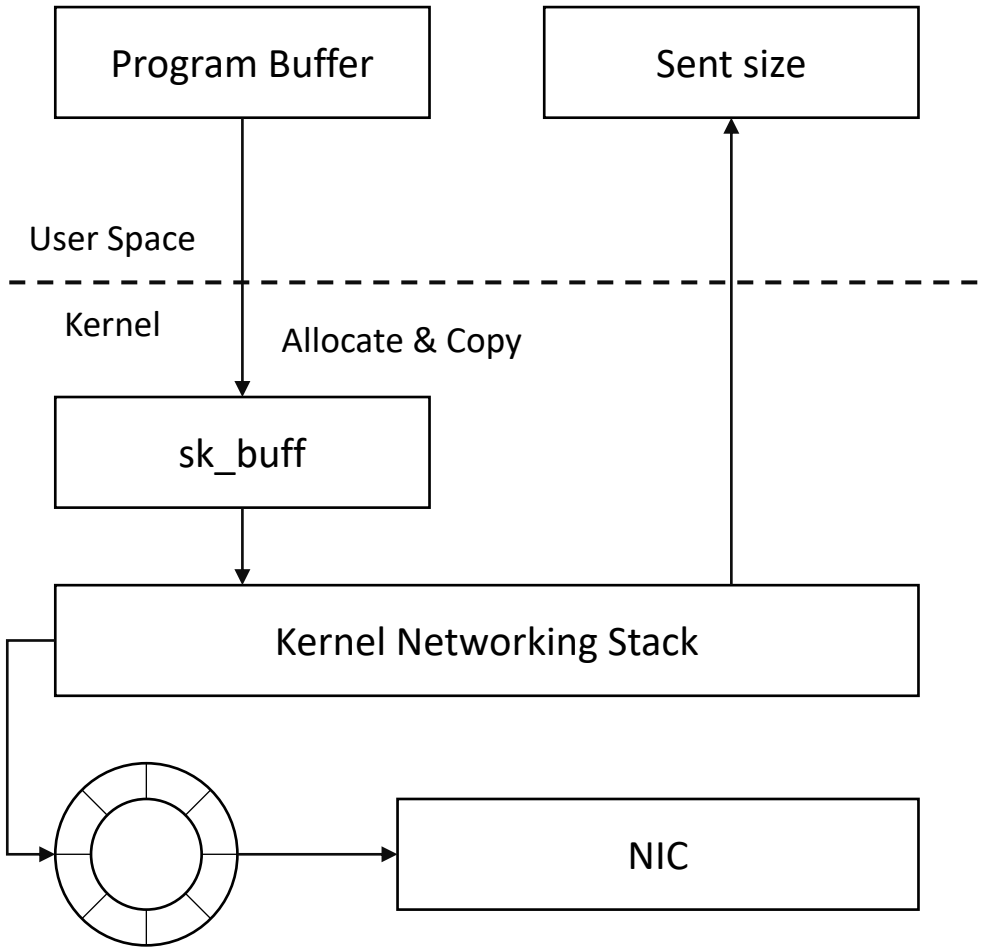
Understanding RDMA Programming

Junxue ZHANG

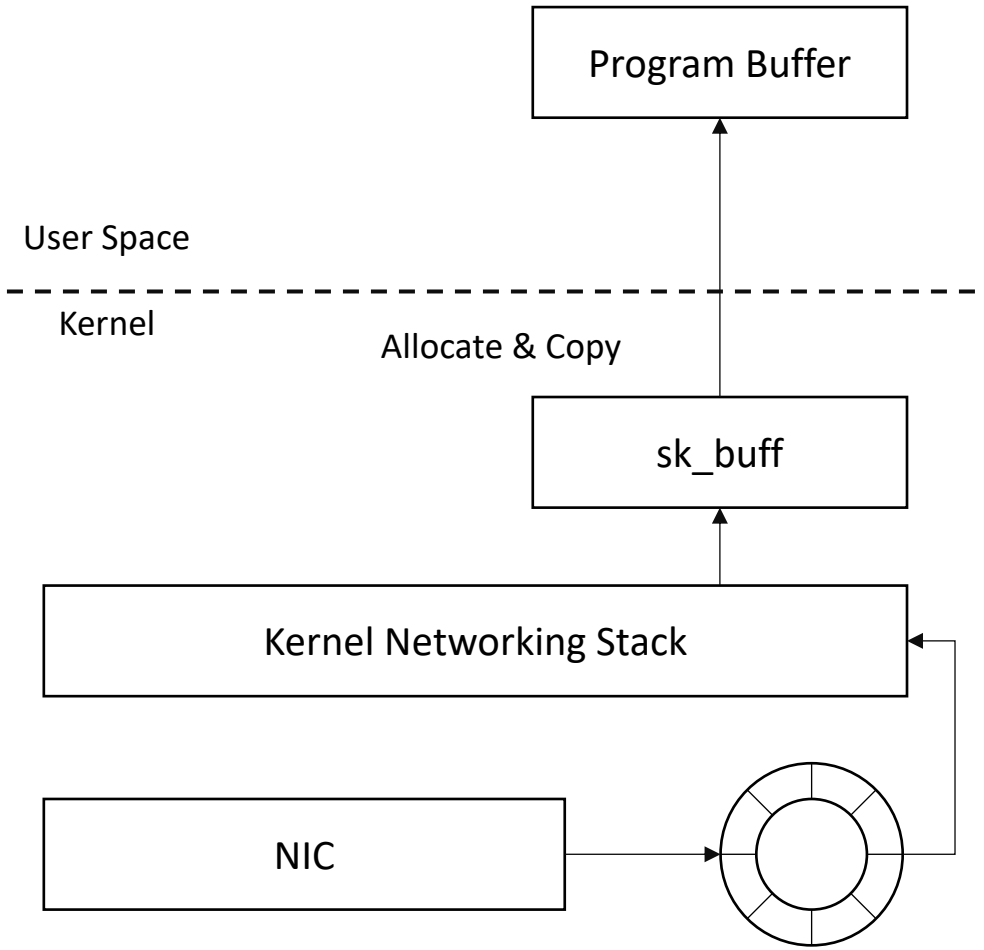
31-03-2019

The Evil of TCP

ssize_t send (int *sockfd*, **const void ***buf*, **size_t** *len*, **int** *flags*);**



ssize_t recv(int *sockfd*, **void ***buf*, **size_t** *len*, **int** *flags*);**

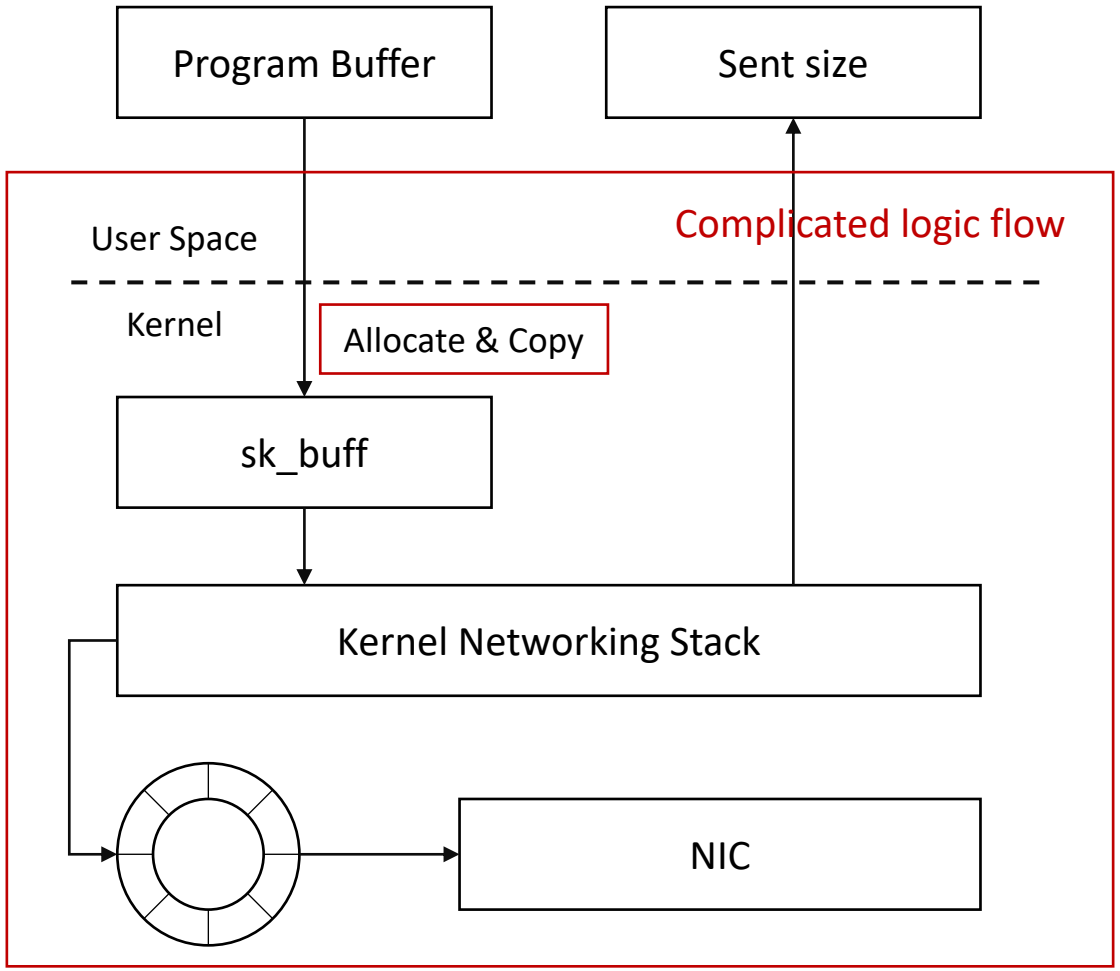


TCP cannot fit the ultra high-performance networking

Why ?

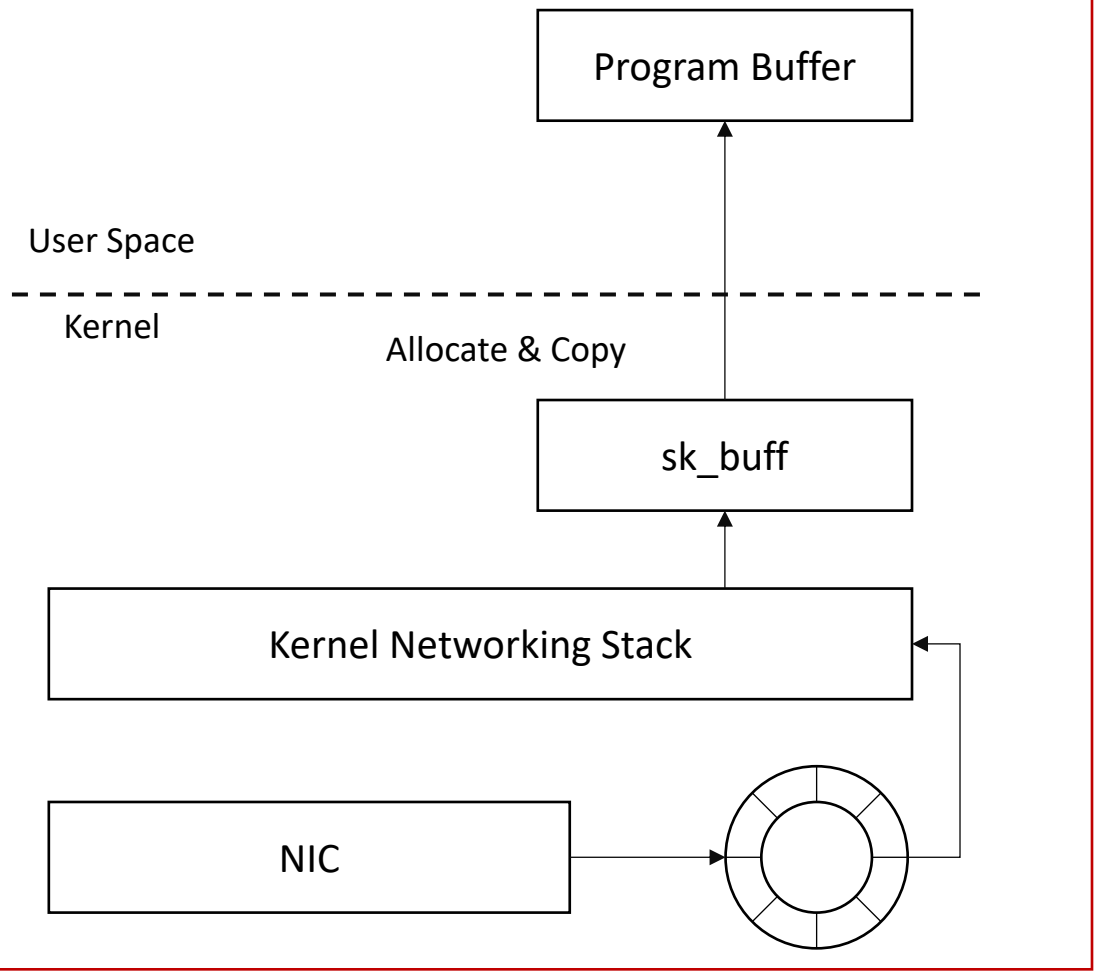
The Evil of TCP

ssize_t send (int *sockfd*, **const void ***buf*, **size_t** *len*, **int** *flags*);**



The receiver also need to participate

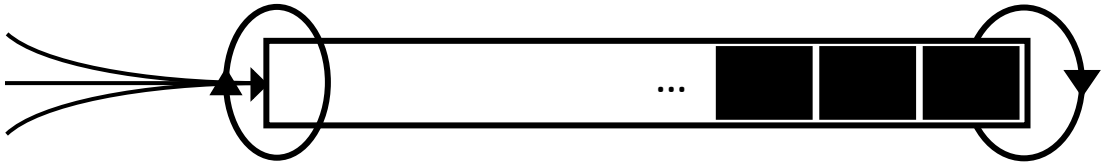
ssize_t recv(int *sockfd*, **void ***buf*, **size_t** *len*, **int** *flags*);**



TCP cannot fit the ultra high-performance networking

1. The complicated logic flow & block the caller → good data structure
2. Too much copy → memory management
3. Receiver always exists → one sided operation

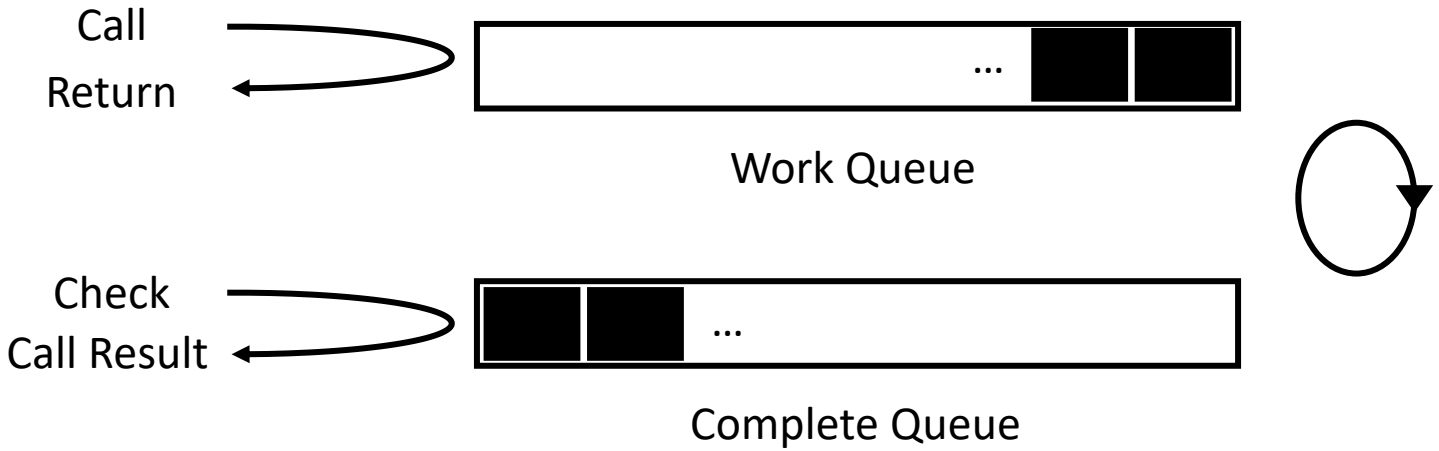
Decouple event start & end



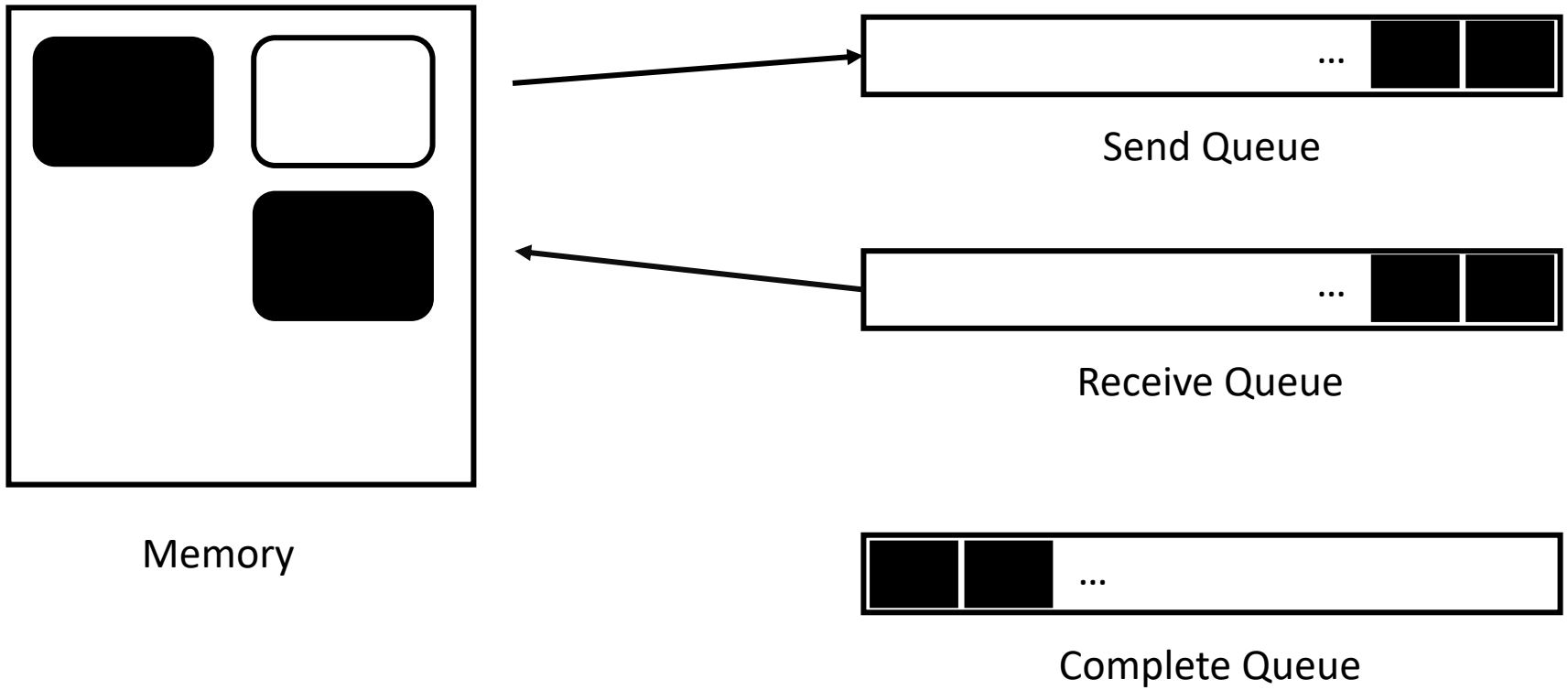
Thread 1

Thread 2

Queue is a good structure



Queue Pair in RDMA



Operating RDMA ≈ Add/Fetch elements in Queue Pairs

```
struct ibv_send_wr {
      uint64_t              wr_id;                /* User defined WR ID */
      struct ibv_send_wr    *next;                /* Pointer to next WR in list, NULL if last WR */
      struct ibv_sge        *sg_list;             /* Pointer to the s/g array */
      int                   num_sge;              /* Size of the s/g array */
      enum ibv_wr_opcode    opcode;               /* Operation type */
      int                   send_flags;           /* Flags of the WR properties */
      uint32_t              imm_data;             /* Immediate data (in network byte order) */
      union {
            struct {
                  uint64_t      remote_addr;      /* Start address of remote memory buffer */
                  uint32_t      rkey;             /* Key of the remote Memory Region */
            } rdma;
            struct {
                  uint64_t      remote_addr;      /* Start address of remote memory buffer */
                  uint64_t      compare_add;      /* Compare operand */
                  uint64_t      swap;             /* Swap operand */
                  uint32_t      rkey;             /* Key of the remote Memory Region */
            } atomic;
            struct {
                  struct ibv_ah  *ah;             /* Address handle (AH) for the remote node address */
                  uint32_t      remote_qpn;       /* QP number of the destination QP */
                  uint32_t      remote_qkey;      /* Q_Key number of the destination QP */
            } ud;
      } wr;
};
```

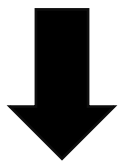
Operating RDMA ≈ Add/Fetch elements in Queue Pairs

```
struct ibv_sge {
        uint64_t        addr;       /* Start address of the local memory buffer */
        uint32_t        length;     /* Length of the buffer */
        uint32_t        lkey;       /* Key of the local Memory Region */
};
```

Opcode:

IBV_WR_SEND
IBV_WR_SEND_WITH_IMM
IBV_WR_RDMA_WRITE
IBV_WR_RDMA_WRITE_WITH_IMM
IBV_WR_RDMA_READ
IBV_WR_ATOMIC_CMP_AND_SWP
IBV_WR_ATOMIC_FETCH_AND_ADD

int ibv_post_send (struct ibv_qp **qp*, **struct ibv_send_wr** **wr*, **struct ibv_send_wr** ***bad_wr*);



Put the work queue element (WQE) into the send queue

Check Completion

int ibv_poll_cq (**struct ibv_cq** **cq*, **int** *num_entries*, **struct ibv_wc** **wc*);

```
do {
        int num_comp = ibv_poll_cq(cq, 1, &wc);
        if (num_comp < 0) {
                fprintf(stderr, "Failed to poll completions from the CQ: ret = %d\n", num_comp);
        return -1;
        }

        /* there may be an extra event with no completion in the CQ */
        if (num_comp == 0)
                continue;

        if (wc.status != IBV_WC_SUCCESS) {
                fprintf(stderr, "Completion with status 0x%x was found\n", wc.status);
                return -1;
        }
} while (num_comp);
```

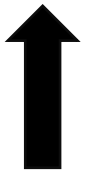
Avoid Unnecessary Copy

Directly move data from one address to another (remote) → safety concerns

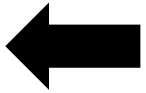
Memory should be registered before using in RDMA

uint32_t lkey; (local key)	uint32_t rkey; (remote key)

struct ibv_mr *ibv_reg_mr (**struct ibv_pd** **pd*, **void** **addr*, **size_t** *length*, **enum ibv_access_flags** *access*);



struct ibv_pd *ibv_alloc_pd (**struct ibv_context** **context*);



Create a group that objects inside can work together

Problems: Memory Management

Register memory → Cost time

Register memory before each RDMA operation → 1. Very Slow; 2.Multi-round communication
e.g. Early version of Gloo

Register large memory region → Manage memory by yourself
1. Ring Buffer
2. malloc & free → BFC allocator

Integrate with the application memory management system → avoid memory copy

Send & Receive vs RDMA

Sender: Put WQE in the sending queue

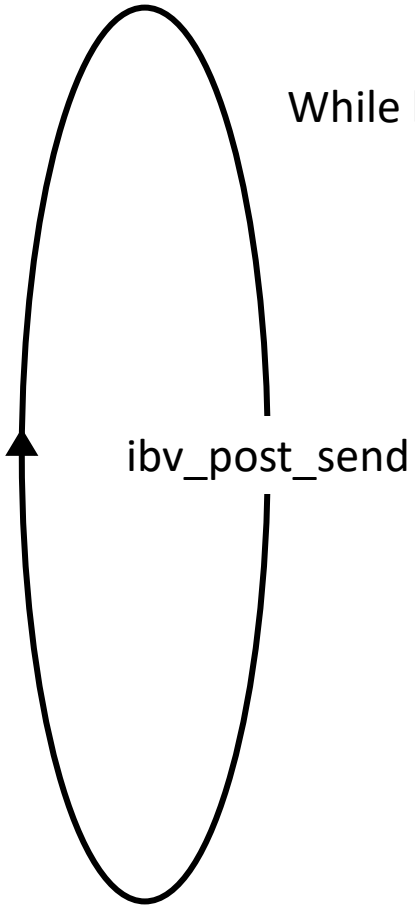
```
struct ibv_send_wr wr, *bad_wr = NULL;
struct ibv_sge sge;

wr.wr_id = (uintptr_t)conn;
wr.opcode = IBV_WR_SEND;
wr.sg_list = &sge;
wr.num_sge = 1;
wr.send_flags = IBV_SEND_SIGNALED;

sge.addr = (uintptr_t)send_addr;
sge.length = BUFFER_SIZE;
sge.lkey = lkey;

ibv_post_send(conn->qp, &wr, &bad_wr)
```

Receiver: Put WQE in the receiving queue

```
struct ibv_recv_wr wr, *bad_wr = NULL;
struct ibv_sge sge;

wr.wr_id = (uintptr_t)conn;
wr.next = NULL;
wr.sg_list = &sge;
wr.num_sge = 1;

sge.addr = (uintptr_t) receive_addr;
sge.length = BUFFER_SIZE;
sge.lkey = lkey;

ibv_post_recv(conn->qp, &wr, &bad_wr);
```

Flow Control

Sender: Put WQE in the sending queue



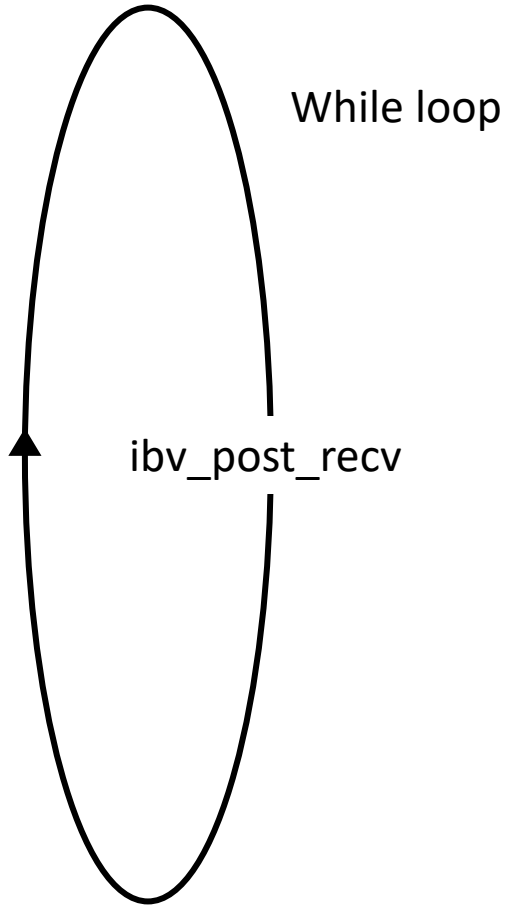
What if the sender post_send a WQE before receiver post_recv a WQE ? → Error occurs

post_recv some WQE in advance

How many ?

Flow Control strategy

Receiver: Put WQE in the receiving queue



Send & Receive vs RDMA

```
struct ibv_send_wr wr, *bad_wr = NULL;
struct ibv_sge sge;

wr.wr_id = (uintptr_t)conn;
wr.opcode = IBV_WR_RDMA_WRITE;
wr.sg_list = &sge;
wr.num_sge = 1;
wr.send_flags = IBV_SEND_SIGNALED;
wr.wr.rdma.remote_addr = peer_mr_addr;
wr.wr.rdma.rkey = peer_mr_rkey;
```

How do we know these information ?

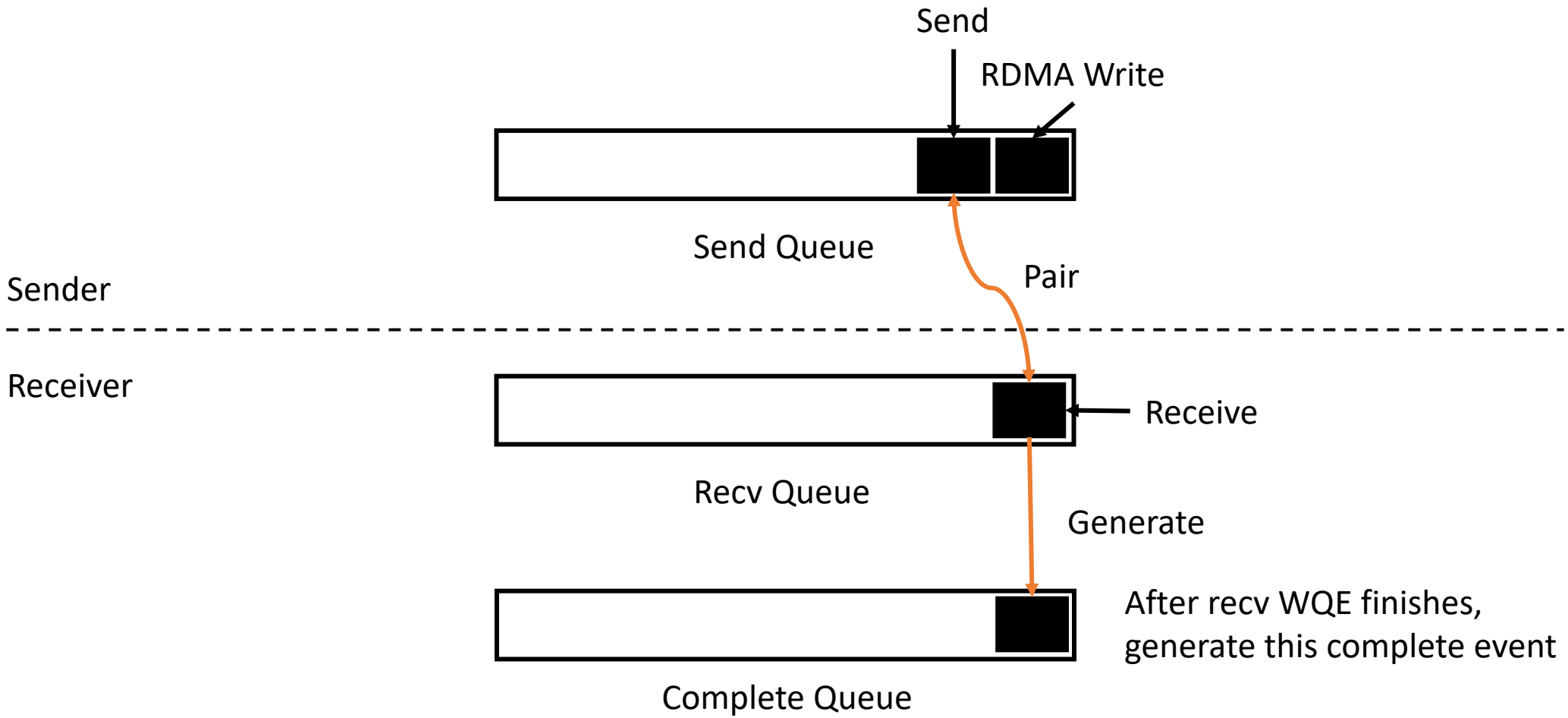
```
sge.addr = local_region;
sge.length = RDMA_BUFFER_SIZE;
sge.lkey = local_lkey;
```

No extra code needed
No CPU involved

Out of band communication

Problems

How to let the receiver know the RDMA operation has done ?



Understanding RDMA from Memory Perspective

Why ?
Because RDMA Integrate directly with memory management of an application
RDMA is not just a networking stack
RDMA is more like a tool connecting memory of different machines

Best practice:
remote_malloc() & remote_free()
To achieve 100% 0-copy data flow

https://github.com/tarickb/the-geek-in-the-corner

Thanks