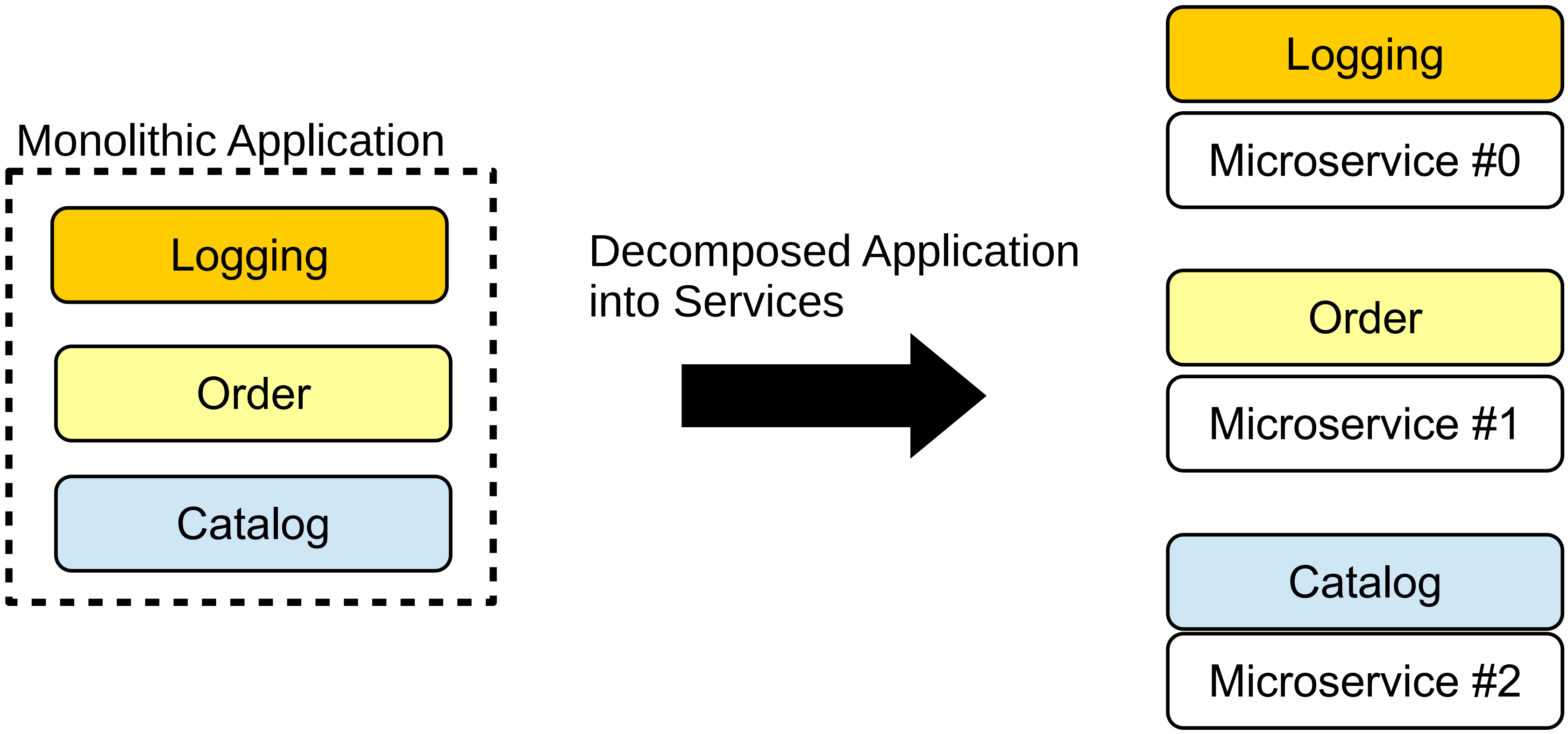
Developing and Deploying Microservices with Toro Kernel

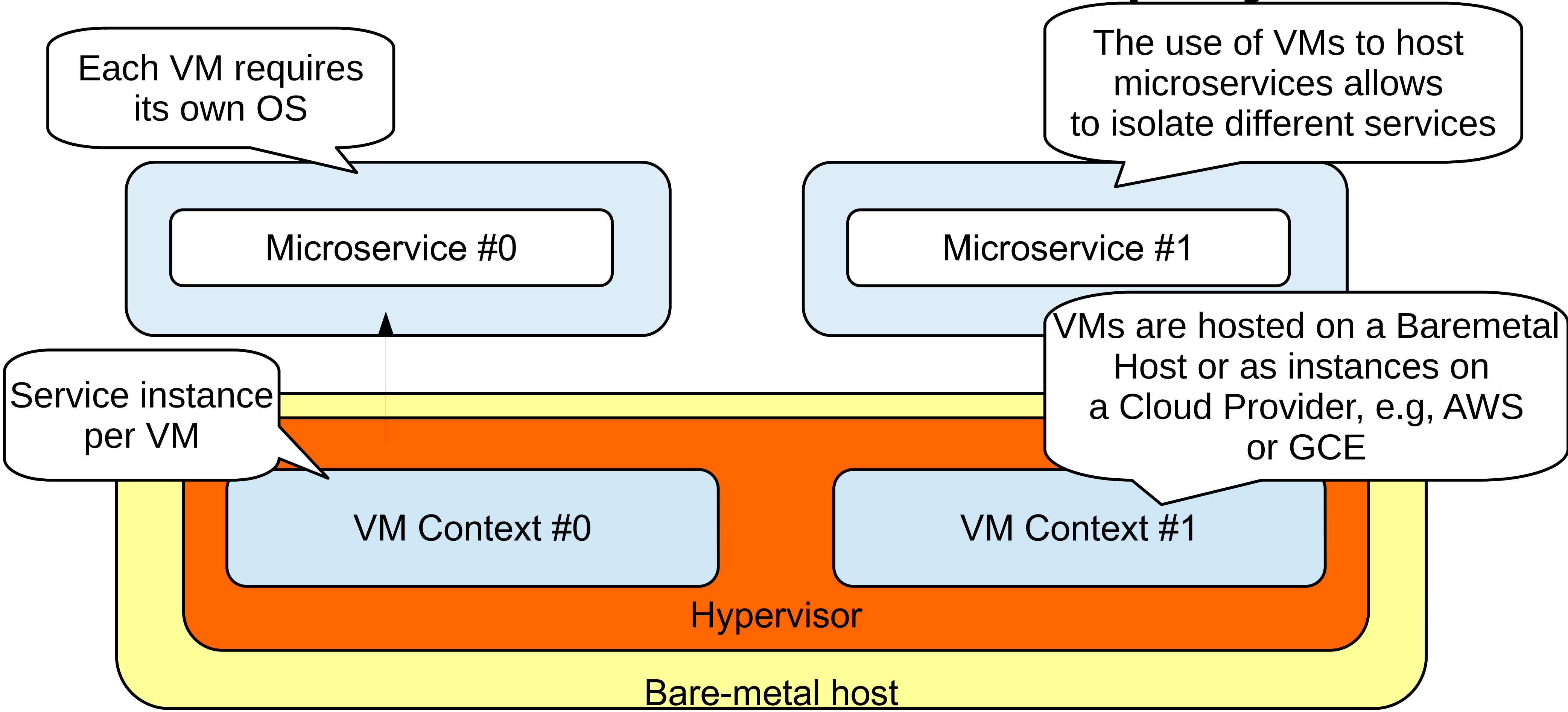
www.torokernel.io

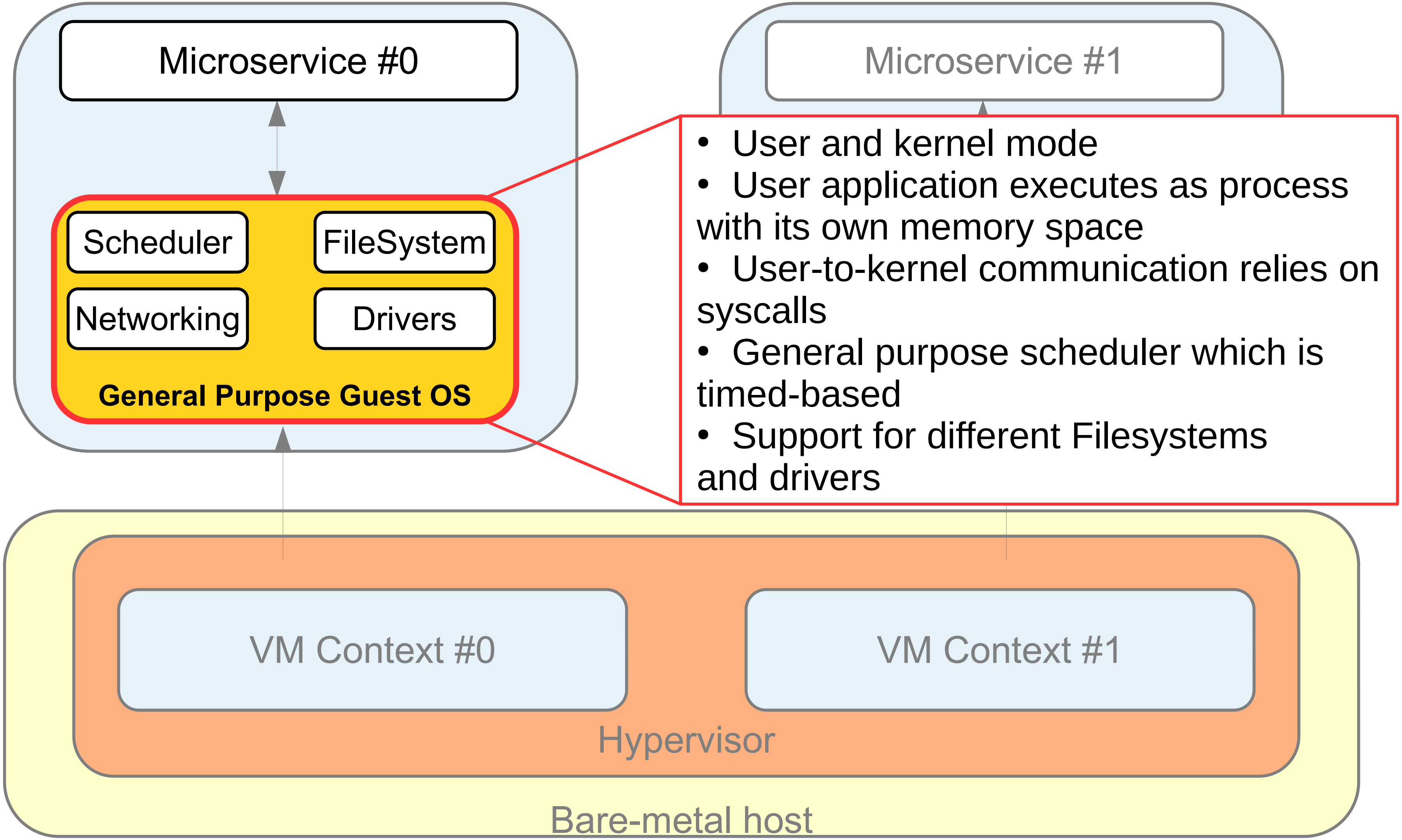
Matias Vara Larsen
matiasevara@gmail.com

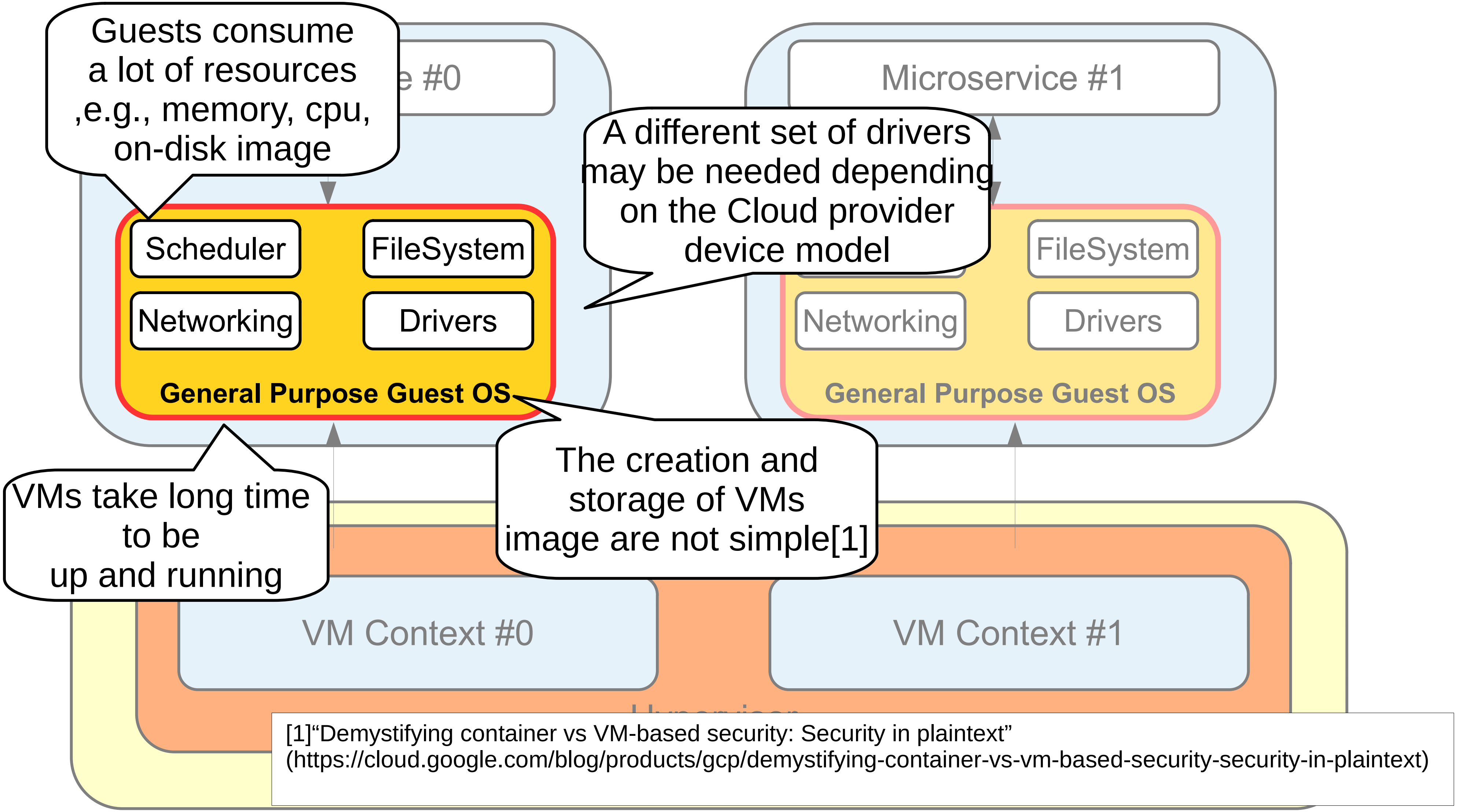
What are microservices?



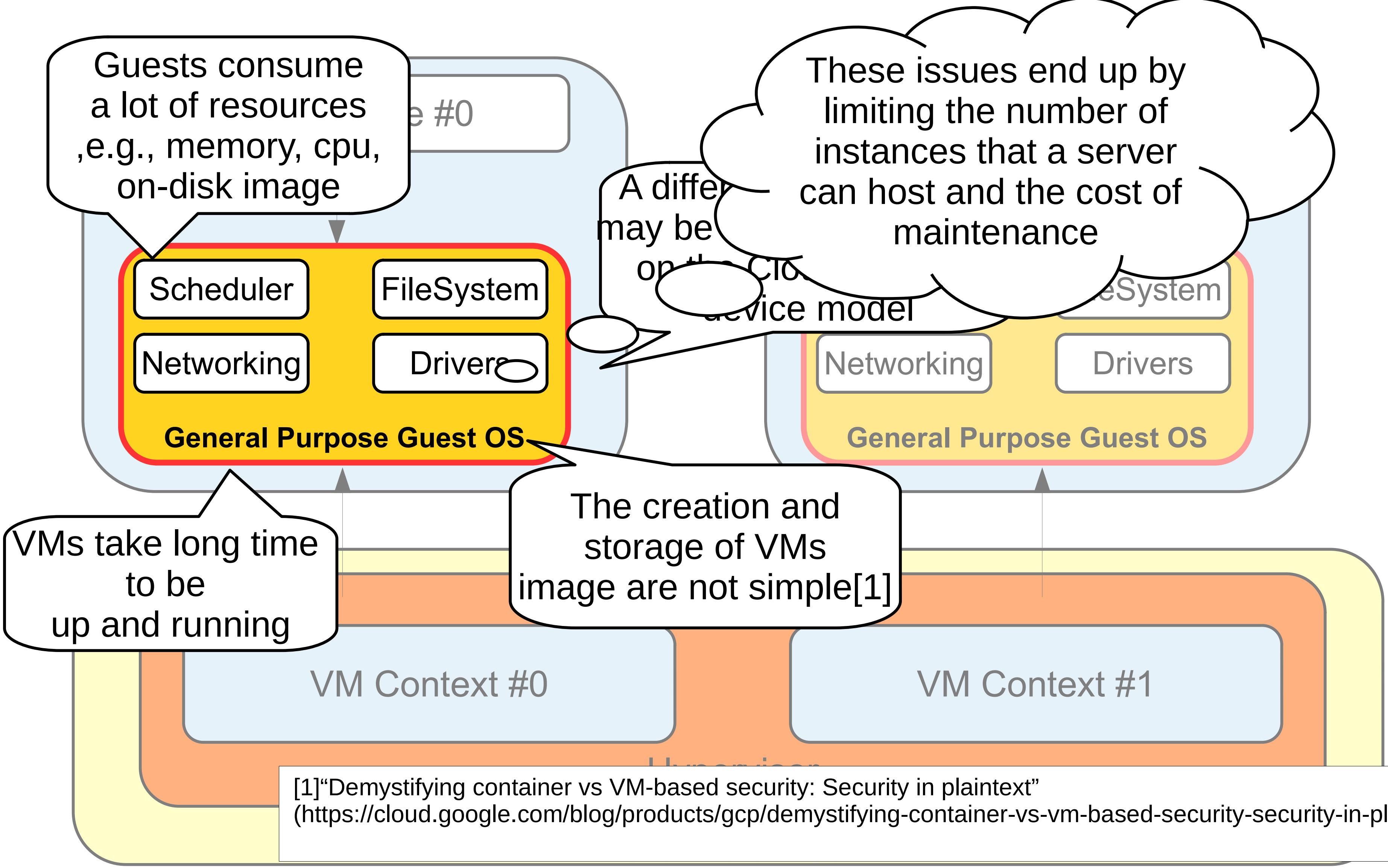
How are microservices deployed?



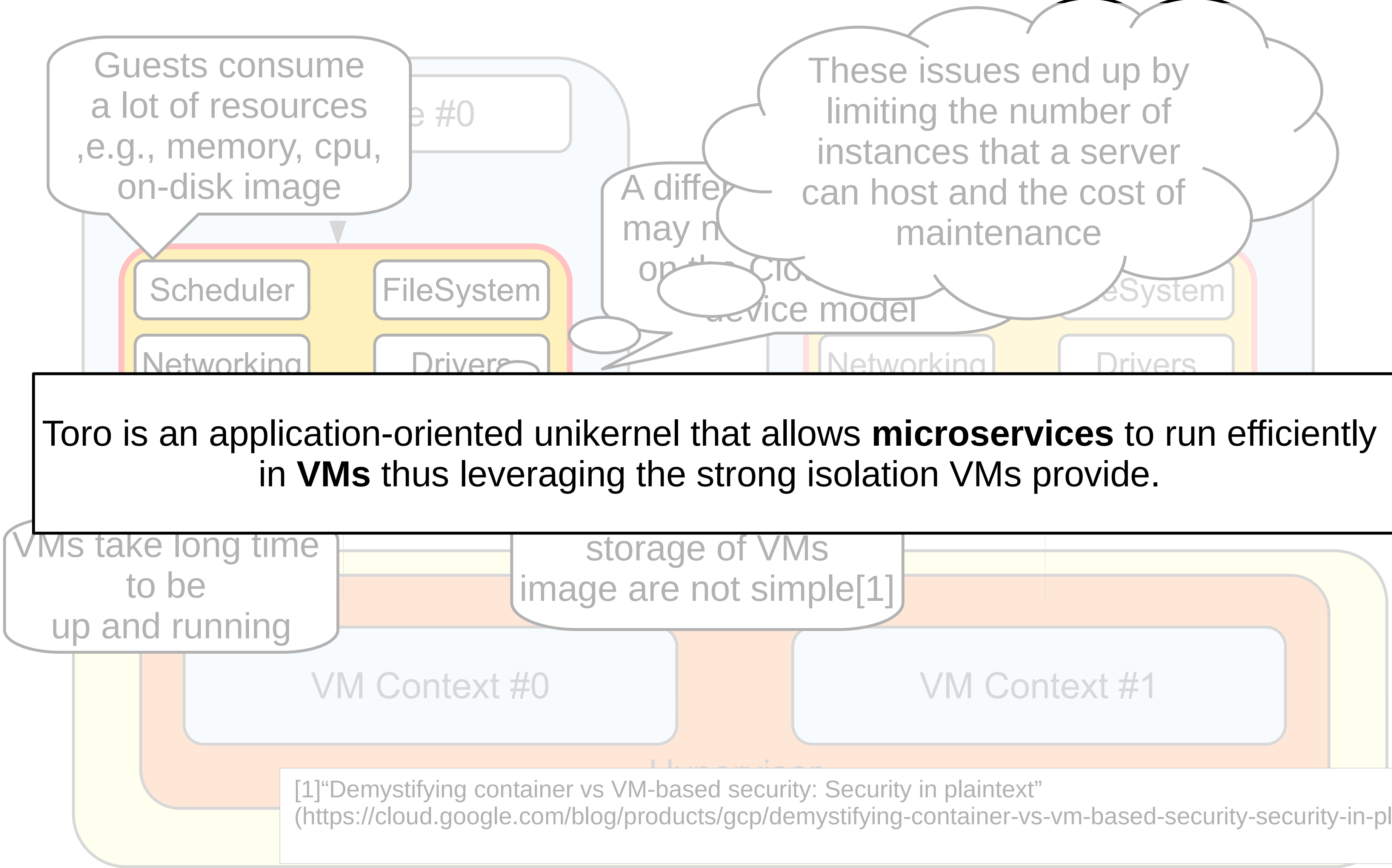

Microservice #0
Scheduler
FileSystem
Networking
Drivers
General Purpose Guest OS
Microservice #1
• User and kernel mode
• User application executes as process with its own memory space
• User-to-kernel communication relies on syscalls
• General purpose scheduler which is timed-based
• Support for different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

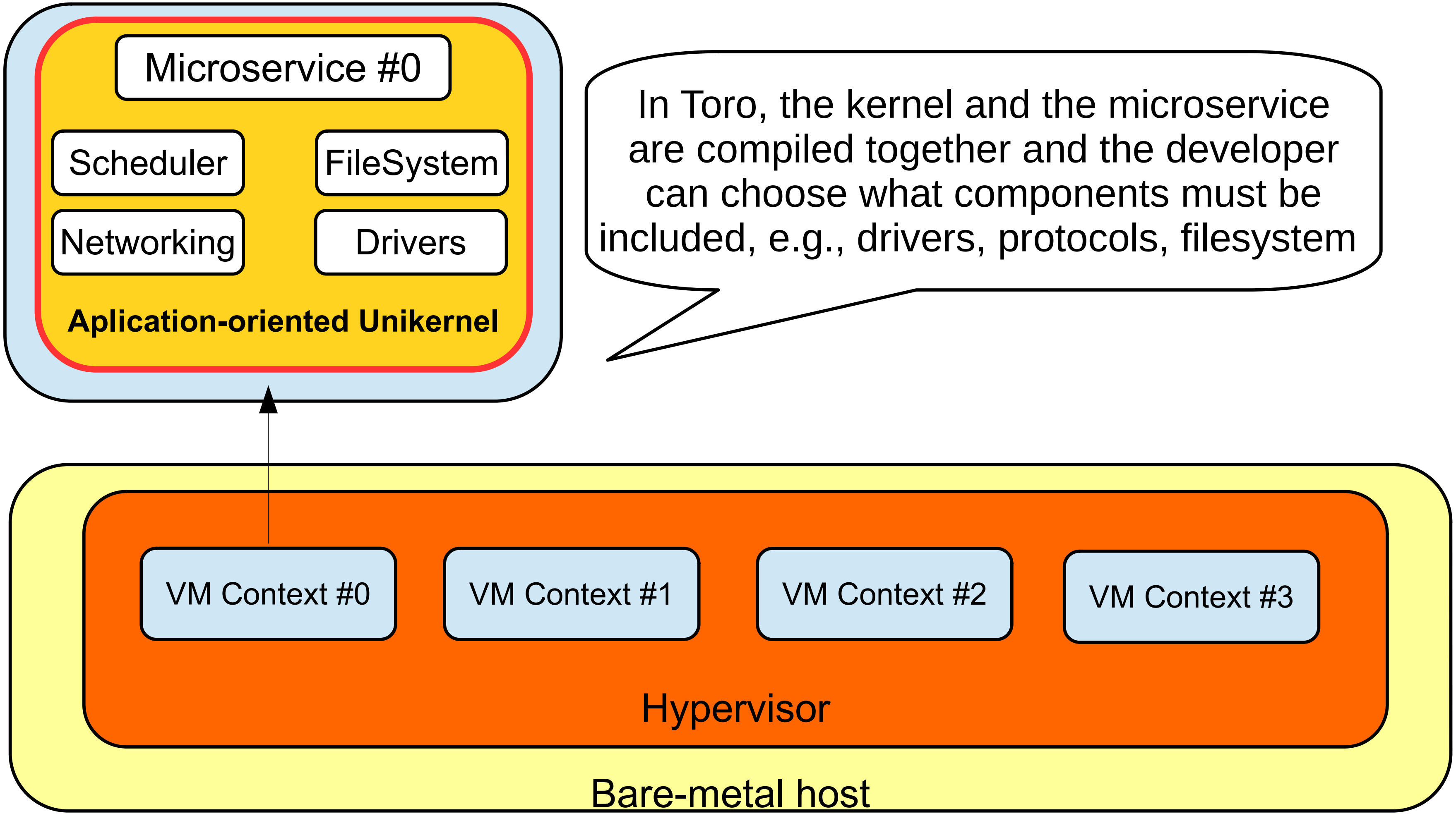


[1]“Demystifying container vs VM-based security: Security in plaintext” (https://cloud.google.com/blog/products/gcp/demystifying-container-vs-vm-based-security-security-in-plaintext)



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

[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
Scheduler
FileSystem
Networking
Drivers
Aplication-oriented Unikernel
In Toro, the kernel and the microservice are compiled together and the developer can choose what components must be included, e.g., drivers, protocols, filesystem
VM Context #0
VM Context #1
VM Context #2
VM Context #3
Hypervisor
Bare-metal host

Ingredients for ToroKernel

- A kernel’s architecture that improves the execution of a single user application which is based on:
  - Simple Scheduler
  - Dedicated Resources
- A kernel’s architecture that improves the execution of a single user application as a guest by providing:
  - Fast instantiation of user application
  - Reduced guest’s footprint
  - Improved netwoking and filesystem by relying on VirtIO devices

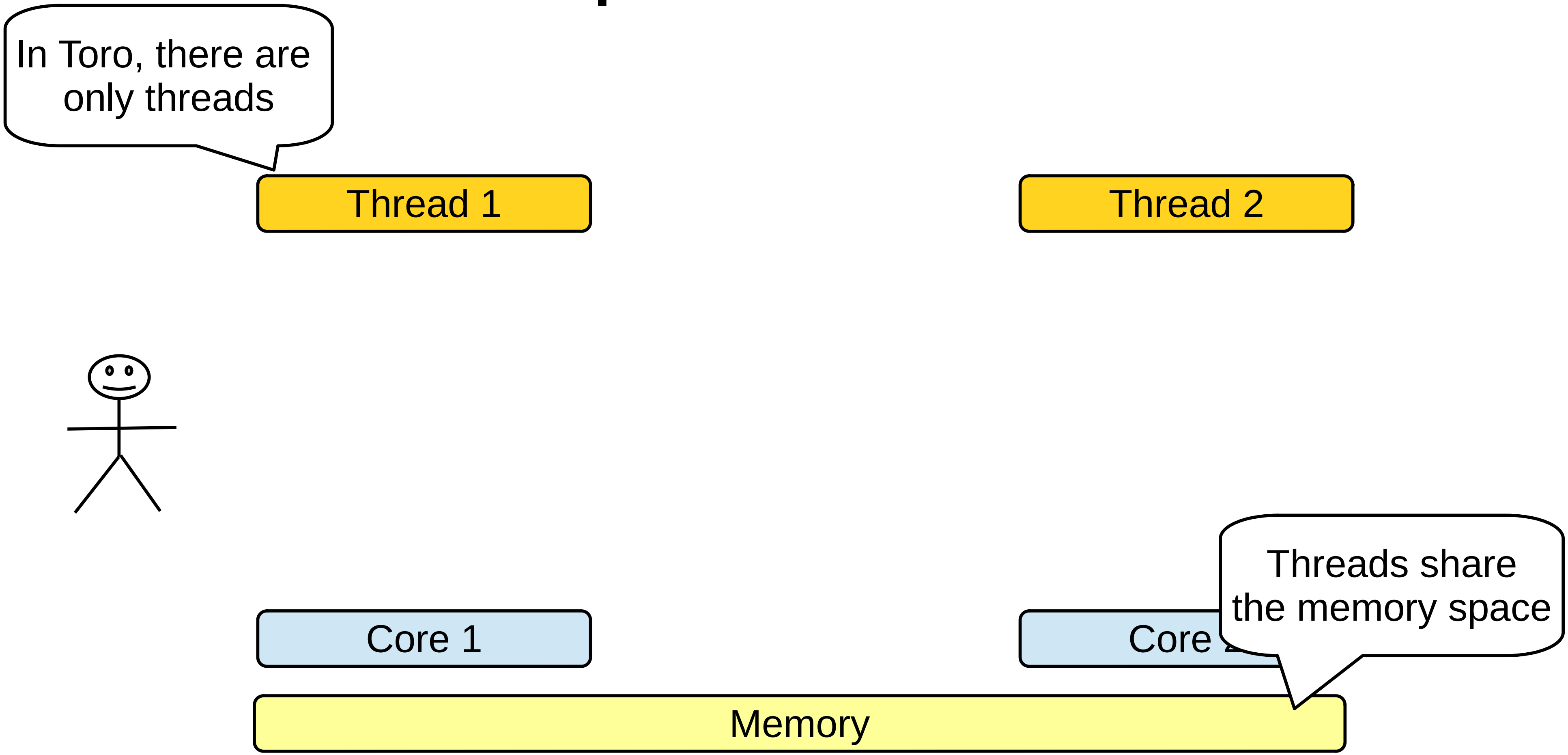
Ingredients for ToroKernel

- A kernel’s architecture that improves the execution of a single user application which is based on:
  - Simple Scheduler
  - Dedicated Resources
- A kernel’s architecture that improves the execution of a single user application as a guest by providing:
  - Fast instantiation of user application
  - Reduced guest’s footprint
  - Improved netwoking and filesystem by relying on VirtIO devices

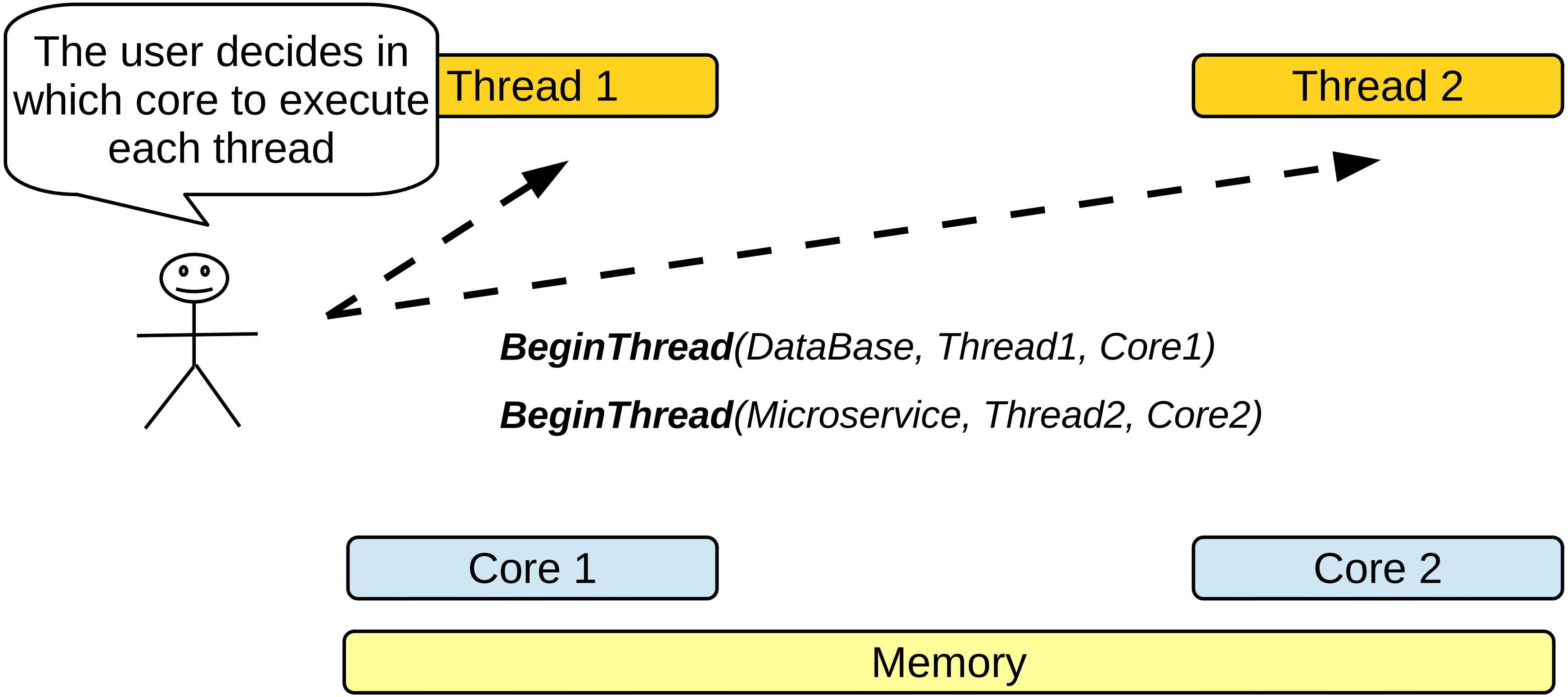
Simple Scheduler Requirements

- Scale with the number of cores
- Simple scheduler’s algorithm, i.e., Cooperative threading model
- Minimalist context switches

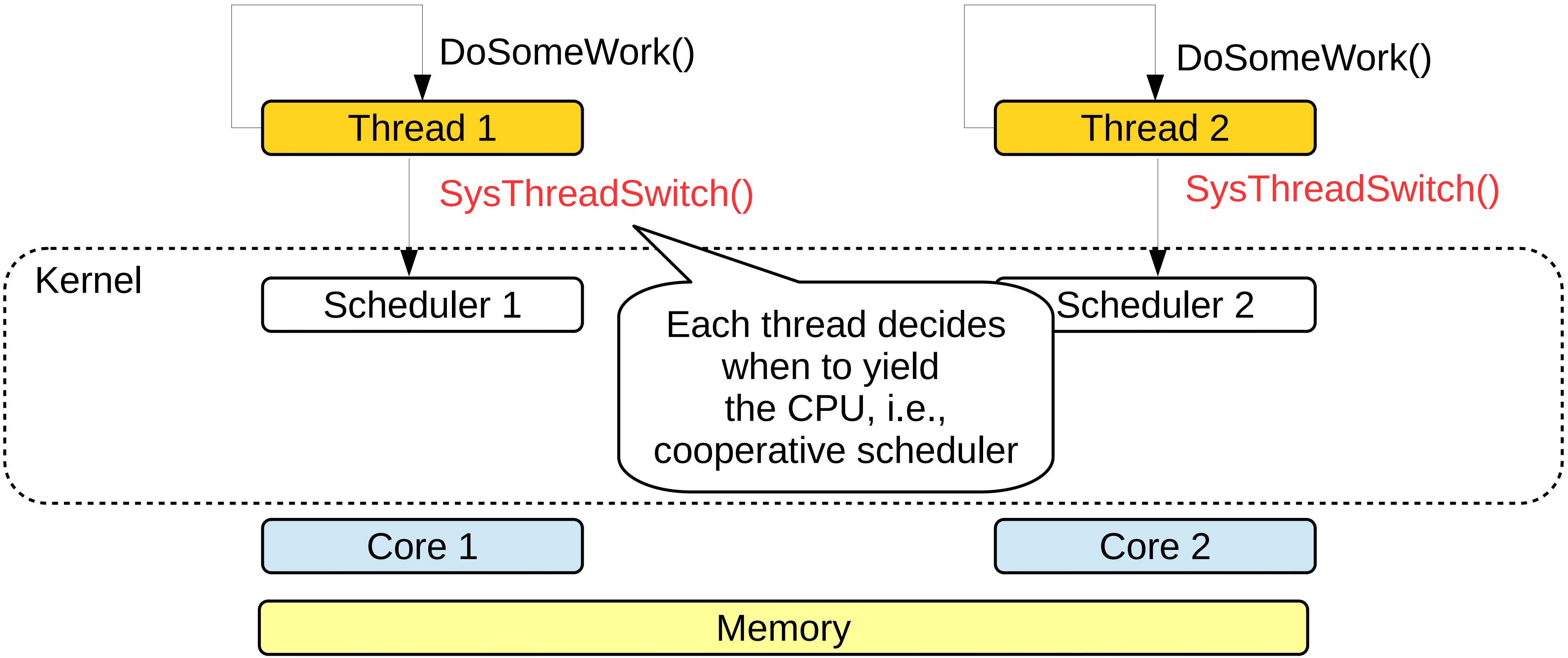
Simple Scheduler



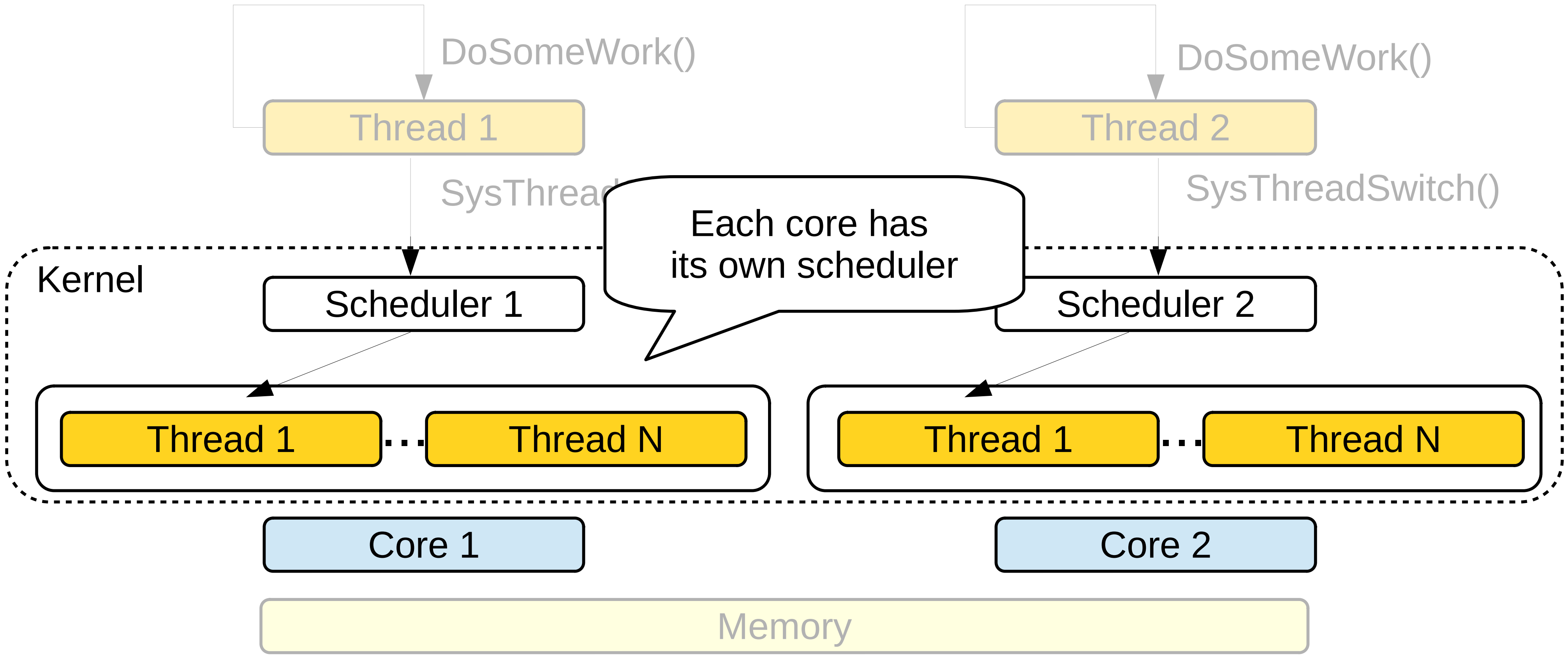
Simple Scheduler



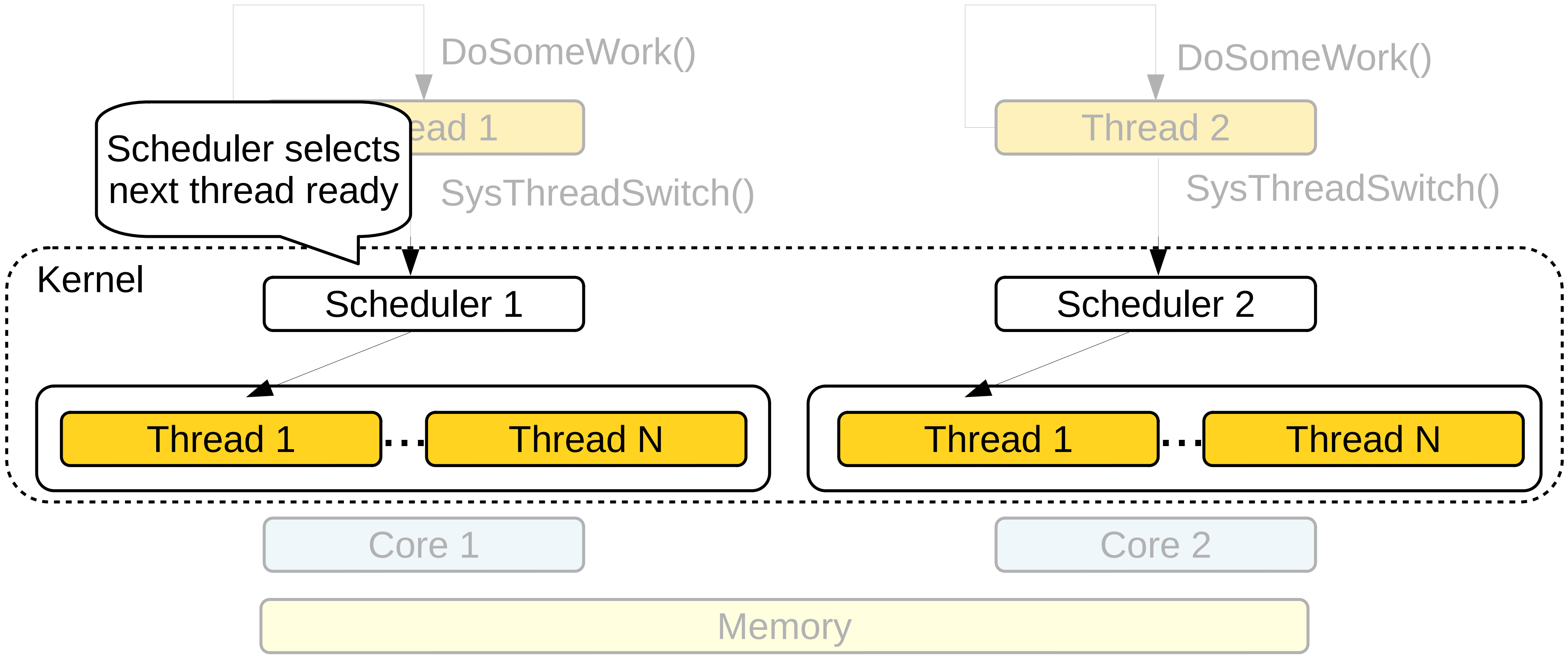
Simple Scheduler



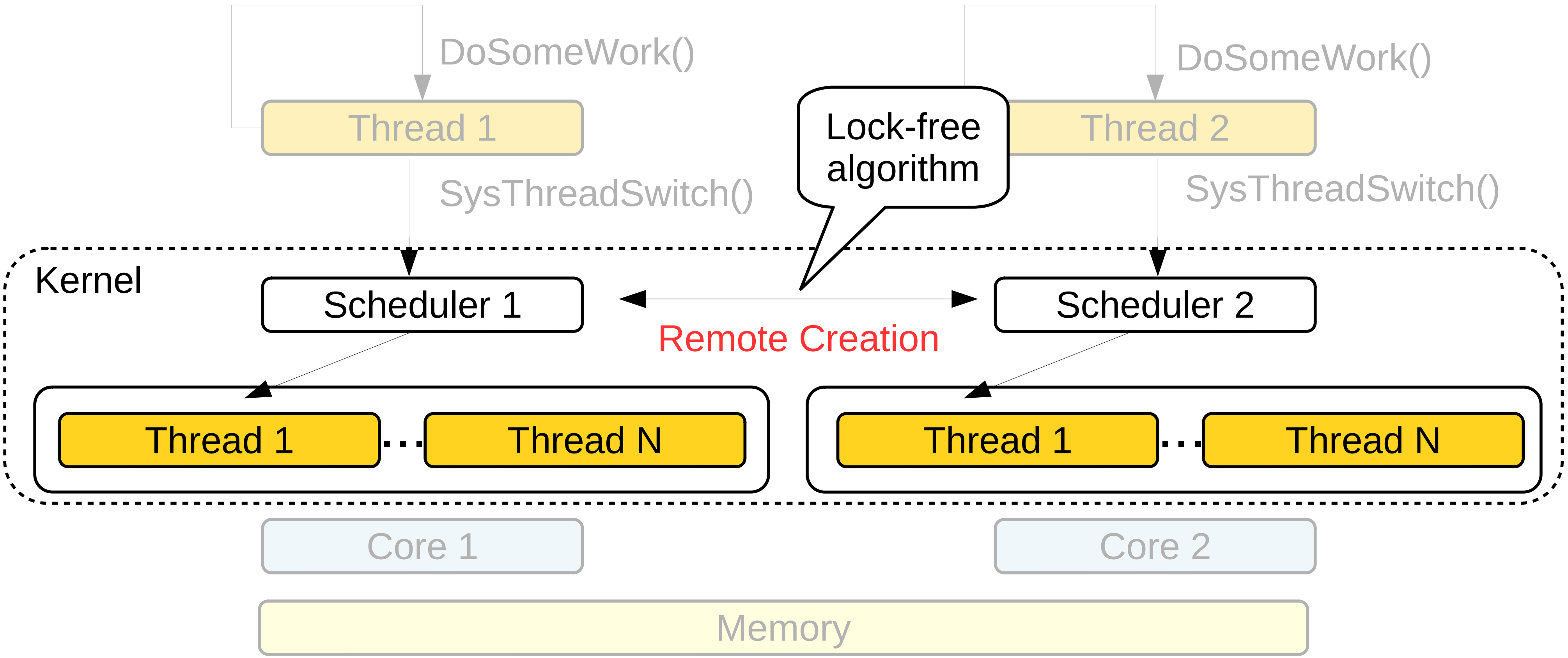
Simple Scheduler



Simple Scheduler



Simple Scheduler



Benefits of Simple Scheduler

- The use of threads makes context switching cheaper
- The use of a cooperative scheduler:
  - simplifies user’s and kernel’s code by avoiding to implement protection inside the scheduler
  - reduces number of context switches since it does not relies on interruptions

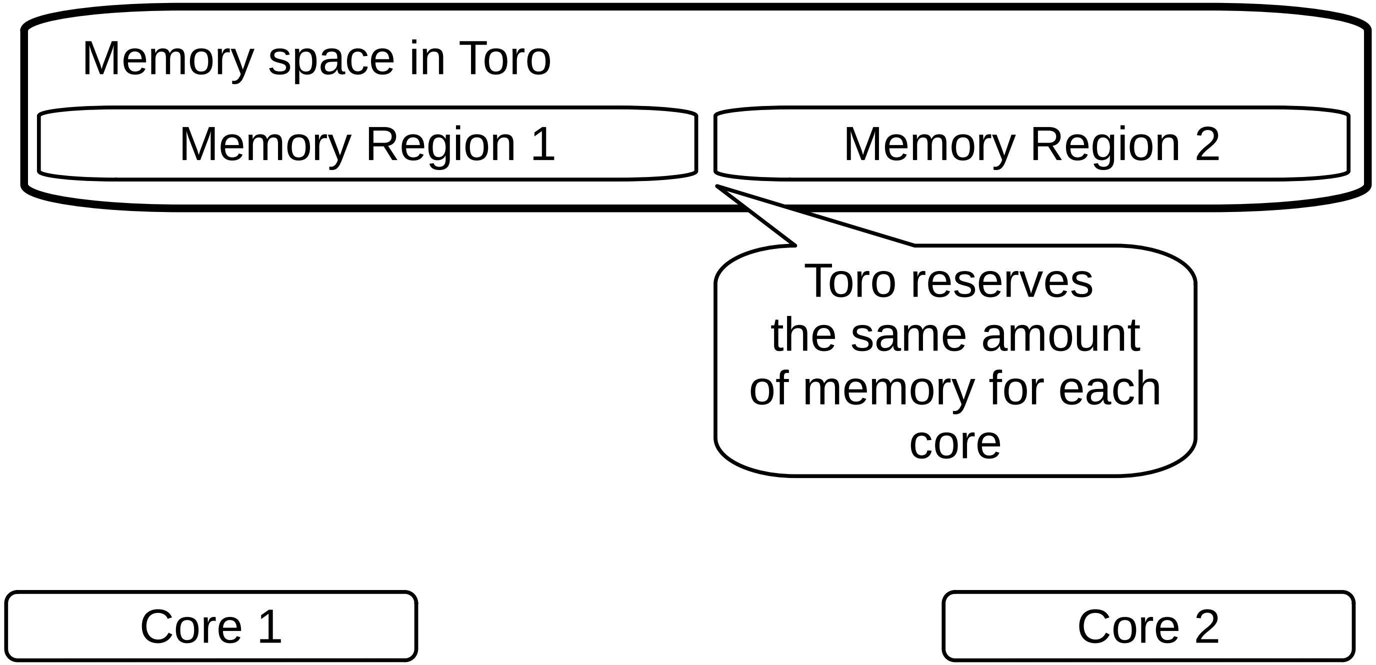
Dedicated Resources

- In a multicore system, the problematic resource is the shared memory. The use of shared memory causes:
  - Overhead in the memory bus
  - Overhead in the cache to keep it coherent
  - Overhead to guaranty mutual exclusion when spin-locks are used

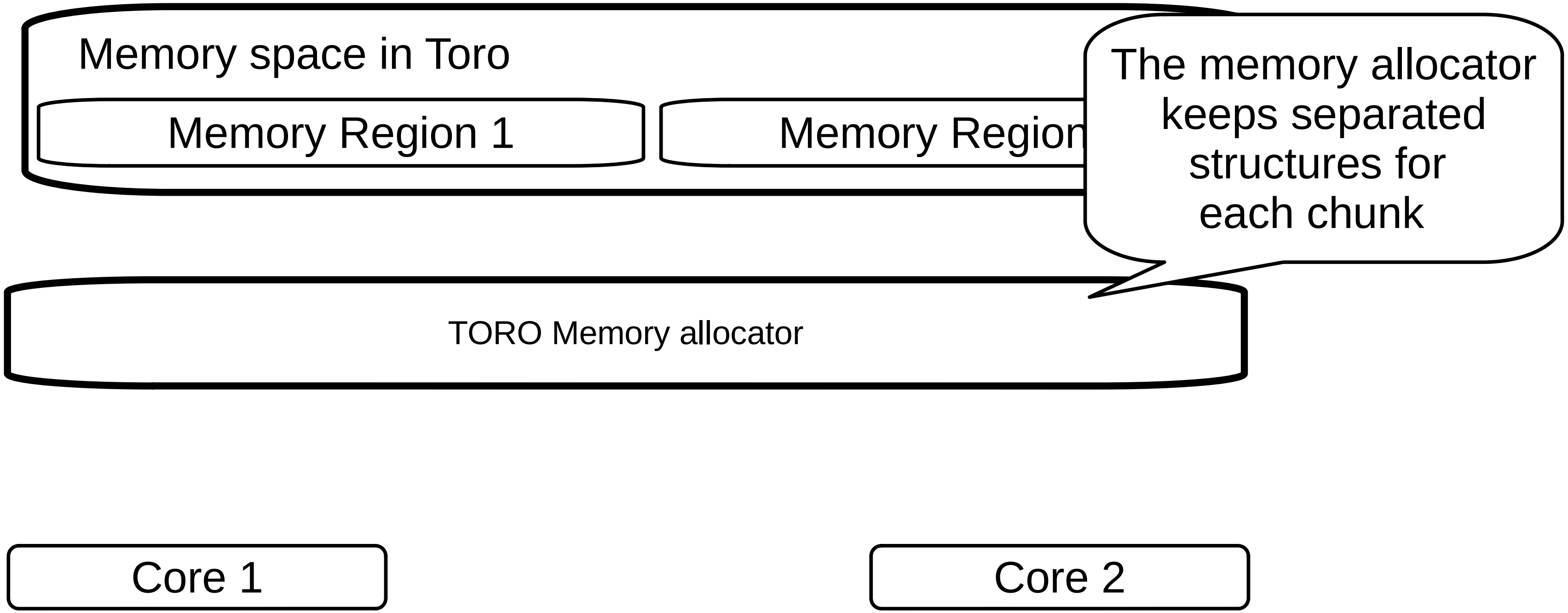
Dedicated Resources Proposal

- Toro improves memory access by
  - keeping the resources locals:
    - The memory is dedicated per core
    - The kernel data structures are dedicated per core
    - The access to kernel data structures is lock free
  - leveraging NUMA technologies, e.g., Hypertransport, Intel QuickPath

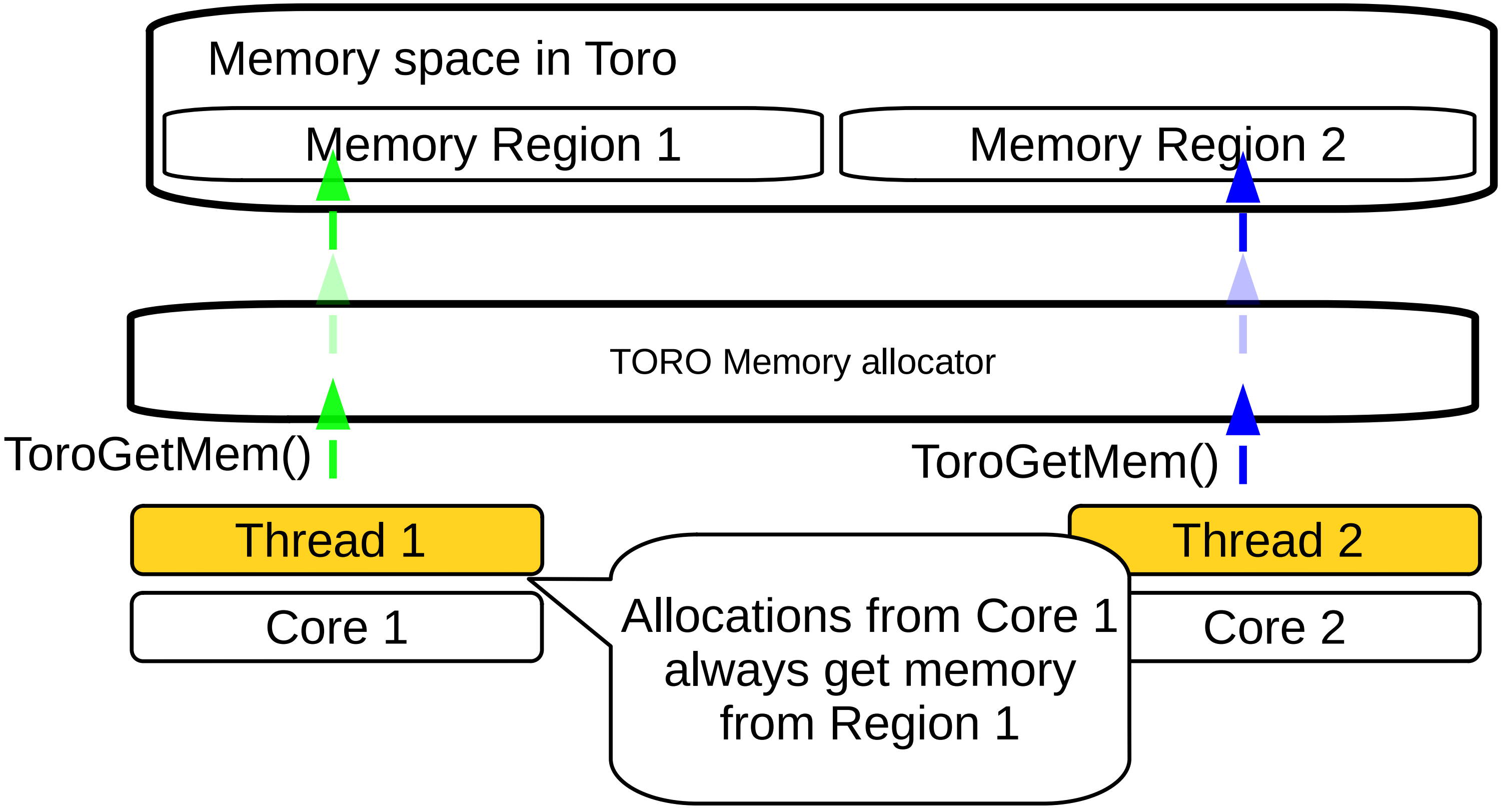
Dedicated Resources Memory



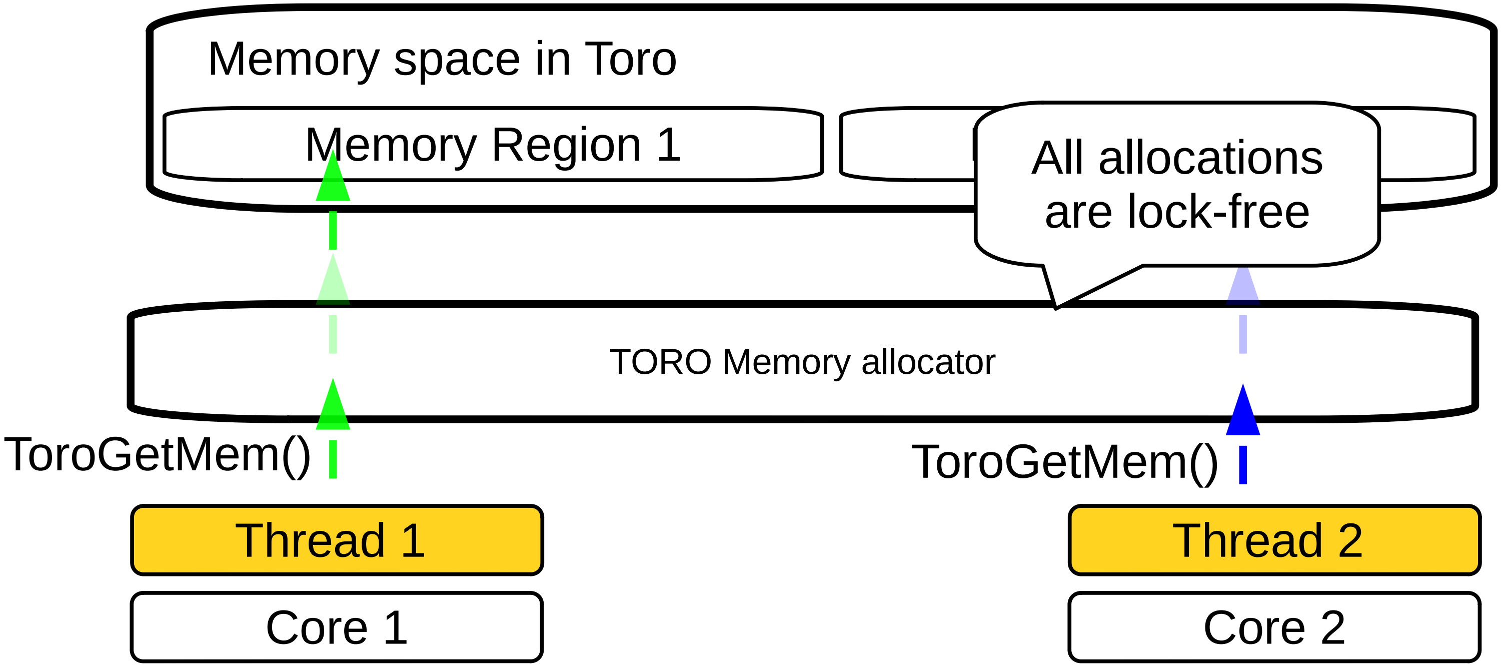
Dedicated Resources
Memory



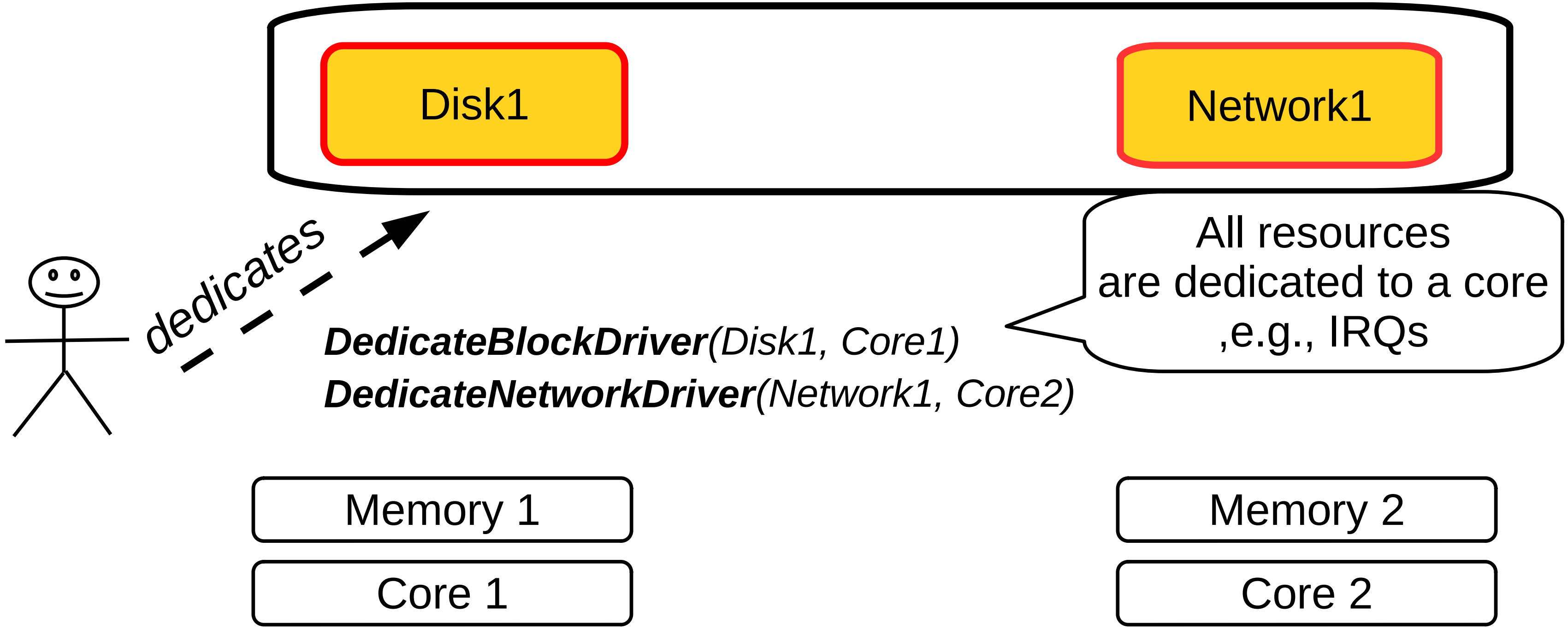
Dedicated Resources Memory



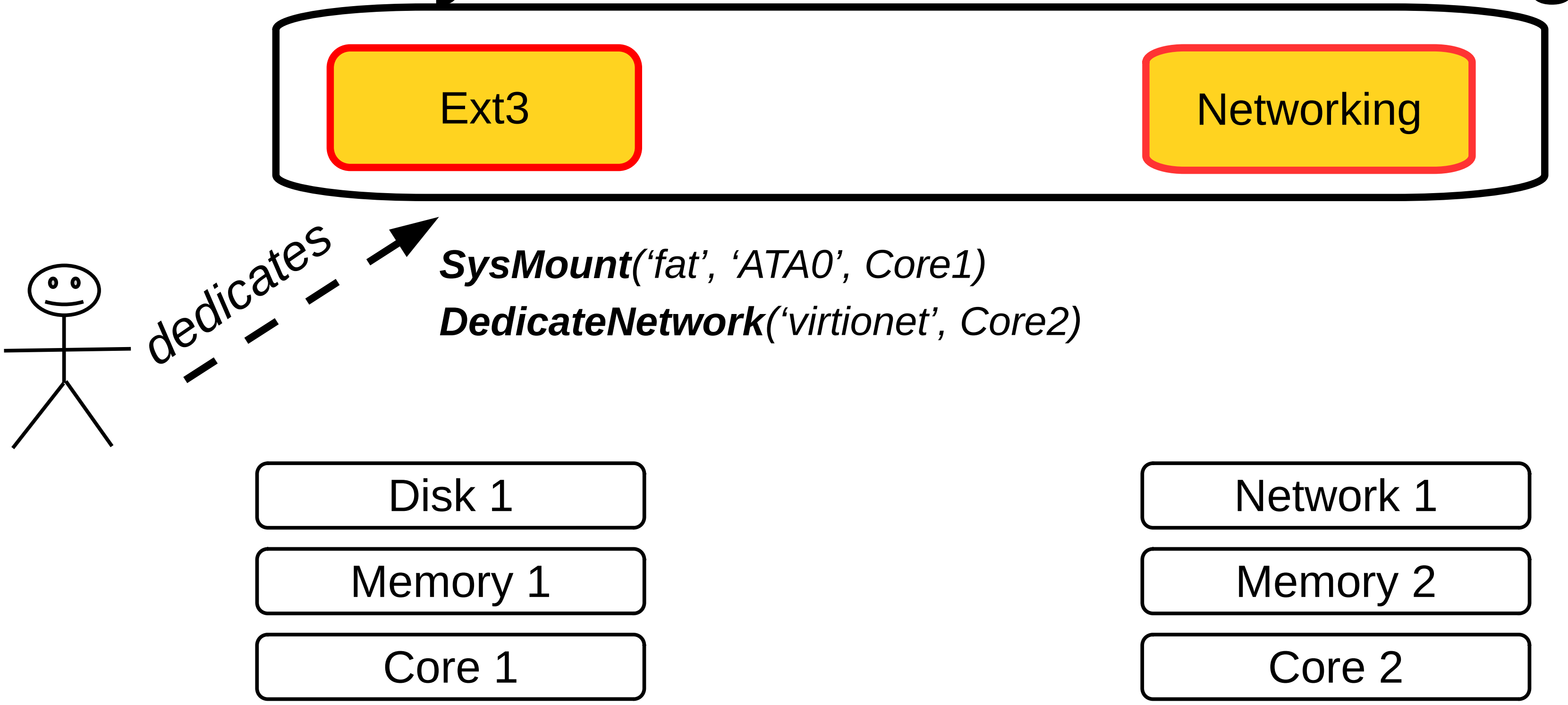
Dedicated Resources
Memory



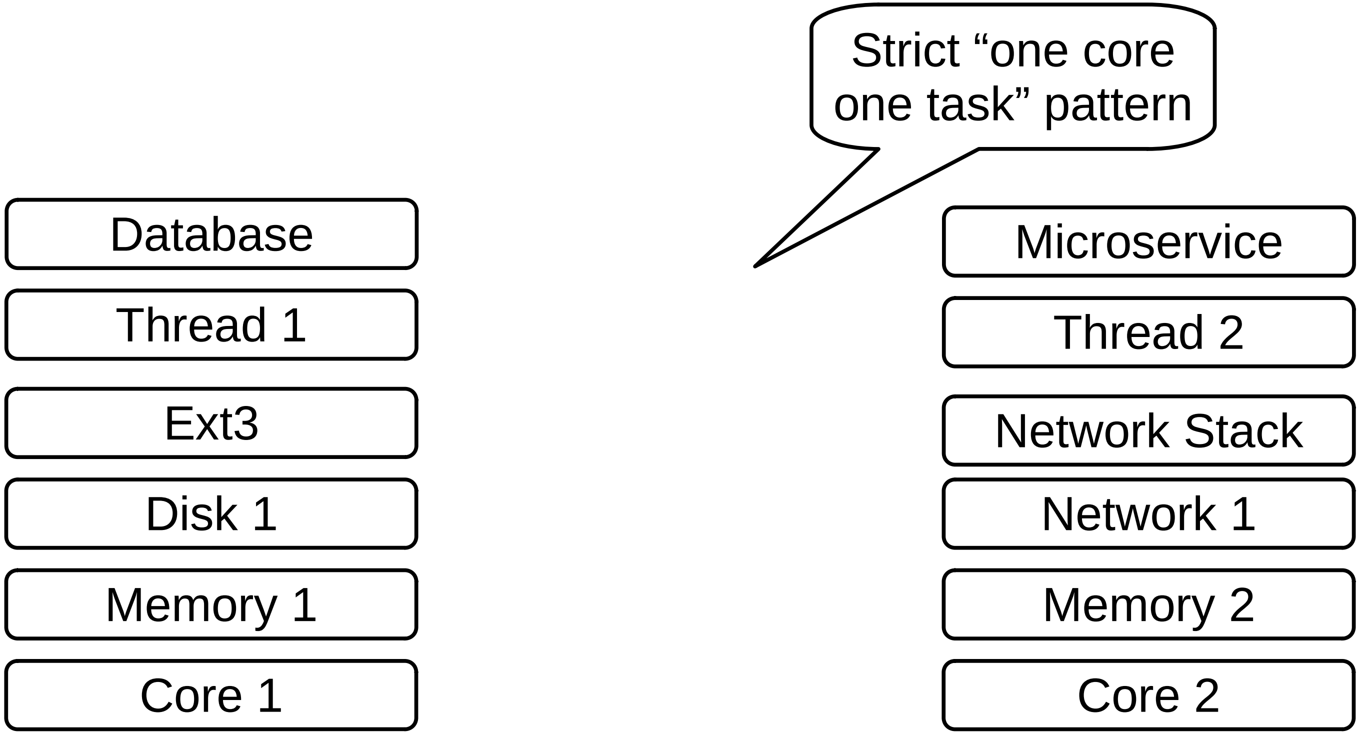
Dedicated Resources
Devices



Dedicated Resources
Filesystems and Networking



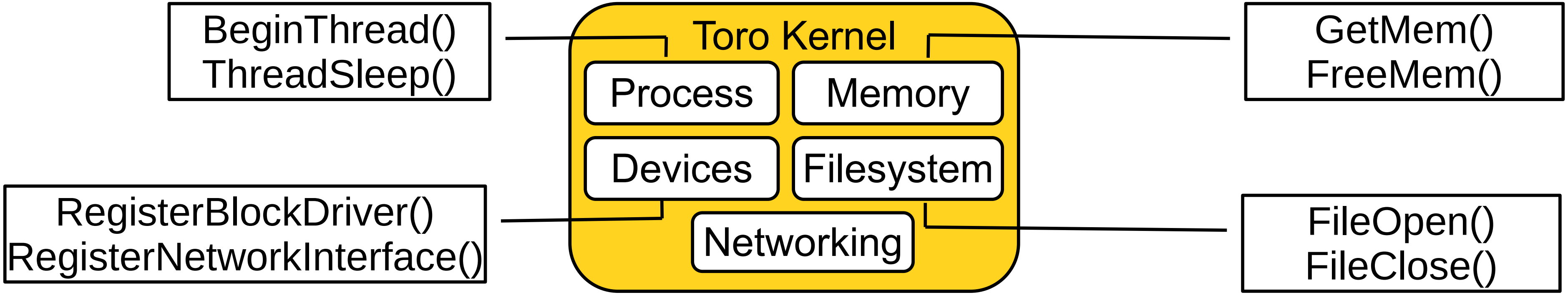
Dedicated Resources Filesystems and Networking



Ingredients for ToroKernel

- A kernel’s architecture that improves the execution of a single user application which is based on:
  - Simple Scheduler
  - Dedicated Resources
- A kernel’s architecture that improves the execution of a single user application as a guest by providing:
  - Fast instantiation of user application
  - Reduced guest’s footprint
  - Improved netwoking and filesystem by relying on VirtIO devices

Application-oriented Kernel

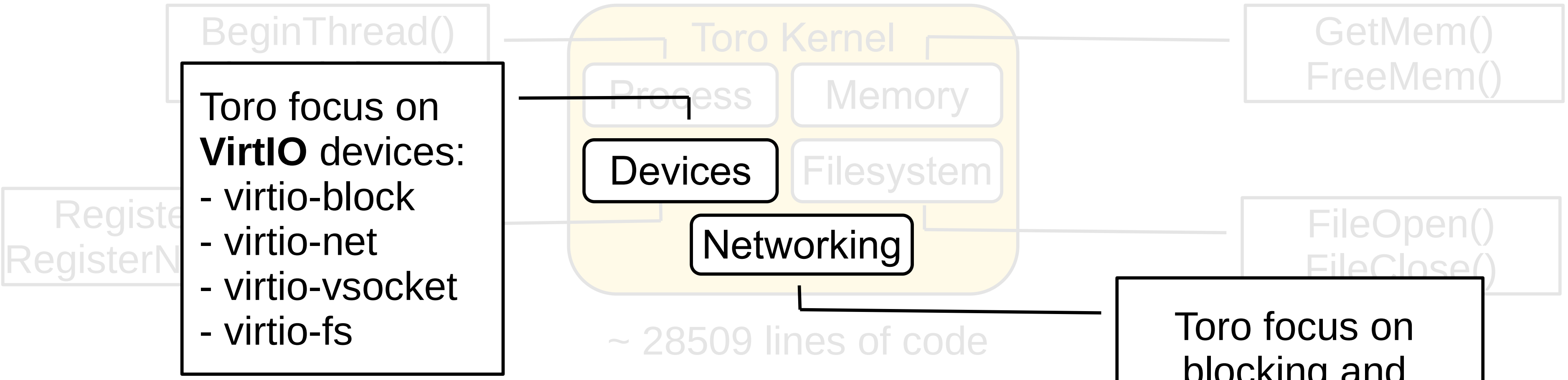


~ 30 KLOC

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

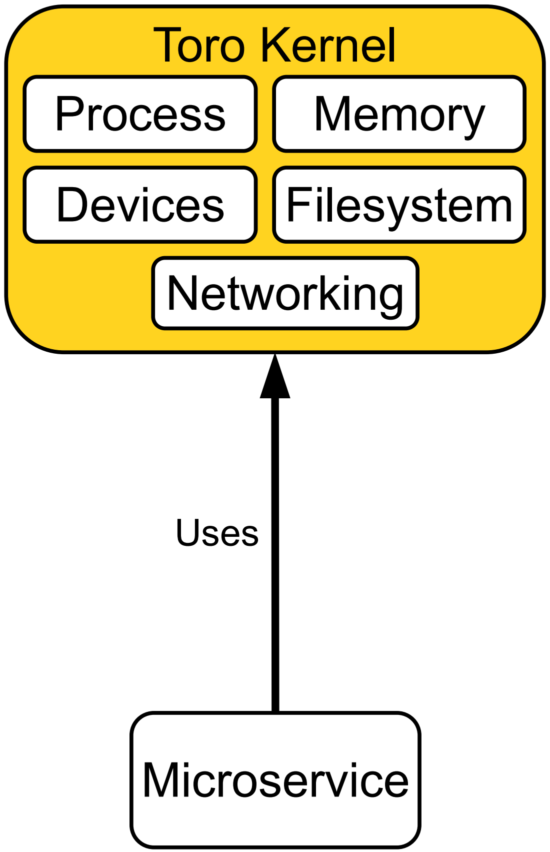
Application-oriented Kernel



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

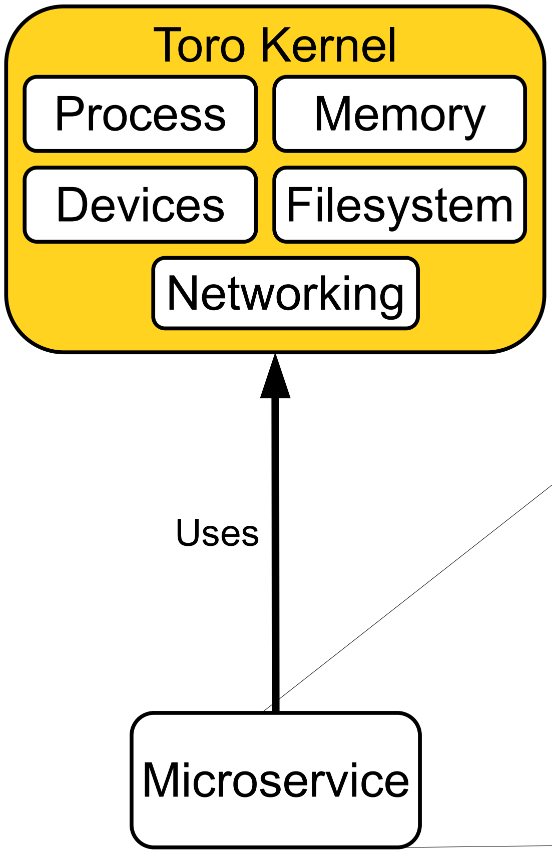
Each unit provides minimalist APIs accessible from the embedded application

Application-oriented Kernel



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

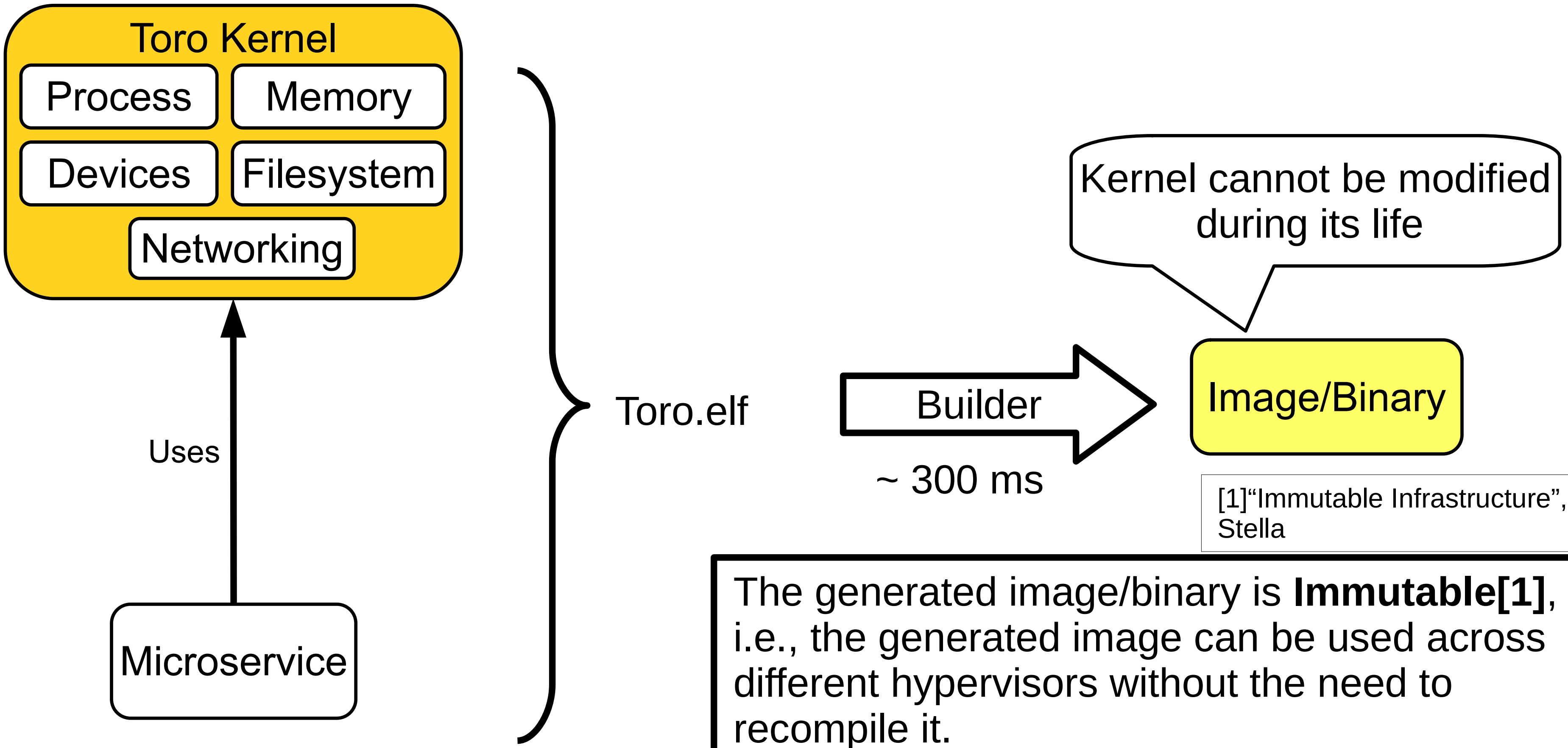
Application-oriented Kernel



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

```
program MyMicroservice;
uses
  Memory,
  Filesystem,
  Process,
  Networking,
  Ext3,
  E1000;
Begin
//
// Your Code
//
End.
```

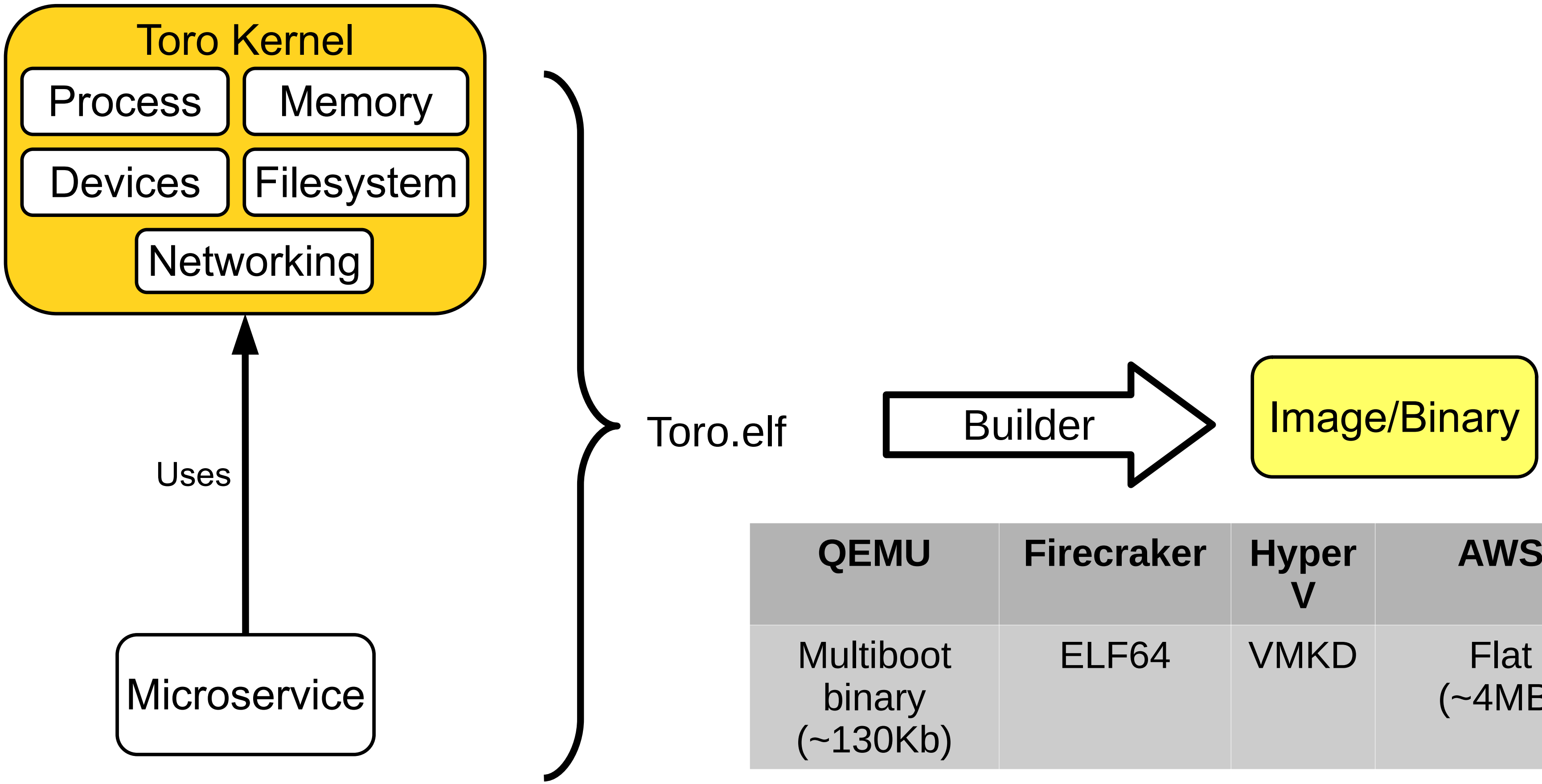
Application-oriented Kernel



[1]“Immutable Infrastructure”, Stella

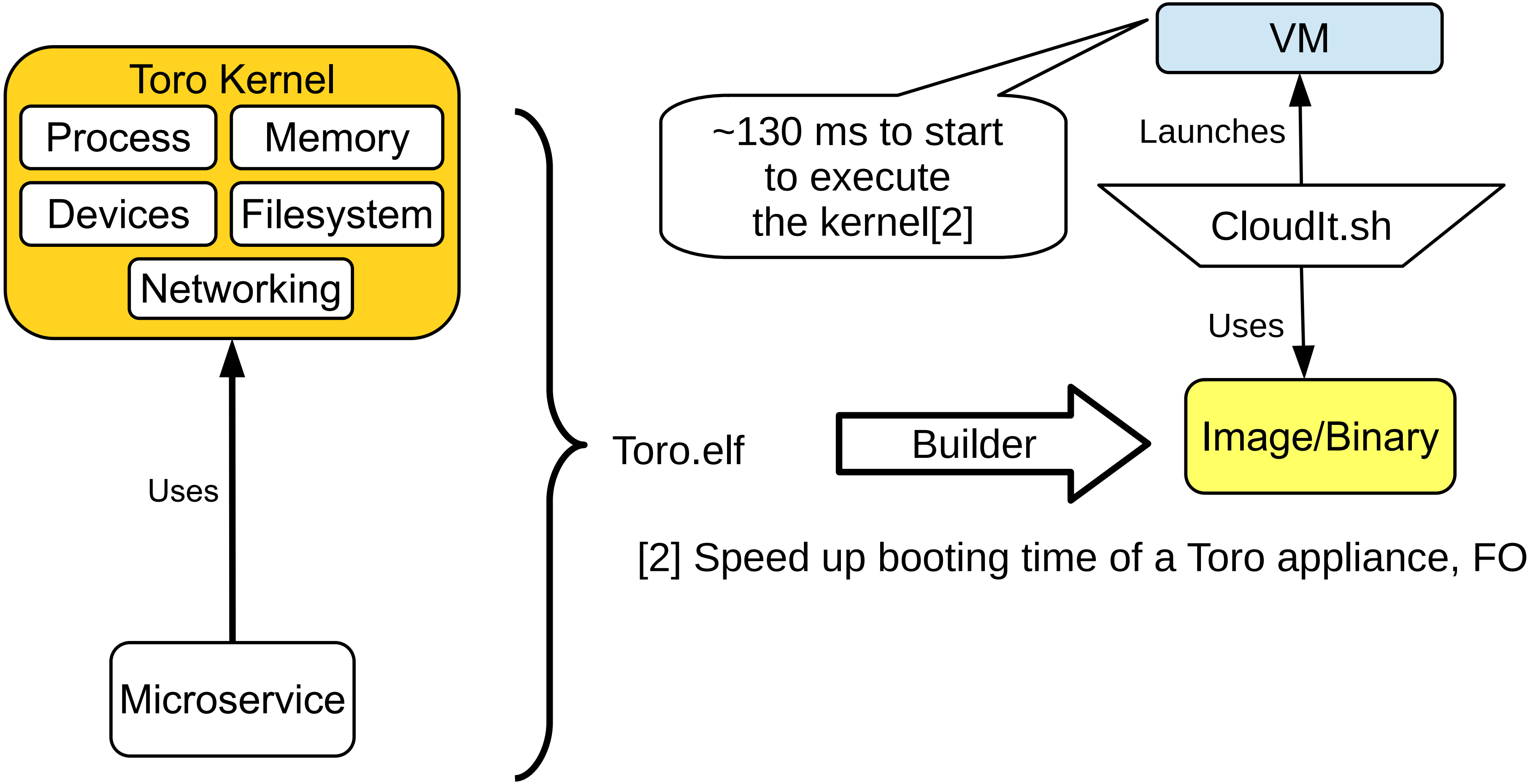
The generated image/binary is **Immutable[1]**, i.e., the generated image can be used across different hypervisors without the need to recompile it.

Application-oriented Kernel



QEMU	Firecraker	Hyper V	AWS	GCE
Multiboot binary (~130Kb)	ELF64	VMKD	Flat (~4MB)	tar.gz

Application-oriented Kernel



[2] Speed up booting time of a Toro appliance, FOSDEM’19

Time until kernel starts to execute

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

Approach	Image (~ 4MB)	Binary (~ 130kB)	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/

Benefits of Application-oriented Kernel

- Security is based on the hypervisor which ensures a reduced attack surface
- The kernel size is smaller since it only includes the units required for the application
- The smaller kernel size reduces the booting time and eases image manipulation, i.e., your program is your kernel!
- No communication overhead since application is using the kernel APIs

Benefits of Application-oriented Kernel

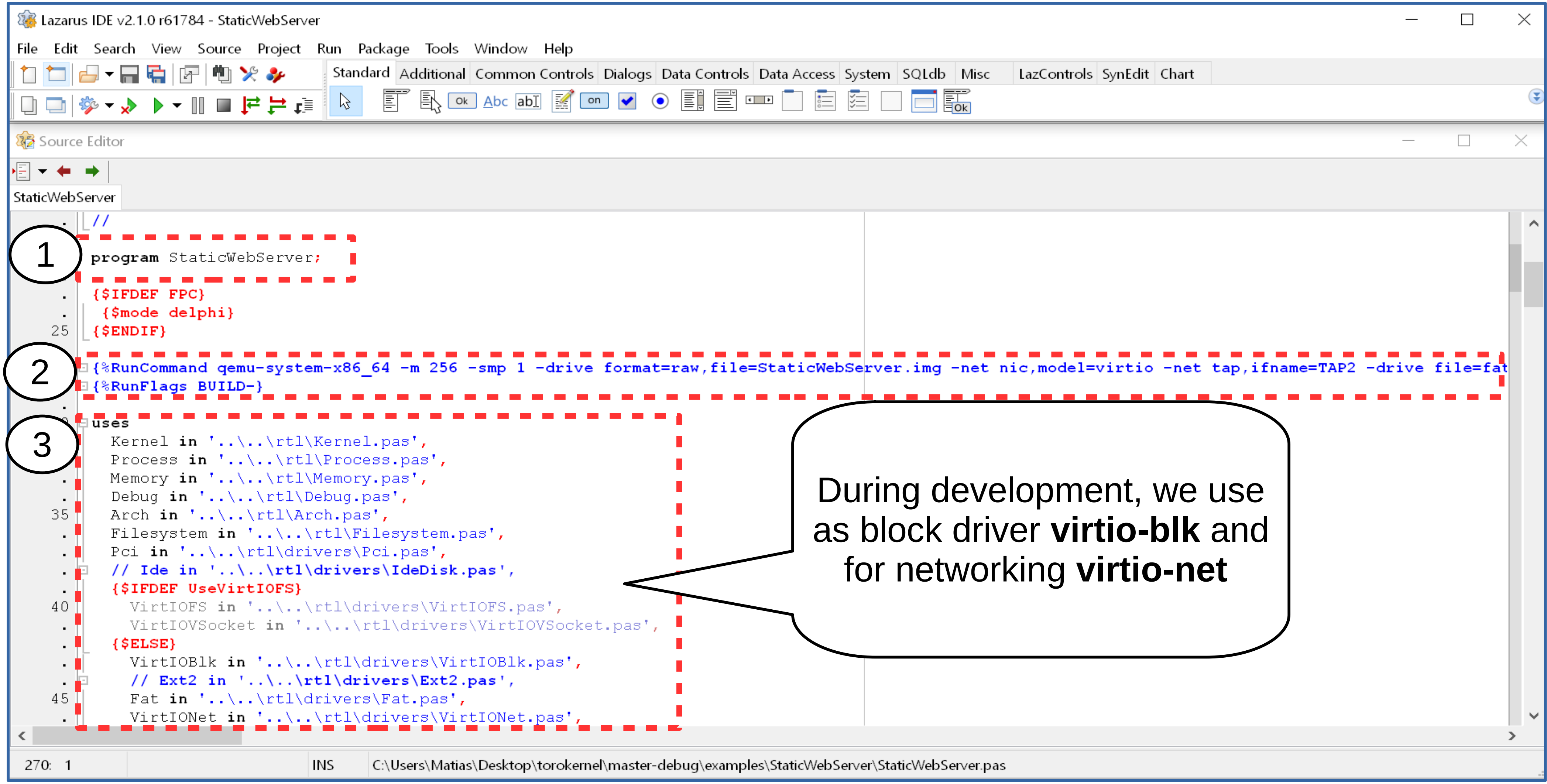
- Security is based on the hypervisor which ensures a reduced attack surface
- Th re
- Th es image manipulation, i.e., your kernel is your program!
- No communication overhead since application is using the kernel APIs

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

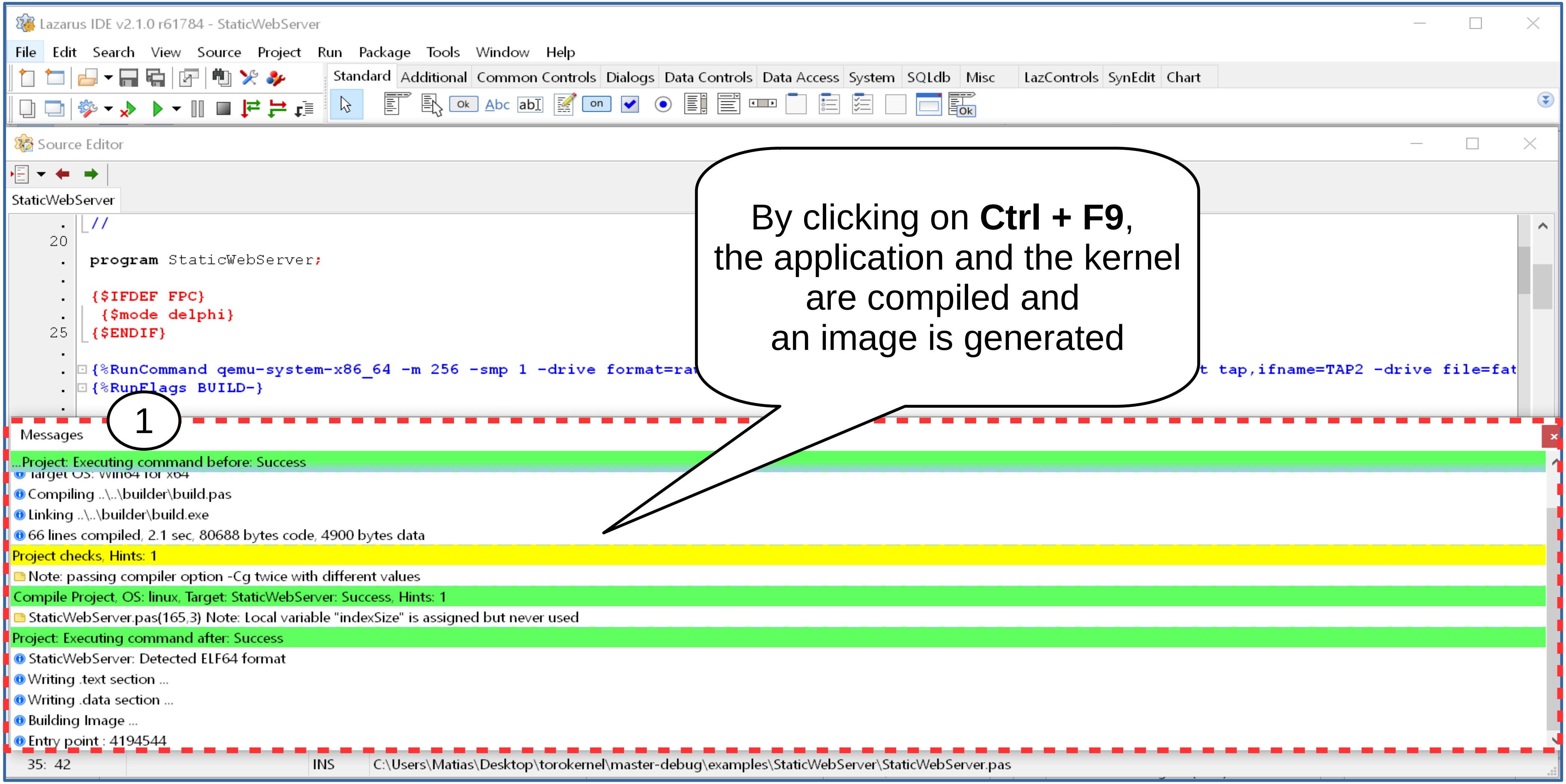
Example: The Static WebServer

- Simple microservice that serves files by using the HTTP protocol
  - https://github.com/torokernel/torokernel/tree/master/examples/StaticWebServer (among other examples ;))
- For development, we base on Windows
- For deployment, we base on a baremetal Linux Host (KVM guest) on Scaleway

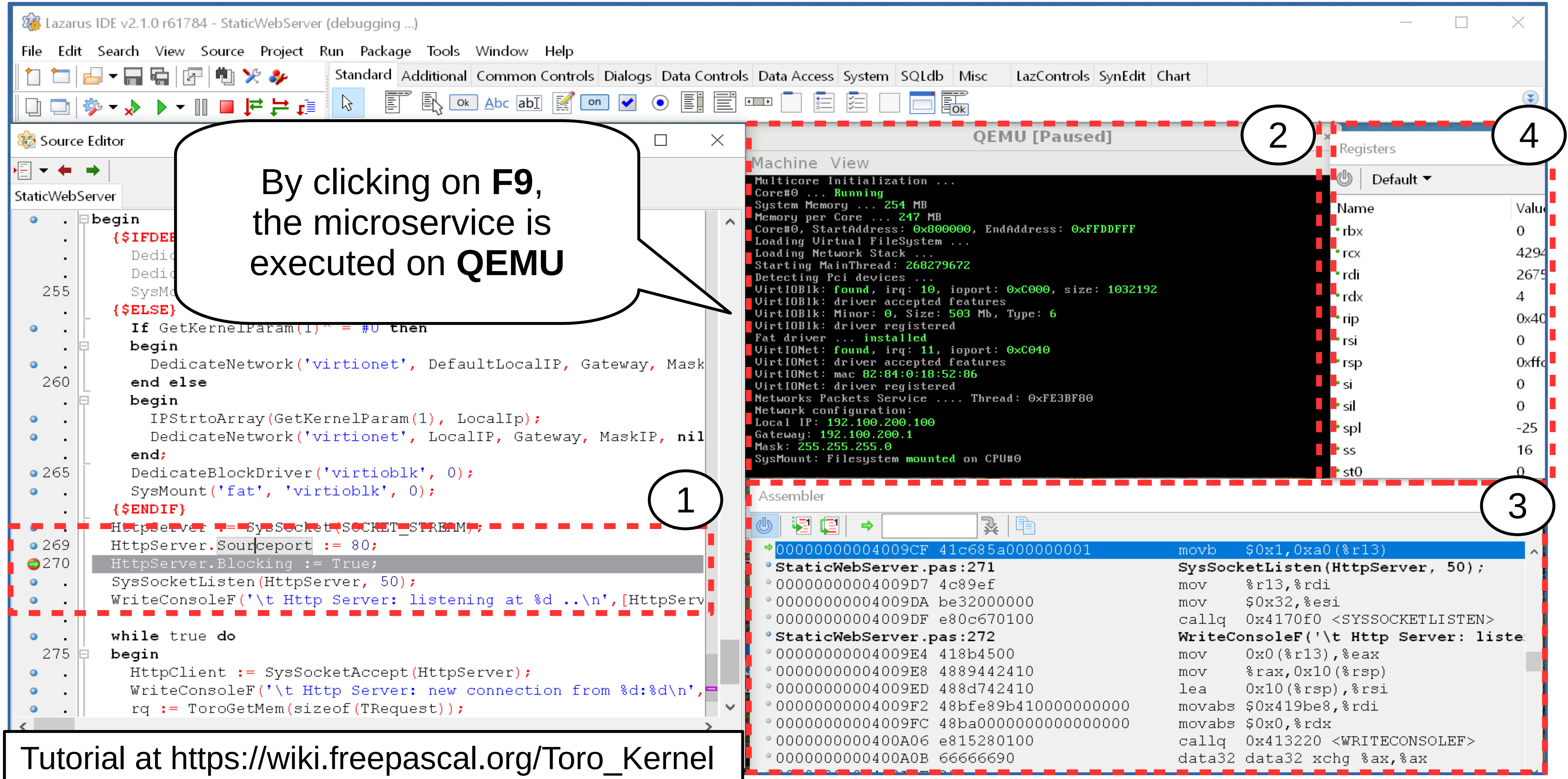
Development Workbench



Development Workbench



Debugging the kernel from the IDE



Tutorial at https://wiki.freepascal.org/Toro_Kernel

Deployment Workbench

KVM Guest (**Scaleway** Baremetal Host)

$ CloudIt.sh **StaticWebServer** “-append **virtiovsocket,virtiofs,myfstoro**”

```
Booting from ROM..Loading Toro ... HEAD:a990437
Multicore Initialization ...
Core#0 ... Running
System Memory ... 4094 MB
Memory per Core ... 3063 MB
Core#0, StartAddress: 0x800000, EndAddress: 0xBFFDFFFF
Loading Virtual FileSystem ...
Loading Network Stack ...
Starting MainThread: 3221077912
Detecting Pci devices ...
VirtIOFS: Detected device tagged: myfstoro, queues: 1
VirtIOFS: Device accepted features
VirtIOFS: Queue 0, size: 1024, initiated, irq: 11
Fat driver ... installed
VirtIOVSocket: found, irq: 10, ioport: 0xC040
VirtIOVSocket: Guest ID: 3
VirtIOVSocket: RX_QUEUE was initiated
VirtIOVSocket: EVENT_QUEUE was initiated
VirtIOVSocket: TX_QUEUE was initiated
VirtIOVSocket: driver registered
Networks Packets Service .... Thread: 0xBE83BFA8
DedicateNetworkSocket: success on core #0
SysMount: Filesystem mounted on CPU#0
26/10/2019-21:07:33 Http Server: listening at 80 ..
```

Check on http://www.torokernel.io

Summary

- Toro design is improved in five main points:
  - Booting time and building time
  - Kernel API with zero overhead function call
  - Access to shared memory
  - Networking
  - CPU usage

Future Work

- Improve tooling to build, test and debug microservices
- Investigate new use-cases
- Port networking applications to provide starter kits: SMTP relay, HTTP proxy, Web tracking
- Investigate the use of microVM technologies, e.g., Firecraker, NEMU.
- Investigate the use of OpenStack to ease the deployment of Toro appliances
- Compare with other approaches, e.g., unikernels, containers [3]

[3] Toro, a dedicated kernel for microservices, OSSummit'18

QA