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

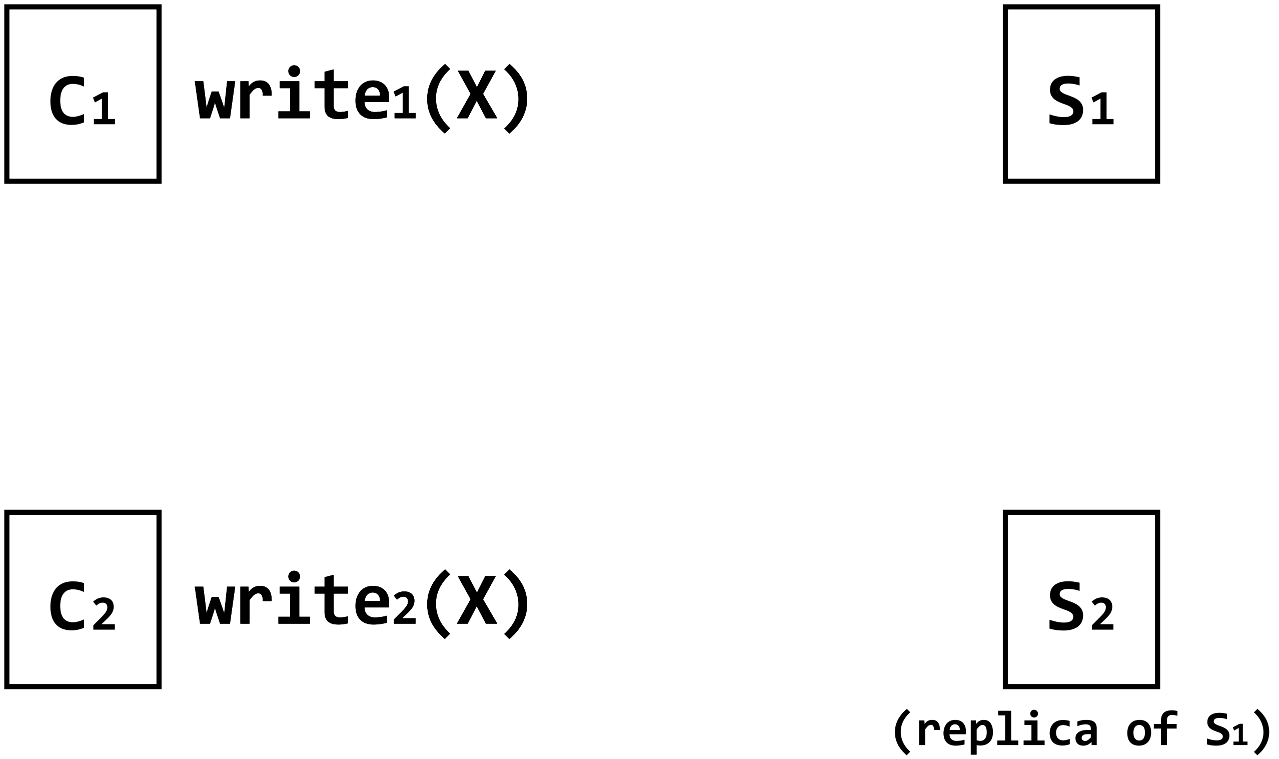
the abstraction that makes that easier is

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



C_1 $\texttt{write}_1(X)$

S_1

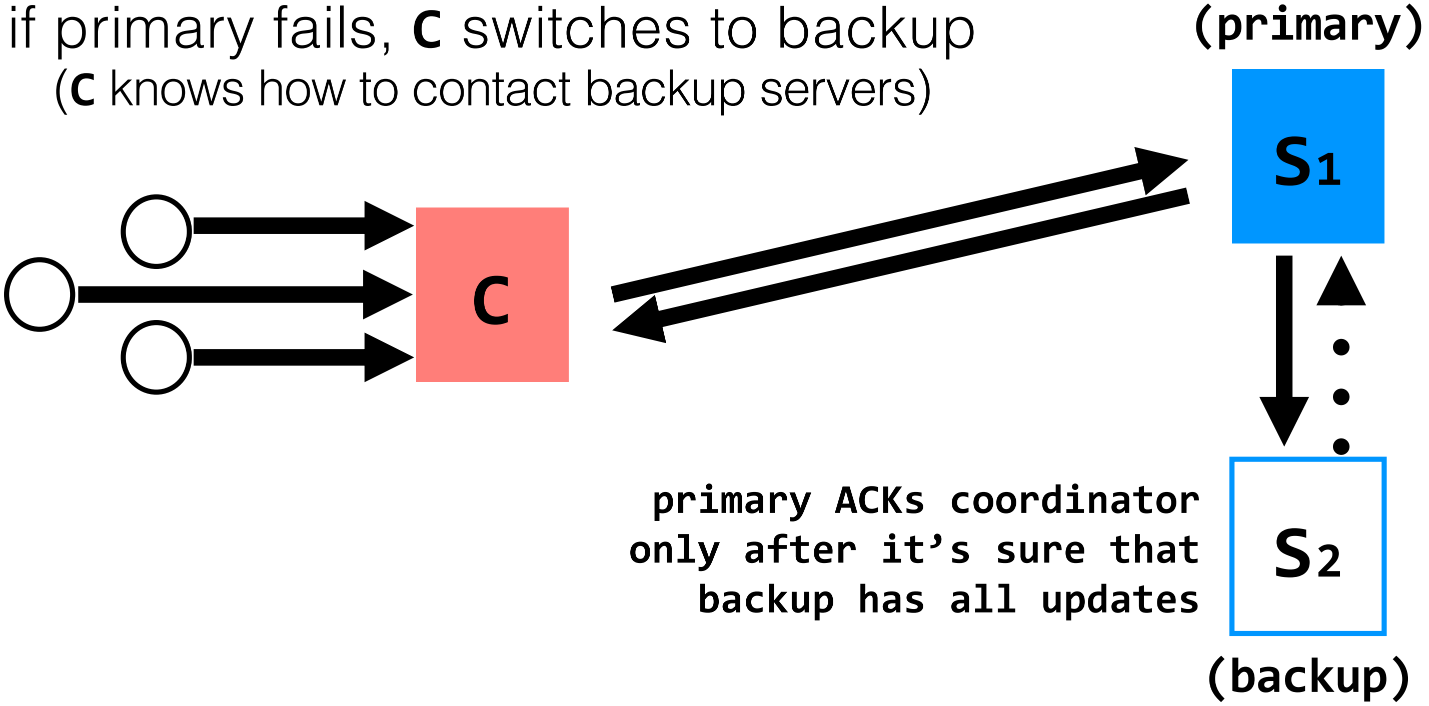
C_2 $\texttt{write}_2(X)$

S_2

(replica of S_1)

problem: replica servers can become inconsistent

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

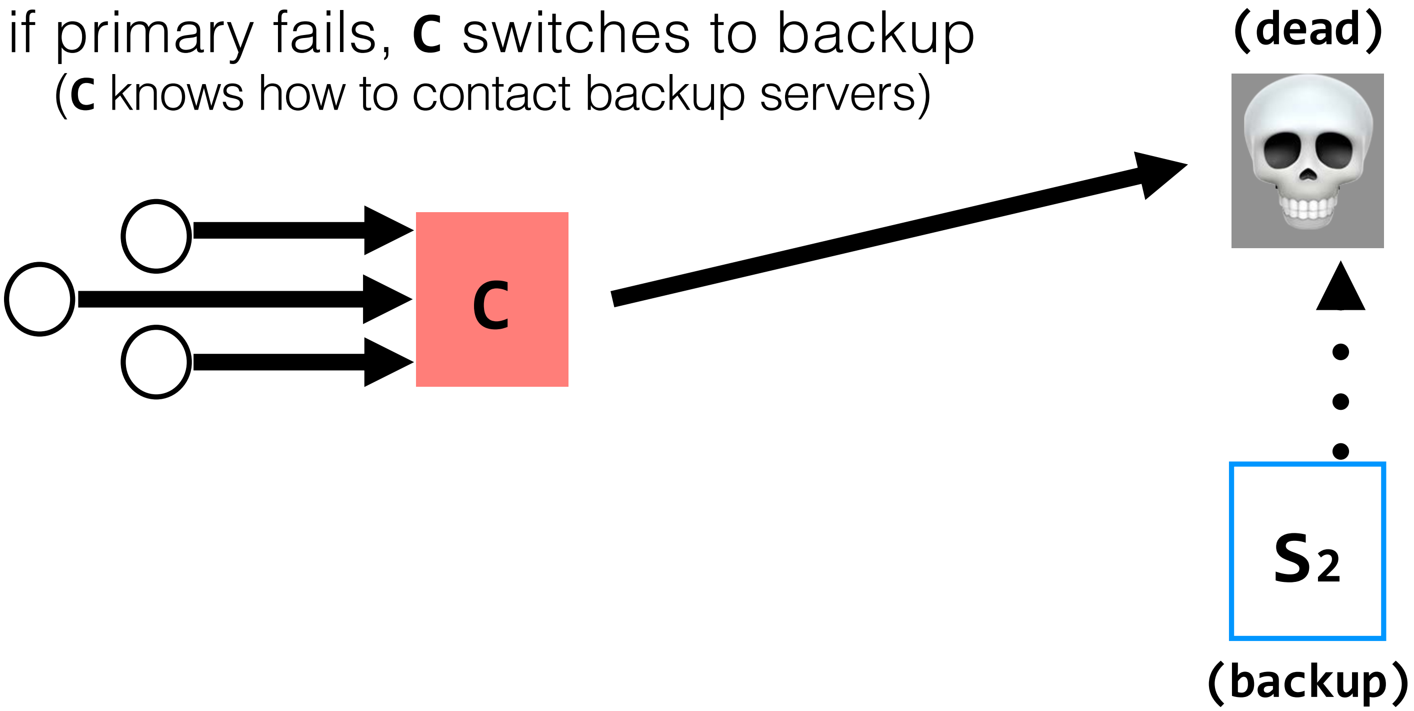


primary chooses order of operations, decides all non-deterministic values

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

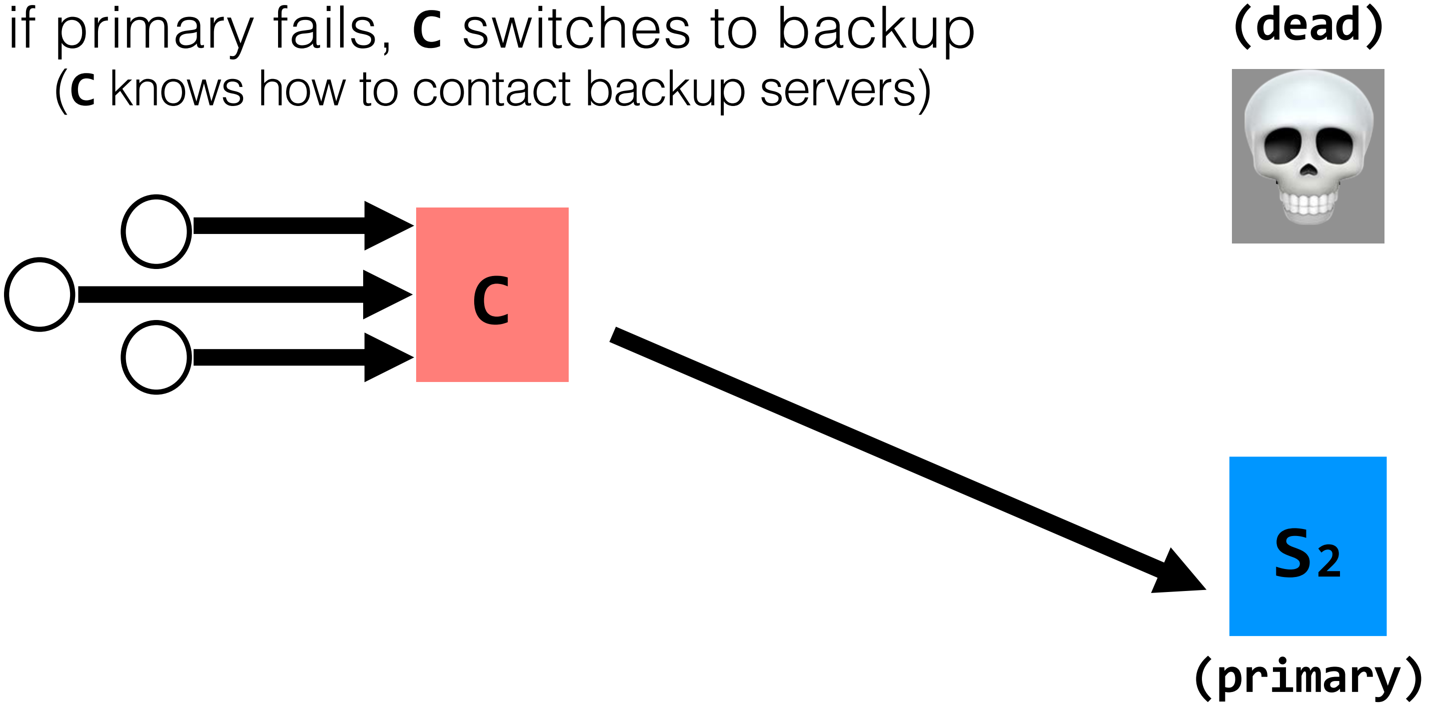
attempt: coordinators communicate with primary servers, who communicate with backup servers

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



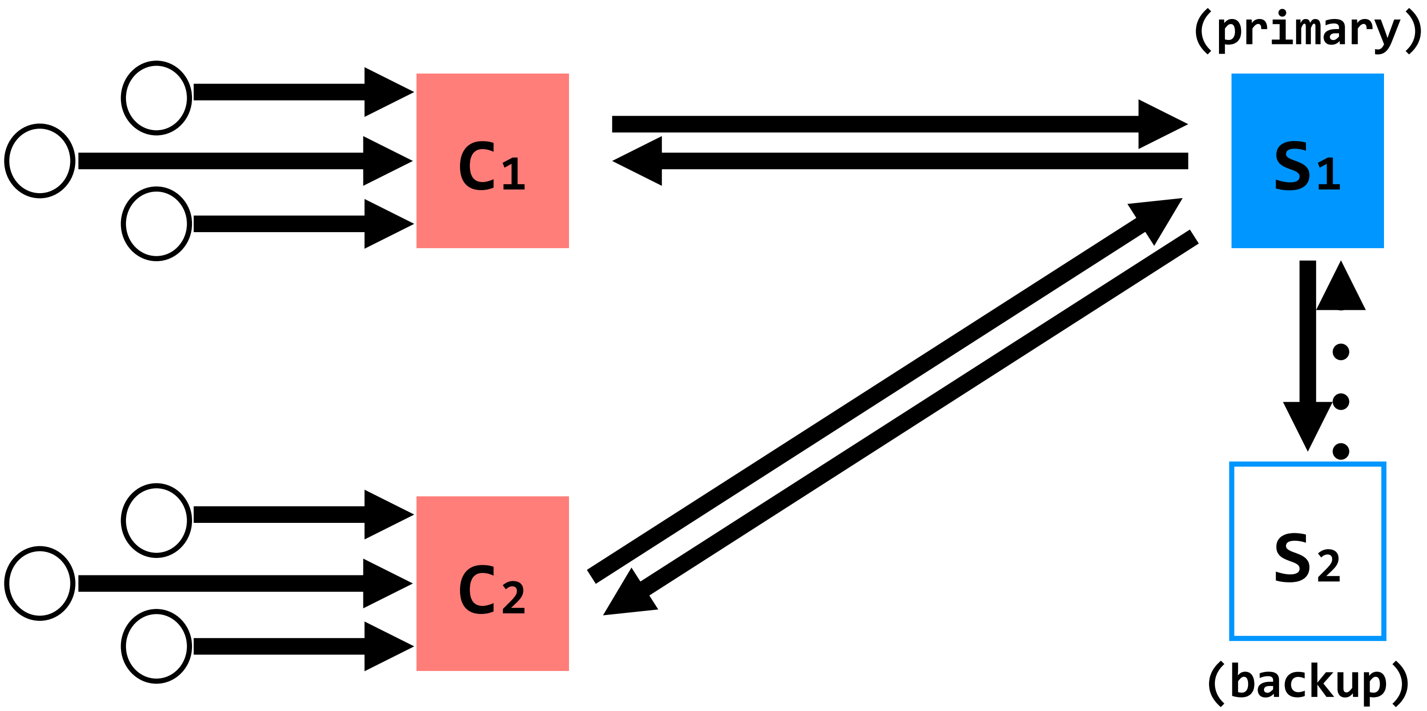
attempt: coordinators communicate with primary servers, who communicate with backup servers

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



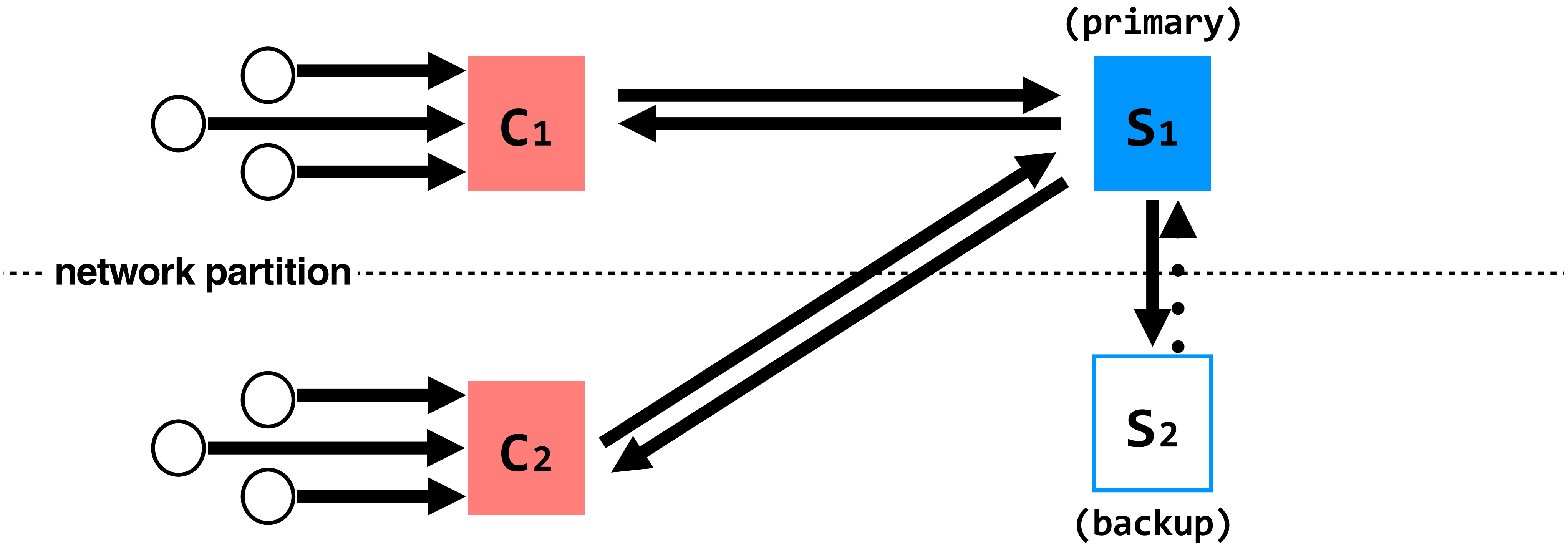
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems



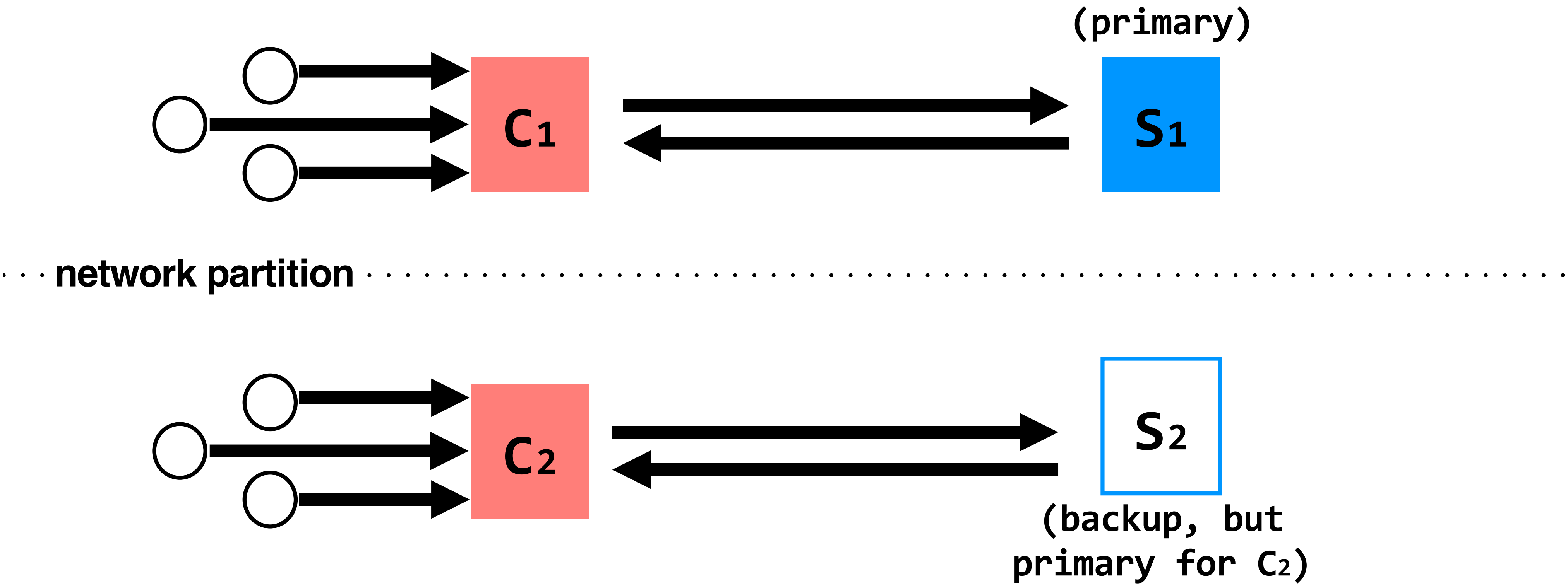
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems



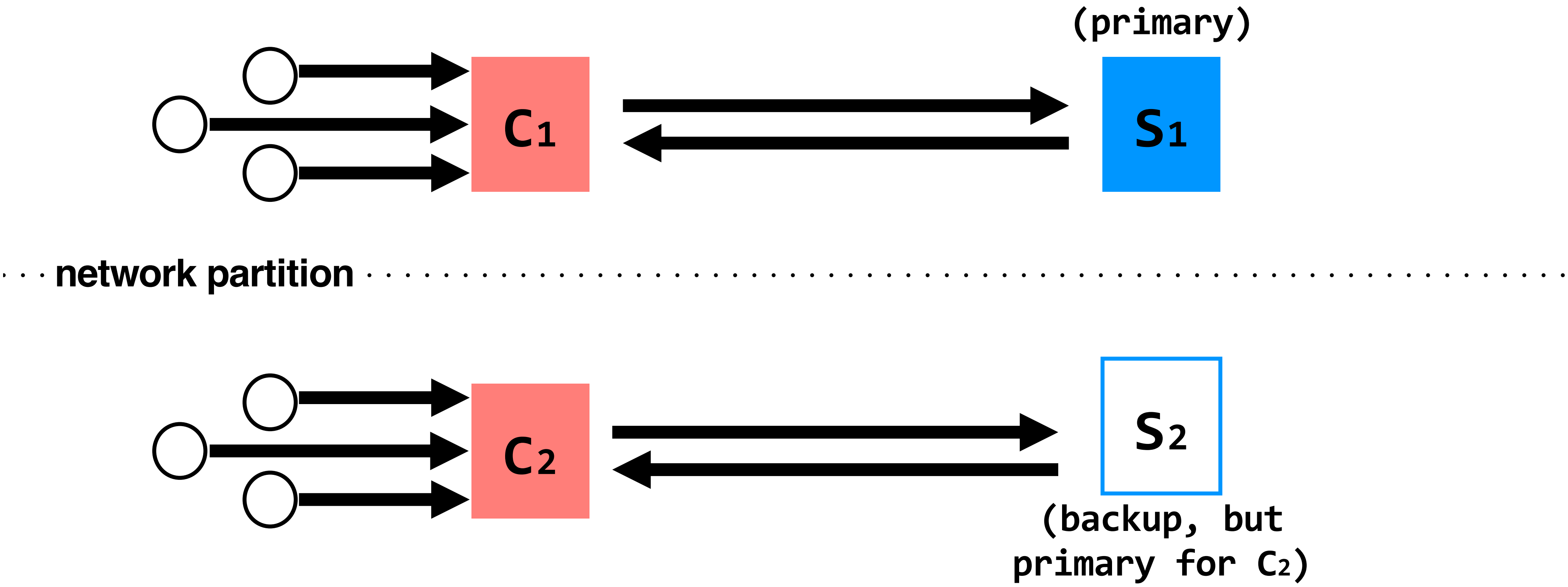
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems



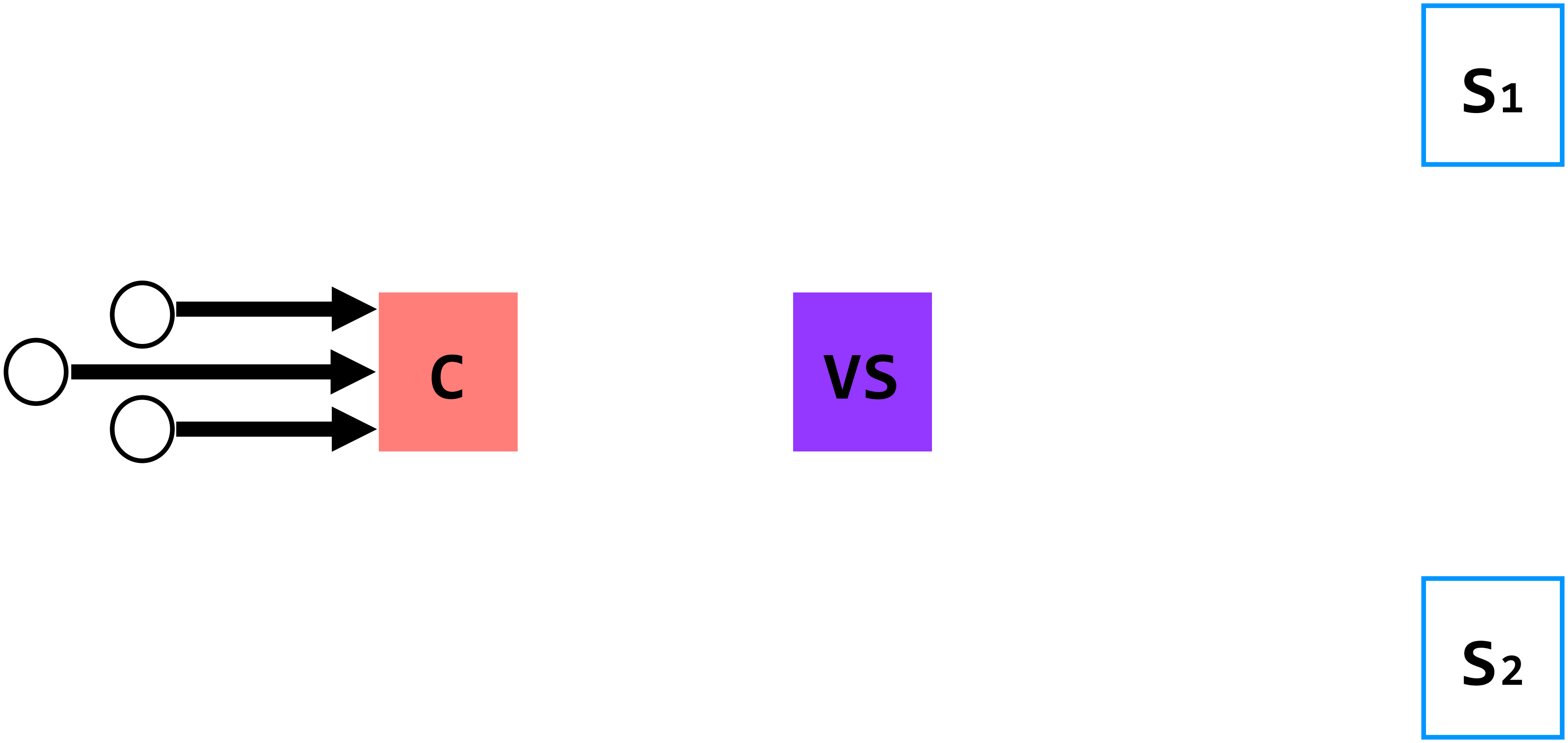
attempt: coordinators communicate with primary servers, who communicate with backup servers

multiple coordinators + the network = problems

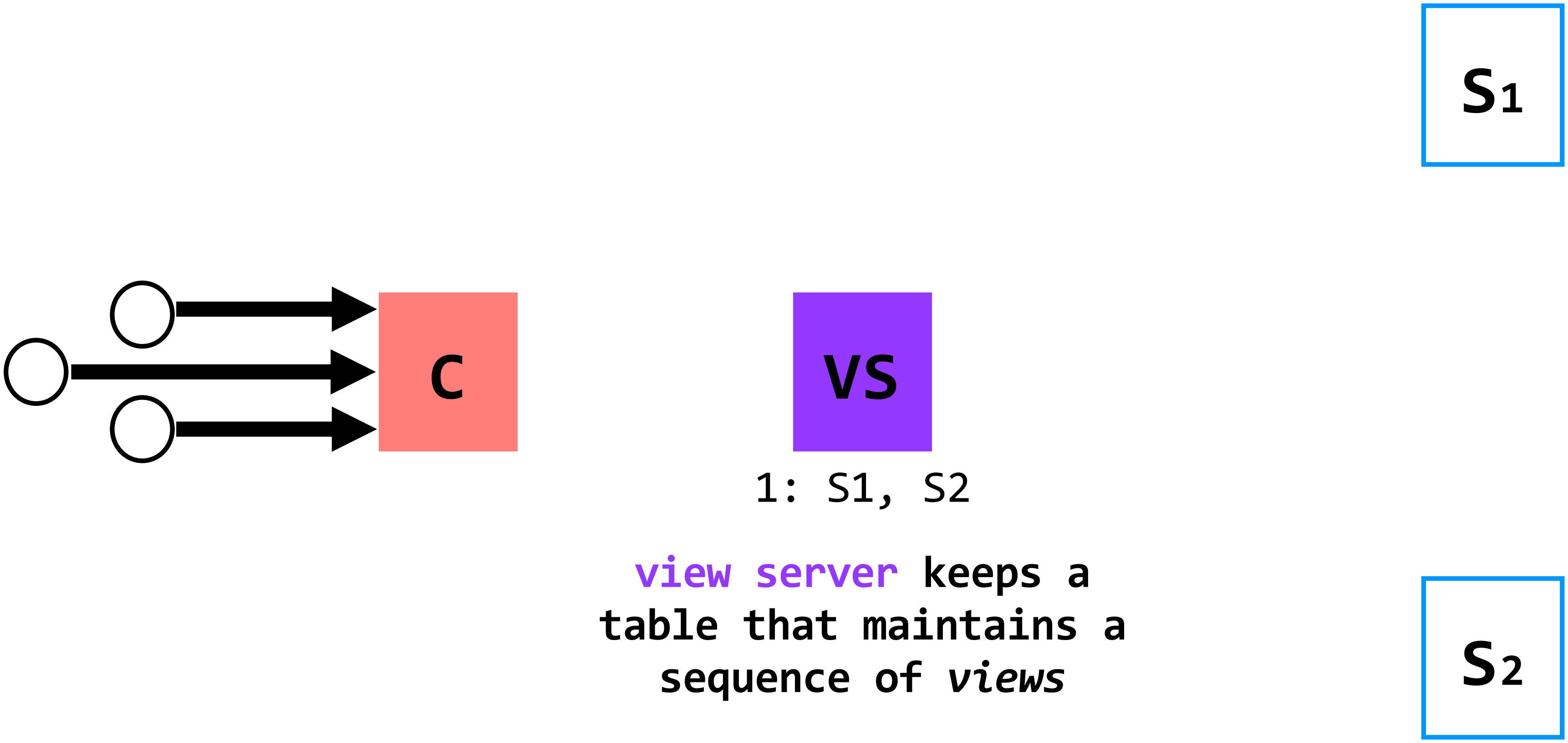


C_1 and **C_2** are using different primaries; **S_1** and **S_2** are no longer consistent

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



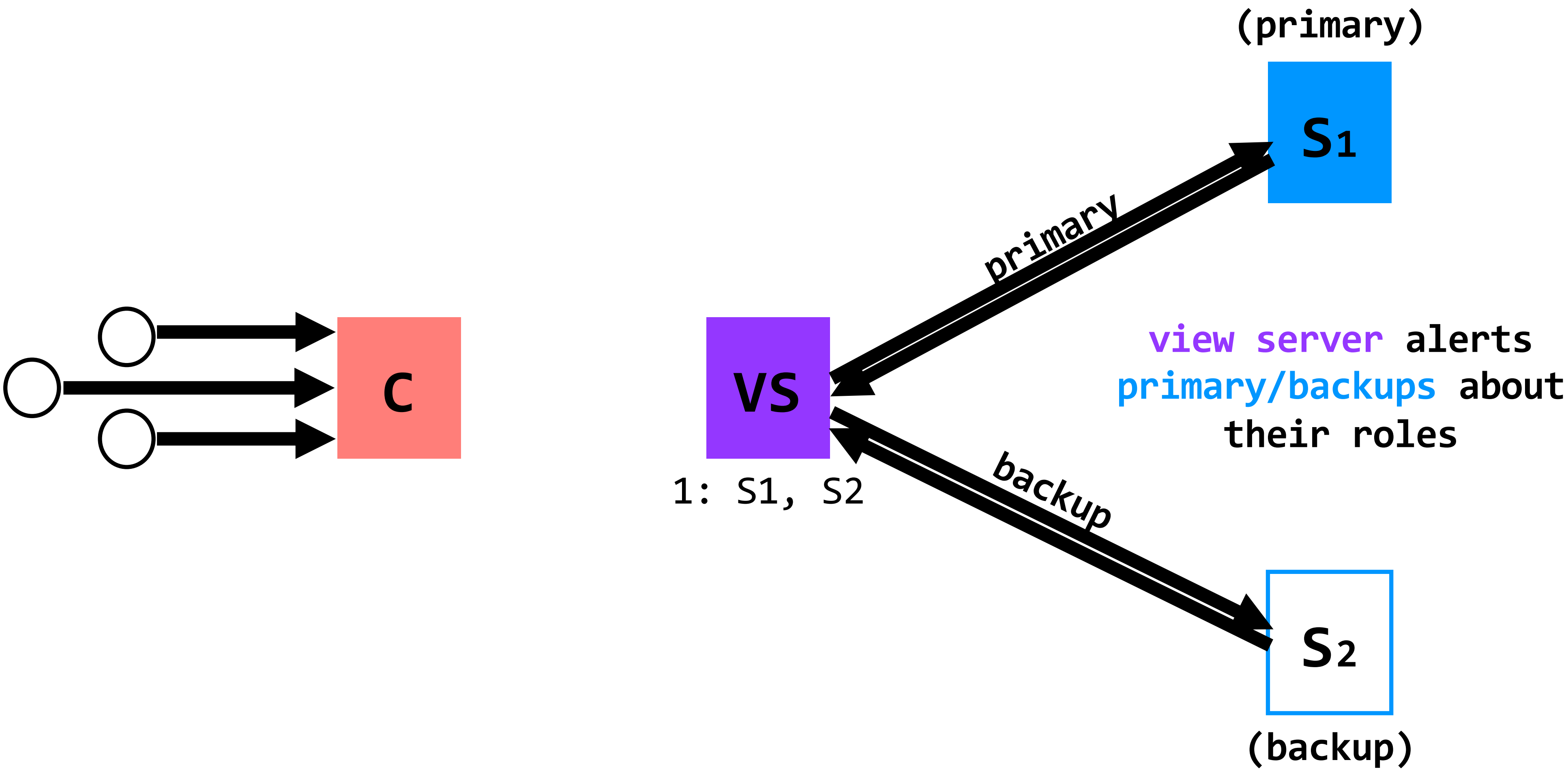
use a **view server**, which determines which replica is the primary



1: S1, S2

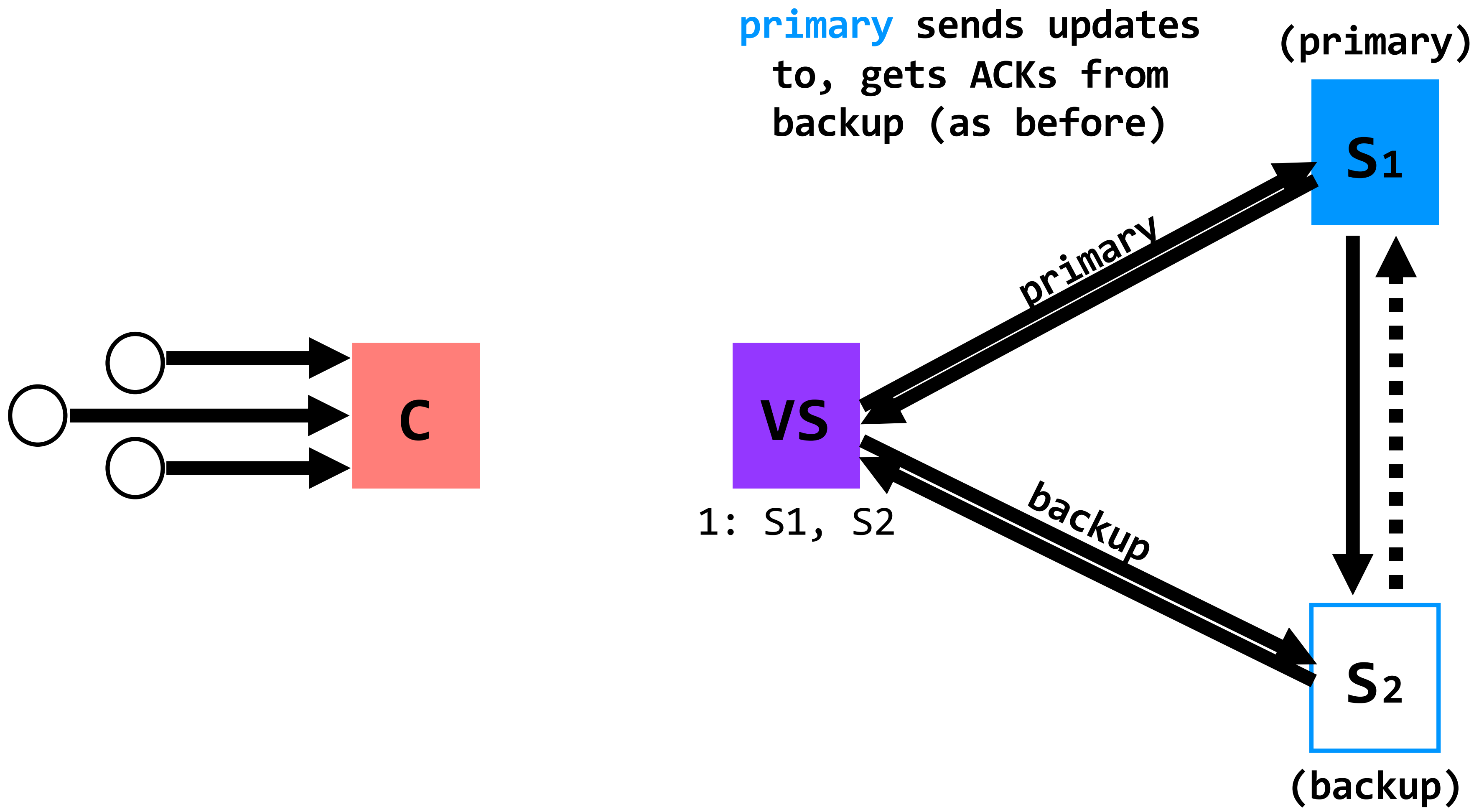
view server keeps a table that maintains a sequence of *views*

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

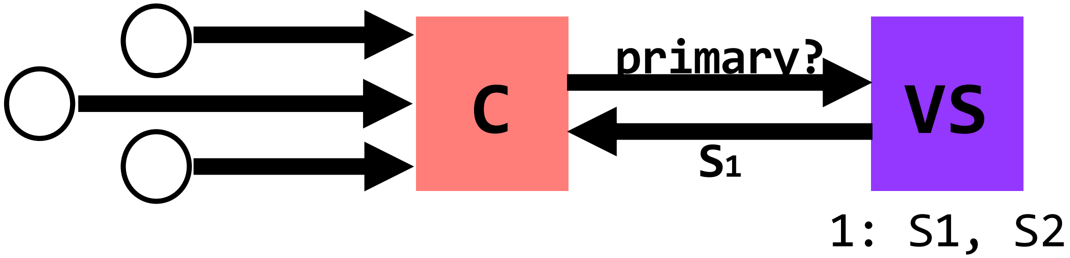


view server alerts primary/backups about their roles

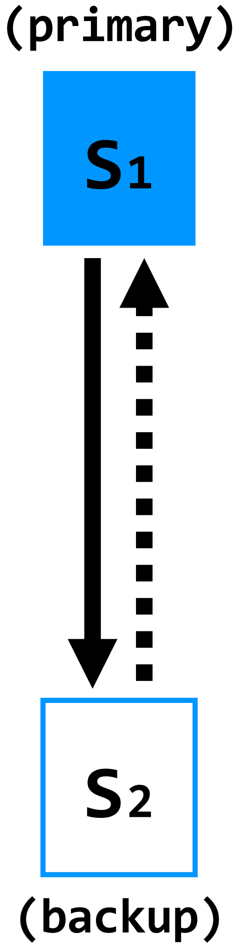
use a **view server**, which determines which replica is the primary



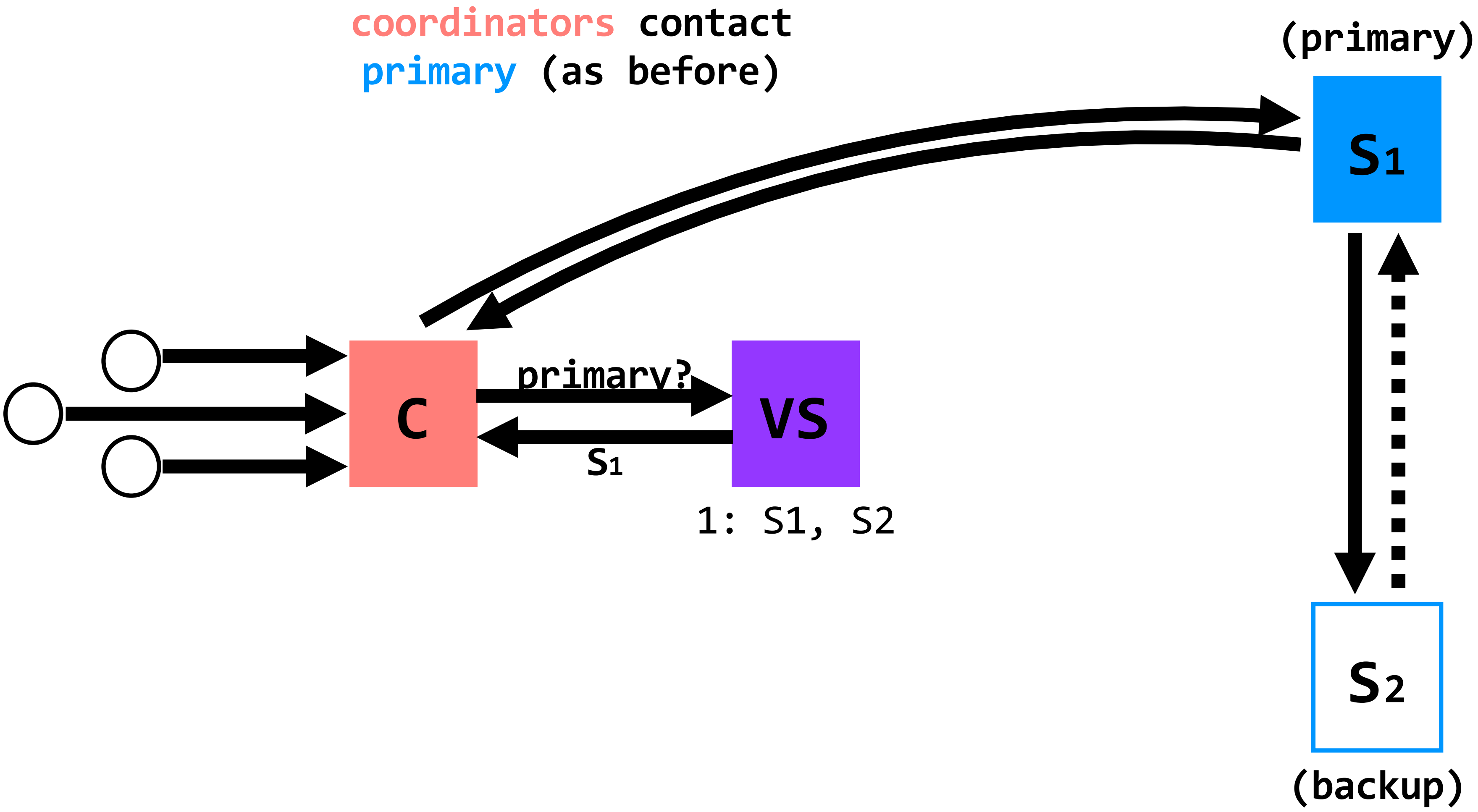
use a **view server**, which determines which replica is the primary



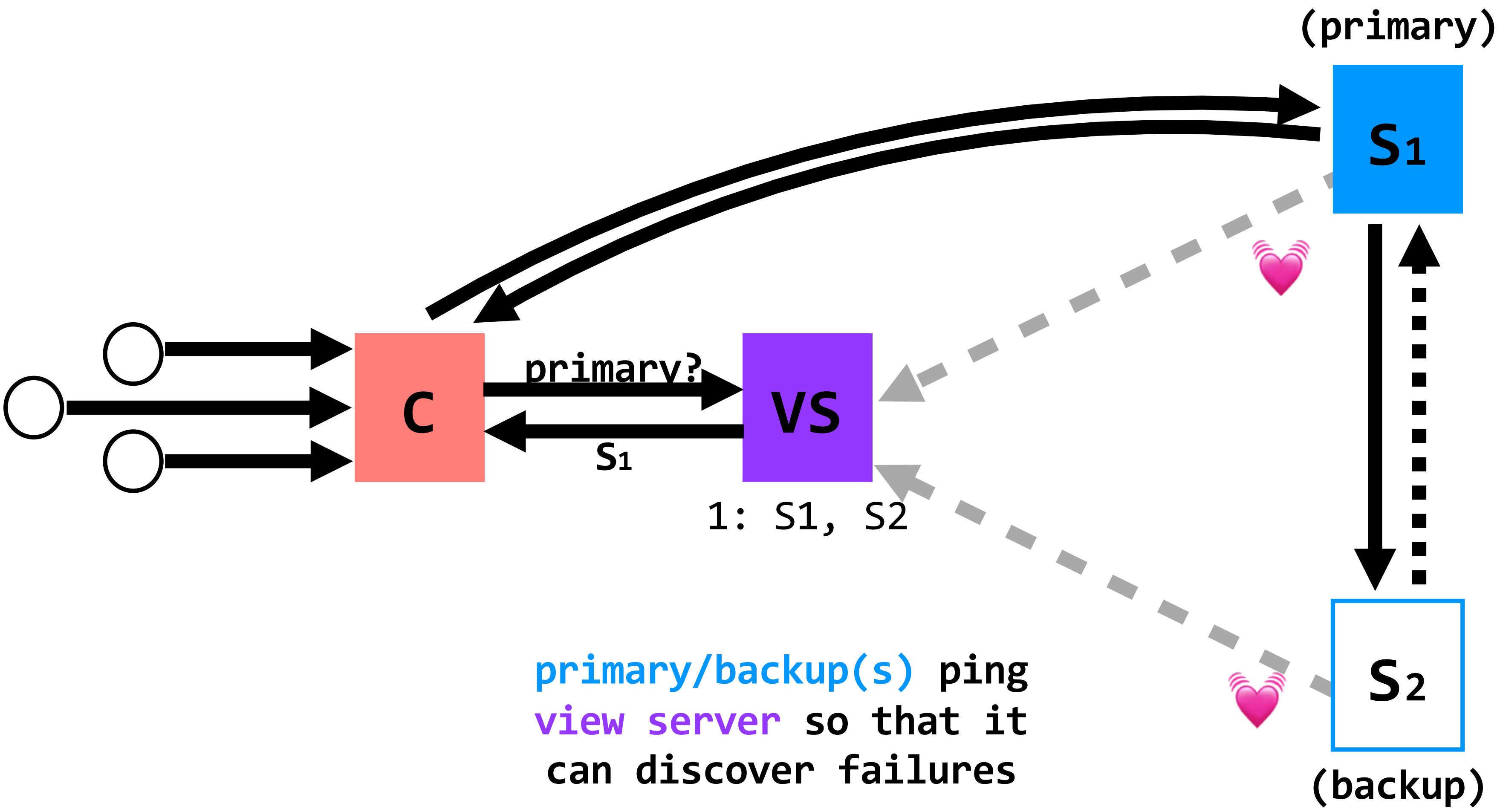
coordinators make requests to view server to find out who is primary



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



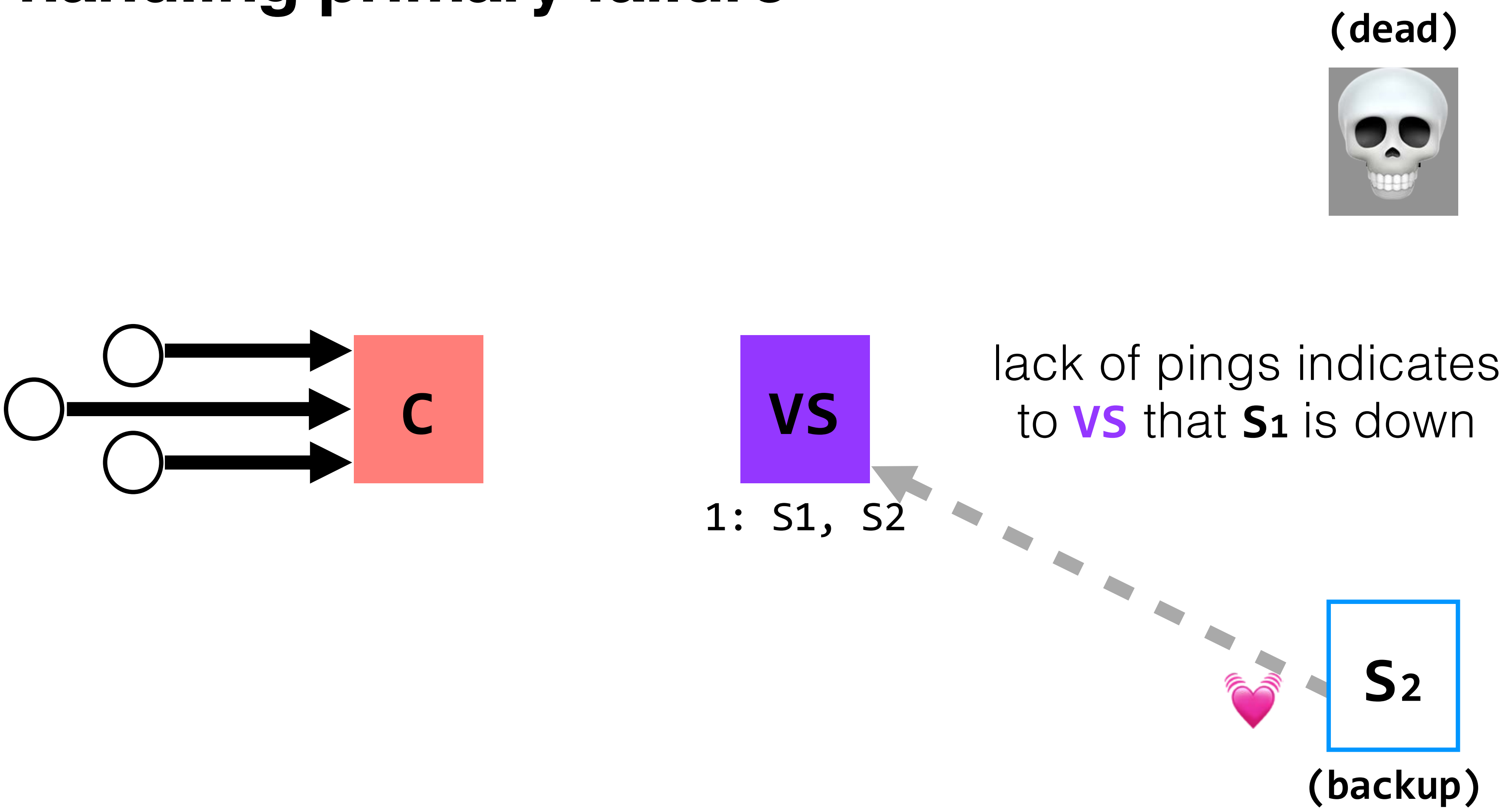
use a **view server**, which determines which replica is the primary



primary/backup(s) ping view server so that it can discover failures

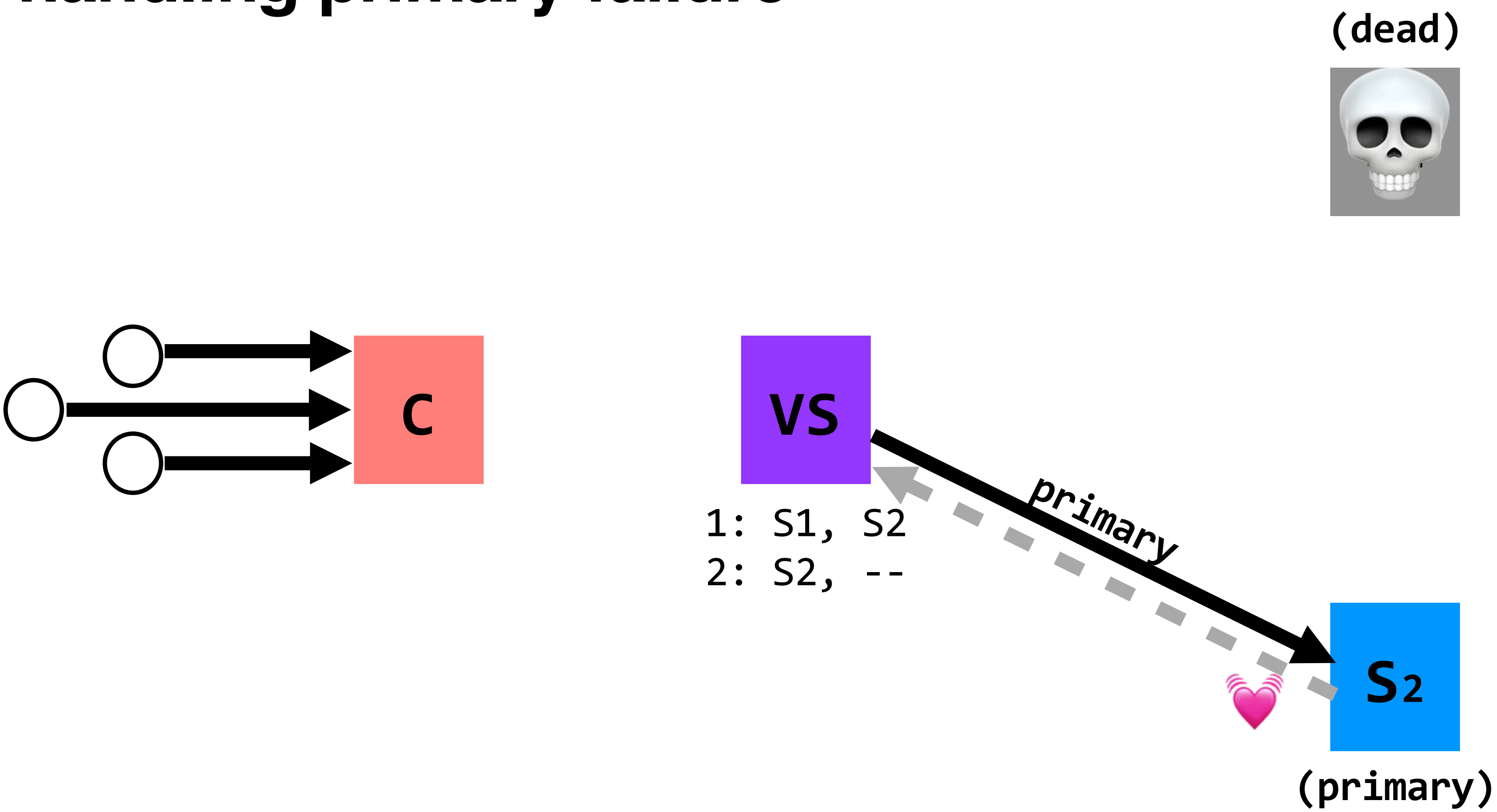
use a **view server**, which determines which replica is the primary

handling primary failure



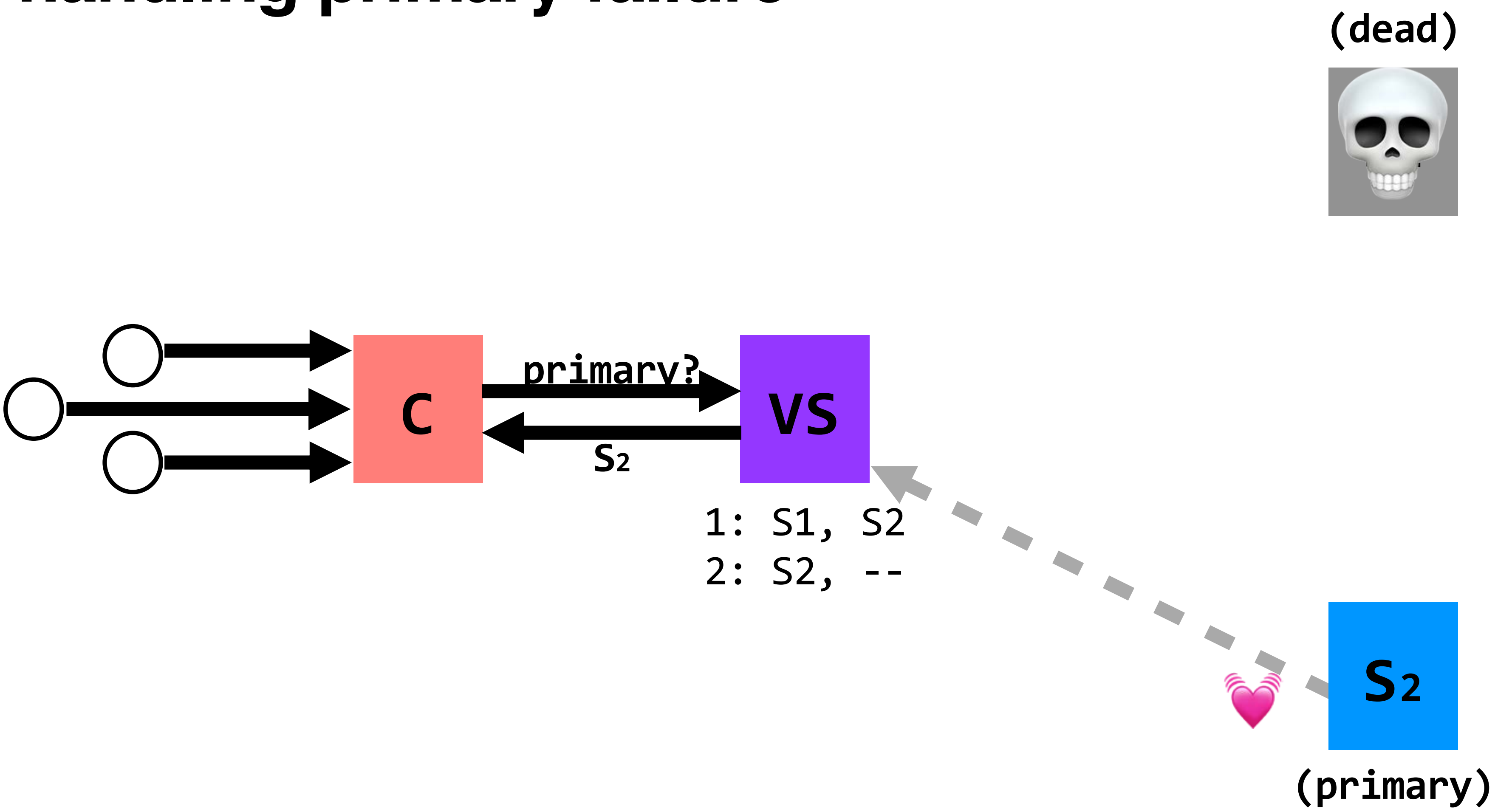
lack of pings indicates to VS that S_1 is down

handling primary failure

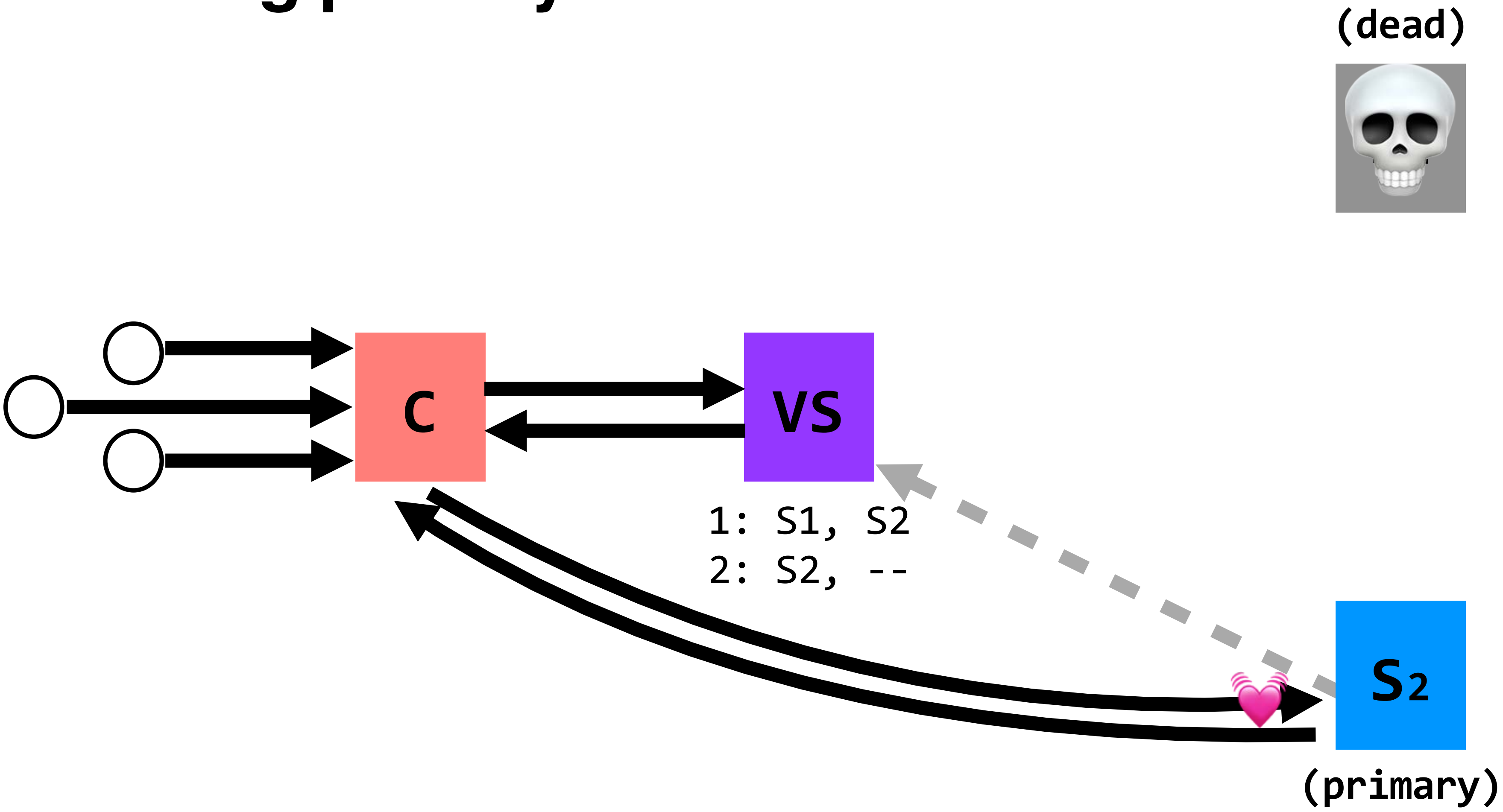


```
1: S1, S2
2: S2, --
```

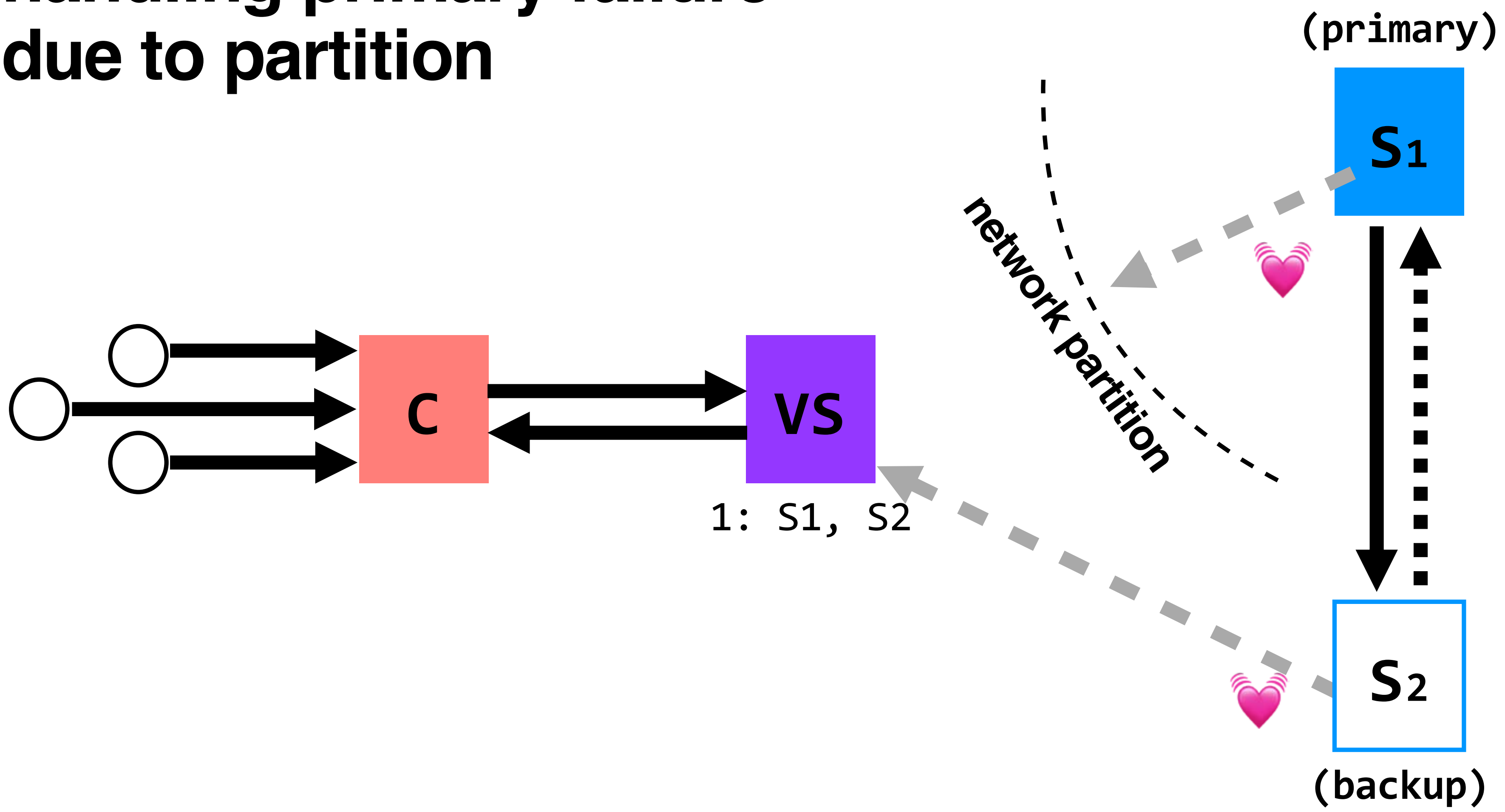
handling primary failure



handling primary failure

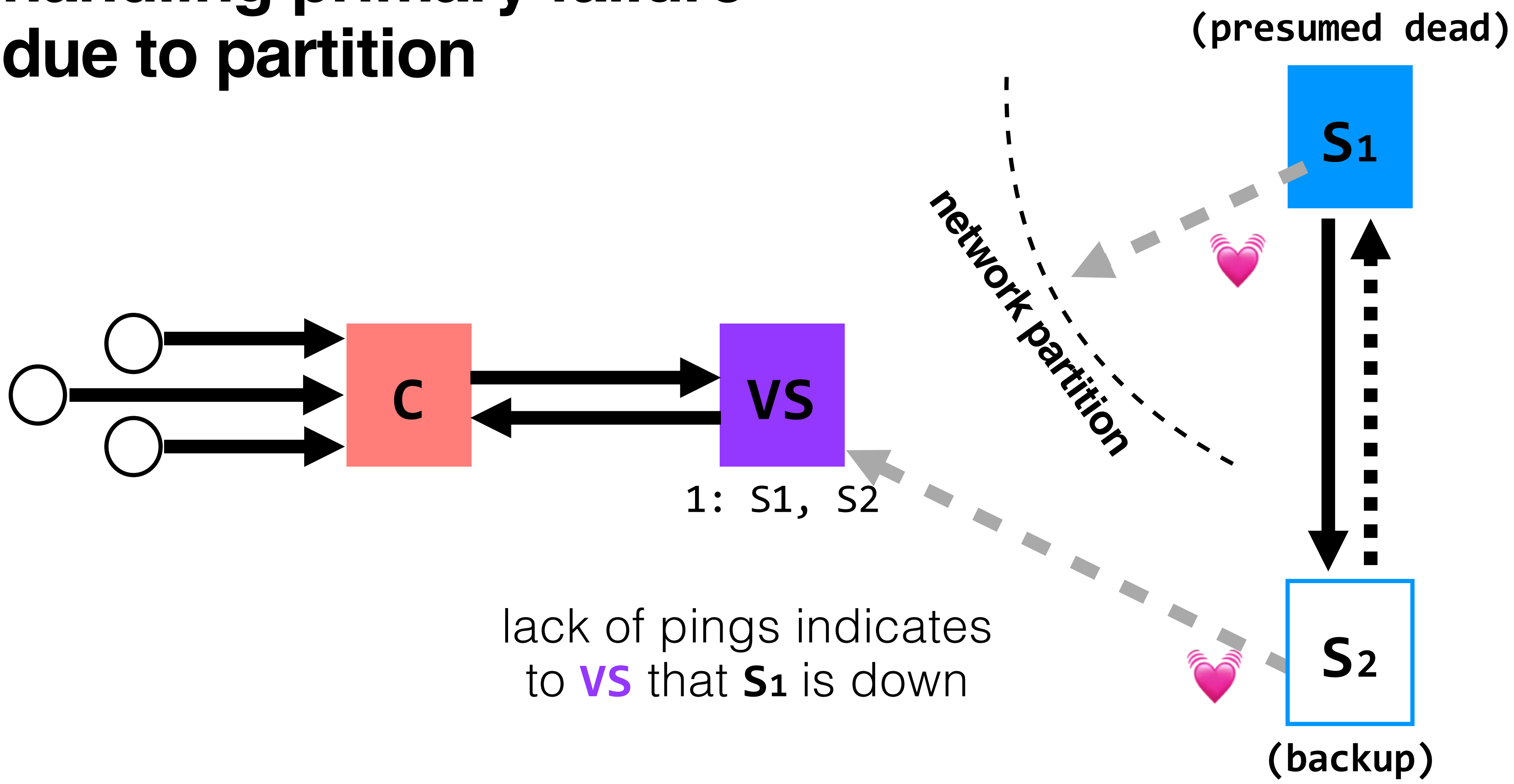


handling primary failure due to partition



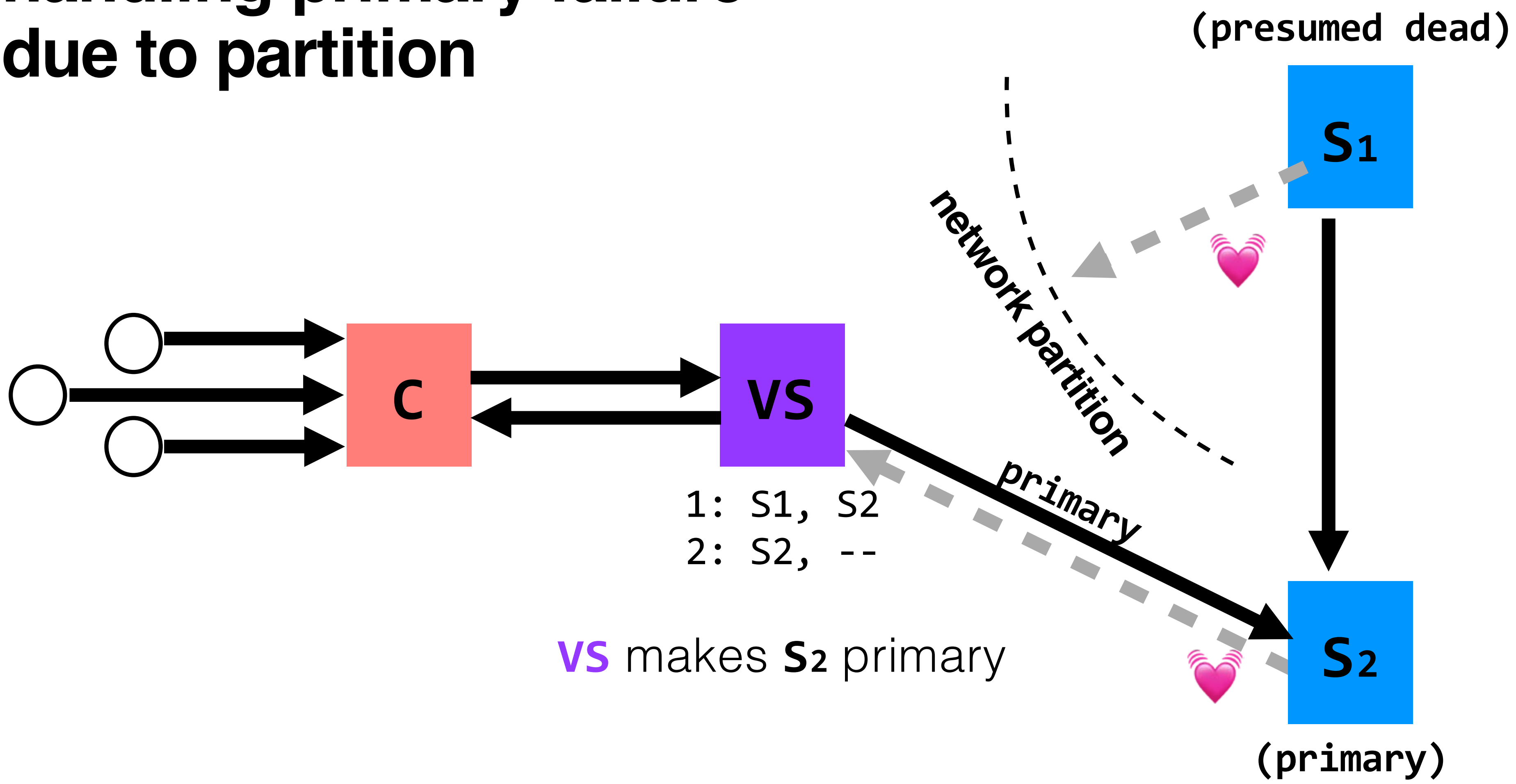
pose a partition keeps **S1** from communicating with the **view se**

handling primary failure due to partition



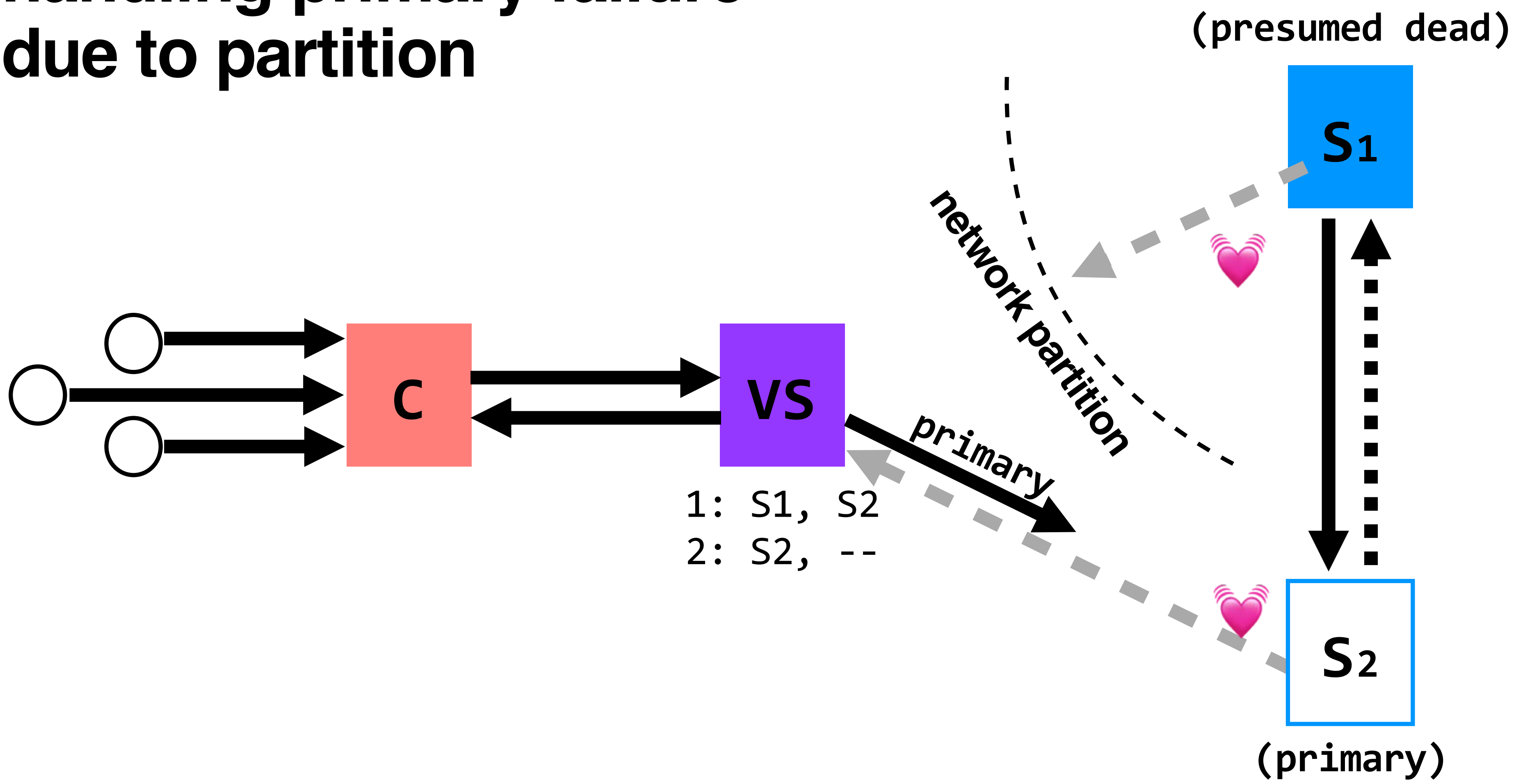
lack of pings indicates to **VS** that **S1** is down

handling primary failure due to partition



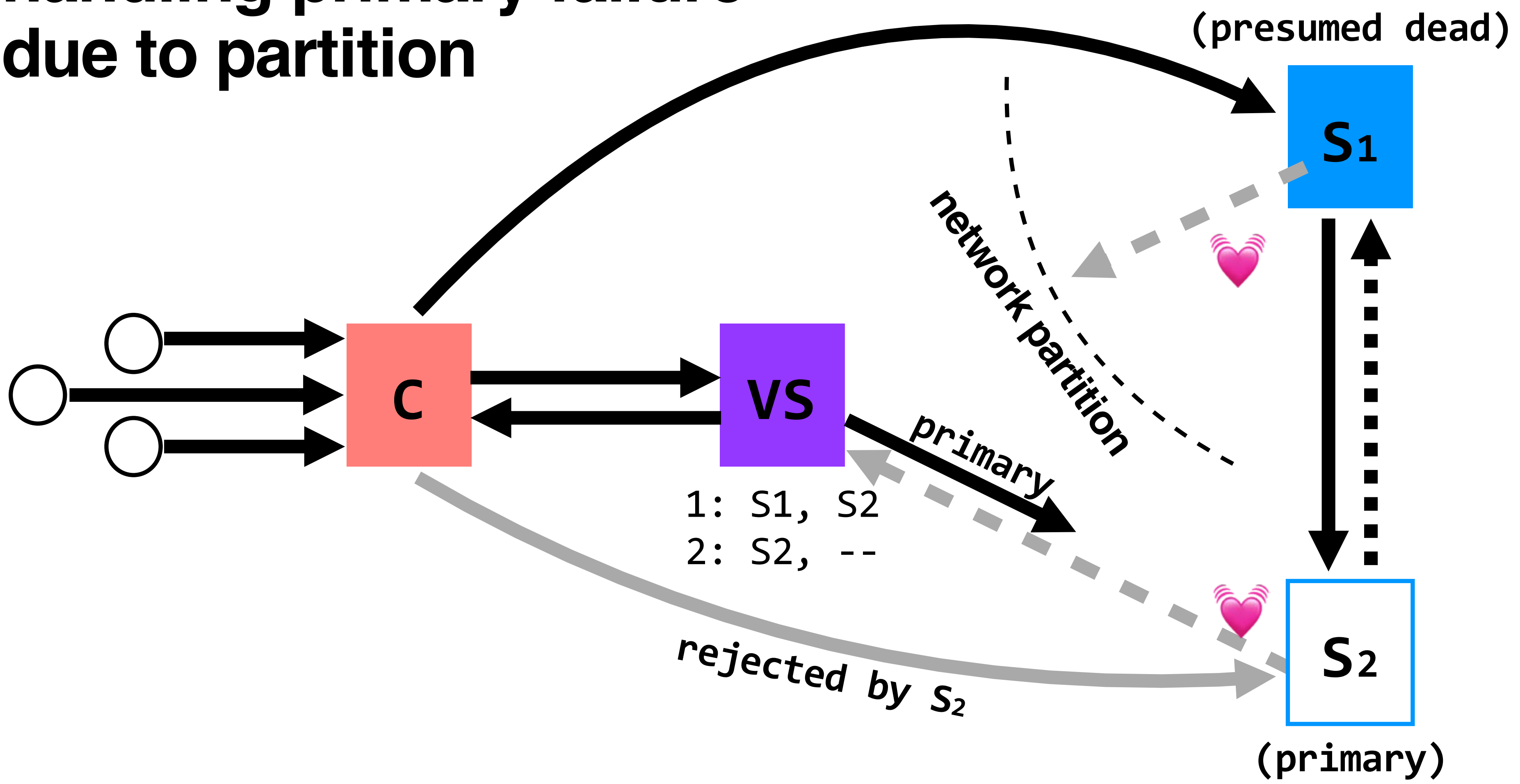
VS makes S2 primary

handling primary failure due to partition



question: what happens before S2 knows it's the primary?

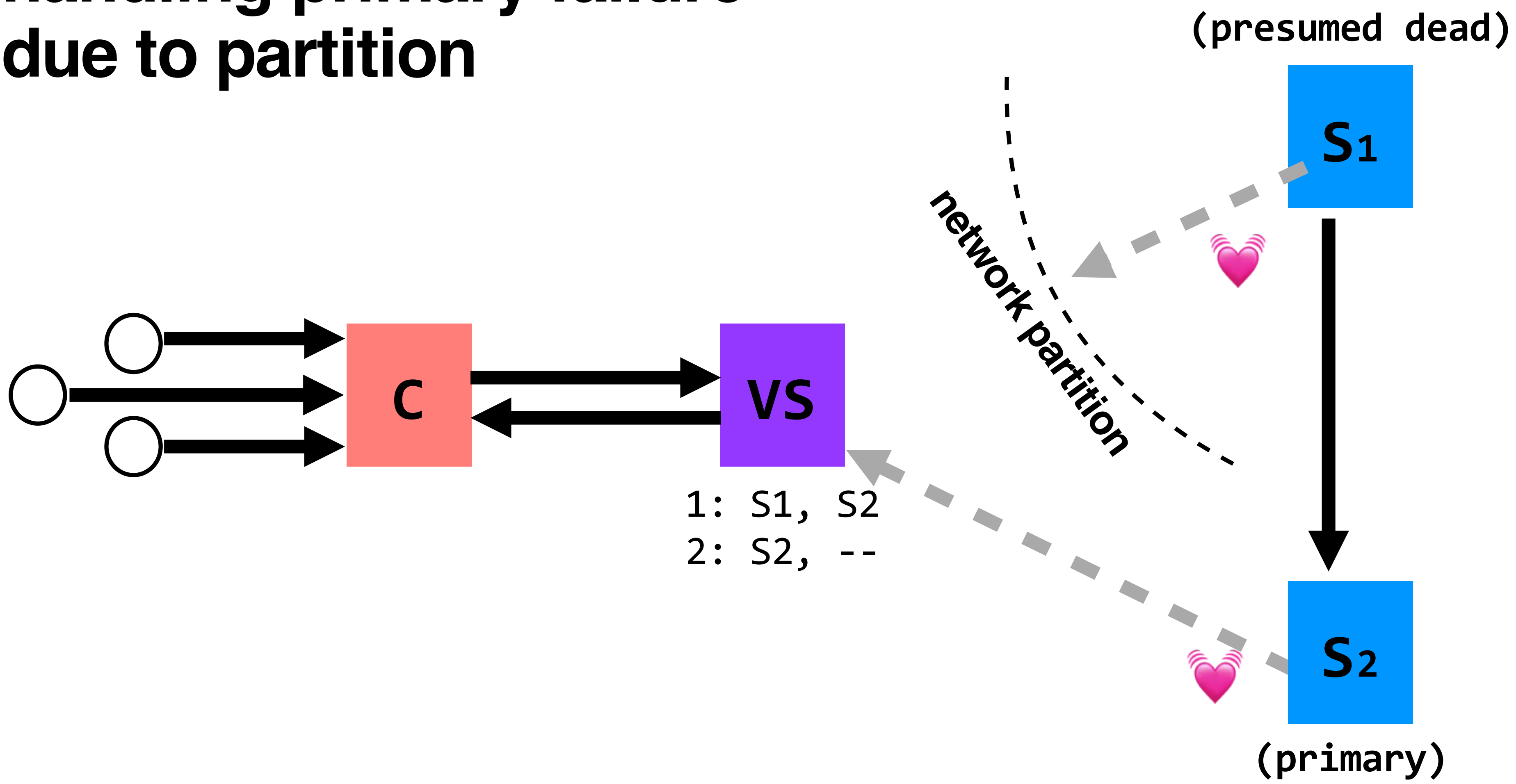
handling primary failure due to partition



S2 will act as backup

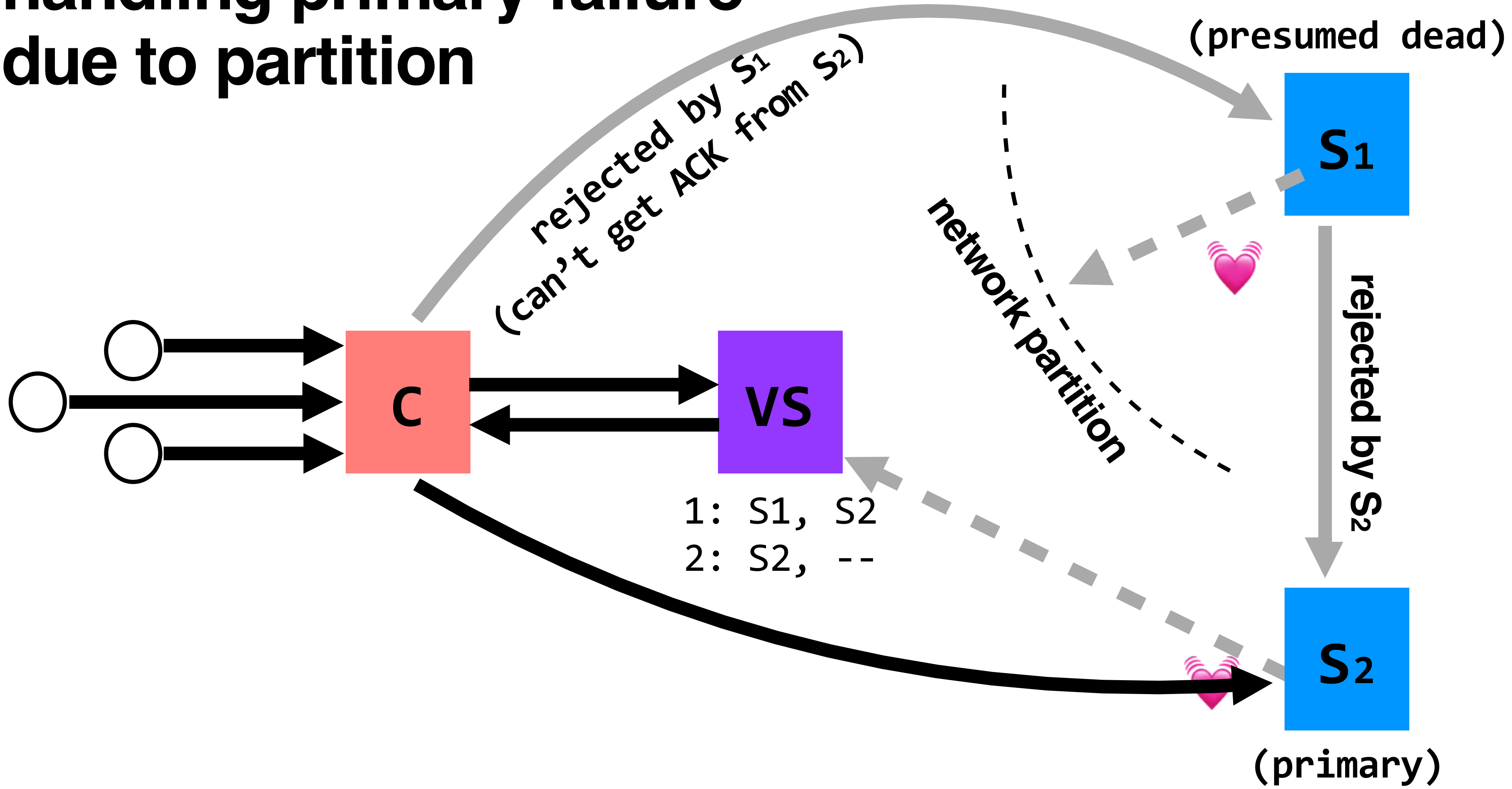
(accept updates from S1, reject coordinator requests)

handling primary failure due to partition



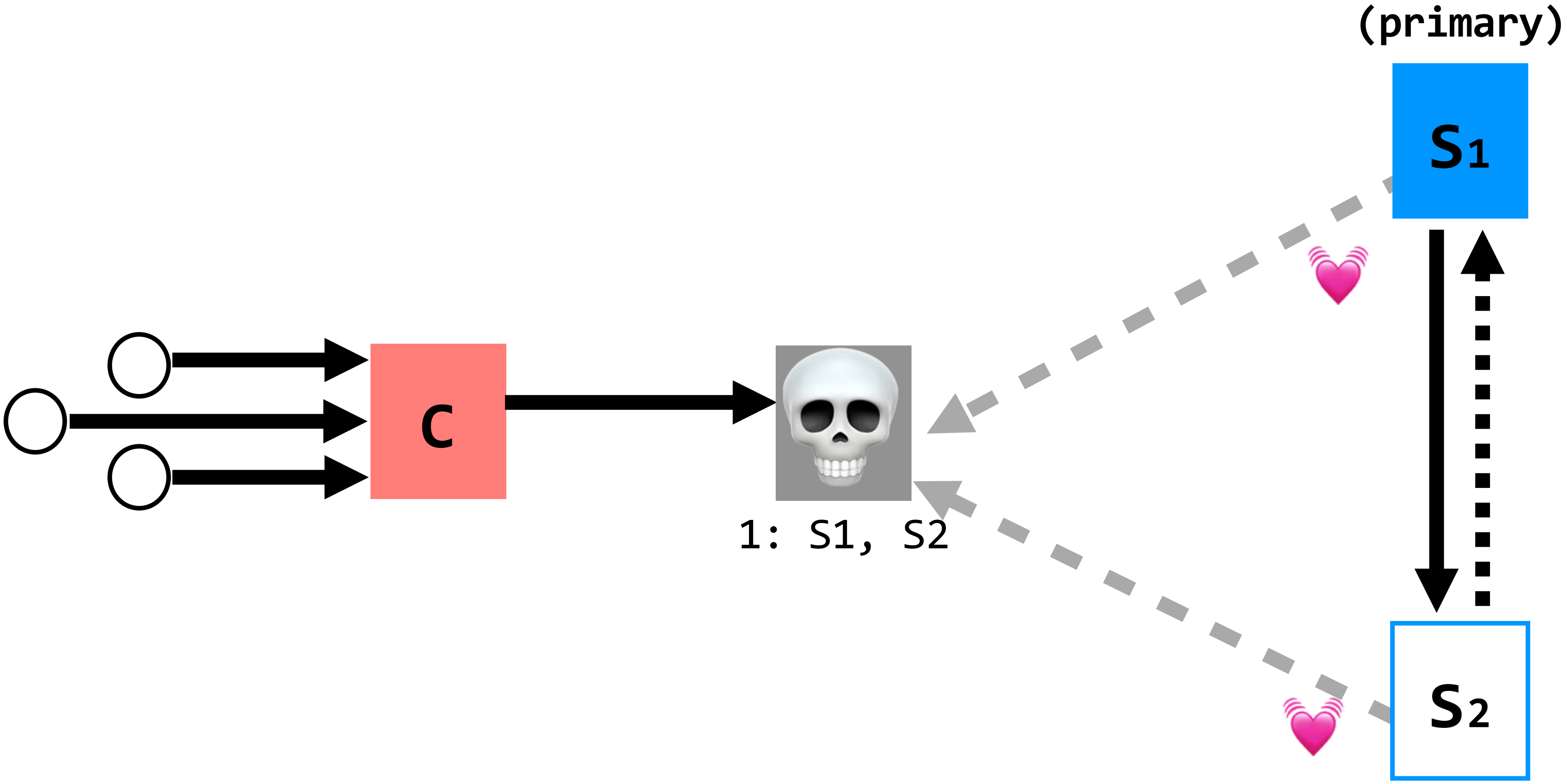
question: what happens after S2 knows it's the primary, but S1 also thinks it is?

handling primary failure due to partition

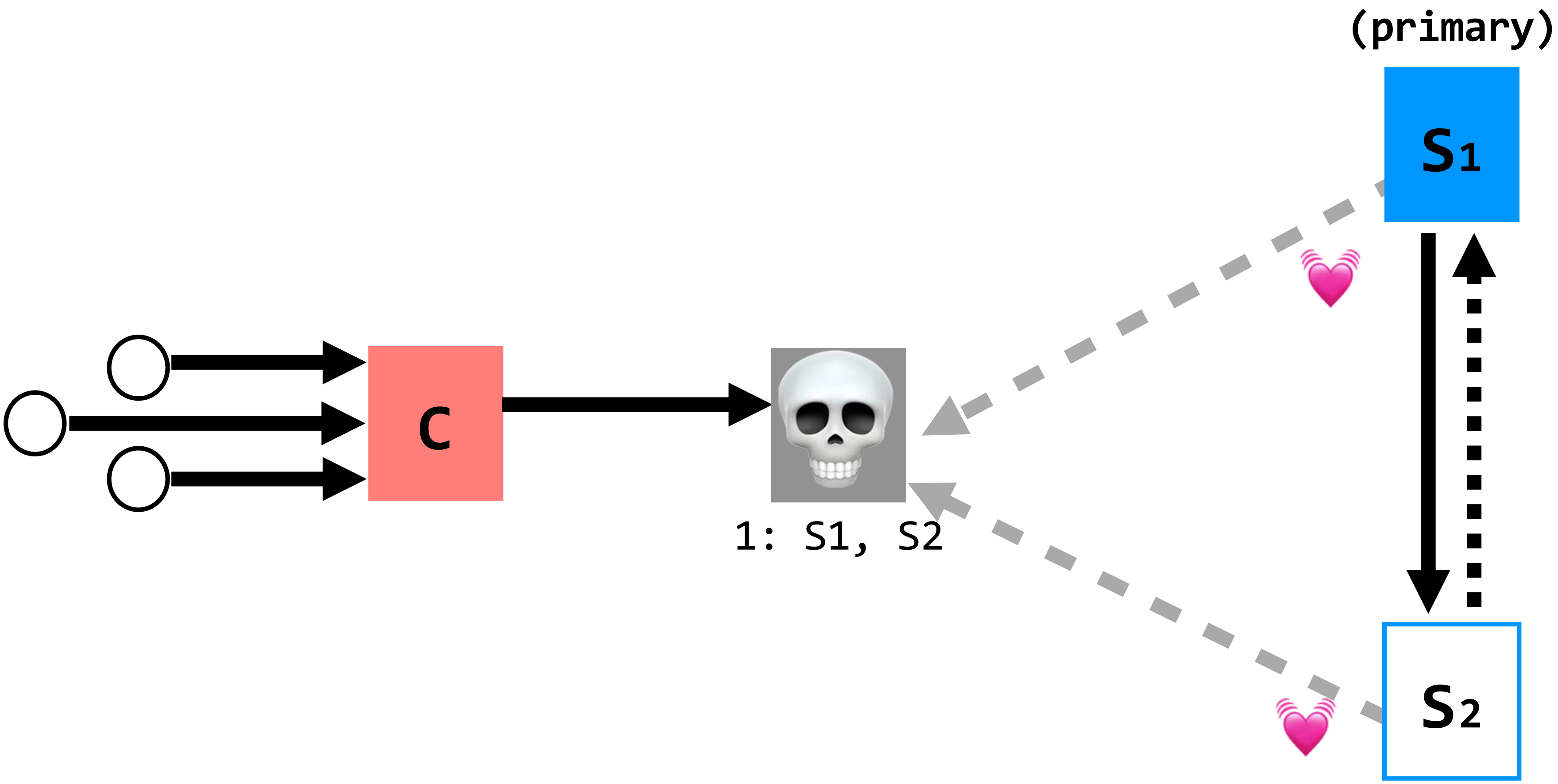


S₁ won't be able to act as primary

(can't accept client requests because it won't get ACKs from S₂)



problem: what if view server fails?



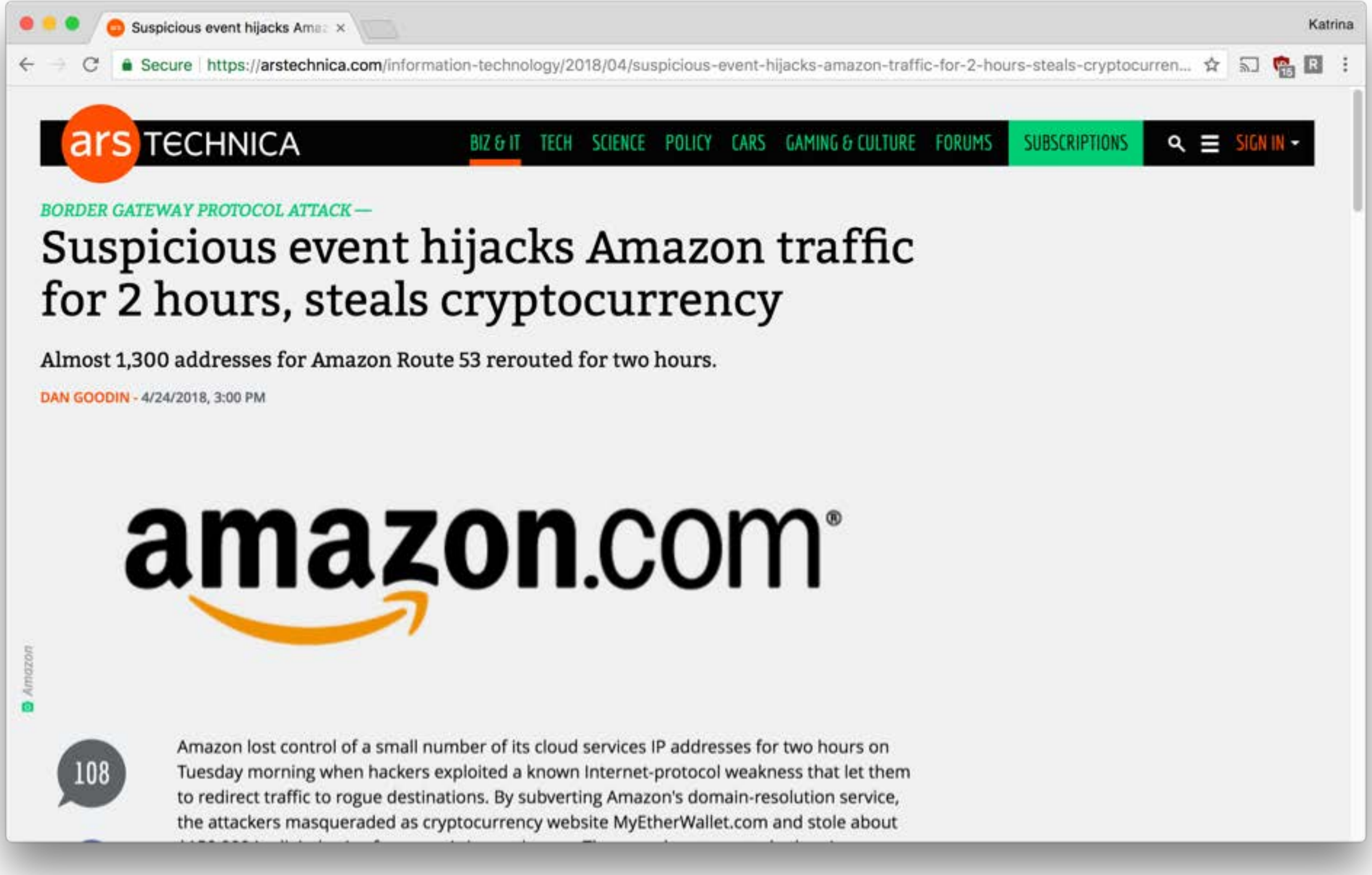
problem: what if view server fails?

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 Amaz ×
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
amazon.com®
Amazon
108

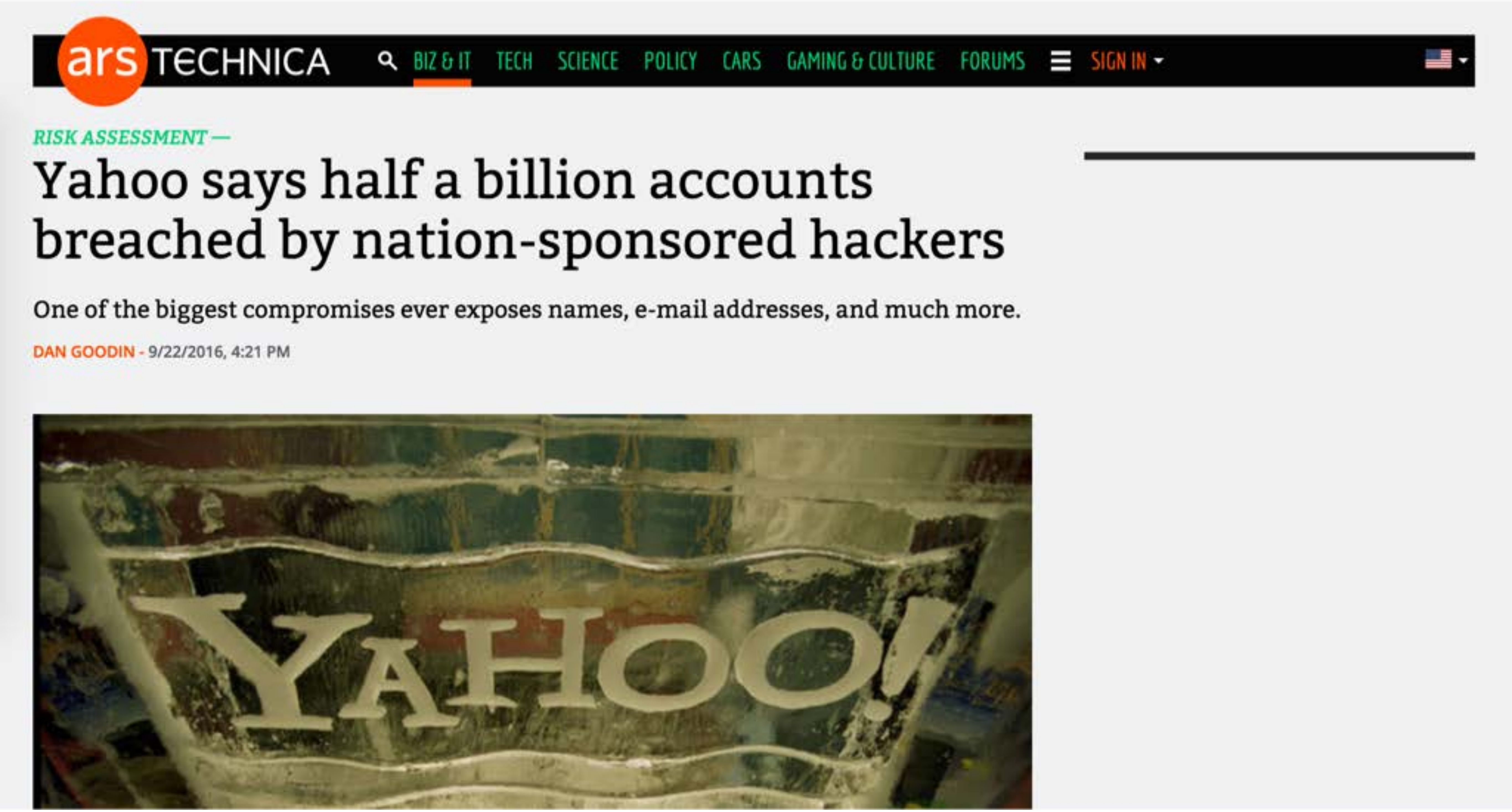
BORDER GATEWAY PROTOCOL ATTACK —

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 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

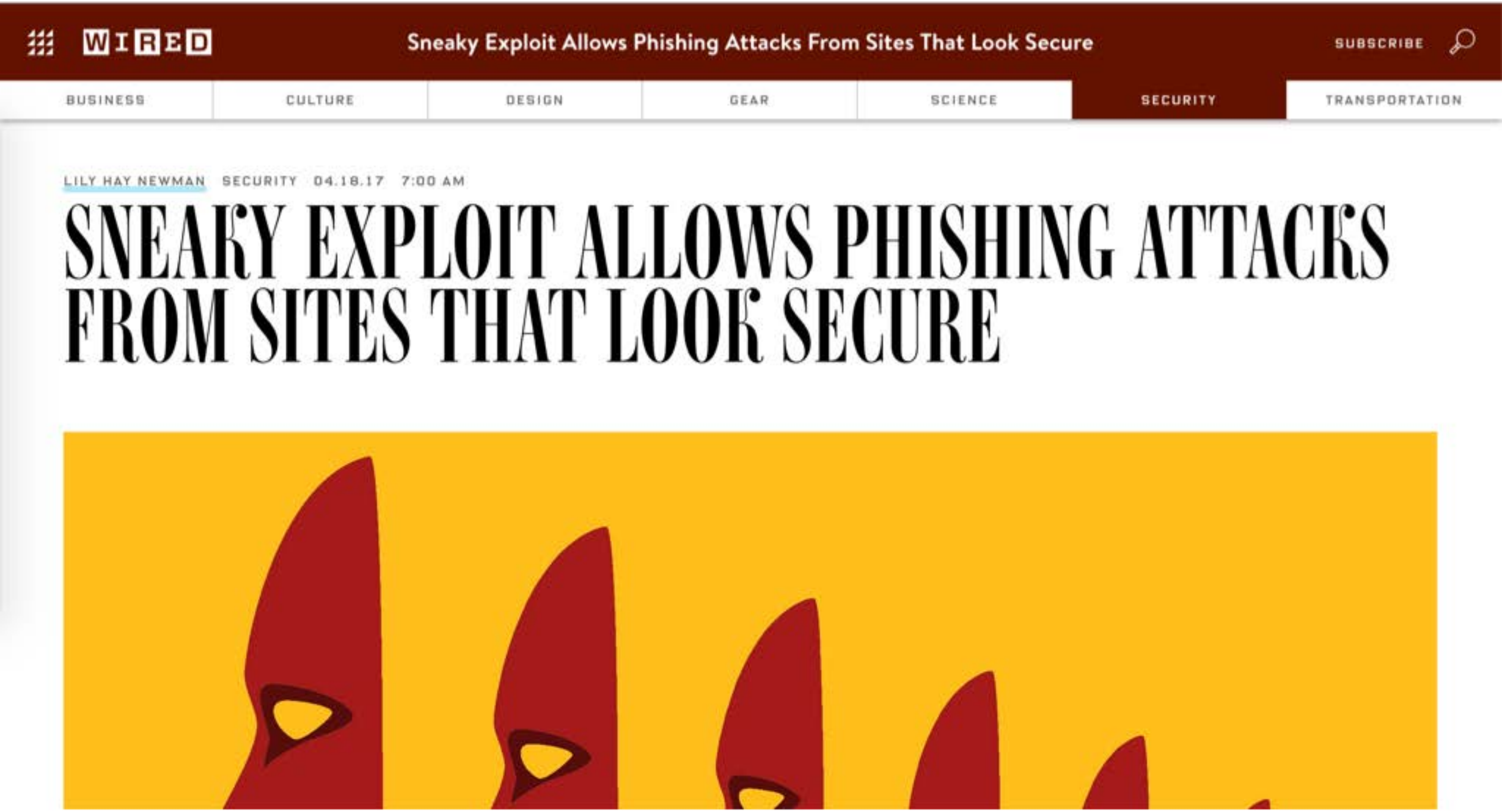


RISK ASSESSMENT —

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



WIRED Sneaky Exploit Allows Phishing Attacks From Sites That Look Secure SUBSCRIBE

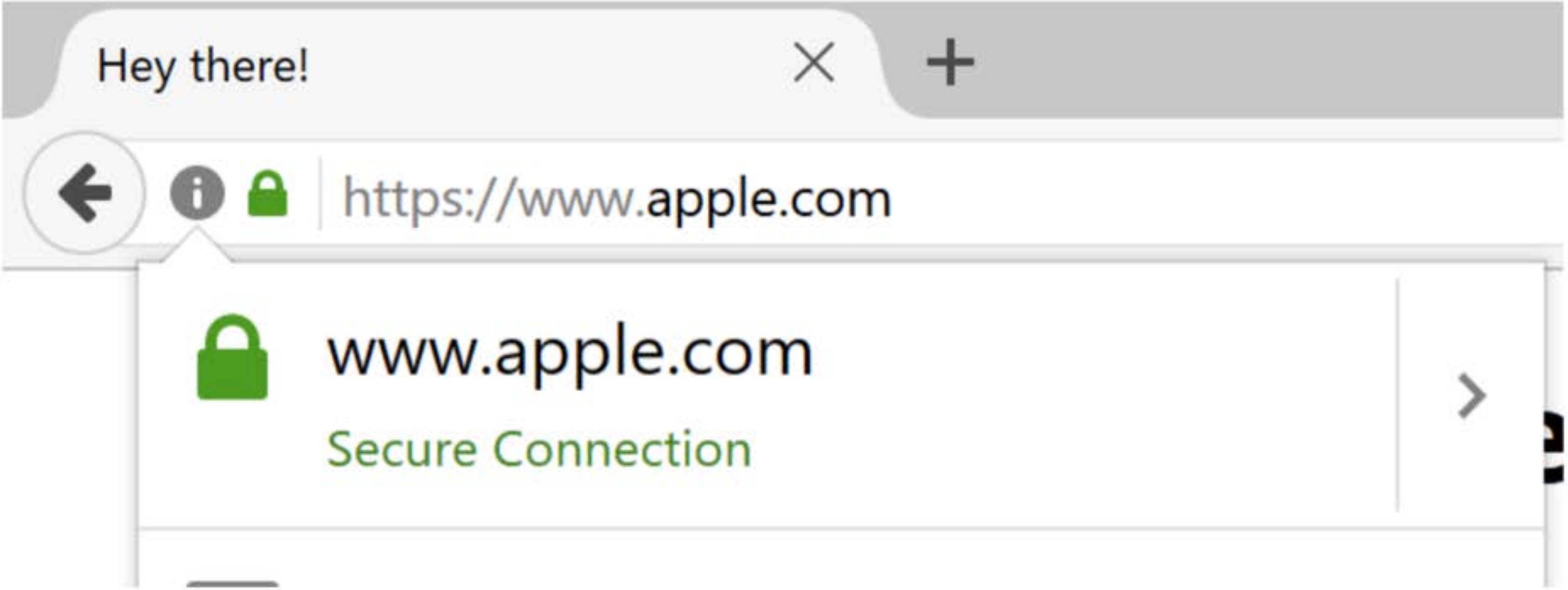
BUSINESS CULTURE DESIGN GEAR SCIENCE SECURITY TRANSPORTATION

LILY HAY NEWMAN SECURITY 04.18.17 7:00 AM

SNEAKY EXPLOIT ALLOWS PHISHING ATTACKS FROM SITES THAT LOOK SECURE

Phishing with Unicode Domains

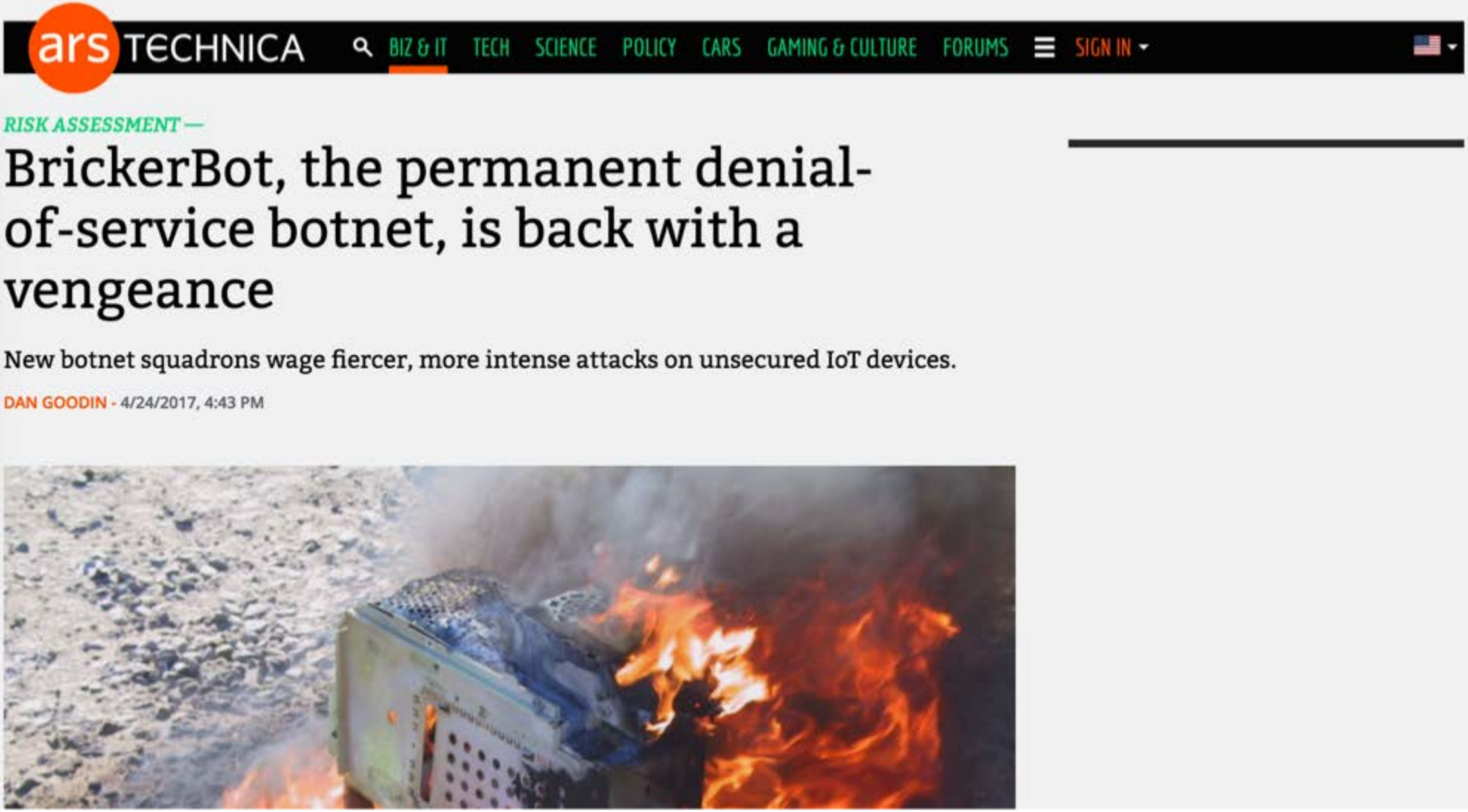
Posted by Xudong Zheng on April 14, 2017



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-

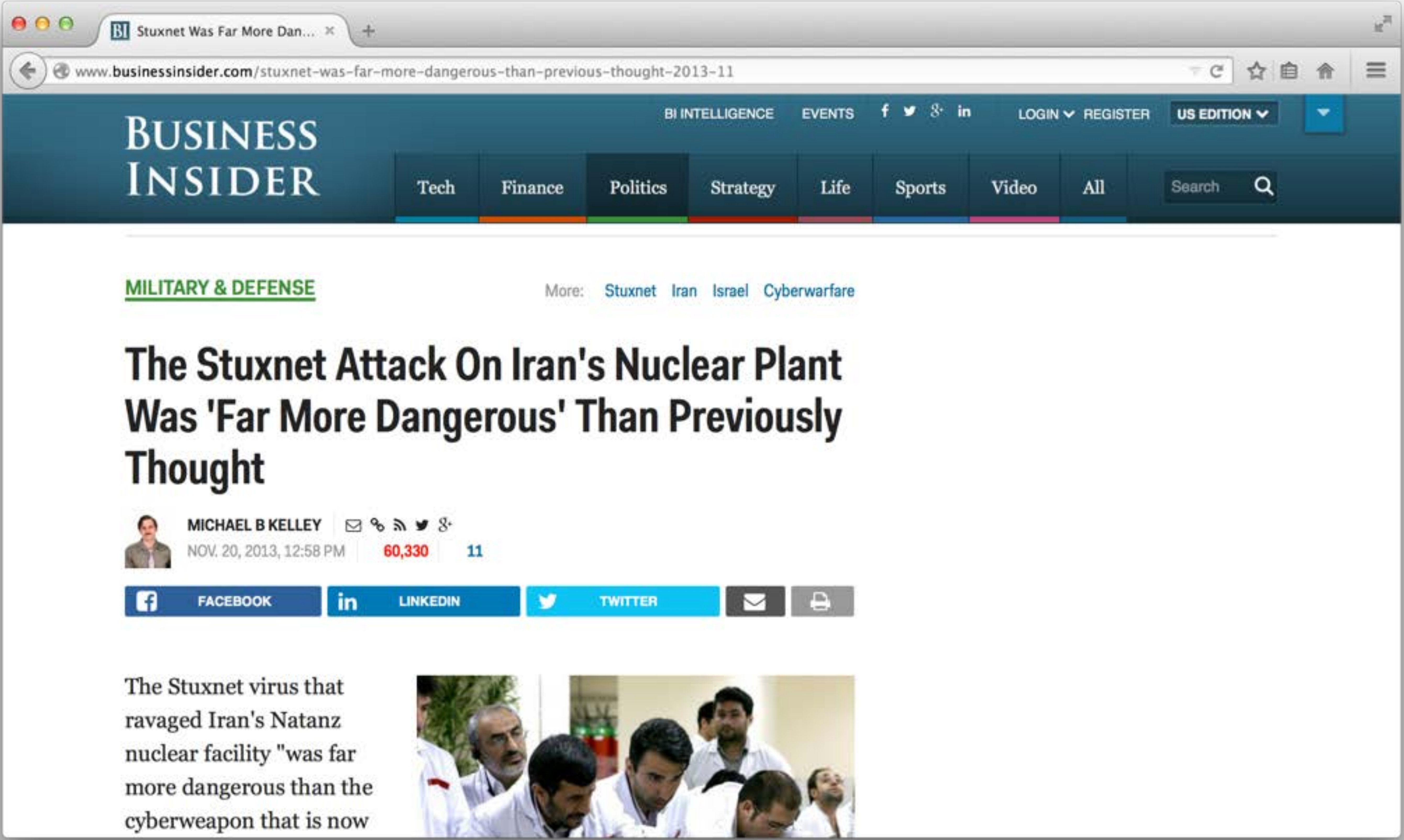


RISK ASSESSMENT —

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

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

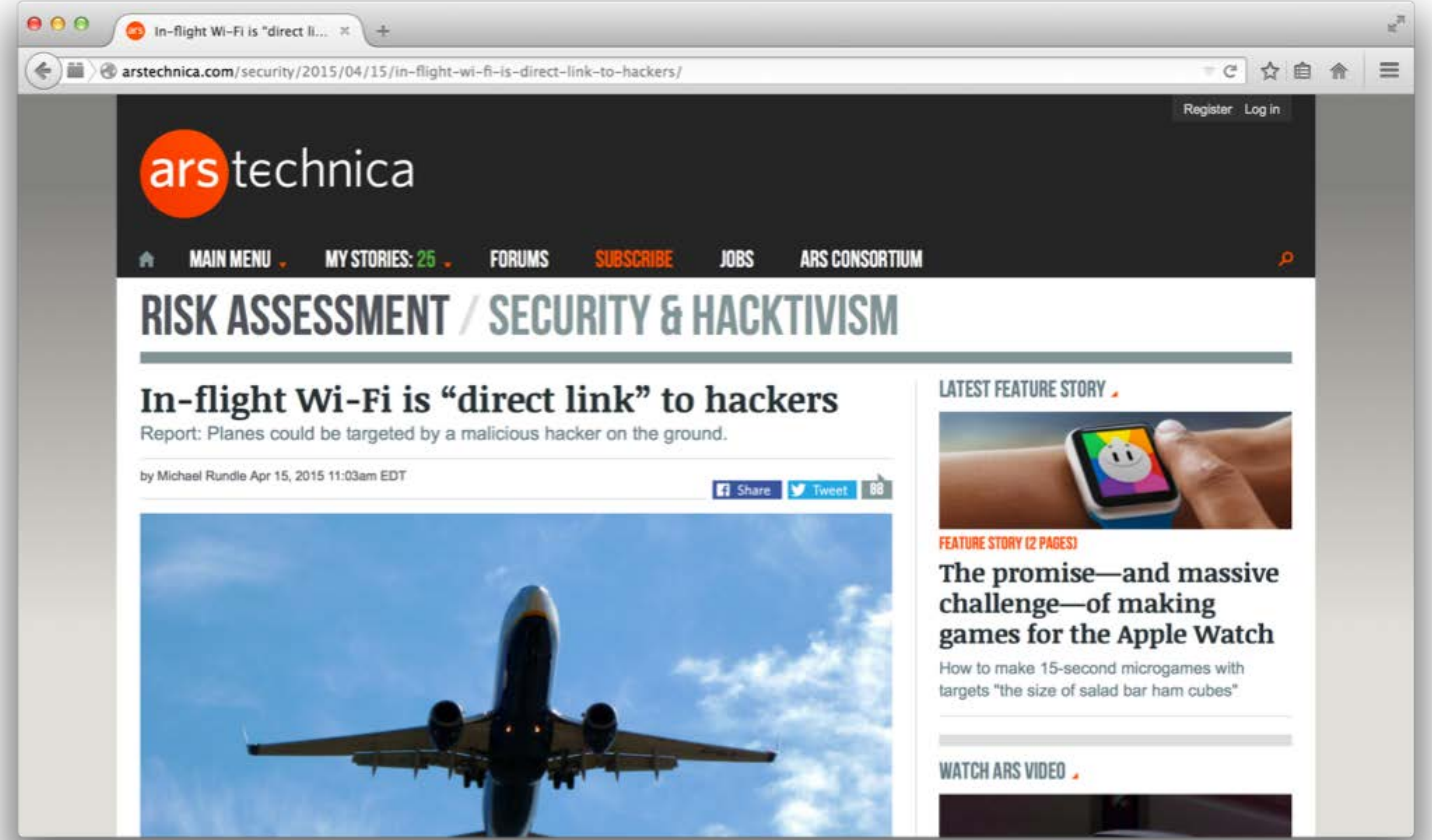
DAN GOODIN - 4/24/2017, 4:43 PM



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

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



RISK ASSESSMENT / SECURITY & HACKTIVISM

In-flight Wi-Fi is “direct link” to hackers

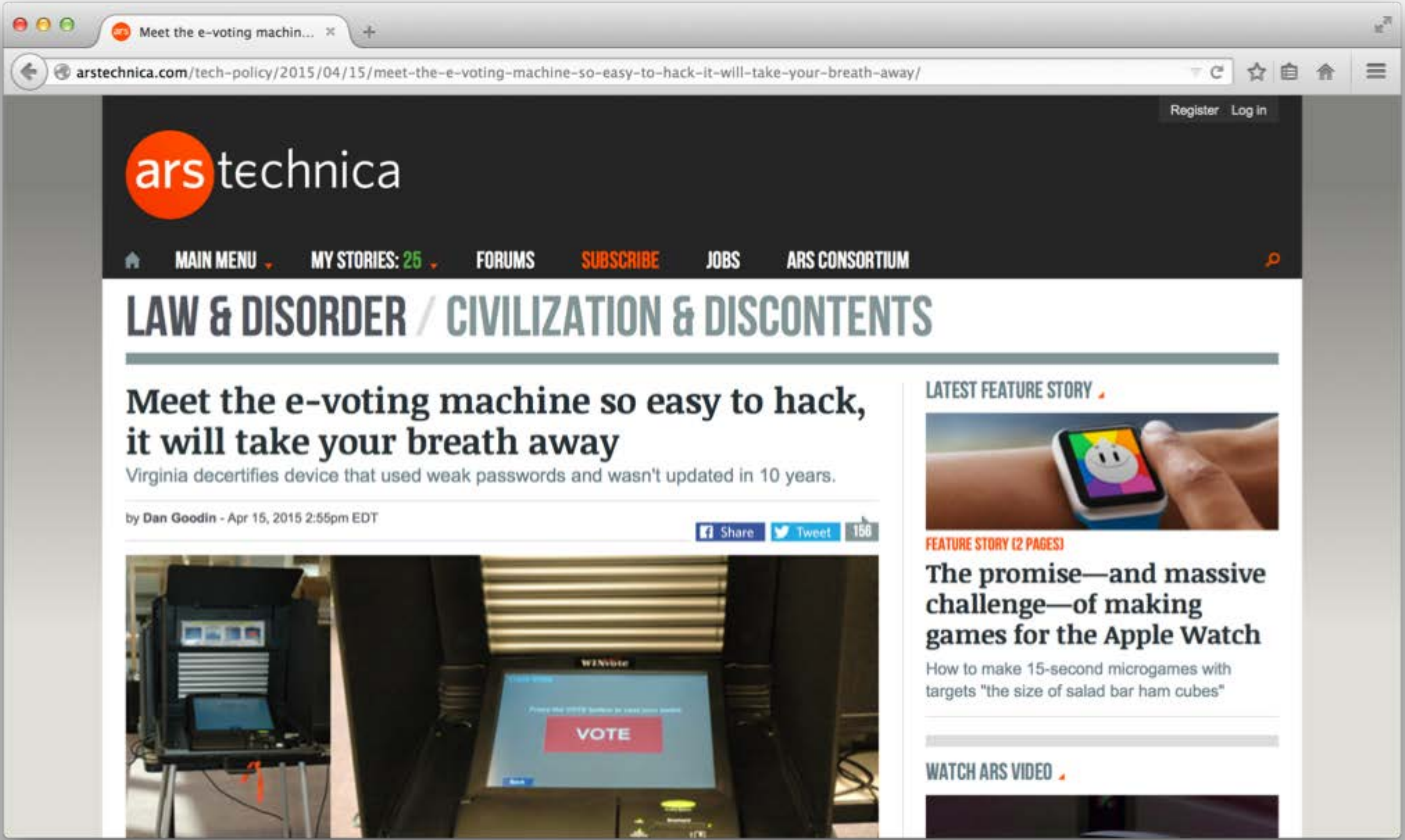
Report: Planes could be targeted by a malicious hacker on the ground.

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

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"



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
Share
Tweet
156
WINvote
VOTE

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

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

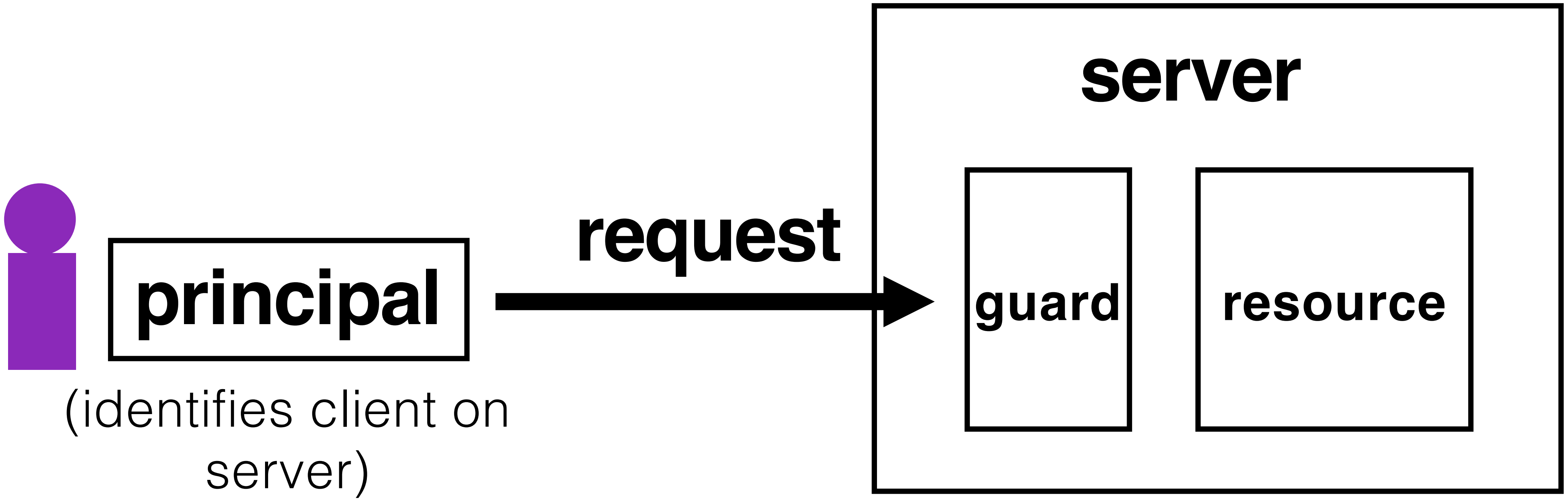
what makes computer security special?

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.

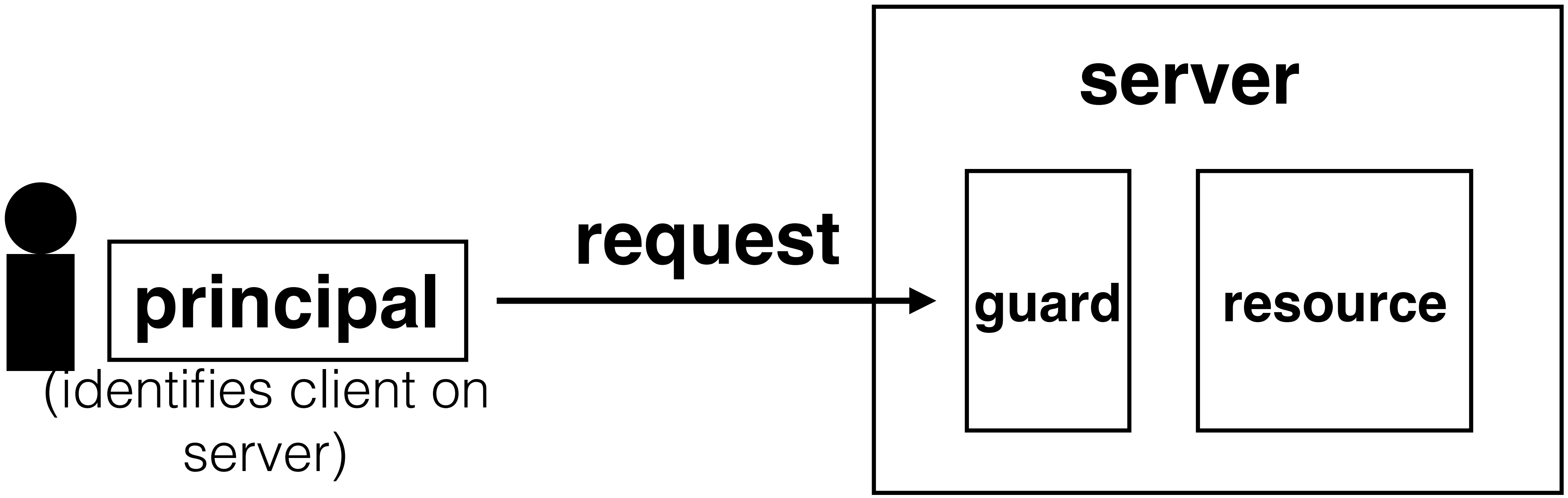
- Securing a system starts by specifying our goals (**policy**) and assumptions (**threat model**).

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

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
```

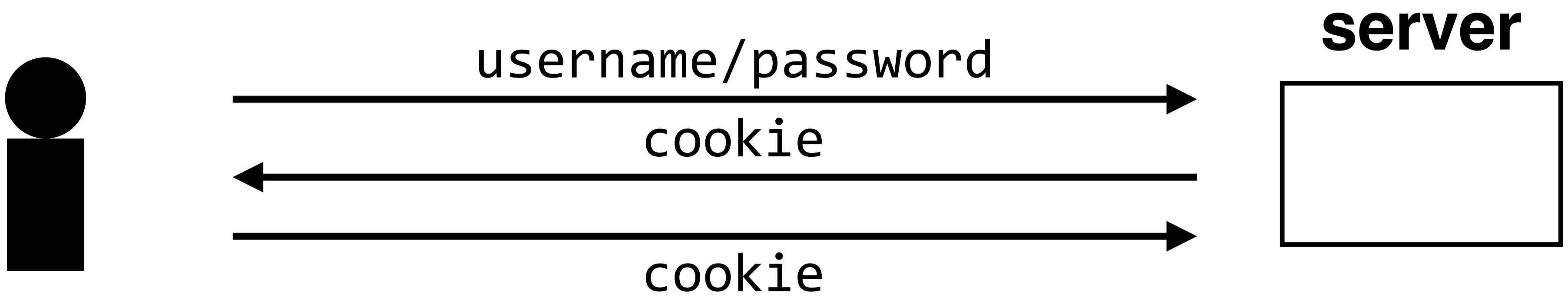
problem: adversary can still create rainbow tables for the most common passwords stored in plaintext

```
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
```

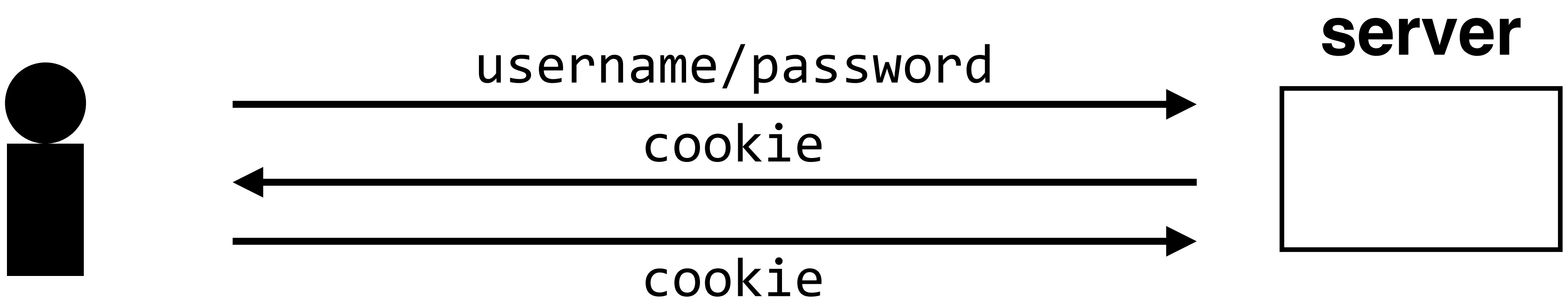
adversary would need a separate rainbow table for every possible salt

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

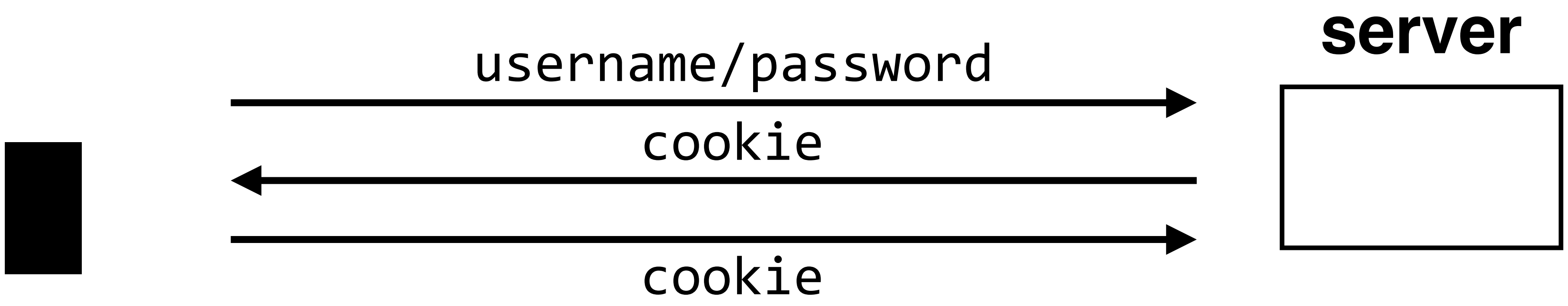
how can we avoid transmitting the password over and over?



`cookie = {username, expiration} ?`

problem: adversaries could easily create their own cookies

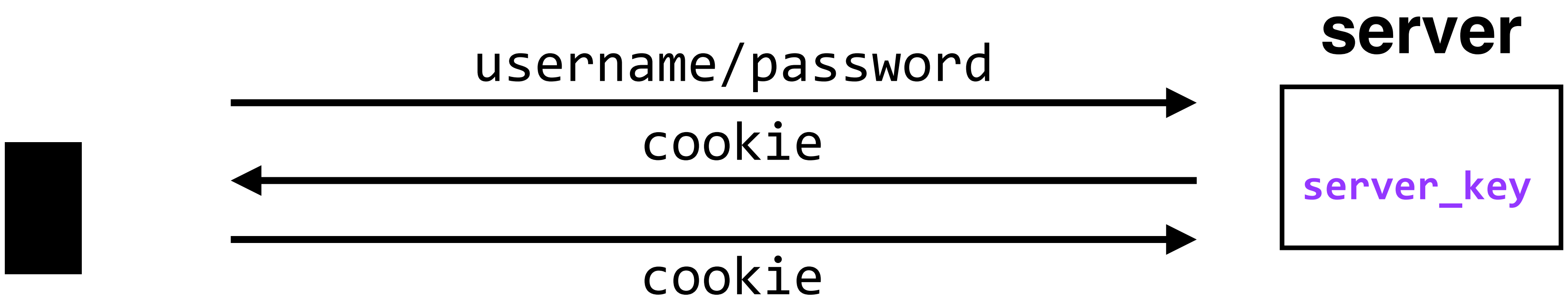
how can we avoid transmitting the password over and over?



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

problem: adversaries could still easily create their own cookies

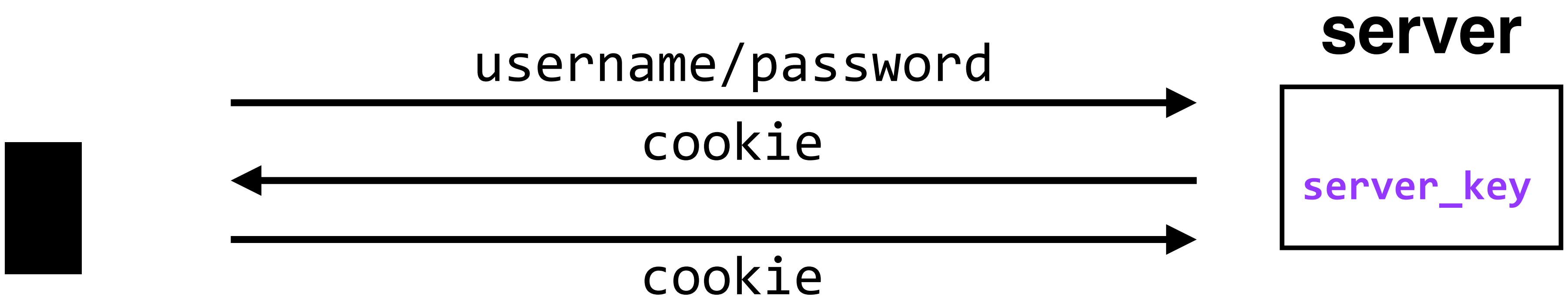
how can we avoid transmitting the password over and over?



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

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

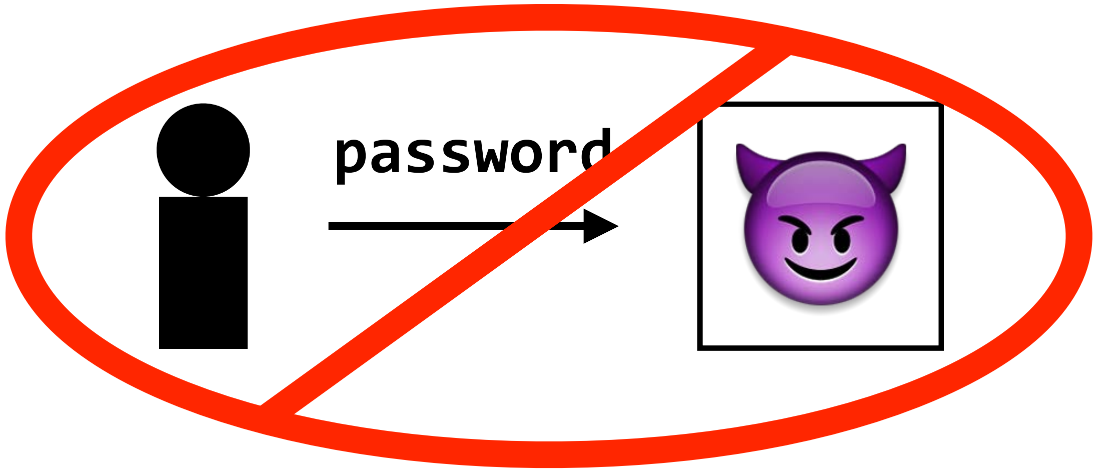
how can we avoid transmitting the password over and over?



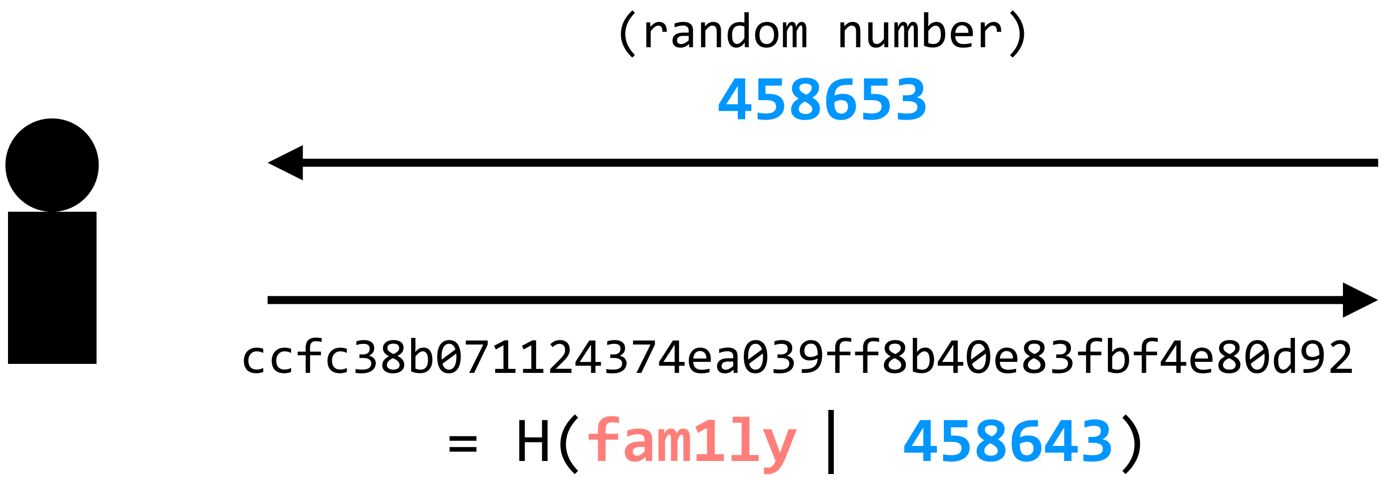
`{username, expiration, H(server_key | username | expiration)}`

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)

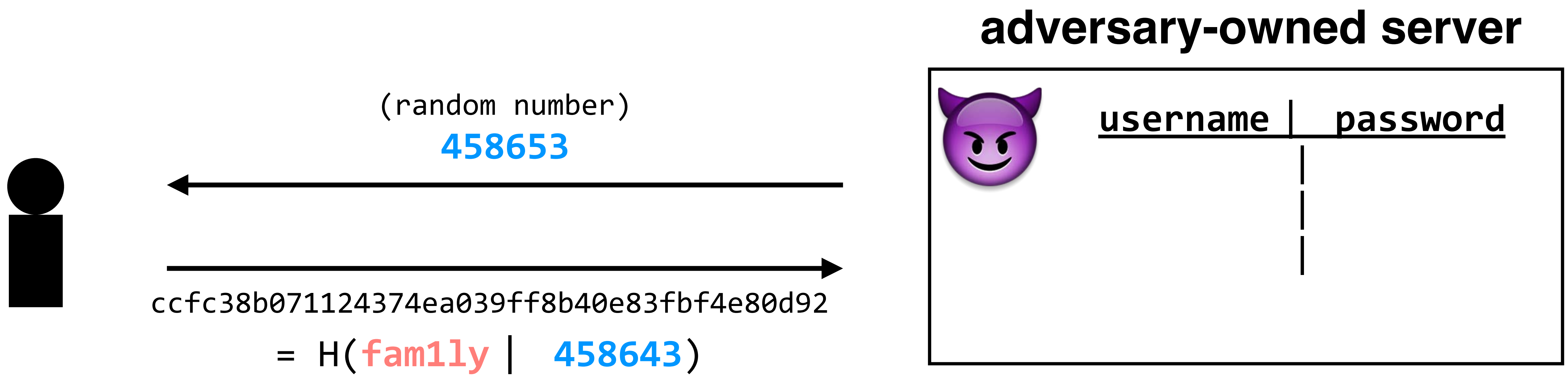
password is never sent directly

valid server

username	password
dom	fam1ly
han	dr1ftnNt0ky0
roman	Lamb0s4ever
tej	31173h4ck3r

server computes H(fam1ly | 458643) and checks

challenge-response protocol



adversary-owned server

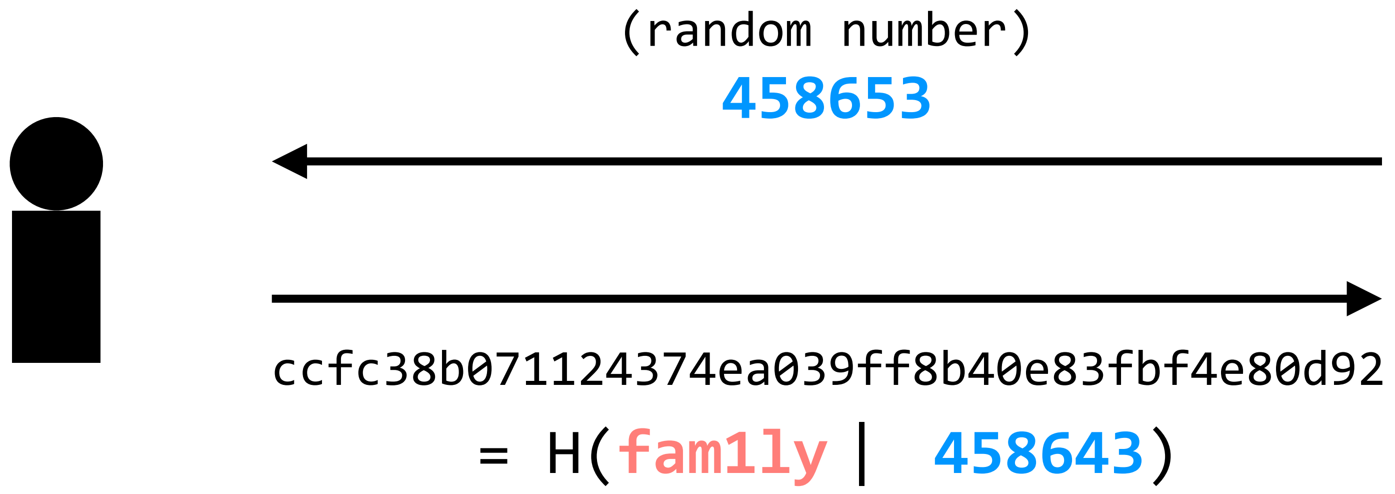
username | password

(random number)
458653

ccfc38b071124374ea039ff8b40e83fbf4e80d92
= H(fam1ly | 458643)

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

challenge-response protocol



(random number)

458653

ccfc38b071124374ea039ff8b40e83fbf4e80d92

= H(fam1ly | 458643)

valid server

username	password
dom	fam1ly
han	dr1ftnNt0ky0
roman	Lamb0s4ever
tej	31173h4ck3r

password is never sent directly

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?

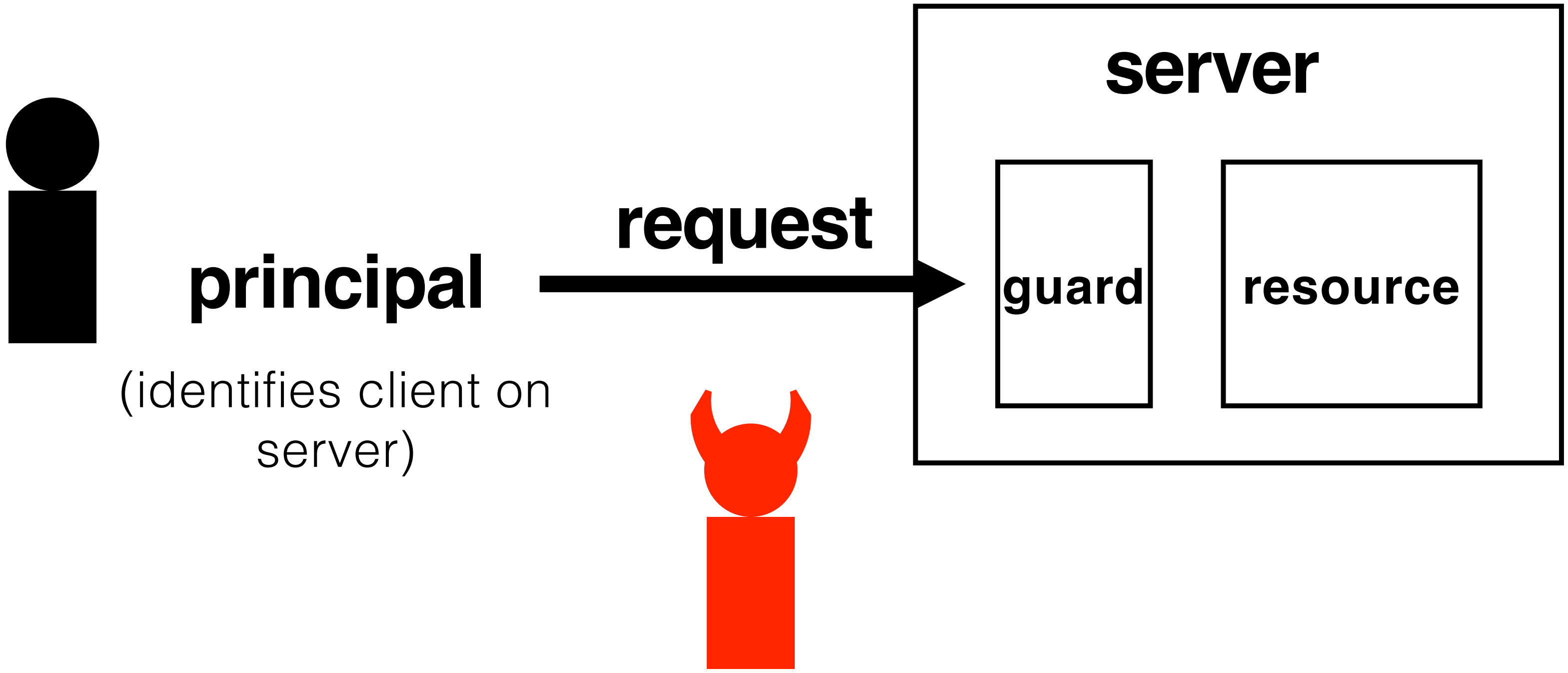
- Using passwords securely takes some effort. Storing **salted hashes**, incorporating **session cookies**, dealing with **phishing**, and **bootstrapping** are all concerns.

- 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.

Secure Channels

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/


server
request
principal
(identifies client on
server)
guard
resource

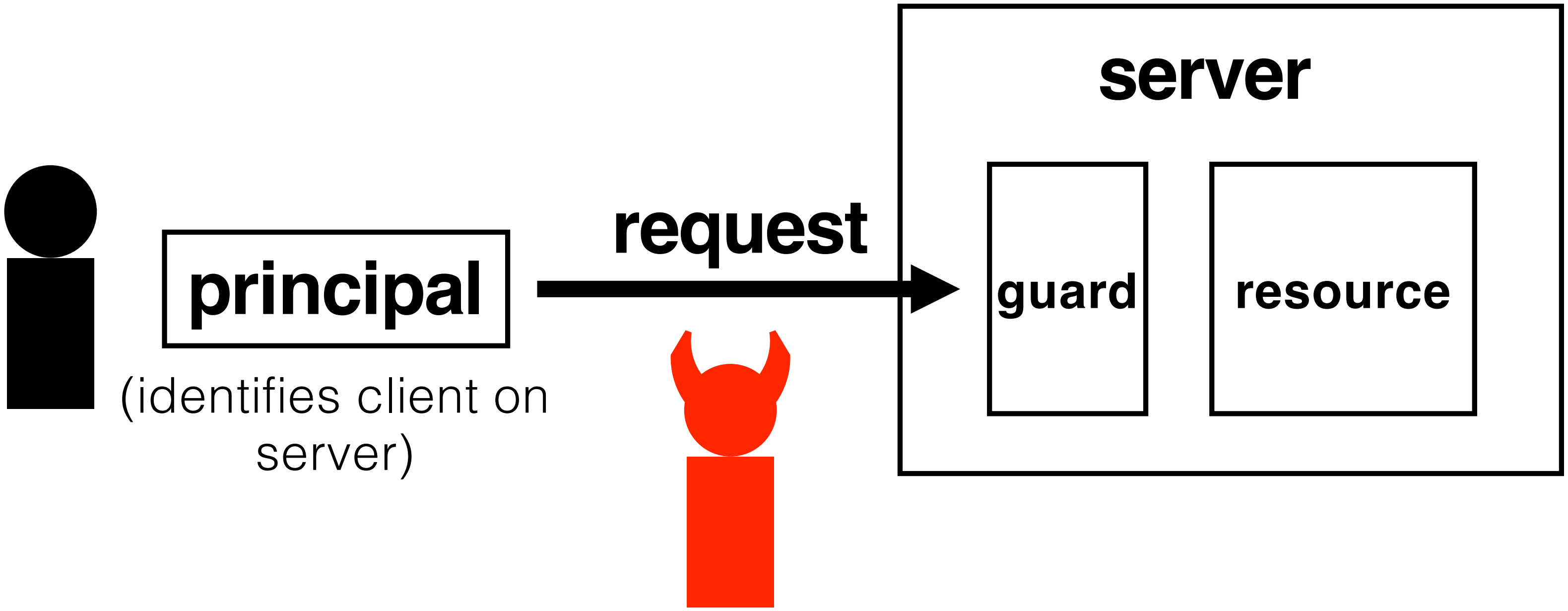
```
14:49:19.858386 2805536312us tsft -95dB noise antenna 1 5785 MHz 11a ht/40+ [bit 20] CF +QoS IP
17.253.11.201.80 > 10.189.53.19.54191: Flags [.], seq 3088997:3090365, ack 0, win 124, options [nop,nop,TS
val 295799082 ecr 1238603892], length 1368: HTTP
    0x0000:  aaaa 0300 0000 0800 4500 058c 37fd 4000  ........E...7.@.
    0x0010:  3b06 a4d9 11fd 0bc9 0abd 3513 0050 d3af  ;.........5..P..
    0x0020:  f692 6b9d 0186 6995 8010 007c 60b6 0000  ..k...i....|`...
    0x0030:  0101 080a 11a1 892a 49d3 9874 626a 6563  .......*I..tbjec
    0x0040:  7473 2e6e 6962 2e6d 6574 6155 5808 00e3  ts.nib.metaUX...
    0x0050:  8ee3 5a89 29e3 5a50 4b01 021e 0314 0000  ..Z.).ZPK.......
    0x0060:  0863 00b7 359b 4c5e bd8f e3c1 0900 00e9  .c..5.L^........
    0x0070:  1200 0079 000c 0000 0000 0000 0000 40a4  ...y..........@.
    0x0080:  814c ab1c 0650 6179 6c6f 6164 2f68 696c  .L...Payload/hil
    0x0090:  6c64 6173 6832 2e61 7070 2f48 7355 4952  ldash2.app/HsUIR
    0x00a0:  6573 6f75 7263 6542 756e 646c 652e 6275  esourceBundle.bu
    0x00b0:  6e64 6c65 2f68 7353 7570 706f 7274 4d61  ndle/hsSupportMa
    0x00c0:  696e 2e73 746f 7279 626f 6172 6463 2f78  in.storyboardc/x
    0x00d0:  7965 2d32 722d 456a 6b2d 7669 6577 2d38  ye-2r-Ejk-view-8
    0x00e0:  394e 2d70 532d 3437 647e 6970 6164 2e6e  9N-pS-47d~ipad.n
    0x00f0:  6962 2f72 756e 7469 6d65 2e6e 6962 5558  ib/runtime.nibUX
    0x0100:  0800 e38e e35a 8929 e35a 504b 0102 1e03  .....Z.).ZPK....
    0x0110:  1400 0008 0000 b735 9b4c 5cf6 7335 8500  .......5.L\.s5..
    0x0120:  0000 8500 0000 7e00 0c00 0000 0000 0000  ......~.........
    0x0130:  0040 a481 b4b5 1c06 5061 796c 6f61 642f  .@......Payload/
    0x0140:  6869 6c6c 6461 7368 322e 6170 702f 4873  hilldash2.app/Hs
    0x0150:  5549 5265 736f 7572 6365 4275 6e64 6c65  UIResourceBundle
    0x0160:  2e62 756e 646c 652f 6873 5375 7070 6f72  .bundle/hsSuppor
    0x0170:  744d 6169 6e2e 7374 6f72 7962 6f61 7264  tMain.storyboard
    0x0180:  632f 7879 652d 3272 2d45 6a6b 2d76 6965  c/xye-2r-Ejk-vie
    0x0190:  772d 3839 4e2d 7053 2d34 3764 7e69 7061  w-89N-pS-47d~ipa
```

```
14:15:57.156383 731851825us tsft -95dB noise antenna 0 2412 MHz 11g ht/20 26.0 Mb/s MCS 3 20 MHz lon GI greenfield BCC FEC
[bit 20] CF +QoS IP dhcp-18-111-89-99
.dyn.mit.edu.57061 > 17.154.66.156.https: Flags [P.], seq 0:517, ack 1, win 8192, length 517

        0x0000:  aaaa 0300 0000 0800 4500 022d 9fd8 4000  ........E..-..@.
        0x0010:  4006 d8ea 126f 5963 119a 429c dee5 01bb  @....oYc..B.....
        0x0020:  f7f4 9d92 e59a 1614 5018 2000 ae38 0000  ........P....8..
        0x0030:  1603 0102 0001 0001 fc03 0359 077b 5d64  ...........Y.{]d
        0x0040:  6a53 0208 0cde 5c0a 26e8 5732 151d c778  jS....\.&.W2...x
        0x0050:  16c3 d1cc d5e6 c8a1 b940 3220 3ce6 c3c9  .........@2.<...
        0x0060:  ccb5 f523 3ae1 bf92 cd1f 1ac9 efc4 b155  ...#:..........U
        0x0070:  576a 4af8 4bc9 5b38 38dd 5d0e 0026 00ff  WjJ.K.[88.]..&..
        0x0080:  c02c c02b c024 c023 c00a c009 c030 c02f  .,.+.$.#.....0./
        0x0090:  c028 c027 c014 c013 009d 009c 003d 003c  .(.'.........=.<
        0x00a0:  0035 002f 0100 018d 0000 001d 001b 0000  .5./............
        0x00b0:  1870 3331 2d62 7579 2e69 7475 6e65 732e  .p31-buy.itunes.
        0x00c0:  6170 706c 652e 636f 6d00 0a00 0800 0600  apple.com.......
        0x00d0:  1700 1800 1900 0b00 0201 0000 0d00 1200  ................
        0x00e0:  1004 0102 0105 0106 0104 0302 0305 0306  ................
        0x00f0:  0333 7400 0000 1000 3000 2e02 6832 0568  .3t.....0...h2.h
        0x0100:  322d 3136 0568 322d 3135 0568 322d 3134  2-16.h2-15.h2-14
        0x0110:  0873 7064 792f 332e 3106 7370 6479 2f33  .spdy/3.1.spdy/3
        0x0120:  0868 7474 702f 312e 3100 0500 0501 0000  .http/1.1.......
        0x0130:  0000 0012 0000 0017 0000 0015 00f7 0000  ................
        0x0140:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0150:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0160:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0170:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0180:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0190:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x01a0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x01b0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x01c0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x01d0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x01e0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x01f0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0200:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0210:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0220:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0230:  0000 0000 00                             .....
```

```
14:05:50.087089 195784191us tsft bad-fcs -78dB signal -96dB noise antenna 1 5785 MHz 11a ht/40+ [bit 20]
CF +QoS IP 18.111.23.61.64677 > 104.199.110.216.80: Flag
s [P.], seq 1:323, ack 1, win 4136, options [nop,nop,TS val 605691701 ecr 1821306901], length 322: HTTP:
GET /img/inj9/b/p0k/x6jl.png HTTP/1.1
        0x0000:  aaaa 0300 0000 0800 4500 0176 a863 4000  ........E..v.c@.
        0x0010:  4006 8fd3 126f 173d 68c7 6ed8 fca5 0050  @....o.=h.n....P
        0x0020:  9d4a 295a 0fc9 838f 8018 1028 b54f 0000  .J)Z.......(.O..
        0x0030:  0101 080a 241a 1f35 6c8e f015 4745 5420  ....$..5l...GET.
        0x0040:  2f69 6d67 2f69 6e6a 392f 622f 7030 6b2f  /img/inj9/b/p0k/
        0x0050:  7836 6a6c 2e70 6e67 2048 5454 502f 312e  x6jl.png.HTTP/1.
        0x0060:  310d 0a48 6f73 743a 2069 6e6a 392e 6d6a  1..Host:.inj9.mj
        0x0070:  742e 6c75 0d0a 4163 6365 7074 3a20 696d  t.lu..Accept:.im
        0x0080:  6167 652f 706e 672c 696d 6167 652f 7376  age/png,image/sv
        0x0090:  672b 786d 6c2c 696d 6167 652f 2a3b 713d  g+xml,image/*;q=
        0x00a0:  302e 382c 2a2f 2a3b 713d 302e 350d 0a41  0.8,*/*;q=0.5..A
        0x00b0:  6363 6570 742d 4ce1 4d67 7561 6765 3a20  ccept-L.Mguage:.
        0x00c0:  656e 2d75 730d 0a43 6f6e 6e65 6374 696f  en-us..Connectio
        0x00d0:  6e3a 206b 6565 702d 616c 6976 650d 0a41  n:.keep-alive..A
        0x00e0:  6363 6570 742d 456e 636f 6469 6e67 3a20  ccept-Encoding:.
        0x00f0:  677a 6970 a18c 7b65 666c 6174 650d 0a55  gzip..{eflate..U
        0x0100:  7365 722d 4167 656e 743a 204d 6f7a 696c  ser-Agent:.Mozil
        0x0110:  6c61 2f35 2e30 2028 6950 686f 6e65 3b20  la/5.0.(iPhone;.
        0x0120:  4350 5520 6950 686f 6e65 204f 5320 3130  CPU.iPhone.OS.10
        0x0130:  5f33 5f31 206c 696b 6520 4d61 6320 4f53  _3_1.like.Mac.OS
        0x0140:  2058 2920 4170 706c 6557 6562 4b69 742f  .X).AppleWebKit/
        0x0150:  3630 332e 312e 3330 2028 4b48 544d 4c2c  603.1.30.(KHTML,
        0x0160:  206c 696b 6520 4765 636b 6f29 204d 6f62  .like.Gecko).Mob
        0x0170:  696c 652f 3134 4533 3034 0d0a 0d0a       ile/14E304....
14:05:29.947459 104653458us tsft -70dB signal -92dB noise antenna 0 2412 MHz 11g ht/20 39.0 Mb/s MCS 10
20 MHz lon GI mixed BCC FEC [bit 20] CF +QoS IP 10.189.6.135.5353 > 224.0.0.251.5353: 0*- [0q] 2/0/3
(Cache flush) PTR Bobs-iPhone.local., (Cache flush) PTR Bobs-iPhone.local. (217)

        0x0000:  aaaa 0300 0000 0800 4500 00f5 2053 0000  ........E....S..
        0x0010:  ff11 a865 0abd 0687 e000 00fb 14e9 14e9  ...e............
        0x0020:  00e1 5867 0000 8400 0000 0002 0000 0003  ..Xg............
        0x0030:  0137 0135 0144 0133 0139 0130 0138 0133  .7.5.D.3.9.0.8.3
        0x0040:  0135 0135 0139 0144 0144 0141 0143 0130  .5.5.9.D.D.A.C.0
        0x0050:  0130 0130 0130 0130 0130 0130 0130 0130  .0.0.0.0.0.0.0.0
        0x0060:  0130 0130 0130 0130 0130 0138 0145 0146  .0.0.0.0.0.8.E.F
        0x0070:  0369 7036 0461 7270 6100 000c 8001 0000  .ip6.arpa.......
        0x0080:  0078 0015 0d44 3139 8b64 432d 6950 686f  .x.....Bobs-iPho
        0x0090:  6e65 056c 6f63 616c 0003 3133 3501 3603  ne.local..135.6.
        0x00a0:  3138 3902 3130 0769 6e2d 6164 6472 c050  189.10.in-addr.P
        0x00b0:  000c 8001 0000 0078 0002 c060 c00c 002f  .......x...`.../
        0x00c0:  8001 0000 0078 0006 c00c 0002 0008 c075  .....x.........u
        0x00d0:  002f 8001 0000 0078 0006 c075 0002 0008  ./.....x...u....
        0x00e0:  0000 2905 a000 0011 9400 1200 0400 0e00  ..).............
        0x00f0:  256e 8dc1 7d01 b16c 8dc1 7d01 b1         %n..}..l..}..
```



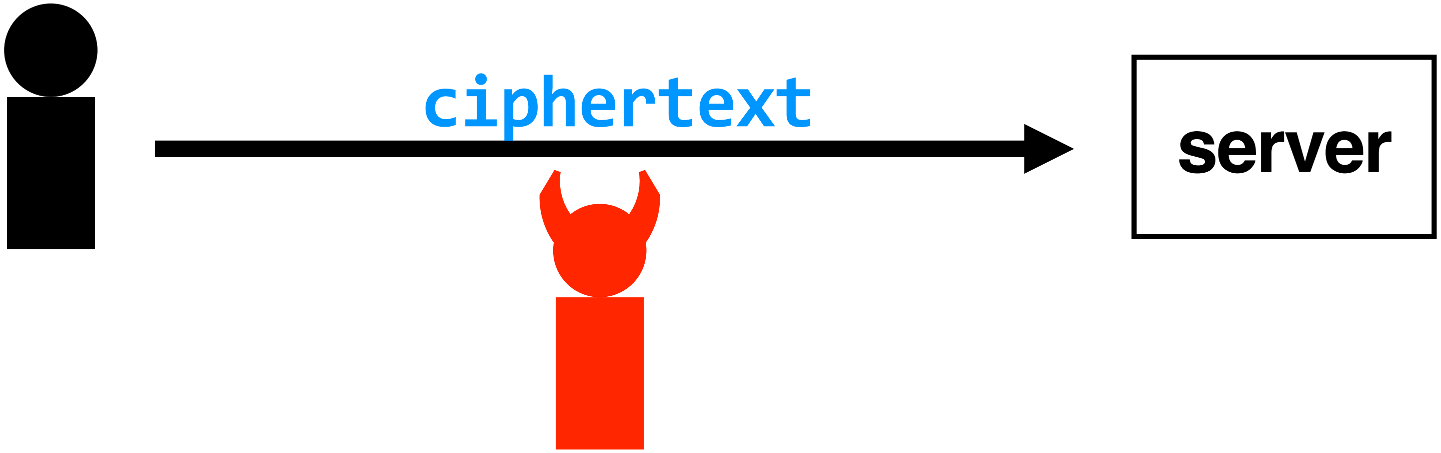
confidentiality: adversary cannot learn message contents

integrity: adversary cannot tamper with message contents
(if they do, client and/or server will detect it)

```
encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message
```

```
encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world
```

property: given the `ciphertext`, it is (virtually) impossible to obtain the `message` without knowing the `key`

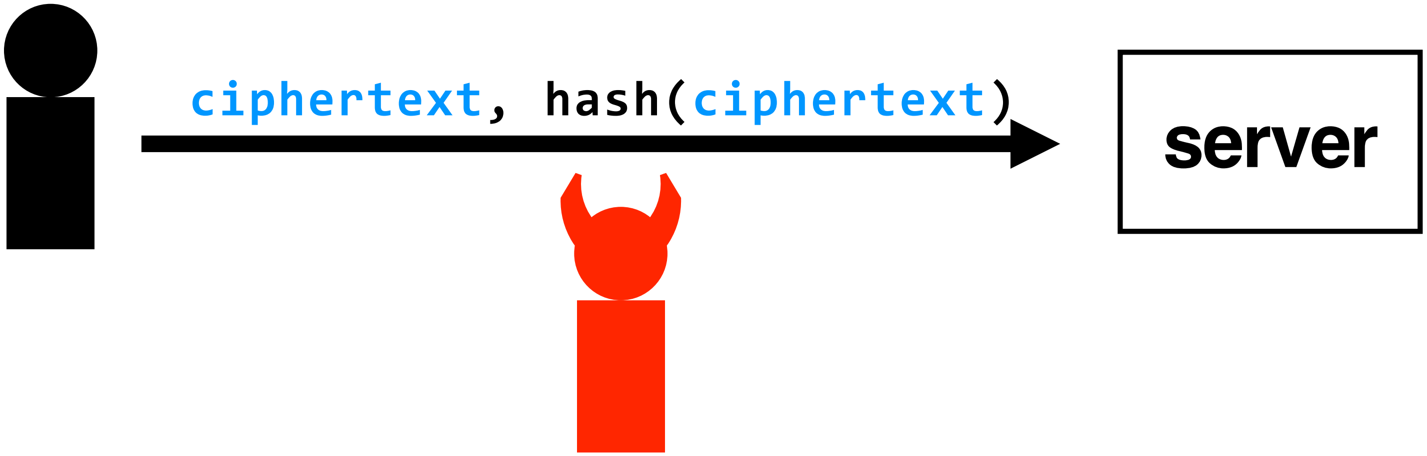


adversary can’t determine `message`, *but* might be able to cleverly alter `ciphertext` so that it decrypts to a different message

```
encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message
```

```
encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world
```

property: given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key



ciphertext, hash(ciphertext)

server

no good — if the adversary changes ciphertext, it can also (correctly) update the hash

```
encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message
```

```
encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world
```

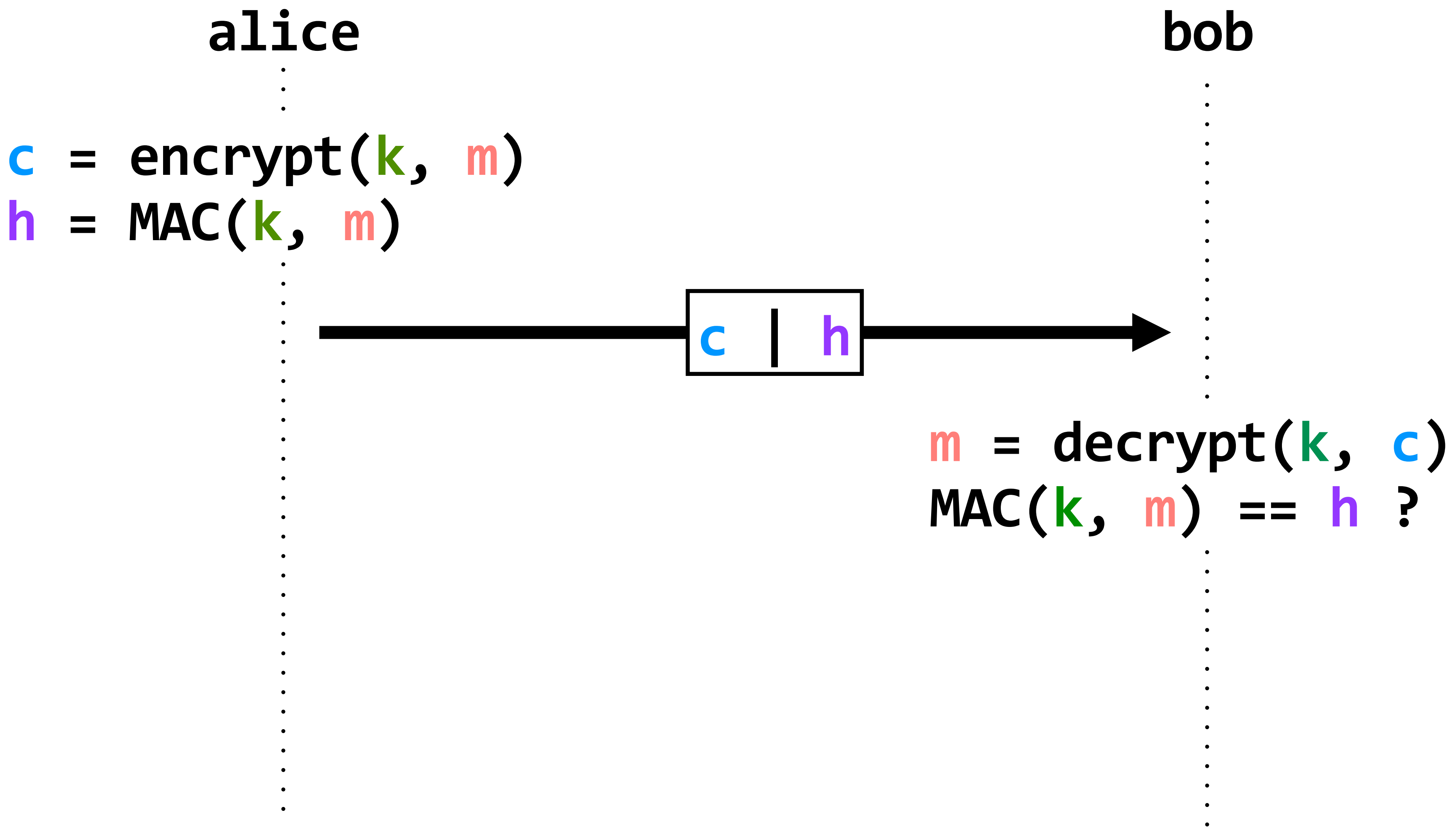
property: given the `ciphertext`, it is (virtually) impossible to obtain the `message` without knowing the `key`

```
MAC(key, message) → token
```

```
MAC(34fbcbd1, “hello, world”) = 0x59cccc95723737f777e62bc756c8da5c
```

property: given the `message`, it is (virtually) impossible to obtain the `token` without knowing the `key`

(it is also impossible to go in the reverse direction)



alice

c = encrypt(k, m)

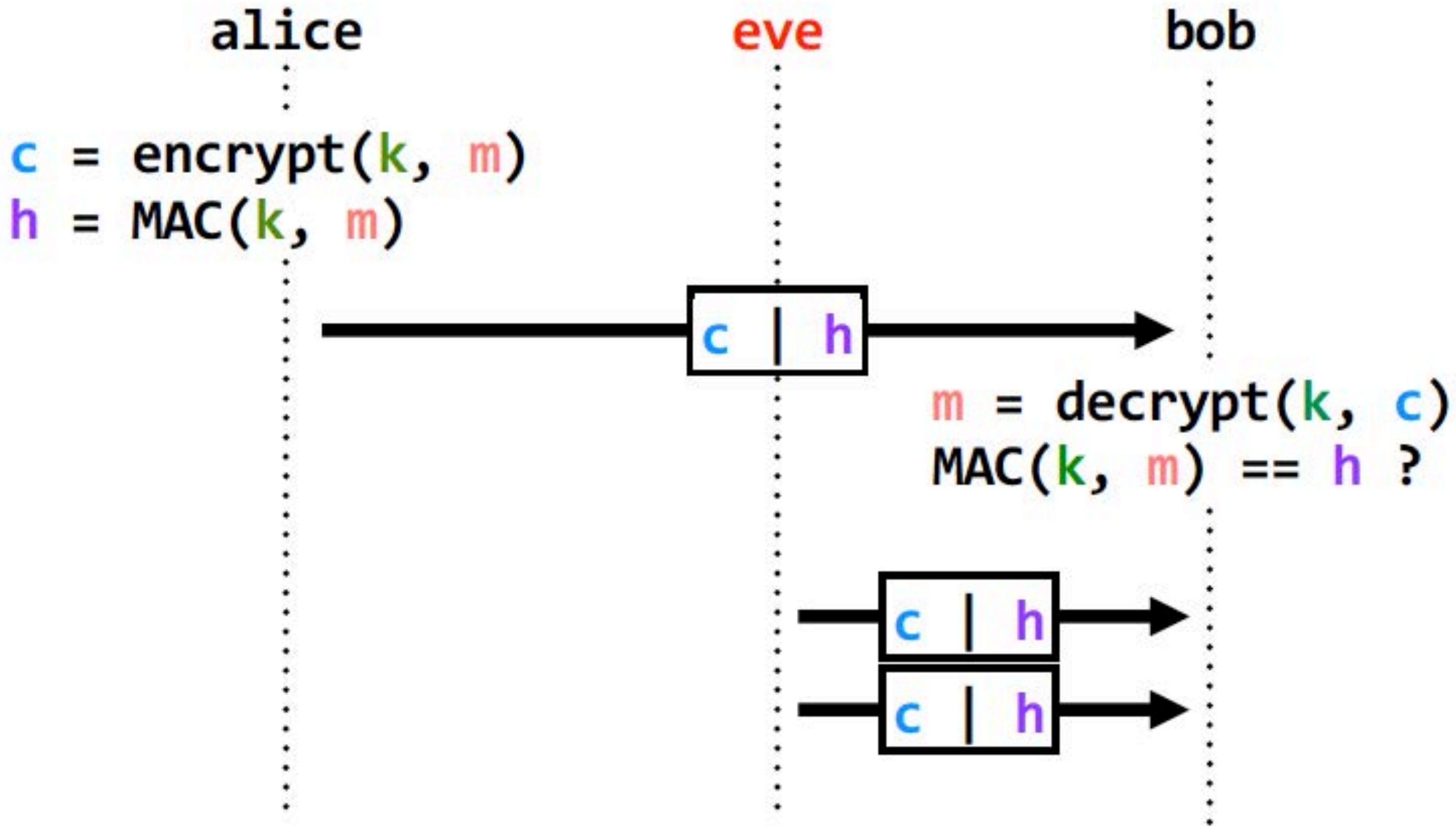
h = MAC(k, m)

c | h

bob

m = decrypt(k, c)

MAC(k, m) == h ?

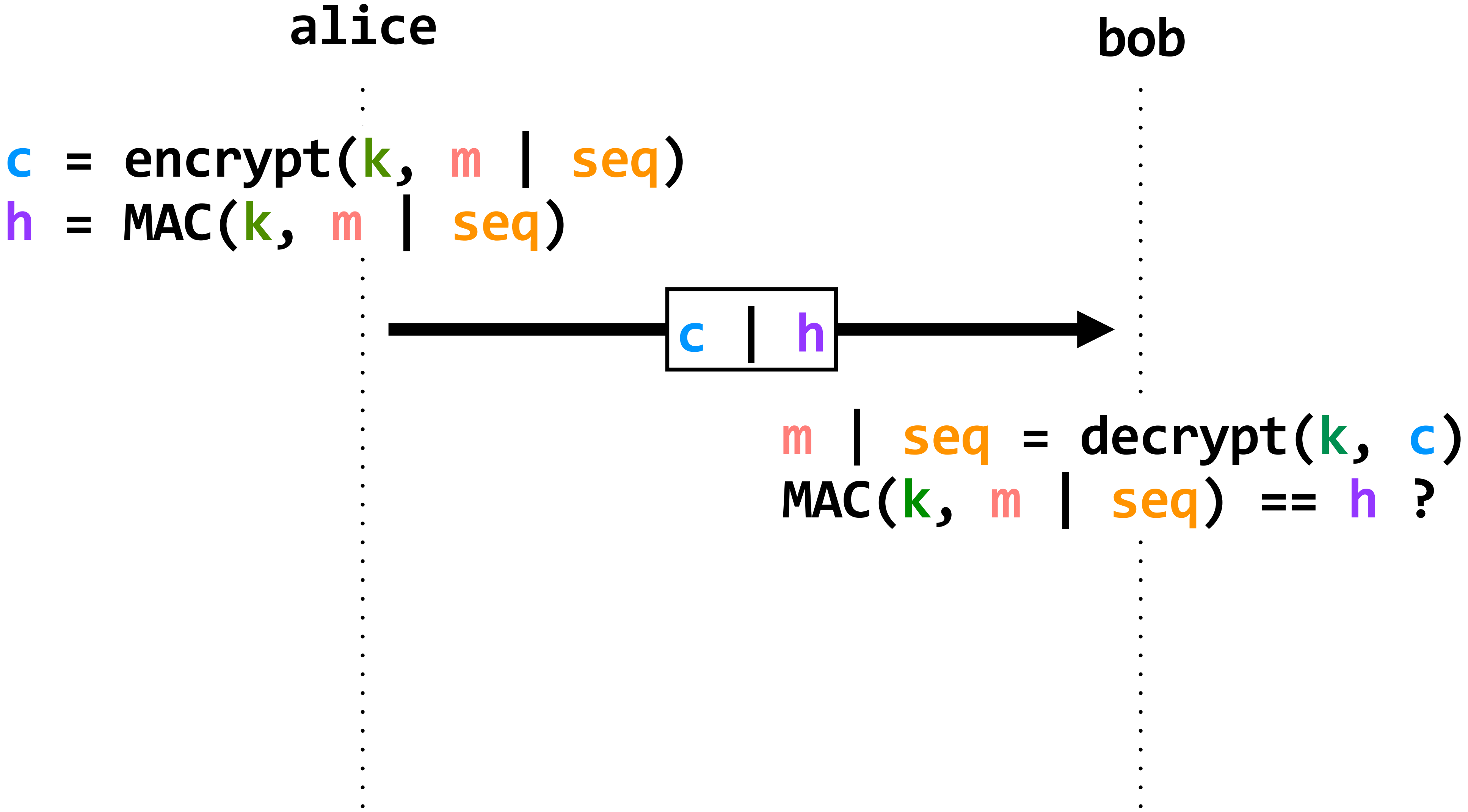


```
c = encrypt(k, m)
h = MAC(k, m)
```

```
m = decrypt(k, c)
MAC(k, m) == h ?
```

problem: replay attacks

(adversary could intercept a message, re-send it at a later time)

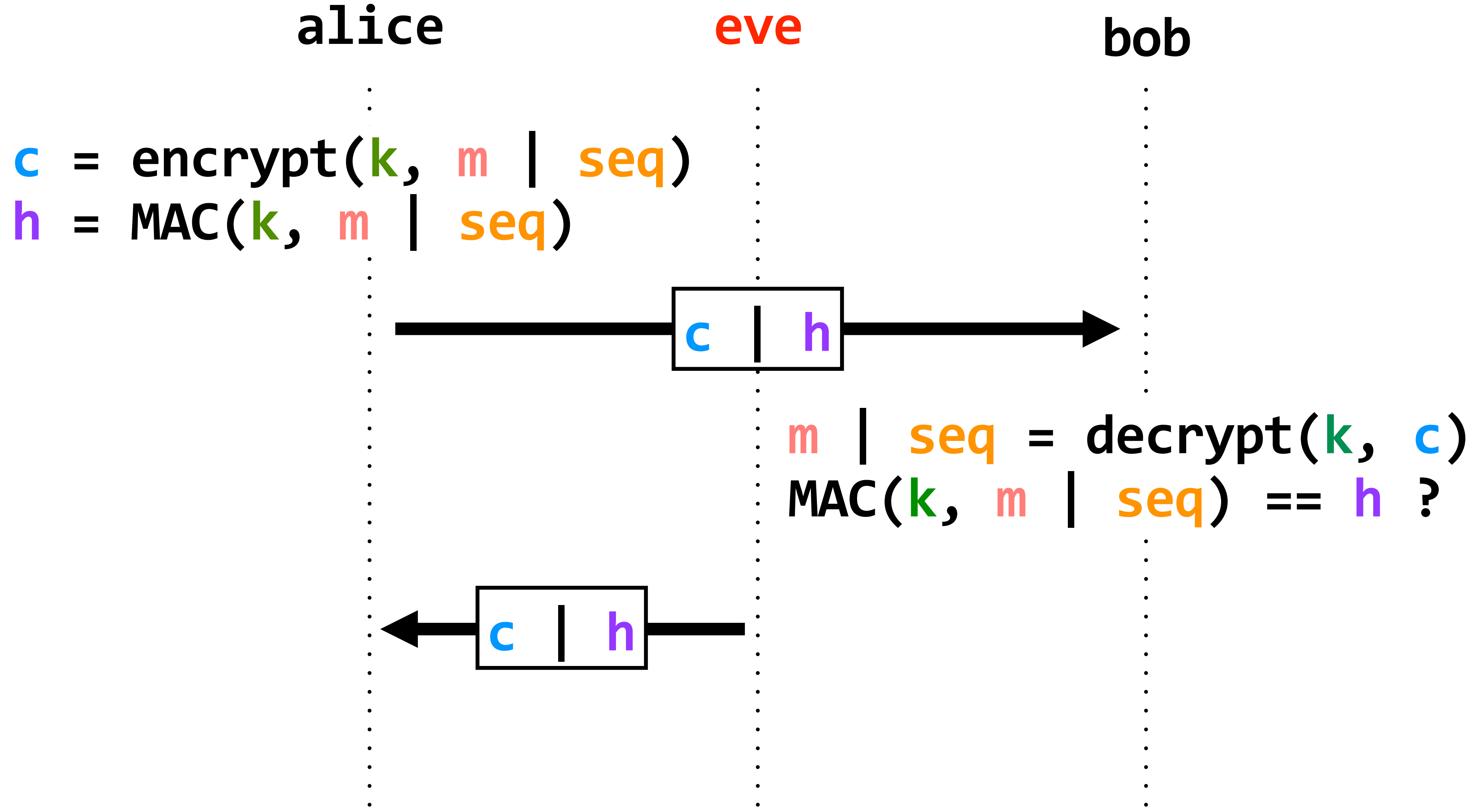


alice

bob

```
c = encrypt(k, m | seq)
h = MAC(k, m | seq)
```

```
m | seq = decrypt(k, c)
MAC(k, m | seq) == h ?
```



alice

eve

bob

c = encrypt(k, m | seq)

h = MAC(k, m | seq)

c | h

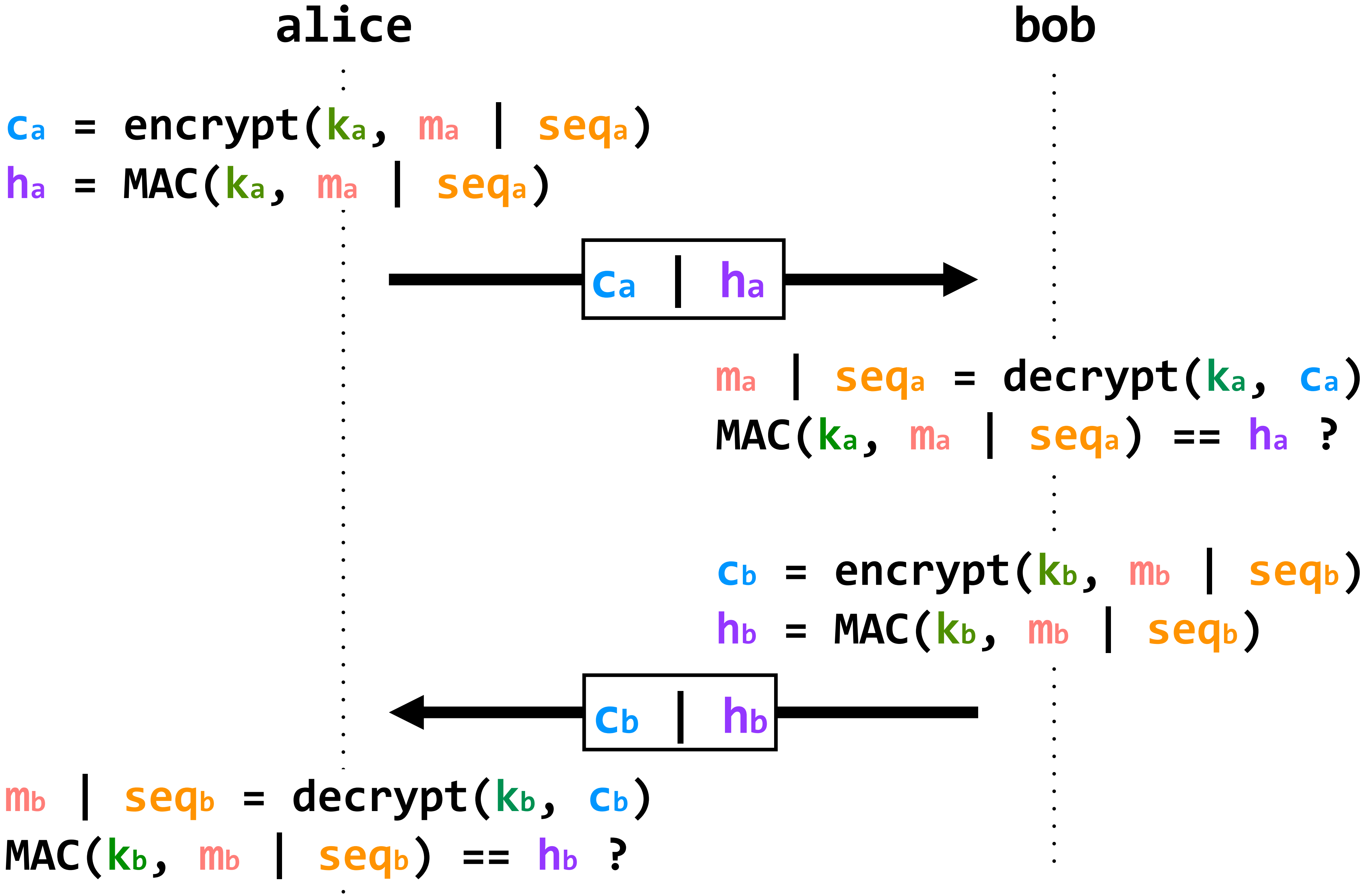
m | seq = decrypt(k, c)

MAC(k, m | seq) == h ?

c | h

problem: reflection attacks

(adversary could intercept a message, re-send it at a later time in the opposite direction)



alice

bob

$c_a = \text{encrypt}(k_a, m_a \mid seq_a)$

$h_a = \text{MAC}(k_a, m_a \mid seq_a)$

$c_a \mid h_a$

$m_a \mid seq_a = \text{decrypt}(k_a, c_a)$

$\text{MAC}(k_a, m_a \mid seq_a) == h_a$?

$c_b = \text{encrypt}(k_b, m_b \mid seq_b)$

$h_b = \text{MAC}(k_b, m_b \mid seq_b)$

$c_b \mid h_b$

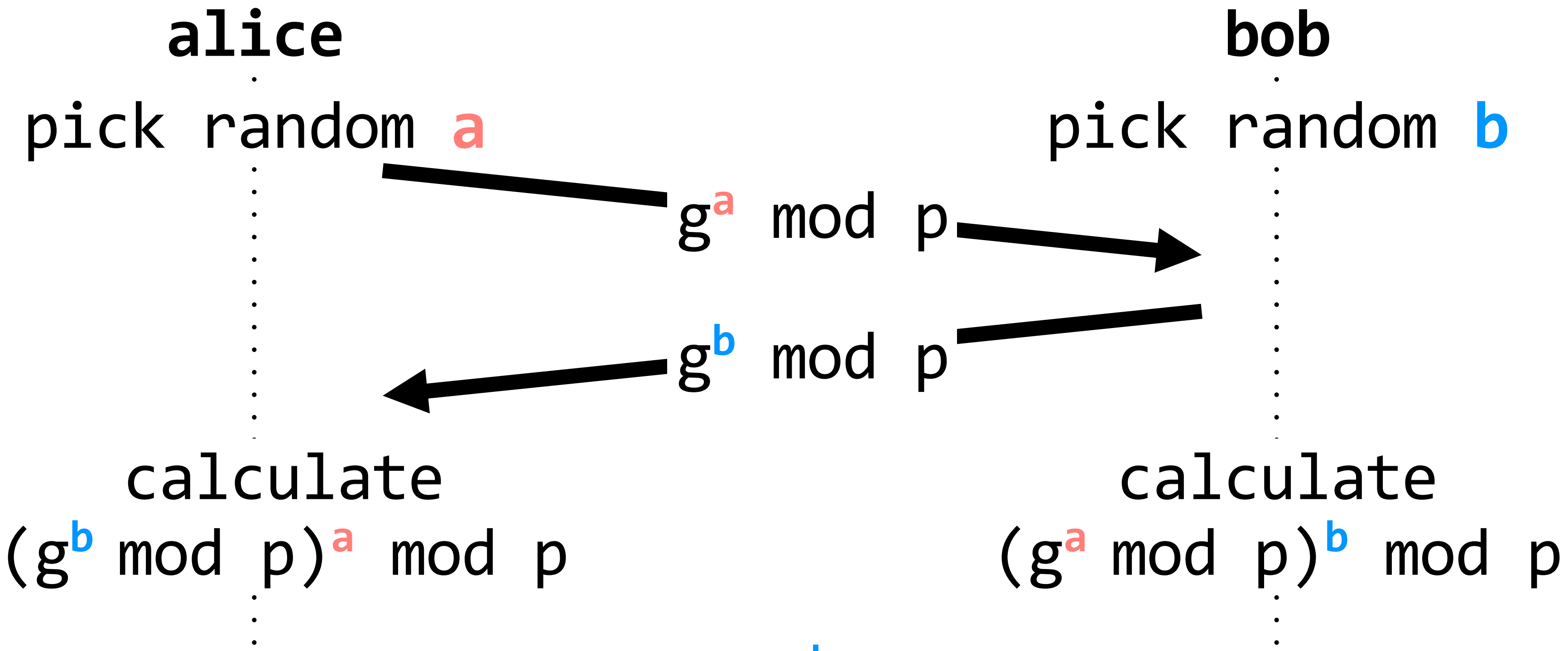
$m_b \mid seq_b = \text{decrypt}(k_b, c_b)$

$\text{MAC}(k_b, m_b \mid seq_b) == h_b$?

problem: how do the parties know the keys?

known: `p (prime), g`

property: given $\mathbf{g^r \bmod p}$, it is (virtually) impossible to determine **r** *even if* you know **g** and **p**



alice

pick random a

$g^a \bmod p$

calculate

$(g^b \bmod p)^a \bmod p$

bob

pick random b

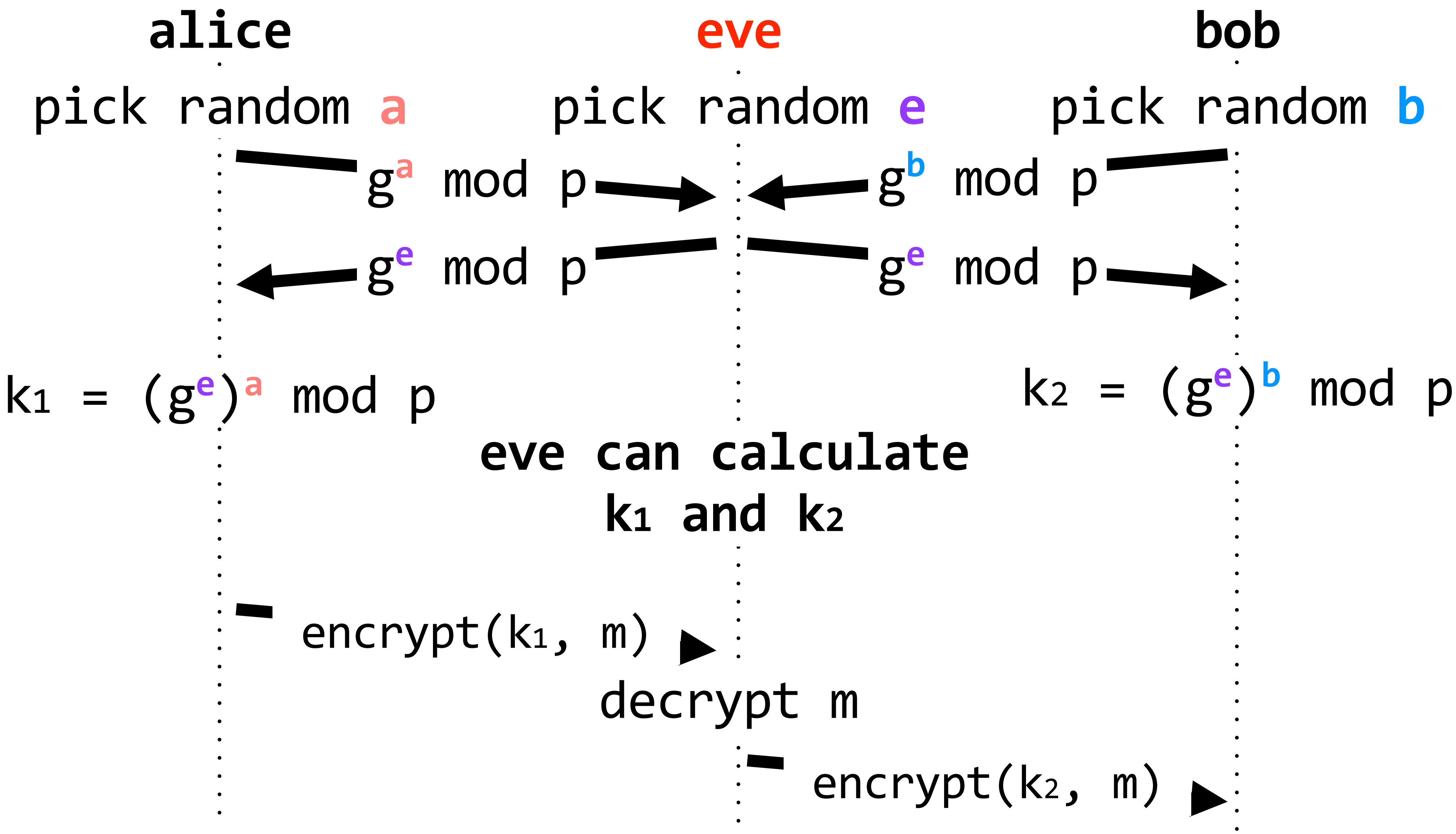
$g^b \bmod p$

calculate

$(g^a \bmod p)^b \bmod p$

$$\mathbf{key} = g^{ab} \bmod p$$

$(g^x \bmod p)^y \bmod p = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p$



alice **eve** **bob**

pick random a pick random e pick random b

$g^a \bmod p$ $g^b \bmod p$

$g^e \bmod p$ $g^e \bmod p$

$k_1 = (g^e)^a \bmod p$ $k_2 = (g^e)^b \bmod p$

eve can calculate k_1 and k_2

encrypt(k_1, m)

decrypt m

encrypt(k_2, m)

problem: alice and bob don't know they're not communicating directly

cryptographic signatures

allow users to verify identities using public-key cryptography

users generate key pairs

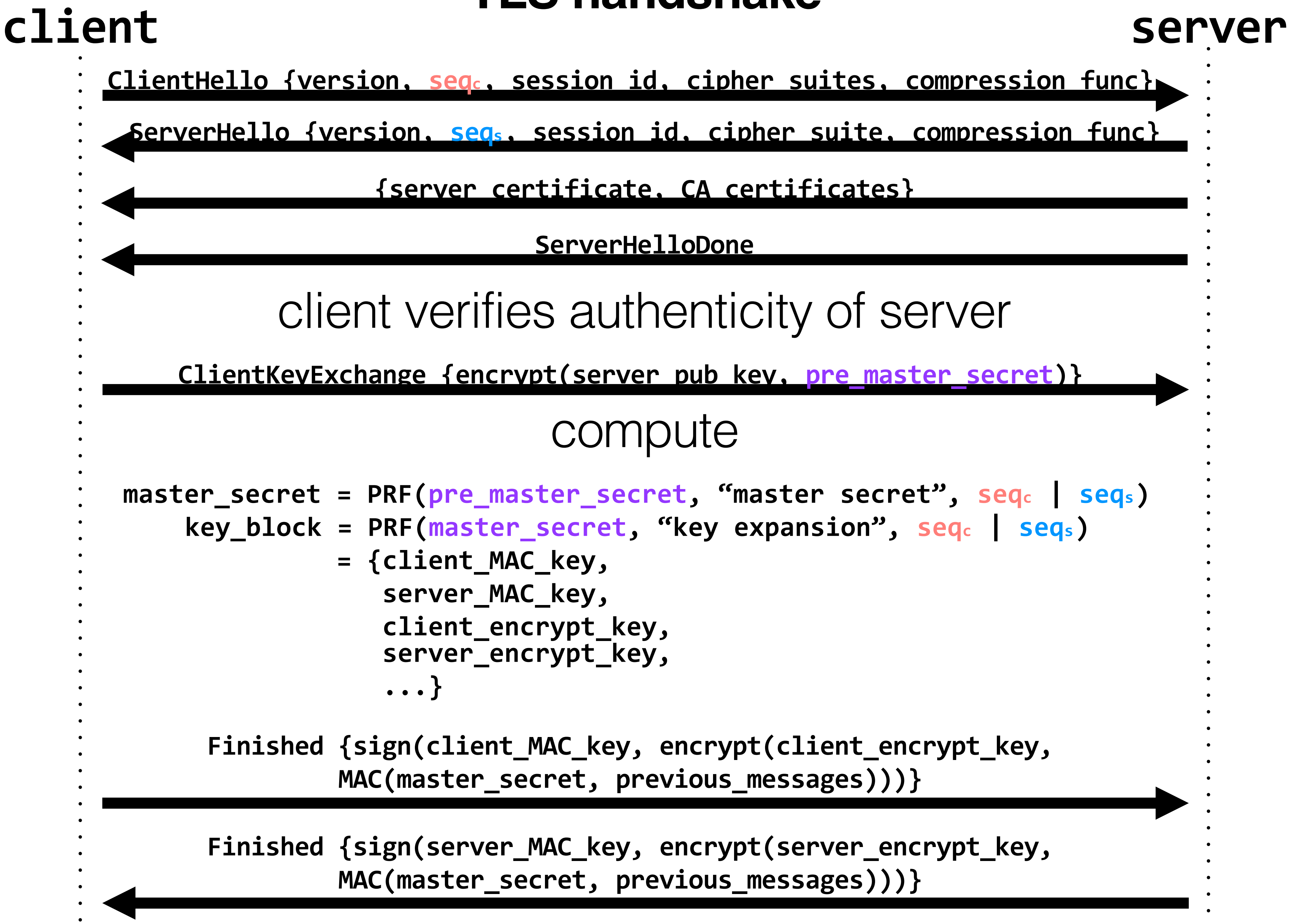
the two keys in the pair are related mathematically

```
{public_key, secret_key}
```

```
sign(secret_key, message) → sig
verify(public_key, message, sig) → yes/no
```

property: it is (virtually) impossible to compute `sig` without `secret_key`

TLS handshake



client

server

ClientHello {version, seq_c, session id, cipher suites, compression func}

ServerHello {version, seq_s, session id, cipher suite, compression func}

{server certificate, CA certificates}

ServerHelloDone

client verifies authenticity of server

ClientKeyExchange {encrypt(server pub key, pre_master_secret)}

compute

master_secret = PRF(pre_master_secret, “master secret”, seq_c | seq_s)

key_block = PRF(master_secret, “key expansion”, seq_c | seq_s)

= {client_MAC_key,
server_MAC_key,
client_encrypt_key,
server_encrypt_key,
...}

Finished {sign(client_MAC_key, encrypt(client_encrypt_key, MAC(master_secret, previous_messages)))}

Finished {sign(server_MAC_key, encrypt(server_encrypt_key, MAC(master_secret, previous_messages)))}

- **Secure channels** protect us from adversaries that can observer and tamper with packets in the network.

- Encrypting with **symmetric keys** provides secrecy, and using **MACs** provides integrity. **Diffie-Hellman key exchange** lets us exchange the symmetric key securely.

- To verify identities, we use **public-key cryptography** and cryptographic **signatures**. We often distribute public keys with **certificate authorities**, though this method is not perfect.