Playing Lego with Virtualization Components

Andreea Florescu
fandree@amazon.com

Samuel Ortiz
samuel.ortiz@intel.com

Ever heard about Rust?

Ever actually used Rust?

Rust Ownership

Memory safety

Safe and simple concurrency

No garbage collector

Rust and VMMs: Why?

VMM Components/Requirements	Rust Features
Memory model, virtio, etc	Memory Safety
vCPU threads, I/O workers	Safe Concurrency
Virtualization overhead Latency	Performance

rust-vmm

What is rust-vmm?

- Building blocks for VMMs written in Rust
- Virtualization components (crates)
- Open Source

Why rust-vmm?

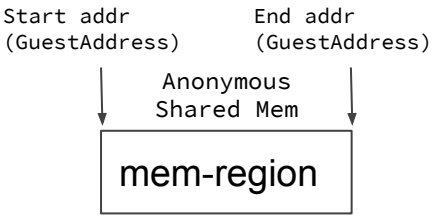
- Faster development for new custom VMMs
- Security & Testability
- Clean interface

vm-memory - Firecracker

- Guest Address

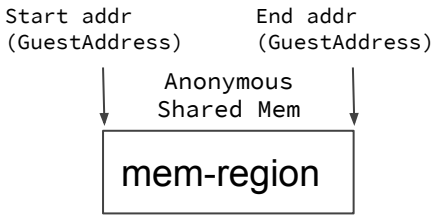
vm-memory - Firecracker

- Guest Address
- Memory Region



vm-memory - Firecracker

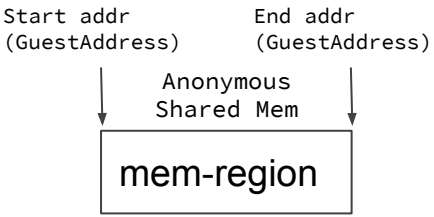
- **struct** Guest Address
- **struct** Memory Region
- **struct** Guest Memory



Guest Memory

vm-memory - Firecracker

- **struct** Guest Address
- **struct** Memory Region
- **struct** Guest Memory



vm-memory - rust-vmm

- **Trait** Guest Address
- **Trait** Memory Region
- **Trait** Guest Memory

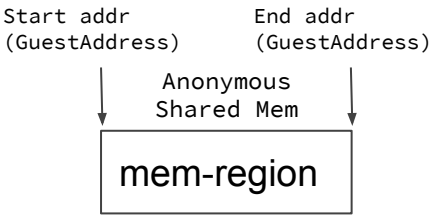
Guest Memory

What is a trait?

- *“A trait is a collection of methods defined for an unknown type: Self”*
- Interface
- Methods with default implementation

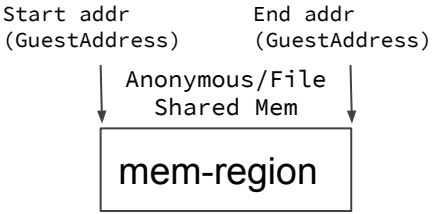
vm-memory - Firecracker

- **struct** Guest Address
- **struct** Memory Region
- **struct** Guest Memory



vm-memory - rust-vmm

- **trait** Guest Address
- **trait** Memory Region
- **trait** Guest Memory



Guest Memory

Component Dependency

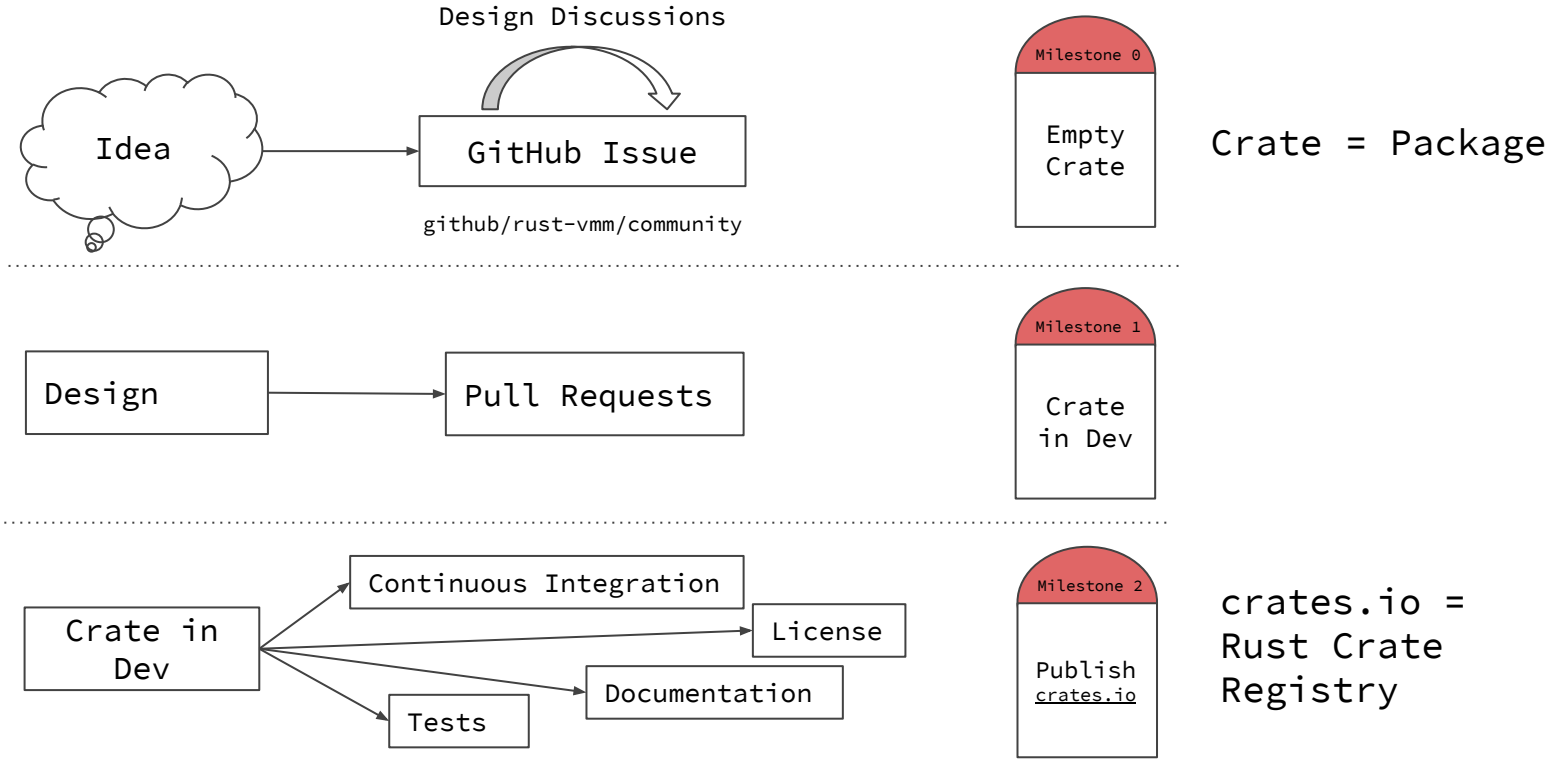
vm-memory

```
pub trait GuestMemory {
  fn read_from<F>(...);
  …
}
pub struct GuestMemoryMmap {
  regions: Arc<Vec<GuestRegionMmap>>,
}
impl GuestMemory for GuestMemoryMmap {
 fn read_from<F>(...) { … }
}
```

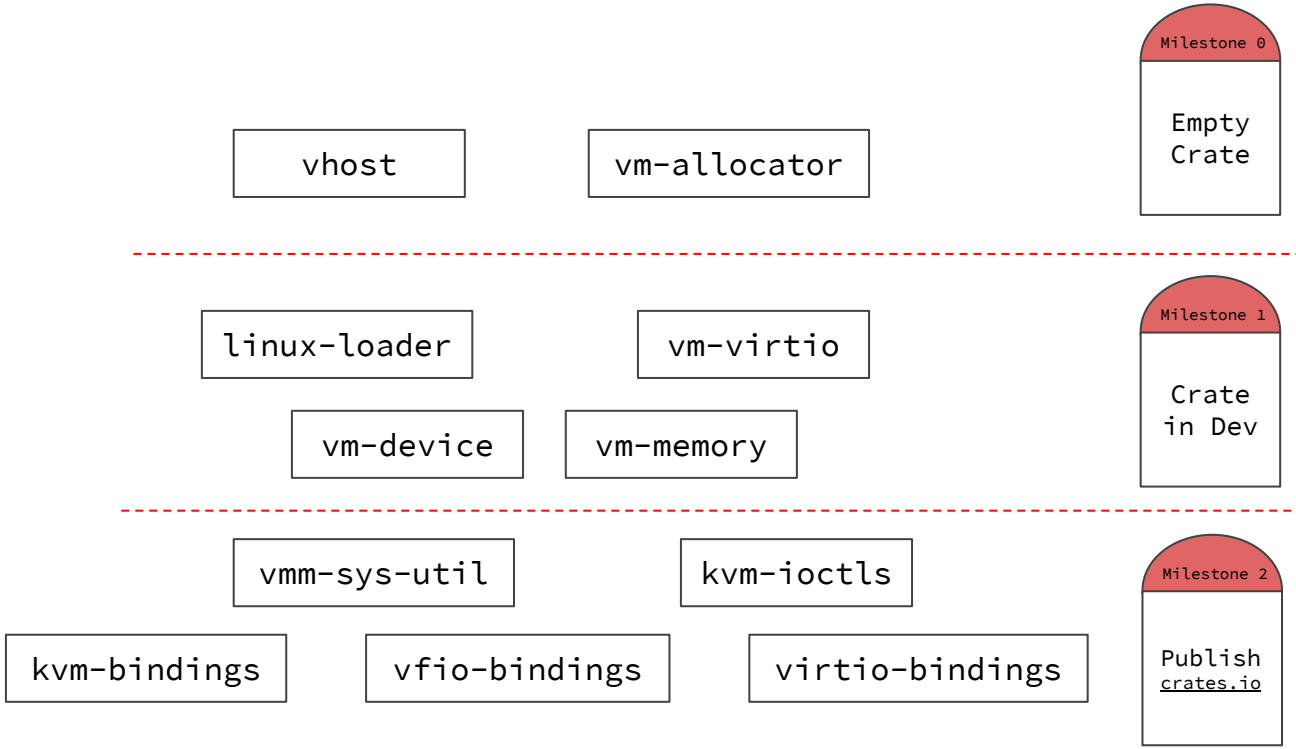
linux-loader

```
fn load<F, M: GuestMemory>(
      guest_mem: &M,
      kernel_image: &mut F,
      highmem_start_address: GuestAddress,
)
```

From idea to published crate



Current Status



rust-vmm in practice

Contribute

Alibaba

AWS (Firecracker)

CloudBase

Google (CrosVM)

Intel (Cloud Hypervisor)

Red Hat

Contribute and Consume

AWS (Firecracker)

Intel (Cloud Hypervisor)

How do you build a VMM from rust-vmm?

How do you build a VMM from rust-vmm?

Cherry-pick your features

How do you build a VMM from rust-vmm?

Cherry-pick your features

Use functional rust-vmm crates (crates.io or github deps)

How do you build a VMM from rust-vmm?

Cherry-pick your features

Use functional rust-vmm crates (crates.io or github deps)

Fork incomplete rust-vmm crates

How do you build a VMM from rust-vmm?

Cherry-pick your features

Use functional rust-vmm crates (crates.io or github deps)

Fork incomplete rust-vmm crates

Implement missing rust-vmm crates

How do you build a VMM from rust-vmm?

Cherry-pick your features

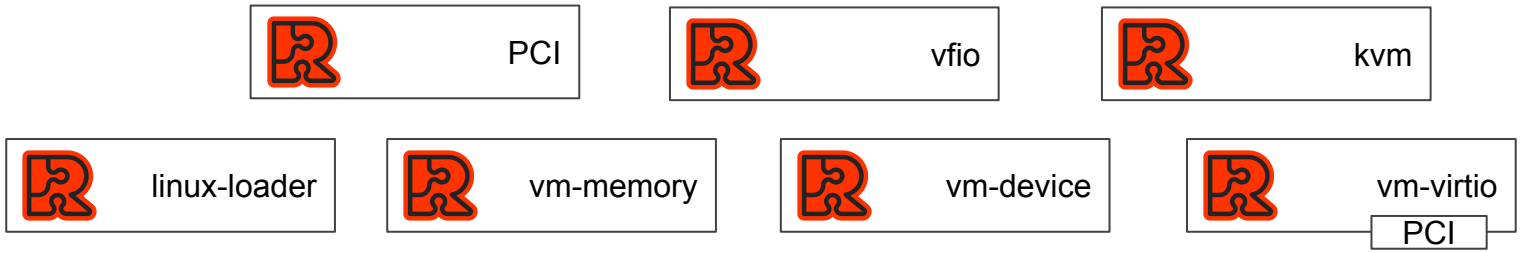
Use functional rust-vmm crates (crates.io or github deps)

Fork incomplete rust-vmm crates

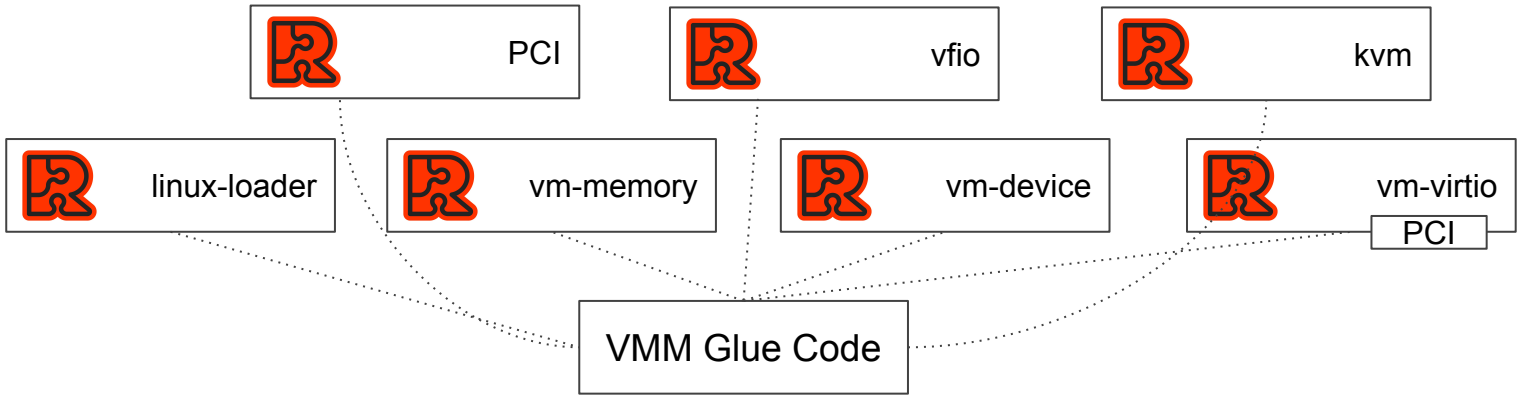
Implement missing rust-vmm crates

Implement the VMM glue code

KVM



KVM


PCI
vfio
kvm
linux-loader
vm-memory
vm-device
vm-virtio
PCI
VMM Glue Code

KVM

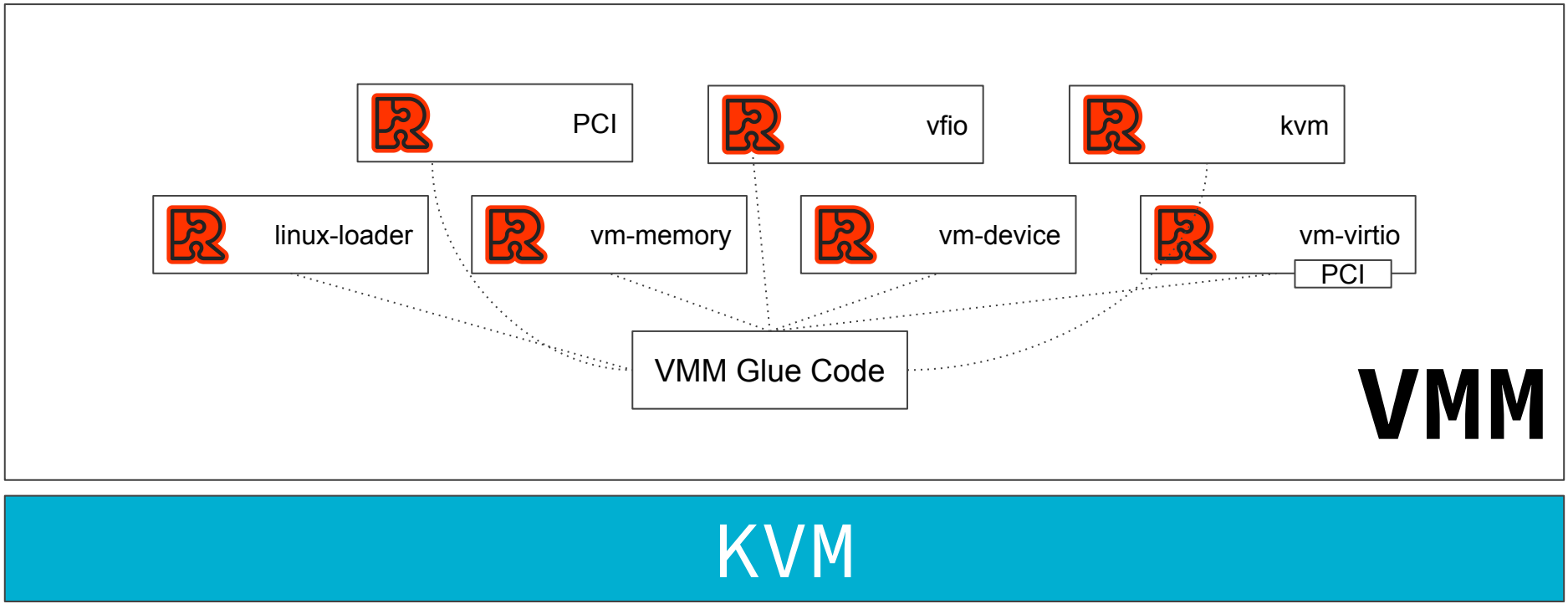
```
[dependencies.vm-memory]
git = "https://github.com/rust-vmm/vm-memory"
features = ["backend-mmap"]
```

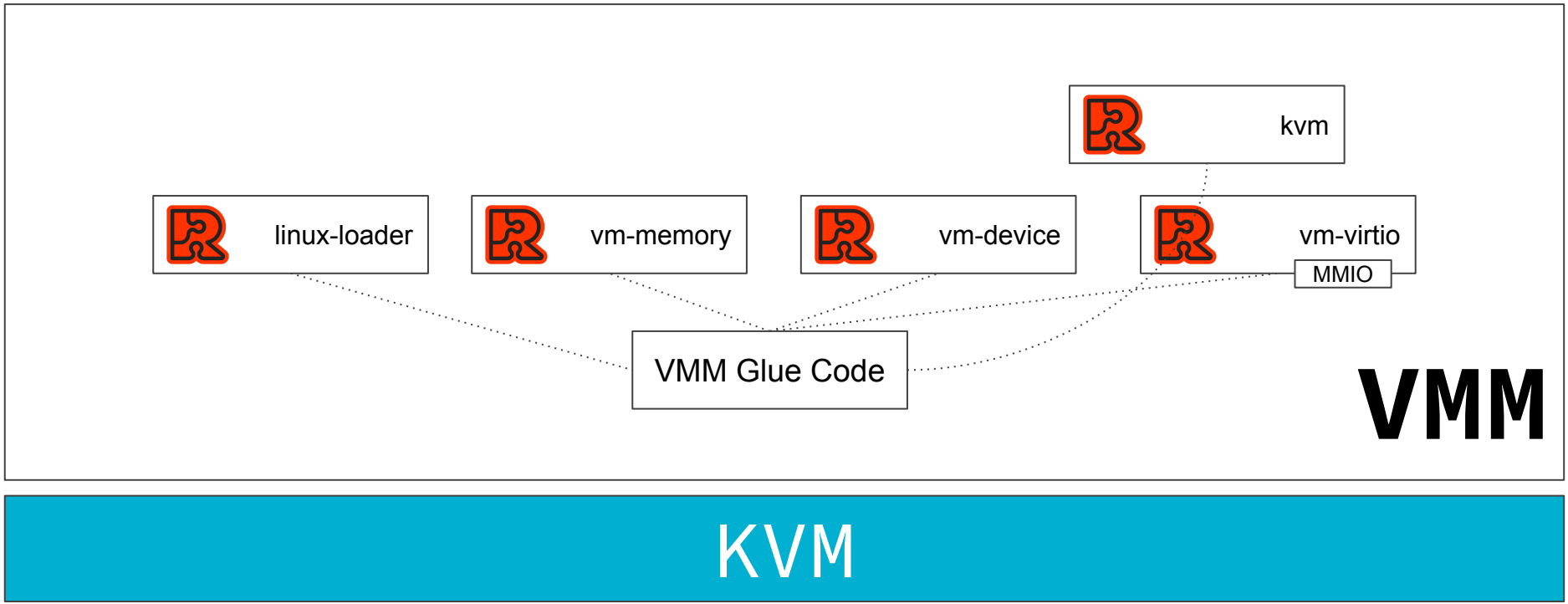
```
use vm_memory::guest_memory::FileOffset;
use vm_memory::{
    Address, Bytes, Error as MmapError, GuestAddress, GuestMemory, GuestMemoryMmap,
    GuestMemoryRegion, GuestUsize,
};
```

```rust
let guest_memory = match config.memory.file {
    Some(ref file) => {
        [ SNIP SNIP SNIP ]
        GuestMemoryMmap::with_files(&mem_regions).map_err(Error::GuestMemory)?
    }
    None => GuestMemoryMmap::new(&ram_regions).map_err(Error::GuestMemory)?,
};

guest_memory
    .with_regions(|index, region| {
        let mem_region = kvm_userspace_memory_region {
            slot: index as u32,
            guest_phys_addr: region.start_addr().raw_value(),
            memory_size: region.len() as u64,
            userspace_addr: region.as_ptr() as u64,
            flags: 0,
        };

        // Safe because the guest regions are guaranteed not to overlap.
        unsafe { fd.set_user_memory_region(mem_region) }
    })
    .map_err(|_| Error::GuestMemory(MmapError::NoMemoryRegion))?;
```


PCI
vfio
kvm
linux-loader
vm-memory
vm-device
vm-virtio
PCI
VMM Glue Code
VMM
KVM


kvm
linux-loader
vm-memory
vm-device
vm-virtio
MMIO
VMM Glue Code
VMM
KVM

Cloud Hypervisor

Cloud ~~Hypervisor~~ VMM

Cloud ~~Hypervisor~~ VMM

KVM only

Cloud ~~Hypervisor~~ VMM

KVM only

Cloud workloads

Cloud ~~Hypervisor~~ VMM

KVM only

Cloud workloads

x86_64 and aarch64

Cloud ~~Hypervisor~~ VMM

KVM only

Cloud workloads

x86_64 and aarch64

PCI based, VFIO, ACPI, Migration, vhost-user

rust-vmm crates

All functional rust-vmm crates

rust-vmm crates

All functional rust-vmm crates

Bindings (KVM, virtio, VFIO)

KVM (kvm-ioctls)

Memory model (vm-memory)

Kernel loader (linux-loader)

Utilities (vmm-sys-util)

Fork WIP crates

---

vm-virtio

vm-device

VFIO

Implement missing crates

PCI

qcow

migration

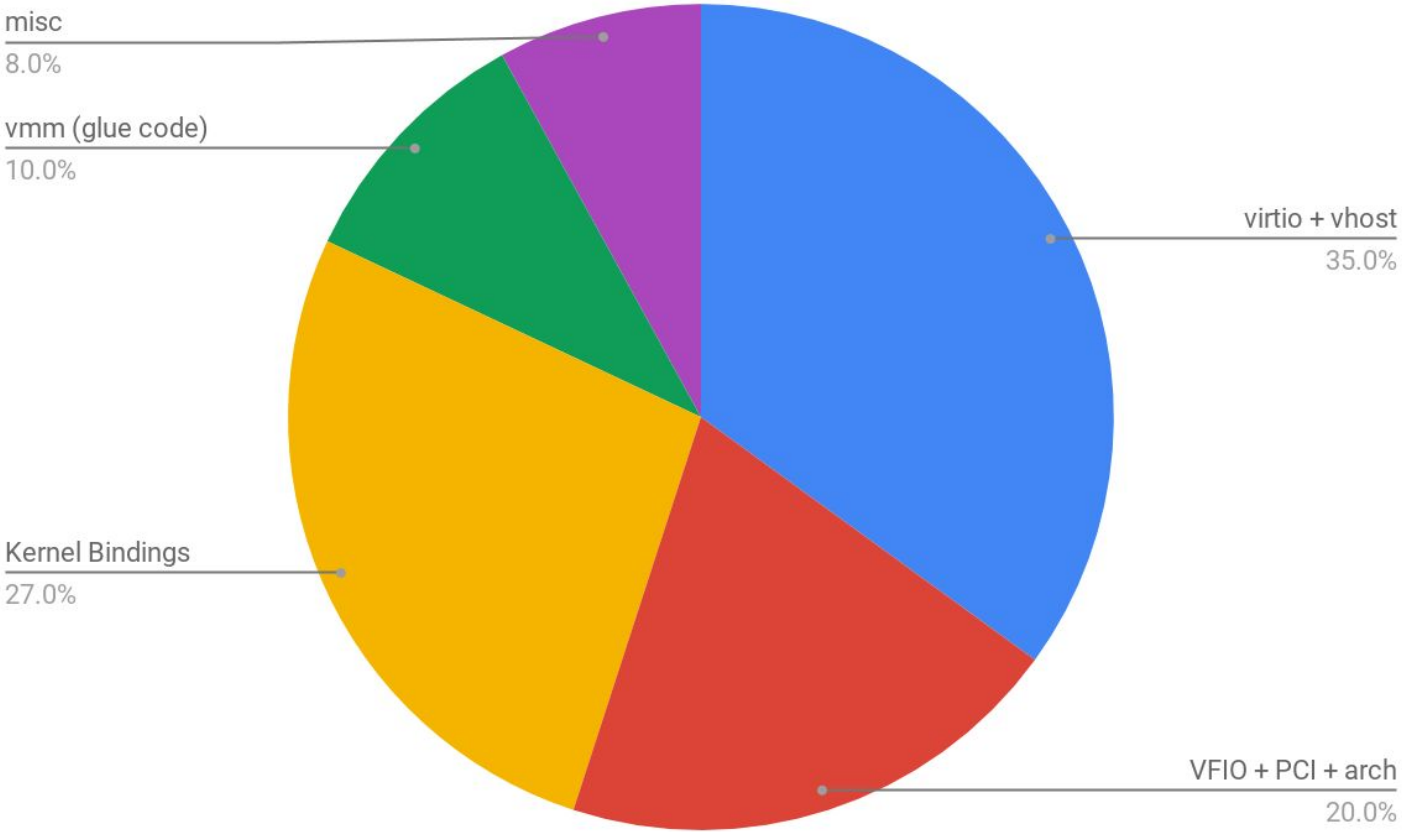
arch


SNOWMEN
expectation
reality

Reality Check

Still 40K Lines of Code

Fundamental crates missing from rust-vmm


misc
8.0%
vmm (glue code)
10.0%
virtio + vhost
35.0%
Kernel Bindings
27.0%
VFIO + PCI + arch
20.0%

Actions from the rust-vmm meetup

`vm-device`

`vm-virtio`

https://github.com/rust-vmm/

Backup

```rust
fn make_vec() -> Vec<i32> {
    let mut vec = Vec::new();
    vec.push(42);
    vec
}
```

```rust
fn make_vec() -> Vec<i32> {
    let mut vec = Vec::new();
    vec.push(42);
    vec
}

fn print_vec(vec: Vec<i32>) {
    for i in vec.iter() {
        println!("{}", i)
    }
}
```

```rust
fn make_vec() -> Vec<i32> {
    let mut vec = Vec::new();
    vec.push(42);
    vec
}

fn print_vec(vec: Vec<i32>) {
    for i in vec.iter() {
        println!("{}", i)
    }
}

fn main() {
    let vec = make_vec();
    print_vec(vec);
}
```

```rust
fn make_vec() -> Vec<i32> {
    let mut vec = Vec::new();
    vec.push(42);
    vec
}

fn print_vec(vec: Vec<i32>) {
    for i in vec.iter() {
        println!("{}", i)
    }
}

fn main() {
    let vec = make_vec();
    print_vec(vec);
}
```

```
$ cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.02s
     Running `target/debug/outofbound`
42
```

```rust
fn make_vec() -> Vec<i32> {
    let mut vec = Vec::new();
    vec.push(42);
    vec
}

fn print_vec(vec: Vec<i32>) {
    for i in vec.iter() {
        println!("{}", i)
    }
}

fn main() {
    let vec = make_vec();
    print_vec(vec);
    println!("Vector length: {}", vec.len());
}
```

```rust
fn make_vec() -> Vec<i32> {
    let mut vec = Vec::new();
    vec.push(42);
    vec
}

fn print_vec(vec: Vec<i32>) {
    for i in vec.iter() {
        println!("{}", i)
    }
}

fn main() {
    let vec = make_vec();
    print_vec(vec);
    println!("Vector length: {}", vec.len());
```

```
error[E0382]: borrow of moved value: `vec`
  --> src/main.rs:16:35
   |
14 |     let vec = make_vec();
   |         --- move occurs because `vec` has type `std::vec::Vec<i32>`, which does not implement the `Copy` trait
15 |     print_vec(vec);
   |               --- value moved here
16 |     println!("Vector length: {}", vec.len());
   |                                   ^^^ value borrowed here after move

error: aborting due to previous error
```