NSO Performance

How to make sure your NSO system meets your performance requirements

Agenda

- Introduction, the big picture
- Deep dive, the NSO transactional model
- NSO Deployment Models
  - Commit Queue
  - Layered Service Architecture, LSA Cluster
  - Device Cluster
  - High Availability Cluster
  - Host Machines
- NSO System Configuration
- NED and Devices Configuration
- Developing the service code

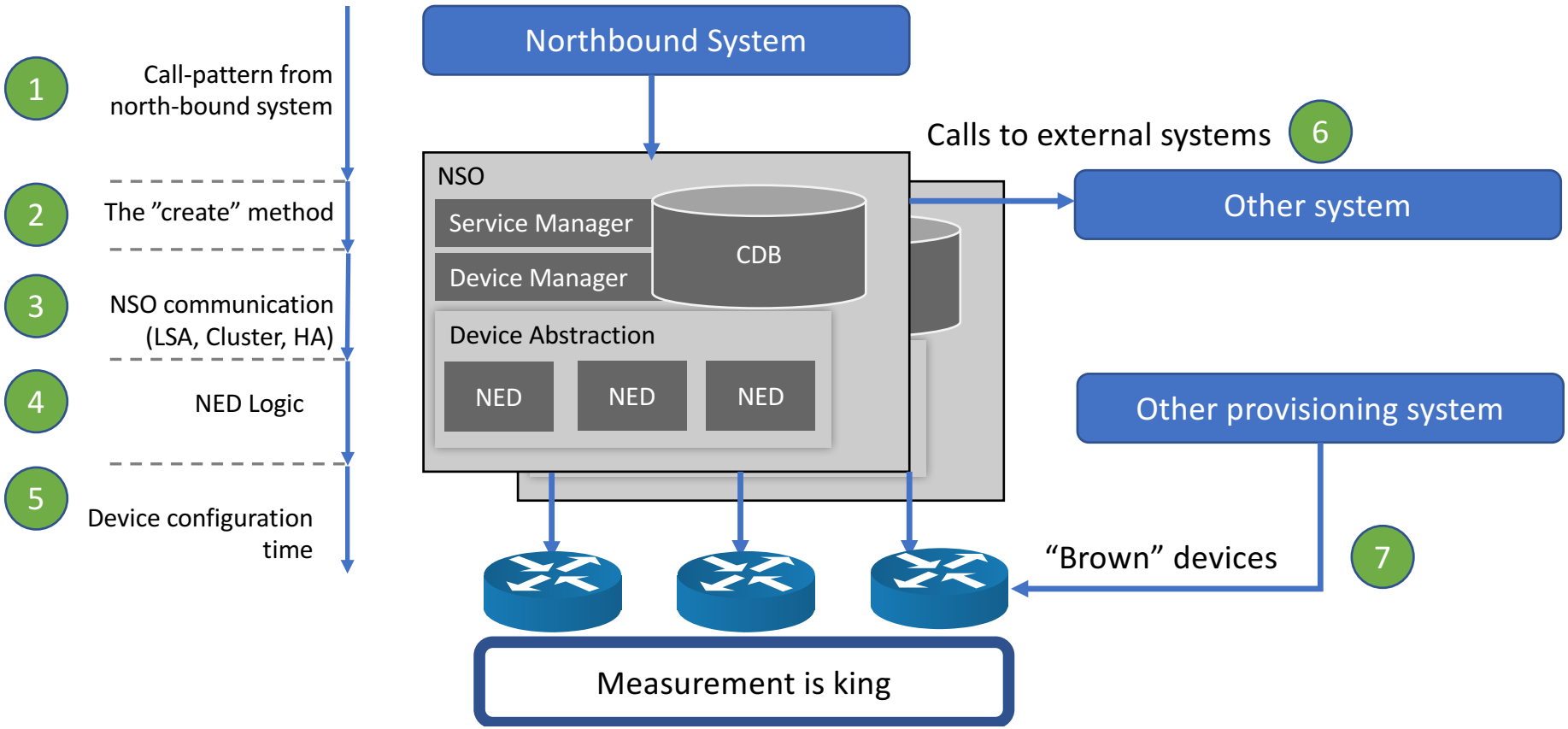
- Putting the pieces together: typical system deployments

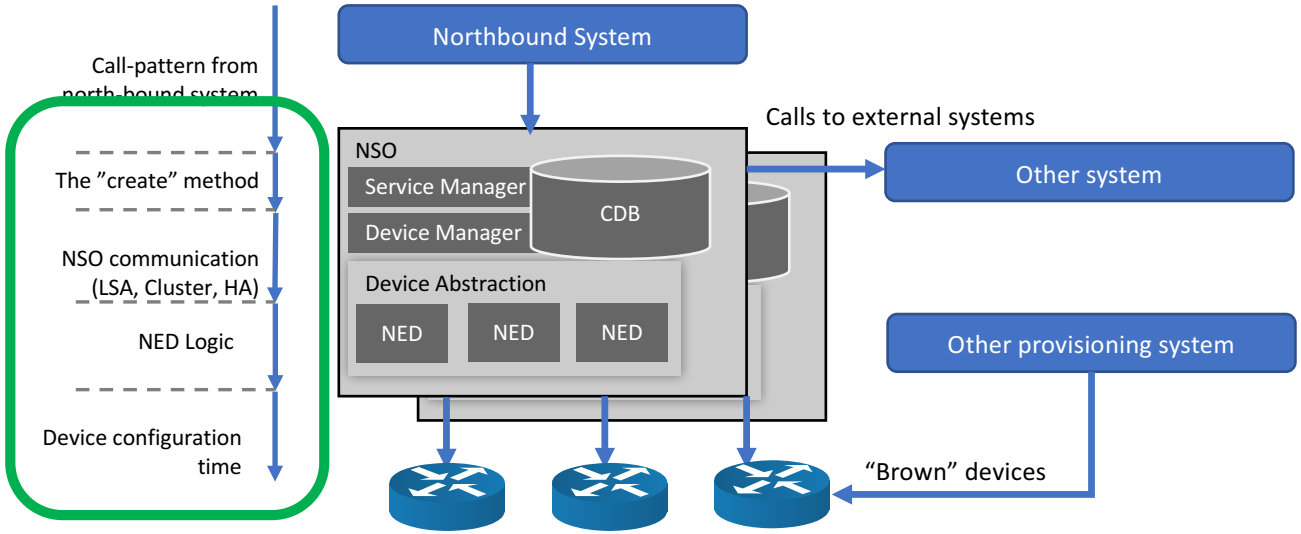
Terminology

- Throughput
  - Maximum rate of requests being processed
- Response time
  - The time taken to respond to a request
- Scalability
  - The capability to manage a large network; number of devices and services
- Reliability
  - The capability to function for a particular amount of time

Different characteristics, different NSO knobs

The Big Picture: what affects performance?

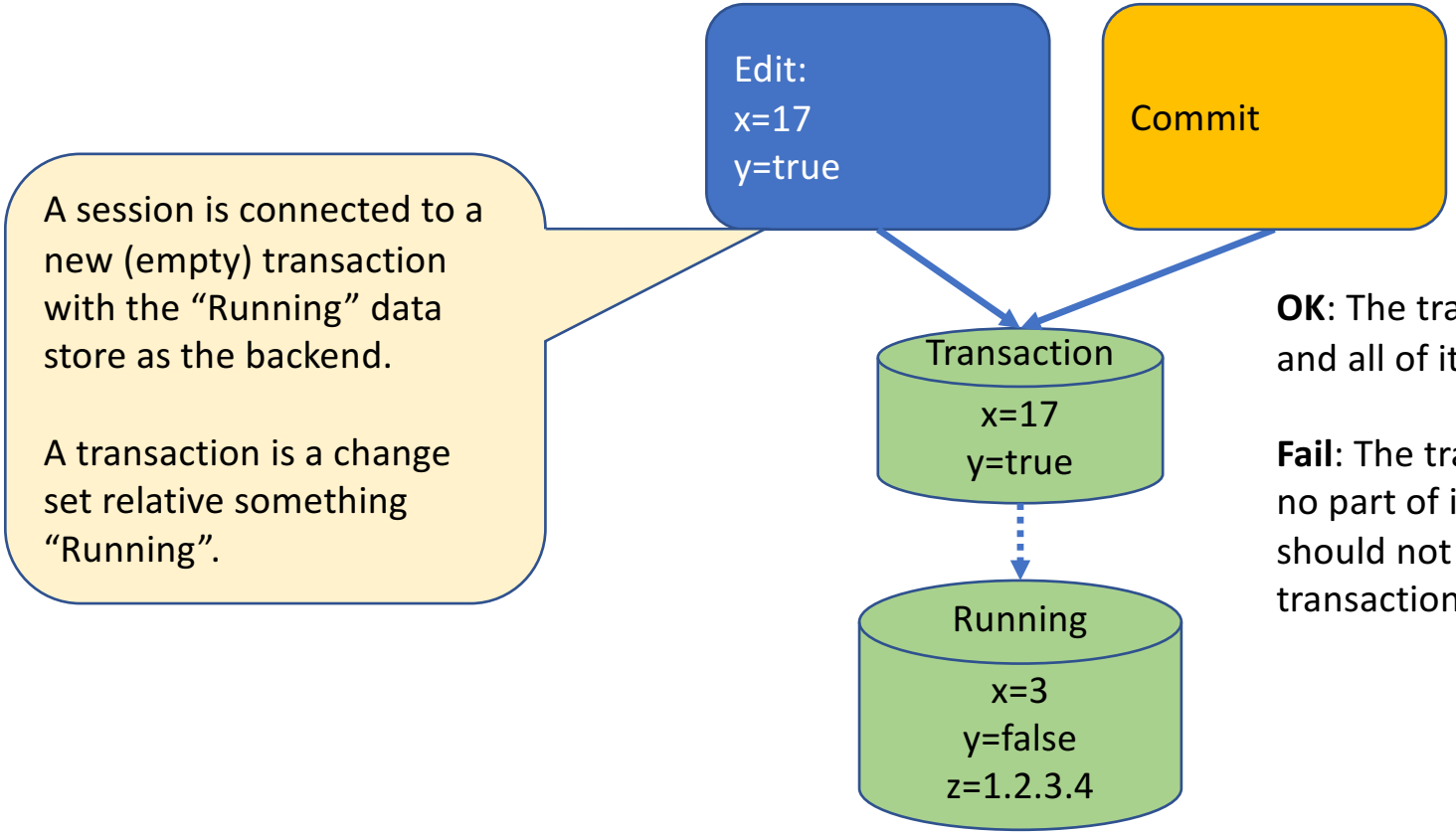




Deep Dive – the NSO transactional model

Important to understand the characteristics of your platform

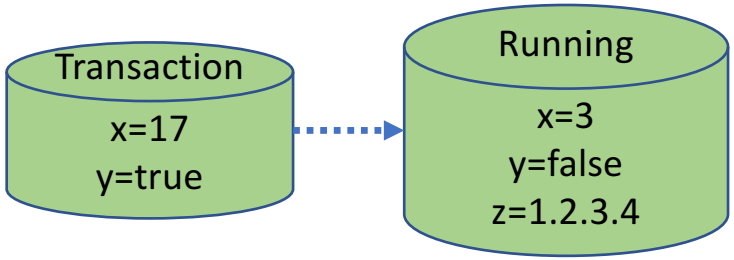
Commit

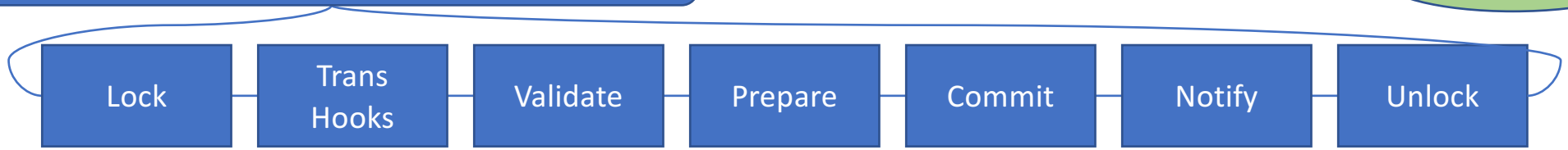


OK: The transaction was processed without error and all of it has been acted upon.

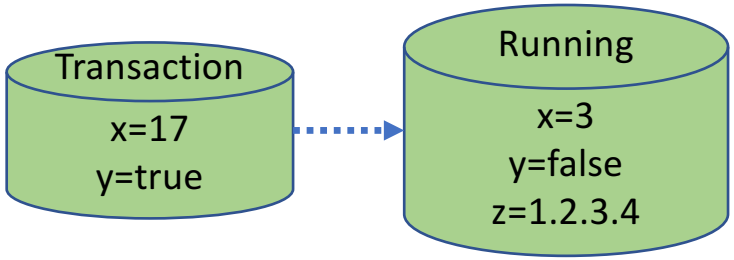
Fail: The transaction failed due to an error, and no part of it has been acted upon. In principle it should not be possible to observe a failed transaction for an outside observer.

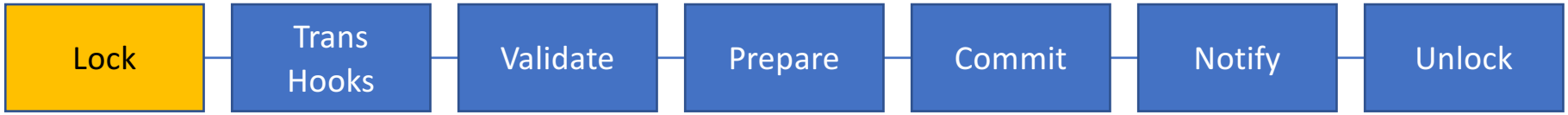
A closer look at the Commit Sequence





Lock Running

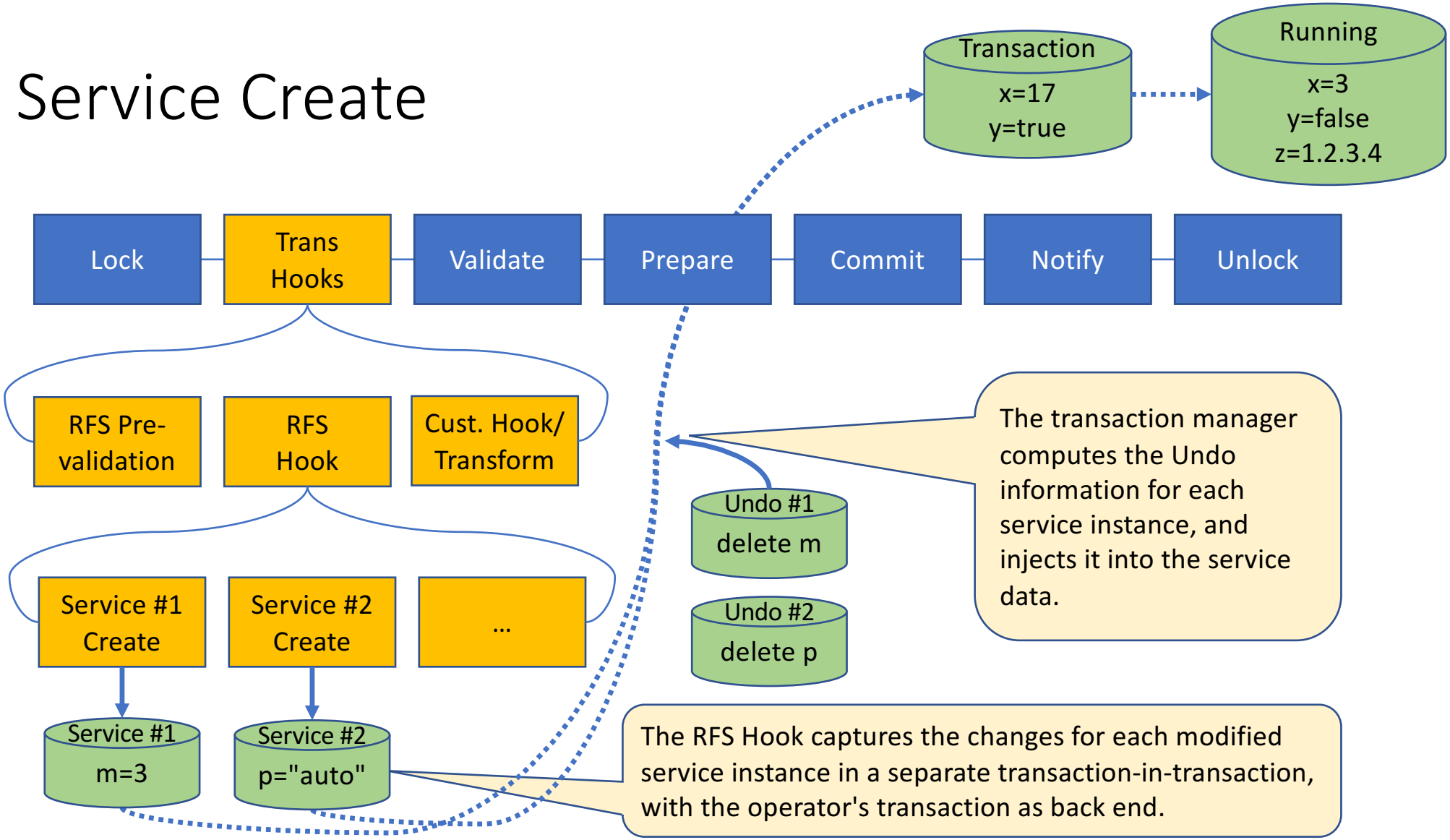




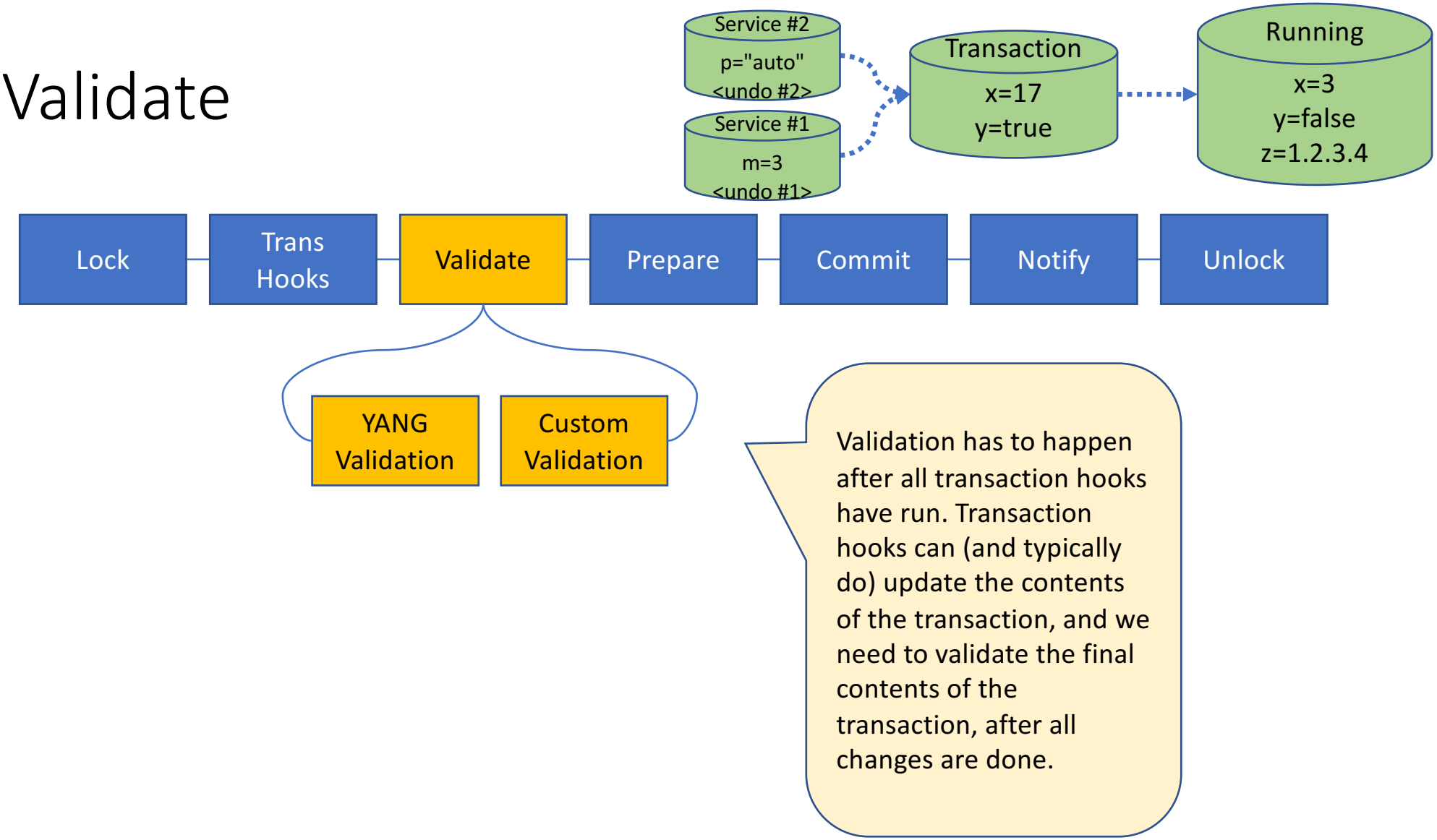
Locking, so that no other changes are under way while we are processing this transaction, is key to a simple programming model.

On the other hand, this constraint limits the maximum throughput (transactions per minute) of the system.

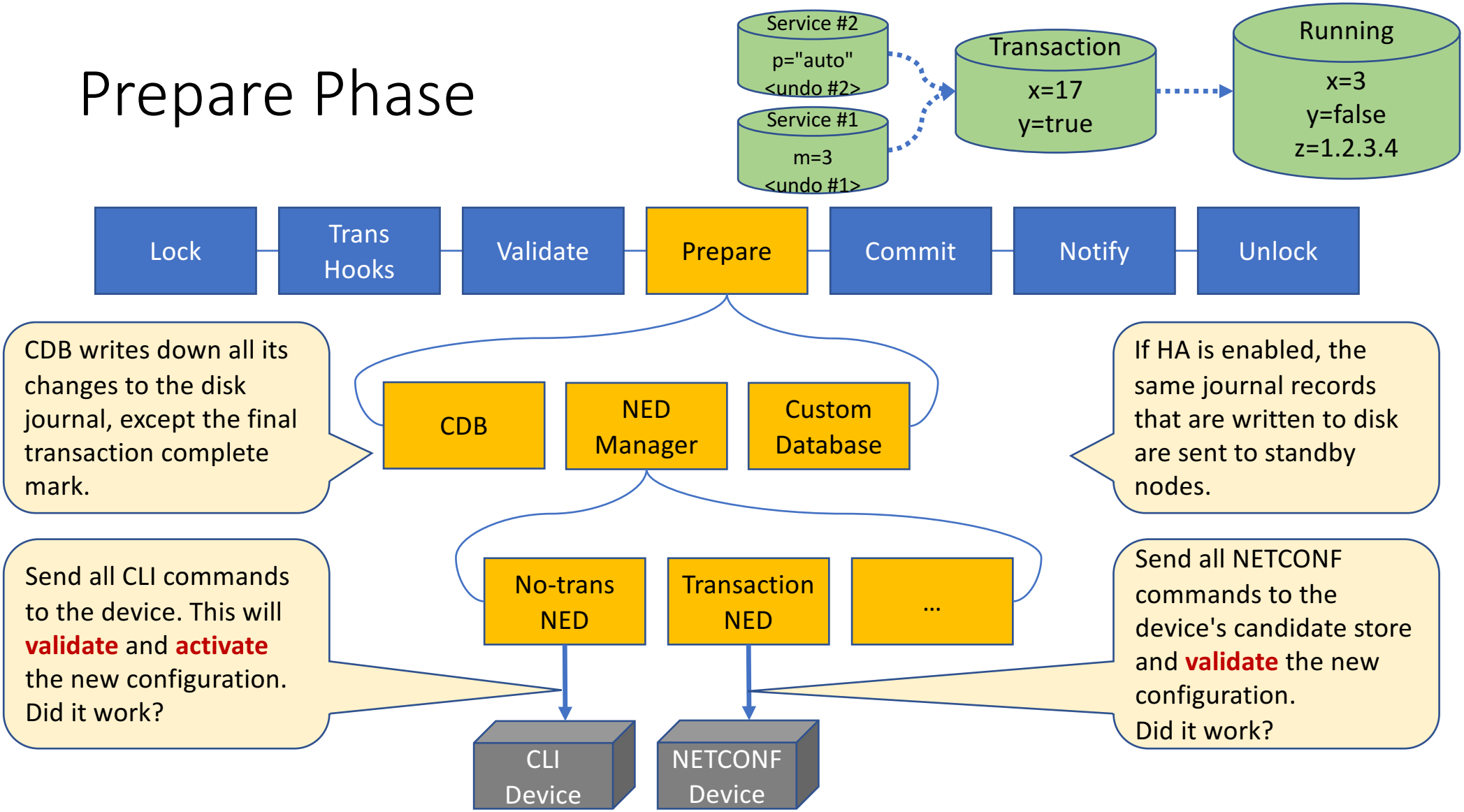
Service Create



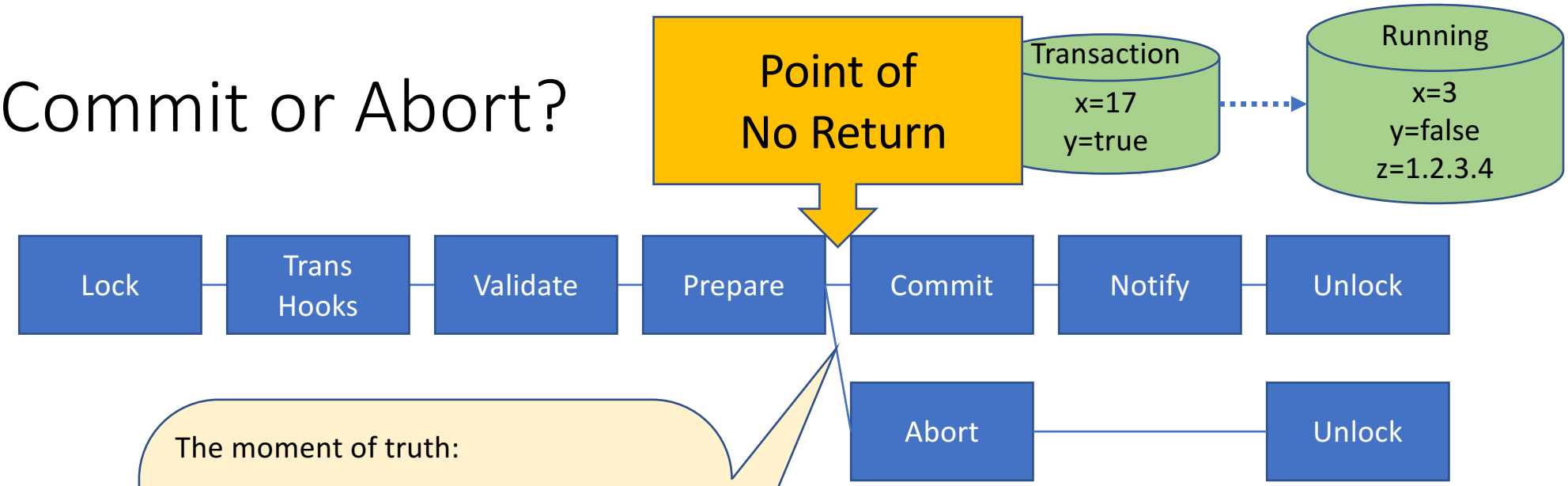
Validate


Service #2
p="auto"
<undo #2>
Service #1
m=3
<undo #1>
Transaction
x=17
y=true
Running
x=3
y=false
z=1.2.3.4
Lock
Trans Hooks
Validate
Prepare
Commit
Notify
Unlock
YANG Validation
Custom Validation
Validation has to happen after all transaction hooks have run. Transaction hooks can (and typically do) update the contents of the transaction, and we need to validate the final contents of the transaction, after all changes are done.

Prepare Phase



Commit or Abort?



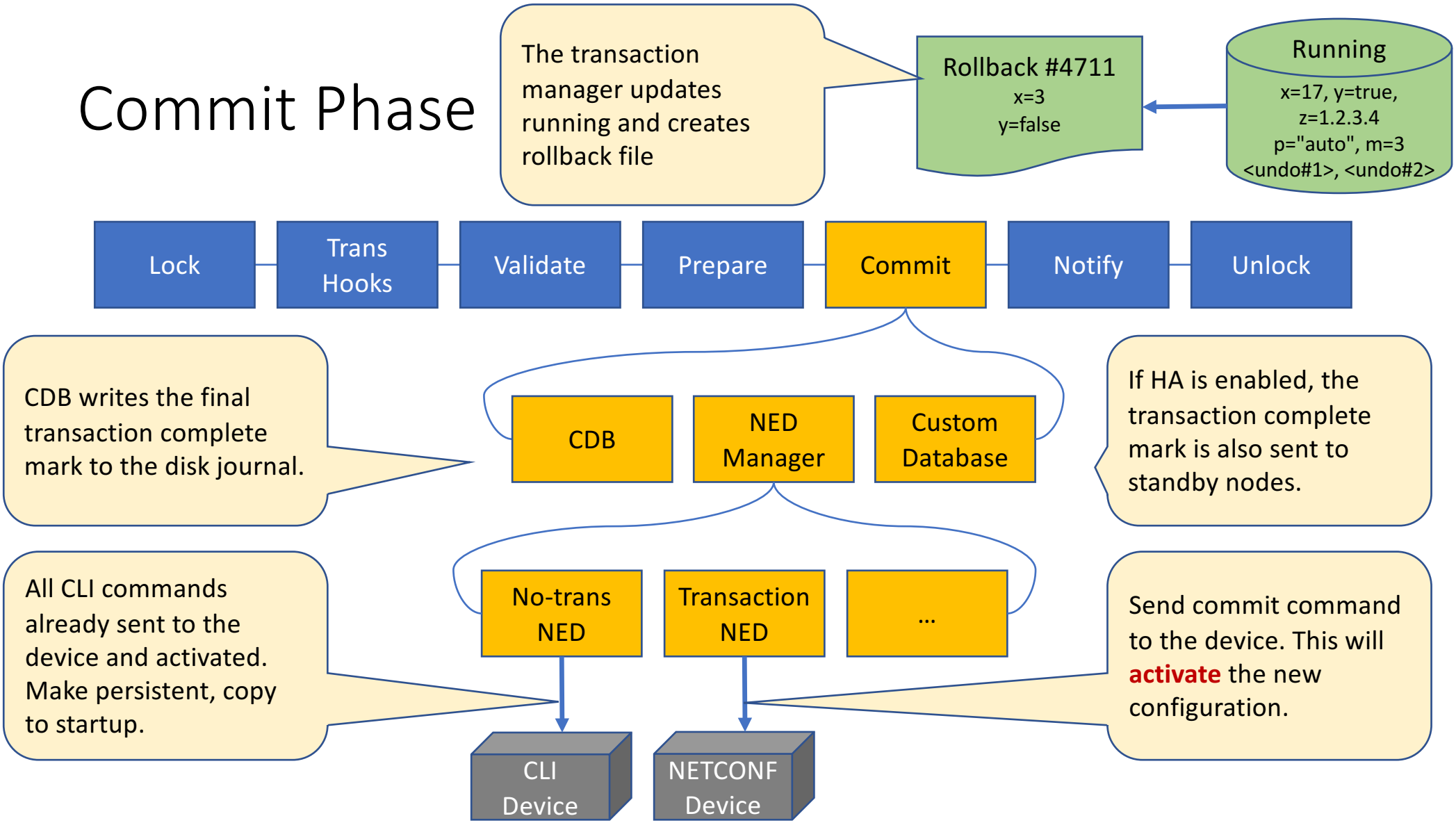
The moment of truth:

If any transaction participant returns failure, take the abort path.

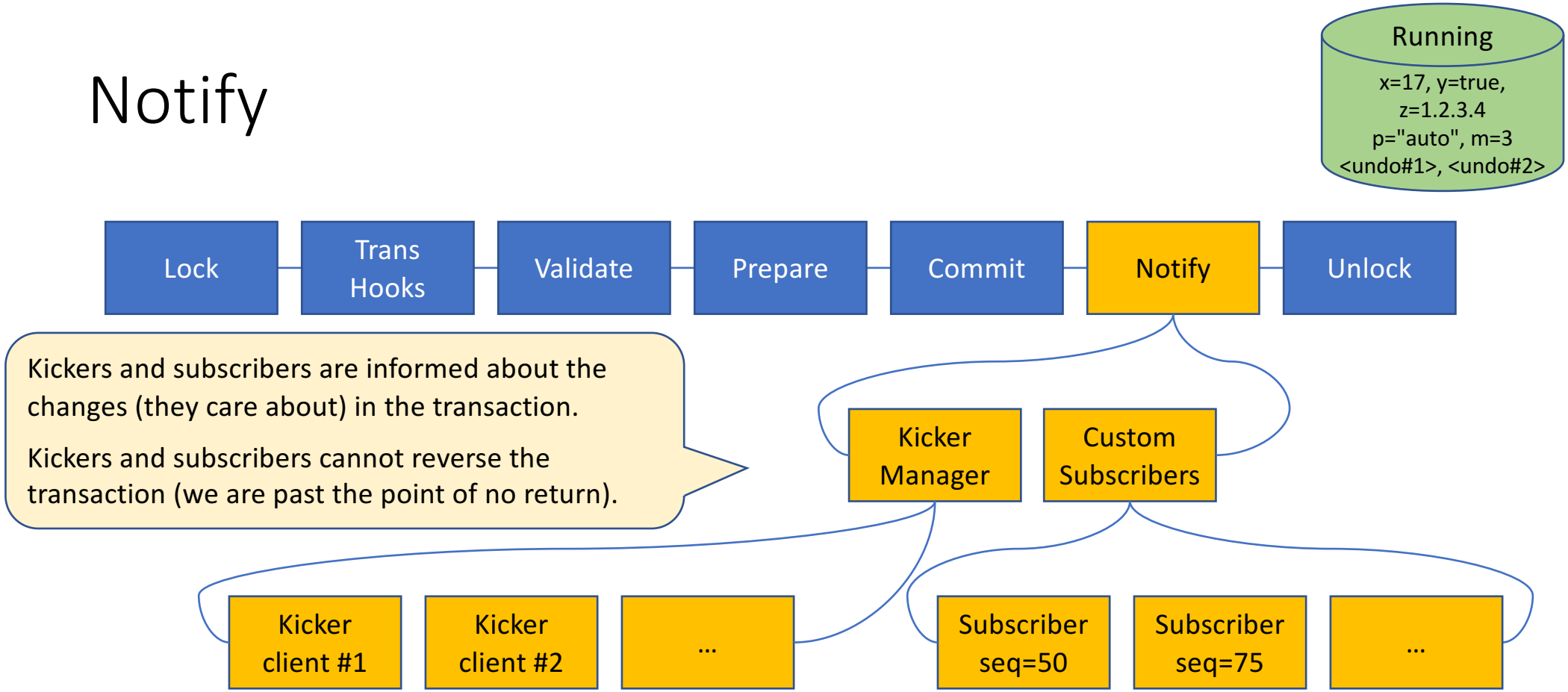
Otherwise proceed with Commit.

This is the point of no return. This is when we decide if the transaction went through or not. We will not (cannot) change our minds after this. This is defined by standard transaction theory.

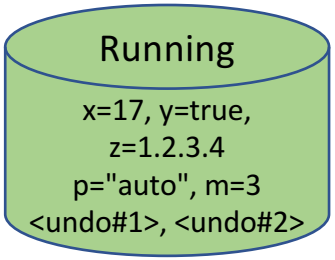
Commit Phase

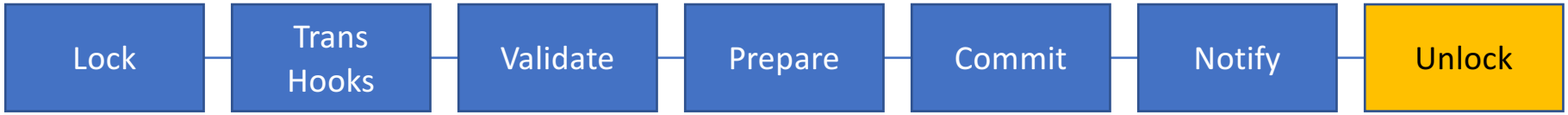


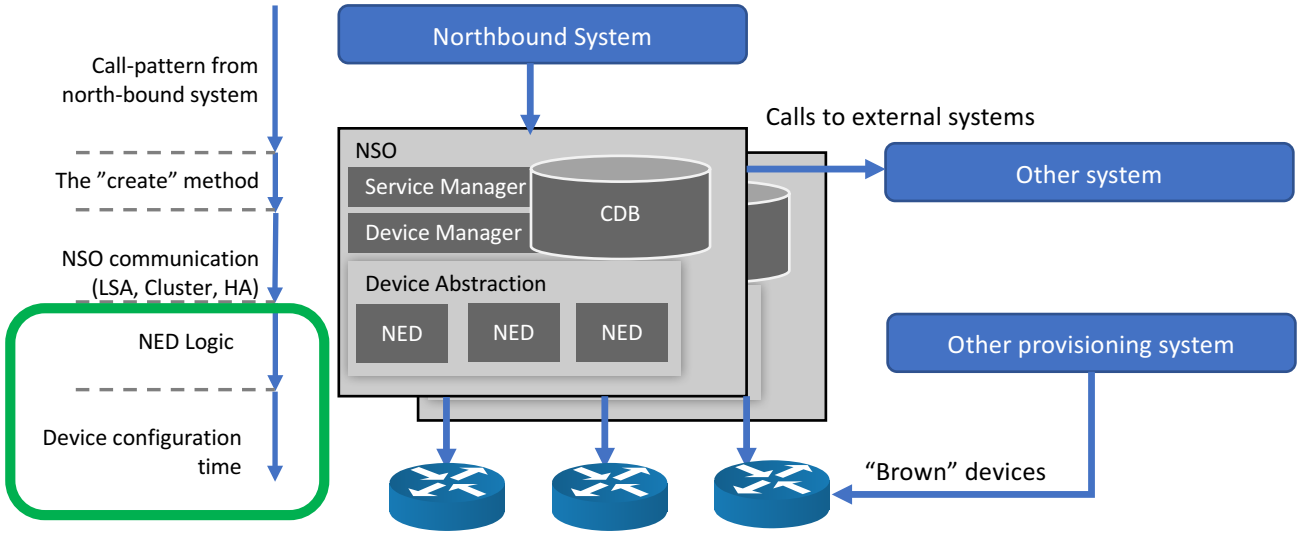
Notify


Running
x=17, y=true,
z=1.2.3.4
p="auto", m=3
<undo#1>, <undo#2>
Lock
Trans Hooks
Validate
Prepare
Commit
Notify
Unlock
Kickers and subscribers are informed about the changes (they care about) in the transaction.
Kickers and subscribers cannot reverse the transaction (we are past the point of no return).
Kicker Manager
Custom Subscribers
Kicker client #1
Kicker client #2
...
Subscriber seq=50
Subscriber seq=75
...

Unlock Running



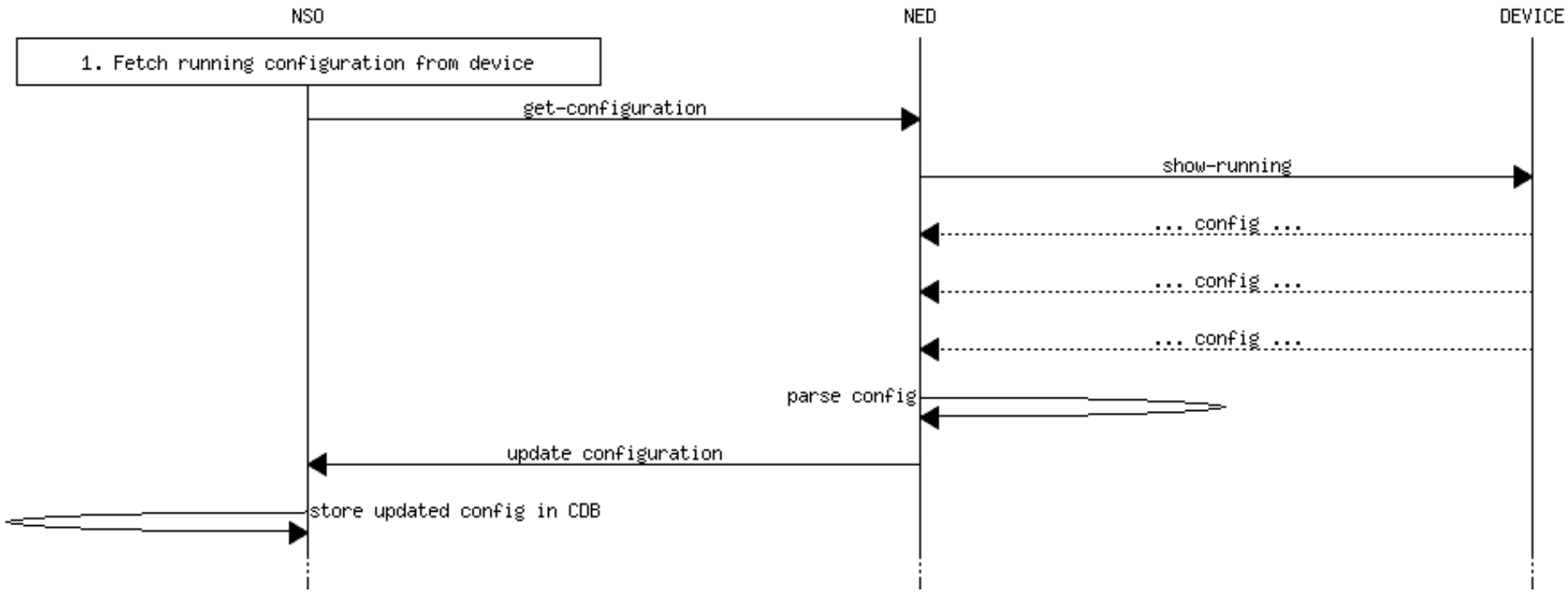




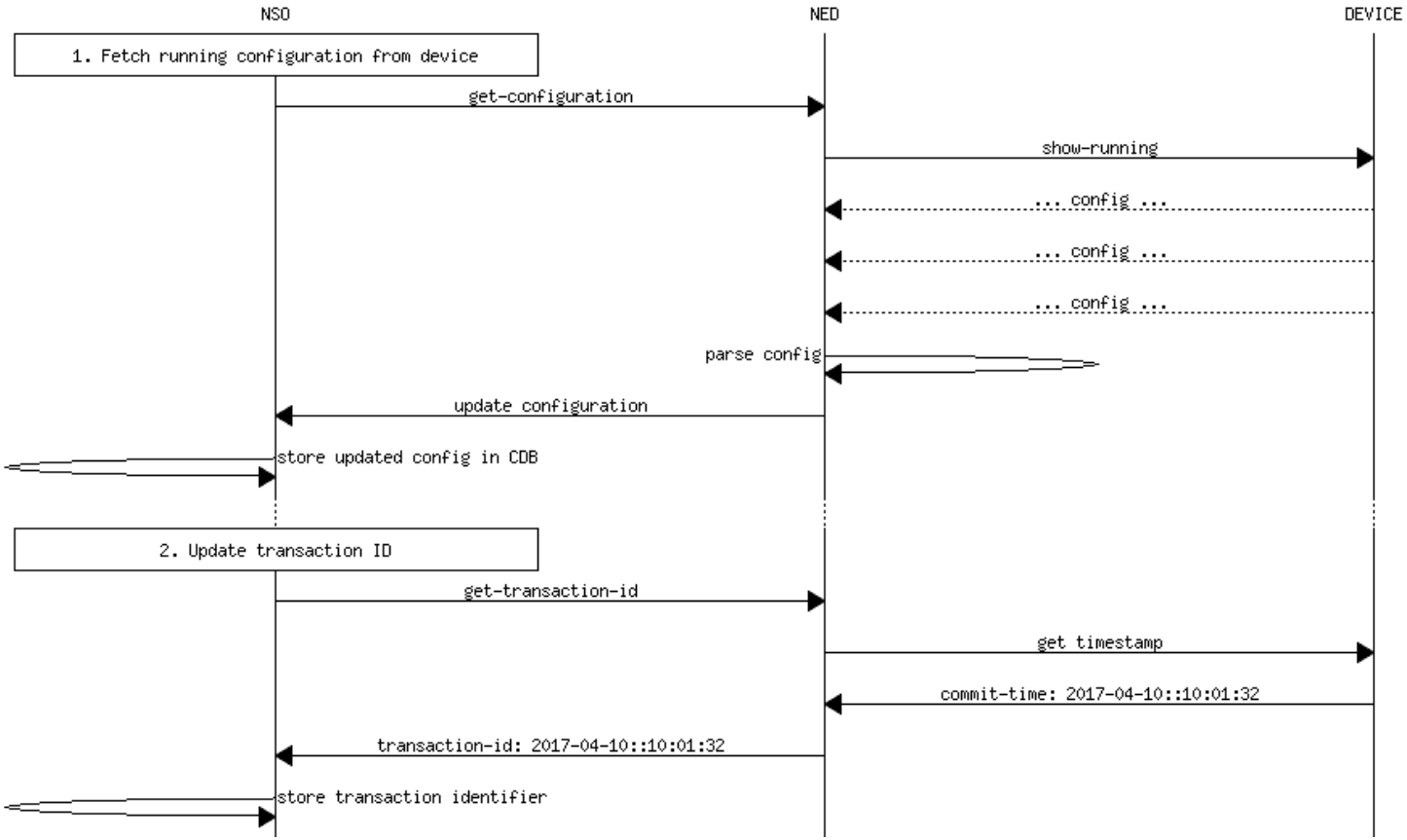
Deep Dive – Understanding "device cost"

Important to understand the characteristics of your platform

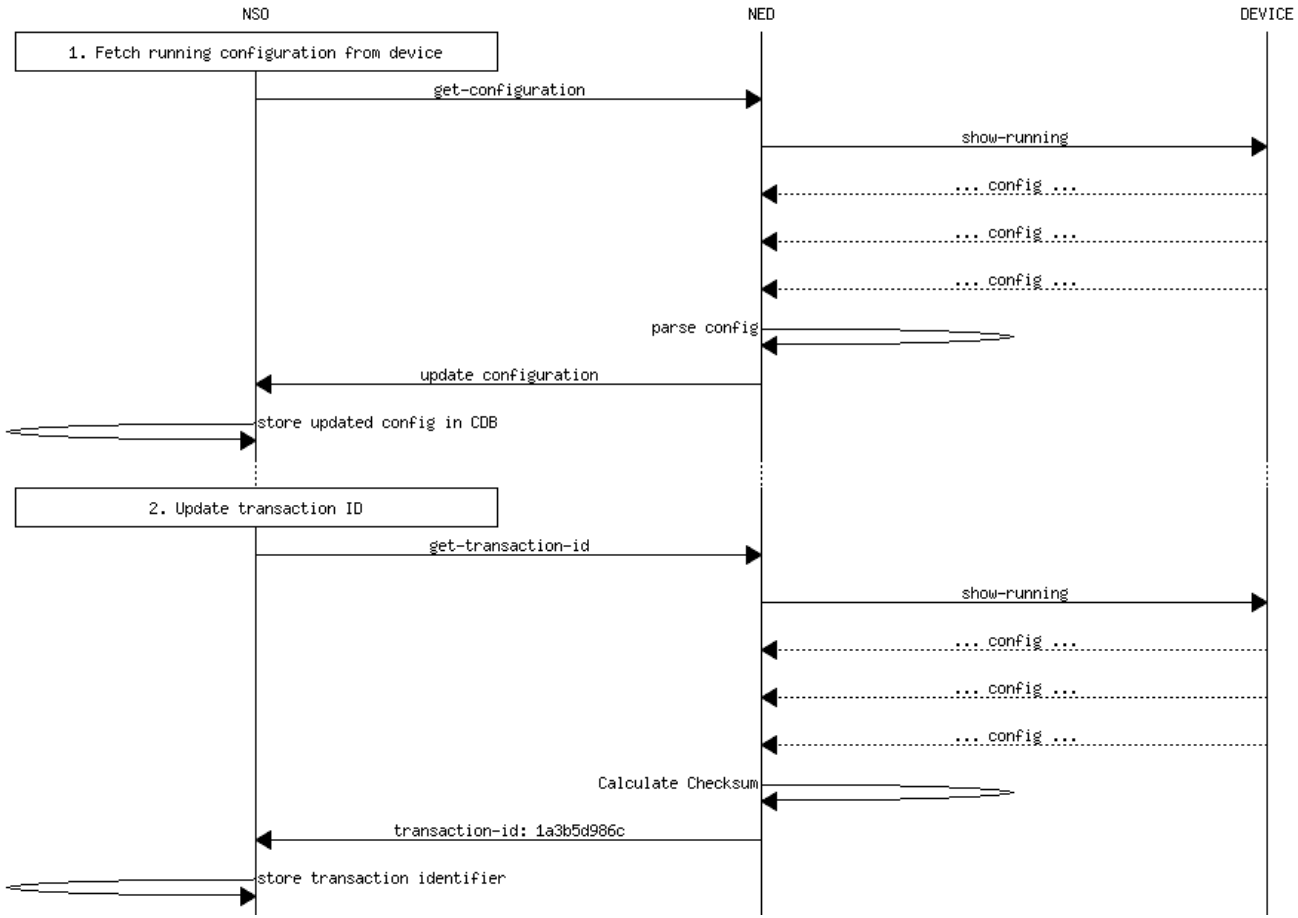
What is a sync-from really?



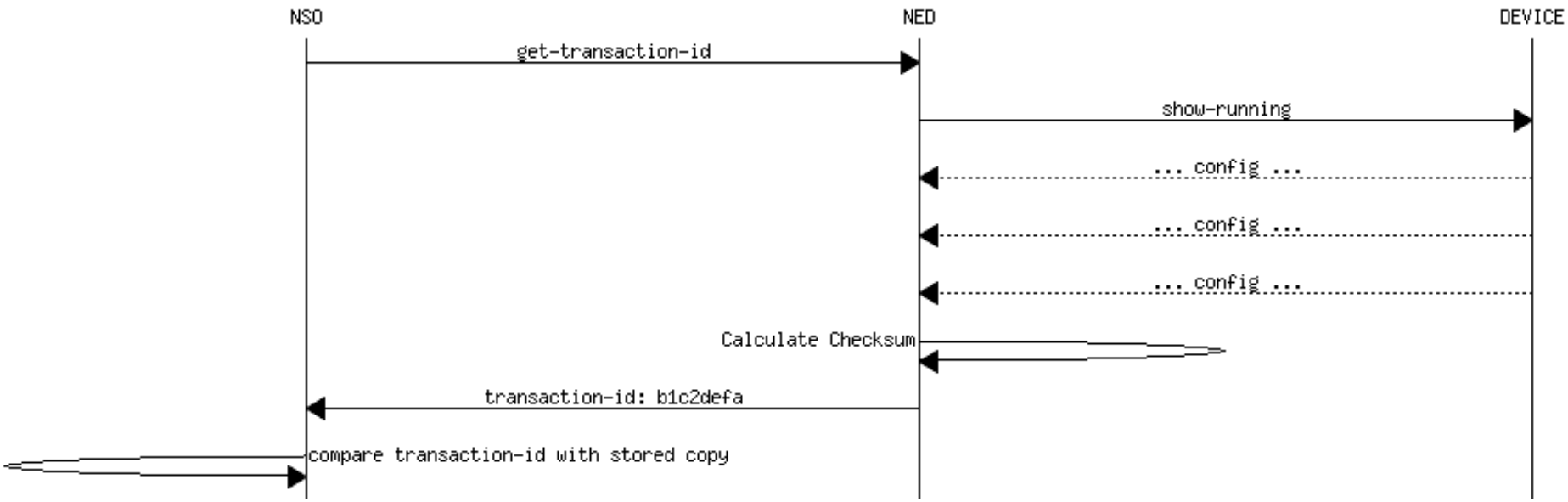
Sync-from, device has time-stamp



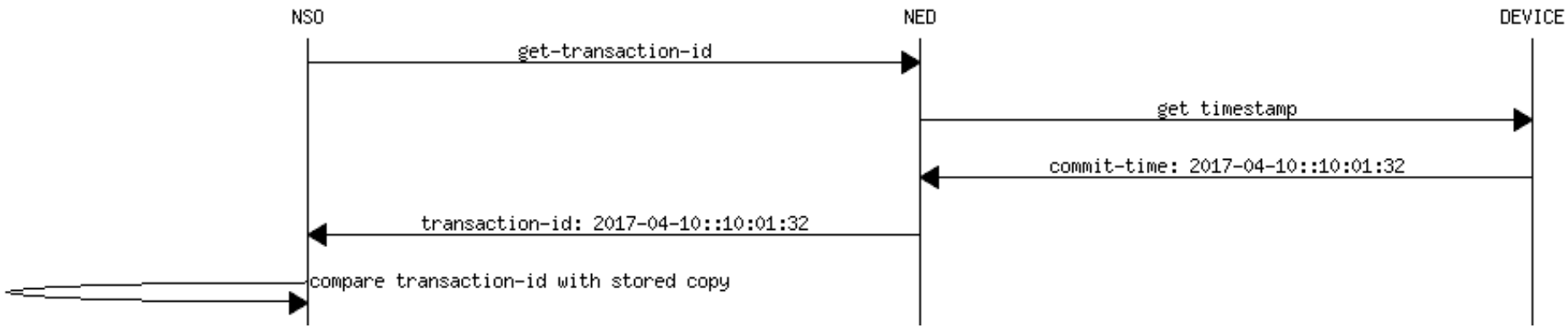
Sync-from, device has no time-stamp



Check-sync device with no transaction-id/time-stamp

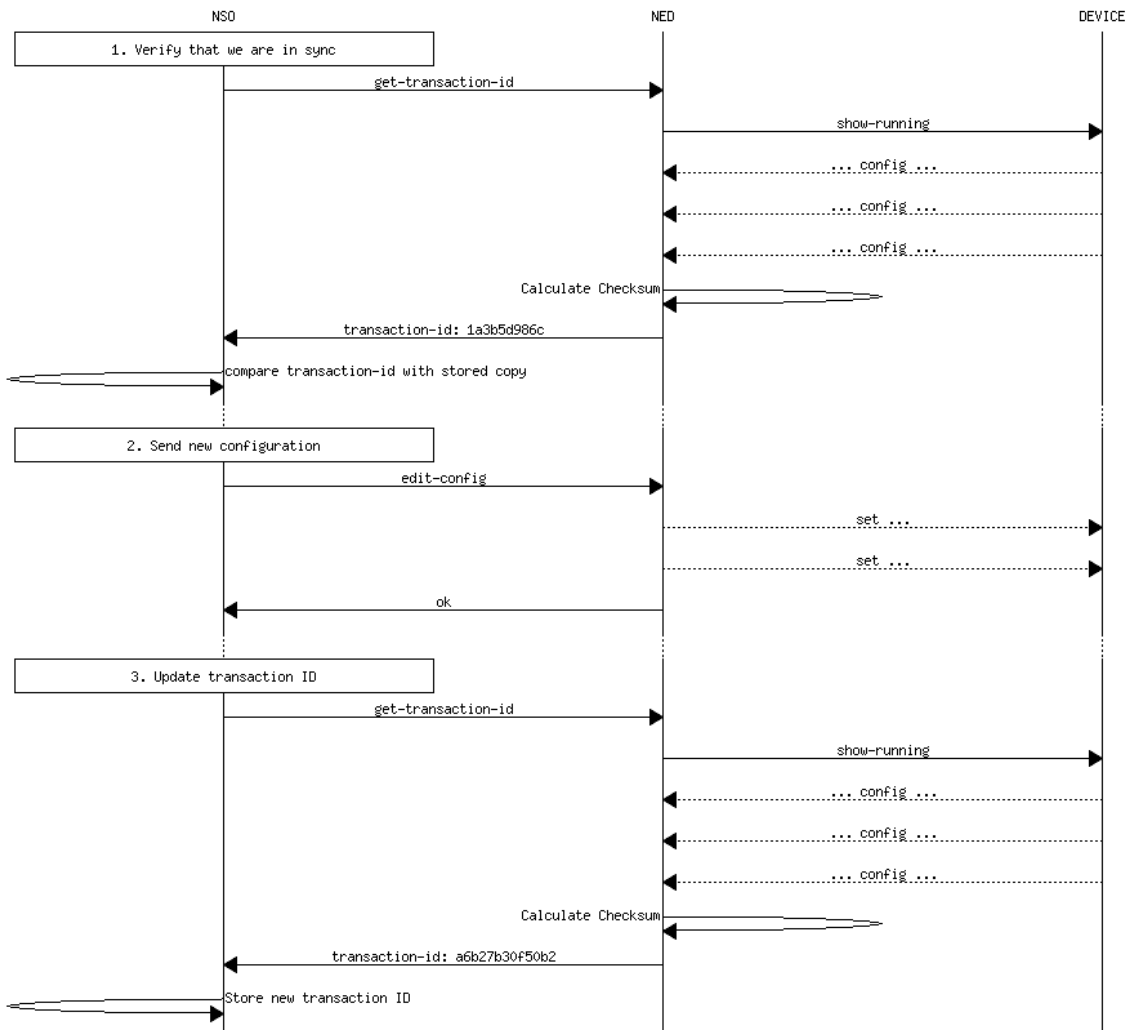


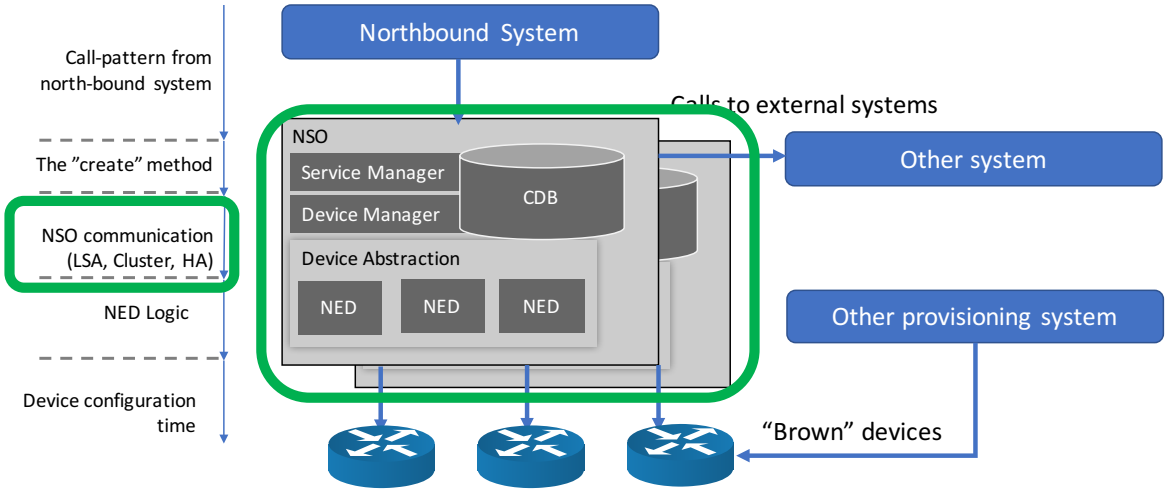
Check-sync device has time-stamp



Example of complete sequence

Normal Commit





Deployment Models

Commit Queue

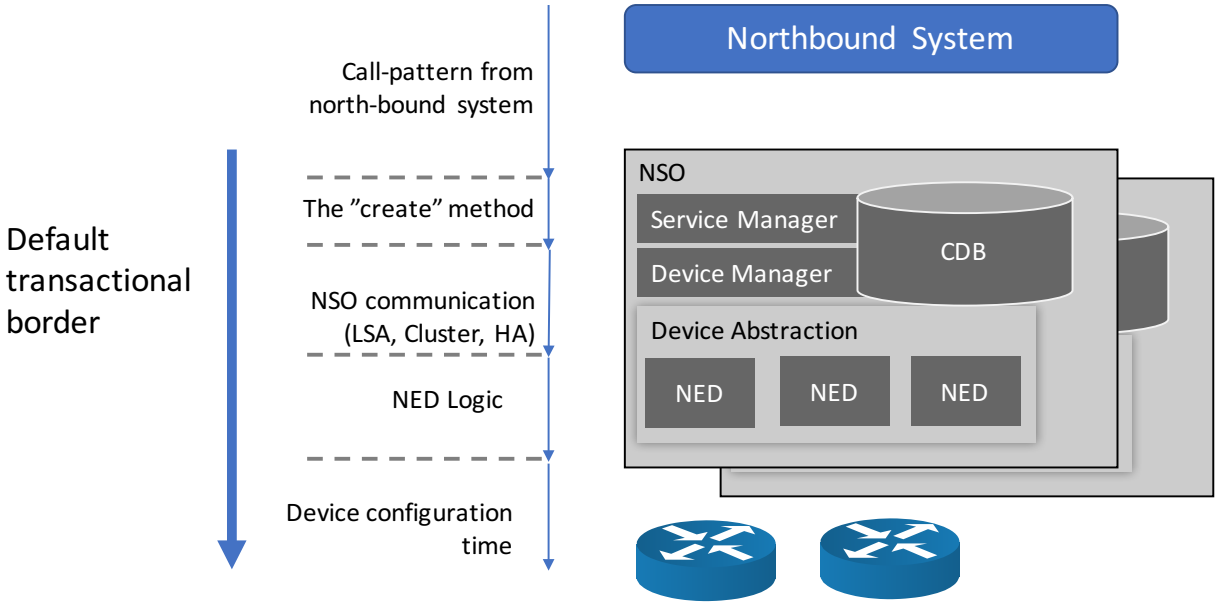
Layered Service Architecture, LSA Cluster

Device Cluster

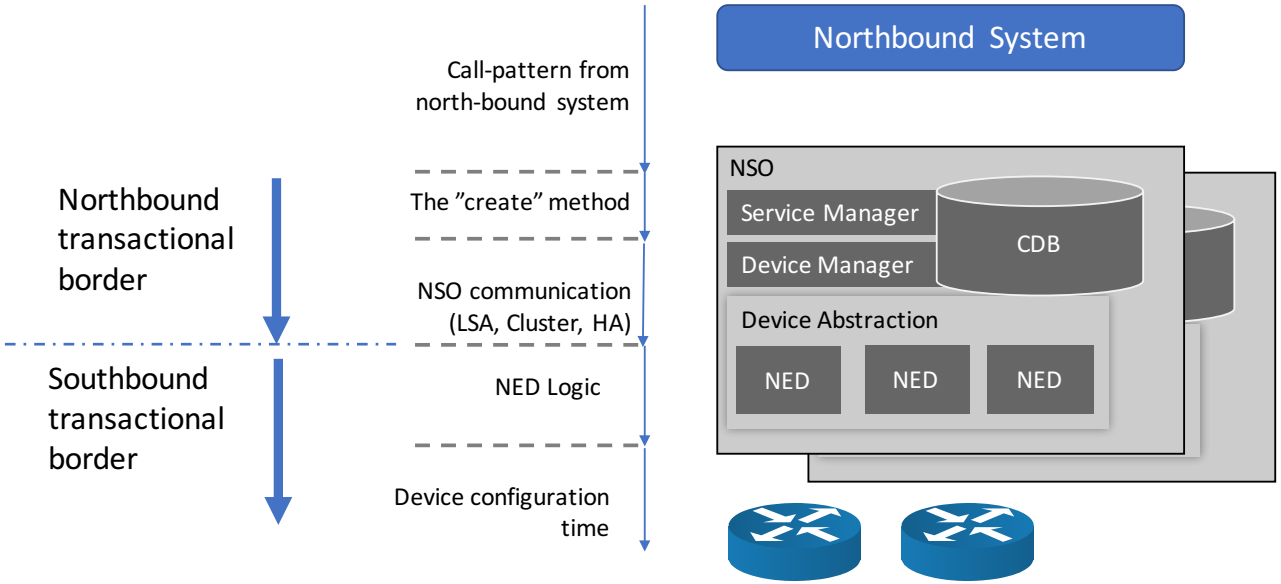
High Availability Cluster

Host Machines

Default NSO transactional model



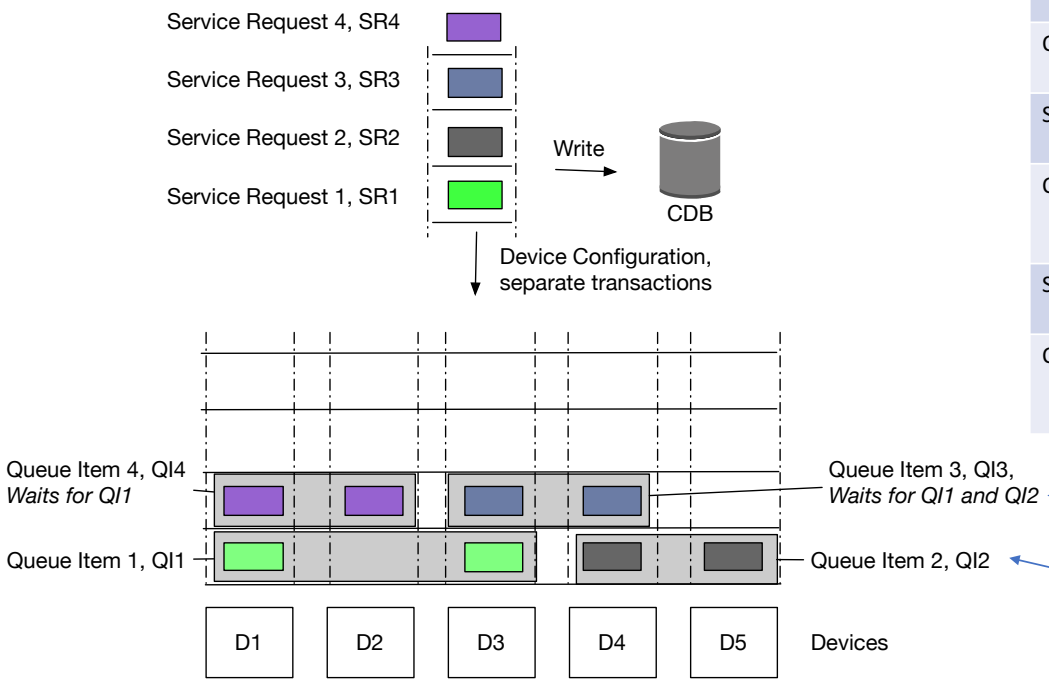
With Commit Queue



- Throughput ++
- Response time + -
- Scalability 0
- Reliability + -

Main Mechanism for throughput
No impact on code
No ACID transaction including the network, "eventual consistency"
Atomic and non-atomic commit queue

Commit queue behavior

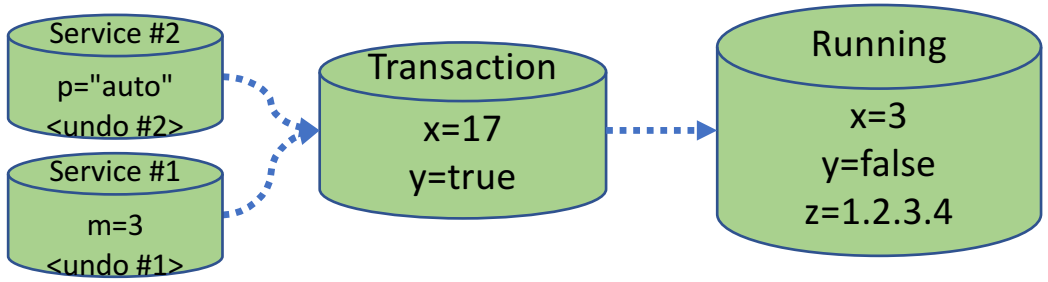


	Transactions
SR1	Perform mapping logic, short time inside lock. Enqueue QI1 in Commit Queue
QI1	One transaction for D1 and D3, outside lock, southbound diff calculated here.
SR2	Perform mapping logic, short time inside lock. Enqueue QI2 in Commit Queue
QI2	One transaction for D4 and D5,outside lock, southbound diff calculated here. Runs in parallel with QI1
SR3	Perform mapping logic, short time inside lock. Enqueue QI3 in Commit Queue.
QI3	One transaction for D3 and D4, outside lock, southbound diff calculated here. QI3 is waiting for QI1 and QI2

QI3 and QI4 might run in parallel (Depending on when QI1 and QI2 ends)

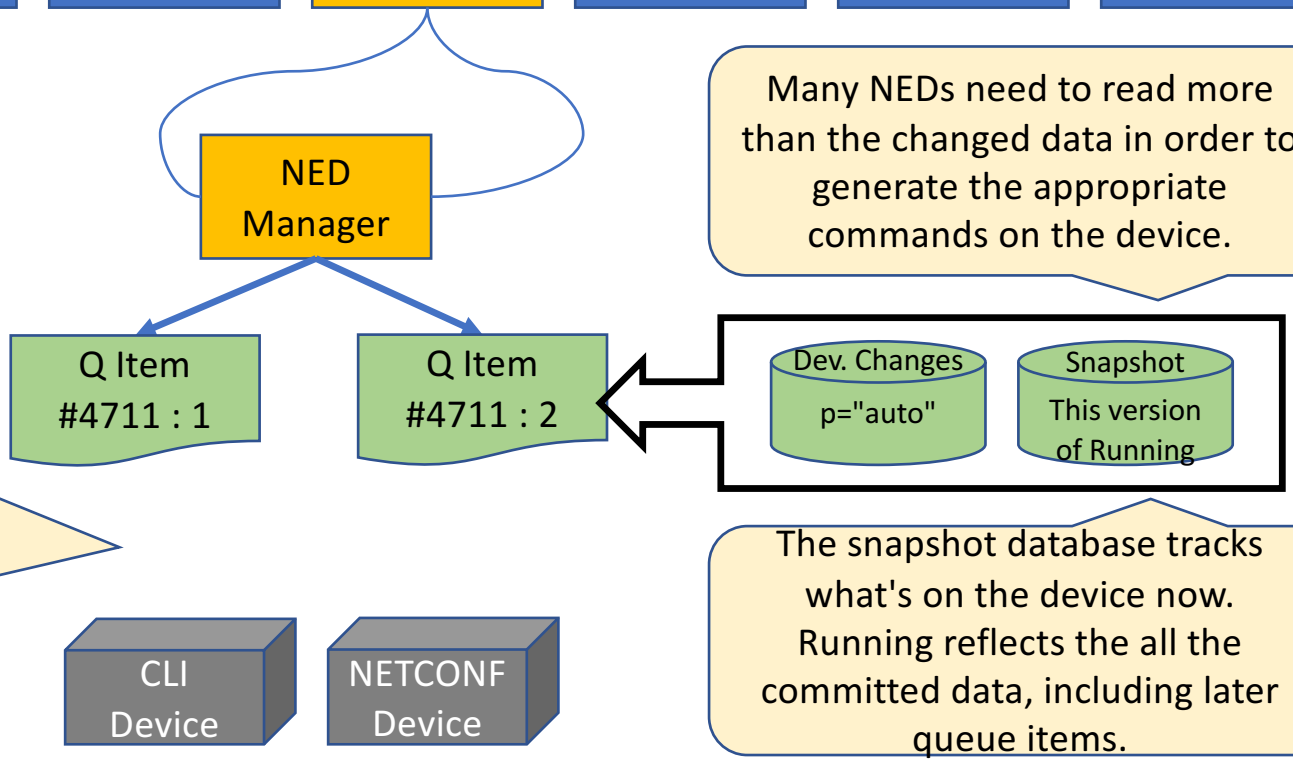
QI1 and QI2 runs in parallel

Commit Queue: Prepare



When commit queues are enabled, the NED Manager prepares a queue item for each involved device.

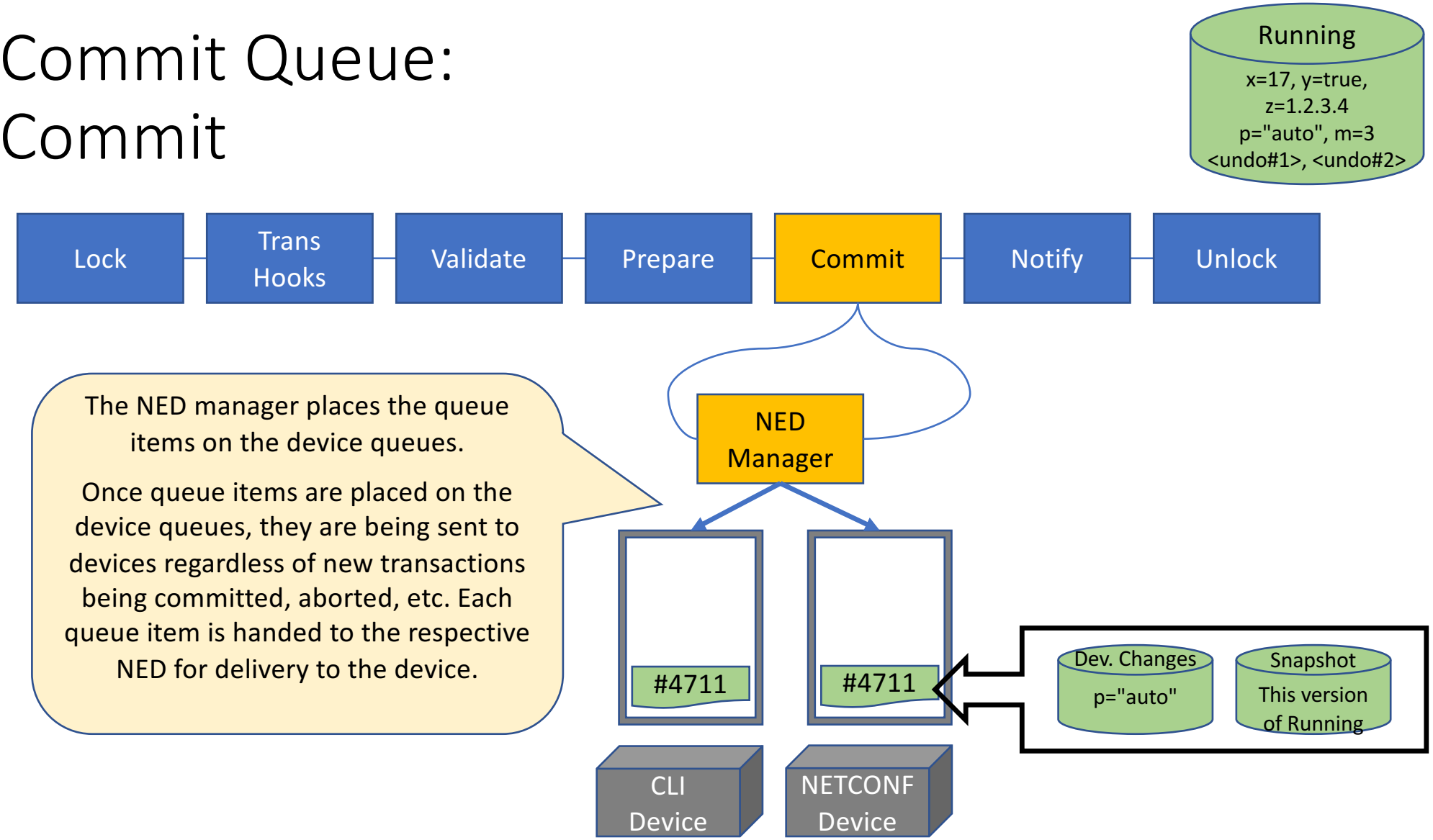
Each queue item consists of the device specific change set and a snapshot handle to enable reading from Running as it looks at the time of queueing.



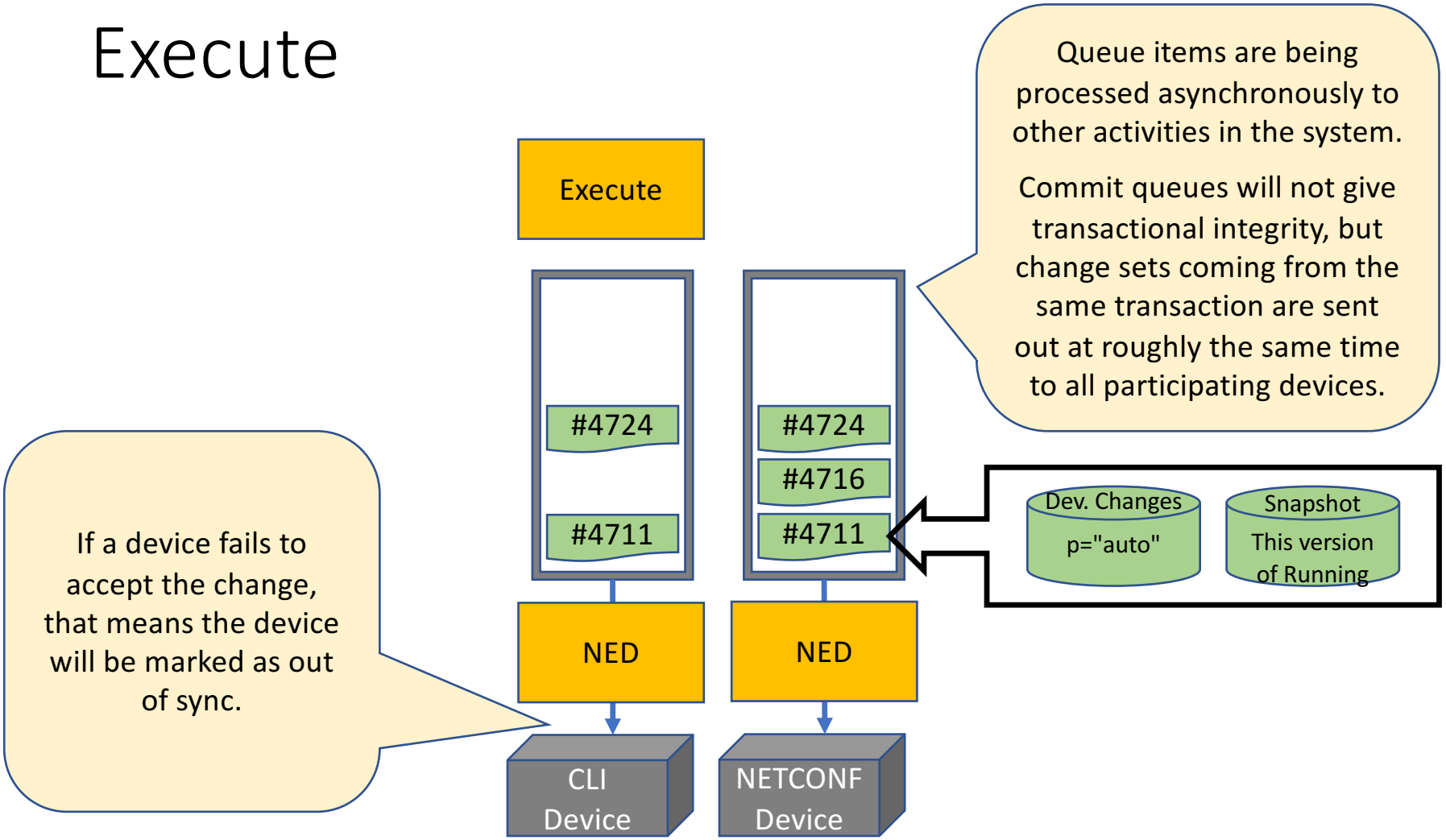
Many NEDs need to read more than the changed data in order to generate the appropriate commands on the device.

The snapshot database tracks what's on the device now. Running reflects the all the committed data, including later queue items.

Commit Queue: Commit



Commit Queue: Execute

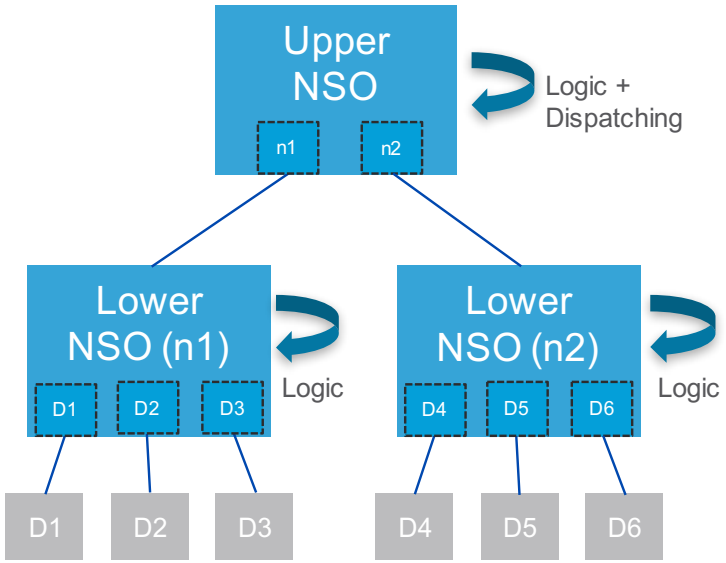


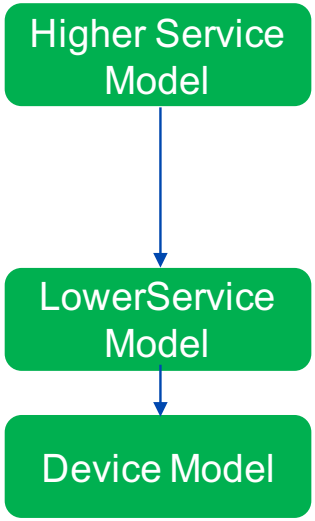
Queue items are being processed asynchronously to other activities in the system.

Commit queues will not give transactional integrity, but change sets coming from the same transaction are sent out at roughly the same time to all participating devices.

If a device fails to accept the change, that means the device will be marked as out of sync.

LSA Cluster

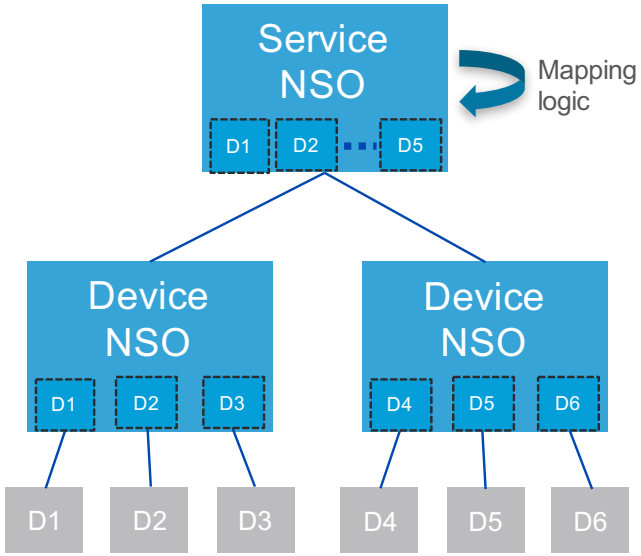




- Throughput +
- Response time -
- Scalability +
- Reliability + -

Service Code impacted
Split the devices across NSO nodes
Parallelism
Always combined with commit queues to gain performance

Device Cluster



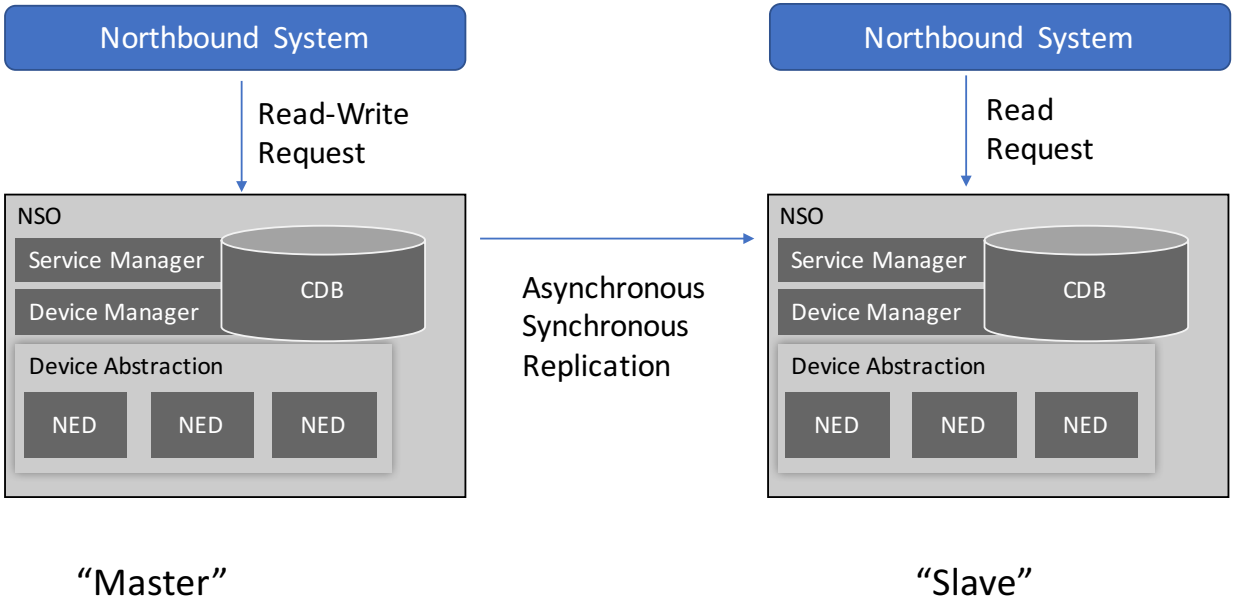
- Throughput --
- Response time -
- Scalability +
- Reliability + -

Device Cluster was the first scalability cluster model for NSO
Evolved to LSA
Factor 5 of performance hit in NSO-NSO communication
Used to localize NSO close to devices

LSA Cluster vs Device Cluster

Device Cluster	LSA Cluster
Top node has device meta-data. “Sees” all devices.	Top node has no device knowledge
Top node does not “see” the south NSO node. The dispatching of device operations is managed by NSO	Top node sees the south NSO nodes as another NSO.
Service code is not impacted by the deployment	Service code need to be split between the NSO nodes
High penalty in NSO – NSO communication	Much less penalty since the only communication is the service diff.
Tight coupling between all NSO nodes	Loose coupling between NSO nodes

High Availability Cluster

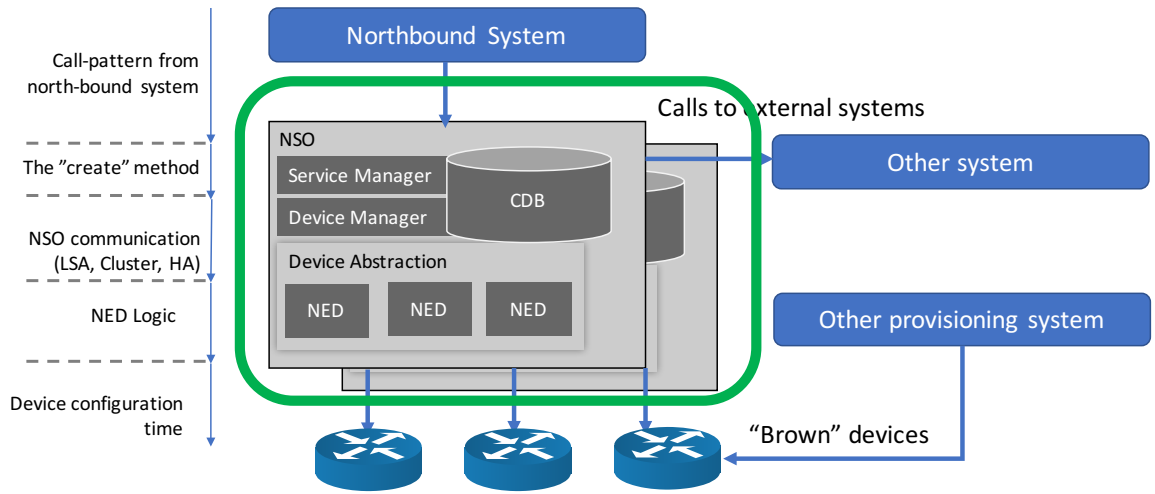


"Slowest" NSO and link latency will limit performance

- Throughput (- if sync)
- Response time (- if sync)
- Scalability
- Reliability + (if sync)

Host machine, some obvious statements

- The host machine characteristics has huge impact
  - CPU
  - Memory
  - Local Disc
  - Virtual or bare metal
- Develop and test on relevant machine



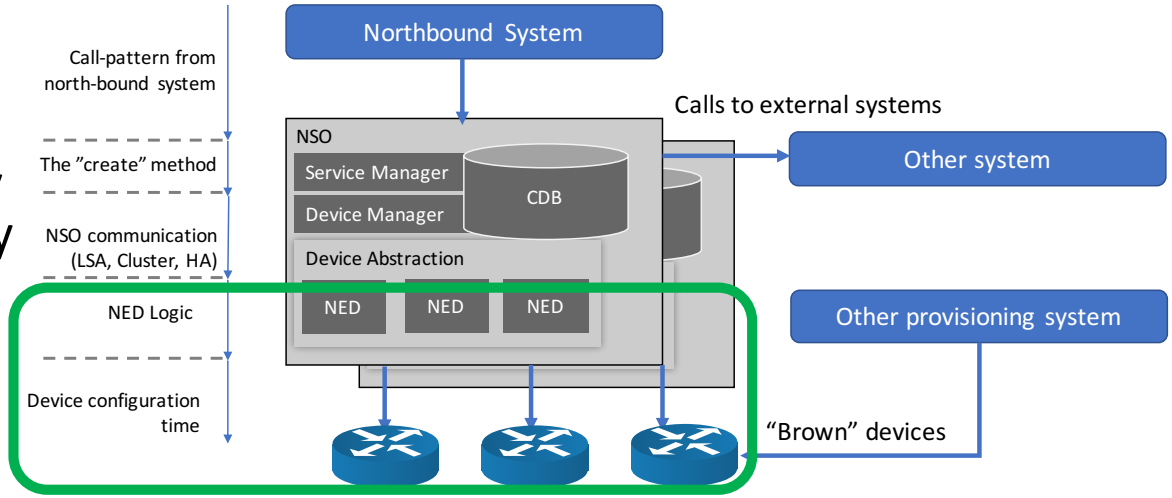
NSO System Configuration

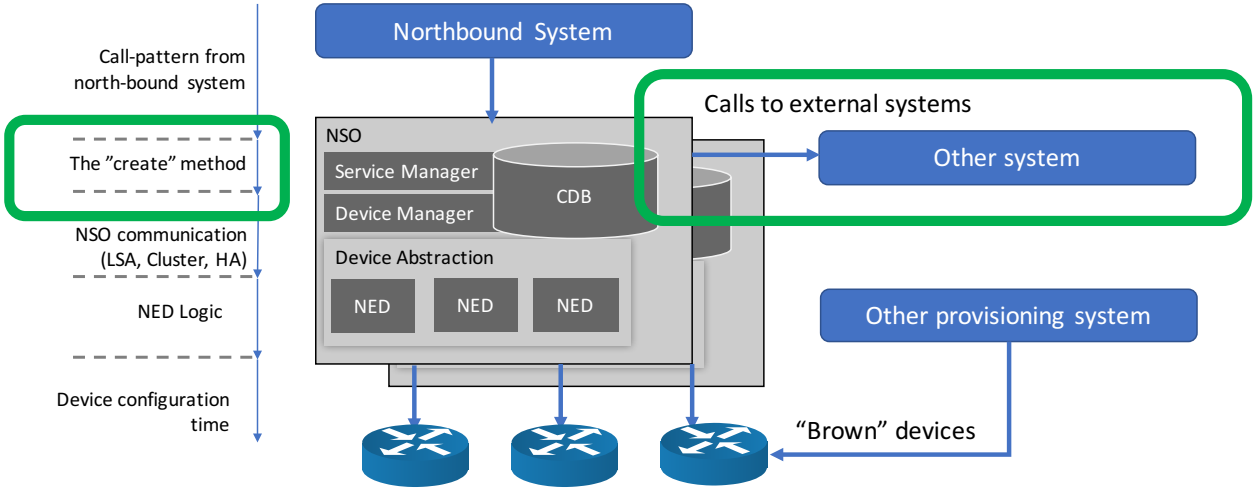
Performance impact

- Study different configuration options for NSO
  - *ncs.conf*
  - Examples
    - Trace files
    - Logging
    - Rollbacks
    - Diffs
    - Pre populate snap-shot db
    - …

Device and NED Configuration

- NEDs can be configured for different check-sync mechanisms
  - Hash or Transaction id
  - Data transfer method (scp or show config)
- Device behavior
  - The time for a device to apply the configuration can differ by configuration options
  - HA-behaviour takes time
  - Commit script on the device
  - …





Developing the service code

The Create Method

```
 */
@ServiceCallback
(servicePoint = "l3vpn-servicepoint", callType = ServiceCBType.CREATE
 ) public Properties create(ServiceContext context,
                            NavuNode service,
                            NavuNode ncsRoot,
                            Properties o
    throws ConfException {
    ParamPad pad = setupServiceParams(co                                e);
    LOGGER.info("L3vpn create: " + pad.s
    int epCountdown = 0;
    boolean failure = false;

    try {
        createNCCAccount(pad);
```

- CDB locked 

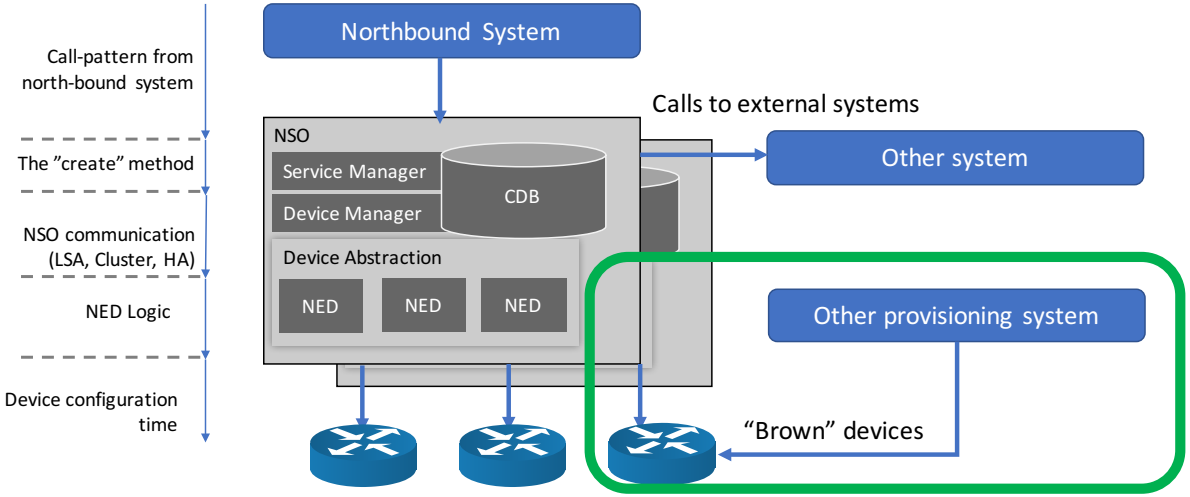
- Measure and instrument: **<< 1 second!**
- “Avoid” other logic then service to device mapping
- Watch out:
  - Anything computational heavy, how does your algorithms scale, measure time.
  - Run performance tests for the intended size of the network.
  - XPATH expressions, tune the expression for performance
  - Validation code, check carefully that the validation code scales to the size of the network. Again measure.
  - Never perform sync-from in the create method
- If you can make the create method a template and not code, go for it.
  - Design-time validation as well

The Create Method

```
 */
@ServiceCallback
(servicePoint = "l3vpn-servicepoint", callType = ServiceCBType.CREATE
 ) public Properties create(ServiceContext context,
                            NavuNode service,
                            NavuNode ncsRoot,
                            Properties o
    throws ConfException {
    ParamPad pad = setupServiceParams(co                                e);
    LOGGER.info("L3vpn create: " + pad.s
    int epCountdown = 0;
    boolean failure = false;

    try {
        createNCCAccount(pad);
```

- “Externalize” things that take time
  - put these steps in separate actions
  - the complete sequence can be combined in several ways
    - Use Reactive FastMap:
      - the create method releases quick and will be triggered again when that state is reached
    - Let the northbound system call the actions + create method
    - Let the northbound system pass in the data rather then NSO calculating it
- Split a “large” service into smaller services
  - Like touching many devices
  - To achieve small diff sets



Brown Networks

Brown deployments

Many legacy system: The network is the single source of truth

Provisioning scripts read network state to

- allocate resources, make sure it isn’t in use already
- performs pre-checks, decide which config to create
- avoid overwriting local modifications

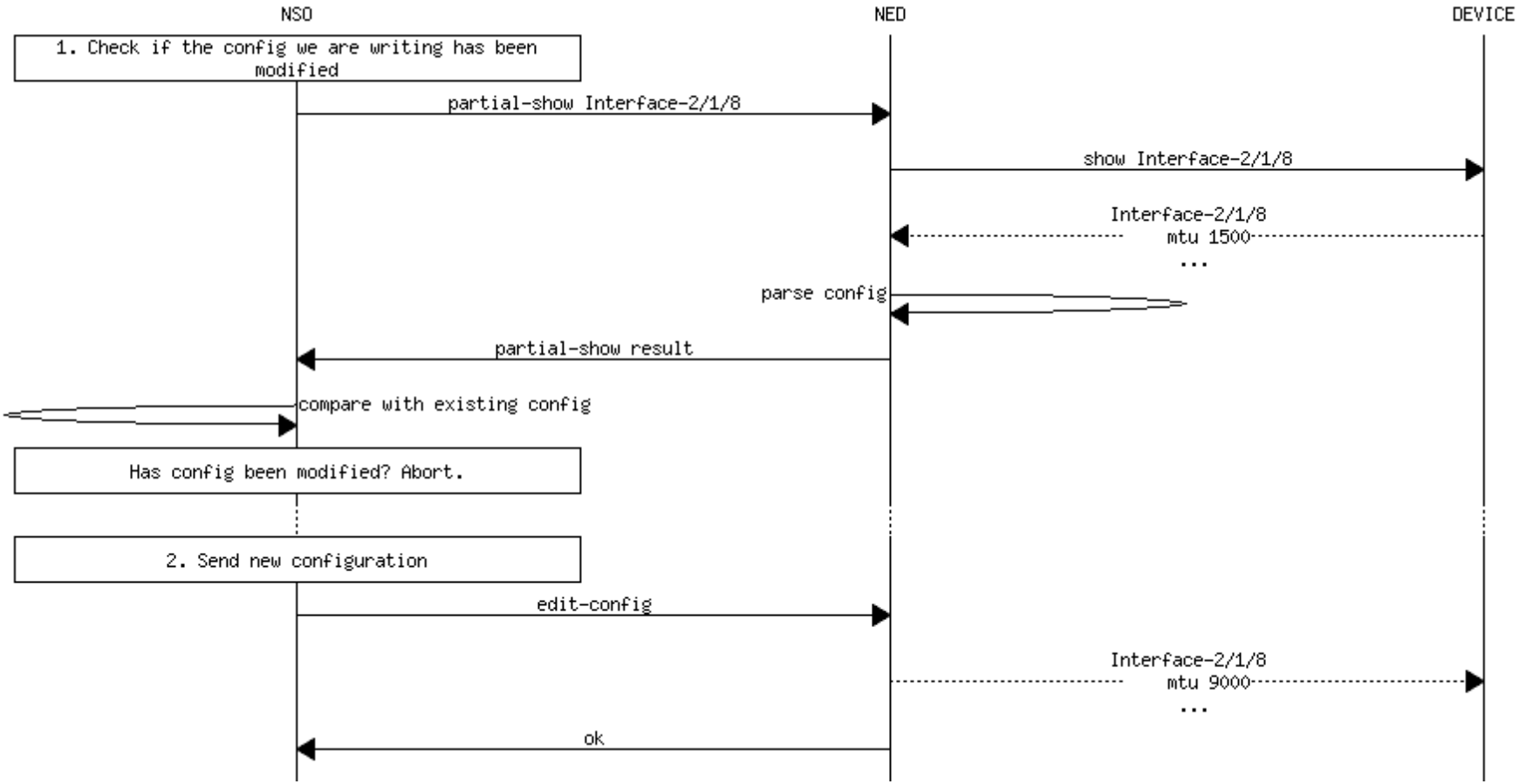
When moving to NSO, this should not translate to frequent sync-from!

Managing brown devices

Devices will be out-of-sync!
Intended

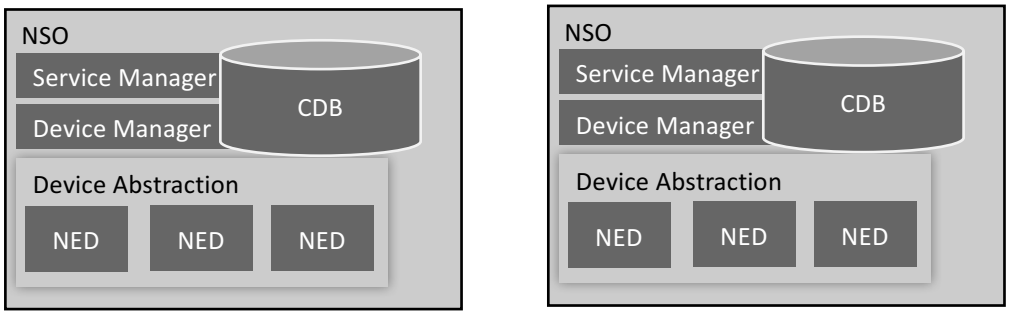
1. Read state of network using modelled stats, or exec commands
2. No syncing of device configuration in the create method
   - Do **not** sync-from in the create method
   - Use *partial* sync-from before invoking service
   - *Only for data that the service **reads***
     - **Not** data that the service **writes**
     - Like allocated VLANs
3. Commit with *no-overwrite*,
   - Checks only device configuration that corresponds to the service, not the complete device configuration.
   - Configure per transaction, device profiles, or per device
4. Partial sync-from and **Service** check-sync to detect service changes

Overall flow: no-overwrite



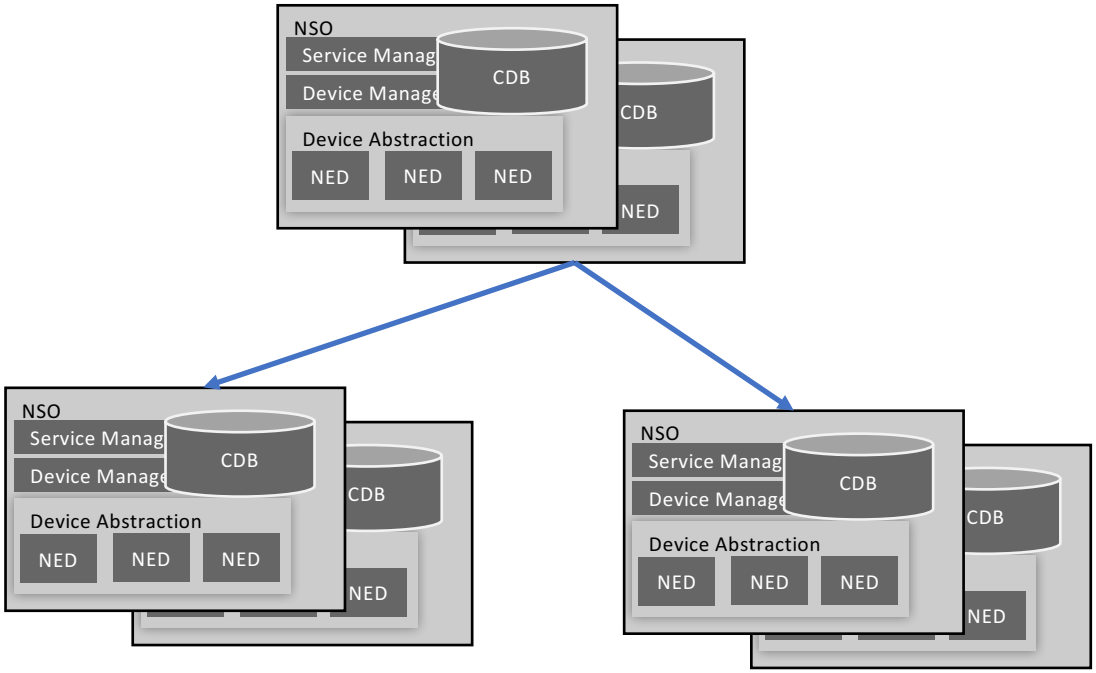
Deployement Models

One System, HA-Cluster



- Commit queue
  - Throughput
- Big machine, lots of RAM
  - Large networks
- Easy to manage
- No NSO-NSO communication penalty

LSA Cluster with Commit Queue



- HA for each cluster node
- Commit queue in all nodes
- Scale
  - Devices partitioned in lower NSO nodes
- Througput
  - Parallellism
- Response-time
  - Commit queue in top node