sysjail

systrace user-land virtualisation

New York City BSD Conference
October, 2006

Kristaps Džonsons

"I tell ya Park Avenue leads to Skid Row… "

table of contents

what is sysjail?

- sysjail is a **FreeBSD jail**[1] system for OpenBSD and NetBSD

[1] docs.freebsd.org/44doc/papers/jail/jail.html

what is sysjail?

- sysjail is a **systrace user-land virtualiser**
- jailed processes
  - have **filtered access to the host's resources**
  - operate in a **file-system chroot**

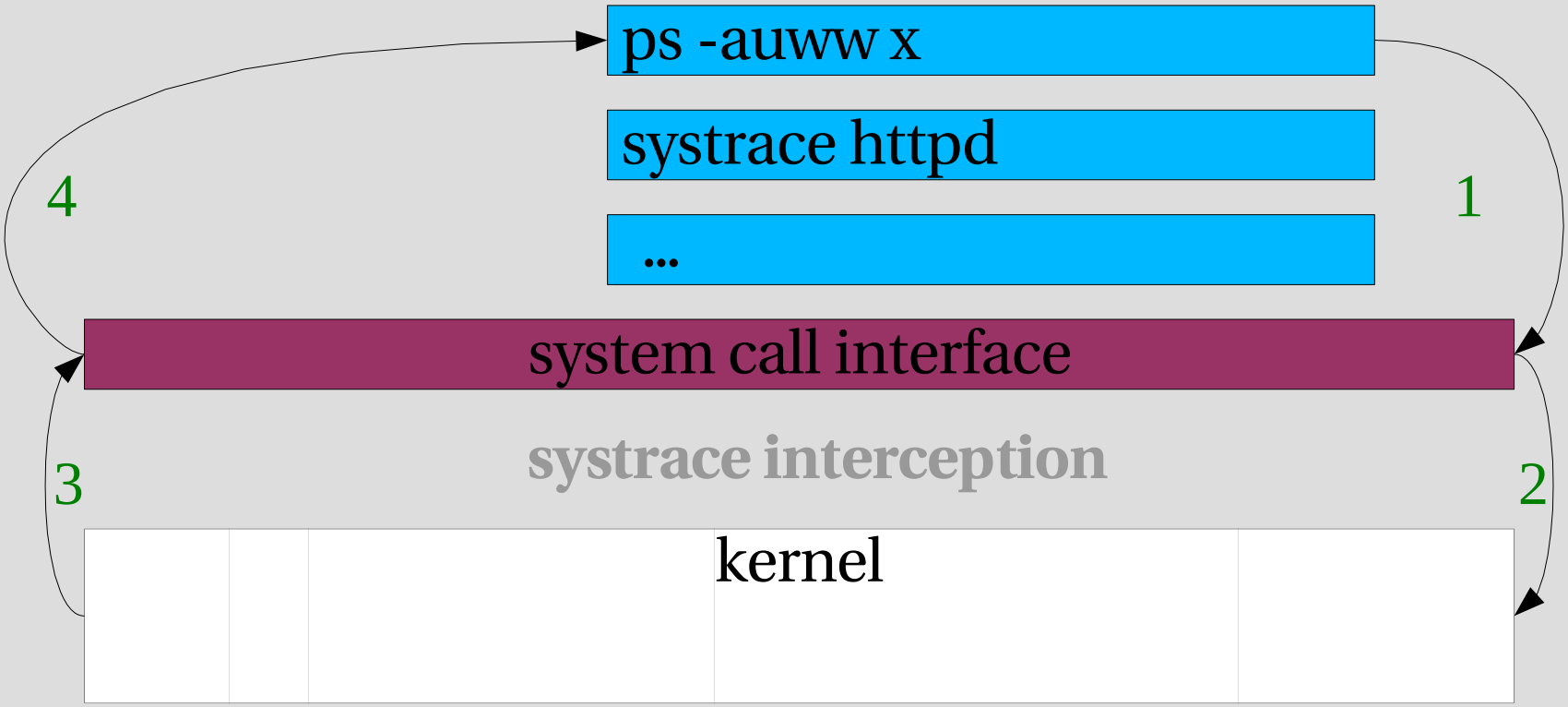
what is sysjail?

- sysjail is a **systrace user-land virtualiser**

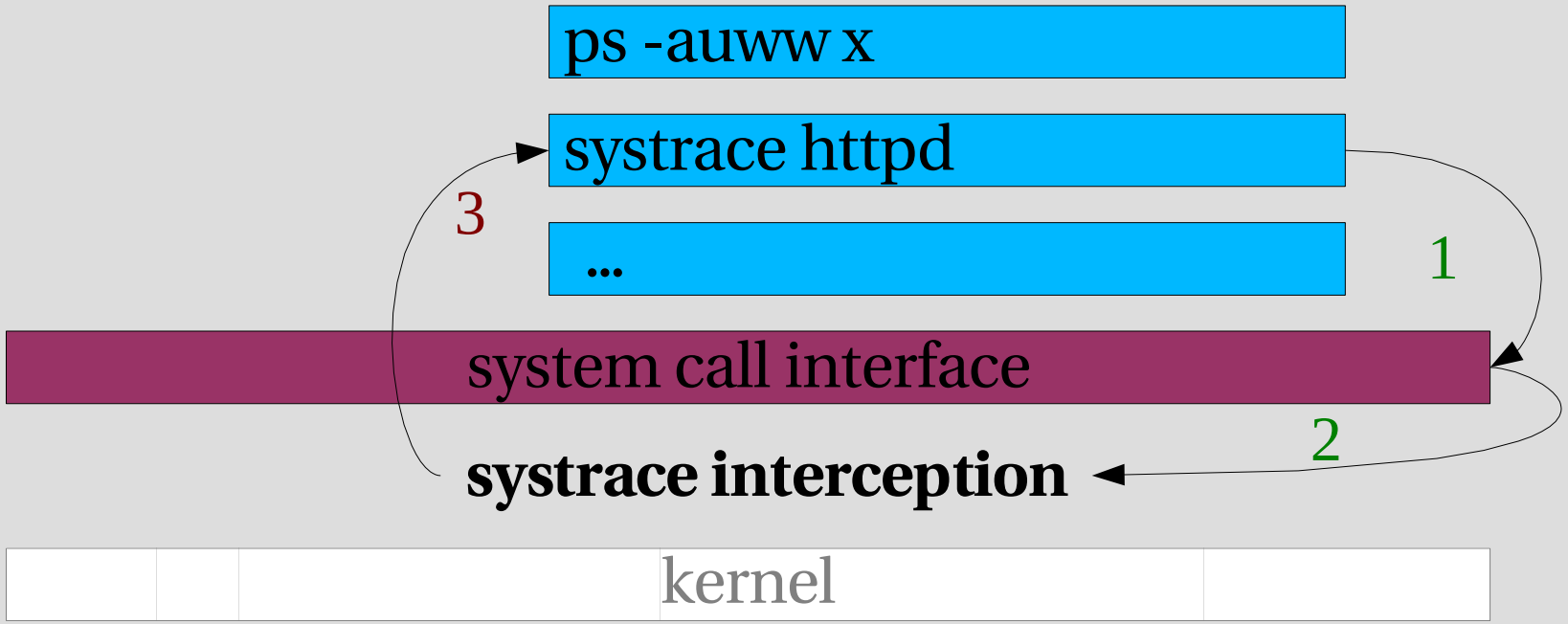
- systrace
  uses `systrace(4)` device:
  "attaches to processes and enforces policies for system calls"[1]
- user-land
  runs in user-mode: external to the kernel
- virtualisation
  processes in jail disallowed access to host resources

[1] systrace(4) OpenBSD 3.9 manual page

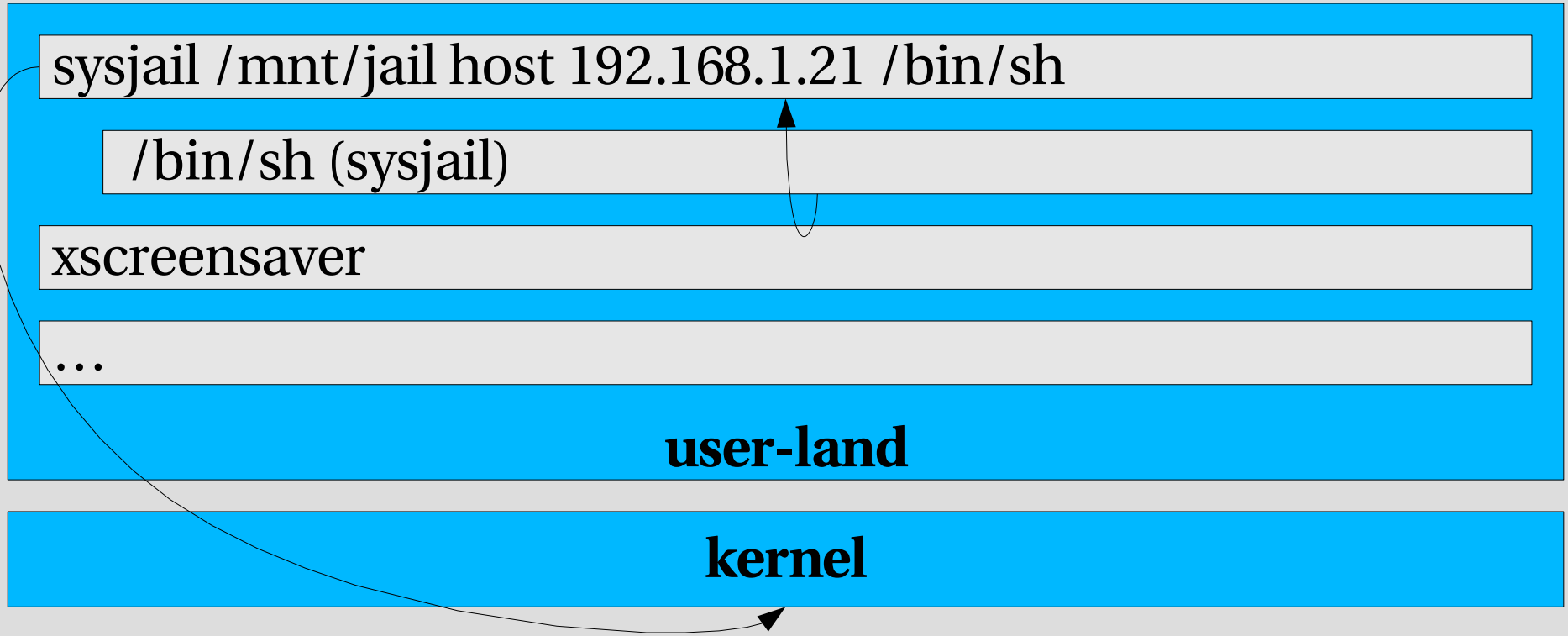
systrace(4)


ps -auww x
systrace httpd
...
4
1
system call interface
systrace interception
3
2
kernel

systrace(4)

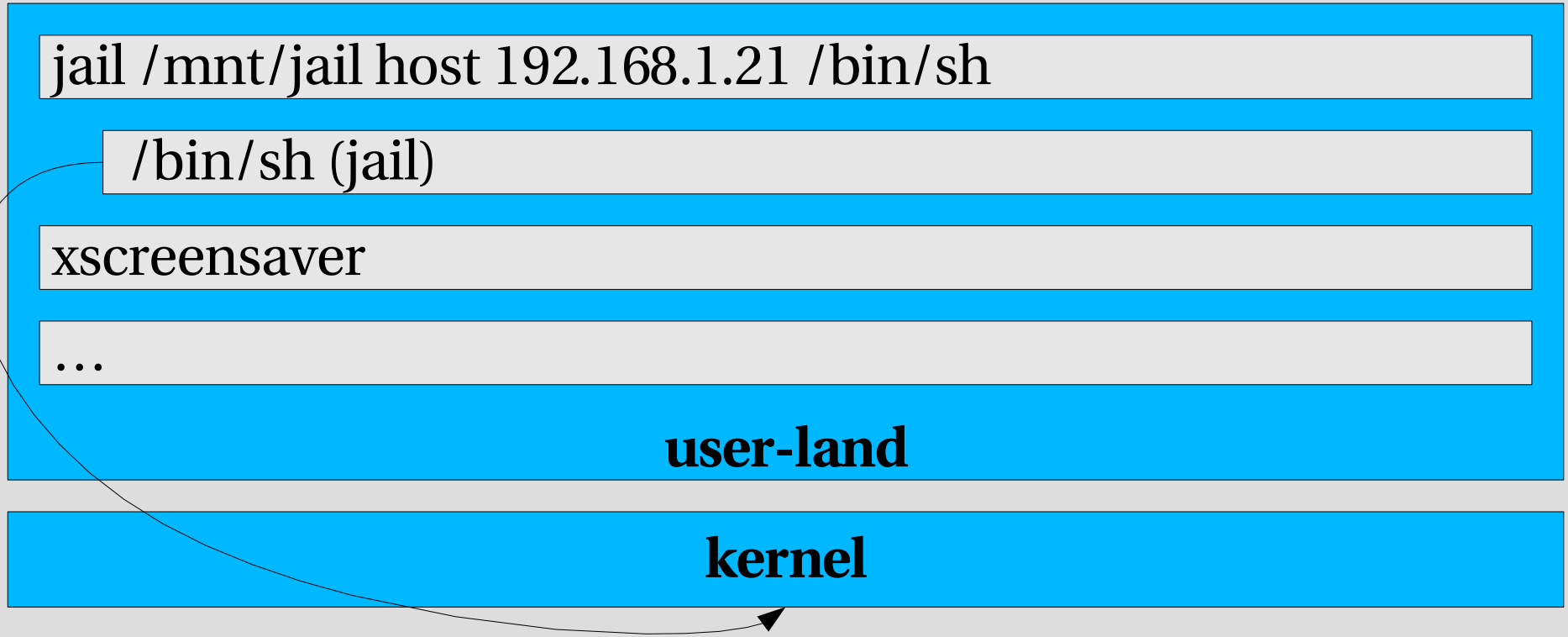

ps -auww x
systrace httpd
...
3
1
system call interface
2
systrace interception
kernel

user-land



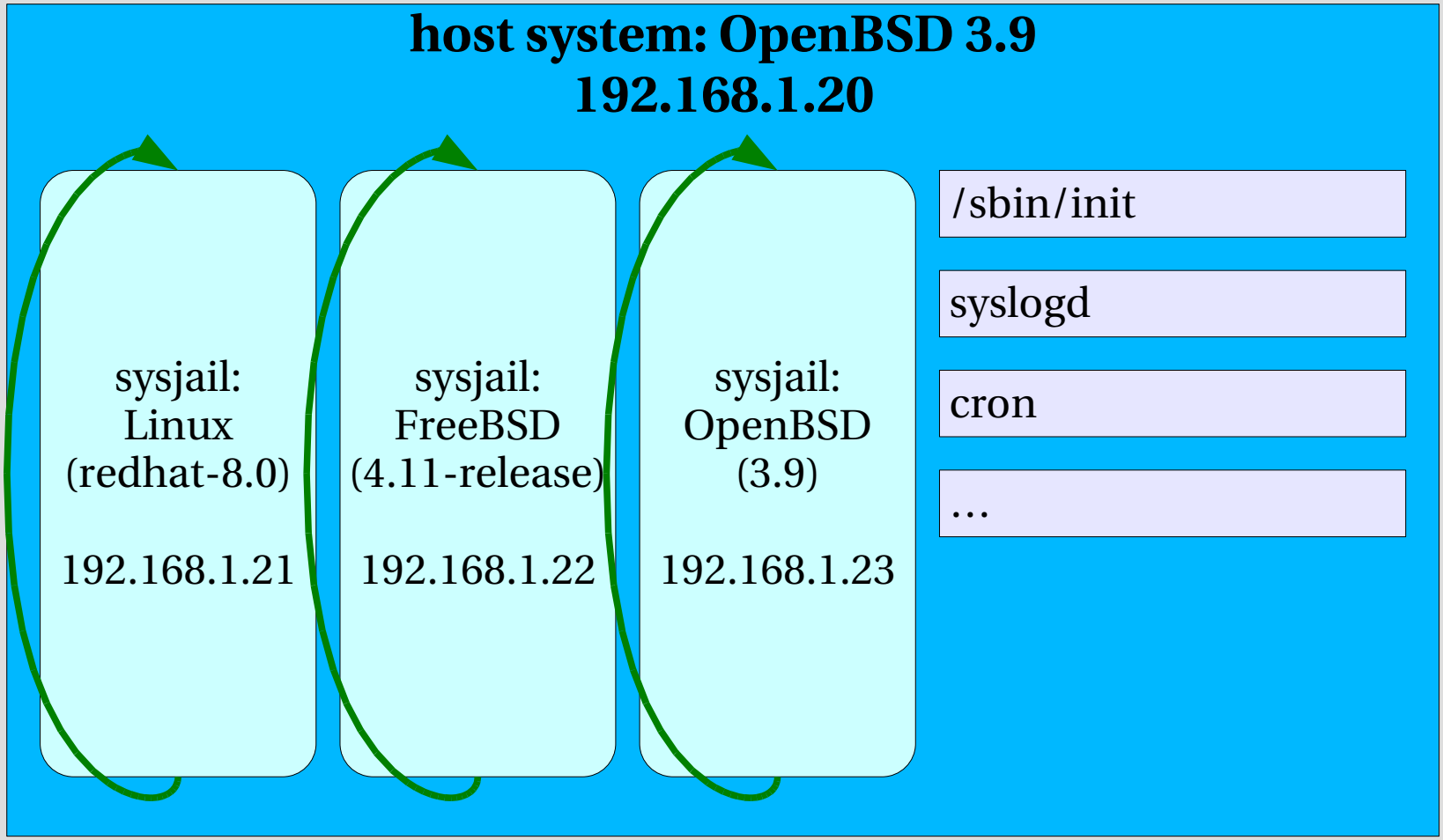
user-space control: sysjail

user-land

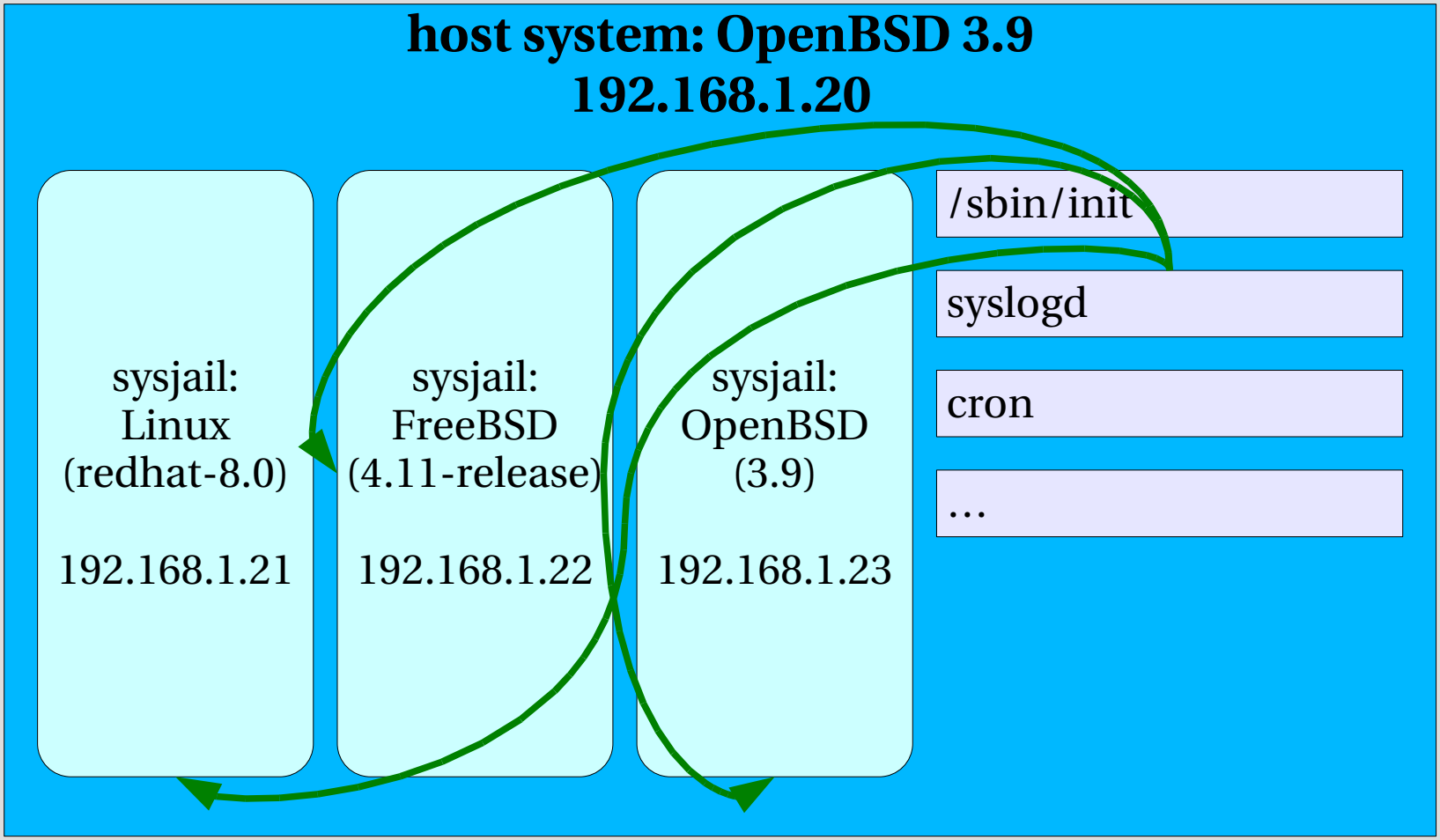


(versus) kernel-space control: FreeBSD jail

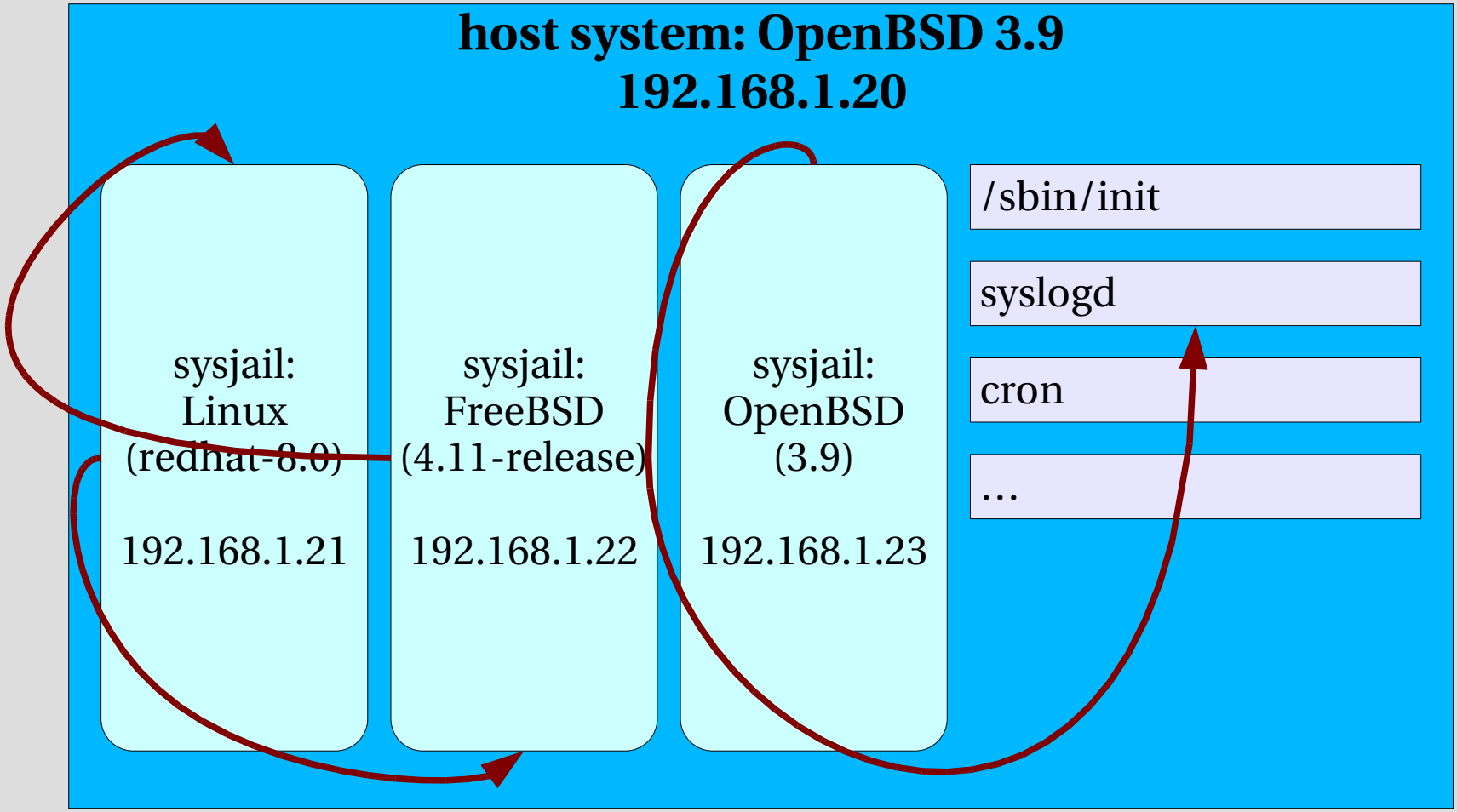
virtualisation



virtualisation



virtualisation



what isn't sysjail?

- not a full-system virtualiser
  - only system calls are intercepted (versus Xen, UML etc.)
- not fast
  - at times, significant overhead in interception
- not the "correct" solution
  - the problem domain will always be bigger
  - every operating system update requires consideration
  - **systems shouldn't need protective enclosure!**

so, why use sysjail?

- fast *enough*
  - overhead is not noticeable in most applications
- easy to modify
  - user-space applications may be modified without needing kernel recompilation
- easy to extend
  - code is architected with extension in mind
- useful
  - per-user "systems", process containment etc.

sysjail support

- OpenBSD 3.9
  - no reported issues
- NetBSD 3.0
  - has unresolved issue with cobalt/pmax architectures
- Linux 2.4.21, 2.6.1
- Mac OS 10.2
- FreeBSD 4.5, 5.2
  - not yet ported (systrace support still beta)

(as of October, 2006)

sysjail operation

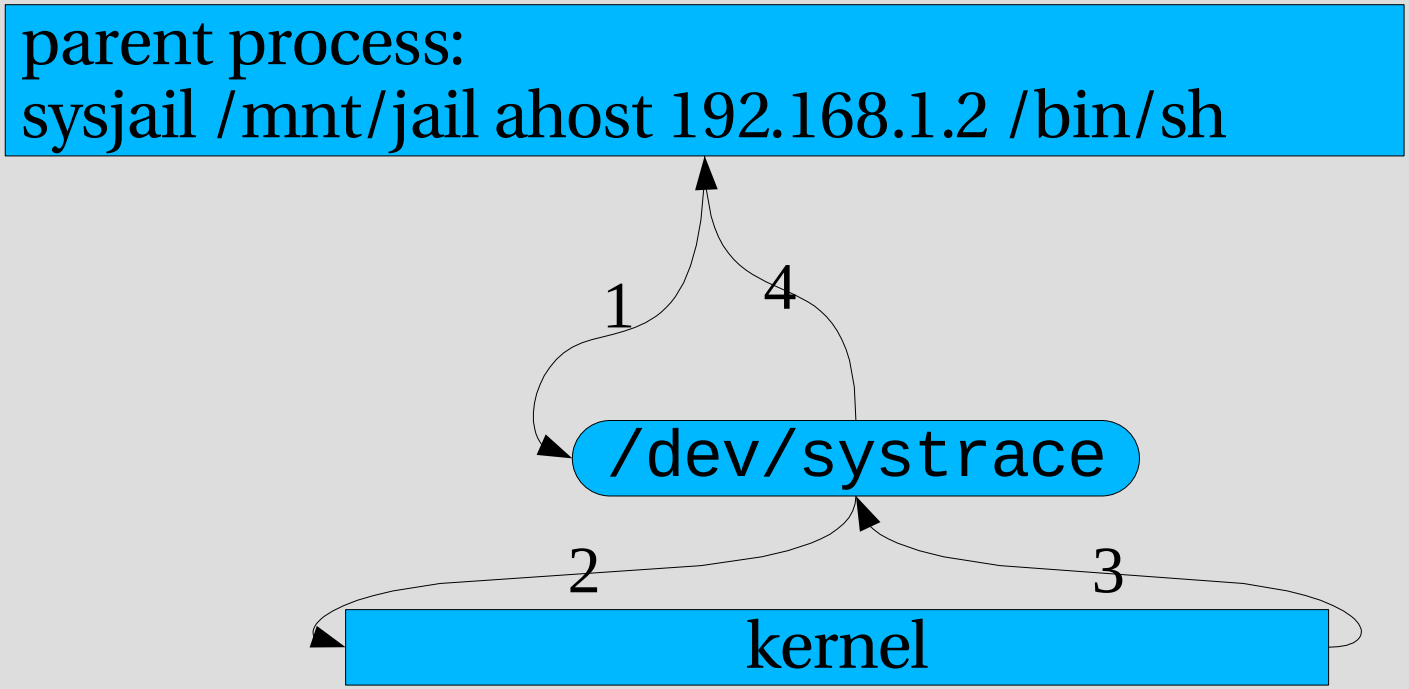
1. "Parent" sysjail image systrace profile.
2. Parent forks "master" child.
3. Parent binds master to policy.
4. Master chroots, sheds privileges.
5. Parent loops on system call interceptions.

sysjail operation

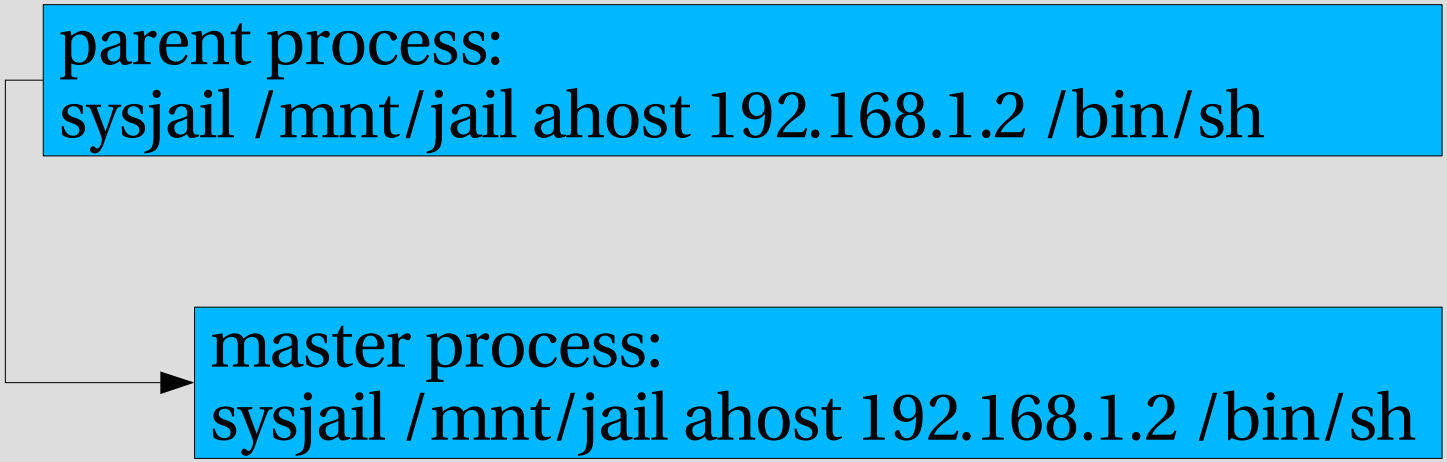
parent process:
sysjail /mnt/jail ahost 192.168.1.2 /bin/sh

pre-initialisation

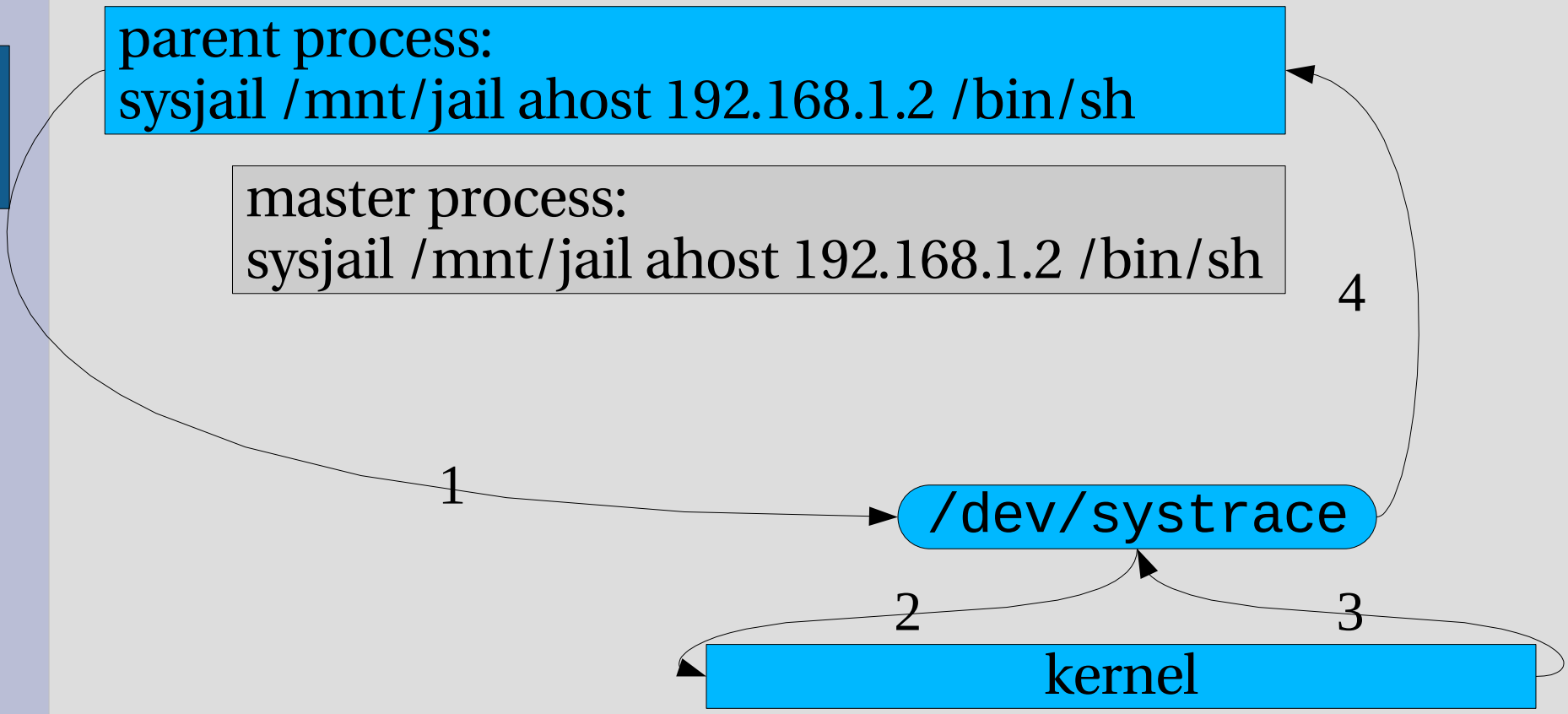
sysjail operation



sysjail operation

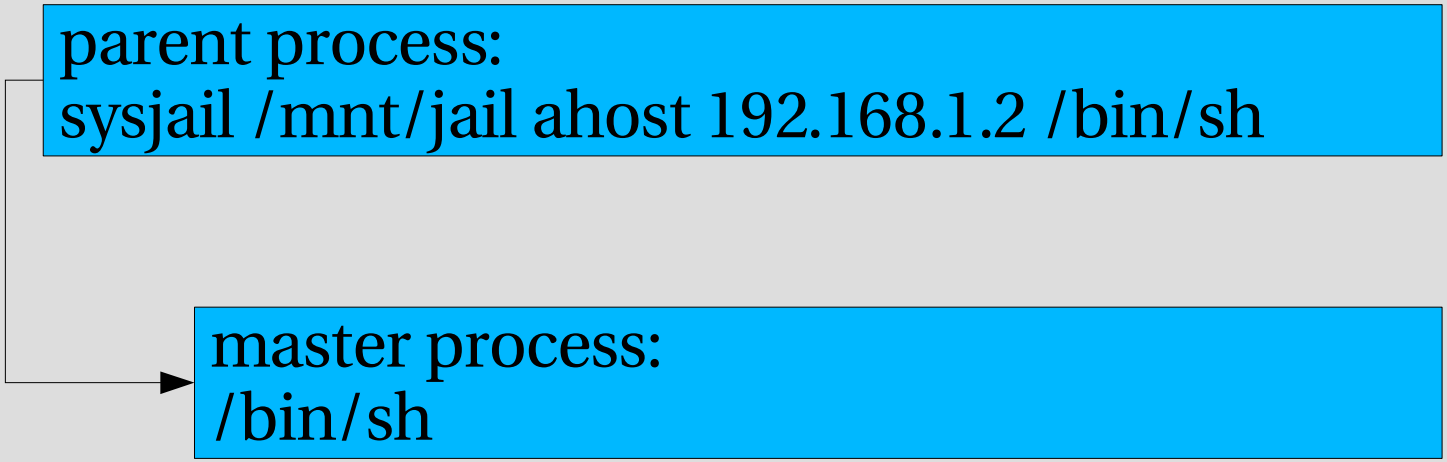


sysjail operation



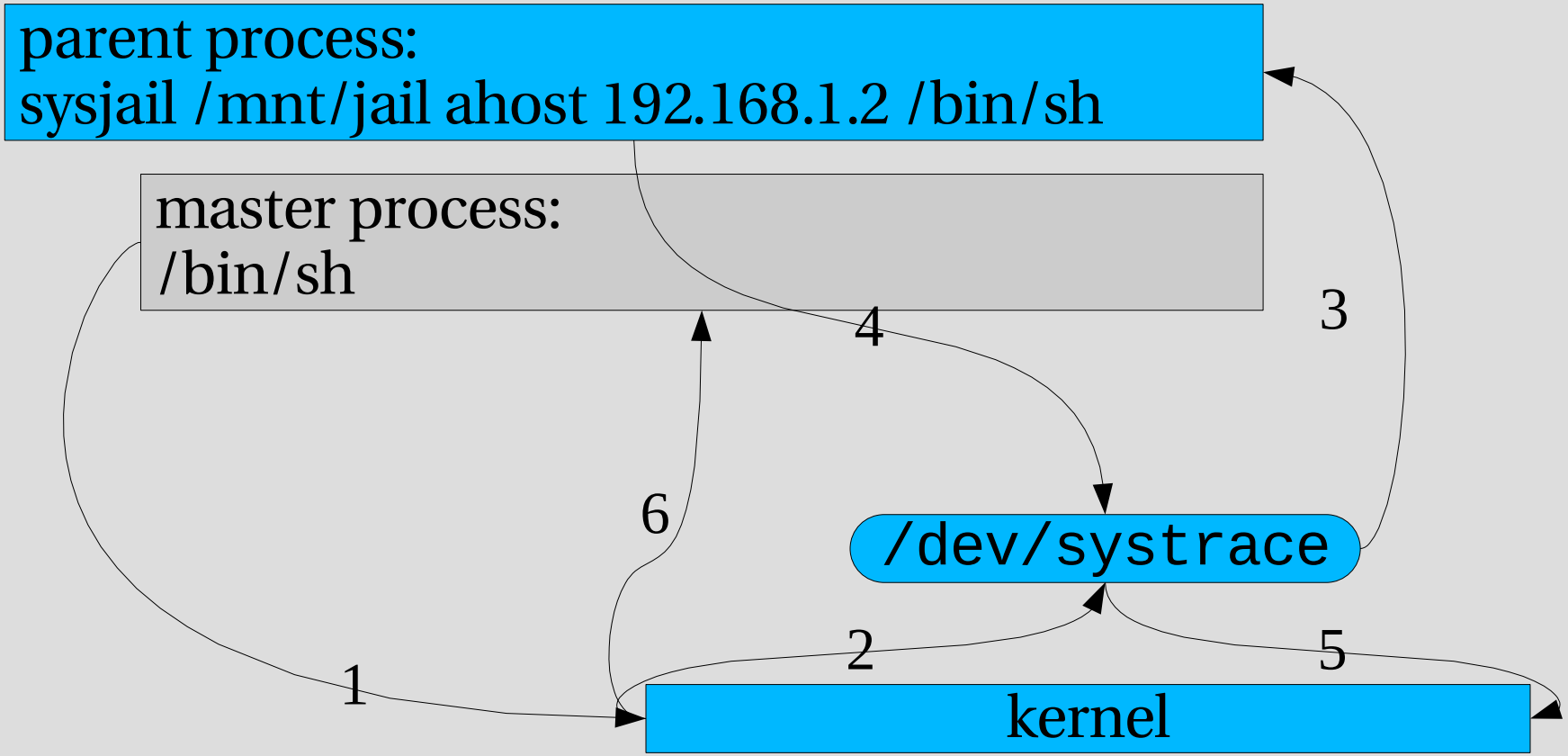
systrace binding (3)

sysjail operation



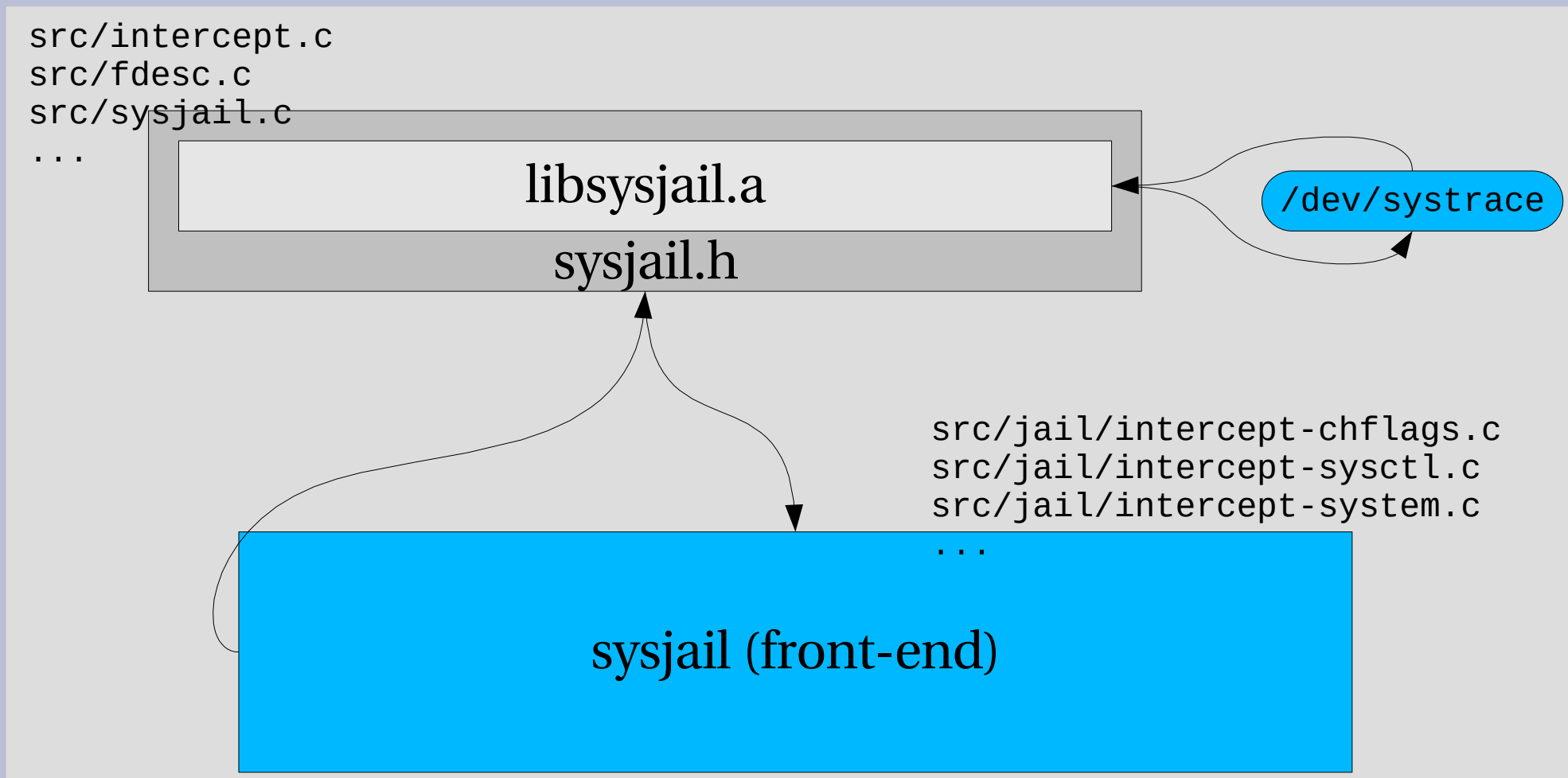
master sheds, execs (4)

sysjail operation



system call interception (4)

sysjail architecture

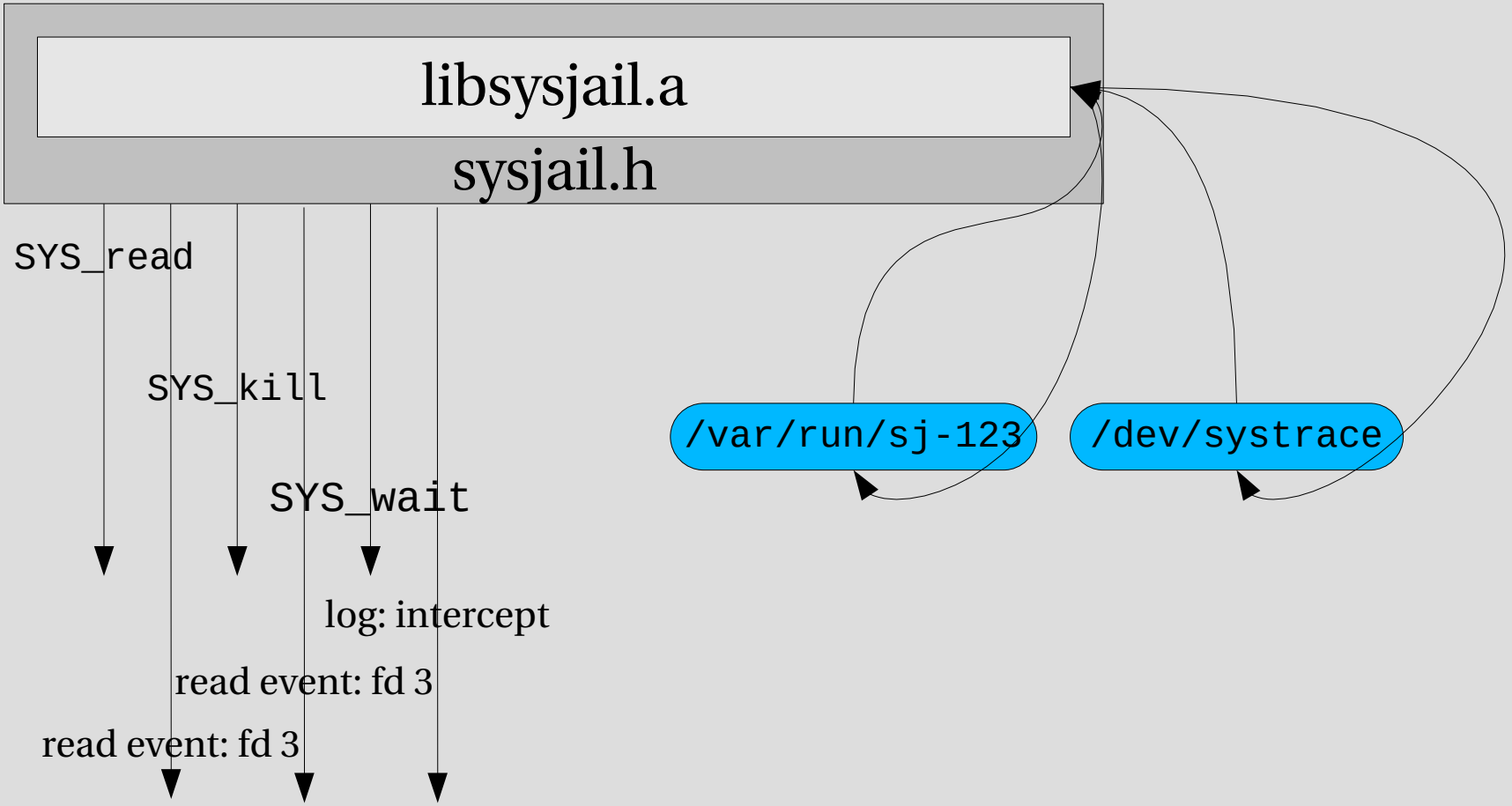


sysjail architecture

- ./src/* (libsysjail.a, sysjail.h)
  - interception of file descriptor events
  - interception of system calls

  - management of log & report data
  - management of process tree
  - management of intercept handling

sysjail architecture



sysjail architecture

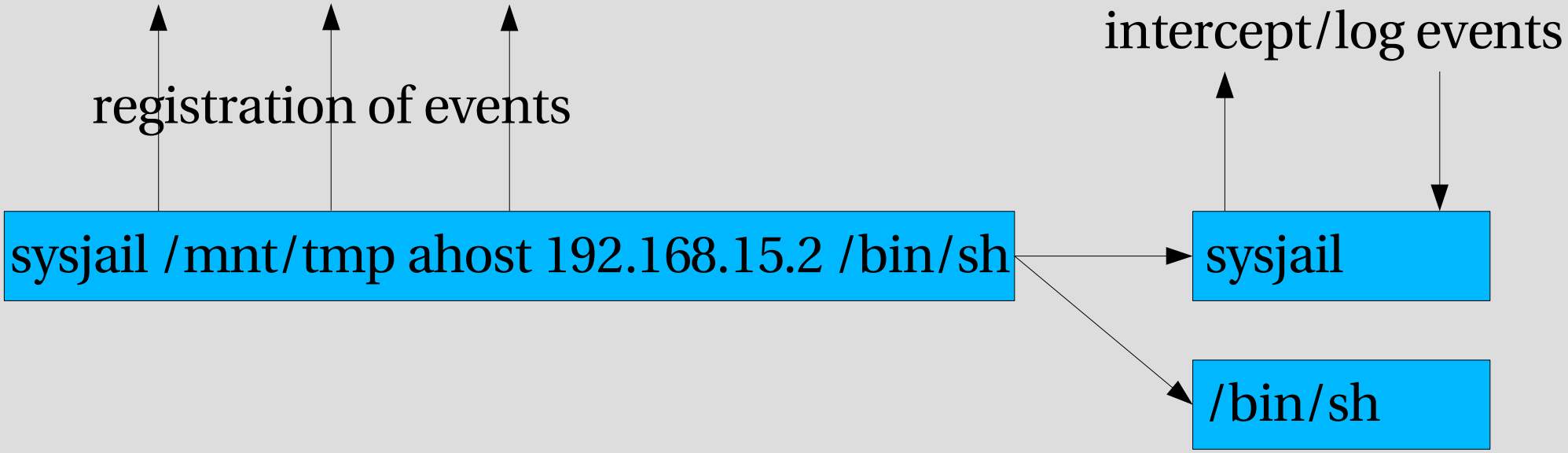
- ./src/jail/*

  FreeBSD jail-like "front-end" to sysjail system

  - registration of file descriptor events
  - registration of system calls

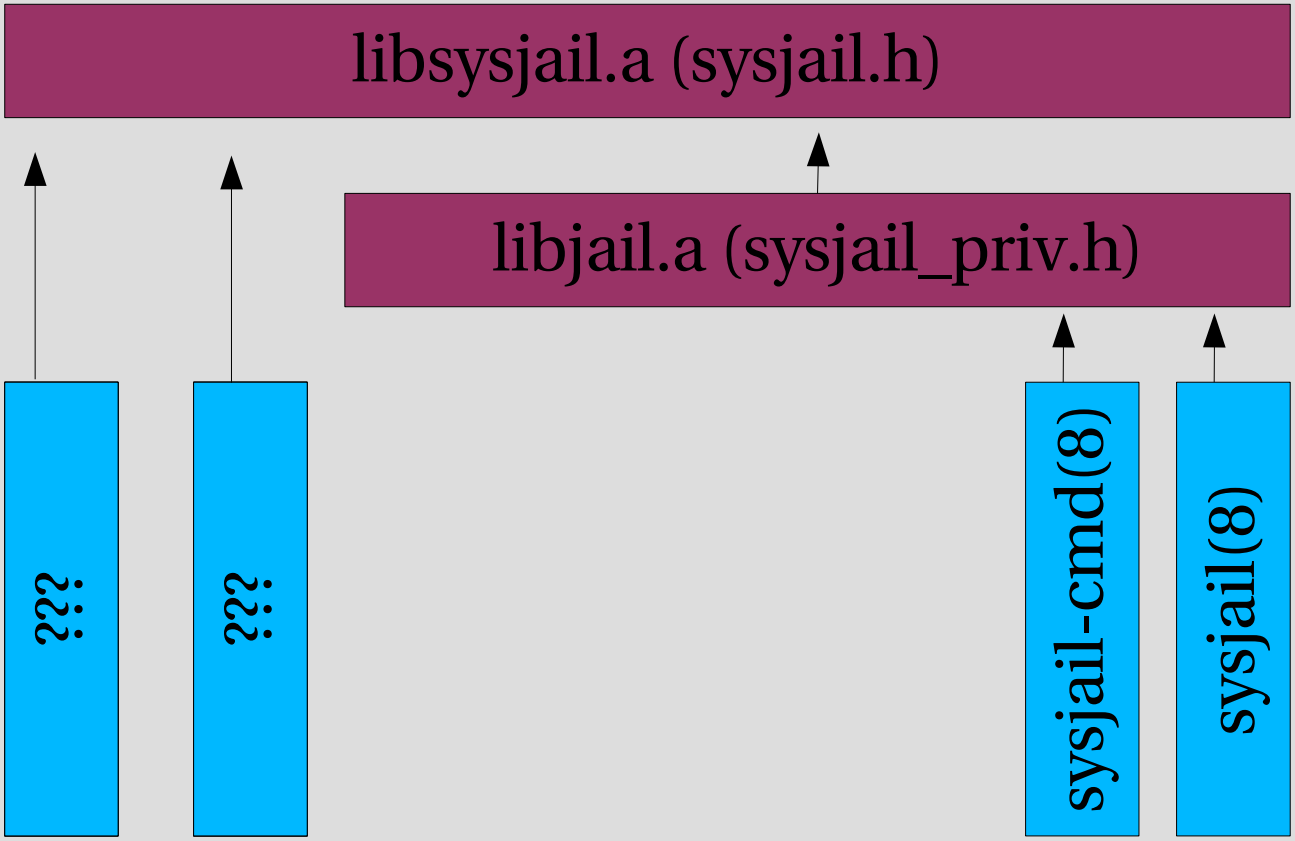
  - handling report & log messages
  - handling of intercept messages

sysjail architecture



sysjail architecture

- jail front-end itself has a backing library
- allows for multiple sysjai/jail-like systems



sysjail implementation

Case study: a simple sysjail implementation.

- full implementation of a jail
- uses sysjail version 1.0.4 (old)
- runs `/bin/echo` from within the jail

sysjail implementation

Source file: example.c

```
1.    #include <sys/param.h>
2.    #include <dev/systrace.h>
3.    #include <sys/syscall.h>
4.
5.    #include <err.h>
6     #include <stdlib.h>
7.    #include <string.h>
8.    #include <unistd.h>
9.
10.   #include "sysjail.h"
```

sysjail implementation

Source file: example.c

```
1.    #include <sys/param.h>
2.    #include <dev/systrace.h>
```

Necessary inclusions for systrace- on NetBSD, the location would be `<sys/systrace.h>`.

```
10.   #include "sysjail.h"
```

The main sysjail header file.

sysjail implementation

Source file: example.c

```
12.   int do_inter(const struct sysjail *jail, int fd,
13.       struct str_message *msg, int *err, int *errf,
14.       int emul)
15.   {
16.       /* Act on system call! */
17.   }
18.
19.   int do_init_syst(const struct sysjail *jail, struct
20.       sj_inter_args *args)
21.   {
22.       if (args[SYS_kill].ifp != NULL)
23.           return(-1);
24.       args[SYS_kill].ifp = &do_inter;
25.       return(0);
26.   }
```

sysjail implementation

Source file: example.c

```
12.   int do_inter(const struct sysjail *jail, int fd,
13.       struct str_message *msg, int *err, int *errf,
14.       int emul)
15.   {
16.       /* Act on system call! */
17.   }
```

Call-back to actually perform the interception. In the sysjail application, these functions (per class of interception, e.g., those acting on the process-tree jail) filter the process's request.

sysjail implementation

Source file: example.c

```
19.   int do_init_syst(const struct sysjail *jail, struct
20.       sj_inter_args *args)
21.   {
22.       if (args[SYS_kill].ifp != NULL)
23.           return(-1);
24.       args[SYS_kill].ifp = &do_inter;
25.       return(0);
26.   }
```

Call-back to initialise the intercept table. This sets the intercept of `SYS_kill` to the function `do_inter`.

sysjail implementation

Source file: example.c

```
28.   int main(void) {
29.       struct sysjail j;
30.       strlcpy(j.hostname, "jail", sizeof(j.hostname));
31.       strlcpy(j.path, "/tmp/jail/", sizeof(j.path));
32.       j.version = 0;
33.       j.ip_number = 0;
34.       if (sysjail(&j) == -1)
35.           errx(1, "Unable to execute jail.");
36.       execl("/bin/echo", "/bin/echo", "foo", NULL);
37.       err(1, "execl");
38.   }
```

sysjail implementation

Source file: example.c

```
29.       struct sysjail j;
```

Main structure for sysjail implementations:

```
struct sysjail {
    char hostname[MAXHOSTNAMELEN];
    char path[PATH_MAX];
    u_int32_t jid;
    u_int32_t version;
    u_int32_t ip_number;
    pid_t master;
};
```

sysjail implementation

Source file: example.c

```
30.      strlcpy(j.hostname, "jail", sizeof(j.hostname));
31.      strlcpy(j.path, "/tmp/jail/", sizeof(j.path));
32.      j.version = 0;
33.      j.ip_number = 0;
```

Initialises the structure to some bogus values:

`j.hostname` `"jail"`
: Virtual hostname of jailed process tree.

`j.path` `"/tmp/jail/"`
: Path root of jailed file-system.

`j.version` `0`
: Version of sysjail (not used).

`j.ip_number` `0`
: IP number in host-long format.

sysjail implementation

Source file: example.c

```
34.       if (sysjail(&j) == -1)
35.           errx(1, "Unable to execute jail.");
```

Execute the sysjail as given. Returns control as the child (“master„) process; callbacks (for example, when child processes fork, or when debugging messages are raised) occur in the parent context.

This odd behaviour is a result of FreeBSD jail conformance, and will likely be later abandoned in favour of a cleaner interface.

sysjail implementation

Source file: example.c

```
36.       execl("/bin/echo", "/bin/echo", "foo", NULL);
37.       err(1, "execl");
```

Child context. Replaces current process image with that of echo. It is assumed that these executables exist under the jailed file-system root.

sysjail implementation

Compiling & running:

- compile

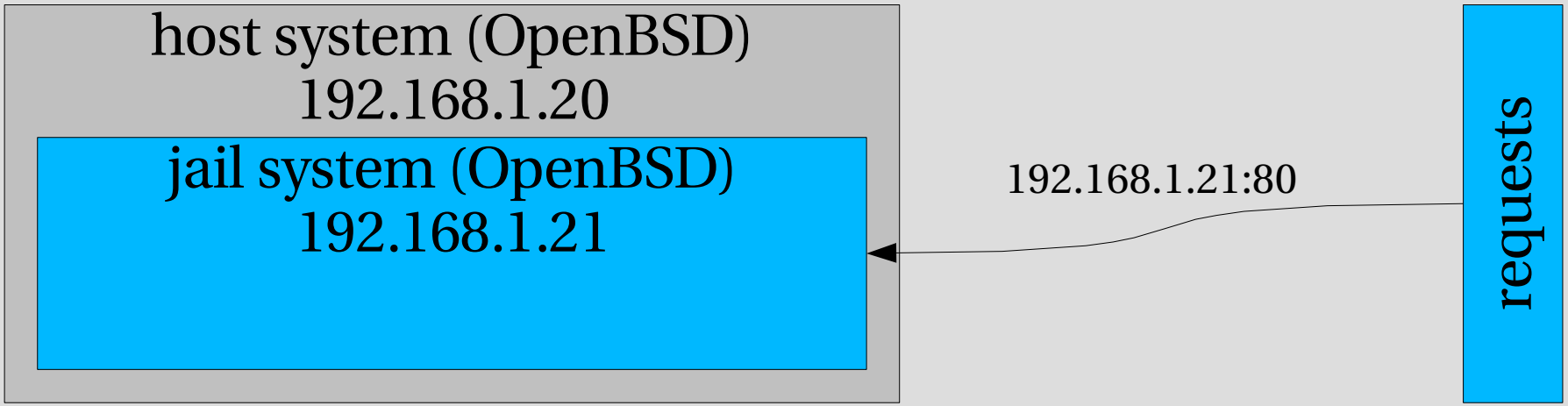
```
$ gcc -I/usr/local/include/sysjail sysjail.c \
   /usr/local/lib/libsysjail.a
```

- run

```
$ sudo ./a.out
```

sysjail example

- simple sysjail example
  - tailoring an OpenBSD system, with httpd, to be jailed



sysjail example

- create file-system root
- add necessary devices to file-system root
- modify jailed configuration files
- add to host's startup file
- start sysjail

sysjail example

- create file-system root

Copy binaries from an OpenBSD mirror by distributed script:

```
# mksysjail-base -s ftp.openbsd.org \
    /pub/OpenBSD/`uname -r`/`uname -m`/ /tmp/jail/
```

Or copy binaries directly:

```
# wget ftp://ftp.openbsd.org/pub/OpenBSD/3.9/i386/\*.tgz
# tar -zvxf *.tgz -C /tmp/jail/
```

sysjail example

- add necessary devices to file-system root

You may use the distributed script:

```
# mksysjail-dev -c /tmp/jail/ https
```

Or you may edit the files by hand:

```
# cd /tmp/jail/dev
# sh ./MAKEDEV std random
```

sysjail example

- modify jailed configuration files

Since we're only interested in http(s), we only need concern ourselves with system startup:

  - `/tmp/jail/etc/rc.conf.local`

sysjail example

- modify jailed configuration files
  - `/tmp/jail/etc/rc.conf.local`

```
1.    sshd_flags=NO
2.    sendmail_flags=NO
3.    inetd=NO
4.    httpd_flags=""
```

sysjail example

- modify jailed configuration files
  - `/tmp/jail/etc/rc.conf.local`

```
1.    sshd_flags=NO
```

Disables the ssh server.

sysjail examples

- modify jailed configuration files
  - `/tmp/jail/etc/rc.conf.local`

```
2.    sendmail_flags=NO
```

Disables the sendmail server.

sysjail example

- modify jailed configuration files
  - `/tmp/jail/etc/rc.conf.local`

```
3.    inetd=NO
```

Disables the inetd super-server.

sysjail example

- modify jailed configuration files
  - `/tmp/jail/etc/rc.conf.local`

```
4.    httpd_flags=""
```

Instructs the http daemon to start without arguments.

sysjail example

- add to host's startup file
  - /etc/rc.conf.local

```
1.    if [ -x /usr/local/sbin/sysjail ]
2.    then
3.       echo -n ' sysjail'
4.       /sbin/ifconfig vr0 inet 192.168.1.21 netmask \
5.          255.255.255.0 alias
6.       /usr/local/sbin/sysjail --quiet --allow-sysvipc \
7.          --cmd-enable /tmp/jail/ jail 192.168.1.21 \
8.          /bin/sh /etc/rc </dev/null &
9.    fi
```

sysjail example

- add to host's startup file
  - `/etc/rc.conf.local`

```
4.        /sbin/ifconfig vr0 inet 192.168.1.21 netmask \
5.           255.255.255.0 alias
```

Configures a network alias (we use vr0 as the example device). One may, of course, have a dedicated device.

sysjail example

- add to host's startup file
  - `/etc/rc.conf.local`

```
6.        /usr/local/sbin/sysjail --quiet --allow-sysvipc \
7.           --cmd-enable /tmp/jail/ jail 192.168.1.21 \
8.           /bin/sh /etc/rc </dev/null &
```

Starts jail with suppressed output, SVID IPC (required by https) and a command pipe (for statistics).

sysjail example

- start the jail

```
# sysjail -i --quiet --allow-sysvipc --cmd-enable \
    /tmp/jail/ jail 192.168.1.21 /bin/sh /etc/rc </dev/null &
732918629
#
# sjls
      JID  IP Address      Hostname         Path
732918629  192.168.1.21    jail             /tmp/jail/
# netstat -a
Proto Recv-Q Send-Q Local Address    Foreign Address (state)
tcp   0      0      192.168.1.21.www *.*             LISTEN
[snipped]
```

current implementation

- sysjail has reached version 1.1 in time for the NYCBSDCon 2006
  - re-write of logging mechanism
  - framework support for binary emulation
  - unstable support for Linux emulation
  - clean separation of front- and back-ends

current implementation

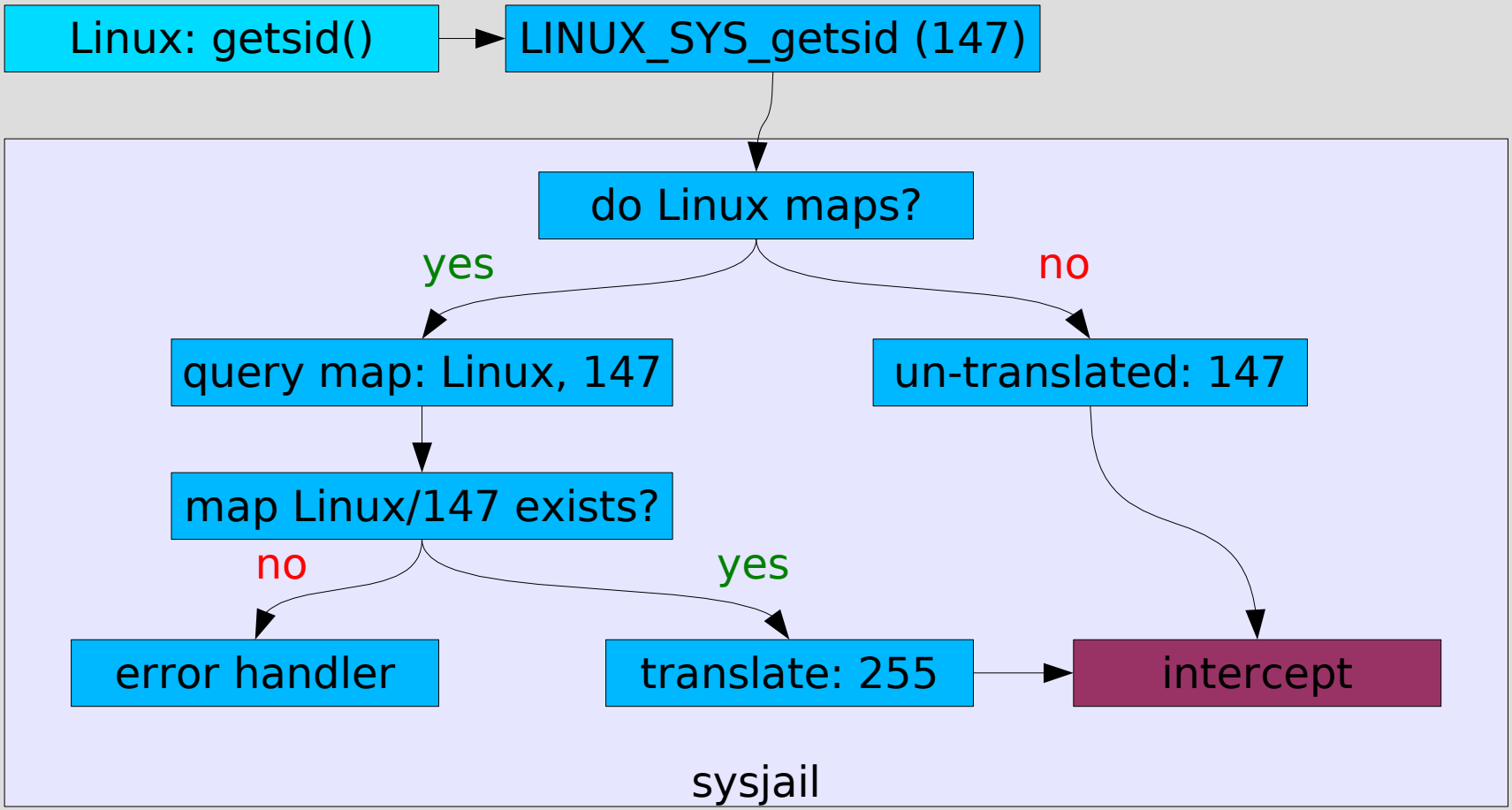
- re-write of logging mechanism
  - log messages passed to front-end implementations via call-backs and message structures, not formatted strings, allowing front-ends to specify
    - localisation
    - different output formats (XML, ...)
    - different output locations (socket, file, ...)
- framework support for binary emulation
- unstable support for Linux emulation
- clean separation of front- and back-ends

current implementation

- re-write of logging mechanism
- framework support for binary emulation
  - when adding call-backs for system calls, one may specify which are *mapped*, for example:
    - `LINUX_SYS_getsid` (147) would map to `SYS_getsid` (255)
    - intercept routines in sysjail front-end are now emulation-intelligent (there are often differences in system call parameters and behaviour)
- unstable support for Linux emulation
- clean separation of front- and back-ends

current implementation

- binary emulation sequence



current implementation

- re-write of logging mechanism
- framework support for binary emulation
- unstable support for Linux emulation
  - the Linux system call layer has been partially implemented within sysjail's intercepts
  - problems arise due to lack of clear documentation
- clean separation of front- and back-ends

current implementation

- re-write of logging mechanism
- framework support for binary emulation
- unstable support for Linux emulation
- clean separation of front- and back-ends
  - the front-end and back-end libraries now communicate exclusively through `sysjail.h`
  - before there was a mix of header files common to both codebases
  - this makes the task of writing new front-ends significantly easier

current implementation

- status: host system
  - NetBSD: no
  - OpenBSD: yes
- status: target system
  - Linux: unstable
  - FreeBSD: no
  - HP-UX, ...: no

future work: sysjail back-end

- most important: binary emulation
  - imagine being able to host FreeBSD, NetBSD, Linux, SunOS etc. system roots on a single OpenBSD machine
  - significant work required to research parameters and behaviour of per-system system calls
- testing: a per-emulation testing system should be in place to make sure that all systems behave properly

future work: sysjail front-end

- `sysjail-cmd`
  - extended command utilities
  - dynamic switches between permitting & logging all other system calls (beyond those in the current profile)
- `sysjail`
  - sending log data to sockets
  - having a sysjail log server to show real-time status and intercepts
  - sending intercept routines through sockets to a central server
    - ability to coördinate multiple sysjail systems in parallel: useful for testing network systems
  - ability to understand `systrace(1)` policy files

sysjail examples

- (on-line examples)

credits & acknowledgements

- NYCBSDCon 2006 coördinators
  for the conference and my sponsorship
- FreeBSD, NetBSD and OpenBSD
  for their fine operating systems
- Maikls Deksters
  for his leadership and management
- sysjail contributors
  for their bug-reports and comments

end

Copyright © 2006, Kristaps Džonsons

Redistribution of these slides in any form, with or without modifications, are permitted provided that the above copyright notice is retained.

Riga, Latvija, 2006