libkrun: More than a VMM, in Dynamic Library Form

Sergio López Pascual, Red Hat
Principal Software Engineer

What is libkrun?

libkrun in a single quote

- “A dynamic library that enables other programs to easily gain KVM-based isolation capabilities, with the minimum possible footprint”

libkrun goals and non-goals

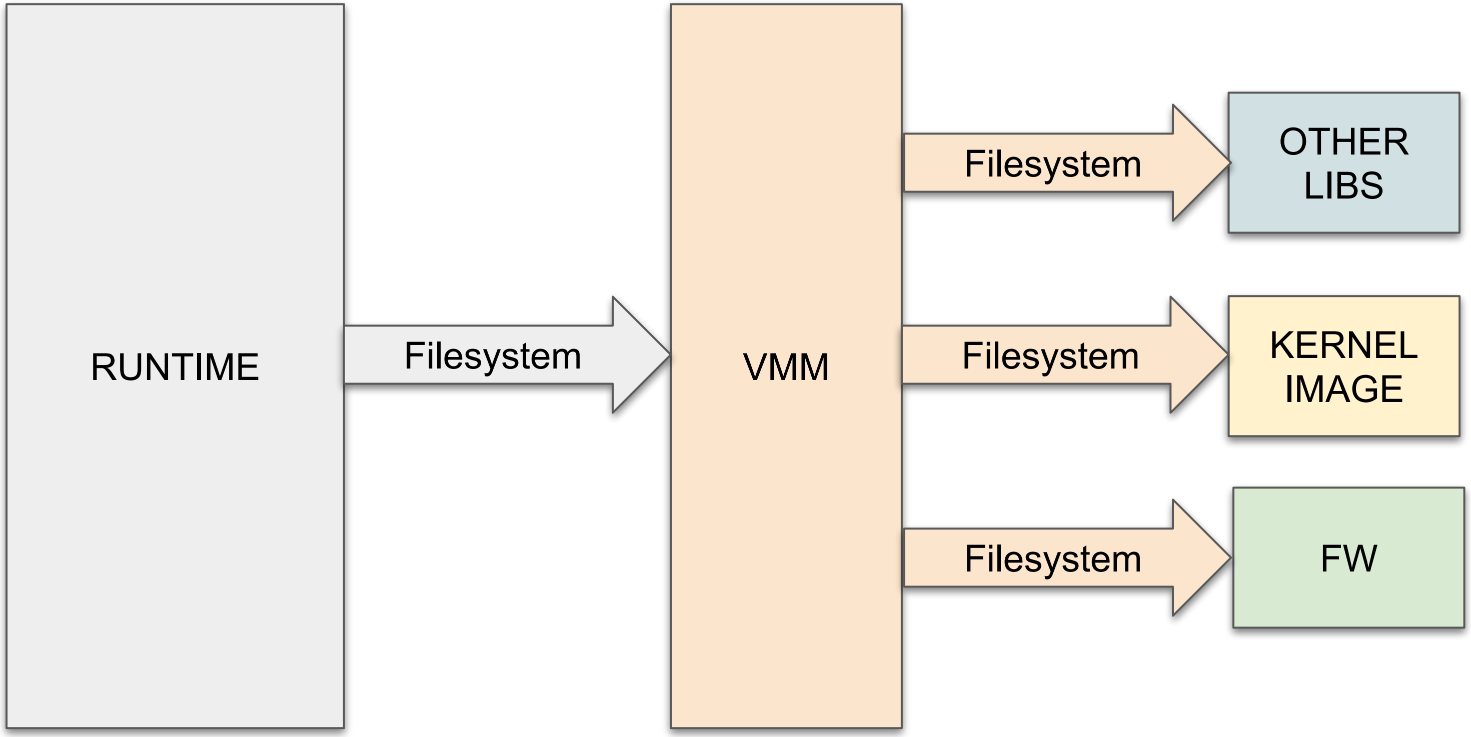
- Goals
  - Be easy to use.
  - Integrate all the features needed for for its purpose, with minimal external dependencies.
  - Be as small as possible in code size.
  - Have the minimum possible memory footprint.
  - Provide a friendly environment for microservice and container workloads.
- Non-goals
  - Support conventional virtualization workloads.

libkrun integrated components

- Provided by libkrun
  - C-bindings to interact with the library.
  - Virtual Machine Monitor (VMM) based on rust-vmm crates.
  - Arch-dependent devices.
  - An integrated virtio-fs server.
  - A minimal set of virtio devices: virtio-console, virtio-fs, virtio-balloon (partial), virtio-vsock.
- Provided by libkrunfw (libkrun links against this library)
  - An interface to access the guest payload.
  - A bundled, minimalist Linux kernel as payload.

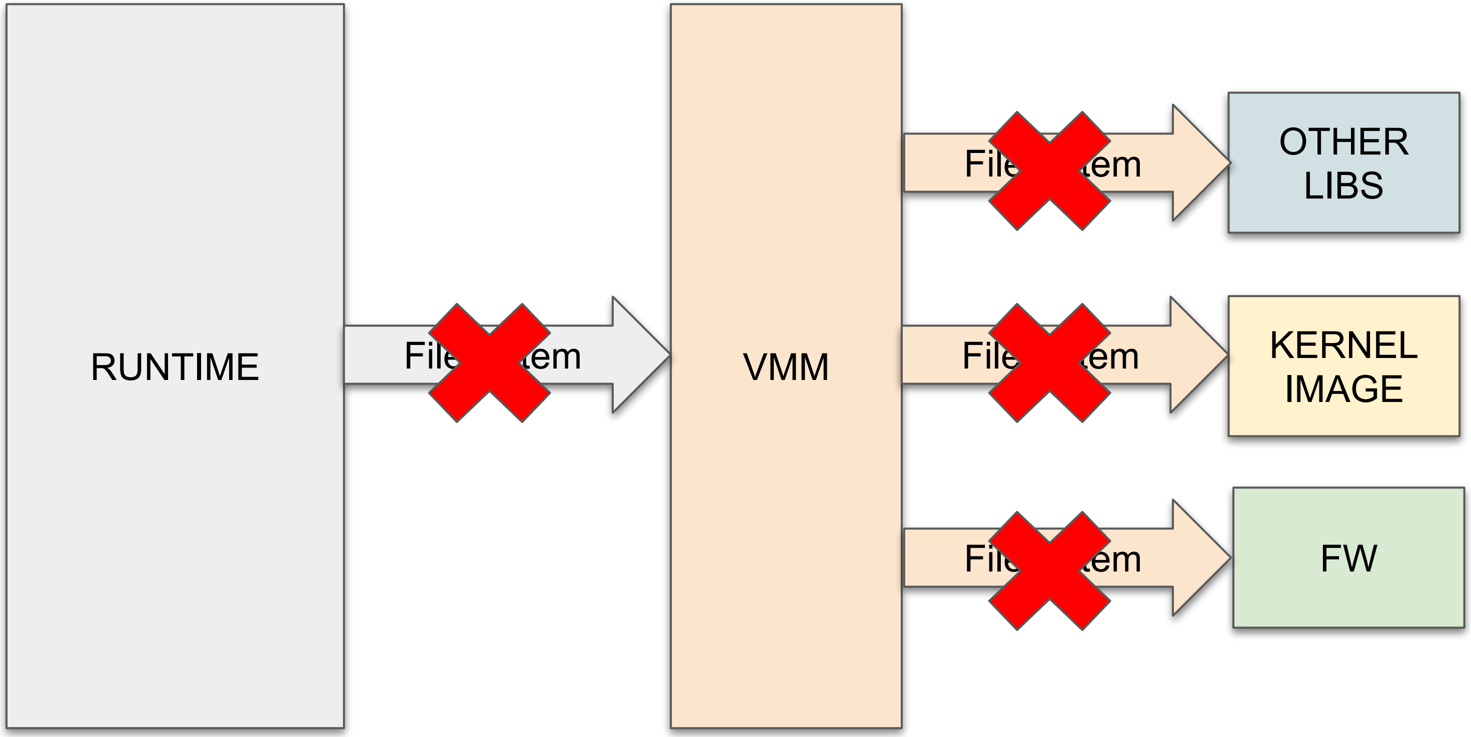
Okay, but why a dynamic library?

Using an external VMM



Okay, but why a dynamic library?

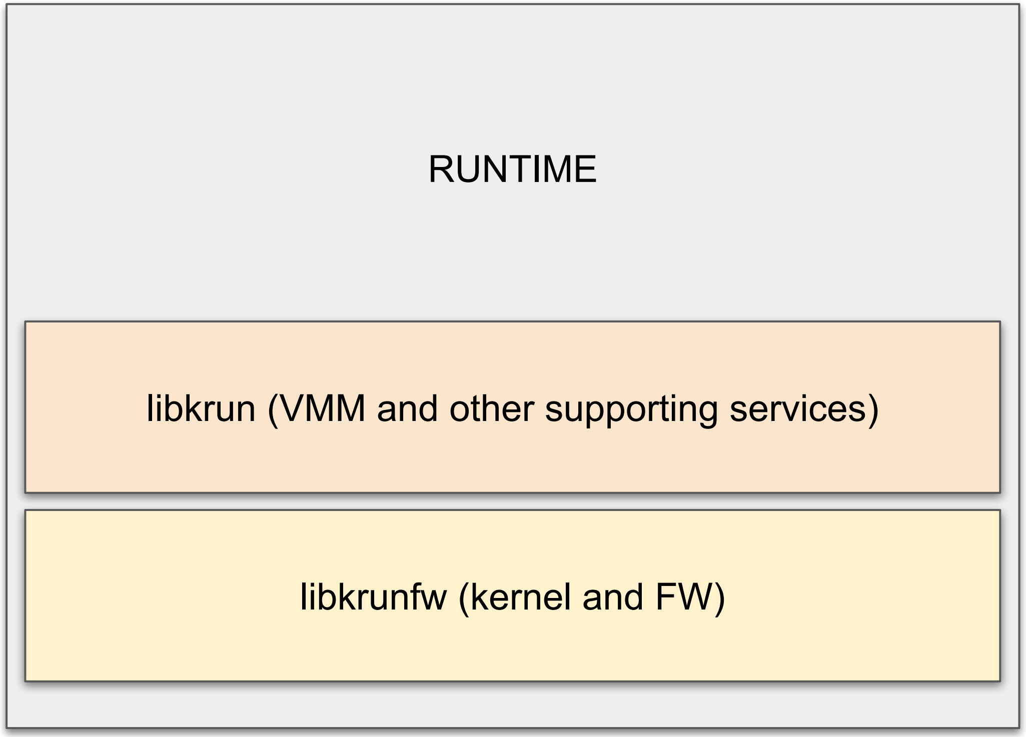
Using an external VMM, after the Runtime switches to a new mountpoint namespace



Okay, but why a dynamic library?

With libkrun

Process memory map



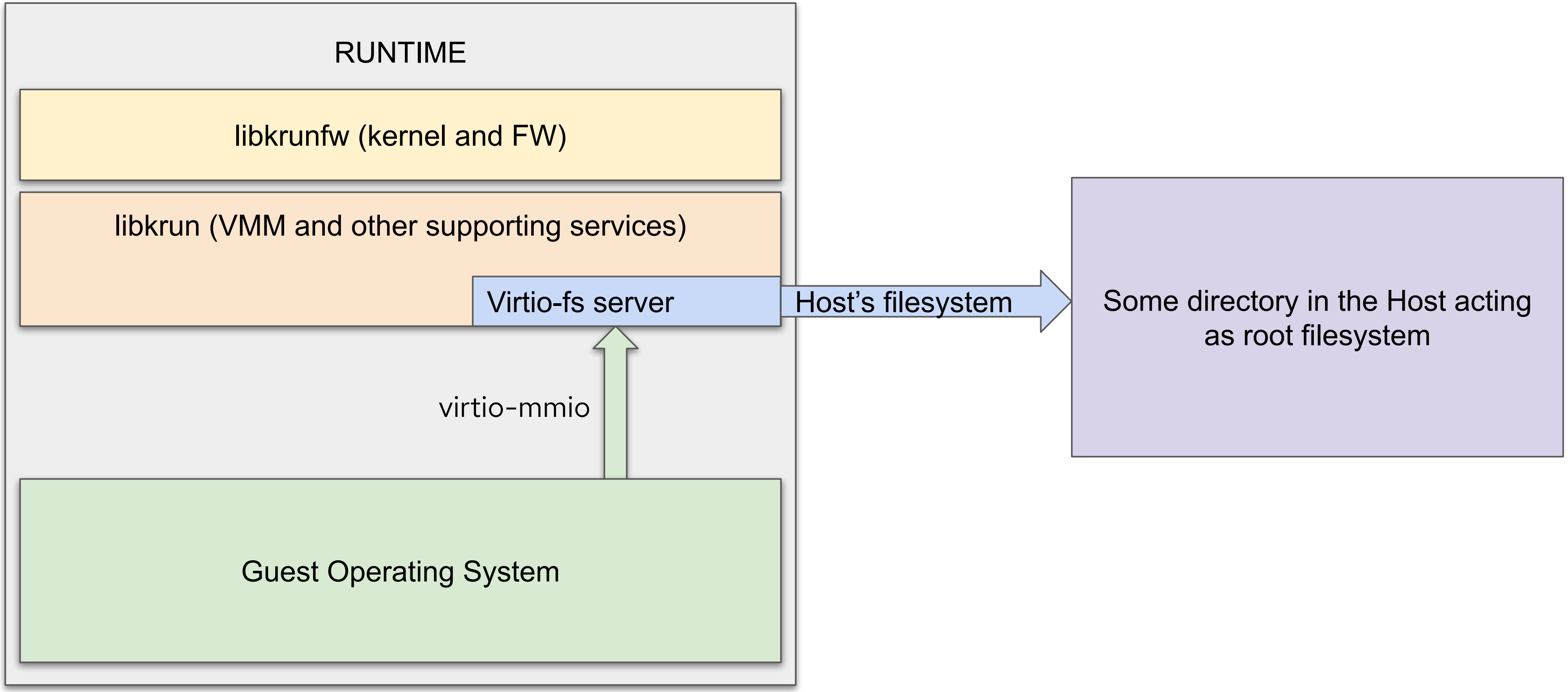
RUNTIME

libkrun (VMM and other supporting services)

libkrunfw (kernel and FW)

Doing storage without block devices (I)

Using virtio-fs to use any directory in the Host as the Guest’s root filesystem


RUNTIME
libkrunfw (kernel and FW)
libkrun (VMM and other supporting services)
Virtio-fs server
Host’s filesystem
Some directory in the Host acting as root filesystem
virtio-mmio
Guest Operating System

Doing storage without block devices (II)

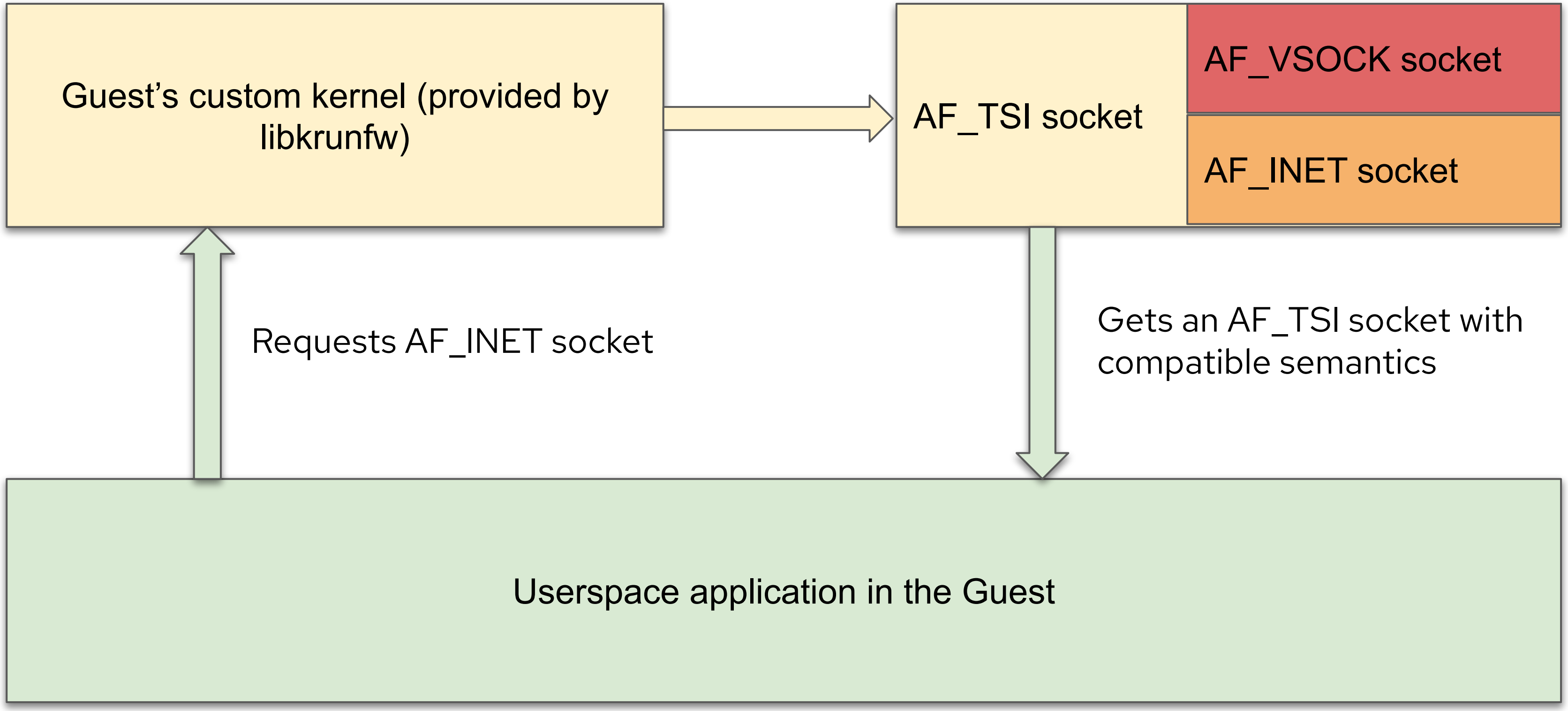
- Advantages of this mechanism
  - Zero storage management (image management, partitioning, layering a FS...)
  - Allows to easily share files between Host and the Guest out-of-the-box.
  - Very friendly to microservice and container workloads.
- Disadvantages
  - Performance is not as good as when using block-based devices.
    - Cache in the Guest vs. cache in the Host.
      - Albeit this is good for our memory footprint!
  - The attack surface is larger than using virtio-blk.
    - More code, more syscalls.

Doing storage without block devices (III)

- The SEV-enabled version of libkrun replaces virtio-fs with virtio-blk
  - It’s better suited for running confidential workloads.
  - It’s smaller, requires less syscalls and allows us to rely on LUKS2 for integrity and encryption
  - More about this on the “Don’t Peek Into My Container” talk that follows this one.

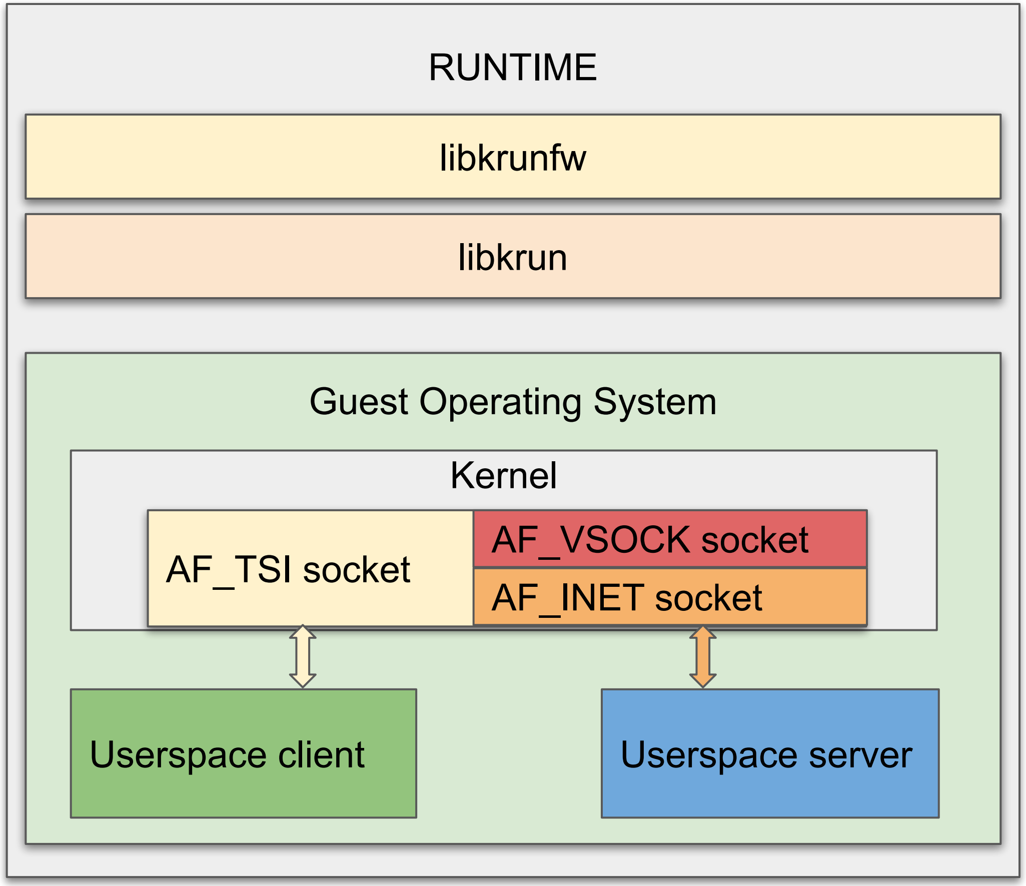
Doing networking without a network interface (I)

Transparent Socket Impersonation (TSI)


Guest's custom kernel (provided by libkrunfw)
AF_TSI socket
AF_VSOCK socket
AF_INET socket
Requests AF_INET socket
Gets an AF_TSI socket with compatible semantics
Userspace application in the Guest

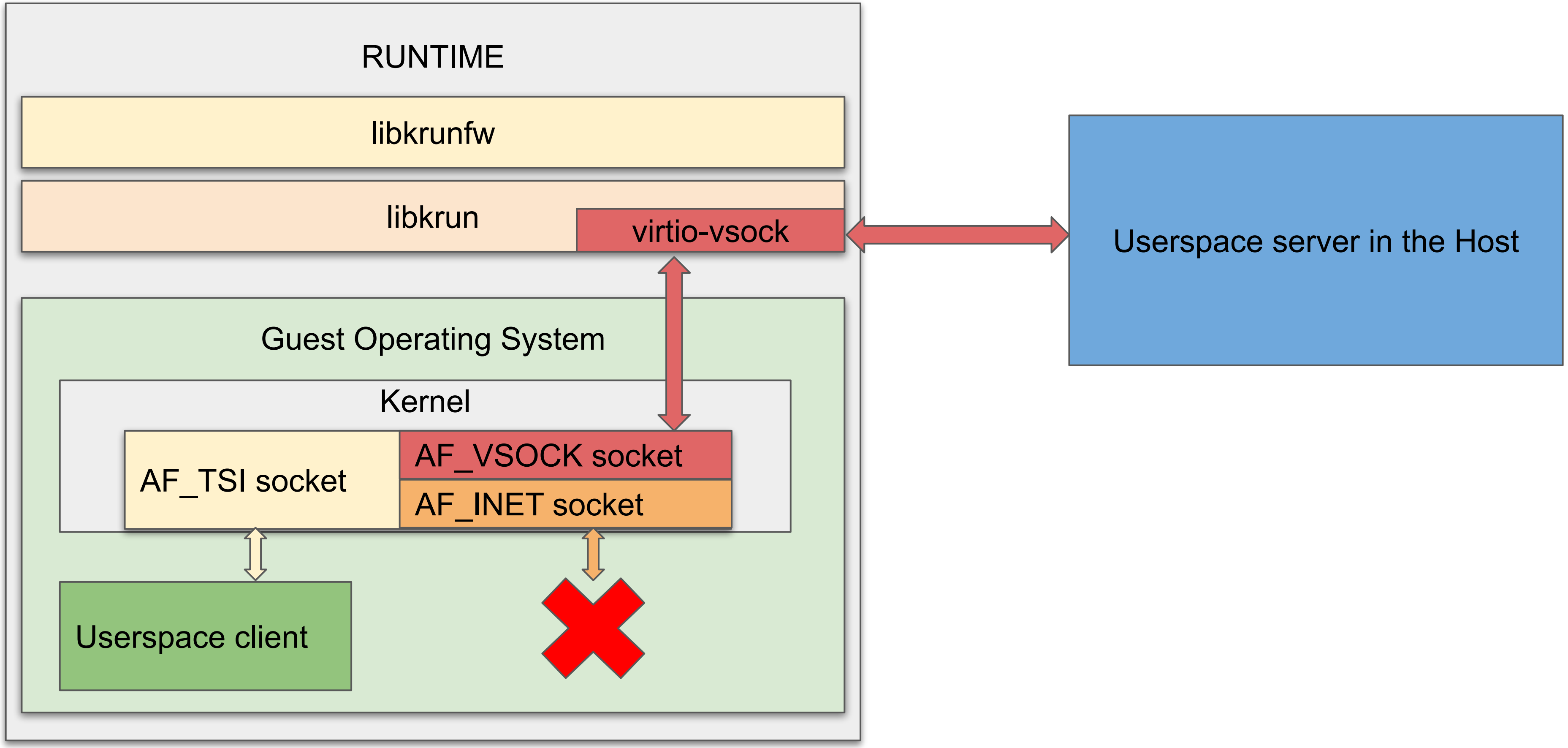
Doing networking without a network interface (II)

Connecting to a local endpoint



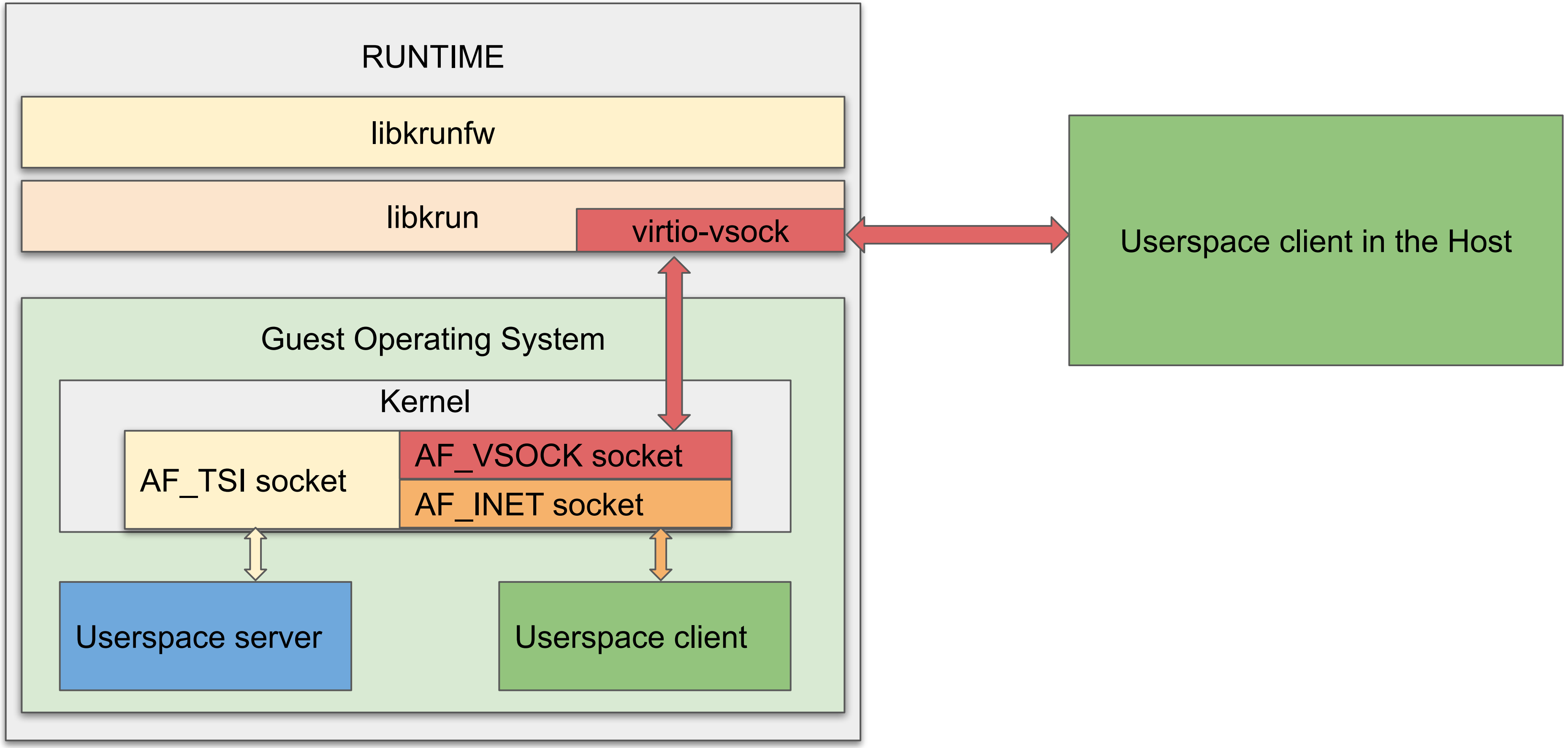
Doing networking without a network interface (III)

Connecting to an external endpoint



Doing networking without a network interface (IV)

Listening on both the external and the internal endpoints



Doing networking without a network interface (V)

- Advantages of this mechanism
  - Minimal (just DNS) network configuration.
  - Allows libkrun to act on behalf of the userspace applications running in the guest, without the need of implementing a TCP network stack in the library.
  - From the host's perspective, all connections come from/go to the libkrun-enabled runtime, and are visible in the network namespace of the runtime's context.
  - There's no need to use network bridges nor iptables rules.
  - As a result of all the above, the environment is very friendly to container workloads.
    - Things such as Istio sidecars work out-of-the-box!
- Disadvantages
  - Requires explicit support for each address family (only AF_INET streams supported ATM)
  - No raw sockets.

Using libkrun

Obtaining libkrun

- Binaries
  - Shipped by openSUSE Tumbleweed
  - COPR repository for Fedora
    - https://copr.fedorainfracloud.org/coprs/fulltext/?fulltext=libkrun
  - Homebrew repository (Tap) for macOS/M1 (uses Hypervirsor.framwork instead on KVM)
    - https://github.com/slp/homebrew-krun

- Building from sources
  1. https://github.com/containers/libkrunfw
  2. https://github.com/containers/libkrun

Obtaining libkrun

- Headers
    - libkrun.h - Includes documentation for each function.
- Libraries
    - libkrun.(so|dylib) (will bring libkrunfw into the mix)
- Linking
    - gcc -o minimal minimal.c -lkrun

Minimal example

```
#include <libkrun.h>
void main()
{
    char *const envp[] = { 0 };
    int ctx_id = krun_create_ctx();
    krun_set_vm_config(ctx_id, 1, 512);
    krun_set_root(ctx_id, "rootfs");
    krun_set_exec(ctx_id, "/bin/sh", 0, &envp[0]);
    krun_start_enter(ctx_id);
}
```

Examples and use cases

- Projects already using libkrun
  - Create lightweight VMs from OCI images
    - **krunvm**: libkrun’s sister project.
    - https://github.com/containers/krunvm
  - Provide Virtualization-based isolation for containers
    - **crun**: OCI runtime used by podman, which already supports using libkrun.
    - https://github.com/containers/crun
- Ideas being worked on using libkrun
  - Run fully-encrypted workloads using AMD SEV-SNP and Intel TDX.
- Other ideas
  - Enable conventional services to self-isolate.
  - Enable a microservice platform to deploy functions in Virtualization-isolated environments.

Thank you

Red Hat is the world’s leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat