6.033 Spring 2019

Lecture #19

- Distributed transactions
  - Availability
  - Replicated State Machines

goal: build reliable systems from unreliable components

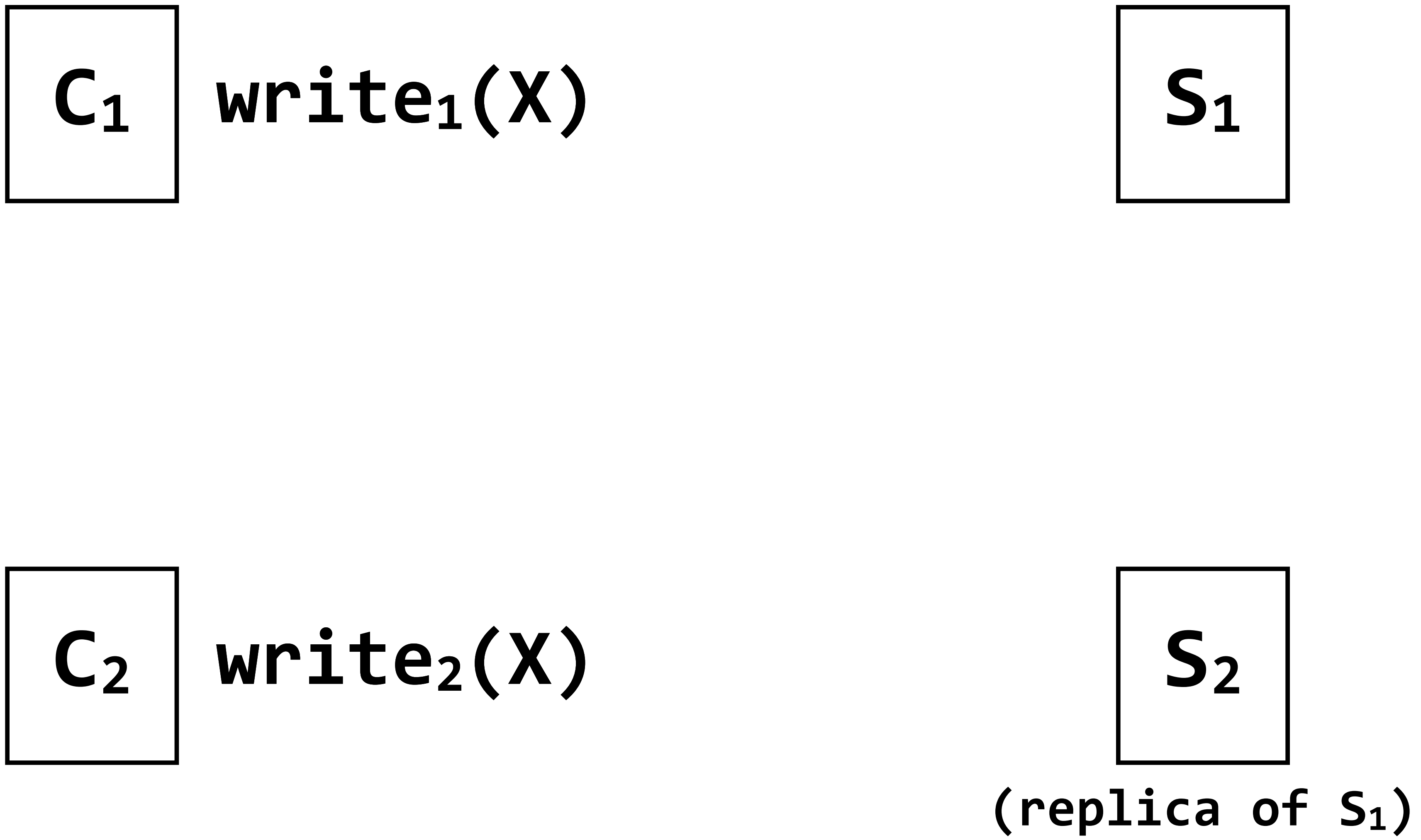
the abstraction that makes that easier is

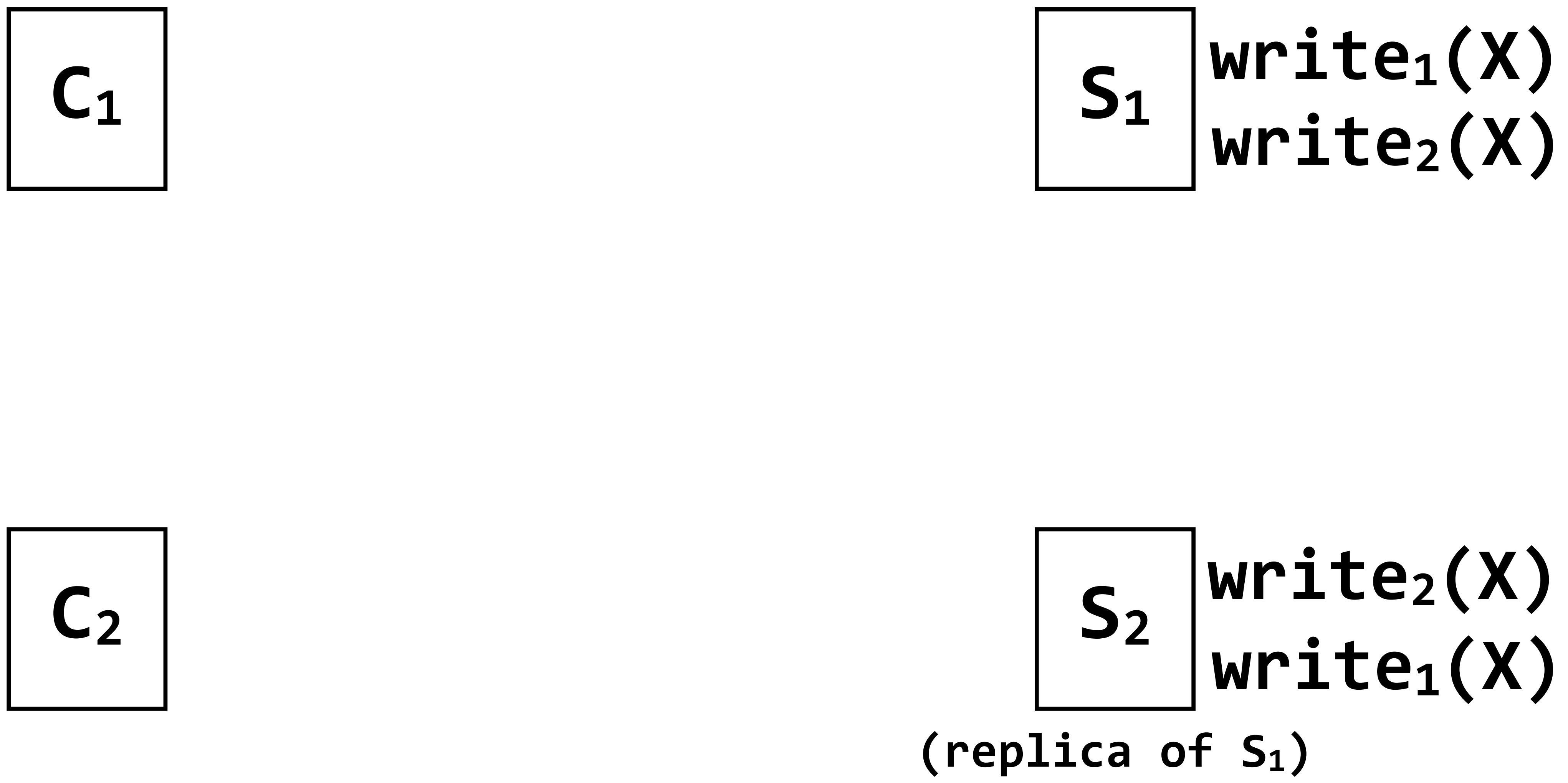
transactions, which provide **atomicity** and **isolation**, while not hindering **performance**

atomicity $\longrightarrow$ **shadow copies** (simple, poor performance) or **logs** (better performance, a bit more complex)

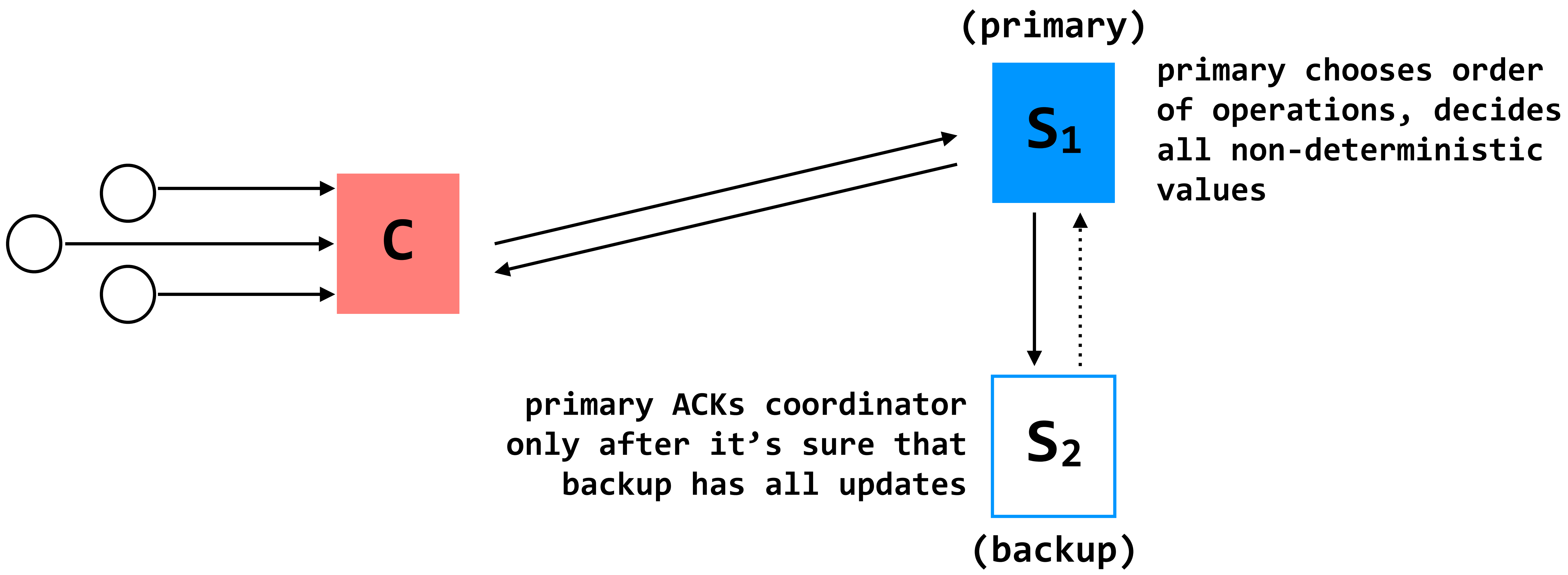
isolation $\longrightarrow$ **two-phase locking**

we also want transaction-based systems to be **distributed** — to run across multiple machines — and to remain **available** even through failures


C_1
write$_1$(X)
S_1
C_2
write$_2$(X)
S_2
(replica of S_1)



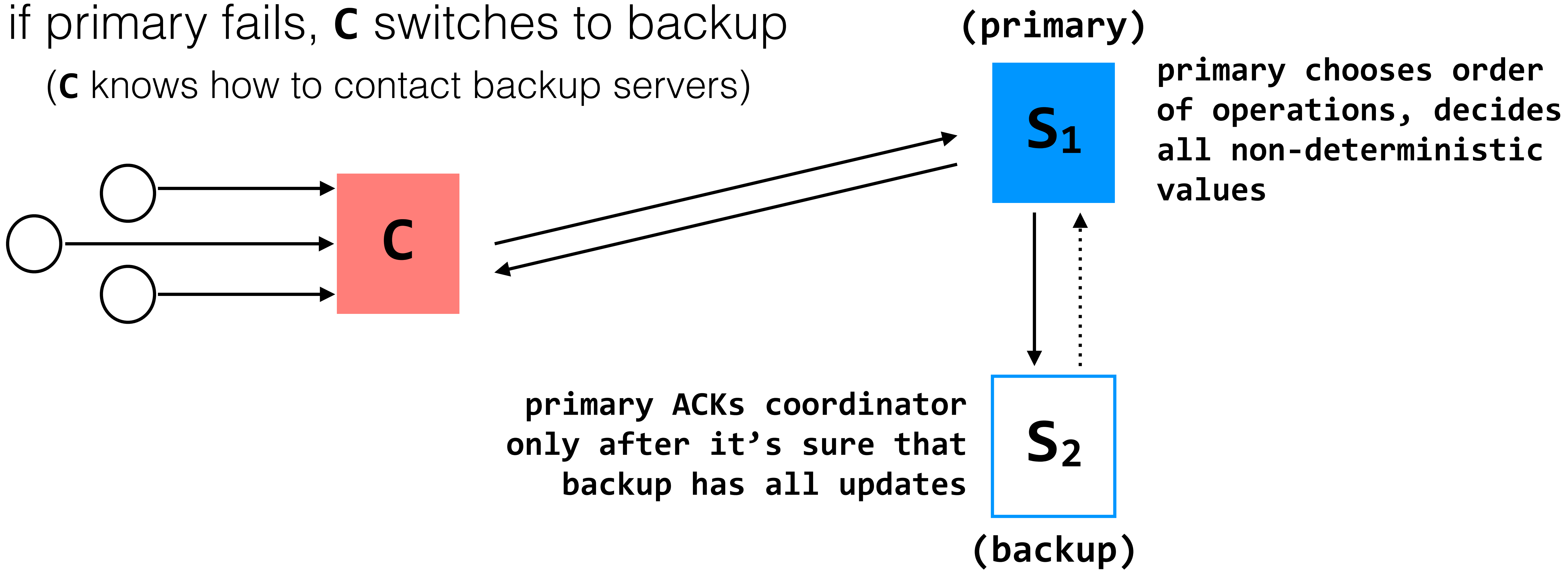
problem: replica servers can become inconsistent


(primary)
S1
primary chooses order of operations, decides all non-deterministic values
C
S2
primary ACKs coordinator only after it's sure that backup has all updates
(backup)

attempt: coordinators communicate with primary servers, who communicate with backup servers

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

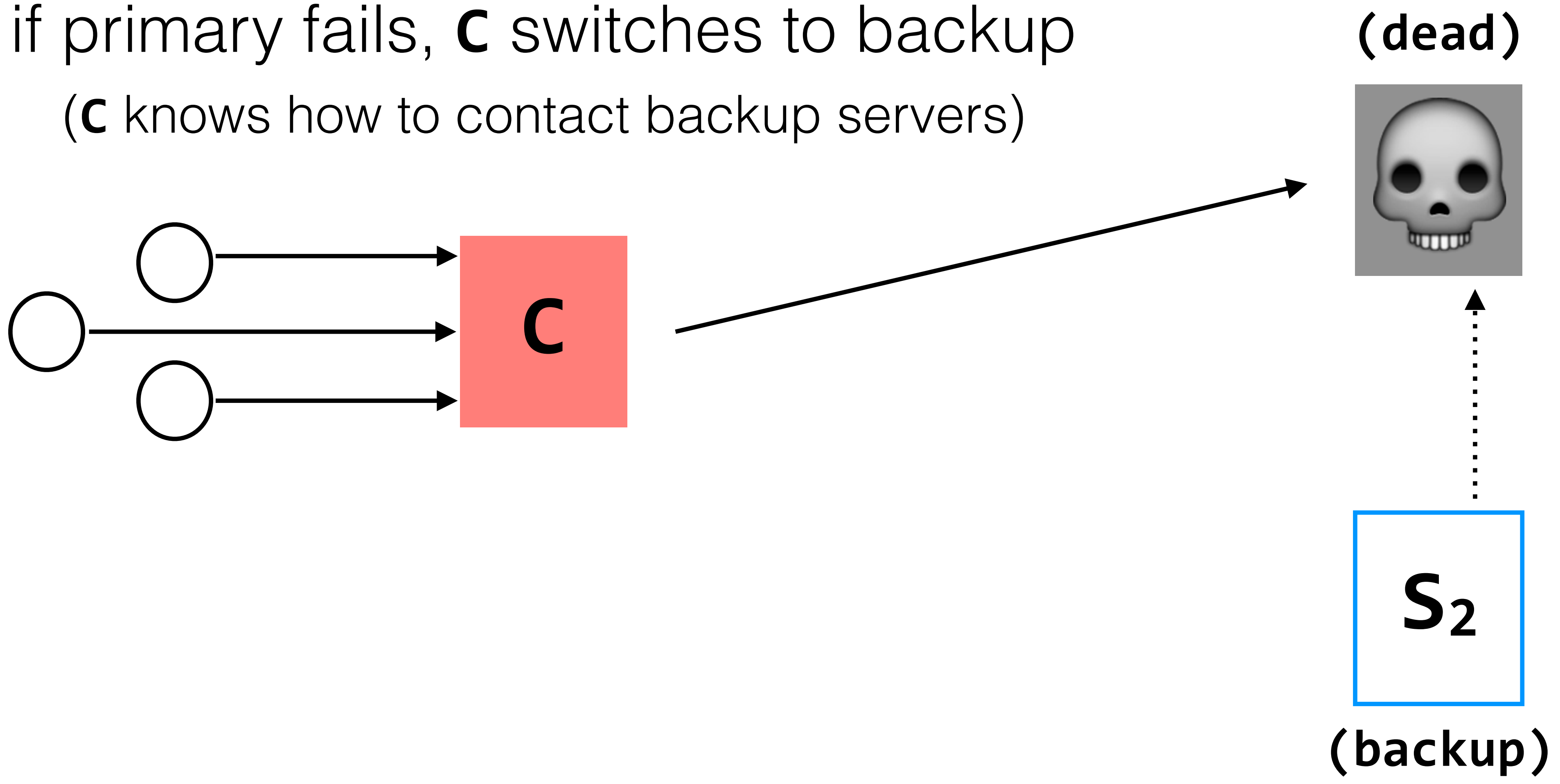
(**C** knows how to contact backup servers)


(primary)
S1
primary chooses order of operations, decides all non-deterministic values
C
primary ACKs coordinator only after it's sure that backup has all updates
S2
(backup)

attempt: coordinators communicate with primary servers, who communicate with backup servers

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

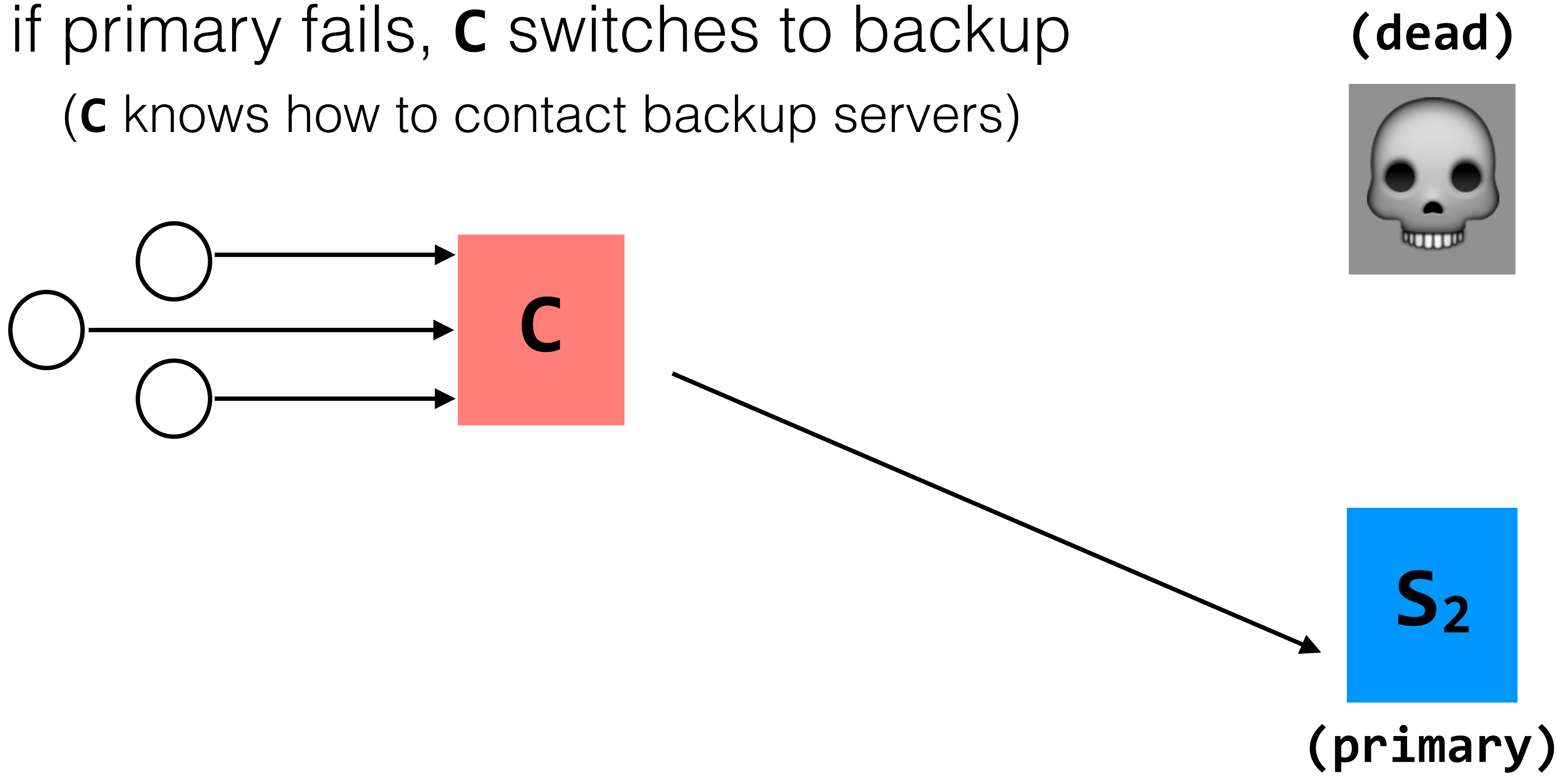
(**C** knows how to contact backup servers)



attempt: coordinators communicate with primary servers, who communicate with backup servers

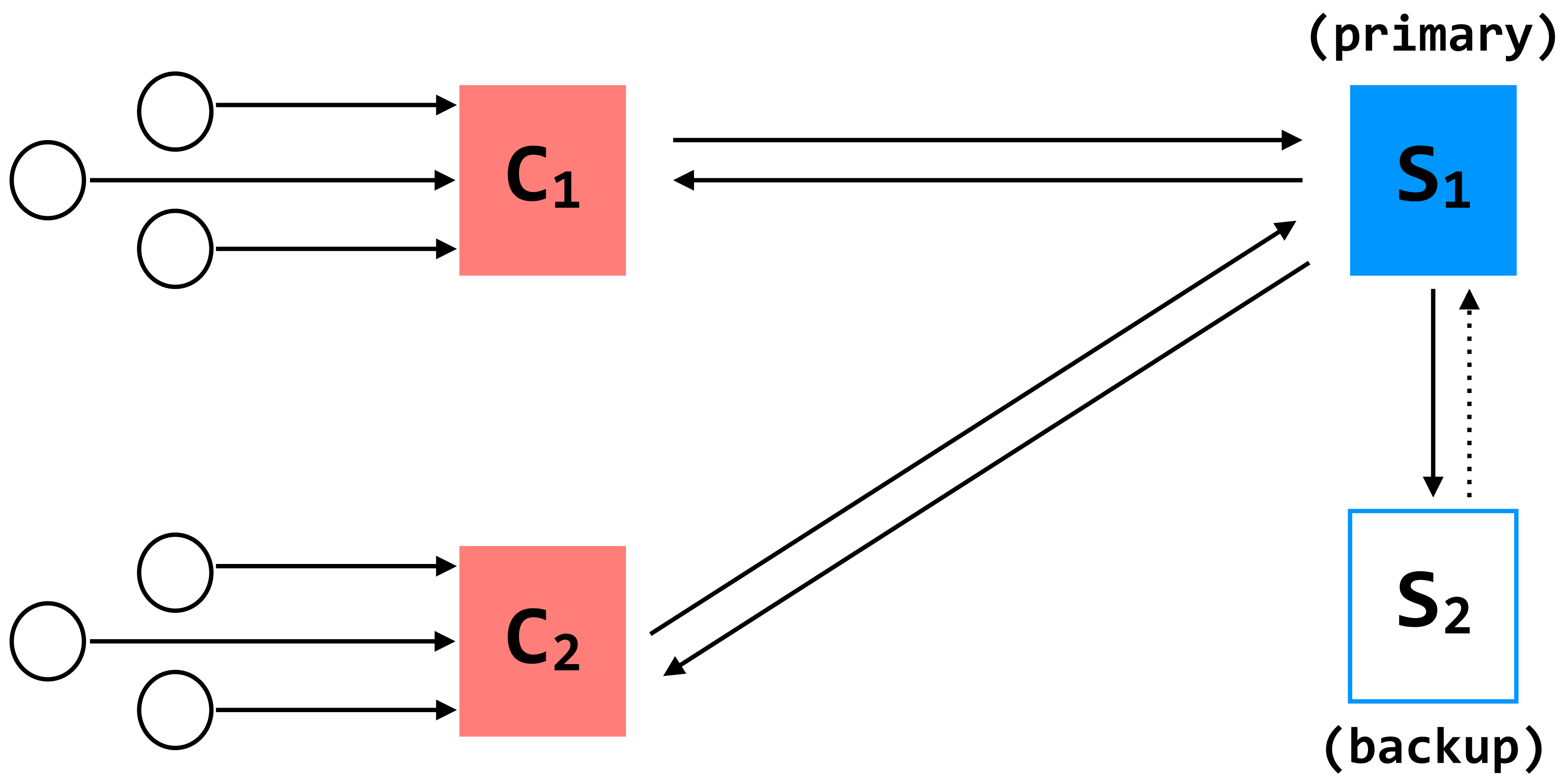
if primary fails, **C** switches to backup

(**C** knows how to contact backup servers)



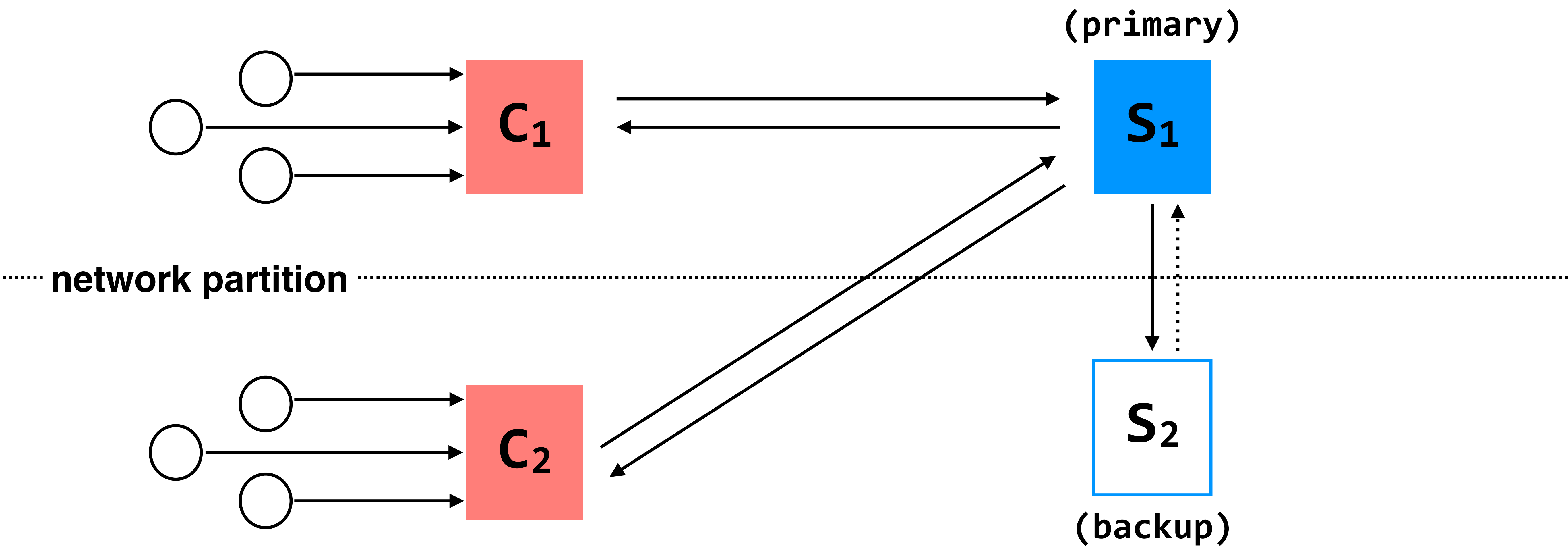
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems



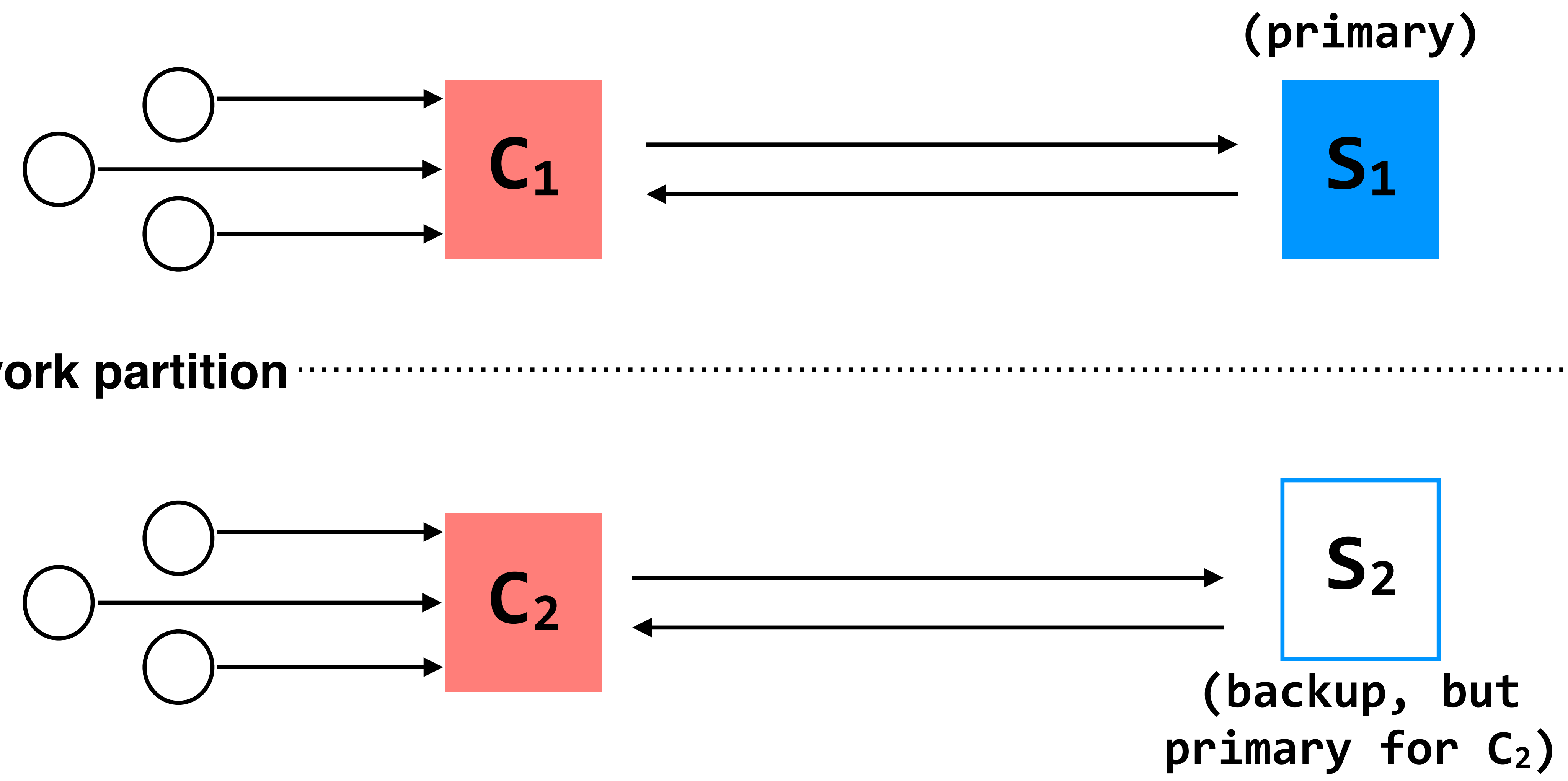
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems



attempt: coordinators communicate with primary servers, who communicate with backup servers

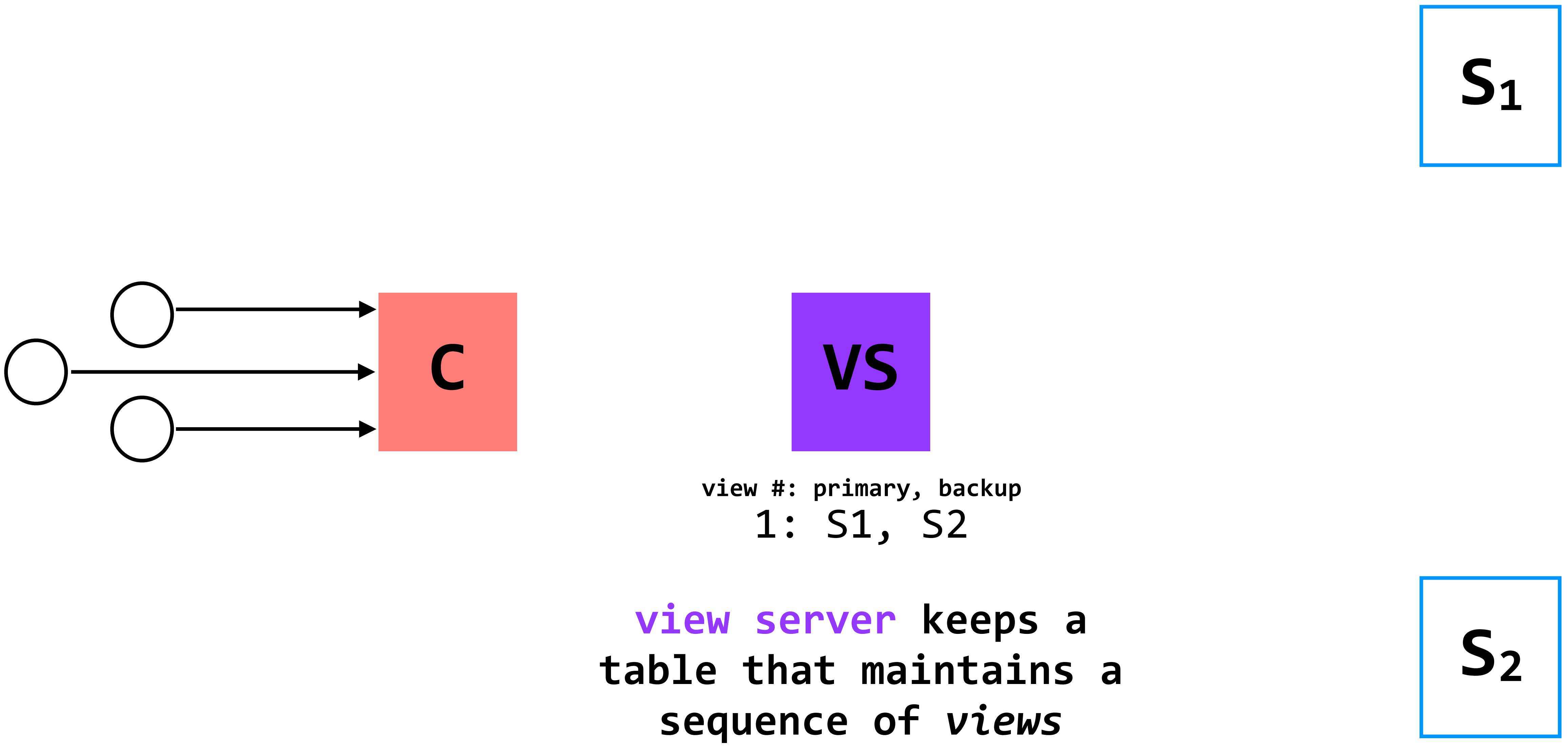
multiple coordinators + the network = problems



C_1 and **C_2** are using different primaries;

S_1 and **S_2** are no longer consistent

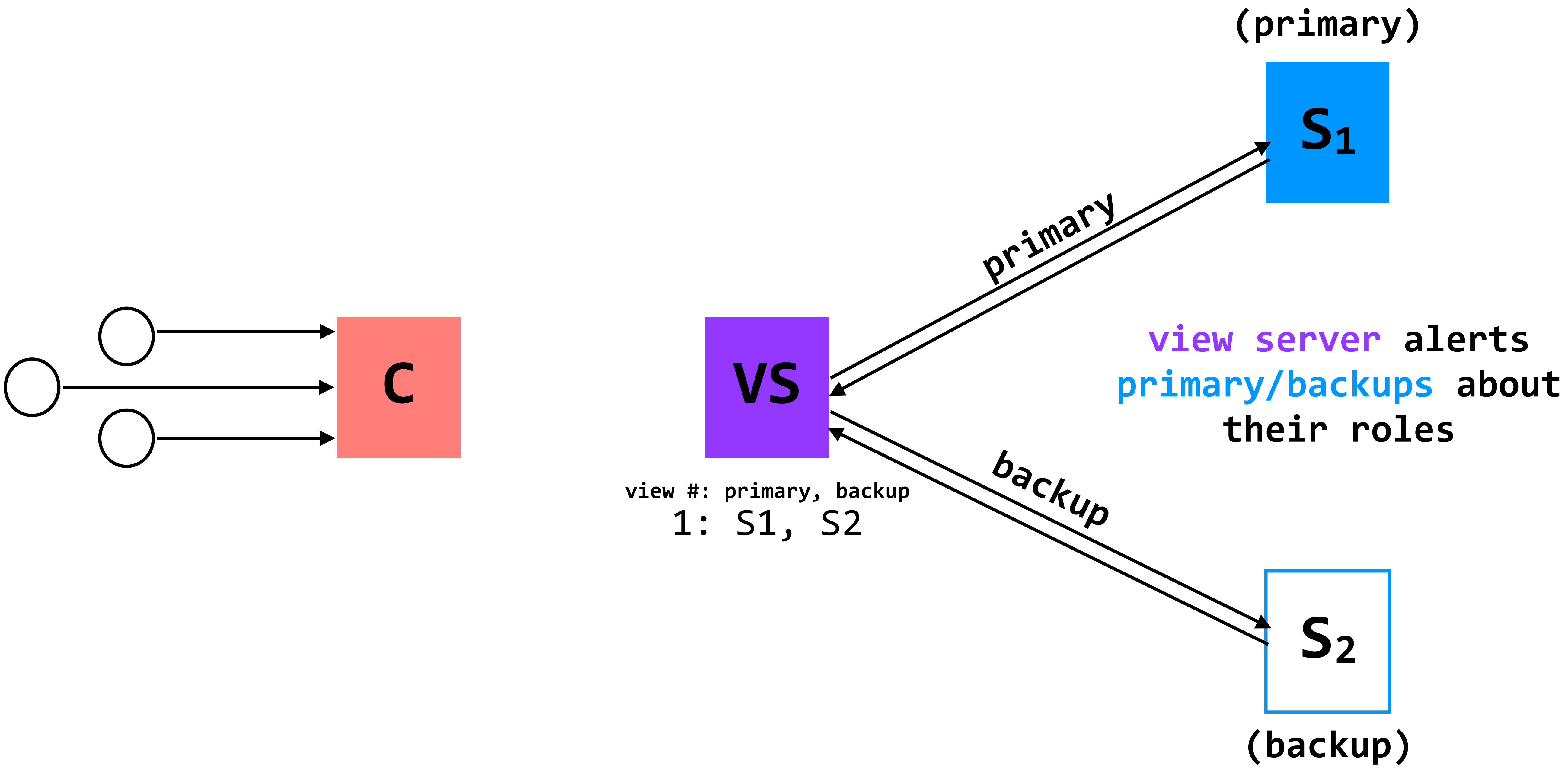
attempt: coordinators communicate with primary servers, who communicate with backup servers



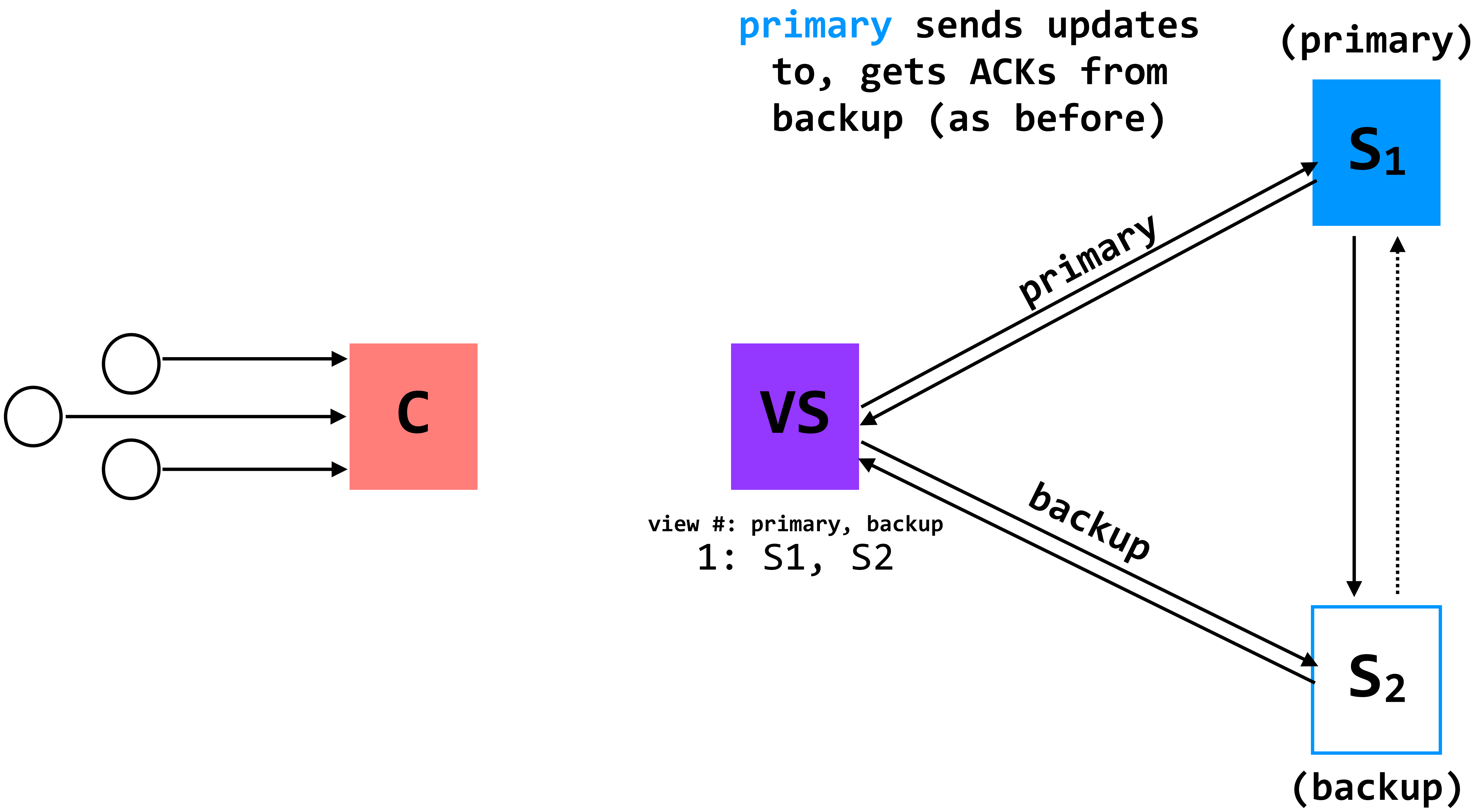
view #: primary, backup
1: S1, S2

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

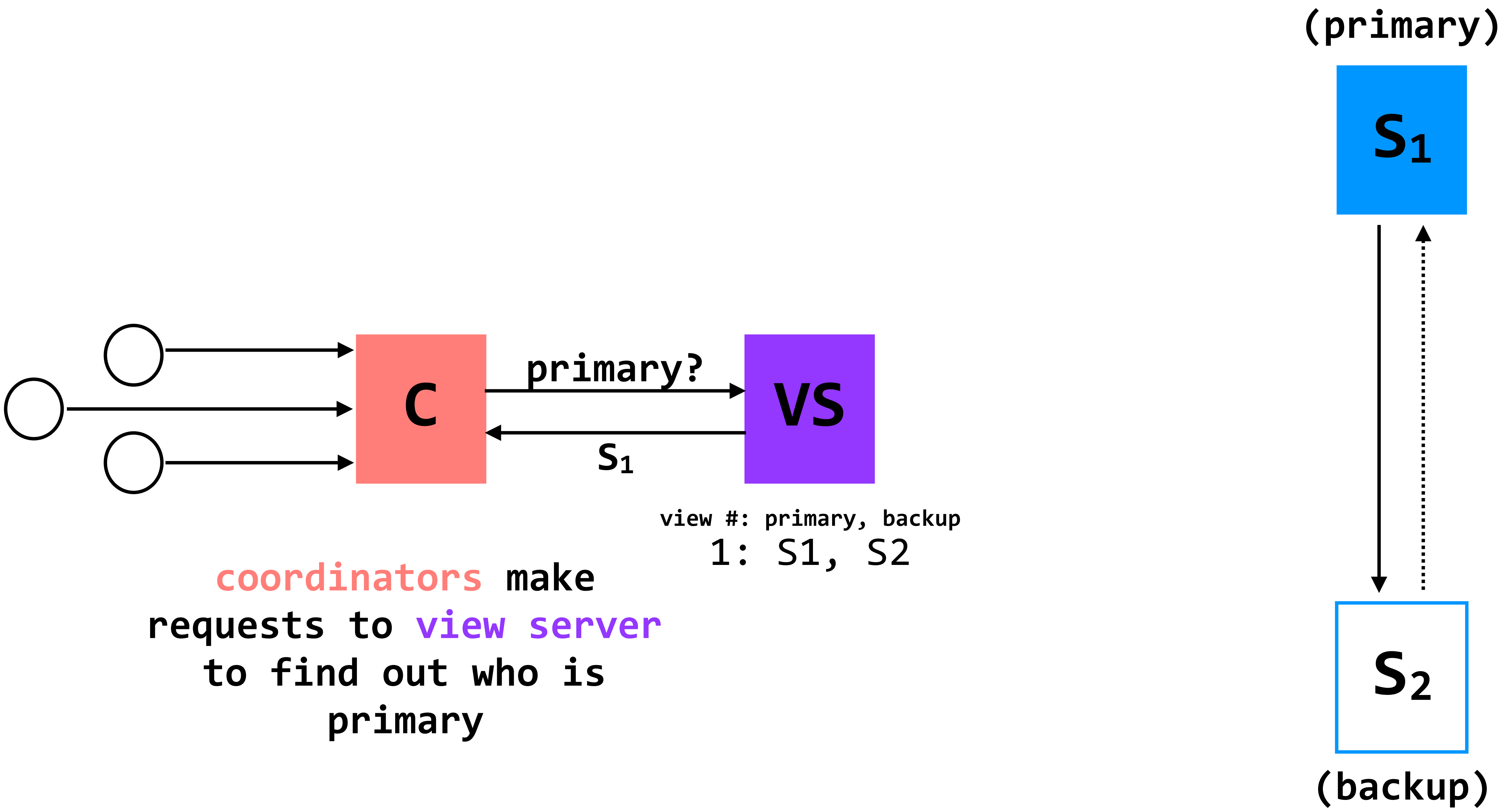
use a **view server**, which determines which replica is the primary



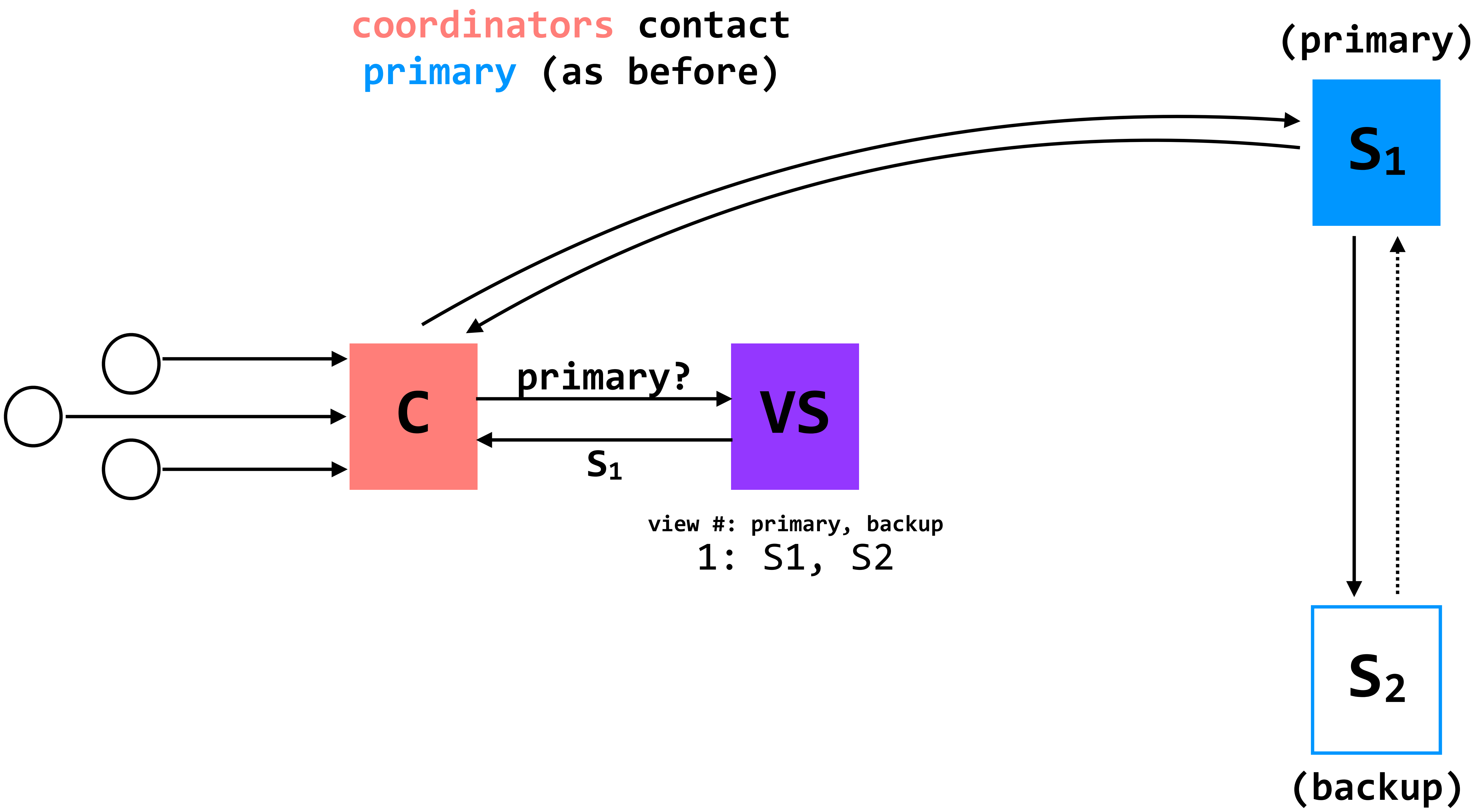
use a **view server**, which determines which replica is the primary



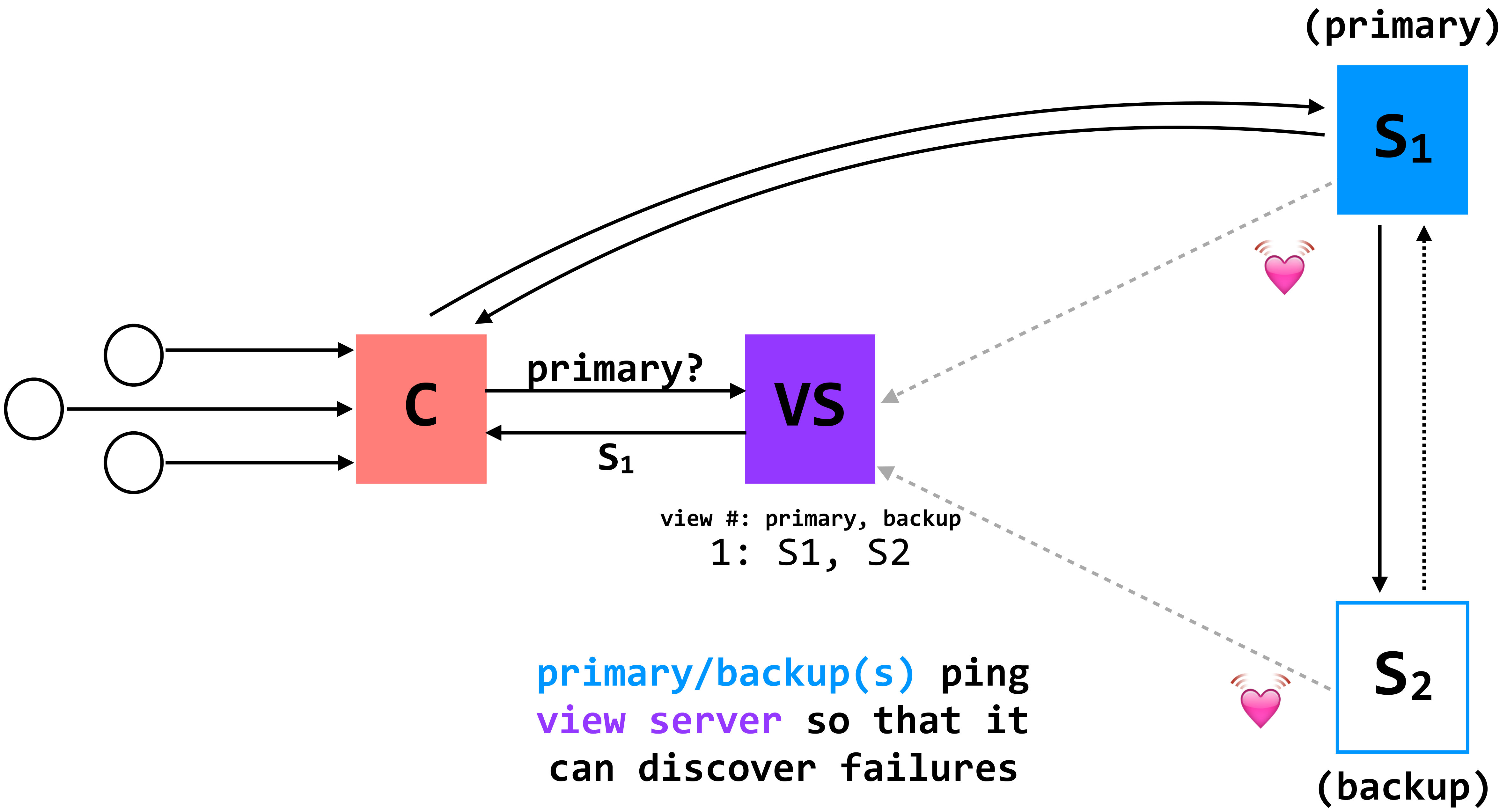
use a **view server**, which determines which replica is the primary



use a **view server**, which determines which replica is the primary

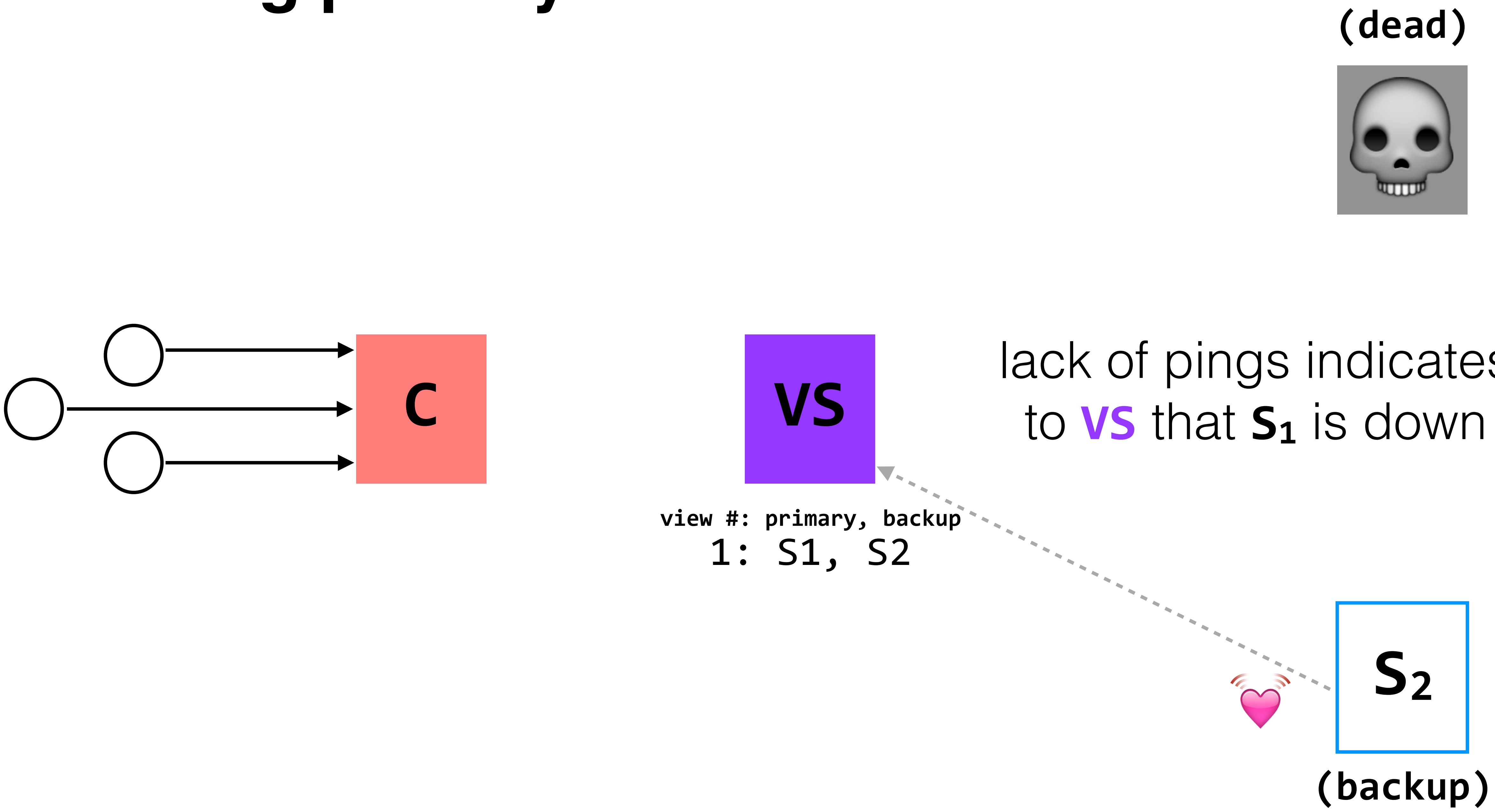


use a **view server**, which determines which replica is the primary



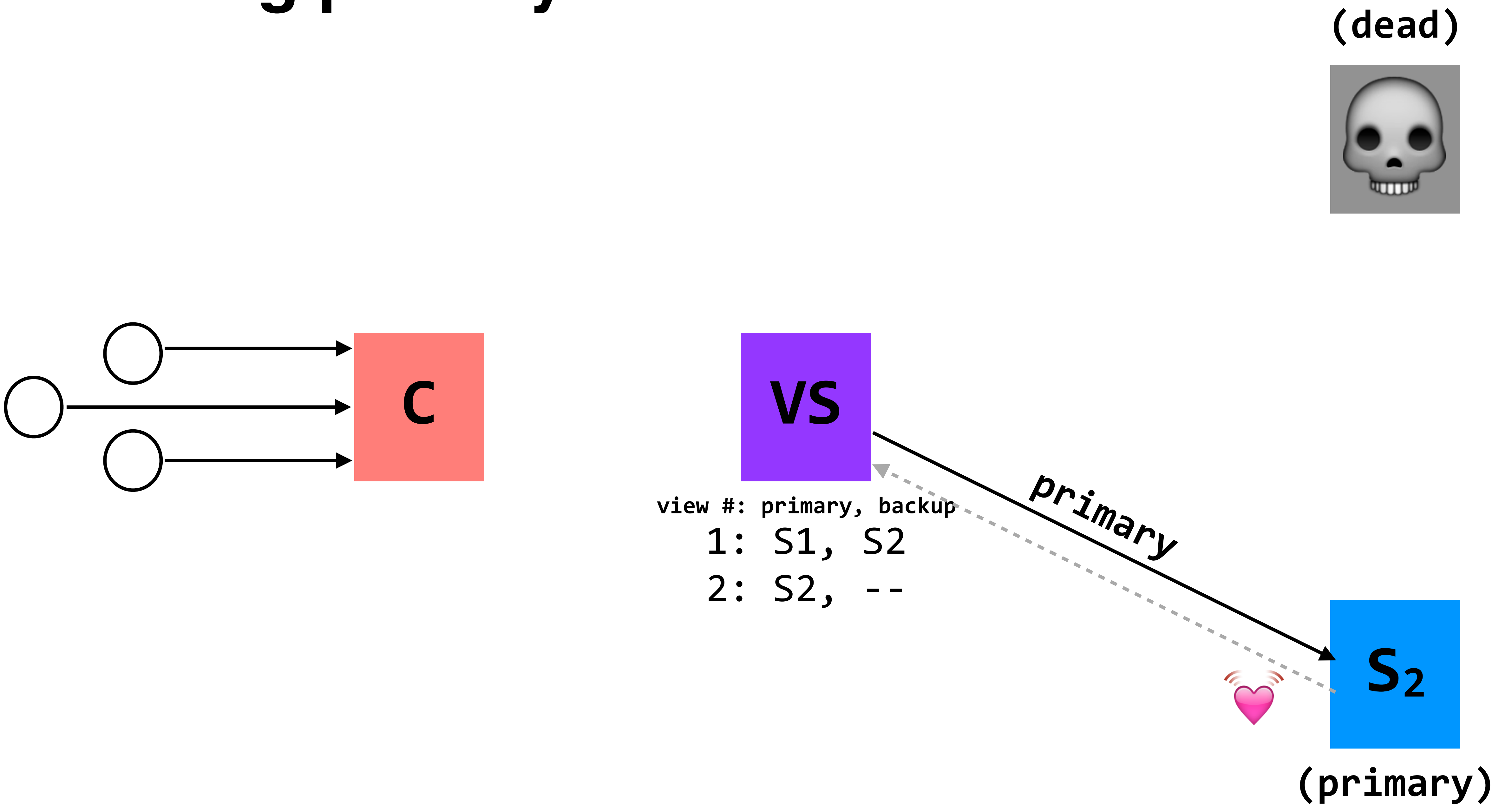
use a **view server**, which determines which replica is the primary

handling primary failure

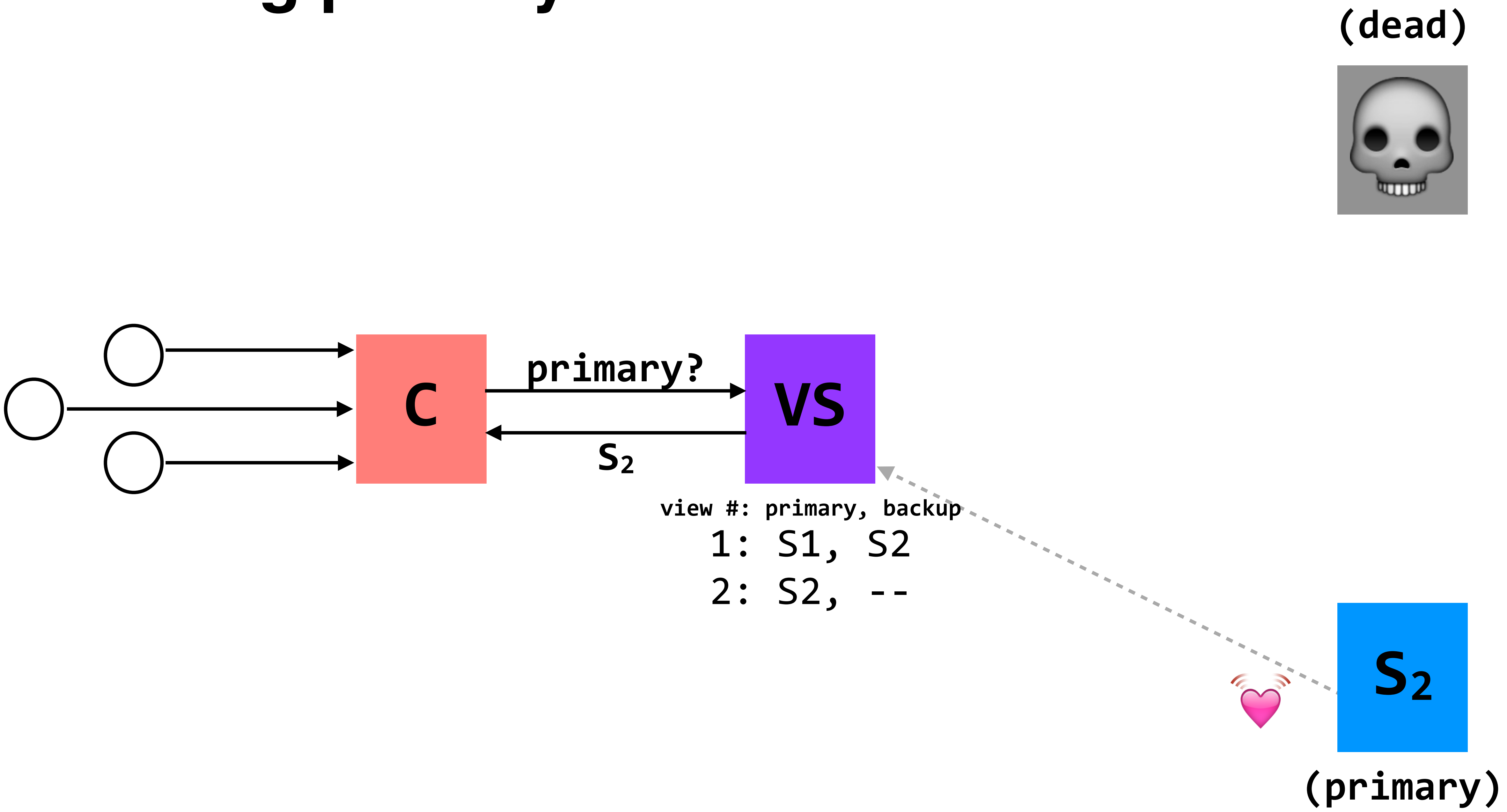


lack of pings indicates to **VS** that $\mathbf{S_1}$ is down

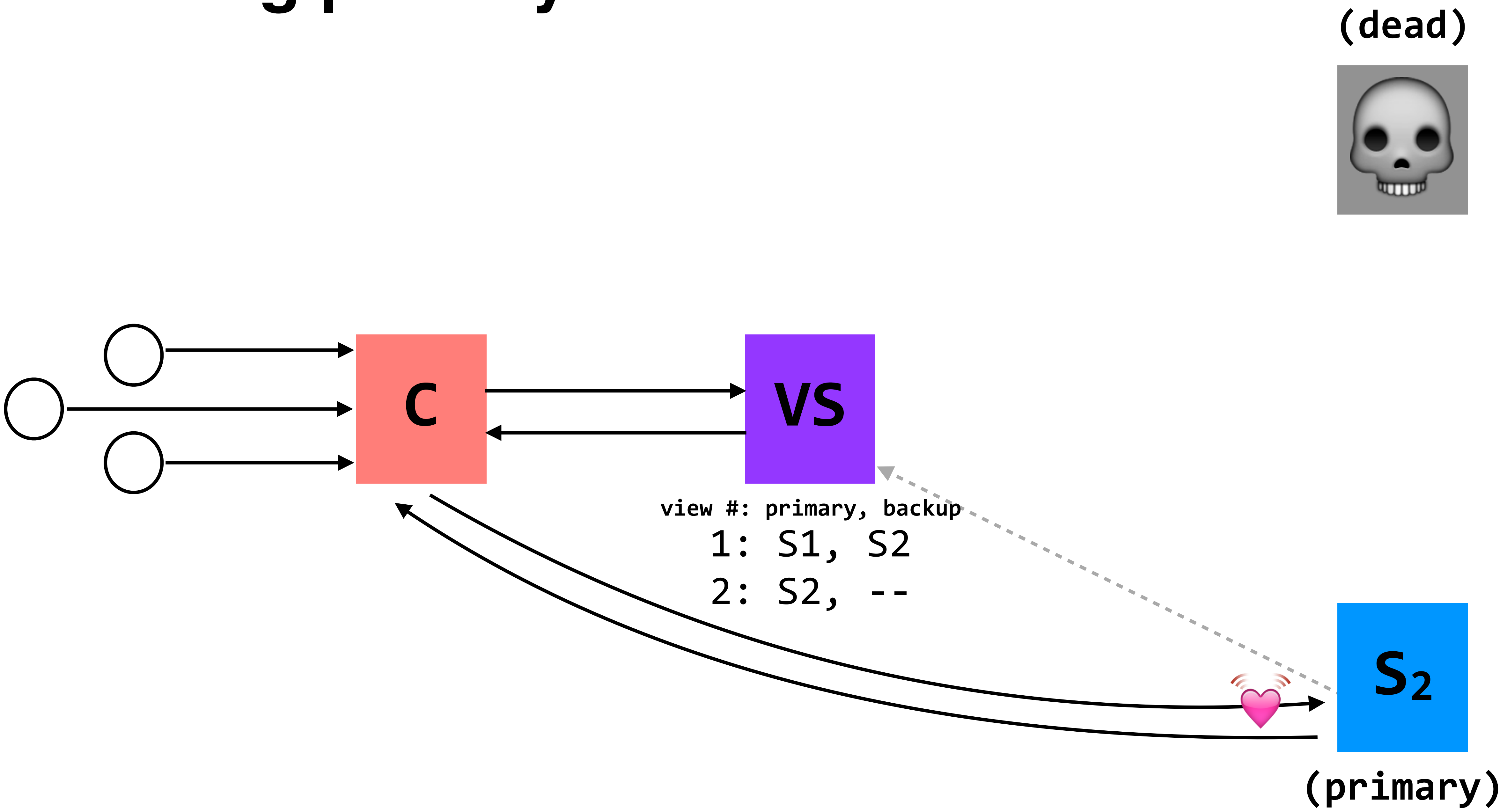
handling primary failure



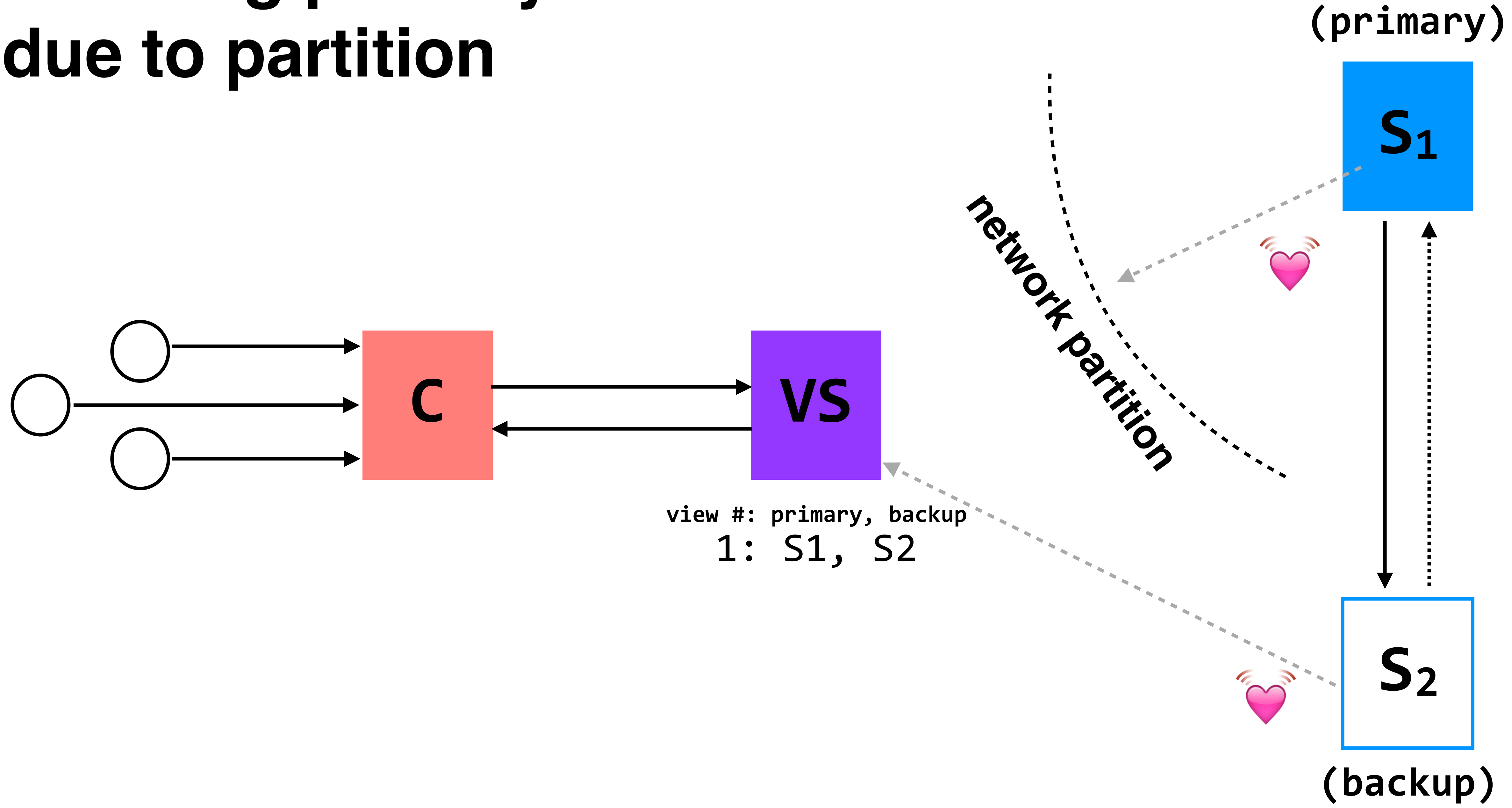
handling primary failure



handling primary failure

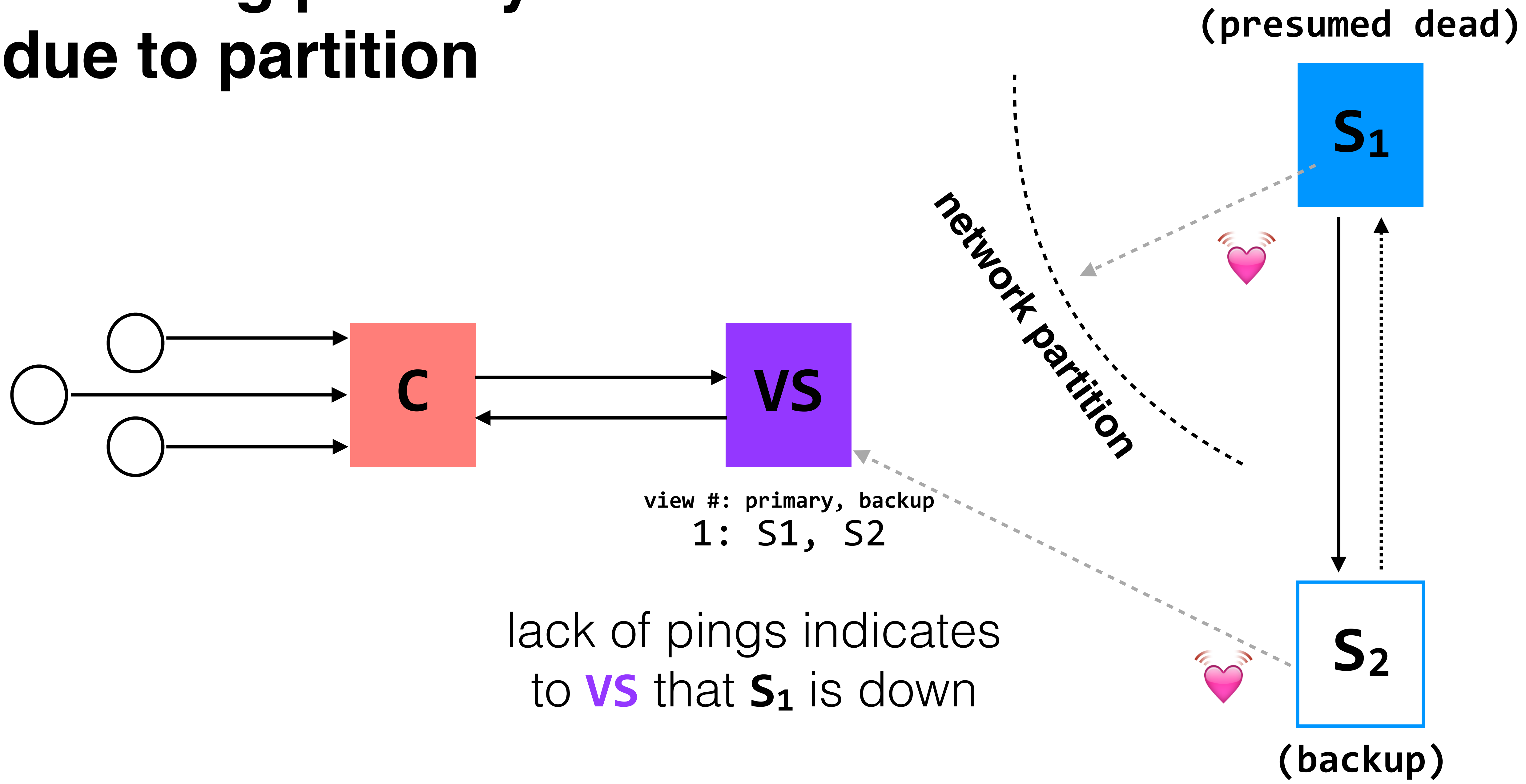


handling primary failure due to partition



suppose a partition keeps S_1 from communicating with the **view server**

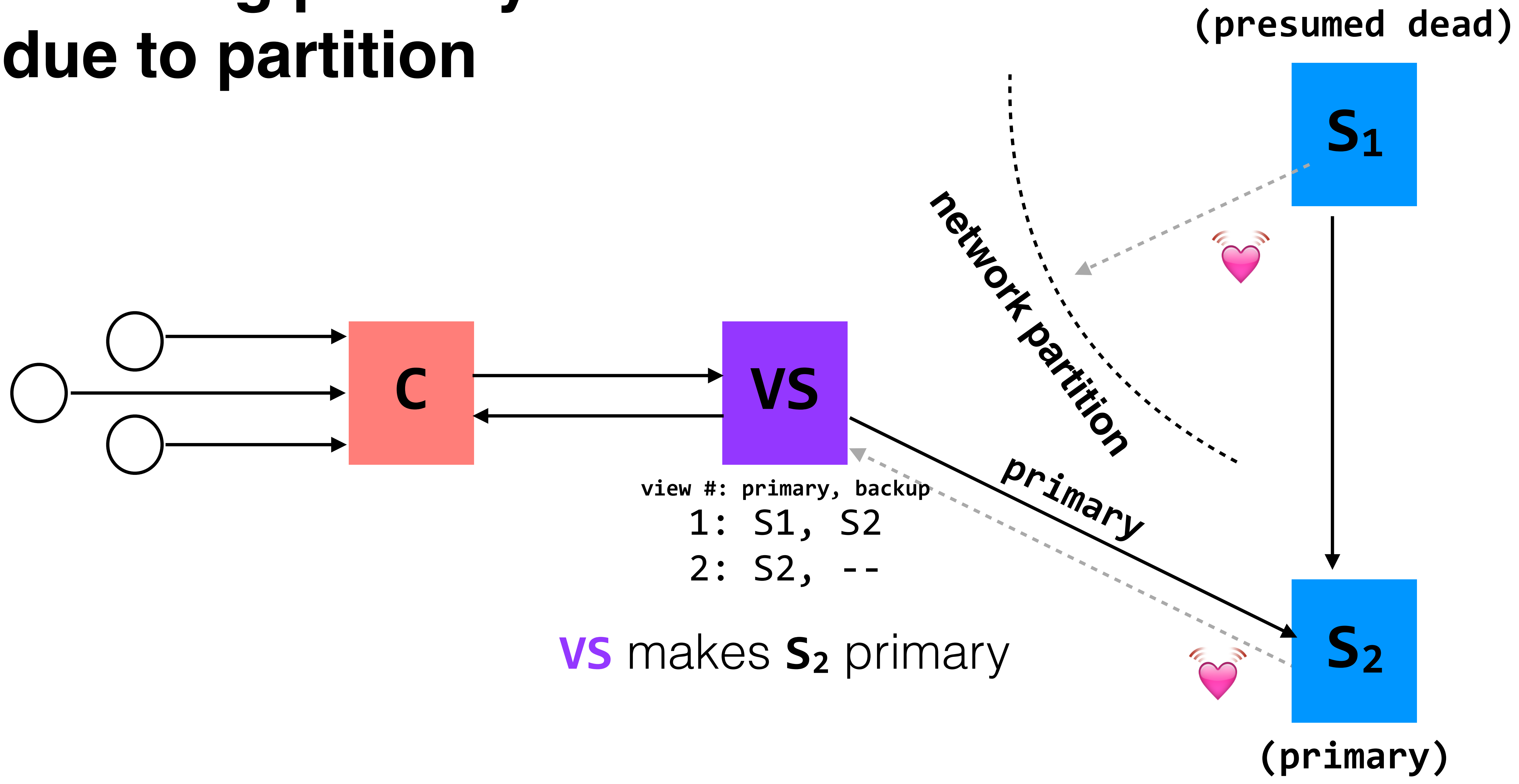
handling primary failure due to partition



view #: primary, backup
1: S1, S2

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

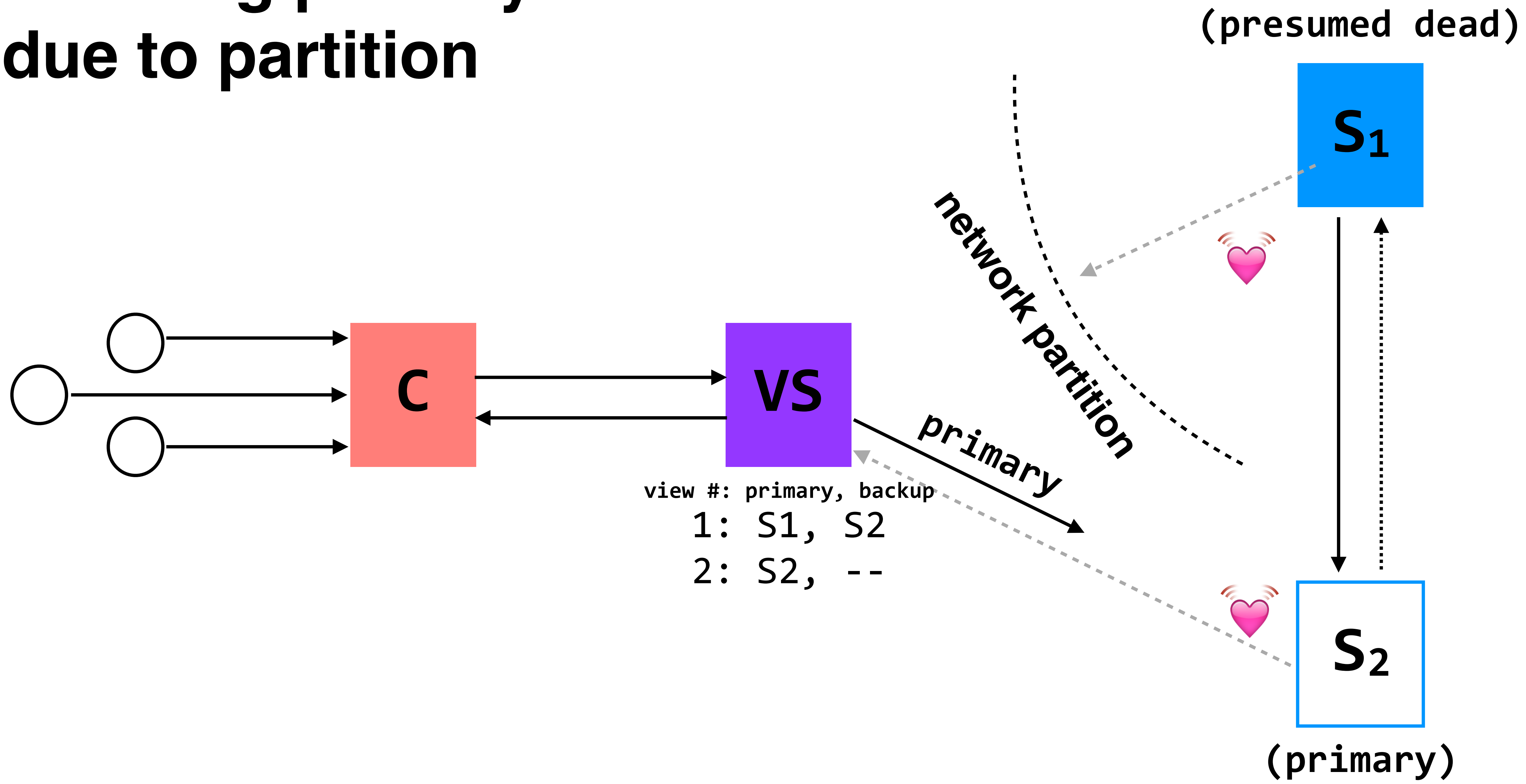
handling primary failure due to partition



```
view #: primary, backup
  1: S1, S2
  2: S2, --
```

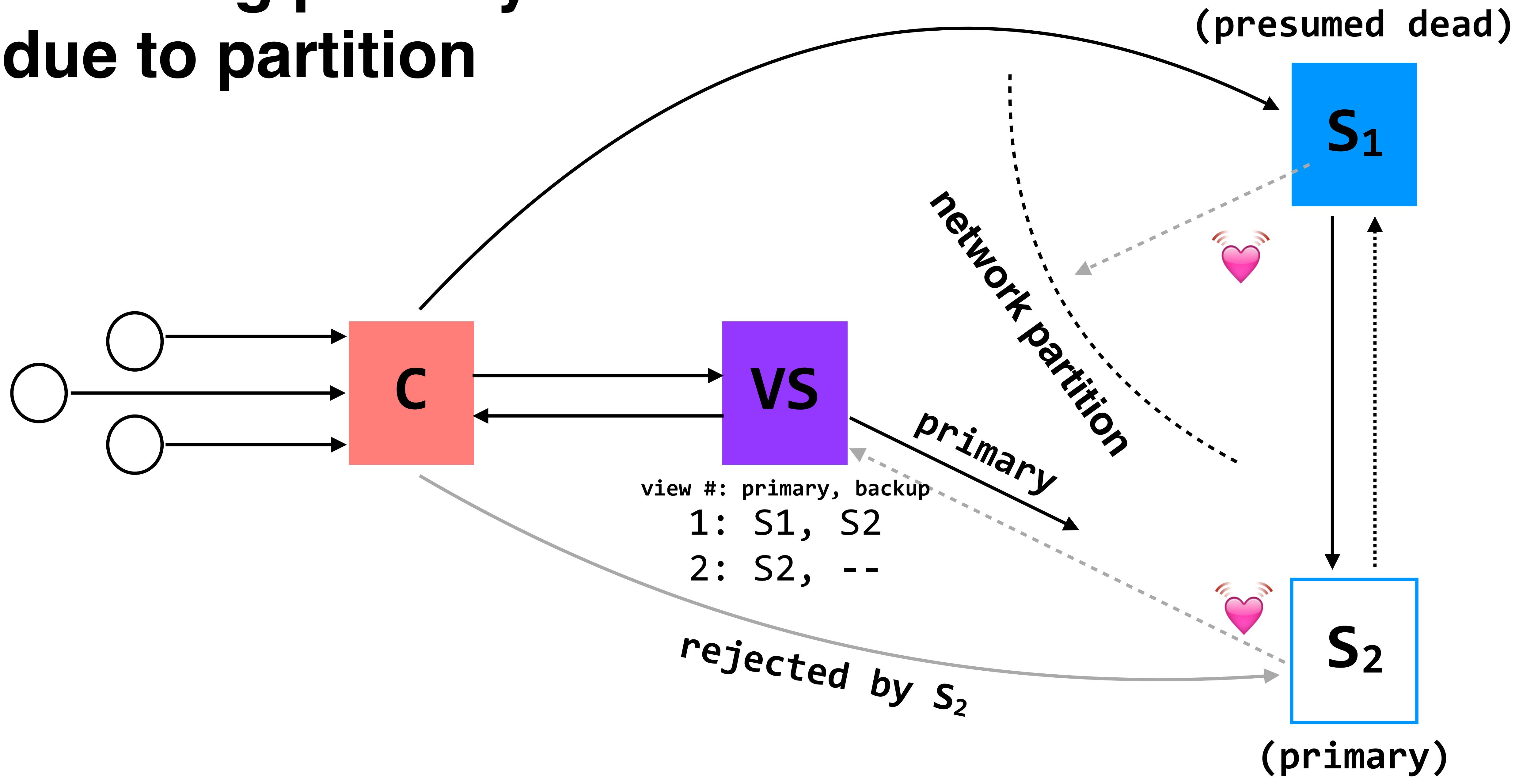
VS makes **S₂** primary

handling primary failure due to partition



question: what happens before S_2 knows it's the primary?

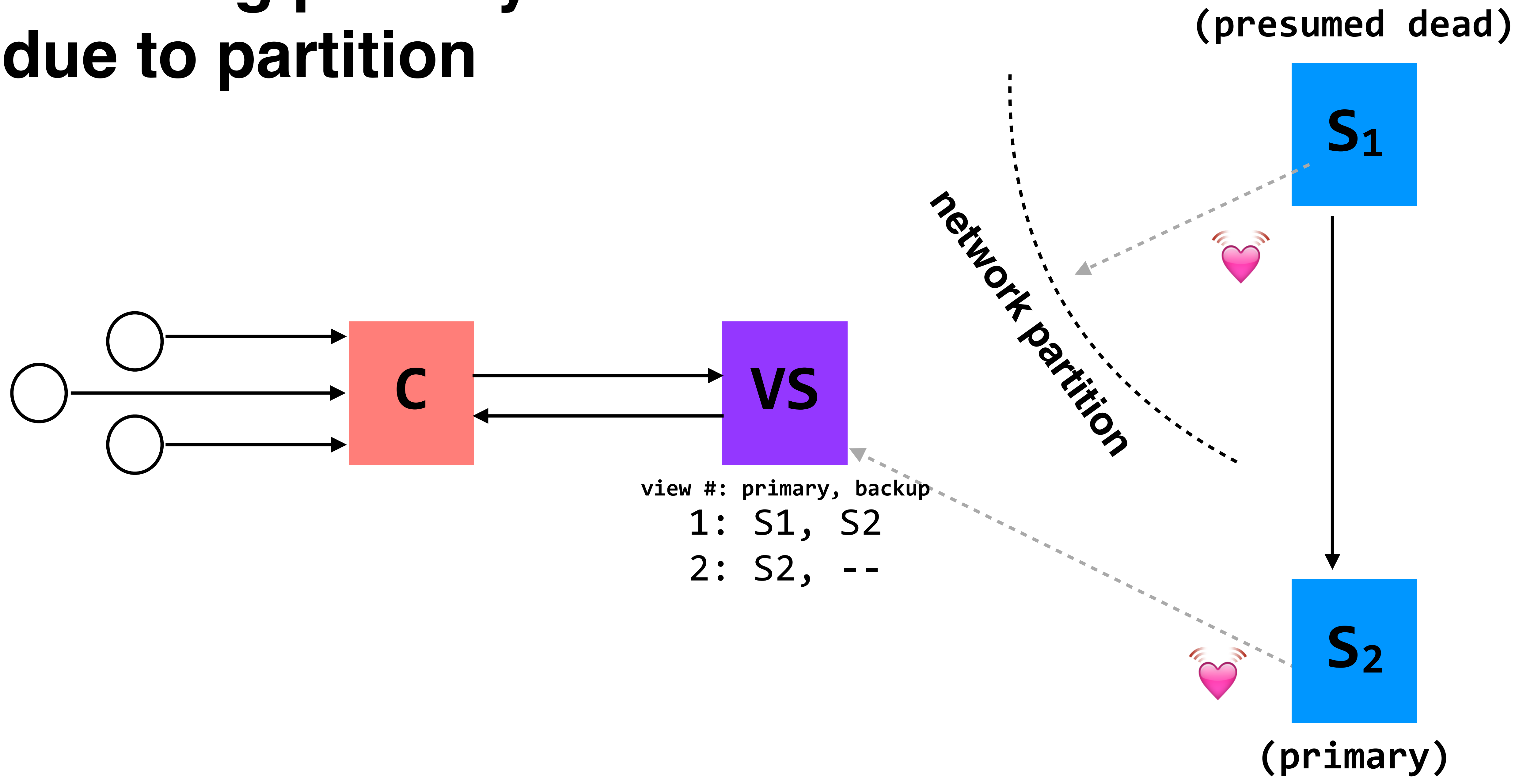
handling primary failure due to partition



S_2 will act as backup

(accept updates from S_1, reject coordinator requests)

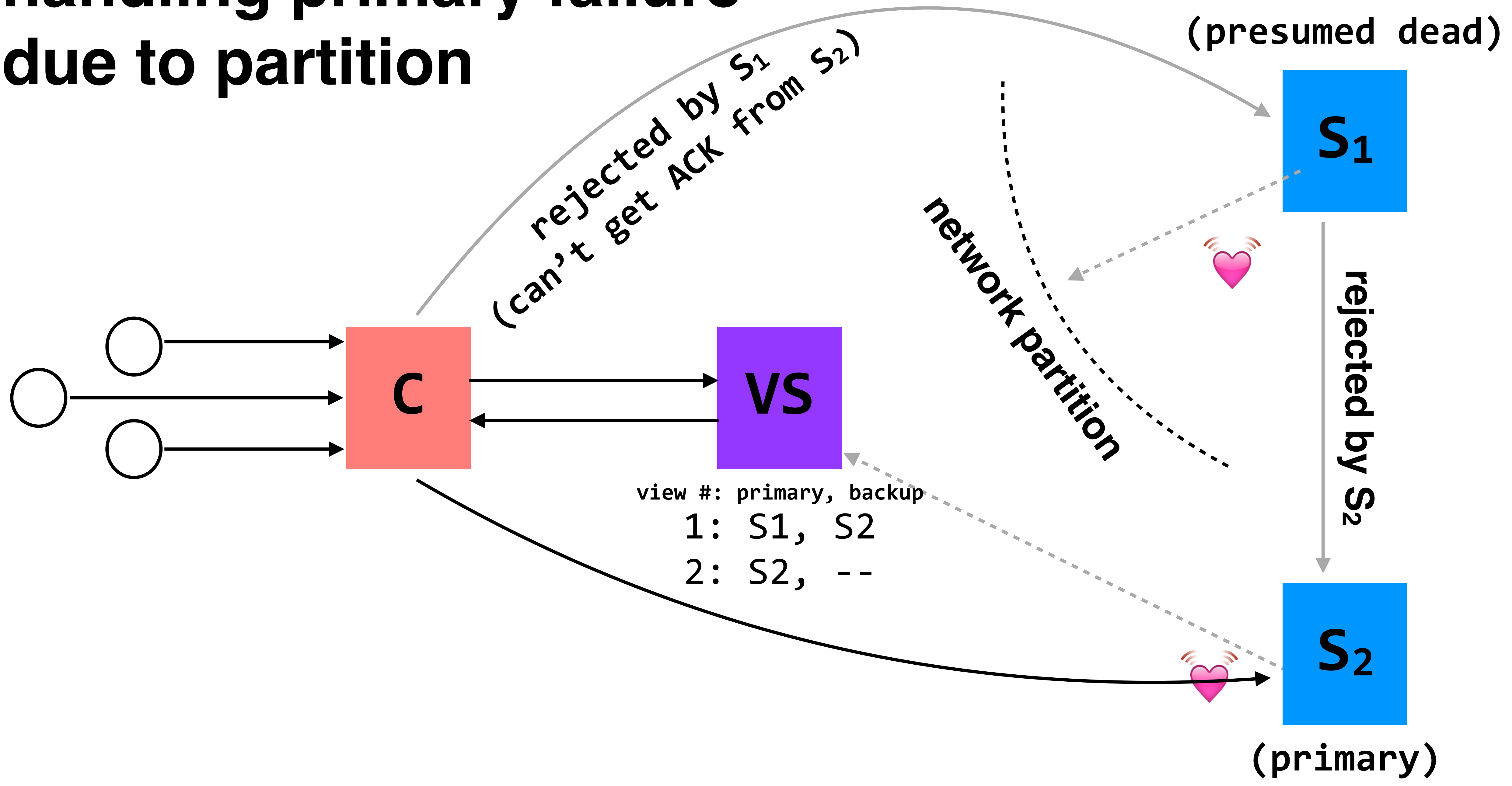
handling primary failure due to partition



```
view #: primary, backup
   1: S1, S2
   2: S2, --
```

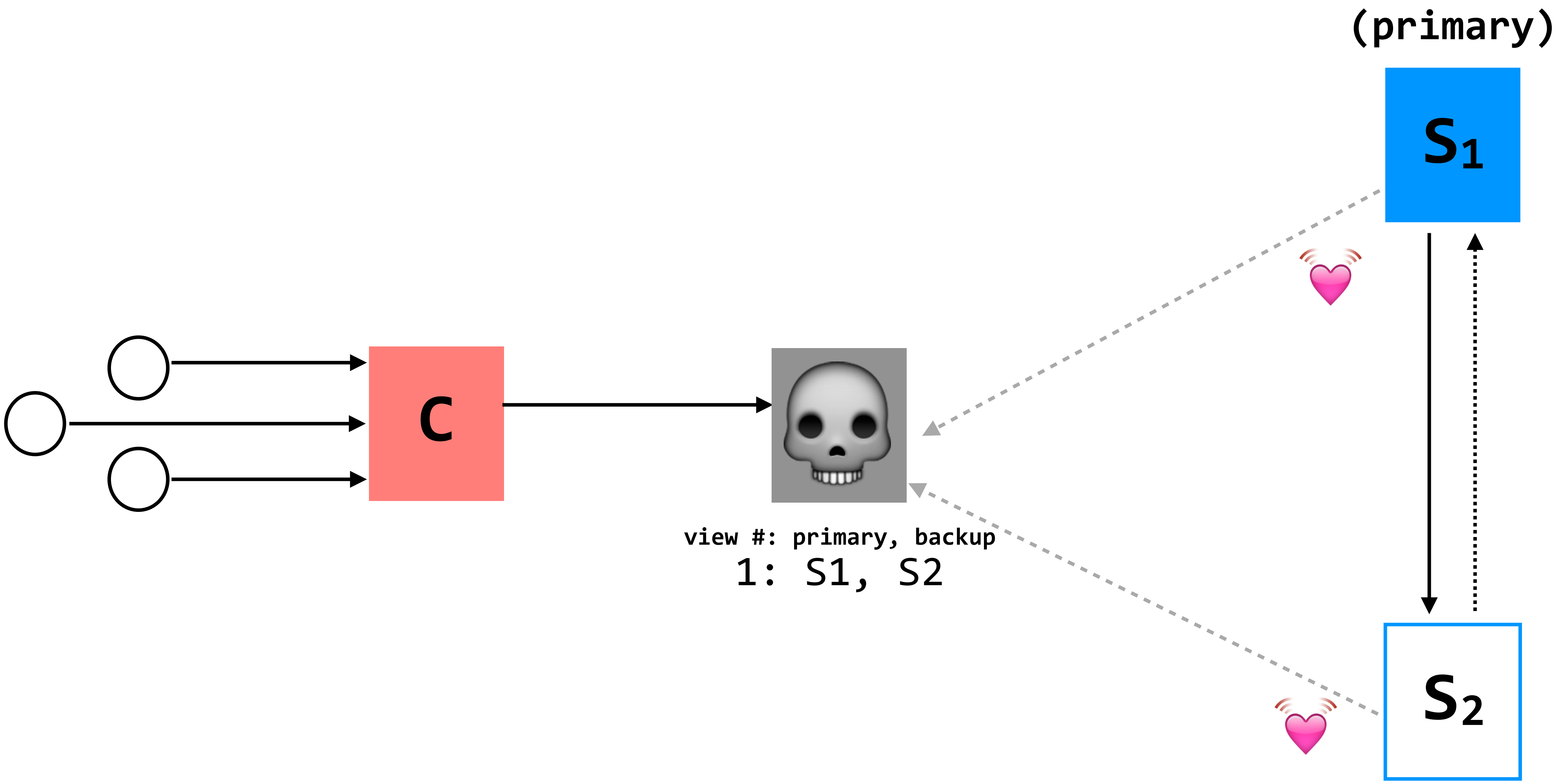
question: what happens after S_2 knows it's the primary, but S_1 also thinks it is?

handling primary failure due to partition



S_1 won't be able to act as primary

(can't accept client requests because it won't get ACKs from S_2)



problem: what if view server fails?

go to recitation tomorrow and find out!

- **Replicated state machines (RSMs)** provide **single-copy consistency**: operations complete 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. It can also recruit new backups after servers fail.
- To extend this model to handle view-server failures, we need a mechanism to provide **distributed consensus**; see tomorrow's recitation (on Raft).