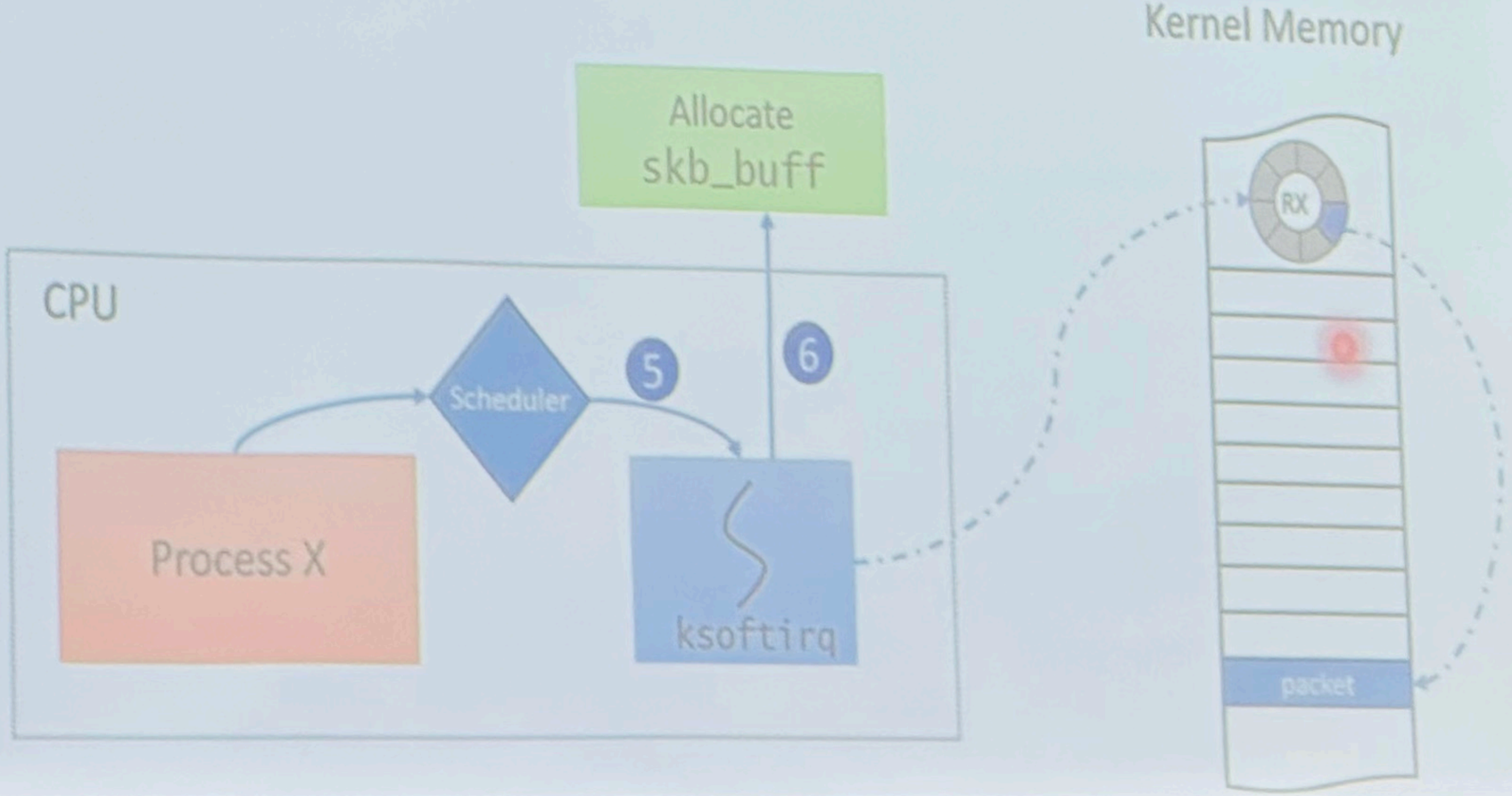
Network Interrupt handling

- Interrupt handling from NIC involves lot of work
  - Ethernet processing, IP routing, TCP reliability and congestion control, ...
- To avoid excessive disruption to interrupted process, NIC interrupt handling split into two parts
- Top half interrupt handler acknowledges interrupt, does minimal processing, disables future interrupts
- Top half schedules a kernel process for full interrupt handling, called bottom half interrupt handler
- Bottom half processes all packets received so far, re-enables interrupts

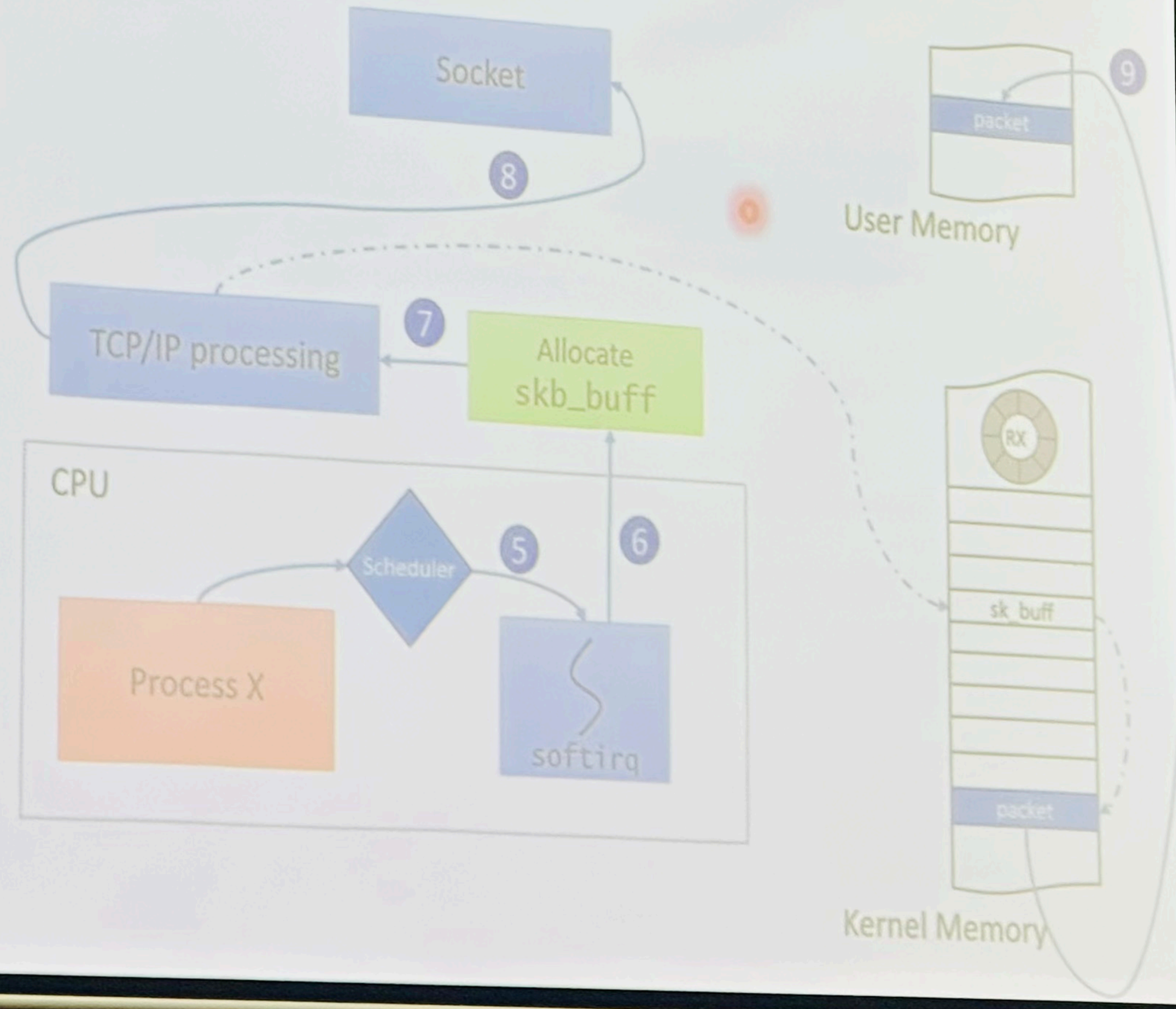
Bottom half interrupt handler

- Bottom half or ksoftirq process scheduled when CPU is free
- Processes all packets collected in the RX ring since the last round
  - Allocates socket buffer (sk_buff) structure for each packet
  - sk_buff contains packet headers, payload

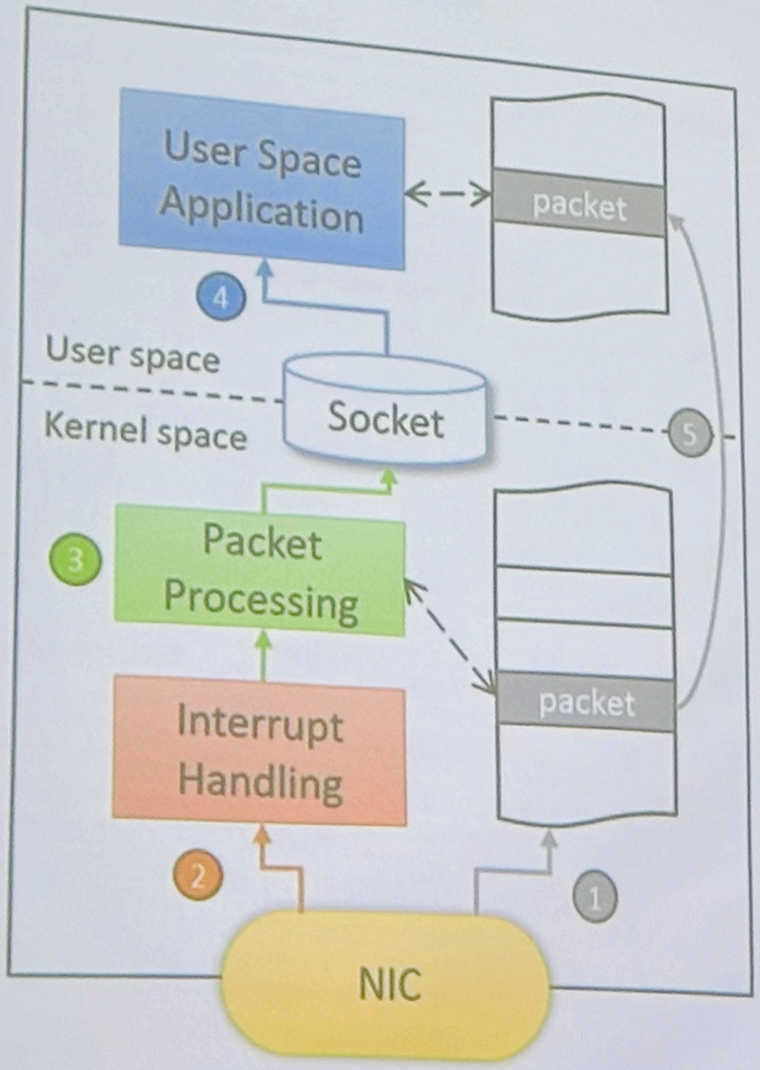


Packet copy to sockets

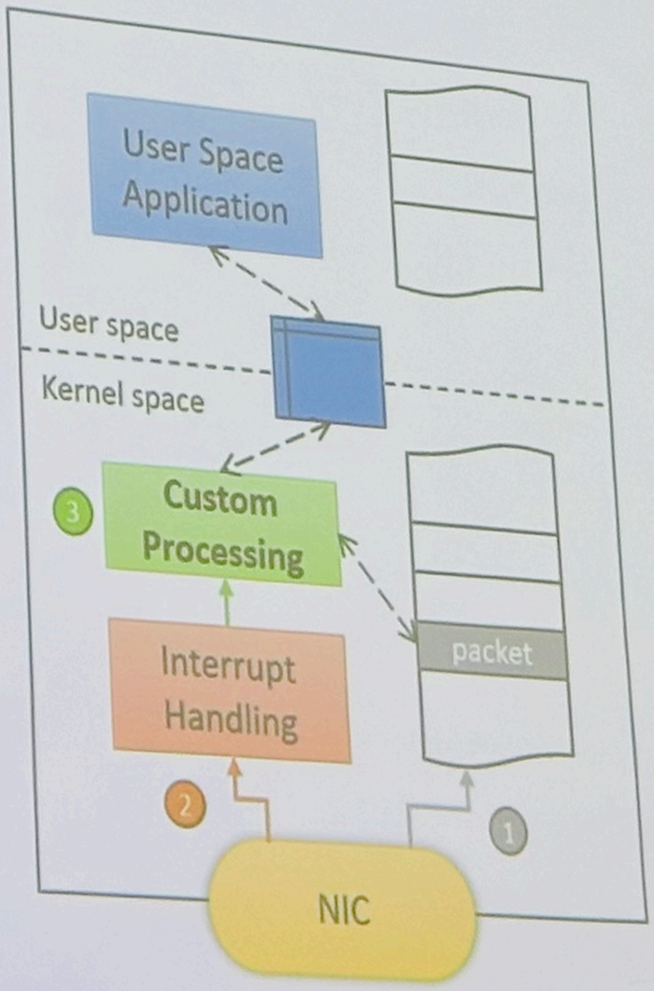
- Packet headers (port number) used to map received packet to socket
- On read from application, packet payload copied from kernel memory to user memory



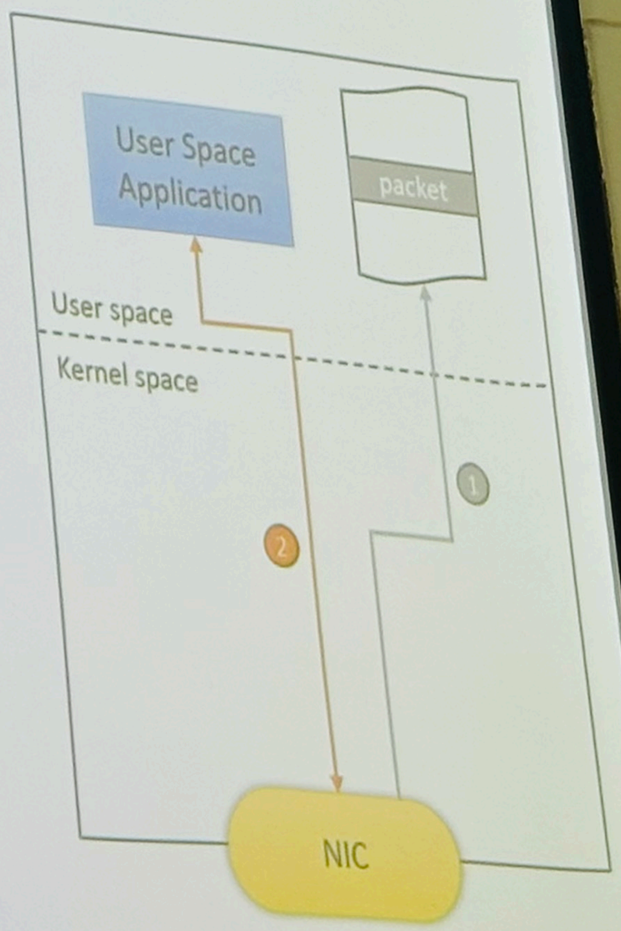
Fast I/O techniques



Generic Kernel Network Stack



In-Kernel Program Offload: Push some extra application functionality into the kernel



Kernel Bypass: Get packets directly into userspace application

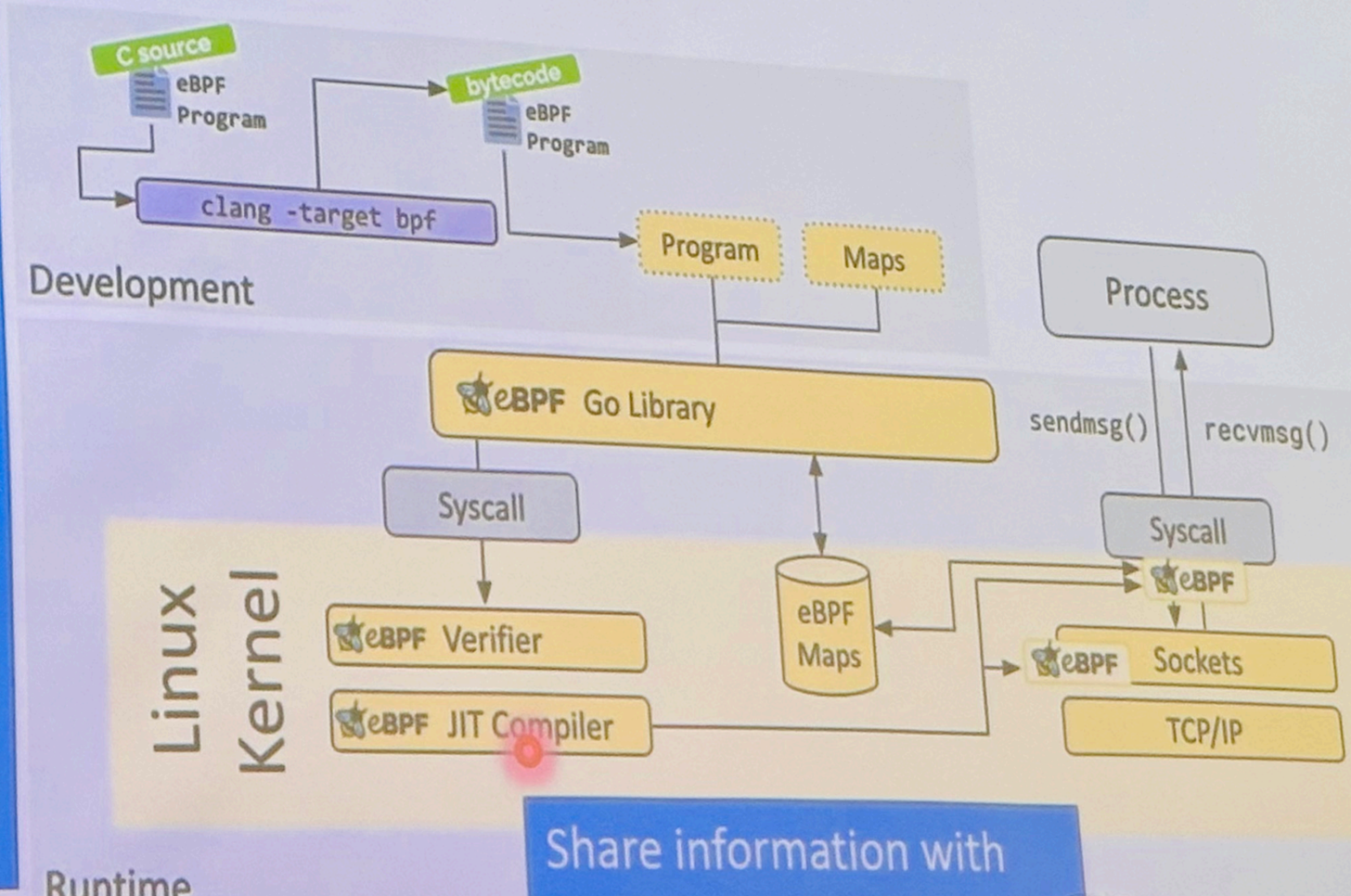
In-kernel packet processing with eBPF

- eBPF (extended Berkeley Packet Filter) is a way to embed custom packet processing code in the kernel safely at specific hook points

User writes C-like code and loads into eBPF hooks

Code verified to be safe (does not crash kernel)

Compiled byte code loaded at kernel hook points



User's compiled code runs inside kernel at desired hook points

Share information with userspace via eBPF maps

Image source: [ebpf.io]

eBPF hook points

- Which hook points to use when?
- XDP hook invoked in device driver when packet arrives
- TC hook invoked during TCP/IP processing
- Socket filter hook during socket selection
- Each hook has different fields of information about packet available, suitable hook chosen

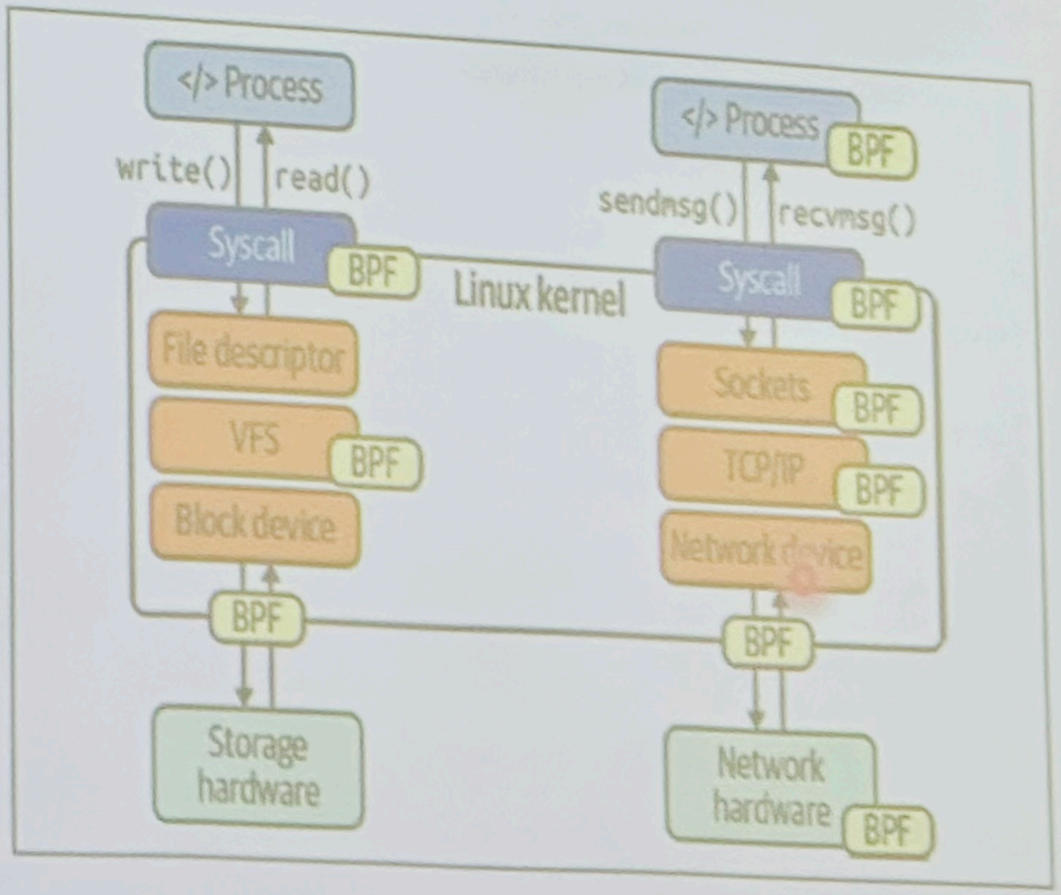


Image source: Security and Observability with eBPF, O'Reilly Report

Kernel bypass with AF_XDP

- Regular sockets (AF_INET) receive packets after TCP/IP processing
- AF_XDP is special type of socket that can receive packets directly from XDP hook
- Packet DMA directly into user space (with driver support), no extra copy, no kernel stack processing overhead
- Program at XDP hook notifies user space app via poll/interrupt mechanisms
- Higher throughputs possible
- Receives “raw” packets with TCP/IP headers