[544] Capability-Based Security with UNIX Sockets

Tyler Caraza-Harter

Capabilities

Definition 1 (for this session): "a communicable, unforgeable token of authority. It refers to a value that references an object along with an associated set of access rights.
A user program on a capability-based operating system must use a capability to access an object." - https://en.wikipedia.org/wiki/Capability-based_security

Definition 2 (seccomp session): a Linux abstraction that resulted from decomposing "root" priviliges into smaller parts.

Socket Domains

int s = socket(domain, type, protocol);

Internet Domain (AF_INET) - most familiar type

- SOCK_STREAM type: TCP
- SOCK_DGRAM type: UDP

example address: "myserver:5000"
advantage: can communicate between different machines

UNIX Domain (AF_UNIX)

example address: "/tmp/connection.sock"
advantage: more secure -- who can access is controlled by Linux file permissions

Socket Domains

int s = socket(domain, type, protocol);

Internet Domain (AF_INET) - most familiar type

- SOCK_STREAM type: TCP
- SOCK_DGRAM type: UDP

example address: "myserver:5000"
advantage: can communicate between different machines
calls: `send, recv`

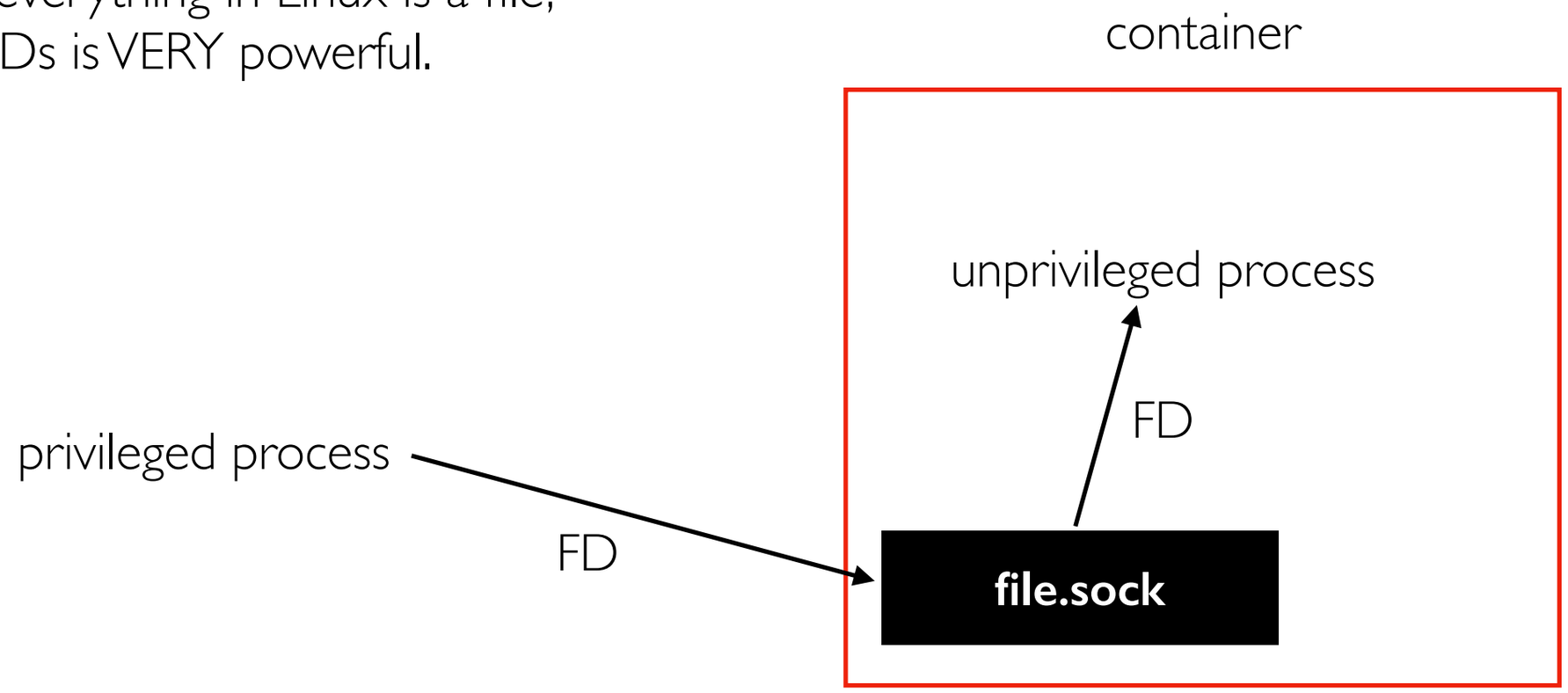
UNIX Domain (AF_UNIX)

example address: "/tmp/connection.sock"
advantage: more secure -- who can access is controlled by Linux file permissions
calls: `send, recv`, `sendmsg, recvmsg`

bytes bytes + FDs

Use Cases

As they say, everything in Linux is a file,
so sending FDs is VERY powerful.



FDs for

- file that process should mmap
- directory that should be used for chroot
- namespace that should be joined
- cgroup that should be joined
- etc.

gVisor (used by Google Cloud Functions)

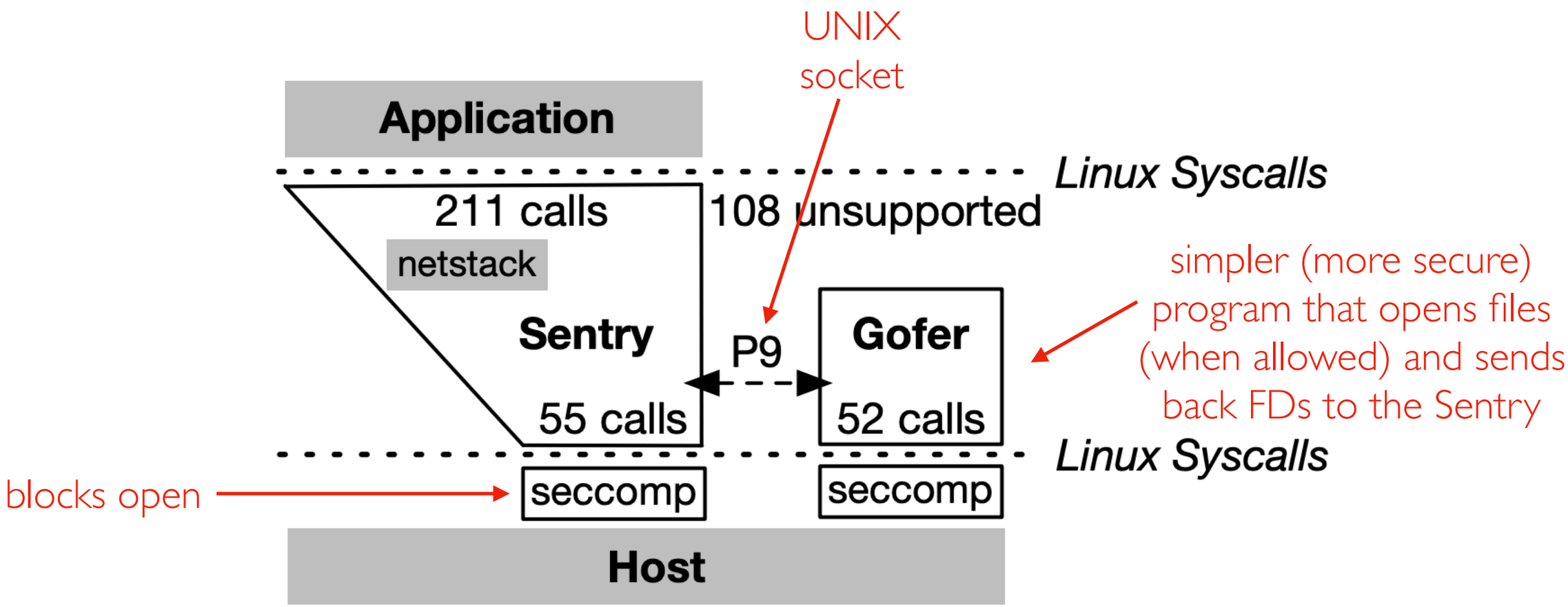


Figure 1: **gVisor Architecture.**

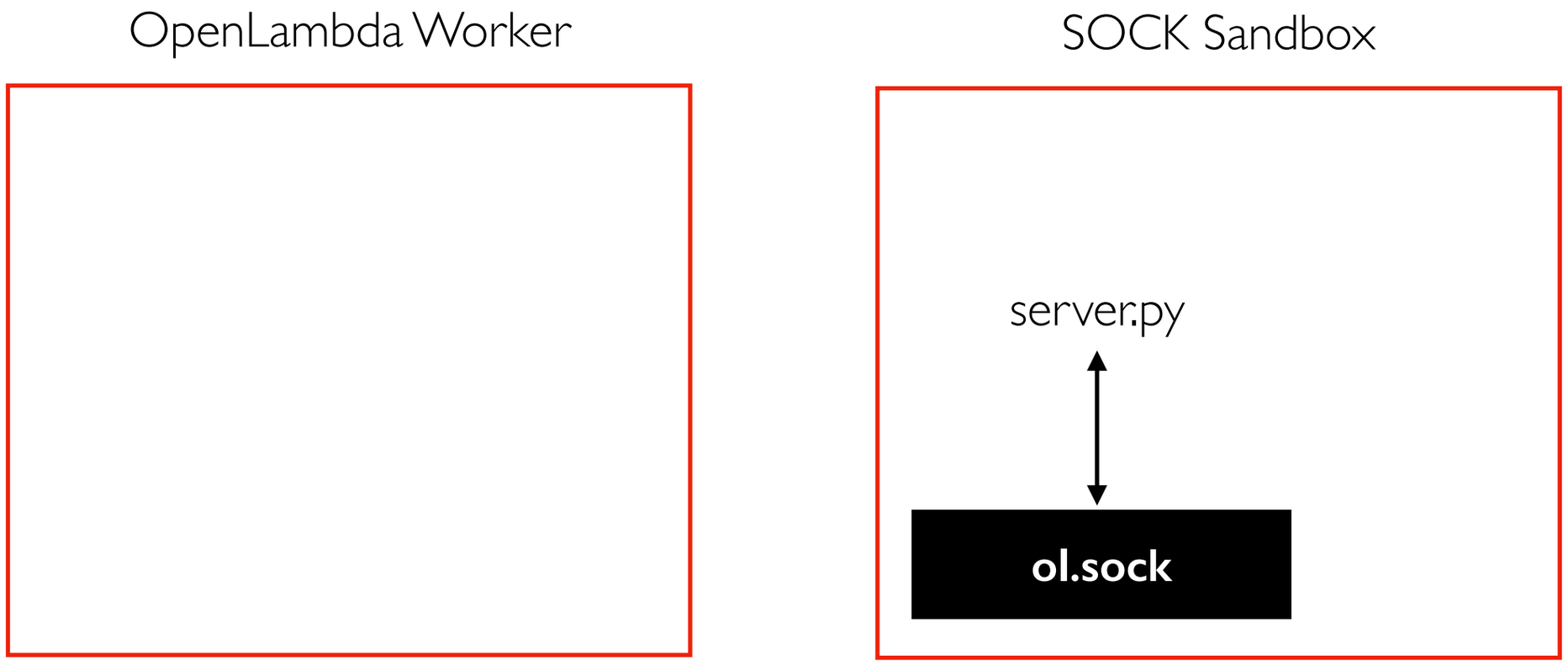
The True Cost of Containing: A gVisor Case Study

Ethan G. Young, Pengfei Zhu, Tyler Caraza-Harter,
Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

University of Wisconsin, Madison

https://www.usenix.org/system/files/hotcloud19-paper-young.pdf

OpenLambda

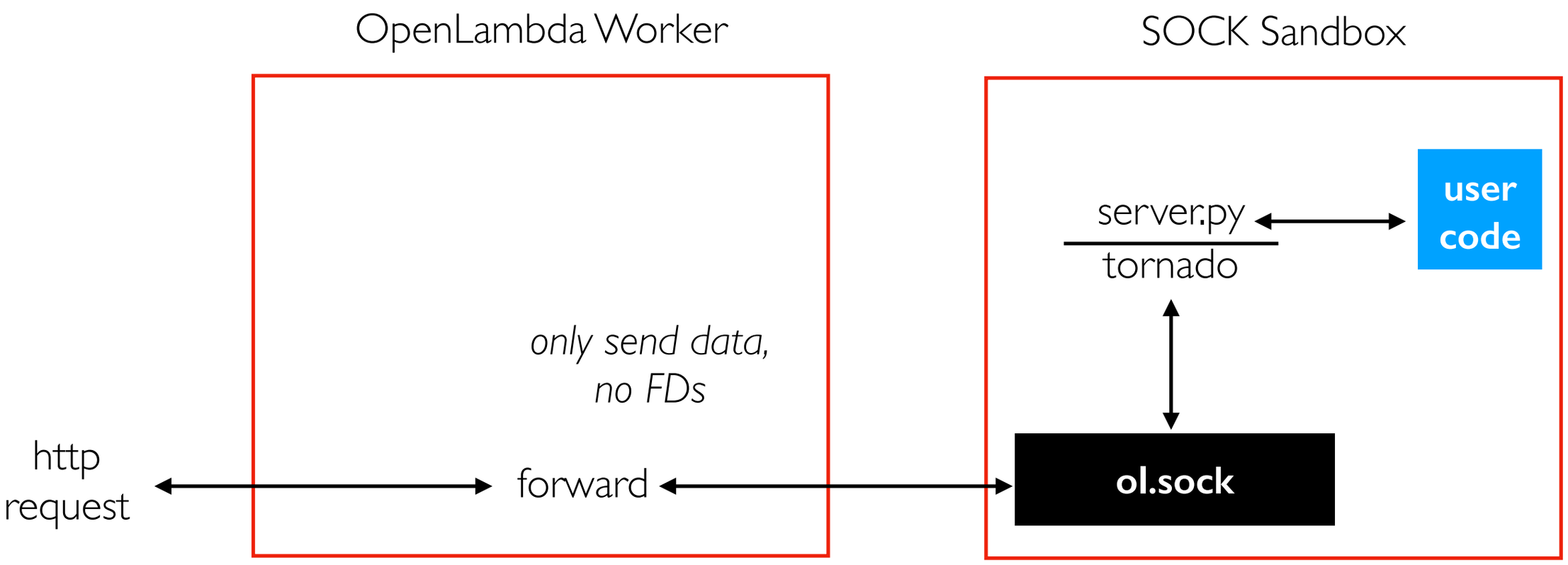


server.py runs in two modes, both of which use a UNIX socket named "ol.sock"

- web_server() - for Lambda instances
- fork_server() - for Zygotes

https://github.com/open-lambda/open-lambda/blob/main/min-image/runtimes/python/server.py

OpenLambda



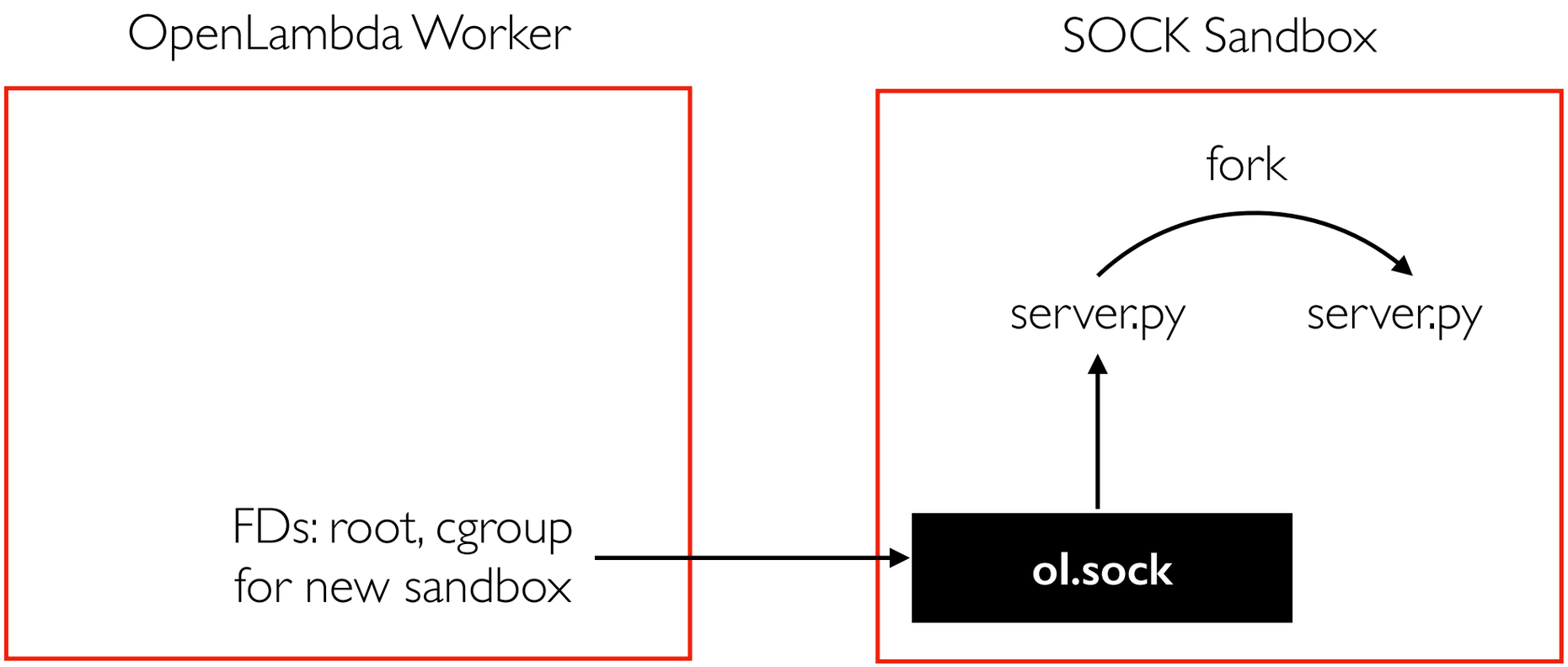
https://github.com/tornadoweb/tornado

server.py runs in two modes, both of which use a UNIX socket named "ol.sock"

- web_server() - for Lambda instances
- fork_server() - for Zygotes

https://github.com/open-lambda/open-lambda/blob/main/min-image/runtimes/python/server.py

OpenLambda

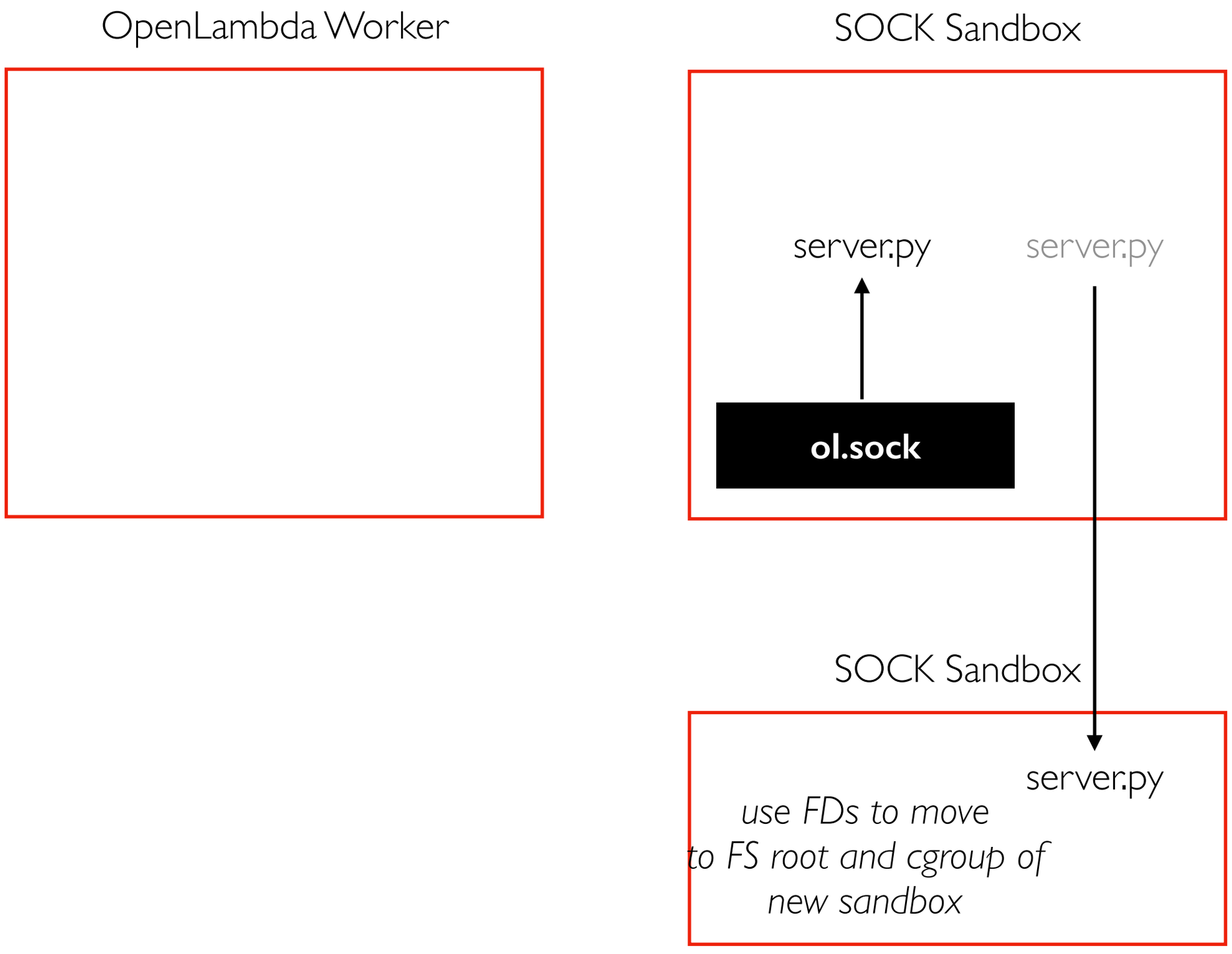


server.py runs in two modes, both of which use a UNIX socket named "ol.sock"

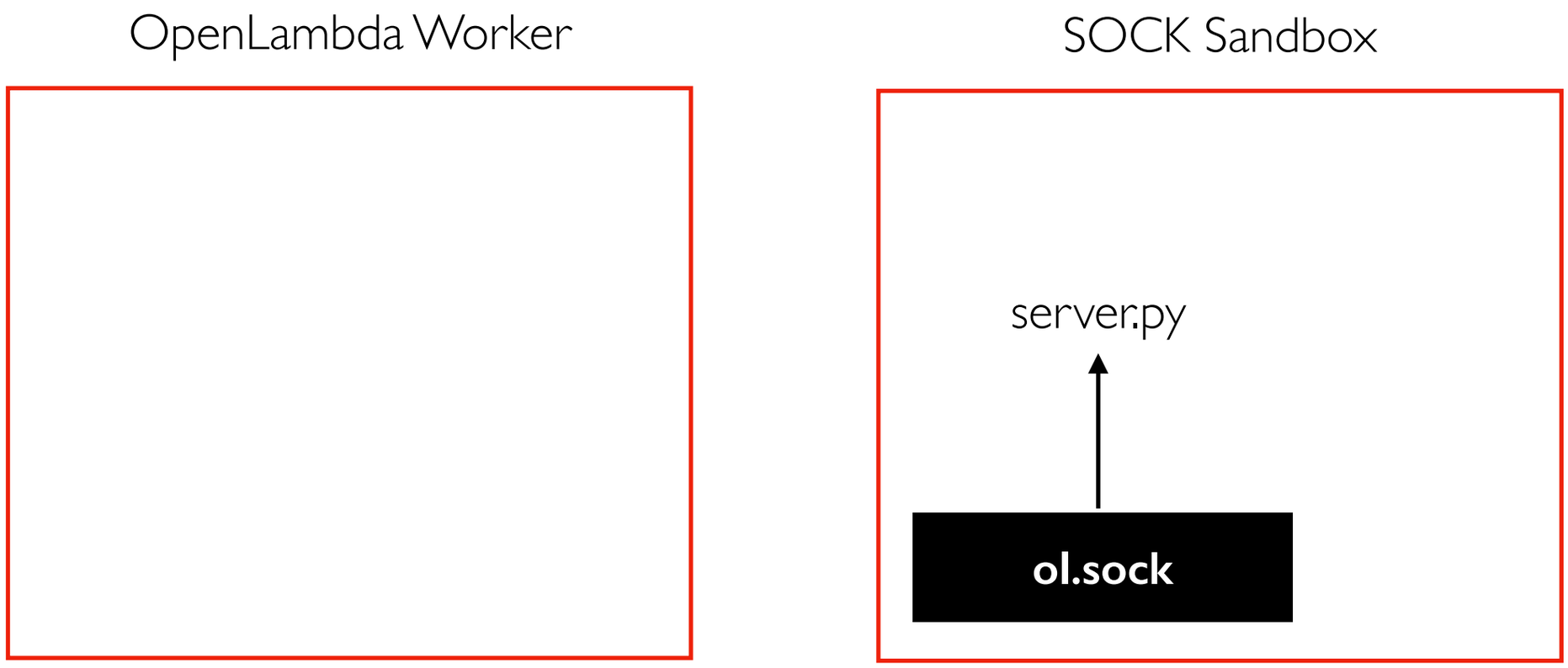
- web_server() - for Lambda instances
- fork_server() - for Zygotes

https://github.com/open-lambda/open-lambda/blob/main/min-image/runtimes/python/server.py

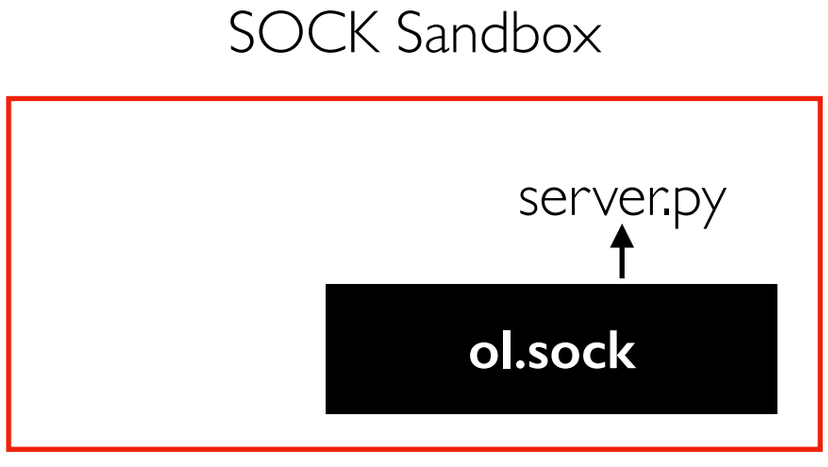
OpenLambda


OpenLambda Worker
SOCK Sandbox
server.py
server.py
ol.sock
SOCK Sandbox
server.py
use FDs to move
to FS root and cgroup of
new sandbox

OpenLambda



new server.py might run fork_server() again if this is a child Zygote, or it might start web_server() if it's a leaf.



Namespace FDs

Originally, OpenLambda used `setns` to join existing namespaces, with namespaces FDs passing over the UNIX socket (along with root dir and cgroup FDs).

Namespaces cannot exist apart from a process, so the original SOCK paper described an "init" process, now "spin" -- we should get rid of this (only the Docker backend uses it, not SOCK at this point).

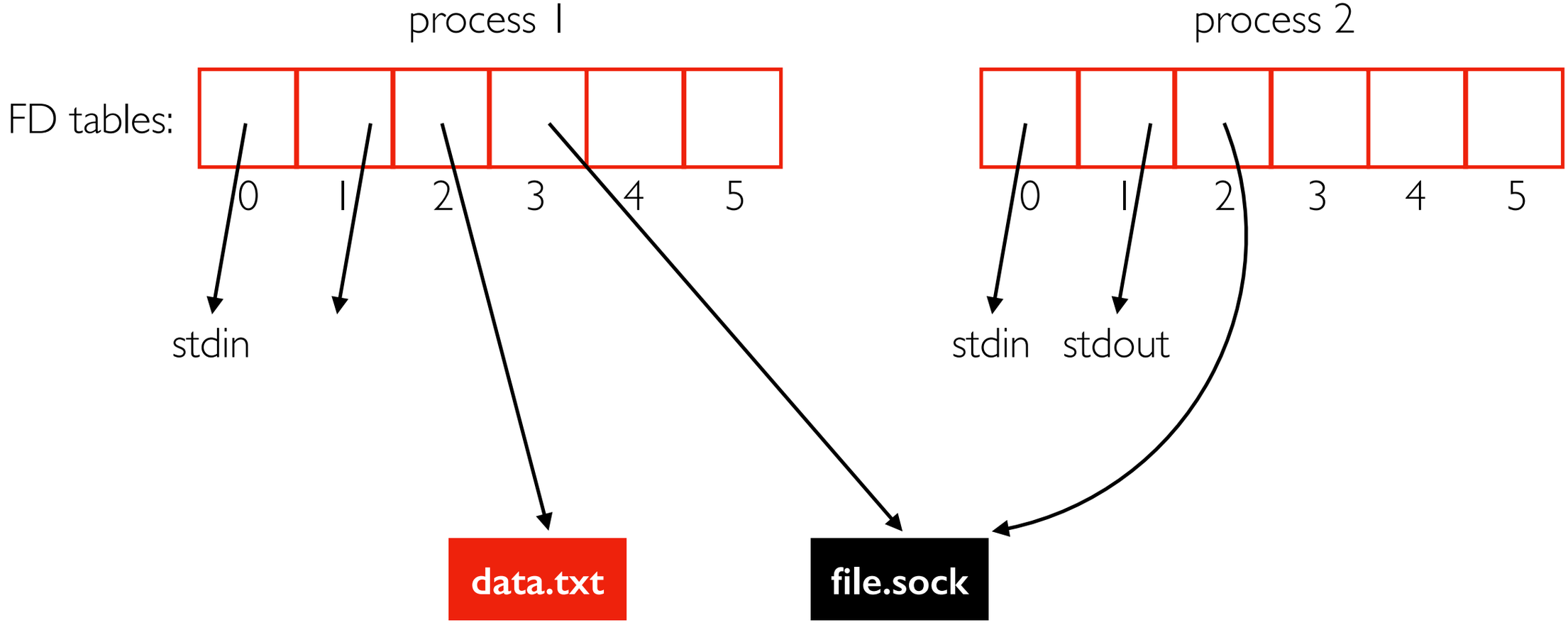
Now, SOCK uses `unshare` to create the new namespaces.

SOCK paper was published using the 4.13 kernel. The 5.3 kernel (2019) has `clone3`, with parameters for cgroups and namespaces. SOCK ought to be updated to use this instead.

File Descriptor Tables

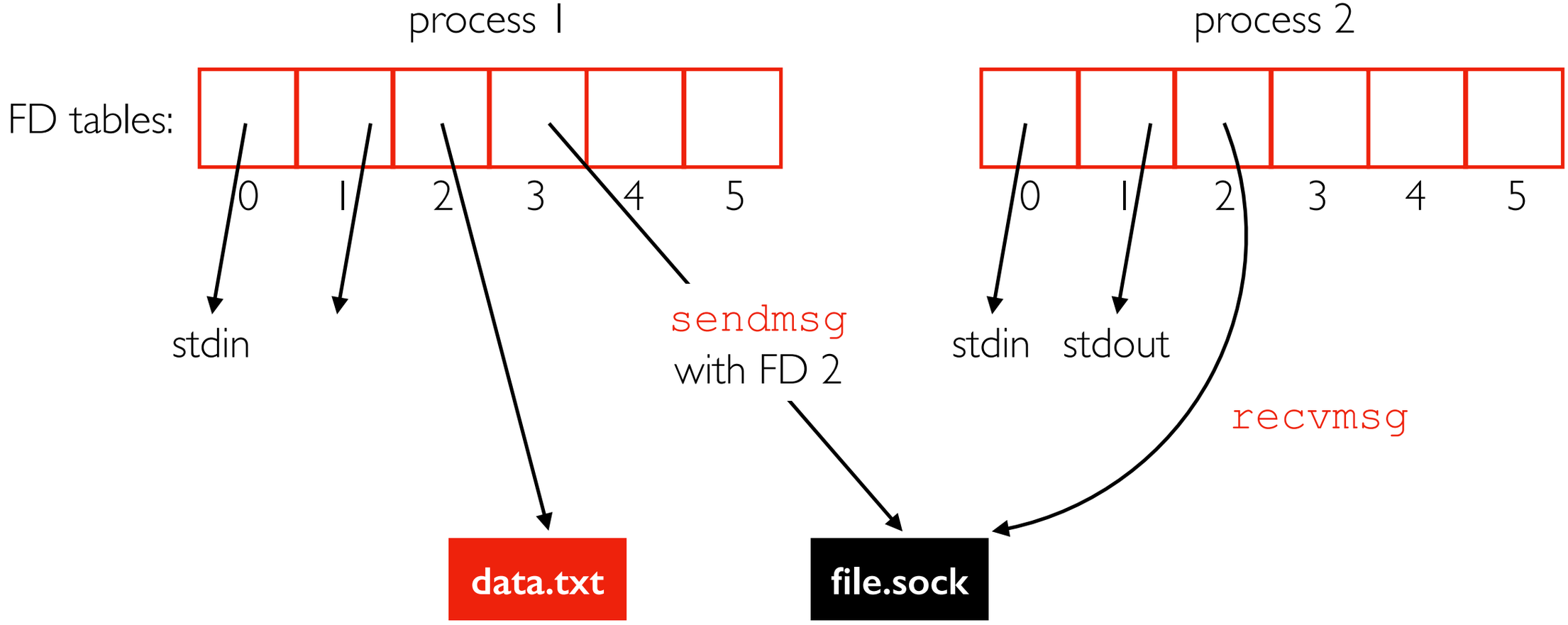
FDs in the FD tables contain

- reference to an object in the kernel
- offset, permissions
- etc.

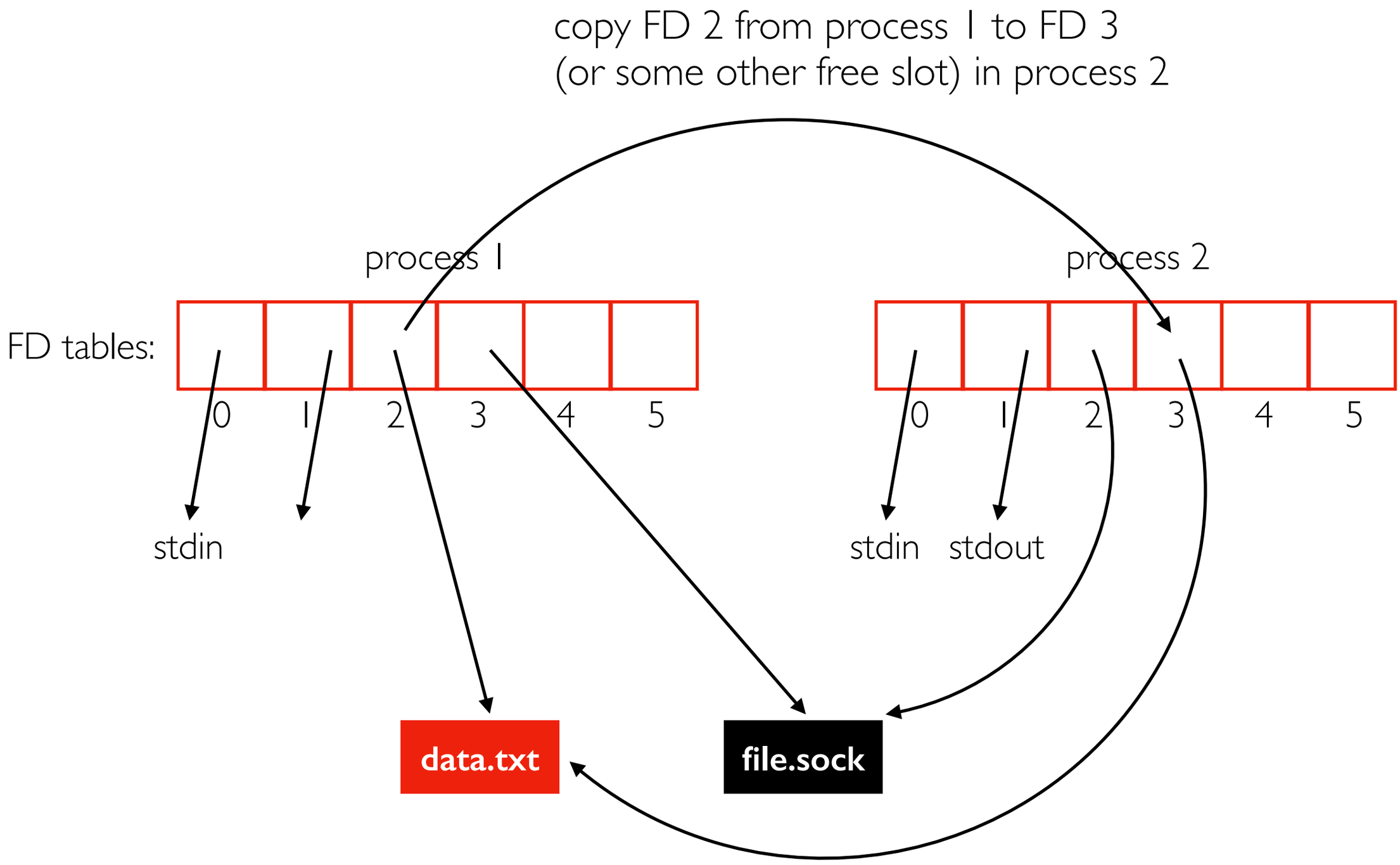


File Descriptor Tables

say process 1 wants to give process 2 access to data.txt



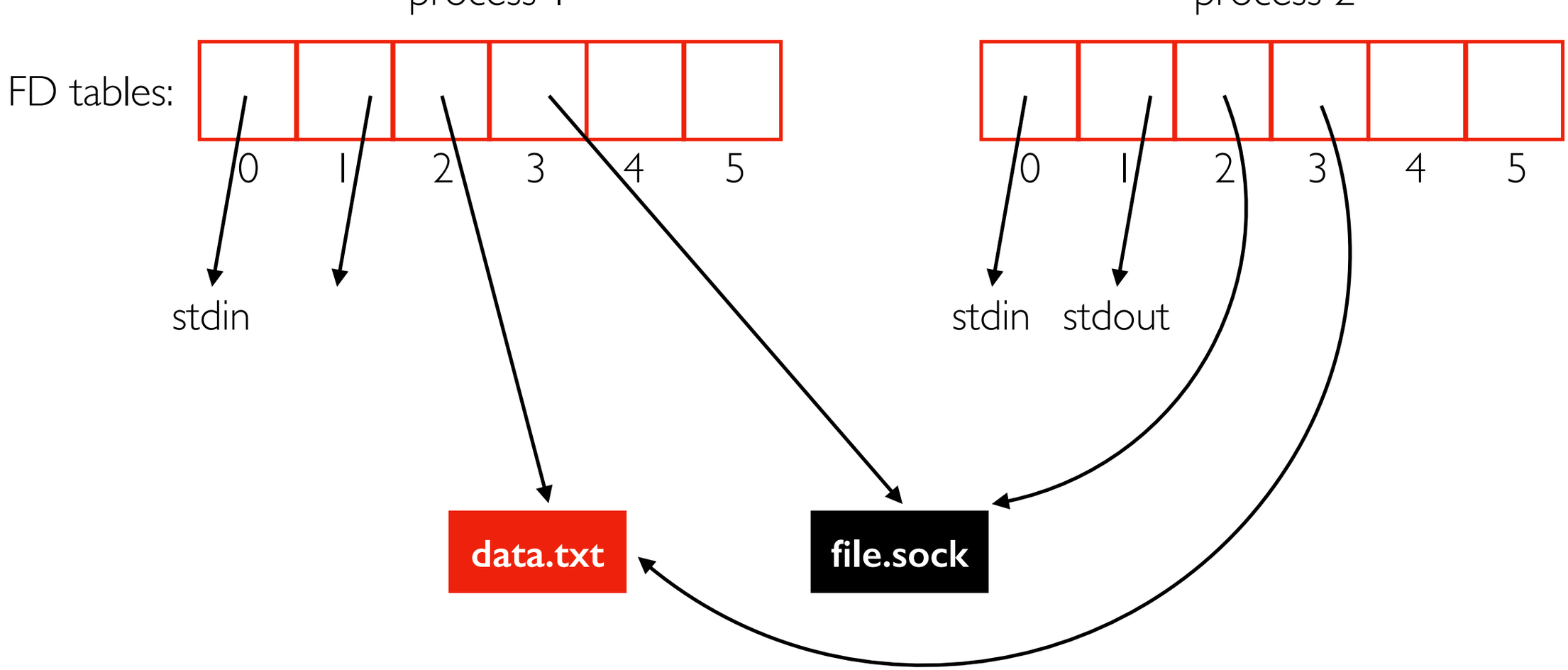
File Descriptor Tables



File Descriptor Tables

```
// process 1:
// to read data.txt
rc = read(2, buf, count);
```

```
// process 1:
// to read data.txt
rc = read(3, buf, count);
```



Practice...