The UALink Switch API

A Common Abstraction for Scale-Up AI Switches

Presented on behalf of the UALink Management Working Group

Ayyappa Nuthalapati *[Marvell]*

Justin King *[AMD]*

Ravindranath Kanakarajan *[Marvell]*

OCP PLATINUM® Member

What You'll Take Away

01

Overview

One SMA, driven by a vendor-agnostic API, works with any vendor's switch.

02

API Model

Objects, attributes, key-based CRUD, bulk operations, telemetry and event notifications.

03

Bring-Up Workflow

Pod Controller-driven: discovery → port bring-up → neighbor validation → virtual-pod isolation → route install.

04

Early Collaboration

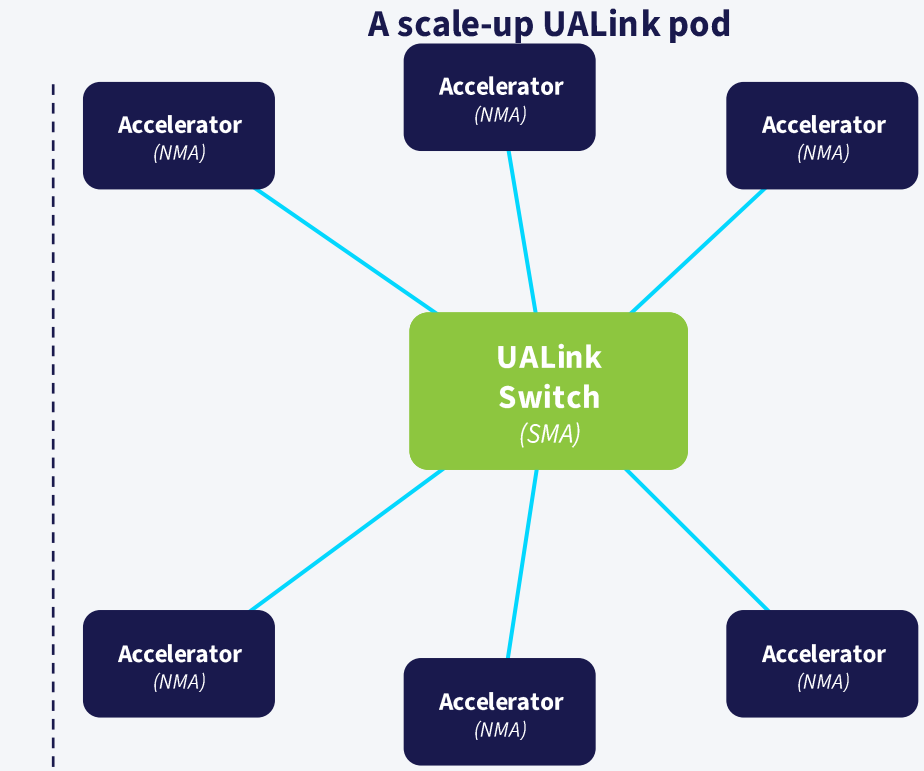
Where UALink's Switch API stands today, and how to help shape it while implementations are early.

*Focus of this session: the software↔Physical Switch interface — the **UALink Switch API** that abstracts UALink switch control & management*

Anatomy of a Scale-Up UALink Pod

OVERVIEW

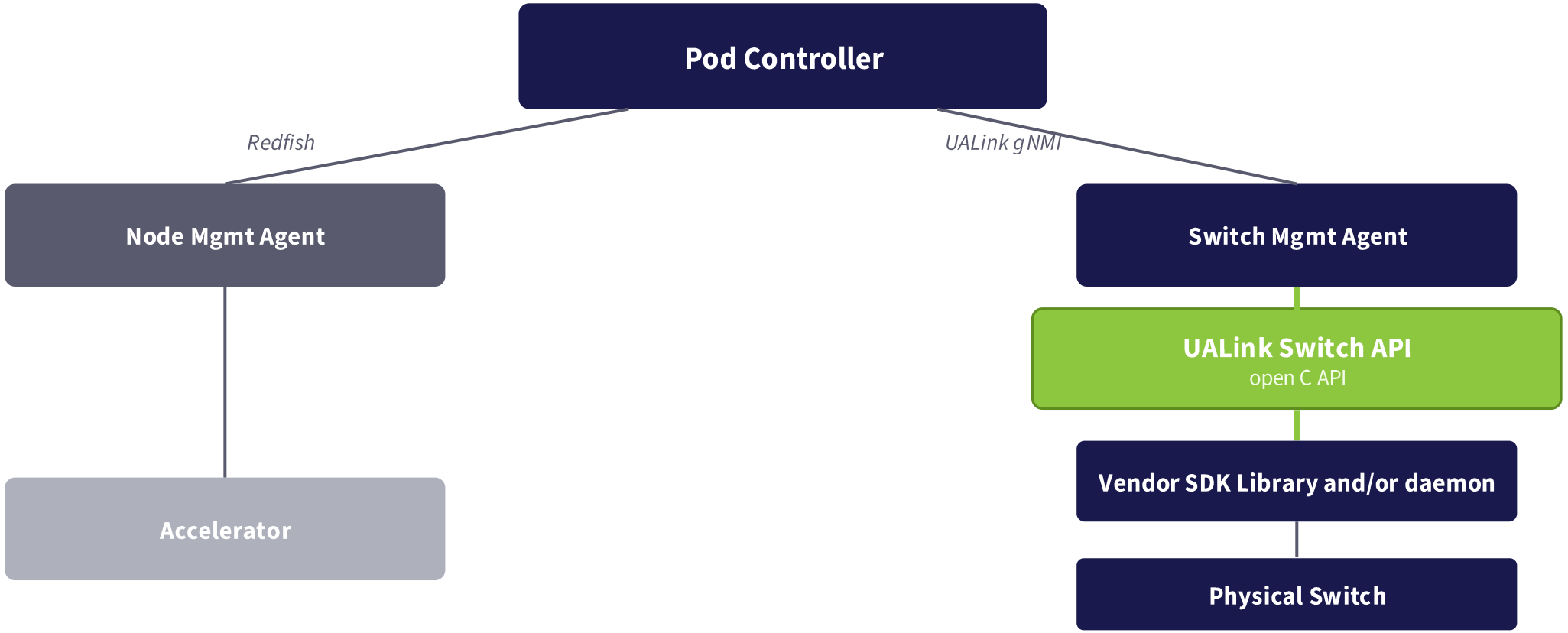
- **Scale-up, memory-semantic fabric.** Directly connects Accelerators for load, store, and atomic memory operations, with low latency and high bandwidth purpose-built for tightly-coupled AI compute.
- **Open, standards-based, and layered.** UALink 200G reuses the IEEE 802.3dj Ethernet PHY and existing cables/retimers; the Consortium defines Data Link, Transport, and Protocol layers above it.
- **Managed at scale, by three actors.** As Pods grow to hundreds of Accelerators across multi-vendor Switches, a Pod Controller, NMA (Node Management Agent), and SMA (Switch Management Agent) divide the work — this talk follows the SMA.



Where the UALink Switch API Fits

OVERVIEW

The manageability architecture defines several interfaces. This talk is about one of them.



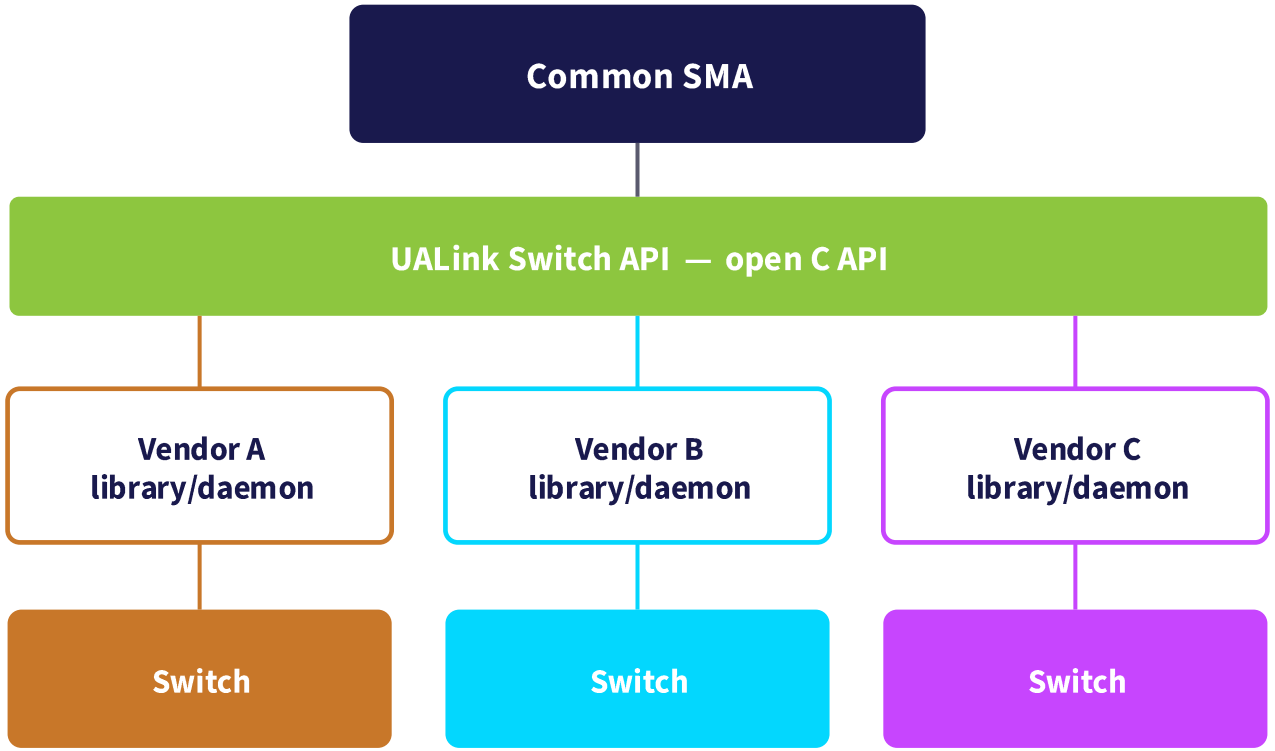
- *Grey = broader architecture, out of scope today*

UALink Switch API sits between the SMA and the Switch silicon — **the one interface this talk covers.**

One SMA, Any Switch

OVERVIEW

- **Vendor-agnostic SMA.** The UALink Switch API ensures that vendor-specific library implementations exhibit the same behavior.
- **Standard API contract.** UALink Switch API defines a standardized C API—not the physical interface or transport to the switch.
- **Vendor-owned library/daemon.** Each vendor provides a library/daemon behind the API to handle switch communications and program its silicon.
- **Common abstraction from day one.** UALink Switch API establishes a shared management boundary while the switch ecosystem is still emerging.



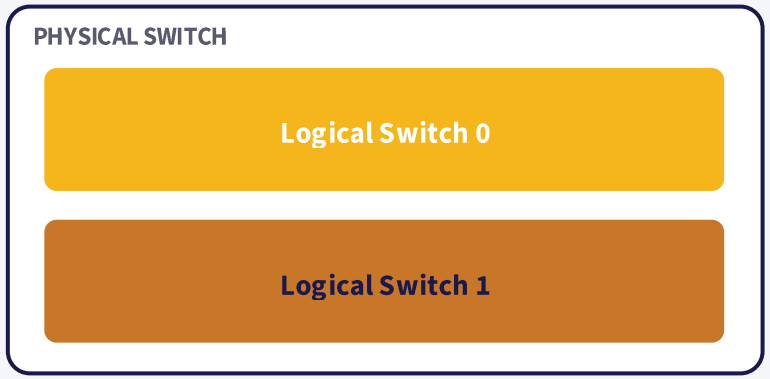
Same SMA and workflow. Different silicon underneath.

Logical Entities: Logical Switches & vPods

OVERVIEW

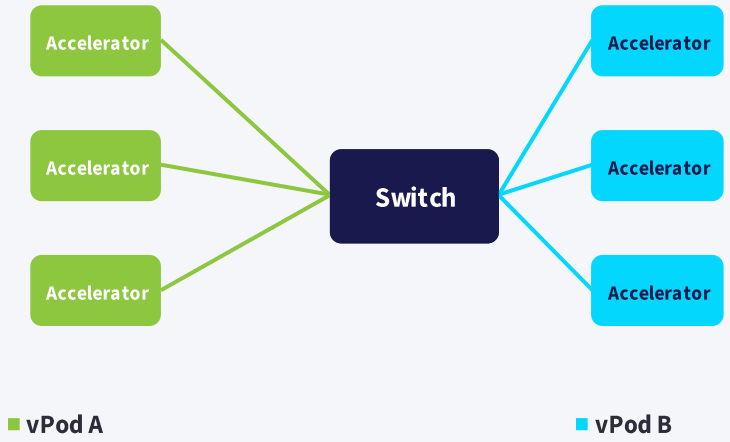
*A switch partitions hierarchically — into Logical Switches for **bandwidth and resiliency,** and within each, into vPods for **tenant isolation**.*

One physical switch → multiple switching planes



Ports join a plane via LOGICAL_SWITCH_ID; a switch advertises LOGICAL_SWITCH_COUNT.

One physical pod → two isolated vPods



- **Switching planes.** A physical switch implements one or more Logical Switches — independent planes for bandwidth and resiliency.
- **Tenant isolation.** vPods isolate tenants or workloads — e.g., separate inference models — side by side, isolated by routing entries in every physical switch.
- **Port assignment.** Ports join a plane via LOGICAL_SWITCH_ID and a vPod by VPOD_ID

Key Entities the SMA Manages

API MODEL

The vocabulary first — in a UALink fabric the SMA manages a small set of "objects".*

MODELED AS API OBJECTS

Properties: Identity, Config, Oper State and Telemetry

Platform

API implementation instance; discovers and owns the Switches.

Switch

A UALink physical switch — identity & state.

Port

A 128G/200G port; per-UALink protocol layer control and monitoring.

Neighbor

The peer observed on a port; validated vs intent.

Route

A forwarding and vPod Isolation entry: key → egress port.

EXPRESSED VIA KEYS & ATTRIBUTES

Logical Switch

A switching plane within a physical switch.

`via LOGICAL_SWITCH_ID`

Virtual Pod (vPod)

An isolated tenant partition of the fabric.

`via{LOGICAL_SWITCH_ID, VPOD_ID, Dest APU_ID} as compound key.`

Accelerator

The compute endpoint the fabric connects.

`via apu_id in keys`

→ Next: how each object exposes its **attributes and CRUD operations.**

** Definitions are still evolving as the spec moves toward v1.0.*

An Object-Based API Model

API MODEL

The entities from the previous slide become six API modules — each addressed by a typed object key, each with a consistent set of operations.

Module	Addressed by (object key)	Operations
Platform	`singleton — no key`	**Get · Set**
Switch	`switch_index*`	**Get · Set**
Port	`switch_index, port_number`	**Get · Set · Stats**
Neighbor	`switch_index, port_number`	**Create · Delete · Get · Set · Stats**
Route — Type-1 vPod	`switch_index, vpod_id, logical_switch_id, dst_apu_id`	**Create · Delete · Get · Set**
Route — Type-2 Flow	`switch_index, src_apu_id, src_port_number, dst_apu_id`	**Create · Delete · Get · Set**

Every object type can also raise **event notifications** through a single registered callback — more on that shortly.

** switch_index is immutable — assigned by the API implementation, unchanged across restarts or upgrades.*

Attribute – Properties of an Object

API MODEL

- **Attribute Roles**
  - Every attribute plays Identity, Config, Oper, or Telemetry role.
- **Attribute = id + value**
  - Each attribute is an ID paired with a typed union value (int, lists …)
- **Flags define behavior**
  - Flags tell the SMA how each attribute may be used
  - Eg: SET, READ_ONLY, CREATE_ONLY, or CREATE_AND_SET
- **Self-documenting**
  - Annotations live with the definition – the header is the spec
  - Eg: @type, @default, @validonly, @min/@max
- **Extensible**
  - New attributes append with out altering any function signatures.

```
/**
 * @brief Admin state of the switch
 *
 * @type    ualink_switch_admin_state_t
 * @flags   SET
 * @default UALINK_SWITCH_ADMIN_STATE_DISABLED
 */
UALINK_SWITCH_ATTR_ADMIN_STATE = 17,

/**  @brief   DL down-delay after RS fault (ms)
 *   @type    uint8_t
 *   @flags   SET
 *   @default 5
 *   @min 0
 *   @max 10
 */
UALINK_PORT_ATTR_DL_DOWN_DELAY = 3,
```

Key-Based CRUD, Bulk by Design

API MODEL

Key-based addressing
- Objects are named by typed key structs
- Eg key structs : switch_index, port_number…

Bulk-first operations
Every call operates in bulk — an object list or key array in, per-object status out.

```
/* create/set: one inout object-list in, one reflection list out */
ualink_status_t create_neighbor(
    ualink_neighbor_object_list_t *create_neighbors,
      /* inout: key+attr in, status out */
    ualink_neighbor_object_list_t *reflect_neighbors);
      /* out: post-op state, count=0 if unsupported */

/* remove/get: still a flat key array in */
ualink_status_t remove_neighbor(
    ualink_uint32_t                neighbor_count,
    const ualink_neighbor_obj_key_t *neighbor_obj_key_arr_p,
    ualink_status_t                *status_arr_p);

/* Switch, Port & Route share this shape too: */
/* create/set = one inout list; remove/get = flat key array. */
```

Robust partial failure

• Atomic per object. • Per-object status. • Attribute-indexed errors. • Accurate state correlation.

• **Config-reflection output.** A second reflect_<obj_type> list returns the object's real post-op state — distinct from the notification-based reflection covered next.

Telemetry

API MODEL

- **Bulk counter reads**
  - *get_port_stats* fetches selected counters for many ports in one call.
- **Stats mode**
  - Operations to be performed on the counter.
  - Eg: READ, CLEAR, or READ_AND_CLEAR(atomic)
- **Support for Histograms**
  - Histograms expose distributions, not just totals.
  - Eg: FEC-error histogram, credit-occupancy histogram

```
ualink_status_t get_port_stats(
  ualink_uint32_t              port_count,
  const ualink_port_obj_key_t *port_obj_key_arr_p,
  ualink_uint32_t              number_of_counters,
  const ualink_port_stat_id_t *counter_ids_p,
  ualink_stats_mode_t          stats_mode,  /* R|C|R&C */
  ualink_uint64_t             *counters_p);
```

- TX / RX bytes
- PL FEC-error histogram
- PL 64b/66b decode errs
- DL bad/good CRC
- DL LLR replays
- TL messages tx/rx
- TL / UPLI credit histograms
- PL alignment & RS faults

Event Notifications

API MODEL

- **One callback.** The SMA registers a single callback — with an opaque user-data pointer — and selects which modules to hear from via an API bitmap.
- **Config reflection.** Each event carries the object's key plus its full current attribute state — not just the one field that changed.
- **Timestamped.** Every notification is stamped, so the Pod Controller can order and reconcile state changes.

```
ualink_register_callback(
  registration_args,  /* .api_bitmap */
  user_data_p,        /* opaque ctx */
  callback_fn_p);

/* callback — user_data_p passed back: */
cb(notification_count,
   notification_list_p,
   user_data_p);

/* each notification: */
{ timestamp, object_key, attr_list }
```

Why full state, not a delta?

- No lost-update races — the snapshot is authoritative.
- The SMA needs no event-replay buffer to stay correct.
- Timestamp ensures latest state is consumed even if received out-of-order

Bring-Up Workflow, End to End

BRING-UP WORKFLOW

Initiated and controlled by the Pod Controller — the SMA carries out each step as a stateless passthrough through the UALink Switch API.

1 DISCOVER

Platform init & switch discovery

SMA inits API; Pod Controller → SMA discovers presence and operational state of all objects.

`get_platform`

2 CONFIGURE

Switch bring-up

Set device ID; admin-enable.

`get / set_switch`

3 BRING UP PORTS

Port configuration

Set speed/mode & admin state; watch PL·DL·TL·UPLI.

`get / set_port`

4 VALIDATE

Neighbor validation

Detect(Through Notification) each link peer; confirm it matches intent.

`create / get_neighbor`

5 ISOLATE

vPod isolation

Assign switch ports to vPods for tenant isolation

`set_port (VPOD_ID)`

6 ROUTE

Route installation

Tenant isolation via logical plane+vPod scoped routes programming

`create_route_type_1/2`

End-user workload begins

Call to Action

COLLABORATE

The UALink Switch API is being defined at the **inception** of the switch ecosystem — now is the moment to shape a common abstraction.

1

Shaping for release

The annotated API header set is being finalized under the UALink Consortium (Apache-2.0) — releasing to the community Github soon.

2

Providing a reference to build on

A mock SMA and a mock vendor SDK exercise the API end to end — a concrete starting point for implementers.

3

Join the Consortium

Membership opens the door to help refine attributes, telemetry, and workflows through the Management WG — while vendor implementations are still early.

Join the UALink Consortium — Consortium members can then help standardize switch management through the Management Working Group.

References: https://ualinkconsortium.org/

Working Group Leadership & Contributors

COLLABORATE

*The UALink Switch API is developed in the open within the **UALink Management Working Group.***

MANAGEABILITY WORKING GROUP — CO-CHAIRS

JK

Justin King

AMD

Co-Chair · UALink Management WG

AS

Arun Satyanarayana

Google

Co-Chair · UALink Management WG

Want to contribute? Reach out to the co-chairs.
Help extend the UALink Switch API to cover industry-wide use cases and strengthen open, multi-vendor switch manageability.

Thank You!

The UALink Switch API

A Common Abstraction for Scale-Up AI Switches

Presented on behalf of the UALink Management Working Group

Explore the open headers & reference SMA — and join the conversation.



OCP GLOBAL SUMMIT

October 12-15, 2026
San Jose, California