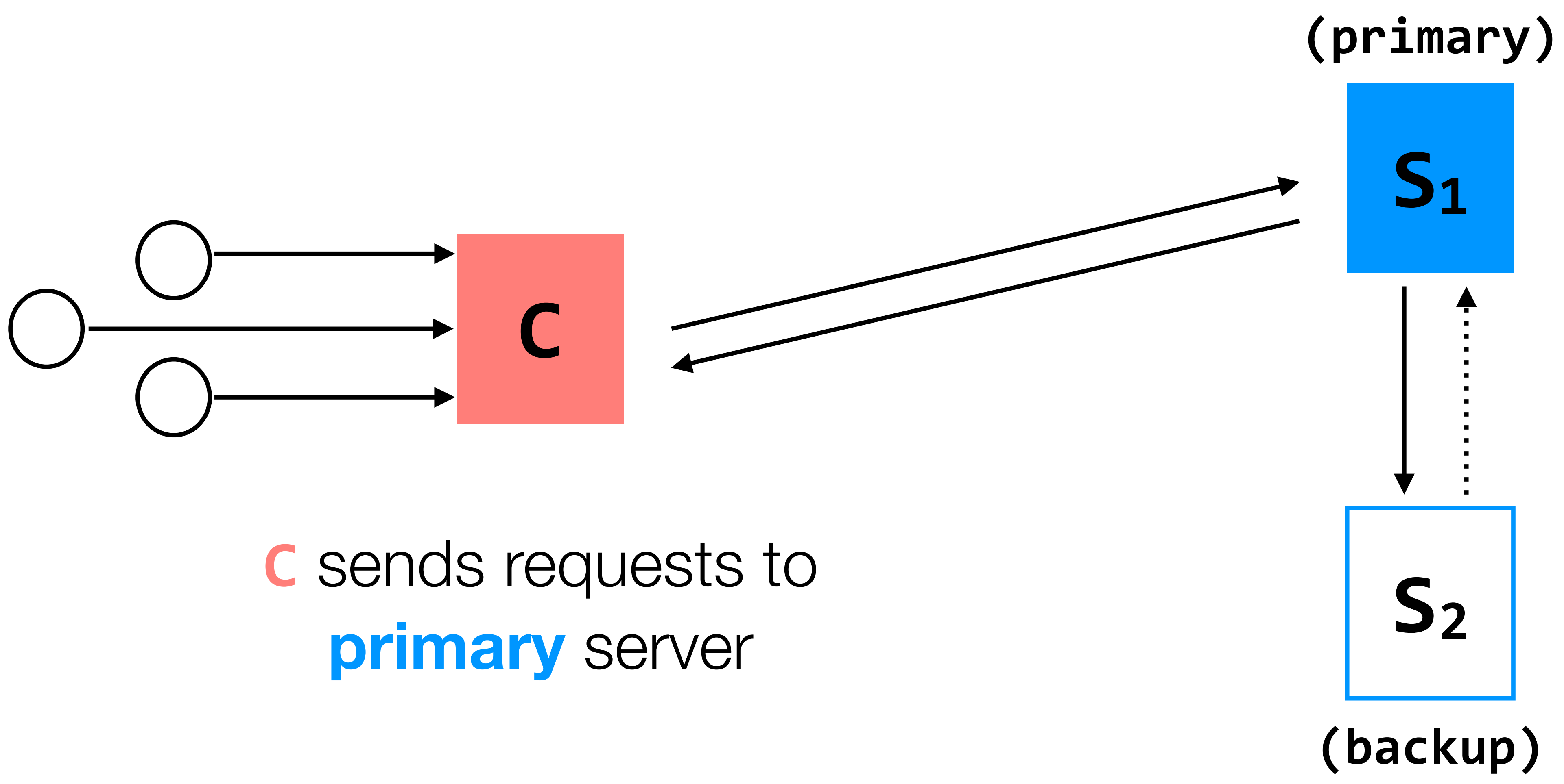
C sends requests to **primary** server

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

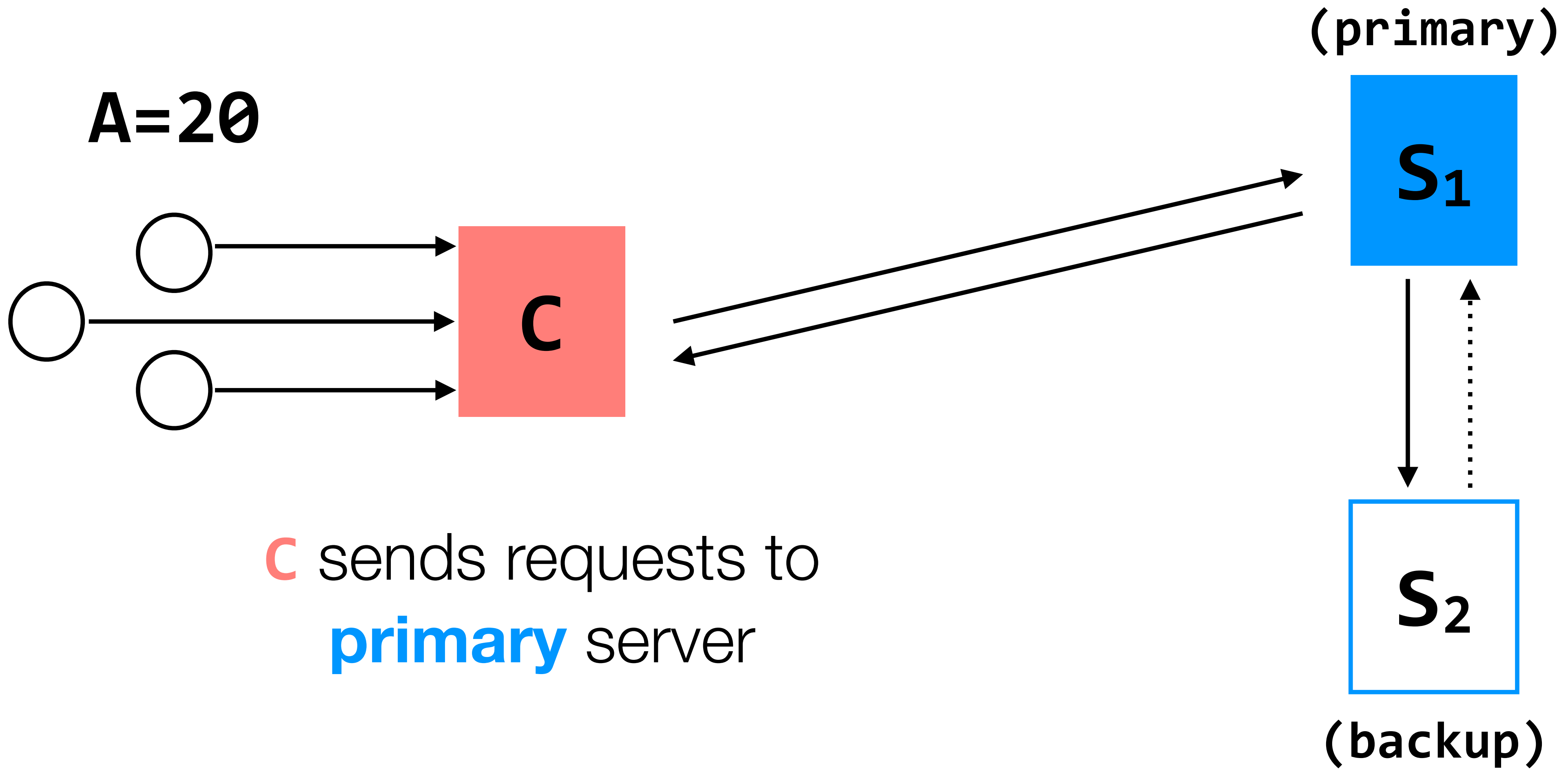
primary chooses order of operations, decides all non-deterministic values

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

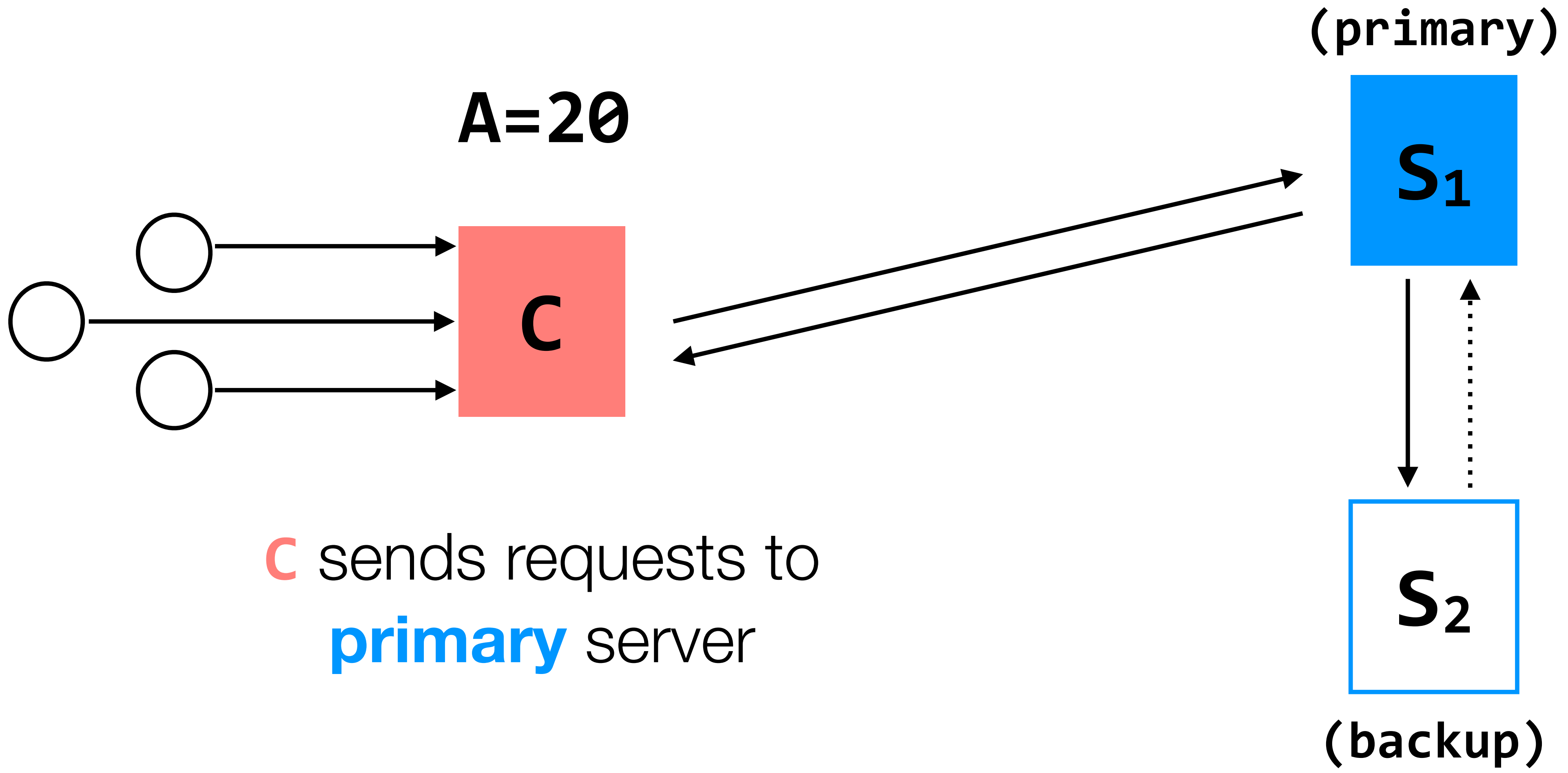
primary chooses order of operations, decides all non-deterministic values

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

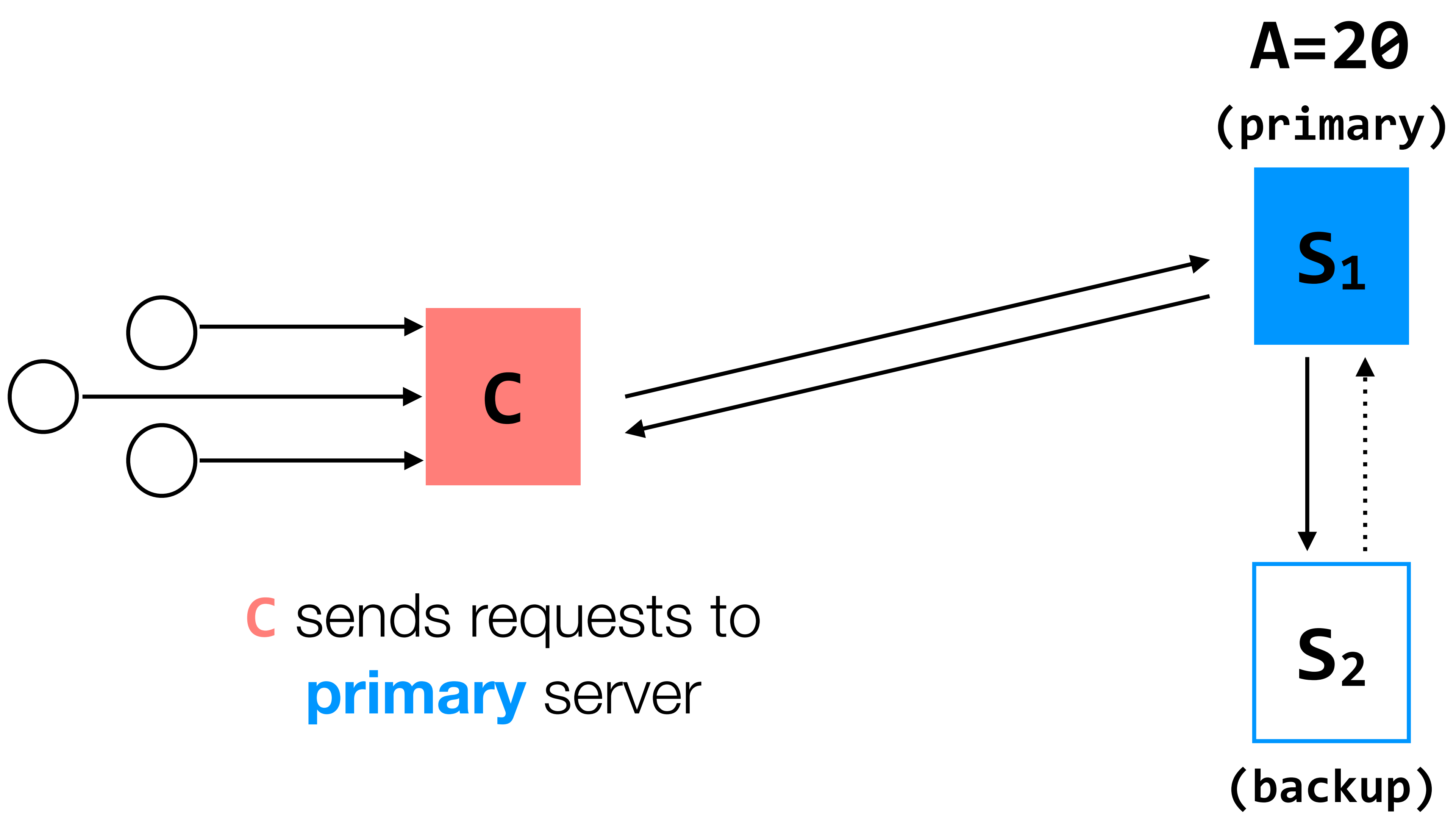
primary chooses order of operations, decides all non-deterministic values

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

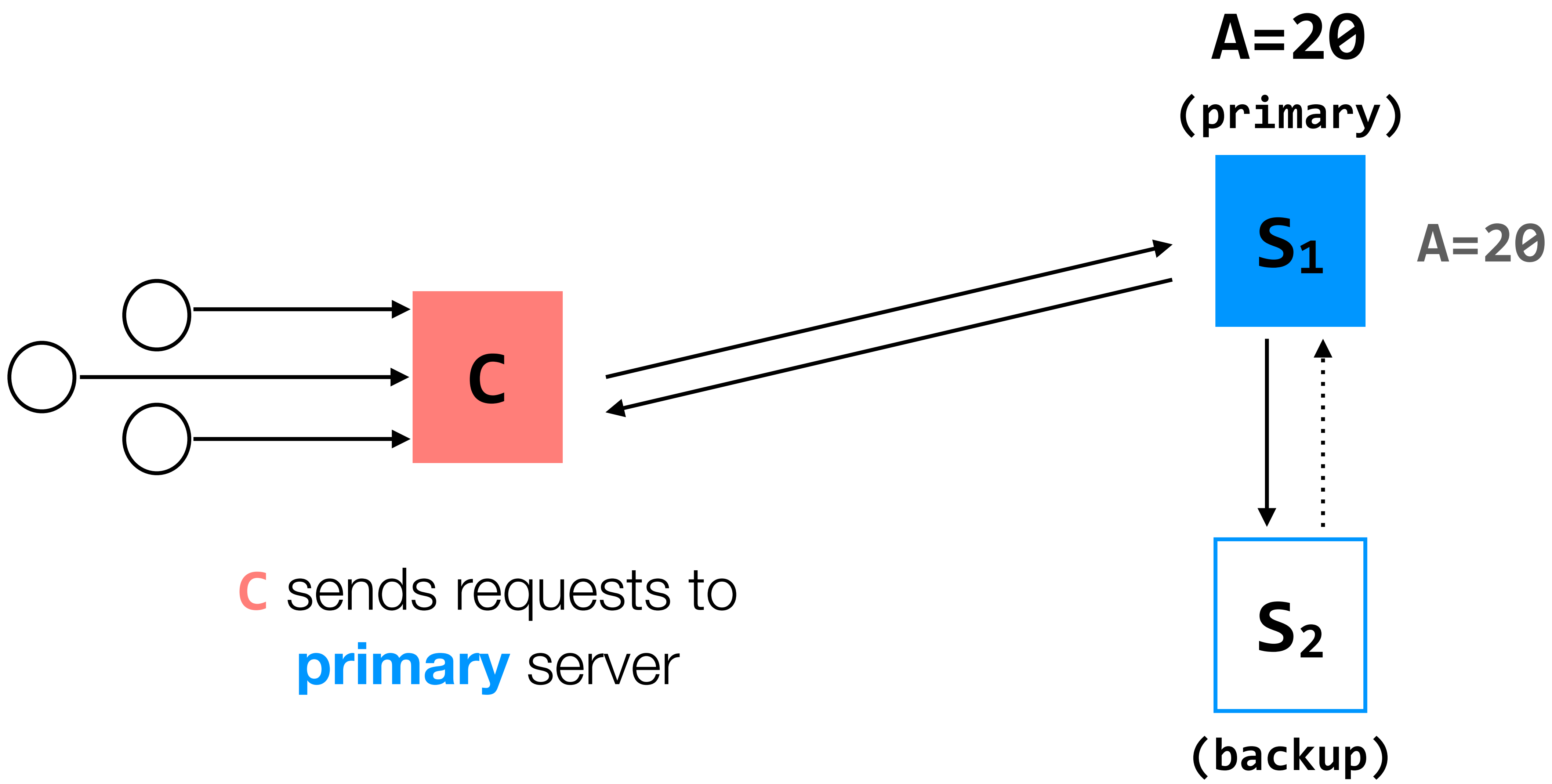
primary chooses order of operations, decides all non-deterministic values

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

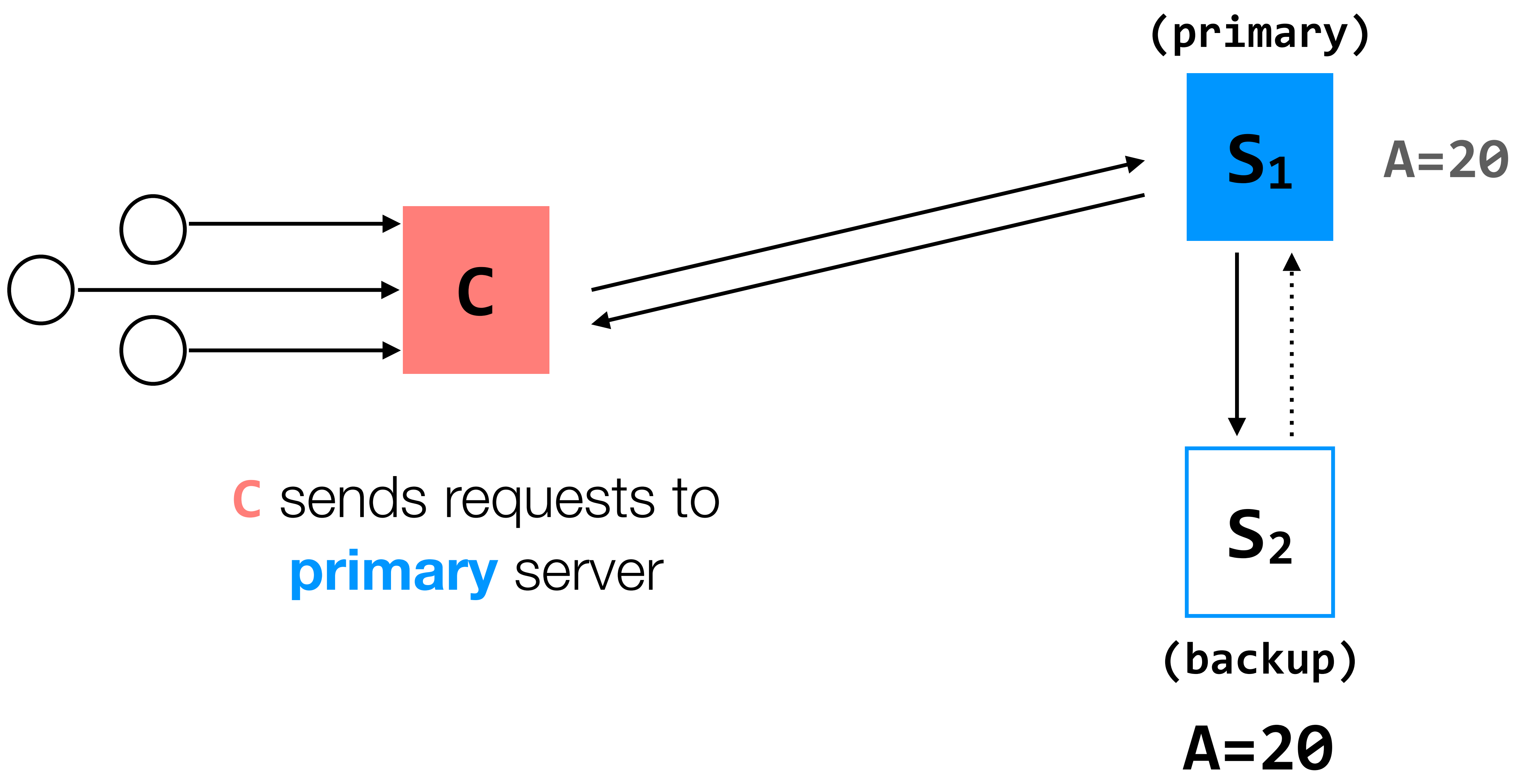
primary chooses order of operations, decides all non-deterministic values

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

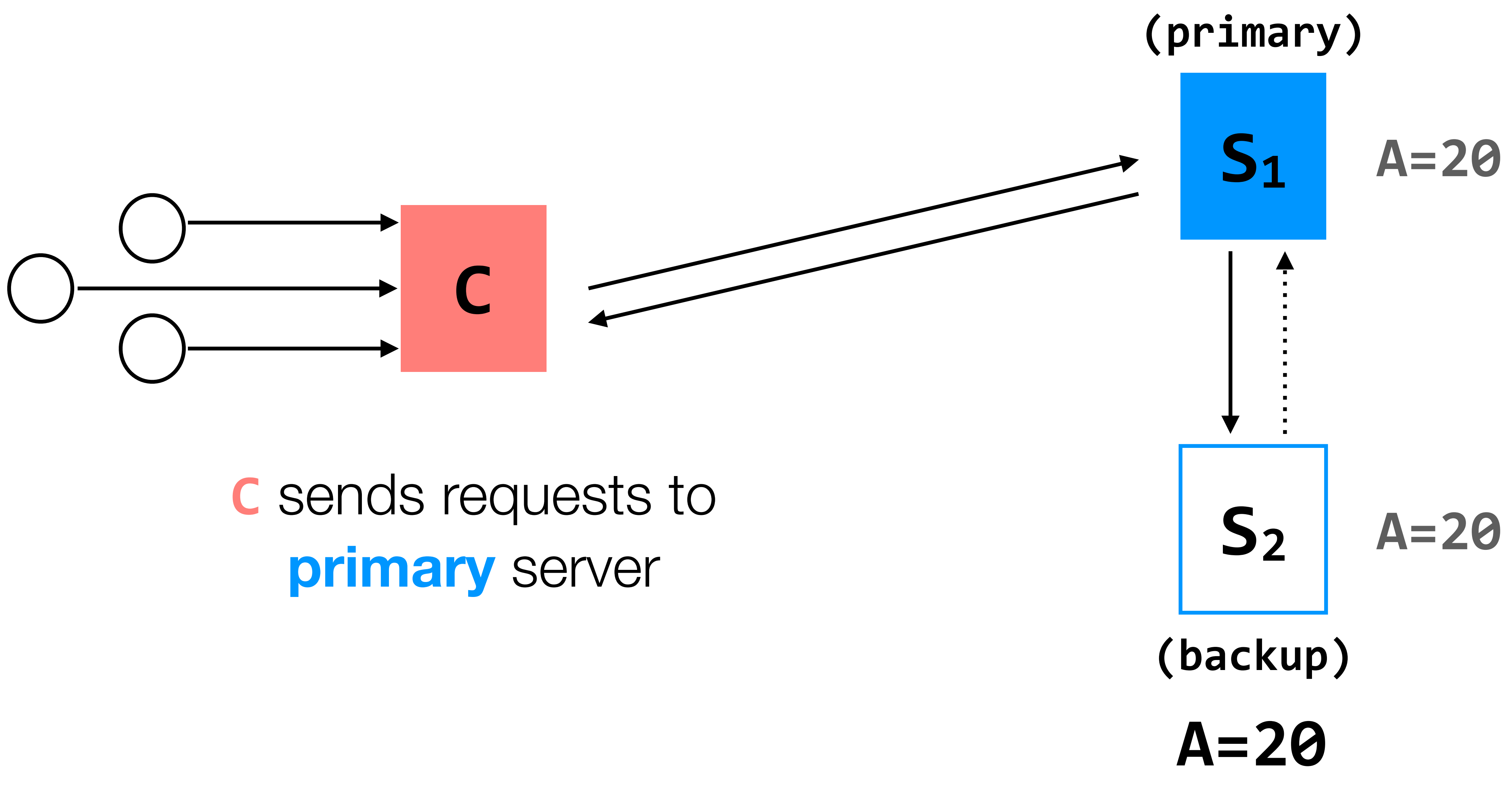
primary chooses order of operations, decides all non-deterministic values

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

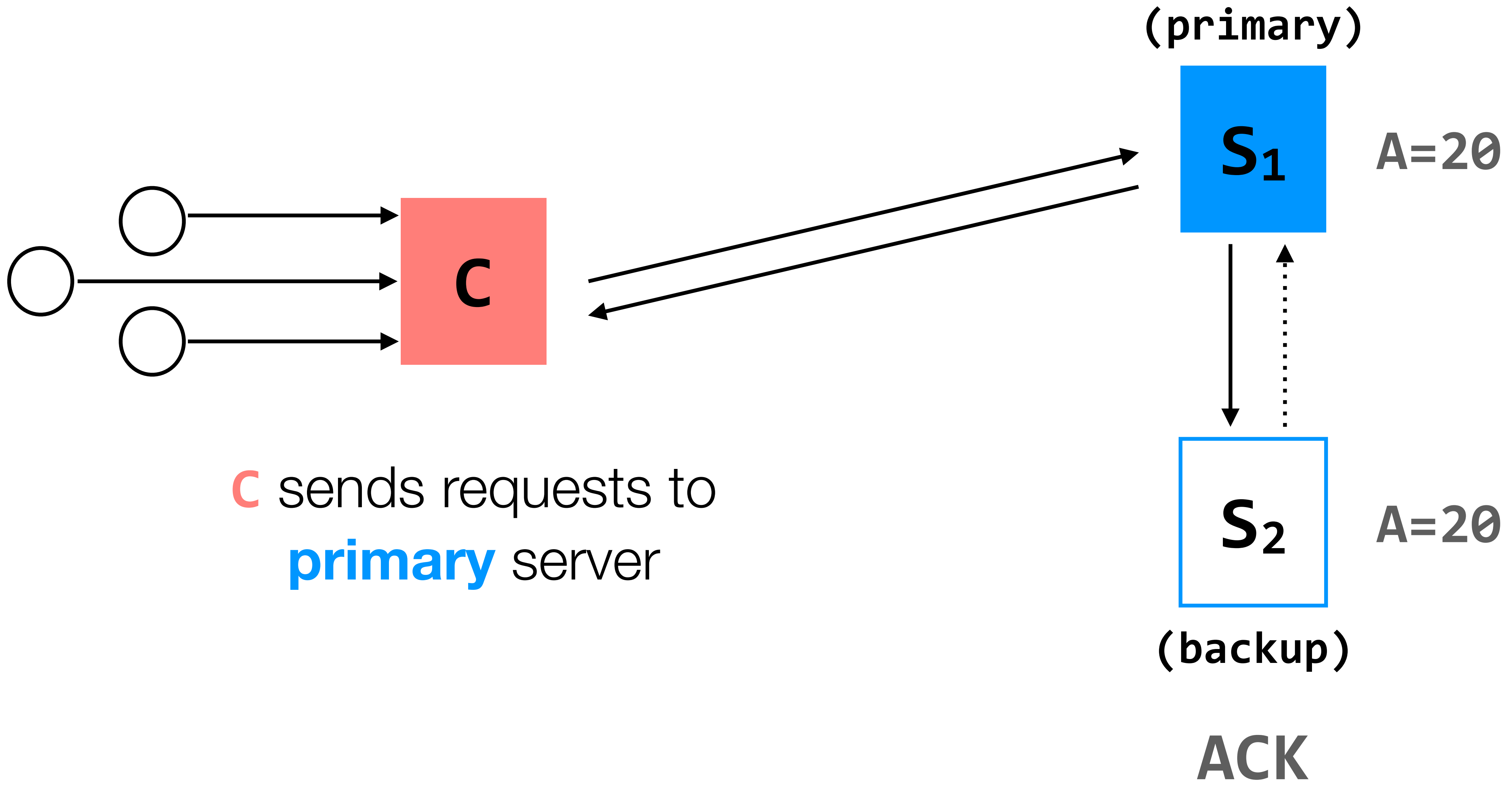
primary chooses order of operations, decides all non-deterministic values

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

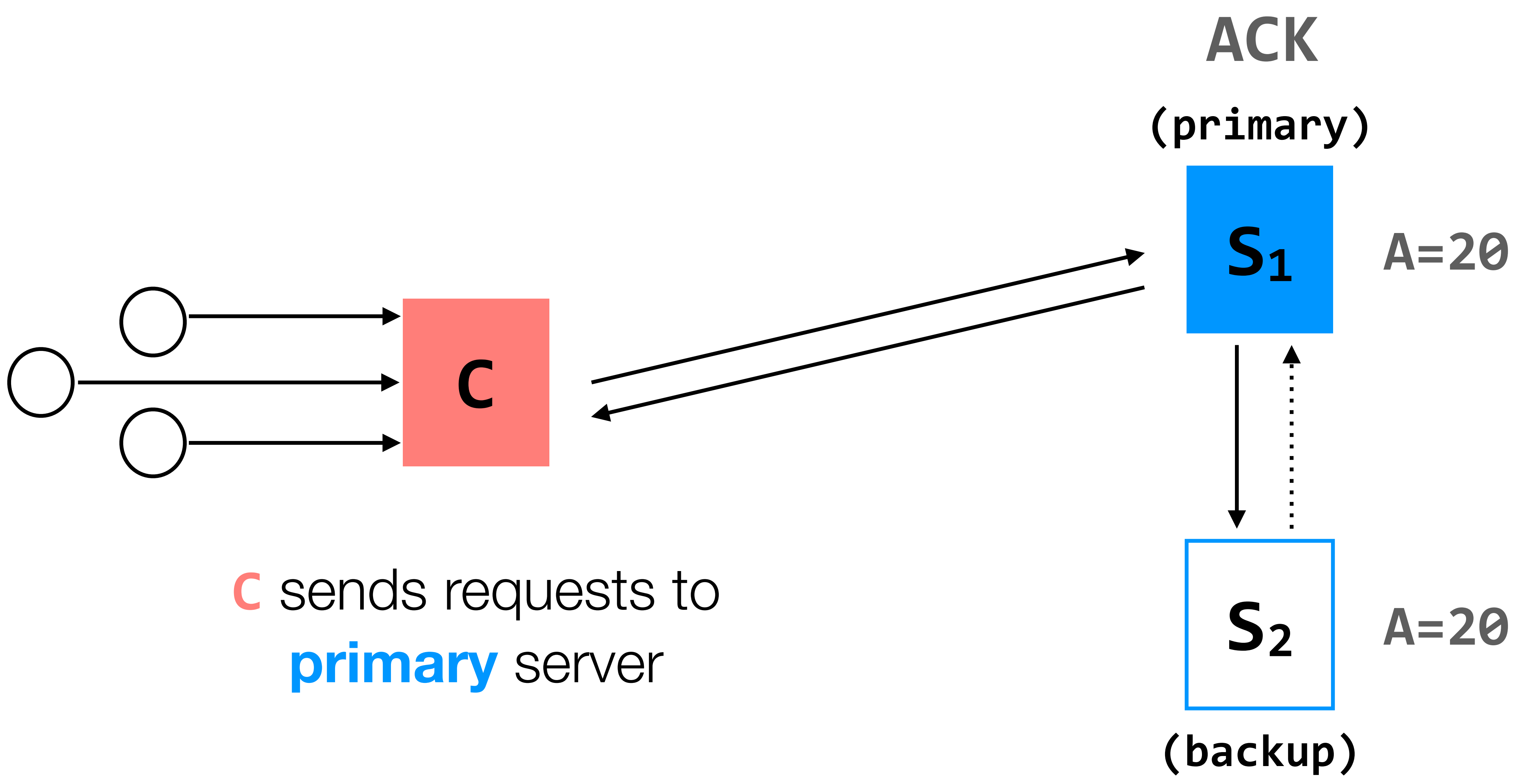
primary chooses order of operations, decides all non-deterministic values

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

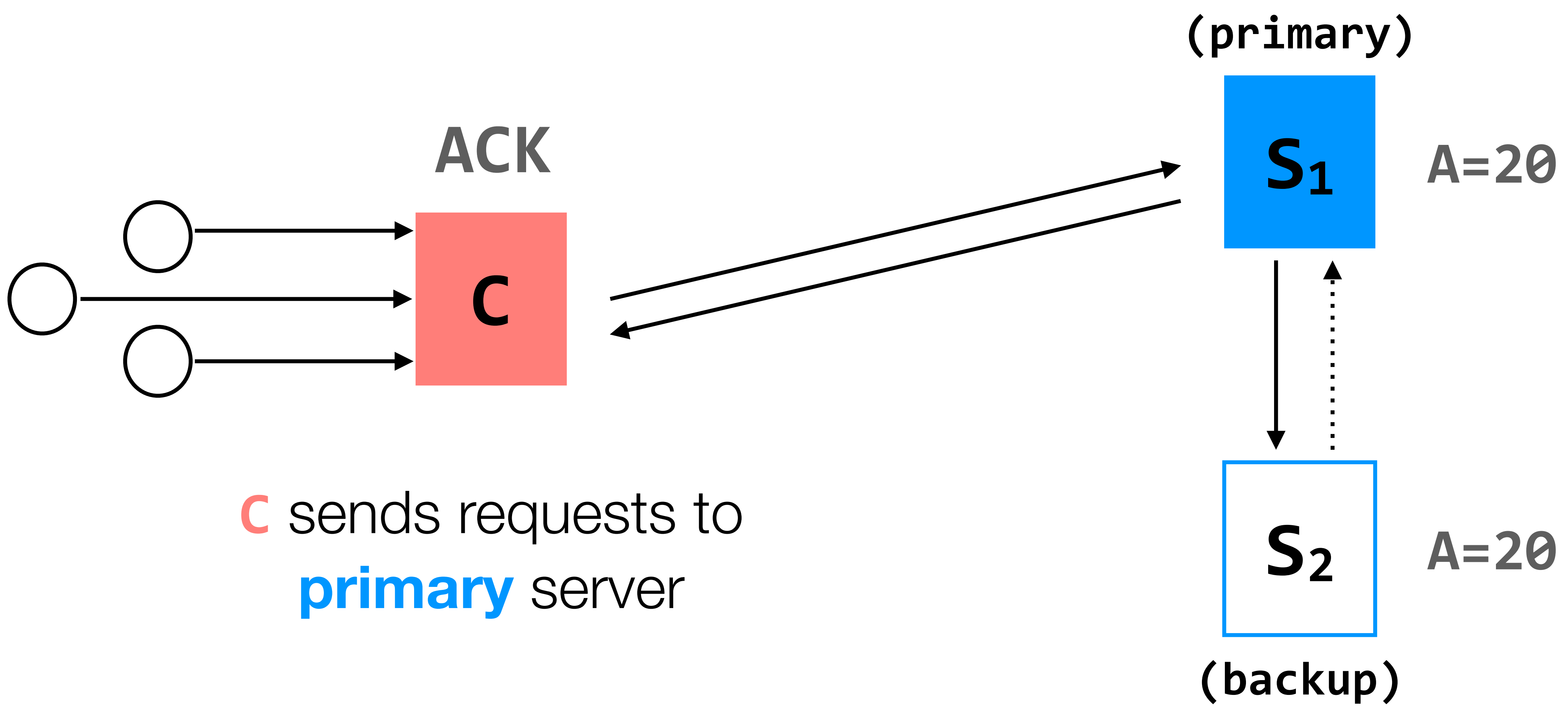
primary chooses order of operations, decides all non-deterministic values

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

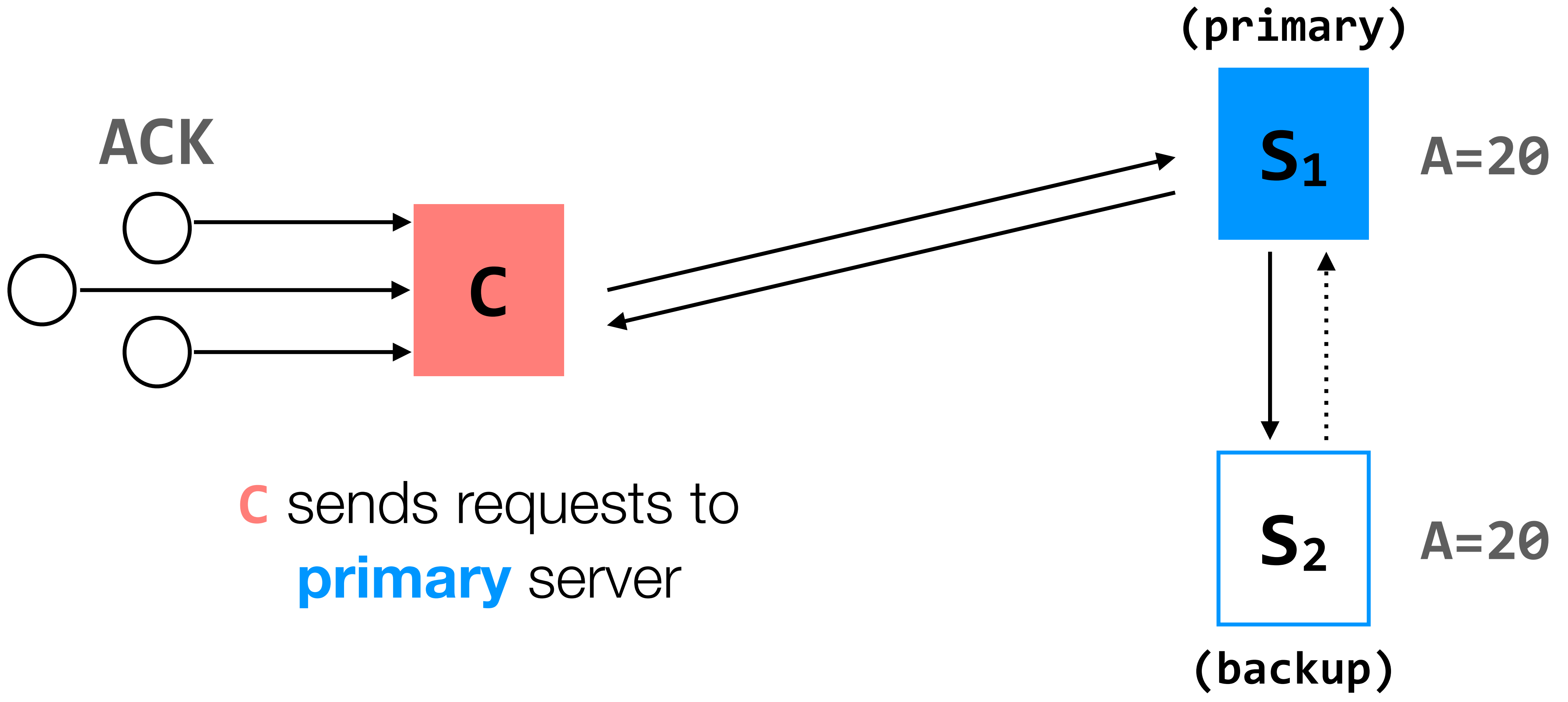
primary chooses order of operations, decides all non-deterministic values

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

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

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

clients communicate only with C, not with replicas



C sends requests to **primary** server

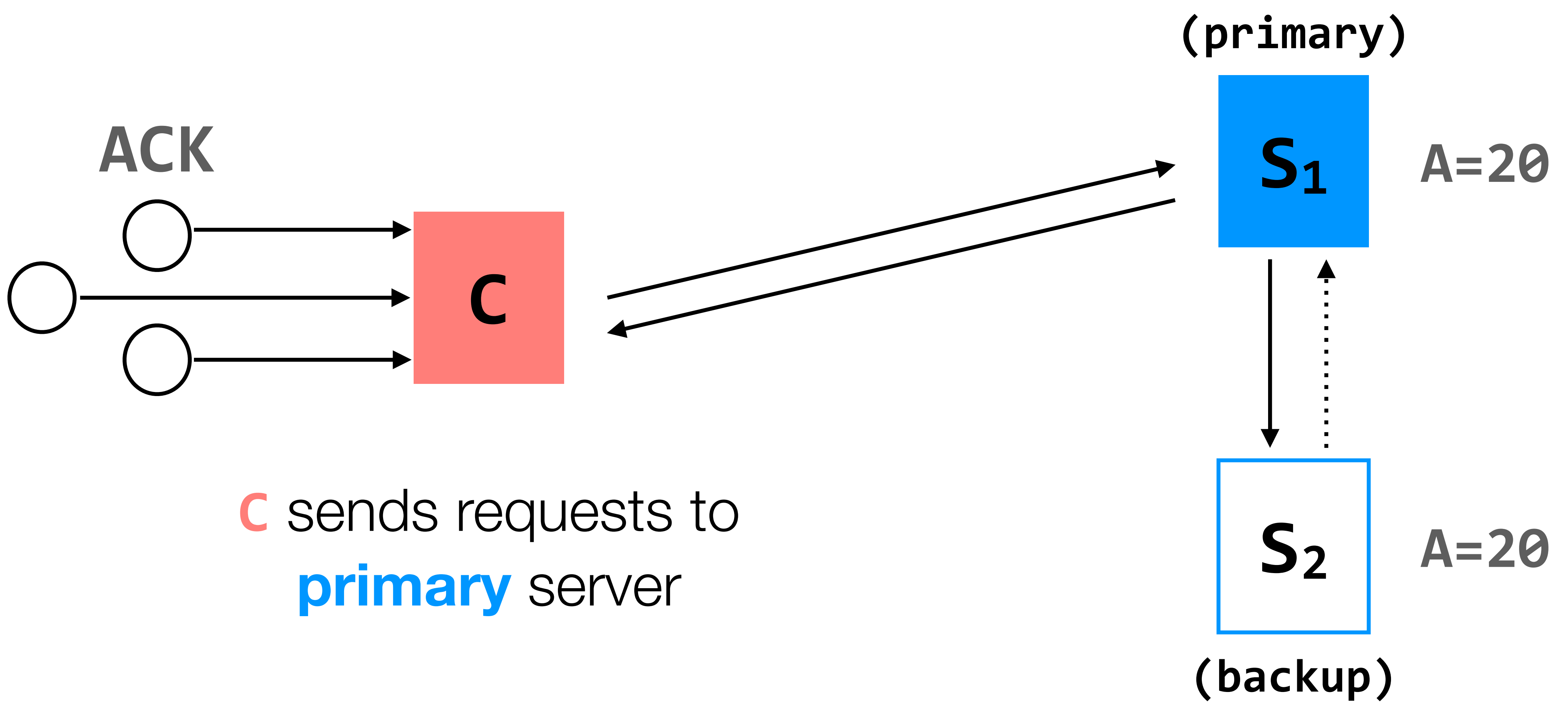
primary chooses order of operations, decides all non-deterministic values

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

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

attempt 2: make one replica the **primary** replica, and have **coordinators** 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

all coordinators send requests to the **primary** server, which avoids the problem we saw in our first attempt

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

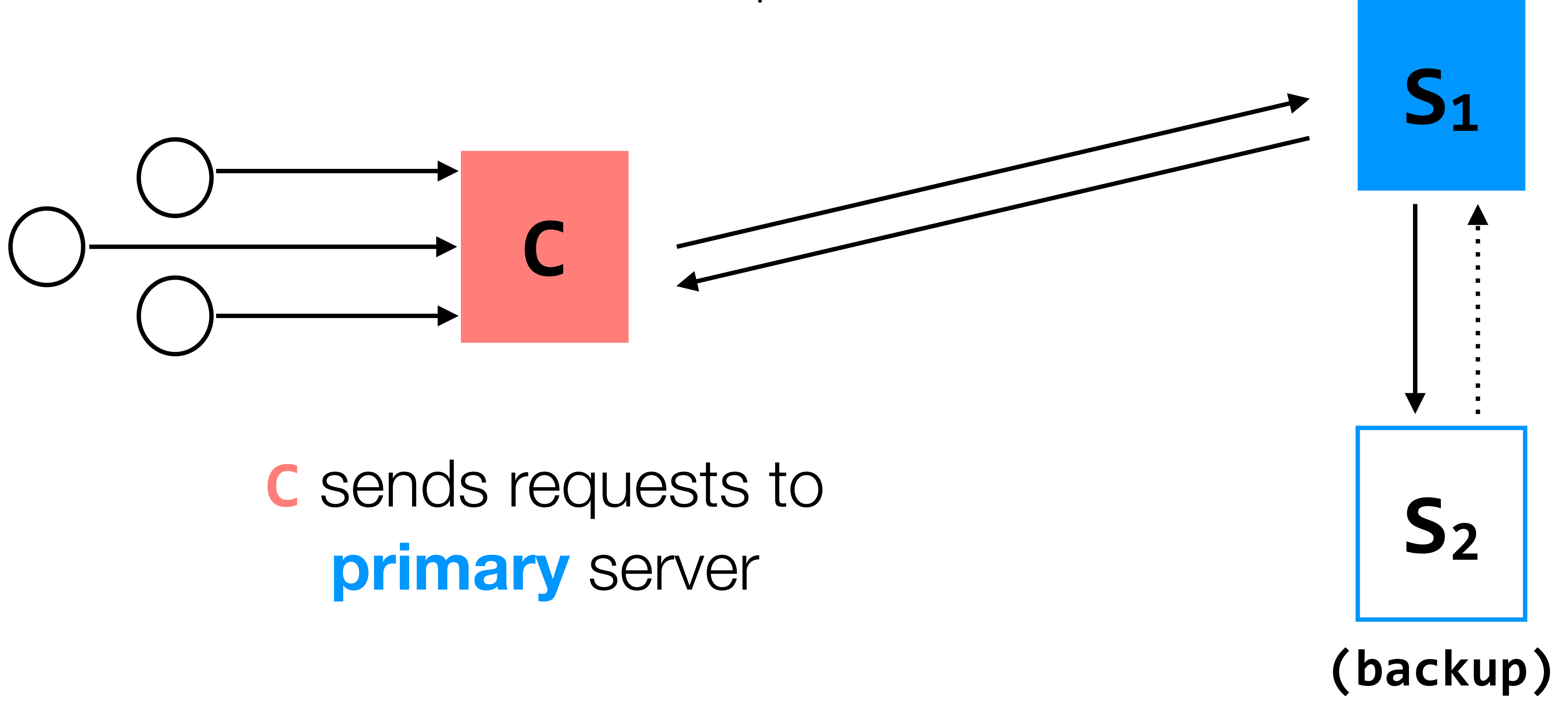
attempt 2: make one replica the **primary** replica, and have **coordinators** in place to help manage failures

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

C knows how to contact backup server

clients communicate only with C, not with replicas

(primary)



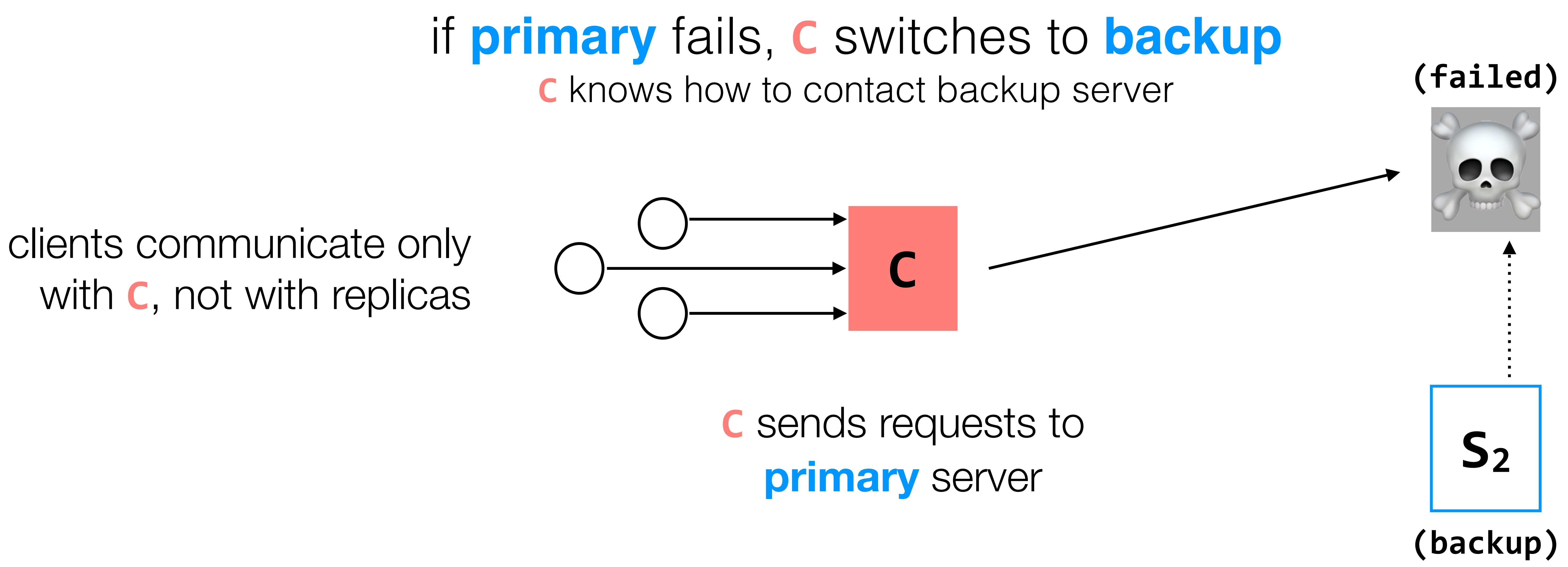
C sends requests to **primary** server

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

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

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

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



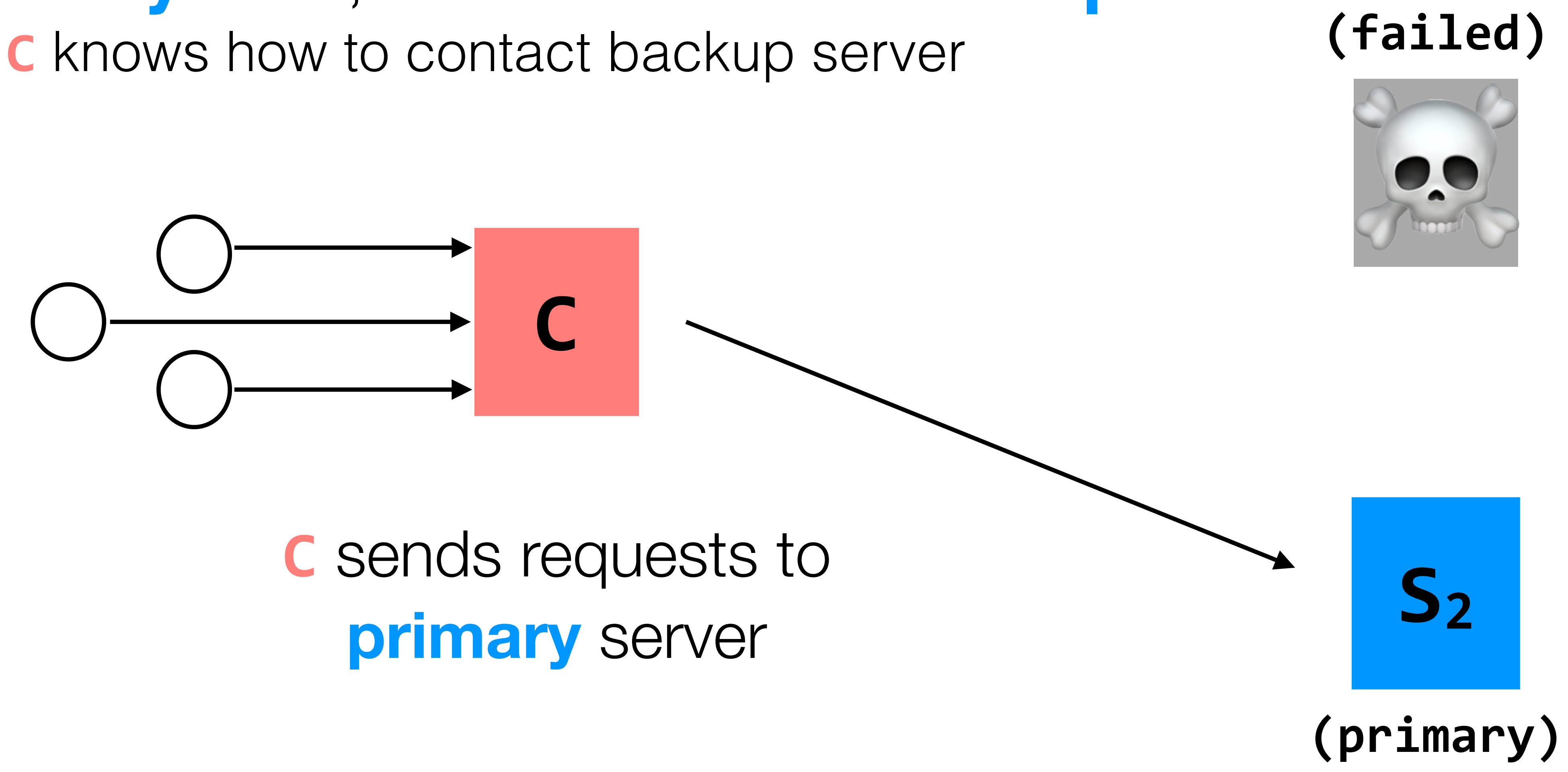
to increase availability, let's try replicating data on two servers

attempt 2: make one replica the **primary** replica, and have **coordinators** 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



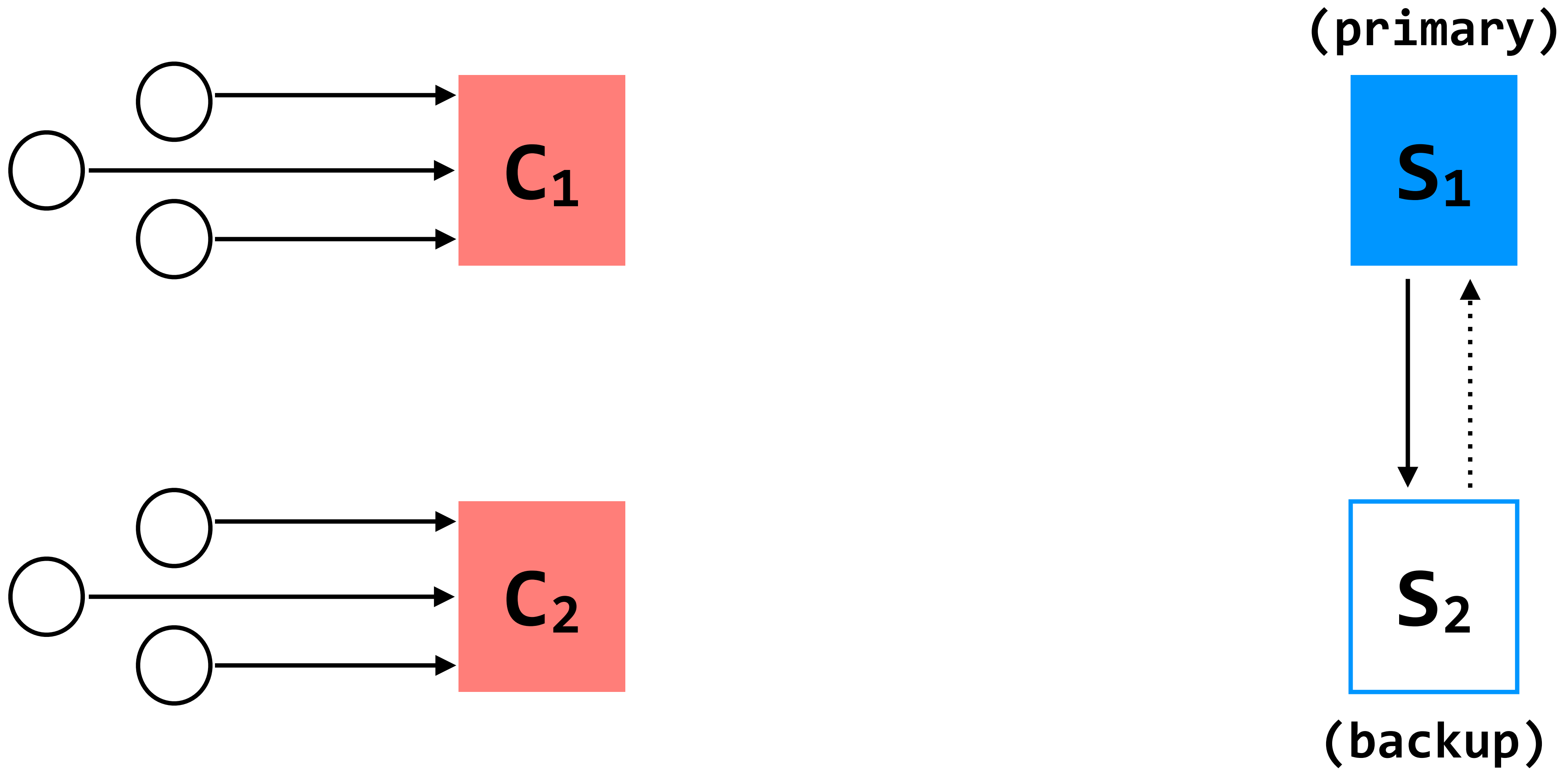
C sends requests to **primary** server

ideally, S_1 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 **coordinators** in place to help manage failures

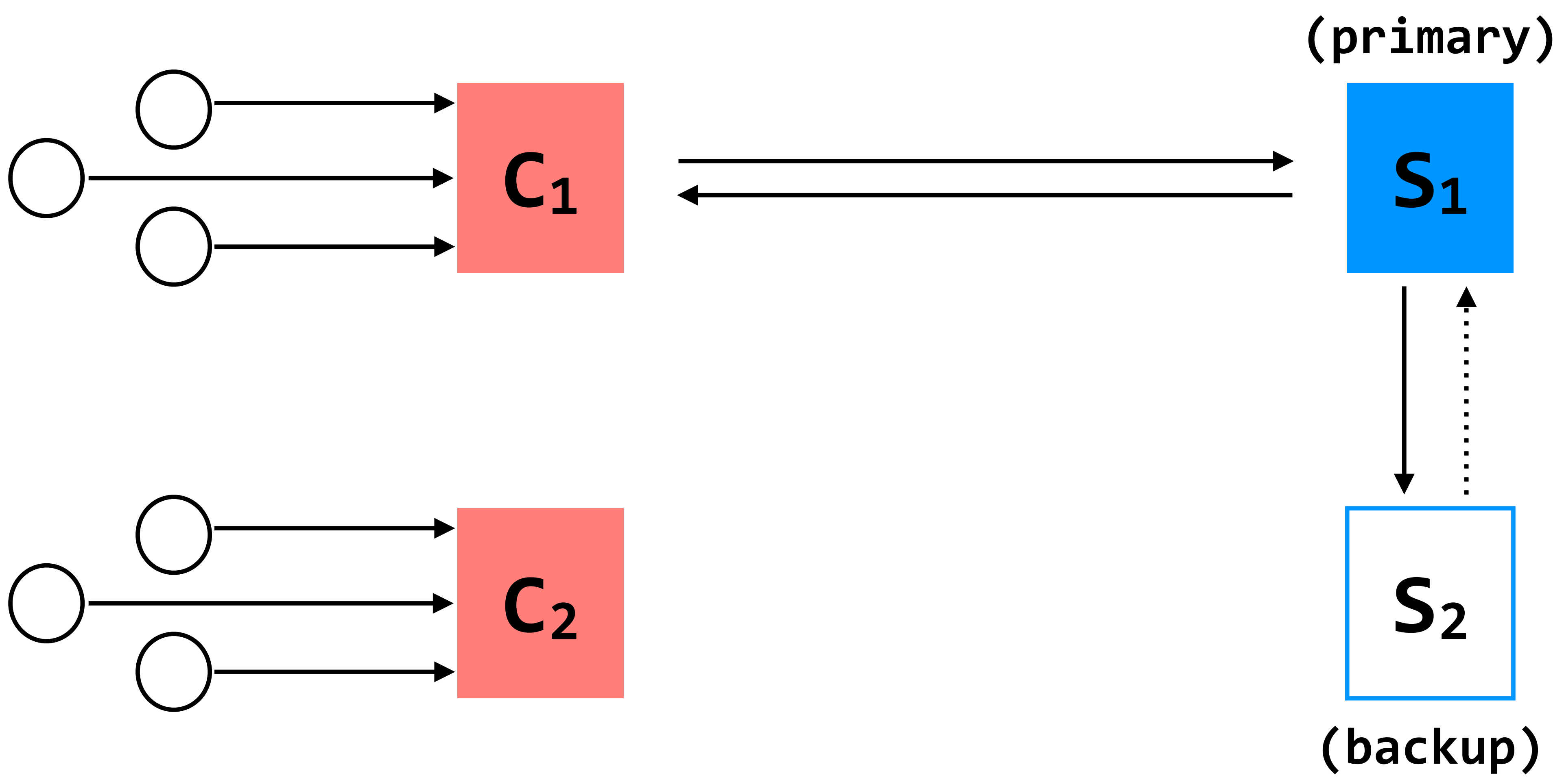
for a single transaction, a client would communicate with a single `coordinator`



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

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

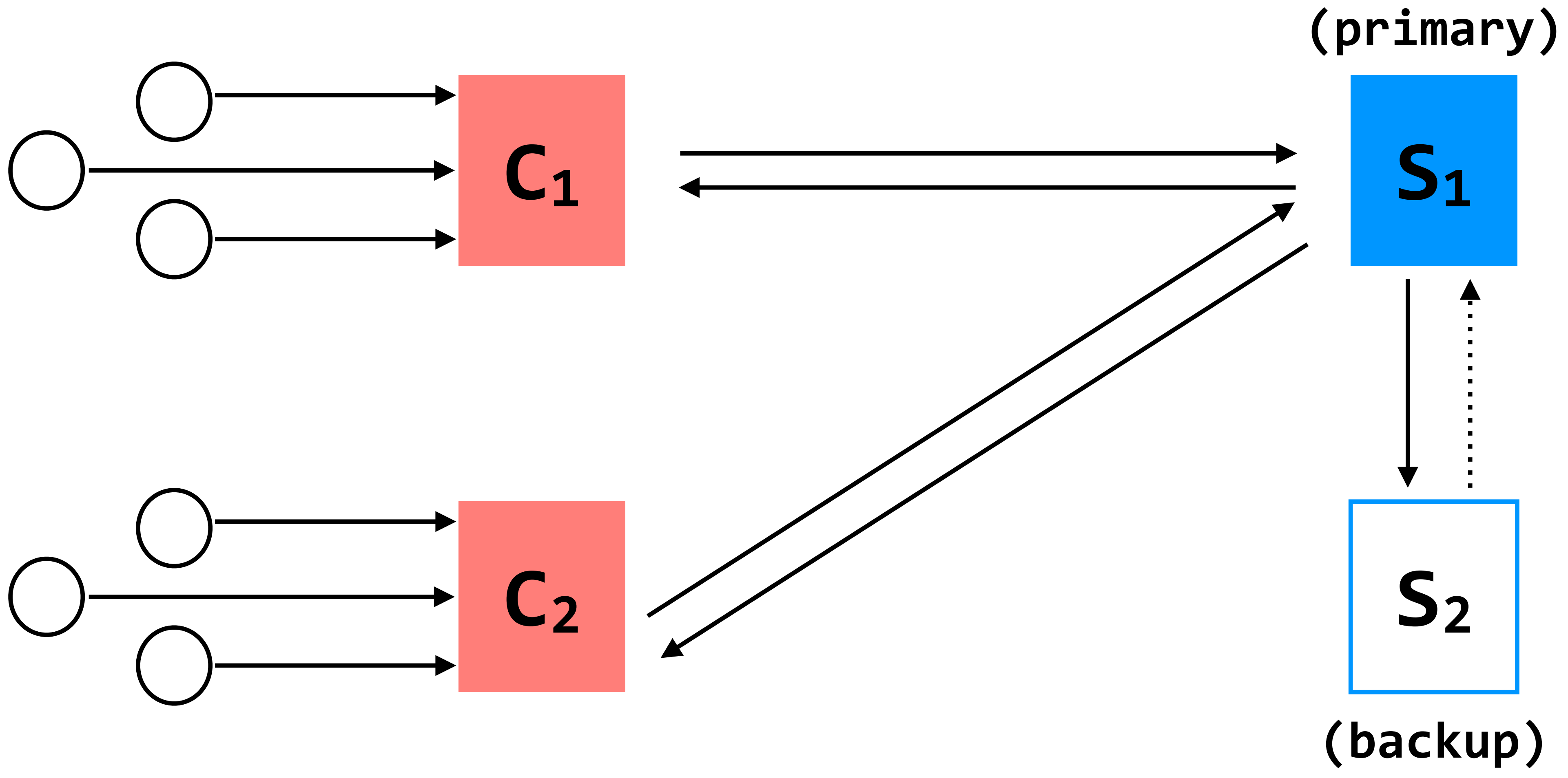
for a single transaction, a client would communicate with a single `coordinator`



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

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

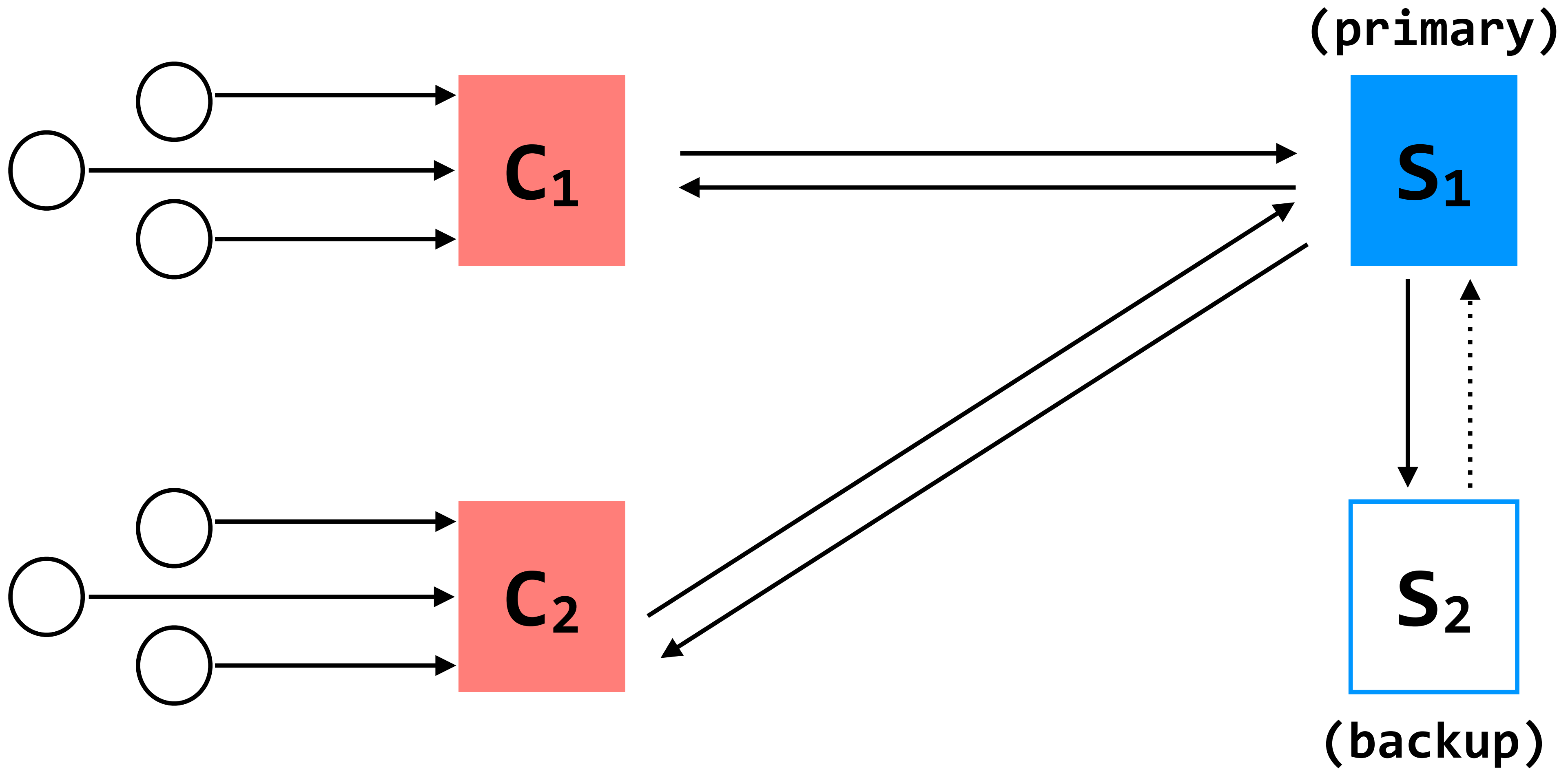
for a single transaction, a client would communicate with a single `coordinator`



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

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

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

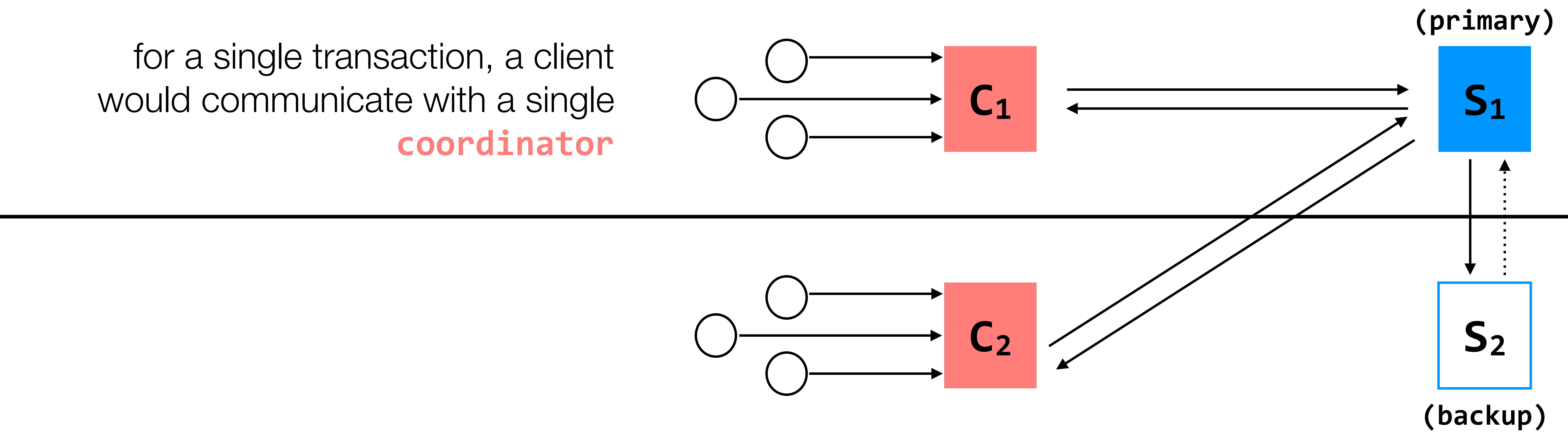


suppose that all machines remain up, but that there is a **network partition** that effectively splits this network in half

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

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

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



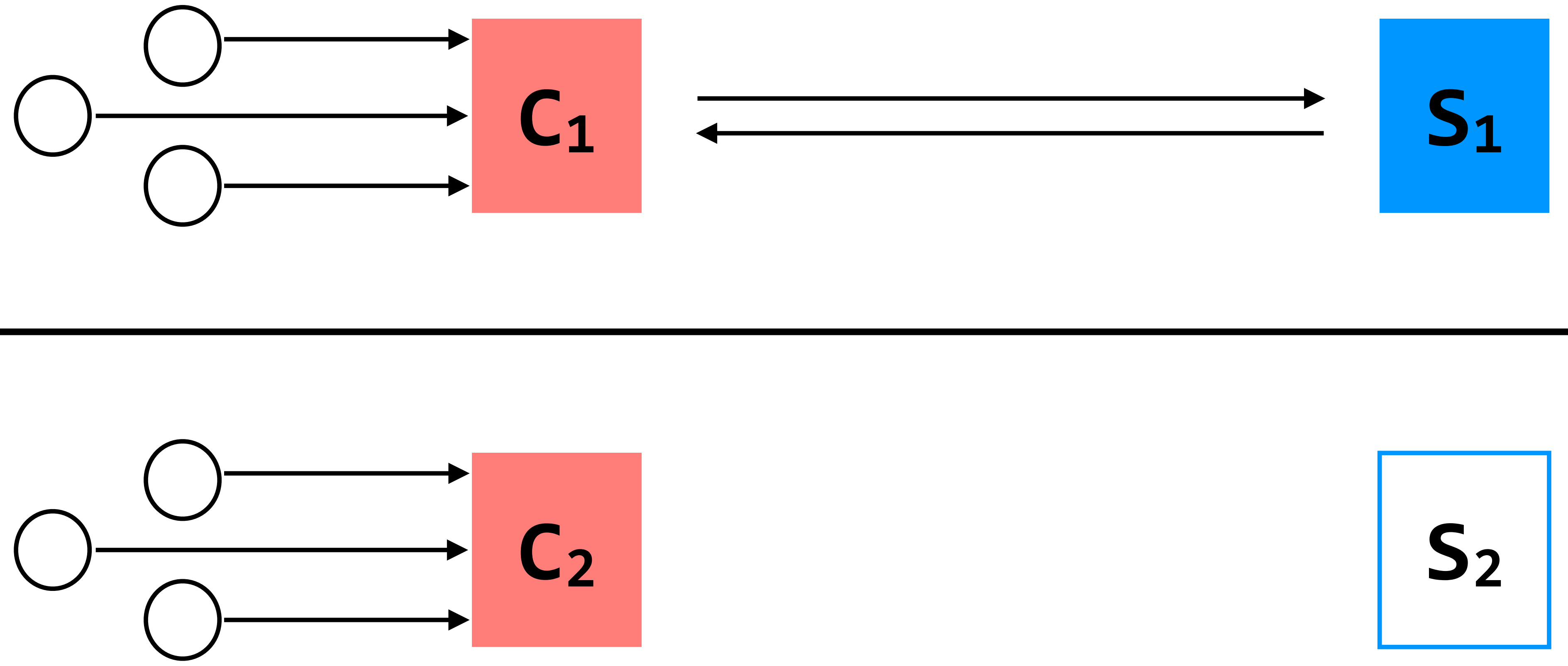
a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side

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

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

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

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

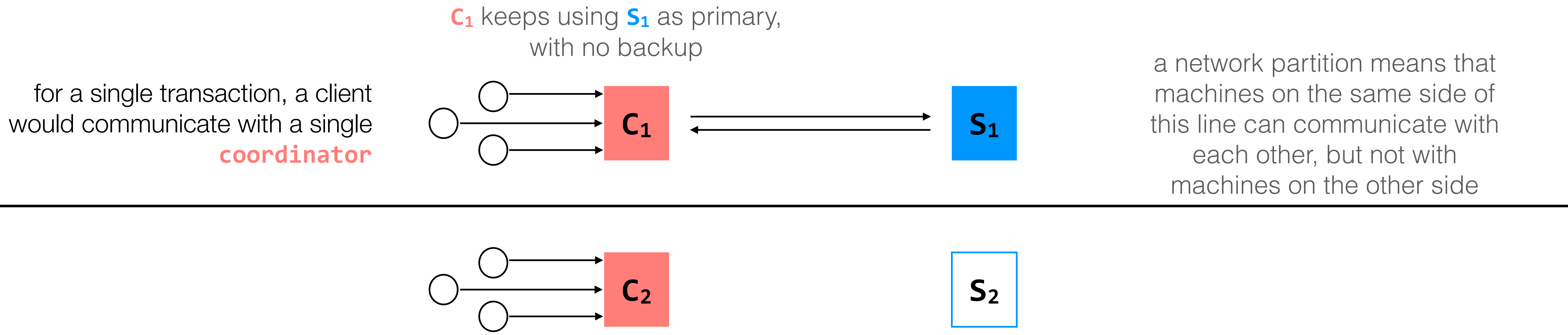


a network partition means that machines on the same side of this line can communicate with each other, but not with machines on the other side

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

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

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



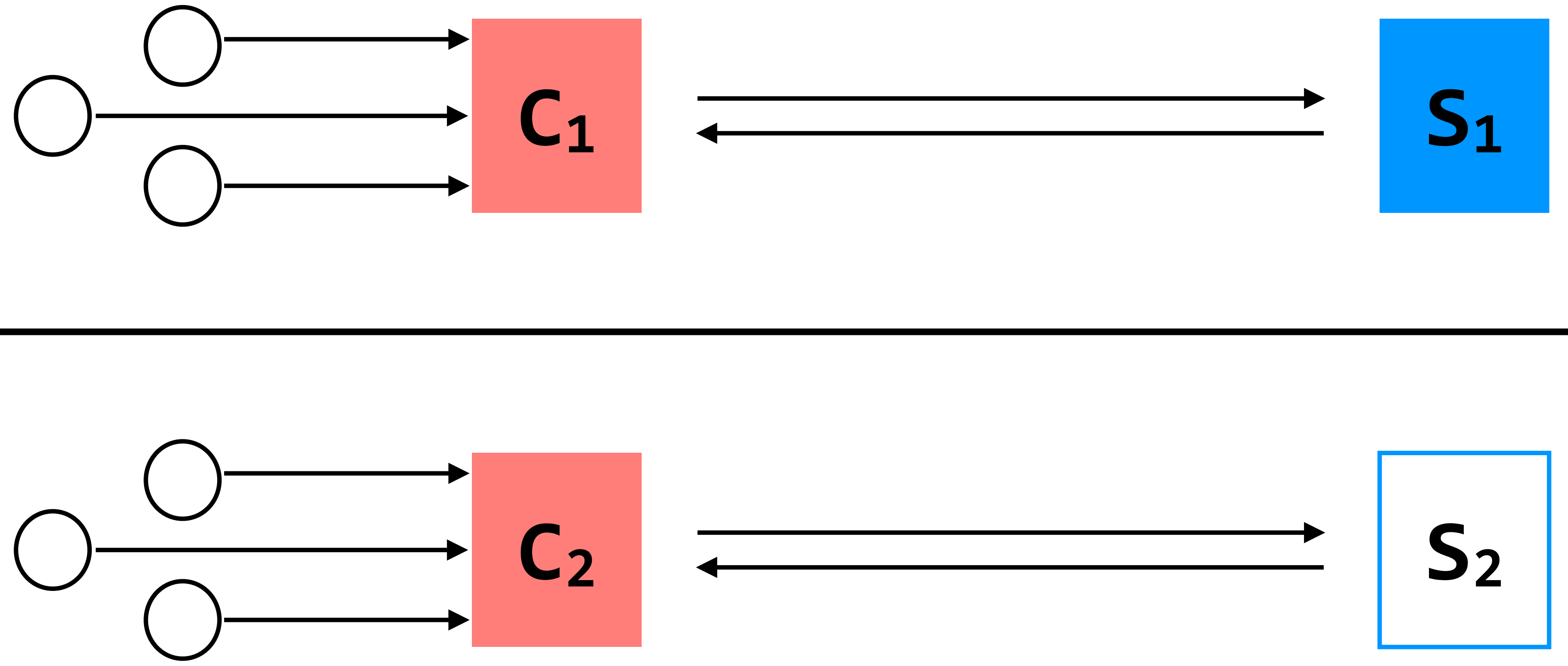
suppose that all machines remain up, but that there is a **network partition** that effectively splits this network in half

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

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

C_1 keeps using S_1 as primary, with no backup

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

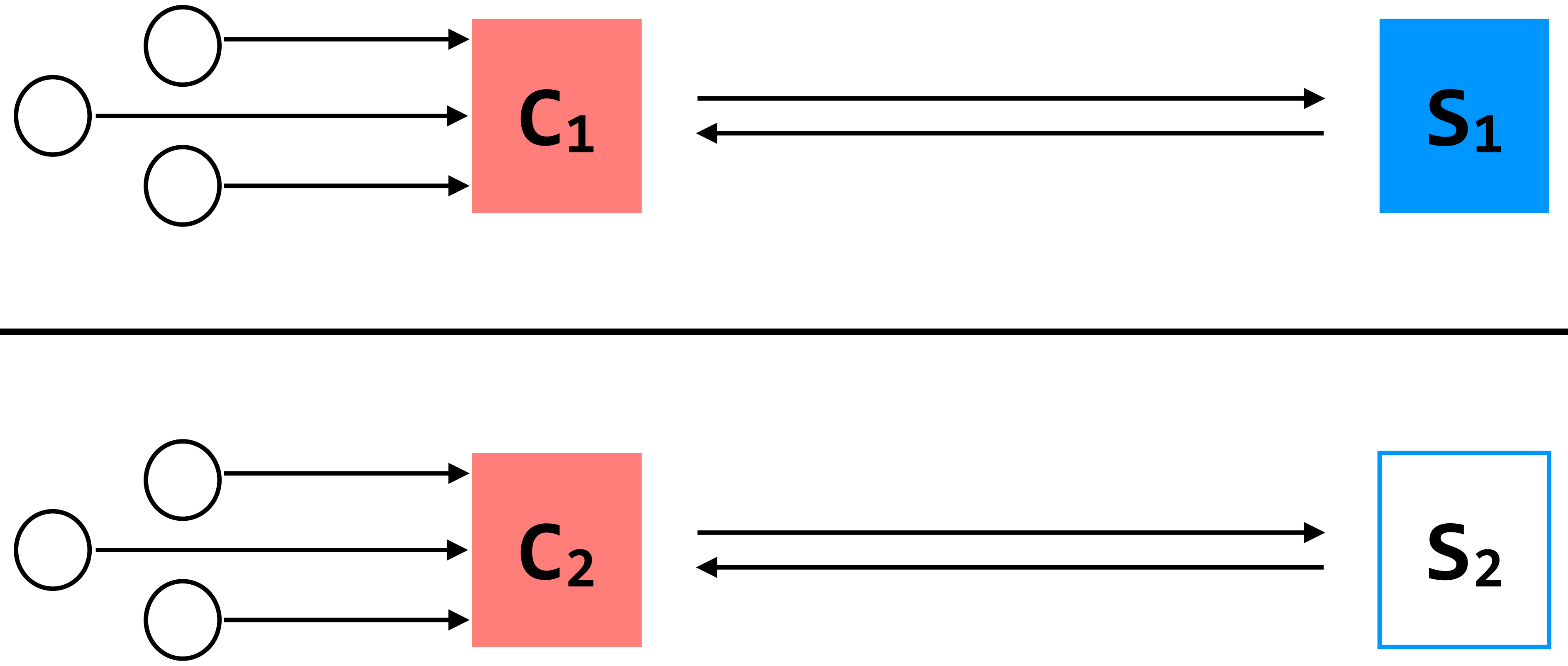
suppose that all machines remain up, but that there is a **network partition** that effectively splits this network in half

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

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

C_1 keeps using S_1 as primary, with no backup

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

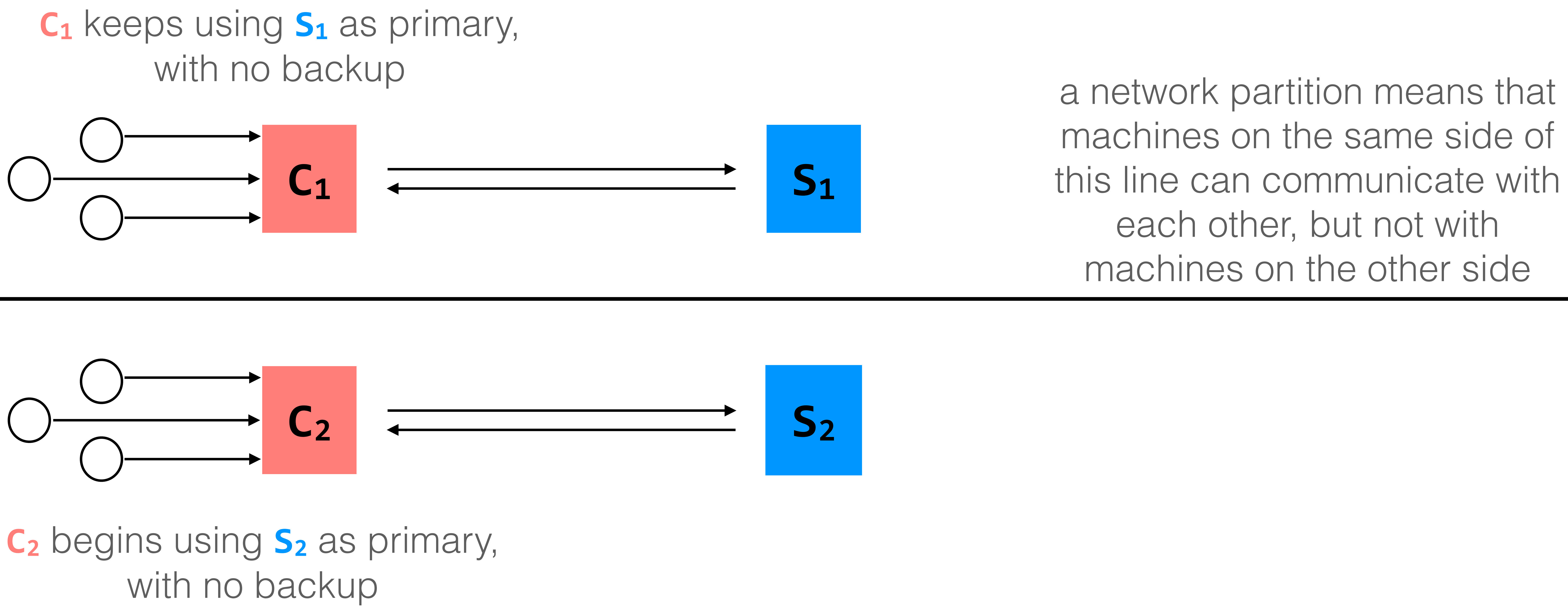
C_2 begins using S_2 as primary, with no backup

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 **coordinators** in place to help manage failures

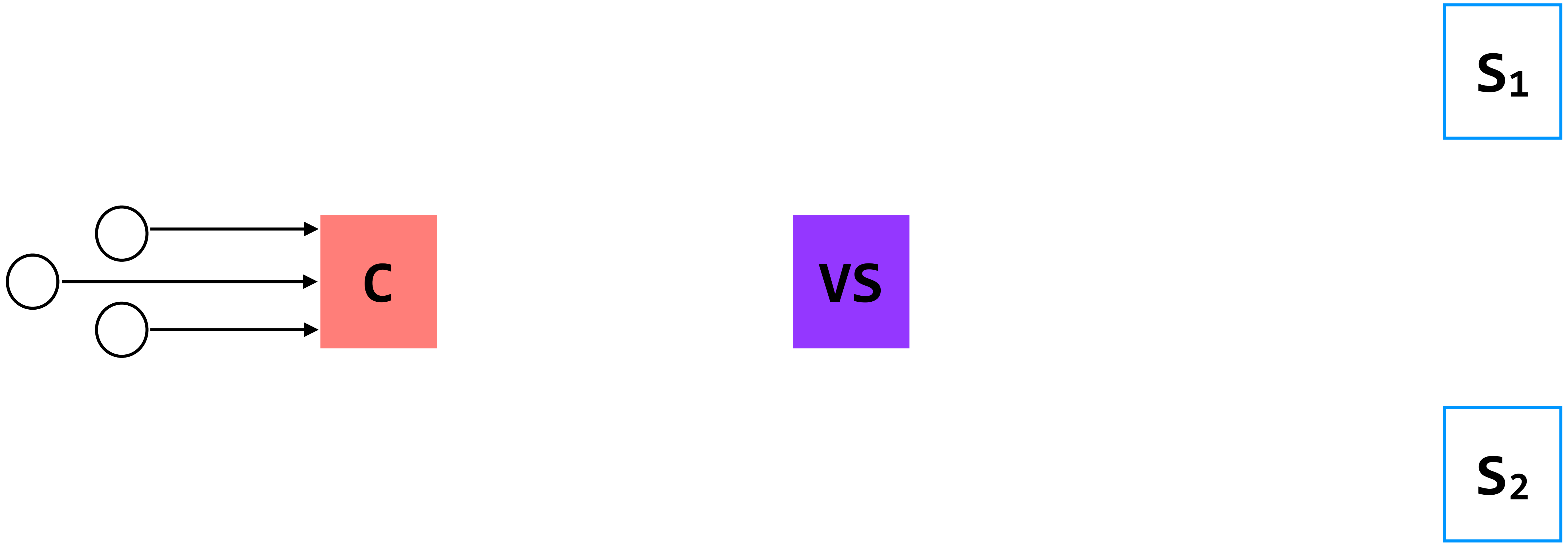
for a single transaction, a client would communicate with a single `coordinator`



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

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



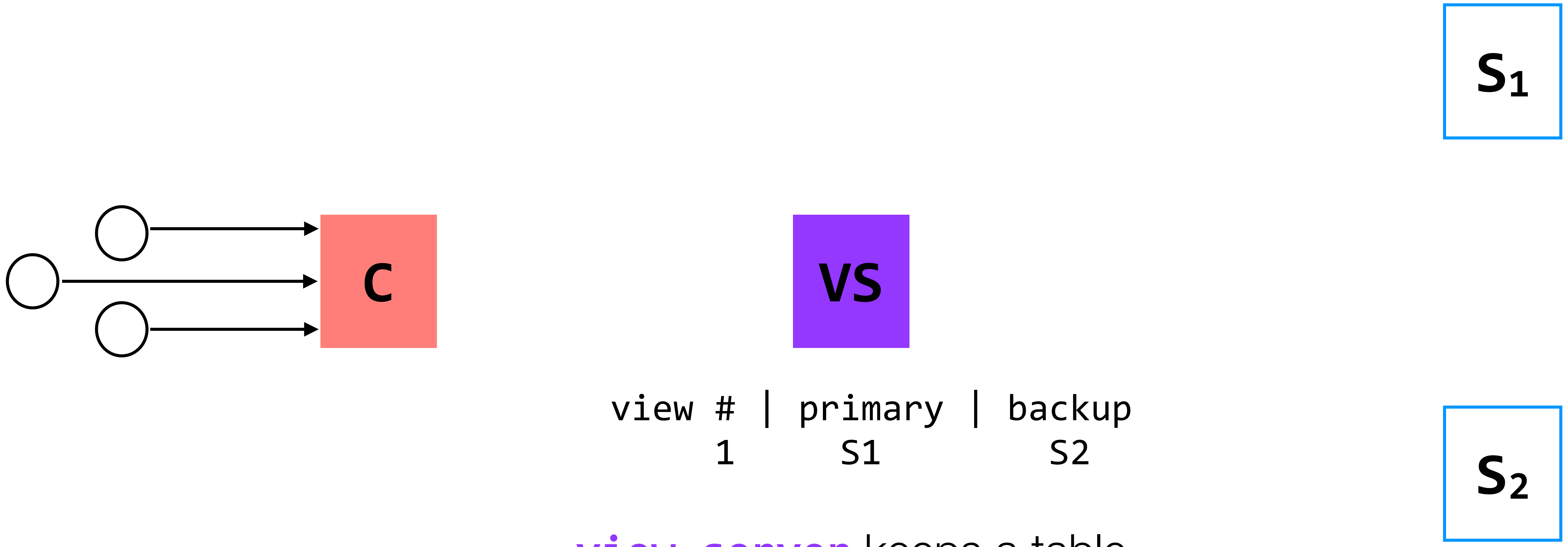
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

`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

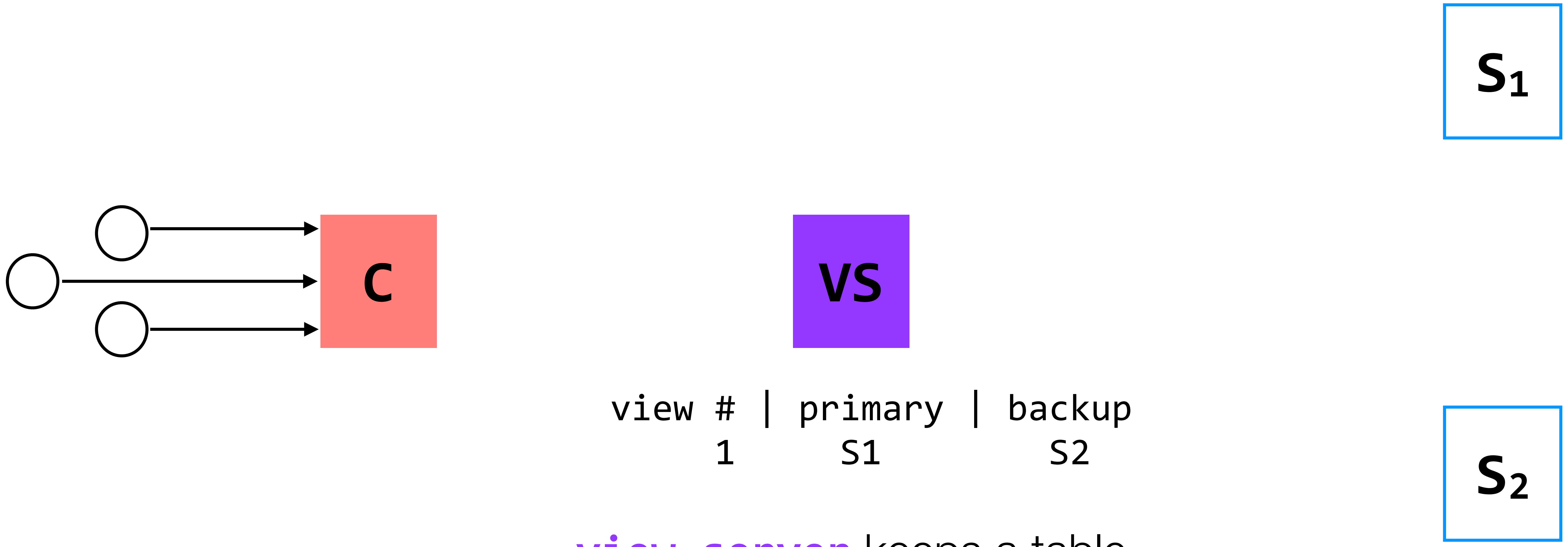


view #	primary	backup
1	S1	S2

`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

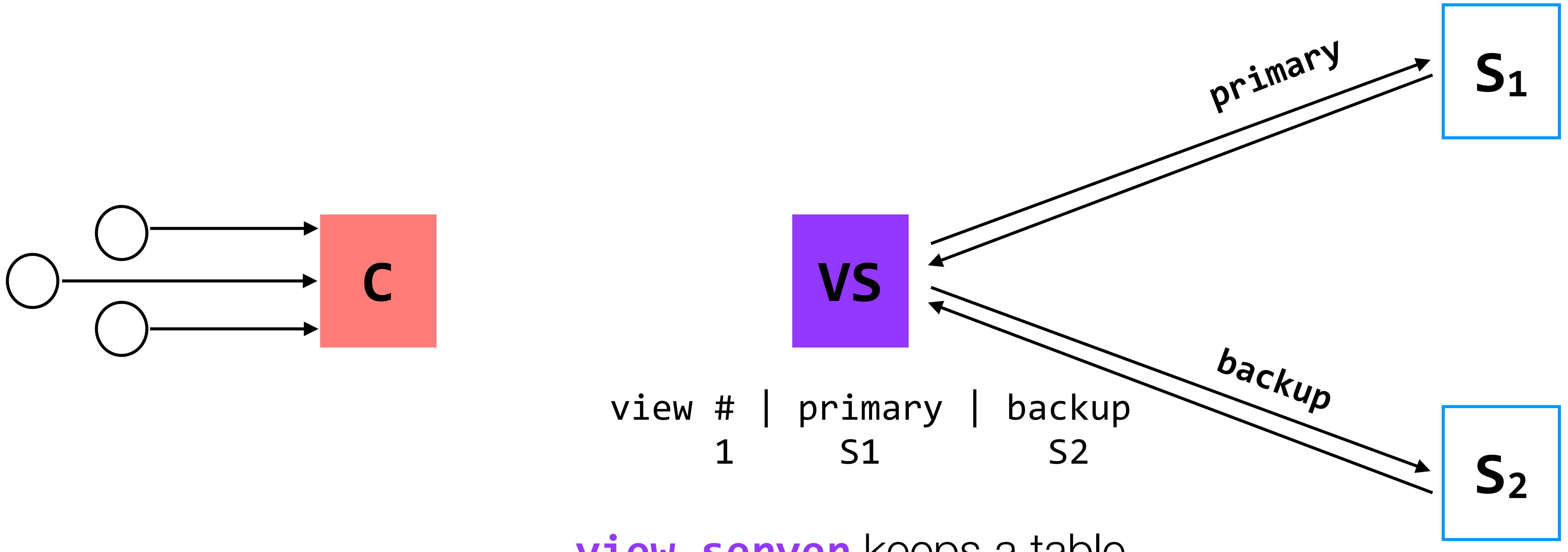


view server alerts **primary**/**backups** about their roles

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



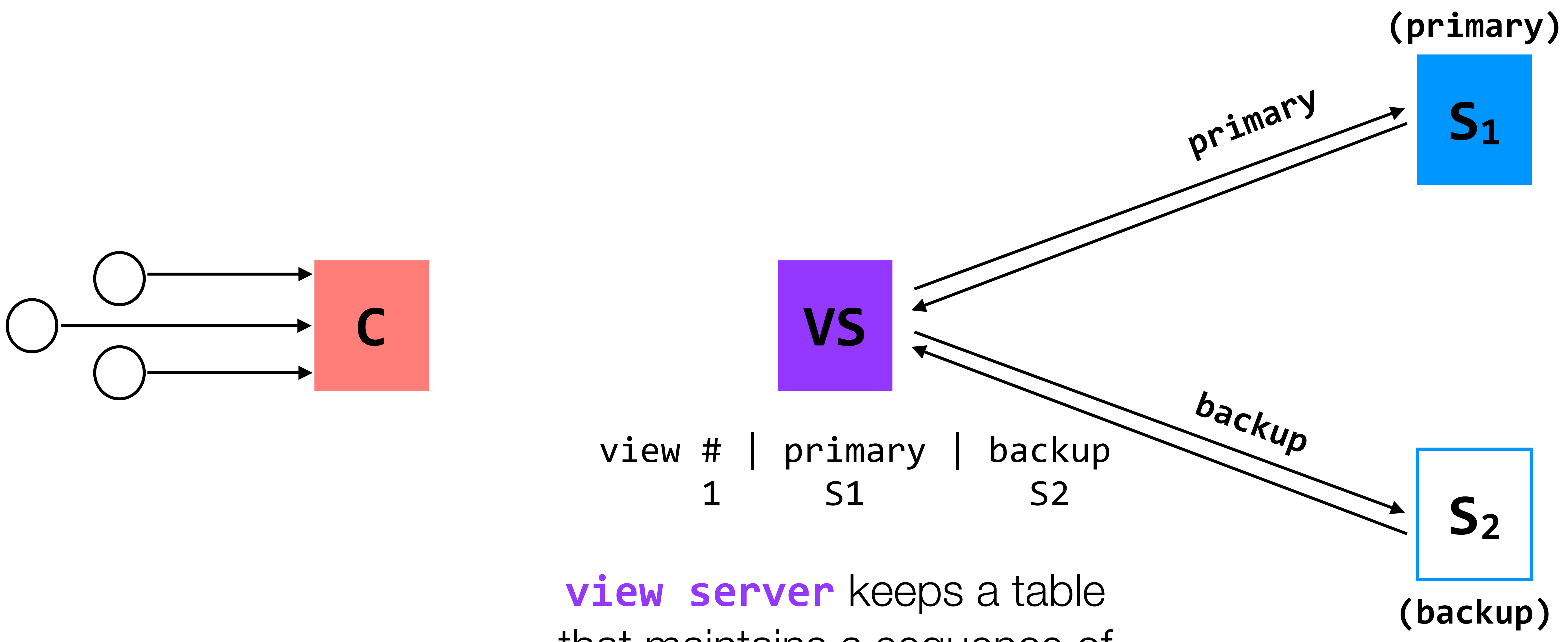
view # | primary | backup
1 S1 S2

view server keeps a table that maintains a sequence of views

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

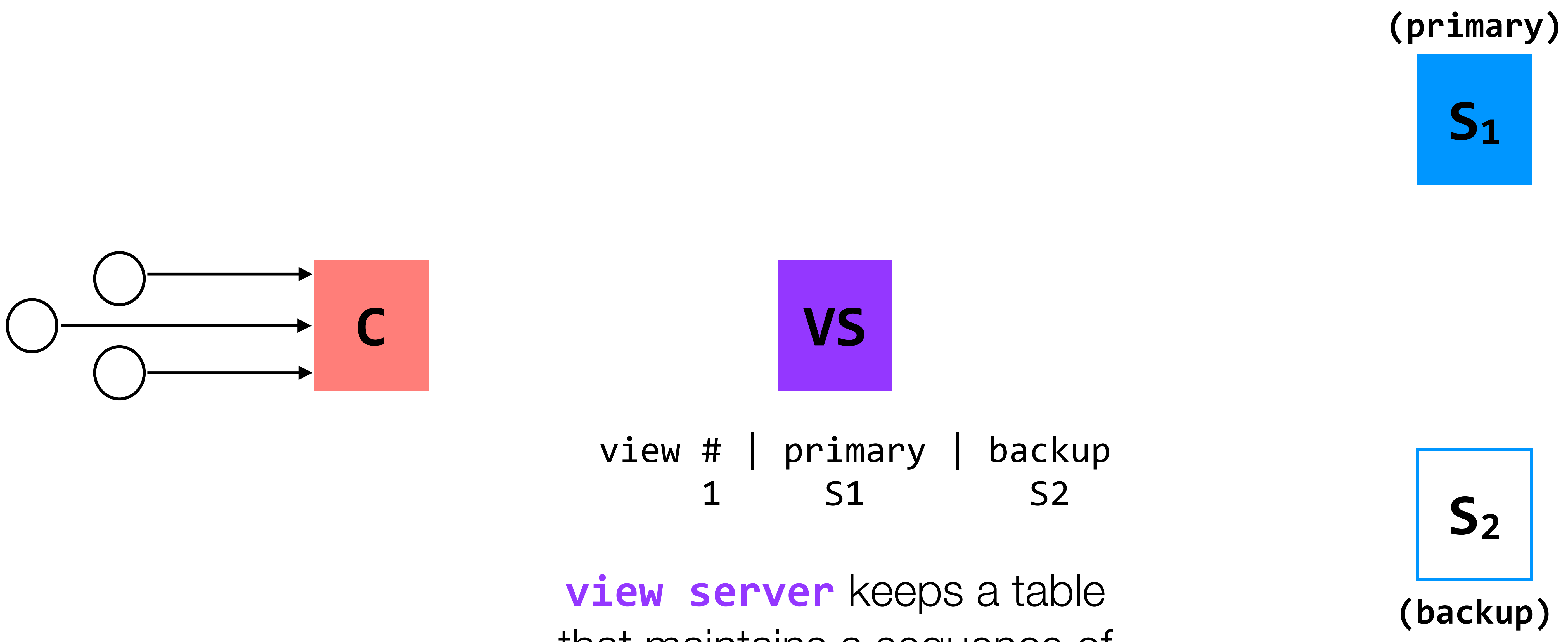


view server keeps a table that maintains a sequence of views

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



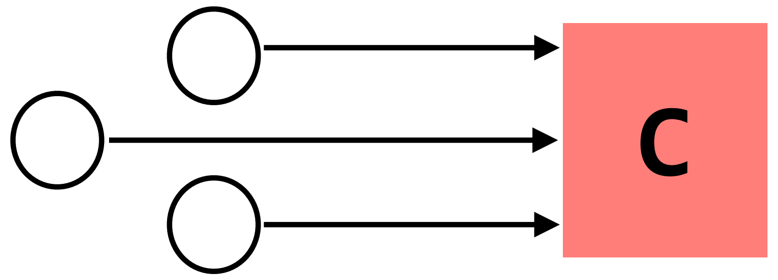
view #	primary	backup
1	S1	S2

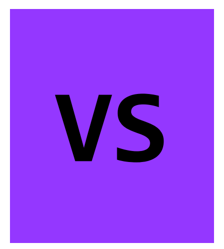
`view server` keeps a table that maintains a sequence of views

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

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





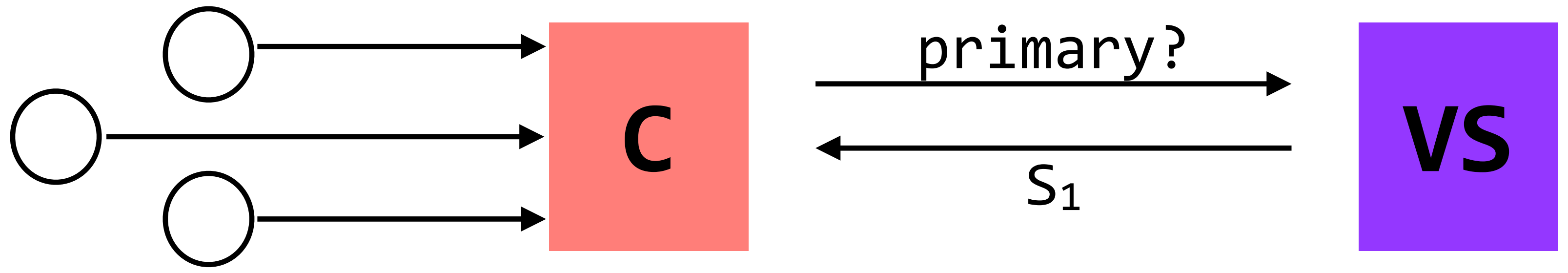
view #	primary	backup
1	S1	S2

`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

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



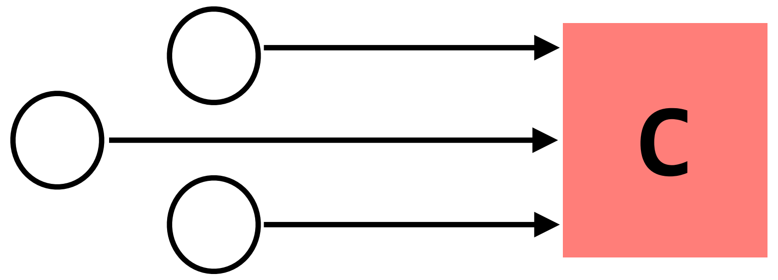
view #	primary	backup
1	S1	S2

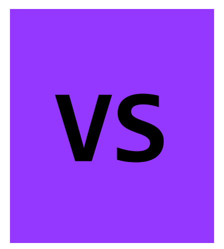
`view server` keeps a table that maintains a sequence of views

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

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





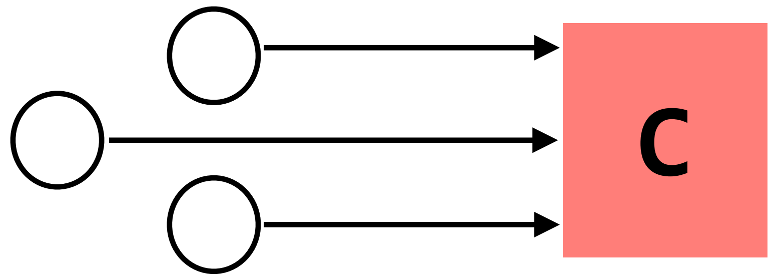
view #	primary	backup
1	S1	S2

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

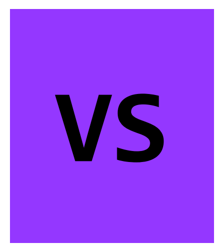
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

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

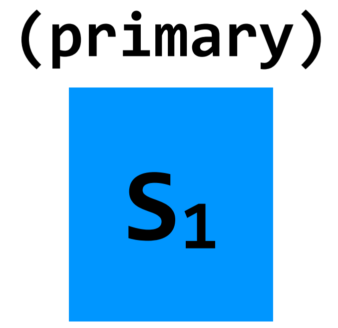


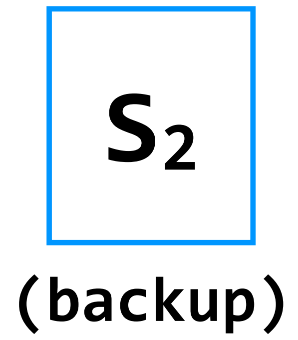
coordinators contact primary (as before)



view #	primary	backup
1	S1	S2

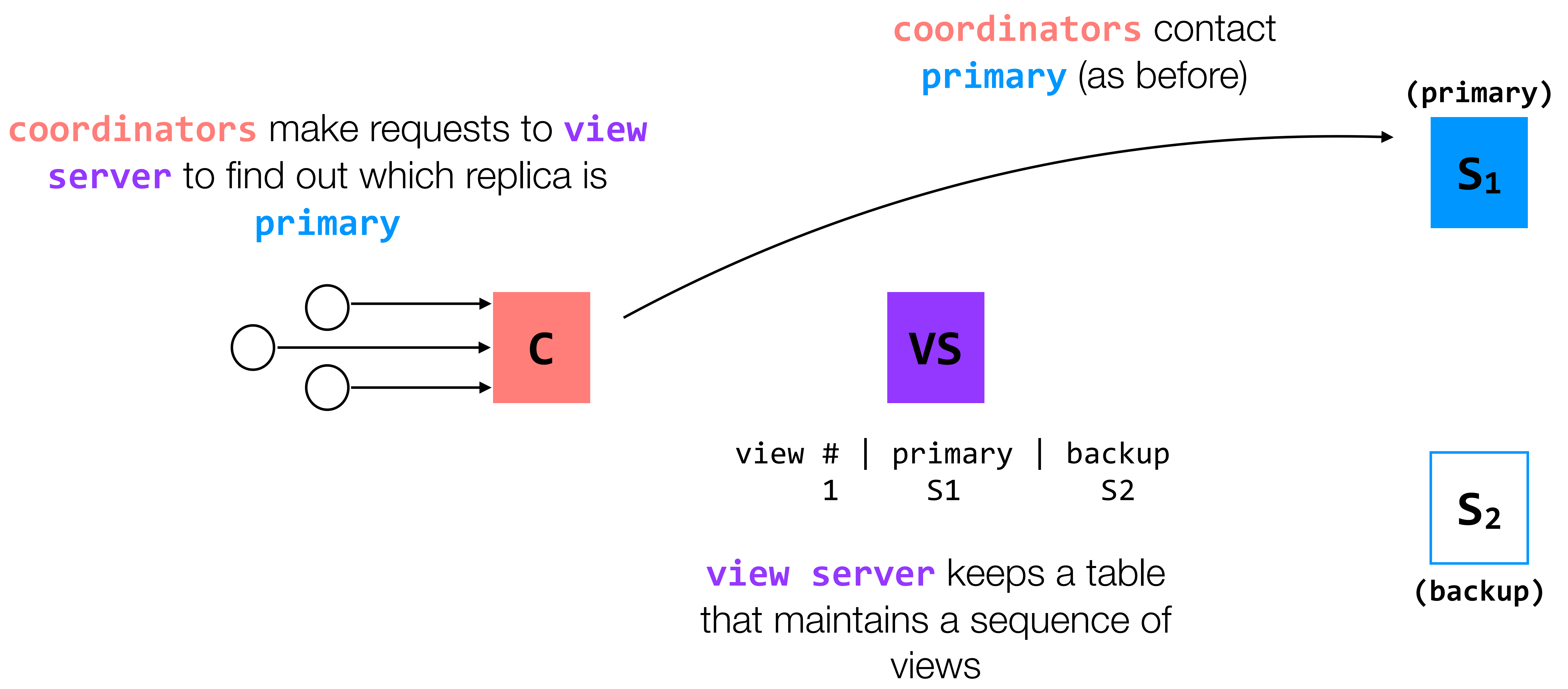
view server keeps a table that maintains a sequence of views





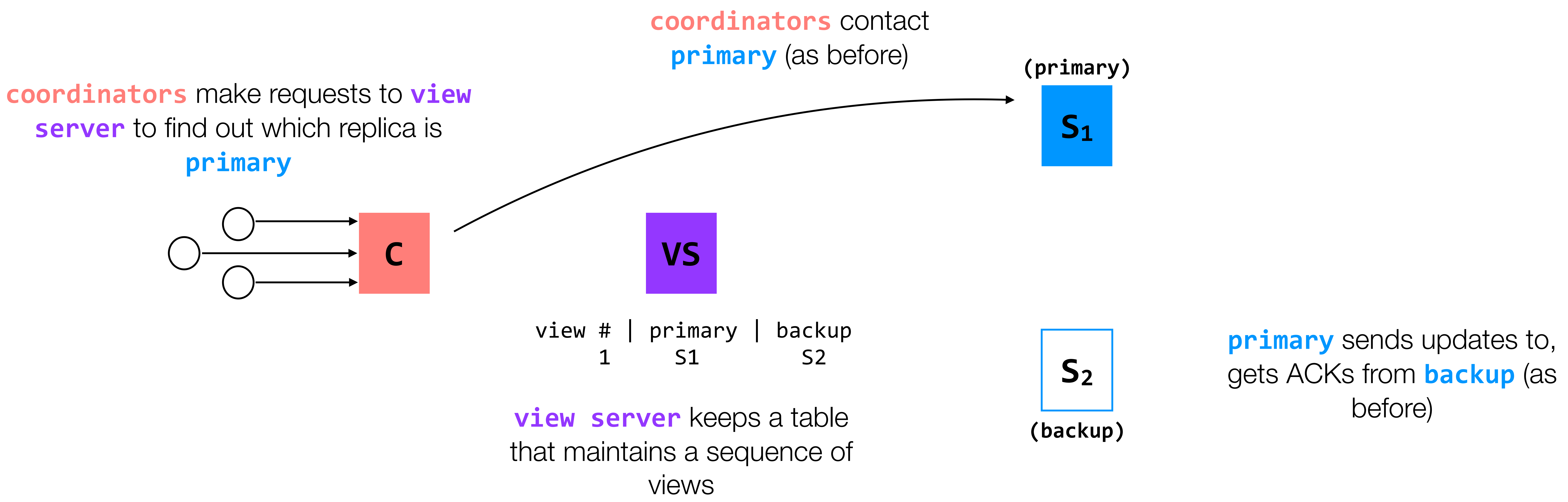
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



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

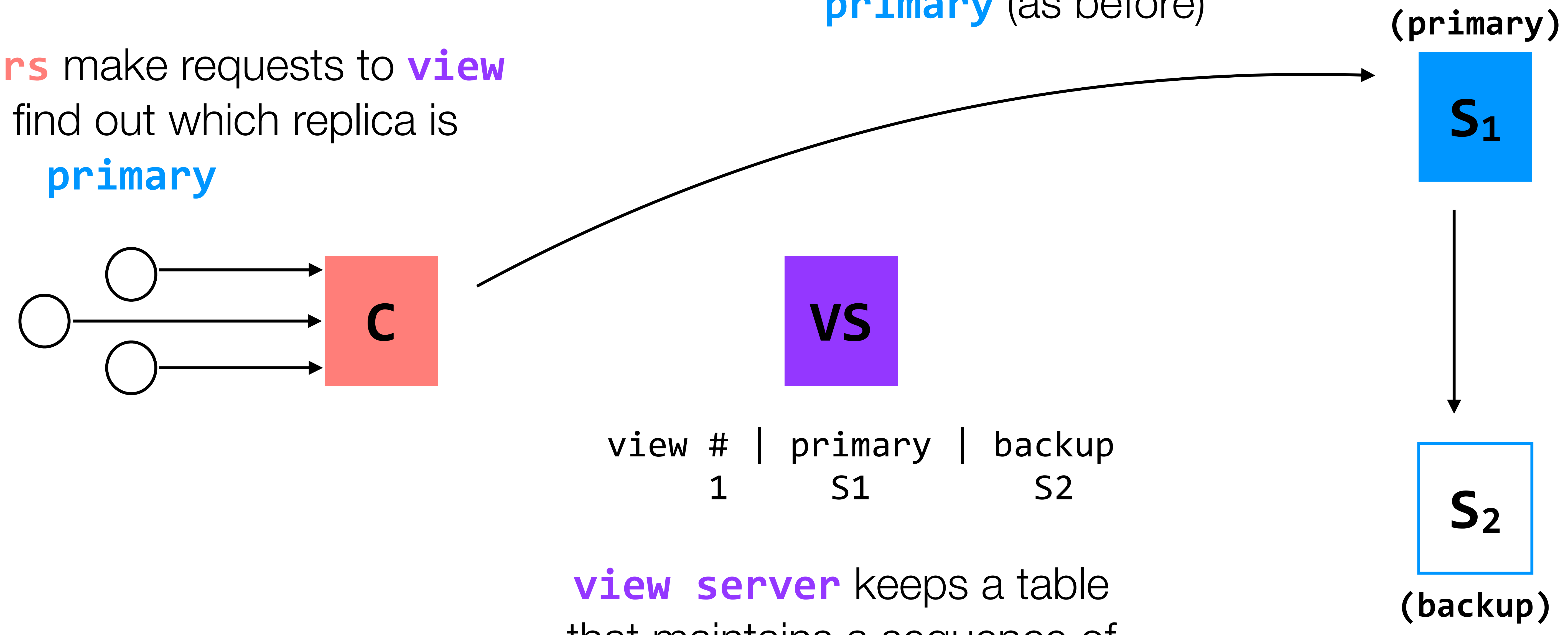


to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

coordinators contact primary (as before)

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



view #	primary	backup
1	S1	S2

view server keeps a table that maintains a sequence of views

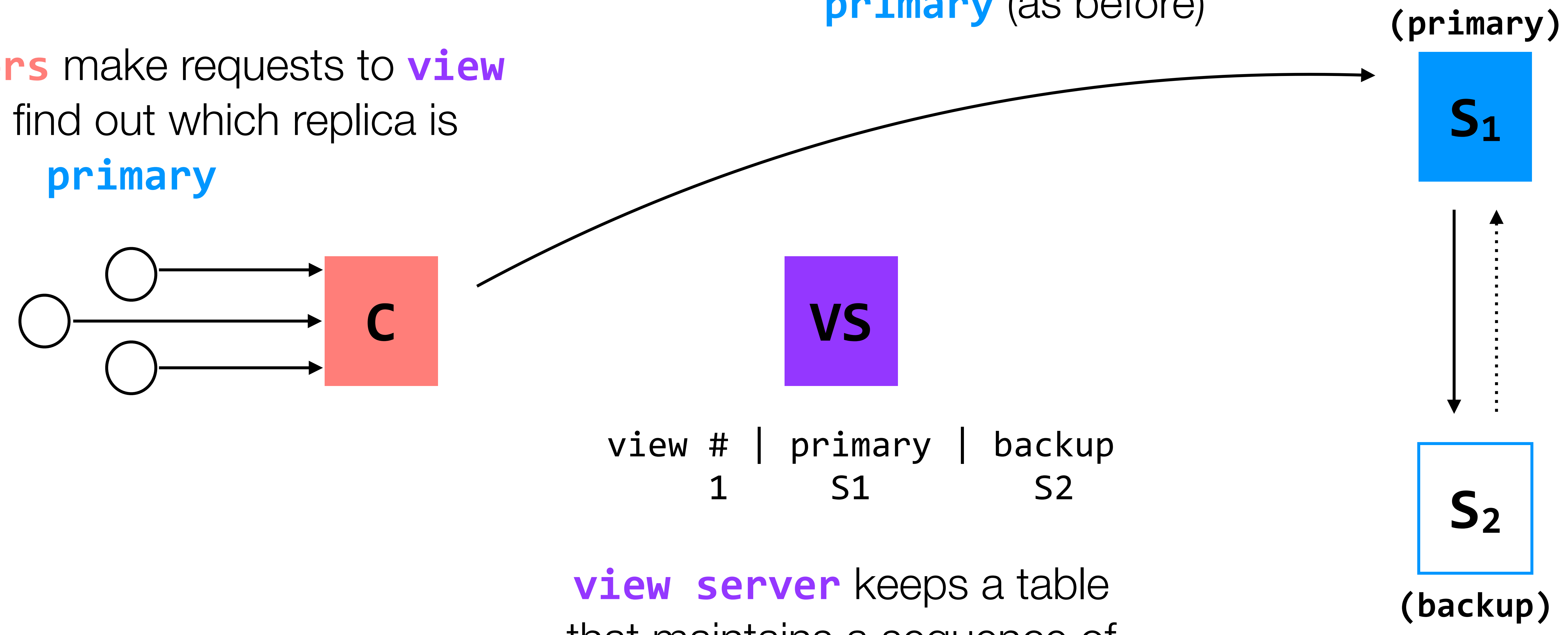
primary sends updates to, gets ACKs from backup (as before)

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

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

`coordinators` contact `primary` (as before)



view #	primary	backup
1	S1	S2

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

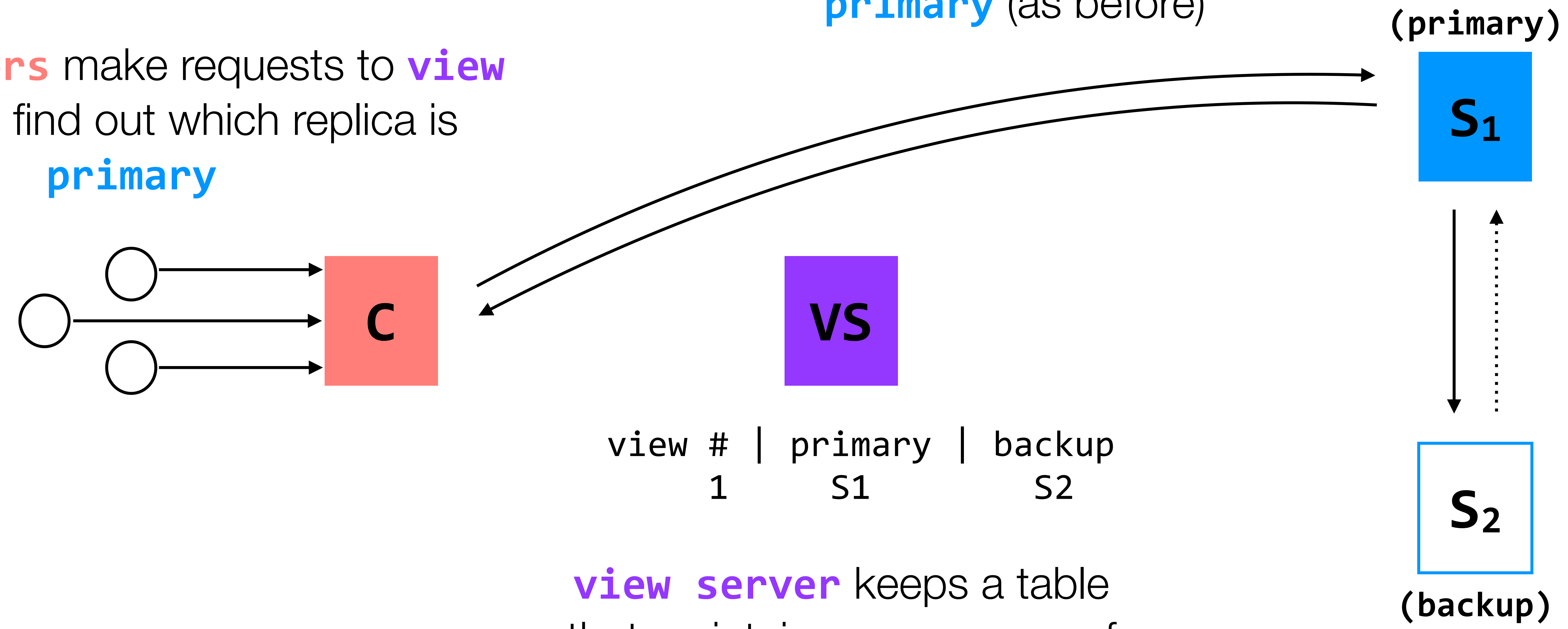
`primary` sends updates to, gets ACKs from `backup` (as before)

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

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

coordinators contact primary (as before)



view server keeps a table that maintains a sequence of views

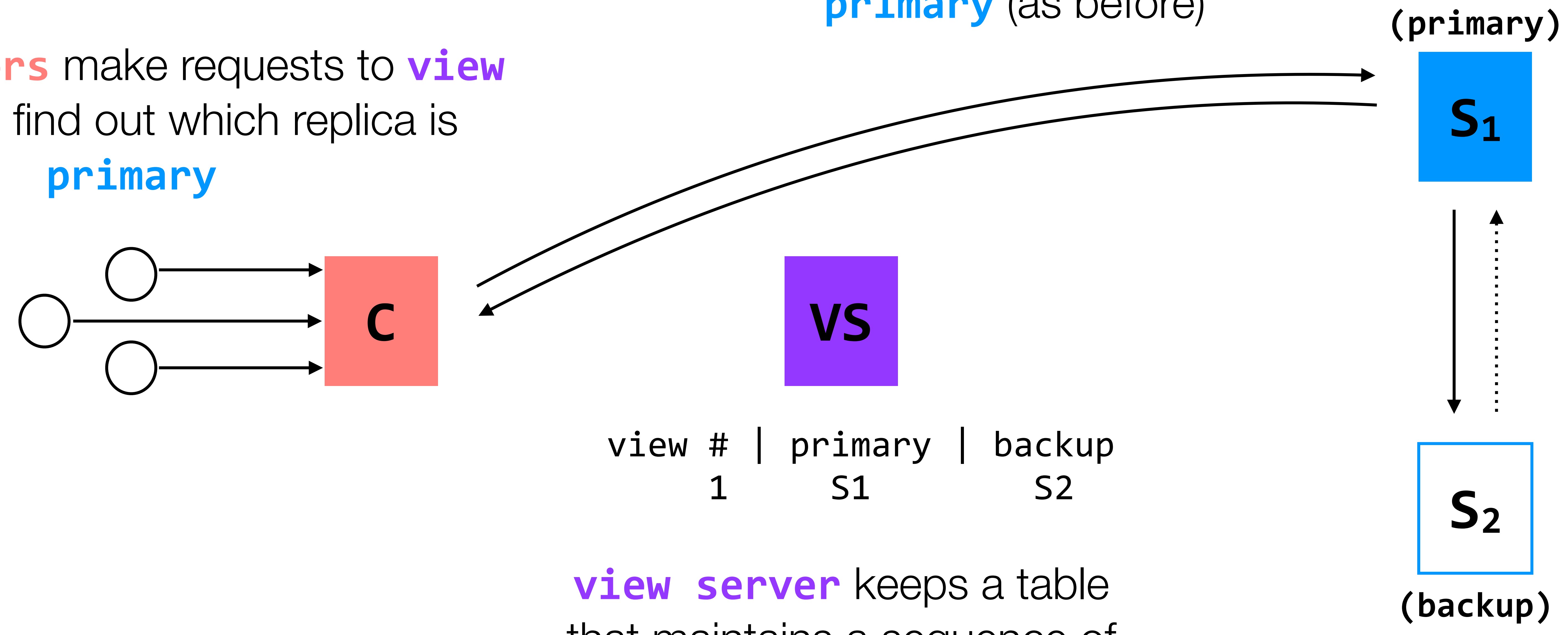
primary sends updates to, gets ACKs from backup (as before)

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

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

`coordinators` contact `primary` (as before)



view #	primary	backup
1	S1	S2

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

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

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

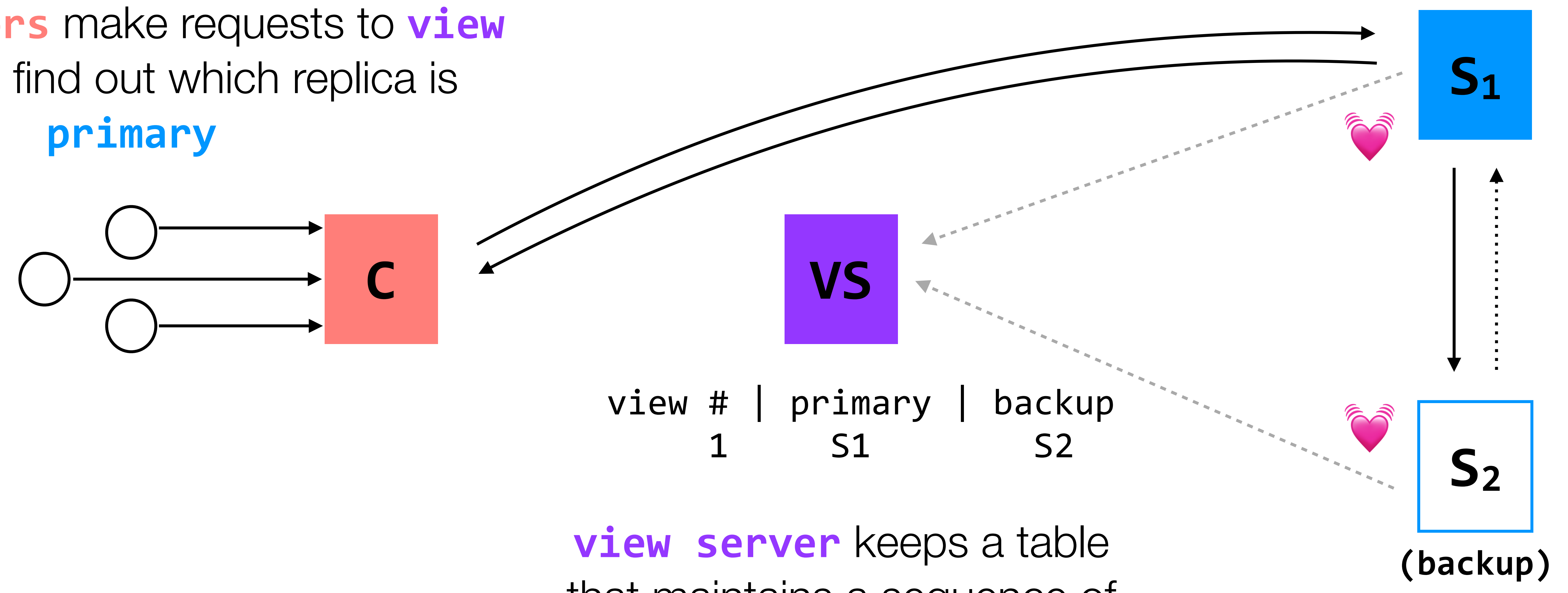
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

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

coordinators contact primary (as before)

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



view #	primary	backup
1	S1	S2

view server keeps a table that maintains a sequence of views

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

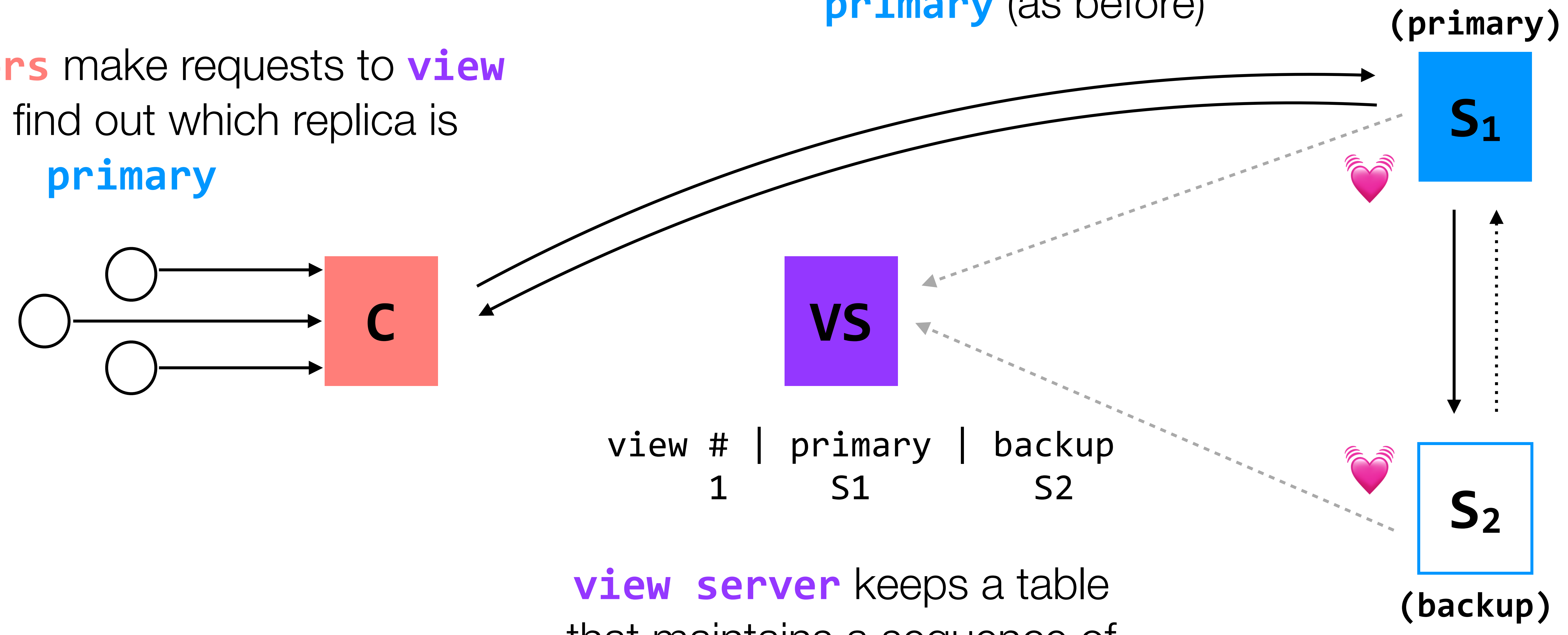
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

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

`coordinators` contact `primary` (as before)

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



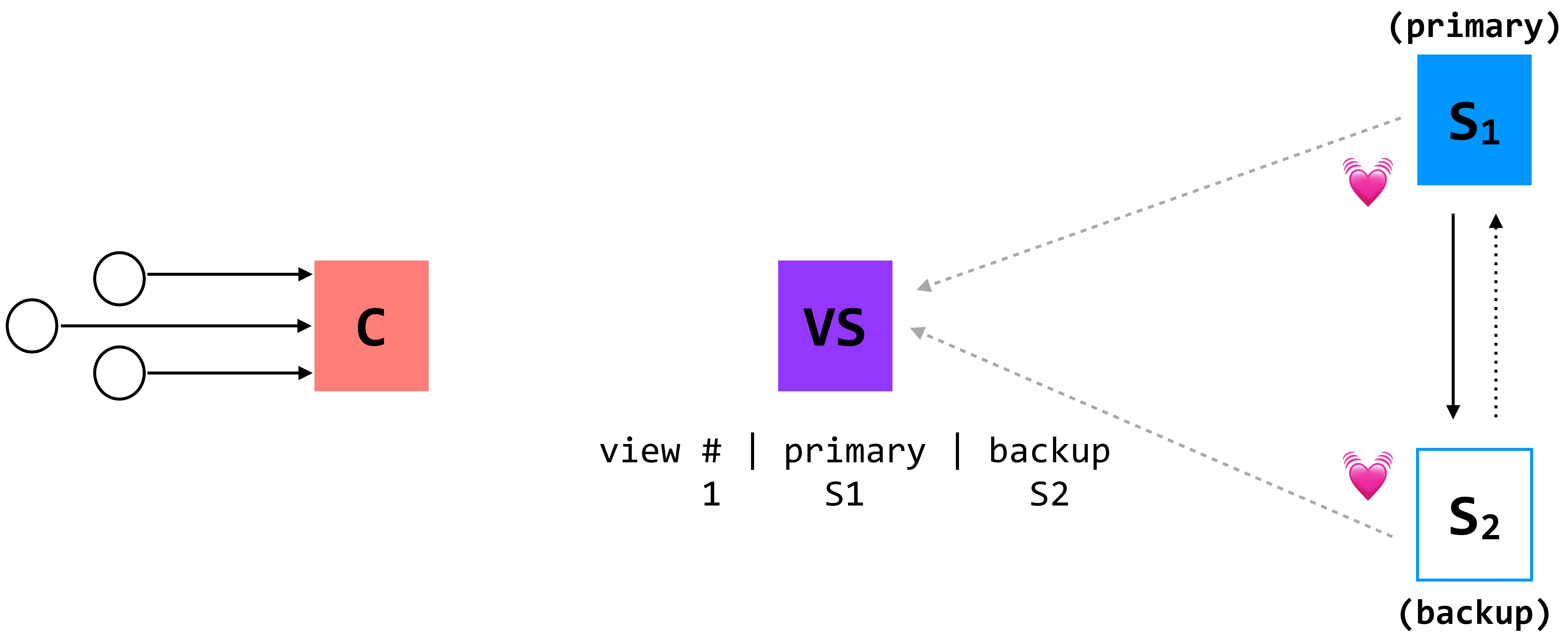
`primary` sends updates to, gets ACKs from `backup` (as before)

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

question: in our set-up, there is **one view server** for this entire system, whereas there can be multiple coordinators. why might having a single view server help us when failures (such as the examples you've already seen) occur?

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

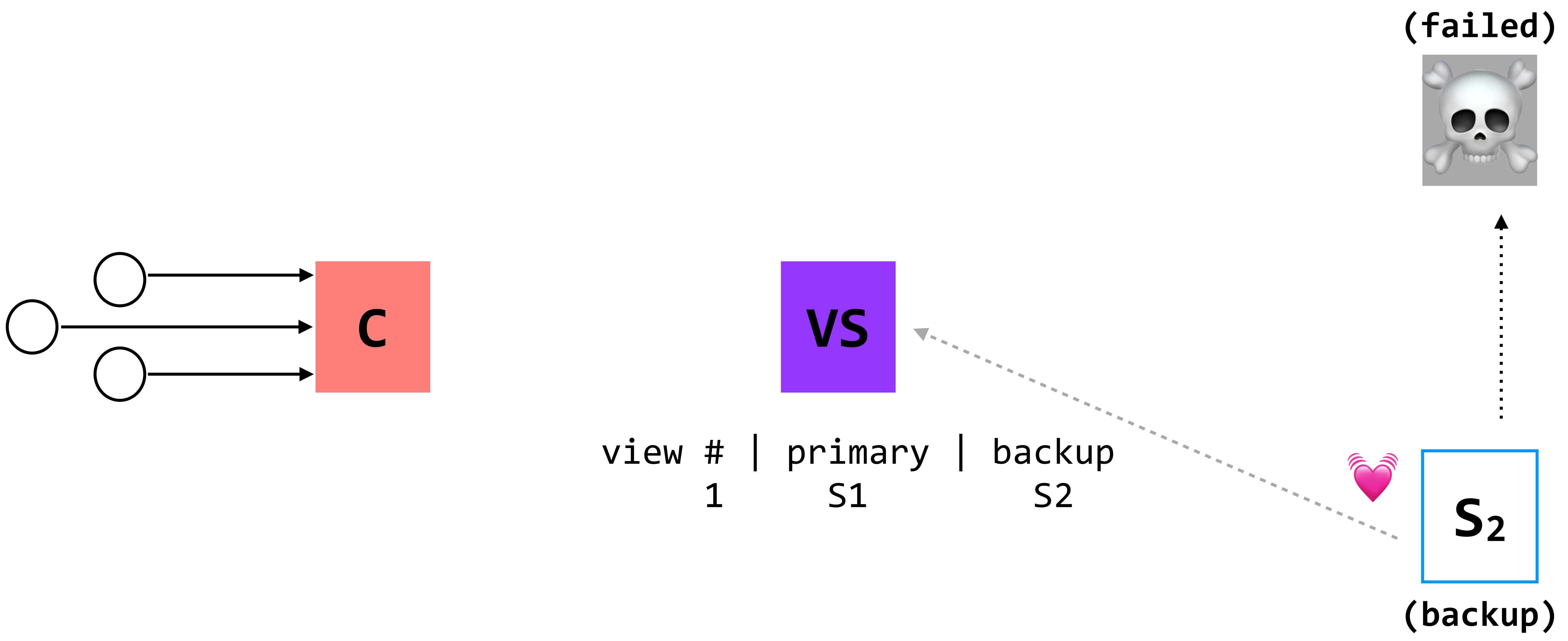
attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



what happens if the primary replica fails?

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

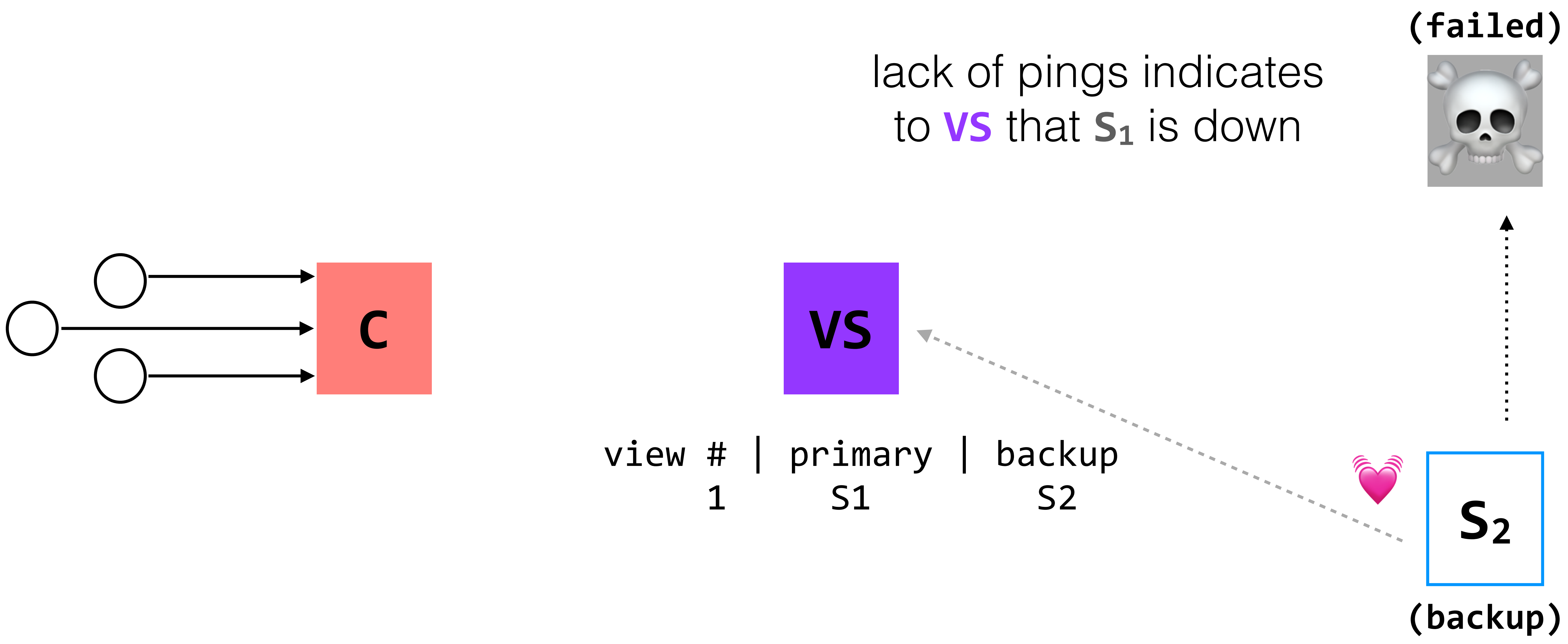
attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



what happens if the primary replica fails?

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

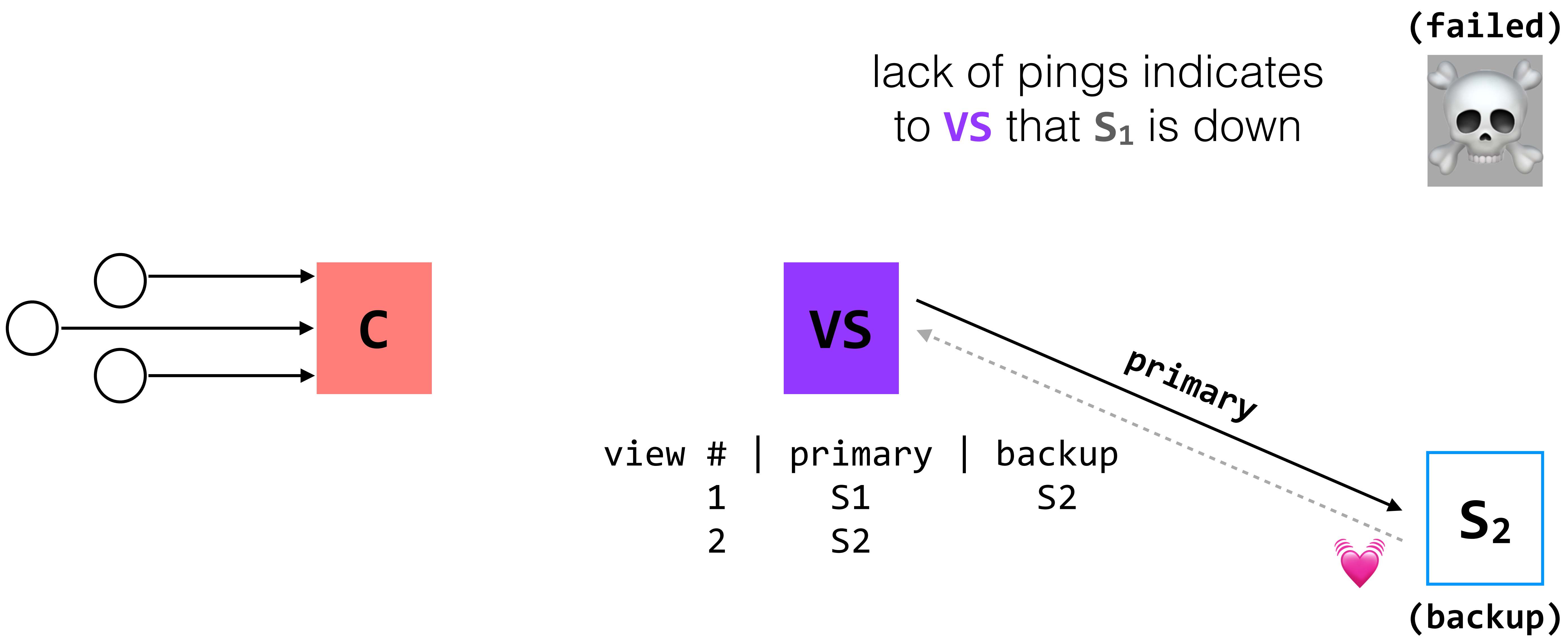
attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



what happens if the primary replica fails?

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

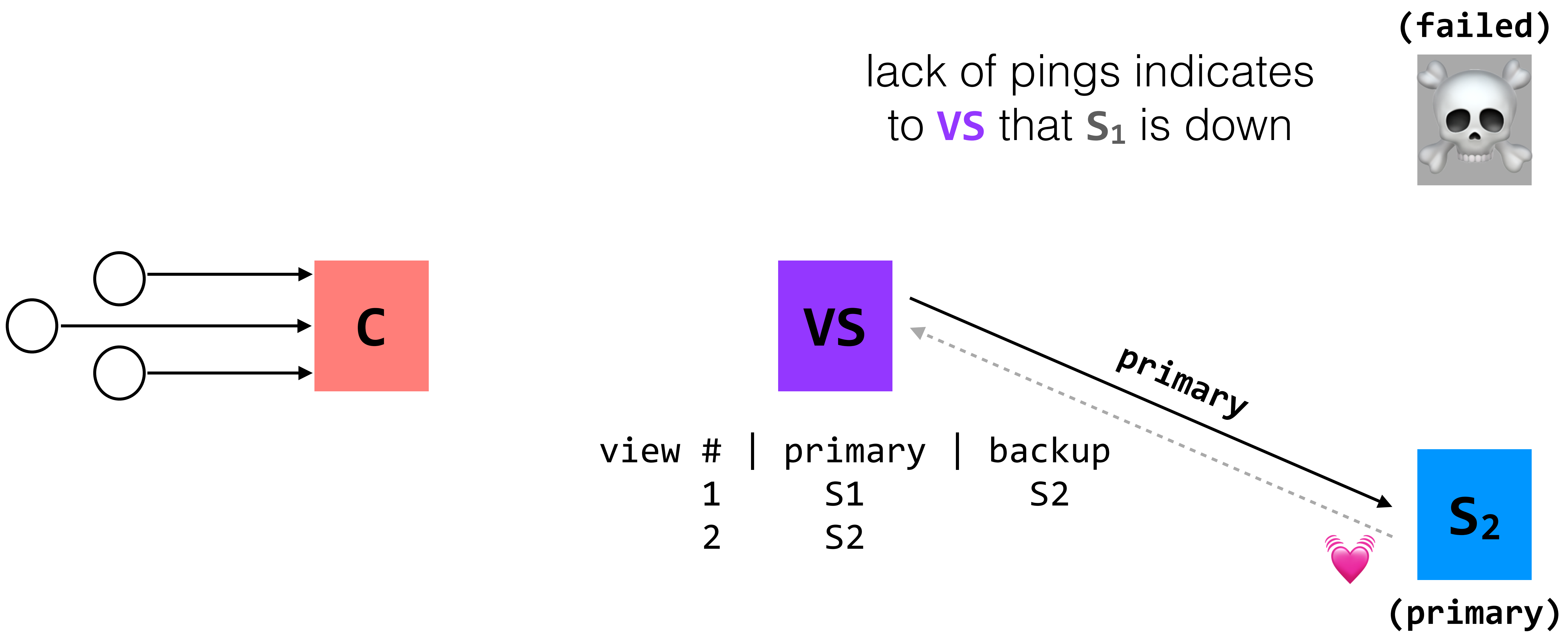
attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



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

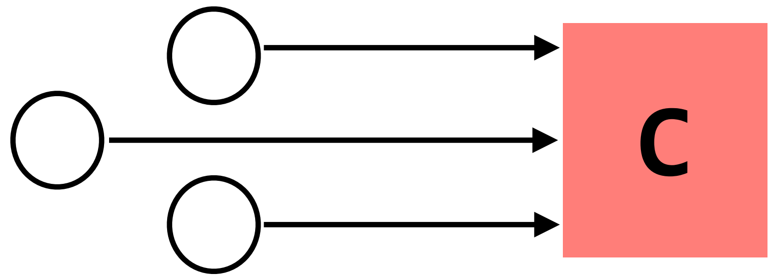


what happens if the primary replica fails?

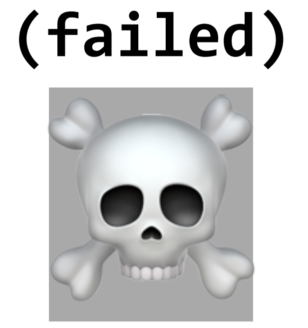
to increase availability, let's try replicating data on two servers

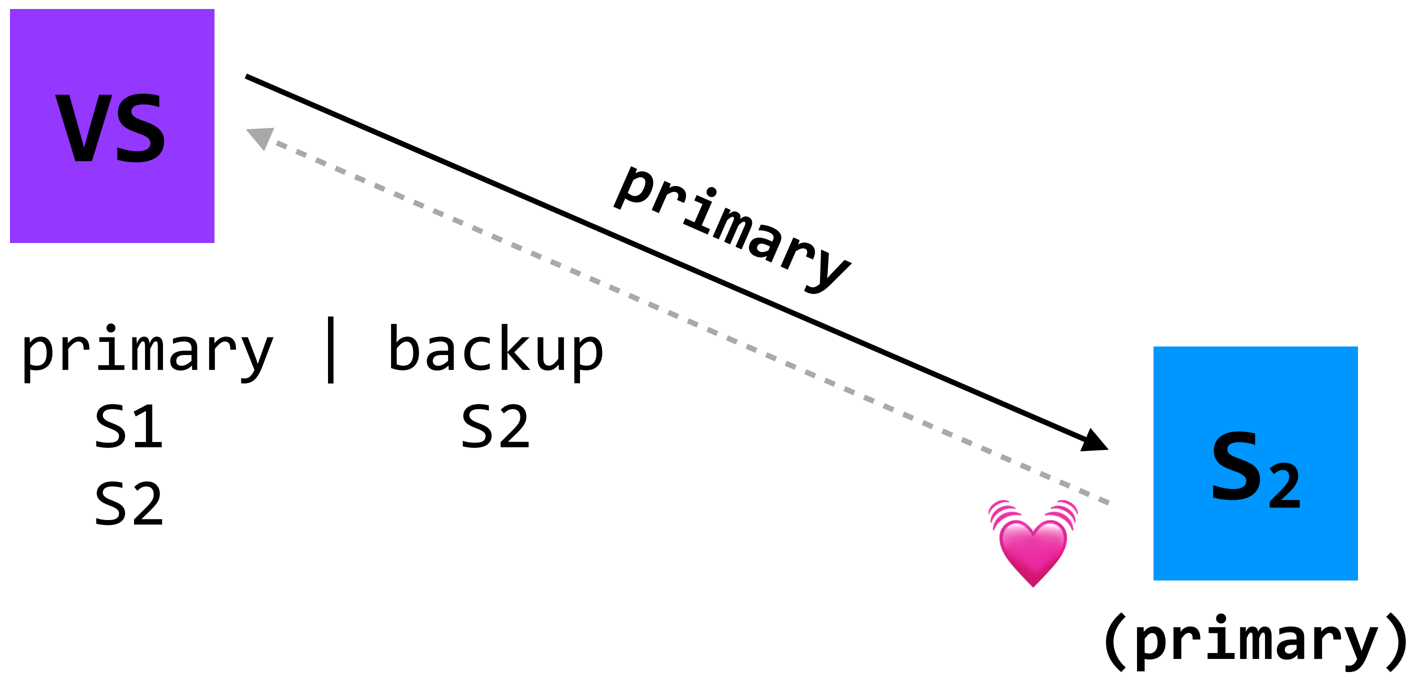
attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

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





view #	primary	backup
1	S1	S2
2	S2	

what happens if the primary replica fails?

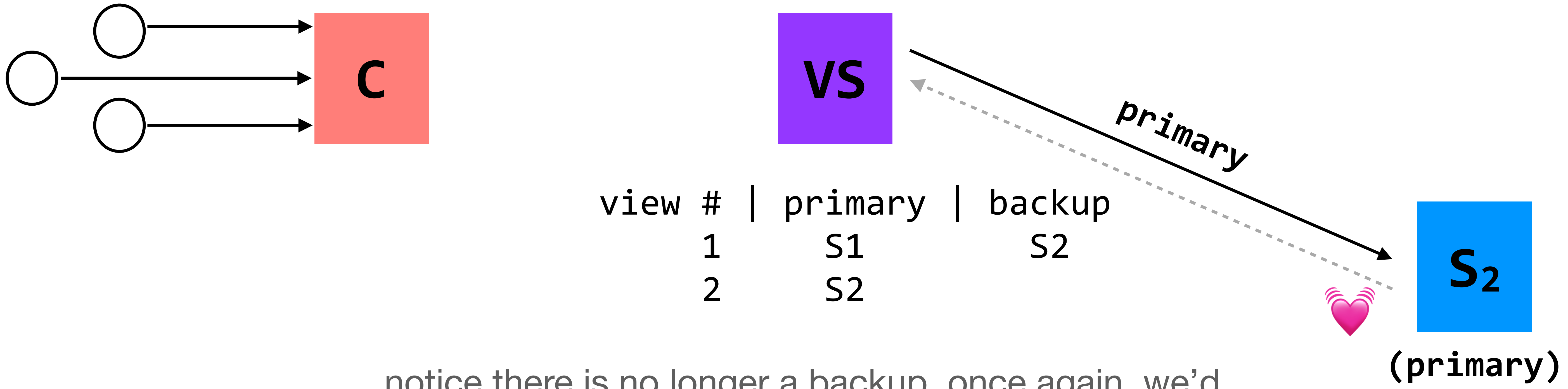
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions

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

(failed)



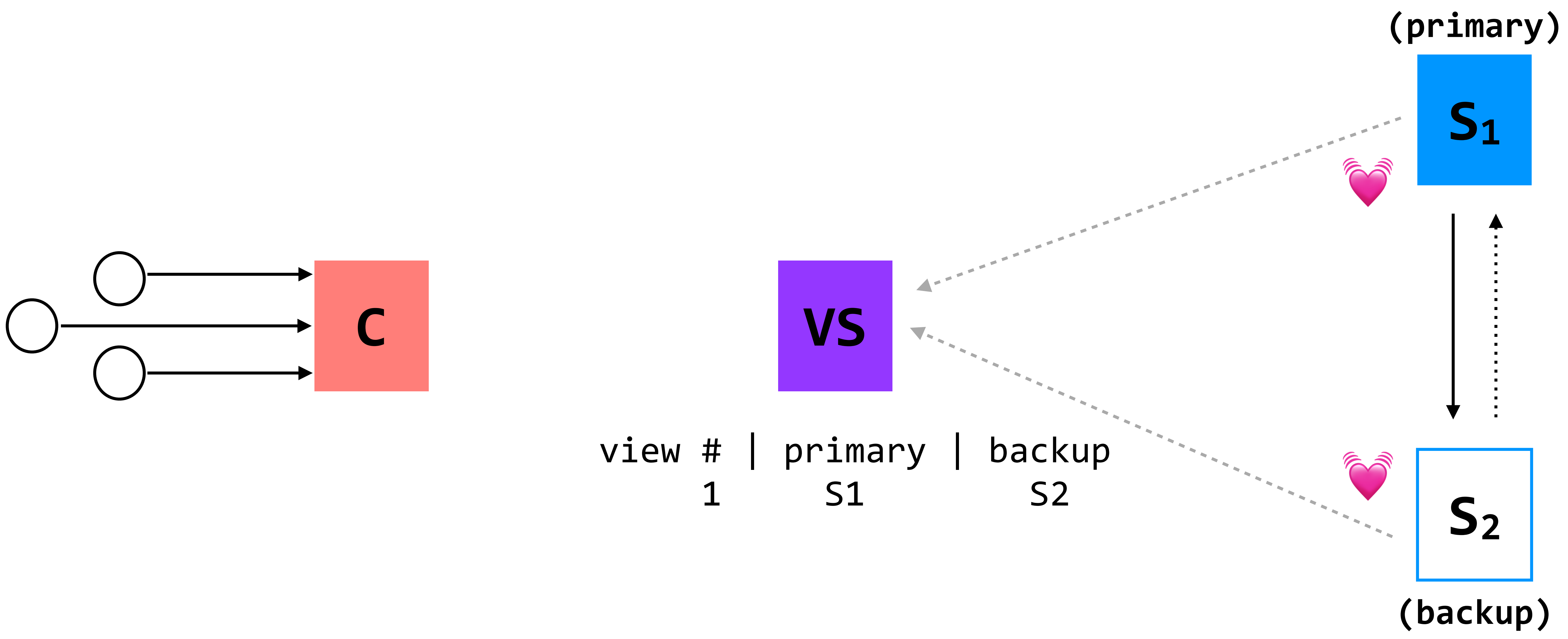
view #	primary	backup
1	S1	S2
2	S2	

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?

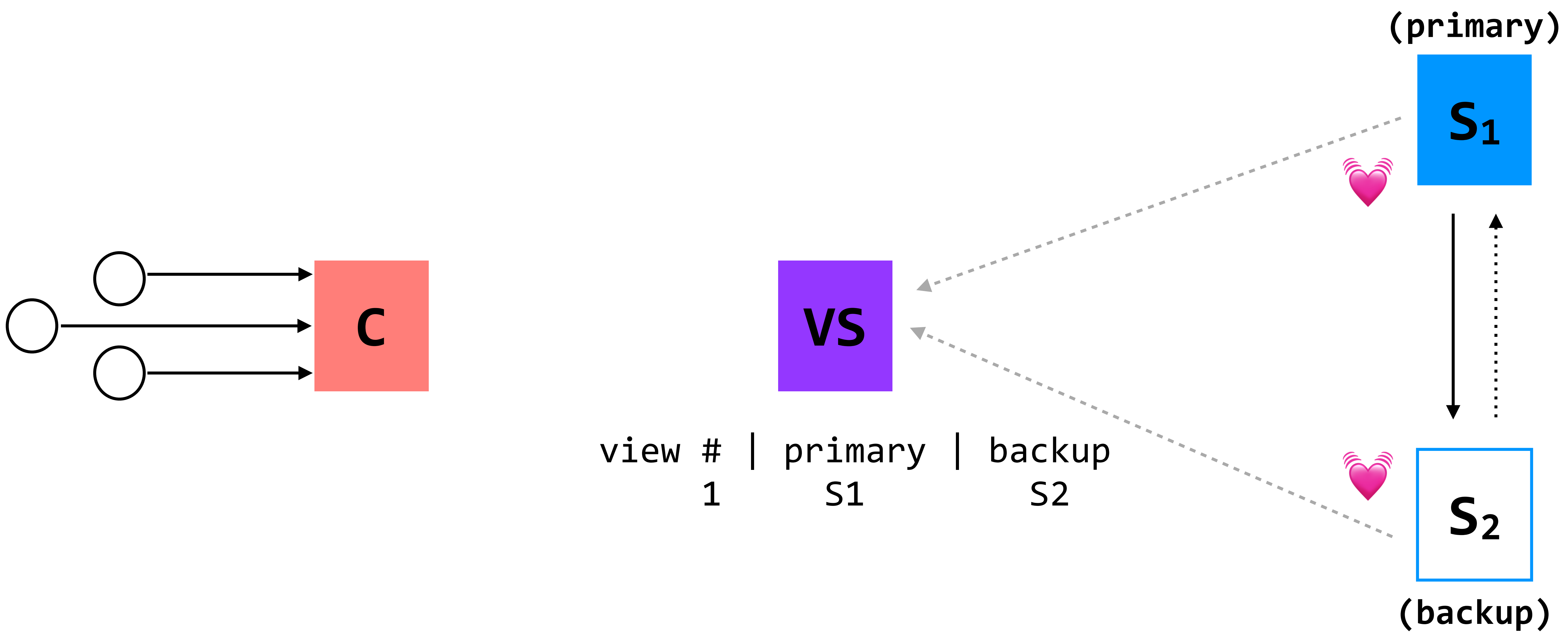
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



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

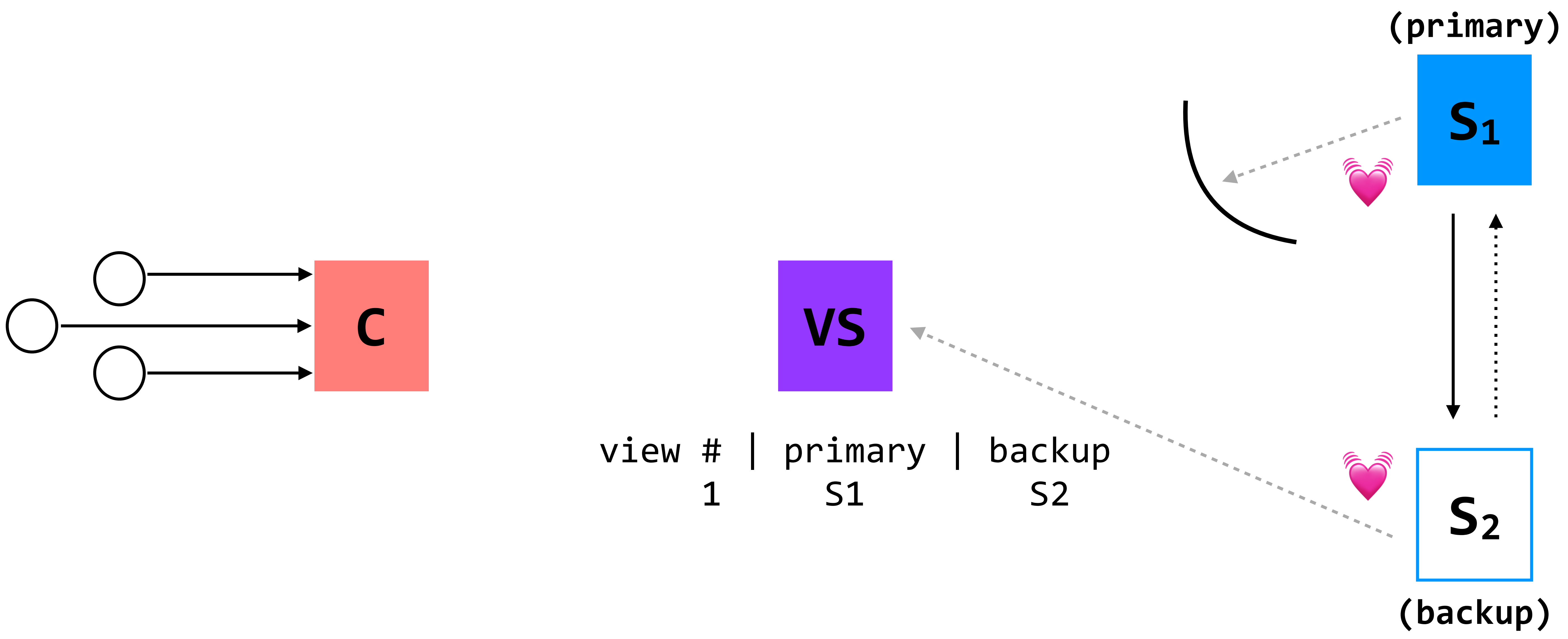
attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



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

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

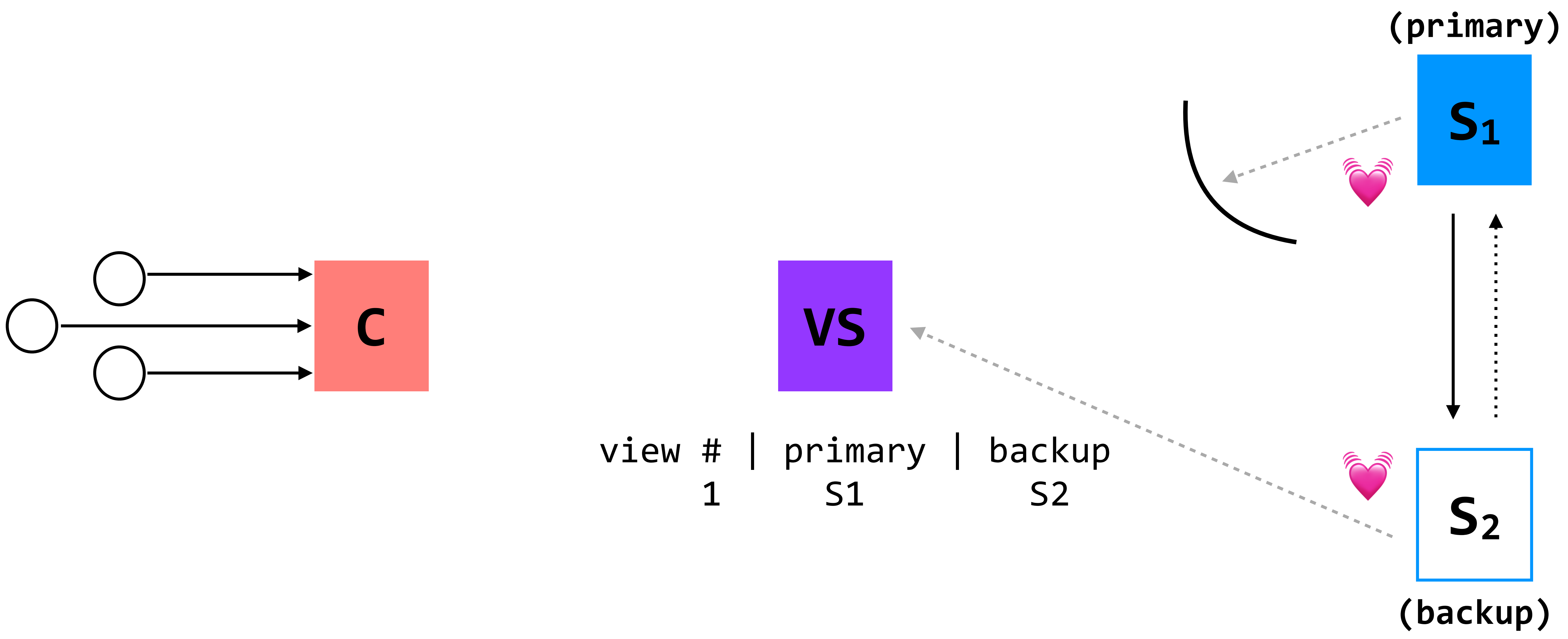
attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



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

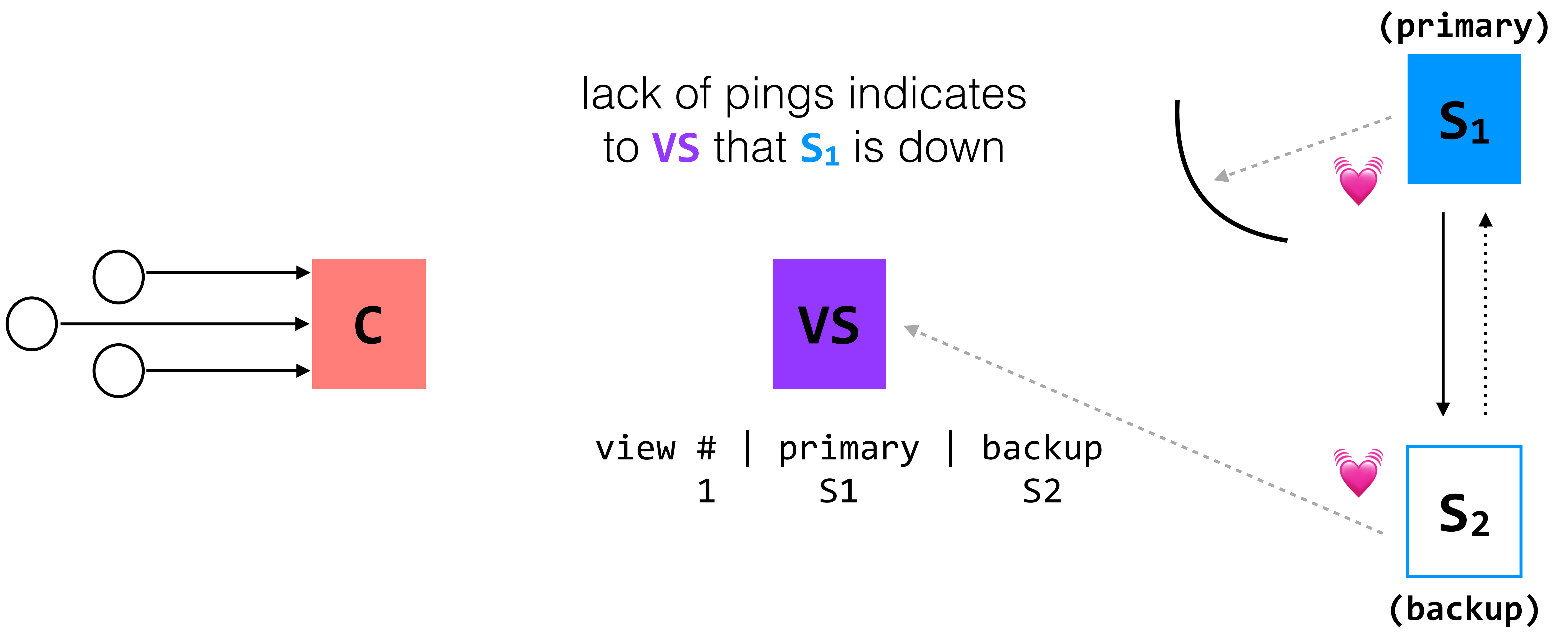


what happens if a network partition prevents S_1 from communicating with VS?

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

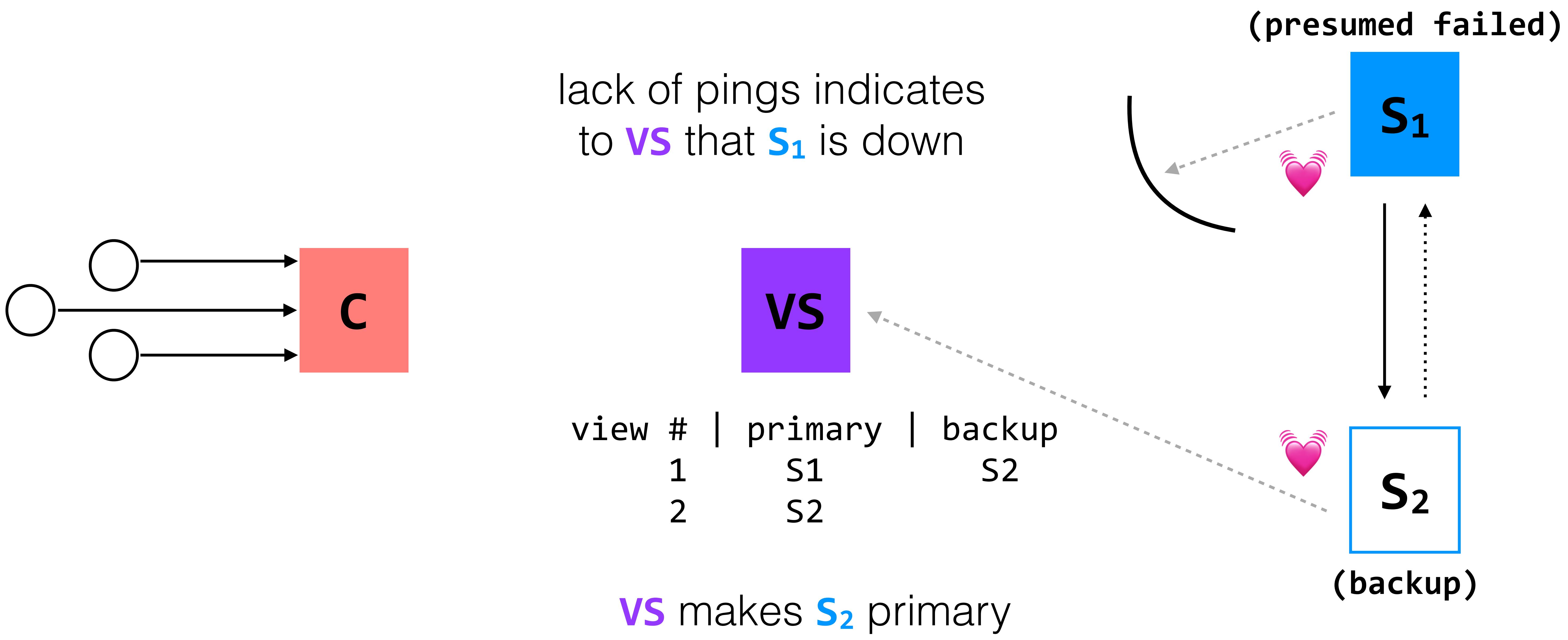


what happens if a network partition prevents S_1 from communicating with VS?

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

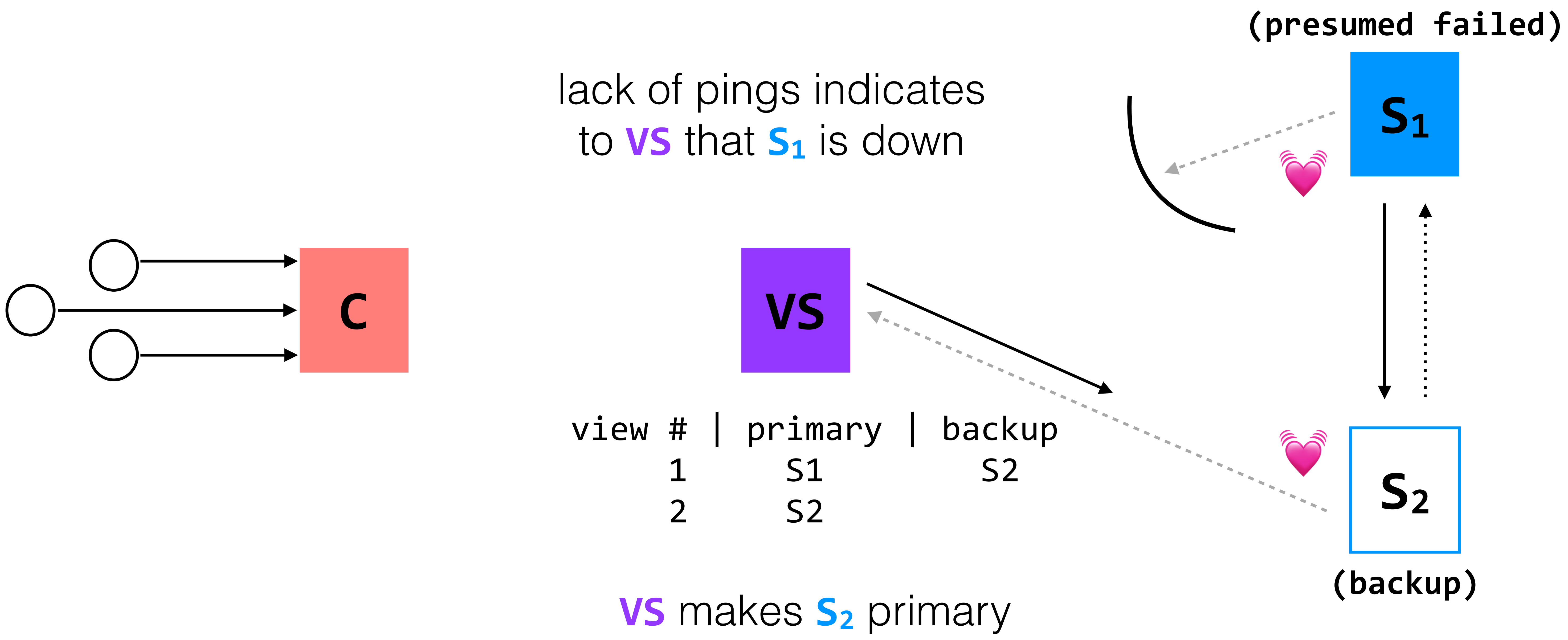


what happens if a network partition prevents S_1 from communicating with VS?

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

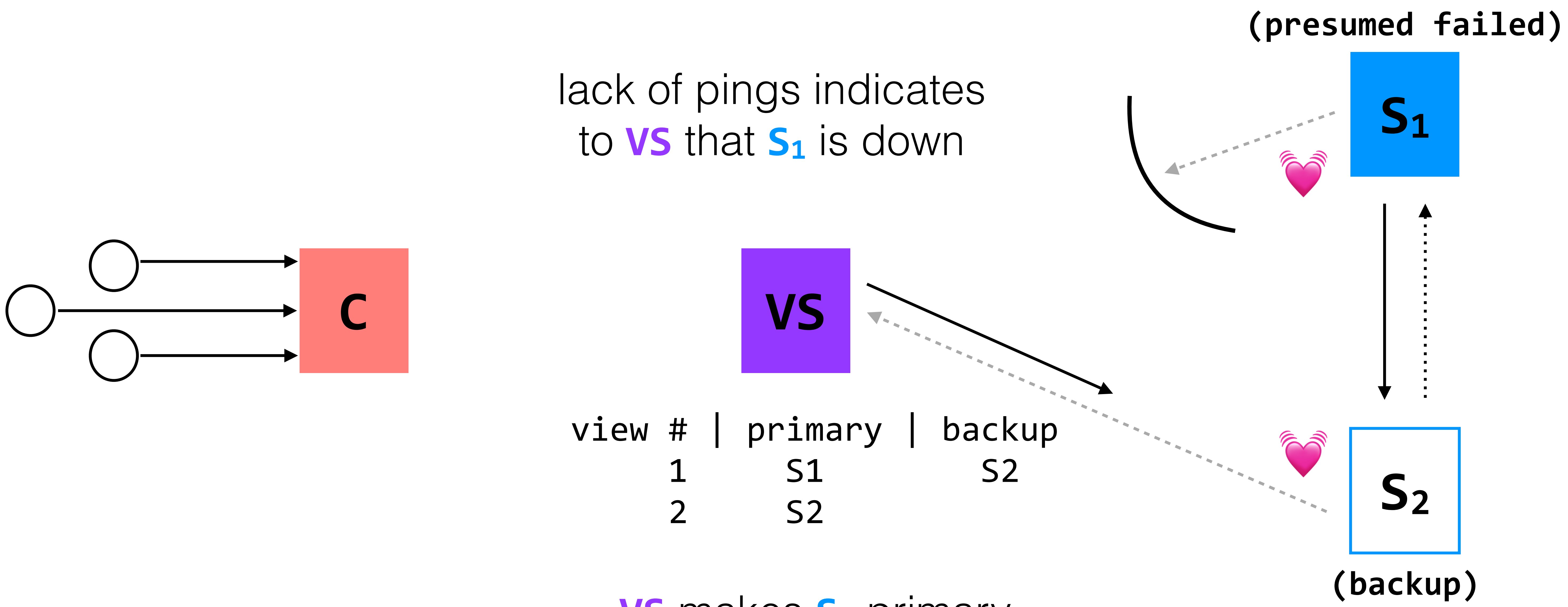


what happens if a network partition prevents S_1 from communicating with VS?

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



view #	primary	backup
1	S1	S2
2	S2	

VS makes S_2 primary

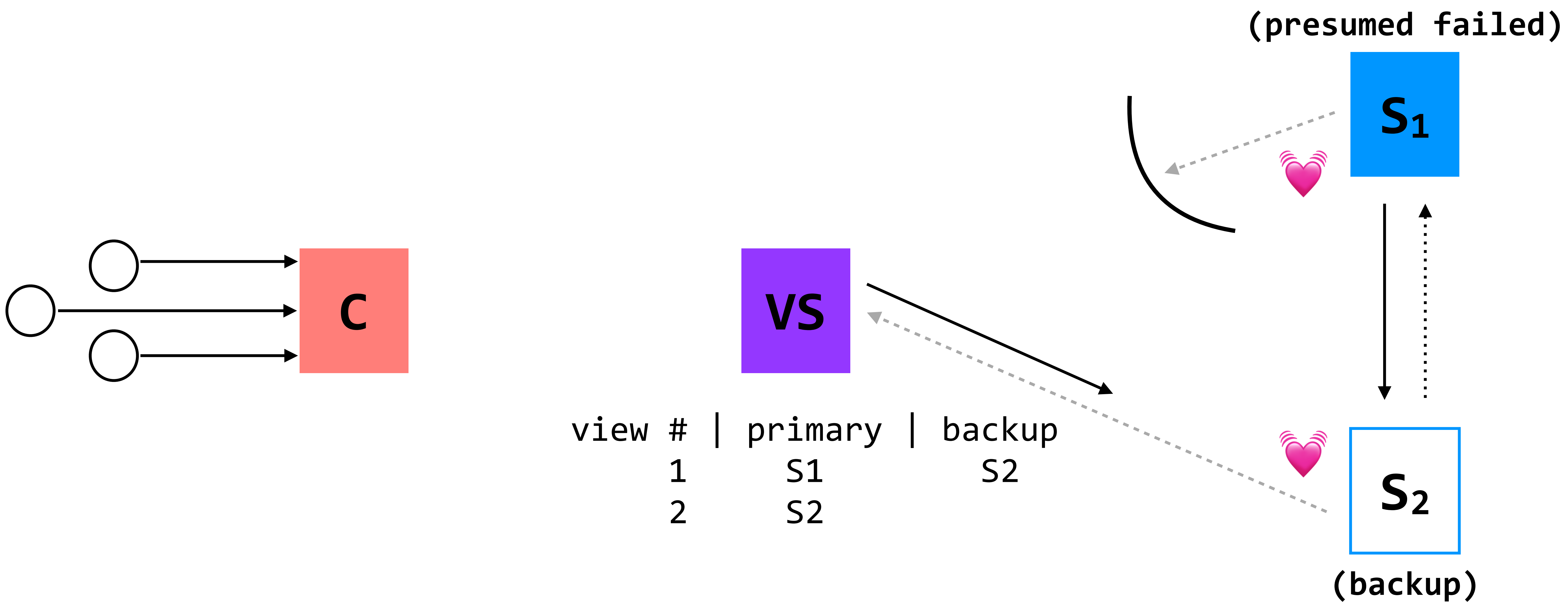
at this stage, VS thinks S_2 is primary; S_2 and S_1 think S_1 is primary

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

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




```
view # | primary | backup
   1       S1        S2
   2       S2
```

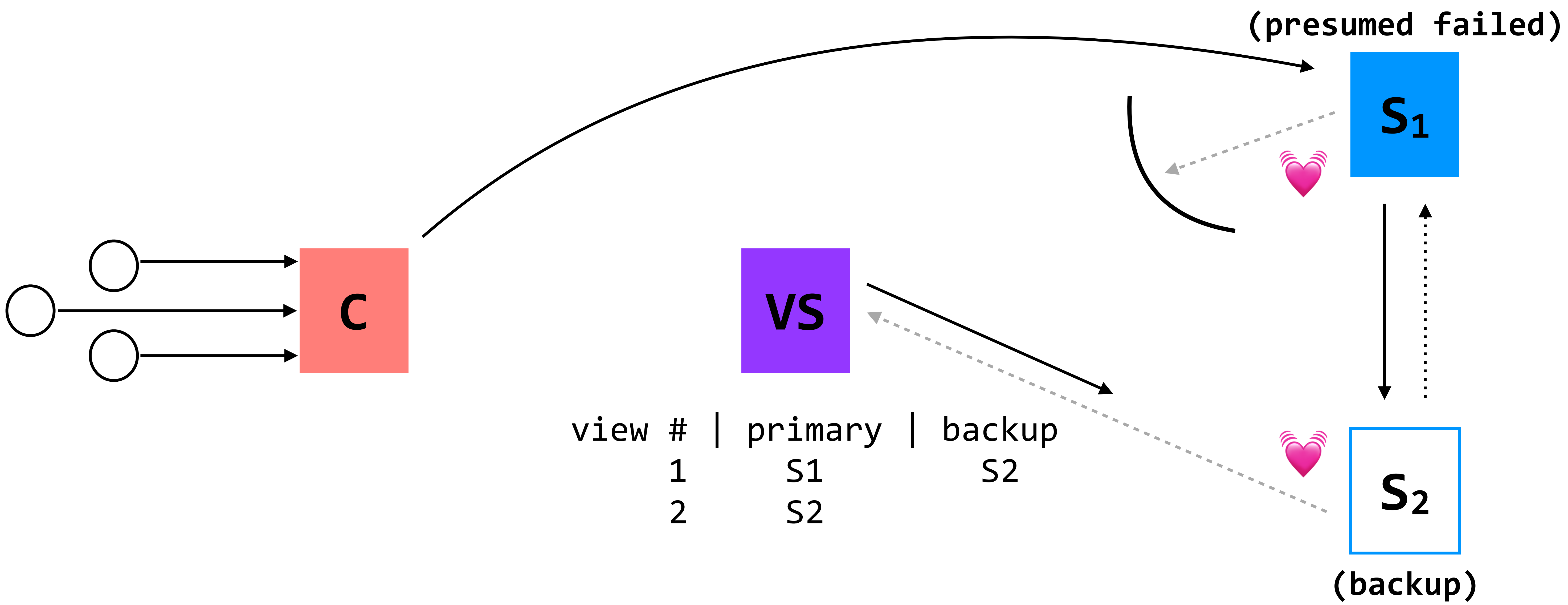
at this stage, **VS** thinks **S_2** is primary; **S_2** and **S_1** think **S_1** is primary

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**

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



at this stage, **VS** thinks **S_2** is primary; **S_2** and **S_1** think **S_1** is primary

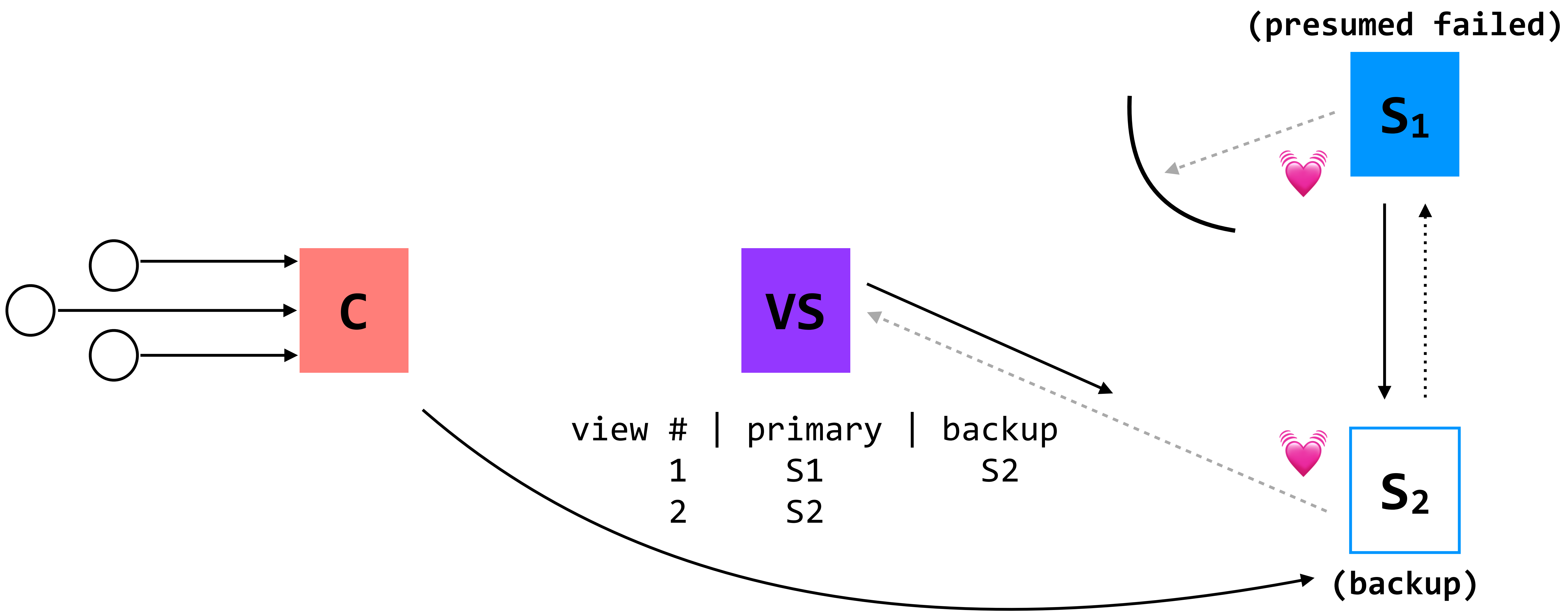
if **S_1** receives any requests from **C**, it will behave as primary with **S_2** as backup

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



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

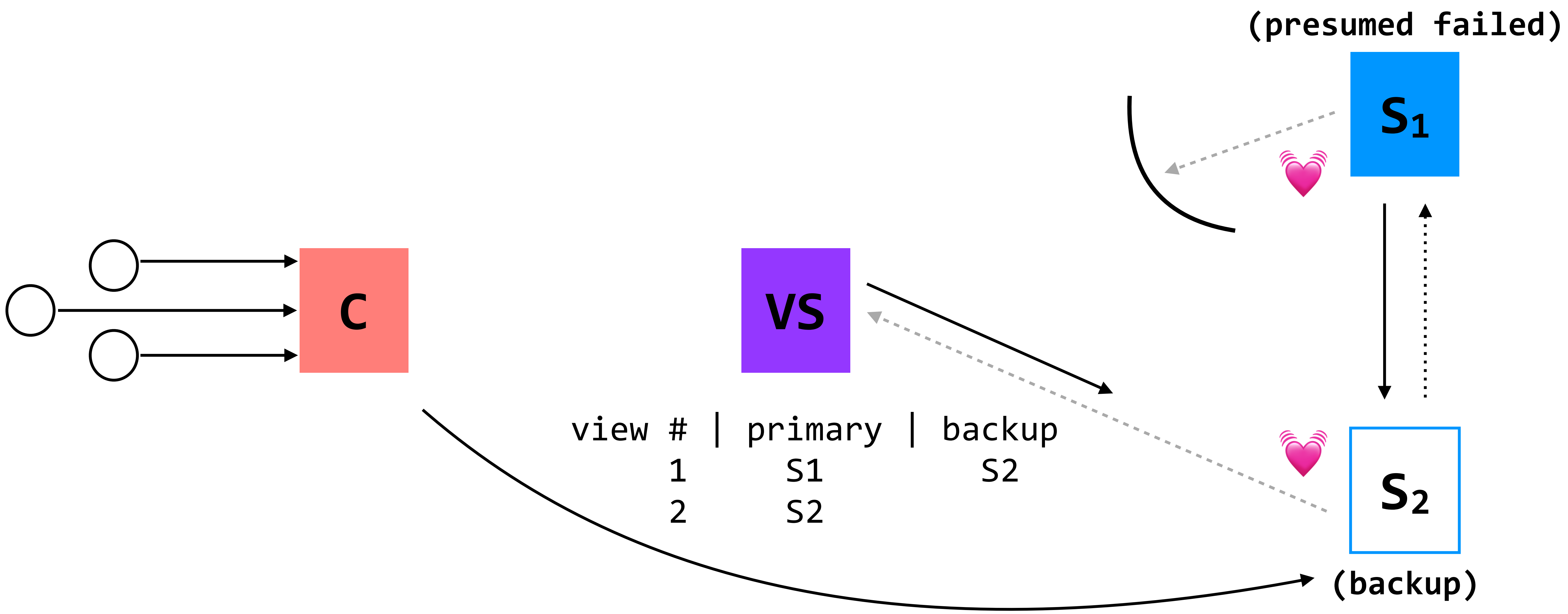
if **S2** receives any requests from **C**, it will reject them; it believes that it is the backup (and so does not communicate directly with **C**)

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



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

if S_1 receives any requests from C, it will behave as primary with S_2 as backup

if S_2 receives any requests from C, it will reject them; it believes that it is the backup (and so does not communicate directly with C)

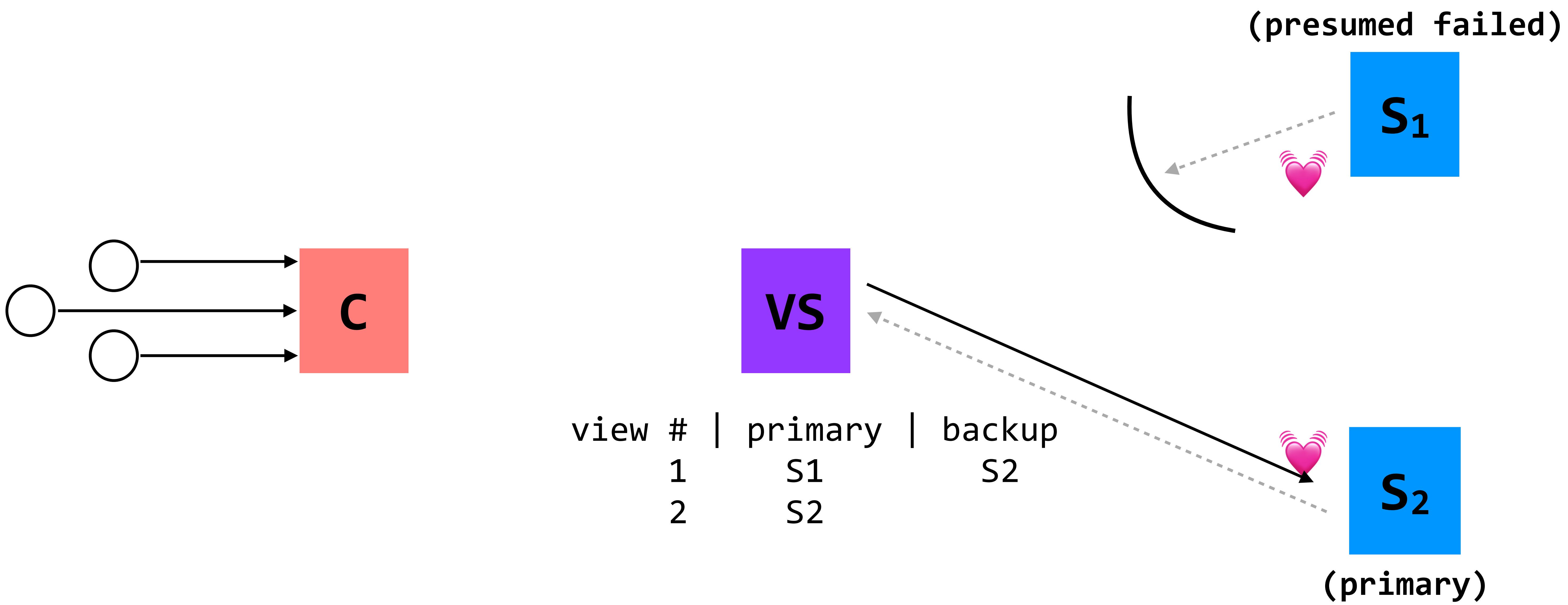
new detail: backups **reject** any requests from coordinators

what happens if a network partition prevents 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

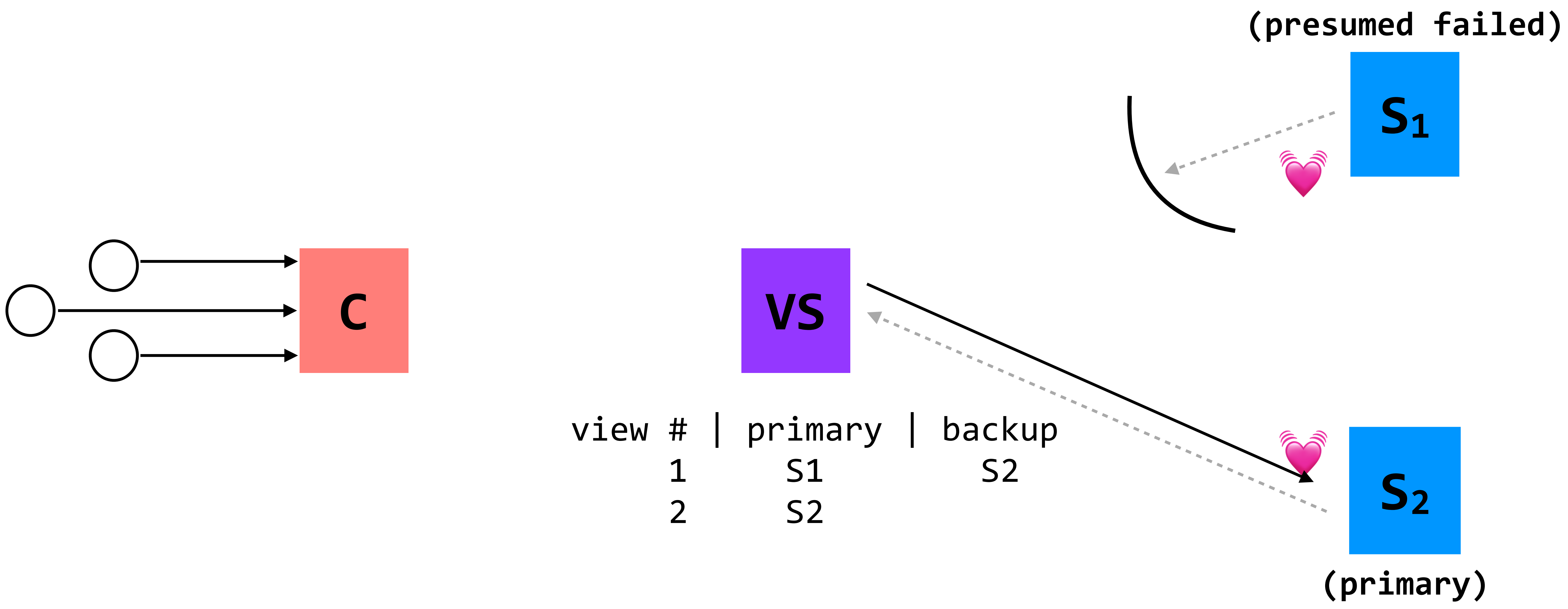


what happens if a network partition prevents S_1 from communicating with VS?

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



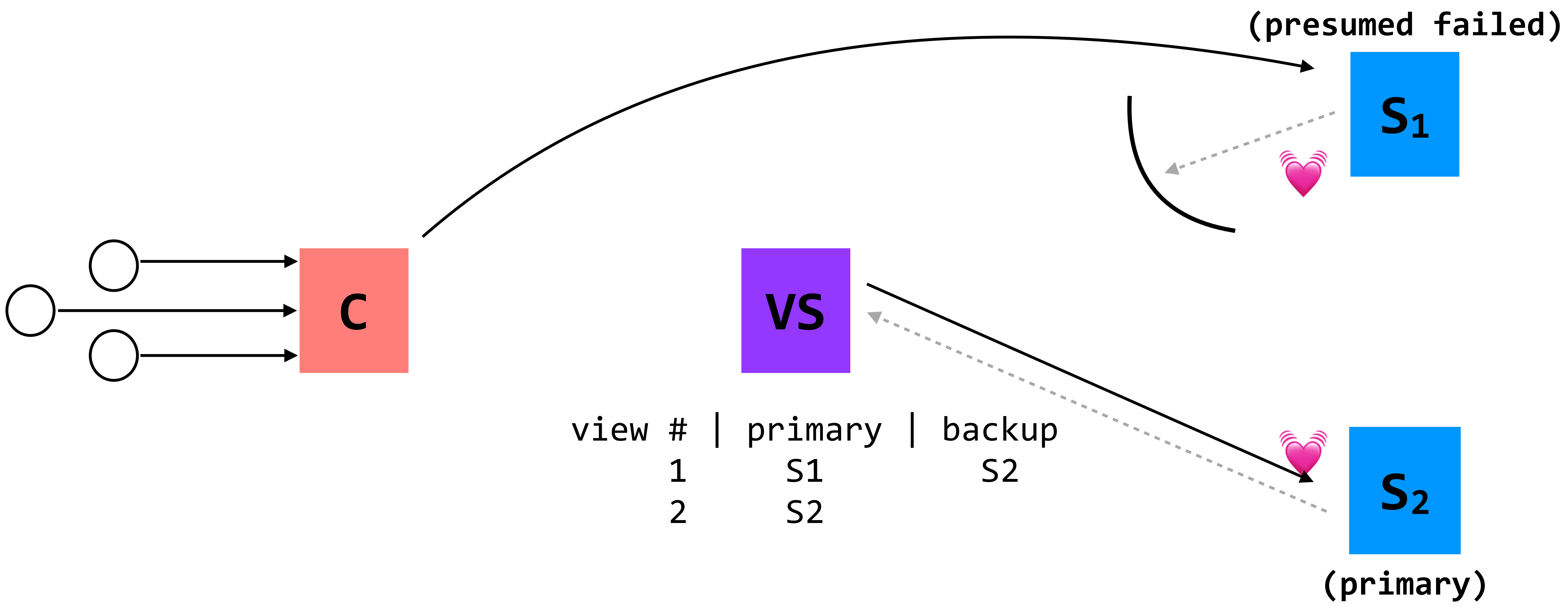
at this stage, **VS** and **S_2** think **S_2** is primary; **S_1** thinks **S_1** is primary

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**

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



at this stage, **VS** and **S_2** think **S_2** is primary; **S_1** thinks **S_1** is primary

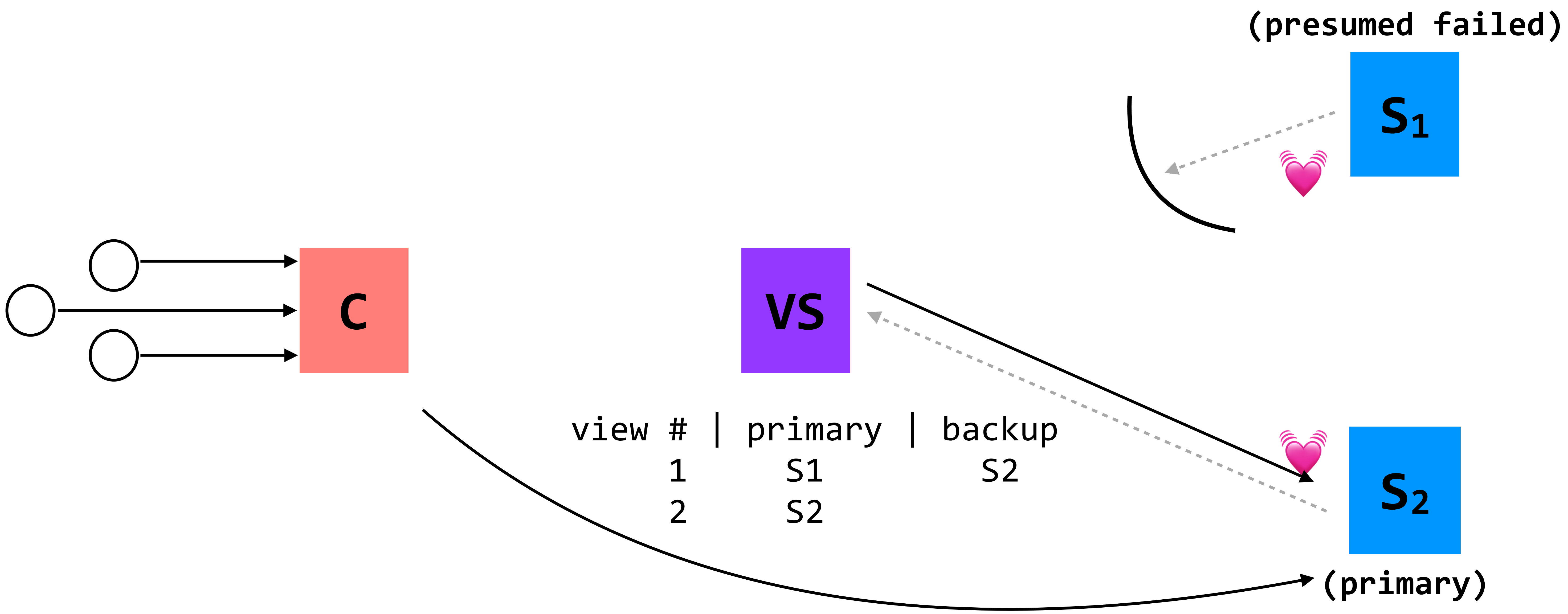
if **S_1** receives any requests from **C**, it won't be able to get an ACK from **S_2**, and so will reject

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



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

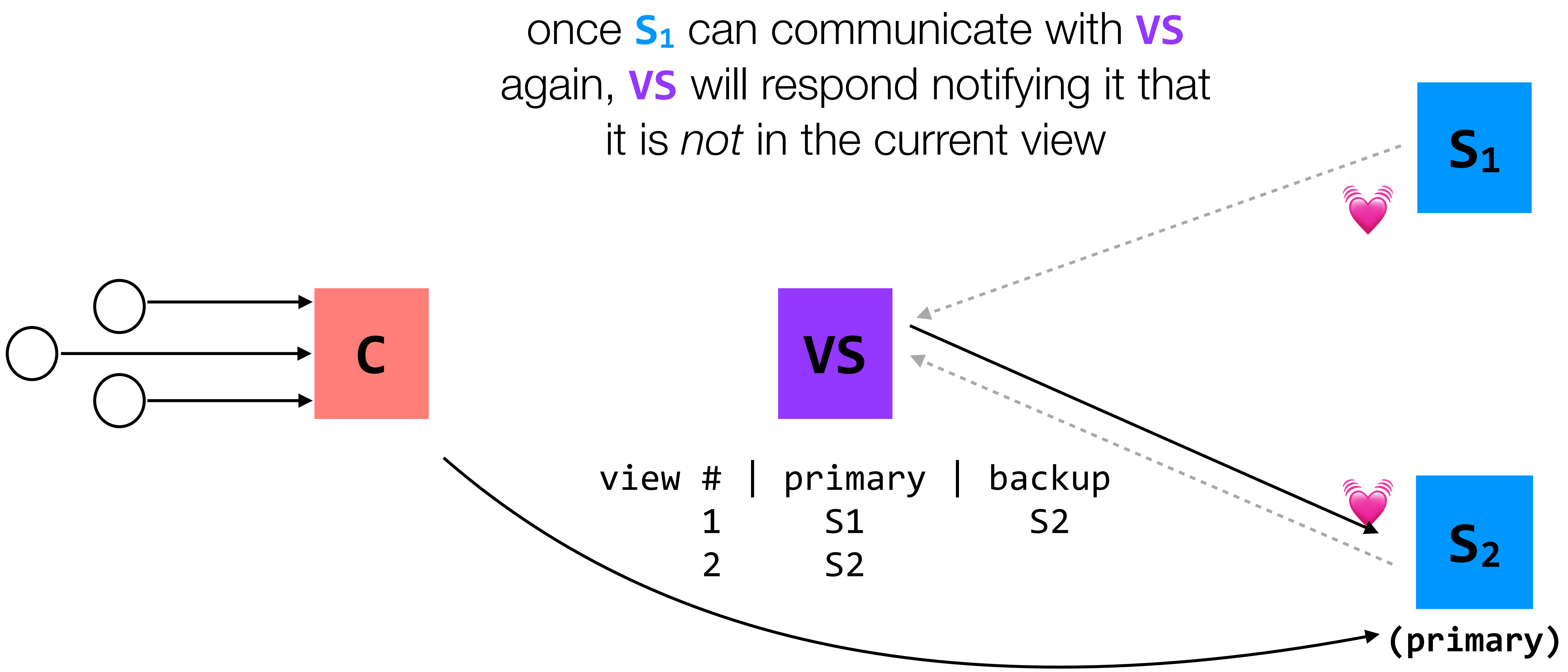
if **S_2** receives any requests from **C**, it will respond as the primary (in line with what **VS** expects)

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



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?

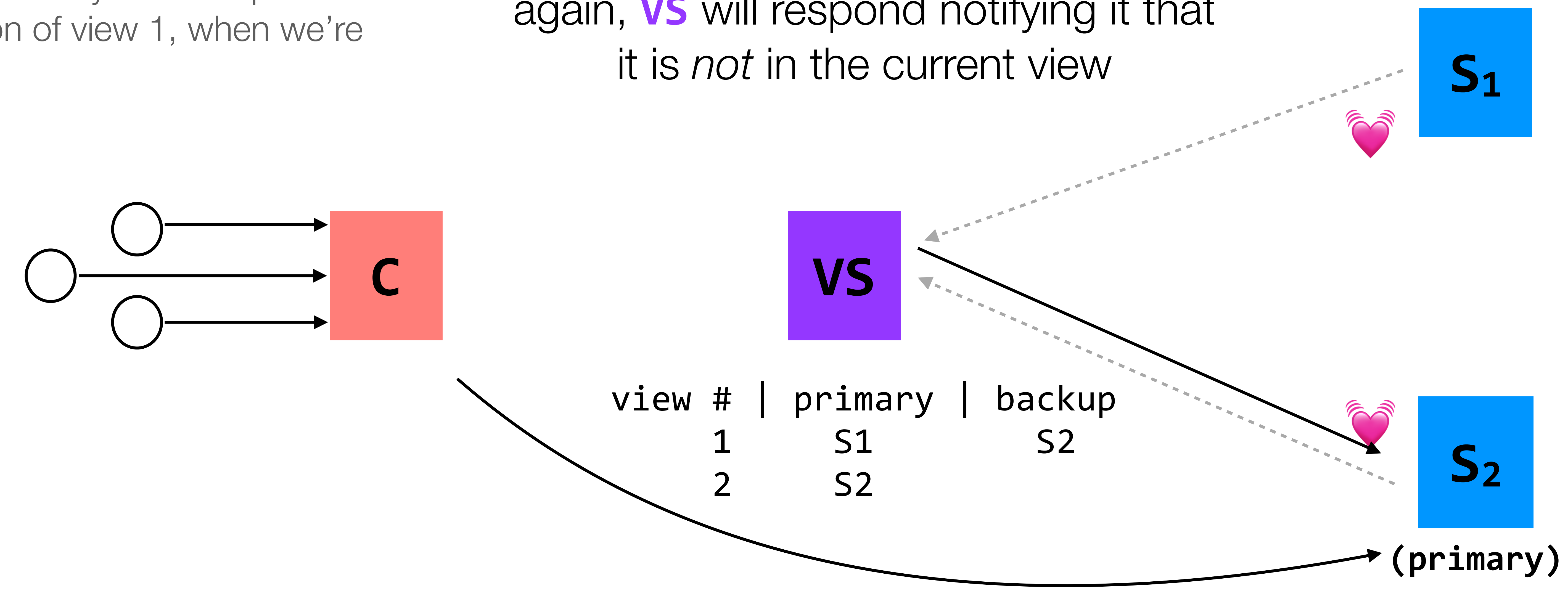
in a sense, this is the worst possible partition: VS is going to presume S_1 has failed (and so switch to using S_2 as a backup), while S_1 can still communicate with everyone *except* VS

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

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)

once S_1 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?

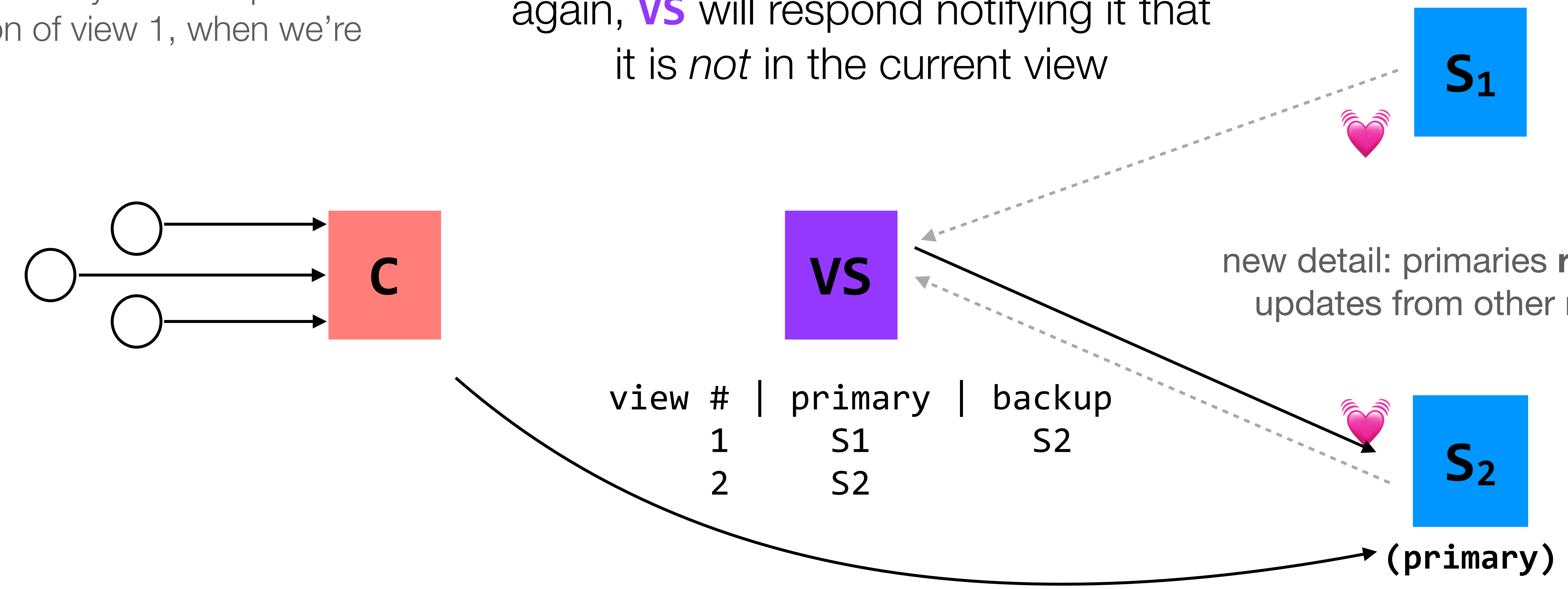
in a sense, this is the worst possible partition: **VS** is going to presume S_1 has failed (and so switch to using S_2 as a backup), while S_1 can still communicate with everyone *except* **VS**

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

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)

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




```
view # | primary | backup
   1      S1        S2
   2      S2
```

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

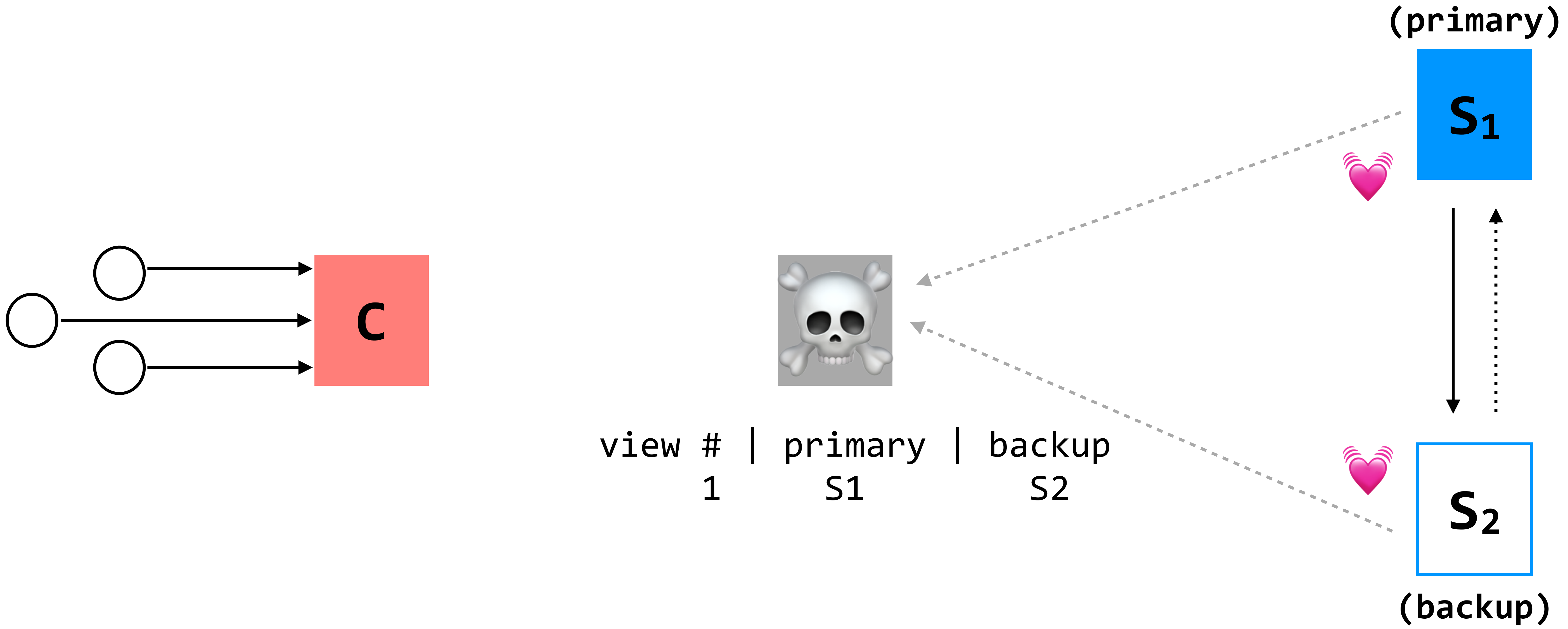
if S_2 receives any requests from **C**, it will respond as the primary (in line with what **VS** expects)

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**

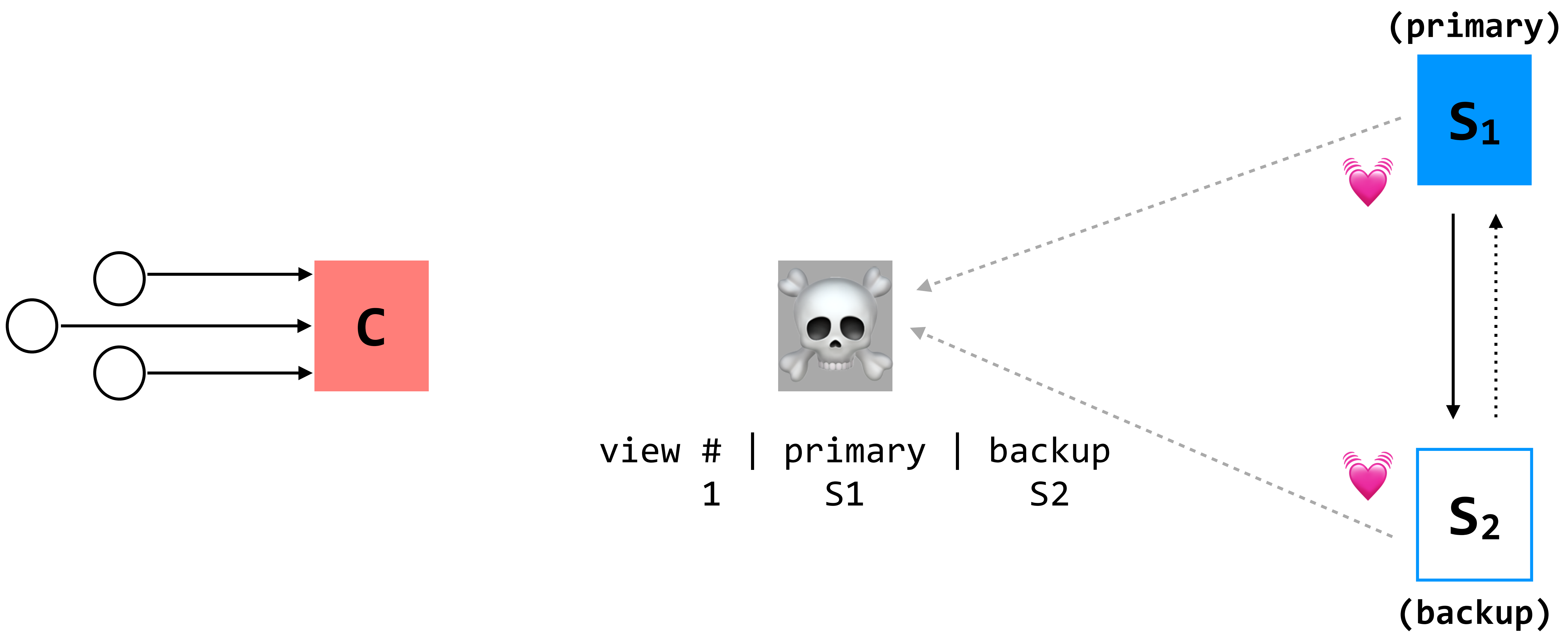
to increase availability, let's try replicating data on two servers

attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



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

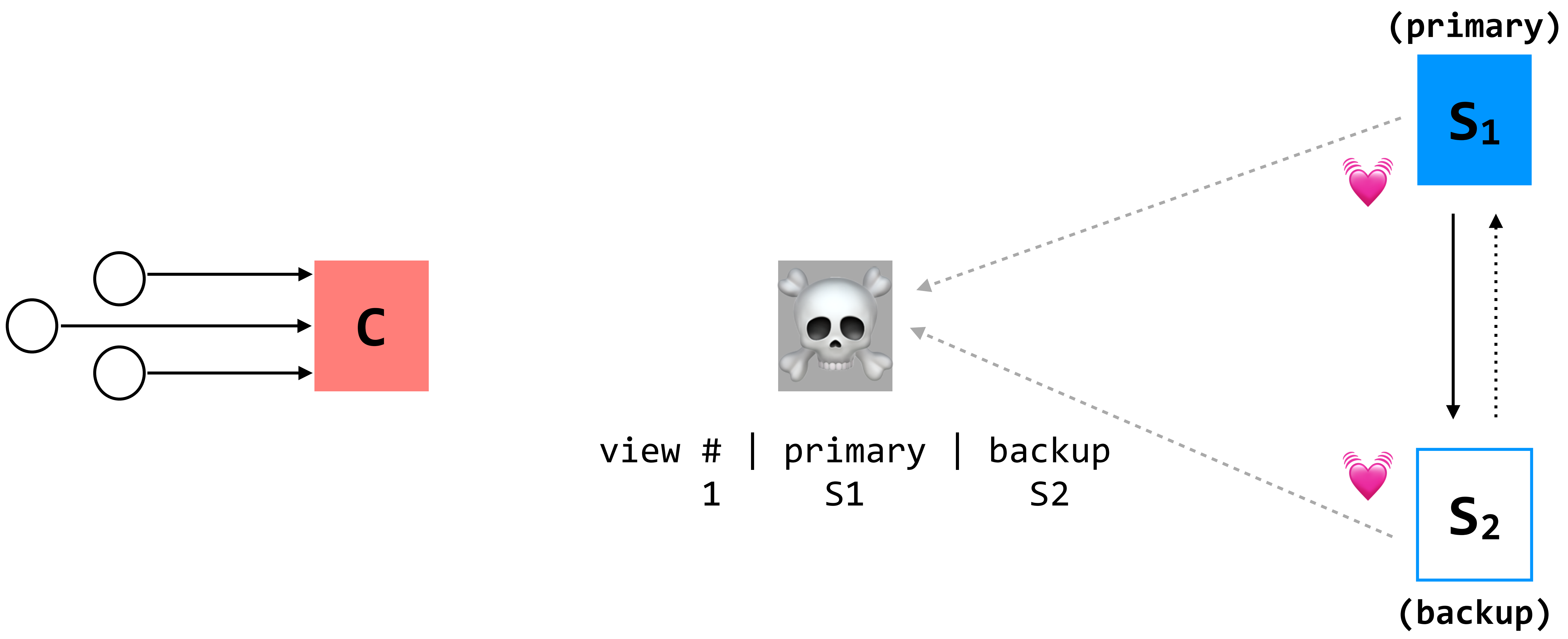
attempt 3: use a **view server** to determine which replica is primary, in hopes that we can deal with network partitions



what happens if VS 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



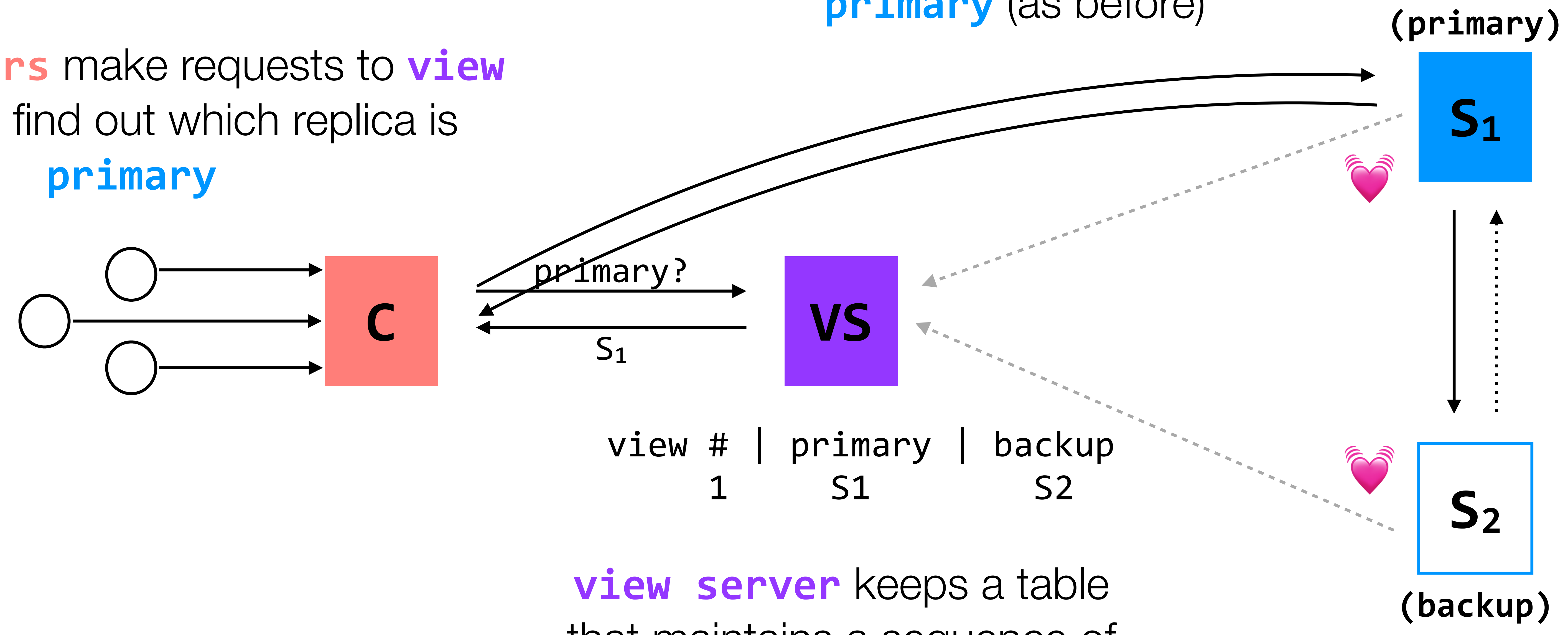
what happens if VS fails?

find out in Tuesday's recitation

replicated state machines

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

coordinators contact primary (as before)



view #	primary	backup
1	S1	S2

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

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

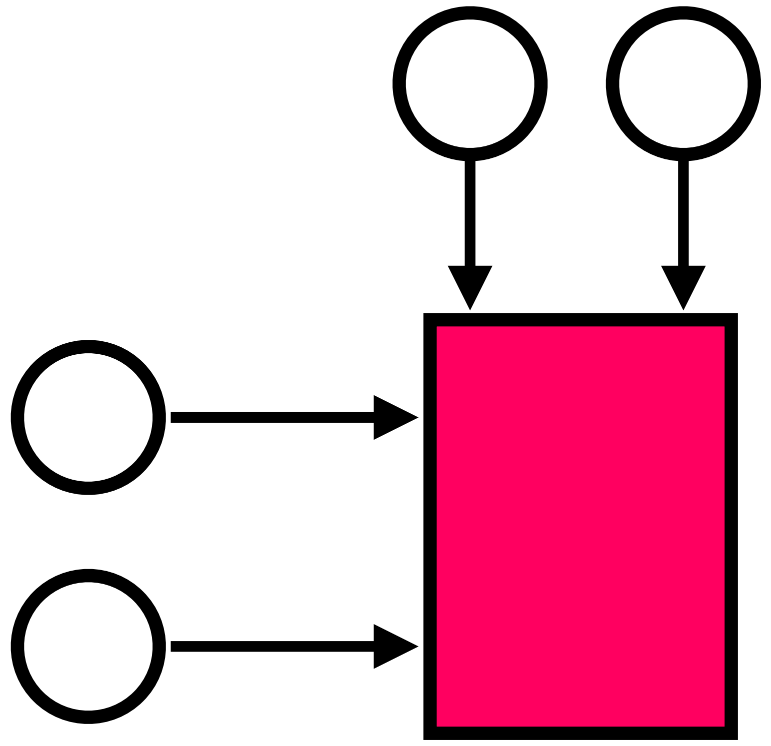
view server keeps a table that maintains a sequence of views

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)

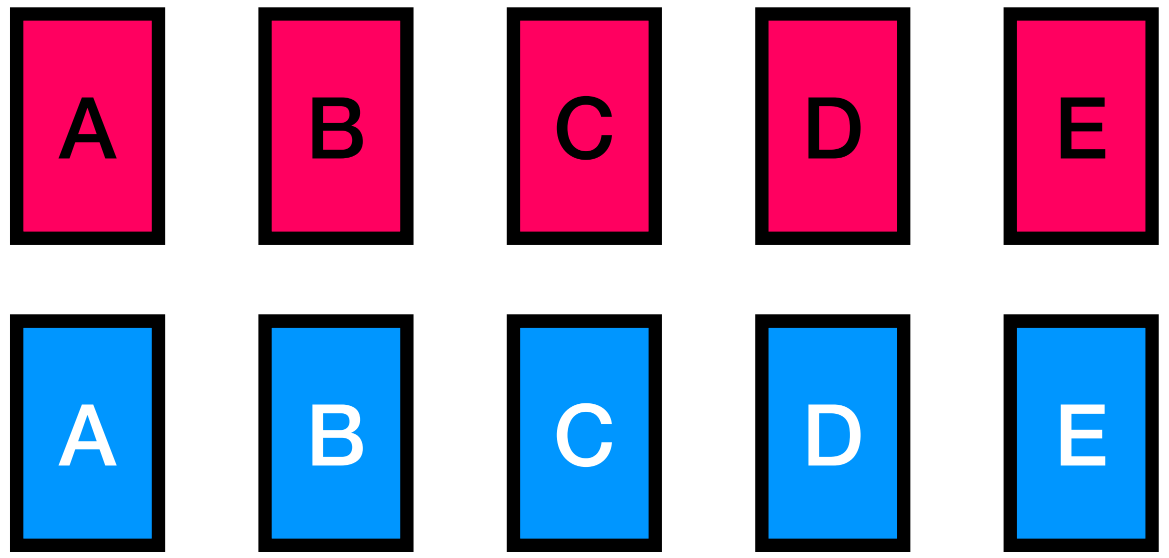
primary **must** get an ACK from its backups before completing the update

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



replicated state machines give us single-copy consistency even with replicated data



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