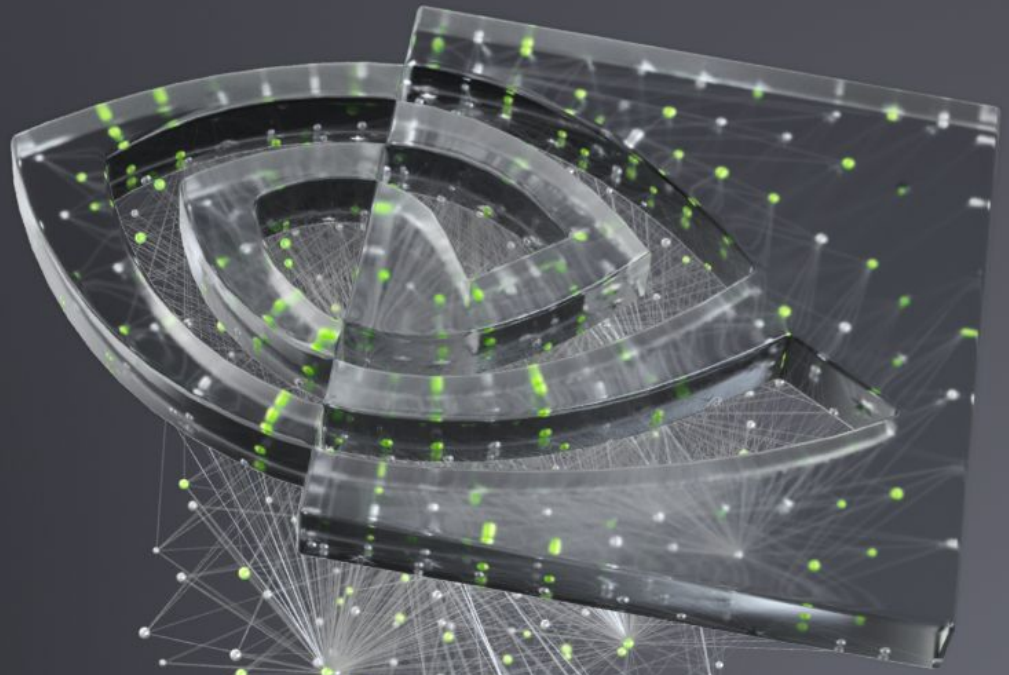


OVN-Northd Incremental Processing in C

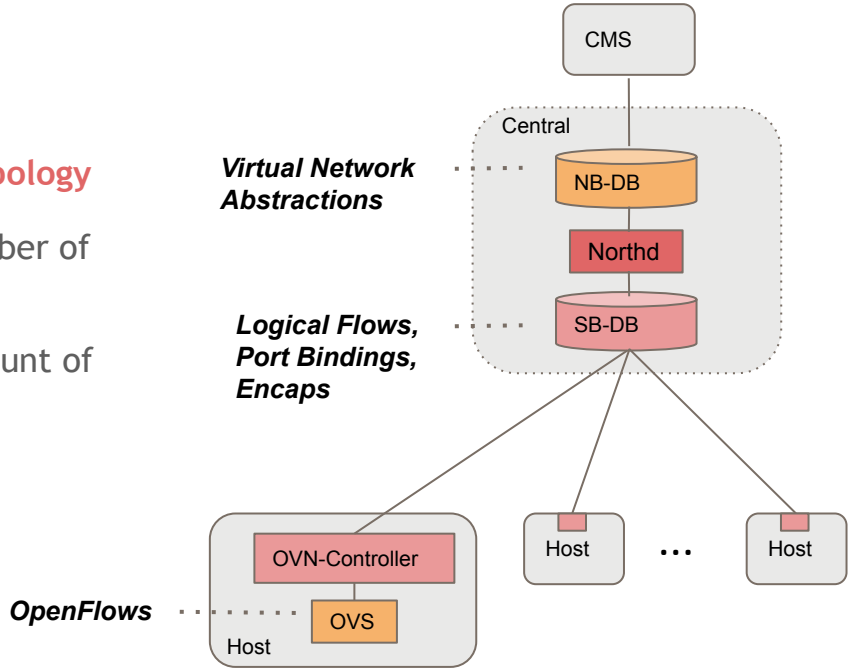
Methodology, Achievements, and Challenges

Han Zhou
NVIDIA
OVSCON 2023

OVN Control Plane

Scale Challenges Overview

- Bottlenecks
  - Northd
    - Processes large size of logical topology
  - SB-DB
    - JSON-RPC sessions for a large number of hosts
  - OVN-Controller
    - Processes and generates huge amount of flows



Incremental Processing

Analogy of Materialized Views in A Relational DB

Table: Departments

DepartmentID	DepartmentName	ManagerID
1	IT	101
2	Marketing	102

Table: Employees

EmployeeID	EmployeeName	DepartmentID	Position
101	John Doe	1	Software Engineer
102	Jane Smith	2	Marketing Specialist
103	Alex Johnson	1	Network Engineer

Join

View: Employees & Departments

EmployeeName	DepartmentName	Position
John Doe	IT	Software Engineer
Jane Smith	Marketing	Marketing Specialist
Alex Jonnson	IT	Network Engineer

Incremental Processing

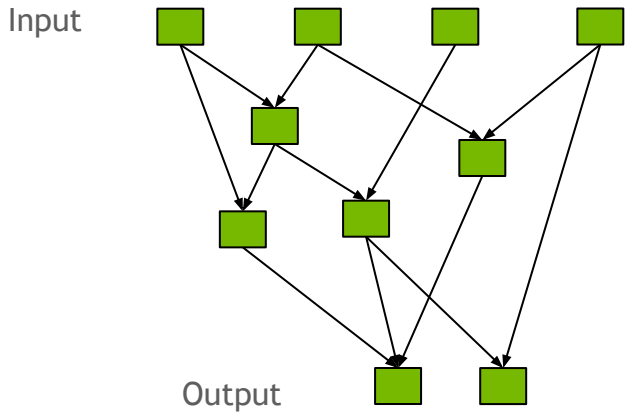
Two approaches

- DDlog implementation (paused)
  - Domain language
  - Always incremental processing
  - Slow startup/initial processing
  - Ensures correctness but debugging can be challenging

- C implementation
  - Incremental processing when necessary
    - only when necessary!
  - Falls back to recompute for infrequent changes
  - Complexity increases quickly with more incremental processing
  - Requires extra effort for correctness, but (hopefully) easier to debug

Incremental Processing Engine

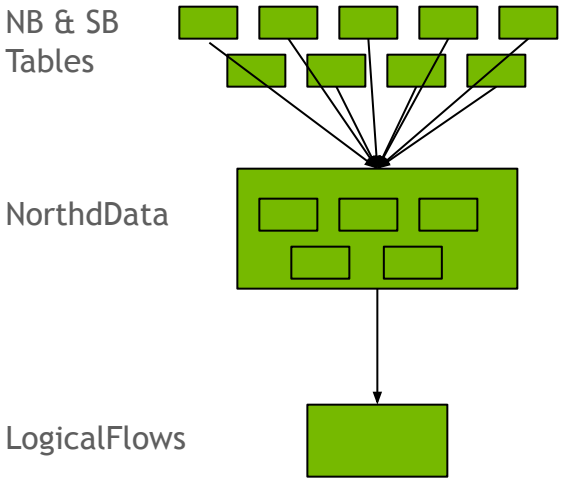
- Things to consider when implementing an I-P engine node (same for code review)
  - Data of the node
  - Dependency
  - How is the data computed
  - How are the input changes handled
  - What changes are tracked
  - How are the changes handled by its children



Initial State

No change, no compute.
Any change, recompute.

- NB & SB Tables
  - Input nodes
- Northd Data
  - A single node that captures all intermediate data structures
- LogicalFlows
  - Output of the engine



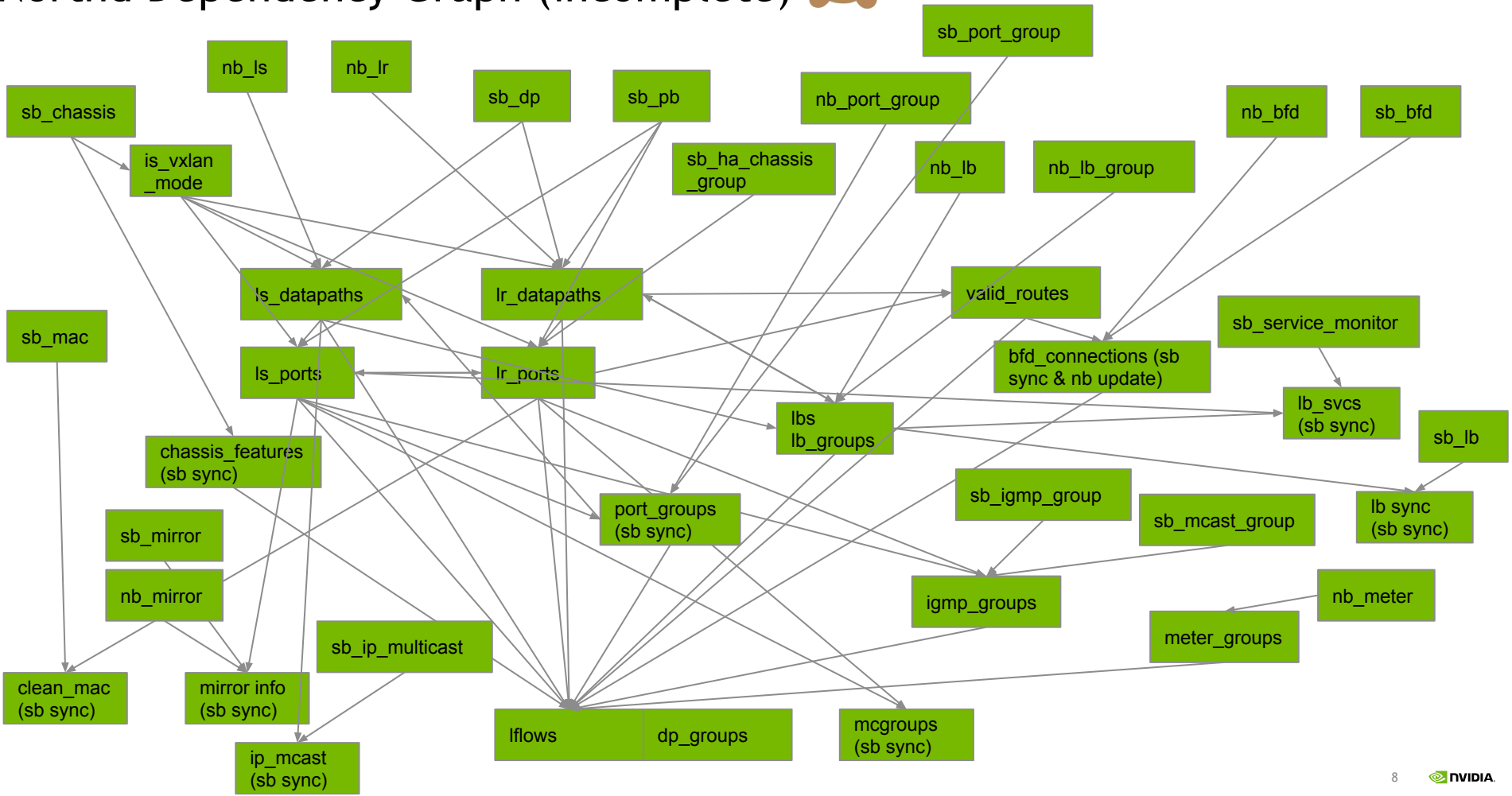
Refactoring

Identify the dependencies

Dependencies must be traceable through function parameters

- Data structure reorganization
  - Separation between logical switches v.s routers, LSPs v.s. LRPs
- Removing global variables
  - Avoid implicit dependencies
- Sanitizing function inputs
  - Avoid passing wrapping structures (hidden dependencies)
  - Remove unnecessary inputs

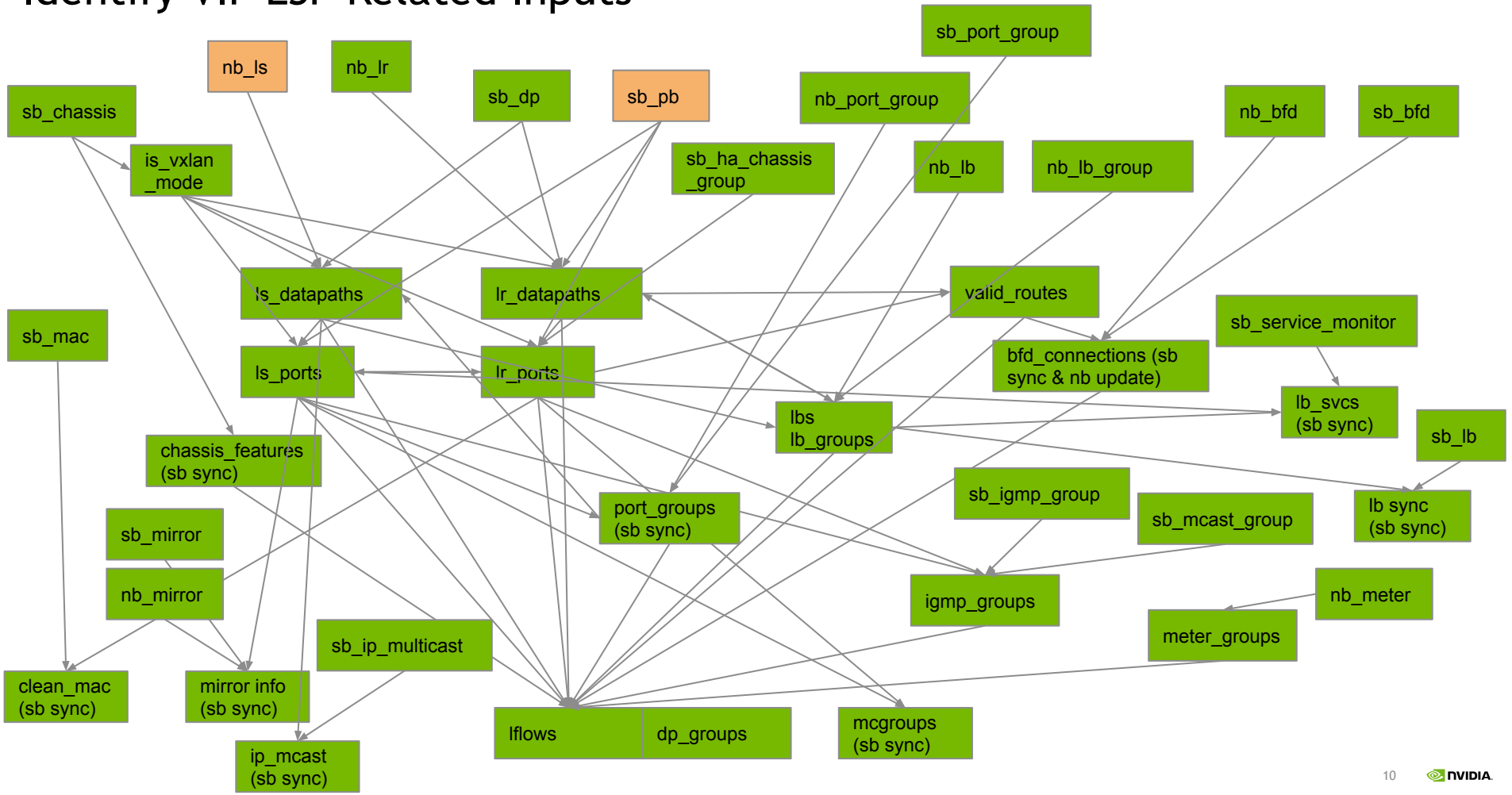
Northd Dependency Graph (incomplete)



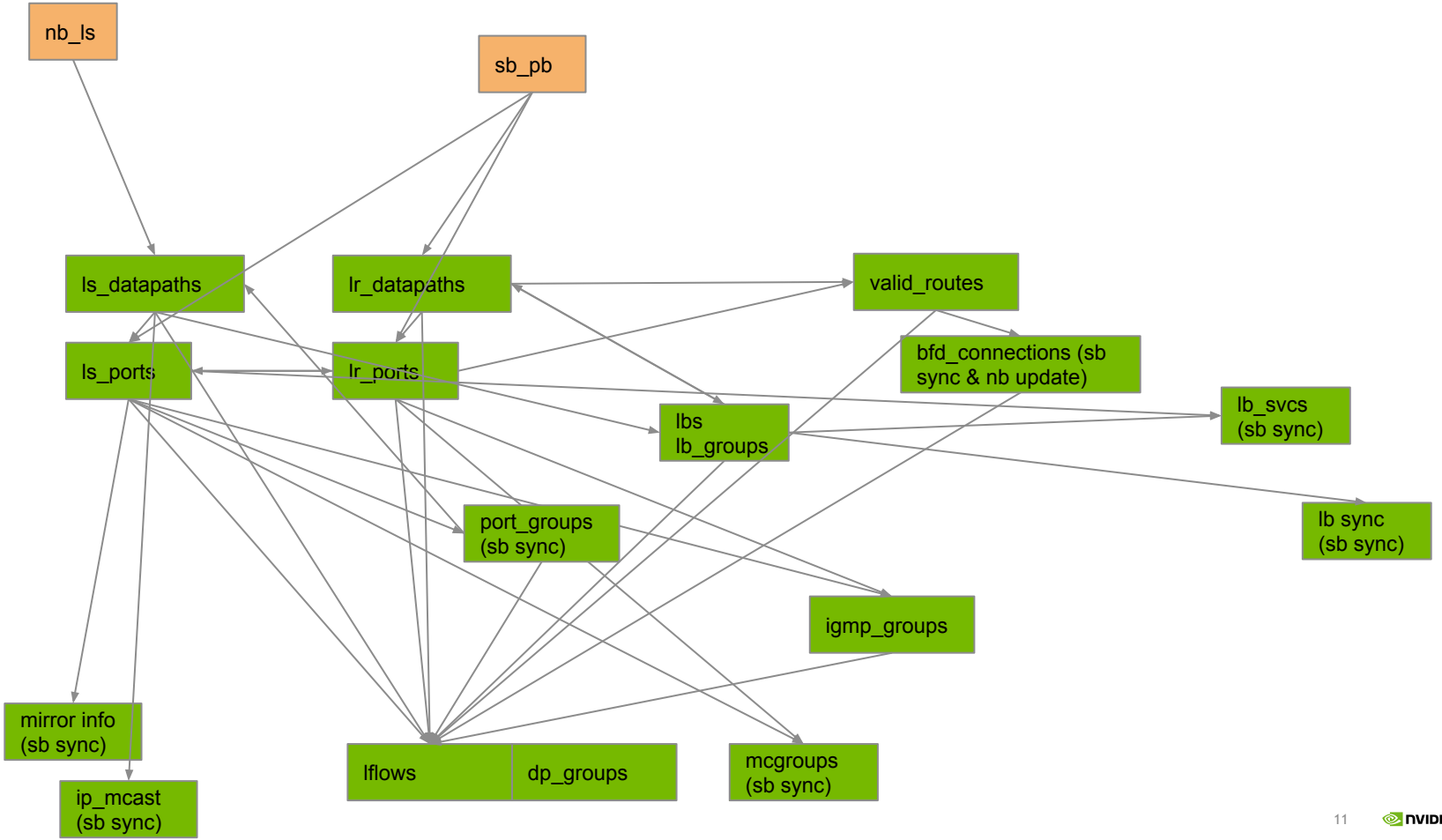
Incremental Development

- Identify the most frequent change
  - e.g. VIF LSP creation and binding
- Top-down dependency analysis
- Implement corresponding nodes and change handlers

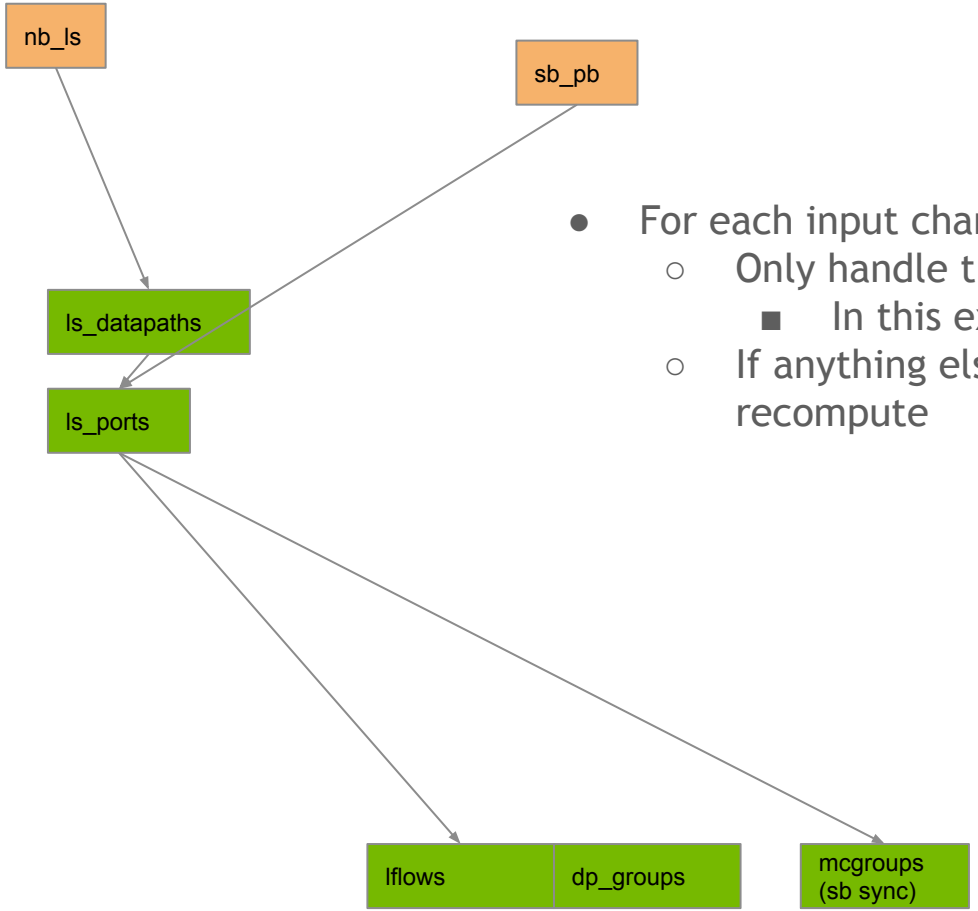
Identify VIF LSP Related Inputs



VIF LSP Related Nodes

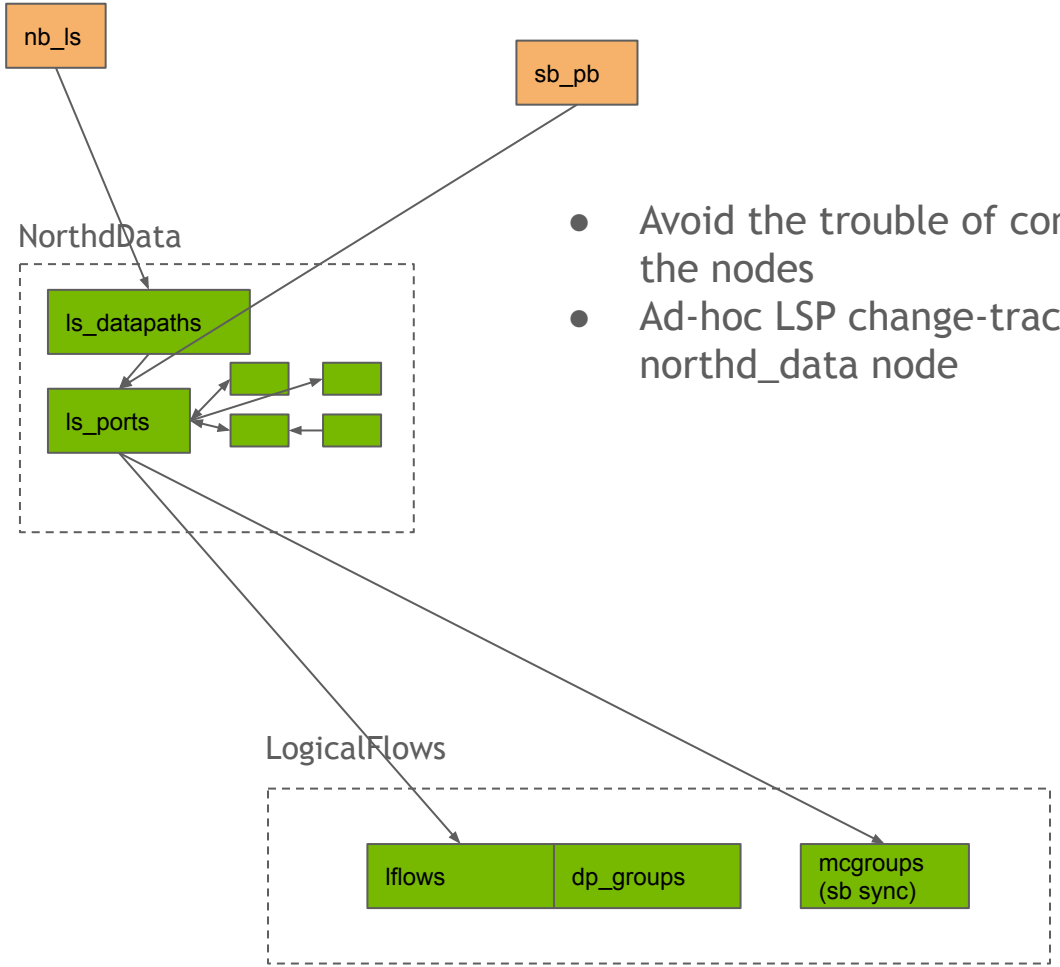


VIF LSP Related Nodes (Filtered)



- For each input change
  - Only handle the specific type of change
    - In this example, only VIF LSP related
  - If anything else is changed, fall back to recompute

VIF LSP Related Nodes (Compromised)



- Avoid the trouble of complex references between the nodes
- Ad-hoc LSP change-tracking inside the northd_data node

Achievements

- Scale test of ovn-k8s topology with 500 nodes x 50 lsp per node
  - Simulated by https://github.com/hzhou8/ovn-test-script
  - Intel(R) Core(TM) i9-7920X CPU @ 2.90GHz, Single thread for northd
  - VIF creation & binding 773ms -> 30ms: more than **95%** reduction (or **20x** faster)
  - Constraints:
    - 1) The LSPs are not service backends
    - 2) The LSPs are not part of any port-group attached to ACLs
- (Dumitru) Port-group I-P
  - Partially removed the constraint 2)
- (Numan) LoadBalancer I-P in NorthdData node
  - **>50%** CPU savings (or **2x** faster) in ovn-heater test for ocp-500-density-heavy (see commit 280bef8b)

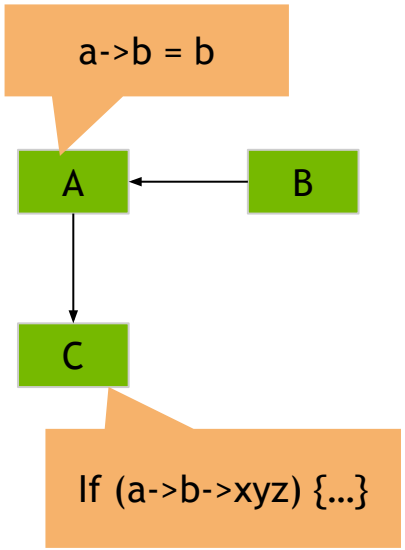
Incremental Processing Disciplines

For Correctness and Maintainability

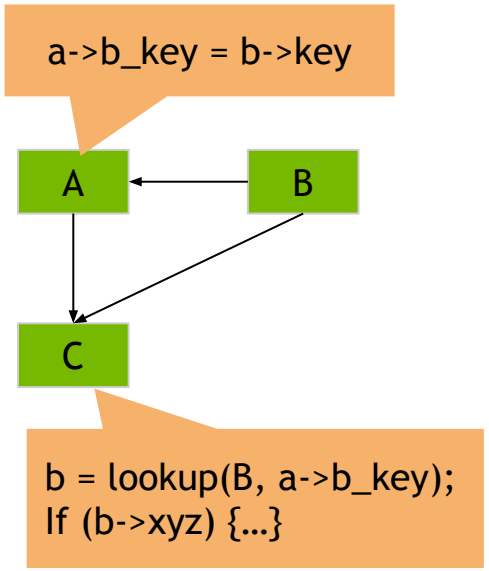
- Input node data access should be read-only.
- Avoid direct pointer references between nodes (implicit dependency).
- Avoid silently ignoring input changes in change-handler.
- Strictly follow recompute logic when implementing change-handlers.
- Fall-back to recompute when unsure.
- Test case: compare I-P result with recompute result.
- Exceptions may exist with good reason, but should be carefully documented.

References Between Nodes

Avoid Implicit Dependencies



❌

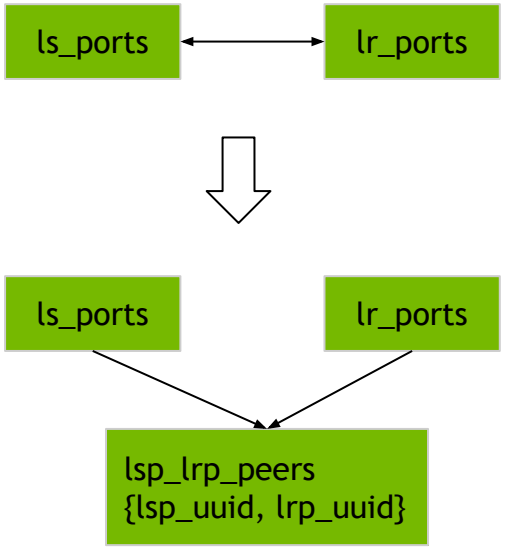


✅

References Between Nodes

Break Circular Dependency

- Example:
  - LSP.peer = LRP
  - LRP.peer = LSP



Multiple Inputs I-P

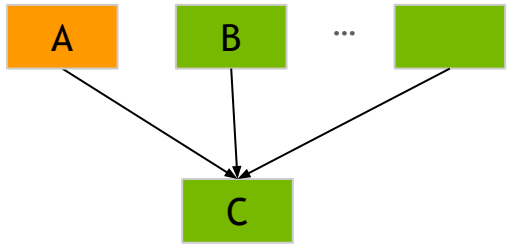
Primary & Secondary Input

- Recompute of C:

```
For each a in A:
    build_c_for_each_a(a, B, …)
```

- Handling “add” of A in C:

```
For each new a:
    build_c_for_each_a(a, B, …)
```

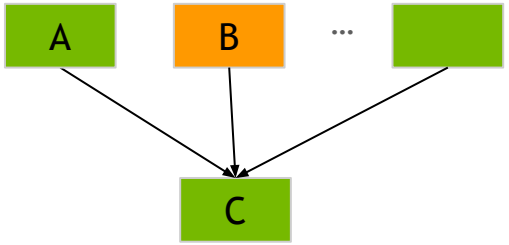


- A is the **primary** input, B is the **secondary** input
- Example: build_lswitch_arp_nd_responder_known_ips(op, meter_groups)

Multiple Inputs I-P

Secondary Input I-P

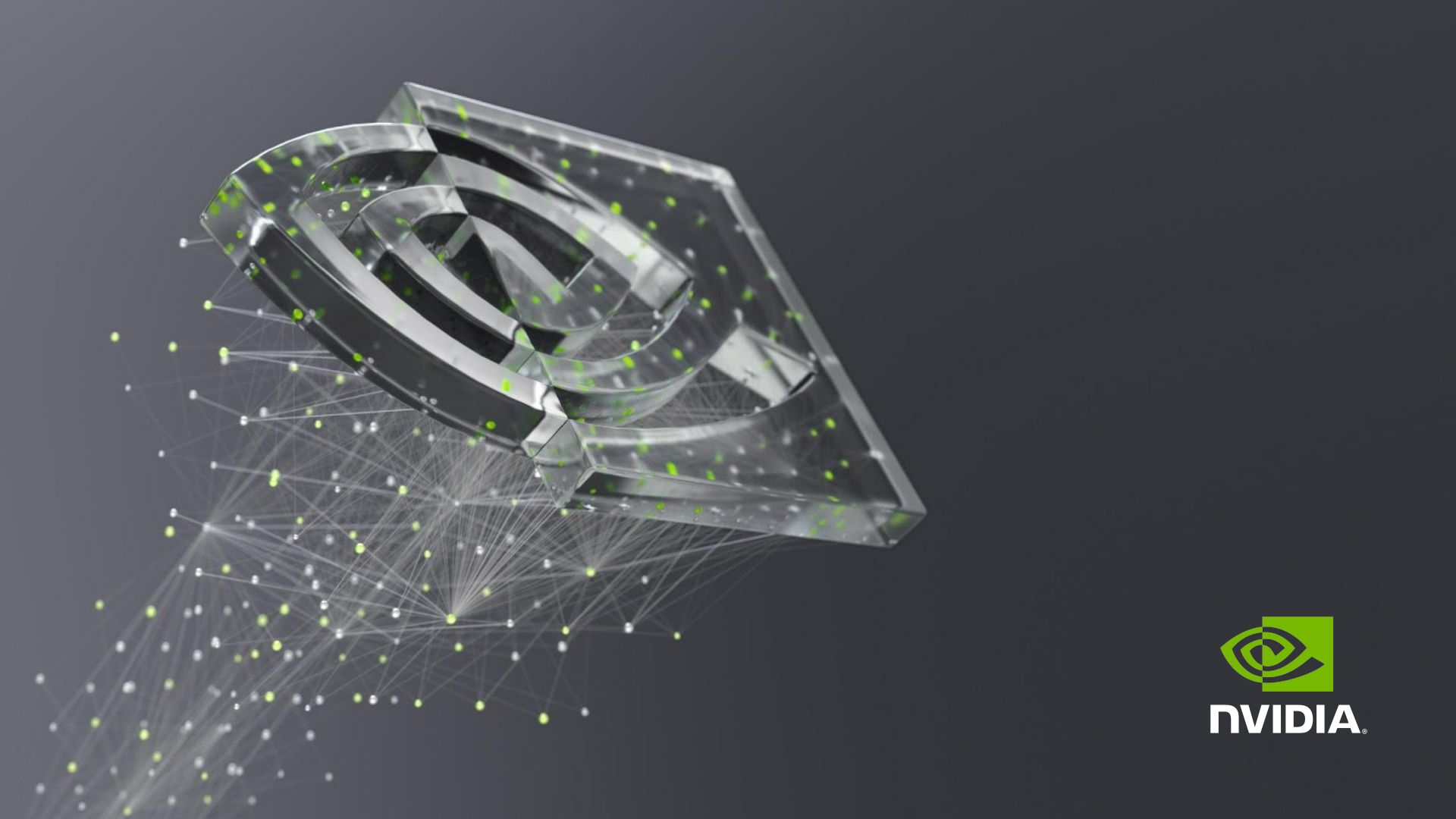
- Handling “add” of B in C:
  - For which objects in A should we call build_c_for_each_a(a, B, …)?
  - Or, should we implement build_c_for_each_b(b, A, …)?



An Example of Incomplete Change Handler

This Is Not Fun

- Example: When the first/last LSP is added/deleted to/from a logical switch, the ARP request flows related to router IPs need to be updated.
- In build_lswitch_rport_arp_req_flow():
  - if (od->n_router_ports != od->nbs->n_ports) { … }
- Missed in the I-P handler
  - LSP add/delete was handled by calling build_xxx_by_lsp(lsp)
  - However, there were no tracked changes to the router ports, no changes were made to the related ARP request flows.
- The fix - check the implicite dependency in change handler:
  - a property “has_only_router_ports” of the lswitch
- Lessons:
  - Ad-hoc implementation is error-prone
  - More testing
  - Code review


NVIDIA