SE 464
Week 8

Replication, Intro to Security

Availability via Replication

The following content is sourced from Computer Systems Design from MIT OCW
https://ocw.mit.edu/courses/6-033-computer-system-engineering-spring-2018/pages/week-11/

goal: build reliable systems from unreliable components

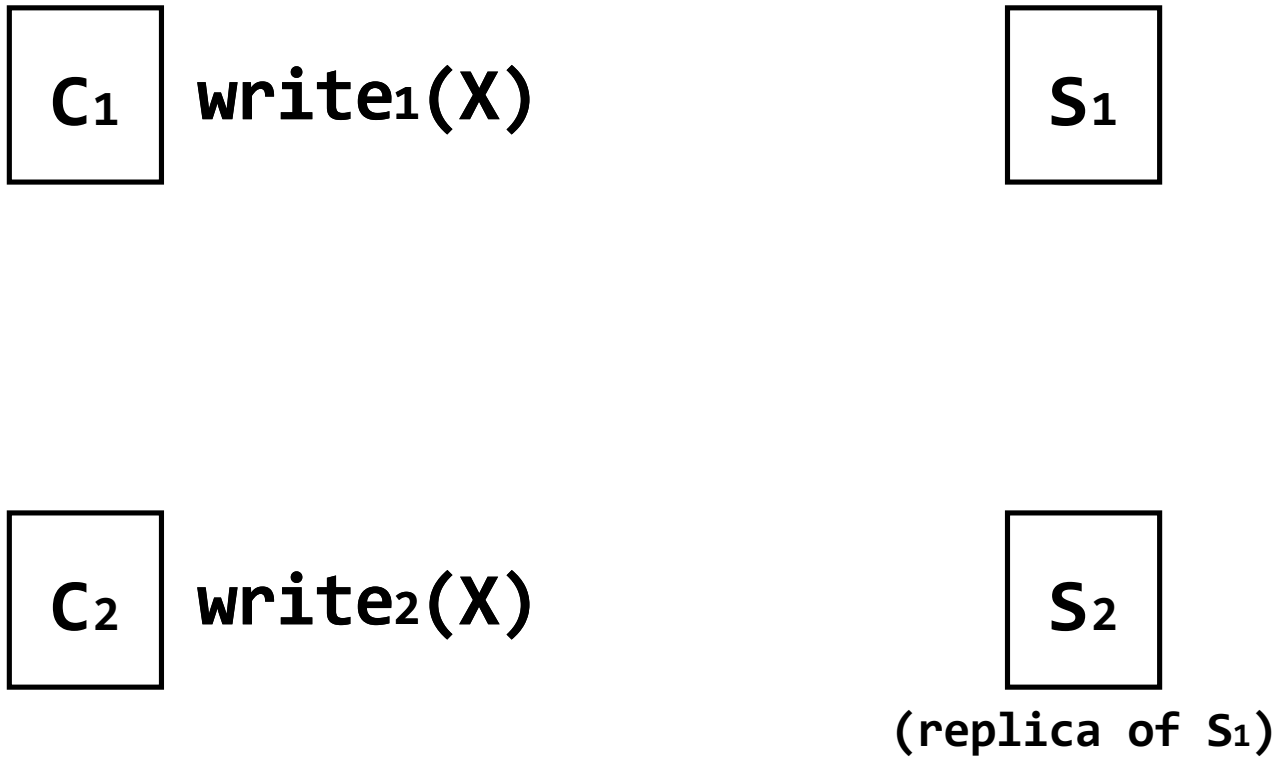
the abstraction that makes that easier is

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

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

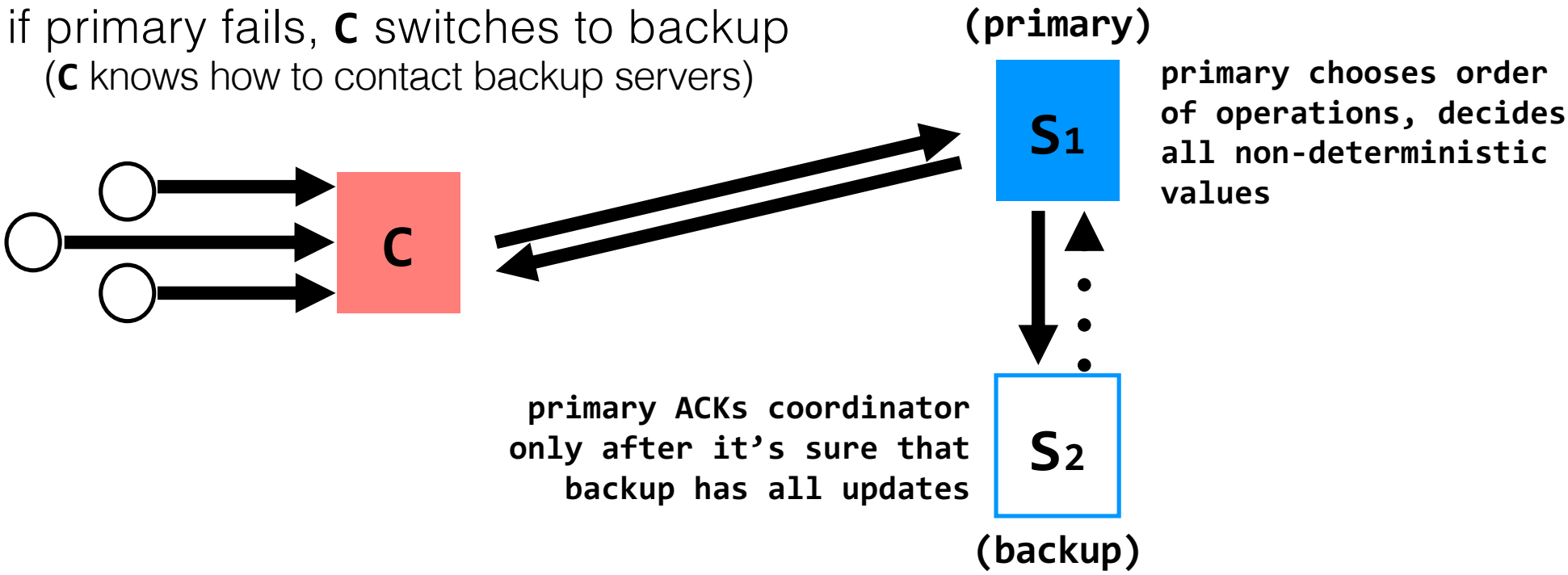
isolation → **two-phase locking**

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



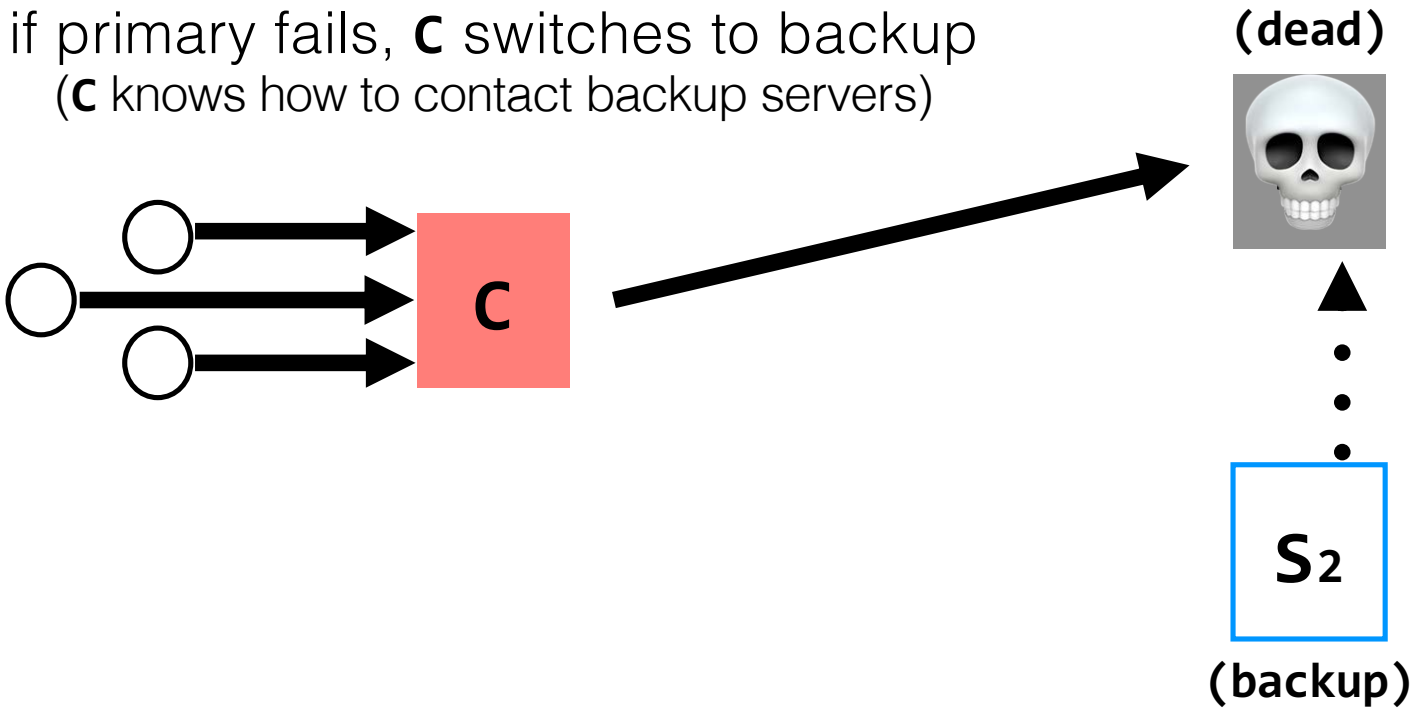
problem: replica servers can become inconsistent

if primary fails, **C** switches to backup
(**C** knows how to contact backup servers)



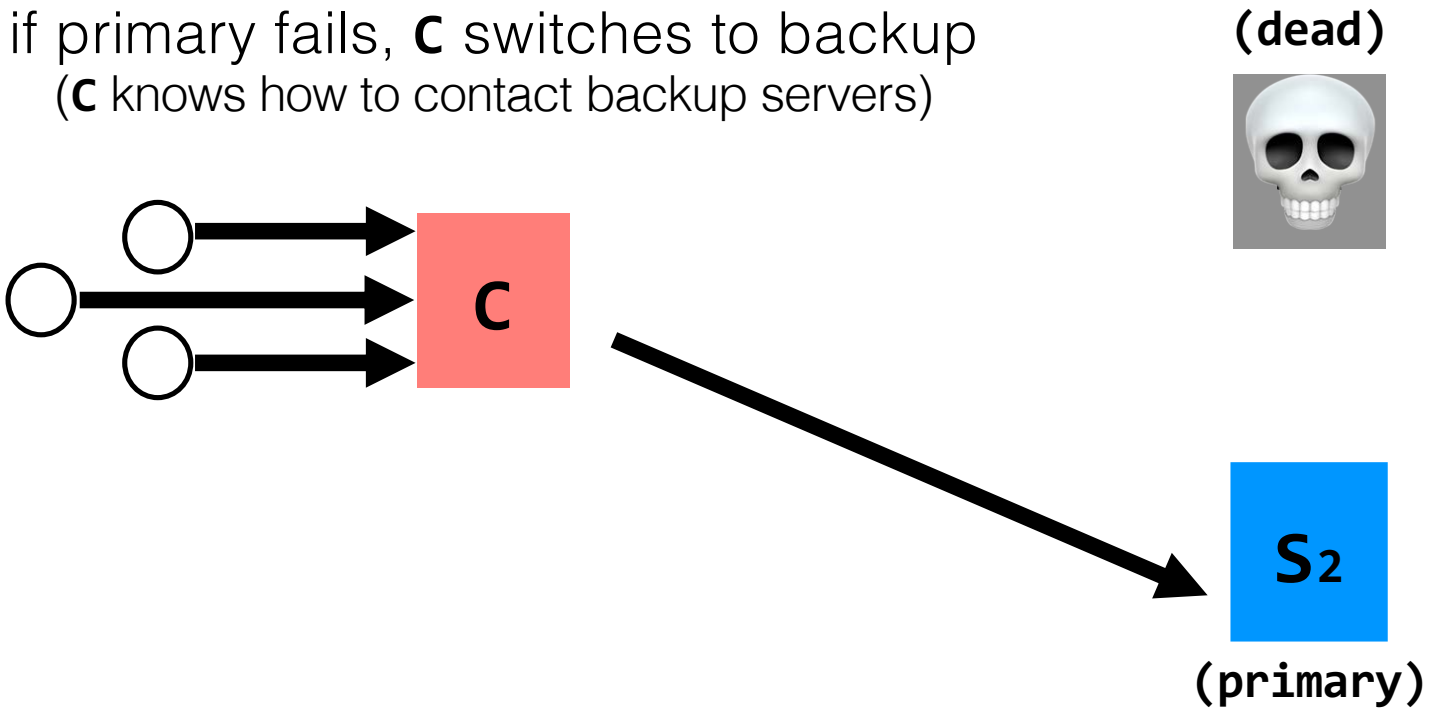
attempt: coordinators communicate with primary servers, who communicate with backup servers

if primary fails, **C** switches to backup
(**C** knows how to contact backup servers)



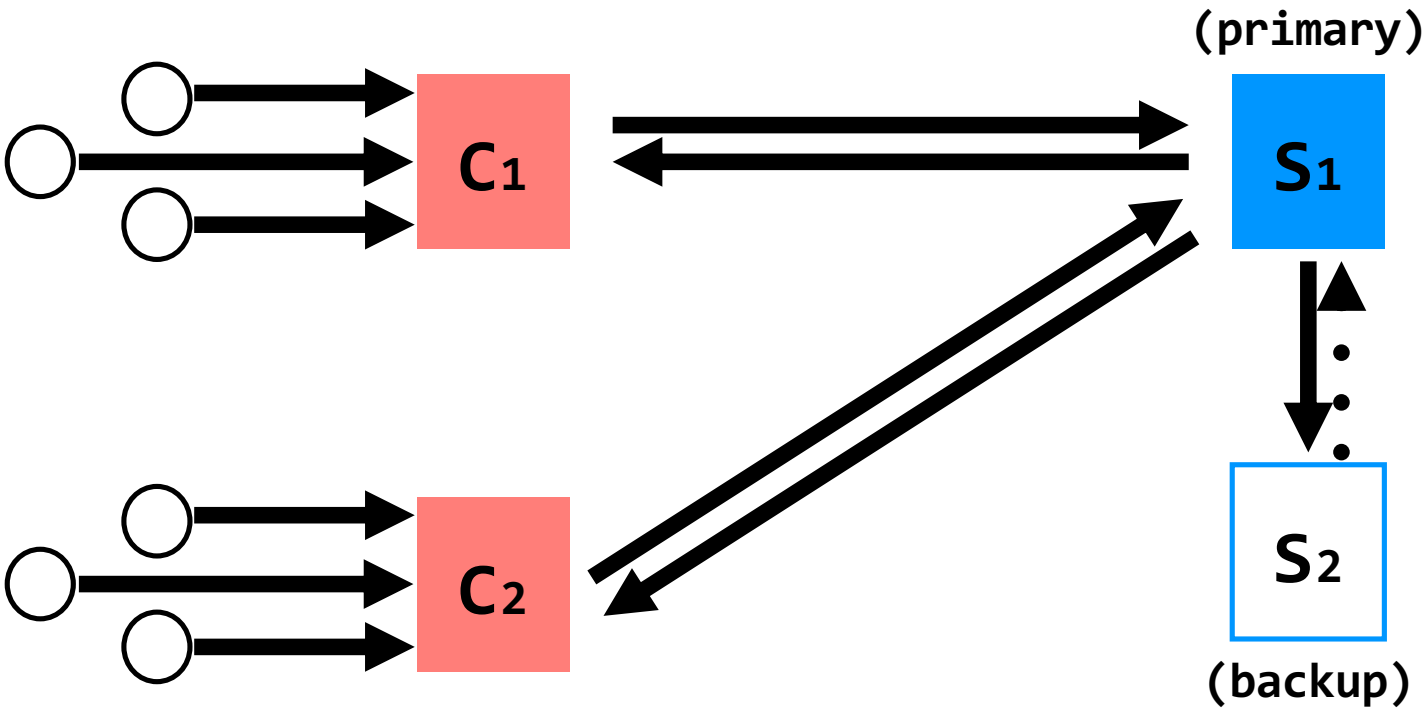
attempt: coordinators communicate with primary servers, who communicate with backup servers

if primary fails, **C** switches to backup
(**C** knows how to contact backup servers)



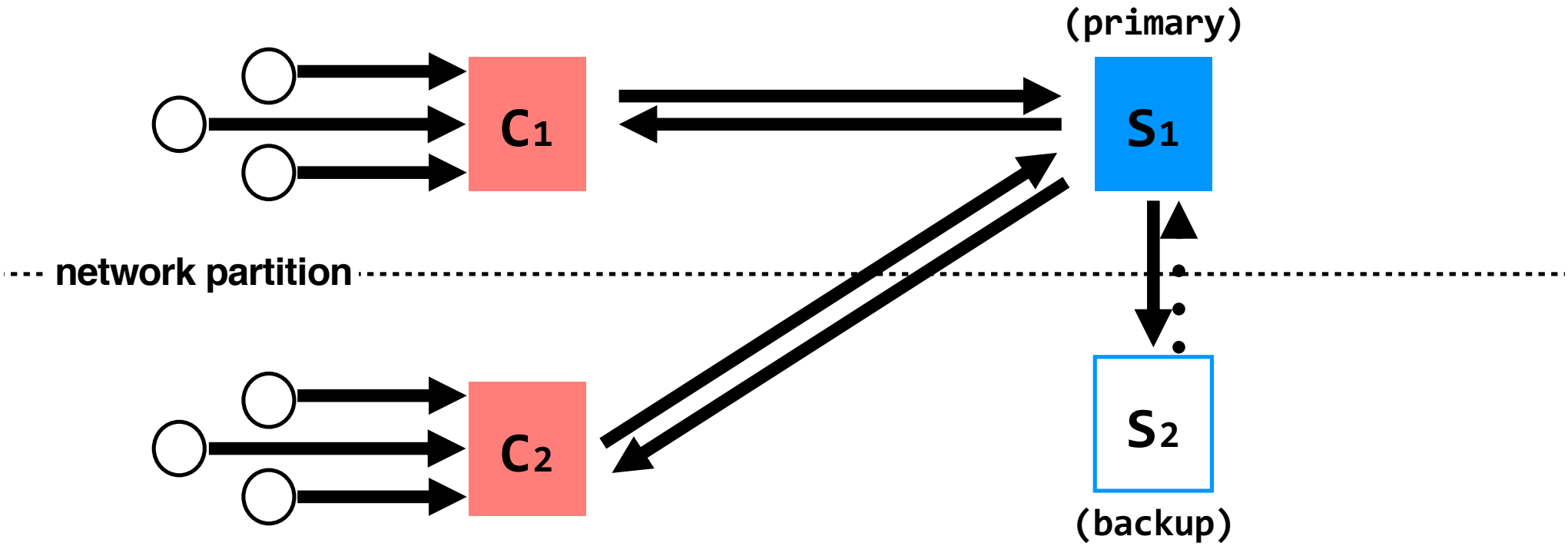
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems



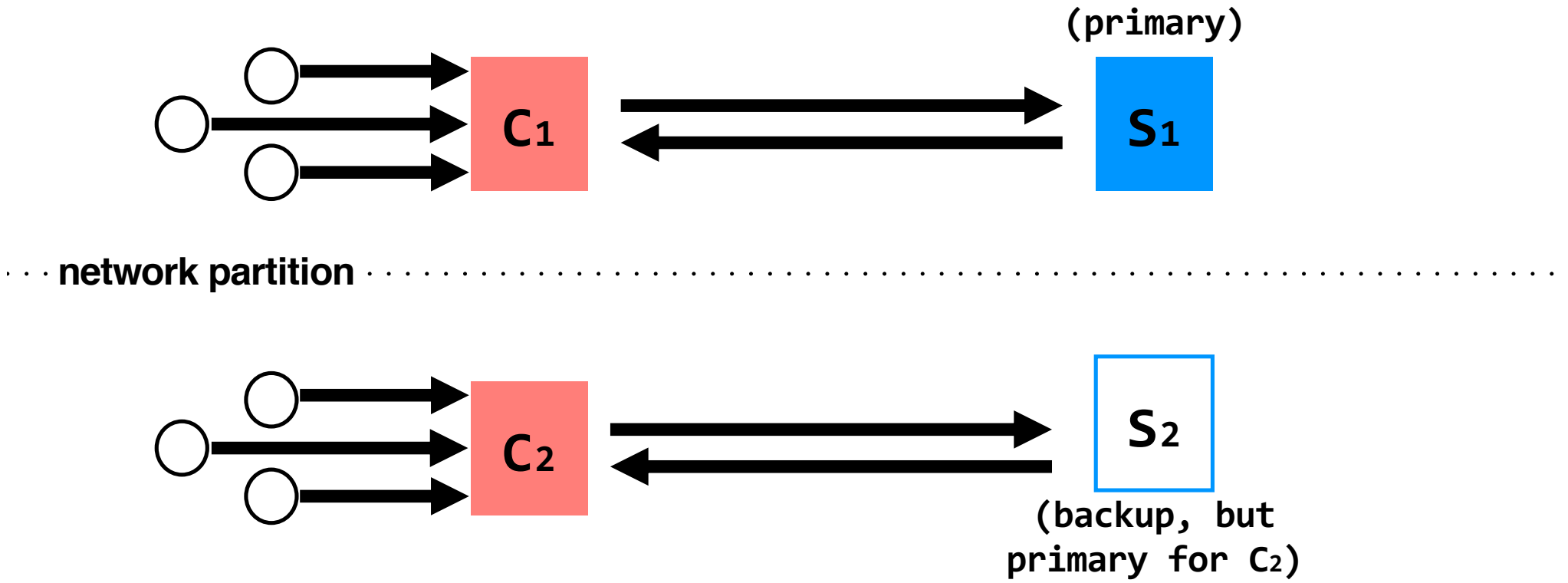
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems



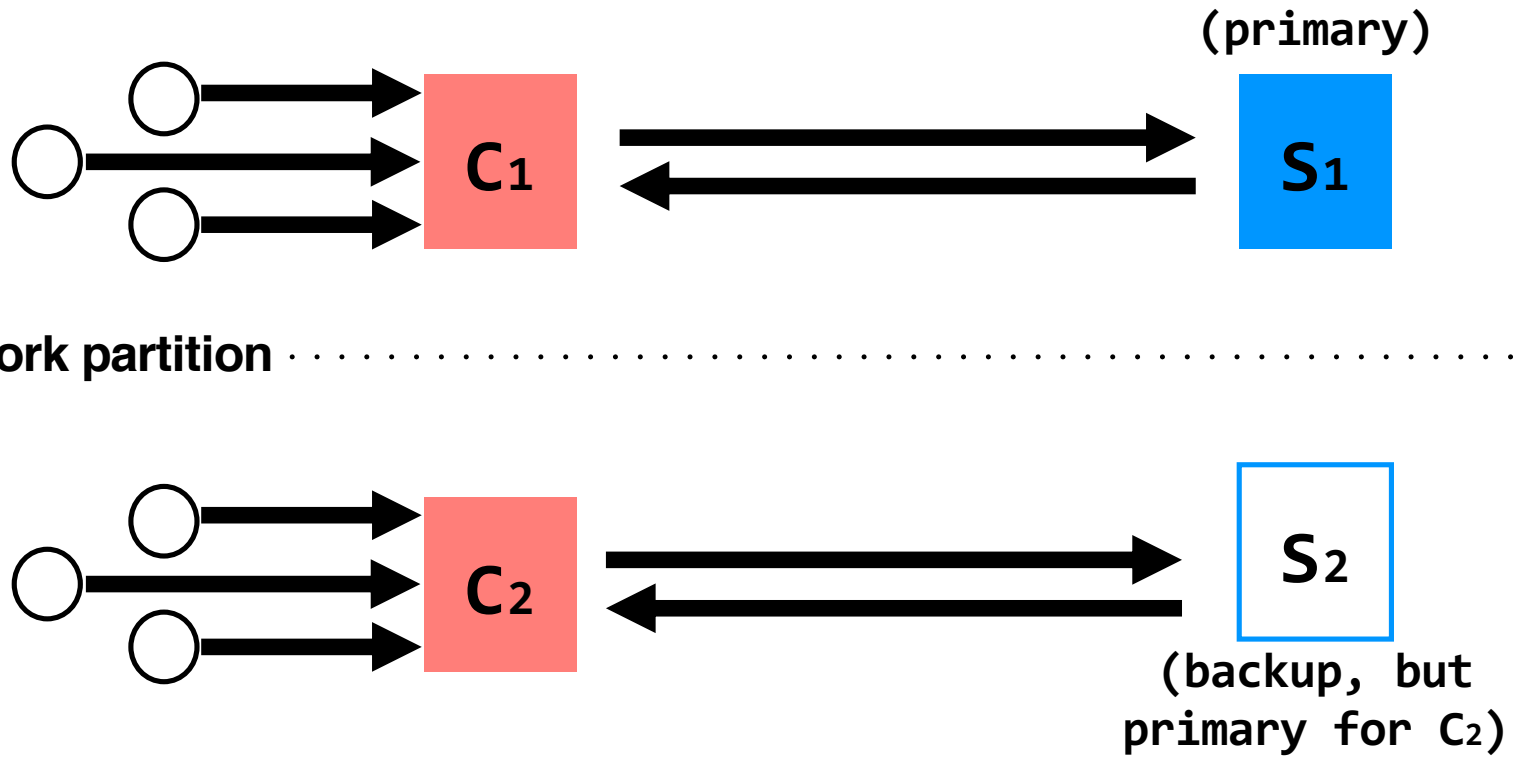
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems



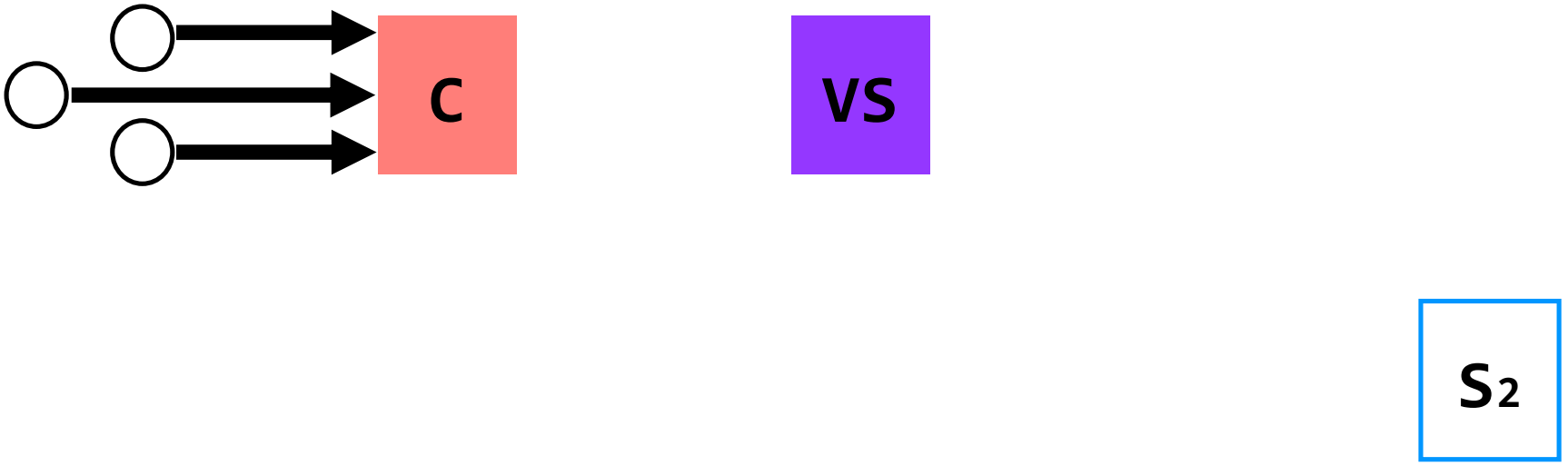
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems

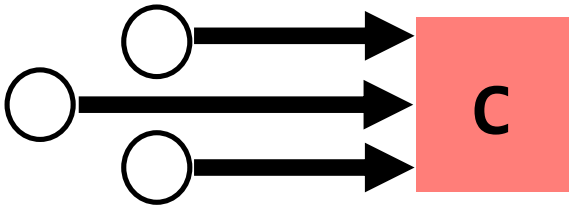


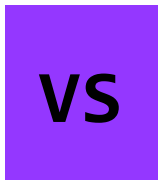
C1 and **C2** are using different primaries;
S1 and **S2** are no longer consistent

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



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

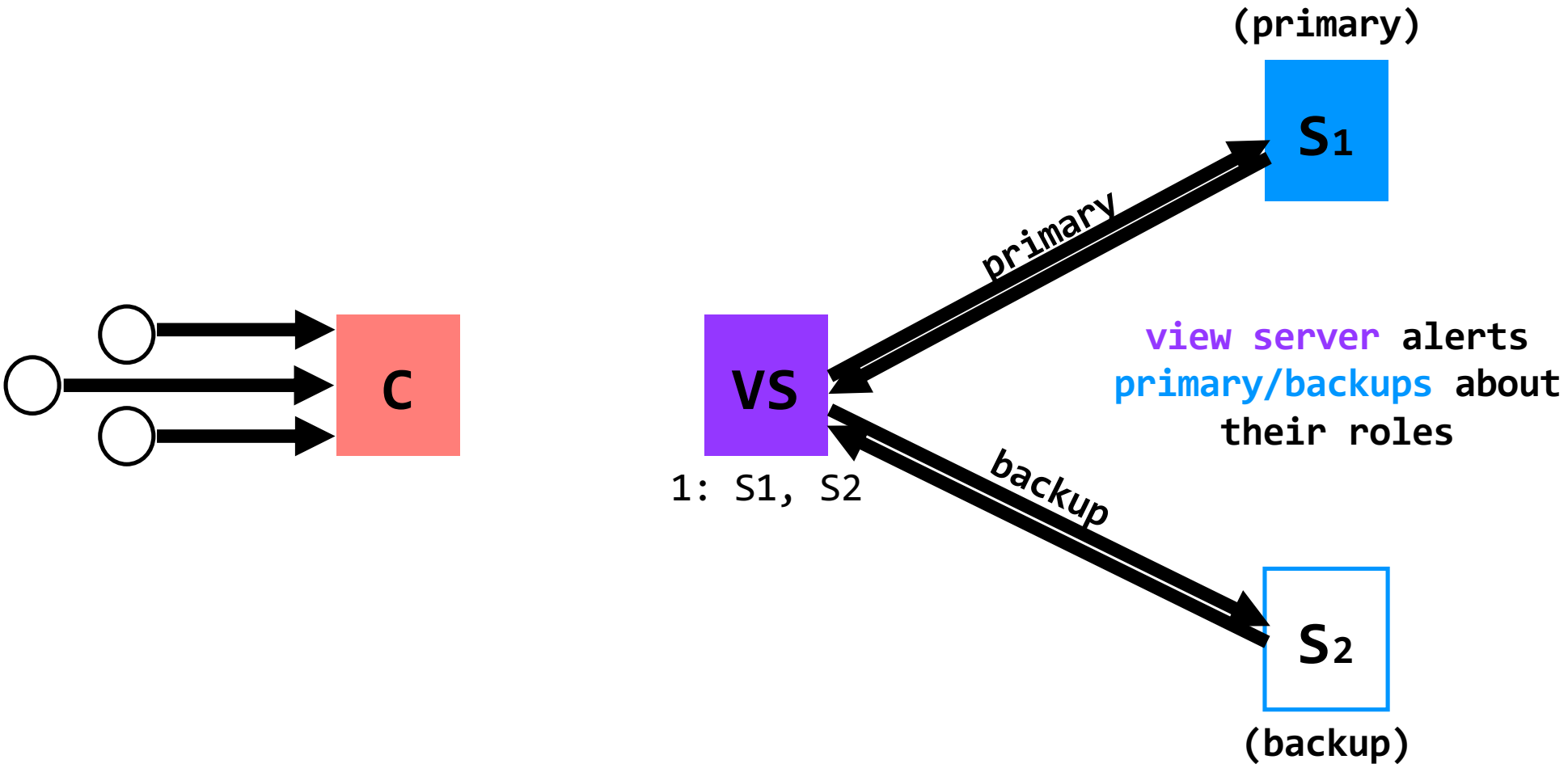




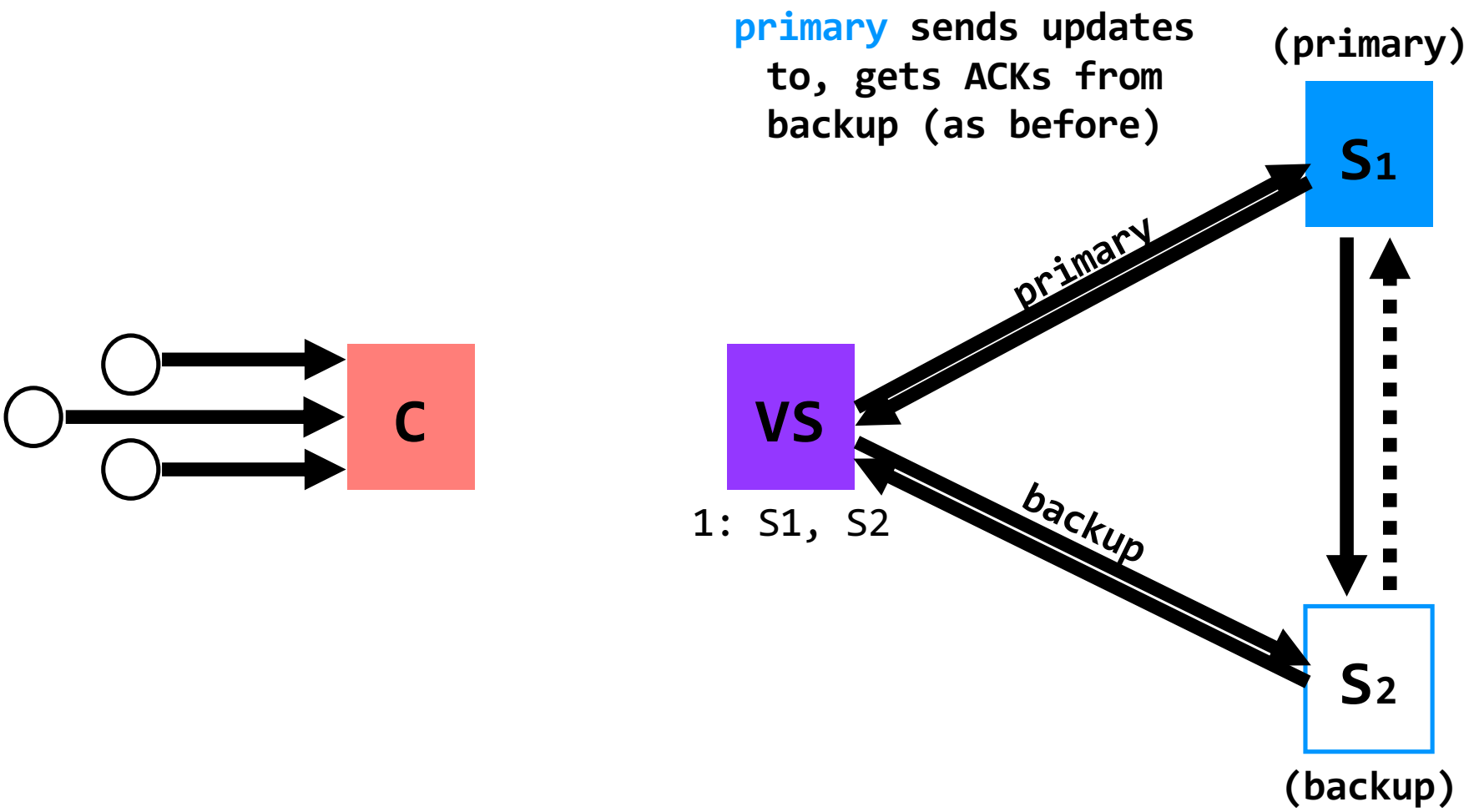
1: S1, S2

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

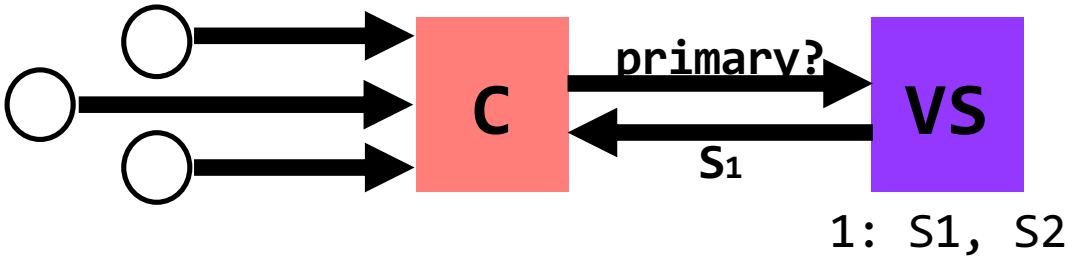
use a **view server**, which determines which replica is the primary



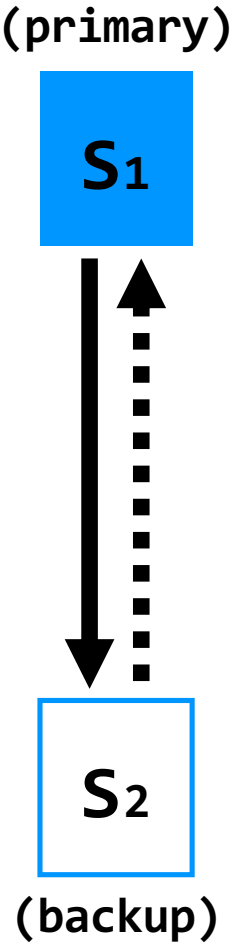
use a **view server**, which determines which replica is the primary



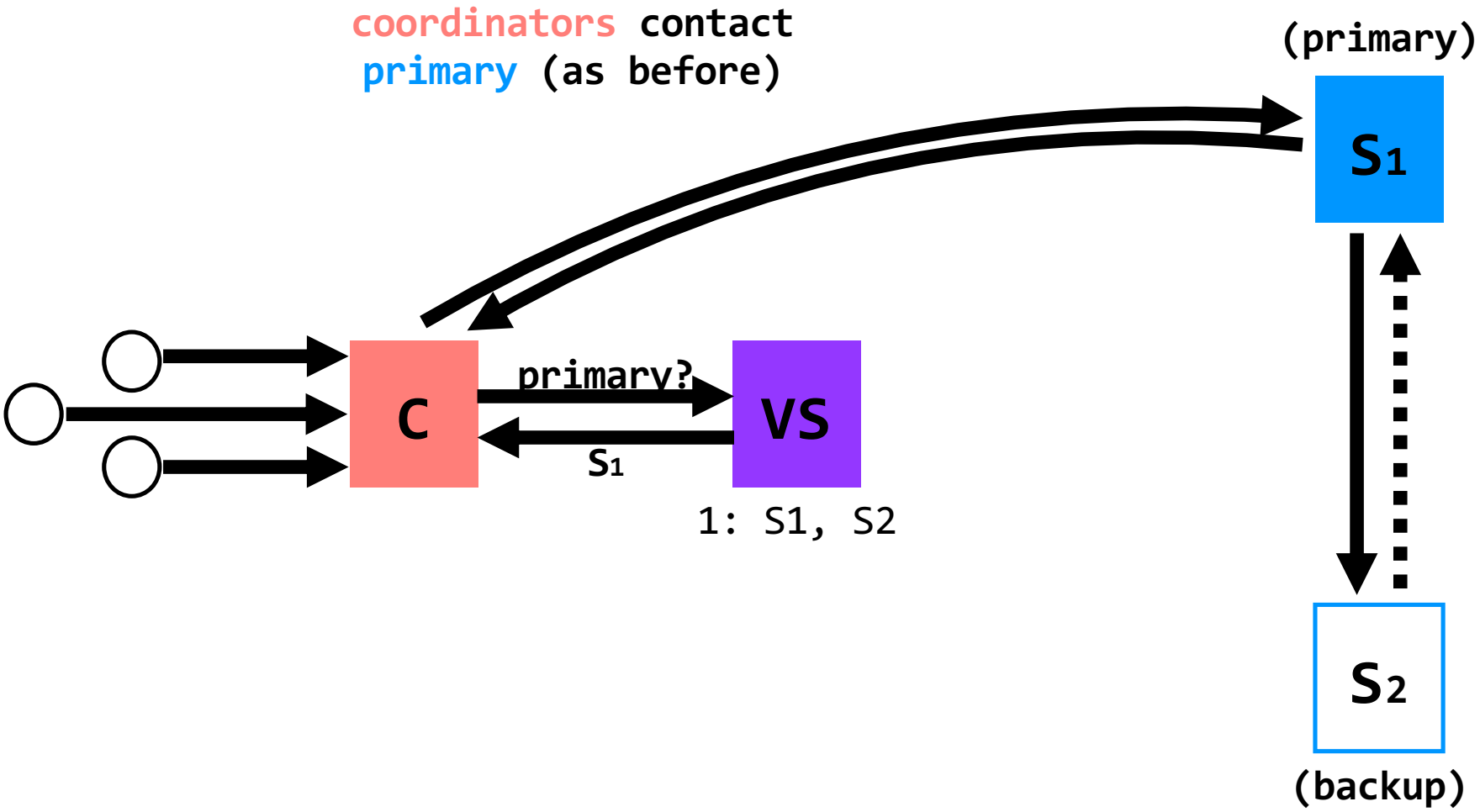
use a **view server**, which determines which replica is the primary



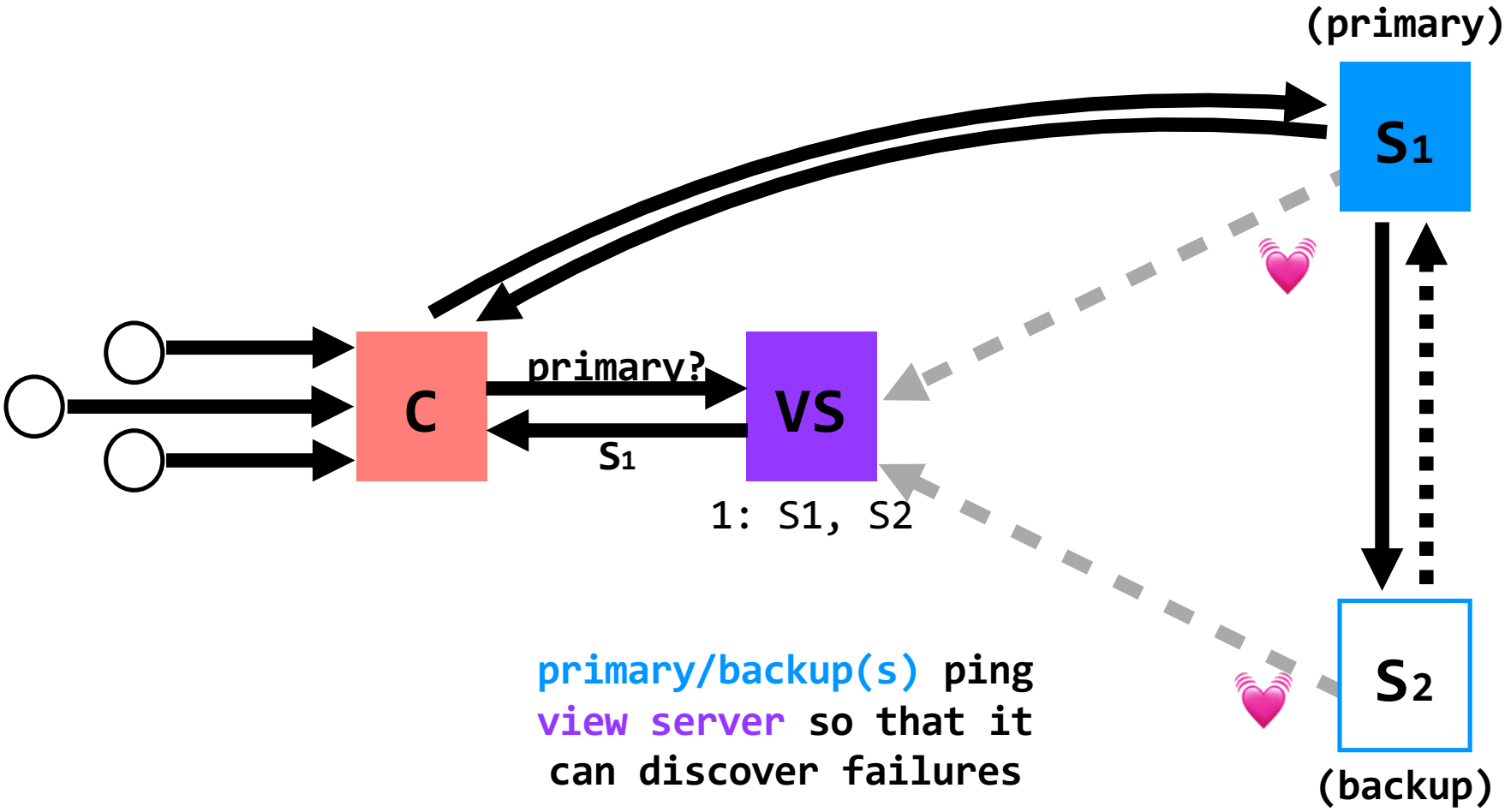
coordinators make requests to view server to find out who is primary



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

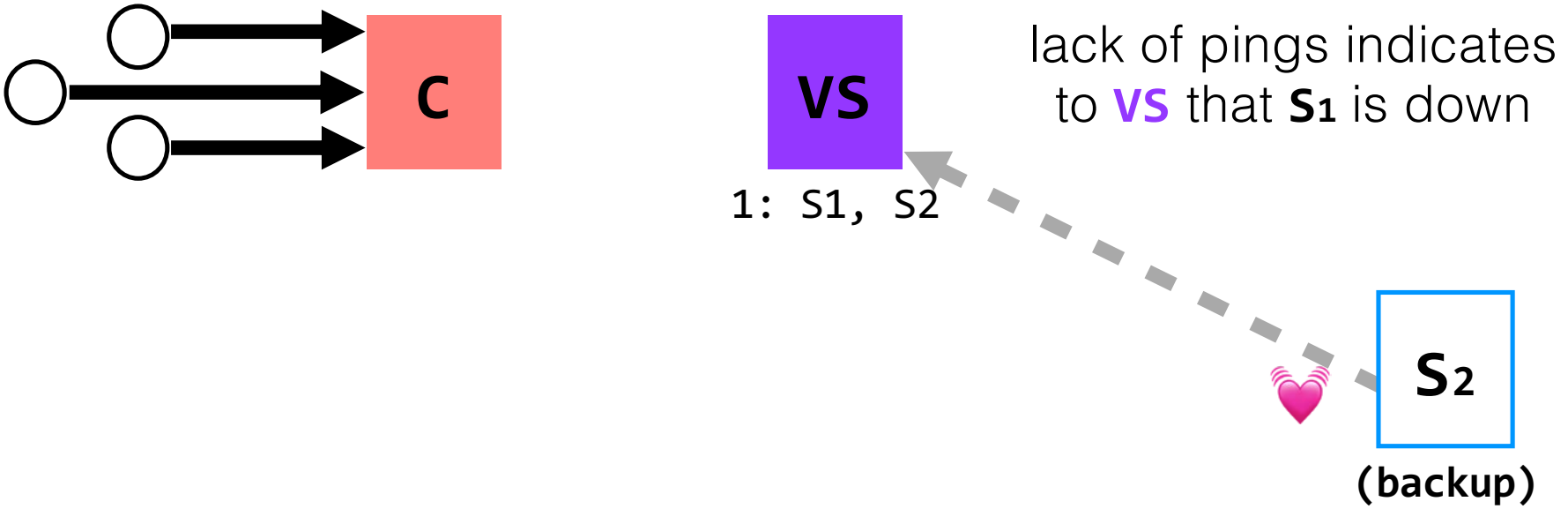


use a **view server**, which determines which replica is the primary

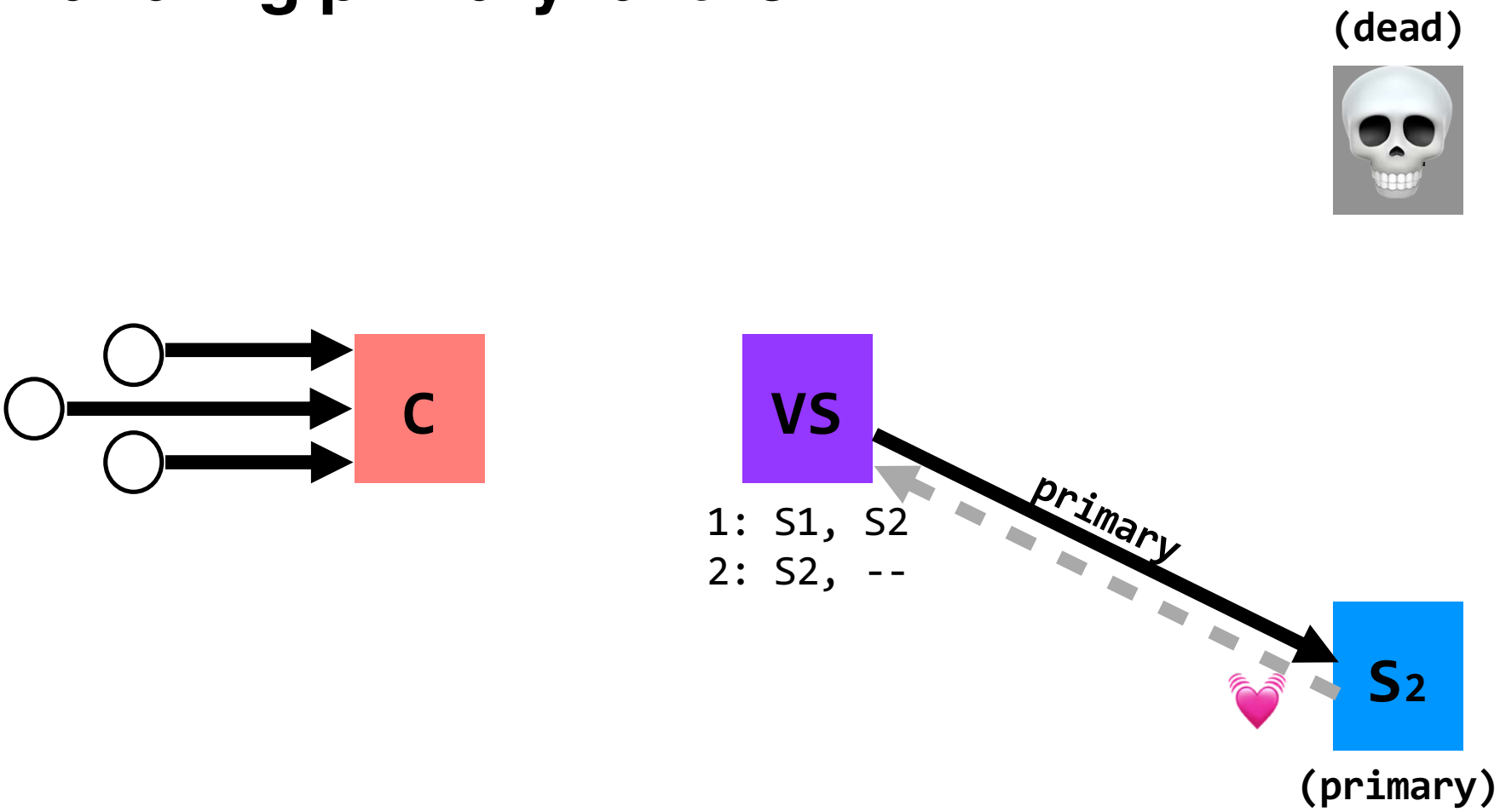


use a **view server**, which determines which replica is the primary

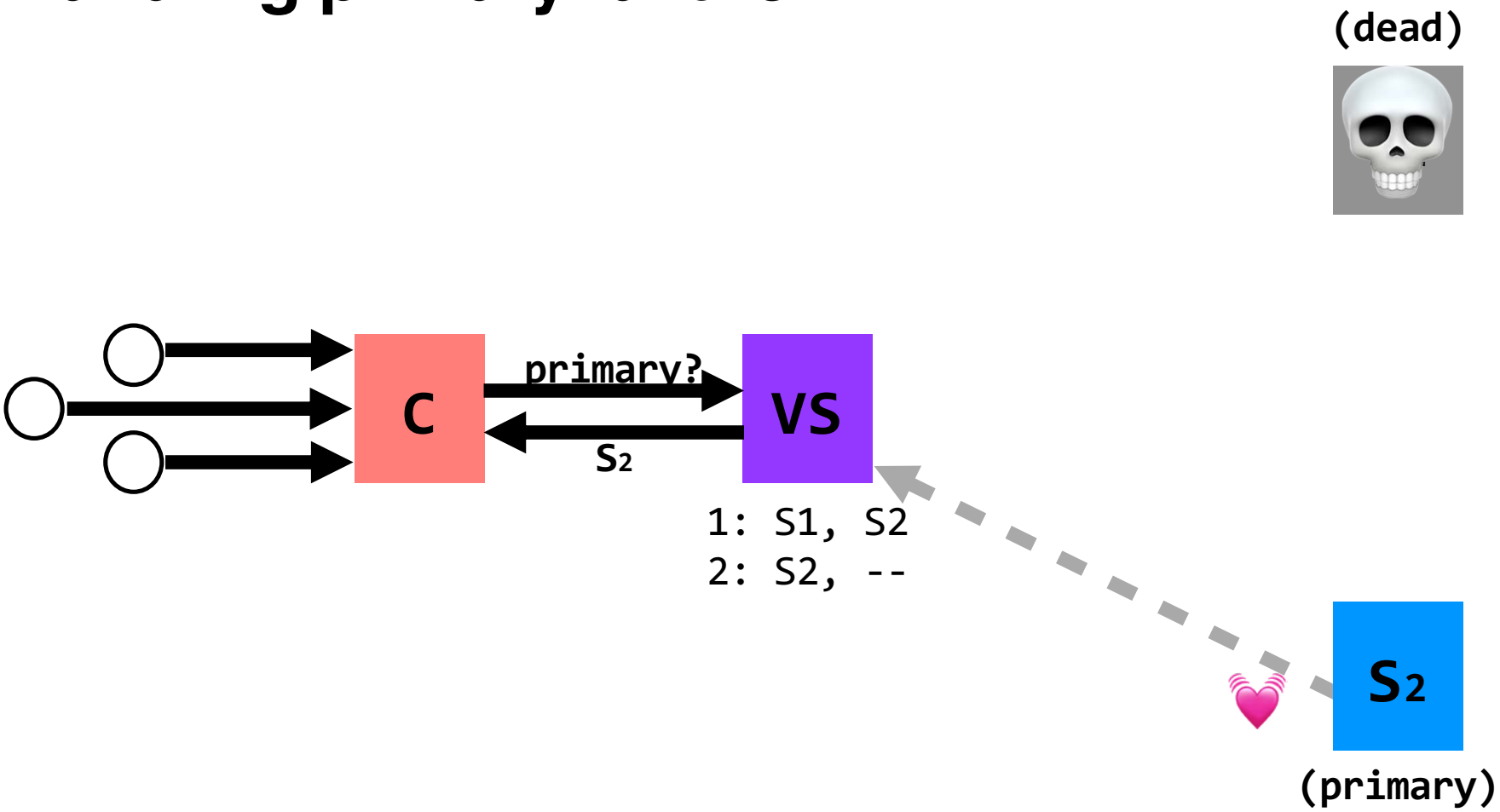
handling primary failure



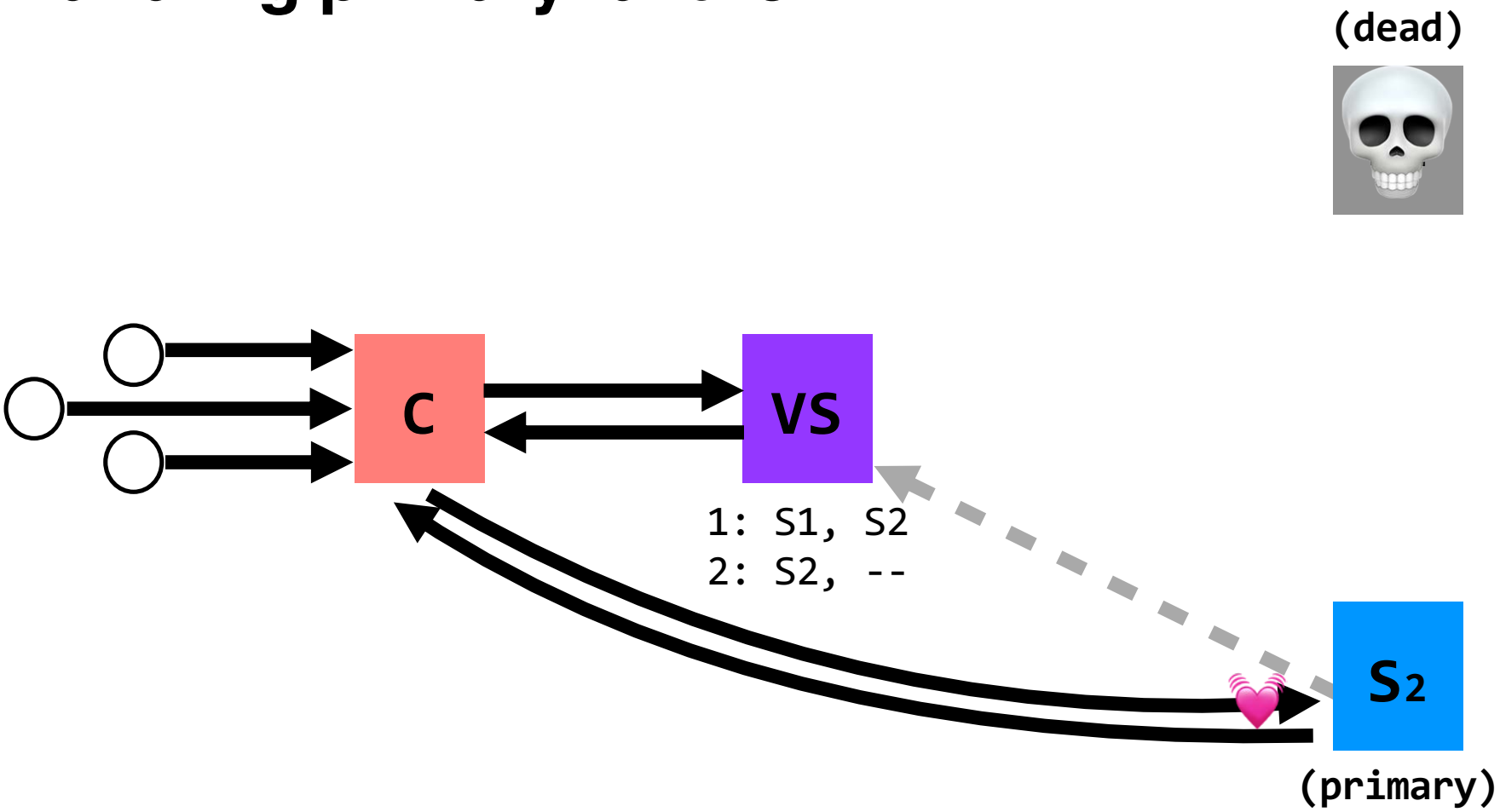
handling primary failure



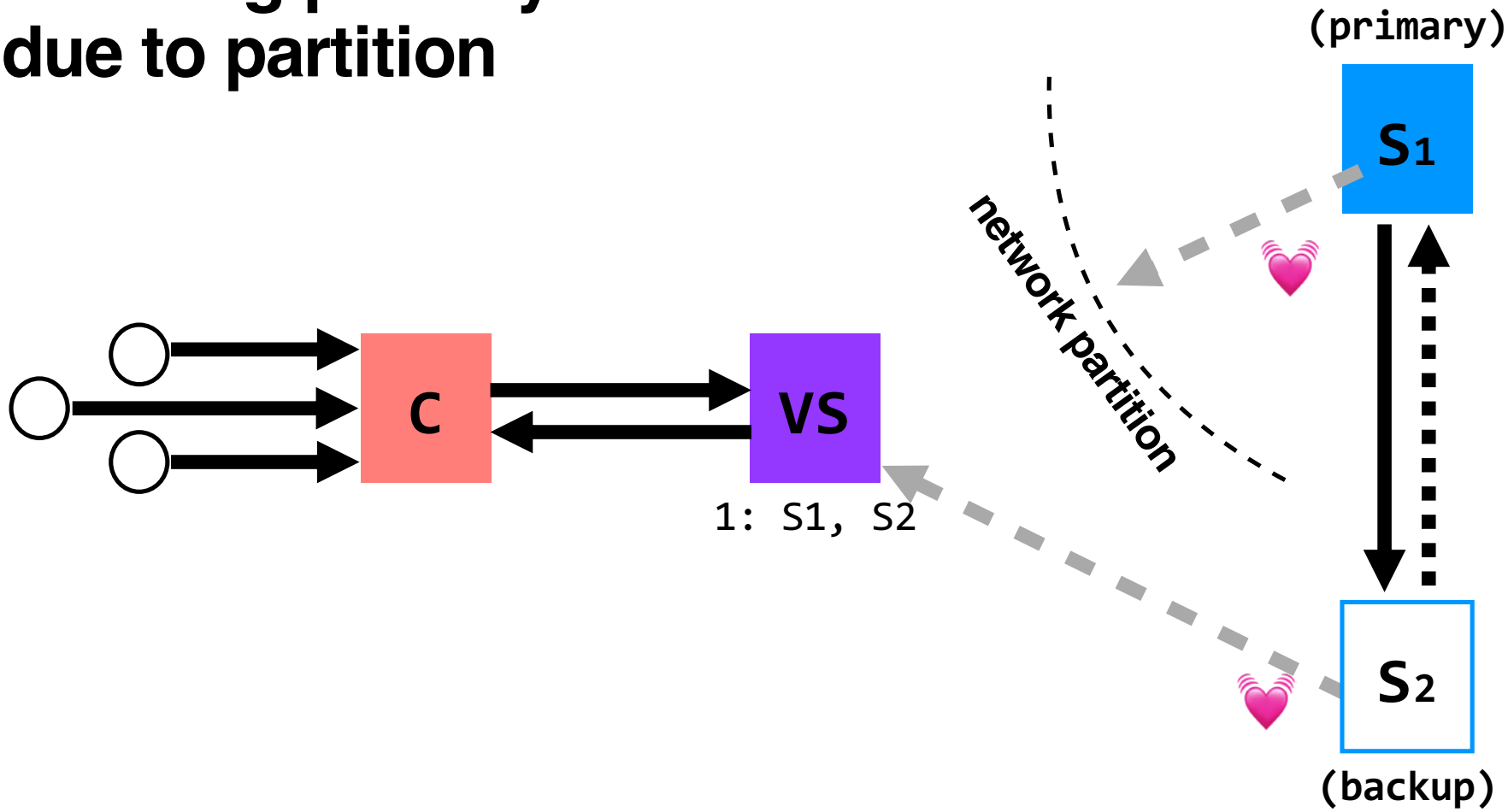
handling primary failure



handling primary failure

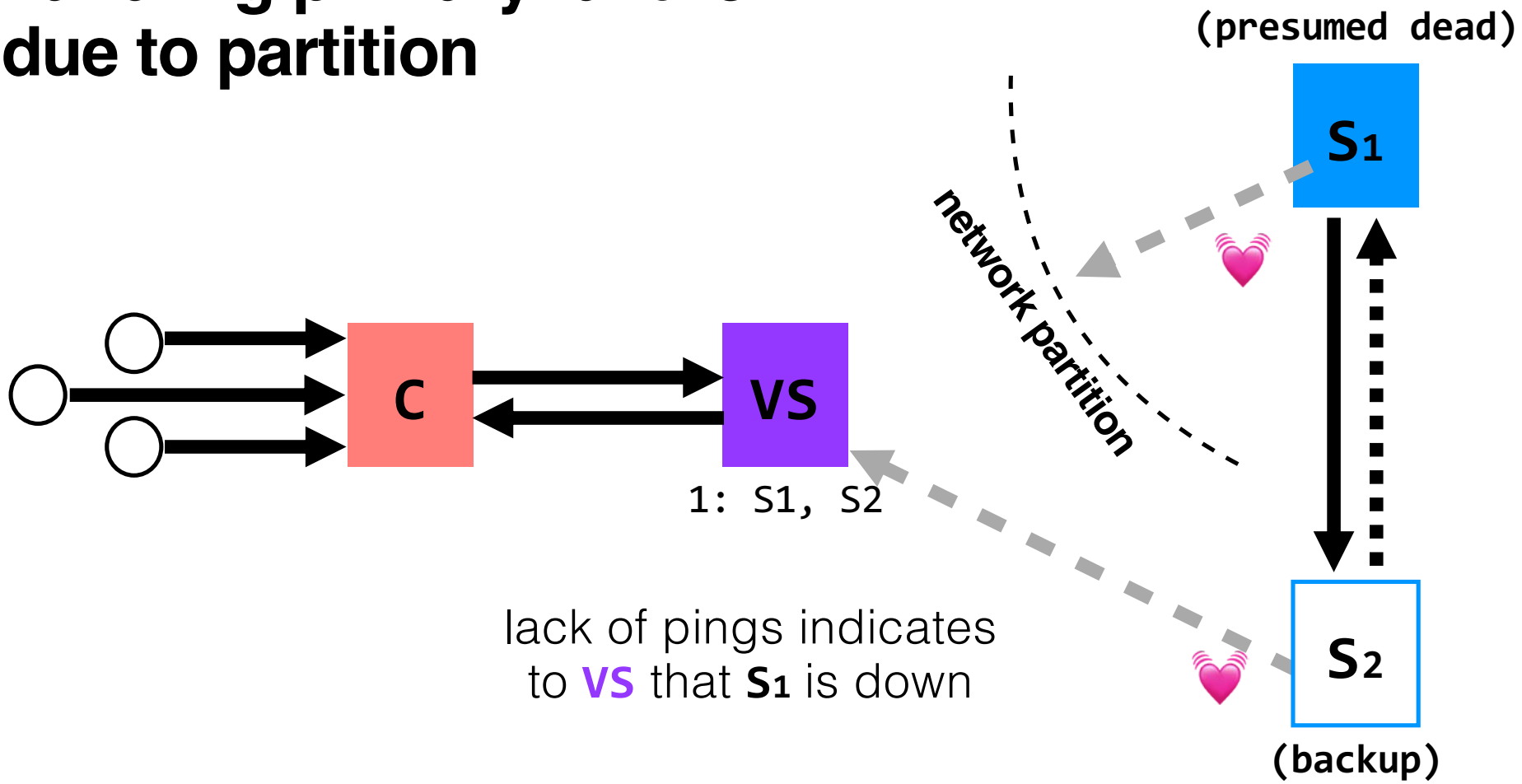


handling primary failure due to partition

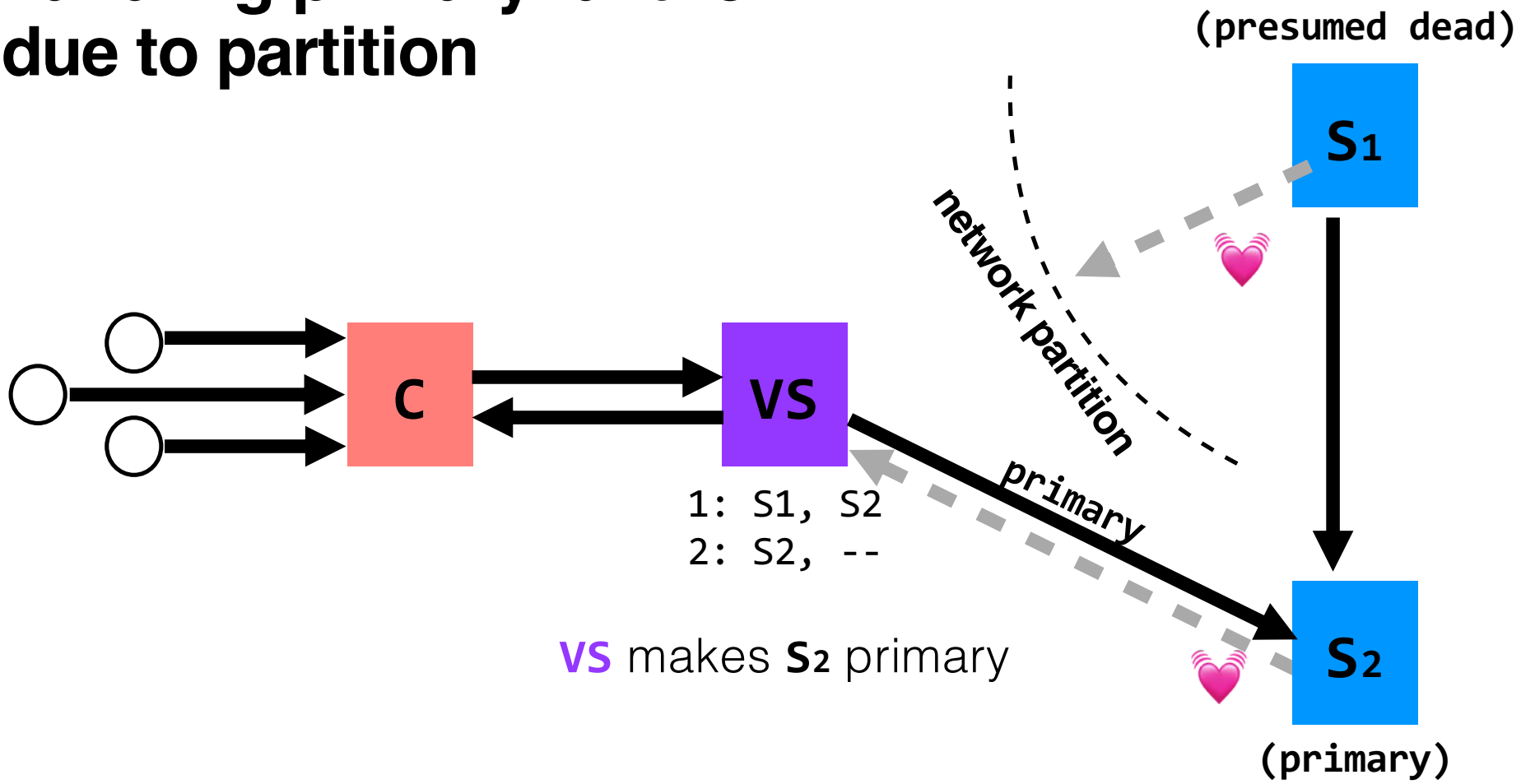


pose a partition keeps S1 from communicating with the view se

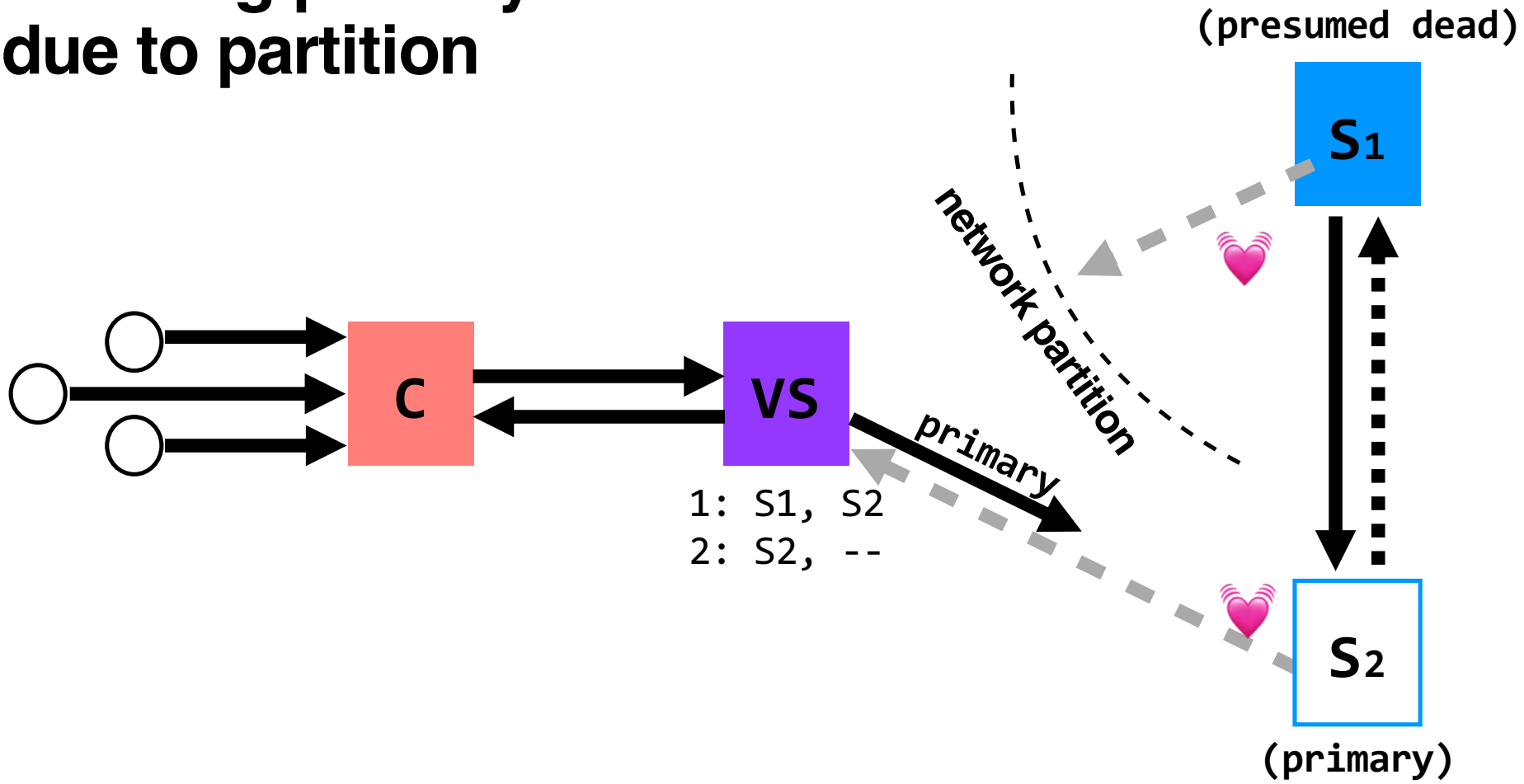
handling primary failure due to partition



handling primary failure due to partition

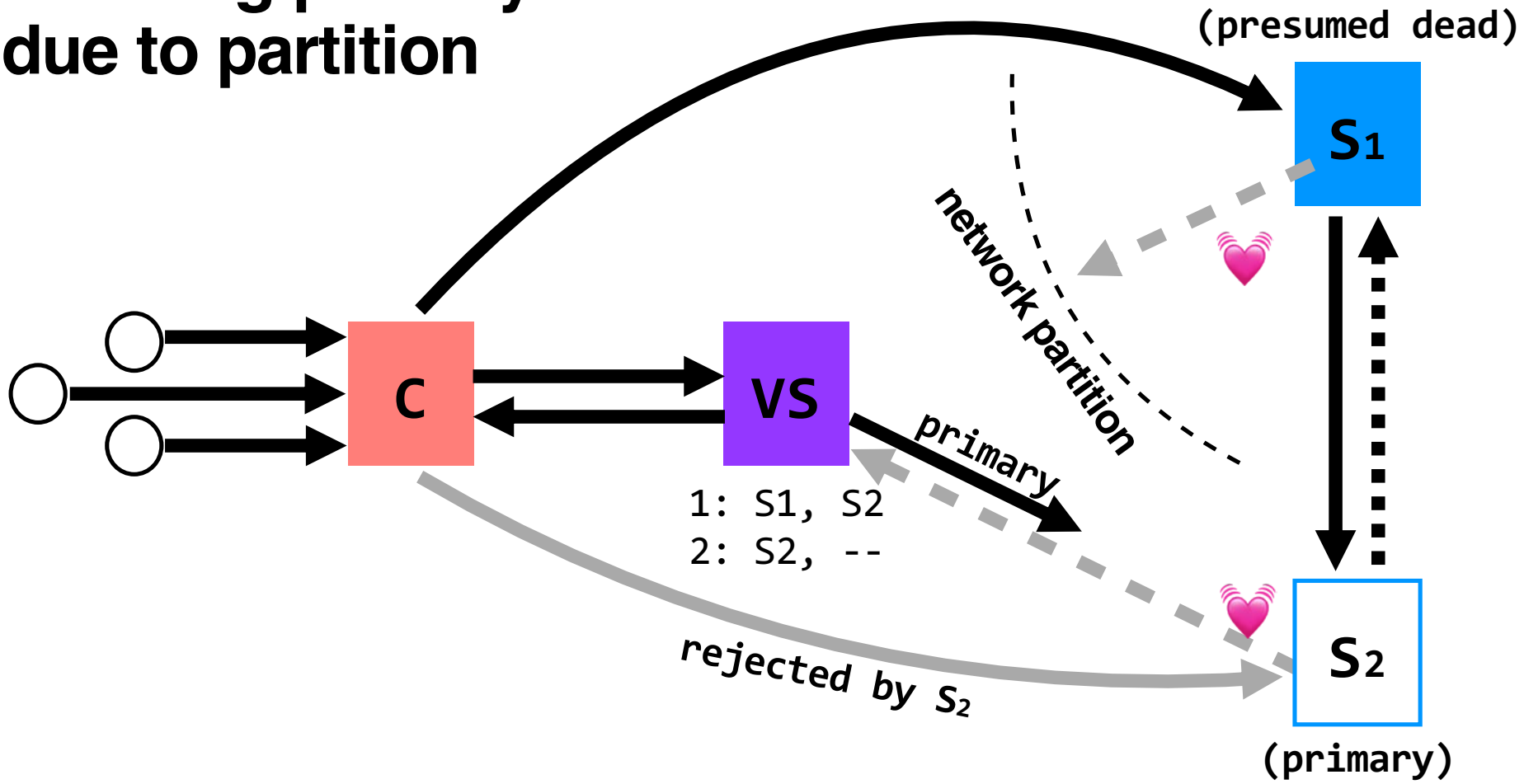


handling primary failure due to partition



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

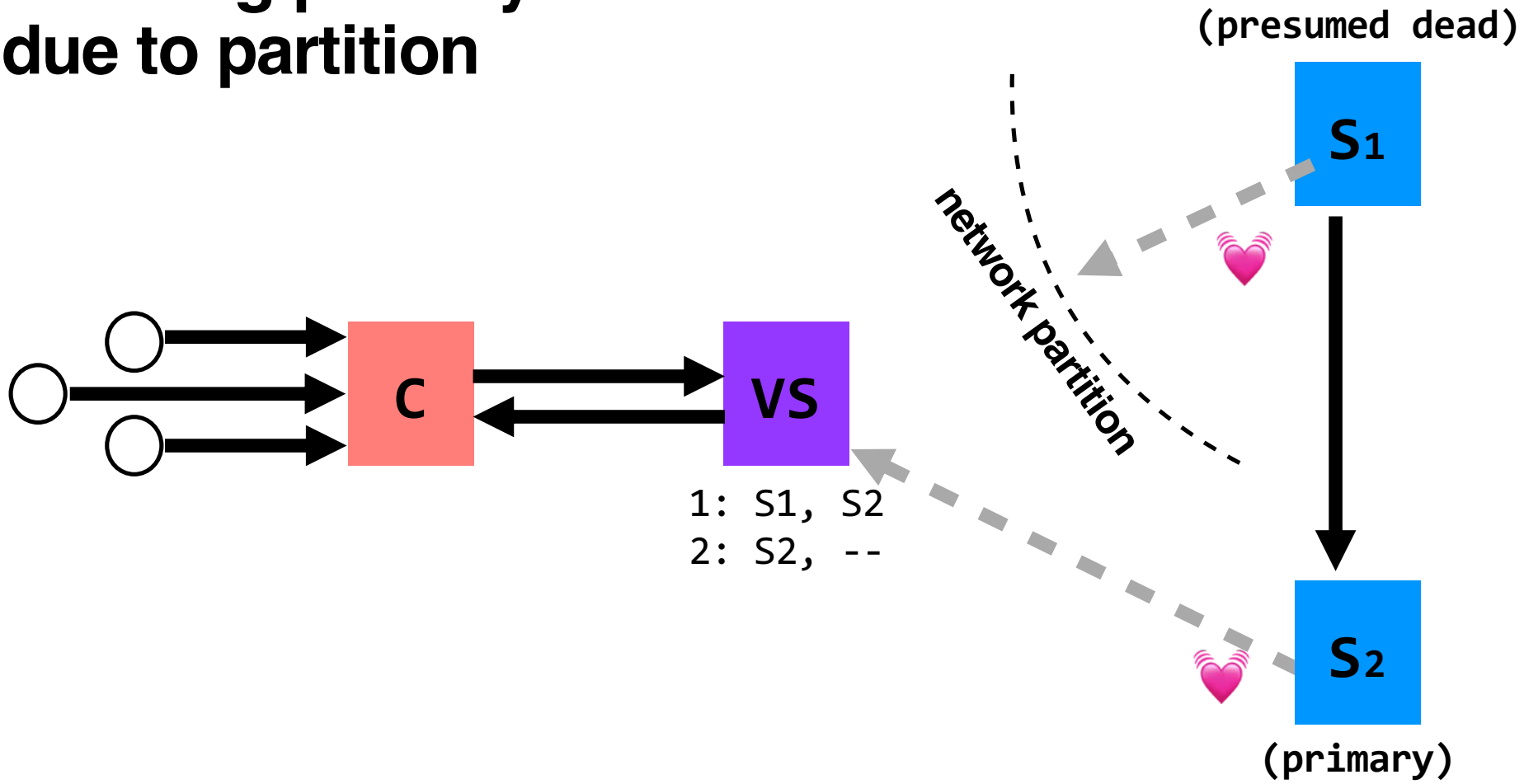
handling primary failure due to partition



S2 will act as backup

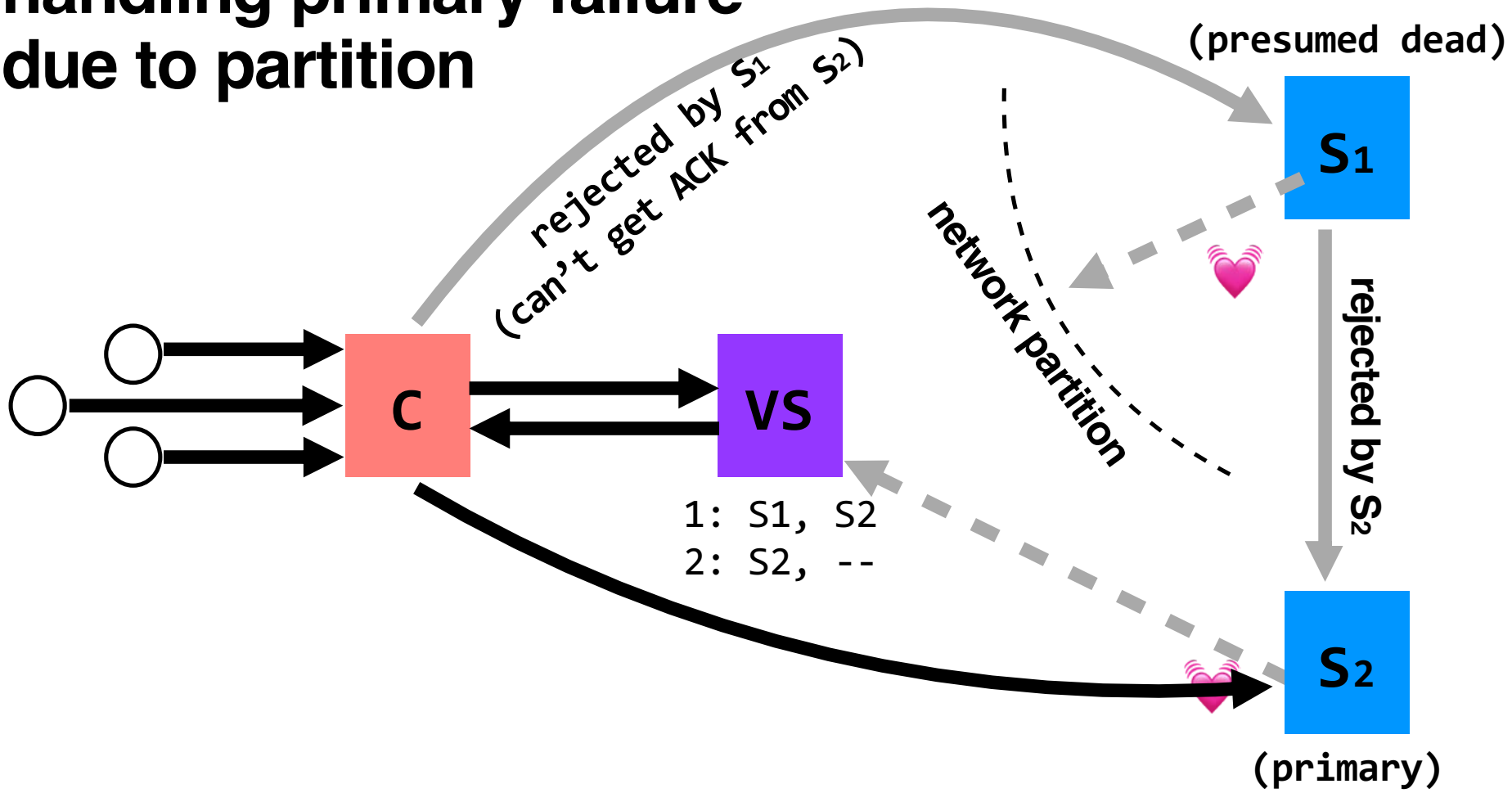
(accept updates from S1, reject coordinator requests)

handling primary failure due to partition



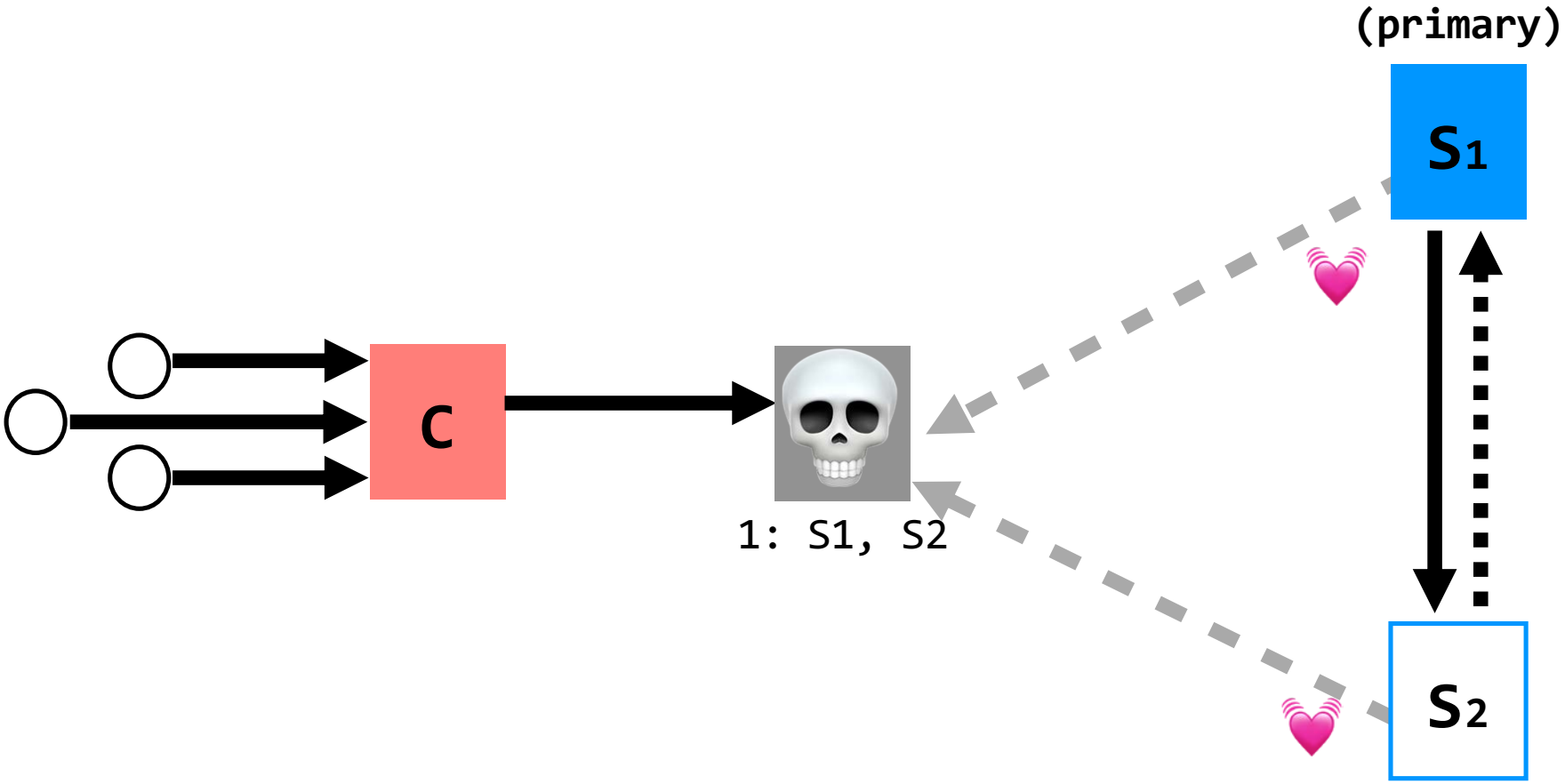
question: what happens after S_2 knows it's the primary, but S_1 also thinks it is?

handling primary failure due to partition

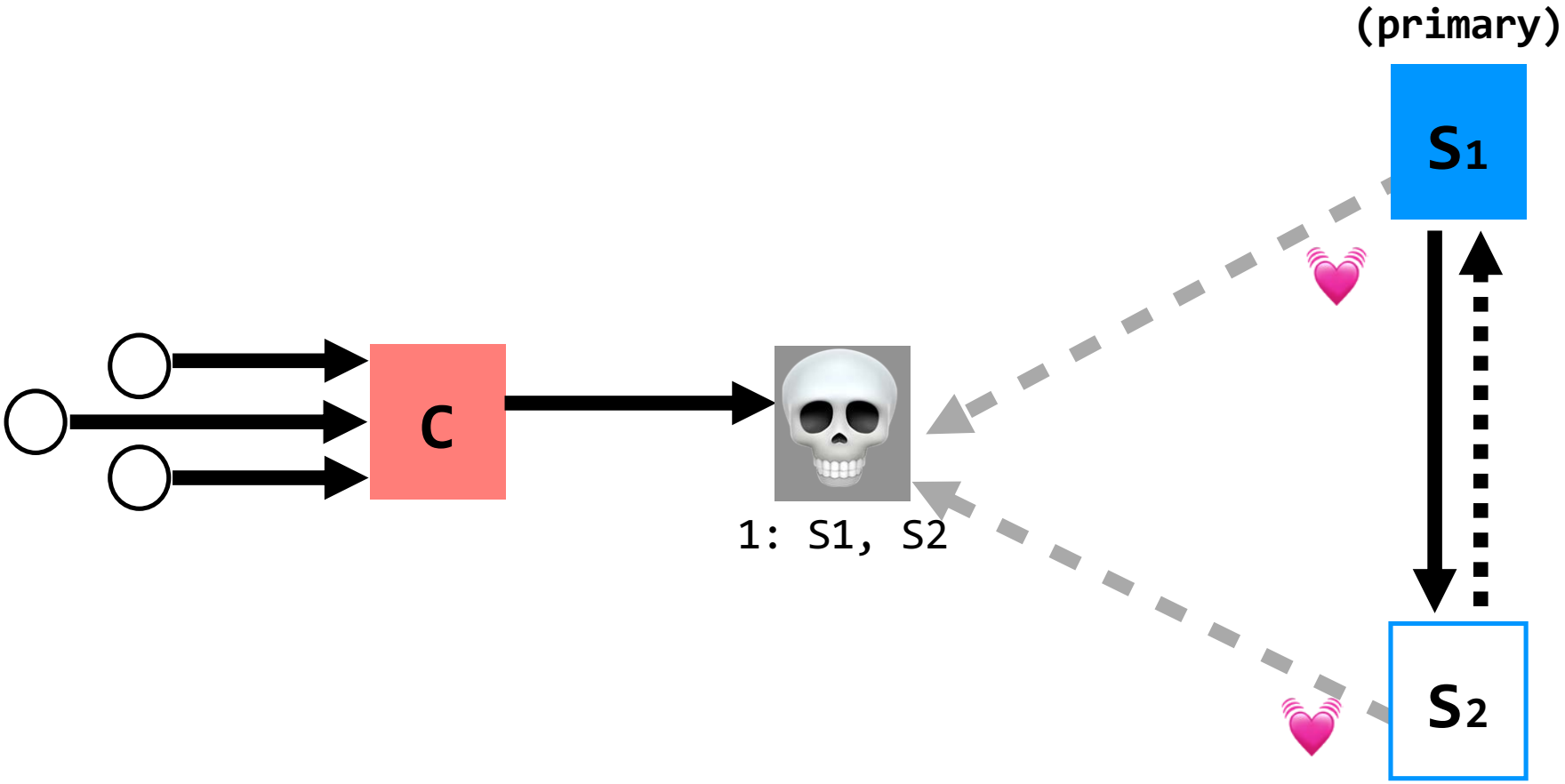


S_1 won't be able to act as primary

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



problem: what if view server fails?



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).

Intro to Security

The following content is sourced from Computer Systems Design from MIT OCW
https://ocw.mit.edu/courses/6-033-computer-system-engineering-spring-2018/pages/week-11/

Suspicious event hijacks Ama ×

Katrina

Secure | https://arstechnica.com/information-technology/2018/04/suspicious-event-hijacks-amazon-traffic-for-2-hours-steals-cryptocurren...

ars TECHNICA

BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE FORUMS SUBSCRIPTIONS SIGN IN

BORDER GATEWAY PROTOCOL ATTACK —

Suspicious event hijacks Amazon traffic for 2 hours, steals cryptocurrency

Almost 1,300 addresses for Amazon Route 53 rerouted for two hours.

DAN GOODIN - 4/24/2018, 3:00 PM

amazon.com®

Amazon

108

Amazon lost control of a small number of its cloud services IP addresses for two hours on Tuesday morning when hackers exploited a known Internet-protocol weakness that let them to redirect traffic to rogue destinations. By subverting Amazon's domain-resolution service, the attackers masqueraded as cryptocurrency website MyEtherWallet.com and stole about

RISK ASSESSMENT —

Yahoo says half a billion accounts breached by nation-sponsored hackers

One of the biggest compromises ever exposes names, e-mail addresses, and much more.

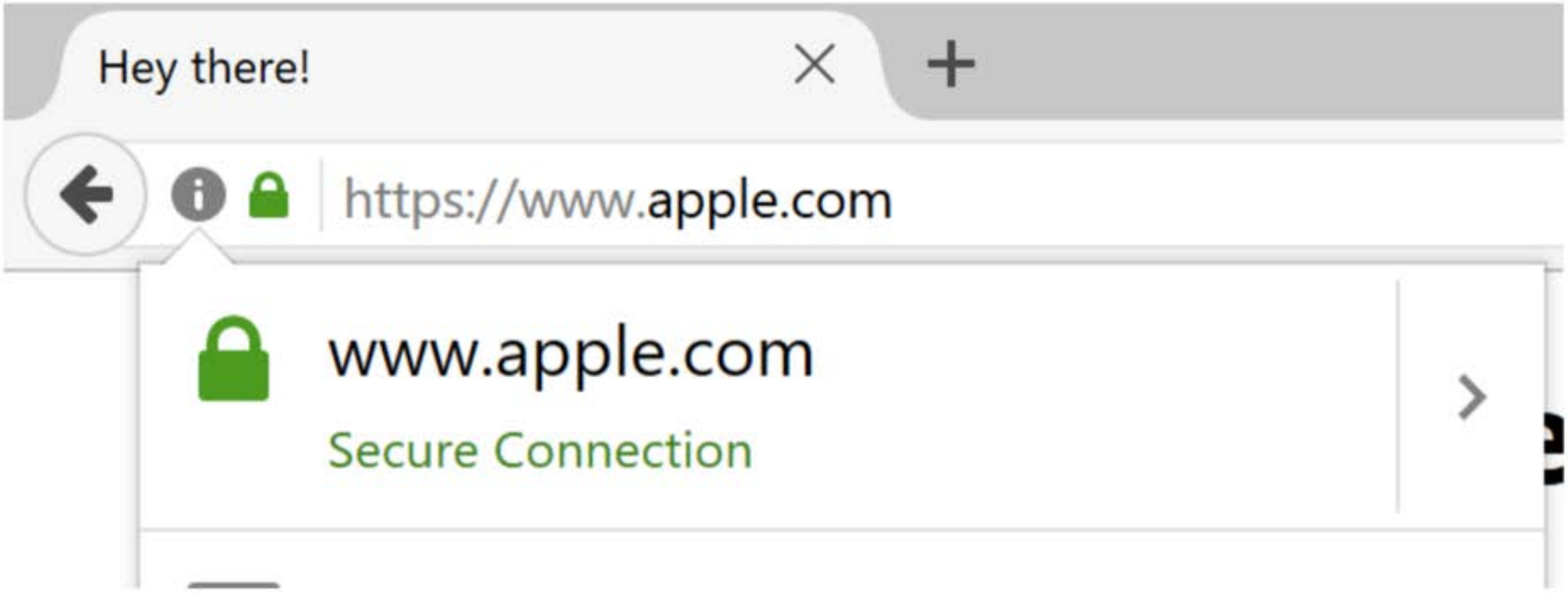
DAN GOODIN - 9/22/2016, 4:21 PM

LILY HAY NEWMAN SECURITY 04.18.17 7:00 AM

SNEAKY EXPLOIT ALLOWS PHISHING ATTACKS FROM SITES THAT LOOK SECURE

Phishing with Unicode Domains

Posted by Xudong Zheng on April 14, 2017



Before I explain the details of the vulnerability, you should take a look at the proof-of-concept.

Punycode makes it possible to register domains with foreign characters. It works by converting individual domain label to an alternative format using only ASCII characters. For example, the domain "xn--s7y.co" is equivalent to "短.co".

From a security perspective, Unicode domains can be problematic because many Unicode characters are difficult to distinguish from common ASCII characters. It is possible to register domains such as "xn--pple-

RISK ASSESSMENT —

BrickerBot, the permanent denial-of-service botnet, is back with a vengeance

New botnet squadrons wage fiercer, more intense attacks on unsecured IoT devices.

DAN GOODIN - 4/24/2017, 4:43 PM

Stuxnet Was Far More Dan...

www.businessinsider.com/stuxnet-was-far-more-dangerous-than-previous-thought-2013-11

BUSINESS INSIDER

BI INTELLIGENCE EVENTS LOGIN REGISTER US EDITION

Tech Finance Politics Strategy Life Sports Video All Search

MILITARY & DEFENSE

More: Stuxnet Iran Israel Cyberwarfare

The Stuxnet Attack On Iran's Nuclear Plant Was 'Far More Dangerous' Than Previously Thought

MICHAEL B KELLEY

NOV. 20, 2013, 12:58 PM 60,330 11

FACEBOOK LINKEDIN TWITTER

The Stuxnet virus that ravaged Iran's Natanz nuclear facility "was far more dangerous than the cyberweapon that is now

In-flight Wi-Fi is "direct li...

arstechnica.com/security/2015/04/15/in-flight-wi-fi-is-direct-link-to-hackers/

Register Log in

ars technica

MAIN MENU MY STORIES: 25 FORUMS SUBSCRIBE JOBS ARS CONSORTIUM

RISK ASSESSMENT / SECURITY & HACKTIVISM

In-flight Wi-Fi is "direct link" to hackers

Report: Planes could be targeted by a malicious hacker on the ground.

by Michael Rundle Apr 15, 2015 11:03am EDT

Share Tweet 88

LATEST FEATURE STORY

FEATURE STORY (2 PAGES)

The promise—and massive challenge—of making games for the Apple Watch

How to make 15-second microgames with targets "the size of salad bar ham cubes"

WATCH ARS VIDEO

Meet the e-voting machin...

arstechnica.com/tech-policy/2015/04/15/meet-the-e-voting-machine-so-easy-to-hack-it-will-take-your-breath-away/

Register Log in

ars technica

MAIN MENU MY STORIES: 25 FORUMS SUBSCRIBE JOBS ARS CONSORTIUM

LAW & DISORDER / CIVILIZATION & DISCONTENTS

Meet the e-voting machine so easy to hack, it will take your breath away

Virginia decertifies device that used weak passwords and wasn't updated in 10 years.

by **Dan Goodin** - Apr 15, 2015 2:55pm EDT

Share Tweet 156

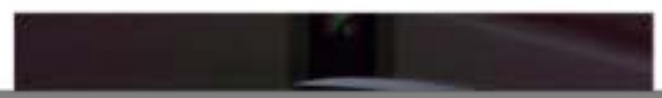
LATEST FEATURE STORY

FEATURE STORY (2 PAGES)

The promise—and massive challenge—of making games for the Apple Watch

How to make 15-second microgames with targets "the size of salad bar ham cubes"

WATCH ARS VIDEO



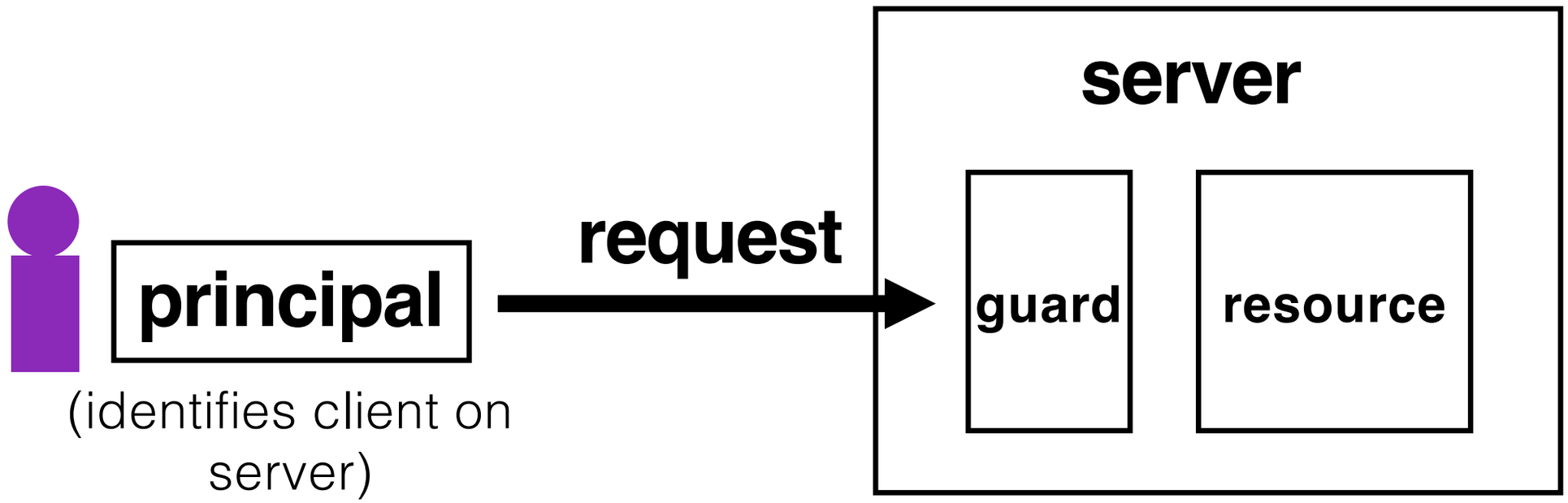
what makes computer security special?

why is security difficult?

steps towards building a more secure system:

1. be clear about goals (policy)
2. be clear about assumptions (threat model)

complete mediation: every request for resource goes through the guard



authentication: is the principal who they claim to be?

authorization: does principal have access to perform request on resource?

what can go wrong with the guard model?

sql injection demo

username	email	public?
karen	karen@fake.com	yes
peter	peter@fake.com	yes
katrina	no	

```
SELECT username, email FROM users WHERE
username='<username>' AND public='yes'
```

Let `<username>` = **`katrina' OR username='`**

sql injection demo

username	email	public?
karen	karen@fake.com	yes
peter	peter@fake.com	yes
katrina	no	

```
SELECT username, email FROM users WHERE
username='katrina' OR username='' AND
public='yes'
```

```
> cd /mit/bob/project
> cat ideas.txt
Hello world.
...
> mail alice@mit.edu < ideas.txt
```

what can go wrong with the guard model?

- **Adversarial attacks** are different from “normal” failures. They’re targeted, rarely random, and rarely independent. Just one successful attack can bring down a system.

- Securing a system starts by specifying our goals (**policy**) and assumptions (**threat model**).

- The **guard model** provides **complete mediation**. Even though things can still go wrong, systems that use this model avoid common pitfalls.

Security (Cont.)

The follow slides are used with permission from Professor Werner Dietl from Fall 2022

Software Architecture & Design

SE 464

Week 5: 04.10.2022
Security

Werner Dietl

https://ece.uwaterloo.ca/~wdietl/

NFP: Security

"The protection afforded to an automated information system in order to attain the applicable objectives of preserving the **integrity**, **availability** and **confidentiality** of information system resources (includes hardware, software, firmware, information/data, and telecommunications)."

National Institute of Standards and Technology

NFP: Security

- Confidentiality: Preserving the confidentiality of information means preventing unauthorized parties from accessing the information or perhaps even being aware of the existence of the information.
- Integrity: Maintaining the integrity of information means that only authorized parties can manipulate the information and do so only in authorized ways.
- Availability: Resources are available if they are accessible by authorized parties on all appropriate occasions.

Design Principles for Security

- **Least Privilege**: give each component only the privileges it requires
- **Fail-safe Defaults**: deny access if explicit permission is absent
- **Economy of Mechanism**: adopt simple security mechanisms
- **Complete Mediation**: ensure every access is permitted
- **Open Design**: do not rely on secrecy for security

Design Principles for Security

- **Separation of Privilege**: introduce multiple parties to avoid exploitation of privileges
- **Least Common Mechanism**: limit critical resource sharing to only a few mechanisms
- **Psychological Acceptability**: make security mechanisms usable
- **Defense in Depth**: have multiple layers of countermeasures

Security terminology

https://www.us-cert.gov/bsi/sdlc/design

http://web.mit.edu/Saltzer/www/publications/protection/Basic.html

Architectural Access Control Models

Decide whether access to a protected resource should be granted or denied

- Discretionary access control
  Based on the identity of the requestor, the resource, and whether the requestor has permission to access
- Mandatory access control
  Policy based

Discretionary Access Control

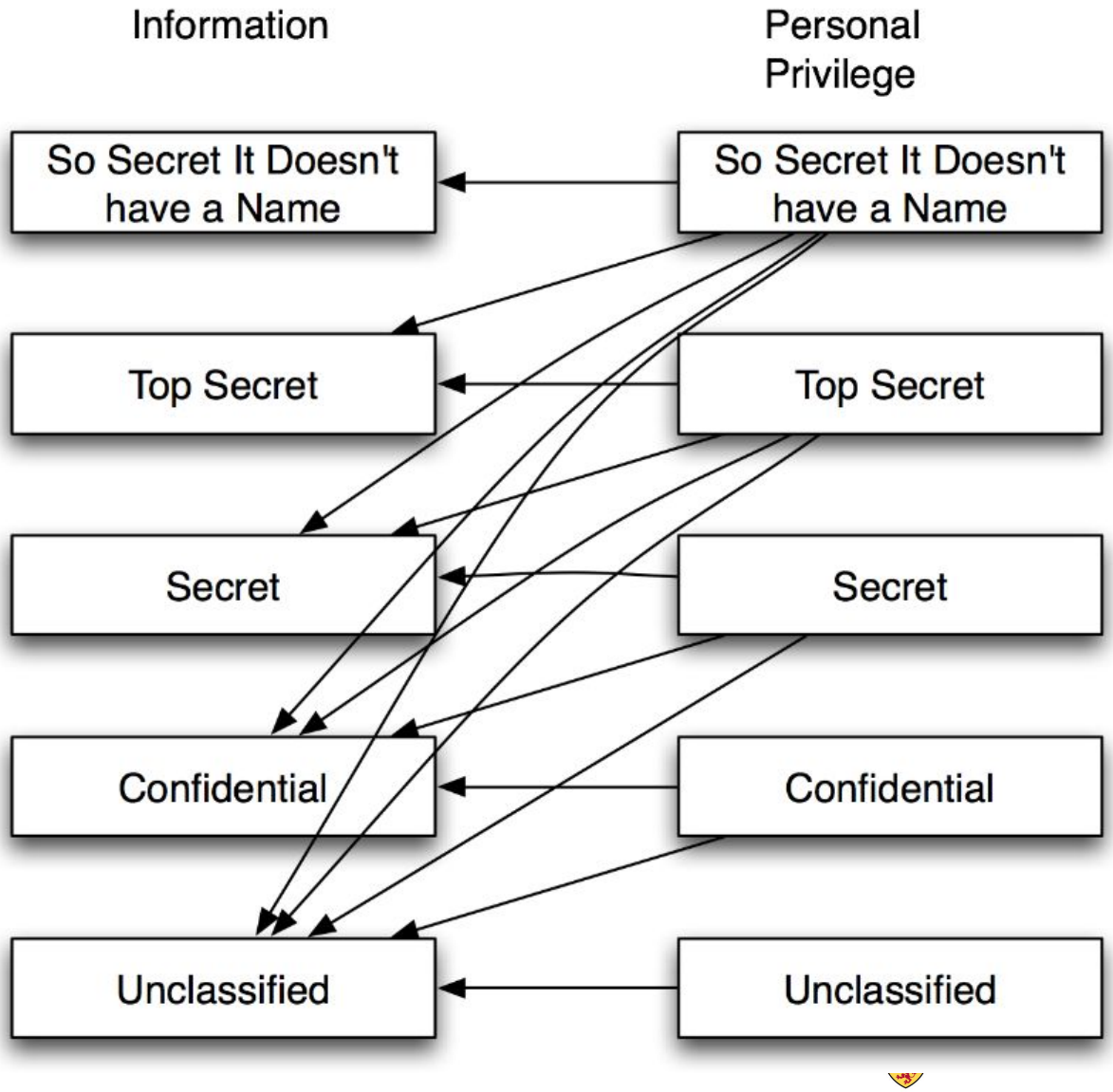
	Database A	Component Q	Interface F
Alice	Read-Write; Always	Bend	Yes
Bob	Read-Write; Between 9 and 5	Fold	No
Charles	No access	Spindle	No
Dave	No access	Mutilate	Yes
Eve	Read-only; Always	None	No

Mandatory Access Control

Bob: Secret

Alice: Confidential

Tom: Top Secret



Impersonation

Bob

Bob is reliable and everyone has a good opinion about Bob

Alice

"I am Bob"

Mallory
(malicious)

Misrepresentation

Bob

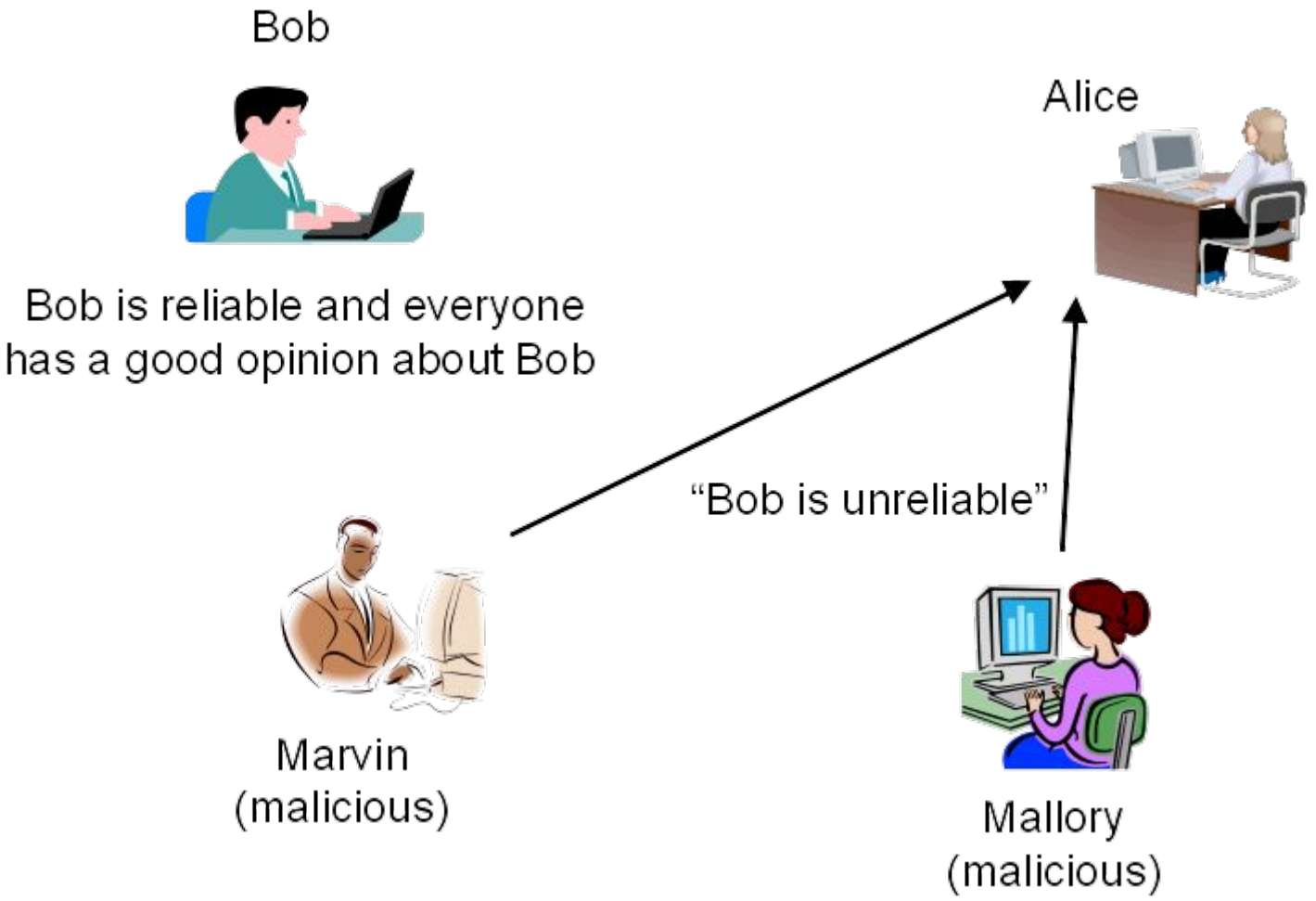
Bob is reliable and everyone has a good opinion about Bob

Alice

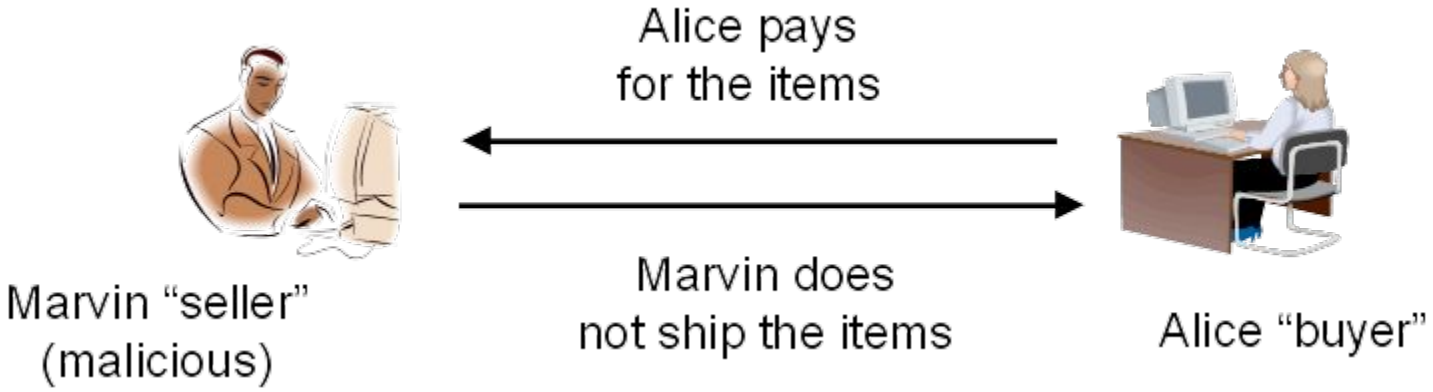
"Bob is unreliable"

Mallory
(malicious)

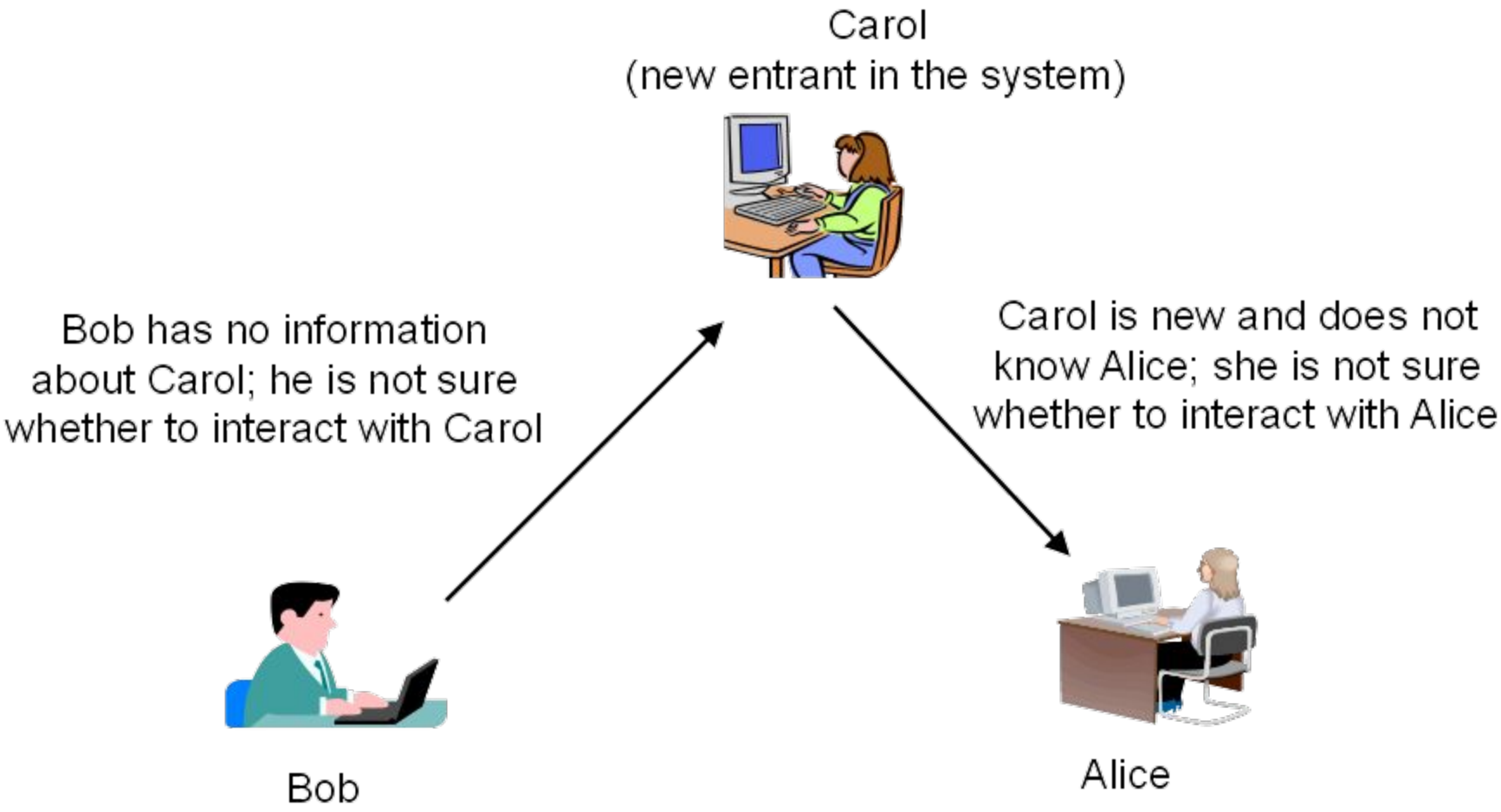
Collusion


Bob
Alice
Bob is reliable and everyone has a good opinion about Bob
"Bob is unreliable"
Marvin (malicious)
Mallory (malicious)

Fraudulent Actions



Addition of Unknowns



Trust Management

- **Trust** is a particular level of the subjective probability with which an agent assesses that another agent will perform a particular action in a context that affects his actions.
- **Reputation** is the expectation about an entity's behavior based on past behavior. May be used to determine trust

Two types of trust management systems

- Credential and Policy-based
- Reputation-based

Architecture in Practice: Chrome

Read:

- Browser Security: Lessons from Google Chrome
  http://queue.acm.org/detail.cfm?id=1556050
- The Security Architecture of the Chromium Browser
  http://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf

Security risks

Online content is insecure and can compromise:

- Confidentiality: Leak user data
- Integrity: Read/write arbitrary data on disk
- Availability: Crash host application and/or OS

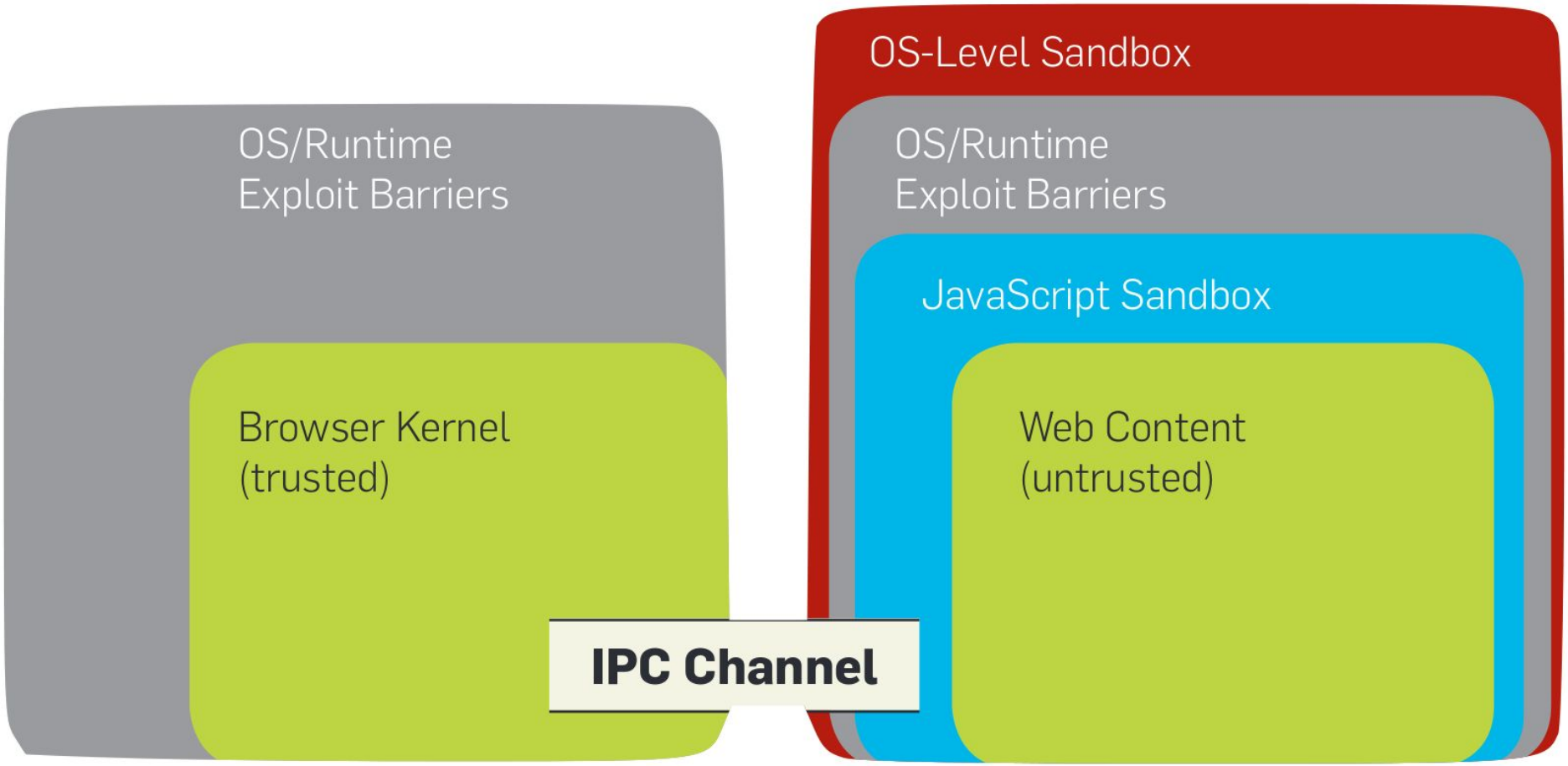
Chrome relies on **least privilege**, **separation of privilege**, and **defence in depth** to securely parse and render insecure content.

Main factors for Chrome security

Reduce

- the severity of vulnerabilities
  - Sandboxing
- the window of vulnerability
  - Auto-update (needs automated testing)
- the frequency of exposure
  - Warn about malicious sites

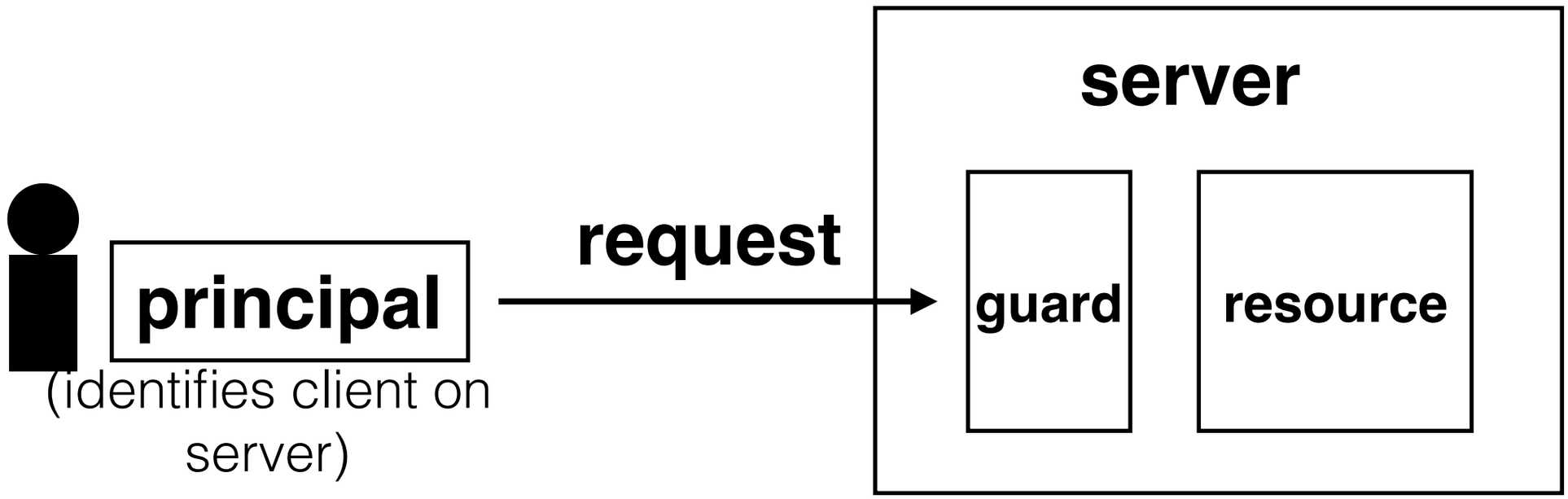
Chrome architecture



Authentication and Passwords

The following content is sourced from Computer Systems Design from MIT OCW
https://ocw.mit.edu/courses/6-033-computer-system-engineering-spring-2018/pages/week-12/

complete mediation: every request for resource goes through the guard



guard typically provides:

authentication: is the principal who they claim to be?

authorization: does principal have access to perform request on resource?

Rank	2011	2012	2013	2014	2015	2016	2017
1	password	password	123456	123456	123456	123456	123456
2	123456	123456	password	password	password	password	password
3	12345678	1234567	12345678	12345	12345678	12345	12345678
4	qwerty	abc123	qwerty	12345678	qwerty	12345678	qwerty
5	abc123	qwerty	abc123	qwerty	12345	football	12345
6	monkey	monkey	123456789	123456789	123456789	qwerty	123456789
7	1234567	letmein	111111	1234	football	1234567890	letmein
8	letmein	dragon	1234567	baseball	1234	1234567	1234567
9	trustno1	111111	iloveyou	dragon	1234567	princess	football
10	dragon	baseball	adobe123	football	baseball	1234	iloveyou
11	baseball	iloveyou	123123	1234567	welcome	login	admin
12	111111	trustno1	admin	monkey	123456789	welcome	welcome
13	iloveyou	1234567	1234567890	letmein	abc123	solo	monkey
14	master	sunshine	letmein	abc123	111111	abc123	login
15	sunshine	master	photoshop	111111	1qaz2wsx	admin	abc123
16	ashley	123123	1234	mustang	dragon	121212	starwars
17	bailey	welcome	monkey	access	master	flower	123123
18	passw0rd	shadow	shadow	shadow	monkey	passw0rd	dragon
19	shadow	ashley	sunshine	master	letmein	dragon	passw0rd
20	123123	football	12345	michael	login	sunshine	master
21	654321	jesus	password1	superman	princess	master	hello
22	superman	michael	princess	696969	qwertyuiop	hottie	freedom
23	qazwsx	ninja	azerty	123123	solo	loveme	whatever
24	michael	mustang	trustno1	batman	passw0rd	zaq1zaq1	qazwsx
25	Football	password	000000	trustno1	starwars	password1	trustno1

problem: users pick terrible passwords

source: SplashData, via wikipedia

username	password
dom	fam1ly
han	dr1ftnNt0ky0
roman	Lamb0s4ever
tej	31173h4ck3r

```
check_password(username, inputted_password):
    stored_password = accounts_table[ username]
    return    stored_password ==  inputted_password
```

problem: adversary with access to server can get passwords

username	hash(password)
dom	e5f3c4e1694c53218978fae2c302faf4a817ce7b
han	365dab99ab03110565e982a76b22c4ff57137648
roman	ed0fa63cd3e0b9167fb48fa3c1a86d476c1e8b27
tej	0e0201a89000fe0d9f30adec170dabce8c272f7c

```
check_password (username, inputted_password):
    stored_hash = accounts_table[ username]
    inputted_hash = hash(inputted_password)
 return    stored_hash ==  inputted_hash
```

problem: hashes are fast to compute, so adversary could quickly create a “rainbow table”

username	slow_hash(password)
dom	gamynjSAIeYZ4iOBT4uaO3r5ub8O
han	JXYWVPkpoQ6W1tbA21t6c66G4QUo
roman	Xn5U1QvQz5MGOzdfJWgF8OiDFv1q
tej	lo5WIidPPZePoSyMB2O.fUz3fLeZ

```
check_password (username, inputted_password):
    stored_hash = accounts_table[ username]
    inputted_hash = slow_hash(inputted_password)
 return    stored_hash ==  inputted_hash
```

problem: adversary can still create rainbow tables for the most common passwords

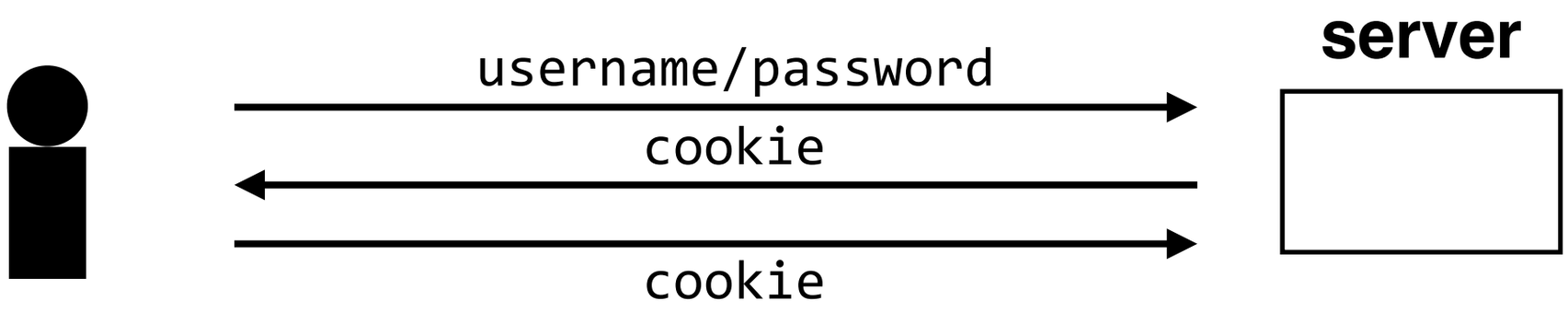
stored in plaintext

username	salt	slow_hash(password | salt)
dom	LwVx6kO4SNY3jPVfOpfYe.	M4ayLRWuzU.sSQtjoteIrIjNXI4UX
han	UbDsytUST6d0cFpmuhWu.e	Y8ie/A18u9ymrS0FgVh9IOVx2Qe48
roman	CnfkXqUJz5C5OfucP/UKIu	3GDJu07gk2iL7mFVquOzPt3L3IITe
tej	cBGohtI6BwsaVs0SAo0u7.	8/v1Kl6rImUMYVw/.oGmA/BaRAlgC

```
check_password (username, inputted_password)
    stored_hash = accounts_table[ username].hash
    salt = accounts_table[ username].salt
    inputted_hash = slow_hash(inputted_password | salt)
return    stored_hash ==  inputted_hash
```

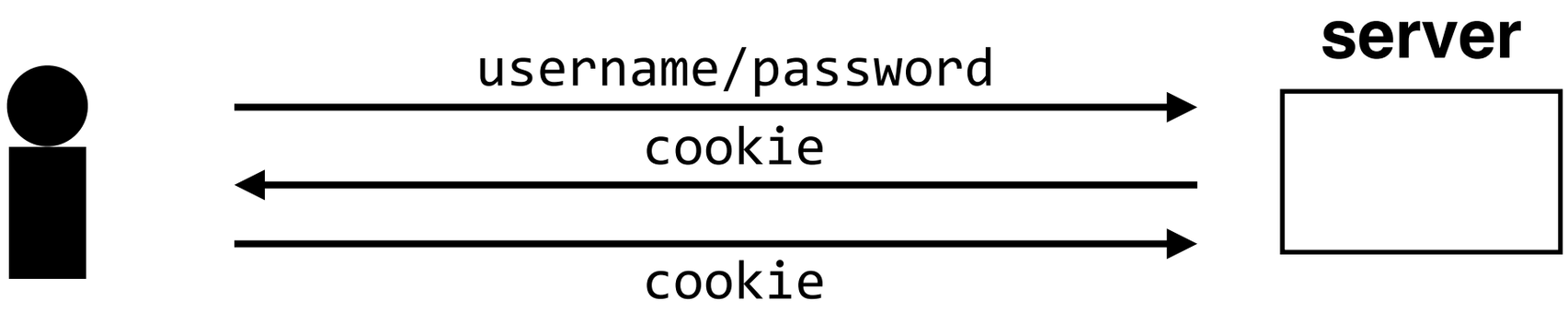
adversary would need a separate rainbow table for every possible salt

how can we avoid transmitting the password over and over?



once the client has been authenticated, the server will send it a “cookie”, which it can use to keep authenticating itself for some period of time

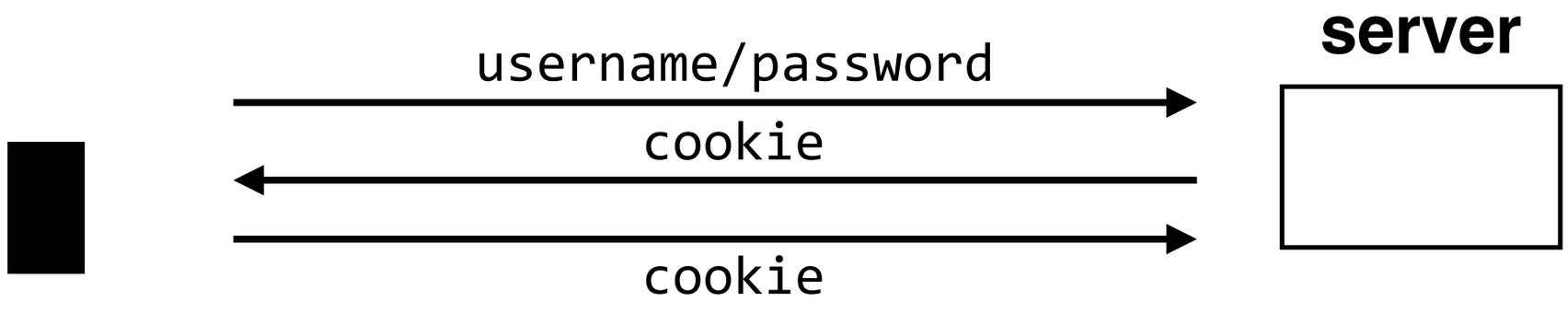
how can we avoid transmitting the password over and over?




```
cookie = {username, expiration} ?
```

problem: adversaries could easily create their own cookies

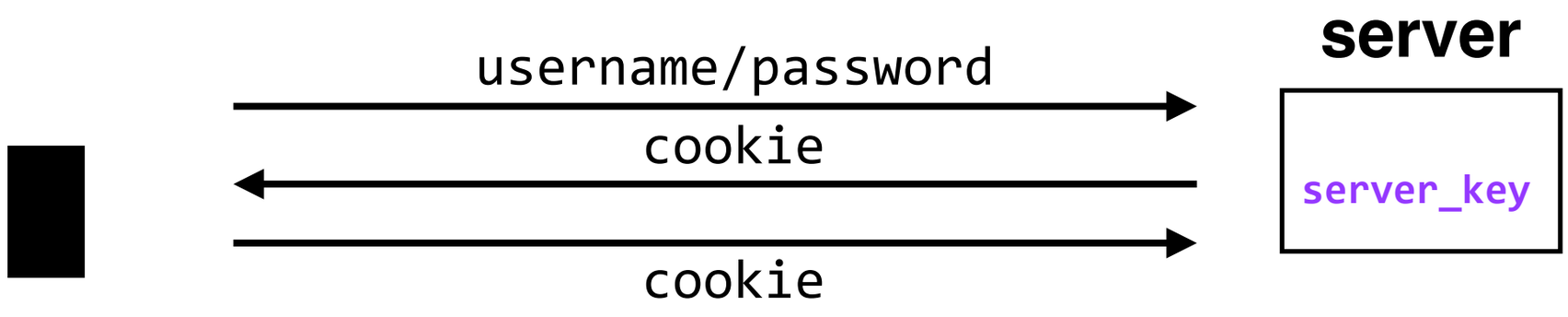
how can we avoid transmitting the password over and over?



`cookie = {username, expiration, H(username | expiration)} ?`

problem: adversaries could still easily create their own cookies

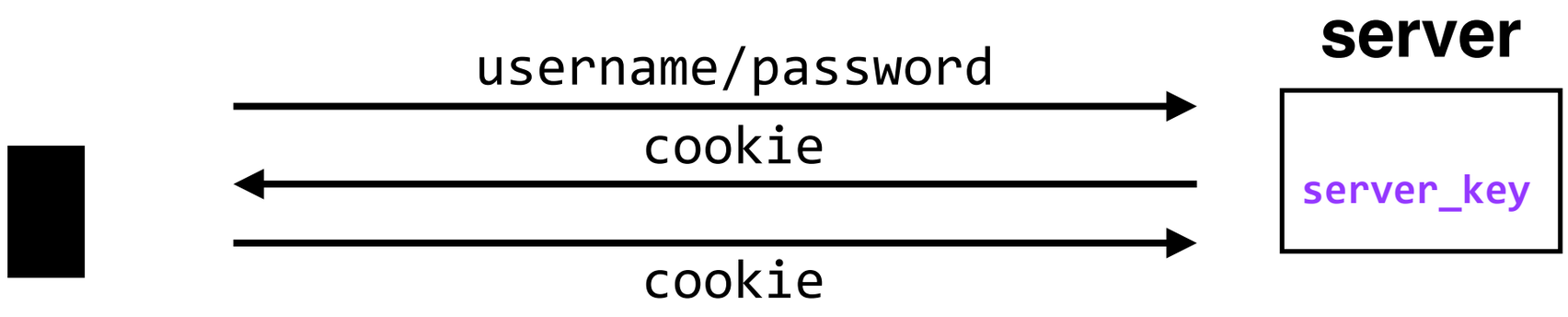
how can we avoid transmitting the password over and over?




```
cookie = {username, expiration, server_key, H(username |
                                              expiration)} ?
```

problem: adversaries could *still* easily create their own cookies

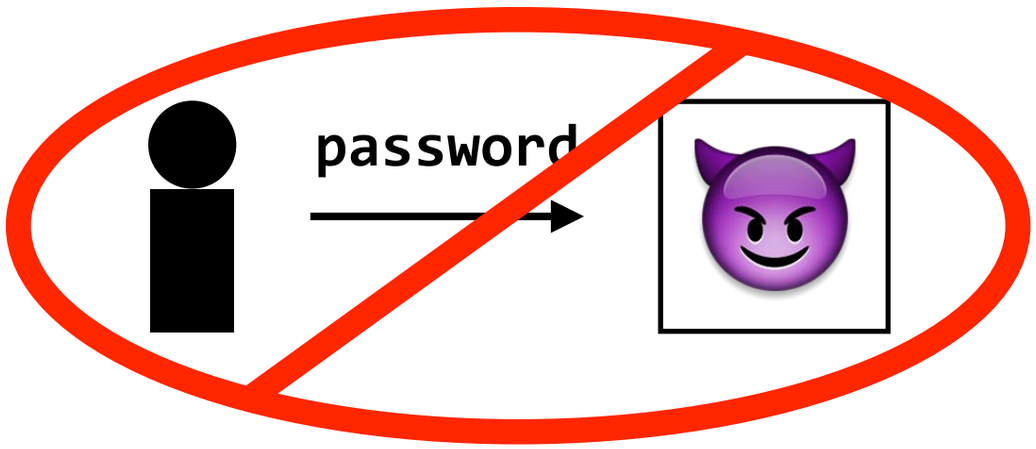
how can we avoid transmitting the password over and over?



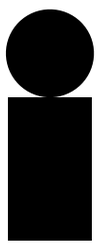
{username, expiration, H(server_key | username | expiration)}

how can we protect against phishing attacks, where an adversary tricks a user into revealing their password?

must avoid sending the password to the server entirely, but still allow valid servers to authenticate users



challenge-response protocol



(random number)

458653

ccfc38b071124374ea039ff8b40e83fbf4e80d92

= H(fam1ly | 458643)

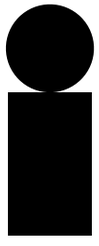
password is never sent directly

valid server

username	password
dom	fam1ly
han	dr1ftnNt0ky0
roman	Lamb0s4ever
tej	31173h4ck3r

server computes H(fam1ly | 458643) and checks

challenge-response protocol



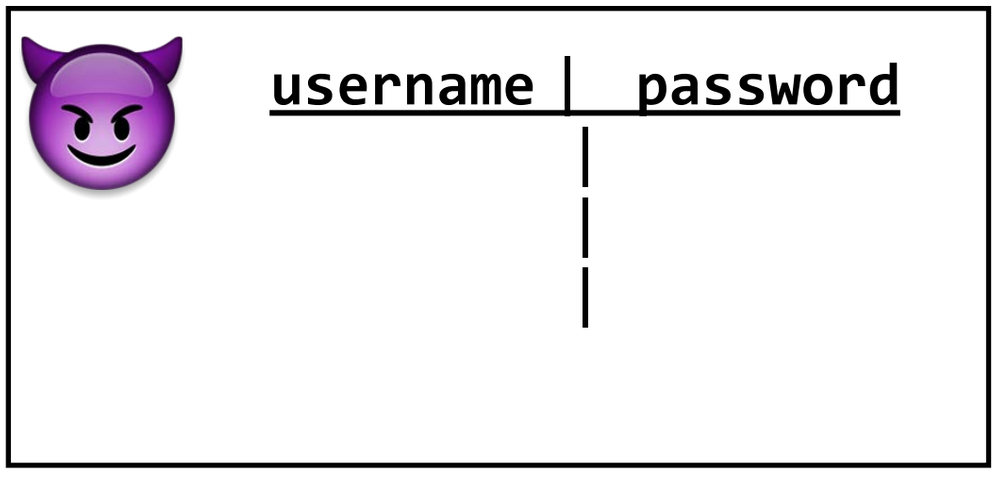
(random number)

458653

ccfc38b071124374ea039ff8b40e83fbf4e80d92

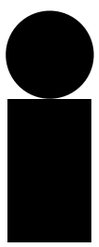
= H(fam1ly | 458643)

adversary-owned server



adversary only learns H(fam1ly | 458643); can't recover the password from that

challenge-response protocol



(random number)

458653

ccfc38b071124374ea039ff8b40e83fbf4e80d92

= H(fam1ly | 458643)

password is never sent directly

valid server

username	password
dom	fam1ly
han	dr1ftnNt0ky0
roman	Lamb0s4ever
tej	31173h4ck3r

server computes H(fam1ly | 458643) and checks

adversary-owned servers (that don't know passwords) won't learn the password; client never sends password directly

problems arise when the server stores (salted) hashes — as it should be doing — but there are challenge-response protocols that handle that case

how do we initially set (bootstrap) or reset a password?

are there better alternatives to passwords?

- Using passwords securely takes some effort. Storing **salted hashes**, incorporating **session cookies**, dealing with **phishing**, and **bootstrapping** are all concerns.

- Thinking about how to use passwords provides more **general lessons**: consider human factors when designing secure systems, in particular.

- There are always **trade-offs**. Many “improvements” on passwords add security, but also complexity, and typically decrease usability.