Asynchronous intrusion recovery for interconnected web services

Ramesh Chandra, **Taesoo Kim**, Nickolai Zeldovich

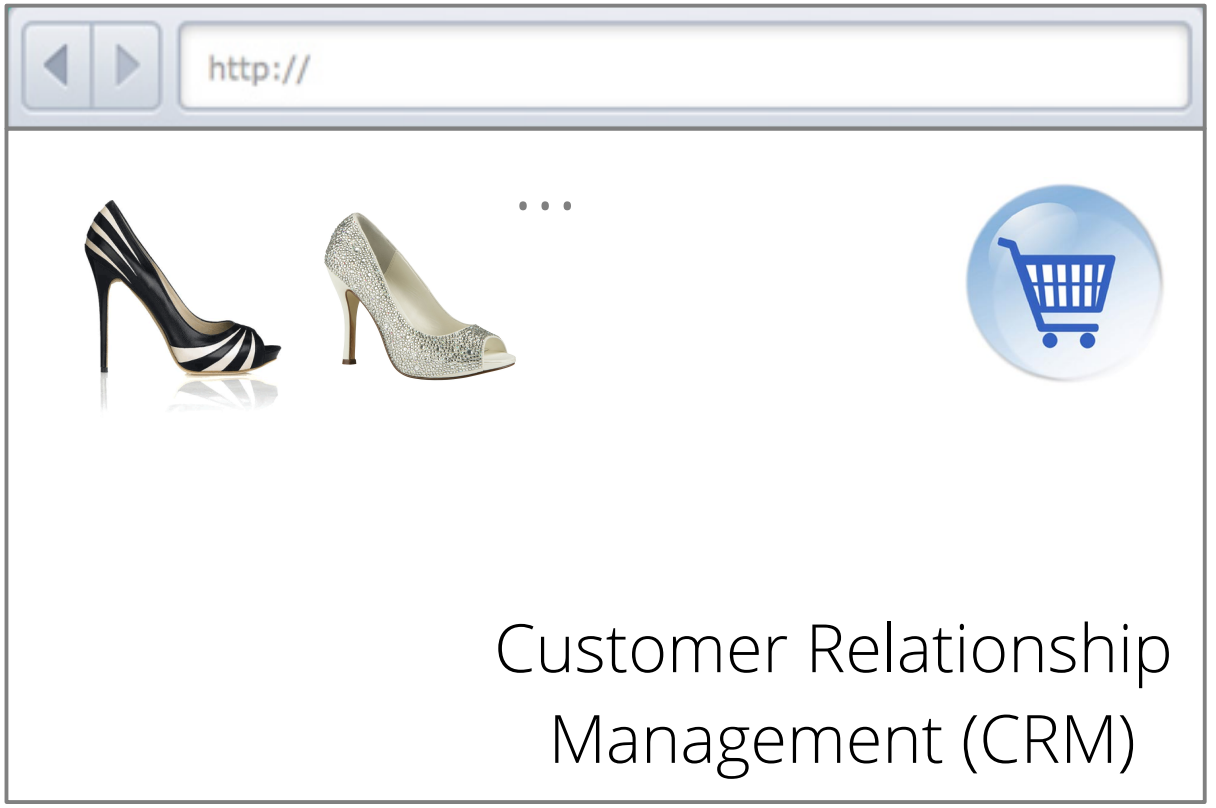
MIT CSAIL

Today's web services are highly interconnected

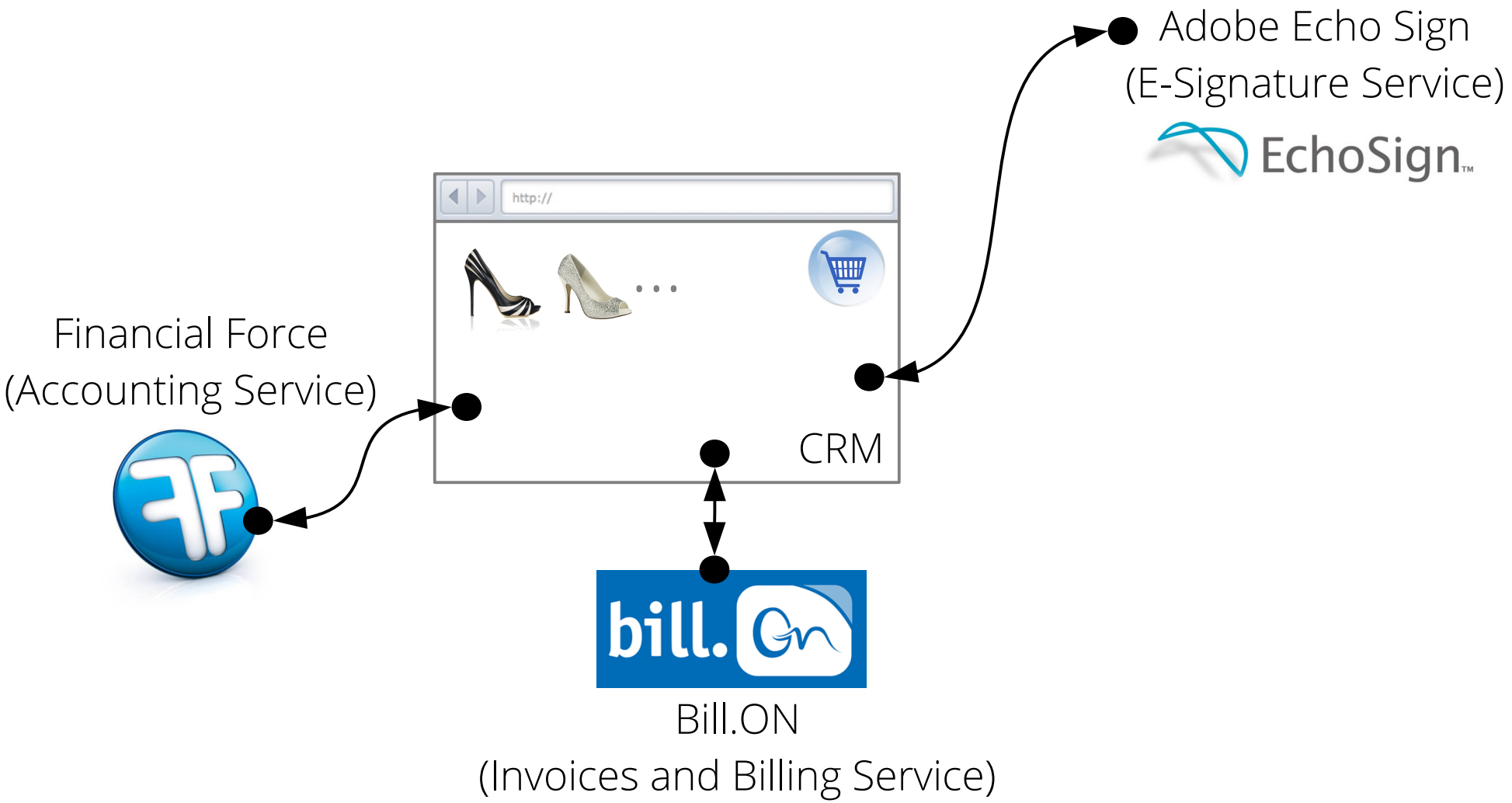
- Many web services provide APIs to other sites

- Many websites integrate those APIs:
  - Authentication: Facebook Connect, Google+ ...
  - Data sharing: Dropbox ...
  - Business process management: Salesforce ...
  - ...

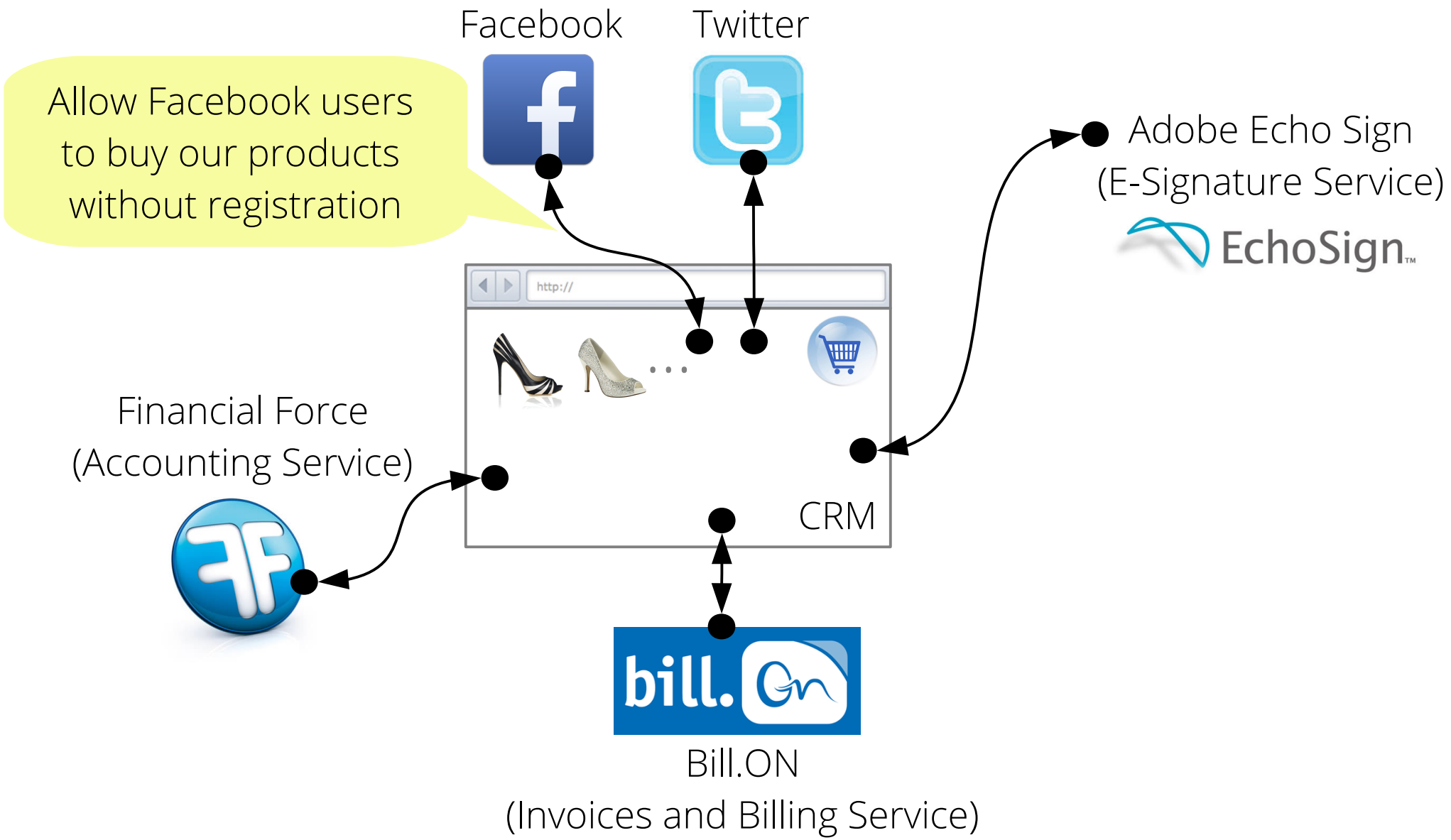
Example: online shopping mall



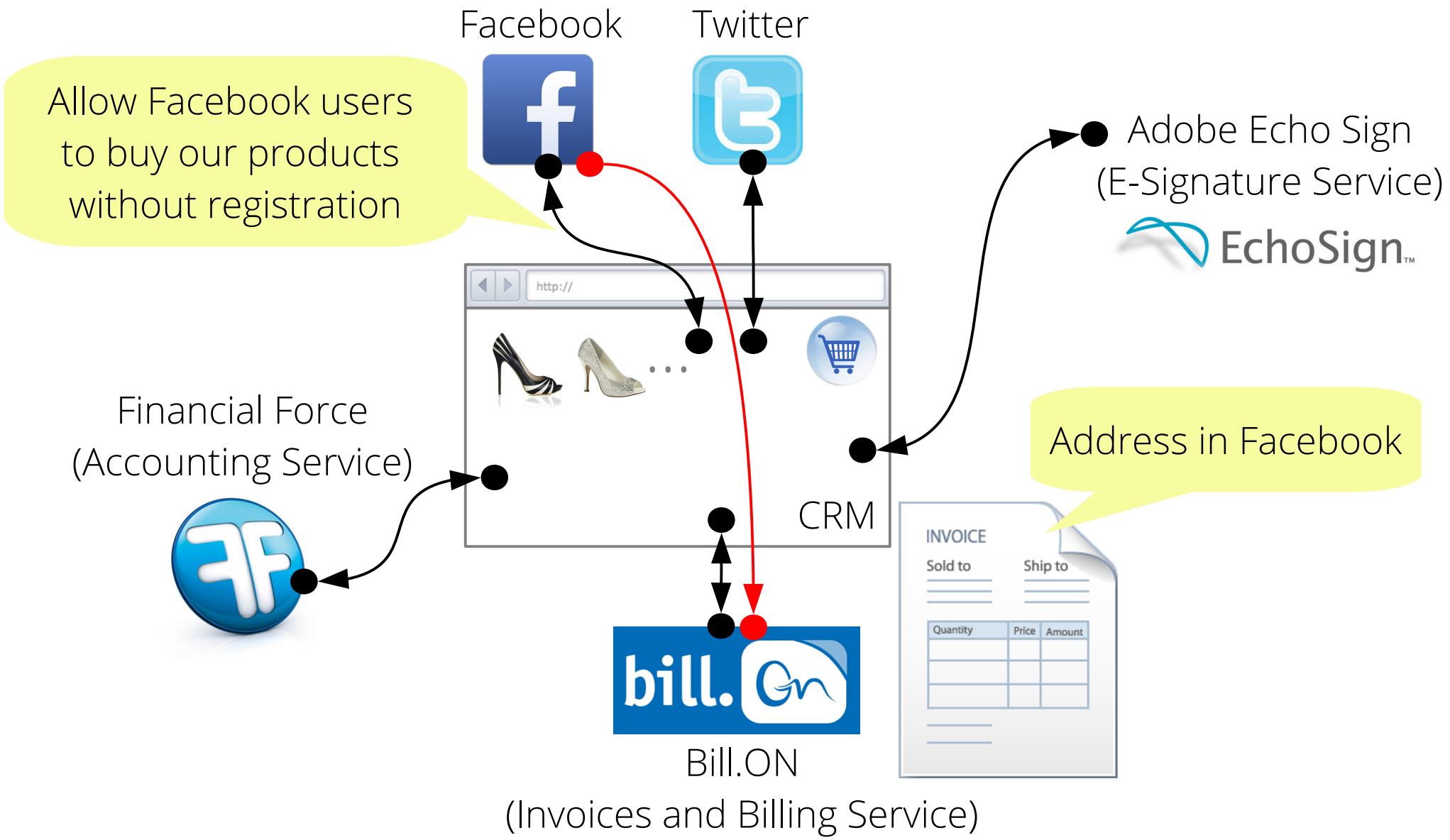
Example: online shopping mall



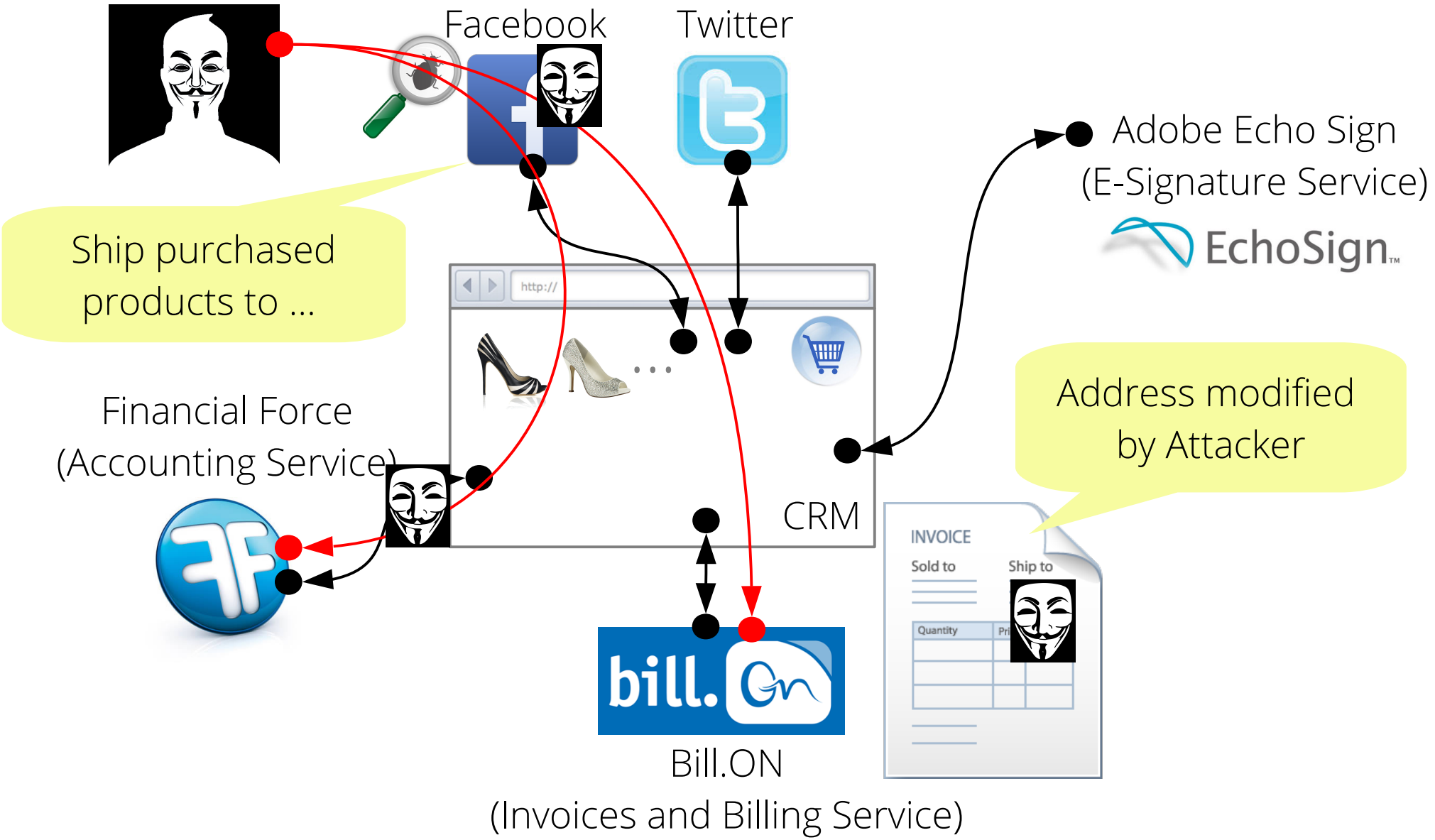
Example: online shopping mall



Example: online shopping mall



Attack in one service can spread between services



Bugs in web services are commonplace

- Facebook (Mar 29th 2013):
  - Attackers can intercept full permission access tokens

BREAK SECURITY

Bugs in web services are commonplace

- Facebook (Mar 29th 2013):
  - Attackers can intercept full permission access tokens
- Many web services have similar bugs
  - Twitter (Aug 20th 2013)
  - Instagram (May 2nd 2013)
  - Microsoft Yammer (Aug 4th 2013)


SECURITY

Bloke leaks '1000s' of Twitter login tokens, says he can hack ANY twit

Known vulnerability sat on by Tw

By John Leyden, 20th August 2013

12

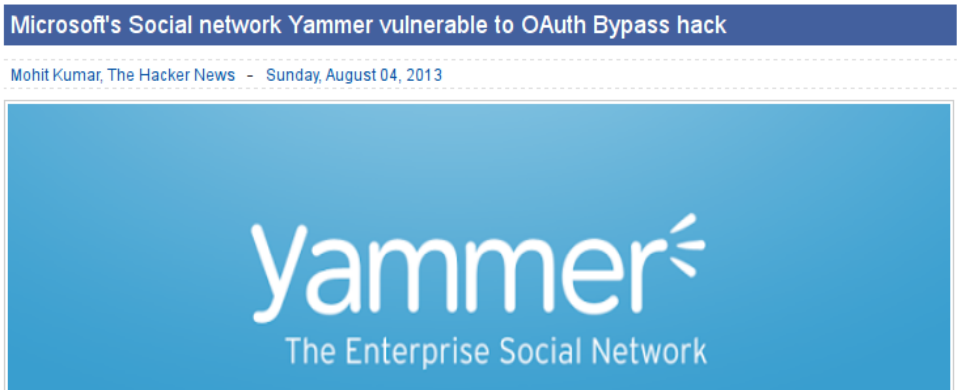
A hacker calling himself the "M
Twitter user account on the pla
Turkish tweeters.

Hacking Instagram Accounts using OAuth vulnerability

Mohit Kumar, The Hacker News - Thursday, May 02, 2013

'*Nir Goldshlager*' known as Facebook hacker and founder of Break Security , who reported many critical bugs in Facebook OAuth mechanism in past few months, today disclose a critical vulnerability in Instagram Oauth that allow an attacker to hack any account.

Succesful hack allows attacker to access private photos, ability to delete victim's photos and to edit comments and also the ability to post new photos.


Microsoft's Social network Yammer vulnerable to OAuth Bypass hack

Mohit Kumar, The Hacker News - Sunday, August 04, 2013

yammer

The Enterprise Social Network

Goal

- **Recovering integrity** in interconnected services
  - Repair the state of affected services as if the attack never occurred

- State-of-the-art: manual recovery
  - Admin doesn't trust other sites for recovery
  - Require manual interaction (e.g., email other admin)

General plan for automatic recovery

- Use **rollback-and-replay** for recovering integrity in single machine
  - Prior works: Retro [OSDI '10], Warp [SOSP '11]

- **Extend** rollback-and-replay to **many web services**!

Challenges

- Rollback-and-replay requires **global coordinator**
  - Each service cannot decide what to do for repair
- All services must be **available** during recovery
  - We want to repair some services even if others are down
  - Consistency problem: some services are not repaired yet

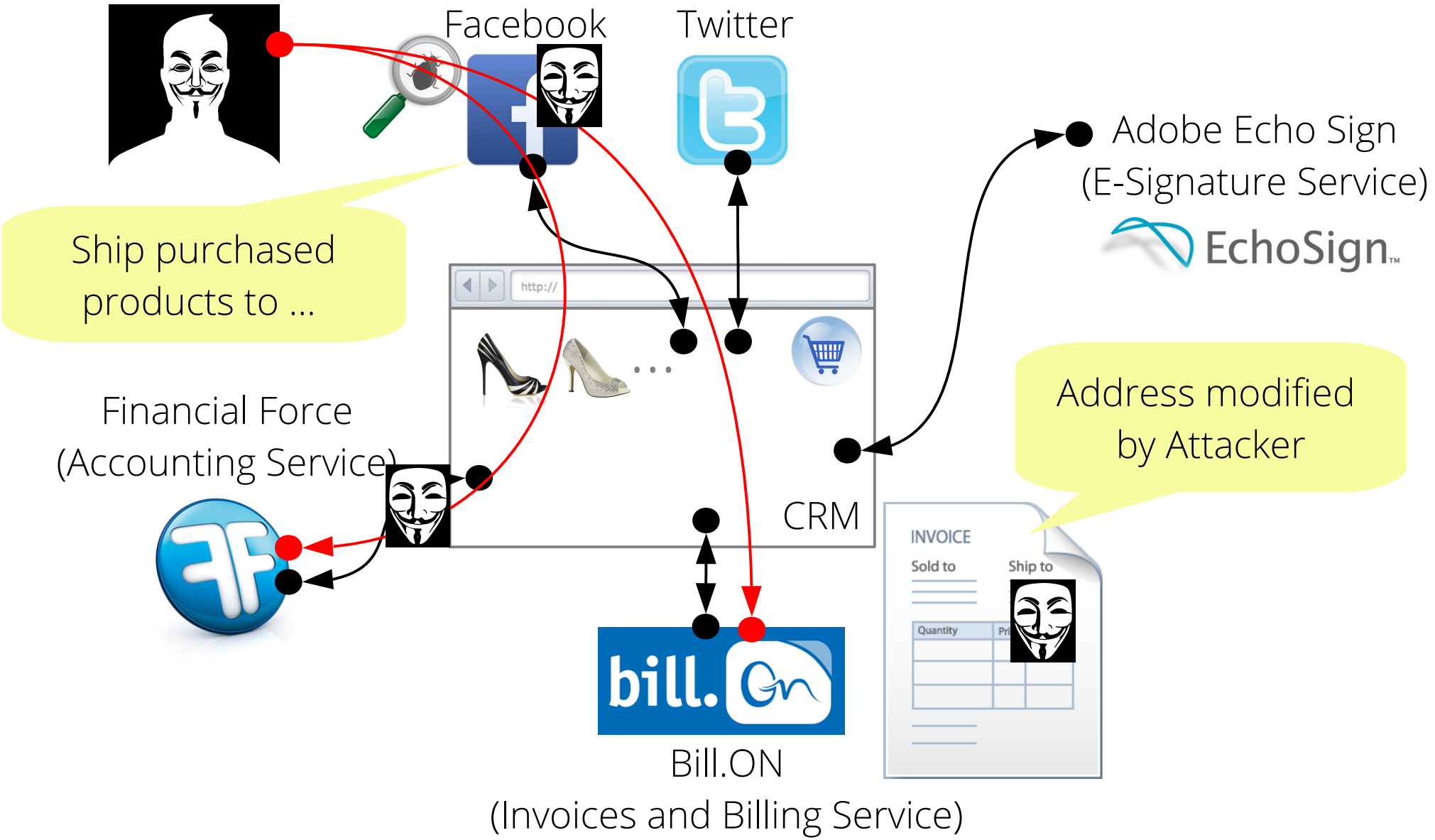
Contributions

Enable **automatic** intrusion recovery in **distributed** web services

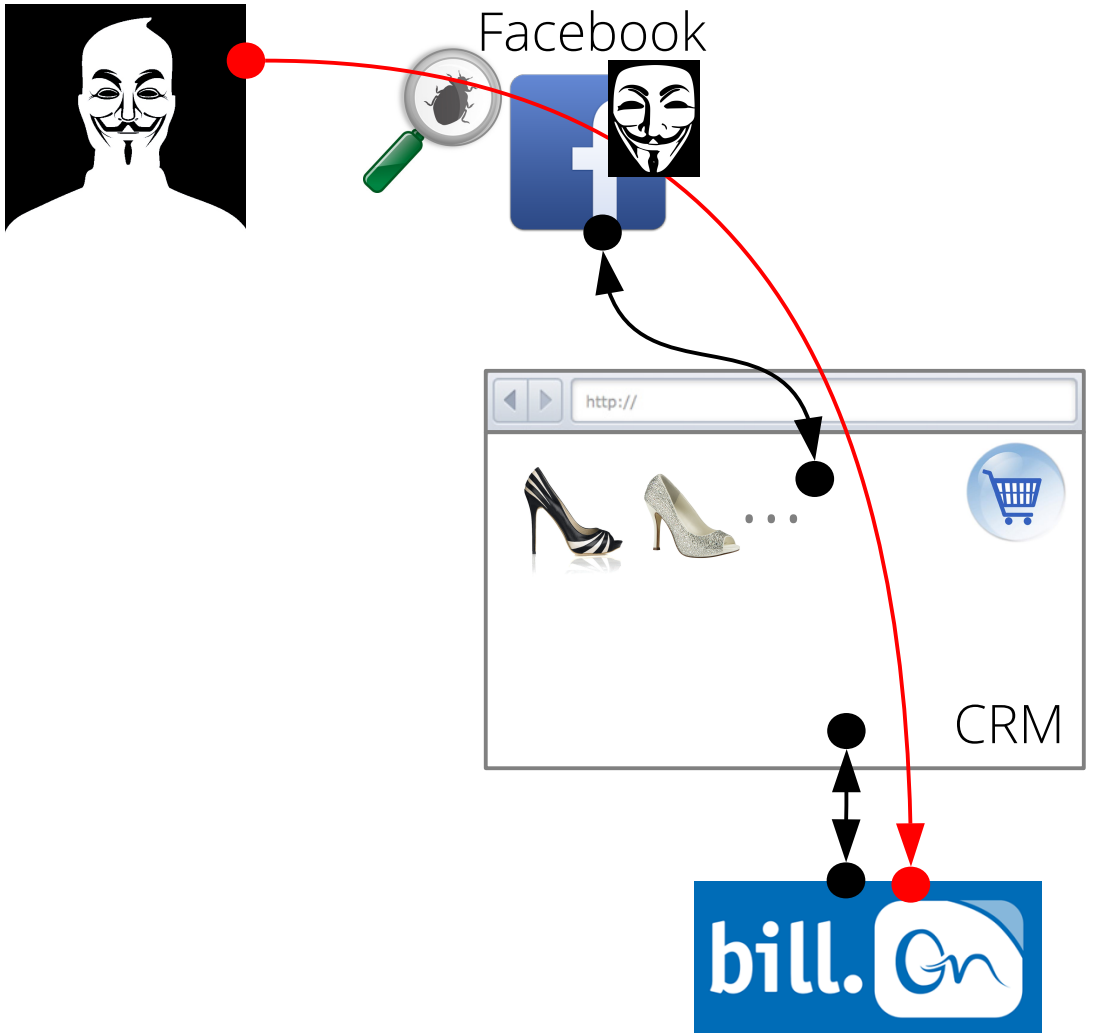
1. **Repair protocol** between services
   - No central coordinator
   - Each service controls its repair

2. **Asynchronous repair**
   - Proceed repair even with unavailable services
   - Consistency in partially repair state

Running example of an attack



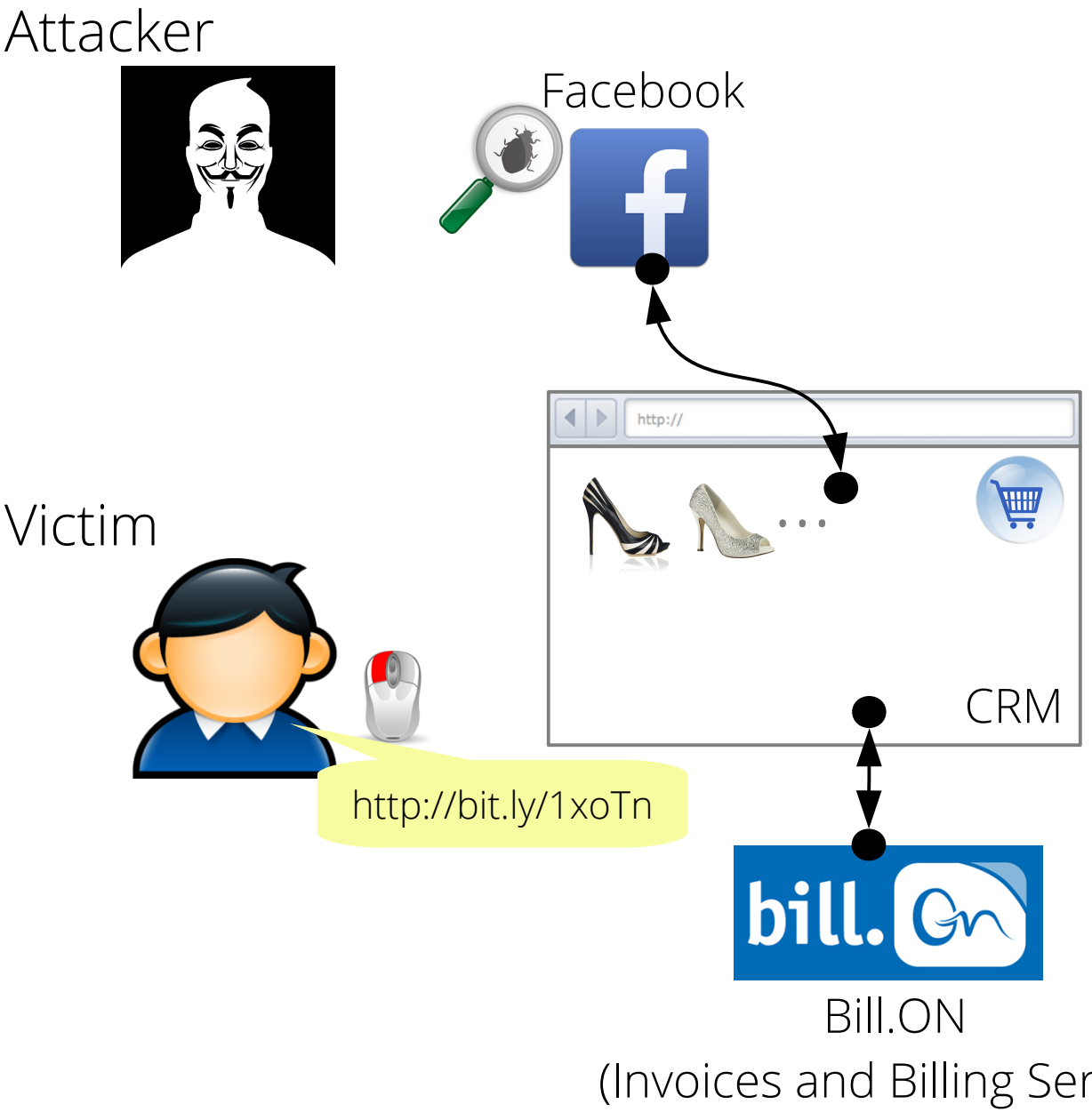
Running example of an attack



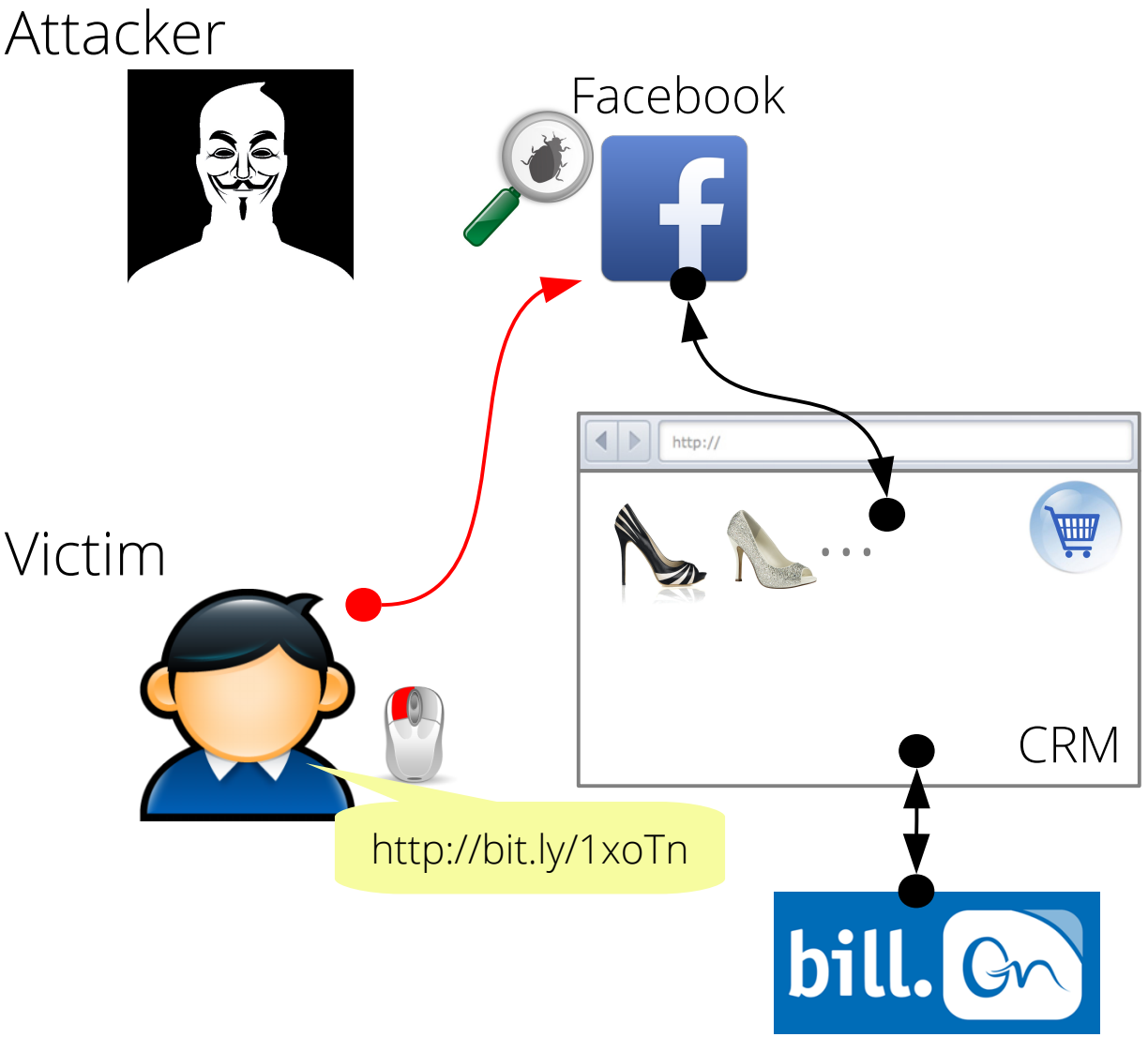
Bill.ON

(Invoices and Billing Service)

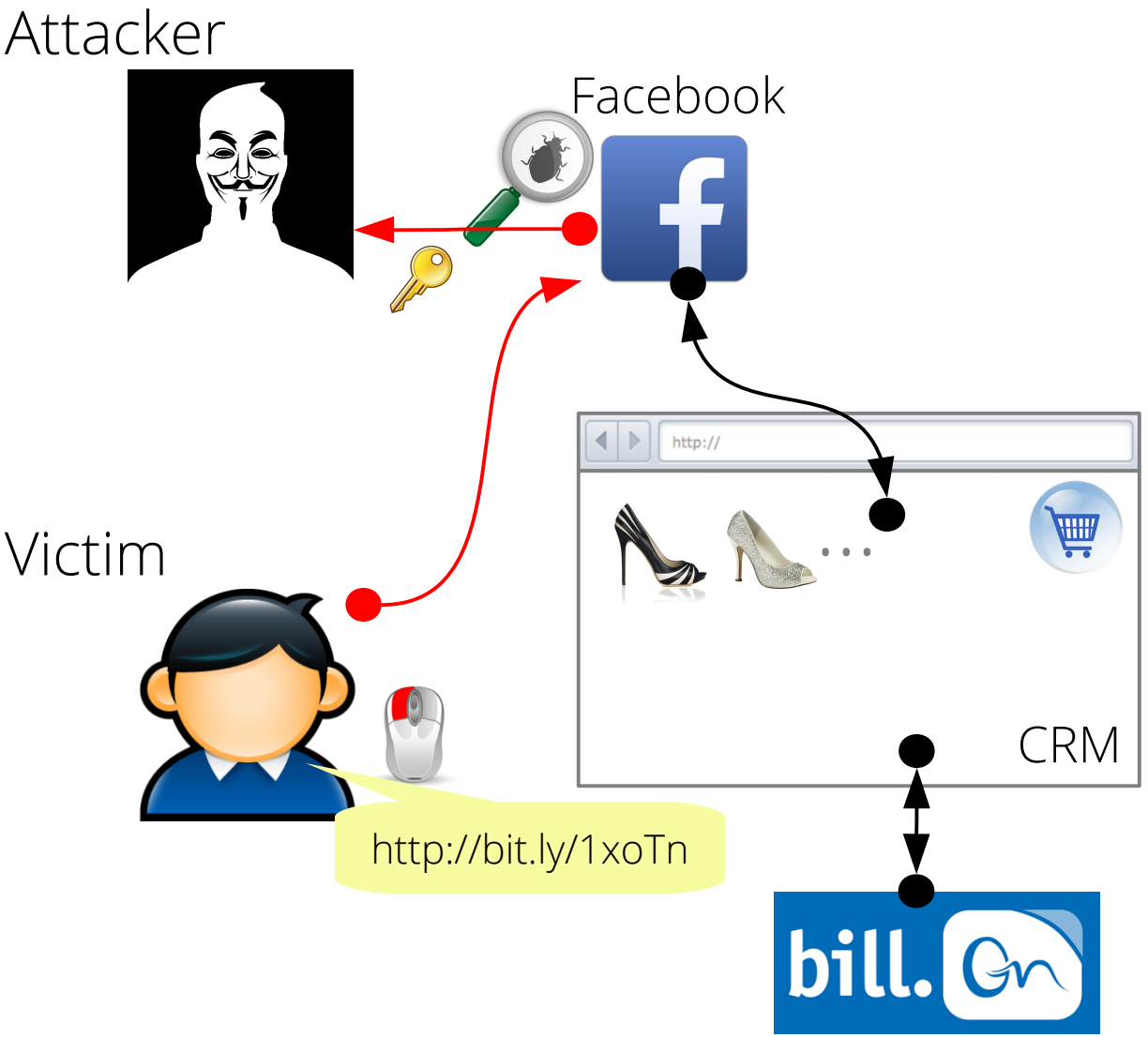
Running example of an attack



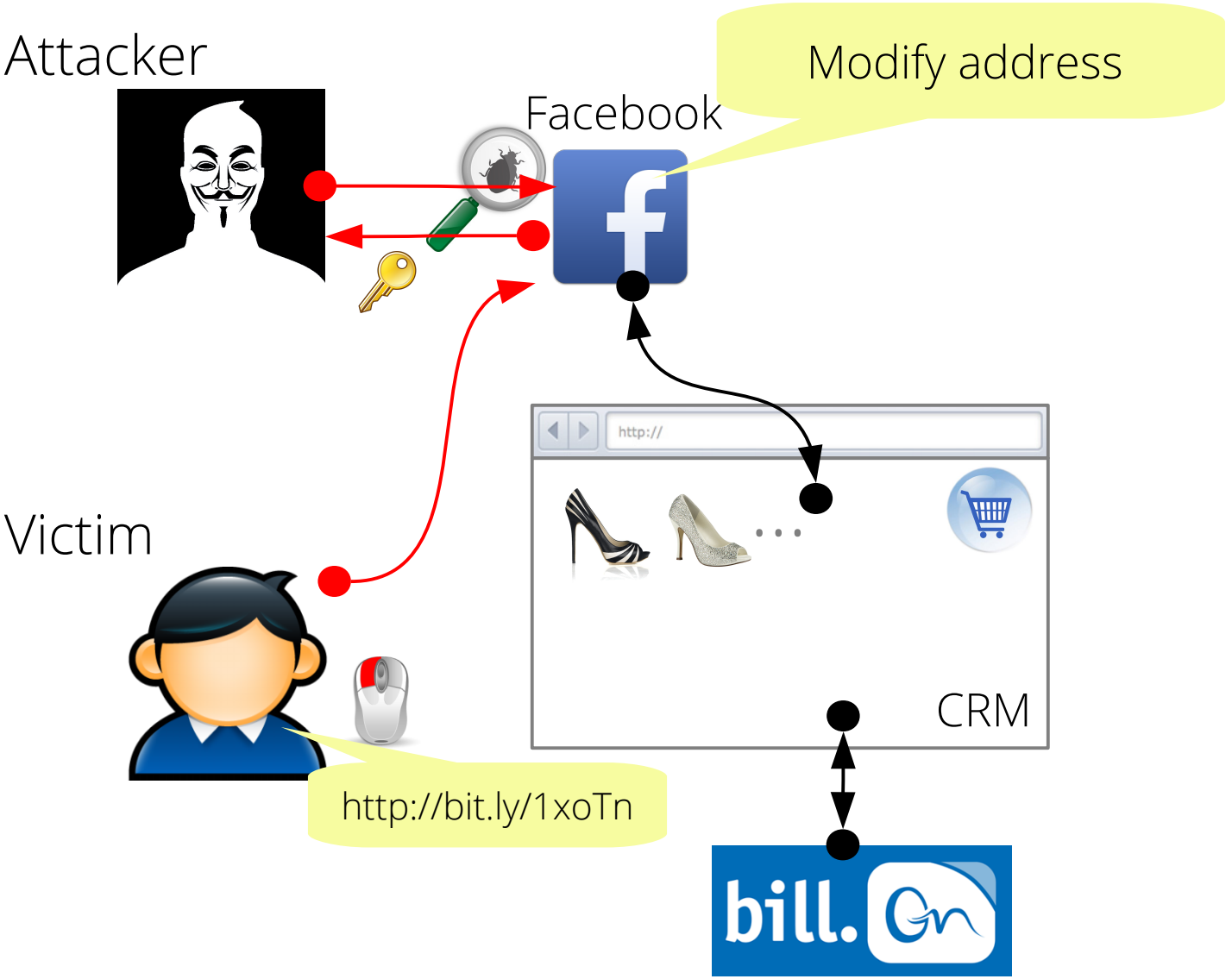
Running example of an attack



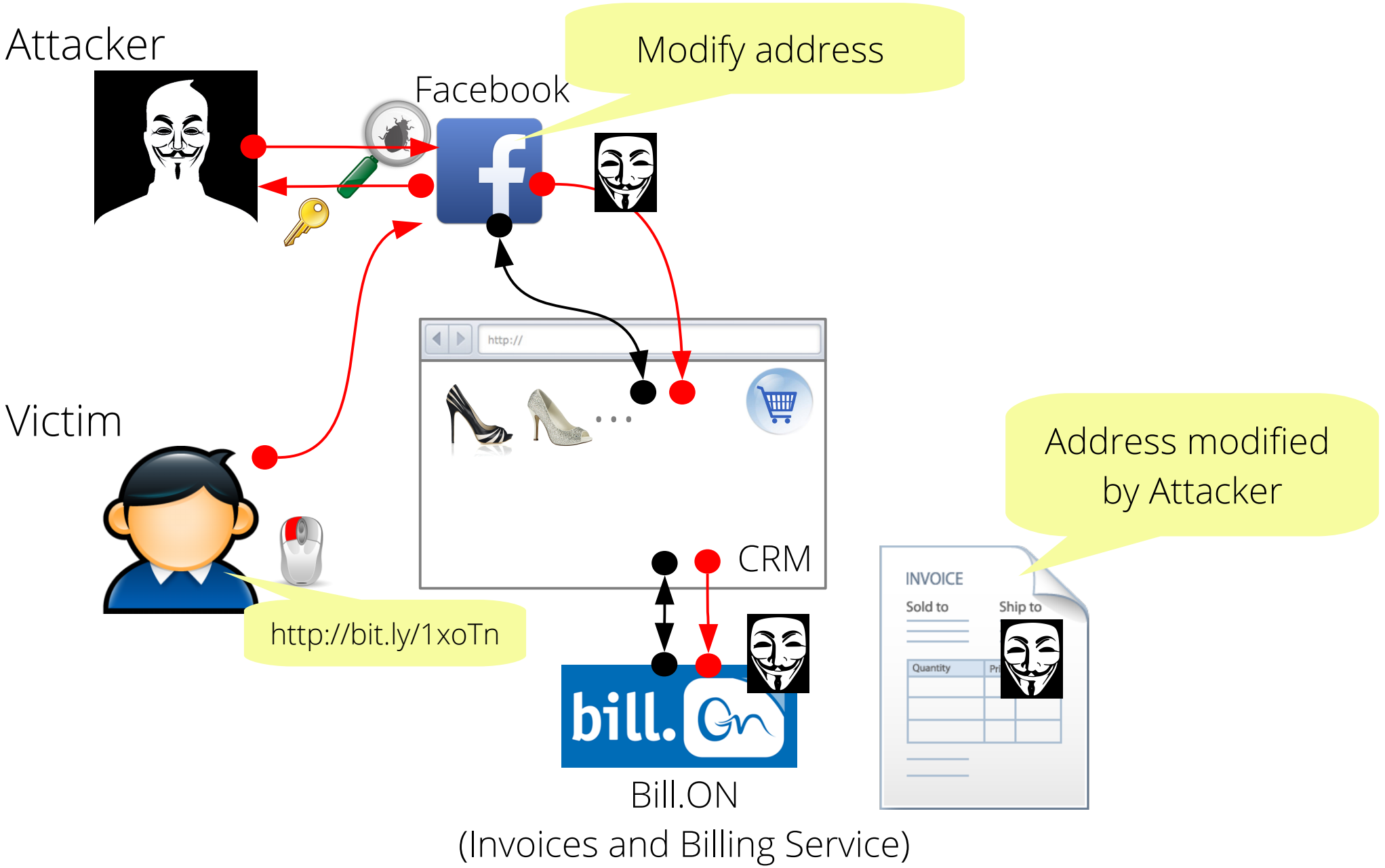
Running example of an attack



Running example of an attack



Running example of an attack



Timeline of the attack

Attacker

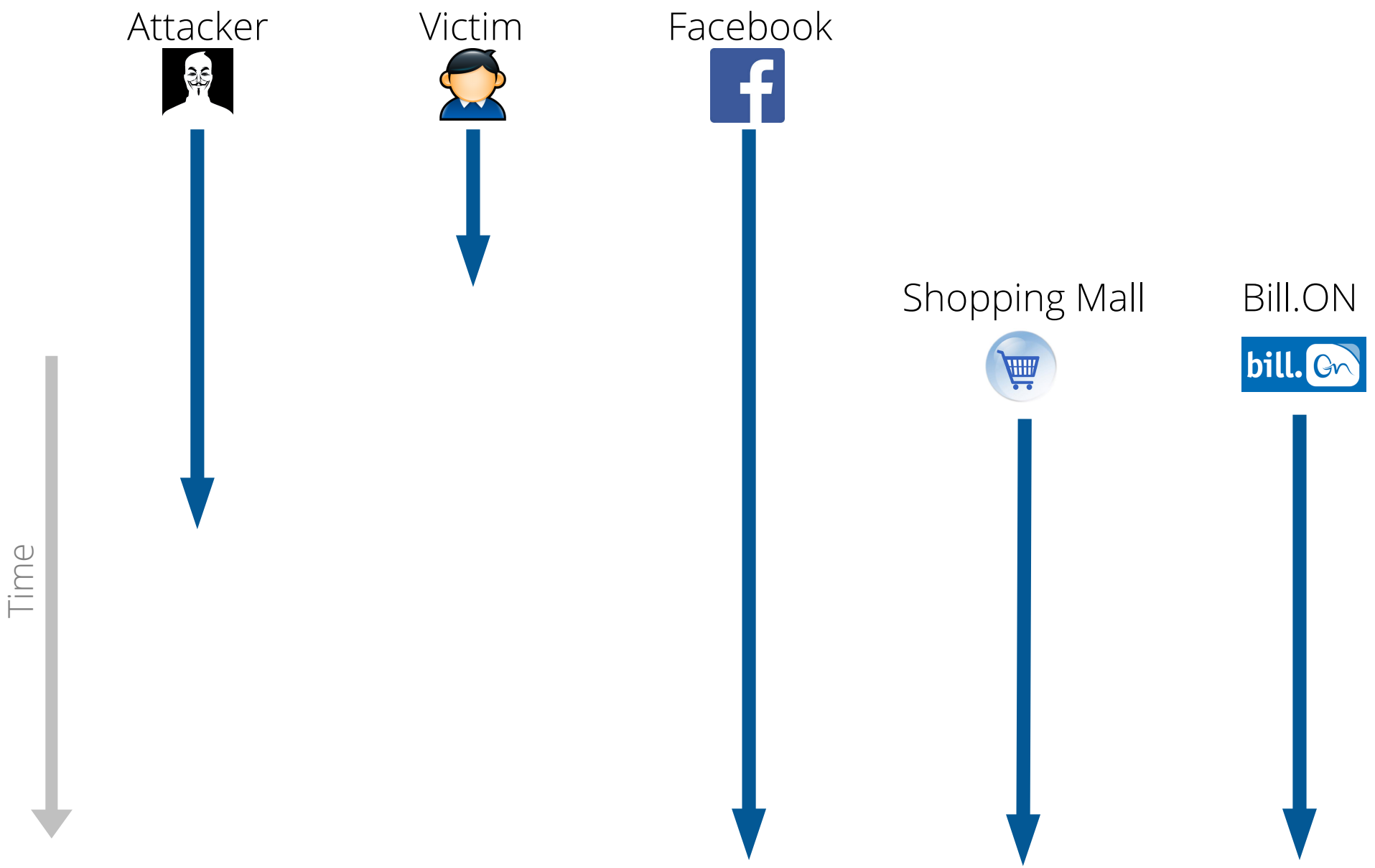
Victim

Facebook

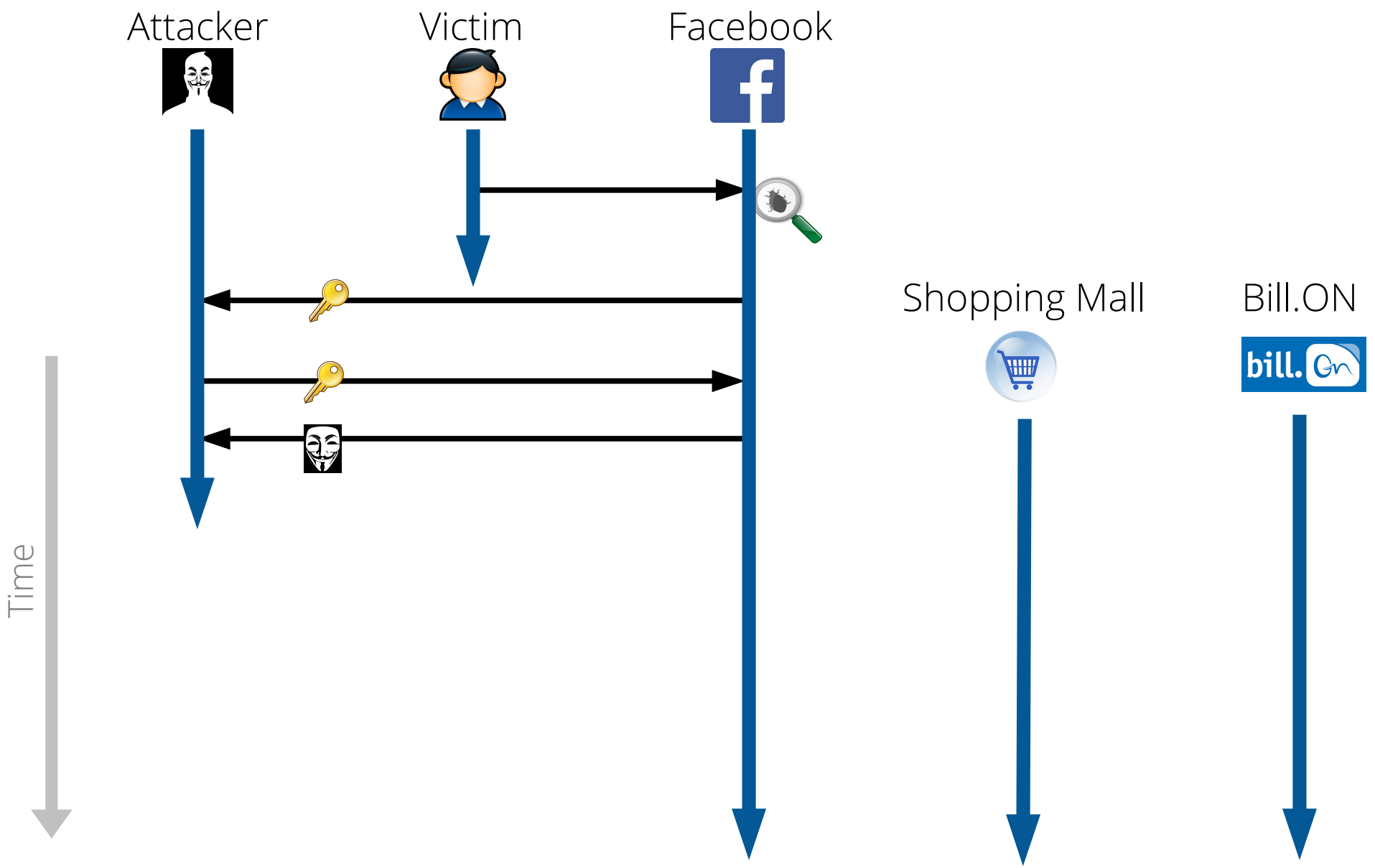
Shopping Mall

Bill.ON

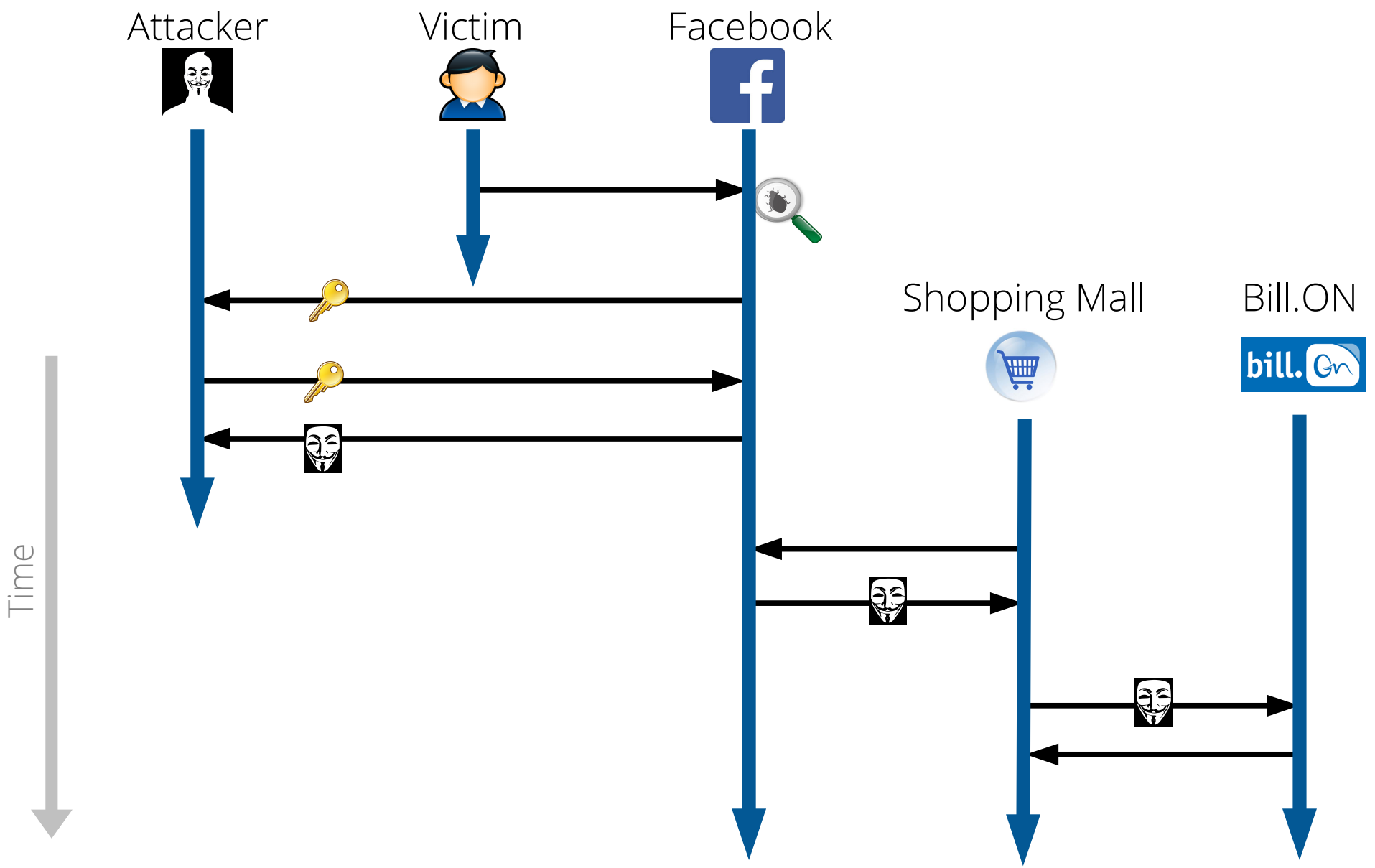
Timeline of the attack



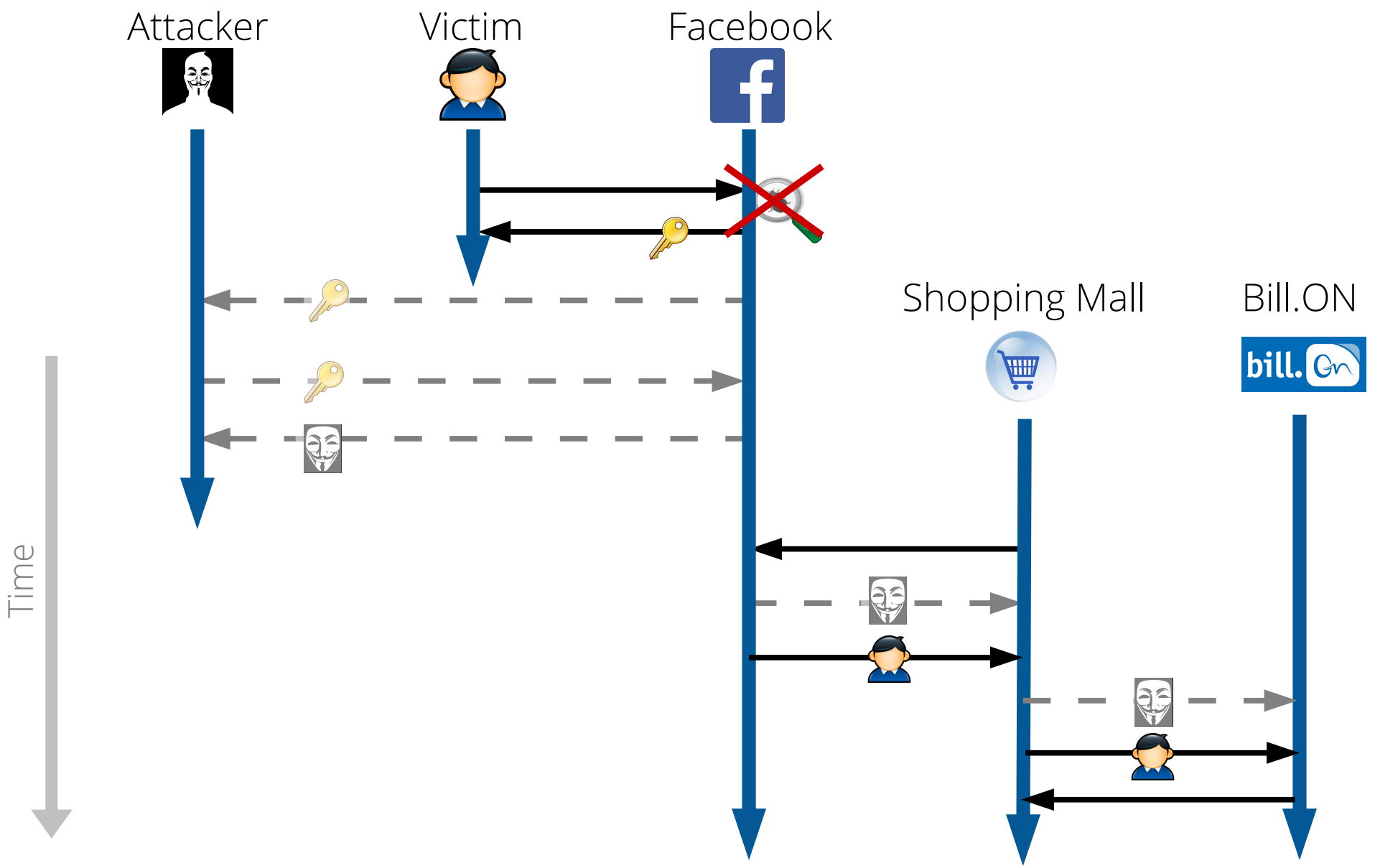
Timeline of the attack



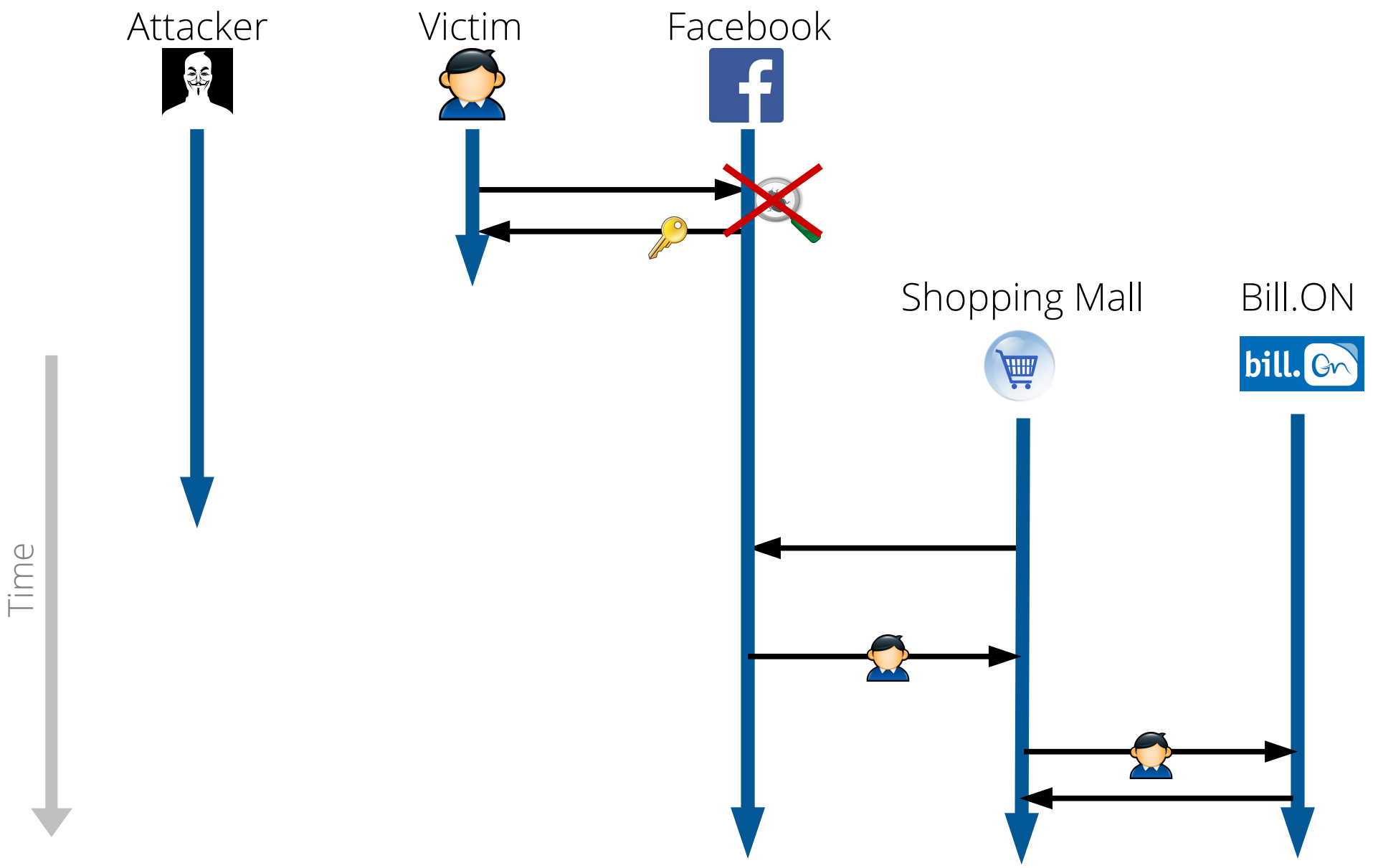
Timeline of the attack



Goal: attack did not take place



Goal: attack did not take place



Overview of system execution

- **Normal execution**:
  - Record enough information for rollback-and-replay

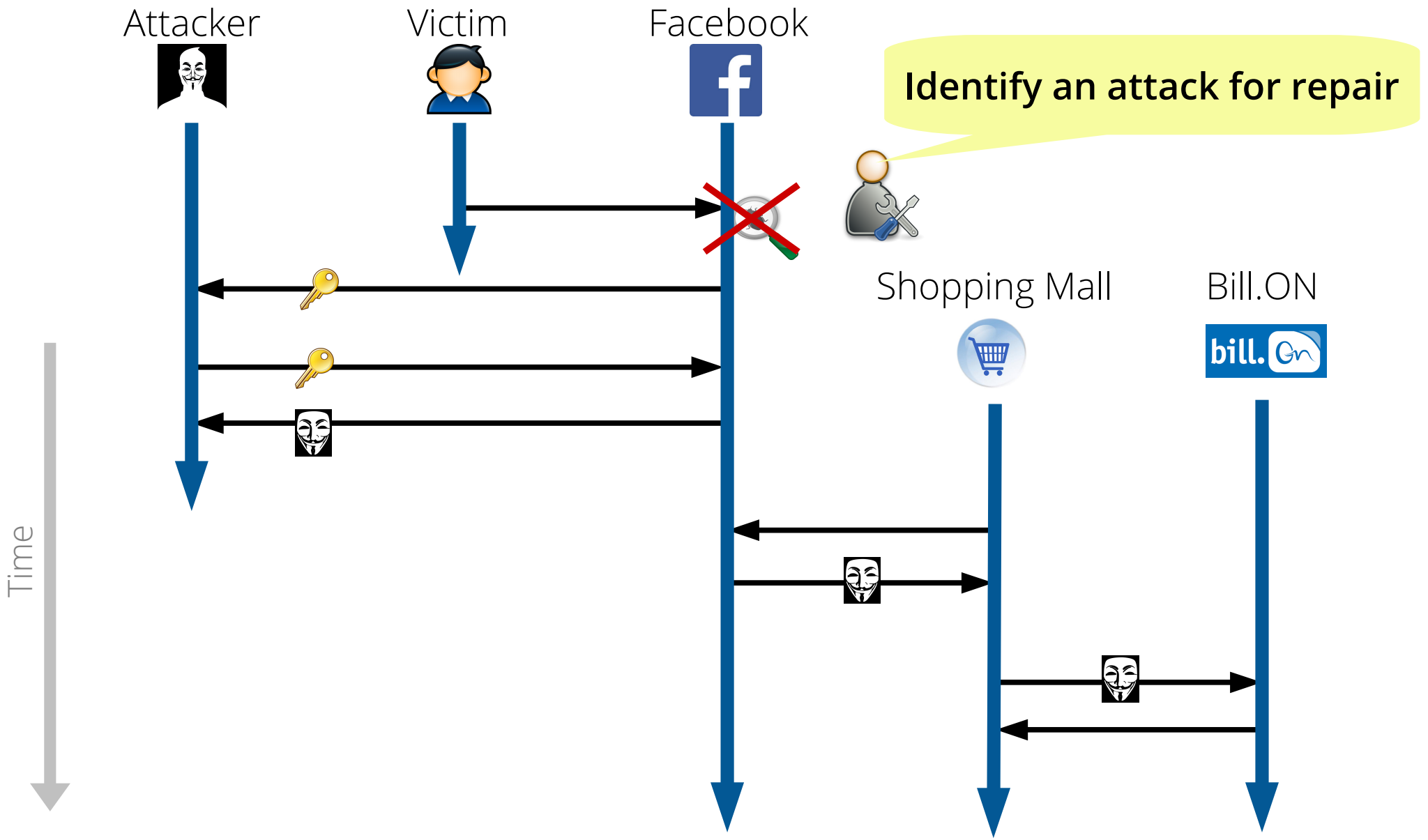
- **Repair:**
  - Identify an attack to initiate repair
  - Repair local state: rollback and replay recorded requests
  - Propagate repair whenever local repair affects others

Overview of system execution

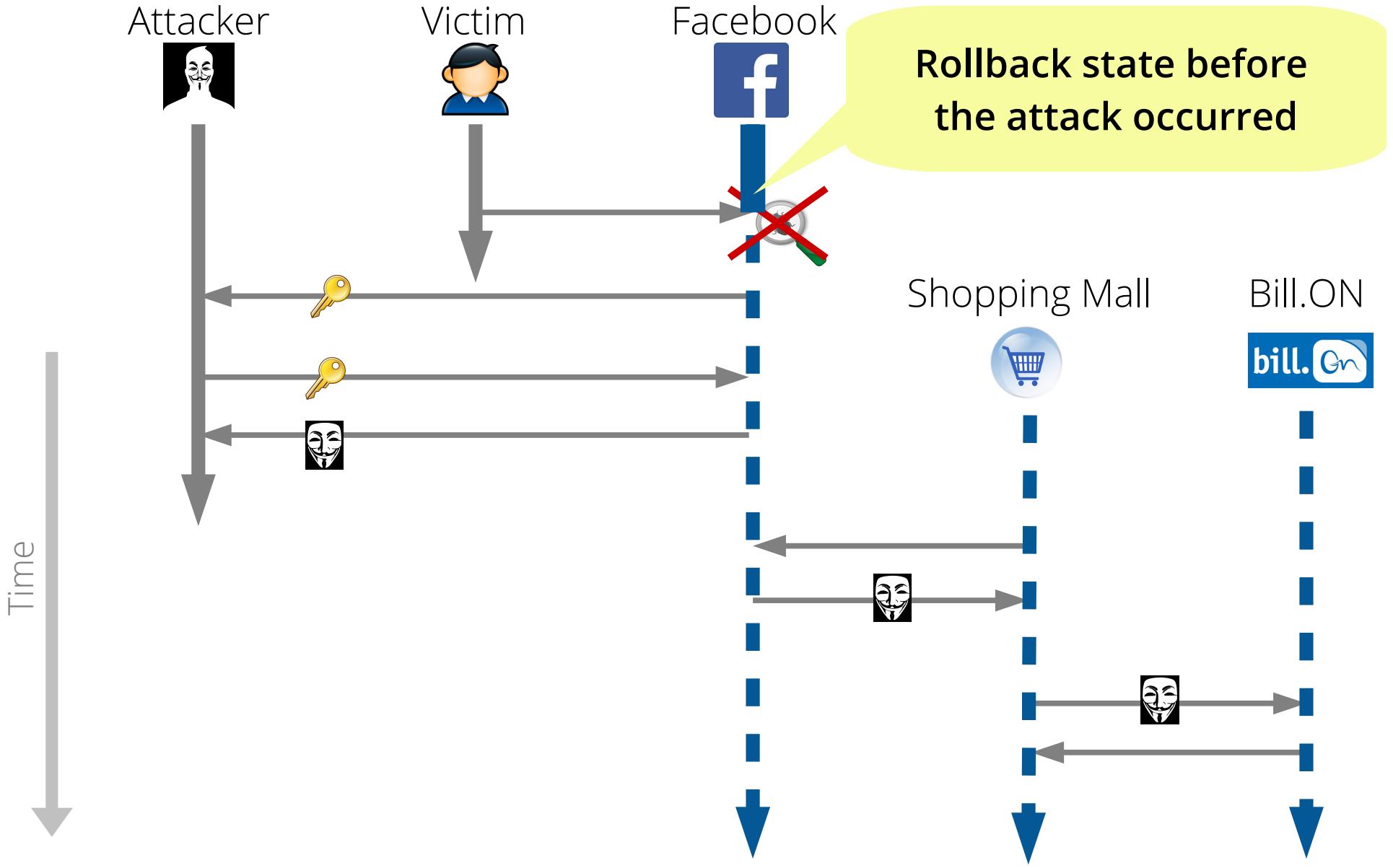
- **Normal execution**:
  - Record enough information for rollback-and-replay

- **Repair:**
  - Identify an attack to initiate repair
  - Repair local state: rollback and replay recorded requests
  - Propagate repair whenever local repair affects others

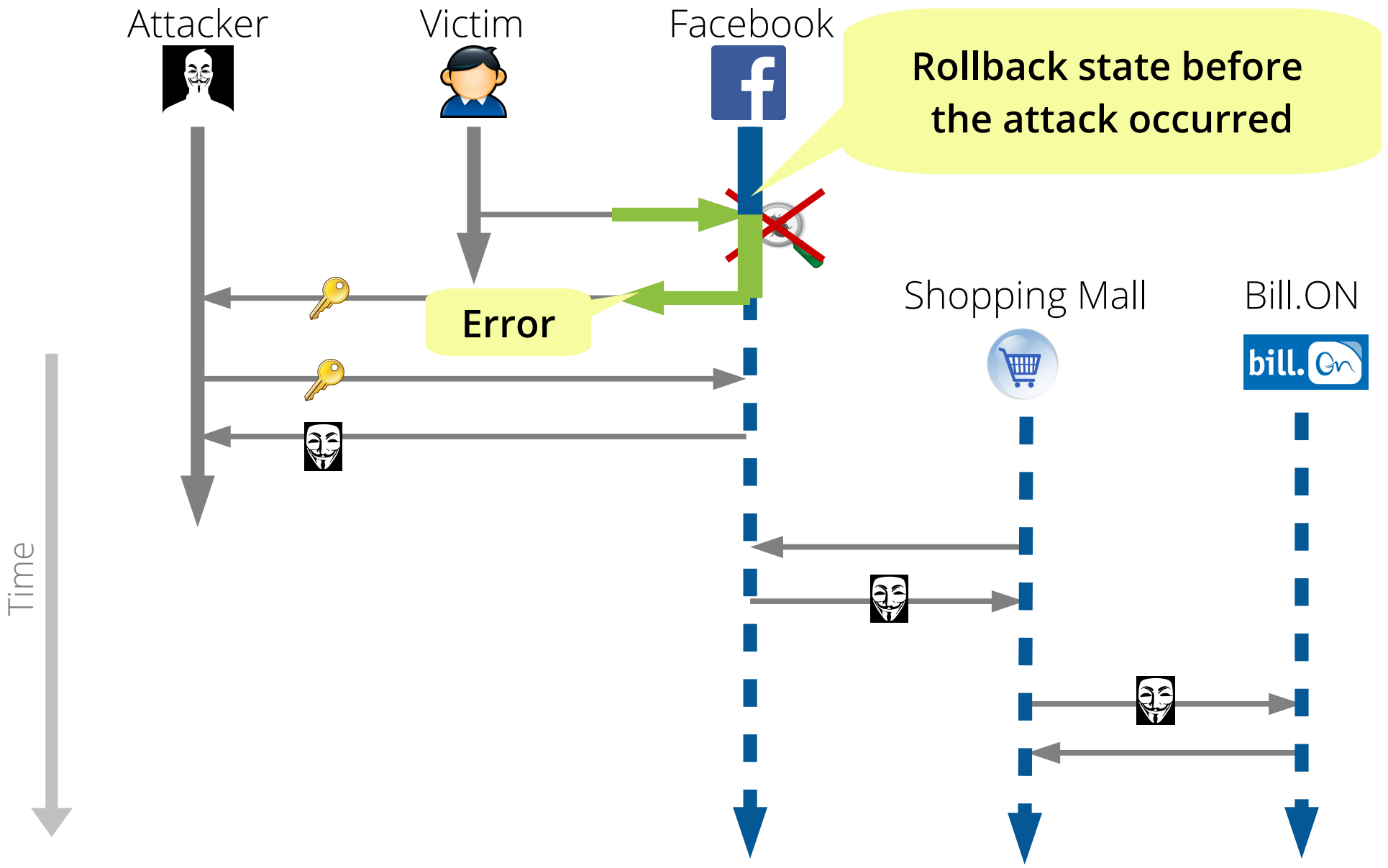
Strawman: repair with global coordinator using rollback-and-replay



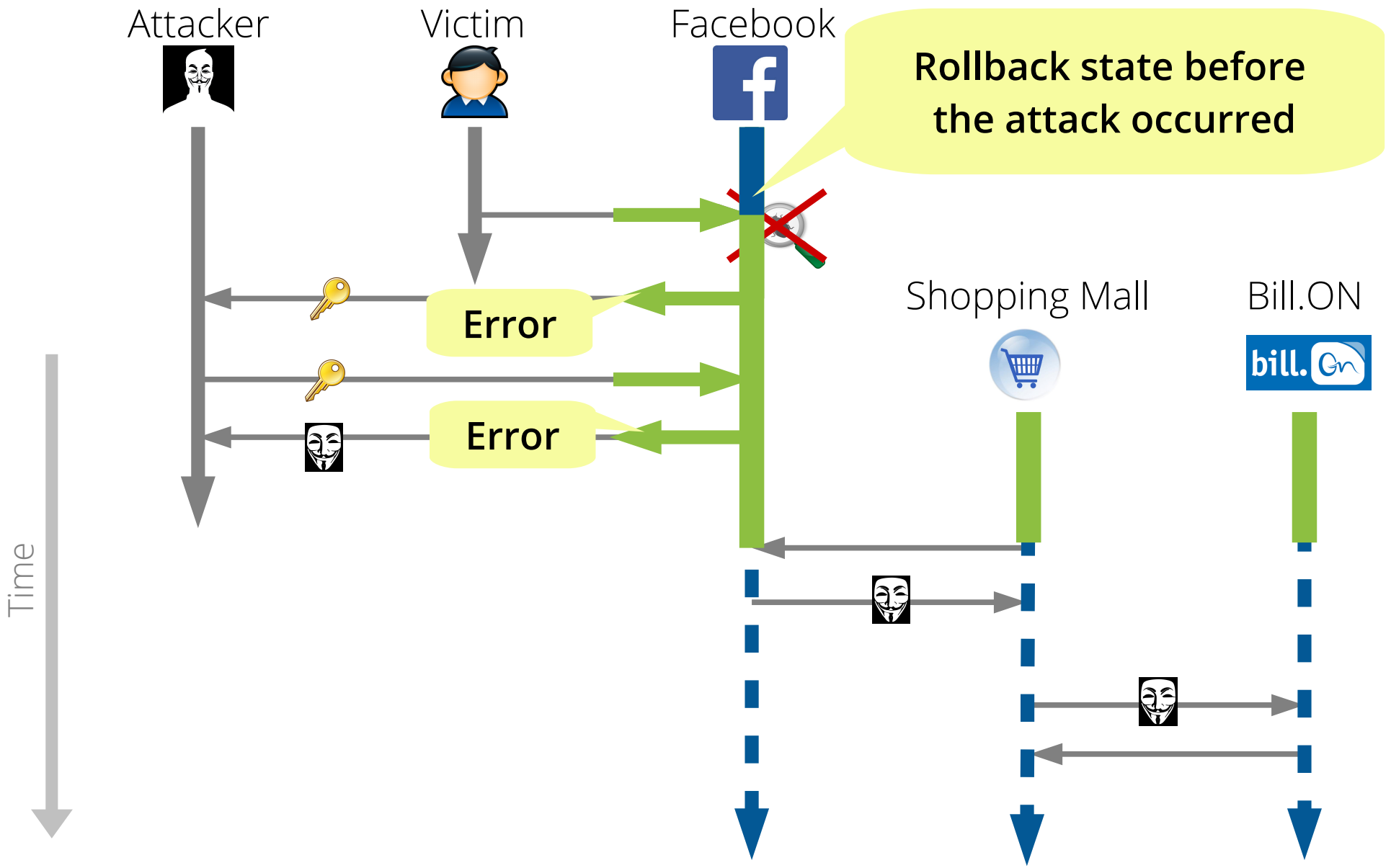
Strawman: repair with global coordinator using rollback-and-replay



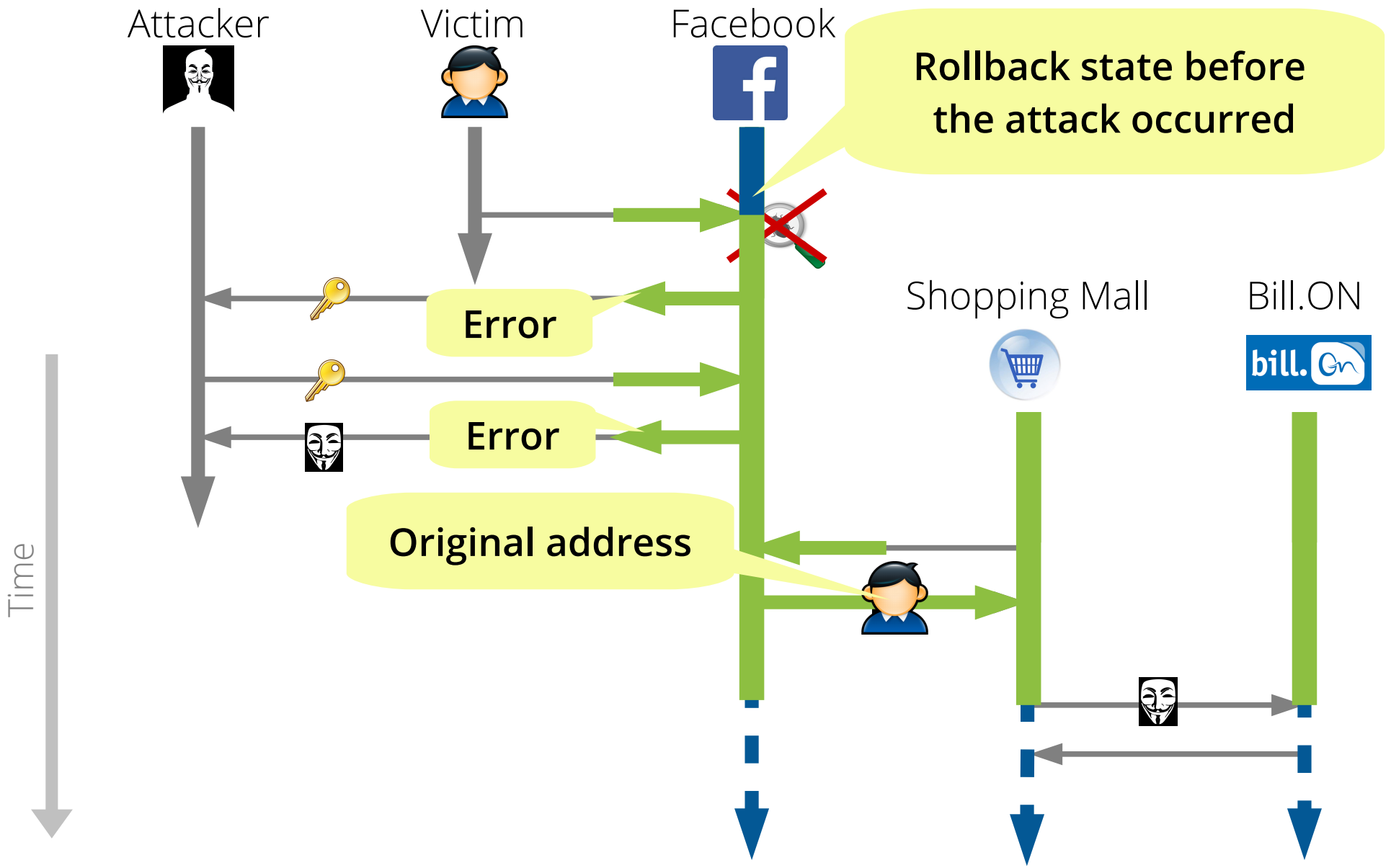
Strawman: repair with global coordinator using rollback-and-replay



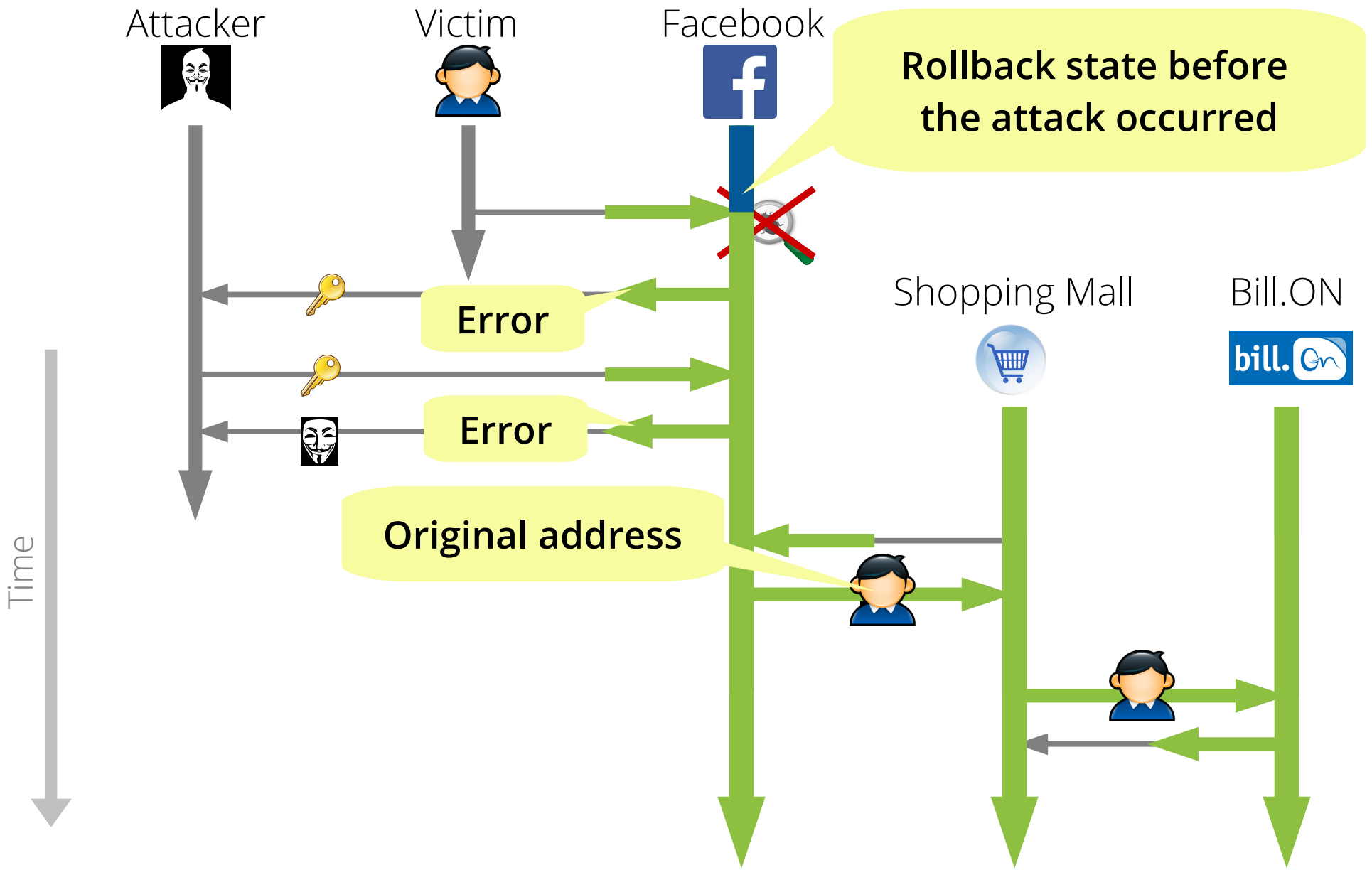
Strawman: repair with global coordinator using rollback-and-replay



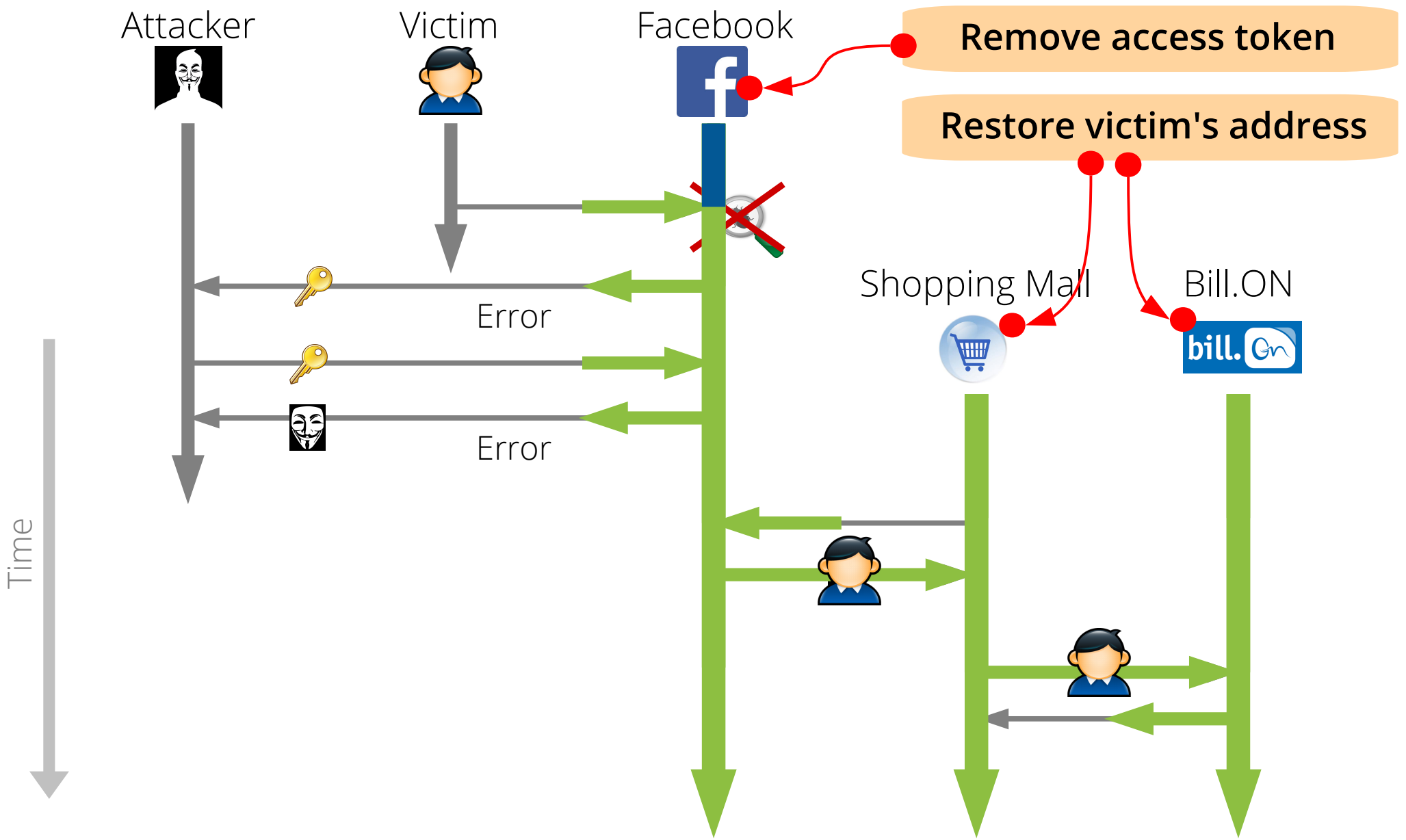
Strawman: repair with global coordinator using rollback-and-replay



Strawman: repair with global coordinator using rollback-and-replay



Strawman: repair with global coordinator using rollback-and-replay



Problems in Strawman design

- P1. All services must be available

  → Support asynchronous repair with speculation

- P2. Require global coordinator

  → Define repair APIs between services

Problems in Strawman design

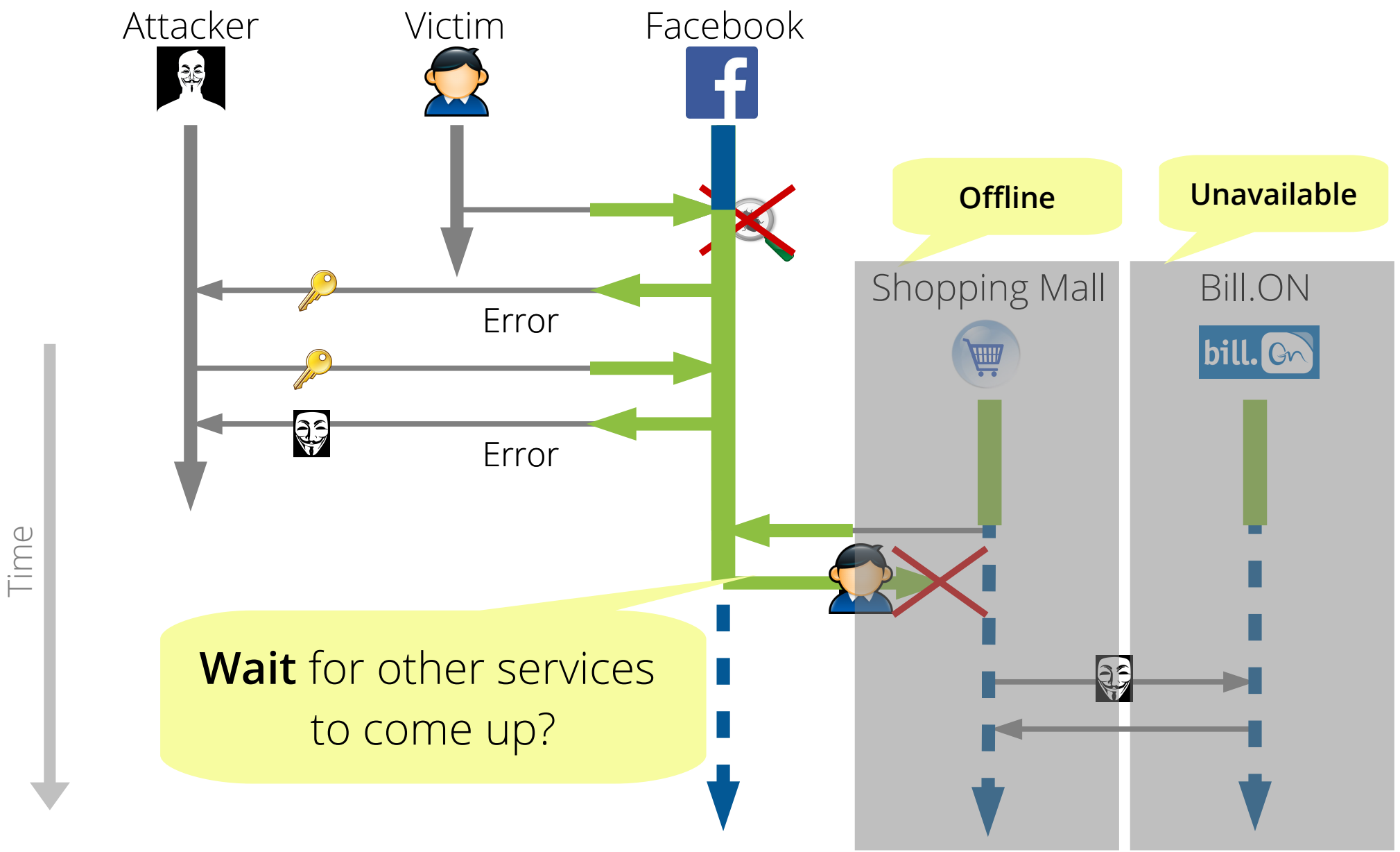
- P1. All services must be available

  → Support asynchronous repair with speculation

- P2. Require global coordinator

  → Define repair APIs between services

Challenge: cooperating with unavailable web services

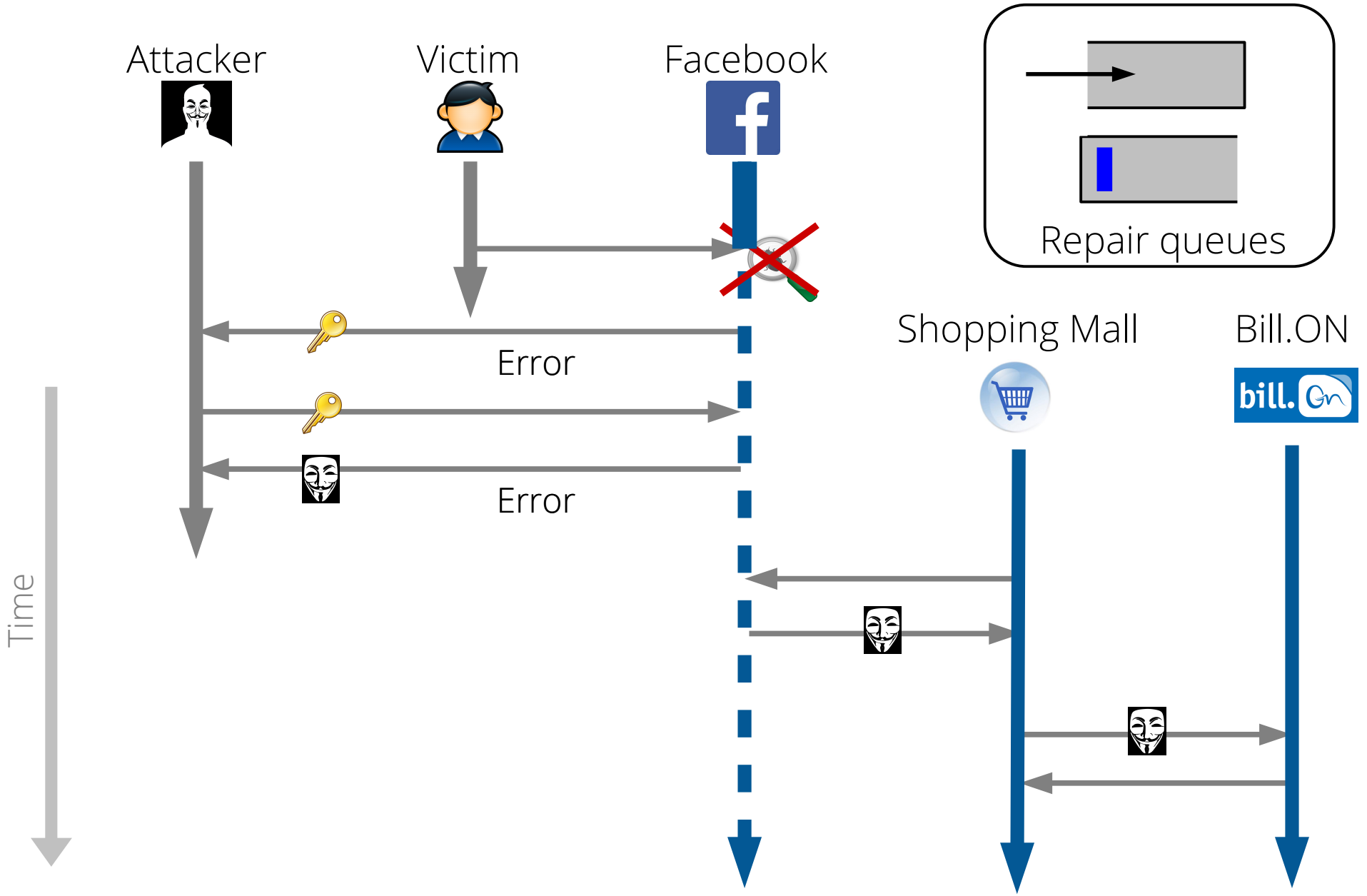


Solution: asynchronous repair

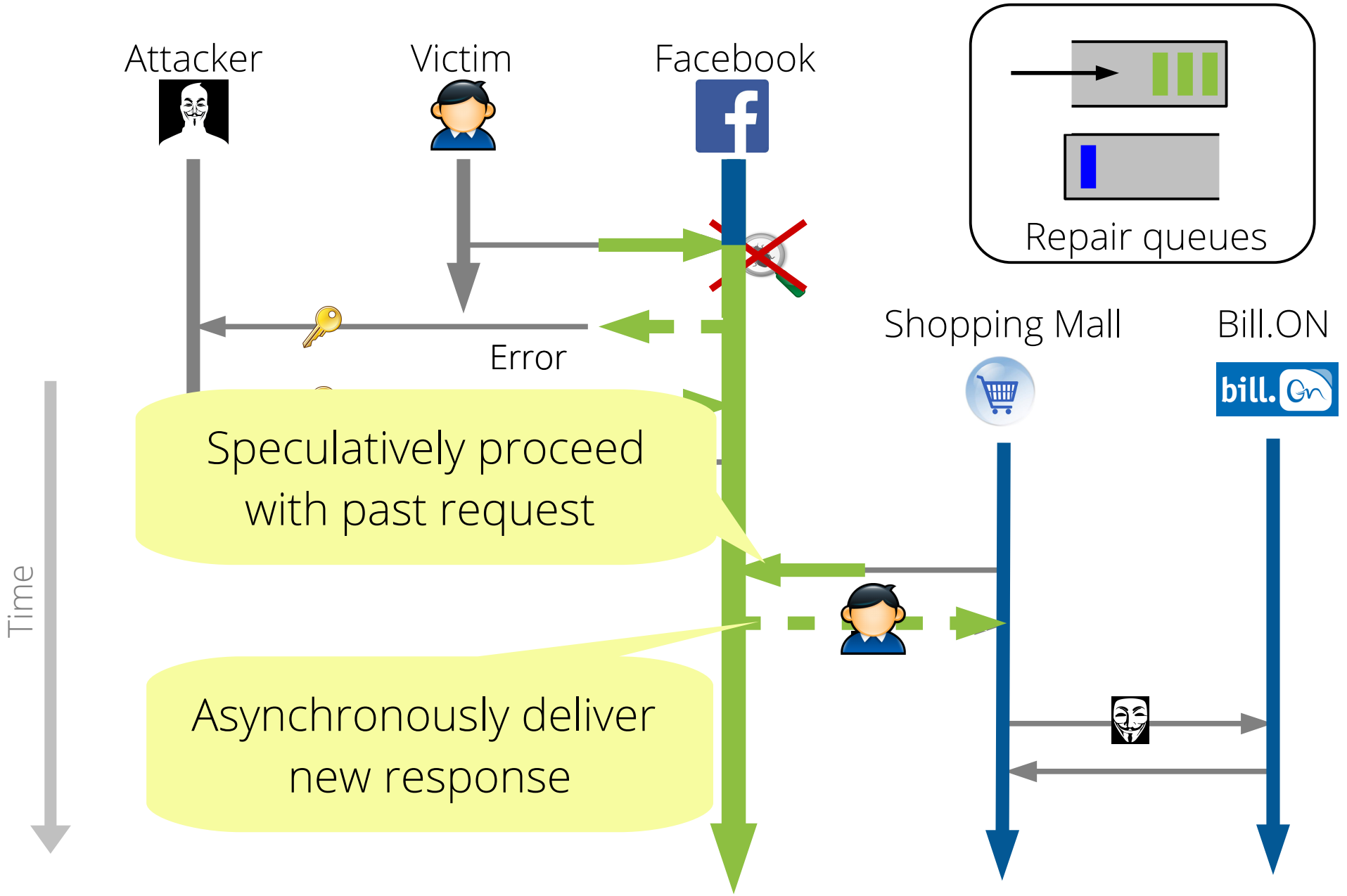
- Asynchronously deliver repair requests
- Speculatively proceed local repair with past responses (or timeout responses)
- Expose repaired state after local repair

- **Intuition:** why asynchronous repair works?
  - Many web services are designed for independent operation, prepared for handling others failures

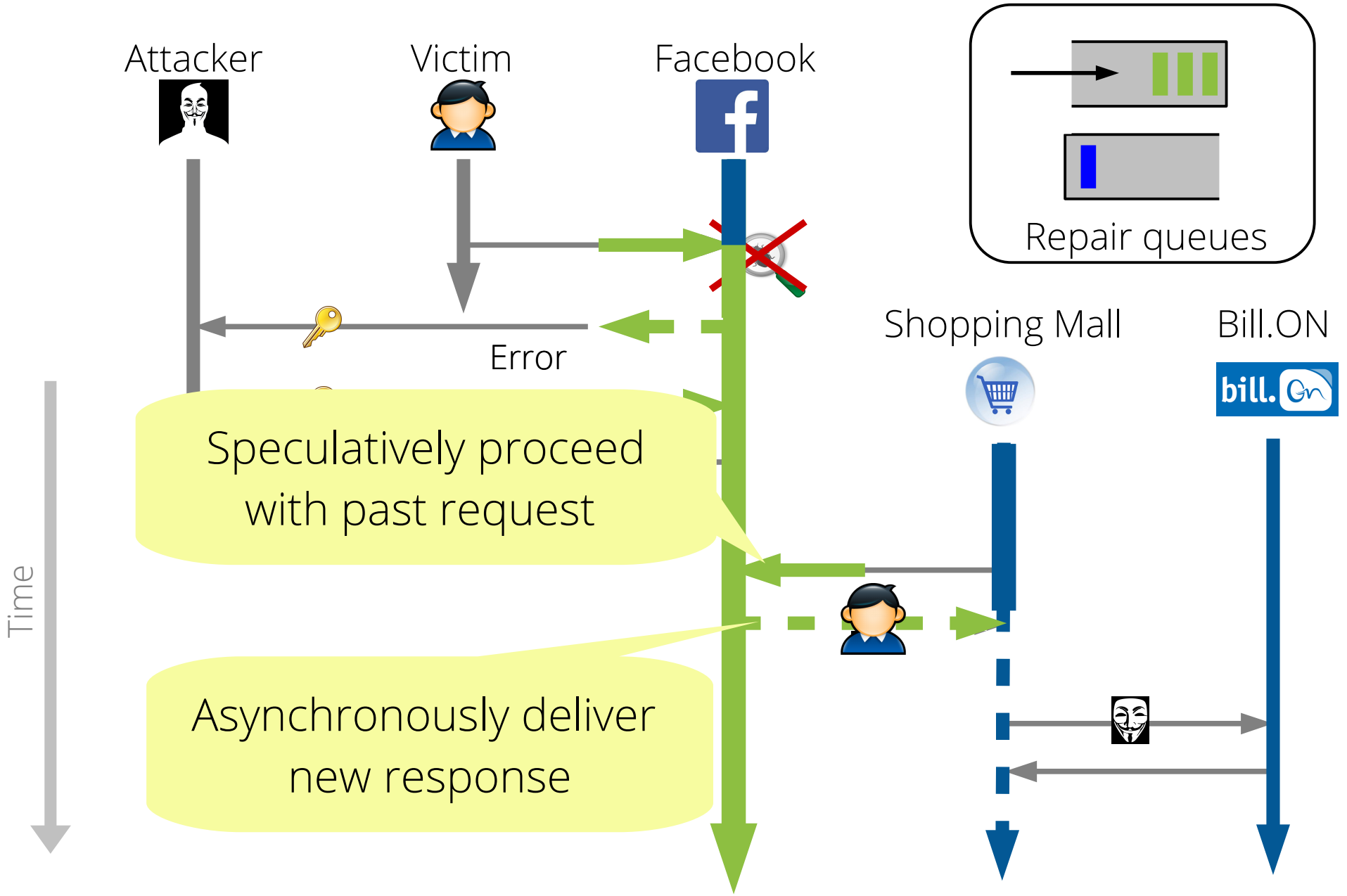
Example: asynchronous repair



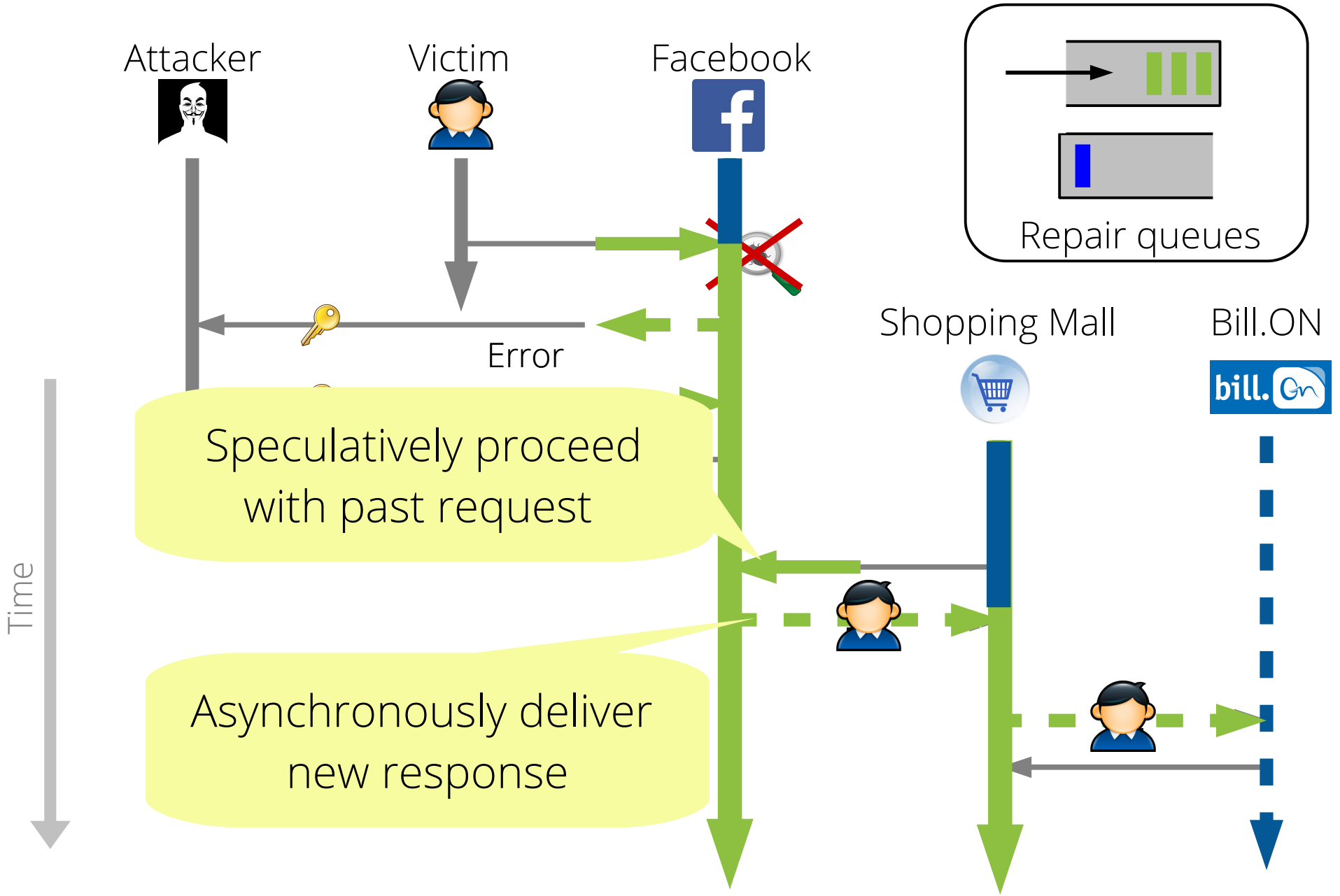
Example: asynchronous repair



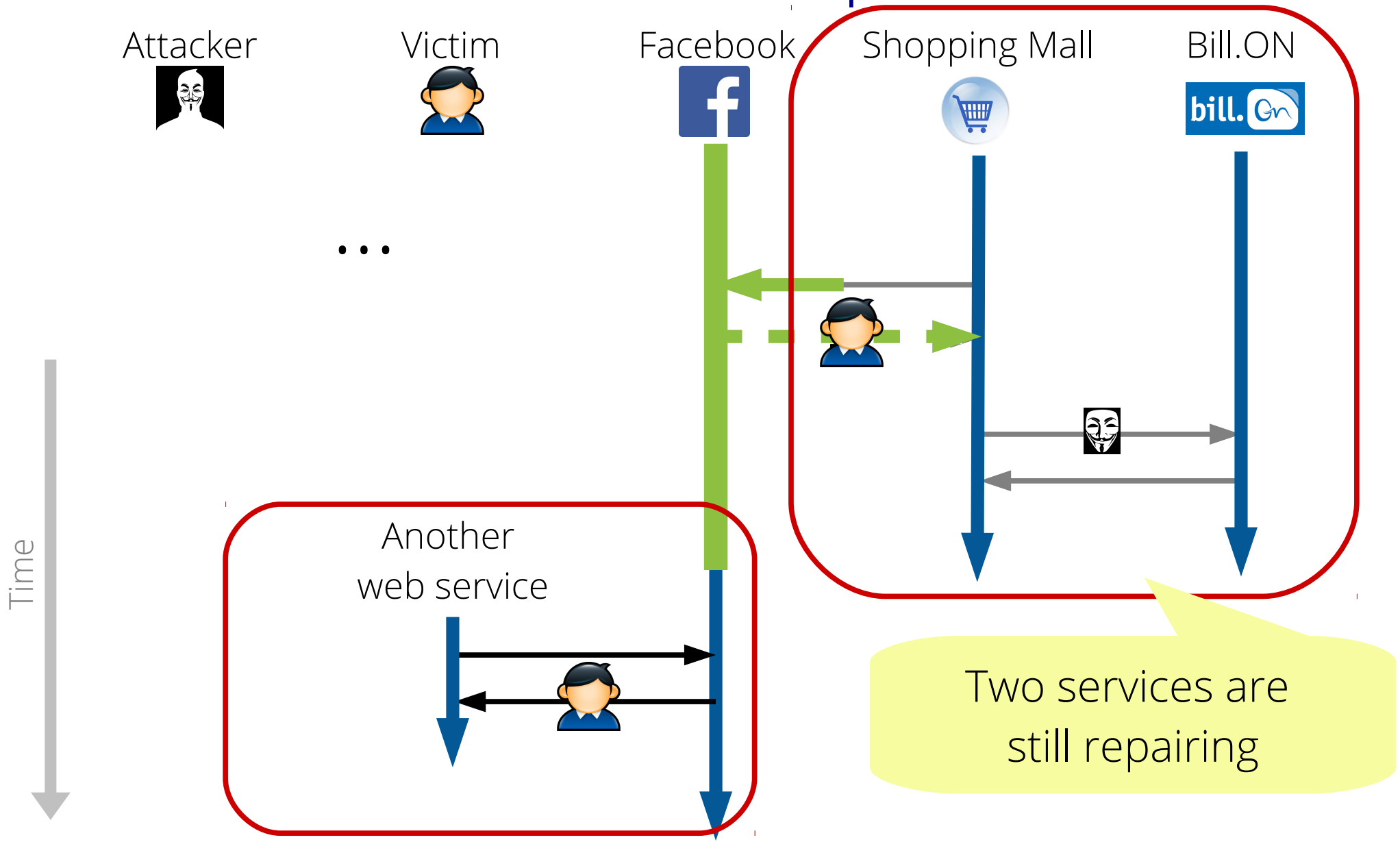
Example: asynchronous repair



Example: asynchronous repair



Example: exposing state after local repair

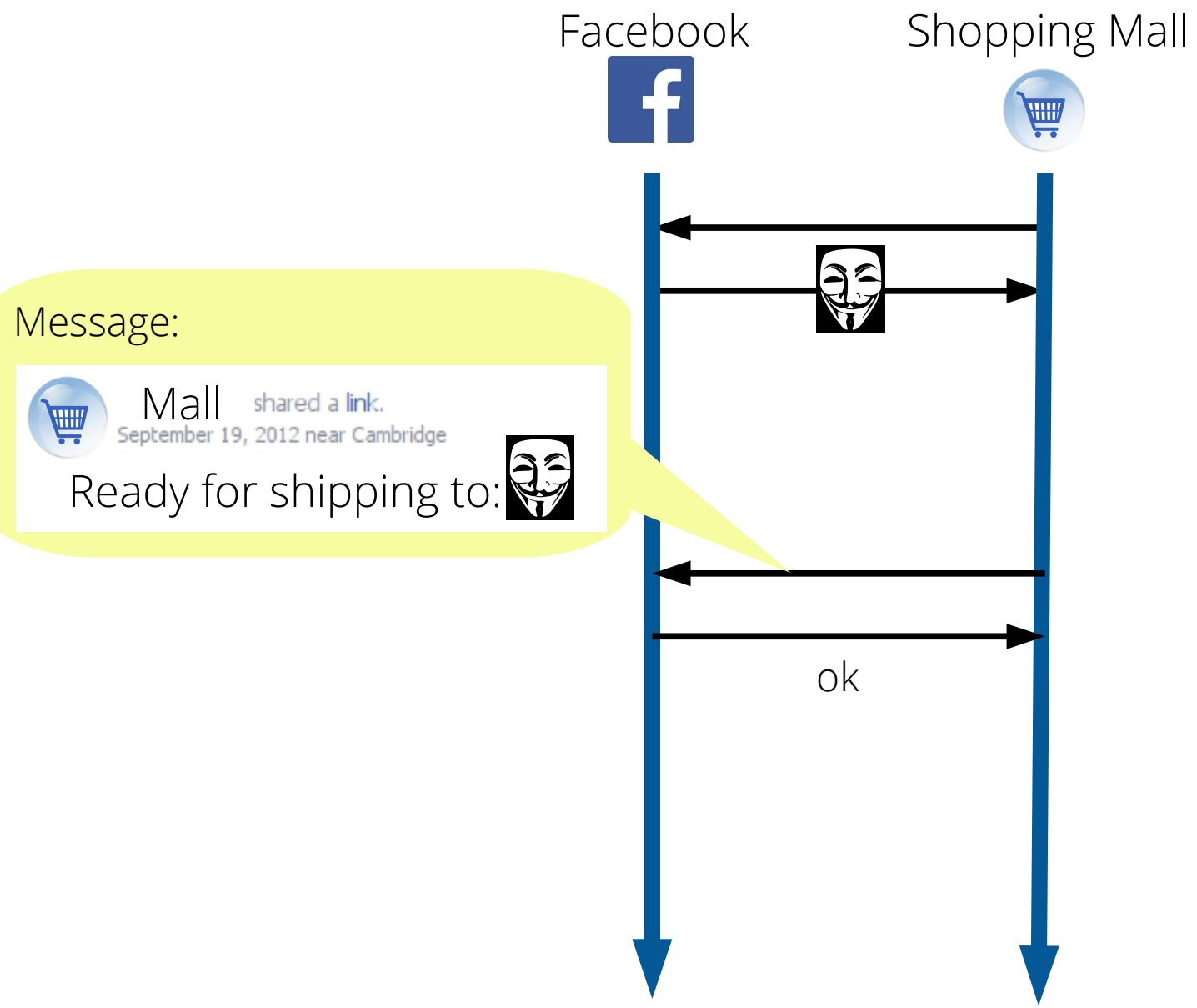


What if speculation fails?

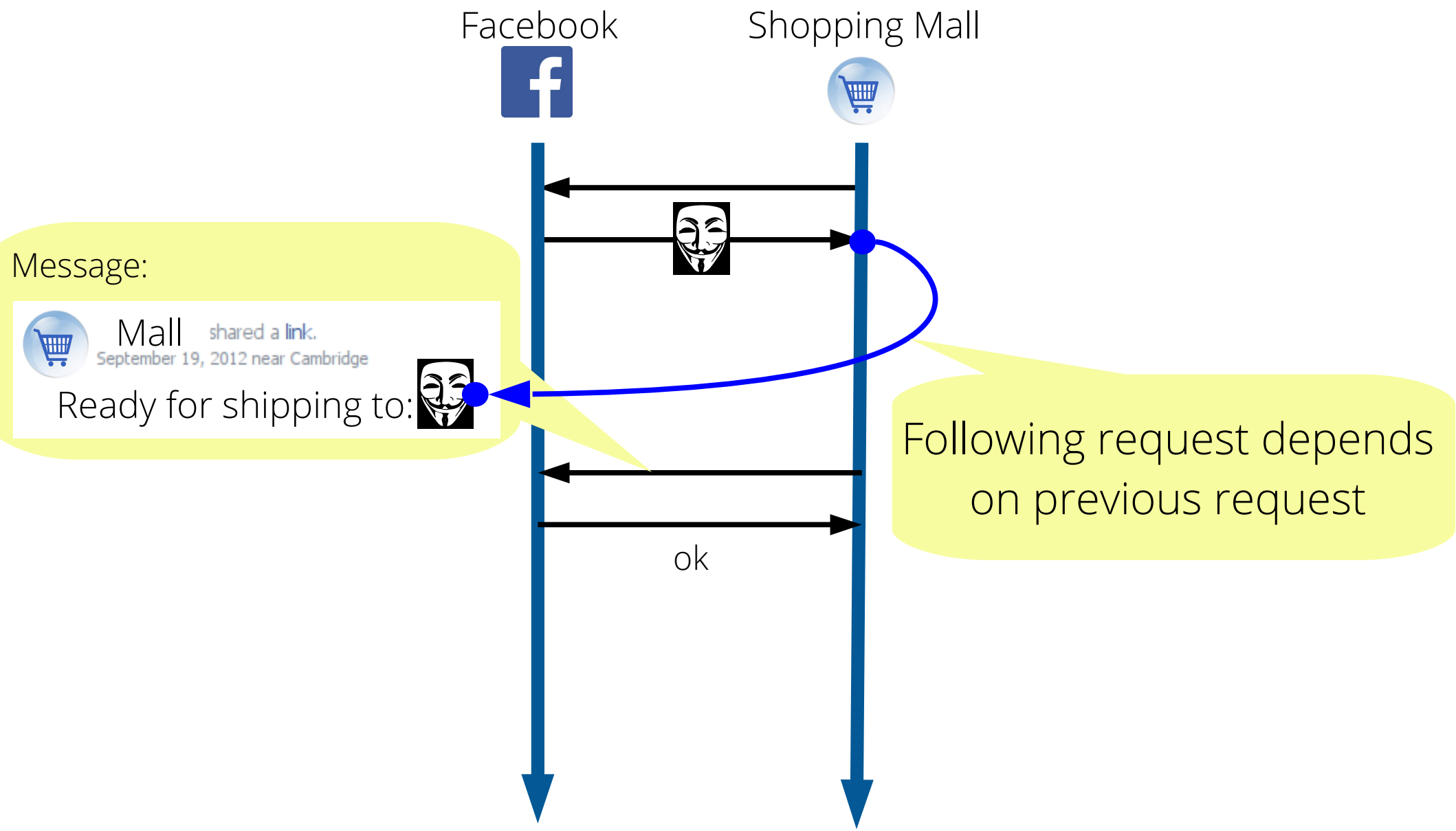
- If service responds differently,
  - **Restart local repair** with the new response
  - In fact, it is not different from initiating new repair

- Asynchronous repair will converge to the correctly repaired state at the end

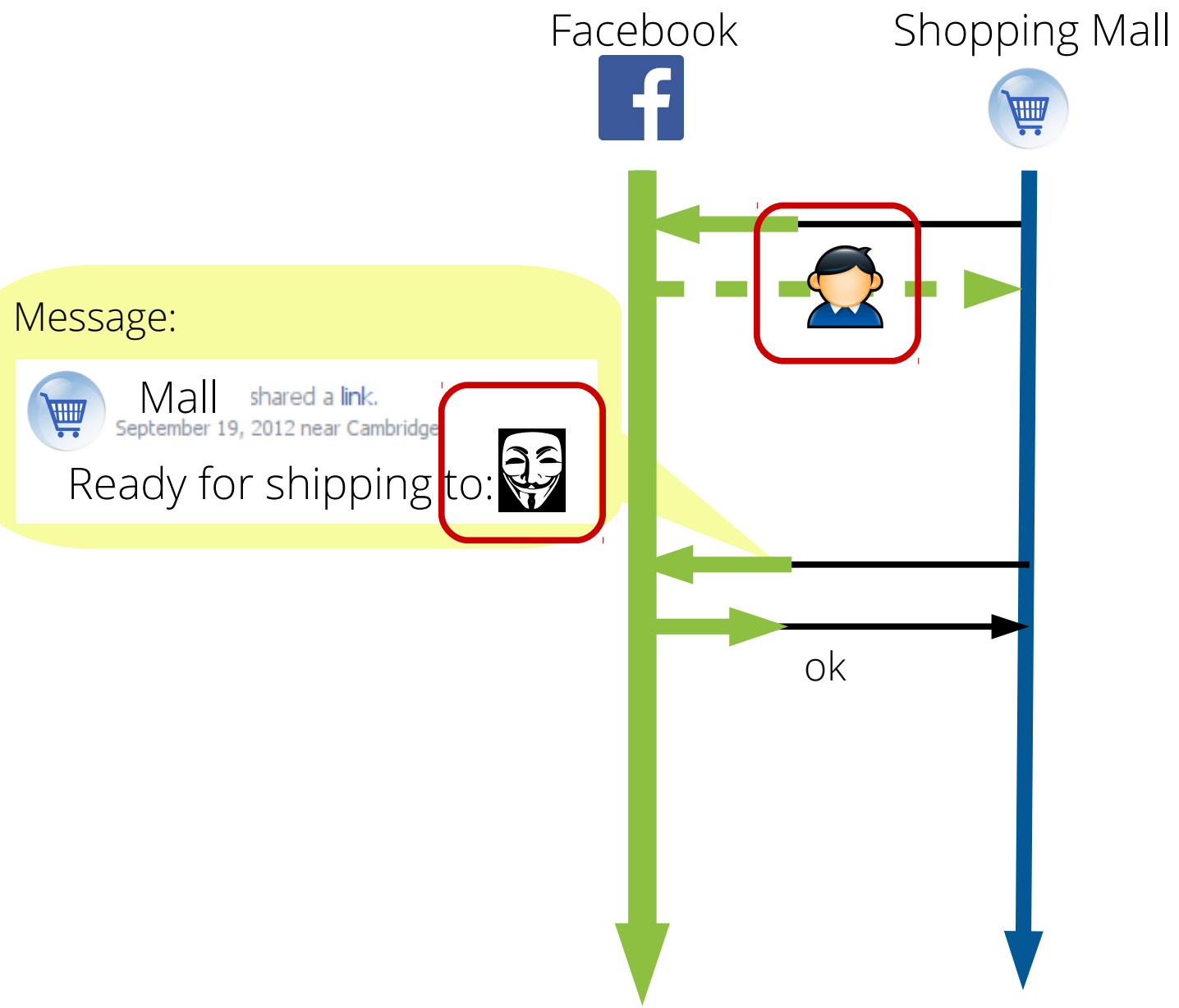
Example: speculation failure



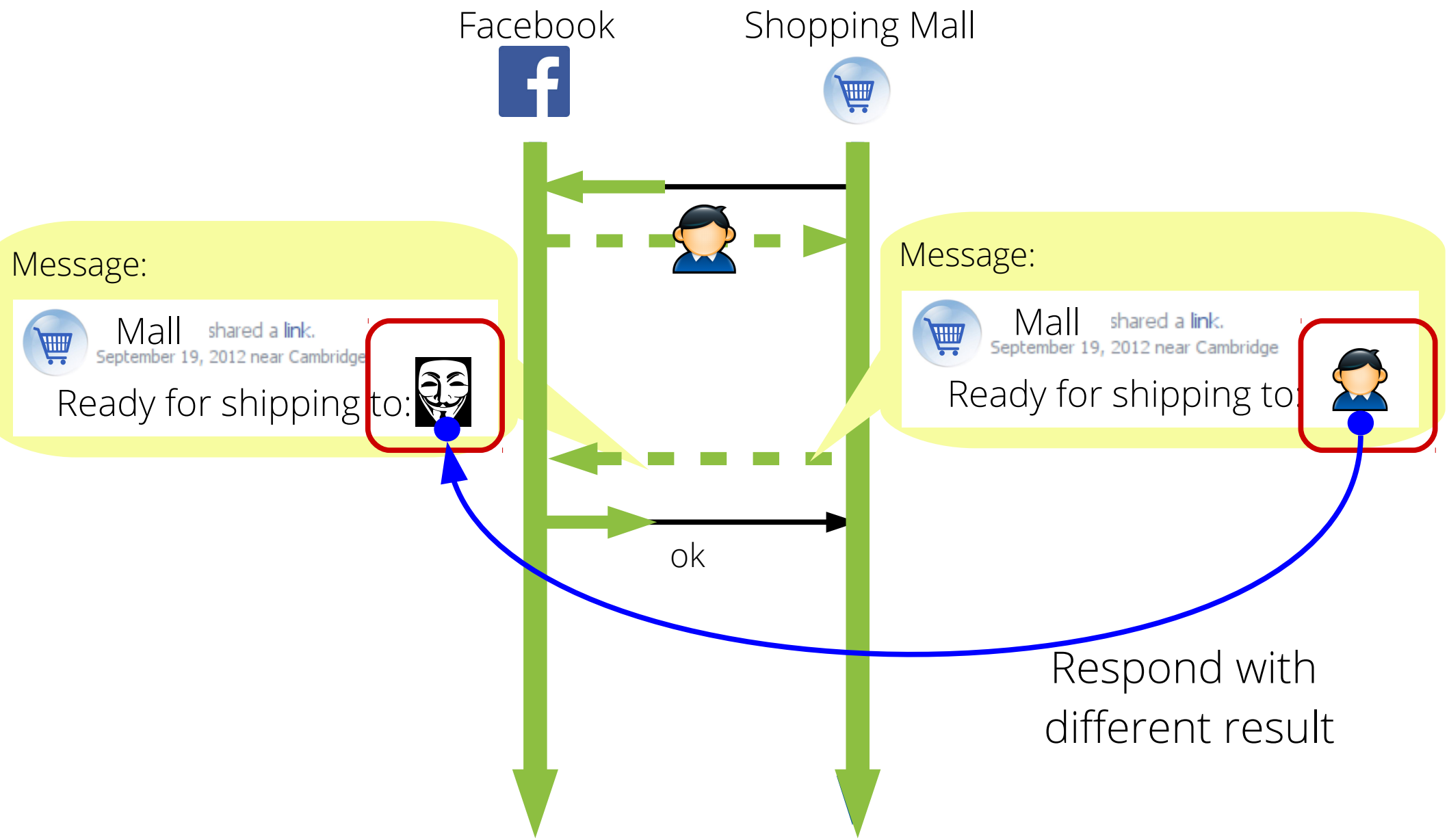
Example: speculation failure



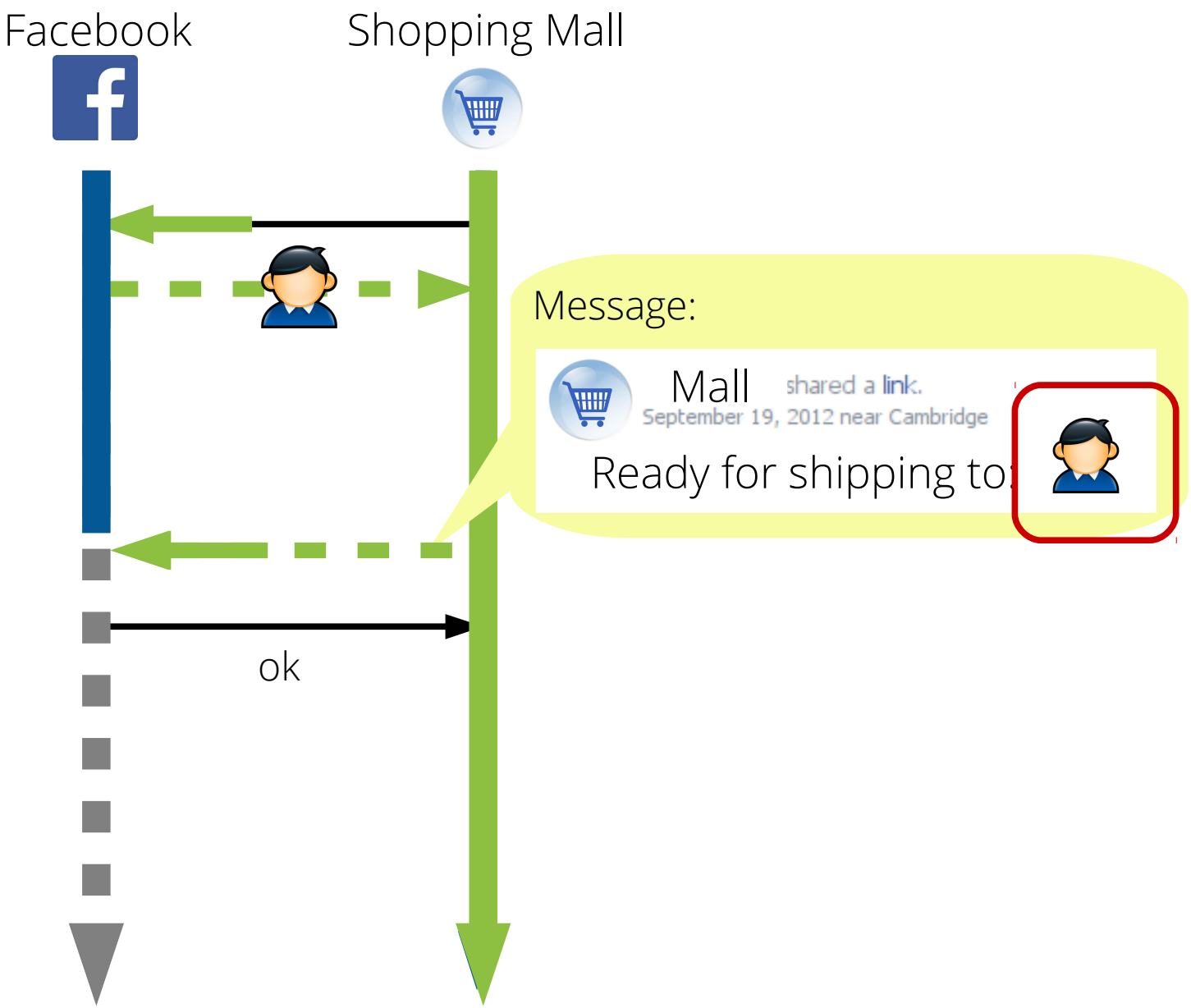
Example: speculation failure



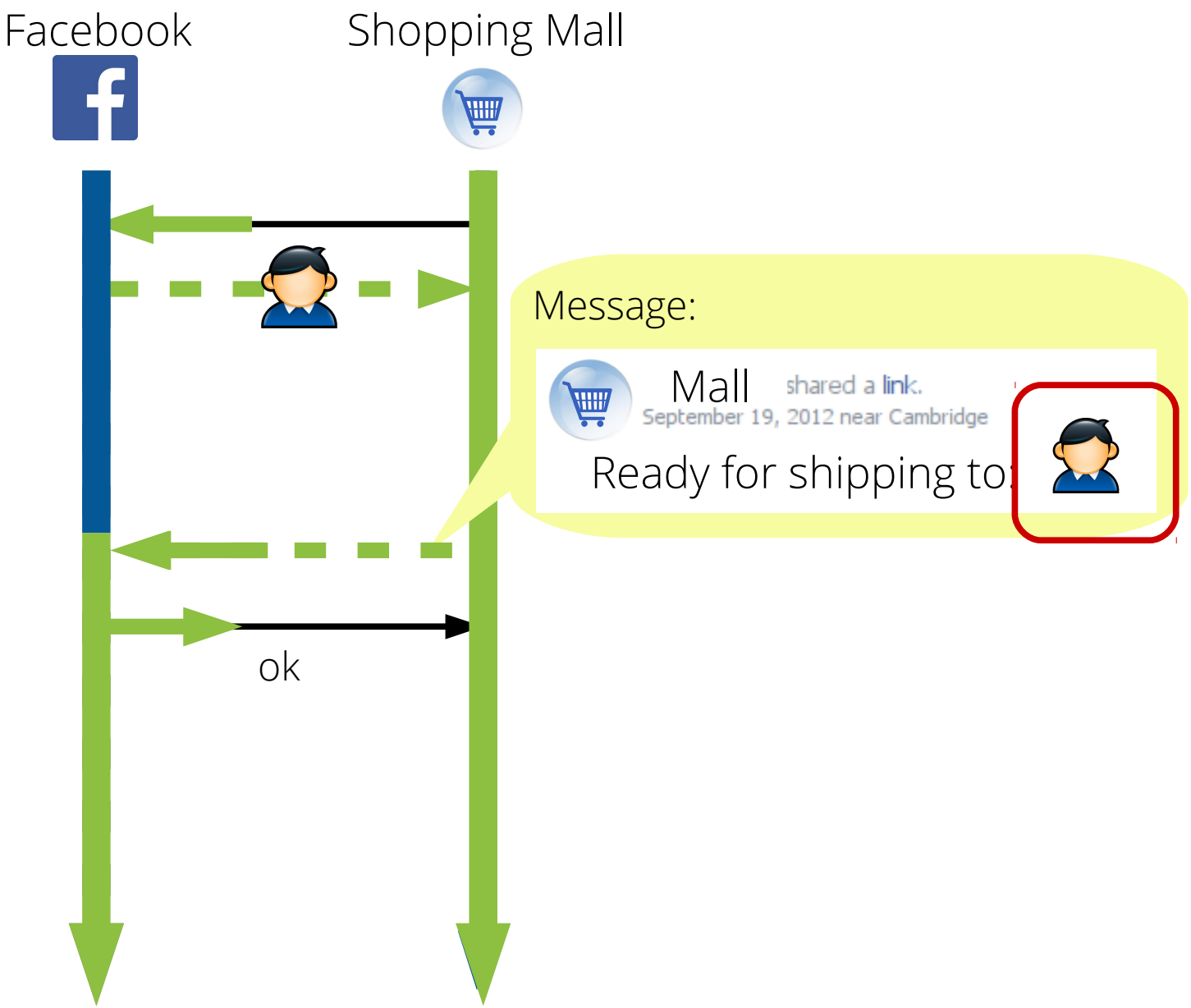
Example: speculation failure



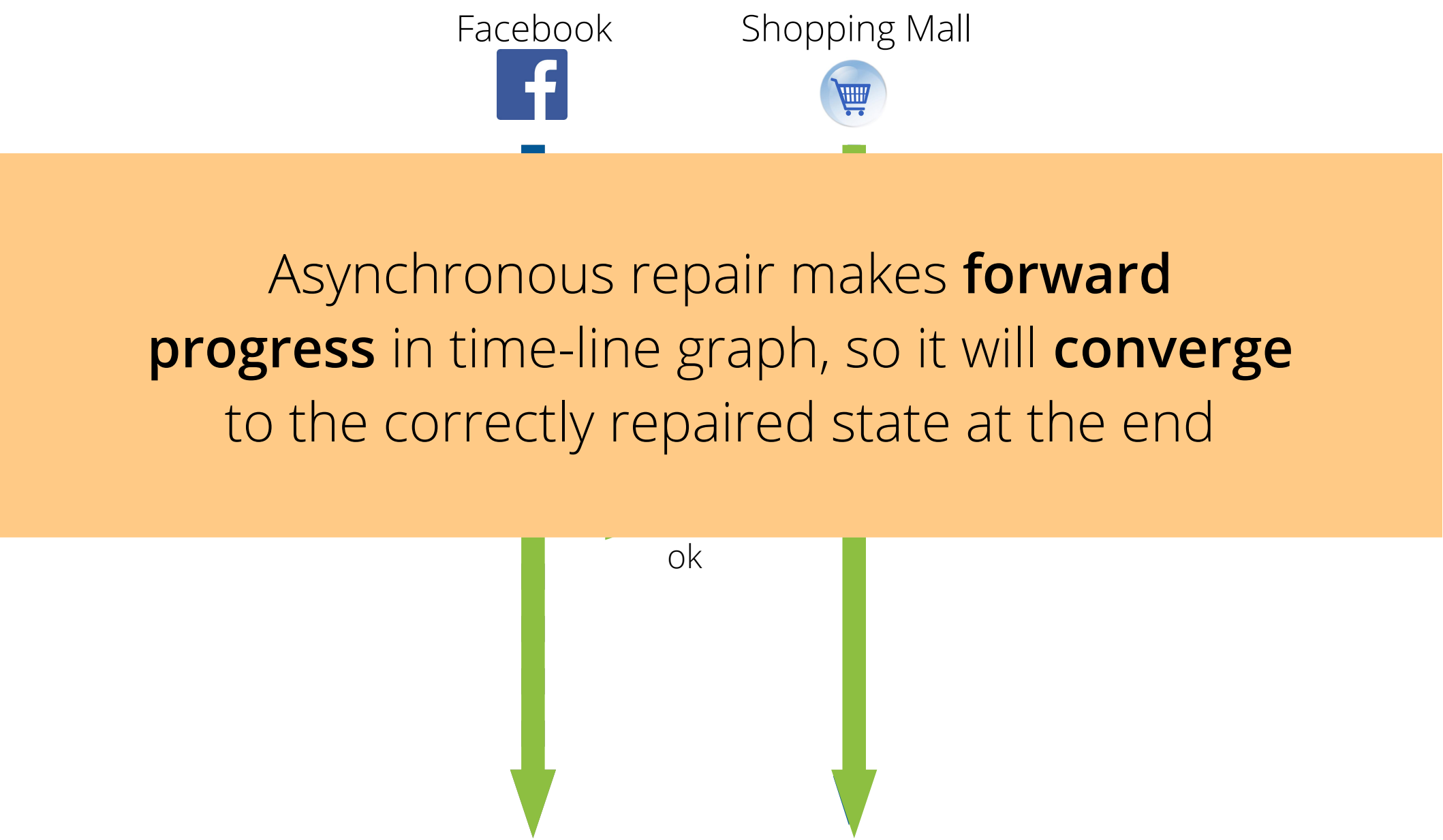
Example: speculation failure



Example: speculation failure

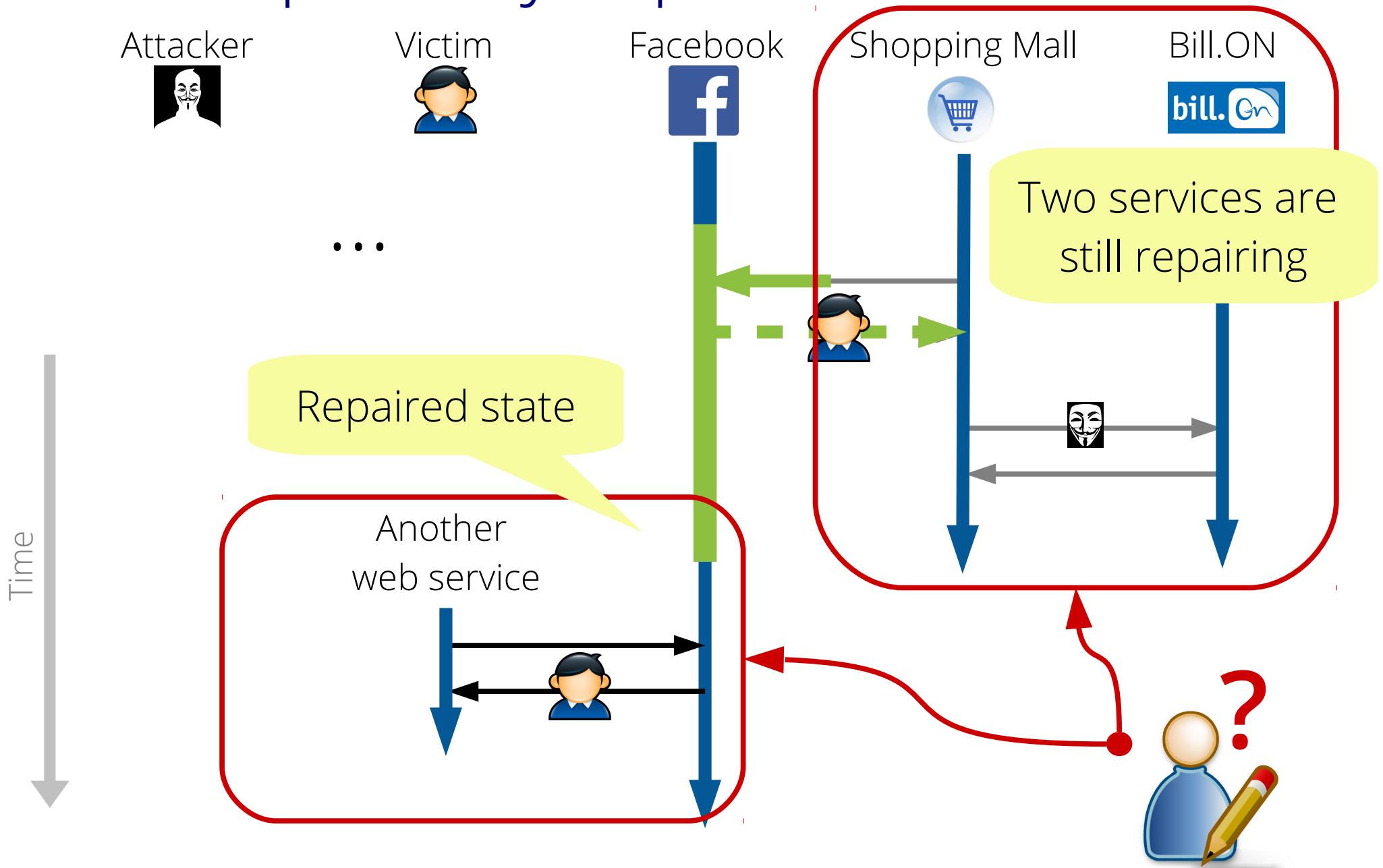


Example: speculation failure



Asynchronous repair makes **forward progress** in time-line graph, so it will **converge** to the correctly repaired state at the end

Consistency problem: partially repaired state



Consistency problem: partially repaired state

- Exposing partially repaired state might diverge global state
  - But it is not something new that our recovery mechanism introduces more
  - Most of web services **already cope with this problem**

Exposing partially repaired state might violate service invariants

- **Service invariants**: guarantees by service provider

  (e.g., locking serivce: when lock is held, no concurrent access)
- **In theory: yes** (for arbitrary tightly coupled systems)
- **In practice: no**
  - RESTful APIs usually provide consistency per API
  - Web services are in nature **loosely coupled**

Consistency: partial repair state

- Services and clients **already deal with concurrency**
- Repair of a service is modeled as:
  - Being performed by a concurrent **repair actor client**
  - That uses the service's regular API calls
- So, partially repaired state can be considered as state resulting from yet another concurrent operations

Problems in Strawman design

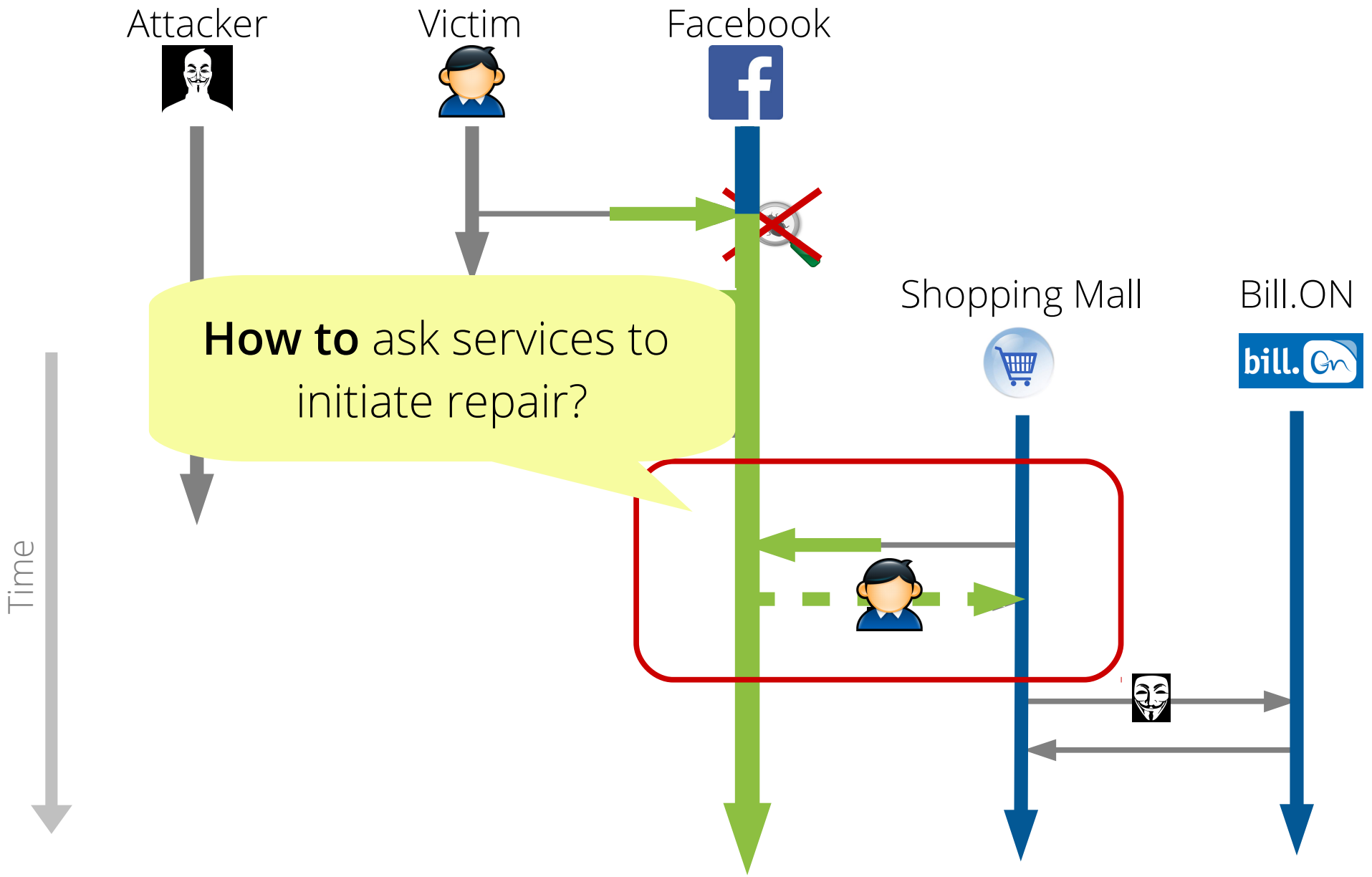
- P1. All services must be available

  → Support asynchronous repair with speculation

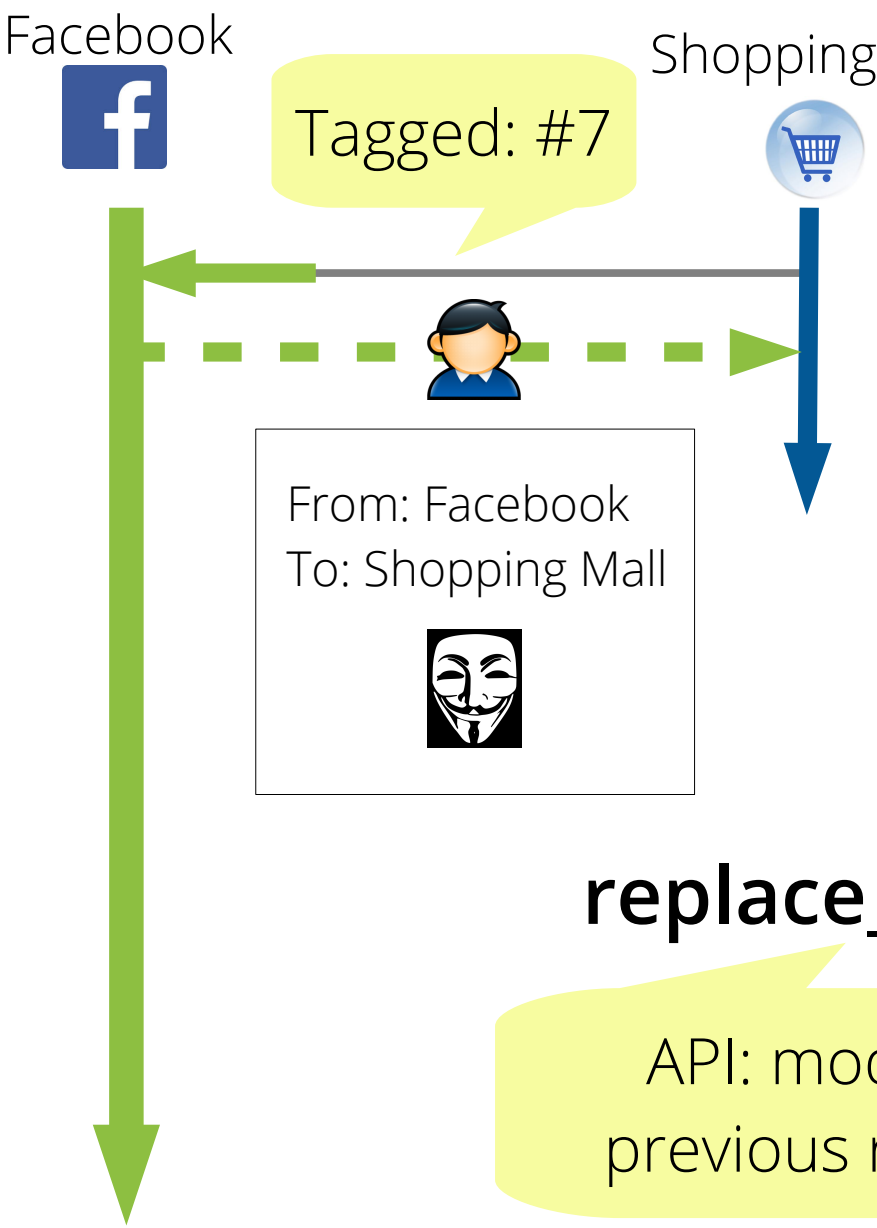
- P2. Require global coordinator

  → Define repair APIs between services

How to propagate repair requests without global coordinator?



Requesting repair with APIs (RPC)



- **Tag** each request in normal exec.
- Each service runs **repair controller**

New response (repaired)

Tag

replace_response(#7, From: Facebook To: Shopping Mall)

API: modify the previous response

Repair APIs (RPC)

- No centralized coordinator, each server invokes following repair APIs to recover from the attack
  - **replace_response**(tag, resp): replace past response
  - **replace_request**(tag, req): replace past request
  - **delete**(tag): delete past request
  - **create**(req, before, after): execute new requests in the past

Repair APIs (RPC)

- No centralized coordinator, each server invokes following repair APIs to recover from the attack
  - **replace_response**(tag, resp): replace past response
  - **replace_request**(tag, req): replace past request
  - **delete**(tag): delete past request
  - **create**(req, before, after): execute new requests in the past

If service supports those 4 APIs,
it can participate in decentralized recovery

Authentication of repair APIs

- Too application specific
  - (e.g. Email service: sender can delete recipient's emails?)
- **Delegate** authentication to original web services
  - Implement application specific policy
    (e.g. ask admin for confirmation of repair)
  - Assign a credential to repair requests

Summary of design

1. **Asynchronous repair**
    - Proceed repair with offline or unavailable services
    - Consistency in partially repair state

2. **Repair APIs** between services
    - No central coordinator
    - Each service controls its repair
    - Delegate authentication

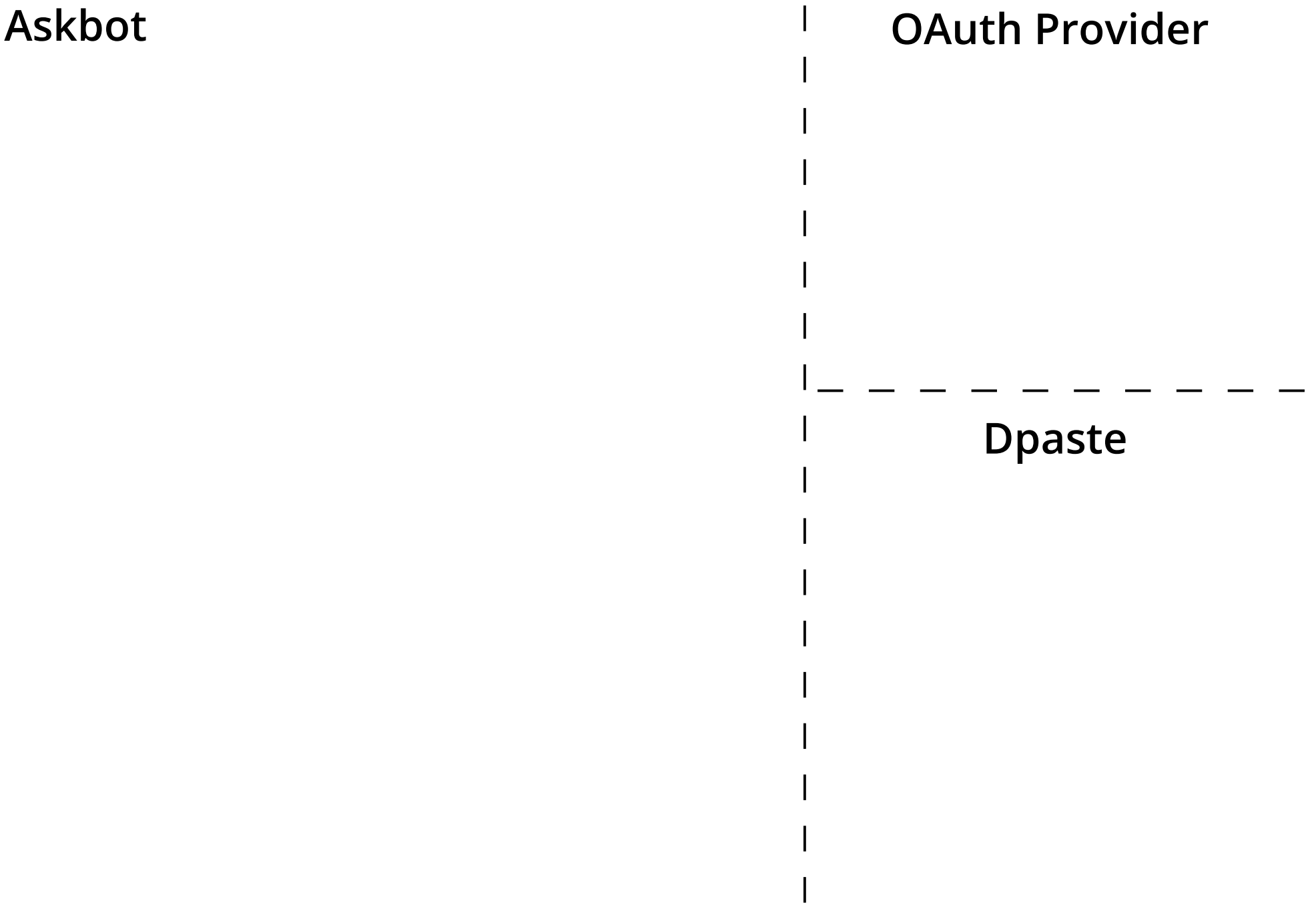
Implementation

- Prototype implementation: **Aire**
  - Extend **Django** web framework
  - Support existing Django app. with **few modifications**
    - Support Askbot, Django-OAuth, and Dpaste
    - e.g., Askbot's authentication policy: 55 LoC
  - Total: 5700 lines of Python code

Evaluation questions

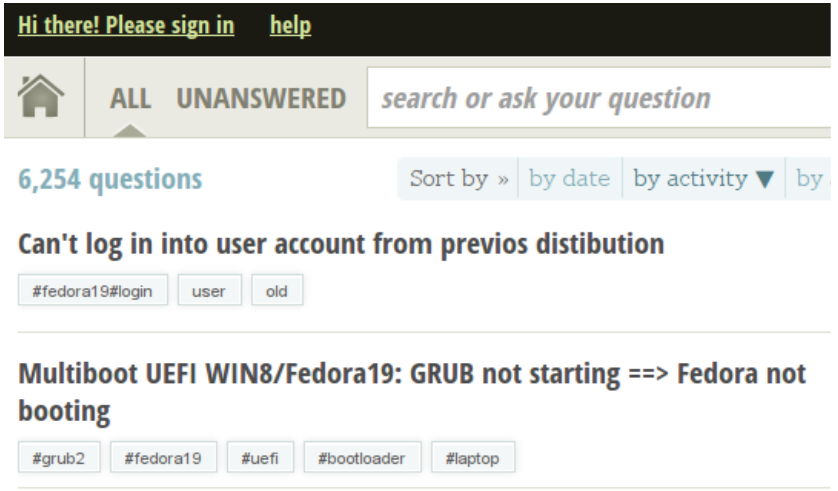
- Can Aire support real web services?
- Can Aire recover from distributed attacks?
- What are the runtime overheads of Aire?

Aire supports real web services



Aire supports real web services

Askbot

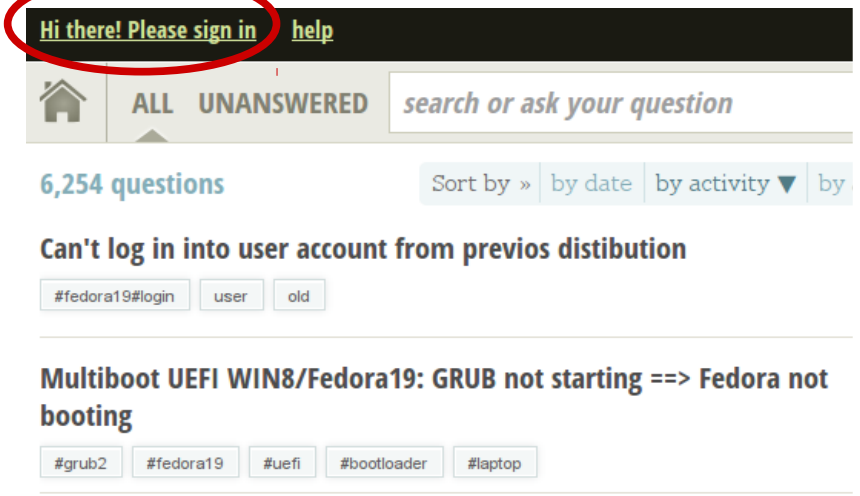


OAuth Provider

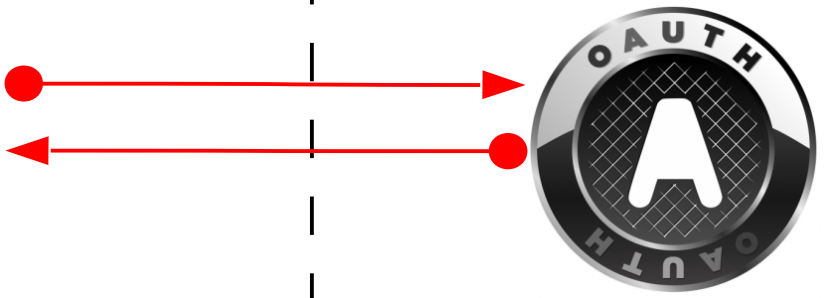
Dpaste

Aire supports real web services

Askbot



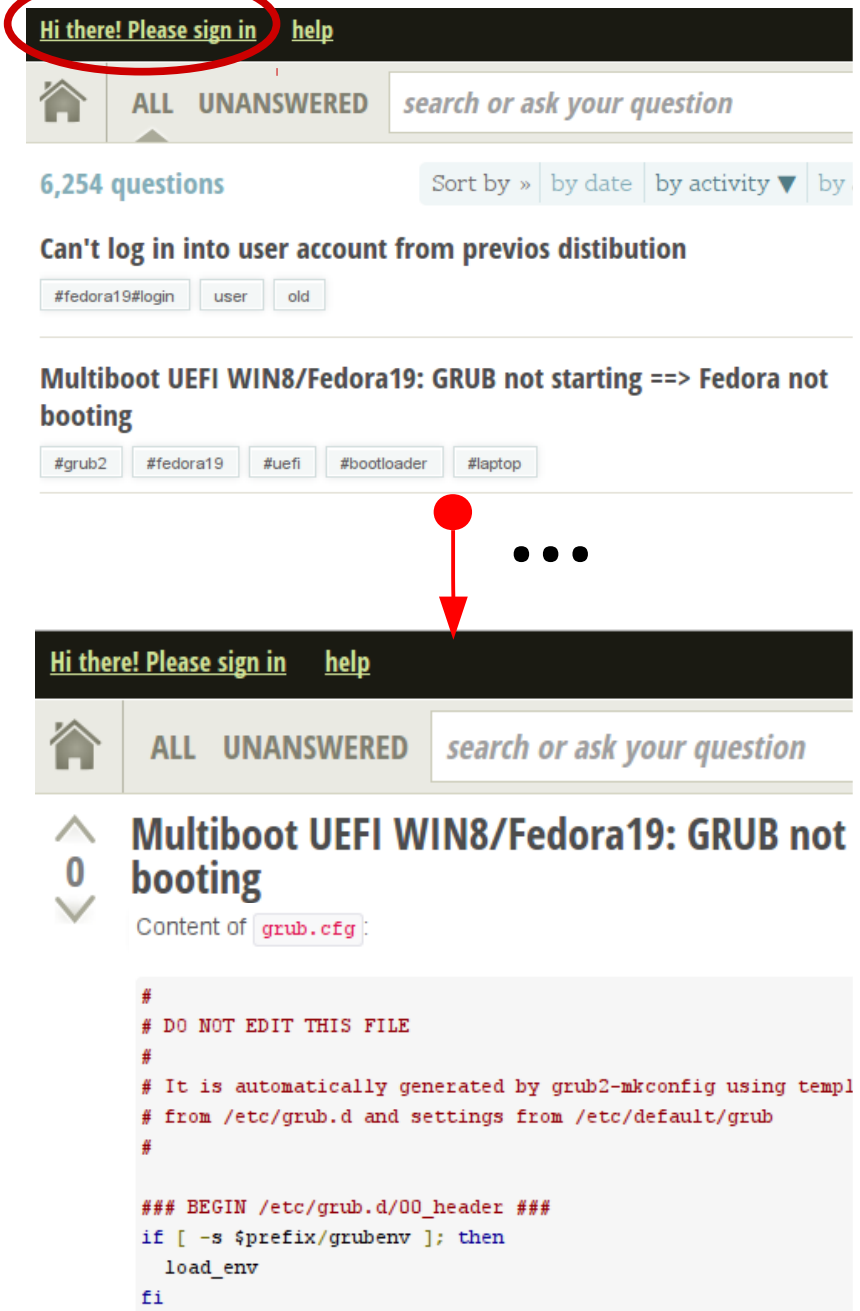
OAuth Provider



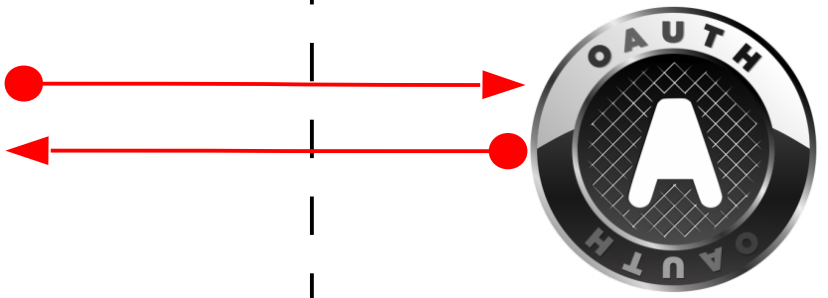
Dpaste

Aire supports real web services

Askbot



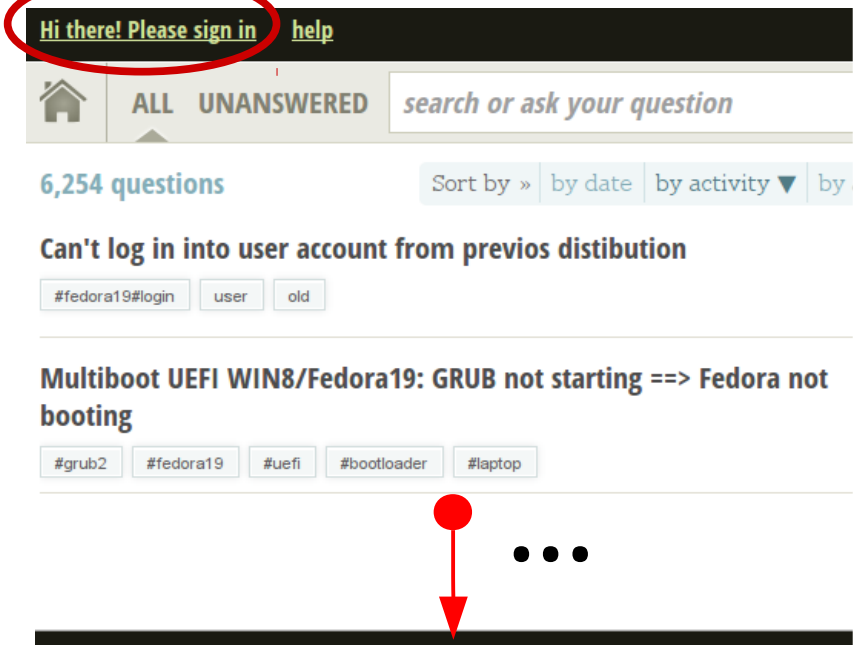
OAuth Provider



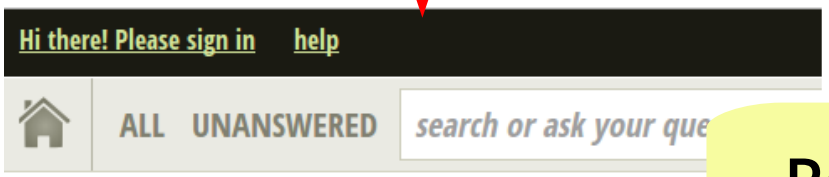
Dpaste

Aire supports real web services

Askbot



...



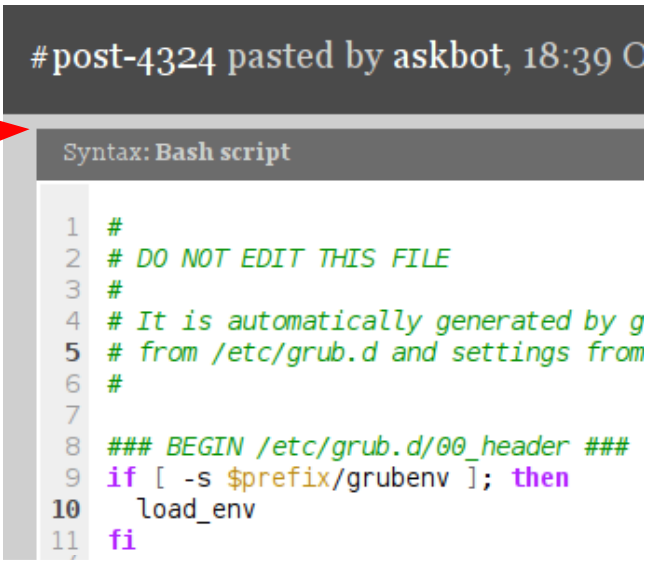

```
#
# DO NOT EDIT THIS FILE
#
# It is automatically g
# from /etc/grub.d and
#

### BEGIN /etc/grub.d/00_header ###
if [ -s $prefix/grubenv ]; then
  load_env
fi
```

OAuth Provider

Dpaste

Post code

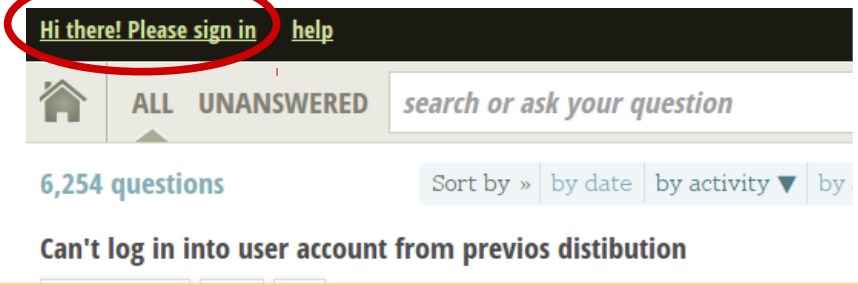


Append a link to Dpaste

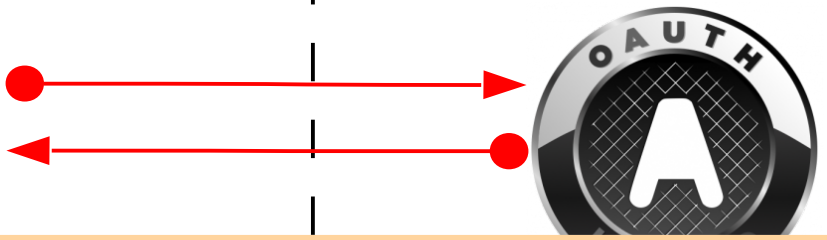
Share link: http://dpaste.com/4324

Aire supports real web services

Askbot



OAuth Provider



Askbot + OAuth + Dpaste
= 183K LoC!

Aire can support large Django web applications

Content of `grub.cfg`:

```
#
# DO NOT EDIT THIS FILE
#
# It is automatically g
# from /etc/grub.d and
#

### BEGIN /etc/grub.d/00_header ###
if [ -s $prefix/grubenv ]; then
  load_env
fi
```

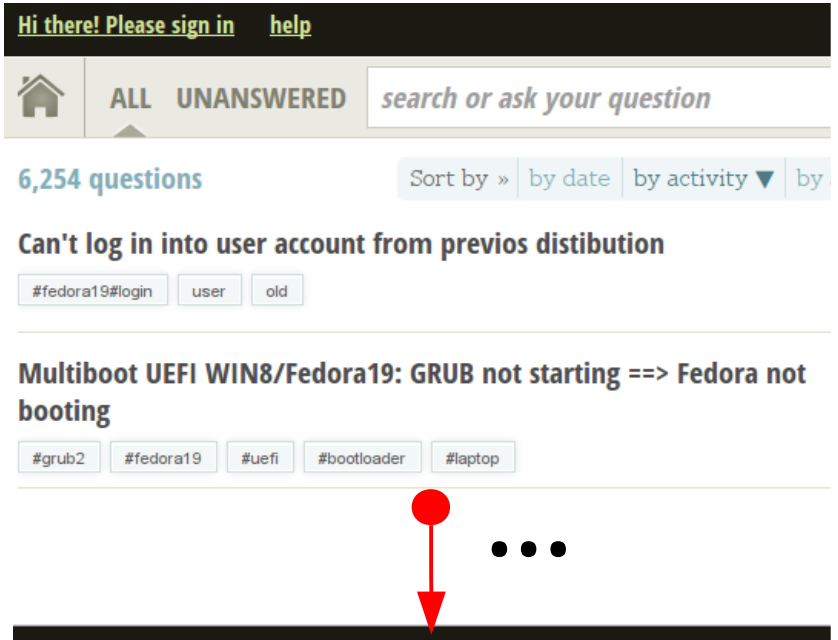
Append a link to Dpaste

Share link: http://dpaste.com/4324

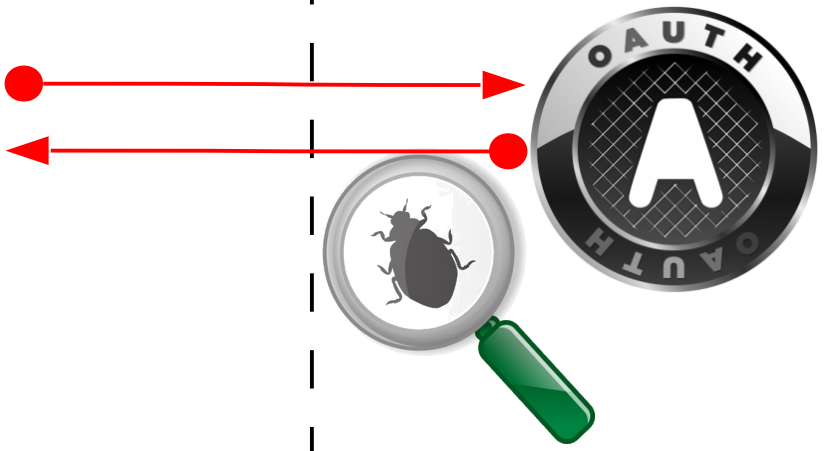
```
2  # DO NOT EDIT THIS FILE
3  #
4  # It is automatically generated by g
5  # from /etc/grub.d and settings from
6  #
7
8  ### BEGIN /etc/grub.d/00_header ###
9  if [ -s $prefix/grubenv ]; then
10   load_env
11 fi
```

Aire enables automatic recovery

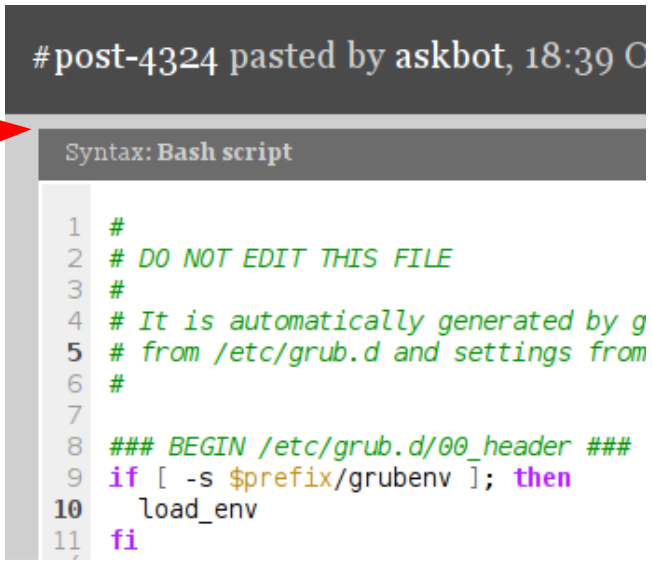
Askbot



OAuth Provider



Dpaste



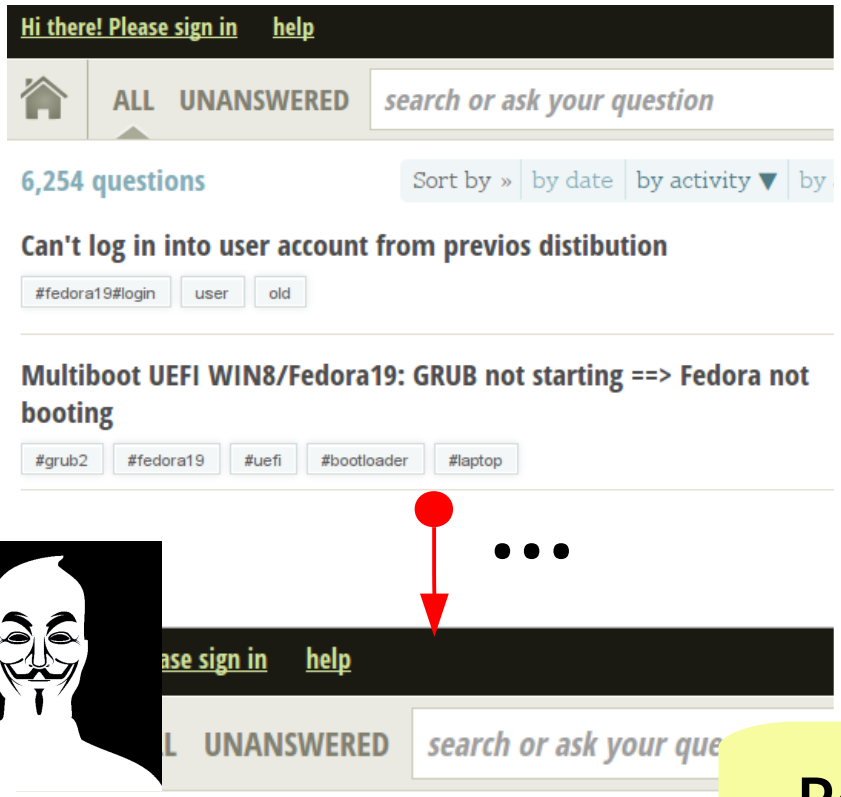
Post code

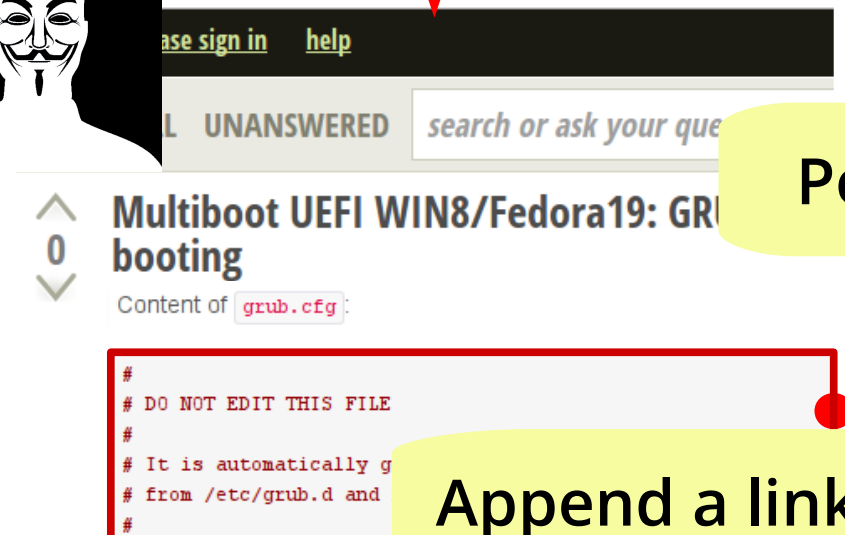
Append a link to Dpaste

Share link: http://dpaste.com/4324

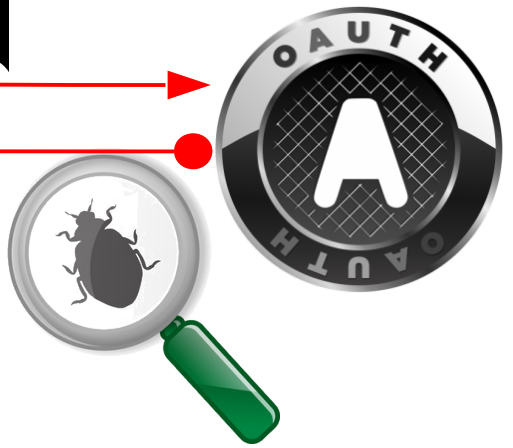
Aire enables automatic recovery

Askbot

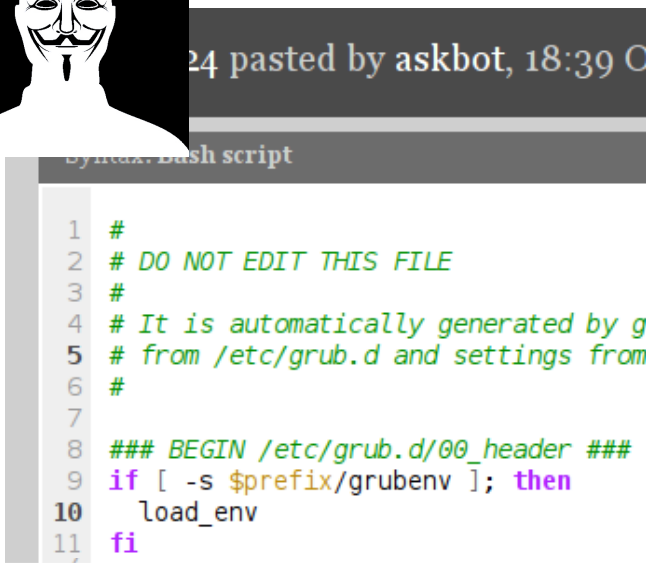




OAuth Provider



Dpaste



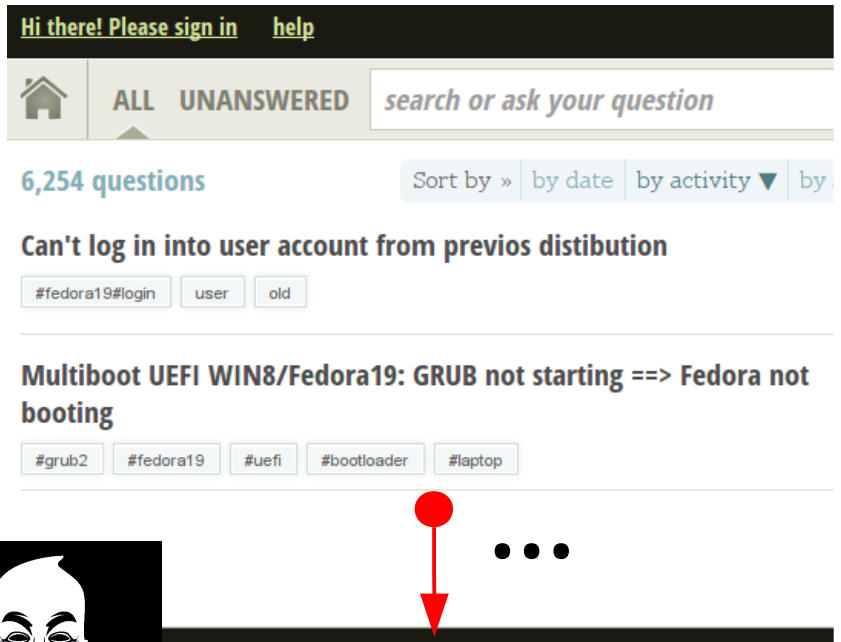
Post code

Append a link to Dpaste

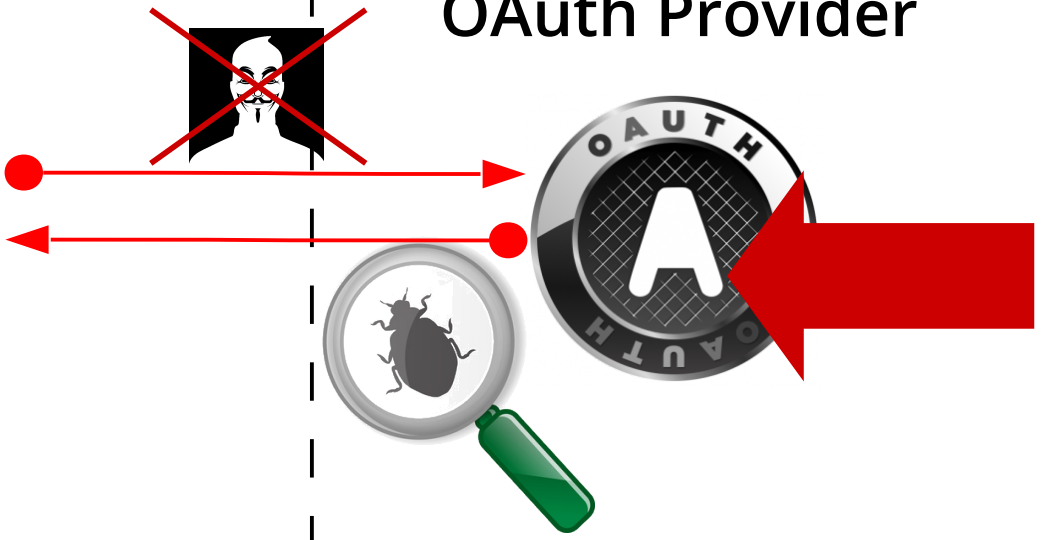
Share link: http://dpaste.com/4324

Aire enables automatic recovery

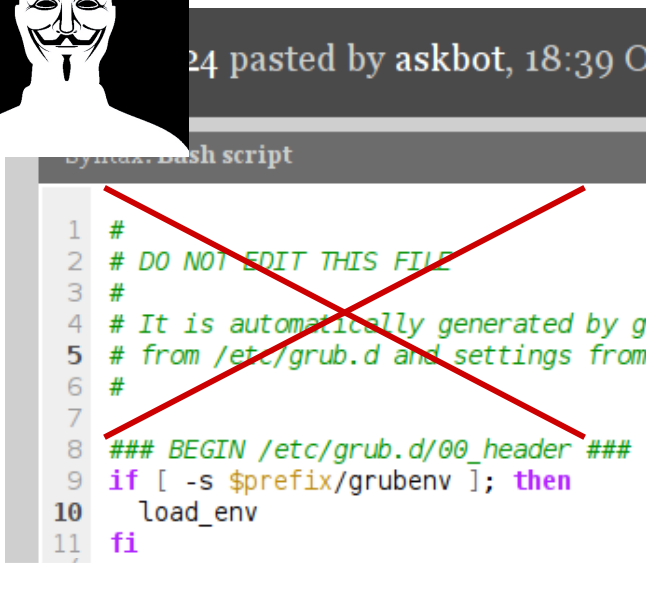
Askbot



OAuth Provider



Dpaste



Post code

Append a link to Dpaste

Share link: http://dpaste.com/4324

Aire enables automatic recovery

- Askbot, OAuth, and Dpaste are correctly recovered
  - Even when Dpaste is temporary unavailable
  - Even when Dpaste goes offline

- **More examples** in paper:
  - Intrusion recovery (synthetic)
  - Mistakes on ACL setting
  - Misconfigured versioning spreadsheet

Aire has moderate runtime overheads

Workload	Req/s without Aire	Req/s with Aire	Logs / req With Aire
Reading	21.58	17.58	5.52 KB
Writing	23.26	16.20	9.24 KB

- 19-30% throughput reduction
- 5-9KB/req storage overheads

→ Moderate overheads for websites which care integrity more than performance

Aire's repair is efficient

	Askbot	OAuth	DPaste
Repaired Reqs	105 / 2196	2 / 9	1 / 496
Remote repair reqs	1	1	0
Local repair time	84.06 sec	0.10 sec	3.91 sec
Normal exec. time	177.58 sec	0.01 sec	0.02 sec

- Experiment setting:
  - Attacker logins as a victim user and writes a post
  - 100 legitimate users post 5 questions and navigate
  - All users are affected by the attack (read attacker's post)

Aire's repair is

Repair in Askbot **propagates** to OAuth and Dpaste

	Askbot	OAuth	DPaste
Repaired Reqs	105 / 2196	2 / 9	1 / 496
Remote repair reqs	1	1	0
Local repair time	84.06 sec	0.10 sec	3.91 sec
Normal exec. time	177.58 sec	0.01 sec	0.02 sec

- Experiment setting:
  - Attacker logins as a victim user and writes a post
  - 100 legitimate users post 5 questions and navigate
  - All users are affected by the attack (read attacker's post)

Aire's repair is efficient

5% of requests are repaired

	Askbot	OAuth	DPaste
Repaired Reqs	105 / 2196	2 / 9	1 / 496
Remote repair reqs	1	1	0
Local repair time	84.06 sec	0.10 sec	3.91 sec
Normal exec. time	177.58 sec	0.01 sec	0.02 sec

- Experiment setting:
  - Attacker logins as a victim user and writes a post
  - 100 legitimate users post 5 questions and navigate
  - All users are affected by the attack (read attacker's post)

Aire's repair is efficient

	Askbot	OAuth	DPaste
Repaired Reqs	105 / 2196	2 / 9	1 / 496
Remote repair reqs	1	1	0
Local repair time	84.06 sec	0.10 sec	3.91 sec
Normal exec. time	177.58 sec	0.01 sec	0.02 sec

Total repair takes **x2 shorter** than normal execution,
although **x10 slower** in replaying
a request for repair

ost

- 100 legitimate users post 5 questions and navigate
- All users are affected by the attack (read attacker's post)

Related work

- Intrusion recovery with selective re-execution:
  - **Retro** [OSDI'10], **Warp** [SOSP'11]

  → Use them as building blocks for **asynchronous** repair

- Intrusion recovery in distributed systems:
  - **Heat-ray** [SOSP'09], **Polygraph** [EuroSys'09], **Dare** [APsys'12]

  → Automatic recovery in **loosely coupled web services**

Summary

- **Aire** recovers integrity of distributed web services
  - Define a **repair protocol**
  - Support **asynchronous** and **decentralized** repair
  - Propose partial repair consistency