Practical and Effective Sandboxing for Non-root users

Taesoo Kim and Nickolai Zeldovich

MIT CSAIL

Why yet another sandbox for desktop applications?

- There are many existing sandbox mechanisms
  - Chroot / Lxc (Unix/Linux)
  - Jail (Freebsd)
  - Seatbelt (Mac OS X)
  - VM?

    …
- Difficult-to-use, requiring root privilege, or slow!

TL;DR

Our tool

Unknown binary downloaded from the Internet

```
$ mbox -- ./downloaded-bin
...
Network Summary:
 > [11279] -> 173.194.43.51:80
 > [11279] Create socket(PF_INET,...)
 > [11279] -> a00::2607:f8b0:4006:803:0
 ...
Sandbox Root:
 > /tmp/sandbox-11275
 >   N:/tmp/index.html
[c]ommit, [d]iff, [i]gnore, [l]ist, [s]hell, [q]uit ?>
```

TL;DR

```
$ mbox -- ./downloaded-bin
...
Network Summary:
 > [11279] -> 173.194.43.51:80
 > [11279] Create socket(PF_INET,...)
 > [11279] -> a00::2607:f8b0:4006:803:0
 ...
Sandbox Root:
 > /tmp/sandbox-11275
 >   N:/tmp/index.html
[c]ommit, [d]iff, [i]gnore, [l]ist, [s]hell, [q]uit ?>
```

Where to connect?

TL;DR

```
$ mbox -- ./downloaded-bin
...
Network Summary:
 > [11279] -> 173.194.43.51:80
 > [11279] Create socket(PF_INET,...)
 > [11279] -> a00::2607:f8b0:4006:803:0
 ...
Sandbox Root:
 > /tmp/sandbox-11275
 >   N:/tmp/index.html
[c]ommit, [d]iff, [i]gnore, [l]ist, [s]hell, [q]uit ?>
```

Protecting the host filesystem from modification

TL;DR

```
$ mbox -- ./downloaded-bin
...
Network Summary:
 > [11279] -> 173.194.43.51:80
 > [11279] Create socket(PF_INET,...)
 > [11279] -> a00::2607:f8b0:4006:803:0
 ...
Sandbox Root:
 > /tmp/sandbox-11275
 >   N:/tmp/index.html
[c]ommit, [d]iff, [i]gnore, [l]ist, [s]hell, [q]uit ?>
```

TL;DR

Without root privilege!

```
$ mbox -- ./downloaded-bin
...
Network Summary:
 > [11279] -> 173.194.43.51:80
 > [11279] Create socket(PF_INET,...)
 > [11279] -> a00::2607:f8b0:4006:803:0
 ...
Sandbox Root:
 > /tmp/sandbox-11275
 >   N:/tmp/index.html
[c]ommit, [d]iff, [i]gnore, [l]ist, [s]hell, [q]uit ?>
```

Design overview

- **Layered sandbox filesystem**
  - Overlaying the host filesystem
  - Confining modification made by sandboxed processes
  - Persistent storage: in fact, just a regular directory

- **System call interposition**
  - Commodity OSes provide one for non-root users
  - Enabling a variety of applications: installing pkgs, restricting network, build/dev. env ...

Design overview

- **Layered sandbox filesystem**
  - Overlaying the host filesystem
  - Confining modification made by sandboxed processes
  - Persistent storage: in fact, just a regular directory

- **System call interposition**
  - Commodity OSes provide one for non-root users
  - Enabling a variety of applications: installing pkgs, restricting network, build/dev. env …

Installing packages as normal user

```
$ mbox -R -- apt-get install git
(-R: emulate a fakeroot environment)
```

- Mbox provides a writable sandbox layer on top of the host filesystem
  - User owns the sandbox directory
  - Contain newly installed files, and package databases
- Mbox emulates a fakeroot environment
  - Use standard package managers without modification
  - Support: apt-get (Ubuntu), dpkg (Debian), pip (Python)

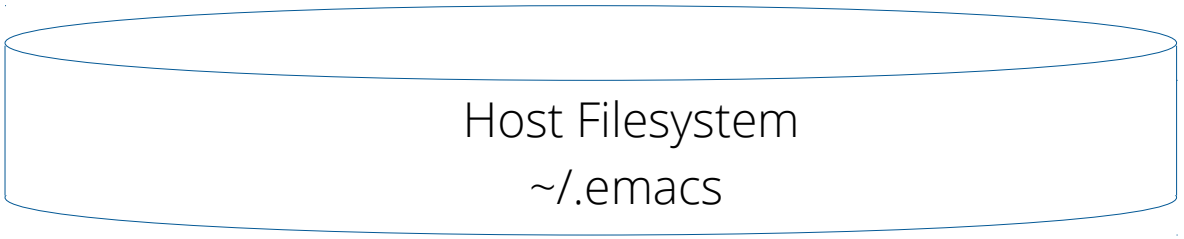
Running unknown binary safely

```
$ mbox -n -- ./downloaded-bin
(-n: disable remote network accesses)
```

- Mbox protects the host filesystem from modifications
- Mbox restricts or monitors network accesses
  - Interpret socket-like system calls
  - Summarize network activity when terminated

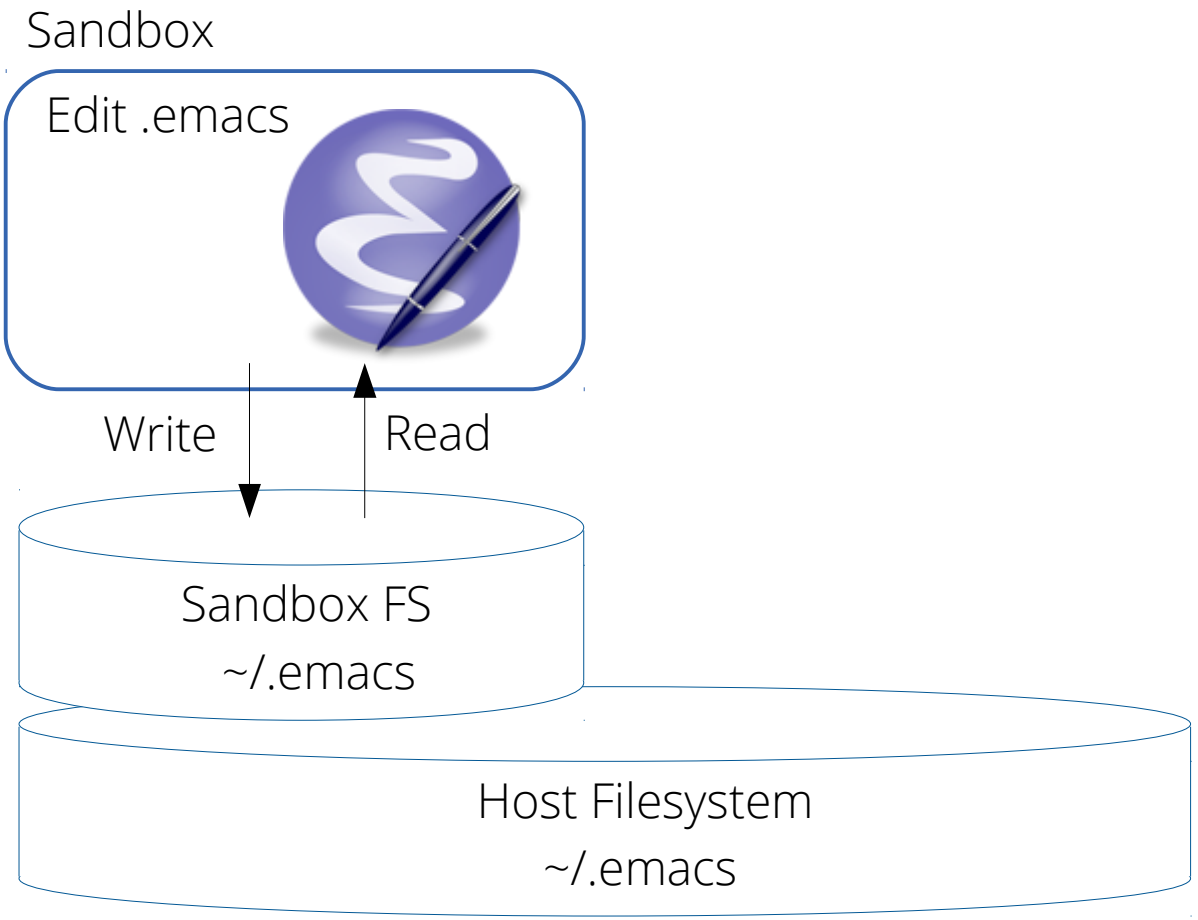
Checkpointing filesystem

```
$ mbox -i -- emacs ~/.emacs
(-i: enable interactive commit-mode)
```



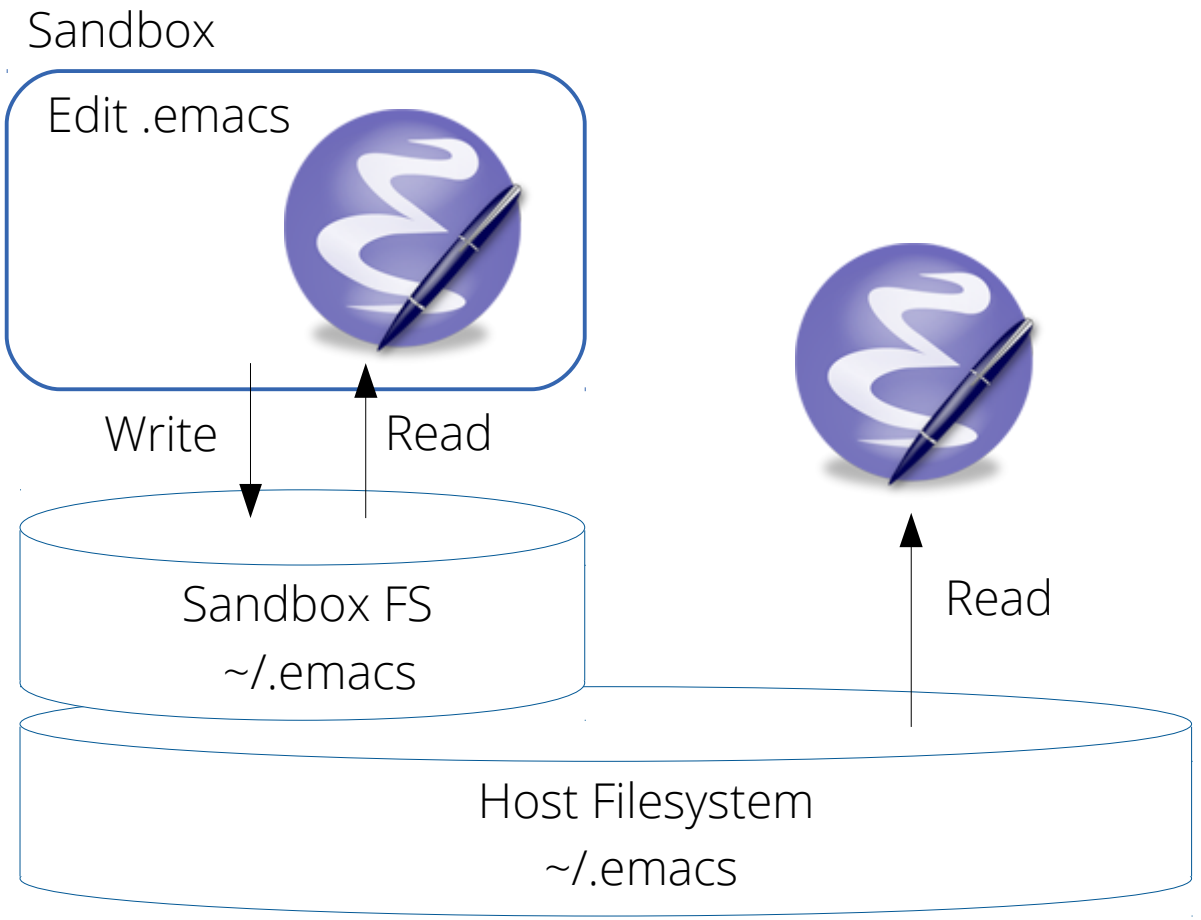
Checkpointing filesystem

```
$ mbox -i -- emacs ~/.emacs
(-i: enable interactive commit-mode)
```



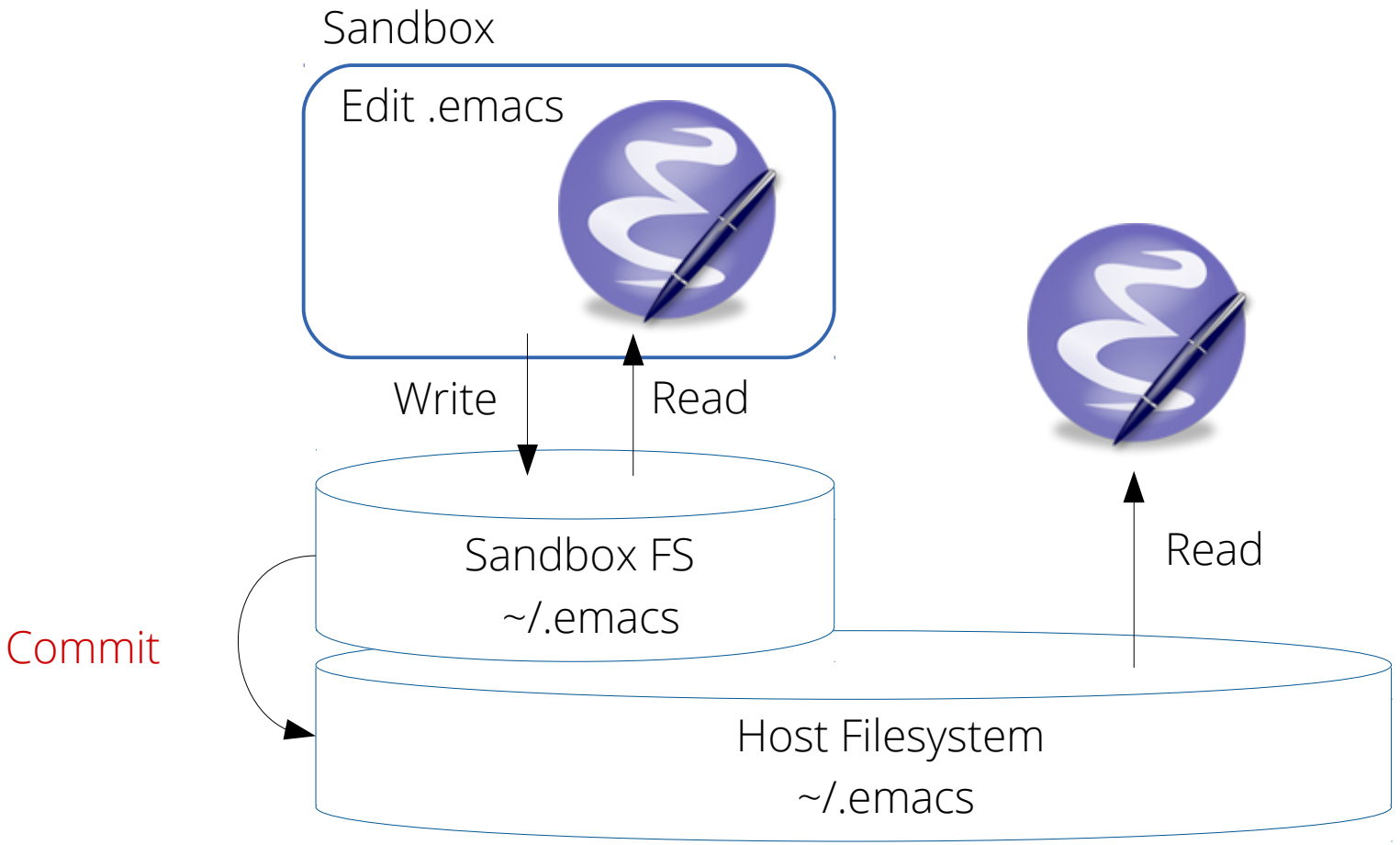
Checkpointing filesystem

```
$ mbox -i -- emacs ~/.emacs
(-i: enable interactive commit-mode)
```



Checkpointing filesystem

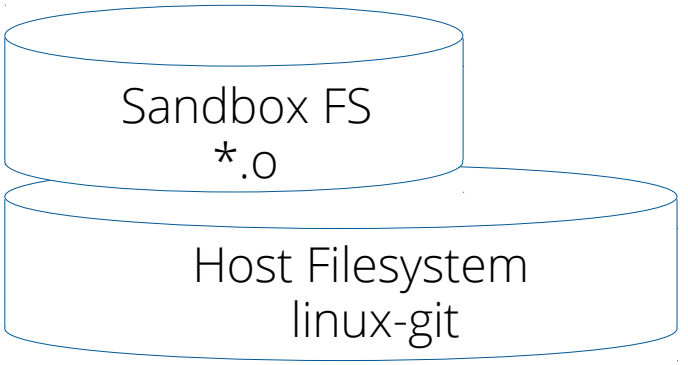
```
$ mbox -i -- emacs ~/.emacs
(-i: enable interactive commit-mode)
```



Build/development environment

```
$ tree linux-git
  ...
  +--mm--mmap.c
       +-mlock.c

$ mbox -r outdir -- make
(-r dir: specify a sandbox directory)
```

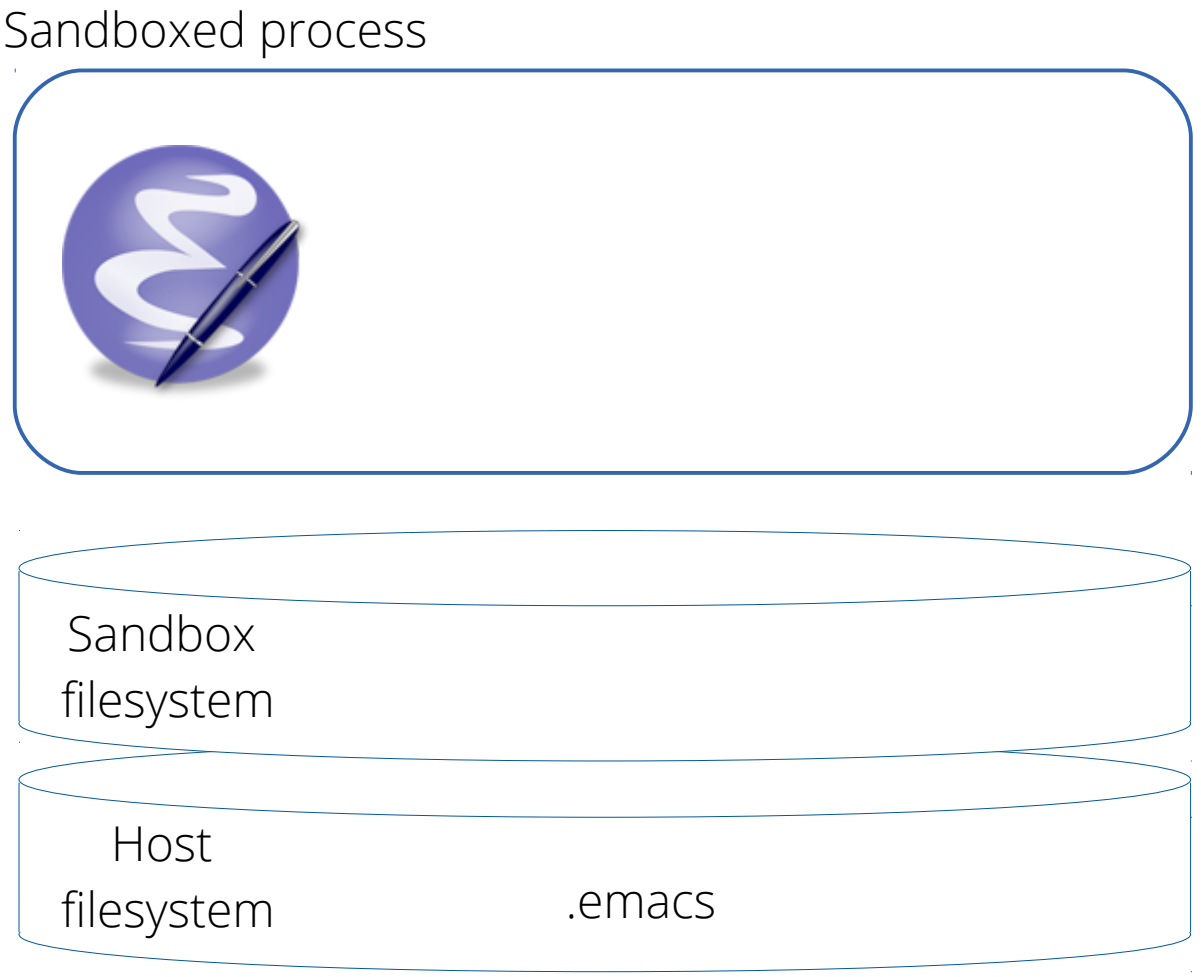


- Mbox can separate out the generated obj files
  - make clean == rm -rf outdir
- Mbox can also be used for virtual dev. env.
  - Install packages with standard package managers

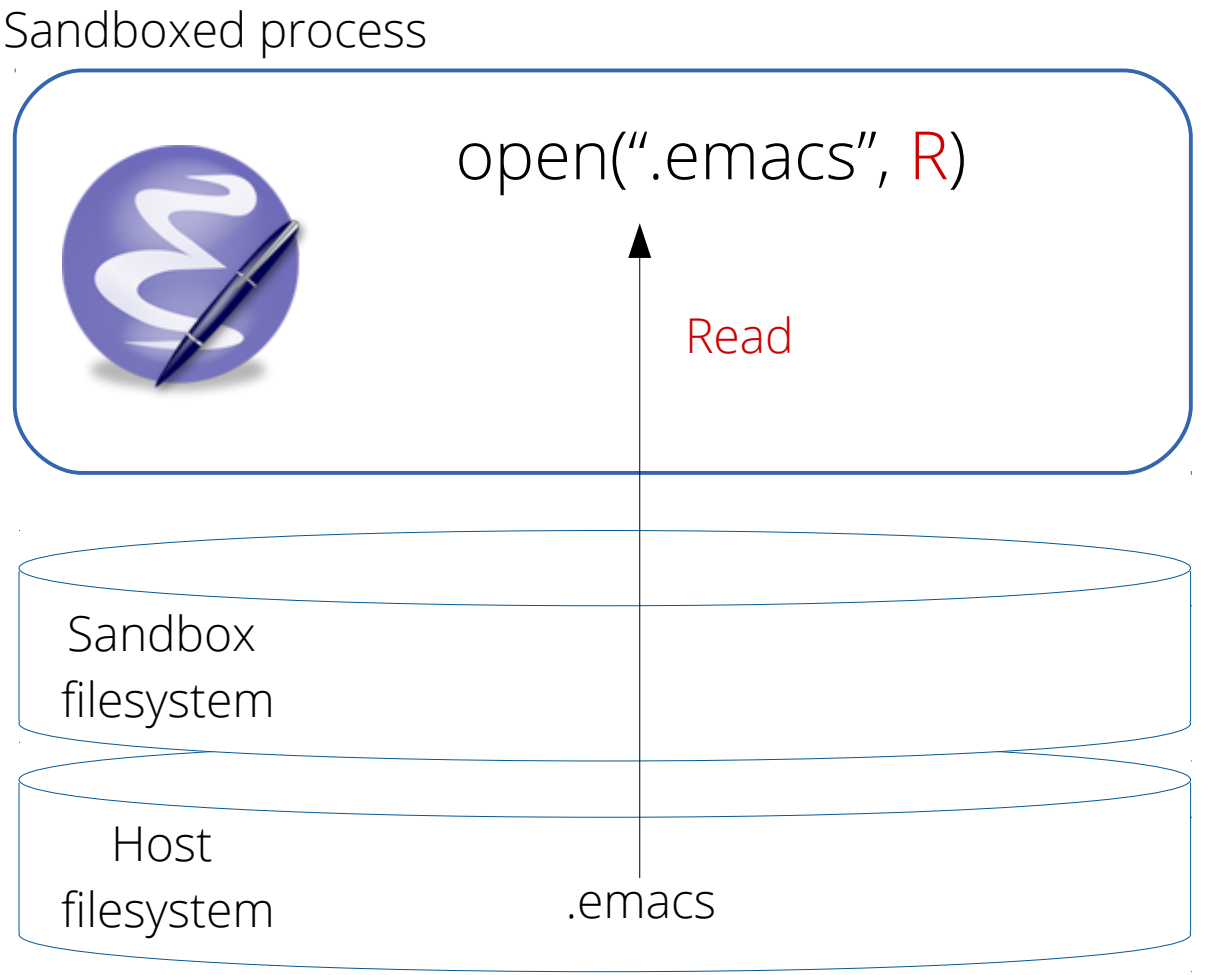
Outline

- Motivation / use cases
- Layered sandbox filesystem
- System call interposition (using seccomp/BPF)
- Implementation / evaluation
- Related work
- Summary

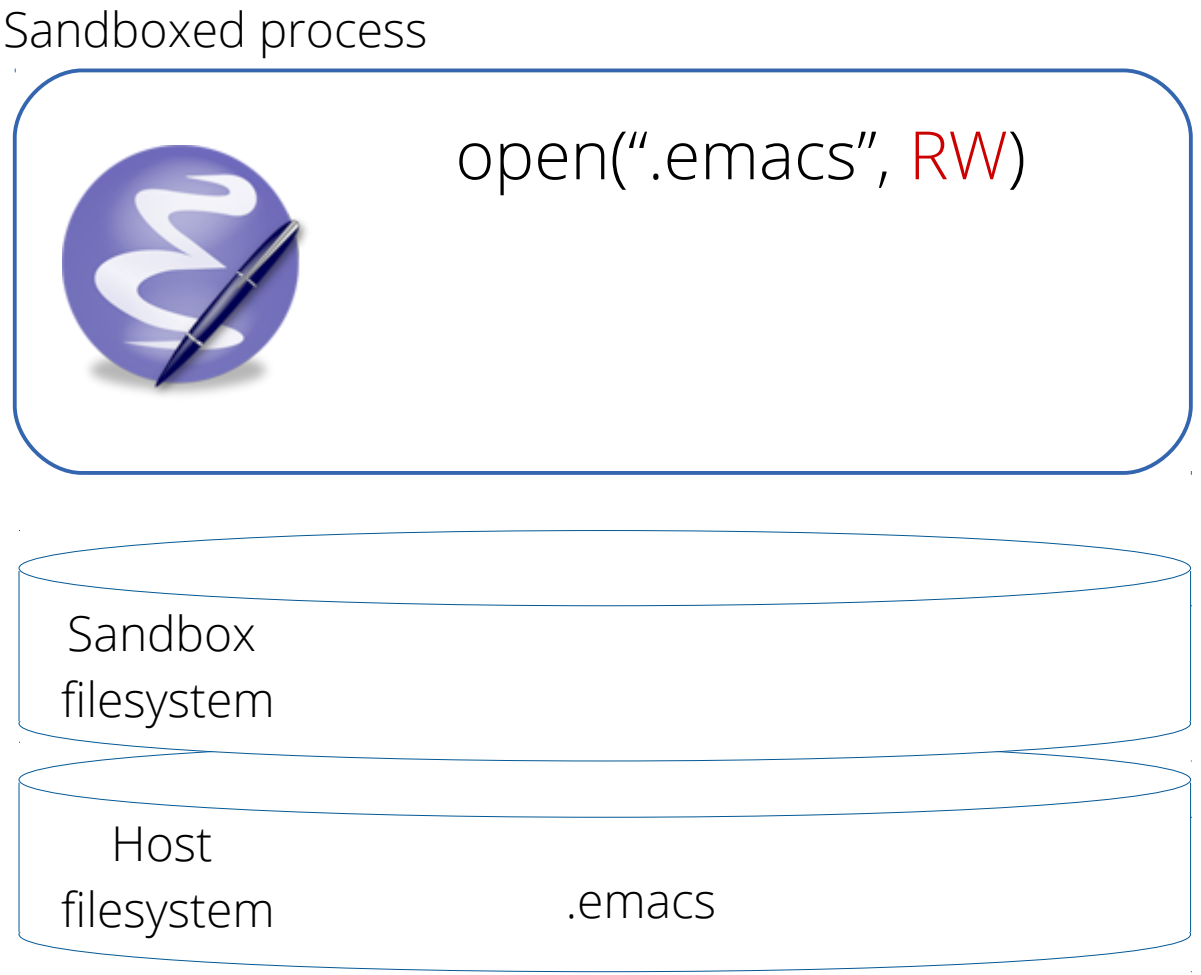
Sandbox filesystem supports copy-on-write



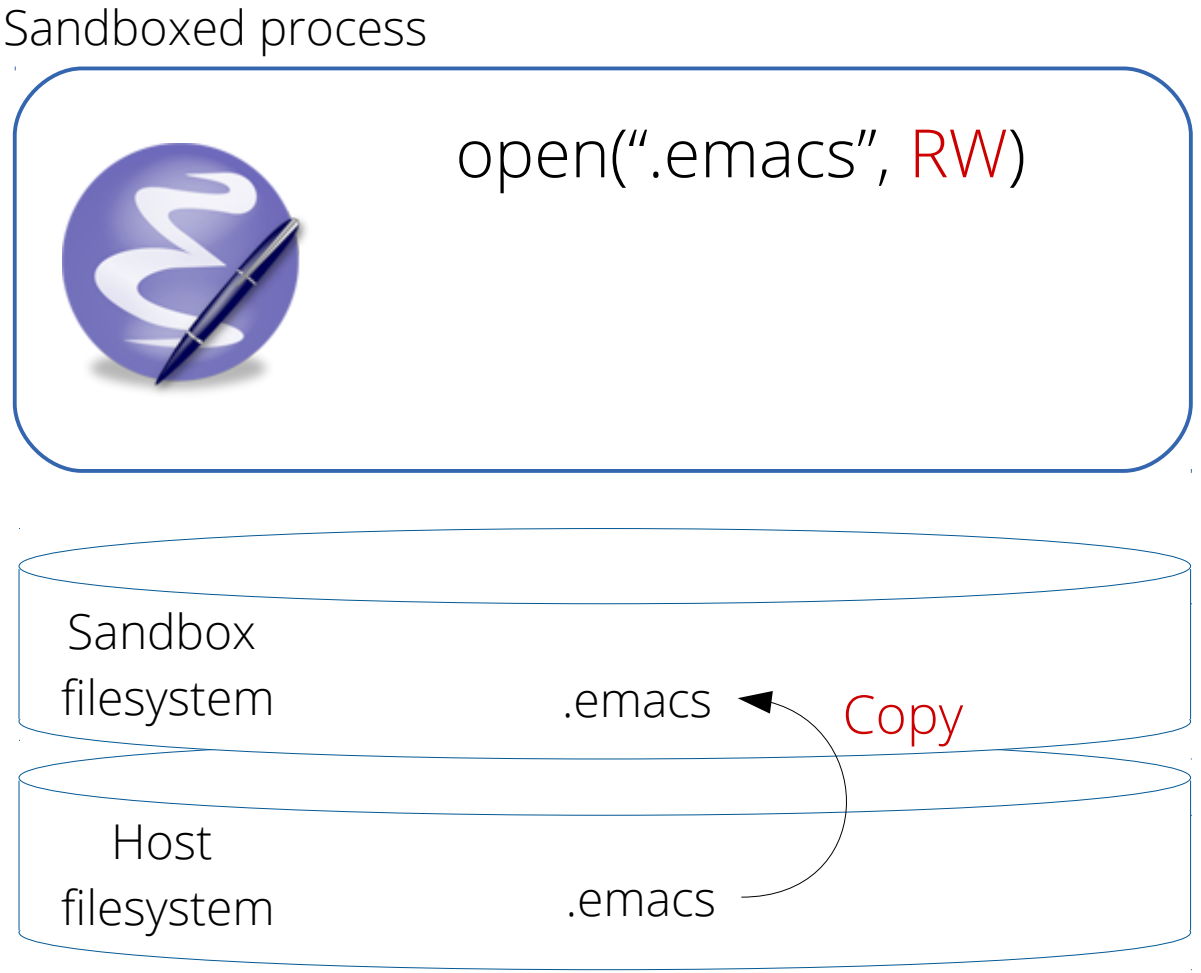
Sandbox filesystem supports copy-on-write



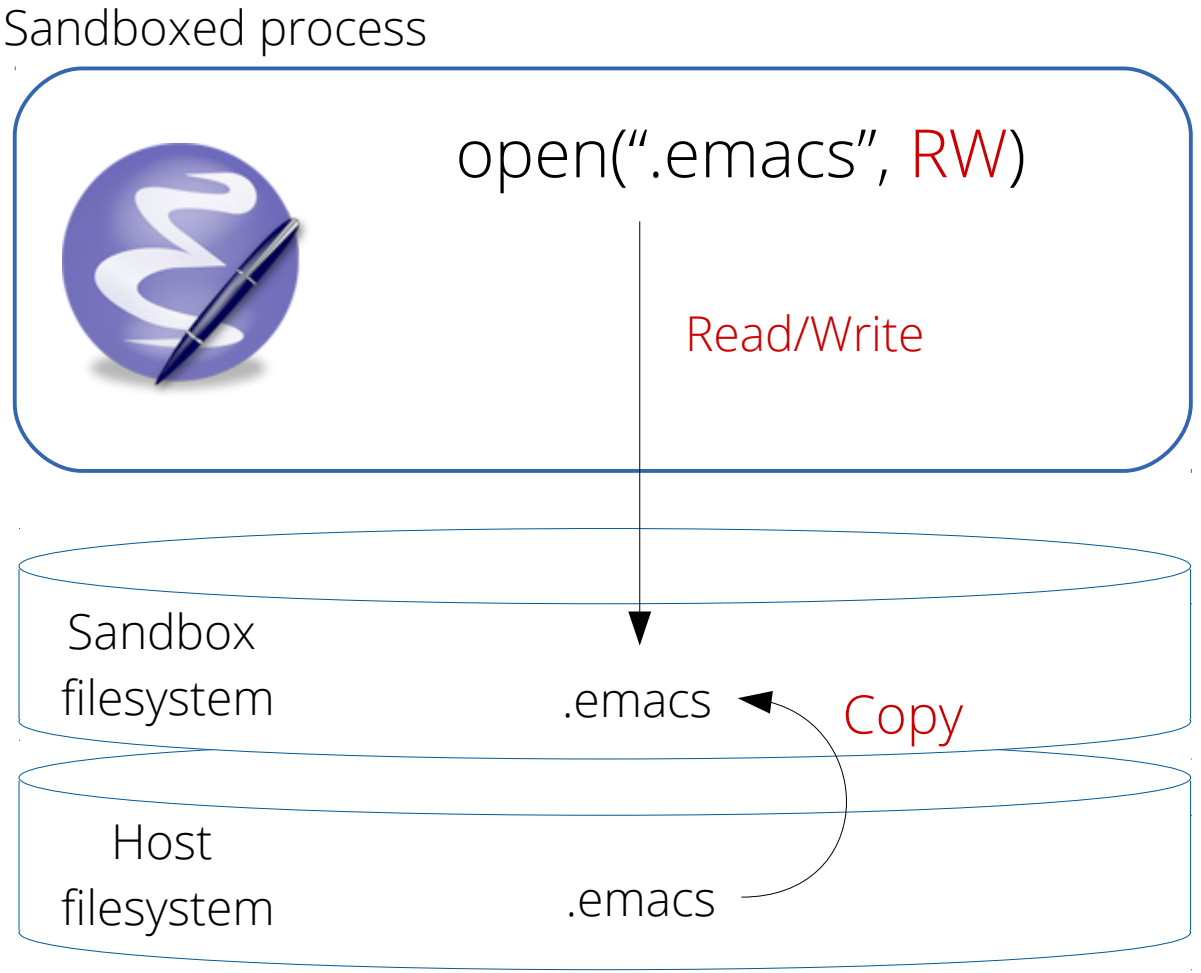
Sandbox filesystem supports copy-on-write



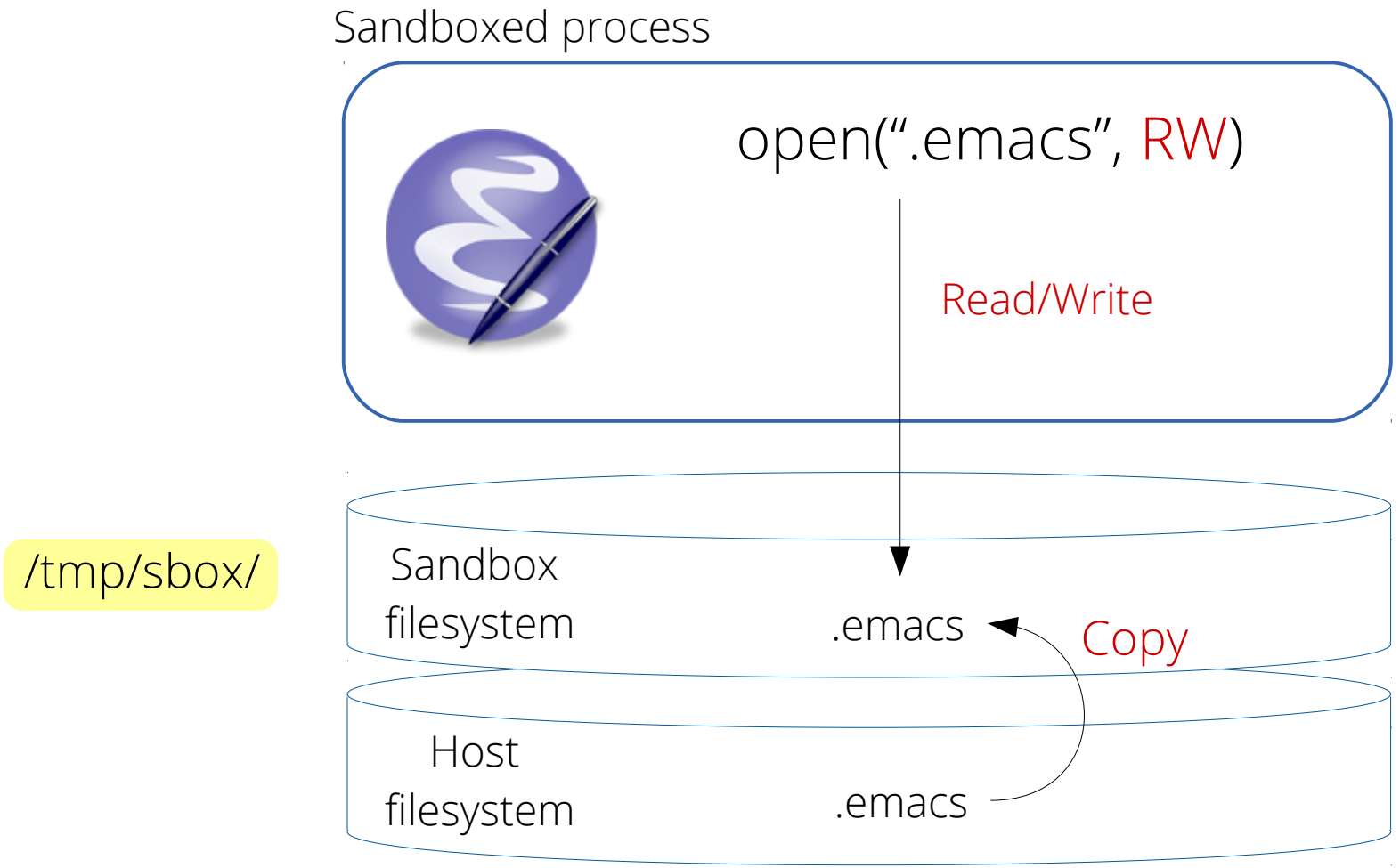
Sandbox filesystem supports copy-on-write



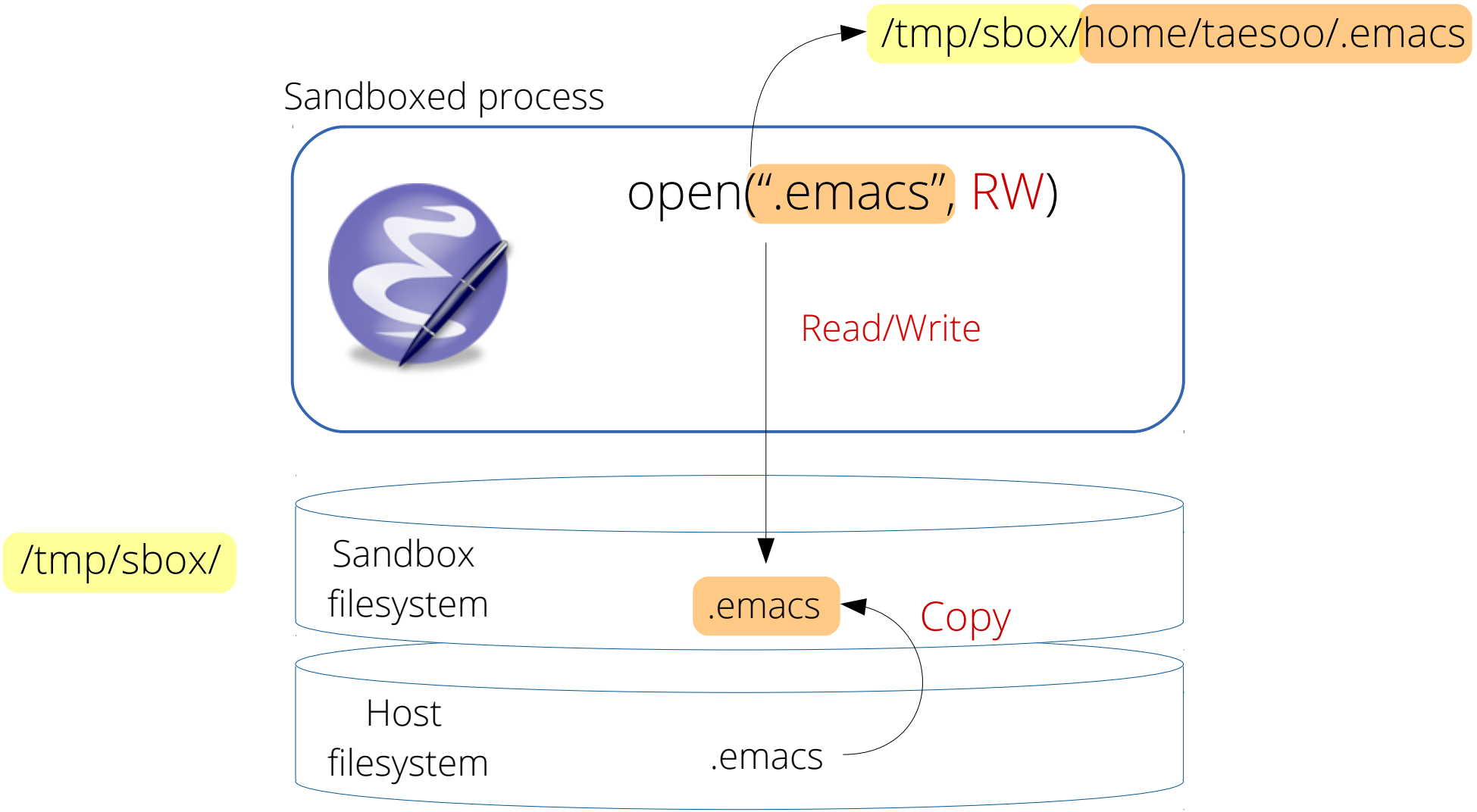
Sandbox filesystem supports copy-on-write



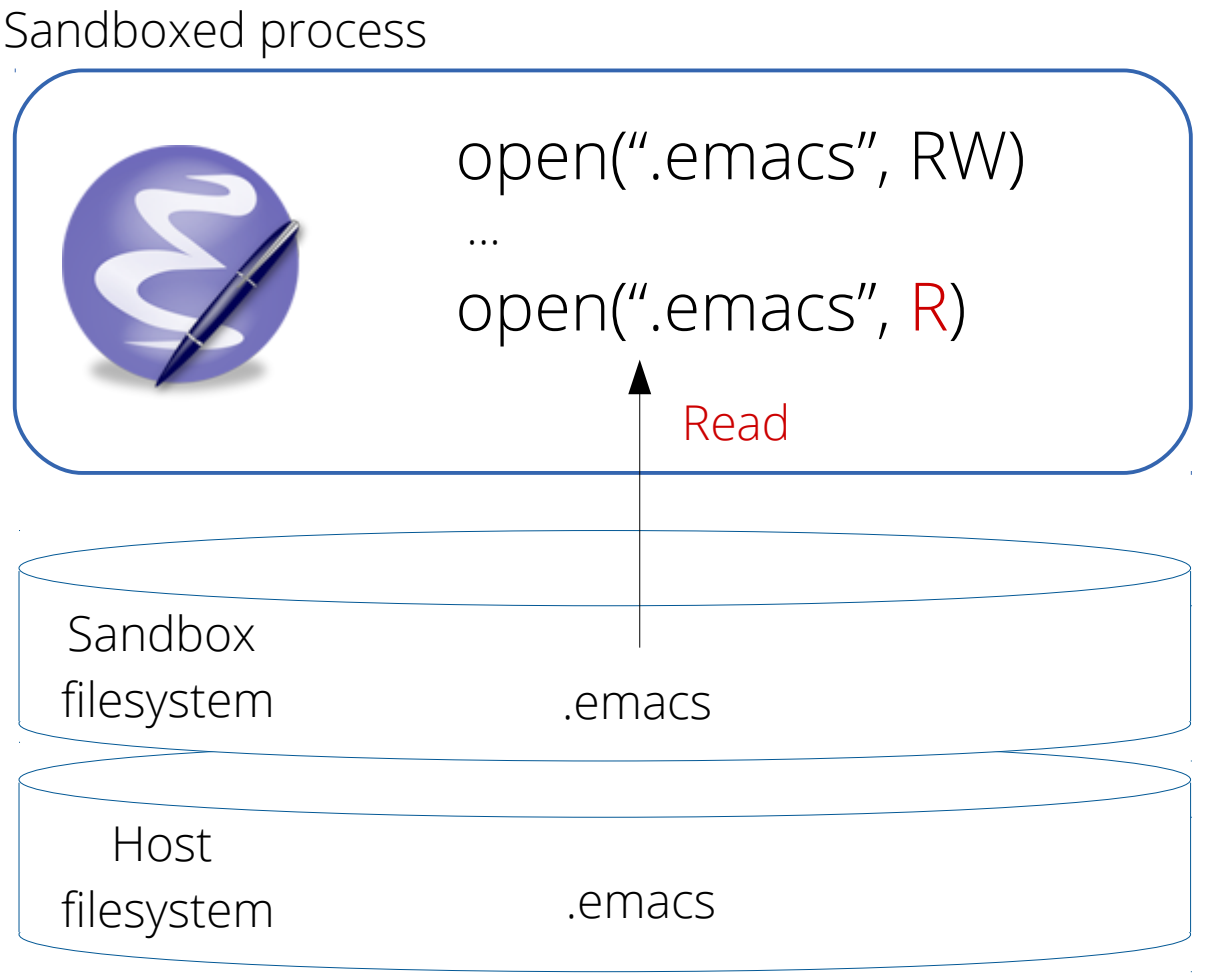
Copy-on-write by rewriting path arguments



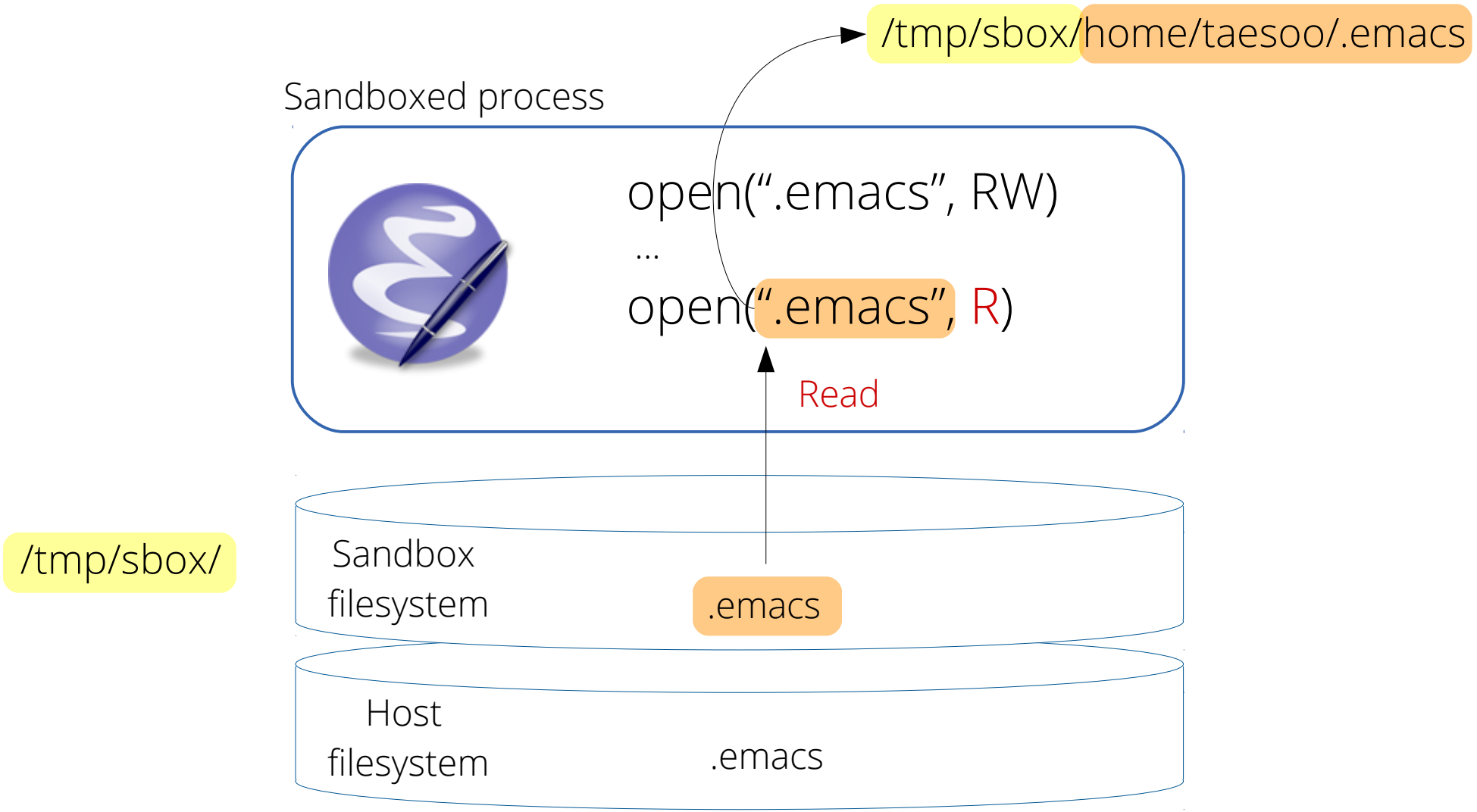
Copy-on-write by rewriting path arguments



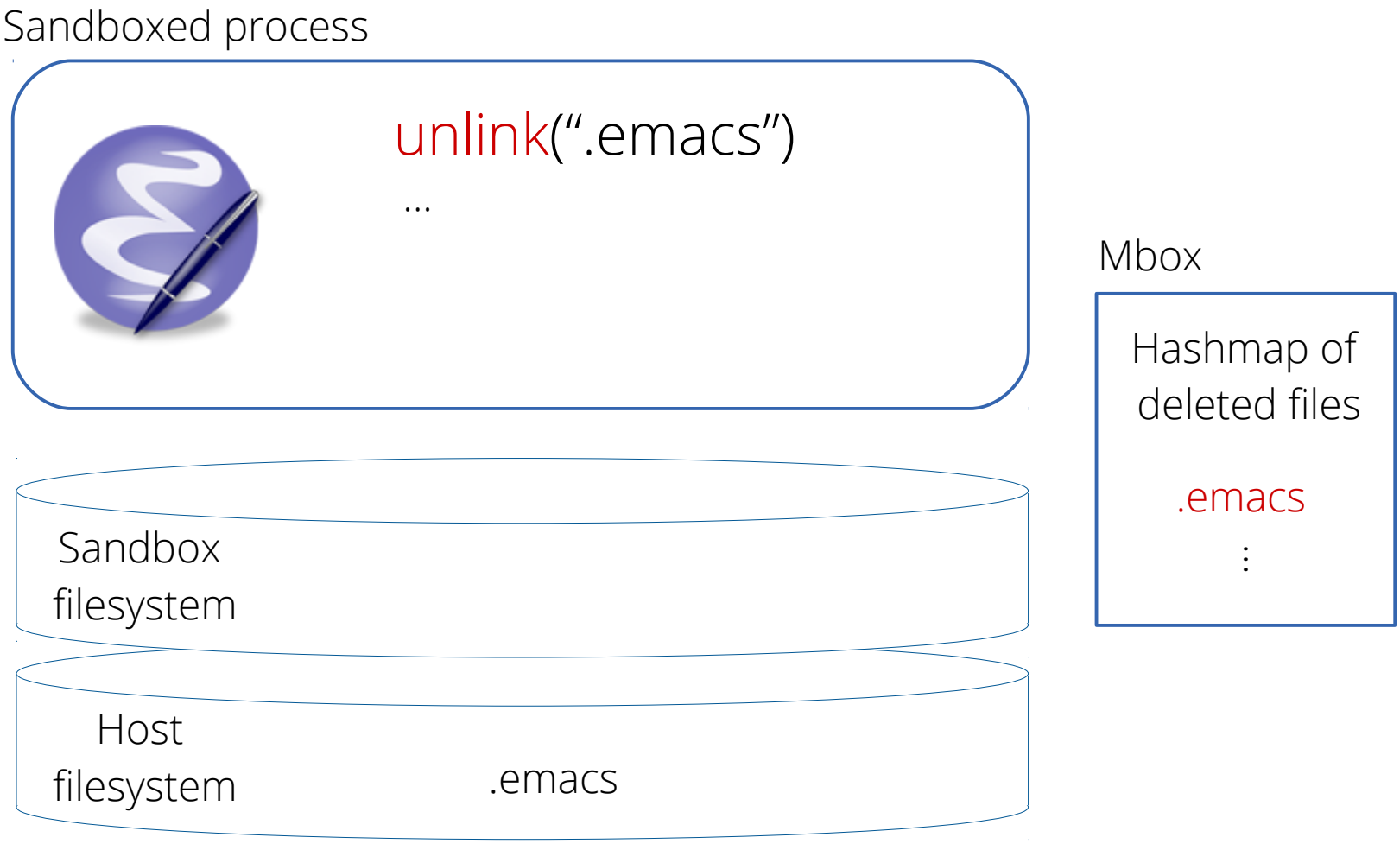
All subsequent read/write should happen on the sandbox filesystem



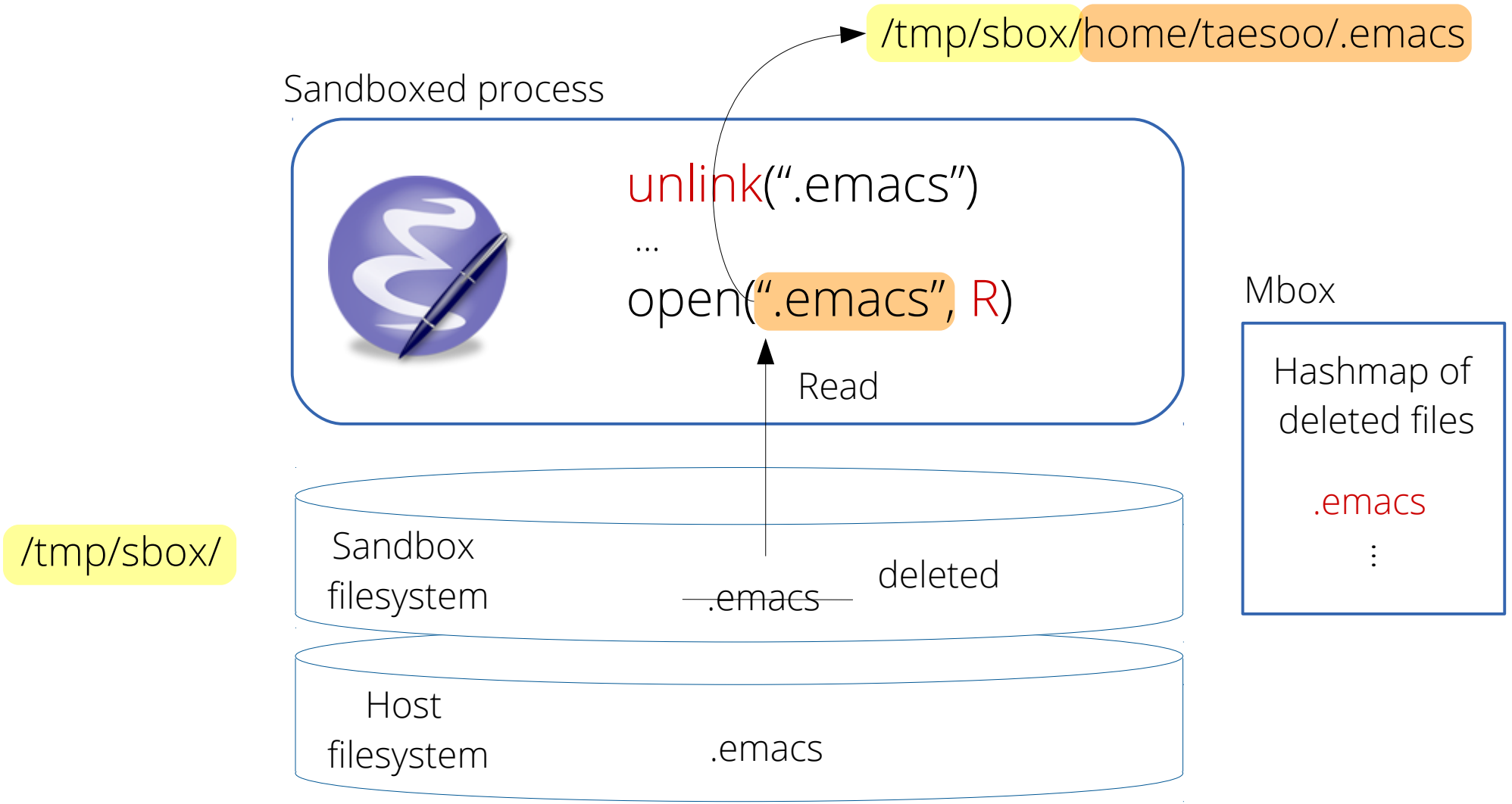
All subsequent read/write should happen on the sandbox filesystem



Sandbox filesystem keeps track of deleted files



Sandbox filesystem keeps track of deleted files



Mbox doesn't have to interpose on every system call

```
fd = open(".emacs", R)
read(fd, buf, size)
```

```
fd = open(".emacs", RW)
write(fd, buf, size)
```

- After redirecting the path in open(), we don't have to interpose on read/write() system calls
- Mbox needs to interpose on 48 system calls getting a path argument to provide a layered sandbox filesystem

Mechanism: system call interposition

- Ptrace is a common technique, but slow
  - Interpose entry/exit of every system call
  - Serialize system calls of child processes

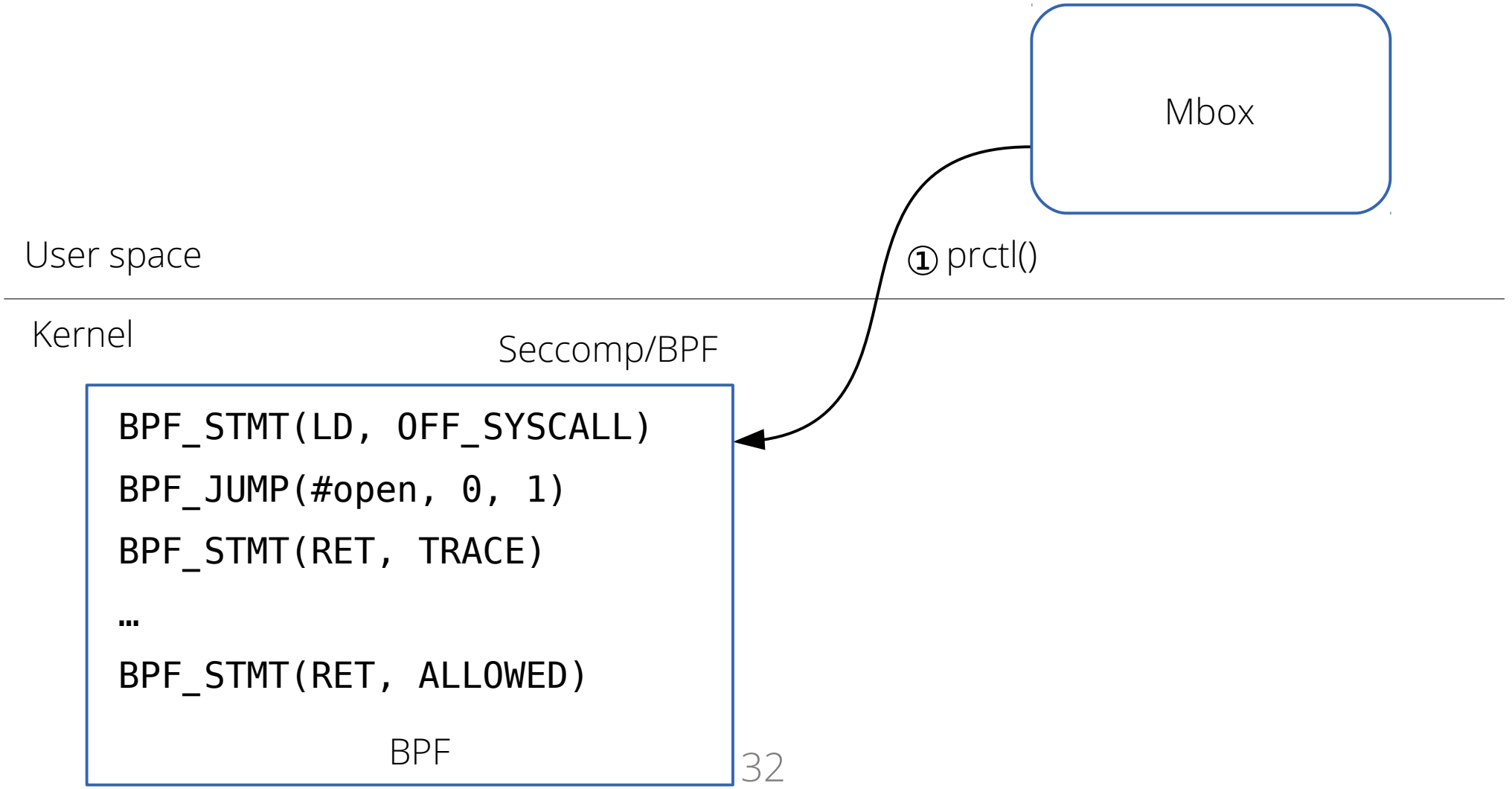
- Using seccomp/BPF (>= Linux 3.5)
  - Seccomp is a security mechanism for isolating a process by allowing a certain set of system calls
  - Seccomp/BPF uses BPF (Berkeley Packet Filter) to specify rules for filtering system calls

BPF program for interposition

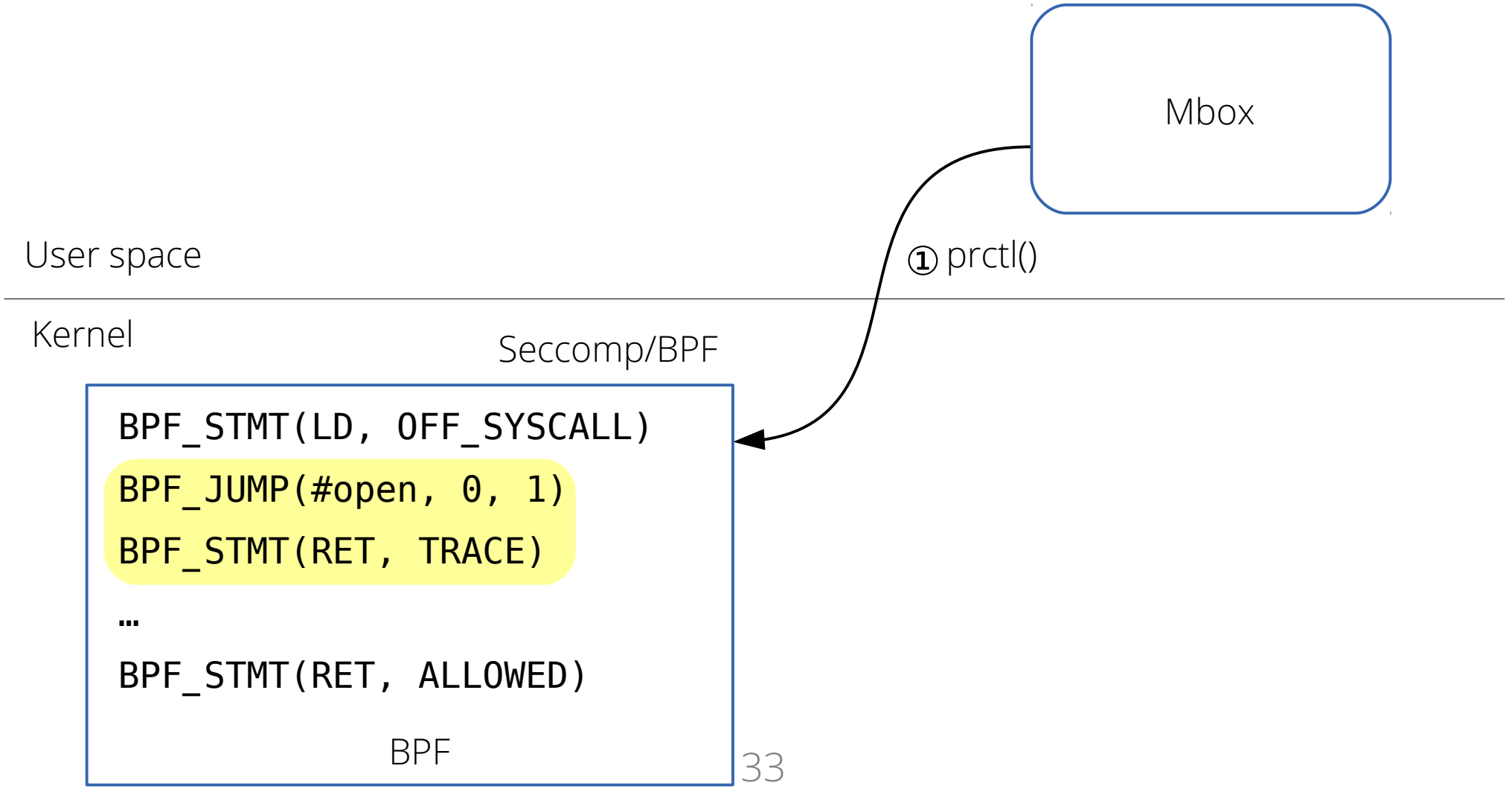
User space

Kernel

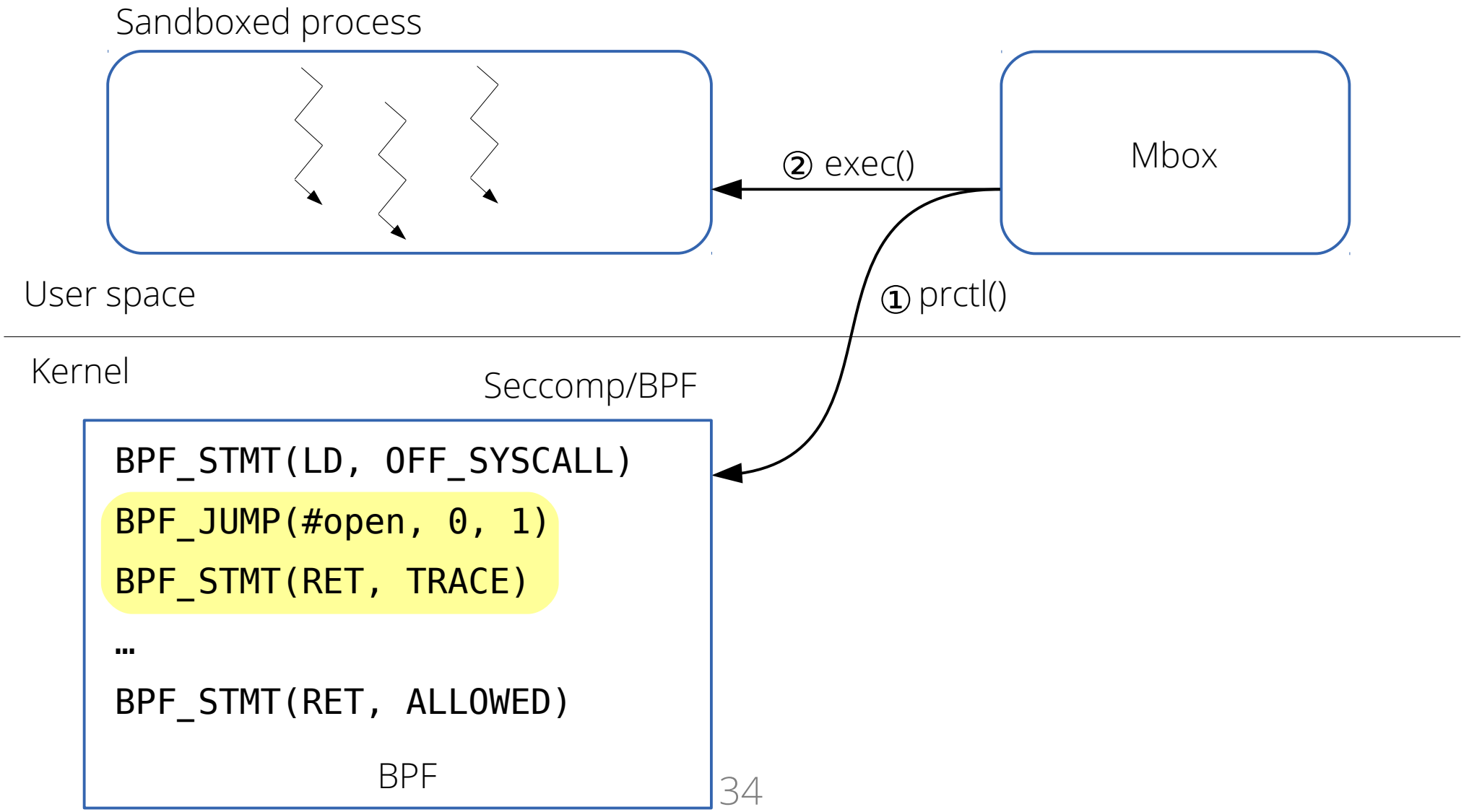
BPF program for interposition



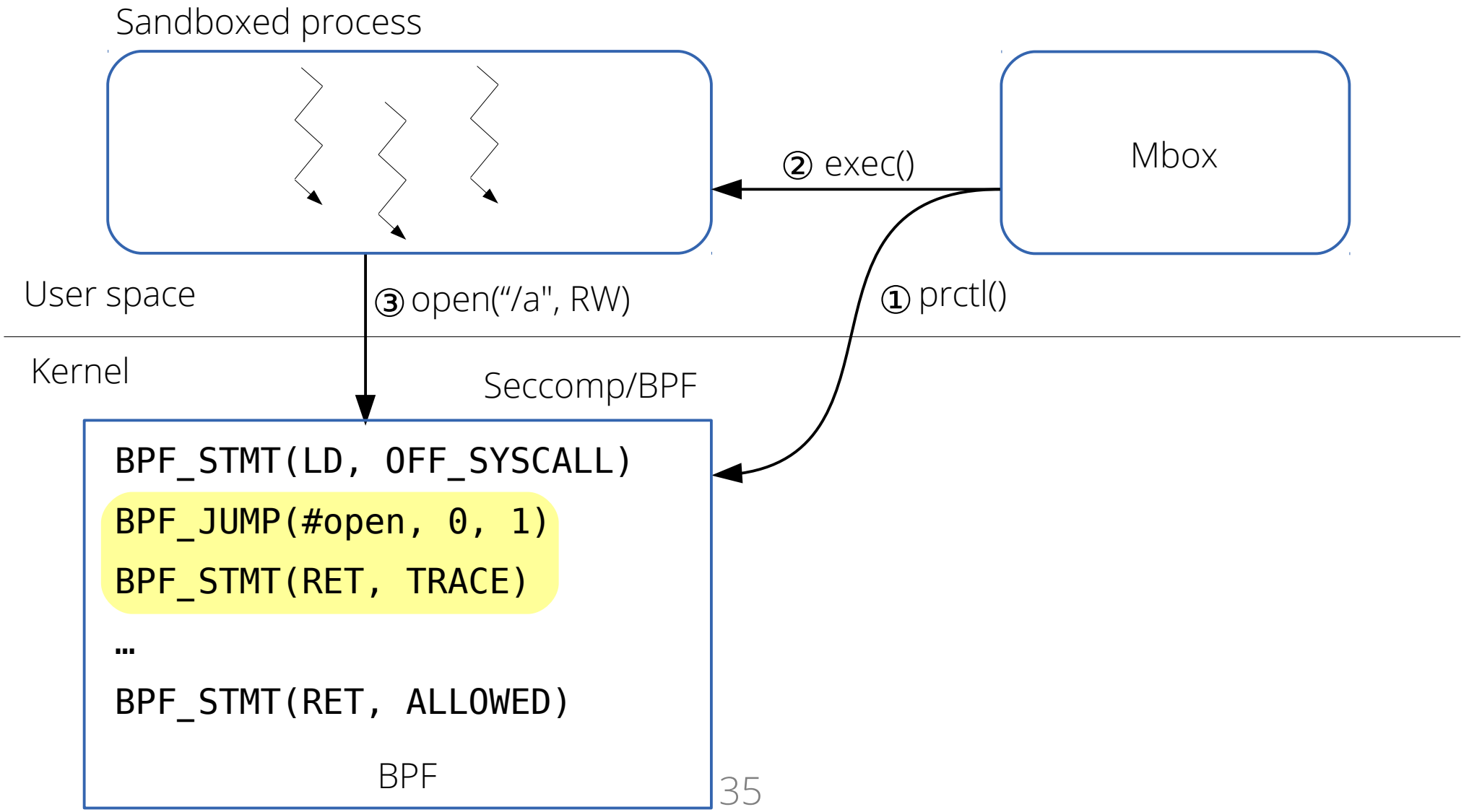
BPF program for interposition



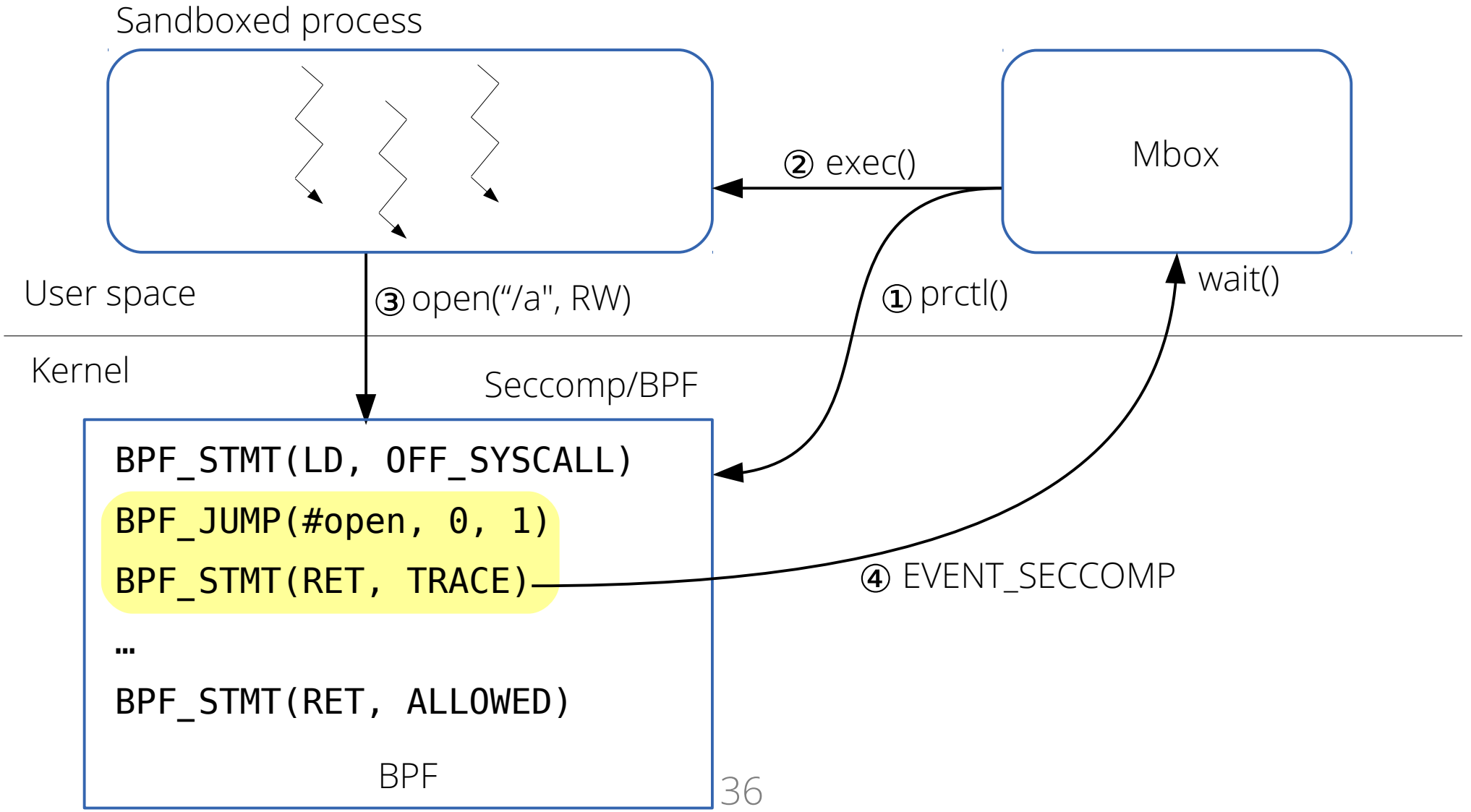
BPF program for interposition



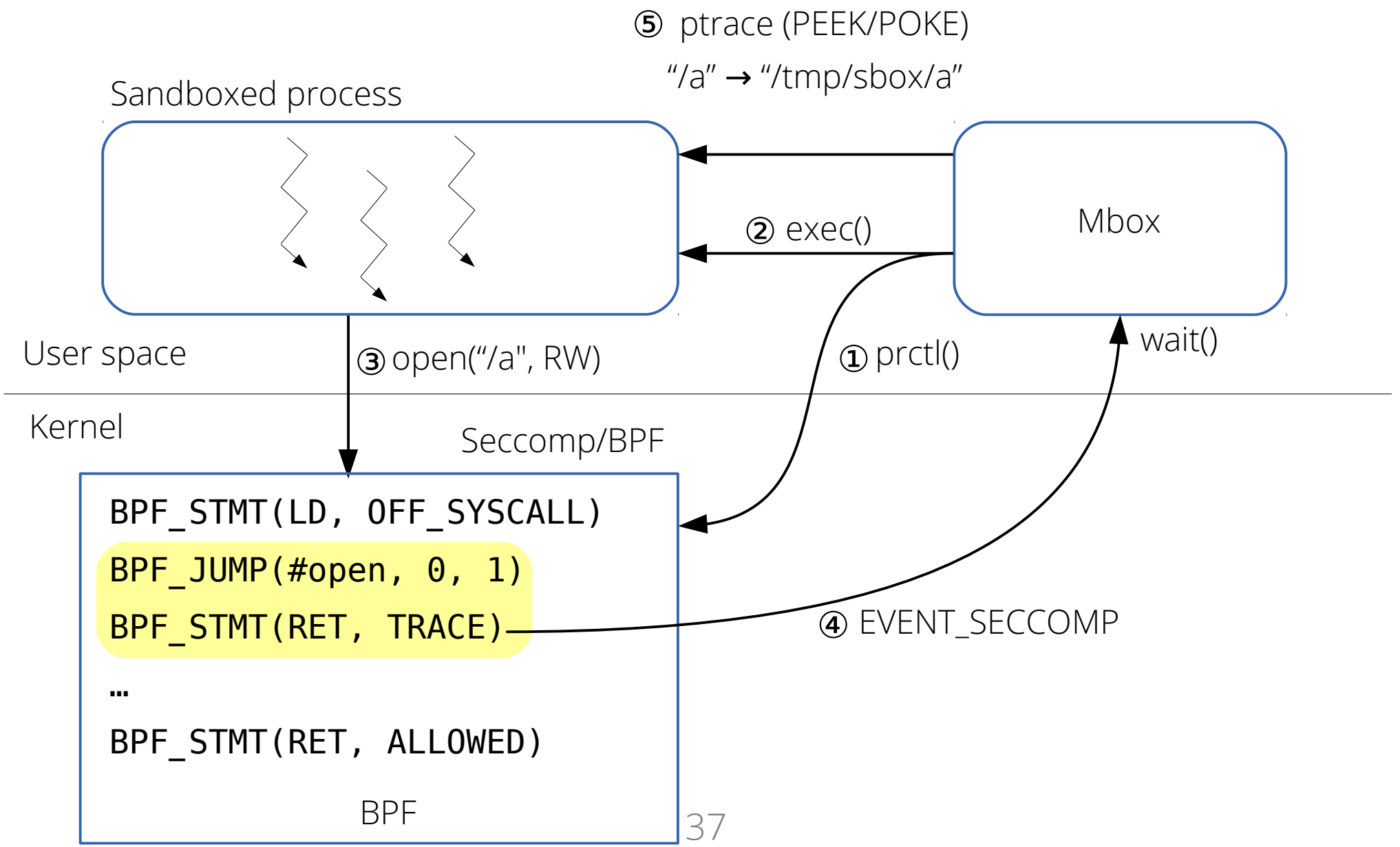
BPF program for interposition



BPF program for interposition



BPF program for interposition



More story to come …

- How to avoid time-of-check-to-time-of-use?
- How to avoid replicating OS state?
- …

Please, check the paper!

Implementation

- Mbox: a prototype for Linux (>= 3.5, x86-64)
  - Using seccomp/BPF and ptrace
  - Extending strace 4.7
  - 1,500 Lines of code
  - Distributions: Ubuntu 12.04 and Arch 64bit

Performance evaluation

- How much overhead does Mbox exhibit?
- How much faster is seccomp/BPF than ptrace?

Benchmark

Task	Description
Octave	Octave Benchmark calculating matrix
Zip	Compress source files of Linux 3.8
Untar	Decompress source files of Linux 3.8
Build Linux (-j1)	Compile Linux 3.8 kernel

- Following the benchmark from Apiary
- Run each benchmark in three configurations
  - Normal
  - Mbox with ptrace
  - Mbox with seccomp/BPF

Mbox imposes modest end-to-end performance overhead

Task	Normal	Mbox	
		Seccomp/BPF	
Octave	2.1s	2.1s	0.1%
Zip	15.6s	17.4s	12.0%
Untar	13.6s	16.4s	20.9%
Build Linux (-j1)	43.6s	49.7s	13.9%

- 0.1% ~ 20.9% overhead
- Octave: a computation-heavy workload
  - Exhibits negligible performance overhead (0.1%)
  - Spends 98% of its execution in userspace

Seccomp/BPF reduces the interposition overhead

Task	Normal	Mbox			
		Ptrace		Seccomp/BPF	
Octave	2.1s	2.1s	0.1%	2.1s	0.1%
Zip	15.6s	21.2s	36.5%	17.4s	12.0%
Untar	13.6s	19.0s	40.3%	16.4s	20.9%
Build Linux (-j1)	43.6s	53.2s	21.9%	49.7s	13.9%

- Compare overheads of using ptrace and seccomp/BPF
- Seccomp/BPF reduces overhead up to 24.5%

Seccomp/BPF has better concurrency than ptrace

Task	Normal	Mbox			
		Ptrace		Seccomp/BPF	
Build Linux (-j1)	43.6s	53.2s	21.9%	49.7s	13.9%
Build Linux (-j4)	21.7s	45.6s	110.1%	31.5s	45.2%

- When compiling the Linux kernel with 4 parallel jobs, performance improves 64.9% compared to ptrace
- By avoiding unnecessary serialization of system calls, multiple processes execute system calls concurrently

Seccomp/BPF has better concurrency than ptrace

Task	Normal	Mbox			
		Ptrace		Seccomp/BPF	
Build Linux (-j1)	43.6s	53.2s	21.9%	49.7s	13.9%
Build Linux (-j4)	21.7s	45.6s	110.1%	31.5s	45.2%

- When compiling the Linux kernel with 4 parallel jobs, performance improves 64.9% compared to ptrace
- By avoiding unnecessary serialization of system calls, multiple processes execute system calls concurrently

Related work

- **Layered filesystems**: UnionFS [Quigley '06] / Aufs
  - Following unification rules / copy-on-write
    - → Require no modifications in commodity OSes
- **System call interposition**: Ostia [Garfinkel '04]
  - Enforcing security policies / studied common pitfalls
    - → Summarize our experience of using seccomp/BPF
- **Namespace**: Plan9 [Pike '90] / Lxc container (Docker)
  - Private namespace for each process
    - → Enabling various applications via system call interposition

Summary

Mbox: a lightweight sandboxing mechanism

- **Layered** sandbox filesystem
- Revision-control-system like sandbox usage model
- Interposing on system calls with **seccomp/BPF**
- Enabling a variety of applications for **non-root** users

http://pdos.csail.mit.edu/mbox

Questions (if you don't have any)

- What if files are modified by other processes running outside of Mbox?
- Why 20% on tar? just rewriting path arguments doesn't seem to be demanding work.
- How complicated the BPF program? Why not implement everything in BPF then?
- Why does Mbox support only 64bit? and is Mbox ready for users (not developers)?
- Can Mbox be used for A, B and C ... ?