THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

From Handcraft to Unikraft:
Simpler Unikernelization of Your Application

Florian Schmidt

Research Scientist, NEC Labs Europe

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements no. 675806 ("5G CITY") and 761592 ("5G ESSENCE"). This work reflects only the author's views and the European Commission is not responsible for any use that may be made of the information it contains.

THE LINUX FOUNDATION

VMs vs Containers

VMs have been around for a long time

- They allow consolidation, isolation, migration, …

VMs vs Containers

- VMs have been around for a long time
  - They allow consolidation, isolation, migration, …
- Then containers came and many people LOVED them. Why?

VMs vs Containers

VMs have been around for a long time

- They allow consolidation, isolation, migration, …

Then containers came and many people LOVED them. Why?

- “Containers are much faster to bring up than VMs.
  My VM takes a minute to boot, my container only a second.”

VMs vs Containers

VMs have been around for a long time

- They allow consolidation, isolation, migration, …

Then containers came and many people LOVED them. Why?

- “Containers are much faster to bring up than VMs.
  My VM takes a minute to boot, my container only a second.”
- “Containers are much smaller.
  My VM takes 10 GB, my container only a few hundred MB.”

VMs vs Containers

VMs have been around for a long time

- They allow consolidation, isolation, migration, …

Then containers came and many people LOVED them. Why?

- "Containers are much faster to bring up than VMs.
  My VM takes a minute to boot, my container only a second."
- "Containers are much smaller.
  My VM takes 10 GB, my container only a few hundred MB."
- "Containers are much easier to create and deploy.
  I just write this Dockerfile and I'm done."

Containers vs Unikernels

I don’t want to bash containers.

- Containers can be great!
- For example, I love them for build environments

Containers vs Unikernels

- I don’t want to bash containers.
  - Containers can be great!
  - For example, I love them for build environments
- But VMs have their advantages
  - Most importantly, strong isolation

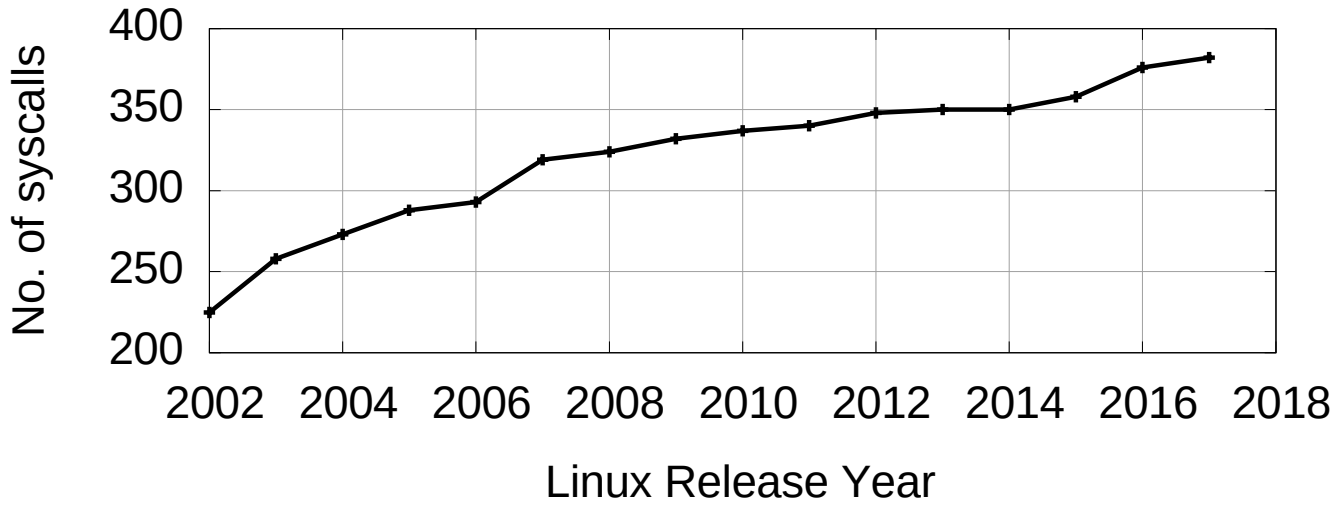
Containers vs Unikernels

I don’t want to bash containers.

- Containers can be great!
- For example, I love them for build environments

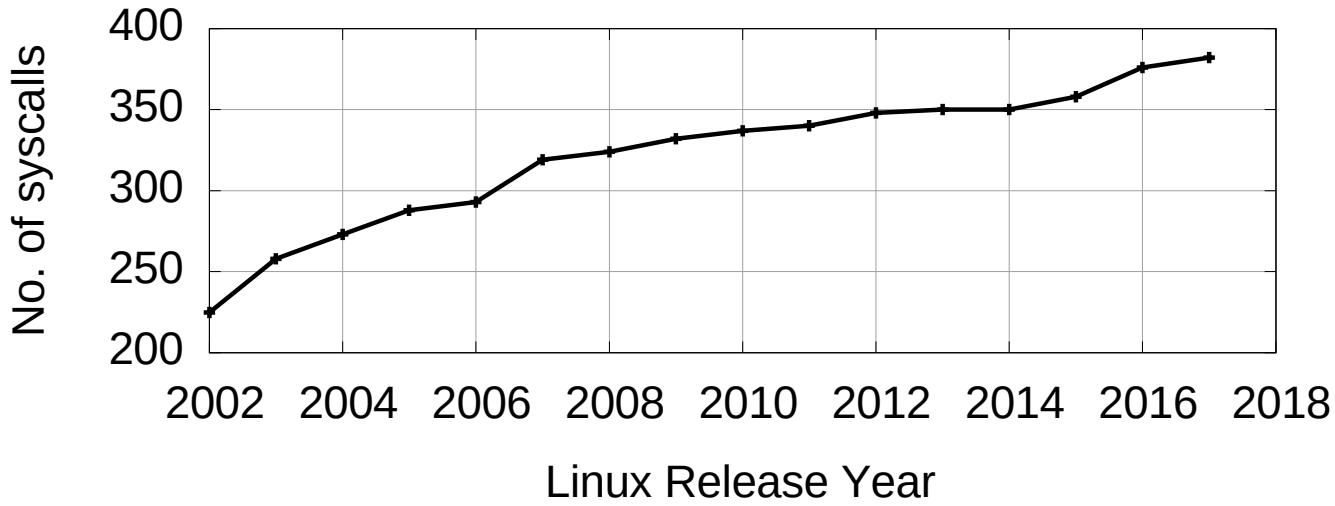
But VMs have their advantages

- Most importantly, strong isolation



Containers vs Unikernels

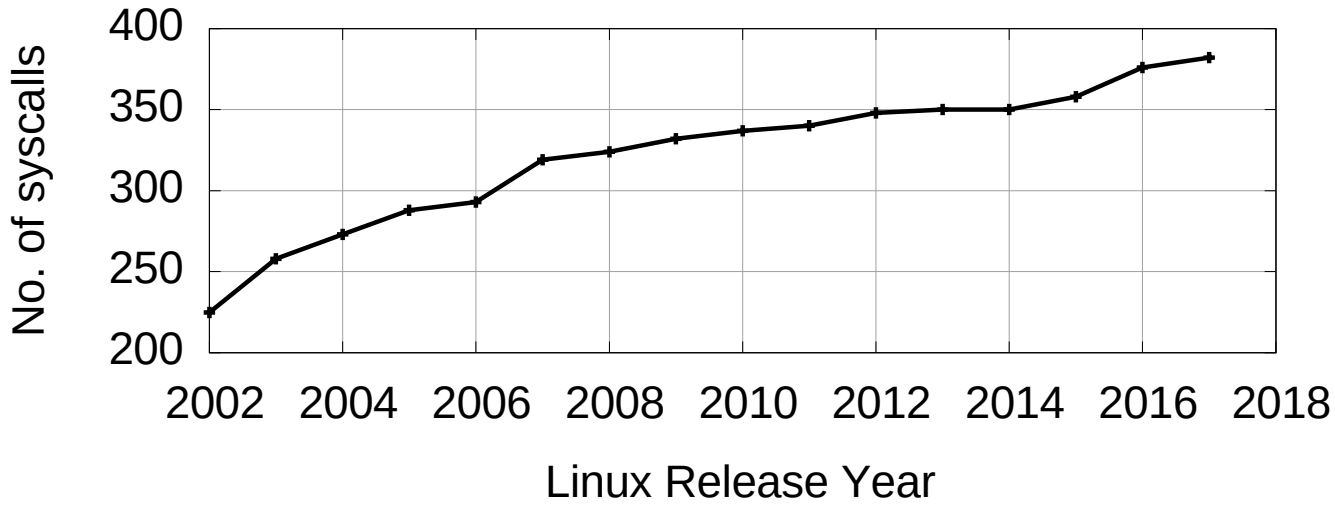
- I don’t want to bash containers.
  - Containers can be great!
  - For example, I love them for build environments
- But VMs have their advantages
  - Most importantly, strong isolation



- And they do not *have* to be large, slow, and complicated

Containers vs Unikernels

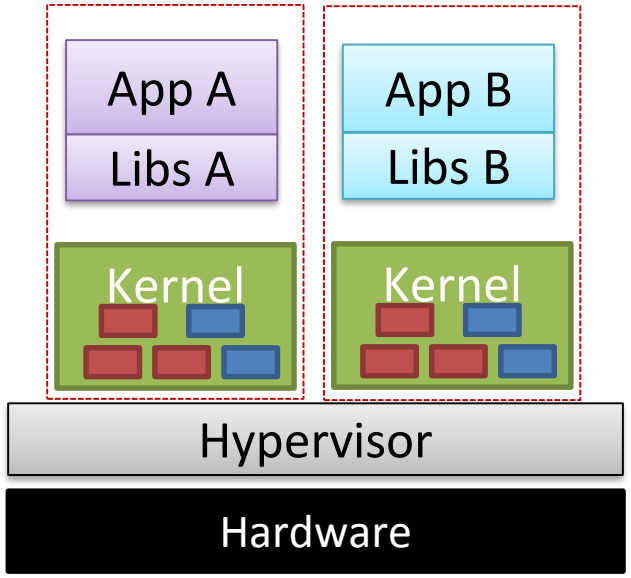
- I don’t want to bash containers.
  - Containers can be great!
  - For example, I love them for build environments
- But VMs have their advantages
  - Most importantly, strong isolation



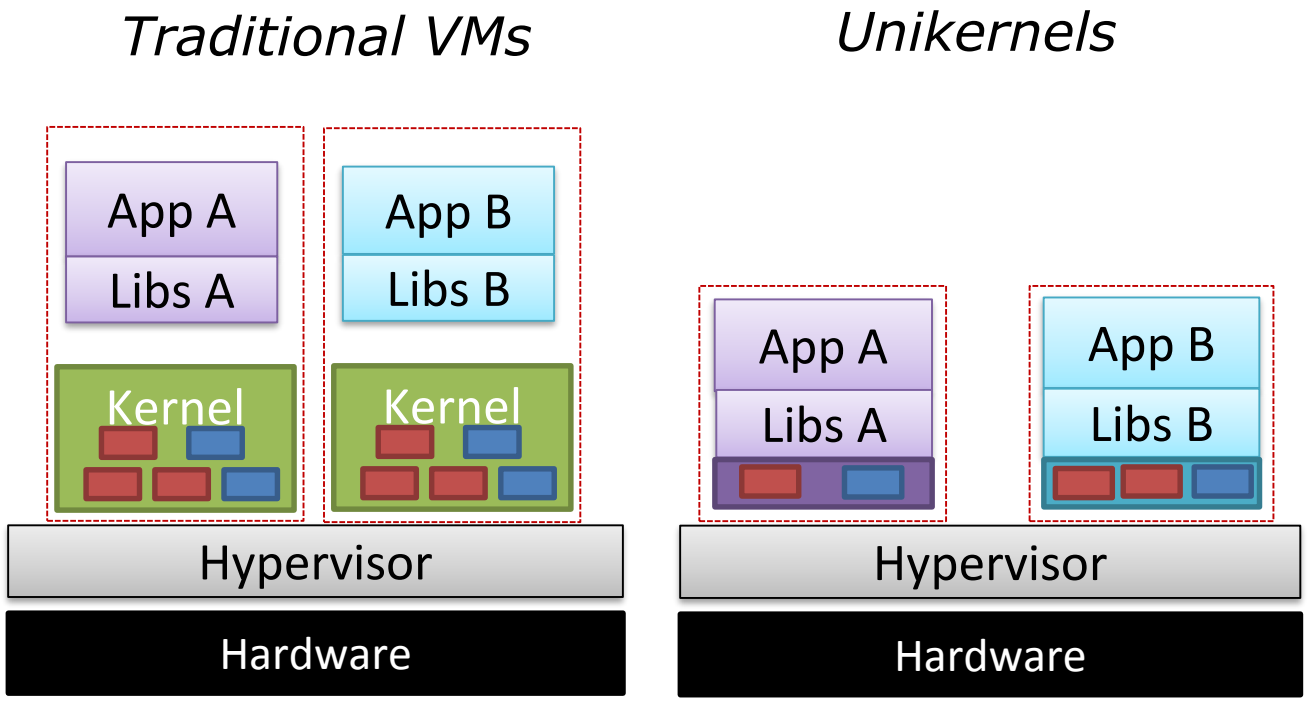
- And they do not *have* to be large, slow, and complicated
  - This is where unikernels come in

Traditional VMs vs. Unikernels

Traditional VMs



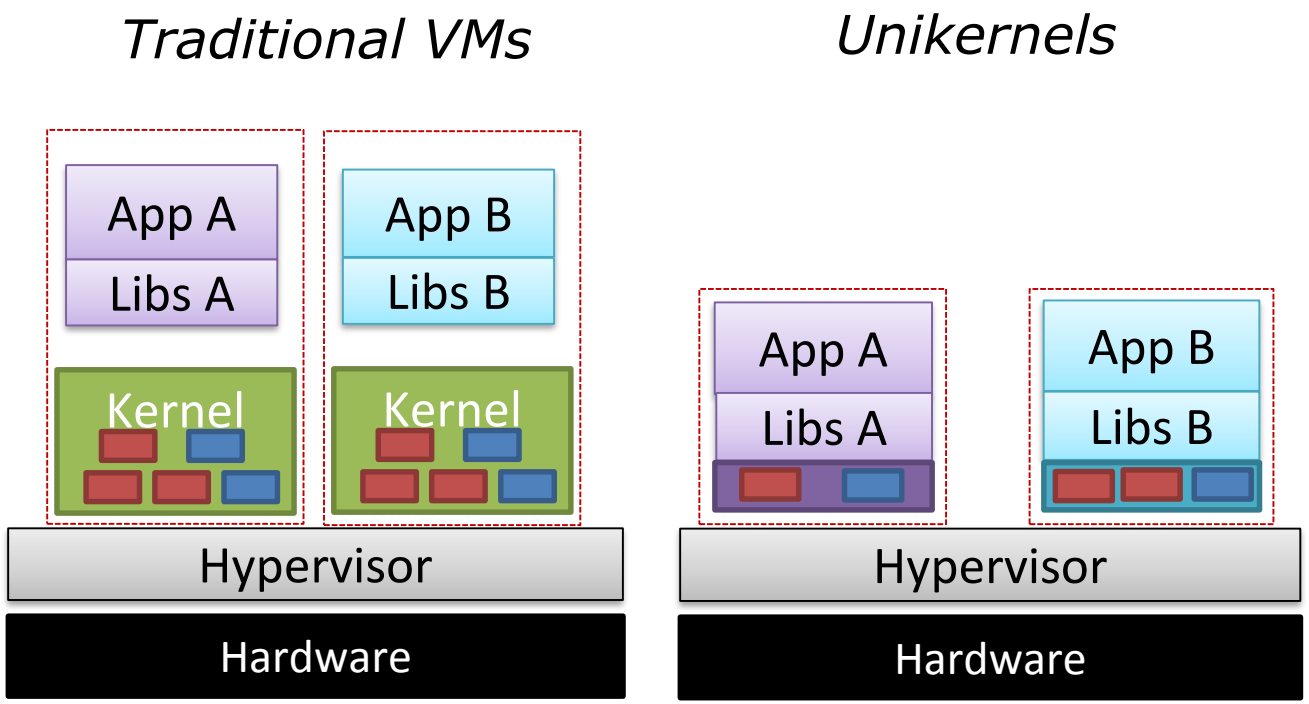
Traditional VMs vs. Unikernels



Unikernels are purpose-built

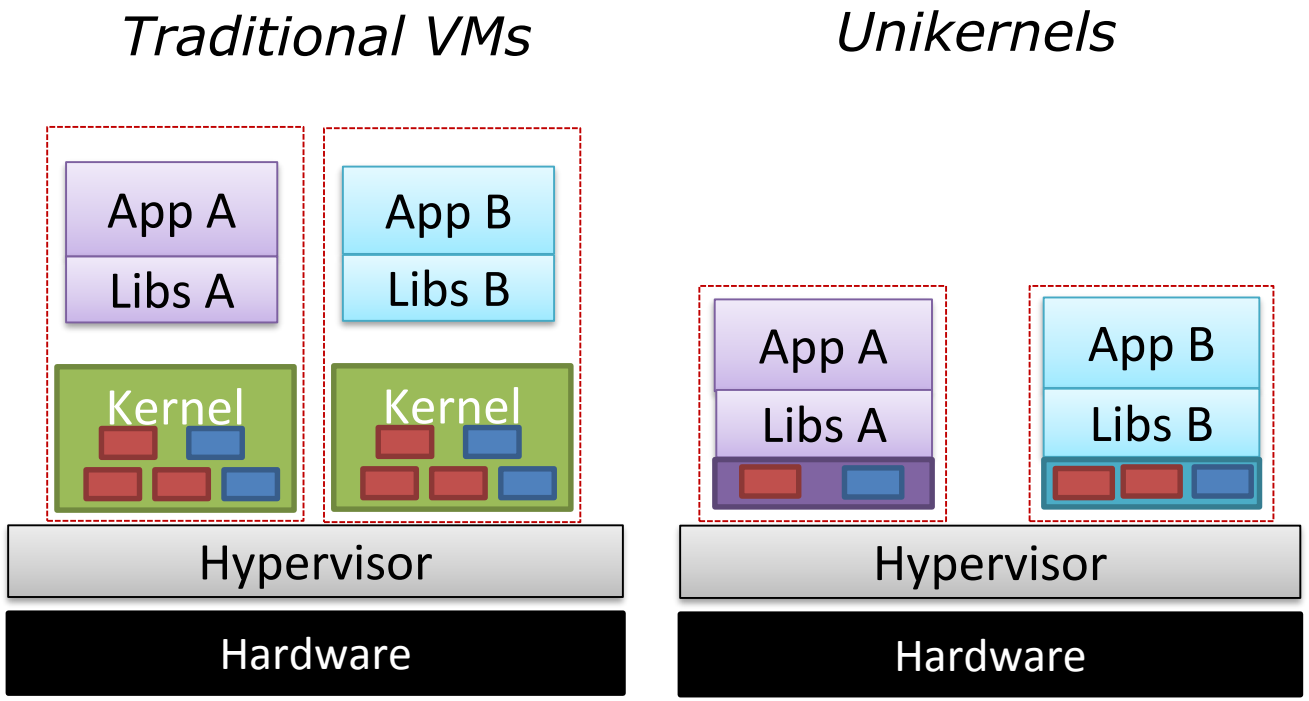
- A single binary containing OS and (single) application
- One application → Flat and single address space

Traditional VMs vs. Unikernels



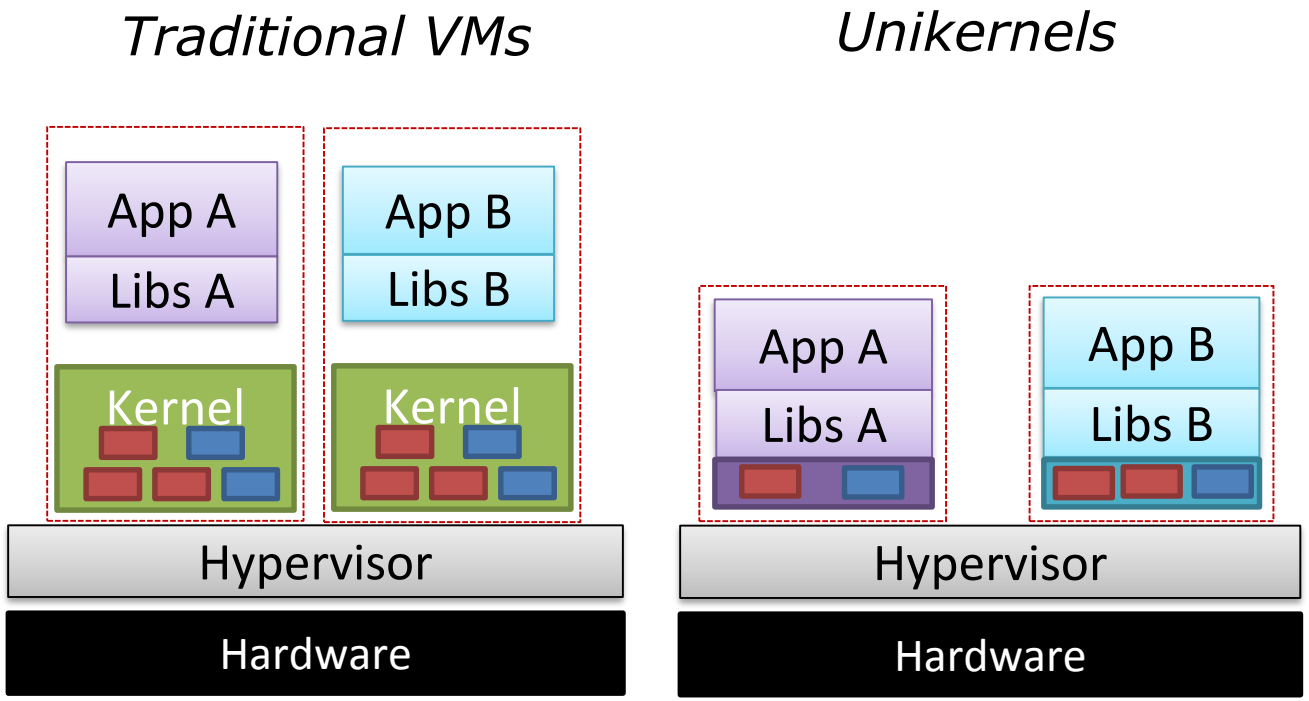
- Unikernels are purpose-built
  - A single binary containing OS and (single) application
  - One application → Flat and single address space
- No isolation within unikernel, but by the hypervisor

Traditional VMs vs. Unikernels



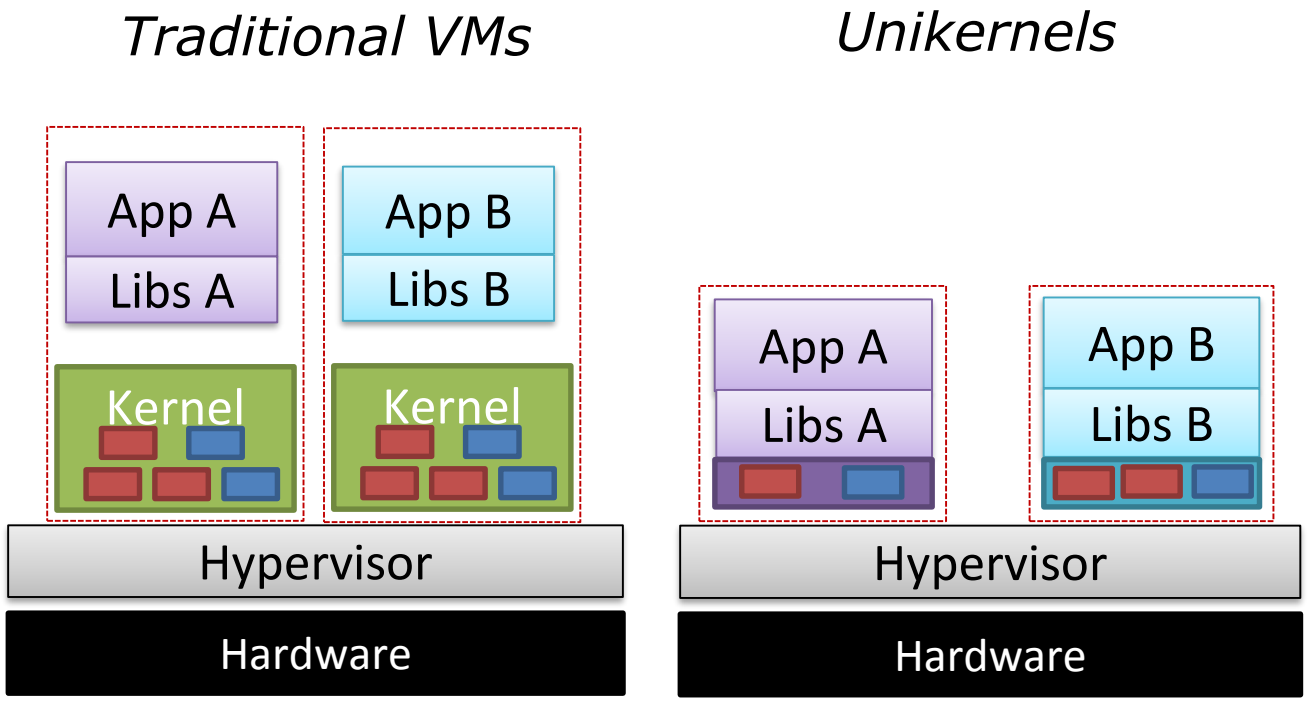
- Unikernels are purpose-built
  - A single binary containing OS and (single) application
  - One application → Flat and single address space
- No isolation within unikernel, but by the hypervisor
- Can be extremely small and blazingly fast
- Example: unikernel web server
  - 5-6x more req/s than standard nginx

Traditional VMs vs. Unikernels



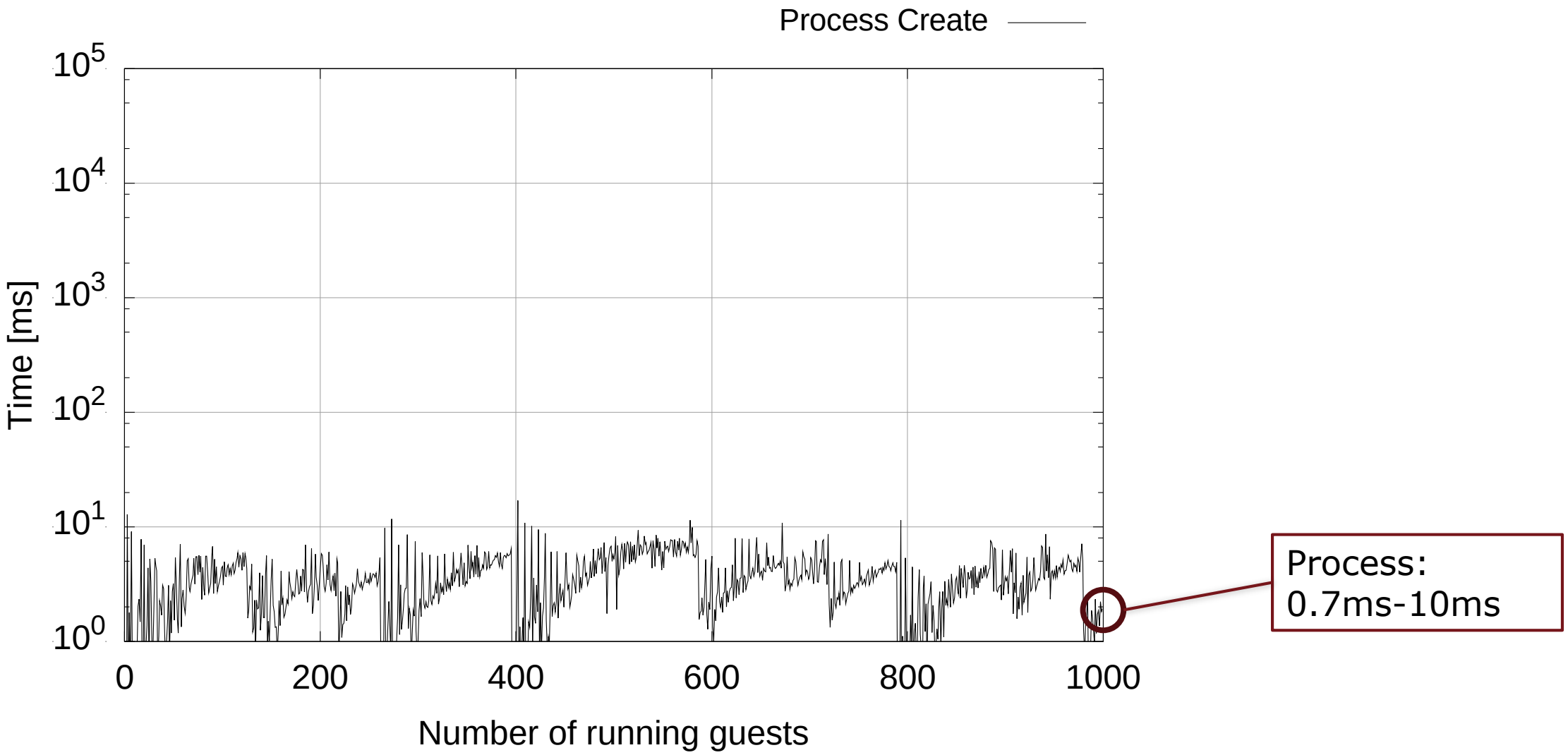
- Unikernels are purpose-built
  - A single binary containing OS and (single) application
  - One application → Flat and single address space
- No isolation within unikernel, but by the hypervisor
- Can be extremely small and blazingly fast
- Example: unikernel web server
  - 5-6x more req/s than standard nginx
  - Nearly saturates 40Gb/s link

Traditional VMs vs. Unikernels



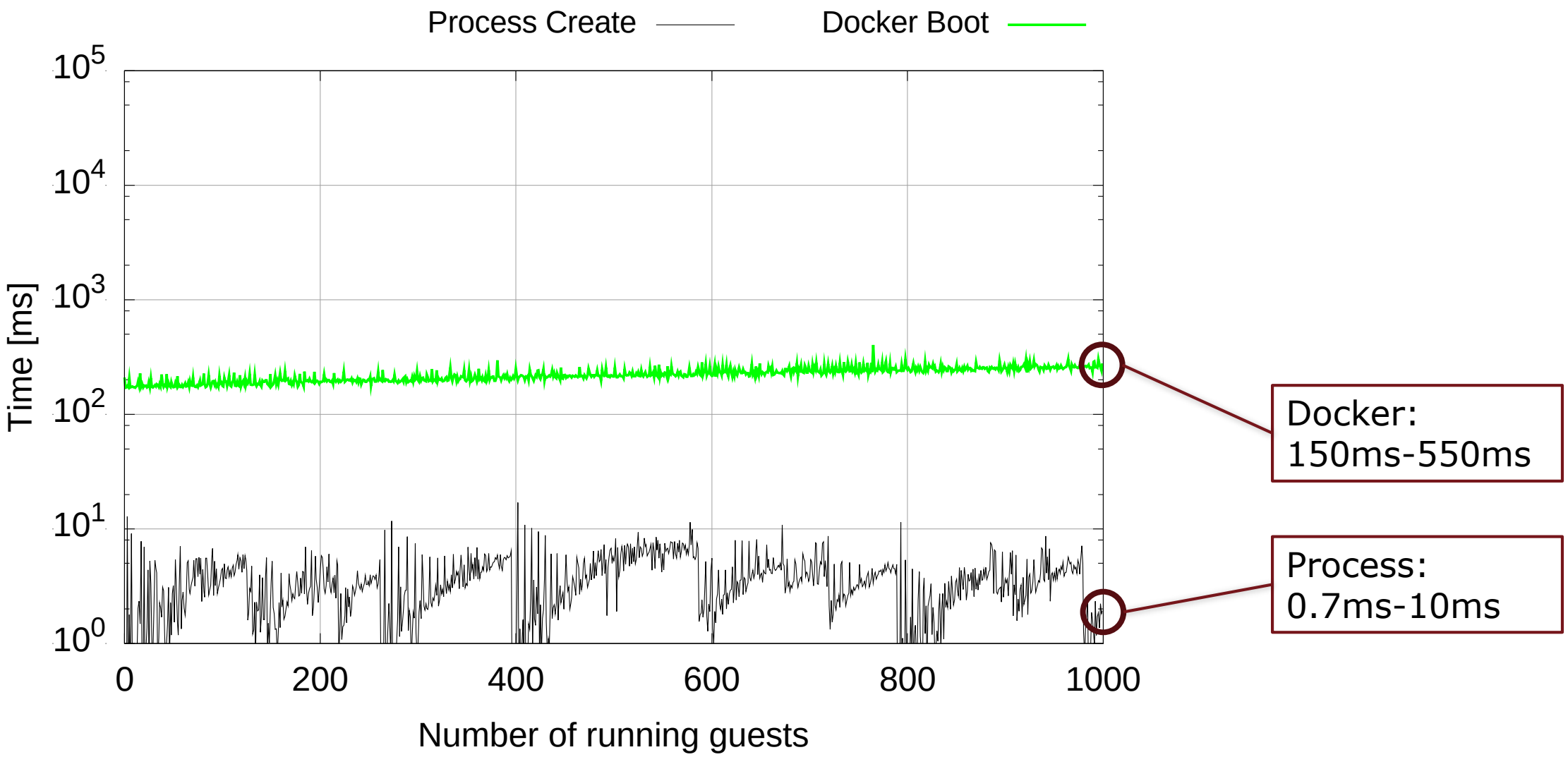
- Unikernels are purpose-built
  - A single binary containing OS and (single) application
  - One application → Flat and single address space
- No isolation within unikernel, but by the hypervisor
- Can be extremely small and blazingly fast
- Example: unikernel web server
  - 5-6x more req/s than standard nginx
  - Nearly saturates 40Gb/s link
  - Image: 670kB, RAM: <32MB

Example: Instantiation Time Comparison



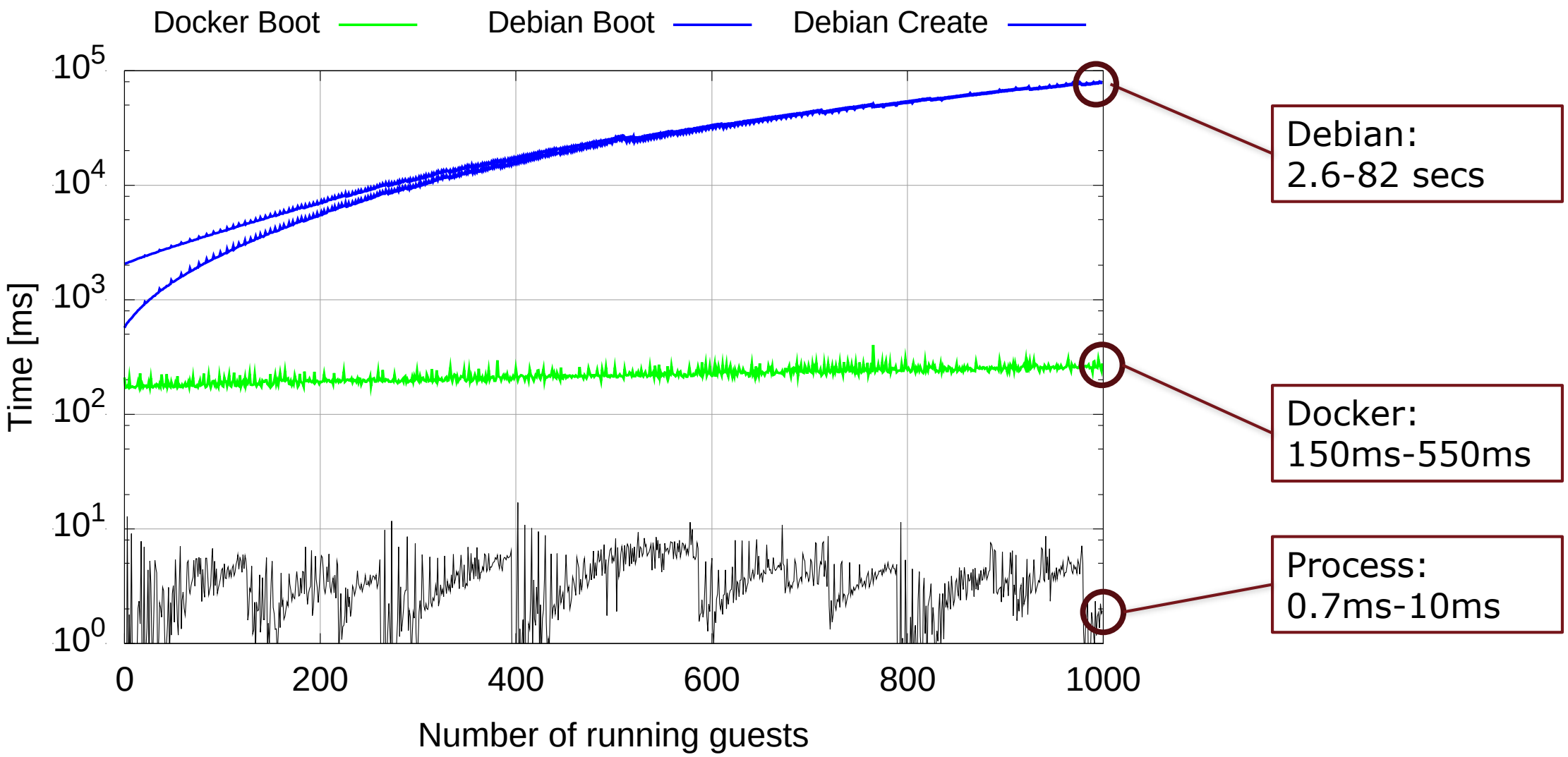
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Example: Instantiation Time Comparison



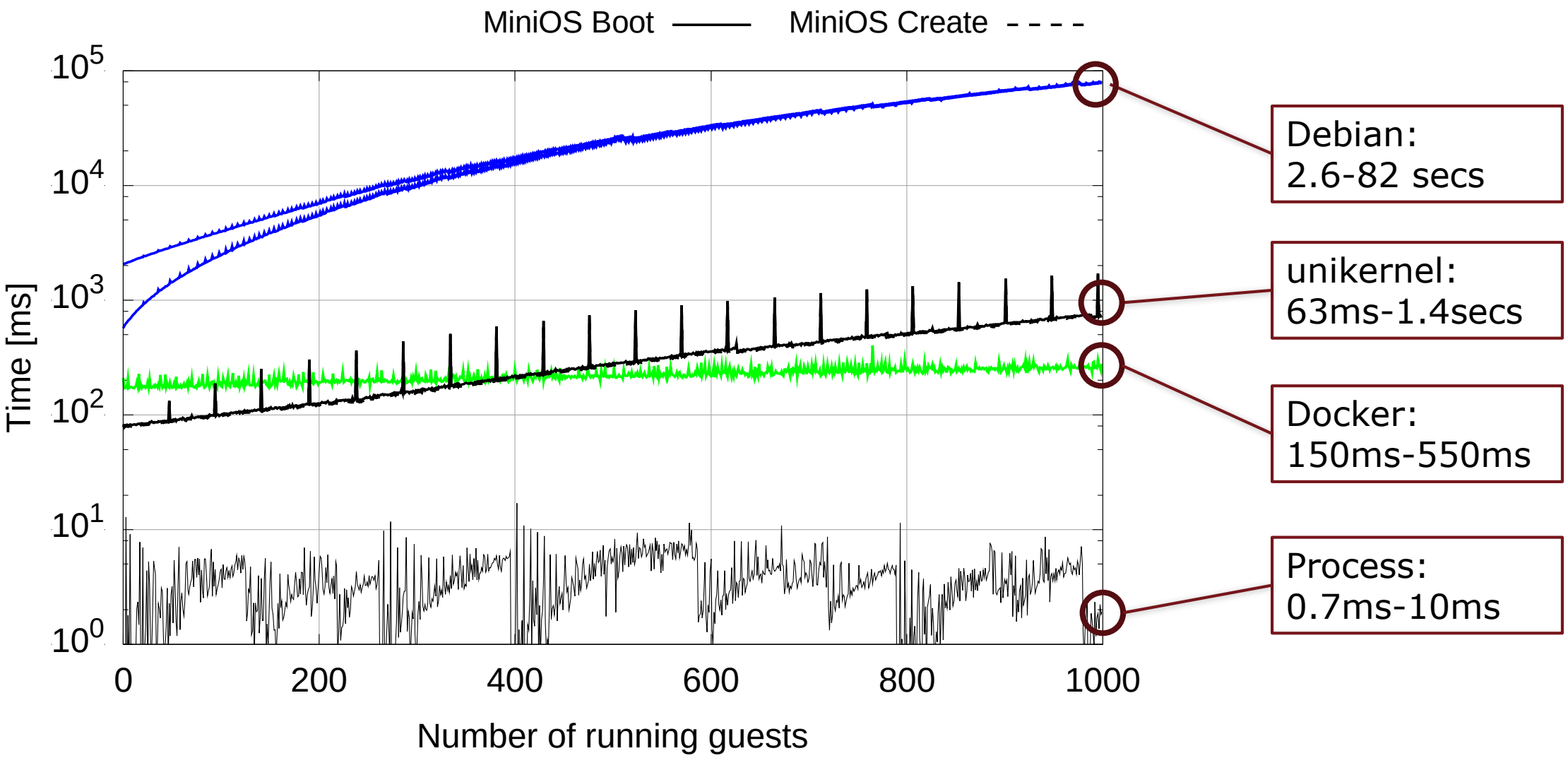
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Example: Instantiation Time Comparison



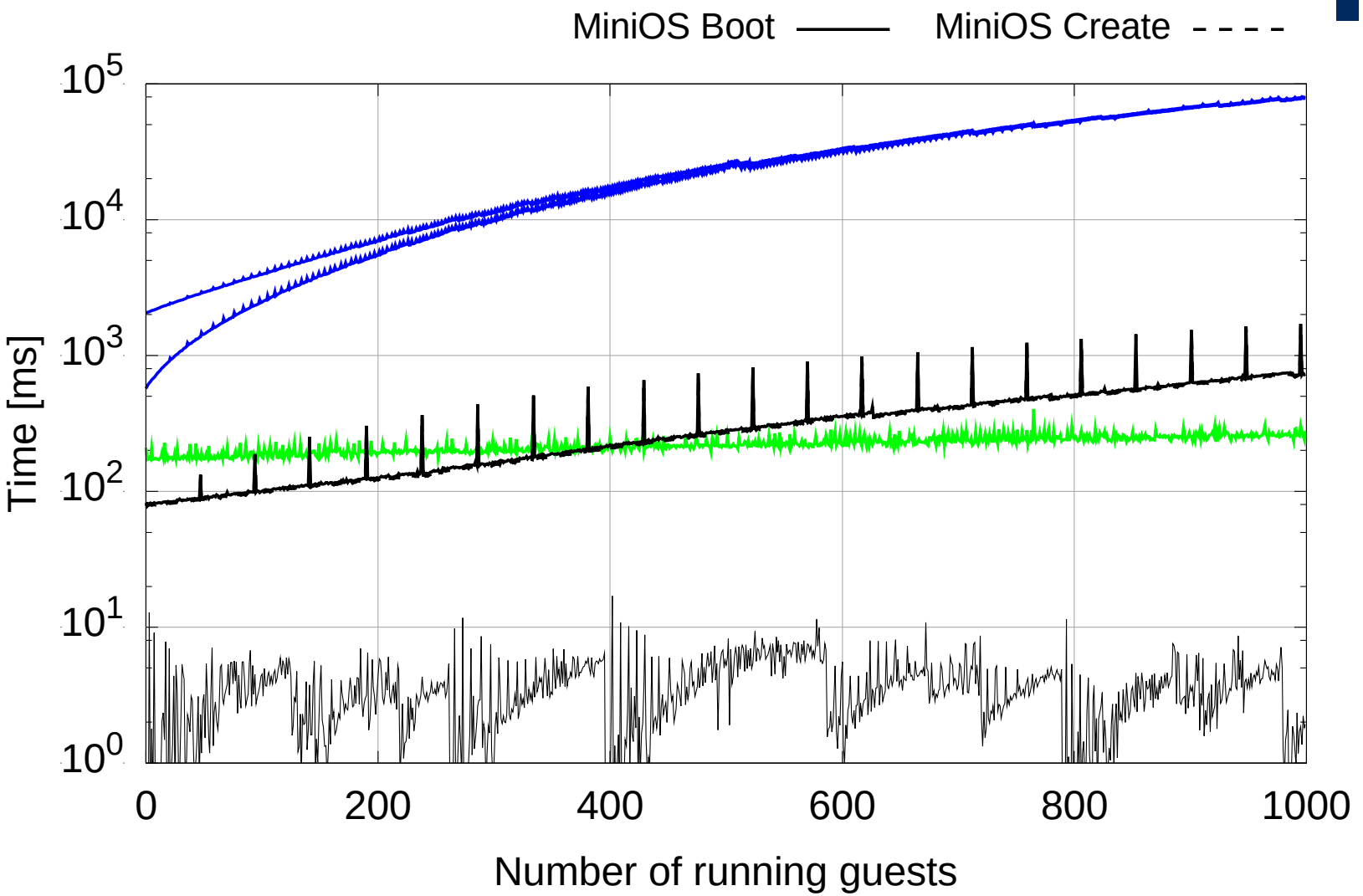
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Example: Instantiation Time Comparison



Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Example: Instantiation Time Comparison



Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Unikernels can instantiate as fast as containers

- Often faster
- Except when many are colocated
  - Speaking of which, what is going wrong there?!
  - If you're interested, talk to me later
  - Bottom line: this is a solvable implementation problem

The Downside

So, unikernels:

- Give you the speed and size of containers
- At the **strong isolation** of VMs

The Downside

- So, unikernels:
  - Give you the speed and size of containers
  - At the **strong isolation** of VMs
- So why isn’t everyone using them already?

The Downside

- So, unikernels:
  - Give you the speed and size of containers
  - At the **strong isolation** of VMs
- So why isn't everyone using them already?

- The big problem is unikernel *development:*
  Optimized unikernels are manually built

The Downside

- So, unikernels:
  - Give you the speed and size of containers
  - At the **strong isolation** of VMs
- So why isn't everyone using them already?

- The big problem is unikernel *development:* Optimized unikernels are manually built
  - Building takes several months or even longer
    - We've done it before, multiple times

The Downside

- So, unikernels:
  - Give you the speed and size of containers
  - At the **strong isolation** of VMs
- So why isn't everyone using them already?

- The big problem is unikernel *development:*
  Optimized unikernels are manually built
  - Building takes several months or even longer
    - We've done it before, multiple times
  - Potentially lather, rinse, repeat for each target application
    - We've done that too…

The Downside

- So, unikernels:
  - Give you the speed and size of containers
  - At the **strong isolation** of VMs
- So why isn't everyone using them already?

- The big problem is unikernel *development:*
  Optimized unikernels are manually built
  - Building takes several months or even longer
    - We've done it before, multiple times
  - Potentially lather, rinse, repeat for each target application
    - We've done that too…

- That's not an effective way of doing things

Unikraft - A Unikernel Framework

Unikraft - A Unikernel Framework

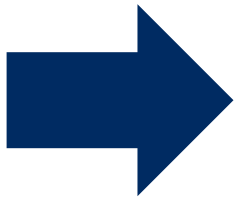
Motivation

- Support wide range of use cases
- Provide common code base for unikernel development
- Simplify building and optimizing
- Support different hypervisors and CPU architectures

Unikraft - A Unikernel Framework

Motivation

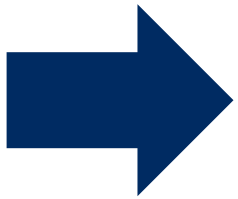
- Support wide range of use cases
- Provide common code base for unikernel development
- Simplify building and optimizing
- Support different hypervisors and CPU architectures



Unikraft - A Unikernel Framework

Motivation

- Support wide range of use cases
- Provide common code base for unikernel development
- Simplify building and optimizing
- Support different hypervisors and CPU architectures



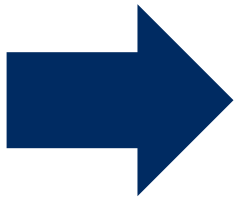
Our Approach

- Decompose OS functionality into libraries
- Unikraft's two components:
  - Library Pool
  - Build Tool

Unikraft - A Unikernel Framework

Motivation

- Support wide range of use cases
- Provide common code base for unikernel development
- Simplify building and optimizing
- Support different hypervisors and CPU architectures



Our Approach

- Decompose OS functionality into libraries
- Unikraft's two components:
  - Library Pool
  - Build Tool

- Started as an internal project at NEC Labs in early 2017
- Made public early on
  - Discussed ideas at Xen Summit 2017
  - Accepted as a Xen incubator project in October 2017
  - First public code release in December 2017

The Unikraft Way

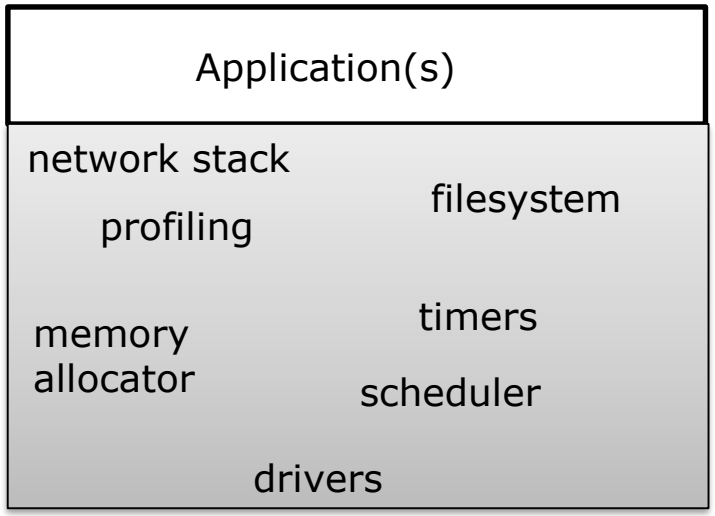
Decompose OS into a set of libraries (“Everything is a library”)

Recompose them to meet the needs of particular applications

The Unikraft Way

Decompose OS into a set of libraries (“Everything is a library”)

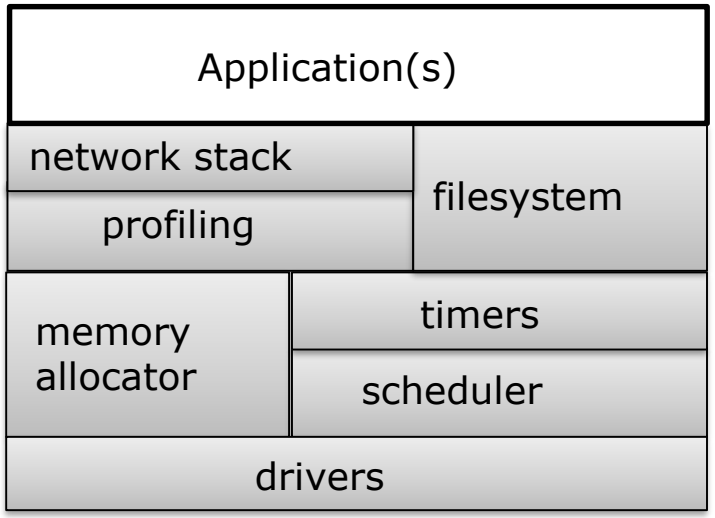
Recompose them to meet the needs of particular applications



The Unikraft Way

Decompose OS into a set of libraries (“Everything is a library”)

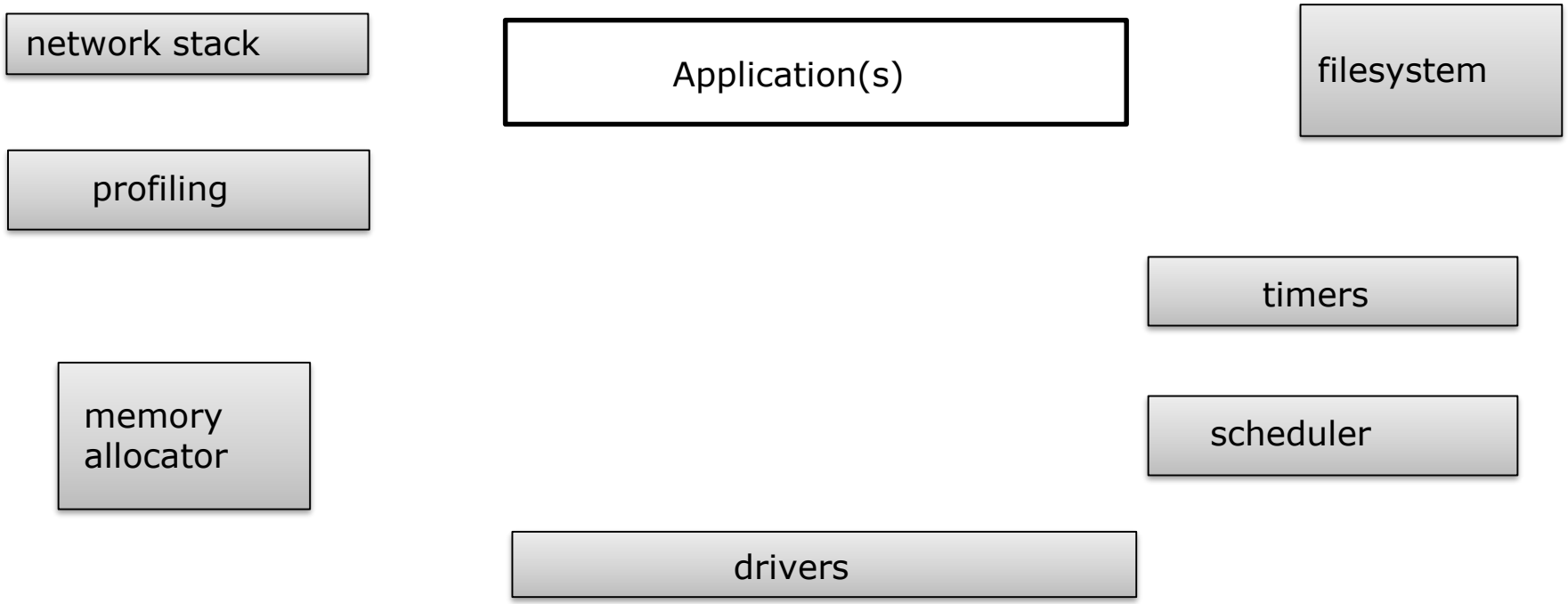
Recompose them to meet the needs of particular applications



The Unikraft Way

Decompose OS into a set of libraries (“Everything is a library”)

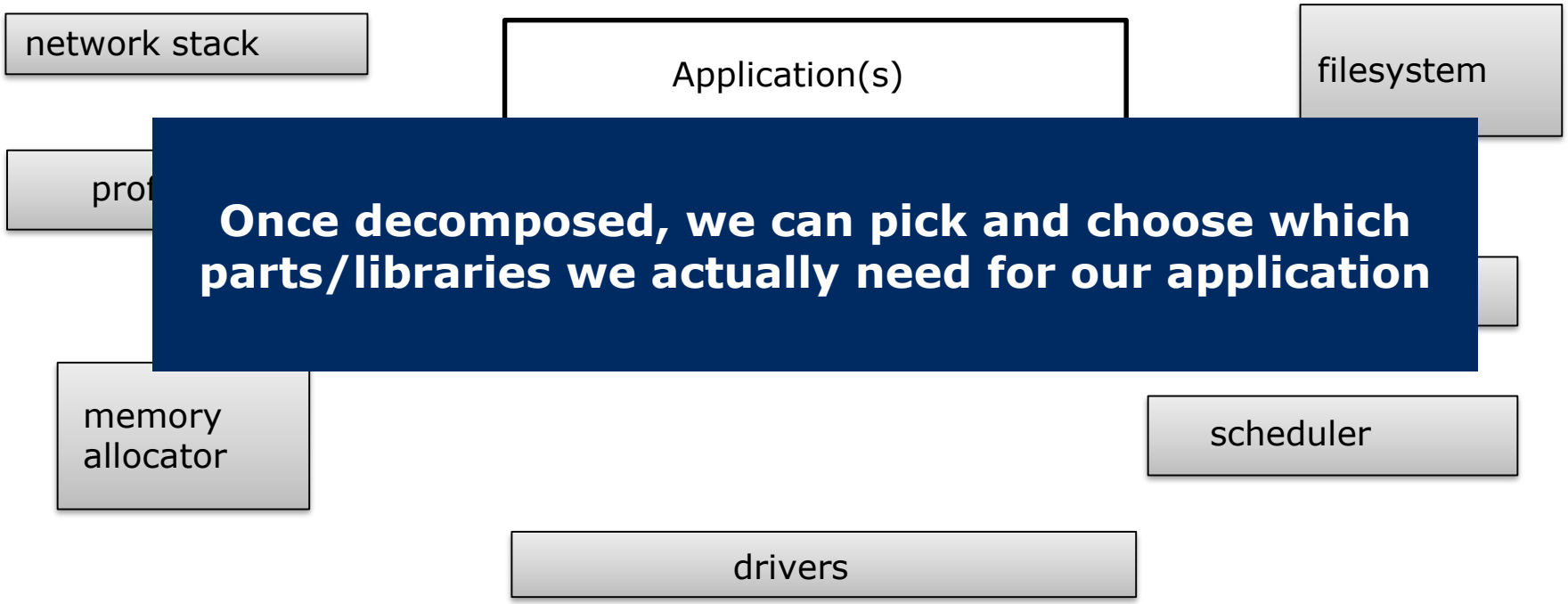
Recompose them to meet the needs of particular applications



The Unikraft Way

Decompose OS into a set of libraries (“Everything is a library”)

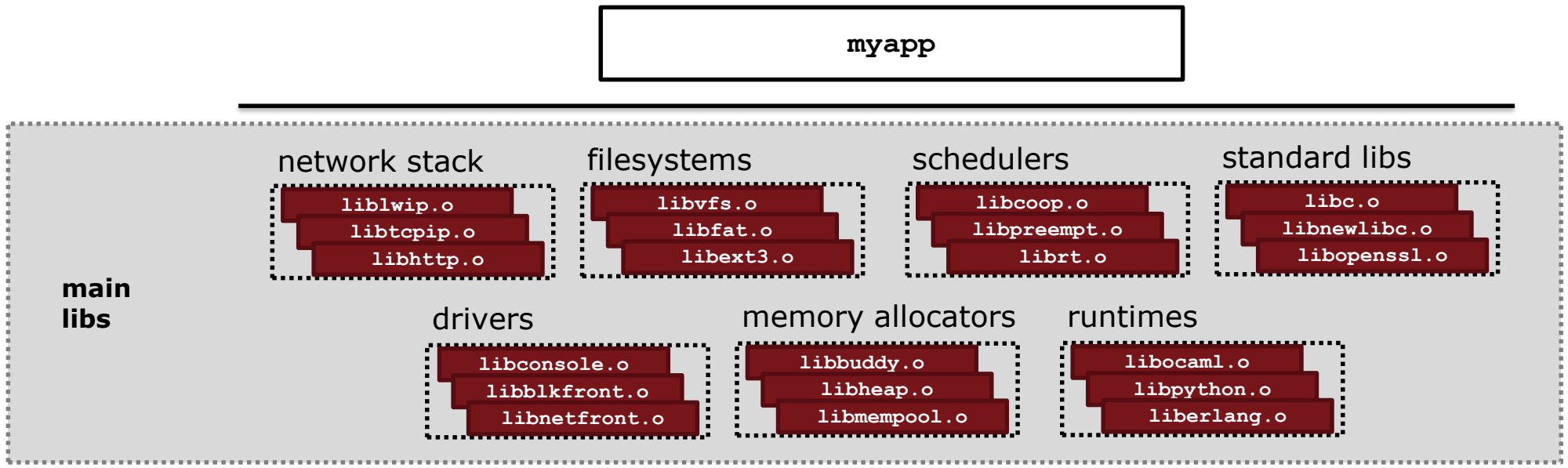
Recompose them to meet the needs of particular applications



Unikraft Overview – Everything as a Library

① **Select / build / port Application**

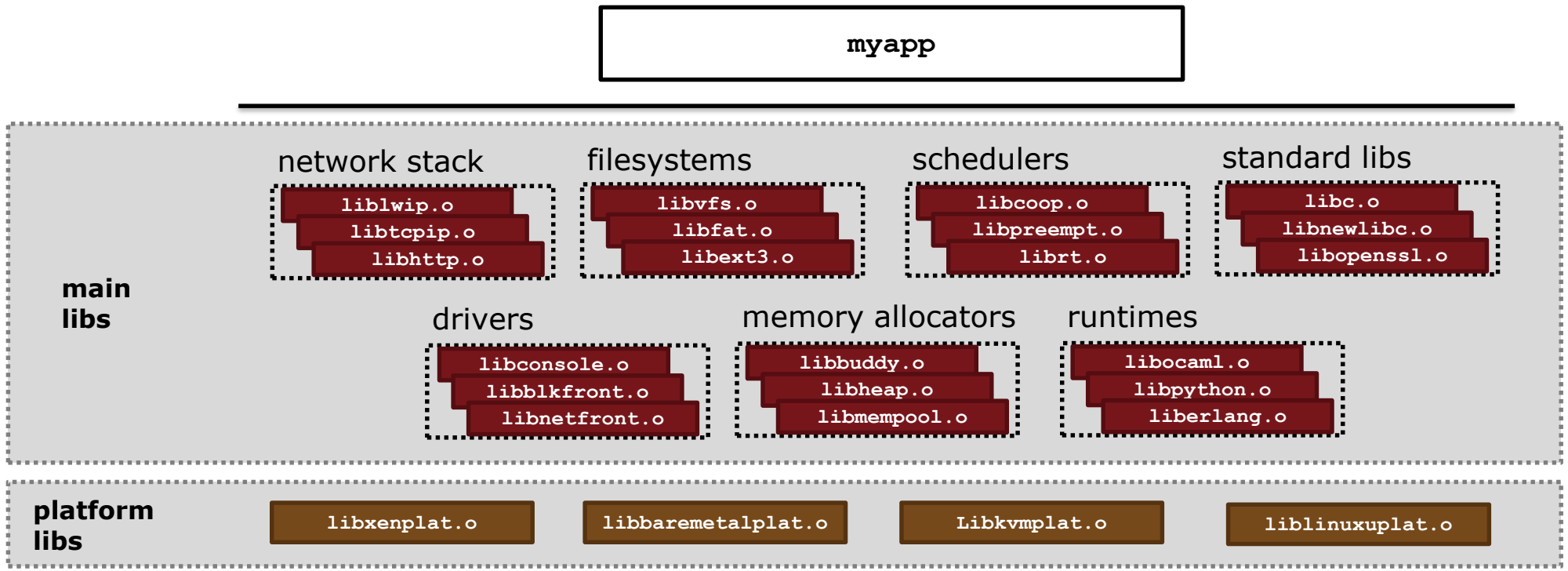
Unikraft Overview – Everything as a Library



1. **Select / build / port Application**

2. **Select and configure libraries**

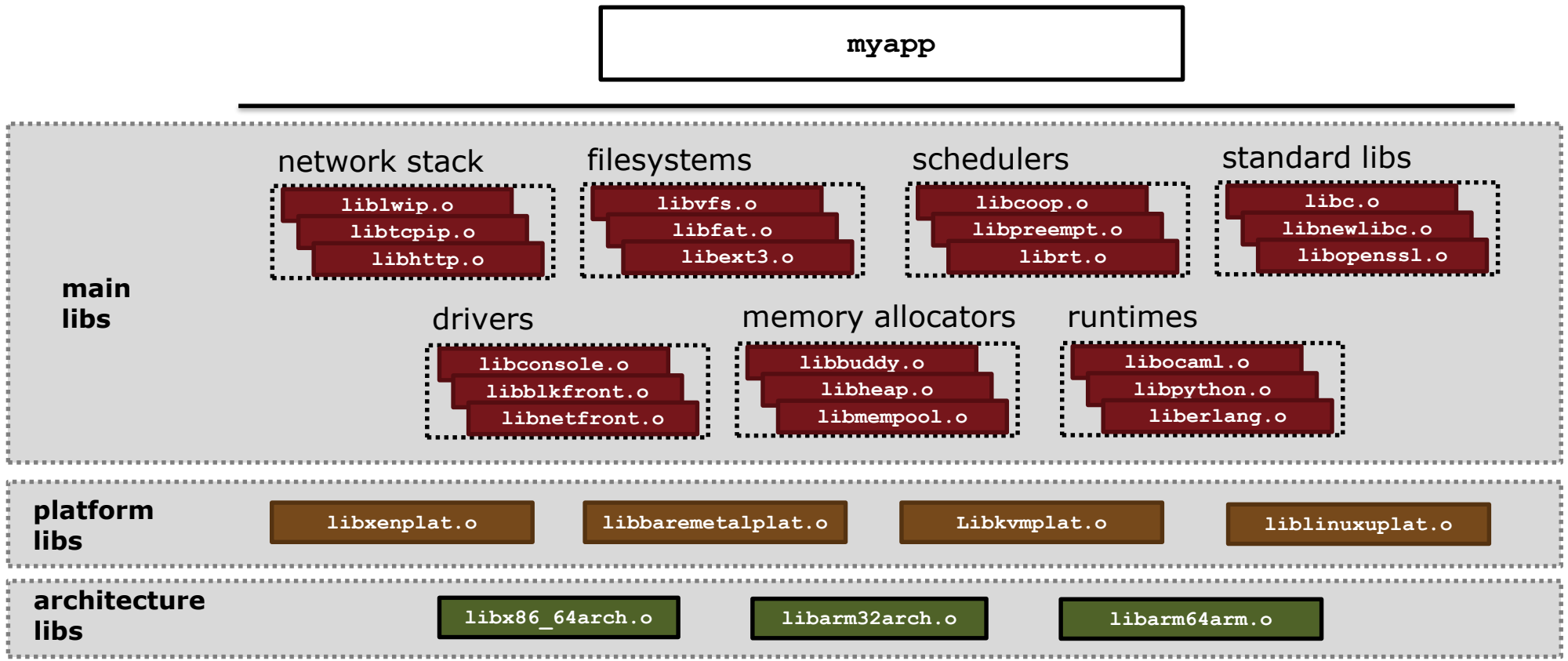
Unikraft Overview – Everything as a Library



① **Select / build / port Application**

② **Select and configure libraries**

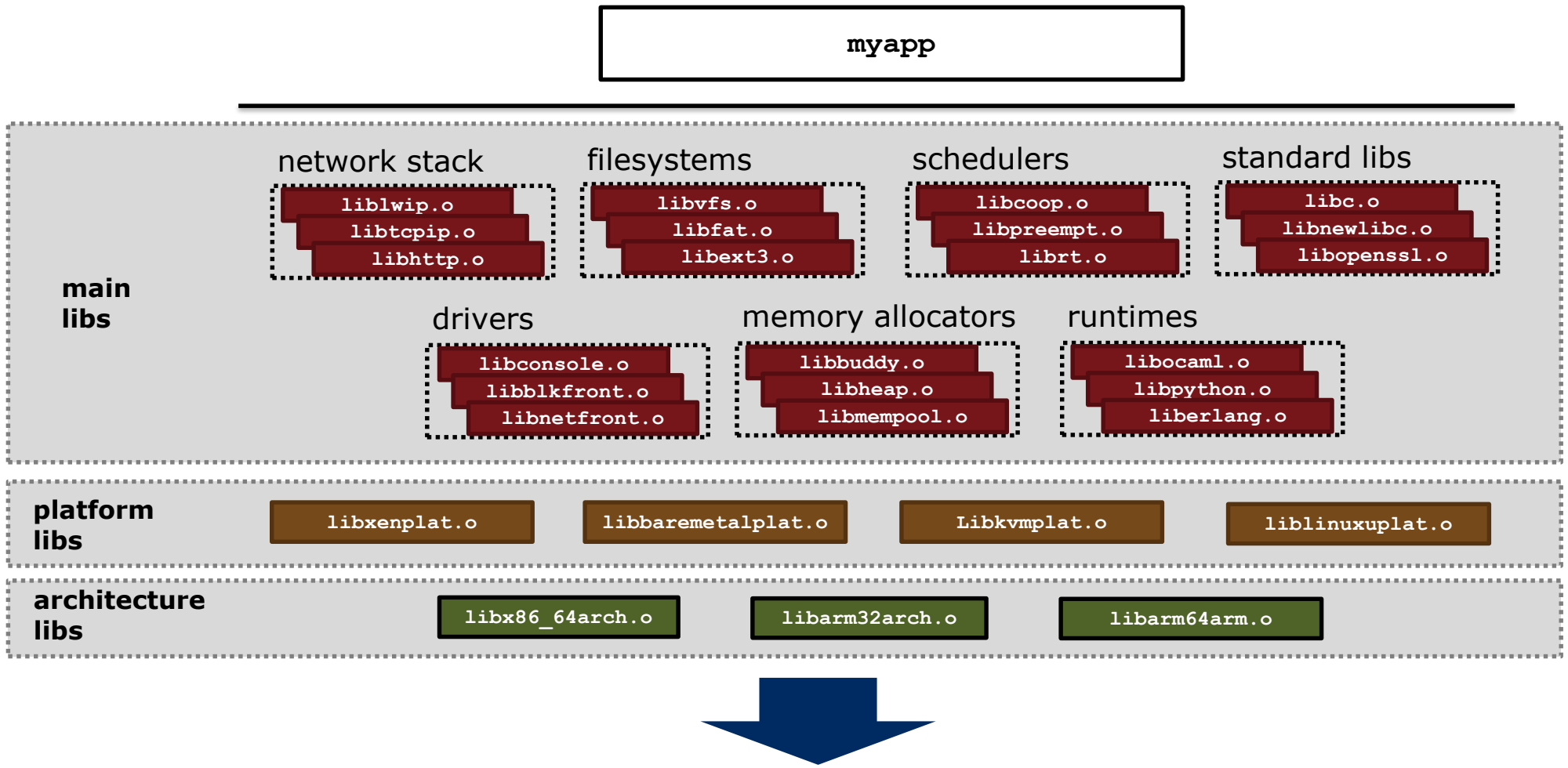
Unikraft Overview – Everything as a Library



① **Select / build / port Application**

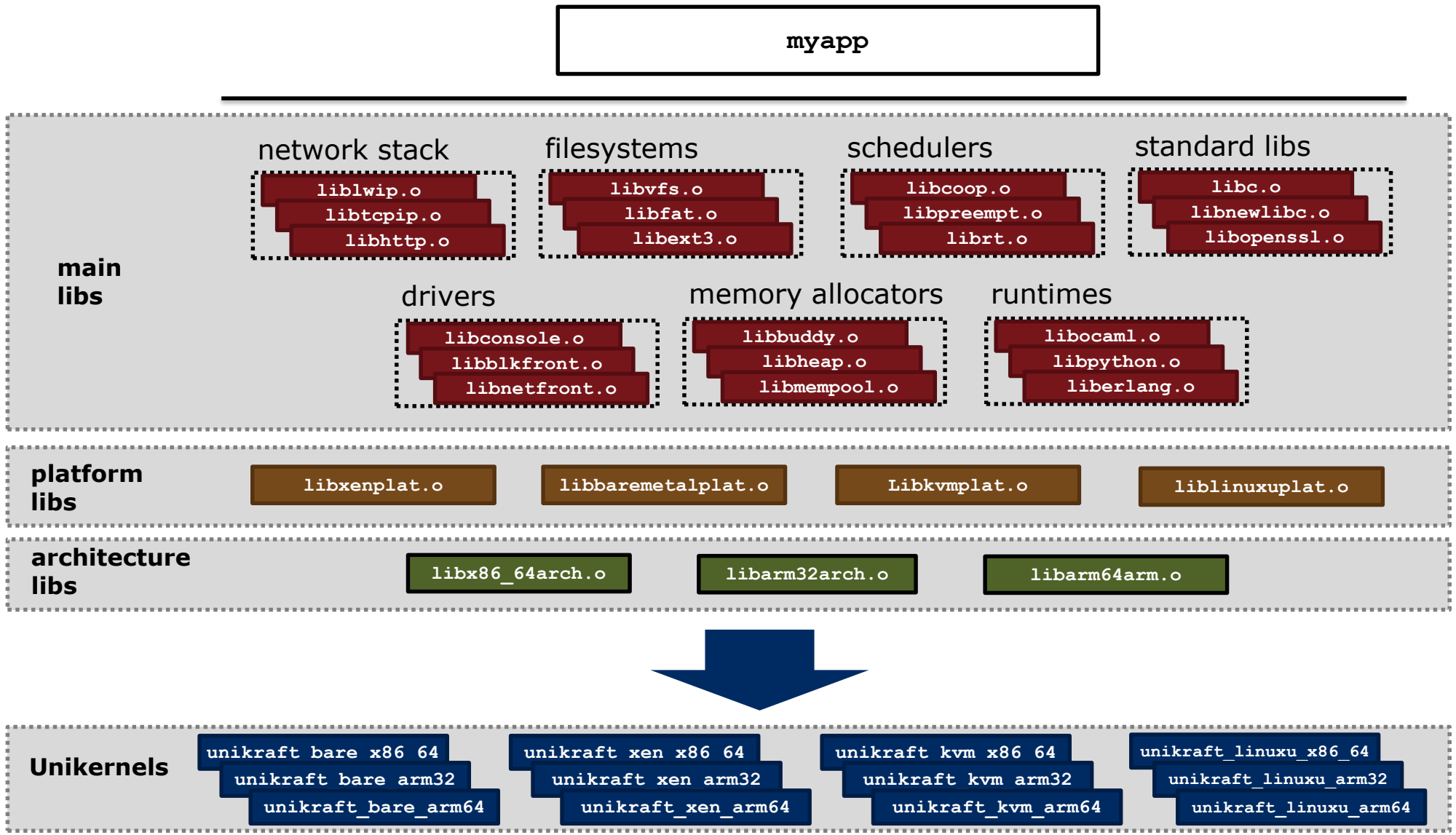
② **Select and configure libraries**

Unikraft Overview – Everything as a Library



1. **Select / build / port Application**
2. **Select and configure libraries**
3. **Build**

Unikraft Overview – Everything as a Library



Two Library Types

- **Built-in**: functionality specific to Unikraft, live in the main unikraft repo
  - ukboot
  - ukschedpreempt
  - …

Two Library Types

Built-in: functionality specific to Unikraft, live in the main unikraft repo

- ukboot
- ukschedpreempt
- …

External: software projects external to Unikraft, have their own unikraft-lib repos

- lwip
- micropython
- …

Example System

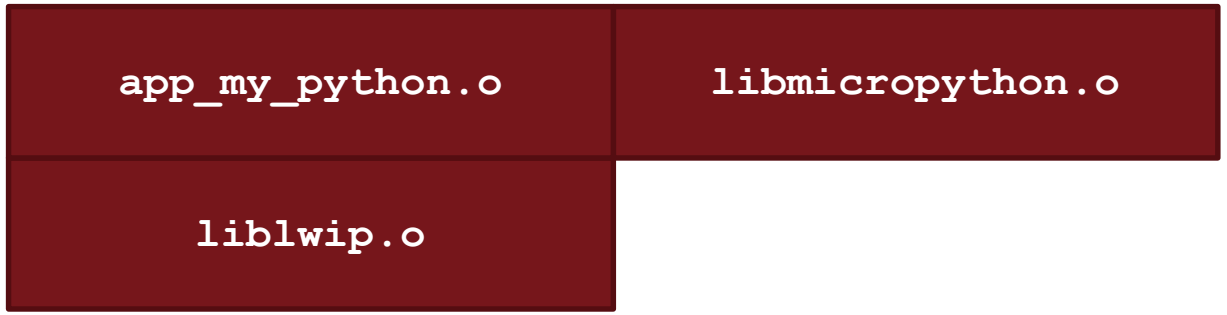
- Micropython Unikernel for KVM on x86_64

Example System

- Micropython Unikernel for KVM on x86_64

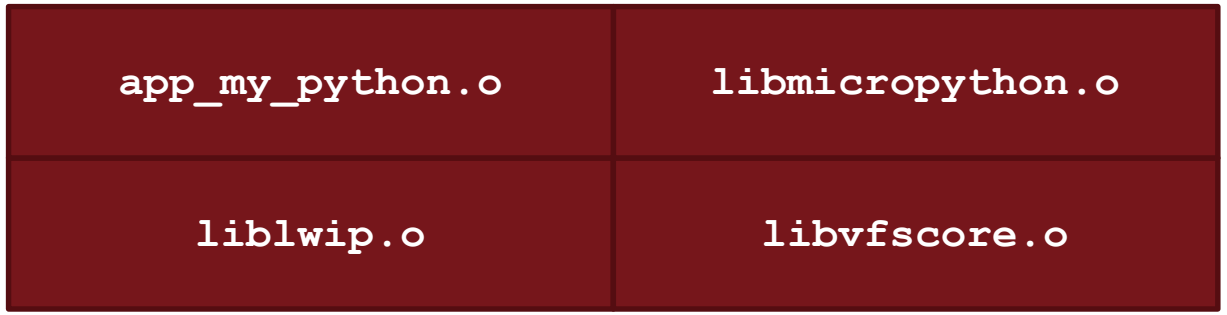
Example System

- Micropython Unikernel for KVM on x86_64



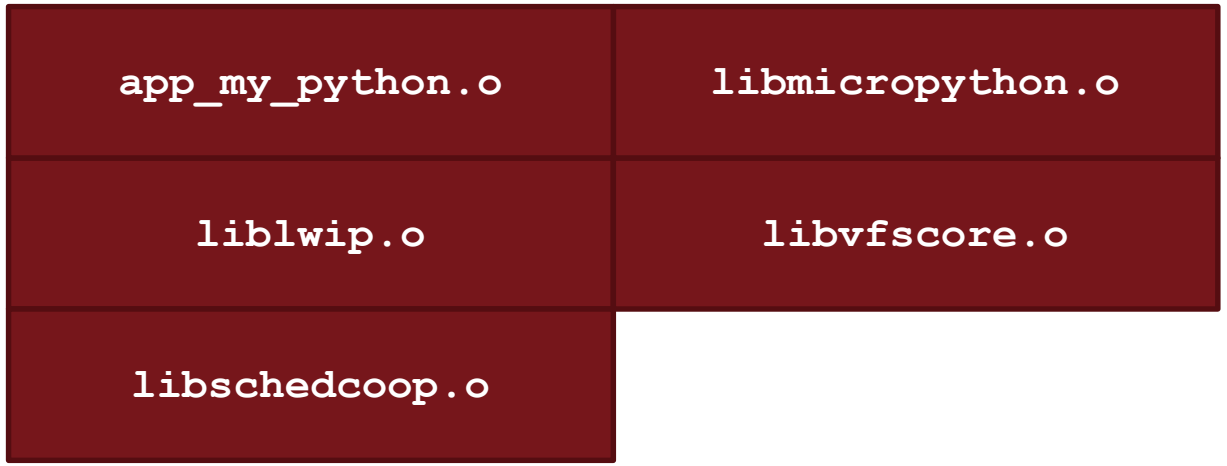
Example System

- Micropython Unikernel for KVM on x86_64



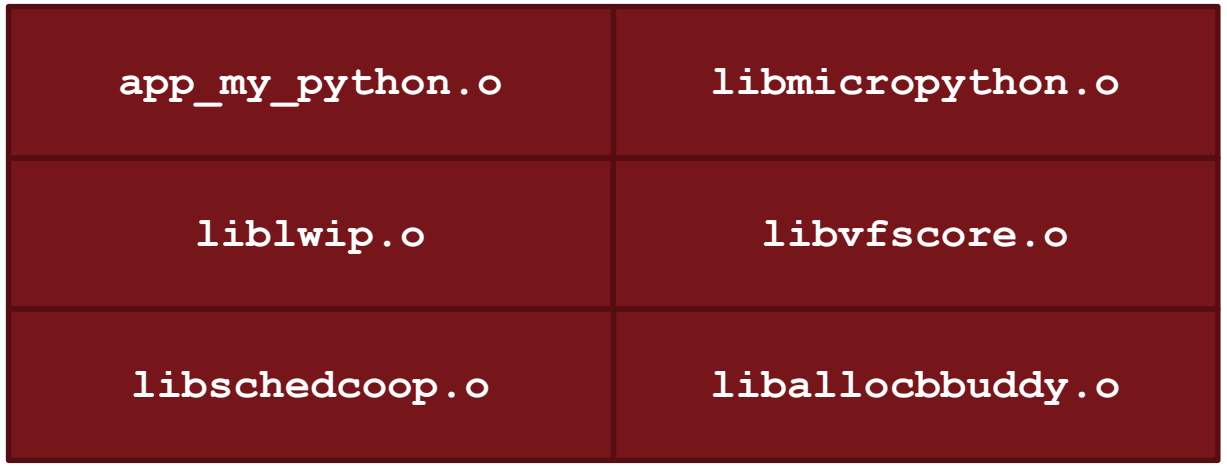
Example System

- Micropython Unikernel for KVM on x86_64



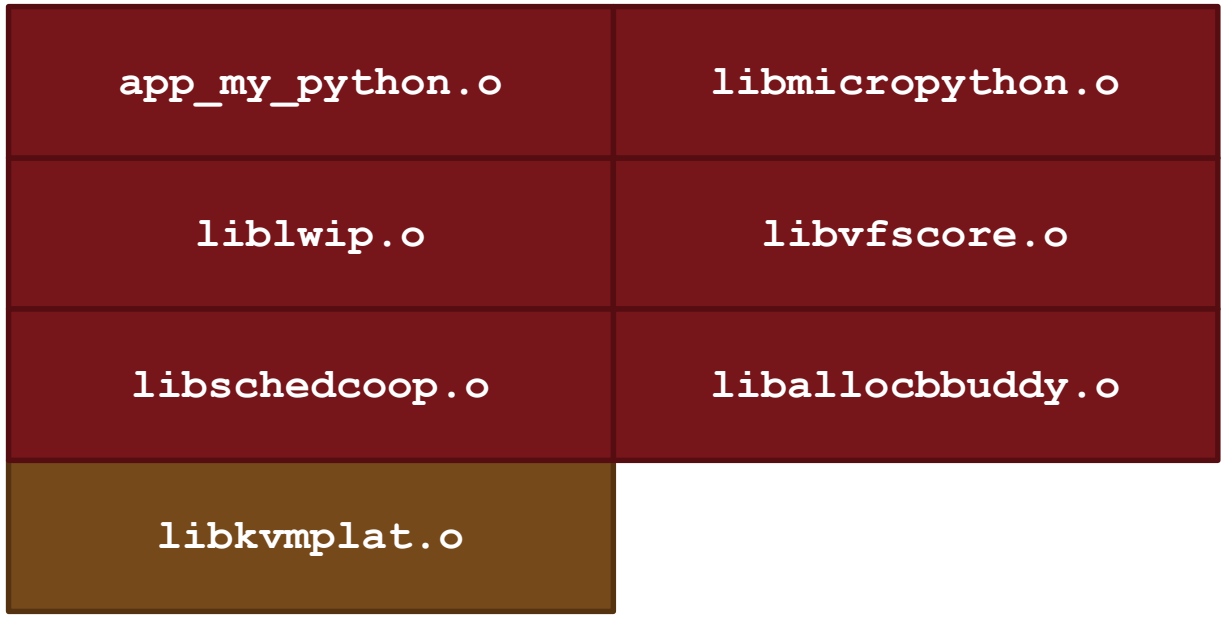
Example System

Micropython Unikernel for KVM on x86_64



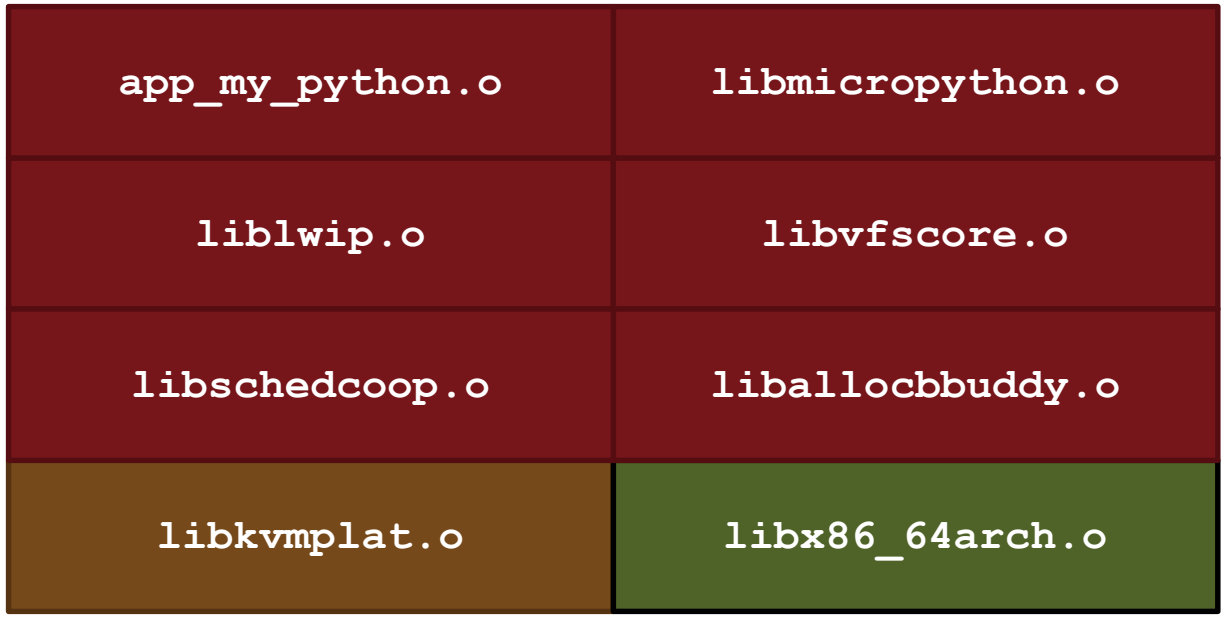
Example System

Micropython Unikernel for KVM on x86_64



Example System

Micropython Unikernel for KVM on x86_64

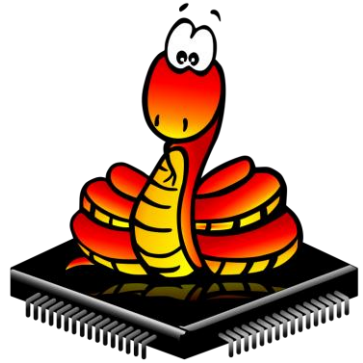


Example System

Micropython Unikernel for KVM on x86_64

app_my_python.o	libmicropython.o
liblwip.o	libvfscore.o
libschedcoop.o	liballocbbuddy.o
libkvmplat.o	libx86_64arch.o

=



Unikernel

Putting Things Together – The Unikraft Build Tool

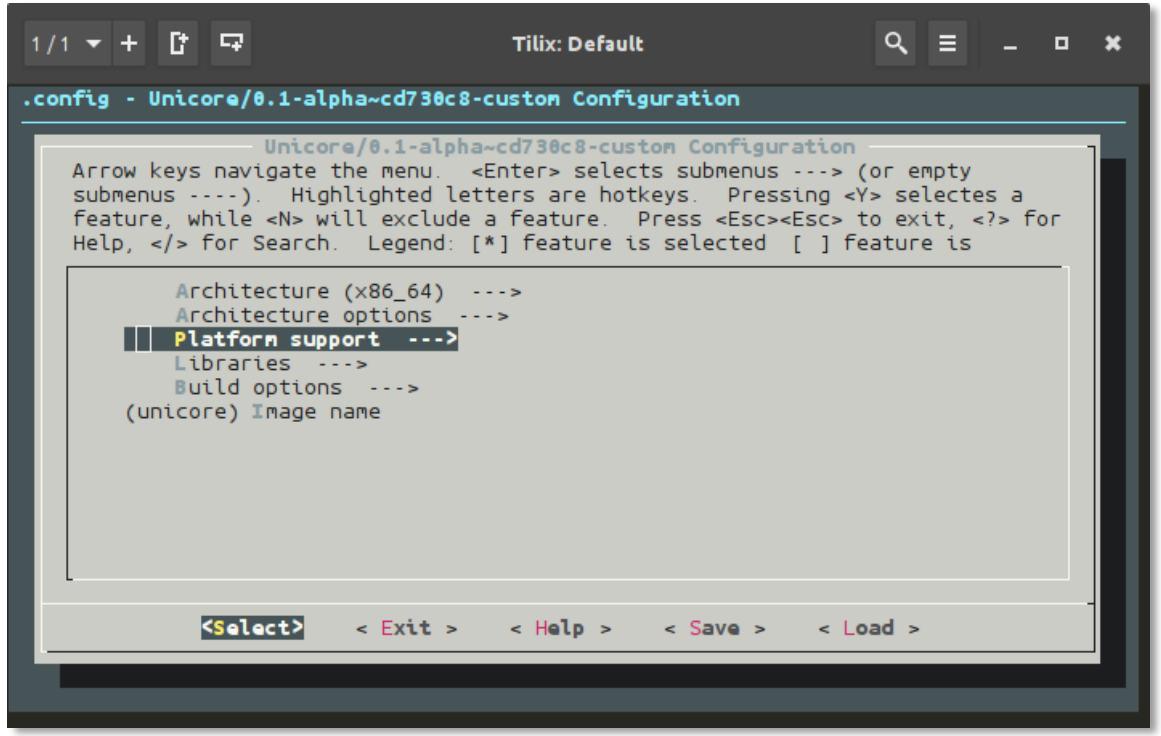
Putting Things Together – The Unikraft Build Tool

- Kconfig/Makefile based

Putting Things Together – The Unikraft Build Tool

Kconfig/Makefile based

`make menuconfig`

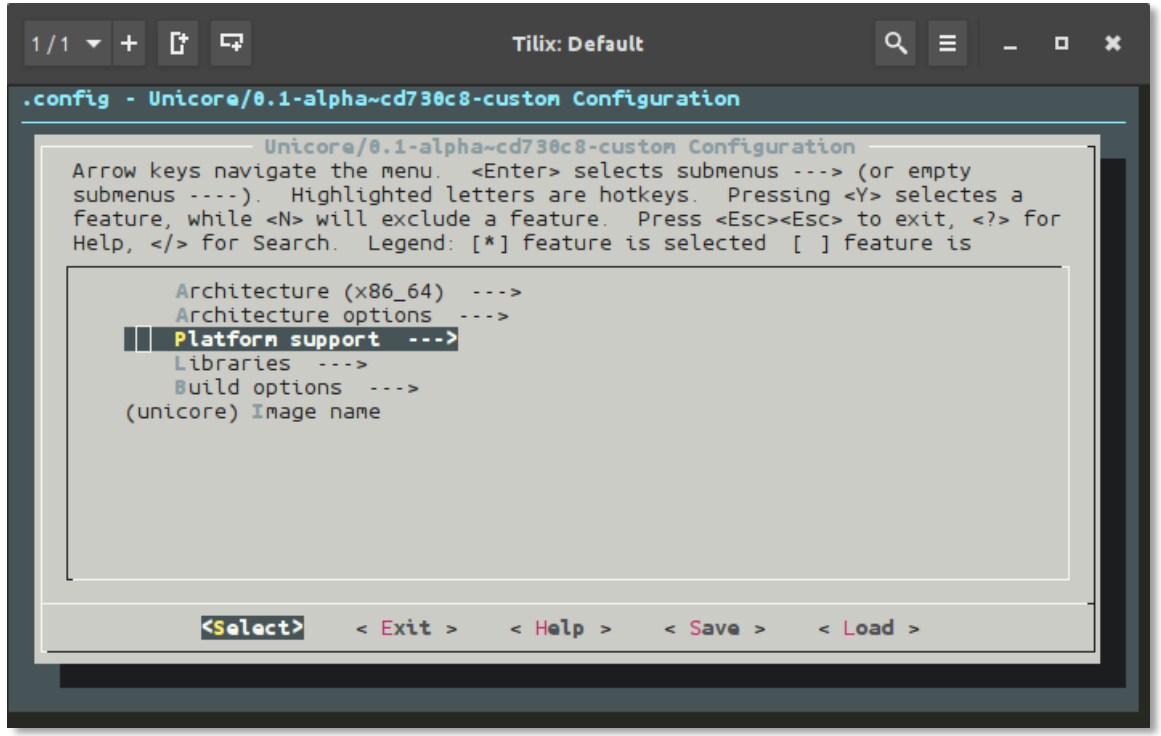


Putting Things Together – The Unikraft Build Tool

Kconfig/Makefile based

`make menuconfig`

- Choose options in the menu that you want for your application

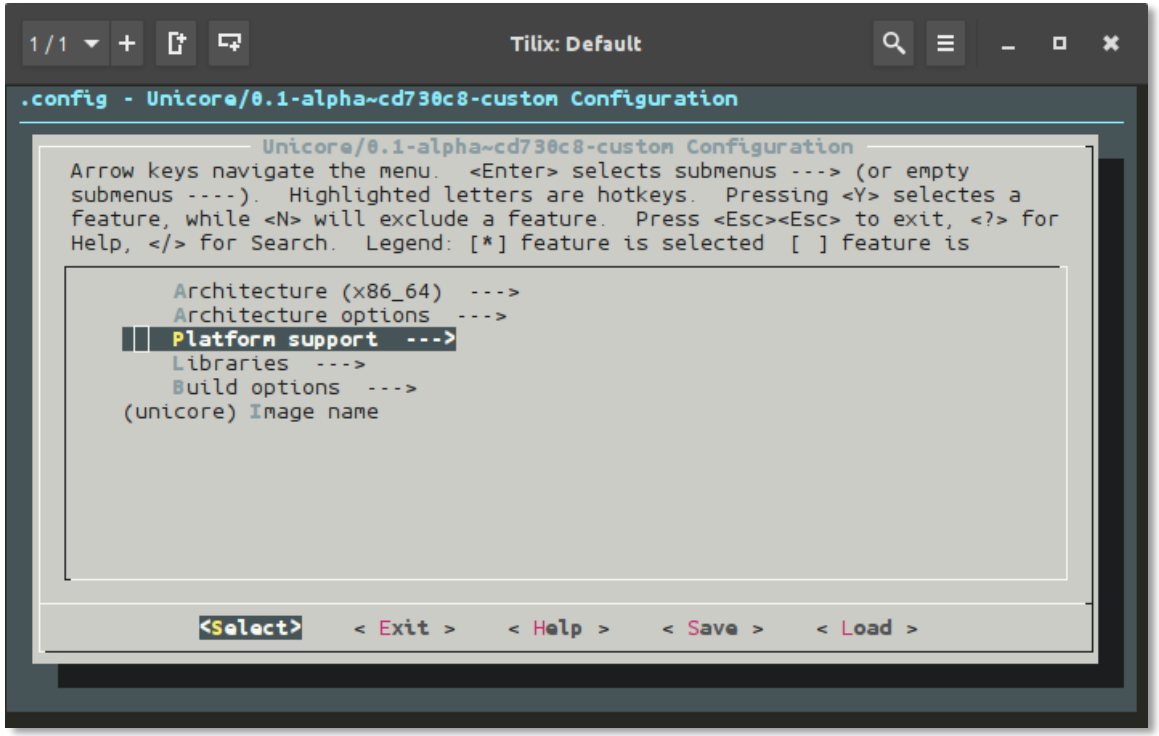


Putting Things Together – The Unikraft Build Tool

Kconfig/Makefile based

`make menuconfig`

- Choose options in the menu that you want for your application
- Choose your target platform(s) (currently: Xen, KVM, Linux) and architectures

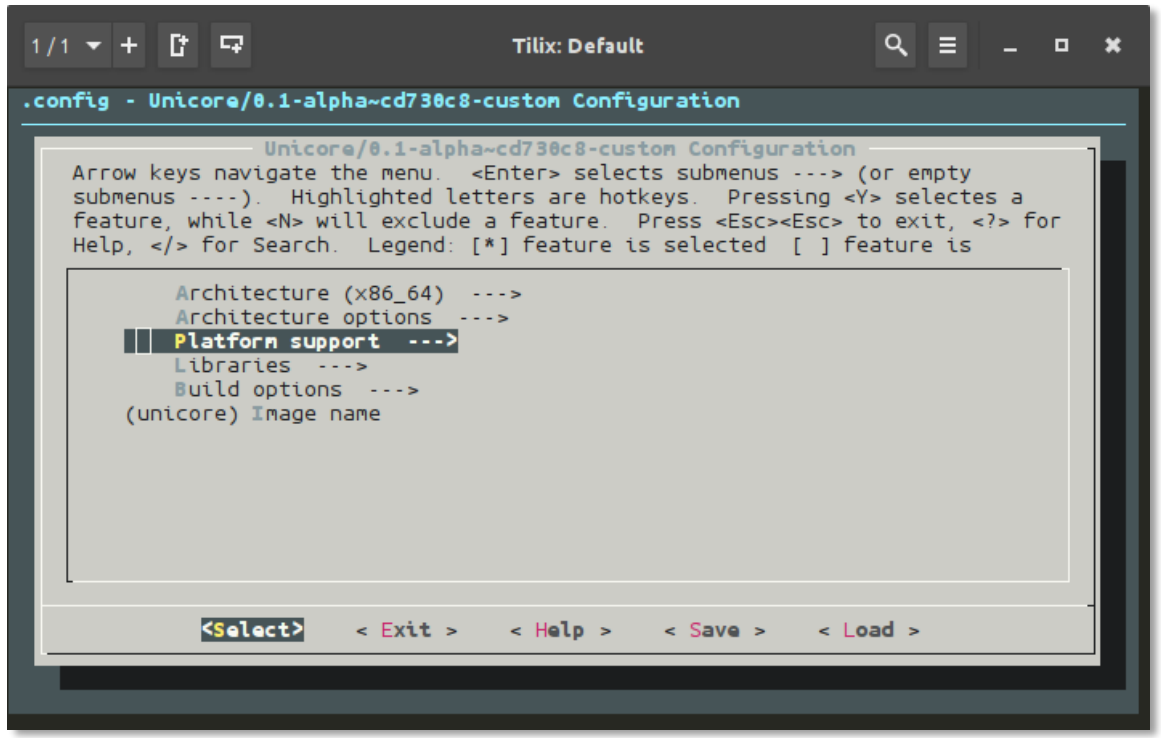


Putting Things Together – The Unikraft Build Tool

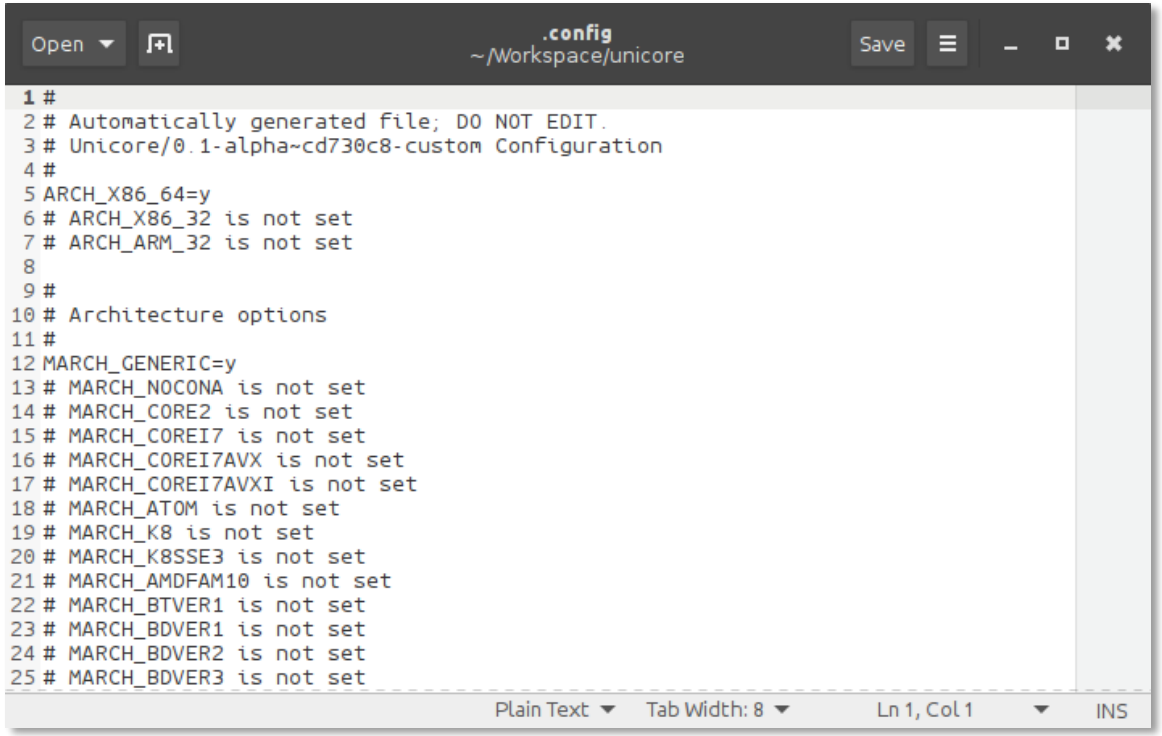
- Kconfig/Makefile based

  `make menuconfig`

  - Choose options in the menu that you want for your application
  - Choose your target platform(s) (currently: Xen, KVM, Linux) and architectures
- Save config and `make`



.config



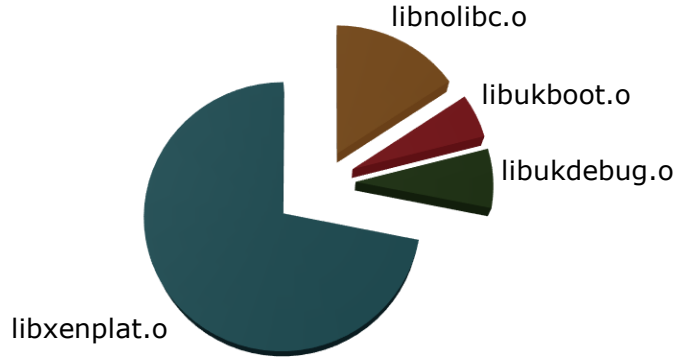
A Baseline Example…

- *Xen PV x86_64 binary*

A Baseline Example…

Xen PV x86_64 binary

unikraft_xen-x86_64.o



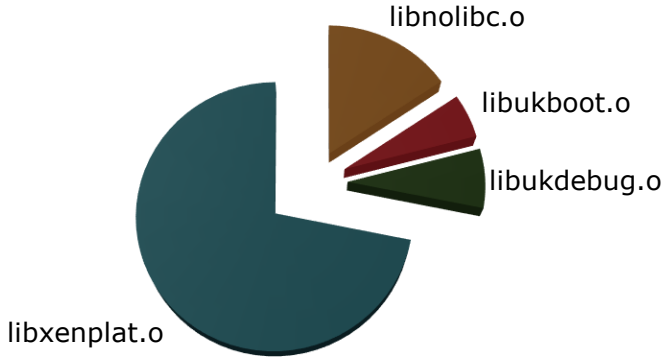
Final linking

unikraft_xen-x86_64
(32,7kB)

A Baseline Example…

Xen PV x86_64 binary

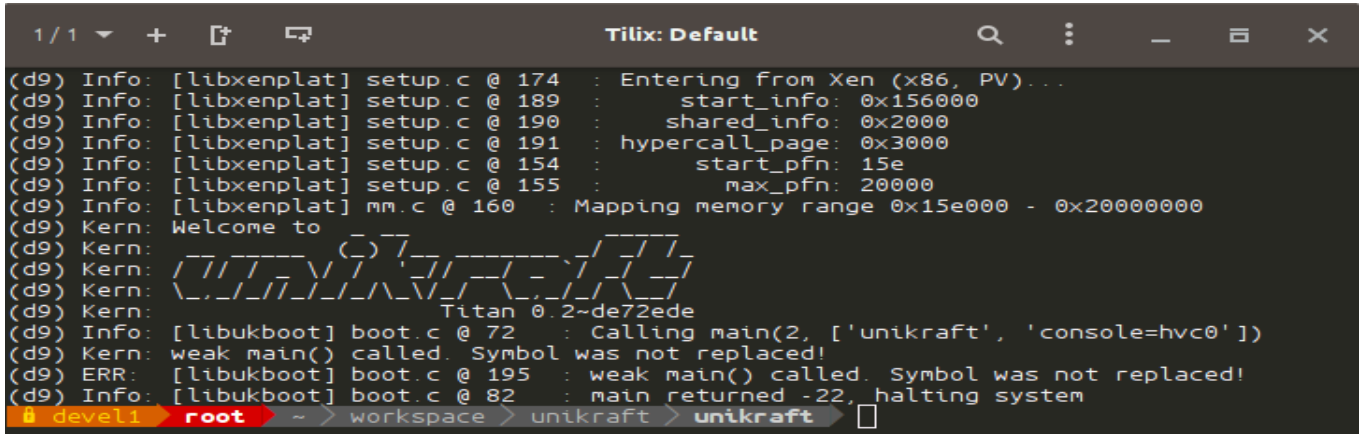
unikraft_xen-x86_64.o



Final linking

unikraft_xen-x86_64
(32,7kB)

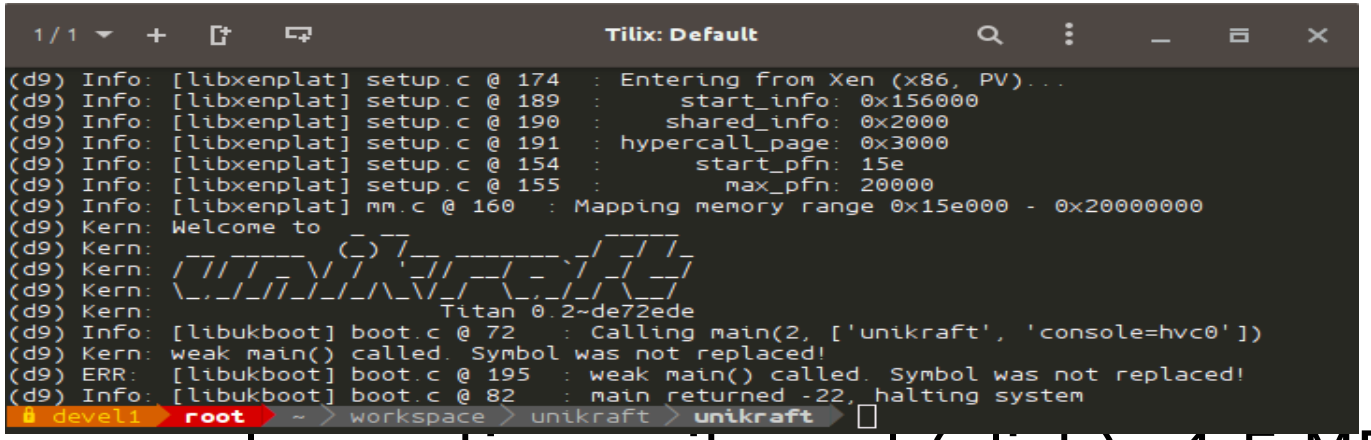
Boots and prints messages to debug console (with min. 208kB RAM)



A Baseline Example…

- *Xen PV x86_64 binary*

- Boots and prints messages to debug console (with min. 208kB RAM)



- More functional example: routing unikernel (click): 4.5 MB (8 MB RAM)

Building a Unikraft Hello World App

Repo Structure

- Clone the main Unikraft repo

```
git clone git://xenbits.xen.org/unikraft/unikraft.git
```

- Clone any external library repos

```
git clone git://xenbits.xen.org/unikraft/libs/newlib.git
```

- Create repo for the actual application

Repo Structure

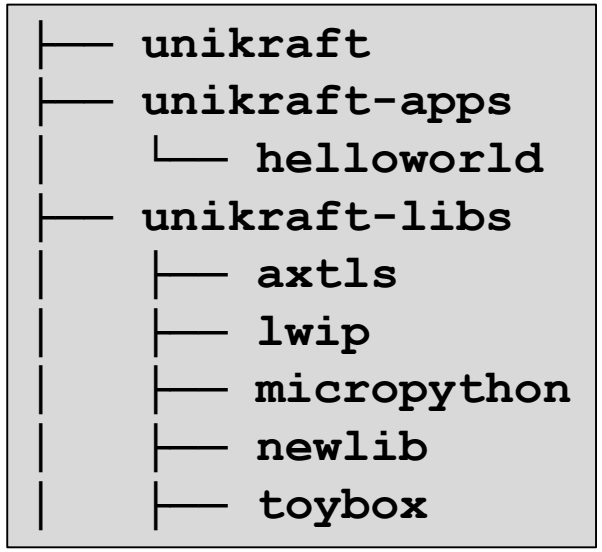
- Clone the main Unikraft repo

```
git clone git://xenbits.xen.org/unikraft/unikraft.git
```

- Clone any external library repos

```
git clone git://xenbits.xen.org/unikraft/libs/newlib.git
```

- Create repo for the actual application



Repo Structure

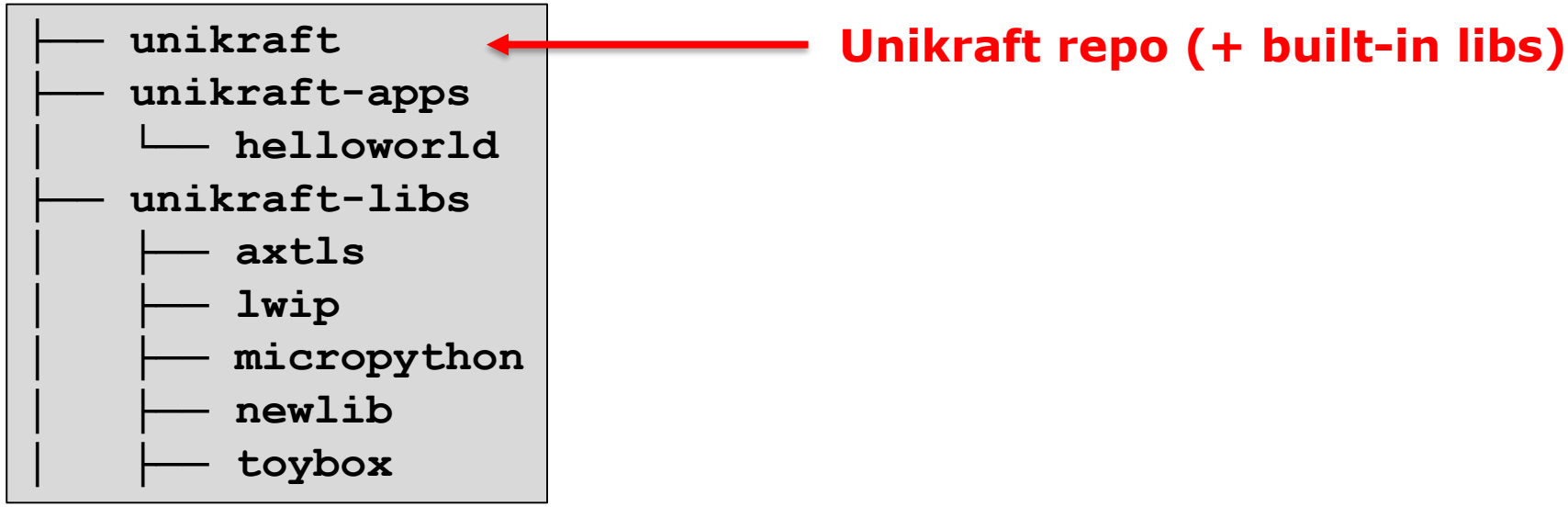
- Clone the main Unikraft repo

```
git clone git://xenbits.xen.org/unikraft/unikraft.git
```

- Clone any external library repos

```
git clone git://xenbits.xen.org/unikraft/libs/newlib.git
```

- Create repo for the actual application



Repo Structure

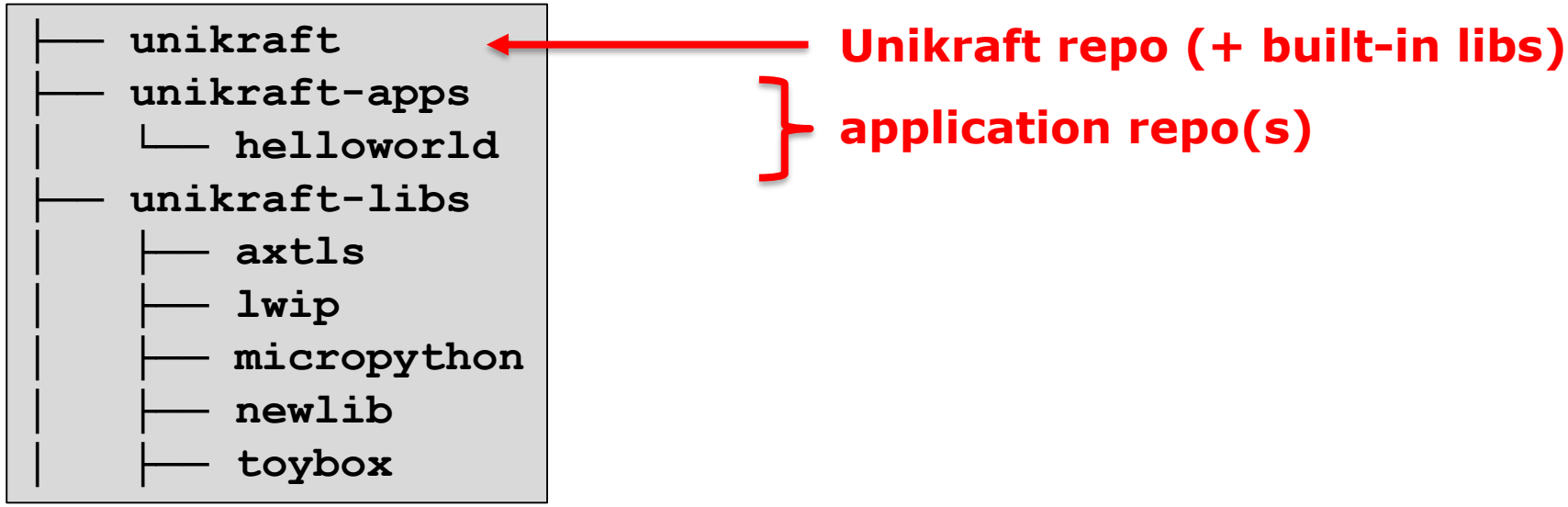
- Clone the main Unikraft repo

```
git clone git://xenbits.xen.org/unikraft/unikraft.git
```

- Clone any external library repos

```
git clone git://xenbits.xen.org/unikraft/libs/newlib.git
```

- Create repo for the actual application



Repo Structure

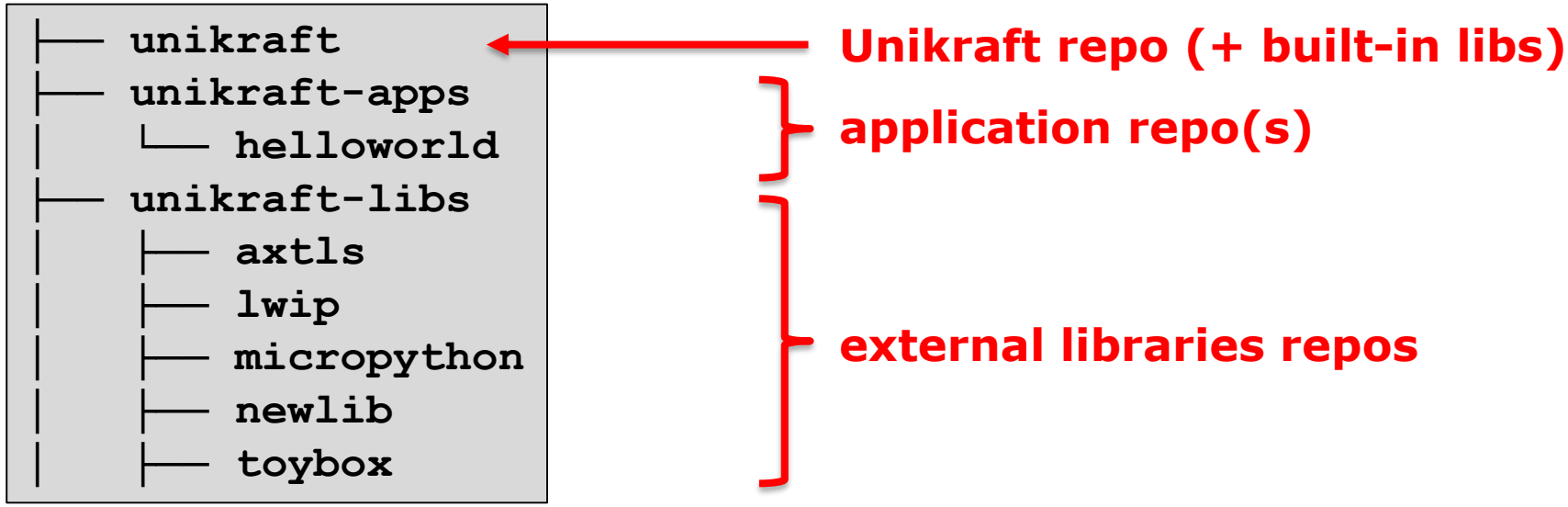
- Clone the main Unikraft repo

```
git clone git://xenbits.xen.org/unikraft/unikraft.git
```

- Clone any external library repos

```
git clone git://xenbits.xen.org/unikraft/libs/newlib.git
```

- Create repo for the actual application



Hello World – Four Required Files (I)

Makefile: specify where the main Unikraft repo is,
as well as repos for external libraries

```
UK_ROOT ?= $(PWD)/../../unikraft
UK_LIBS ?= $(PWD)/../../unikraft-libs
LIBS := $(UK_LIBS)/newlib

all:
	@make -C $(UK_ROOT) A=$(PWD) L=$(LIBS)

$(MAKECMDGOALS):
	@make -C $(UK_ROOT) A=$(PWD) L=$(LIBS) $(MAKECMDGOALS)
```

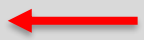
Hello World – Four Required Files (I)

Makefile: specify where the main Unikraft repo is,
as well as repos for external libraries

```
UK_ROOT ?= $(PWD)/../../unikraft
UK_LIBS ?= $(PWD)/../../unikraft-libs
LIBS := $(UK_LIBS)/newlib

all:
	@make -C $(UK_ROOT) A=$(PWD) L=$(LIBS)

$(MAKECMDGOALS):
	@make -C $(UK_ROOT) A=$(PWD) L=$(LIBS) $(MAKECMDGOALS)
```



path to unikraft repo

Hello World – Four Required Files (I)

Makefile: specify where the main Unikraft repo is,
as well as repos for external libraries

```
UK_ROOT ?= $(PWD)/../../unikraft
UK_LIBS ?= $(PWD)/../../unikraft-libs
LIBS := $(UK_LIBS)/newlib

all:
	@make -C $(UK_ROOT) A=$(PWD) L=$(LIBS)

$(MAKECMDGOALS):
	@make -C $(UK_ROOT) A=$(PWD) L=$(LIBS) $(MAKECMDGOALS)
```

← path to unikraft repo

← path to external libs

Hello World – Four Required Files (I)

Makefile: specify where the main Unikraft repo is,
as well as repos for external libraries

```
UK_ROOT ?= $(PWD)/../../unikraft
UK_LIBS ?= $(PWD)/../../unikraft-libs
LIBS := $(UK_LIBS)/newlib

all:
	@make -C $(UK_ROOT) A=$(PWD) L=$(LIBS)

$(MAKECMDGOALS):
	@make -C $(UK_ROOT) A=$(PWD) L=$(LIBS) $(MAKECMDGOALS)
```

← path to unikraft repo

← path to external libs

← external libs needed

Hello World – Four Required Files (II)

Makefile.uk: specifies the sources to build for the application

```
$(eval $(call addlib,apphelloworld))

APPHELLOWORLD_SRCS-y += $(APPHELLOWORLD_BASE)/main.c
```

Hello World – Four Required Files (II)

Makefile.uk: specifies the sources to build for the application

```
$(eval $(call addlib,apphelloworld))

APPHELLOWORLD_SRCS-y += $(APPHELLOWORLD_BASE)/main.c
```

register app with unikraft build system

Hello World – Four Required Files (II)

Makefile.uk: specifies the sources to build for the application

```
$(eval $(call addlib,apphelloworld))

APPHELLOWORLD_SRCS-y += $(APPHELLOWORLD_BASE)/main.c
```

register app with unikraft build system

Add main.c to build

Hello World – Four Required Files (III)

Config.uk: to populate Unikraft's menu with application-specific option

```
### Invisible option for dependencies
config APPHELLOWORLD_DEPENDENCIES
        bool
        default y
        select LIBNOLIBC if !HAVE_LIBC

### App configuration
config APPHELLOWORLD_PRINTARGS
        bool "Print arguments"
        default y
        help
          Prints argument list (argv) to stdout
```

Hello World – Four Required Files (IV)

main.c: source file to provide (at least) a `main()` function

```
#include <stdio.h>
/* Import user configuration: */
#include <uk/config.h>

int main(int argc, char *argv[])
{
	printf("Hello world!\n");
#if CONFIG_APPHELLOWORLD_PRINTARGS
	int i;
	printf("Arguments:s");
	for (i=0; i<argc; ++i)
		printf(" \"%s\"", argv[i]);
	printf("\n");
#endif
}
```

Hello World – Four Required Files (IV)

main.c: source file to provide (at least) a `main()` function

```
#include <stdio.h>
/* Import user configuration: */
#include <uk/config.h>

int main(int argc, char *argv[])
{
        printf("Hello world!\n");
#if CONFIG_APPHELLOWORLD_PRINTARGS
        int i;
        printf("Arguments:s");
        for (i=0; i<argc; ++i)
                printf(" \"%s\"", argv[i]);
        printf("\n");
#endif
}
```

Unikernel entry point after boot

Hello World – Four Required Files (IV)

main.c: source file to provide (at least) a `main()` function

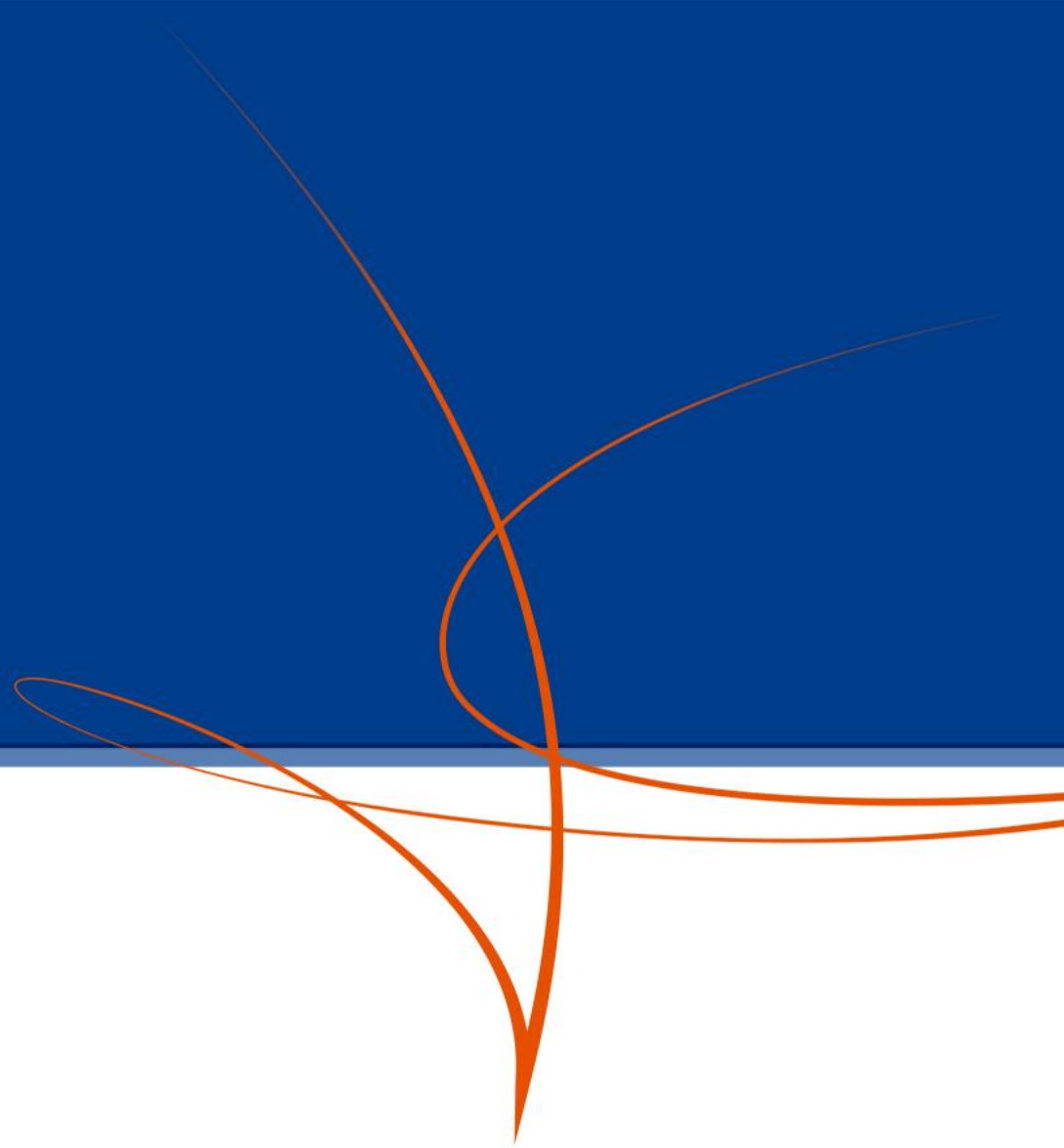
```
#include <stdio.h>
/* Import user configuration: */
#include <uk/config.h>

int main(int argc, char *argv[])
{
        printf("Hello world!\n");
#if CONFIG_APPHELLOWORLD_PRINTARGS
        int i;
        printf("Arguments:s");
        for (i=0; i<argc; ++i)
                printf(" \"%s\"", argv[i]);
        printf("\n");
#endif
}
```

Unikernel entry point after boot

defined by Config.uk

Porting an External Library



How To Port an External Library

How To Port an External Library

- Write **`Makefile.uk`** and add the external library source files to it
  - The library's original Makefile can serve as a template
  - I'll show how to in a second

How To Port an External Library

- Write **`Makefile.uk`** and add the external library source files to it
  - The library's original Makefile can serve as a template
  - I'll show how to in a second

- Write **`Config.uk`** with library-specific options

How To Port an External Library

Write `Makefile.uk` and add the external library source files to it

- The library's original Makefile can serve as a template
- I'll show how to in a second

Write `Config.uk` with library-specific options

- Isn't strictly required
- But at least a simple "library on/off" option for kbuild is a good idea

How To Port an External Library

Write **`Makefile.uk`** and add the external library source files to it

- The library's original Makefile can serve as a template
- I'll show how to in a second

Write **`Config.uk`** with library-specific options

- Isn't strictly required
- But at least a simple "library on/off" option for kbuild is a good idea

Sometimes a bit of *glue* code is needed

- E.g., in newlib to link POSIX thread creation to Unikraft's thread library

How To Port an External Library

- Write **`Makefile.uk`** and add the external library source files to it
  - The library's original Makefile can serve as a template
  - I'll show how to in a second
- Write **`Config.uk`** with library-specific options
  - Isn't strictly required
  - But at least a simple "library on/off" option for kbuild is a good idea
- Sometimes a bit of *glue* code is needed
  - E.g., in newlib to link POSIX thread creation to Unikraft's thread library
- In this early phase: implement core unikraft functionality that is required to support your library, for example:
  - File descriptors/sockets
  - Threading support

How To Port an External Library – Makefile.uk

```
###############################################################################
# Library registration
###############################################################################
$(eval $(call addlib_s,libaxtls,$(LIBAXTLS)))
```

How To Port an External Library – Makefile.uk

```
##############################################################################
# Library registration
##############################################################################
$(eval $(call addlib_s,libaxtls,$(LIBAXTLS)))
```

← **Register library with unikraft build system**

How To Port an External Library – Makefile.uk

```
##########################################################################
# Library registration
##########################################################################
$(eval $(call addlib_s,libaxtls,$(LIBAXTLS)))

##########################################################################
# Source Download
##########################################################################
# Nothing here: sources are small and included directly in the uk library repo
```

← **Register library with unikraft build system**

How To Port an External Library – Makefile.uk

```
##########################################################################
# Library registration
##########################################################################
$(eval $(call addlib_s,libaxtls,$(LIBAXTLS)))

##########################################################################
# Source Download
##########################################################################
$(eval $(call fetch,libaxtls,$(LIBAXTLS_URL)))
$(eval $(call patch,libaxtls,$(LIBAXTLS_PATCHDIR),newlib-$(LIBAXTLS_VERSION)))
```

← **Register library with unikraft build system**

← **Download and patch library code**

How To Port an External Library – Makefile.uk

```
##############################################################################
# Library registration
##############################################################################
$(eval $(call addlib_s,libaxtls,$(LIBAXTLS)))

##############################################################################
# Source Download
##############################################################################
# Nothing here: sources are small and included directly in the uk library repo
$(eval $(call fetch,libaxtls,$(LIBAXTLS_URL)))
$(eval $(call patch,libaxtls,$(LIBAXTLS_PATCHDIR),newlib-$(LIBAXTLS_VERSION)))
##############################################################################
# Library includes
##############################################################################
CINCLUDES-y +=    -I$(LIBAXTLS_BASE)/include \
                  -I$(LIBAXTLS_BASE)/crypto  \
                  -I$(LIBAXTLS_BASE)/ssl
```

← **Register library with unikraft build system**

← **Download and patch library code**

← **The library's original include directories**

How To Port an External Library – Makefile.uk

```
##########################################################################
# Library registration
##########################################################################
$(eval $(call addlib_s,libaxtls,$(LIBAXTLS)))

##########################################################################
# Source Download
##########################################################################
# Nothing here: sources are small and included directly in the uk library repo
$(eval $(call fetch,libaxtls,$(LIBAXTLS_URL)))
$(eval $(call patch,libaxtls,$(LIBAXTLS_PATCHDIR),newlib-$(LIBAXTLS_VERSION)))
##########################################################################
# Library includes
##########################################################################
CINCLUDES-y +=    -I$(LIBAXTLS_BASE)/include \
                  -I$(LIBAXTLS_BASE)/crypto  \
                  -I$(LIBAXTLS_BASE)/ssl

##########################################################################
# sources
##########################################################################
LIBAXTLS_SRCS-y += $(LIBAXTLS_BASE)/crypto/aes.c
LIBAXTLS_SRCS-y += $(LIBAXTLS_BASE)/crypto/bigint.c
…
LIBAXTLS_SRCS-y += $(LIBAXTLS_BASE)/crypto/sha512.c
```

← **Register library with unikraft build system**

← **Download and patch library code**

← **The library's original include directories**

← **The library's original source files**

How To Port an External Library: Outlook

- This can be a lot of busywork
- What about special cases?
  - Special build systems
  - Additional steps other than compiling/linking
  - Preprocessing/dependencies?

```
################################################################################
# Library registration
################################################################################
$(eval $(call addlib_s,libaxtls,$(LIBAXTLS)))

################################################################################
# Source Download
################################################################################
# Nothing here: sources are small and included directly in the uk library repo



################################################################################
# Library includes
################################################################################
CINCLUDES-y +=    -I$(LIBAXTLS_BASE)/include \
                  -I$(LIBAXTLS_BASE)/crypto  \
                  -I$(LIBAXTLS_BASE)/ssl

################################################################################
# sources
################################################################################
LIBAXTLS_SRCS-y += $(LIBAXTLS_BASE)/crypto/aes.c
LIBAXTLS_SRCS-y += $(LIBAXTLS_BASE)/crypto/bigint.c
…
LIBAXTLS_SRCS-y += $(LIBAXTLS_BASE)/crypto/sha512.c
```

How To Port an External Library: Outlook

- This can be a lot of busywork
- What about special cases?
  - Special build systems
  - Additional steps other than compiling/linking
  - Preprocessing/dependencies?
- Coming soon:
  Build passthrough mode
  - Currently under review
  - Vastly simplifies Makefile creation work
  - Especially for large or complicated libraries

```
################################################################################
# Library registration
################################################################################
$(eval $(call addlib_s,libaxtls,$(LIBAXTLS)))

################################################################################
# Source Download
################################################################################
# Nothing here: sources are small and included directly in the uk library repo



################################################################################
# Library includes
################################################################################
CINCLUDES-y +=    -I$(LIBAXTLS_BASE)/include \
                  -I$(LIBAXTLS_BASE)/crypto  \
                  -I$(LIBAXTLS_BASE)/ssl
```

```
################################################################################
# build
################################################################################
UK_ARS-y += $(LIBAXTLS_ORIGIN)/$(LIBAXTLS_DIR)/build/libaxtls.o
LIBAXTILS/.prepared:
            $(call verbose_cmd,CONFIGURE,libaxtls: $@,\
            mkdir -p $(LIBAXTLS_ORIGIN)/$(LIBAXTLS_DIR) && \
            ./configure && make)
```

Unikraft 0.2 Titan

Current Status

Available Libraries

Core Libraries

- libfdt
  - Flat device tree parser
- libnolibc
  - A tiny libc replacement
- libukalloc
  - Memory allocator abstraction
- libukallocbbuddy
  - Binary buddy allocator
- libukargparse
  - Argument parser library
- libukboot
  - Unikraft bootstrapping
- libukdebug
  - Debug and kernel printing
  - Assertions, hexdump
- libuksched
  - Scheduler abstraction
- libukschedcoop
  - Cooperative scheduler
- libukbus
  - abstraction for device buses, e.g., PCI
- libuklock
  - mutexes and semaphores
- libukmpi
  - message-passing interface
- libuknetdev
  - network device support
- libukswrand
  - pseudo-RNG interface
- libuktimeconv
  - time calculation/conversion
- libvfscore
  - basic file descriptor management / mapping / handling

External Libraries

- libnewlib
  - libc originally aimed at embedded devices
- liblwip
  - lightweight TCP/IP stack

Architecture Libraries

- libarmmath
  - 64bit arithmetic on ARMv7
- libx86ctx
  - Extended register support for x86 ctx switch

Platform Libraries

- libxenplat
  - Xen (PV)
  - x86_64, ARMv7
- libkvmplat
  - QEMU/kvm
  - x86_64, ARM64, virtio-net support
- liblinuxu
  - Linux userspace
  - x86_64, ARMv7

Current work: coming soon (in the pipeline) or being ported

Core Libraries

- libukschedpreempt
  - Pre-emptive scheduler

External Libraries

- libclick
  - Click modular router (e.g., for NFV)
- libaxtls
  - TLS support aimed at embedded devices
- libstdc++
- libmicropython
  - Python implemented for microcontrollers

Architecture Libraries

- libarmctx
  - Extended register support for Arm ctx switch

Platform Libraries

- libxenplat
  - ARM64 support
  - netfront support
- liblinuxu
  - tap device based networking support

The road ahead

- First public alpha release (without much functionality) in December
- Released as a Xen incubator project
- Initially, mostly internal contributors from NEC Labs

- Currently external contributors from
  - Romania (netfront, scheduling; from University Politehnica Bucharest)
  - Israel (bare-metal support)
  - China (ARM64 support; from ARM)

- We welcome additional contributors!

- Resources:
  - Code: https://xenbits.xen.org/gitweb/?pf=unikraft (make sure you check out staging!)
  - On-line documentation: unikraft.org
  - IRC: #unikraft @ freenode
  - Mailing list: minios-devel@lists.xen.org

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

EUROPE

THE LINUX FOUNDATION