UCP ACTIVE MESSAGES

DECEMBER 2021

AGENDA

Introduction

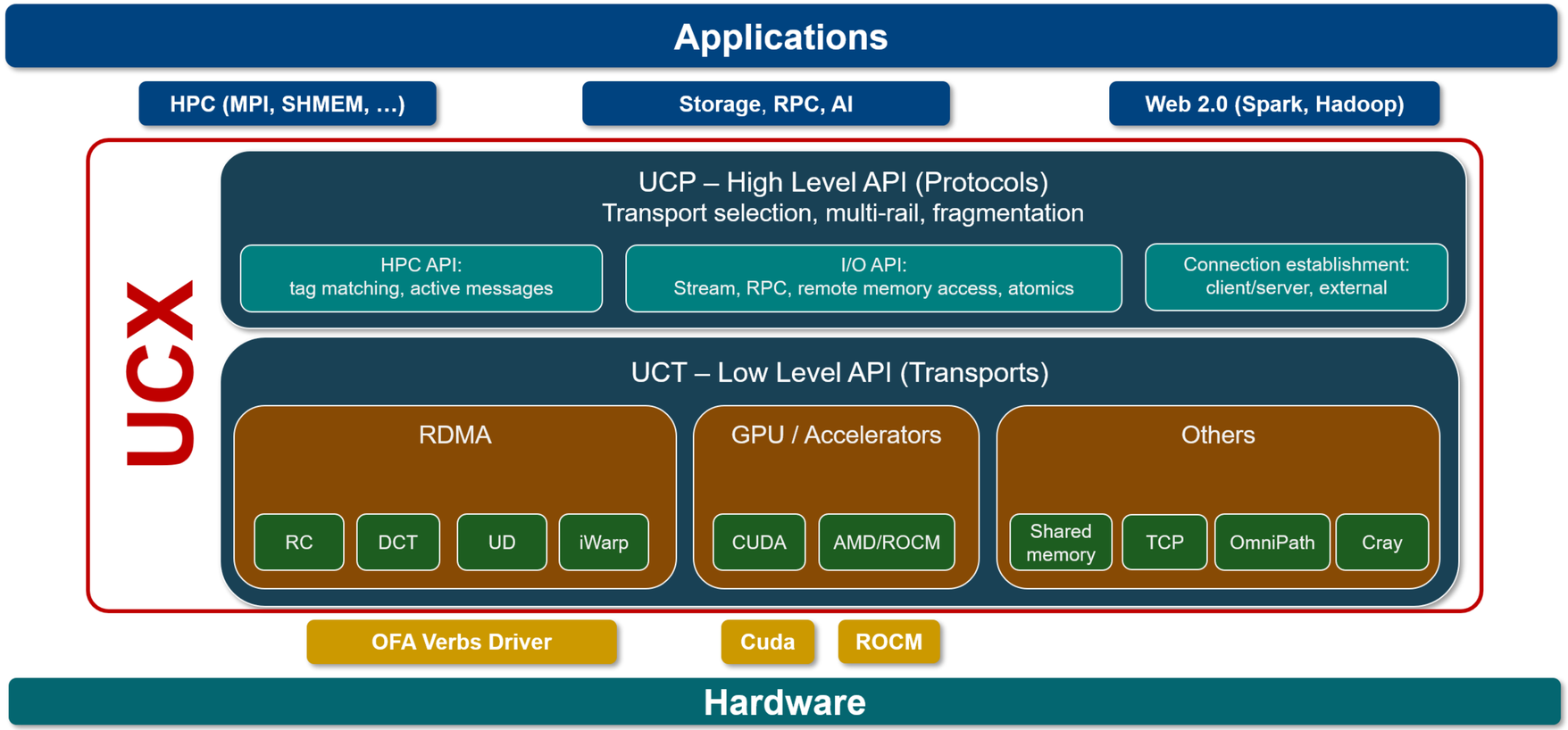
Use Cases

Active Message API

Future Work

INTRODUCTION

What is UCX



UCP provides high level APIs:

- Remote memory access – one-sided communications, targeting MPI, SHMEM, other PGASes
- Tag based send/receive API – targeting MPI implementations, but is also used for storage and other PGAS frameworks
- Stream based send/receive – is not actively developed now (no rendezvous, no GPU memory support, etc)
- Active Messages - communications driven by message handlers

INTRODUCTION

Supported Features (TAG vs AM APIs)

Features	TAG API	AM API
Rendezvous protocol	V	V
Multi-rail	V	V
GPU memory support	V	V
Tag matching	V	X
Zero copy on the receiver with eager protocol	X	V
Full error handling support	X	V
User header support	X	V

USE CASES

Where AM can be useful

- Cloud storage applications (UCX iodemo example)
- Spark-GPU and Rapids
- AI (ps-lite)
- Charm++
- GASNet
- Collectives (UCC, others)

ACTIVE MESSAGE API

Basic API

- Set AM handler:

```
ucs_status_t
ucp_worker_set_am_recv_handler(ucp_worker_h worker,
                               const ucp_am_handler_param_t *param);
```

- Clear AM handler:

Invoke `ucp_worker_set_am_recv_handler()` with `param->cb = NULL`

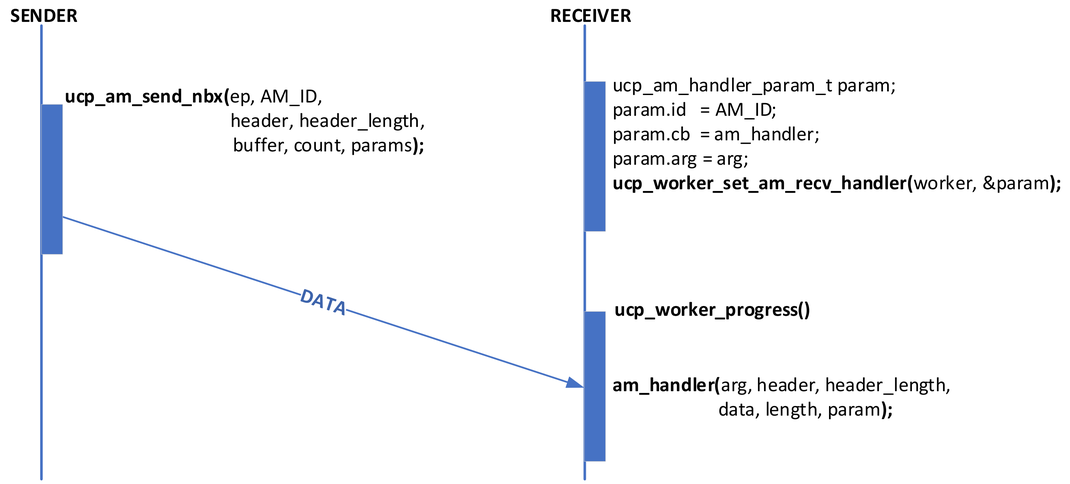
- Data handler semantic

```
ucs_status_t
(*ucp_am_recv_callback_t)(void *arg, const void *header,
                          size_t header_length,
                          void *data, size_t length,
                          const ucp_am_recv_param_t *param);

 struct ucp_am_recv_param {
     uint64_t           recv_attr;
     ucp_ep_h           reply_ep;
 };
```

- Send function:

```
ucs_status_ptr_t
ucp_am_send_nbx(ucp_ep_h ep, unsigned id, const void *header, size_t header_length,
                const void *buffer, size_t count,const ucp_request_param_t *param);
```



ACTIVE MESSAGE API

API flags

- Send flags (passed to `ucp_am_send_nbx()` in `param->flags`):
  - `UCP_AM_SEND_FLAG_REPLY`: ep which can be used for replying to sender will be passed to receive callback on the receiver side
  - `UCP_AM_SEND_FLAG_EAGER`: send using eager protocol regardless of the message length
  - `UCP_AM_SEND_FLAG_RNDV`: send using rendezvous protocol regardless of the message length

- Receive callback flags (passed to `ucp_am_recv_callback_t` in `param->recv_attrs`) :
  - `UCP_AM_RECV_ATTR_FIELD_REPLY_EP`: indicates that `param->reply_ep` contains valid ep associated with the sender. Lifetime of this ep is the scope of the receive callback.
  - `UCP_AM_RECV_ATTR_FLAG_DATA`: Indicates that received data pointer is persistent, receiver may keep using it outside the callback (to free this data later need to call `ucp_am_data_release()`). Mutually exclusive with `UCP_AM_RECV_ATTR_FLAG_RNDV`.
  - `UCP_AM_RECV_ATTR_FLAG_RNDV`: Indicates that `data` argument is not a real data, but a descriptor needed to initiate rendezvous request. Mutually exclusive with `UCP_AM_RECV_ATTR_FLAG_DATA`.

Possible return codes from the AM callback

Flags	UCP_AM_RECV_ATTR_FLAG_DATA	UCP_AM_RECV_ATTR_FLAG_RNDV	No Flag
Keep data	UCS_IN_PROGRESS	UCS_IN_PROGRESS	Not allowed
Release data	UCS_OK	UCS_OK	UCS_OK

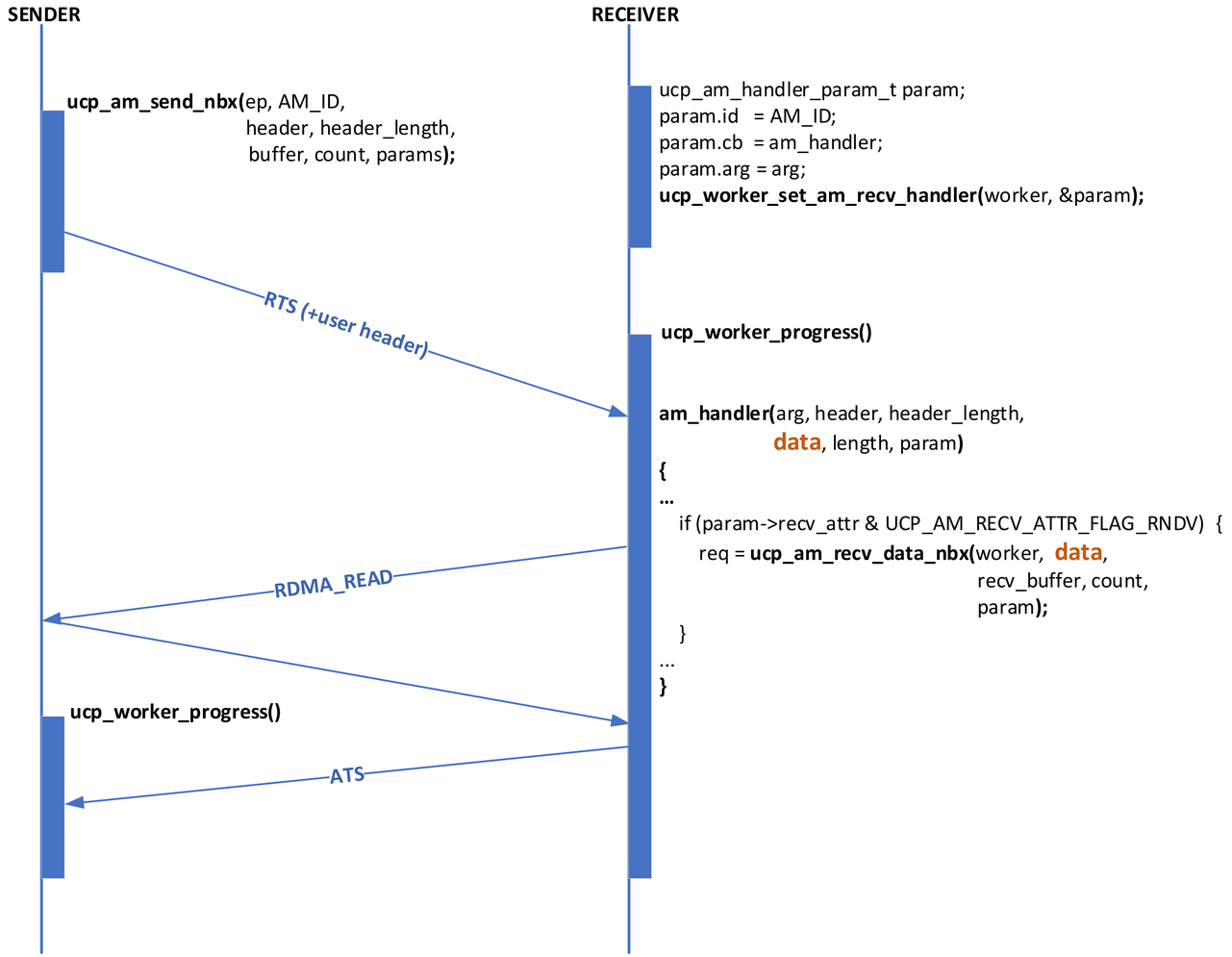
ACTIVE MESSAGE API

Rendezvous protocol

- Sender sends RTS with user header, not real data
- By default, send protocol is selected implicitly for the user (eager or rendezvous)
- Send protocol can be specified explicitly by passing `UCP_AM_SEND_FLAG_EAGER/RNDV` flag to `ucp_am_send_nbx()` routine
- On the receiver data callback is invoked with:
  - `UCP_AM_RECV_ATTR_FLAG_RNDV` flag
  - `data` argument is not a real data, but special descriptor
- Once receive buffer is ready, receiver may initiate rendezvous receive by

```
ucs_status_ptr_t
ucp_am_recv_data_nbx(ucp_worker_h worker,
                     void *data_desc,
                        void *buffer,
                        size_t count,
                        const ucp_request_param_t *param);
```

where `data_desc` is `data` argument received in am callback



FUTURE WORK

Multi-fragment eager protocol

- UCP maximal eager message size is limited
- Eager messages bigger than the limit are sent in multiple fragments
- With current AM API, UCP invokes AM handler once, also for multi-frag message:
  - UCP allocates an internal buffer to assemble the message fragments
  - Severe performance overheads: extra memory allocation and copy
- UCP AM API extension is proposed in UCX PR #7055:
  - Introduce “per-chunk” mode, when AM handler gets invoked for every arrived fragment
    Supports zero-copy on the receiver even for multi-fragmented message
  - Allow user to invoke ucp_am_recv_data_nbx() for the first chunk to provide UCP a buffer to be used for assembling a message
    Eliminates memory copy if user need to store incoming message in its own buffer

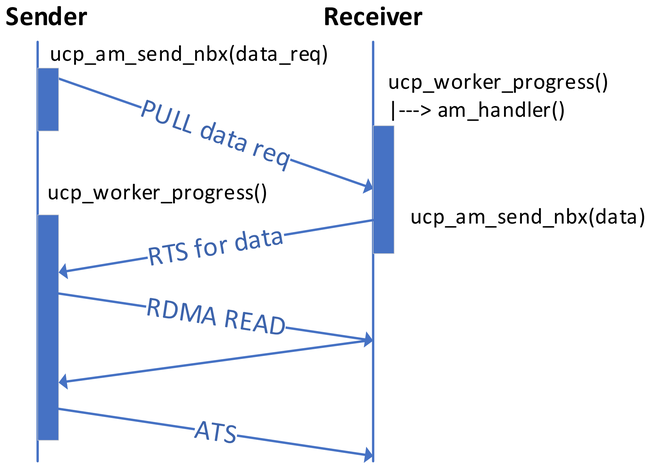
FUTURE WORK

Data fetch API

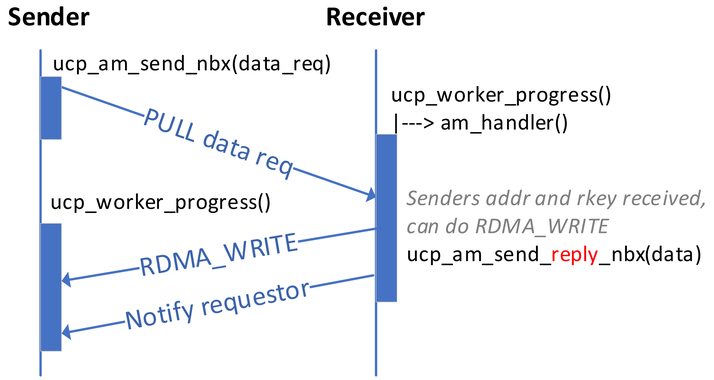
- Support for “read” (io-demo) or “pull” (ps-lite) use cases (sender wants to fetch data from the receiver)
- The goal is to minimize number of auxiliary protocol messages:
  - Requestor (sender) can send its address and rkey in the initial “pull” request
  - Responder (receiver) can issue RDMA_WRITE based on the details received in “pull” request
- Initial design proposed in UCX PR#5594 (currently closed)

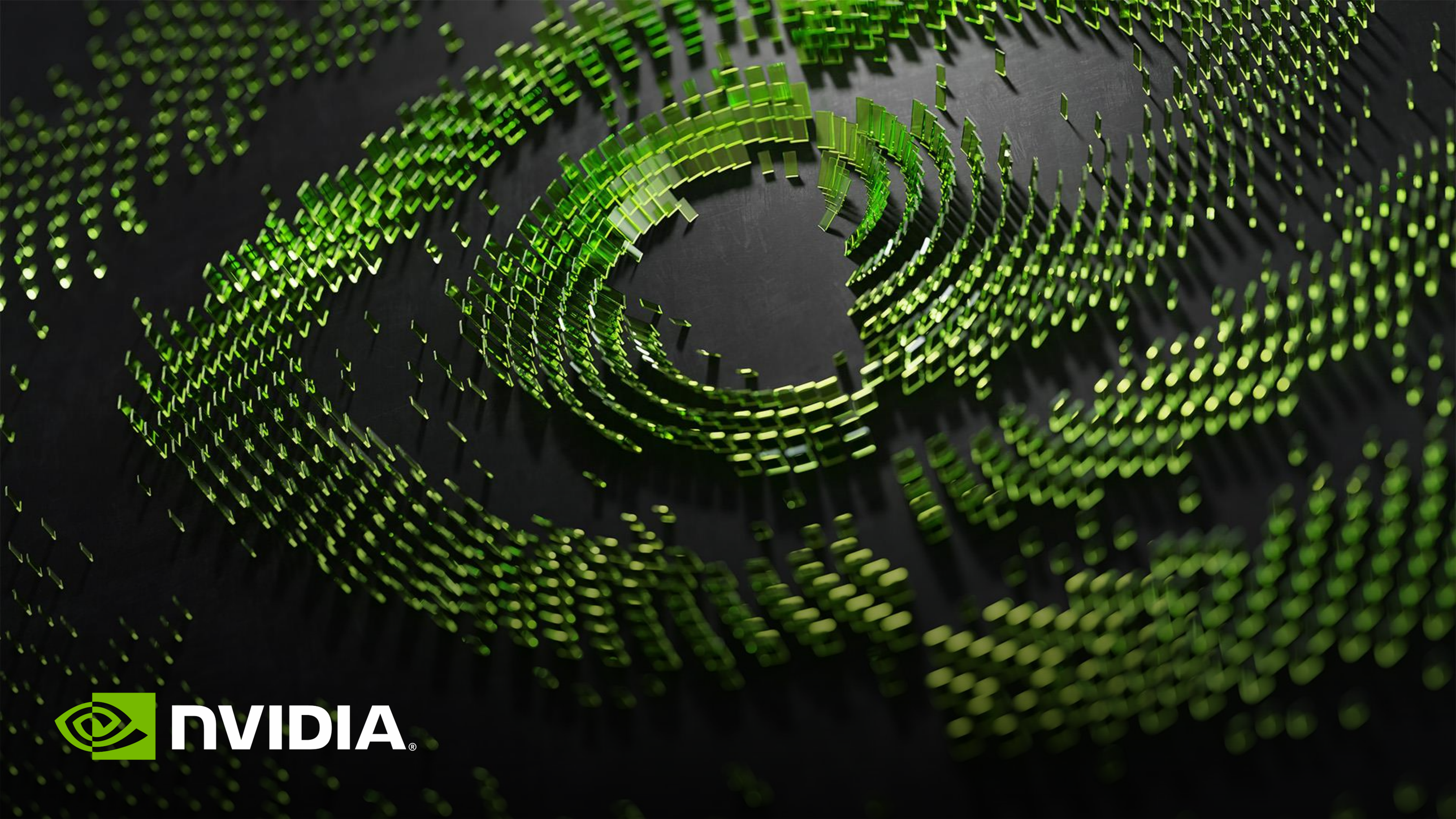
- Pros:
  - Performance improvement for “pull” patterns
  - More flexible API, no need to mix APIs
  - No need for explicit memory management (register/deregister, keys exchange)
- Cons:
  - Complicates UCP AM API, can be confusing, e.g.:
    - How to provide receive buffer count and datatype to **`ucp_am_send_nbx()`** routine
    - What arguments to define in new **`ucp_am_send_reply_nbx()`**routine
  - Similar optimization pattern can be implemented by using mix of AM and RMA APIs

Current flow



Proposed flow




NVIDIA