UNIVERSITY OF CALIFORNIA UCRIVERSIDE

Name Space Analysis (NSA): Verification of Named Data Network Data Planes

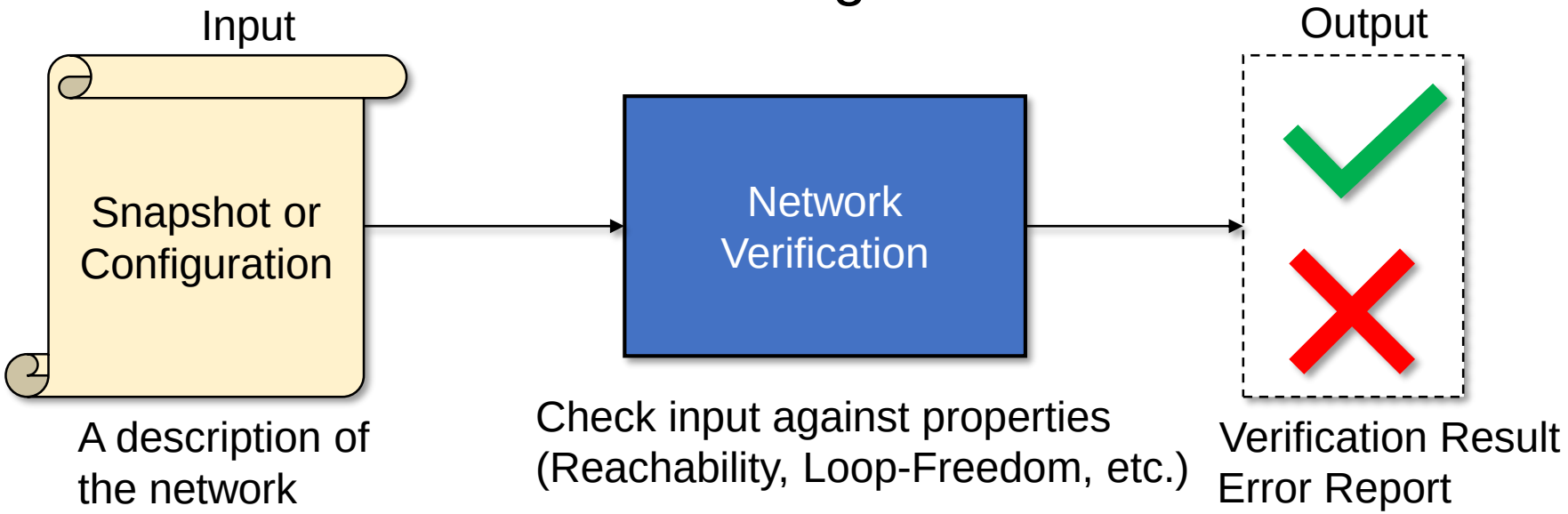
Mohammad Jahanian and K. K. Ramakrishnan

University of California, Riverside

ACM ICN 2019

Network Verification is important

- Network data planes are complex and hard to prove
  - Combination of many interacting protocols and data structures
  - E.g., important questions: Can host A reach host B? Are there any loops? …
- Network verification tries to solve this
  - Formal methods to model a network (state) and specify its properties
  - Tools to automate the verification and generate results

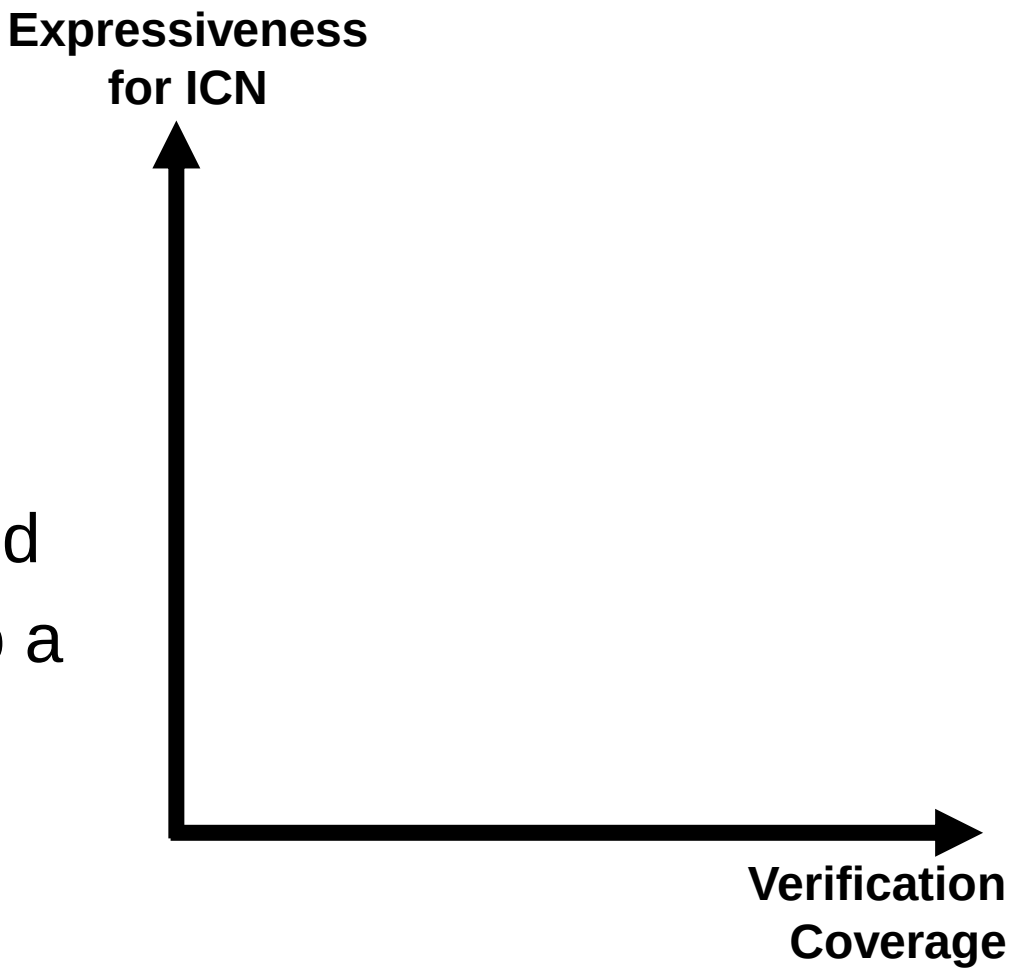


NDN: verification important & needs special attention

- NDN doesn't make things much less complicated!
  - Name-based forwarding strategies, forwarding hints, name trees, etc.
- Complexity grows with scale (e.g., Internet-scale network, large name space); NDN depends on current data plane *state*.
- Verification of NDN data planes important
  - Many operations depend on it (content request, key retrieval, etc.)
- Existing verification tools aren't sufficient
  - They model host-centric (typically, IP-based) networks
  - Not suitable for ICN networks; fundamental differences
    - Different network design: name-based vs address-based.
    - Different intents: host-to-host reachability vs host-to-content.
  - Need to re-visit formal analysis and network verification for NDNs

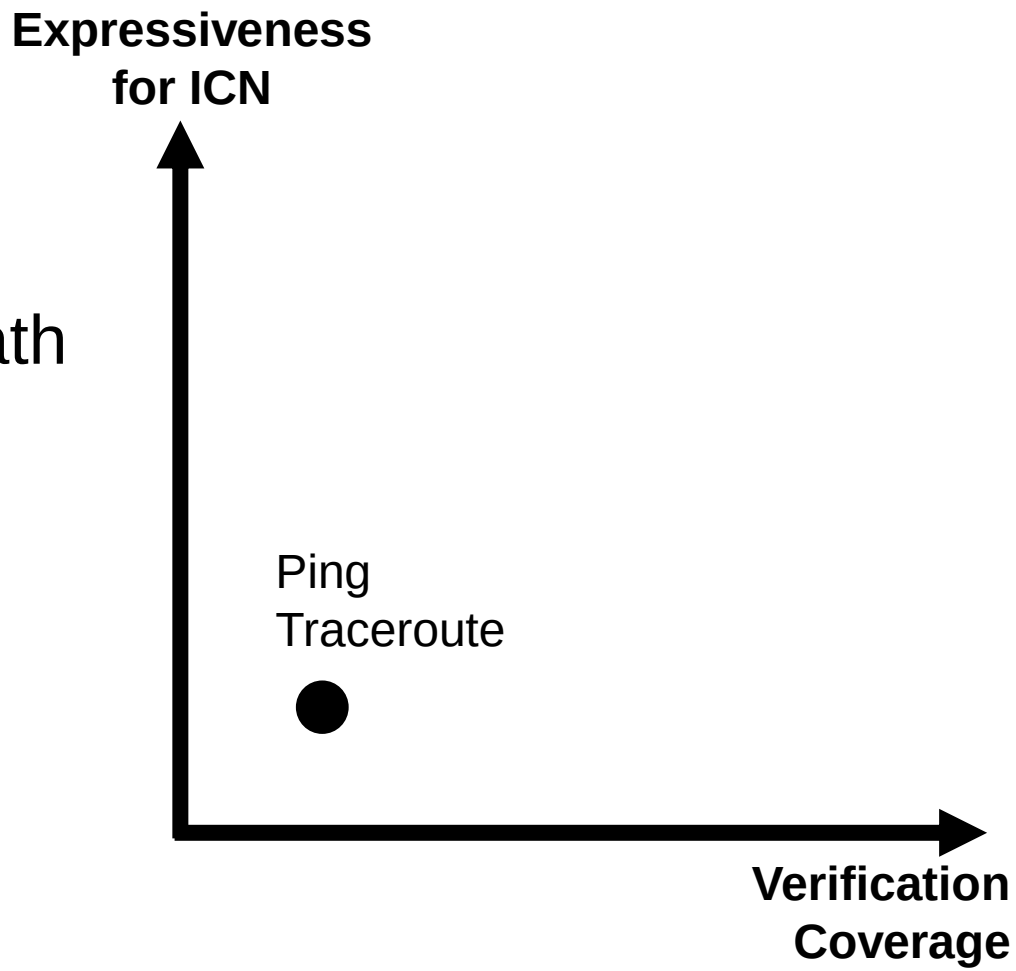
Towards ICN Data Plane Verification

- Expressiveness for ICN
  - Its design and intent
  - ICN is a superset of host-centric communication (e.g., host-to-content)
- Verification Coverage
  - How many packets and their states covered
  - Ranges from a single-packet verification to a whole (single or multiple) data plane verification



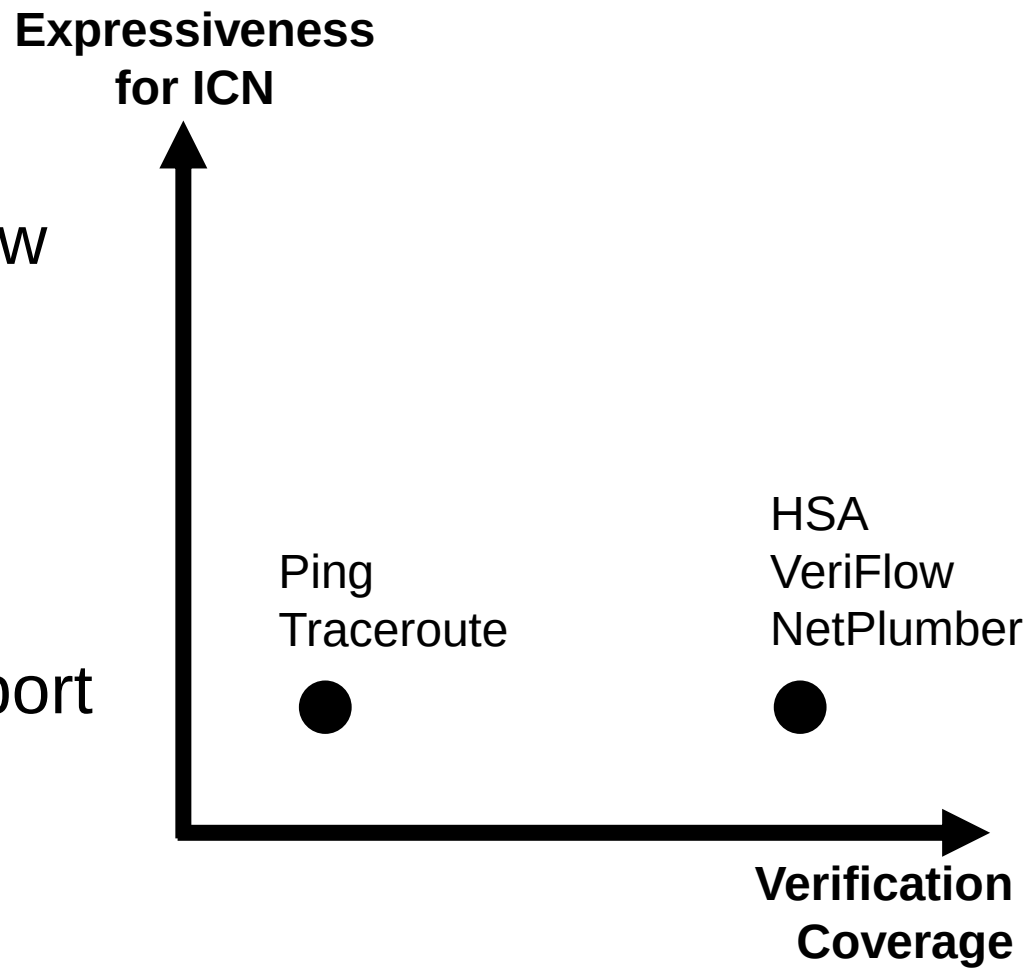
Towards ICN Data Plane Verification

- Simple Ping and Traceroute
  - Classic network diagnostic tools
  - For current IP-based networks
  - Coverage limited to a single packet and path
  - Infeasible (computationally) to cover all possible packets and their possible paths
  - Thus, we need a formal method for high-coverage verification



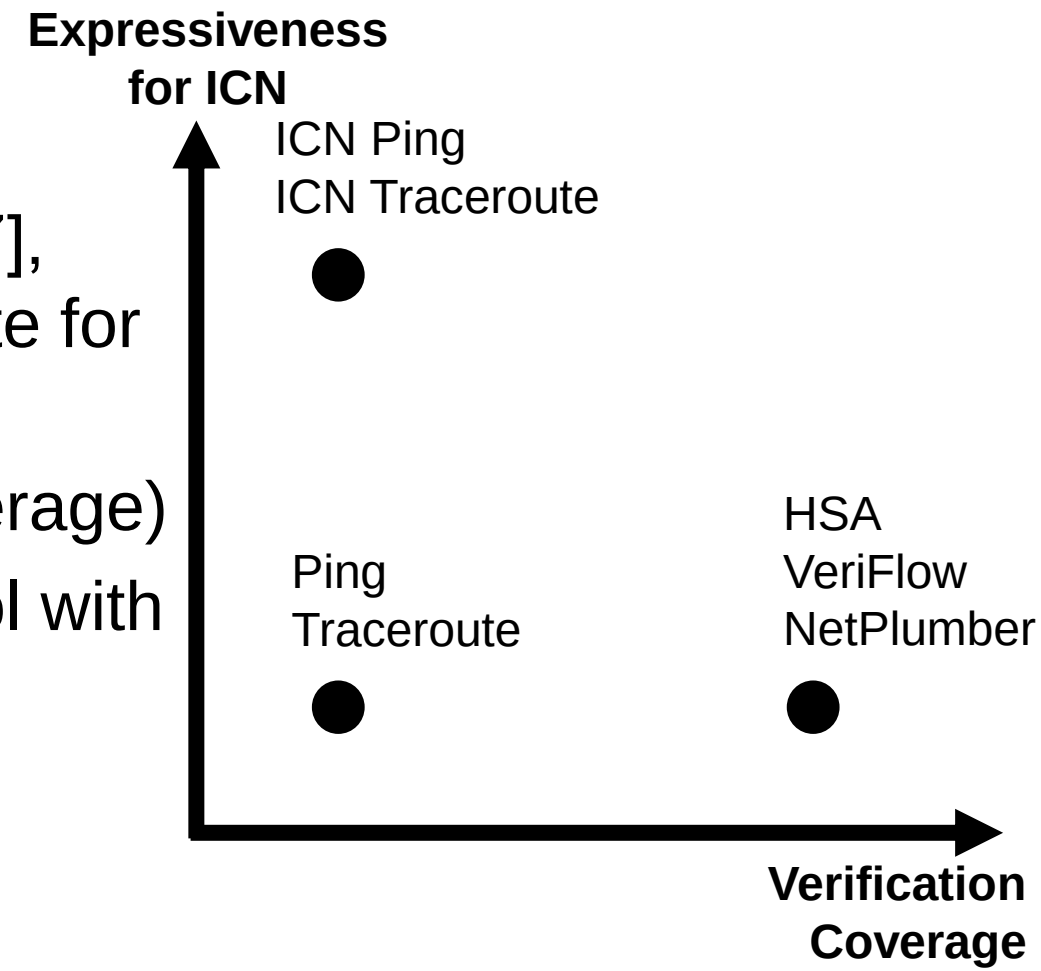
Towards ICN Data Plane Verification

- Current data plane verification tools
  - High coverage for verification
  - Header Space Analysis (NSDI'12), VeriFlow (HotSDN '12), NetPlumber (NSDI'13), Validating Datacenters (Sigcomm'19)…
  - Useful and popular, but insufficient for ICN verification
  - We need a formal method and tool to support ICN design and intents



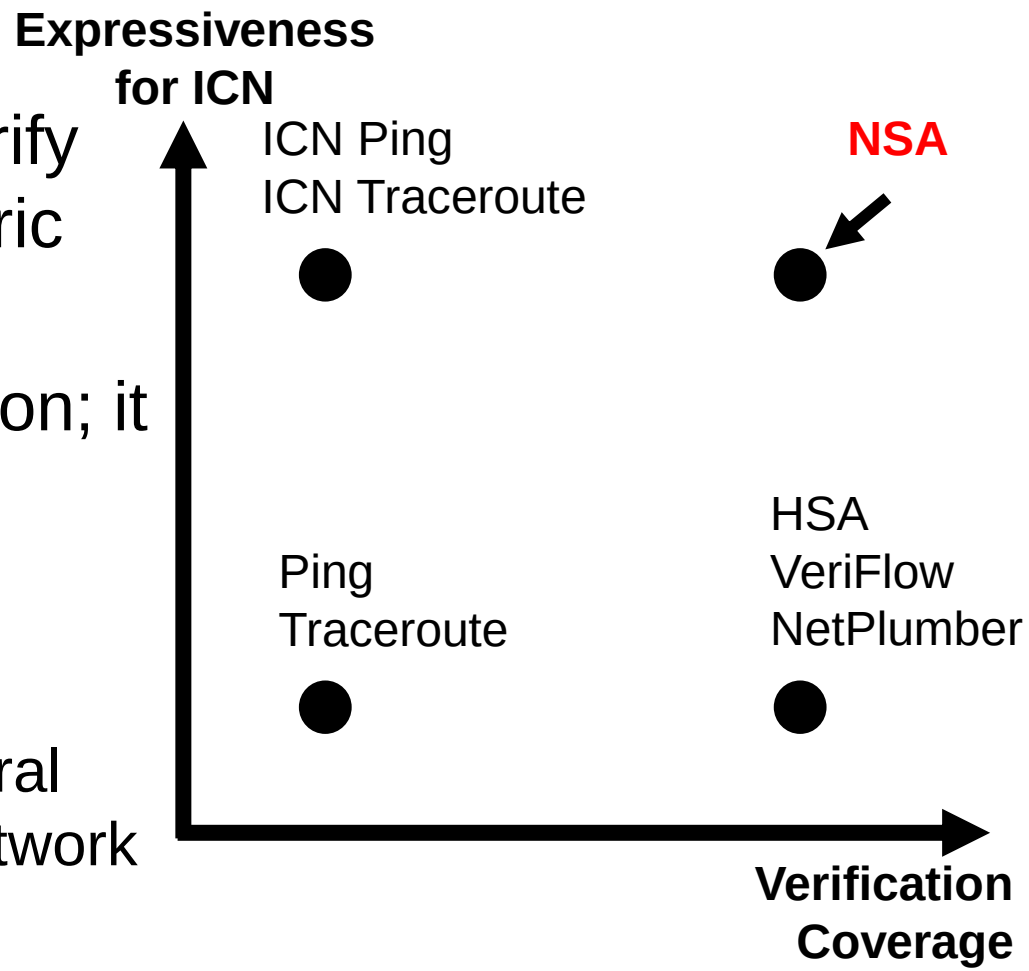
Towards ICN Data Plane Verification

- Ping/Traceroute for ICN/NDN
  - ICN Ping [IETF, ongoing]
  - Contrace [IETF, 2018], NDN-trace [ICN'17], ICN Traceroute [IETF, ongoing], Traceroute for NDN [TR-2017]
  - Useful for limited checks (i.e., limited coverage)
  - But we need a formal NDN verification tool with high coverage!

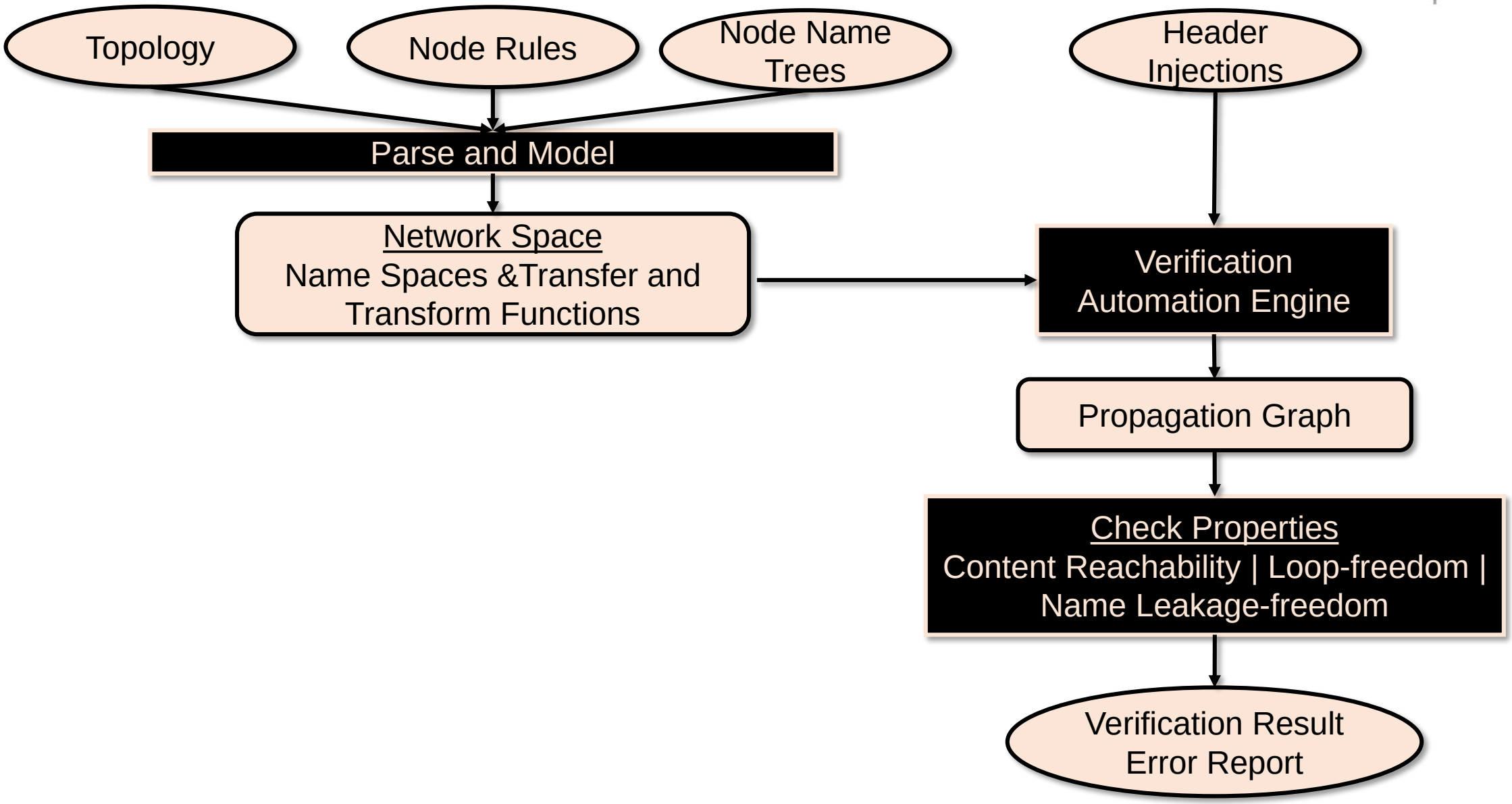


Towards ICN Data Plane Verification

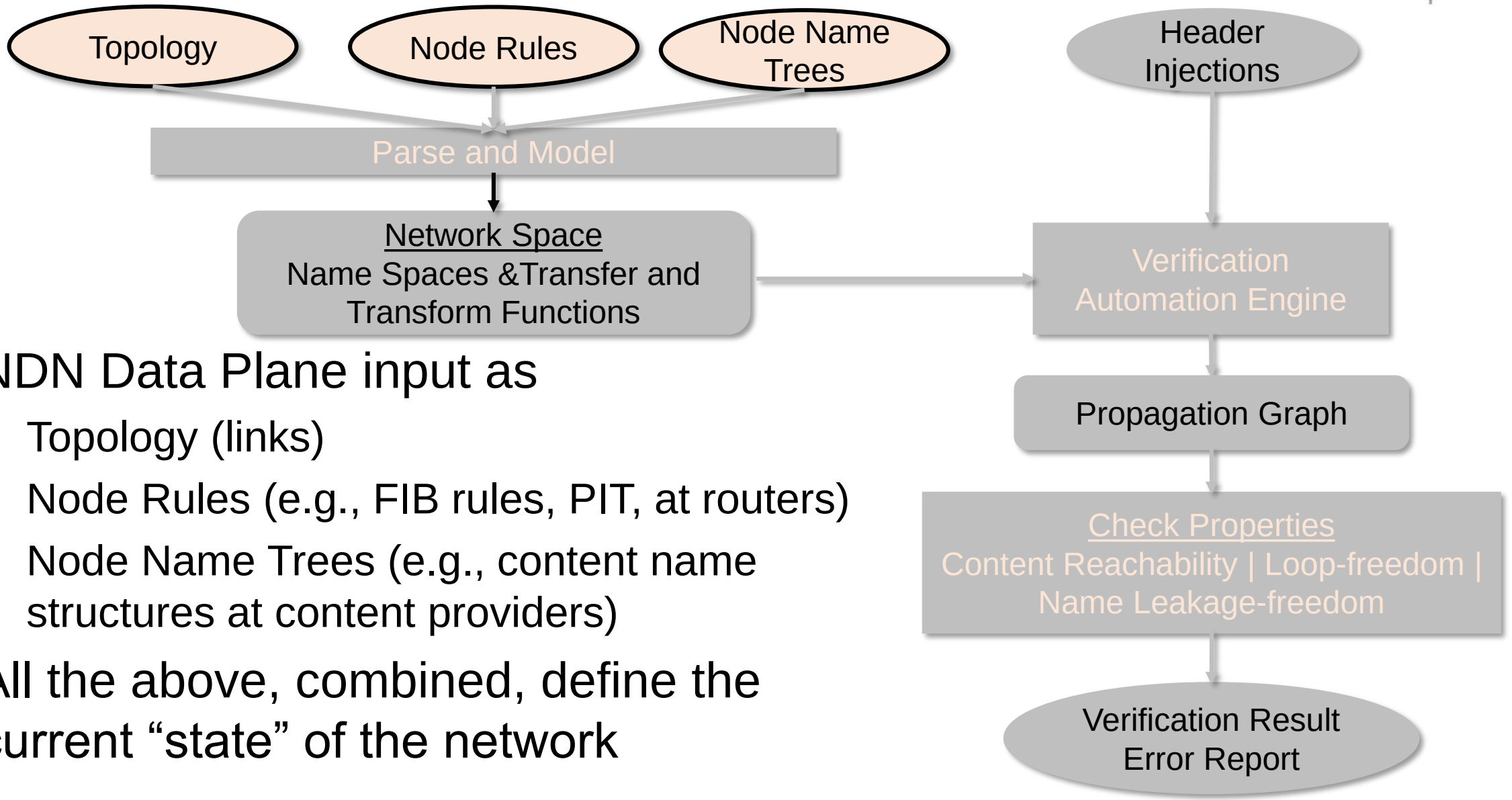
- Name Space Analysis (NSA)
  - A formal method and tool to model and verify NDN data planes against information-centric intents; high verification coverage
  - NSA does not (re-)invent network verification; it extends existing approaches
  - NSA builds on the theory of HSA
    - Uses its building blocks and extends it
    - NSA models named headers; names as integral part of networking, and information-centric network invariants



NSA Overview

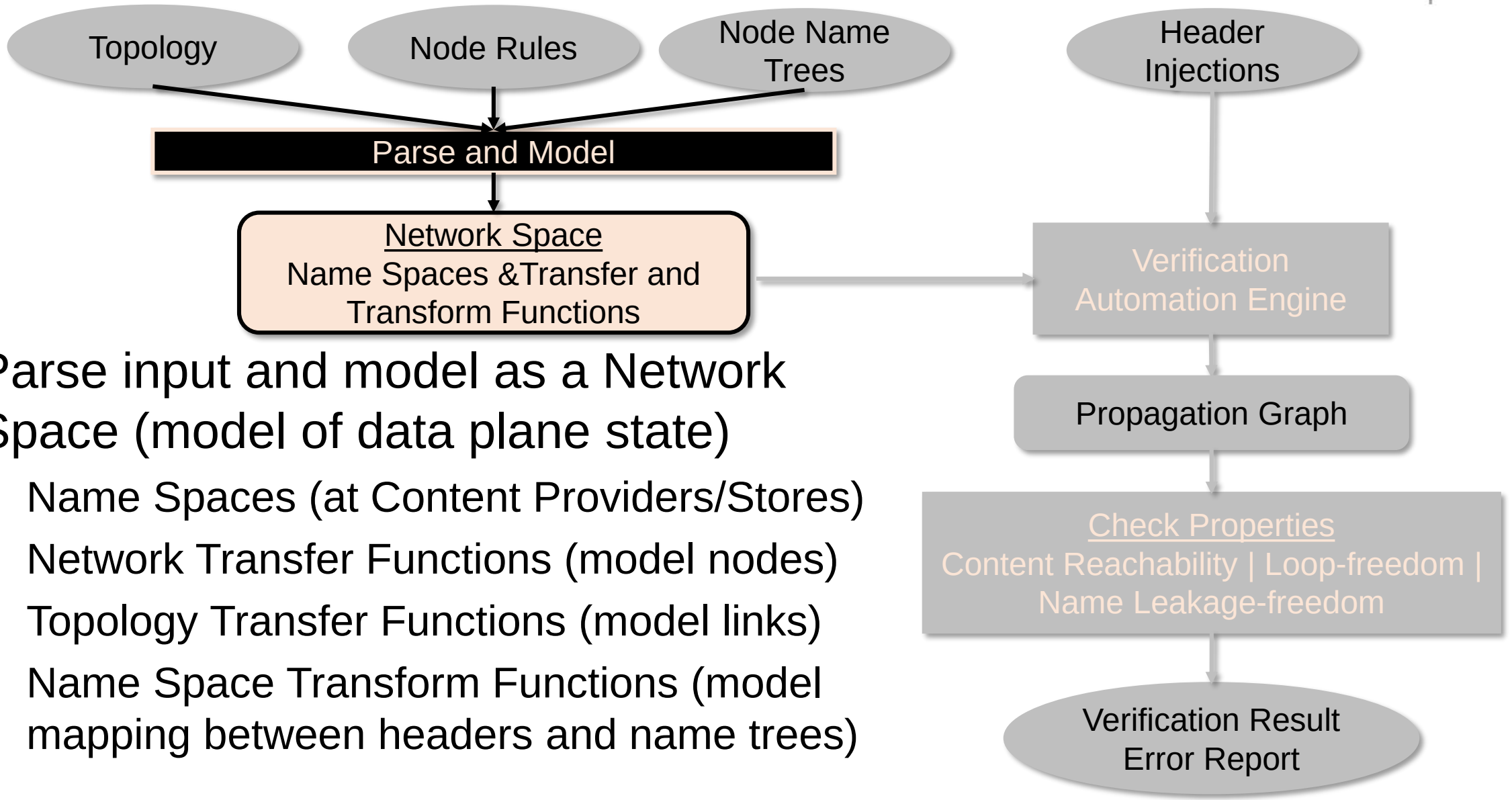


NSA Overview



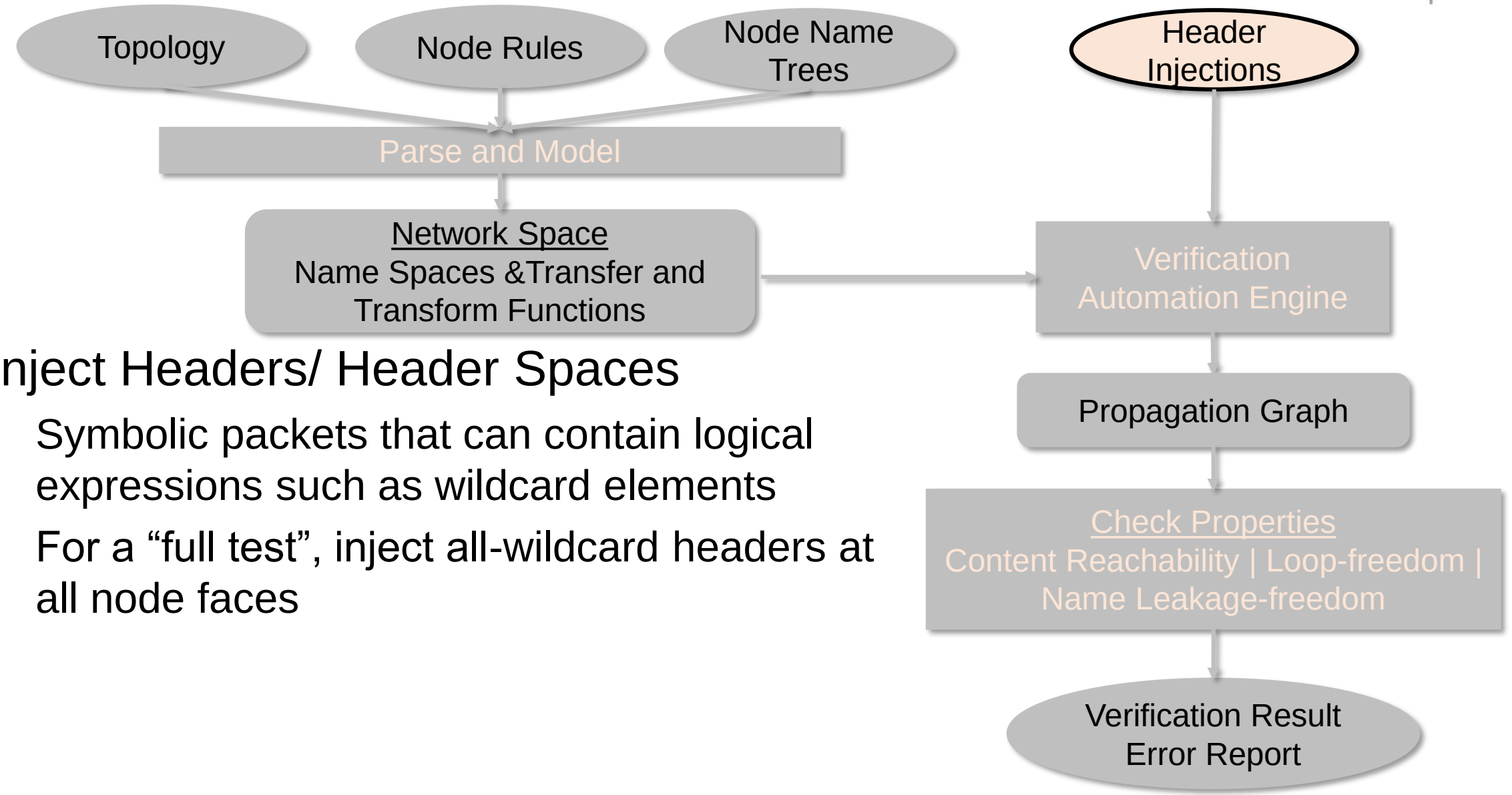
- NDN Data Plane input as
  - Topology (links)
  - Node Rules (e.g., FIB rules, PIT, at routers)
  - Node Name Trees (e.g., content name structures at content providers)
- All the above, combined, define the current “state” of the network

NSA Overview



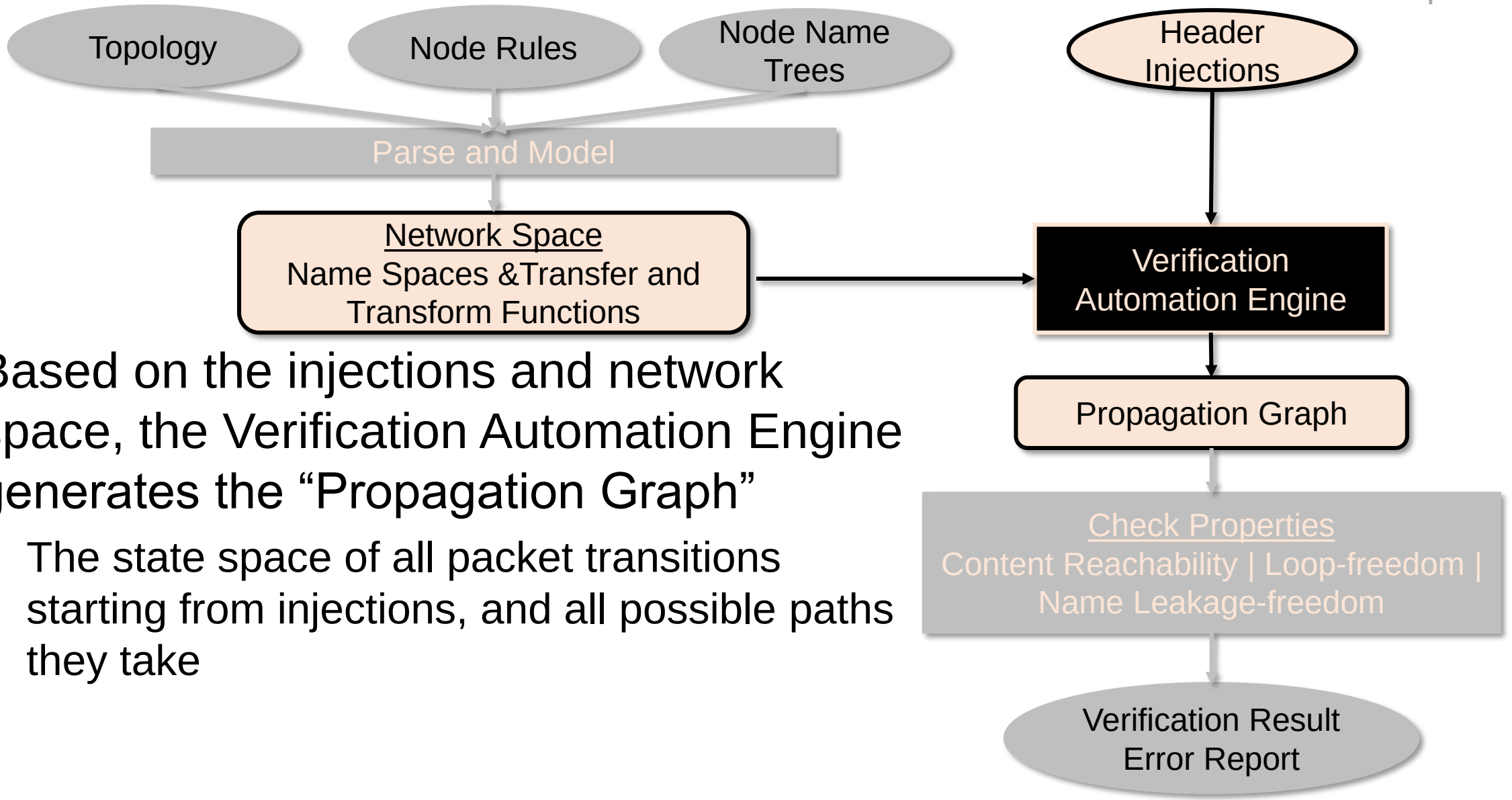
- Parse input and model as a Network Space (model of data plane state)
  - Name Spaces (at Content Providers/Stores)
  - Network Transfer Functions (model nodes)
  - Topology Transfer Functions (model links)
  - Name Space Transform Functions (model mapping between headers and name trees)

NSA Overview



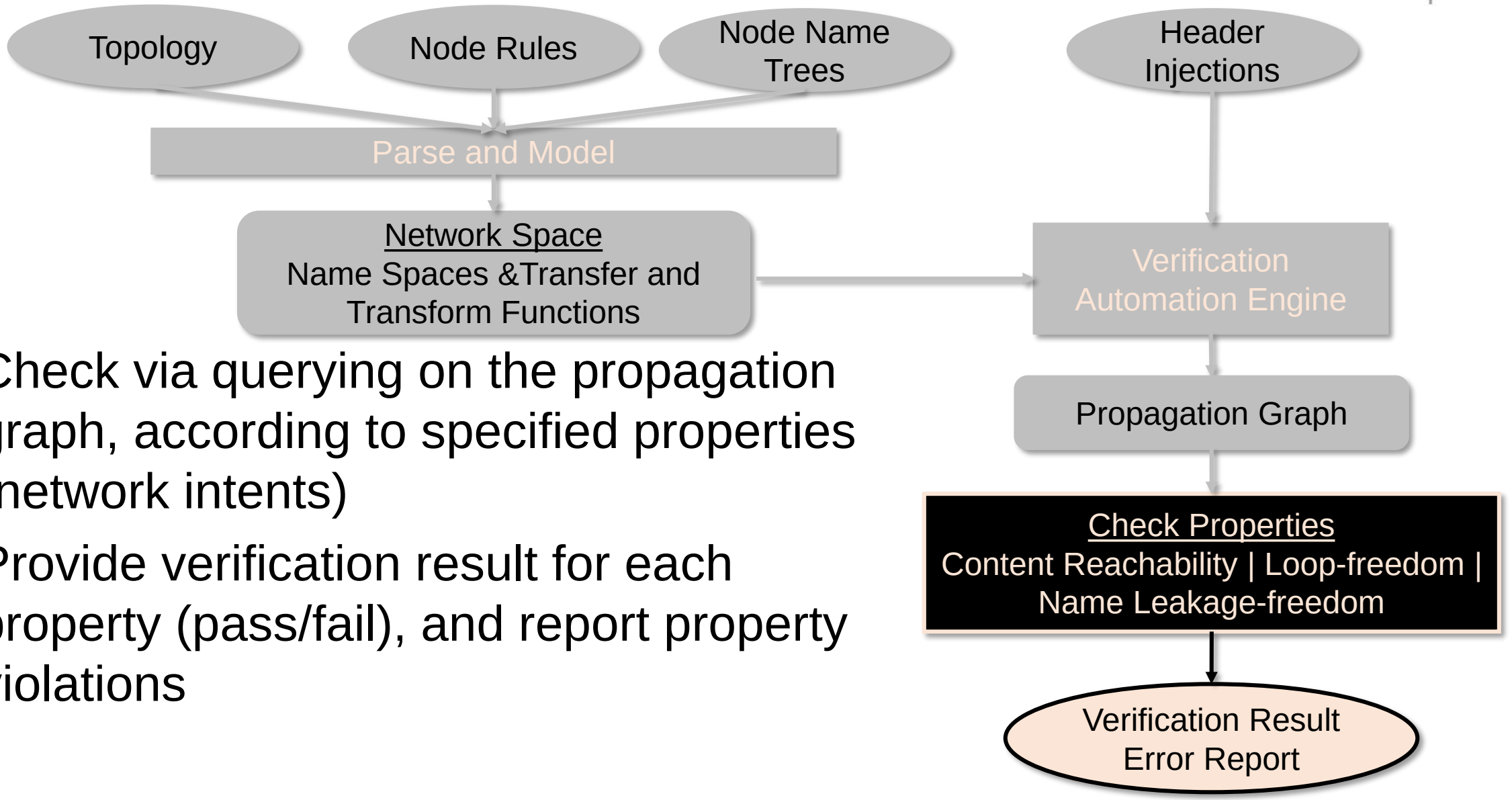
- Inject Headers/ Header Spaces
  - Symbolic packets that can contain logical expressions such as wildcard elements
  - For a “full test”, inject all-wildcard headers at all node faces

NSA Overview



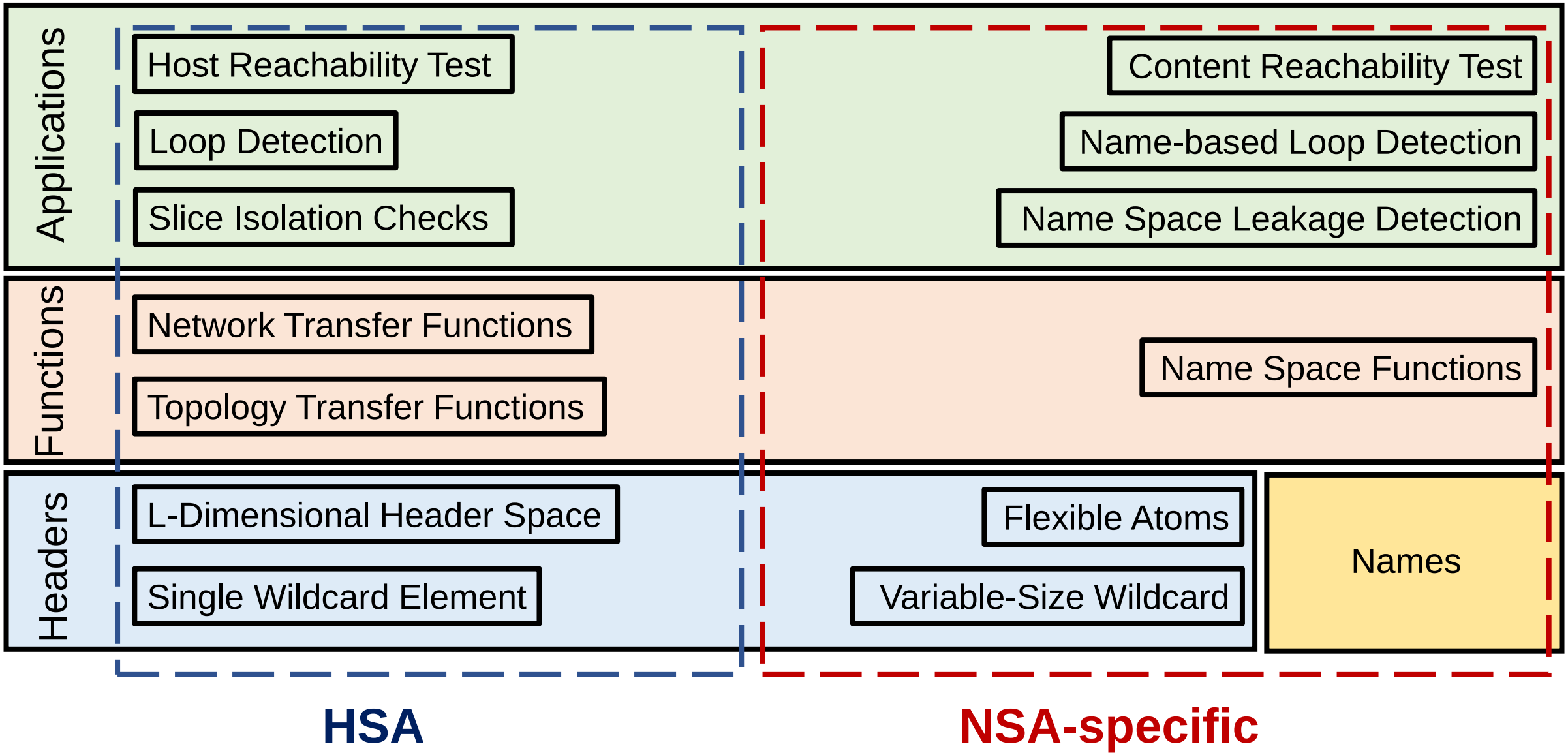
- Based on the injections and network space, the Verification Automation Engine generates the “Propagation Graph”
  - The state space of all packet transitions starting from injections, and all possible paths they take

NSA Overview



- Check via querying on the propagation graph, according to specified properties (network intents)
- Provide verification result for each property (pass/fail), and report property violations

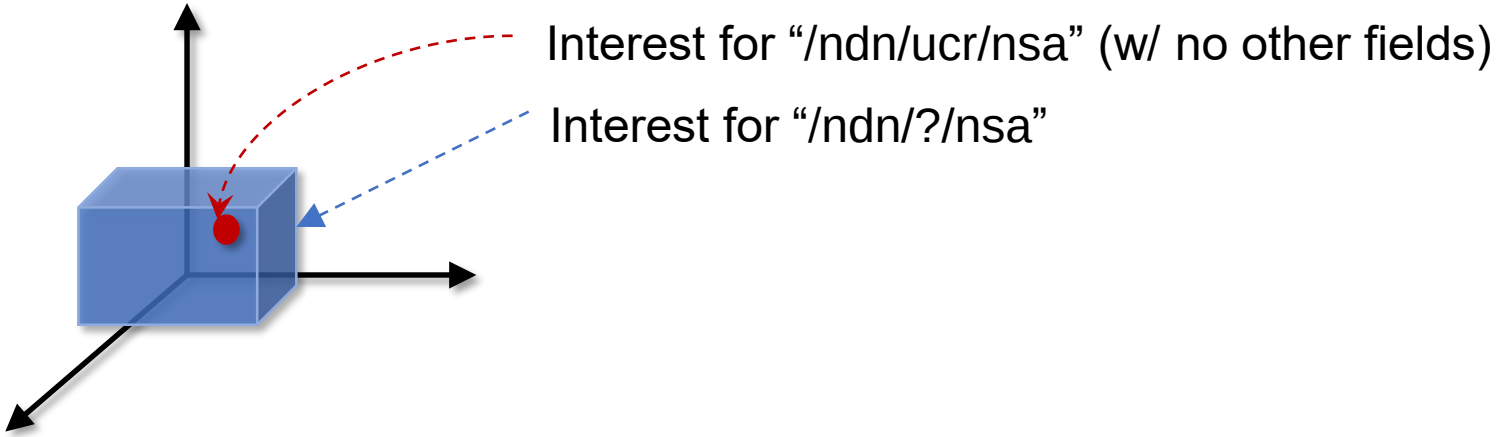
From HSA to NSA: Architecture



NSA Building Blocks/Definitions

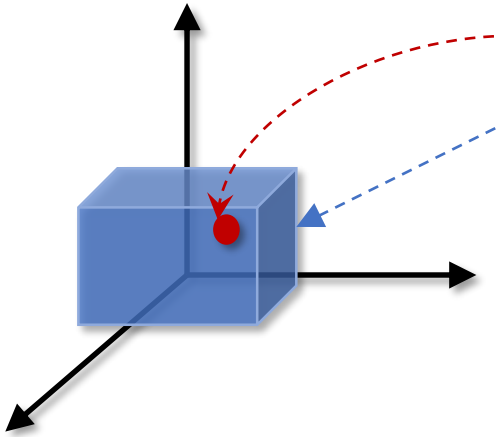
Modeling NDN Header Spaces

- Geometric view of packet headers
  - A header is a point in a multi-dimensional space
  - A header with wildcard elements forms a header space
    - A wildcard element can take all possible values according to an “alphabet”
  - Can manipulate header spaces using a number of defined set operations
- Geometric view: only for purposes of ease of conceptualizing and understanding



Modeling NDN Header Spaces

- NDN packets have a nested TLV format (Unlike IP's fixed structure)
  - Packets for us are basically just all the headers
  - Smallest primitive is byte (octet)
    - NSA atoms can be bytes, field-values, or name components
  - Need single ([?]) and variable-size ([∗]) wildcards in the model
  - Support variable-length headers
    - However, for verification finiteness, set bound L on number of dimensions



Interest for "/ndn/ucr/nsa"

Interest for "/ndn/?/nsa"

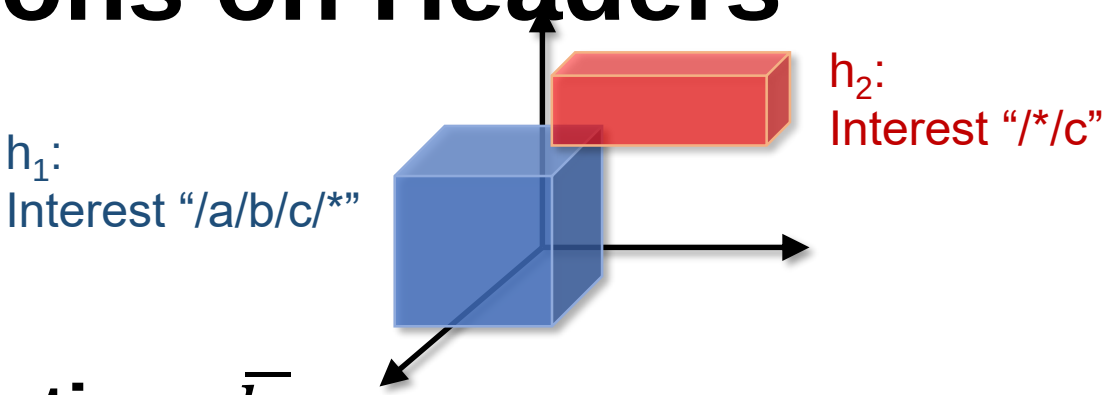
Interest for "/ndn/*/nsa"

Single Wildcard

Variable Wildcard

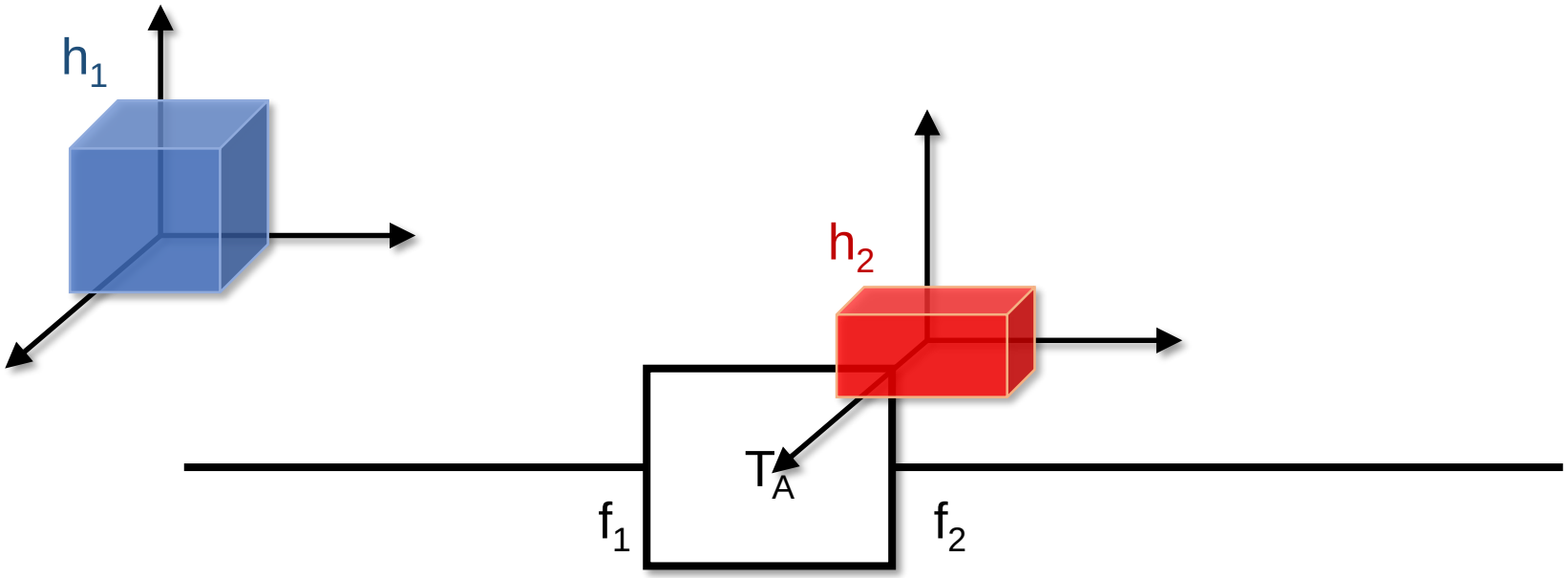
[∗] = ØU[?]U[?][?]U … until limited by L.

Set-Operations on Headers



- **Complementation**: $\bar{h_1}$
  - All values other than h_1 according to atom alphabet
- **Union**: $h_1 \cup h_2$
  - (may or may not be simplifiable)
- **Intersection**: $h_1 \cap h_2$
  - Interest “/a/b/c” (Interest with prefix “/a/b/c” and no suffix)
- **Difference**: $h_1 - h_2 = h_1 \cap \bar{h_2}$
  - Interest “/a/b/c/*/$\bar{c}$” (Interest with prefix “/a/b/c” and not ending with “/c”)

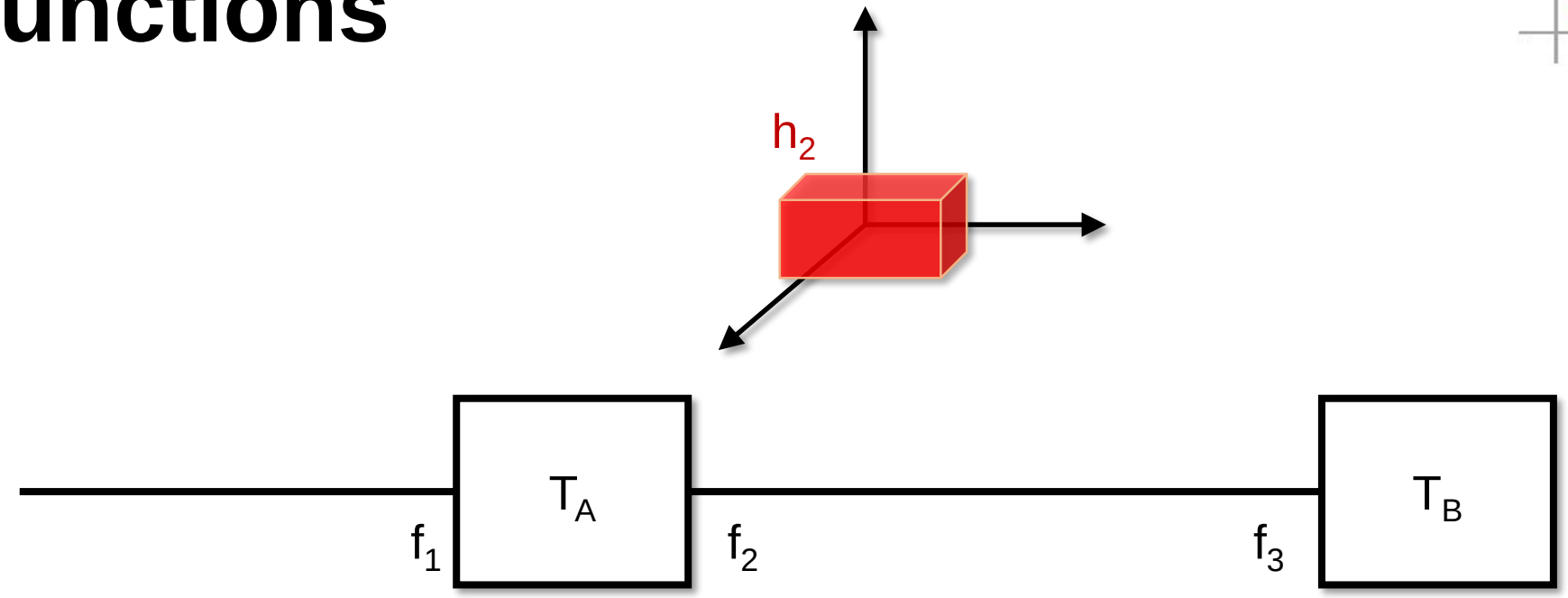
Transfer Functions



- Network Transfer Functions $T_A(h_1, f_1) = (h_2, f_2)$
  - Moves and modifies header from input face to output face of same node
  - Models network nodes (e.g., forwarders)
  - Check conditionals using set-operations (e.g., intersection), and manipulate headers

$$T(h,f) = \begin{cases} (h', f') \; if \; \ldots \\ \ldots \end{cases}$$

Transfer Functions



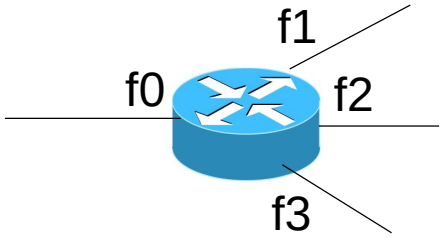
- Topology Transfer Functions
  - Moves header from output face of one node to input face of another node through the connecting link
  - Models link behaviors (connection between two faces)

$$\boldsymbol{\Gamma(h2, f2) = (h2, f3)}$$

$$\Gamma(h, f) = \begin{cases} (h, f') & \text{if } f \text{ connected to } f' \\ \emptyset, & \text{otherwise} \end{cases}$$

Modeling NDN Nodes

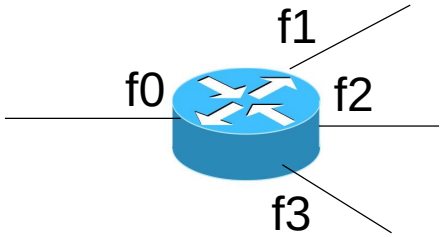
- NDN nodes can be modeled using network transfer functions



- Model NDN Interest and Data processing pipelines in a node, with composition of multiple transfer functions
  - Each pipeline stage as a network transfer function
    - Generally, $T(h_0, f_0) \rightarrow \{(h_1, f_1), (h_2, f_2), \ldots\}$
    - May or may not change input header
    - May or may not depend on the incoming face
    - Support multiple output faces (NDN multicast forwarding strategy)
- Example: Interest pipeline $T_I(h, f) = T_{I.fwd}(T_{I.CS}(T_{I.PIT}(h, f)))$

Modeling NDN Nodes

- NDN nodes can be modeled using network transfer functions



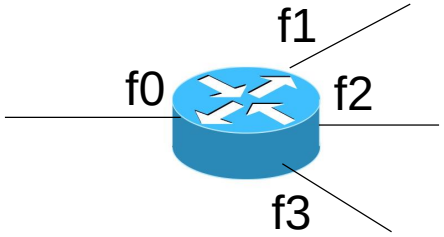
- Example: **Interest forwarding function** $T_{I.fwd}$

$$T_{I.fwd}(h,f) = \begin{cases} \mathrm{U}(h, f_i^{n_1}), & \text{if } FIBM(name(h), n_1), \forall f_i^{n_1} \in SF(n_1) \\ \mathrm{U}(h, f_i^{n_2}), & \text{if } FIBM(name(h), n_2), \forall f_i^{n_2} \in SF(n_2) \\ \emptyset, & \text{otherwise} \end{cases}$$

  - $name(h)$: extract name part of h
  - $FIBM(n, n')$: True if n matches (LPM) index n' of FIB
  - $SF(n)$: set of faces associated with n, according to its strategy

Modeling NDN Nodes

- NDN nodes can be modeled using network transfer functions



FIB	Prefix	Face(s)
	n1	f1, f3
	n2	f2, f3

Fwd. Strategy Table	Prefix	Strategy
	n1	Multicast
	n2	Best-Route

- Example: **Interest forwarding function** $T_{I.fwd}$

Example:
SF(n1) = {f1, f3}
SF(n2) = {f2, ~~f3~~}

- $T_{I.fwd}(h,f) = \begin{cases} \mathrm{U}(h, f_i^{n_1}), & \text{if } FIBM(name(h), n_1), \forall f_i^{n_1} \in SF(n_1) \\ \mathrm{U}(h, f_i^{n_2}), & \text{if } FIBM(name(h), n_2), \forall f_i^{n_2} \in SF(n_2) \\ \emptyset, & \text{otherwise} \end{cases}$
  - $name(h)$: extract name part of h
  - $FIBM(n, n')$: True if n matches (LPM) index n' of FIB
  - $SF(n)$: set of faces associated with n, according to its strategy

Modeling Name Spaces

- We model content name trees (tries) at content providers/stores as a geometric space as well, i.e., Name Spaces
- Interaction of header spaces and name spaces is important – headers carry names, and each operation on a packet depends on the name & the name space.

Modeling Name Spaces

- We model content name trees (tries) at content providers/stores as a geometric space as well, i.e., Name Spaces
- Interaction of header spaces and name spaces are important
- Name Space Function (Ω), to check/compare this interaction
  - Transforms a header space to a name space (header domain to name domain)
    - 1) Extract name components (prefixes) in the headers
    - 2) Construct name tree from prefixes (all possible names)

Header Space

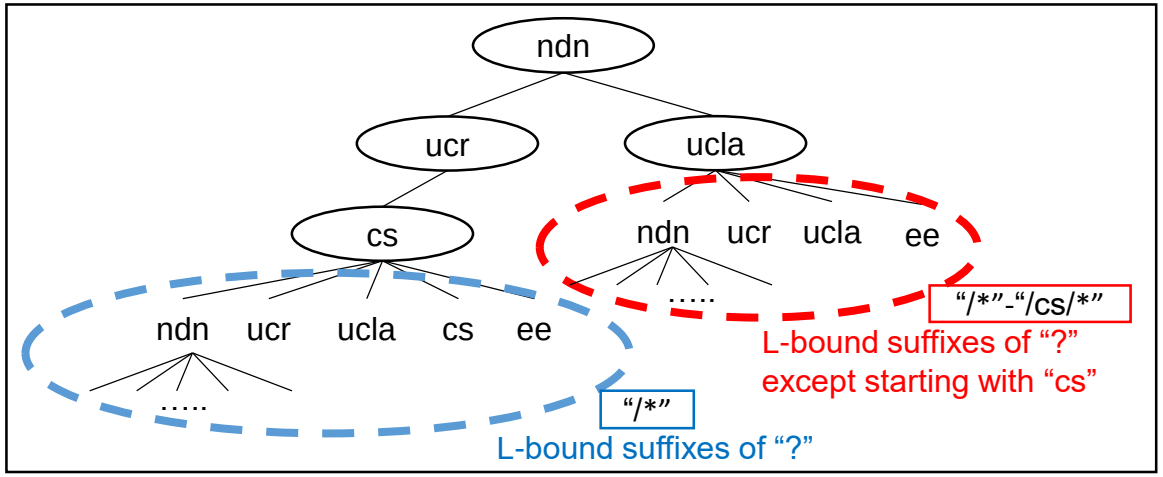
Interest“/ndn/ucr/cs/*” ∪
Interest“/ndn/ucla/$\overline{cs}$/*”

Name Space Function (Ω)

Atom Alphabet (known)

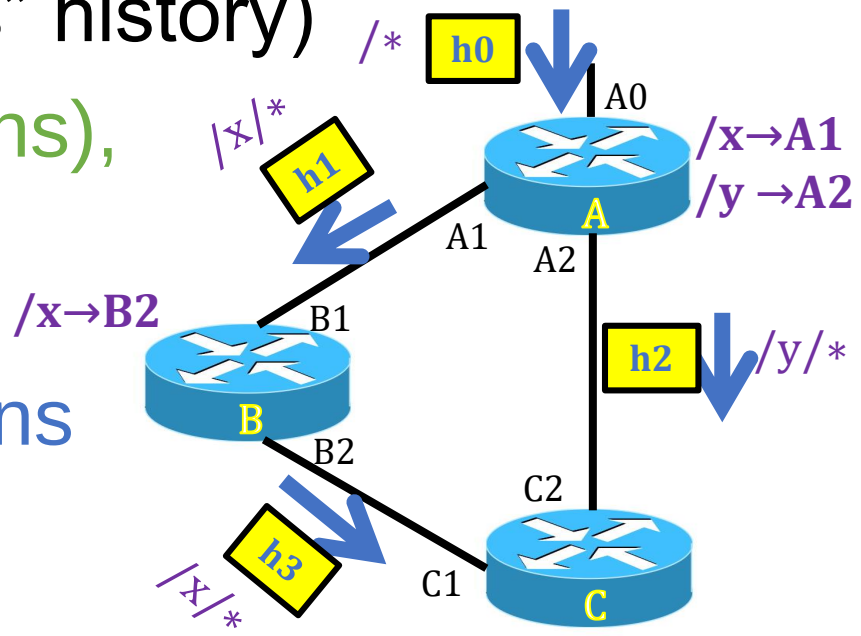
{“ndn”, “ucr”, “ucla”, “cs”, “ee”}

Name Space (generated from the header space)

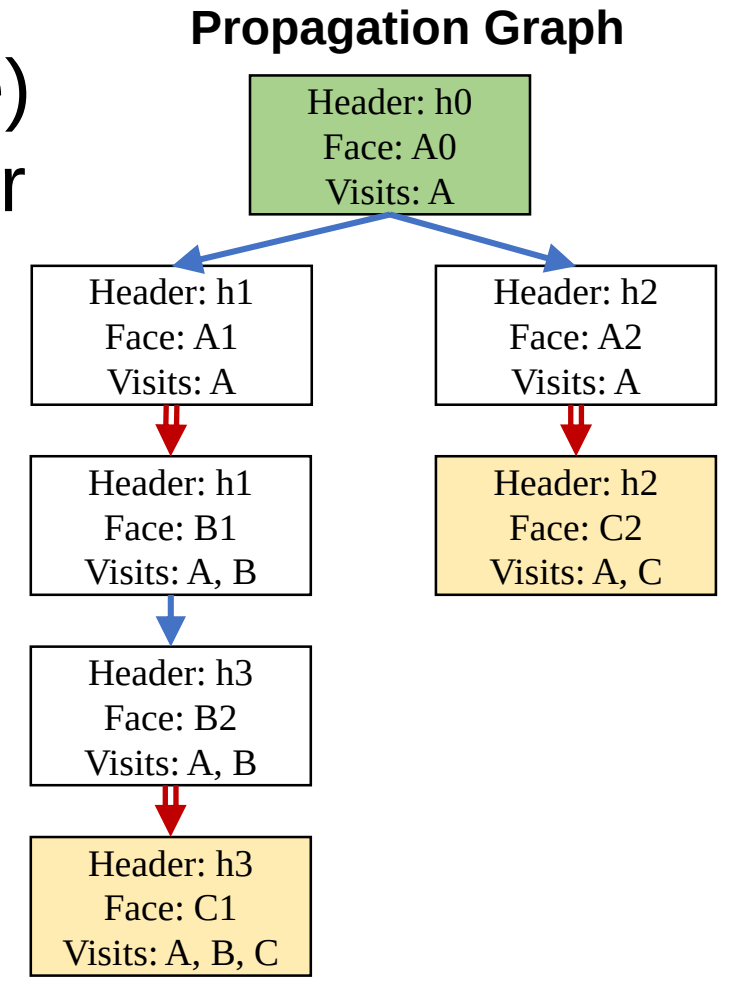


Propagation Graph

- A graph that represent transitions of packets in the network
  - All possible paths of a packet, rather than a single trace
- Each node is a packet state; mainly (header, face) pair (header leaving or arriving at face), plus other possible info' (e.g., "visits" history)
- Initial states (pkt. injections), final states (no more transitions possible), network transfer transitions (→), topology transfer transitions (⟹)



Example: Topology & Injections



NSA Verification Applications

Using NSA for NDN Verification

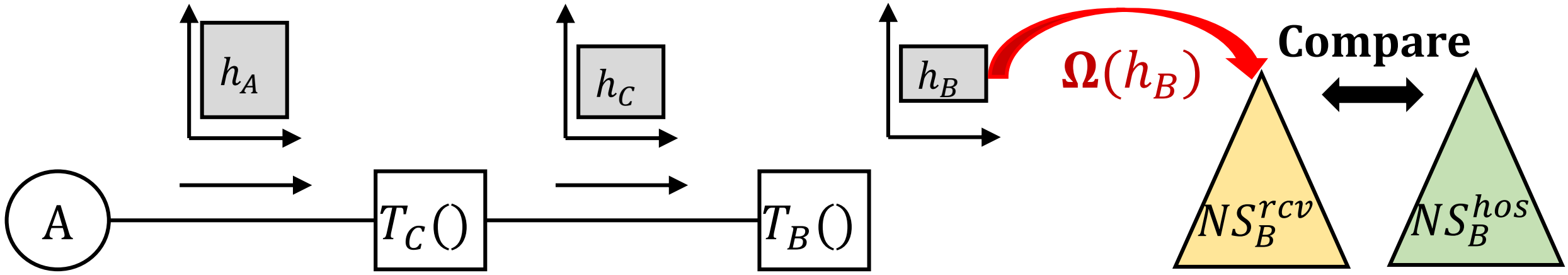
- NSA models a data plane using name spaces, header spaces, and transfer functions

- Verification applications, to check NDN properties
  - Content Reachability: every consumer can reach named content correctly
  - Loop Detection: a named packet should not infinitely loop
  - Name Leakage: a private name should not enter an un-authorized zone

- Enables automated verification
  - Propagation graphs to model the state space

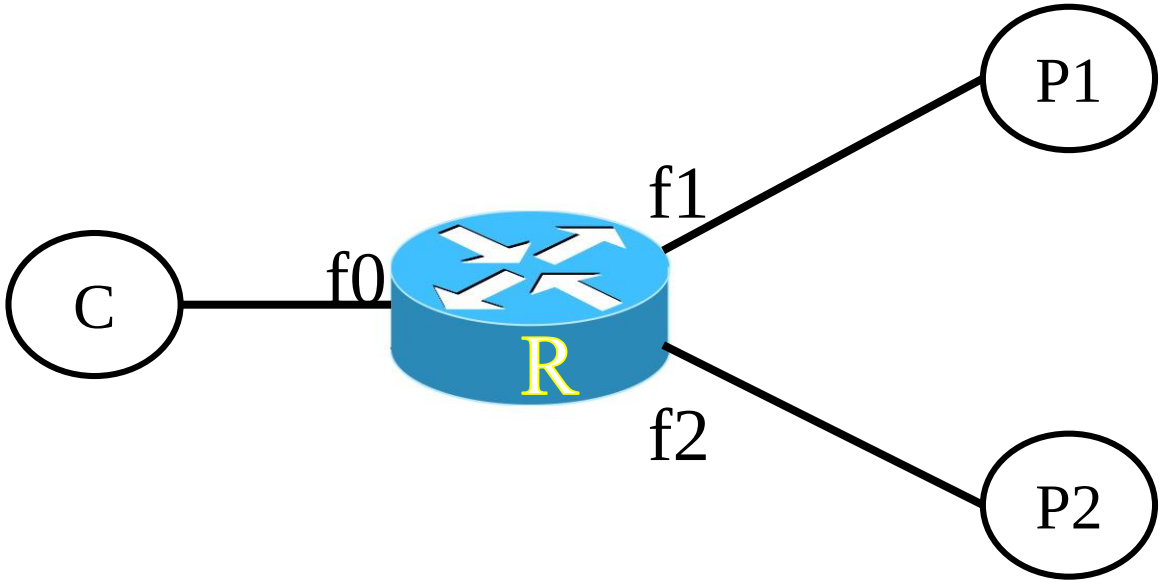
Content Reachability (CR) Analysis

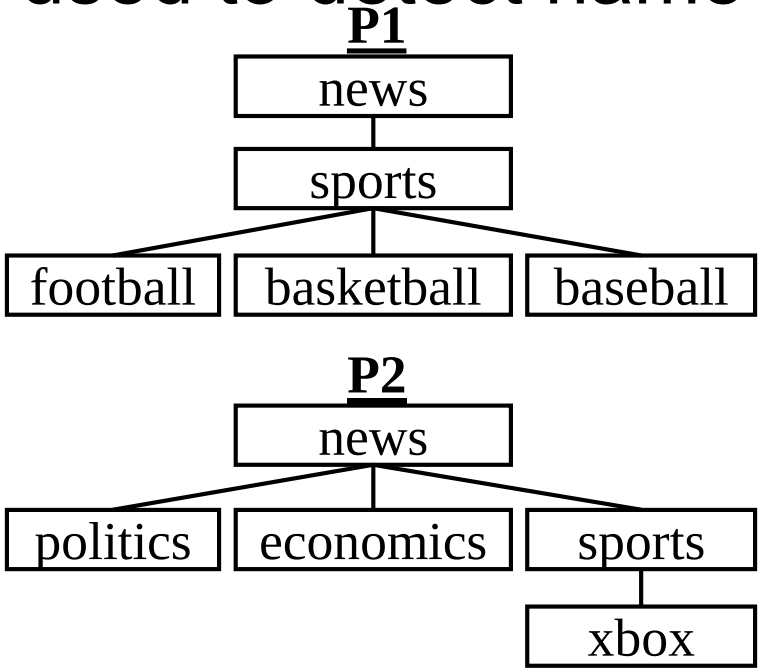
- Reachable content names at repositories (content providers/stores)
  - $CR_{A\to B}(h,f) = \cup_{A\to B\ paths}\{\Omega(T_n(\Gamma(T_{n-1}(\ldots\Gamma(T_1(h,f))))))\}$
    - Range: Name space received at content repository B, when injected h at face f of A
- At B: compare NS_B^{rcv} (generated NS recvd.) with NS_B^{hos} (hosted NS)
  - Ideally, we want $NS_B^{rcv} = NS_B^{hos}$
  - If $NS_B^{rcv} - NS_B^{hos} \neq \emptyset$, then requests received for non-existing names (unsolicited names)
  - If $NS_B^{hos} - NS_B^{rcv} \neq \emptyset$, then part of NS not accessible (unreachable names)



Use Case of Content Reachability Application: Name Space Conflict Detection

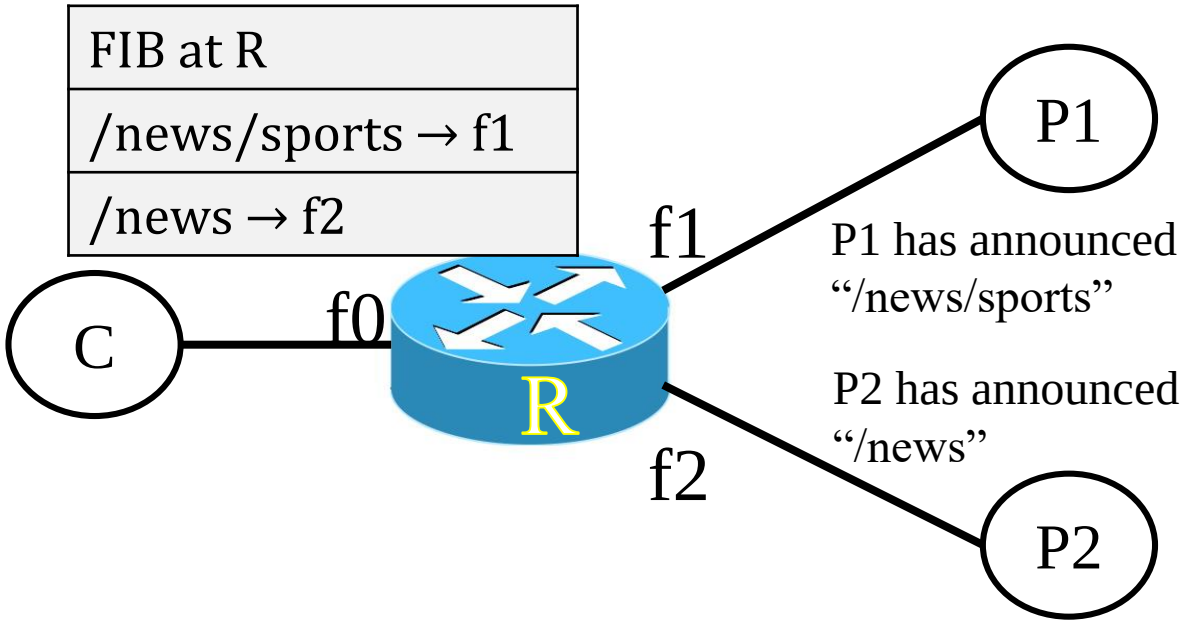
- NSA's content reachability test can be used to detect name space conflicts in the data plane

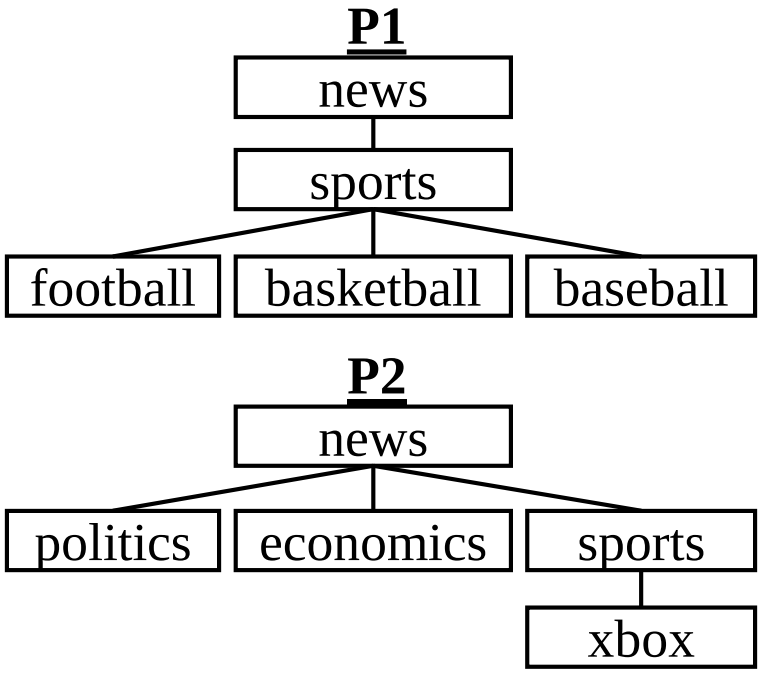




Use Case: Name Space Conflict Detection

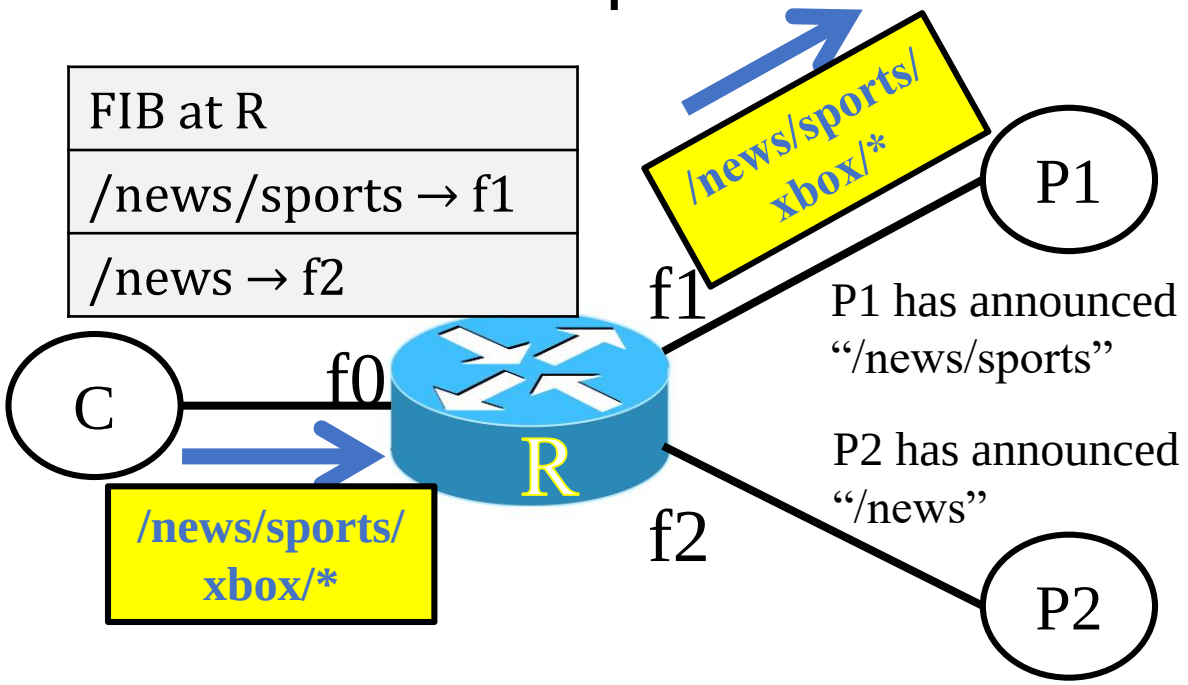
- NSA's content reachability test can be used to detect name space conflicts in the data plane

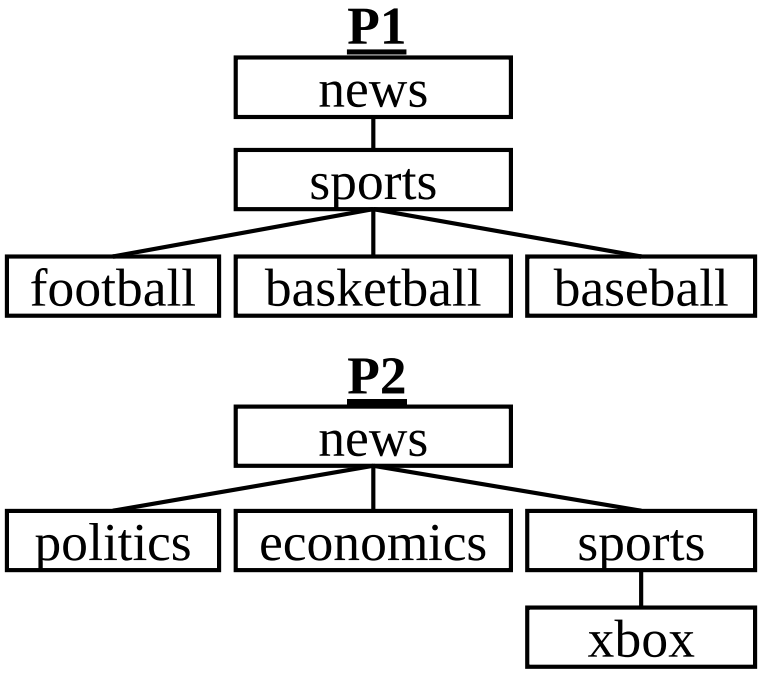




Use Case: Name Space Conflict Detection

- NSA's content reachability test can be used to detect name space conflicts in the data plane

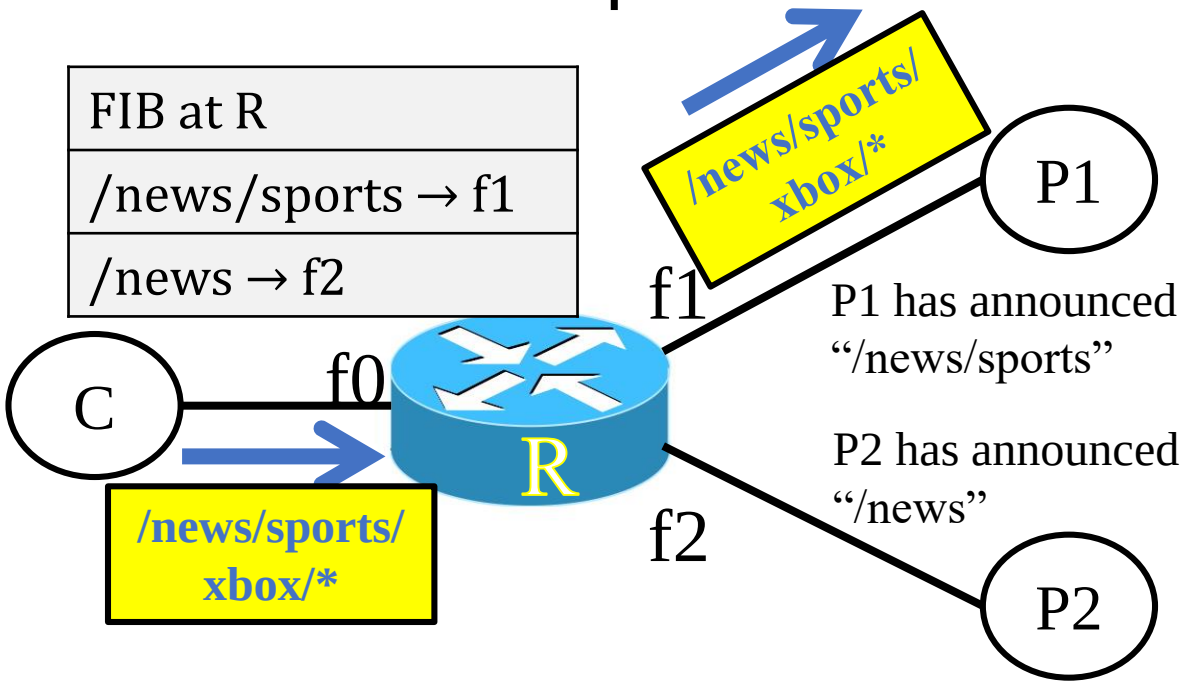


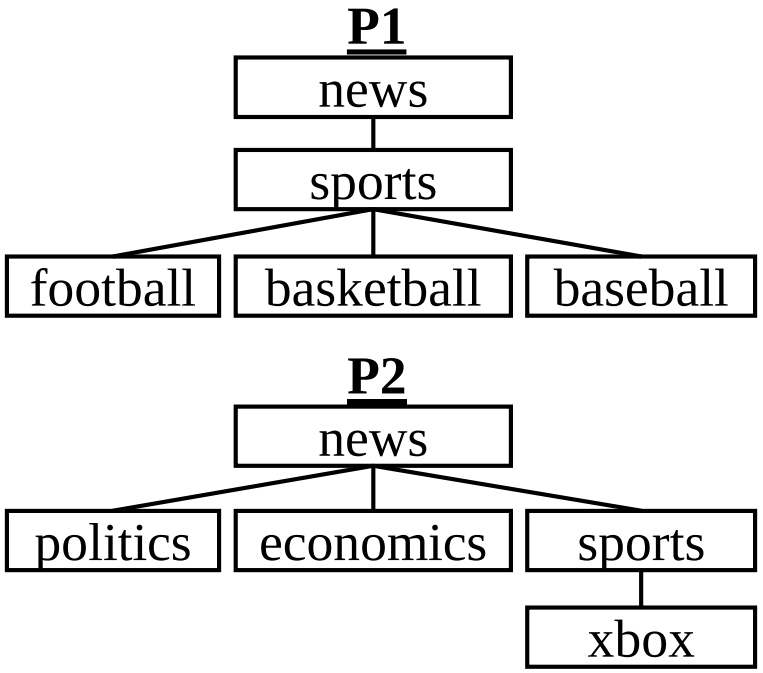


P1 would receive interest for "/news/sports/xbox" instead of P2!

Use Case: Name Space Conflict Detection

- NSA's content reachability test can be used to detect name space conflicts in the data plane





P1 would receive interest for "/news/sports/xbox" instead of P2!

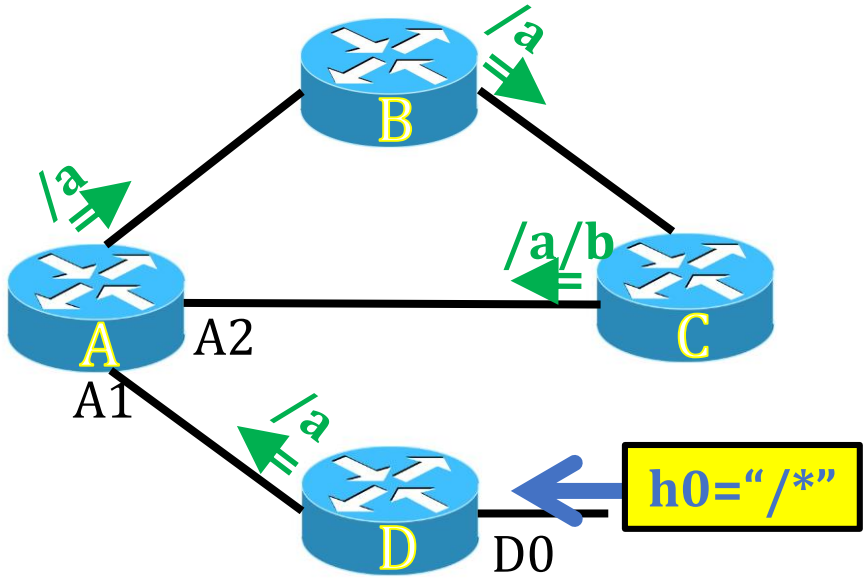
- A protocol for name registration can incorporate this check

Loop Detection

- Looped Interest - problem in NDN; Dead Nonce List is used for this
  - However, such reactive loop detection is not enough:
    - Loop has already occurred (waste of resources, etc.)
    - Solves the router's problem by discarding, but not consumer's problem
      - Often, looping Interest means an Interest is not satisfied
  - Need a method to check "all possible loops"
    - NSA provides that

Loop Detection

- Detect Loops: Inject all-wildcard (“/*”) headers and check if a node visited more than once in one path, i.e., as h and h'



/prefix FIB rule for “/prefix” and its output face direction

Header: h0 = “/*”
Face: D0
Visits: D

…

Header: h=“/a/*”
Face: A1
Visits: D, A

…

Header: h’=“/a/b/*”
Face: A2
Visits: D, A, B, C, A

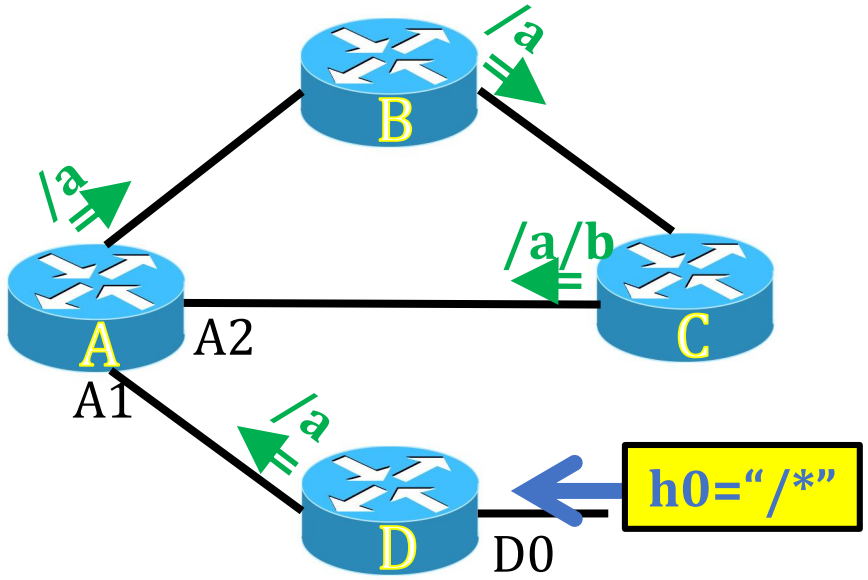
Loop detected!

Loop Detection

- Detect Loops: Inject all-wildcard (“/*”) headers and check if a node visited more than once in one path, i.e., as h and h'
- Detect Infinite Loops: If for the two headers, we have $h' \subseteq h$, then loop is infinite

@ A:
h: Interest “/a/*”
h’: Interest “/a/b/*”

/prefix ⇒ FIB rule for “/prefix” and its output face direction



Header: h0 = “/*”
Face: D0
Visits: D

…

Header: h=“/a/*”
Face: A1
Visits: D, A

…

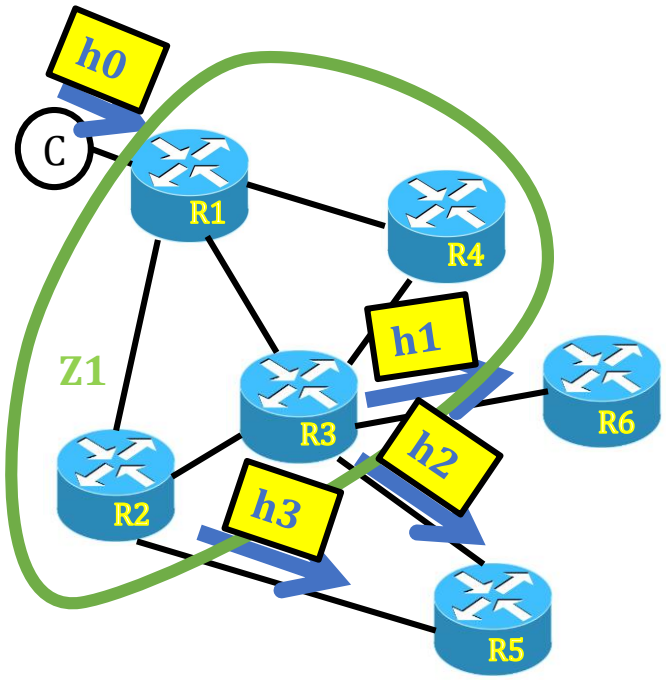
Header: h’=“/a/b/*”
Face: A2
Visits: D, A, B, C, A

Loop detected!

$h' \subseteq h \Rightarrow$ Infinite loop!

Name Leakage Detection

- Check if a name, name component, or name space does not leak out of an authorized zone (e.g., a client ID)
- Zone Z1 is the authorized zone (e.g., a VPN)
- Check union-ed header spaces leaving Z1
  - $H_{out} = h1 \cup h2 \cup h3$
- Check H_{out} against the prohibited header space
  - Must have $H_{out} \cap H_{prohibited} = \emptyset$
- Also can prohibit an entire namespace:
  - $\Omega(H_{out}) \cap NS_{prohibited} = \emptyset$
- Incorrect data planes can be remedied using correct configs or addition of ACL rules for VPNs



Other Use Cases

- Variations of content reachability analysis
  - Correctness of outcome of route computation
    - Check if a particular content request reaches the nearest (or all/any) content
  - Security infrastructure soundness
    - Check if all public keys (data with “/KEY” prefix) can be reached appropriately
  - Content censorship-freedom
    - Check if all content names are reachable despite possible censoring nodes
  - Content neutrality
    - Check if two replicated content providers’ name spaces are reached equally
  - Check equivalence between multiple data plane states
    - E.g., content reachability before and after a content provider’s mobility

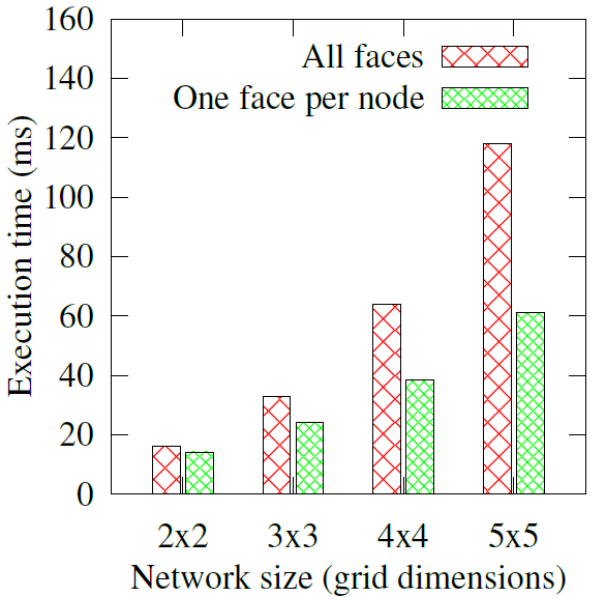
Experiments with NSA: Characterizing Performance & Scalability

NSA Implementation

- We have implemented NSA and its essential components
  - Parsers, building blocks, propagation graph generation, verification applications
  - Java implementation: https://github.com/mjaha/NameSpaceAnalysis
- Injecting all-wildcard headers at all faces (by default), NSA provides automated and thorough verification of a data plane
- Added a number of optimizations to enhance NSA's performance
  - Limiting injection per node to one face $\Longrightarrow$ smaller verification time
  - Aggregating "similar" headers in the propagation graph $\Longrightarrow$ smaller verification time

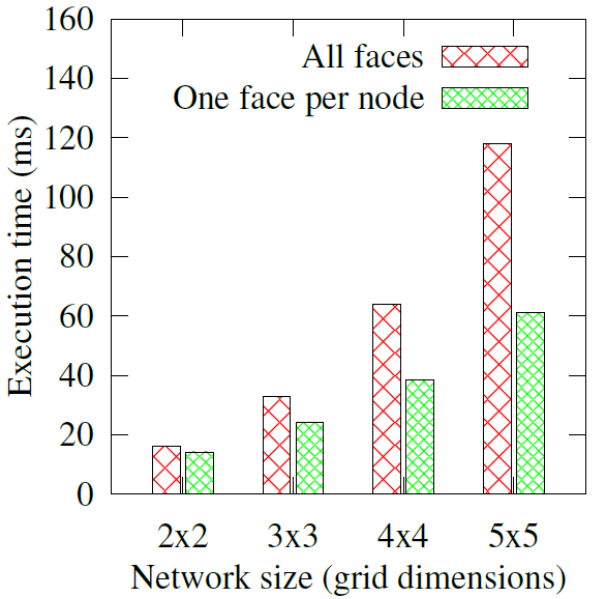
Performance

- Experiments on synthetic snapshots ($n \times n$ grids)
  - Reasonable growth rate (linear) with network size
  - Injection face limitation decreases verification time
  - More experiments on other applications, topologies and optimizations in the paper!



Performance

- Experiments on synthetic snapshots ($n \times n$ grids)
  - Reasonable growth rate (linear) with network size
  - Injection face limitation decreases verification time
  - More experiments on other applications, topologies and optimizations in the paper!



- Experiments on NDN Testbed data plane (http://ndndemo.arl.wustl.edu/)
- Found 450 content reachability and 704 loop-freedom errors
- (few secs. to verify)

Application	Best-Route	Multicast
Content Reachability Analysis	196	2,481
Content Reachability Analysis (w/aggregation)	75	342
Loop Detection	190	2,416

Summary

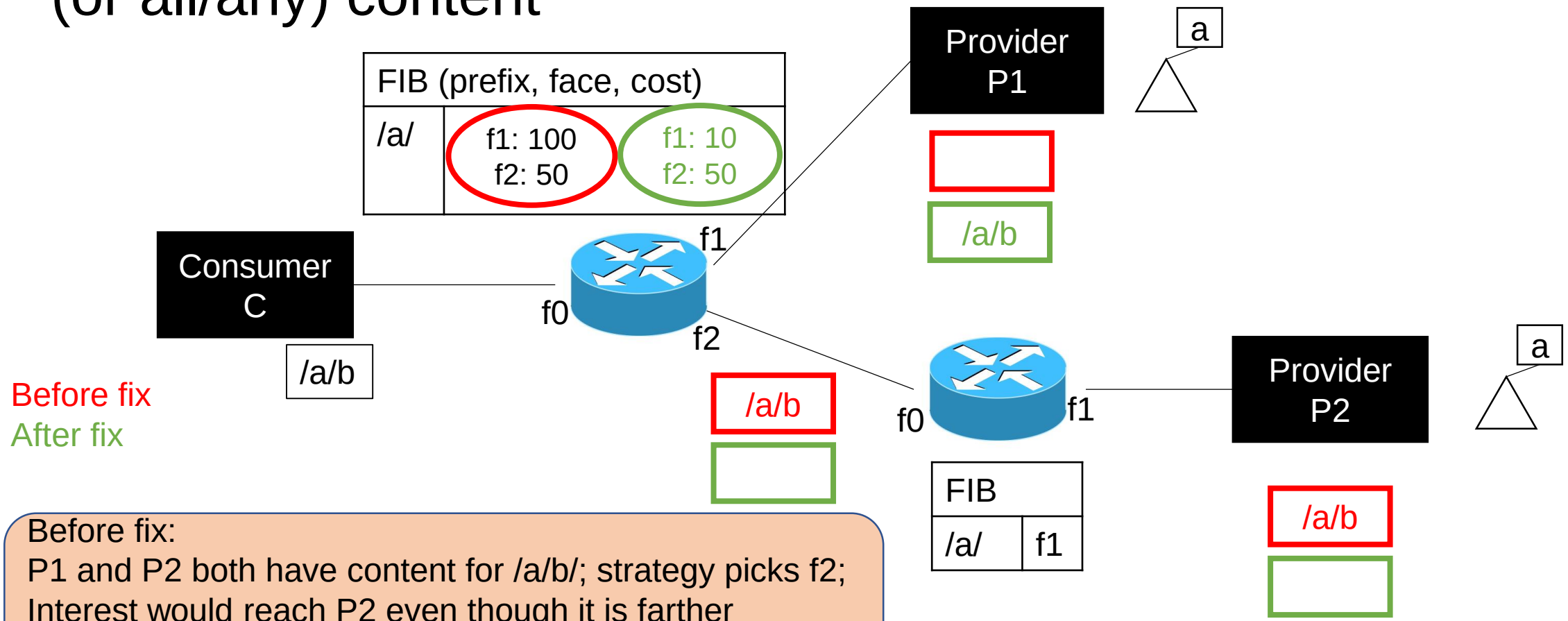
- Name Space Analysis (NSA): a formal method and tool to verify NDN data planes
- Models name spaces, name-based transfer functions and headers
- Verifies NDN intents: content reachability, loop-freedom, name leakage-freedom
- Effective in finding data plane errors and is efficient
- Available at https://github.com/mjaha/NameSpaceAnalysis

Extended Material

More use cases of NSA

Route computation correctness

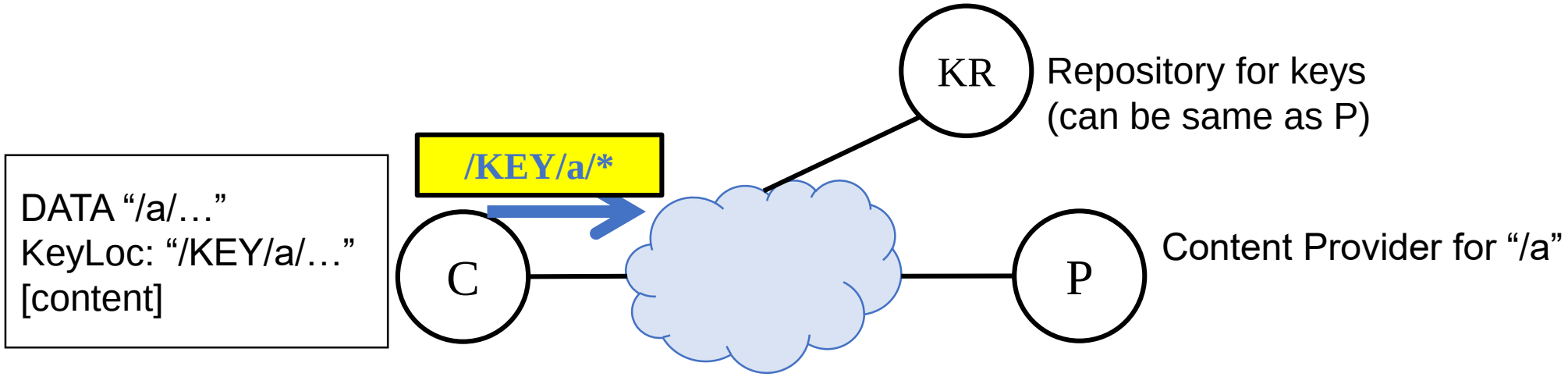
- NSA can check if a particular content request reaches the nearest (or all/any) content



Before fix:
P1 and P2 both have content for /a/b/; strategy picks f2;
Interest would reach P2 even though it is farther
(incorrect route costs)
NSA detects this

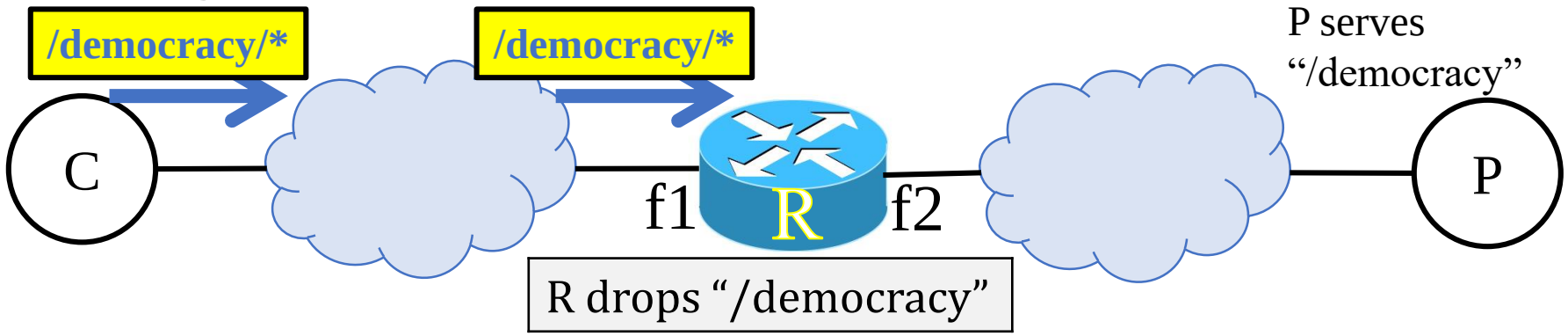
Key infrastructure soundness check

- Key name is appended to Data in Key Locator field
- For data-oriented authentication, the key has to be retrieved using Interest
  - Otherwise, data cannot be authenticated which has security issues
- NSA can check if the key can be retrieved from any consumer



Content Censorship and neutrality

- Content censorship-freedom: Check if all content names are reachable despite possible censoring nodes

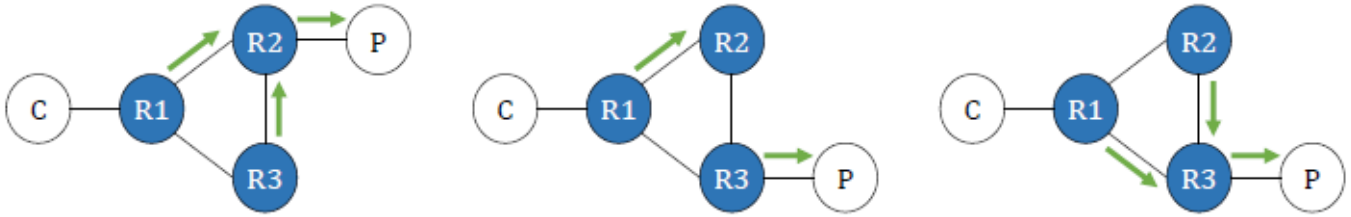


P would not receive interests for "/democracy" which it serves, since R has censored it!

- Content neutrality: Check if two replicated content providers' name spaces are reached equally (receive "same" interests)
- While NSA cannot find the root of such errors, it can be used to check if a data plane is "free" from censorship and neutrality violation errors

Multi-snapshot checks

- NSA can check equivalence between multiple data plane states
- Example: content reachability before and after a content provider's mobility



(a) Initial state (s1) (b) Intermediate state (s1.5) (c) Final state (s2)

- Provider P moves, and the FIBs get re-populated
- NSA can check states s1 and s2, and compare "Received Name Spaces" at P, at s1 and s2; i.e., $NS^{rcv}_{P,s1}$ and $NS^{rcv}_{P,s2}$
  - Ideally, we want the two name spaces to be equal
    - To prove that the mobility handling preserves consistency