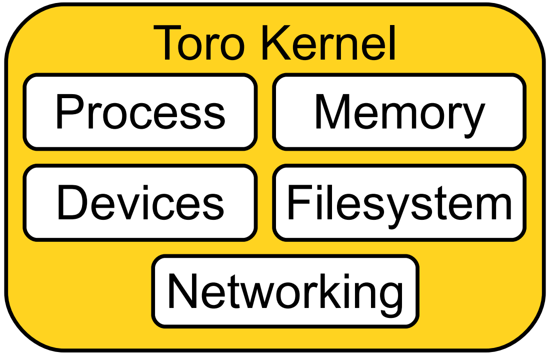
# Speeding up the Booting Time of a Toro Appliance

Matias E. Vara Larsen

www.torokernel.io
matiasevara@gmail.com

# Application-oriented Kernel

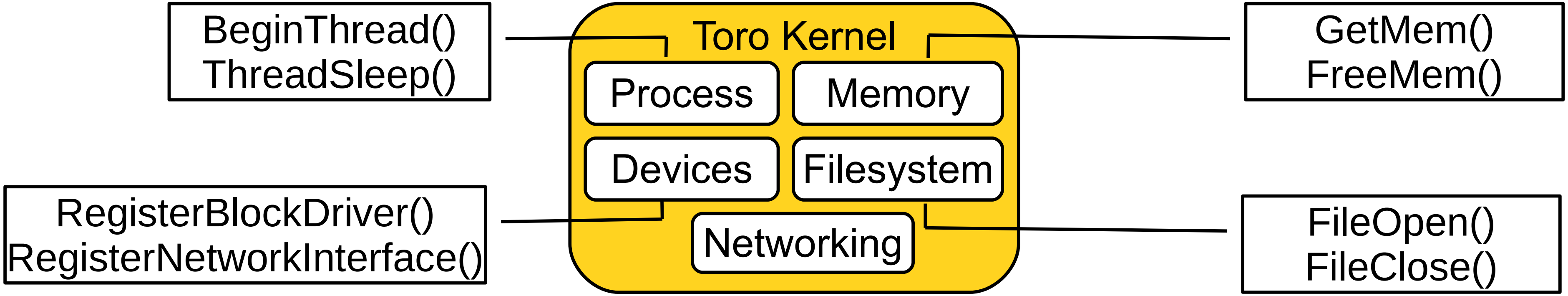


Toro is an embedded kernel including five units:

- Process
- Memory
- Filesystem
- Networking
- Devices, e.g., Block Device, Network Device

Each unit provides minimalist APIs accessible from the embedded application

# Application-oriented Kernel



BeginThread()
ThreadSleep()

RegisterBlockDriver()
RegisterNetworkInterface()

Toro Kernel

Process

Memory

Devices

Filesystem

Networking

GetMem()
FreeMem()

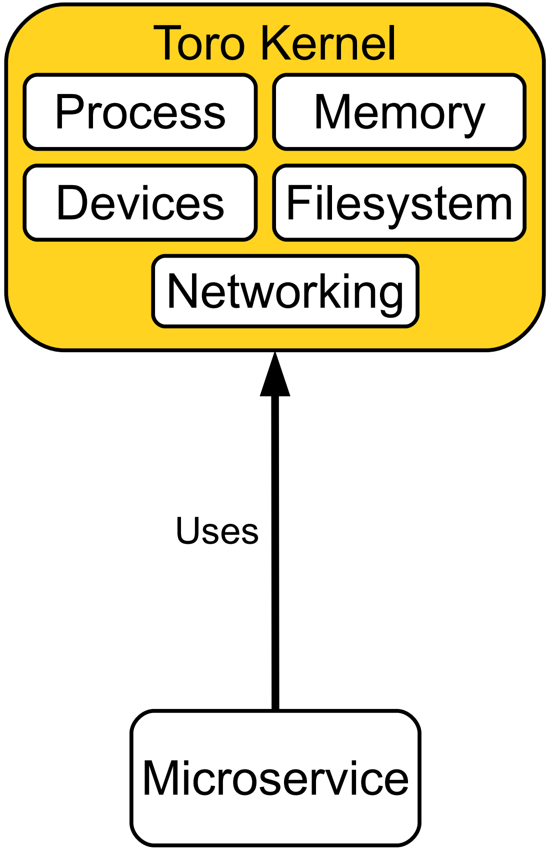
FileOpen()
FileClose()

Toro is an embedded kernel including five units:

- Process
- Memory
- Filesystem
- Networking
- Devices, e.g., Block Device, Network Device

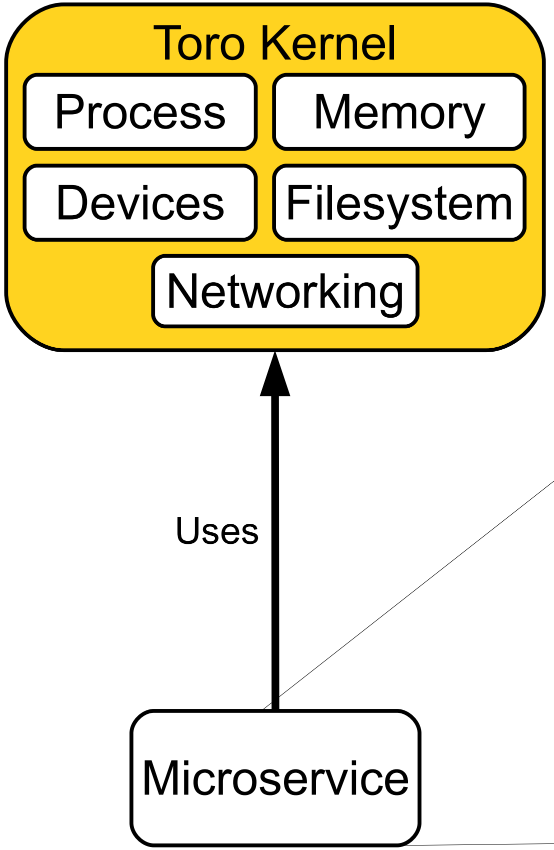
Each unit provides minimalist APIs accessible from the embedded application

# Application-oriented Kernel



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

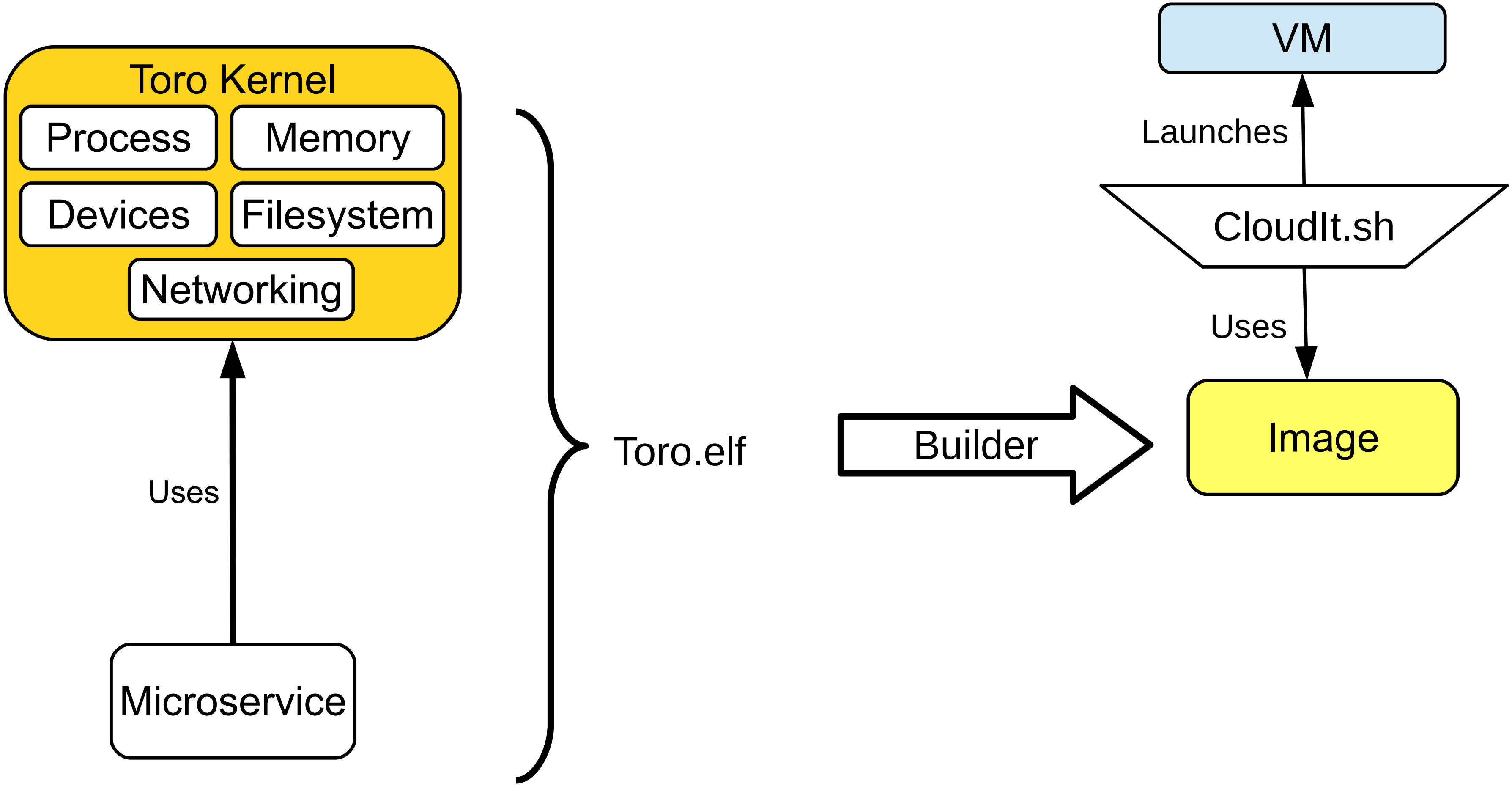
# Application-oriented Kernel



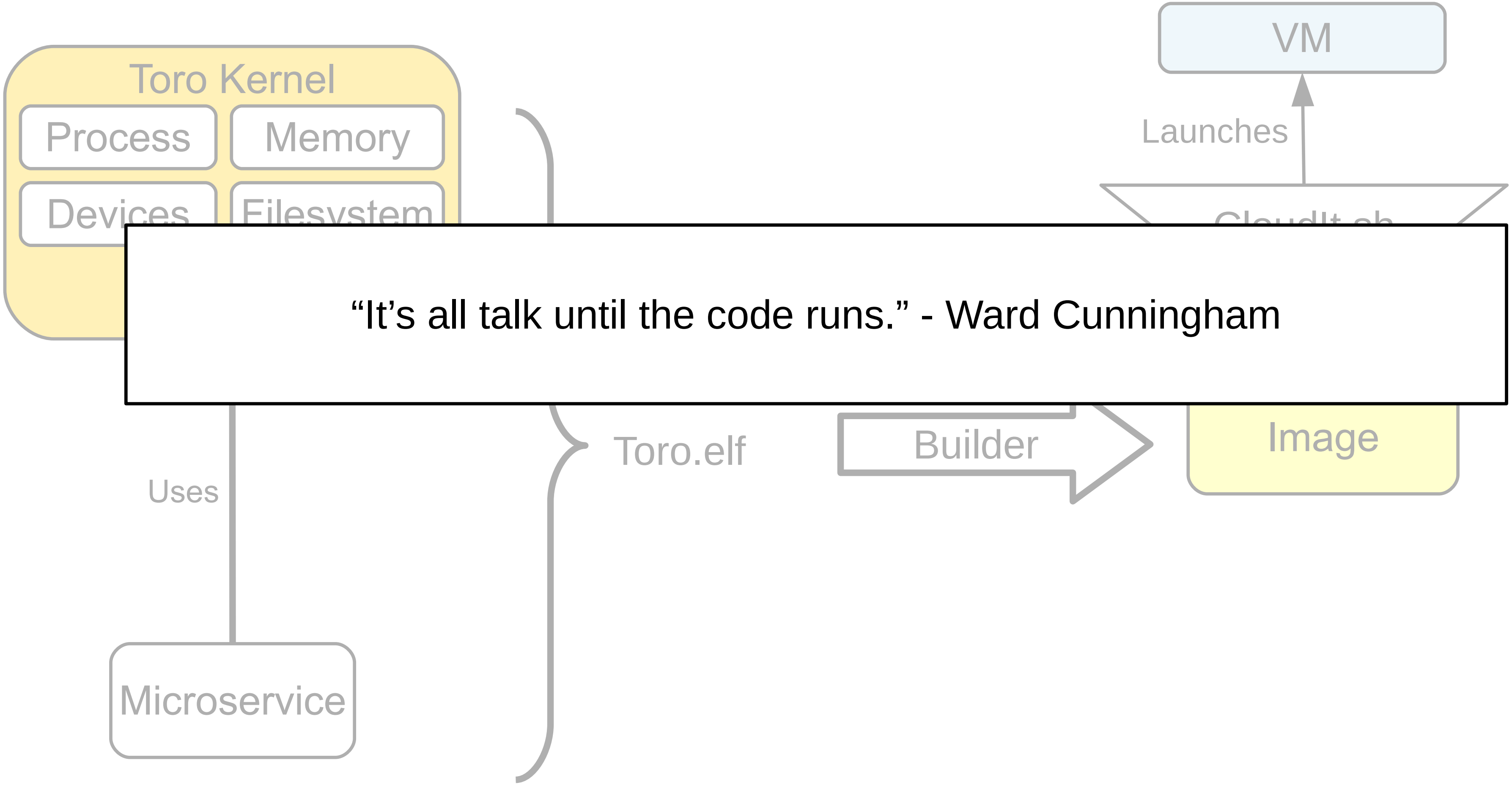
- User application and kernel units are compiled in a single binary
- The com

```
program HelloWorld;

uses
  Memory,
  Filesystem,
  Ext2,
  E1000;

begin
//
// Your Code
//
end.
```

# Application-oriented Kernel

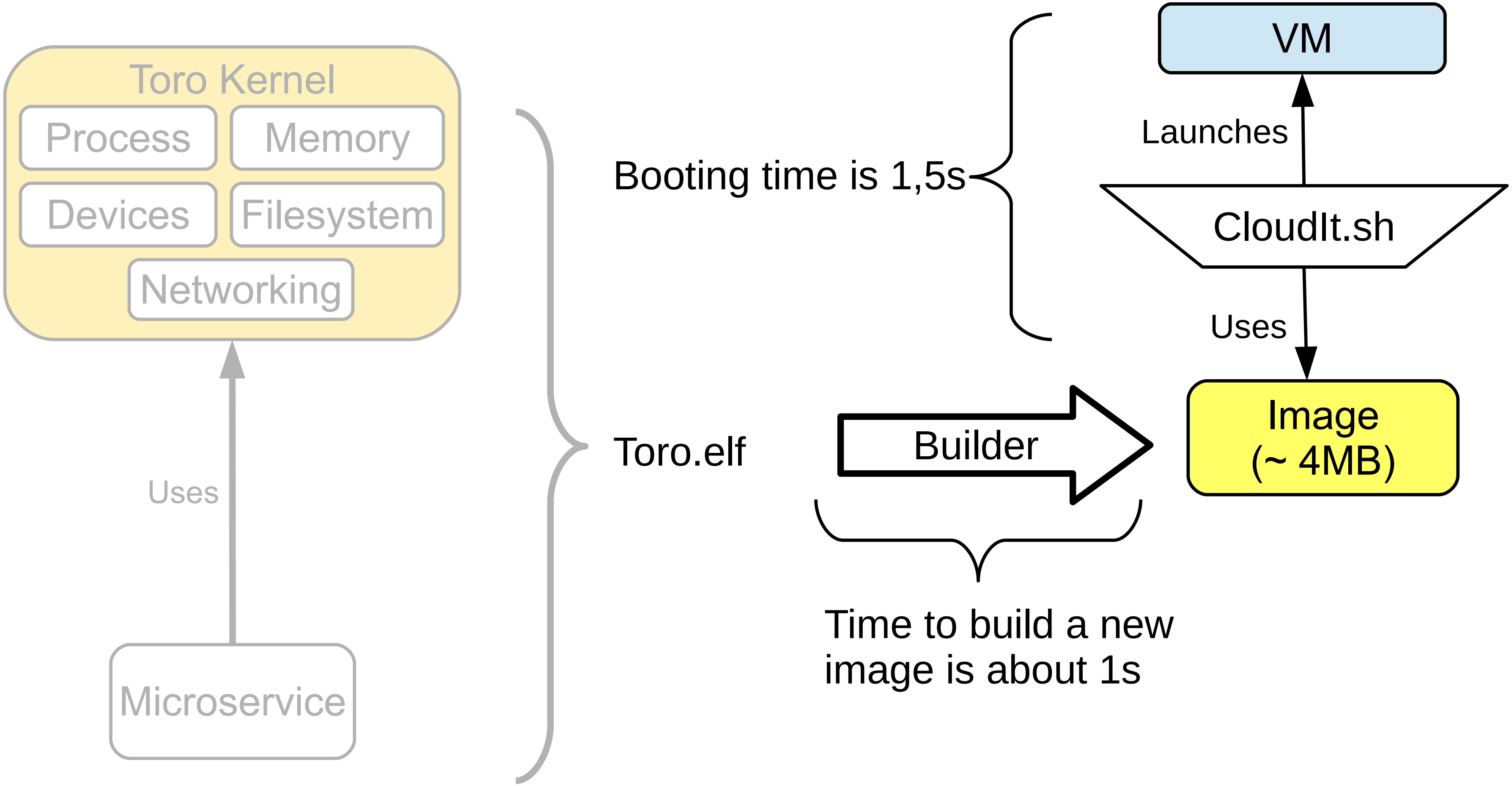


# Application-oriented Kernel

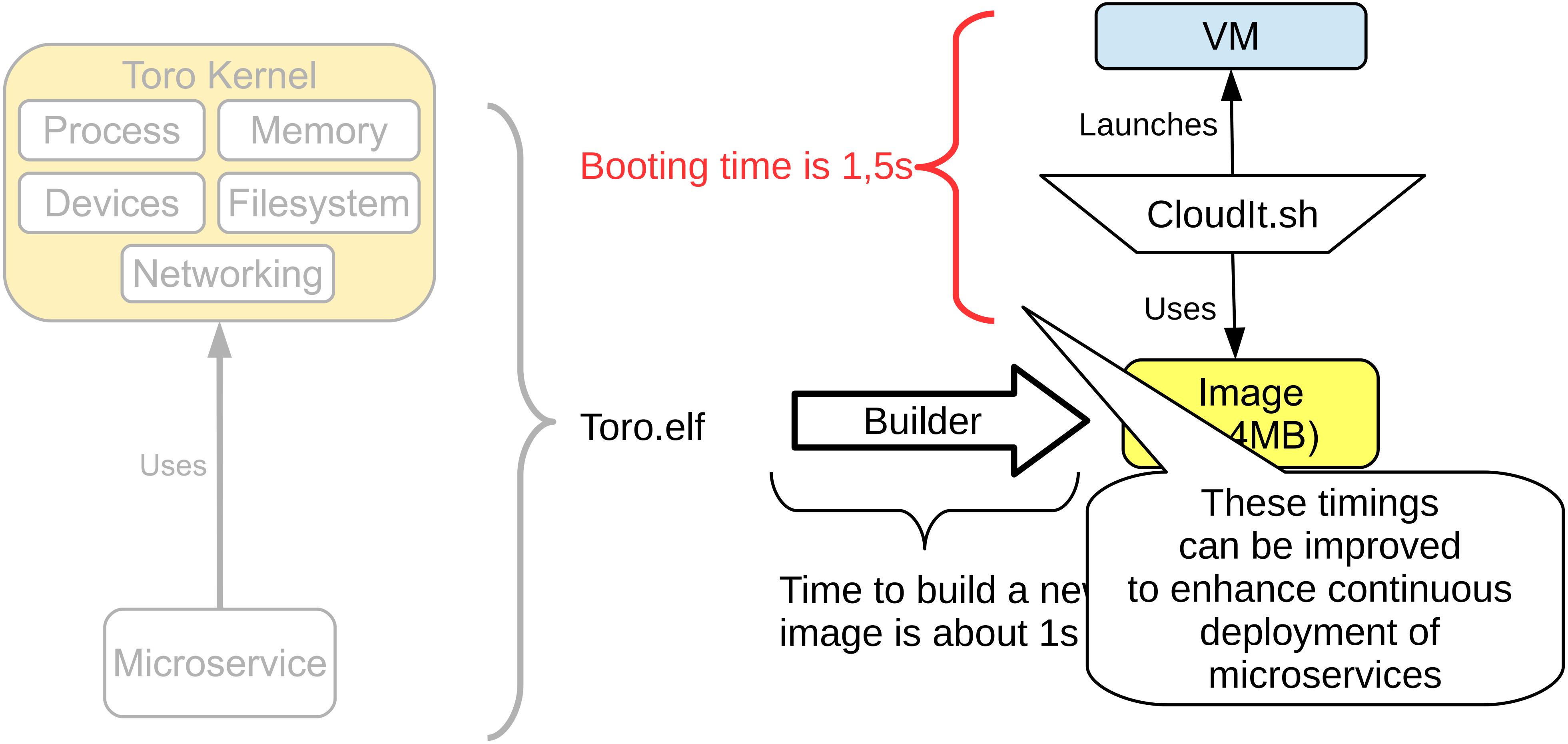


“It’s all talk until the code runs.” - Ward Cunningham

# Application-oriented Kernel



# Application-oriented Kernel



Toro Kernel

Process

Memory

Devices

Filesystem

Networking

Uses

Microservice

Toro.elf

Builder

Time to build a ne
image is about 1s

Booting time is 1,5s

VM

Launches

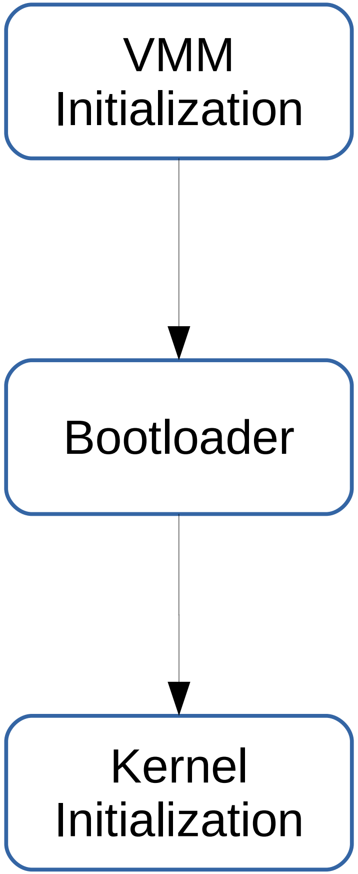
CloudIt.sh

Uses

Image
4MB)

These timings
can be improved
to enhance continuous
deployment of
microservices

# Booting in Toro

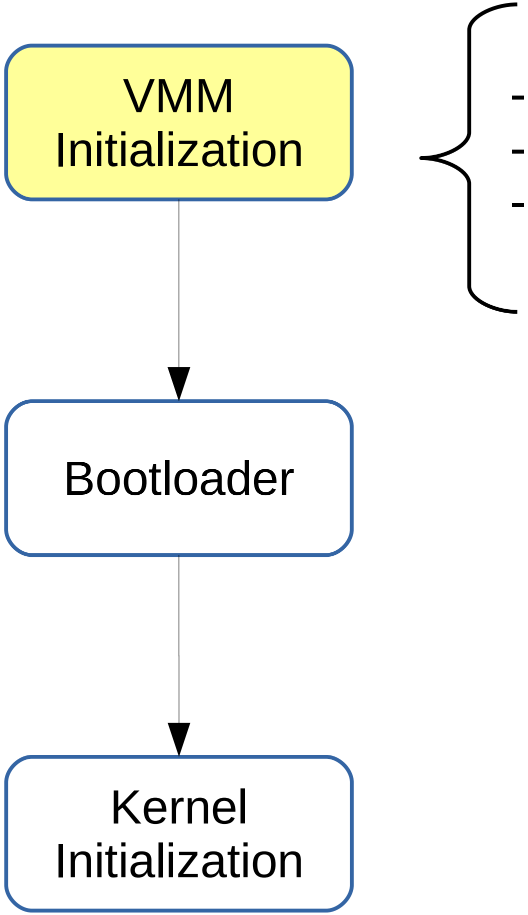


VMM Initialization

Bootloader

Kernel Initialization

# Booting in Toro



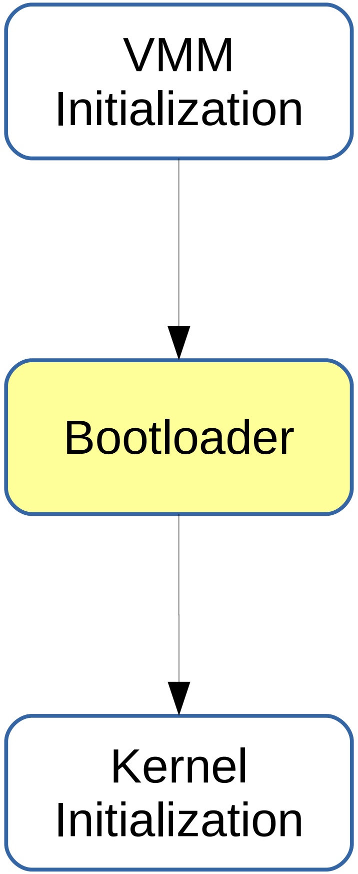
VMM Initialization

Bootloader

Kernel Initialization

- Initialization of the device model
- BIOS
- Other stuff

# Booting in Toro



VMM Initialization

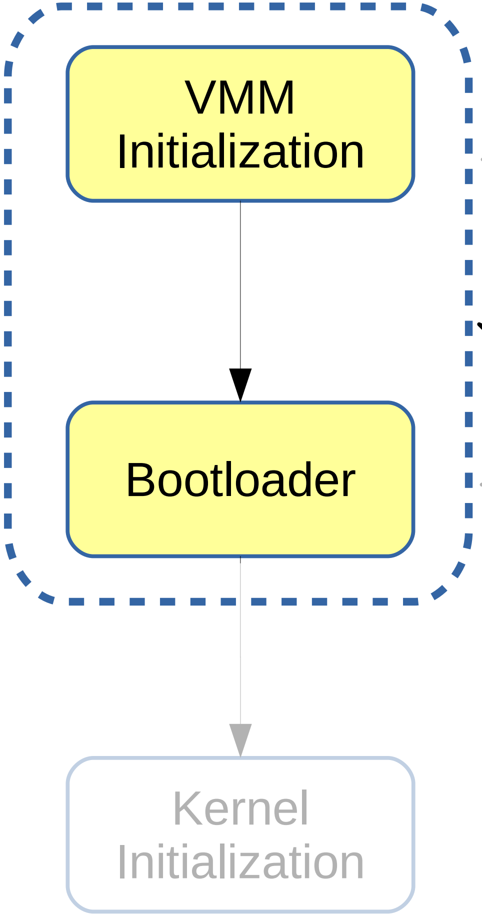
- Initialization of the device model
- BIOS
- Other stuff

Bootloader

- Initialize hardware
- Initialize processors, e.g., setup and enable paging, enable long mode, etc
- Load the kernel into memory. In this case the image’s size is very important

Kernel Initialization

# Booting in Toro



- Initialization of the device model
- BIOS
- Other stuff

Initialize hardware

- lize processors, e.g., setup and enable

ory. In this case the

ant

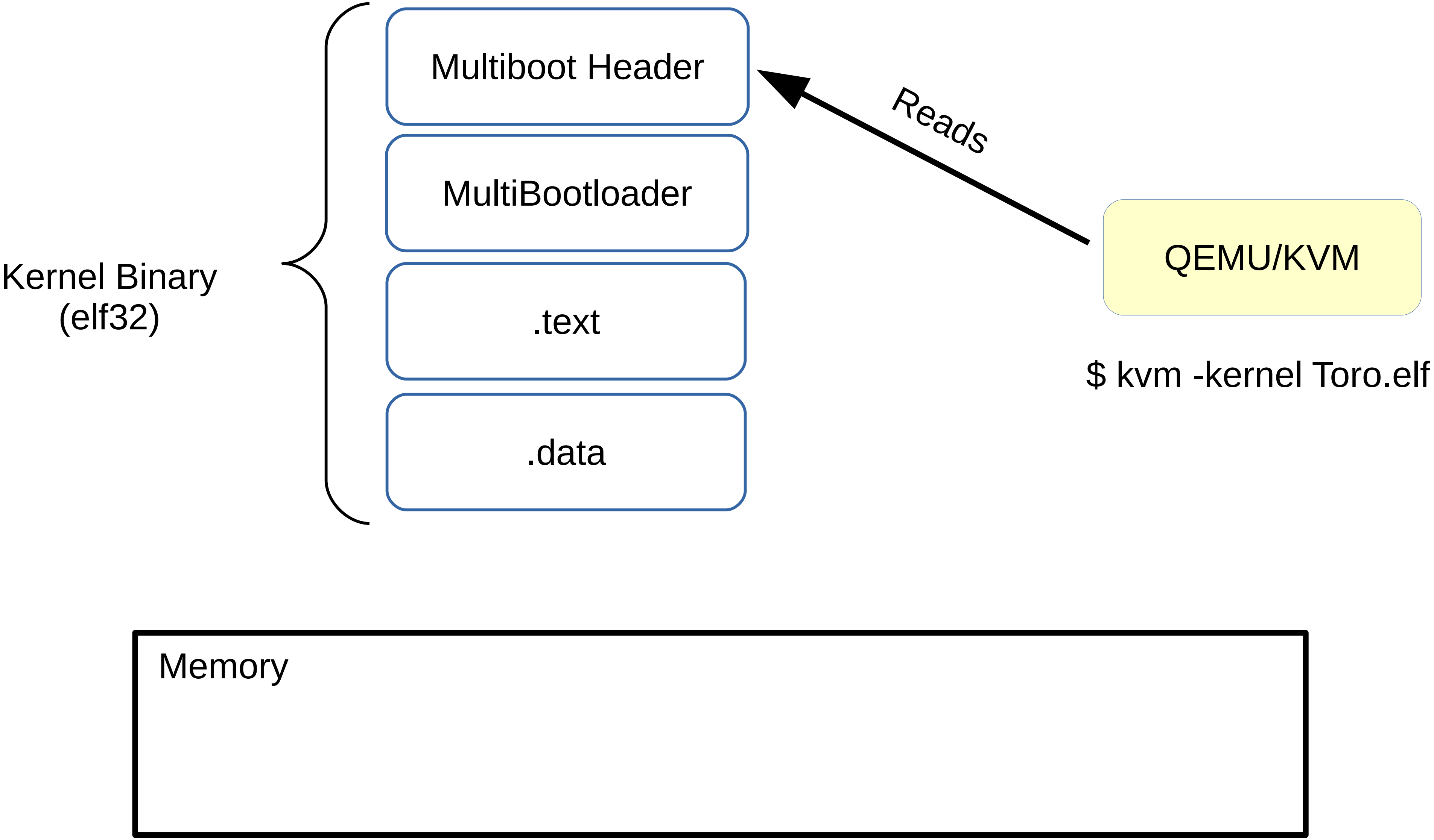
This presentation deals with different approaches to improve these times

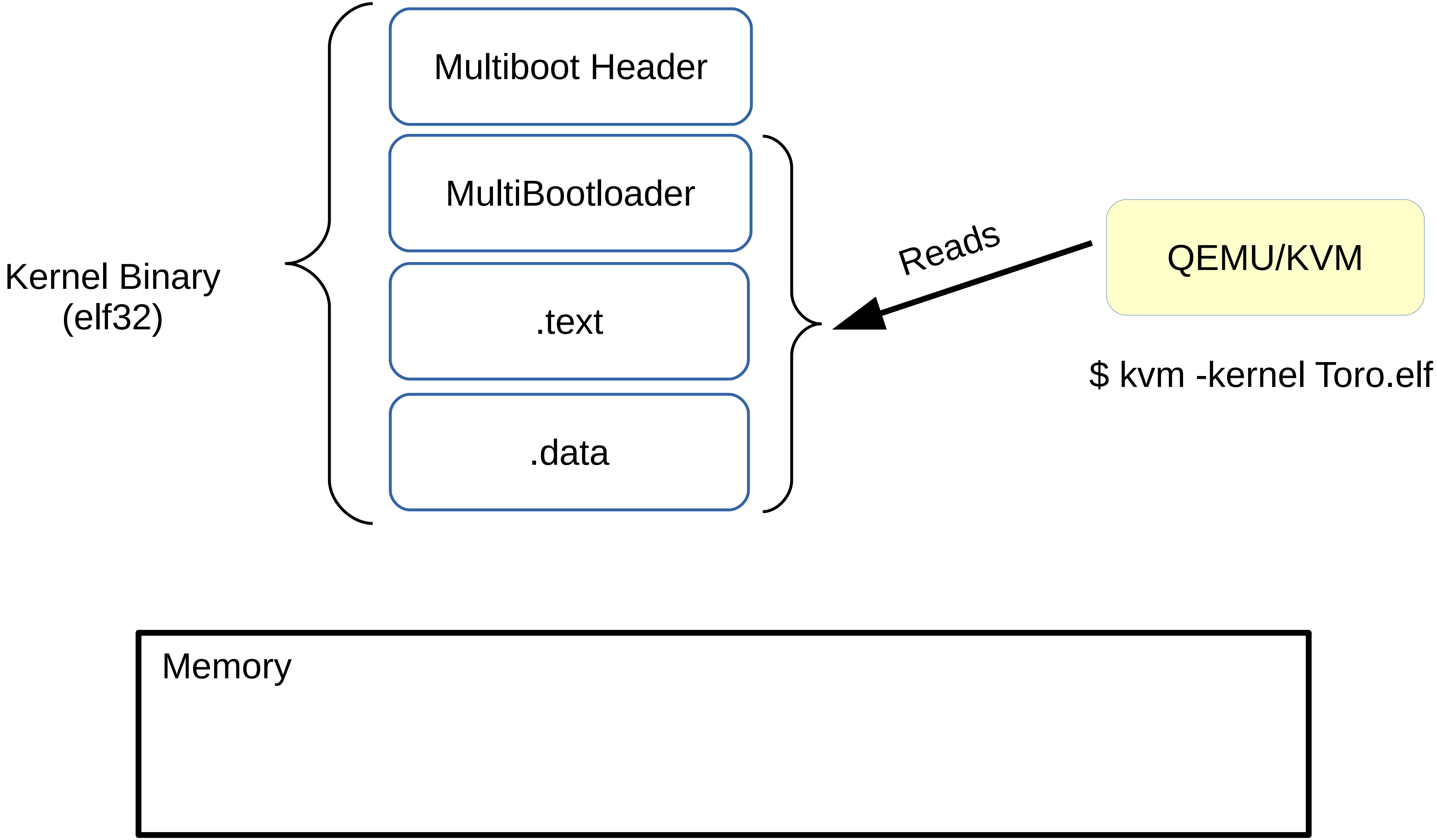
# Outline

- Speeding Up the Bootloader
- Speeding Up the Virtual Machine Monitor (VMM)
- Evaluation
- Conclusion
- QA

# Speeding Up the Bootloader

- Context:
  - The generated image is a copy of the kernel in memory
  - The bootloader just read from the disk the image and then it writes it to memory
- Problem:
  - The resulting image is huge
  - The bootloader is still complex
- Proposal:
  - Load Toro by using the “**-kernel**” option in QEMU/KVM (see Issue #223 at Github)


Kernel Binary
(elf32)
Multiboot Header
MultiBootloader
.text
.data
Reads
QEMU/KVM
$ kvm -kernel Toro.elf
Memory



Kernel Binary (elf32)

Multiboot Header

MultiBootloader

.text

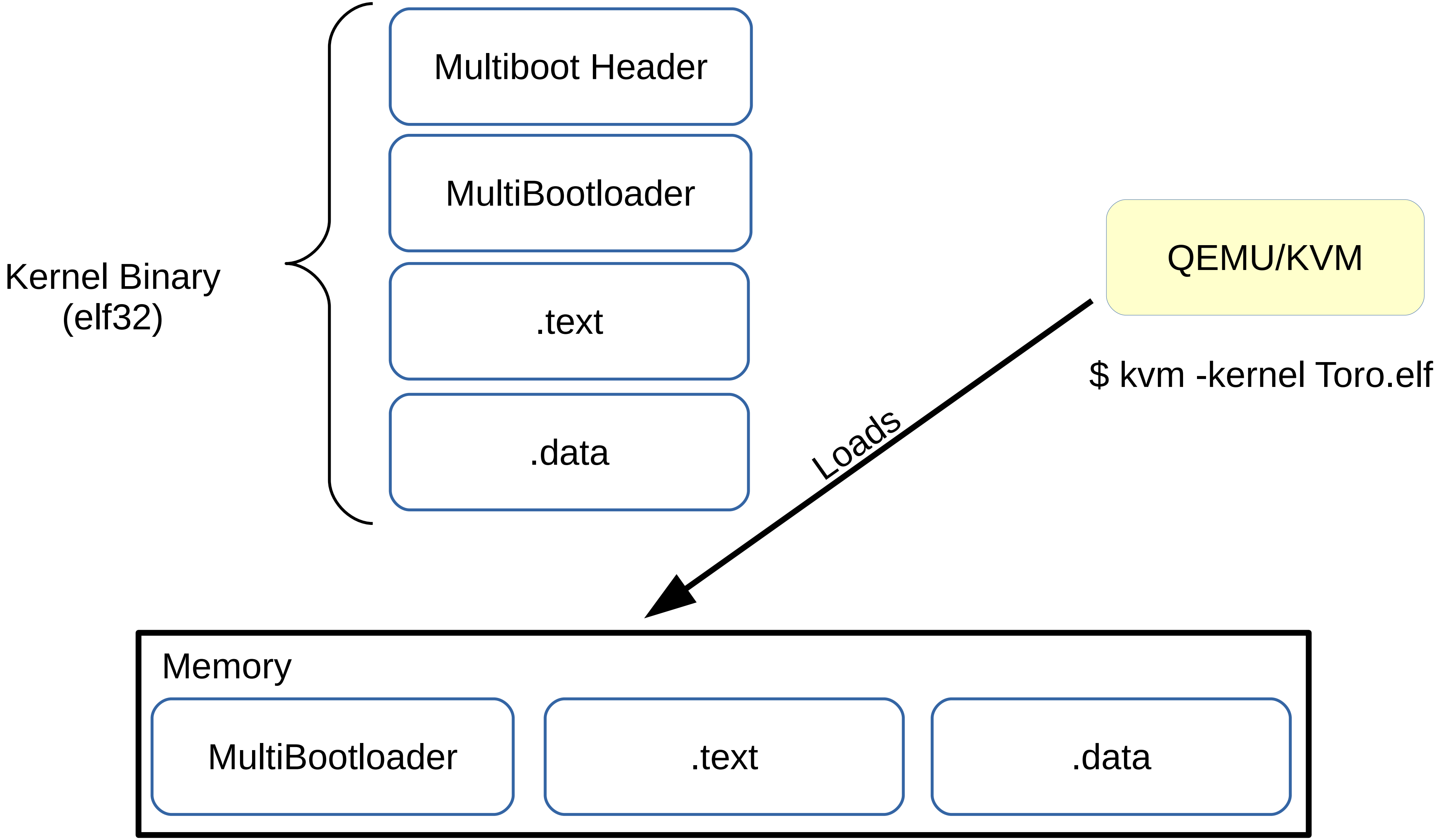
.data

Reads

QEMU/KVM

```
$ kvm -kernel Toro.elf
```

Memory



Kernel Binary (elf32)

Multiboot Header

MultiBootloader

.text

.data

QEMU/KVM

```
$ kvm -kernel Toro.elf
```

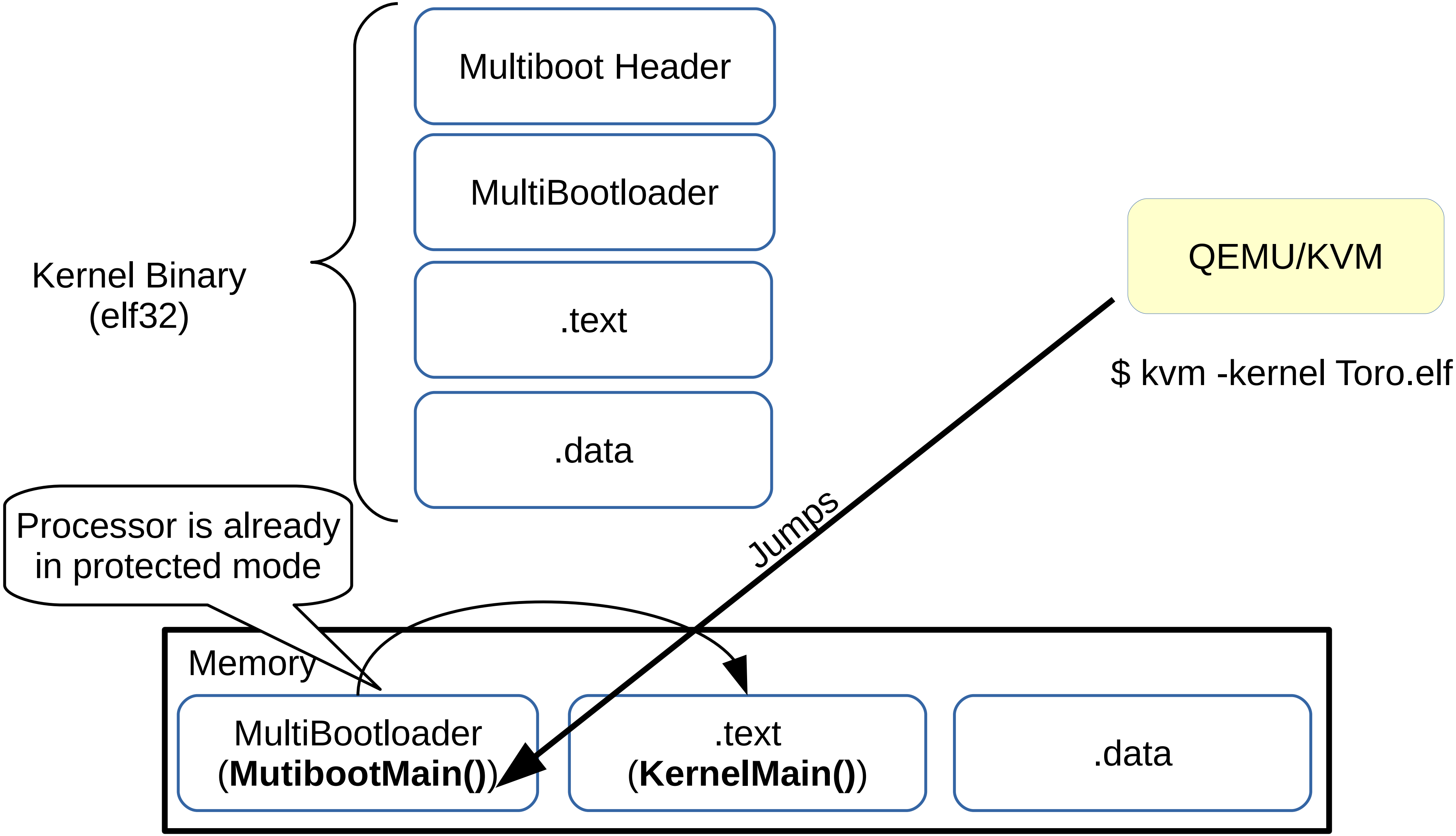
Loads

Memory

MultiBootloader

.text

.data



Kernel Binary (elf32)

Multiboot Header

MultiBootloader

.text

.data

QEMU/KVM

$ kvm -kernel Toro.elf

Jumps

Processor is already in protected mode

Memory

MultiBootloader (**MutibootMain()**)

.text (**KernelMain()**)

.data

# Speeding Up the Bootloader

- Benefits:
  - Reduce image size since it is only an elf32 binary from 4MB to 130kb
  - Reduce bootloader complexity since QEMU loads the kernel into memory and yield the CPU to protected mode
  - Reduce booting time from 1.5s to 0.5s

# Speeding Up the Bootloader

- Benefits:
  - Reduce image size since it is only an elf32 binary from 4MB to 130kb
  - Reduce bootloader complexity since QEMU loads the kernel into memory and yield the CPU to protected mode
  - Reduce booting time from 1.5s to 0.5s
- Drawbacks:
  - VMM has to support the loading of a multiboot kernel
  - Supports only elf32, so some magic is needed to make it work with elf64
  - We still have to jump to long mode

# Speeding Up the Bootloader

- Benefits:
  - Reduce image size since it is only an elf32 binary from 4MB to 130kb
  - Reduce bootloader complexity since QEMU loads the kernel into memory and yield the CPU to protected mode
  - Reduce booting time from 1.5s to 0.5s
- Drawbacks:
  - VMM has to support the loading of a multiboot kernel
  - Supports only elf32, so some magic is needed to make it work with elf64
  - We still have to jump to long mode

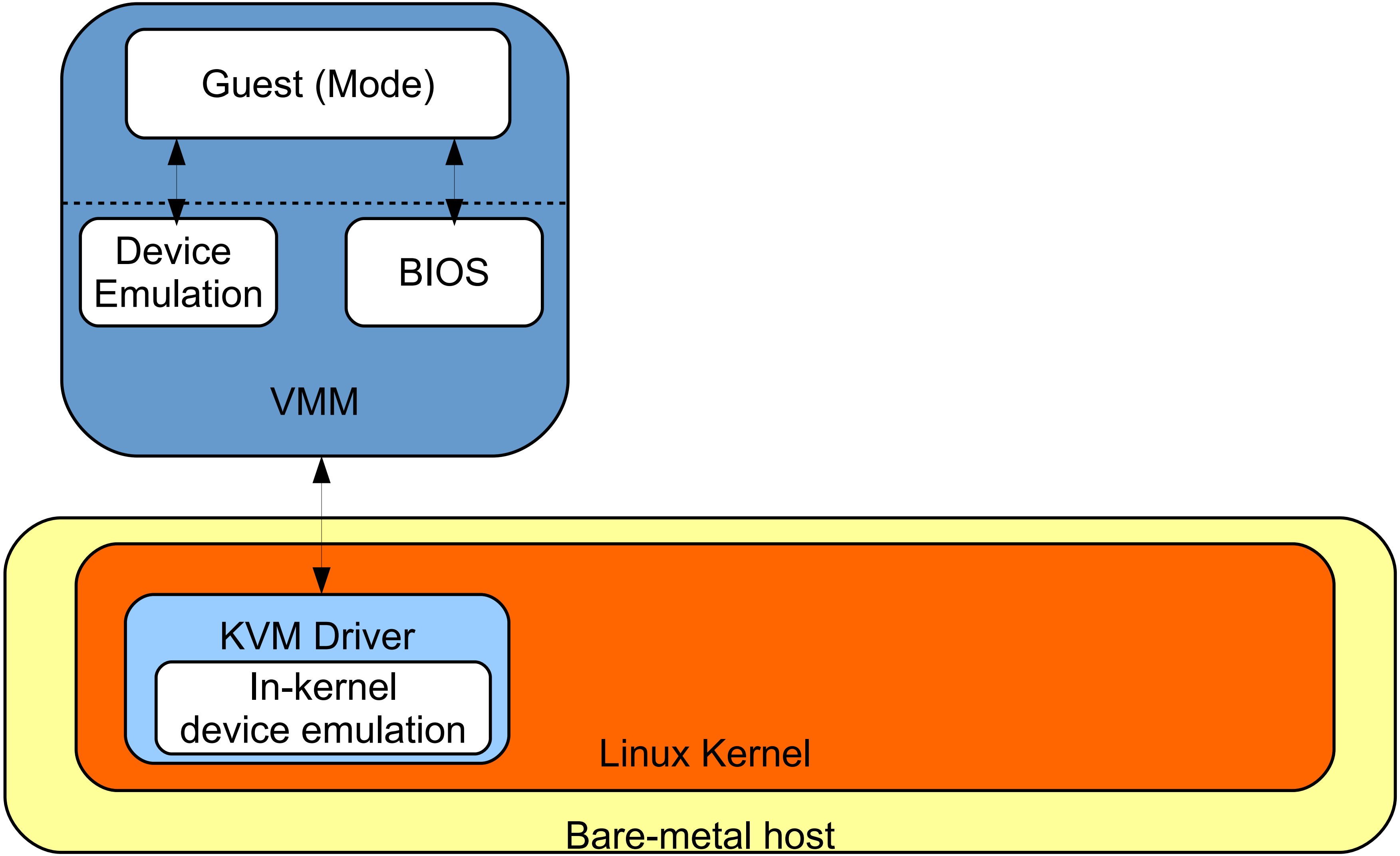
Qemu-lite works around these but project seems discontinued
(Port of Toro at Issue #192)

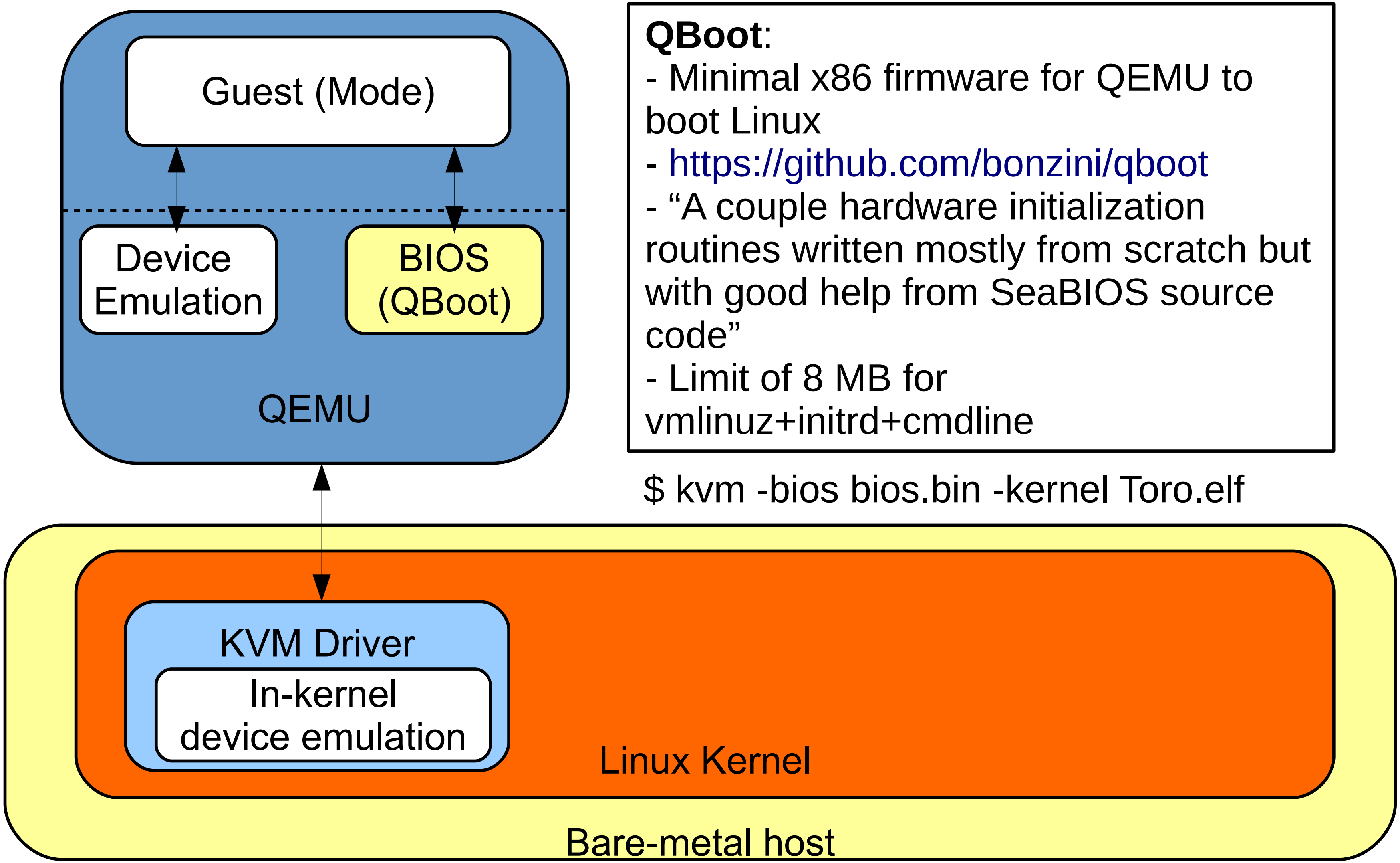
# Outline

- Speeding Up the Bootloader
- Speeding Up the VMM
- Evaluation
- Conclusion
- QA

# Speeding Up the VMM

- We study three approaches to improve the time spent in VMM initialization
- We focus on KVM/QEMU-based VMM
- These approaches are: QBoot, NEMU and Firecraker
- These approaches simplifies some aspect of the VMM, e.g., loading the of the kernel, hardware initialization or device model

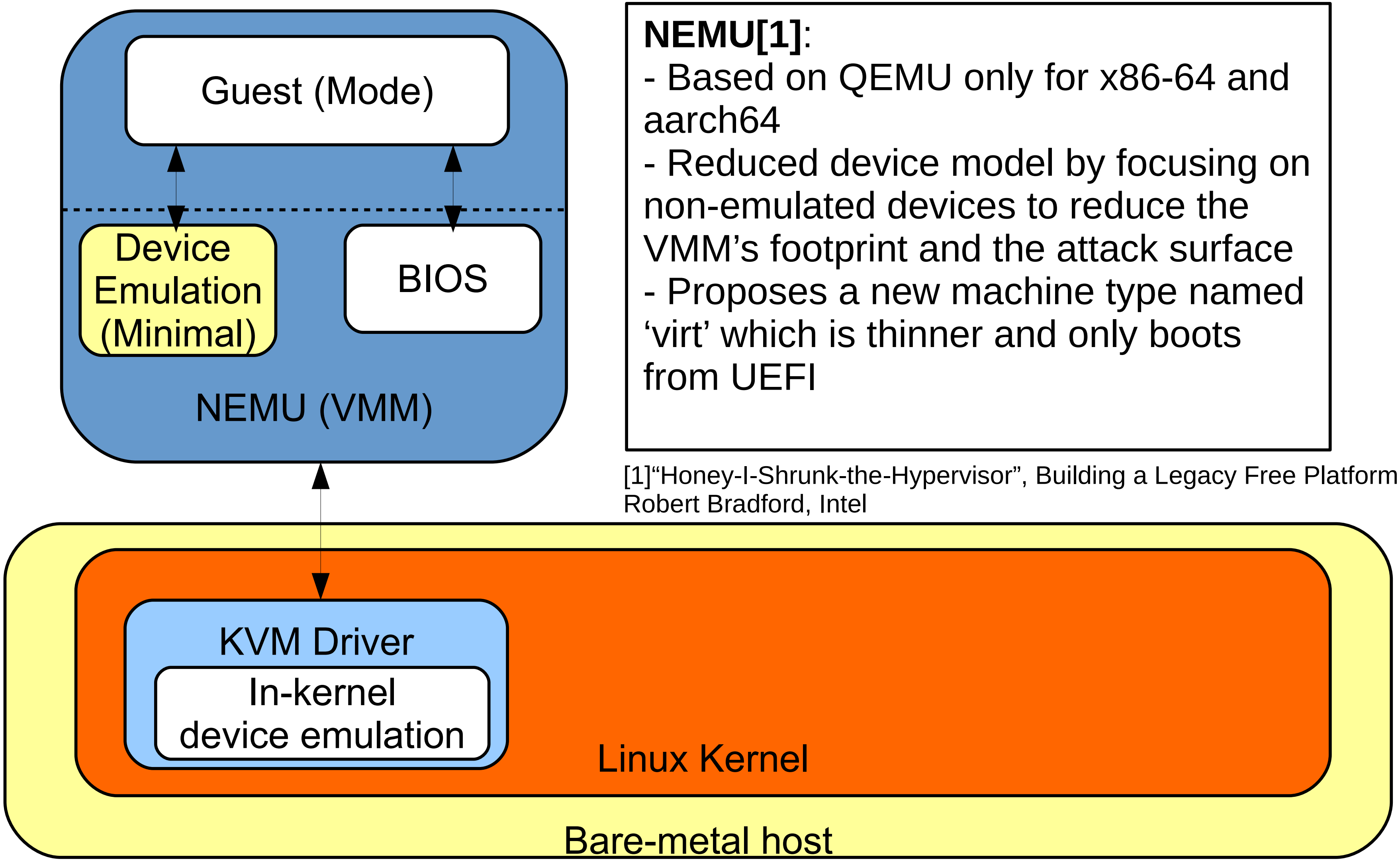

Guest (Mode)
Device Emulation
BIOS
VMM
KVM Driver
In-kernel device emulation
Linux Kernel
Bare-metal host



**QBoot**:

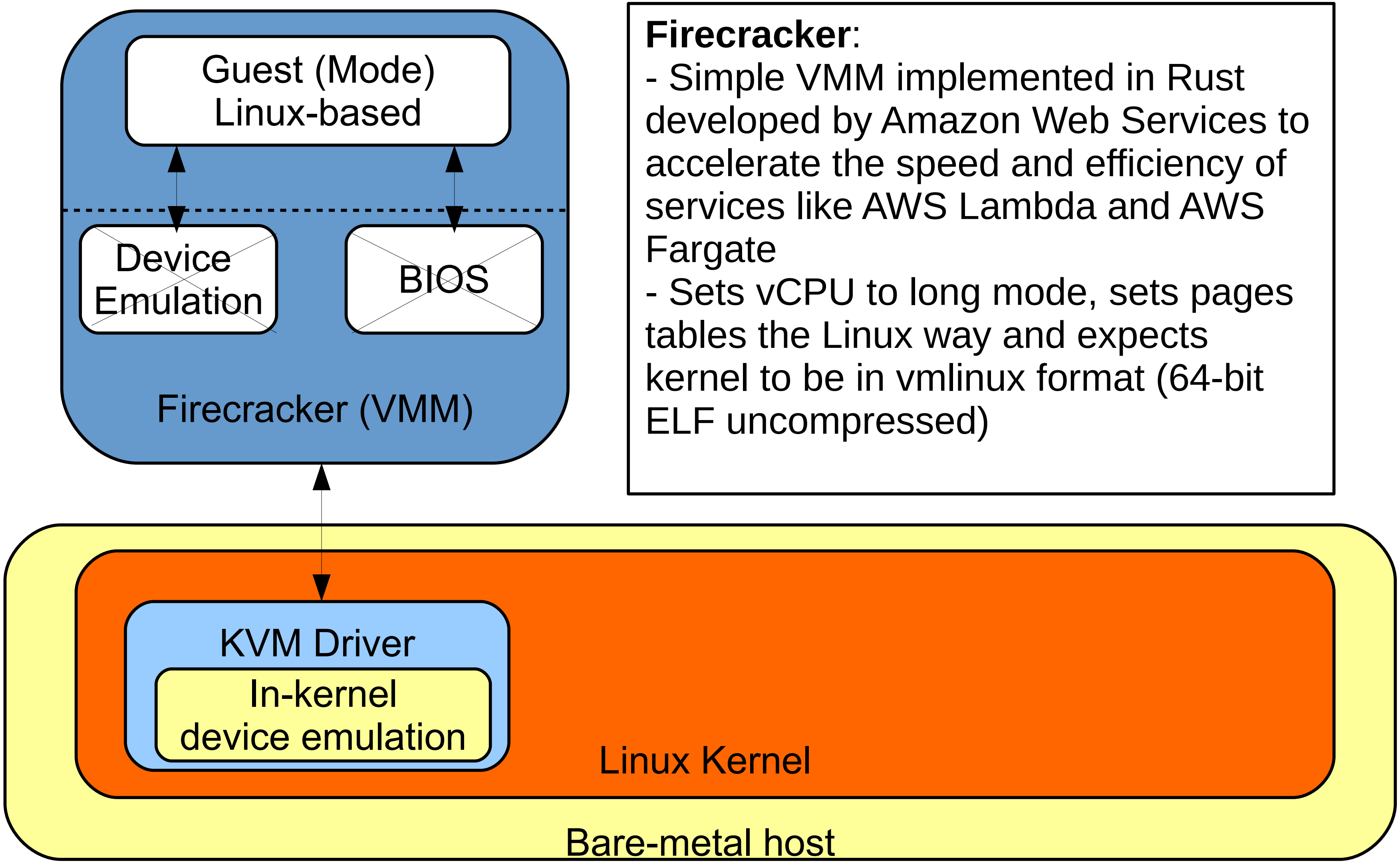
- Minimal x86 firmware for QEMU to boot Linux
- https://github.com/bonzini/qboot
- “A couple hardware initialization routines written mostly from scratch but with good help from SeaBIOS source code”
- Limit of 8 MB for vmlinuz+initrd+cmdline

```
$ kvm -bios bios.bin -kernel Toro.elf
```



**NEMU[1]**:
- Based on QEMU only for x86-64 and aarch64
- Reduced device model by focusing on non-emulated devices to reduce the VMM’s footprint and the attack surface
- Proposes a new machine type named ‘virt’ which is thinner and only boots from UEFI

[1]“Honey-I-Shrunk-the-Hypervisor”, Building a Legacy Free Platform for QEMU, Robert Bradford, Intel



**Firecracker**:
- Simple VMM implemented in Rust developed by Amazon Web Services to accelerate the speed and efficiency of services like AWS Lambda and AWS Fargate
- Sets vCPU to long mode, sets pages tables the Linux way and expects kernel to be in vmlinux format (64-bit ELF uncompressed)

# Evaluation

- We measured the time that takes the kernel to start to execute, i.e., the time since the VM is launched until the KernelMain() is executed
- We compared these times by using the presented solutions
- See Issue #276 at Github for more information

# Results

4 cores Intel(R) Atom(TM) CPU  C2550  @ 2.40GHz
8 GB of physical memory

| Approach | Image | Binary | Binary with QBoot |
|---|---|---|---|
| QEMU/KVM (2.5.0) | 1457 ms | 452 ms | 132 ms |
| NEMU (#39af42) | | 309 ms | 95 ms |
| Firecracker (0.14.0) | | 17ms | |

```
$ echo “Hello World!”
avg: 2.629263ms
```

https://blog.iron.io/the-overhead-of-docker-run/

# Conclusion

- Booting time improved by a factor x11 when using multiboot and QBoot
- Booting time improved by a factor x85 when using Firecracker
- Trade-off between the needed work to adapt the kernel and minimizing booting time

# QA

- http://www.torokernel.io
- torokernel@gmail.com
- Twitter @torokernel
- Torokernel wiki at github
  - My first Three examples with Toro
- Test Toro in 5 minutes (or less...)
  - torokernel-docker-qemu-webservices at Github

# QA

- http://www.torokernel.io
- torokernel@gmail.com

torokernel

ki at github

e examples with Toro

5 minutes (or less...)

nel-docker-qemu-webservices at

b

That's all folks!