

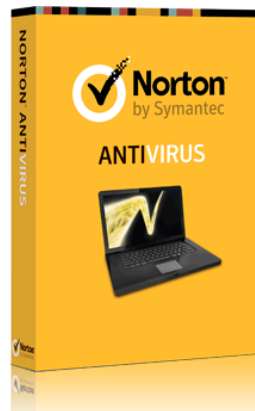
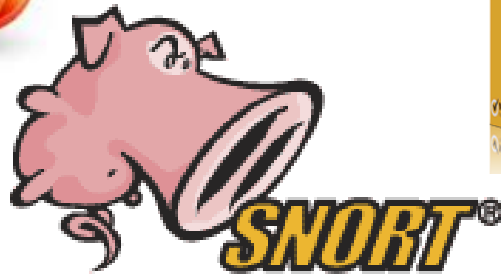
Automatic intrusion recovery with system-wide history

Taesoo Kim

MIT CSAIL

Current focus of system security: preventing attacks

- System hardening tools/techniques
 - (e.g.,) Firewall, AntiVirus ...



My work on preventing attacks (proactive security)

- **StealthMem** [Security '12]
- **Morula** [Oakland '14]
- **UserFS** [Security '10]
- **Mbox** [ATC '13]
- **VMsec** [APSys '13]

My work on preventing attacks (proactive security)

- **StealthMem** [Security '12]

Cloud
(HyperV)

- **Morula** [Oakland '14]

Mobile
(Android)

- **UserFS** [Security '10]

- **Mbox** [ATC '13]

- **VMsec** [APSys '13]

My work on preventing attacks (proactive security)

- **StealthMem** [Security '12]

Cloud
(HyperV)

- **Morula** [Oakland '14]

Mobile
(Android)

- **UserFS** [Security '10]

- **Mbox** [ATC '13]

- **VMsec** [APSys '13]

Linux

Attackers routinely compromise computer systems

Here's how Google Palestine was hacked; local root server confirms repair

by *Nina Curley*, August 27, 2013

As other outlets are reporting, Google.ps, Google's landing page was hacked yesterday, to protest the naming conventions on Google Maps. Four hackers named Cold z3ro, Ham13t, Sas, and Dr@g took responsibility, naming themselves as members of Hacktech, a Palestinian website that features tech news and hacker forums.

بالعربي ع

Comment 0



uncle google we say hi from palestine to remember you that the country in google map not called israel. its called Palestine

Attackers routinely compromise computer systems

Here's how Google Palestine was hacked; local root server confirms repair

by [Nina Curley](#), August 27, 2013

As other outlets are reporting, Google.ps, Google's landing page was hacked yesterday, to protest the naming conventions on Google Maps. Four hackers named Cold z3ro, Ham13t, Sas, and Dr@g took responsibility, naming themselves as members of Hackteach, a Palestinian website that features news and hacker forums.



uncle google we say hi from palestine to remember you that the country in google map not called israel. its called Palestine

Facebook remote code execution bug nets researcher \$33,500

Summary: The social network's payout represents the largest bug bounty it has ever rewarded a researcher with.



By [Michael Lee](#) | January 23, 2014 -- 03:41 GMT (19:41 PST)

Facebook has paid out its largest bug bounty ever of \$33,500 to a security researcher who could have potentially taken full control of a server within its network.

Since 2012, Brazilian computer engineer Reginaldo Silva has been toying with vulnerabilities in OpenID, the open technology that allows users to use an account with an existing identity provider to sign in to other compatible services. For example, a user can trust Symantec's Personal Identity Portal to create an OpenID account, then use that one account to sign in to WordPress.

In the event that users forget their passwords, Facebook itself can use an OpenID provider to verify the identity of the user. As part of the communication process, Facebook communicates with the provider, receiving an XML document and parsing it to verify that it is indeed the correct provider.

Attackers routinely compromise computer systems

Here's how Google Palestine was hacked; local root server confirms repair

by [Nina Curley](#), August 27, 2013

As other outlets are reporting, Google.ps, Google's landing page was hacked yesterday, to protest the naming conventions on Google Maps. Four hackers named Cold z3ro, Ham13t, Sas, and Dr@g took responsibility, naming themselves as members of Hackteach, a Palestinian website that features news and hacker forums.



uncle google we say hi from palestine to remember you that the country is called Palestine

Facebook remote code execution bug nets researcher \$33,500

Summary: The social network's payout represents the largest bug bounty it has ever rewarded a researcher with.



By [Michael Lee](#) | January 23, 2014 -- 03:41 GMT (19:41 PST)

Facebook has paid out its largest bug bounty ever of \$33,500 to a security researcher who could have potentially taken full control of a server within its network.

Since 2012, Brazilian computer engineer Beninaldo Silva has been toying with vulnerabilities in OpenID, a protocol used by many websites to sign in to other services.

GitHub hacked, millions of projects at risk of being modified or deleted

By Sebastian Anthony on March 5, 2012 at 7:22 am

21 Comments



GitHub, one of the largest repositories of commercial and open source software on the web, has been hacked. Over the weekend, developer Egor Homakov exploited a gaping vulnerability in GitHub that allowed him (or anyone else with basic hacker know-how) to gain administrator access to projects such as Ruby on Rails, Linux, and millions of others. Homakov could've deleted the entire history of projects such as jQuery, Node.js, Reddit, and Redis.

Share This Article

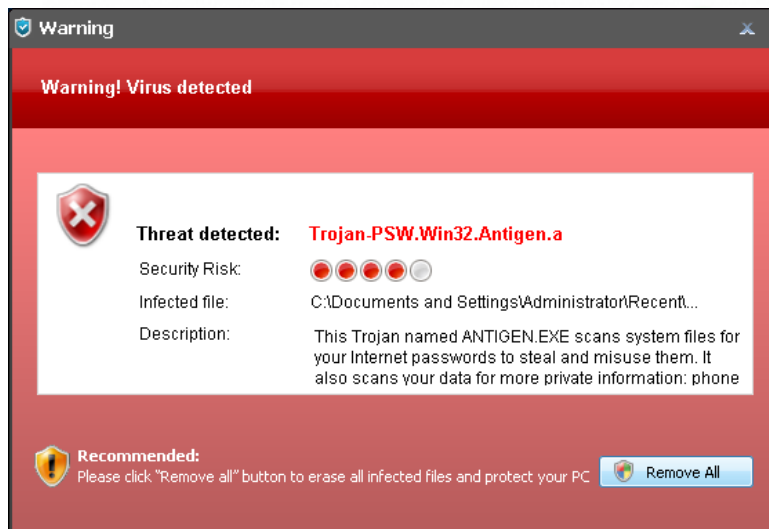
Attackers routinely compromise computer systems

Here's how Google Palestine was hacked; local root server confirms repair

by [Nina Curley](#), August 27, 2013

As other outlets are reporting, Google.ps, Google's landing page was hacked yesterday, to protest the naming conventions on Google Maps. Four hackers named Cold z3ro, Ham13t, Sas, and Dr@g took responsibility, naming themselves as members of Hacktech, a Palestinian website that features news and hacker forums.

Google Owned
No News Is a Good News
Cold z3ro - Ham13t - Sas - Dr@g
From Palestine: We are the Best of the Rest



Facebook remote code execution bug nets researchers

Summary: The social networked a researcher with.



By Michael Lee | Jan

Facebook has paid out its largest potentially taken full control of a

Since 2012, Brazilian computer



GitHub hacked, millions of projects at risk of being modified or deleted

By Sebastian Anthony on March 5, 2012 at 7:22 am

21 Comments



Share This Article



Compromises inevitable

- Programmers write buggy code
 - A single bug can lead to system compromises
- Admins mis-configure policies
- Users choose weak, guessable passwords

Compromises inevitable

Need both *proactive security* mechanism
and *reactive recovery* mechanism!

Recovering integrity is required to
continue operating!

Existing recovery tools are limited

- Anti-virus tools
 - Only repair from predictable attacks
- Backup tools
 - Attack may be detected days or weeks later
 - Restoring from backup discards *all* changes

Existing recovery tools are limited

- Anti-virus tools
 - Only repair from predictable attacks


Admins spend days or weeks *manually* tracking down all effects of the attack with *no guarantee* that everything is cleaned up!

Example: kernel.org

- A main repository of code for the Linux kernel
 - Also host open source projects like Git and Android


The Linux Kernel Archives

Welcome to the Linux Kernel Archives. This is the primary site for the Linux kernel source, but it has much more than just Linux kernels.
[Frequently Asked Questions](#)



Protocol	Location
HTTP	http://www.kernel.org/pub/
FTP	ftp://ftp.kernel.org/pub/
RSYNC	rsync://rsync.kernel.org/pub/

Latest Stable Kernel:

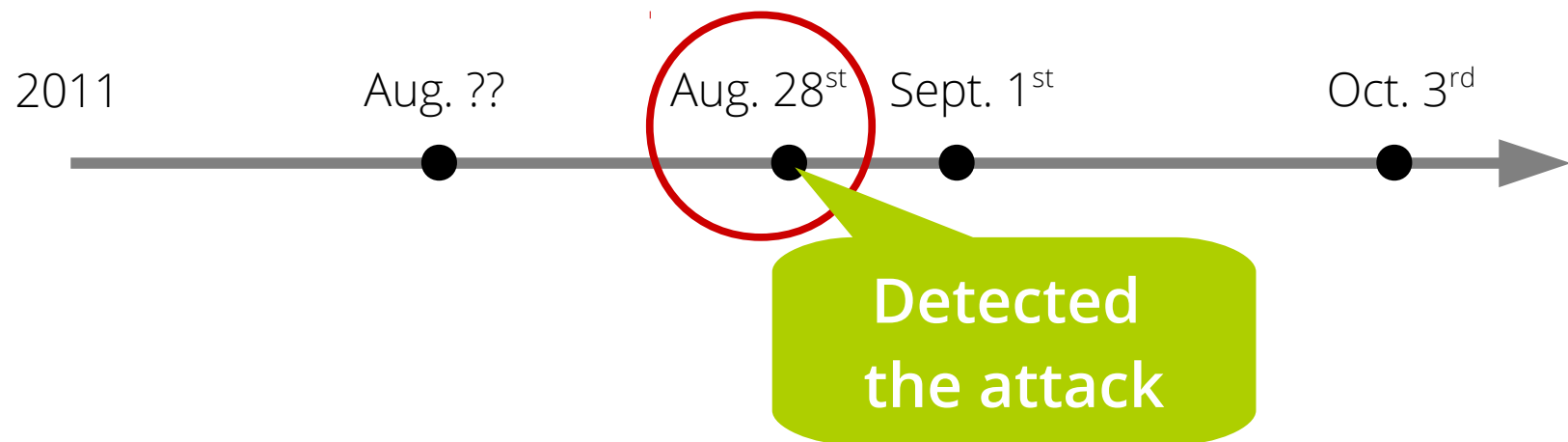
[3.0.3](#)

linux-next:	next-20110826	2011-08-26	[Patch]	[View Patch]	[Gitweb]			
linux-next:	next-20110826	2011-08-26	[Patch]	[View Patch]	[Gitweb]			
snapshot:	3.1-rc3-git6	2011-08-27	[Patch]	[View Patch]				
mainline:	3.1-rc3	2011-08-22	[Full Source]	[Patch]	[View Patch]	[Gitweb]	[Changelog]	
stable:	3.0.3	2011-08-17	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
stable:	2.6.39.4	2011-08-03	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
stable:	2.6.38.8	2011-06-03	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
stable:	2.6.37.6	2011-03-27	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
longterm:	2.6.35.14	2011-08-01	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
longterm:	2.6.34.10	2011-06-26	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
longterm:	2.6.33.18	2011-08-16	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
longterm:	2.6.32.45	2011-08-16	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
longterm:	2.6.27.59	2011-04-30	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]

Changelogs are provided by the kernel authors directly. Please don't write the webmaster about them.
[Customize the patch viewer](#)

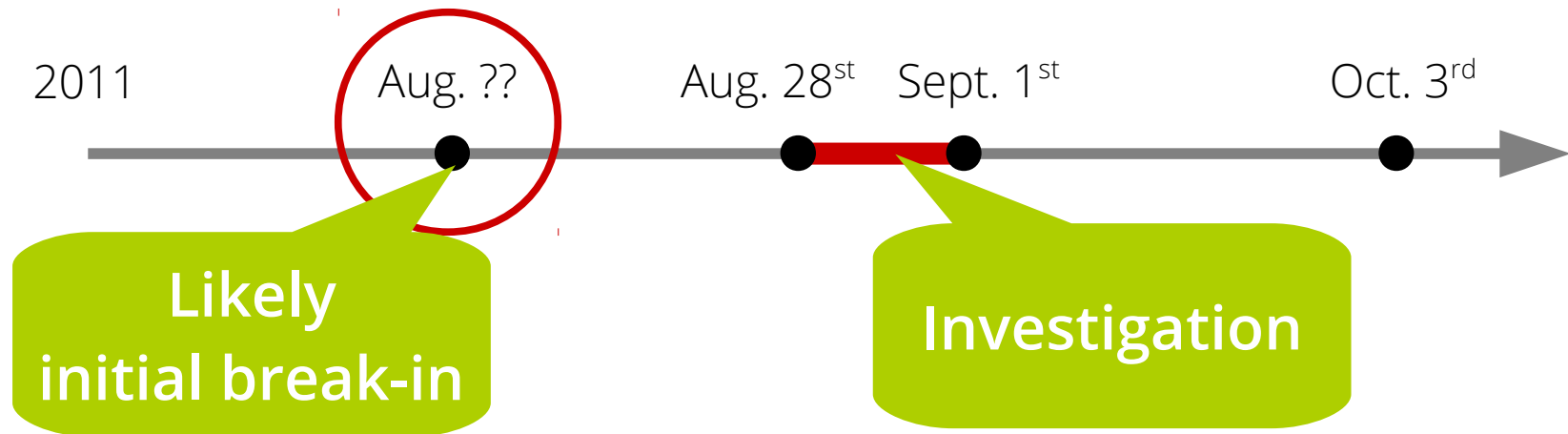
Example: kernel.org attack

- Detected that kernel.org had been compromised
 - Noticed error messages from a program that administrators never installed themselves



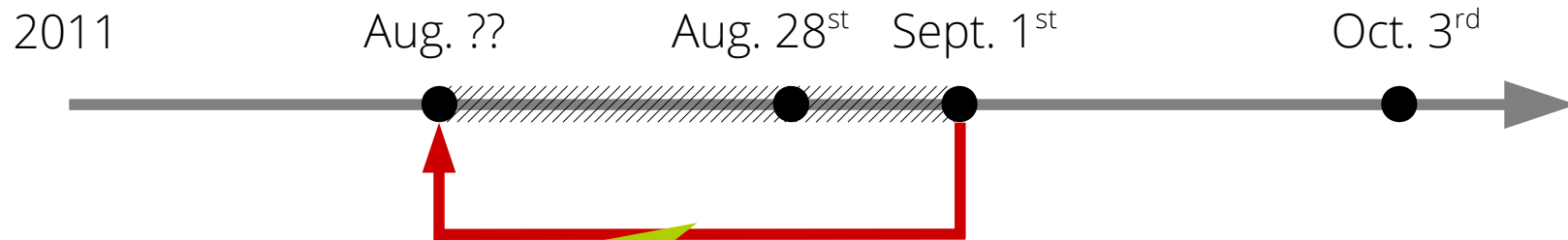
Example: kernel.org attack

- Investigated the attack for three days
 - The initial break-in likely happened a month ago (Trojaned SSHD was modified around that time)



Example: kernel.org attack

- Fully re-installed all servers with the latest backup
 - Rollback is only safe option (too many suspects to clean up)
 - Took a month for security experts to fully recover



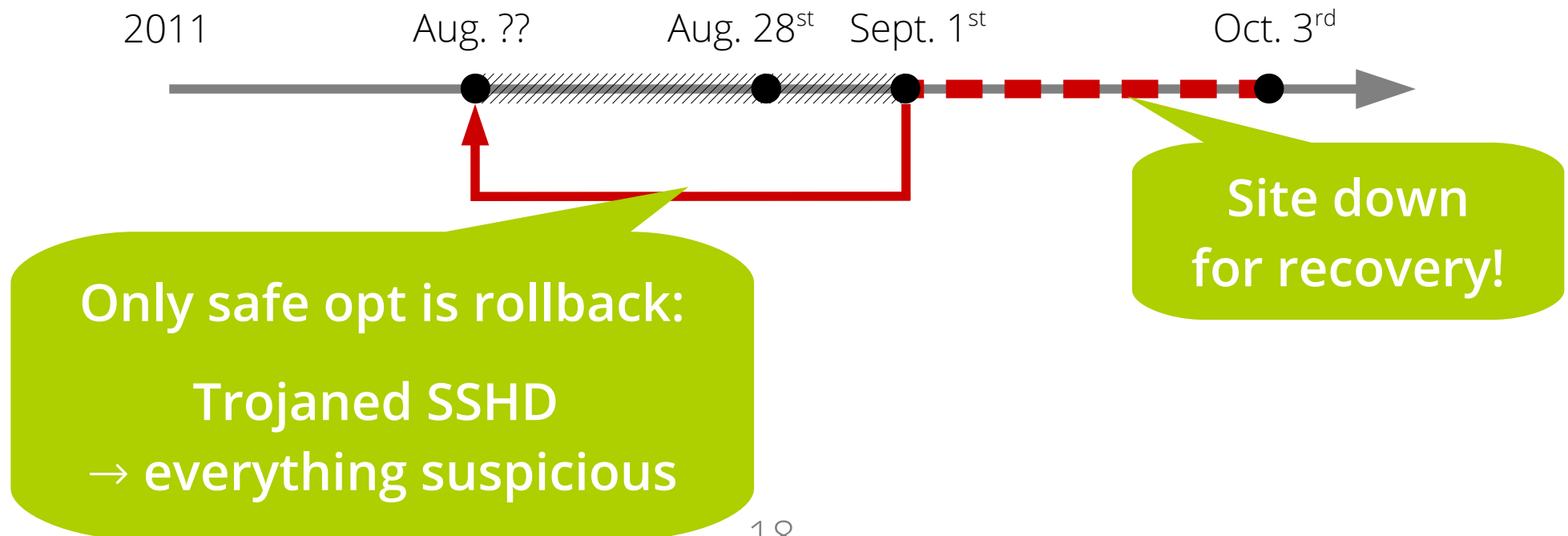
Only safe opt is rollback:

Trojaned SSHD

→ everything suspicious

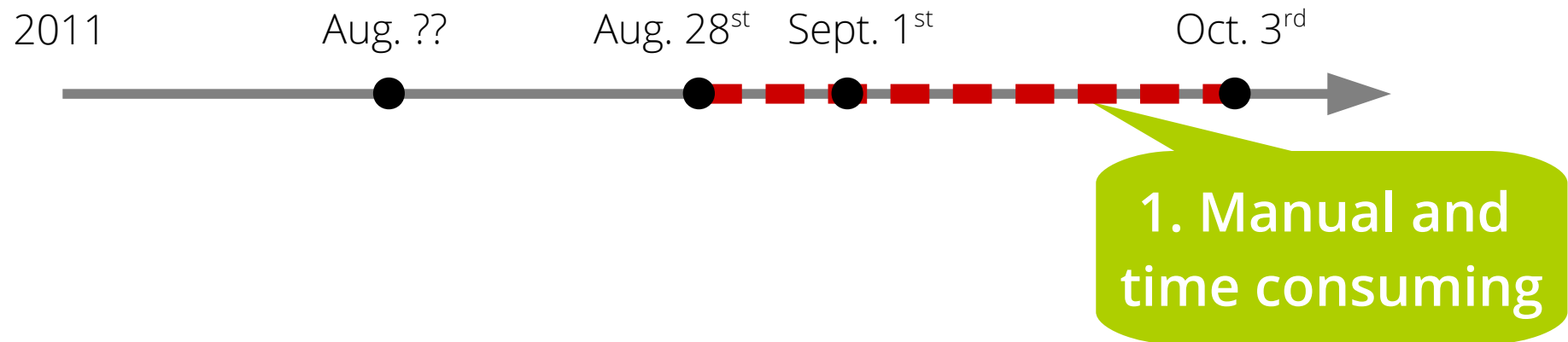
Example: kernel.org attack

- Fully re-installed all servers with the latest backup
 - Rollback is only safe option (too many suspects to clean up)
 - Took a month for security experts to fully recover



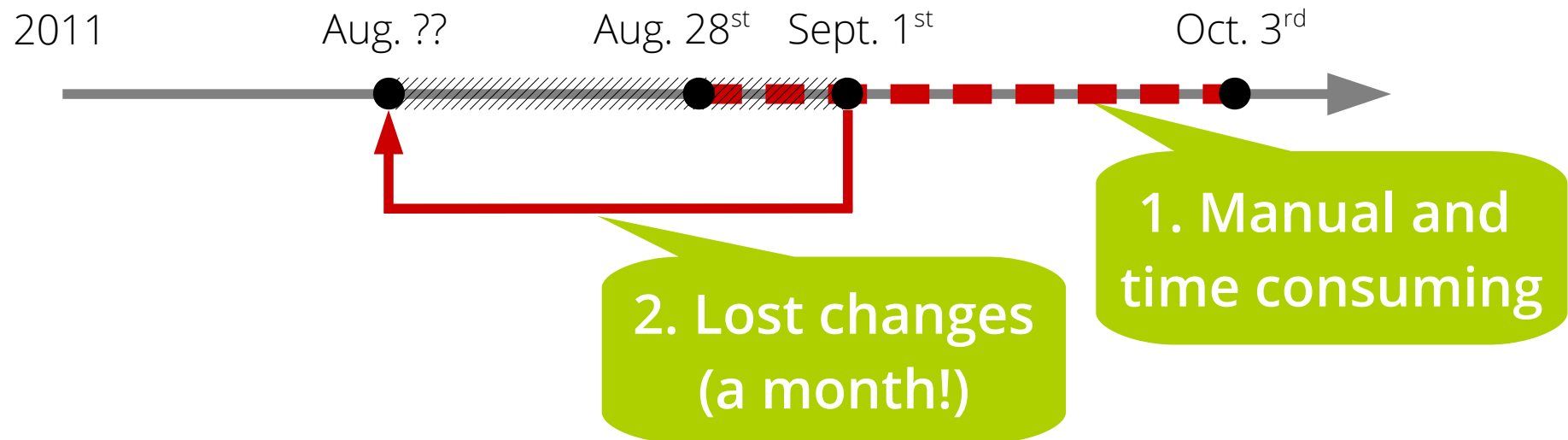
Problems in today's repair strategies

- **Manual** analysis & recovery is **time consuming**



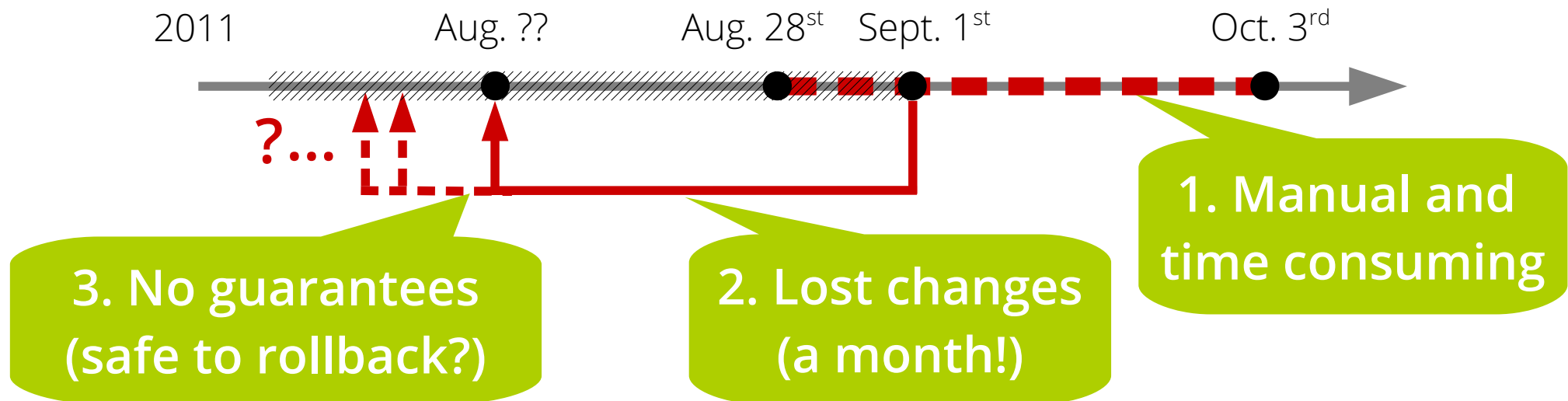
Problems in today's repair strategies

- **Manual** analysis & recovery is **time consuming**
- Rollback ends up **losing changes**



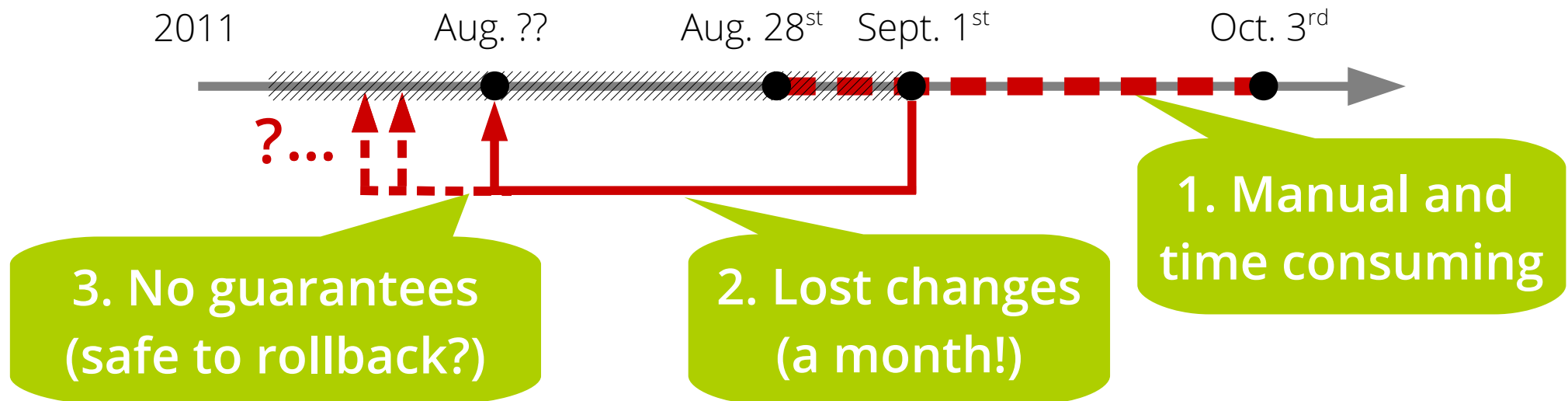
Problems in today's repair strategies

- **Manual** analysis & recovery is **time consuming**
- Rollback ends up **losing changes**
- **No guarantees** of complete removal of attack



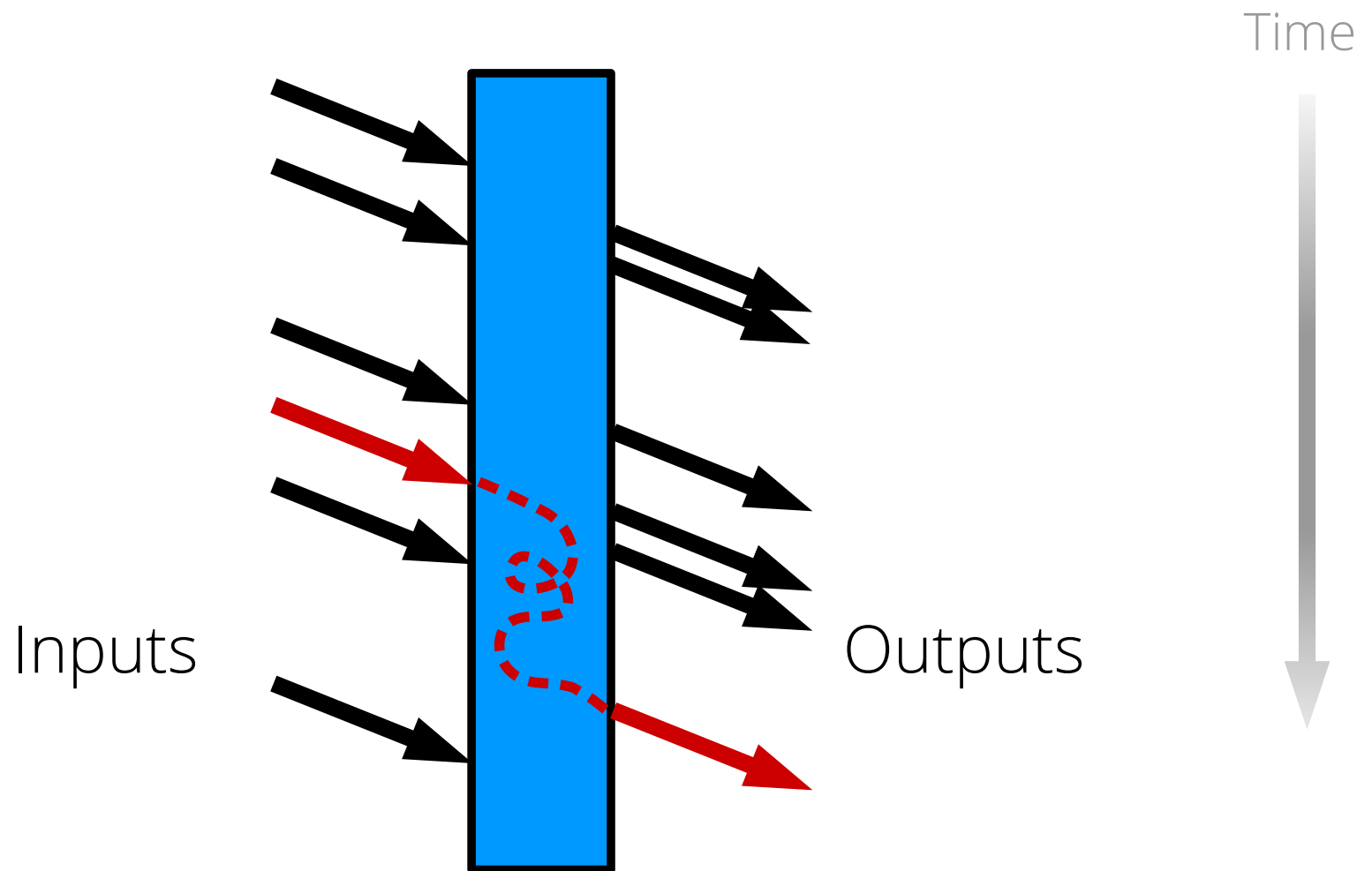
Problems in today's repair strategies

How can we design *automate* recovery system *that preserves* legitimate changes and *provides guarantees*?



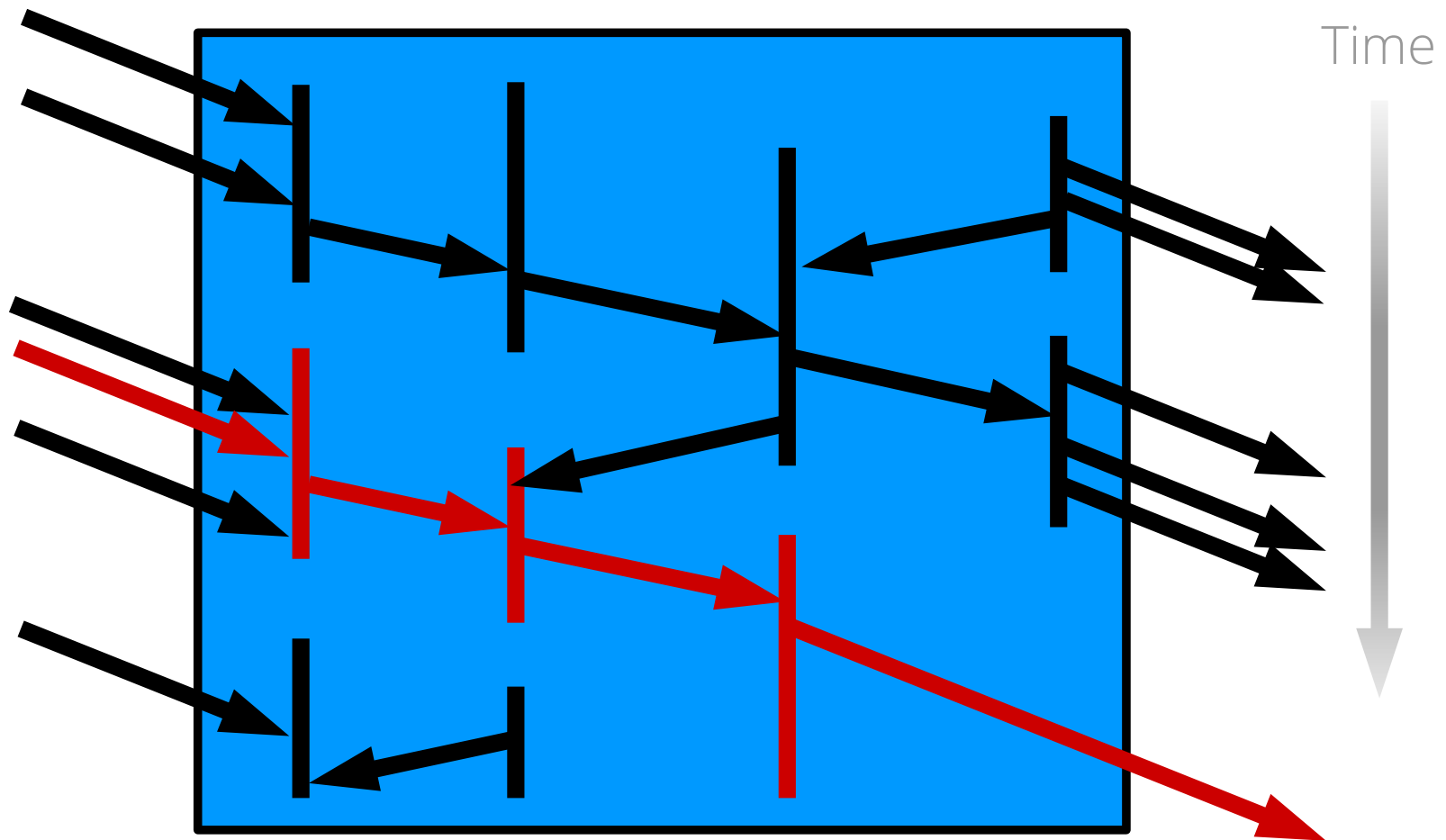
Idea: keep *complete* history of computations

- Inputs/outputs on time-line



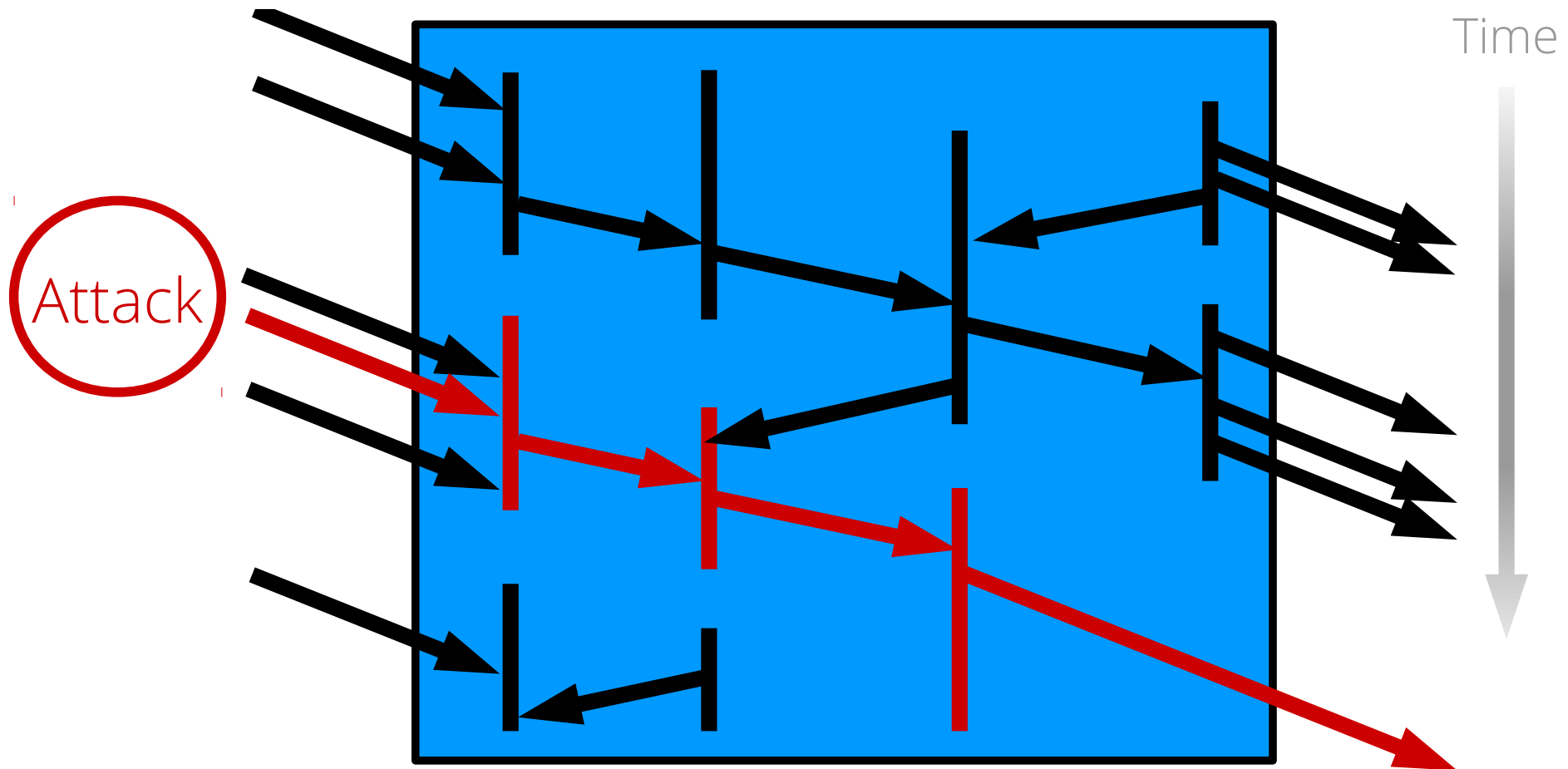
Idea: keep *complete* history of computations

- Represent computer in fine-grained details



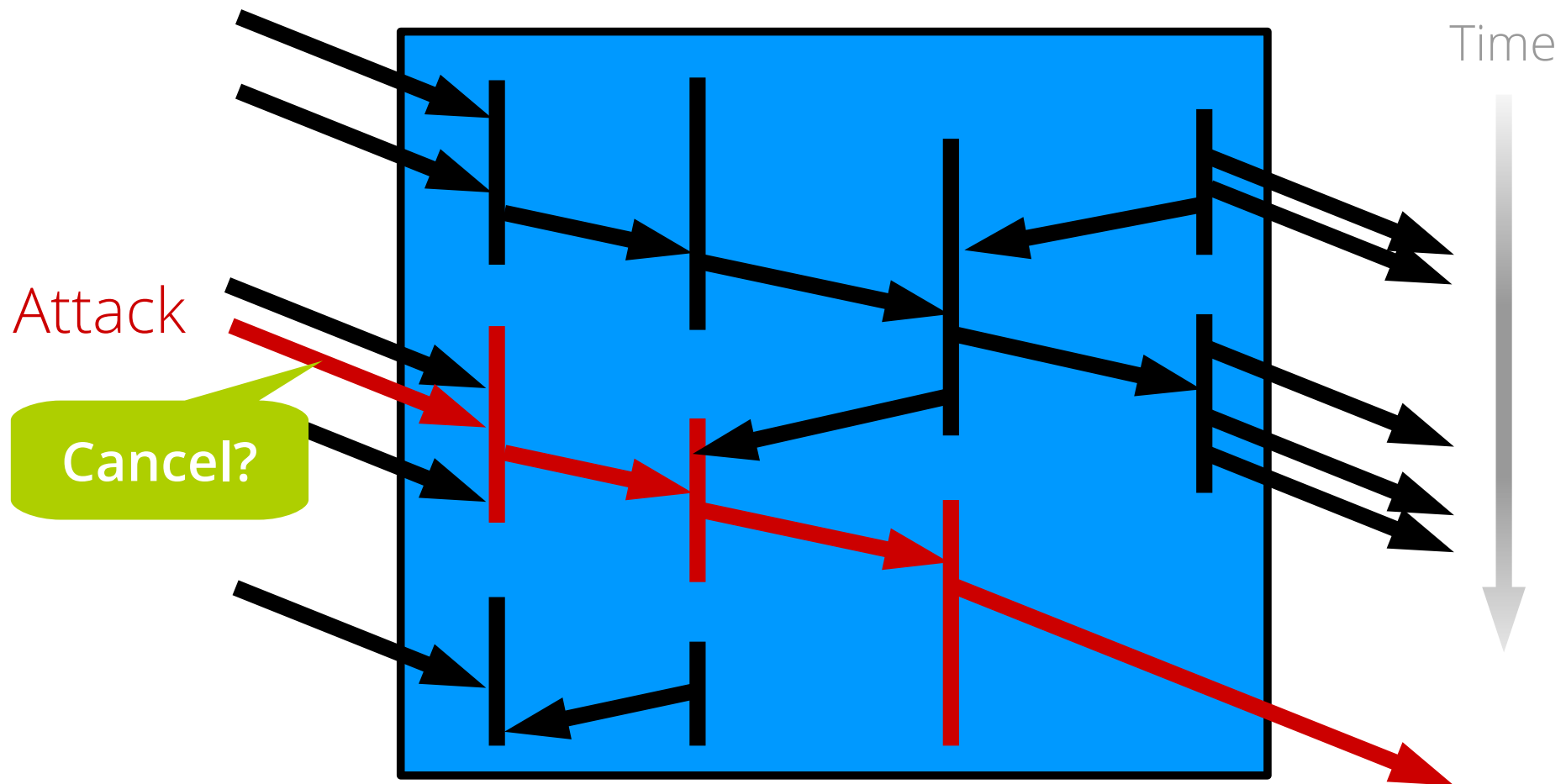
Idea: keep *complete* history of computations

New opportunities to track down attacks!



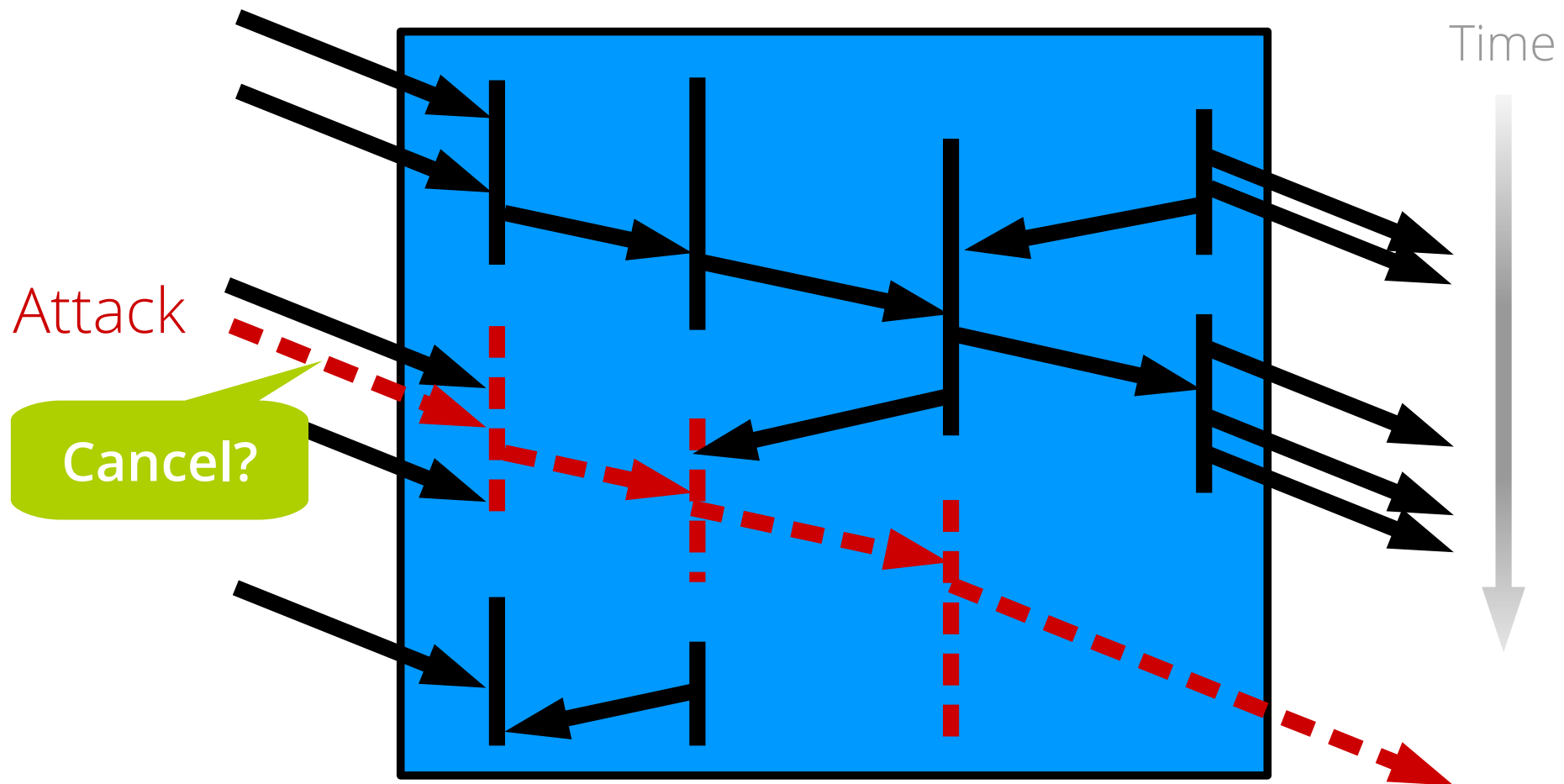
Approach: change our past with history of computations

- Recovery → cancel the initial attack input



Approach: change our past with history of computations

- Recovery → cancel the initial attack input
- Reconstruct states as if attack never happened!

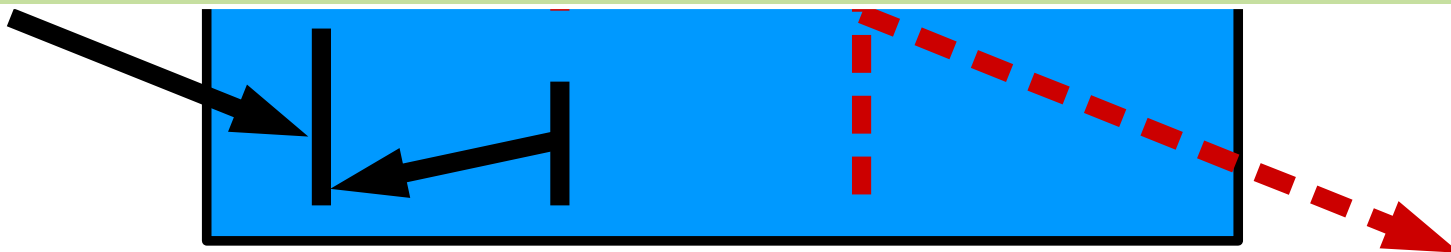


Approach: change our past with history of computations

- Recovery → cancel the initial attack input
- Reconstruct states as if attack never happened!



Turn problem of manual recovery into problem of manipulating history!



Challenges in real systems

- **Existing systems are not designed for history**
 - Implicit dependencies and time-line
- **Attacks can be anywhere in the history**
 - Attacks are often detected days or weeks later
- **History can not be changed in some cases**
 - External dependencies: spam sent out

Contribution:

built real-world systems

- **Automatic recovery**
 - Operating system: **Retro** [OSDI'10]
 - Web application: **Warp** [SOSP'11]
 - Distributed web services: **Aire** [SOSP'13]
- **Automatic detection of attacks**
 - Web application: **Poirot** [OSDI'12]

Today's talk

- **Automatic recovery**
 - Operating system: **Retro** [OSDI'10]
 - Web application: **Warp** [SOSP'11]
 - Distributed web services: **Aire** [SOSP'13]
- **Automatic detection of attacks**
 - Web application: **Poirot** [OSDI'12]
- **Future research agenda**

Today's talk

- **Automatic recovery**
 - Operating system: **Retro** [OSDI'10]
 - Web application: **Warp** [SOSP'11]
 - Distributed web services: **Aire** [SOSP'13]
- **Automatic detection of attacks**
 - Web application: **Poirot** [OSDI'12]
- **Future research agenda**

Example attack scenario



Attacker



Admin



Alice

Example attack scenario



Attacker

- Adds new account for himself
(→ modifies `/etc/passwd`)
- Installs trojaned `pdflatex`



Admin



Alice

Example attack scenario



Attacker

- Adds new account for himself
(→ modifies `/etc/passwd`)
- Installs trojaned `pdflatex`

- Adds new account for Alice
(→ modifies `/etc/passwd`)



Admin



Alice

Example attack scenario



Attacker

- Adds new account for himself
(→ modifies `/etc/passwd`)
- Installs trojaned `pdflatex`



Admin

- Adds new account for Alice
(→ modifies `/etc/passwd`)

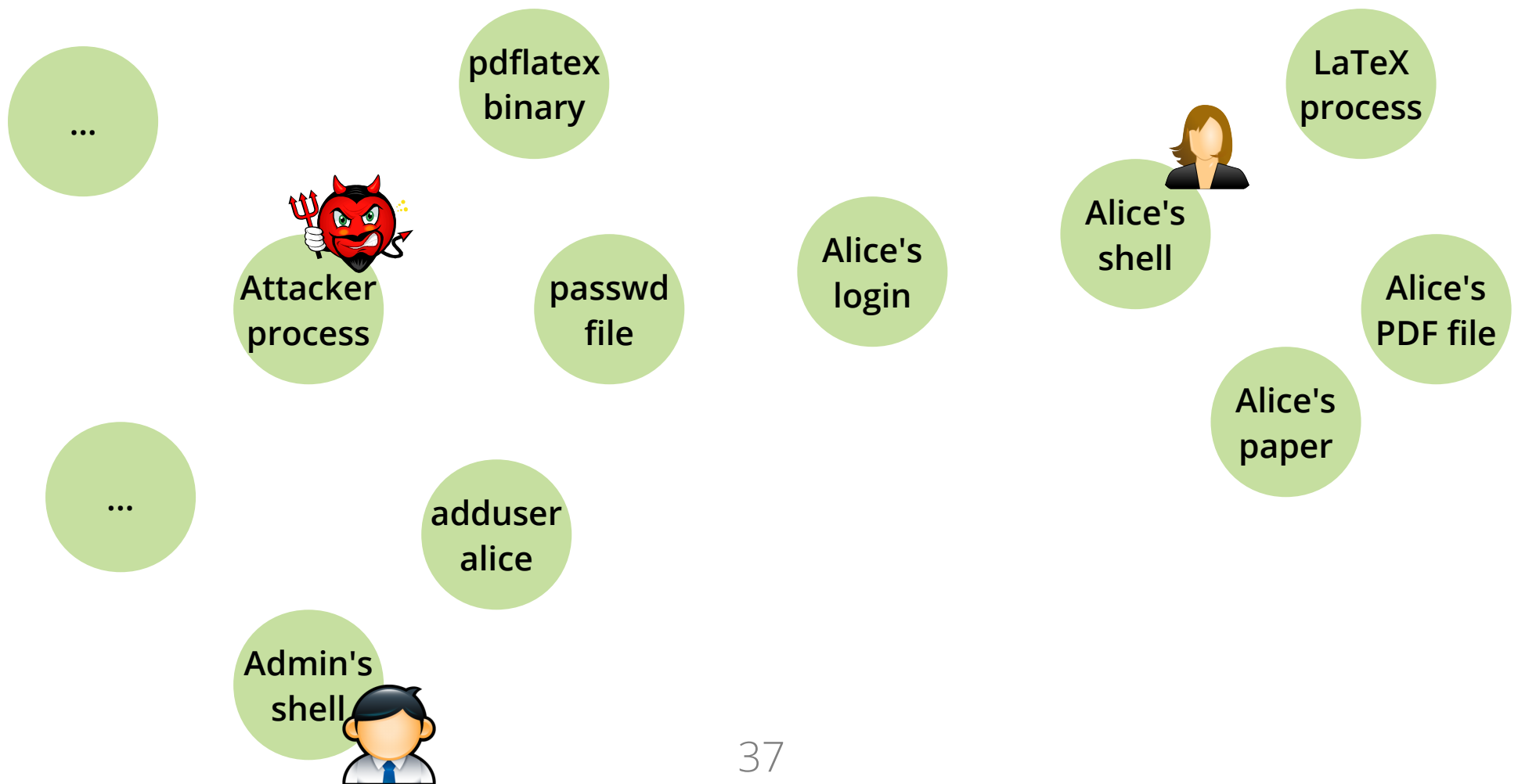


Alice

- Logs in via SSH
(→ SSHD reads `/etc/passwd`)
- Runs trojaned `pdflatex`

History strawman 1:

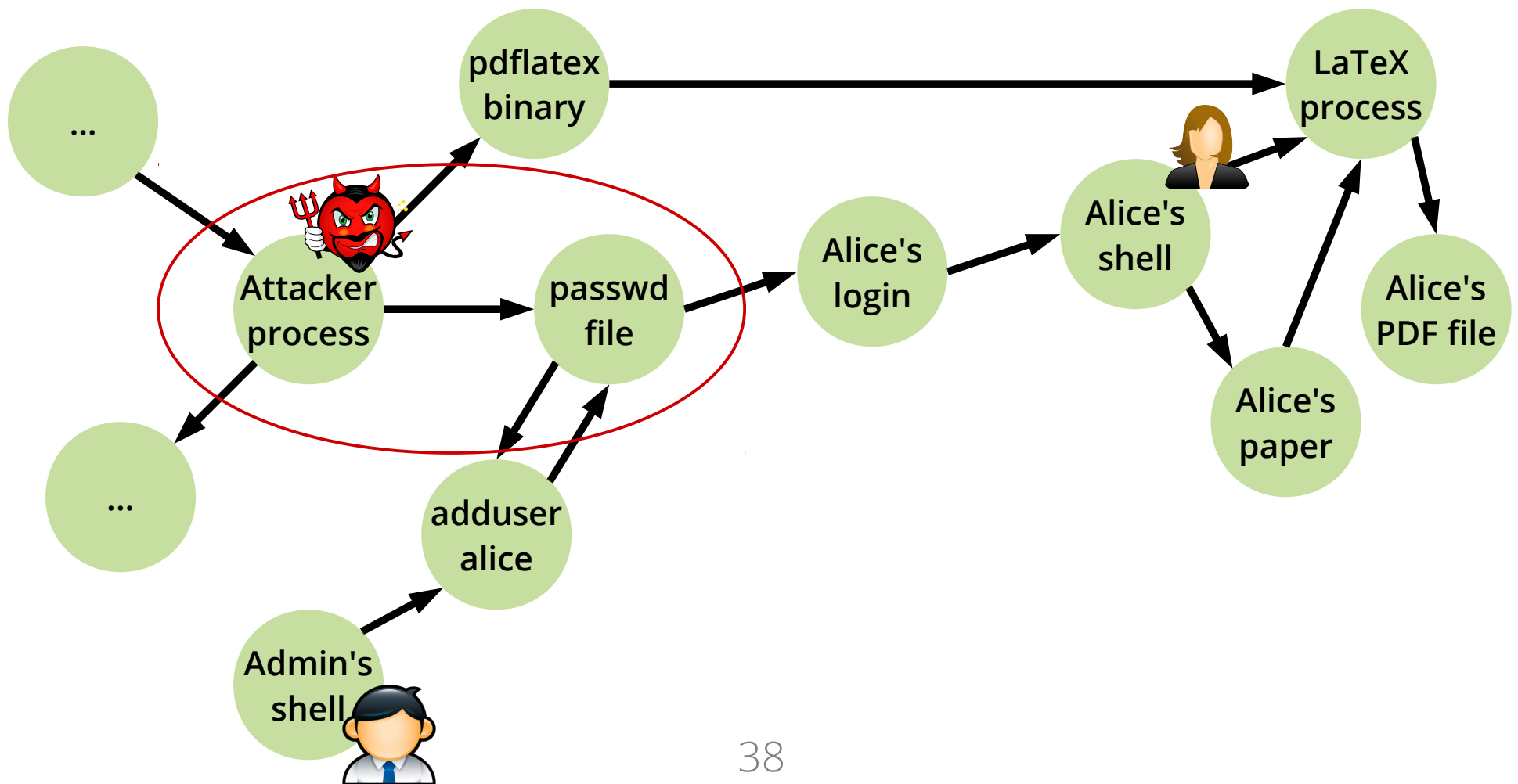
Taint tracking



History strawman 1:

Taint tracking

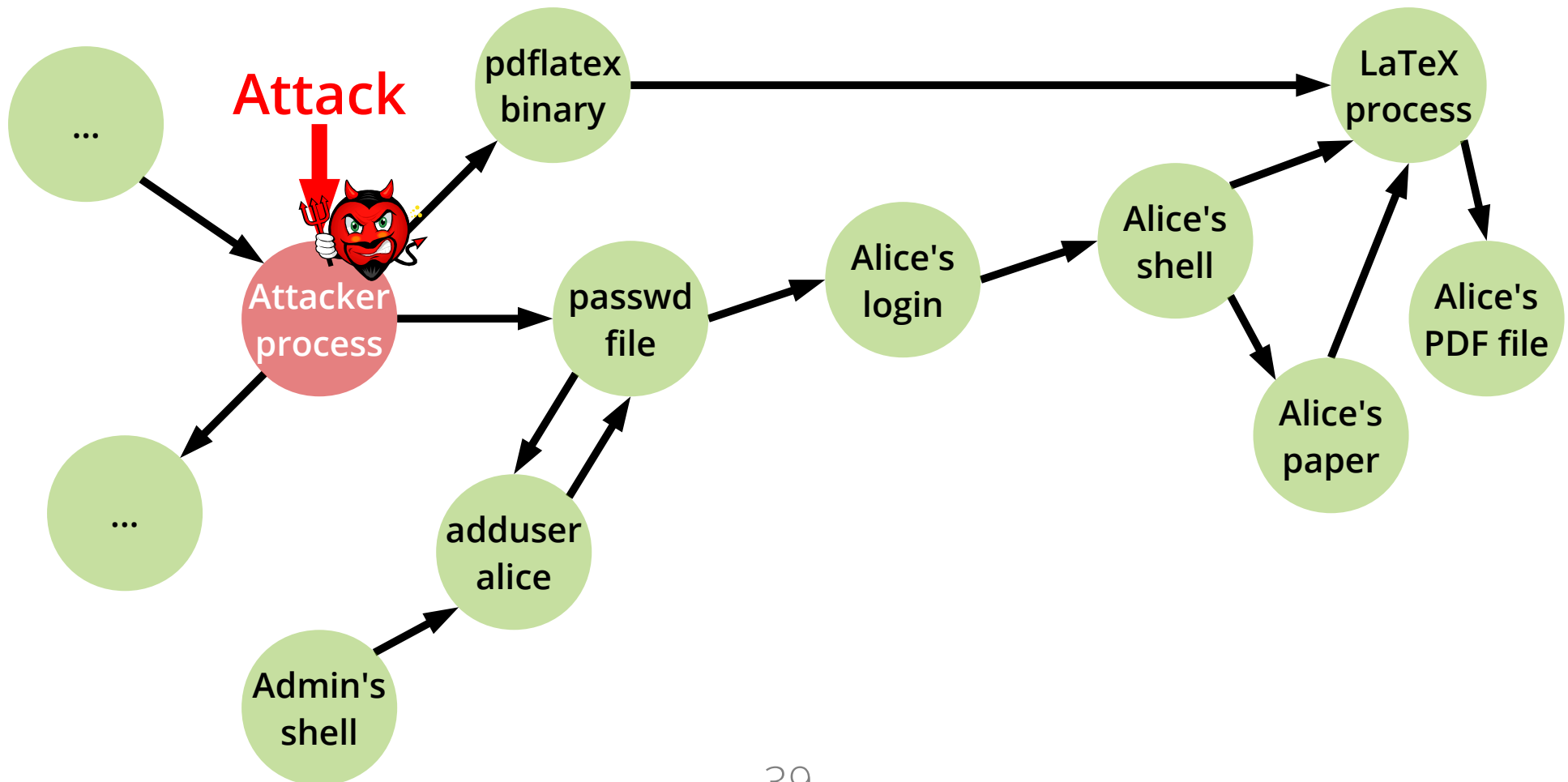
- Track dependencies between processes & files



History strawman 1:

Taint tracking

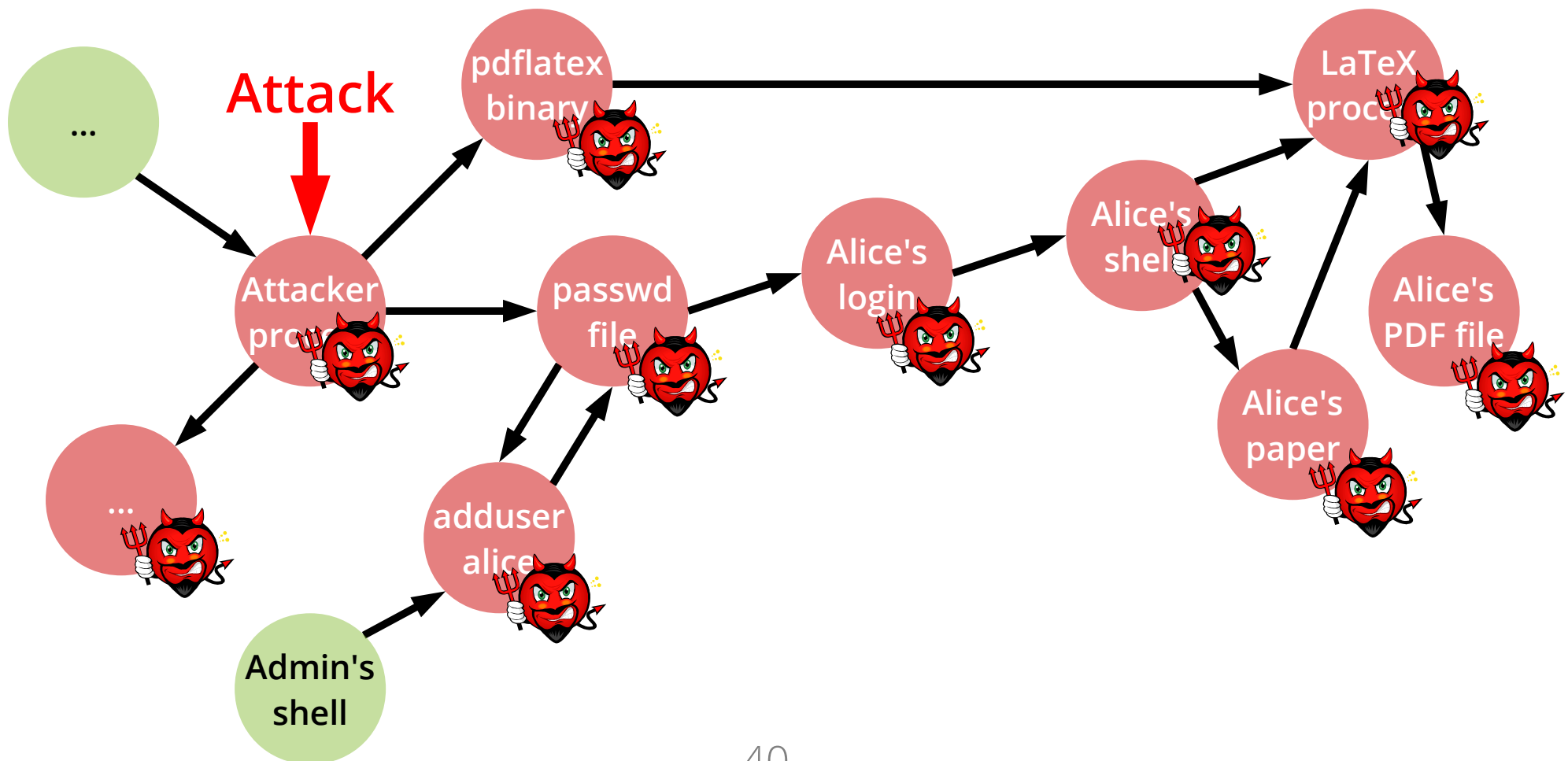
- Given attack, track down all affected files → restore those files from earlier backup



History strawman 1:

Taint tracking

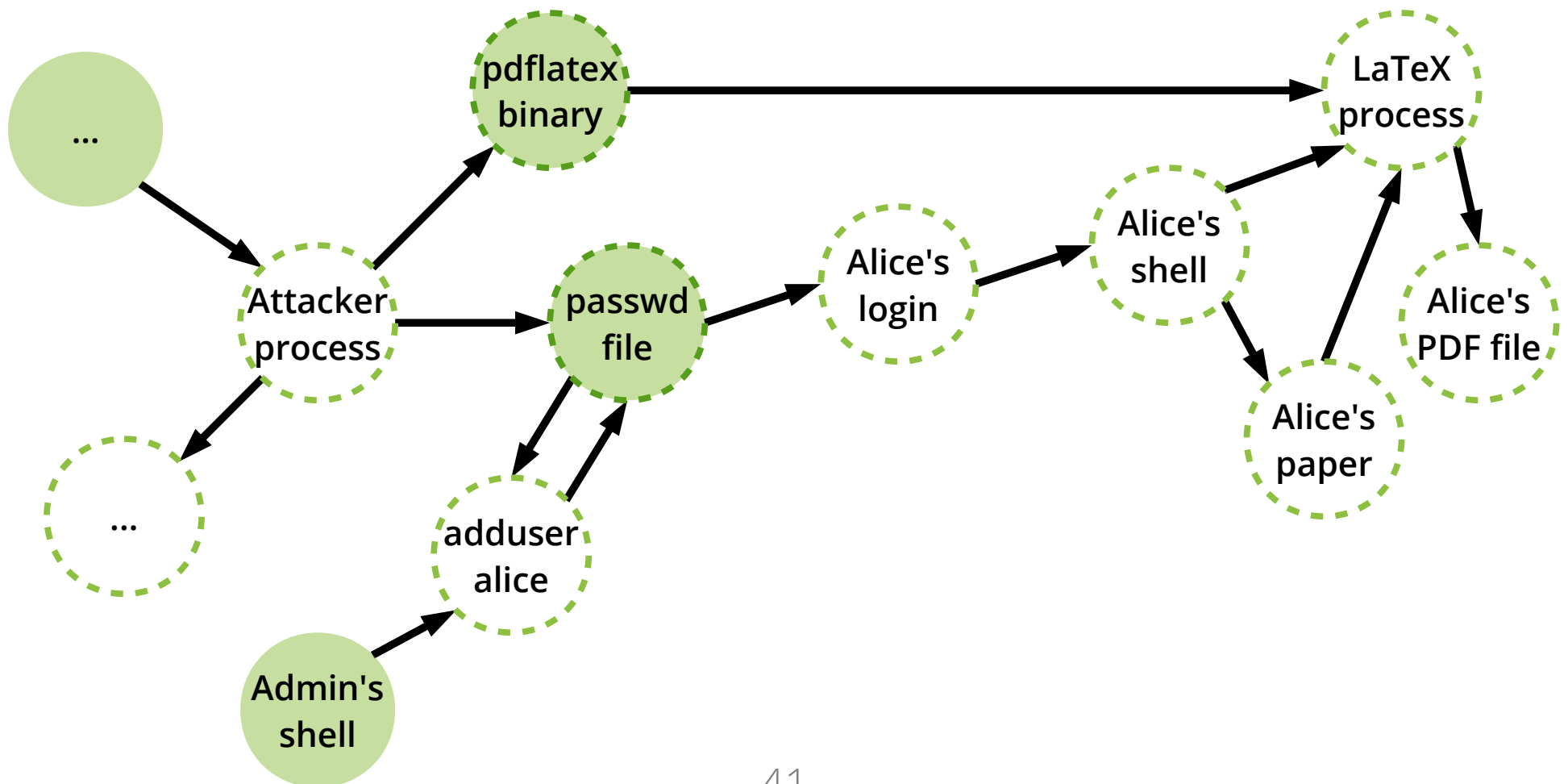
- Given attack, track down all affected files → restore those files from earlier backup



History strawman 1:

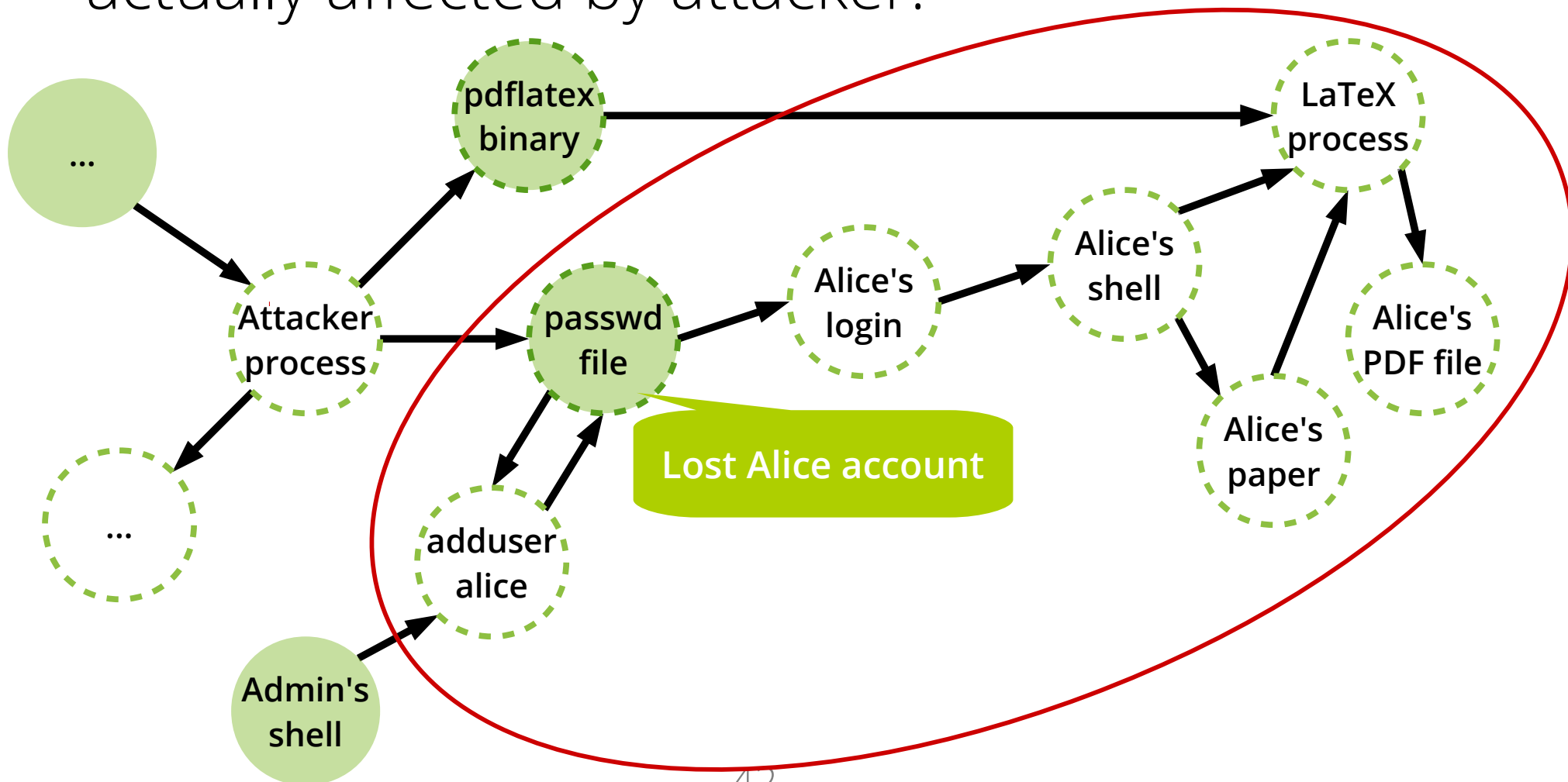
Taint tracking

- Given attack, track down all affected files → restore those files from earlier backup

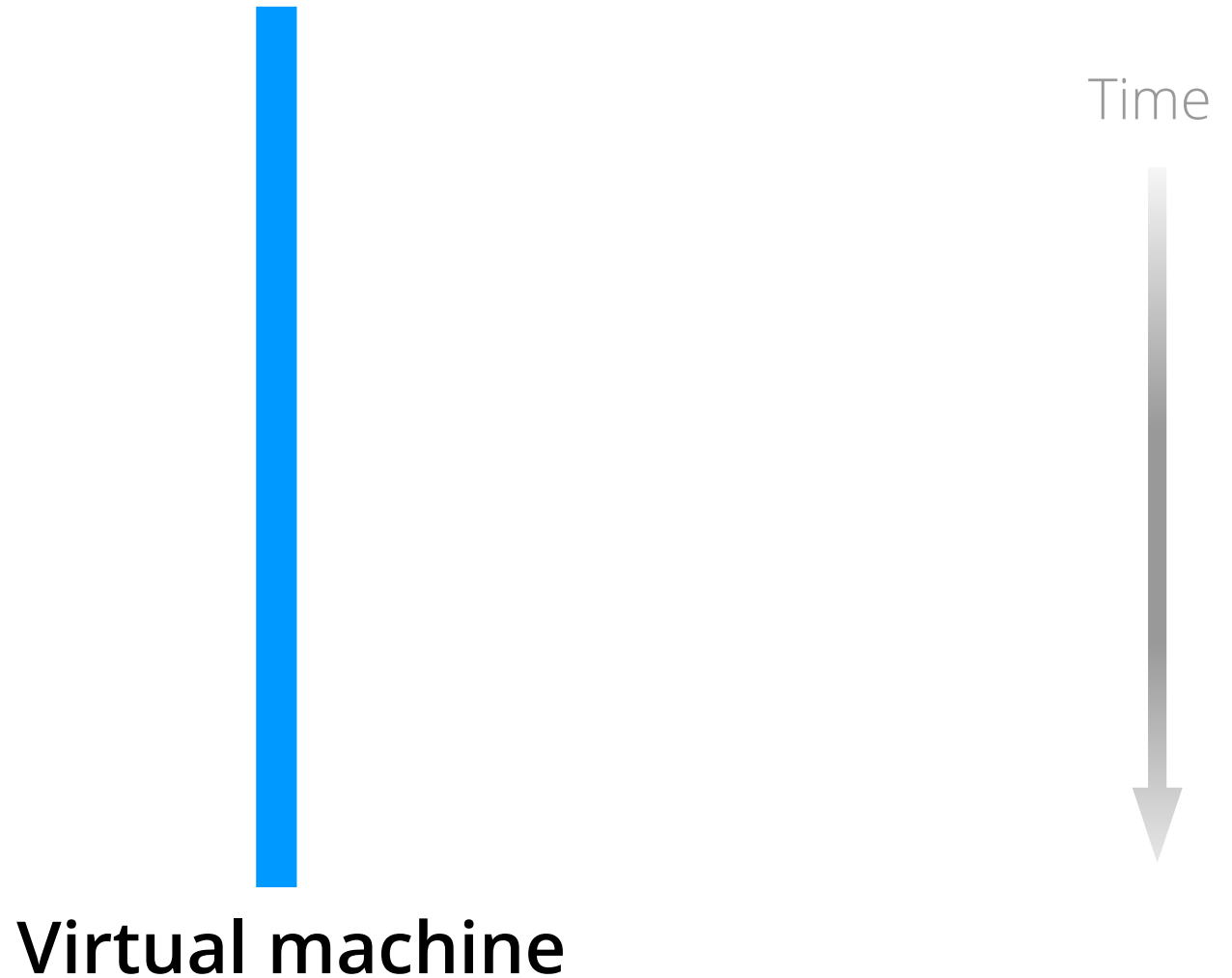


Problem with taint tracking: false positives

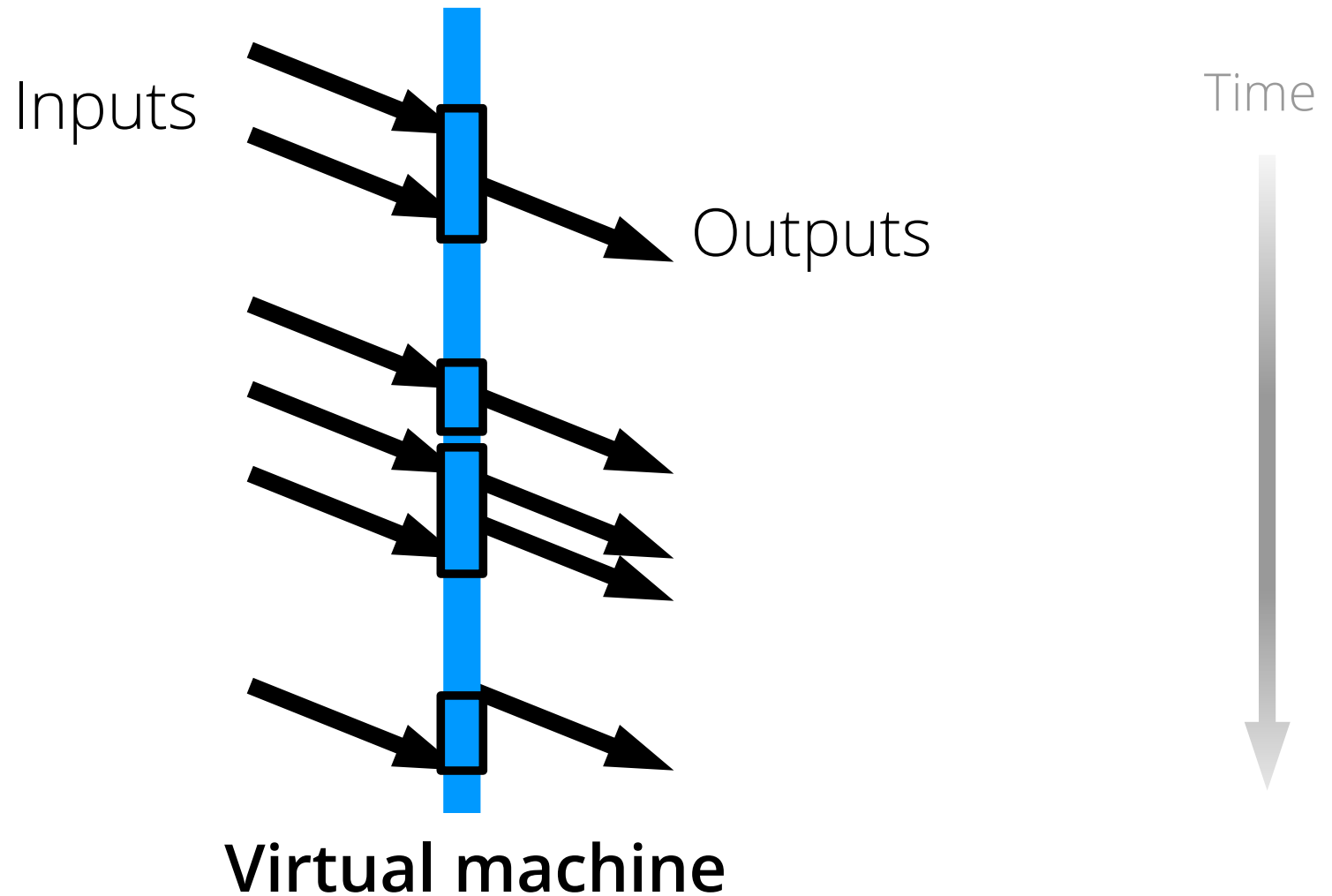
- Lost Alice's account and files that are not actually affected by attacker!



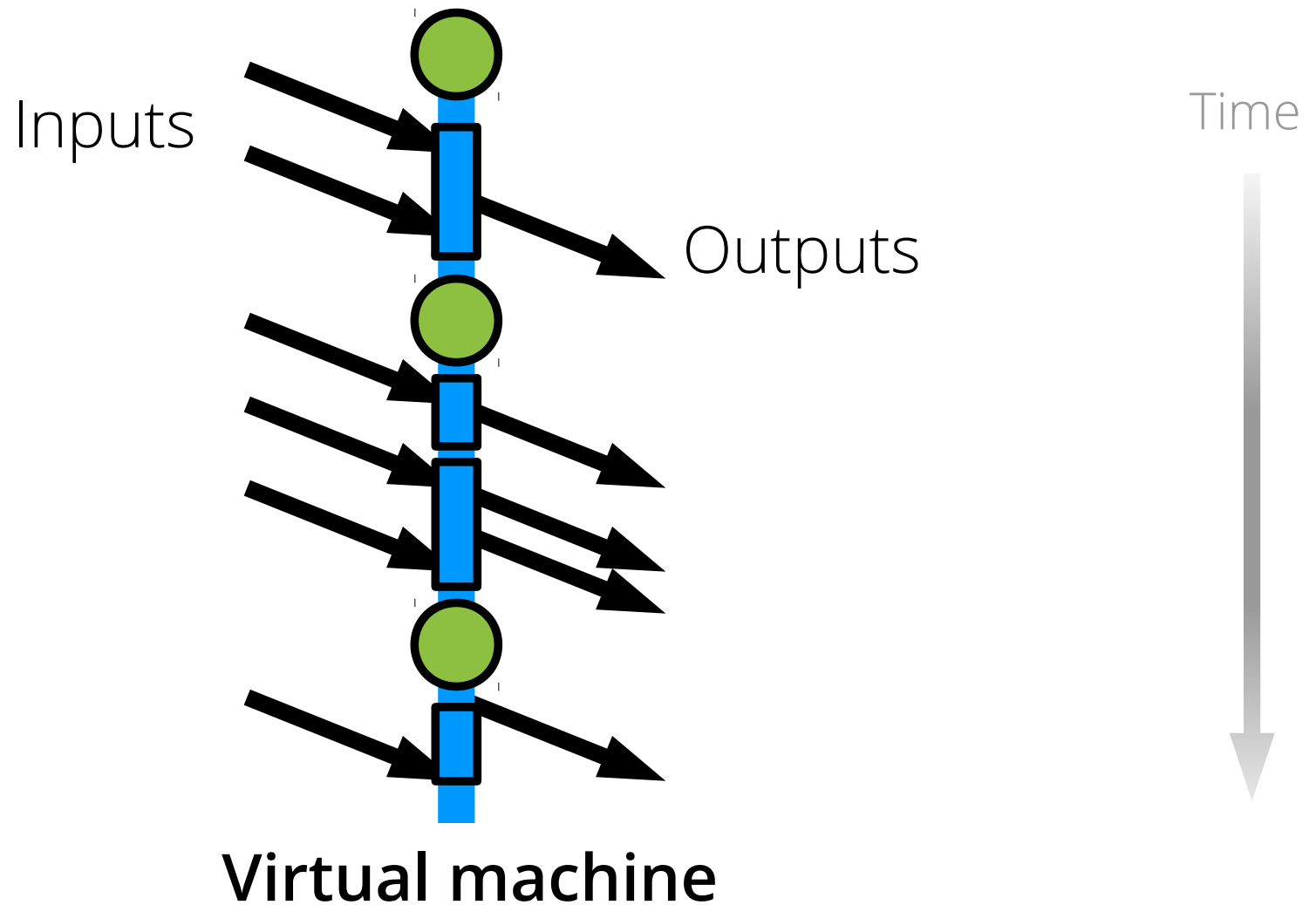
History strawman 2: VM replay



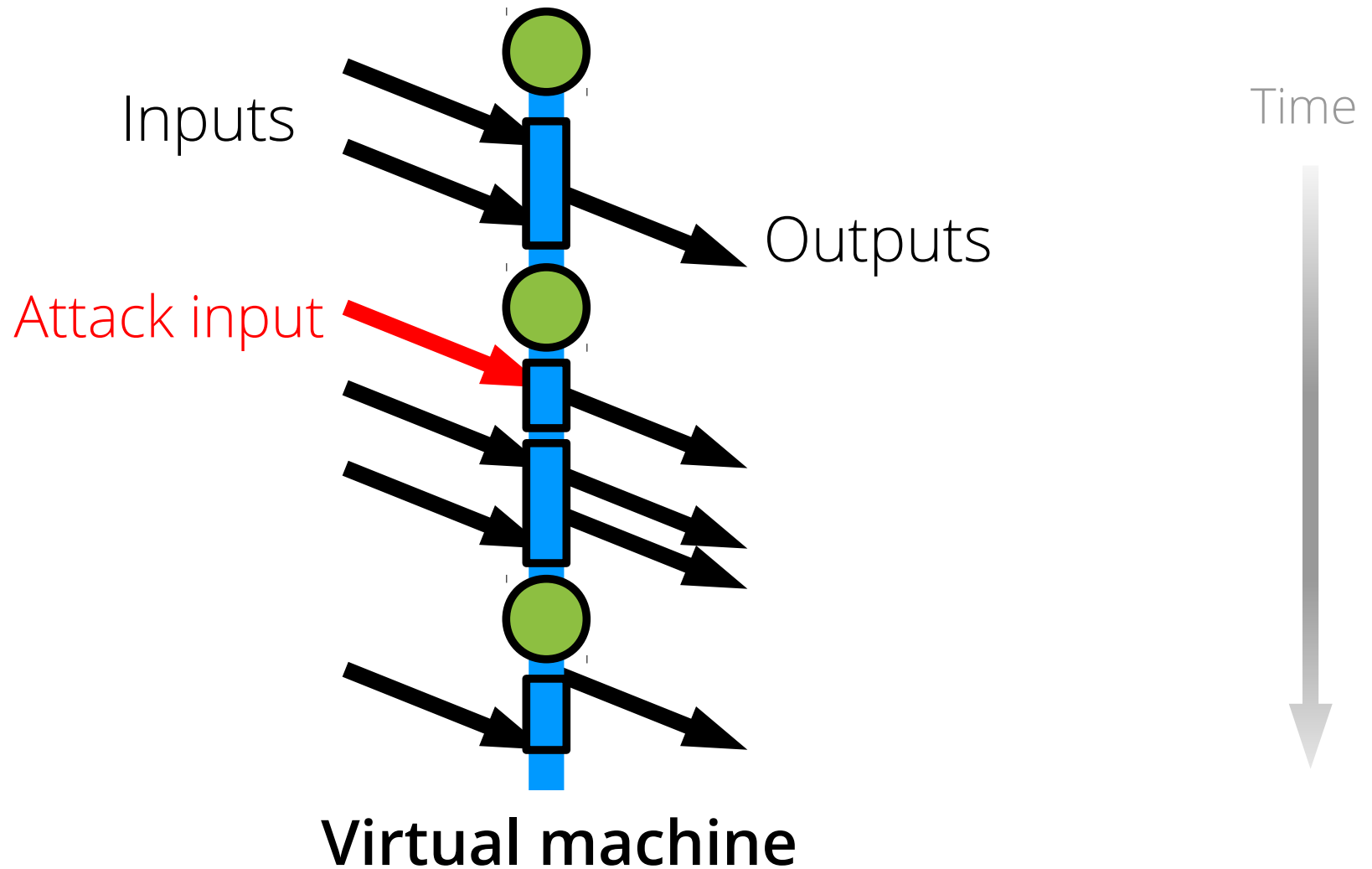
History strawman 2: VM replay



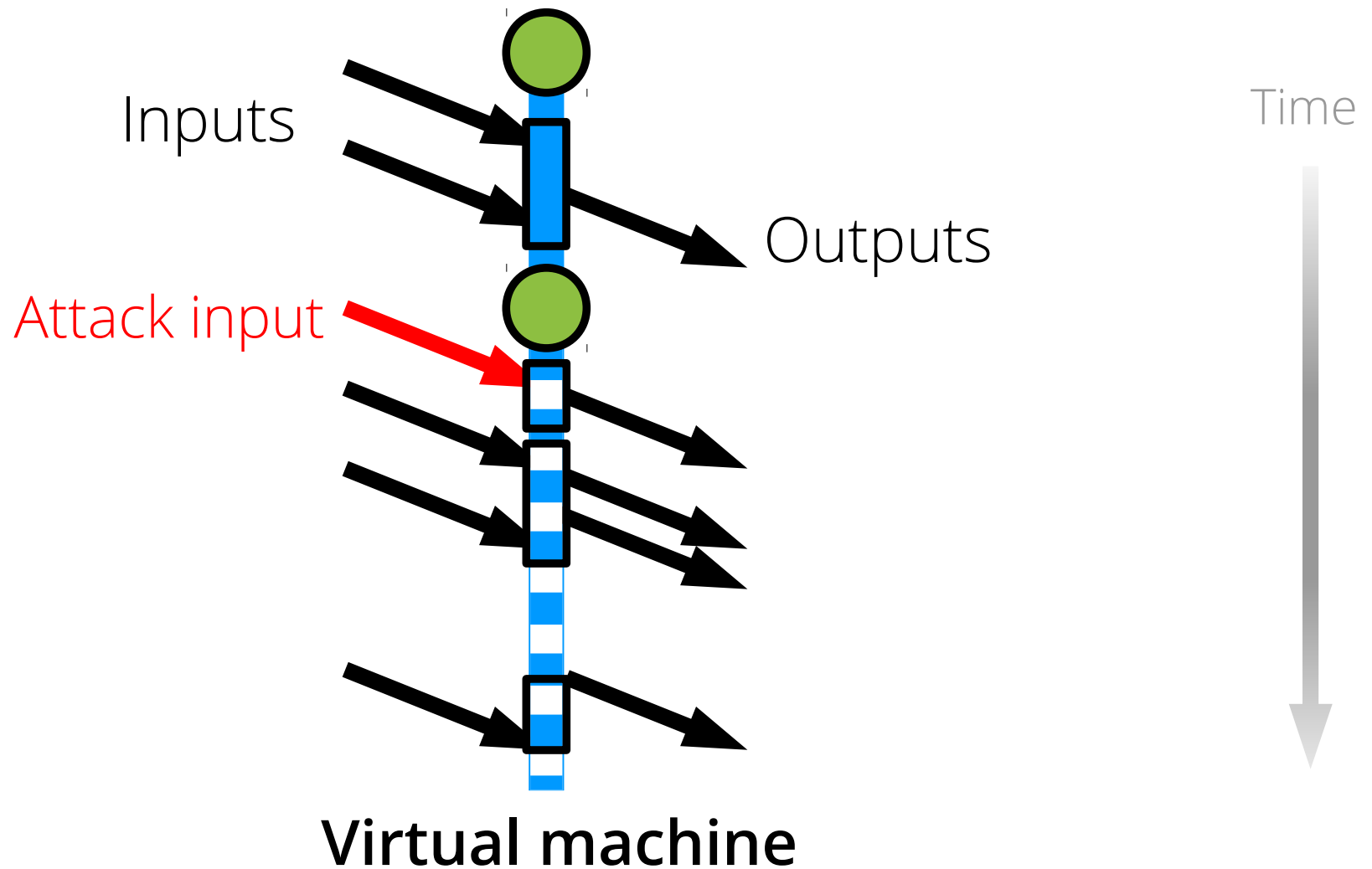
Periodic VM checkpoints



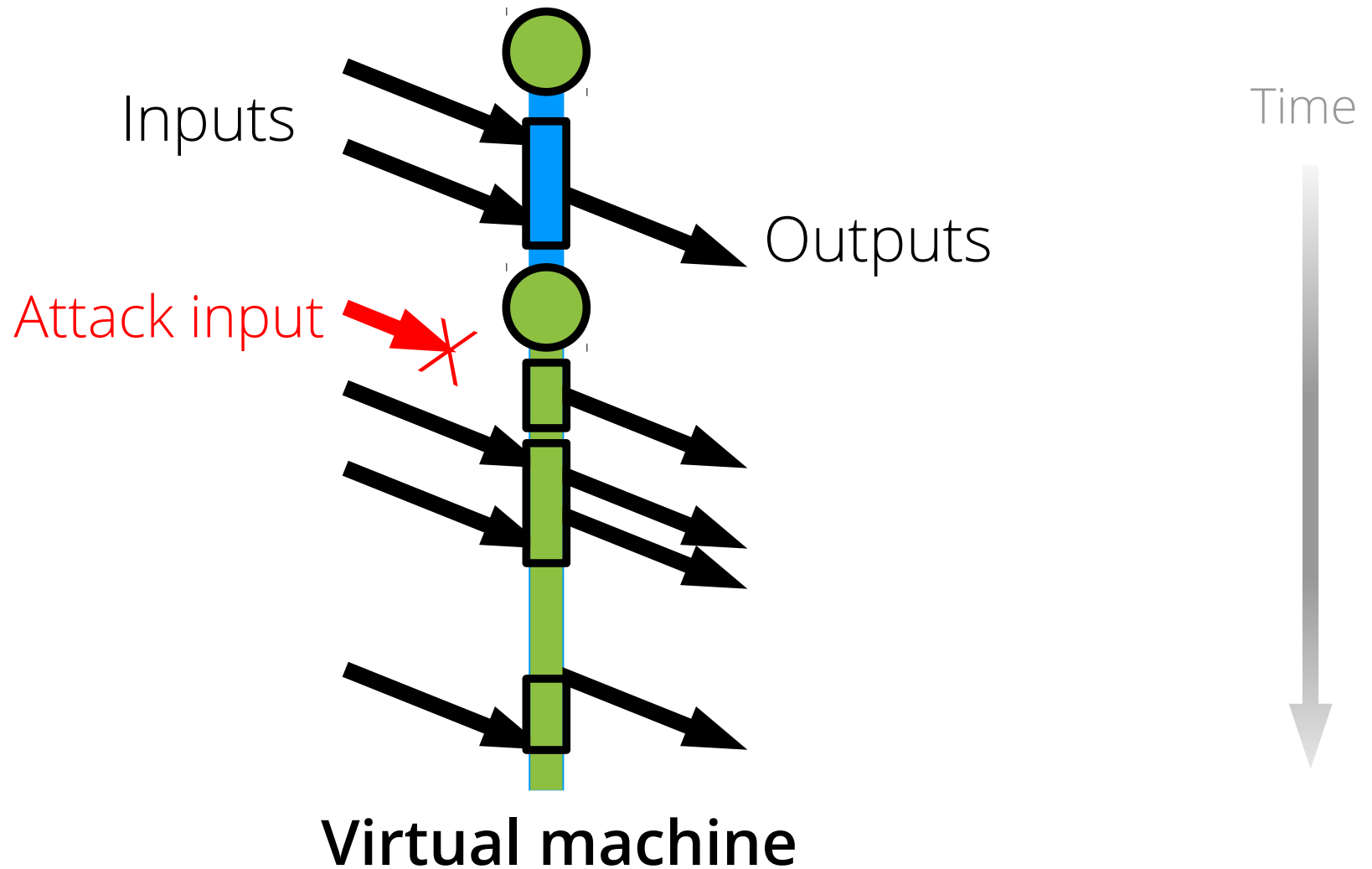
Step 1: identify attack input



Step 2: rollback to the latest checkpoint



Step 3: replay non-attack inputs



Problems with VM replay

- **VM replay is expensive**
 - Repairing a week-old attack needs a week for replay
- **Past inputs are meaningless to new system**
 - Non-determinism: new SSH crypto keys ...
 - Deterministic replay won't work

Retro's approach:

Action history graph

- **Represent fine-grained history**
 - Includes kernel objects, system calls, function calls, ...
 - Assume tamper-proof kernel, storage

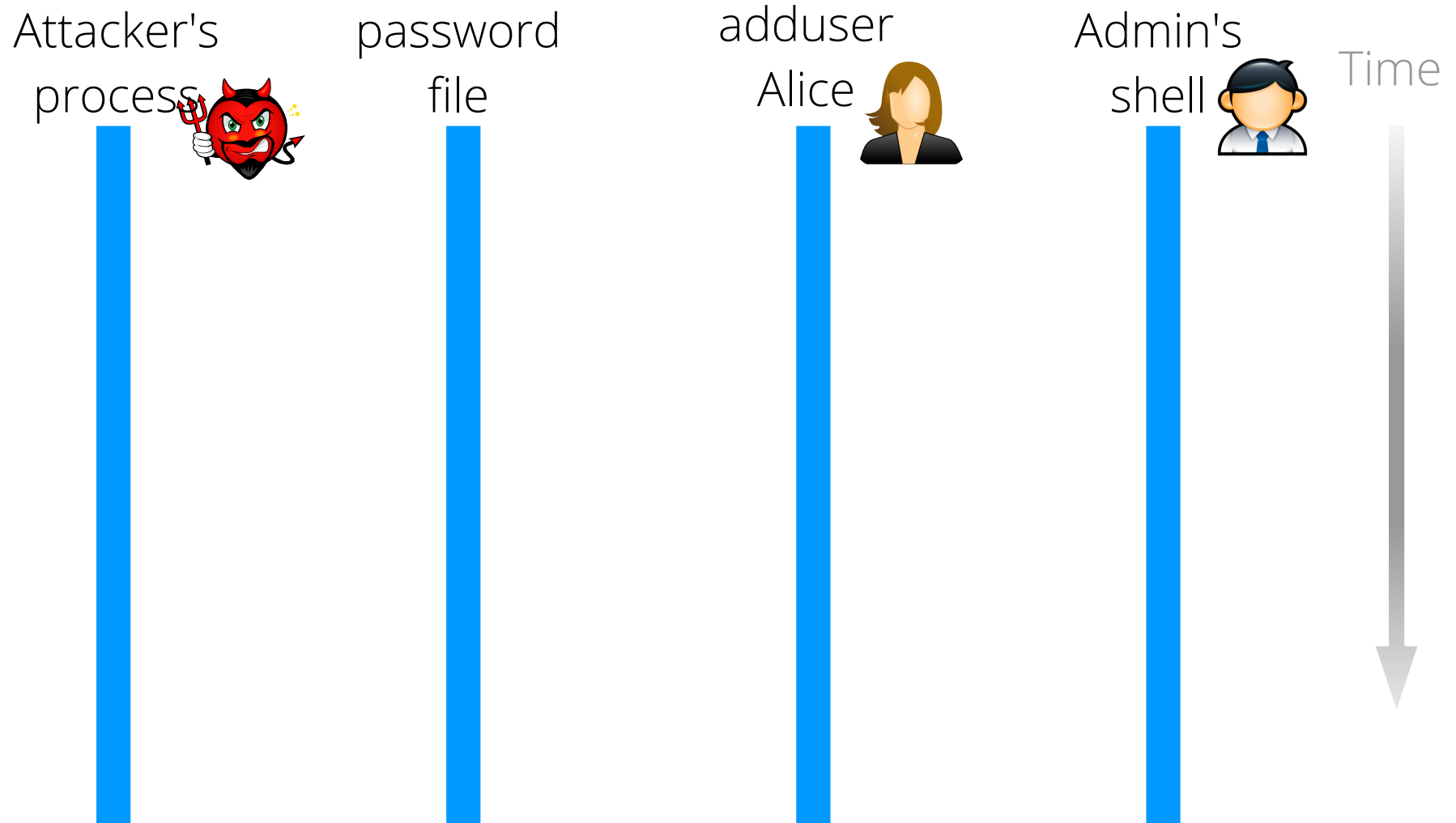
Retro's approach:

Action history graph

- **Represent fine-grained history**
 - Includes kernel objects, system calls, function calls, ...
 - Assume tamper-proof kernel, storage
- ***Rollback*** objects directly affected by attack
 - Avoid the false positives of Taint tracking
- ***Selectively re-execute*** indirectly affected actions
 - Avoid the expensive VM replay

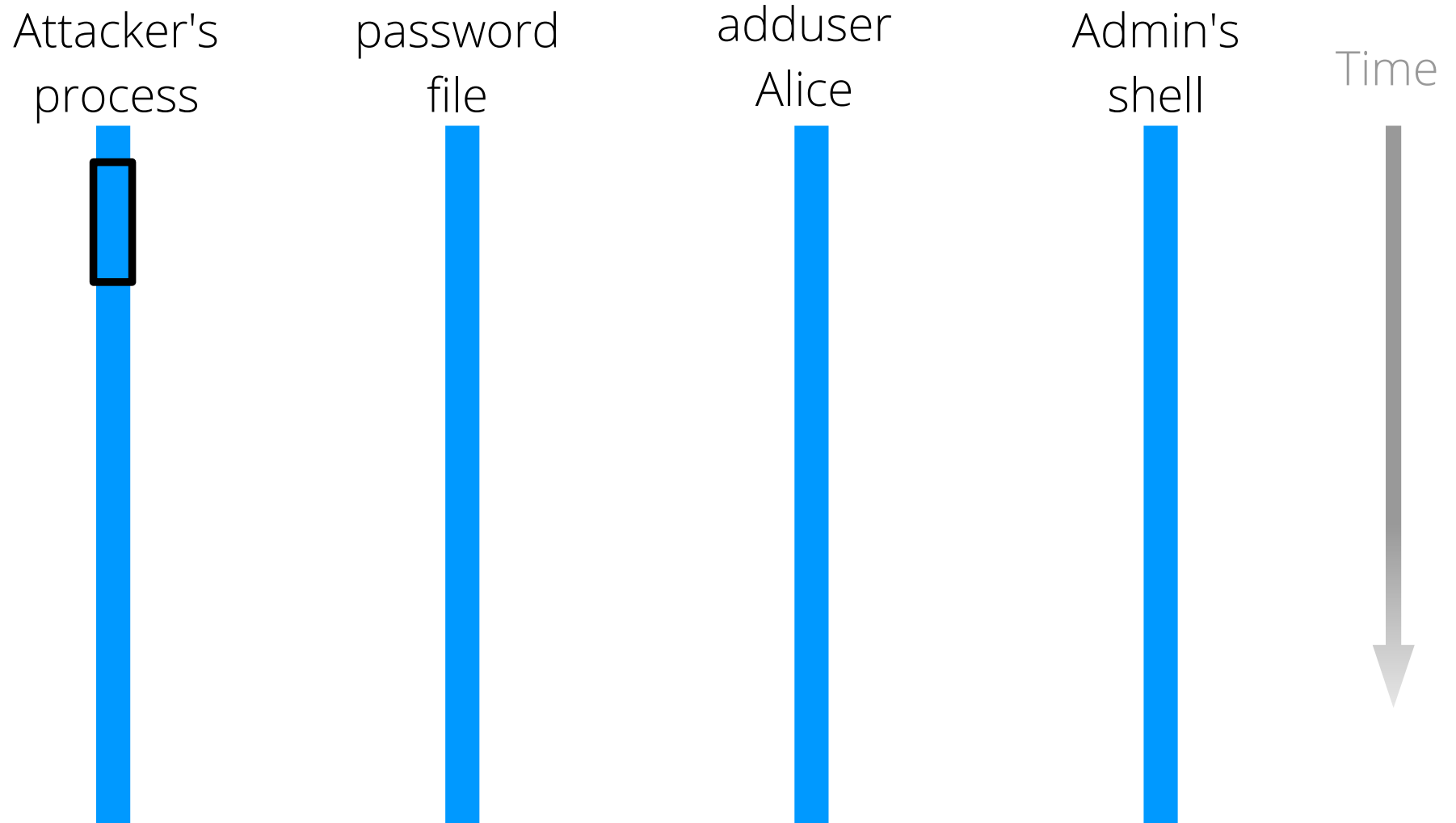
Action history graph:

Objects represent files, processes



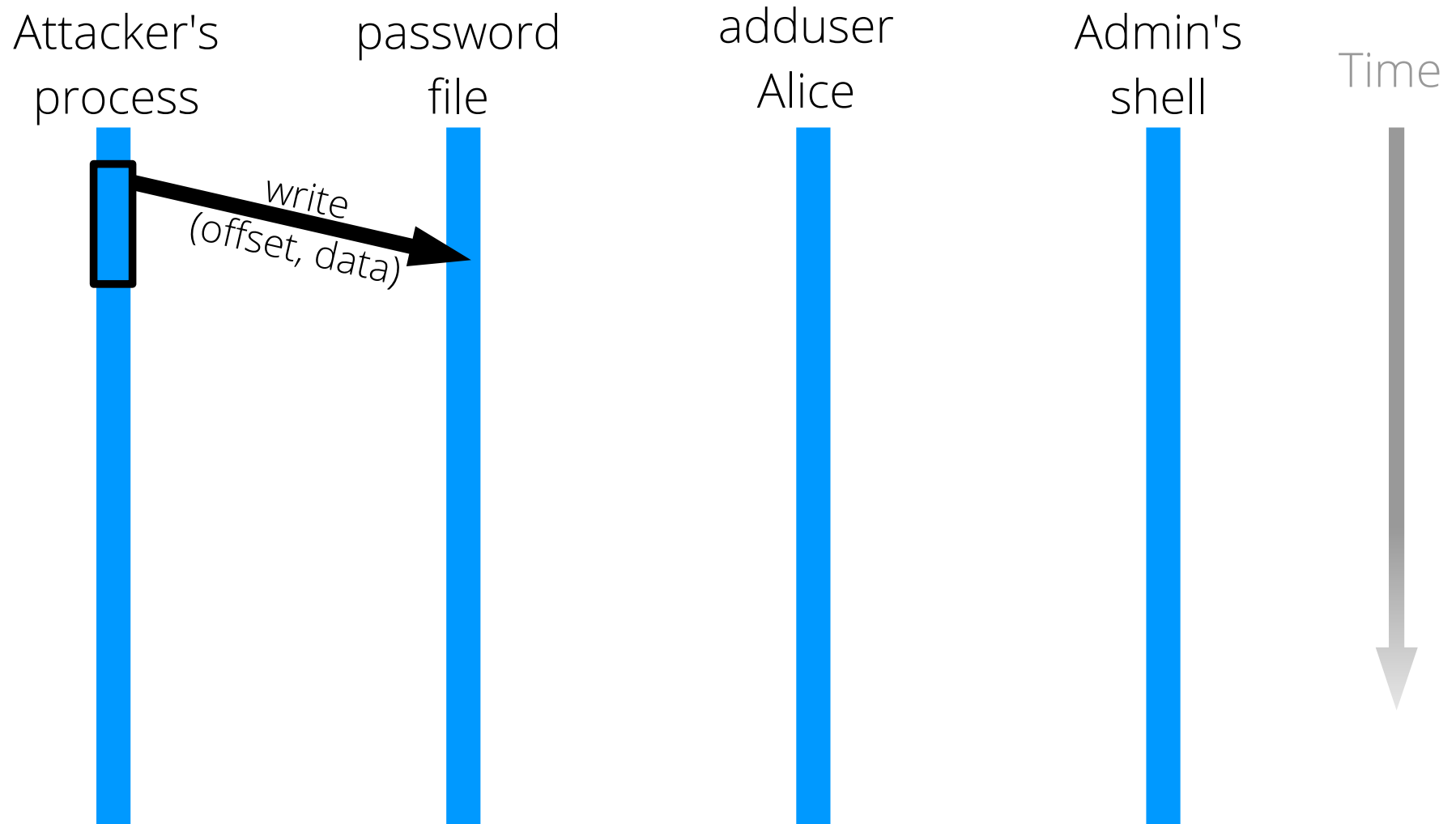
Action history graph:

Actions represent execution (syscall)



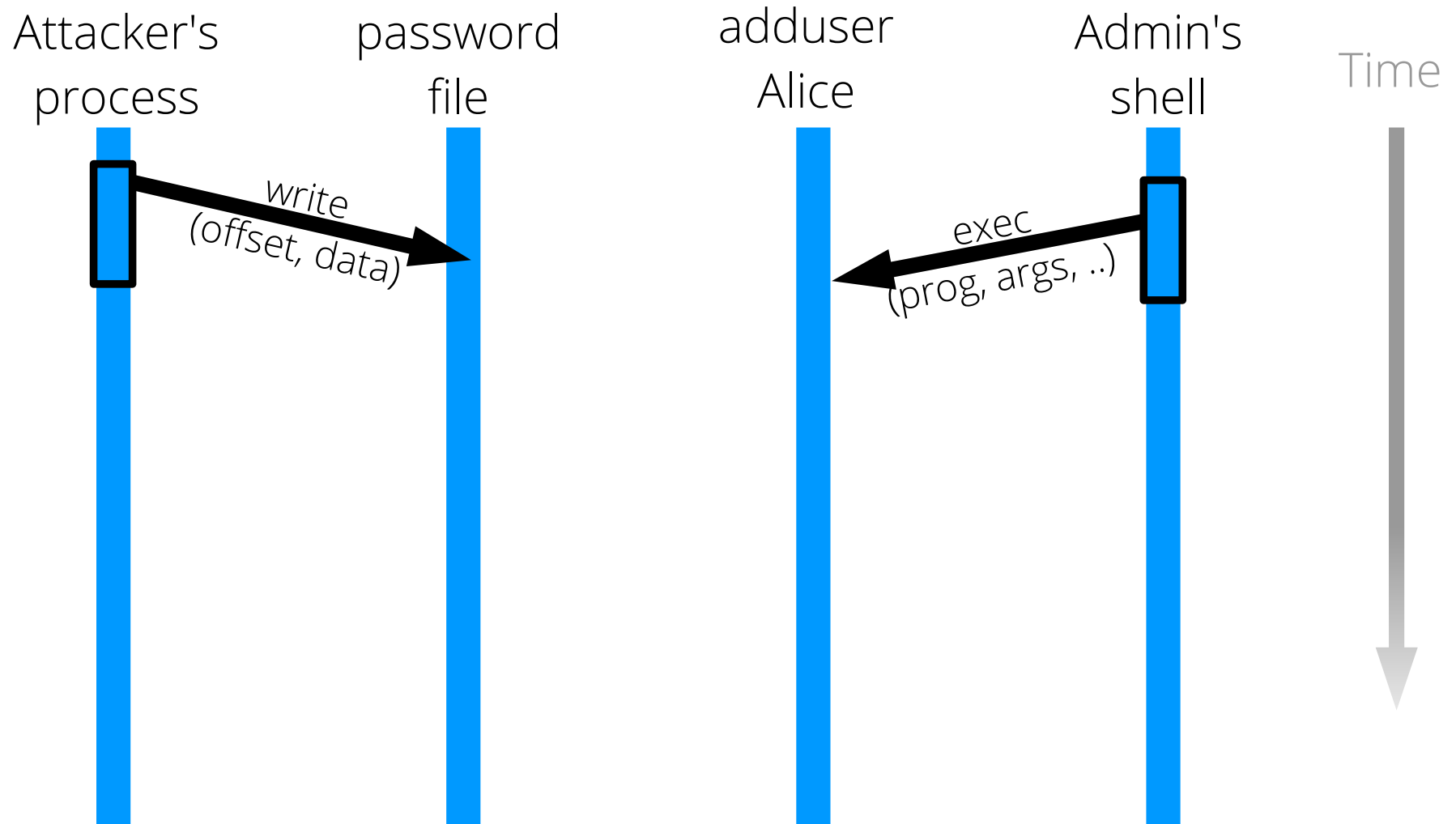
Action history graph:

Actions have dependencies



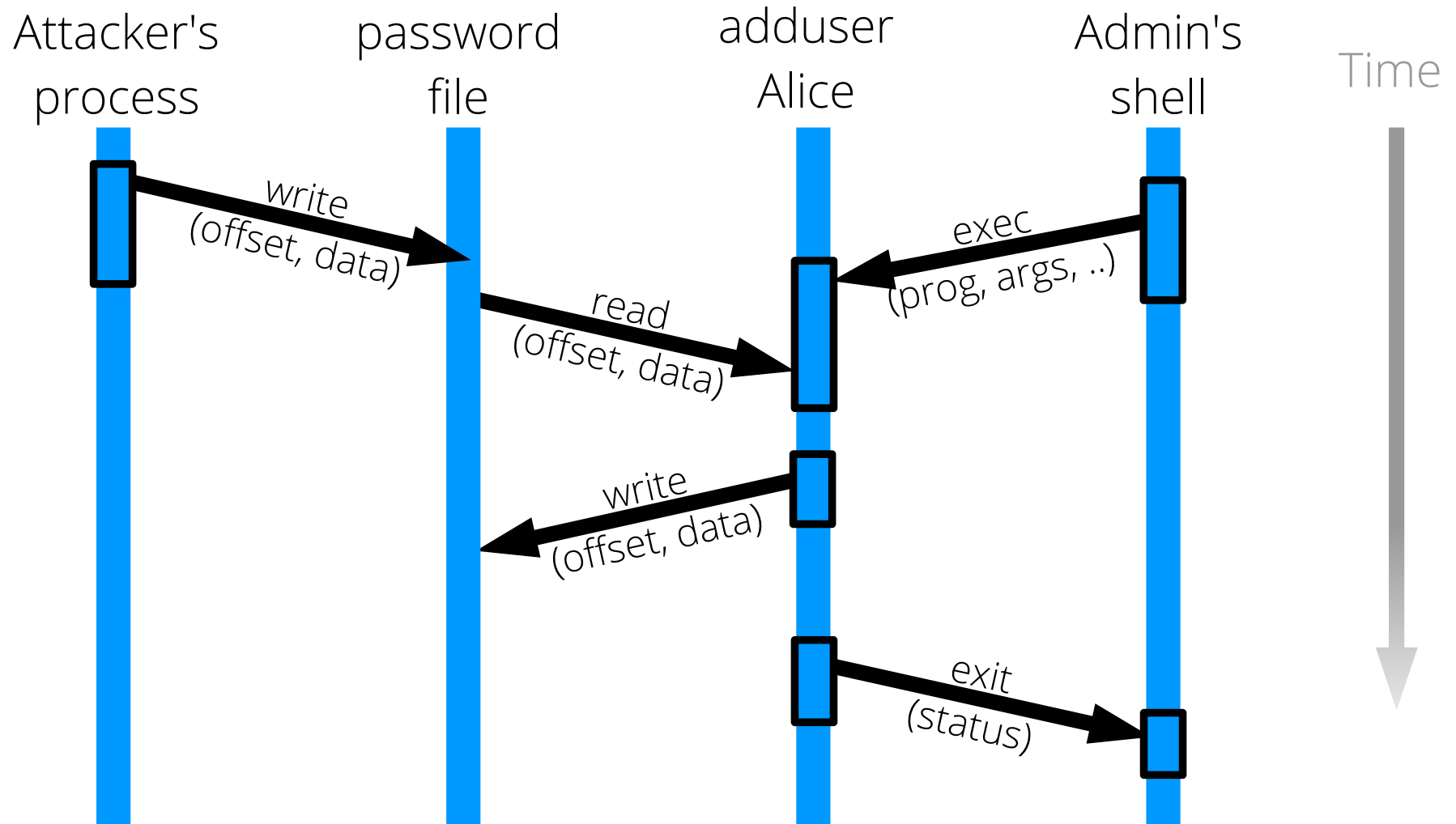
Action history graph:

Actions have dependencies



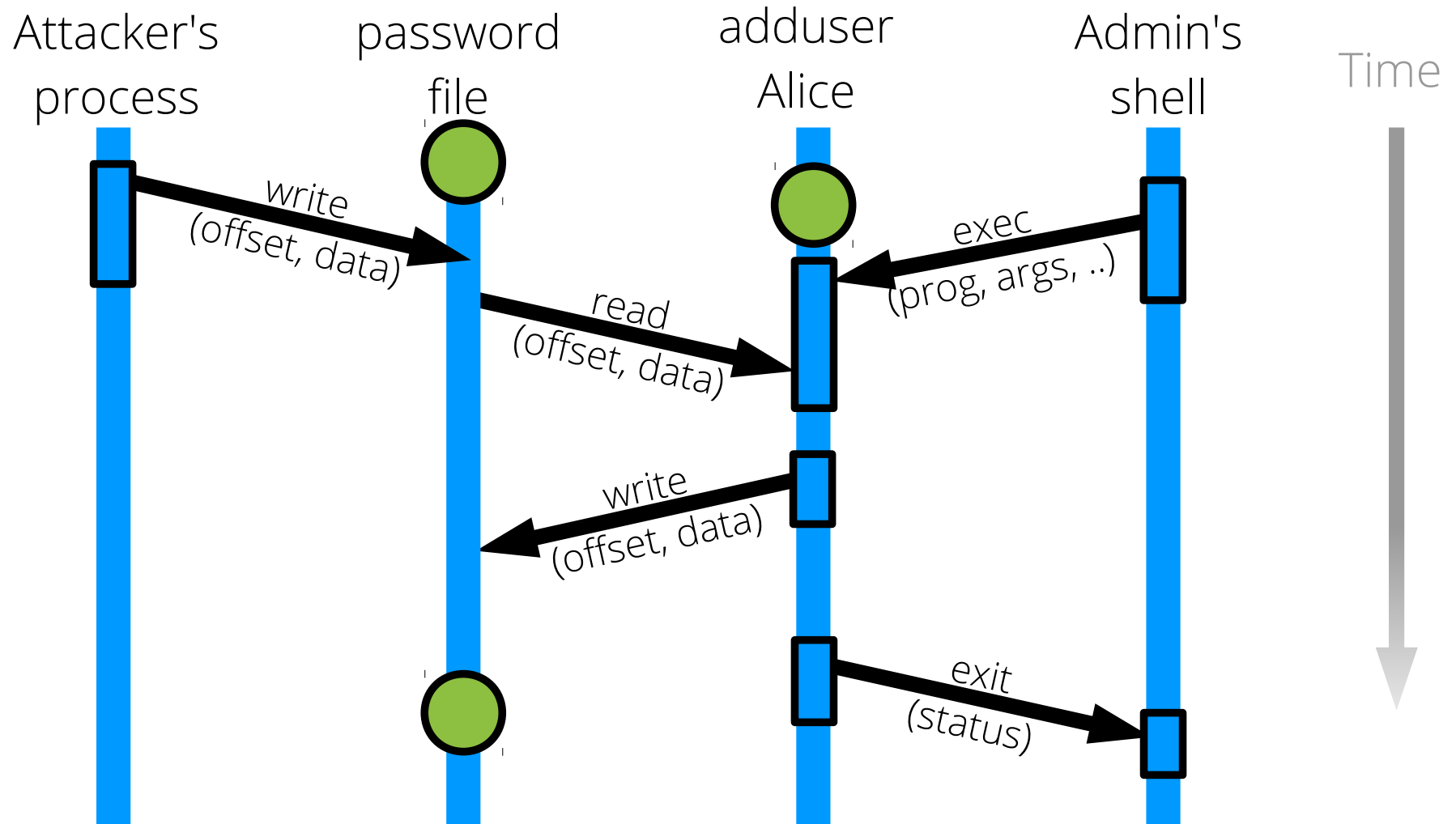
Action history graph:

Actions have dependencies

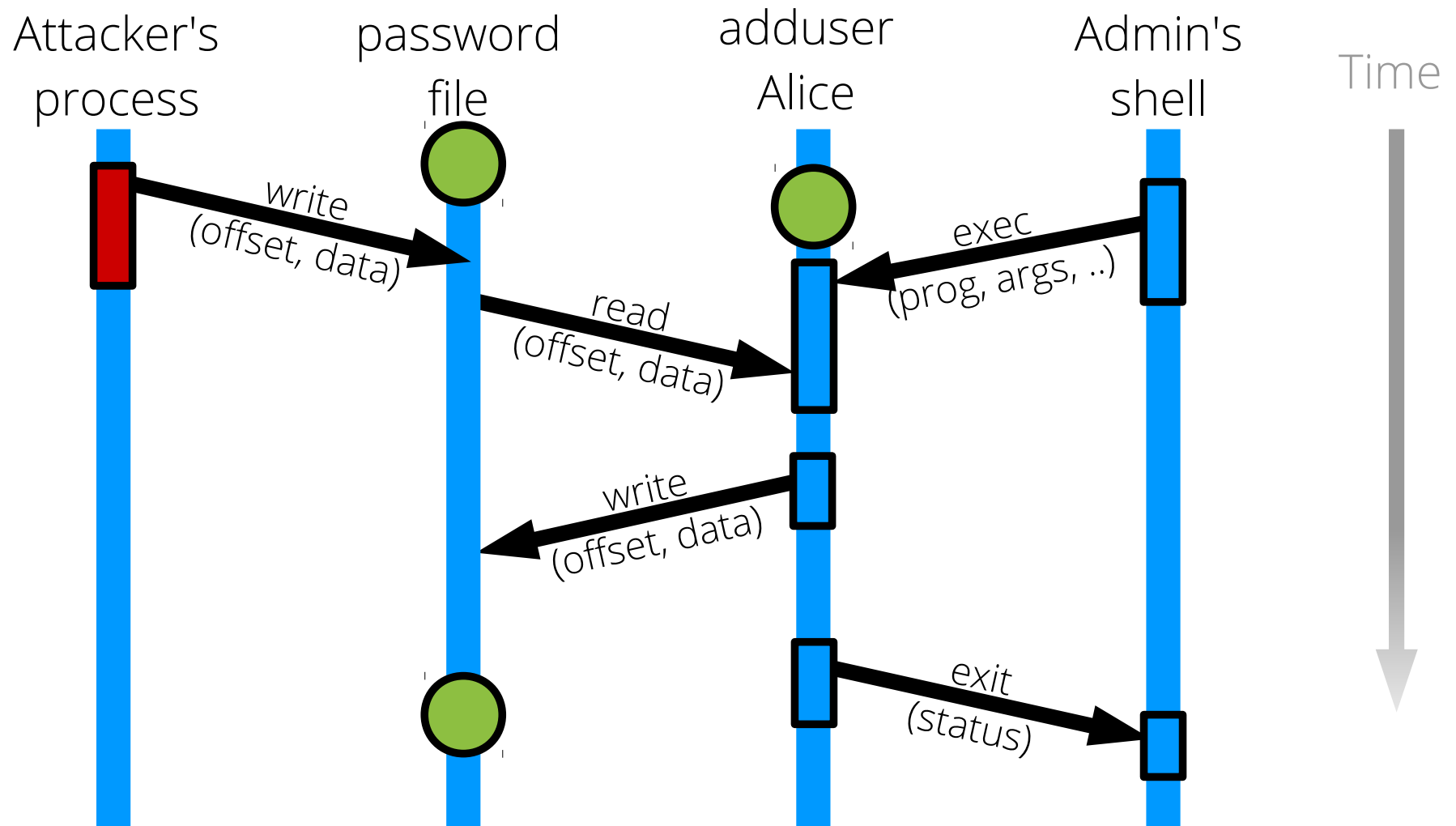


Action history graph:

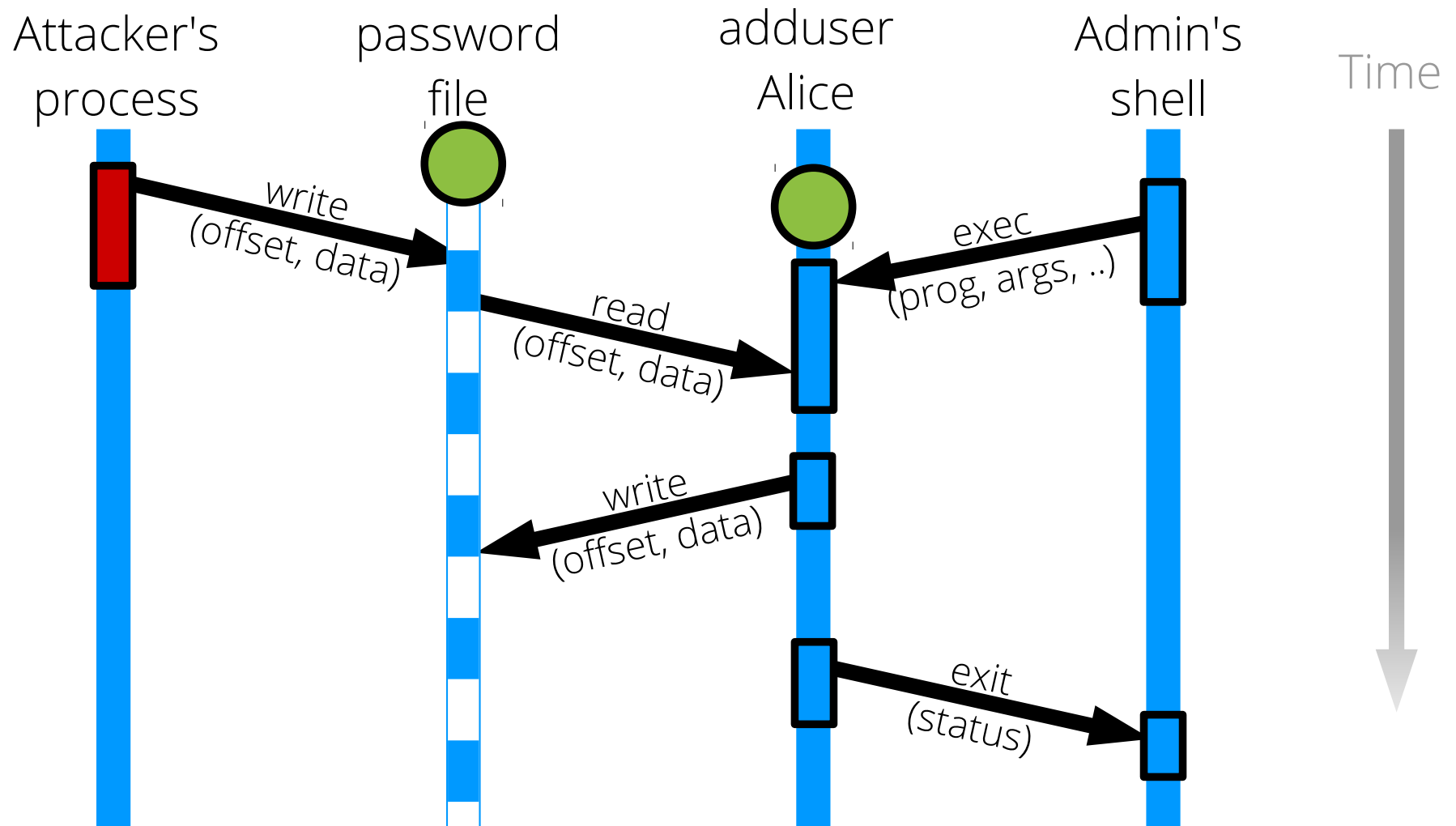
Objects have checkpoints



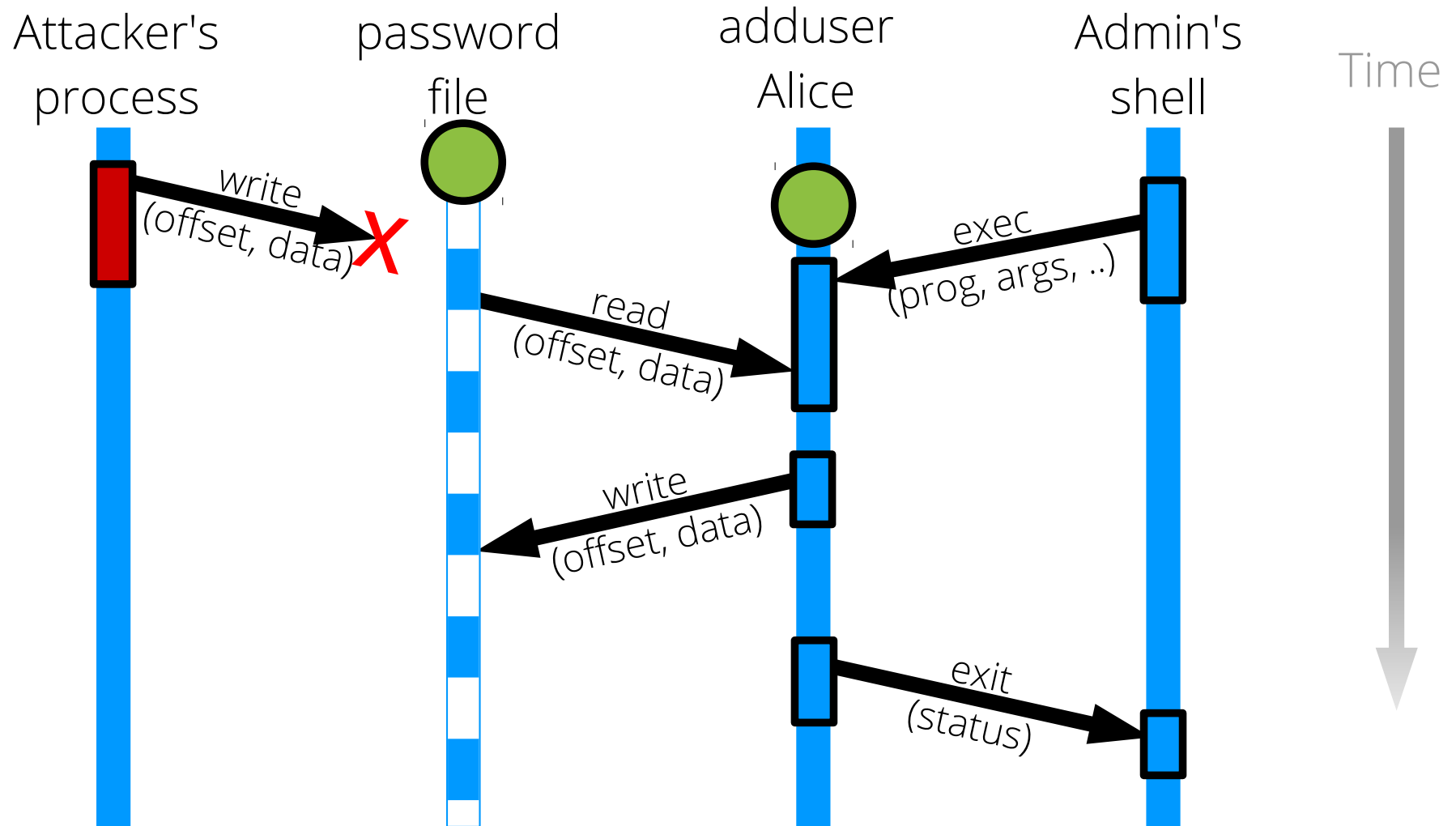
Step 1: find attack action



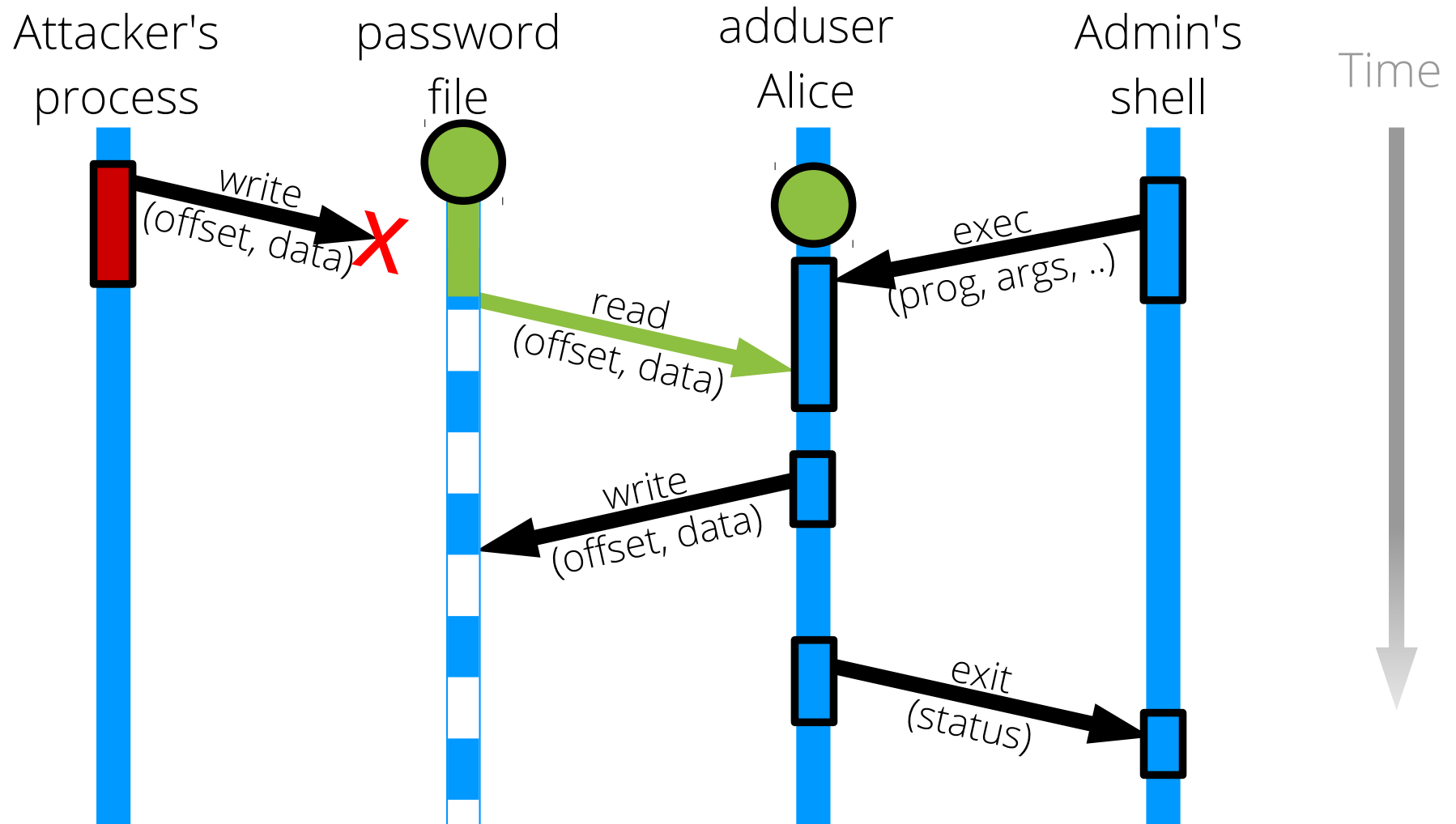
Step 2: rollback affected objects



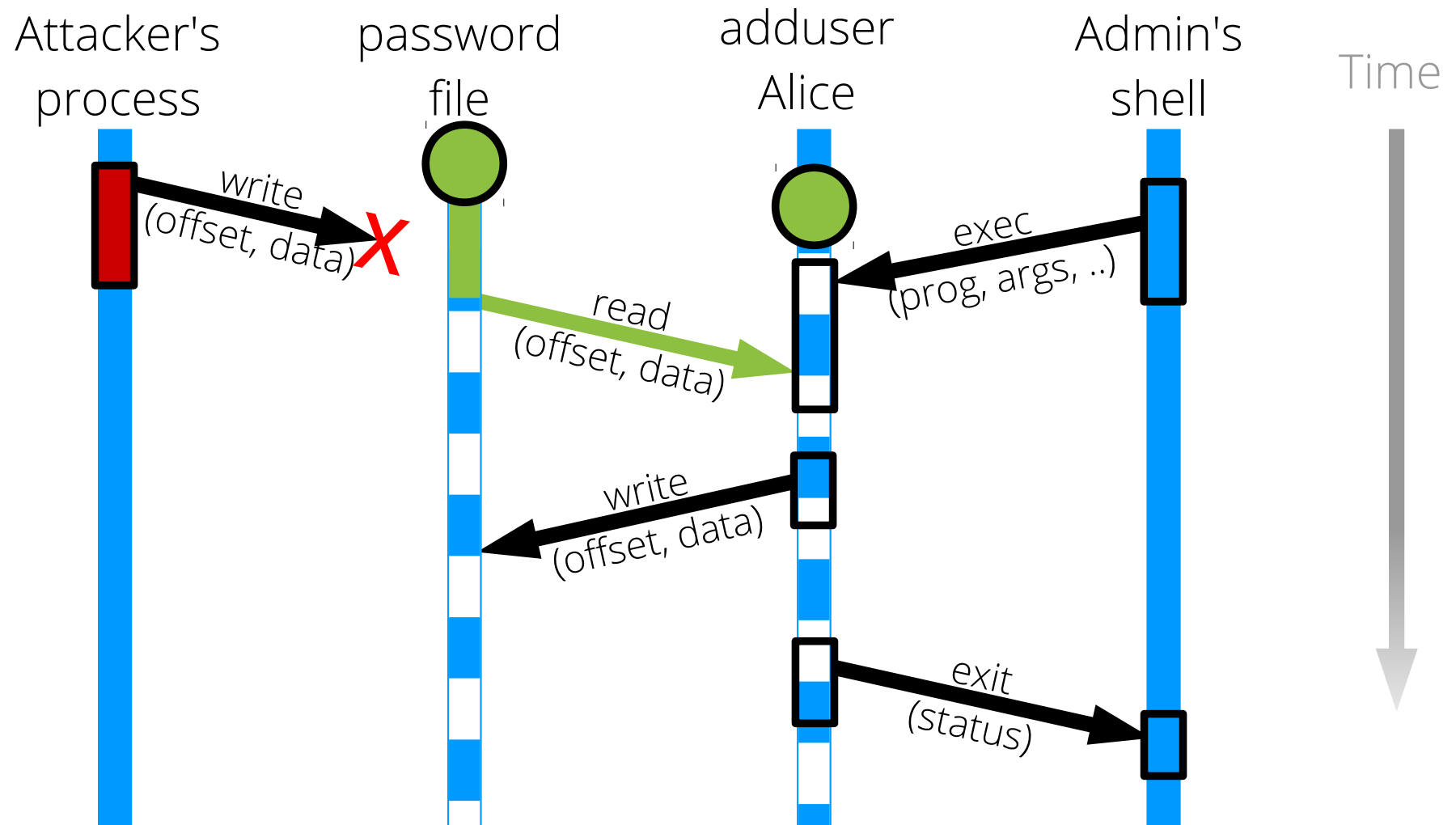
Step 3: skip attack action



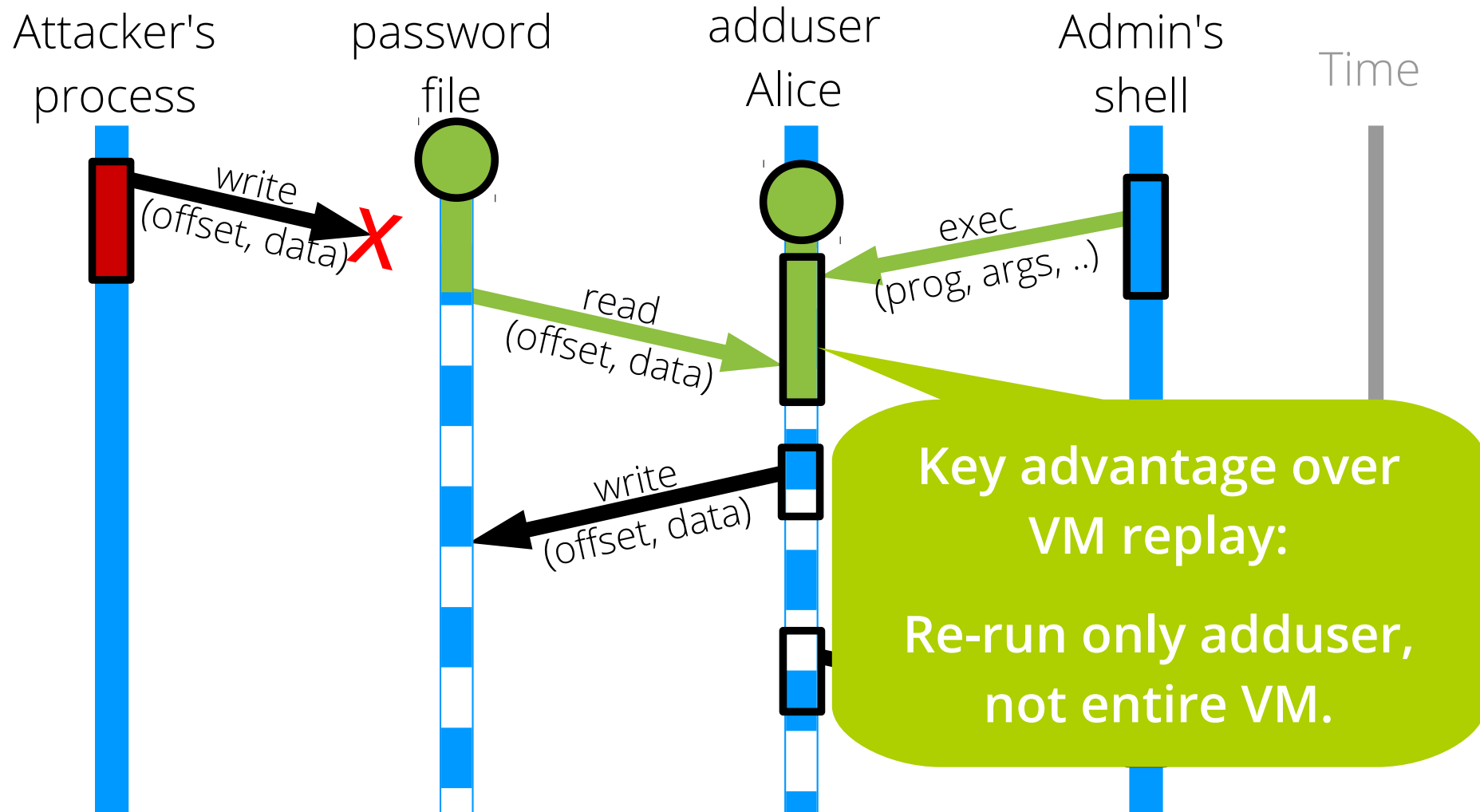
Step 4: redo non-attack actions



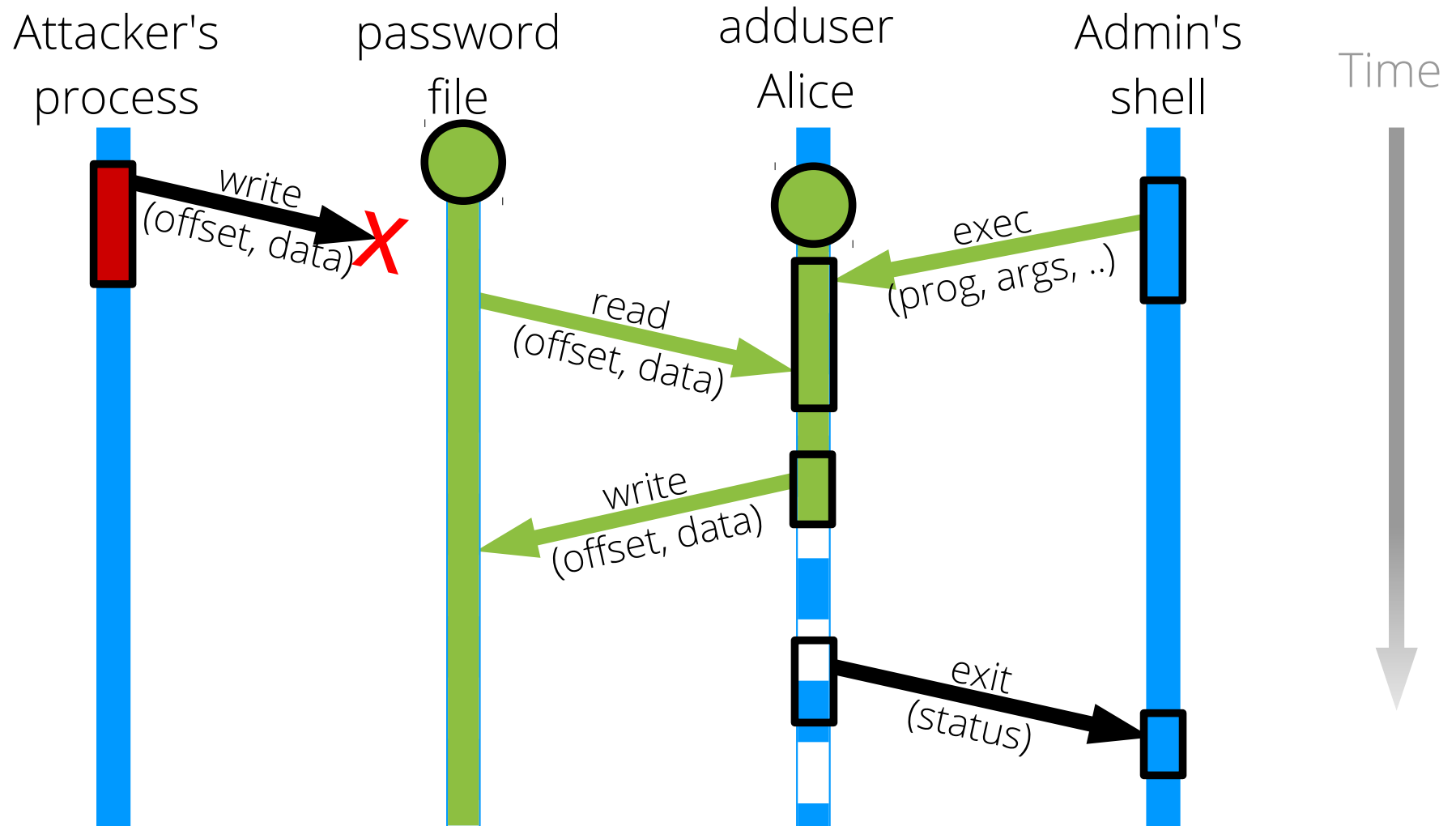
Repeat step 2: rollback objects



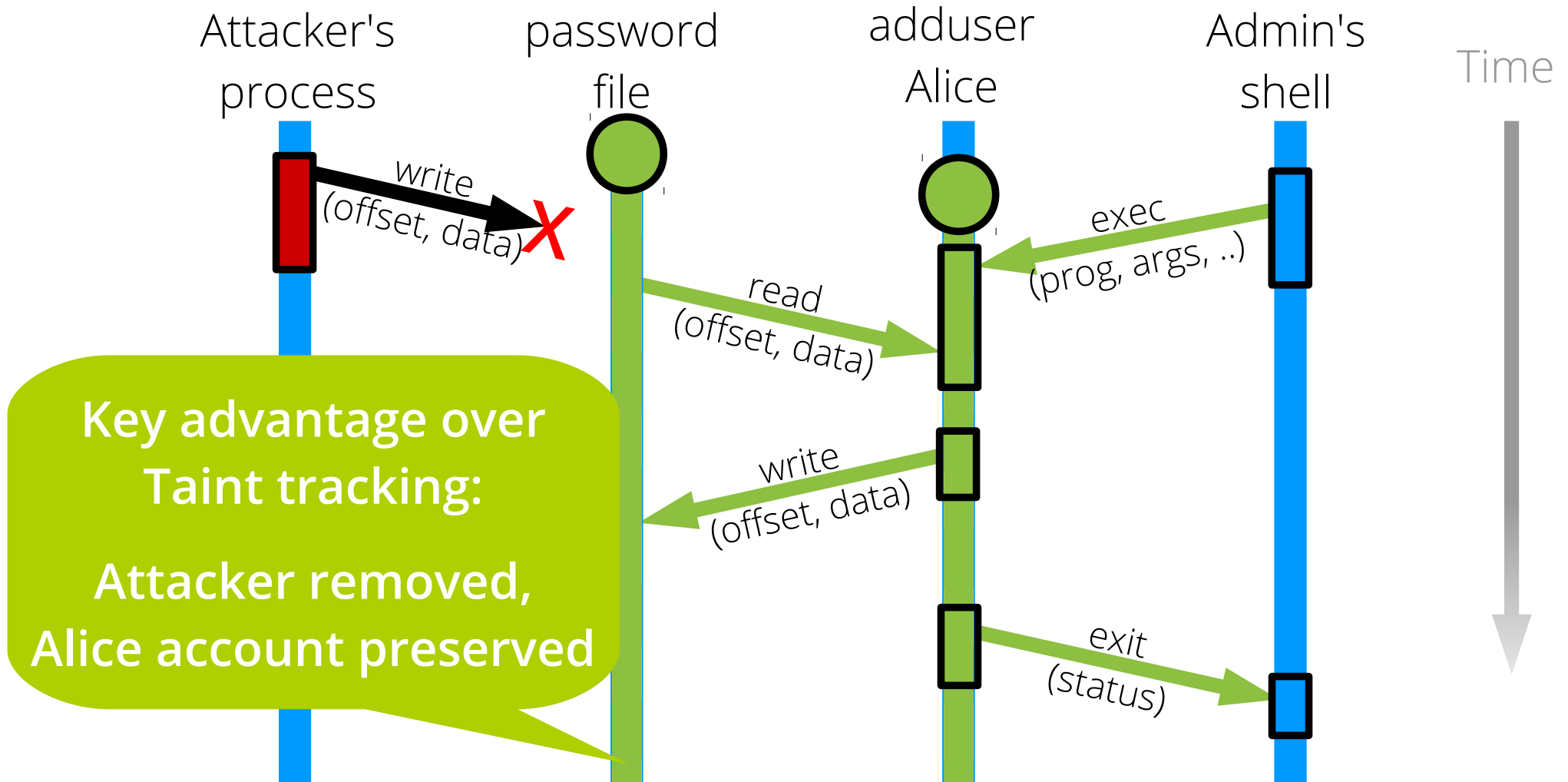
Repeat step 3: redo actions



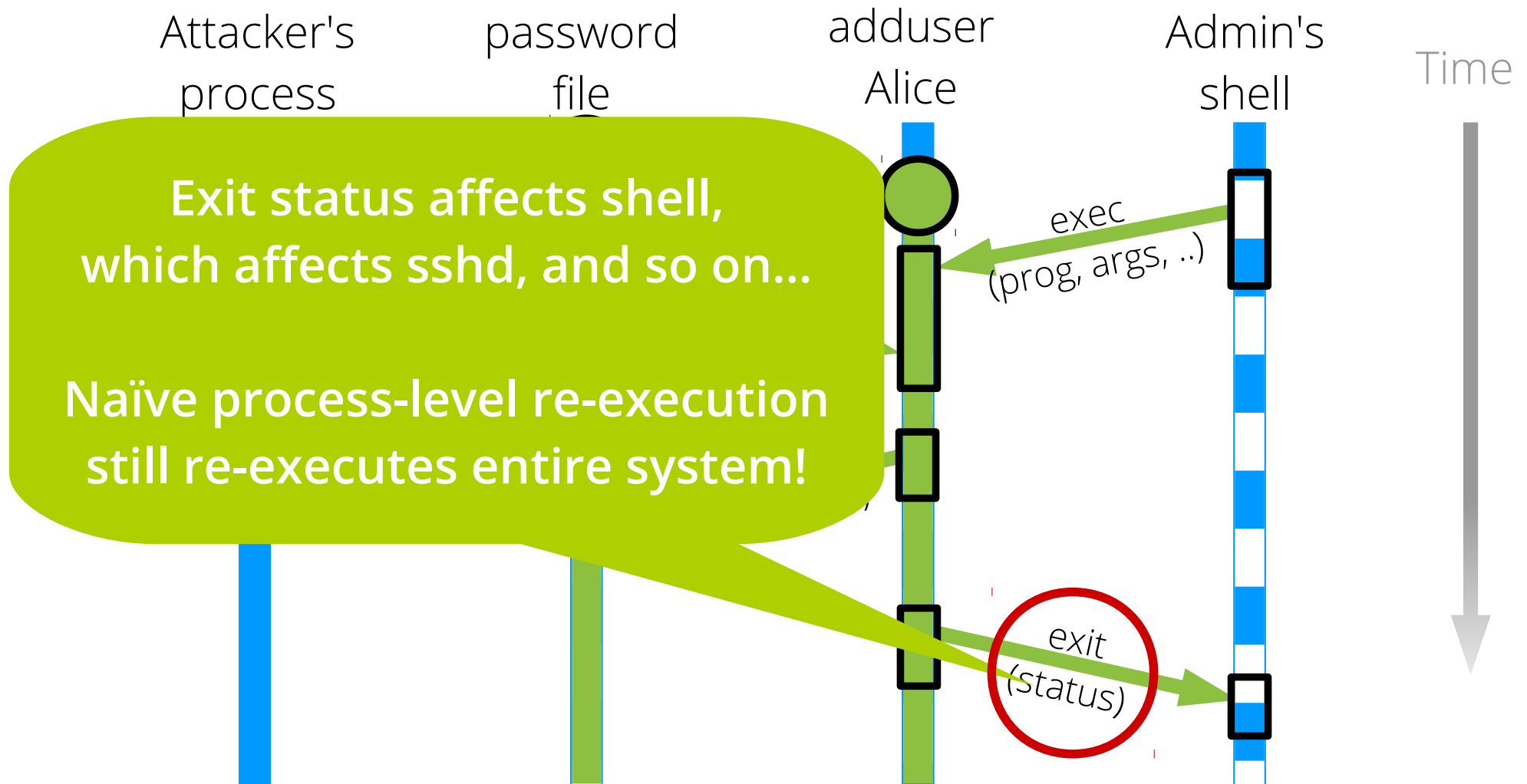
Repeat step 3: redo actions



Repeat step 3: redo actions



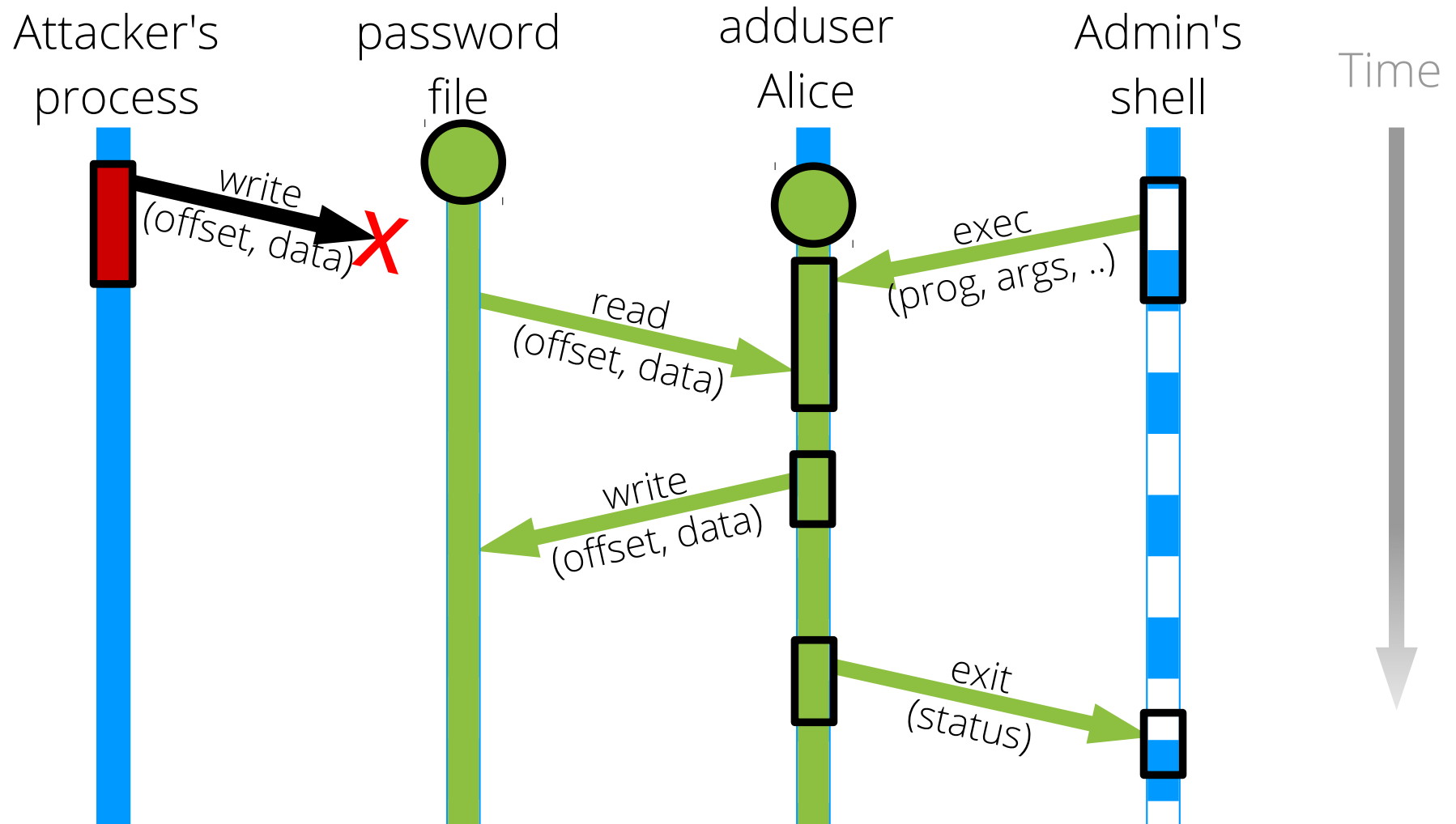
Challenge: how to avoid re-executing everything?



Observation: Admin's shell was not affected

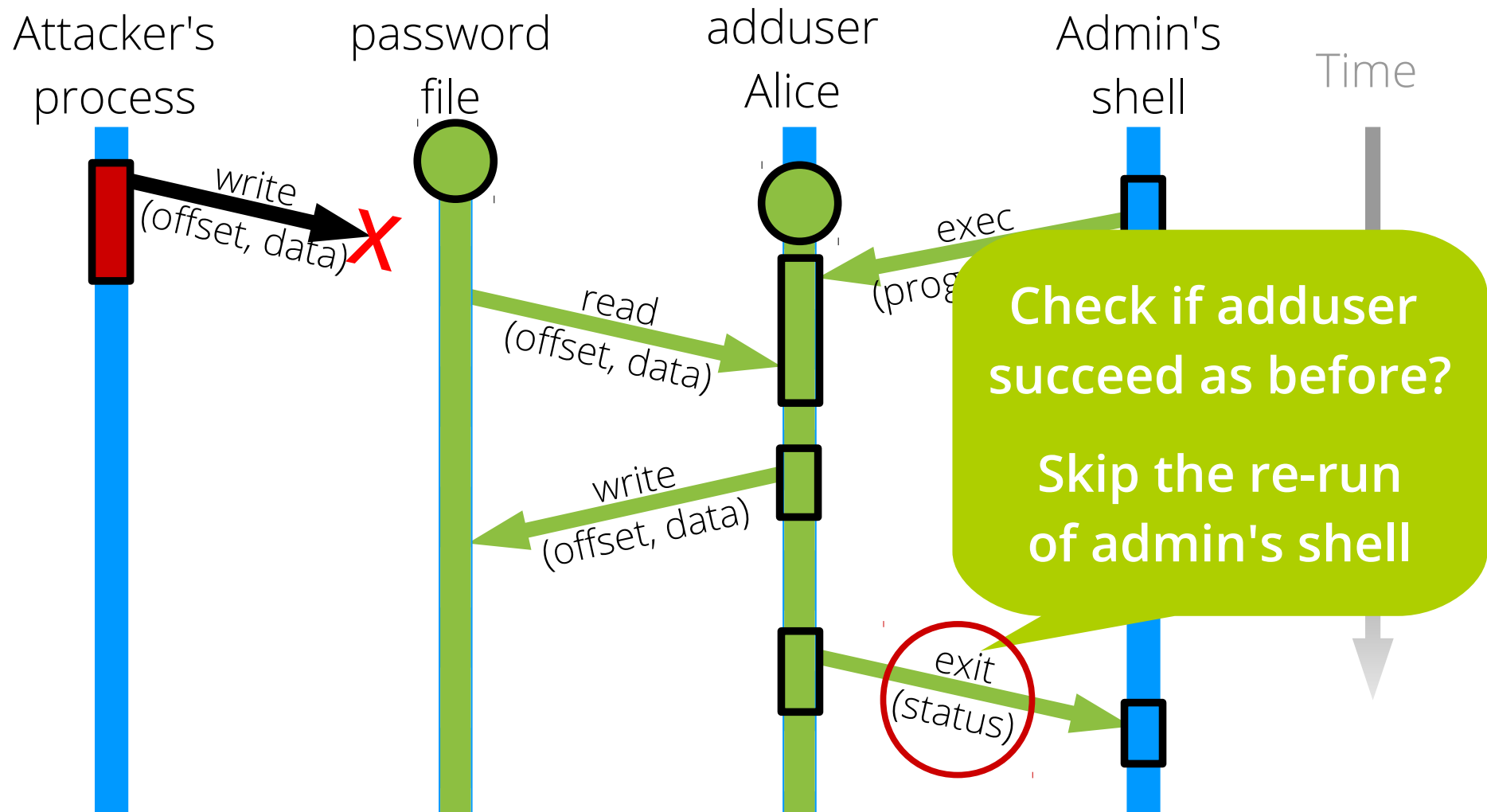
- **“Adduser alice” succeed as before**
 - This is what Admin wanted to do
 - If failed, need to re-execute Admin's shell

Example 1: exit status to shell unchanged

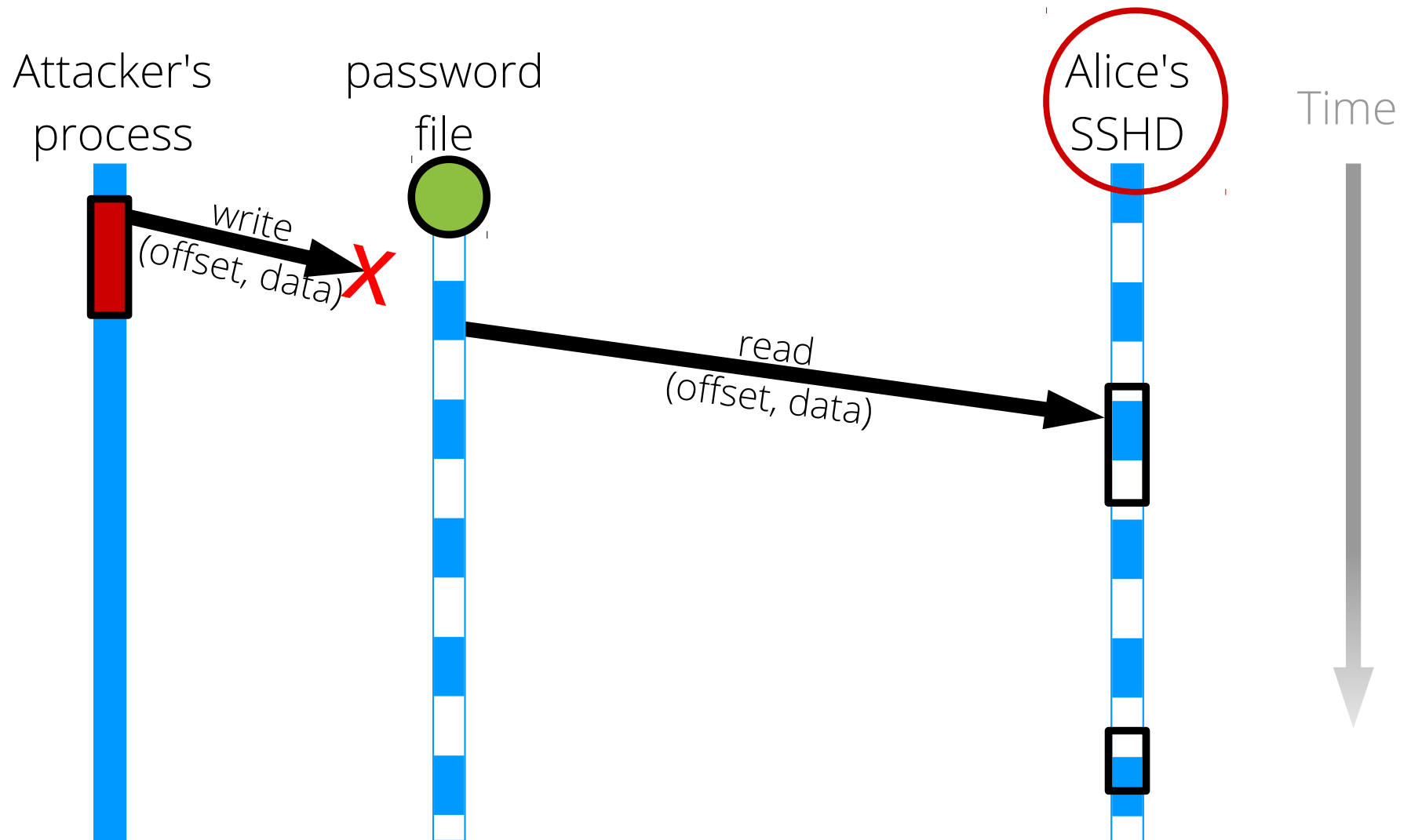


Predicates:

avoid equivalent re-execution



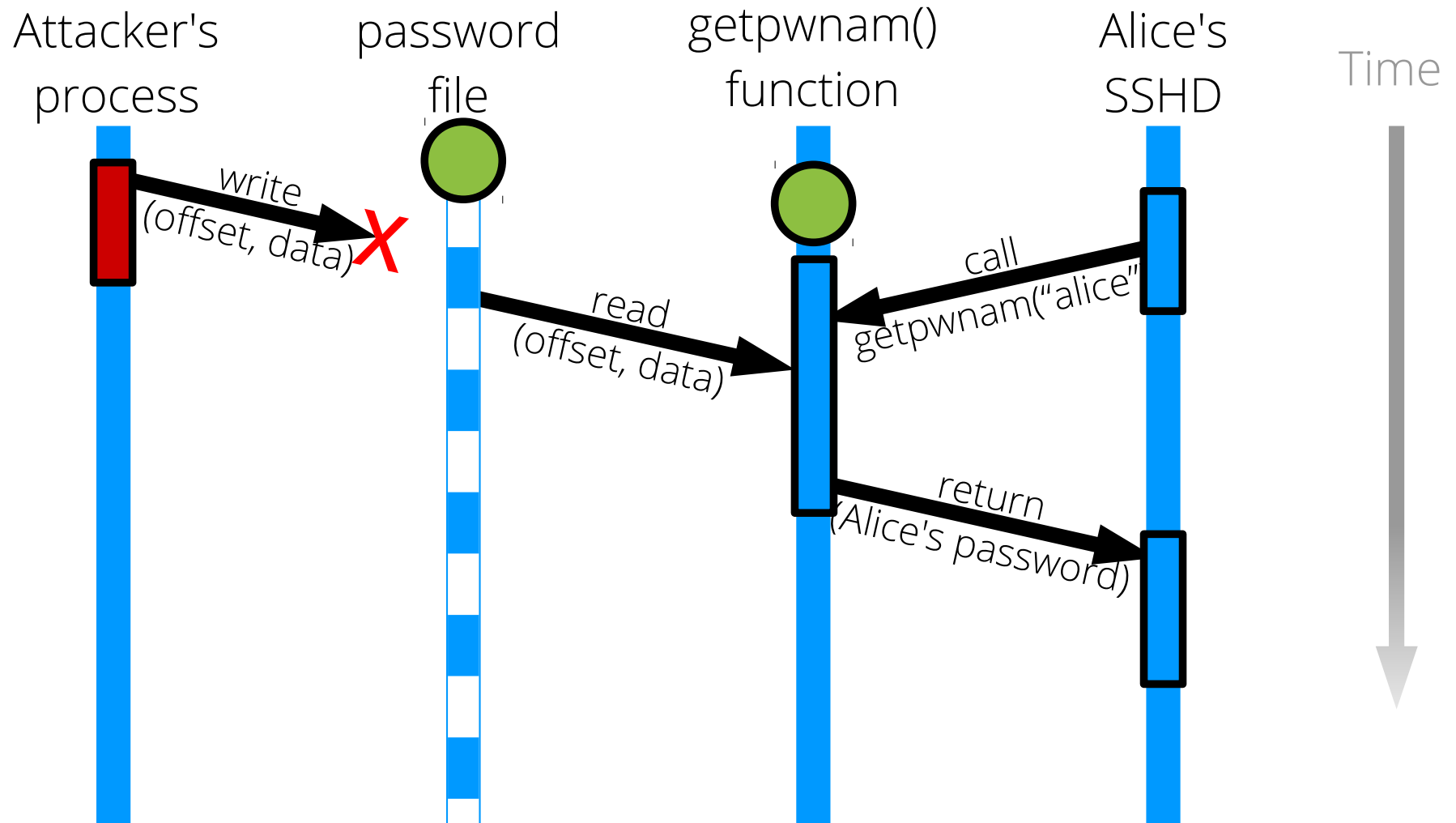
Example 2: user's password unchanged



Observation: Alice's SSHD was not affected

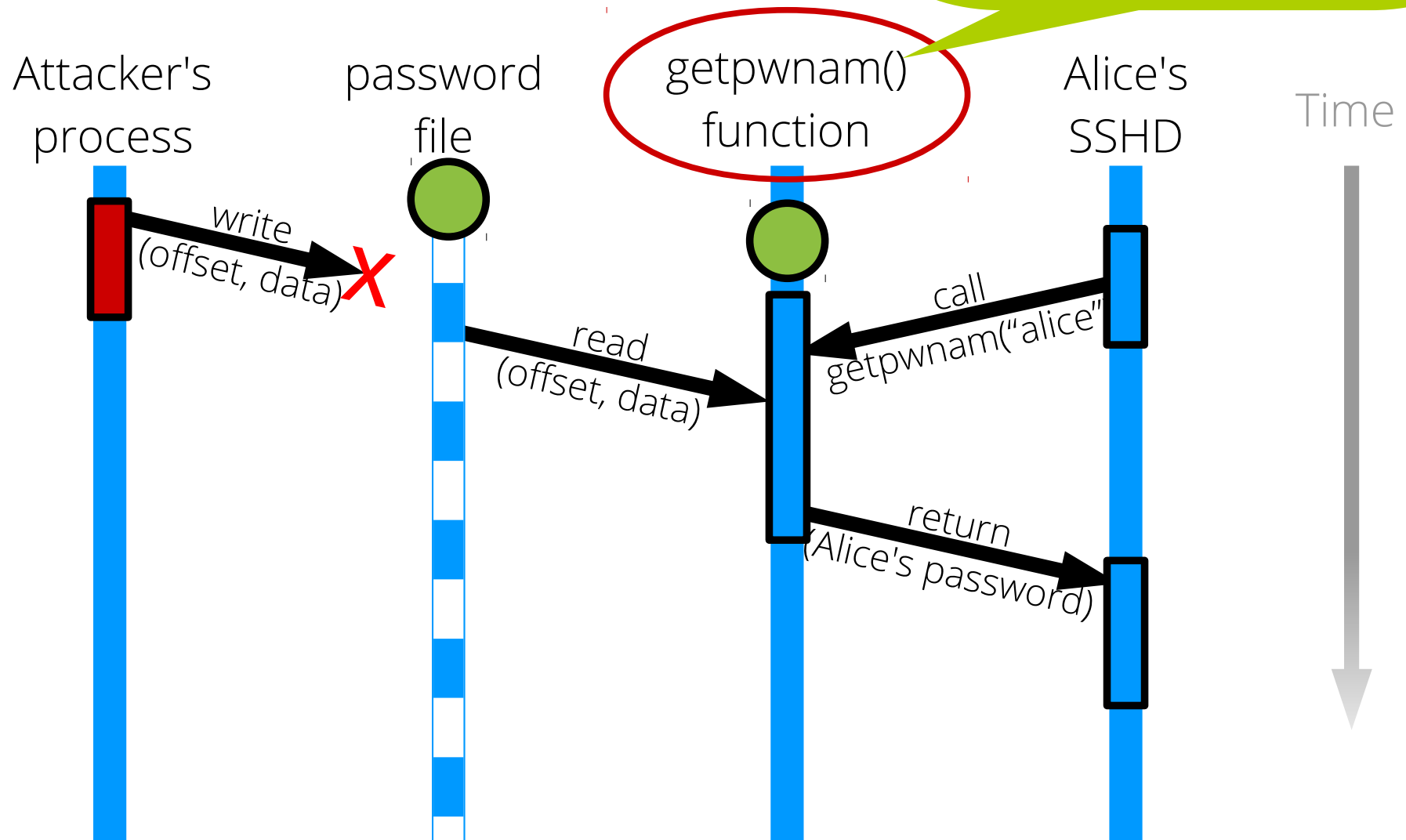
- **Alice's SSHD checked only Alice's account**
 - This is what Alice's SSHD wanted to do
 - If Alice's account changed, need to re-execute SSHD

Refinement: exploits high-level semantics

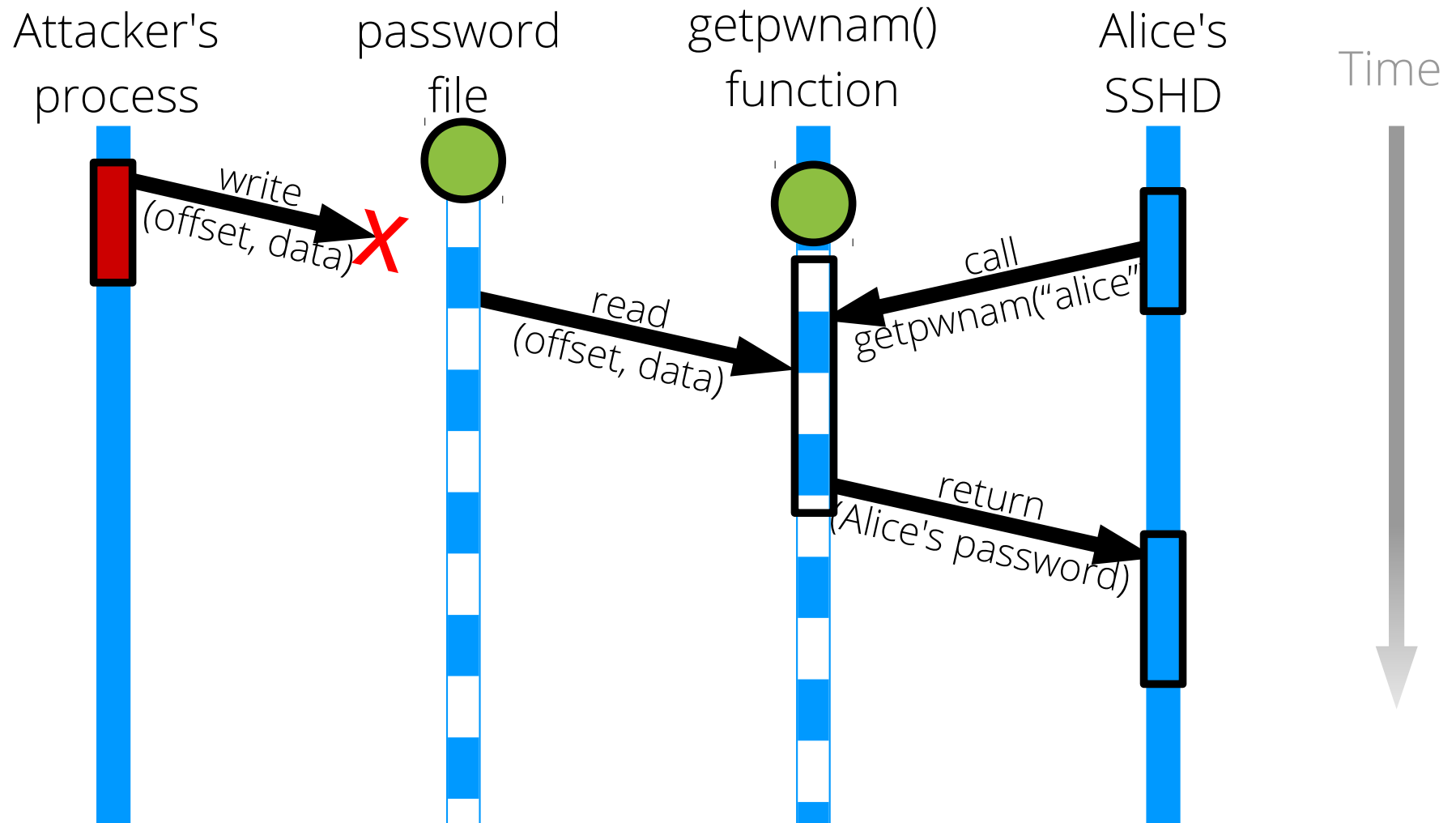


Refinement: exploits high-level se

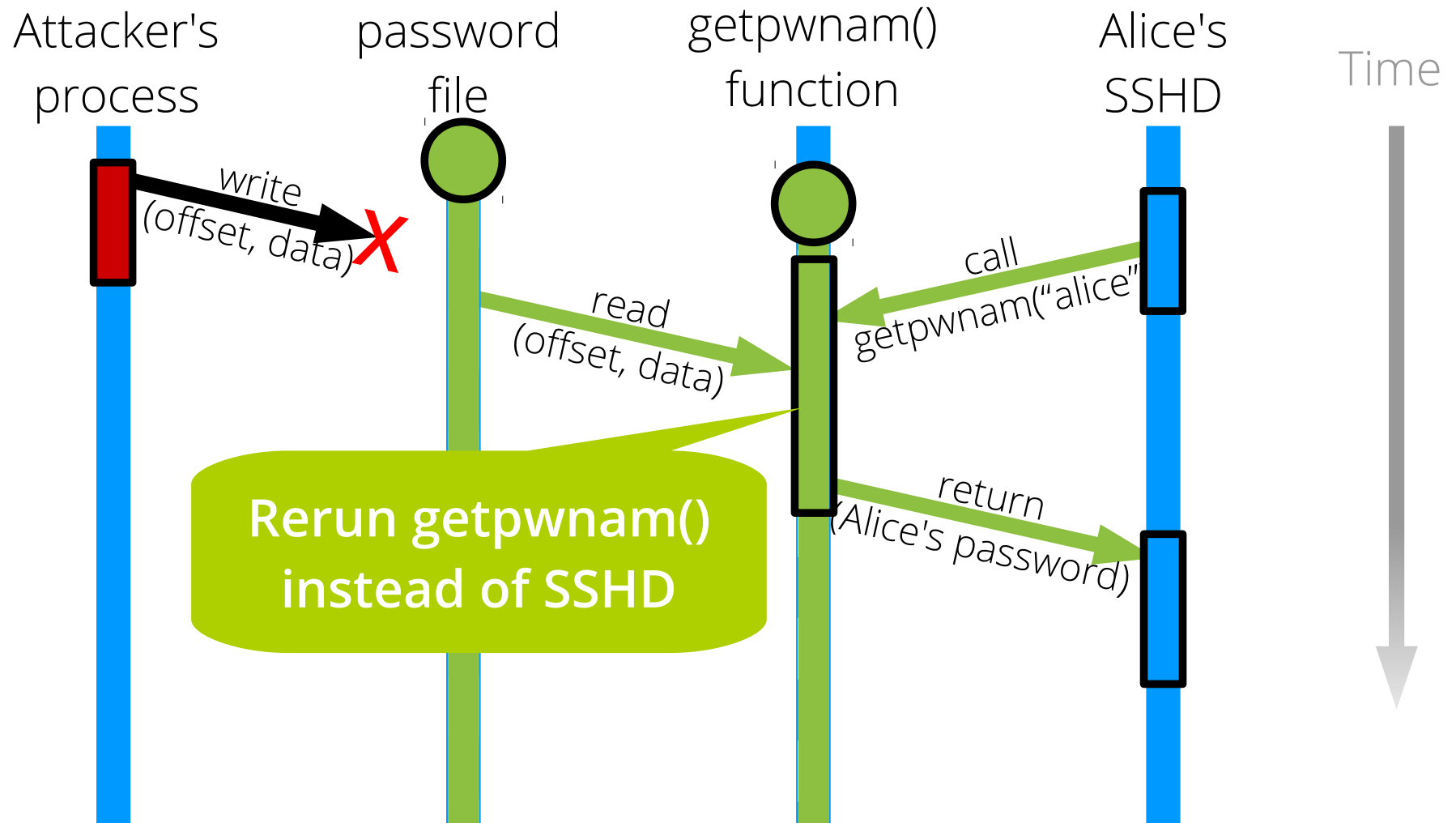
Get username,
return passwd entry



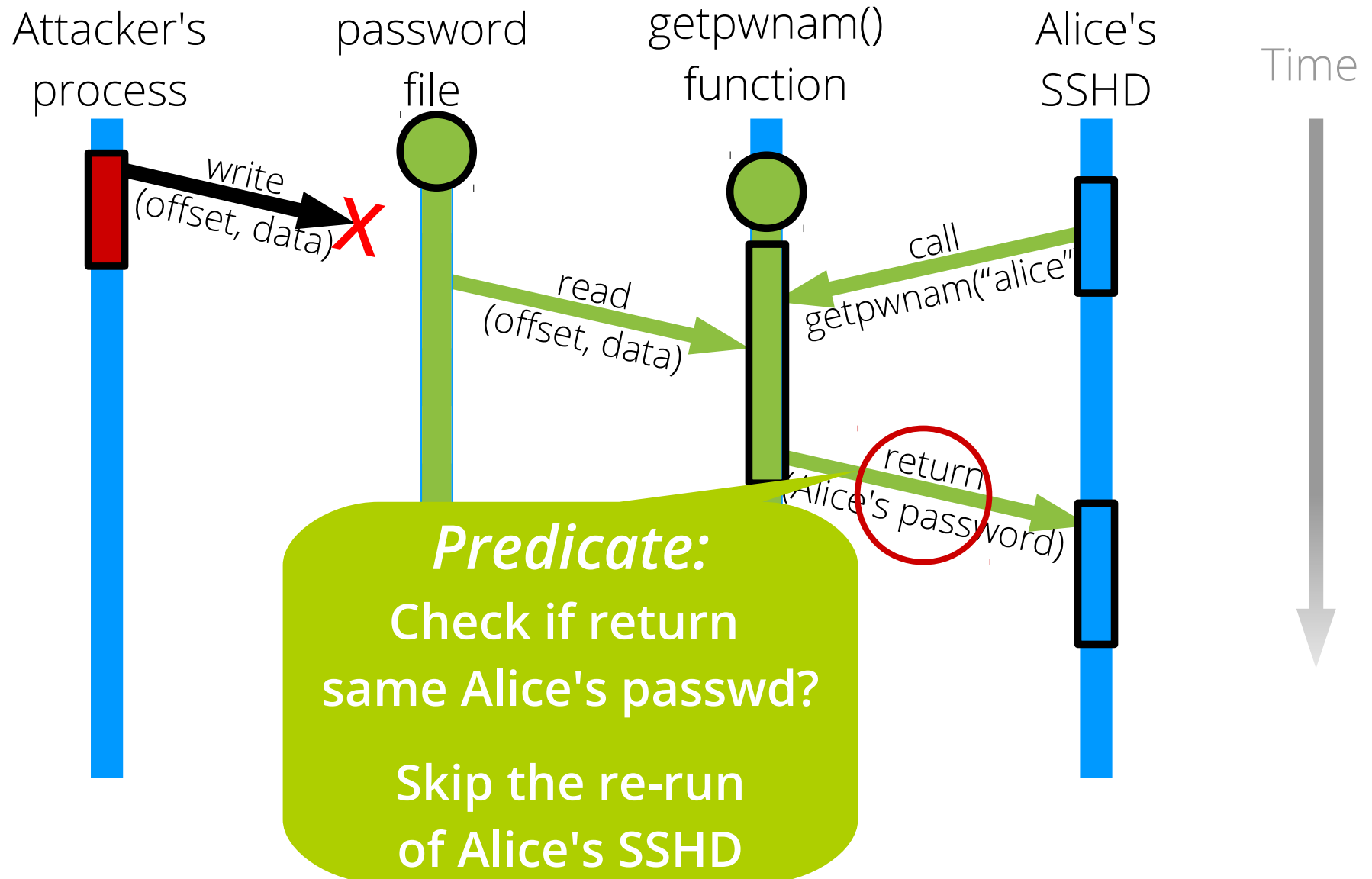
Refinement: exploits high-level semantics



Refinement: exploits high-level semantics



Refinement: exploits high-level semantics



Quick summary:

Retro's approach

- ***Action history graph***: represent history in detail
- **Two techniques** to minimize re-execution:
 - ***Predicates***: skips equivalent computations
 - ***Refinement***: re-executes fine-grained actions

Challenge: external dependencies

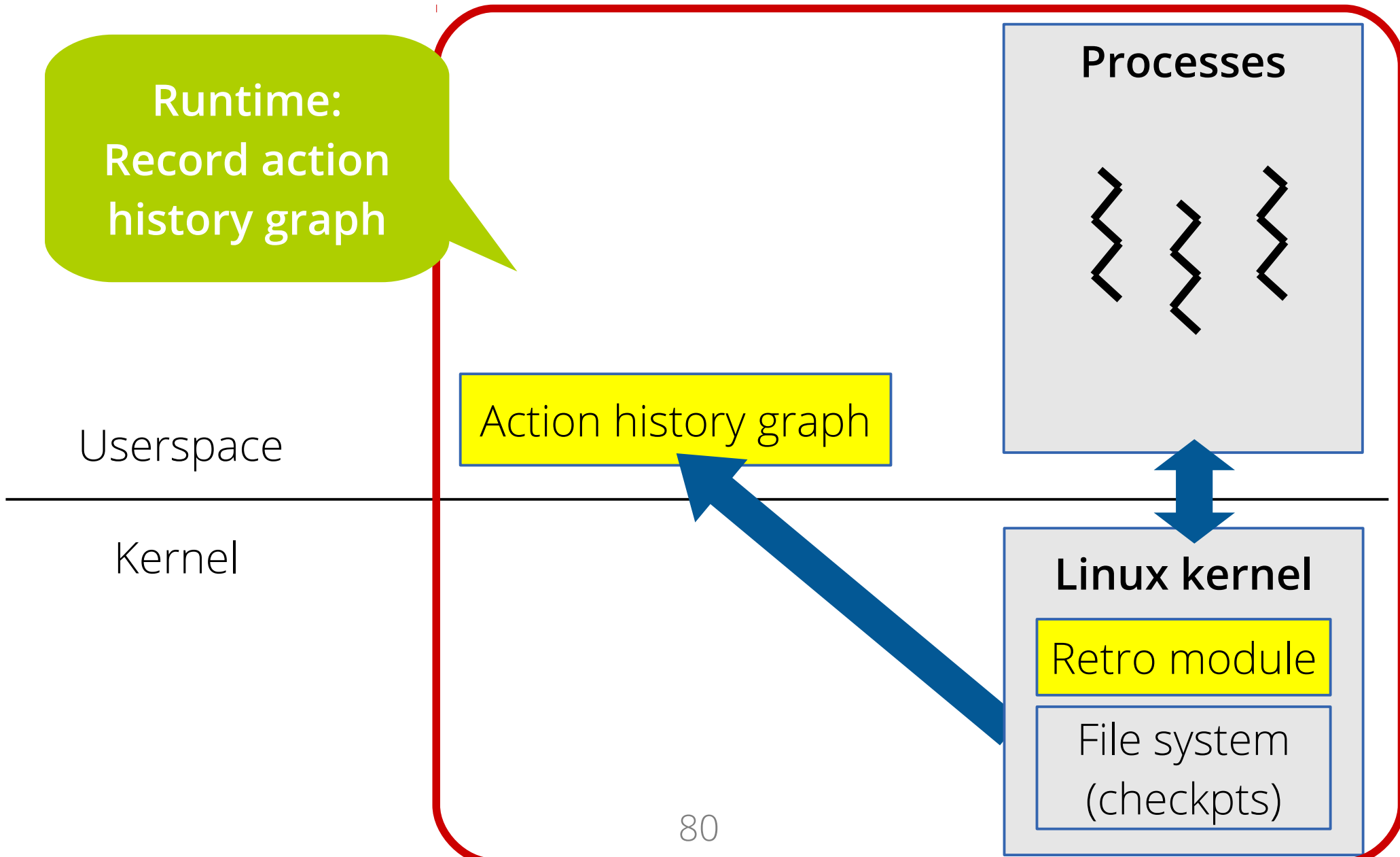
- **What if the attack was externally-visible?**
 - Spam sent out ...
 - Hard in general case → ask for user's decision
- **Help users to understand repaired state**
 - (e.g.) notify user spam email was sent out ...

Compensating action: notify changes in terminal output

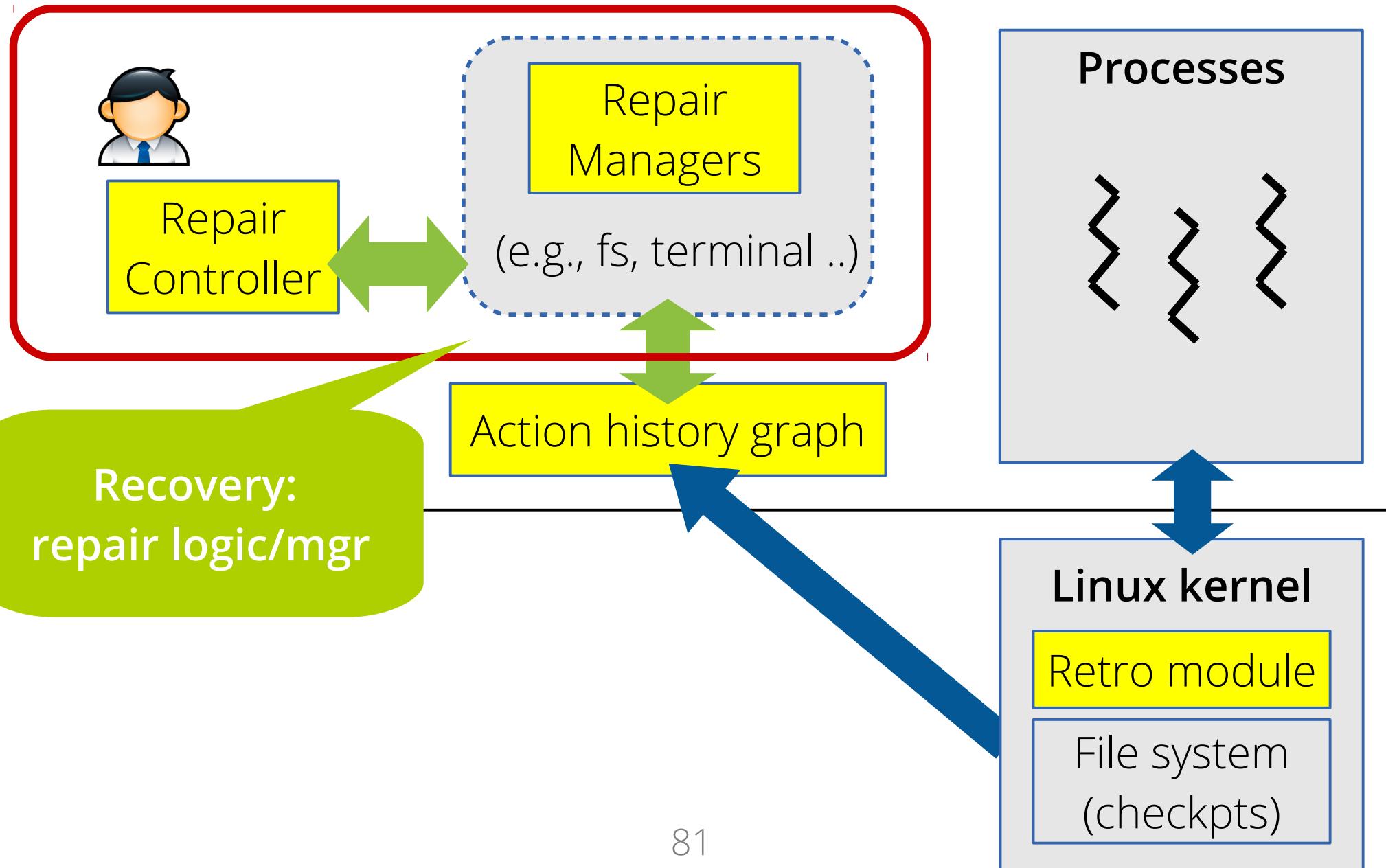
```
...  
[redo] cat ~/.ssh/authorized_keys  
...  
! --- old  
! +++ new  
! @@ -1,3 +1,2 @@  
! ssh-rsa AAAAB3NzaC1yc2EAAAABIw... vagrant  
! -ssh-rsa AAAAB3NzaC1yc2EAAAADAQ... attacker  
! ssh-rsa AAAAB3NzaC1yc2EAAAADAQ... new pubkey  
...
```

You should not have
seen this output!

Retro implementation



Retro implementation



Application specific mgrs
using well-defined API

Implementation



Repair
Controller

Repair
Managers

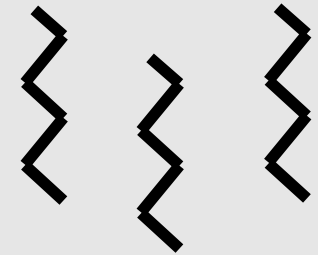
(e.g., fs, terminal ..)

Action history graph

Userspace

Kernel

Processes



Linux kernel

Retro module

File system
(checkpts)

Demo: recovering from inadvertently installed virus

- *Backtracking tool*
- *Selective re-execution*
- *Compensating action*

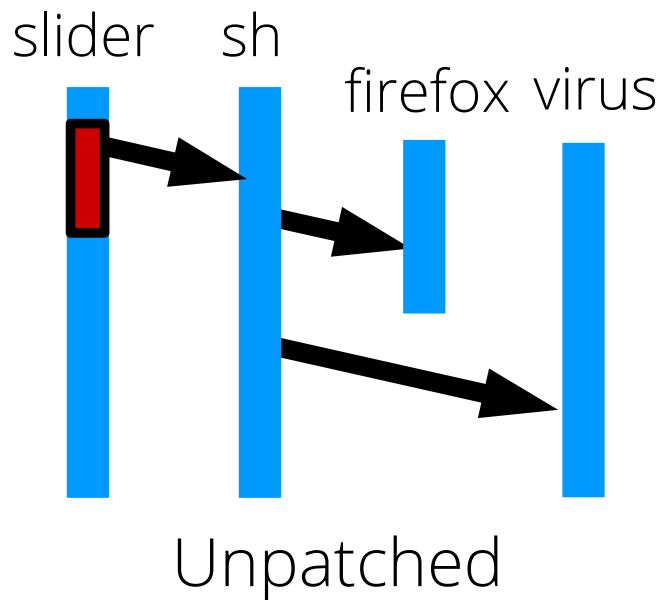
Problem: detecting an entry point of attacks is hard

- How to find one-month-old attack?
- Too much information
 - Manual analysis is time-consuming

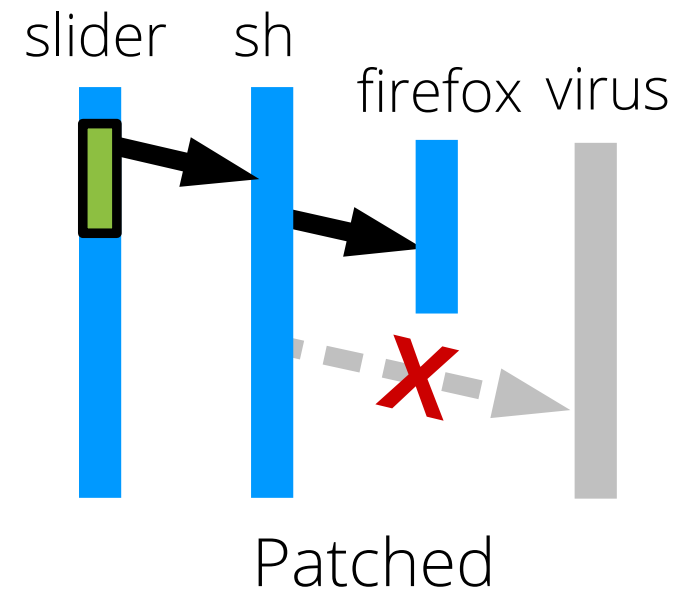
Observation: security patch renders attack harmless

- Escape URL arguments for firefox

```
// slider.c  
- sprintf(cmd, "firefox %s", evt->uri);  
+ sprintf(cmd, "firefox %s", escape(evt->uri));
```

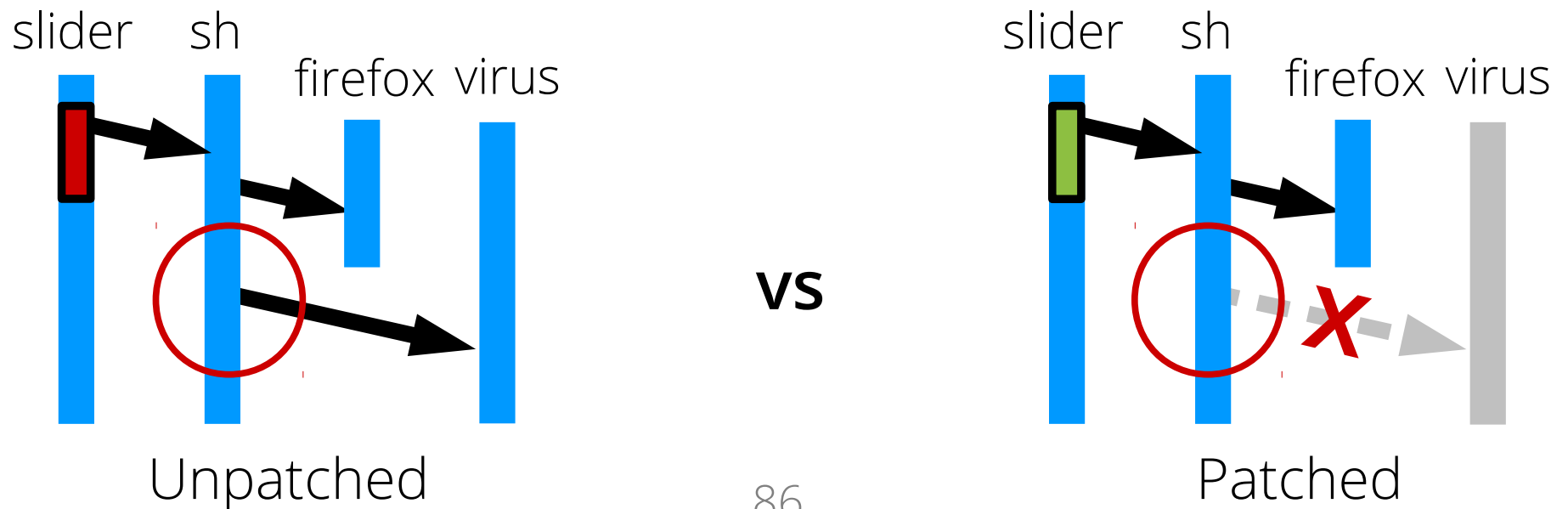


vs



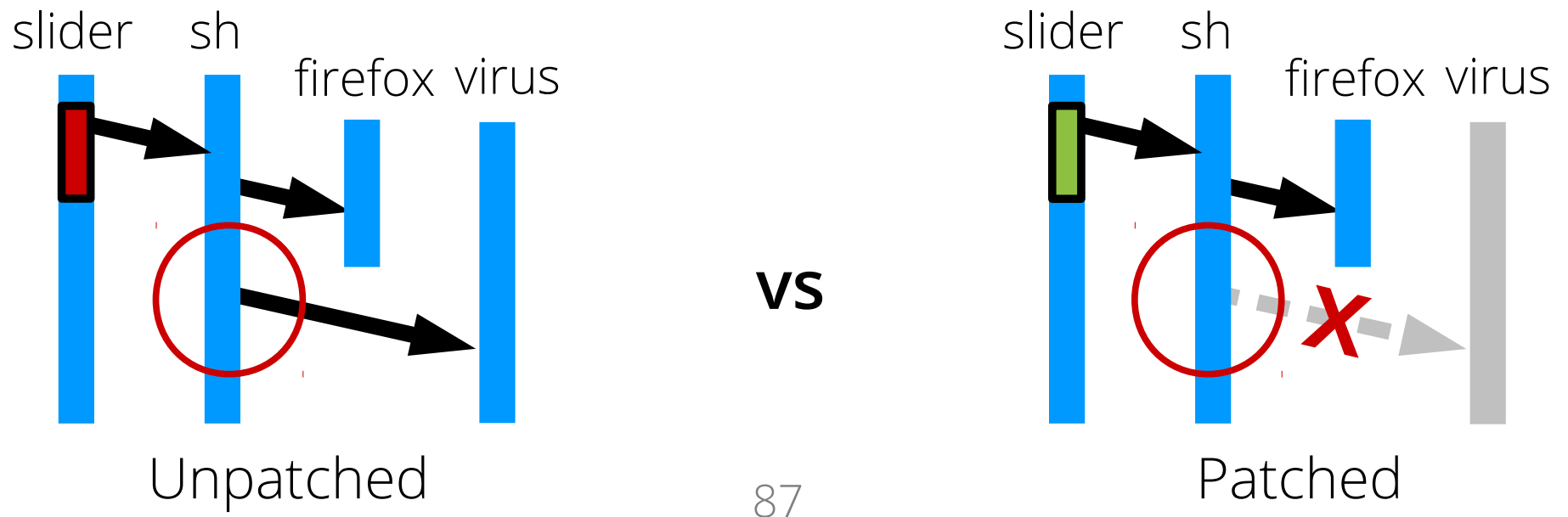
Approach: comparing both histories to detect past attacks

- How can we get history of patched execution?
 - Replay inputs after applying security patches
 - Different history → potential threats



Approach: comparing both histories to detect past attacks

Turn manual effort of auditing process
into computational problem!
(*patch-based auditing*)



Challenge: performance

- **Re-executing is costly for busy computer**
 - Auditing requests → re-executes all requests again
 - Auditing one month → takes another month!

Three techniques developed for partial re-execution

- ***Control flow filtering***
 - Audit possibly affected executions
- ***Function-level auditing***
 - Compare function-level executions
- ***Memoized re-execution***
 - Avoid duplicated executions while replaying

Putting all together: fixing our past & future with patch



Patch from upstream
(fixing a bug in SSHD)



3. No guarantees
(safe to rollback?)

2. Lost changes
(a month!)

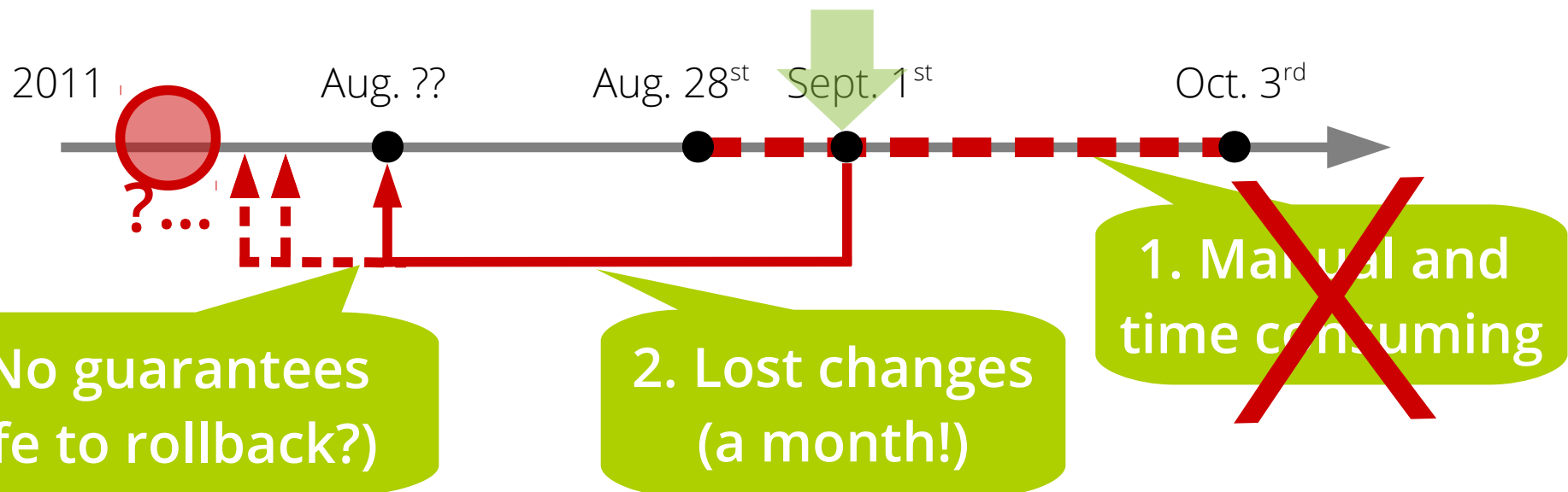
1. Manual and
time consuming

Putting all together: fixing our past & future with patch

- Automatic detection

+++

Patch from upstream
(fixing a bug in SSHD)

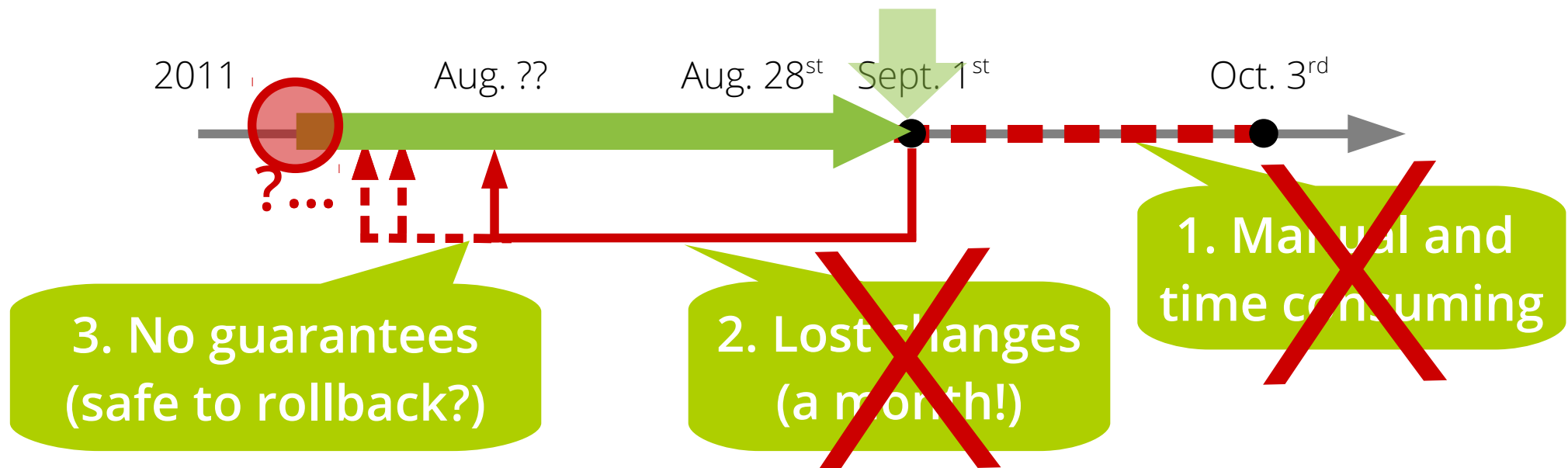


Putting all together: fixing our past & future with patch



Patch from upstream
(fixing a bug in SSHD)

- Automatic detection
- Preserve changes

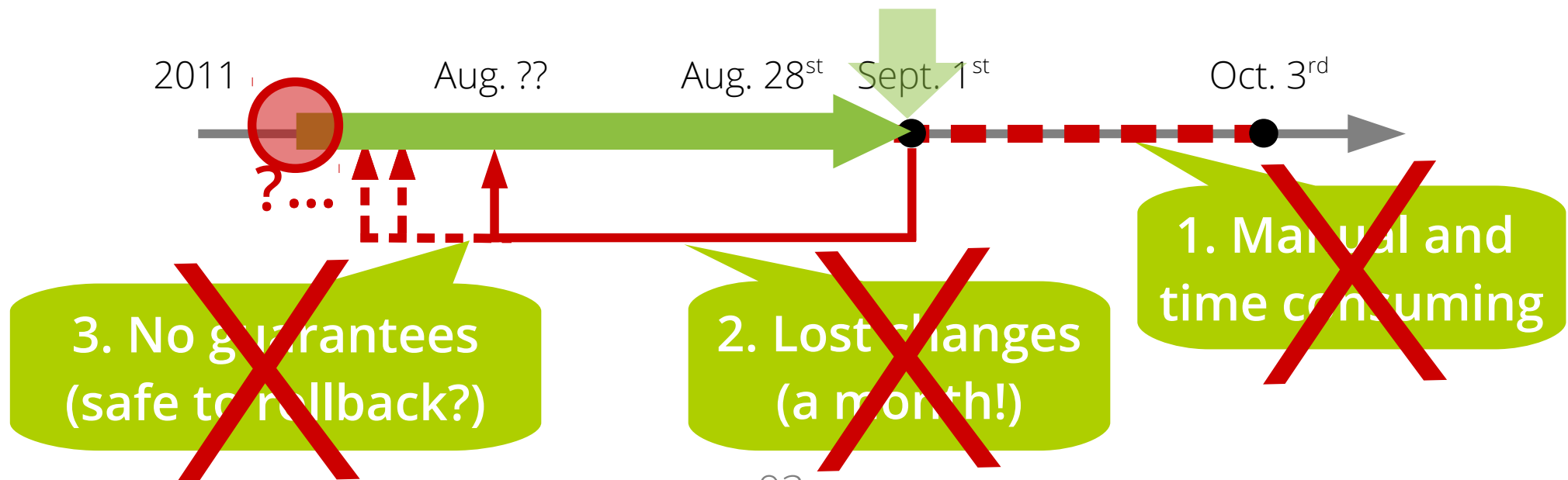


Putting all together: fixing our past & future with patch

+++

Patch from upstream
(fixing a bug in SSHD)

- Automatic detection
- Preserve changes
- Strong guarantees



Putting all together: fixing our past & future with patch



Patch from upstream
(fixing a bug in SSHD)

- Automatic detection
- Preserve changes
- Strong guarantees

Whenever new patches are released,
not only prevent *future* attacks,
but also detect and repair *past* attacks *for free*!

Summary of our approach: building real systems

- **Existing systems are not designed for history**
 - Implicit dependencies and time-line
- **Attacks can be anywhere in the history**
 - Attacks are often detected days or weeks later
- **History can not be changed in some cases**
 - External dependencies: spam sent out

Summary of our approach: building real systems

- **Existing systems are not designed for history**
 - Action history graph & re-execution techniques
- **Attacks can be anywhere in the history**
 - Attacks are often detected days or weeks later
- **History can not be changed in some cases**
 - External dependencies: spam sent out

Summary of our approach: building real systems

- **Existing systems are not designed for history**
 - Action history graph & re-execution techniques
- **Attacks can be anywhere in the history**
 - Patch-based auditing
- **History can not be changed in some cases**
 - External dependencies: spam sent out

Summary of our approach: building real systems

- **Existing systems are not designed for history**

→ Action history graph & re-execution techniques

- **Attacks can be anywhere in the history**

→ Patch-based auditing

- **History can not be changed in some cases**

→ (Not solved) compensating actions in some cases
(see our recent work, Aire [SOSP'13] in this direction of research)

Evaluation questions

- **Automatic intrusion recovery**
 - How much better than manual repair?
 - How much runtime overhead?
- **Patch-based auditing**
 - What attacks can be detected?
 - How fast is re-execution?

Experimental setup for Retro (automatic recovery)

- 2.8 GHz Intel Core i7, 8 GB RAM
- 64-bit Linux 2.6.35
- Tested with
 - 2 real-world attacks from Honeypot
 - 8 synthetic attacks

Retro recovers from real-world and synthetic attacks

- 2 real-world attacks from Honeypot
 - Remove log entries, add accounts, run botnet
- 8 synthetic attacks
 - 2 examples: LaTeX and SSHD trojan
 - 6 scenario: File sharing, Web servers ...

Retro's runtime overheads in realistic workloads

Workload	CPU cost	Storage overhead
HotCRP conference web site	35%	4GB / day

Retro's runtime overheads in challenging workloads

Workload	CPU cost	Storage overhead
HotCRP conference web site	35%	4GB / day
Apache, small static files	127%	100GB / day
Continuous kernel recompile	89%	150GB / day

- Can store 2 weeks of logs on 2TB disk (\$100) even for worst-case workloads

Retro imposes acceptable overheads in practice

Workload	CPU cost	w/ 2 nd core	Storage overhead
HotCRP conference web site	35%	2%	4GB / day
Apache, small static files	127%	33%	100GB / day
Continuous kernel recompile	89%	18%	150GB / day

- Can store 2 weeks of logs on 2TB disk (\$100) even for worst-case workloads
- Can off-load CPU overhead to dedicated core

Retro imposes acceptable overheads in practice

Workload	CPU cost	w/ 2 nd core	Storage overhead
HotCRP conference web site	35%	2%	4GB / day
Apache, small static files	127%	33%	100GB / day
Continuous kernel recompile	89%	18%	150GB / day

**For systems where recovery is critical,
Retro's overheads can be acceptable**

Experimental setup for Poirot (patch-based auditing)

- 3.07 GHz Core i7-950, 12GB RAM
- PHP 5.3.6
- No application changes required
- Tested with
 - Security patches in Wikipedia and HotCRP
 - Under real Wikipedia traces

Poirot efficiently audits attacks

- **34 real patches** in Wikipedia
- Auditing **3.4h** of executions
 - **29 patches** → **<0.2 sec** (rarely executed code)
 - **5 patches** → **~9.2 min** (commonly executed code)

**Poirot can re-execute 12-51x faster
than the original execution
even for worst-case patches**

Poirot detects real attacks

- **Wikipedia**: detected **5 different types** of attacks
(e.g., Stored XSS, CSRF ...)
- **HotCRP**: detected **4 info. leak** vulnerabilities
(e.g., accepted papers ...)

Poirot imposes reasonable runtime overheads

- Testing with real Wikipedia traces
 - **14.1% latency** overhead
 - **15.3% throughput** overhead
 - **5.4 KB/req** storage overhead

For systems where integrity is critical,
Poirot's overheads can be acceptable

Related work

- **Tracking down attacks:** BackTracker, IntroVirt
 - Not for recovery, but only for analyzing attacks
- **Taint tracking for recovery:** Taser, Polygraph
 - False positives: recovering too conservatively
- **Selective undo/redo:** Undoable mail store
 - Fixing configuration errors in email server

Today's talk

- **Automatic recovery**
 - Operating system: **Retro** [OSDI'10]
 - Web application: **Warp** [SOSP'11]
 - Distributed web services: **Aire** [SOSP'13]
- **Automatic detection of attacks**
 - Web application: **Poirot** [OSDI'12]
- **Future research agenda**

Research agenda

Idea: use history for everything

① Undoable OS

Can undoability be part of our daily computing life?

- New design of components / interfaces in OS
- Usable / intuitive user interface

Research agenda

Idea: protect history from adversaries

② Haskell Kernel

Can kernel be secure by design?

- Track and keep history safe?
- Purely functional → better undo/redo-ability

Research agenda

Idea: connect history of all computers

③ **Security Analytics**

Can we understand security for larger systems?

- Better understand security with concrete histories
- Leverage recent tools for Big Data

Summary: building secure systems with *system-wide history*

- Big step toward “*undo computing*”
- Automatic recovery
 - Operating system: **Retro** [OSDI'10]
 - Web application: **Warp** [SOSP'11]
 - Distributed web services: **Aire** [SOSP'13]
- Automatic detection of attacks
 - Web application: **Poirot** [OSDI'12]

Summary: building secure systems with *system-wide history*

- Big step toward *“undo computing”*

Thank you!

Work in collaboration with:

Ramesh Chandra, Meelap Shah, Neha Narula, Xi Wang,
Nickolai Zeldovich, M. Frans Kaashoek