Rust Meetup Zürich

Running Rust on the Rumprun Unikernel

Sebastian Wicki
gandro@rumpkernel.org

Why Rust

```ruby
class ::String
  def blank?
    /\A[[:space:]]*\z/ == self
  end
end
```

Ruby

Ruby:
964K iter/sec

https://www.youtube.com/watch?v=_wy4tuFEpz0 - Niko Matsakis: Why Rust?

Why Rust

```c
static VALUE
rb_str_blank_as(VALUE str)
{
  rb_encoding *enc;
  char *s, *e;

  enc = STR_ENC_GET(str);
  s = RSTRING_PTR(str);
  if (!s || RSTRING_LEN(str) == 0) return Qtrue;

  e = RSTRING_END(str);
  while (s < e) {
    int n;
    unsigned int cc = rb_enc_codepoint_len(s, e, &n, enc);

    switch (cc) {
      case 9:
      case 0xa:
      case 0xb:
      case 0xc:
      case 0xd:
      case 0x20:
      case 0x85:
      case 0xa0:
      case 0x1680:
      case 0x2000:
      case 0x2001:
      case 0x2002:
      case 0x2003:
      case 0x2004:
      case 0x2005:
      case 0x2006:
      case 0x2007:
      case 0x2008:
      case 0x2009:
      case 0x200a:
      case 0x2028:
      case 0x2029:
      case 0x202f:
      case 0x205f:
      case 0x3000:
#if ruby_version_before_2_2()
      case 0x180e:
#endif
          /* found */
          break;
      default:
          return Qfalse;
    }
    s += n;
  }
  return Qtrue;
}
```

C

github.com/SamSaffron/fast_blank

Ruby:
964K iter/sec

C:
10.5M iter/sec

Why Rust

```ruby
class ::String
  def blank?
    /\A[[:space:]]*\z/ == self
  end
end
```

Ruby

```rust
extern “C” fn blank(buf: Buf) -> bool {
  buf.as_slice()   // string slice
     .chars()      // unicode iterator
     .all(|c| c.is_whitespace())
}
```

Rust

Ruby:
964K iter/sec

C:
10.5M iter/sec

Rust:
11M iter/sec

Why Rust

```
fn load_images(paths: &[PathBuf]) -> Vec<Image> {
  paths.iter()
       .map(|path| {
         Image::load(path)
       })
       .collect()
}
```

Why Rust

```
extern crate rayon;

fn load_images(paths: &[PathBuf]) -> Vec<Image> {
  paths.par_iter()
       .map(|path| {
         Image::load(path)
       })
       .collect()
}
```

Why Rust

```
extern crate rayon;

fn load_images(paths: &[PathBuf]) -> Vec<Image> {
  let mut jpegs = 0;
  paths.par_iter()
       .map(|path| {
         if path.ends_with(“.jpg”) { jpegs += 1; }
         Image::load(path)
       })
       .collect()
}
```

The Rust compiler will statically prevent this data race!

hacking

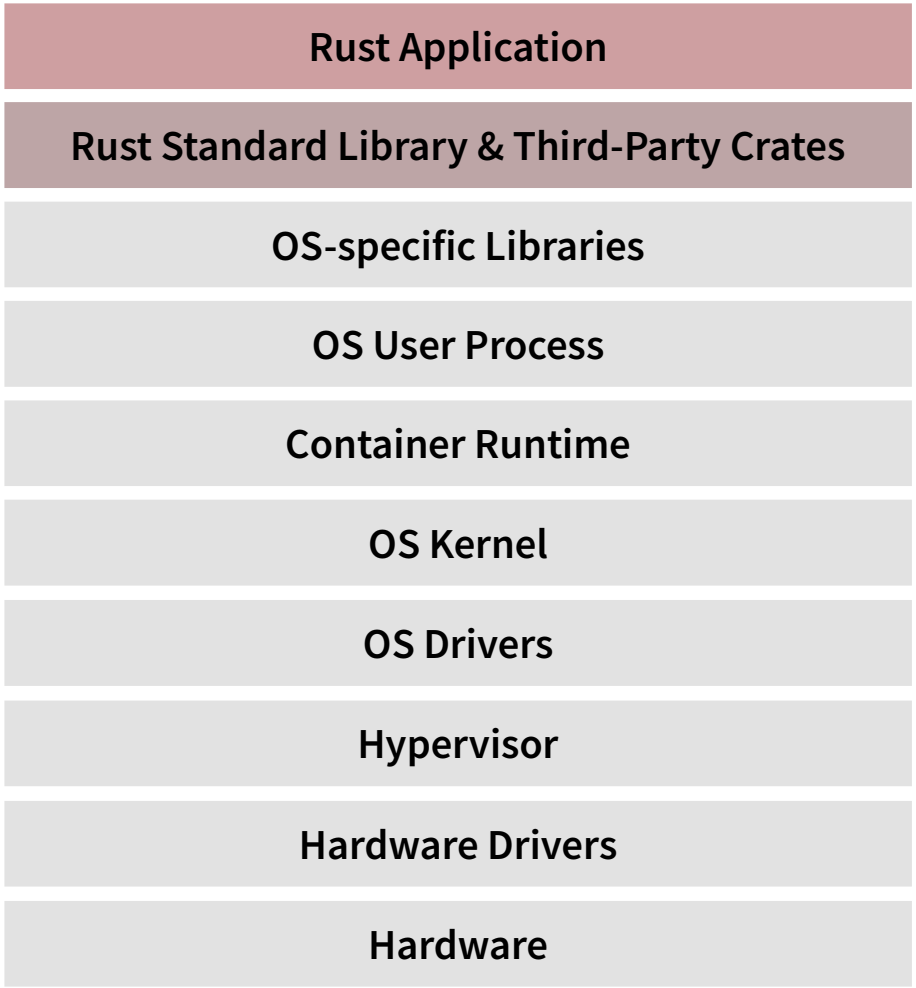
without fear

Why Rust

- a safe, high-level language ...
  - memory safety
  - thread safety
  - strong, static type system
- ... with minimal run-time costs
  - no garbage collector
  - no threading run-time
  - explicit boxing
  - explicit dynamic dispatch
- don't pay what you don't use
  - small standard library
  - extendable through libraries
- cargo
  - build system & package manager
  - dependency management
    - based on semantic versioning
    - static linking
  - repository at crates.io

Running a Rust application on a single server

- not so close to the metal anymore
  - definitely not a zero-cost abstraction
- redundancy
  - e.g. resource management, isolation
- complexity
  - many applications & drivers involved
  - large attack surface



Rust on **bare-metal**

- `#[no_std]`
  - stdlib is optional, like in C
  - great for embedded systems
- many Rust-based OSes
  - rustboot, intermezzOS, blogOS
  - RedoxOS, Robigalia, Rux
  - TockOS, Discovery/f3, zinc
  - …
- … but, no std means
  - no threading
  - no memory allocation
  - no networking
  - not many crates
- what now?

unikernels

to the rescue

uni · kernel

stripped-down operating system running a single application

- runs directly on hypervisor/bare-metal
- single protection domain
- clear isolation boundaries
- low footprint (3-5MB)
- very short boot time (0.1-0.5s)

many different flavors

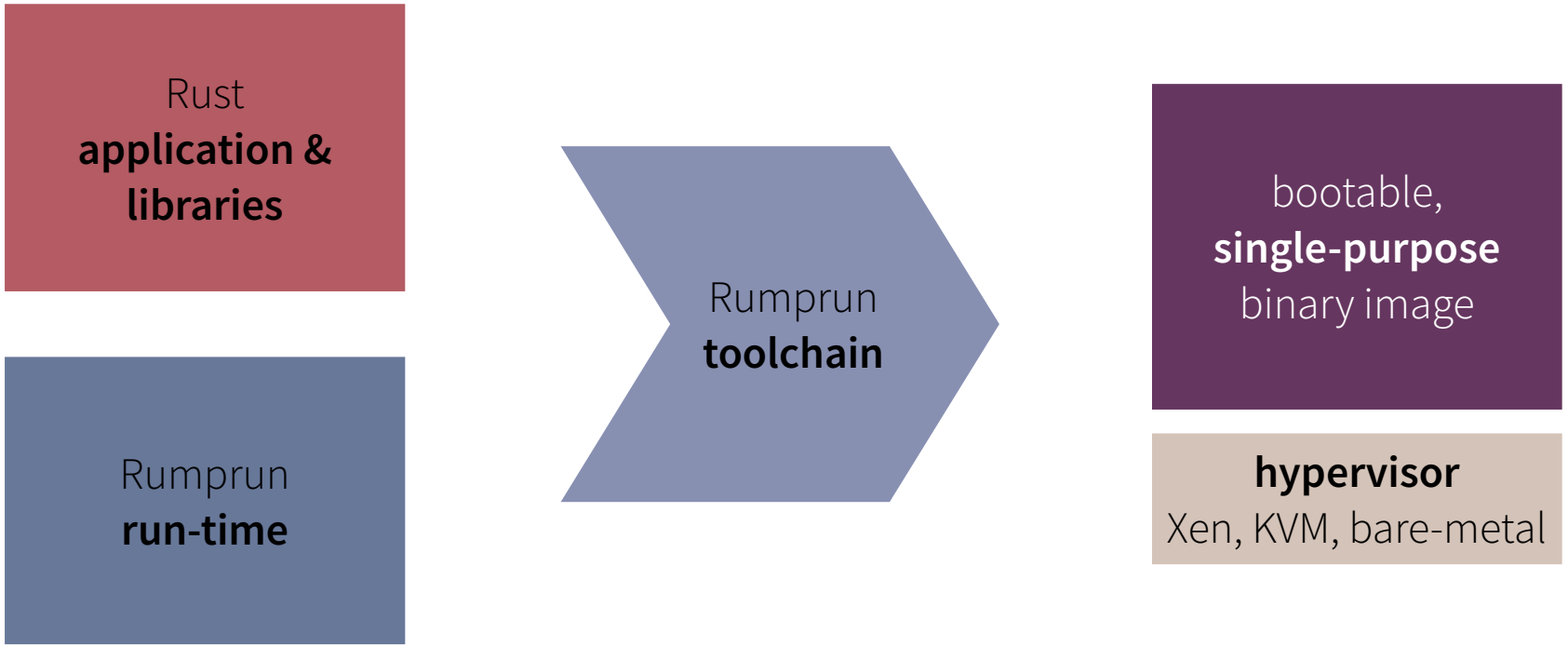
language-specific:

MirageOS (OCaml), IncludeOS (C++), HalVM (Haskell), LING (Erlang/OTP), runtime.js (Javascript), Clive (Go), ...

backwards-compatible:

OSv (POSIX), **Rumprun** (POSIX), Drawbridge (Win32)

The **Rumprun** Unikernel



Rumprun **workflow**

step 1: cross-compile

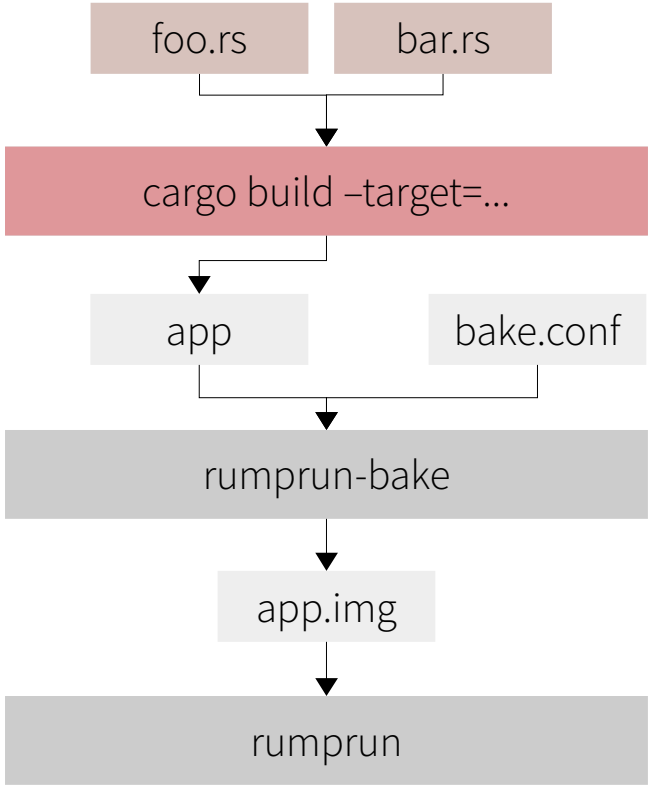
- compile against Rumprun's libc
- support for autotools & cmake

step 2: bake

- choose hypervisor, drivers & subsystems

step 3: launch

- mount points for block devices
- configure network
- environment variables, main args



getting started

```
$ git clone http://repo.rumpkernel.org/rumprun
$ cd rumprun
$ git submodule update --init
$ CC=cc ./build-rr.sh hw
[...]
>> Built rumprun for hw : x86_64-rumprun-netbsd
>> cc: x86_64-rumprun-netbsd-gcc
>>
>> ./build-rr.sh ran successfully
$ rustup target add x86_64-rumprun-netbsd
info: downloading component 'rust-std' for 'x86_64-rumprun-netbsd'
info: installing component 'rust-std' for 'x86_64-rumprun-netbsd'
```

demo

Rust on Rumprun

"Pssst, want a portable, kernel-quality TCP/IP stack?"

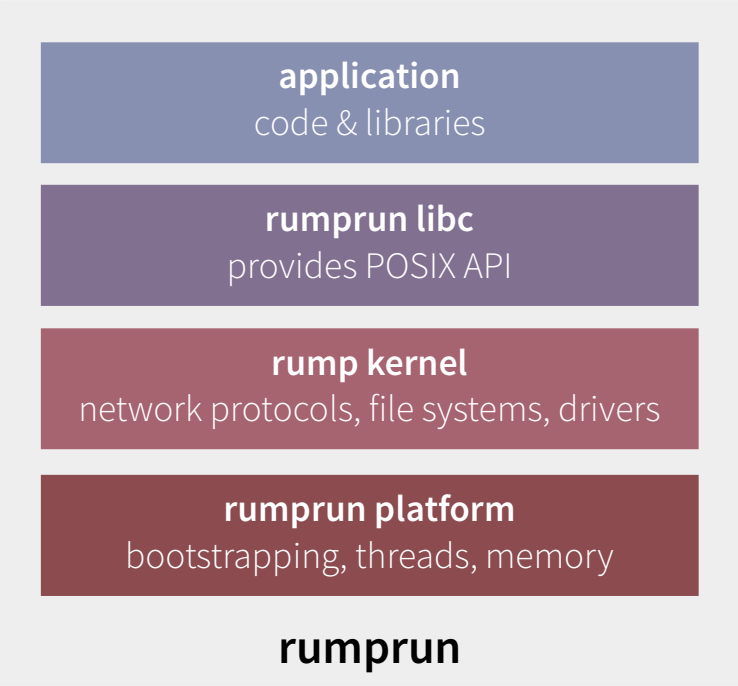
rump kernels

- free, reusable, componentized, kernel-quality drivers
  - hardware drivers
  - file systems, network protocols
  - POSIX system calls

https://twitter.com/rhatr/status/740244315411251201
https://blog.xenproject.org/2015/08/06/on-rump-kernels-and-the-rumprun-unikernel/

Rumprun: unikernel based on rump kernels

- from rump/NetBSD
  - rump kernel & drivers
  - (mostly) unmodified libc
- our own
  - platform-specific bootstrapping
  - "bare-metal" hypercall implementation
    - thread scheduler
    - memory allocator
    - console output



supported **packages** & **platforms**

packaged applications

- apache2, nginx, haproxy
- redis, mysql, sqlite, leveldb
- tor, mpg123, …

programming languages

- C/C++ (from toolchain)
- Lua, PHP, Python, Ruby, node.js
- Java, Rust, Erlang, Go

hardware platforms

- KVM, Xen, Qemu, EC2, bare-metal
- x86 (32/64bit)
- experimental ARM, RISC-V

Rust minimum requirements

- rustc, std since Rust 1.5 (Dec '15)
- rustup since Rust 1.9 (April '16)
- x86-64 only

debugging unikernels

gdb

- using qemu's debugging interface
  - same for Xen
- unikernel is a single ELF file
  - can step through the full stack

rump sysproxy

rumpctrl

- "remote shell"
- ifconfig, mount, sysctl

syscalls over TCP/IP

- not enabled by default
- even works for bare-metal

limitations

single address-space

- no `std::process`
  - no signals
- no virtual memory
  - no guard pages
  - some `mmap` calls are supported
- no `std::io::stdin`
  - use files or sockets

threading

- cooperative scheduler
- supports `std::thread`
- no multi-processor support (yet)
  - unikernel uses single core

more **rump kernel**

frankenlibc

- alternative rump unikernel
- interesting software architecture
- runs on Linux/FreeBSD/NetBSD
  - seccomp & Capsium support

nolibc Rumprun

- directly use the rump kernel
- suitable for Rust `#[no_std]`
- some assembly required
- experimental Linux/LibOS support
  - Linux drivers instead of NetBSD

contributing to Rust

- shared codebase with **NetBSD** port
  - some special cases during initialization
  - issues with legacy C symbols
- Rust community is incredibly welcoming
  - mentoring and feedback on pull requests
  - code is now part of continuous integration tests
    - Tier 2 platform
    - `libc` bindings tested with `ctest`

~~**bugs**~~ opportunities for contribution

toolchain issues

- requires ELF toolchain
  - some buggy versions
  - in doubt, Ubuntu 16.04 or Docker
- `rustup` NDK support?

unwinding is broken

- `panic=abort`

NetBSD support in crates

- does your crate support NetBSD?

cargo rumpbake

- used to exist
  - baking an image via cargo
- Rust integration in UniK

getting started:

http://rumpkernel.org

https://rustup.rs

@rumpkernel

#rumpkernel irc.freenode.net

contact me:

gandro@rumpkernel.org

twitter.com/@gandro23

gandro on irc.freenode.net

documentation:

- wiki, tutorials, how-to
- video tutorials
- rump man pages & book

rumprun code:

repo.rumpkernel.org/rumprun

repo.rumpkernel.org/rumprun-packages