Linux High Performance IO

Боян Кроснов

OpenFest 2021

За мен

- Chief Product Officer и съосновател на StorPool
- 20+ години in ISPs, SDN, SDS
- IT инфраструктура на световно ниво
- Изобретения, производителност и ефективност
- Висока добавена стойност

Boyan Krosnov

https://www.linkedin.com/in/krosnov/
@bkrosnov
bk@storpool.com

StorPool DISTRIBUTED STORAGE

За StorPool

- Българска фирма основана 2011 г. доставяща собствен продукт на глобални клиенти

"StorPool is built from the ground up to ***provide public/private cloud builders with the fastest and most reliable software-defined storage solution on the market!"***

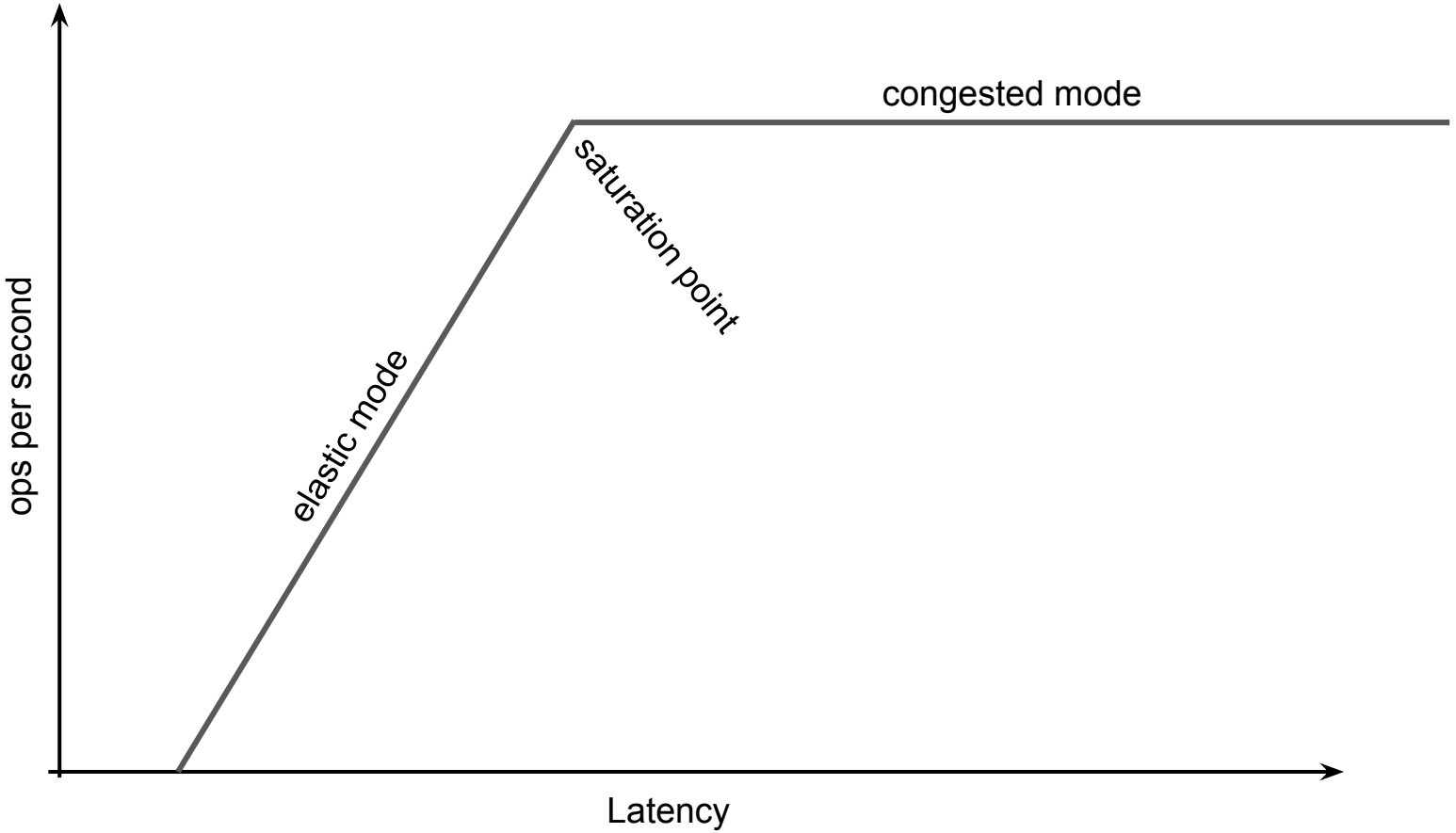
- Фокус върху Modern IT use-cases – KVM, Kubernetes (still supports legacy VMware/Hyper-V/XenServer, etc. stacks
- Considerable scale - Petabytes of data and thousands of servers in production
- Потребител и създател на open source софтуер

StorPool DISTRIBUTED STORAGE

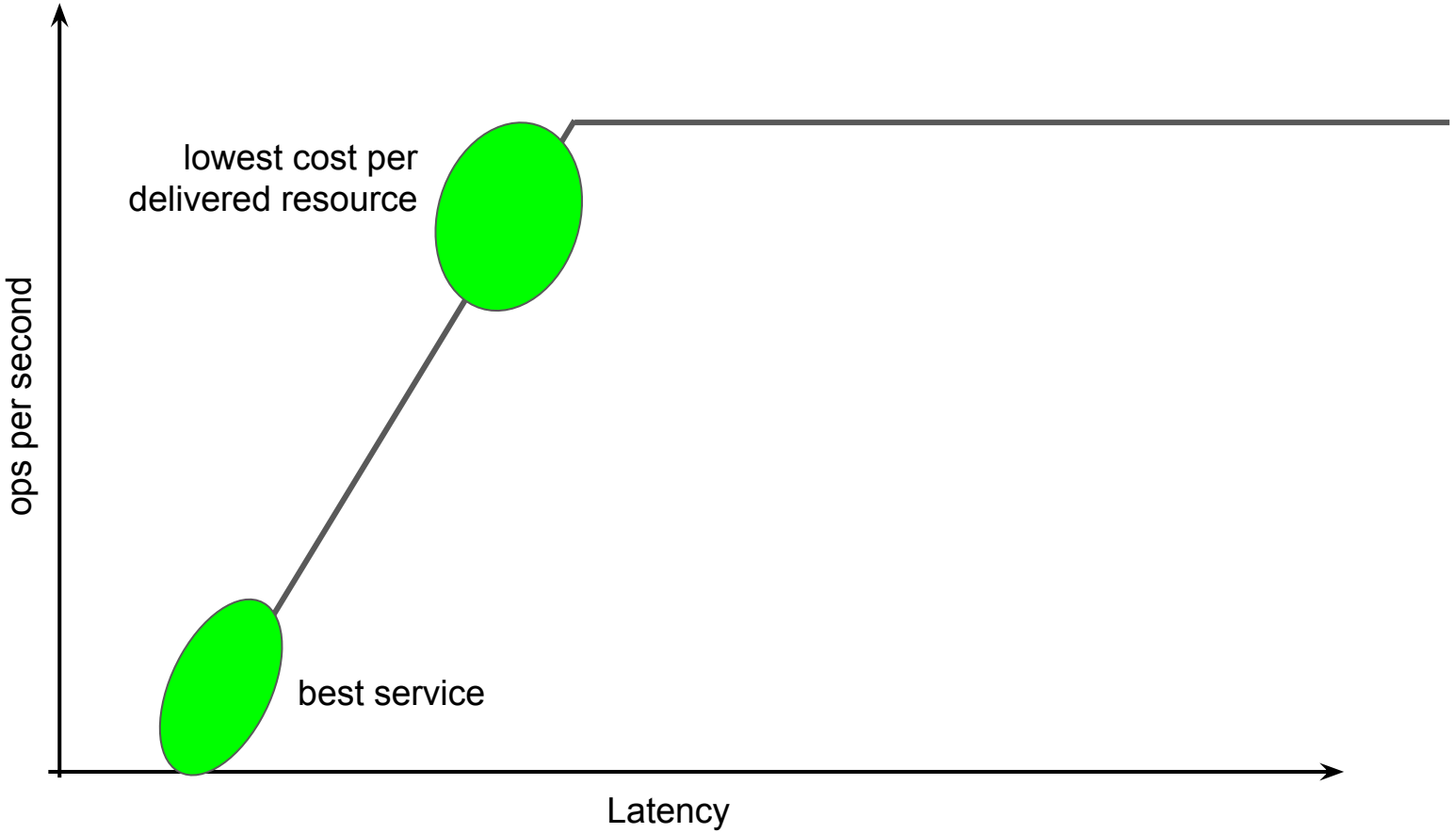
План

1. Understanding performance
2. User vs Kernel
3. Syscalls
4. io_uring
5. Kernel bypass
6. RDMA
7. Заключение

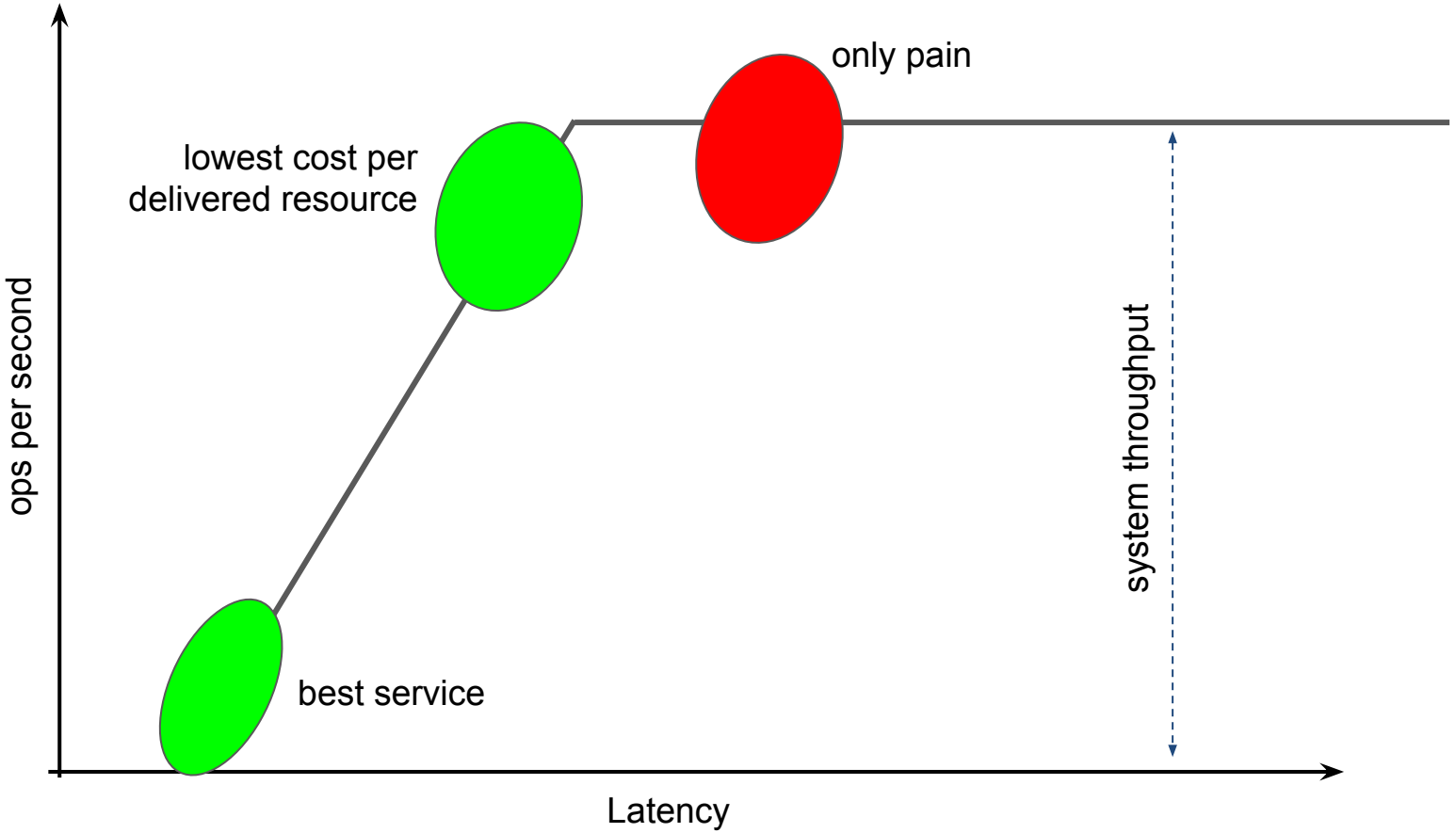
Latency vs throughput



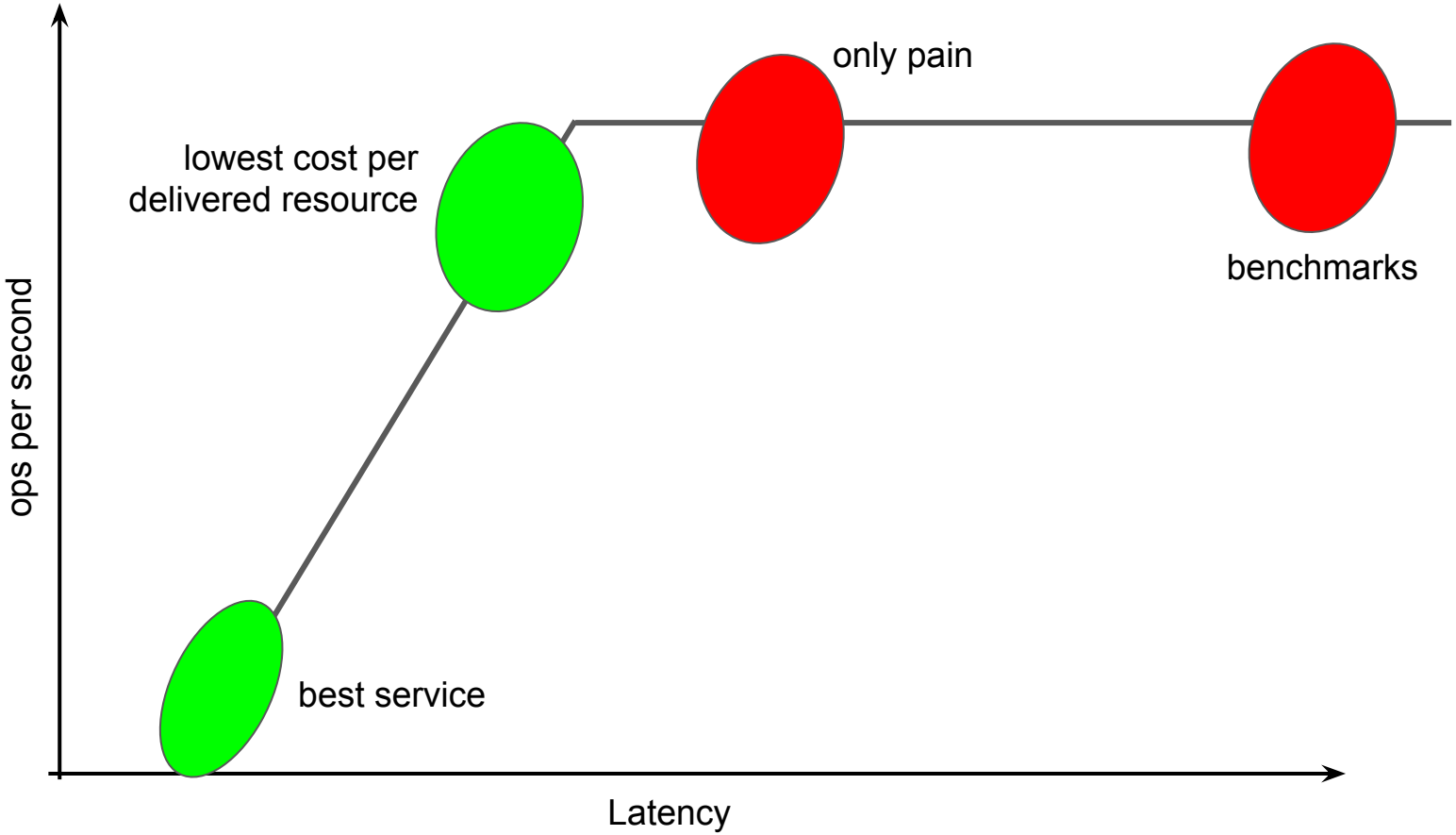
StorPool
DISTRIBUTED STORAGE



StorPool
DISTRIBUTED STORAGE



StorPool
DISTRIBUTED STORAGE



StorPool
DISTRIBUTED STORAGE

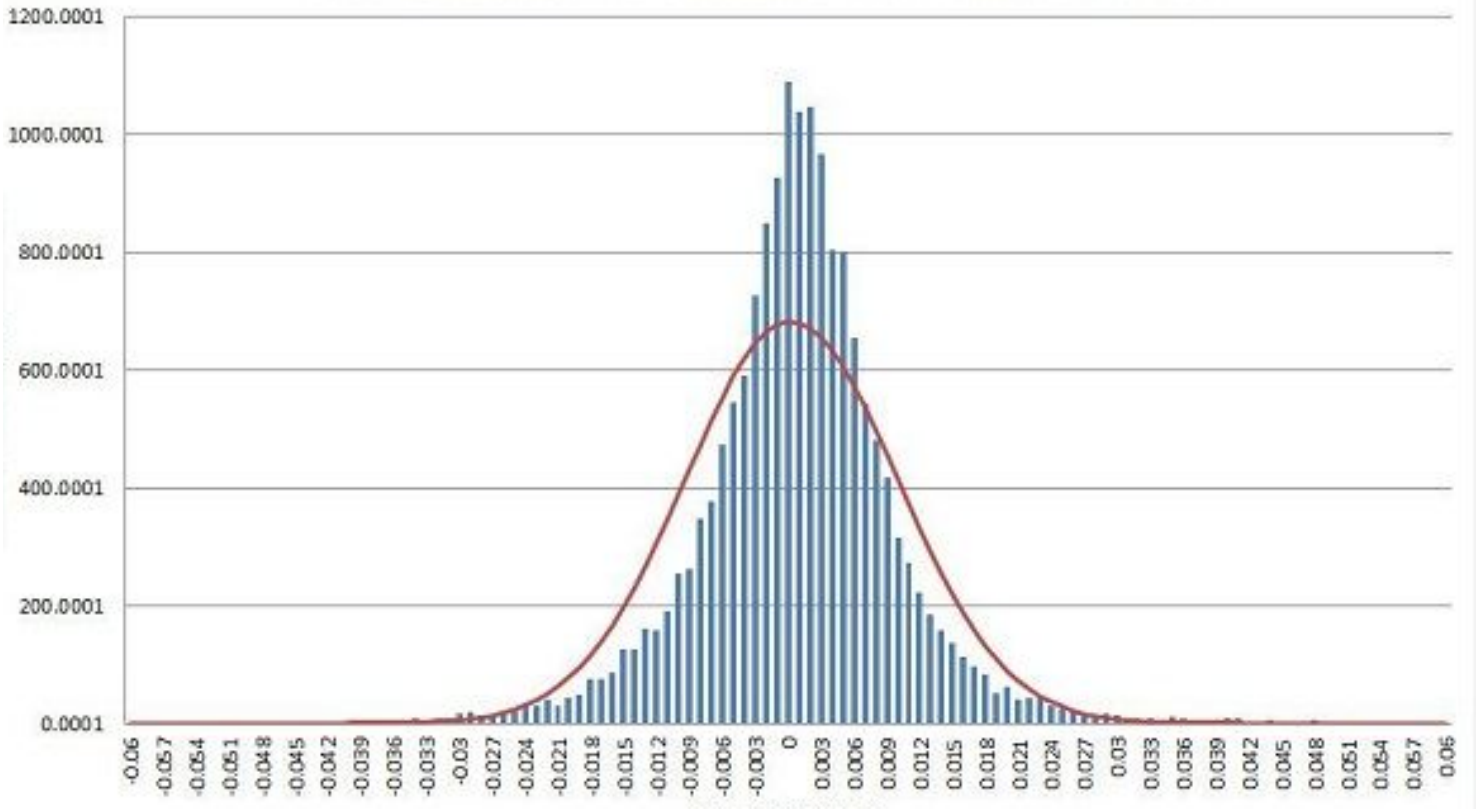
Да мерим вярното нещо

- Системата може ли да поддържа 1000 TPS и как се държи при такова натоварване?

- Колко TPS може да достигне системата и как се държи при максимално натоварване?

- Очевидно различни въпроси!

Статистика



StorPool
DISTRIBUTED STORAGE

Статистика

- 1 проба (sample) не е достатъчна
- средната стойност?
- 99th percentile?
- зърнена статистика™ - "гледам числата и ми изглежда, че е около 100" - not ok

Complex systems

- Ако знаем как се държи една система, дали можем да предвидим как ще се държи друга система, която е "подобна"?
  - Ако сменим мрежата с по-бърза какво ще стане?
  - Ако сменим истинското приложие и load със synthetic benchmark, който прави нещо подобно

- В общия случай не!
- В частни случаи да с "изследване на пространството"

План

1. Understanding performance
2. **User vs Kernel**
3. Syscalls
4. io_uring
5. Kernel bypass
6. RDMA
7. Заключение

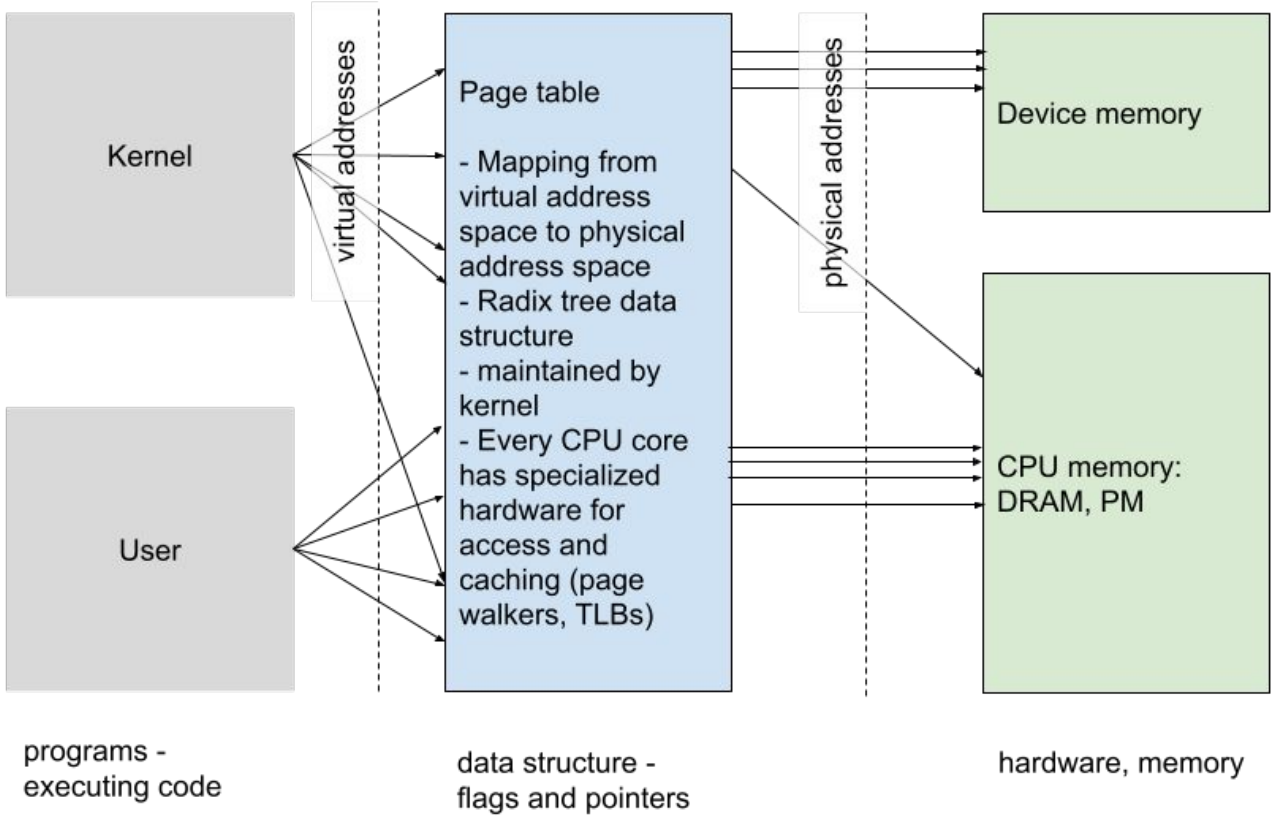
Scheduling & preemption

- Всеки потребителски процес се изпълнява на ядро временно.
- Може по всяко време ядрото да ни трябва за друго и да спрем потребителския процес (preemption, migration)
- Може процеса да влезе в блокиращ call (спане или изчакване на IO).
- Може kernel-а да прецени че има да свърши някаква background работа на същото ядро. (softirq)
- Бавно и скъпо. За да се избегне:

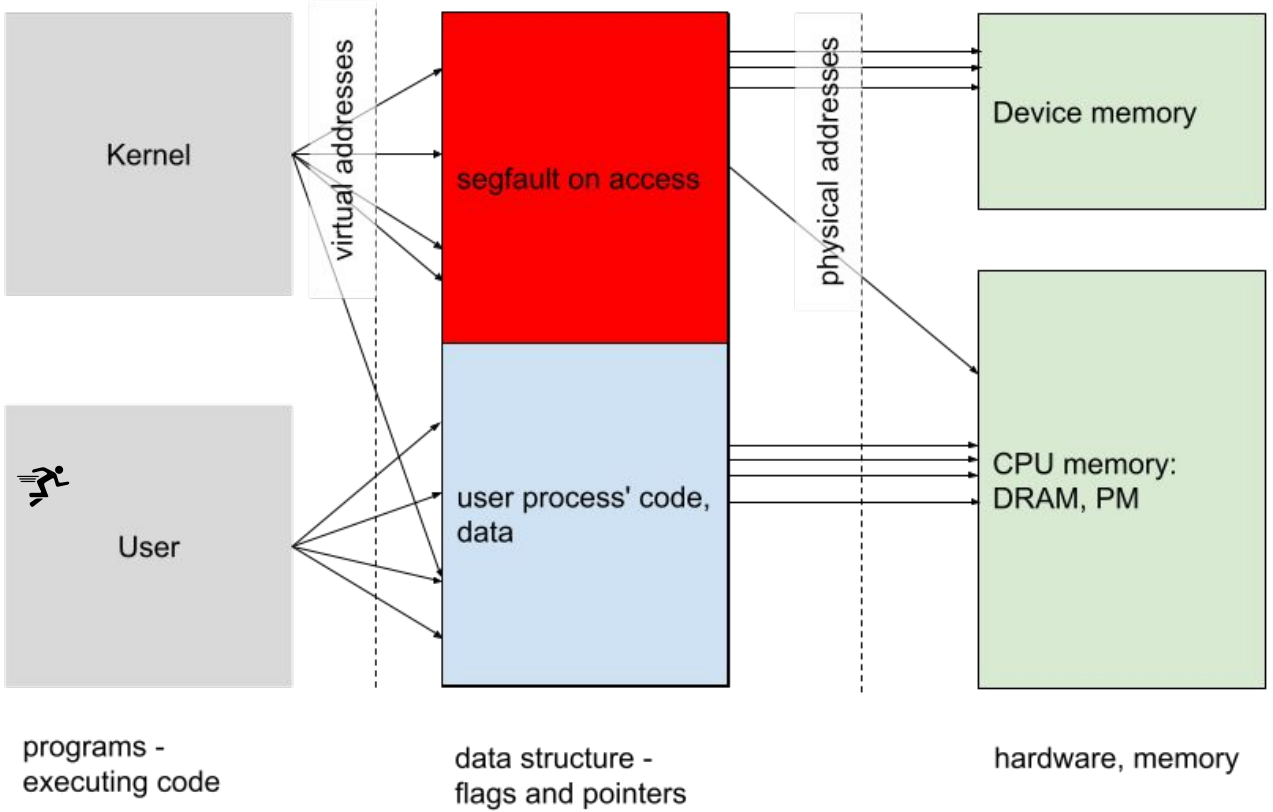
  - трябва да имаме по-малко процеси/нишки отколкото ядра - cpu pinning, event-driven worker architecture

  - трябва да не спим и да не влизаме в блокиращи (синхронни) calls - polling, async IO

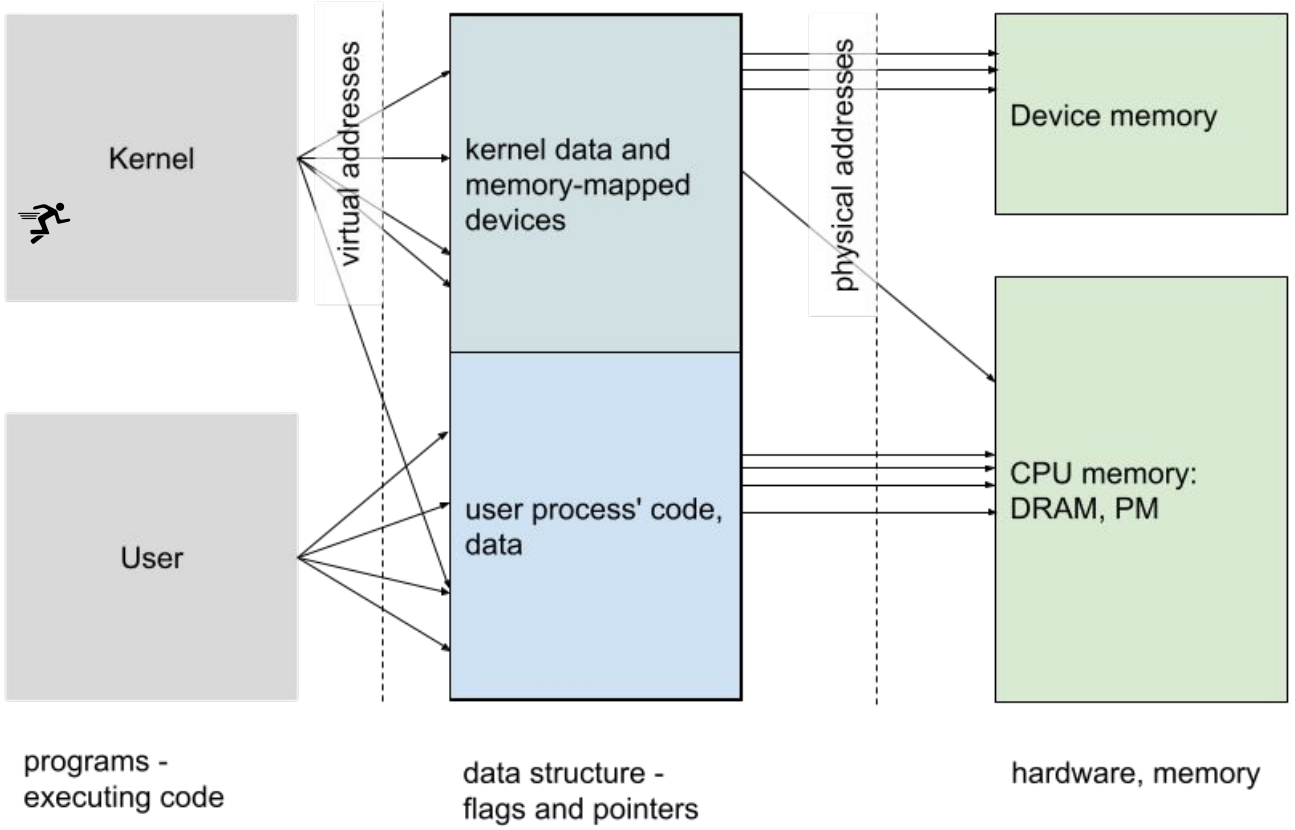
Kernel space vs user space



Kernel space vs user space



Kernel space vs user space



План

1. Understanding performance
2. User vs Kernel
3. Syscalls
4. io_uring
5. Kernel bypass
6. RDMA
7. Заключение

Ping

```
[root@cn1 ~]# ping -s 1000 10.5.41.12
PING 10.5.41.12 (10.5.41.12) 1000(1028) bytes of data.
1008 bytes from 10.5.41.12: icmp_seq=1 ttl=64 time=0.135 ms
1008 bytes from 10.5.41.12: icmp_seq=2 ttl=64 time=0.116 ms
1008 bytes from 10.5.41.12: icmp_seq=3 ttl=64 time=0.119 ms
1008 bytes from 10.5.41.12: icmp_seq=4 ttl=64 time=0.115 ms
1008 bytes from 10.5.41.12: icmp_seq=5 ttl=64 time=0.112 ms

--- 10.5.41.12 ping statistics ---

5 packets transmitted, 5 received, 0% packet loss, time 3999ms

rtt min/avg/max/mdev = 0.112/0.119/0.135/0.012 ms
```

119 µs

StorPool DISTRIBUTED STORAGE

Syscalls

```
09:32:51.031128 socket(AF_INET, SOCK_RAW, IPPROTO_ICMP) = 3

09:32:51.032082 sendto(3, "\10\0D\307a"..., 1008, 0,
{sa_family=AF_INET, sin_port=htons(0),
sin_addr=inet_addr("10.5.41.12")}, 16) = 1008

09:32:51.032157 recvmsg(3, {msg_name={sa_family=AF_INET,
sin_port=htons(0), sin_addr=inet_addr("10.5.41.12")},
msg_namelen=128->16, ...
```


StorPool DISTRIBUTED STORAGE

Ping с по-малко приспиване и събуждане

```
[root@cn1 ~]# ping -f -q -c 1000 -s 1000 10.5.41.12
PING 10.5.41.12 (10.5.41.12) 1000(1028) bytes of data.

--- 10.5.41.12 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time 92ms
rtt min/avg/max/mdev = 0.080/0.081/0.124/0.010 ms, ipg/ewma 0.092/0.082 ms
```

81 µs

StorPool DISTRIBUTED STORAGE

План

1. Understanding performance
2. User vs Kernel
3. Syscalls
4. io_uring
5. Kernel bypass
6. RDMA
7. Заключение

Beyond syscalls

Alice: Hey Bob, we have the most important development in practical OS architecture since syscalls in the 1970s!

Bob: What did you call it?

Alice: "io_uring". Obviously. Because it is for IO, and it is a ring, and it is for talking to userspace.

Nobody ever: That's a good name. 👍👍

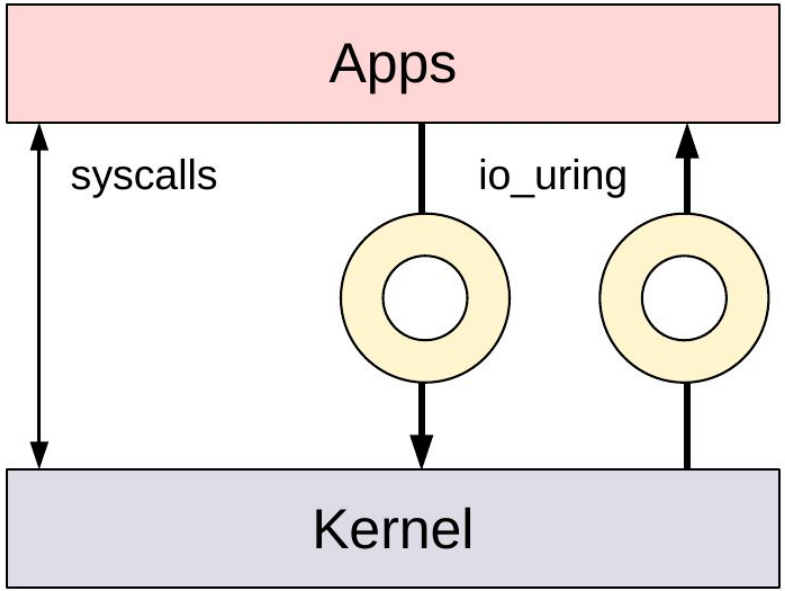
io_uring

- Modern hardware-alike and hardware-friendly user-kernel interface
- Since Linux Kernel 5.0 (Mar 2019)
- Alternative to syscalls
- Shared memory between user and kernel
  - submission queue (ring buffer)
  - completion queue (ring buffer)

io_uring

Faster syscalls using shared ring buffers

- Send and completion ring buffers
- Allows I/O to be batched and async
- Primary use cases network and disk I/O



io_uring OPs

```
IORING_OP_NOP,
IORING_OP_READV,
IORING_OP_WRITEV,
IORING_OP_FSYNC,
IORING_OP_READ_FIXED,
IORING_OP_WRITE_FIXED,
IORING_OP_POLL_ADD,
IORING_OP_POLL_REMOVE,
IORING_OP_SYNC_FILE_RANGE,
IORING_OP_SENDMSG,
IORING_OP_RECVMSG,
IORING_OP_TIMEOUT,
IORING_OP_TIMEOUT_REMOVE,
IORING_OP_ACCEPT,
IORING_OP_ASYNC_CANCEL,
IORING_OP_LINK_TIMEOUT,
IORING_OP_CONNECT,
IORING_OP_FALLOCATE,
```

```
IORING_OP_OPENAT,
IORING_OP_CLOSE,
IORING_OP_FILES_UPDATE,
IORING_OP_STATX,
IORING_OP_READ,
IORING_OP_WRITE,
IORING_OP_FADVISE,
IORING_OP_MADVISE,
IORING_OP_SEND,
IORING_OP_RECV,
IORING_OP_OPENAT2,
IORING_OP_EPOLL_CTL,
IORING_OP_SPLICE,
IORING_OP_PROVIDE_BUFFERS,
IORING_OP_REMOVE_BUFFERS,
IORING_OP_TEE,
IORING_OP_SHUTDOWN,
IORING_OP_RENAMEAT,
IORING_OP_UNLINKAT,
```

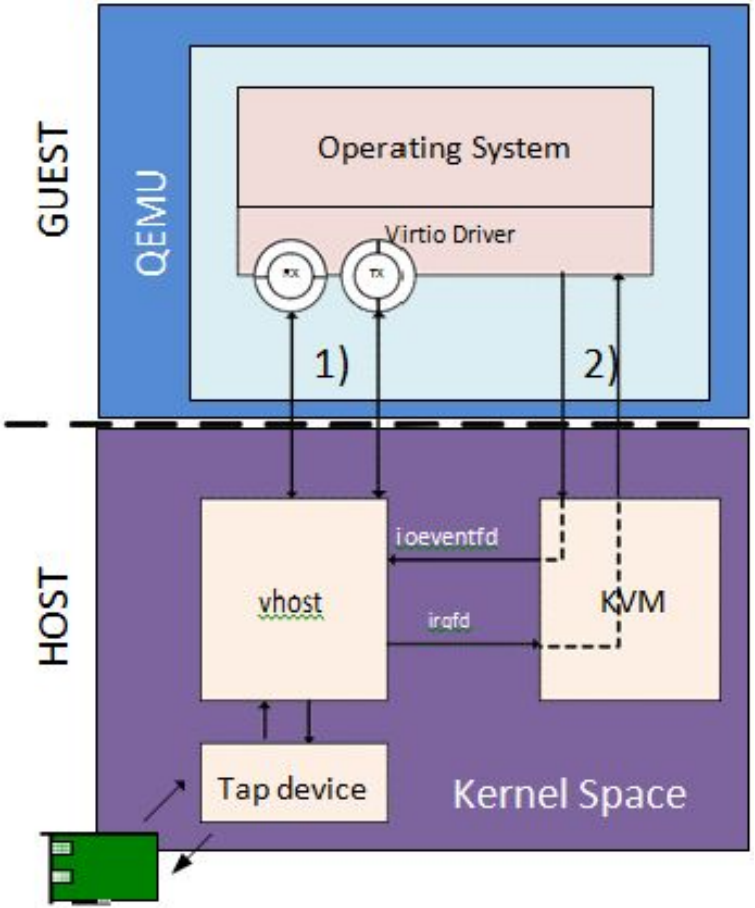
io_uring

- Modern hardware-alike and hardware-friendly user-kernel interface

… and more:

- Linked commands
- Restrictions
- SQ/CQ polling
- io_context polling
- Registered buffers, registered file descriptors
- Eventfd integration

Подобно на vhost, но общо-приложимо

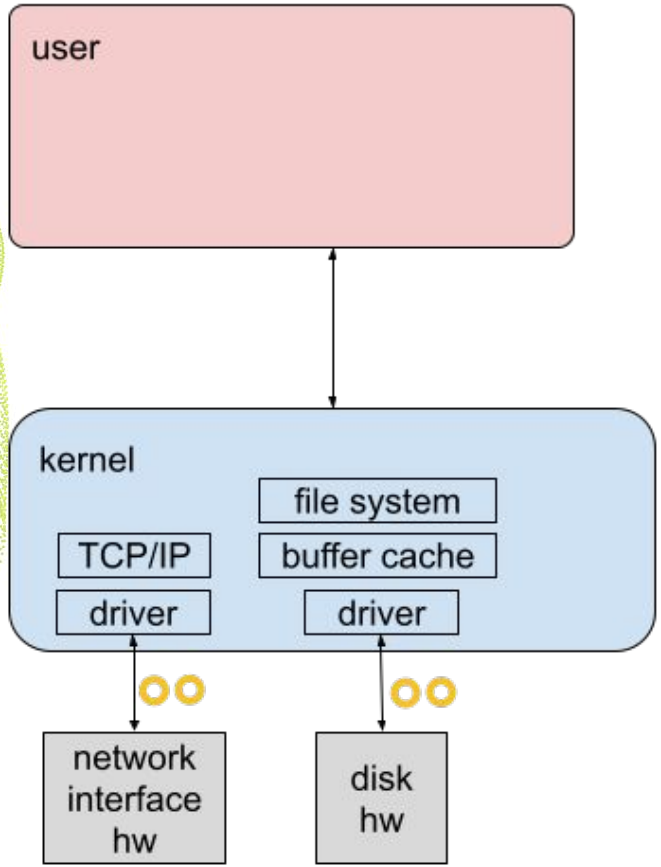


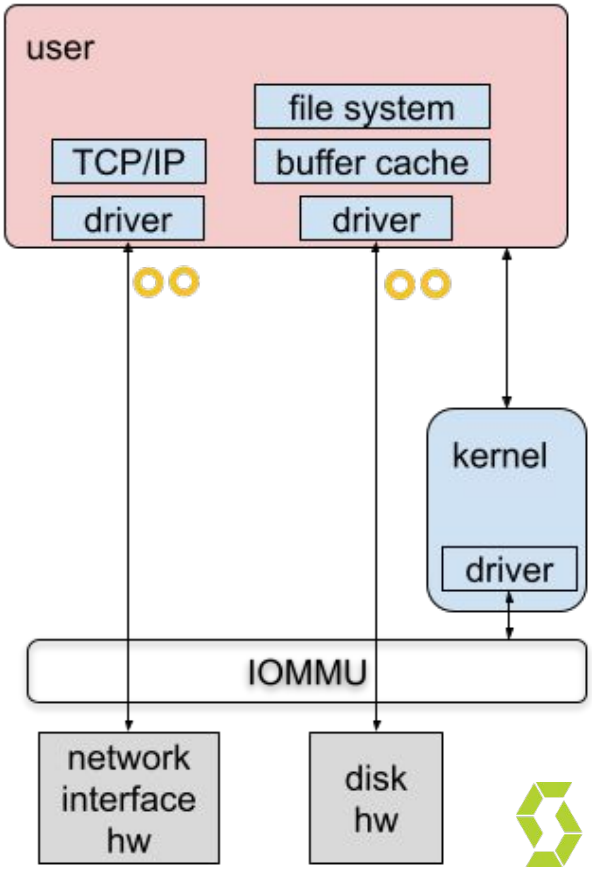
StorPool
DISTRIBUTED STORAGE

План

1. Understanding performance
2. User vs Kernel
3. Syscalls
4. io_uring
5. **Kernel bypass**
6. RDMA
7. Заключение

Kernel bypass





StorPool
DISTRIBUTED STORAGE

Kernel bypass

- местим драйверите, мрежов стек, файлова система, т.н. в user space
- безопасност/сигурност/изолация с фиксирана памет (pinned, unswappable, huge pages), IOMMU, PCI ACS
- Hardware virtualization, SR-IOV, VFs
- vfio driver
- DPDK, SPDK, FD.io, Snabb, other frameworks

- Example: OVS-DPDK (VM networking)

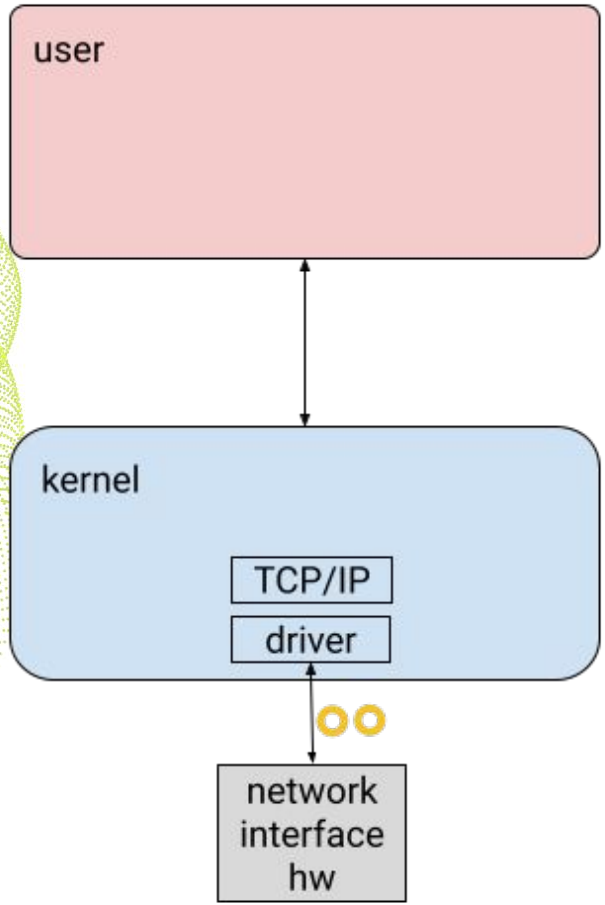
Kernel bypass

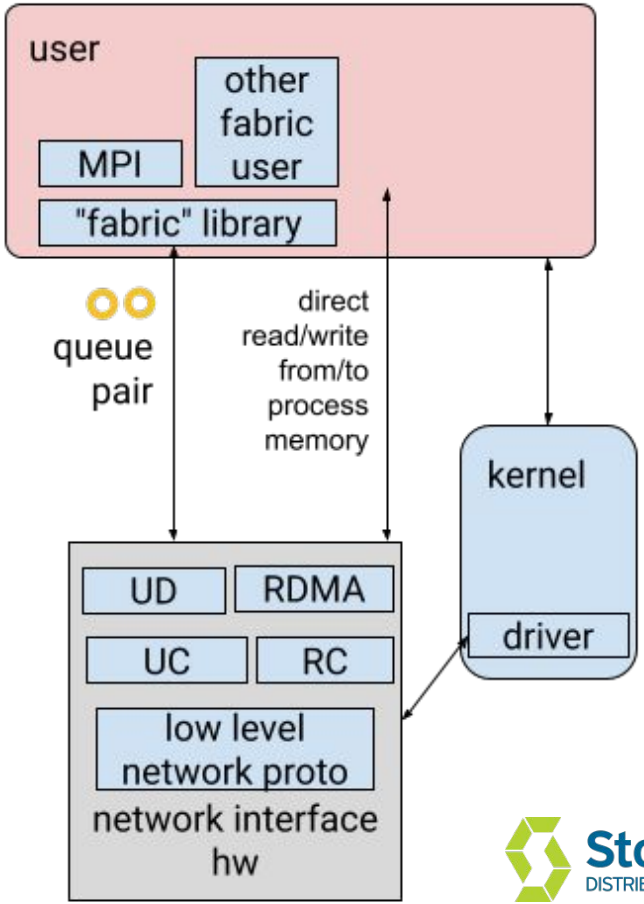
```
[root@cn1 ~]# /usr/lib/storpool/storpool_ping -s 1000 c12
net0/0/0: reply from 32780: 0.022 ms
net1/1/1: reply from 32780: 0.027 ms
net0/0/0: reply from 32780: 0.015 ms
net1/1/1: reply from 32780: 0.019 ms
net0/0/0: reply from 32780: 0.017 ms
net1/1/1: reply from 32780: 0.021 ms
net0/0/0: reply from 32780: 0.016 ms
net1/1/1: reply from 32780: 0.019 ms
net0/0/0: reply from 32780: 0.015 ms
net1/1/1: reply from 32780: 0.018 ms
net0/0/0: reply from 32780: 0.016 ms
net1/1/1: reply from 32780: 0.020 ms
^C
--- ping stats ---
net0: 6 packets transmitted, 6 received, 0% packet loss
net0: rtt min/avg/max 0.015/0.016/0.022 ms
net1: 6 packets transmitted, 6 received, 0% packet loss
net1: rtt min/avg/max 0.018/0.020/0.027 ms
```

План

1. Understanding performance
2. User vs Kernel
3. Syscalls
4. io_uring
5. Kernel bypass
6. RDMA
7. Заключение

RDMA





StorPool
DISTRIBUTED STORAGE

RDMA

- местим мрежовия стек в картата (NIC, HCA)
- даваме на всеки процес отделна двойка опашки (QP) - send queue, receive queue, за команди и събития от/към мрежовия стек
- сигурност/изолация с фиксирана памет, регистрирана в картата

- Infiniband, RoCEv2, Omni-Path™, Amazon EFA™
- OpenFabrics Aliance, OFED, OFI

- Example: OVS-DPDK (VM networking)

План

1. Understanding performance
2. User vs Kernel
3. Syscalls
4. io_uring
5. Kernel bypass
6. RDMA
7. **Заключение**

Кое кога

- io_uring - повечето приложения. Всеки, който в момента ползва async io под Linux.
- Kernel bypass - редки и специализирани случаи, когато има полза от заобикалянето на kernel-ския мрежов стек и файлова система. Напр. StorPool.
- RDMA - HPC clusters? AI/ML use-cases, especially w/ GPUs?

Благодарности и въпроси!

https://storpool.com/force