OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP

# OFVWG: User-space Memory Windows

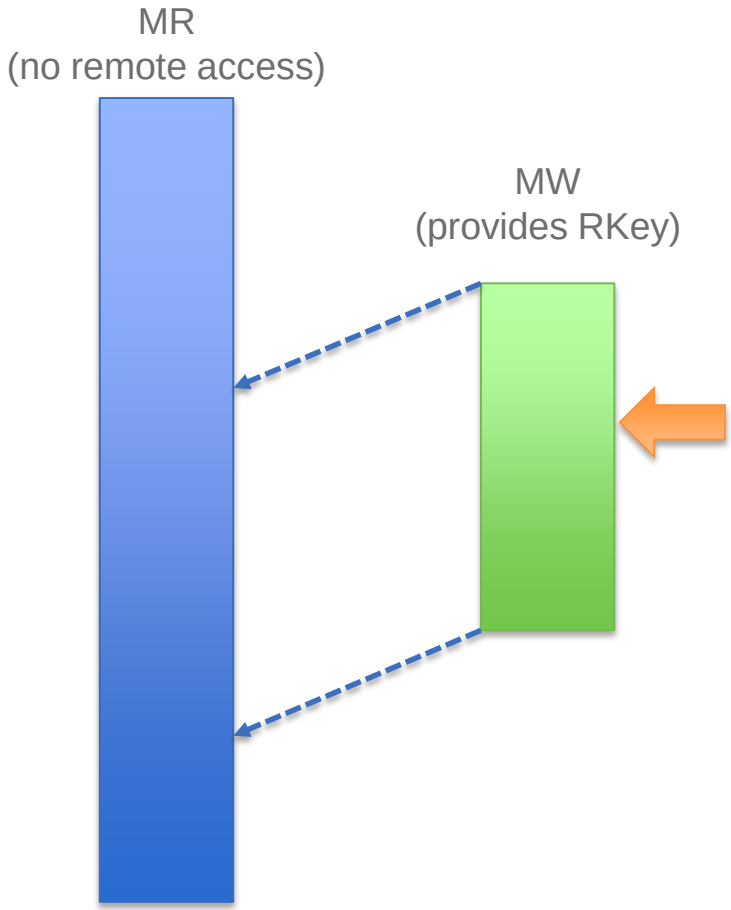
Yishai Hadas, Liran Liss

# Agenda

- Introduction – what are Memory Windows?
  - Motivation and use-cases
  - Window types: Type 1, Type 2A/2B
- User APIs
  - Device capabilities
  - Memory window allocation
  - Type1 Bind and Unbind
  - Type2 Bind and Invalidation

# What are windows?

- Motivation: efficient, safe remote transactions
  - Register once local memory buffer with no remote access
  - Provide peer proper access rights and aperture only throughout the duration of the expected transaction
- Open a “window” into an existing MR with elevated remote rights
  - Does not alter or consume new translation resources
  - Not a privileged operation
    - Accomplished through posting a WR to the Send Queue



# Type1 and type2

- Type1
  - MW associated only with an MR
  - CA owns Rkey
- Type2
  - MW associated with both an MR ***and*** a QP
    - Granular, per-connection access enforcement
  - Consumer owns ***key*** portion of the RKey
  - Supports ***remote*** invalidation
  - Supports zero-based virtual addresses (ZBVA)

| | Type 1 | Type 2 |
|---|---|---|
| **ZBVA** | No | Yes |
| **Send with invalidate** | No | Yes |
| **Local invalidate** | No | Yes |
| **Bind MW via standard PostSend() Verb** | No | Yes |
| **Bind MW via dedicated BindMW() Verb** | Yes | No |
| **Key ownership** | CI | Consumer |

# Type 2A/2B

- Differ in destruction semantics
  - Type2A MW requires unbinding all MWs before destroying a QP
  - Type2B MW allows destroying QPs with bound MWs
    - PD checks are added avoid misuse if QP allocated to another process
- A CA can support either one, but not both

| | Type 2A | Type 2B |
|---|---|---|
| **Post Bind** | Same PD | Same PD |
| **Invalidate** | Same QP | Same PD and QP |
| **MW Access** | Same QP | Same PD and QP |
| **QP Destruction** | First unbind/destroy all type 2A MW | Even with type 2B MW associated |

# Device Capability APIs

- IBV_DEVICE_MEM_WINDOW
  - Support for Type1
- IBV_DEVICE_MEM_MGT_EXTENSIONS
  - Support Type2 Windows
  - Lots of other stuff (mainly privileged Verbs)
- If Type2 windows are supported, an indication of either
  - IBV_DEVICE_MEM_WINDOW_TYPE_2A
  - IBV_DEVICE_MEM_WINDOW_TYPE_2B

# Allocation APIs

```
enum ibv_mw_type {
        IBV_MW_TYPE_1                   = 1,
        IBV_MW_TYPE_2                   = 2
};

struct ibv_mw {
        struct ibv_context     *context;
        struct ibv_pd          *pd;
        uint32_t                rkey;
        uint32_t                handle;
        enum ibv_mw_type        type;
};

struct ibv_mw *ibv_alloc_mw(struct ibv_pd *pd,
                            enum ibv_mw_type type);

int ibv_dealloc_mw(struct ibv_mw *mw);
```

- No need to distinguish between 2A/2B
  - Implied by CA capabilities

# Type1 Bind/Unbind APIs

```
struct ibv_mw_bind_info {
        struct ibv_mr  *mr;             /* The MR to bind the MW to */
        uint64_t       addr;            /* The address the MW should start at */
        uint64_t       length;          /* The length (in bytes) the MW should span */
        int            mw_access_flags; /* Access flags to the MW. Use ibv_access_flags */
};

struct ibv_mw_bind {
        uint64_t                wr_id;
        int                     send_flags;
        struct ibv_mw_bind_info bind_info;
};

int ibv_bind_mw(struct ibv_qp *qp, struct ibv_mw *mw, struct ibv_mw_bind *mw_bind);

enum ibv_wc_opcode {      ...
       IBV_WC_BIND_MW     ...
};
```

- Posting a Bind work request via dedicated Verb
  - ibv_bind_mw()
- Allows consumer to observe CA assigned RKey immediately
  - No need to wait for Bind completion

# Type1 Bind/Unbind APIs (cont.)

- Access flags
  - IBV_ACCESS_REMOTE_WRITE/READ/ATOMIC
- Send flags
  - IBV_SEND_FENCE
  - IBV_SEND_SIGNALED
- RKey returned in mw->rkey
  - Takes affect only after the operations successfully completes on the QP
  - Application may rely on WR ordering to send the Rkey
    - If the bind fails, the send will be flushed in error
- A MW may be continuously re-bound
  - Unbind performed by calling ibv_bind_mw() with length=0

# Type2 Bind APIs

```
enum ibv_wr_opcode {       ...
       IBV_WR_BIND_MW,     ...
};

struct ibv_send_wr { ...
       struct {
               struct ibv_mw   *mw;
               uint32_t                rkey;
               struct ibv_mw_bind_info bind_info;
       } bind_mw;
};

uint32_t ibv_inc_rkey(uint32_t rkey);
```

- Binding posted using ibv_post_send()
  - A MW may not be re-bound unless first invalidated
- User responsible for determining Rkey
  - By calling ibv_inc_rkey()

# Type2 Invalidation APIs

```
enum ibv_wr_opcode {            ...
       IBV_WR_LOCAL_INV,        ...
       IBV_WR_SEND_WITH_INV,    ...
};

enum ibv_wc_opcode { ...
       IBV_WC_LOCAL_INV, ...
};

enum ibv_wc_flags { ...
       IBV_WC_WITH_INV, ...
};
```

- RKey provided in WR immediate data
  - While posting local/remote invalidations
- RKey returned in WC immediate data
  - Indicated by IBV_WC_LOCAL_INV opcode for local invalidations
  - Indicated by IBV_WC_WITH_INV flag for remote invalidations

# Thank You

OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP