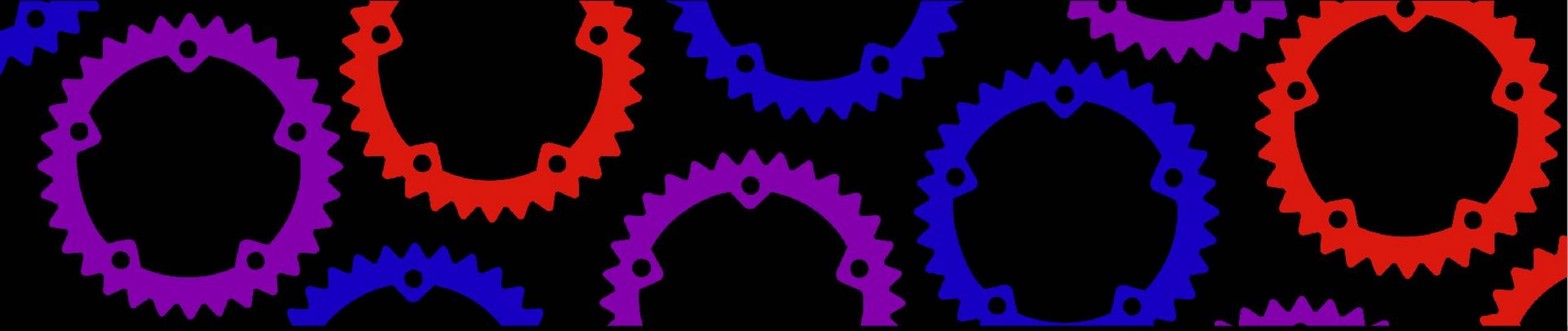


# let rust = ∏

Hawkingrei & Wayslog

2019.04.21

# CONTENTS

## rust在某站二三事

Remote cache
Thumbnail service
Golang vs rust
Rust ffi
actix vs rocket

## Things about aster

why rust
some tips
performance tools

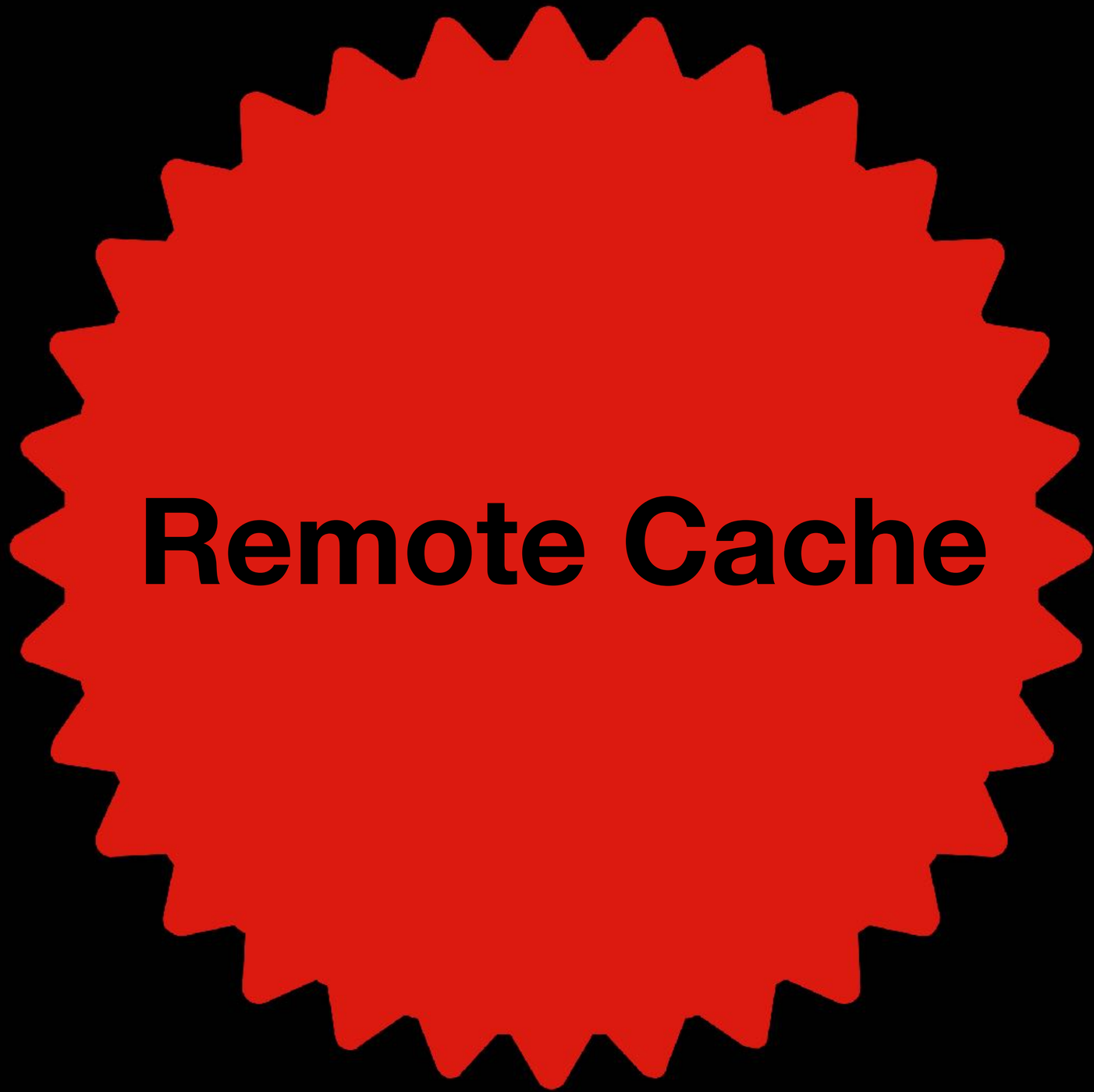
# PHP

# 动态语言一时爽
# 代码重构火葬场

# Golang

# Advantage of Golang

- Simple
- Garbage collection
- Goroutine and channel


Remote Cache

# CI/CD

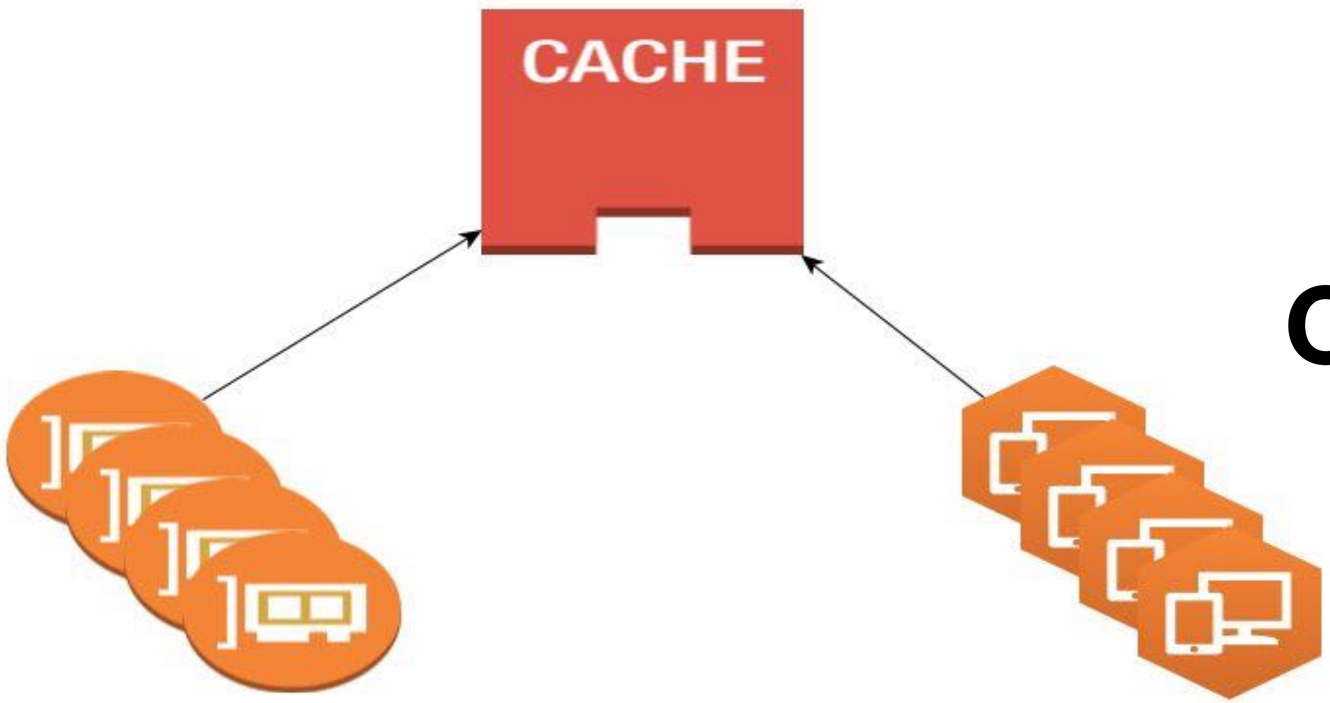
## 大仓库

**Bazel && Gradle => 保存编译产物**

**Prow => 分布式编译**

# Remote Cache Service

CI/CD




Office

# Remote Cache Service

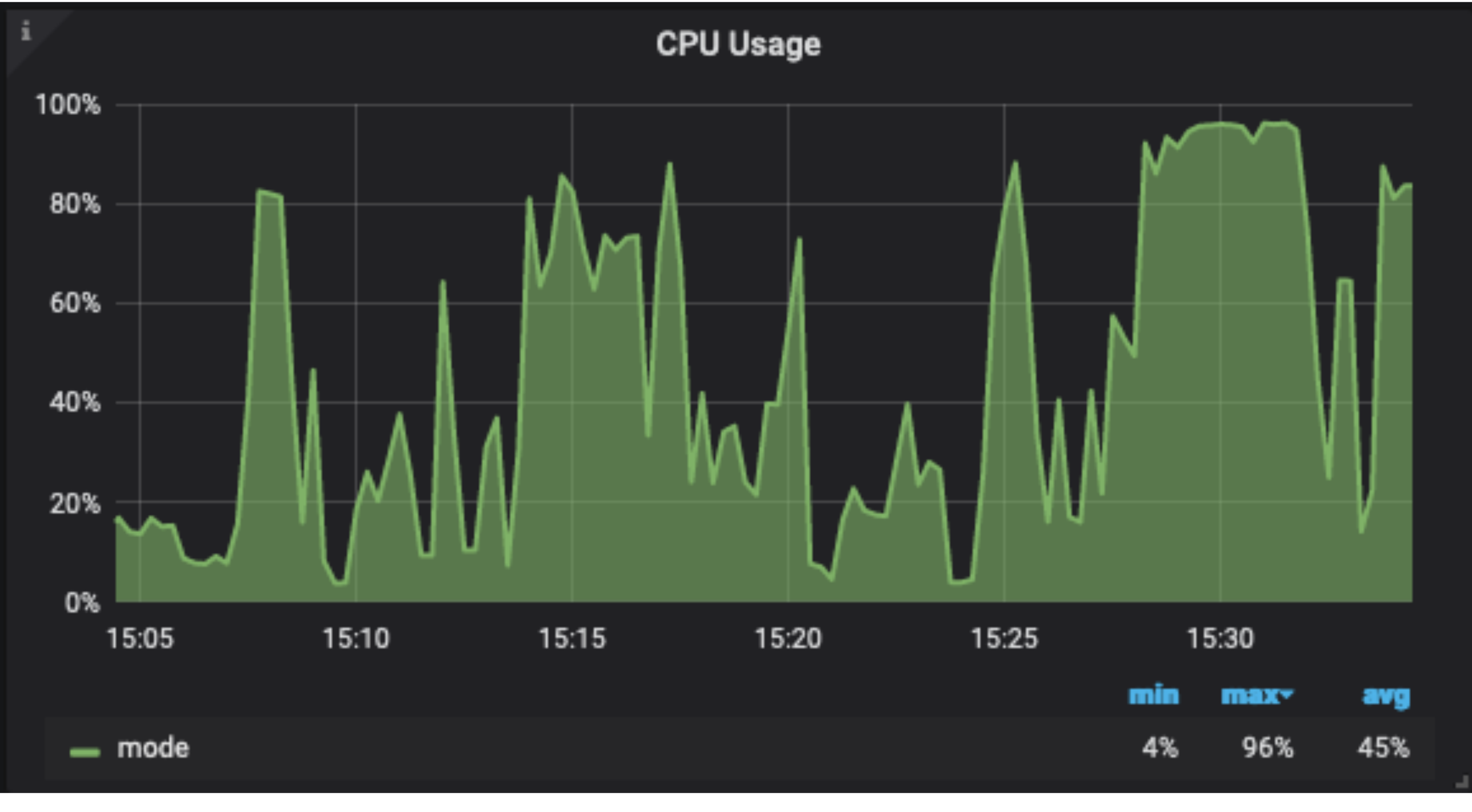
- 5000qps
- 一台服务器：
- 3TB上传量/天  每天不断增加

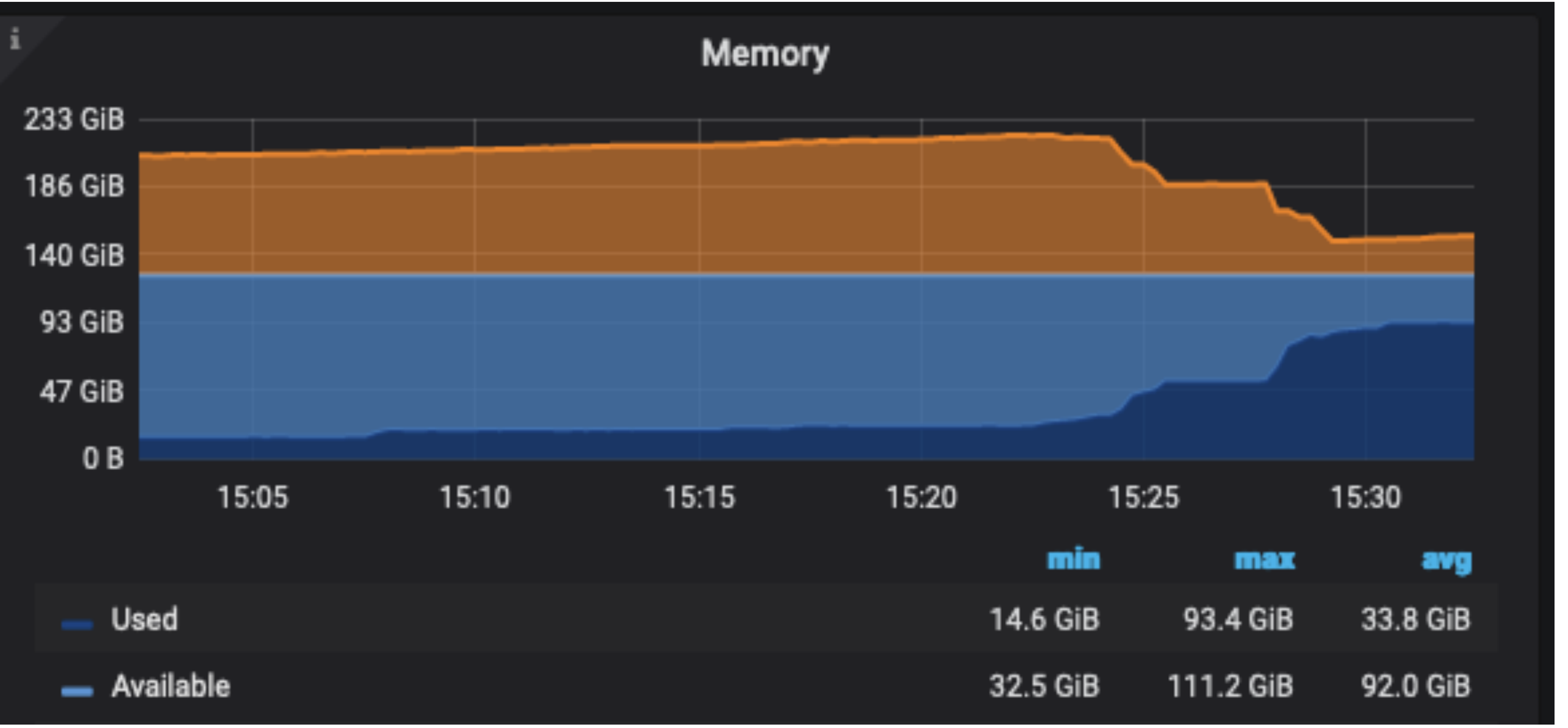
# kubernetes Greenhouse

https://github.com/kubernetes/test-infra/tree/master/greenhouse

# bgreenhouse

- Add zstd compression （CGO）
- Add disk garbage collection
- Support gradle


CPU Usage
100%
80%
60%
40%
20%
0%
15:05
15:10
15:15
15:20
15:25
15:30
min
max
avg
mode
4%
96%
45%


Memory
233 GiB
186 GiB
140 GiB
93 GiB
47 GiB
0 B
15:05
15:10
15:15
15:20
15:25
15:30
min
max
avg
Used
14.6 GiB
93.4 GiB
33.8 GiB
Available
32.5 GiB
111.2 GiB
92.0 GiB

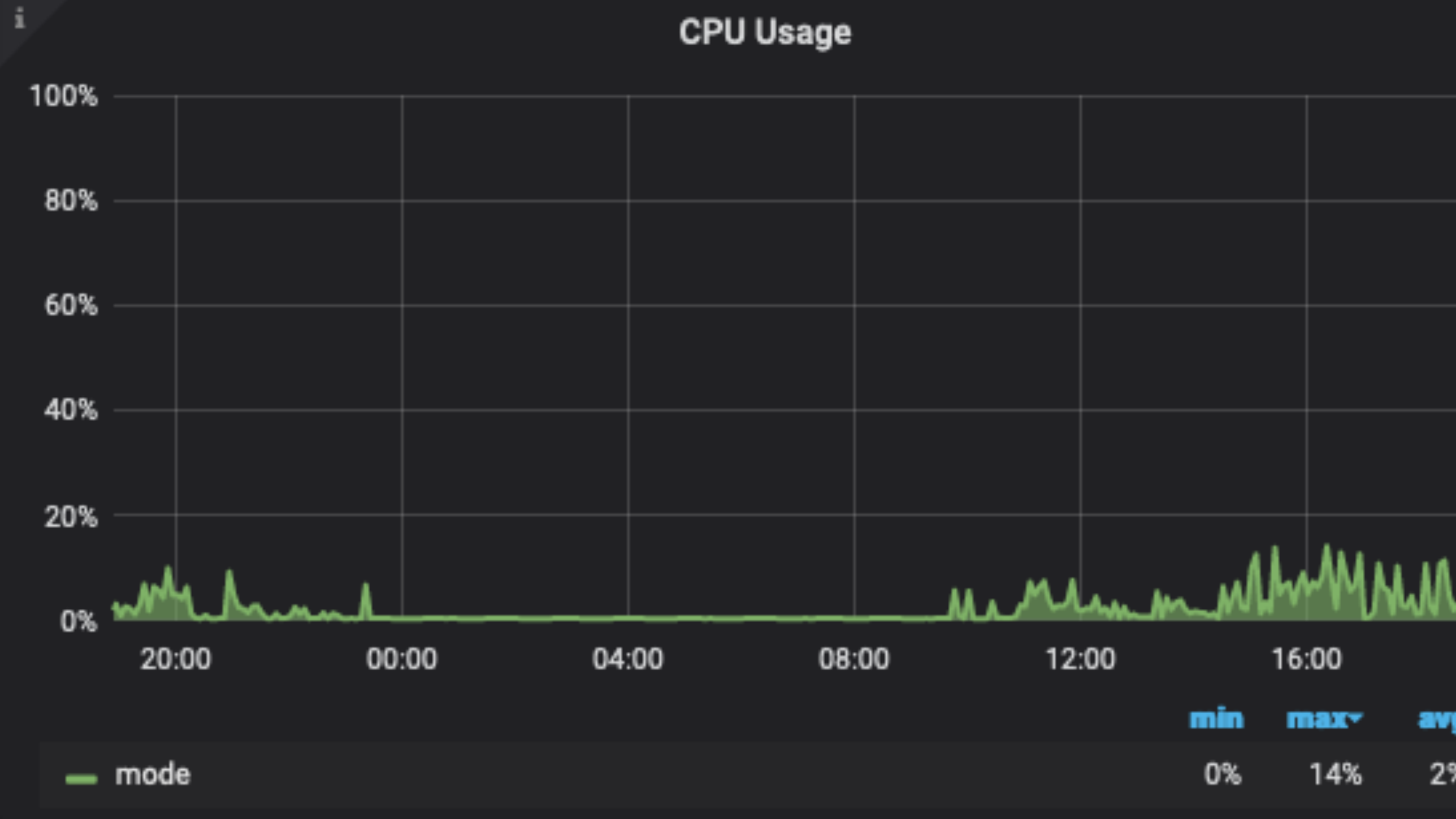

Boom!

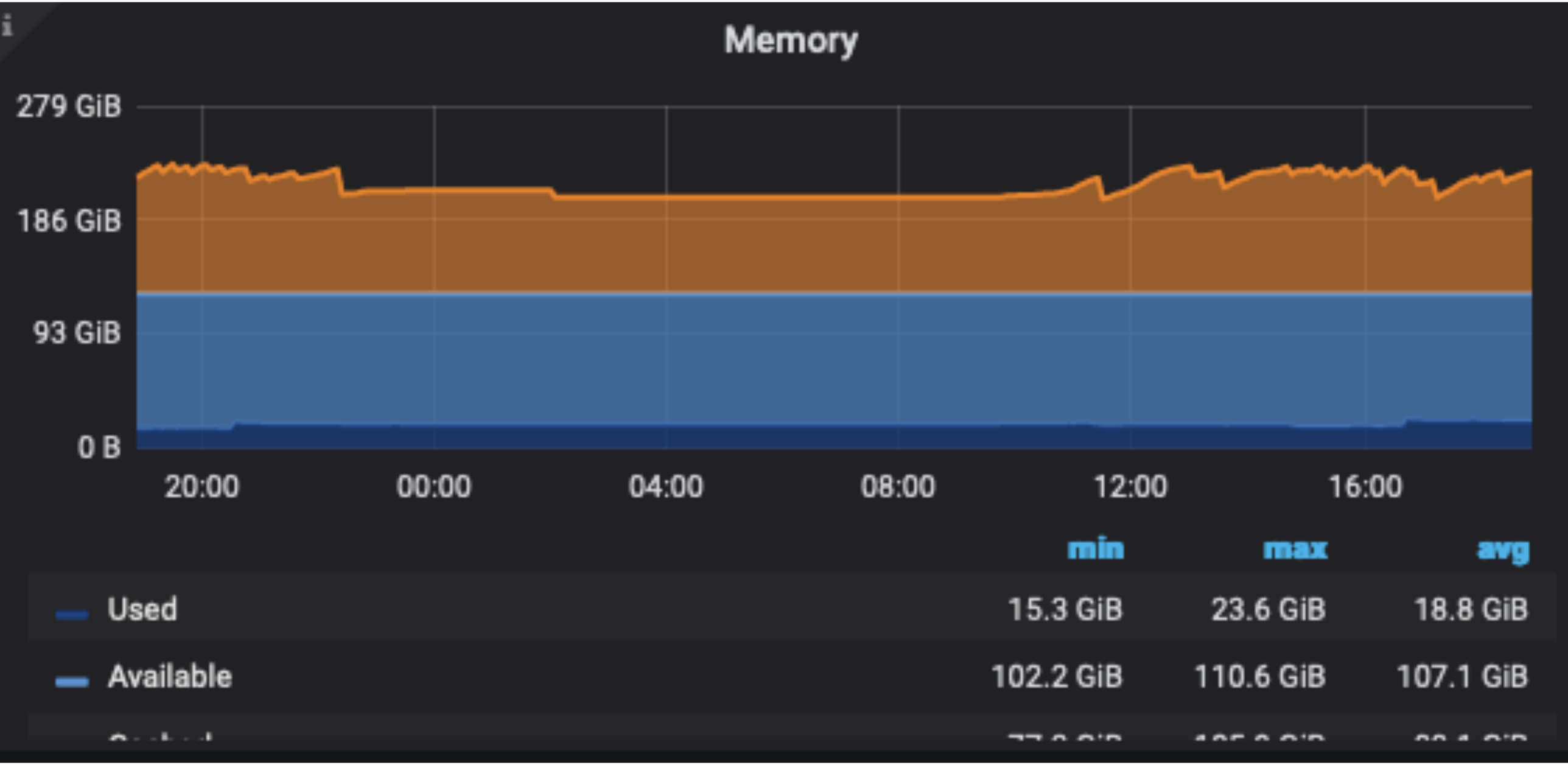
# Cgo is not Go

*https://dave.cheney.net/2016/01/18/cgo-is-not-go*

# Greenhouse-rs

*Rocket && zstd-rs && rust-prometheus*


CPU Usage
100%
80%
60%
40%
20%
0%
20:00
00:00
04:00
08:00
12:00
16:00
min
max
mode
0%
14%


Memory
279 GiB
186 GiB
93 GiB
0 B
20:00
00:00
04:00
08:00
12:00
16:00
min
max
avg
Used
15.3 GiB
23.6 GiB
18.8 GiB
Available
102.2 GiB
110.6 GiB
107.1 GiB

# Golang VS Rust

# Golang
# 没有泛型、没有枚举... ☹️

# Performance

# Simd in golang

```
//go:noescape

func popcntMaskSliceAsm(s, m []uint64) uint64
```

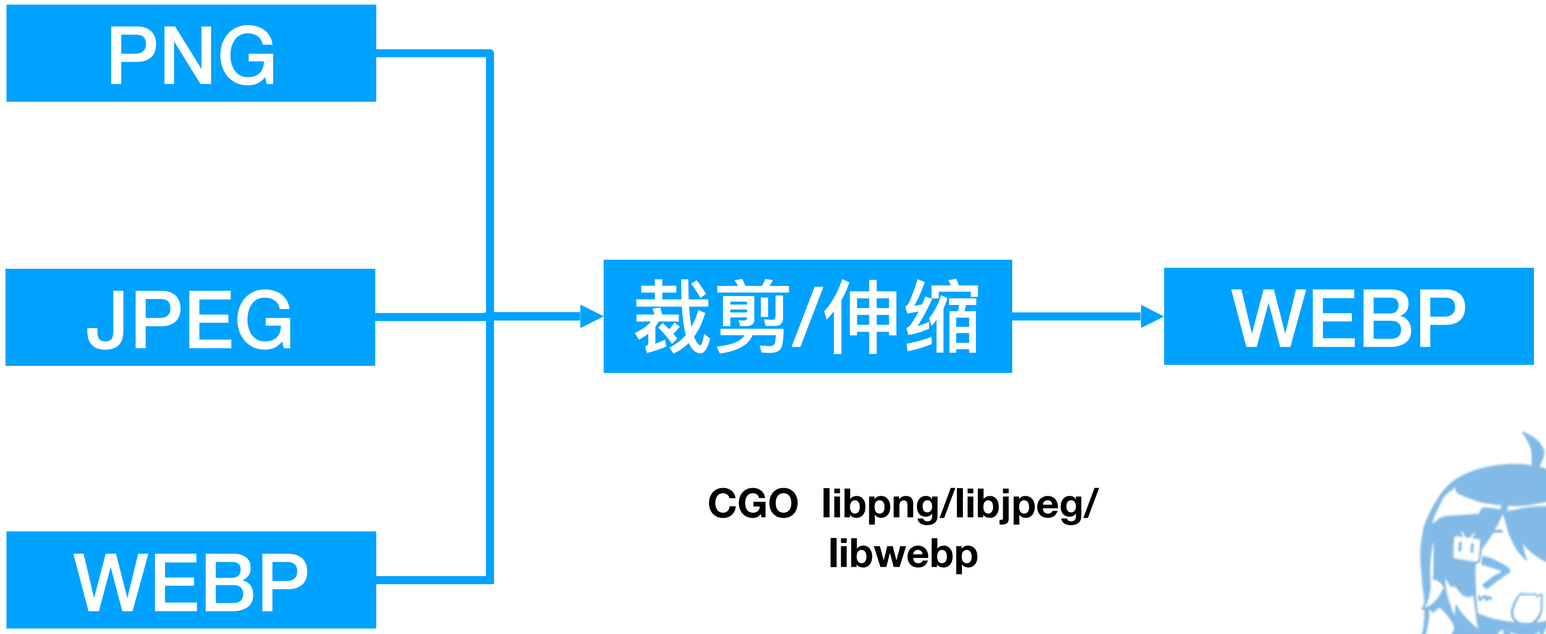
```
TEXT ·popcntMaskSliceAsm(SB),4,$0-56
XORQ    AX, AX
MOVQ    s+0(FP), SI
MOVQ    s_len+8(FP), CX
TESTQ   CX, CX
JZ              popcntMaskSliceEnd
MOVQ    m+24(FP), DI
popcntMaskSliceLoop:
MOVQ    (DI), DX
NOTQ    DX
ANDQ    (SI), DX
POPCNTQ_DX_DX
ADDQ    DX, AX
ADDQ    $8, SI
ADDQ    $8, DI
LOOP    popcntMaskSliceLoop
popcntMaskSliceEnd:
MOVQ    AX, ret+48(FP)
RET
```

# Simd in Rust

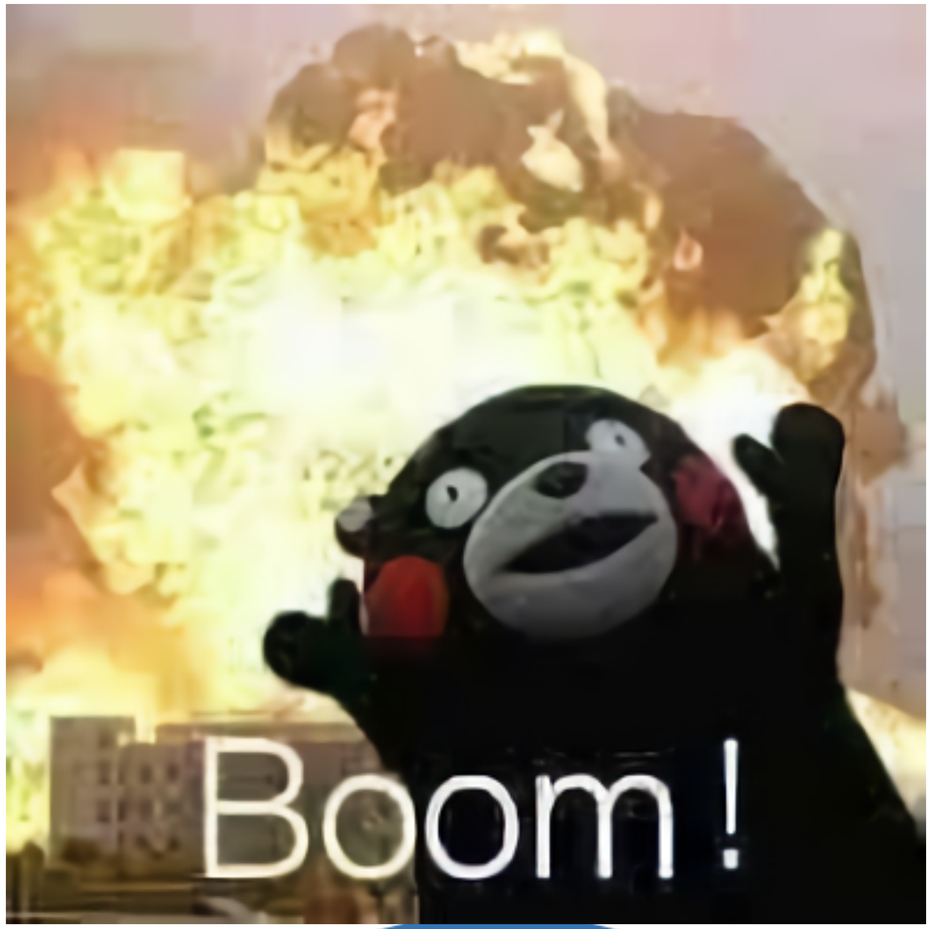
```
let x = arch::_mm_xor_si128(
    arch::_mm_clmulepi64_si128(x, k3k4, 0x10),
    arch::_mm_srli_si128(x, 8),
);
let x = arch::_mm_xor_si128(
    arch::_mm_clmulepi64_si128(
        arch::_mm_and_si128(x, arch::_mm_set_epi32(0, 0, 0, !0)),
        arch::_mm_set_epi64x(0, K5),
        0x00,
    ),
    arch::_mm_srli_si128(x, 4),
);
```


Thumbnail
service

# Thumbnail service




CGO libpng/libjpeg/
libwebp


Boom !

# Golang cgo -> Rust FFI

# Thumbnail-rs

# Bindgen

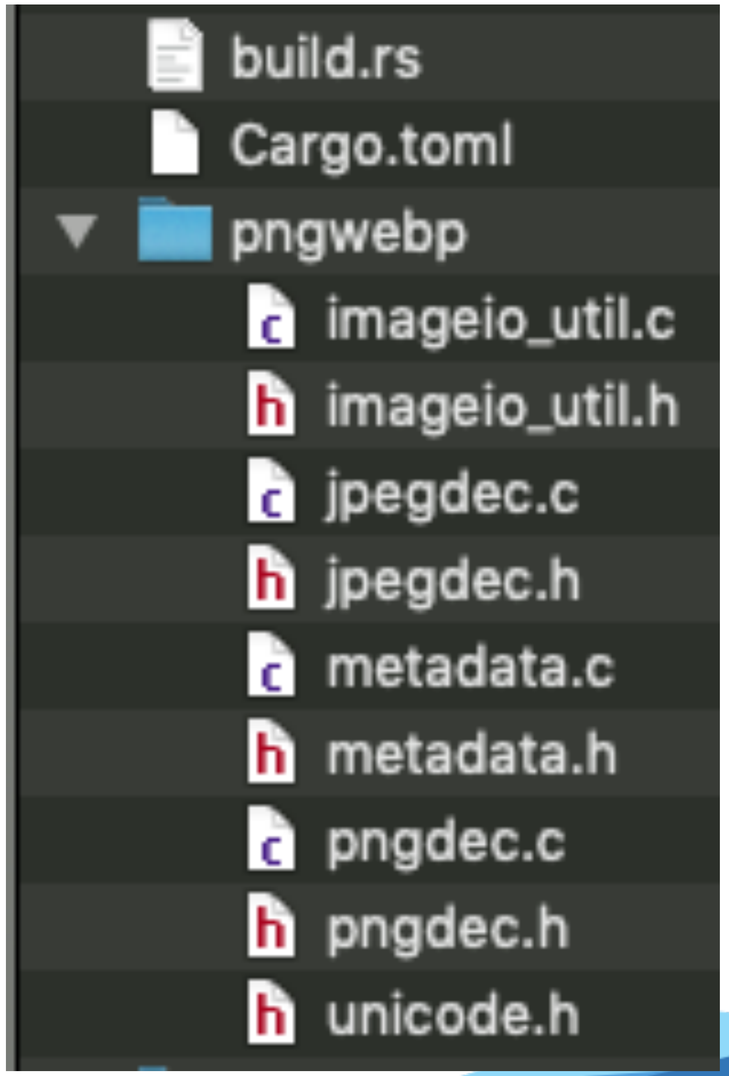
wrapper.h

```
#include <bzlib.h>
```

```
extern crate bindgen;

use std::env;
use std::path::PathBuf;

fn main() {
    // Tell cargo to tell rustc to link the system bzip2
    // shared library.
    println!("cargo:rustc-link-lib=bz2");

    // The bindgen::Builder is the main entry point
    // to bindgen, and lets you build up options for
    // the resulting bindings.
    let bindings = bindgen::Builder::default()
        // The input header we would like to generate
        // bindings for.
        .header("wrapper.h")
        // Finish the builder and generate the bindings.
        .generate()
        // Unwrap the Result and panic on failure.
        .expect("Unable to generate bindings");

    // Write the bindings to the $OUT_DIR/bindings.rs file.
    let out_path = PathBuf::from(env::var("OUT_DIR").unwrap());
    bindings
        .write_to_file(out_path.join("bindings.rs"))
        .expect("Couldn't write bindings!");
}
```

# But …

# c++ wrapper and bindgen

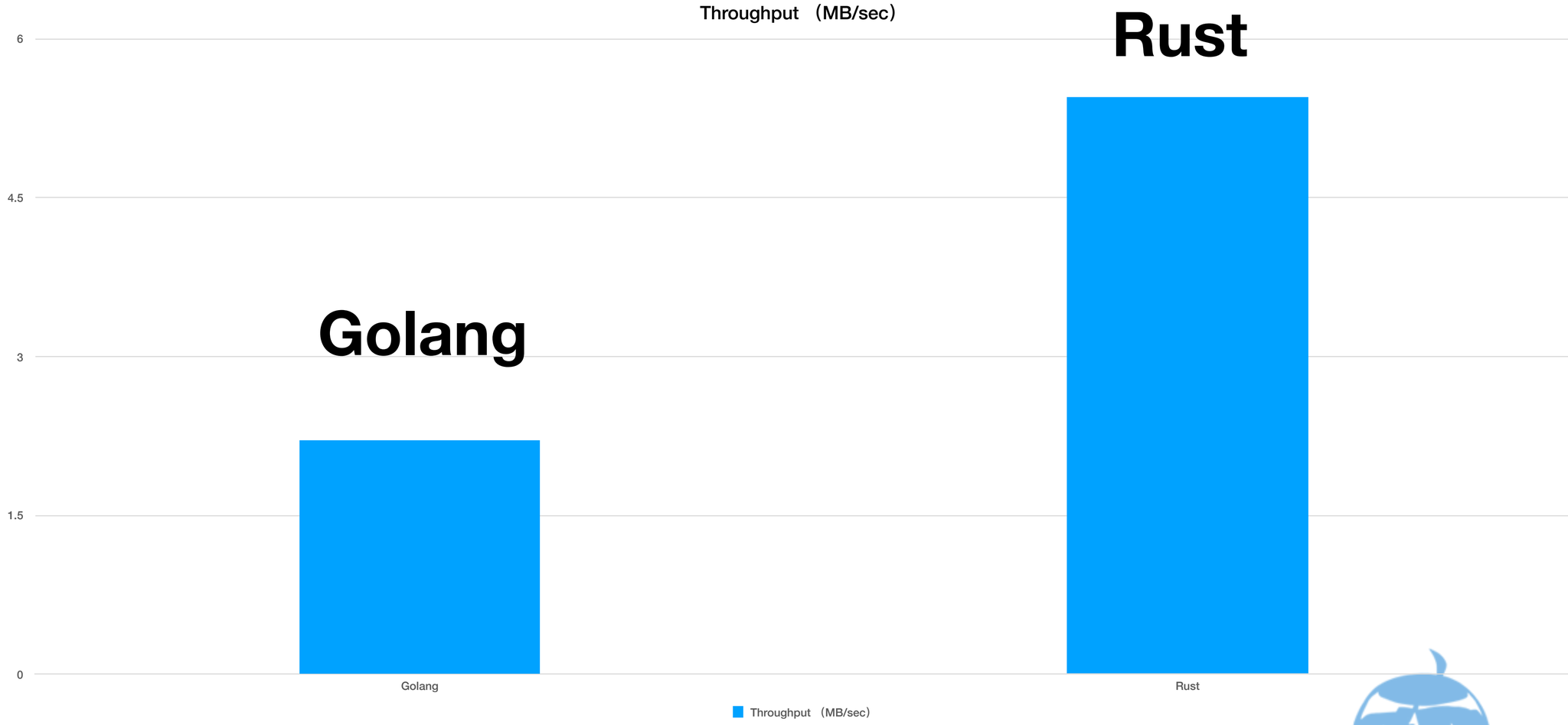



```rust
// build.rs

extern crate cc;

fn main() {
    cc::Build::new()
        .file("foo.c")
        .file("bar.c")
        .compile("foo");
}
```

# Library version

# Cmake && bindgen

```rust
extern crate bindgen;
extern crate cmake;

use std::env;
use std::path::PathBuf;
use bindgen::Builder;
use cmake::Config;

fn main() {
    let dst = Config::new("libwebp").build_target("webp").build();
    println!("cargo:rustc-link-search=native={}/build", dst.display());
    println!("cargo:rustc-link-lib=static=webp");

    let bindings = Builder::default()
        .no_unstable_rust()
        .header("wrapper.h")
        .trust_clang_mangling(false)
        .generate()
        .expect("Unable to generate bindings");
```



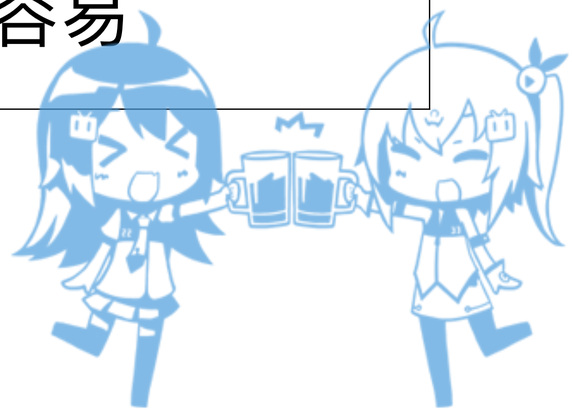

Golang

Rust

# Actix_web vs Rocket

# Actix_web vs Rocket

| | Actix web | Rocket |
|---|---|---|
| Macro router | NO | Yes |
| future | Yes | No |
| 性能 | 高 | 一般 |
| 难易程度 | 难 | 容易 |



# rust 编译太慢了...😇

# Prow

https://github.com/kubernetes/test-infra/tree/master/prow

# Bazel

{Fast, Correct} - Choose two

Build and test software of any size, quickly and reliably

bazelbuild / rules_rust  Add Repo  View Repository  Unwatch 25  Unstar 99  Fork 54

Code  Issues 33  Pull requests 5  Projects 0  Wiki  Insights

Rust rules for Bazel https://bazelbuild.github.io/rules_rust/

google / cargo-raze  Add Repo  View Repository  Watch 9  Star 68  Fork 14

Code  Issues 22  Pull requests 3  Projects 0  Wiki  Insights

Generate Bazel BUILD from Cargo dependencies!

# Bazel

- 优势
  - remote cache
  - 分布式编译
  - 跨语言
- 缺点
  - 从零开始学习rust编译 😂😇

# sccahe

https://github.com/mozilla/sccache

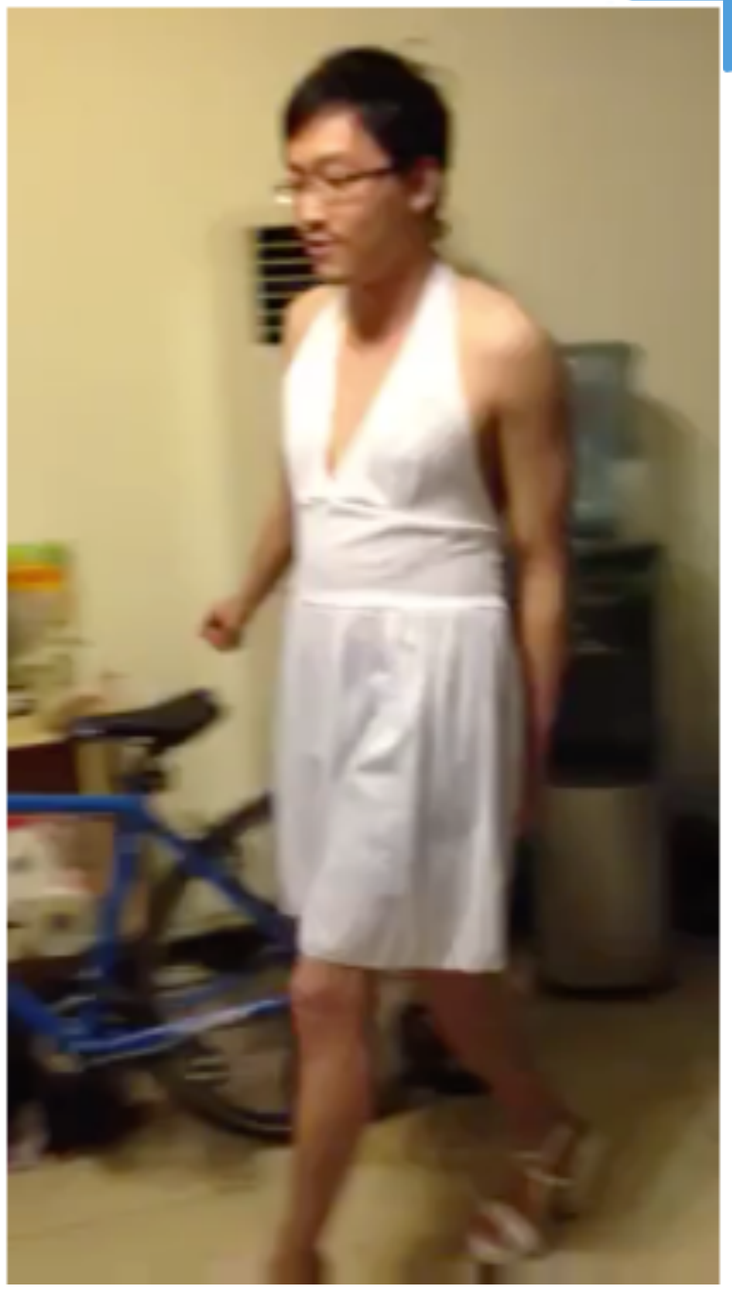
# sccahe

- 优势
- 储存支持：local, redis, memcache, s3
- 使用简单：

```
cargo install sccache
RUSTC_WRAPPER=[path to sccache] cargo build
```

- 缺点
- It doesn't support ccache's direct mode.
- It doesn't support all kinds of compiler flags
- It doesn't support an option like CCACHE_BASEDIR.


Cache Proxy



Things About Aster

介绍：技术的深度决定技术的广度

苦行之旅：无处安放的类型转换

苦行之旅：drop函数与唤醒

苦行之旅：让人头秃的profile

苦行之旅：parser回溯与DC老师的二三事

苦行之旅：我最亲爱的syscall别闹了

插入知识点：零拷贝技术

终极解法：▪▪▪▪▪▪▪▪▪▪▪▪▪（刮开有奖）

# aster： 一个简单的缓存代理

1.更快

- QPS 和 latency 指标更好

2.扩展性更强

- 对比C语言做新功能的接入能力比较强

**3.更稳定**

- 相信borrow checker的力量（只要没有ICE）

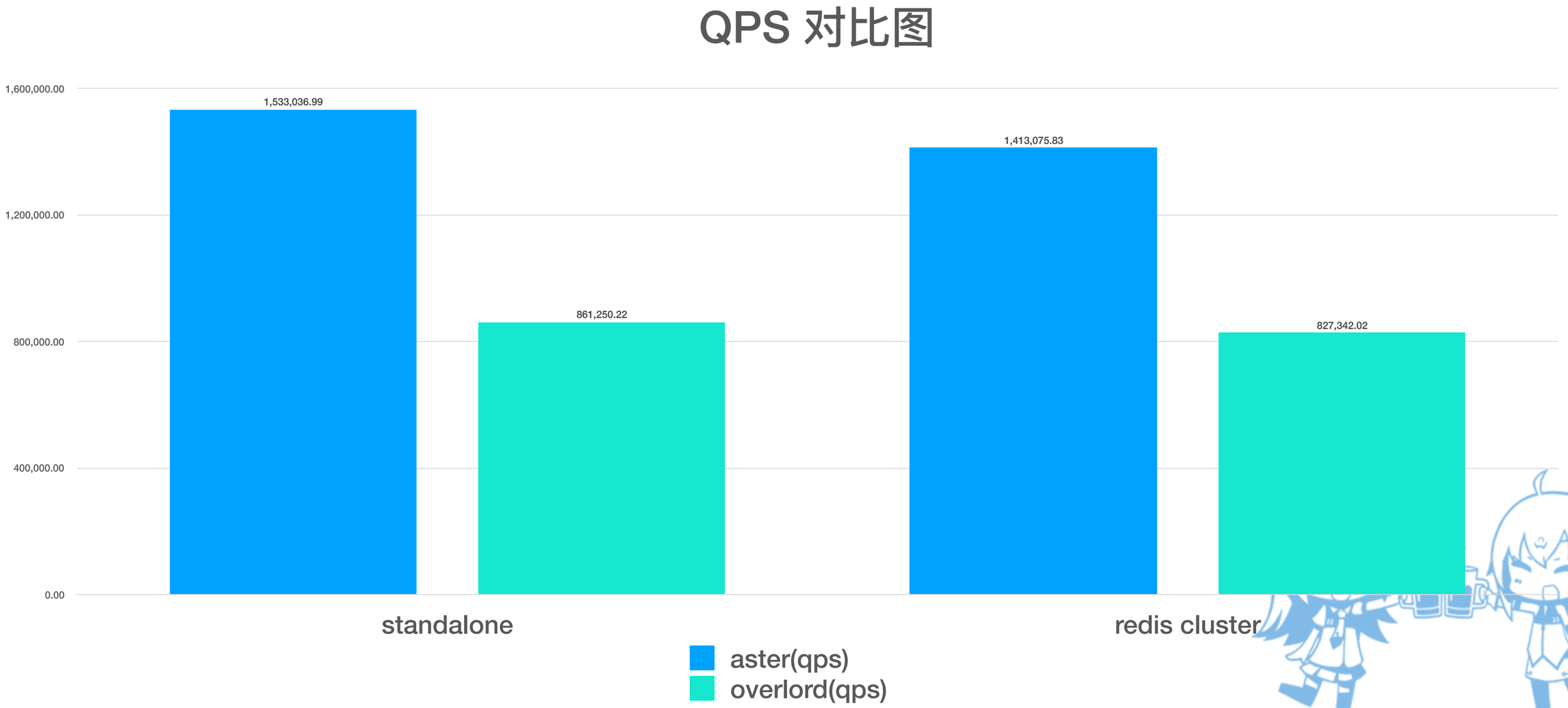
4.Rust写的

- **来本次大会的理由**

# 缓存代理的进化史

| corvus | overlord | aster |
| --- | --- | --- |
| **1. 协议单一** | **1. 多协议支持** | **1. 多协议支持** |
| 2. 速度快 | 2. 速度不够快 | 2. 速度够快 |
| 3. 扩展能力差 | 3. 扩展能力强 | 3. 扩展能力强 |
| **4. 先行者** | **4. 紧张刺激的开发中** | **4. 紧张刺激的开发中** |
| **5. 维护复杂** | 5. 附带平台 | 5. 可直接与overlord匹配 |
| **6. 不再维护** | **6. 高级功能支持** | **6. 高级功能开发中** |

# 测试数据对比


QPS 对比图
1,600,000.00
1,200,000.00
800,000.00
400,000.00
0.00
1,533,036.99
861,250.22
1,413,075.83
827,342.02
standalone
redis cluster
aster(qps)
overlord(qps)

# Things About Aster

介绍：技术的深度决定技术的广度

苦行之旅：无处安放的类型转换

苦行之旅：drop函数与唤醒

苦行之旅：让人头秃的profile

苦行之旅：parser回溯与DC老师的二三事

苦行之旅：我最亲爱的syscall别闹了

插入知识点：零拷贝技术

终极解法：▪▪▪▪▪▪▪▪▪▪▪▪▪（刮开有奖）

# 苦行之旅：无处安放的类型转换(一)

# 苦行之旅：无处安放的类型转换(二)

- 写 Future/Stream**/Codec** 的时候尽量统一 Item 和 Error **类型**
- **善用 failure::Error**
- **记得处理 SendError 类型**

Things About Aster

介绍：技术的深度决定技术的广度

苦行之旅：无处安放的类型转换

苦行之旅： drop函数与唤醒

苦行之旅：让人头秃的profile

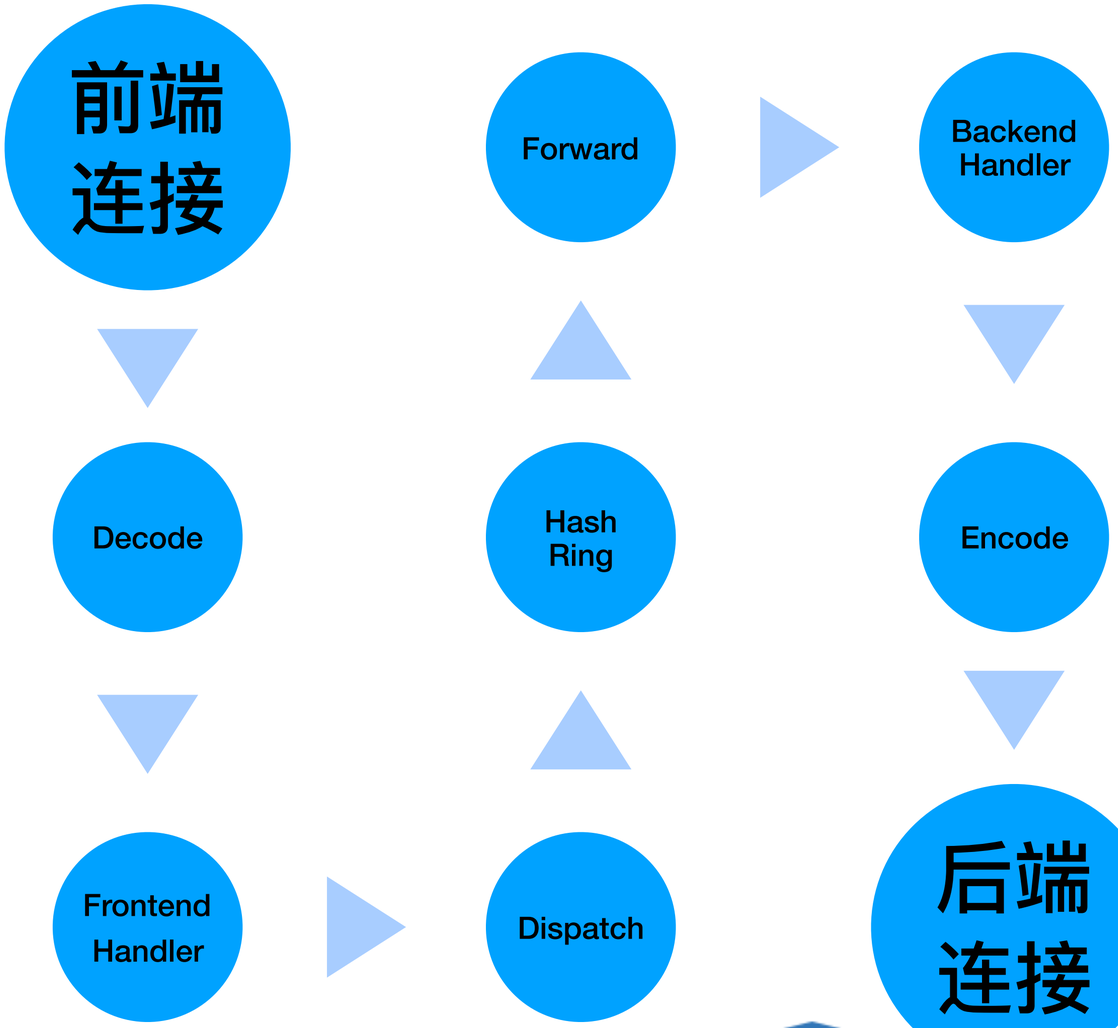
苦行之旅：parser回溯与DC老师的二三事

苦行之旅：我最亲爱的syscall别闹了

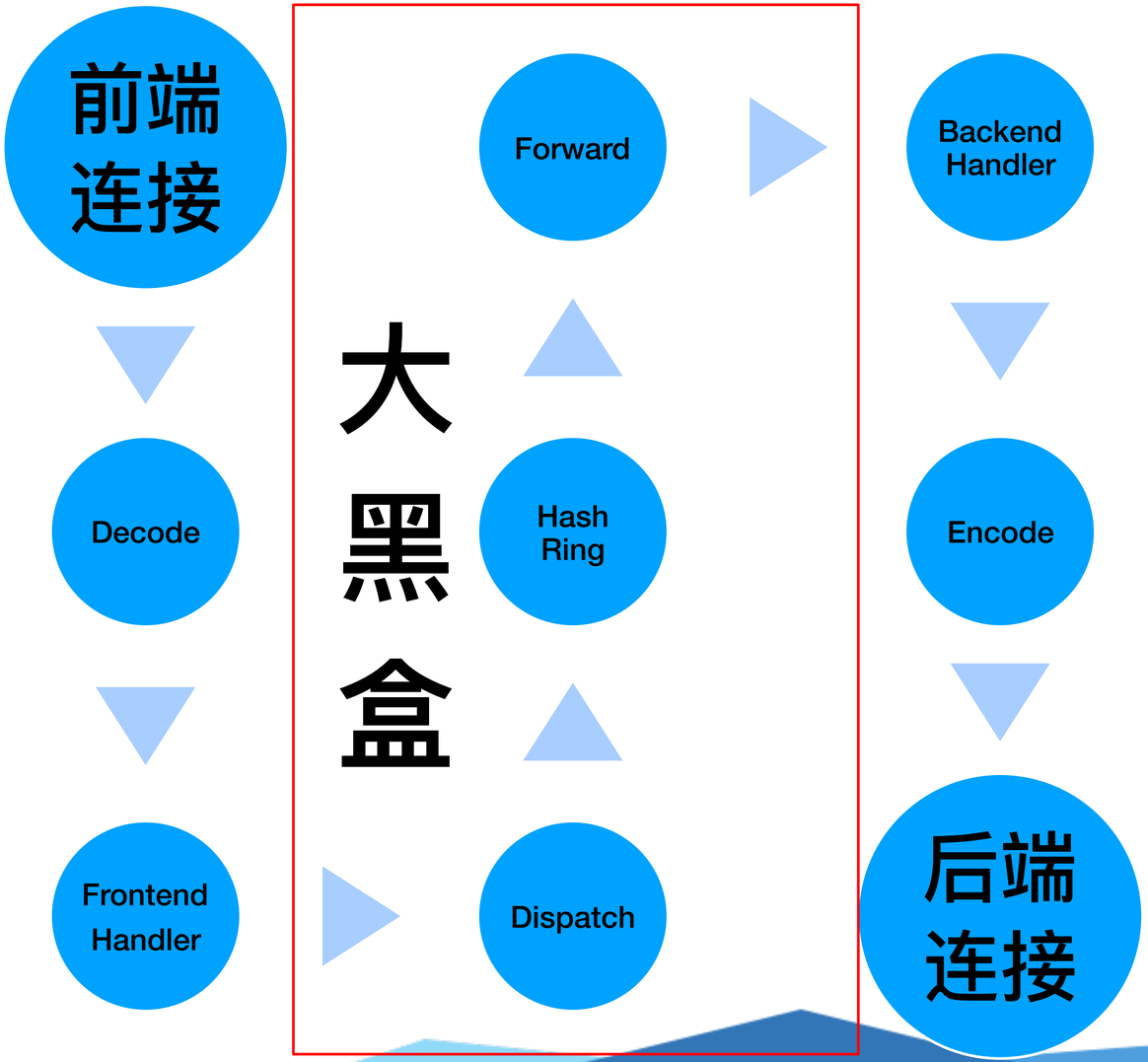
插入知识点：零拷贝技术

终极解法：▪▪▪▪▪▪▪▪▪▪▪▪▪（刮开有奖）

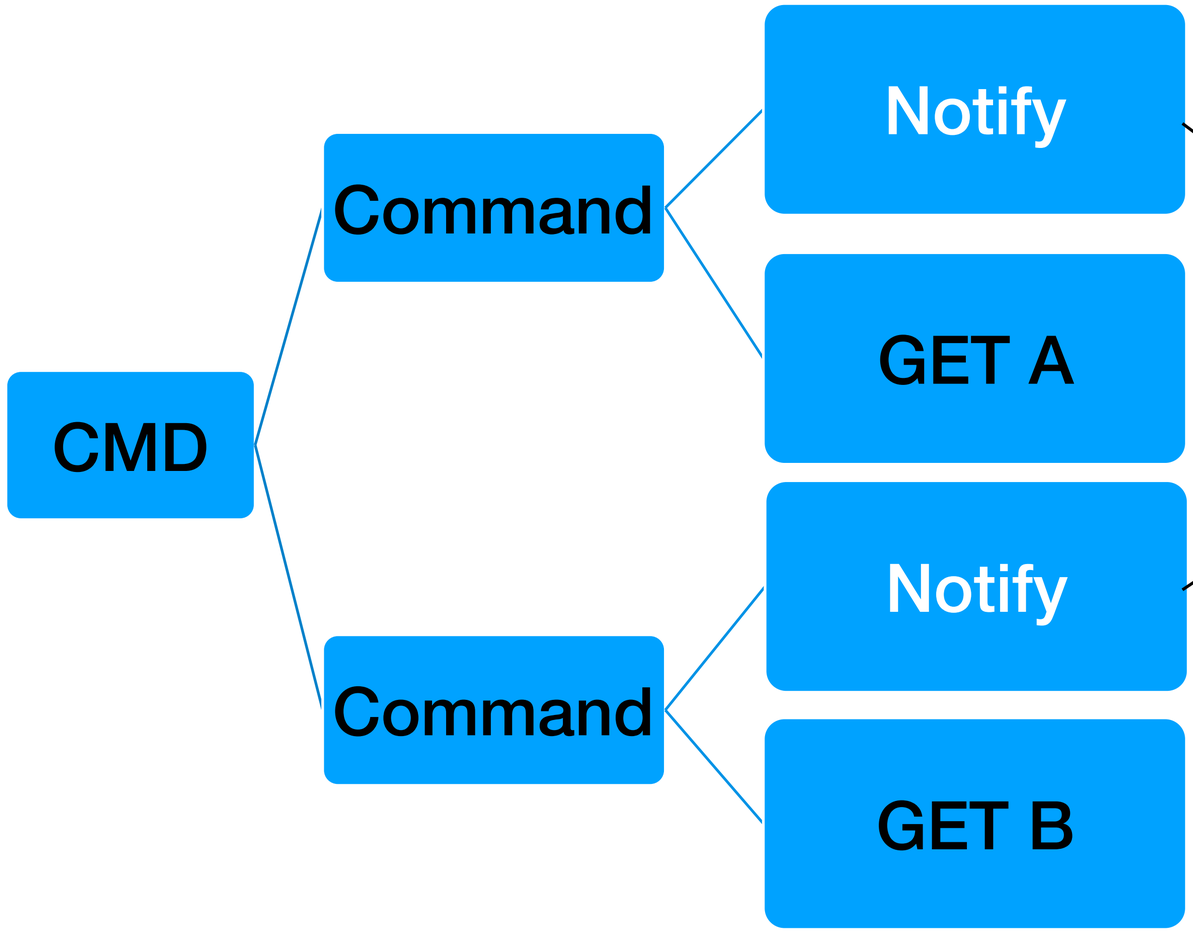
# 苦行之旅: drop函数与唤醒(一)




# 苦行之旅: drop函数与唤醒(二)




# 苦行之旅: drop函数与唤醒(三)




```
struct Notify {
    task: Option<Task>,
    count: Rc<Cell<usize>>,
}
```

Things About Aster

介绍：技术的深度决定技术的广度

苦行之旅：无处安放的类型转换

苦行之旅： drop函数与唤醒

苦行之旅：让人头秃的profile

苦行之旅：parser回溯与DC老师的二三事

苦行之旅：我最亲爱的syscall别闹了

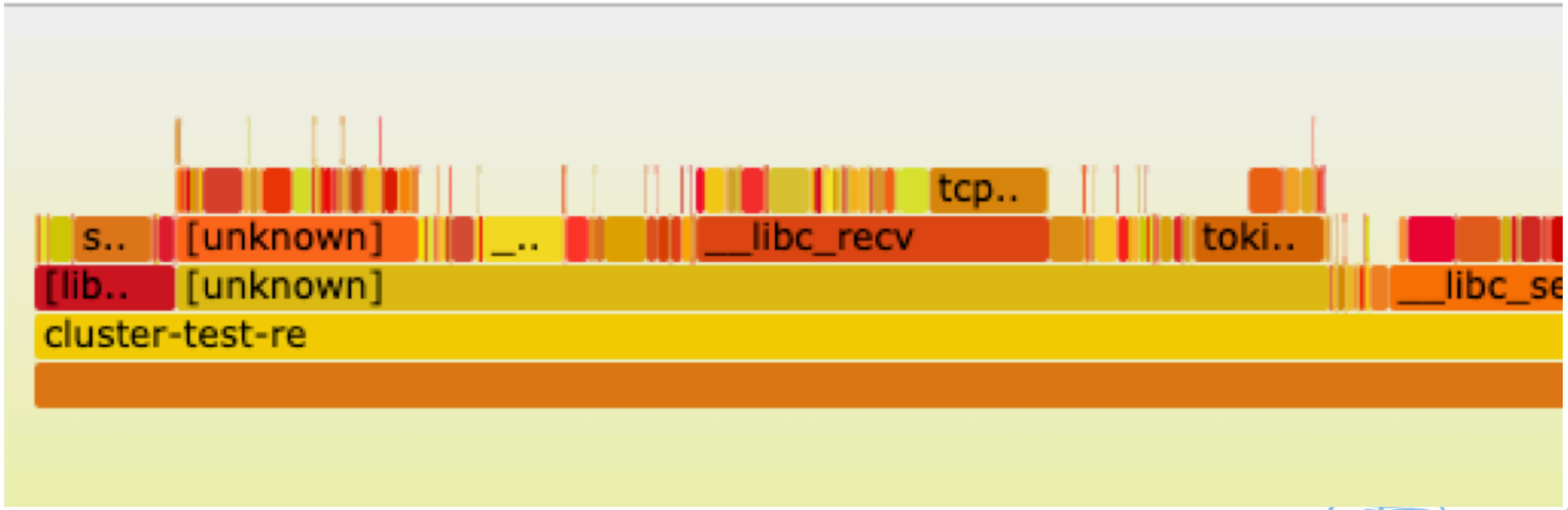
插入知识点：零拷贝技术

终极解法：▪▪▪▪▪▪▪▪▪▪▪▪▪（刮开有奖）

# 苦行之旅：让人头秃的profile(一)

```
perf record -F 99 -a -g -p ${pid} -- sleep 60
perf script | ./stackcollapse-perf.pl > out.perf-folded
./flamegraph.pl out.perf-folded > perf-aster.svg
```

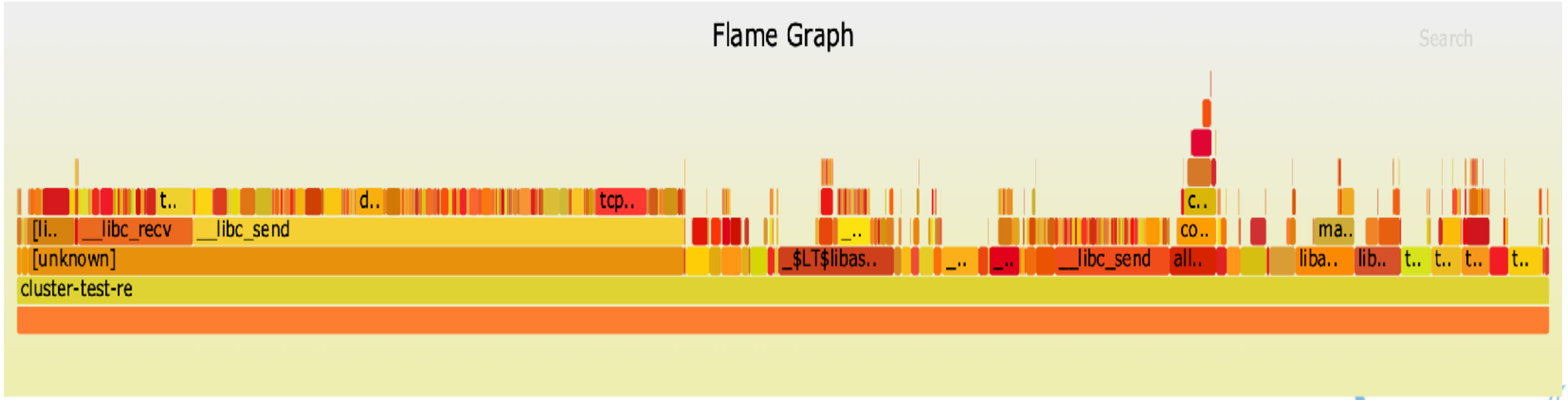
# 苦行之旅：让人头秃的profile(二)




# 苦行之旅：让人头秃的profile(三)

```
perf record -F 99 --call-graph DWRAF -a -g -p ${pid} -- sleep 60
perf script | ./stackcollapse-perf.pl > out.perf-folded
./flamegraph.pl out.perf-folded > perf-aster.svg
```

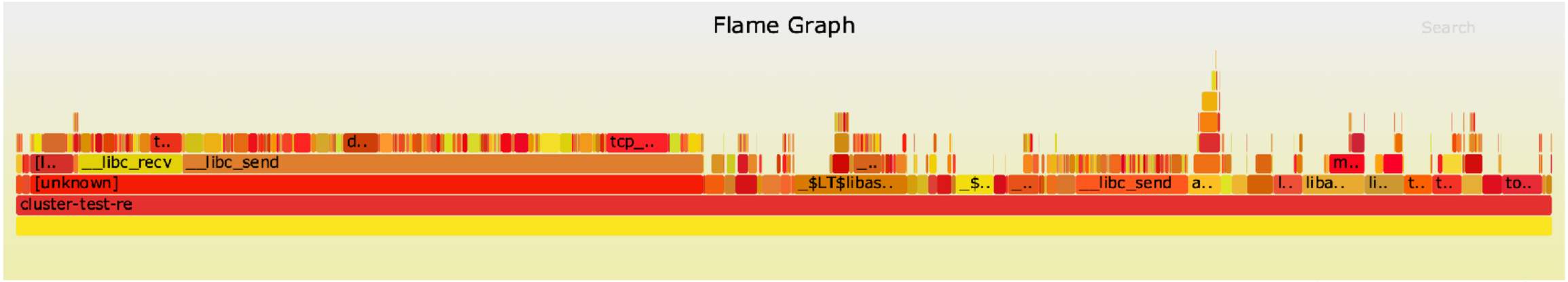
# 苦行之旅：让人头秃的profile(四)

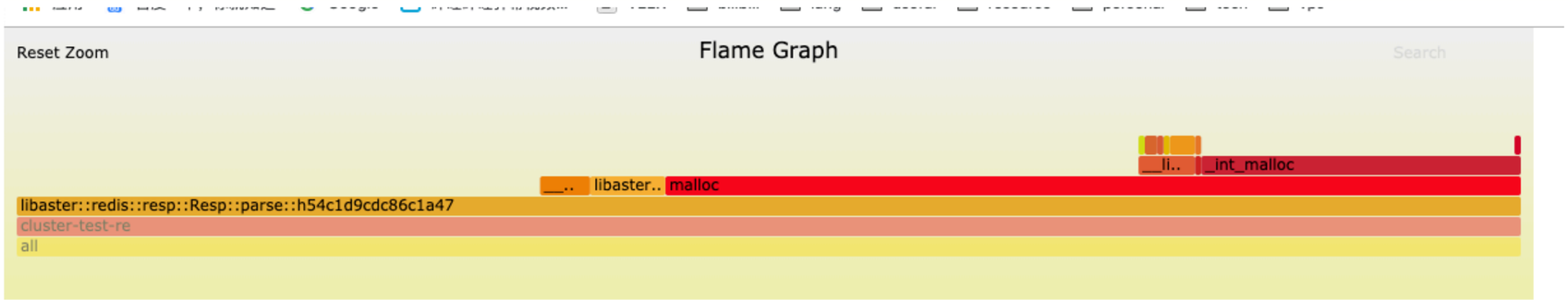



# 苦行之旅：让人头秃的profile(五)

```
RUSTFLAGS="-C force-frame-pointers=y" cargo rustc --release --bins
# run your benchmarks

perf record -F 99 --call-graph DWRAF -a -g -p ${pid} -- sleep 60
perf script | ./stackcollapse-perf.pl > out.perf-folded
./flamegraph.pl out.perf-folded > perf-aster.svg
```

# 苦行之旅：让人头秃的profile(六)

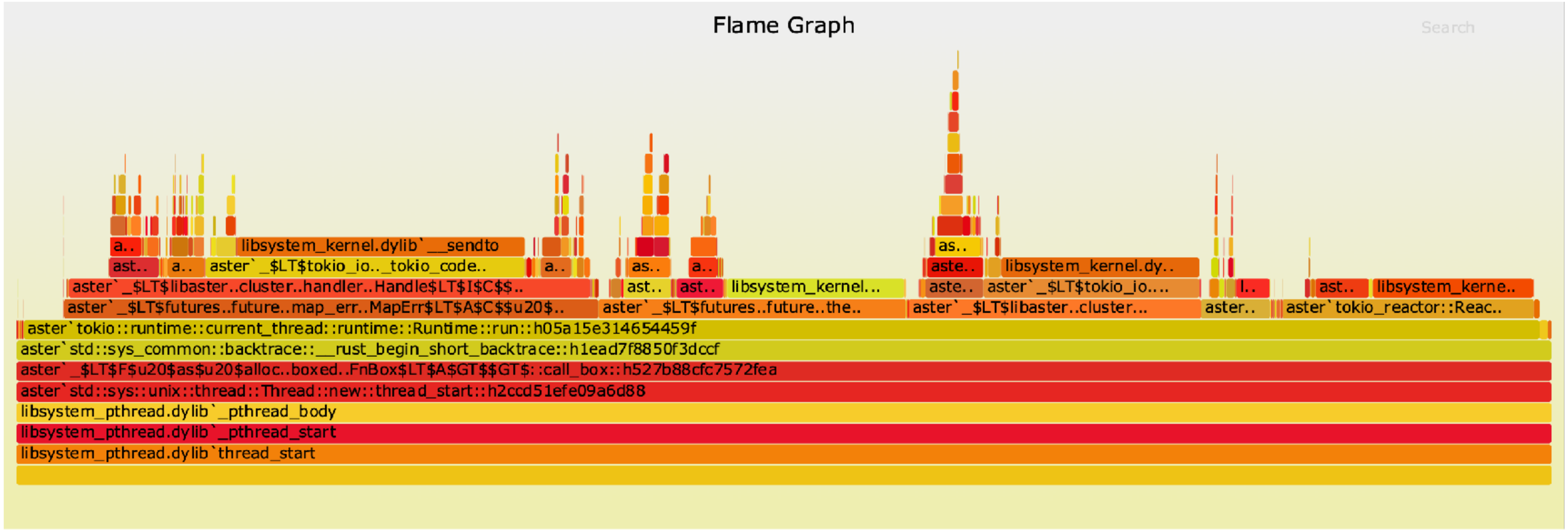



# 苦行之旅：让人头秃的profile(七)

```
sudo cargo flamegraph --release --bin aster
// EQUALS WITH:
// sudo dtrace -o cargo-flamegraph.stacks \
//  -x ustackframes=100 \
//  -n "profile-997 /pid == $target/ { @[ustack(100)] = count(); }"

# send benchmark request
Ctrl-c...
stackcollapse.pl cargo-flamegraph.stacks > flamegraph.folded
flamegraph.pl flamegraph.folded > aster.svg
```

# 苦行之旅：让人头秃的profile(八)



# Things About Aster

介绍：技术的深度决定技术的广度

苦行之旅：无处安放的类型转换

苦行之旅：drop函数与唤醒

苦行之旅：让人头秃的profile

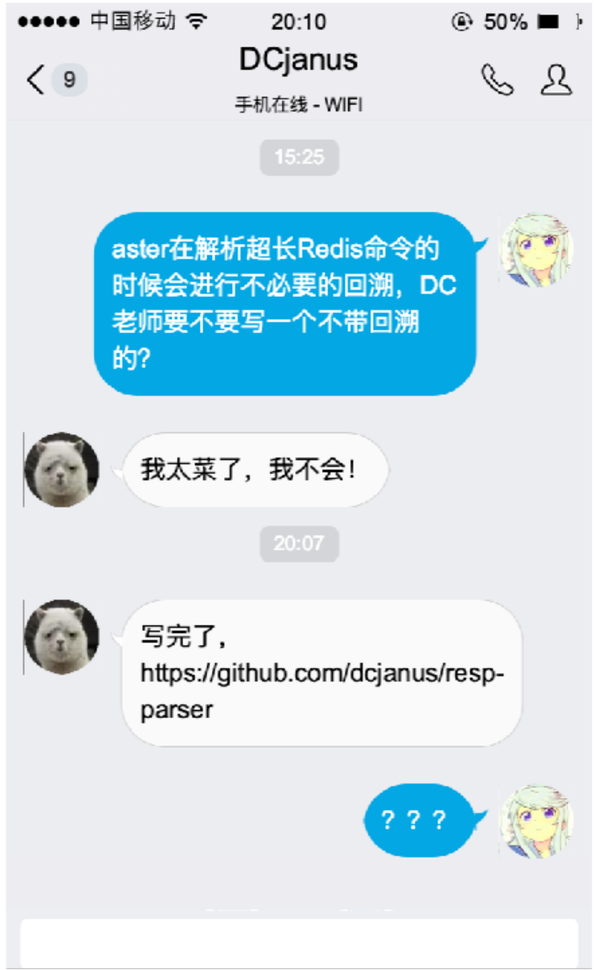
苦行之旅：parser回溯与DC老师的二三事

苦行之旅：我最亲爱的syscall别闹了

插入知识点：零拷贝技术

终极解法：▪▪▪▪▪▪▪▪▪▪▪▪▪（刮开有奖）

# 苦行之旅：parser回溯与DC老师的二三事





Cursor写进空数组般的笑声


# 苦行之旅：FSM的解法

```rust
pub struct Parser {
    state: State,
    // work for line scan
    scanned: usize,
    // work for bulk and array
    expect: usize,
    // work for array parse
    current: Option<Box<Parser>>,
    // work for buffer
    buffer: Option<Vec<Message>>,
}
```

慢(Slow)

```rust
State::ArraySize => {
    if let Some(size) = self.scan_integer(src)? {
        if size < 0 {
            self.state = State::Init;
            return Ok(Some(Message::Array(None)));
        }
        self.expect = size as usize;
        self.state = State::ArrayBody;
        self.current = Some(Box::new(Parser::default()));
        self.buffer = Some(Vec::with_capacity(self.expect));
    }
}
State::ArrayBody => {
    if let Some(message) = self.current.as_mut().unwrap().decode(src)? {
        let buffer = self.buffer.as_mut().unwrap();
        buffer.push(message);

        if buffer.len() == self.expect {
            self.state = State::Init;
            let result = std::mem::replace(&mut self.buffer, None);
            return Ok(Some(Message::Array(result)));
        }
    }
}
```

# Things About Aster

介绍：技术的深度决定技术的广度

苦行之旅：无处安放的类型转换

苦行之旅：drop函数与唤醒

苦行之旅：让人头秃的profile

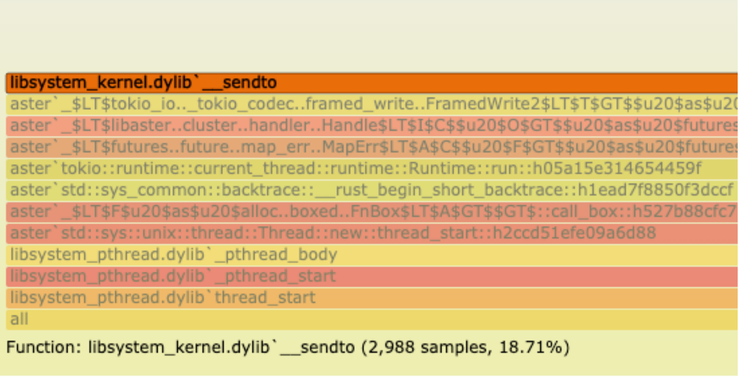
苦行之旅：parser回溯与DC老师的二三事

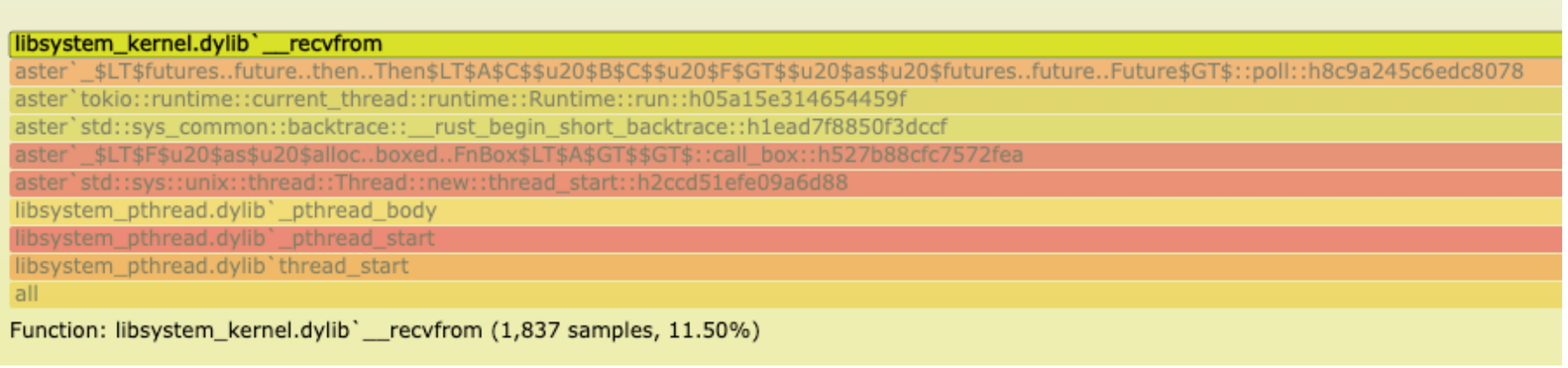
苦行之旅：我最亲爱的syscall别闹了

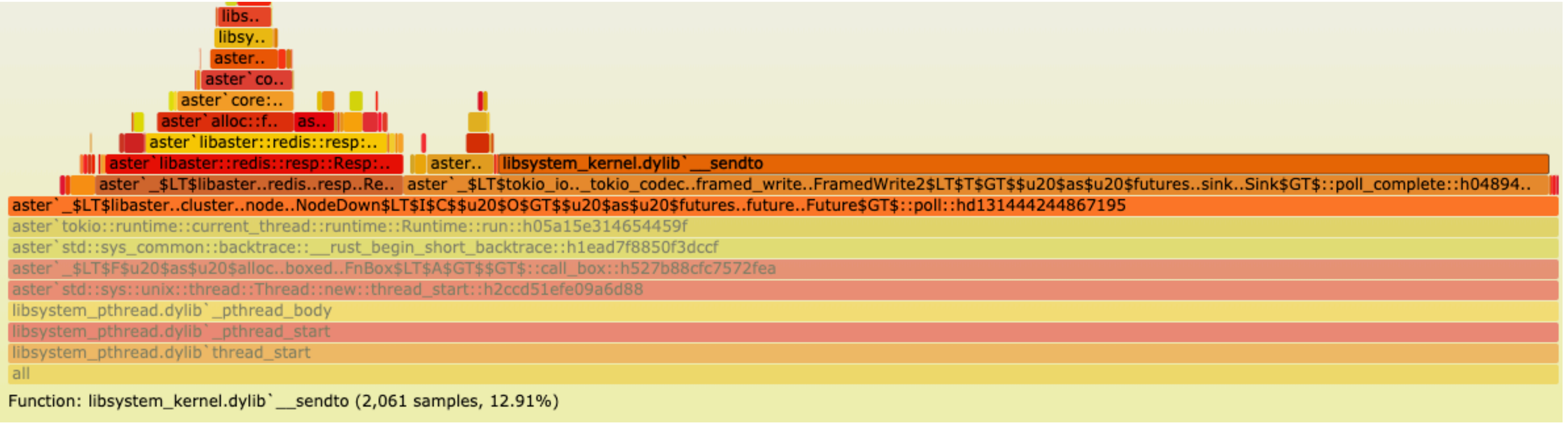
插入知识点：零拷贝技术

终极解法：▪▪▪▪▪▪▪▪▪▪▪▪▪（刮开有奖）

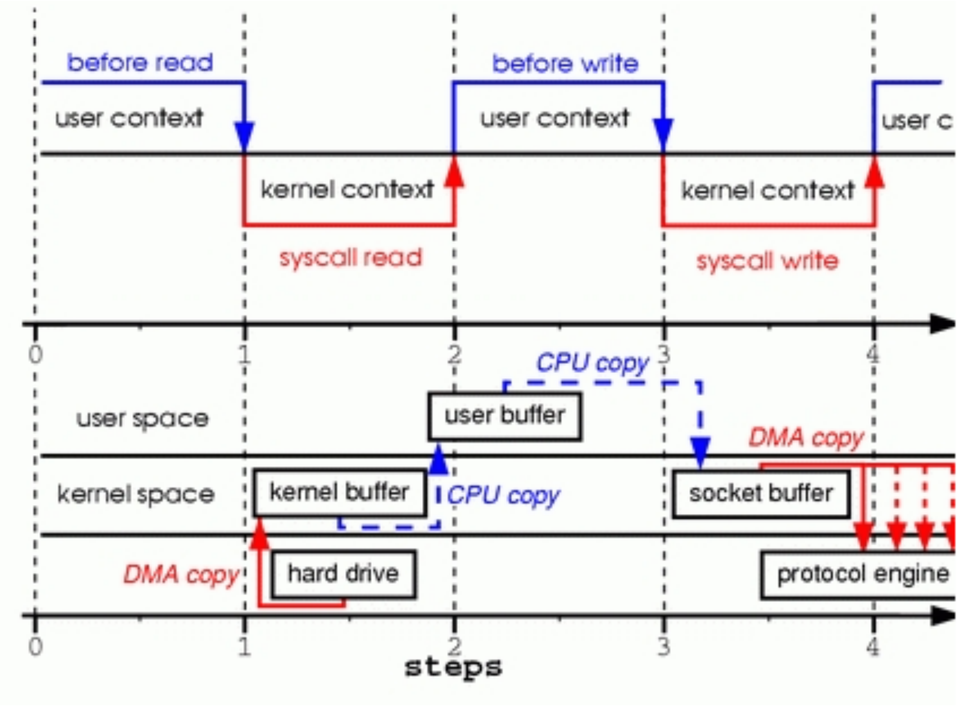
# 苦行之旅：我最亲爱的syscall别闹了



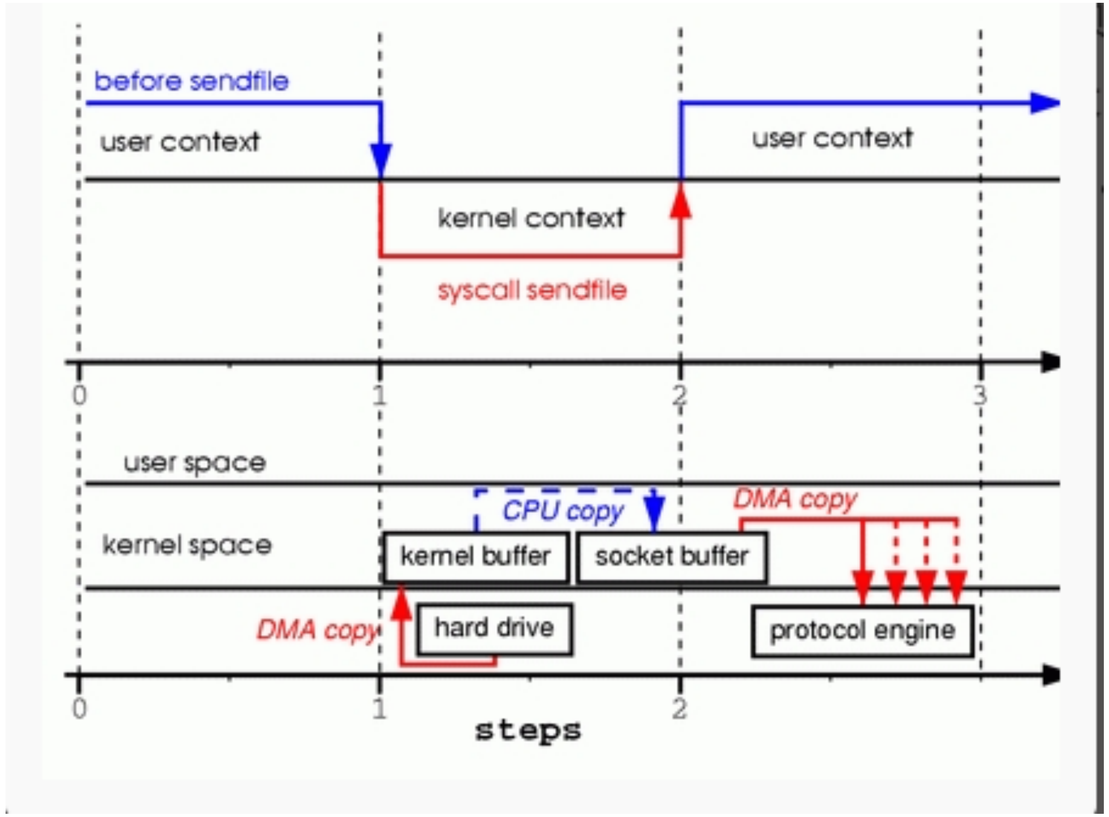



# 插入知识点：零拷贝技术



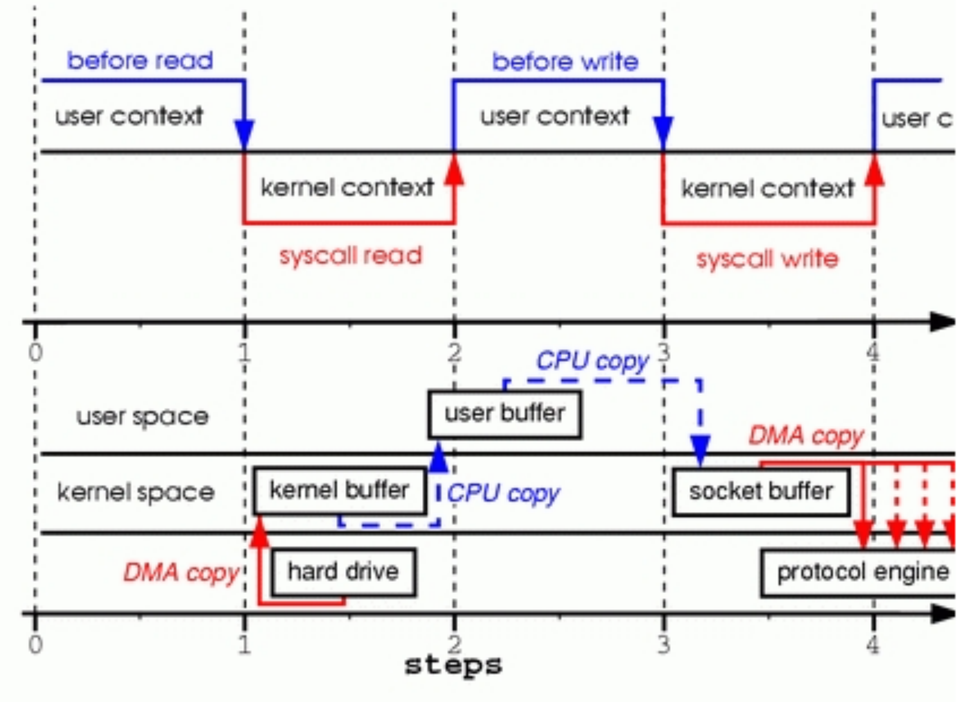

一次看起来还算正常的数据读写过程

四次数据拷贝！！！



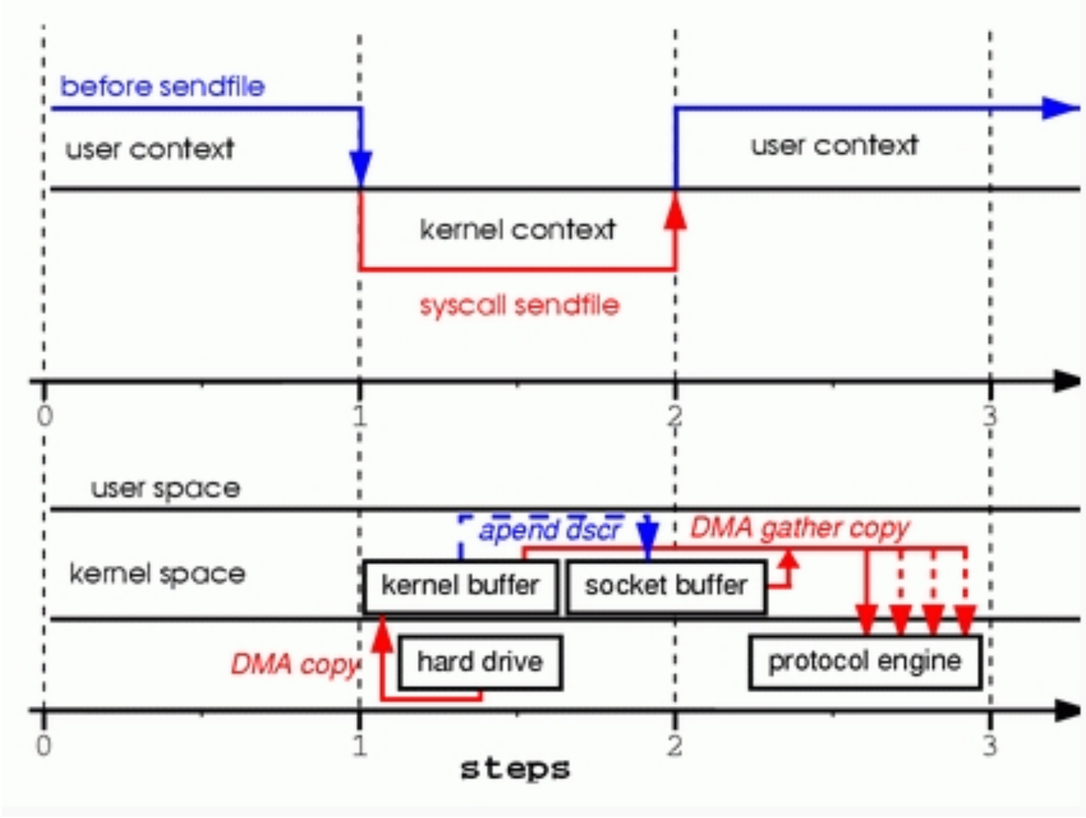

使用 sendfile() 优化IO  三次拷贝！

# 插入知识点：零拷贝技术




一次看起来还算正常的数据读写过程

四次数据拷贝！！！




支持**SCATTER-GATHER**的网卡

两次拷贝！！！

# Things About Aster

介绍：技术的深度决定技术的广度

苦行之旅：无处安放的类型转换

苦行之旅：drop函数与唤醒

苦行之旅：让人头秃的profile

苦行之旅：parser回溯与DC老师的二三事

苦行之旅：我最亲爱的syscall别闹了

插入知识点：零拷贝技术

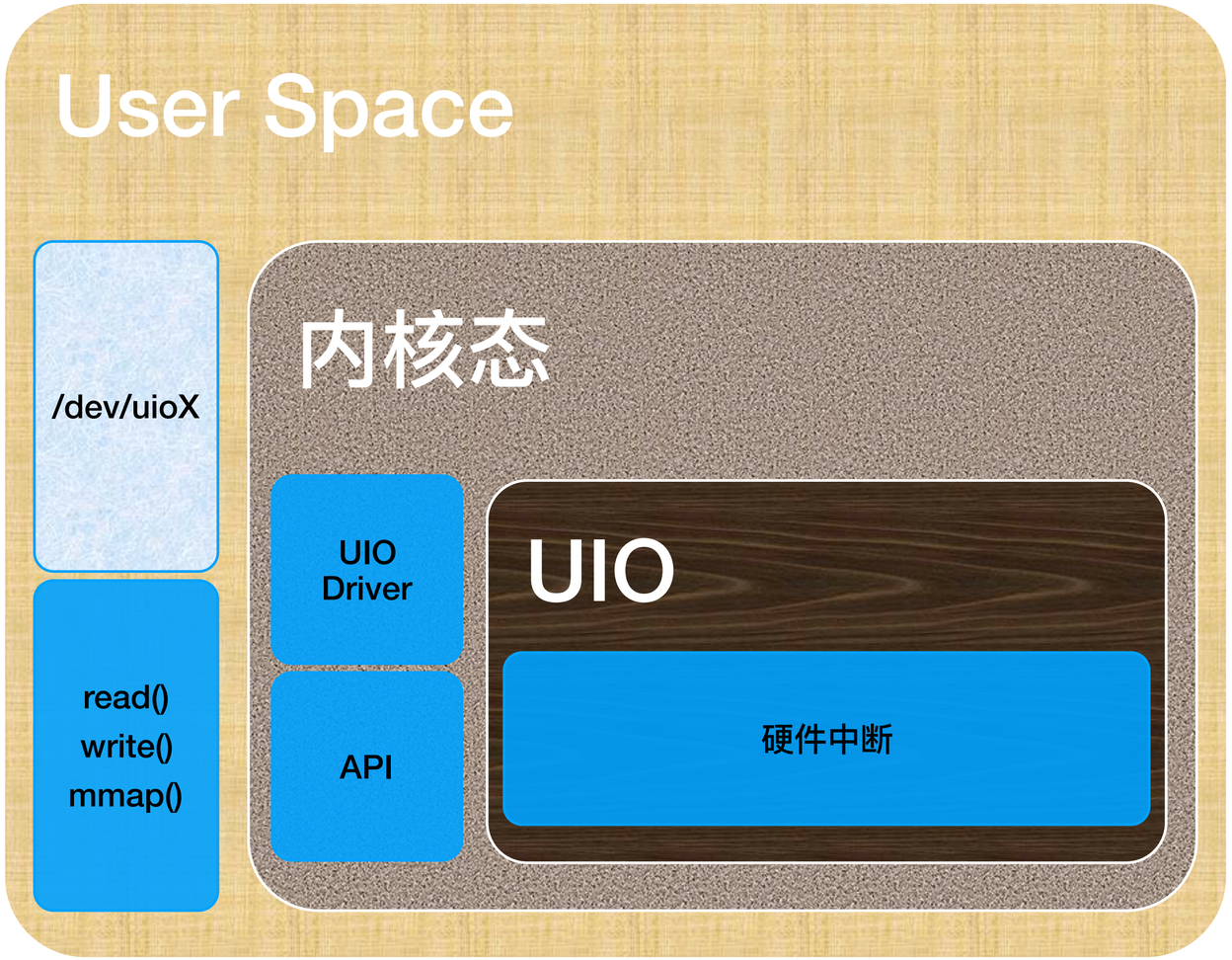
终极解法：DPDK+用户态协议栈

# 终极解法：DPDK+用户态协议栈

DPDK
DATA PLANE DEVELOPMENT KIT

+

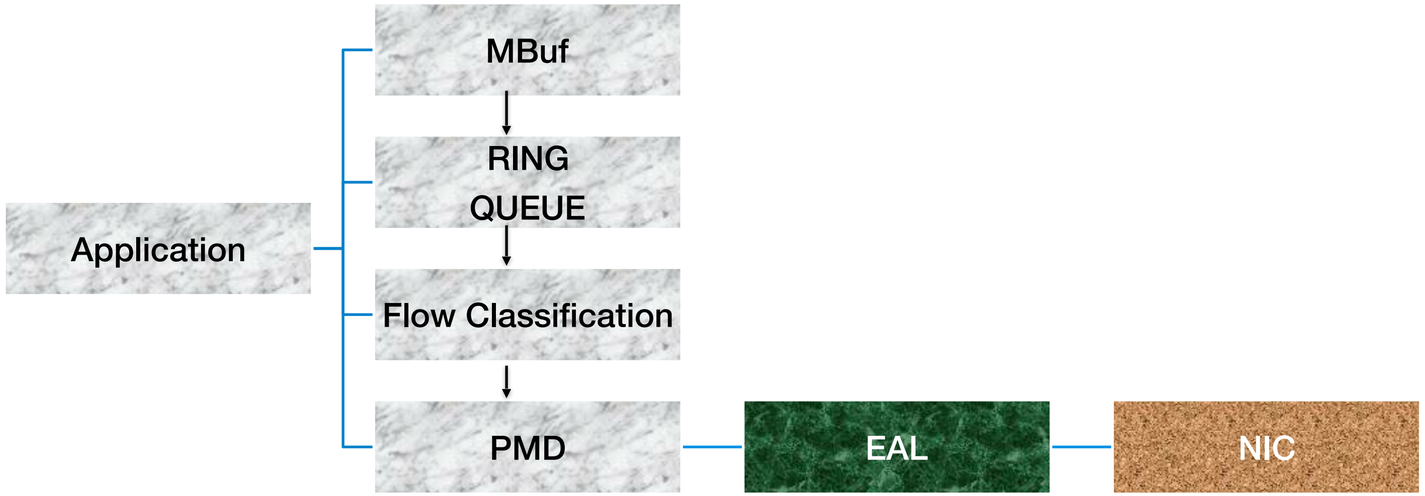
Linux UIO



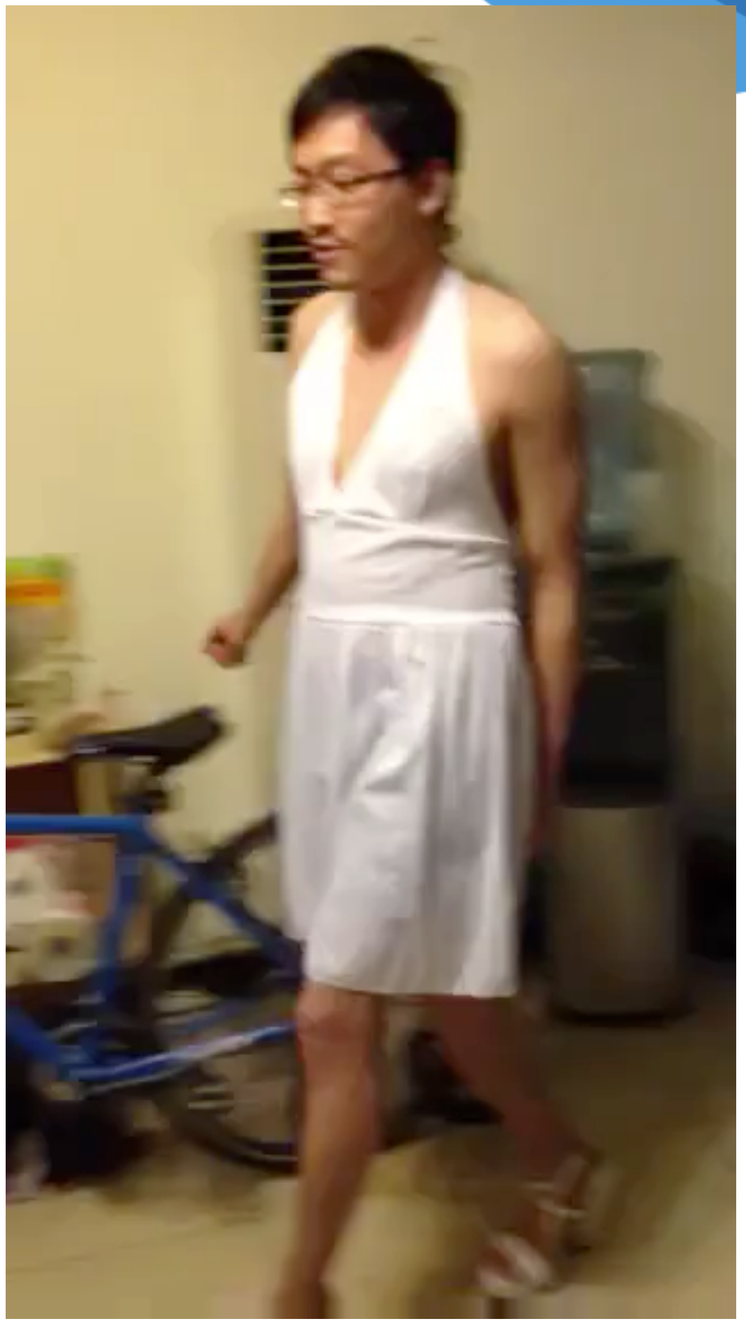
# 终极解法：DPDK+用户态协议栈

DPDK
DATA PLANE DEVELOPMENT KIT

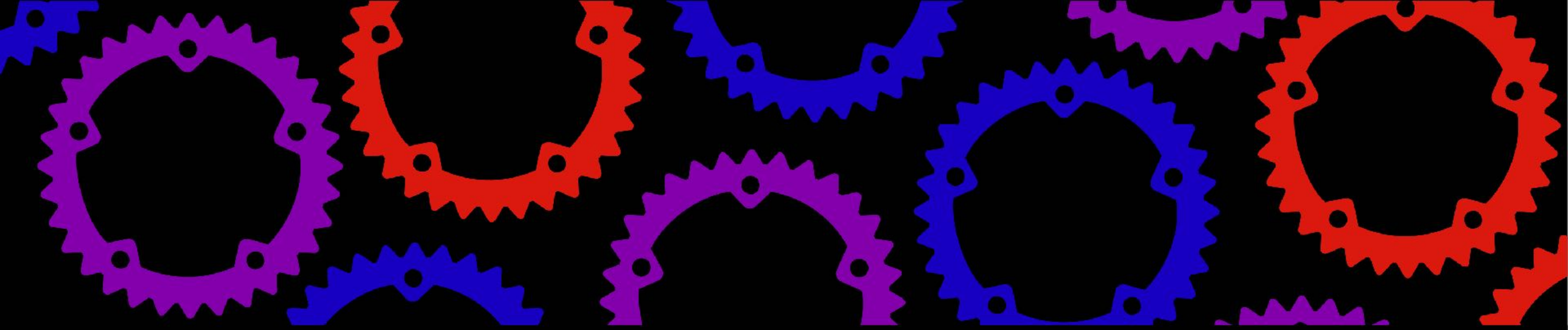
+




Poll Model **Driver**



# 冠位指定



# THANKS