Sonic Architecture & Interactions

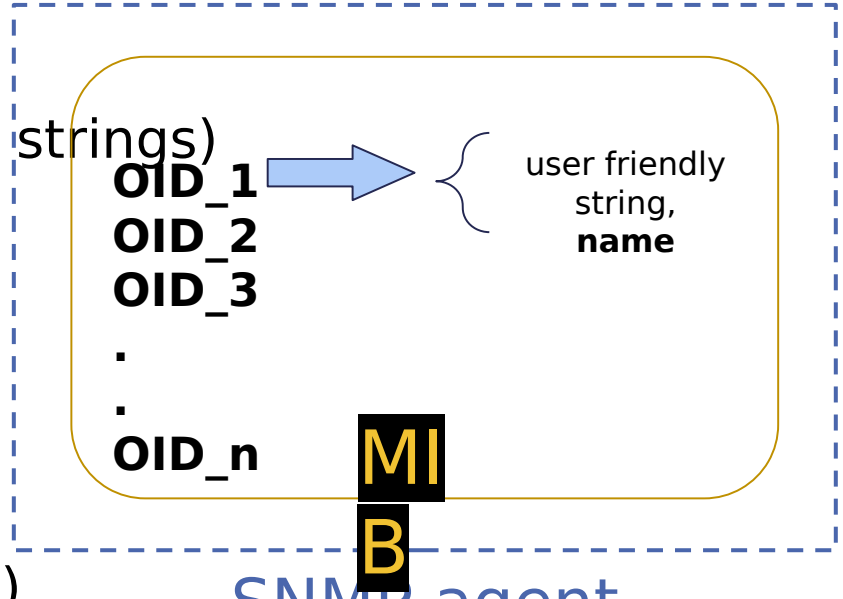
How data flows and containers interact

Table of Contents (Index)

SNMP: Simple Network Management Protocol

MIB: Management Information Base (user friendly strings)

OID: Object Identifier (numerical)



Information flows:-

1) Manager can query (polling)
2) Agents can send Traps (something like interrupt)
3) SNMP versions (1, 2c and v3: has authentication and encryption is supported)

Used for: Monitoring and modifying network configurations.

1	SNMP **server** -> SNMP **Manager**
2	**switch/ router/ UPS/ Host** -> SNMP **Agent**
3	**SNMP agent ->** contains **MIB**.
4	**MIB -> contains OIDs** with their **variable,** of which values that can be get and set

BGP: Border Gateway Protocol

1.	**IGP**: Interior Gateway Protocol	I. RIP (Routing Information Protocol) II. OSPF (Open Shortest Path First) III. EIGRP (Enhanced Interior Gateway Routing Protocol) IV. IS-IS (Intermediate System Intermediate System)
2.	**EGP**: Exterior Gateway Protocol	works as network routing info and reachability
4.	**FIB**: Forwarding Information Base	Holds information about the next hop. Closely associated with Route Tables
5.	**AD:** Administrative Distance vs **Metric**	The cost of route inclusive of protocol, hops and various other factors. (different protocols in consideration). Lower the administrative distance, higher priority it has. Whereas, metric is similar cost but when protocol is same. Higher the metric the higher priority a route has.
6.	**AS:** Autonomous Systems	A fully configured internal network that requires a gateway to access the external network. AS number used for management of the entire group as a whole. Ease of configuration by Network Managers.

7.	**OSPF**: Open Shortest Path First	Cost = Ref Bandwidth / Link Speed
8.	**RIP**: Routing Information Protocol	Broadcast, cost = shortest hops
9.	**BGP**: Border gateway Protocol	Usually an exterior protocol. But can be used internally in explicit cases Several metrics for cost, called **PA**: Path Attributes **Path Vector**: bgp gives us the entire path(direction), data will move on.

BGP:

- is used to connect two different autonomous systems.
- Is used as gateway to an external network (Internet)
- needs both, end routers to be configured

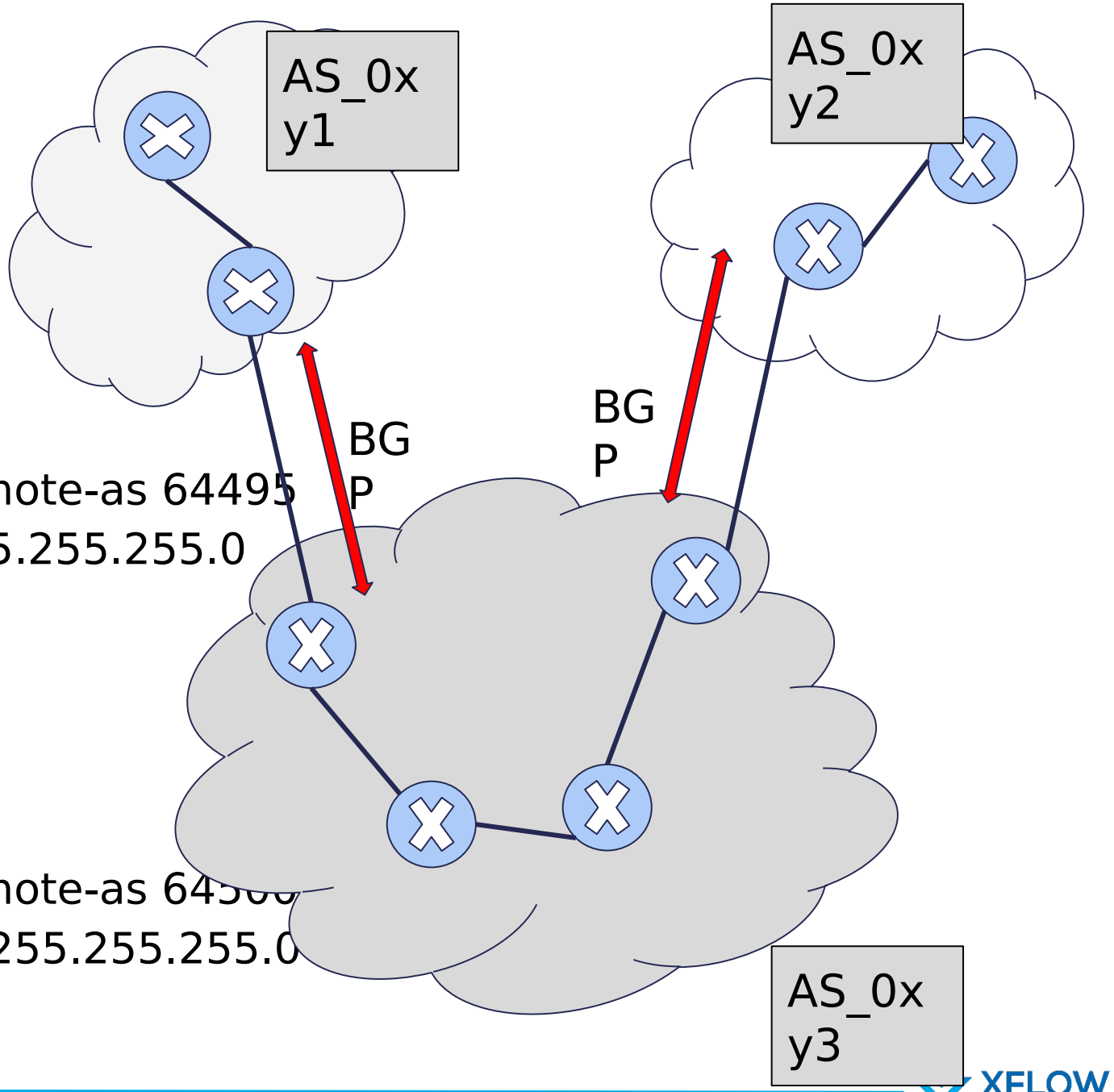
BGP configuration (static config)

R1:

1. conf term
2. router bgp 64500
3. neighbour 198.51.100.2 remote-as 64495
4. network 192.0.2.0 mask 255.255.255.0
5. end

R2:

6. conf term
7. router bgp 64495
8. neighbour 198.51.100.1 remote-as 64500
9. network 203.0.113.0 mask 255.255.255.0
10. end

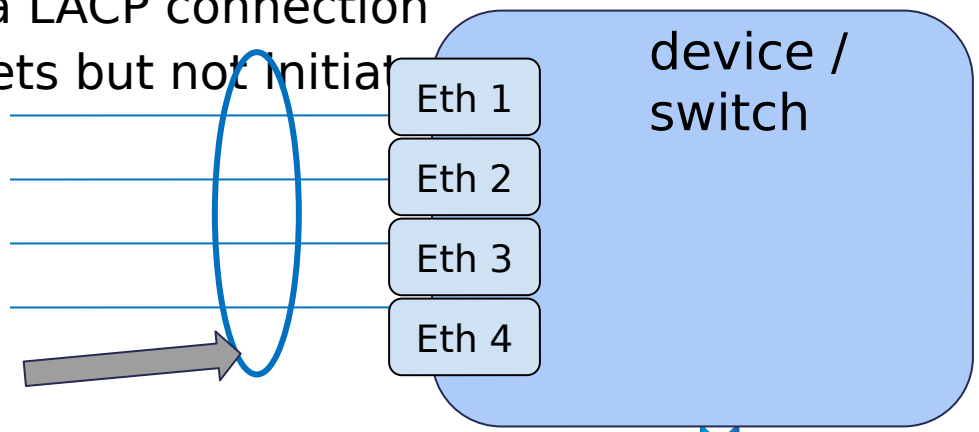


LAG: Link Aggregation Group

- IEEE standard (802.3ad) set in 2000
- combine multiple physical ethernet links to form one logical link **(LAG)**
- max 16 ethernet ports of same kind
- **upto 8 eth ports** in LAG are **active** rest are standby
- **Divides** traffic b/w links **deterministically**, using **hashing algorithms**
- Works with **multiple vendors**
- This logical link will show as a **single path** from **spanning tree protocol** (Layer 2)

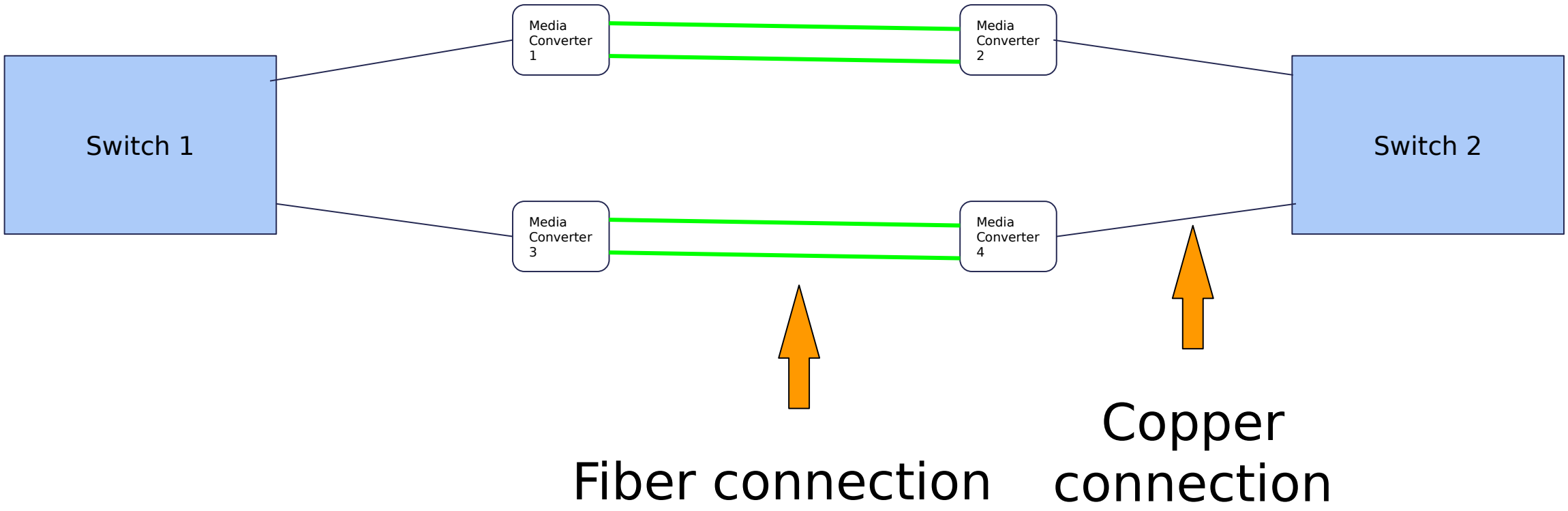
★ **LACP**: Link aggregation control protocol (**part of LAG**)
  - **Active**: keeps sending packets to keep alive a LACP connection
  - **Passive**: Interface can respond to LACP packets but not initiat

★ **Modes of connectivity**
  - **dynamic LAGs**: links **with** LACP (loadbalances evenly on all ports)
  - **static LAGs**: links **without** LACP

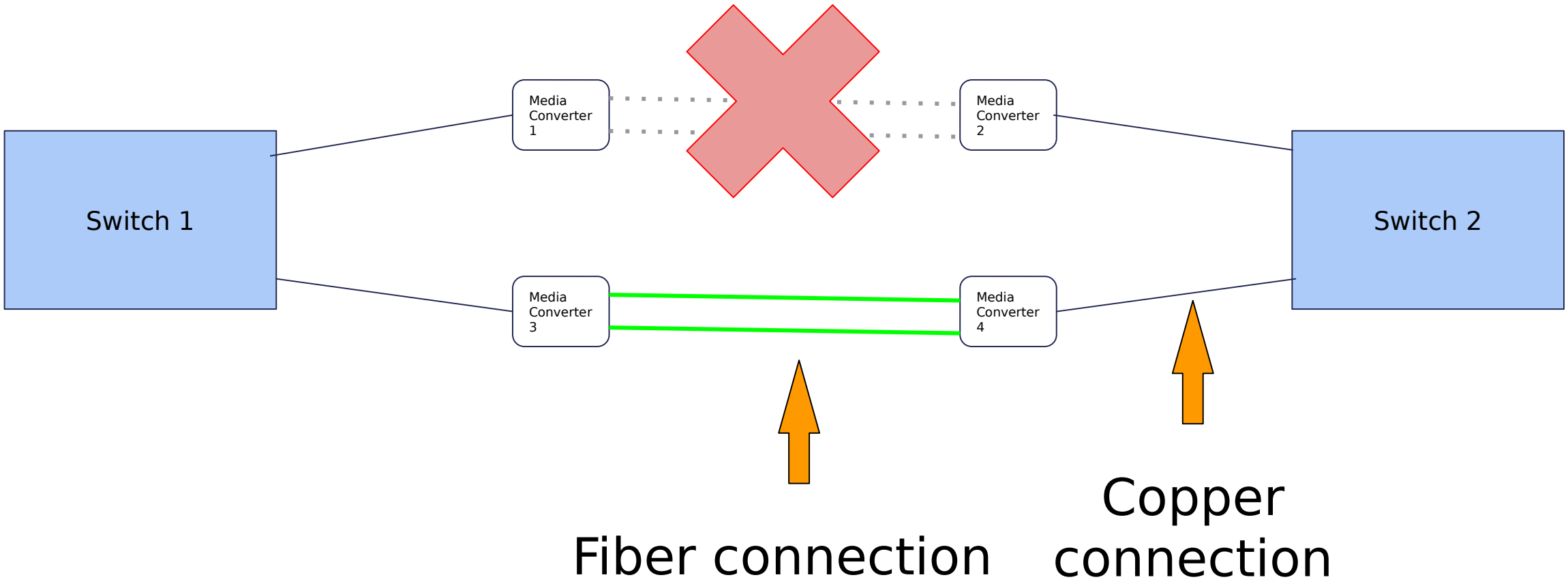


Single LAG

Dynamic (LACP) vs Static

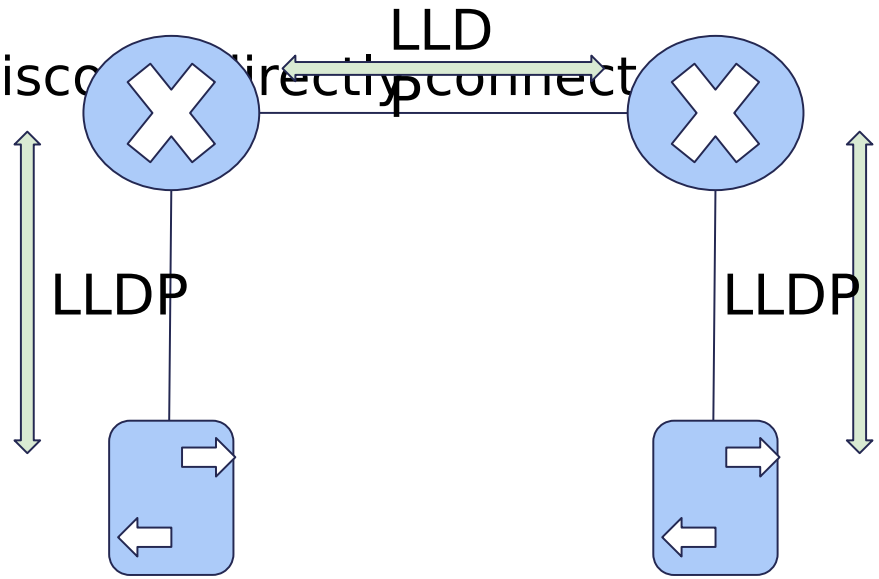


Dynamic (LACP) vs Static



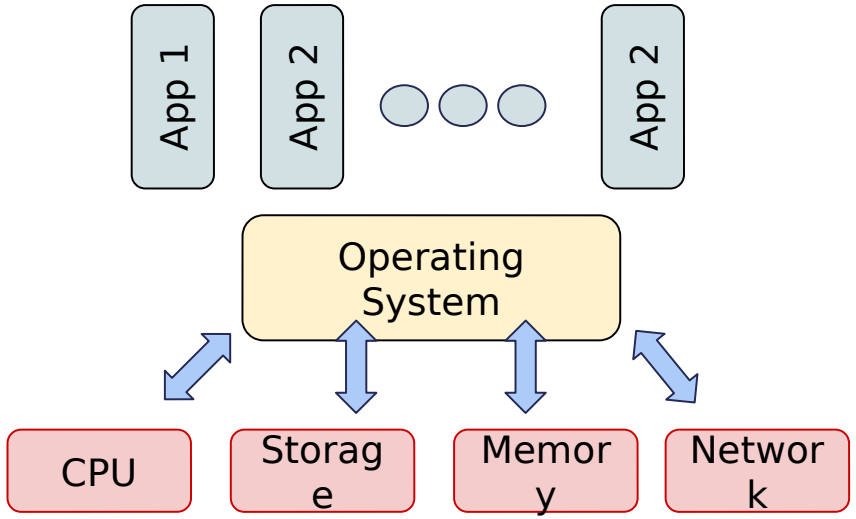
LLDP: Link Layer Discovery Protocol

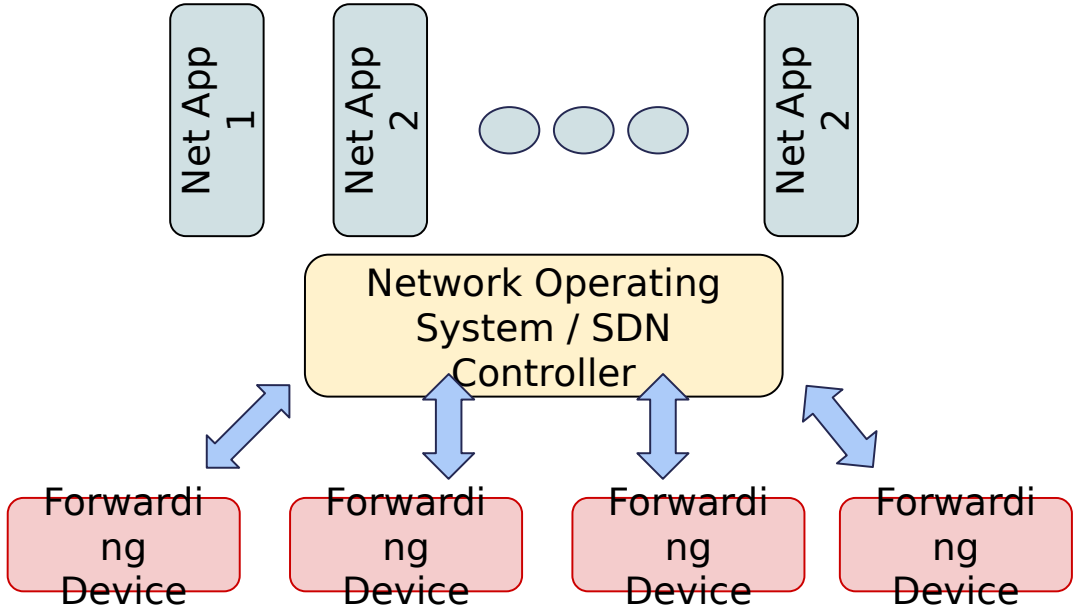
- Layer 2 protocol (uses specific frames for the purpose)
- SImilar to proprietary CDP (Cisco Discovery Protocol)
- **gathers info** about **neighboring devices** and shares about self to other devices
- **Network devices** store such info in **MIB**
- **MIBs** help the Network Management System to build topology through **SNMP query**.
- **TLV (Type, Length** and **Value)**, attributes to disco directly connect neighbors
  - Mandatory:
    - Chassis ID
    - Port ID
    - TTL, Time to live
    - End ofLLDPDU TLV
  - Optional:
    - port desc
    - Sys Description
    - Sys Name
    - Sys Capabilities
    - etc.



SDN: Software Defined Networking

Operating System Model

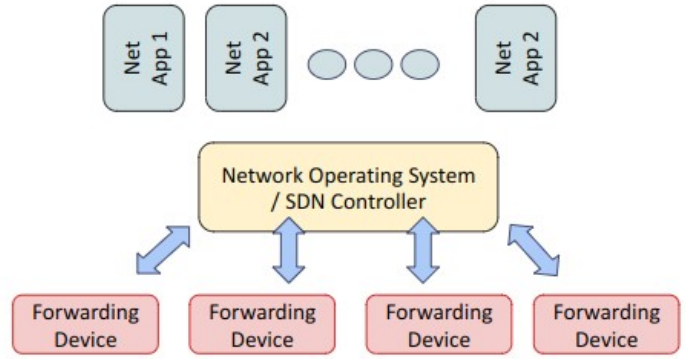




SDN Model

- Open network and programmable
- Existing and future functionalities to be part of the network at great speeds.’

- Application interface -> **Northbound** APIs
- NOS -> **Control Plane** (SDN controller)
- OpenFLow, OVSDB, NETCONF, SNMP -> **So**
- Forwarding Devices ->



Forwarding Device tasks

1. dropping packet
2. Modifying packet headers
3. Sending packet to the port/s intended

1	Packet arrives at forwarding device -> **query** (**OpenFlow**) controller / already know
2	SDN **controller applications** determine what to do. And **dictate (OpenFlow)** the **forwarding** devices.
3	Cache, instructions for future packets. **Fast Path** until flow entries on forwarding devices **expire** when **timers** are up

Virtualization: Hypervisors and Containerization

- History
  - Each application on a new server
- Utilizing server’s full potential
  - Virtual Machines were developed
  - simulating hardware and software
- Hypervisors, allow one machine to run several VMs
  - **Type-I**: bare metal hypervisors, are usually run directly on hardware without any hosting OS. e.g VMware ESX-i, Microsoft Hyper-V, Citrix XenServer, KVM
  - **Type-I**I: are hosted over an underlying OS. e.g Virtualbox, VMware workstation
- Virtualization types
  - virtual machines
  - cloud
  - **containerization** (application packaged with all configs, dependencies etc)
    - solves the distribution and out of the box configuration problem, simply plug and play kind of tool.
    - **docker** is a container engine used often.
    - Containers are **light** and **fast** as they dont package the entire OS

SONiC: Software for Open Networking in the Cloud

Important Factors that make SONiC stand out

- Modularity
  - Isolated containers on their functionality, so that in an event of upgrade there is no downtime and operations are un-interrupted.
- Decoupling
  - The SAI (Switch Abstraction Interface), ensures that software runs independent of the underlying hardware. Hence, vendor agnostic infrastructure is ensured.
- Adaptability and scalability
  - The SONiC is designed with scalability and flexibility in mind. Can be used as a core device at an ISP and is flexible to be deployed in data centers.

Introduction

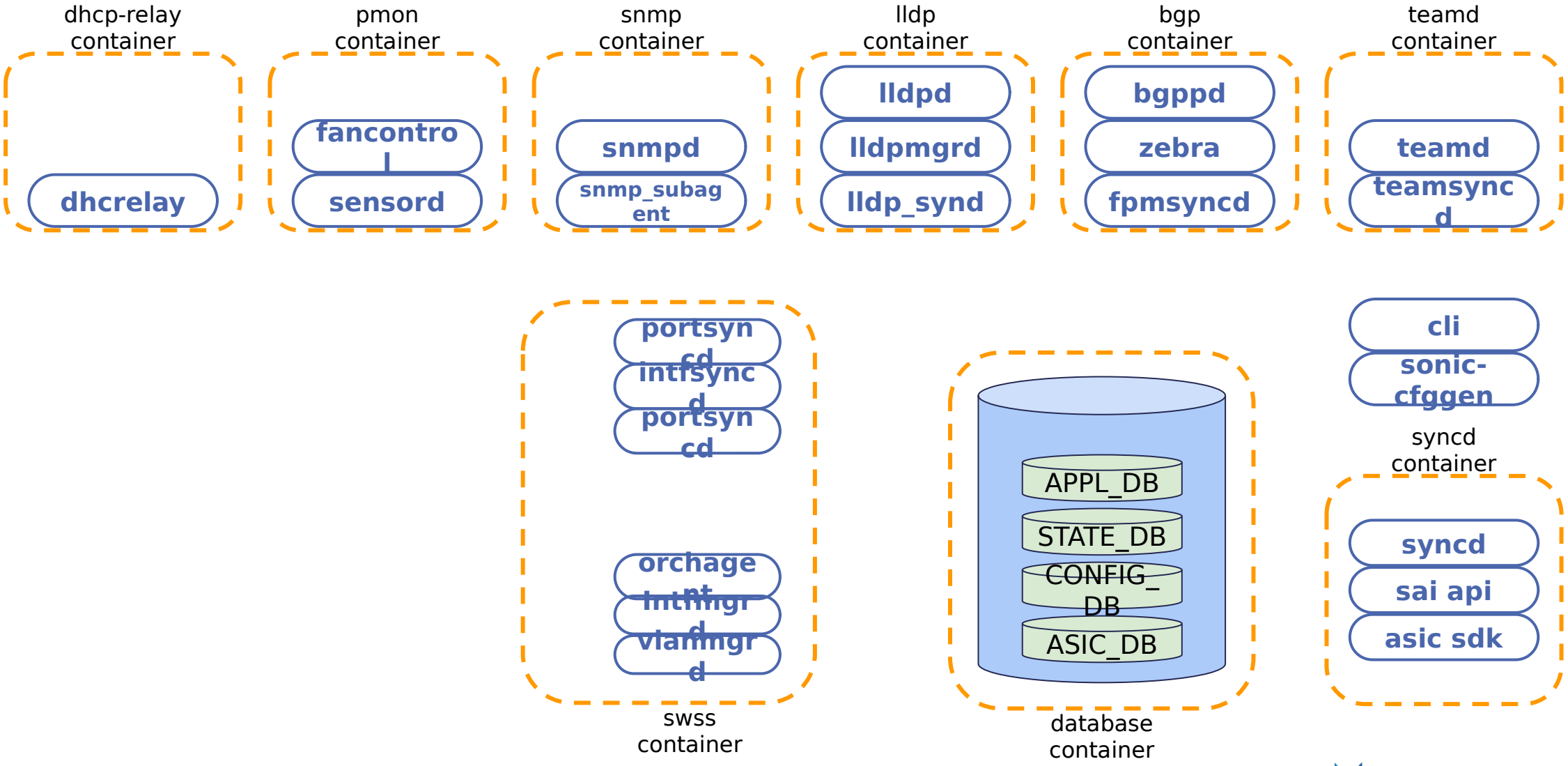
The different containers at application layer (Management Plane) are as follows;

1. Dhcp relay (If no DHCP server is there in current VLAN then it gets relayed to other VLAN)
2. P mon (hardware and sensor related logging agents)
3. Snmp (network configuration and management related, routers and switches as agent)
4. Lldp (Link Layer Discovery Protocol-Layer 2, MAC addresses related)
5. Bgp (Hosts Free Range Routing, FRR agents to manage Layer 3 routing)
6. Teamd (Link Aggregation, LAG management and south bound subsystem interactions)

Rest of the containers (Control Plane) are as follows;

7. Swss (Switch state service, implements the Switch Abstraction Interface, SAI APIs)
8. Db (Hosts and manages the Redis-DBs for the overall architecture)
9. Sync_d (Provided by the vendor and has the ASIC SDK to implement vendor

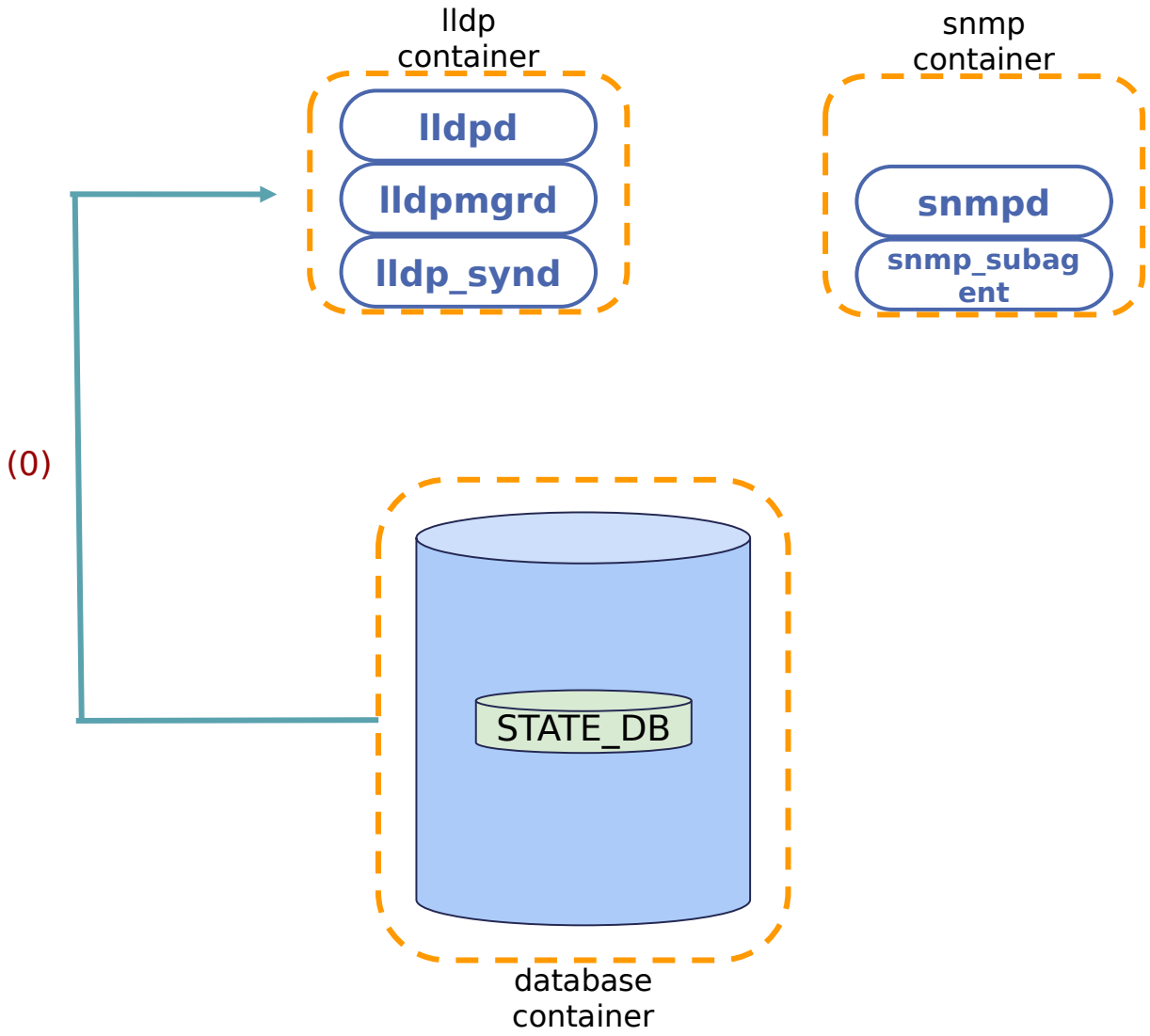
Architecture Diagram



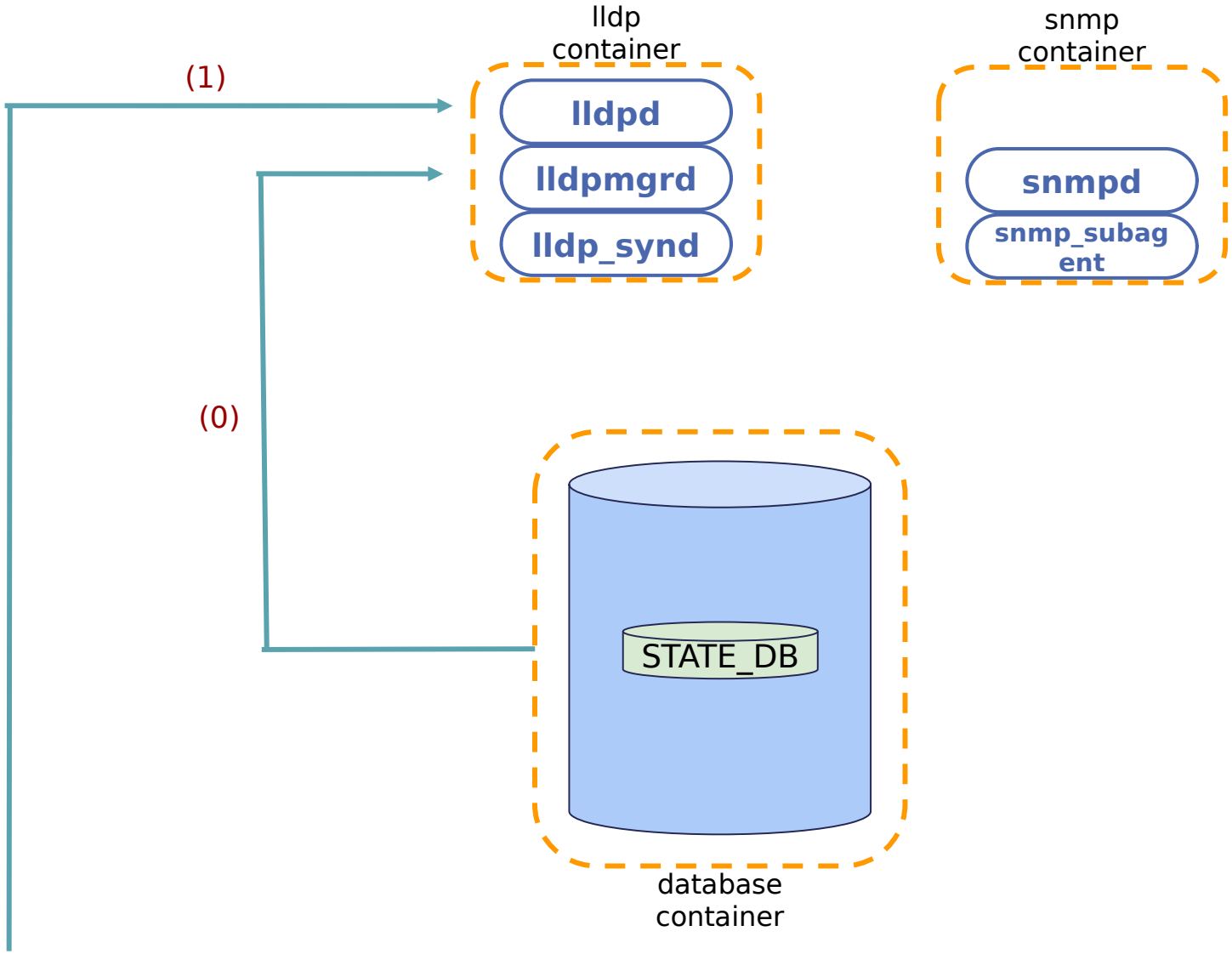
XFLOW RESEARCH

State Interactions: Lldp

1. **Initialization**: subscribe to **state_db** to get live state of **physical ports**
   - **lldpmgr** is subscribed
   - has a **poll** of every **5 seconds**

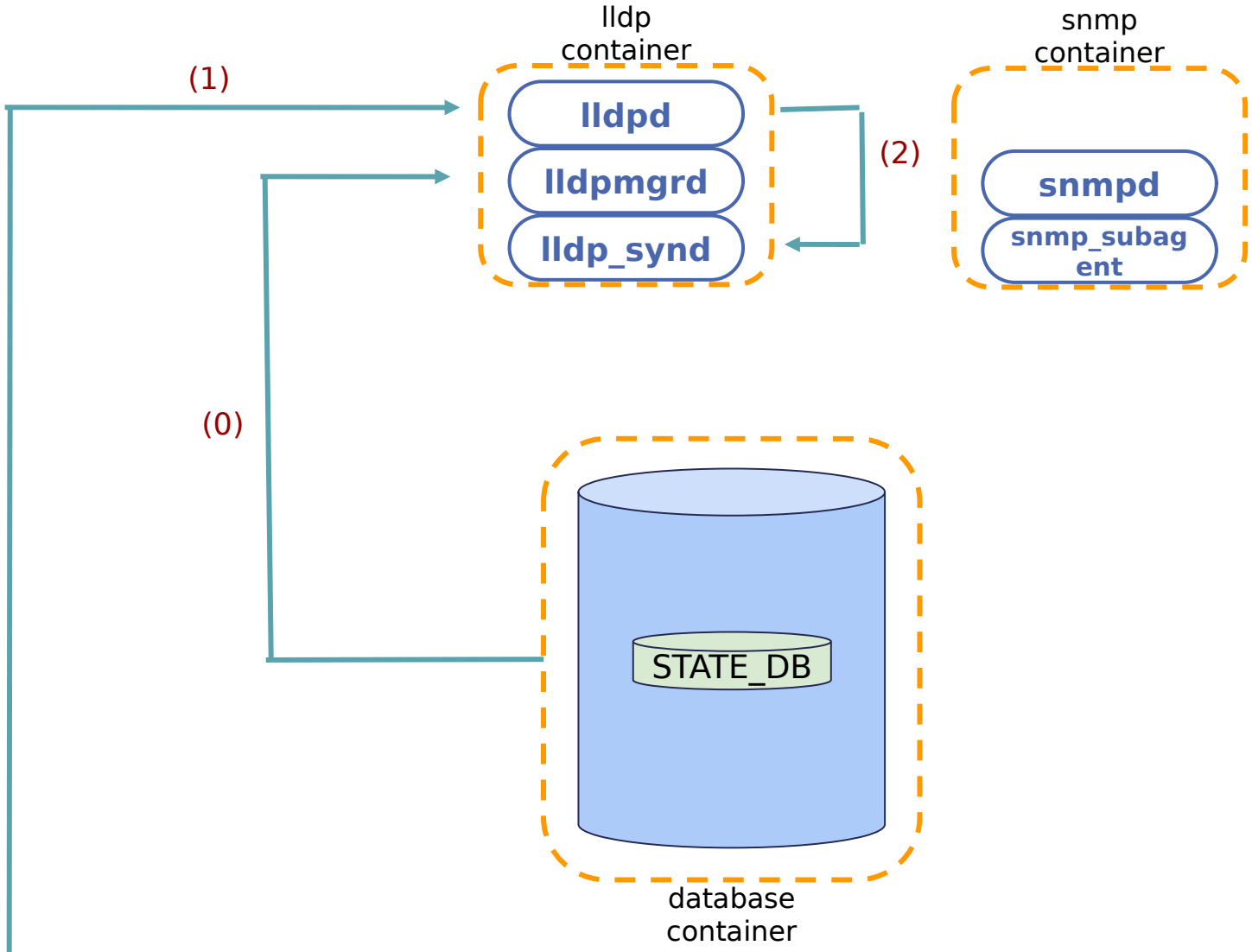


1. **Initialization**: subscribe to **state_db** to get live state of **physical ports**
2. **new LLDP** packet at **kernel** lldp socket. Gets received at **lldpd** (lldp process) and is **parsed**
   - packet arrives at the **lldp socket** in **kernel**

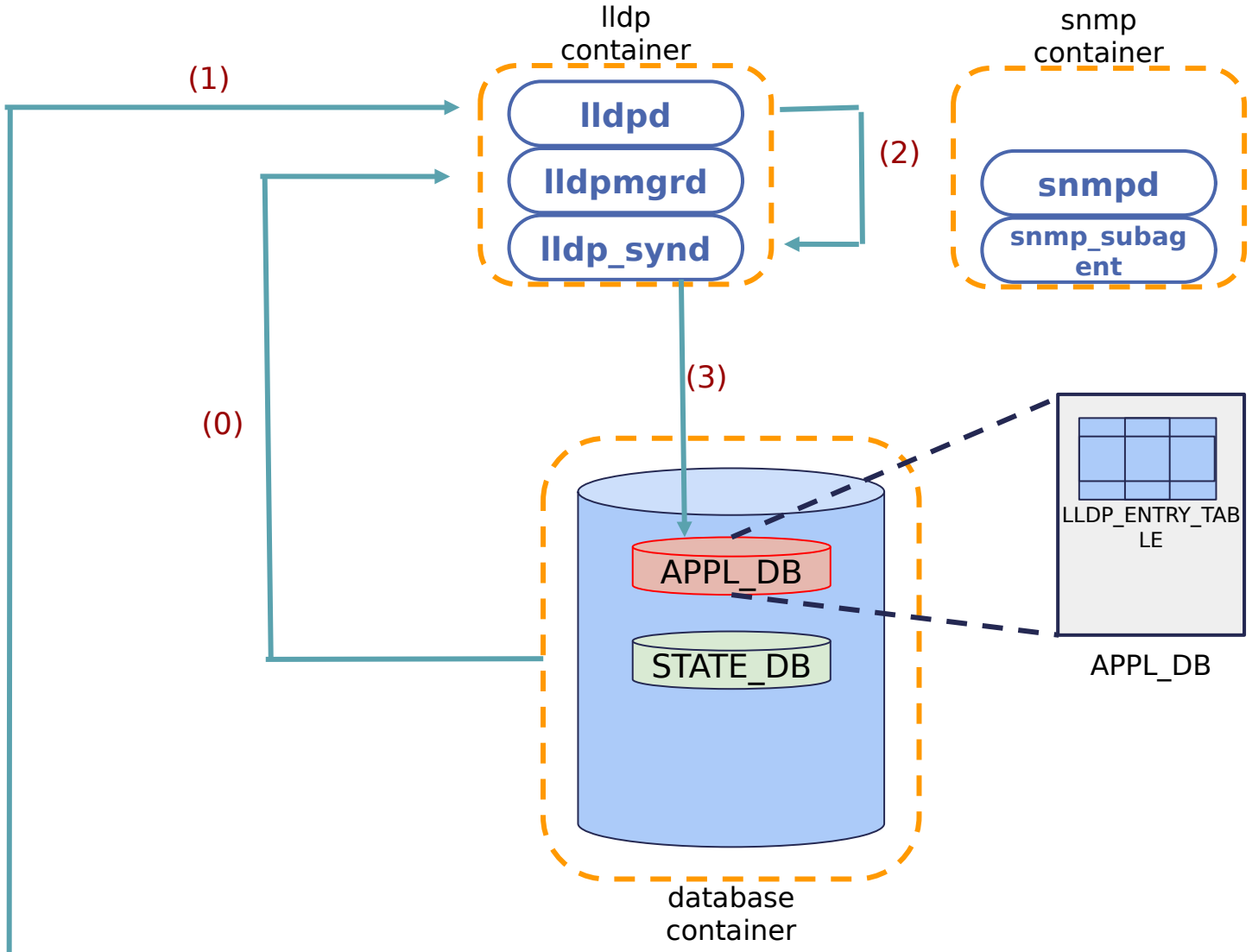


XFLOW RESEARCH

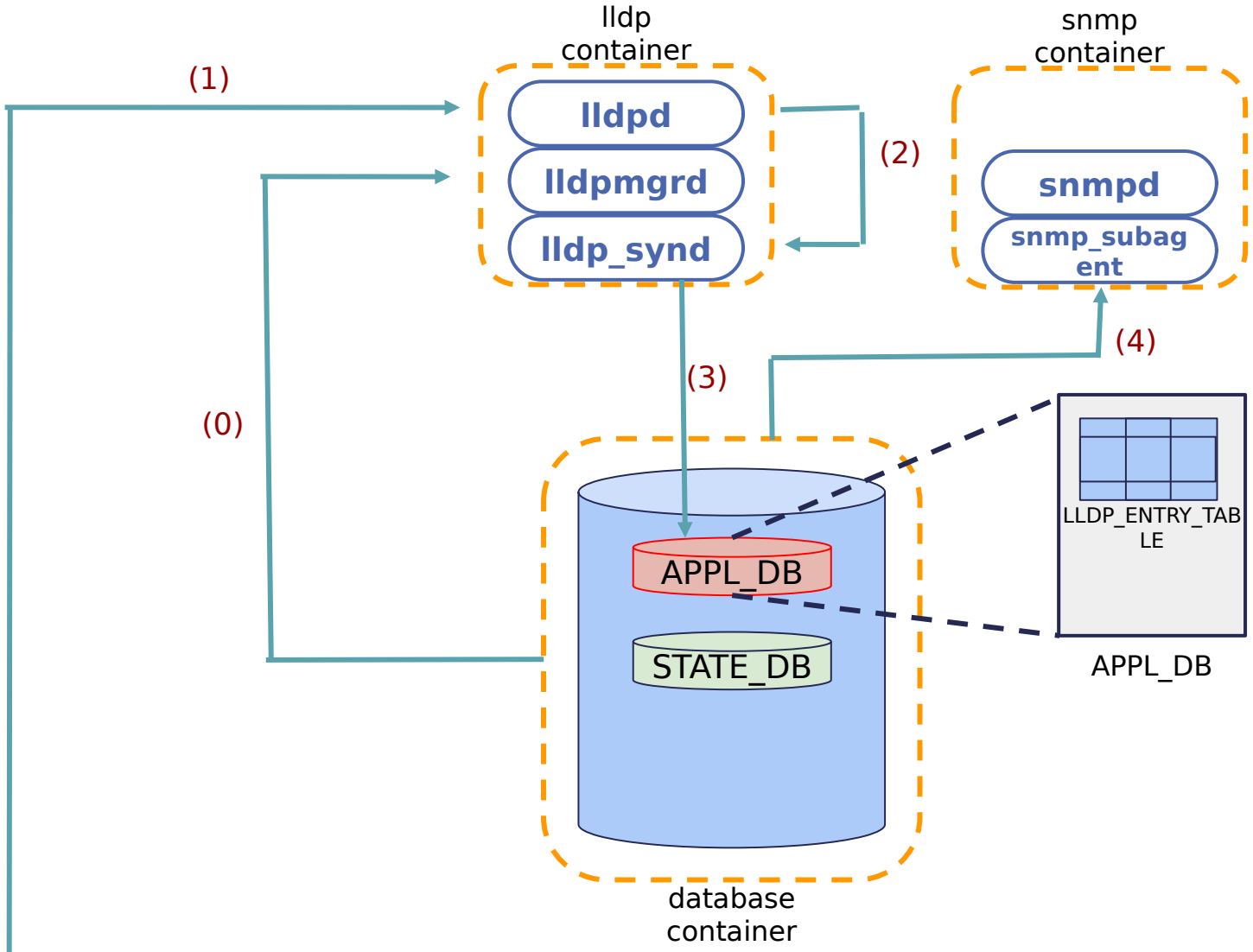
1. **Initialization**: subscribe to **state_db** to get live state of **physical ports**
2. **new LLDP** packet at **kernel** lldp socket. Gets received at **lldpd** (lldp process) and is **parsed**
3. **Lldp syncd**: receives information **digest** from **lldpd**



1. **Initialization**: subscribe to **state_db** to get live state of **physical ports**
2. **new LLDP** packet at **kernel** lldp socket. Gets received at **lldpd** (lldp process) and is **parsed**
3. **Lldp syncd**: receives information **digest** from **lldpd**
4. **push state**: to **APPL_DB**, to **LLDP_ENTRY_TABLE**

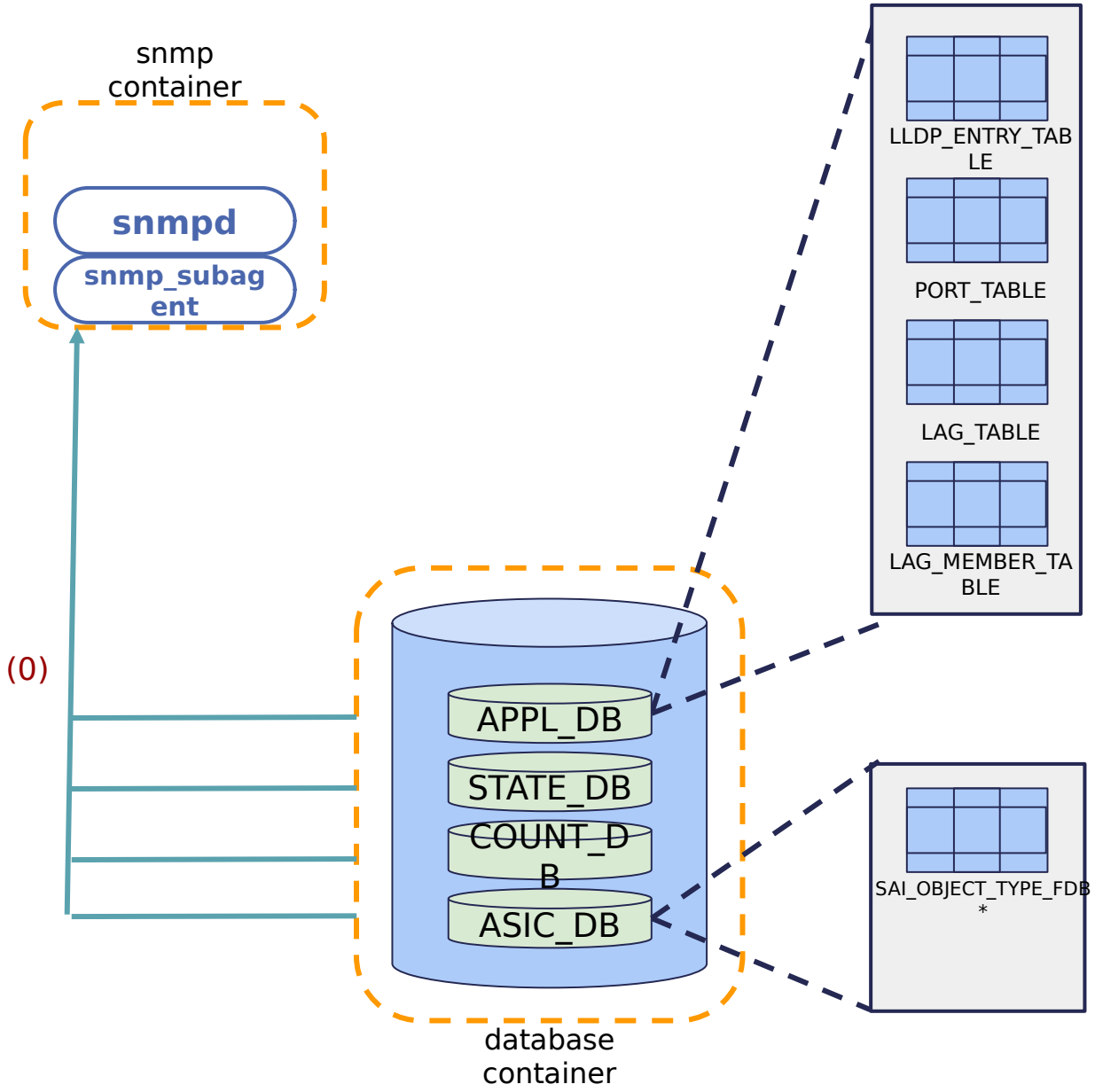


1. **Initialization**: subscribe to **state_db** to get live state of **physical ports**
2. **new LLDP** packet at **kernel** lldp socket. Gets received at **lldpd** (lldp process) and is **parsed**
3. **Lldp syncd**: receives information **digest** from **lldpd**
4. **push state**: to **APPL_DB**, to **LLDP_ENTRY_TABLE**
5. **Subscribers**: all the subscribers receive the updated state (**snmp**).



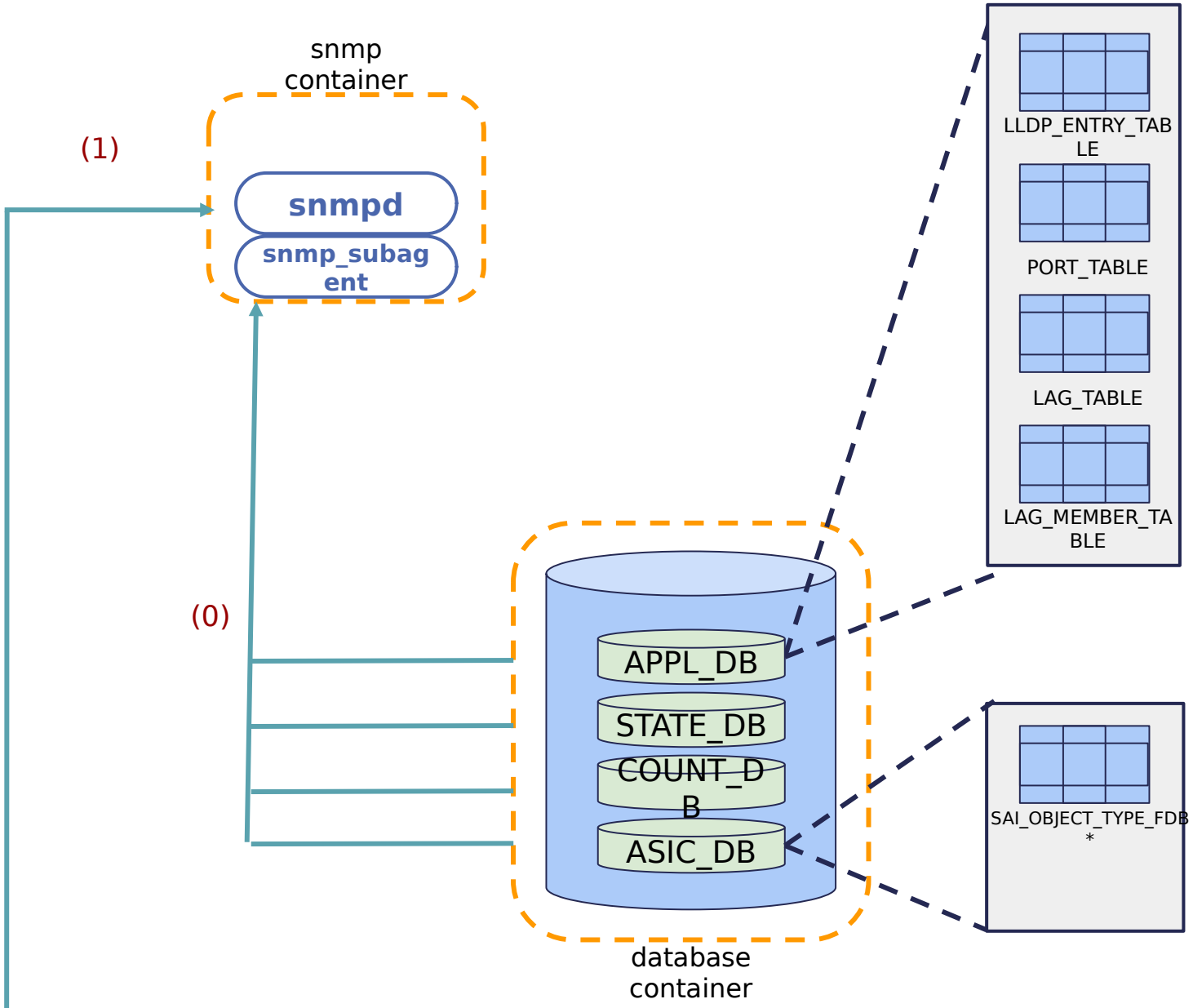
State Interactions: Snmp

1. **Initialization**:**connectivity** with various **DBs (local cache** refreshed within a second**)**



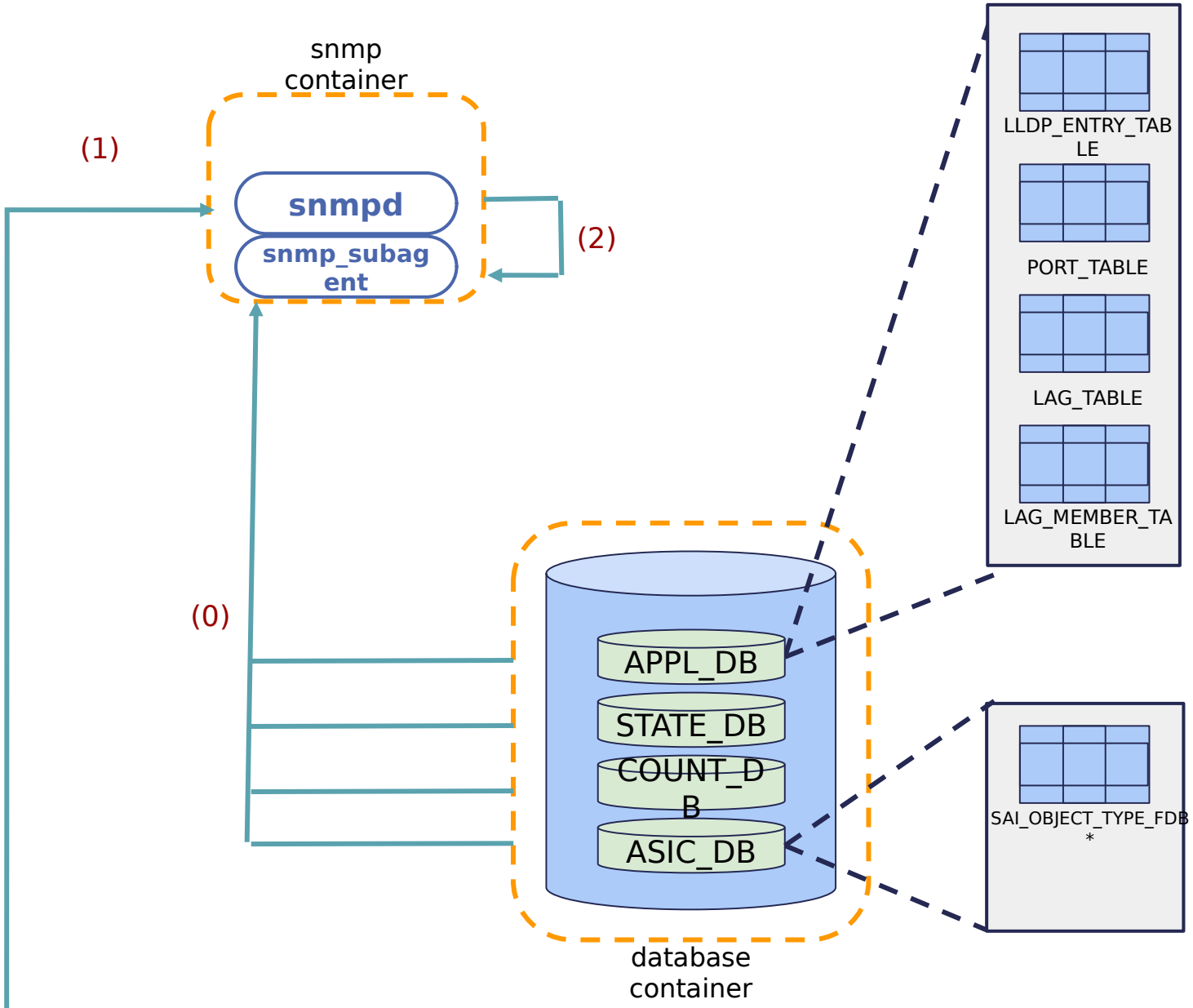
XFLOW RESEARCH

1. **Initialization**:**connectivity** with various **DBs** **(local cache** refreshed within a second**)**
2. **new query**: packet at **kernel** received at **snmpd** (snmpd process)



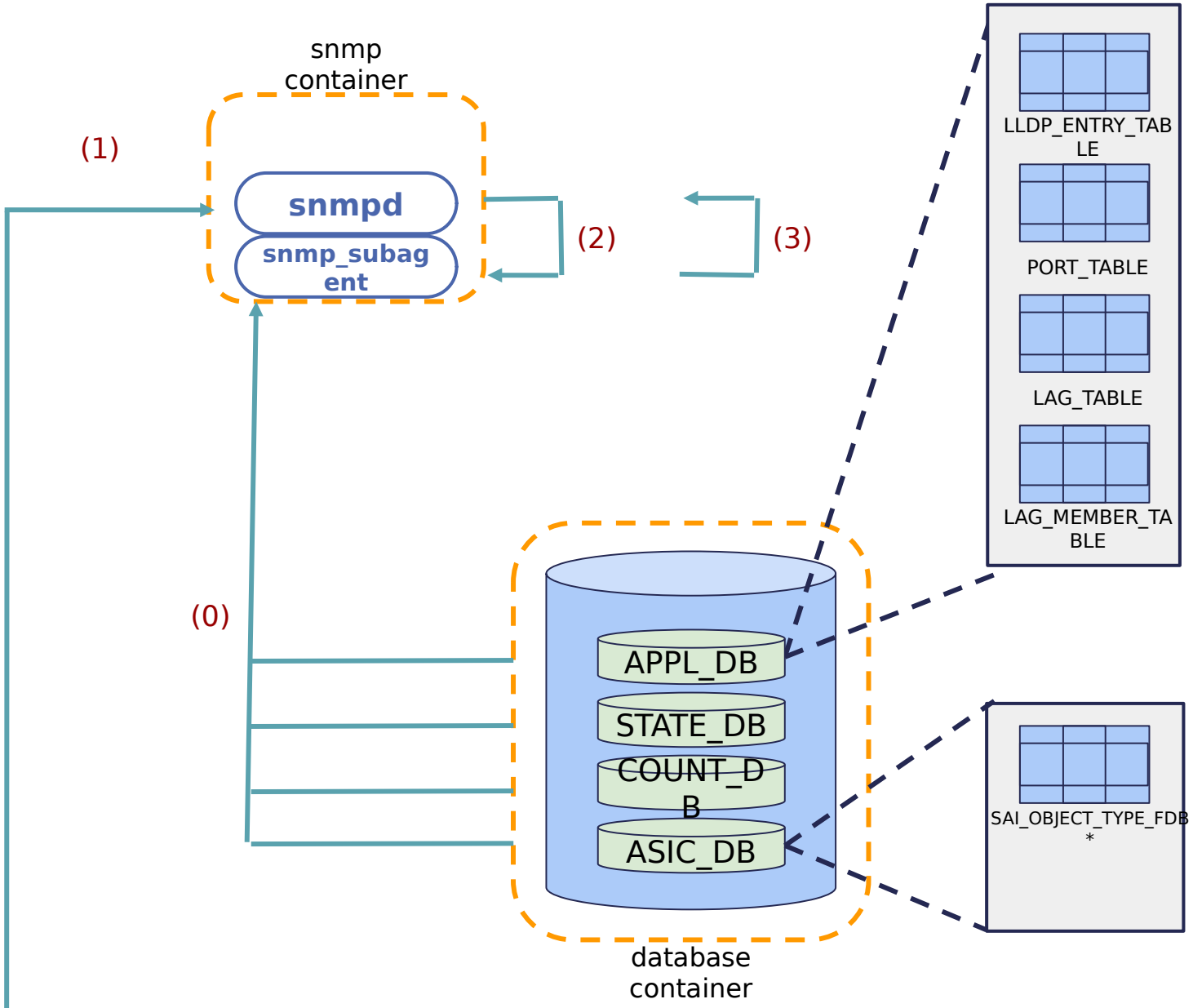
XFLOW RESEARCH

1. **Initialization**:**connectivity** with various **DBs** **(local cache** refreshed within a second**)**
2. **new query**: packet at **kernel** received at **snmpd** (snmpd process)
3. **snmpd**: receives and parses the request to forward to sub-agent



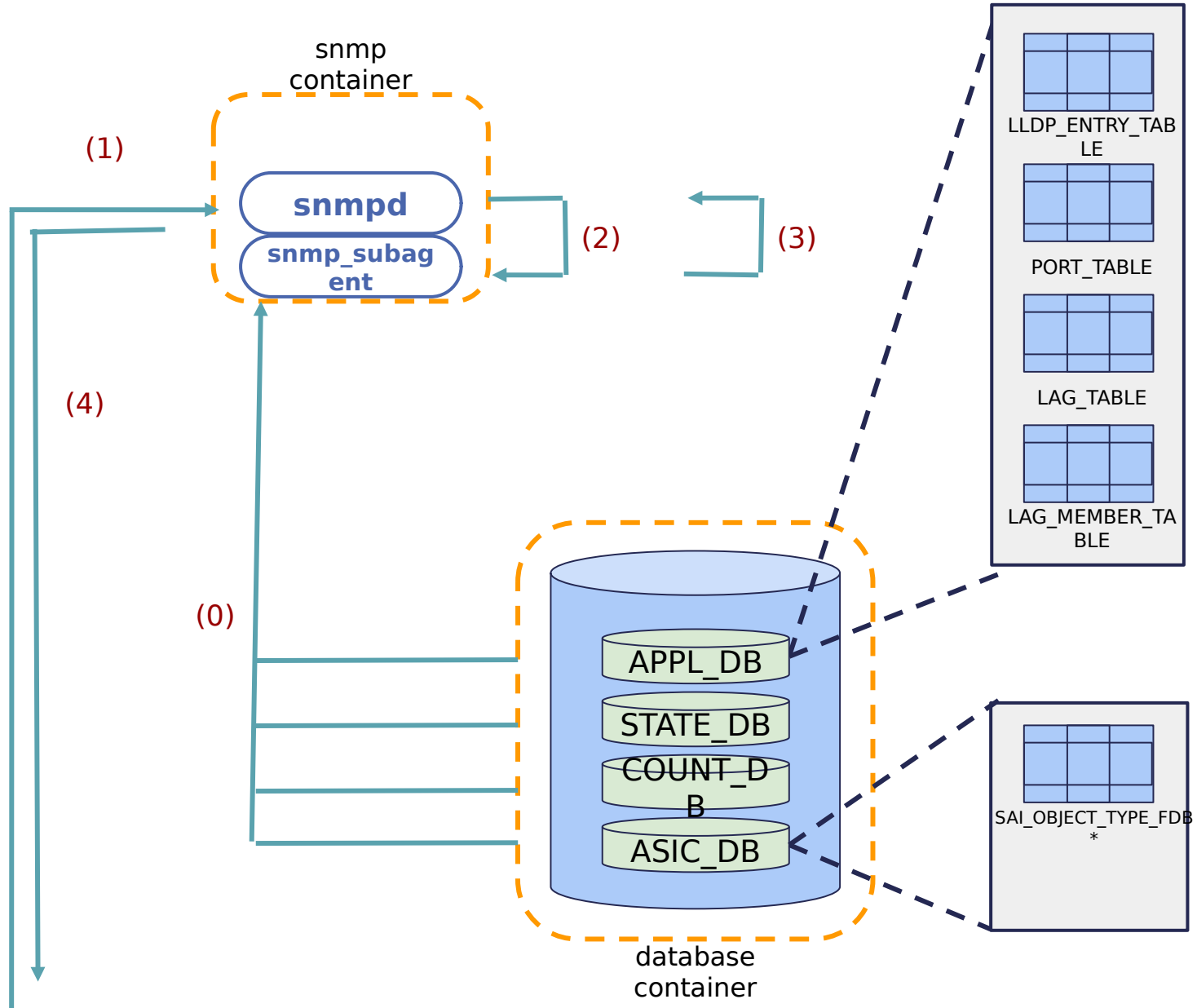
XFLOW RESEARCH

1. **Initialization**:**connectivity** with various **DBs (local cache** refreshed within a second**)**
2. **new query**: packet at **kernel** received at **snmpd** (snmpd process)
3. **snmpd**: receives and parses the request to forward to sub-agent
4. **serve query**: snmp agent **sends back** the information to **snmpd** process.



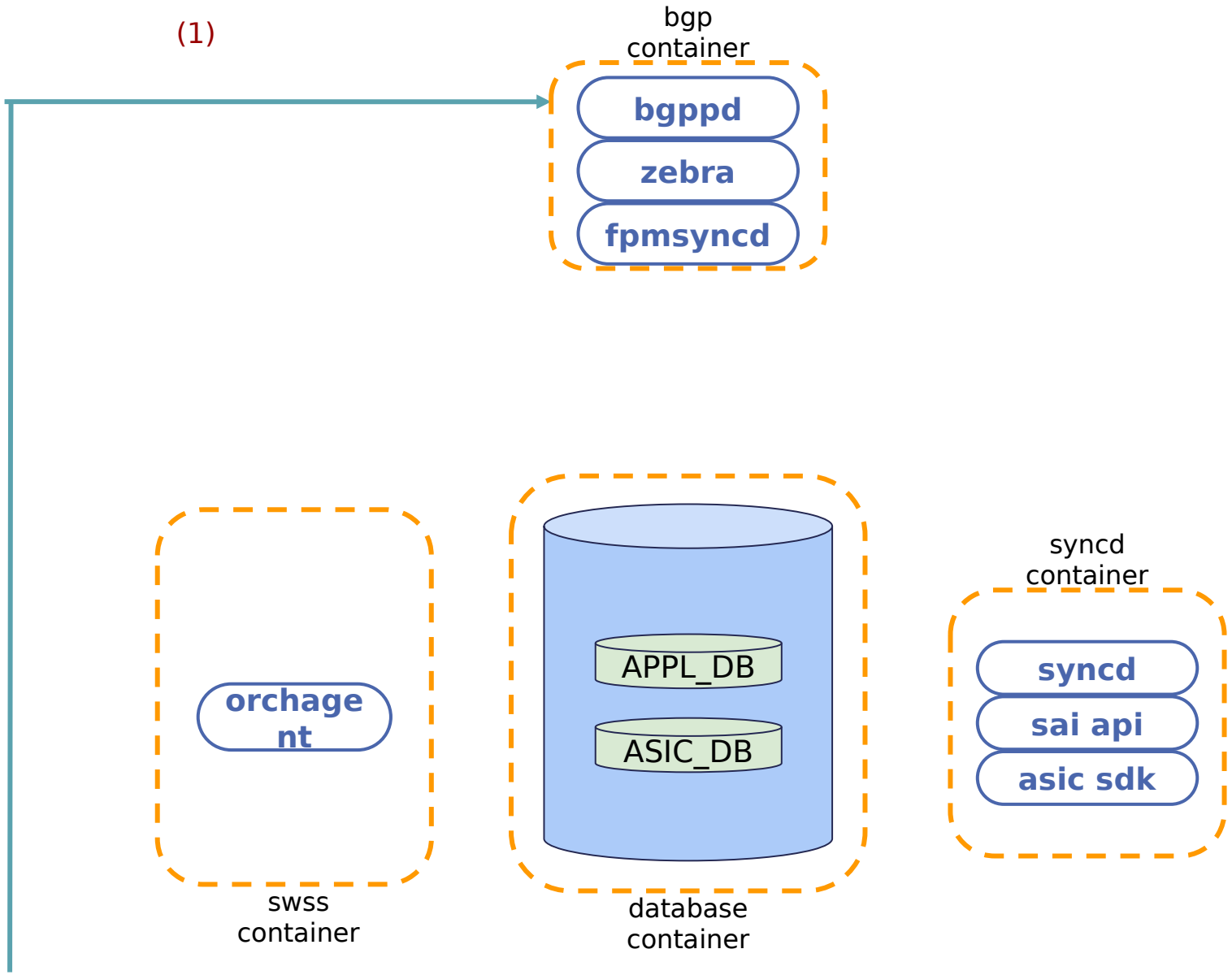
XFLOW RESEARCH

1. **Initialization**:**connectivity** with various **DBs** **(local cache** refreshed within a second**)**
2. **new query**: packet at **kernel** received at **snmpd** (snmpd process)
3. **snmpd**: receives and parses the request to forward to sub-agent
4. **serve query**: snmp agent **sends back** the information to **snmpd** process.
5. **reply back**: **snmpd** sends a reply back to the originator through

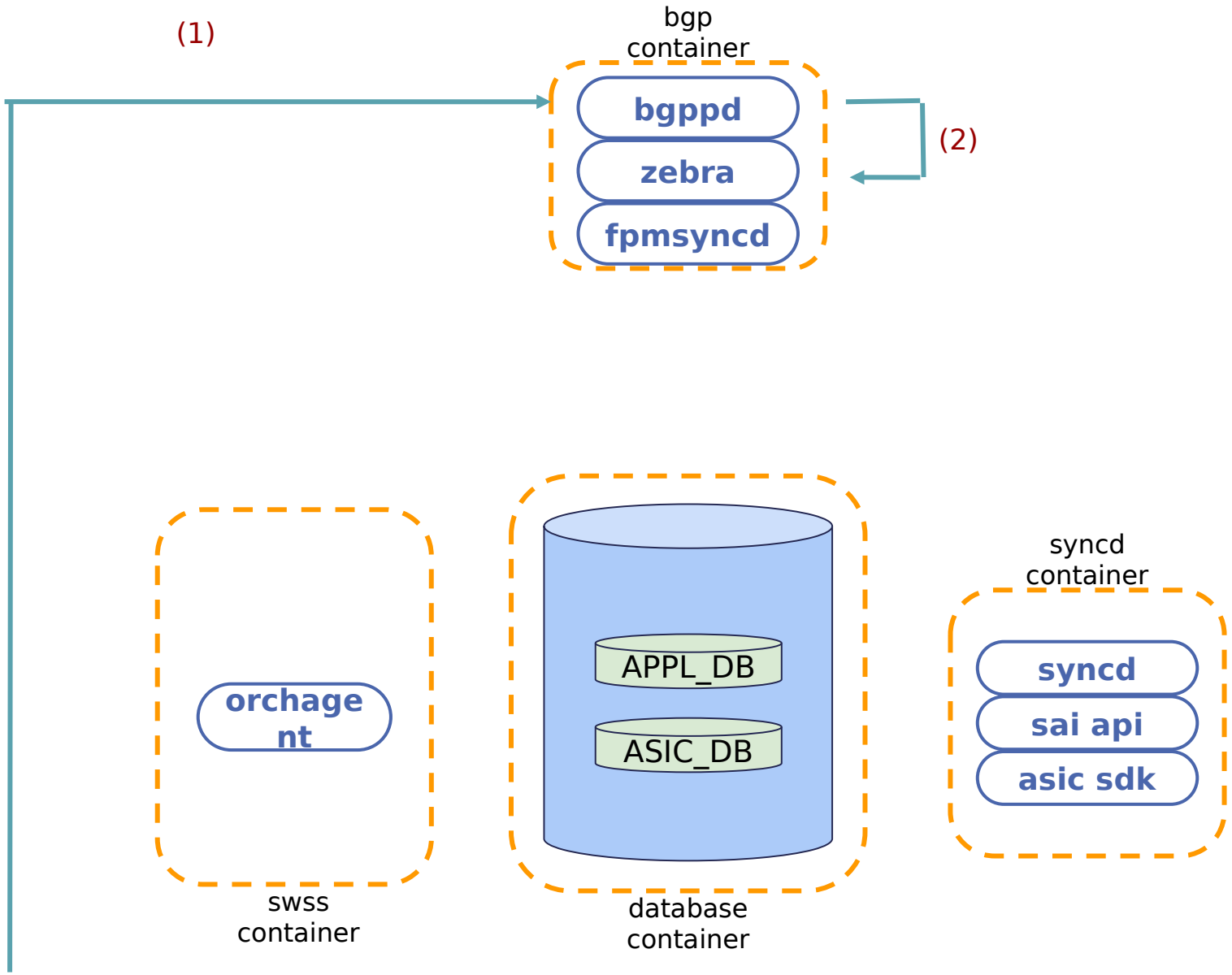


State Interactions: Routing (process a new route from eBGP peer)

1. **Initialization**: **zebra** connects to **fpmsyncd** using **TCP socket**
2. **new packet**: on **TCP** at **kernel** received. and passed to **bgpd**

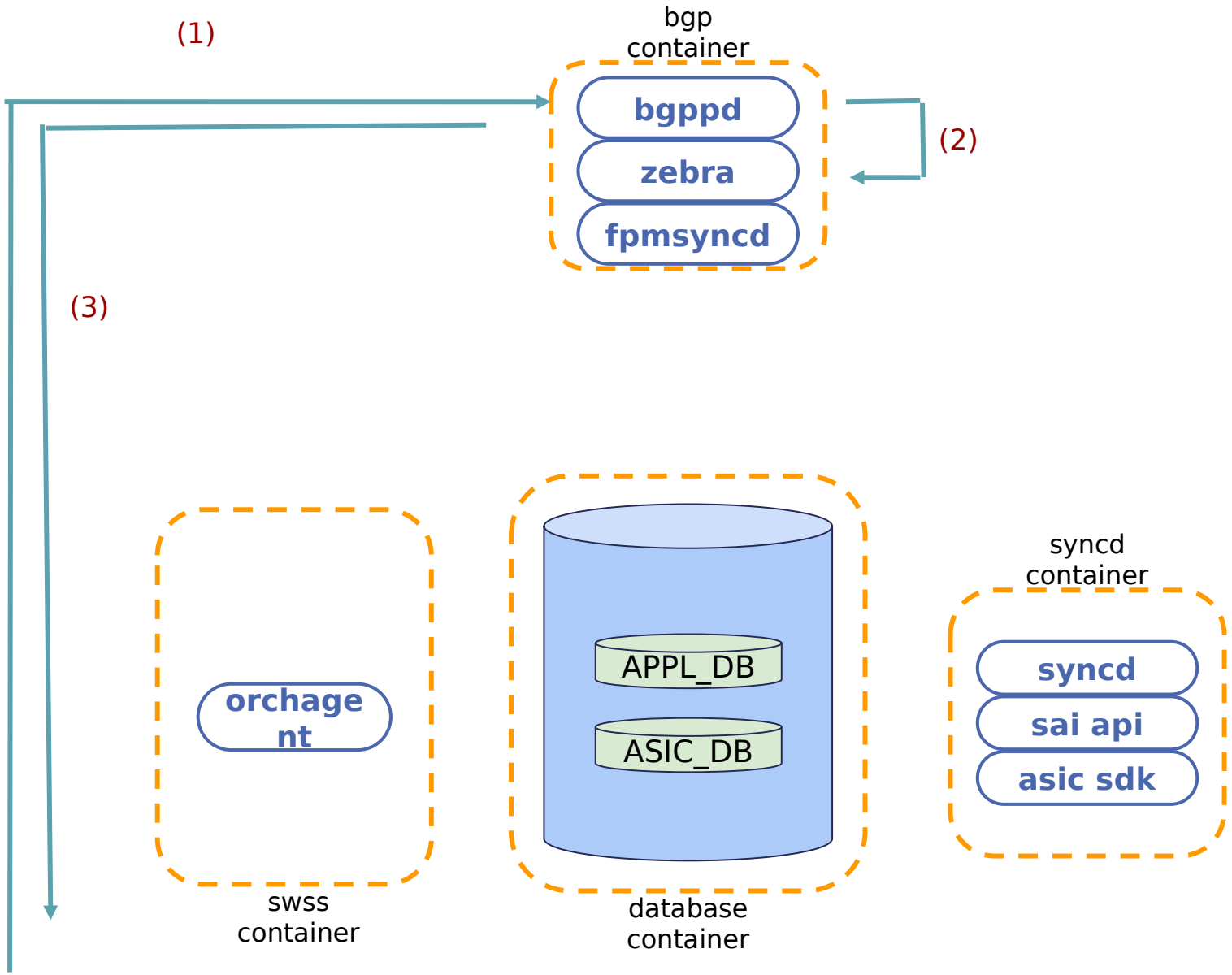


1. **Initialization**: **zebra** connects to **fpmsyncd** using **TCP socket**
2. **new packet**: on **TCP** at **kernel** received. and passed to **bgpd**
3. **bgp update**: after parsing, notify **zebra** about new info.



1. **Initialization**: **zebra** connects to **fpmsyncd** using **TCP socket**
2. **new packet**: on **TCP** at **kernel** received. and passed to **bgpd**
3. **bgp update**: after parsing, notify **zebra** about new info.
4. **Determine**: **feasibility/ reachability**, inject in **kernel**

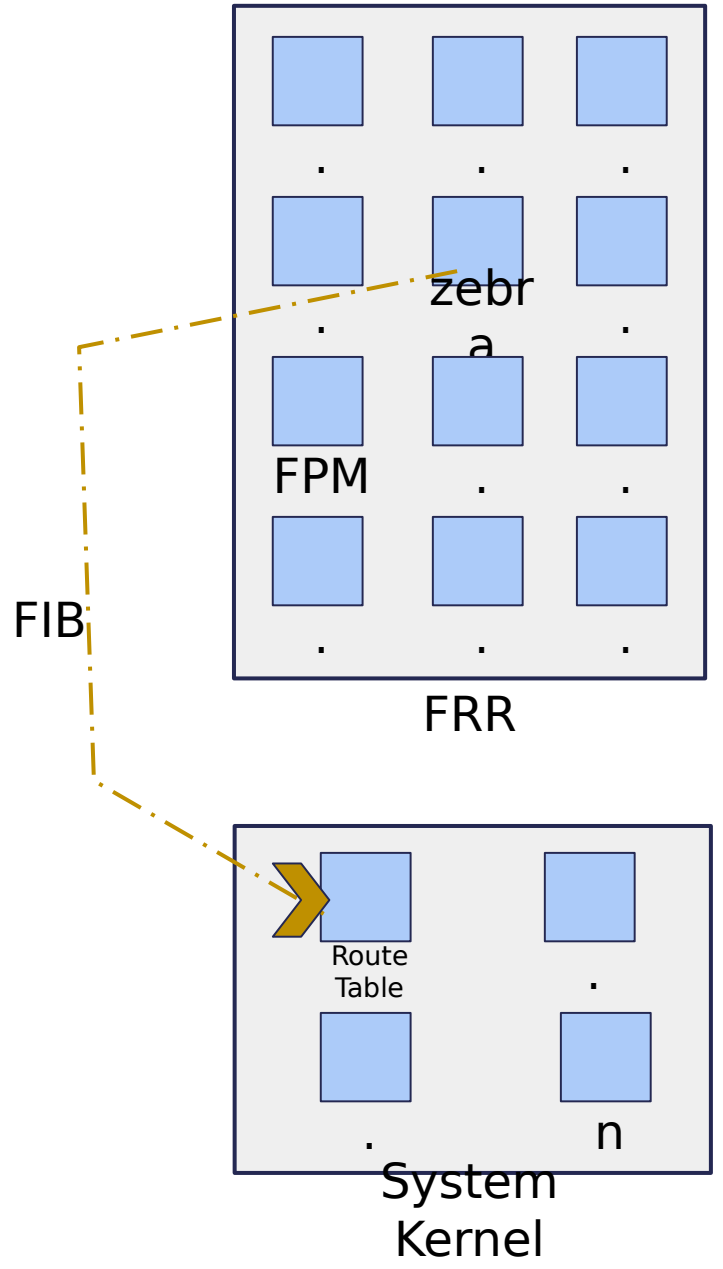
NOTE: For kernel's route table



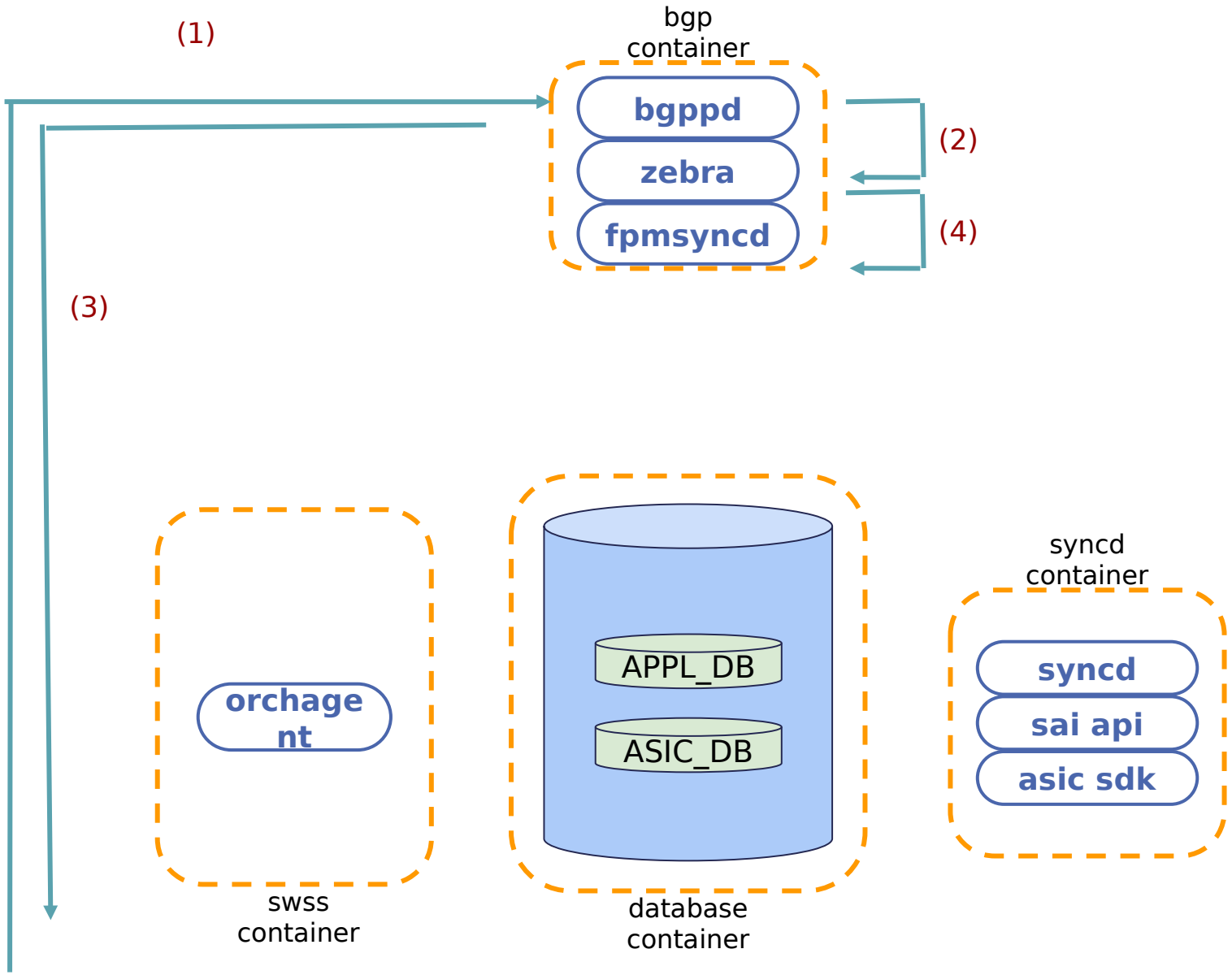
XFLOW RESEARCH

Understanding the important keywords

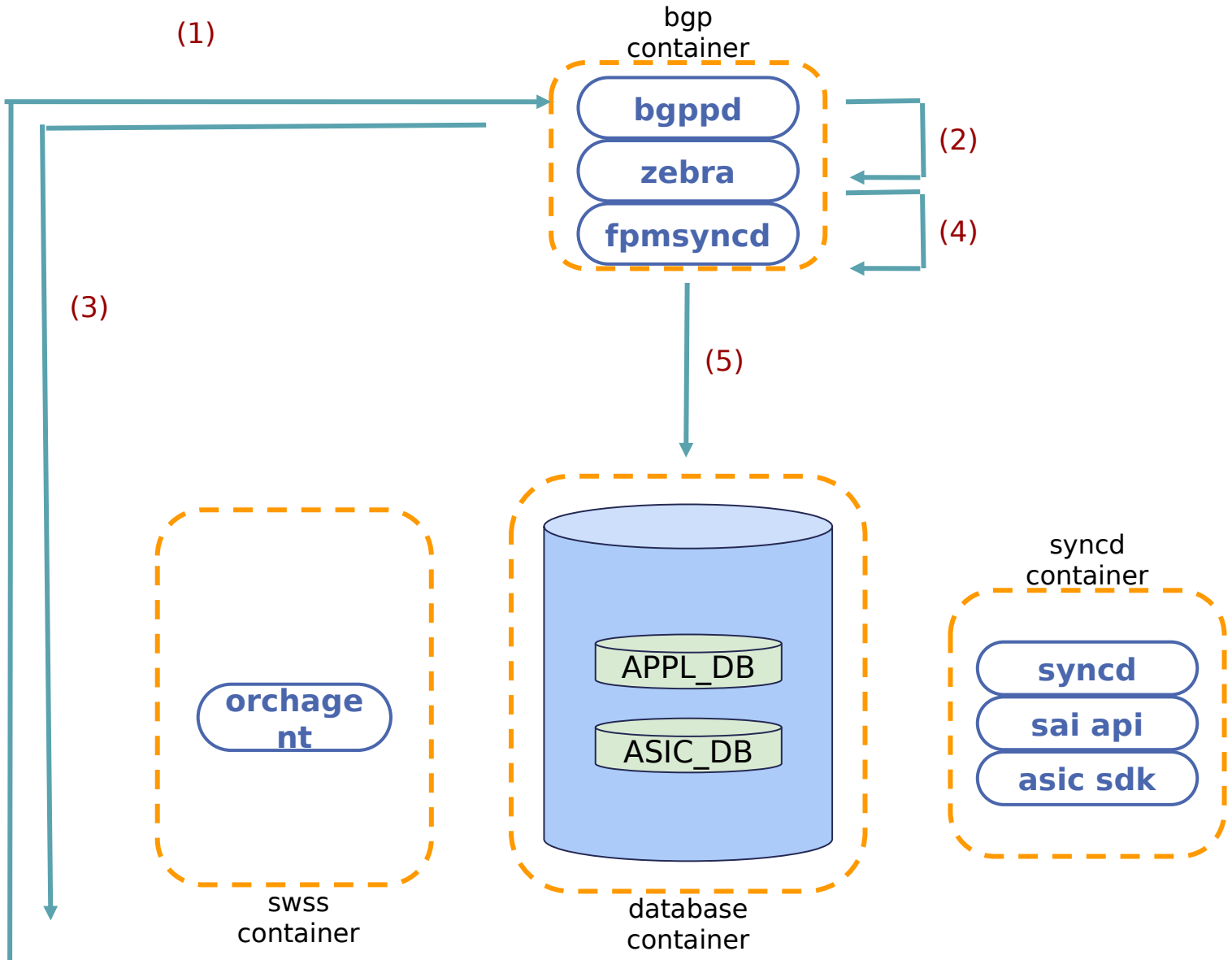
1.	FRR (Open Source)	**Free Range Routing**, a suite/package of tools for network configurations
2.	Zebra (Open Source)	**Part of FRR**, part of FRR as a sub-package. Works as a route engine. Manages, various protocols. provides 1. kernel routing-table updates 2. interface-lookups and route-redistribution services across different protocols 3. Zebra takes care of **pushing** the **computed FIB** down, **a. kernel** (through netlink interface) b. to **south-bound components** involved in the forwarding process (through FPM-interface).
4.	FPM (Open Source)	**Part of FRR**, as a module Forwarding Plane Manager. Complements zebra in message formats, like **Netlink** or **Protobuf** for various protocols.
5.	FIB	Forwarding Information Base, has info about next hops



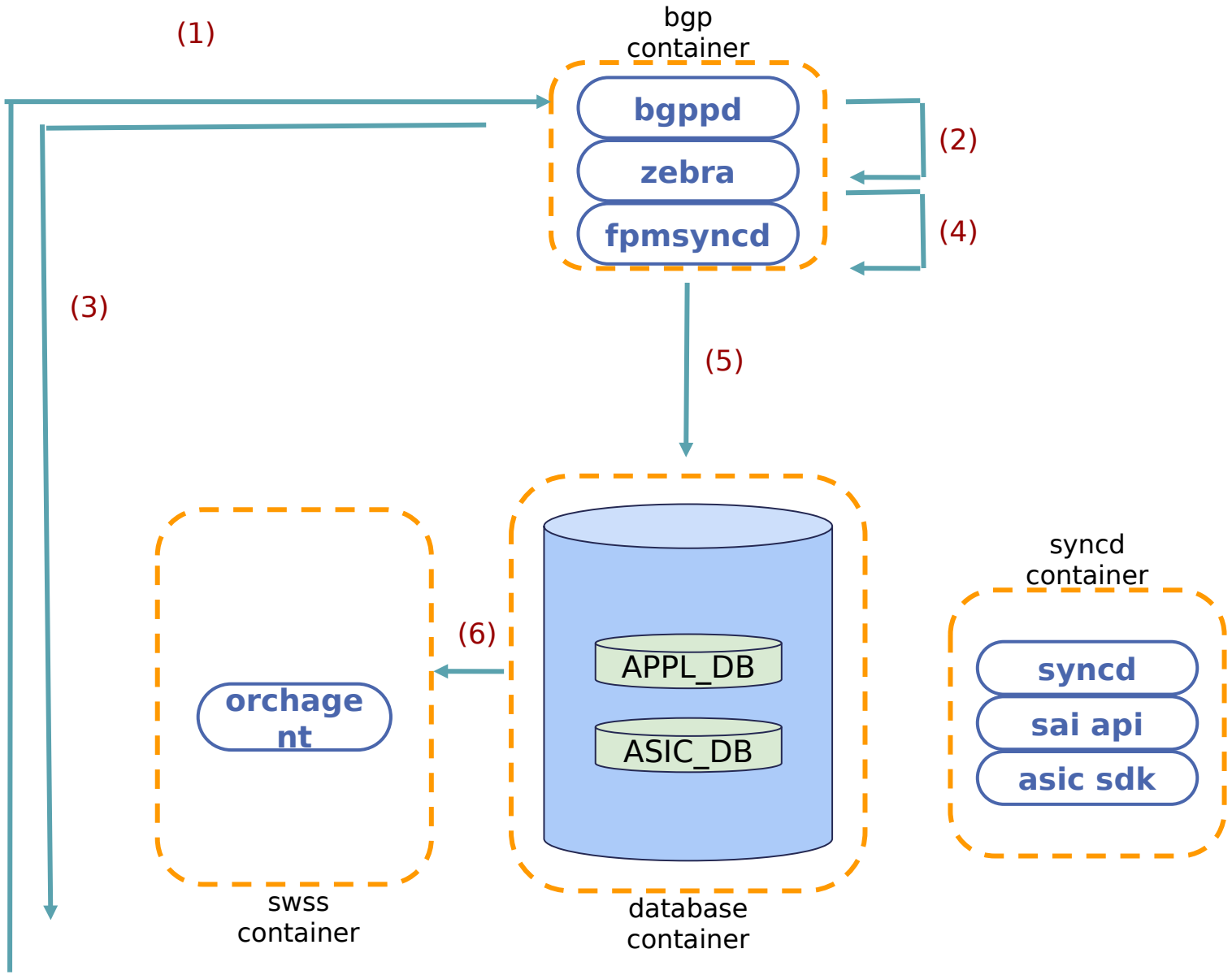
1. **Initialization**: **zebra** connects to **fpmsyncd** using **TCP socket**
2. **new packet**: on **TCP** at **kernel** received. and passed to **bgpd**
3. **bgp update**: after parsing, notify **zebra** about new info.
4. **Determine**: **feasibility/ reachability**, inject in **kernel**
5. **Deliver**: to **fpmsyncd** using FPM interface.



1. **Initialization**: **zebra** connects to **fpmsyncd** using **TCP socket**
2. **new packet**: on **TCP** at **kernel** received. and passed to **bgpd**
3. **bgp update**: after parsing, notify **zebra** about new info.
4. **Determine**: **feasibility/ reachability**, inject in **kernel**
5. **Deliver**: to **fpmsyncd** using FPM interface.
6. **Push state**: **fpmsyncd** processes and pushes state to the **APPL_DB**

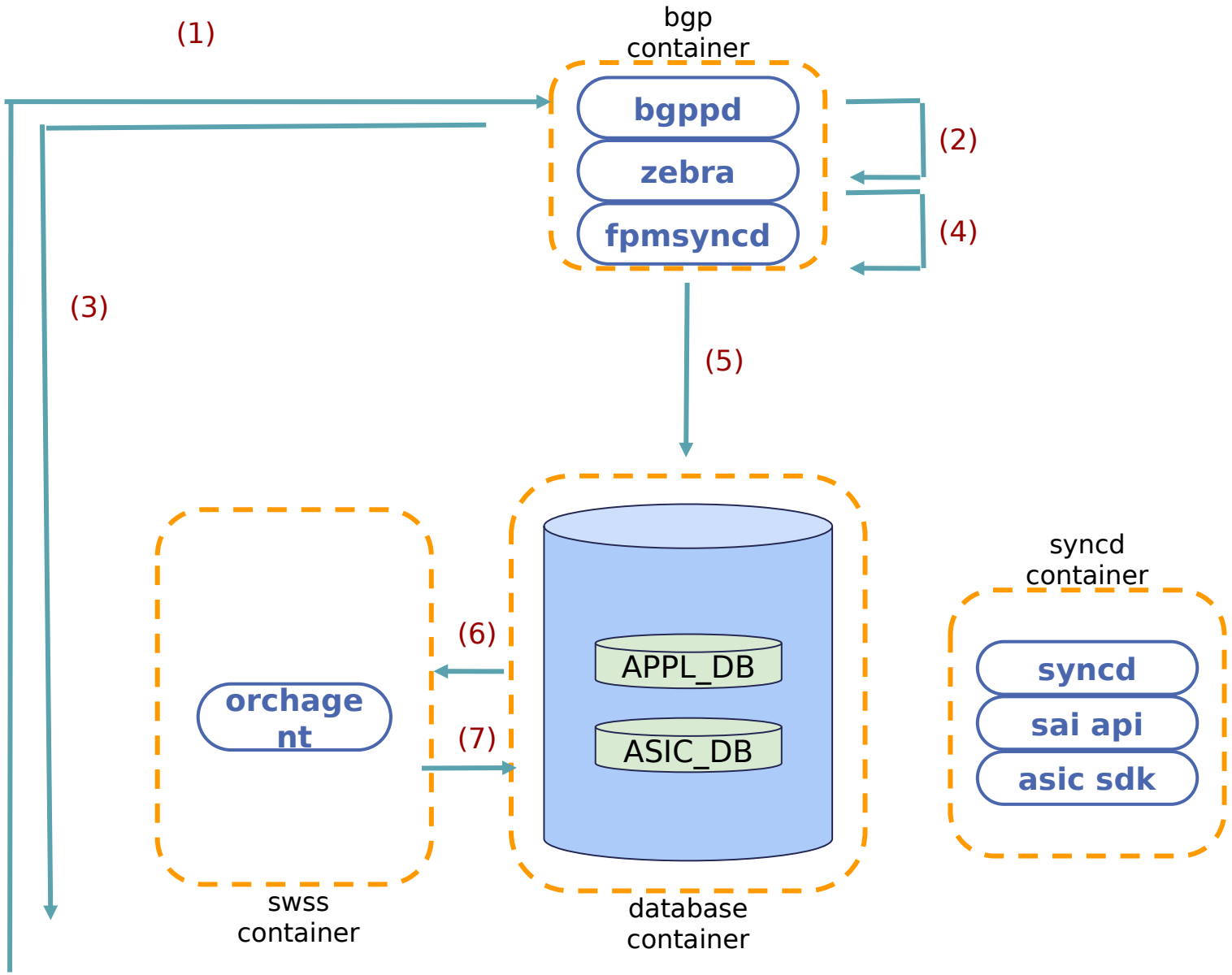


6. Subscribers notified: **orchagent** receives data from APPL_DB

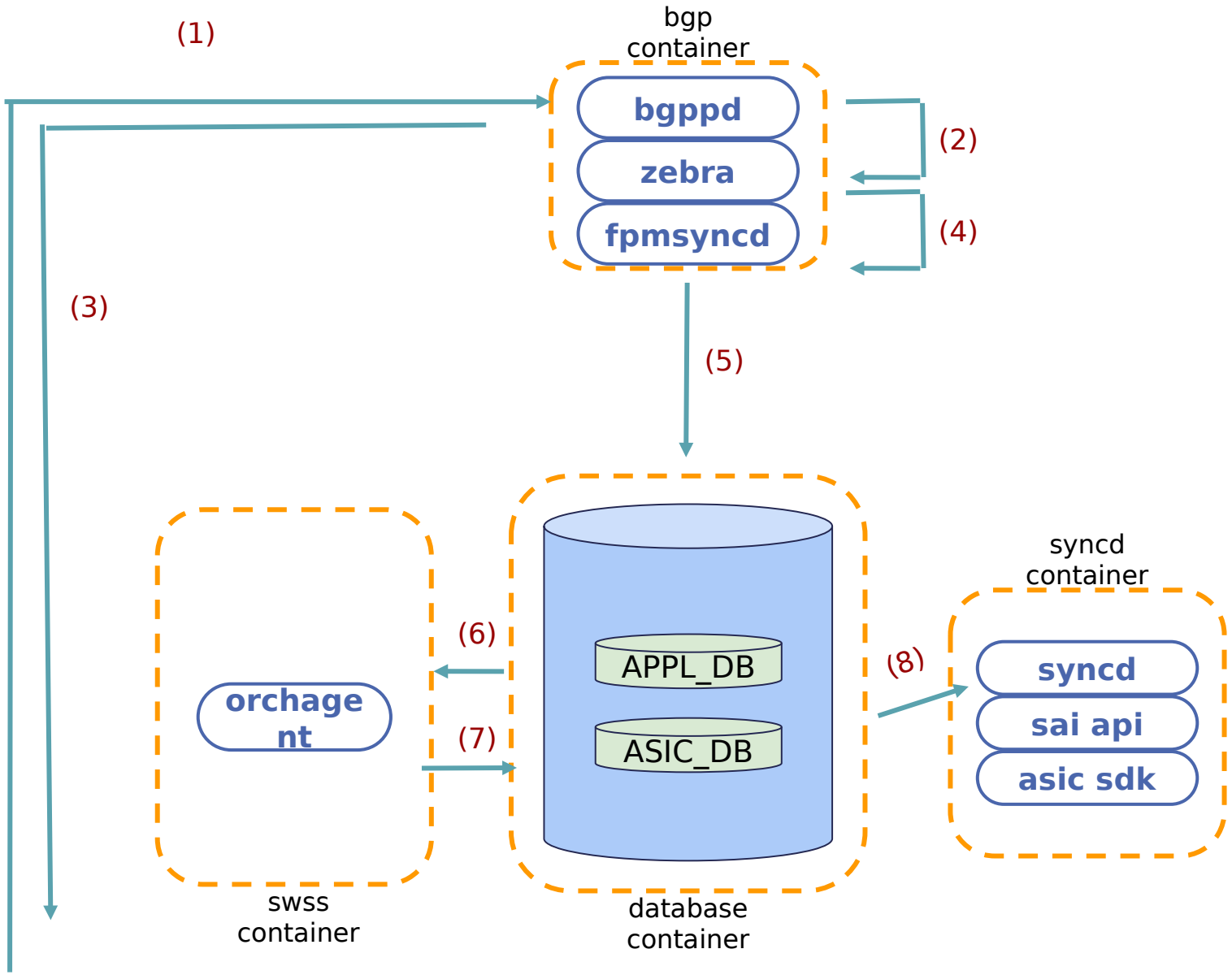


XFLOW RESEARCH

6. **Subscribers notified**: **orchagent** receives data from APPL_DB
7. **Inject Route (switch)**: after processing, **route** pushed to **ASIC_DB** using **sairedis APIs**

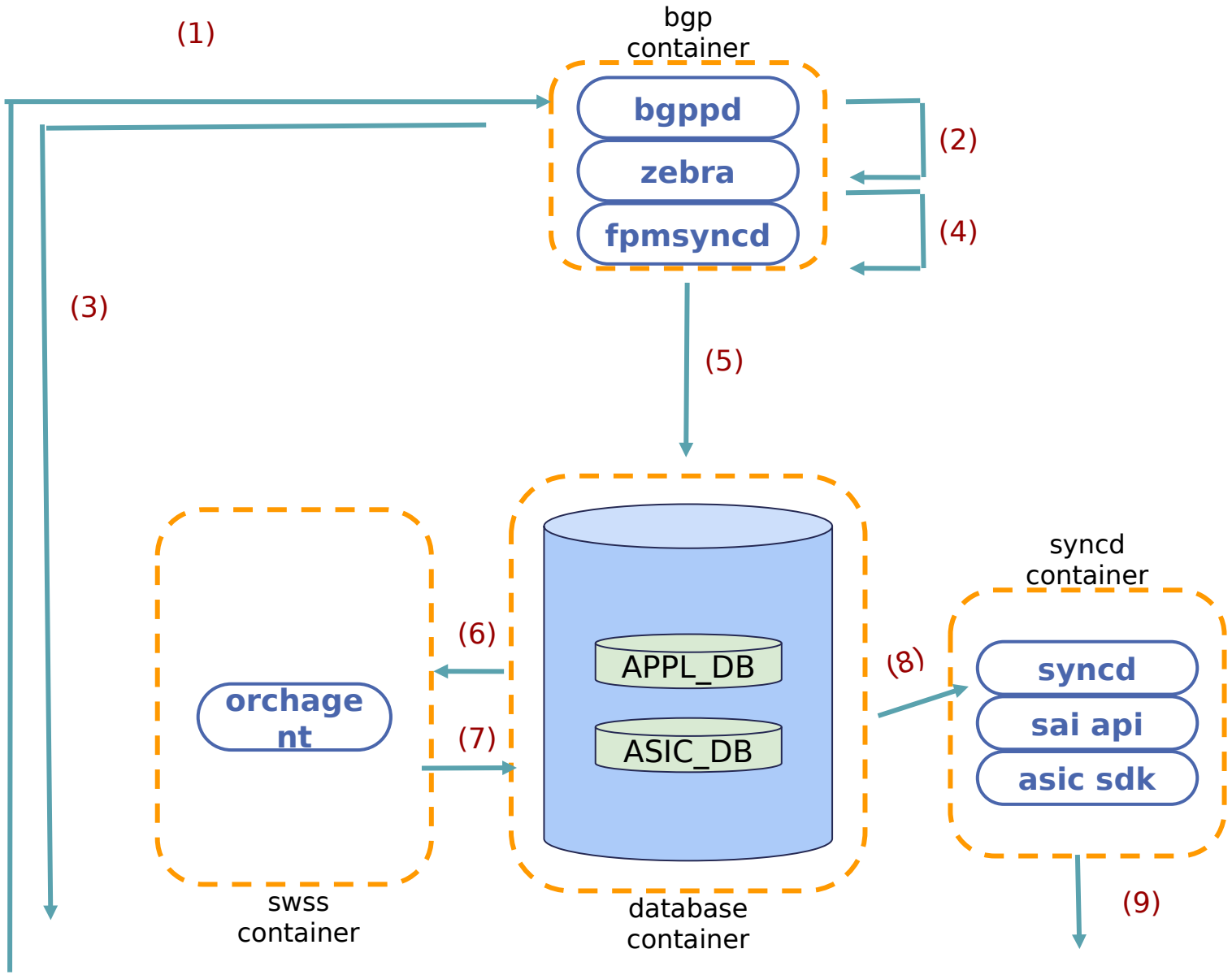


6. **Subscribers notified**: **orchagent** receives data from APPL_DB
7. **Inject Route (switch)**: after processing, **route** pushed to **ASIC_DB** using **sairedis APIs**
8. **Info Served**: to the subscriber, **syncd**



6. **Subscribers notified**: **orchagent** receives data from APPL_DB
7. **Inject Route (switch)**: after processing, **route** pushed to **ASIC_DB** using **sairedis APIs**
8. **Info Served**: to the subscriber, **syncd**
9. **Inject state to driver**: using **SAI** APIs

Hence, new route is pushed to the hardware.



XFLOW RESEARCH

State Interactions: Port (related to layer 2)

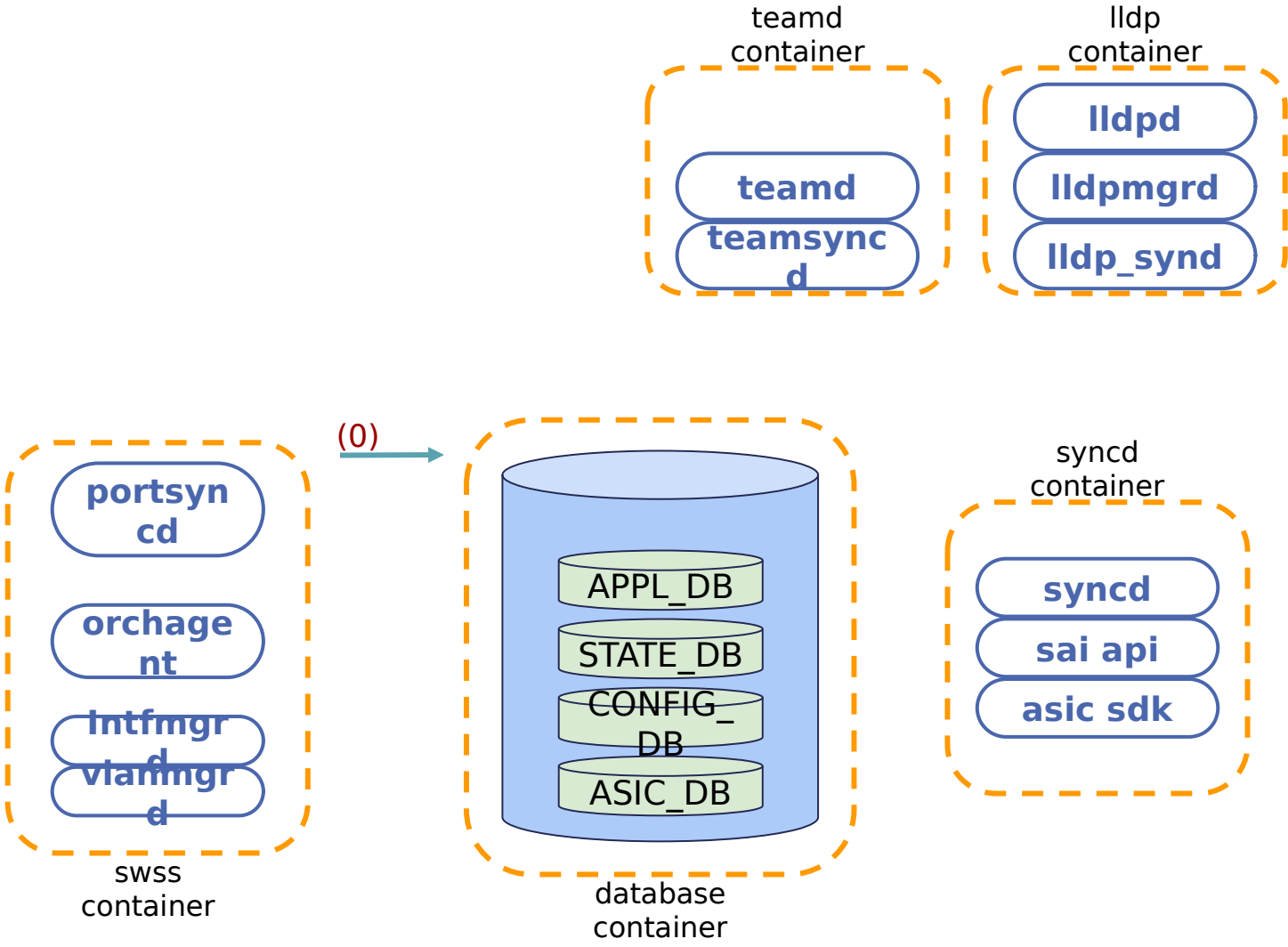
1. Initialization:

portsyncd, establishes communication channels

pub -> APPL_DB, STATE_DB

sub -> CONFIG_DB**,**

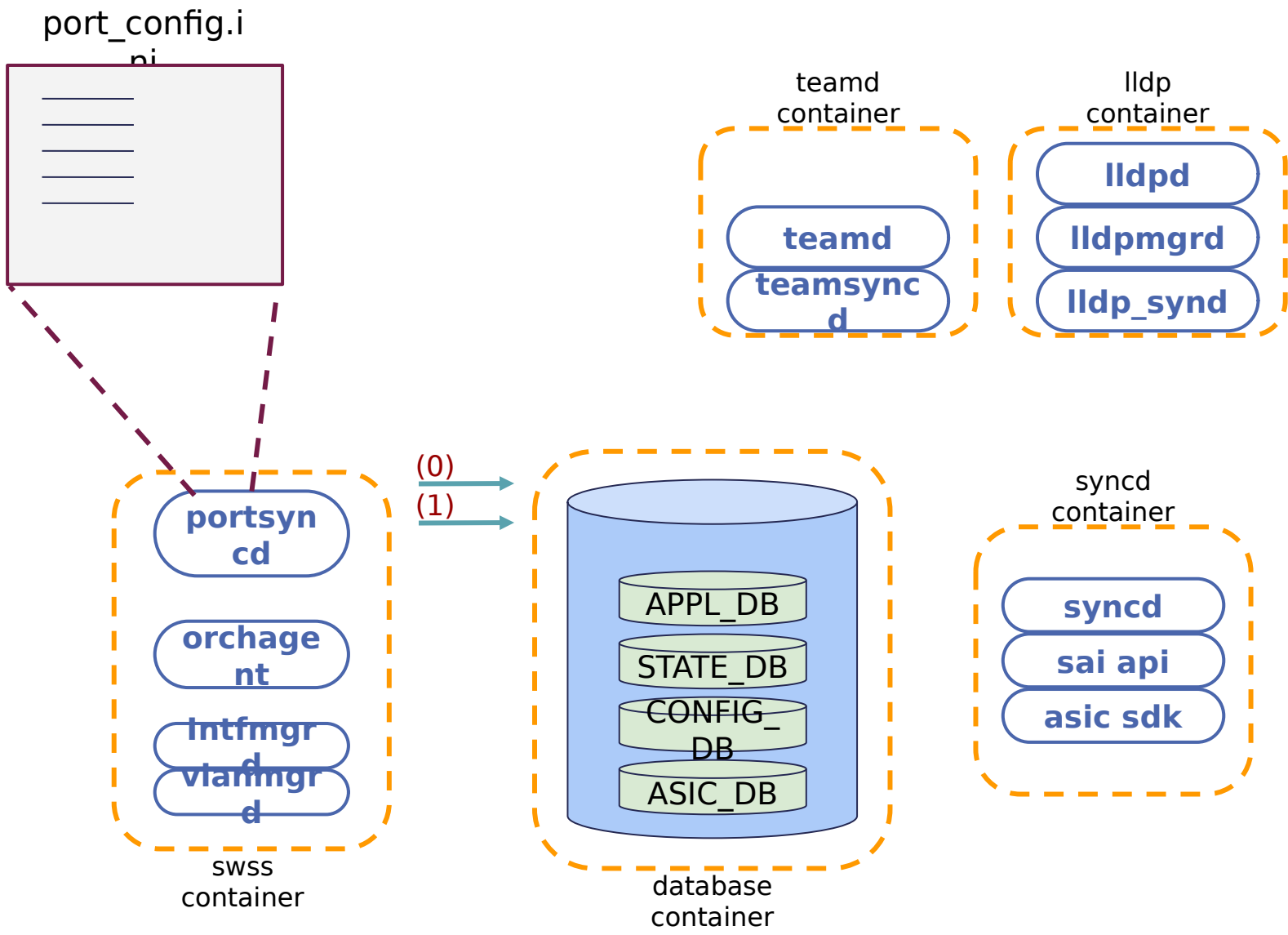
pub	1. APPL_DB 2. STATE_DB
sub	1. CONFIG_DB **2. netlink** (port/link state)



XFLOW RESEARCH

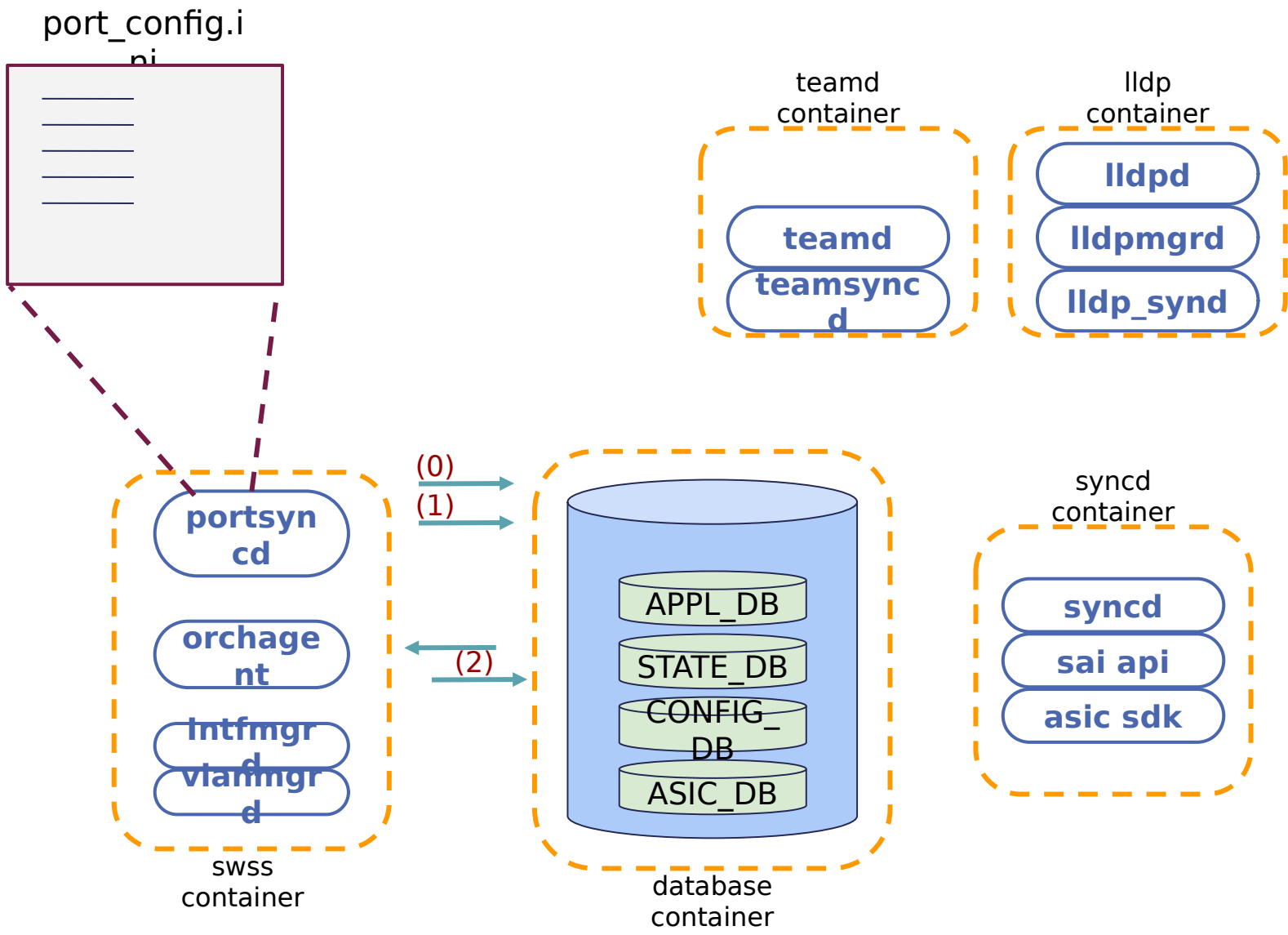
1. **Initialization**:
   portsyncd,
   pub -> APPL_DB, STATE_DB
   sub -> CONFIG_DB**,**
   netlink(system/port config)
2. **config set**:
   portsyncd, parses the port-config file.
   port_config.ini

   And sets the **channel** for **database** communication, accordingly.

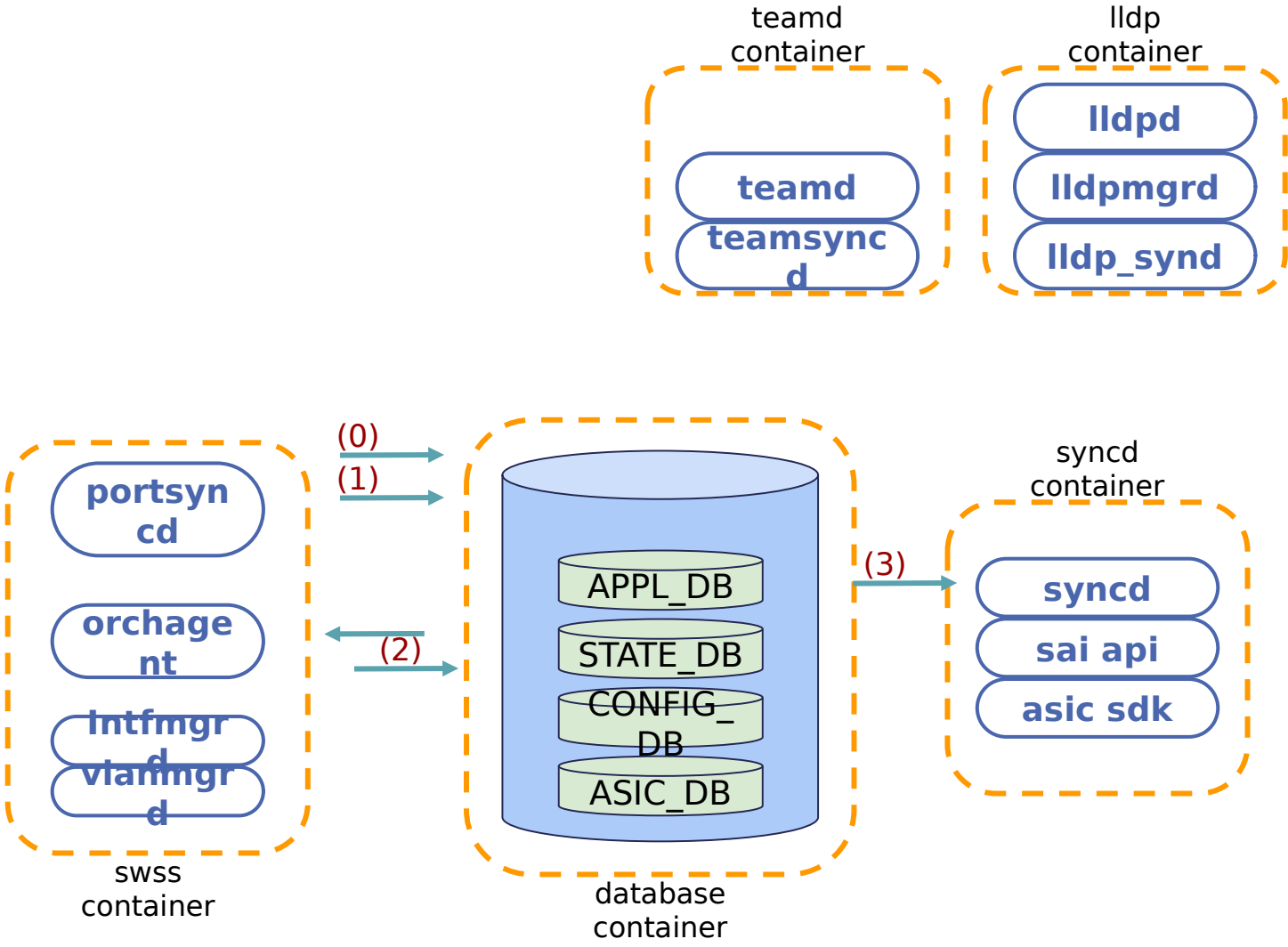


1. **Initialization**: **portsyncd, pub** -> APPL_DB, STATE_DB **sub** -> CONFIG_DB**, netlink**(system/port config)
2. **config set**: **portsyncd,** parses the port-config file. **port_config.ini**
3. **await**, **orchagent** will

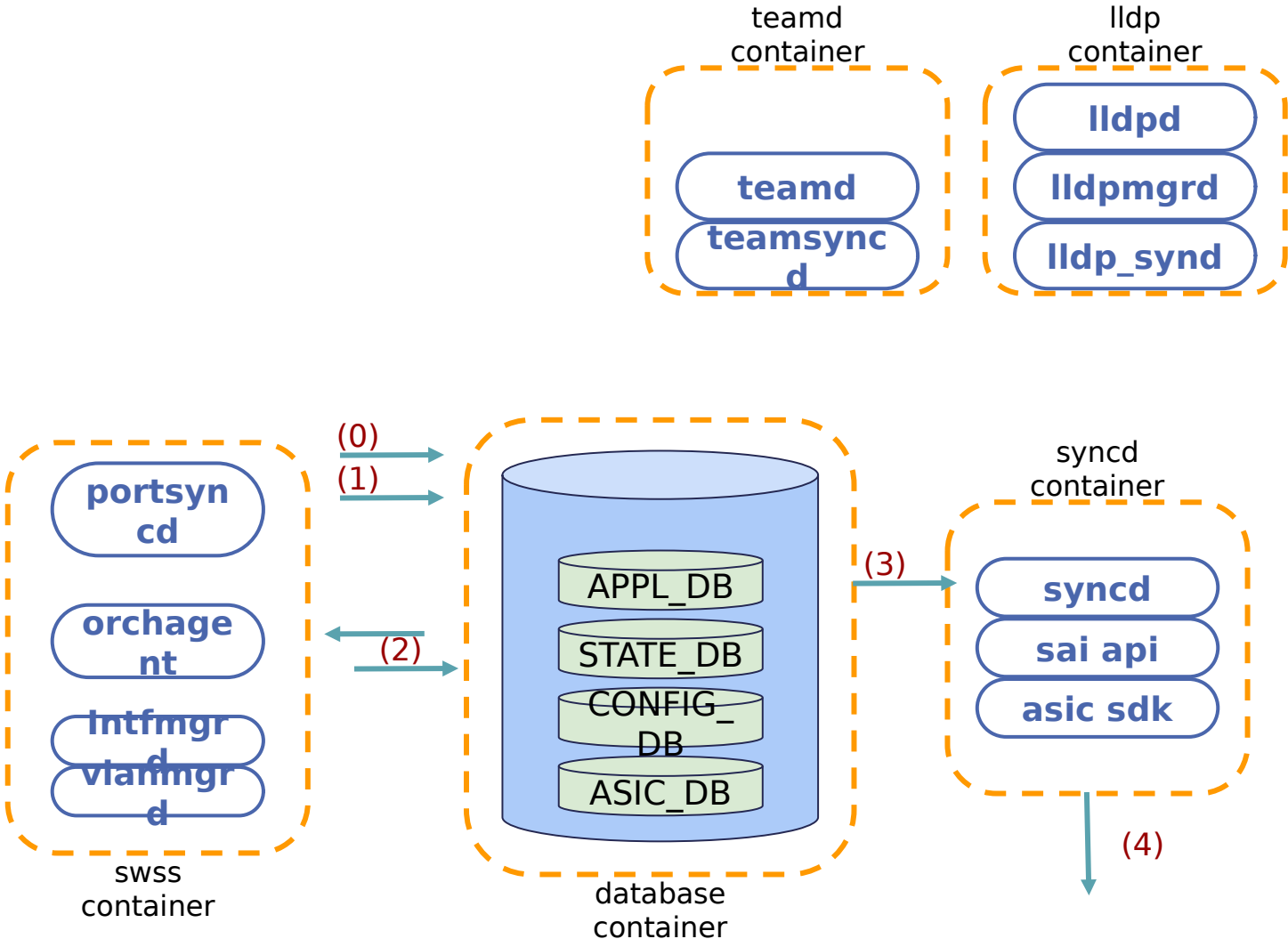
initialization	1. port 2. interfaces
invoke	sairedis APIs, deliver to syncd through ASIC_DB interface



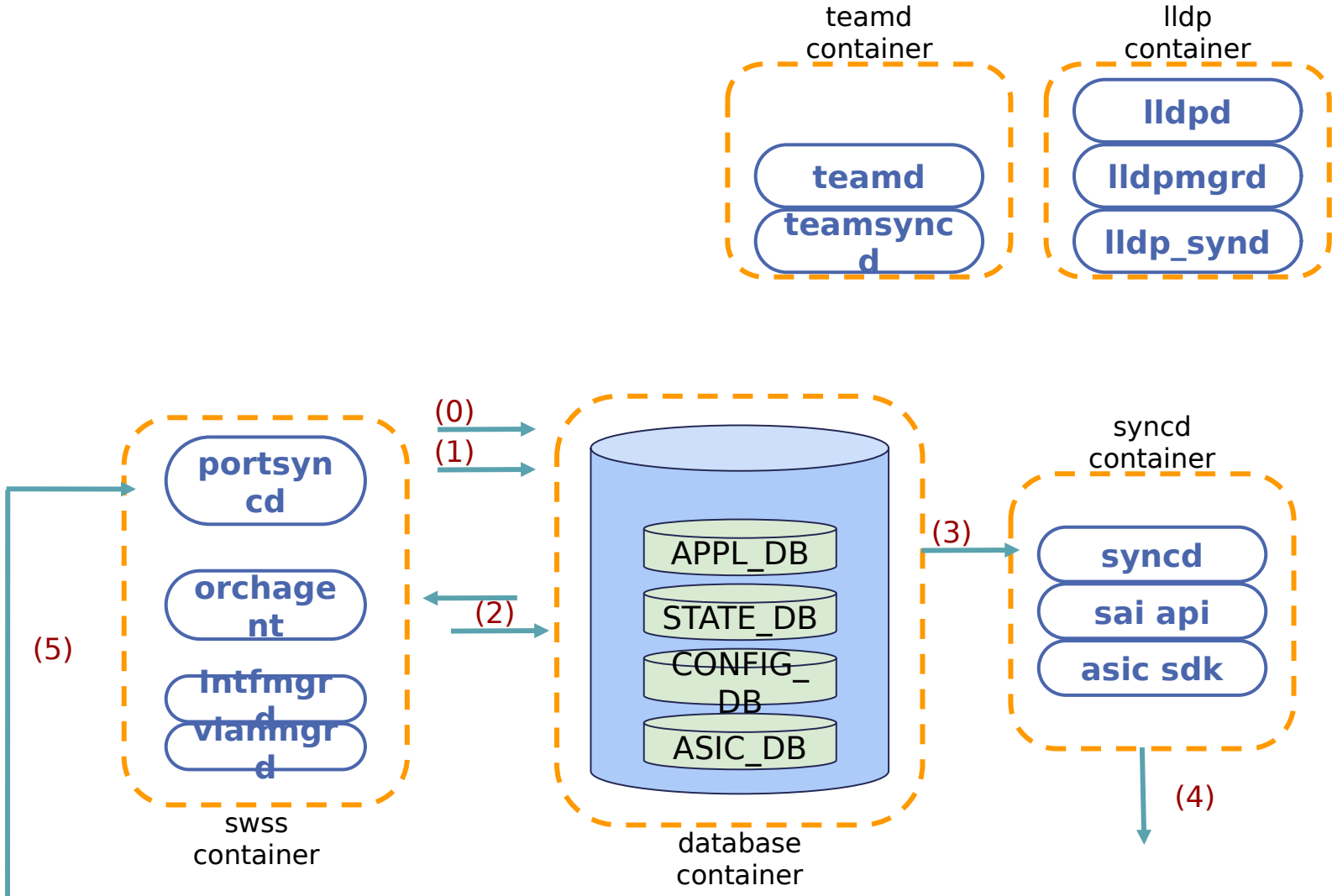
1. **Initialization**: **portsyncd, pub** -> APPL_DB, STATE_DB **sub** -> CONFIG_DB**, netlink**(system/port config)
2. **config set**: **portsyncd,** parses the port-config file. **port_config.ini**
3. **await: notification** from **portsyncd, orchagent initailizes** interfaces & ports along with ASIC_DB interface
4. **new request:** at **syncd** through



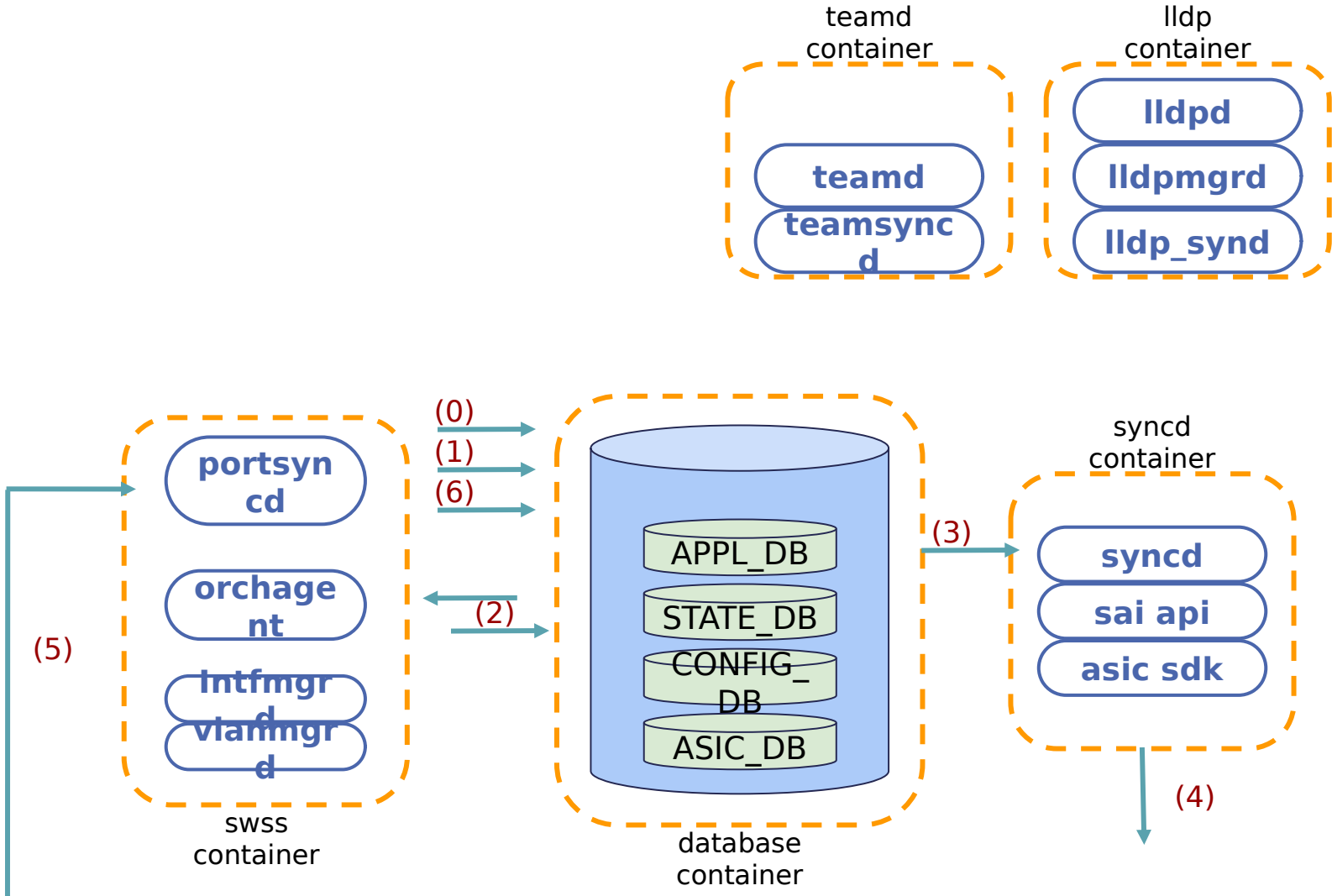
4. associate port: at **kernel space + ASIC** using **ASIC**-SDK and **SAI** APIs



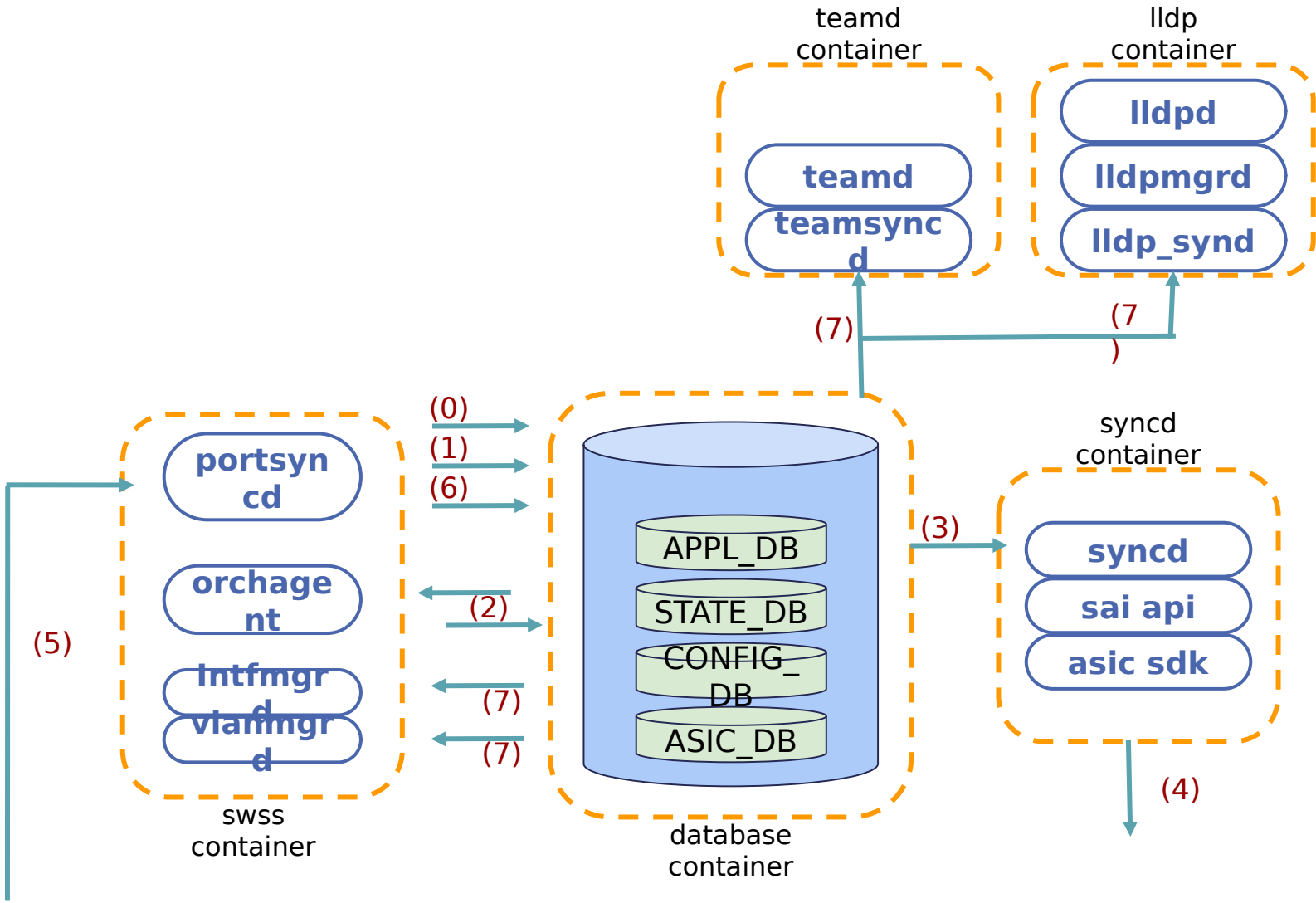
4. **associate port:** at **kernel space + ASIC** using **ASIC**-SDK and **SAI** APIs
5. **netlink response:** at **portsyncd** a **netlink** message received, **initialization complete (ASIC state change)**



4. **associate port:** at **kernel space + ASIC** using **ASIC**-SDK and **SAI** APIs
5. **netlink response:** at **portsyncd** a **netlink** message received, **initialization complete**
6. **record entry: success** to **STATE_DB** for each port

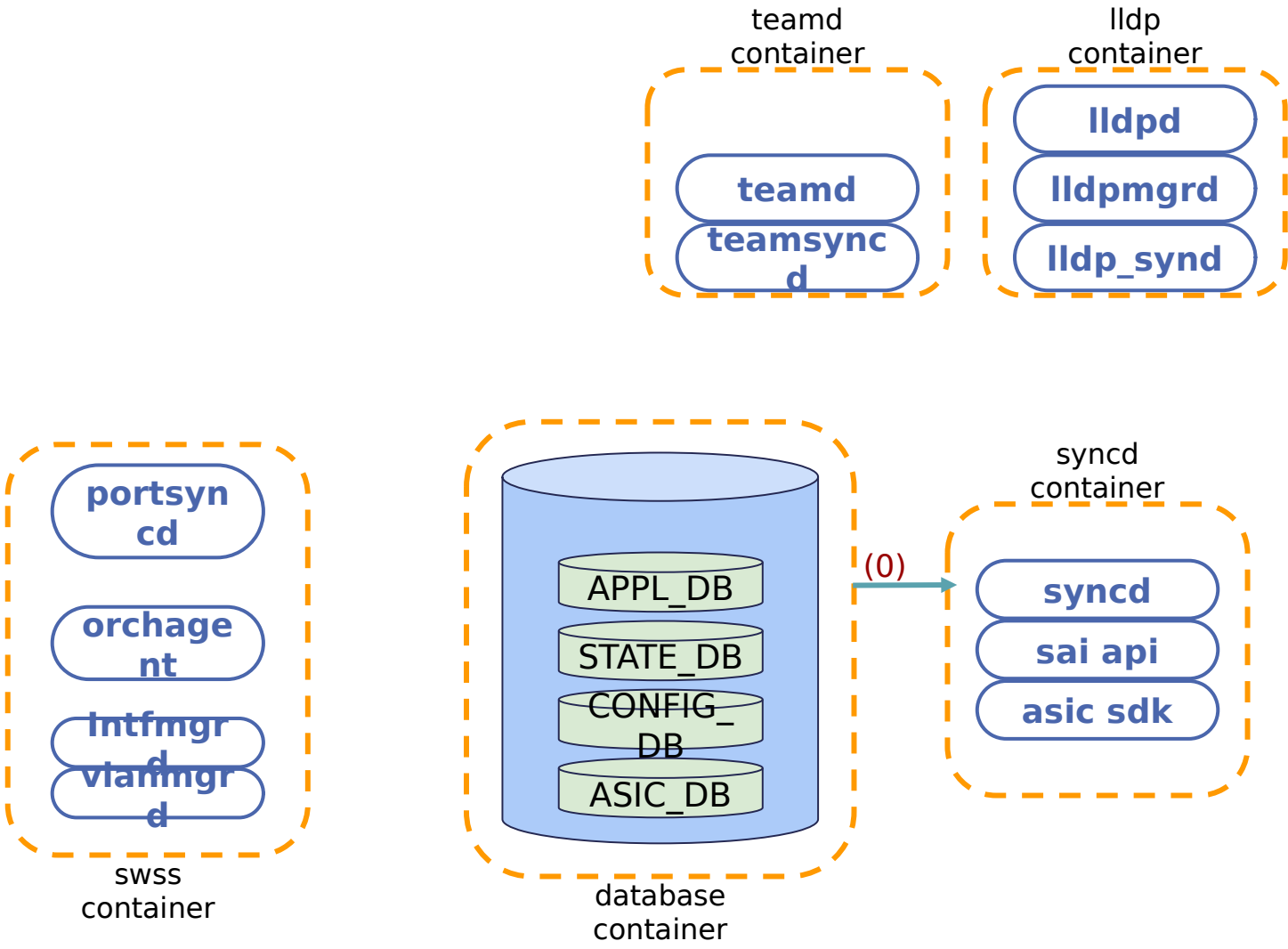


4. **associate port:** at **kernel space + ASIC** using **ASIC**-SDK and **SAI** APIs
5. **netlink response:** at **portsyncd** a **netlink** message received, **initialization complete**
6. **record entry: success** to **STATE_DB** for each port
7. **subscribers served:** with the **latest data** of STATE_DB, they start using these ports

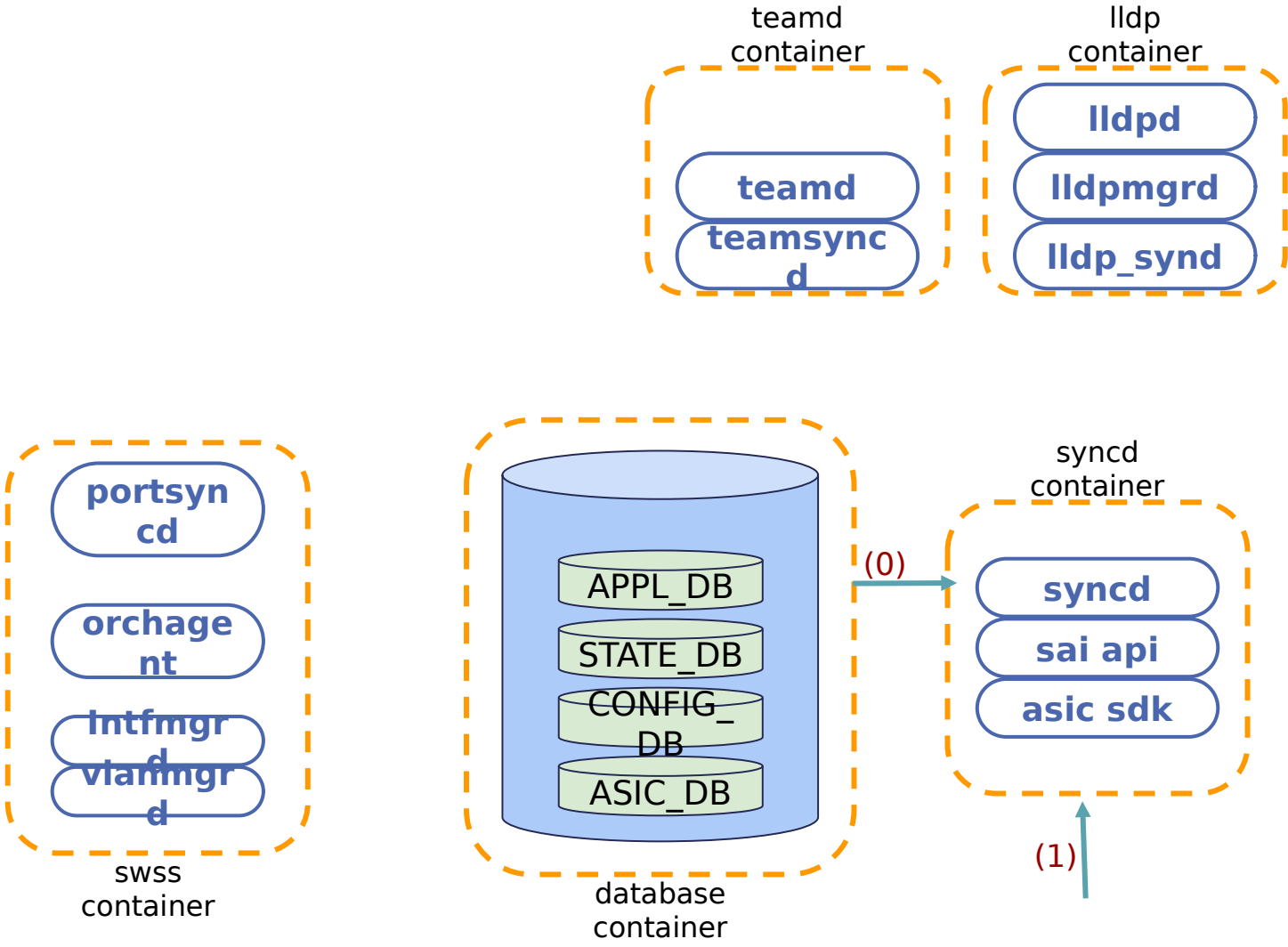


State Interactions: Port (when port goes down)

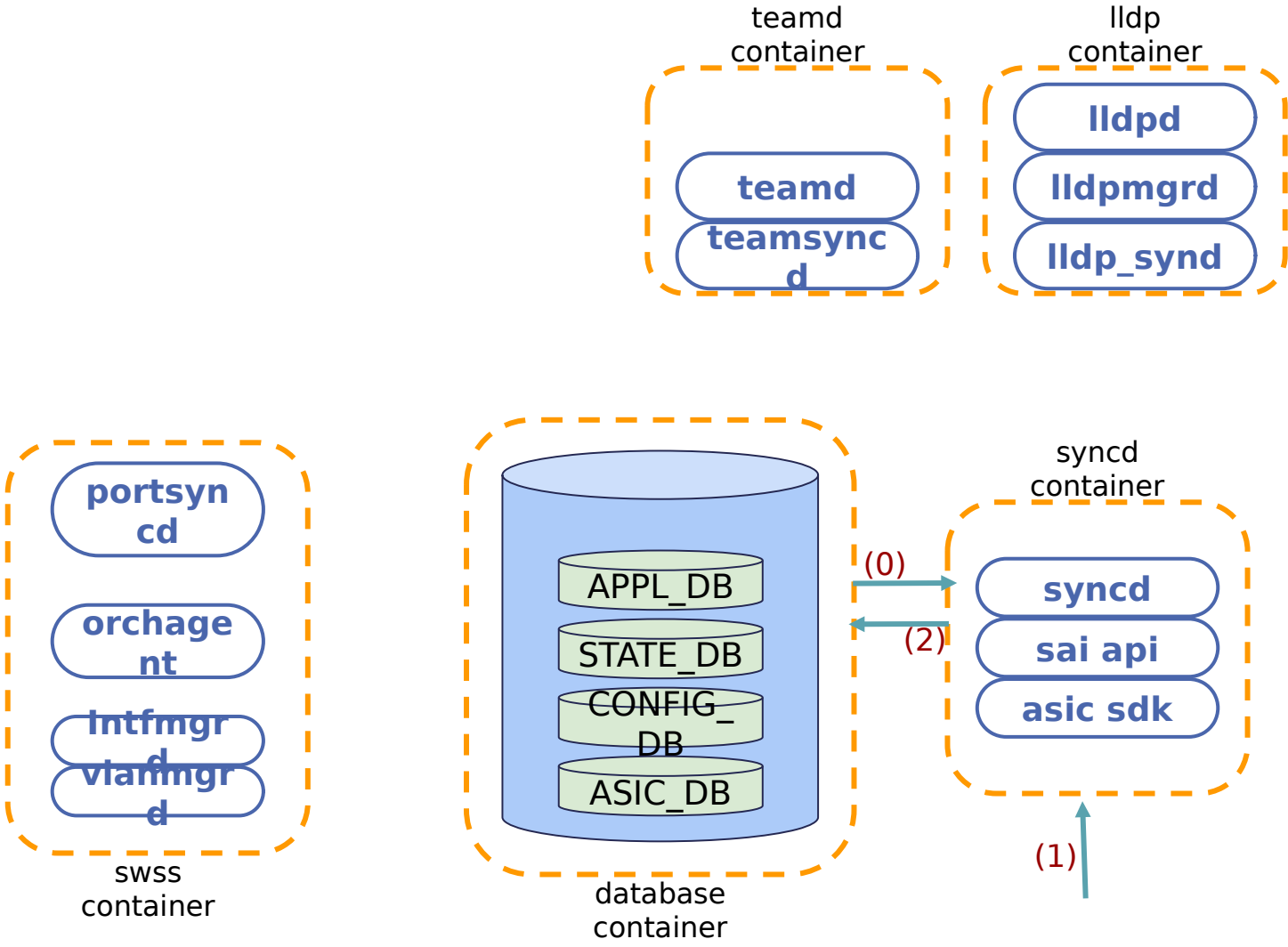
1. syncd, ASIC_DB:
include both publish and subscribe routine.
Sub -> Record hardware events/**interrupts**
Pub -> Northbound Notify



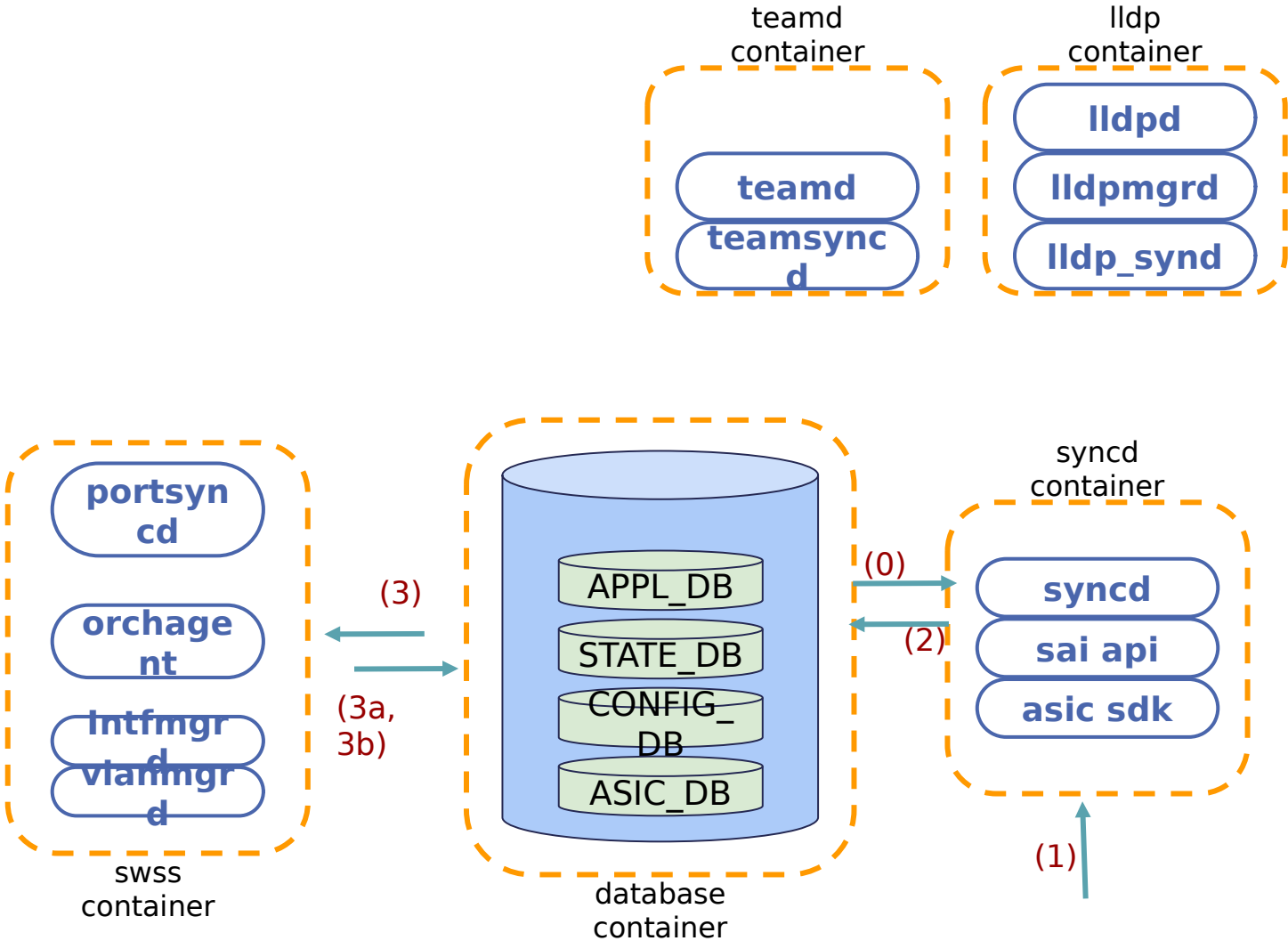
1. **syncd, ASIC_DB**: include both **publish** and **subscribe** routine
2. **device disconnect**: upon detection, by respective **optical module** event sent to **driver** and later received by **syncd**



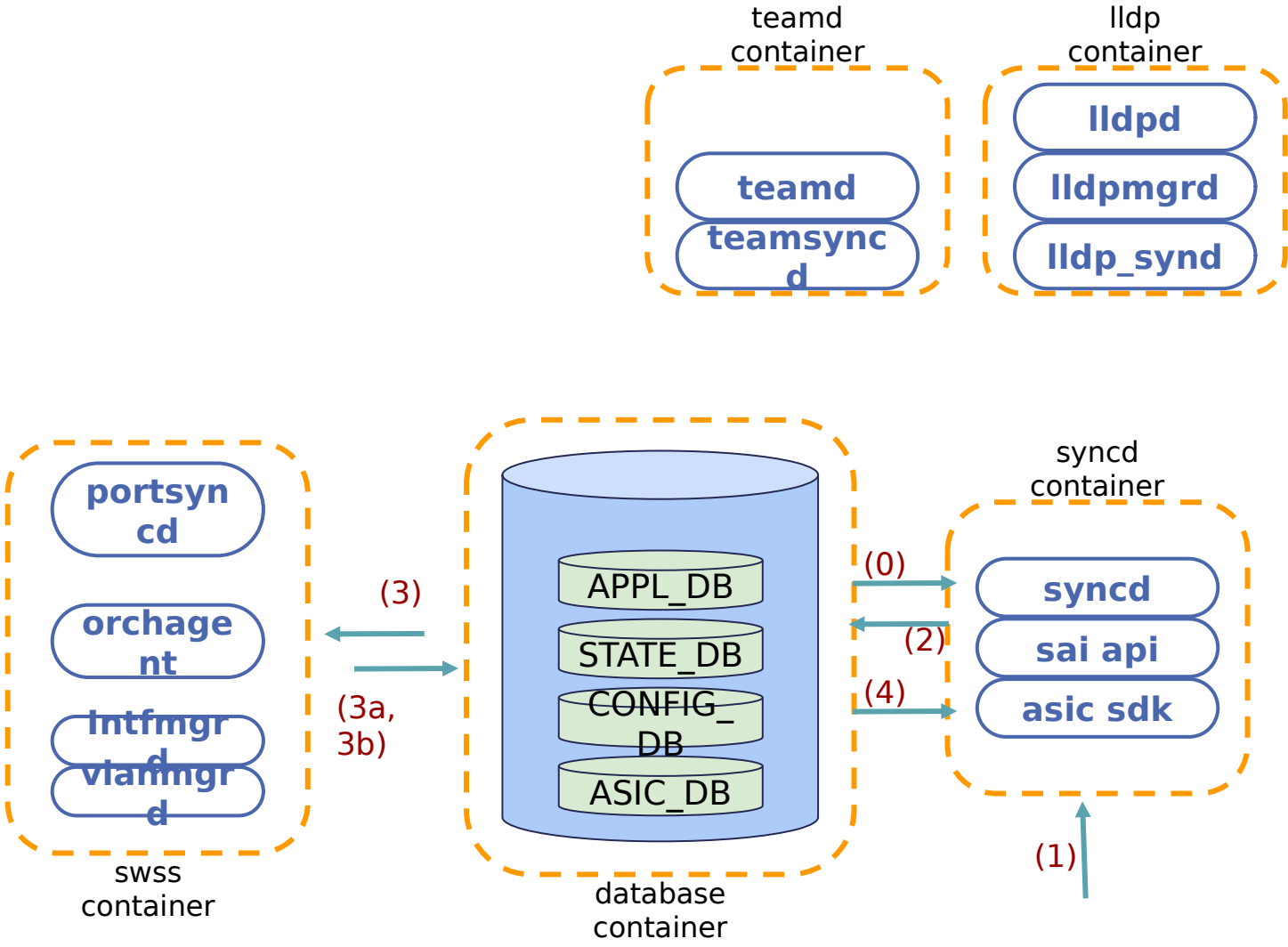
1. **syncd, ASIC_DB**: include both **publish** and **subscribe** routine
2. **device disconnect**: **hardware** event read by **driver** and later received by **syncd**
3. **invoke notify**: handler at **syncd** sends **port-down** to **ASIC_DB**



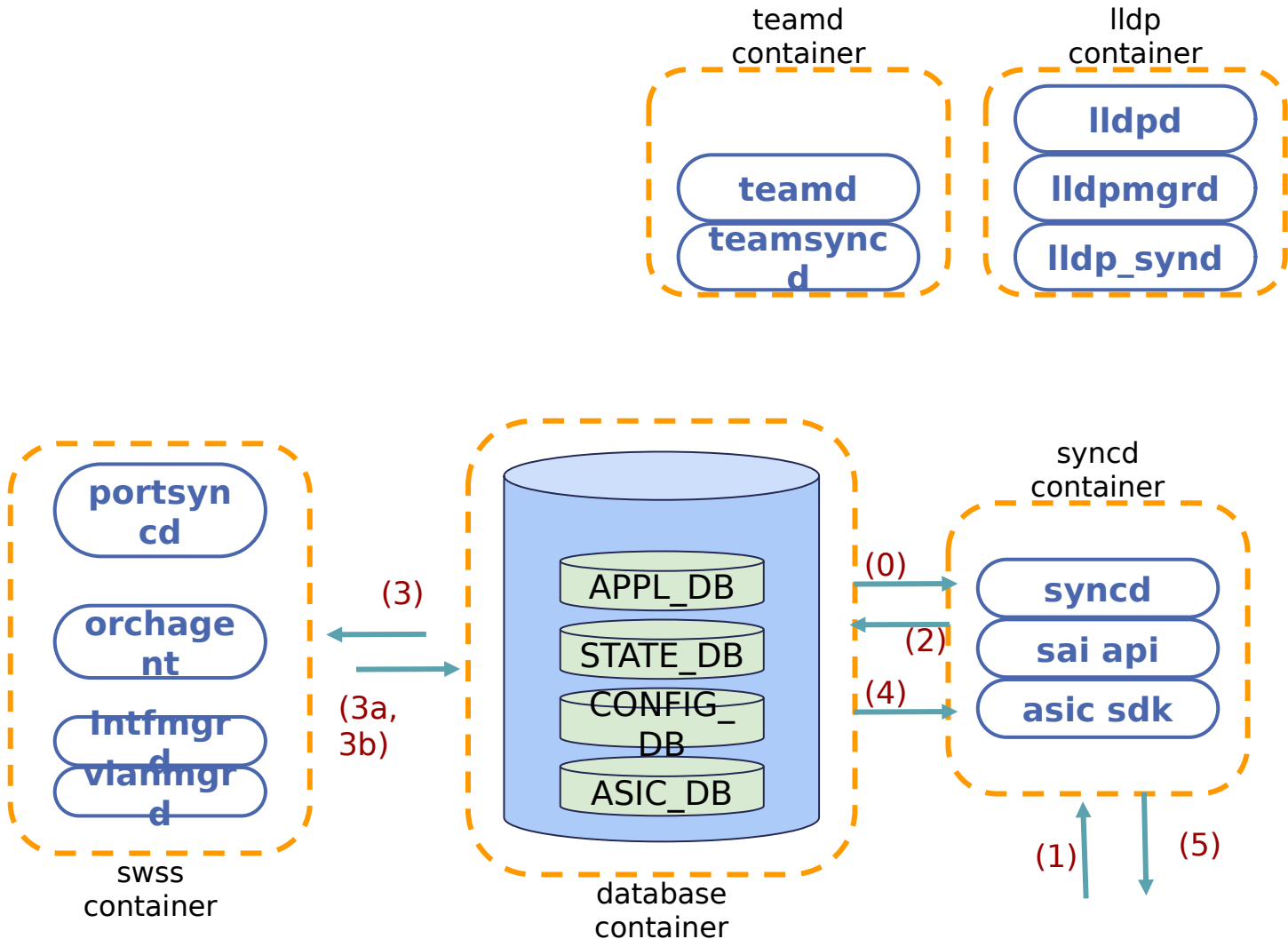
1. **syncd, ASIC_DB:** include both **publish** and **subscribe** routine
2. **device disconnect:** **hardware** event read by **driver** and later received by **syncd**
3. **invoke notify:** handler at **syncd** sends **port-down** to **ASIC_DB**
4. **ASIC_DB:** handler at **orchagent,** invokes "**port state-change"**
   a. **notify apps:** of port down through APPL_DB
   b. **notify kernel:** through sairedis APIs, ASIC_DB



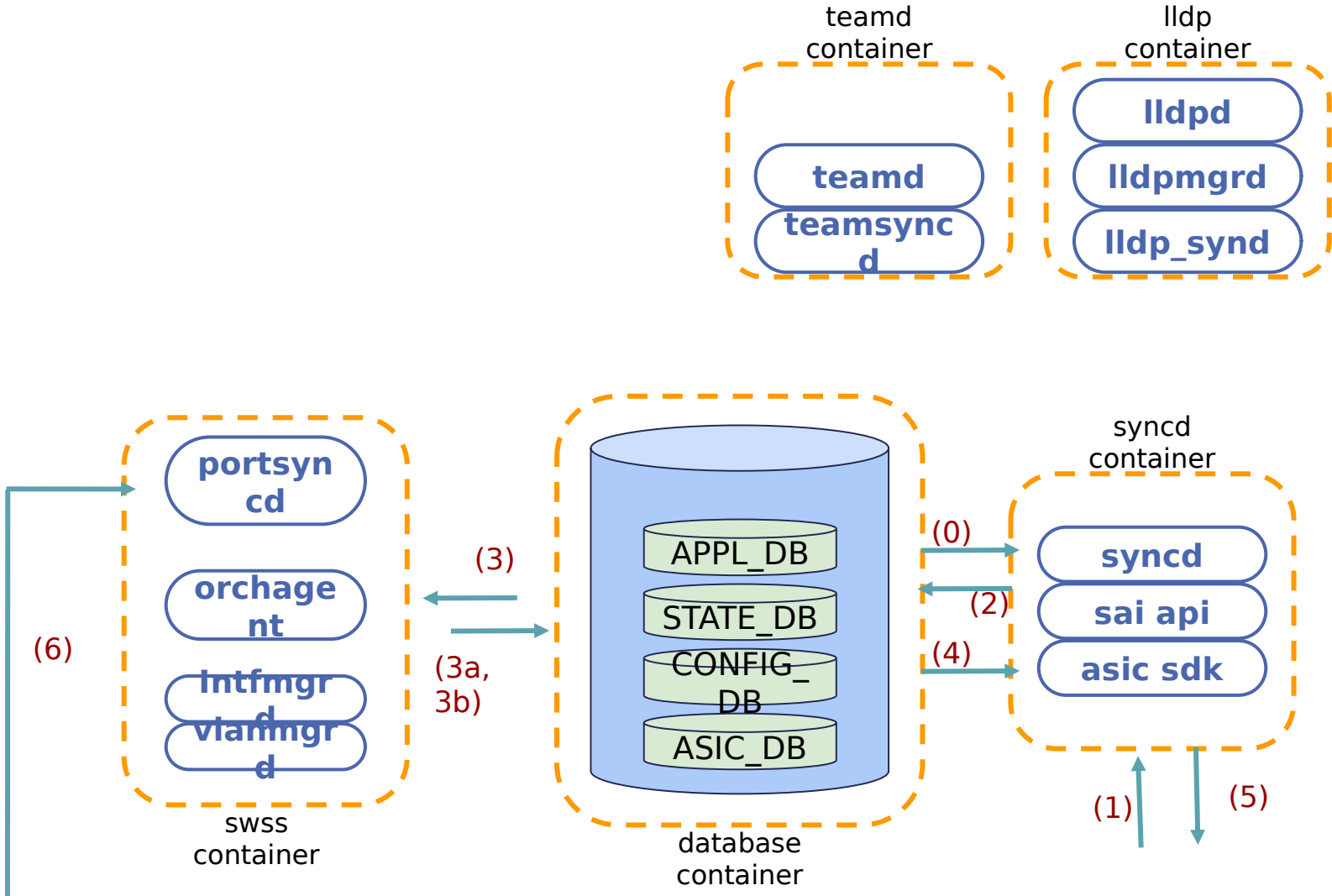
4. syncd: receives state from ASIC_DB and prepares SAI APIs to perform kernel update



4. **syncd:** receives state from ASIC_DB and prepares SAI APIs to perform kernel update
5. **kernel update:** syncd uses SAI APIs and ASIC SDK to notify state, **down**



4. **syncd:** receives state from ASIC_DB and prepares SAI APIs to perform kernel update
5. **kernel update:** syncd uses SAI APIs and ASIC SDK to notify state, **down**
6. **netlink response:** received at **portsyncd** (discarded, as **all sonic components** know the **port** is **down)**



Thank You: Questions and feedback are most welcome