ifunc: Remote Function Injection and Invocation Interface for UCX

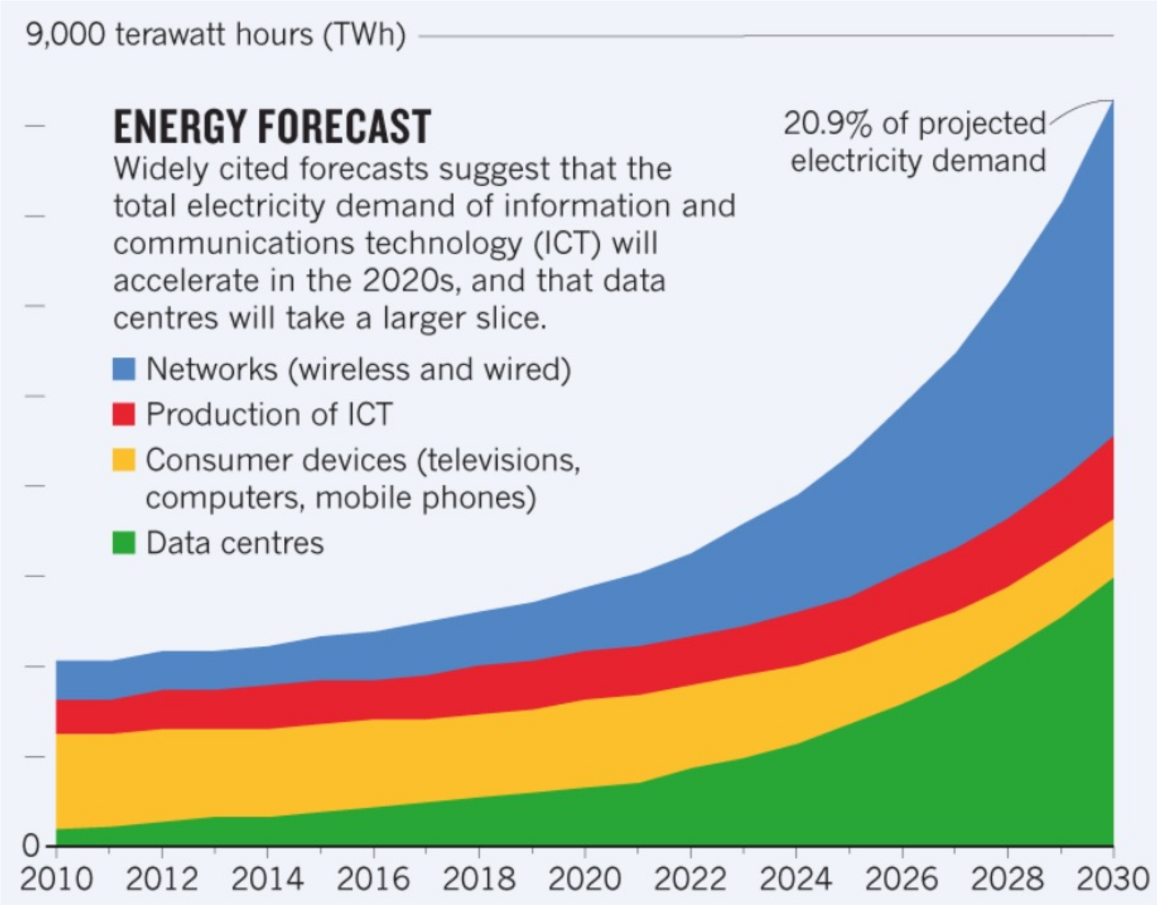
Wenbin Lü
Luis E. Peña
Pavel Shamis
Steve Poole

UCF Workshop 2021

12/02/2021

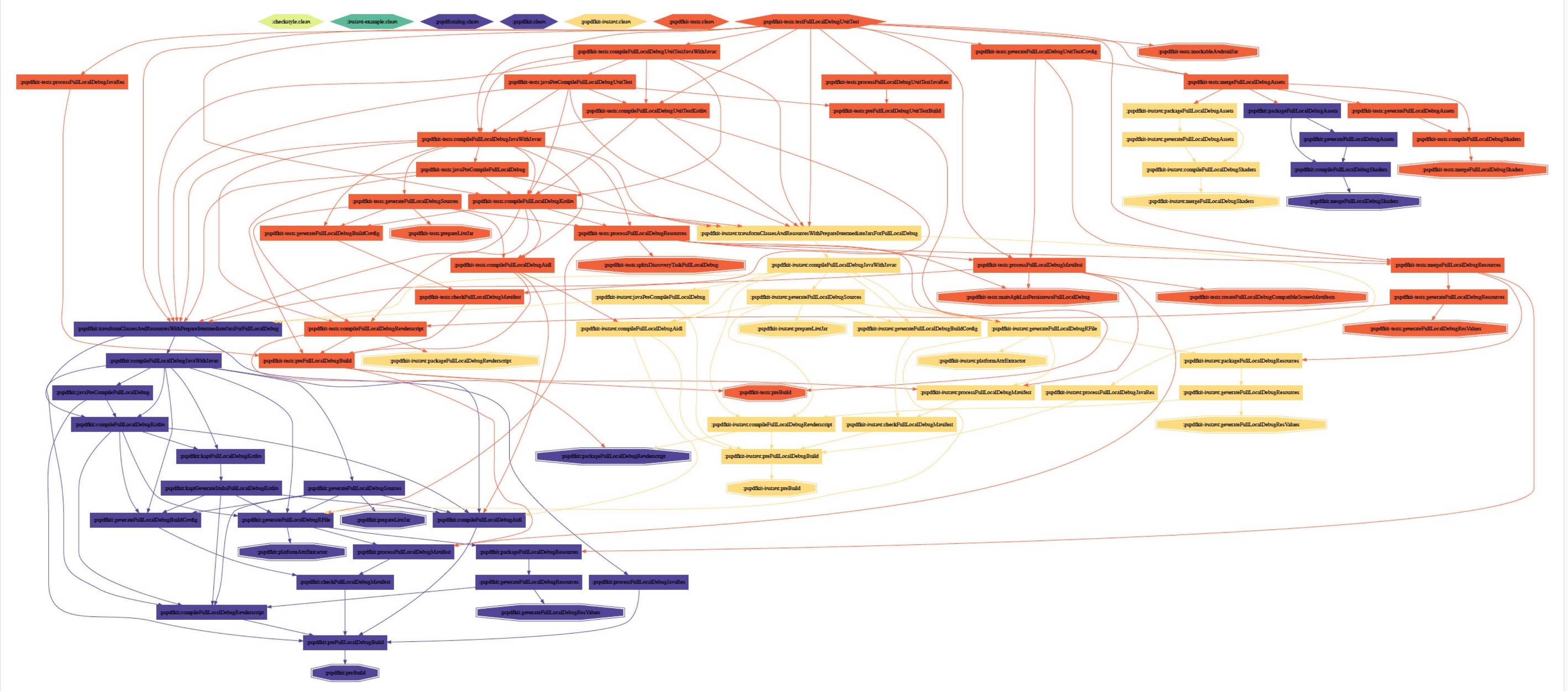
Motivation:

- Information explosion is happening non-stop
- Storing, processing and serving all these data consumes enormous amounts of energy
- Non-trivial financial and environmental impact



Source: https://www.nature.com/articles/d41586-018-06610-y

Motivation: Data-dependent Dynamic Applications

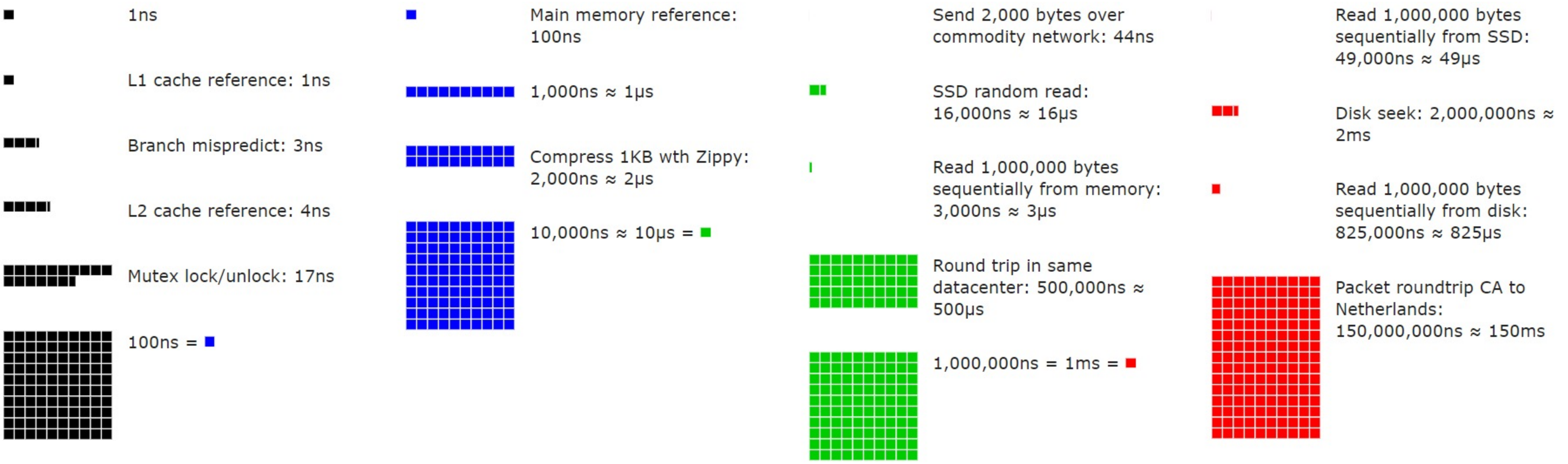


Source: https://twitter.com/flashmasterdash/status/1006564546142760960

arm

Motivation: The Cost of Moving Data Around

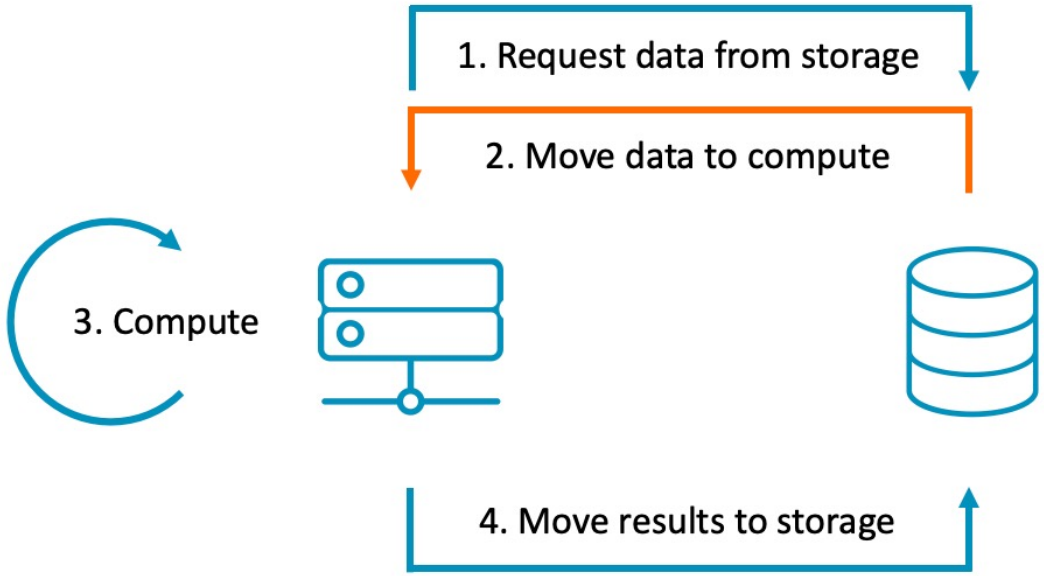
Latency numbers every programmer should know



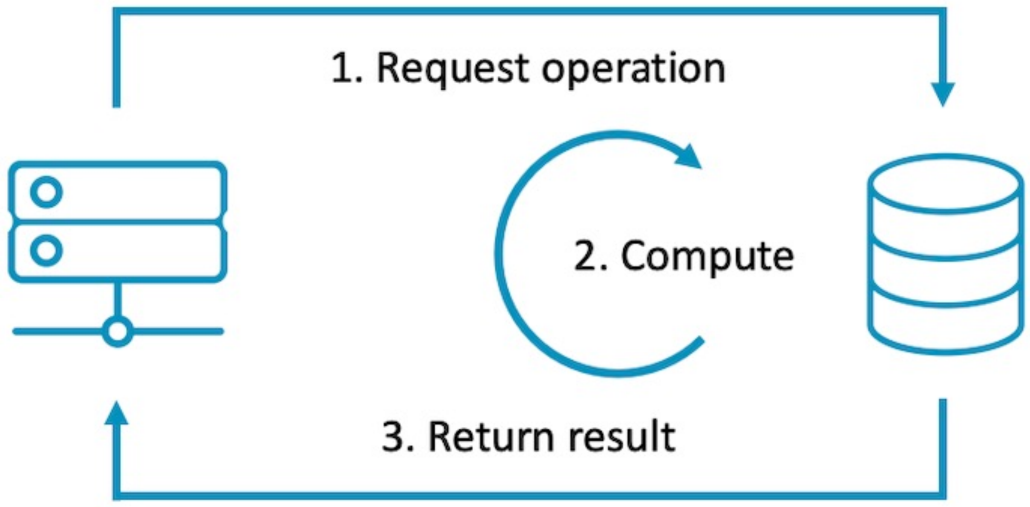
Source: https://colin-scott.github.io/personal_website/research/interactive_latency.html

Motivation: Let’s Move Compute to Data Instead

Traditional storage systems

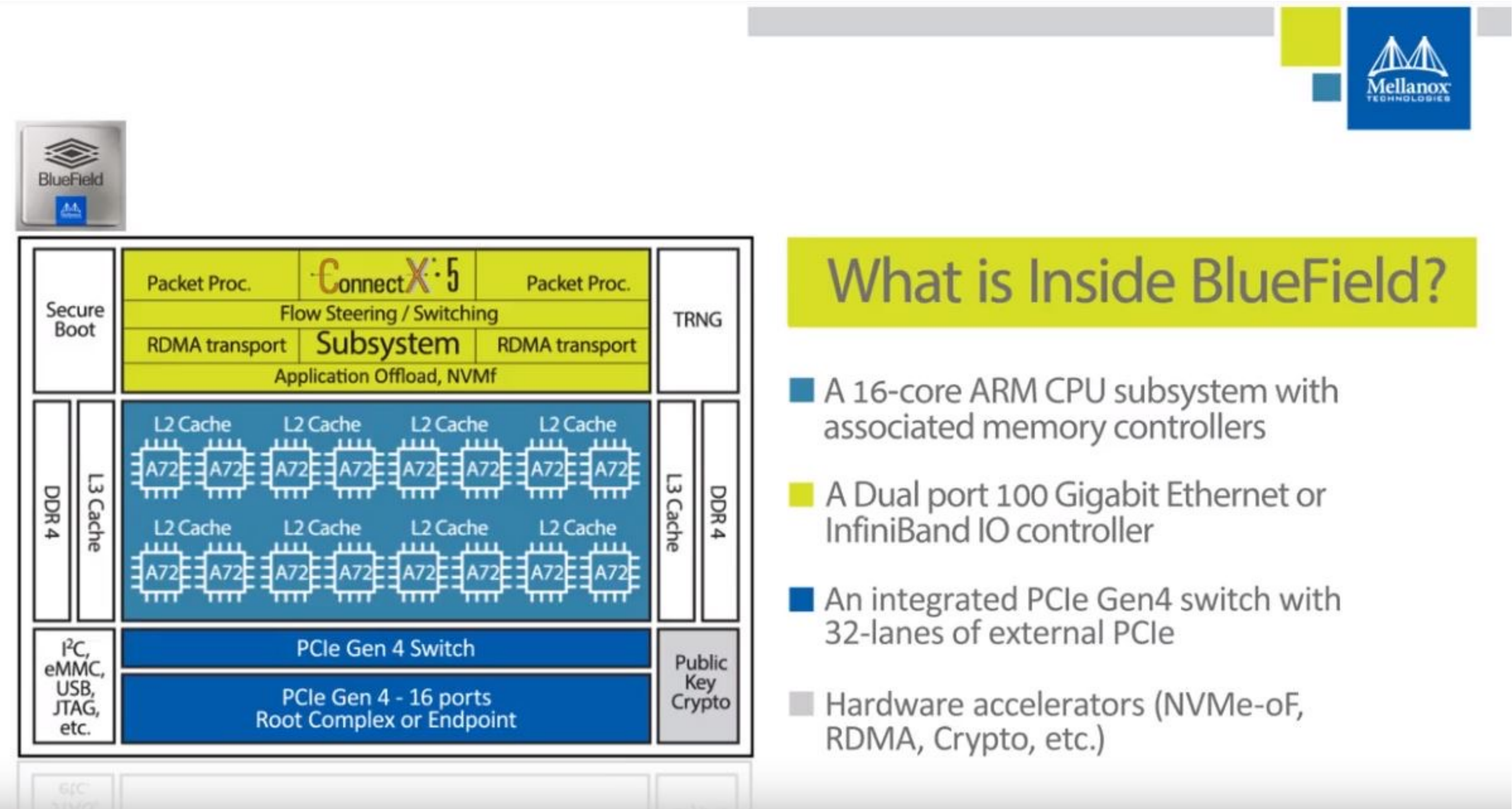


Computational storage systems



Source: https://community.arm.com/iot/b/internet-of-things/posts/computational-storage-is-bringing-processing-closer-to-the-data

Motivation: SmartNICs/DPUs are Also Coming

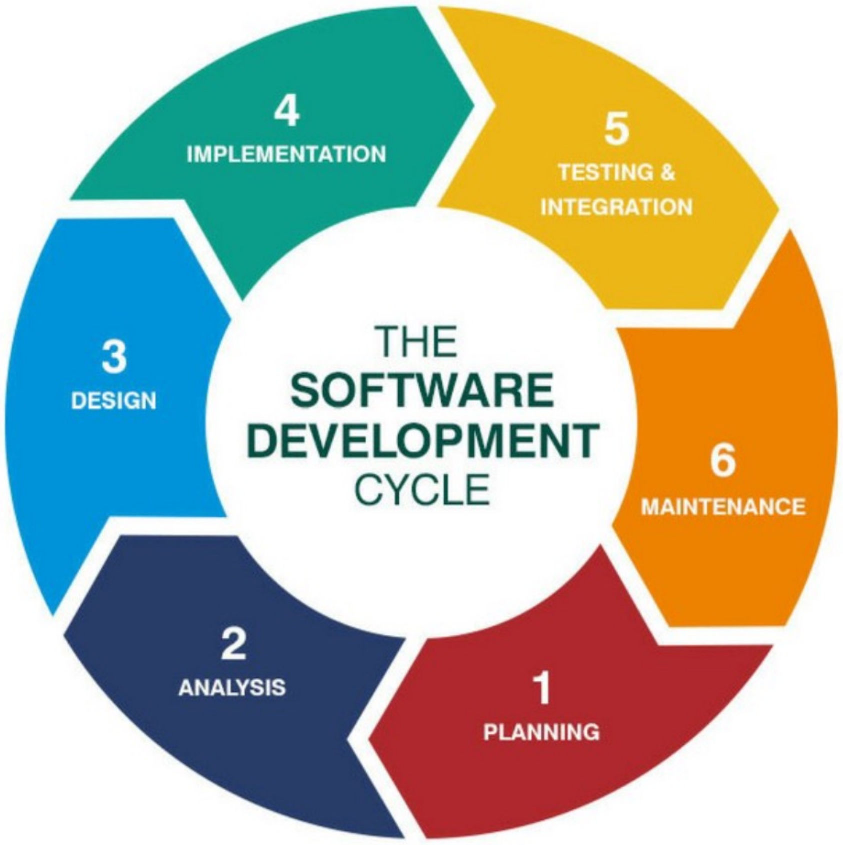


Source: Mellanox

Motivation: Is the Software Side Ready?

- How to move “compute” around?
  - Portability
  - Scalability
  - Maintainability

- Integration with existing libraries and development workflows?
  - Compatible with established solutions
  - Flexible enough to add new features with ease



Source: https://medium.com/@jilvanpinheiro/40d46afbe384

arm

Project Goal

API for moving compute to data in the form of remotely injected functions

&

Optimized implementation for said API

API name: ifunc

Outline

- Background
- The ifunc API design
- The ifunc API implementation
- Performance evaluation
- Conclusion & future work

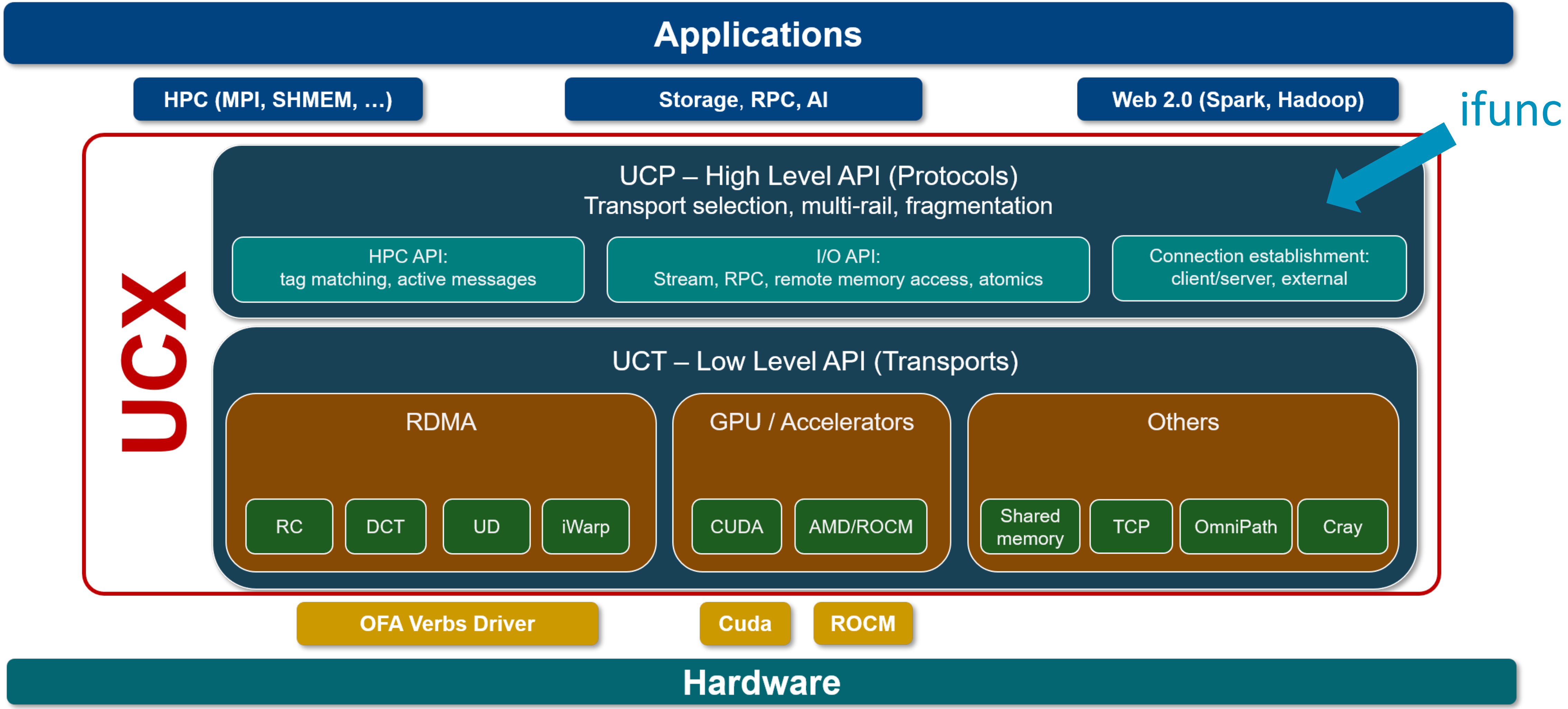
Background: The Two-Chains Framework

- Package, transfer and execution of code and data
  - A set of API and toolchain
- Fast, lightweight, portable
  - Low-latency & high-throughput
  - CPU <-> CSD <-> DPU <-> GPU
- Based on the UCX communication framework
  - Our work is open source, and we plan to submit it to the upstream

CLUSTER 2021: Two-Chains: High Performance Framework for Function Injection and Execution

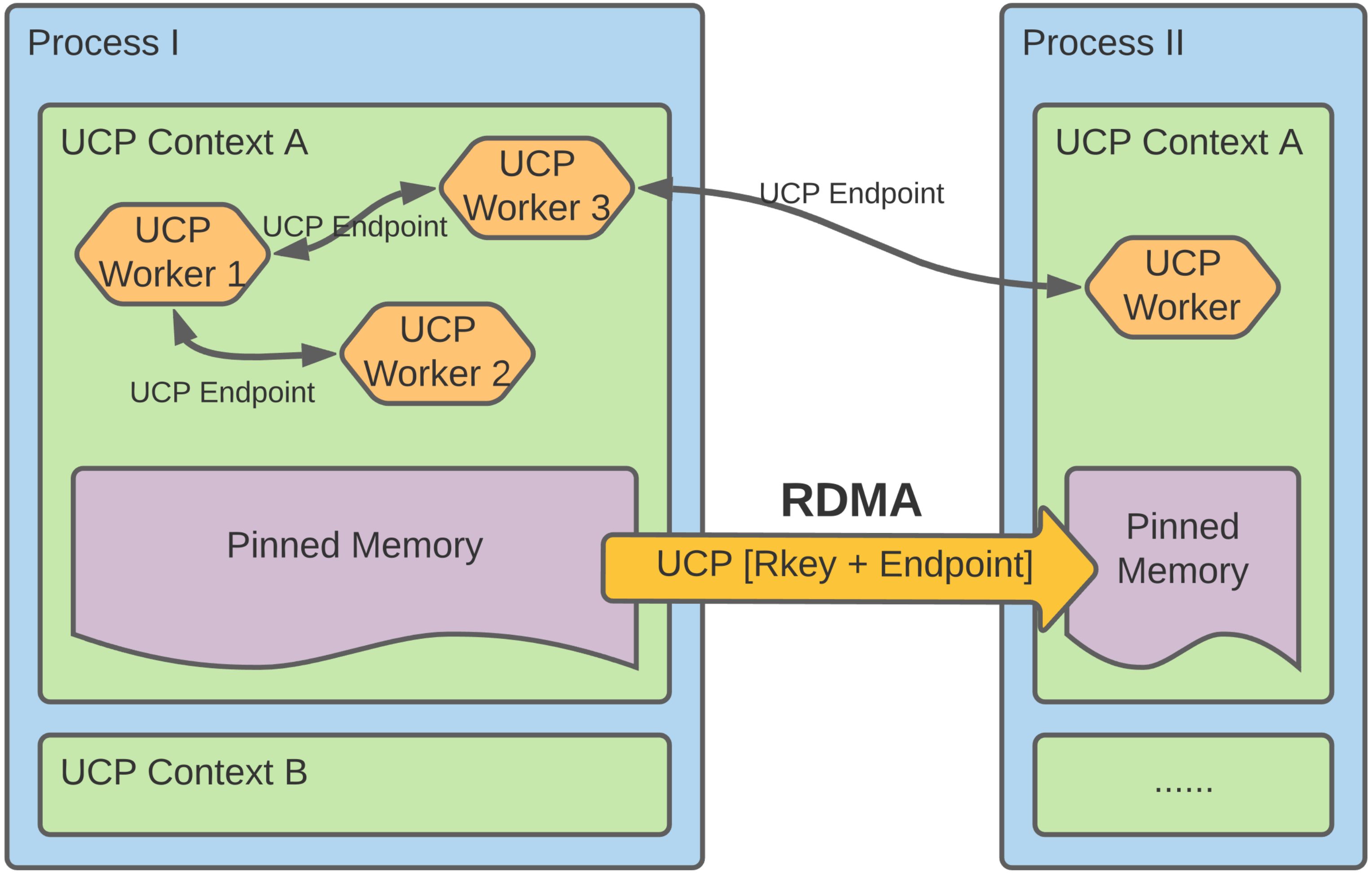
arm

Background: Where ifunc Fits in the Picture



Source: https://openucx.org/

Background: RDMA-PUT-Based ifunc



Basic Idea of RDMA-based ifunc

- A C function will be compiled and shipped to a remote process in the form of an ifunc message
  - RDMA writes are used to deliver the message
- The message also contains a set of arguments (a.k.a payload) for the ifunc
- The ifunc could access code and/or data on the target process (target arguments)

```
void foo_main(void *payload, size_t payload_size, void *target_args)
```

Comparison with UCX Active Messages

- UCX AM
  - User-defined handlers; transfer of payloads; active polling required
  - Handlers are registered on the target process
  - Handlers are rereferred to using compile-time determined numeric IDs
  - Internal on-demand message buffers

- ifunc
  - User-defined functions; transfer of payloads; active polling required
  - ifunc libs are loaded on the source process
  - ifunc libs are loaded at run-time and are identified using C strings (ifuncs' names)
  - Requires RDMA-enabled buffers allocated by the user

arm

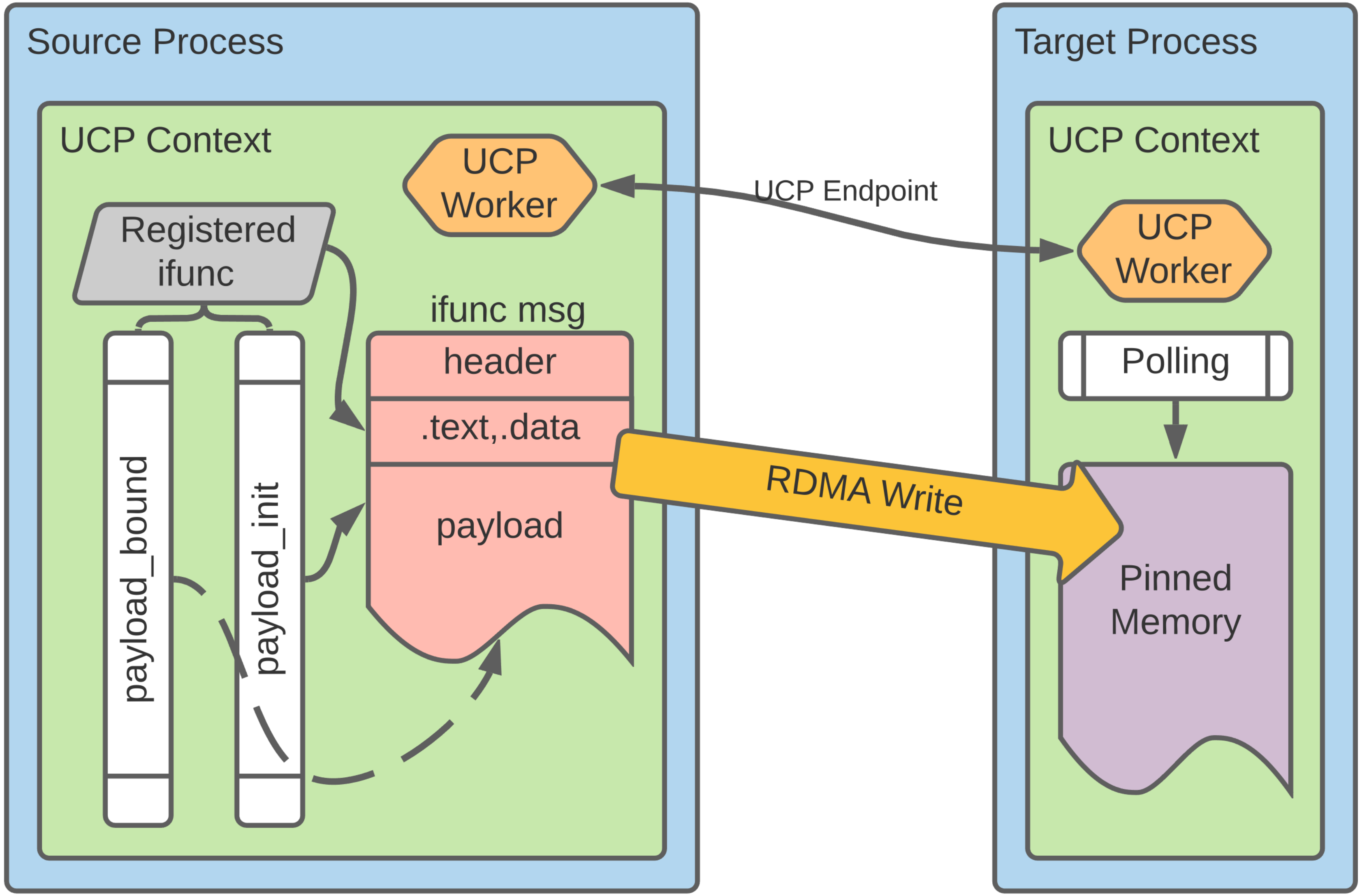
Creating an ifunc Library

- These three functions must be present (suppose your ifunc is called “foo”)

```
size_t
foo_payload_bound(void *source_args, size_t source_args_size)

int
foo_payload_init(void *source_args, size_t source_args_size,
                 void *payload,      size_t *payload_size)

void
foo_main(void *payload, size_t payload_size, void *target_args)
```

- Compile the library into foo.so
  - Placed in a directory accessible by the UCX application (export UCX_IFUNC_LIB_DIR=“…”)
  - Requires ISA-dependent compilation toolchain, more on this later

RDMA-based ifunc Workflow



UCP-level ifunc API

```
/* For source process */
typedef struct {
    char *name;
    int pure;
} ucp_ifunc_reg_param_t;

ucs_status_t
ucp_register_ifunc(ucp_context_h context, ucp_ifunc_reg_param_t param, ucp_ifunc_h *ifunc_p)

ucs_status_t
ucp_ifunc_msg_create(ucp_ifunc_h ifunc_h,     void *source_args,
                     size_t source_args_size, ucp_ifunc_msg_t *msg_p)

ucs_status_t
ucp_ifunc_send_nbix(ucp_ep_h ep, ucp_ifunc_msg_t msg, uint64_t remote_addr, ucp_rkey_h rkey)

/* For target process */
ucs_status_t
ucp_poll_ifunc(ucp_context_h context, void *buffer, size_t buffer_size, void *target_args)
```

RDMA-based ifunc Workflow

- Target process
  - Allocates an RDMA-enabled (pinned) memory buffer to receive ifunc messages
    - Tell the target about its virtual address and size
  - Poll the buffer for delivered ifunc messages and execute them
- Source process
  - Registers an ifunc using its name
  - Creates ifunc messages using source arguments
    - UCX runtime prepares the payloads with `payload_bound` and `payload_init` of the ifunc library
  - RDMA write the ifunc messages to the target process's buffer

Sample ifunc Library

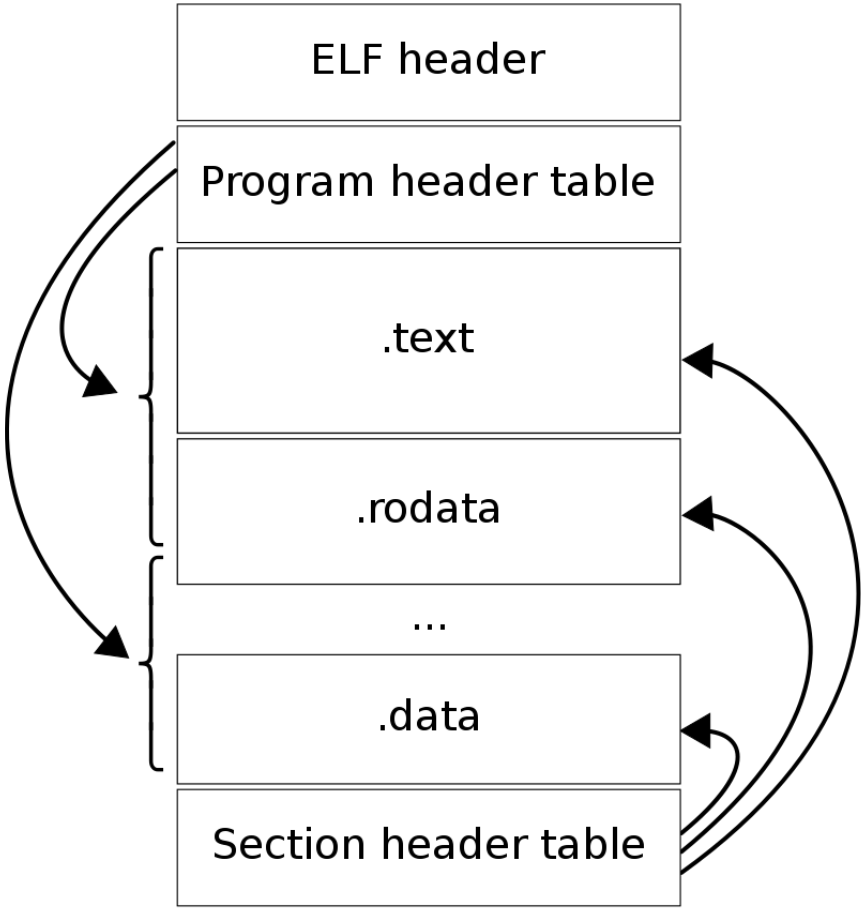
```
size_t foo_payload_bound(void *source_args, size_t source_args_size)
{
    return est_encode_size(source_args, source_args_size);
}

int foo_payload_init(void *source_args, size_t source_args_size,
                     void *payload,     size_t *payload_size)
{
    encode(payload, payload_size, source_args, source_args_size);
    return 0;
}

void foo_main(void *payload, size_t payload_size, void *target_args)
{
    db_handle dbh = *(db_handle*)target_args;
    decode_insert(dbh, payload, payload_size);
}
```

Sample ifunc Application

```
/* On the source process */
ucp_register_ifunc(ucp_ctx, irp, &ih);  // irp = {"foo", 0}

ucp_ifunc_msg_create(ih, record, record_size, &imsg);

ucp_ifunc_send_nbix(ep, imsg, recv_buffer, rmt_rkey);

ucp_ep_flush_nb(ep, 0, ep_flush_cb);


/* On the target process */
do {
    ret = ucp_poll_ifunc(ucp_ctx, recv_buffer, recv_buffer_size, &db_handle);
} while (ret != UCS_OK);
```

Implementating ifunc

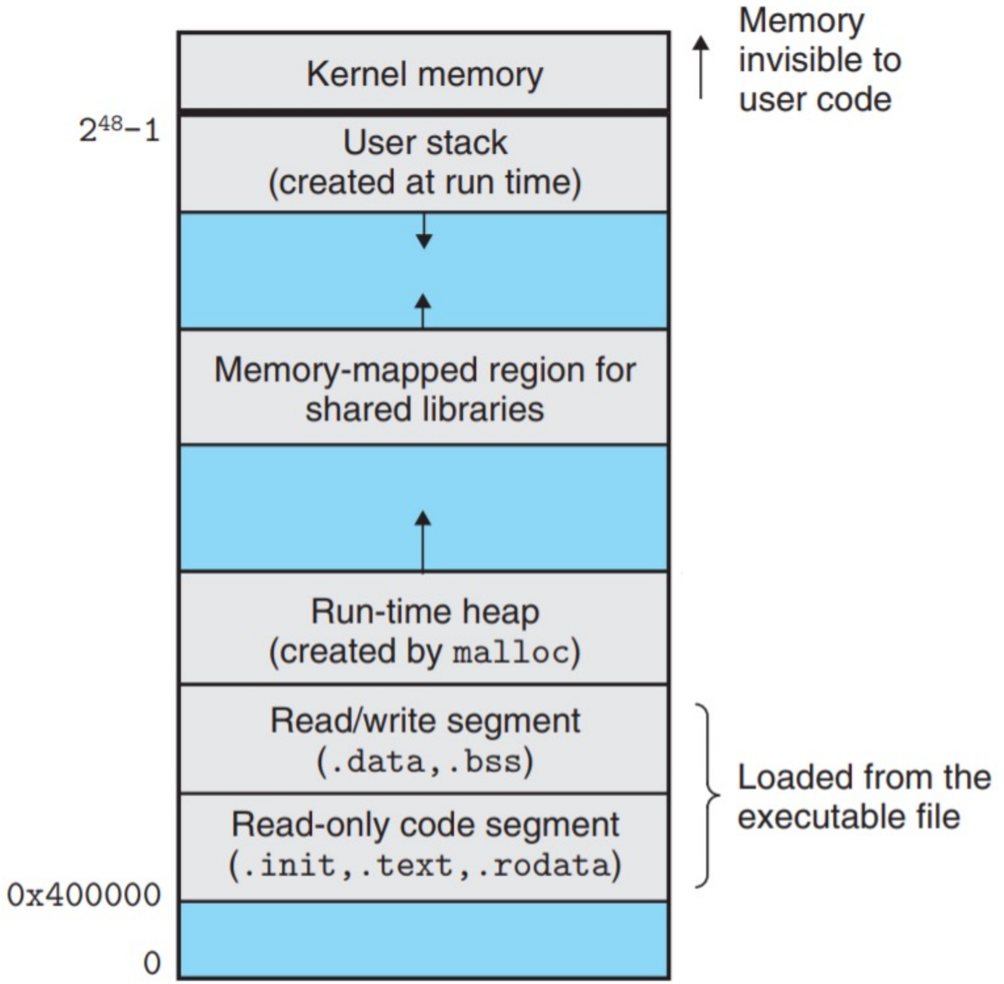
- Use dlopen to load the ifunc dynamic library
  - One-time registration cost
  - Ship the .text, .rodata, .data sections in the message
  - All “internal” functions, global/static variables are working



Source: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

Implementating ifunc

- Use dlopen to load the ifunc dynamic library
  - One-time registration cost
  - Ship the .text, .rodata, .data sections in the message
  - All “internal” functions, global/static variables are working
- What about external symbols?
  - Functions: printf, malloc, clock_gettime, rand, etc.
  - Also: global variables on the target process
  - Address space layout randomization (ASLR)



Source: https://stackoverflow.com/questions/63465933

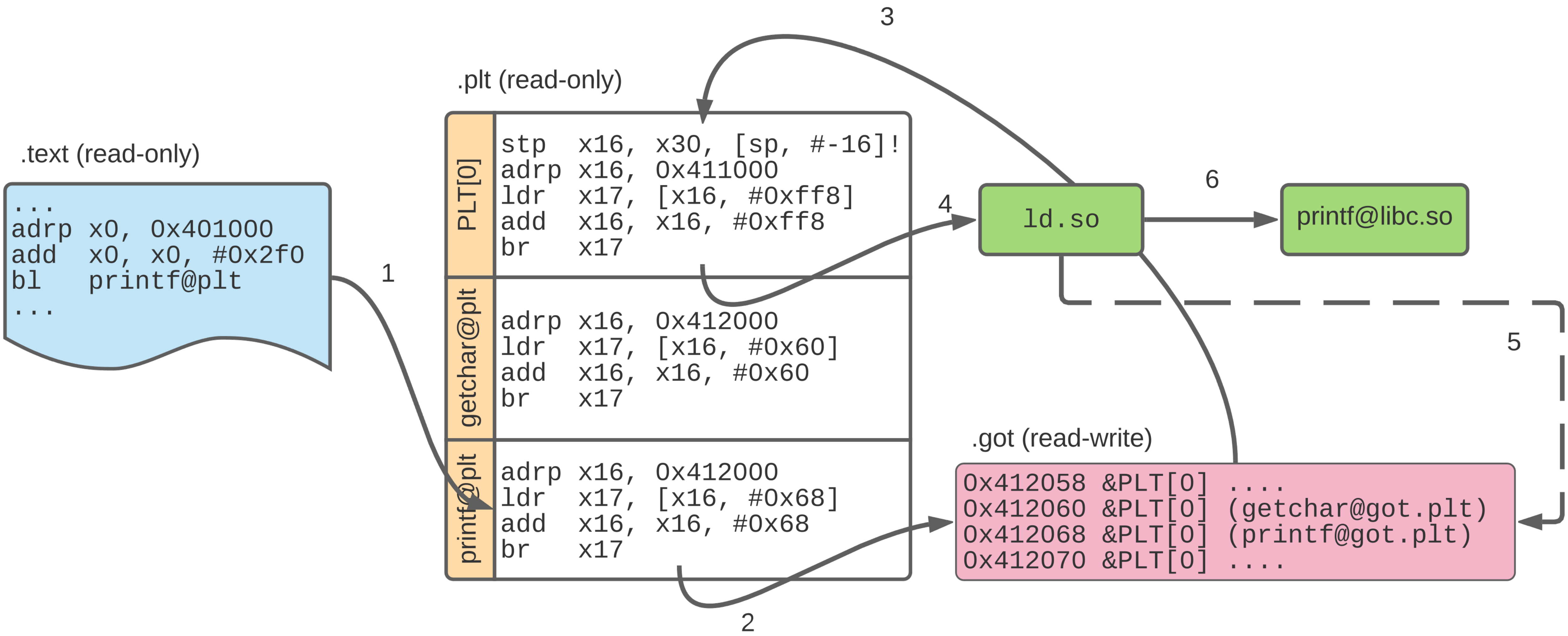
Dynamic Linking: a Recap

- Resolve virtual addresses of symbols at program load-time (or even later)
  - It's the OS's C library's job, read the manpage of ld.so if interested

- The .text section only contains PC-relative offsets to functions and variables
  - ASLR + PIC & PIE = unpredictable relative offsets!

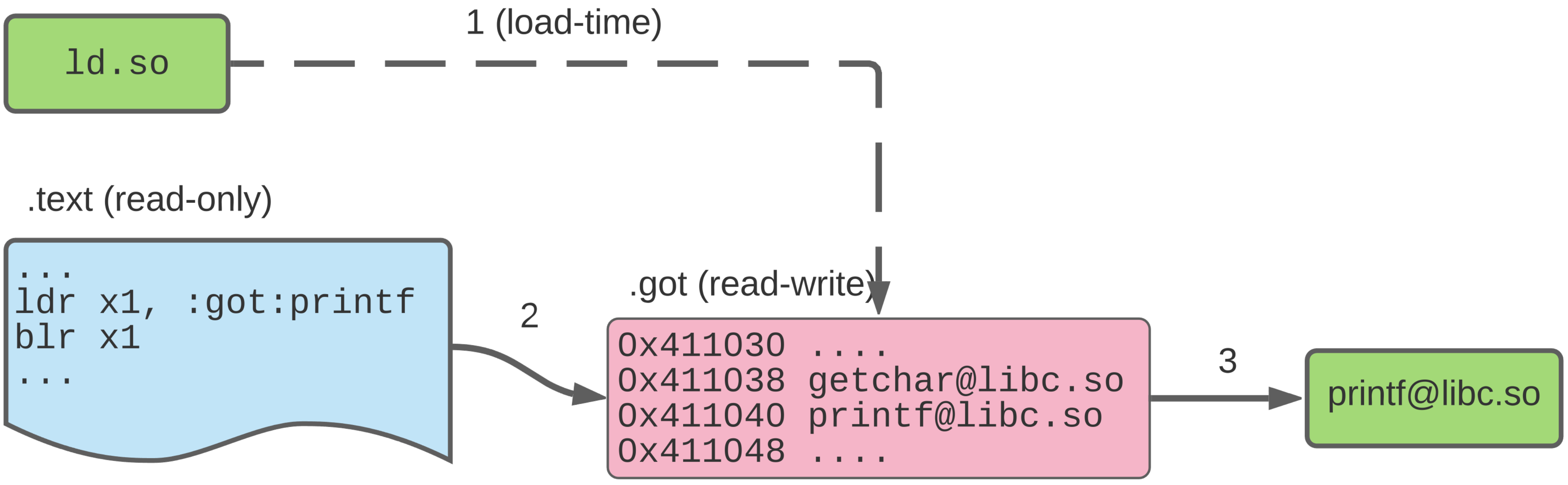
- All problems in computer science can be solved by another level of indirection (fundamental theorem of software engineering).
  - Procedure Linkage Table (PLT) and Global Offset Table (GOT)

Dynamic Linking: PLT and GOT

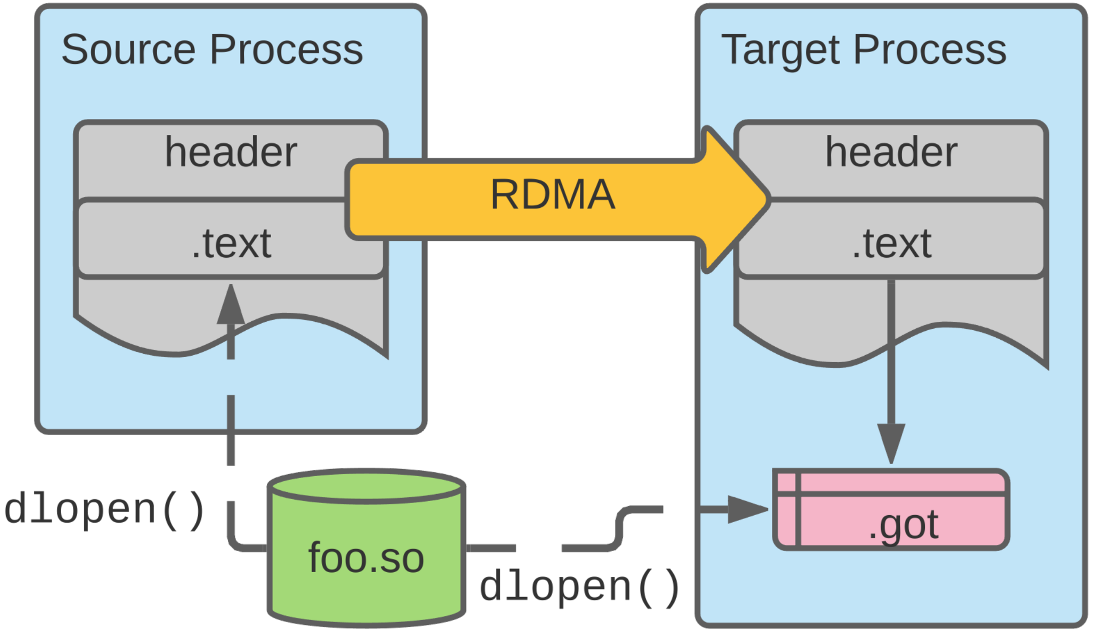


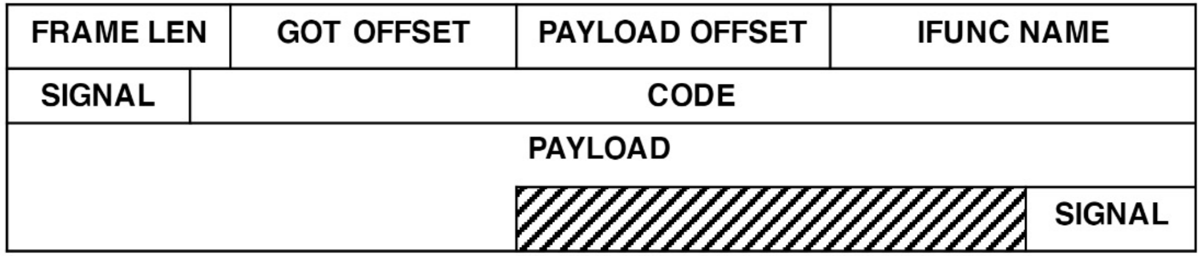
Dynamic Linking: GOT-only Early-binding

- Compile with –fno-plt



Remote Dynamic Linking: Borrowing the GOT





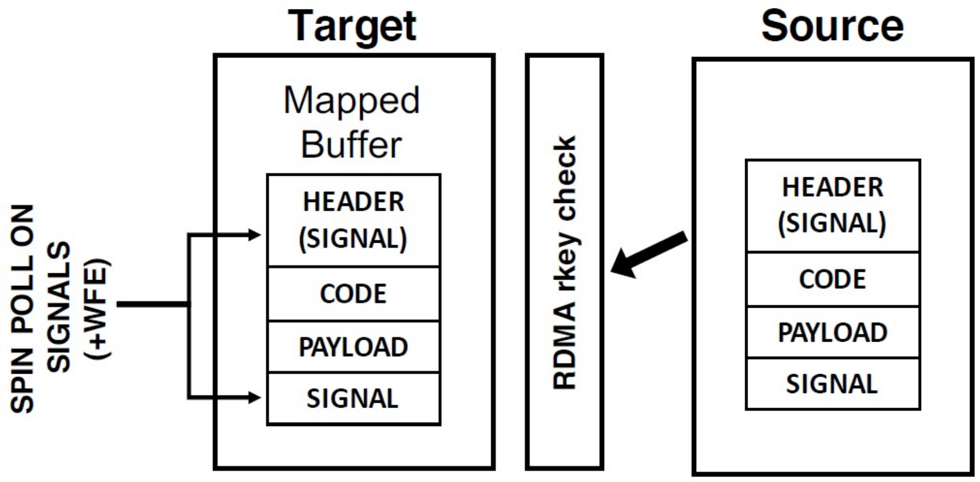

```
ldr x1, :got:printf
blr x1
```

patch_asm.py

```
ldr x1, foo$got
ldr x1, [x1, #:got_lo12:fib]
blr x1
```

Security Concerns

- Isn't this literally executing arbitrary code sent by someone else?
- The InfiniBand standard specifies the use of a 32-bit RKEY to perform writes to pinned memory
- The ifunc dynamic libraries are stored on the filesystem, governed by FS permissions
- As safe as the rest of your application/system



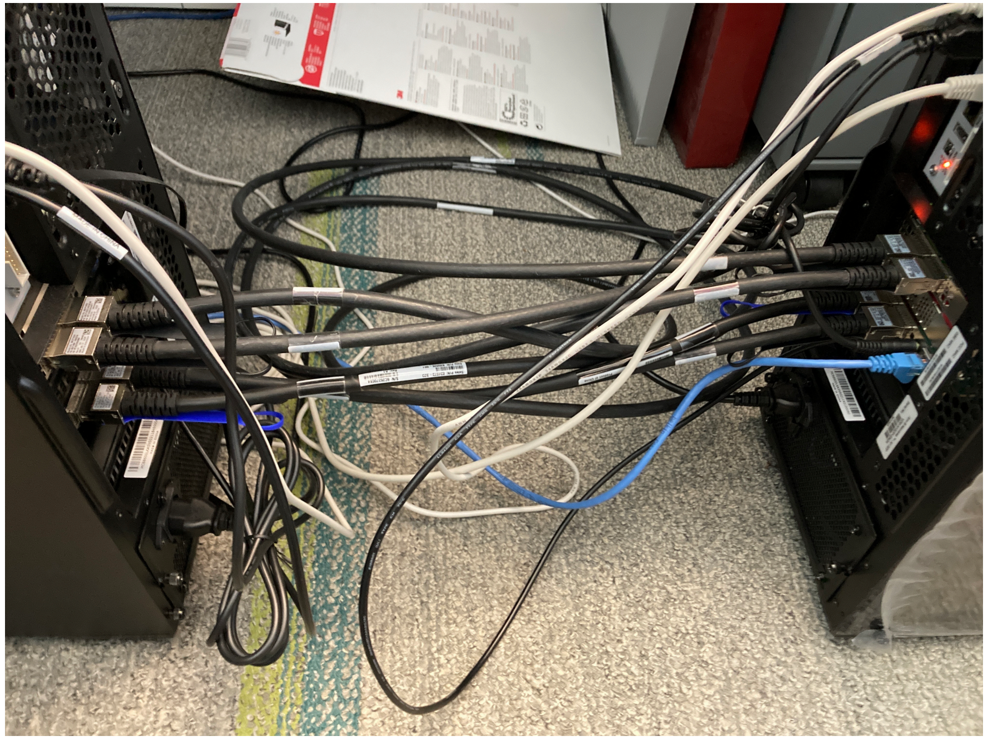
Caveat: Instruction Cache Coherency

- One of our test machine’s L1i & L1d caches are incoherent!
  - We’re in a “code is data” situation so this becomes an issue
- The polling loop checks the content of the buffer until a message arrives
  - The i-cache must be cleared before we branch to the ifunc’s .text section
- Non-trivial performance penalty
  - Especially for small payload sizes

Performance Evaluation

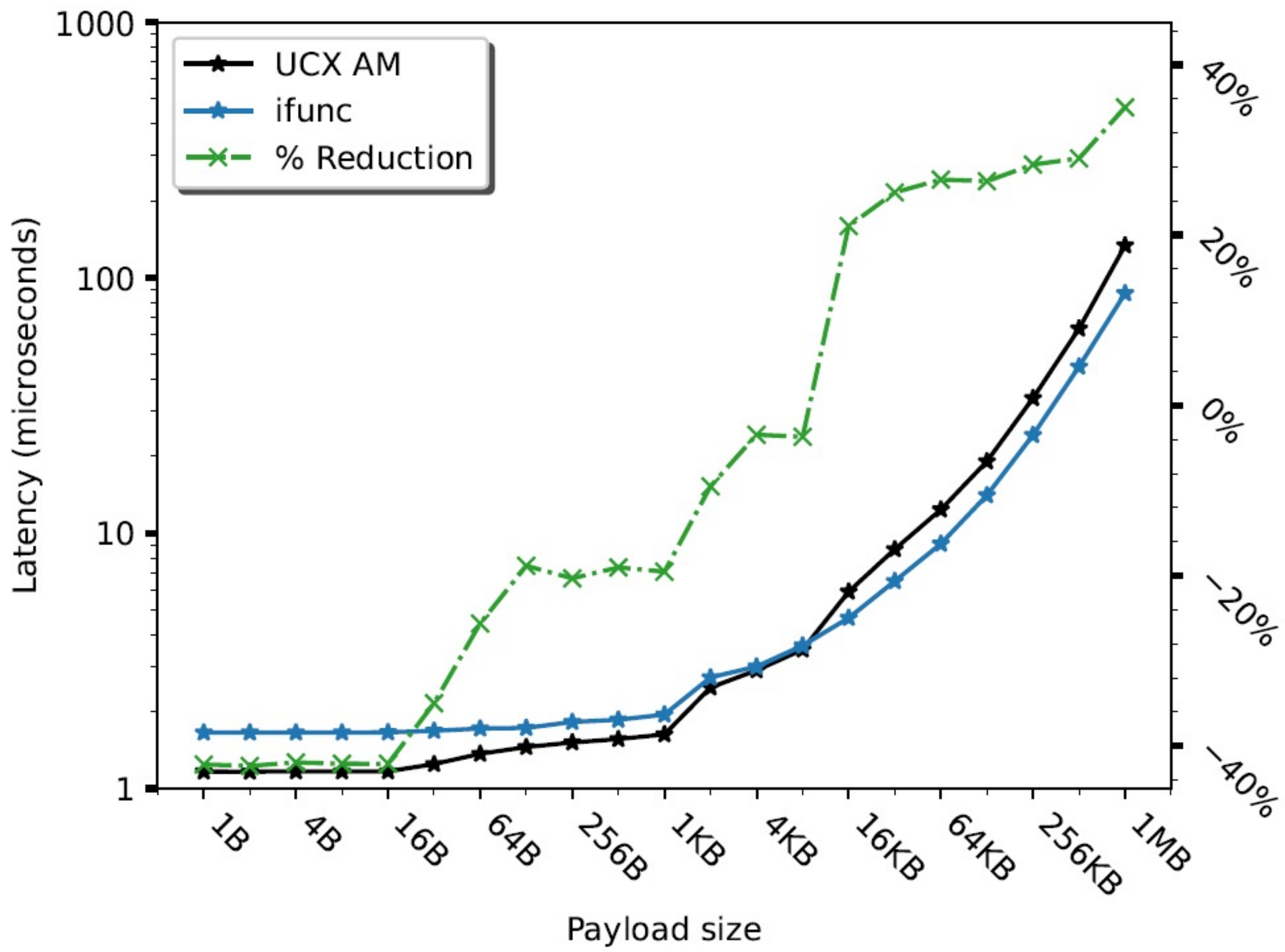
- Measure our implementation's point-to-point message latency and throughput
  - Also compare against UCX AM

- Hardware & software setup:
  - CPU: Neoverse-N1
  - NIC: Mellanox Connectx-6 MT28908 HDR 200Gb/s
    - Connected back-to-back without an IB switch
  - OS: Ubuntu 20.04.2
  - All results are inter-node numbers

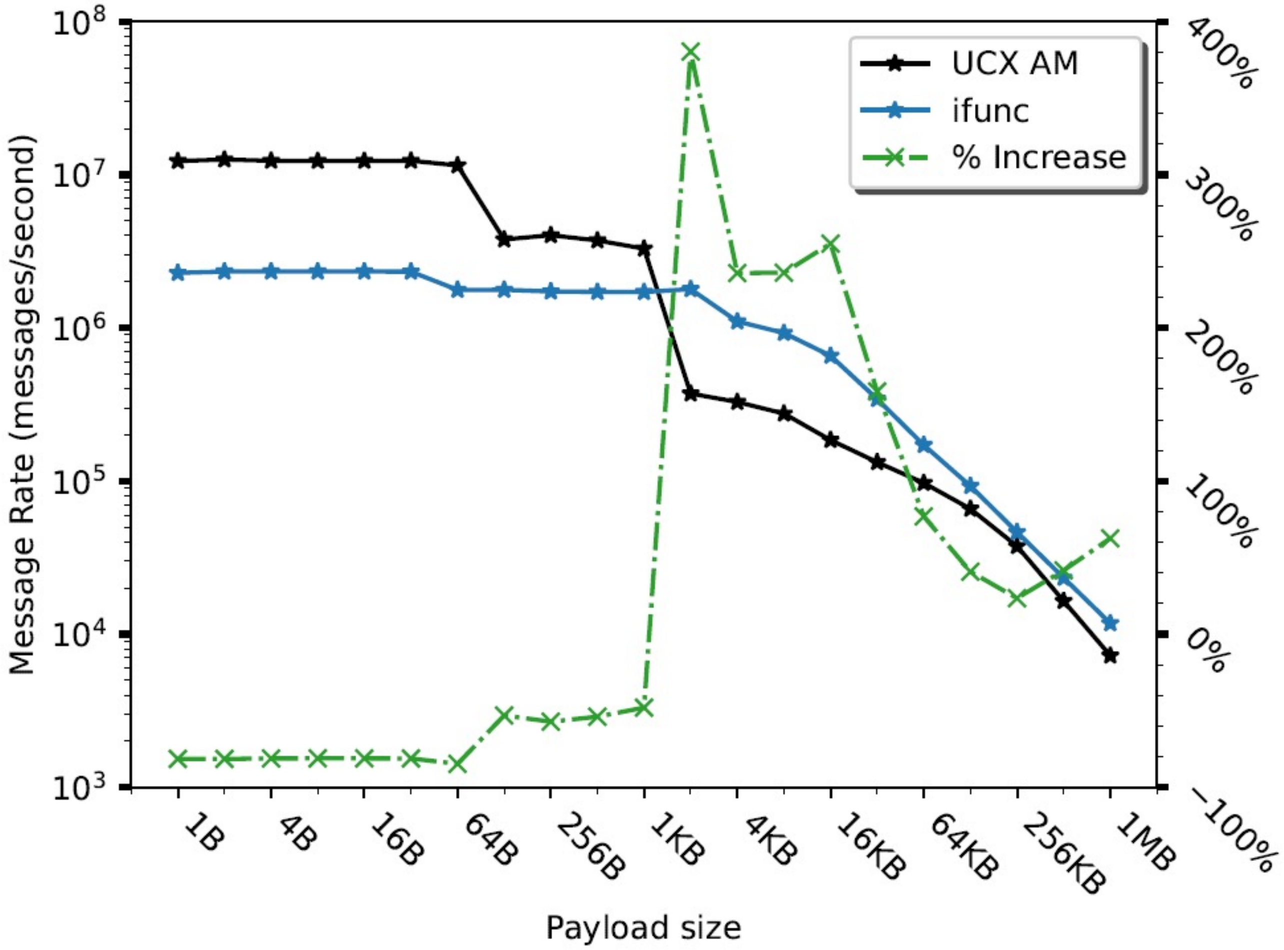


arm

Performance Evaluation: Message Latency



Performance Evaluation: Message Rate

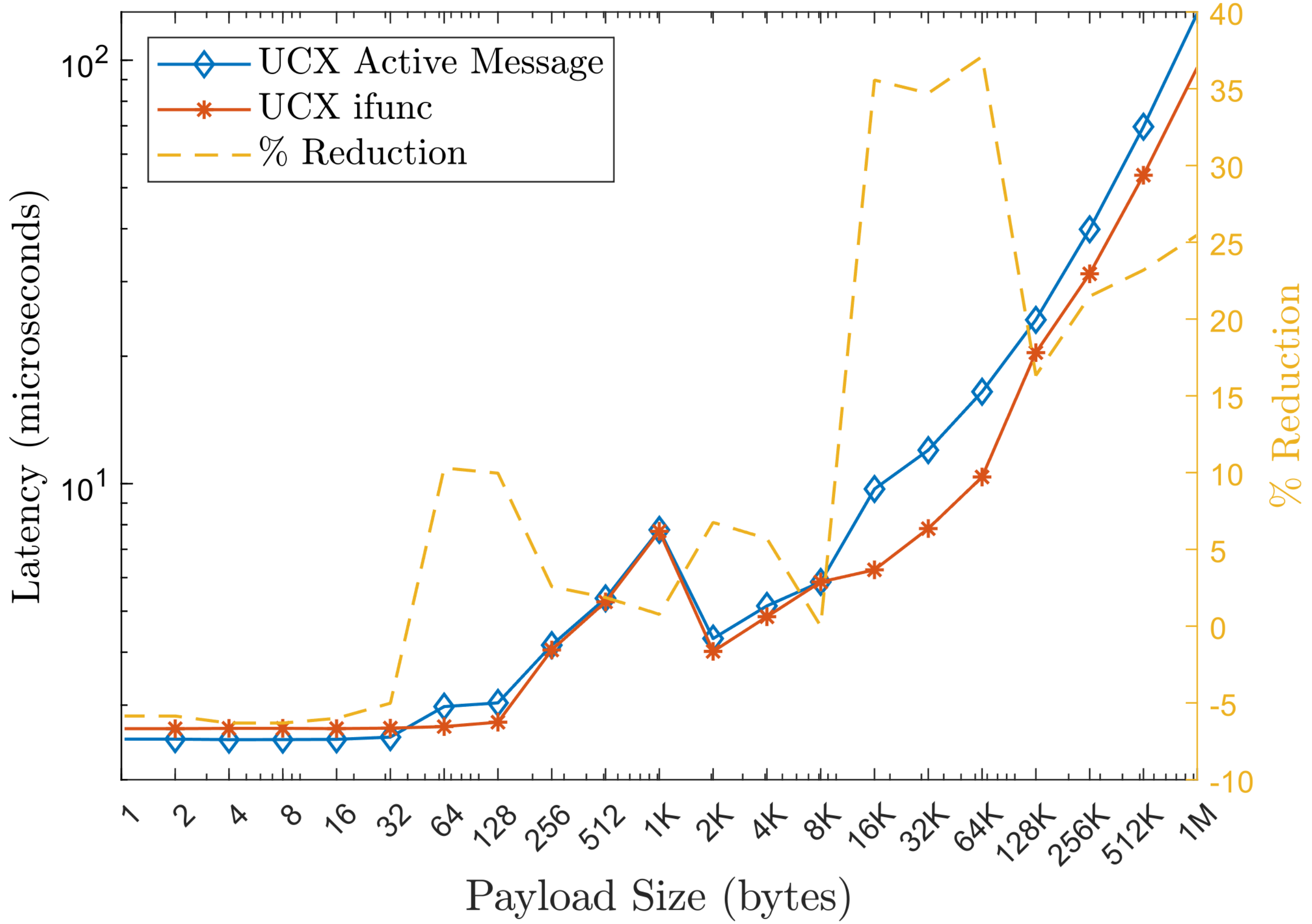


arm

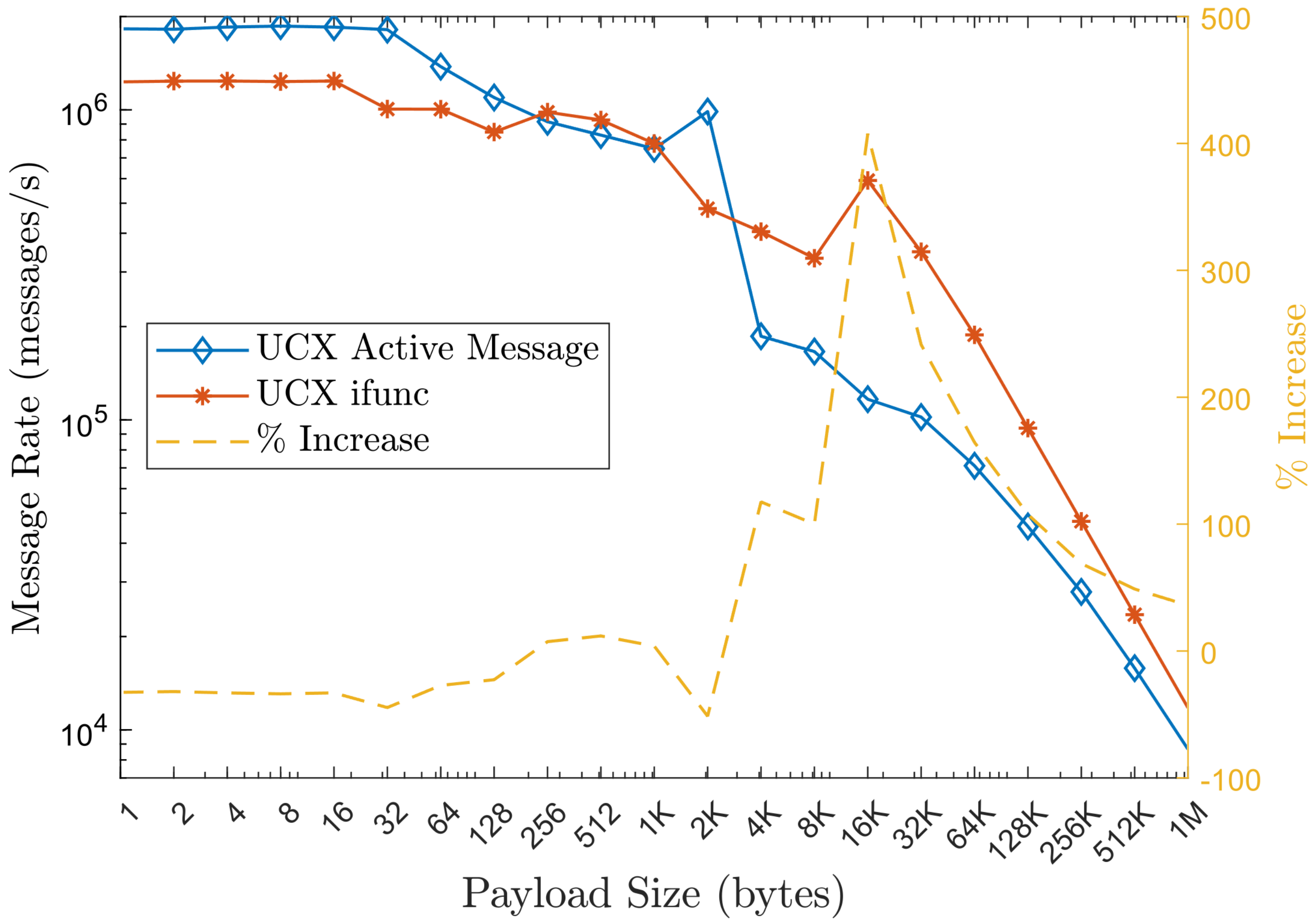
Performance Evaluation, cont.

- Also evaluated on the Ookami cluster at Stony Brook University
  - L1 caches are coherent, no more expensive cache clearing
  - With several improvements/fixes here and there
- Hardware & software setup:
  - CPU: A64FX FX700 (32 GB HBM2)
  - NIC: Mellanox Connectx-6 MT28908 HDR 100Gb/s
  - OS: CentOS 8.1.1911
  - All results are inter-node numbers

Performance Evaluation: Message Latency



Performance Evaluation: Message Rate



Published Results

- Open-source code release under the umbrella of OPENSNAPI working group (collaboration between NVIDIA, LANL, Huawei, Arm): https://github.com/openucx/ucx-two-chains

- Paper accepted by OpenSHMEM Workshop 2021
  - *UCX Programming Interface for Remote Function Injection and Invocation*
  - Authors: Luis E. Peña, Wenbin Lu, Pavel Shamis, and Steve Poole

arm

Conclusions

- Move compute to data to save time and energy
- The RDMA-based ifunc API of the Two-Chains framework is our first step
- Send binary code and data payload to remote processes for execution
- Performance comparable to native UCX active messages for all payload sizes
- Still need to work on remote dynamic linking

Future Work

- Implement full remote dynamic linking
  - The ifunc dynamic library is no longer needed on the target process's filesystem!
- Switch to send-recv communication
  - No more user-managed buffers, no HugePage & RWX privilege compatibility issues
  - Incoming messages are progressed along with other UCX activities
- Portable ifunc library compilation toolchain
  - LLVM?