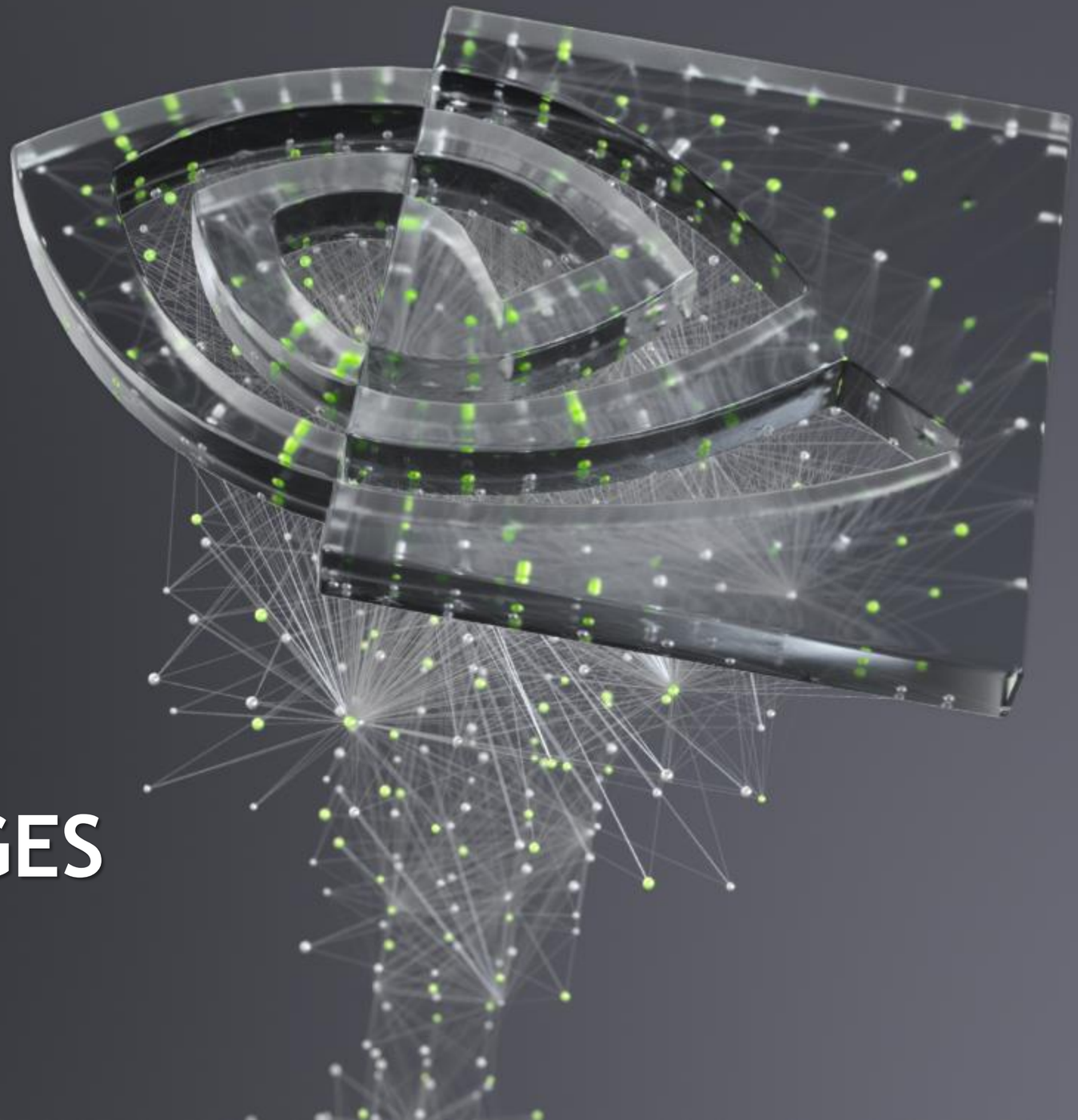


UCX ACTIVE MESSAGES

December 2020

USE CASES

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

AM PROS AND CONS

- Pros:
  - Simple API
  - Zero copy on receiver even with eager protocol (pointer to the network buffer is passed to the data callback)
  - Less overhead than TAG API (no tag matching)
  - Easier error handling, because of point-to-point semantic
- Cons:
  - User must handle (or queue) the message in the callback
  - Can’t receive eager message into specific user buffer/datatype or to GPU memory

AM 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`

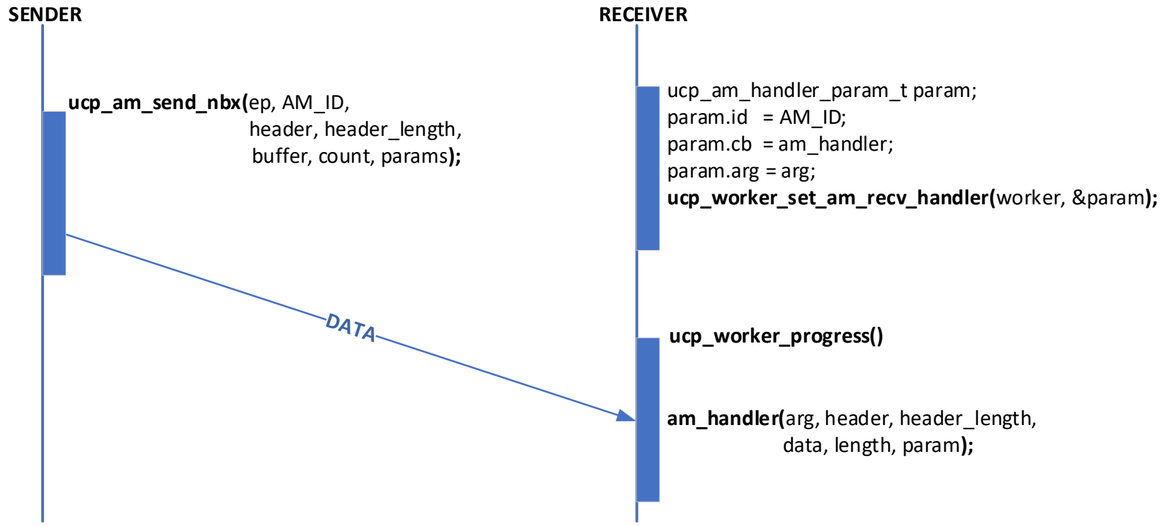
- Data handler semantic:

```
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);
```



AM BASIC API

- Send flags (passed to **`ucp_am_send_nbx()`** in **`param->flags`**):
  - **`UCP_AM_SEND_FLAG_REPLY`**: relevant reply ep 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

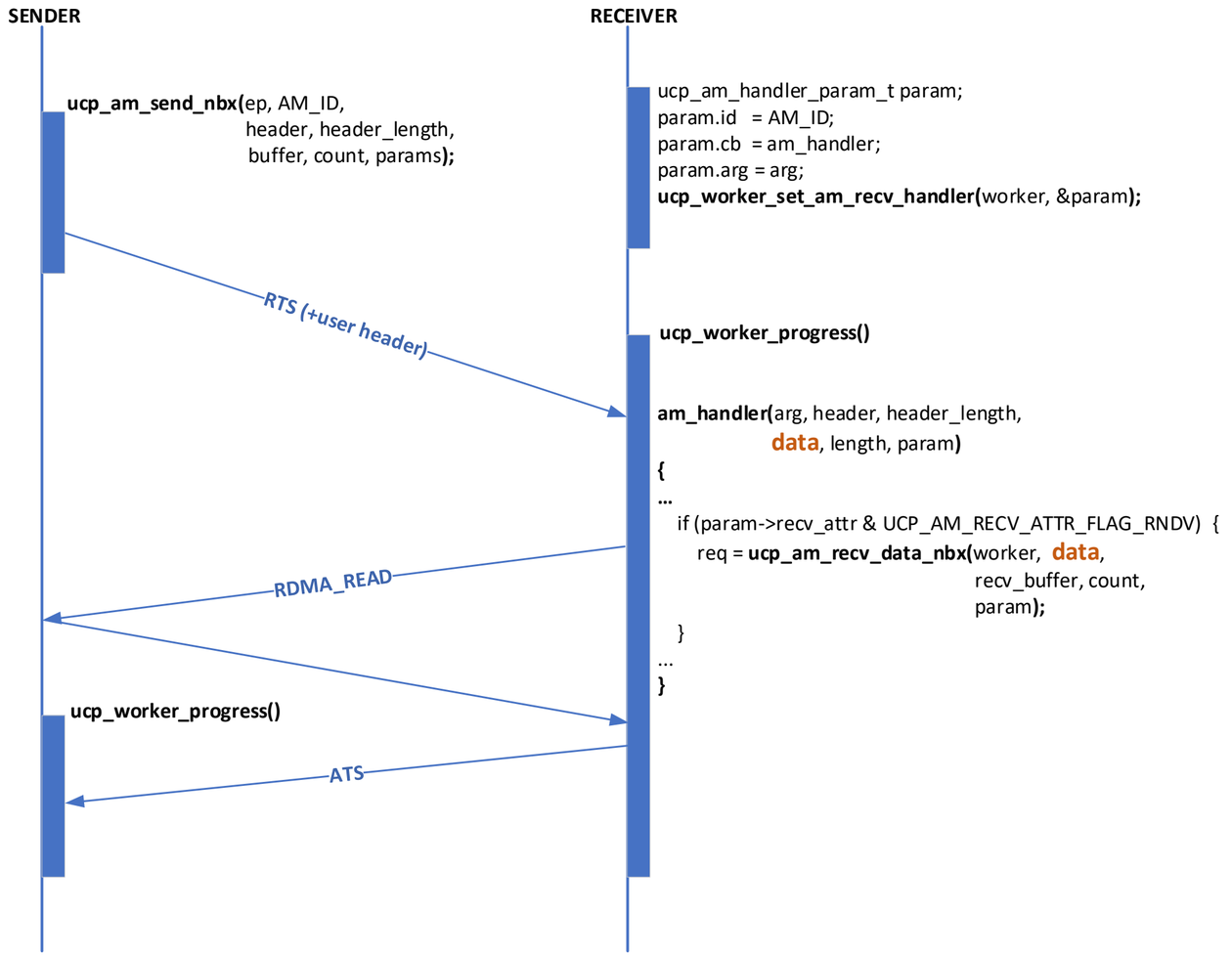
- Receive callback flags (passed to **`ucp_am_recv_callback_t`** in **`param->recv_attrs`**) :
  - **`UCP_AM_RECV_ATTR_FIELD_REPLY_EP`**: indicates that **`param->reply_ep`** contains valid ep associated with the sender. Lifetime of this ep is the scope of the receive callback.
  - **`UCP_AM_RECV_ATTR_FLAG_DATA`**: Indicate 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`**.

AM RENDEZVOUS

- 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

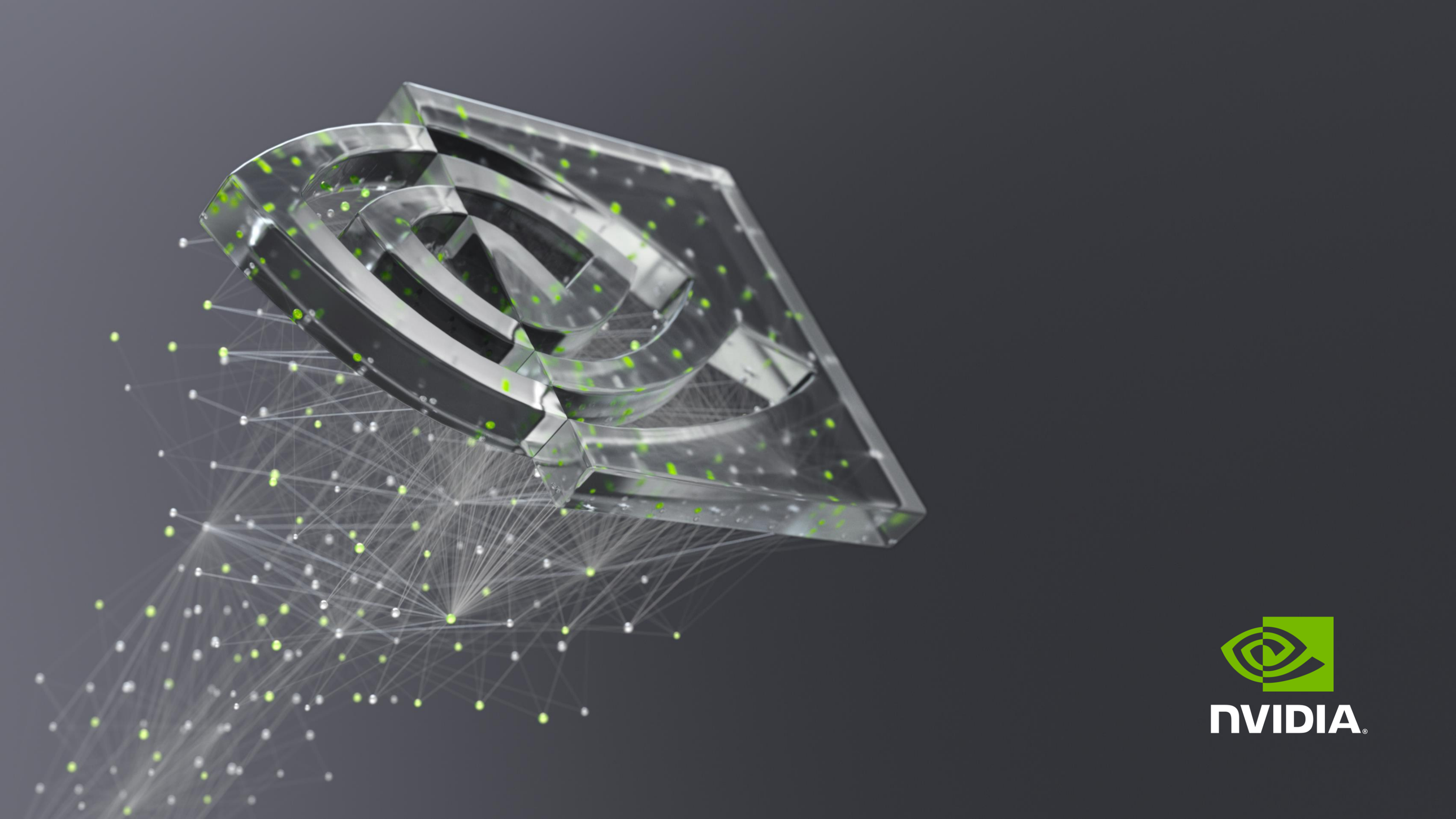
```
ucs_status_ptr_t
ucp_am_recv_data_nbx(ucp_worker_h worker, void *data_desc,
                     void *buffer, size_t count,
                     const ucp_request_param_t *param);
```

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



GPU SUPPORT

- No efforts needed for rendezvous protocol (memory detection/transfer is done in `ucp_am_recv_data_nbx()`)
- With eager protocol data always arrives to CPU memory
- Need some API to “unpack”/copy data to GPU buffer
- Can use `ucp_am_recv_data_nbx()` for eager data as well:
  - **Pros:**
    - single receive routine suitable for both eager and rendezvous protocol
    - can also be used for unpacking eager data to non-contig datatypes
  - **Cons:**
    - no easy way to handle inlined data (if UCP_AM_RECV_ATTR_FLAG_DATA is not specified in data callback)
  - **Proposal:**
    - define new flag for `ucp_worker_set_am_recv_handler`(), which will guarantee persistent data descriptor for every incoming eager message
    - adds some overhead for small messages, probably not that important with GPU memory


NVIDIA

FETCH API EXTENSION

- Proposed in UCX PR #5594
- The goal is to have a simple API for requesting data from the peer (needed for storage applications)
- Can be achieved by mixing existing APIs(TAG + RMA or AM + RMA), but:
  - Performance overheads(extra latency for aux messages, memory management routines, etc)
  - Complex code (mixing APIs, memory management routines)
- API proposal:
  - New flag for **`ucp_am_send_nbx()`** – **`UCP_AM_SEND_FLAG_FETCH_DATA`**. If it is set, need to provide **`reply_buffer/count/datatype`** in params as well.
  - New flag for receive callback - **`UCP_AM_RECV_ATTR_FLAG_FETCH_DATA`**, will indicate that data request has arrived
  - New routine - **`ucp_am_send_reply_nbx()`** for sending data back to the sender, will accept **`data`** descriptor (similar to the existing rendezvous flow)

FETCH API EXTENSION

- Pros:
  - No need for explicit memory management (register/deregister, keys exchange)
  - Need just single AM API
  - Less auxiliary messages
- Cons:
  - API is consufing, many corner cases, like:
  - can’t send data together with fetch request (only header)
  - 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