ConfD And CDP API Intro

Nabil Michraf

Solutions Architect

Tail-f, Cisco

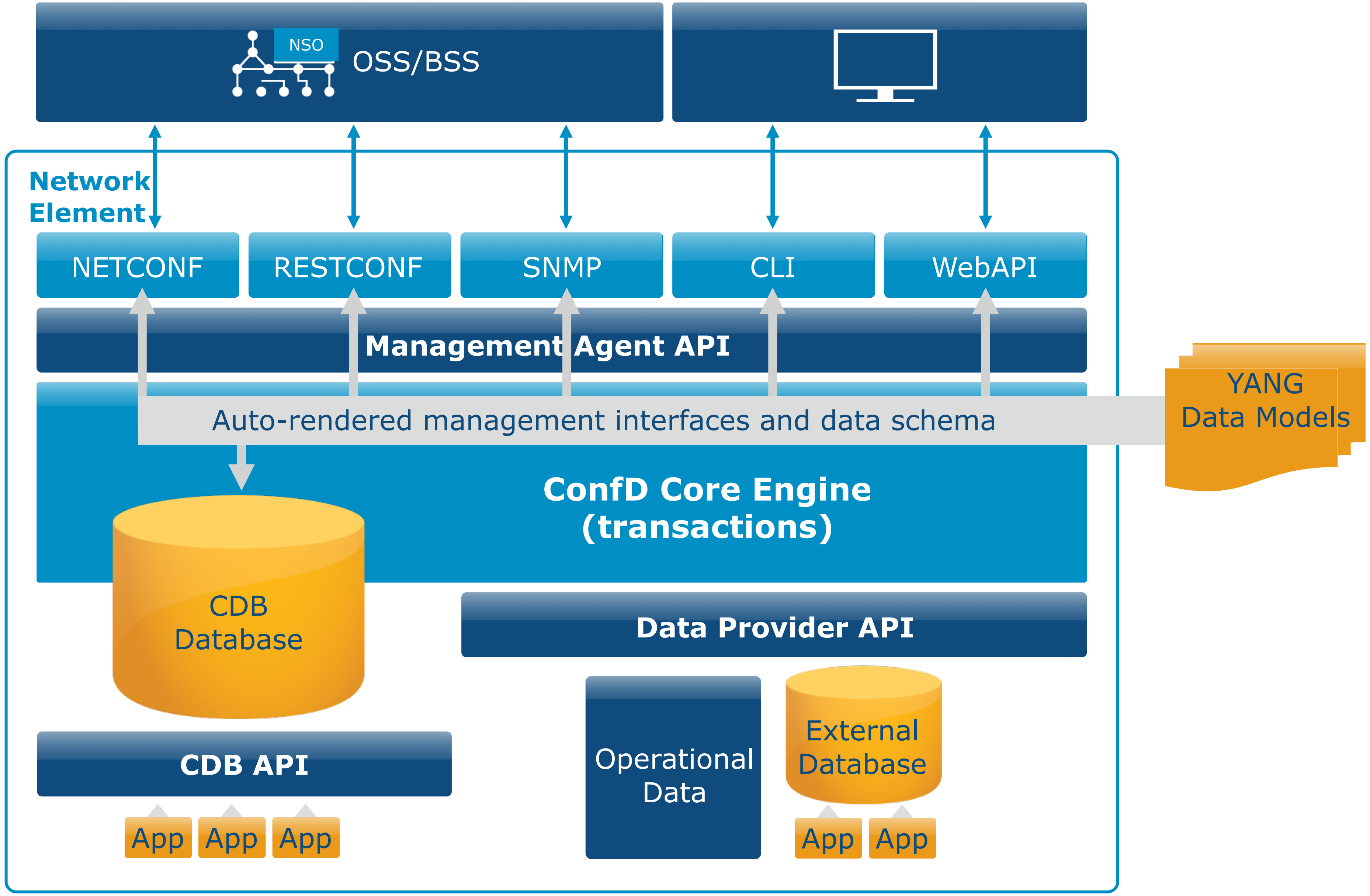
What is ConfD?

- ConfD is a data model driven framework which:
  - Provides a full set of transactional management plane services
  - Includes a variety of standards based northbound interfaces
  - Enables building programmable physical or virtual network elements
- ConfD works hand in hand with NSO to enable the programmable network
- ConfD is recognized throughout the industry as the gold standard NETCONF & YANG management software for network elements
- ConfD has been deployed by several tier-1 vendors along with many others
  - Currently over 90 programs under active support

ConfD is the Key Enabler for Programmability

- ConfD is a flexible data model driven transactional framework
- ConfD provides a variety of open standards based interfaces including NETCONF & RESTCONF
- YANG data models are used to automate runtime processing and reduce development effort
- ConfD provides an ideal solution for producing programmable network elements for SDN and NFV

ConfD Overview



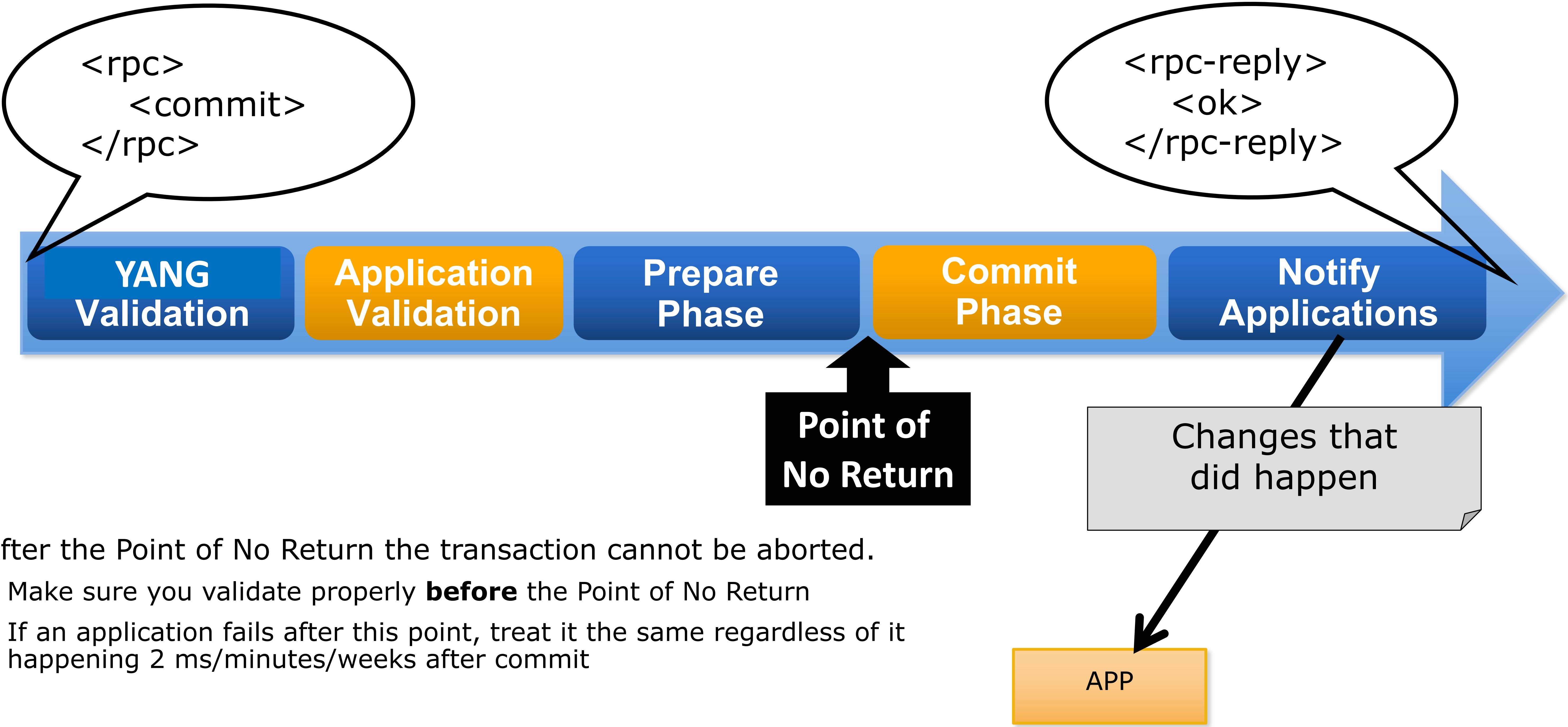
Transactions

- The ConfD core engine is a data model driven transaction engine

- The ACID properties define a transaction:
  - Atomicity
    - Transactions are indivisible, all-or-nothing
  - Consistency
    - Transactions are all-at-once; order does not matter
      - {create A, create B} and {create B, create A} are identical
  - Independence
    - Transactions do not interfere with each other
  - Durability
    - Committed data remains in the system even in case of a restart, etc

- Benefits of using transactions:
  - Increased robustness of the management plane
  - Application development and testing is easier and faster since the amount of error recovery code which needs to be written is reduced

Transaction Overview



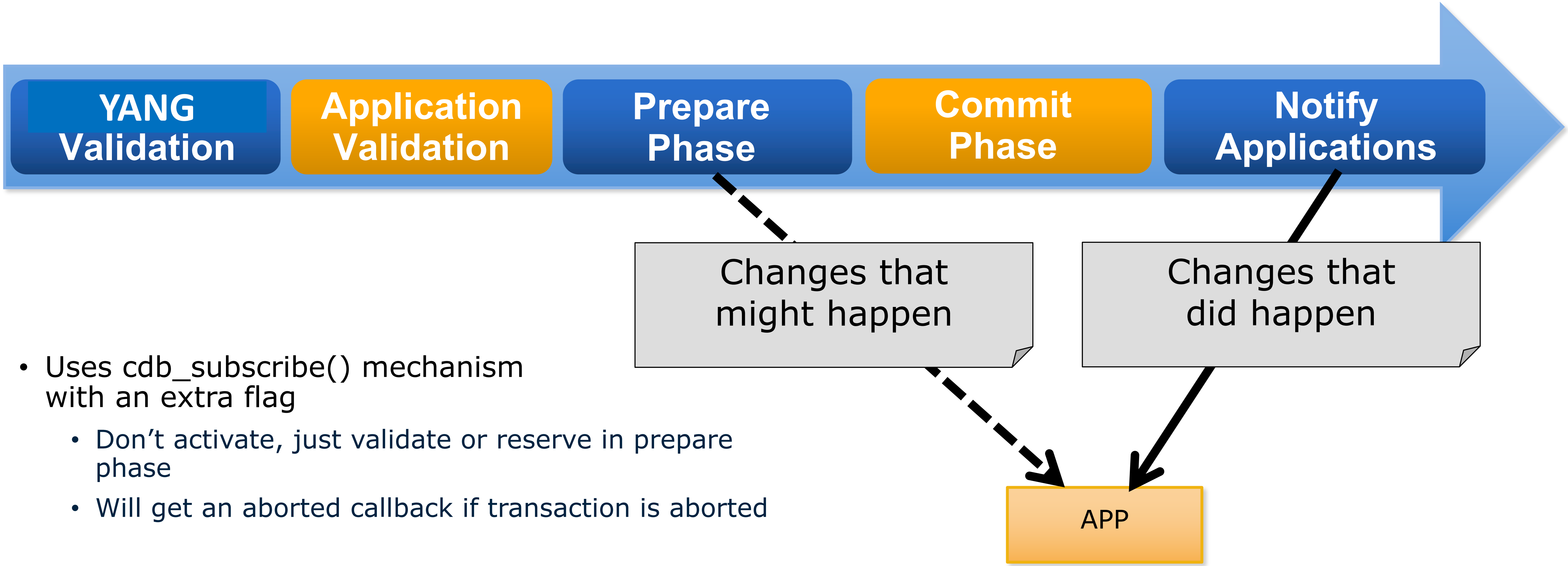
After the Point of No Return the transaction cannot be aborted.

- Make sure you validate properly **before** the Point of No Return
- If an application fails after this point, treat it the same regardless of it happening 2 ms/minutes/weeks after commit

Validators and Two Phase Subscribers

Resource reservation can be handled by a
Two Phase Subscriber

- Called during Prepare & Commit Phases



- Uses cdb_subscribe() mechanism with an extra flag
  - Don’t activate, just validate or reserve in prepare phase
  - Will get an aborted callback if transaction is aborted

What does Data Model Driven Mean?

- Data models are written in the YANG data modeling language (RFC 7950)
- The ConfD core engine loads the data model set when it starts up
- The data model is used to drive and automate processing
  - Auto-renders northbound interfaces
  - Automatically controls CDB database schema
  - Automatically performs syntactic and semantic data validation
  - Determines API interaction
    - Data items are identified based on path in the data model organization
  - + more
  - Reduces your code development
- Model once; write once; use many.
  - Instrumentation code is written in terms of the data model not the northbound interface
  - A YANG model and its common set of instrumentation can be used across all APIs and northbound interfaces

YANG Example

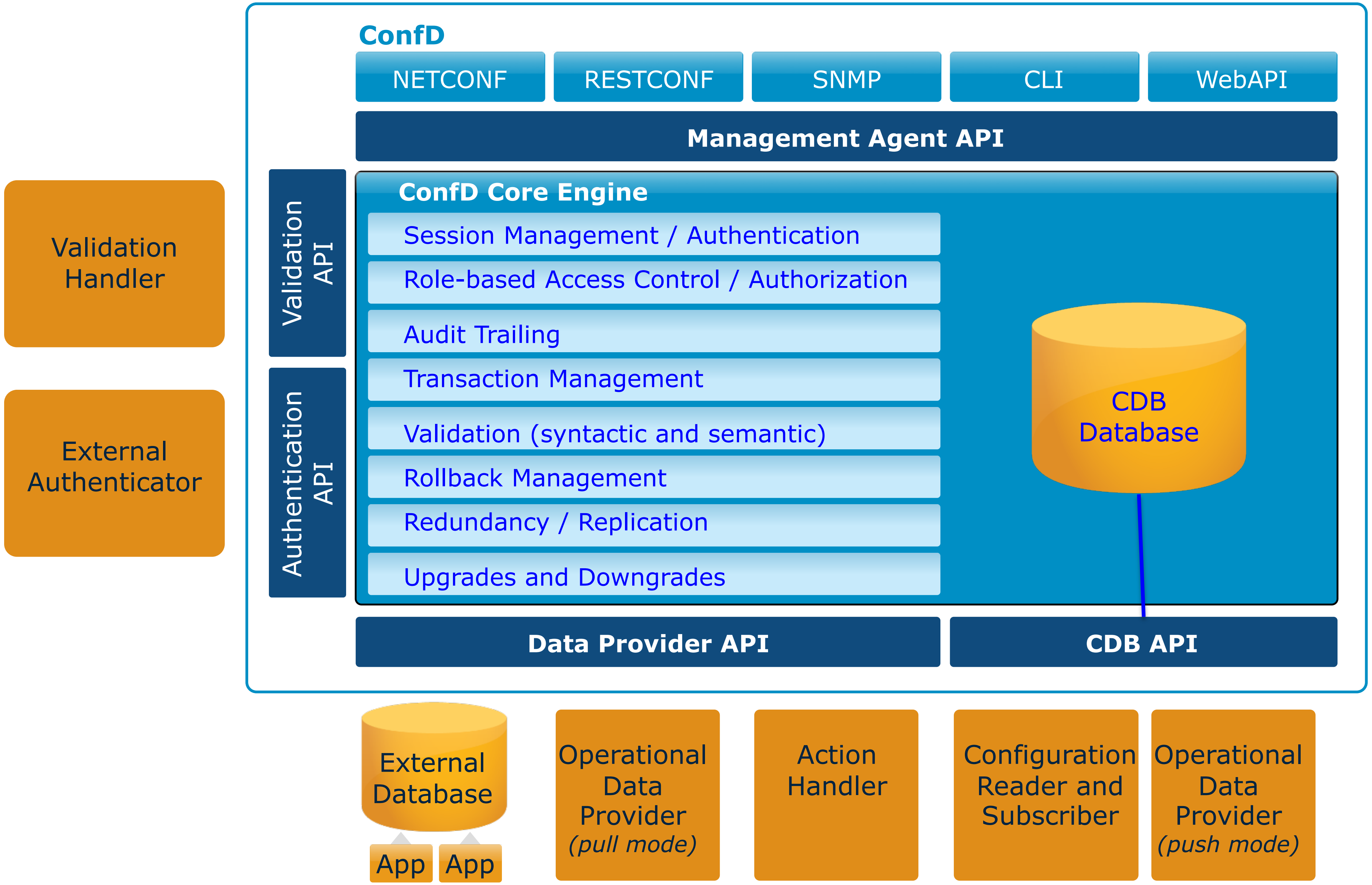
```
typedef ipv4-address {
   type string {
     pattern '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}'
          +  '([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])';
     }
 }
 container interfaces {
   list interface {
     key "name";
     unique "ip_addr";
     leaf name {
       type string;
     }
     leaf ip_addr {
       type ipv4-address;
     }
     leaf metric {
       type uint32 {
         range "1..100";
       }
     }
     must "sum(../interface/metric) <= 100";
   }
 }
```

Validation Automation

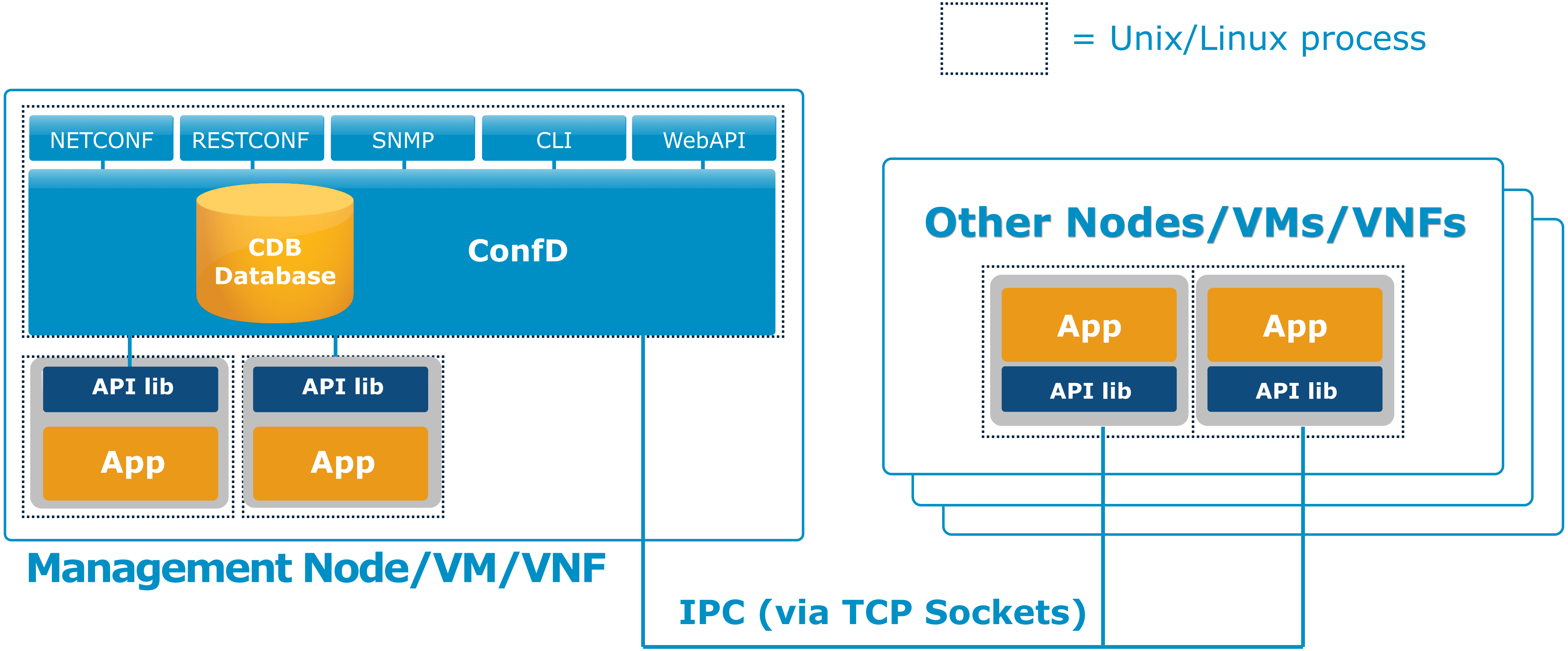
- ConfD automatically enforces all syntactic and semantics constraints from the YANG models

```
typedef ipv4-address {
   type string {
      pattern '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}'
            +  '([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])';
     }
 }
 container interfaces {
   list interface {
     key "name";
      unique "ip_addr";
     leaf name {
       type string;
     }
     leaf ip_addr {
       type ipv4-address;
     }
     leaf metric {
       type uint32 {
           range "1..100";
       }
     }
      must "sum(../interface/metric) <= 100";
   }
 }
```

ConfD Core Engine

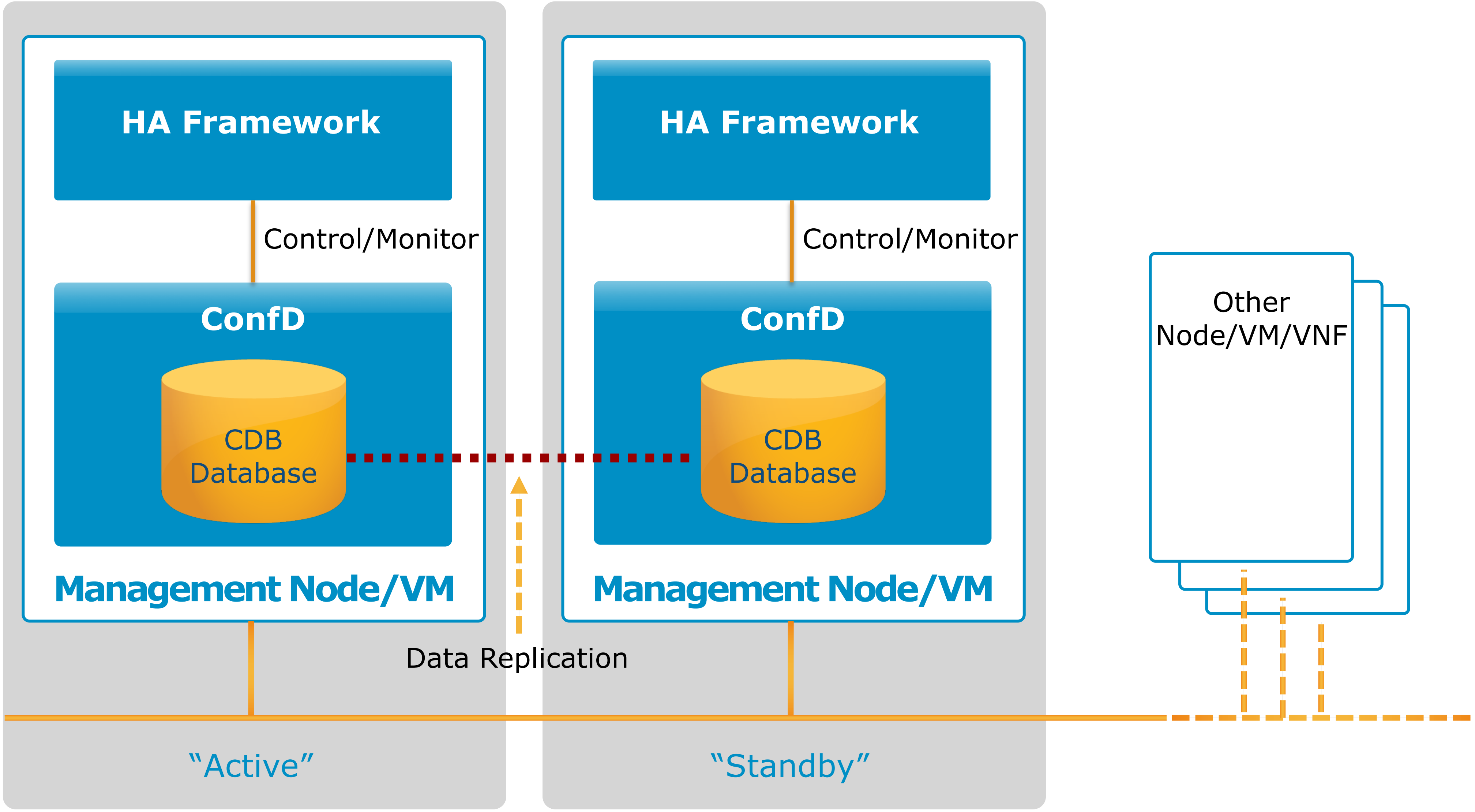


ConfD Process Architecture



API libraries available for: C, Erlang, Java, Python, and JavaScript (JSON-RPC)

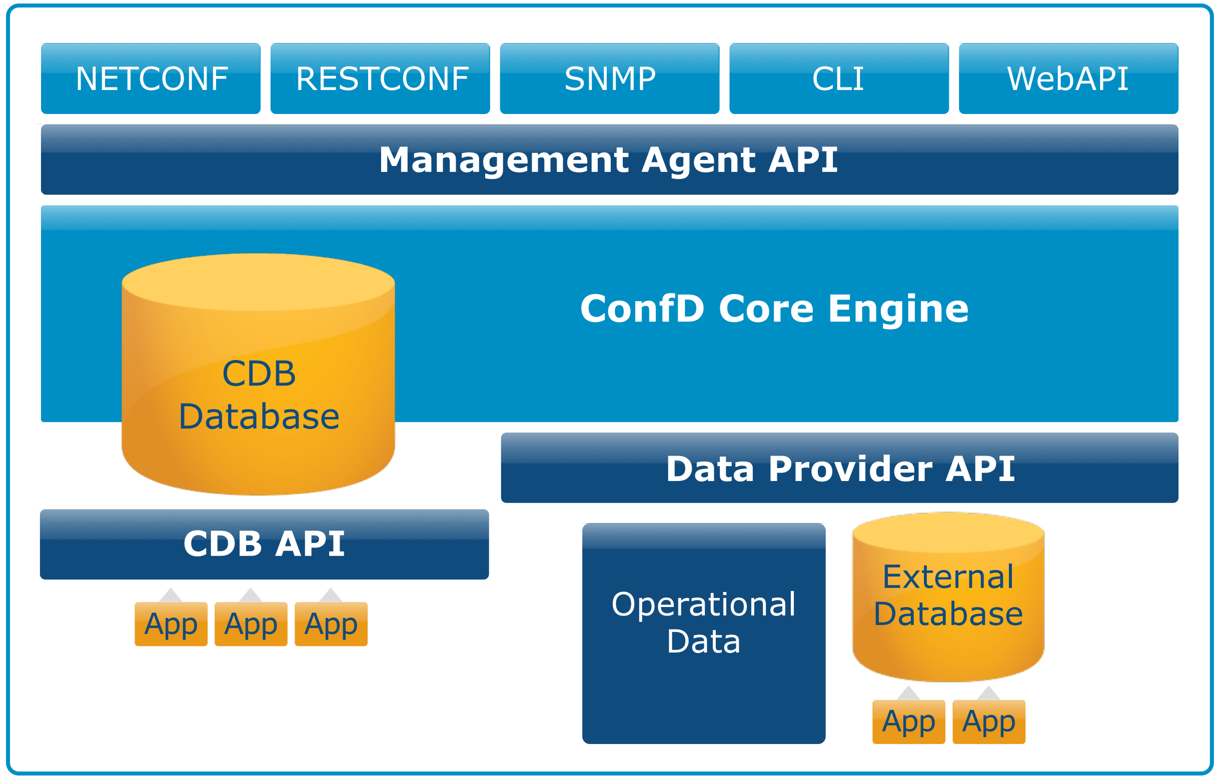
High Availability



Note: An application note and demo code is available which shows how to achieve active-active support for NFV environments using standard features.

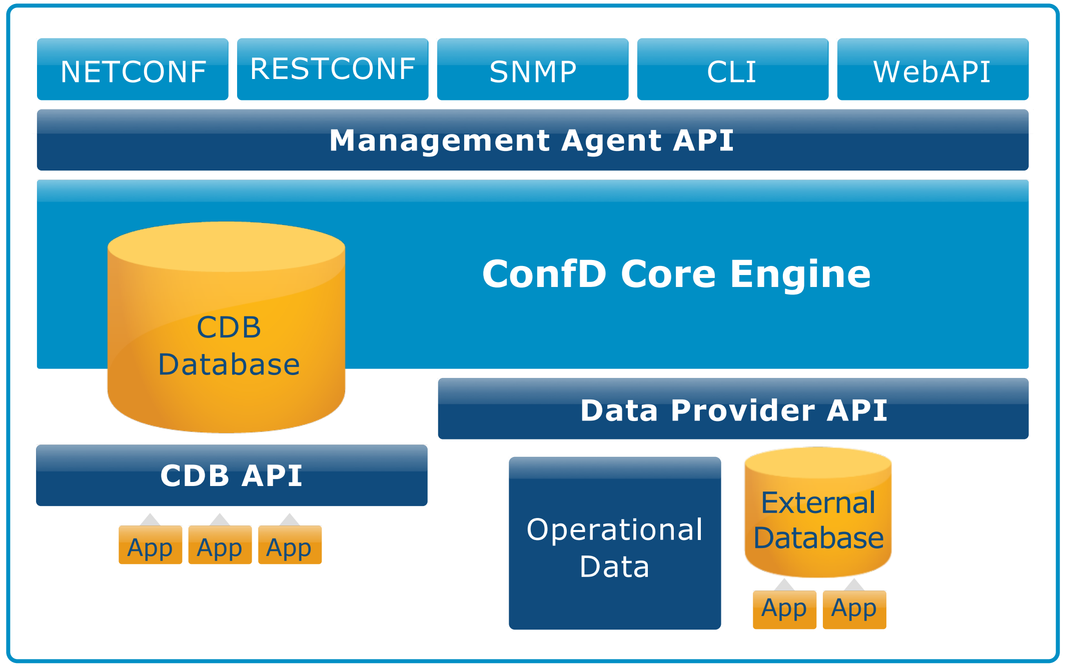
CDB Database

- Hierarchical database
  - ACID compliant
  - Fast, lightweight, fault-tolerant
  - Compact binary XML-like format
  - Memory resident with journal in persistent storage
  - Schema automatically derived from YANG
  - Multiple datastores per NETCONF standards
    - Startup, running, candidate, operational
- Supports 1:N data replication
- Supports automatic schema version up/downgrades
- Automatic loading of initial data
- Applications read data, then subscribe to relevant configuration changes
- Subscription notifications with priority level ordering



NETCONF Interface

- NETCONF is an XML RPC style protocol
  - Provides a data model driven programmable management interface (i.e. API)
- World’s gold standard NETCONF server implementation
- RFC 6241 – NETCONF 1.1
- RFC 6242 – Using NETCONF over SSH
- RFC 7950 – YANG 1.1
- RFC 5277 – NETCONF Event Notifications
- RFC 6022 – NETCONF Monitoring (inc. get-schema)
- RFC 6243 – With-defaults capability
- RFC 6470 – NETCONF Base Notifications
- RFC 6536 – NETCONF Access Control Model (NACM)
- RFC 7895 – YANG module Library
- + more
- Support included for various IETF standard YANG data models



NETCONF OPERATIONS

<get-config>
<edit-config>
<delete-config>
<lock>
<unlock>
<get>
<close-session>
<kill-session>
<commit>
<discard-changes>
<partial-lock>
<partial-unlock>
<get-schema>

NETCONF CAPABILITIES

:writeable-running
:candidate
:confirmed-commit
:rollback-on-error
:validate
:startup
:URL
:XPath

NETCONF Example Configuration Sequence

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.1” message-id="5">
  <edit-config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <target>
      <candidate/>
    </target>
    <test-option>test-then-set</test-option>
    <error-option>rollback-on-error</error-option>
    <config>
      <interface xmlns=”urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <name>eth1</name>
        <ipv4-address>192.168.5.10</ipv4-address>
        <macaddr>aa:bb:cc:dd:ee:ff</macaddr>
      </interface>
    </config>
  </edit-config>
</rpc>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.1"
      message-id="5">
  <ok/>
</rpc-reply>
```

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.1” message-id="6">
  <validate>
    <source>
      <candidate/>
    </source>
  </validate>
</rpc>
```

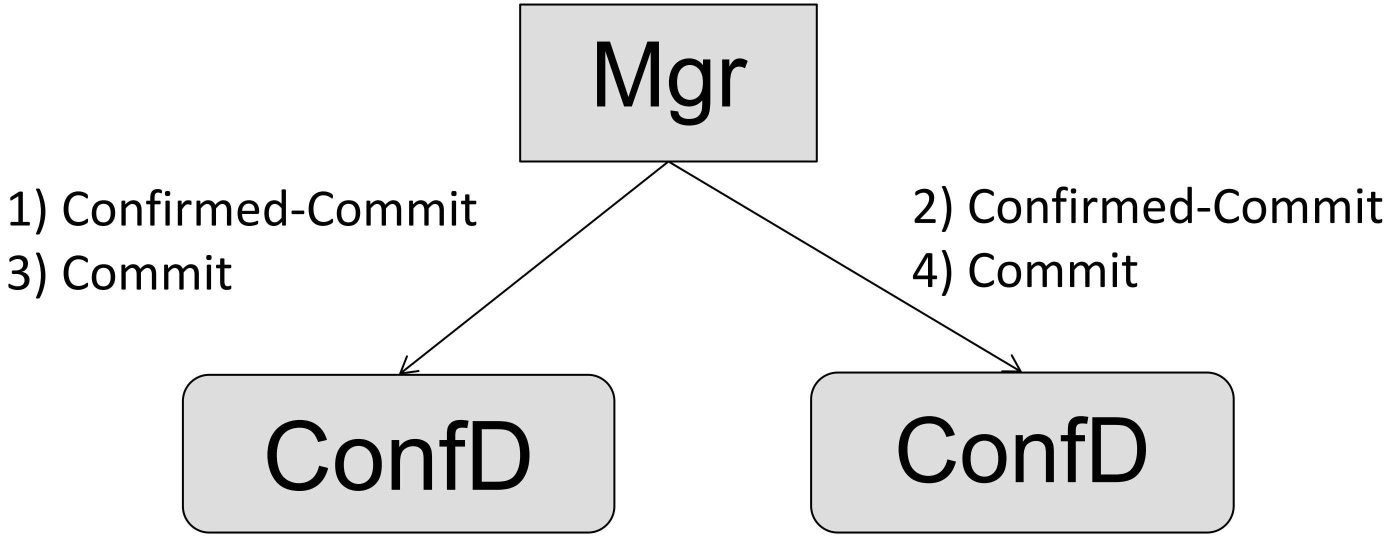
```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.1"
       message-id="6">
  <ok/>
</rpc-reply>
```

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.1” message-id="7">
  <commit>
    <confirmed/>
  </commit>
</rpc>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.1"
       message-id=“7">
  <ok/>
</rpc-reply>
```

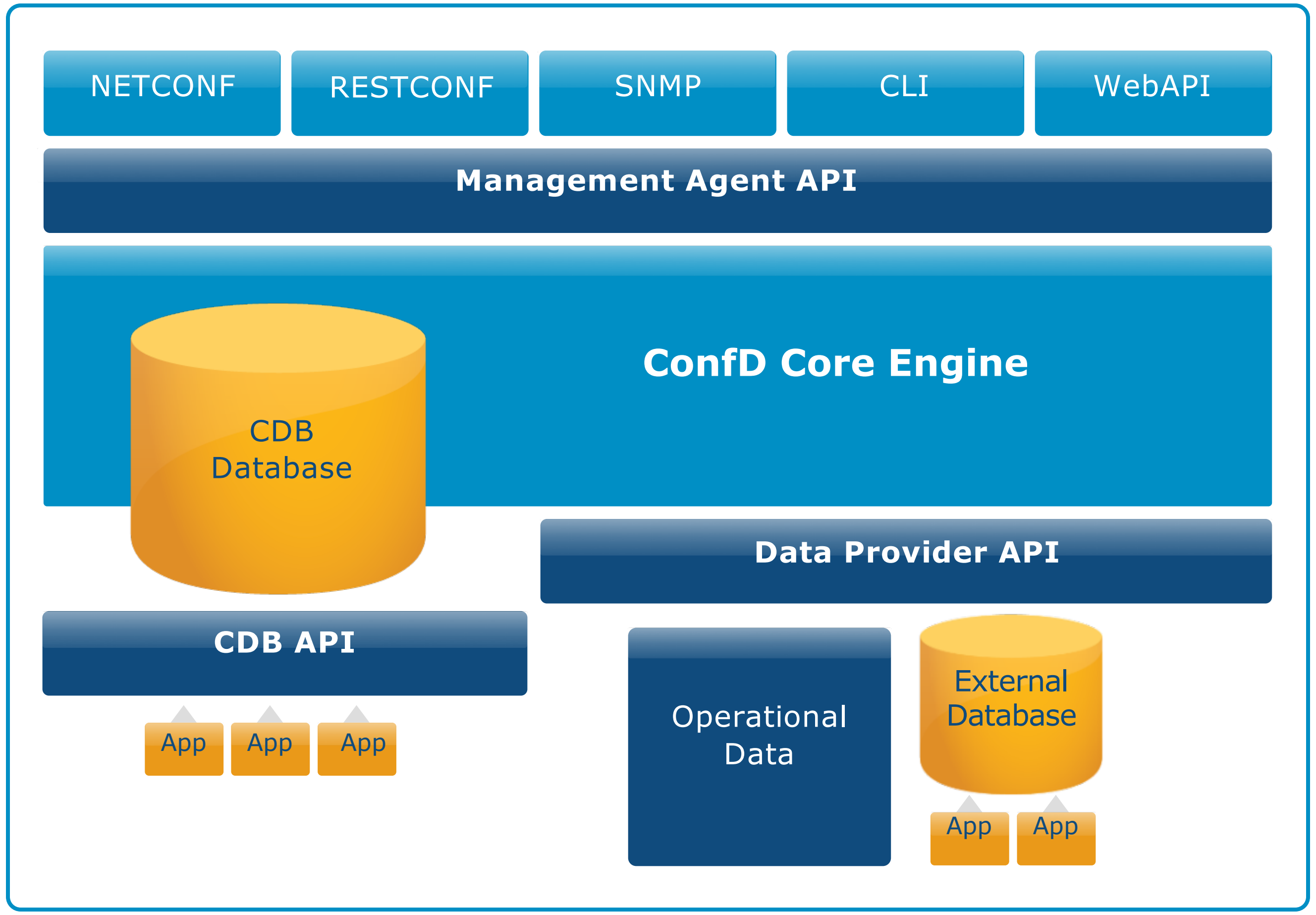
NETCONF <Confirmed-Commit> – Key Network Programmability Feature

- Key feature for implementing network-wide transactions
- Confirmed-Commit solves many tricky failure scenarios
  - ConfD rolls-back automatically if no commit within timeout or SSH closes
- If a network element does not support confirmed-commit
  - Manager has to keep the the revert diff and send the precise undo information to the device
- If a ConfD application rejects a config change - the whole transaction is aborted
- For ConfD to support Confirmed-Commit, northbound candidate is required.



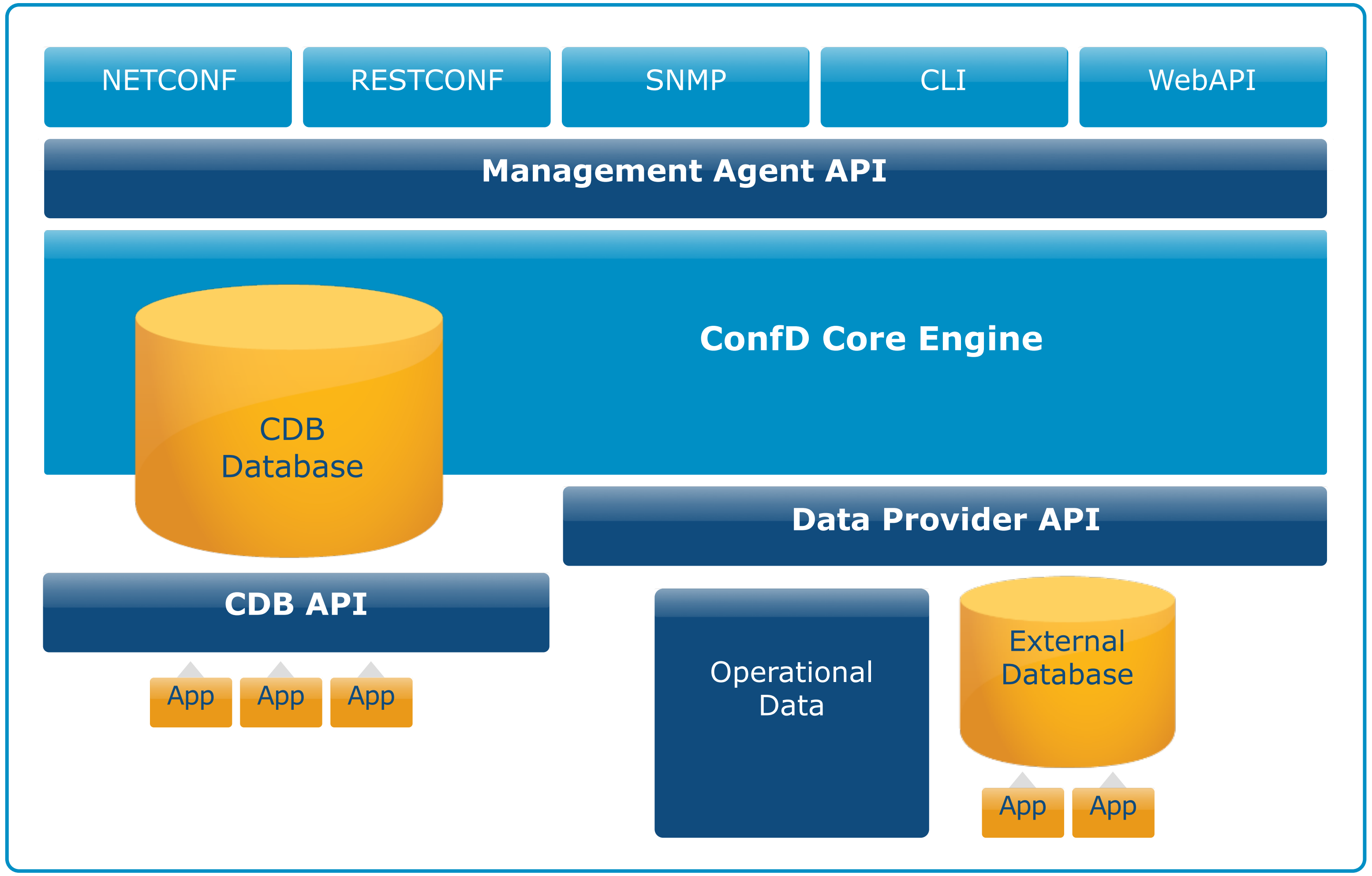
RESTCONF

- RESTCONF is defined in RFC8040
- RESTCONF standardizes how to use REST techniques to manipulate the resources described by YANG models
- RESTCONF uses the verbs of the HTTP transport layer to manipulate resources:
  - GET : get resources
    - Selectors : shallow, deep
  - PUT : replace existing resource
  - POST : create resource
  - DELETE : delete resource
  - PATCH (RFC5789) : modify existing resource
  - HEAD, OPTIONS
- Stateless, client-server
- XML or JSON as data containers
- Links to available data stores and operations



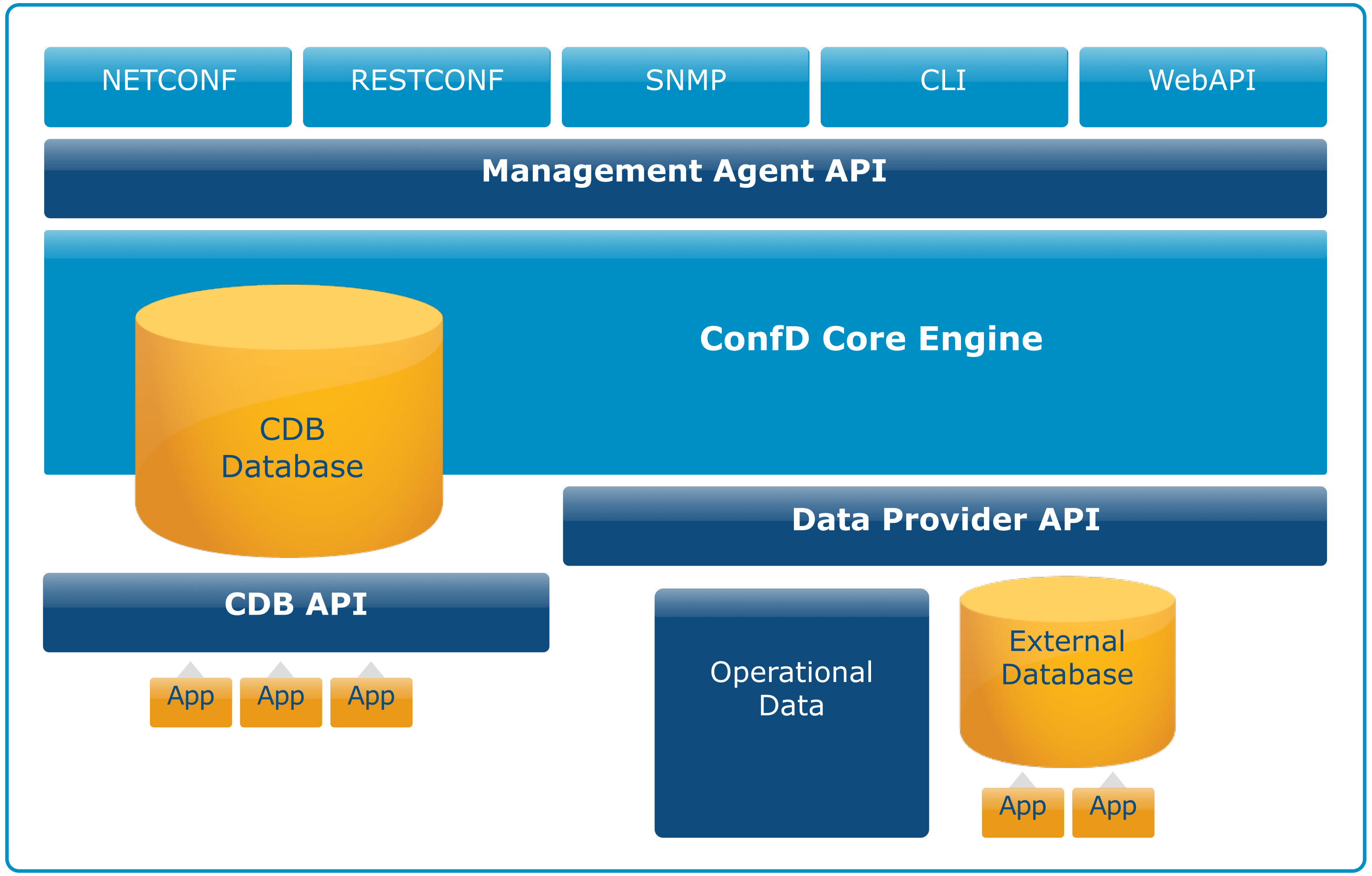
SNMP Agent

- Full featured SNMP interface
  - Any combination of v1, v2c, v3
  - Supports all operations
  - Send SNMP notifications (traps)
  - MIBs implemented by ConfD include:
    - RFC-3411 (SNMP-FRAMEWORK-MIB)
    - RFC-3414 (USM)
    - RFC-3415 (VACM)
    - RFC-3418 (SNMPv2-MIB)
    - + more
- MIB to YANG translator
- YANG to MIB translator



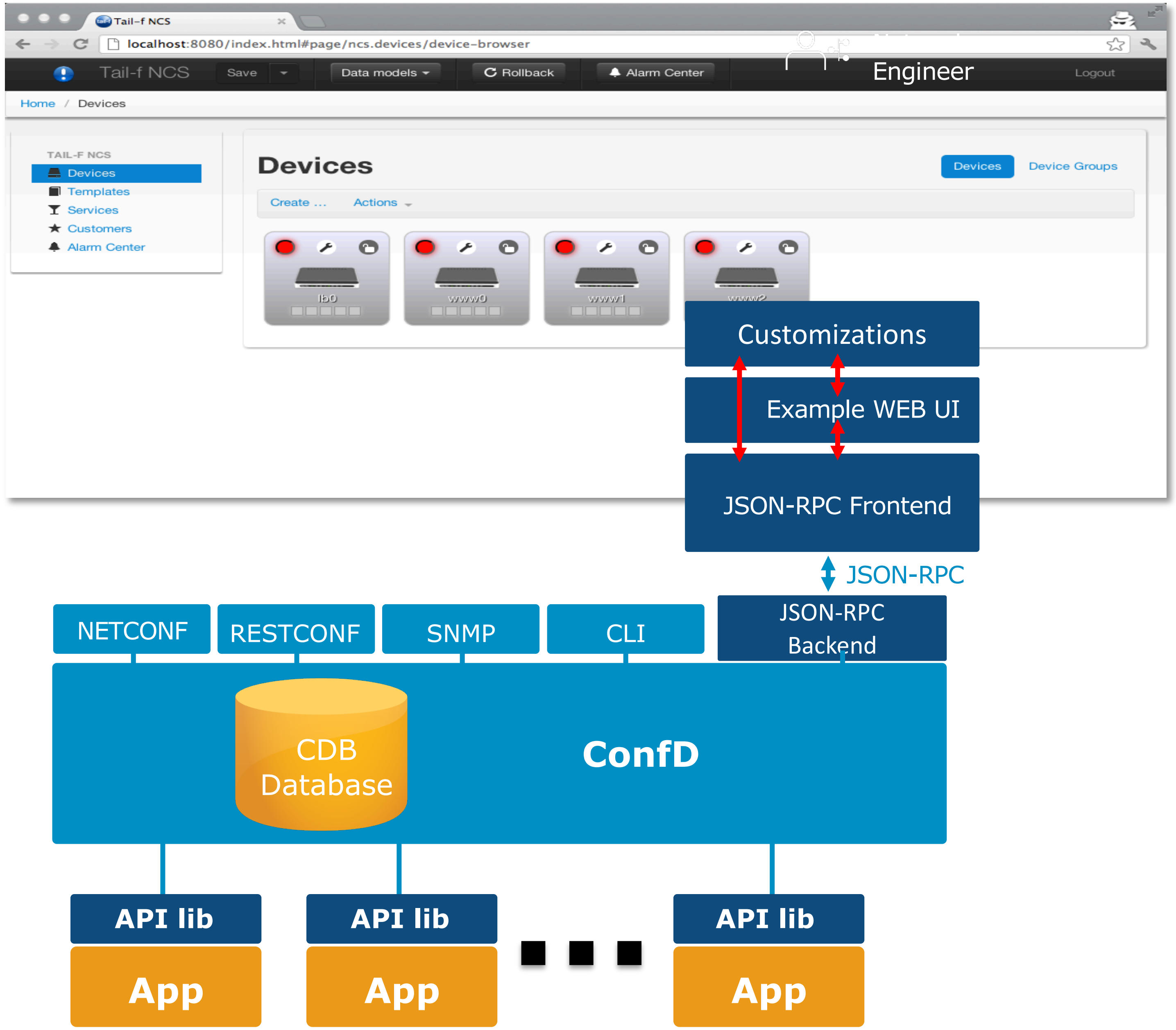
Command Line Interface (CLI)

- Auto rendering of three CLI styles:
  - Cisco – XR style
  - Cisco – legacy IOS style
  - Juniper – JUNOS style
- Rich editing with tab-completion for commands, static elements, and dynamic instances
- History, hints, help, etc.
- Extremely customizable
  - Customize auto-rendering
  - Configurable options
  - Add new commands
- Use with internal SSH server or external SSH server; e.g. OpenSSH



Web

- Integrated Web Server
- JSON RPC API
  - Access data model schema information
    - get-schema
  - Access data
  - Run transactions and rollbacks
  - Do validation
  - Execute actions
- Customer choice of toolsets and frameworks allows for preservation of existing Web content
- Example of how to implement an auto-rendered Web UI provided



Why ConfD

Make your customer

Make your device

- Manageable
- Programmable
- Standards compliant

- NETCONF, RESTCONF, CLI, SNMP, Web
- Transactions and rollbacks
- Validations
- Configuration and monitoring
- No feature lag

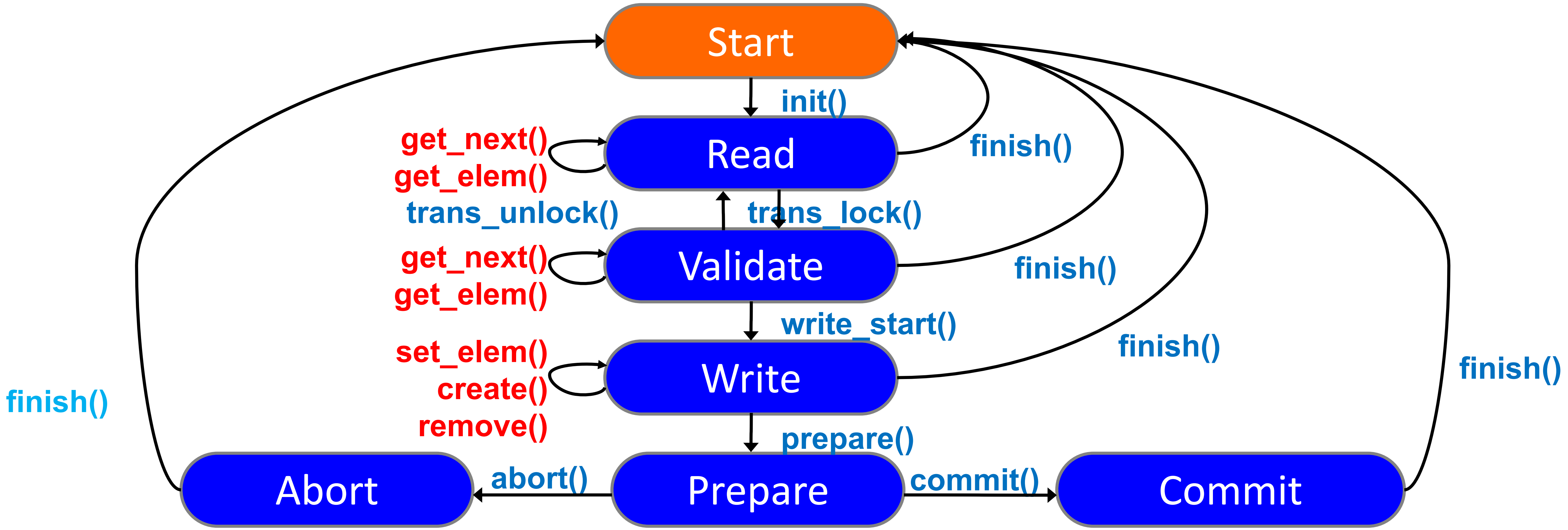
Save time

- Auto-render management interfaces
- Data model driven
- Iterative development model
  - Perfect for agile development

- Core components
- Embedded database
- Powerful, easy-to-use APIs
- Market-proven
  - Shipping in 100+ products in operator networks worldwide

tail-f

Data Provider API



A transaction can be viewed as a conceptual state machine

- Phases of a transaction correspond to states
- API transaction callback function invocations are state transitions

CDB API

Database API.

With this API, applications read configuration data from the database and subscribe to be notified about configuration changes. Applications may also write operational data into the operational datastore in CDB.

Settings in ConfD

```
<cdb>
        <enabled>true</enabled>                      Yes, you can actually run without CDB!
        <dbDir>/var/confd/cdb</dbDir>                Where A.cdb and C.cdb is stored
        <persistent>true</persistent>                False = memory only CDB(!)
        <initPath>                                   Where to look for .xml files
                <dir>/var/confd/cdb-init</dir>       during startup
                <dir>/opt/local/app/init</dir>
        </initPath>
        <clientTimeout>infinity</clientTimeout>
        <replication>sync</replication>              How HA replication is done
</cdb>

…
```

ConfD clients

- Subscriber

An application using the CDB API that has requested to be notified when an operator modifies parts of the configuration that the application is interested in, e.g. set hostname or add an entry to the interface table. Subscribers only participate in the commit phase of the transaction.

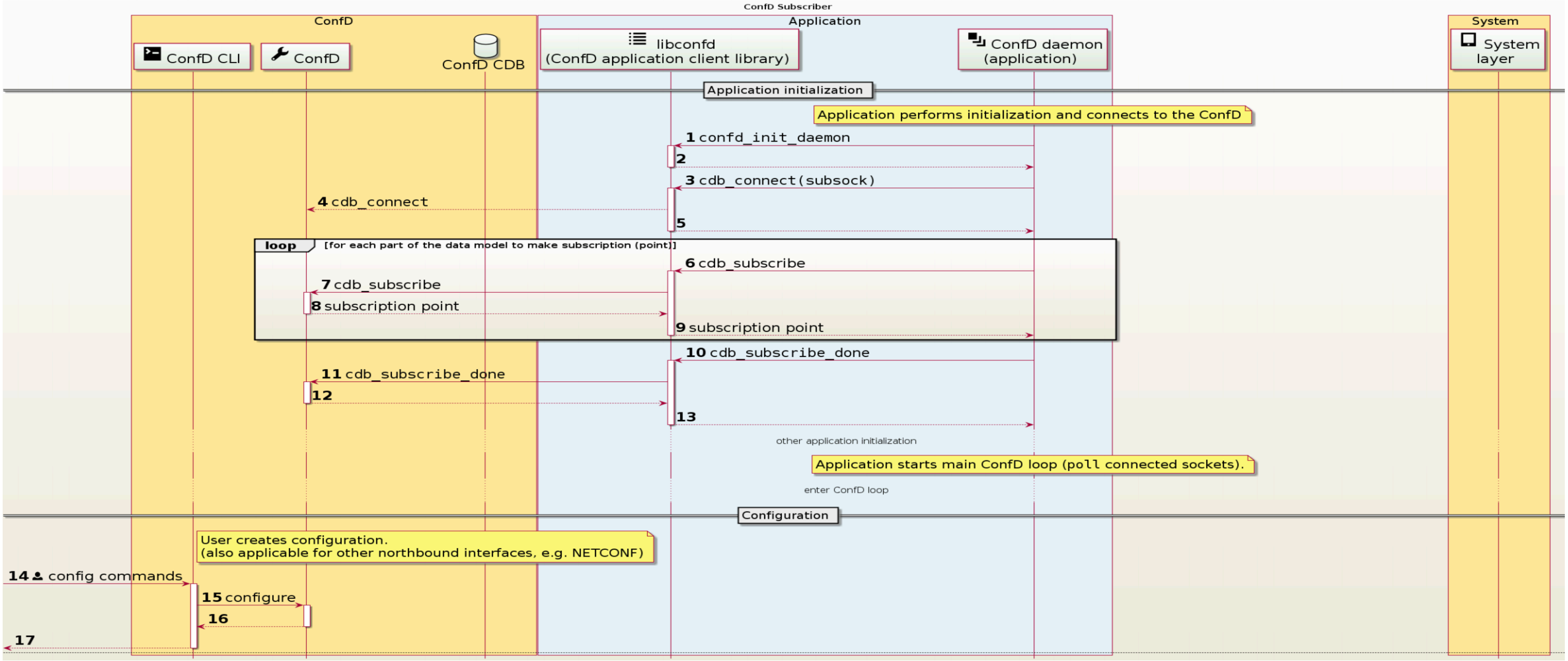
- Two-phase Subscriber

A subscriber application It can thereby reject upcoming configuration changes. that participates in both the prepare and commit phase of a transaction using the CDB API.

CDB Subscriber Flow

1. Connect to ConfD.
2. Connect a read soket.
3. Connect a subscriber socket.
4. Read Data.
5. Register for change notifications.
6. Process notifications and read data.

Subscriber Initialization



ConfD Subscriber

ConfD

ConfD CLI

ConfD

ConfD CDB

Application

libconfd
(ConfD application client library)

ConfD daemon
(application)

System

System
layer

Application initialization

Application performs initialization and connects to the ConfD

1 confd_init_daemon

2

3 cdb_connect(subsock)

4 cdb_connect

5

loop [for each part of the data model to make subscription (point)]

6 cdb_subscribe

7 cdb_subscribe

8 subscription point

9 subscription point

10 cdb_subscribe_done

11 cdb_subscribe_done

12

13

other application initialization

Application starts main ConfD loop (poll connected sockets).

enter ConfD loop

Configuration

User creates configuration.
(also applicable for other northbound interfaces, e.g. NETCONF)

14 config commands

15 configure

16

17

Processing of one subscription point

- Processing of the subscription point in the one phase subscriber or in the COMMIT phase of the two phase subscriber.

- Two approaches:

cdb_diff_iterate(): invokes iter callback function with operation type (e.g. created, modified, etc.) and with the new and the old value (confd_value_t)

cdb_get_modifications(): returns changes in from of confd_tag_value_t array.

Processing of one subscription point with cdb_diff_iterate

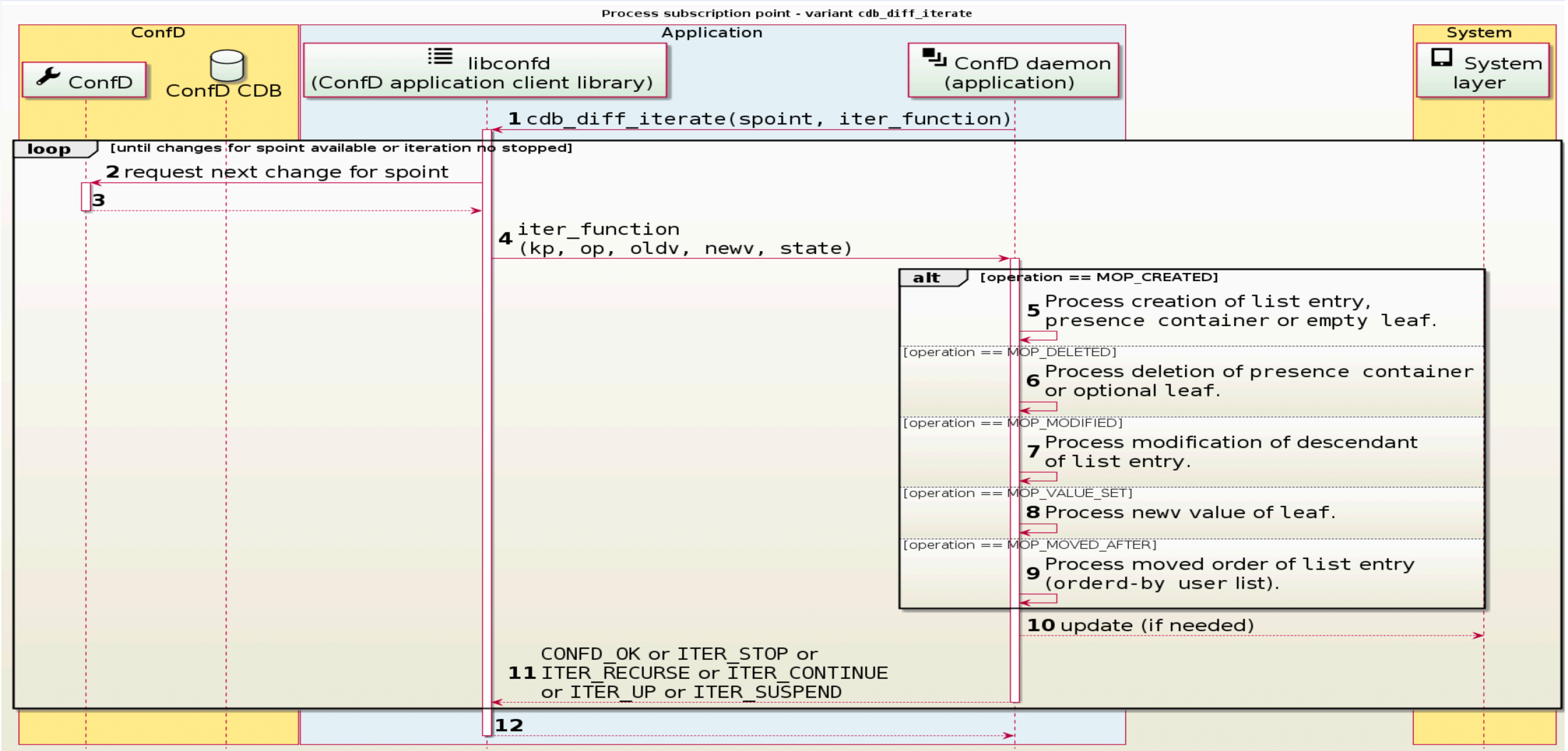
- Application invokes cdb_diff_iterate with subscription point and pointer to the iter function.

- Changes for the given subscription point are requested from ConfD.

- For each change, the iter function is invoked with the keypath, operation type, new and old value.

- After the iter function processes the change, it returns value indicating if processing of changes should continue or should stop.

Sequence



Process subscription point - variant cdb_diff_iterate

ConfD

ConfD

ConfD CDB

Application

libconfd
(ConfD application client library)

ConfD daemon
(application)

System

System
layer

1 cdb_diff_iterate(spoint, iter_function)

loop [until changes for spoint available or iteration no stopped]

2 request next change for spoint

3

4 iter_function
(kp, op, oldv, newv, state)

alt [operation == MOP_CREATED]

5 Process creation of list entry,
presence container or empty leaf.

[operation == MOP_DELETED]

6 Process deletion of presence container
or optional leaf.

[operation == MOP_MODIFIED]

7 Process modification of descendant
of list entry.

[operation == MOP_VALUE_SET]

8 Process newv value of leaf.

[operation == MOP_MOVED_AFTER]

9 Process moved order of list entry
(orderd-by user list).

10 update (if needed)

11 CONFD_OK or ITER_STOP or
ITER_RECURSE or ITER_CONTINUE
or ITER_UP or ITER_SUSPEND

12

Processing of one subscription point with cdb_get_modifications

- Application invokes cdb_get_modifications with subscription point as a parameter.

- The cdb_get_modifications returns array of confd_tag_value_t elements, representing the changes.

- The application iterates through the array of returned elements, the element type indicates operation done.

- Once element value is processed (not needed), the confd_free_value() is called to release memory (possibly) allocated for ConfD value.

- Finally, the free on whole array is called (allocated by cdb_get_modifications).