OPENFABRICS ALLIANCE

Accelerated Verbs

ANNUAL INTERNATIONAL OPENFABRICS SOFTWARE DEVELOPERS' WORKSHOP

Liran Liss

Mellanox Technologies

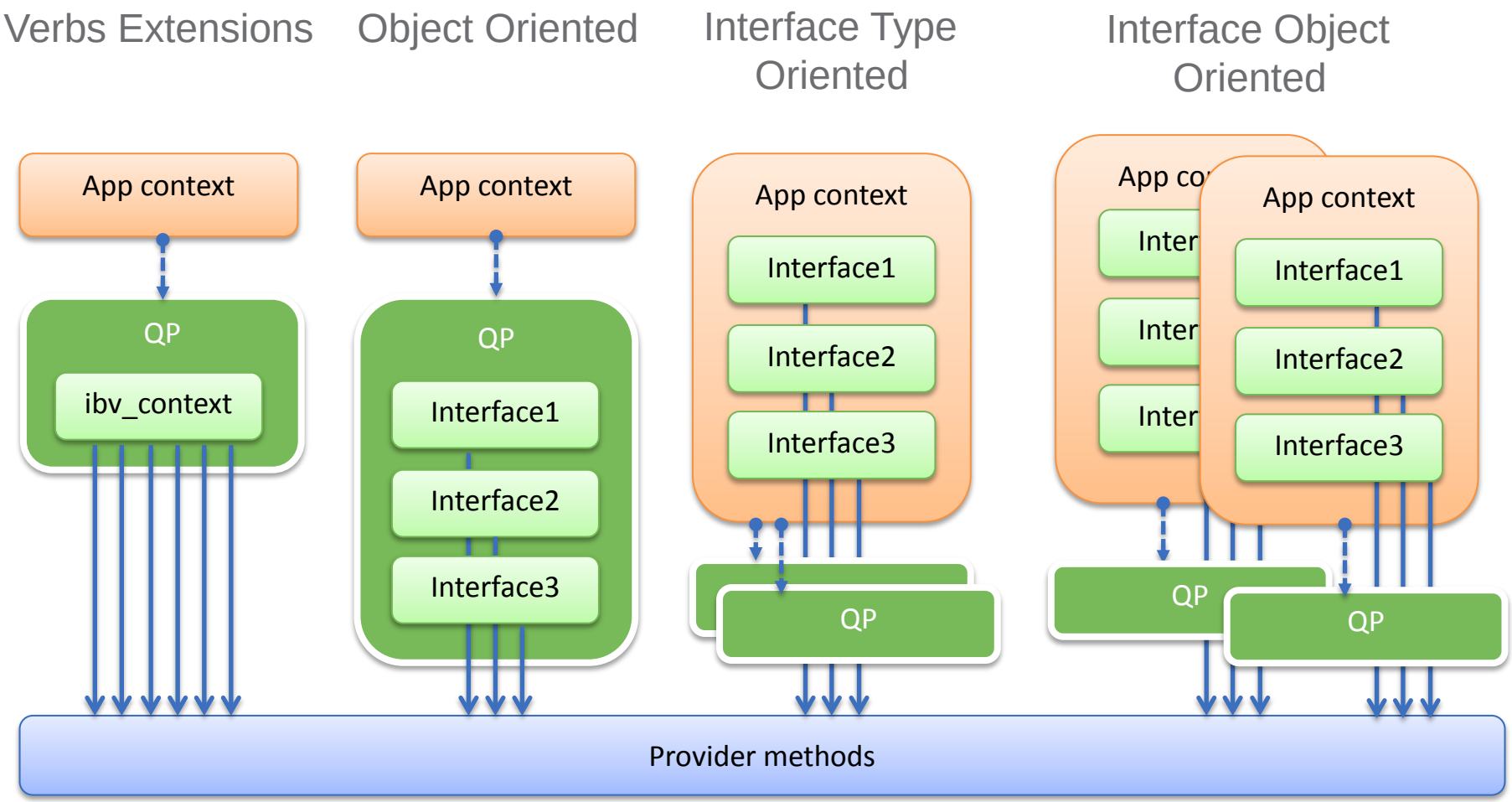
Agenda

- Introduction
- Possible approaches
- Querying and releasing interfaces
- Interface family examples
- Application usage example
- Conclusion

Introduction

- High-end apps require extremely optimized data-path APIs
  - HPC
  - Packet processing
- Goals
  - Provide the best possible performance (*)
  - Focus on the fast path
    - Dedicated functions for initiating IO
    - Dedicated functions for polling completions
  - Maintain 100% functional compatibility with existing Verbs
- Non-goals
  - Optimize all Verbs
  - Change the object model or semantics

Possible Approaches



Interface Object Oriented API

- Obtain accelerated function tables through a parameterized “Query Interface” Verb
  - Benefit from all Object Oriented goodies
    - Type checking
    - Specialization
  - Separate Interface acquirement from invocation
    - All versioning and capability checks done at acquirement time
    - Eliminate all overheads in fast path
  - Gradual addition, extension, and deprecation of interfaces

- Obtained interface valid for requested object only
  - Provider may choose to return the same interface for multiple objects

Query Interface

```
enum ibv_query_intf_flags {
      IBV_QUERY_INTF_FLAG_ENABLE_CHECKS     = (1 << 0),
};

enum ibv_intf_scope {
      IBV_INTF_GLOBAL,
      IBV_INTF_EXPERIMENTAL,
      IBV_INTF_VENDOR,
};

struct ibv_query_intf_params {
      uint32_t                    flags;
      enum ibv_intf_scope         intf_scope;
      uint64_t                    vendor_guid;     /* valid only for Vendor scope */
      uint32_t                    intf;
      uint32_t                    intf_version;
      void                        *obj;
      void                        *family_params;
      uint32_t                    family_flags;
      uint32_t                    comp_mask;       /* for future extensions */
};

static inline void *ibv_query_intf(struct ibv_context *context,
                                   struct ibv_query_intf_params *params,
                                   enum ibv_query_intf_status *status);

int ibv_release_intf(struct ibv_context *context, void *intf);
```

Query Interface (cont.)

```
enum ibv_query_intf_status {
      IBV_INTF_STAT_OK,
      IBV_NTF_STAT_VENDOR_NOT_SUPPORTED,
      IBV_INTF_STAT_INTF_NOT_SUPPORTED,
      IBV_INTF_STAT_VERSION_NOT_SUPPORTED,
      IBV_INTF_STAT_INVAL_PARARM,
      IBV_INTF_STAT_INVAL_OBJ,
};
```

- Notes
  - Interfaces do not introduce new functionality to Verbs
  - Each family is used only for a specific object type (QP, CQ, etc.)
  - If no object is passed the call serves as a capability query
    - Checks provider support for a given family and version
  - Version usage
    - Each increment must support all function calls of previous versions
    - Interface changes require a new family

Burst Family

- Targets Ethernet packet processing Apps
- Optimizes
  - Accumulating packets to be sent
  - Bursts of single SGE packets

```
struct ibv_qp_burst_family {
      int (*send_pending)(struct ibv_qp *qp, uint64_t addr, uint32_t length,
                          uint32_t lkey, uint32_t flags);

      int (*send_pending_inline)(struct ibv_qp *qp, void *addr, uint32_t length,
                          uint32_t flags);

      int (*send_pending_sg_list)(struct ibv_qp *qp, ibv_sge *sg_list,
                          uint32_t num, uint32_t flags);

      int (*send_flush)(struct ibv_qp *qp);

      /* Send/receive burst of single sge packets; 'num' indicates number of packets */
      int (*send_burst)(struct ibv_qp *qp, ibv_sge *sg_list, uint32_t num, uint32_t flags);
      int (*recv_burst)(struct ibv_qp *qp, ibv_sge *sg_list, uint32_t num);
};
```

Poll Family

- Targets minimal polling information
- Optimizes
  - Send completion counts
  - Receive completions for which only the length is of interest
  - Completions that contain the payload in the CQE

```
struct ibv_cq_poll_family {
      int (*poll_cnt)(struct ibv_cq *cq, uint32_t max);
      int (*poll_cq_length)(void* buf, uint32_t *inl);
};
```

Application Code

```
struct context {
      struct ibv_qp *qp;
      struct ibv_qp_ops_msg *msg;

      struct ibv_cq *cq;
      struct ibv_cq_formatted_ops *formatted;
};

int create_ctx(struct context *ctx)
{
      struct ibv_cq_attr cq_attr;
      ...
      ctx->qp = ibv_create_qp(...);
      ctx->msg = ibv_query_intf(ctx->context, ... /* QP channel family */);

      ctx->cq = ibv_create_cq(...);
      cq_attr.format_mask = IBV_WC_BASE | IBV_WC_TS;
      ibv_modify_cq(cq, cq_attr, IBV_CQ_FORMAT_MASK);

      ctx->formatted = ibv_query_intf(ctx->context, ... /* Formatted CQ family */);
      ...
}
```

Application Code (cont.)

```
int send_message(struct context *ctx)
{
      char my_msg[] = "blah";
      struct {
            struct ibv_wc_base base;
            struct ibv_wc_ts ts;
      } my_comp;
      int ret;

      ctx->msg->send_inline(ctx->qp, my_msg, sizeof my_msg, SEND_WR_ID);

      while ( !(ret = ctx->formatted->poll_formatted(ctx->cq, &my_comp, sizeof(my_comp))) );
      if (ret < 0) {
            /* Poll for error and bail out */
            ...
      }
      printf("wr_id:%d timestamp:%lld\n", my_comp.base.wr_id, my_comp.ts.timestamp);
      return ret;
}
```

Conclusion

- Verbs is a robust and efficient interface
  - Solid object model
  - Efficient data path
- Verbs extensions allow adding new APIs in a forward- and backward-compatible manner
  - The standard way to add new Verbs
- Accelerated Verbs provides highly optimized, domain-specific, data-path APIs
  - Always a strict subset of Verbs

Thank You

OPENFABRICS ALLIANCE

ANNUAL INTERNATIONAL OPENFABRICS SOFTWARE DEVELOPERS' WORKSHOP