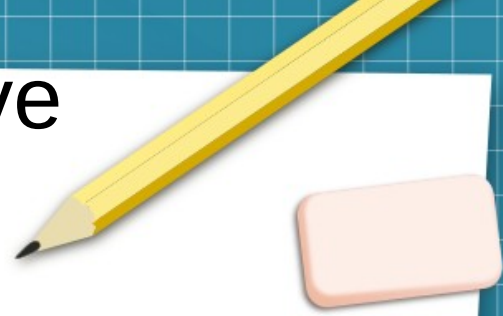




Interprocess Communication (IPC) and Synchronization

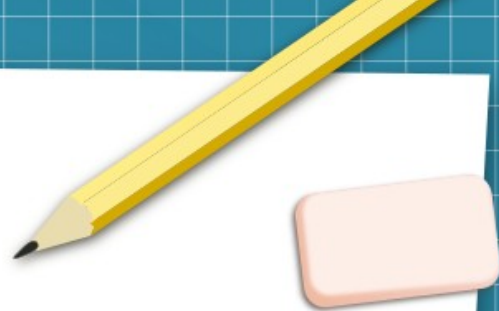
CSE 536 Fall 2026
jedimaestro@asu.edu

IPC and synchronization we've already talked about



- IPC
 - Named and unnamed pipes
 - Sockets
 - Files can be thought of as a form of IPC
- Synchronization
 - Signals
 - Futexes

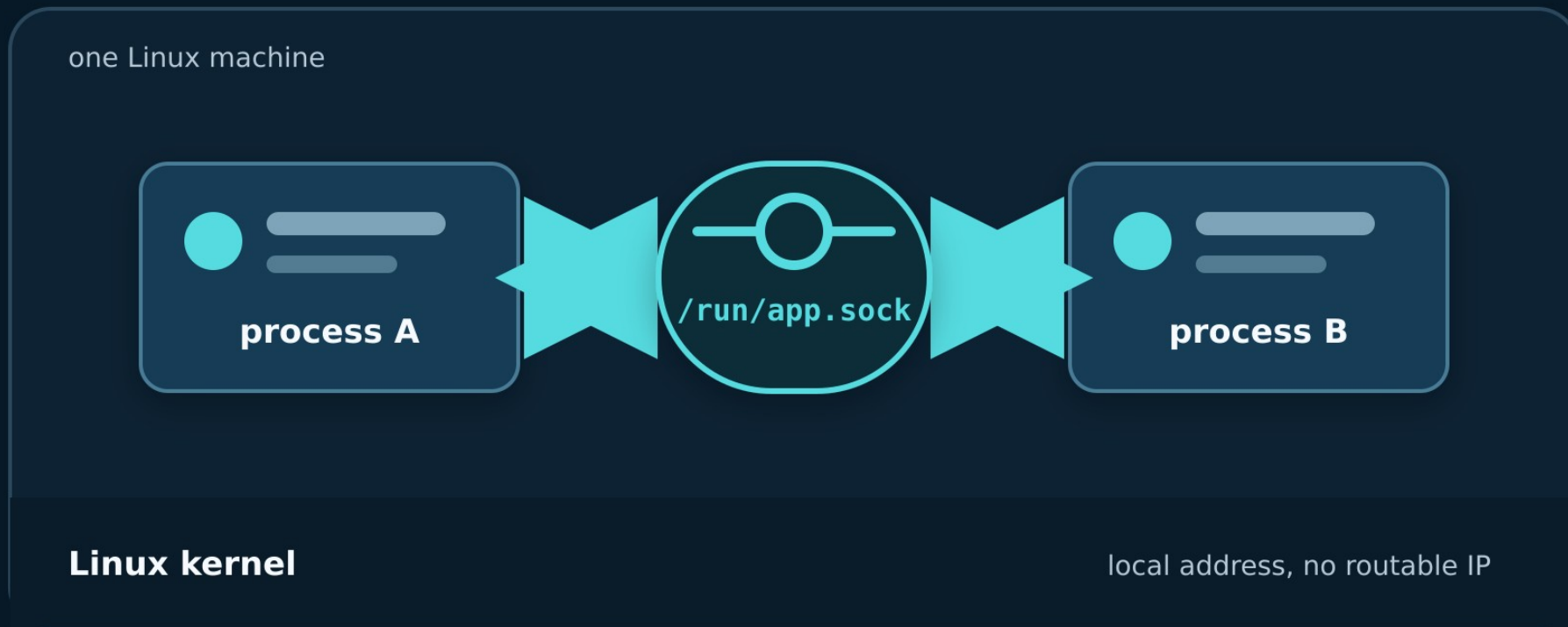
What these slides cover



- Advanced IPC
 - UNIX domain sockets
 - System V and Sun RPC
 - System V and POSIX message queues and shared memory
- Advanced synchronization
 - System V and POSIX semaphores
 - NTSYNC
 - inotify
- Microkernels and message passing, zero-copy
 - D-Bus, XNU, Mach
 - io_uring
 - eBPF

Unix-domain sockets: local IPC

Two processes use the socket API inside one Linux host



What is HAProxy?

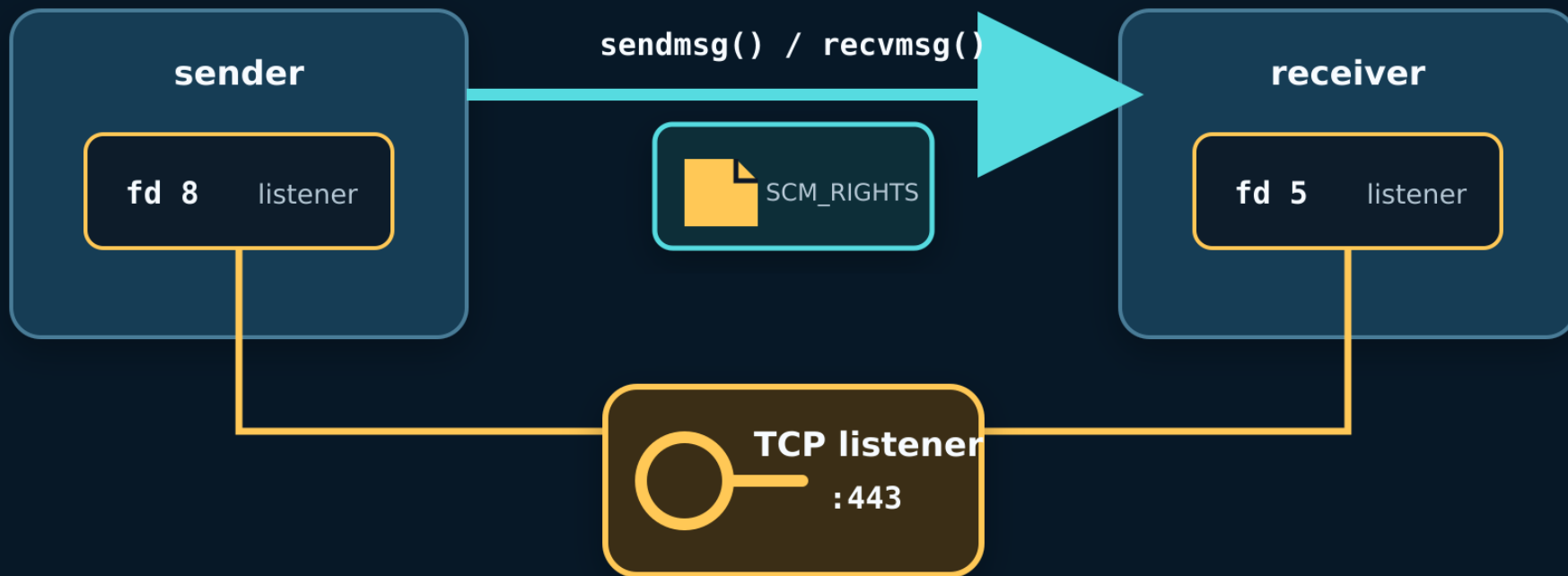
A high-performance, open-source reverse proxy and load balancer



HAProxy accepts the client connection, then opens a connection to one backend.

Unix sockets can pass file descriptors

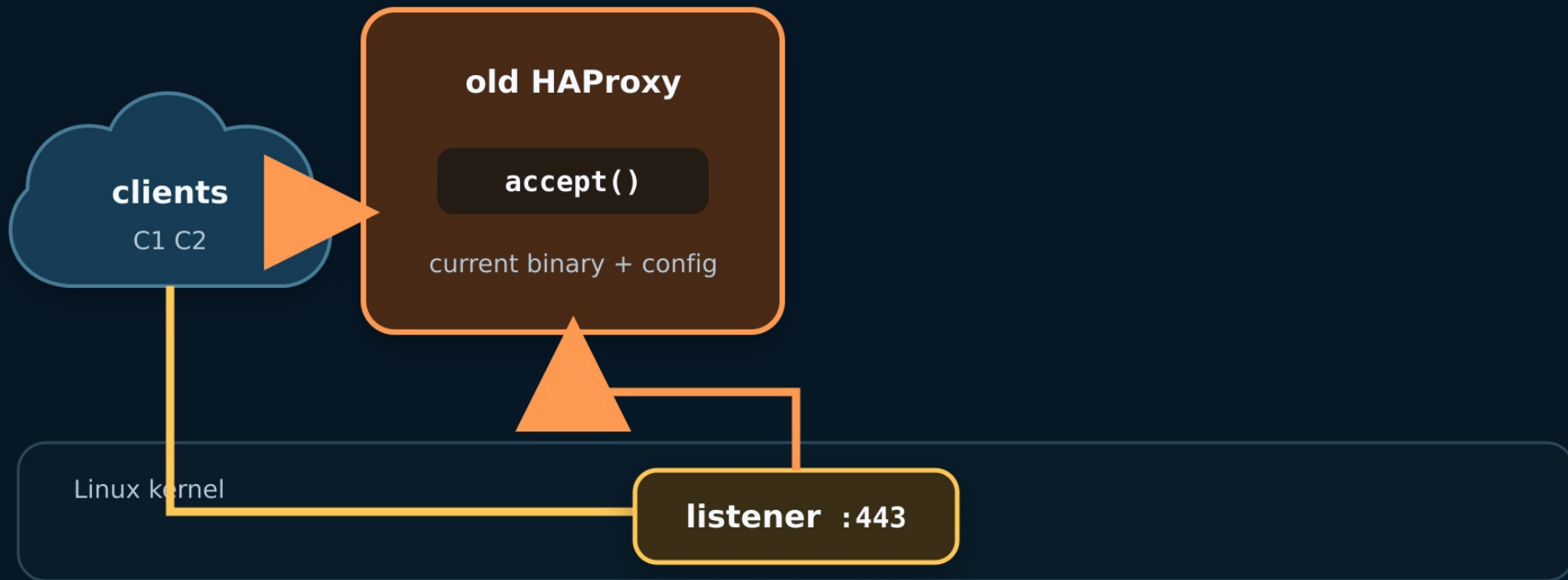
The receiver gets a new reference to the same kernel object



Different FD numbers. Same kernel socket.

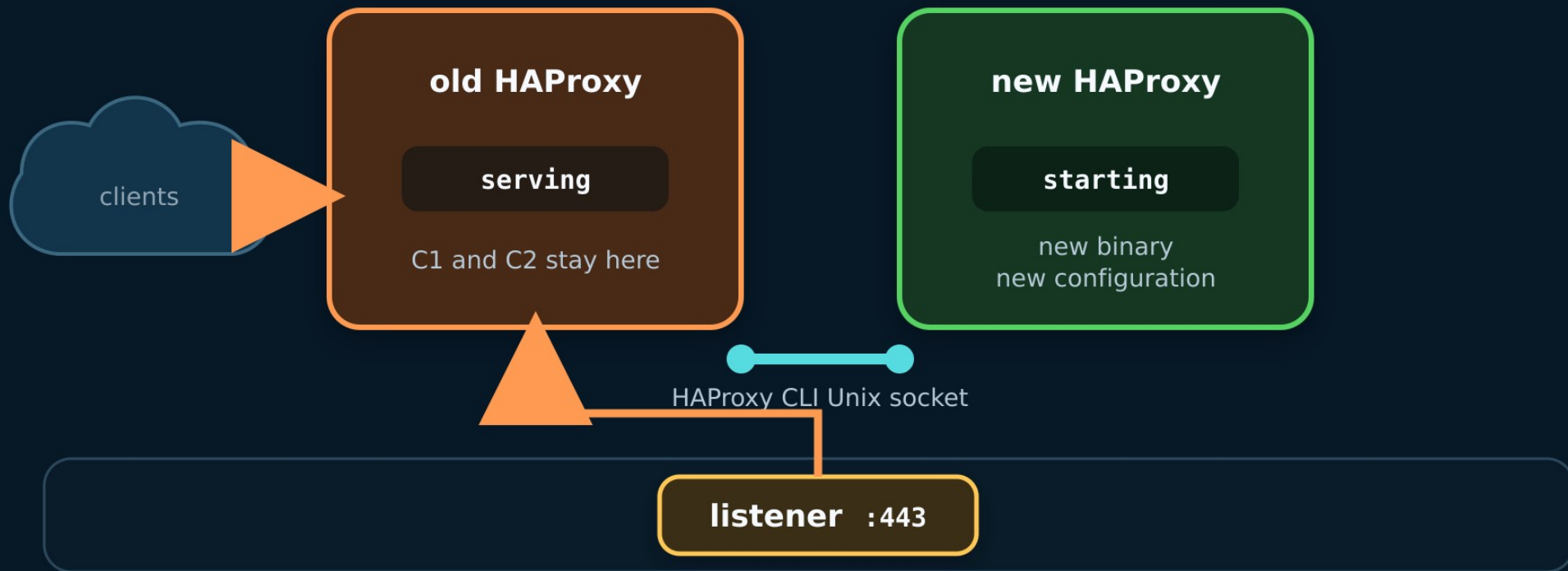
Old HAProxy is live on TCP :443

The listener accepts new clients. The old process owns established sessions.



Reload starts a new HAProxy process

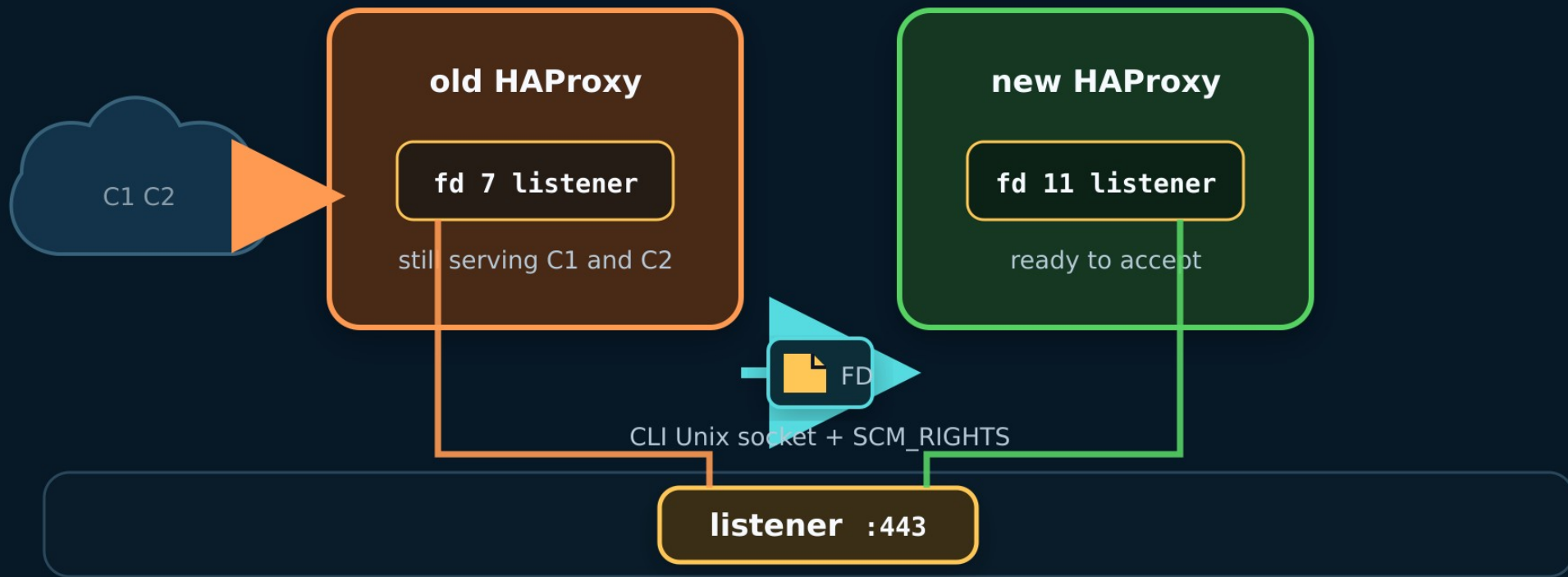
The old process keeps serving while the new process loads code and configuration



start new before stopping old

Pass the listener over the Unix socket

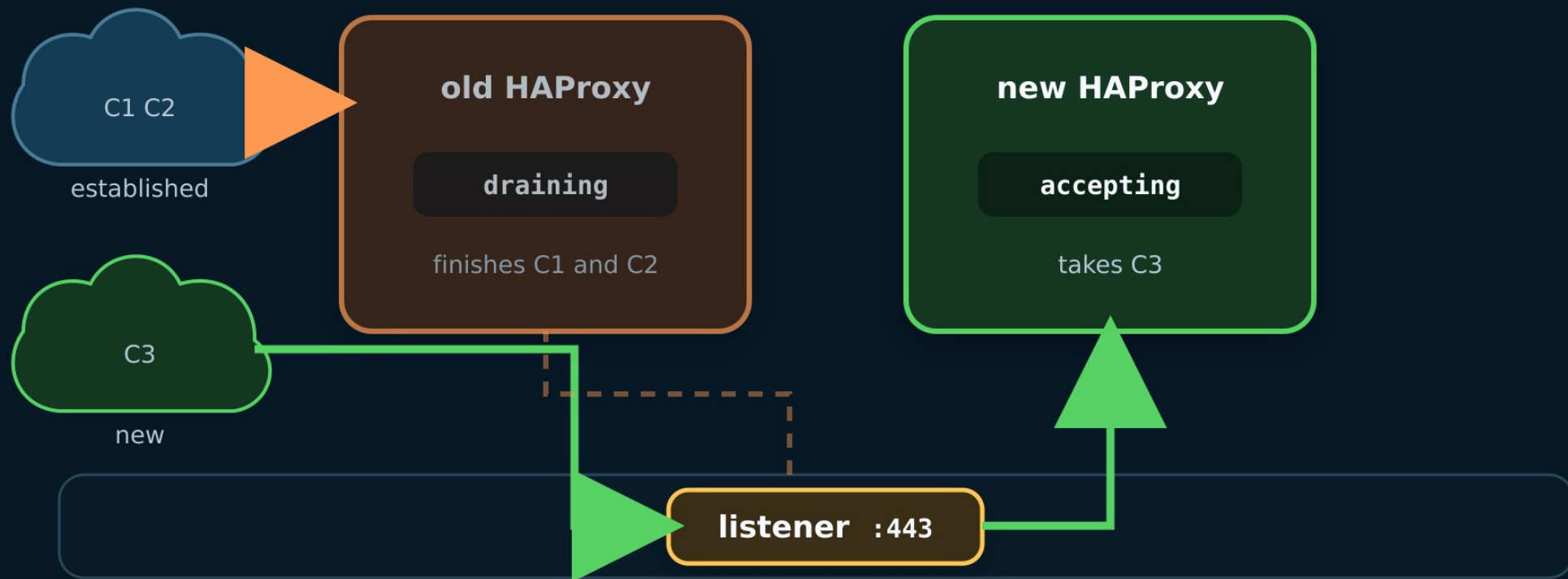
The old process sends the listening FD with SCM_RIGHTS



the listener stays open

New accepts while old drains

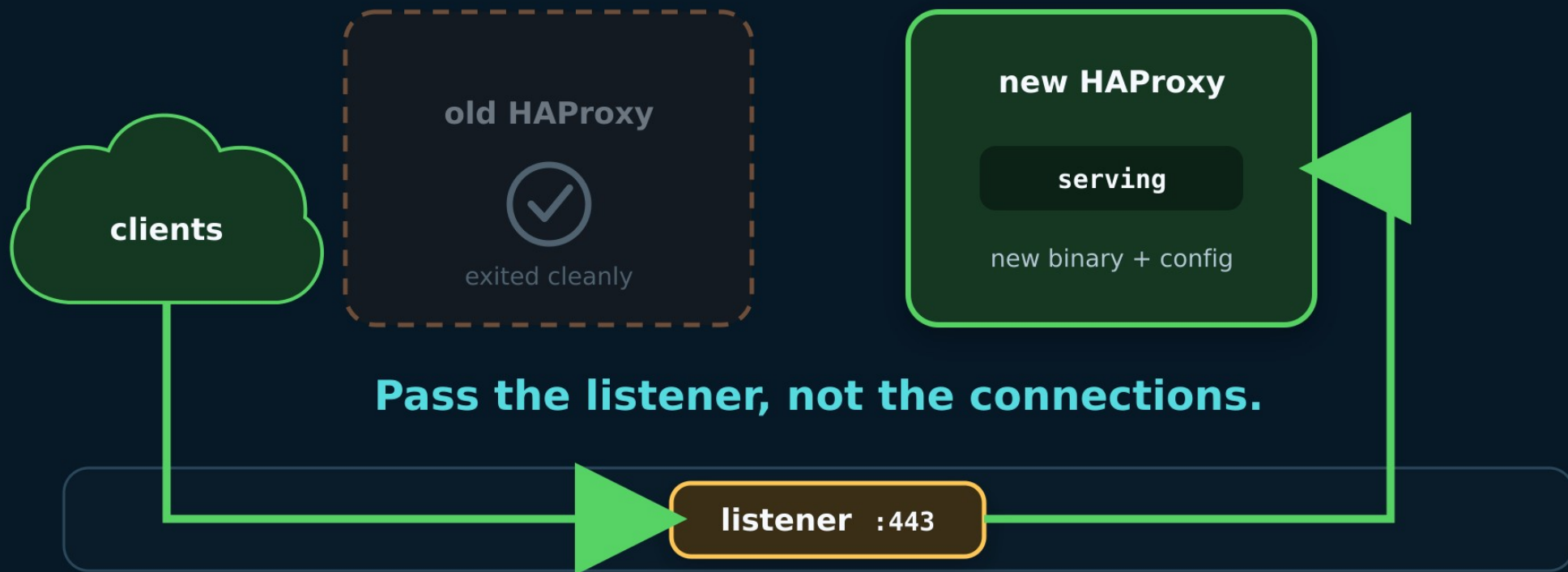
New connections go to the new process. Established sessions stay with the old one.



listener handed off; sessions stay put

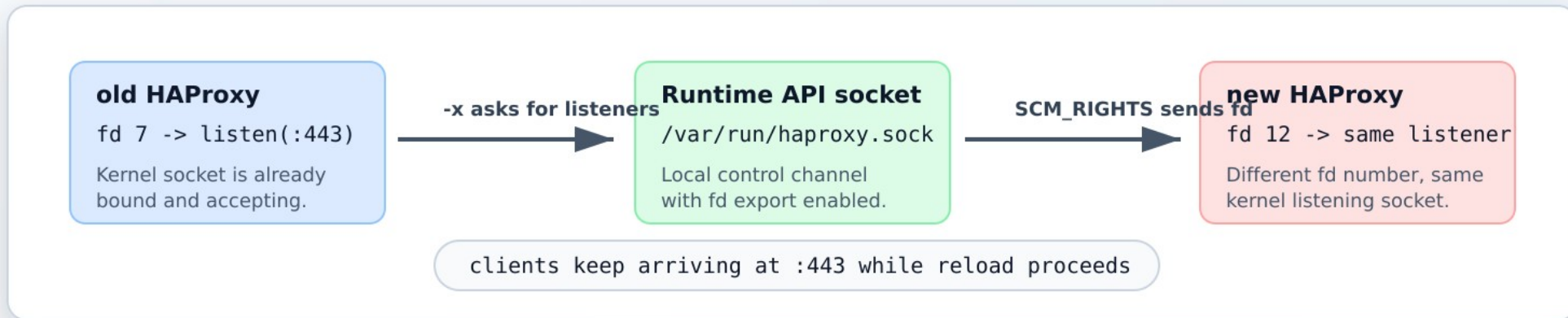
New code, same live port

The old process exits after its sessions finish. The new process keeps the listener.



HAProxy port 443 handoff via Unix sockets

Hitless reload: the old process passes the already-bound listener FD to the new process.



1. Enable fd export

HAProxy Runtime API / reload docs

```
stats socket /run/haproxy.sock
mode 600 expose-fd listeners
haproxy ... -x /run/haproxy.sock
```

The reload asks the old process for listener FDs.

2. HAProxy selects exportable FDs

HAProxy · src/protocol.c

```
if (!(fdtab[rx->fd].state
& FD_EXPORTED))
continue;
rx_agent_getsocks_request(...);
```

Only exportable listener FDs are gathered.

3. Unix socket passes the handle

Linux/Unix ancillary data · SCM_RIGHTS

```
cmsg->cmsg_type = SCM_RIGHTS;
memcpy(CMSG_DATA(cmsg),
&fd_443, sizeof(int));
sendmsg(unix_sock, &msg, 0);
```

The receiver gets a new fd for the same socket.

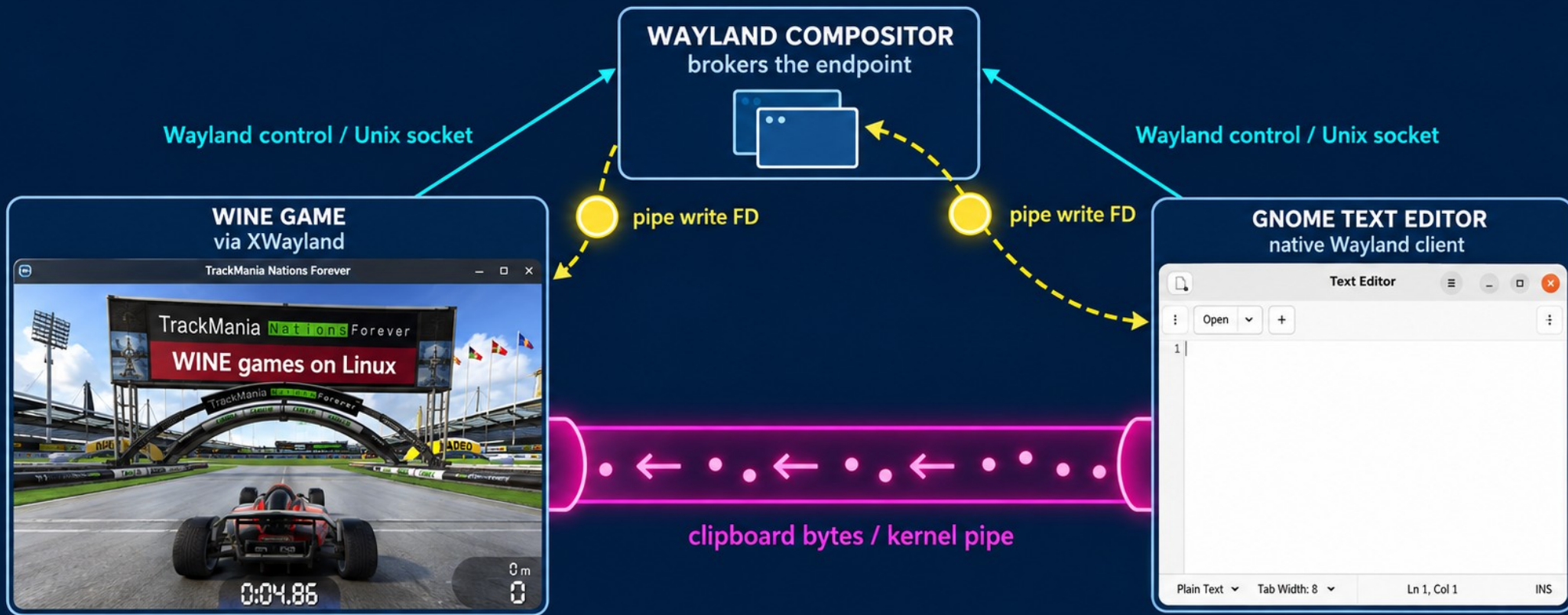
This is channel/capability passing: the message does not contain port 443 traffic. It transfers authority to accept on the existing listener.

Sources: HAProxy Runtime API/docs, HAProxy src/protocol.c, Linux cmsg(3) SCM_RIGHTS

16:9 slide PNG + editable SVG

Wayland: a channel sent through a channel

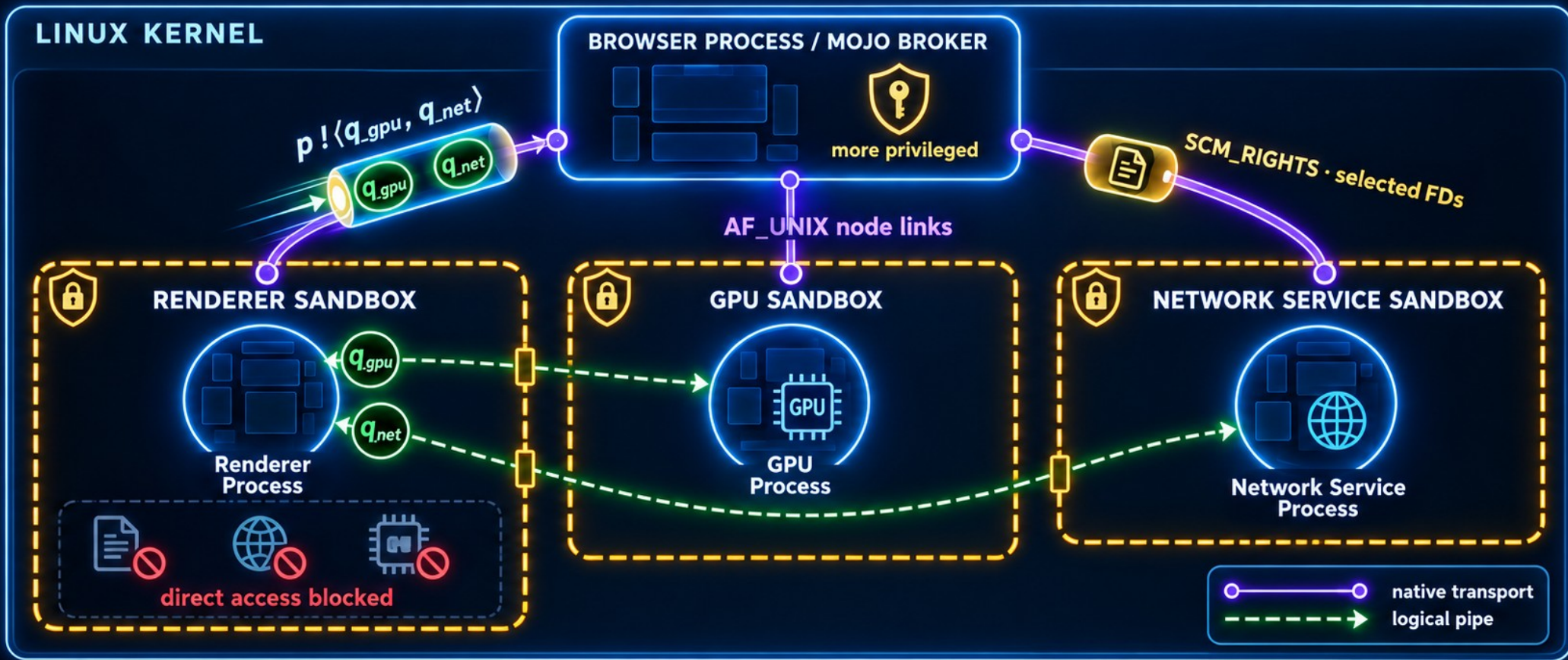
Clipboard and drag-and-drop can relay a pipe endpoint between desktop clients.



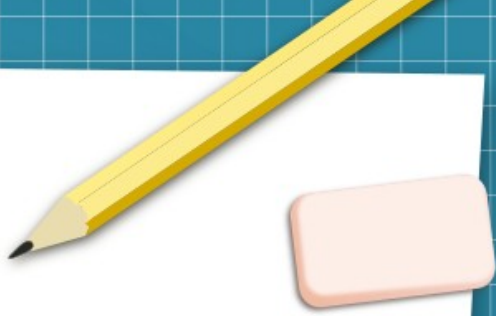
π -calculus analogy: communicating a channel creates a new path.

CHROMIUM · MOJO · LINUX

Chromium Mojo across Linux sandboxes



Endpoints grant communication, not ambient OS access.



Remote Procedure Calls (RPC)...

What Is RPC?

Remote Procedure Call: make a call here; run the code over there.

RPC makes a remote action look like a local function call.



It hides: encoding, transport, dispatch

It does not hide: latency, failures, versions

----- The hard part is that a "call" can be delayed, duplicated, lost, retried, or partially complete. -----

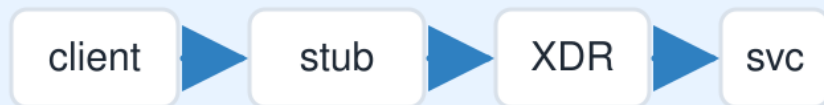
Sun RPC vs. “System V RPC”

One is the famous Unix network RPC; the other phrase usually points to System V IPC or SVR4 TI-RPC.

RPC means: local code calls a procedure, but the work runs in another process or machine.

Sun RPC / ONC RPC

A real RPC protocol family

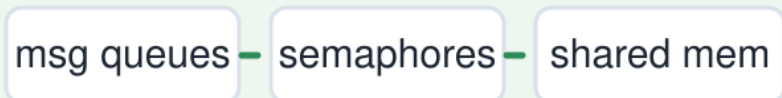


Teaches:

- Interface calls become network messages
- Commonly over UDP or TCP
- Famous because of NFS

System V

Mostly remembered for local IPC primitives



SVR4 twist:

TI-RPC adapted ONC RPC to transport-independent APIs.

So: “System V RPC” usually means System V IPC, or SVR4 TI-RPC.

Takeaway: Sun/ONC RPC is the network-call system; System V is the local IPC toolbox, with later transport-independent RPC APIs.

Modern Linux RPC: ONC/Sun RPC

The living Unix-style example is NFS user space: libtirpc APIs, rpcbind/portmap discovery, XDR data, request/response calls.

1. Pull in the RPC interface

nfs-utils · showmount.c

```
#include <rpc/rpc.h>
#include <rpc/pmap_prot.h>

#include <mount.h>
#include "nfsrpc.h"

// ONC RPC client API
// XDR encodes data
```

Old headers; modern Linux usually gets them from libtirpc.

2. Create a CLIENT handle

nfs-utils · nfs_get_mount_client()

```
rpcprog_t program =
nfs_getrpcbyname(
MOUNTPROG, mount_pgm_tbl);

client = clnt_create(
hostname, program, vers,
"tcp");

if (!client)
client = clnt_create(...);
```

The library finds the service and binds a TCP or UDP transport.

3. Call the remote procedure

nfs-utils · showmount export path

```
mclient->cl_auth =
nfs_authsys_create();

clnt_stat = clnt_call(
mclient, MOUNTPROC_EXPORT,
(xdrproc_t) xdr_void, NULL,
(xdrproc_t) xdr_exports,
(caddr_t) &exportlist,
total_timeout);
```

The call names a procedure plus XDR encoders for args/results.

rpcbind / portmap lookup



CLIENT handle



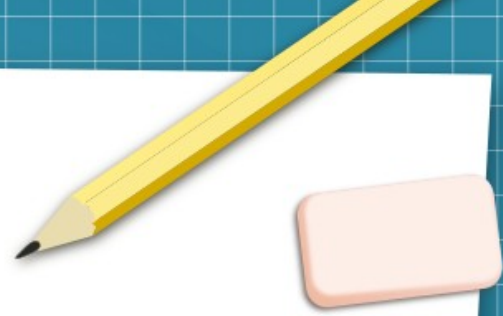
XDR marshaling + clnt_call()

For new app RPC, people often pick gRPC/HTTP. For classic Unix/Linux RPC, the name you will actually encounter is ONC/Sun RPC.

Source snippets: github.com/linux-nfs/nfs-utils, [utils/showmount/showmount.c](#)

16:9 slide PNG + editable SVG

Message queues...



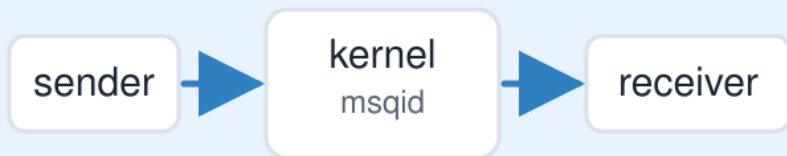
System V vs. POSIX Message Queues

Both pass discrete messages through the kernel. Neither passes channels by itself.

Message queue = sender writes one record; receiver later reads one record.

System V MQ

Older Unix IPC API

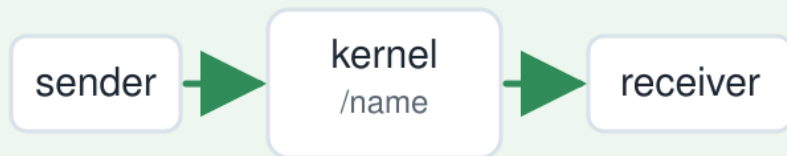


Message shape:

- long type + bytes
- Receiver can filter by type

POSIX MQ

Cleaner standardized API



Message shape:

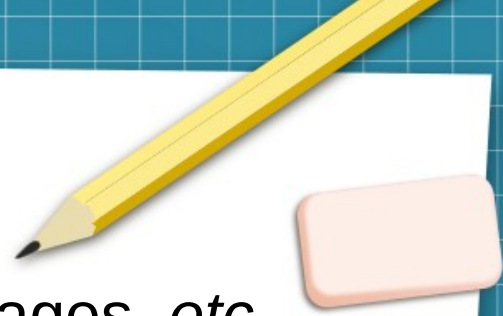
- priority + bytes
- Optional notification on arrival

Not channel passing:

queues can carry a name or ID, but not an open endpoint/capability.

Unix domain sockets with SCM_RIGHTS are the classic Unix mechanism for sending live file descriptors.

Message queues



- Can subscribe to channels, certain types of messages, *etc.*
 - UNIX domain sockets are between two processes, but do the cool passing of file descriptors, which is more like the cool pi calculus kind of message passing we'll cover later this semester
 - Use POSIX message queues if you want priority, asynchronous `mq_notify()`

POSIX message queues in user-space code

An ESP-IDF example opens a named queue, sends integers from one thread, and receives them in another.

1. Create/open the queue

ESP-IDF · posix_mqueue_example_main.c

```
struct mq_attr
configuration = {
    .mq_maxmsg = 10,
    .mq_msgsize = sizeof(int),
};

write_descr = mq_open(
    "/my_queue",
    O_CREAT | O_WRONLY,
    S_IRUSR | S_IWUSR,
    &configuration);
```

A name plus attributes creates the queue.

2. Sender thread posts data

ESP-IDF · sender_function()

```
const unsigned int
MSG_PRIO = 0;

int result = mq_send(
    *write_descr,
    (const char*) &s_counter,
    sizeof(s_counter),
    MSG_PRIO);
```

The payload bytes are copied into the queue.

3. Receiver thread blocks

ESP-IDF · receiver_function()

```
int msg;

int result = mq_receive(
    *read_descr,
    (char*) &msg,
    sizeof(msg),
    NULL);

printf("received: %d\n", msg);
```

The receiver gets a message-sized copy out.

`mq_open("/my_queue")`



`mq_send(&s_counter)`



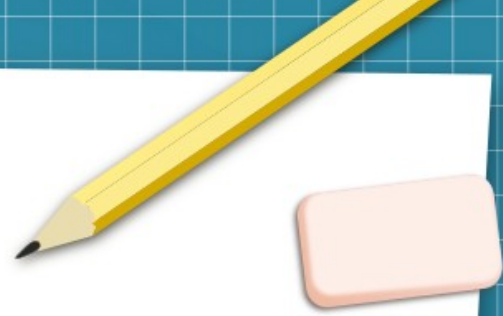
`mq_receive(&msg)`

This is message passing, not channel passing: bytes go through a named queue; descriptors do not.

Source snippets: [github.com/espressif/esp-idf, examples/system/rt_mqueue/main/posix_mqueue_example_main.c](https://github.com/espressif/esp-idf/blob/main/examples/system/rt_mqueue/main/posix_mqueue_example_main.c)

16:9 slide PNG + editable SVG

Shared memory...



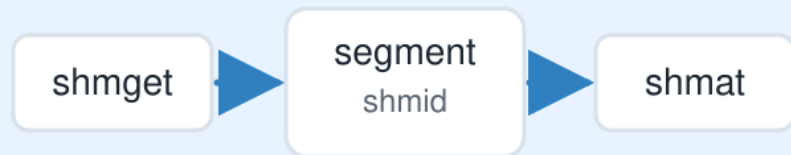
System V vs. POSIX Shared Memory

Both map the same physical pages into more than one process.

Shared memory is fast IPC because data does not have to be copied through a pipe or socket.

System V SHM

Old Unix IPC: key/id object

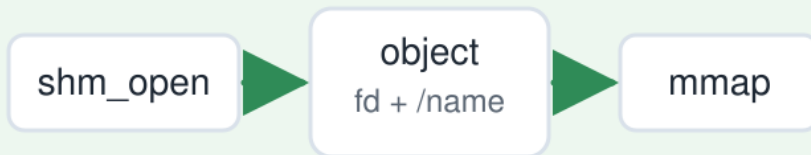


Teaches:

- Kernel object outlives processes
- Cleanup via shmctl(IPC_RMID)

POSIX SHM

File-like API: name + fd + mmap



Teaches:

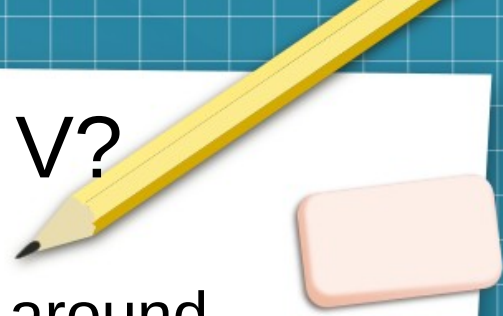
- Fits Unix fd/file permissions
- Cleanup via shm_unlink()

The catch: shared memory moves bytes, not events.

You still need synchronization: futexes, pthread mutexes, semaphores, eventfd, or another wakeup path.

Why POSIX instead of System V?

- `memfd_create()` means you can pass memory around through UNIX domain sockets



POSIX shared memory in real code

PyTorch maps shared tensor storage by naming an object, sizing it, then mapping the same pages.

1. Open a named object

PyTorch · aten/src/ATen/MapAllocator.cpp

```
if ((fd = shm_open(
  filename_.c_str(),
  flags,
  static_cast<mode_t>(
    0600))) == -1) {
  TORCH_CHECK(false,
    "unable to open shm");
}
```

The name resolves to a file descriptor.

2. Give it a size

PyTorch · MapAllocator.cpp

```
if (static_cast<int64_t>(
  size) > file_stat.st_size) {
  if (ftruncate(fd,
    static_cast<off_t>(
      size)) == -1) {
    TORCH_CHECK(false,
      "unable to resize");
  }
}
```

Shared-memory objects start with a size problem.

3. Map pages into memory

PyTorch · MapAllocator.cpp

```
if (flags_ &
  ALLOCATOR_MAPPED_SHAREDMEM) {
  base_ptr_ = mmap(
    nullptr, size_,
    PROT_READ|PROT_WRITE,
    MAP_SHARED, fd, 0);
}

// later: munmap + shm_unlink
```

Different virtual addresses, same backing pages.

shm_open(name)



ftruncate(fd, size)



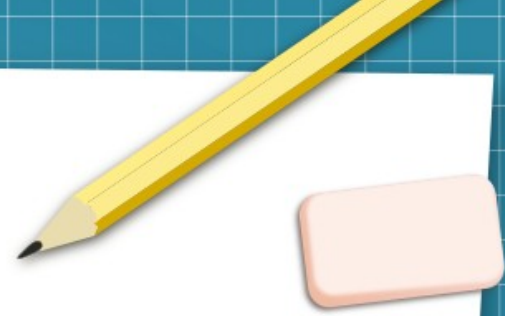
mmap(..., MAP_SHARED, fd)

The pointer is local to each process; the bytes behind it are shared. You still need synchronization.

Source snippets: github.com/pytorch/pytorch, `aten/src/ATen/MapAllocator.cpp`

16:9 slide PNG + editable SVG

Synchronization...



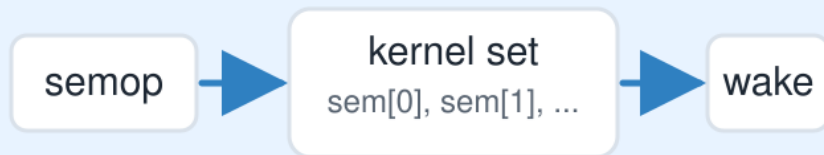
System V vs. POSIX Semaphores

Both are counting semaphores. Binary semaphores are the count = 1 case.

Semaphore = a counter that can make callers sleep until a resource is available.

System V Semaphores

Old Unix IPC: semaphore sets

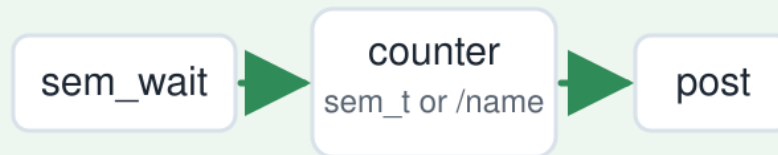


Advanced bits:

- Atomic operations across a set
- Can wait for zero; SEM_UNDO

POSIX Semaphores

Cleaner API: named or unnamed



Advanced bits:

- Named: kernel object with lifetime
- Unnamed: lives in shared memory

Teach this:

semaphores count resources; mutexes own critical sections.

No owner means anyone can post; use care with priority inversion, leaks, and cleanup.

POSIX semaphores in real code

PostgreSQL wraps POSIX semaphores around a small counter: wait decrements, post increments.

1. Initialize the counter

PostgreSQL · src/backend/port/posix_sema.c

```
static void
PosixSemaphoreCreate(
sem_t *sem)
{
    if (sem_init(sem, 1, 1) < 0)
        elog(FATAL,
             "sem_init failed");
}
```

// pshared=1, value=1

Here it behaves like a process-shared mutex.

2. Wait: decrement or sleep

PostgreSQL · PGSemaphoreLock()

```
do
{
    errStatus = sem_wait(
PG_SEM_REF(sema));
} while (errStatus < 0
        && errno == EINTR);

if (errStatus < 0)
    elog(FATAL, "sem_wait...");
```

Interrupted waits are retried.

3. Post: increment or wake

PostgreSQL · PGSemaphoreUnlock()

```
do
{
    errStatus = sem_post(
PG_SEM_REF(sema));
} while (errStatus < 0
        && errno == EINTR);

if (errStatus < 0)
    elog(FATAL, "sem_post...");
```

Posting releases a permit or wakes a blocked waiter.

sem_init(value=1)



sem_wait(): decrement / block

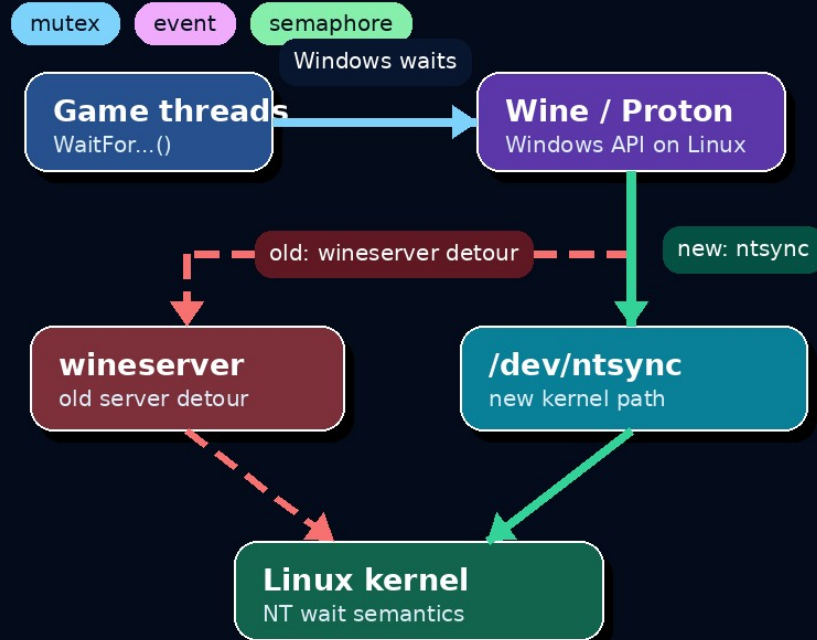


sem_post(): increment / wake

POSIX semaphores are counting semaphores; using initial value 1 makes one act like a binary lock.

ntsync for Wine / Proton

Kernel help for Windows-style wait objects



One-slide version: ntsync is not gameplay code; it is kernel plumbing for Wine sync o

Generated PG game scene; no real game screenshot or branding used.

Dirt 3 example: 110 -> 860 FPS

benchmark from the ntsync kernel patch discussion

Inside /dev/ntsync: wait semantics

One kernel operation: test objects -> sleep if needed -> acquire atomically

WaitForMultipleObjects([sem, mutex, event], ANY / ALL)

state changes on success

Semaphore

count > 0 → count--

Mutex

owner = 0 → owner = T7

Auto event

signaled → reset to 0

WaitAny: acquire one ready object

WaitAll: acquire all together, or modify none

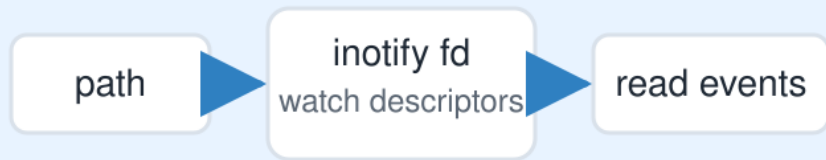
inotify vs. kqueue

Linux and BSD answers to “tell me when this filesystem object changes.”

Both are kernel event queues. Neither is a System V or POSIX IPC primitive.

Linux inotify

Filesystem-specific watch API

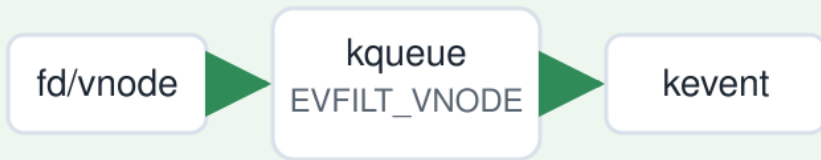


Flavor:

- Directory entries: create/delete/move
- Linux fd fits poll/epoll loops

BSD kqueue

General event queue with EVFILT_VNODE



Flavor:

- Vnode/file descriptor centric
- Same queue can watch files, sockets, timers

Teach this: event APIs are hints; robust programs rescan after overflow or tricky renames.

Closest cousins: Linux fanotify for broader monitoring, macOS FSEvents for tree-scale watching.

inotify in real code

Linux exposes filesystem changes as records read from a file descriptor.

1. Open the notification stream

inotify-tools · inotify-nosys.h

```
static inline int
inotify_init(void) {
    return syscall(
        __NR_inotify_init);
}

// fd is pollable/readable
fd = inotify_init();
```

One descriptor becomes the event queue.

2. Attach paths to that fd

inotify-tools · inotify-nosys.h

```
static inline int
inotify_add_watch(
    int fd, const char *name,
    uint32_t mask) {
    return syscall(
        __NR_inotify_add_watch,
        fd, name, mask);
}
```

A watch descriptor names that path.

3. Decode event records

inotify-tools · inotify-nosys.h

```
struct inotify_event {
    int wd;
    uint32_t mask;
    uint32_t cookie;
    uint32_t len;
    char name[];
};

// mask says what happened
```

read(fd) returns variable-length event records.

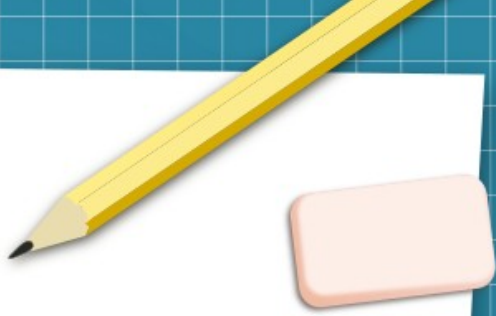
inotify_init()



inotify_add_watch(fd, path, mask)



read(fd) -> events

A yellow pencil and a pink eraser are positioned in the top right corner of the white paper, suggesting a drawing or writing activity.

Microkernels...

<https://dl.acm.org/doi/pdf/10.1145/234215.234473>

TOWARD REAL MICROKERNELS

*The inefficient, inflexible first generation inspired
development of the vastly improved second generation, which
may yet support a variety of operating systems.*

J o c h e n L i e d t k e

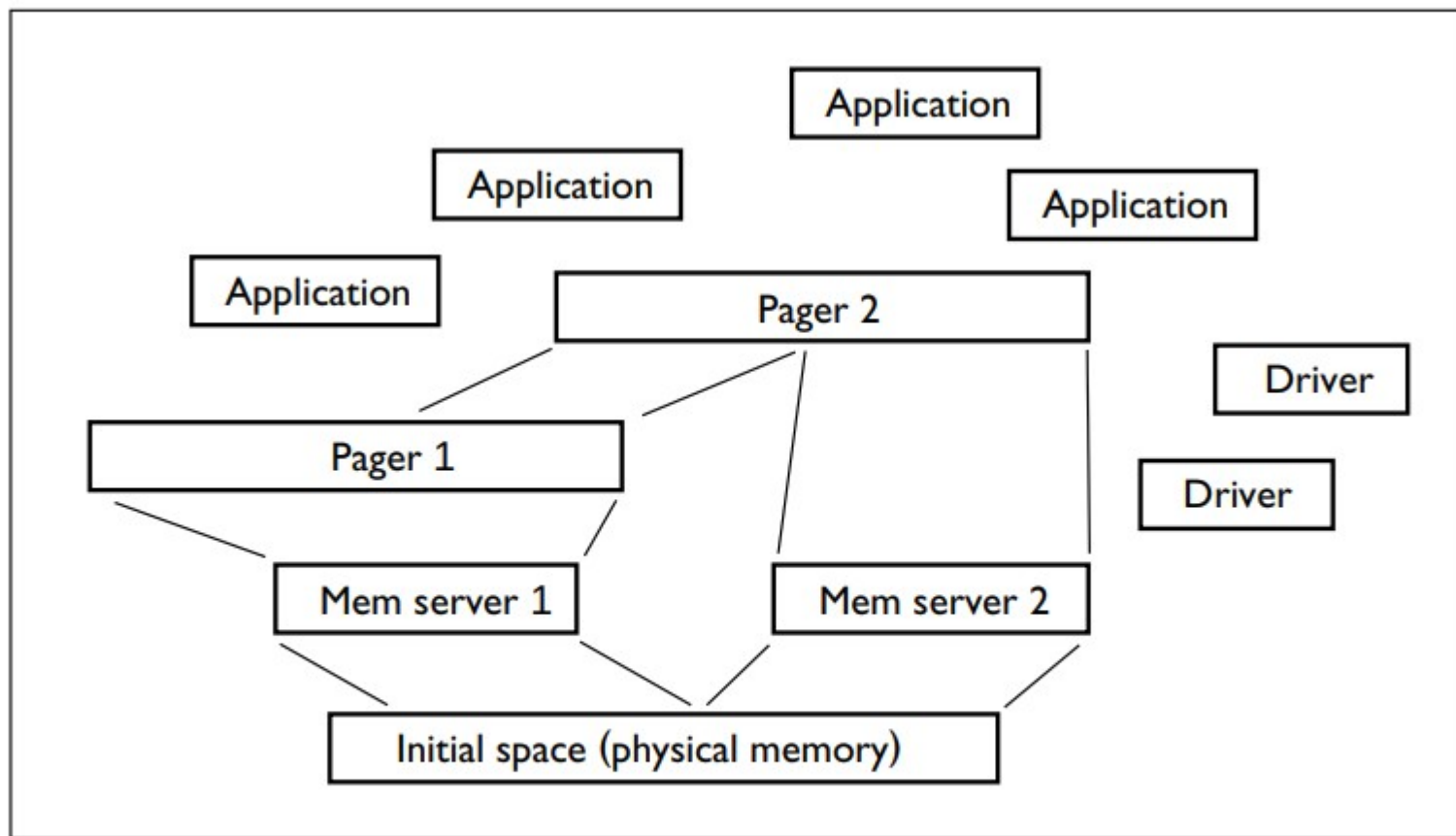
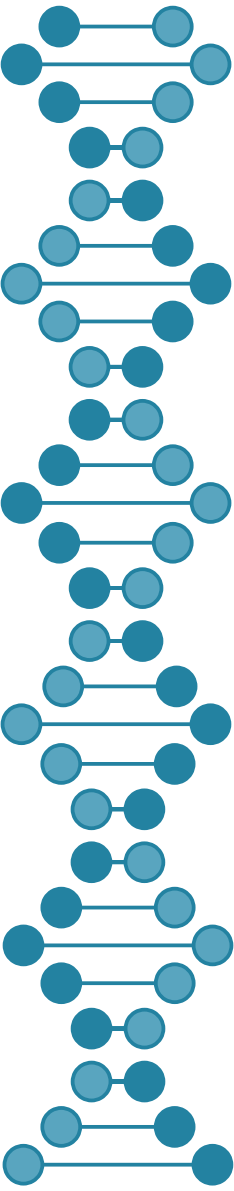


Figure 6. A maps page by IPC to B



A little MacOS history...

https://en.wikipedia.org/wiki/Altair_8800
(1974)



https://en.wikipedia.org/wiki/Apple_I (1976)



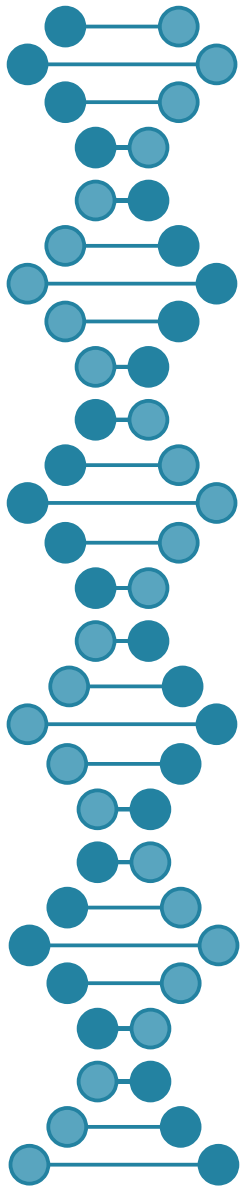
Because the Apple I did not include a case, customers needed to supply their own.

https://en.wikipedia.org/wiki/Apple_II
(1977)



https://en.wikipedia.org/wiki/Apple_Lisa
(1983)





STEVE JOBS: OUT FOR REVENGE

By Phil Patton



Jobs at his NeXT factory in Fremont, Calif. Each circuit board is assembled "unfused by human hands."

He has named this computer NeXT. "What we want," he tells the audience, "is to create the next computing revolution. We want to push the envelope." The same NeXT echoes his claim to the second standard in the industry — a PC with unprecedented power and versatility and an innovative programming system — but it is also an undignified reference to currency about the next chapter in the story of Steve Jobs.

In 1976, at the age of 21, Steven Paul Jobs co-founded Apple Computers with Stephen G. Wozniak, five years his senior, whom Jobs had known since he was a sophomore at Hotchkiss High School, an California's Silicon Valley. Within five years, Apple had become a billion-dollar company. Then in 1985 Jobs was forced out — by John Sculley, whom Jobs himself had hired two years before to be the company's chief executive. Ever since, working in almost total secrecy, Jobs had been preparing a comeback. Now, at age 24, no longer the boy wonder of the computer industry, he was starting over.

IN SILICON VALLEY, A COMPUTER IS CALLED A "box," a sign that the guts may be less important than the skin. The guts of Jobs's new machine are housed in a ribbed black magnesium cube. Keyboard and monitor are separate, connected by cables, the 17-inch screen dramatically cradled over a swooping support. "Computers," Jobs likes to say, "are the metaphor of our time. They should share a certain higher aesthetic."

Not that what is inside the NeXT box is unimportant: a Phil Patton is at work on a book about American design.

new special memory system that uses a laser to move and read up to 700 volumes' worth of information on a single disk, a sound system of CD quality, a powerful array of sophisticated processing chips, and innovative software.

"What other computer can you sit down to," he says, referring to the "Digital Librarian" feature of the NeXT machine, "and blast through the complete works of William Shakespeare in just a couple of seconds?"

Although it looks like a personal computer, the NeXT machine is much more powerful than any PC on the market. It has the capabilities of computers known as workstations, previously used mostly by engineers and scientists. Like most workstations, it employs Unix, an aging but powerful basic software system. But at \$10,000 it costs much less than most workstations of comparable power.

Business has traditionally favored the Unix system, and it still accounts for just 9 percent of the computer market today. But, thanks in part to NeXT, Unix is expected to move that double its share in the next five years. Jobs has increased its complexity in a new variation of the software, a "user shell" that will make it, he says, "usable by more mortals." Called NeXTing, the software employs a "finder" key approach that allows novices as well as experts to combine pre-existing sections of instructions, or "objects," to create the programs they need — an innovative technique known in the industry as "object-oriented programming."

What may be most revolutionary about NeXT, however, is not its technology but the fact that it is the first computer to be sold primarily on the strength of its design.

Critics and champions alike — and Jobs has plenty of each — agree that he has always been at his best as a salesman and evangelist for the computer. Joanna Beddison, who worked with Jobs at Apple and at NeXT, says, "In some ways, Steve got philosophical the way the Greeks did. He always wanted the best. He had these aesthetic notions of perfect shapes and perfect sounds. It is almost Platonic."

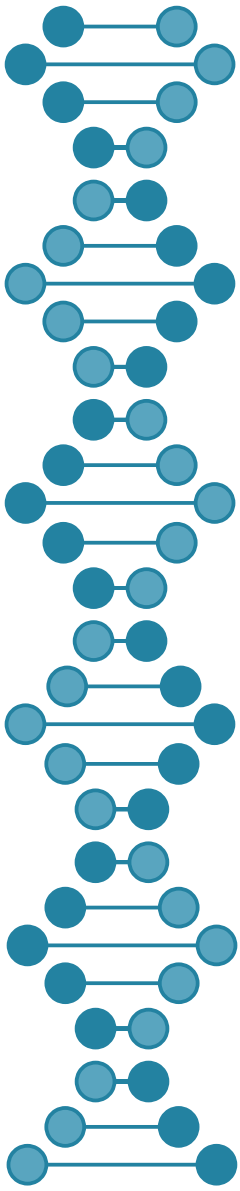
Jobs's obsession with detail, with appearance, is part of his legend. When a small imperfection showed up on the first samples of the computer's case, he flew to Chicago to work with the die maker. At NeXT's automated factory in Fremont, Calif., he had the machines repeatedly re-adjusted to achieve the uniform gray he wanted.

Colleagues may use words like Platonist to describe his aesthetic but for Jobs the technology in "form" and "whiz" — as opposed to "bore" and brain-damaged, he has never claimed to be an is.

Ousted from Apple, the computer legend is back on the fast track with a \$10,000 PC loaded with power.

PHOTOGRAPH

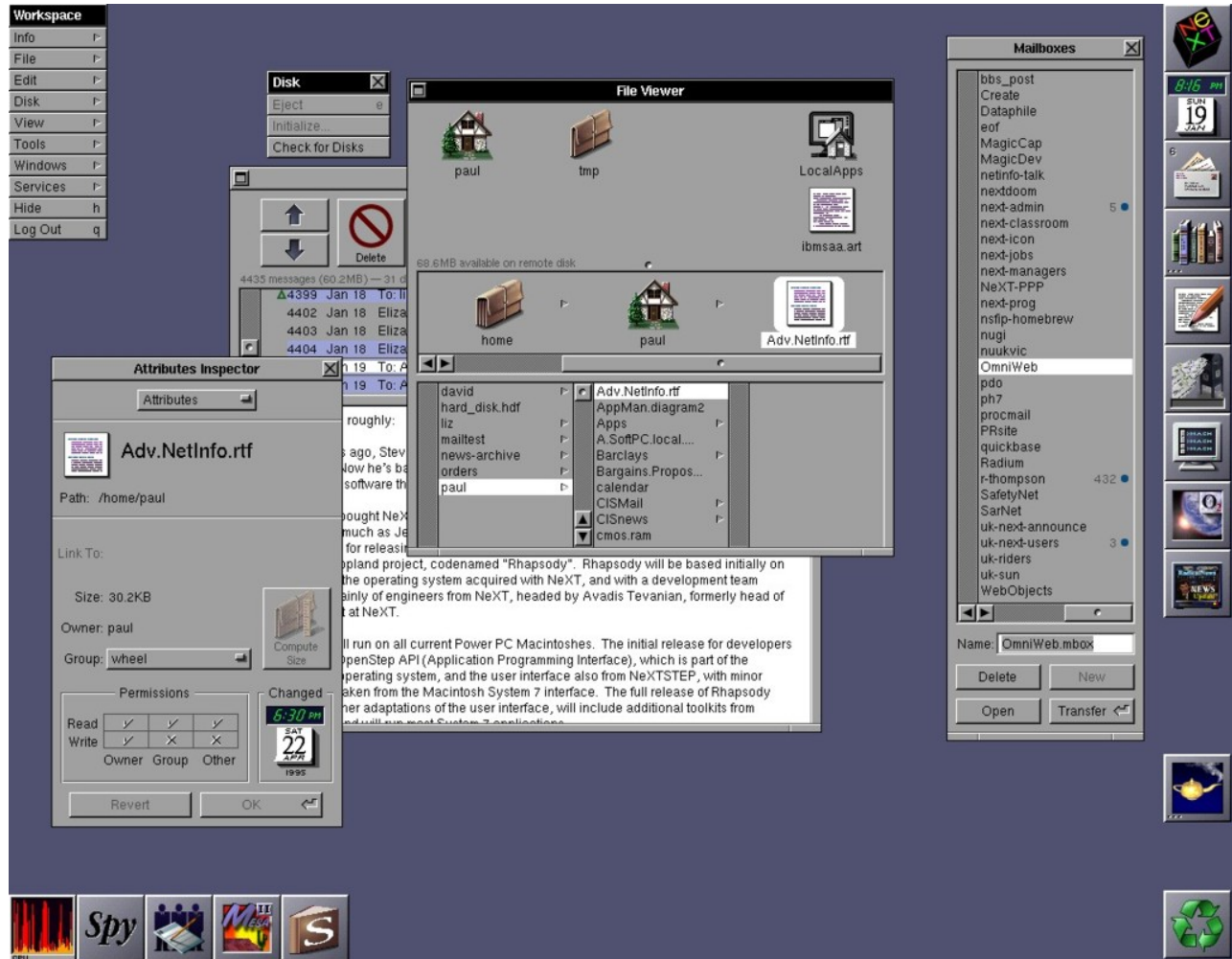
THE NEW YORK TIMES MAGAZINE • AUGUST 1, 1988 33

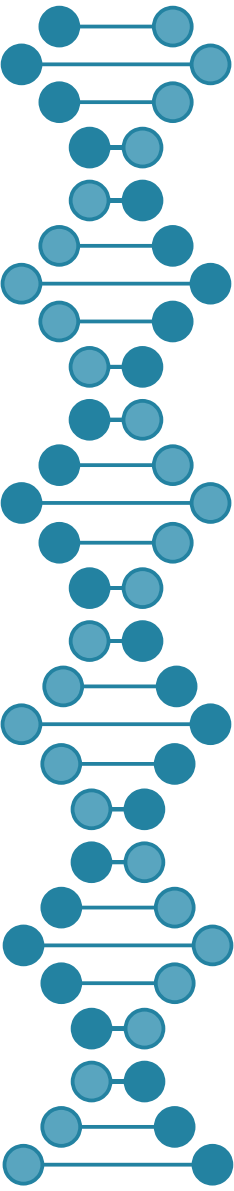


<https://en.wikipedia.org/wiki/NeXTcube>
(1990)



<https://en.wikipedia.org/wiki/NeXTSTEP>





Have you heard of any of these?

- Pixar
- Doom
- Quake
- The World Wide Web



30

AMMO

100%

HEALTH

2 3 4
5 6

ARMS

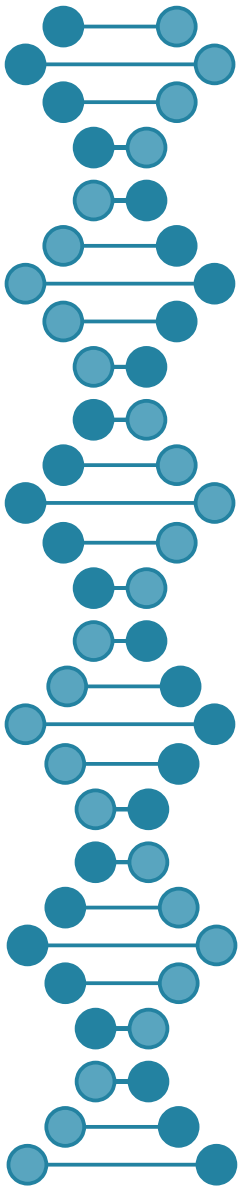


88%

ARMOR

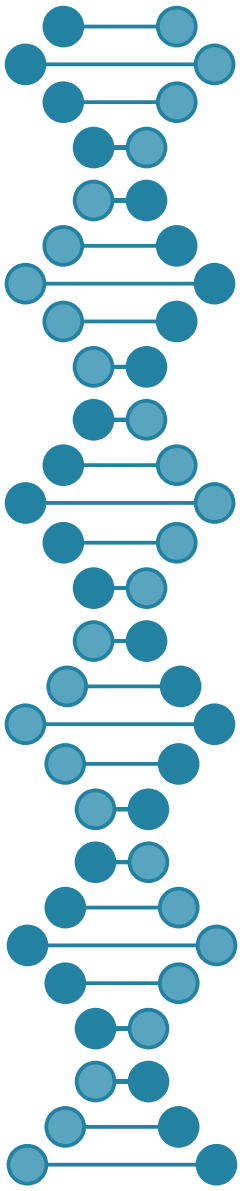
MONKEY

88 / 200
30 / 50
6 / 50
260 / 300



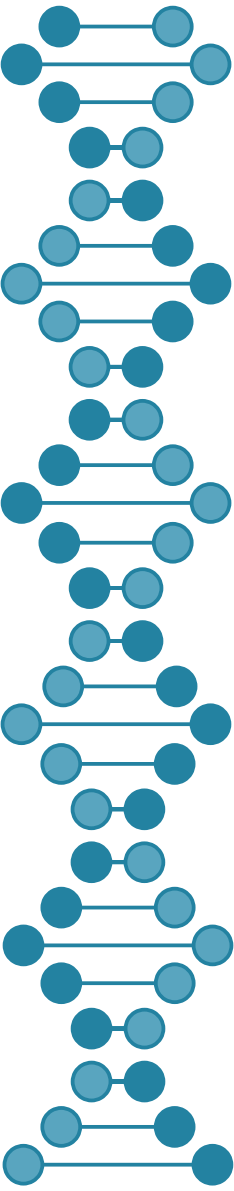
NextSTEP

- Combination of Mach and FreeBSD
- Objective-C
- An object oriented application layer known as “kits”
- The “Dock” in the GUI
- First app store



[https://en.wikipedia.org/wiki/Mach_\(kernel\)](https://en.wikipedia.org/wiki/Mach_(kernel)) (1985 to 1994)

- Microkernel replacement for UNIX
- Started as a monolithic kernel and evolved into a microkernel
- Basis for...
 - GNU Hurd
 - XNU
 - macOS, iOS, iPadOS, tvOS, and watchOS

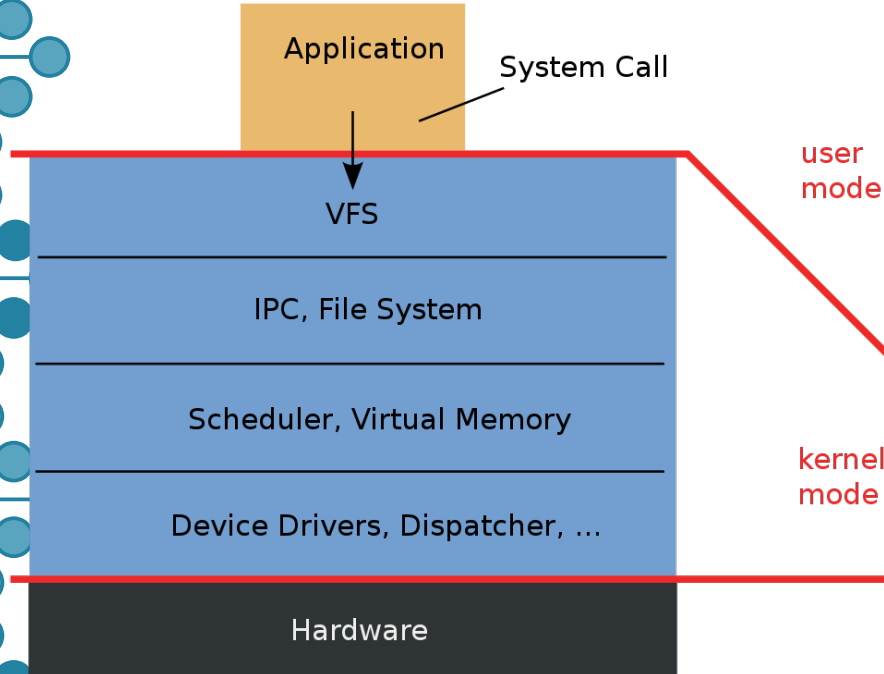


UNIX in the 1980s

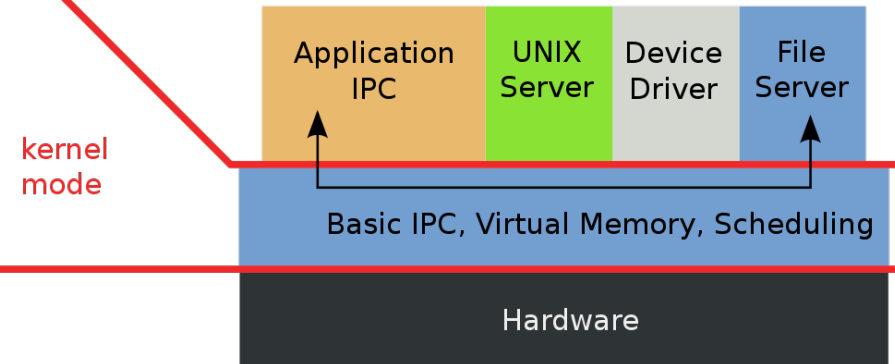
- Everything is all about pipes
- Networking, device drivers, *etc.*
 - A lot of complexity being added
- Aleph kernel at Univ. of Rochester ... OS is modular and communicates over pipes
 - Added shared memory
- Mach based on message passing

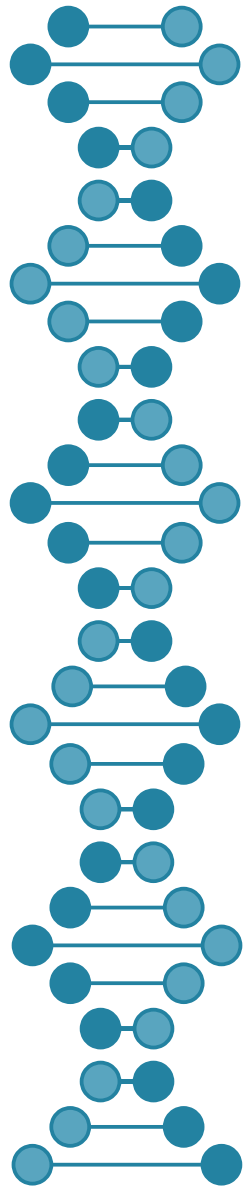
<https://en.wikipedia.org/wiki/Microkernel>

Monolithic Kernel based Operating System



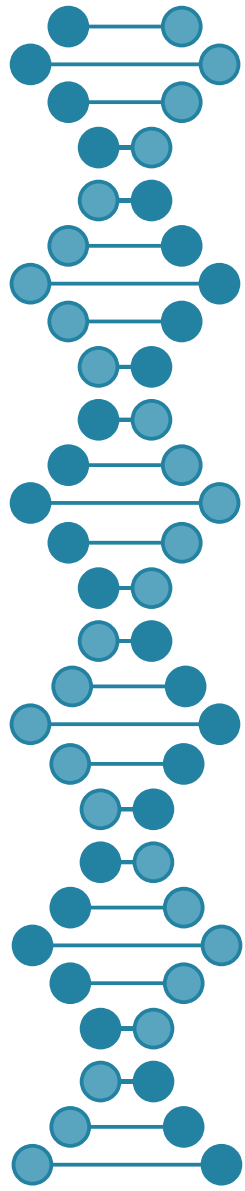
Microkernel based Operating System





Microkernels you may use without knowing

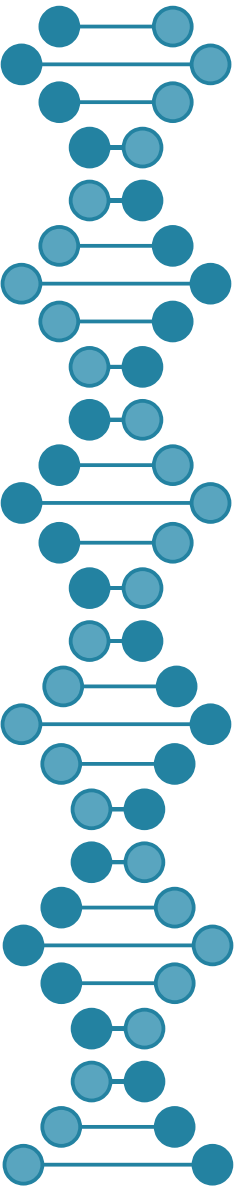
- QNX in your dashboard
- L4 in the modem in your smartphone
- seL4 in airplanes, drones, military hardware
- HarmonyOS, Nintendo Switch, ATMs, robots, trains
- Since 2008, Intel CPUs have a MINIX-based “Intel Management Engine” (Ring -3)



Microkernels you may use without knowing

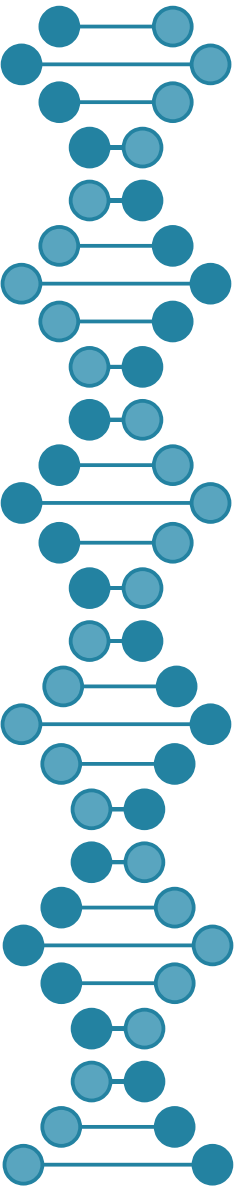
- QNX in your dashboard
- L4 in the modem in your smartphone
- seL4 in airplanes, drones, military hardware
- HarmonyOS, Nintendo Switch, ATMs, robots, trains
- Since 2008, Intel CPUs have a MINIX-based “Intel Management Engine” (Ring -3)

With a few exceptions (e.g., Flock cameras run an outdated version of Android) everything that is spying on you is using a microkernel.



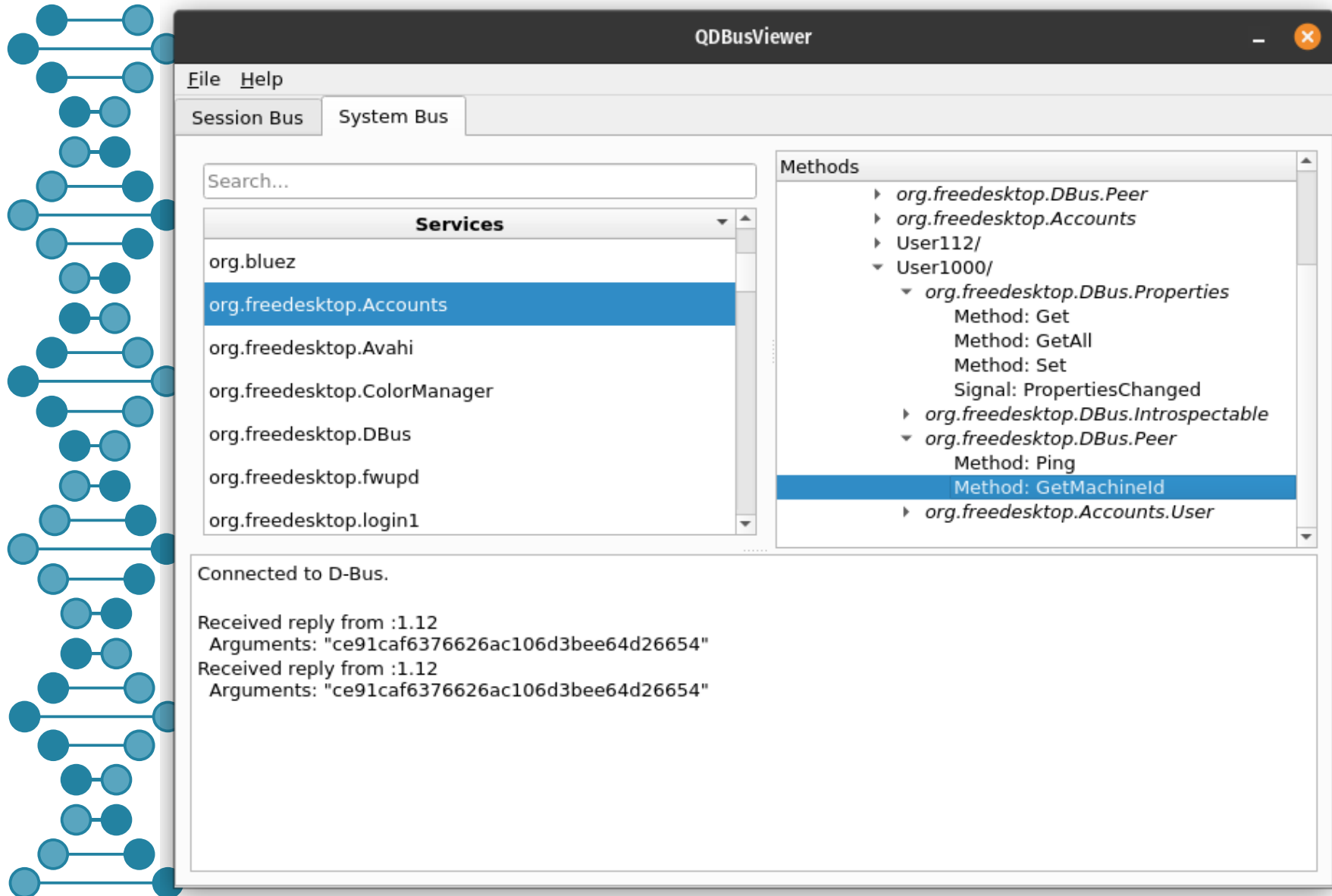
Mach ports

- In Mac, everything from threads to semaphores is a port
- Port rights (send, receive, ...) granted/mediated by kernel
- Marshal memory pointers between processes
 - CoW tricks
- XPC (like D-Bus for Mac) is built on Mach ports



Linux catching up to Apple: D-Bus

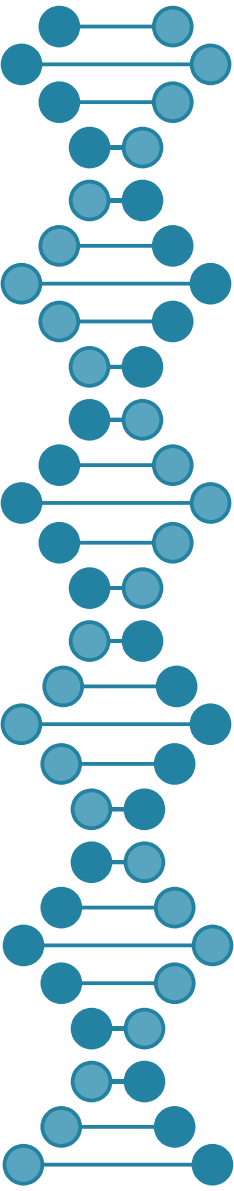
- Supports 1:1 request-reply for method calls and a 1:n publish-subscribe model for signals
- Used by systemd, NetworkManager, GUIs (Gnome and KDE)
 - *E.g.*, send a message to start SSH, change sys configs, *etc.*
 - Notify about a new network that is detected
 - Interact with Bluetooth, WPA supplicant, modems
 - Tell a window it got clicked on or dragged over
 - Keyrings
 - ...





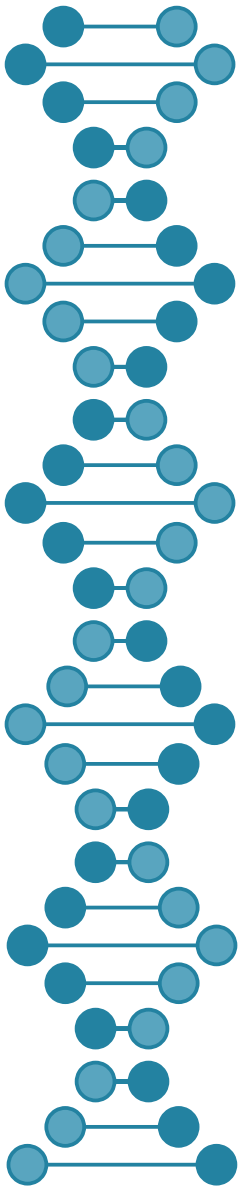


```
jedi@tortuga: ~  
  
        string "active-on-seats"  
        variant                                array [  
            string "seat0"  
        ]  
    )  
]  
)  
dict entry(  
    string "libreoffice-impress.desktop"  
    array [  
    ]  
)  
dict entry(  
    string "google-chrome.desktop"  
    array [  
    ]  
)  
dict entry(  
    string "libreoffice-calc.desktop"  
    array [  
    ]  
)  
]  
]
```



Message passing vs. message queues

- Can pass file descriptors or other forms of capabilities
 - In general, lots of types (errors, signals, security attributes, *etc.*), not just bytes+priority
- Service discovery and startup
- Large data over zero copy



Other kinds of zero copy IPC and IPC-like things
that are cool...

io_uring changes the contract

epoll tells you what is ready; io_uring lets you hand the kernel a batch of work.

epoll + nonblocking I/O

1 ask: which fds are ready?



2 wake up, then call read/write/accept



3 repeat: readiness is not completion

Good for readiness multiplexing.
Work still happens through read/write syscalls.

io_uring

SQE describe operation: read, write, accept, timeout...



enter submit a batch in one trip

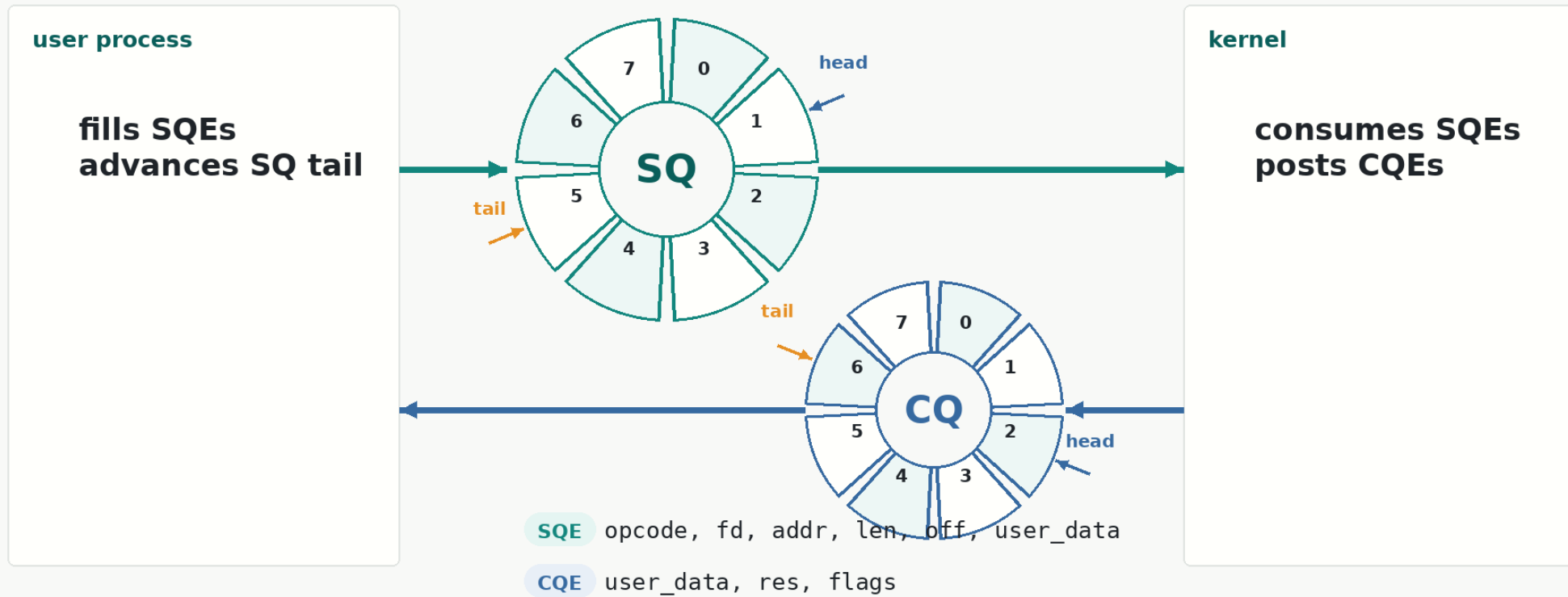


CQE kernel posts completions with result codes

Fast path: shared rings, batched completions, and sometimes SQPOL instead of a submit syscall.

Two shared rings, two directions

The queues are mmap'd into user space; the entries describe work and report results.



Memory barriers: publish before tail

For `io_uring`, the queue index is a signal. A release store makes the filled SQE visible before the tail moves.

without the ordering guarantee

CPU and compiler may reorder independent memory operations.



Kernel sees “work exists” before the descriptor is coherent.

That is not a data race you debug with vibes.

release / acquire handshake

```
man-pages + libuv

1 describe_io(sqe);
2 sqring->array[index] = index;
3 tail++;
4 atomic_store_explicit(sqring->tail,
5     tail, memory_order_release);
6
7 head = atomic_load_explicit(sqhead,
8     memory_order_acquire);
```

release: earlier stores visible before tail update

acquire: after new index, later loads see the SQE

→
not mutual exclusion; visibility ordering

Memory barrier, textbook version

A barrier is an ordering constraint: publish the payload first, then publish the flag.

without a barrier

Thread A: producer

1 data = 42;

2 ready = 1;

Thread B: consumer

3 if (ready) ...

4 r = data; // may still see 0

Bad outcome: flag visible before payload.

with release / acquire

Thread A: producer

1 data = 42;

2 store_release(ready, 1);

release publishes earlier writes

Thread B: consumer

3 if (load_acquire(ready))

4 r = data; // must see 42

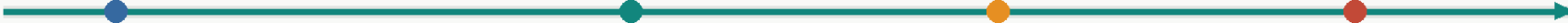
acquire imports that ordering

Barrier instructions across CPUs

Same C idea, different hardware: strong memory models need fewer explicit fences.

stronger ordering

weaker ordering

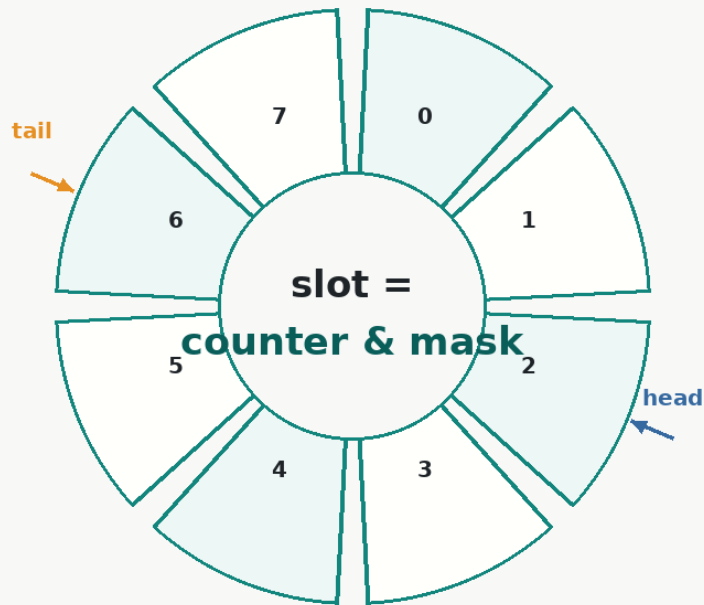


Architecture	Memory model	Representative instructions	Typical lesson
x86-64	TSO-ish: relatively strong	mfence, sfence, lfence lock cmpxchg / xadd	Acquire/release often compiles to plain loads/stores.
ARMv8-A AArch64	weak / relaxed	dmb ish, dsb, isb ldar, stlr	Explicit acquire/release ops are common and cheap-ish.
RISC-V	RVWMO: weak by default	fence rw, rw aq / rl on atomics	Ztso can strengthen ordering; portable code assumes RVWMO.
POWER	very weak	sync, lwsync, isync	Classic example where barriers really matter.

Barrier != semaphore: locked RMWs can act as heavy barriers, but their main job is atomic state change.
Acquire/release usually maps to the lightest sequence the CPU can safely provide.

Ring buffers: finite slots, unbounded counters

The modulo chooses a slot; the full head/tail counters preserve progress across wraparound.



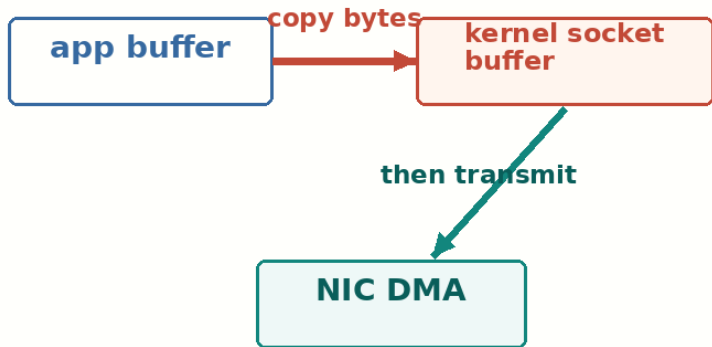
rules of the game

- Producer writes an entry, then advances tail.
- Consumer reads an entry, then advances head.
- Masking wraps counters onto fixed slots: $\text{index} = \text{counter} \& (N - 1)$.
- Submission order is not completion order; `user_data` connects CQEs back to requests.

Zero-copy send: where io_uring wins

Example: a proxy or CDN worker sending a large cached response over TCP.

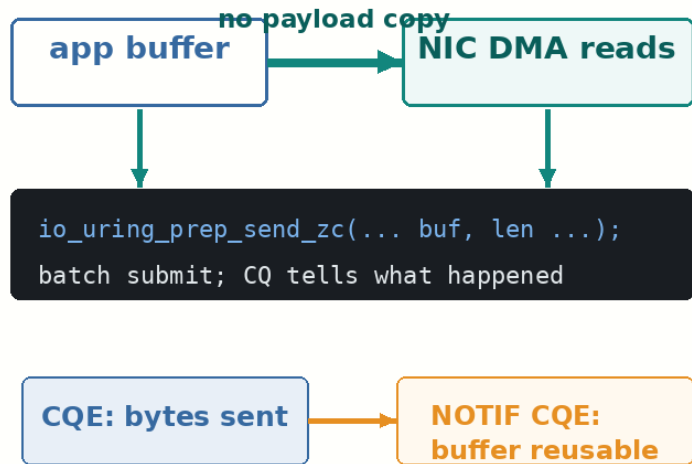
ordinary epoll + nonblocking send()



cost CPU copies payload + cache pollution

loop readiness wakeups, then send calls

io_uring + IORING_OP_SEND_ZC



win fewer syscalls, fewer copies, less cache traffic

Caveat: zero-copy is conditional; the kernel can fall back to copying.

Big wins show up when payloads are large, repeated, and already in user memory.

Real source sightings

Open-source projects using `io_uring` show the same concepts: SQEs, CQEs, release/acquire, and fallbacks.

QEMU block/io_uring.c

github.com/qemu/qemu

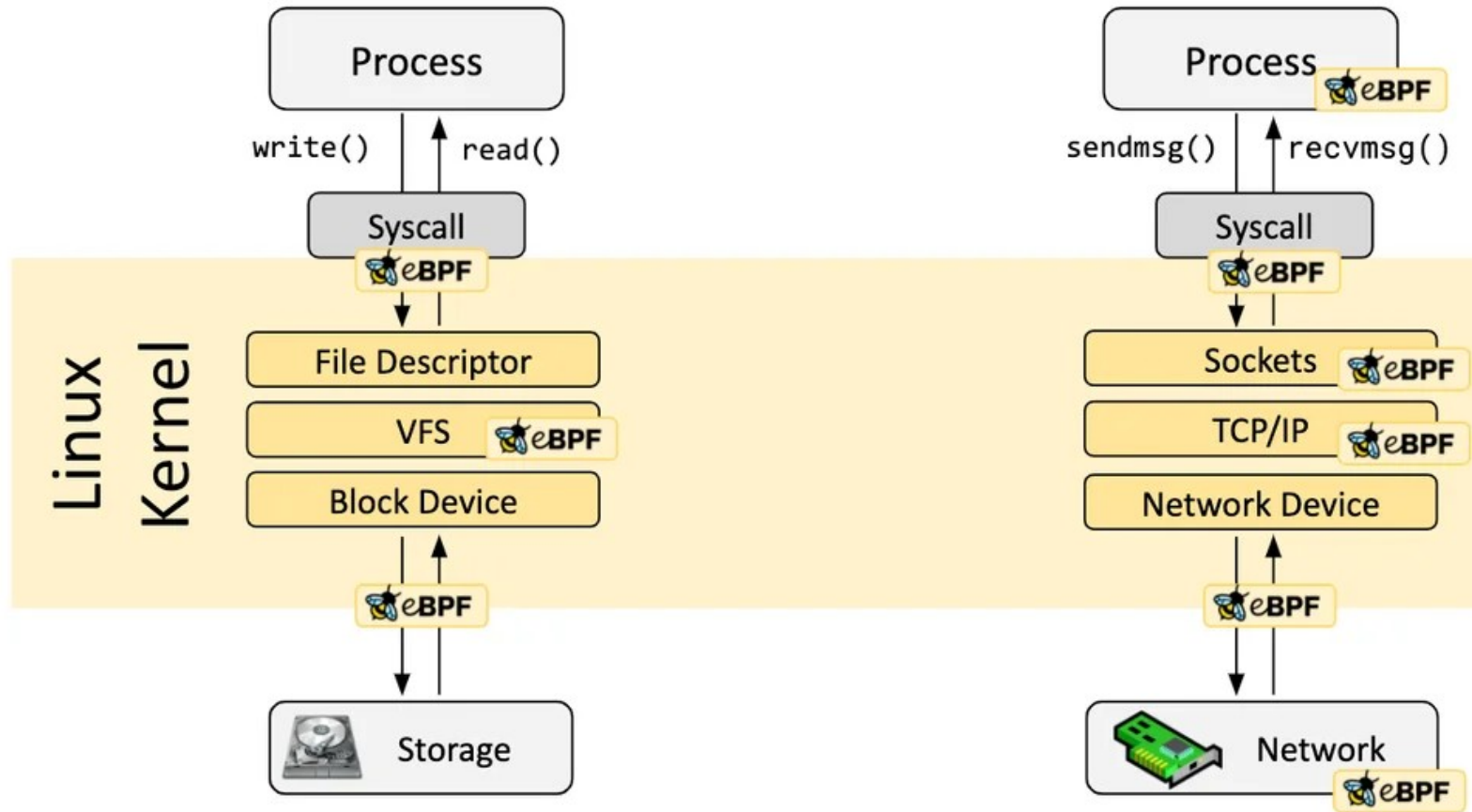
```
47 switch (req->type) {
48 case QEMU_AIO_WRITE:
52   io_uring_prep_writev2(sqe, fd, qiov->iiov,
53     qiov->niov, offset, luring_flags);
64   io_uring_prep_write(sqe, fd, iiov->iiov_base,
65     iiov->iiov_len, offset);
72 case QEMU_AIO_READ:
75   io_uring_prep_readv(sqe, fd, qiov->iiov,
76     qiov->niov, offset);
80   io_uring_prep_read(sqe, fd, iiov->iiov_base,
81     iiov->iiov_len, offset);
86 case QEMU_AIO_FLUSH:
87   io_uring_prep_fsync(sqe, fd, IORING_FSYNC_DATASYNC);
```

libuv src/unix/linux.c

github.com/libuv/libuv

```
3524 head = atomic_load_explicit((_Atomic uint32_t*) iou->sqhead,
3526   memory_order_acquire);
3528 tail = *iou->sqtail;
3532 if ((head & mask) == ((tail + 1) & mask))
3534   return NULL; /* No room in ring buffer. */
3536 slot = tail & mask;
3540 sqe = &sqe[slot];
3542 memset(sqe, 0, sizeof(*sqe));
3543 sqe->user_data = (uintptr_t) req;
3567 atomic_store_explicit((_Atomic uint32_t*) iou->sqtail,
3569   *iou->sqtail + 1,
3571   memory_order_release);
3578 uv__io_uring_enter(iou->ringfd, 0, 0, UV__IORING_ENTER_SQ_WAKEUP);
```

<https://ebpf.io/what-is-ebpf/>



Precursors/alternatives to eBPF

- BPF (Berkeley Packet Filter, before it was extended)
- DTrace
 - Originally for Solaris, now FreeBSD, Windows, netBSD, macOS
- Linux alternatives
 - LTTNg, perf, ftrace, SystemTap
- ETW in Windows
- ProbeVue in AIX



This work is licensed under a Creative Commons
Attribution-ShareAlike 3.0 Unported License.
It makes use of the works of Mateus Machado Luna.

