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

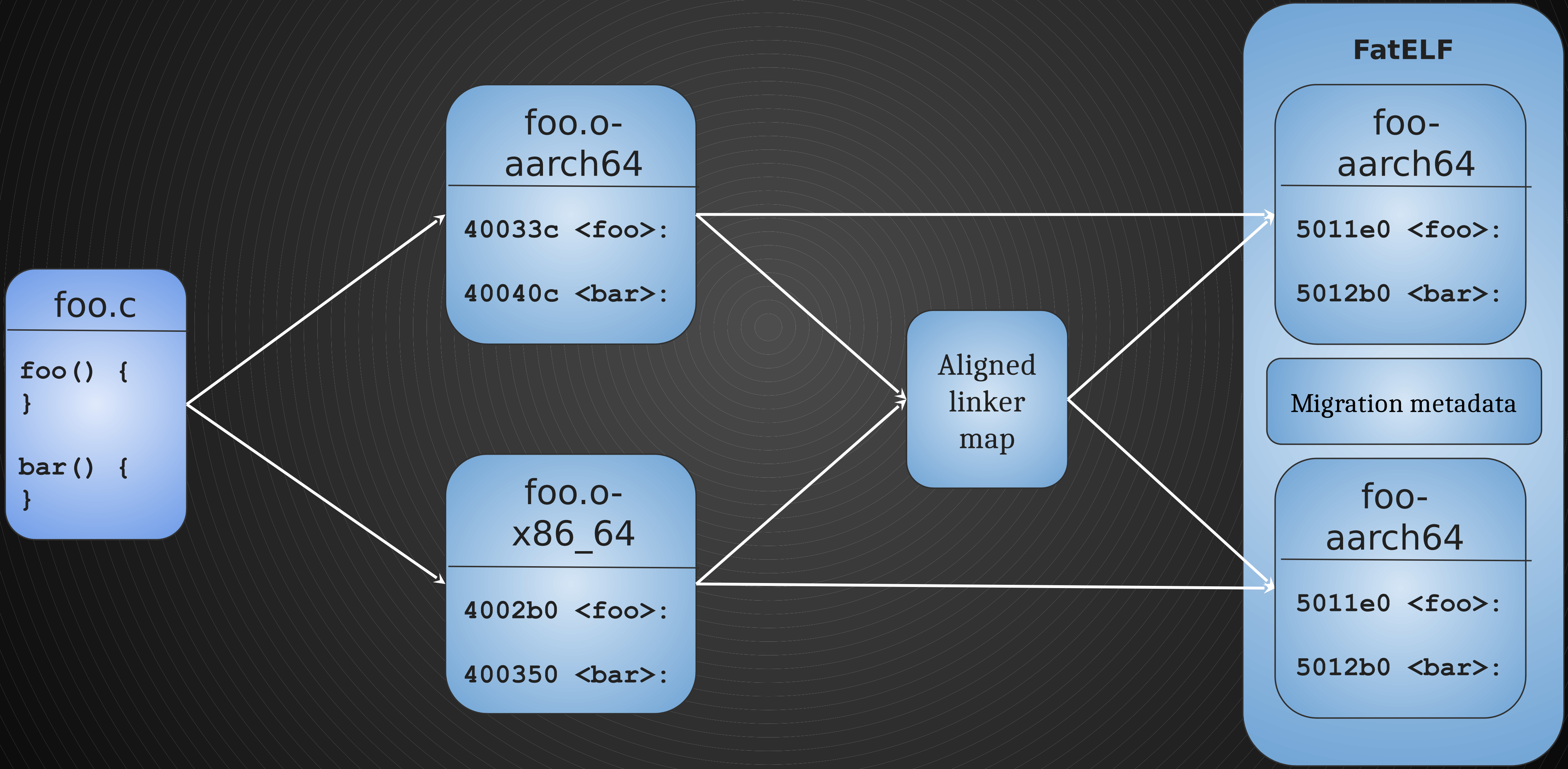
Heterogeneity and power efficiency

- 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

aarch64

```
pc:           5016e8
sp:       7fed1999d0
x[19]:             a
x[20]:            14
```

```
0x7fed1999d0:
d0:  0x0000007fed1999e0
d8:  0x000000000050147c
e0:  0x000000000000000a
e8:  0x0000000000000014
```

x86-64

```
rip:          5016e8
rsp:      7fed1999c0
rbx:               a
r14:              14
```

```
0x7fed1999c0:
c0:  0x0000007fed1999d0
c8:  0x000000000050147c
d0:  0x000000000000000a
d8:  0x0000000000000014
```

Migrating simple application

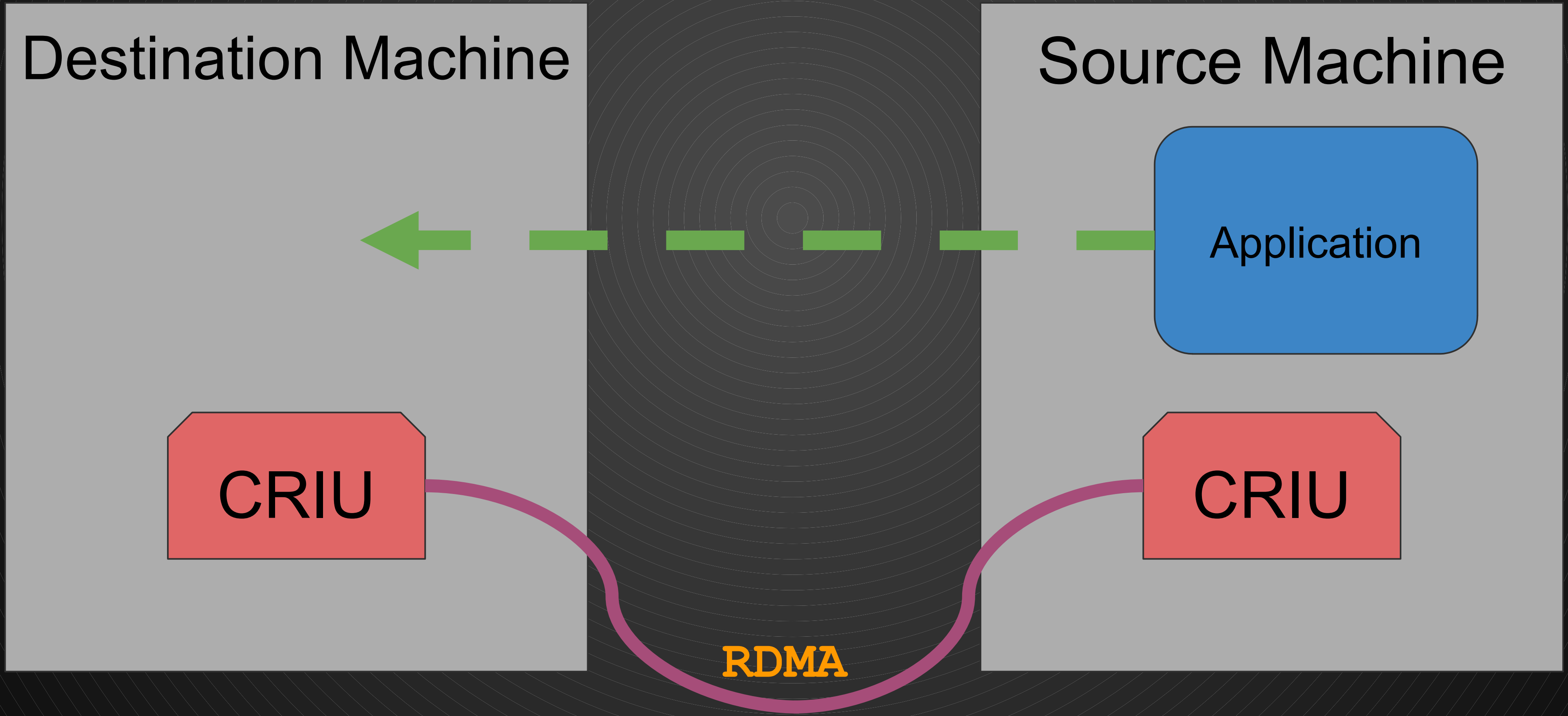
```
void f3(int a, int b) {
    printf("%s: a: %d, b: %d\n", __func__, a, b);
}

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/c5j7RTcYkcQqFGpsqiiuIkoaj

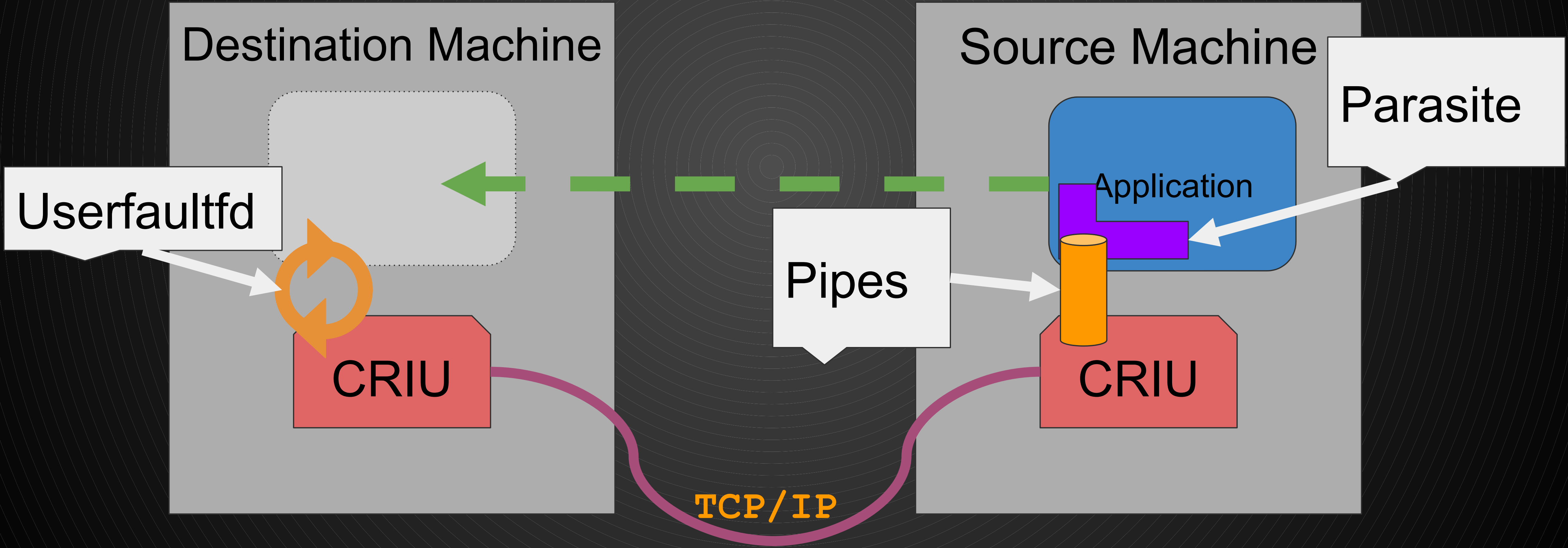
RDMA - registering memory problem

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!