6.033 Spring 2017

Lecture #19

- **Distributed transactions**
  - **Availability**
  - **Replicated State Machines**

goal: build reliable systems from unreliable components

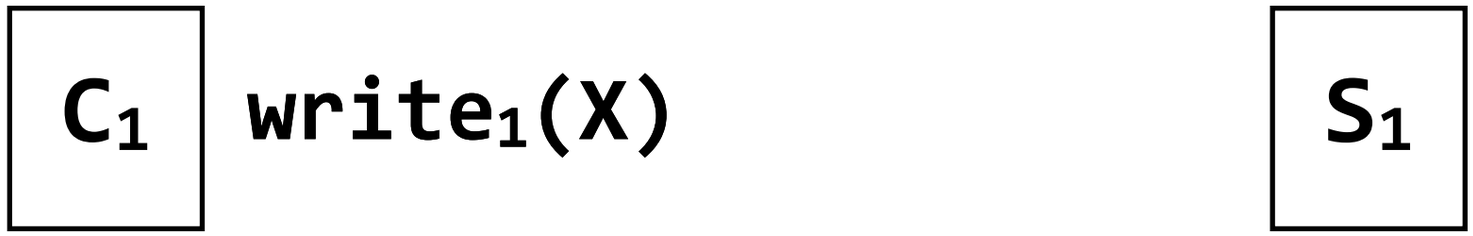
the abstraction that makes that easier is

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

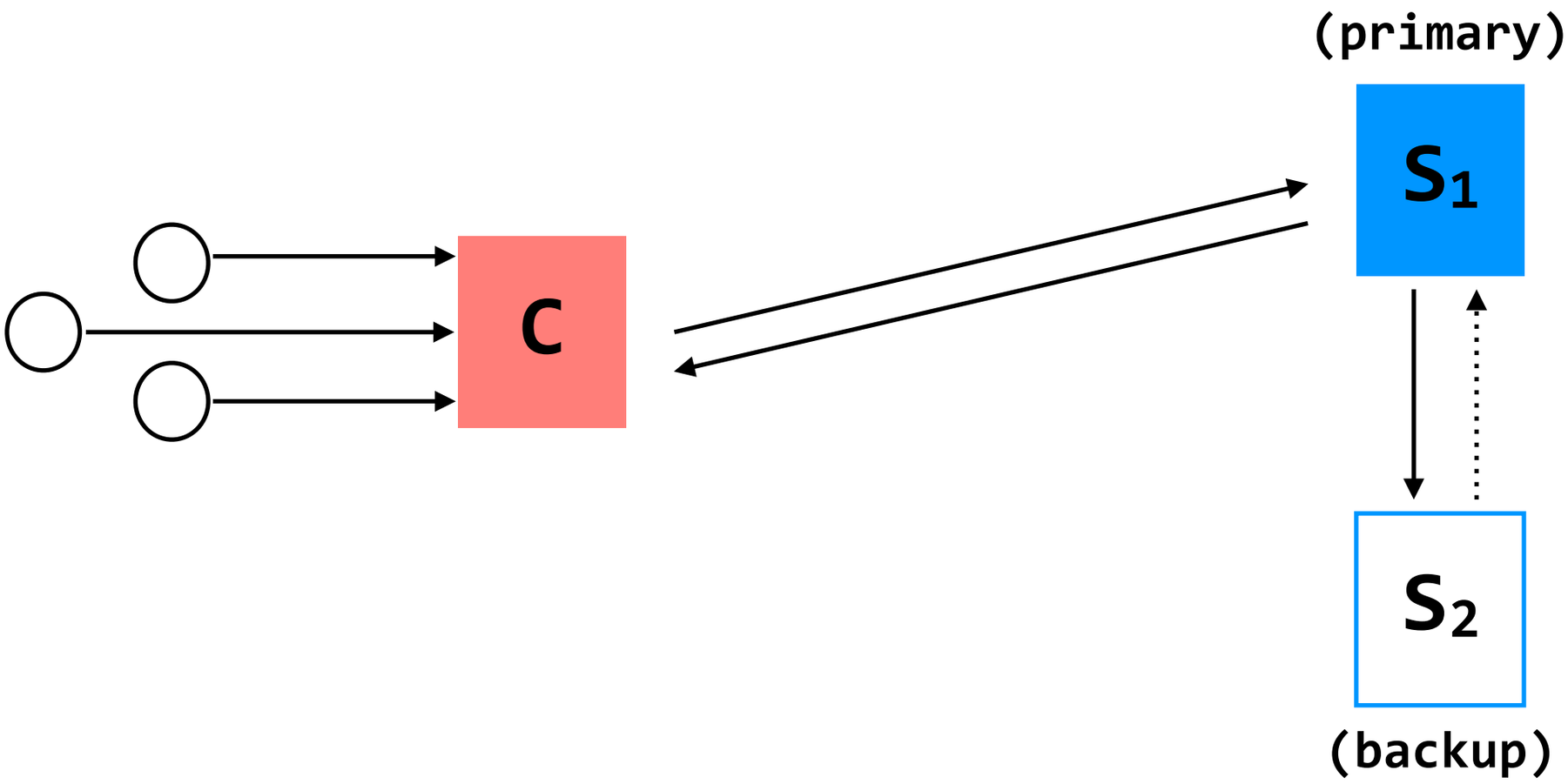
atomicity ⟶ **shadow copies** (simple, poor performance) or **logs** (better performance, a bit more complex)

isolation ⟶ **two-phase locking**

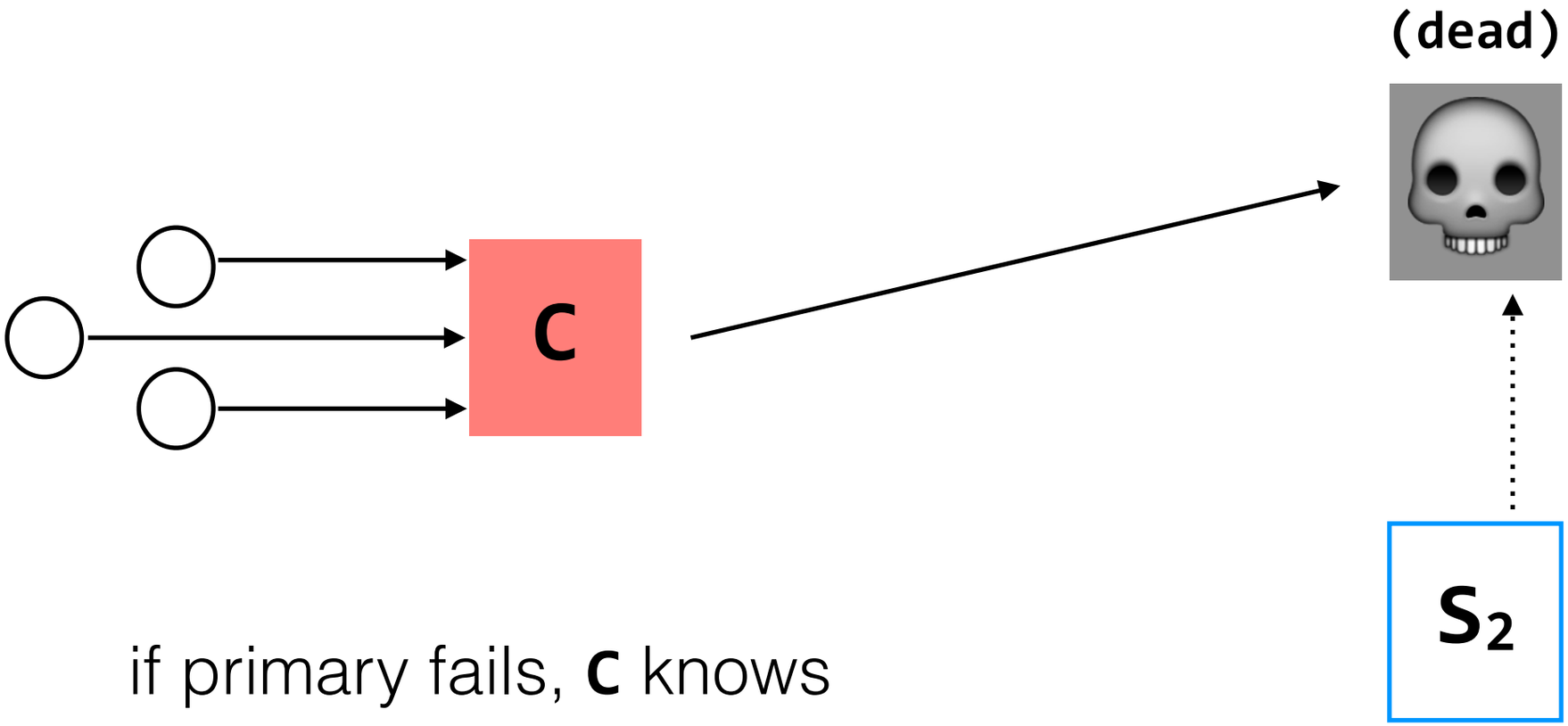
we also want transaction-based systems to be **distributed** — to run across multiple machines — and to remain **available** even through failures


C1
write1(X)
S1
C2
write2(X)
S2
(replica of S1)

problem: replica servers can become inconsistent

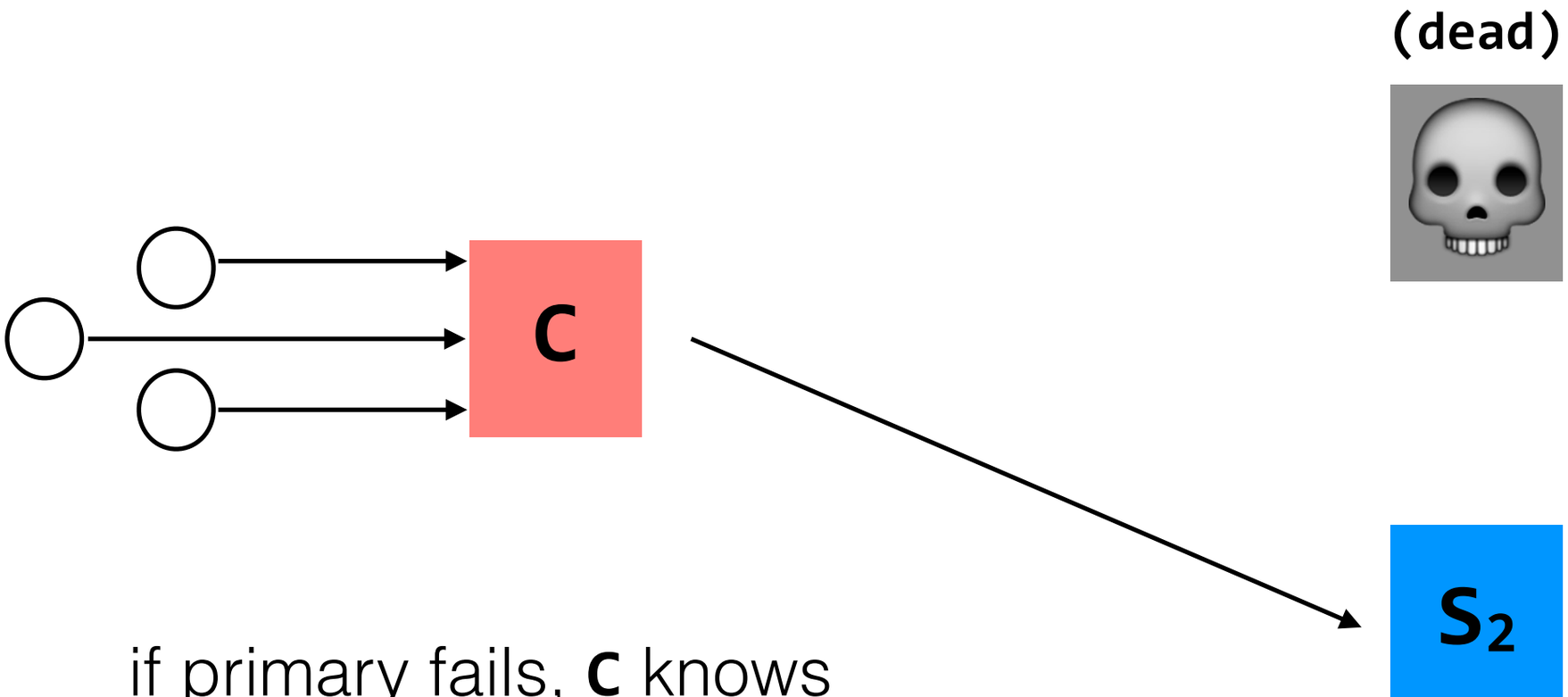


attempt: coordinators communicate with primary servers, who communicate with backup servers



if primary fails, **C** knows about S_2, and switches

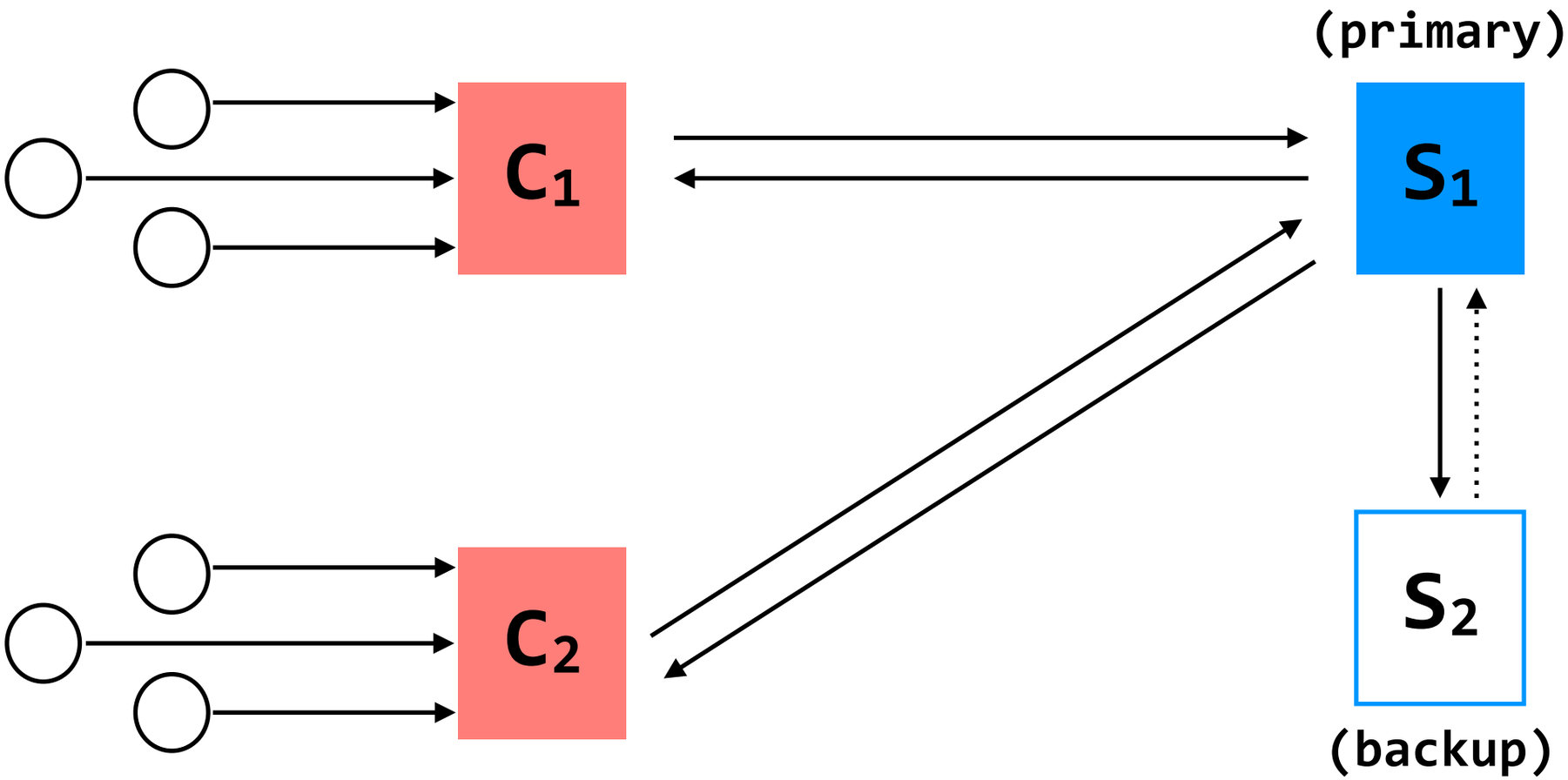
attempt: coordinators communicate with primary servers, who communicate with backup servers



if primary fails, **C** knows about **S_2**, and switches

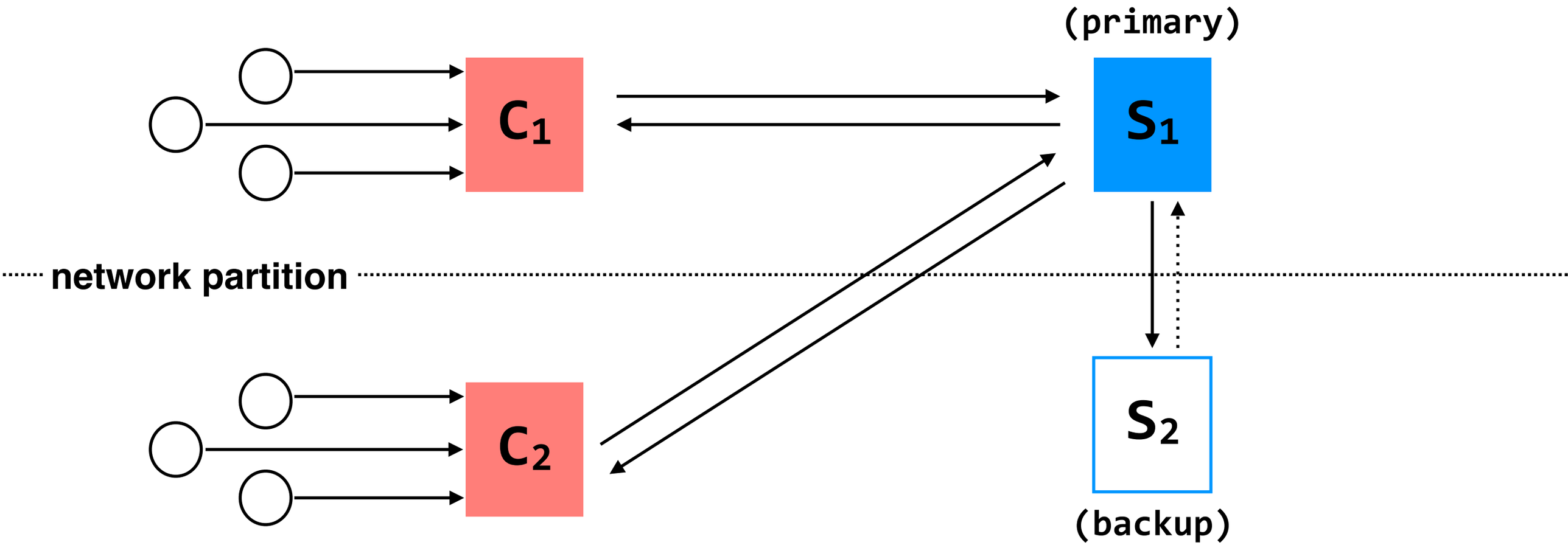
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems



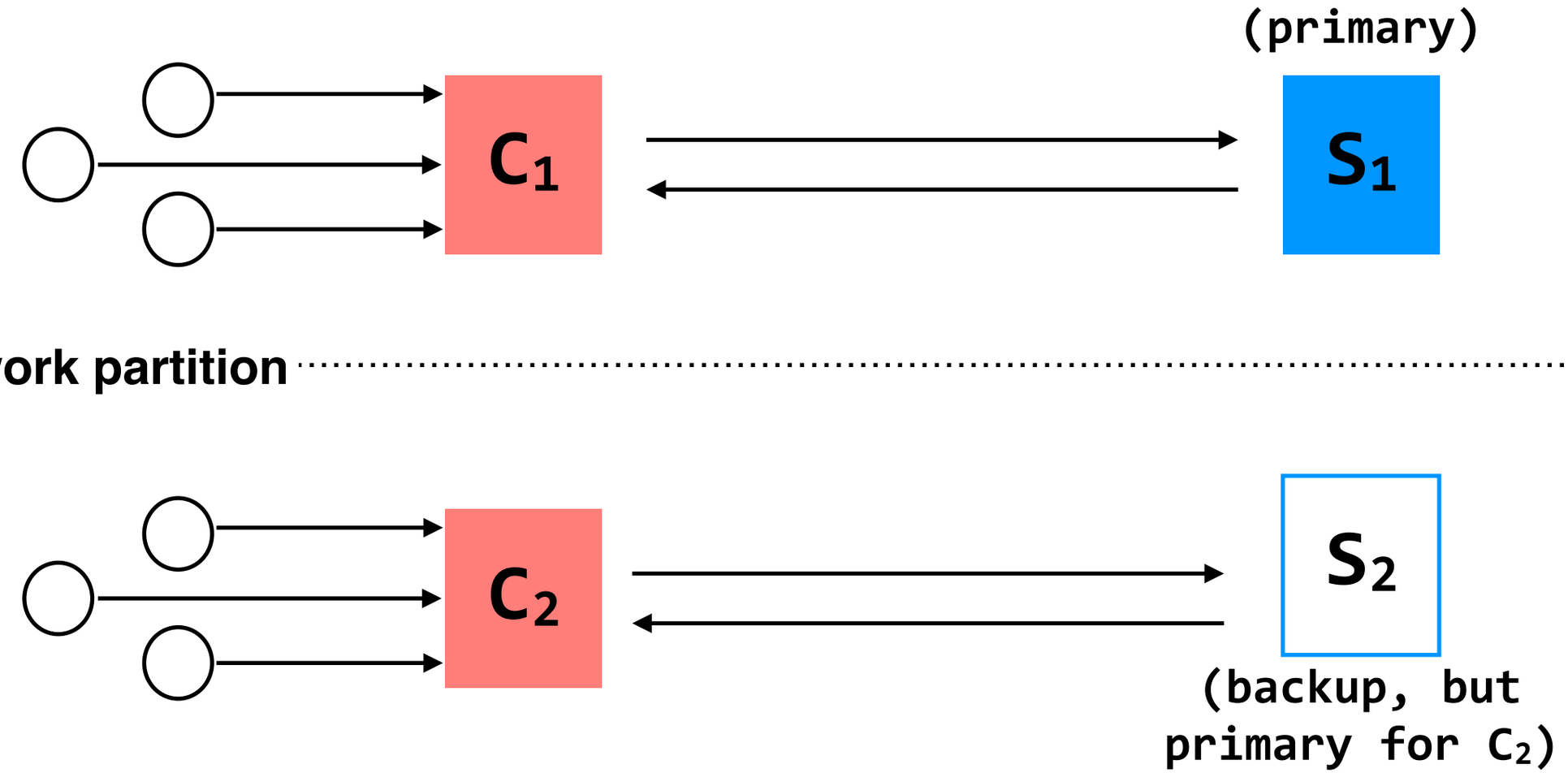
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems



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

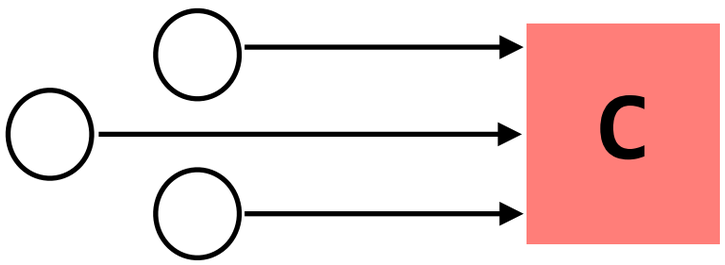
multiple coordinators + the network = problems

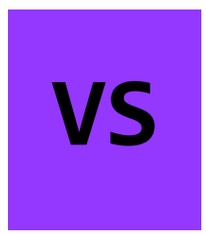


C_1 and C_2 are using different primaries;

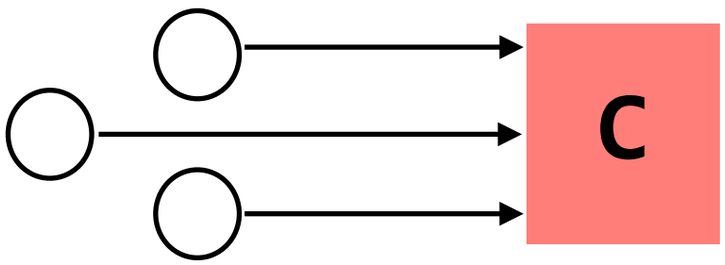
S_1 and S_2 are no longer consistent

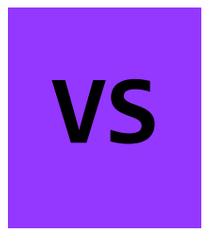
attempt: coordinators communicate with primary servers, who communicate with backup servers



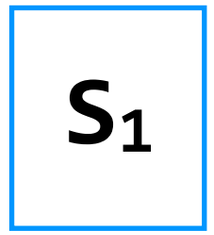


use a **view server**, which determines which replica is the primary

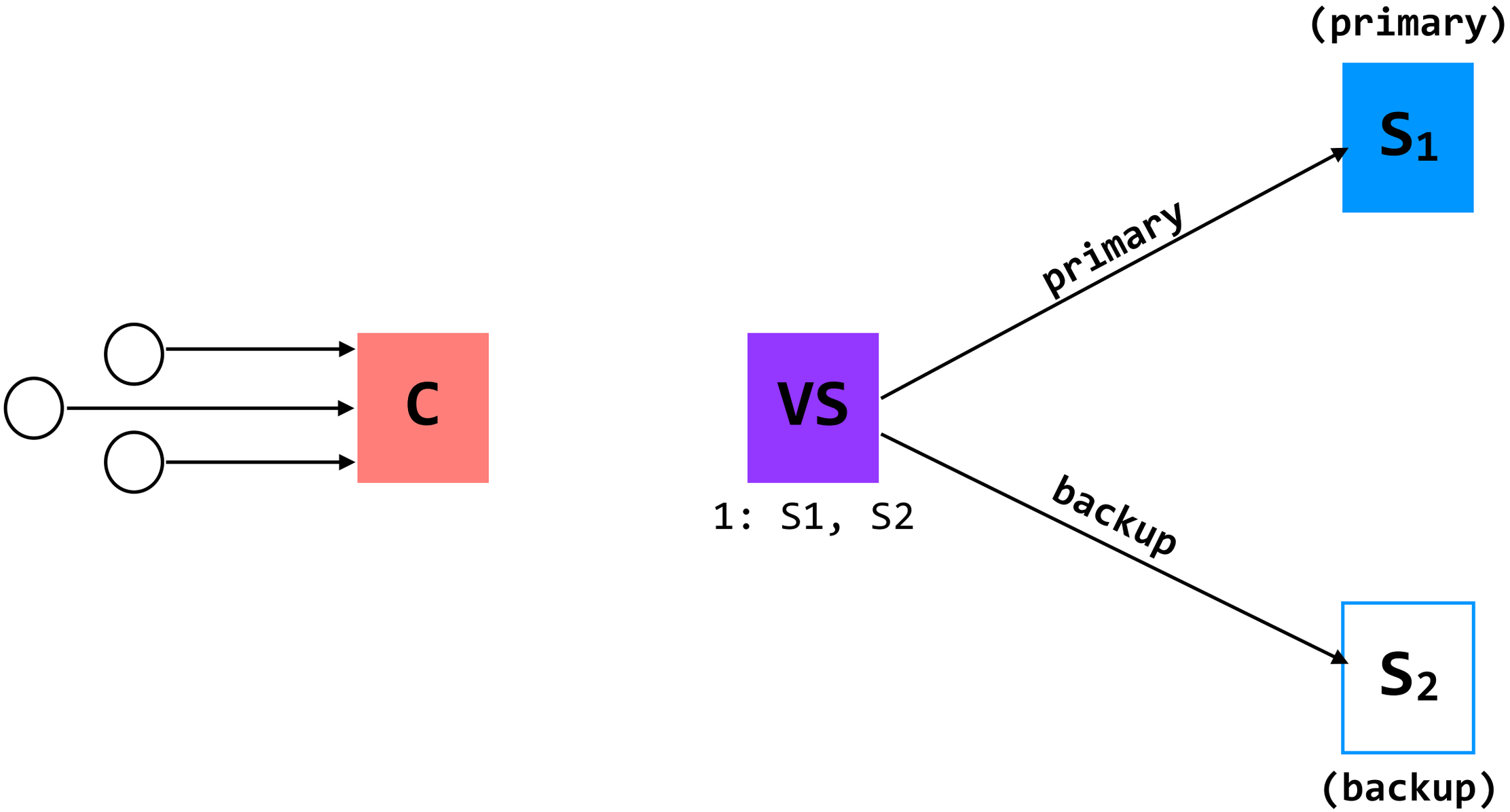




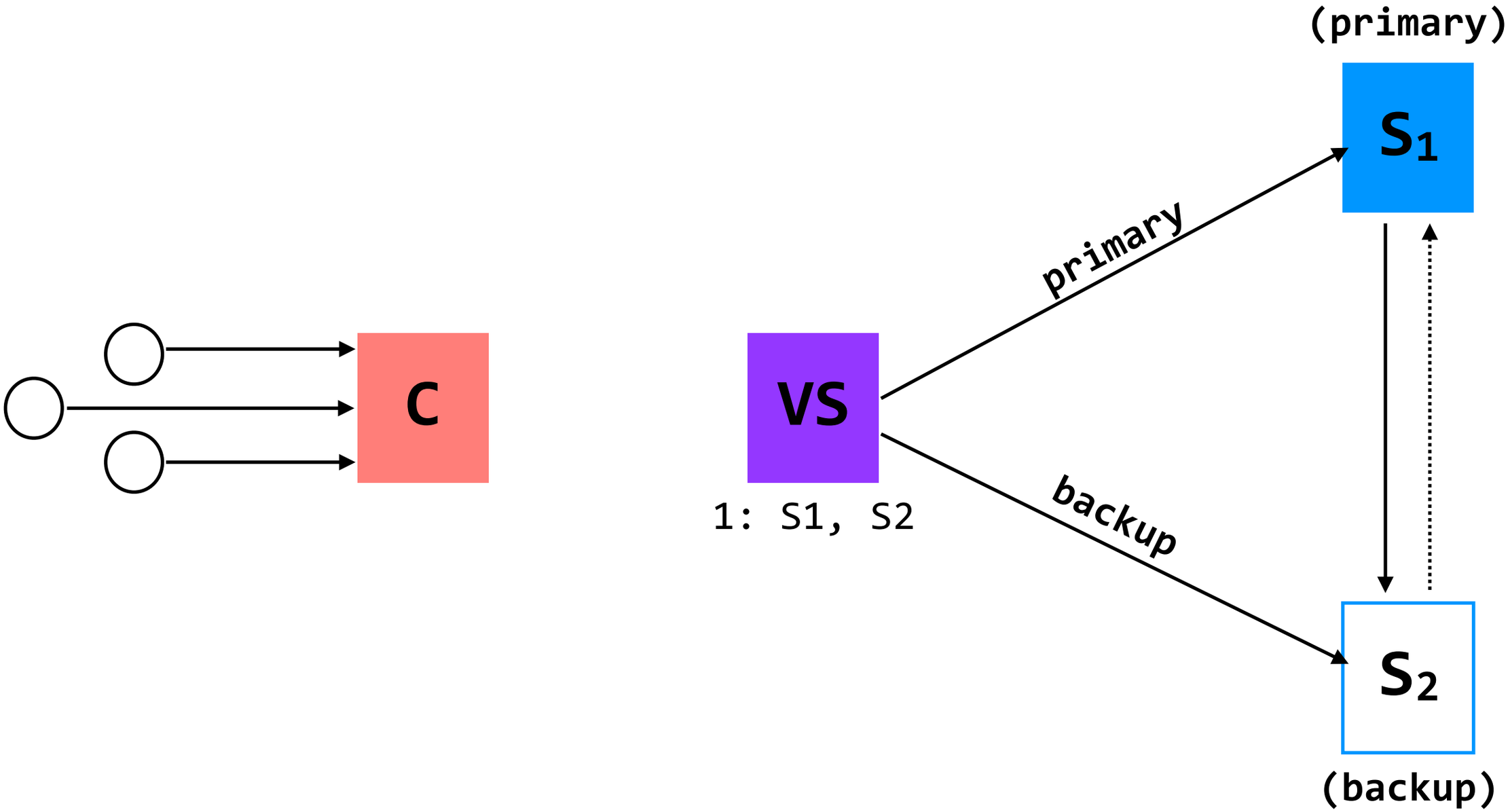
1: S1, S2



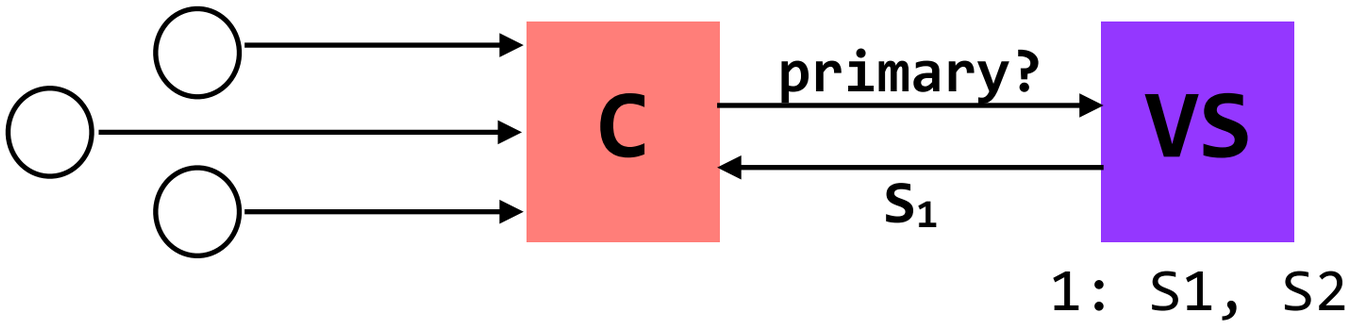
use a **view server**, which determines which replica is the primary

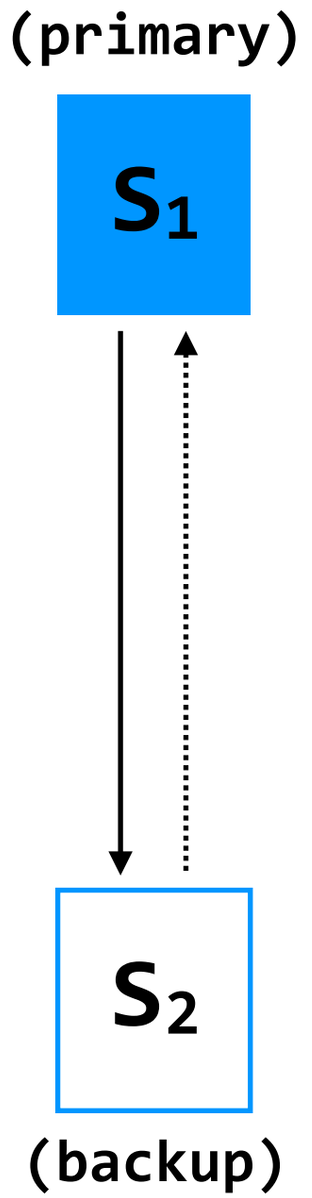


use a **view server**, which determines which replica is the primary

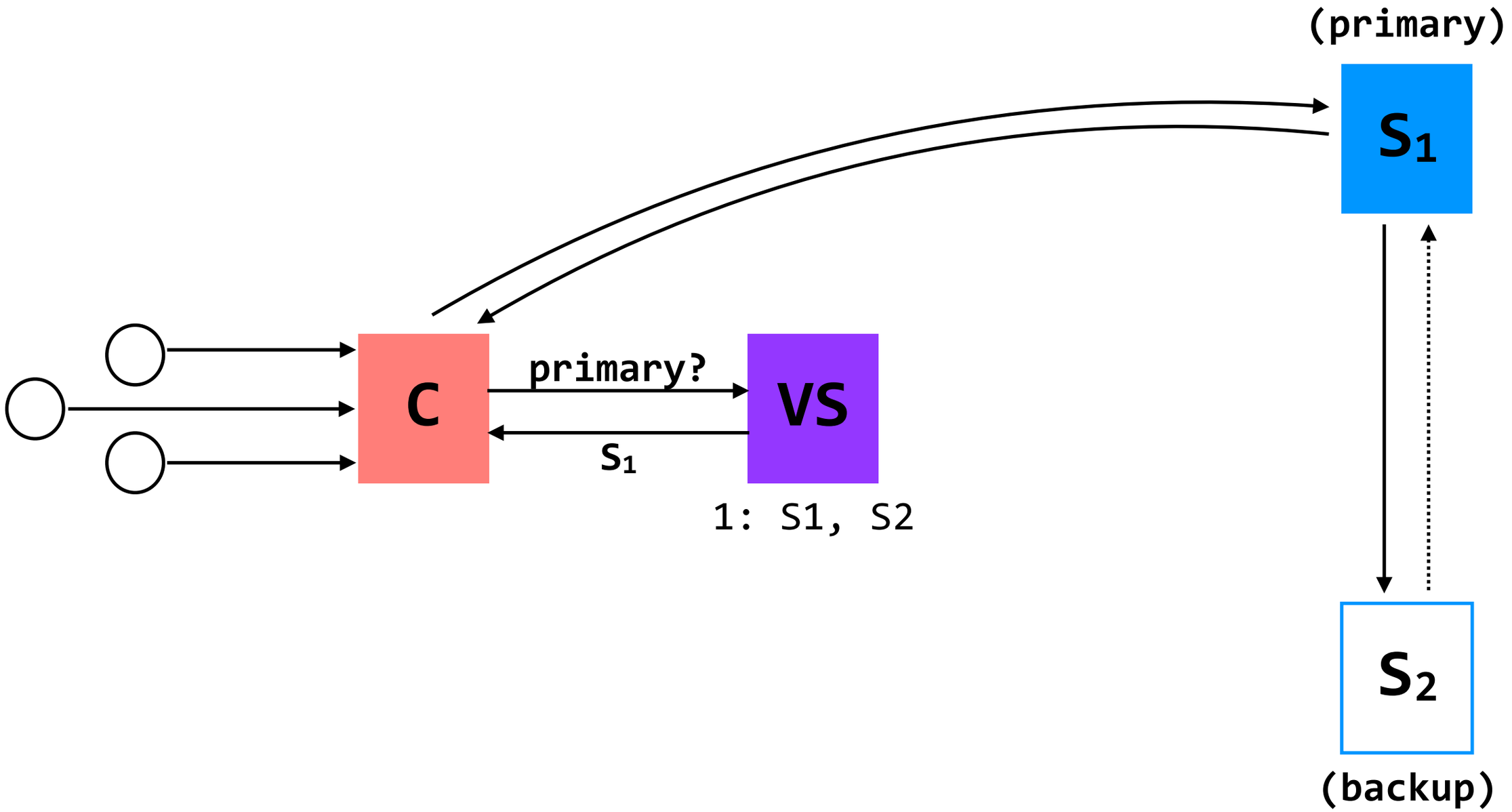


use a **view server**, which determines which replica is the primary

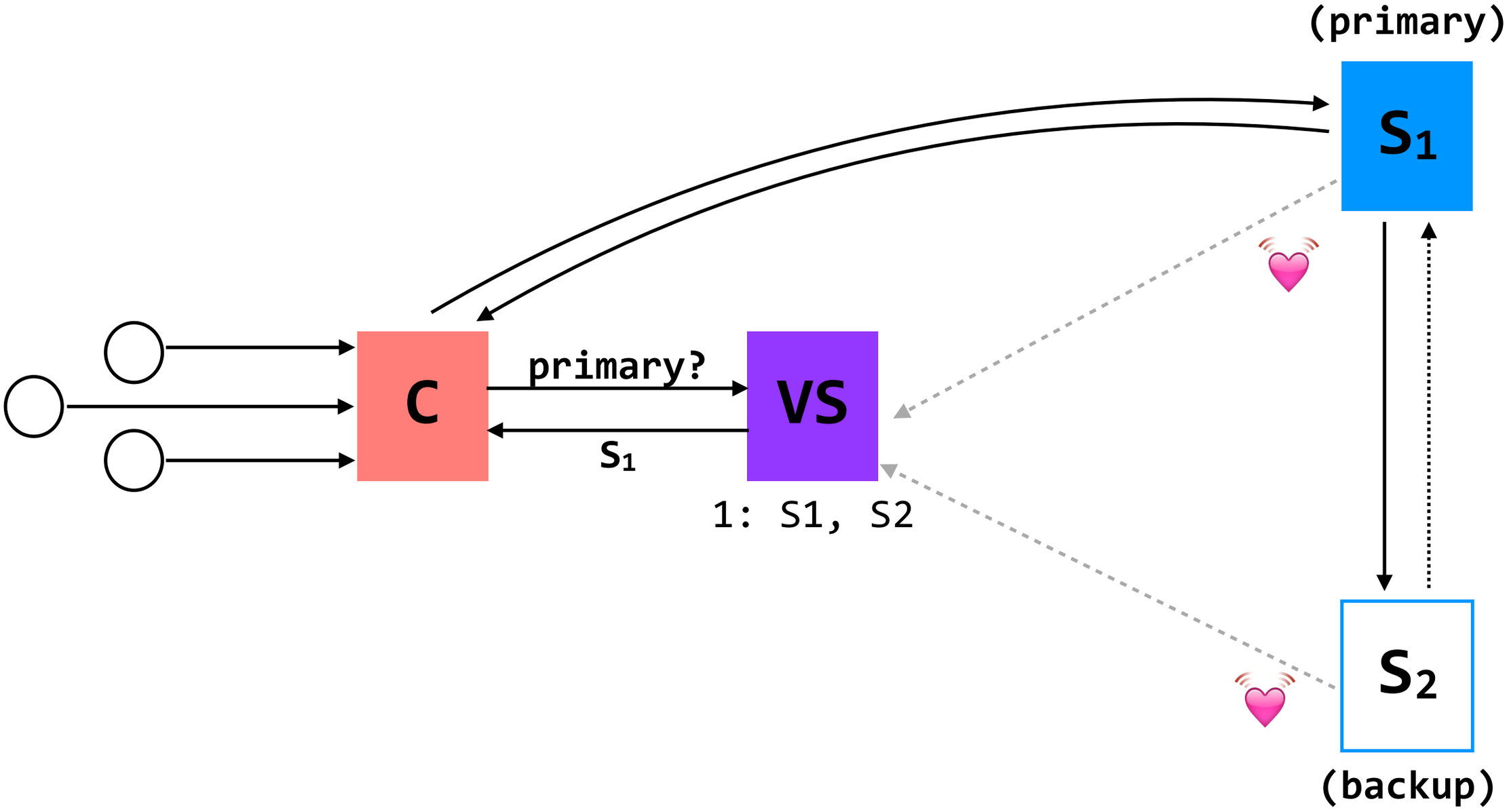




use a **view server**, which determines which replica is the primary

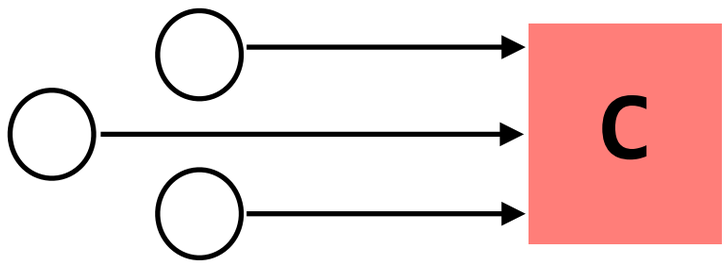


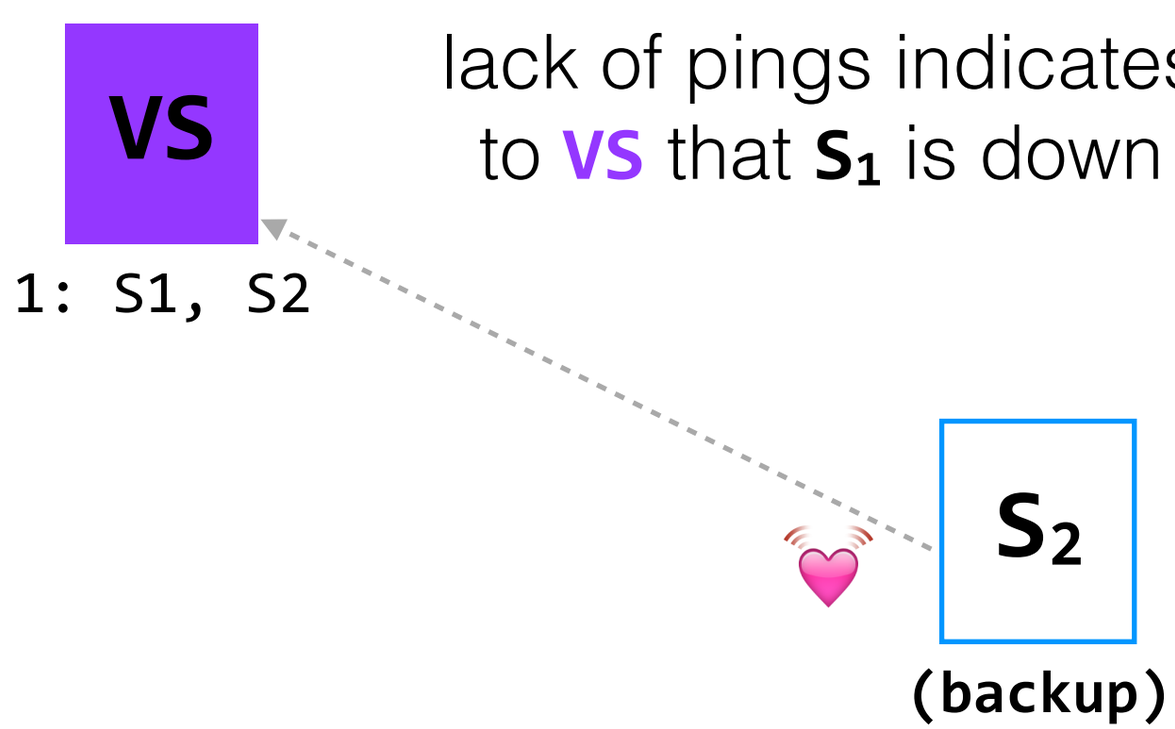
use a **view server**, which determines which replica is the primary



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

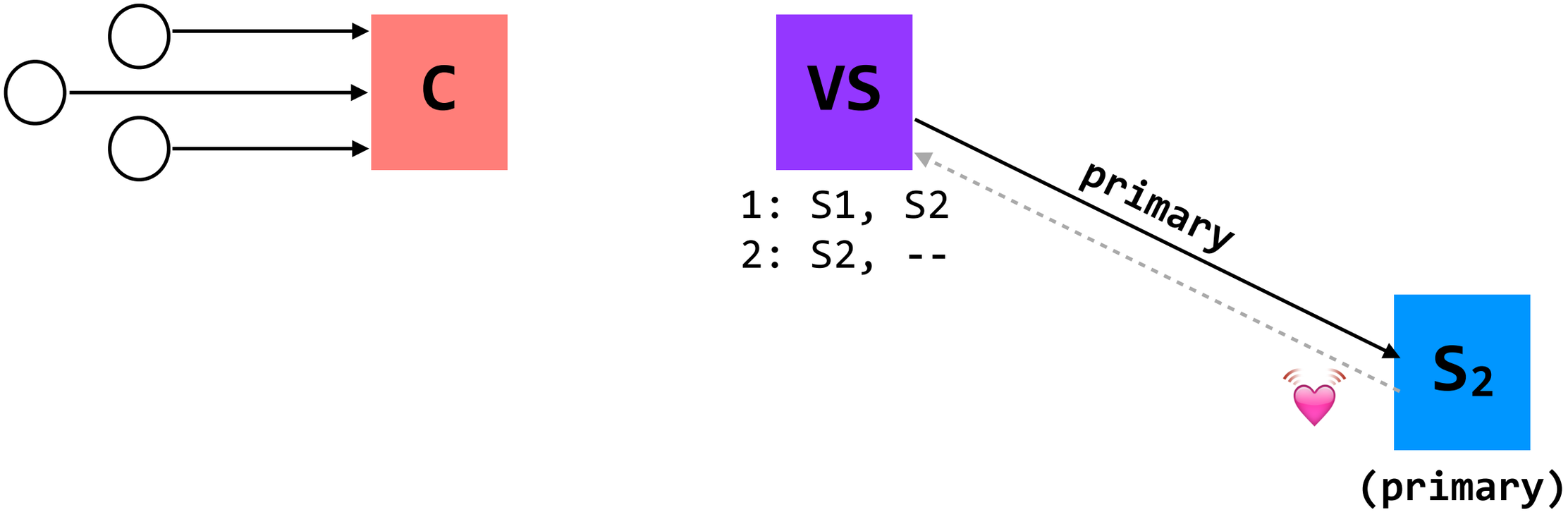
handling primary failure



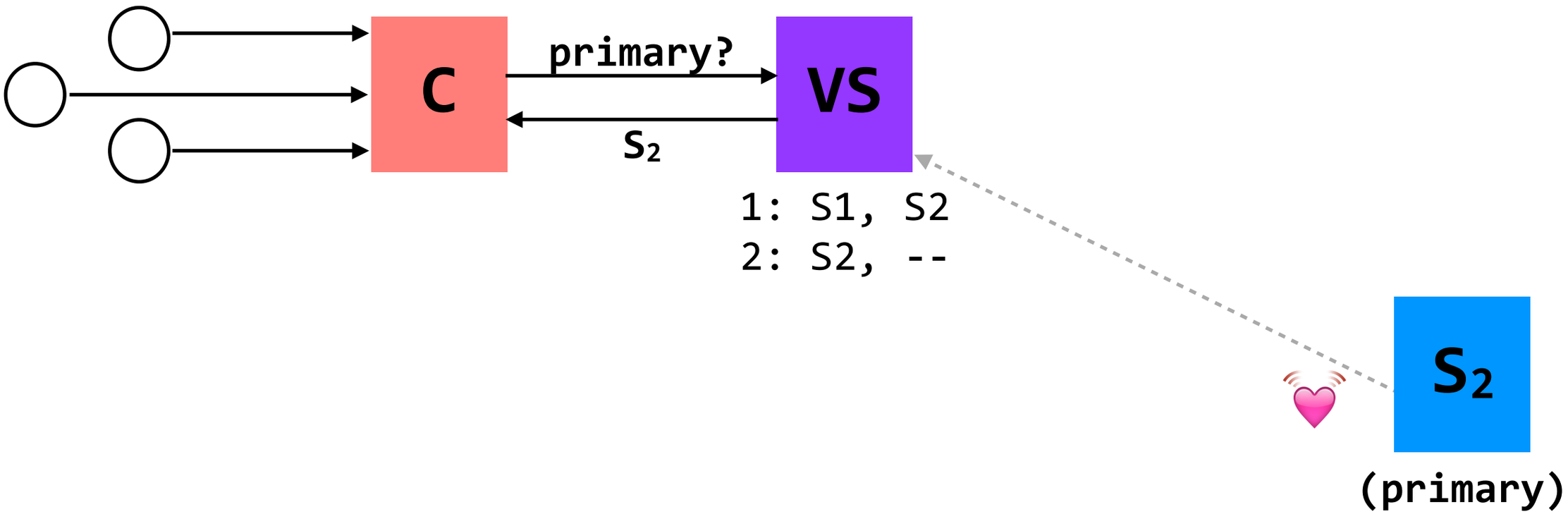


lack of pings indicates to **VS** that S_1 is down

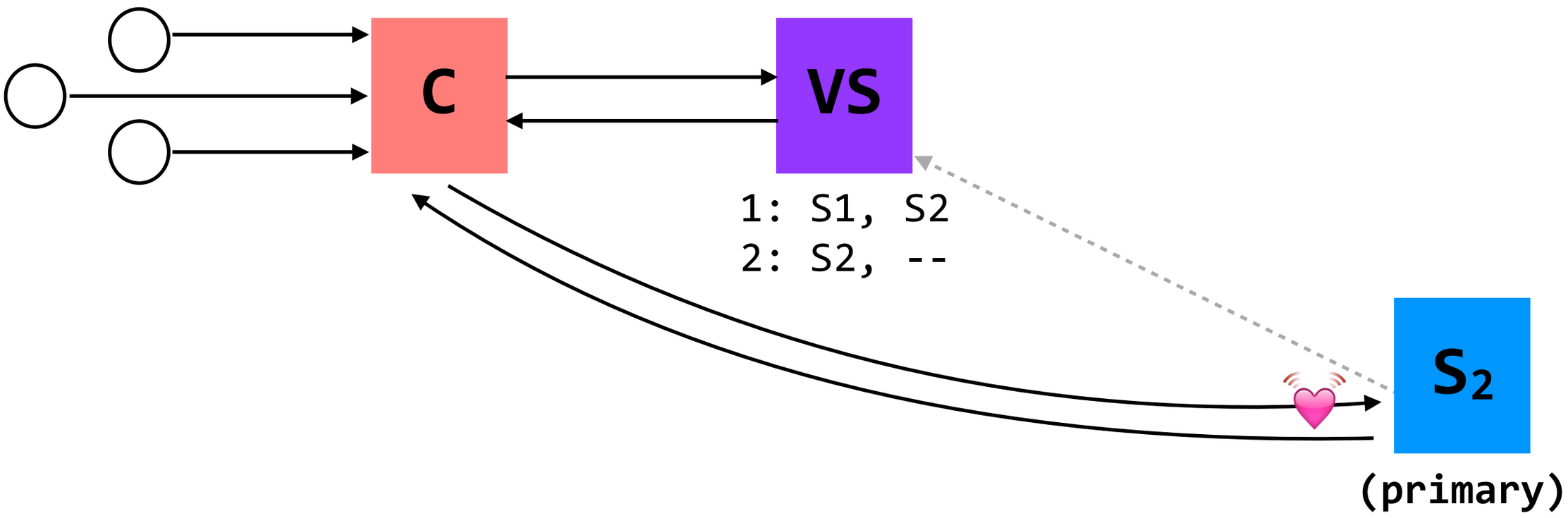
handling primary failure



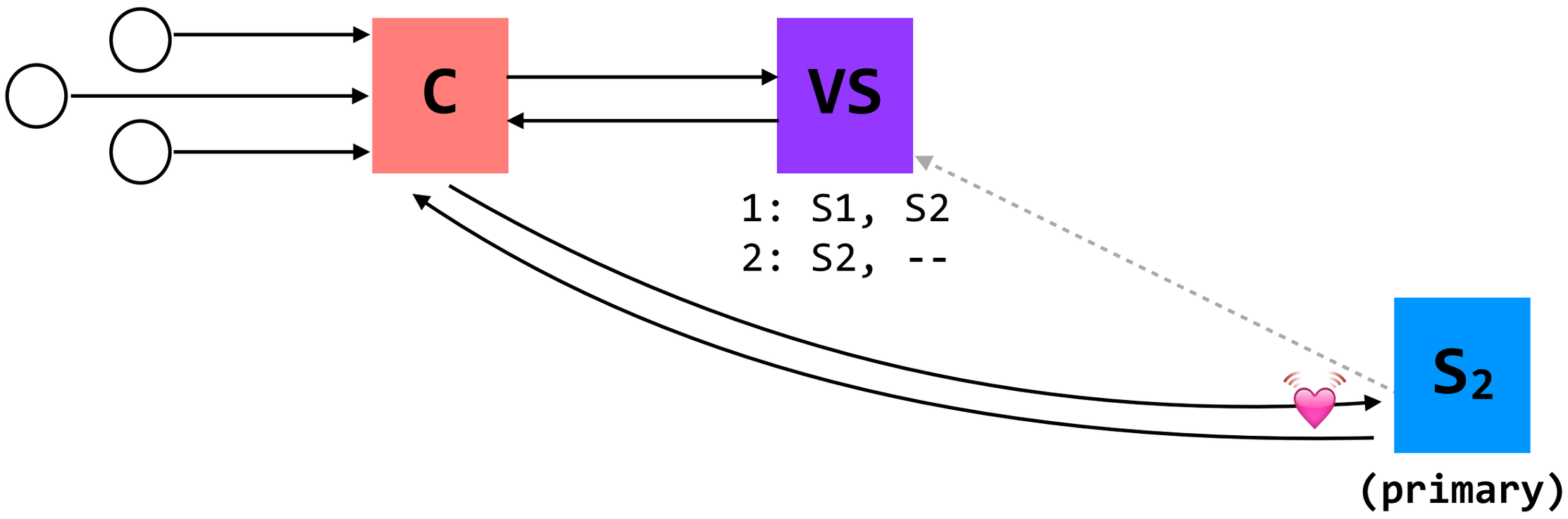
handling primary failure



handling primary failure



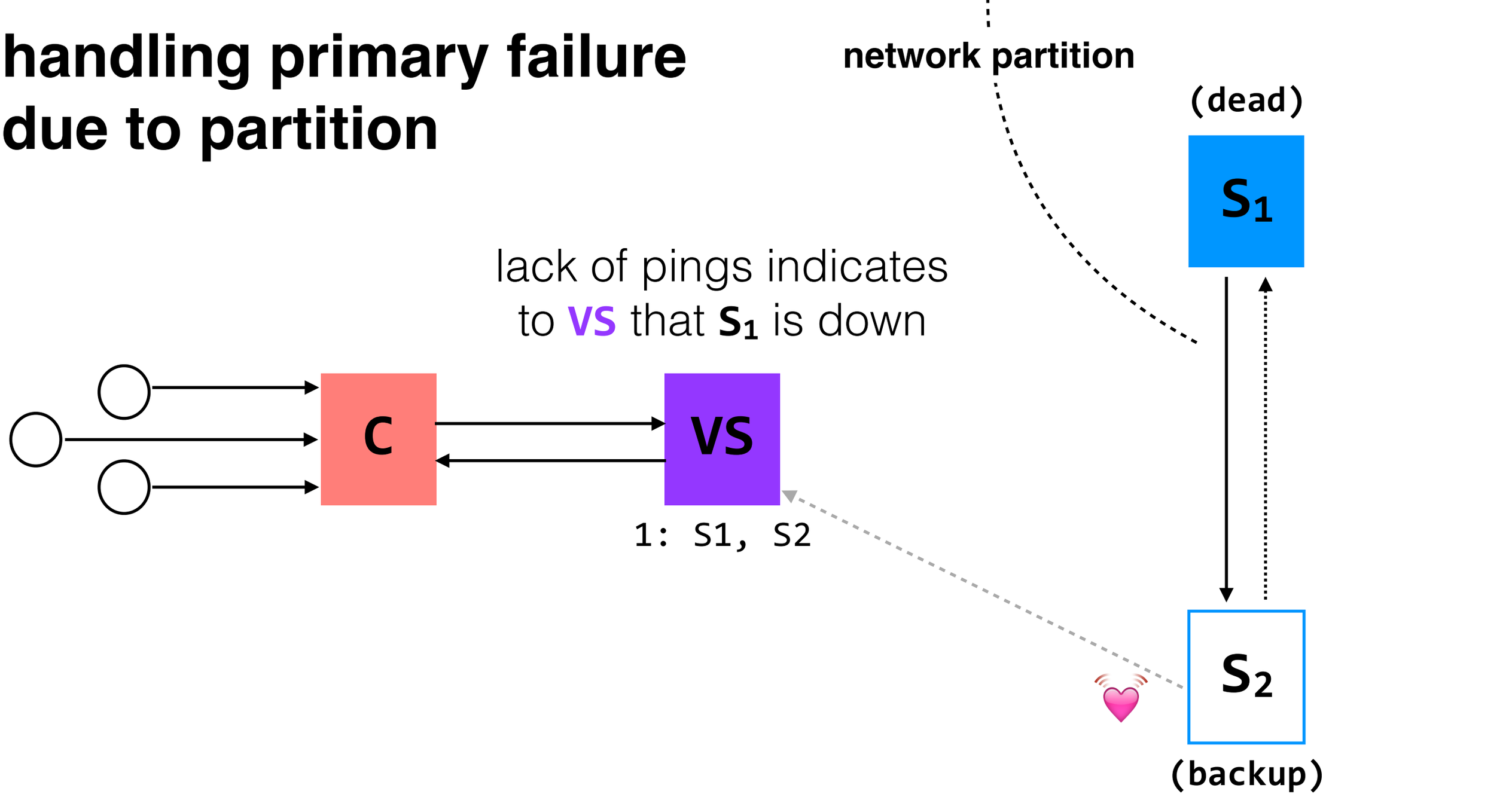
handling primary failure



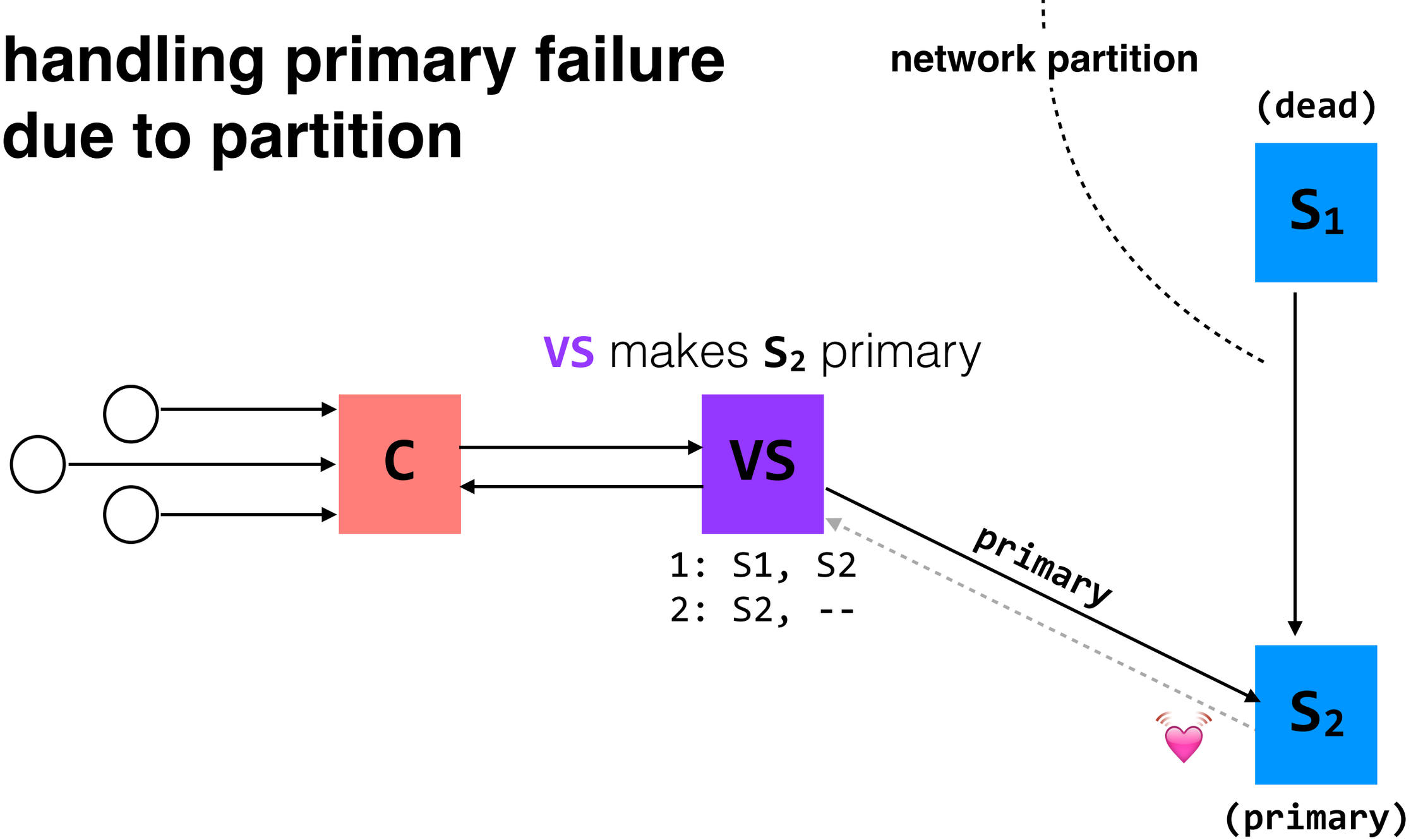
before S_2 knows it's primary, it will reject any requests from clients

(and if clients had contacted S_1 after it failed but before it was deemed dead, they would have received no response)

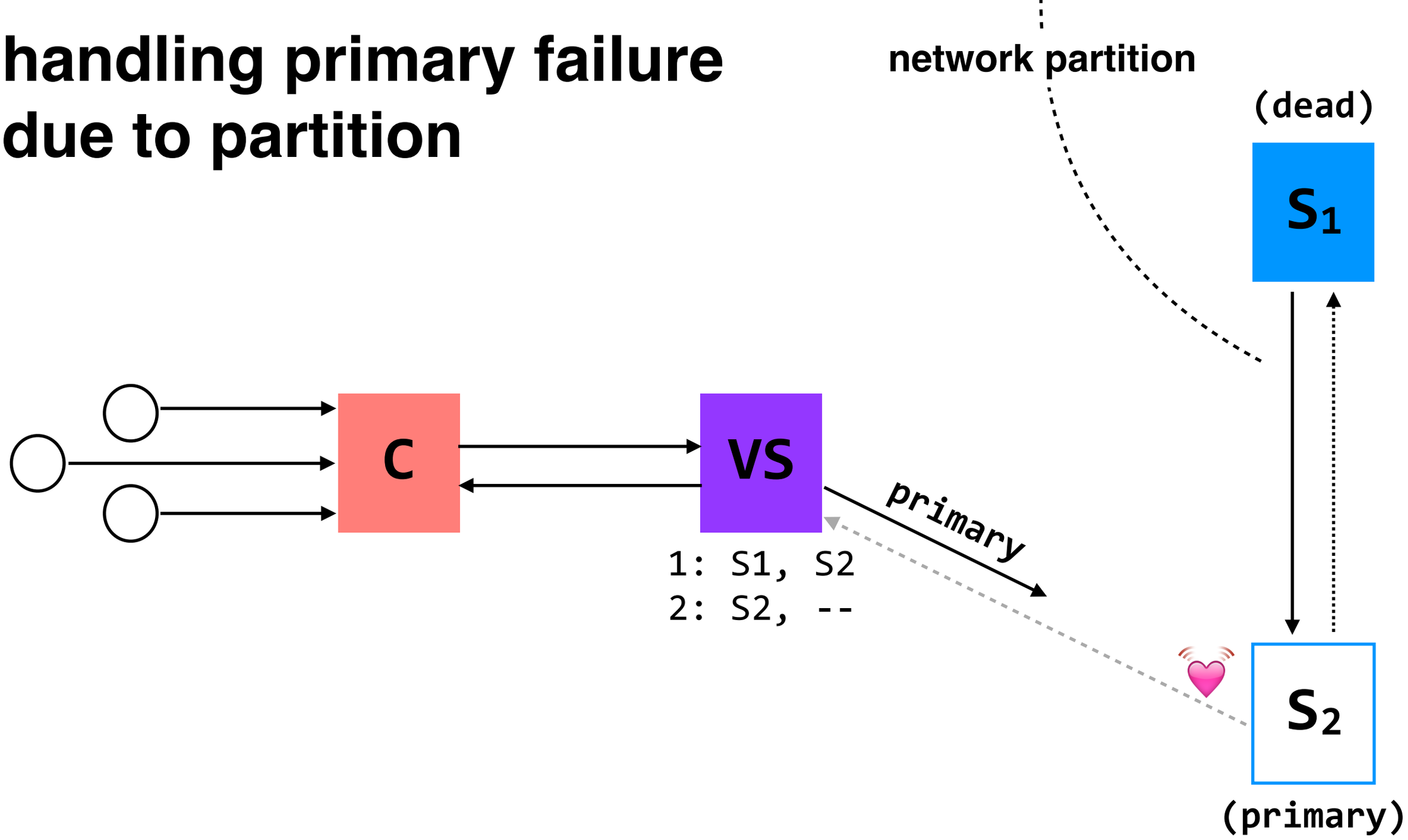
handling primary failure due to partition



handling primary failure due to partition

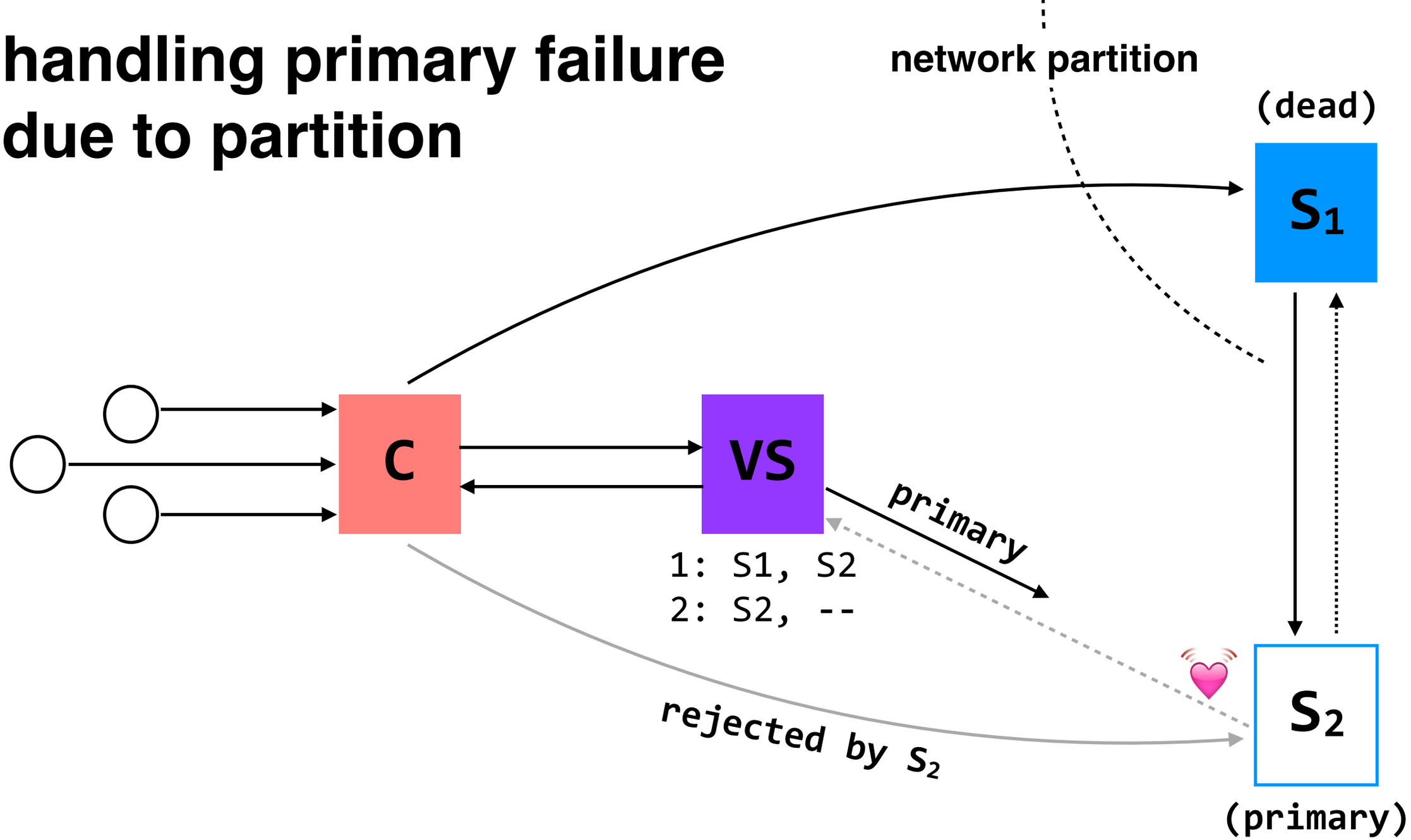


handling primary failure due to partition



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

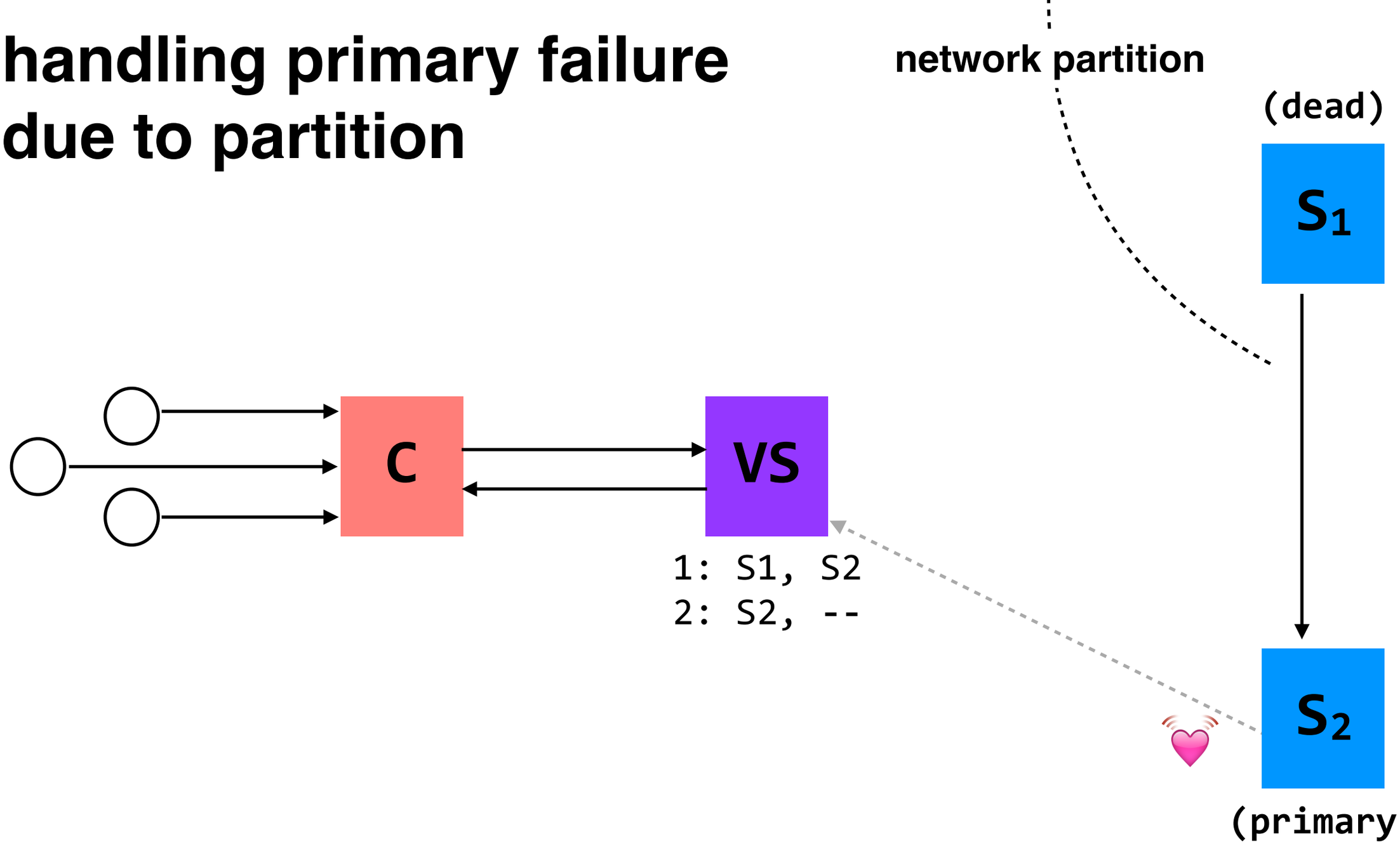
handling primary failure due to partition



it's okay! S_2 will act as backup

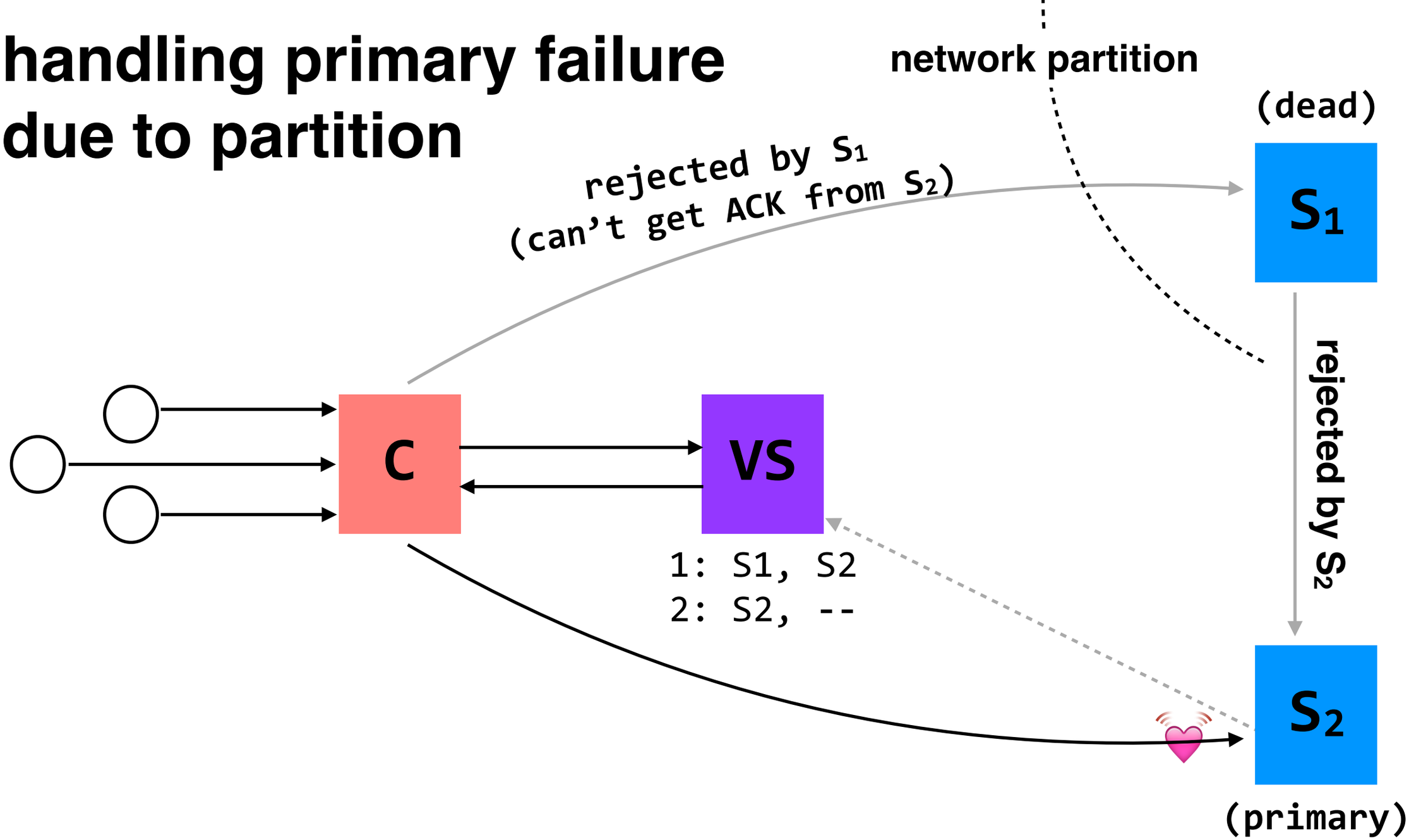
(accept updates from S_1, reject coordinator requests)

handling primary failure due to partition



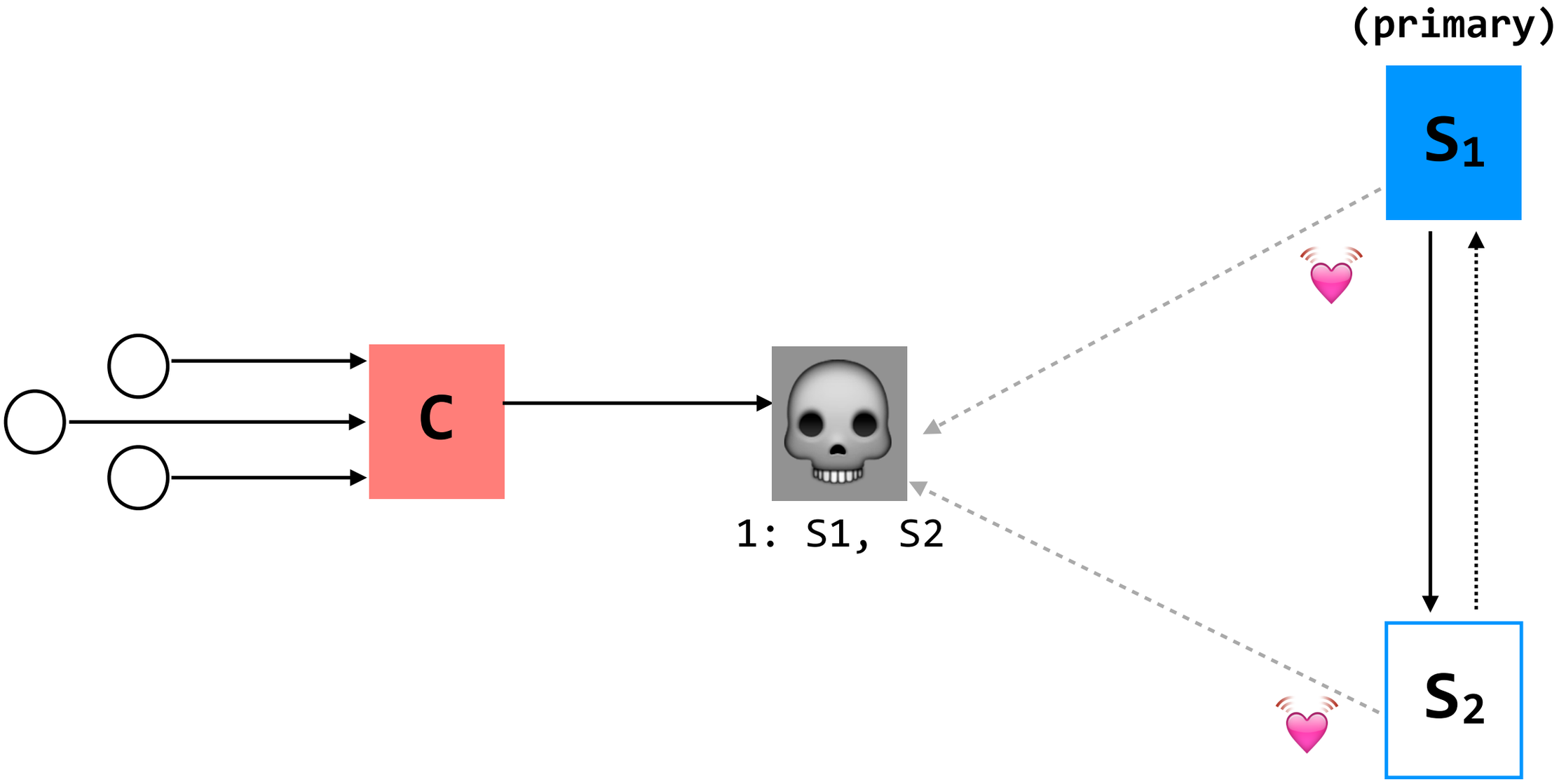
problem: what happens after S_2 knows it's the primary, but S_1 also thinks it is?

handling primary failure due to partition



also okay! 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).