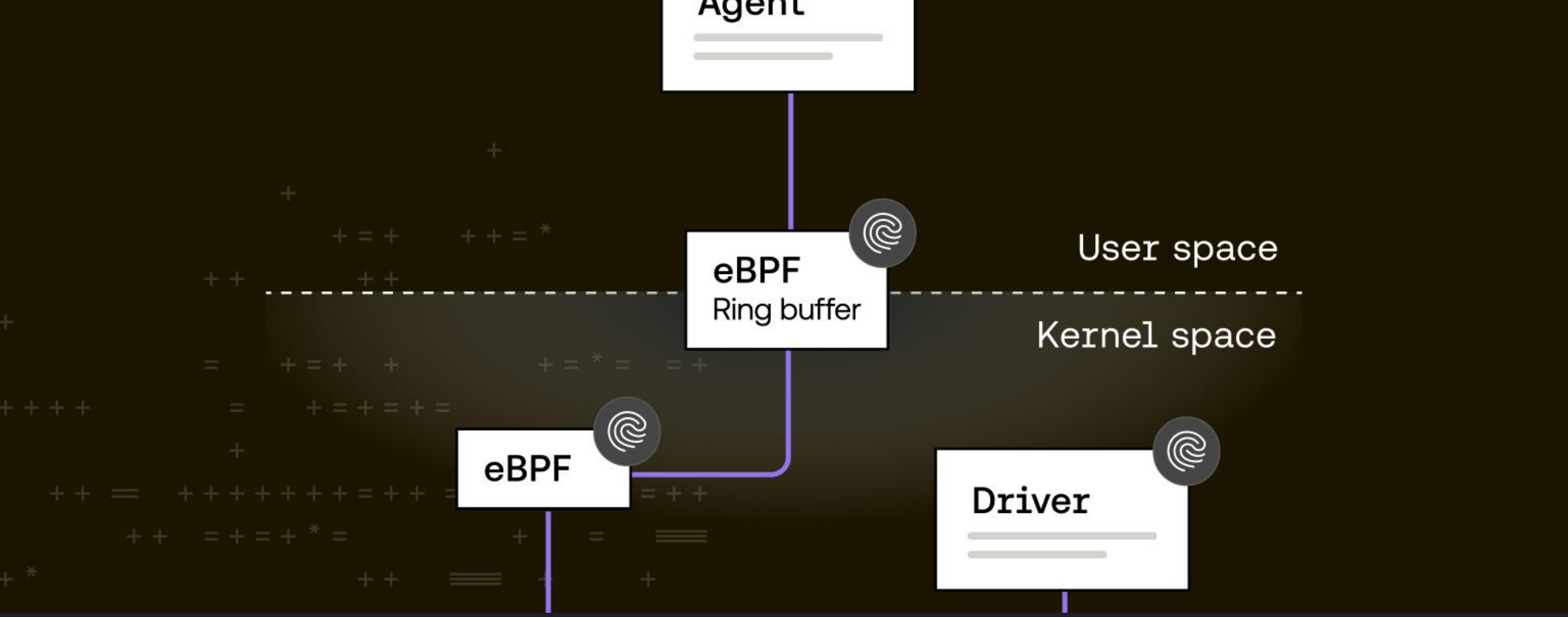
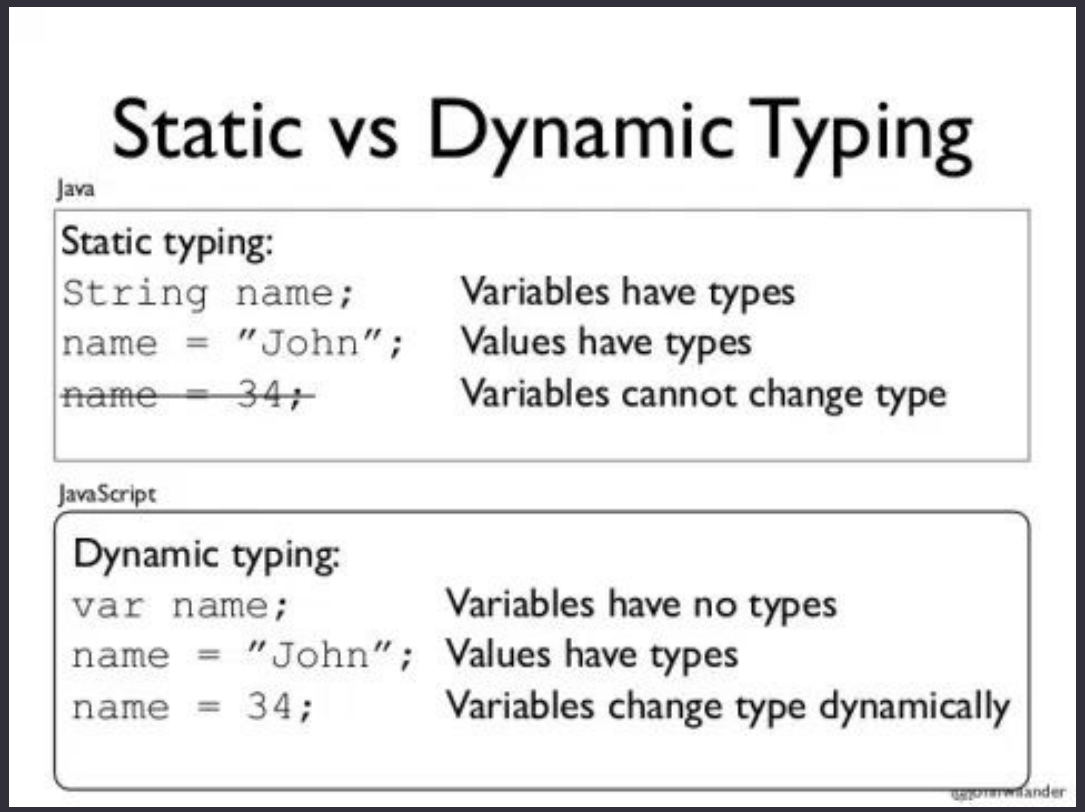


eBPF: Unlocking the Kernel

A Programmable Revolution for Linux

The Concept



Understanding eBPF through analogy:

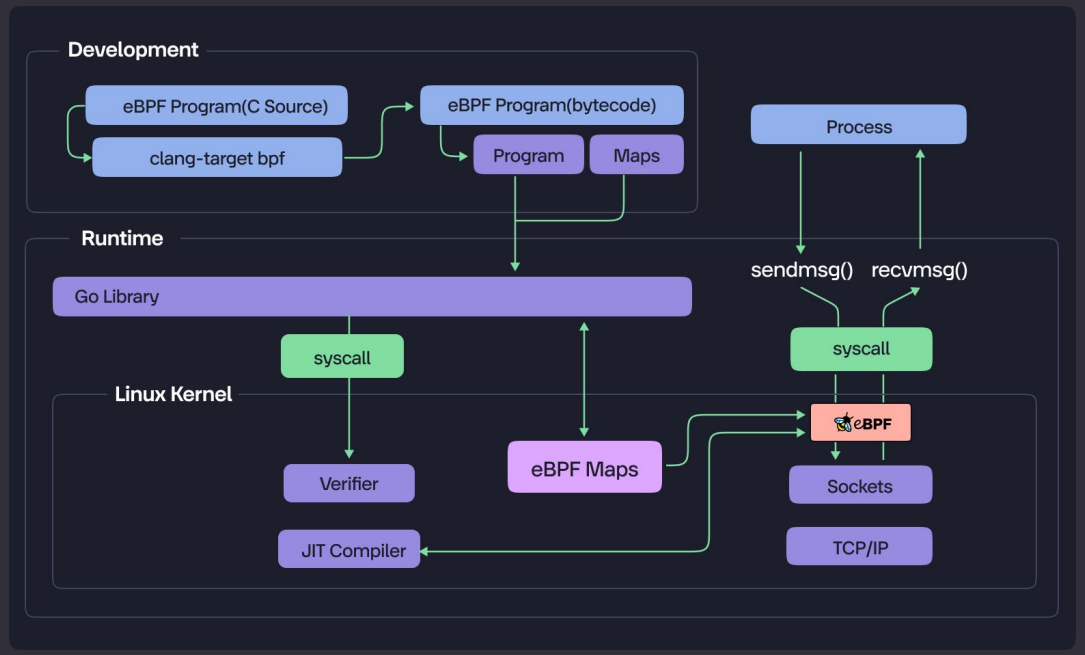
- Old World: Static Kernel, hard to change
- New World: Programmable, event-driven kernel
- Add capabilities at runtime without rebooting

The Superpowers of eBPF

eBPF enables powerful kernel functionalities:

- Observability: See system calls and network packets
- Security: Detect and block threats in real-time
- Networking: Programmable packet processing and load balancing

The eBPF Workflow



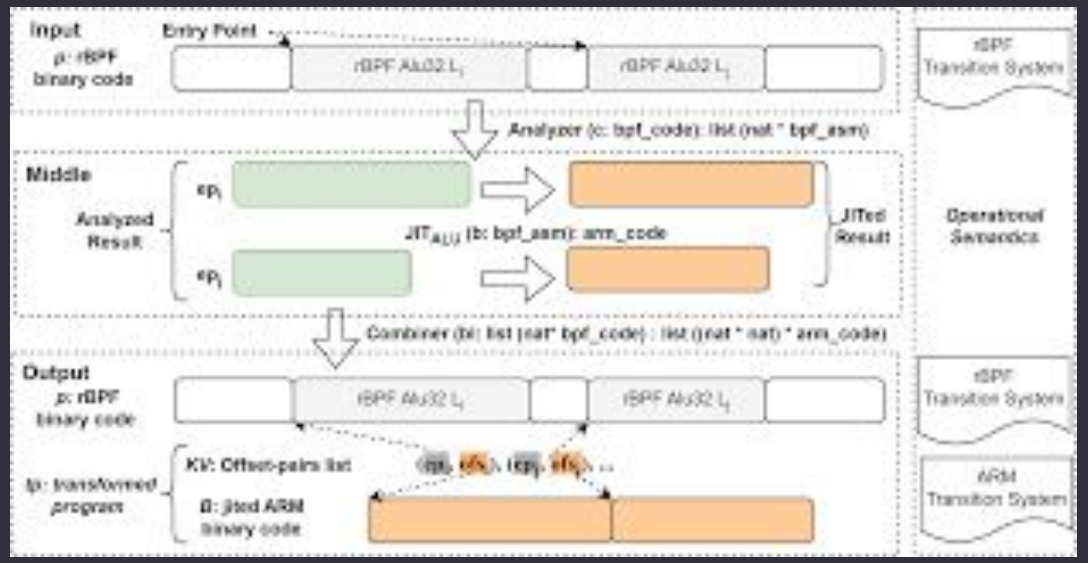
How eBPF programs run inside the kernel:

- Write code in restricted C or Rust
- Compile to eBPF bytecode via LLVM/Clang
- Load bytecode into kernel using a syscall
- eBPF Virtual Machine safely executes the code

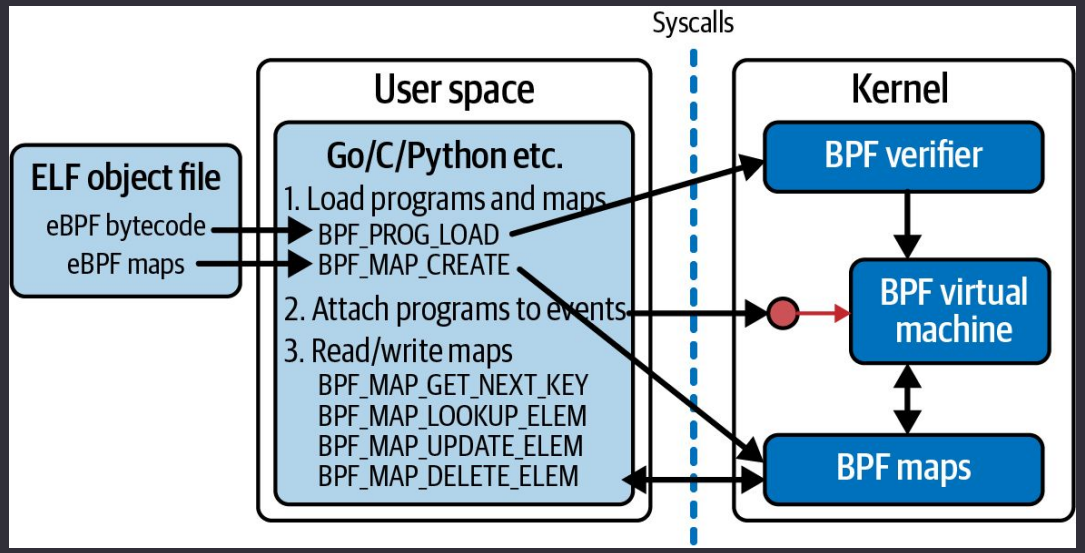
Inside the eBPF Virtual Machine

eBPF runs like a simplified CPU:

- Ten general-purpose registers (r0-r10)
- r0 holds return values; r10 is stack pointer
- 64-bit instruction set (opcodes)
- Just-In-Time (JIT) compiler for native speed



Maps and Attachments: Hooks & Storage



Making eBPF programs practical and interactive:

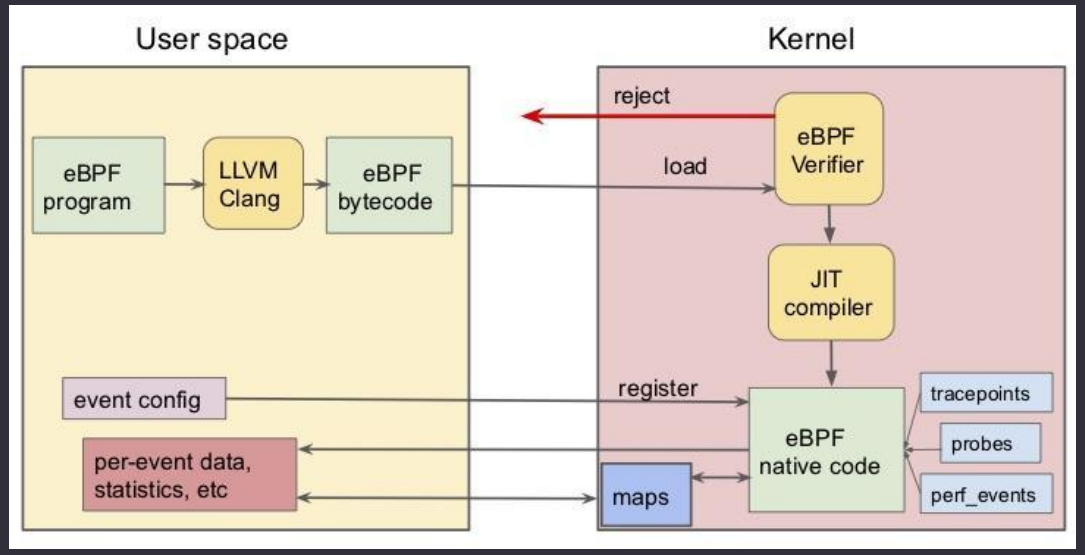
- Attachments (hooks) trigger programs (kprobes, XDP)
- Maps store state: hash tables, arrays, ring buffers
- Maps bridge kernel space and user space communication

The Safety Guarantee: The eBPF Verifier

Ensuring kernel stability and security:

- Kernel modules risk crashing the OS
- Verifier statically analyzes code before running
- Guarantees eBPF programs cannot crash the kernel

The Verification Process



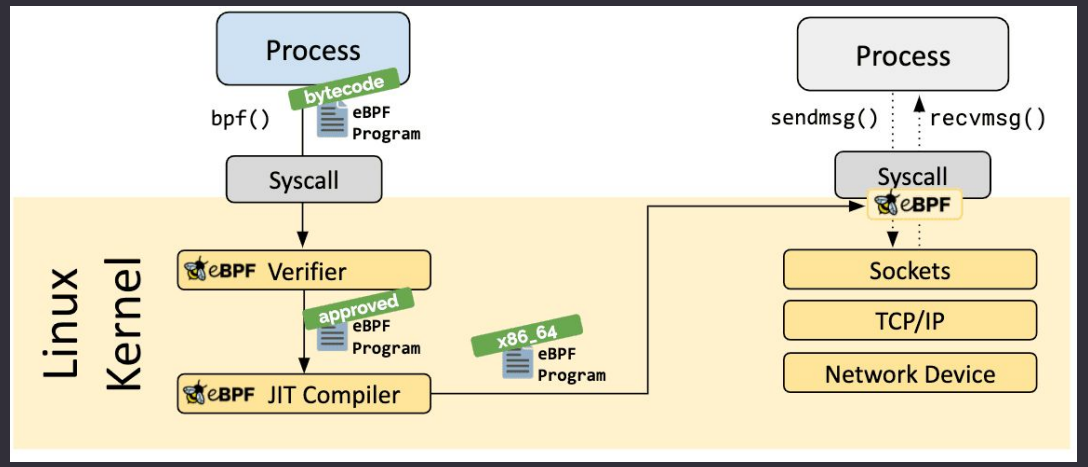
How code safety is assured:

- Static analysis checks all instructions
- Simulates all execution paths (DAG)
- Tracks register states and types precisely

Verifier Constraints for Safety

Rules eBPF programs must follow:

- No invalid memory access or uninitialized variables
- Programs must terminate (no infinite loops)
- Helper function arguments must be correct types
- Limits on program complexity and instruction count



Summary & The Future of eBPF

Key takeaways and outlook:

- eBPF makes the kernel programmable and safe
- High performance with JIT and rich Maps/Hooks
- Future: Windows support, service meshes, enhanced security