Hack Lyon'22 Tutorial, 14th May 2022

Getting started with Unikraft

Gaulthier Gain <gaulthier.gain@uliege.be>

Organization

Part 1: Introduction and general overview of a specific session (+demo):

- 15/20 minutes

Part 2: Online tutorial where you practice:

- **If you are in ENS**: Hugo, Pierre-Olivier and I are here to help you*.
- **If you are on Discord**: Online help by Razvan, Alex, and the Unikraft task force.

*You can also post your questions on Discord

Agenda

Time	Presentation	Presenter
9:00 – 9:15	High-level presentation of unikernels and Unikraft	Pierre-Olivier (UoM)
Now! – 10:30	Getting started with Unikraft	Gaulthier (ULiege)
10:30 – 11:45	Inside Unikraft: Building, configuring, using different libraries	Hugo (UoM)
11:45 – 13:00	Debugging in Unikraft	Hugo (UoM)
13:00 – 14:00	Lunch 🥖	
14:00 – 15:15	Running complex applications	Gaulthier (ULiege)
15:15 – 17:00	FlexOS	Hugo (UoM)

Tutorial Material

TODO

Online Attendees

For live help & support by the open-source community

https://bit.ly/UnikraftDiscord

Unikraft

Unikraft is a fast, secure and open-source Unikernel Development Kit

https://github.com/unikraft/unikraft

The Unikraft Community

~50	**40+**	**12**
Active Contributors	Talks & Presentations	Publications
10	**6**	**4**
Institutes & Companies	Countries	Years since first commit

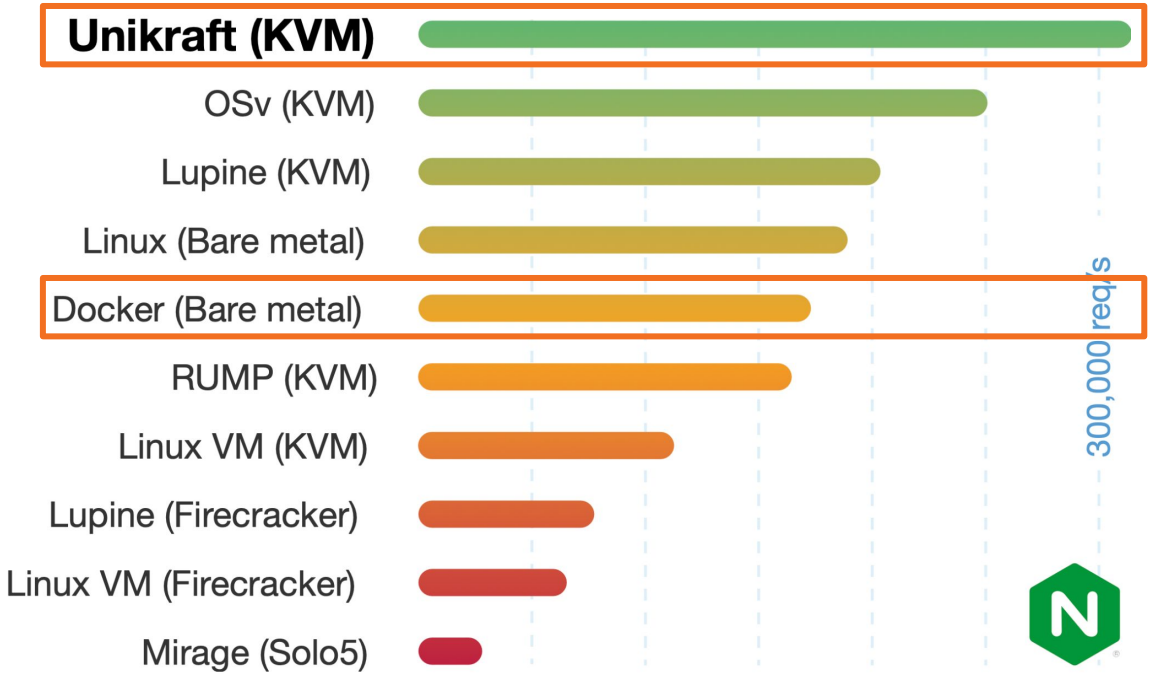
Supported “Native” Applications






Why to choose Unikraft?

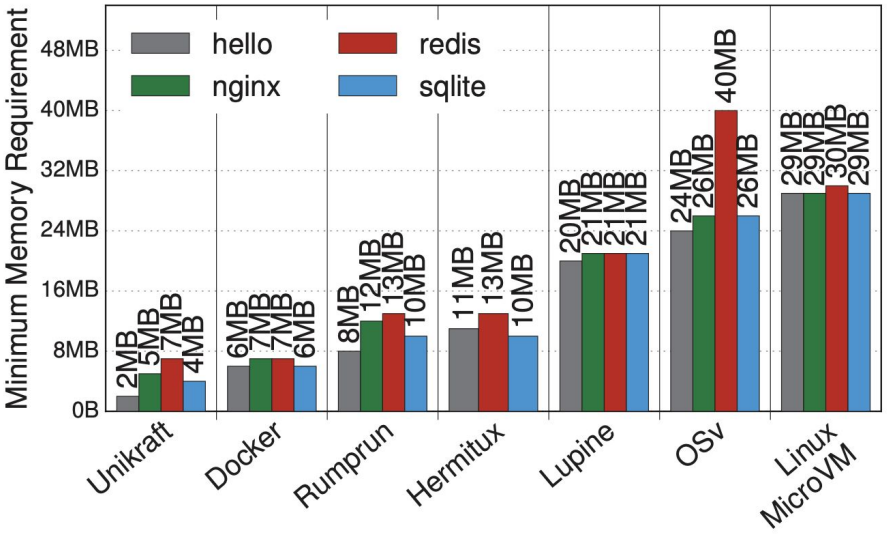
Unikraft provides better performance



On Unikraft, **82%** of increase compared to Docker (Nginx throughput)

Unikraft provides better memory consumption and storage

Memory Usage



Disk Space

Image	Size
docker.io/nginx:1.15.6	42.62 MB
unikraft.io/nginx:1.15.6	1.3 MB

And many more…

See our full paper: https://dl.acm.org/doi/10.1145/3447786.3456248

Artifacts: https://github.com/unikraft/eurosys21-artifacts

The Unikraft Model

Unikraft relies on two main components:

The Unikraft Model

Unikraft relies on two main components:

1. A Core Build System

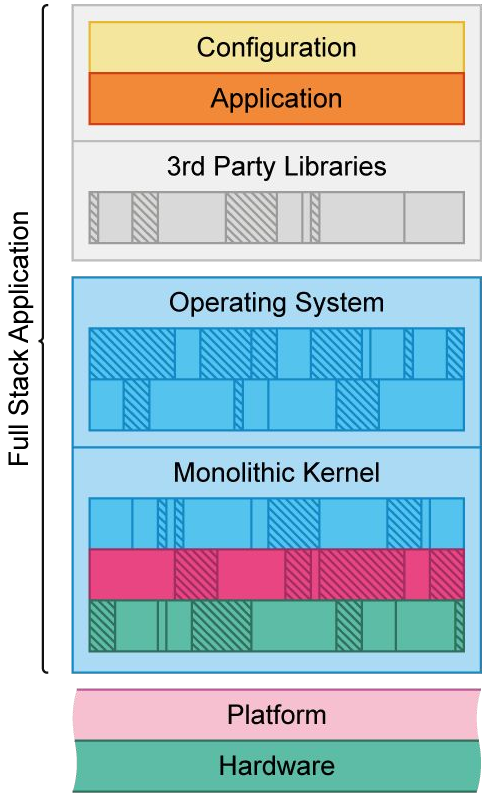
The Unikraft Model

Unikraft relies on two main components:

1. A Core Build System

2. A set of libraries:
“Everything is a library”

The Unikraft Model

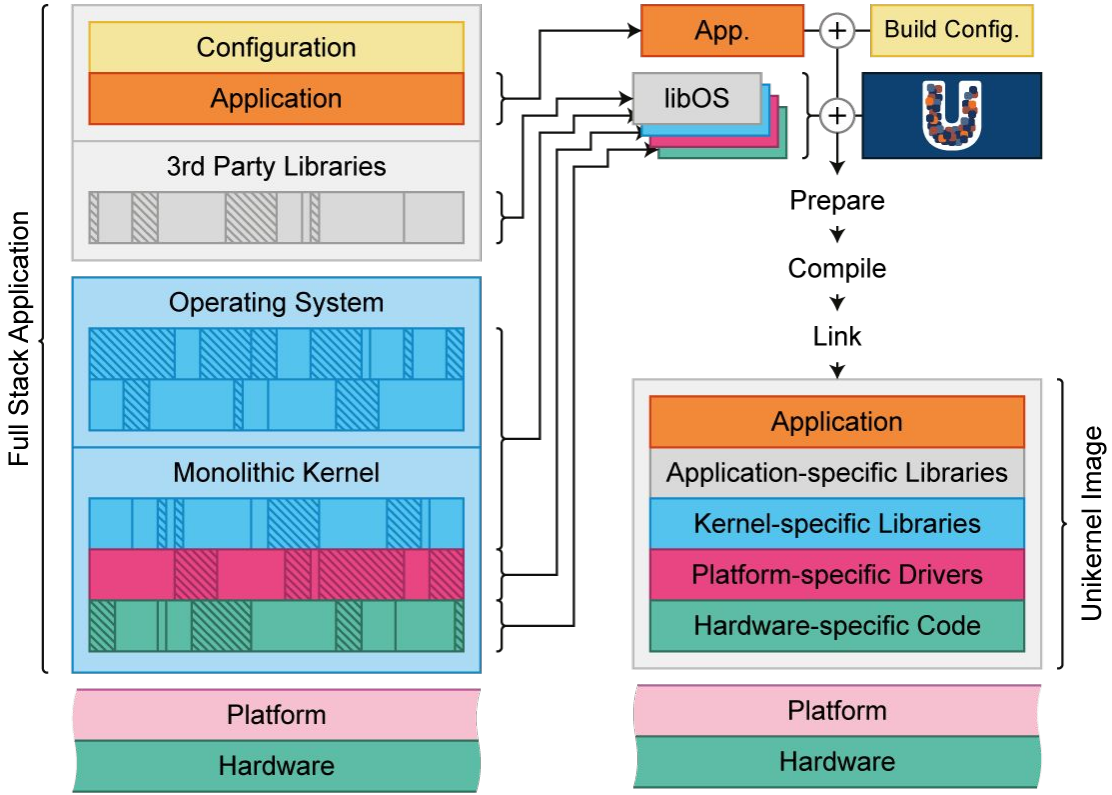


Considering a traditional OS. An application requires:

- external libraries (user space);
- some resources and services from the OS/kernel (drivers, system calls, …)

Contains a lot of useless features (dead-code) and the attack surface is big.

The Unikraft Model



Two Types of Unikraft libraries

A “native” (internal) library

- Unikaft “core” internal libs (e.g., scheduler, memory allocator, …).
- All sources are included with the main unikraft repo.

Two Types of Unikraft libraries

A “native” (internal) library

- Unikaft “core” internal libs (e.g., scheduler, memory allocator, …).
- All sources are included with the main unikraft repo.

A “wrapper” (external) library

- An external library, (e.g., openssl, musl-libc, …).
- Download external sources (from “origin”), patch them (if necessary).

Two Types of Unikraft libraries

A “native” (internal) library

- Unikaft “core” internal libs (e.g., scheduler, memory allocator, …).
- All sources are included with the main unikraft repo.

A “wrapper” (external) library

- An external library, (e.g., openssl, musl-libc, …).
- Download external sources (from “origin”), patch them (if necessary).

Bonus: “Binary Compatibility” objects

(Covered tomorrow by Razvan)

The Unikraft Model

Unikraft relies on two main components:

1. A Core Build System

2. A set of libraries:
“Everything is a library”

The Unikraft Model

But,

How do build unikernels and manage many libraries?

Two approaches to build Unikernels

Using the companion tool “kraft”:

- Companion tool to easily manage multiple libraries from different sources
- Quickly access updates and changes between versions
- Automatically download application source dependencies

Manually build unikernels:

- The manual approach is more complicated (but it gives you potentially more control)
- This will be discussed in more detail in session *02: Behind the Scenes*.
- You must handle yourself the libraries and the build.

Summary of kraft commands

`$ kraft list update`	Update the manifest
`$ kraft list`	List known libraries, apps, platforms & versions
`$ kraft list add https://github.com/me/lib-repo.git`	Add a repo to the manifest
`$ kraft list pull`	Pull a remote repo to your workspace
`$ kraft fetch`	Fetch the “origin” of a Unikraft wrapper library
`$ kraft menuconfig`	Open the KConfig menuconfig
`$ kraft configure`	Configure the application based on kraft.yaml
`$ kraft build`	Build the unikernel
`$ kraft run`	Run the unikernel
`$ kraft up`	Shortcut for fetch + configure + build + run

Access to build VMs

Pre-installed with all the tools you need!

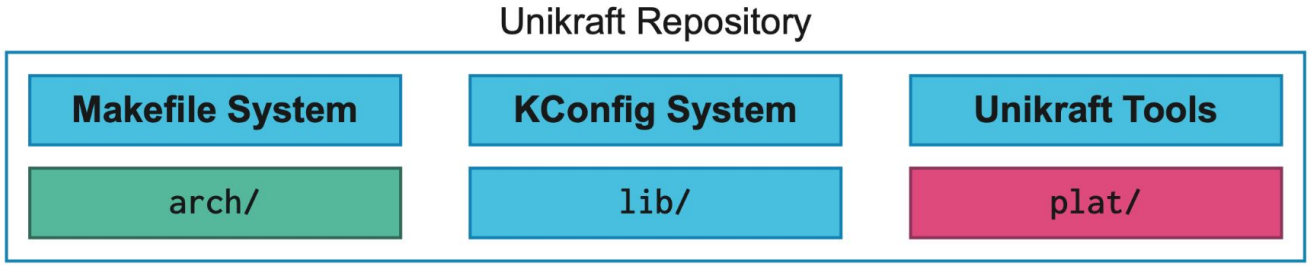
https://guacamole.grid.pub.ro/

Username:	TODO
Password:	TODO

Kindly provided to us/you by University POLITEHNICA Bucharest 🥰

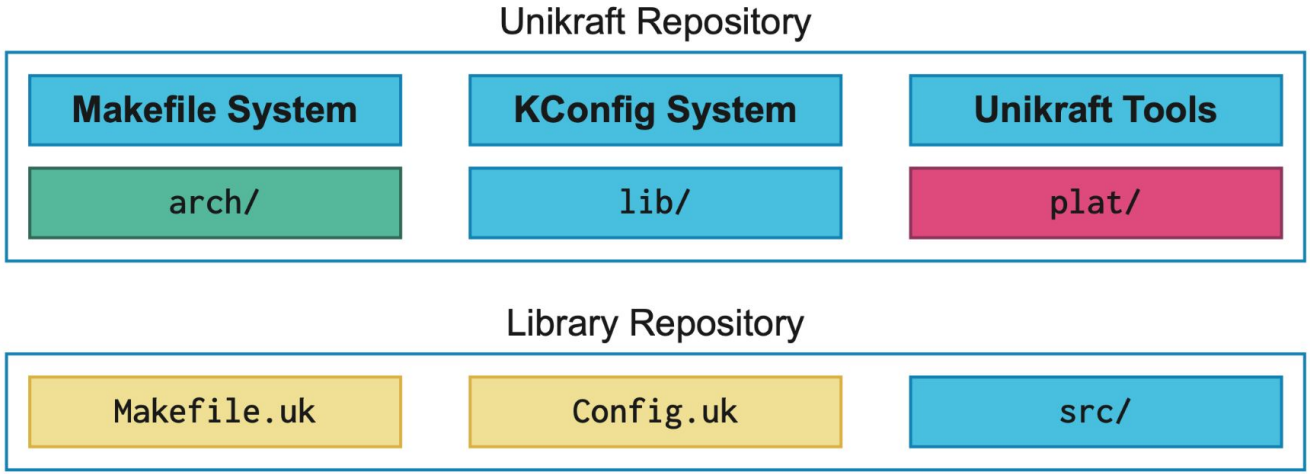
You have now built a Unikraft unikernel

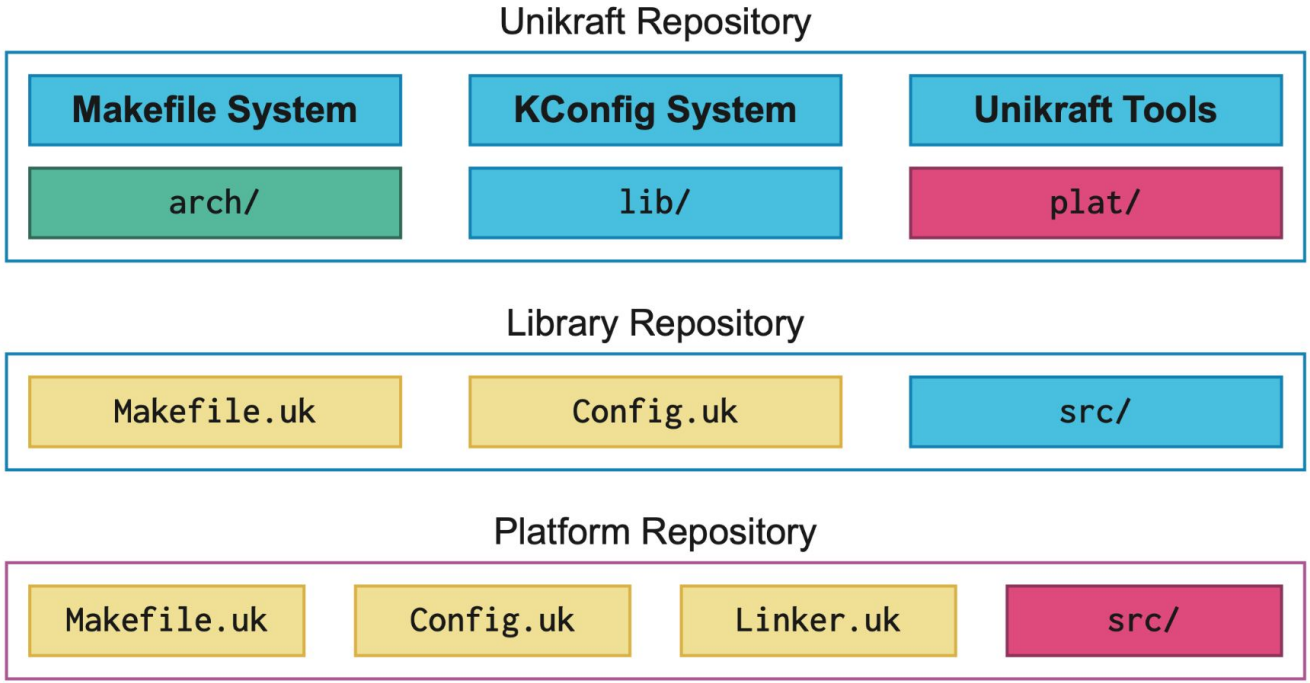
But how does Unikraft work?

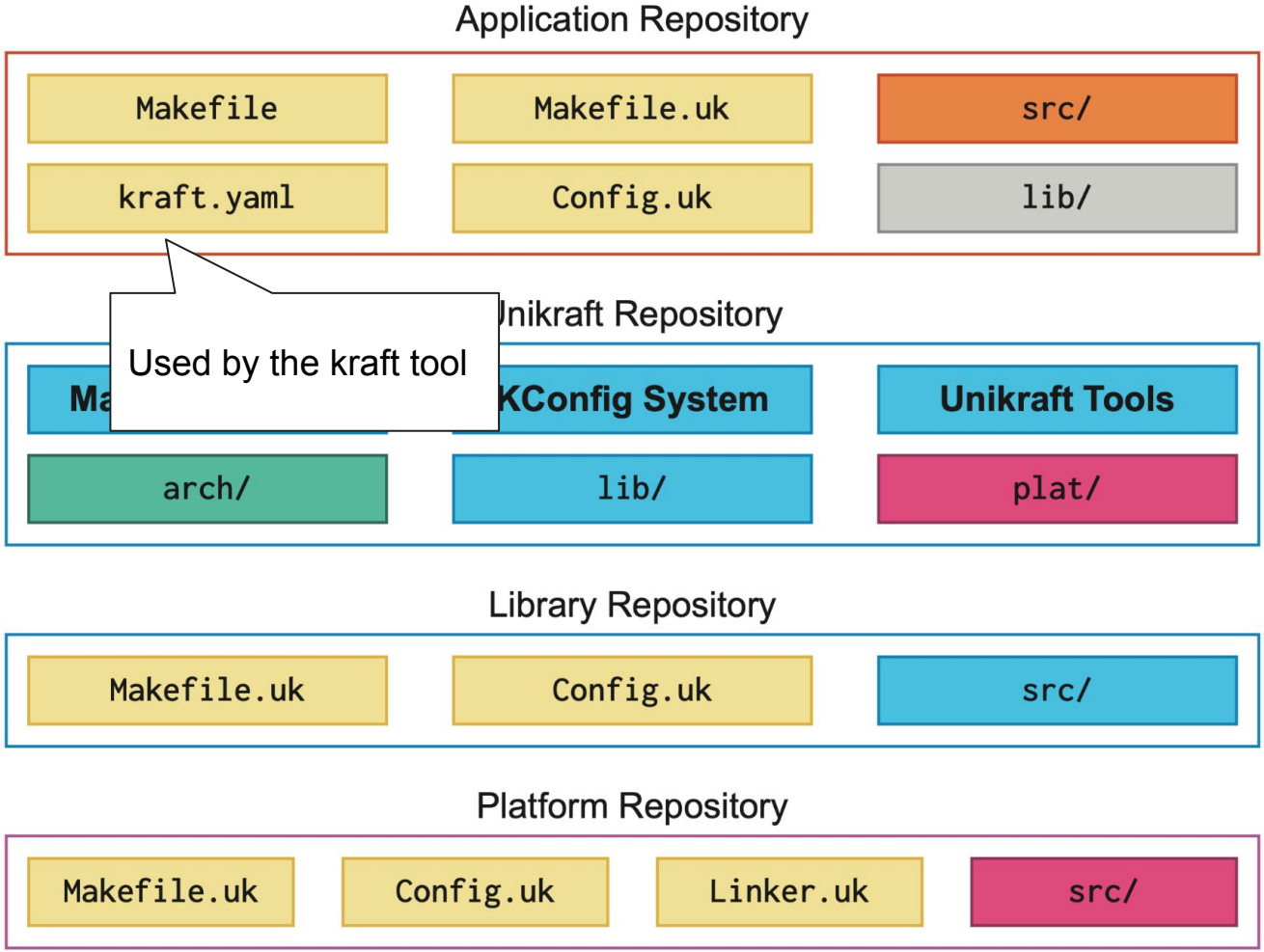

Unikraft Repository
Makefile System
KConfig System
Unikraft Tools
arch/
lib/
plat/

Makefile.uk: contains the rules for the building (e.g., sources, flags, …)

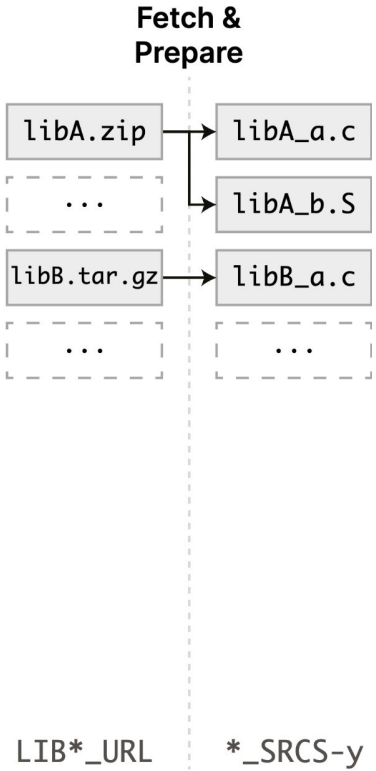
Config.uk: contains the configurations and the dependencies




Unikraft Repository
Makefile System
KConfig System
Unikraft Tools
arch/
lib/
plat/
Library Repository
Makefile.uk
Config.uk
src/
Platform Repository
Makefile.uk
Config.uk
Linker.uk
src/


Application Repository
Makefile
Makefile.uk
src/
kraft.yaml
Config.uk
lib/
Used by the kraft tool
nikraft Repository
Ma
KConfig System
Unikraft Tools
arch/
lib/
plat/
Library Repository
Makefile.uk
Config.uk
src/
Platform Repository
Makefile.uk
Config.uk
Linker.uk
src/

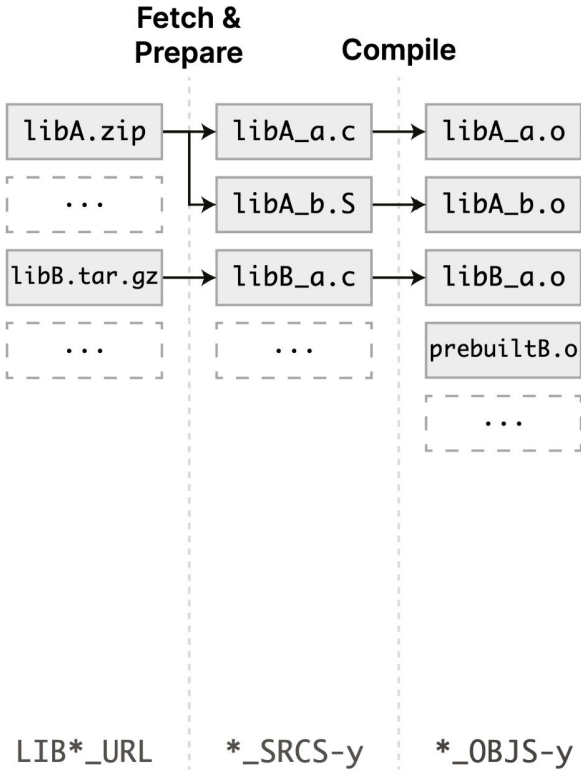
The Unikraft Model



1. **Fetch & prepare:** Download, uncompress, patch, select sources, …

Makefile.uk declarations

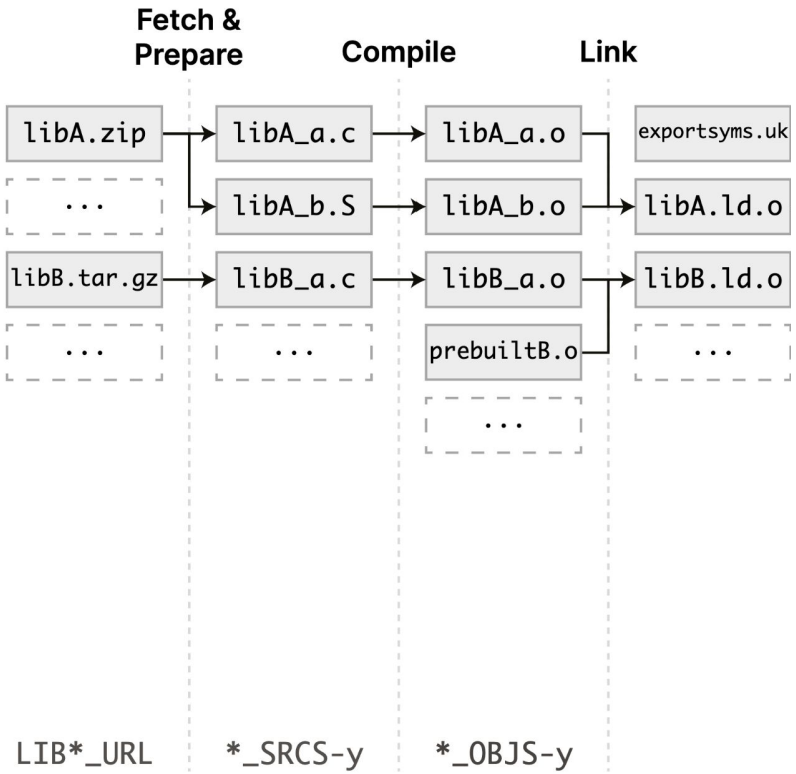
The Unikraft Model



1. **Fetch & prepare:** Download, uncompress, patch, select sources, …
2. **Compile** sources into objects.

Makefile.uk declarations

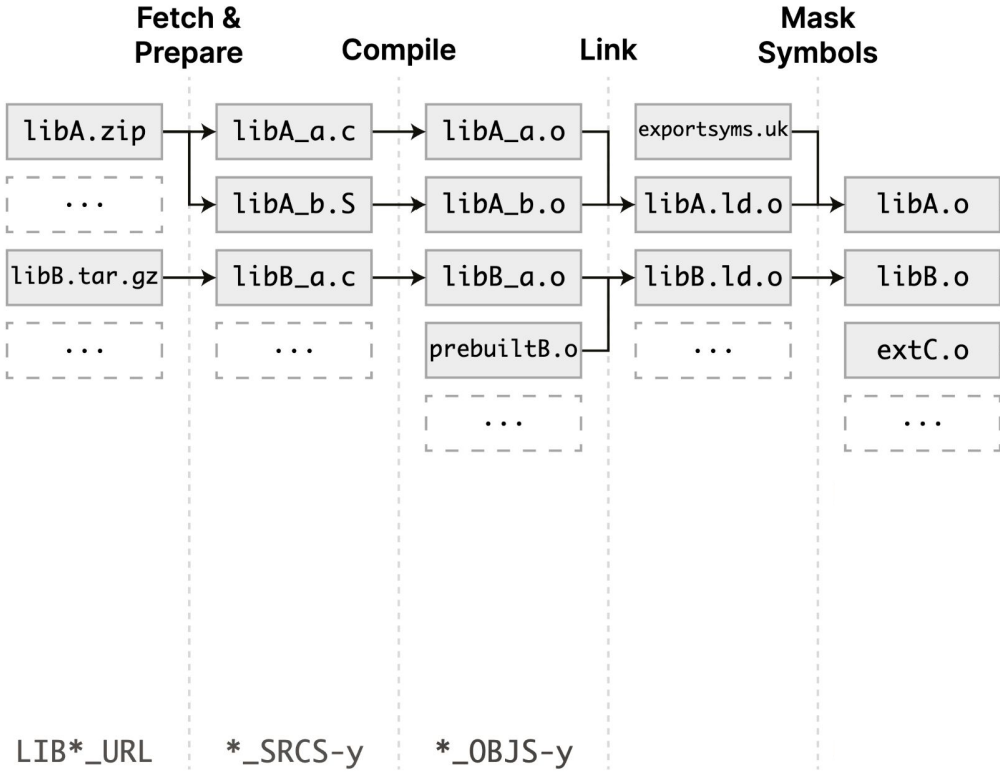
The Unikraft Model



1. **Fetch & prepare:** Download, uncompress, patch, select sources, …
2. **Compile** sources into objects.
3. **Link** objects per library (*ld.o* files)

Makefile.uk declarations

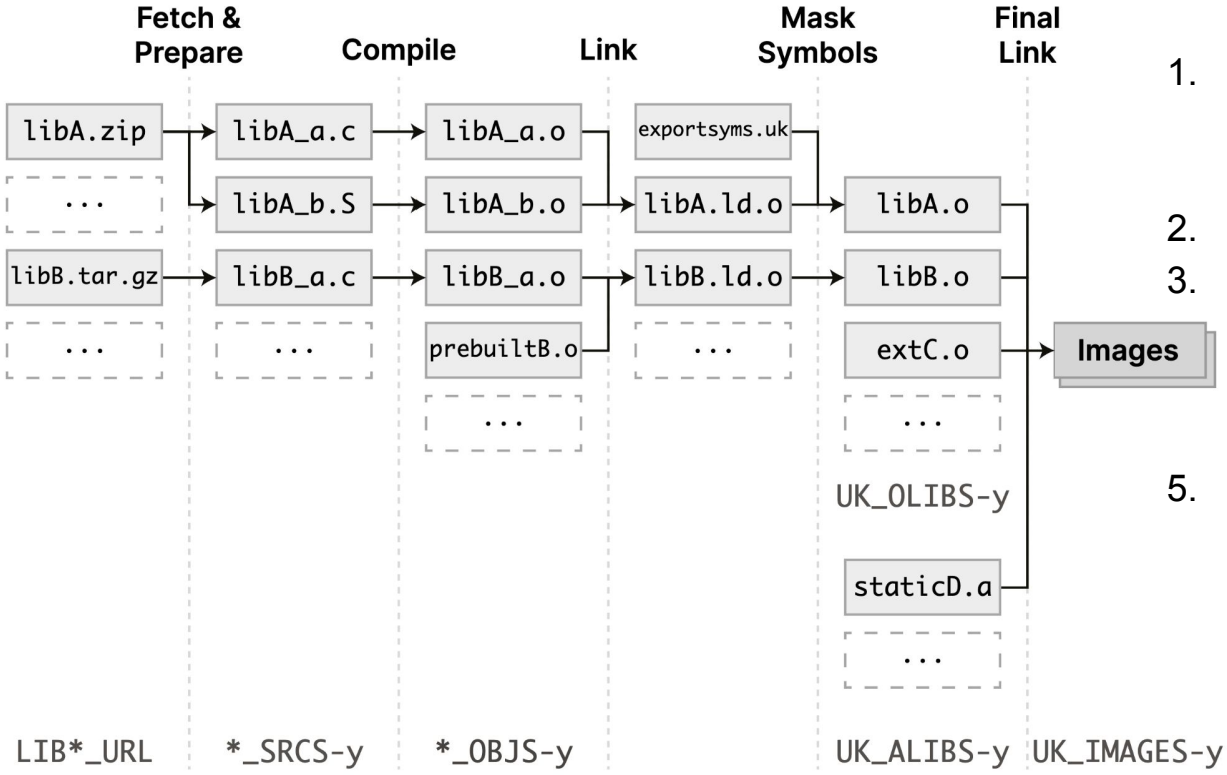
The Unikraft Model



1. **Fetch & prepare:** Download, uncompress, patch, select sources, …
2. **Compile** sources into objects.
3. **Link** objects per library (*ld.o* files)
4. **Mask Symbols** with *exportsyms.uk** file.

* contains the name symbols that should be exported to other libraries

The Unikraft Model



1. **Fetch & prepare:** Download, uncompress, patch, select sources, …
2. **Compile** sources into objects.
3. **Link** objects per library (*ld.o* files)

   Mask Symbols with *exportsyms.uk** file.
5. **Final Linking** of all the objects into an ELF file.
   - External archive(s) or object files can be added.

Makefile.uk declarations

Makefile.uk

```
$(eval $(call addlib_s,LIBMYLIB,$(CONIG_LIBMYLIB)))
```

Register library

```
LIBMYLIB_VERSION=2.1.2
LIBMYLIB_URL=https://releases.mylib.org/v$(LIBMYLIB_VERSION).zip
$(eval $(call fetch,libmylib,$(LIBMYLIB_URL)))
```

Fetch sources (optional)

```
$(LIBMYLIB_BUILD)/.prepared:
    # my preparation steps here
UK_PREPARE-$(CONFIG_LIBMYLIB) += $(LIBMYLIB_BUILD)/.prepared
```

Custom prepare steps (optional)

```
LIBMYLIB_PDIR=$(LIBMYLIB_BASE)/patches
$(eval $(call patch,libmylib,$(LIBMYLIB_PDIR),$(LIBMYLIB_VERSION)))
```

Patch sources (optional)

```
# Include from library directory
LIBMYLIB_CINCLUDES-y += -I$(LIBMYLIB_BASE)/include
# Include from extracted archive
LIBMYLIB_CINCLUDES-y += -I$(LIBMYLIB_ORIGIN)/include
```

Include paths

```
# Include from library directory
LIBMYLIB_SRCS-y += -I$(LIBMYLIB_BASE)/source_a.c
# Include from extracted archive
LIBMYLIB_SRCS-y += -I$(LIBMYLIB_ORIGIN)/source_b.c
```

Include sources to build

```
LIBMYLIB_OBJS-y += $(LIBMYLIB_ORIGIN)/prebuilt.o
UK_ALIBS-$(CONFIG_LIBMYLIB) += $(LIBMYLIB_ORIGIN)/static_lib.a
```

External objects (optional)

https://github.com/unikraft

https://unikraft.org

info@unikraft.io

@UnikraftSDK

Ultimate Performance & Security.