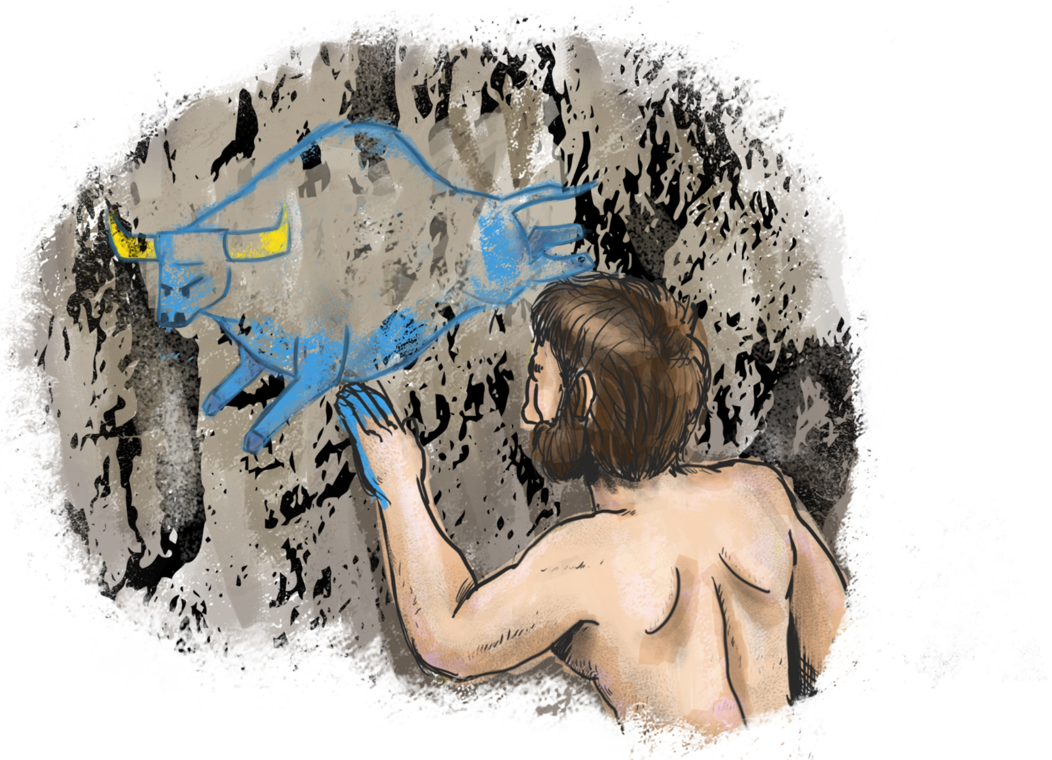
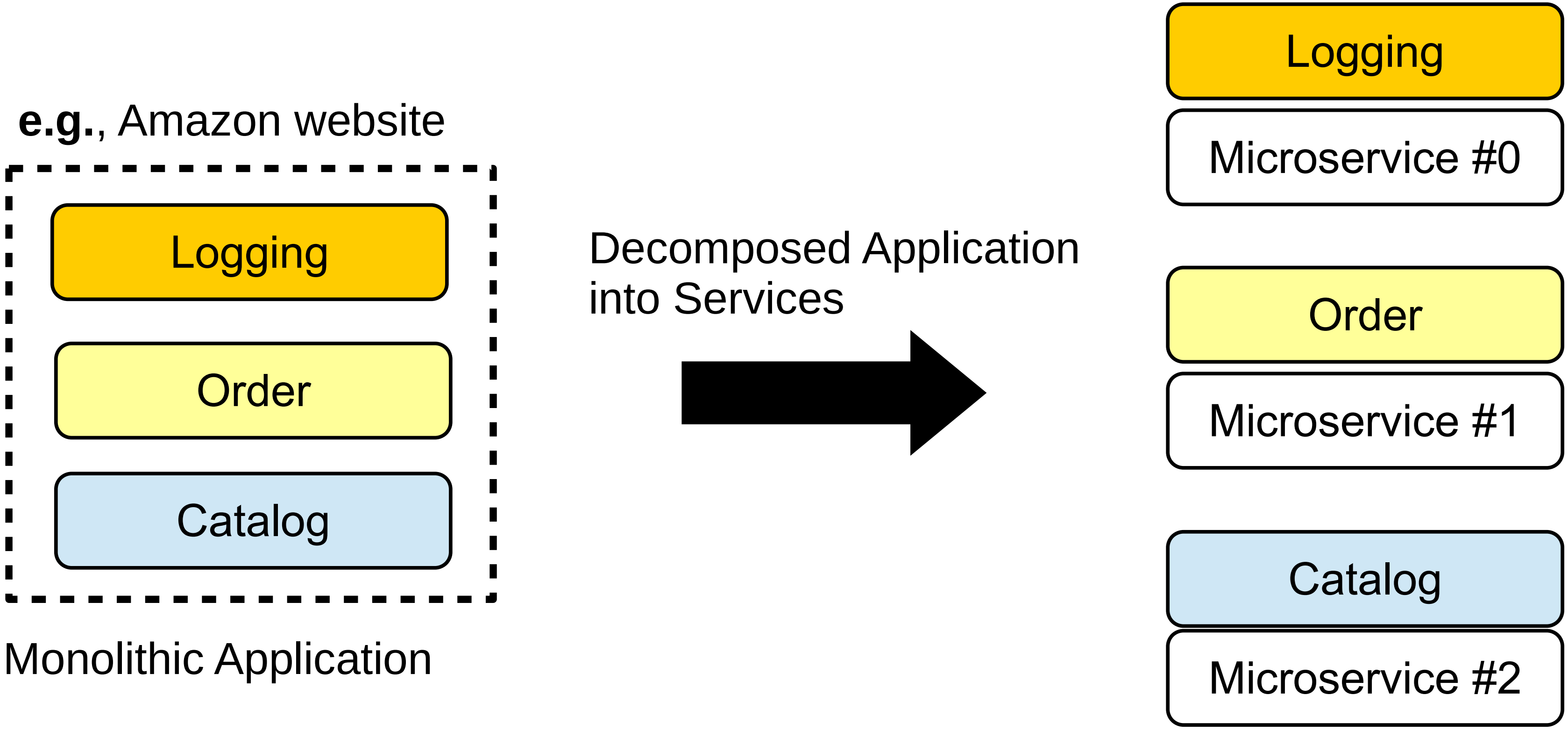


Building a Cloud Infrastructure to Deploy Microservices as MicroVM Toro Guests

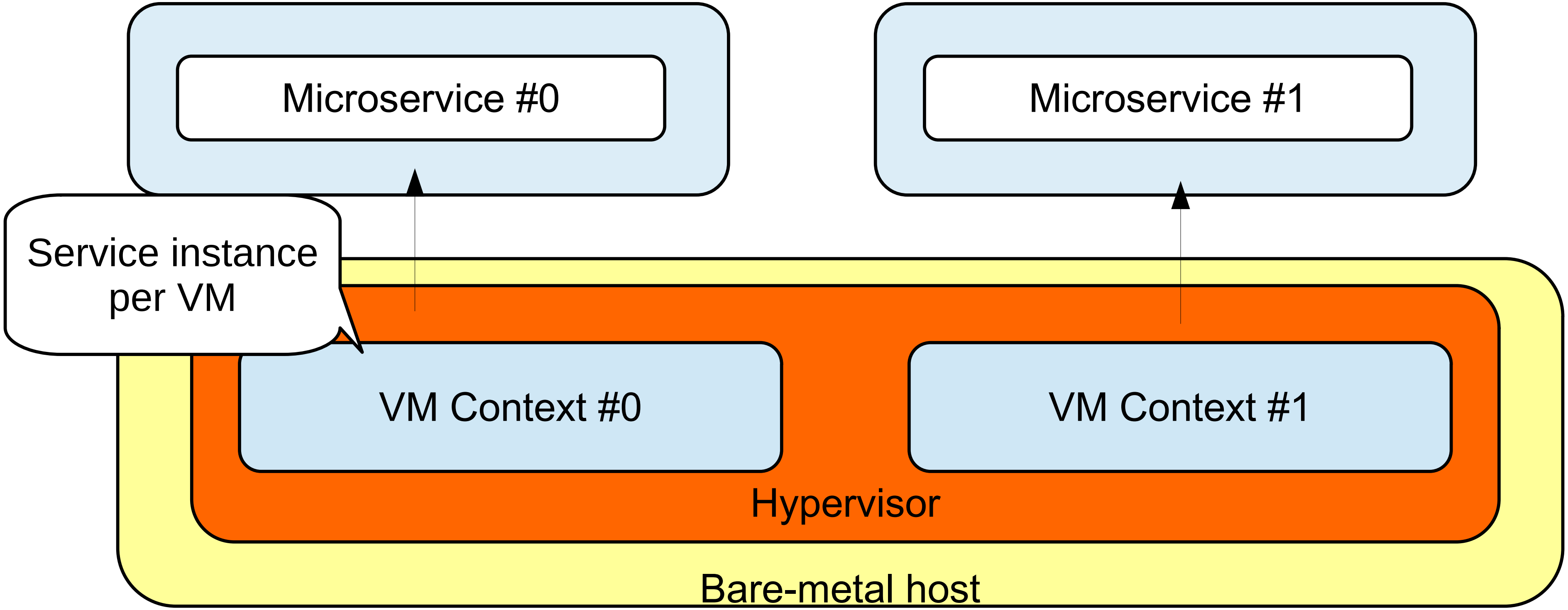
www.torokernel.io

Matias Vara Larsen
matiasevara@gmail.com

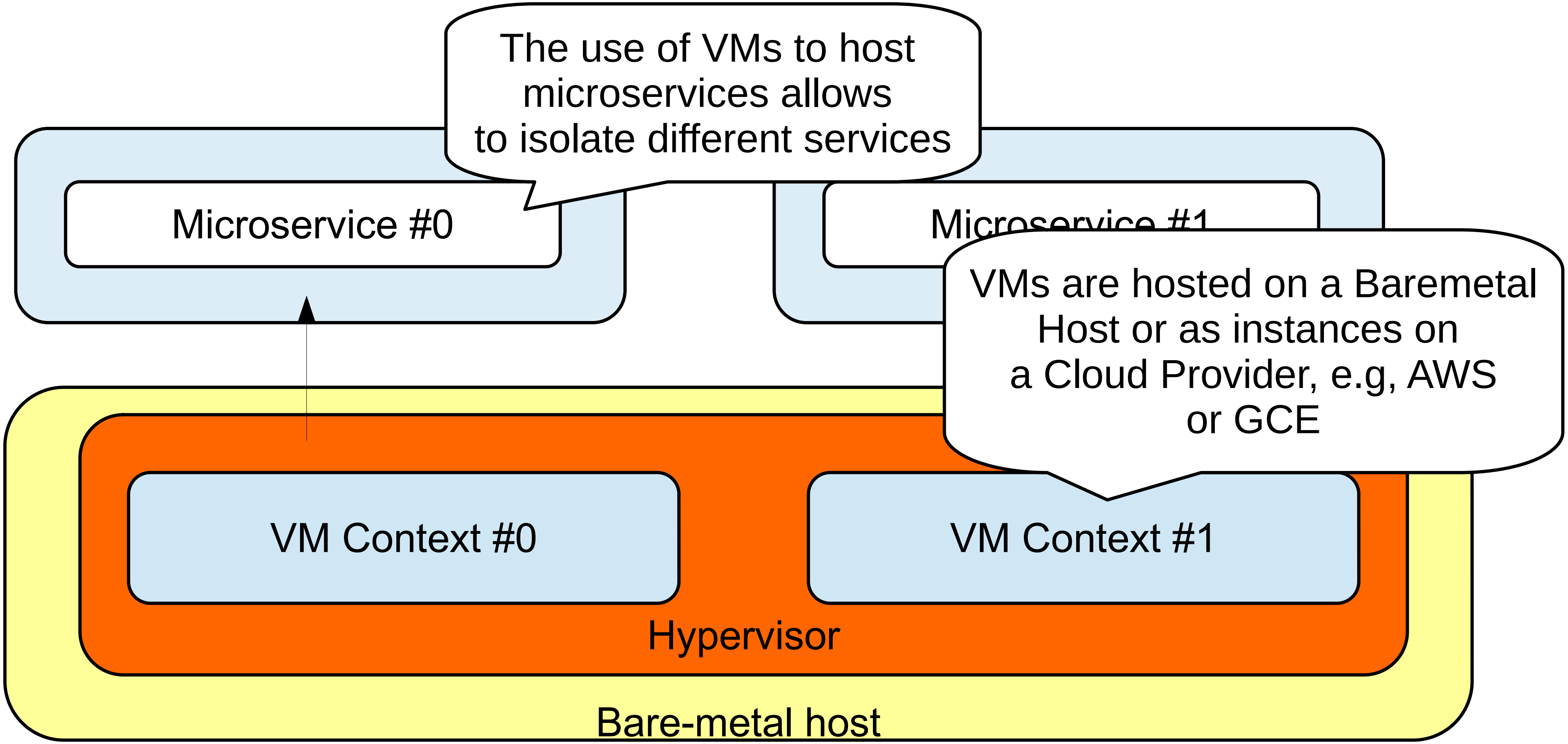
What are microservices?



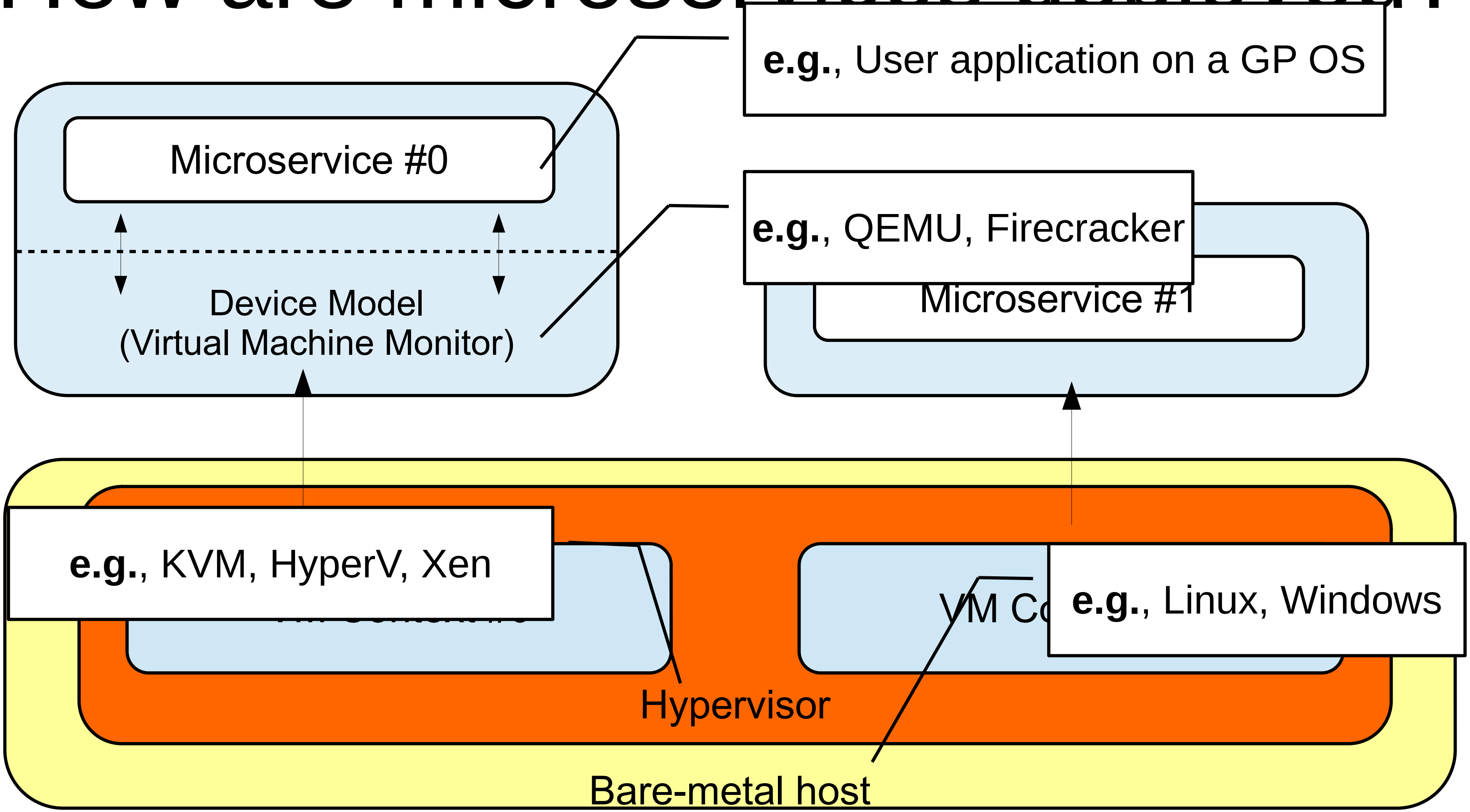
How are microservices deployed?

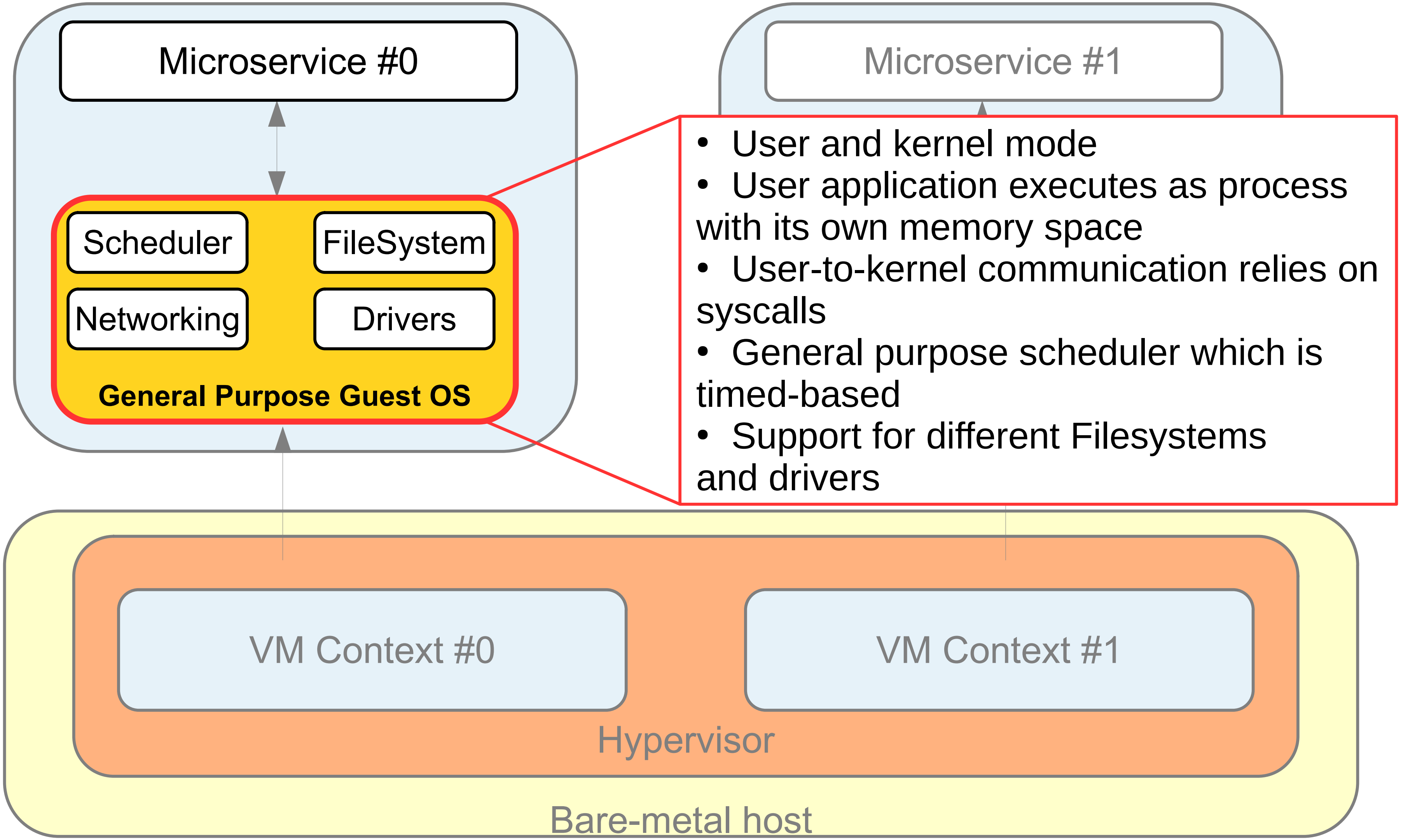


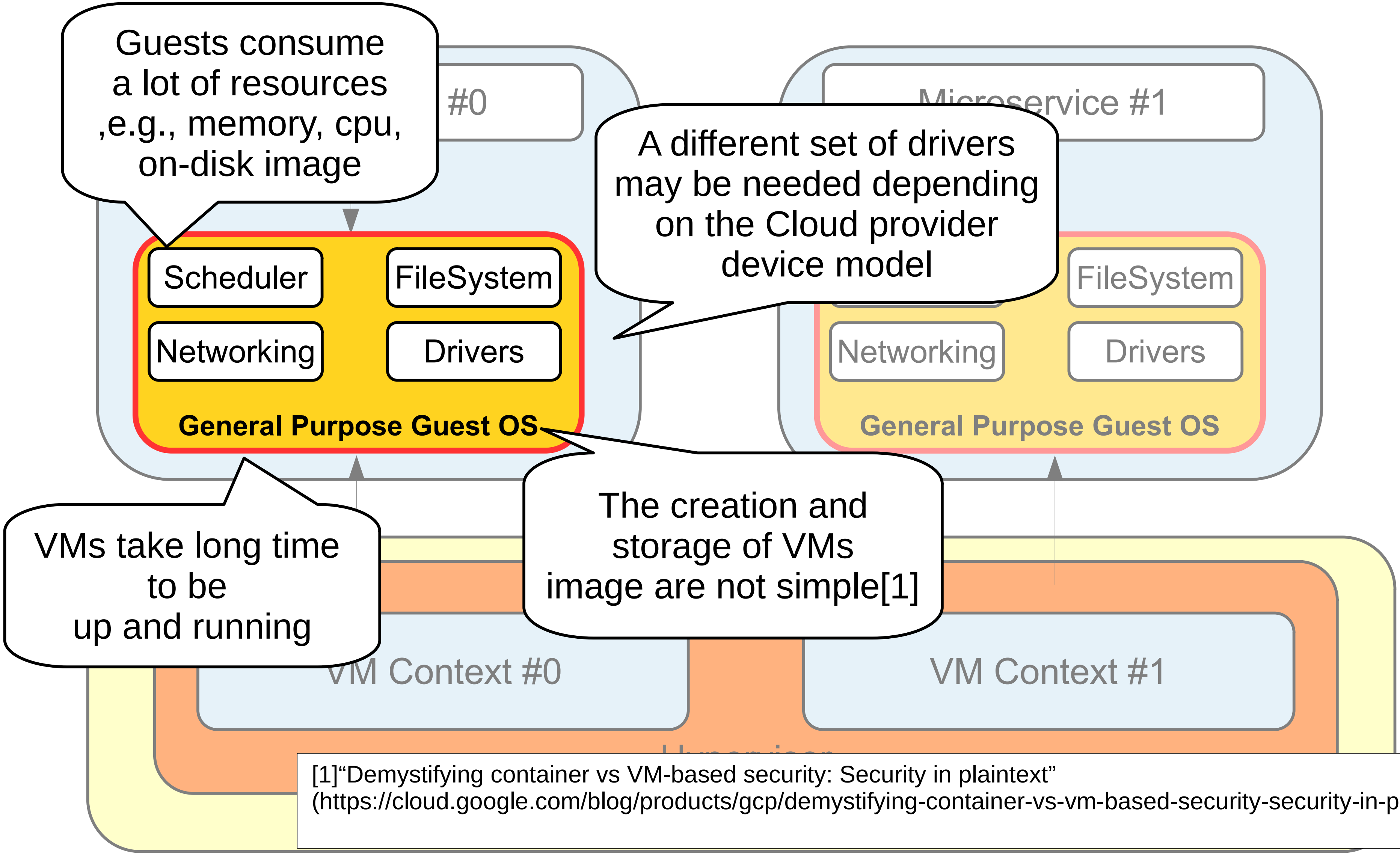
How are microservices deployed?



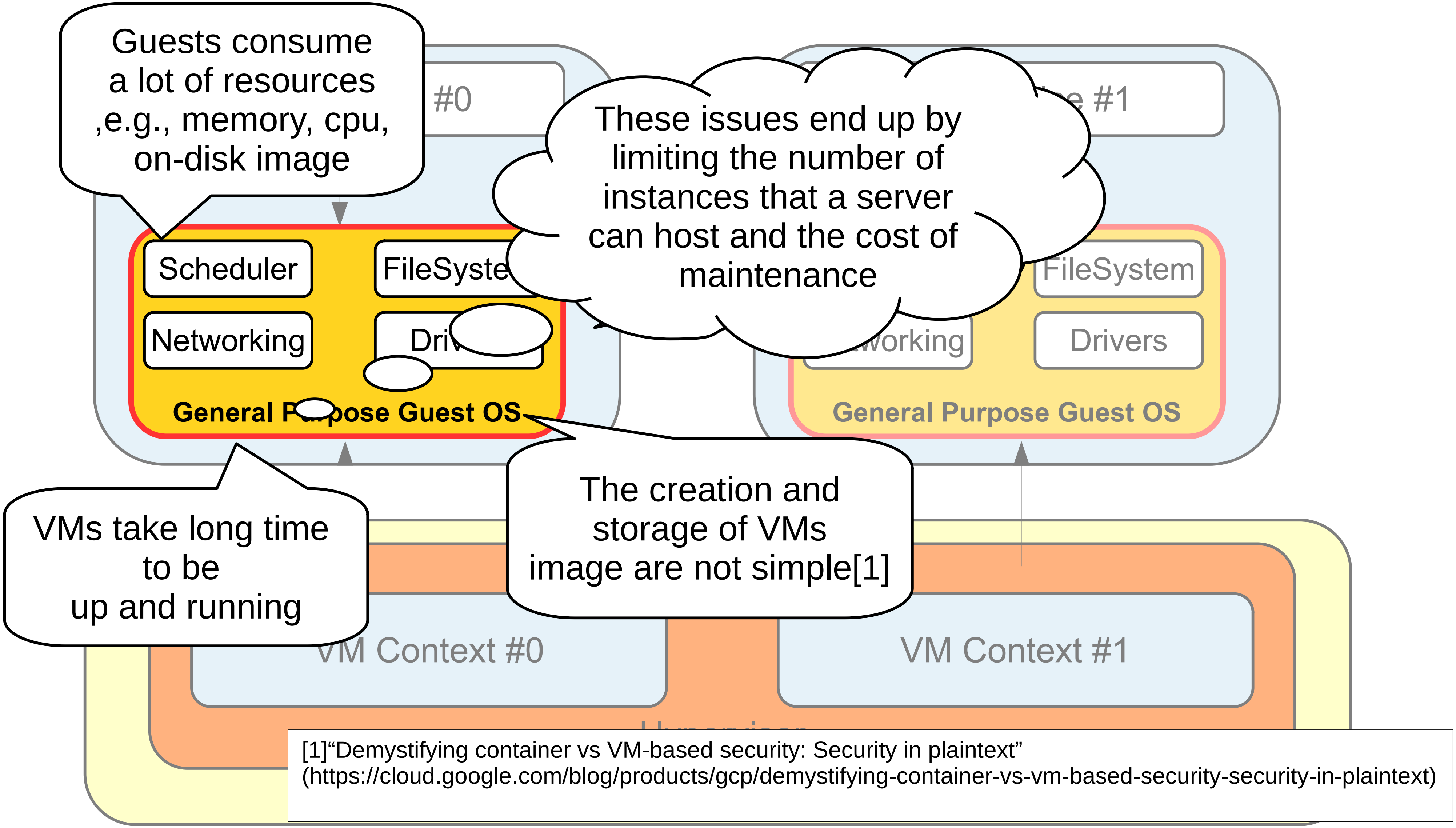
How are microservices deployed?



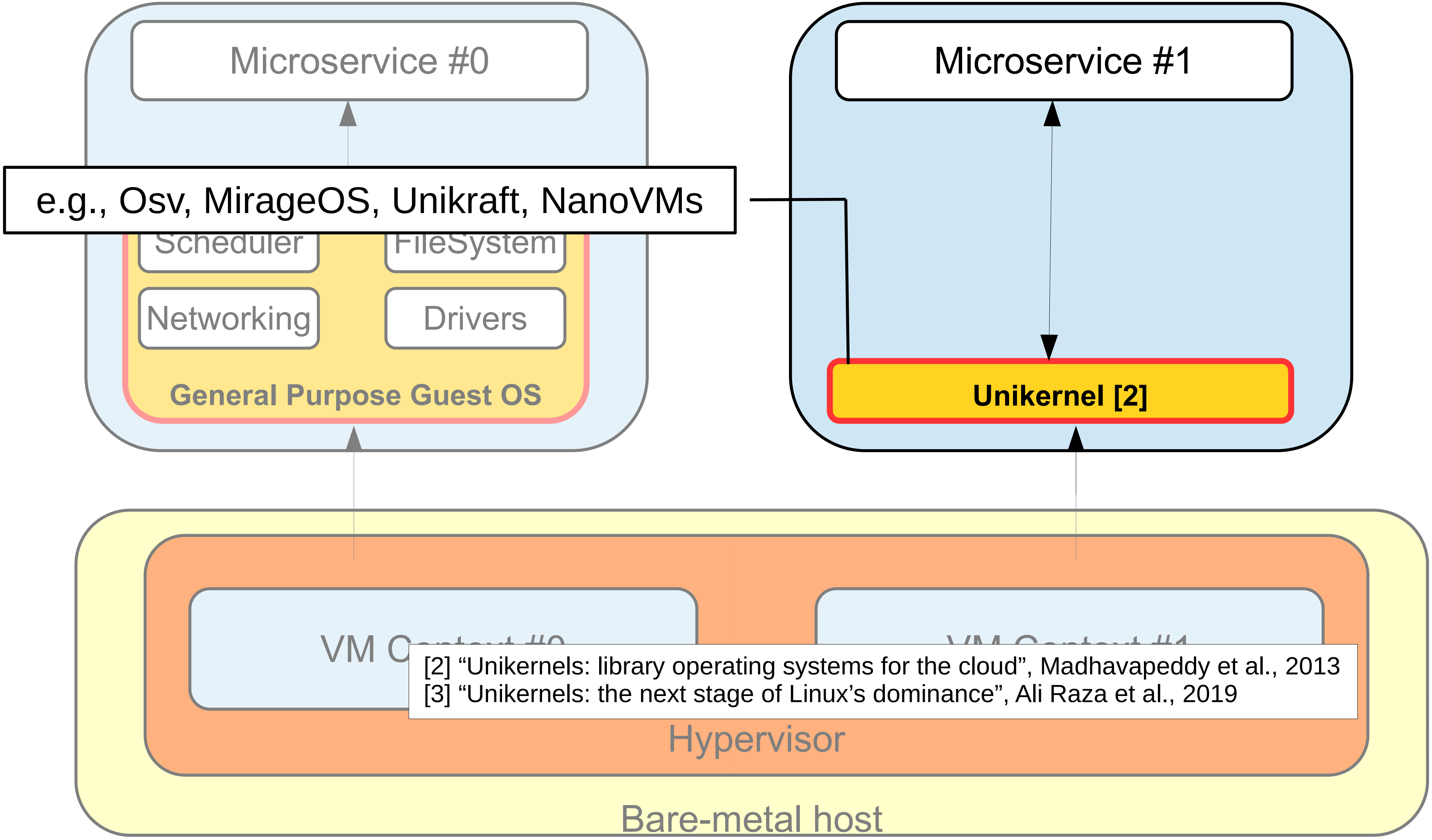

Microservice #0
Microservice #1
Scheduler
FileSystem
Networking
Drivers
General Purpose Guest OS
• User and kernel mode
• User application executes as process with its own memory space
• User-to-kernel communication relies on syscalls
• General purpose scheduler which is timed-based
• Support for different Filesystems and drivers
VM Context #0
VM Context #1
Hypervisor
Bare-metal host

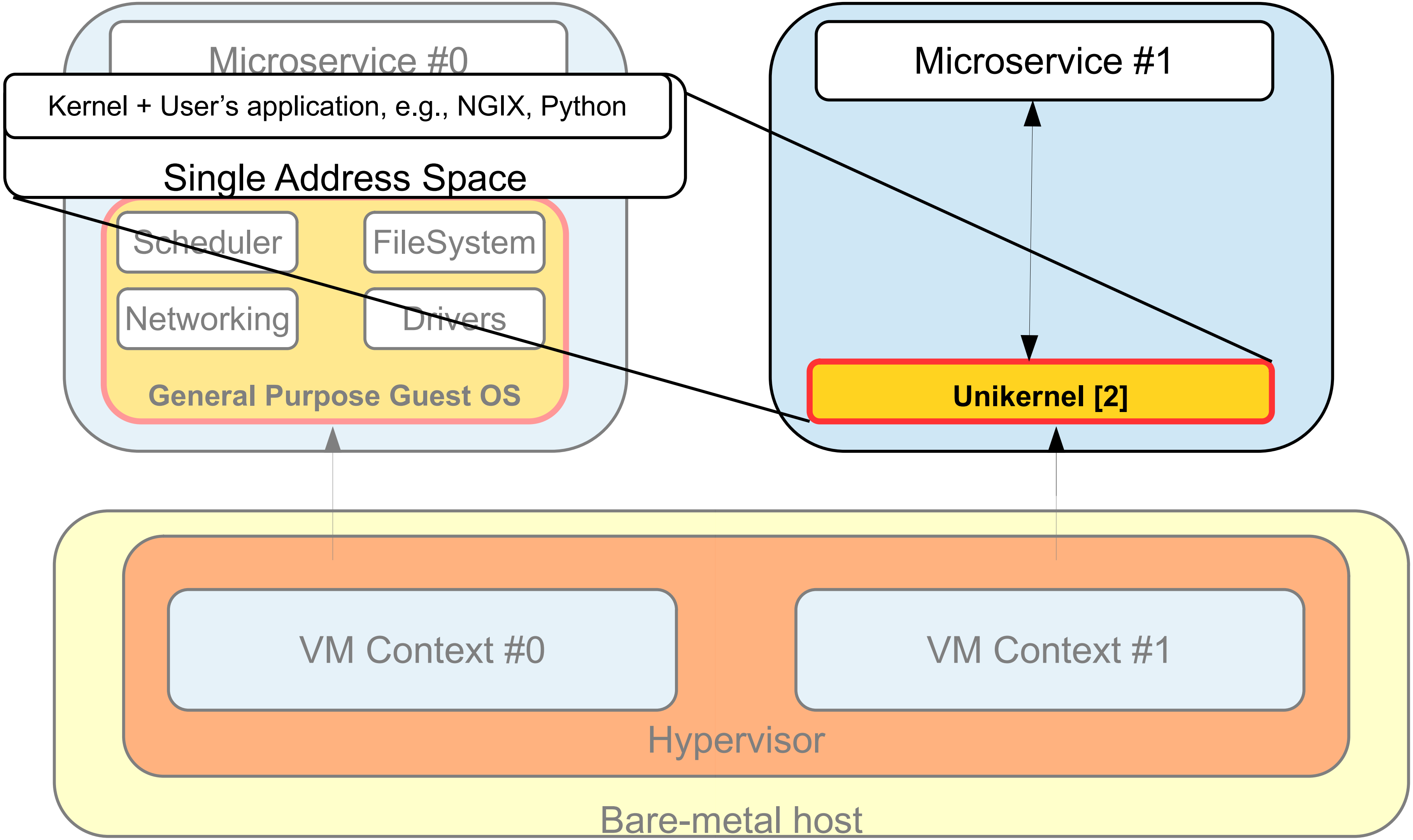


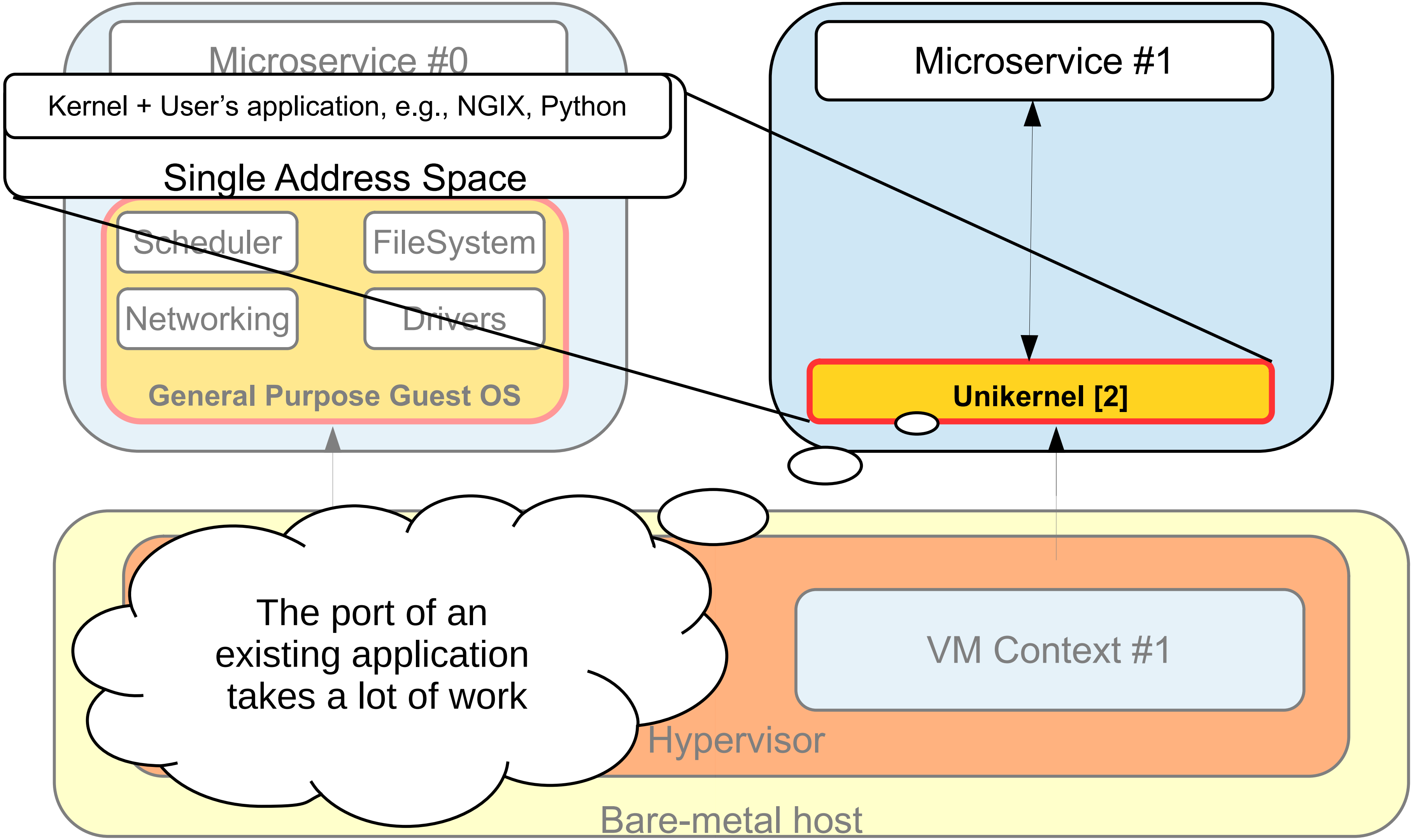
[1]“Demystifying container vs VM-based security: Security in plaintext” (https://cloud.google.com/blog/products/gcp/demystifying-container-vs-vm-based-security-security-in-plaintext)

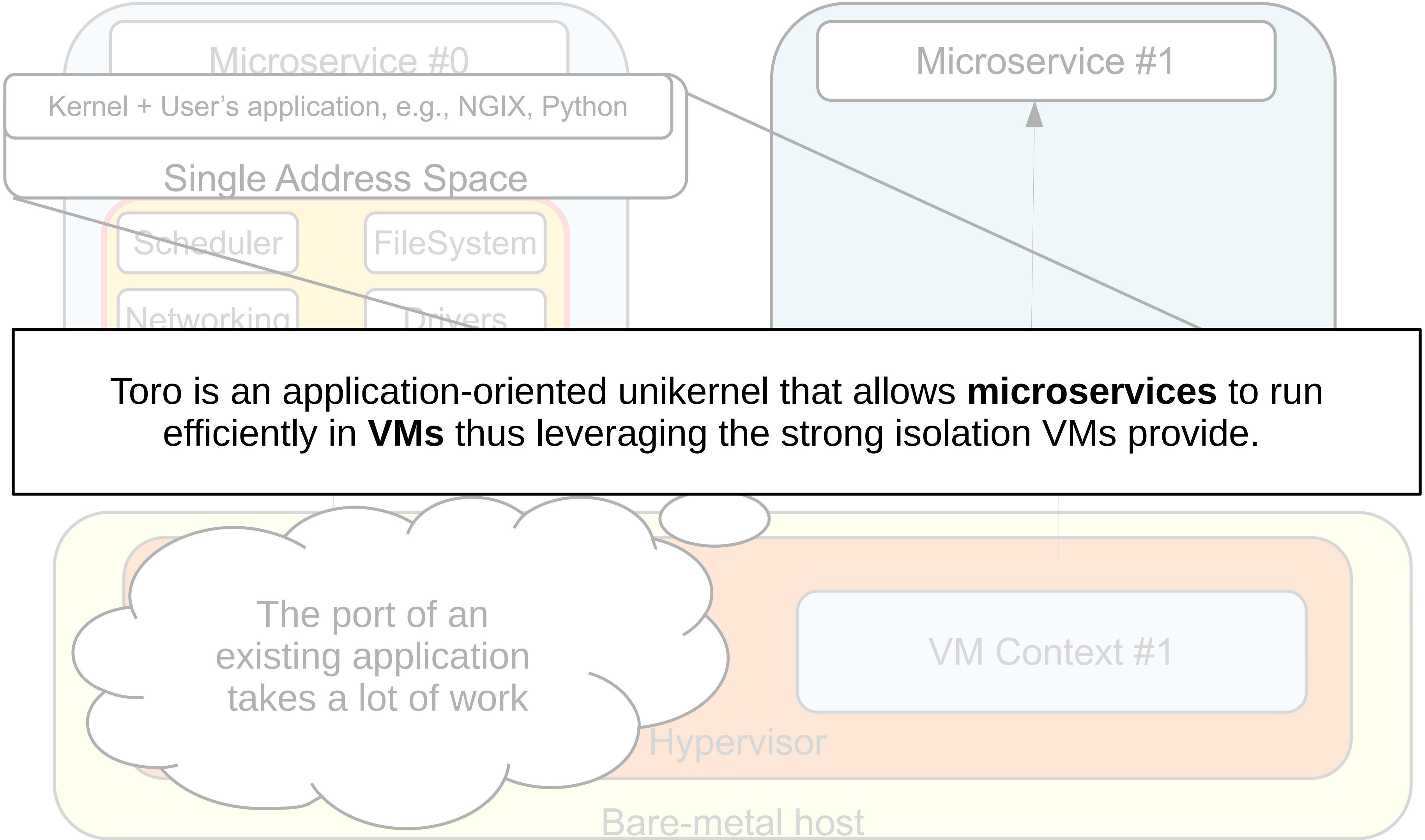


Guests consume a lot of resources ,e.g., memory, cpu, on-disk image
Scheduler
Networking
FileSystem
Drivers
General Purpose Guest OS
These issues end up by limiting the number of instances that a server can host and the cost of maintenance
VMs take long time to be up and running
The creation and storage of VMs image are not simple[1]
VM Context #1
[1]“Demystifying container vs VM-based security: Security in plaintext”
(https://cloud.google.com/blog/products/gcp/demystifying-container-vs-vm-based-security-security-in-plaintext)


Microservice #0
e.g., Osv, MirageOS, Unikraft, NanoVMs
Scheduler
FileSystem
Networking
Drivers
General Purpose Guest OS
Microservice #1
Unikernel [2]
[2] “Unikernels: library operating systems for the cloud”, Madhavapeddy et al., 2013
[3] “Unikernels: the next stage of Linux’s dominance”, Ali Raza et al., 2019
Hypervisor
Bare-metal host

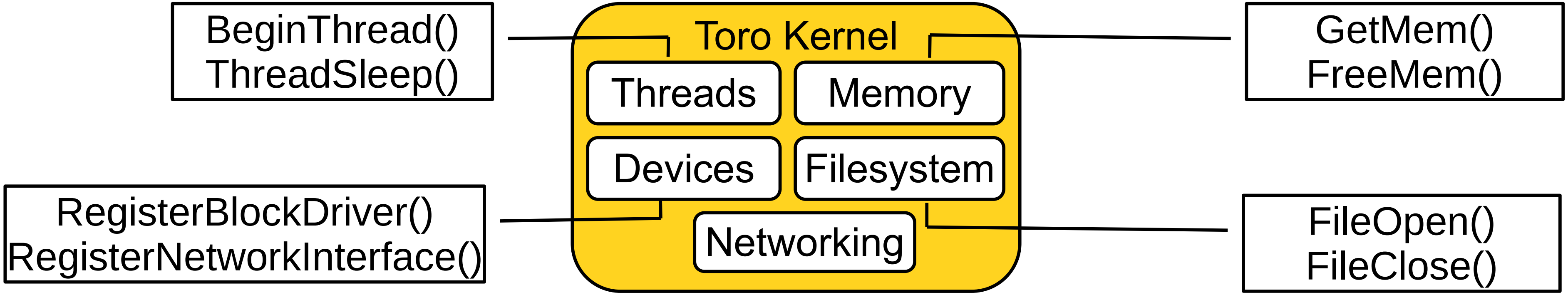

Microservice #0
Kernel + User's application, e.g., NGIX, Python
Single Address Space
Scheduler
FileSystem
Networking
Drivers
General Purpose Guest OS
Microservice #1
Unikernel [2]
VM Context #0
VM Context #1
Hypervisor
Bare-metal host


Microservice #0
Kernel + User's application, e.g., NGIX, Python
Single Address Space
Scheduler
FileSystem
Networking
Drivers
General Purpose Guest OS
Microservice #1
Unikernel [2]
The port of an existing application takes a lot of work
VM Context #1
Hypervisor
Bare-metal host



Toro is an application-oriented unikernel that allows **microservices** to run efficiently in **VMs** thus leveraging the strong isolation VMs provide.

Application-oriented Kernel

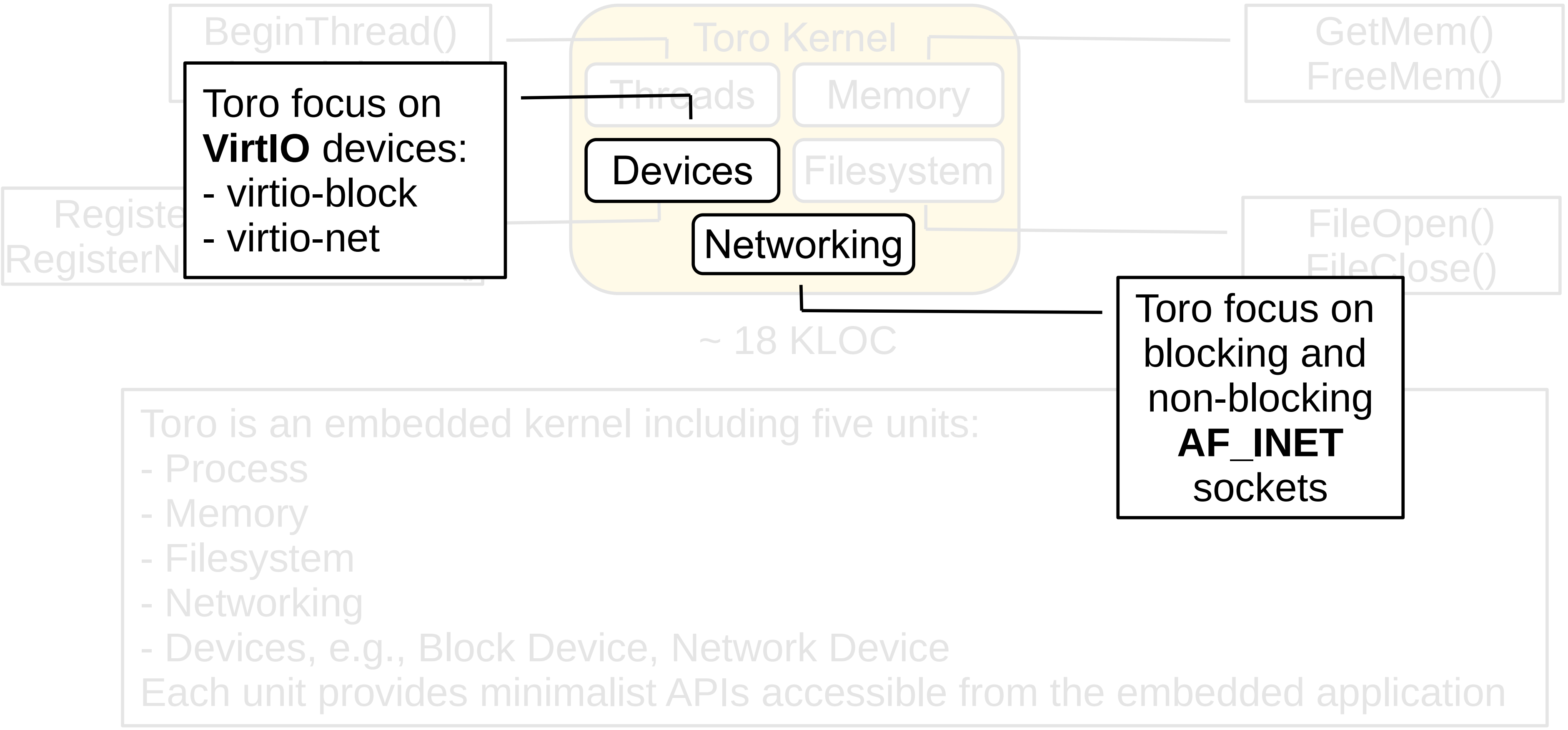


~ 18 KLOC

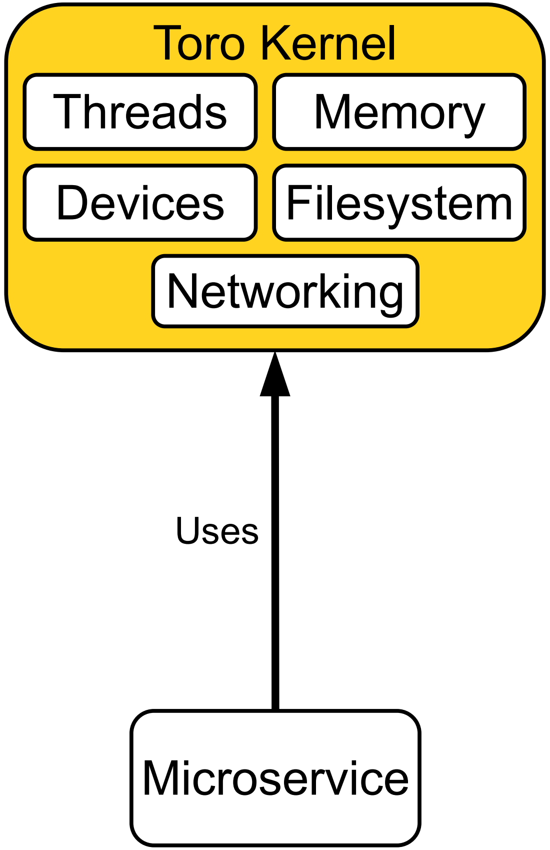
Toro is an embedded kernel including five units:
- Process
- Memory
- Filesystem
- Networking
- Devices, e.g., Block Device, Network Device

Each unit provides minimalist APIs accessible from the embedded application

Application-oriented Kernel

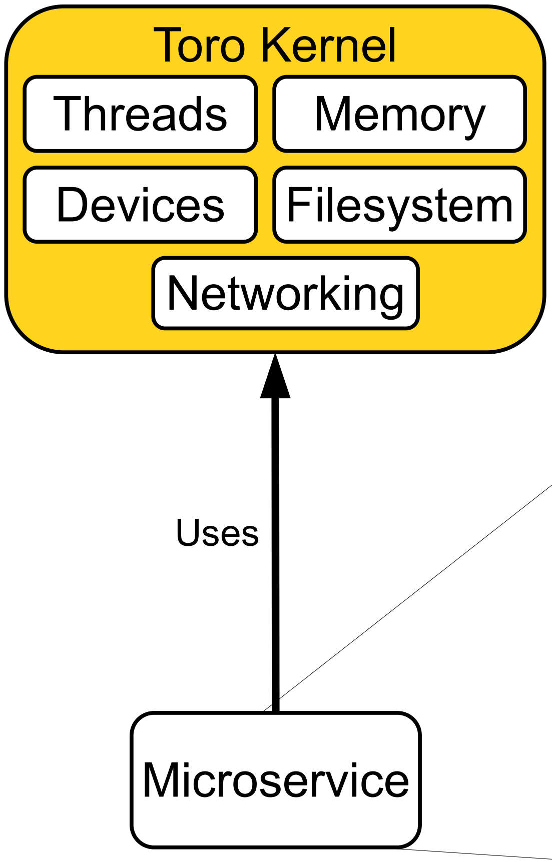


Application-oriented Kernel



- User application and kernel units are compiled in a single binary
- The application includes only the component required

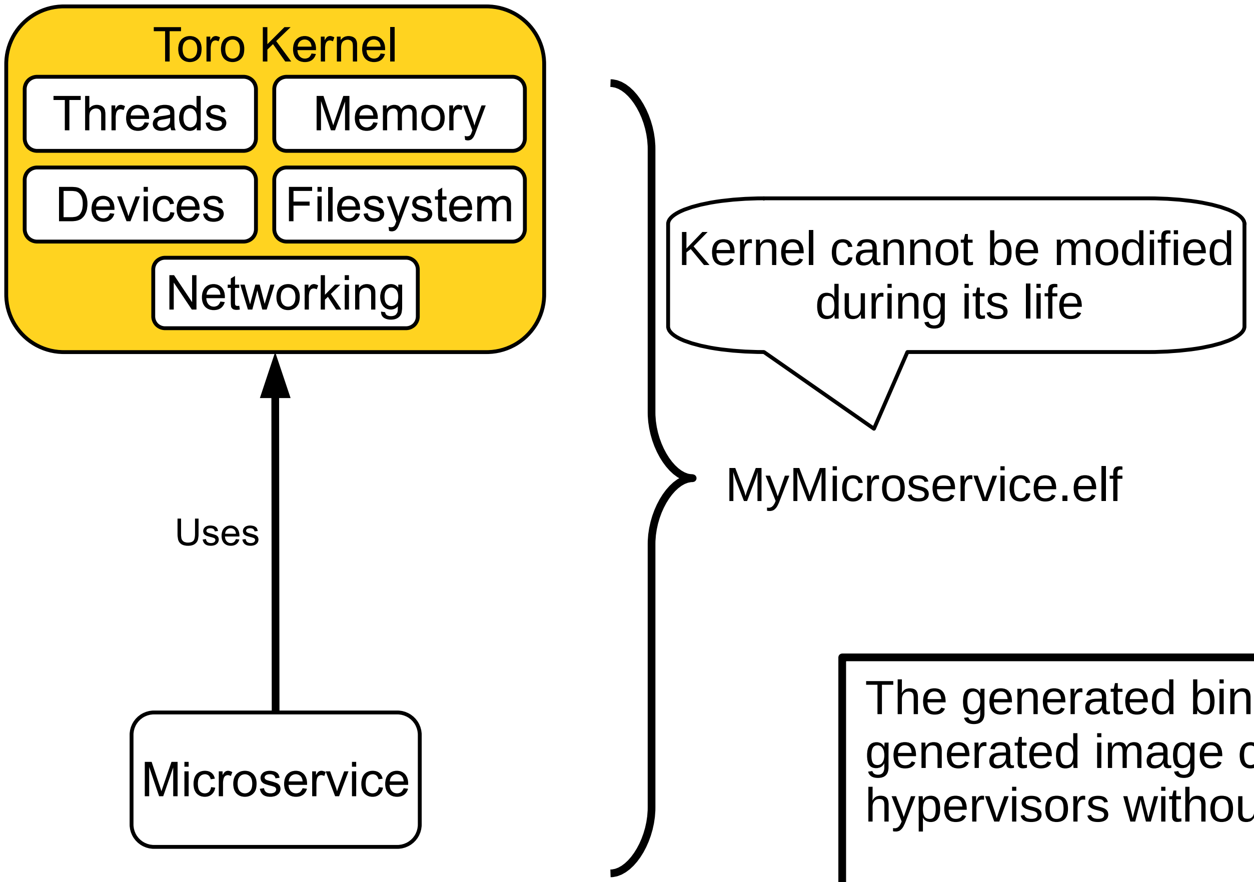
Application-oriented Kernel



- User application and kernel units are compiled in a single binary
- The
  com

```
program WebServerAppliance;
uses
  Memory,
  Filesystem,
  Threads,
  Networking,
  Fat,
  Virtio-blk,
  Virtio-net;
Begin
//
// Your Code Goes Here
//
End.
```

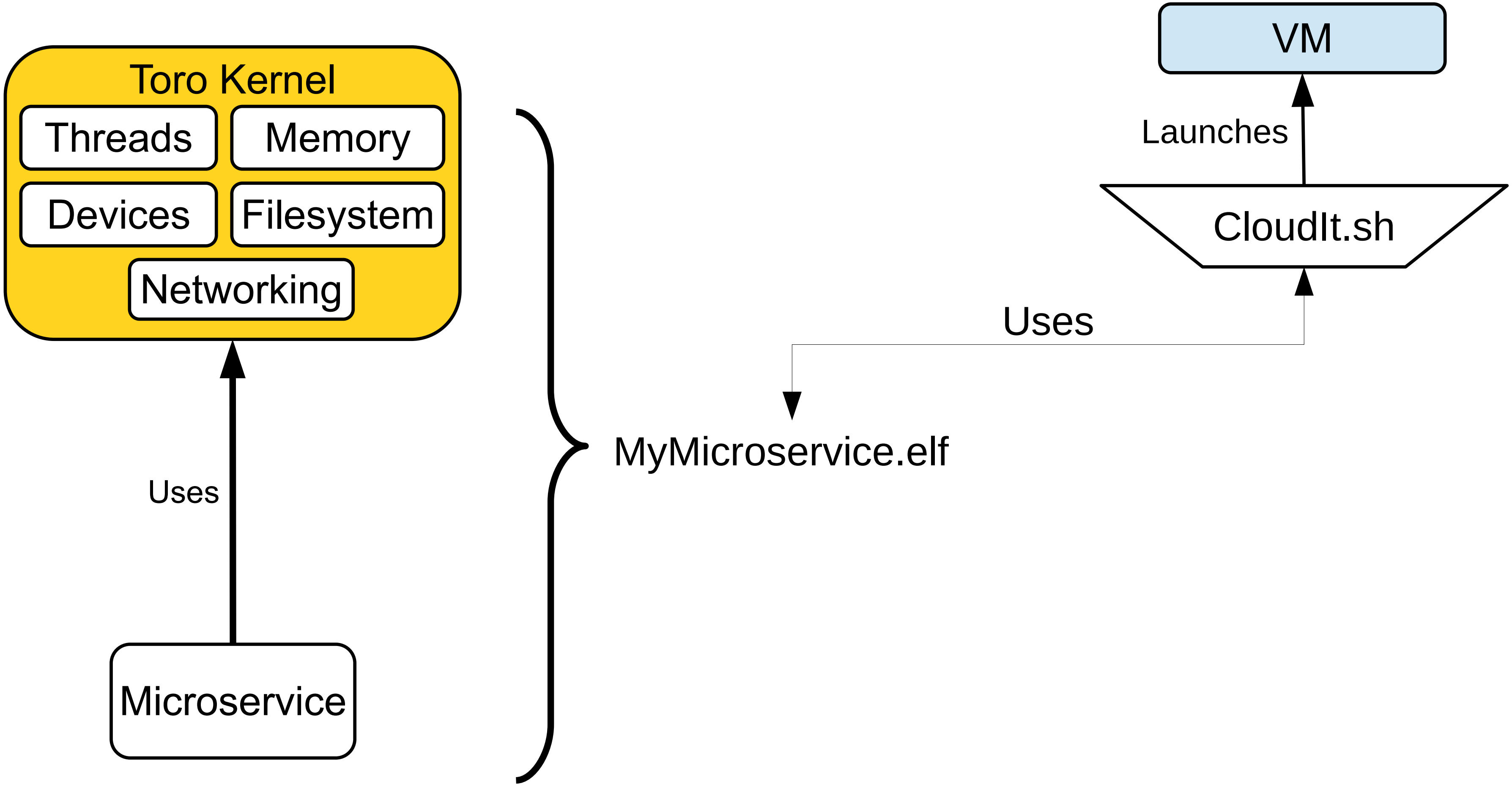
Application-oriented Kernel



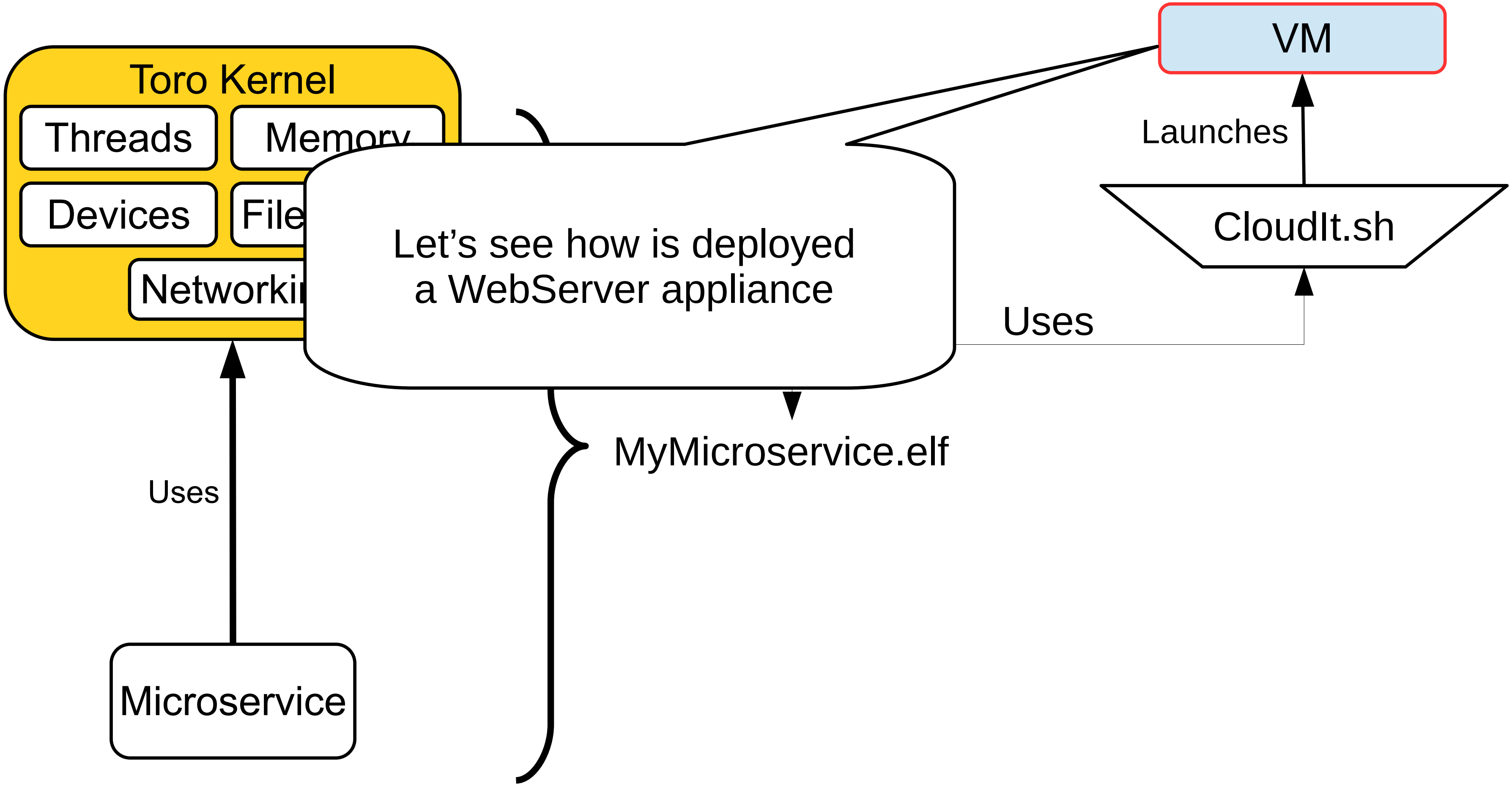
[1]“Immutable Infrastructure”, Stella

The generated binary is **Immutable[1]**, i.e., the generated image can be used across different hypervisors without the need to recompile it.

Application-oriented Kernel



Application-oriented Kernel

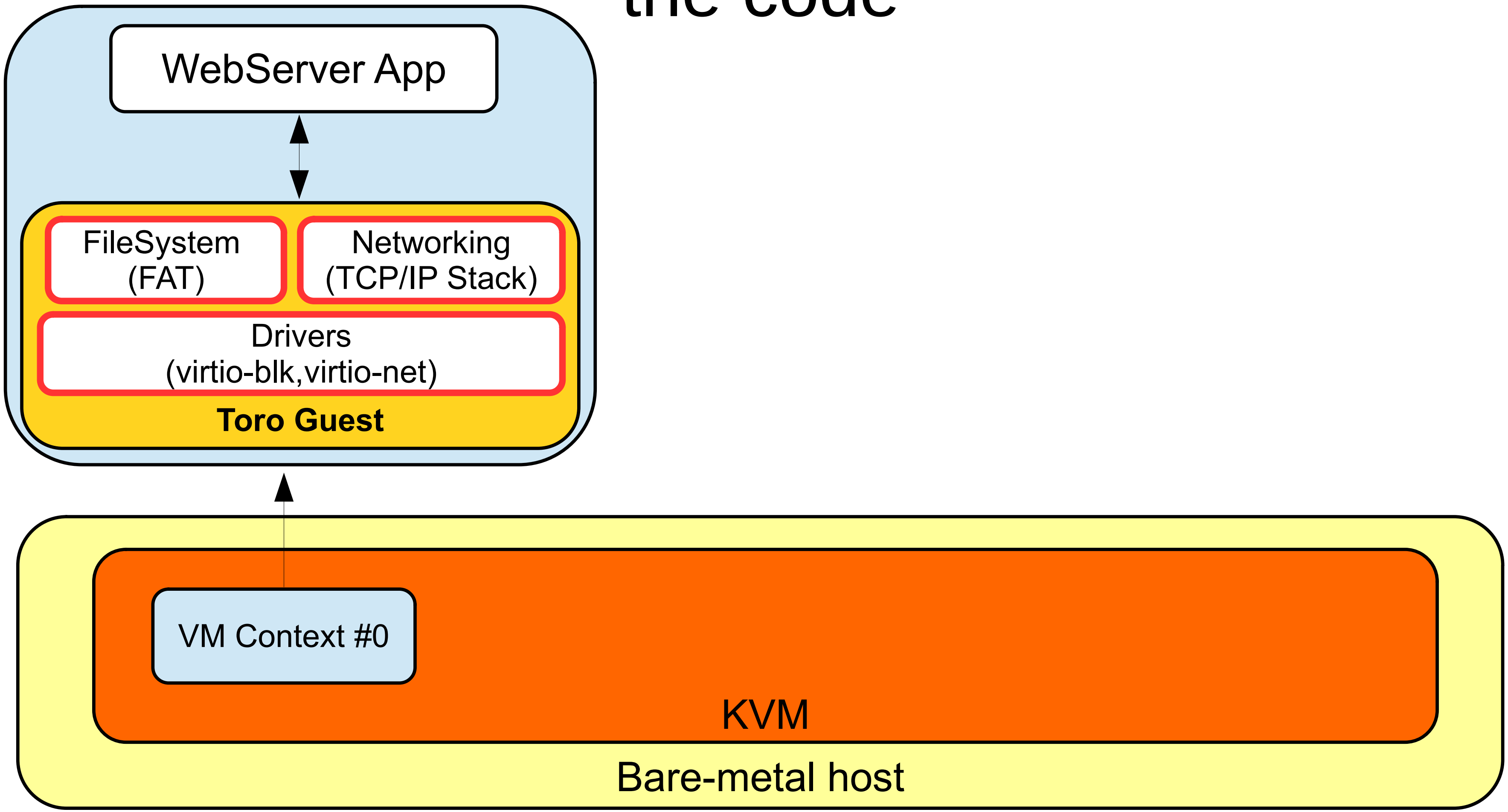


The WebServer Appliance

- Simple microservice that serves files by using the HTTP protocol
  - Find it at https://github.com/torokernel/torokernel among other examples
  - This appliance is used to host Toro’s website (http://www.torokernel.io and click on “View on Toro”)

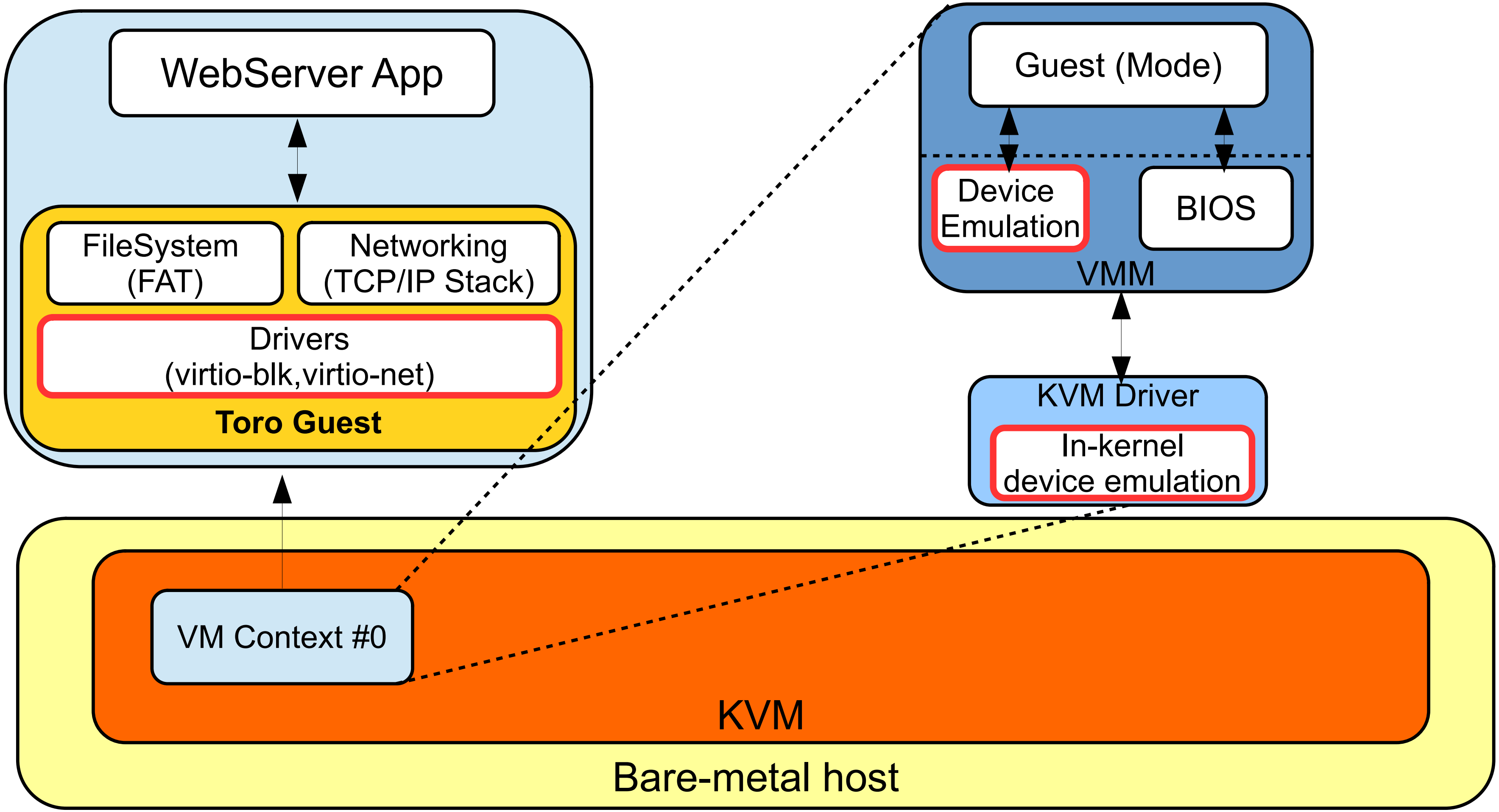
How the appliance is setup?

the code



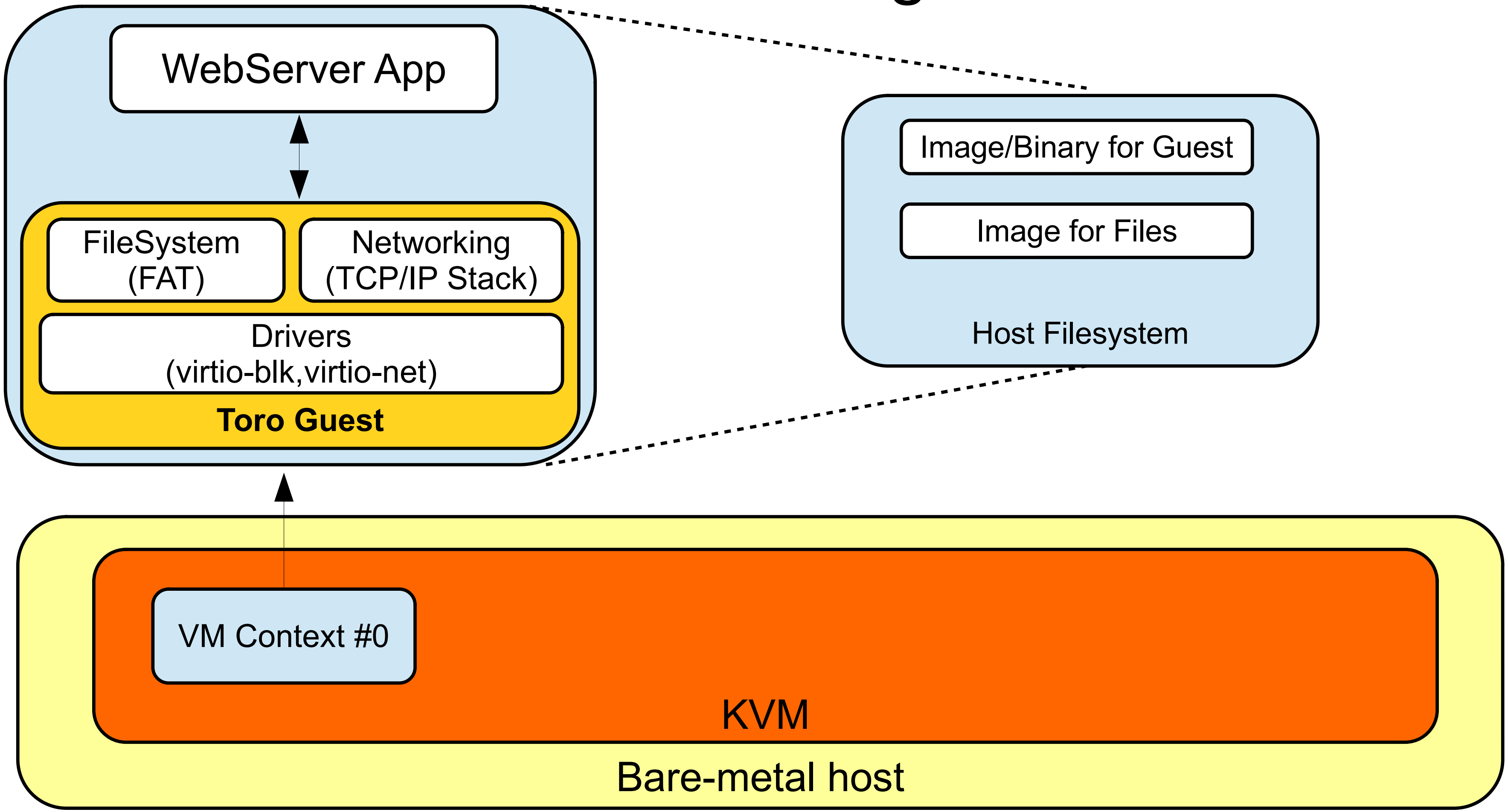
How the appliance is setup?

device model



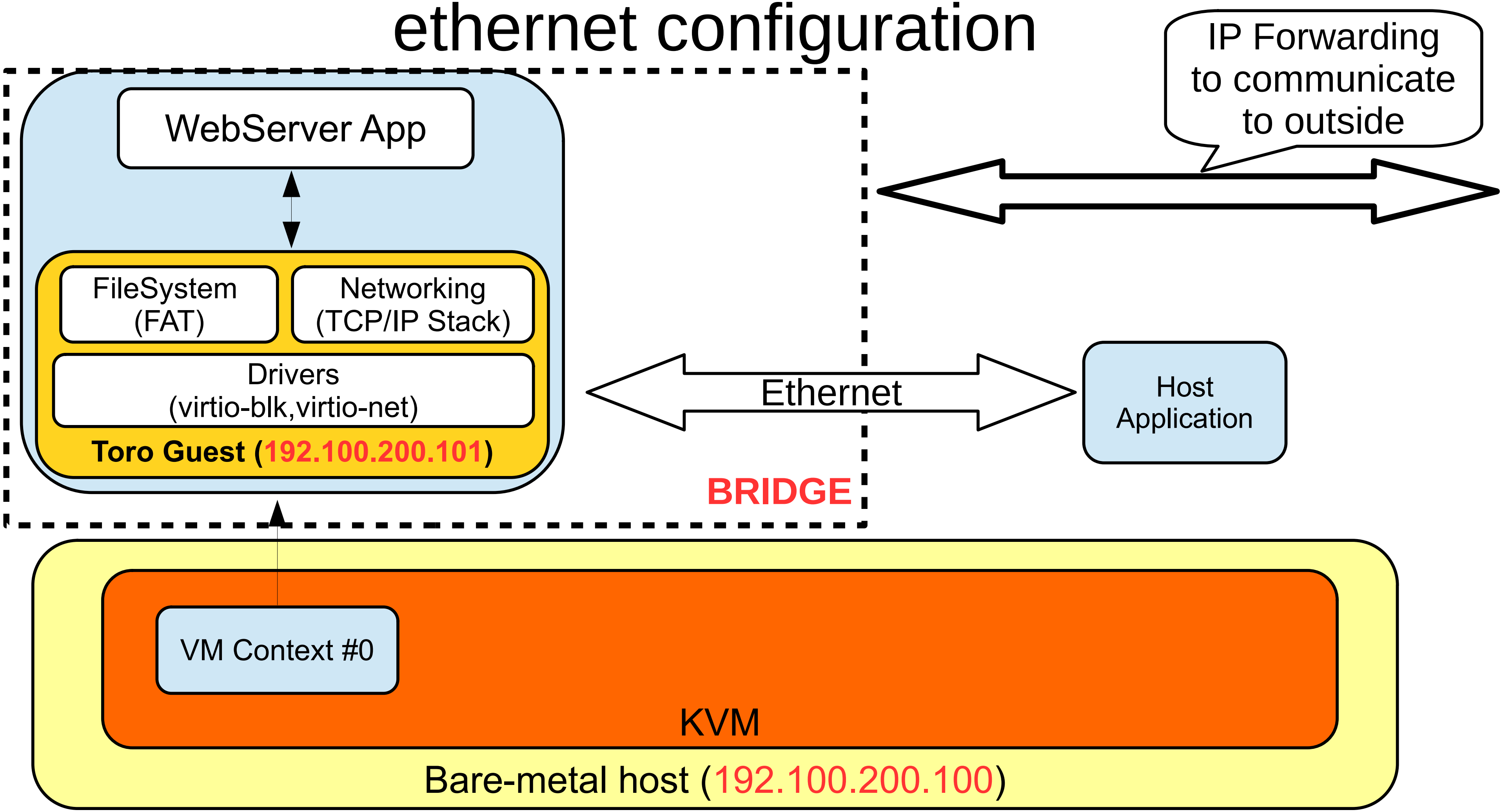
How the appliance is setup?

on-disk images



How the appliance is setup?

ethernet configuration



The Static WebServer
drawbacks

- Disk images consume memory and on-disk space, e.g., each guest has its own image
- Disk images have to be distributed in all the nodes
- The use of a TCP/IP stack requires configurations, e.g., bridge, an IP per guest, guest drivers, devices
- The use of more devices increases the attack surface
- Sharing of files between guests and host is hard
- Relying on a specific FS is not good for immutable images

The Static WebServer
drawbacks

- Disk images consume memory and on-disk space, e.g., each guest has its own image
- Disk images have to be distributed in all the nodes
- The use of a TCP/IP stack requires configurations, e.g., bridge, an IP per guest, guest drivers, devices
- The use of more devices increases the attack surface
- Sharing of files between guests and host is hard
- Relying on a specific FS is not good for imp

Can we do it better?!

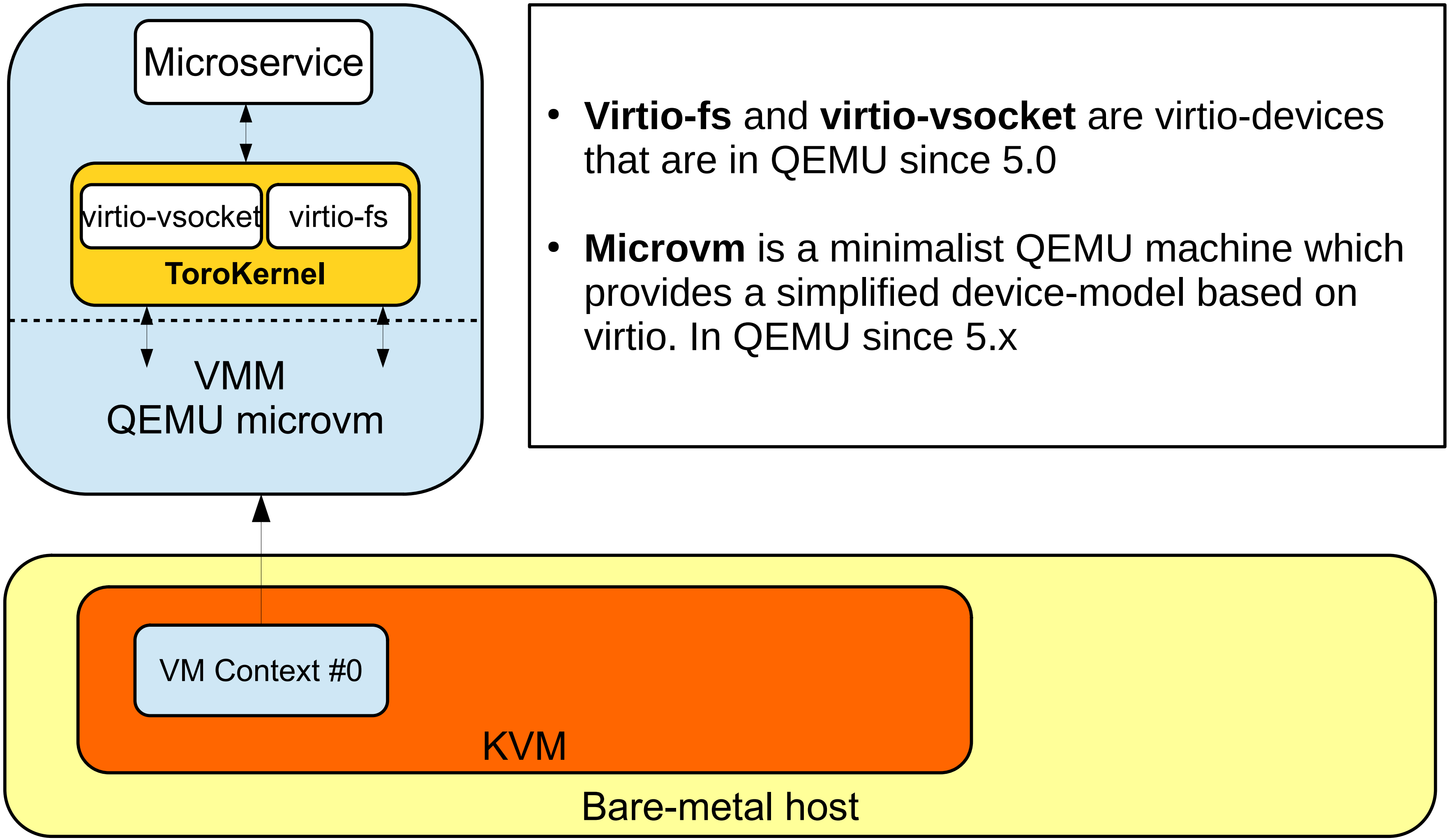
The Static WebServer drawbacks

- Disk images consume memory and on-disk space, e.g., each guest has its own image
- Disk images have to be distributed in all the nodes

We propose to use of **virtio-fs** for filesystem, **virtio-vsocket** for networking and **microvm as QEMU** machine to simplify toro unikernel’s code, reduce attack surface and ease appliance configuration. Also, we propose to use **CephFS** to provide a distributed FS among VMs.

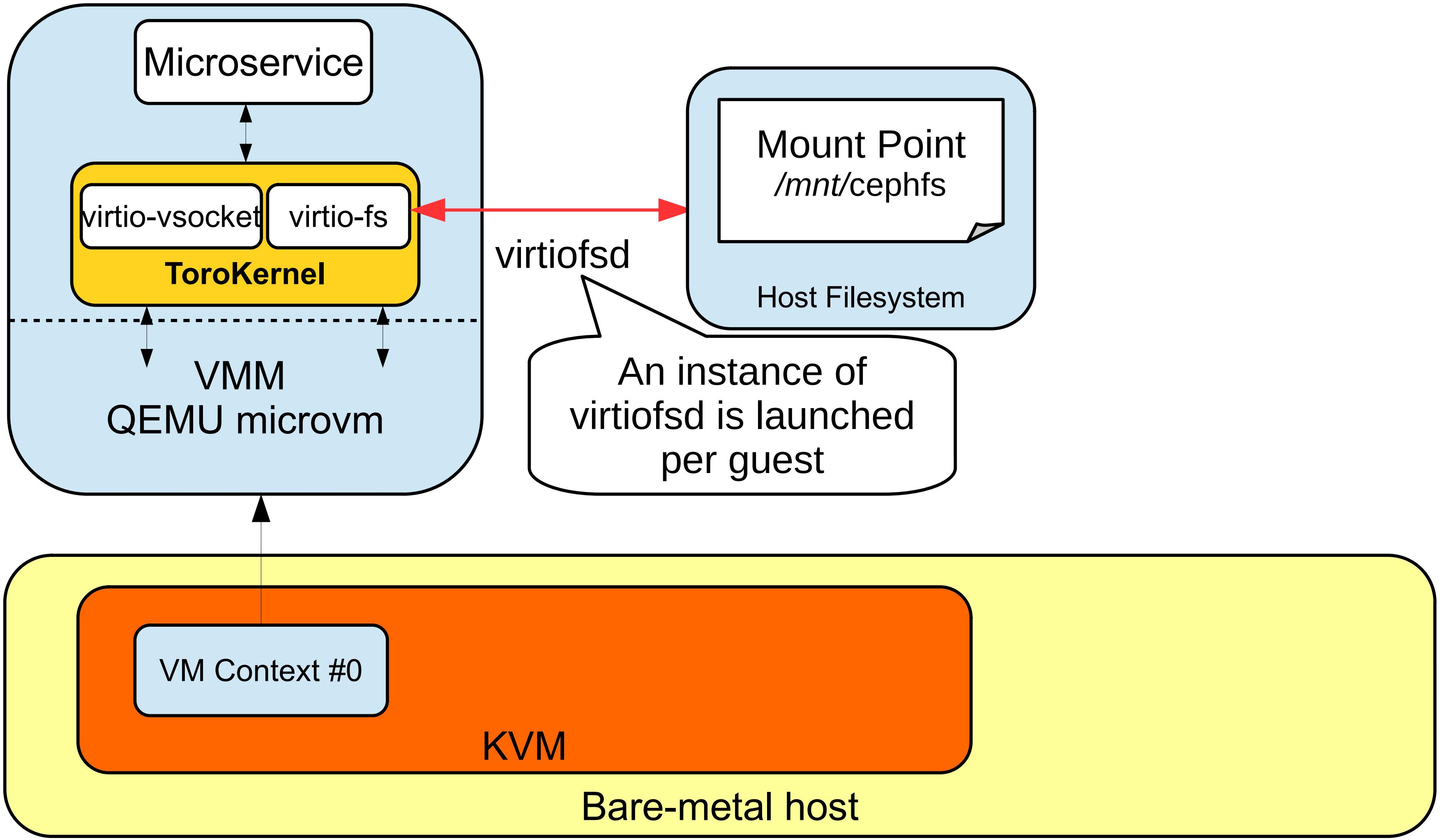
- The use of more devices increases the attack surface
- Sharing of files between guests and host is hard
- Relying on a specific FS is not good for im

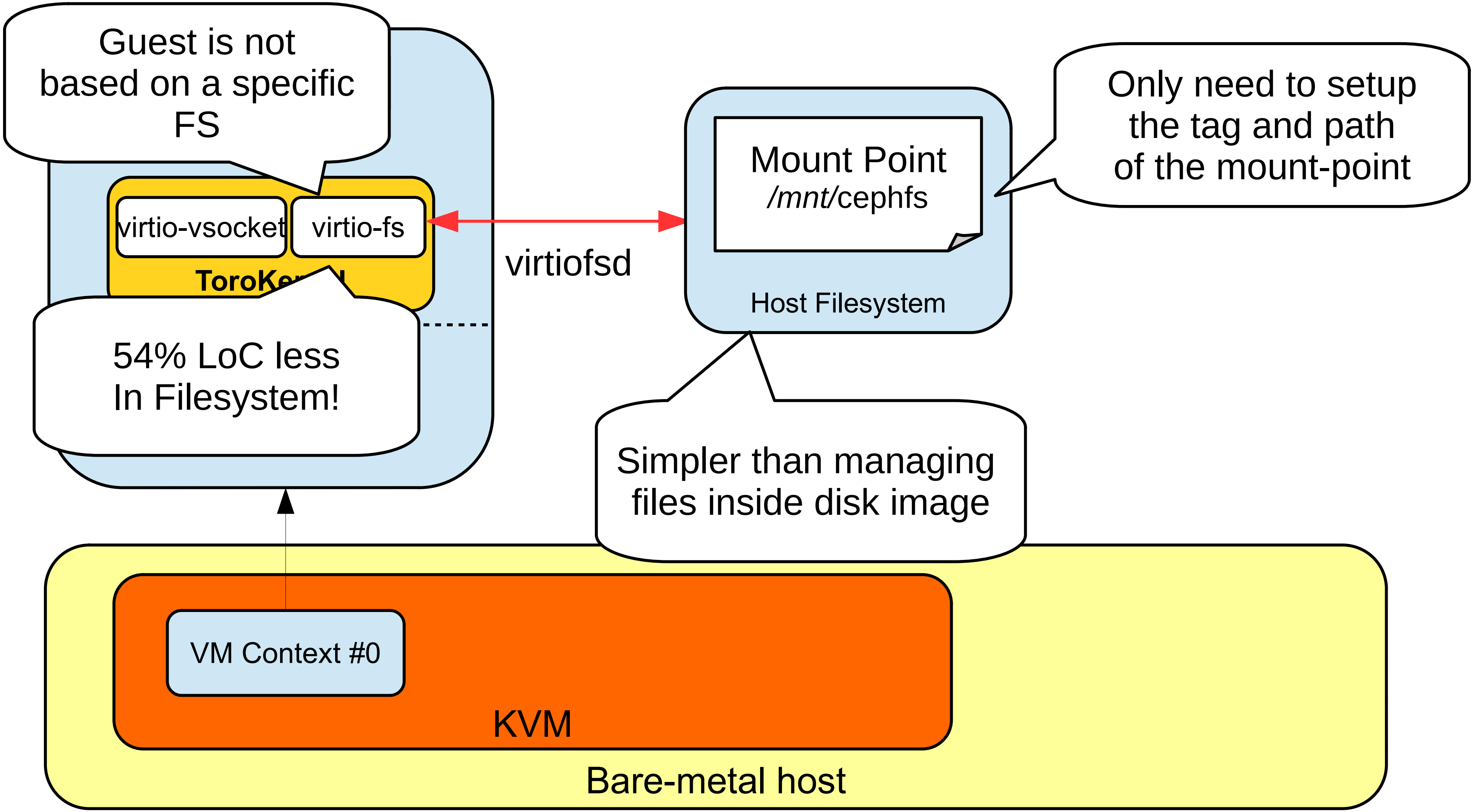
Can we do it better?!

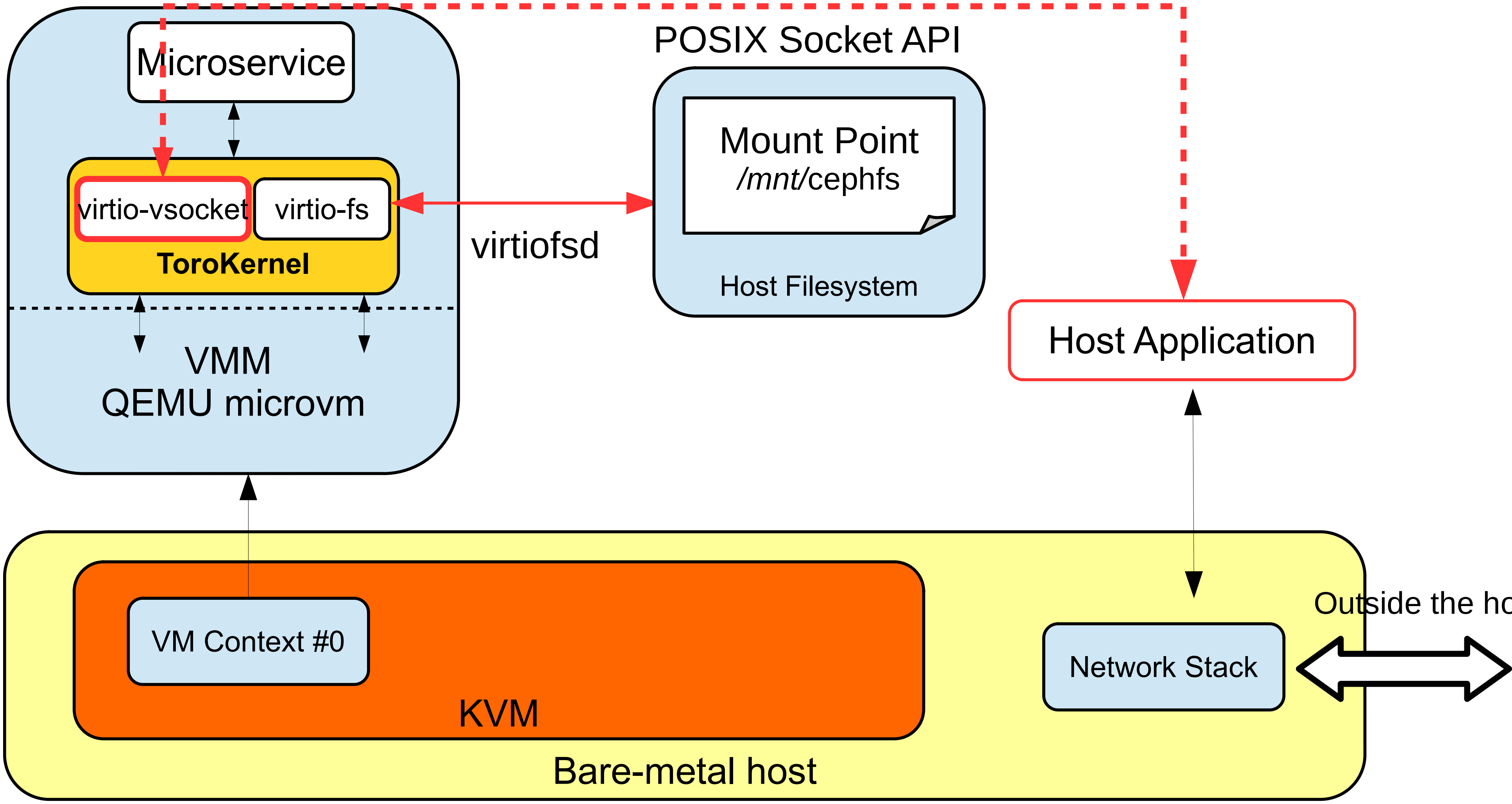


- **Virtio-fs** and **virtio-vsocket** are virtio-devices that are in QEMU since 5.0

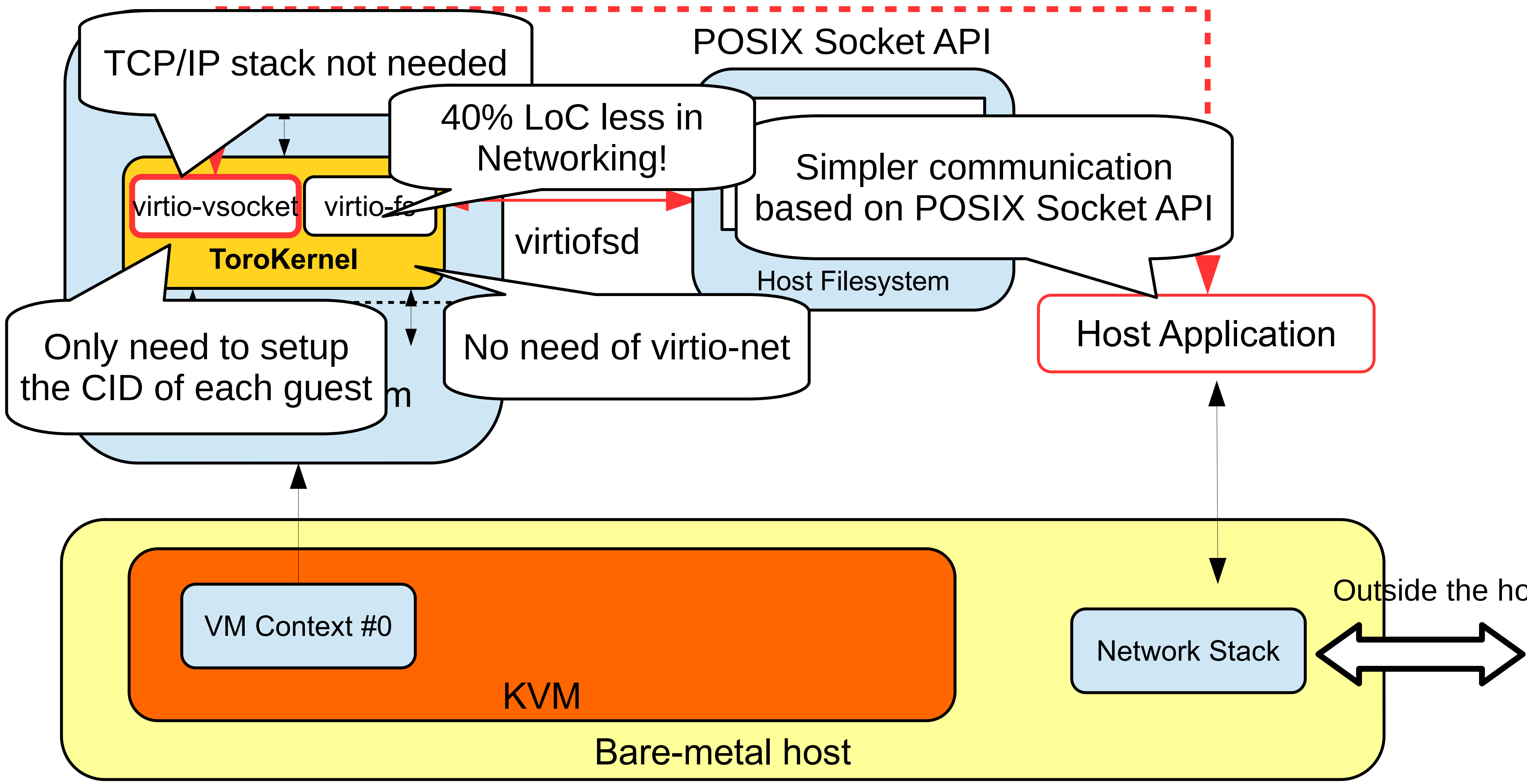
- **Microvm** is a minimalist QEMU machine which provides a simplified device-model based on virtio. In QEMU since 5.x


Microservice
virtio-vsocket
virtio-fs
ToroKernel
VMM
QEMU microvm
virtiofsd
Mount Point
/mnt/cephfs
Host Filesystem
An instance of virtiofsd is launched per guest
VM Context #0
KVM
Bare-metal host

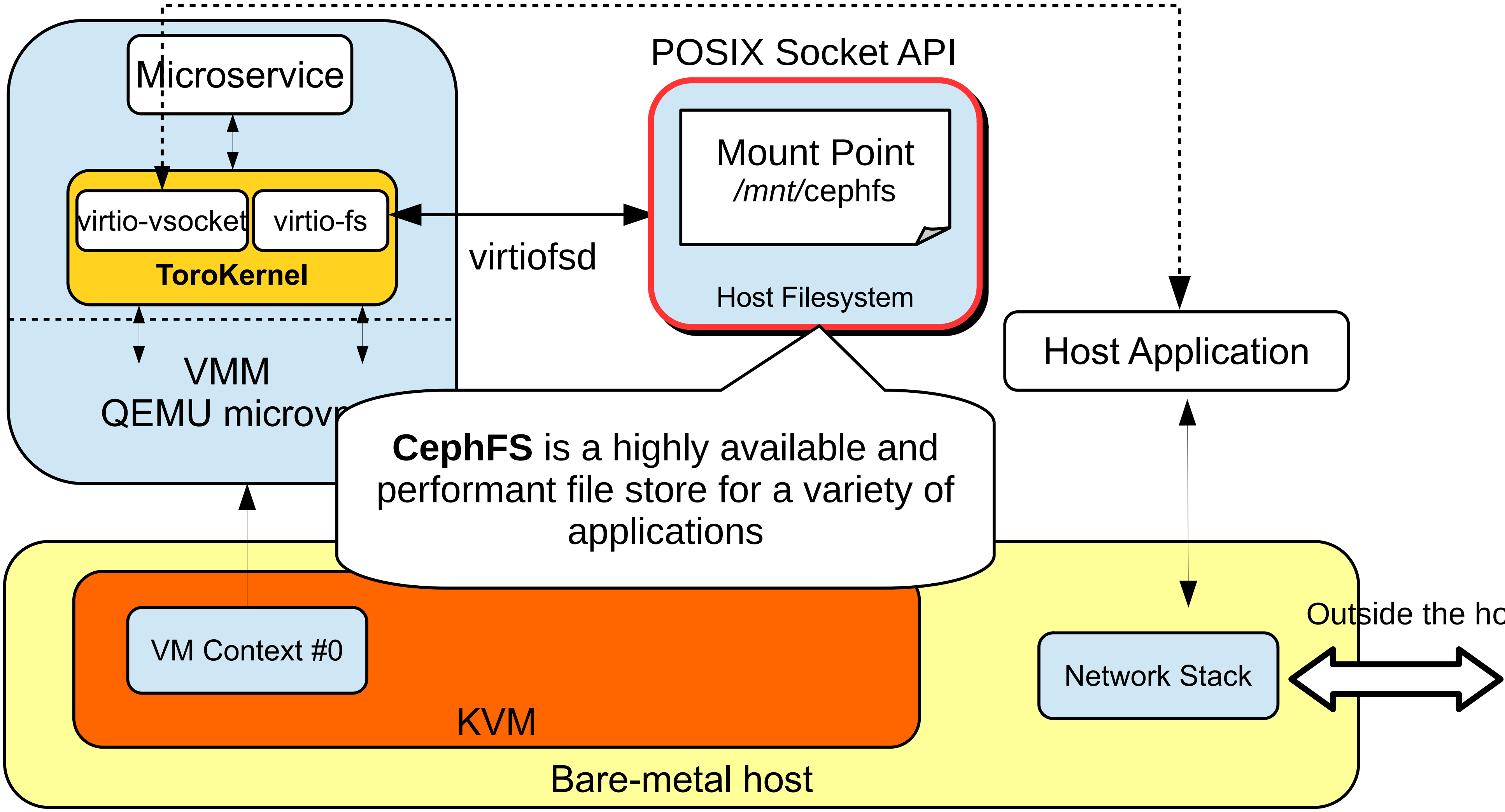

Guest is not based on a specific FS
virtio-vsocket
virtio-fs
ToroKe
54% LoC less In Filesystem!
virtiofsd
Mount Point
/mnt/cephfs
Host Filesystem
Only need to setup the tag and path of the mount-point
Simpler than managing files inside disk image
VM Context #0
KVM
Bare-metal host



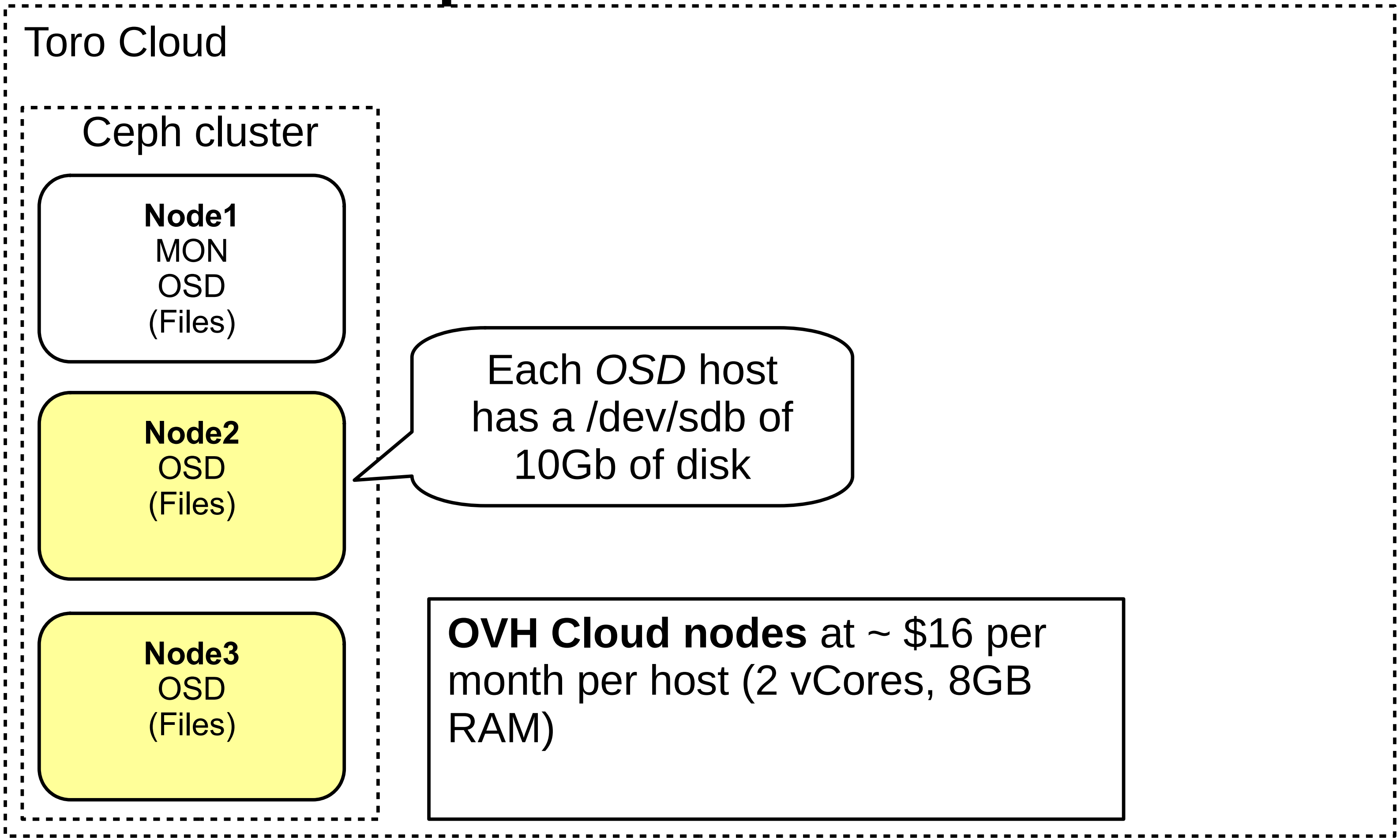
POSIX Socket API
Microservice
virtio-vsocket
virtio-fs
ToroKernel
virtiofsd
Mount Point
/mnt/cephfs
Host Filesystem
Host Application
VMM
QEMU microvm
VM Context #0
KVM
Network Stack
Outside the ho
Bare-metal host



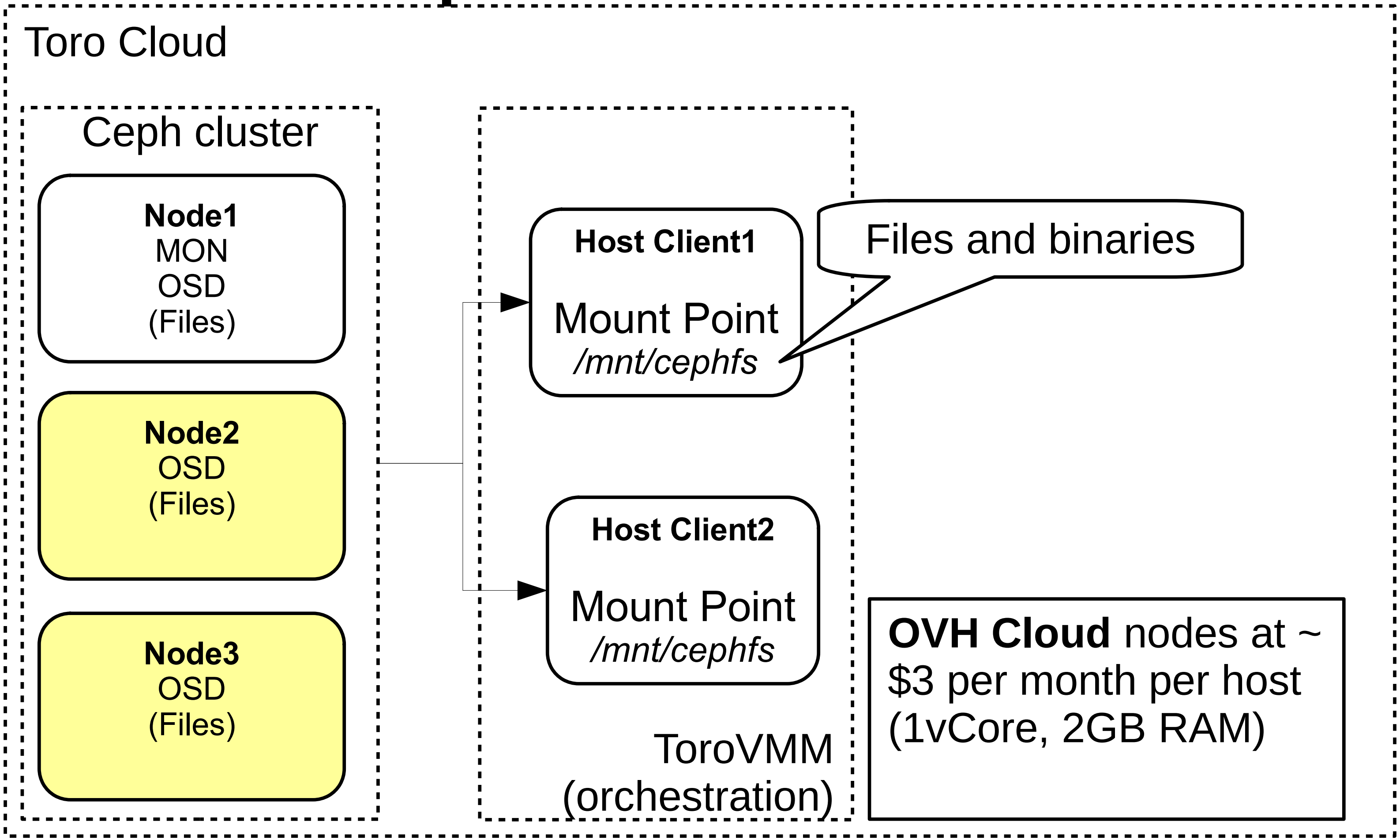
POSIX Socket API
TCP/IP stack not needed
40% LoC less in
Networking!
Simpler communication
based on POSIX Socket API
virtio-vsocket
virtio-fs
ToroKernel
virtiofsd
Host Filesystem
Host Application
Only need to setup
the CID of each guest
No need of virtio-net
VM Context #0
KVM
Network Stack
Outside the ho
Bare-metal host


POSIX Socket API
Microservice
virtio-vsocket
virtio-fs
ToroKernel
virtiofsd
Mount Point
/mnt/cephfs
Host Filesystem
Host Application
VMM
QEMU microvm
CephFS is a highly available and performant file store for a variety of applications
VM Context #0
KVM
Network Stack
Outside the ho
Bare-metal host

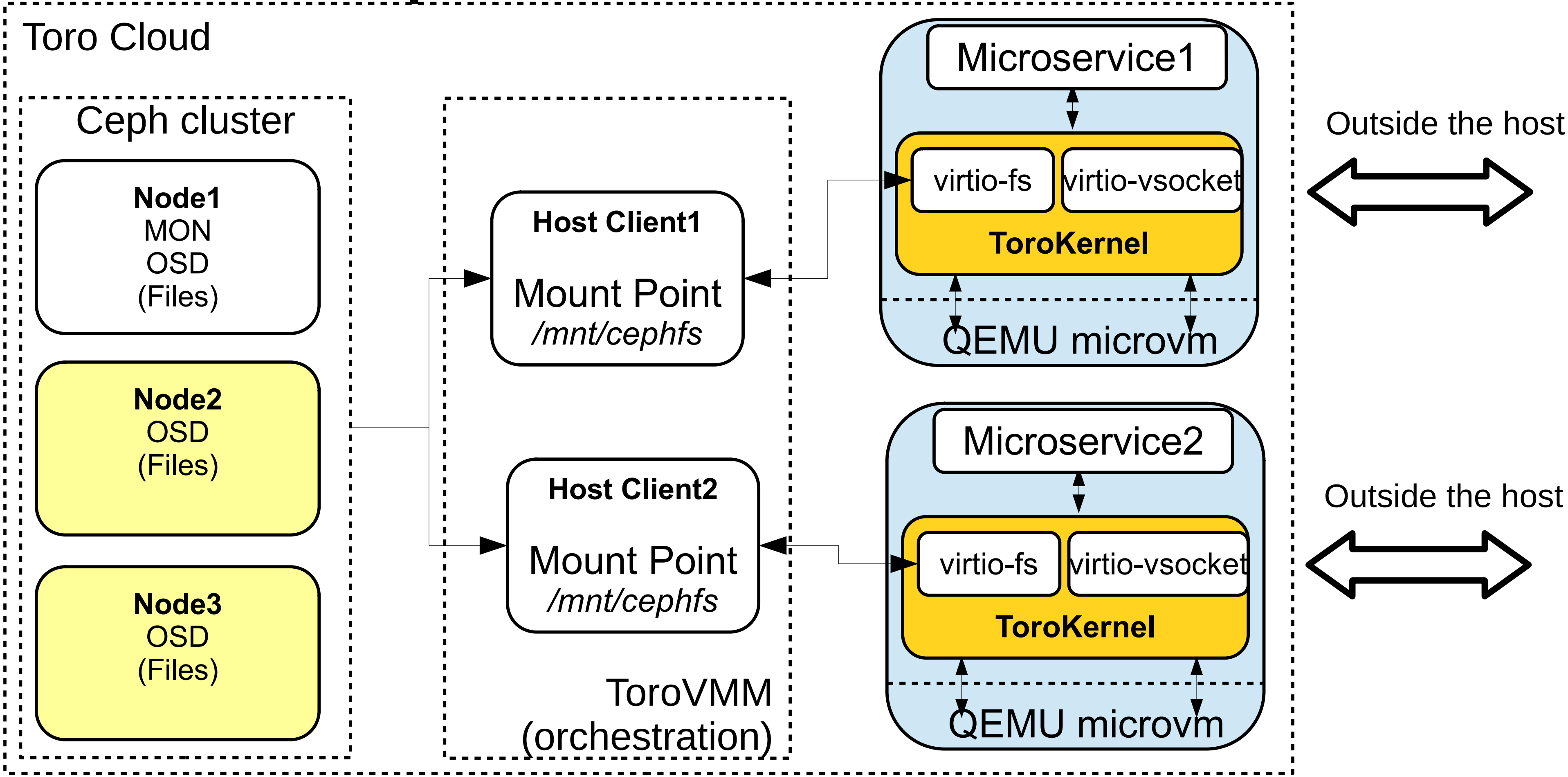
Ceph 3-node cluster



Ceph 3-node cluster



Ceph 3-node cluster



Results

- Binary Size: 235 Kb that includes kernel and user’s application
- Time to rebuild the microservice: ~ 500ms
- Boot cycle: ~ 80ms
  - $echo “Hello World” is ~2.6 ms
- CPU Usage: 90% at high and 10% sleep
- Memory footprint per VM: 2.9% (~ 60Mb) or 35 VMs per hosts
  - QEMU compiled with all enabled
- Price: 58 euros/month ~ 0.85 euros/month per VM
- See https://github.com/torokernel/torocloudscripts

Results

- Binary Size: 235 Kb that includes kernel and user’s application
- Time to rebuild the microservice: ~ 500ms
- Boot cycle:
  - $echo “
- CPU Usage: 90% at high and 10% sleep
- Memory footprint per VM: 2.9% (~ 60Mb) or 35 VMs per hosts
  - QEMU compiled withall the configuration
- Price: 58 euros/month ~ 0.85 euros/month per VM
- See https://github.com/torokernel/torocloudscripts

“It is all talk until code runs.” - Ward Cunningham

Challenges

- Support live-migration which is not currently supported by microvm machine
- Improve bottleneck at vsocket forwarding
- Improve overall performance by using zero copy in virtio-fs and virtio-vsocket
- Improve evaluation by comparing with unikernels/containers/gpos

QA