FOSDEM'21

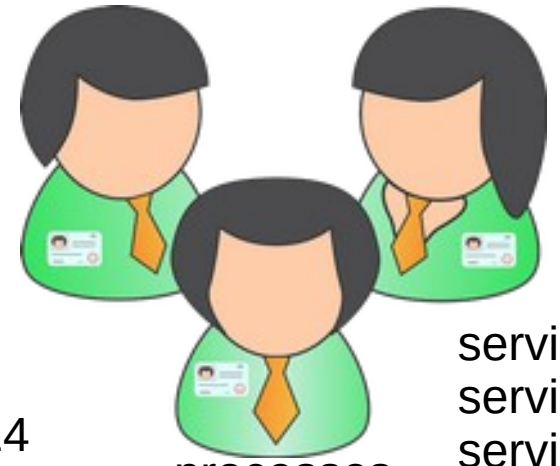
libioth

The definitive API for the Internet of Threads

Renzo Davoli – Michael Goldweber
University of Bologna (Italy) – Xavier University (Cincinnati, USA)
VirtualSquare

Microkernel DevRoom

Internet of Threads (IoTh)



processes

host.company.com → 11.12.13.14

service1.company.com → 2001:1:2::1
service2.company.com → 2001:1:2::2
service3.company.com → 2001:1:2::3

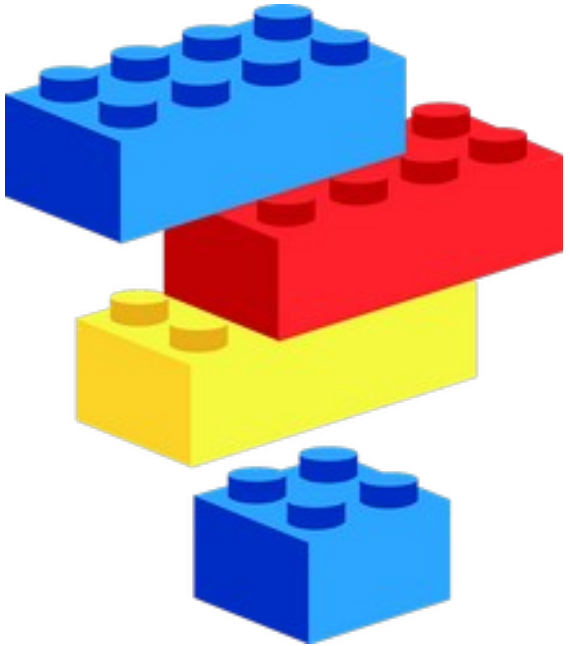
IoTh vs Iold

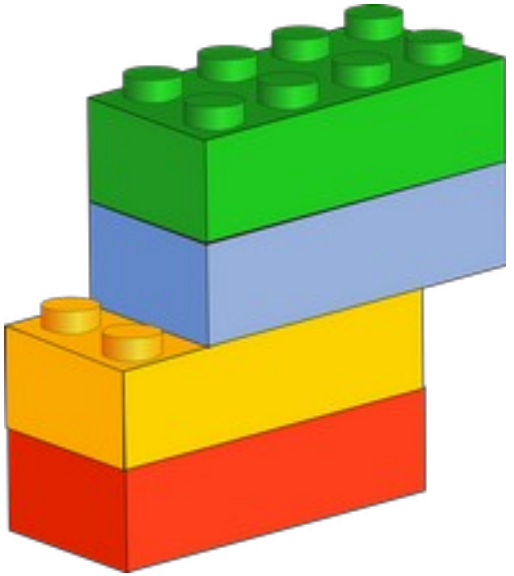
What is an end-node of the Internet?

It depends on what is identified by an IP address.

- Internet of Threads – IoTh – Processes/threads are autonomous *nodes* of the Internet
- Internet of Legacy Devices (in brief I-old in this presentation)
  - Internet of Hosts – Internet of Network Controllers.
  - Internet of Namespaces

PVM – IoTh - µkernel



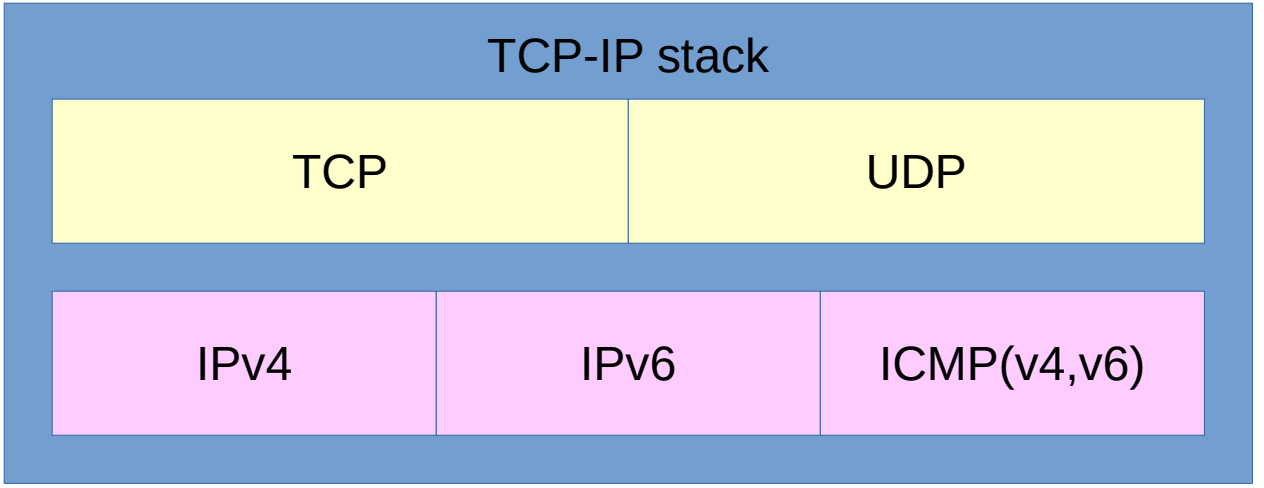


PVM – IoTh - µkernel

- Microkernel, Internet of Threads, Partial Virtual Machines have the common goal to create independent code units implementing services
- They are all against monolithic implementations
- The challenge of this talk is to create contacts, exploit parallelisms, that allow to share results, API, code.

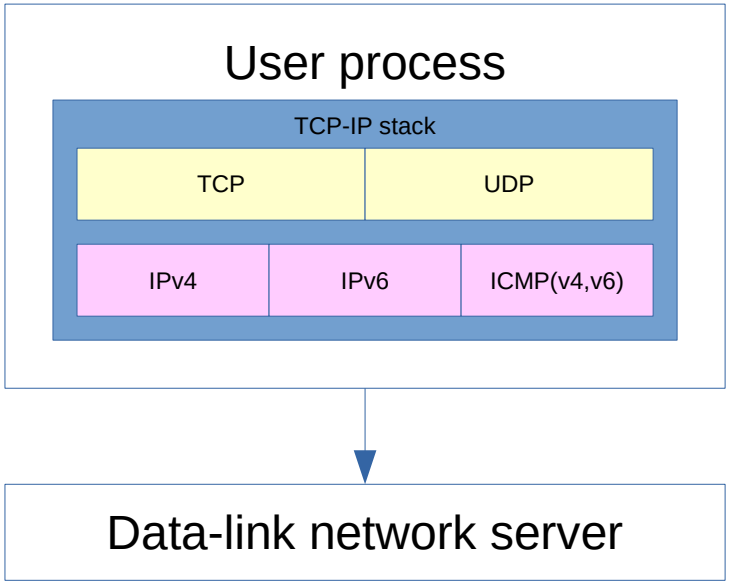
Network Stack

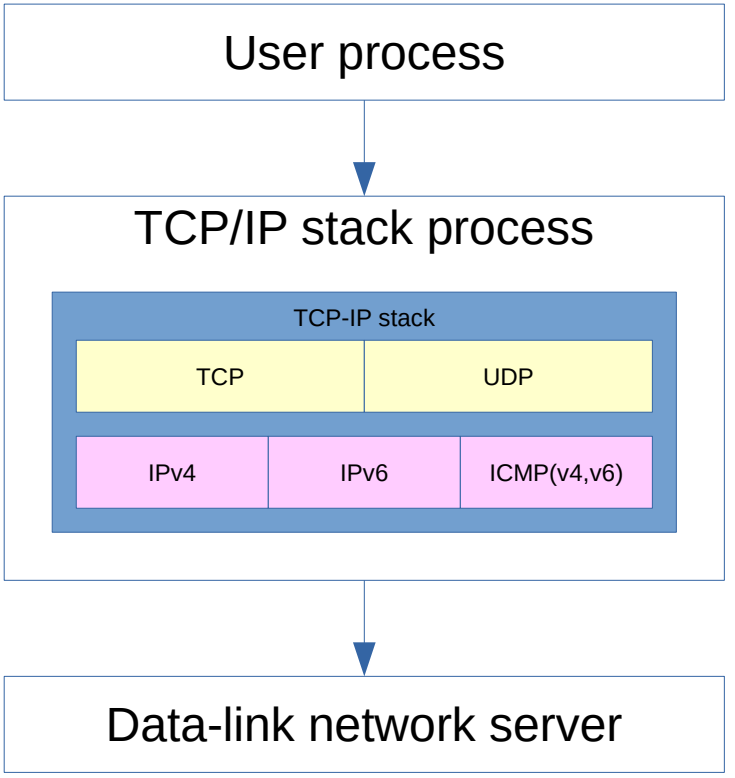
- API to application layer



- API (NPI) to data-link – (e.g. libioth uses VDE: libvdeplug)

IoTh & µkernel





API to the App Layer

- Complete
  - All ops currently available must be supported
- Usable
  - Syntax must be consistent with the major standards
- Minimal/Clean
  - Avoid useless or duplicated ops

API requirements

- Configuration calls
  - Create – Delete a stack instance
  - Configure parameters (as IP address-es, routing etc)
- Communication calls
  - open/close a communication endpoint
  - send/receive/set-get options

API design choices: 1- Stack creation/deletion

- A stack is identified by a descriptor of type:
  - `struct ioth *`
- Stack Creation:

```
struct ioth *ioth_newstack(const char *stack, const char *vnl);
struct ioth *ioth_newstackl(const char *stack, const char *vnl, ... /* (char  *) NULL */);
struct ioth *ioth_newstackv(const char *stack, const char *vnlv[]);
```

  - `ioth_newstack` for one interface (or none if `vnl==NULL`), the others are for more interfaces.
  - The string `stack` selects the stack implementation, loaded as a plugin.
  - `vnl` stands for “Virtual Network Locator”, it selects the VDE network to connect the virtual interface(s).
- Stack deletion:

```
int ioth_delstack(struct ioth *iothstack);
```

API design choices: 2- Communication

- Creation of a communication endpoint:

```
int ioth_msocket(struct ioth *stack, int domain, int type, int protocol);
```

  - It extends `socket(2)`, it allows the choice of the stack
  - It returns a *real* file descriptor that can be used in `poll(2)`, `select(2)` or similar

- for everything else... Berkeley Sockets

```
ioth_close, ioth_bind, ioth_connect, ioth_listen, ioth_accept, ioth_getsockname, ioth_getpeername,
ioth_setsockopt, ioth_getsockopt, ioth_shutdown, ioth_ioctl, ioth_fcntl, ioth_read, ioth_readv,
ioth_recv, ioth_recvfrom, ioth_recvmsg, ioth_write, ioth_writev, ioth_send ioth_sendto and
ioth_sendmsg
```

  - have the same signature and functionalities of their counterpart without the `ioth_` prefix

API design choices: 3- Configuration

- No more calls needed!
- Net configuration via AF_NETLINK sockets
- As defined in:
  - RFC 3549 - Linux Netlink as an IP Services Protocol
- *Helper libs are provided as services using the API (e.g. nlinline+ or iothconf).*

Example: send “ciao\n”

```c
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char *argv[]) {
  struct sockaddr_in dst = {
    .sin_family = AF_INET,
    .sin_port = htons(5000),
    .sin_addr.s_addr = inet_addr("10.0.0.2")
  };

  int fd = socket(AF_INET, SOCK_DGRAM, 0);
  sendto(fd, "ciao\n", 5, 0, (void *) &dst, sizeof(dst));
  close(fd);
}
```

Example: ioth send “ciao\n”

```c
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <ioth.h>

int main(int argc, char *argv[]) {
  struct ioth *stack = ioth_newstack("vdestack", "vde:///tmp/hub");
  struct in_addr myaddr = {.s_addr = inet_addr("10.0.0.1")};
  int ifindex = ioth_if_nametoindex(stack, "vde0");
  ioth_ipaddr_add(stack, AF_INET, &myaddr, 24, ifindex);
  ioth_linksetupdown(stack, ifindex, 1);

  struct sockaddr_in dst = {
    .sin_family = AF_INET,
    .sin_port = htons(5000),
    .sin_addr.s_addr = inet_addr("10.0.0.2")
  };
  int fd = ioth_msocket(stack, AF_INET, SOCK_DGRAM, 0);
  ioth_sendto(fd, "ciao\n", 5, 0, (void *) &dst, sizeof(dst));
  ioth_close(fd);
}
```

The way to libioth...

- ... has been long
- Libioth uses a number of concept and software tools/libraries developed at VirtualSquare.
- UNIX philosophy based:
  - Each tool/library or concept does one thing and *tries to* do it well
  - Tool/libraries or concepts have been designed to interoperate.

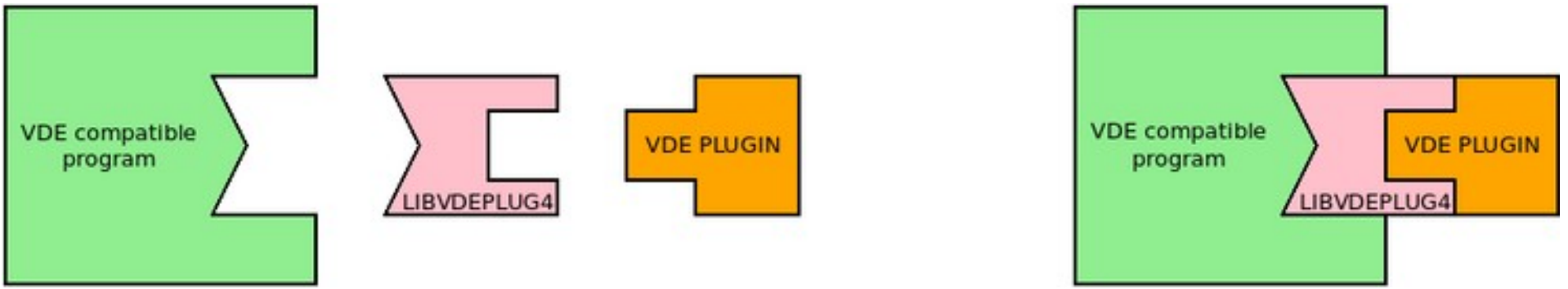
The way to libioth...

- Virtual Distributed Ethernet (VDE)
- Fduserdata
- Libvpoll and vpoll device
- Nlinline
- Libnlq
- Vuos: vunetioth
- Vdestack
- Picoxnet
- ...

... has further destinations...

- Iothconf
- Iothdns
- ...

VDE: virtual distributed ethernet



- libvdeplug supported by
  - VM: qemu/kvm, virtualbox, user-mode linux
  - Namespaces: vdens
  - network stack libs: vdestack, picoxnet, lwipv6

VDEplug API (NPI):

- A VDE connection endpoint is identified by a descriptor of type:
  - `VDECONN *`
- A VDE network is identified by a string: the Virtual Network Locator. e.g.:

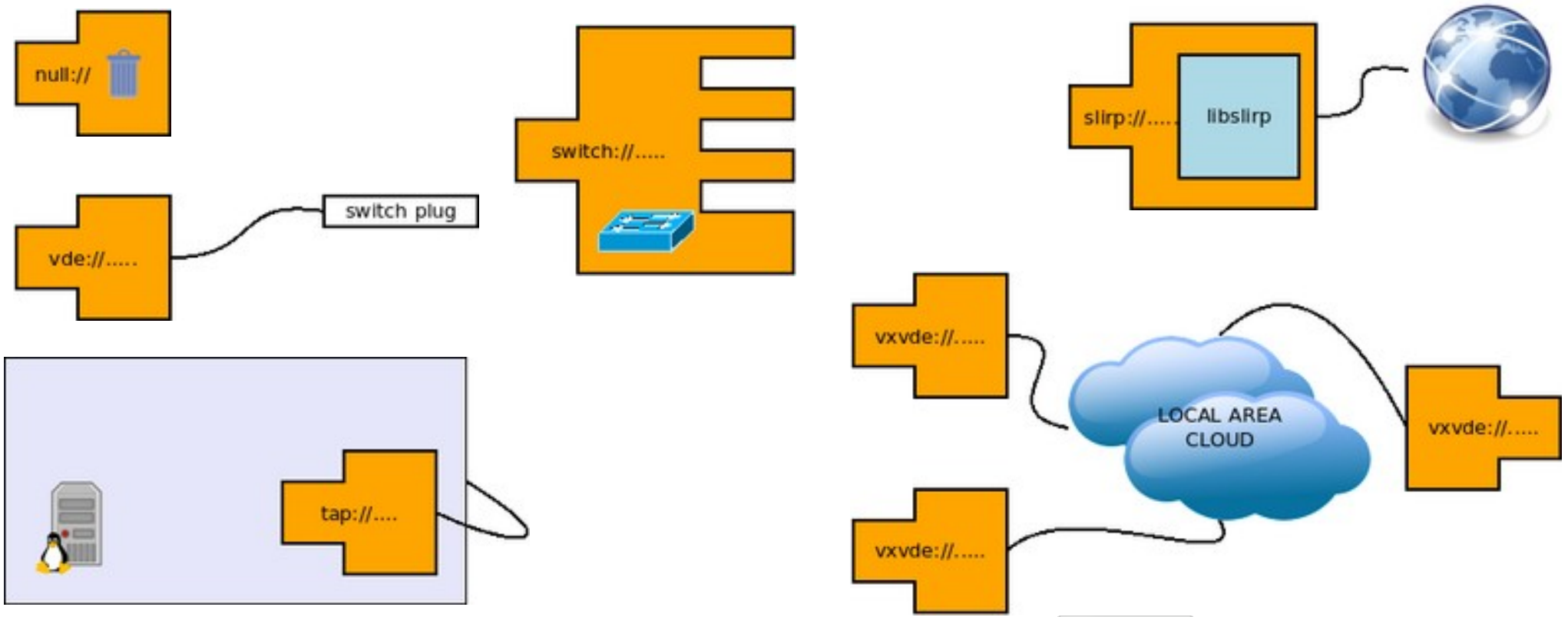
```
null:// switch:///tmp/myswitch vxvde://234.0.0.1 vde:///var/run/sw1 slirp:// tap://tap4
```

- The API is:

```
VDECONN *vde_open(char *vnl, char *description, NULL);
ssize_t vde_send(VDECONN *conn, const void *buf, size_t len, int flags);
ssize_t vde_recv(VDECONN *conn, void *buf, size_t len, int flags);
int vde_datafd(VDECONN *conn); // for poll/select, test for new packet availability
int vde_ctlfd(VDECONN *conn);  // for poll/select, test for conn shutdown
int vde_close(VDECONN *conn);
```

  - `vde_open`: description is for debugging, can be omitted, the third arg is for backwards compatibility.
  - `vde_send, vde_recv`: the last arg `flags` is reserved for future developments. Set to 0.

Some VDE plugins

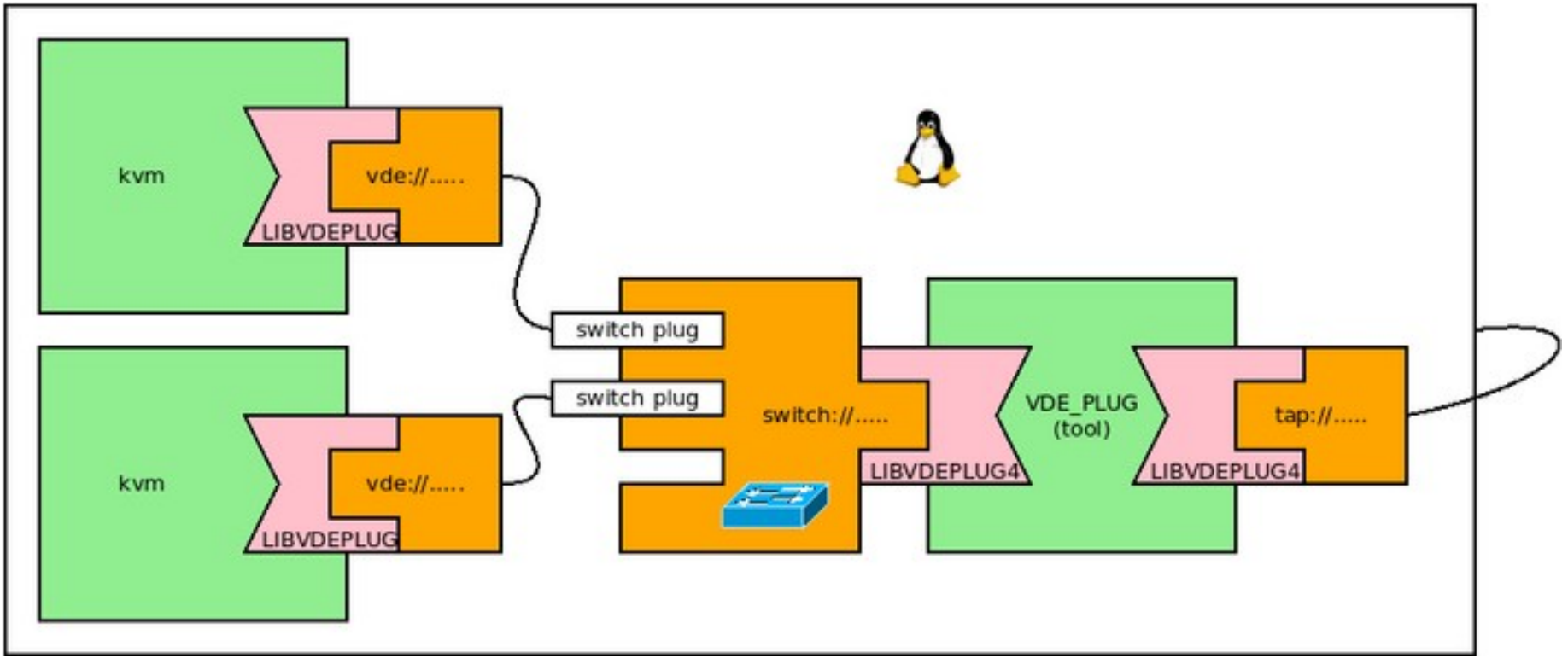


A note about the examples...

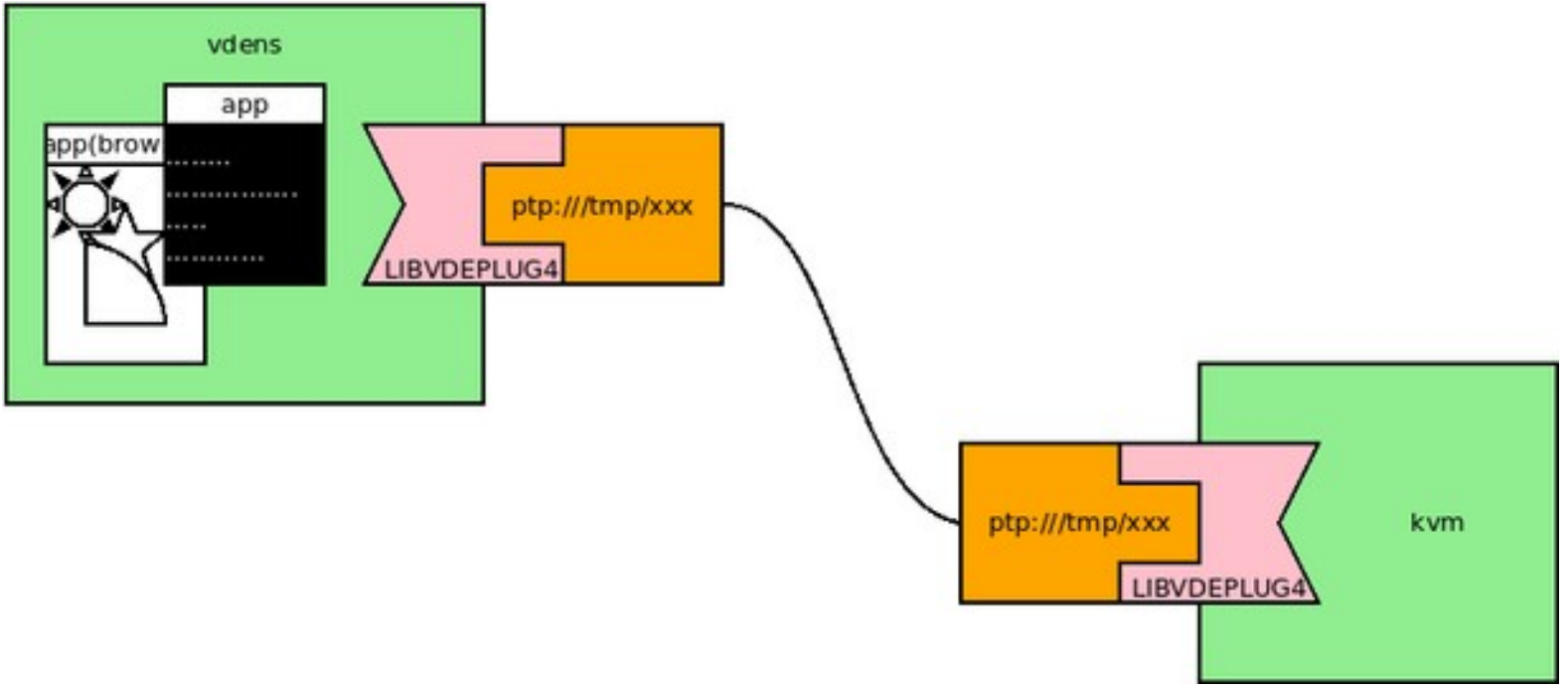
Interested readers can find the complete seqence of commands to set up the scenarios of the examples of the following slides in the tutorial section of Virtualsquare's wiki:

http://wiki.virtualsquare.org

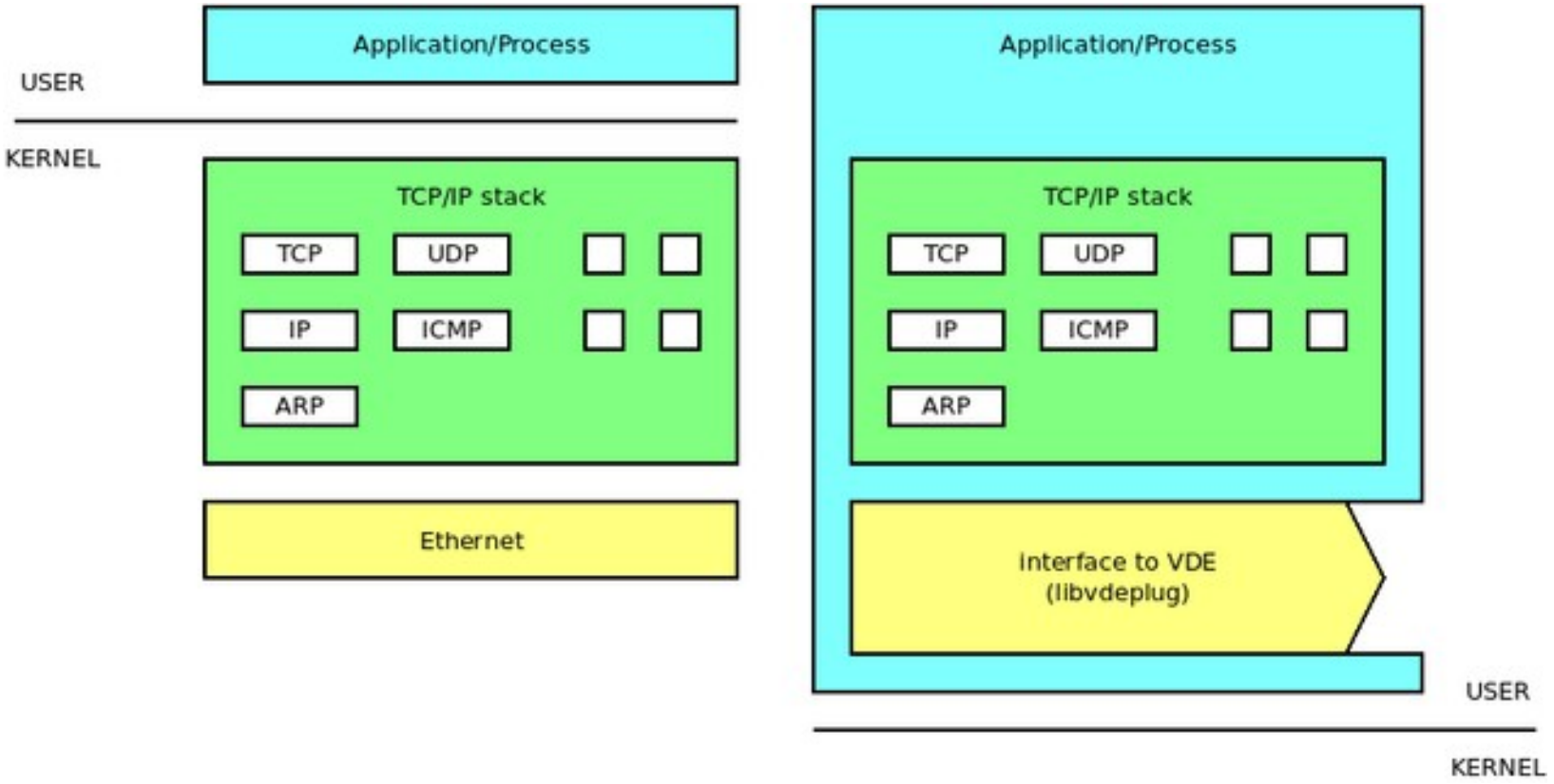
VDE example



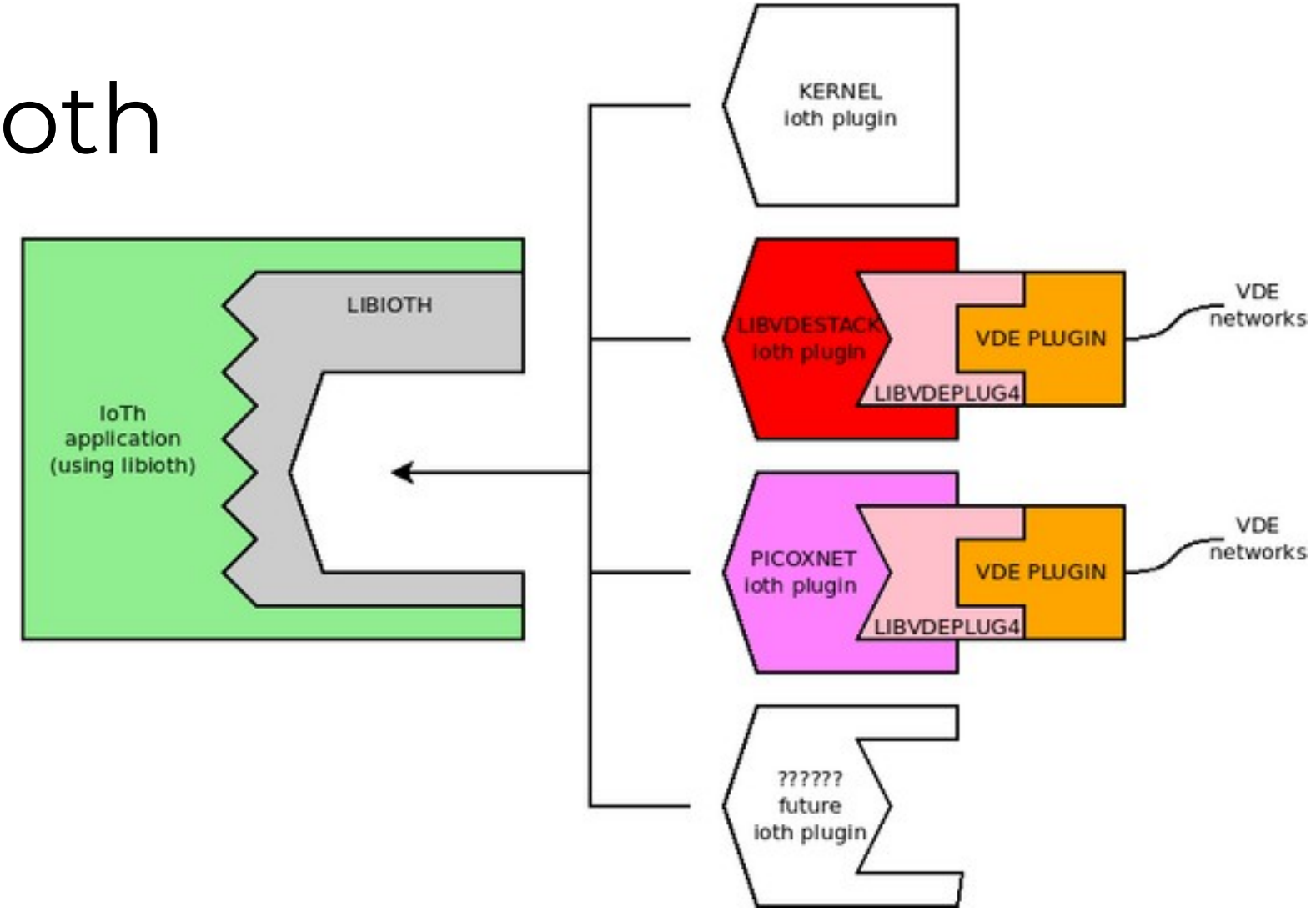
VDE namespaces

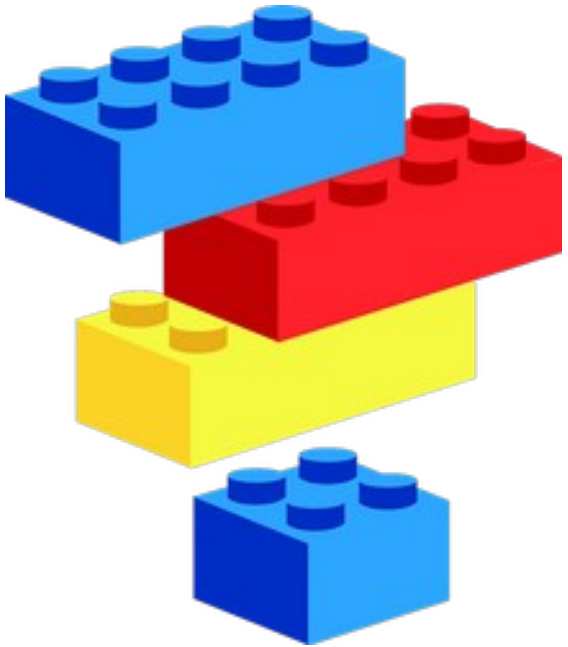


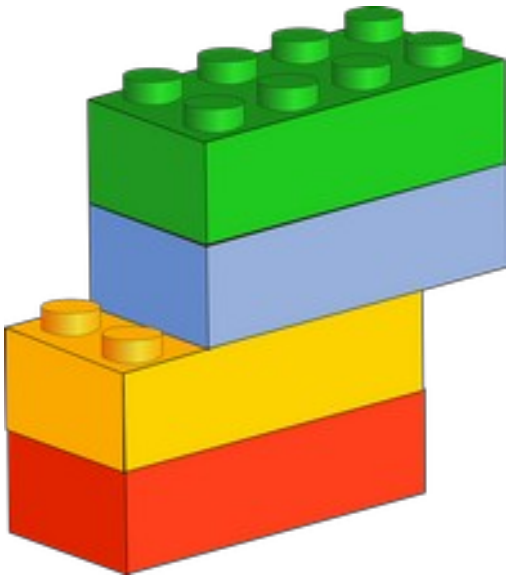
Internet of Threads



libioth







fduserdata

- Associate user data to file descriptors
- Libioth can select the stack instance from the file descriptor
- The signature of all the calls of the Berkeley socket API is preserved.
- Only msocket has one extra argument.

vpoll

- libioth's file descriptors can be used in poll, select, ppoll, pselect, epoll...
- A way to generate arbitrary poll events was missing
- libvpoll uses a kernel module to provide a complete support (or implements an emulation able to manage POLLIN, POLLOUT and partially POLLHUP)

nlinline

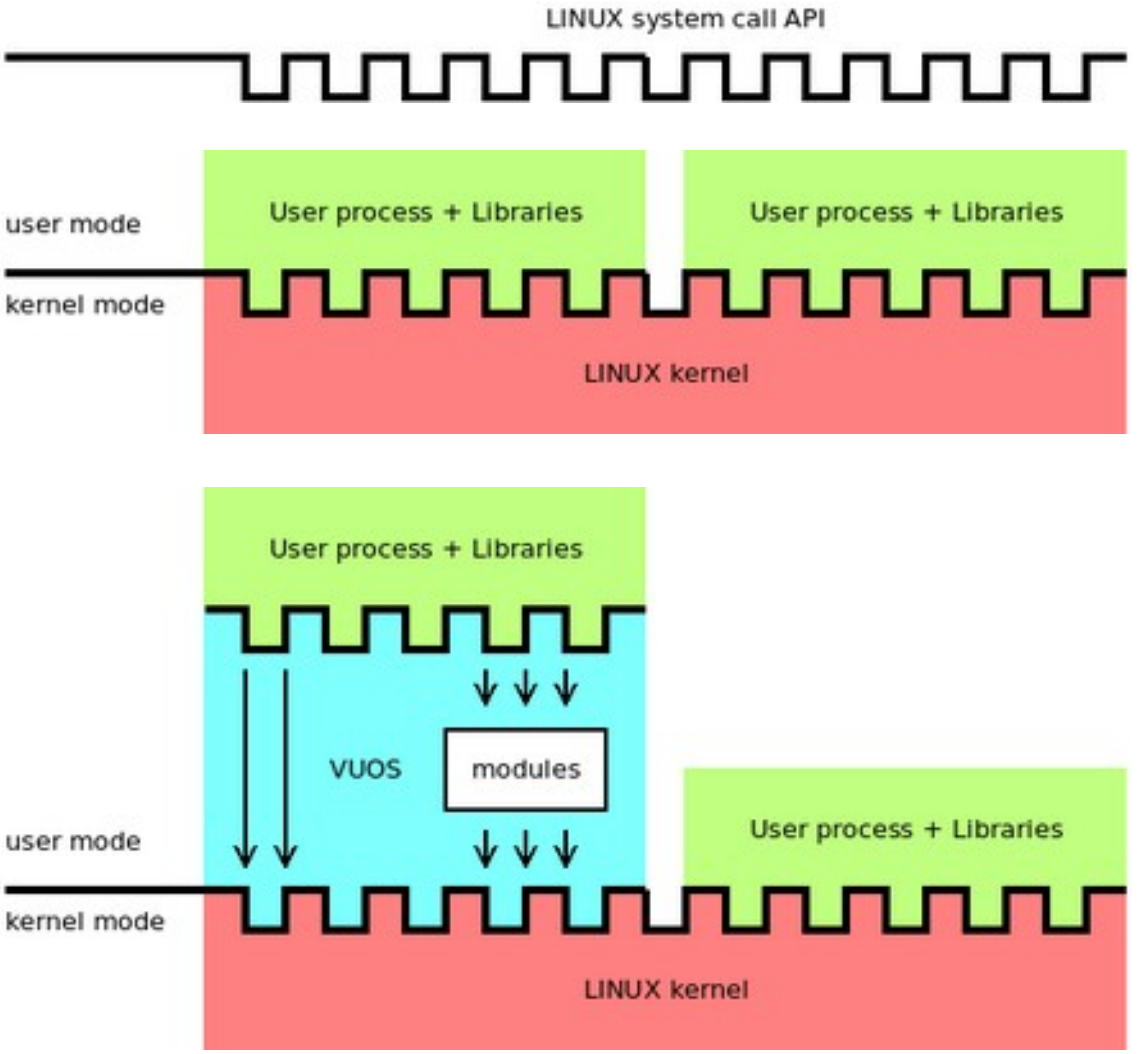
- Network stacks are generally considered as services provided by the kernel. So there are commands to configure the stacks like iproute or the old ifconfig.
- A library providing functions to configure a stack was missing
- nlinline is a light library on inline functions providing access to all the basic configuration ops (add/del an interface, set an interface up/down, add/del IP addresses, add/del routes). It uses netlink as described in RFC3549.

libnlq

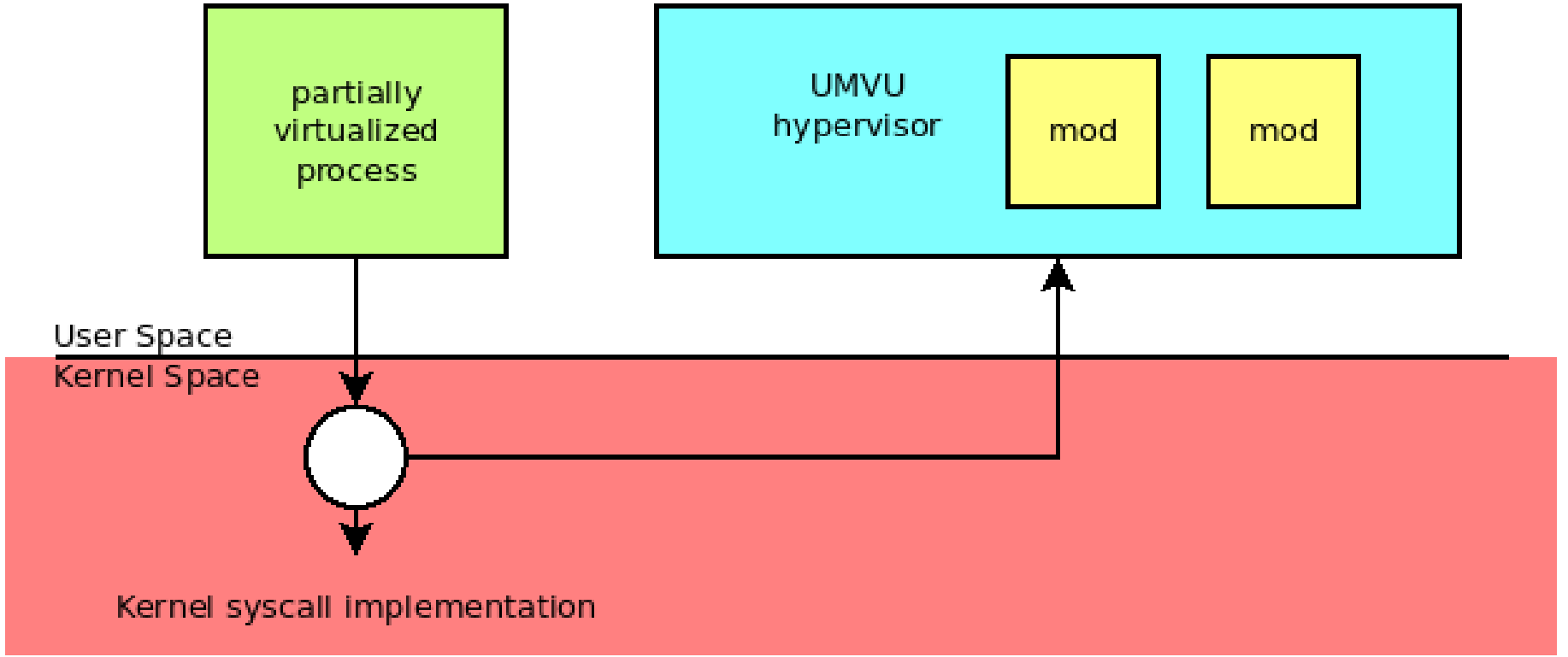
- Network stacks are generally considered as services provided by the kernel. A library able to decode configuration requests via netlink was missing.
- libnlq forges and decodes rt-netlink messages both at client and at server side.
- It also provides an emulation layer to support deprecated (though still used by glibc) netdevice ioctl.

VUOS

- VUOS implements a Partial Virtual Machine
- It gives the abstraction of namespaces entirely implemented in user-space



umvu: user-mode vuos

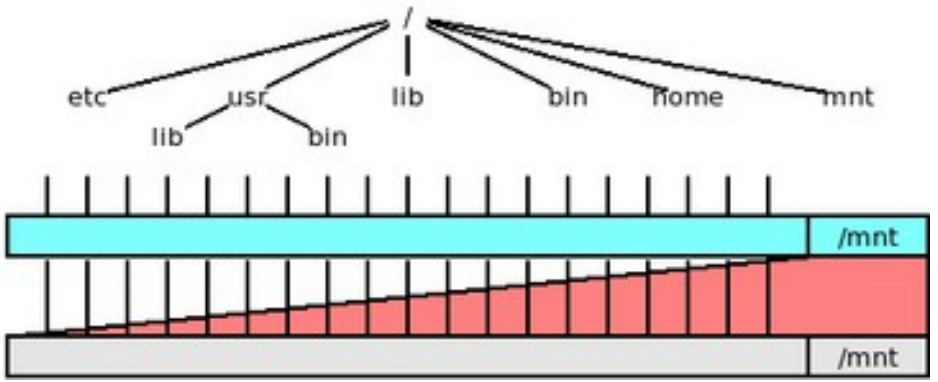


VUOS modules

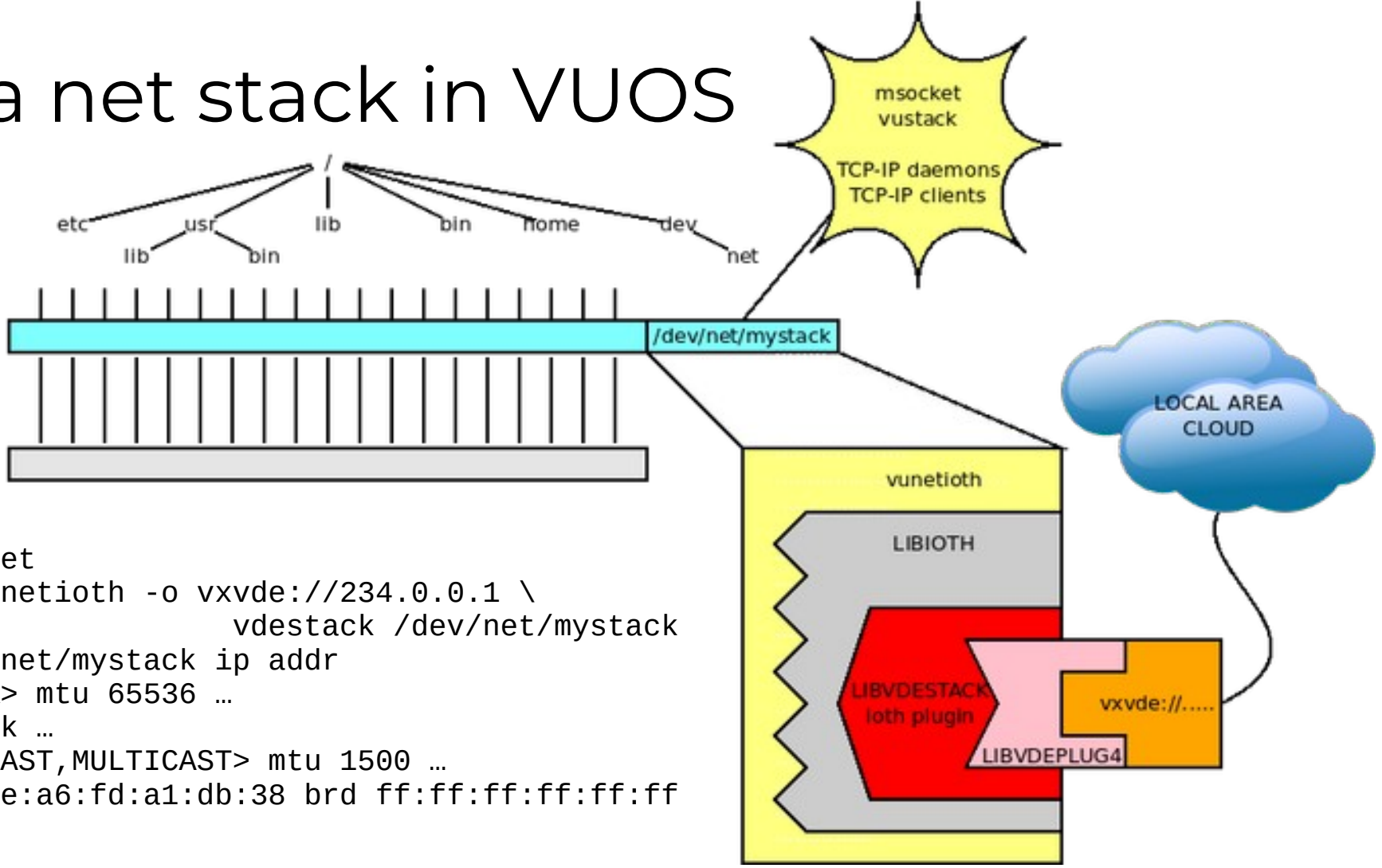
- `vufs`: file system patchworking
- `vufuse`: FUSE for VUOS
- `vudev`: virtual devices
- `vunet`: virtual networking
- `vumisc`: virtual uname/time
- `vubinfmt`: virtual binfmt-misc
- …

`mount` in VUOS

```
$$ vu_insmod vufs
$$ vumount -t vufs / /mnt
$$ ls /mnt
bin   dev  home  lib32  libx32      media  opt   root  sbin  sys  usr
boot  etc  lib   lib64  lost+found  mnt    proc  run   srv   tmp  var
$$ ls /
bin   dev  home  lib32  libx32      media  opt   root  sbin  sys  usr
boot  etc  lib   lib64  lost+found  mnt    proc  run   srv   tmp  var
$$
```



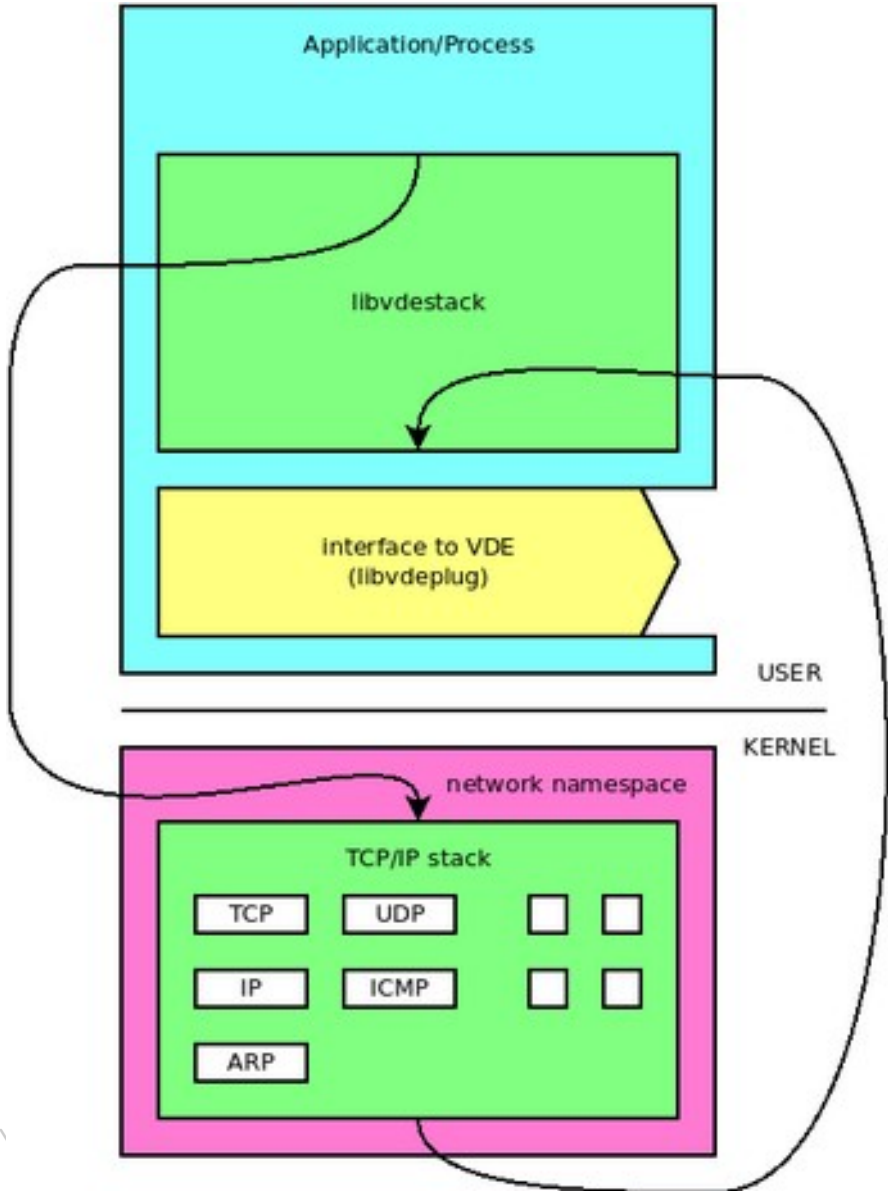
mount a net stack in VUOS




```
$$ vu_insmod vunet
$$ vumount -t vunetioth -o vxvde://234.0.0.1 \
                            vdestack /dev/net/mystack
$$ vustack /dev/net/mystack ip addr
1: lo: <LOOPBACK> mtu 65536 …
    link/loopback …
2: vde0: <BROADCAST,MULTICAST> mtu 1500 …
    link/ether 8e:a6:fd:a1:db:38 brd ff:ff:ff:ff:ff:ff
$$
```

vdestack

- libvdestack is a library which uses the Linux kernel implementation of the TCP-IP and makes it available as a user level library.
- The library “remotely controls” a kernel namespace
- Data link packets are captured through a tap device of the namespace.



picoxnet

- picoxnet is a network stack based on picoTCP-ng designed for the Internet of Thread.
- it supports multiple stacks
- it implements the standard Berkeley Sockets API.
- all the stack/interfaces/addresses/route configuration is supported via netlink
- file descriptors returned by picox_msocket can be used in event-waiting system calls like poll, select and the like...

WIP: iothconf

- A library for ioth (auto) configuration
- It supports:
  - Static configuration (IPv4 and IPv6)
  - DHCP (IPv4, RFC 2131 and 6843)
  - Router Discovery (IPv6, RFC 4861)
  - DHCPv6 (IPv6, RFC 8415 and 4704)

Example: ioth send “ciao\n”

```c
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <ioth.h>
#include <iothconf.h>

int main(int argc, char *argv[]) {
  struct ioth *stack = ioth_newstack("vdestack", "vde:///tmp/hub");
  ioth_config(stack, "eth,ip=10.0.0.1");

  struct sockaddr_in dst = {
    .sin_family = AF_INET,
    .sin_port = htons(5000),
    .sin_addr.s_addr = inet_addr("10.0.0.2")
  };
  int fd = ioth_msocket(stack, AF_INET, SOCK_DGRAM, 0);
  ioth_sendto(fd, "ciao\n", 5, 0, (void *) &dst, sizeof(dst));
  ioth_close(fd);
}
```

WIP: iothconf

- Configuration string examples:
  - Static data:
    `“eth,ip=10.0.0.1/24,gw=10.0.0.2,ip=2001:760::1/64,gw=2001:760::2”`
  - Hash based slaac: `“eth,fqdn=test.v2.cs.unibo.it,rd,slaac”`
  - DCHP: `“eth,dhcp”`
  - Try all the autoconfigurations: `“eth,dhcp,rd,dhcpv6”` or as a shortcut `“auto”`
  - All the autoconfiguration + fqdn and hash slaac:
    `“auto,fqdn=test.v2.cs.unibo.it,slaac”`
- Iothconf provides the following function:

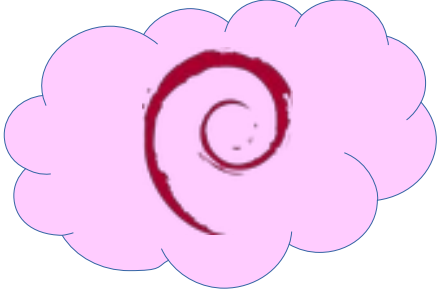
```
char *ioth_resolvconf(struct ioth *stack, char *config);
```

for the configuration of the DNS resolver (the string returned has the syntax of resolv.conf).

WIP iothdns

- Currently: a library to query the dns using ioth stacks + support for client/server side dns message encoding/decoding (RFC1035 and updates)
- Soon: a general purpose dns server/forwarder able to support hash based IPv6 addresses and OTIP (one time IP address).
- (see the proof of concept in vde_dnsutils on GITHUB).

Debian

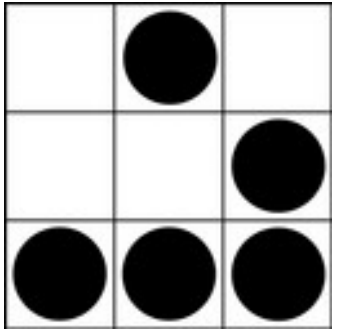


- vde
- fduserdata
- nlinline

- vpoll
- picotcp-ng + picoxnet
- vuos
- libioth
- libnlq

We are still creating art and beauty on a computer:

the art and beauty of revolutionary ideas translated into (libre) code...



renzo, rd235, iz4dje **mikeyg**