# News from academia: FatELF, RDMA and CRIU

Mike Rapoport
<rppt@linux.ibm.com>

Joel Nider
<joeln@il.ibm.com>



This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 688386.

# Heterogeneous computing

- FPGA and GPGPU acceleration
- Hot research topic
  - Heterogeneous ISA multicore chips
  - Heterogeneous ISA single system cluster

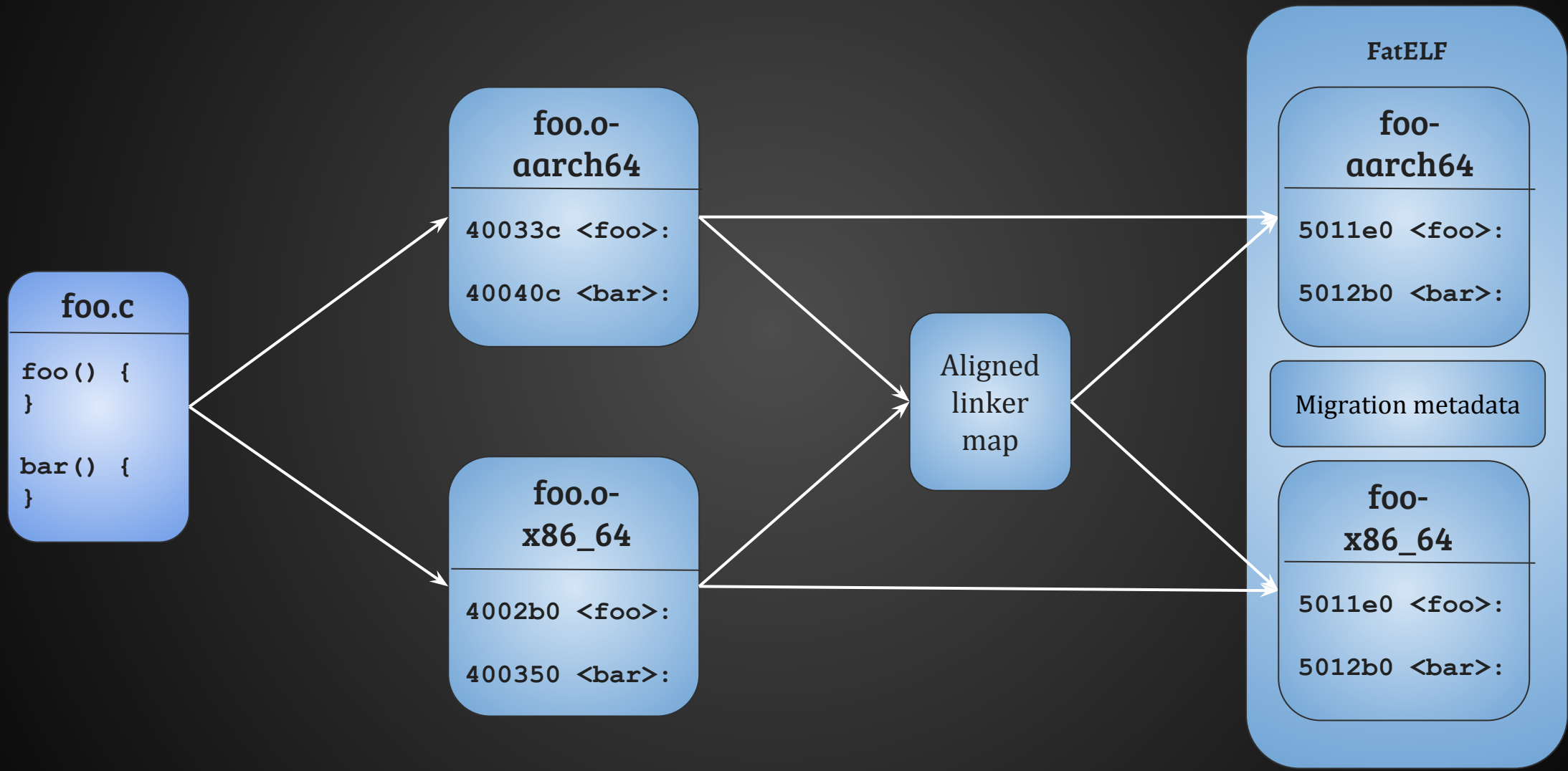
# Heterogeneity and power efficiency

- Low datacenter utilization
- Power aware workload placement and load balancing
- Cross-architecture container migration


# Vision

- Aligned binaries
  - Symbols at the same addresses
  - Objects have identical size
  - Metadata for stack transformation
- Post-copy memory migration
- RDMA for lower page fault latency

# Aligned binaries



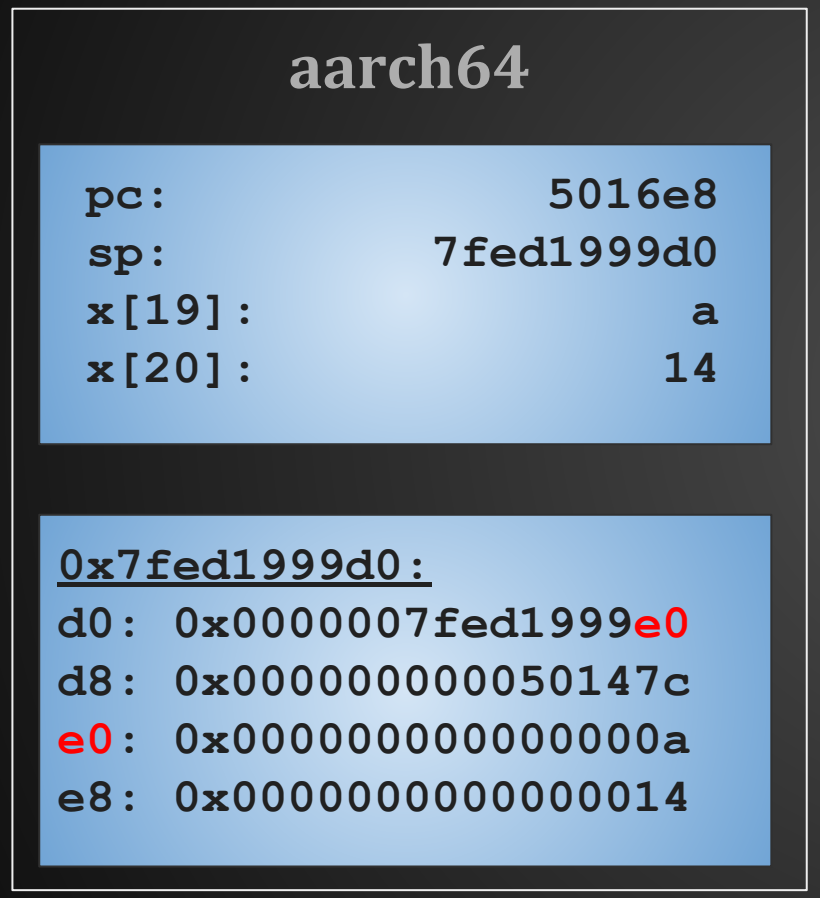
# Migration metadata

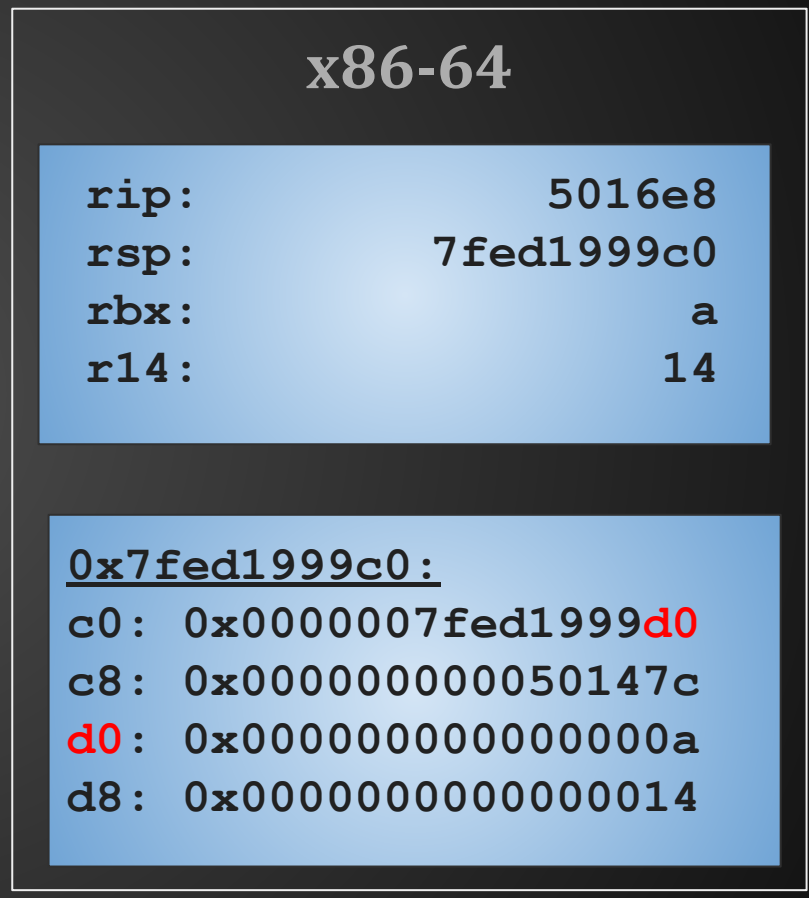
- Migration points
  - Function entry and exit
  - May be any basic block entry
- Live values info
  - Register contents and mapping
  - Stack locations

# CRIU modifications

- Ensure dump stops the task at a “migration point”
  - Insert breakpoints when the task is stopped
  - Continue the task until it hits a breakpoint
- Make restore less strict
  - Allow `core*img` from different architecture
  - Allow different executable size
  - Always use target vDSO
- Extend thread core info with target architecture bits
- Add stack and registers transformation

# Stack transformation



# Migrating simple application

```c
void f3(int a, int b) {
    printf("%s: a: %d, b: %d\n", __func__, a, b);
}

void f2(int a, int b) {
    printf("%s: a: %d, b: %d\n", __func__, a, b);
    usleep((rand() % 10000) * 50);
    f3(a * 2, b * 2);
}

void f1(int a, int b) {
    printf("%s: a: %d, b: %d\n", __func__, a, b);
    usleep((rand() % 10000) * 50);
    f2(a * 2, b * 2);
}

int main(int argc, char *argv[]) {
    int a = 10, b = 20;

    srand(time(NULL));

    for (;;) {
      f1(a, b);
      usleep((rand() % 10000) * 50);
    }

    return 0;
}
```

- Demo

https://asciinema.org/a/Wuojd8R6kWIDs5LXwN5IAyqHP

```
f4: a: 80, b: 160
f1: a: 10, b: 20
f2: a: 20, b: 40
f3: a: 40, b: 80
f4: a: 80, b: 160
f1: a: 10, b: 20
f2: a: 20, b: 40
f3: a: 40, b: 80
f4: a: 80, b: 160
f1: a: 10, b: 20
f2: a: 20, b: 40
f3: a: 40, b: 80
f4: a: 80, b: 160
f1: a: 10, b: 20
f2: a: 20, b: 40
f3: a: 40, b: 80
f4: a: 80, b: 160
Killed
[root@arm64 app]#
```

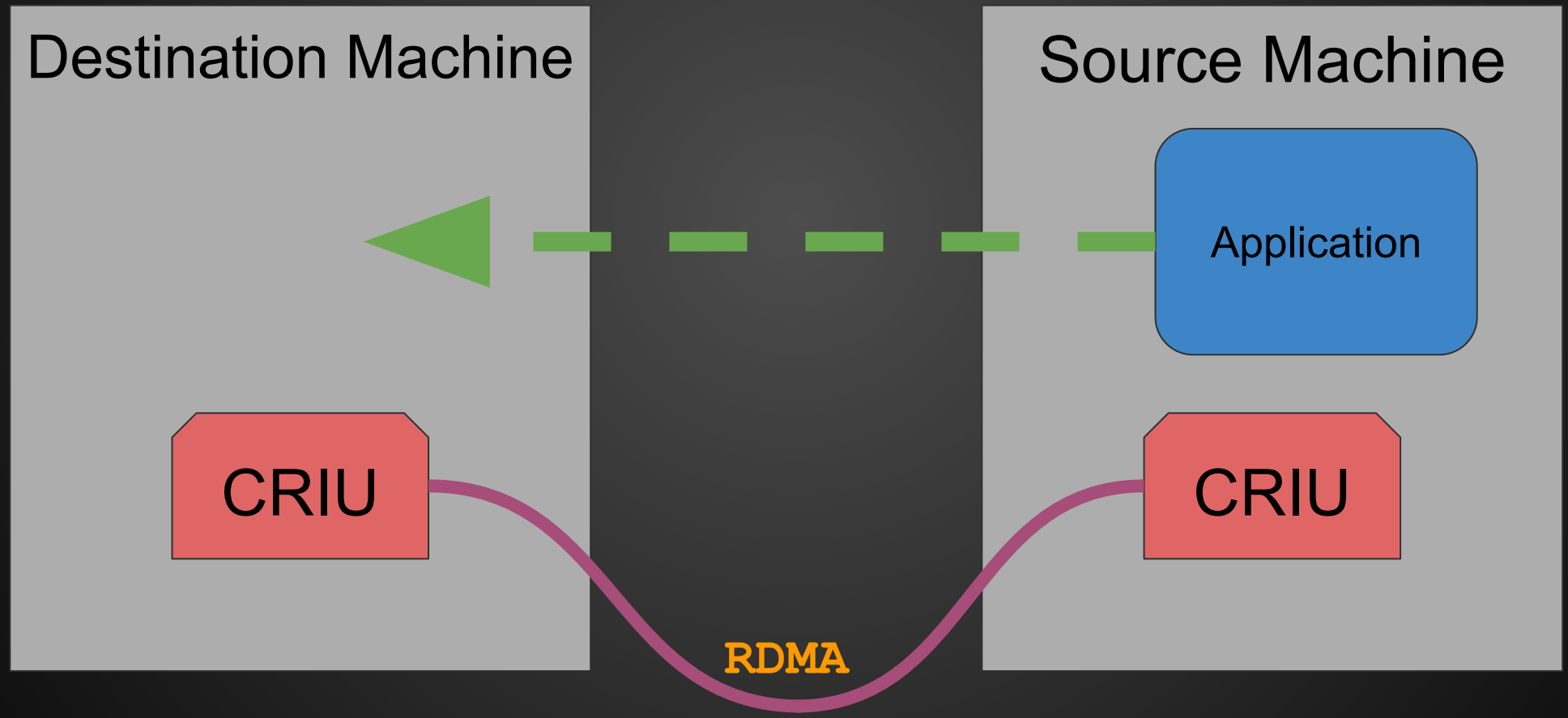
```
Byte Order:          Little Endian
CPU(s):              2
On-line CPU(s) list: 0,1
Thread(s) per core:  1
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
Model:               0
BogoMIPS:            125.00
NUMA node0 CPU(s):   0,1
Flags:               fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
[root@arm64 ~]# /root/git/criu/criu/criu dump -D /home/cr/img/ -v4 -t $(pidof
 app) -o /tmp/dump.log --breakpoints /home/cr/app/breaks.txt --shell-job && e
cho OK
files stat: fs/file-max 192128, fs/nr_open 1048576
rlimit: RLIMIT_NOFILE unlimited for self
Loaded kdat cache from /run/criu.kdat
OK
[root@arm64 ~]#
```

```
rppt@aquarius:~/git/criu$ lscpu | less
rppt@aquarius:~/git/criu$ sudo ./criu/criu restore -v5 -o /tmp/rst.log --shell-job --cross-arch --cross-arch-src /home/cr/a
pp/app-aarch64 --cross-arch-dst /home/cr/app/app-x86-64 -D /home/cr/img
Turn cross arch C/R ON
files stat: fs/file-max 13952158, fs/nr_open 1048576
rlimit: RLIMIT_NOFILE unlimited for self
Loaded kdat cache from /run/criu.kdat
f1: a: 10, b: 20
f2: a: 20, b: 40
f3: a: 40, b: 80
f4: a: 80, b: 160
f1: a: 10, b: 20
f2: a: 20, b: 40
f3: a: 40, b: 80
f4: a: 80, b: 160
```

```
[0] 0:ssh*                                                    "arm64.localdomain" 08:36 05-Nov-18
```

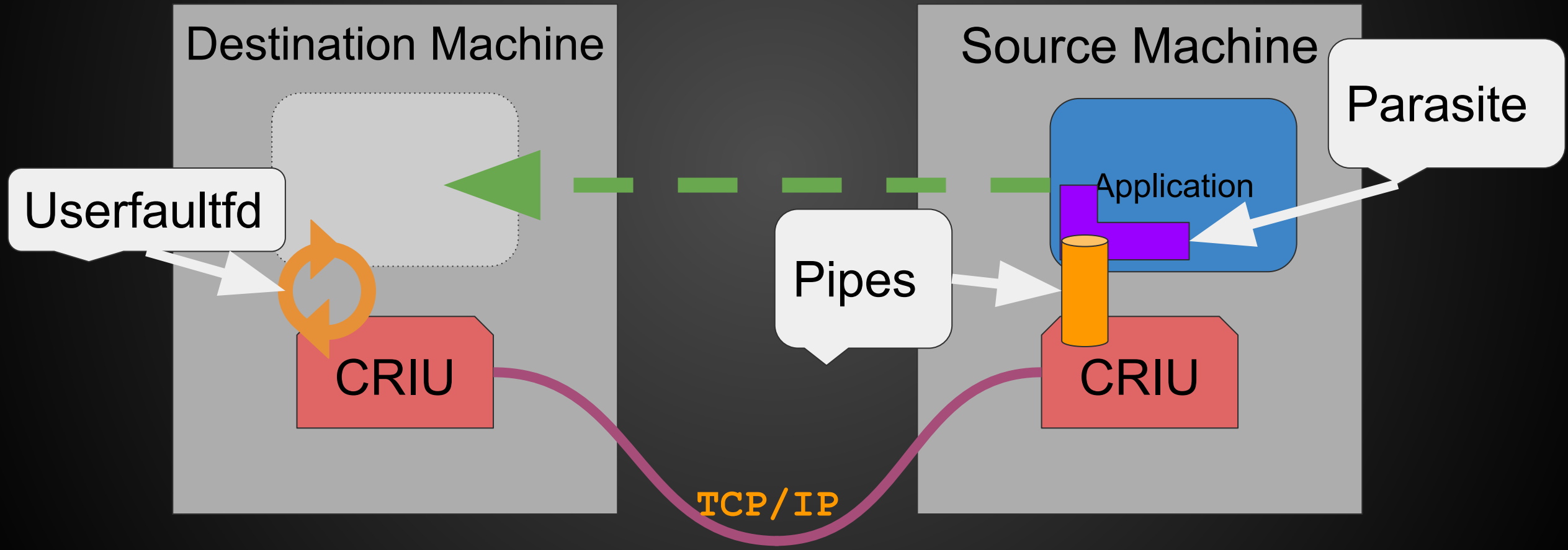
# RDMA - registering memory problem

I want to migrate an application container using post-copy



# RDMA - registering memory problem

How does it work today with TCP/IP?



# RDMA - Requirements

- We would like to avoid the pipes
  - RDMA can remotely access memory directly from application
- All logic must be in CRIU
  - The application should not have to support migration

# RDMA - registering memory problem [2]

- CRIU establishes the RDMA connection - OK!
- CRIU tries to register the memory region on behalf of the migrating process - ???

# RDMA - registering memory problem [3]

## Option 1: Stuff OFED into the parasite

- Parasite is PIE code
- OFED + user mode driver is huge
- Bad combination

# RDMA - registering memory problem [4]

## Option 2: Teach libibverbs to register memory for another process

- Add new function ibv_reg_remote_mr (include/infiniband/verbs.h)

```
mr = ibv_reg_remote_mr(0, 0, (size_t)~(0),
IBV_ACCESS_LOCAL_WRITE|IBV_ACCESS_REMOTE_READ|IBV_ACCESS_ON_DEMAND, pid);
```

- Now you can pass any pid to ~~steal~~ read its memory
- May pose a threat to security

# References

- A measurement study of server utilization in public clouds
- Harnessing ISA diversity: design of a heterogeneous-ISA chip multiprocessor
- Popcorn Linux

# Thank you!