



# Perhaps more about NSO than you wanted to know

With focus on performance

Sebastian Strollo

Principal Engineer

NSO Developer Connect, November 14, 2017

# Agenda

- Introduction
- NSO under the hood: the transactional model
  - *Deeper understanding of an NSO transaction*
- Little knobs
  - *Things to keep in mind while developing*
- Big knobs
  - *Fundamental choices for your NSO deployment*

# Agenda – NSO Under the Hood

*Deeper understanding of an NSO transaction*

- The transactional model
- The Cost of Communicating with devices

# Agenda – Small Knobs

*Things to keep in mind while developing*

- NED Settings
- Service Code
- NSO Host System
- NSO Configuration

# Agenda – Big Knobs

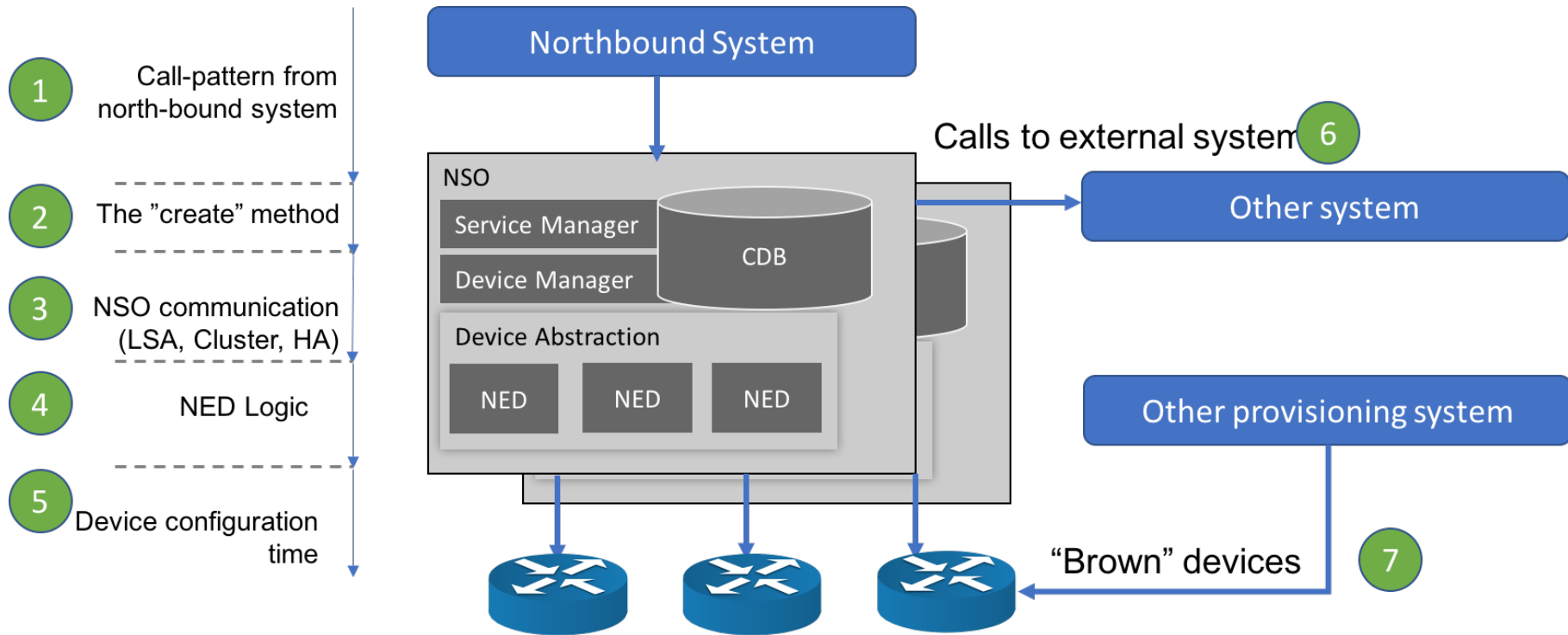
*Fundamental choices for your NSO deployment*

- OOB Model – rethinking sync
- The Commit Queue
- LSA Cluster and Device Cluster
- HA Model

# Introduction

# Performance 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



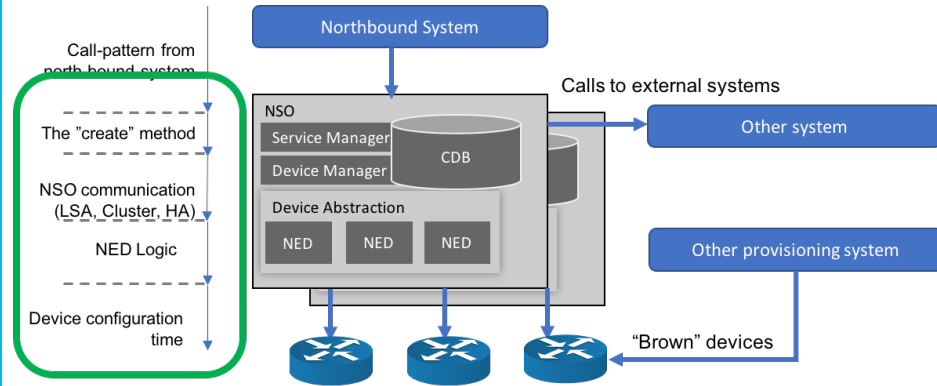
# NSO Under the Hood

# Agenda – NSO Under the Hood

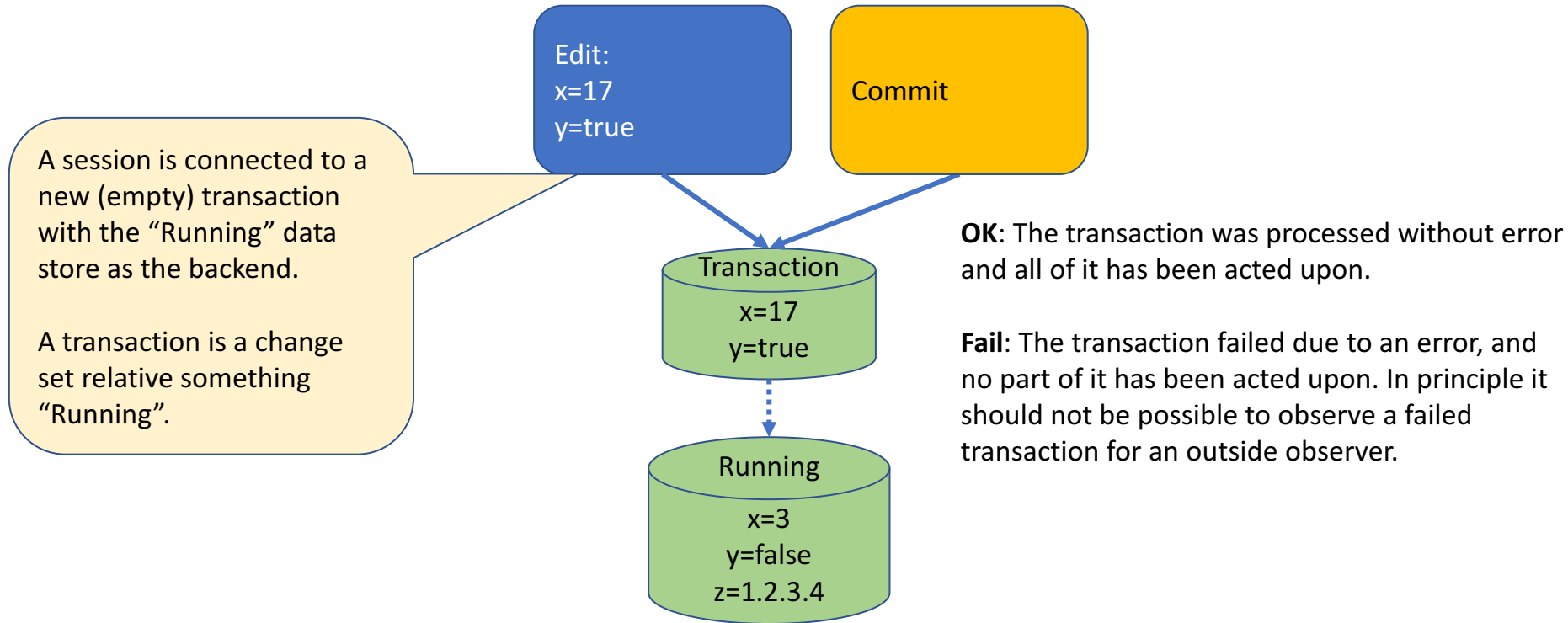
*Deeper understanding of an NSO transaction*

- The transactional model
- The Cost of Communicating with devices

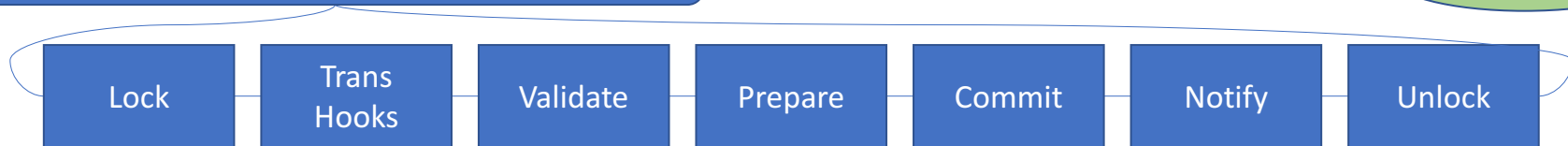
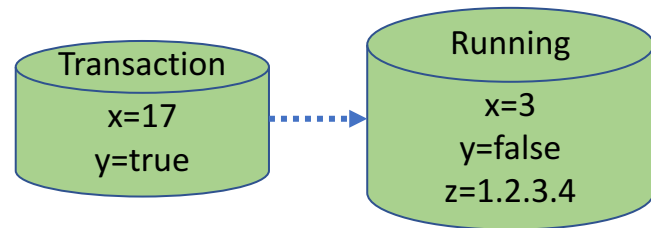
# The NSO Transactional model



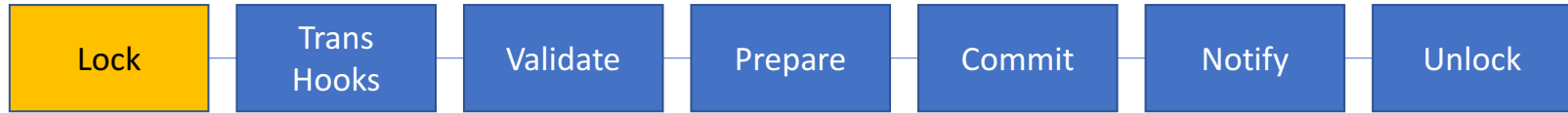
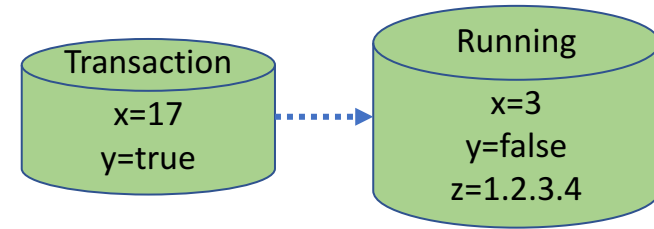
# The topic: What happens at “commit”?



# A closer look at the Commit Sequence



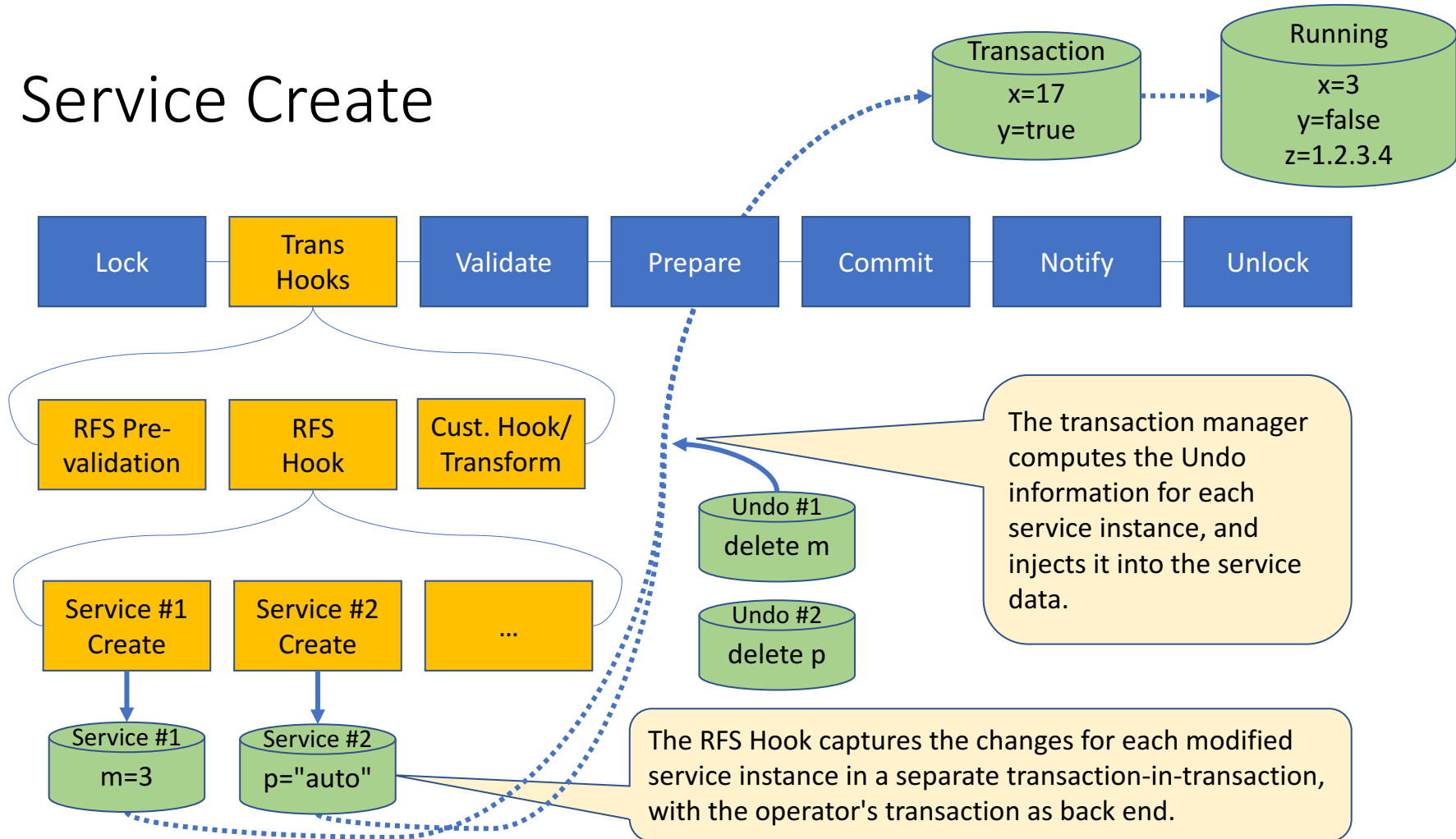
# Lock Running



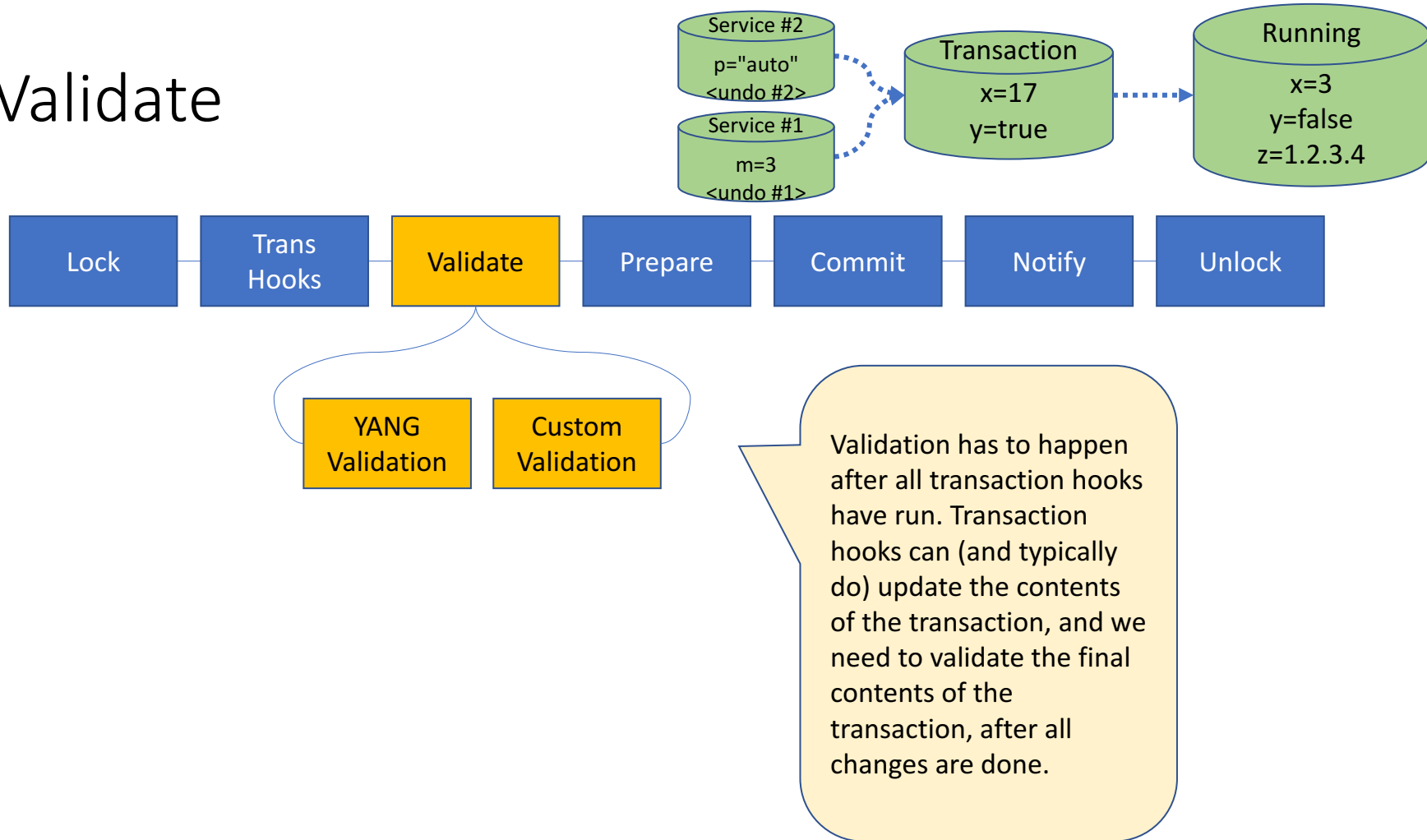
Locking, so that no other changes are under way while we are processing this transaction, is key to a simple programming model.

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

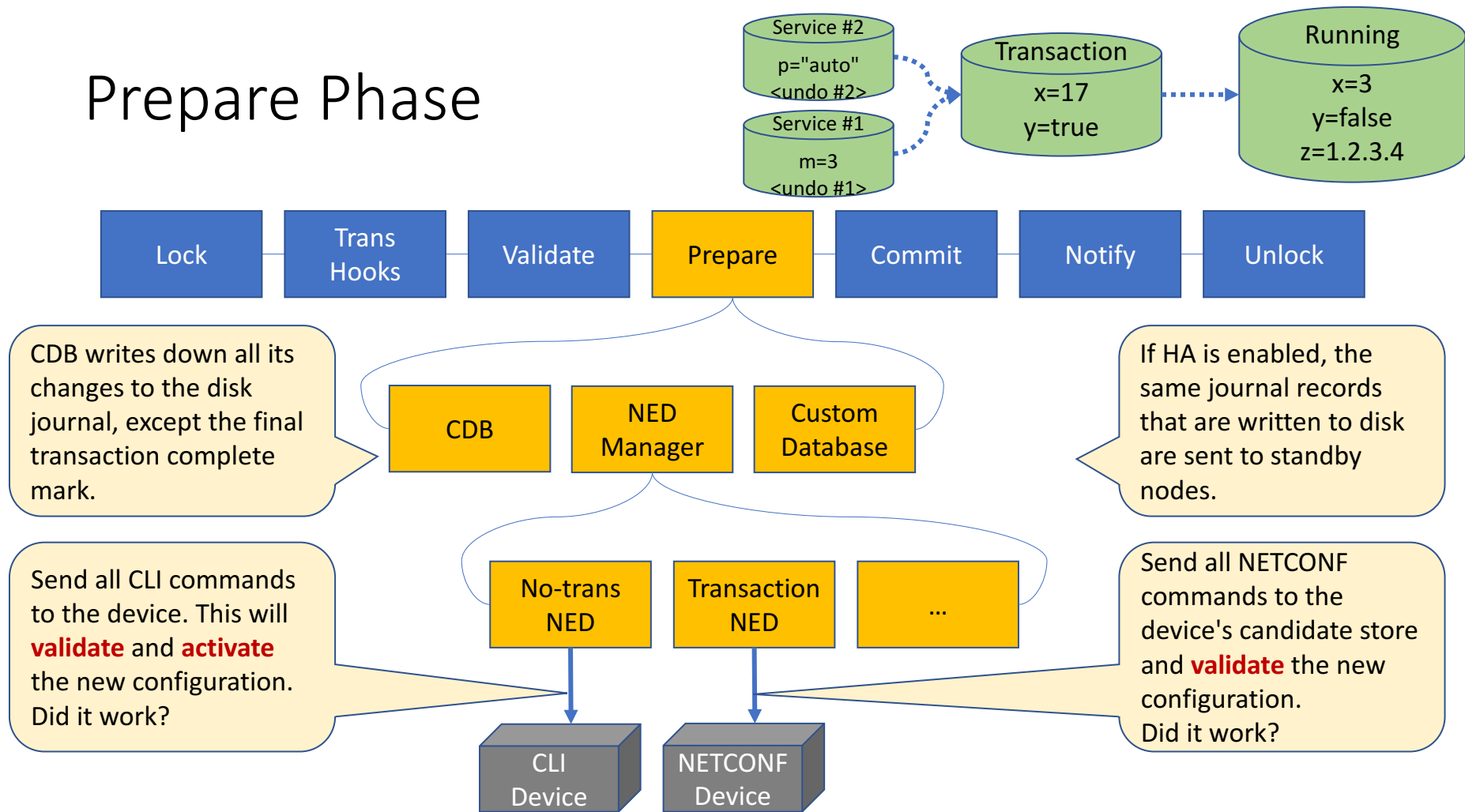
# Service Create



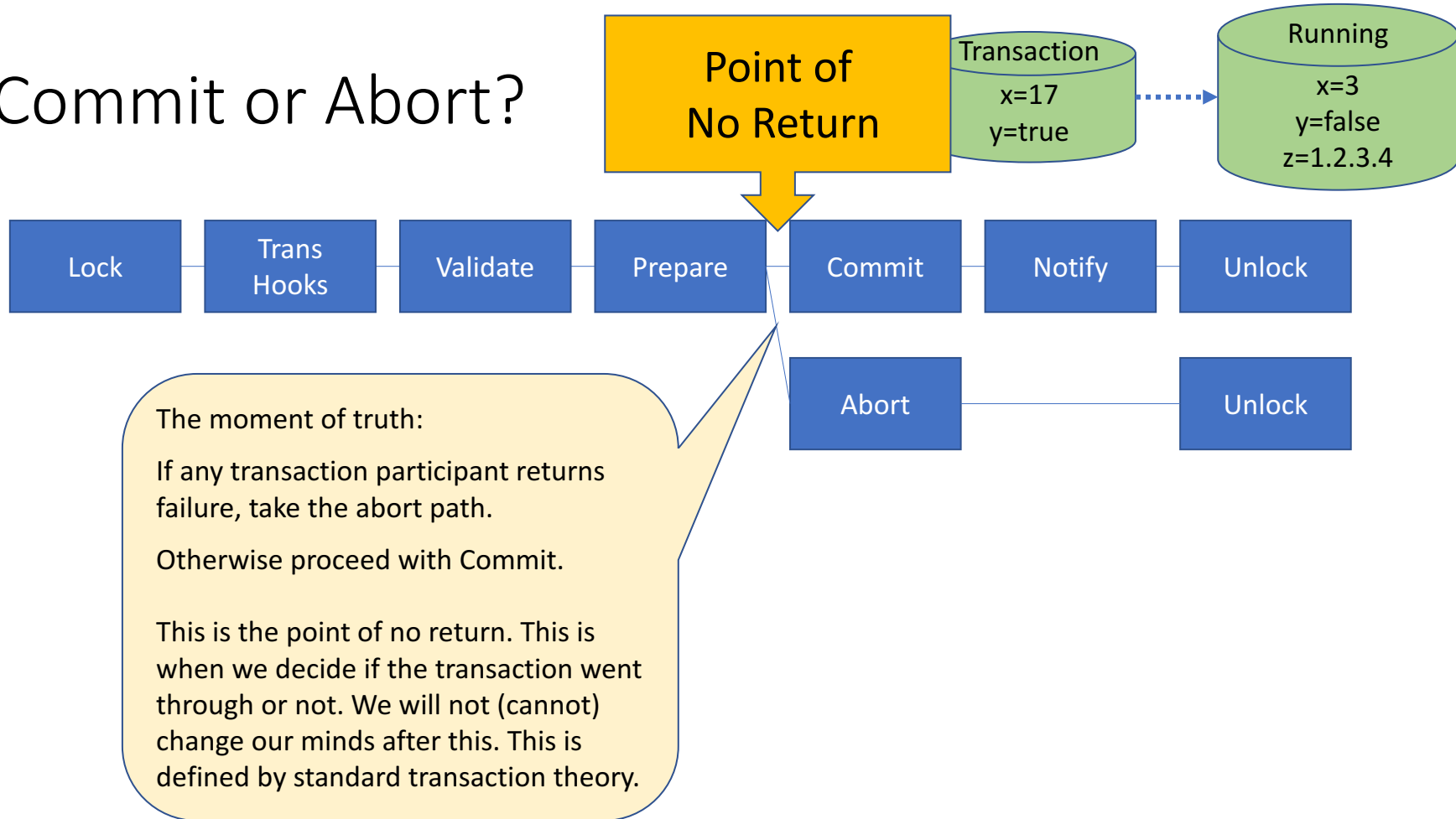
# Validate



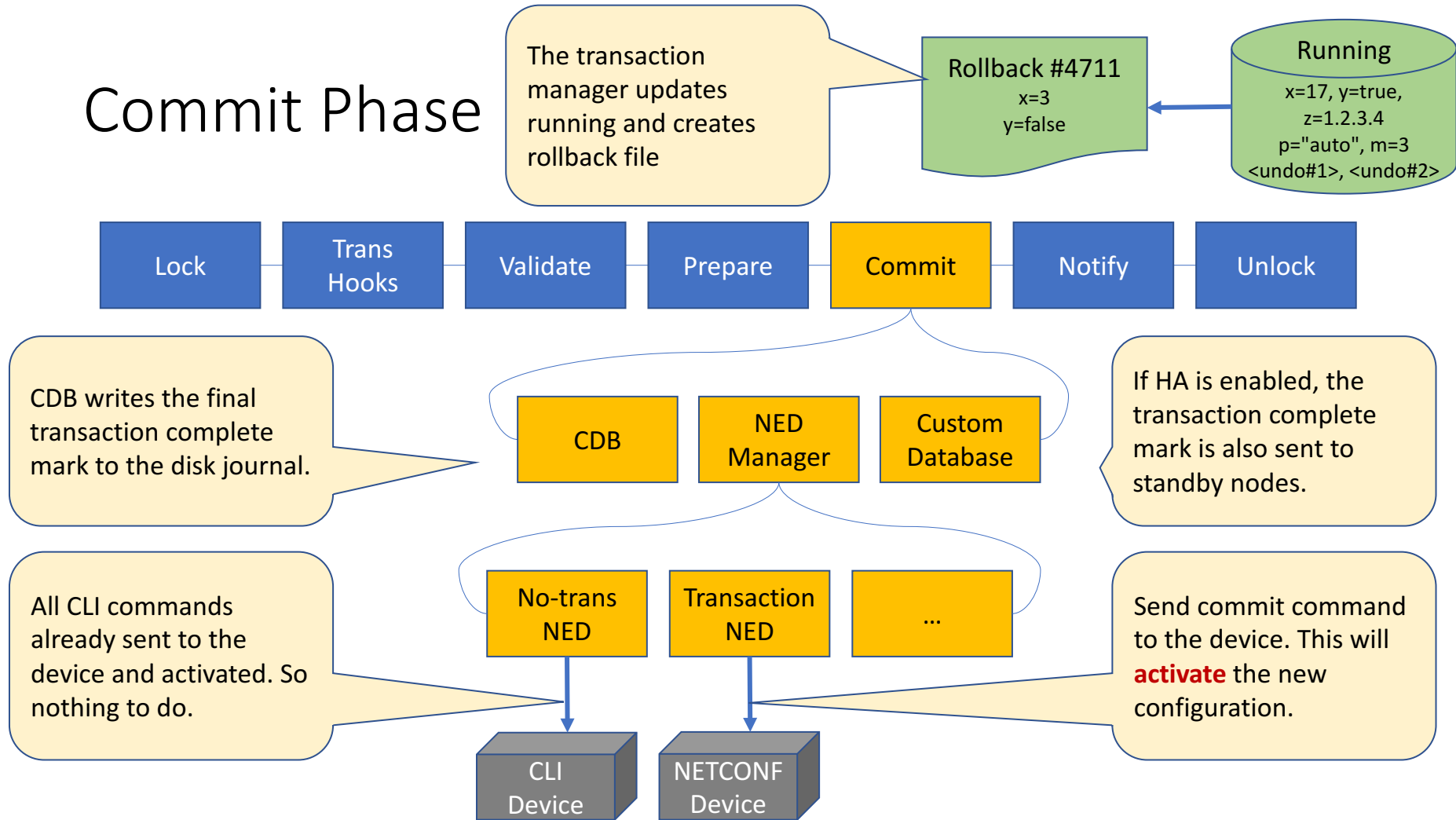
# Prepare Phase



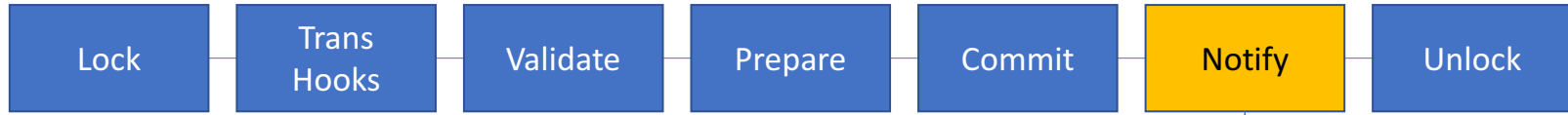
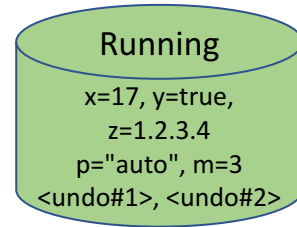
# Commit or Abort?



# Commit Phase

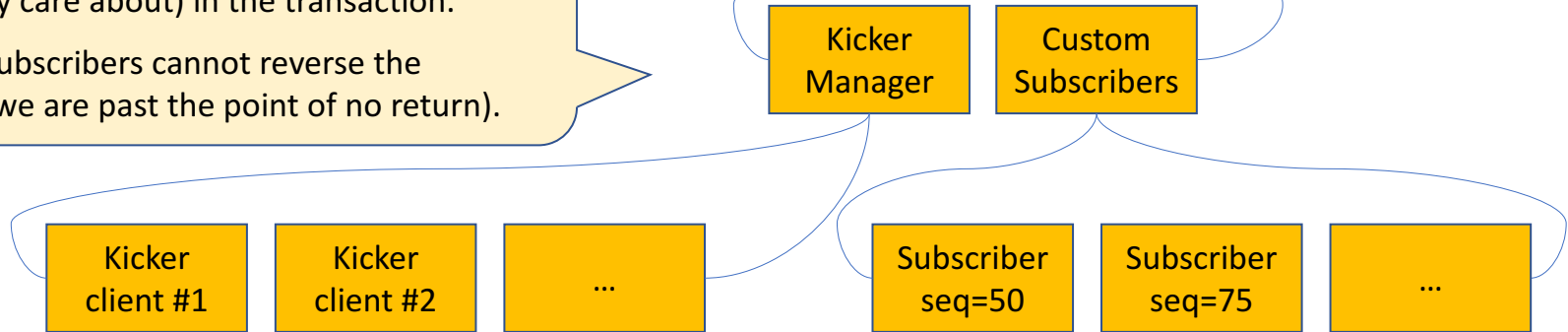


# Notify

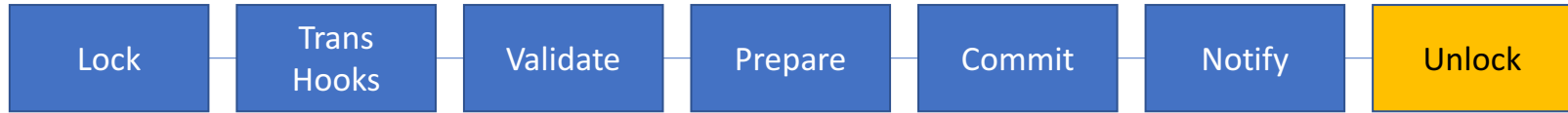
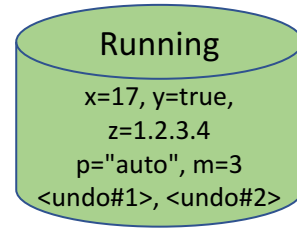


Kickers and subscribers are informed about the changes (they care about) in the transaction.

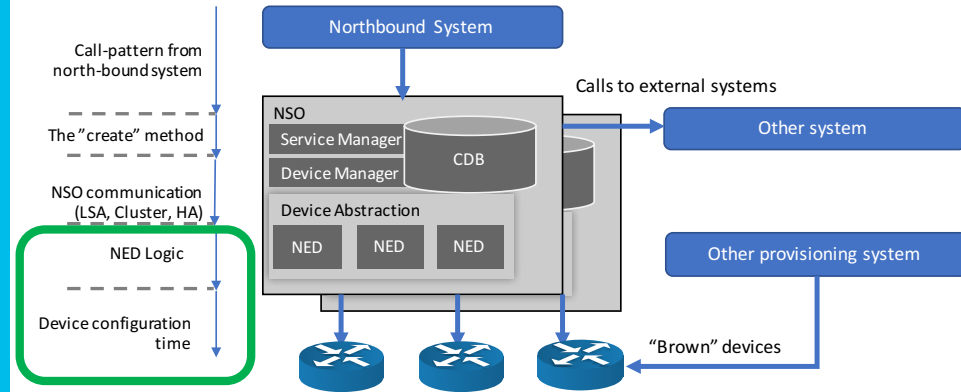
Kickers and subscribers cannot reverse the transaction (we are past the point of no return).



# Unlock Running



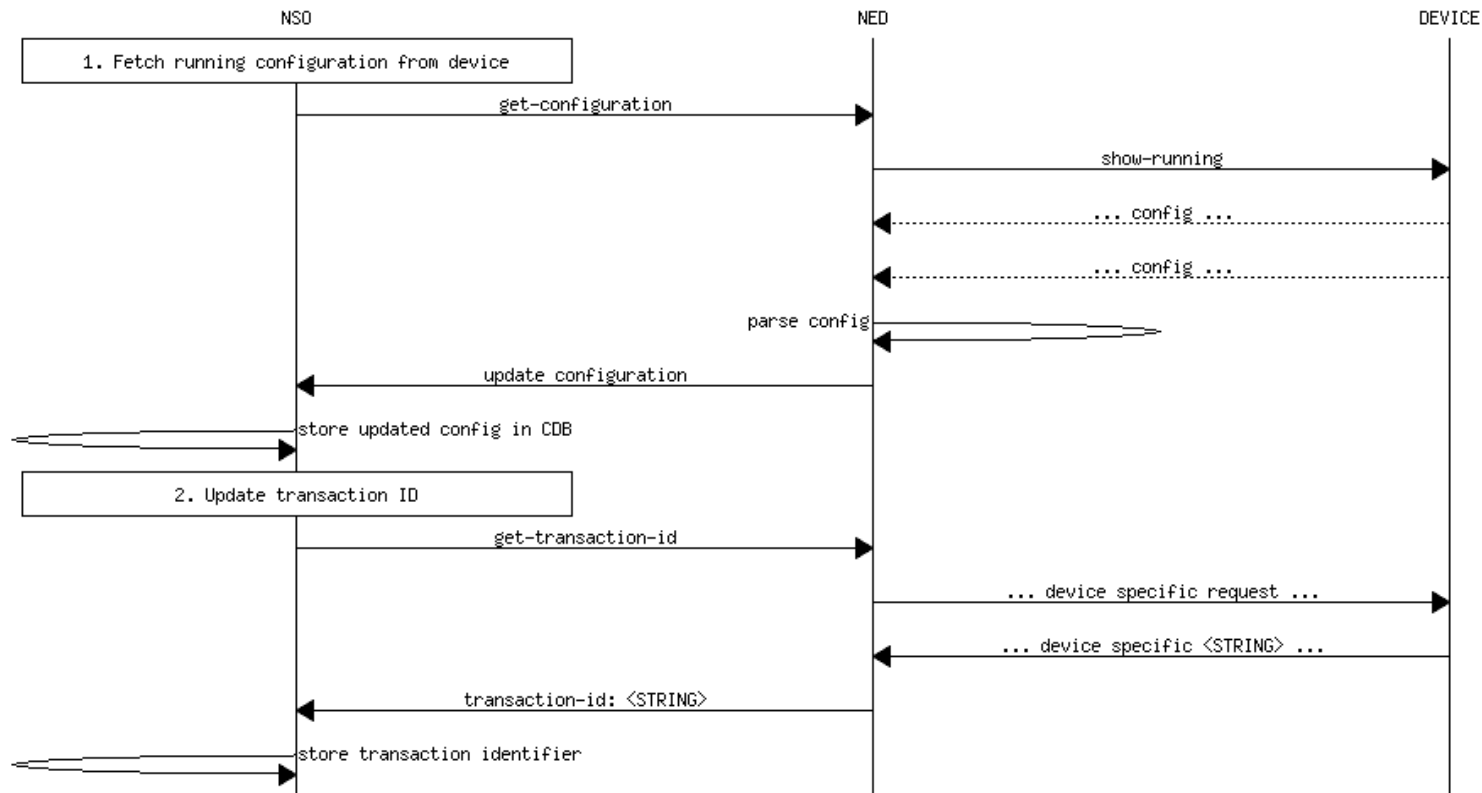
# Understanding the cost of device communication



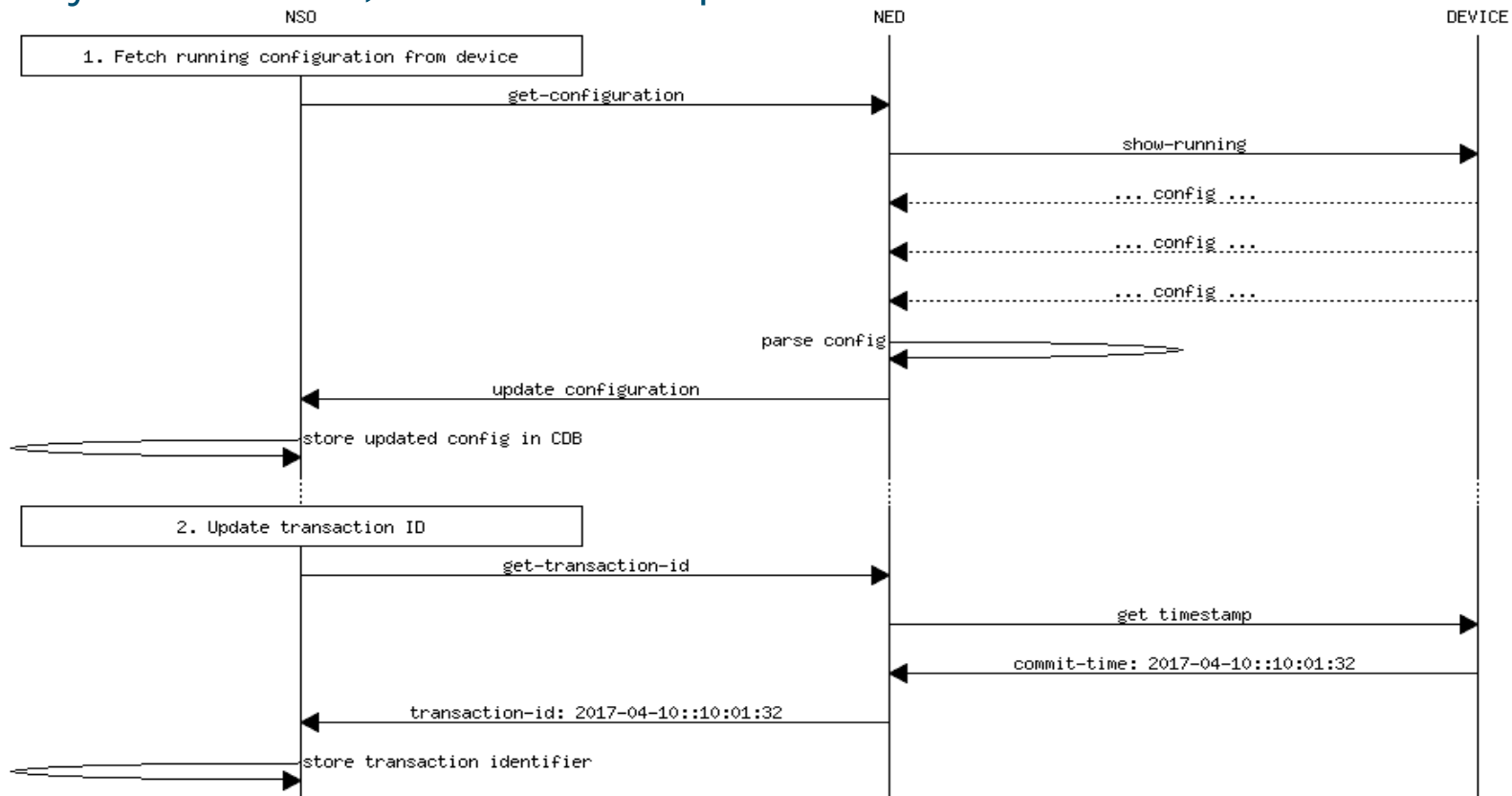
# How does NSO know if a device is in sync?

- NSO stores a *transaction identifier* for every device
- The trans-id is updated every time a transaction is performed towards that device and on sync-from
- The trans-id is opaque to NSO – it is up to the NED to produce it
- Some devices have native support (e.g. a commit timestamp)
- *Default for a CLI NED is to compute a checksum of “show running”*
- Many NED's have specific ned-settings to use a different algorithm

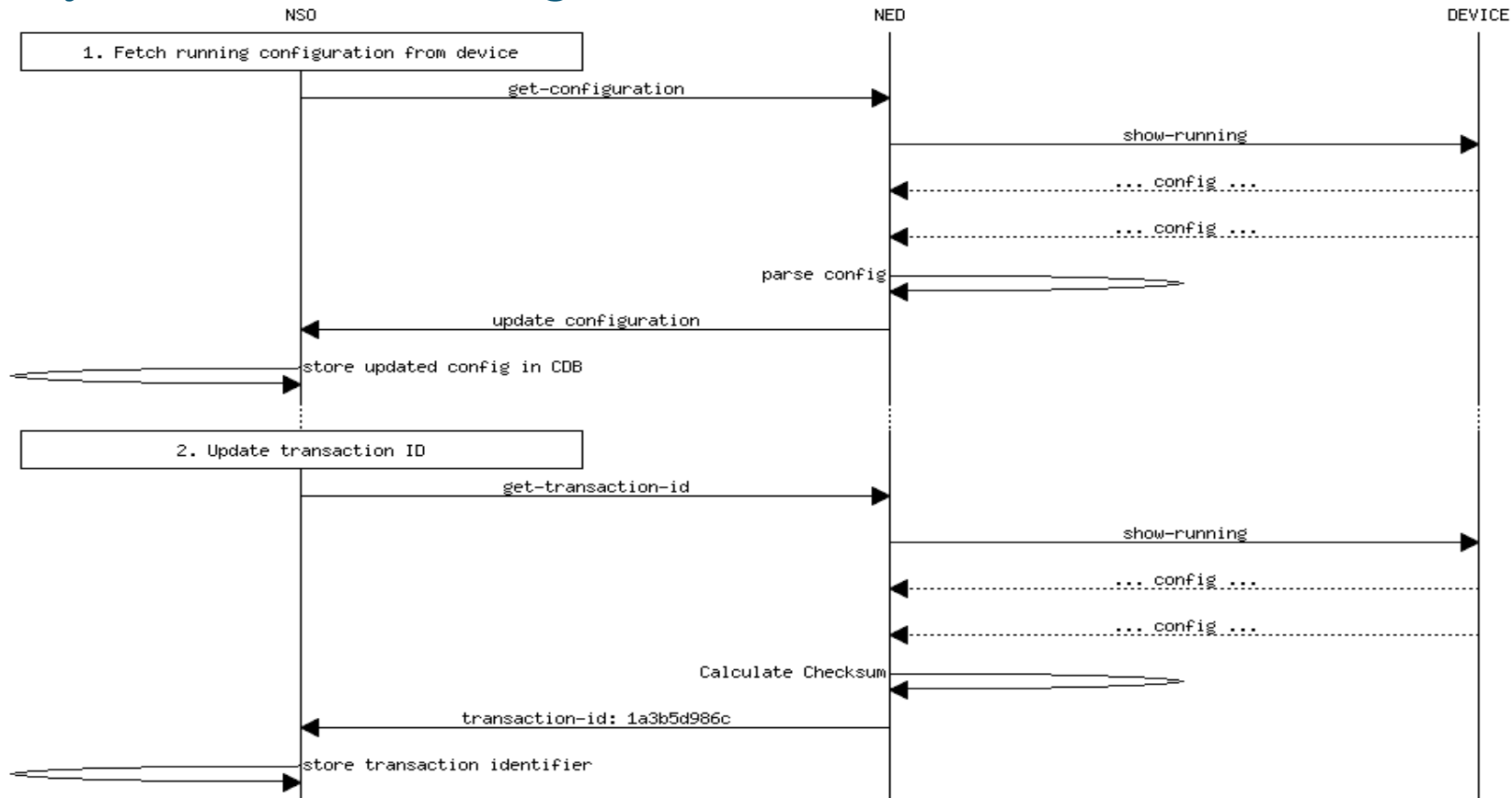
# Anatomy of “sync-from”



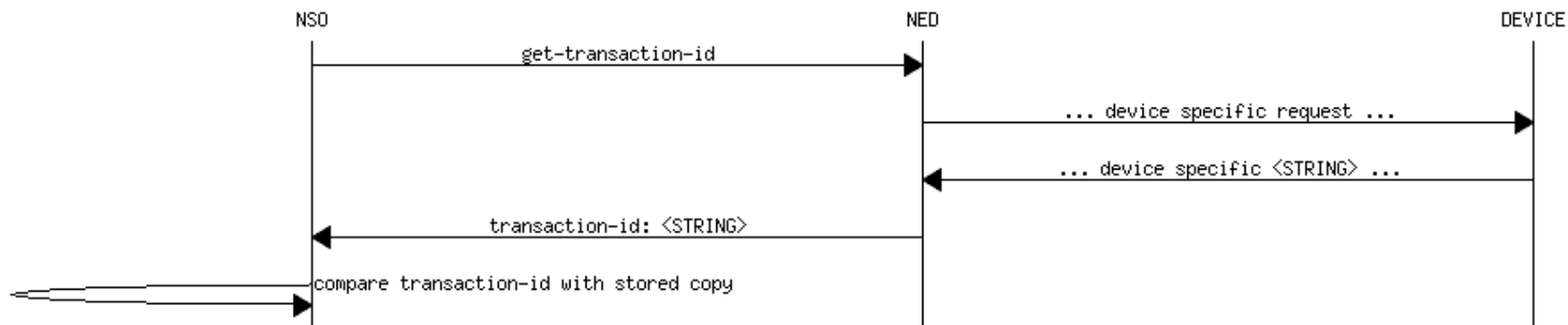
# sync-from, timestamp as transaction identifier



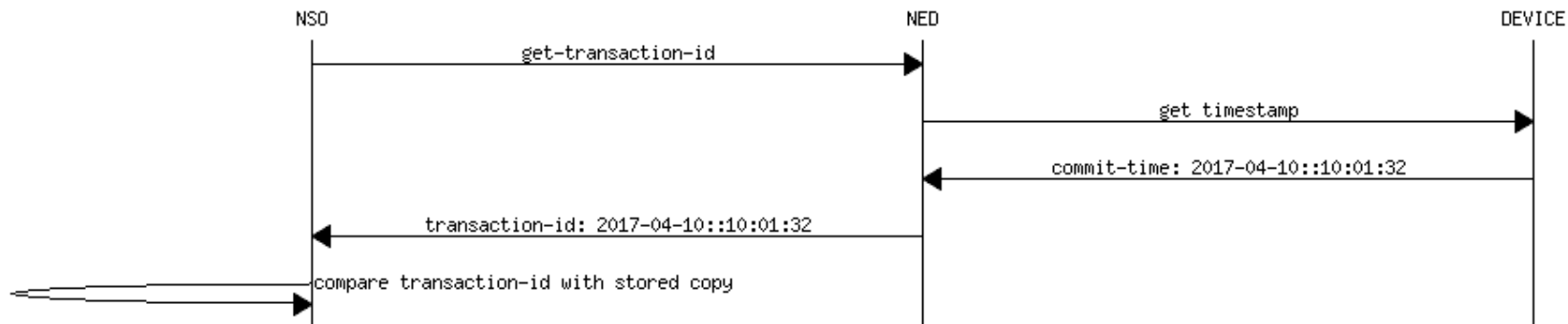
# sync-from, config-hash as transaction identifier



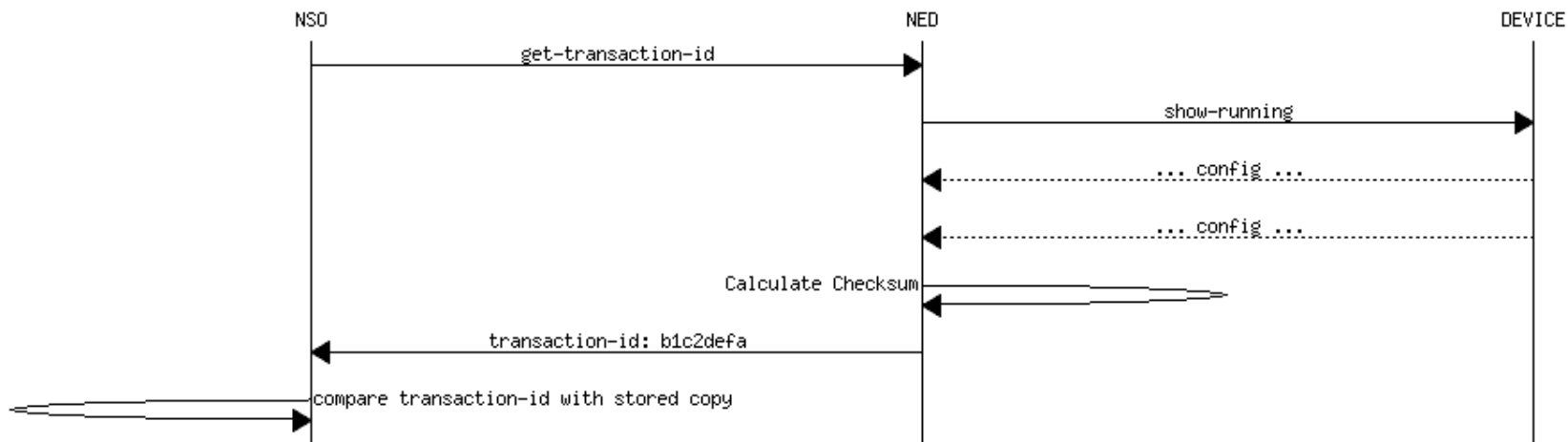
# Anatomy of “check-sync”



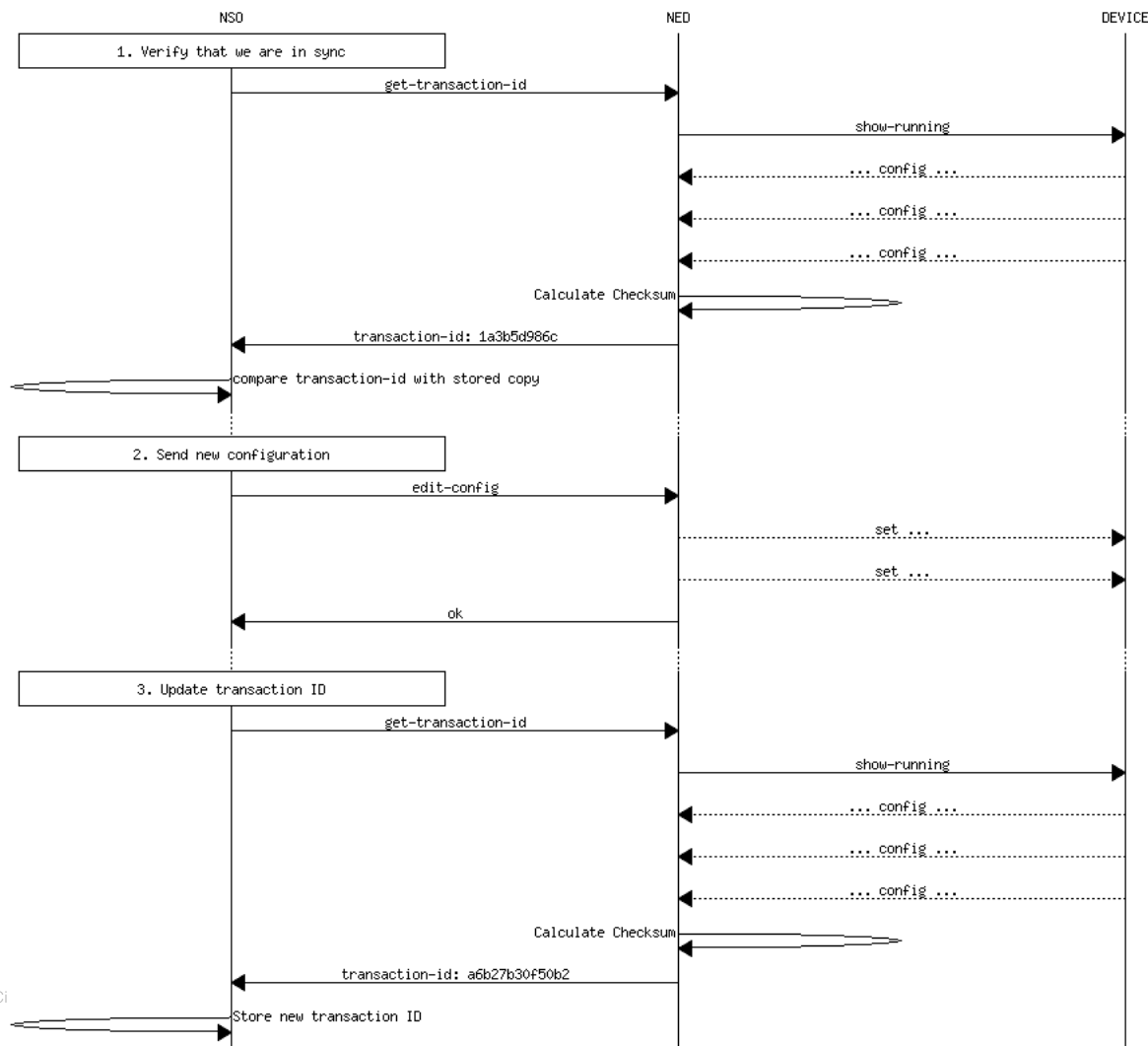
# check-sync, timestamp as transaction identifier



# check-sync, config-hash as transaction identifier



# Regular commit sequence



Small Knobs

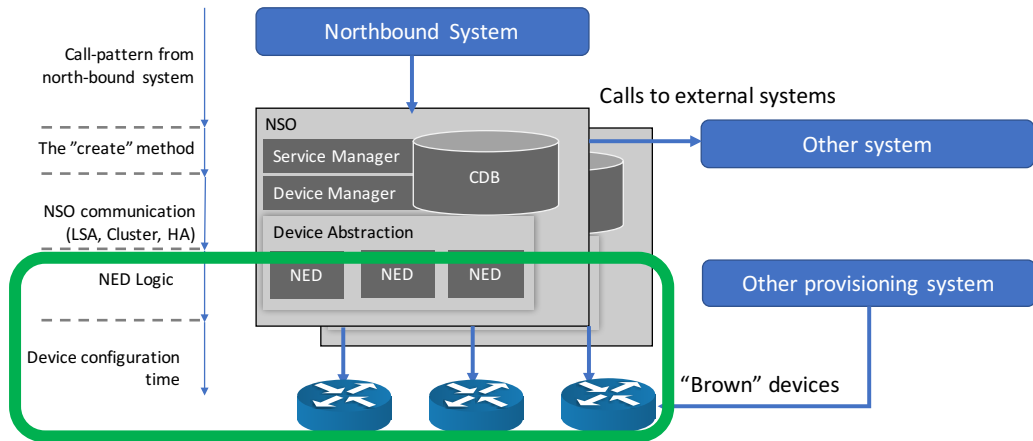
# Agenda – Small Knobs

*Things to keep in mind while developing*

- NED Settings
- Service Code – `create()`
- NSO Host System
- NSO Configuration

## NED Settings (and *device* behavior)

- NEDs can be configured for different check-sync mechanisms
    - Hash or Transaction id
    - Data transfer method (scp/sftp or “show config”)
  - Device behavior
    - The time for a device to apply the configuration can differ by configuration options
    - Current device load
    - Device HA
    - Commit script on the device
- 
- The diagram illustrates the configuration flow from a Northbound System to Network Elements (NEDs) via an NSO. The flow includes a call-pattern from the north-bound system, the 'create' method, NSO communication (LSA, Cluster, HA), NED Logic, and finally Device configuration time.



# NED settings, transaction id methods

	alu-sr	cisco-ios	cisco-staros	juniper-junos
Default	config-hash	config-hash	config-hash	commit-time
Other available methods	rollback-timestamp, last-modified-timestamp, last-saved-timestamp	last-config-change, config-id, ..., config-hash-cached	none	n/a

Some examples, *a/ways* check NED's current **README** and **src/yang/\*-meta.yang**

# NED settings, more options

	alu-sr	cisco-ios	cisco-staros	juniper-junos
“write-mem”	Can disable (admin save)	Command configurable	Can disable	n/a
“Turbo mode” (experimental)	yes	yes	yes	n/a
Other settings	sftp config, ...	...lots...	...	use-private- candidate

Some examples, *a/ways* check NED’s current **README** and **src/yang/\*-meta.yang**

# Viewing current NED settings

```
ncs> show devices device active-settings
```

*and* NED trace file

```
ncs> configure
```

```
ncs% set devices device NAME trace pretty
```

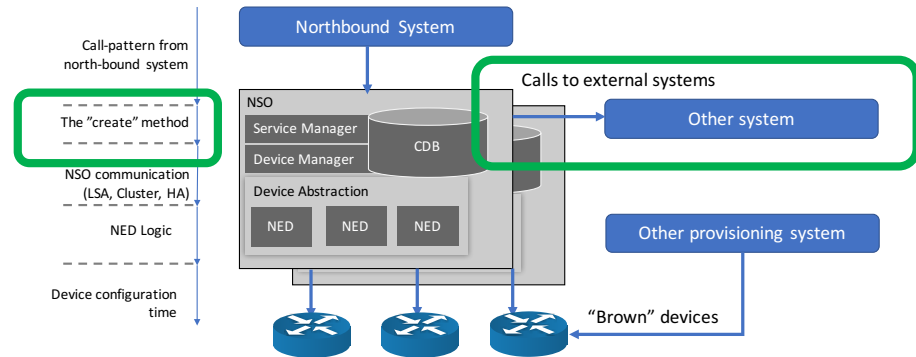
```
ncs% commit
```

```
ncs% exit
```


```
ncs> request devices device NAME disconnect
```

```
ncs> request devices device NAME connect
```

# Service Code



# The create() Method

- 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

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

    try {
        createNCCAccount(pad);
    } catch (Exception e) {
        LOGGER.error("L3vpn create failed: " + e.getMessage());
    }
}
```



# The create() Method

- “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 than NSO calculating it
- Split a “large” service into smaller services
  - Like touching many devices
  - To achieve small diff sets

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

    try {
        createNCCAccount(pad);
    } catch (Exception e);
}
```



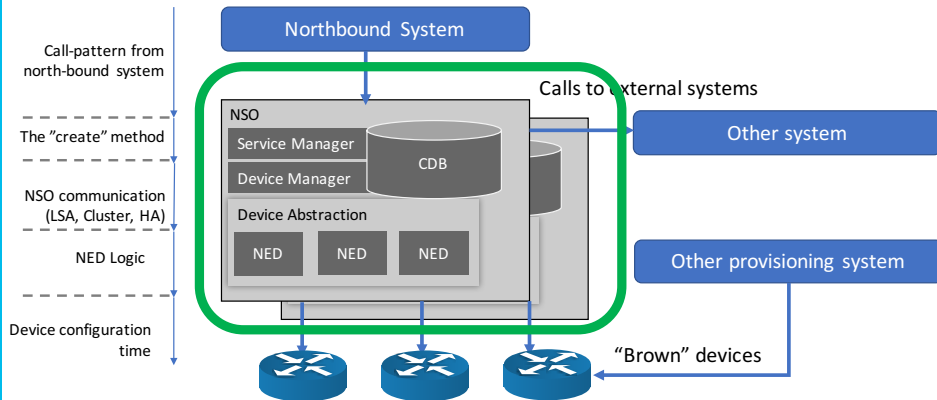
# NSO Host System

The host machine characteristics has huge impact

- CPU
- Memory
- Local Disc
- Virtual or bare metal

Develop and *test* on relevant machine

# NSO Configuration



# NSO Configuration, impact on performance

Study different configuration options for NSO

- *ncs.conf*
- Examples
  - Trace files
  - Logging
  - Rollbacks
  - Diffs
  - Pre populate snap-shot db
  - ...

Big Knobs

# Agenda – Big Knobs

*Fundamental choices for your NSO deployment*

- OOB Model – rethinking sync
- The Commit Queue
- LSA Cluster and Device Cluster
- HA Model

# OOB Model – Rethinking ‘Sync’

# NSO 101

- First: add your devices to NSO
- Second:

`request devices fetch-ssh-host-keys`

`request devices sync-from`

- ...Time passes, things happen to your network, and then...

# NSO 101, Continued

```
admin@ncs% set ...  
admin@ncs% set ...
```

```
admin@ncs% commit
```

```
Aborted: Network Element Driver: device ALBQ35JSC70:  
out of sync  
[error]
```

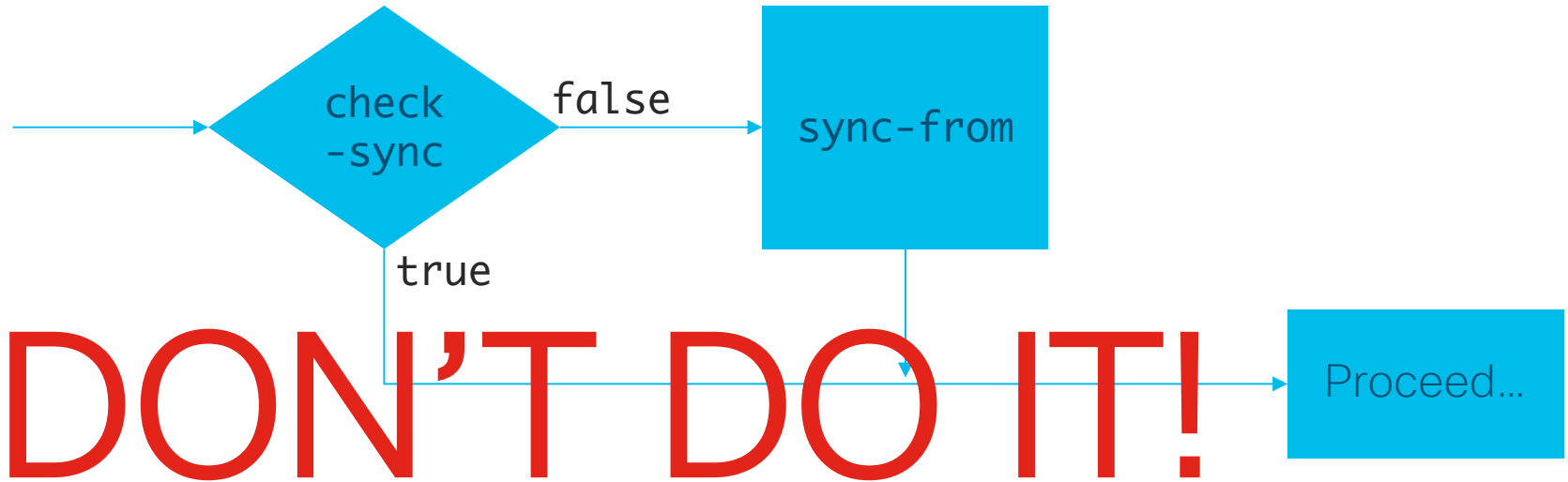
- Now what do you do?

# NSO 101, Continued

- You know the answer:

`request devices device ALBQ35JSC70 sync-from`

# A Common Pattern



DON'T DO IT –  
at least not until you have  
considered the alternatives

# What Environment Are You In? (\*)

1. No Out-Of-Band changes allowed
2. Some Out-Of-Band changes, but always unrelated to NSO service
3. A lot of OOB changes that are out of control (or is part of transition phase)

Are you automating manual tasks or designing for automation from the ground up?

(\*) what environment do you *want* to be in?

# No OOB Changes

- "Standard" NSO
- Being out-of-sync is an exception

# Unrelated OOB Changes

- Let go of the “always in sync” concept
- Set use-transaction-id to false, or out-of-sync-commit-behavior to accept

# Out of Control OOB Changes

- Let go of the “always in sync” concept
- Use “partial-sync-from” (NEW in 4.4.2)
- Use “service check-sync”
- Use “commit no-overwrite”

# More on out of control OOB changes

1. Syncing device configuration
  - **NEVER** sync-from in the create method
  - Use *partial* sync-from
  - Only for data that the service **reads**
  - Not data that the service writes
  - Example: allocated VLANs
2. Service check-sync
3. Commit with no-overwrite,
  - Checks only device configuration that corresponds to the service, not the complete device configuration.
  - Configurable per transaction, device profiles, or per device

# Out of control OOB changes + network as inventory

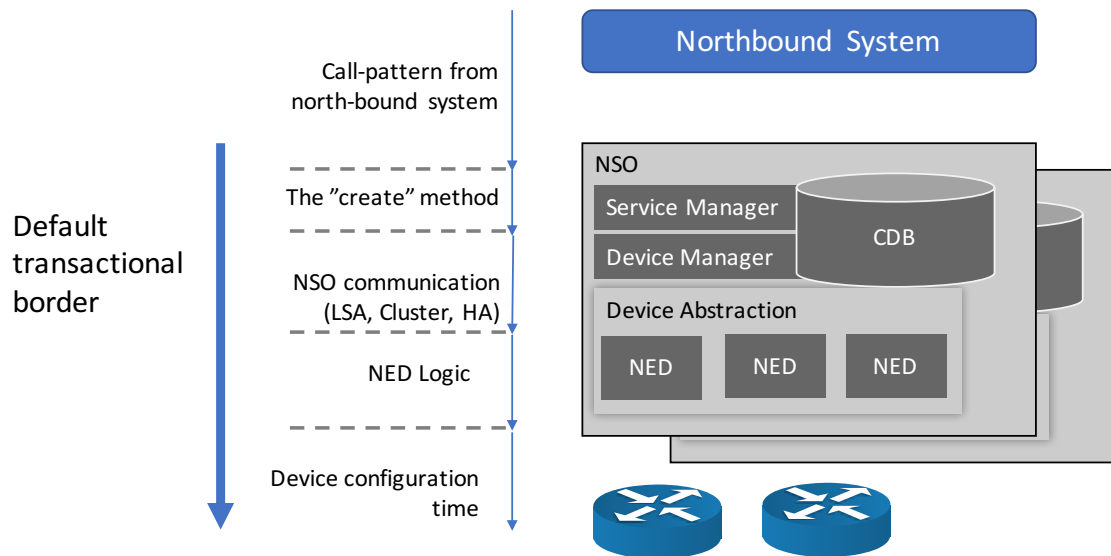
- Legacy system: The network is the single source of truth
- Provisioning scripts read network state to
  - allocate resources, make sure it isn't in use already
  - performs pre-checks, decide which config to create
  - avoid overwriting local modifications
- When moving to NSO, this should not translate to frequent sync-from!

# Out of control OOB changes + network as inventory

- Instead of frequent full sync-from do:
  - read state of network using modelled stats, or exec commands
  - partial sync-from
  - perform service check-sync to detect service changes
  - use no-overwrite when committing

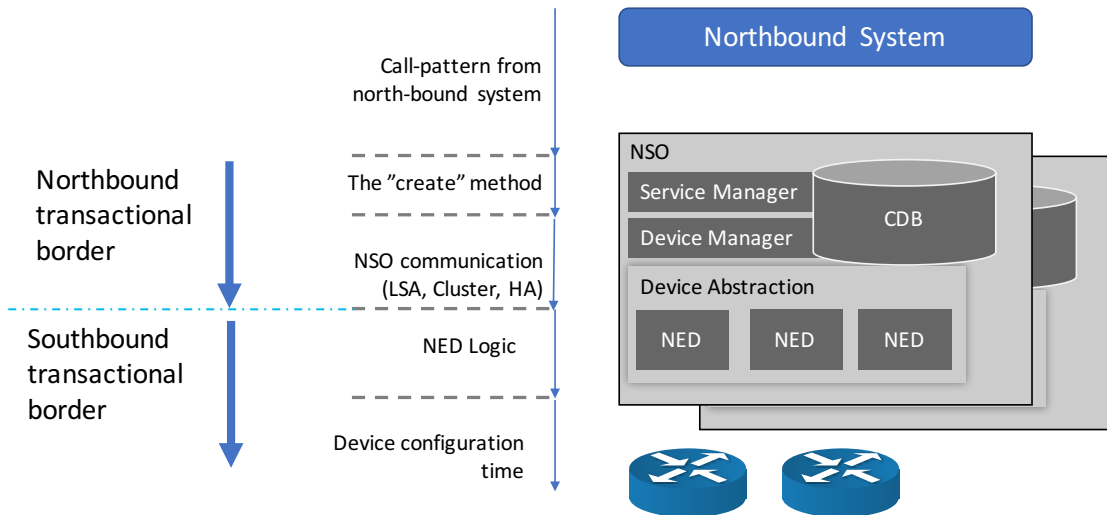
# The Commit Queue

# Default NSO transactional model



NSO stays locked until slowest device is complete  
If transaction fails, config change is backed out of CDB and the devices

# With Commit Queue



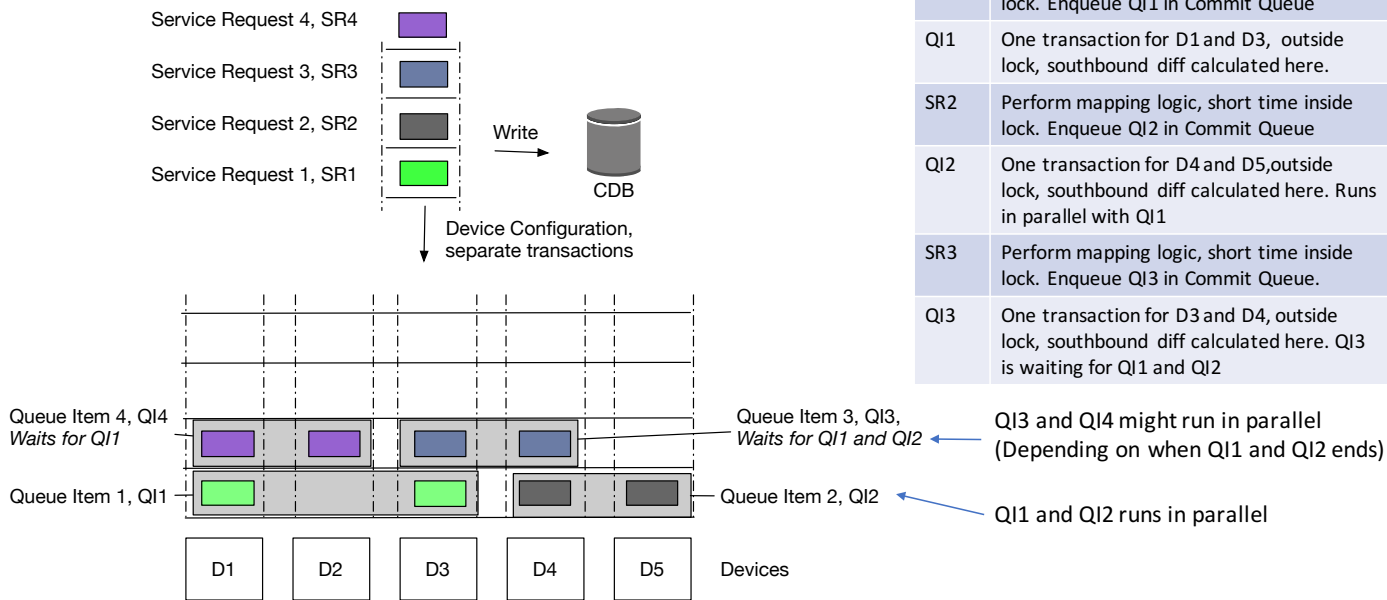
- Throughput ++
- Response time + -
- Scalability 0
- Reliability + -

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

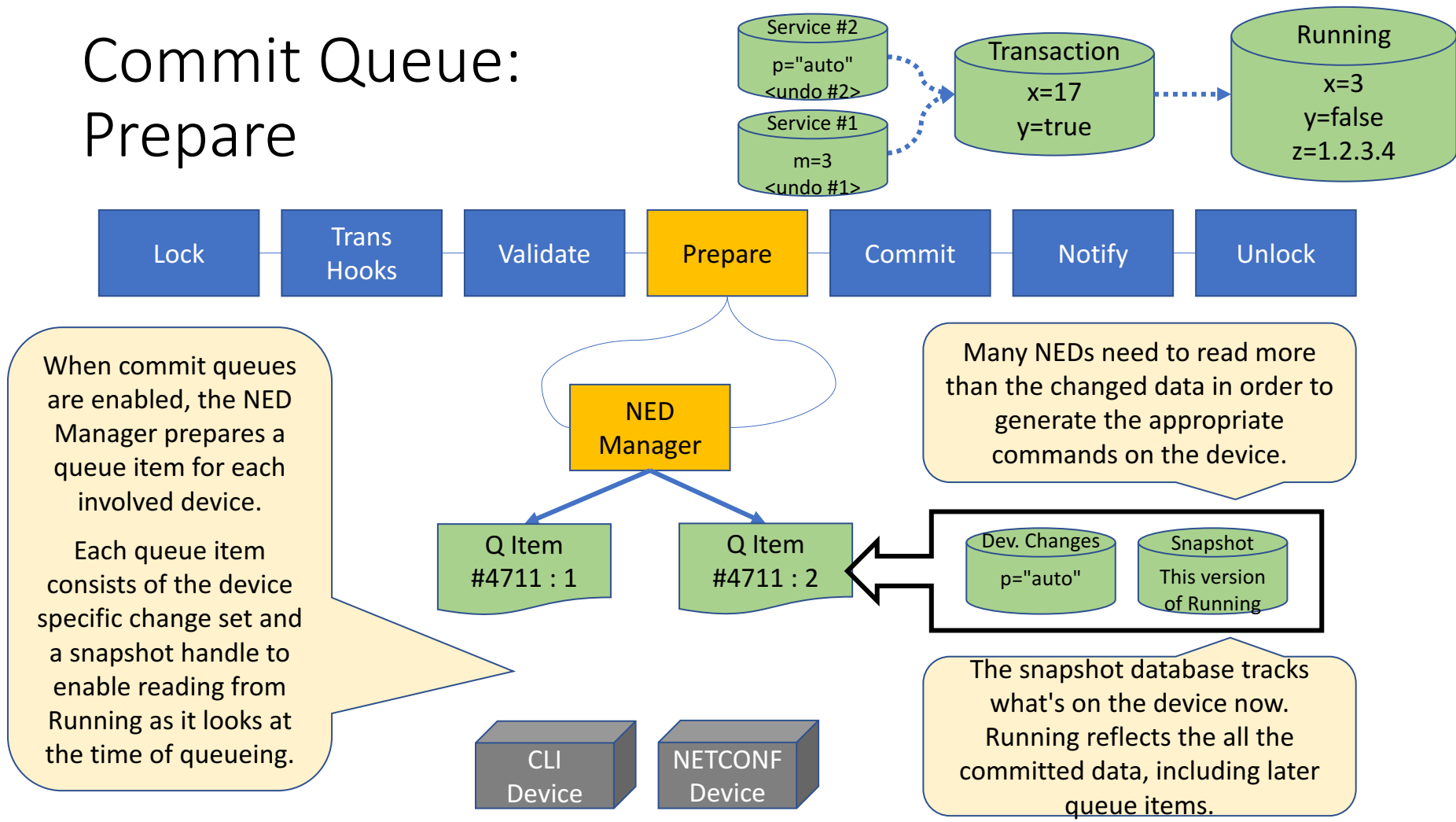
# Commit Queue

- Increased throughput by “eventual consistency”
  - Device communication and “diff calculation” taken out of transaction lock
  - Transactions towards independent sets of devices can run in parallel
- Use NSO 4.4.x ( $x \geq 2$ ) to get the latest improvements:
  - Atomic (all or nothing) behavior even for cq transactions
  - Automatic error recovery
  - Manual rollback of completed items
  - Supported in LSA mode as well
- *Negatives: increases complexity in deployment, error handling is hard, still scant documentation*

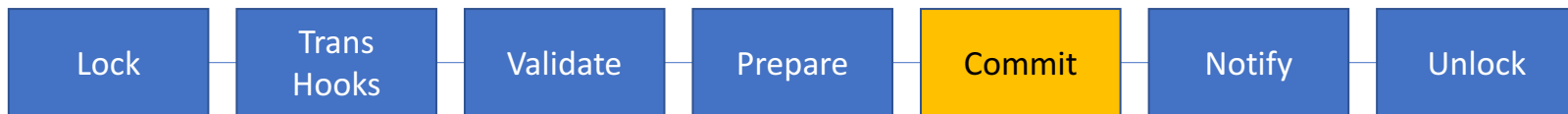
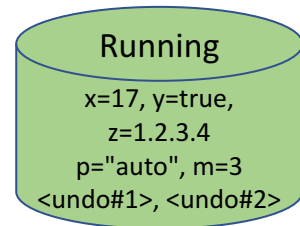
# Commit Queue Example



# Commit Queue: Prepare

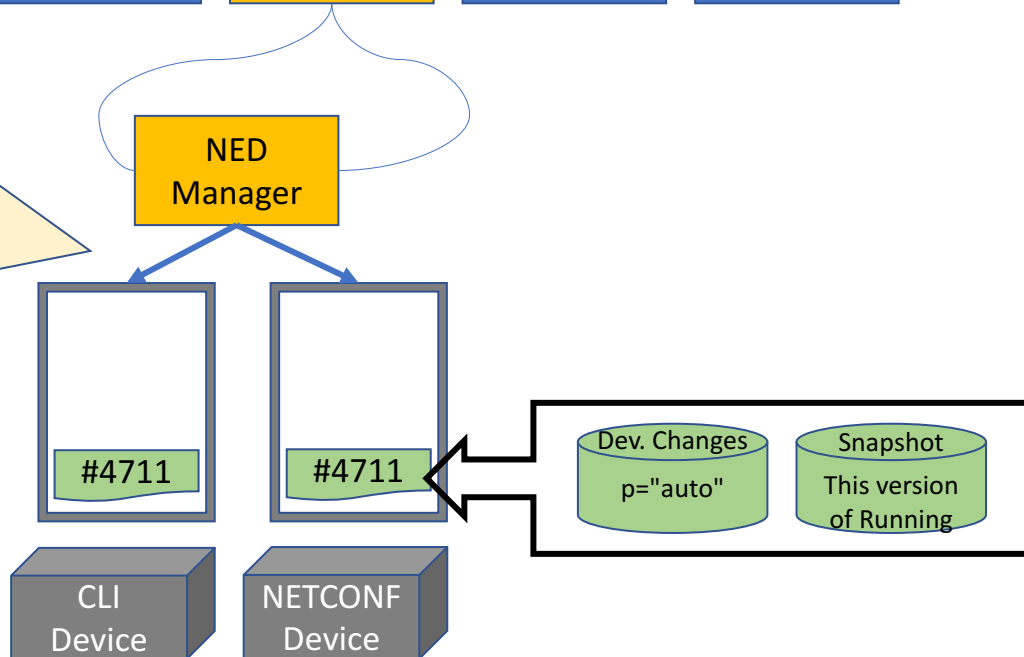


# Commit Queue: Commit

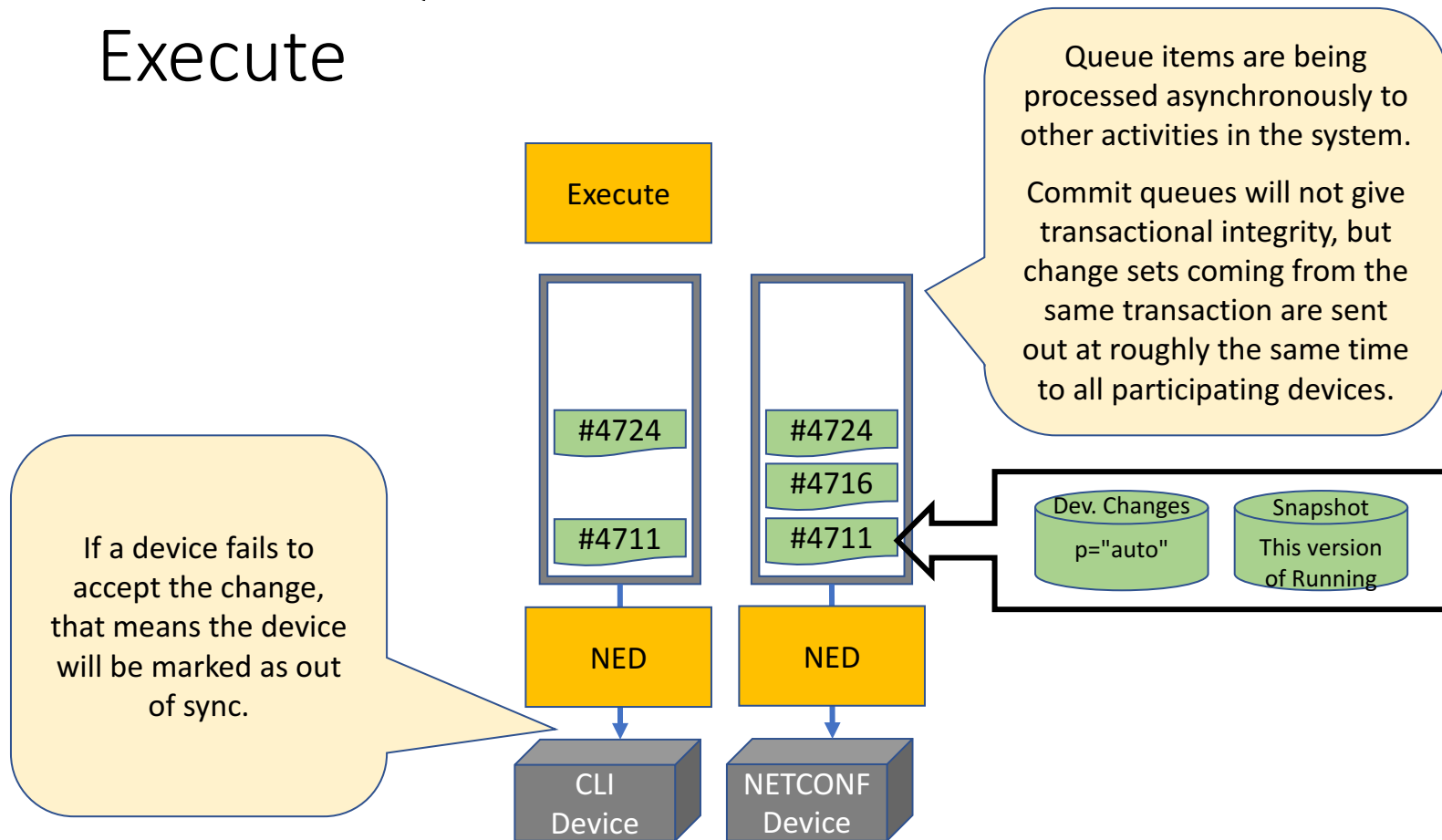


The NED manager places the queue items on the device queues.

Once queue items are placed on the device queues, they are being sent to devices regardless of new transactions being committed, aborted, etc. Each queue item is handed to the respective NED for delivery to the device.

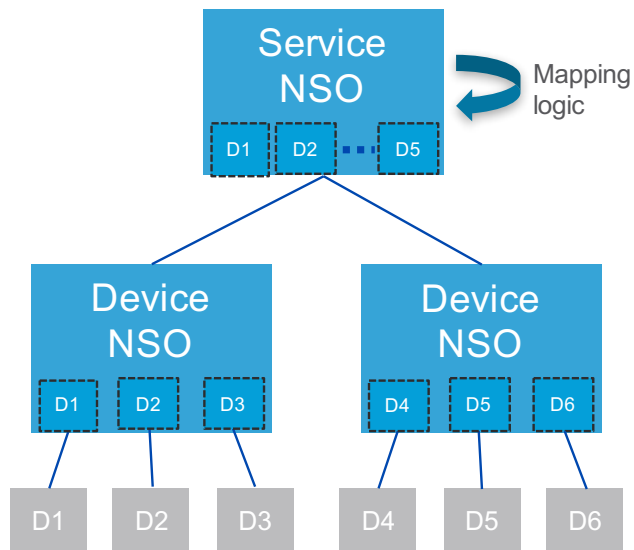


# Commit Queue: Execute



# LSA Cluster and Device Cluster

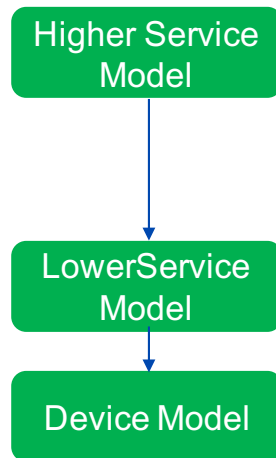
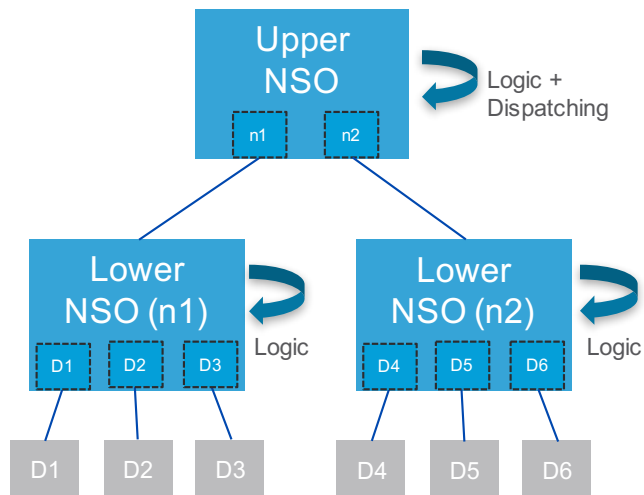
# Device Cluster



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

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

# LSA Cluster



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

Service Code impacted  
Split the devices across NSO nodes  
Parallelism  
Combine with commit queue to gain throughput

# Layered Service Architecture

- Achieving scale through a layer of abstraction
- NSO nodes are more independent
- Requires a stacked service with a dispatch (more work in design and implementation stage)
- Can be used in scenarios where separation is needed for other reasons (network topology)
- Future work: multiple “upper” / service nodes (scaling services)

# Device Cluster vs LSA 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 “device”
Service code is not impacted by the deployment	Service code need to be split between the NSO nodes
High penalty in NSO – NSO communication	Low penalty since the only communication is the service diff
Tight coupling between all NSO nodes	Loose coupling between NSO nodes

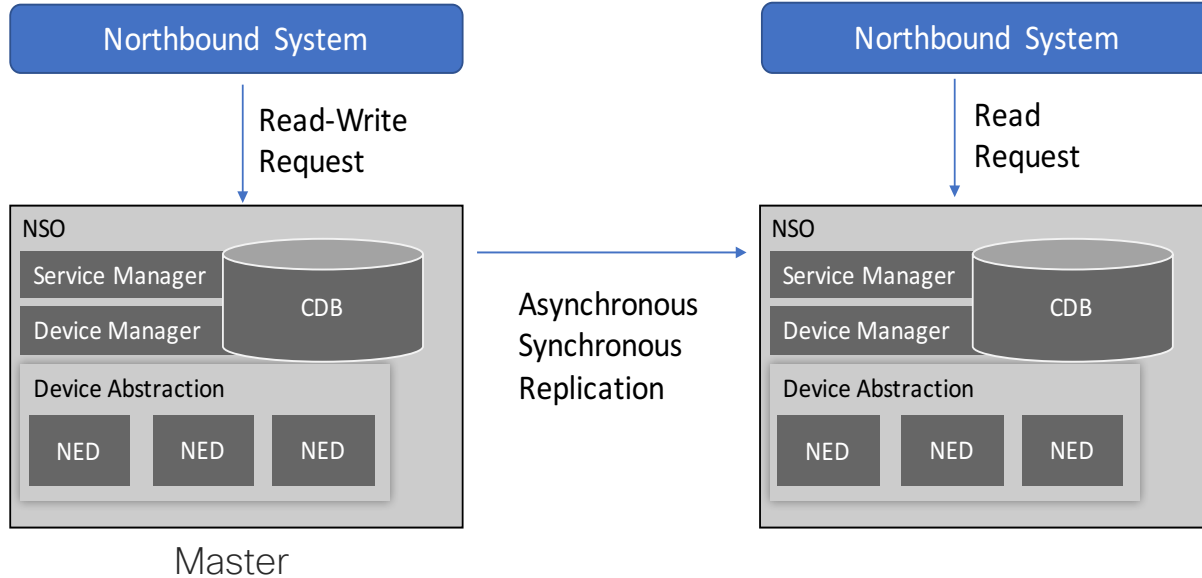
In common: *more NSO nodes = higher operational complexity*

HA Model

# NSO High Availability, recap

- Data integrity by: replication (one to many, chained) synchronous or asynchronous
- Availability by:
  - VIP (if on same L2 segment)
  - BGP routing
- Manual or programmable failover
- When combined with LSA clustering or Device clustering, all nodes in cluster typically run as HA pairs

# NSO HA, as it relates to performance

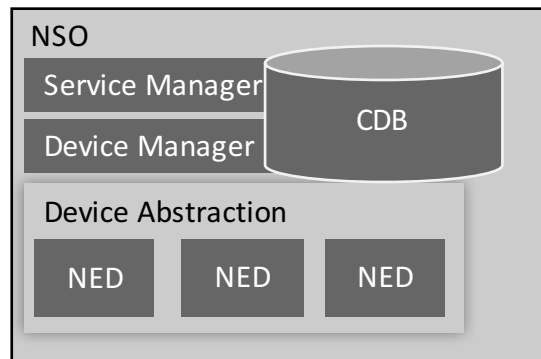
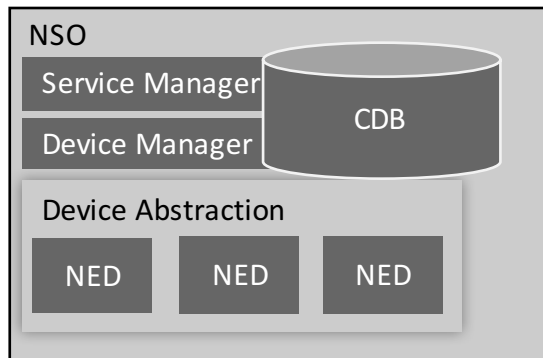


- Throughput (- if sync)
- Response time (- if sync)
- Scalability
- Reliability + (if sync)

“Slowest” NSO and link latency will limit performance

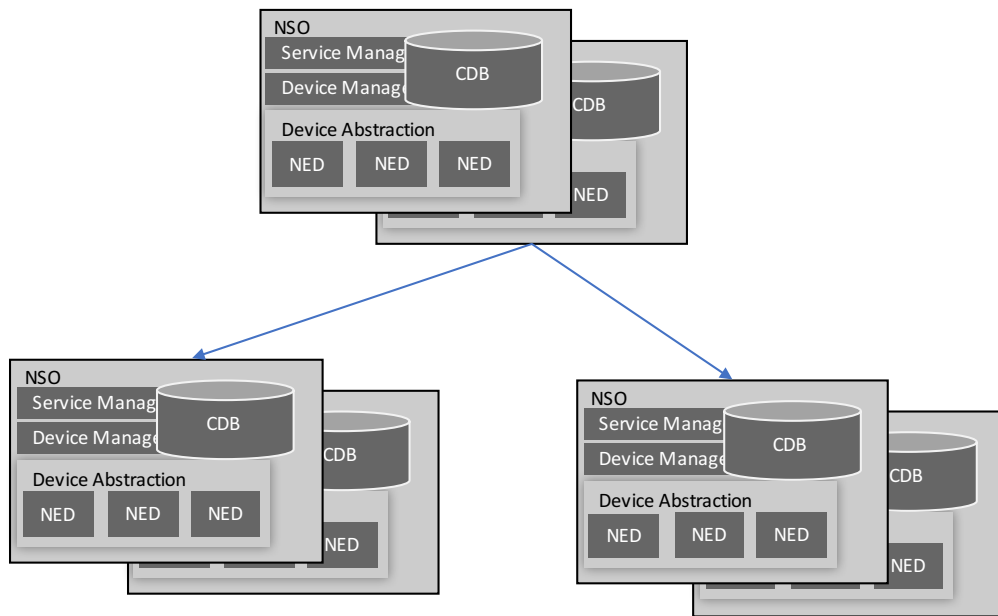
# Two Examples

# Single System with HA



- Use Commit Queue for 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
- Throughput
  - Concurrency
- Response-time
  - Commit Queue in top node

That is pretty much it...

# Some final advice

- Start simple (single node)
- Make POC complete enough to guide architecture
- Decide on an “Out-Of-Band Changes” model
- Decide on commit queue (desired throughput should guide)
- Avoid Device Cluster (special cases only)
- For large scale – design with LSA in mind
  - break down large service into smaller steps
- Implement realistic performance benchmark *early* – **keep measuring**
- Automate, automate, automate

