

Brown Field Services and Interior Design

What color is your IP Address?

Allan Eising
Network Automation Architect



Disclaimer

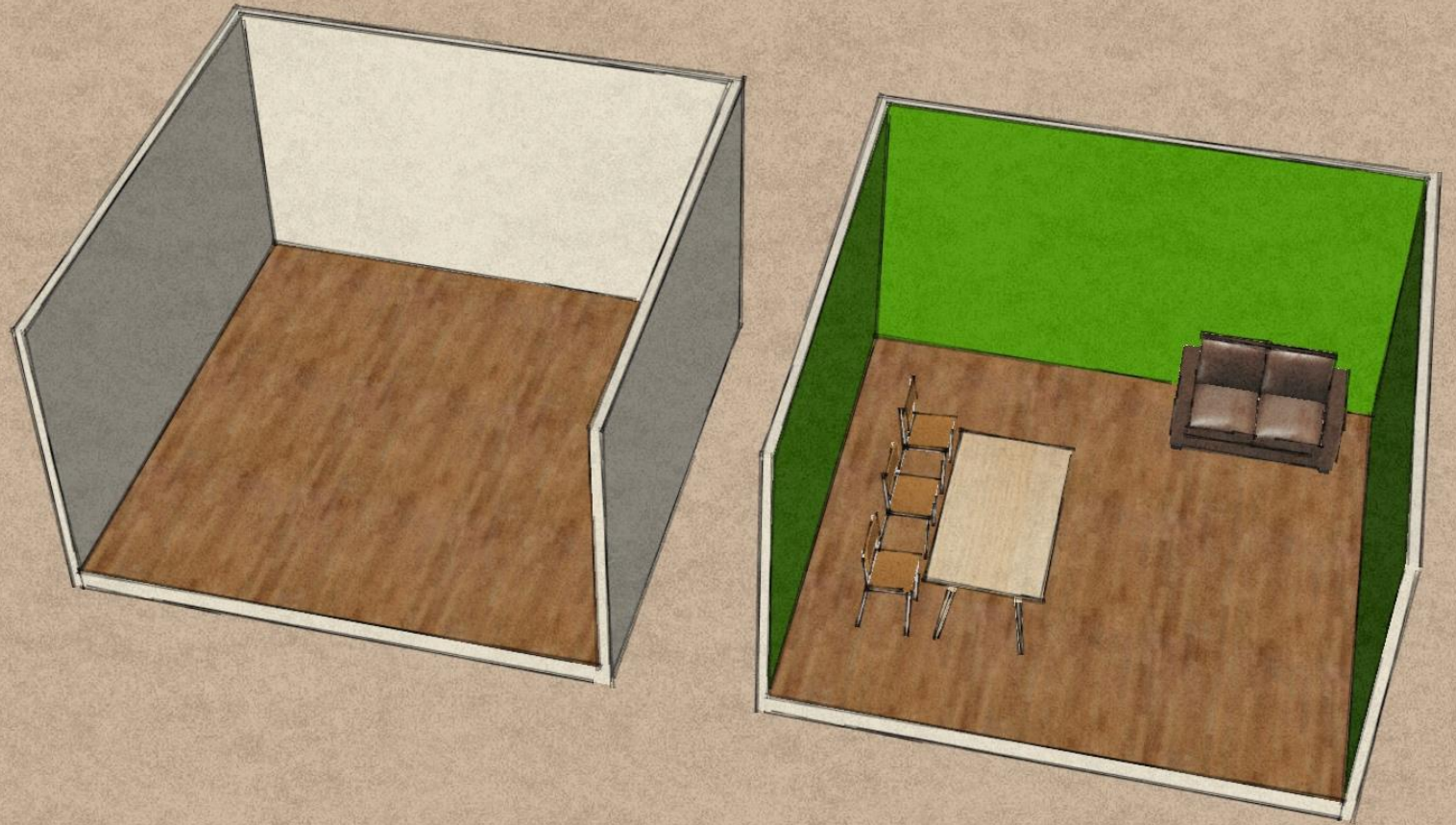
Most challenges in this presentations are common to service providers.

A few are very specific to Cisco NSO's approaches.

Everything should be useful generally.

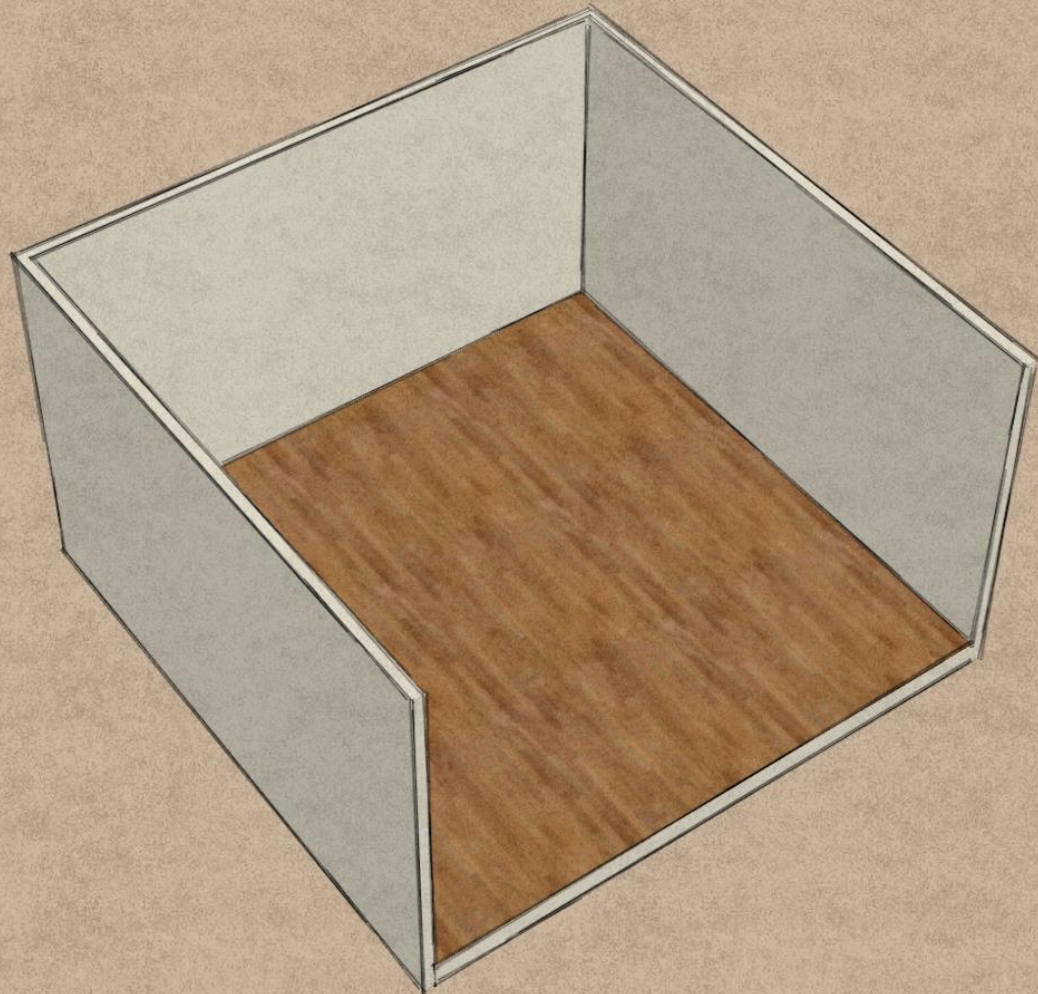


Service automation and interior design



Greenfield

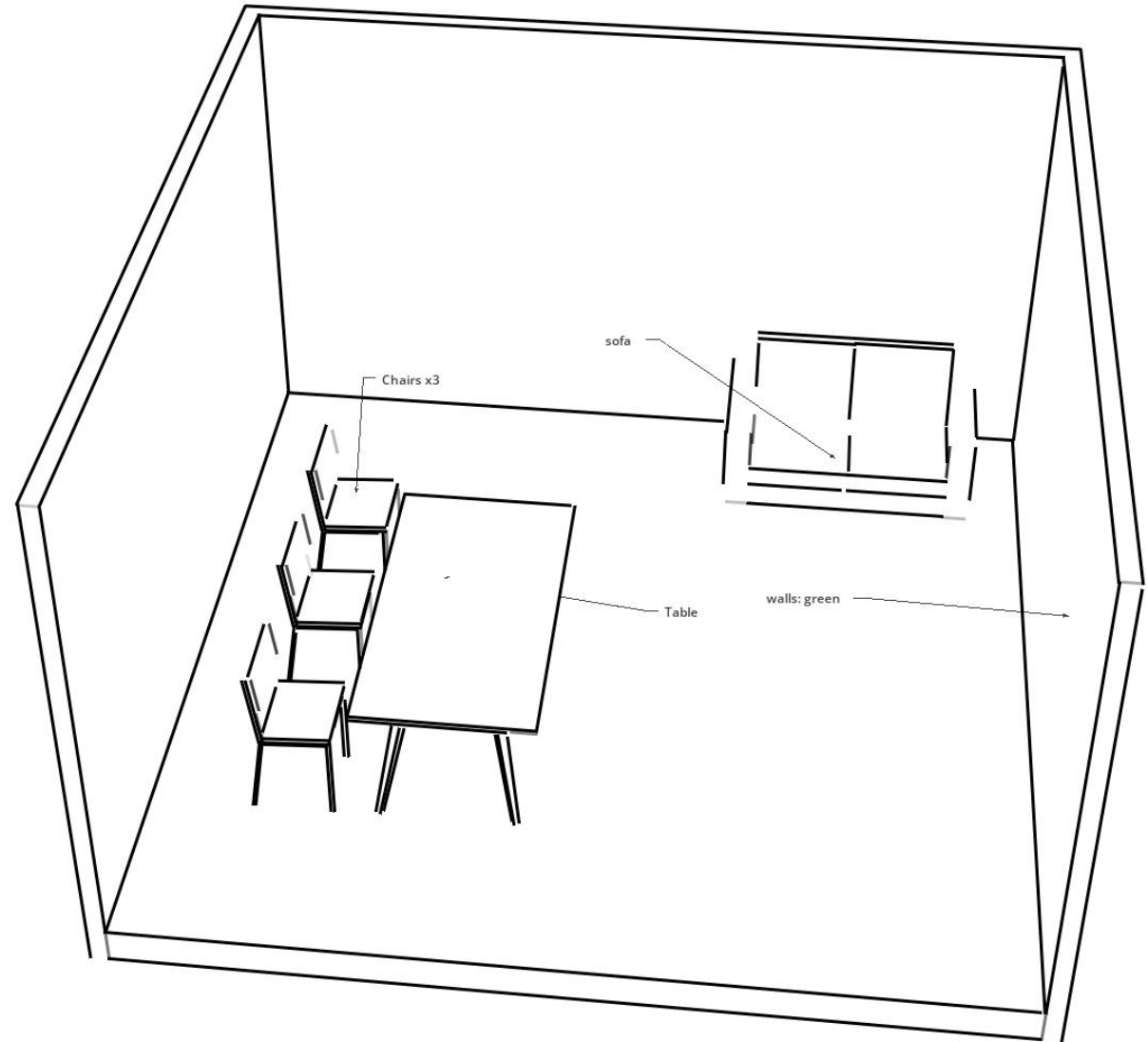
- In this scenario we start from a clean slate
- The walls in this room are unpainted, and no furniture exists



Service

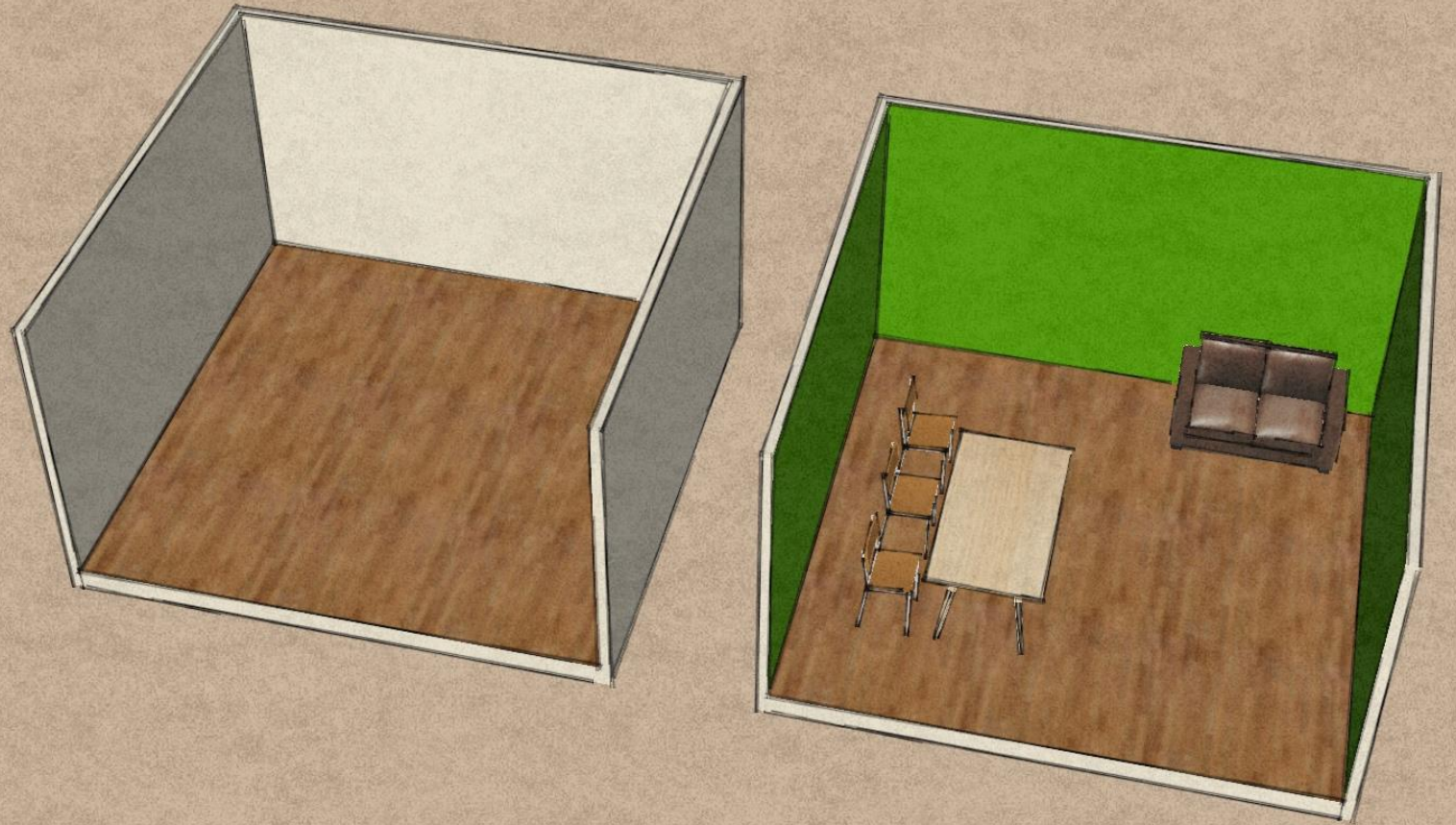
We then describe our desired state:

- Three chairs
- One table
- A sofa
- The walls should be green



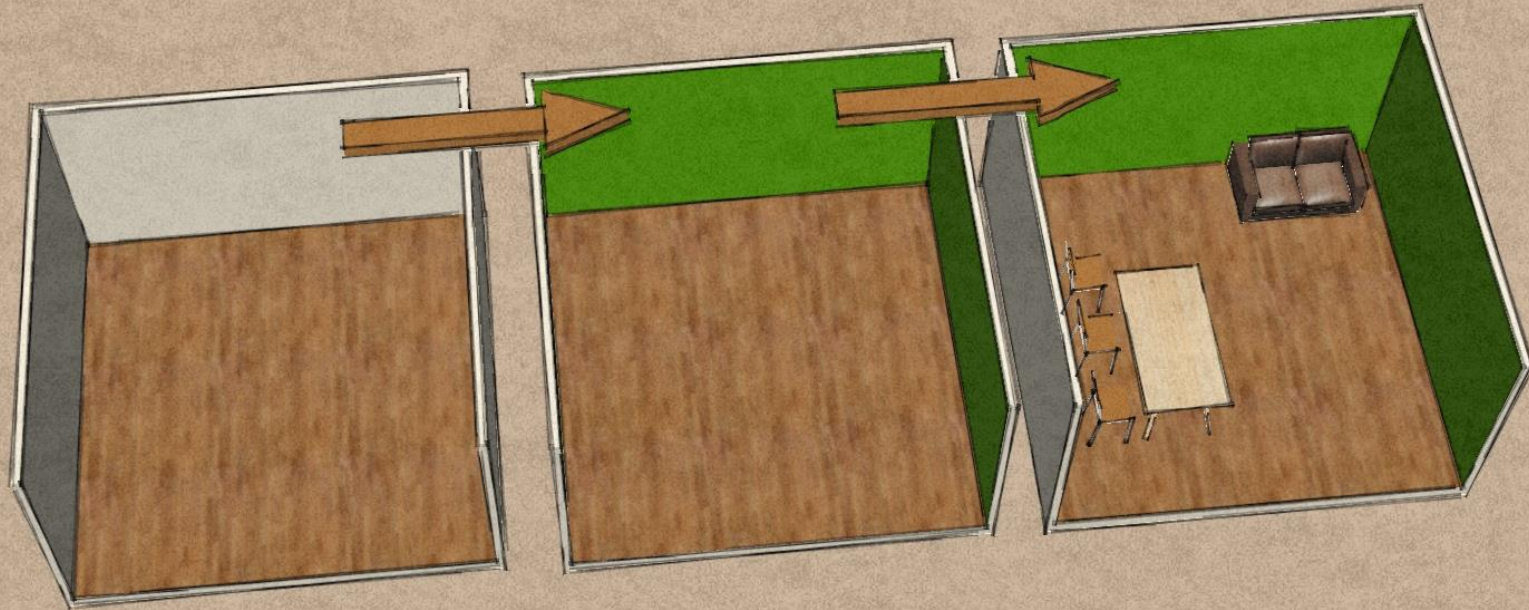
What changed?

- When we're done we can look at the steps we took to get there



Furnishing

- The actions we took are important
- Here:
 1. Paint the walls green
 2. Add the furniture



Create, modify, delete

— If we remember how to go from the empty room to the desired state, we can do the opposite too:

1. Remove the furniture
2. Paint the walls back to their original color

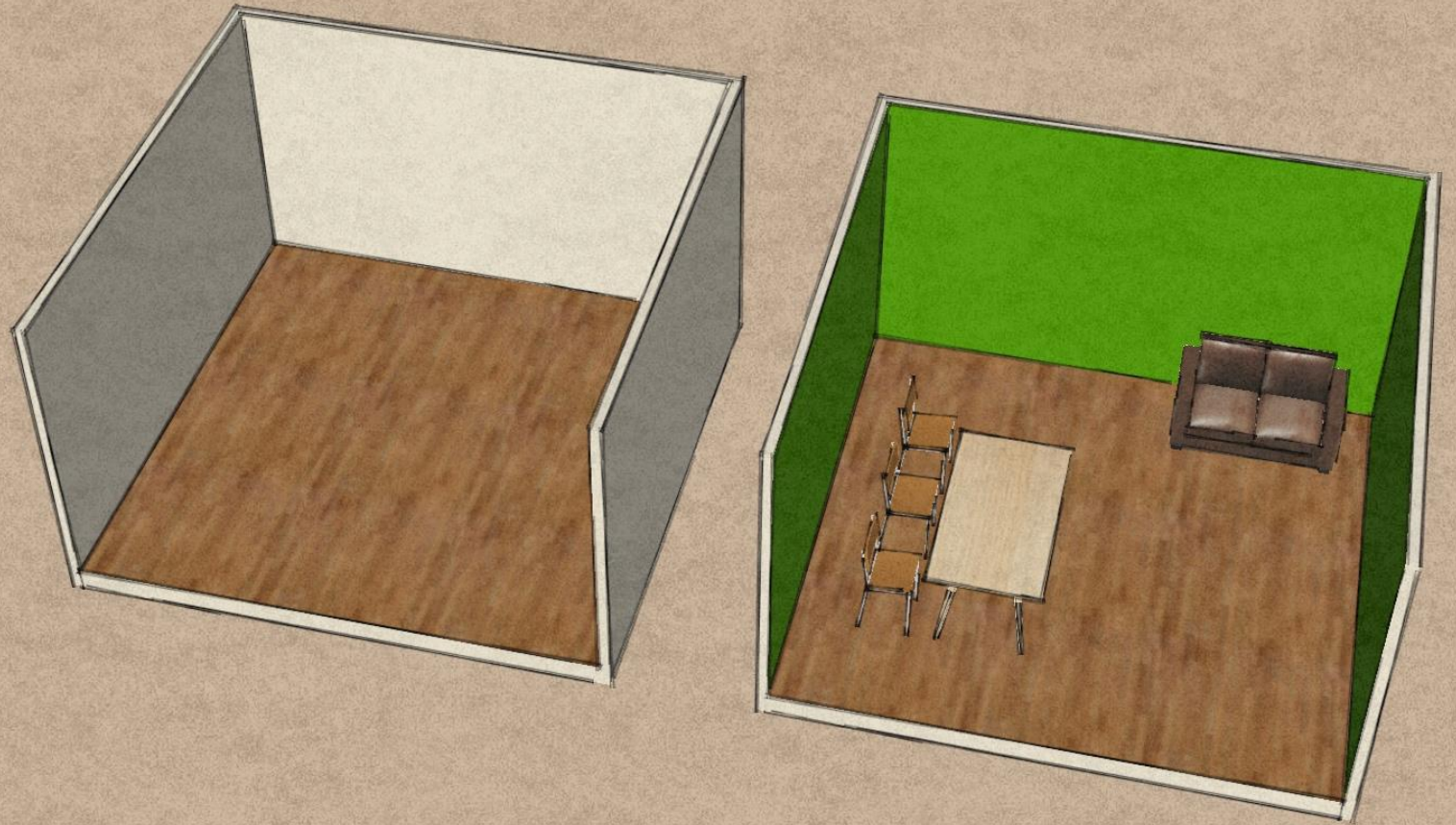
— Now we know how to add and how to remove

— Modifying the room came for free



Network automation

The act of transforming the *current* state into the *desired* state



Confused?
Let's try again...



Service

- Service automation should be declarative
- It describes how network state should look, e.g.,
 - Bandwidth
 - IP addressing

```
{
  "service": {
    "endpoint": "214002eb-a48f-49f7-9bf2-18dddf2f7b45",
    "bandwidth": "100m",
    "ip-addresses": [
      {
        "family": 6,
        "address": "2001:db8:f00::/56",
      },
      {
        "family": 4,
        "address": "192.0.2.1/30"
      }
    ]
  }
}
```



“Furnishing”

- To get the **network** to the desired state, a number of steps will be taken
- We find the end-point and then:
 1. Add the ipv6 address
 2. Add the ipv4 address
 3. Add the QoS policy

```
interface GigabitEthernet0/0/1/0
  ipv6 address 2001:db8:f00::/56
!
interface GigabitEthernet0/0/1/0
  ipv4 address 192.0.2.1/30
!
interface GigabitEthernet0/0/1/0
  service-policy input CUSTOMER_100M
  service-policy output CUSTOMER_100M
!
```



Create, modify, delete

- If we record how to go from the **interface** as it was to the desired state, we can do the opposite too:
 1. Remove the addresses
 2. Remove the service policy
- If we modify the service, we just have to reverse your steps, and start over

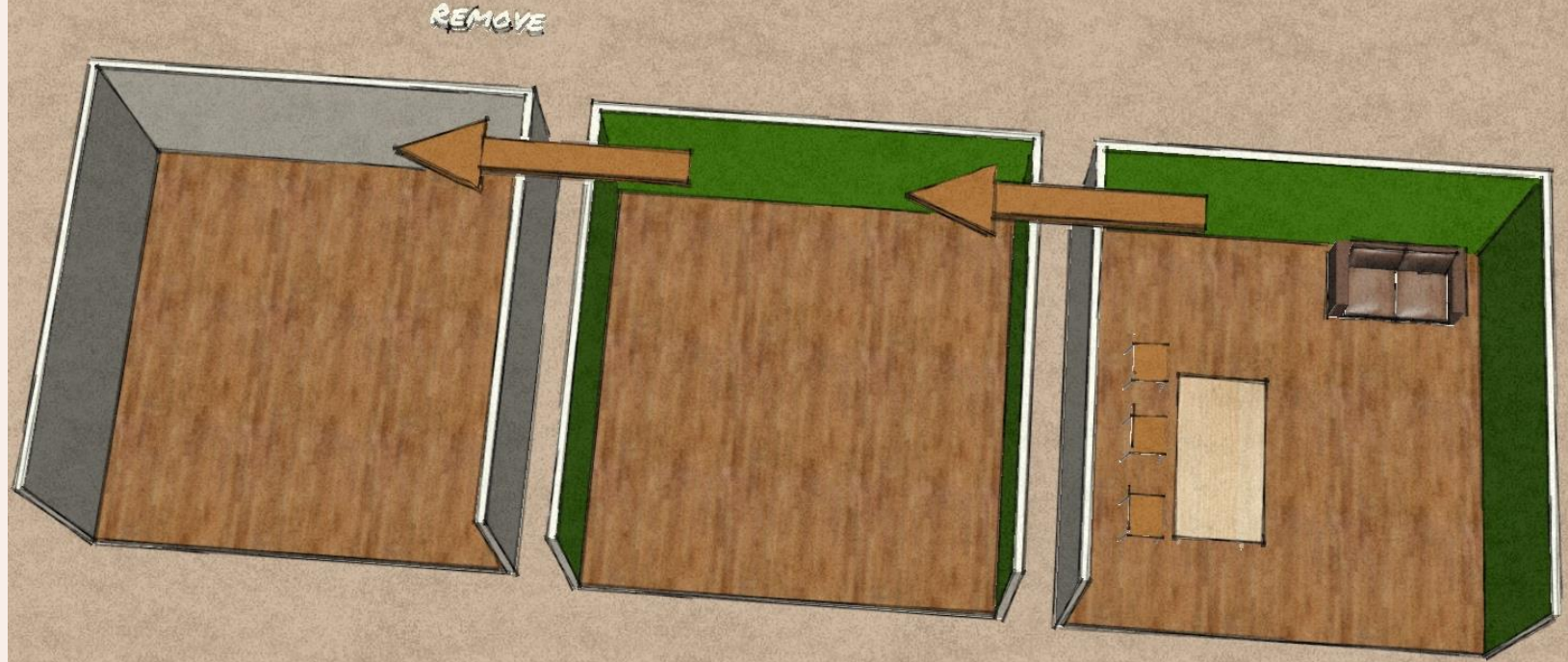
```
interface GigabitEthernet0/0/1/0
+   ipv6 address 2001:db8:f00::/56
+   ipv4 address 192.0.2.1/30
+   service-policy input CUSTOMER_100M
+   service-policy output CUSTOMER_100M
!
```

```
interface GigabitEthernet0/0/1/0
no ipv6 address 2001:db8:f00::/56
no ipv4 address 192.0.2.1/30
no service-policy input CUSTOMER_100M
no service-policy output CUSTOMER_100M
!
```



Network automation

When removing a service the *desired* state should be the state *prior* to activating the service



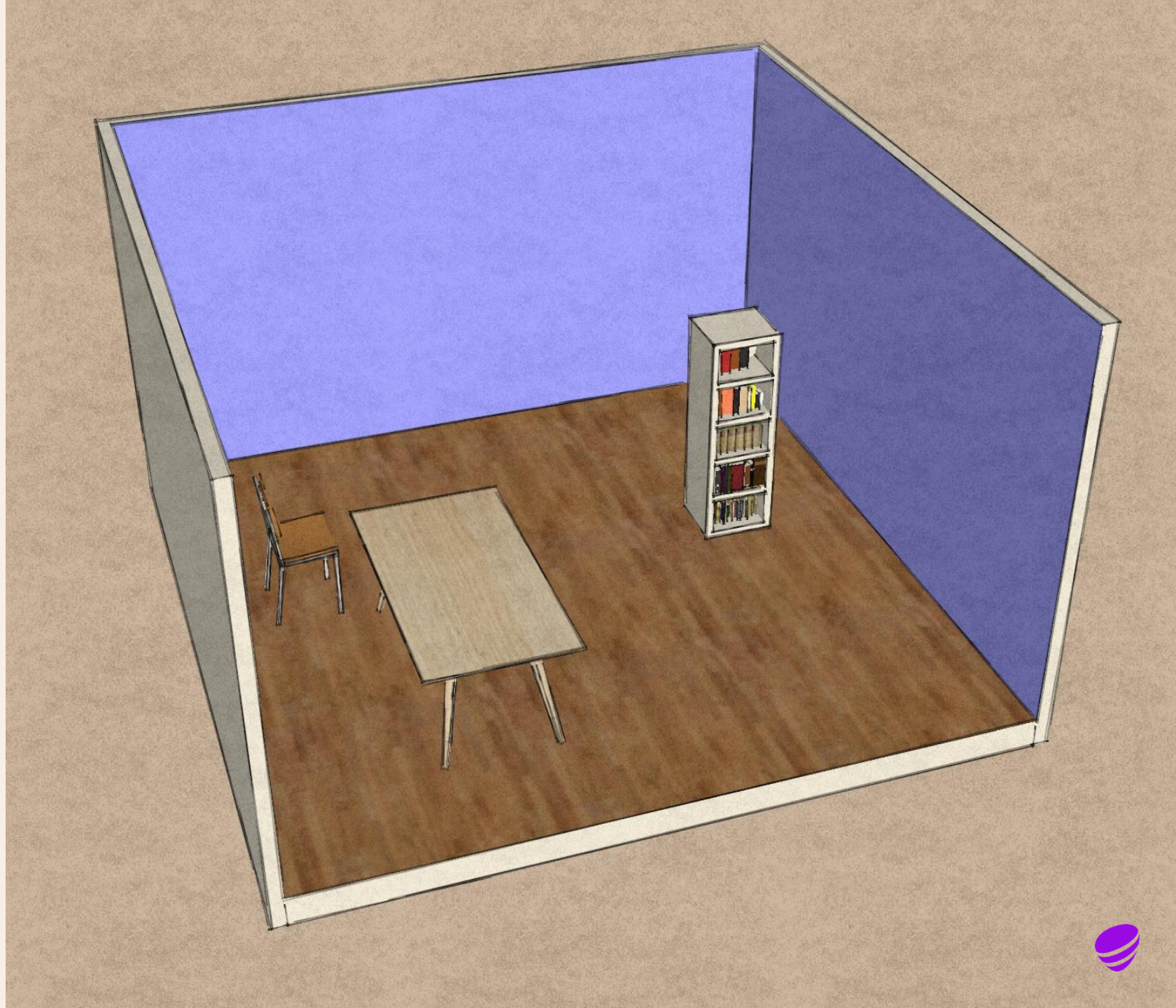
It's never that simple...

Brown field scenarios



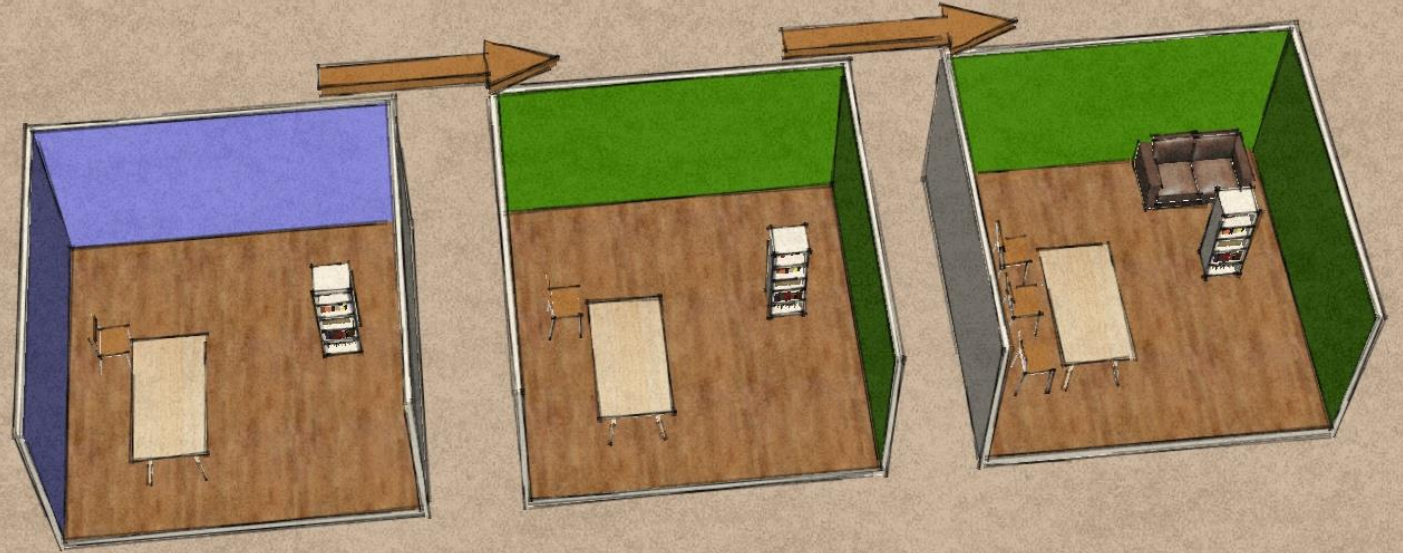
Brown field

- What if instead of an empty room, there is already something in it?
- The walls could have a different color
- Some furniture is already there
- Some of it may be expected, some not
 - like the bookcase



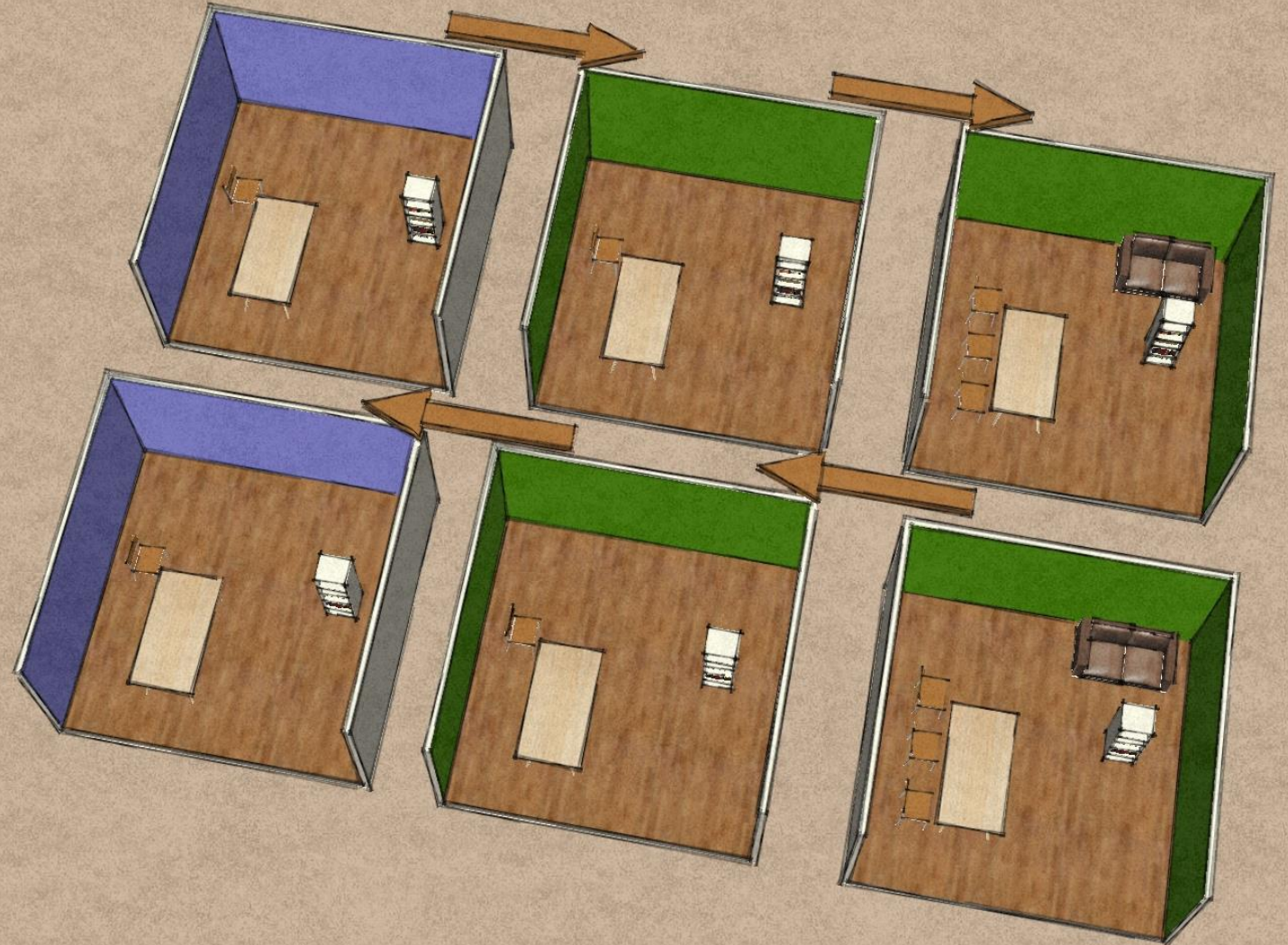
Reaching the intended state

- Our orchestrator should apply the desired state on top of the room as we found it
- The walls will be re-painted to green
- the missing two chairs will be added
- The sofa will be added
- But the bookcase will remain untouched



Removal

- When moving out, we should restore the room to its original state
- Including the old paint and the furniture that was already there
- Just like renting an apartment



How does that apply to network automation?



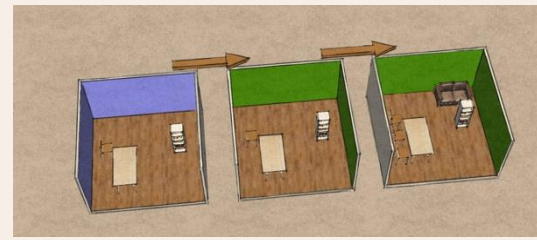
The brownfield network

- Most networks are full of customer services created in the past
- These could have been made by hand or with some other tools we don't use anymore
- Just like our rooms have furniture, our network devices has configuration already
- Just adding **no** in front of all our commands is no longer good enough
- Our documentation may help us



Reaching the intended state

- Our orchestrator should apply the **desired** state on top of the **current** state
- For example:
 - The IPv6 address will be changed to the right one
 - the missing IPv4 address will be added
 - But the mtu will remain untouched



```
interface GigabitEthernet0/0/1/0
  mtu 9216
  ipv6 address 2001:db8:ba0:1234::/64
  service-policy input CUSTOMER_100M
  service-policy output CUSTOMER_100M
!
```

```
interface GigabitEthernet0/0/1/0
  mtu 9216
  ipv6 address 2001:db8:f00::/56
  ipv4 address 192.0.2.1/30
  service-policy input CUSTOMER_100M
  service-policy output CUSTOMER_100M
!
```



Service Diff

- We still only record what changed in the network state
- What wasn't changed during creation, will not be touched during deletion

```
interface GigabitEthernet0/0/1/0
-   ipv6 address 2001:db8:ba0:1234::/64
+   ipv6 address 2001:db8:f00::/56
+   ipv4 address 192.0.2.1/30
!
```



Unexpected and expected existing states

Expected/safe

- Default configuration on interfaces
- Automatic configuration done by the platform

```
interface GigabitEthernet0/0/1/0
  description UNUSED_INTERFACE
  mac address-group PROTECT in
  no ip address
  shutdown
!
```

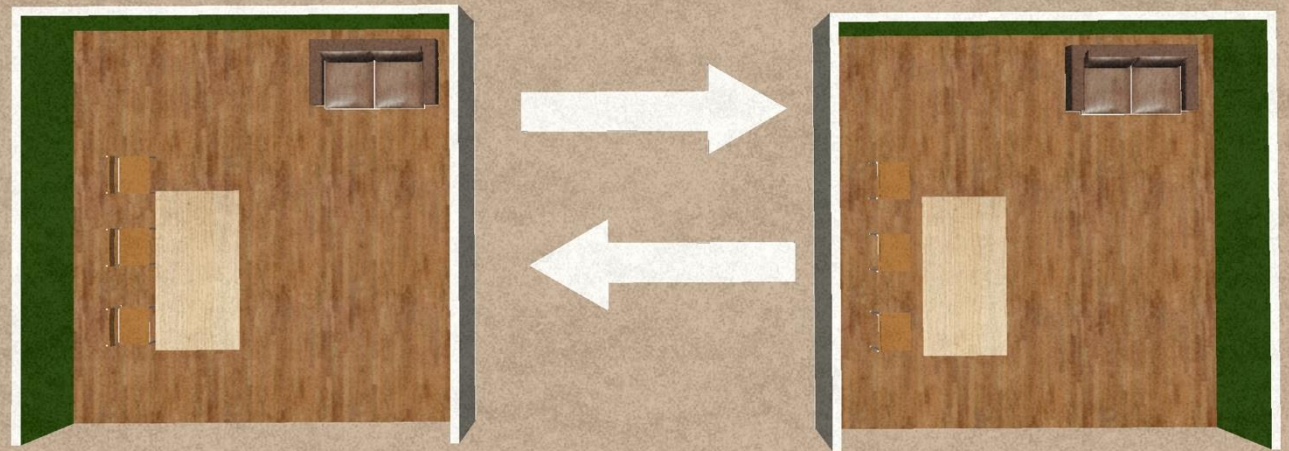
Unexpected/unsafe

- Wrong interface recorded in documentation
- Old configuration not entirely removed
- Overwriting an undocumented service



What if it was right from the start?

- When the automation is run, it has nothing to do
- Once the service is deleted, there are no steps to reverse
- This means that the service is removed, but the result of the service is still in the network
- We thought we removed the service, but it is actually still there!



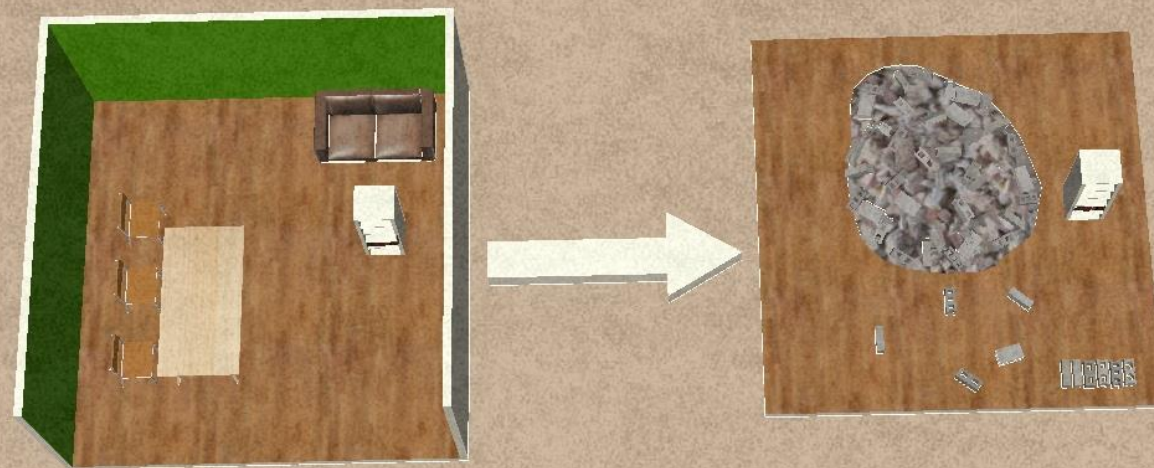
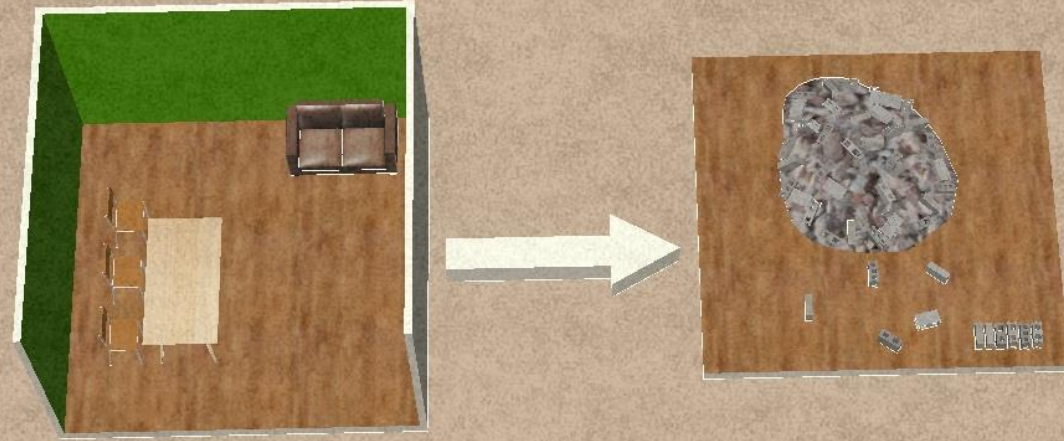
Ownership

- If the sofa was there before, we need to take over the ownership
- If there's an unexpected item, we need to remove it



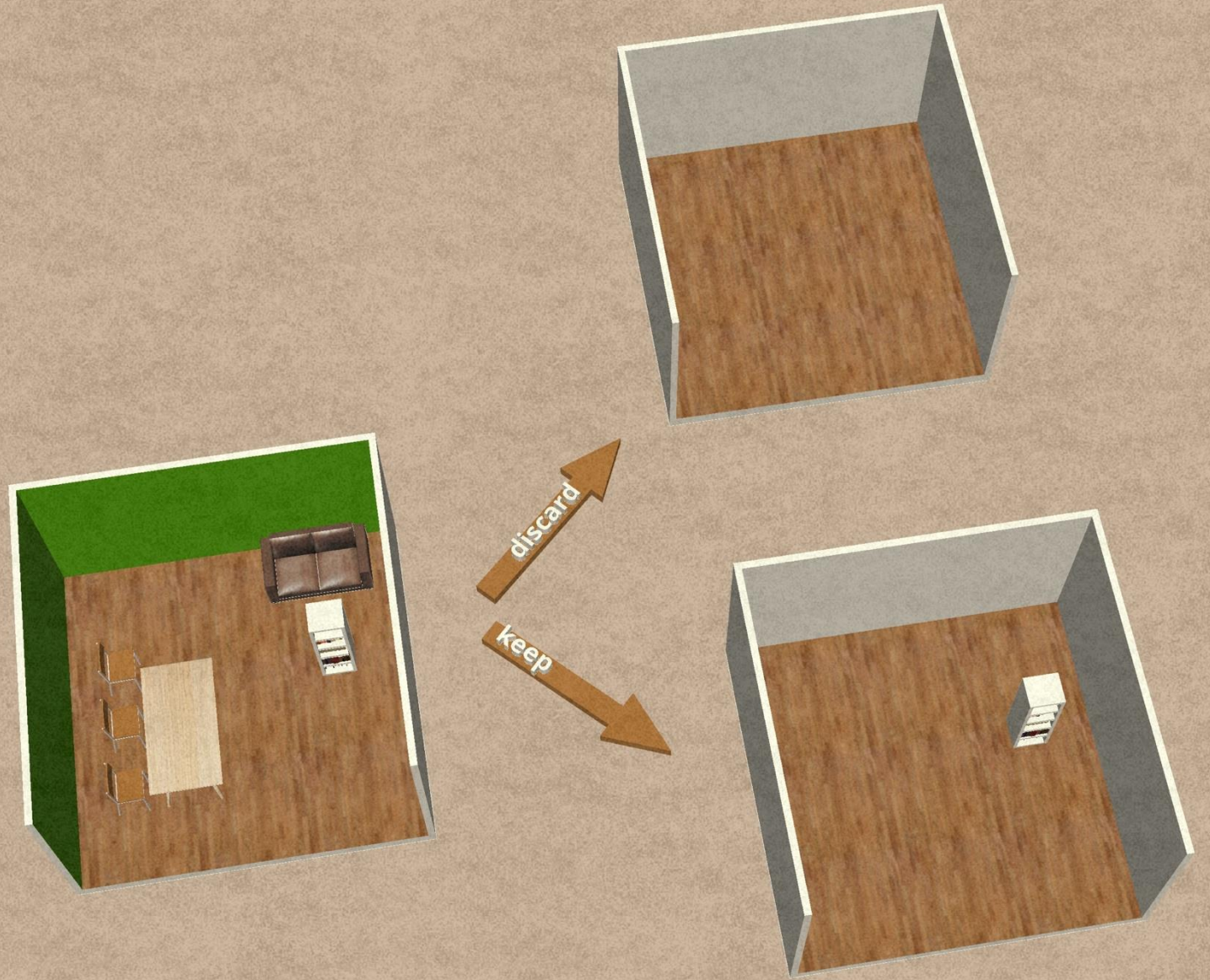
Ownership

- You shouldn't take over too much
- We don't want to tear down the whole room if there are still things inside it



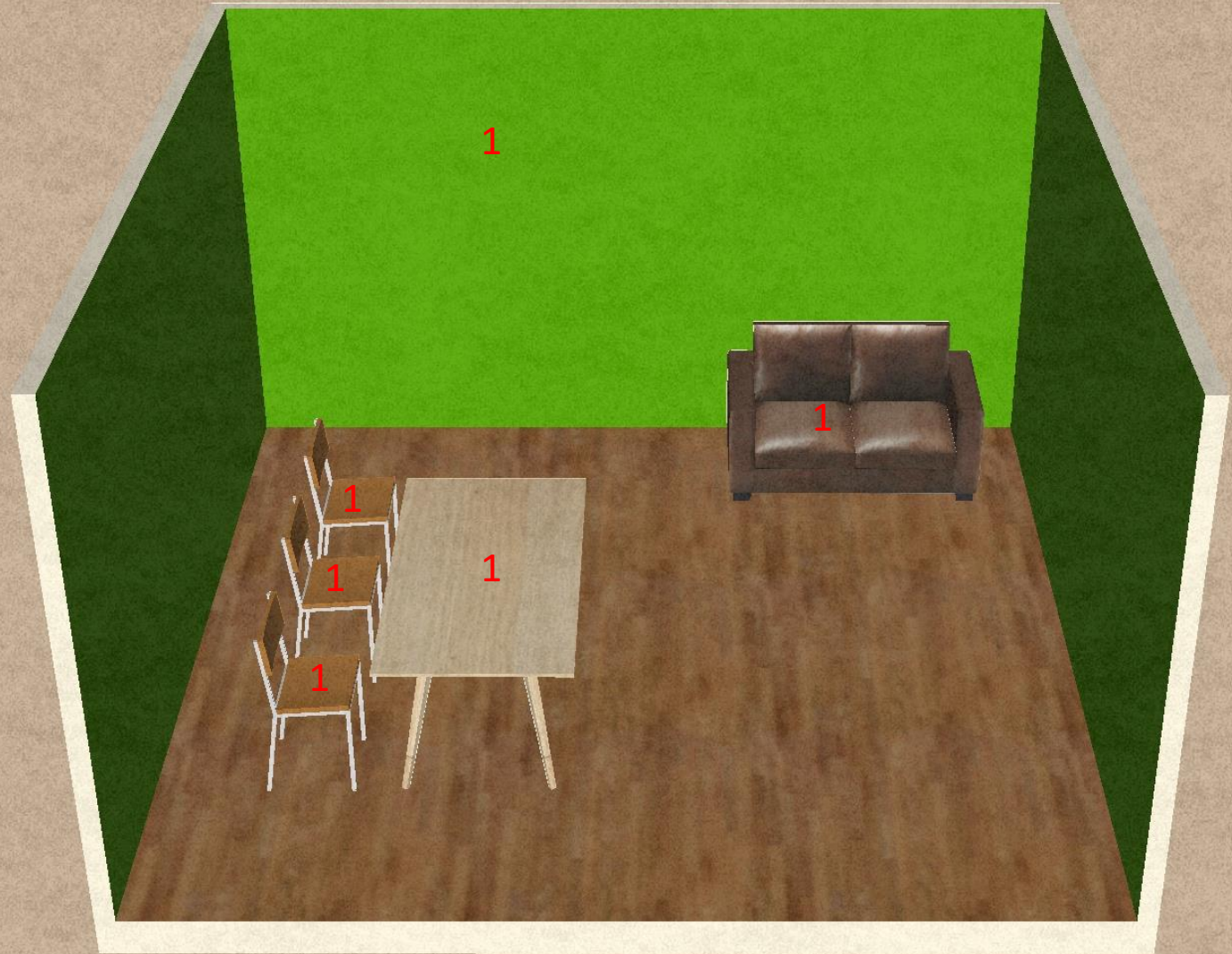
What about the bookcase?

- You need to choose what to do with the elements that are not part of your service
- Keep or remove?
- The bookcase is easy
- In reality it might be more complicated



Claiming ownership

- We also need to track who has a claim to each item
- Each item our service created from scratch is owned by **1**
- If a service wants to add something that is already there, it adds **1** to the counter
- An item should be removed if nobody owns it anymore



Backpointers

- Ownership should be public
- Each claim to an item should come with an identifier
- Each service leaves its address on each item
- This allows us to tie the ref-count to the actual services



Service meta data

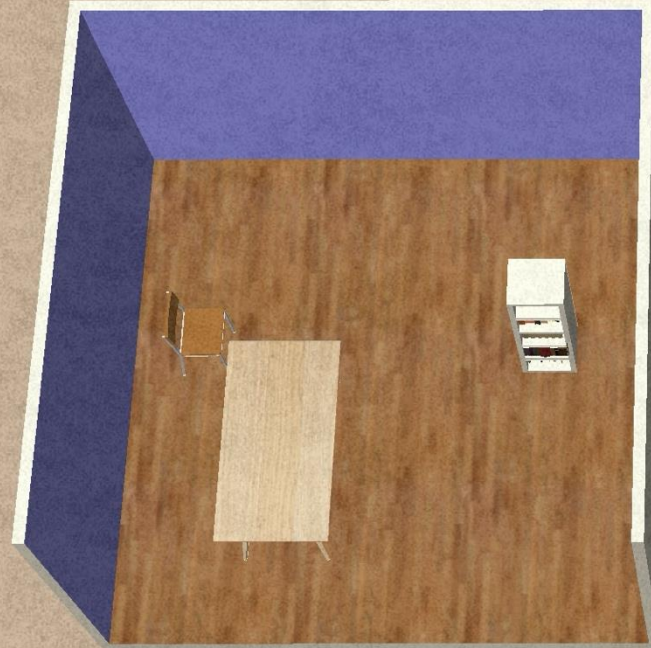
Ref-count: 2

backpointers

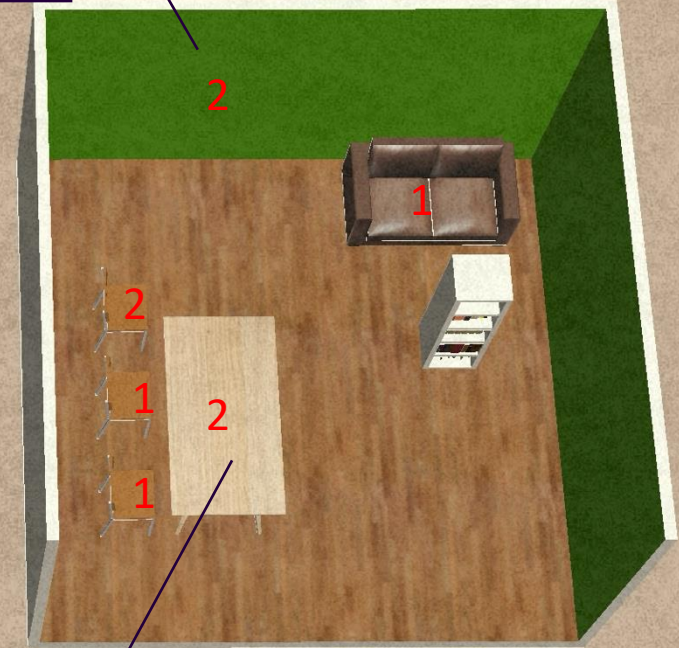
- Service 1
- Service 2

Brownfield

- If something already exists, it will get a ref-count but no backpointer
- The bookshelf is not part of the service, so it will have no ref-count
- Original values of items changed should also be recorded
- In NSO: “re-deploy re-concile”



Service meta data
Ref-count: 2
Original value
Blue



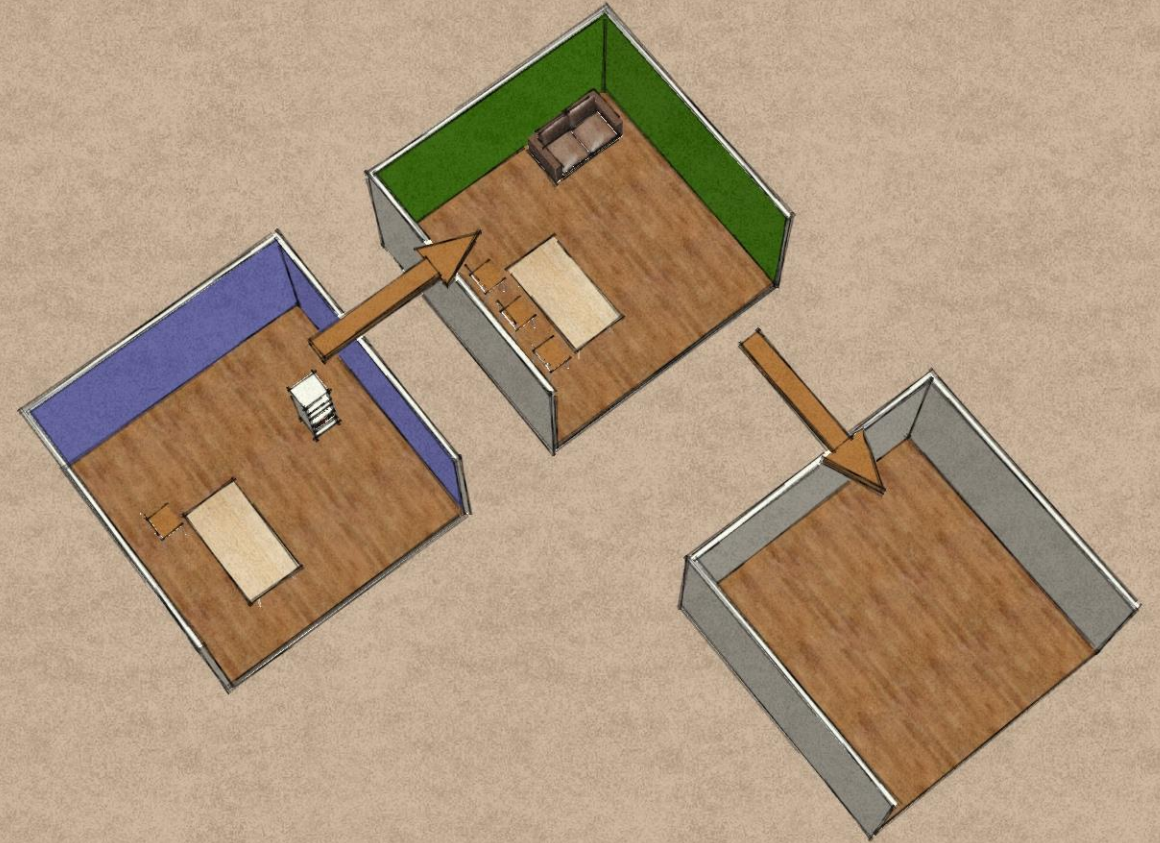
Service meta data
Ref-count: 2
backpointers

- Service 1



Reconciliation

Let's settle the differences



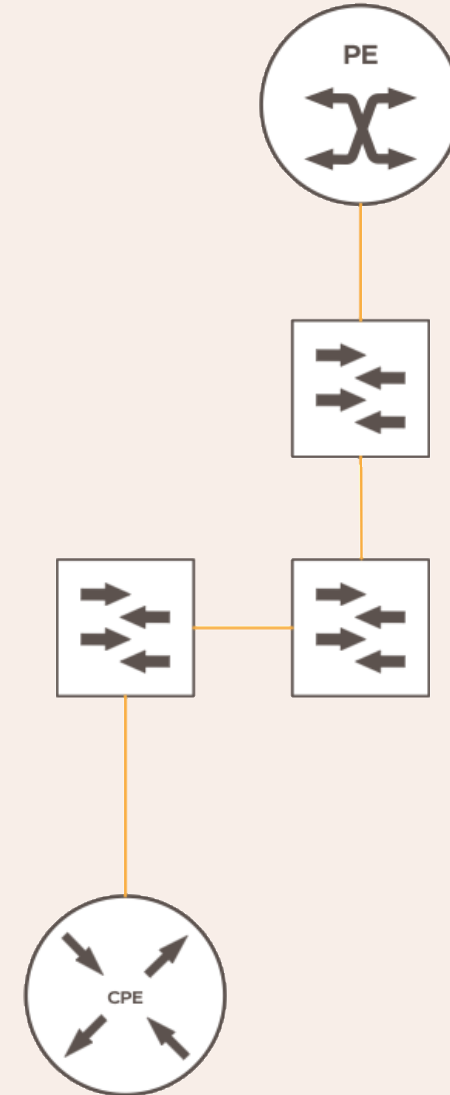
Who is right?

- Was the wall actually supposed to be blue?
- How many chairs did the customer actually want?
- Did the customer actually order a book case but we didn't support that in our standard product?
- This requires some insight in to the customer order



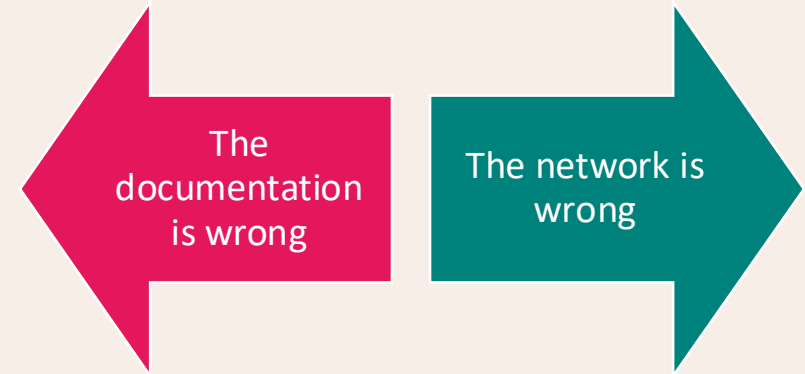
Who is right? (with proper examples)

- Was the ~~wall~~ **vlan** actually supposed to be ~~blue~~ **101**?
- How many ~~chairs~~ **megabit/s** did the customer actually want?
- Did the customer actually order a ~~book case~~ **OSPF** but we didn't support that in NSO or our standard product?



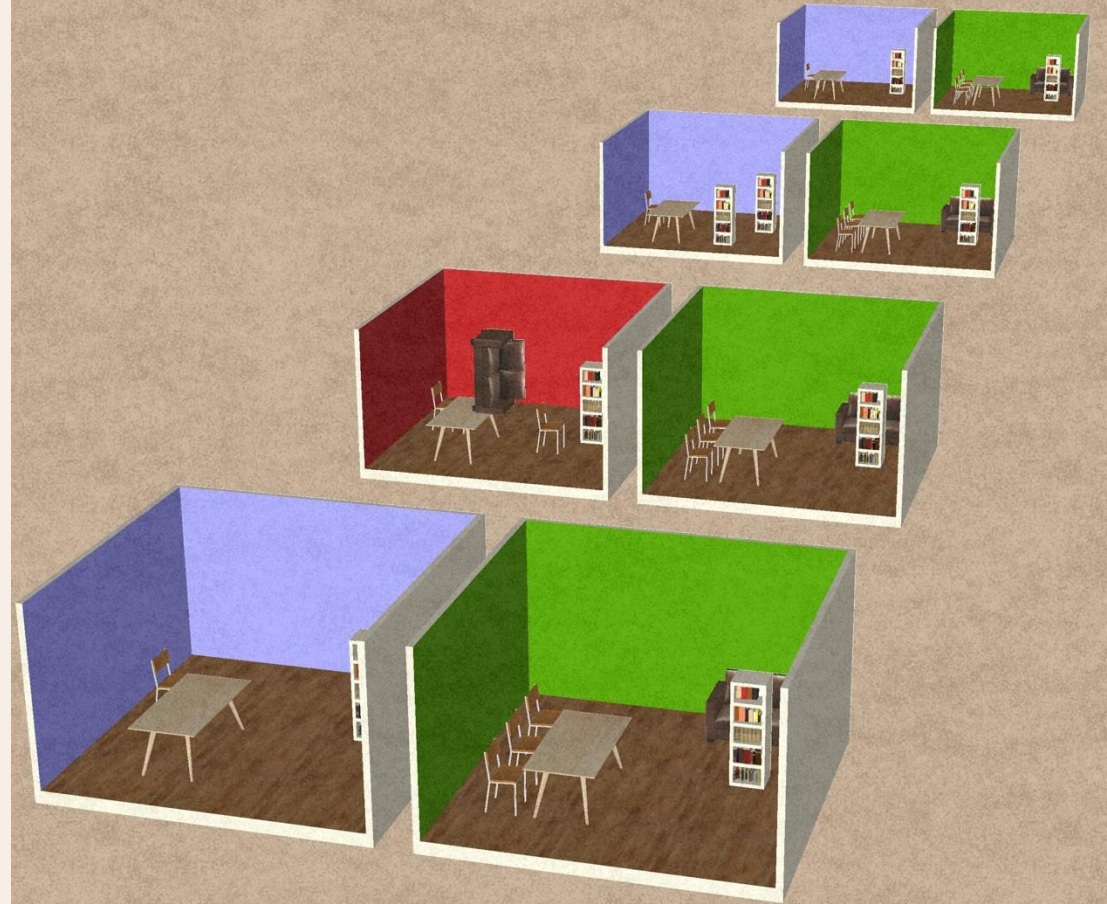
Who is right? (with proper examples)

- Was the ~~wall~~ **vlan** actually supposed to be ~~blue~~ **101**?
- How many ~~chairs~~ **megabit/s** did the customer actually want?
- Did the customer actually order a ~~book case~~ **OSPF** but we didn't support that in NSO or our standard product?



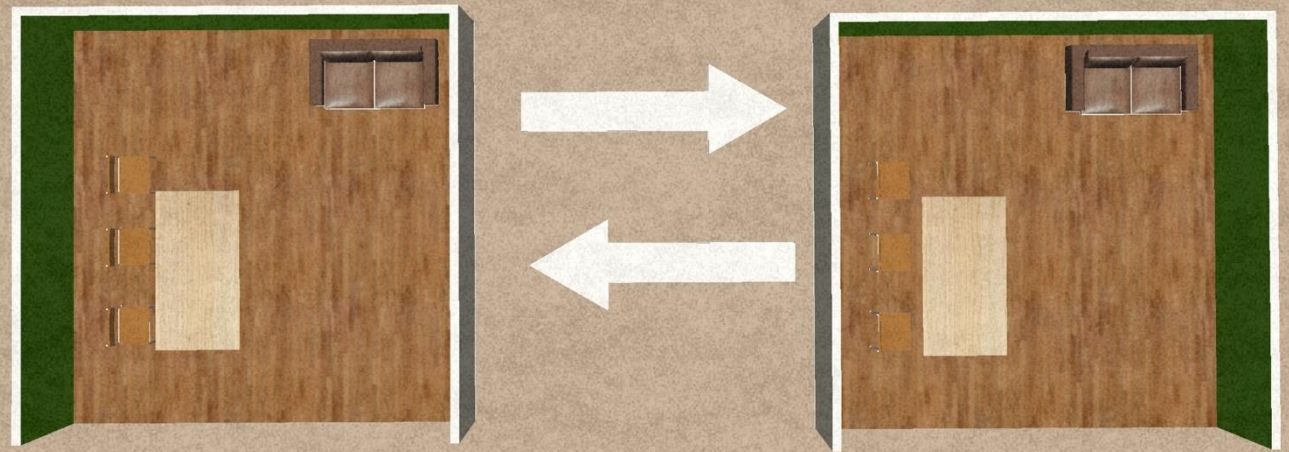
Challenge at scale

- Some of the original states might have been correct
- Some are obviously incorrect
- But it may be hard to tell without knowing
- There may be thousands of services to analyze



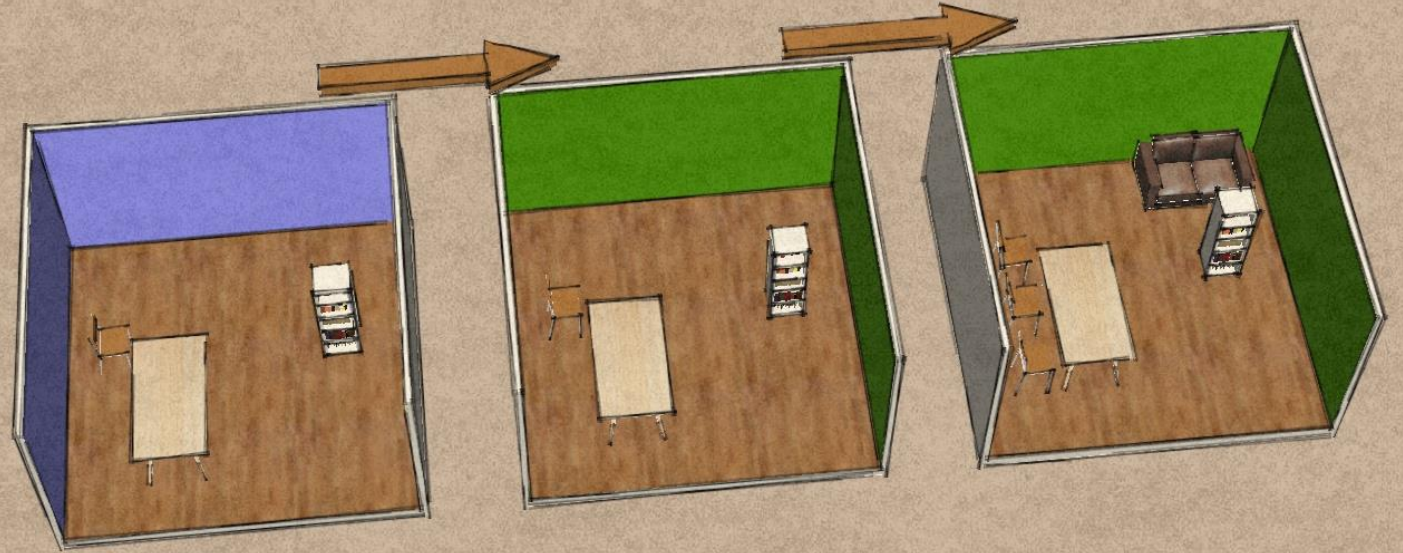
Service analysis

- We expect all existing services to be correct
- We expect the current state to match the desired state



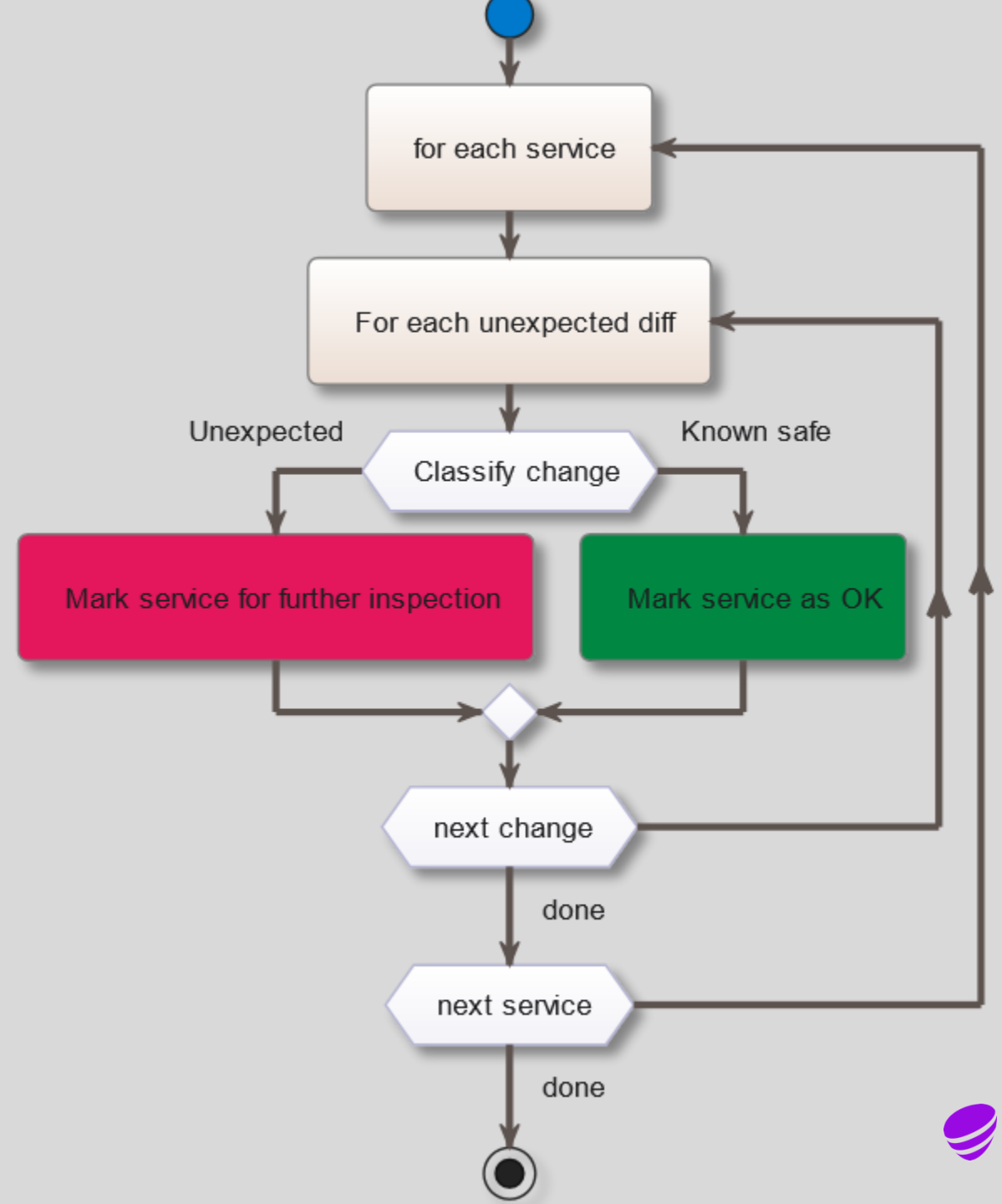
Service analysis

- If this is true, nothing will be done
- If it is not, the actions tell us about a deviation



How do we solve it?

- We can do this at scale
- We can use "commit dry-run" or work with virtual routers
- Any deviation should be analyzed and categorized
- Some changes might be safe and unimportant
- Other changes might indicate something larger



Making it user friendly

- Our first version was a python script that wrote text files
- Now we are building a tool to make it easy
- Here we can process large batches of services on top of a production copy
- The services with differences can be processed

The screenshot shows a web application interface for reconciliation. The interface is divided into a sidebar on the left and a main content area. The sidebar contains navigation options: System, Devices, Alarms, Tools, and Task list. The main content area is titled "RECONCILIATION" and displays a table of reconciliation results for various subscriptions. The table has columns for Subscription, Type, Version, Batches, Status, Result, Created, Updated, and User. The table shows 20 rows of data, with some rows highlighted in red to indicate differences. A search bar is located at the top of the main content area, and a pie chart is displayed above the table. The sidebar also shows the user's name "admin" and the current date and time "Monday 10:29 AM".

| Subscription | Type | Version | Batches | Status | Result | Created | Updated | User |
|-------------------------------|----------|---------|-----------------|--------|--------|---------------------|---------------------|------|
| TIC-200-82655 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-199-86269 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-198-83660 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-197-88816 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-196-83179 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-195-89387 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-194-81930 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-193-81028 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-192-89523 | Internet | 1 | CMFT-1-8632-S04 | ⚠️ | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:27 | |
| TIC-191-81206 | Internet | 1 | CMFT-1-8632-S04 | ⚠️ | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 10:26:09 | |
| TIC-190-86850 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-189-82147 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-188-81655 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-187-80035 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-186-89249 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-185-85866 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-184-85653 | Internet | 1 | CMFT-1-8632-S04 | ⚠️ | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:27 | |
| TIC-183-83883 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |
| TIC-182-86061 | Internet | 1 | CMFT-1-8632-S04 | 🔍 | 🔍 | 2025-02-17 09:02:22 | 2025-02-17 09:31:28 | |



Processing a service

- The operators can drill down on individual services
- They can adjust the service parameters to match
- Or approve or reject the service as it is

The screenshot displays the Telia Reconciliation interface for subscription **TIC-1-86256**. The interface is divided into several sections:

- Parameters:** A list of service parameters including Circuit ID (circuit-182788-1), Company Alias (BrokeButHappy Ltd.), CPE Hostname (cpe-1), CPE LAN IP ID, CPE LAN IP Type, CPE Model (C1100), CPE Vendor (CISDO), CPE WAN IP ID (0/0/0), CPE WAN IP Type (GigabitEthernet), Cust Prefixes (44.0.1.0 255.255.255.0 44.0.1.1), C Tag (1202), Download BW Rate (3), Download BW Unit (Gigabit), FN Access Nodes Ports (rer1:GigabitEthernet0/1/0), Link Prefix (1.2.6.114/31), PW Ether If (22), SEB Primary (seb1), SEB Secondary (seb2), Subscription ID (TIC-142760-1), S Tag (1403), Upload BW Rate (3), Upload BW Unit (Gigabit), VAS Data Elements (These are procedurally generated subscription parameters), and VC Identifier (168).
- Discrepancies:** A section titled "Discrepancy list not yet implemented".
- Services:** A section for "Access" showing "Device Diffs" with configuration snippets for nso, services, and access. It includes "Device Native Config" for interfaces rer1, seb1, and seb2.
- Prolane:** A section for "Device Diffs" showing configuration for cpe-1 nso, including interface GigabitEthernet 0/0/0.1202 and IP addresses 1.2.6.161 and 1.2.6.115. It also shows "Device Native Config" for interfaces seb1, seb2, and cpe-1.
- Events:** A list of events including "Discrepancy Report Done" (2025-02-17 10:33:24) with an "Auto Fail" result, "Discrepancy Report Start" (2025-02-17 10:33:10), and "Import" (2025-02-17 10:33:33).
- Task list:** Shows the current time as Monday 10:36 AM.



Summary

- The key challenges to brown field automation concerns
 - Reconciliation of differences
 - Ownership of configuration
- If your services own too little, they won't clean up after themselves
- If your services own too much, they can cause major outages



Summary

- You should analyze if existing services match what is currently in the network
- This can be done with dry-run simulations
- Deviations should be analyzed carefully
- Use your network delivery engineers. They are experts in this field.



Thank you!

Questions?

