So you want to do RDMA programming?

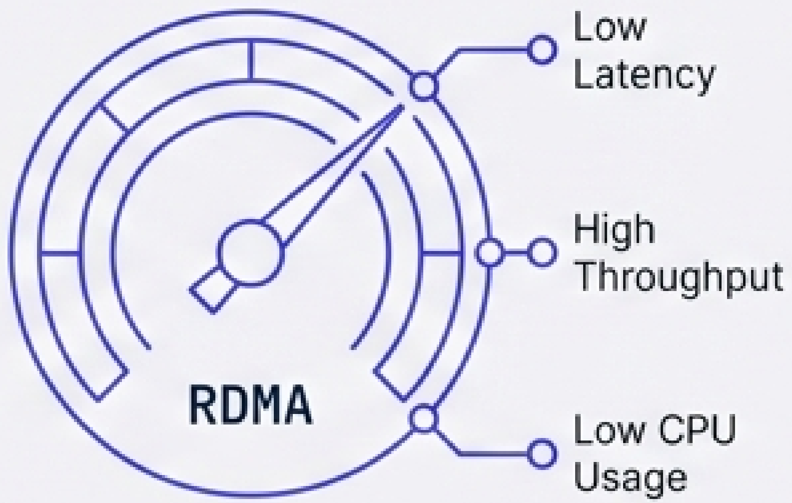
RTRS: A reliable high-speed transport library over RDMA

Haris Iqbal, IONOS SE
Jinpu Wang, IONOS SE

RDMA is powerful... but difficult.

The Standard

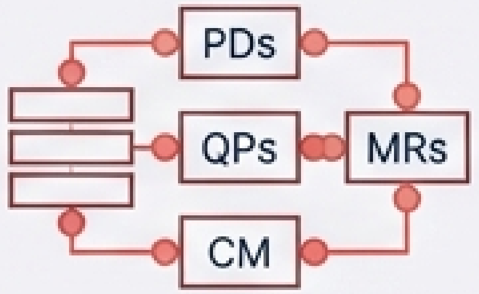
RDMA is the standard in Data Centers and HPC (Infiniband, RoCE, iWARP) due to its low latency, high throughput, and low CPU usage.



Key Takeaway: Many projects end up re-implementing the same RDMA transport layer to solve these issues repeatedly.

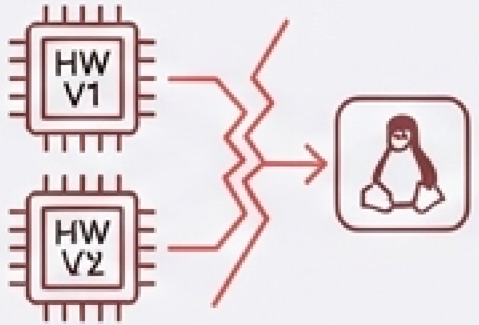
The Friction of Raw Verbs

- **Boilerplate**: Requires managing **PDs**, **QPs**, **CQs**, **MRs**, and **CM**.



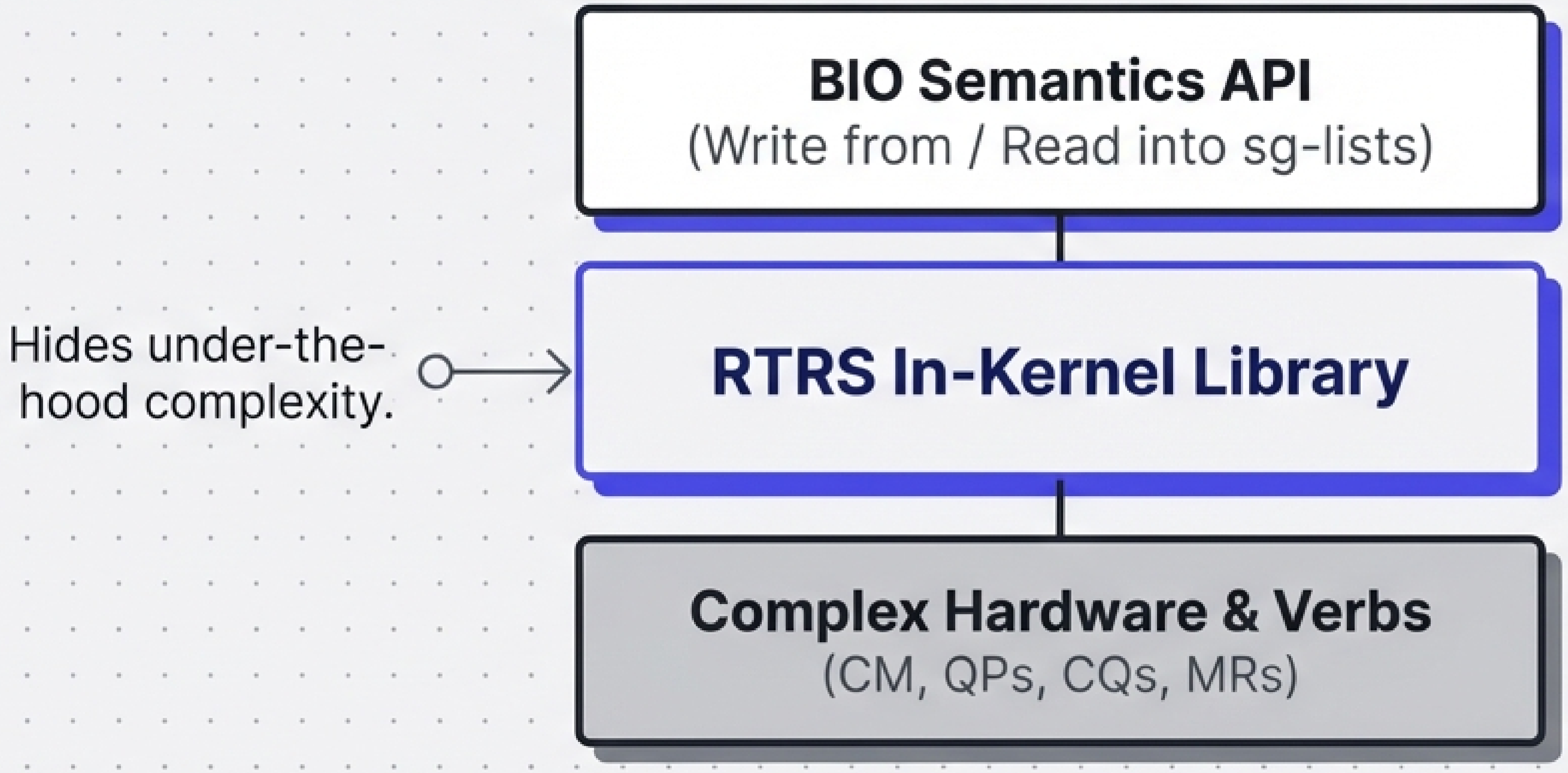
- **Stateful**: You must manually handle reconnects, timeouts, and error paths.

- **Sensitive**: Code is often sensitive to hardware and kernel versions.



Introducing RTRS

RDMA Transport (for Remote Storage)



An `in-kernel`, `reliable`, `high-speed` transport transport library designed specifically for `block-I/O` over `RDMA`.

The API is designed to feel familiar: `Write` from `sg-lists` to the remote side, `Read into sg-lists` from the remote side.

Architecture & Design Philosophy

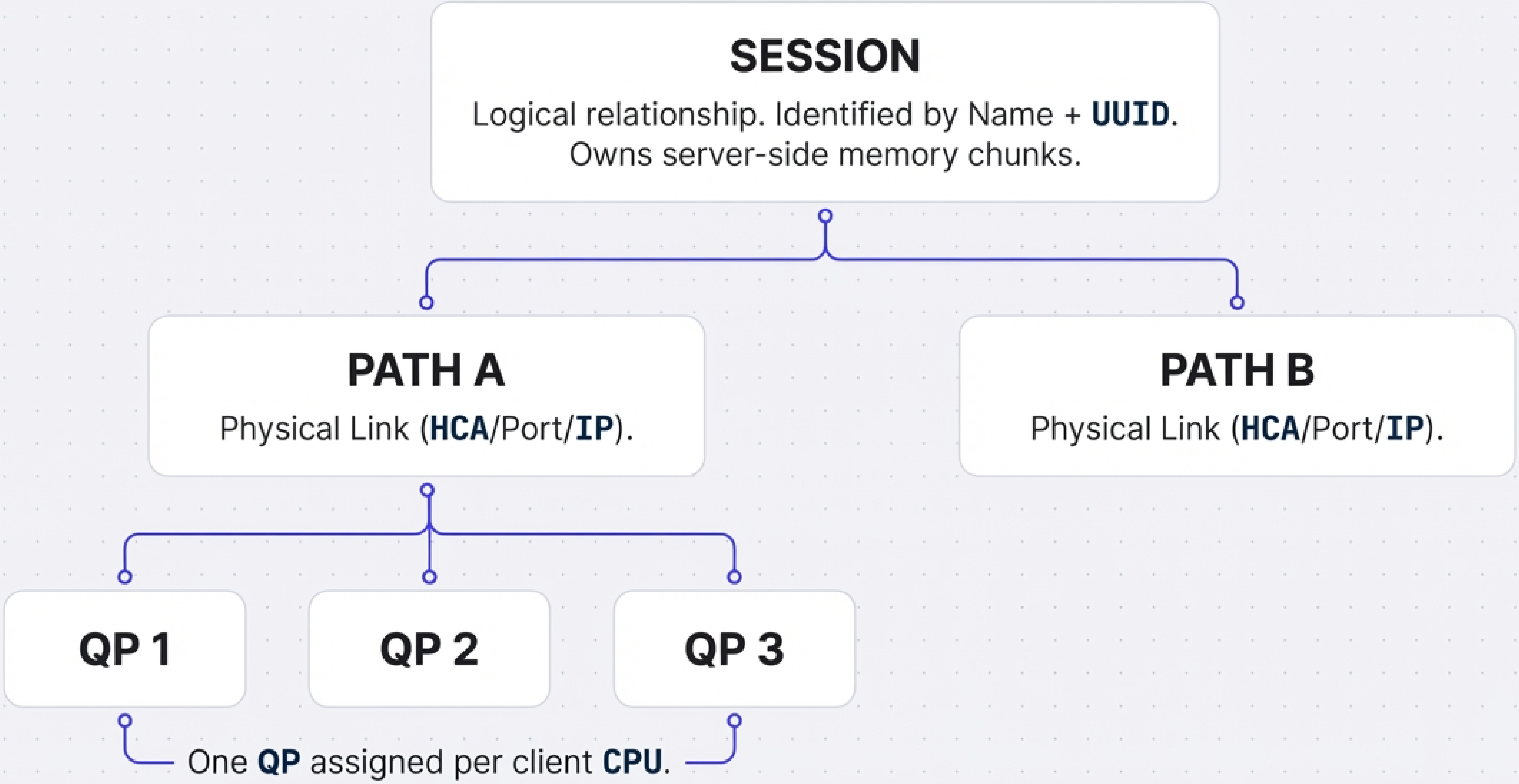
Module Structure

Connection Logic

- One path corresponds to one physical link.
- Optimized specifically to transfer (read/write) IO blocks.
- Utilizes IRQ pinning for efficient data transfer.

1 QP per CPU per path

Core Concepts: Session & Path



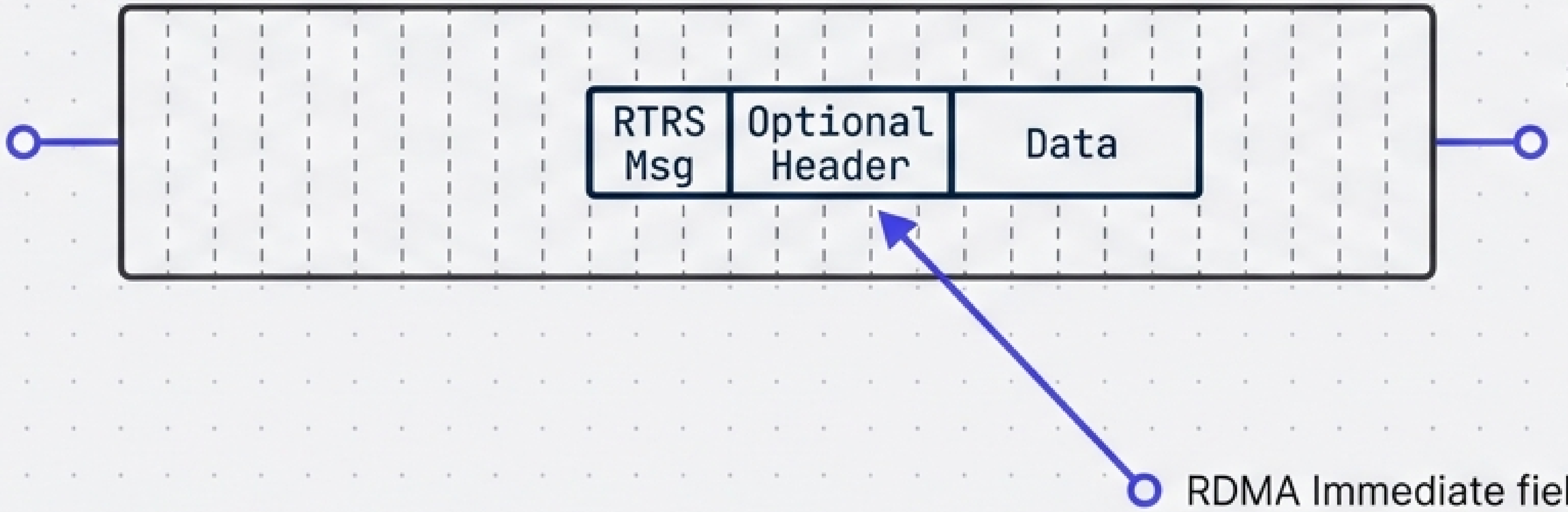
The I/O Model

Server-Side Memory (Pre-allocated & Pre-mapped)

User Perspective

Slate Gray Inter Medium

- User says: “**Write this `sg-list`**; call me back on completion.”
- User says: “**Read into this `sg-list`**; call me back on completion.”



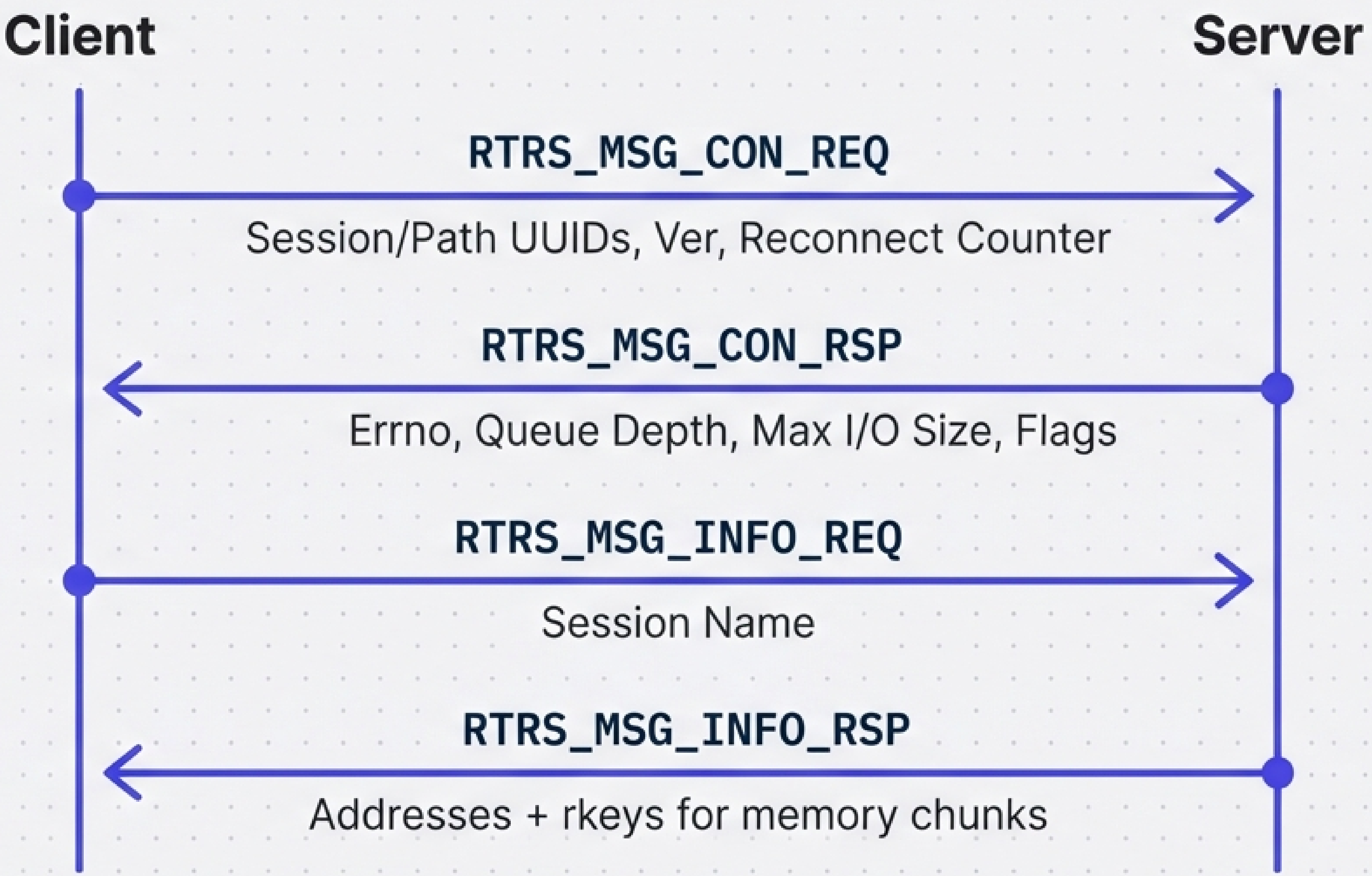
RTRS Perspective

Slate Gray Inter Medium

- RTRS packs message + header + data.
- RTRS hides details of RDMA `READ/WRITE`.

RDMA Immediate field identifies chunk, offset, and I/O ID.

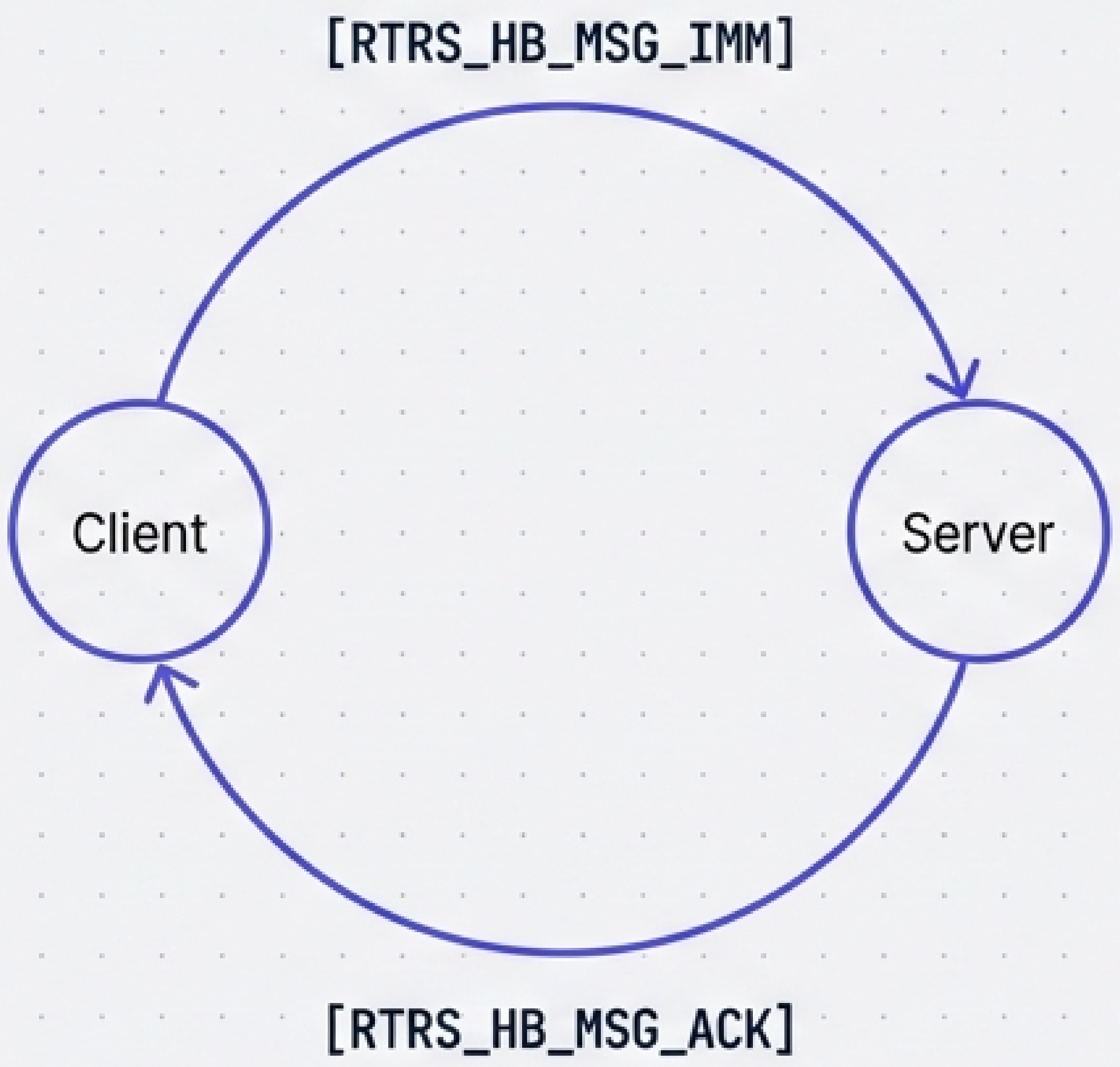
Protocol: Connection Establishment



This handshake occurs for each connection belonging to a path, and for each path.

Protocol: Heartbeats & Message Flow

Heartbeat Mechanism



I/O Message Flow

Write Flow

- Client sends: usr_data + usr_hdr + rtrs_msg_rdma_write
- Server receives: [RTRS_IO_REQ_IMM]
- Server responds: [RTRS_IO_RSP_IMM] (id + errno)

Read Flow

- Client sends: usr_hdr + rtrs_msg_rdma_read
- Server receives: [RTRS_IO_REQ_IMM]
- Server responds: [RTRS_IO_RSP_IMM] with usr_data + (id + errno)

Invalidation Flow

- Server responds: [RTRS_IO_RSP_IMM_W_INV] (includes INV key)

Multipathing & Failover Strategy

Multipath Policies (**mp_policy**)

- **Round-robin:** Cycles through available paths.
- **Min-inflight:** Selects path with fewest active requests.
- **Min-latency:** Uses heartbeat-based `cur_latency` to select fastest path.

Failover Mechanism

1. Periodic Heartbeat Detects Failure
2. Path Marked Disconnected
3. In-flight **I/Os** Failed Over to Healthy Paths
4. Automatic Reconnect Triggered

Security & Performance Trade-offs

Configuration Knob: **always_invalidate**

Option Y (Default)

Performs per-I/O rkey invalidation.

Issues new rkey via `RTRS_MSG_RKEY_RSP`.

Result: **Safer.**

⚠ Cost: ~20% performance cost.

Option N

No per-I/O invalidation.

Result: **Maximum Speed.**

Context: Suitable for trusted environments.

Developer Guide: Client API

Structure Initialization

```
rtrs_ops = (struct rtrs_clt_ops) {
    .priv = sess,
    .link_ev = clt_link_ev,
};
```

Opening a Session

```
rtrs_sess = rtrs_clt_open(&rtrs_ops, sessname,
                    paths, path_cnt, port_nr,
                    0, /* Do not use pdu of rtrs */
                    RECONNECT_DELAY,
                    MAX_RECONNECTS, nr_poll_queues);
```

Closing a Session

```
rtrs_clt_close(struct rtrs_clt_sess *clt);
```

Developer Guide: Server API

Structure Initialization

Inter Regular, Slate Gray

```c
rtrs_ops = (struct rtrs_srv_ops) {
    .rdma_ev = srv_rdma_ev,
    .link_ev = srv_link_ev,
};
```

Opening & Events

Inter Regular, Slate Gray

```c
rtrs_ctx = rtrs_srv_open(&rtrs_ops, port_nr);

srv_link_ev(struct rtrs_srv_sess *rtrs,
            enum rtrs_srv_link_ev ev, void *priv) {
   rtrs_srv_set_path_priv(rtrs, your_sess_ctx);
}
```

Closing

Inter Regular, Slate Gray

```c
rtrs_srv_close(rtrs_ctx);
```

Developer Guide: The I/O Path

Configuration & Permissions

```c
msg_io_conf(void *priv, int errno);

permit = rtrs_clt_get_permit(rtrs_sess,
                    RTRS_IO_CON / RTRS_ADMIN_CON,
                    RTRS_PERMIT_NOWAIT / RTRS_PERMIT_WAIT);
```

Request Execution

```c
req_ops = (struct rtrs_clt_req_ops) {
        .priv = iu,
        .conf_fn = msg_io_conf,
};

err = rtrs_clt_request(WRITE / READ, &req_ops, rtrs_sess,
                       permit, &vec, 1, size, iu->sg, sg_cnt);
```

Status, Performance, & Use Cases

Production Status

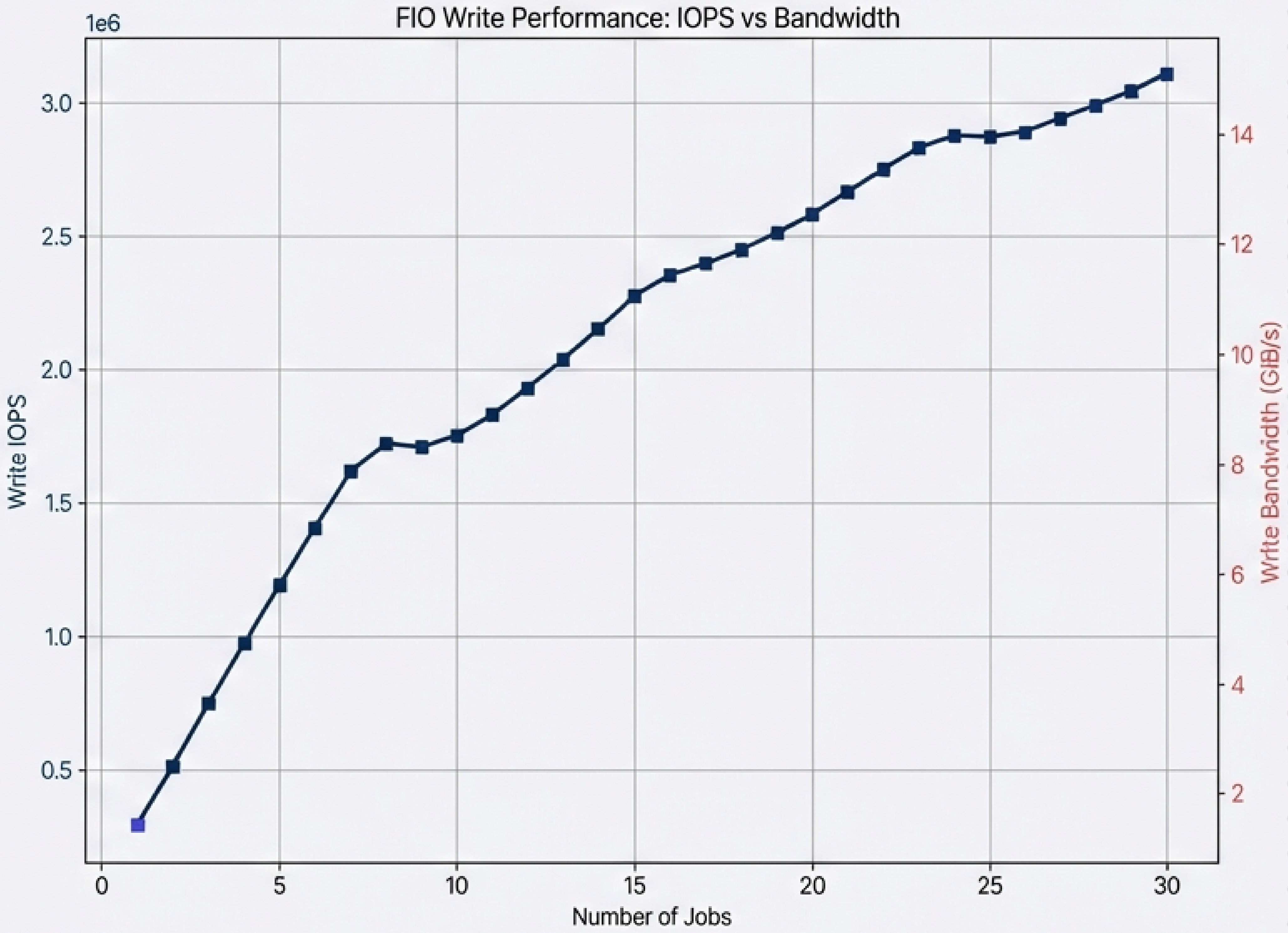
- In-kernel, stable, and in production use.
- Currently running on 5000+ servers in our data centers.

Use Cases

- High-throughput RPC over RDMA.
- ML / AI training: Pre-mapped DMA buffers for streaming large tensors.
- Any kernel component needing fast, reliable RDMA transport.

Getting Started

- Enable RTRS & RNBD, read the docs.
- Study RNBD client/server as reference users.



Let’s discuss RDMA.

We are happy to discuss integration ideas, edge cases, and fabrics (IB, RoCE, iWARP).

Haris Iqbal
haris.iqbal@ionos.com
linkedin.com/in/md-haris-iqbal-00127260/

Jinpu Wang
jinpu.wang@ionos.com
linkedin.com/in/jinpuwang/