NEC

Unikernels Made Easy

Simon Kuenzer <simon.kuenzer@neclab.eu>

Senior Researcher, NEC Laboratories Europe GmbH

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.

usenix LISA18

October 29–31, 2018 | Nashville, TN, USA
www.usenix.org/lisa18 #lisa18

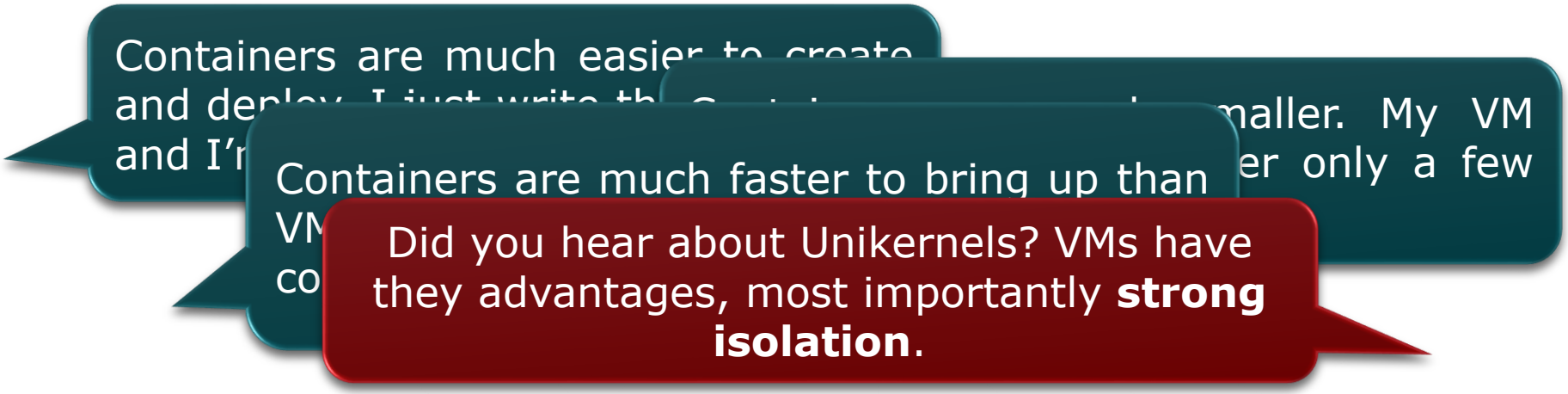
Orchestrating a brighter world

NEC brings together and integrates technology and expertise to create the ICT-enabled society of tomorrow.
We collaborate closely with partners and customers around the world, orchestrating each project to ensure all its parts are fine-tuned to local needs.

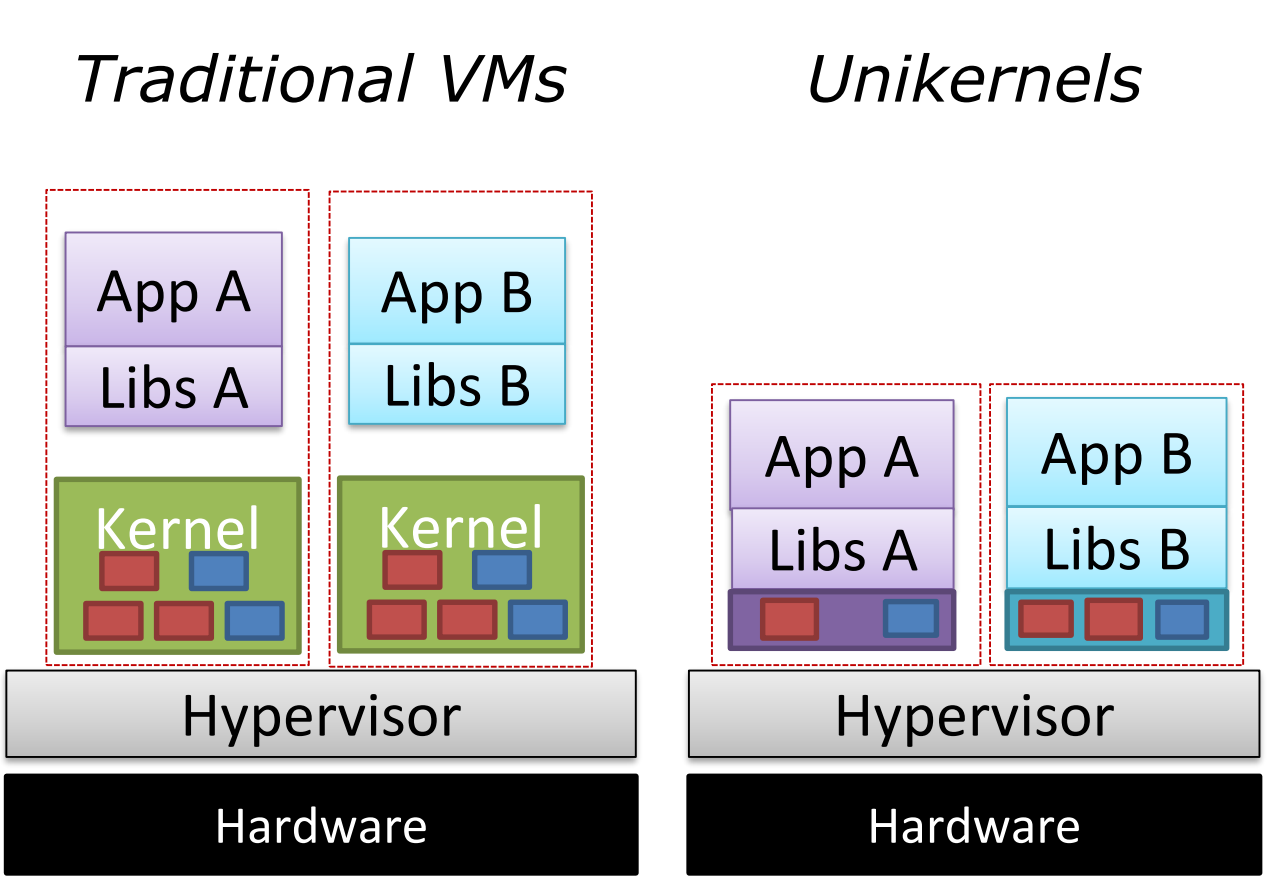
Every day, our innovative solutions for society contribute to greater safety, security, efficiency and equality, and enable people to live brighter lives.

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?



Unikernels as VMs



- Unikernels are purpose-built
  - Thin kernel layer, *only what application needs*
  - Single *monolithic* binary containing OS and application
- No isolation within Unikernel needed, done with hypervisor
  - One application → Flat and single address space
- Further advantages from specialization

Unikernel Advantages

- Fast instantiation, destruction and migration time
  - 10s of milliseconds

- Low memory footprint
  - Few MB of RAM

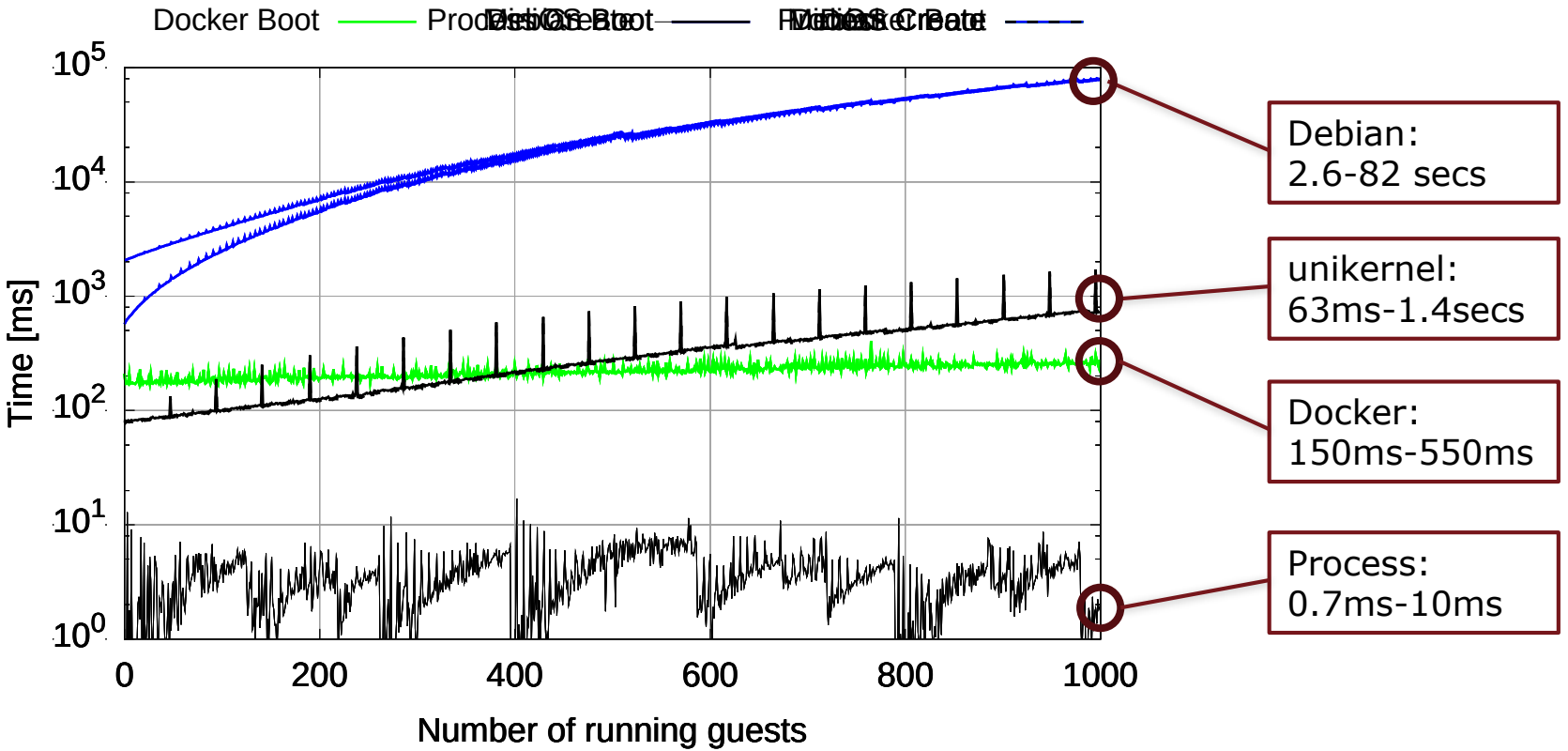
- High density
  - 10k guests on a single server node

- High Performance
  - 10-40Gbit/s throughput
  - 5-6x more req/s than standard nginx

- Reduced attack surface
  - Less components exist in Unikernel
  - Strong isolation by hypervisor

LightVM [Manco SOSP 2017], Elastic CDNs [Kuenzer VEE 2017], Superfluid Cloud [Manco HotCloud 2015] , ClickOS [Martins NSDI 2014]

In Numbers: Instantiation Times



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

Application Domains

Minimal SW Stack

Reactive vNFs,
Serverless,
…

Fast boot, migration, destroy

Resource efficient

Minimal SW Stack

Serverless,
(Per-customer) vNFs,
IoT,
MEC,
…

Specialization

NFV,
MEC,
…

High performance

Mission critical

Small code base
→ Low attack surface
→ Cheaper verification

Automotive,
IoT,
…

The Devil is in the Details

So, Unikernels:

- Give similar speed and size of containers
- But add **strong isolation** with *virtualization* and increase **security** due to *smaller code base*

The problem is *Unikernel development:* Optimized Unikernels are manually built

- Building takes several months or even longer
  - *We've done it before, multiple times*
- Potentially repeat the process for each target application
  - *We've done that too…*

That's not an effective way of doing things!

Unikraft: A Unikernel Framework

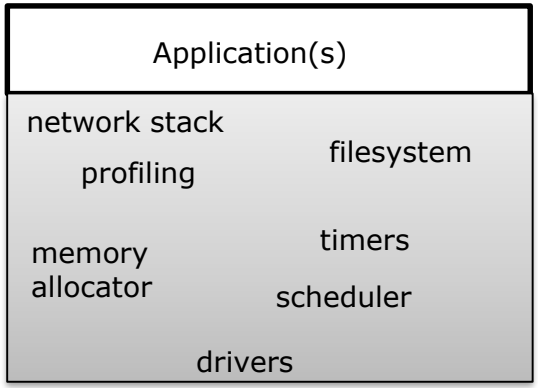
Motivation

- Support wide range of use cases
- Simplify building and optimizing
- Common and shared code base for Unikernel creators
- Support different hypervisors and CPU architectures

Unikraft

- Concept:
  "Everything is a library"
  - Decomposed OS functionality
- Unikraft's two components:
  - Library Pool
  - Build Tool

The Unikraft Way: Everything is a library



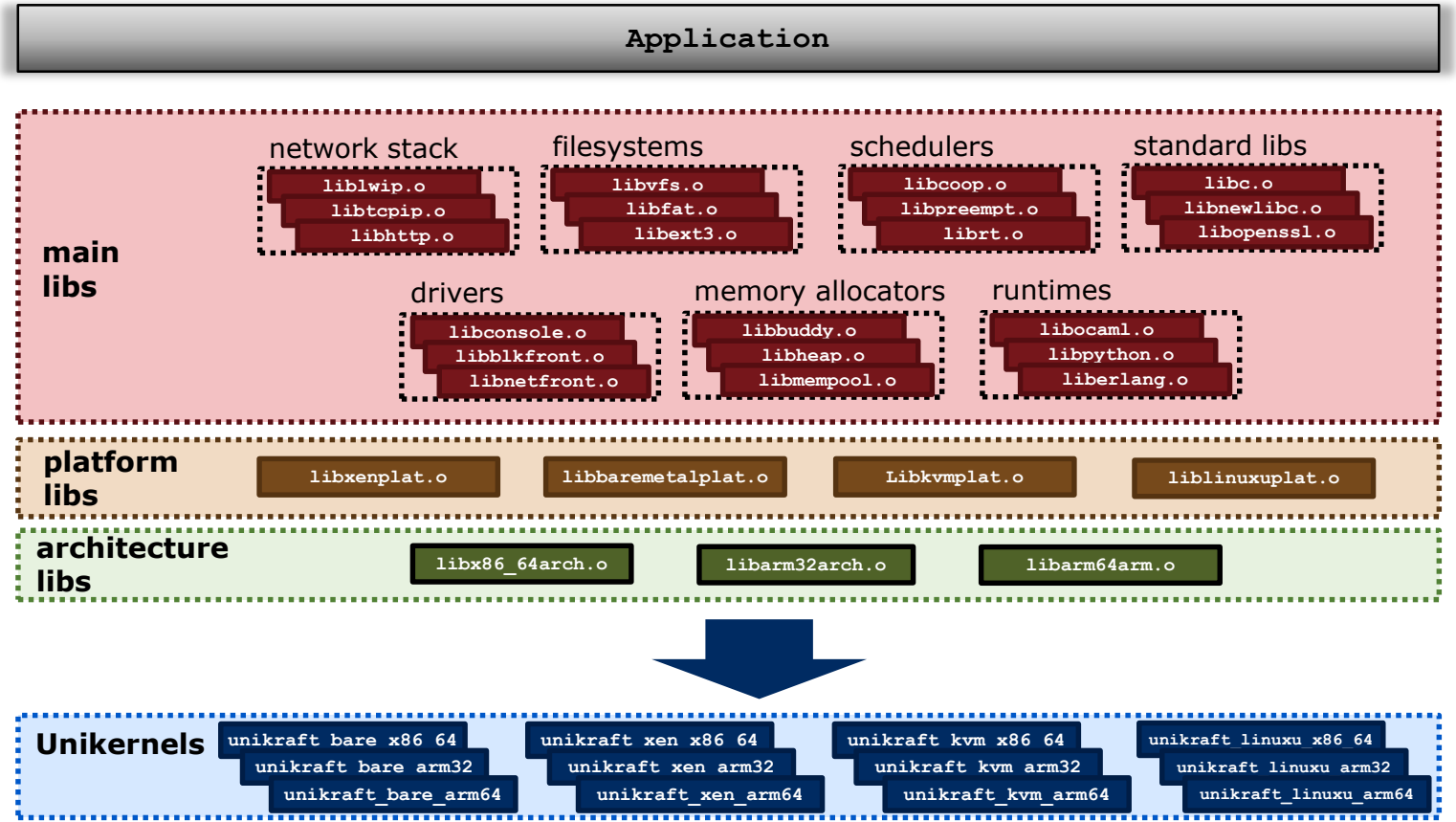
The Unikraft Way: Everything is a library

Decompose OS into a set of libraries

Unikraft

Overview

Unikraft Component 1: Library Pool



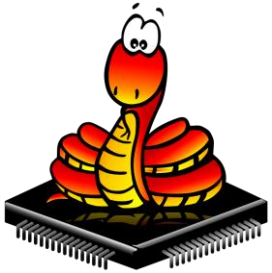
1. **Select/Create Application**
2. **Select and Configure libraries**
3. **Build**
4. **Run**

Example Library Selection

Micropython Unikernel for KVM on x86_64

My Python App	libmicropython.o
liblwip.o	libvfscore.o
libschedcoop.o	liballocbbuddy.o
libkvmplat.o	libx86_64arch.o

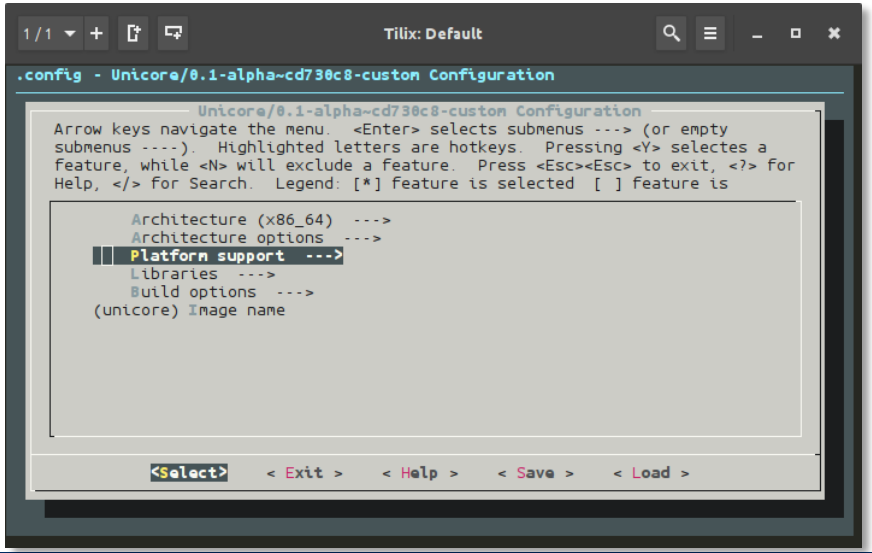
=



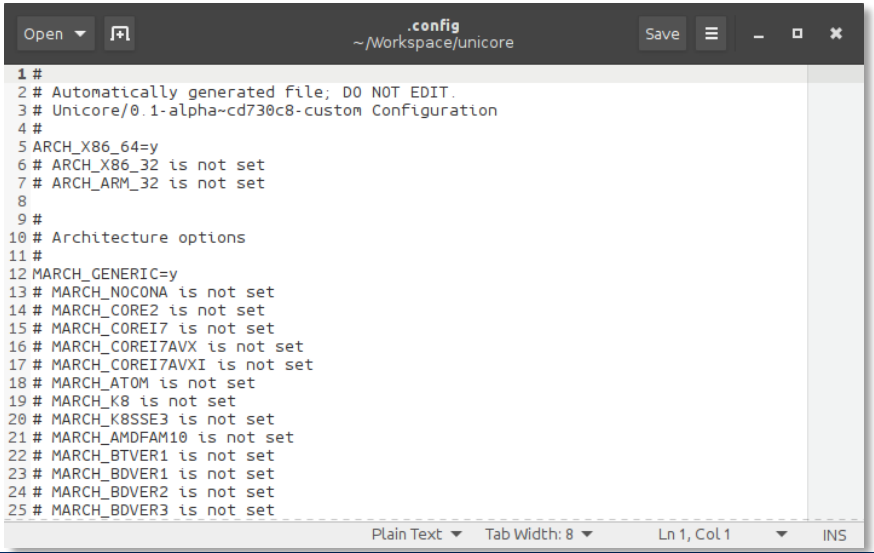
Unikernel

Unikraft Component 2: Build Tool

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



.config



Unikraft

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
  - Scheduling/context switch support for x86

Platform Libraries

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

Current work in the pipeline: Upstream soon

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

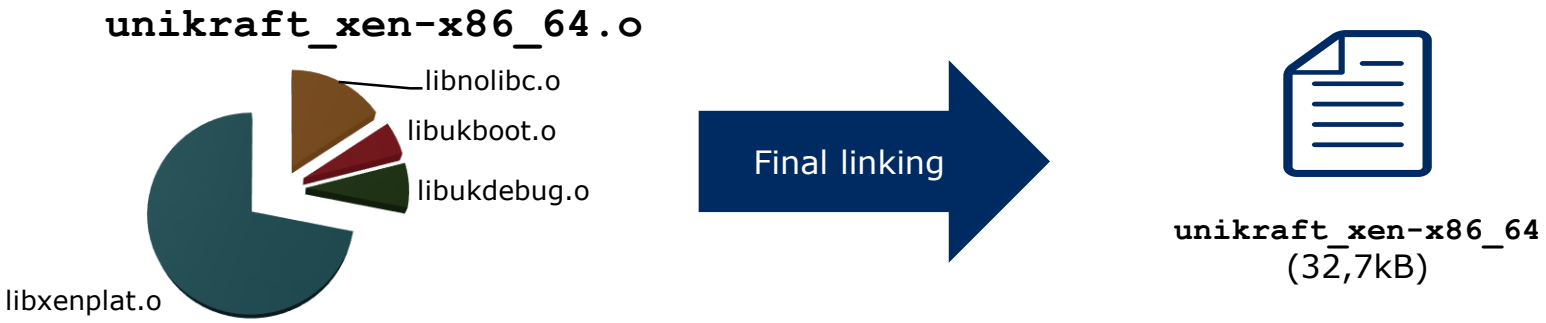
- libarmctx
  - Scheduling/context switch support for Arm

Platform Libraries

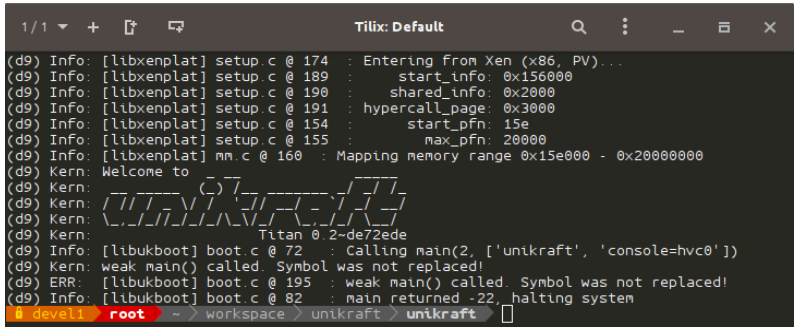
- libxenplat
  - Arm64 support
  - netfront support
- liblinuxuplat
  - tap device based networking support

A Baseline Example…

- *Xen PV x86_64 binary*



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



- More functional example: VNF Unikernel Click: 4.5 MB (8 MB RAM)

Unikraft

It is Open Source!

Join us!

- Unikraft is OpenSource since Dec 2017 and under the umbrella of

- Community is growing! External contributors from
  - Romania (networking, scheduling; from University Politehnica Bucharest)
  - Israel (bare-metal support, VGA driver)
  - China (Arm64 support; from Arm)

There is still a lot to do! Get in touch with us!

Drop us a mail
minios-devel@lists.xen.org
Join our IRC channel
#unikraft on Freenode

Resources

- Wiki
  - https://wiki.xenproject.org/ (Search for Unikraft)
- Dokumentation
  - http://www.unikraft.org
- Sources (GIT)
  - http://xenbits.xen.org/gitweb/ (Namespace: Unikraft)
- Mailing list (shared with Mini-OS)
  - minios-devel@lists.xen.org
- IRC Channel on Freenode
  - #unikraft
- NEC-Team
  - http://sysml.neclab.eu


Orchestrating a brighter world
NEC

Example

"Hello World" with Unikraft

Repo Structure

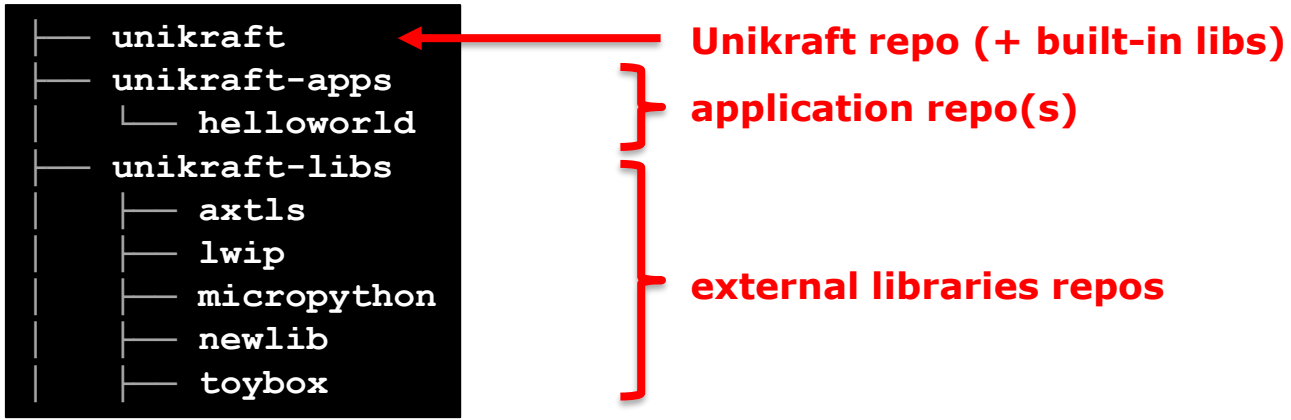
- Clone the main Unikraft repo

```
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” Application

Four files to integrate to Unikraft

- `Makefile` – Entry point for make
- `Makefile.uk` – Describe build for Unikraft
- `Config.uk` – Dependencies and configuration options
- `main.c` – Source code of 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)
```

← path to Unikraft repo

← path to external libs

← external libs needed (colon separated)

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: application source file that provides a `main()` function

```c
#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:");
	for (i=0; i<argc; ++i)
		printf(" \"%s\"", argv[i]);
	printf("\n");
#endif
}
```

Libc functionality is provided by a libc or nolibc (dependency in Config.uk)

Unikernel entry point after boot

defined by Config.uk