What's the best practice for implementation of rumpclient?

Kazuya GODA
<K-goda@iij.ad.jp>

Agenda

- Introduce
- Preliminary knowledge
  - rumpkernel/client, OpenFlow and switch(4)/switch(8)
- Why I took rump kernel to develop switch(4)
- How to implement rump kernel client
- Let's practice
- Conclusions

Who am I?

- Kazuya GODA
- Work at IIJ as software engineer
  - I’ve worked on SEIL team , using NetBSD, and Tornado team, using OpenBSD
- OpenBSD developer <goda@openbsd.org>
  - But I’ll only talk about NetBSD today

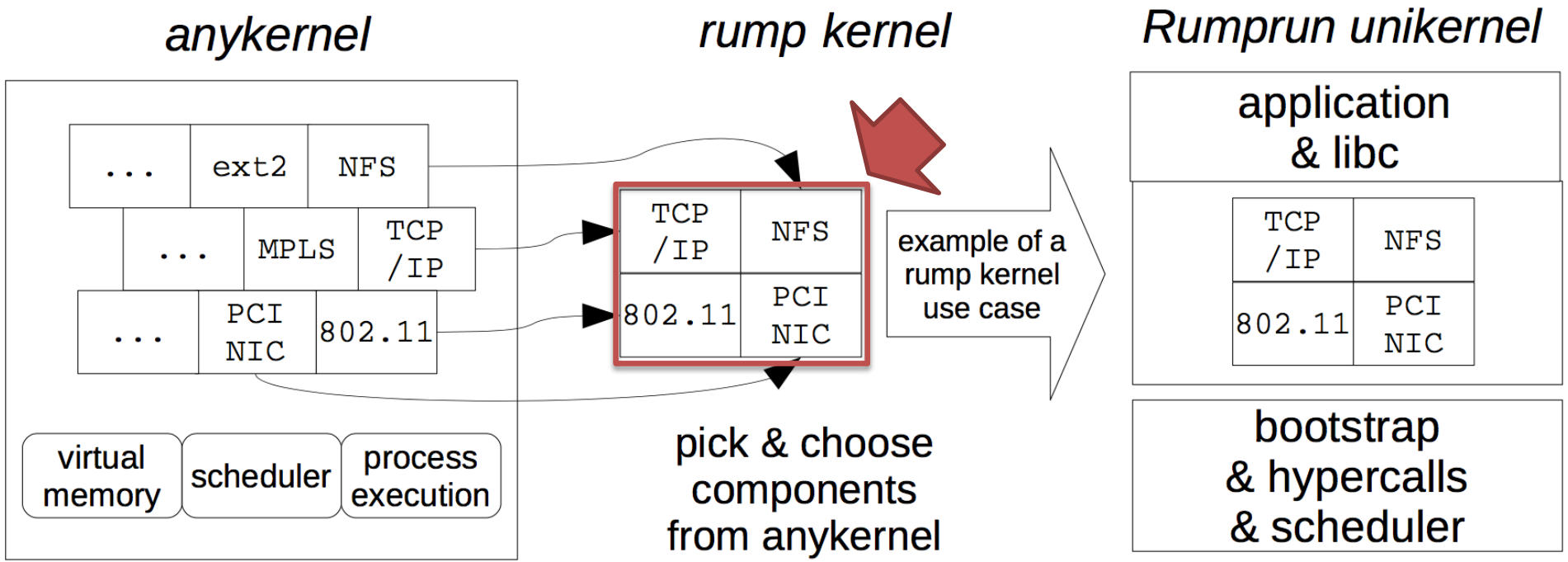
Introduction

- I'm porting switch(4)/switch(8) from OpenBSD
  - switch(4) is an implementation of OpenFlow switch
- I've used rumpkernel to develop it
  - I have to work not only switch(4) but also switch(8) in rump
    - In other word, I have to implement switch(8) as rumpclient
- I've gotten some knowledges from this work
  - I'll share it with you

Agenda

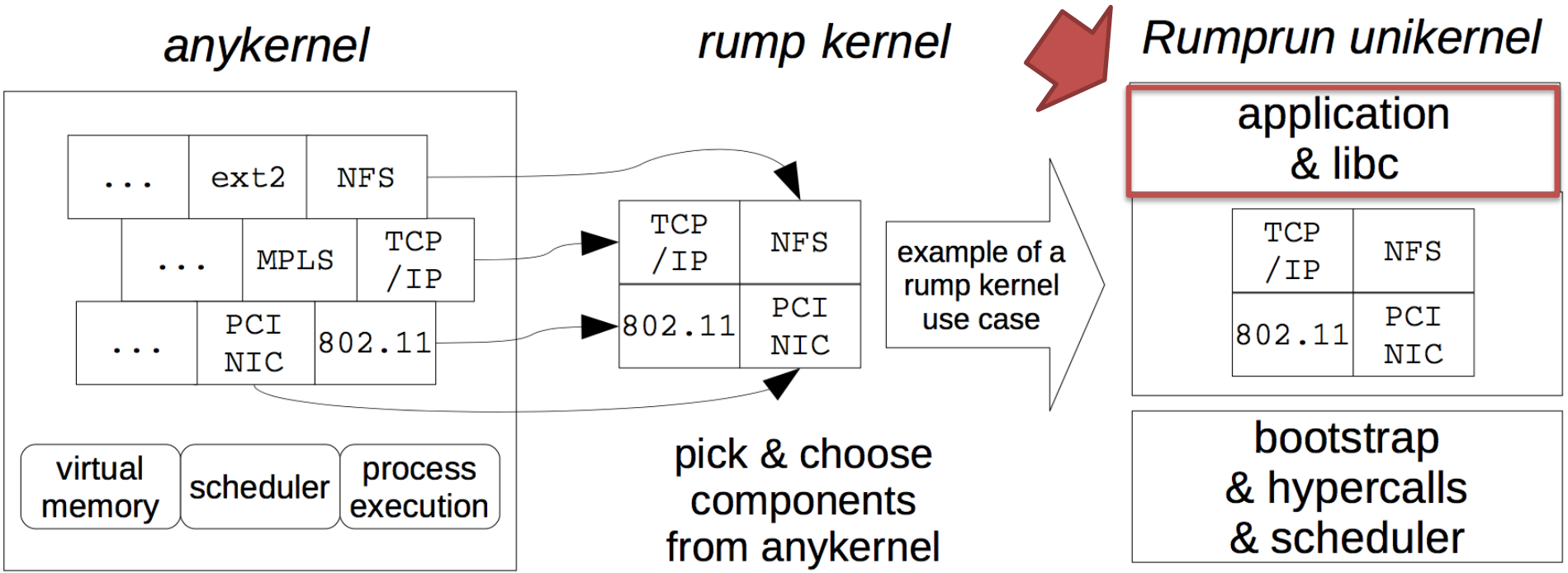
- Introduce
- Preliminary knowledge
  - rumpkernel/client, OpenFlow and switch(4)/switch(8)
- Why I took rump kernel to develop switch(4)
- How to implement rump kernel client
- Let's practice
- Conclusions

Rump kernel



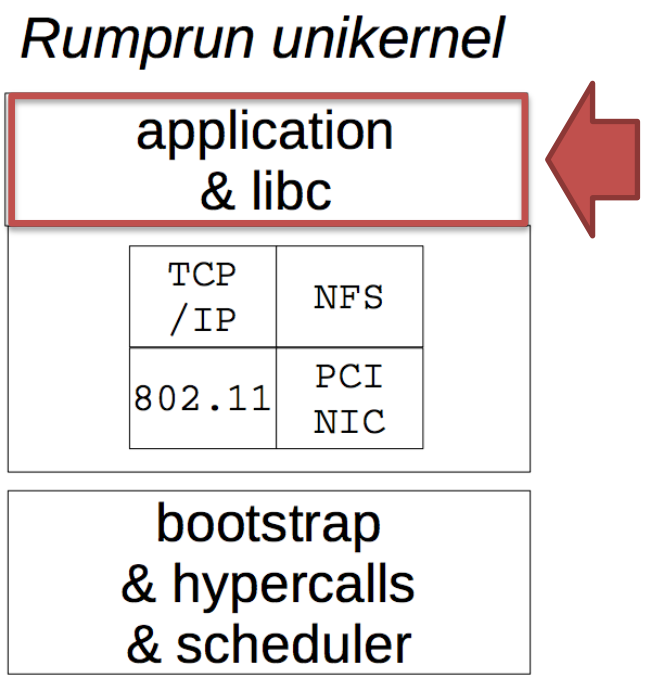
* Figure 2.4: Client types illustrated , THE DESIGN AND INMPLEMENTATION OF THE ANYKERNEL AND RUMP KERNELS

Rump kernel client



* Figure 2.4: Client types illustrated , THE DESIGN AND INMPLEMENTATION OF THE ANYKERNEL AND RUMP KERNELS

Rump Kernel Client

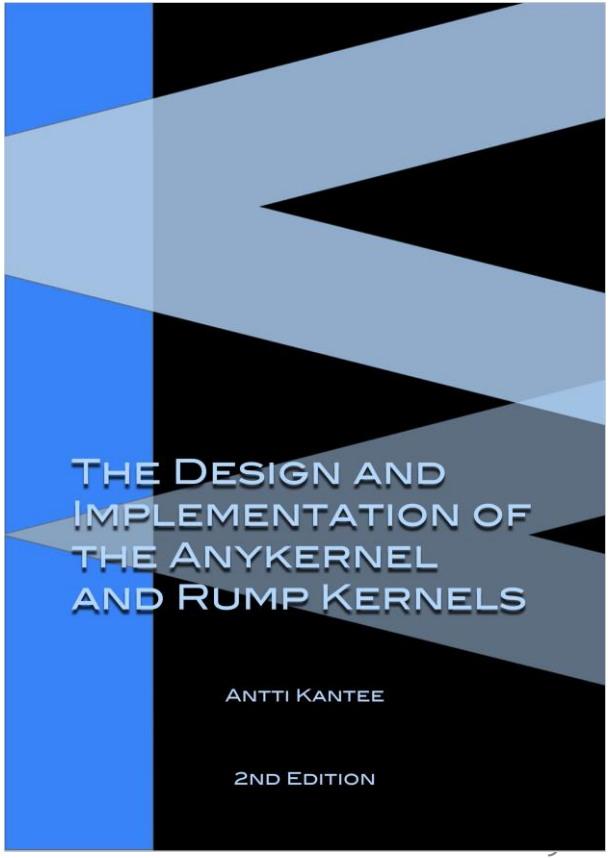


- Request something from a ump kernel
  - It's an example application that be using TCP/IP stack and NFS etc…
- 3 types of rump kernel client
  - local, remote, microkernel

You Want to get more detail

- You must read the book
  - There’re over 200 pages in the book

- THE DESIGN AND IMPLEMENTATION OF THE ANYKERNEL AND RUMP KERNELS
  - http://www.fixup.fi/misc/rumpkernel-book/



OpenFlow protocol

- OpenFlow is considered one of the first SDN standards

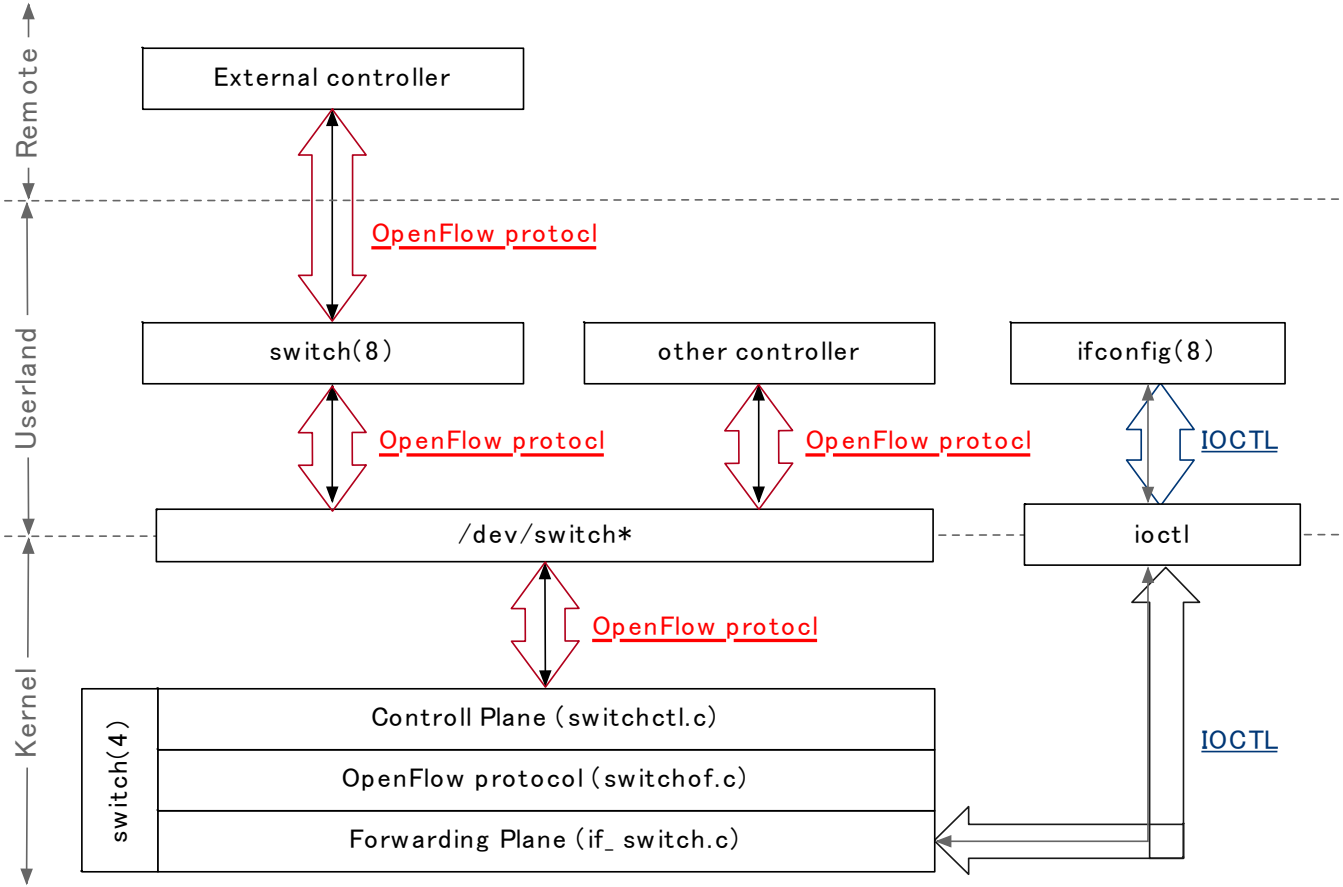
- Communication protocol that enable the SDN controller to directly interact with the forwarding plane

switch(4)/switch(8)

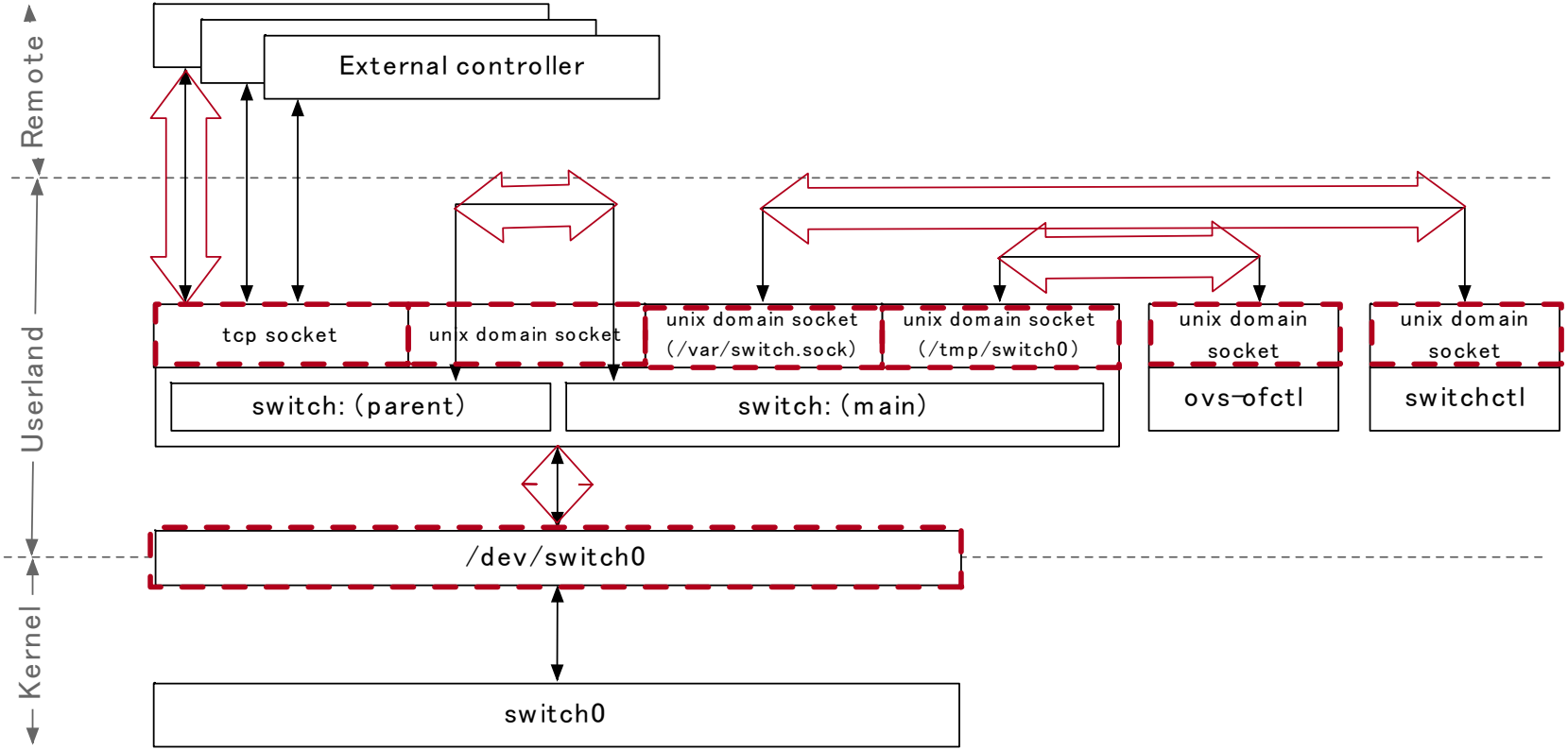
- switch(4) is much the same as bridge(4) except that has capability of OpenFlow switch
  - switch(4) can work OpenFlow switch itself

- switch(8) proxy OpenFlow channel between OpenFlow controller and switch(4)

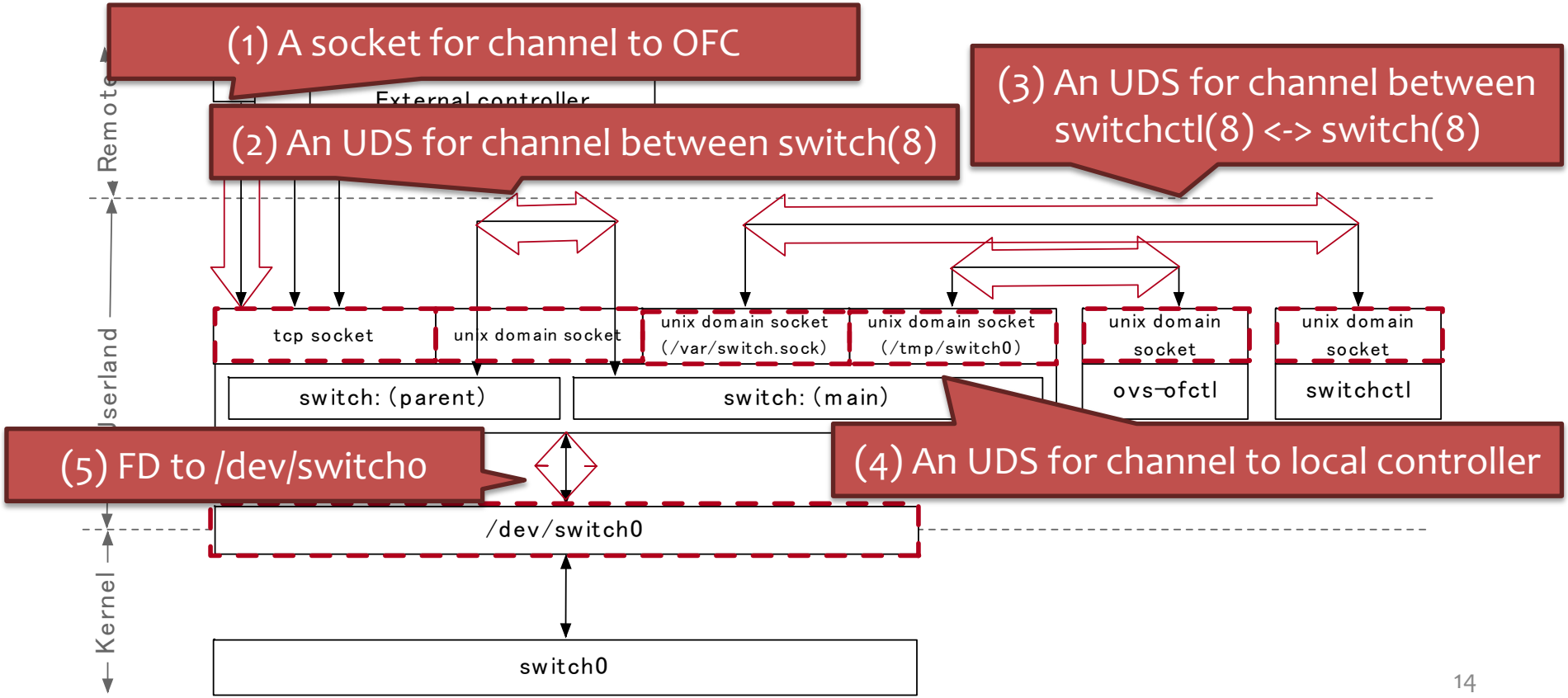
Architecture of switch(4)/switch(8)



I/O resources of switch(8)



I/O resources of switch(8)



Agenda

- Introduce
- Preliminary knowledge
  - rumpkernel/client, OpenFlow and switch(4)/switch(8)
- Why I took rump kernel to develop switch(4)
- How to implement rump kernel client
- Let's practice
- Conclusions

Why I took rump kernel to develop switch(4)

- If switch(4) runs on rump kernel

1. It can be easy to develop / debug to switch(4)
2. It can use some debug / profile tools such as Valgrind
3. It can be useful for development of switch(8)

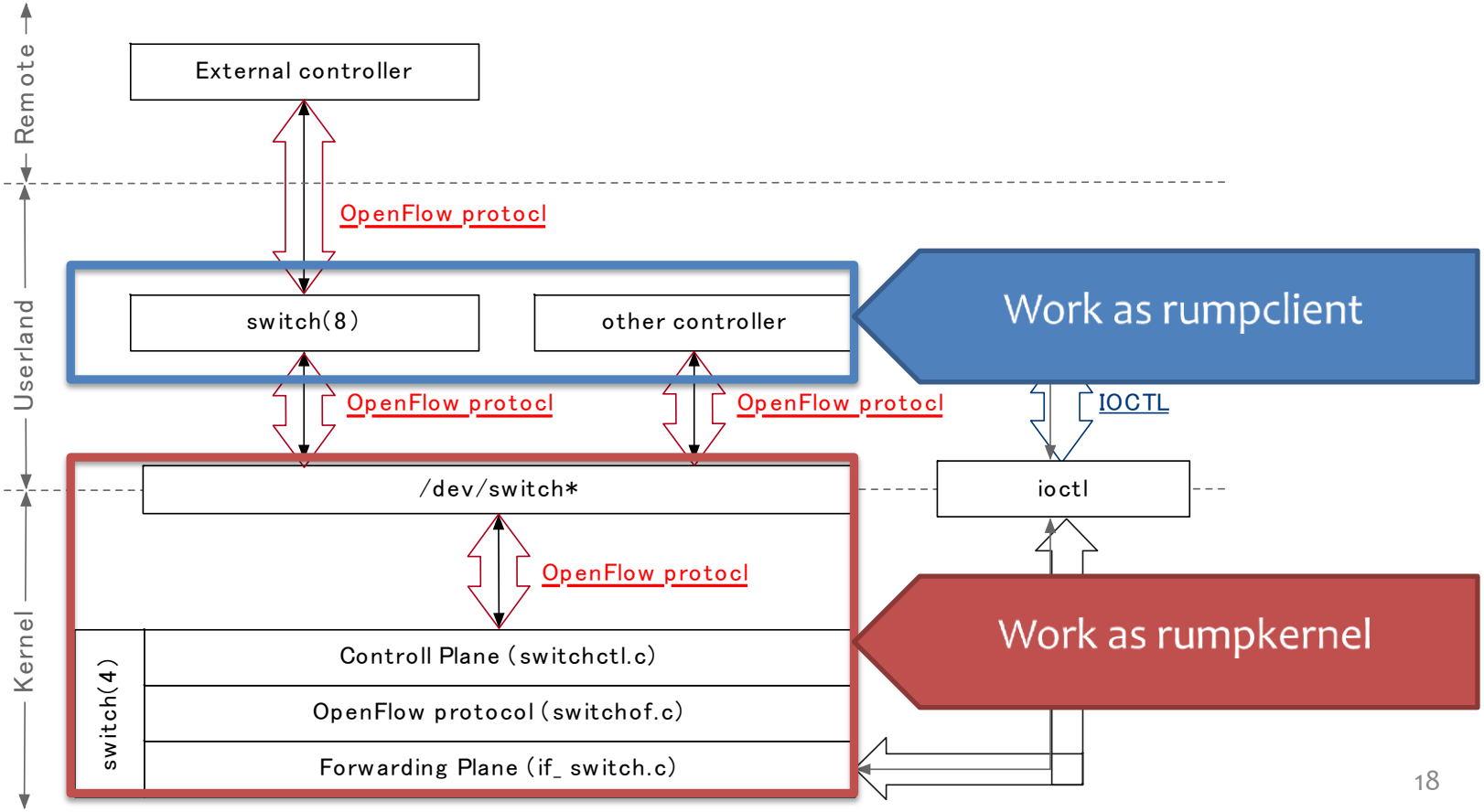
Goal

1. switch(4) can work on rump kernel

2. switch(8) can communicate switch (4) in rump kernel
   - Any OFCs can communicate switch(4) in rump kernel via switch(8)

3. Avoid modifying switch(8) as much as possible
   - It's decided by my own mind that how much it can modify

Architecture of switch(4)/switch(8)

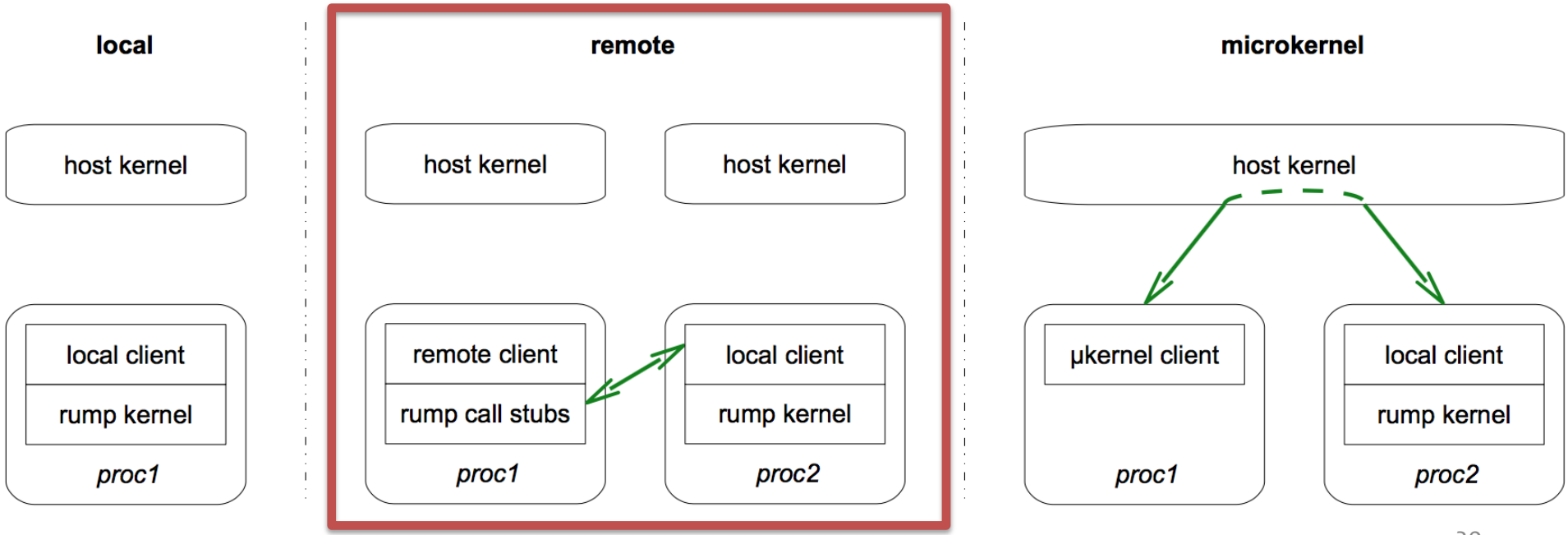


Agenda

- Introduce
- Preliminary knowledge
  - rumpkernel/client, OpenFlow and switch(4)/switch(8)
- Why I took rump kernel to develop switch(4)
- How to implement rump kernel client
- Let's practice
- Conclusions

3 Types of Rump Kernel Client

- I've selected remote



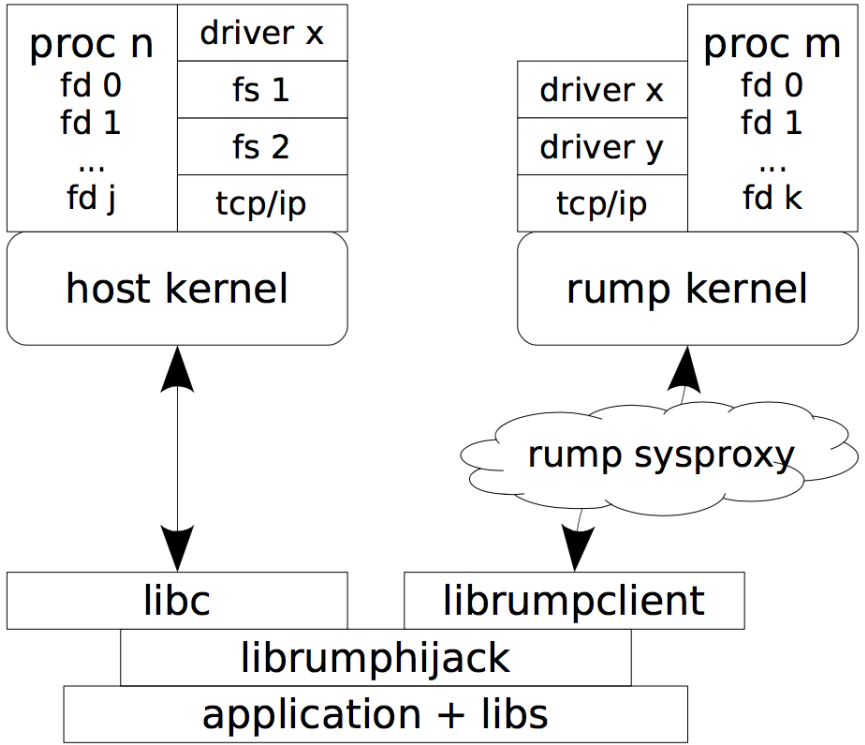
2 way implementation of remote client

- Use librumphijack
  - Hijacks host system call by LD_PRELOAD
  - Not need any modify for rump kernel client

- Modify application to work as rump kernel client

librumphijack

- librumphijack hijacks system call and proxy it to rump kernel



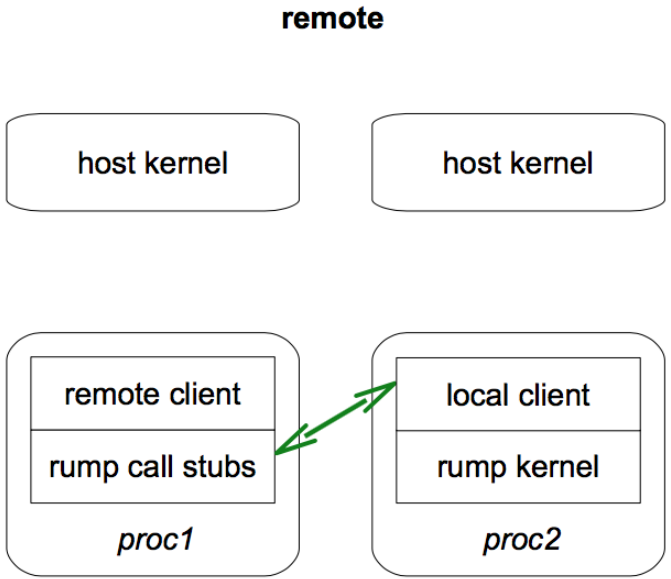
* Figure 3.27: System call hijacking Remote client architecture, THE DESIGN AND INMPLEMENTATION OF THE ANYKERNEL AND RUMP KERNELS

Modify application to work as rump kernel client

```
int
main(int argc, char *argv[])
{
        [....]
        /* bootstrap rump kernel */
        rump_init();

        /* open bpf device, fd is in implicit process */
        if ((fd = rump_sys_open(_PATH_BPF, O_RDWR, 0)) == -1)
                err(1, "bpf open");

        /* set bpf to use it */
        strlcpy(ifr.ifr_name, "virt0", sizeof(ifr.ifr_name));
        if (rump_sys_ioctl(fd, BIOCSETIF, &ifr) == -1)
                err(1, "set if");
         [....]
}
```

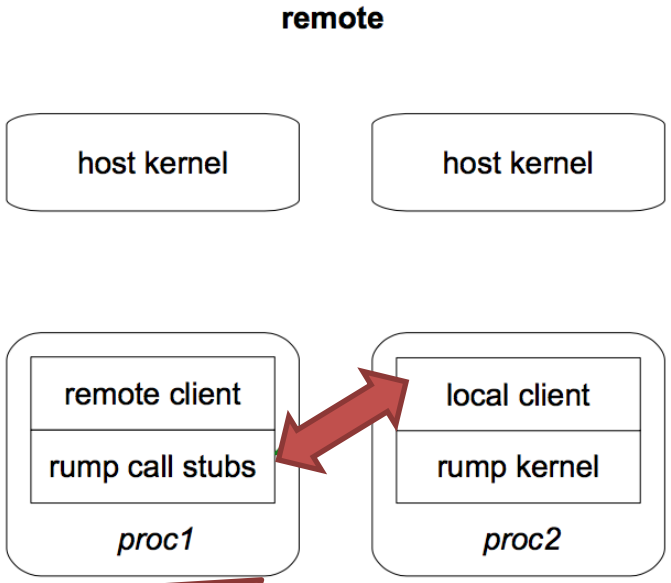


Modify application to work as rump kernel client

```
int
main(int argc, char *argv[])
{
        [....]
        /* bootstrap rump kernel */
        rump_init();

        /* open bpf device, fd is in implicit process */
        if ((fd = rump_sys_open(_PATH_BPF, O_RDWR, 0)) == -1)
                err(1, "bpf open");

        /* set bpf to use it */
        strlcpy(ifr.ifr_name, "virt0", sizeof(ifr.ifr_name));
        if (rump_sys_ioctl(fd, BIOCSETIF, &ifr) == -1)
                err(1, "set if");
         [....]
}
```



Rump call stubs (librumpclient) proxys system call

Agenda

- Introduce
- Preliminary knowledge
  - rumpkernel/client, OpenFlow and switch(4)/switch(8)
- Why I took rump kernel to develop switch(4)
- How to implement rump kernel client
- Let's practice
- Conclusions

Attempt 3 ways to work switch(8) as rump client

1. Apply librumphijack

2. Modify switch(8) for rump kenel client

3. Put I/O Proxy daemon between switch(4) and switch(8)

Attempt 3 ways to work switch(8) as rump client

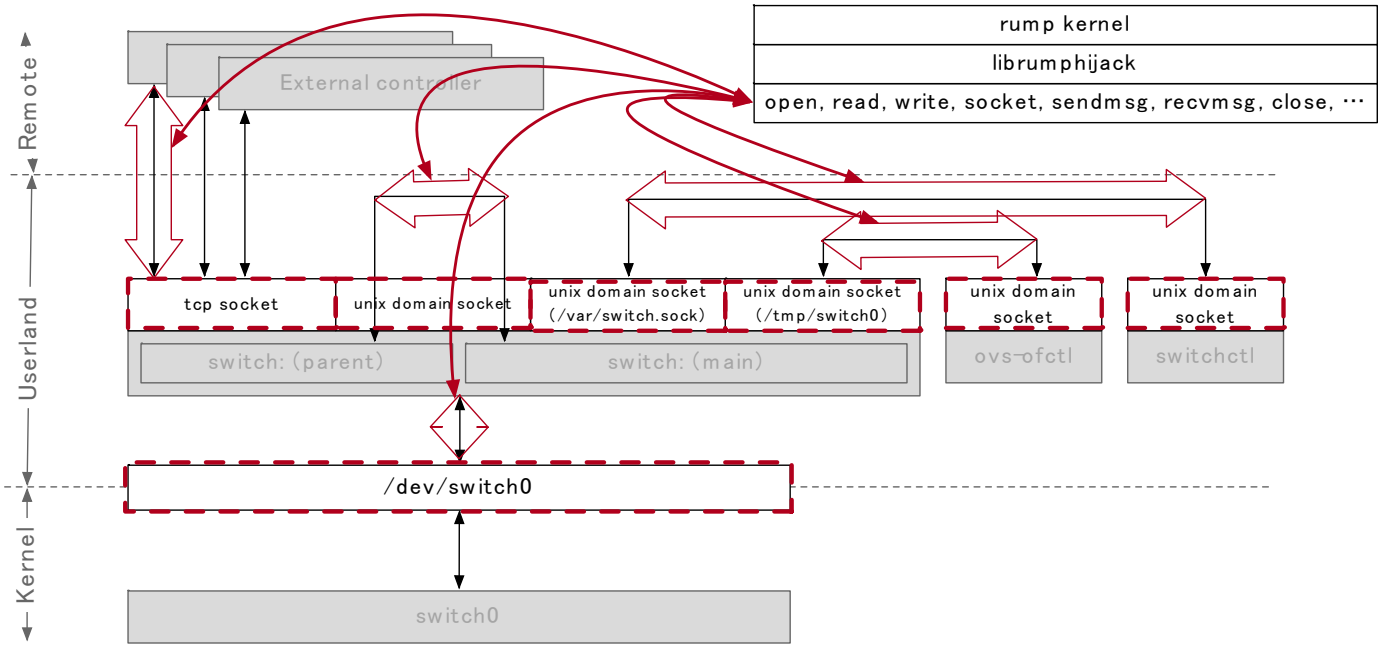
1. Apply librumphijack

2. Modify switch(8) for rump kenel client

3. Put I/O Proxy daemon between switch(4) and switch(8)

Apply librumphijack

- Fortunately, switch(8) only calls rumpkernel-supporting system calls



Couldn’t get good result…

- rumphijack doesn’t support kqueue/kevent
  - *) rump kernel supports kevent/kqeueu

- A Commnet at kevent() in librumhijack

```
/*
 * Check that we don't attempt to kevent rump kernel fd's.
 * That needs similar treatment to select/poll, but is slightly
 * trickier since we need to manage to different kq descriptors.
 * (TODO, in case you're wondering).
 */
```

Review

- That's how it goes
- The comment offers me to implement kqueue/kevent to rumphijack
  - But I guess it's a lot difficult so I didn't it at that moment
- I considered the other way

Attempt 3 ways to work switch(8) as rump client

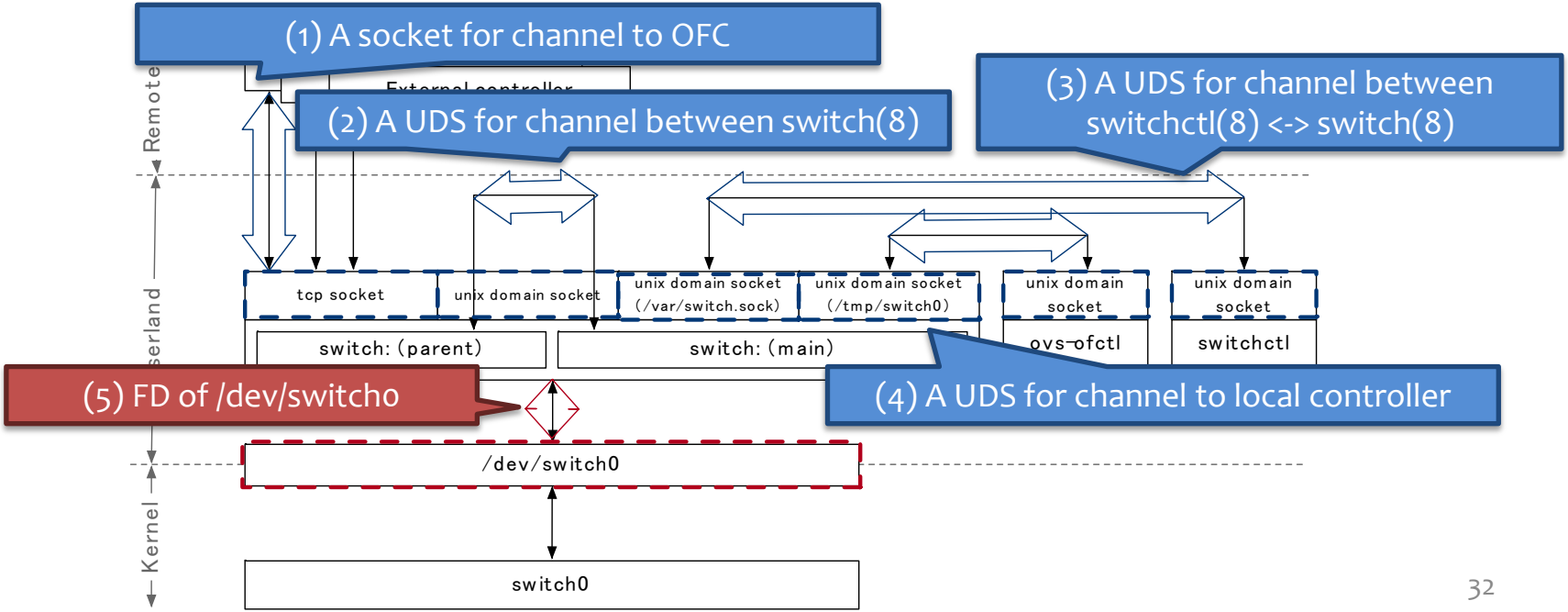
1. Apply librumphijack
   - Failed because it doesn't support kqueu/kevent

2. Modify switch(8) for rump kenel client

3. Put I/O Proxy daemon between switch(4) and switch(8)

Only calls rump kernel's system calls when I/O resources are rump kernel's

- Fortunately, only /dev/switch0 communicates with rump kernel



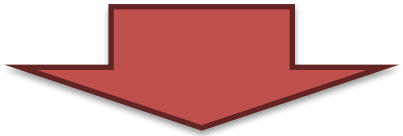
Couldn’t get any good results

- Not enough to consider using difference kernels

- It’s difficult to achieve I/O multiplexer for difference kernels because it’s necessary to work tricky

What's difficult to handle I/O multiplexer?

- switch(8) has multiple I/O such as for /dev/switch0, OpenFlow Controller(OFC), etc…
  - switch(8) uses kqeueu/kevent to I/O multiplexer
- The FD of channel between OFC is held by host kernel
- The FD of channel between switch(4) held by rump kernel



- It's impossible to handle I/O resources in difference kernels by the one kernel

Review

- An approach that handles different kernel's I/O seems not so good, especially I/O multiplexing

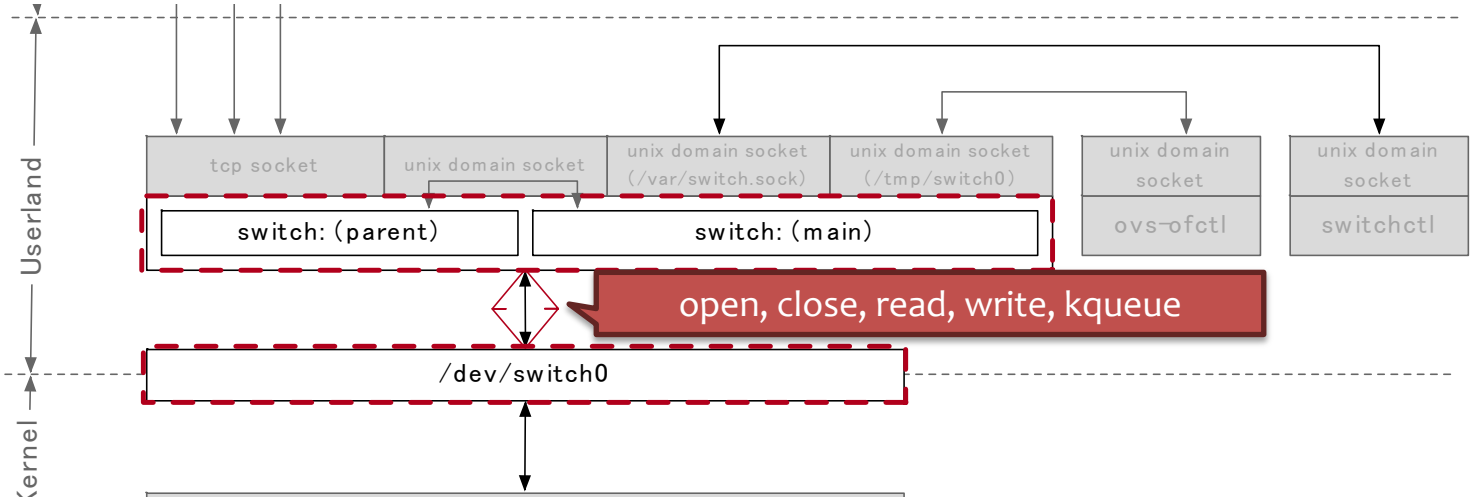
- switch(8) should only handle either I/O resources of rump's or host's
  - switch(8) linked a few external libraries such as libevent, so it have to replace every system call within external libraries
  - I never want to do it!!

Attempt 3 ways to work switch(8) as rump client

1. Apply librumphijack
   - Failed because it doesn't support kqueu/kevent
2. Modify switch(8) for rump kenel client
   - Failed because it's too difficult to work I/O multiplexing
3. Put an I/O proxy daemon between switch(4) and switch(8)

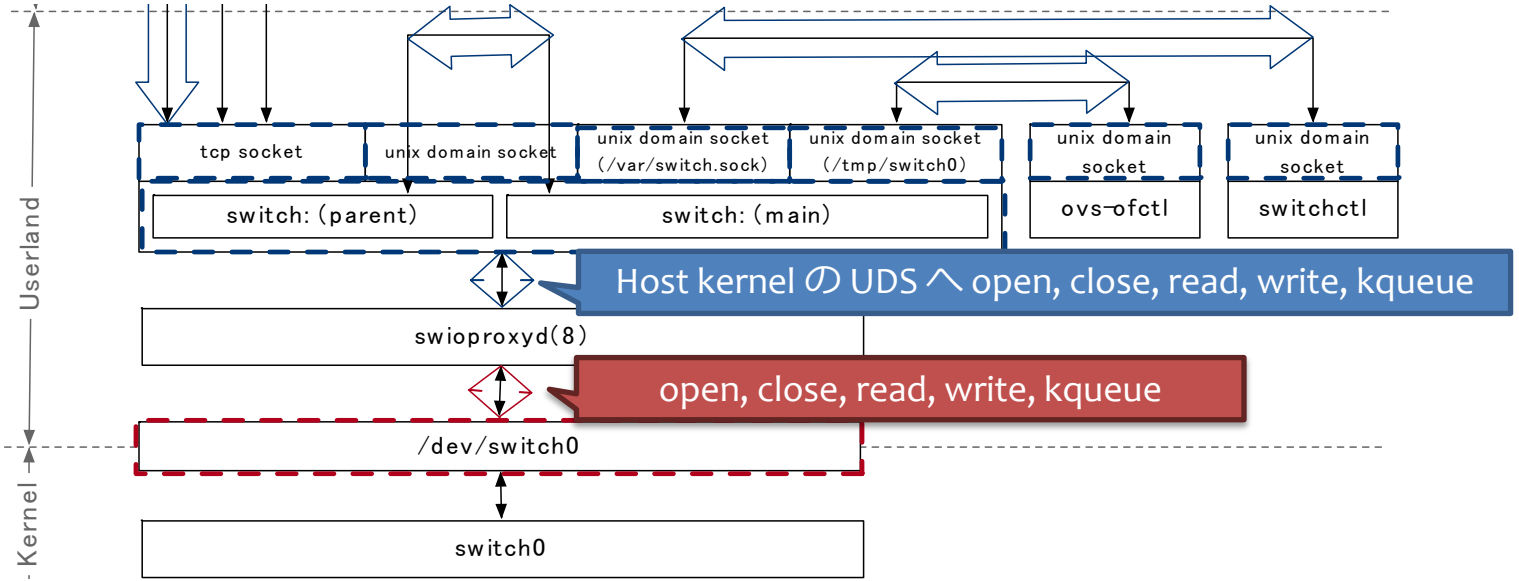
An I/O Proxy Daemon between switch(4) and switch(8)

- Fortunately, called system calls for /dev/switch0 are open, close, read, write, kqueue and kevent
- It can replace easily to Unix Domain Socket



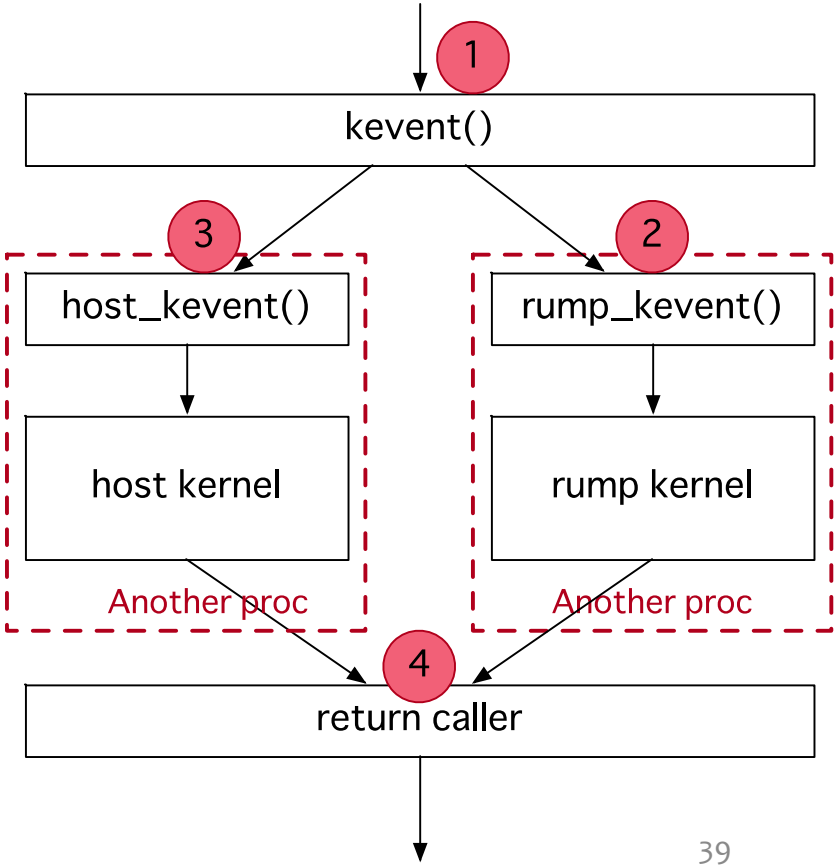
swioproxyd(8)

- Communicate between switch(8) and swioproxyd(8) via Unix Domain Socket
- swioproxyd(8) proxys between switch(8) and rumpkernel



swioproxyd(8)

1. Distinguish between rump's FDs and host's FDs
2. Produces a new thread and calls rump_kevent()
3. Produces a new thread and calls host's kevent()
4. Wait for ready any FDs



Review

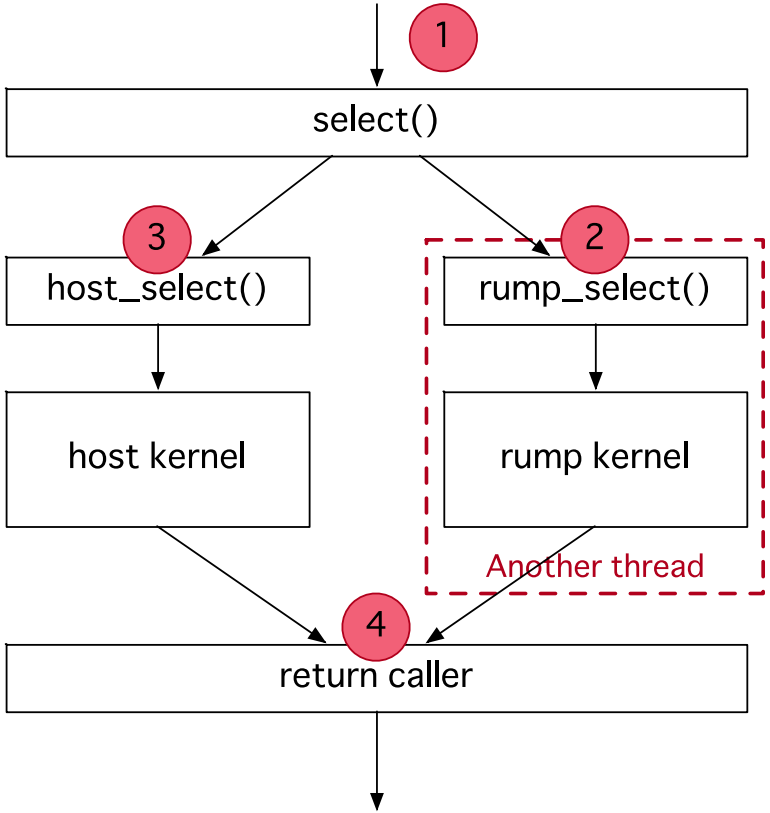
- ✓ switch(4) can work on rump kernel
- ✓ switch(8) can communicate switch (4) in rump kernel
  - Any OFCs can communicate switch(4) in rump kernel via switch(8)
- ✓ Avoid modifying switch(8) as much as possible
  - It's decided by my own mind that how much it can modify

Conclusion

- At any time, using rumphijack is 1st choice
  - But it doesn't work at a few cases such as switch(8)
- It doesn't produce good result to handle both rump and kernel I/O resource in the one program
- It's effective for linked some external libraries program to put on proxy

appendix

How to work select/poll in librumphijack



1. Distinguish between rump's FDs and host's FDs by checking FD_SET
2. Produces a new thread and calls rump_select()
3. Calls select() on Host kernel too
4. Wait for ready any FDs