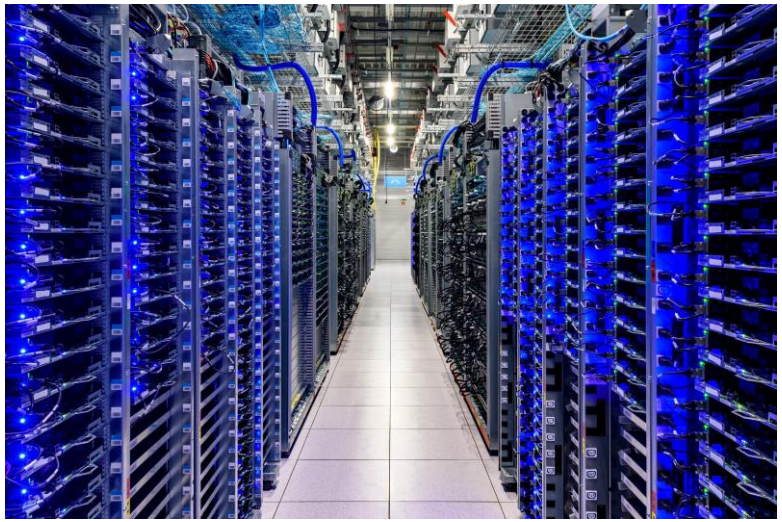
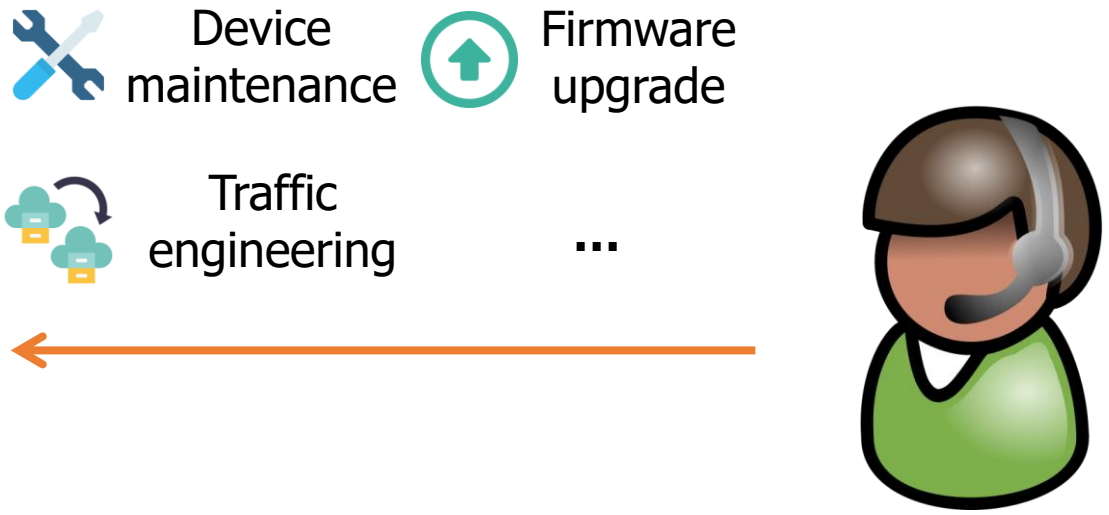
Occam: A Programming System for Reliable Network Management

Jiarong Xing, Kuo-Feng Hsu, **Yiting Xia**, Yan Cai,
Yanping Li, Ying Zhang, Ang Chen

Problem: Managing large-scale networking services/devices



Large-scale data centers

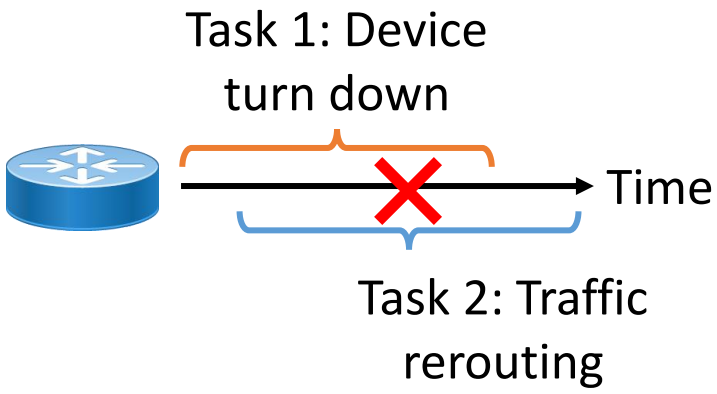


Network management team

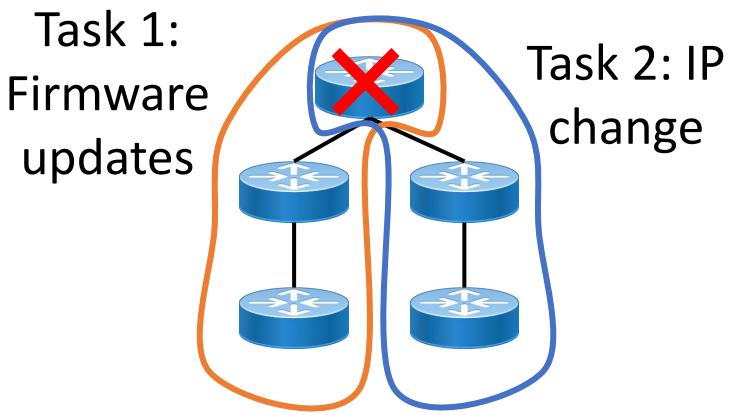
- Modern data centers are constructed with large-scale networks
- Networks are constantly changing for evolving demands
  - e.g., some part of the network is being upgraded every day*
- Network management becomes critical and challenging

*Evolve or Die: High-Availability Design Principles Drawn from Google's Network Infrastructure, SIGCOMM 2016

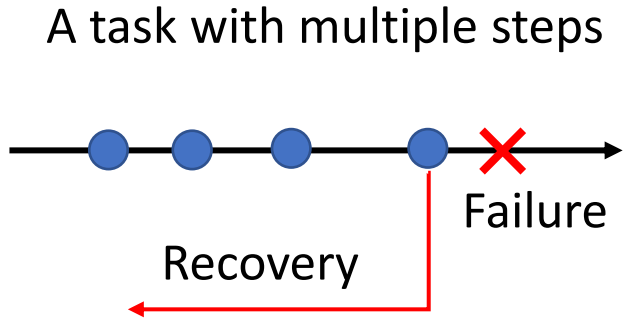
Key challenge: Reliability!



EX1: Network state conflicts



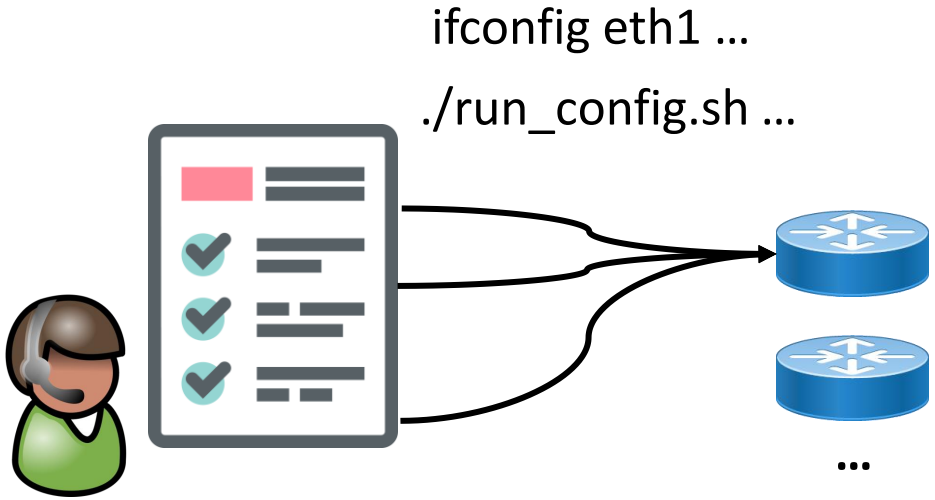
EX2: Device operation conflicts



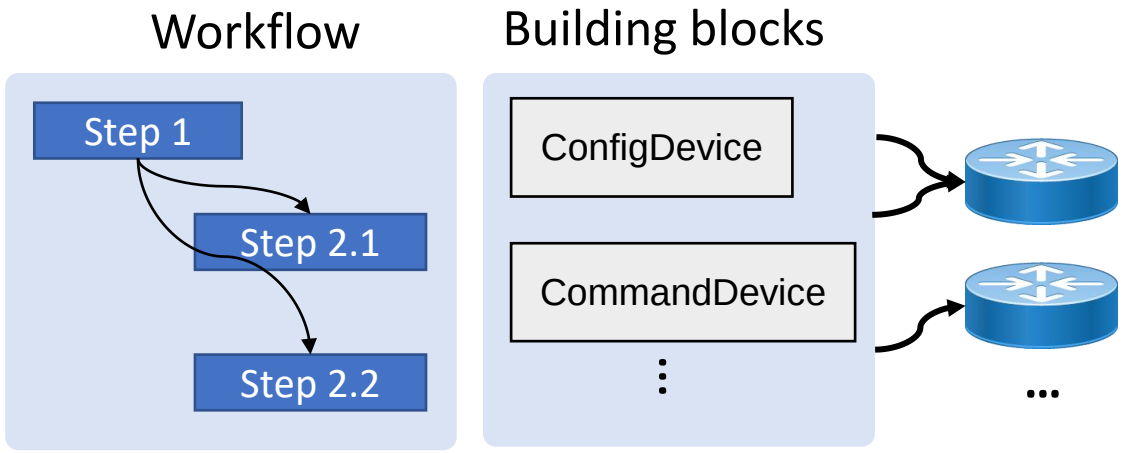
EX3: Failure handling

- Reliability is a key challenge in network management
  - e.g., net. state conflicts, device operation conflicts, failure handling
- Peripheral concerns, but make management very challenging

Existing solutions and limitations



Method of procedure (MOP)



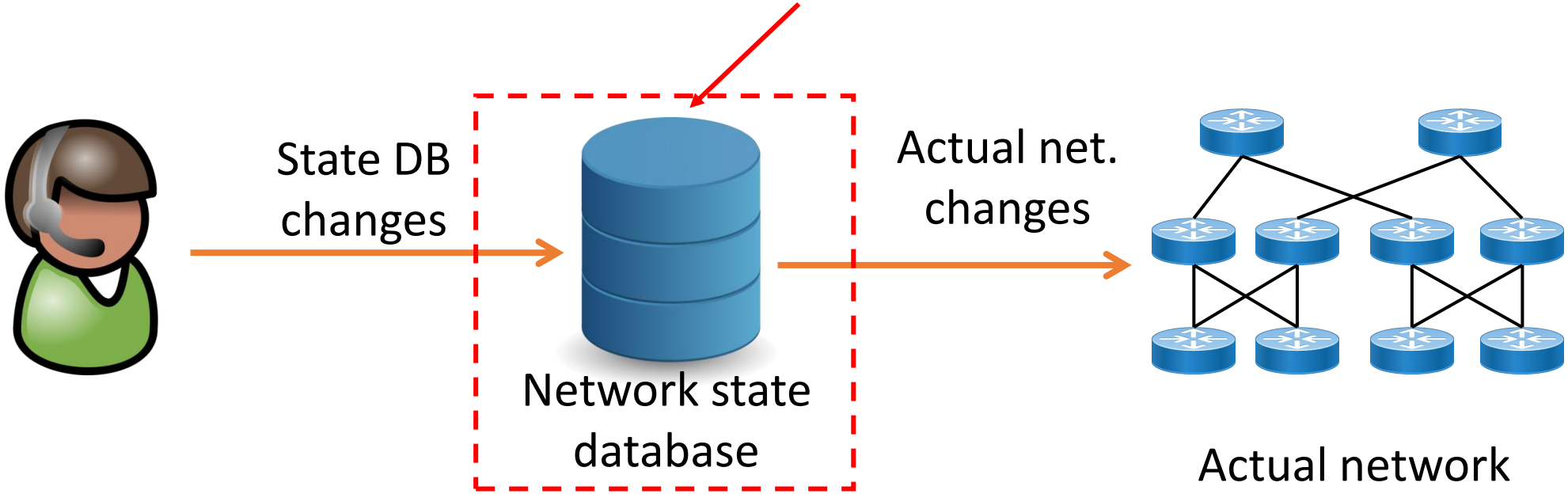
Workflow systems

- Method of procedure (MOP): A guide to manually run the commands
  - Manual operations, inefficient
- Workflow systems with reusable building blocks for automation
  - Not designed to address network-specific issues, e.g., reliability support

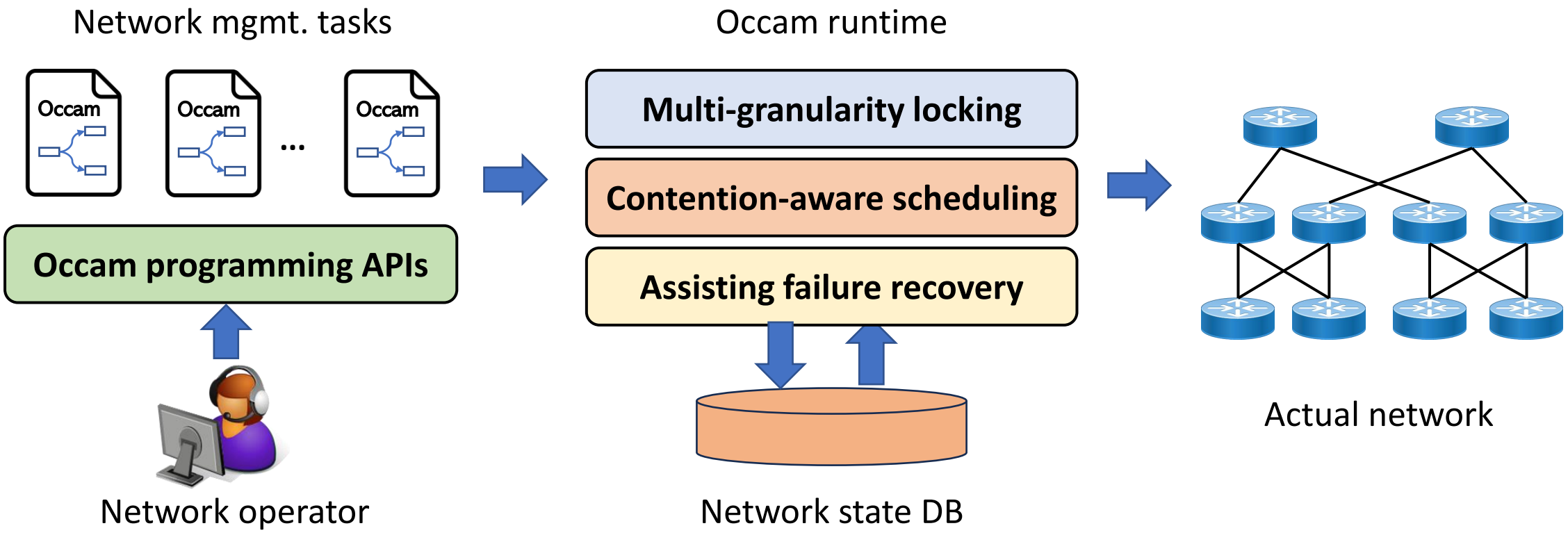
Key question and idea

How to achieve reliable and efficient network management in modern data centers?

Handling reliability concerns using customized DB techniques



Occam: Enabling reliable and efficient network management



- Programming APIs: Hide operators from peripheral concerns
- Multi-granularity locking: Ensures consistent device states
- Contention-aware scheduling: Maximizes task execution efficiency
- Assisting failure recovery: Facilitates management resilience

Programming management tasks using Occam APIs

API	Description
net(regex**)**	Create and scope a network object
net.get([attr_key**])**	Get device attribute with key
net.set([attr_key, val**])**	Set device attribute with key
net.apply(func**)**	Execute func on physical device
net.close()	Commit state changes

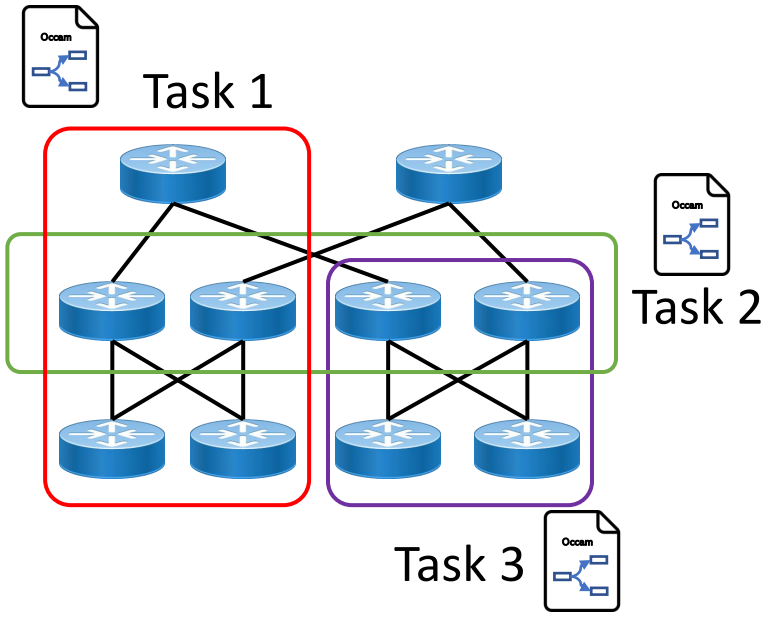
Occam programming APIs

```
# device maintenance
dc1pod3=net("dc1.pod3.*")
dc1pod3.set("DEVICE_STATUS",
   "UNDER_MAINTENANCE")
dc1pod3.apply(f_push)
dc1pod3.close()
```

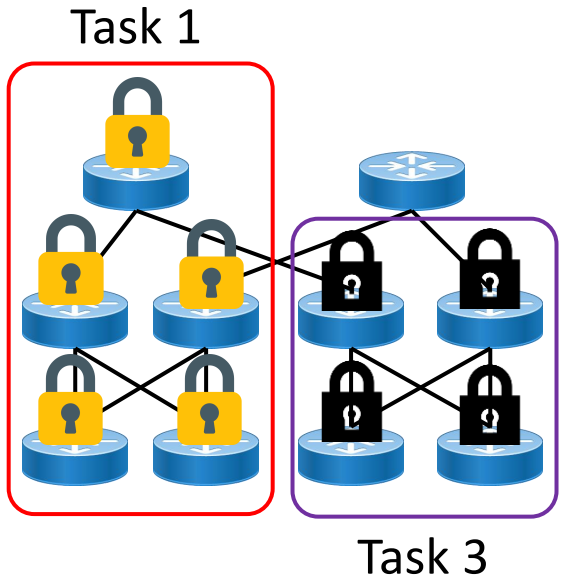
Ex: Device maintenance for a pod

- Regular expression (regex) to construct network object
- Capture the database read/write access with get()/set()
- Capture the device function execution with apply()
- Provides task semantics for Occam runtime

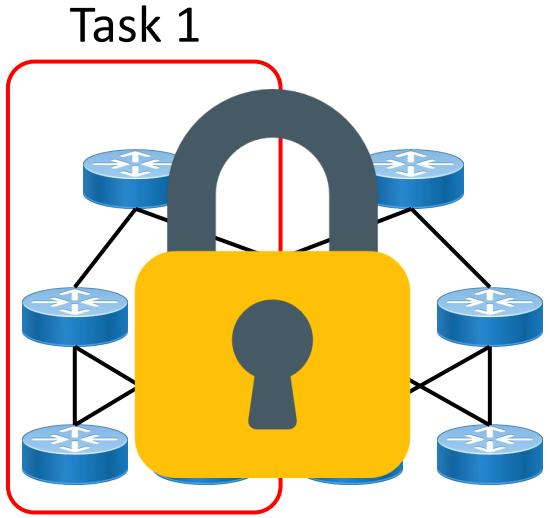
How to ensure consistent device state changes?



Devices requested by tasks 1-3



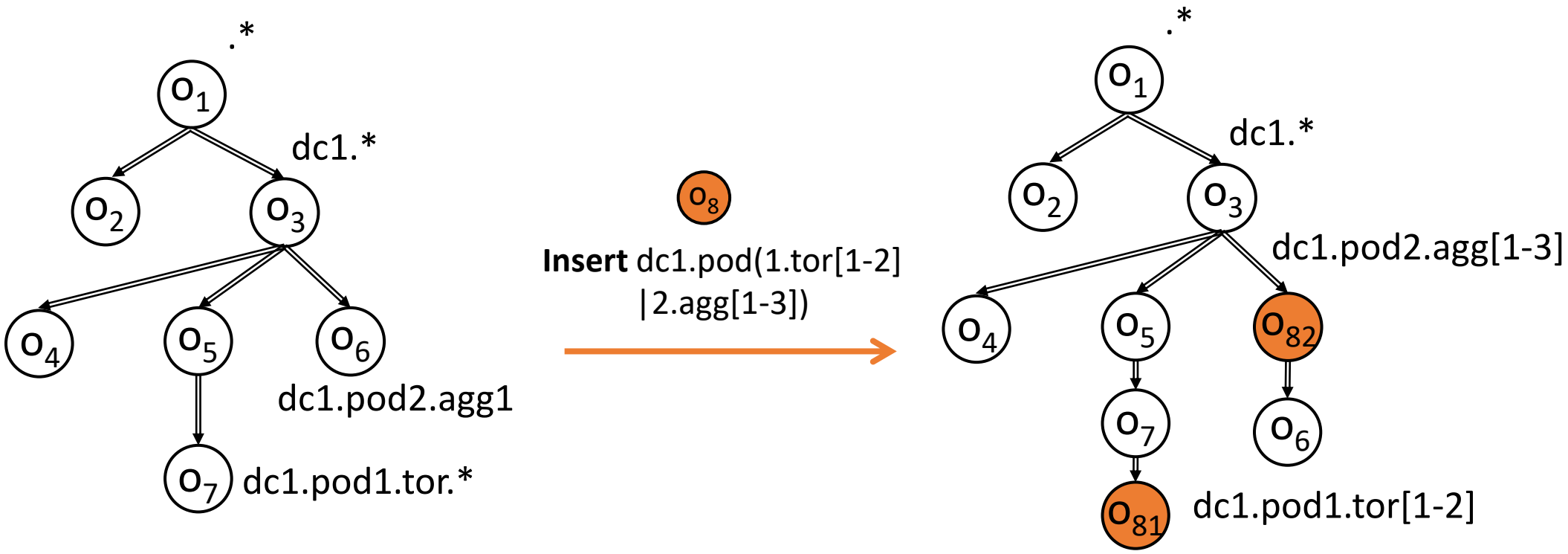
Per-device lock



Per-datacenter lock

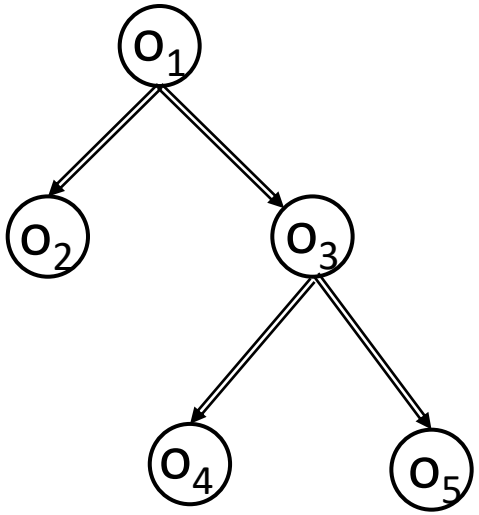
- Different tasks can request to change the same set of devices
  - Could cause inconsistent state changes
- Per-device lock: Too fine-grained, high lock management overhead
- Per-datacenter lock: Too coarse-grained, limited parallelism

Regex-based multi-granularity locking

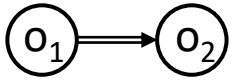


- The locking unit is the network object (scoped by regex)
  - Varied sizes based on the management task
- A regex tree is used to encode containment relationships

Regex-based multi-granularity locking (cont'd)

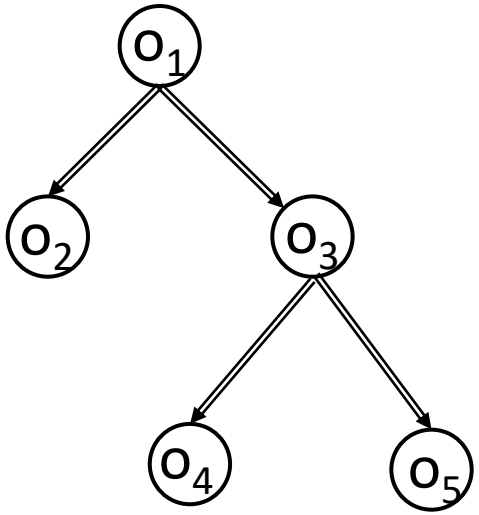



Object

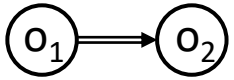


o_1 contains o_2

Regex-based multi-granularity locking (cont'd)



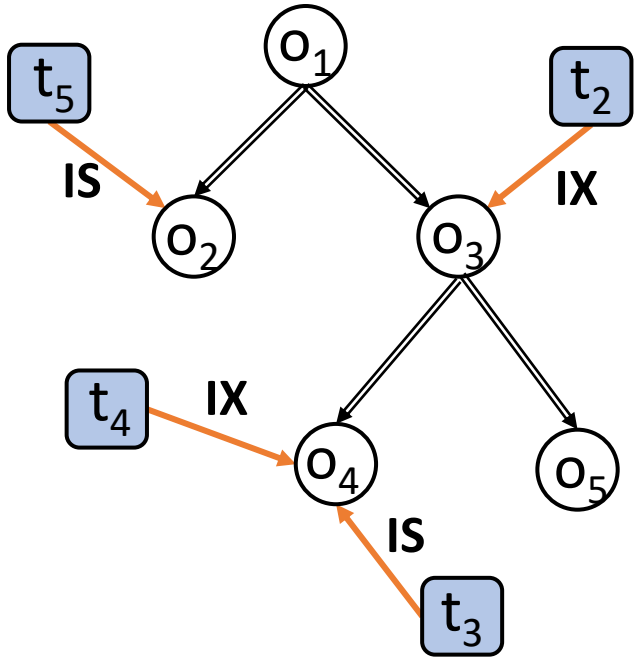

Object

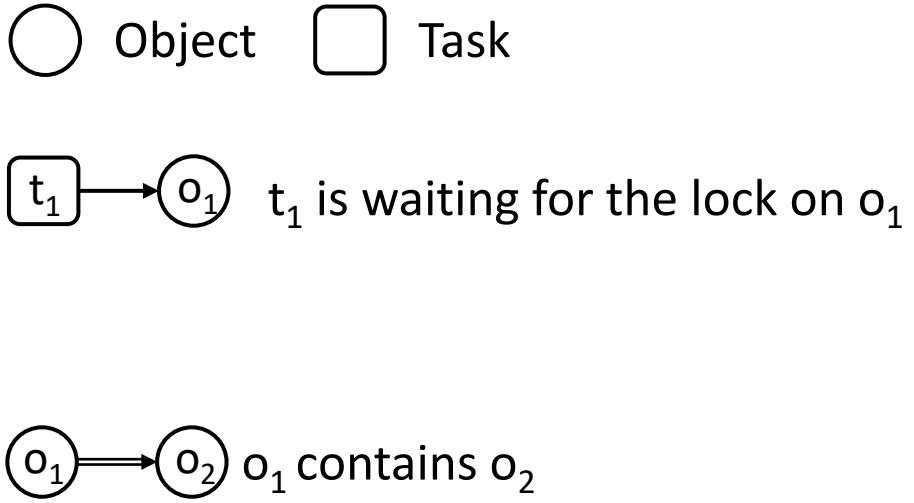


o_1 contains o_2

- The regex tree is enhanced with lock dependency
  - X/S: Exclusive/share locks
  - IX/IS: Intentional exclusive/share locks

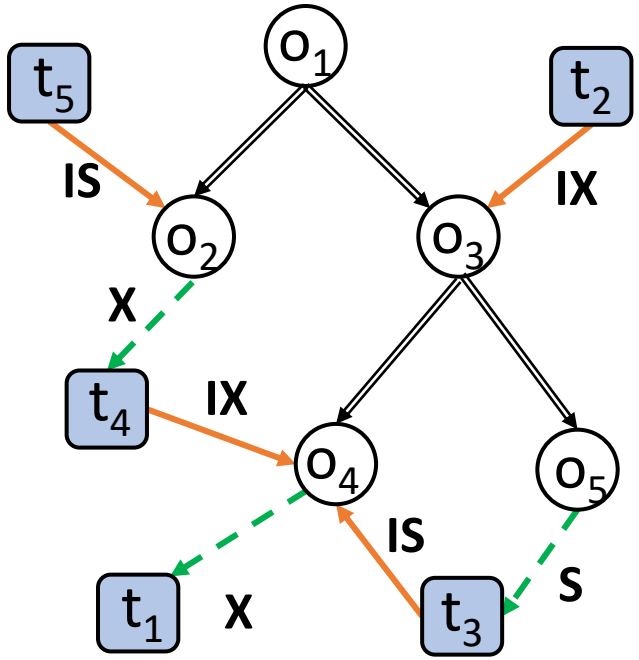
Regex-based multi-granularity locking (cont'd)

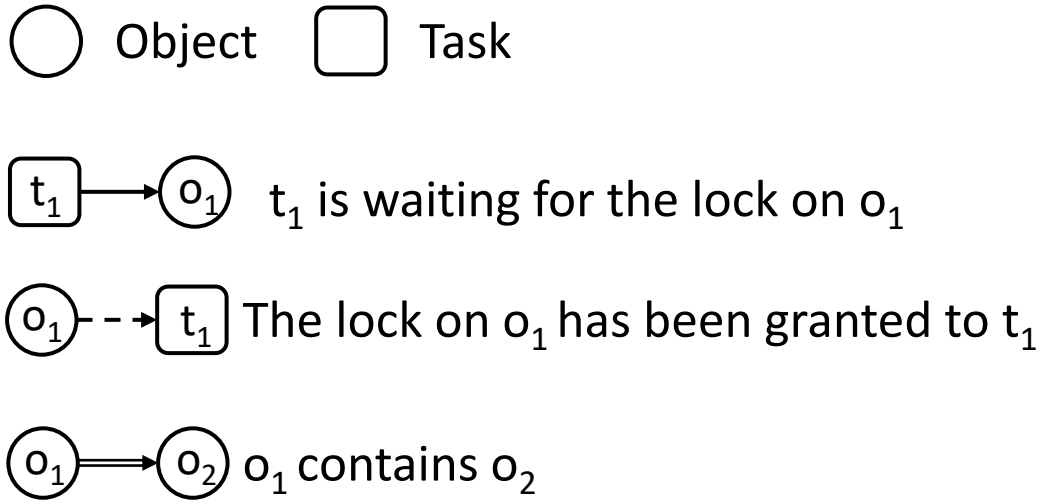




- The regex tree is enhanced with lock dependency
  - X/S: Exclusive/share locks
  - IX/IS: Intentional exclusive/share locks

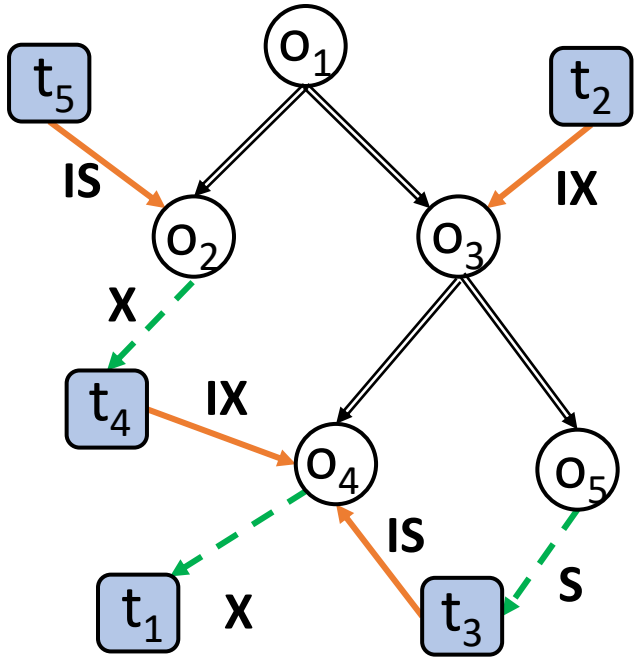
Regex-based multi-granularity locking (cont'd)

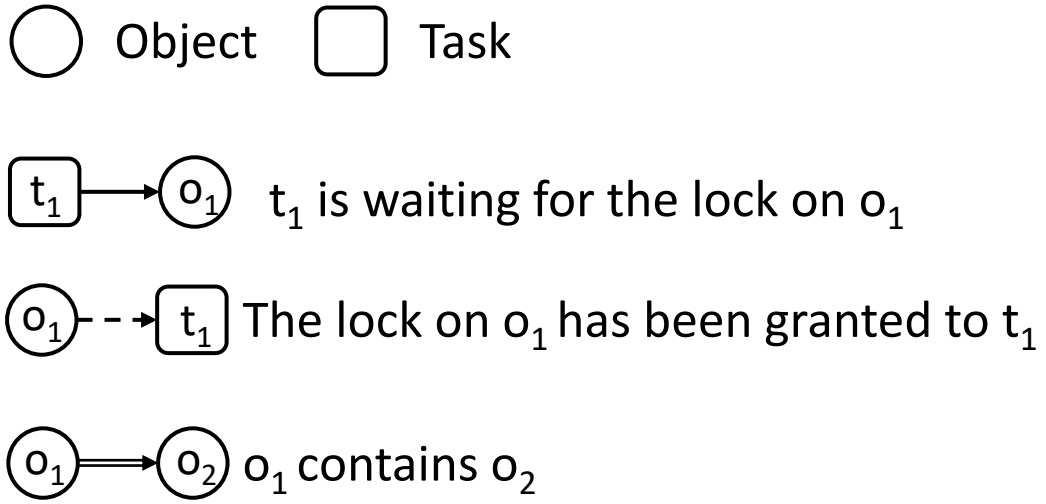




- The regex tree is enhanced with lock dependency
  - X/S: Exclusive/share locks
  - IX/IS: Intentional exclusive/share locks

Regex-based multi-granularity locking (cont'd)



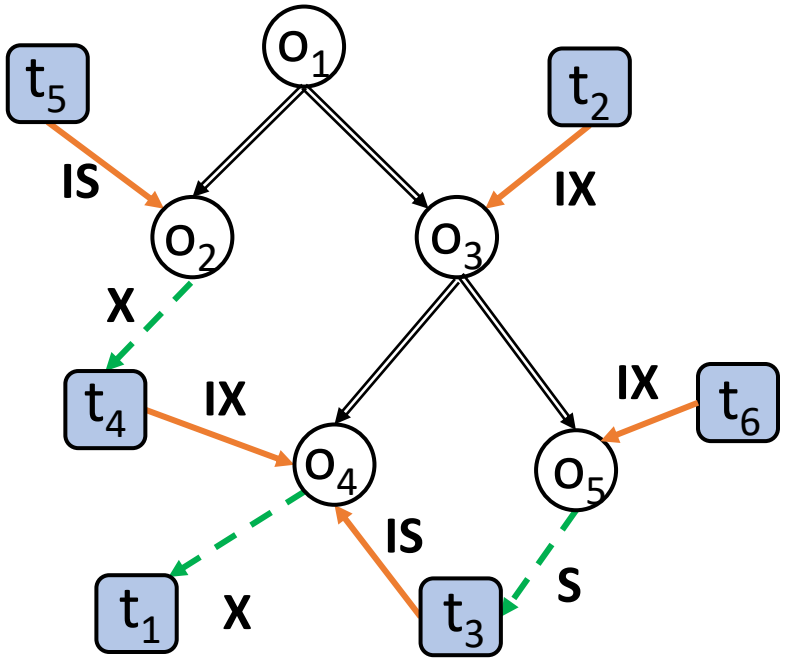


- The regex tree is enhanced with lock dependency
  - X/S: Exclusive/share locks
  - IX/IS: Intentional exclusive/share locks
- Tasks are scheduled based on this dependency graph

Contention-aware scheduling

- Occam supports Largest Dependency Set First (LDSF) scheduling
  - Certain tasks may be blocking more tasks than others
  - Prioritizing them could simultaneously enable more tasks to proceed

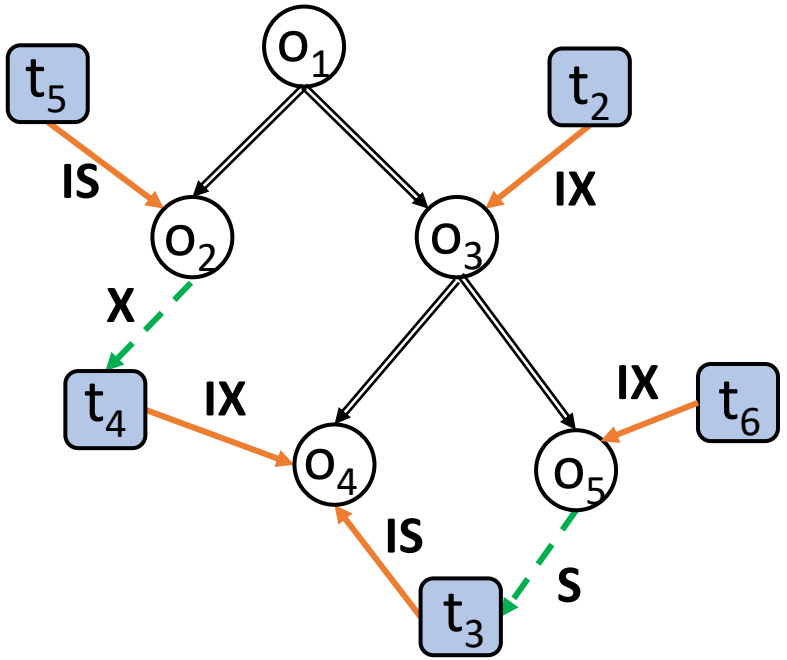
Contention-aware scheduling



When **t1** releases **o4**, should **t3** or **t4** be scheduled?

- Occam supports Largest Dependency Set First (LDSF) scheduling
  - Certain tasks may be blocking more tasks than others
  - Prioritizing them could simultaneously enable more tasks to proceed

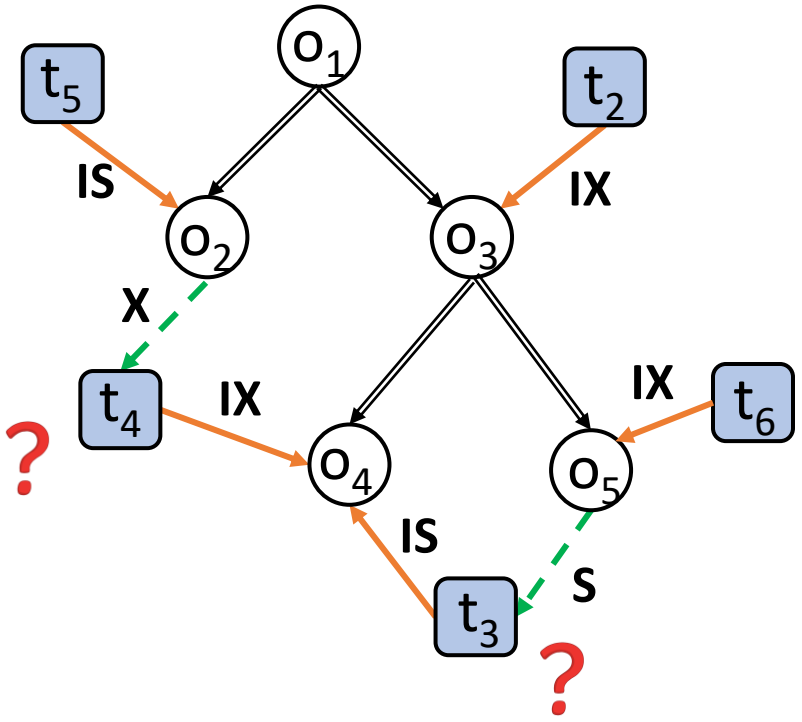
Contention-aware scheduling



When **t1** releases **o4**, should **t3** or **t4** be scheduled?

- Occam supports Largest Dependency Set First (LDSF) scheduling
  - Certain tasks may be blocking more tasks than others
  - Prioritizing them could simultaneously enable more tasks to proceed

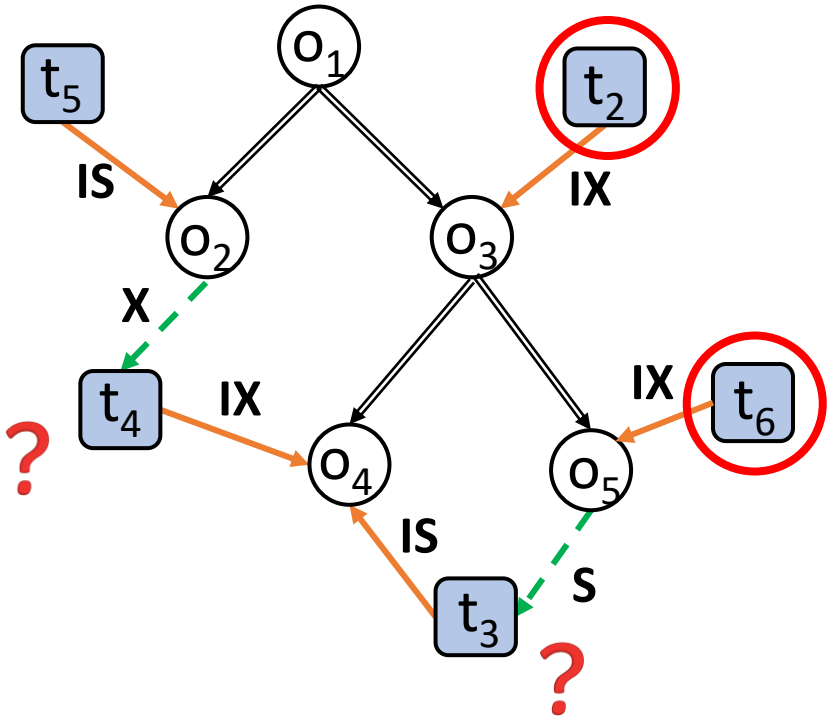
Contention-aware scheduling



When **t1** releases **o4**, should **t3** or **t4** be scheduled?

- Occam supports Largest Dependency Set First (LDSF) scheduling
  - Certain tasks may be blocking more tasks than others
  - Prioritizing them could simultaneously enable more tasks to proceed

Contention-aware scheduling

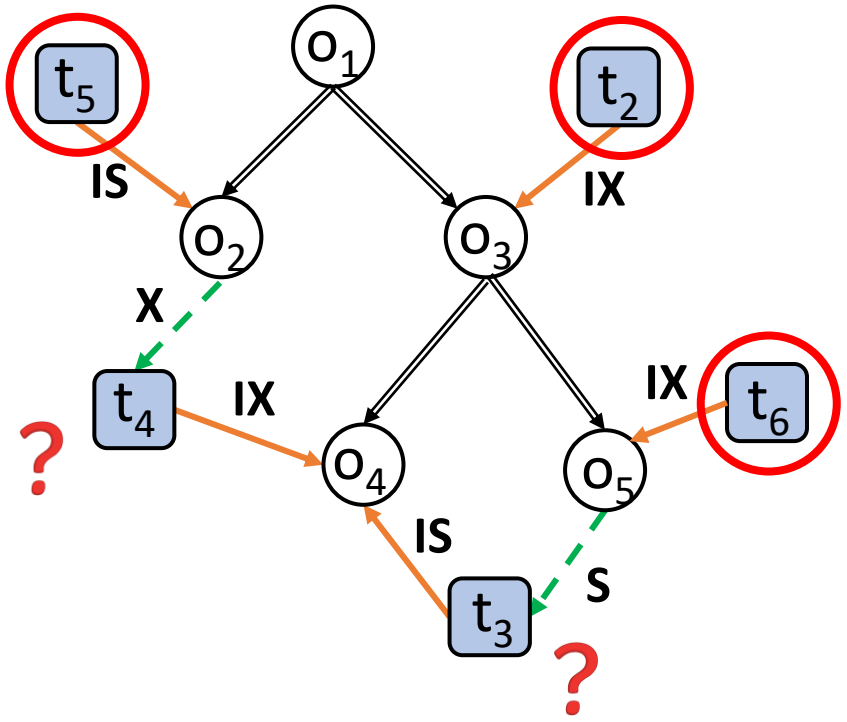


When **t1** releases **o4**, should **t3** or **t4** be scheduled?

t3 blocks **t2** and t6 because of **S** lock on **o5**

- Occam supports Largest Dependency Set First (LDSF) scheduling
  - Certain tasks may be blocking more tasks than others
  - Prioritizing them could simultaneously enable more tasks to proceed

Contention-aware scheduling



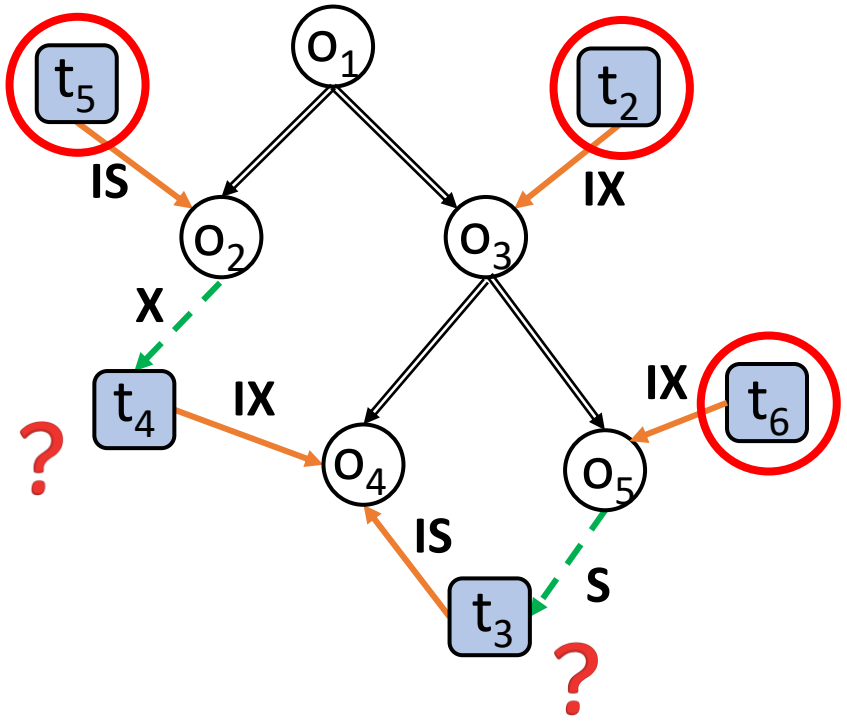
When **t1** releases **o4**, should **t3** or **t4** be scheduled?

t3 blocks **t2** and t6 because of **S** lock on **o5**

t4 blocks **t5** because of **X** lock on **o2**

- Occam supports Largest Dependency Set First (LDSF) scheduling
  - Certain tasks may be blocking more tasks than others
  - Prioritizing them could simultaneously enable more tasks to proceed

Contention-aware scheduling



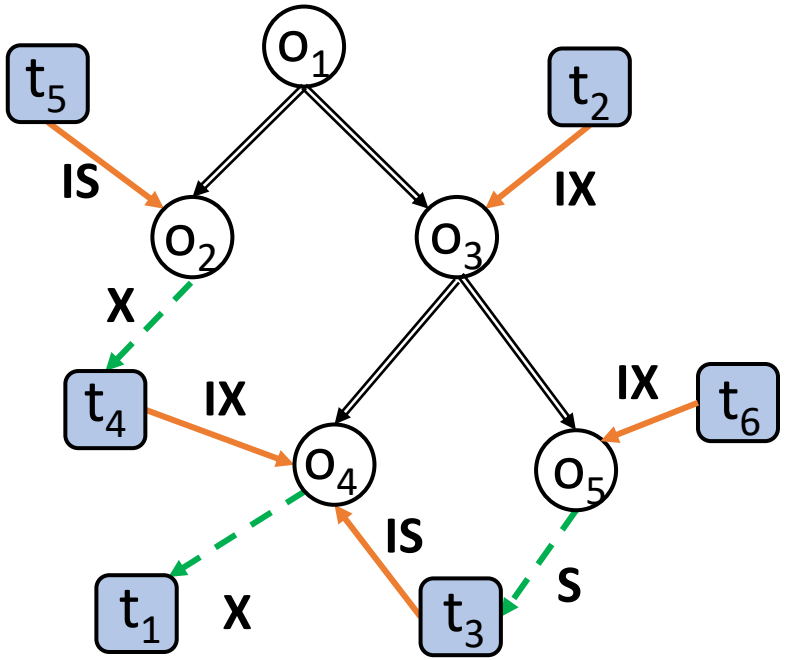
When **t1** releases **o4**, should **t3** or **t4** be scheduled?

t3 blocks **t2** and t6 because of **S** lock on **o5**

t4 blocks **t5** because of **X** lock on **o2**

- Occam supports Largest Dependency Set First (LDSF) scheduling
  - Certain tasks may be blocking more tasks than others
  - Prioritizing them could simultaneously enable more tasks to proceed

Contention-aware scheduling



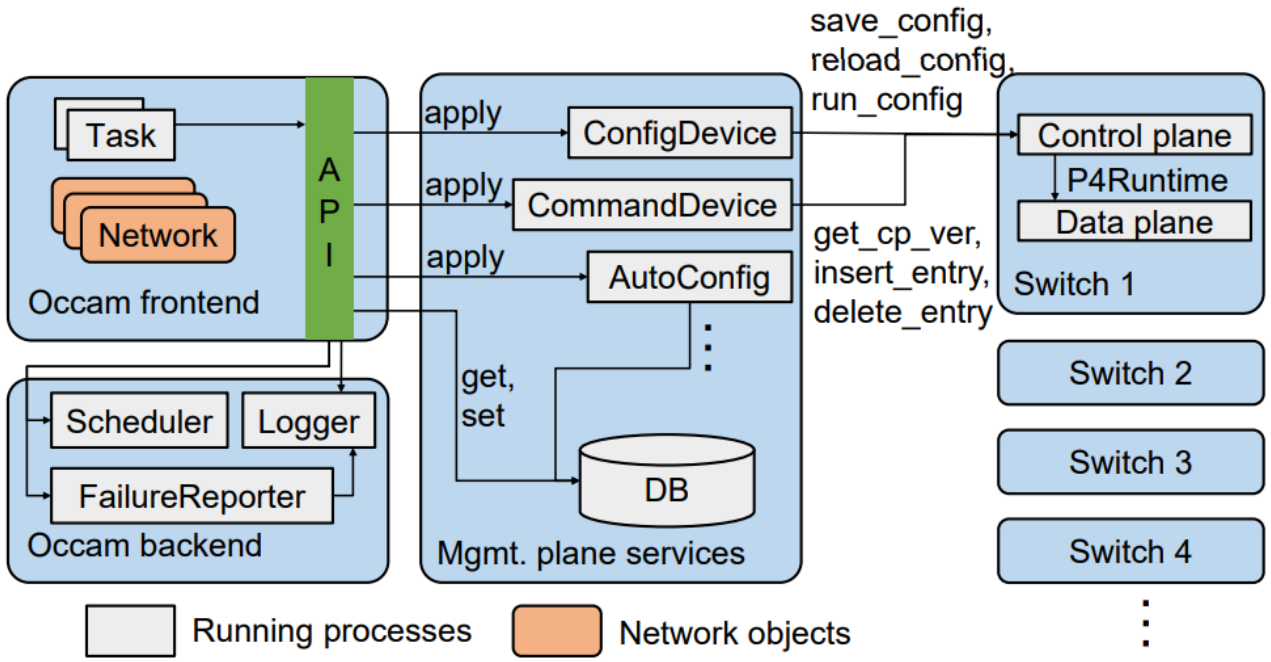
When **t1** releases **o4**, should **t3** or **t4** be scheduled?

t3 blocks **t2** and t6 because of **S** lock on **o5**

t4 blocks **t5** because of **X** lock on **o2**

- Occam supports Largest Dependency Set First (LDSF) scheduling
  - Certain tasks may be blocking more tasks than others
  - Prioritizing them could simultaneously enable more tasks to proceed
- Occam also assists failed tasks recovery (see §6 for details)

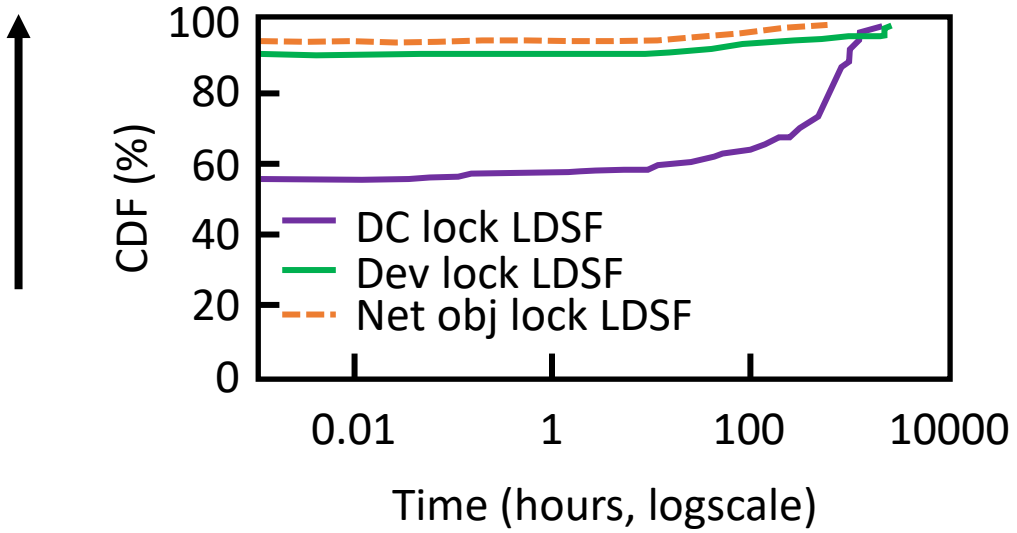
Implementation and evaluation setup



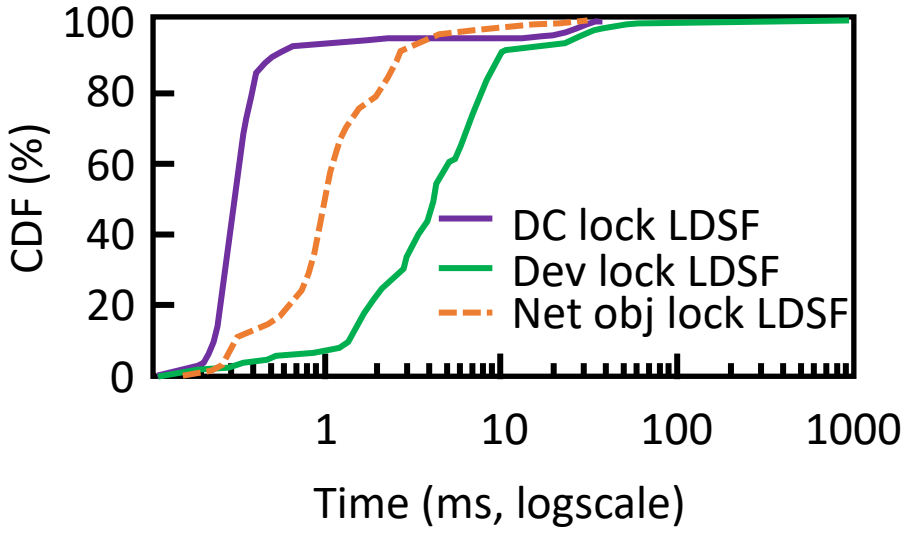
- Mininet-based network management emulation
  - Bmv2 P4 software switches
  - P4Runtime API for state changes
- Large-scale simulation
  - 5-month trace from Meta

Locking and scheduling performance

Higher is better



Task waiting time



Scheduling computation time

- Occam's multi-granularity locking and LDSF scheduling
  - Have minimal waiting time compared to per DC/device locking
  - With reasonable schedule computation time

Summary

- Motivation: Reliable network management for large-scale data centers

- Occam provides
  - Unified API: Enable network-specific design
  - Multi-granularity locking: Avoid conflicts with multi-granularity locks
  - Contention-aware scheduling: Minimize waiting time via LSDF scheduling
  - Assisting failure recovery: Generate plans when failure happens

- Implementation and evaluation:
  - Open-sourced Occam emulator and simulator
  - Minimized waiting time with reasonable scheduling overhead