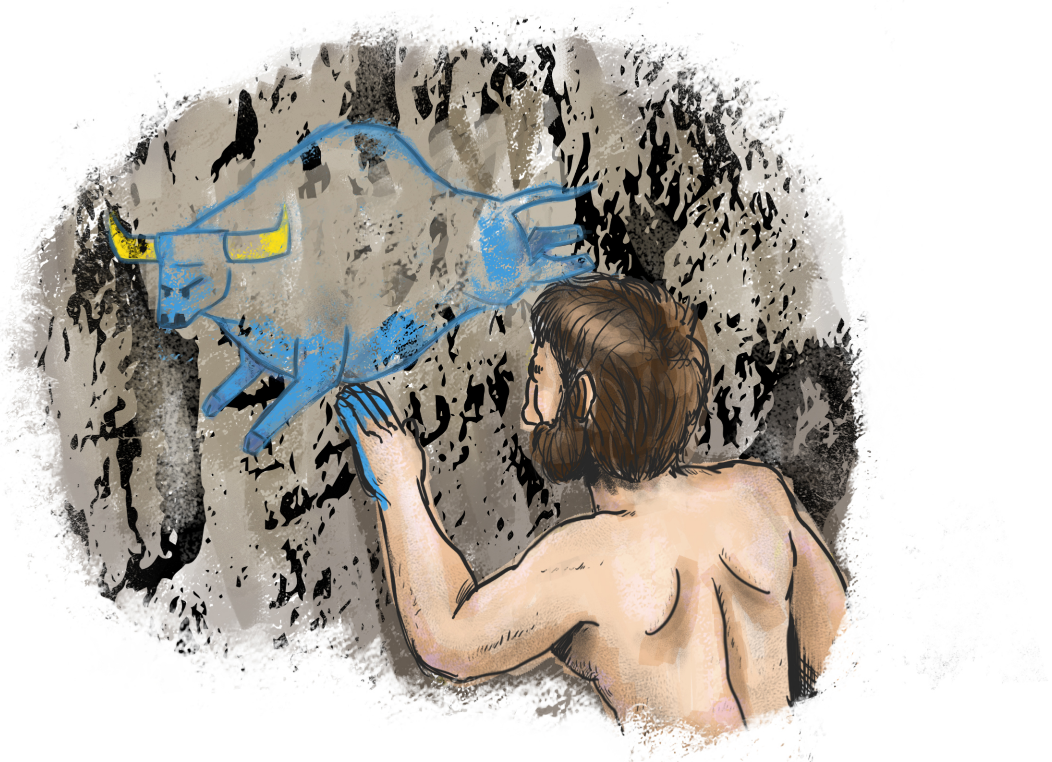


Running MPI applications on Toro unikernel

www.torokernel.io

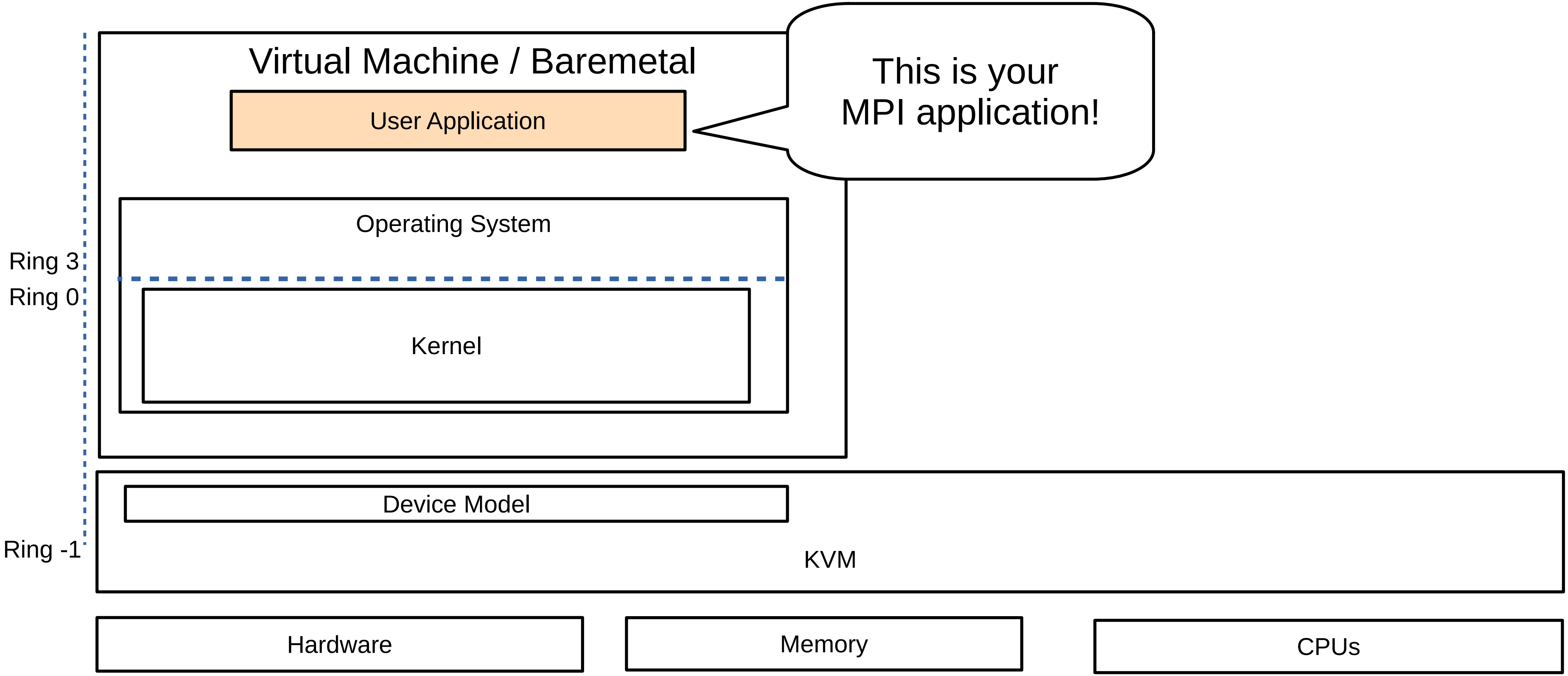
Matias Vara Larsen
matiasevara@gmail.com

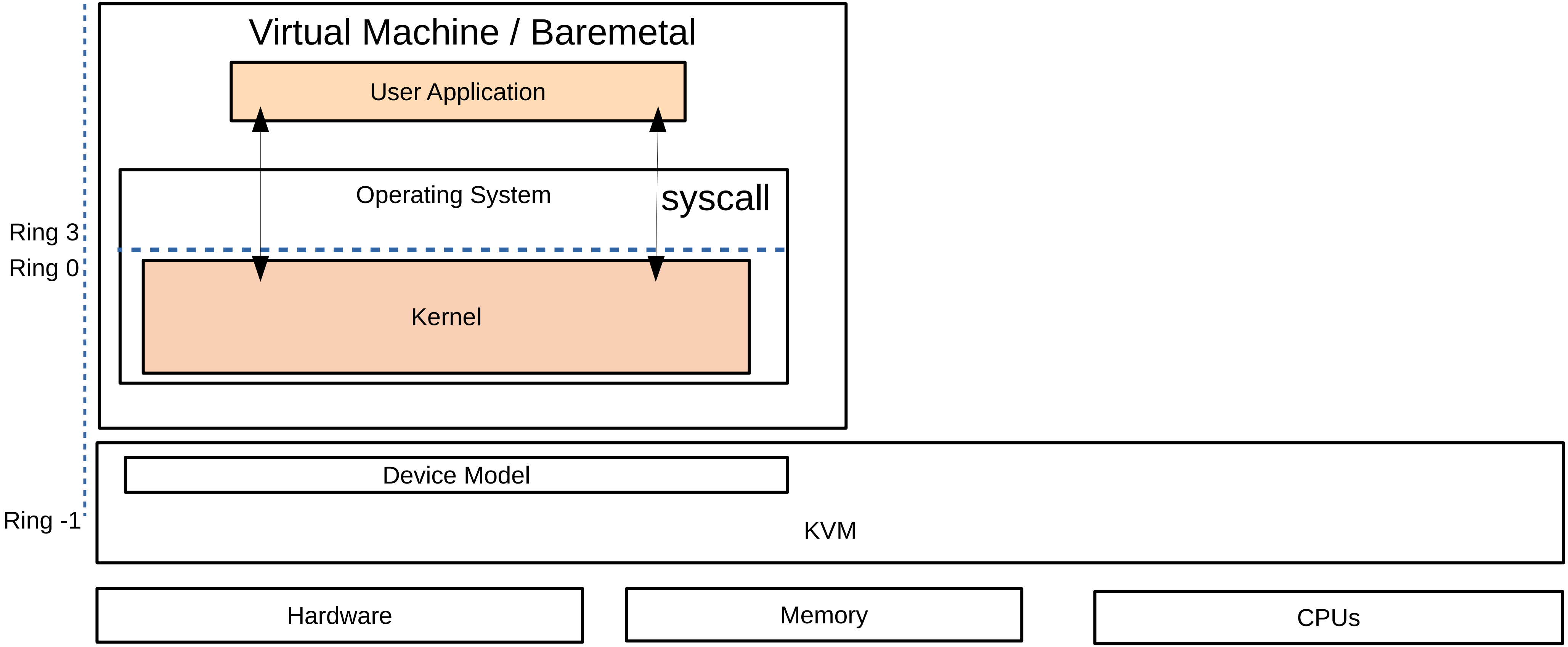
Who am I?

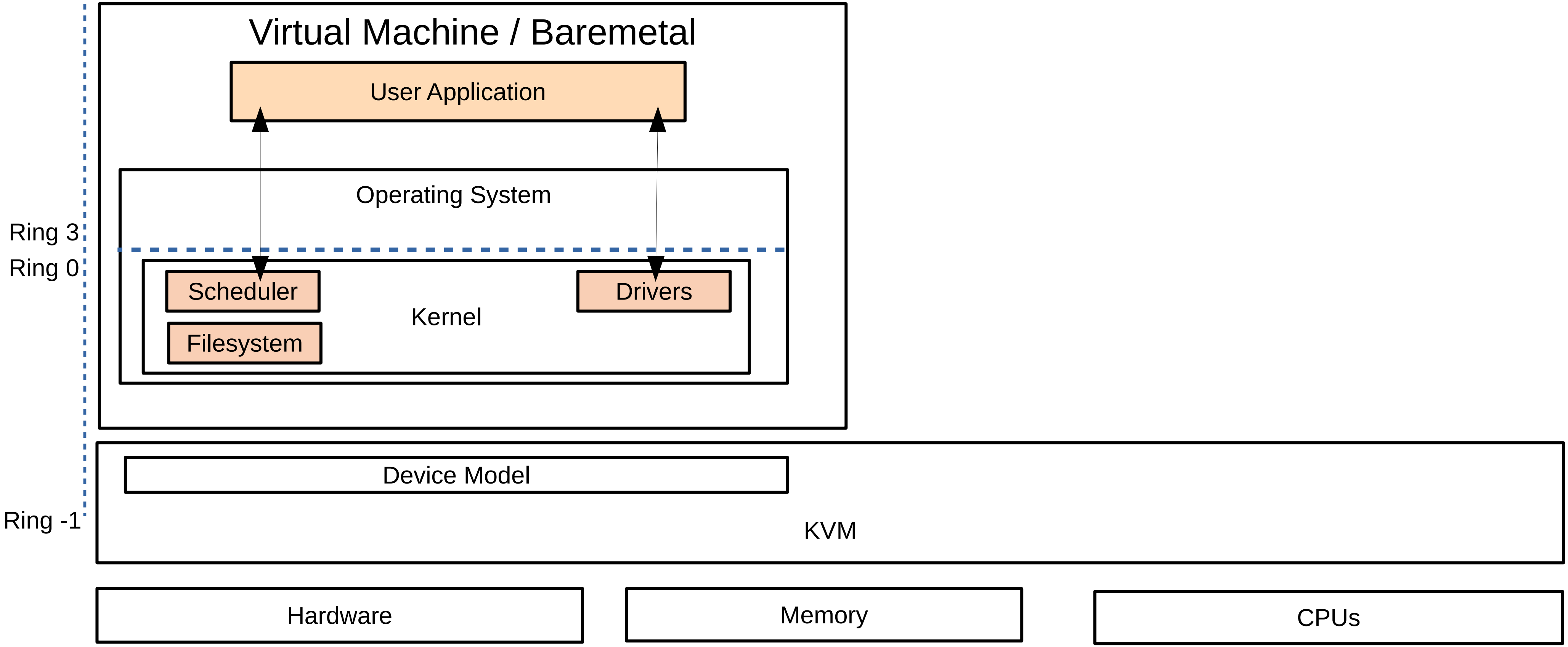
- I am passionate about operating system development and virtualization technologies
- I have worked at Citrix, Tttech, Huawei and currently at Vates
- matiasevara@gmail.com
- https://github.com/MatiasVara

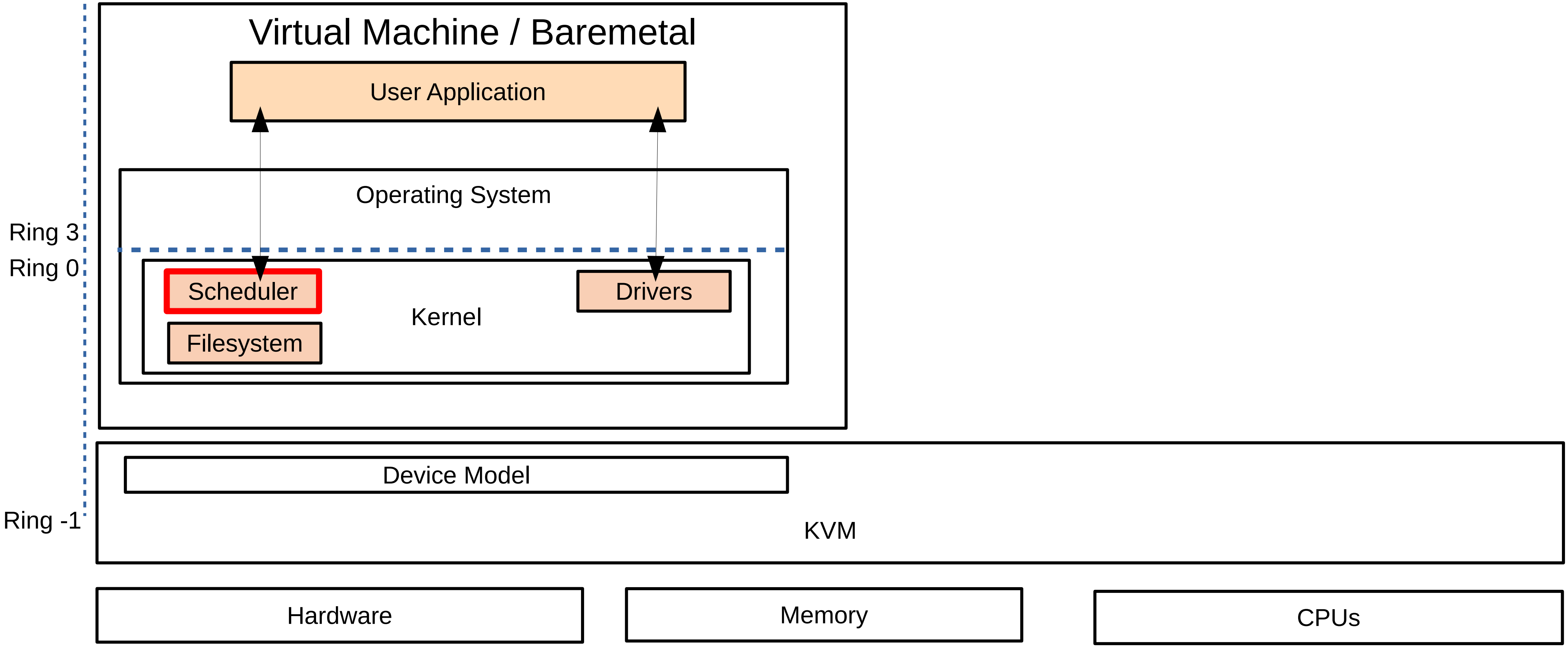
Outline

- Toro unikernel
- MPI over Toro
- OSU benchmarks

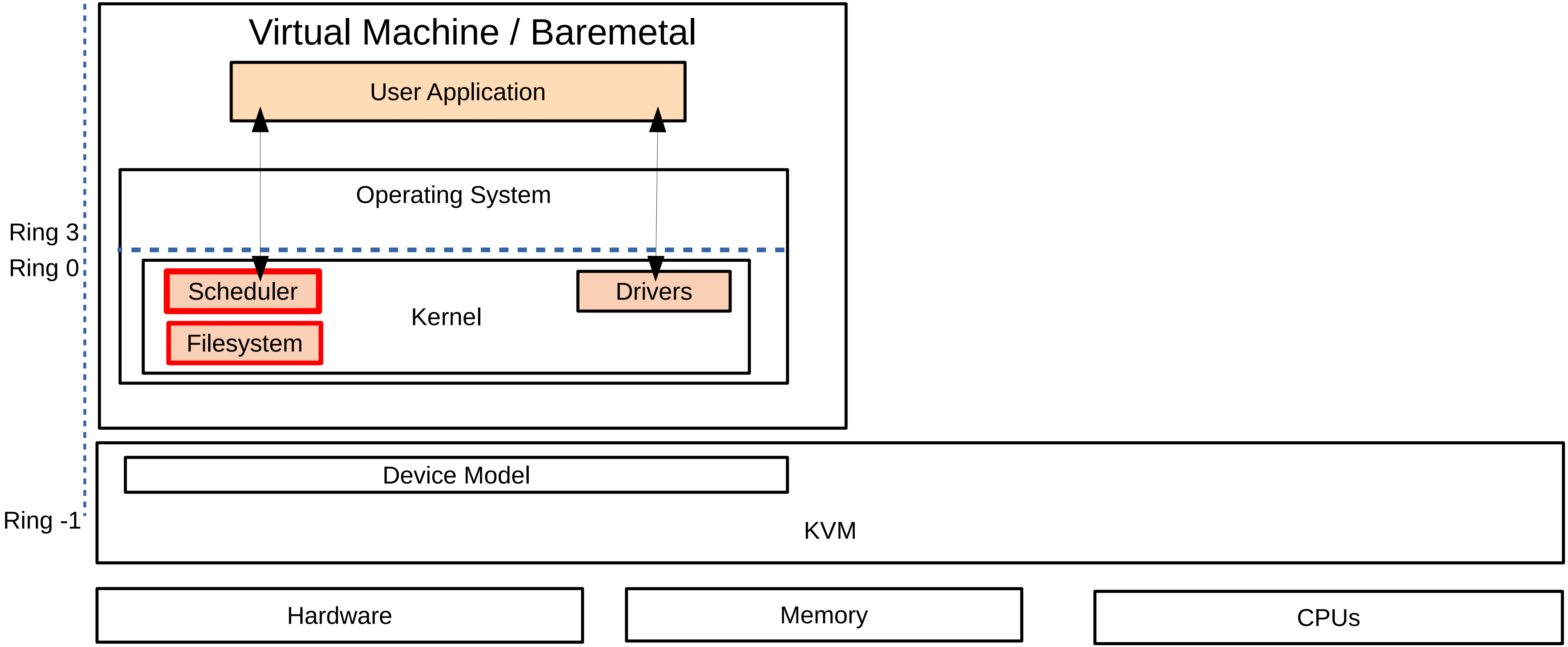

Virtual Machine / Baremetal
User Application
This is your
MPI application!
Operating System
Ring 3
Ring 0
Kernel
Device Model
Ring -1
KVM
Hardware
Memory
CPUs


Virtual Machine / Baremetal
User Application
Operating System
syscall
Ring 3
Ring 0
Kernel
Device Model
Ring -1
KVM
Hardware
Memory
CPUs

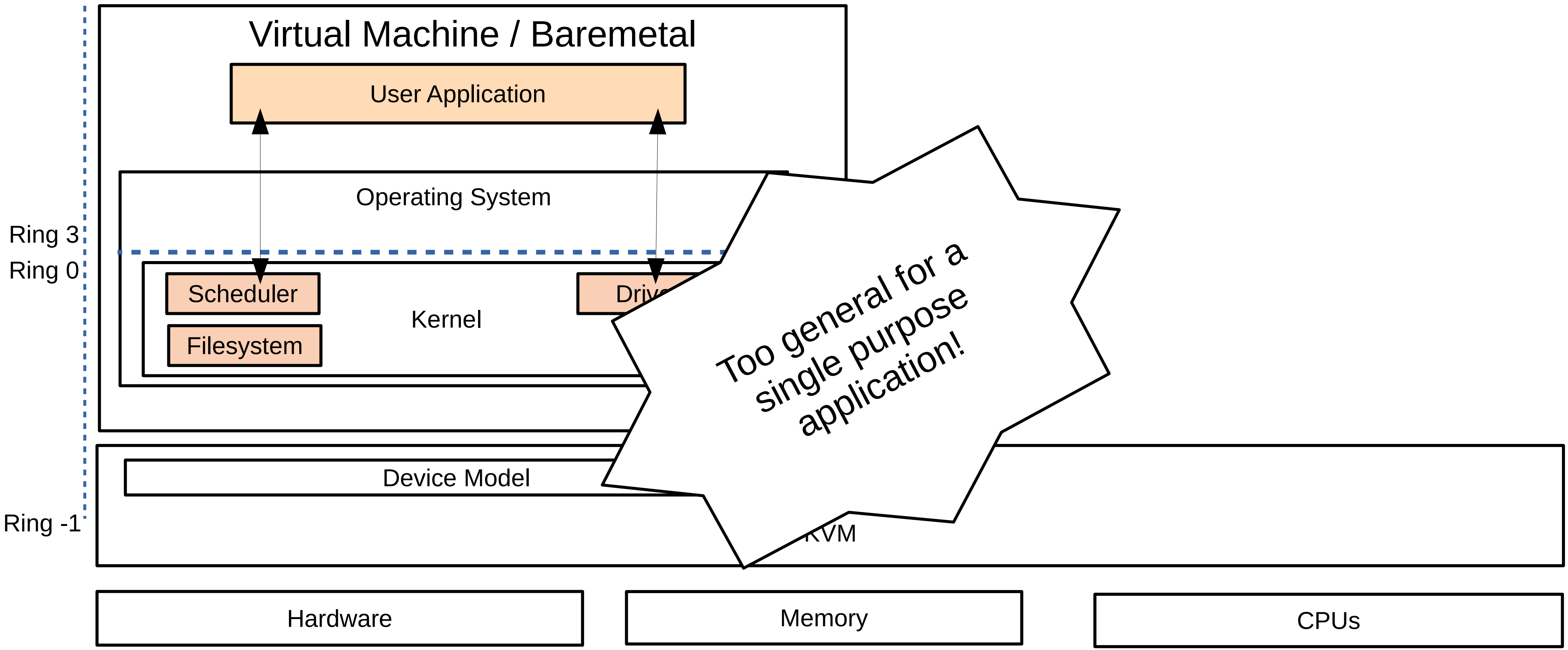

Virtual Machine / Baremetal
User Application
Operating System
Ring 3
Ring 0
Scheduler
Drivers
Kernel
Filesystem
Device Model
Ring -1
KVM
Hardware
Memory
CPUs

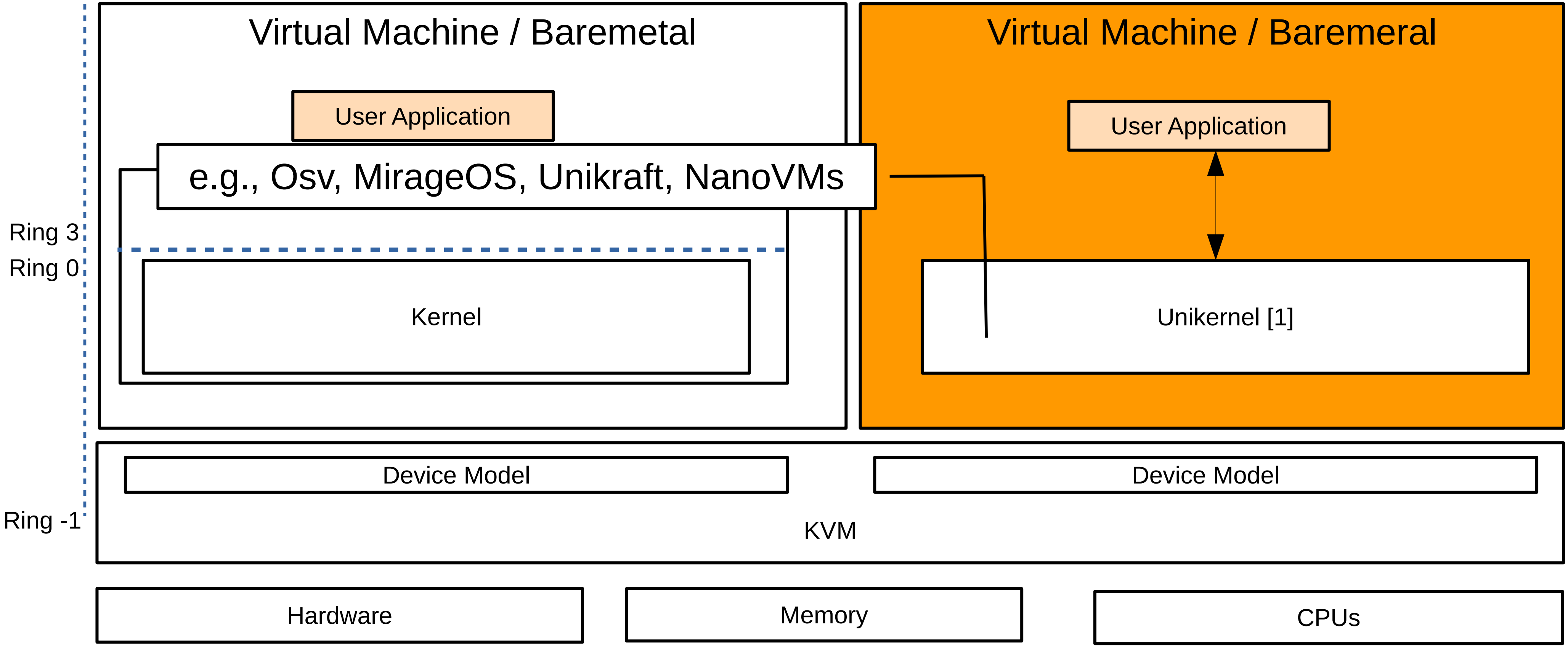


Virtual Machine / Baremetal
User Application
Operating System
Ring 3
Ring 0
Scheduler
Kernel
Drivers
Filesystem
Device Model
Ring -1
KVM
Hardware
Memory
CPUs

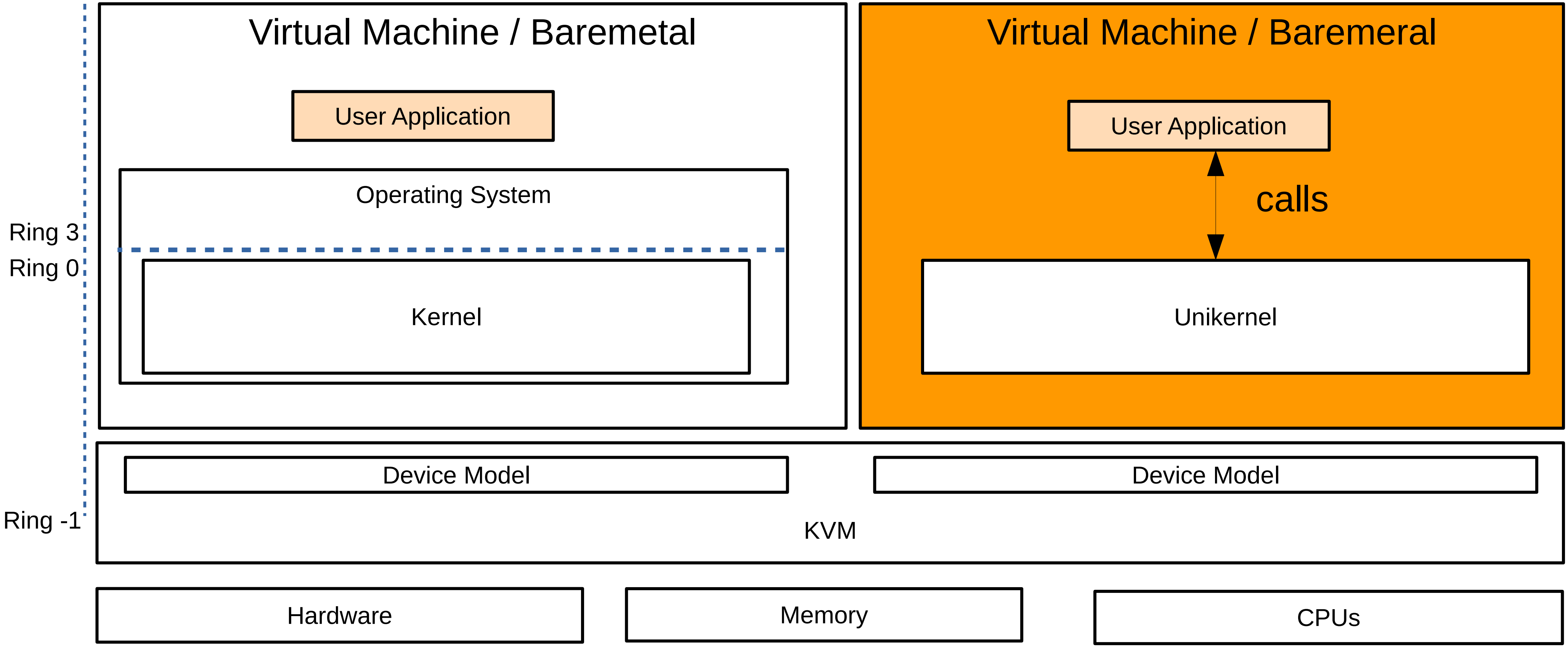


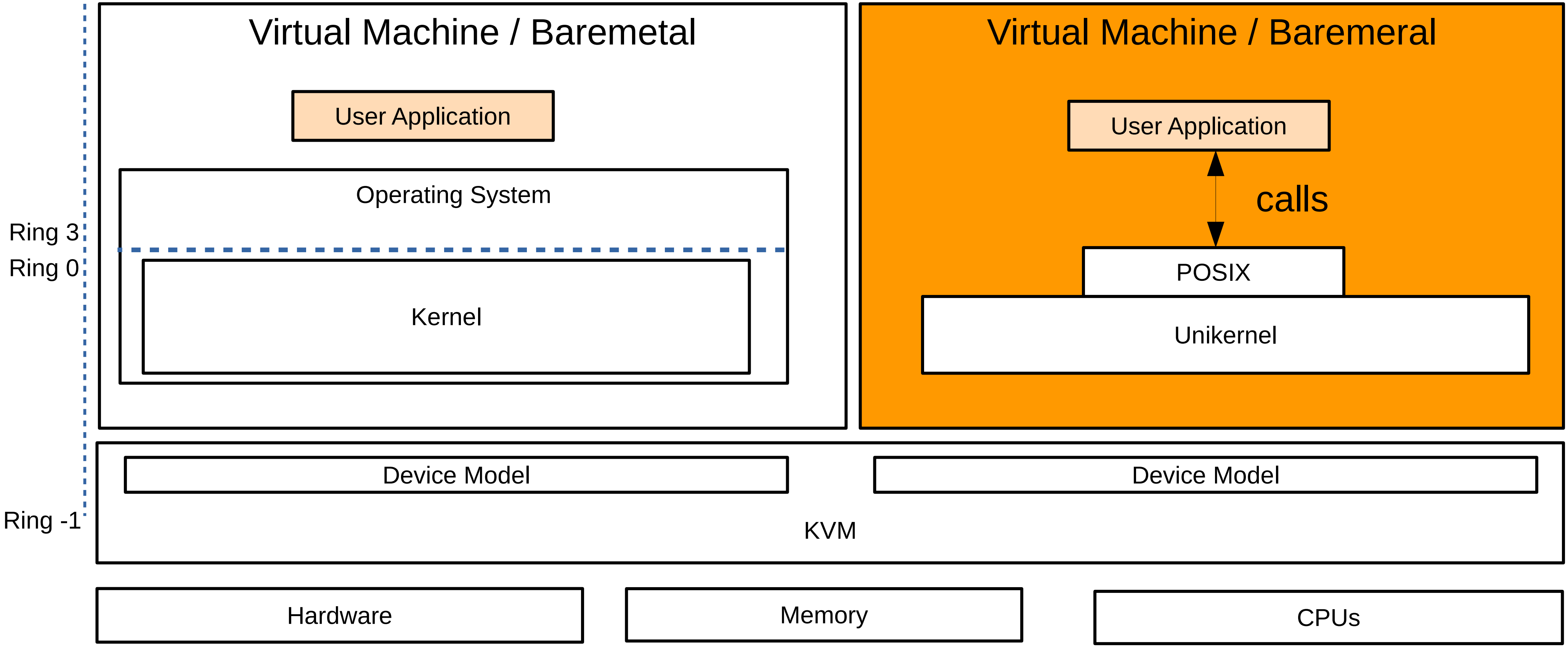
Virtual Machine / Baremetal
User Application
Operating System
Ring 3
Ring 0
Scheduler
Filesystem
Kernel
Drivers
Device Model
Ring -1
KVM
Hardware
Memory
CPUs

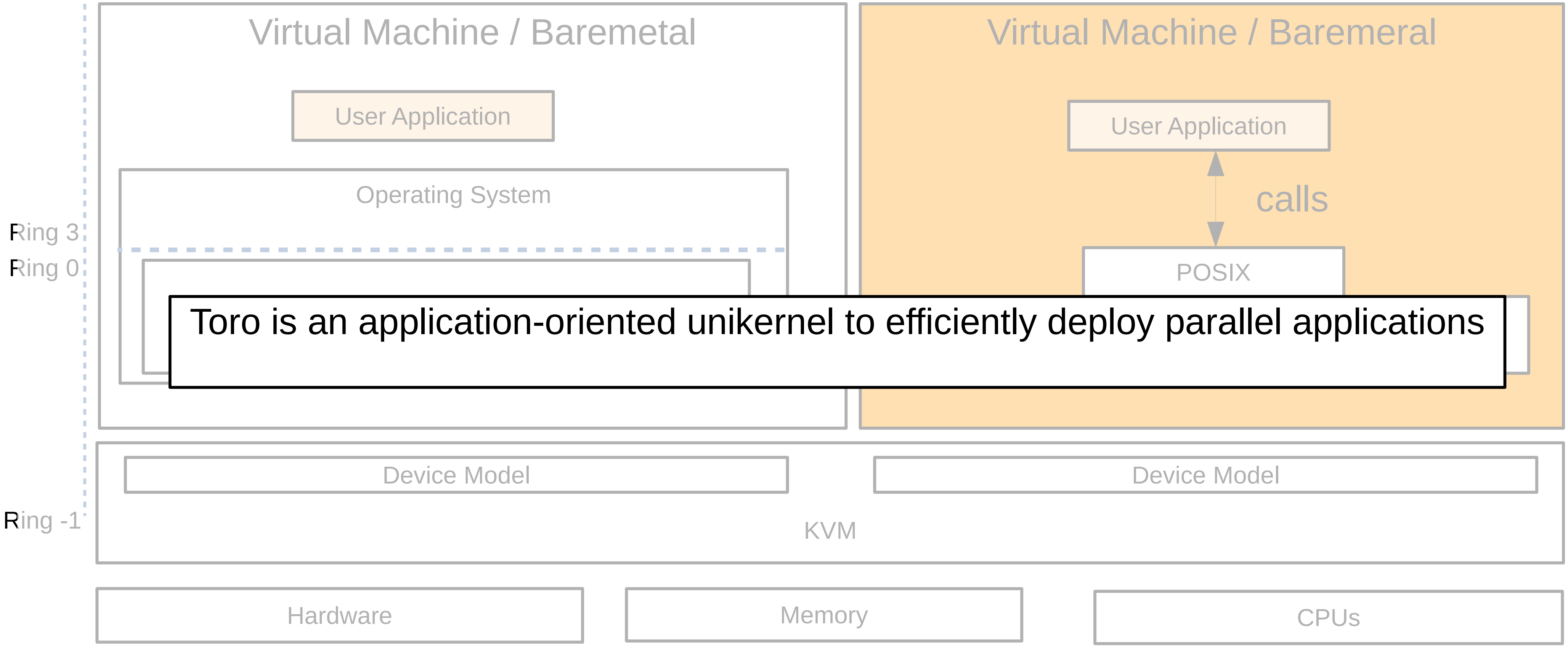

Virtual Machine / Baremetal
User Application
Operating System
Ring 3
Ring 0
Scheduler
Filesystem
Kernel
Too general for a single purpose application!
Device Model
Ring -1
KVM
Hardware
Memory
CPUs



[1] “Unikernels: library operating systems for the cloud”, Madhavapeddy et al., 2013
[2] “Unikernels: the next stage of Linux’s dominance”, Ali Raza et al., 2019


Virtual Machine / Baremetal
User Application
Operating System
Ring 3
Ring 0
Kernel
Virtual Machine / Baremeral
User Application
calls
Unikernel
Device Model
Device Model
Ring -1
KVM
Hardware
Memory
CPUs

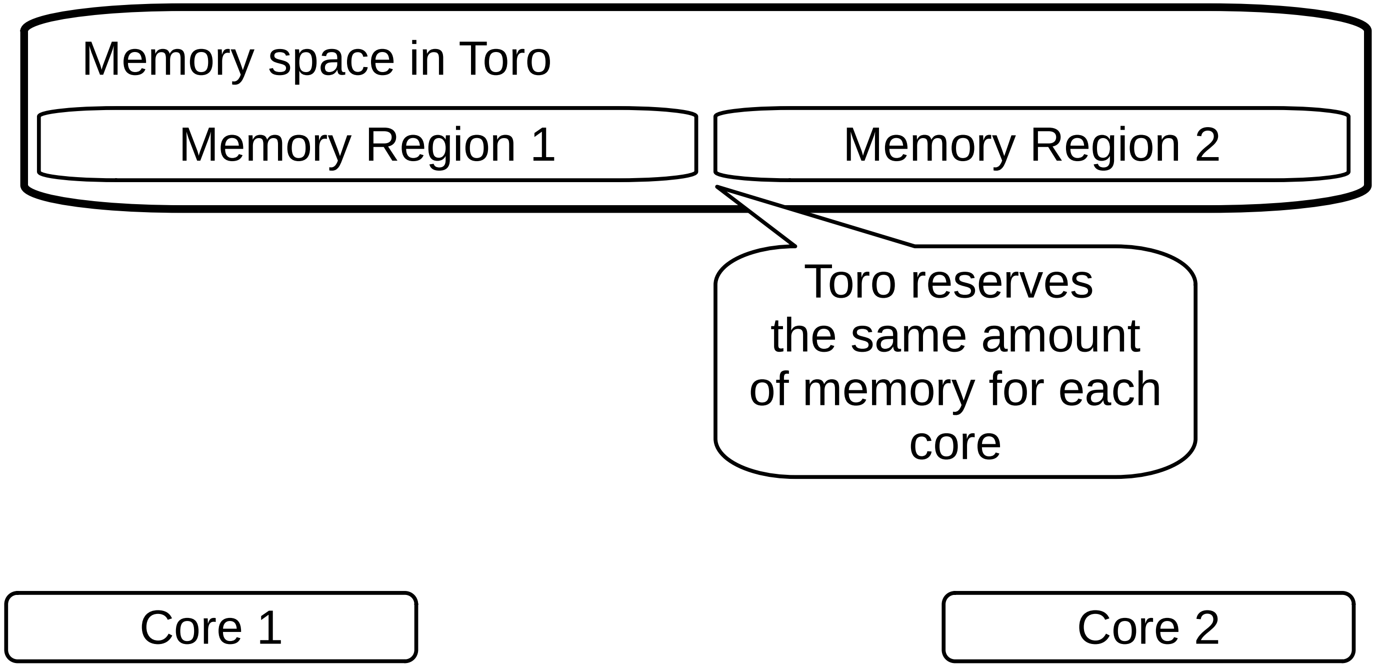

Virtual Machine / Baremetal
User Application
Operating System
Ring 3
Ring 0
Kernel
Virtual Machine / Baremeral
User Application
calls
POSIX
Unikernel
Device Model
Device Model
Ring -1
KVM
Hardware
Memory
CPUs


Virtual Machine / Baremetal
User Application
Operating System
Ring 3
Ring 0
Virtual Machine / Baremeral
User Application
calls
POSIX
Toro is an application-oriented unikernel to efficiently deploy parallel applications
Device Model
Device Model
Ring -1
KVM
Hardware
Memory
CPUs

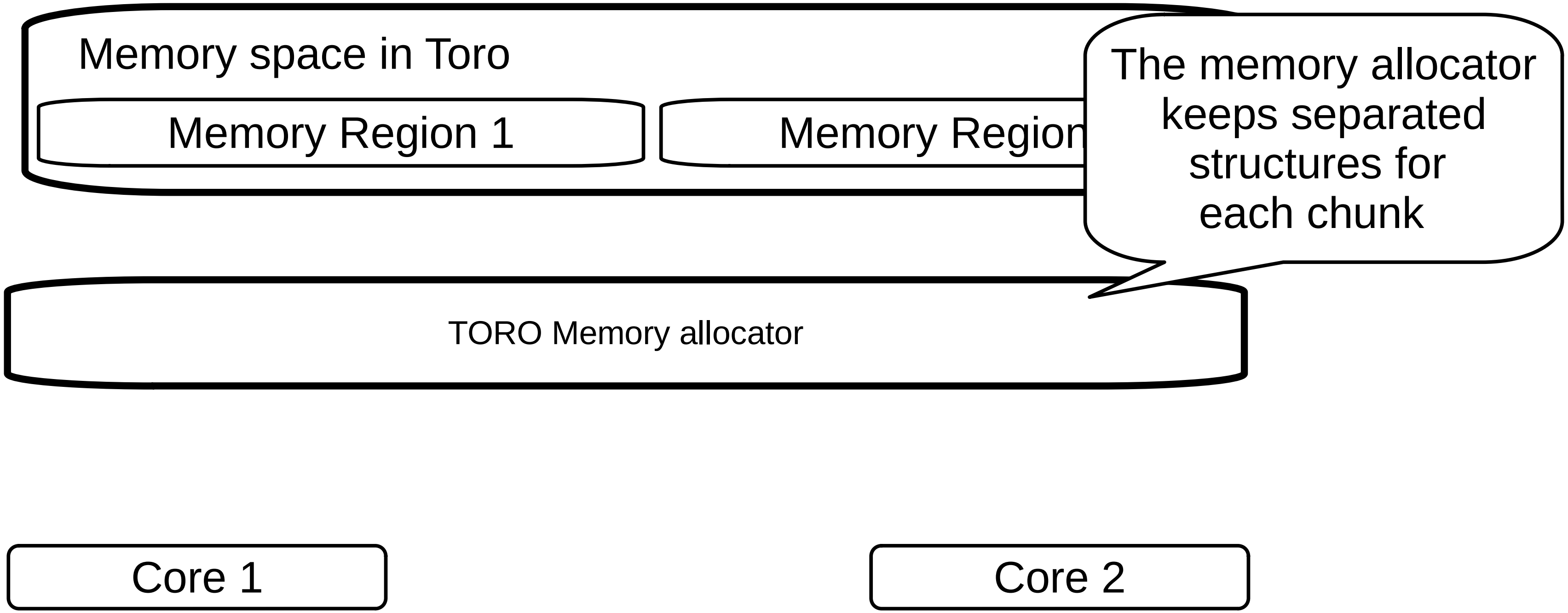
How does Toro leverage multicore?

- Memory per core
- Cooperative Scheduler
- Core to Core communication based on VirtIO

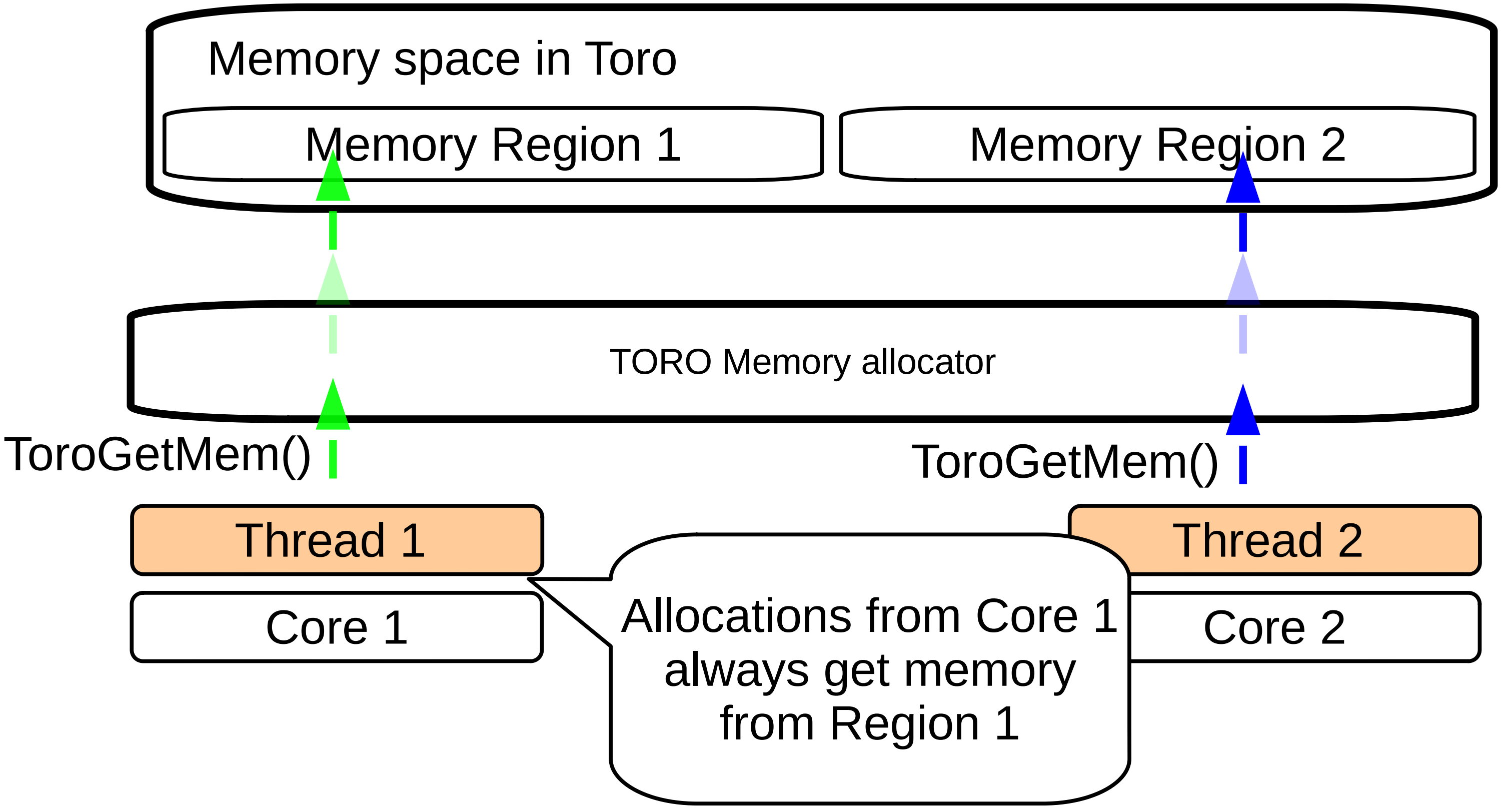
Dedicated Memory



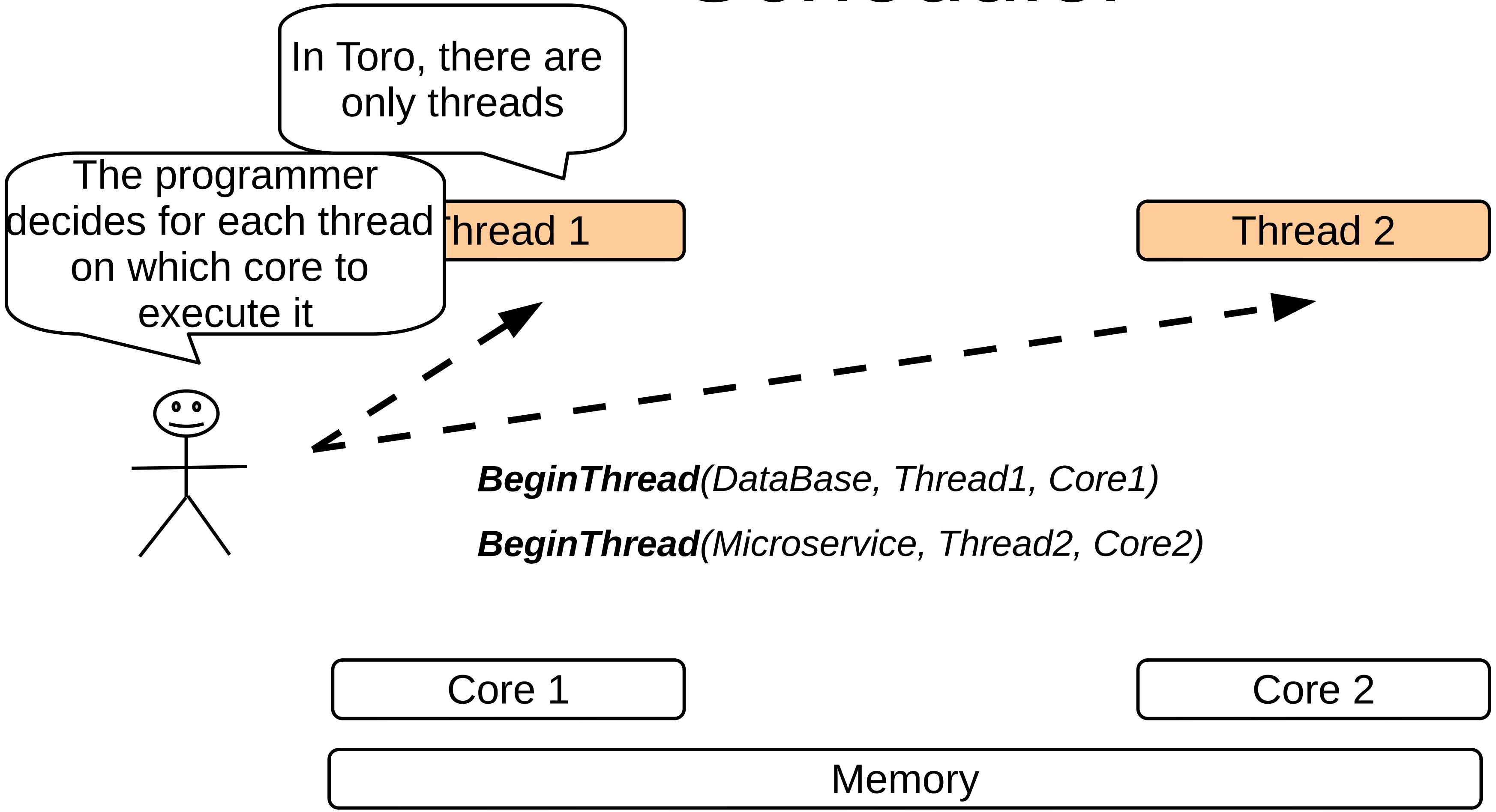
Dedicated Memory



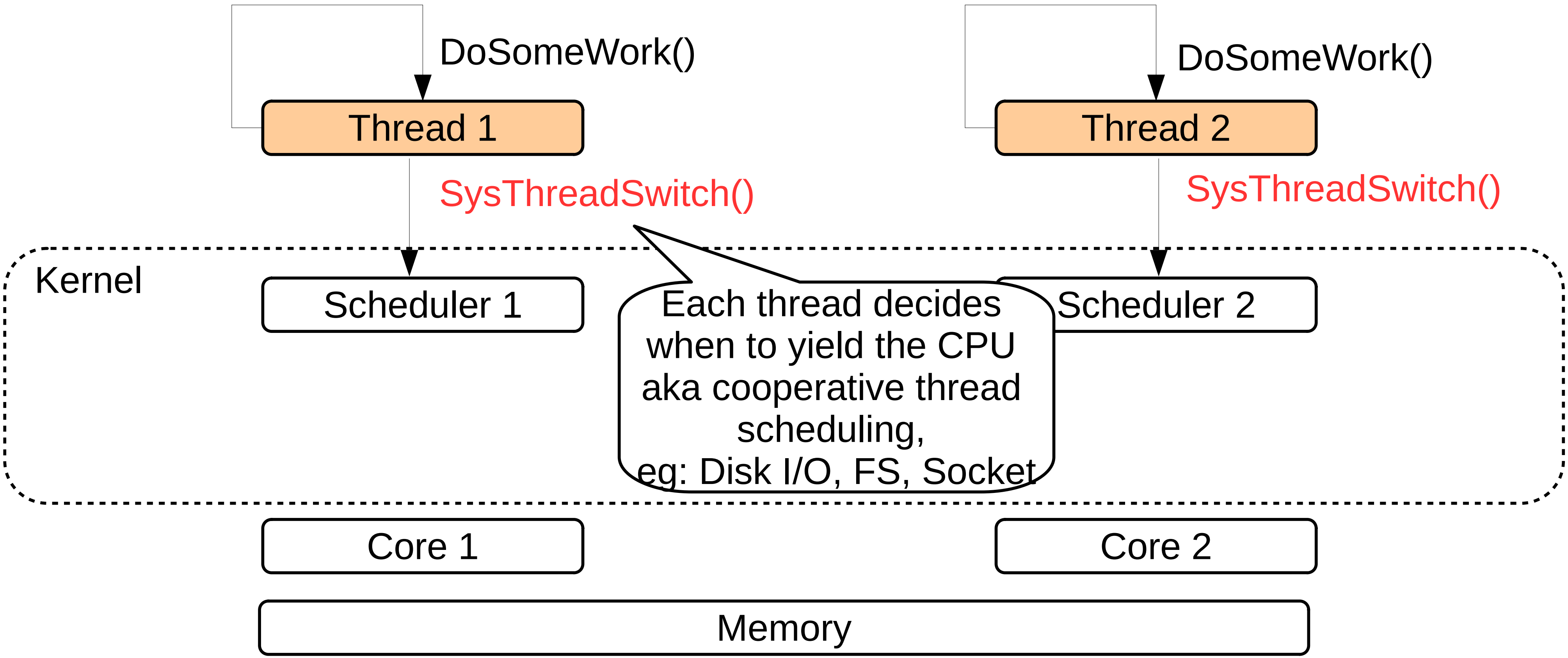
Dedicated Memory



Scheduler

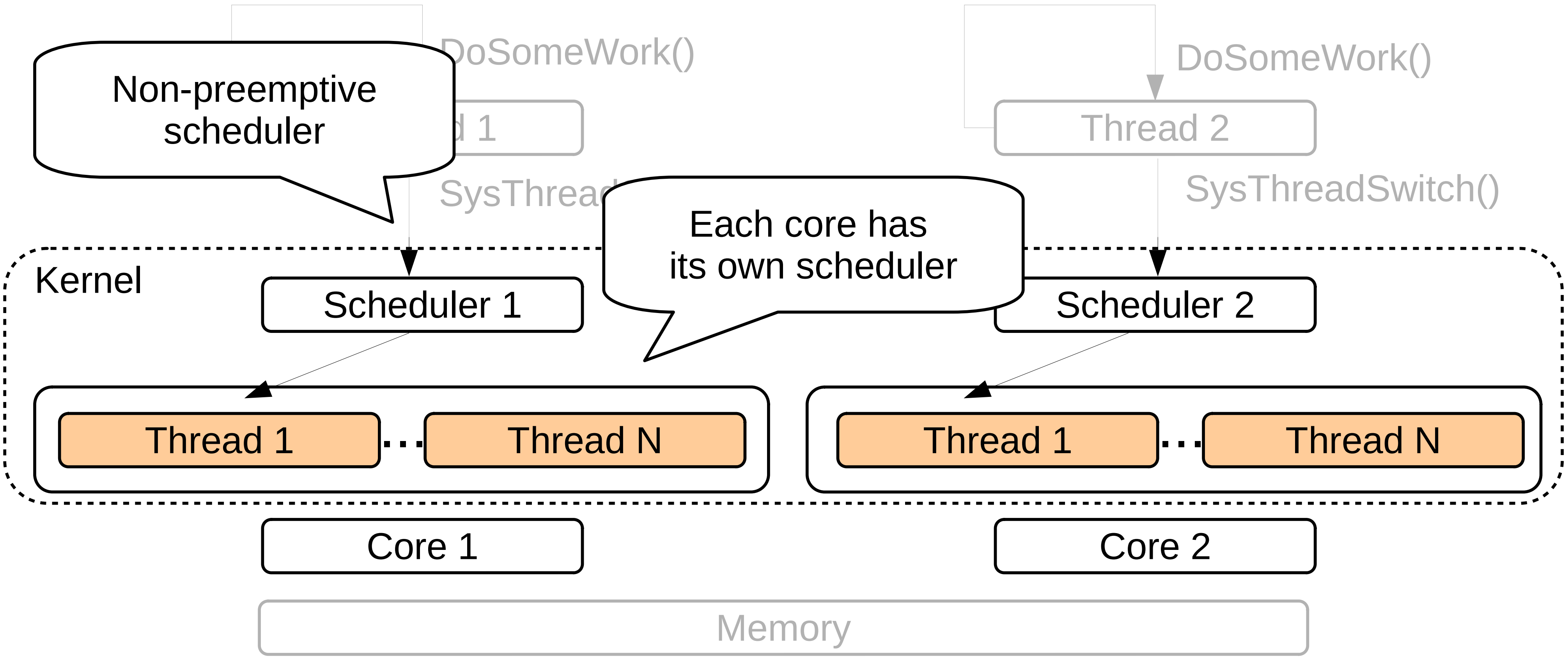

In Toro, there are only threads
The programmer decides for each thread on which core to execute it
hread 1
Thread 2
BeginThread*(DataBase, Thread1, Core1)*
BeginThread*(Microservice, Thread2, Core2)*
Core 1
Core 2
Memory

Scheduler



DoSomeWork()
Thread 1
DoSomeWork()
Thread 2
SysThreadSwitch()
SysThreadSwitch()
Kernel
Scheduler 1
Scheduler 2
Each thread decides when to yield the CPU aka cooperative thread scheduling, eg: Disk I/O, FS, Socket
Core 1
Core 2
Memory

Scheduler

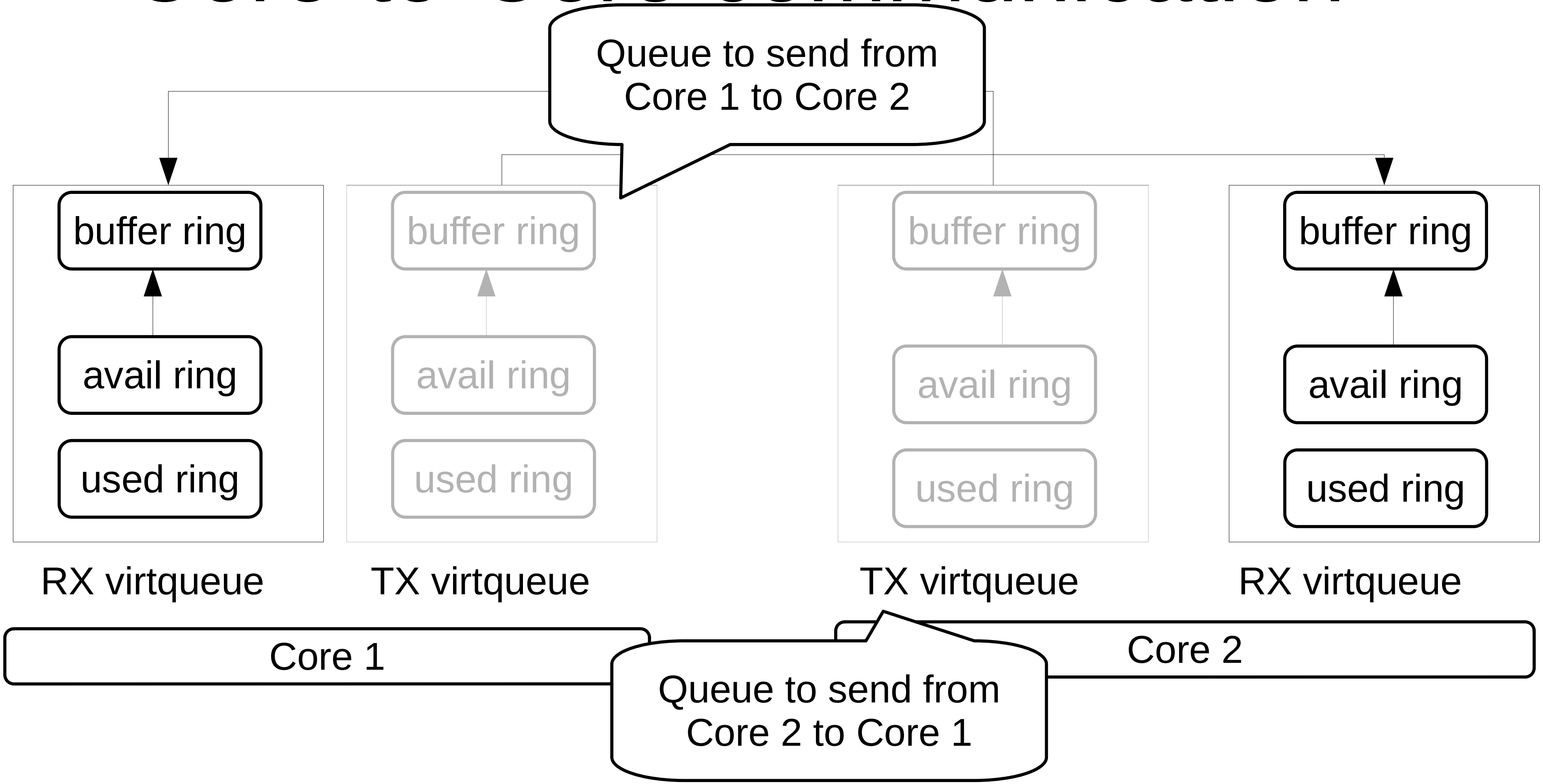


DoSomeWork()
Non-preemptive scheduler
d 1
SysThread
DoSomeWork()
Thread 2
SysThreadSwitch()
Each core has its own scheduler
Kernel
Scheduler 1
Scheduler 2
Thread 1
Thread N
Thread 1
Thread N
Core 1
Core 2
Memory

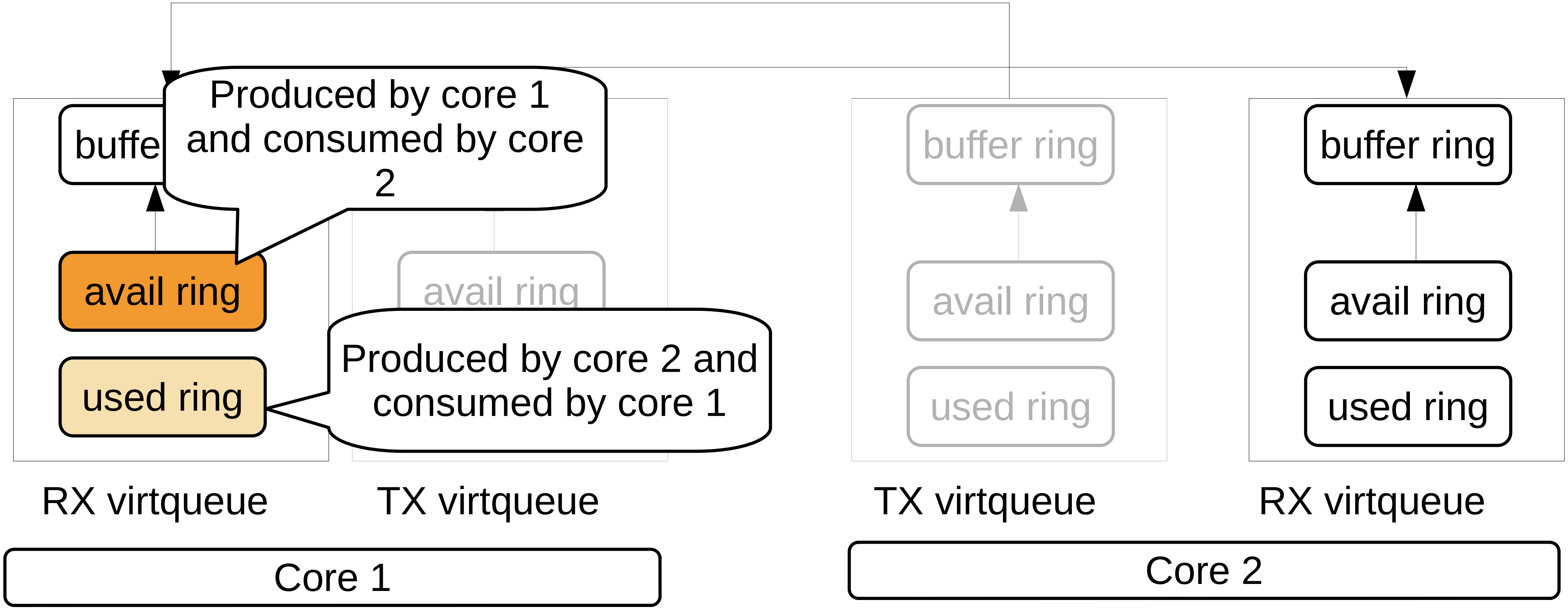
Core-to-Core communication

- Each core can communicate with any other core by using dedicated queues
- It is based on two primitives:
  - procedure SendTo(Core: DWORD; Buffer: Pointer; Len: DWORD);
  - procedure RecvFrom(Core: DWORD; Buffer: Pointer);
- These are the ingredients to implement MPI_Gather(), MPI_Bcast() and MPI_Scatter()

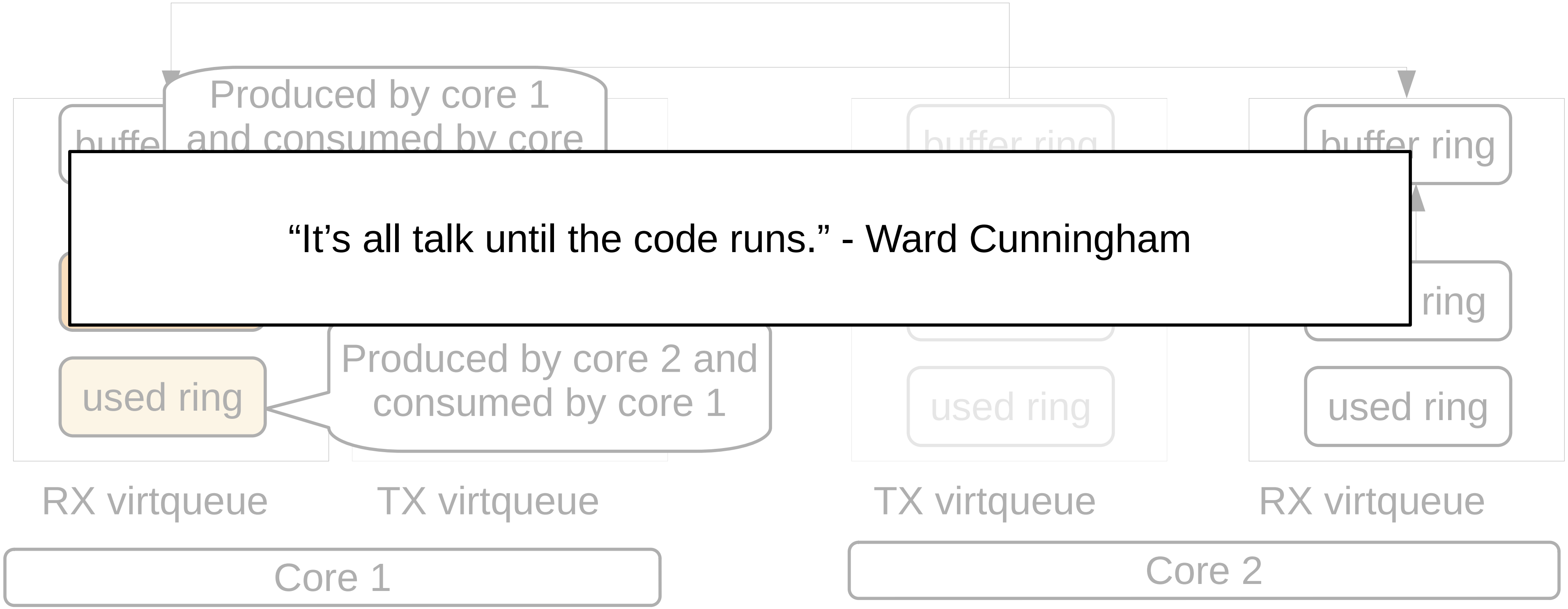
Core-to-Core communication



Core-to-Core communication

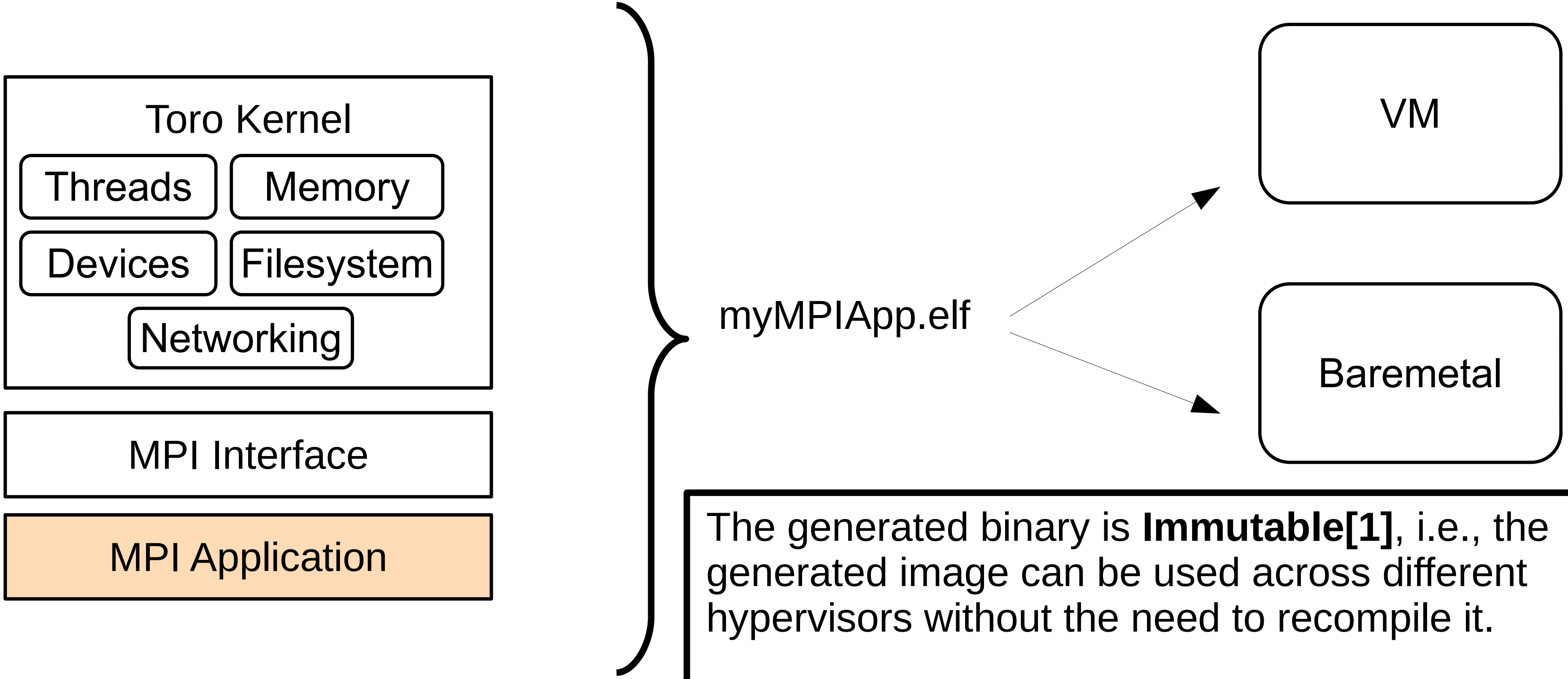


Core-to-Core communication



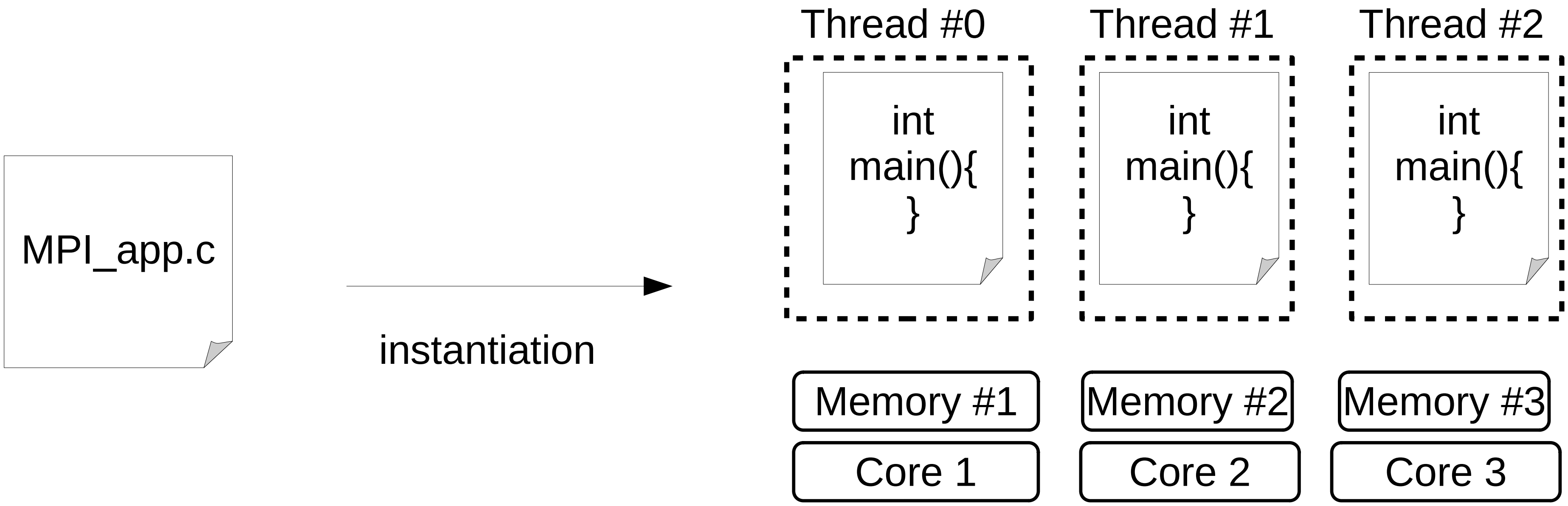
“It’s all talk until the code runs.” - Ward Cunningham

How a MPI application is deployed?



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

How a MPI application is deployed?

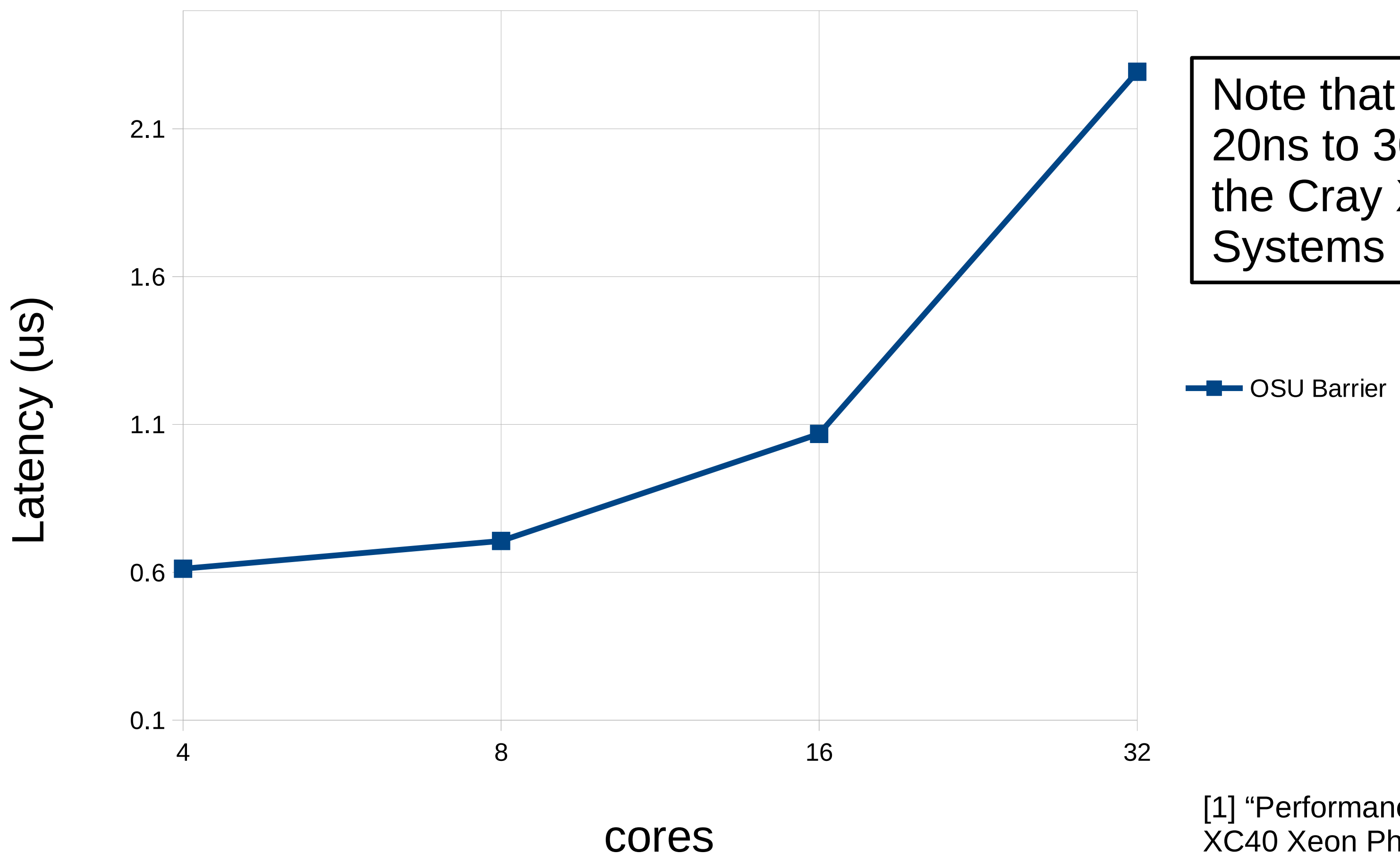


Benchmarking

- I benchmark it by using the OSU MPI_Barrier (see OSU microbenchmarks[1]) that measures the latency of the MPI_Barrier() function for a given number of nodes
- I deploy it by using a single VM (QEMU microvm/KVM) with 4, 8, 16 and 32 cores
- I run it on a 1 x Intel Xeon Gold 6314U, 32 cores @ 2.3 GHz

[1] https://mvapich.cse.ohio-state.edu/benchmarks/
[2] https://github.com/torokernel/torokernel/tree/features-mpi/examples/MPI

OSU MPI_Barrier



Note that [1] reports between 20ns to 30ns for 16 nodes in the Cray XC40 Xeon Phi Systems

[1] “Performance Evaluation of MPI on Cray XC40 Xeon Phi Systems”

Questions?

Thanks!

> Toro kernel is open source on GitHub: https://github.com/torokernel/torokernel

> Follow me on Twitter: https://twitter.com/ToroKernel

> Sponsor me on GitHub: https://github.com/sponsors/MatiasVara

> Watch me on Youtube: https://www.youtube.com/@torokernel3078