UNIX ARCHAEOLOGY

A browser from 1996 Running in 2026

Resurrecting Red Baron to run on modern FreeBSD



Ken Wong / MUUG Nov 2026

A browser from 1996, running in 2026

Reviving Red Baron to run on FreeBSD

- Red Baron is a web browser from the Linux 2.0 era.
- The original program survives, but its world has changed.
- Can a modern FreeBSD system make it run again?

1996
→ 2026

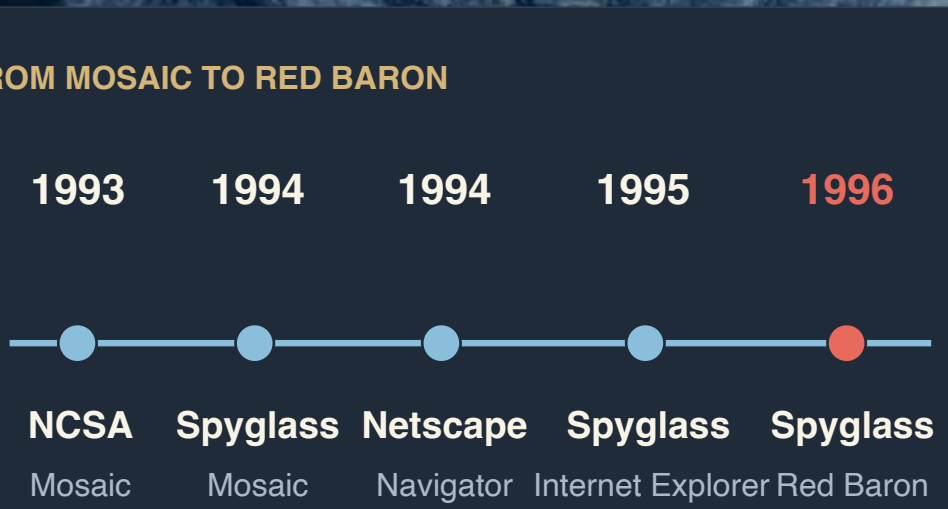


ORIGINAL i386 BINARY

The web before the browser wars

From an academic project to the race for the desktop

- NCSA Mosaic made the graphical web approachable in 1993.
- Spyglass licensed Mosaic technology to 120+ companies, including: MS, RedHat, IBM
- Netscape and Microsoft made browsers mainstream.
- Red Hat shipped a browser with its desktop.

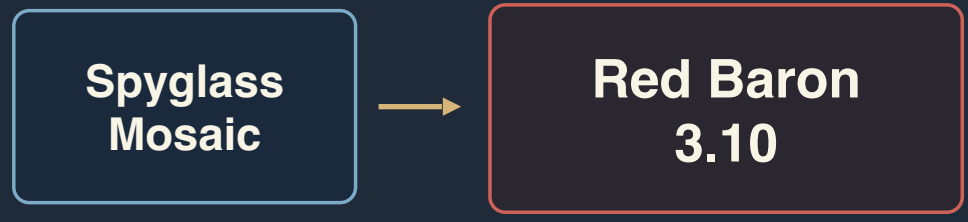


Meet Red Baron

A proprietary browser distributed with Red Hat Linux

- Red Hat Linux 4.0 introduced it in October 1996.
- It was derived from Spyglass Mosaic technology.
- Red Hat Linux 4.2 (May 1997) was its last release.
- The copy we are running is Red Baron version 3.10.

A RED HAT VALUE-ADD

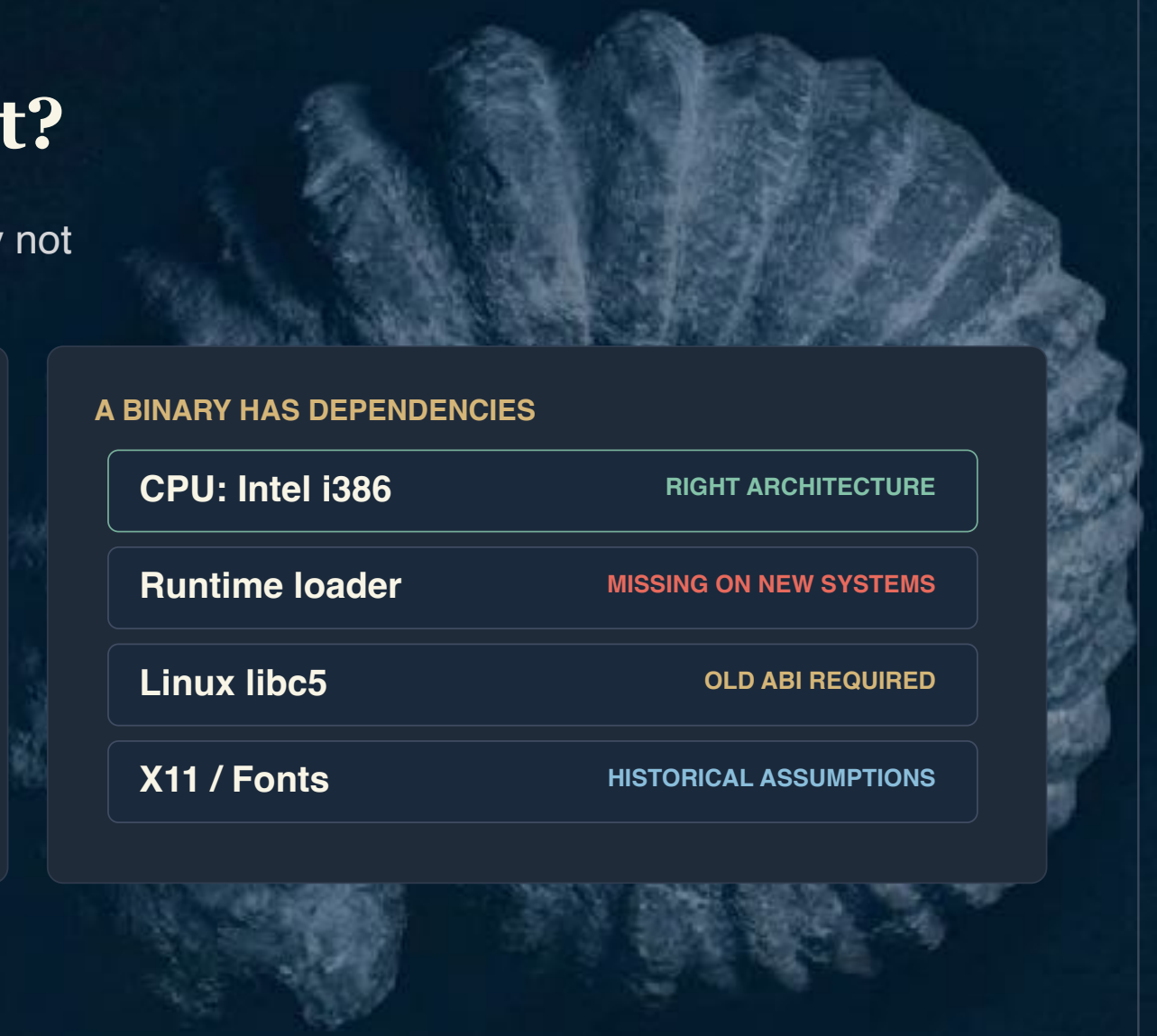


Red Hat's own history describes it as buggy and proprietary.

Why not just double-click it?

The CPU can understand it; the rest of the system may not

- Red Baron is a 32-bit Intel Linux executable.
- It needs a 1990s loader and Linux libc5.
- It relies on historical X11 libraries and fonts.
- New systems provide different interfaces.



A BINARY HAS DEPENDENCIES

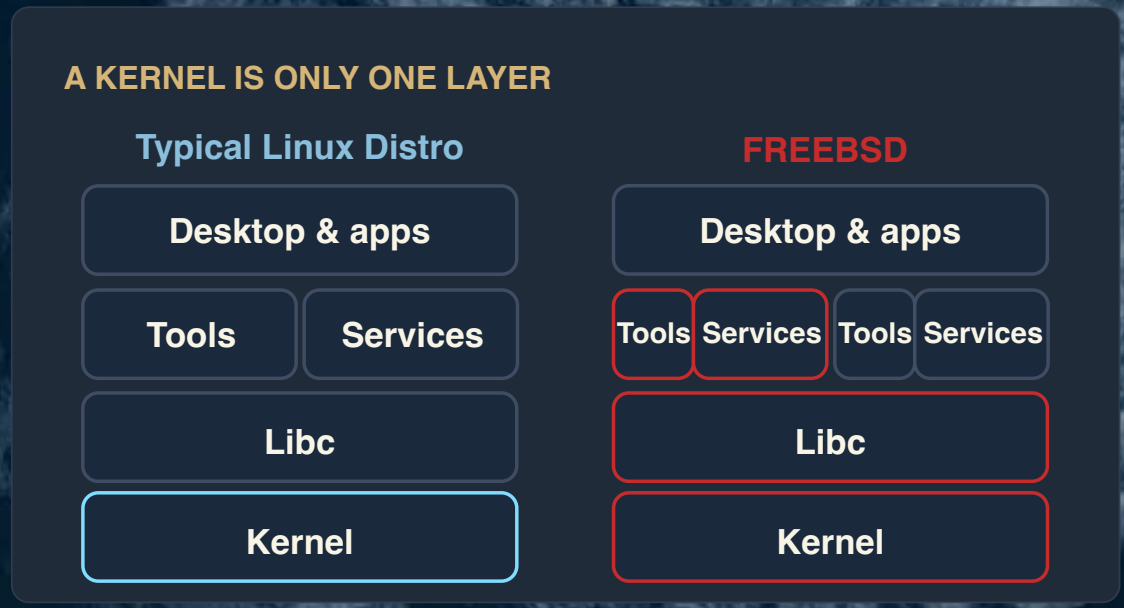
CPU: Intel i386	RIGHT ARCHITECTURE
Runtime loader	MISSING ON NEW SYSTEMS
Linux libc5	OLD ABI REQUIRED
X11 / Fonts	HISTORICAL ASSUMPTIONS

The required ELF interpreter is /lib/ld-linux.so.1.

Linux is a kernel; FreeBSD is a system

An operating system needs much more than its kernel

- Linux is the kernel: processes, memory, devices, and system calls.
- A Linux distribution adds libc, tools, services, and a desktop.
- FreeBSD develops its kernel and base userland together.

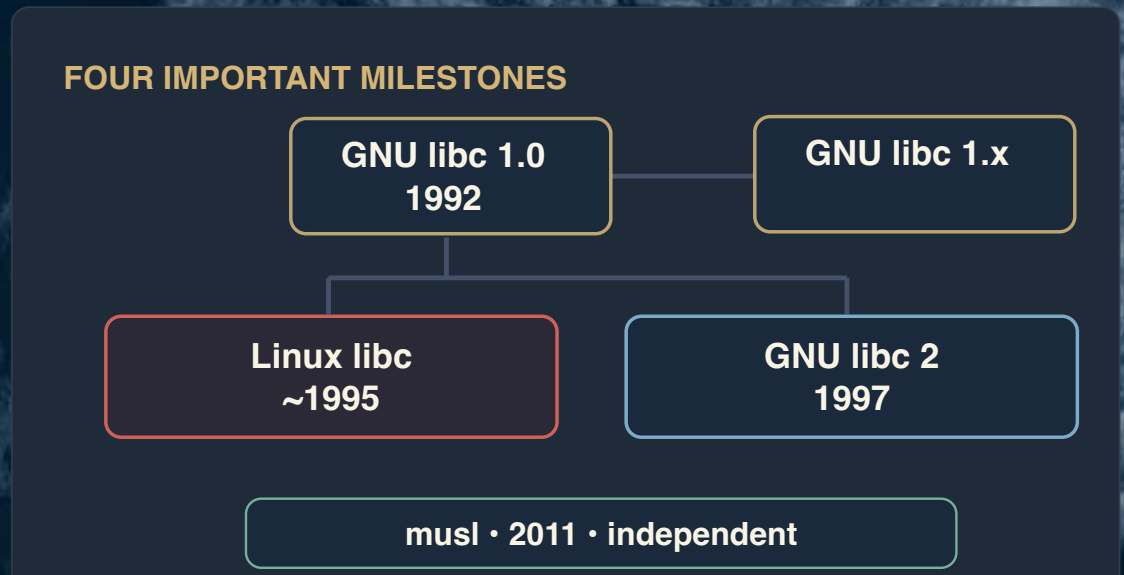


Neither operating system bundles every desktop application or third-party library.

Linux libc family tree

The C library is part of a program’s binary contract

- GNU libc 1.0 appeared in 1992.
- Linux libc powered early ELF Linux systems.
- GNU libc 2.0 released in 1997.
- musl arrived in 2011 as a lean alternative.

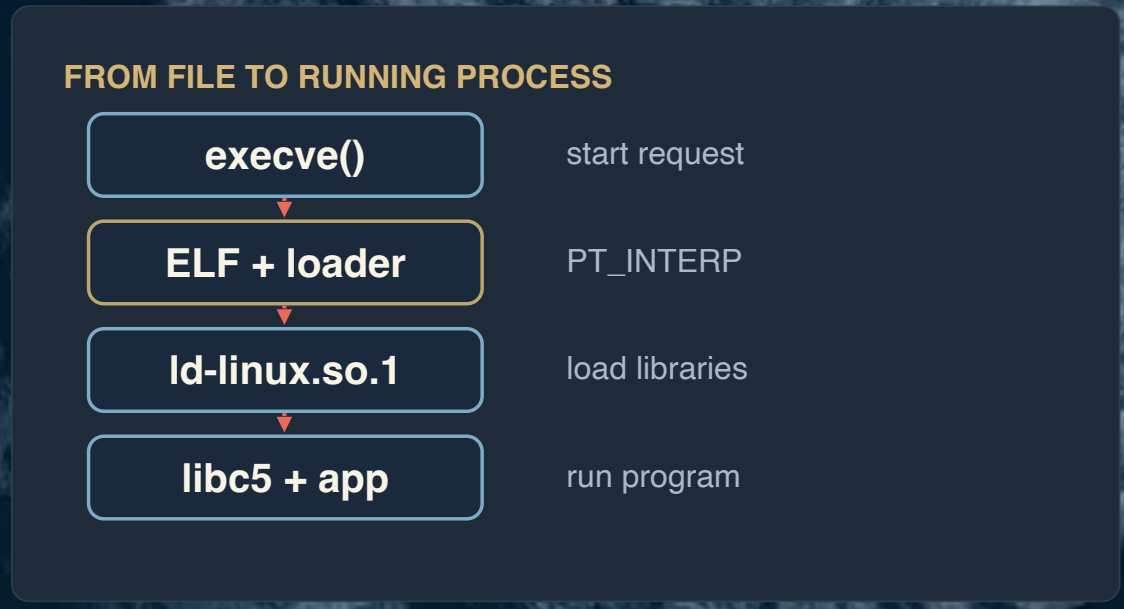


libc.so.5 and libc.so.6 are different binary interfaces, despite similar C APIs.

What happens when a program starts?

Follow a Linux ELF executable into running code

- The shell asks the kernel to execute a program.
- The ELF file names its runtime loader.
- The loader finds the shared libraries it needs.
- Finally, the program calls library functions and system calls.

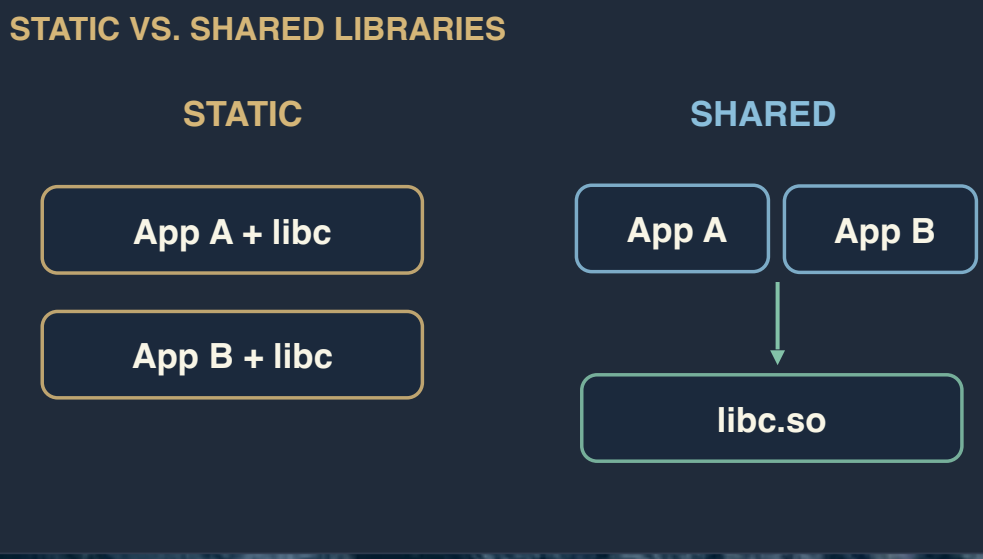


execve() → ELF PT_INTERP → ld-linux.so.1 → libc.so.5 → program entry.

Shared libraries changed Unix

Reuse code instead of embedding a copy in every program

- Static linking copies library code into each executable.
- Shared libraries let many programs use common code.
- SunOS helped establish Unix runtime dynamic linking in the 1980s.
- The runtime linker made new kinds of compatibility tricks possible.

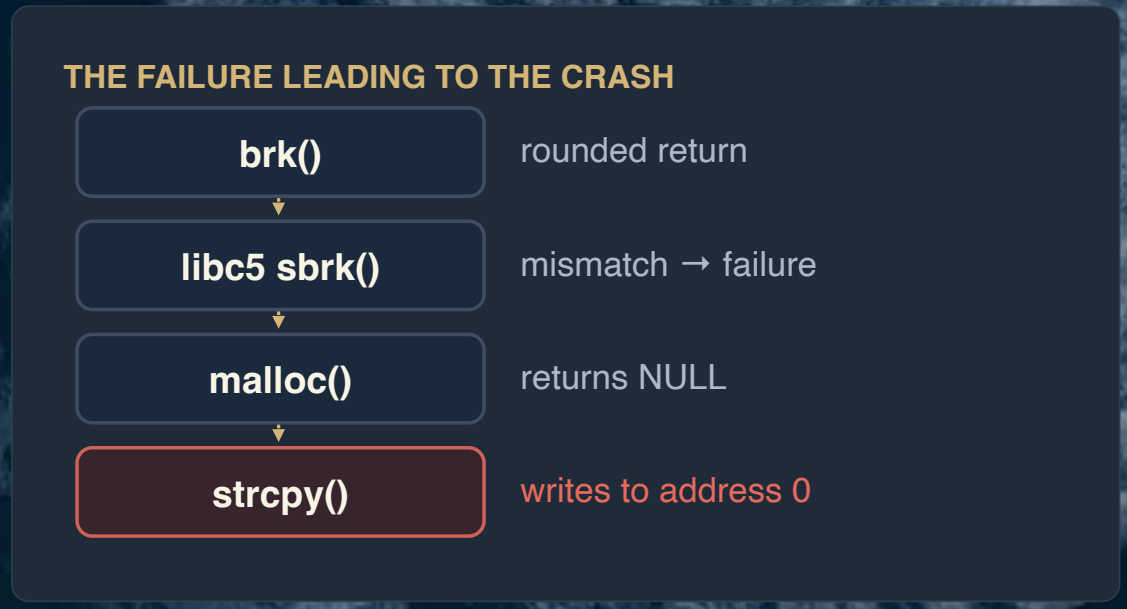


SunOS 4.0 (1988) included ld.so; LD_PRELOAD is present in its historical loader.

The crash: one small mismatch

A failing allocator hid behind a segmentation fault

- Unmodified Red Baron crashed before opening a window.
- Linuxulator rounded the break reported to libc5.
- libc5 sbrk rejected it; malloc returned NULL.
- Red Baron passed NULL to strcpy() and crashed.

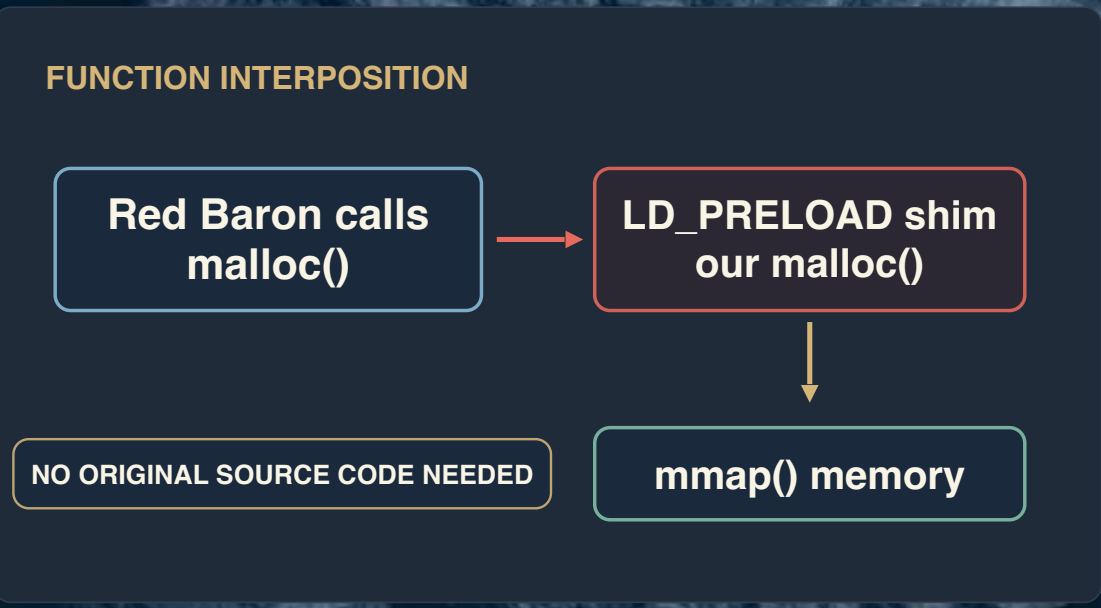


Trace, core dump, libc5 disassembly, and a syscall probe established the cause.

LD_PRELOAD: a clever detour

Substitute a function without rebuilding an old application

- A program asks a shared library for a function.
- LD_PRELOAD loads our replacement library first.
- Our version of malloc() can run instead of the old one.



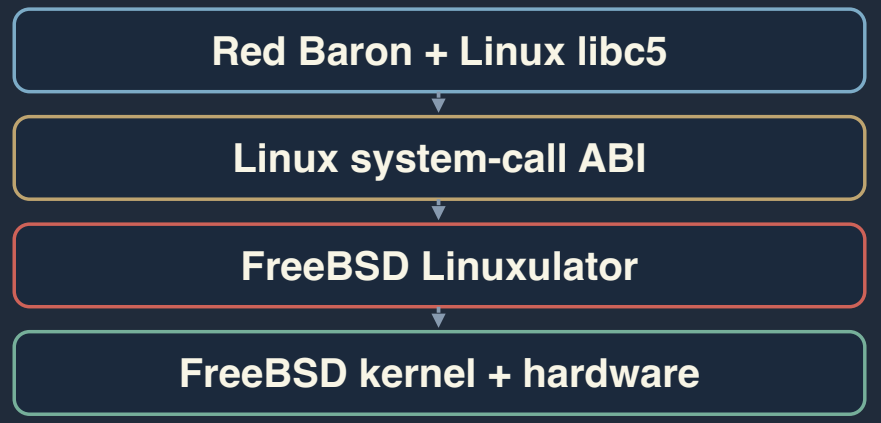
Symbol interposition works with dynamically linked symbols; it is not a universal patch.

API vs ABI: why old binaries care

Source compatibility is not the same as binary compatibility

- API: Can I write and compile against this interface?
- ABI: Can I run an already-compiled program against it?
- Red Baron needs the old libc5 and Linux syscall ABIs.
- Linuxulator reproduces the Linux ABI on FreeBSD.

HOW LINUXULATOR SATISFIES THE ABI

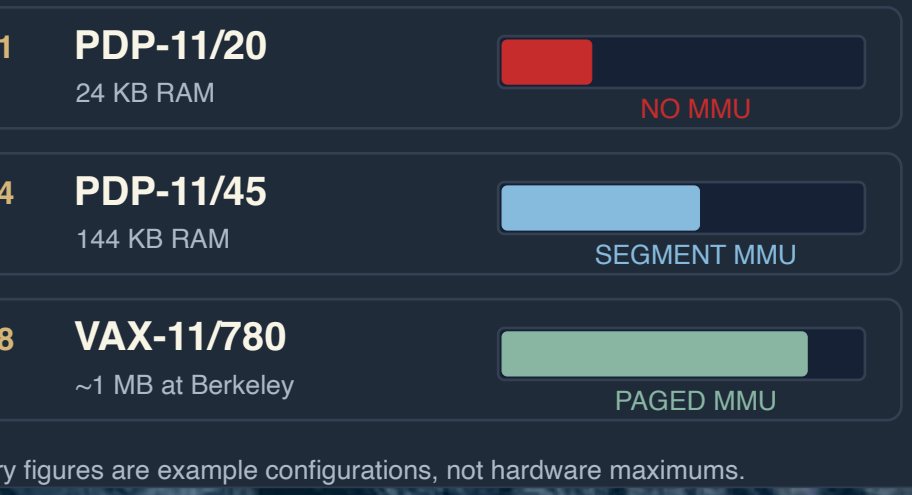


The CPU runs Red Baron directly; no second Linux kernel is booted.

The hardware that invented Unix

From kilobytes of RAM to a machine with paged virtual memory

- First PDP-11 Unix (1971) had just 24 KB of RAM.
- A PDP-11/45 (1974) Unix installation had 144 KB of RAM.
- Berkeley's first VAX-11/780 (1978) had about 1 MB of RAM.
- Memory hardware changed how Unix could manage processes.

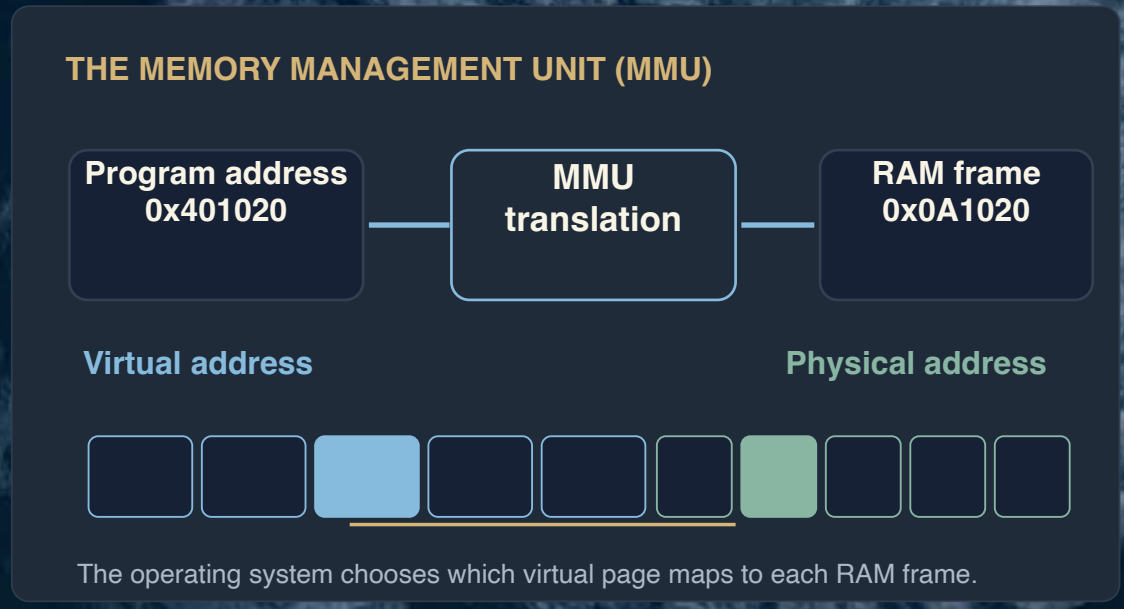


PDP-11/20 lacked a protected MMU; the VAX supported 512-byte virtual pages.

What is an MMU?

Hardware that maps virtual to physical memory and enforces protection.

- The MMU helps protect one process from another.
- A program uses virtual addresses; the MMU translates them to RAM.
- Paged MMUs can map small chunks of a process independently.
- But the operating system must implement paging policy.



PDP-11/20 had no MMU; later PDP-11 systems mapped segments; VAX mapped pages.

Swapping vs. demand paging

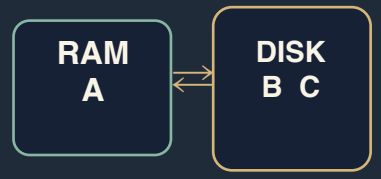
The single most important memory-management distinction

- Swapping moves an entire process between RAM and disk.
- It lets more processes exist than fit in memory at once.
- Demand paging transfers only the needed memory pages.
- One process can then address more memory than physical RAM.

TWO DIFFERENT IDEAS

SWAPPING

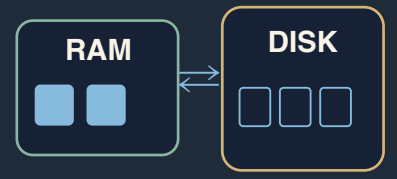
whole process



More processes can exist

DEMAND PAGING

individual pages



One process can exceed RAM

Existing processes are not all executing at once; the CPU schedules runnable work.

From VAX Unix to BSD paging

The hardware had paging in 1978; Unix software caught up

- Bell Labs ported Version 7 Unix to the VAX as 32V in 1978.
- 32V still swapped entire processes despite the VAX MMU.
- Berkeley added demand-paged virtual memory in 3BSD (1979–80).
- This changed what size of program Unix could run.

A PAGED MMU DID NOT AUTOMATICALLY MEAN PAGING

1978 — **Bell Labs 32V**
whole-process swapping

1979–80 — **Berkeley 3BSD**
demand-paged virtual memory

1983 — **4.2BSD**
matured BSD VM system

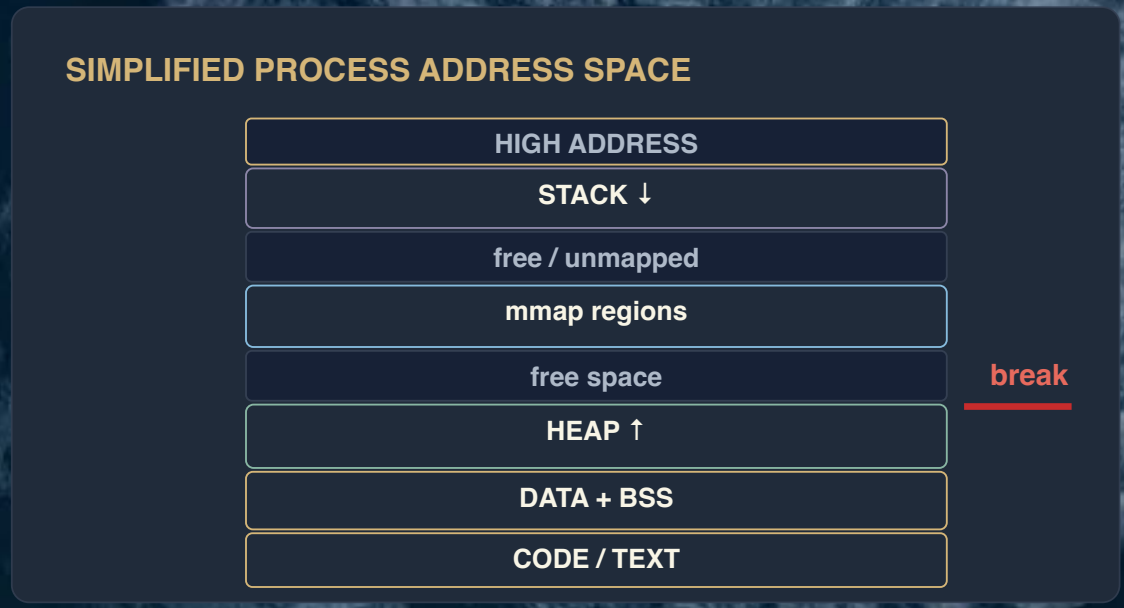
The VAX provided page mapping; BSD supplied demand-paging policy.

The VAX was DEC hardware; 32V and 3BSD were different Unix implementations.

Where does a process put its memory?

Code, data, heap, mapped regions, and stack

- The program starts with code and data loaded in its address space.
- The traditional heap grows upward as the program requests memory.
- The stack grows separately for function calls and local variables.
- Modern systems also create independent memory mappings.



The “program break” is the logical end of the process data/heap region.

brk is older than modern Unix memory

Its ancestor was in the very first Unix manual

- 1971: First Edition Unix exposed the break system call.
- It set the highest address used by a program's data.
- 1973: Fourth Edition added sbrk for relative growth.
- 1975: Sixth Edition introduced the brk C interface.

50+ YEARS OF THE SAME IDEA

1971	1973	1975
First Edition	**Fourth Edition**	**Sixth Edition**
break: syscall #17	sbrk: relative growth	brk: C-level interface

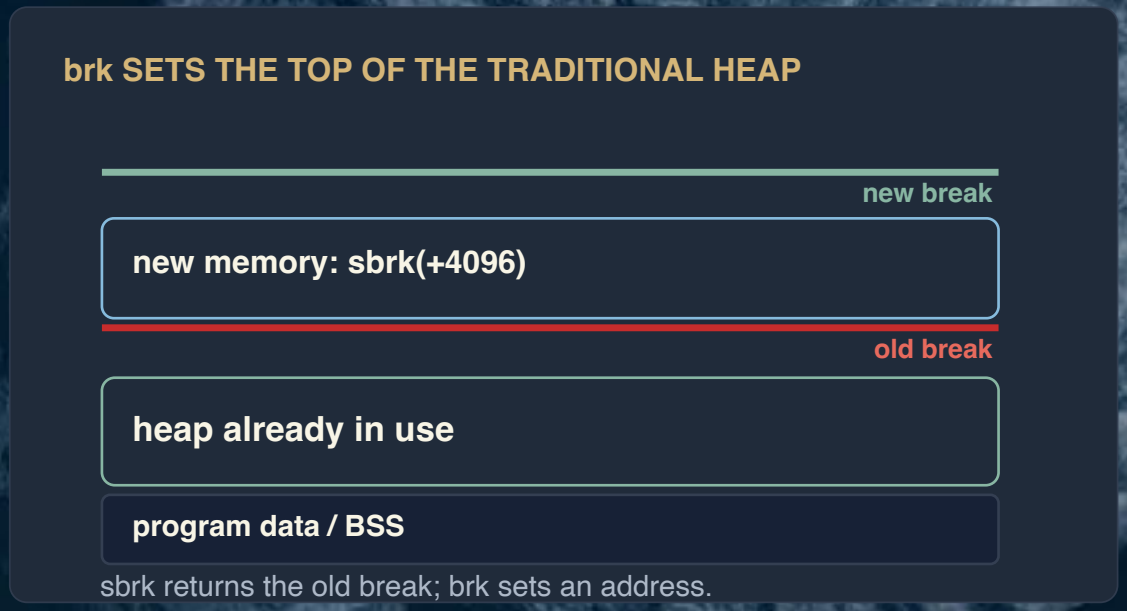
Ken Thompson & Dennis Ritchie built the earliest Unix interfaces at Bell Labs.

V1 break was syscall #17; the 1971 manual credits "ken, dmr."

What brk() and sbrk() actually do

A very small interface with a very long history

- brk(address) sets the logical end of the data segment.
- sbrk(amount) moves that end by a relative number of bytes.
- Old malloc() implementations used them to grow the heap.
- The kernel still manages physical memory in pages.



sbrk(0) reports the current break; a successful sbrk(n) returns the old break.

The break is not the page boundary

One byte-level value, one page-level allocation

- A program can request a break halfway through a page.
- The VM system still allocates and frees whole pages.
- The old Linuxulator returned a rounded break.
- It could reject a valid libc5 request for _end + 40.

ONE EXTRA BYTE CAN STILL MAP A WHOLE PAGE

Requested logical break **0x10005A**



logical break (inside this page)

Page-rounded end: 0x101000

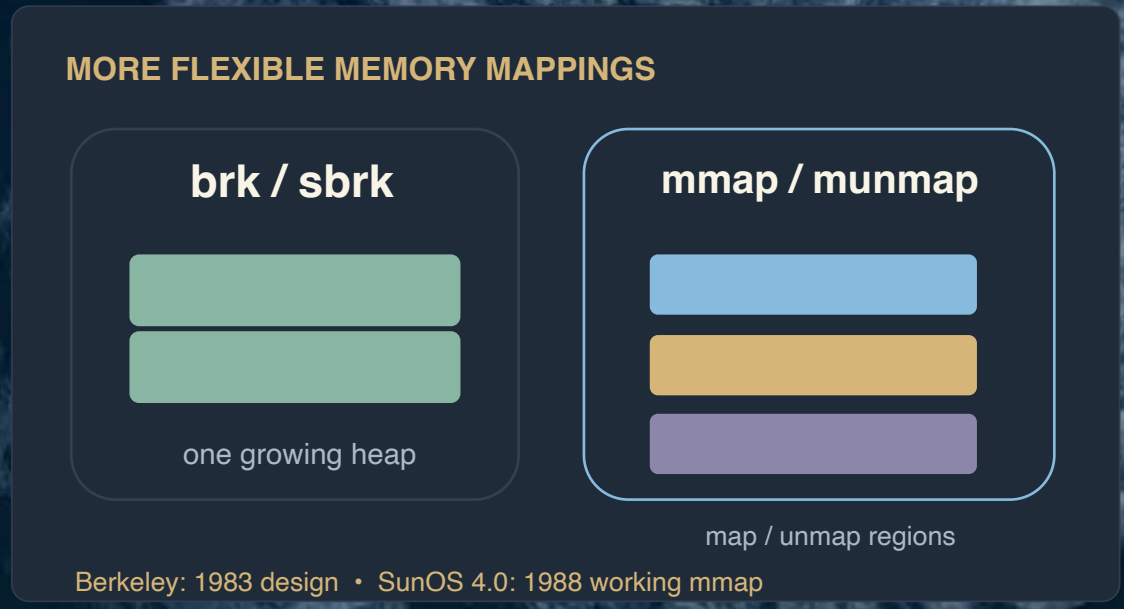
Mapped extent ≠ requested logical break.

A byte-exact logical break is not the same as a mapped page boundary.

What replaced direct brk allocation?

mmap and munmap provide flexible virtual-memory regions

- brk grows or shrinks one contiguous traditional heap.
- mmap maps a file or anonymous memory into an address space.
- munmap removes a mapping when it is no longer needed.
- Modern allocators may use mmap and sometimes brk.



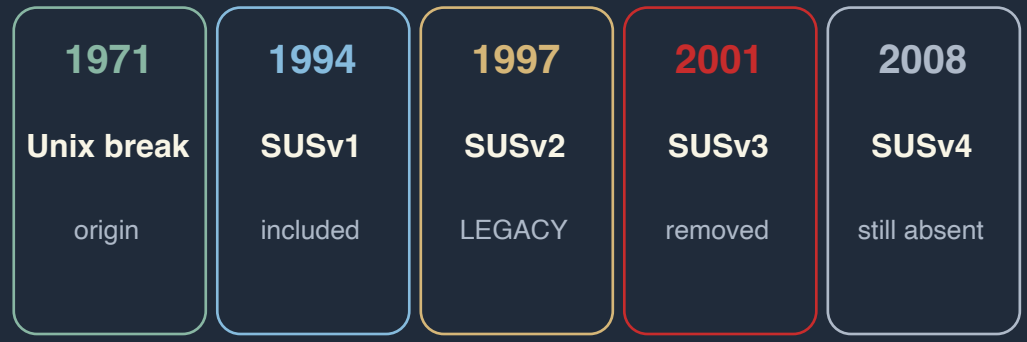
Berkeley CSRG designed mmap for 4.2BSD (1983), but SunOS 4.0 (1988) had the first working implementation.

brk entered SUS — then left

Standards followed Unix practice, then favored newer interfaces

- 1971: break predates POSIX and the Single UNIX Specification.
- 1994: SUSv1 included brk and sbrk.
- 1997: SUSv2 marked them LEGACY.
- 2001: POSIX.1 / SUSv3 removed them from the standard.

STANDARDIZED, THEN RECLASSIFIED



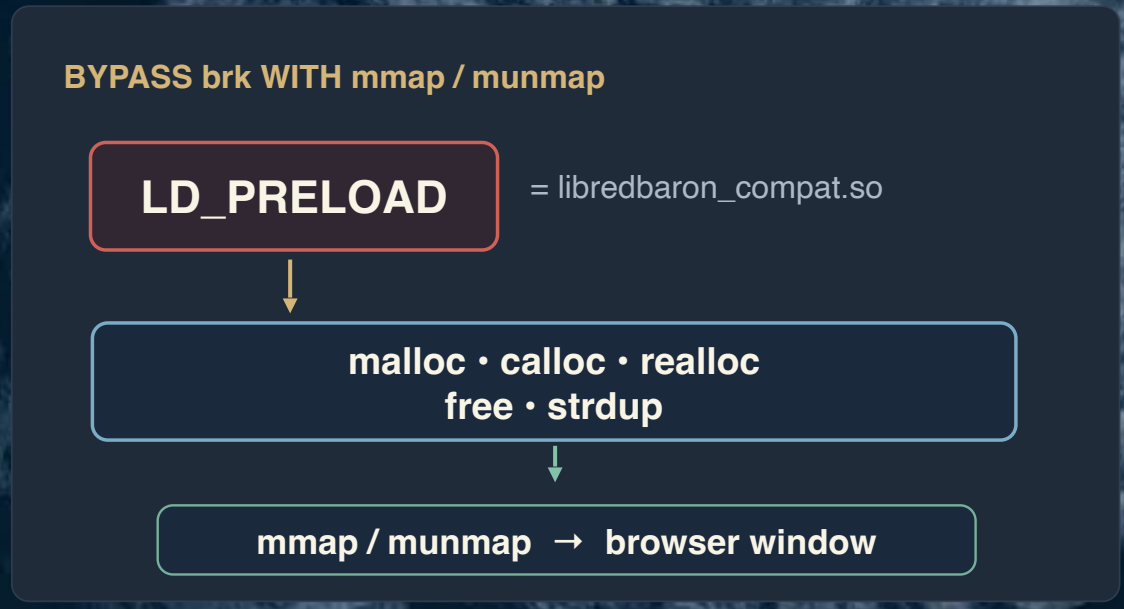
“Deprecated” does not mean a syscall disappears overnight.

SUSv4 / POSIX.1-2008 kept them out; Linux and some BSD architectures still support them.

Repair #1: the mmap malloc shim

An LD_PRELOAD workaround bypasses libc5's broken heap path

- LD_PRELOAD substitutes five allocation functions.
- Each allocation gets its own anonymous mmap() region.
- free() uses munmap(); calloc() receives zeroed memory.
- Red Baron starts without a FreeBSD kernel change.



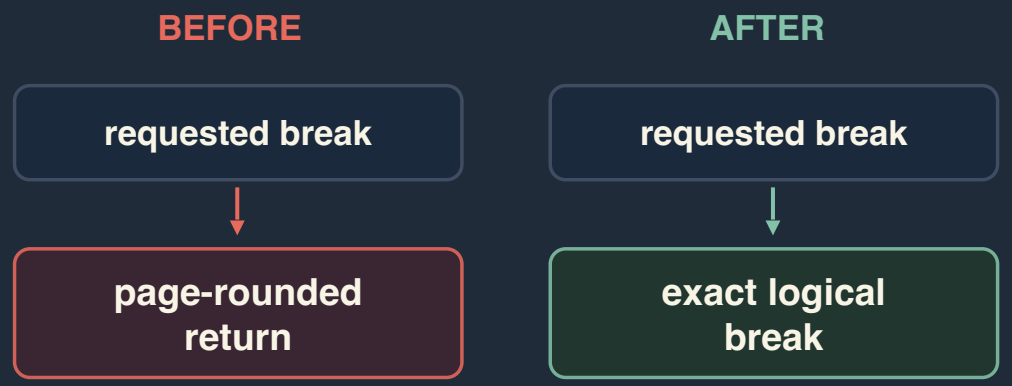
16-byte header; Linux/i386 mmap syscall #90 via int $0x80 bypasses libc5.

Repair #2: patch the kernel

Track the byte-exact break separately from page-rounded mappings

- The ELF loader records the exact data/BSS end in vm_brk_min.
- vm_brk tracks the byte-exact break; mappings remain paged.
- linux_brk returns it, or the old break on failure.
- Original libc5 malloc works without the preload shim.

FIX THE LINUXULATOR ABI, NOT THE APP



fork copies and exec resets break state; native FreeBSD behavior is preserved.

Live demo: welcome to the old web

What an HTML-era browser can do — and what it cannot

- Runs on FreeBSD!
- 1990s HTML: headings, tables, images, forms, frames, legacy SSL.
- No modern CSS, JS, TLS
- Support for image maps!

DEMO

MUUG Home Page

File Edit Navigate Windows Help

URL: http://muug.ca/

Home About Mailing Lists Meetings Newsletter FTP Locate File



MUUG Online



MUUG Online 8.0, is brought to you by the Manitoba UNIX User Group (MUUG), Manitoba's premier group for UNIX®, Linux, BSD, and macOS users, located in Winnipeg.

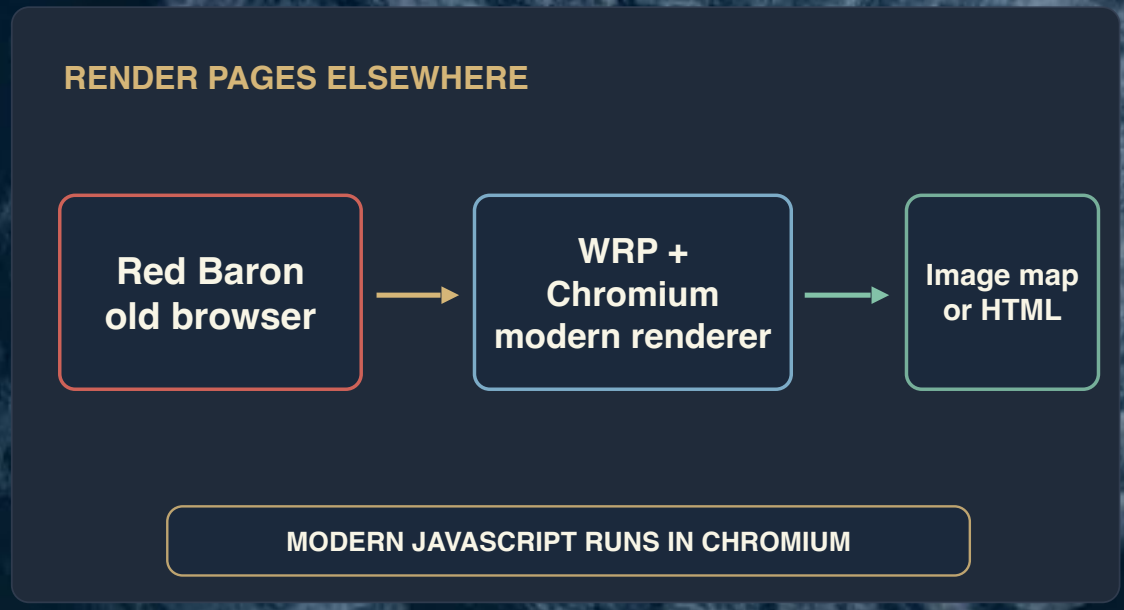
This month's Meeting: November 3, 7:30pm, Red Baron

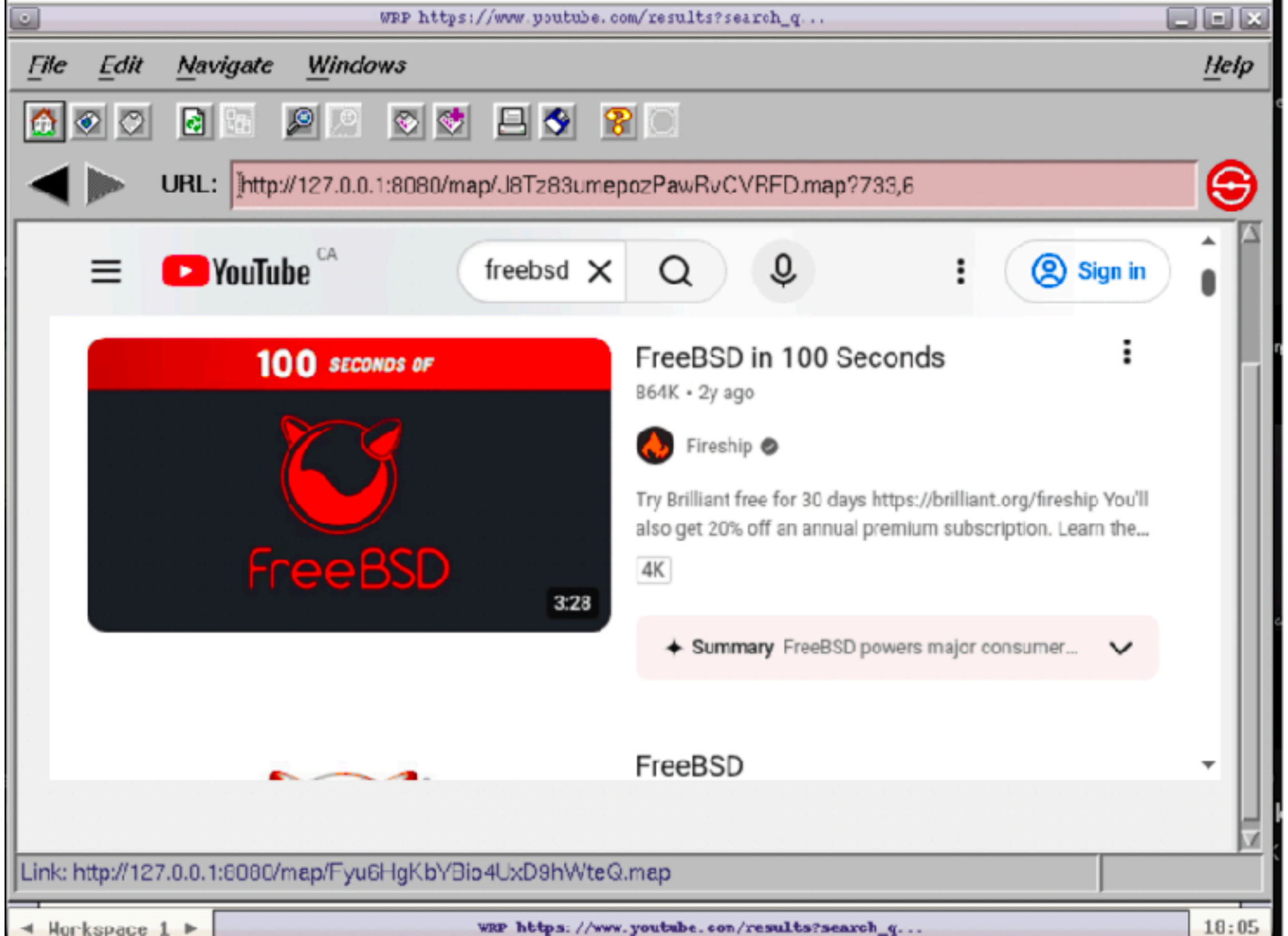
Workspace 1 xterm MUUG Home Page 18:07

WRP: a bridge to today's web

Let a modern browser render; let Red Baron display the result

- Web Rendering Proxy runs a modern Chromium engine.
- It transforms modern pages into images or simple HTML.
- Red Baron receives something its 1990s engine understands.
- Clickable image maps even preserve basic navigation.




WRP https://www.youtube.com/results?search_q...
File Edit Navigate Windows Help
URL: http://127.0.0.1:8080/map/J8Tz83umepozPawRvCVRFD.map?733,6
YouTube CA
freebsd
Sign in
100 SECONDS OF
FreeBSD
3:28
FreeBSD in 100 Seconds
864K • 2y ago
Fireship
Try Brilliant free for 30 days https://brilliant.org/fireship You'll also get 20% off an annual premium subscription. Learn the...
4K
Summary FreeBSD powers major consumer...
FreeBSD
Link: http://127.0.0.1:8080/map/Fyu6HgKbYBib4UxD9hWteQ.map
Workspace 1
WRP https://www.youtube.com/results?search_q...
10:05

Modern Linux

brk fails, different reason

- CONFIG_COMPAT_BRK=y
- ld.so.cache rebuilt in old format
- On Arch, the old loader required a legacy memory layout (search i386 -L).
- LD_PRELOAD

OLD USERLAND, RUNNING ON ARCH

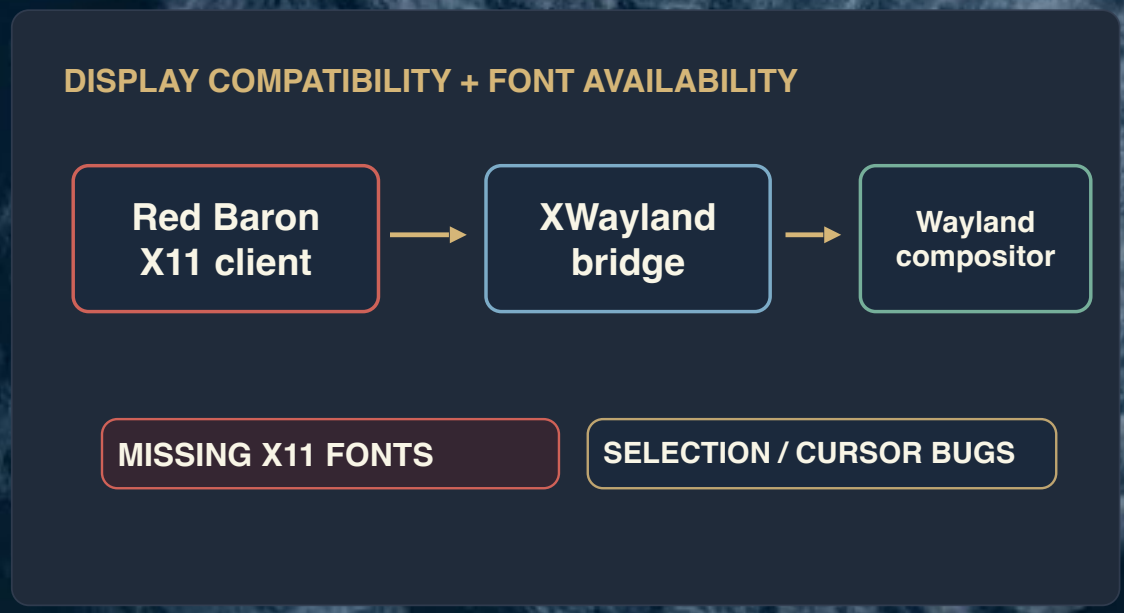
/lib/ld-linux.so.1	interpreter path
libc.so.5	historical C library
ld.so.cache	old cache format
setarch i386 -L	legacy layout

Example: patch PT_INTERP, supply old cache, use setarch i386 -L.

XWayland needs more than a display bridge

FreeBSD had X11 fonts; modern Linux needed extra setup

- Red Baron uses old X11 font mechanisms, not modern Fontconfig alone.
- Our FreeBSD install already had the required fonts.
- Modern Linux/XWayland needed those fonts installed and exposed.
- Text selection and dialog cursor bugs remained separate issues.



Selection crash and invisible cursor were observed; their exact causes remain unproven.

What reviving Red Baron teaches us

Software preservation is really about preserving interfaces

- A compatible CPU does not guarantee a working program.
- The loader, libc, syscalls, and display system all matter.
- We can work around a bug — or repair the ABI itself.
- FreeBSD makes an unexpected home for a Linux fossil.

