

PGA: Using Graphs to Express and Automatically Reconcile Network Policies

Chaithan Prakash^{^*} Jeongkeun Lee[†] Yoshio Turner^{○*} Joon-Myung Kang[†] Aditya Akella[^]

Sujata Banerjee[†] Charles Clark[‡] Yadi Ma[†] Puneet Sharma[†] Ying Zhang[†]

[^]University of Wisconsin-Madison, [†]HP Labs, [○]Banyan, [‡]HP Networking

- Background
- Graph Model
- Graph Composition
- Case Study
- Prototype and Evaluation
- Conclusion

Background

1 @10.0.0.0/8 0.0.0.0/0 0 : 65535 0 : 65535 0x00/0x00
2 @172.16.0.0/12 0.0.0.0/0 0 : 65535 0 : 65535 0x00/0x00
3 @192.168.0.0/16 0.0.0.0/0 0 : 65535 0 : 65535 0x00/0x00
4 @0.0.0.0/0 0.0.0.0/0 0 : 65535 13 : 13 0x11/0xff
5 @0.0.0.0/0 0.0.0.0/0 0 : 65535 37 : 37 0x06/0xff
6 @0.0.0.0/0 0.0.0.0/0 0 : 65535 37 : 37 0x11/0xff
7 @0.0.0.0/0 0.0.0.0/0 0 : 65535 67 : 67 0x11/0xff
8 @0.0.0.0/0 0.0.0.0/0 0 : 65535 113 : 113 0x06/0xff
9 @0.0.0.0/0 0.0.0.0/0 0 : 65535 138 : 138 0x11/0xff
10 @0.0.0.0/0 0.0.0.0/0 0 : 65535 53 : 53 0x11/0xff
11 @0.0.0.0/0 0.0.0.0/0 53 : 53 0 : 65535 0x11/0xff
12 @0.0.0.0/0 0.0.0.0/0 0 : 65535 0 : 65535 0x01/0xff
13 @0.0.0.0/0 0.0.0.0/0 1024 : 65535 33434 : 33600 0x11/0xff
14 @0.0.0.0/0 0.0.0.0/0 33434 : 33600 1024 : 65535 0x11/0xff
15 @0.0.0.0/0 204.152.184.87/32 0 : 65535 22 : 22 0x06/0xff
16 @0.0.0.0/0 204.152.184.88/32 0 : 65535 22 : 22 0x06/0xff

Network Programming Language



- Recent works
 - Tie policy expression to low-level specifics (IP/MAC addresses).
 - Complex with steep learning curve.

Policy Graph Abstraction (PGA)

- Focus on *end-to-end policy* and treat network as *one big switch*.
- Abstraction: intuitive high-level graph model without IP/subnet assignment.
- Composition: independent policy graphs to one coherent composed policy.

Target Scenarios

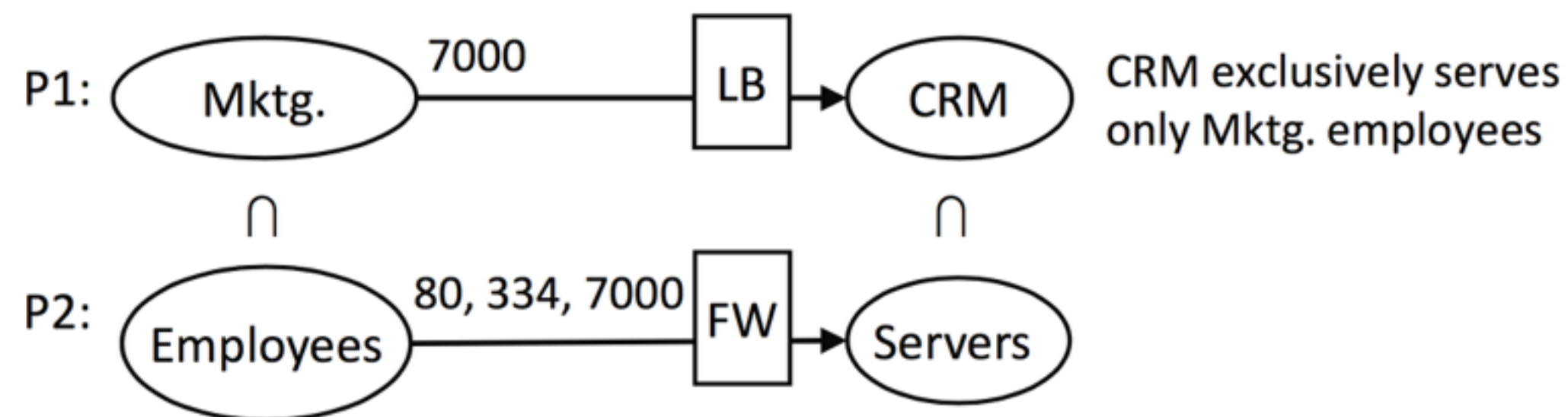
- Enterprise networks
 - administrative domains, applications, protocol, technology.
- Cloud infrastructures
 - multi-tenants, cloud network operator.
- NFV service networks
 - multiple service function chains.

Challenges in Policy Composition

- P1: only marketing employees to send traffic to the CRM servers; the traffic must use TCP port 7000 and must pass through a load balancing service (LB).
- P2: company employees to access company servers only through TCP ports 80, 334, and 7000; and the traffic must pass through a firewall service (FW).
- Naively prioritise P1 over P2 incorrectly allowing non-Marketing traffic to reach CRM servers.

Challenges in Policy Composition

- Manual decomposition and recomposition of ACL and service requirements.
- Impractical and error-prone for thousands of real world policies.



(a) Independently specified policies.

```
if_(match(srcip=Mktg, tcp, dstport=7000, dstip=CRM),
    FW>>LB>>route,
    if_(match(dstip=CRM), drop,
        if_(match(srcip=Empl, tcp, dstport=80|334|7000, dstip=Servers),
            FW>>route, drop)))
```

(b) Pyretic-style composite program.

Subject1: CRM-Access
 tcp, dstport =7000 : Permit, FW-LB chain

Subject2: CRM-Block
 * : Deny

Subject3: Server-Access
 tcp, dstport=80|334|7000 : Permit, FW chain

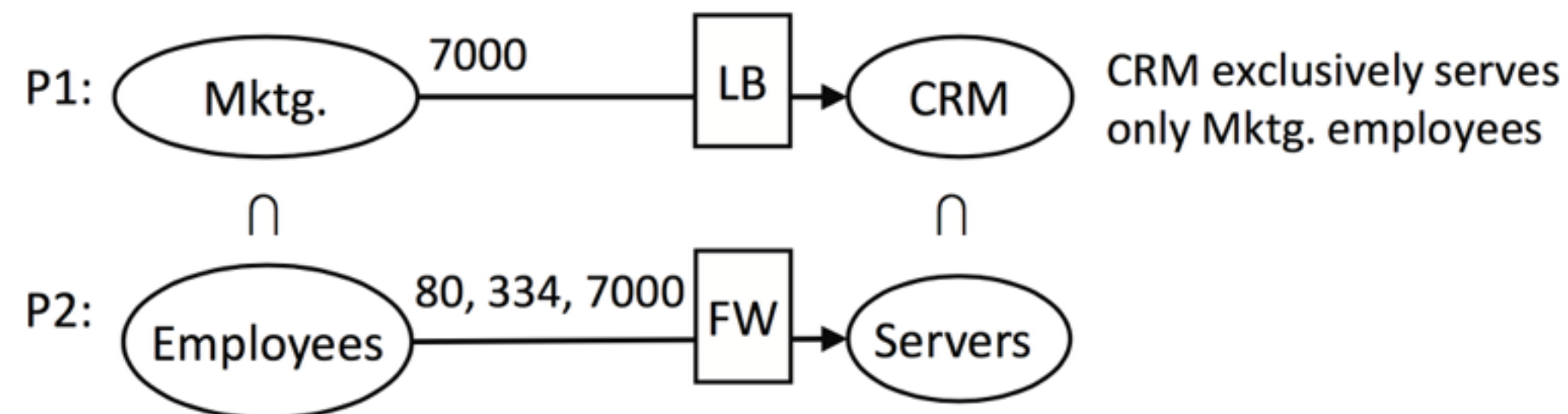
Clauses (Prioritized):

1. Mktg --> CRM : CRM-Access
2. * --> CRM : CRM-Block
3. Employees --> Server : Server-Access

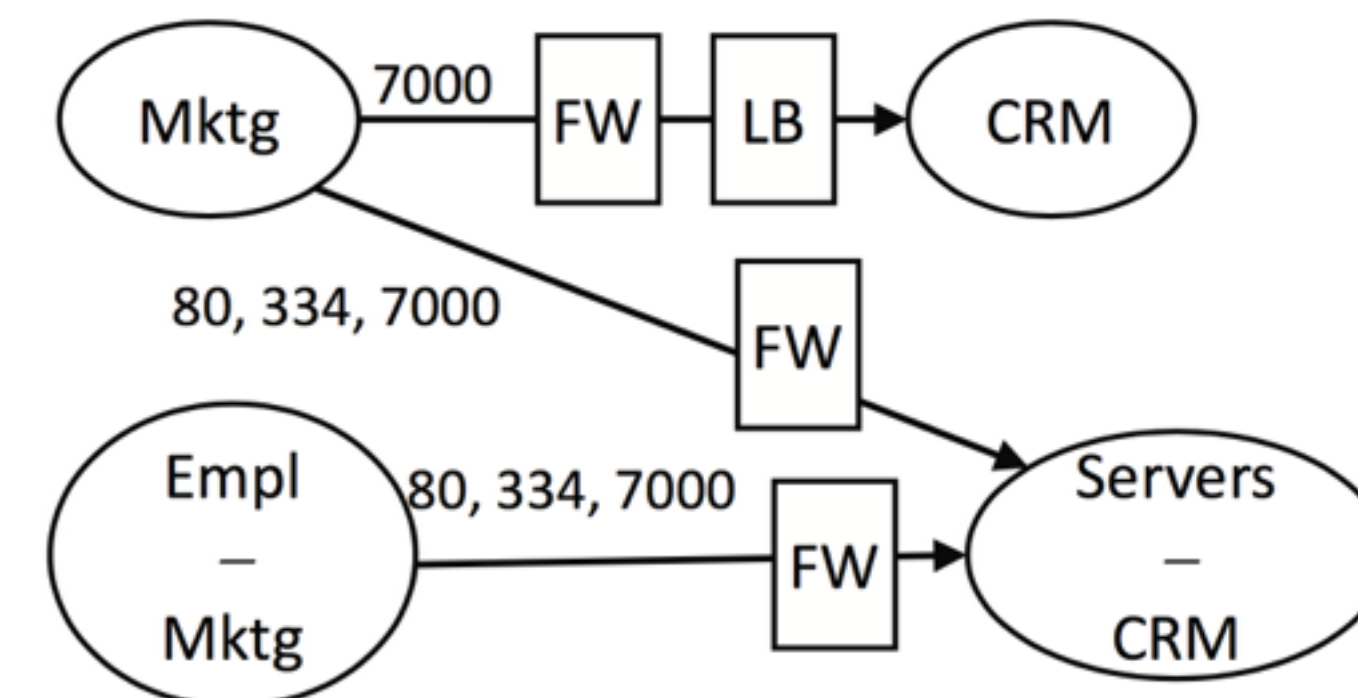
(c) composite policy specified in GBP.

Challenges in Policy Composition

- A correct composition of P1 and P2 in a simple graph.
- An entire policy expressed on the edge.



(a) Independently specified policies.



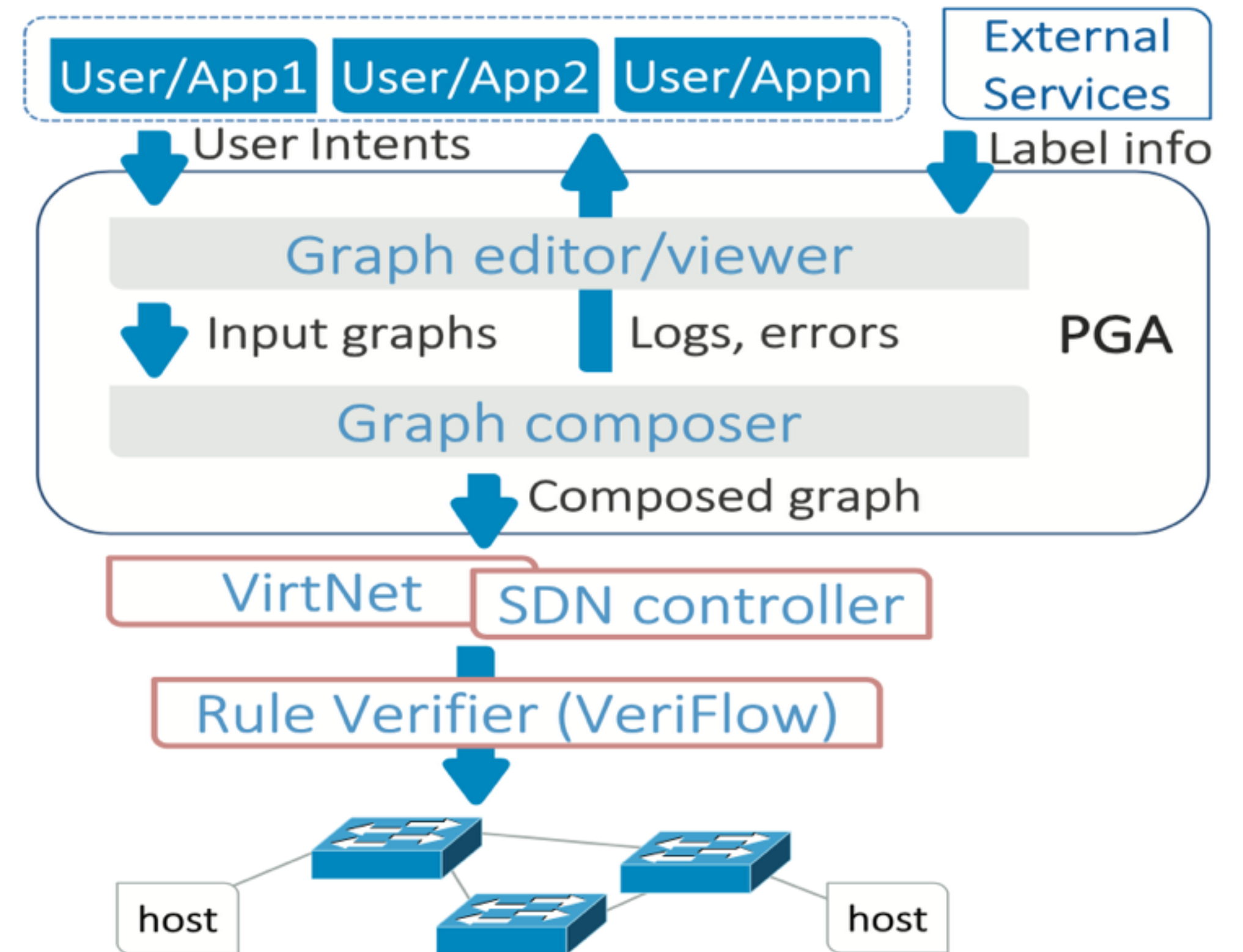
(d) Composed policy in a graph.

Requirements for Policy Framework

- Simple and intuitive
- Independent and composable
- Eager composition (proactive)
- Automated (much less burdensome for human)
- Well-formed (deterministic)
- Service chain analysis (proper order)

System Overview

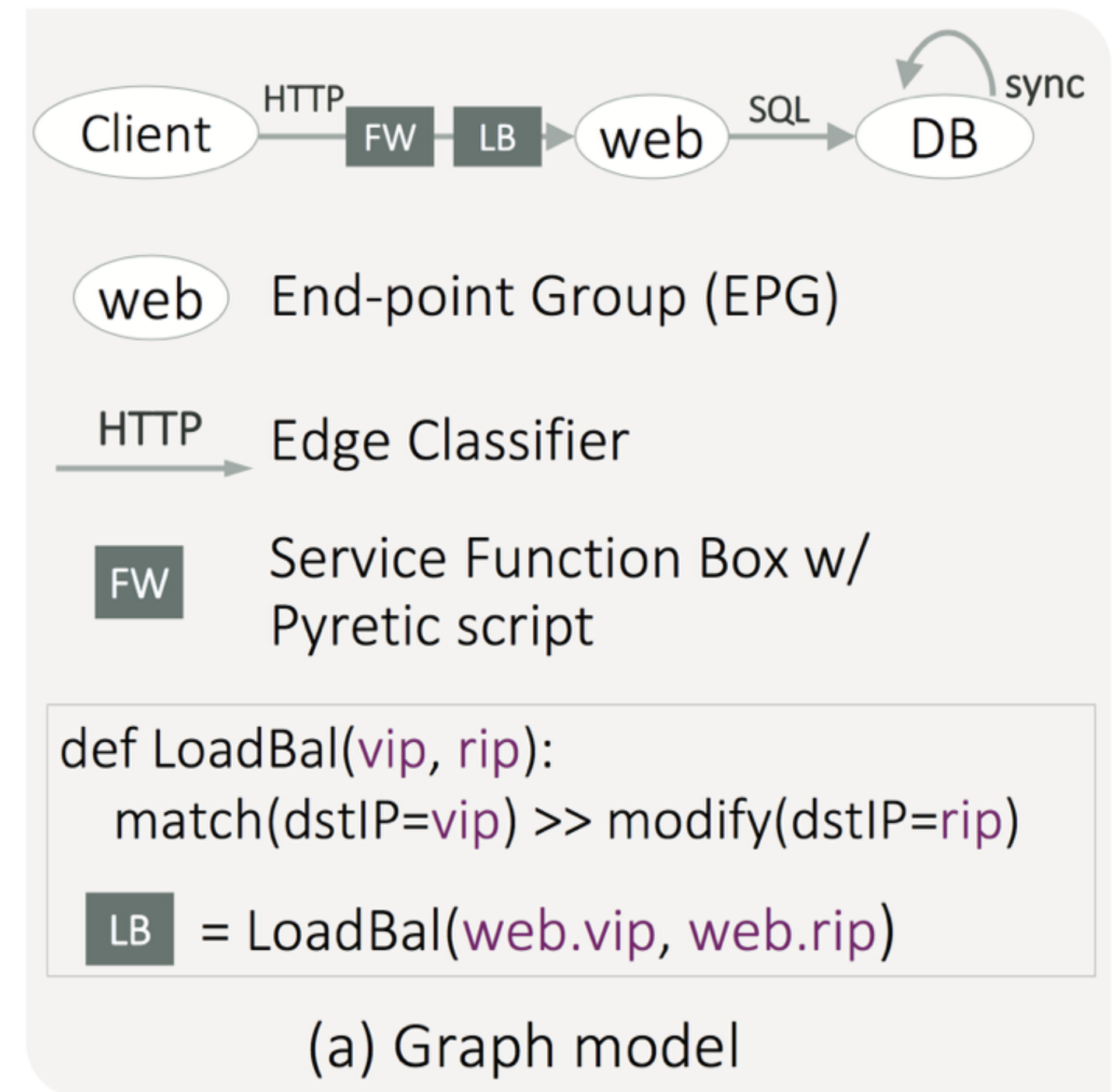
- Input graph
- Composed graph
- Deployed graph
- Low-level configurations/rules



Graph Model

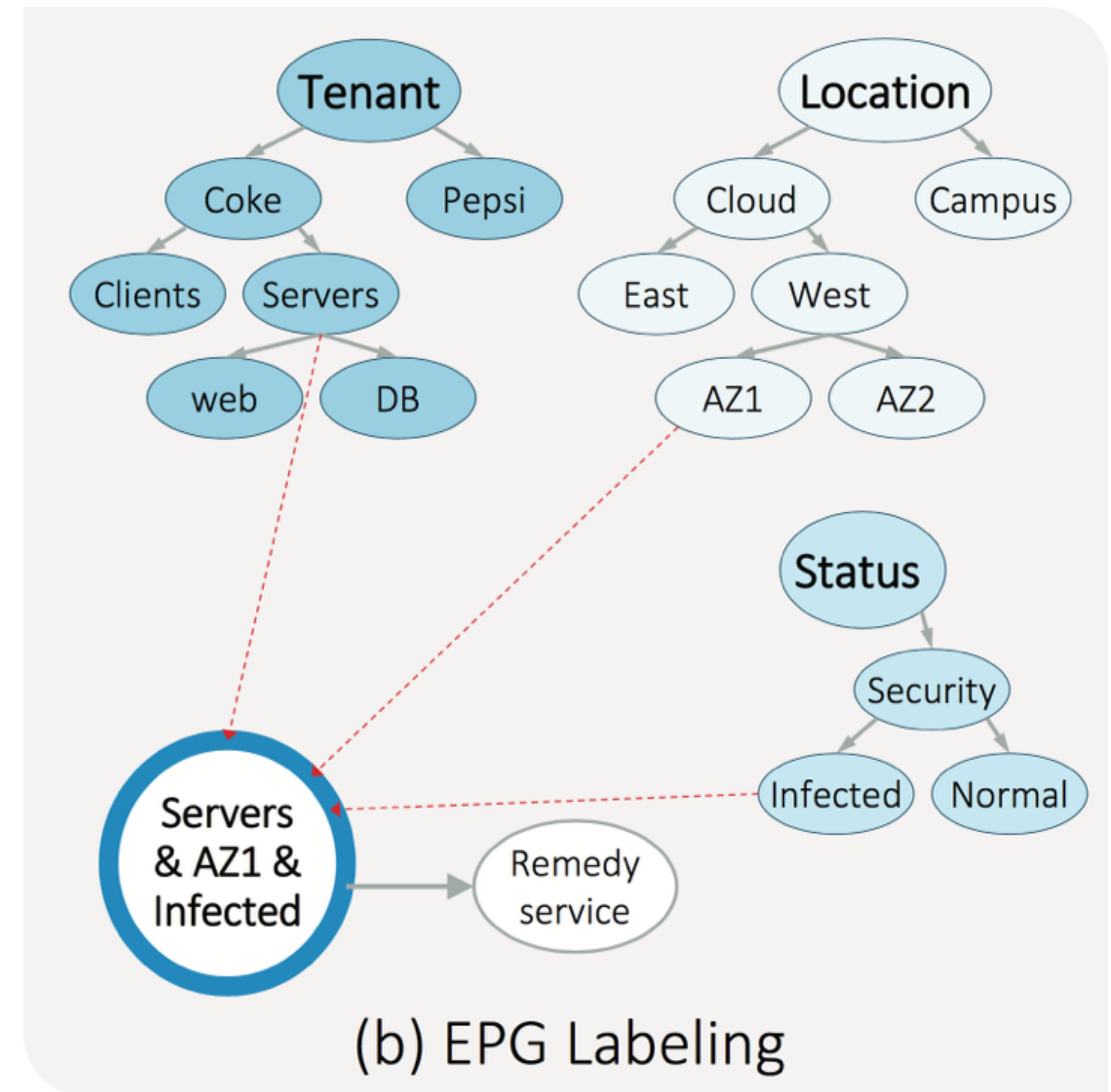
Graph Model

- Vertices
 - vertex: endpoint groups (EPG), comprising all endpoints (EPs) satisfy a membership predicate expressed internally of a *label* or boolean expression over labels.
 - EPG labeling.
- Edges
 - edge: specifies a communication policy between a pair of EPGs, whitelisting model, consists of a *classifier* matching *packet header fields* and a service chain.
 - Composition constraints.
 - Network function box behaviour.

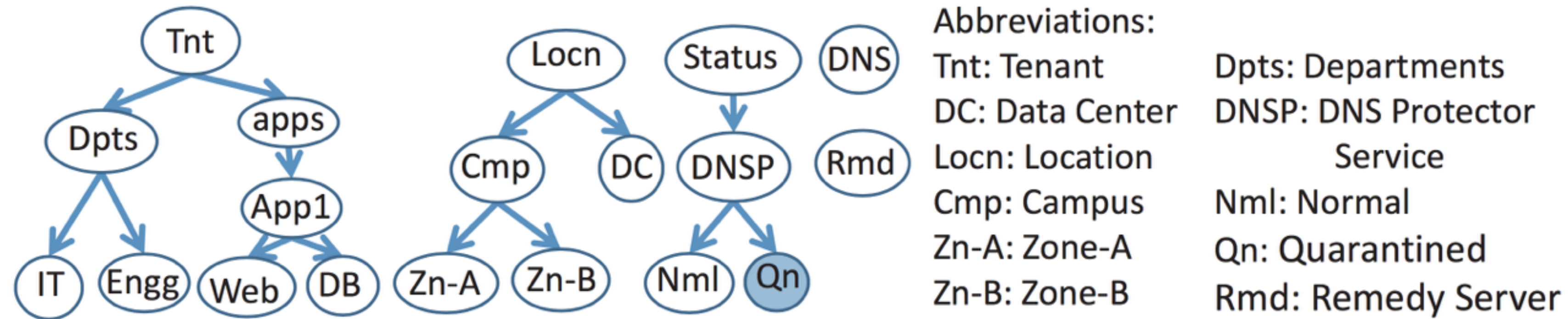


EPG Labeling

- An EPG defined as a boolean expression of logical labels.
- High-level, portable policy specification decoupled from low-level specifics (e.g., IP/MAC addresses).
- Each kind of labels represents endpoint attributes such as tenant ownership, network location or security/QoS status.



EPG Labeling



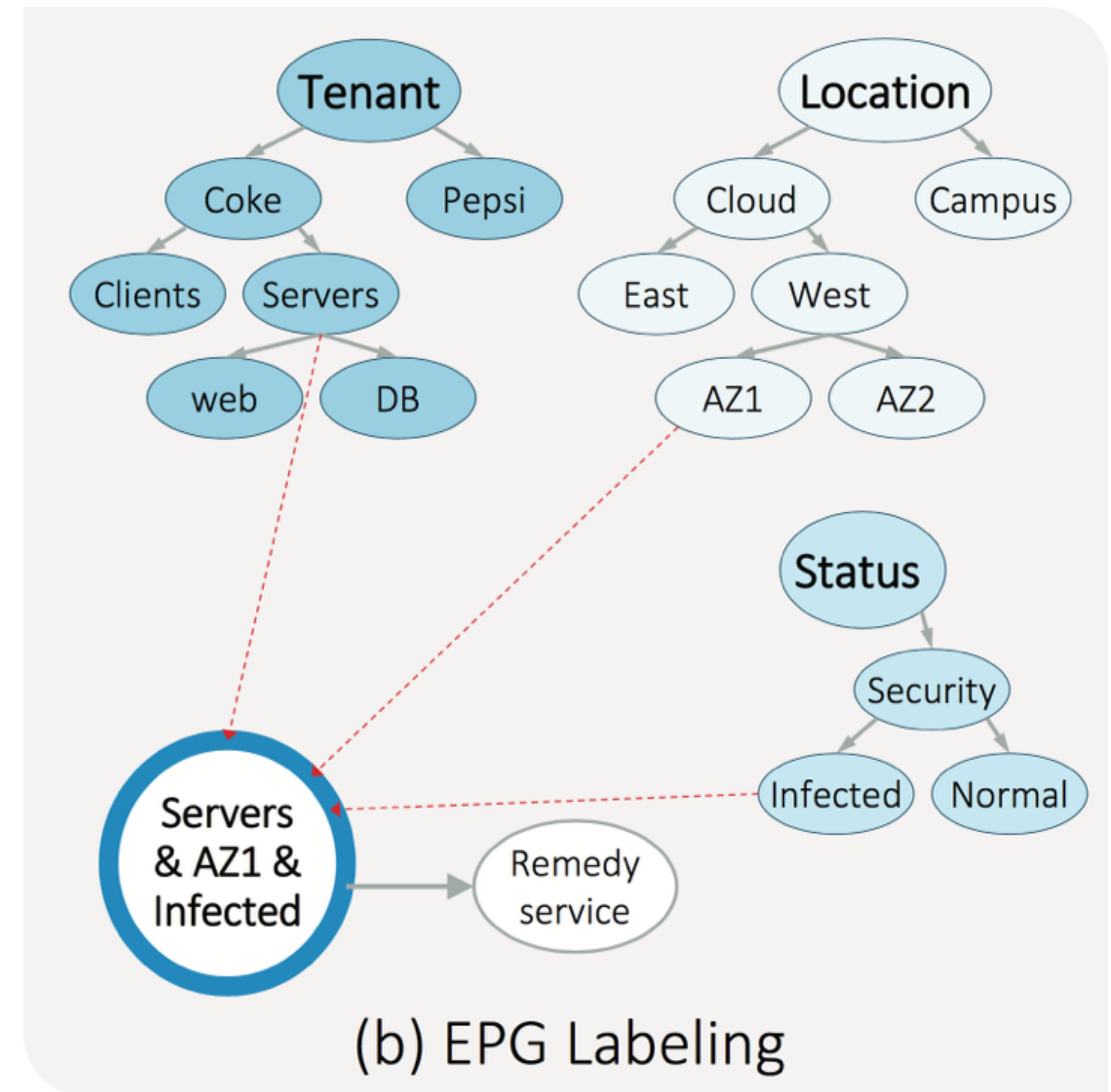
Label mapping inputs:

(IT:Zone-A, Engg:Zone-B, App1:DC, Departments:DNS protector)

- Label hierarchy and label mapping
 - Overlapping or mutually exclusive relation
 - Mutually exclusive labels cannot be assigned on any common endpoint.
 - Two labels from different trees that can be assigned on a common endpoint.


EPG Labeling

- PGA collects label information from external services (tenant authentication, VM placement, SDN control apps, etc.) and displays them in the graph editor to help users easily and correctly create EPGs from the labels.



Composition Constraints

- Edge type.
- Constraints on a policy changes when the policy is composed with other policy graph.

 'Exclusive' EPG: no change to its edges

 MUST allow

 CAN communicate

 Block

 Conditional

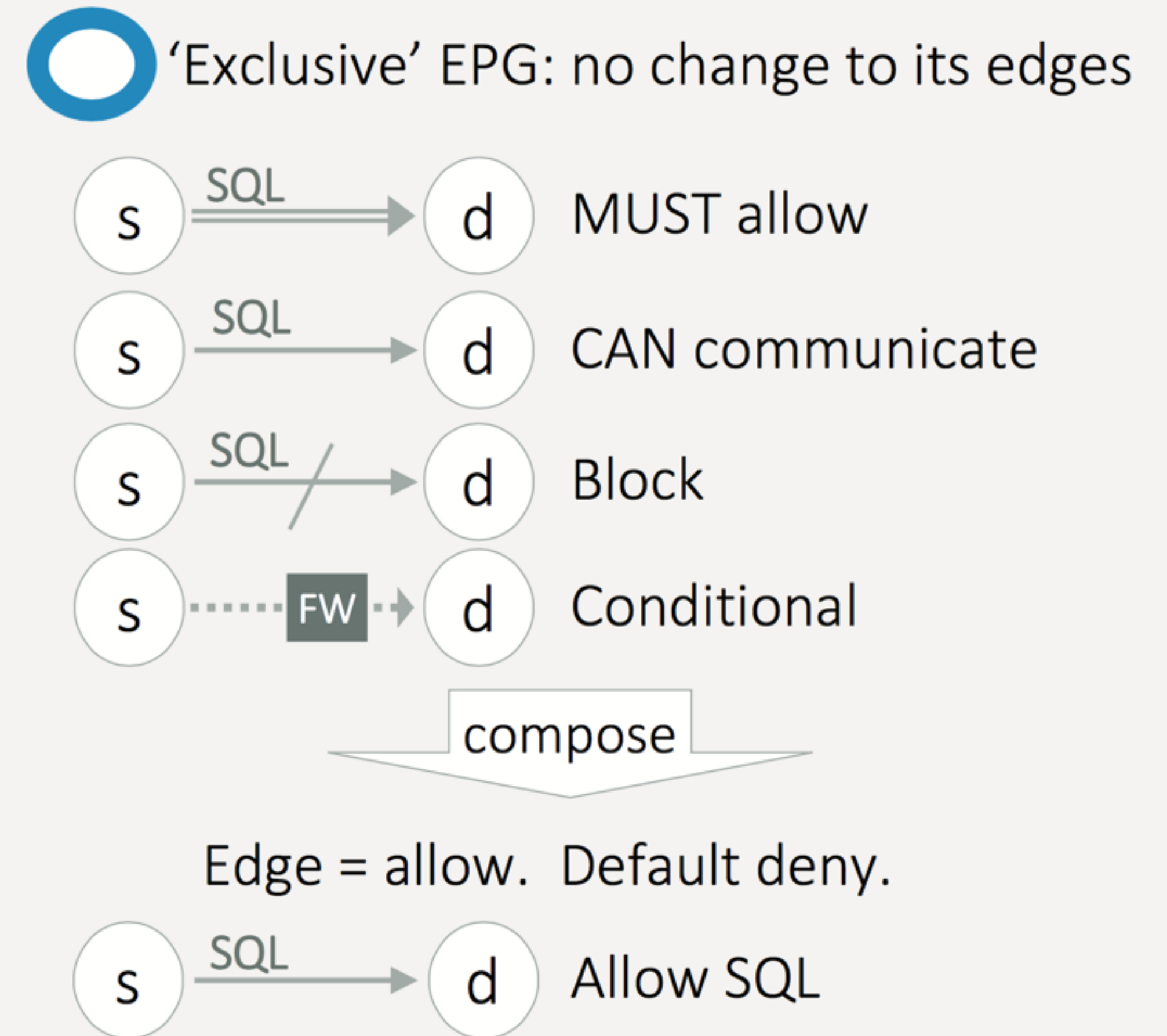
compose

Edge = allow. Default deny.

 Allow SQL


Composition Constraints

- MUST allow
 - Management traffic and service-critical communications.
- CAN communicate
 - ITs opens a set of TCP port for developers.
- Block
 - The infected traffic is not allowed.
- Conditional
 - No need to allow in this graph but it can be allowed in other graphs.



Composition Constraints


- Must/CAN overrides Conditional.
- Block overrides CAN.
- A conflict between Must and Block is reported to users or can be solved based ranking between policies or stakeholders.
- 'Exclusive' EPG: not allow any change or addition of edges of the EPG.

 'Exclusive' EPG: no change to its edges

 MUST allow


 CAN communicate

 Block

 Conditional

compose

Edge = allow. Default deny.

 Allow SQL

Network Function Box Behaviour

- The bounding behaviour of NF Box.
- White boxing, gray boxing.
- Model accurately bounds the output header packet space of the middlebox.
- Not support network functions that duplicate and forward packets along different paths.

```
def LoadBal(vip, rip):  
    match(dstIP=vip) >> modify(dstIP=rip)  
  
LB = LoadBal(web.vip, web.rip)
```

Graph Composition

Graph Composition

- A PGA policy: *match + action*, EPG labels and edge classifiers form the match space, edge types and service chains constitute the action part.
- Take the union of the match spaces of the input graphs.
- Inherit actions from the input graphs for the non-overlapping match spaces.
- Combine actions for the overlapping (intersecting) match spaces subject to composition constraints.

Graph Composition

- Normalisation of input graph
- Graph union

Normalisation of Input Graph

- Translate input graph EPGs to EPGs with globally disjoint membership.
- Replace each composite label with its leaf label equivalents.
- Replicate and merge composition constraints and edge policies to the normalised graph.
- Replicate: EPG (S, D) translated to (S_i, D_j) , $\forall i=1..m, \forall j=1..n$.
- Merge: EPG (G, H) overlaps with EPG (S, D).

Graph Union

- Take union of all the EPGs of the individual graphs.
 - EPGs in two different graphs are either disjoint or equal.
- Copy and merge directed edges from the individual graphs to the composed graph.
 - Check the whether an edge's classifier satisfies the constraints.
 - Check the whether a new classifier overlaps with the existing classifier.

Graph Union

- Combine service chains for the intersecting space
- Determine a proper ordering of service functions in the composed chain, according to the service function model of PGA.
- Each network function is converted to a set of prioritised *match-action* rules indicating the In Packet Space and Out Packet Space.

Graph Union

LB:

Policy:

match: (dstip, Web.virtual_ip) >>
modify: (dstip: Web.real_ips)

Byte Counter:

Policy:

match: (dstip, DC.real_ips) >>
count_bytes: {group_by: [dstip]}

Rules:

match: (dstip, Web.virtual_ip) ->
set([modify: (dstip, Web.real_ips)])

Identity -> set([])

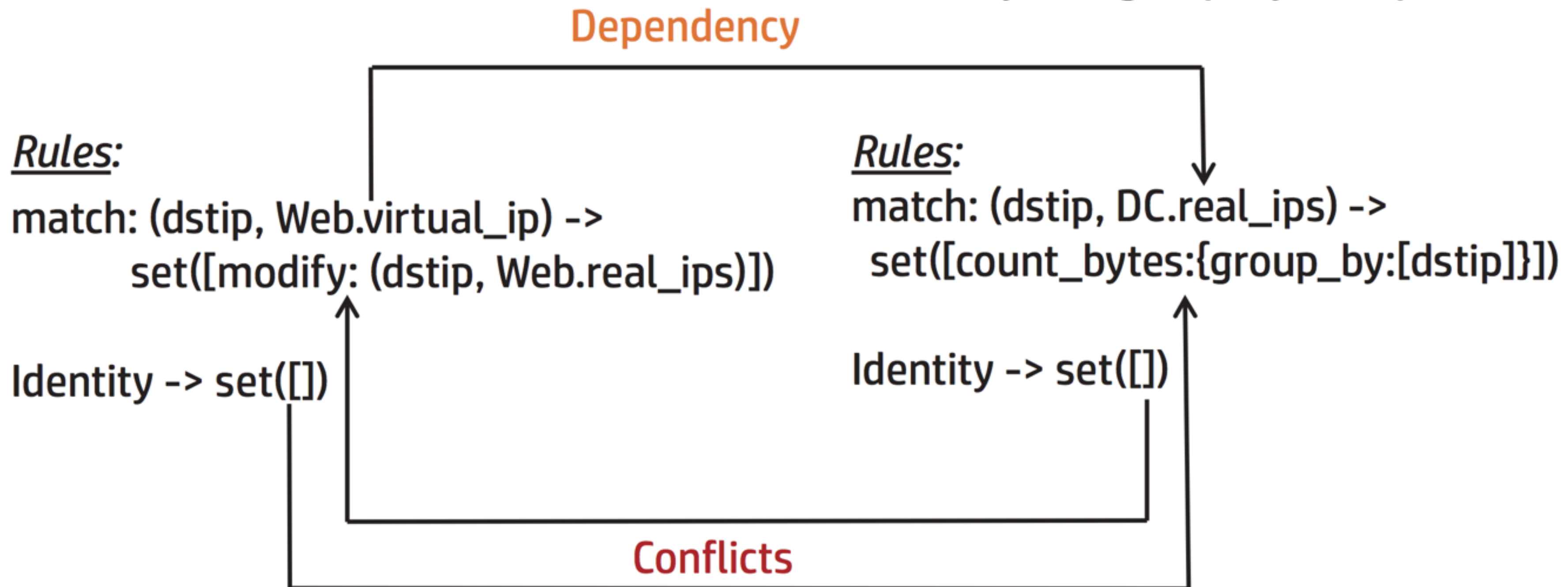
Rules:

match: (dstip, DC.real_ips) ->
set([count_bytes:{group_by:[dstip]}])

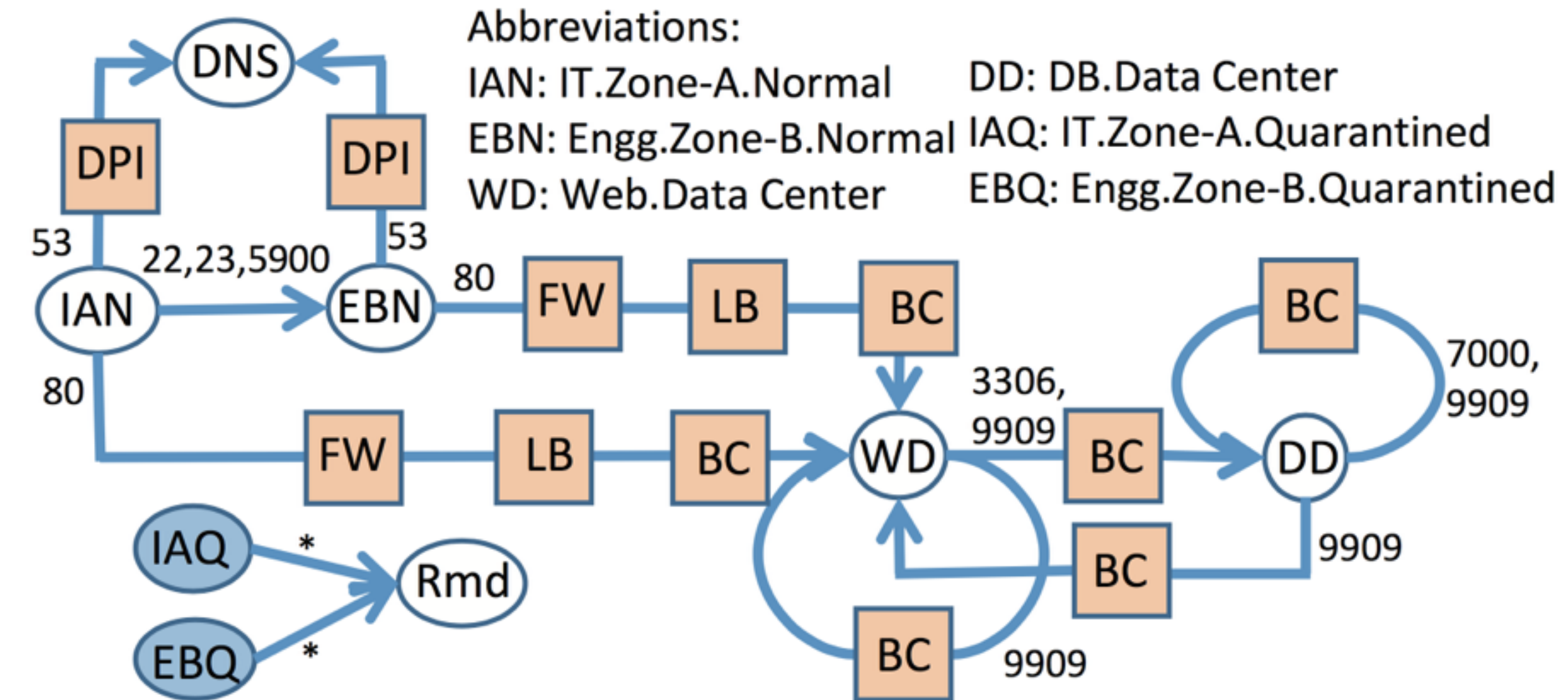
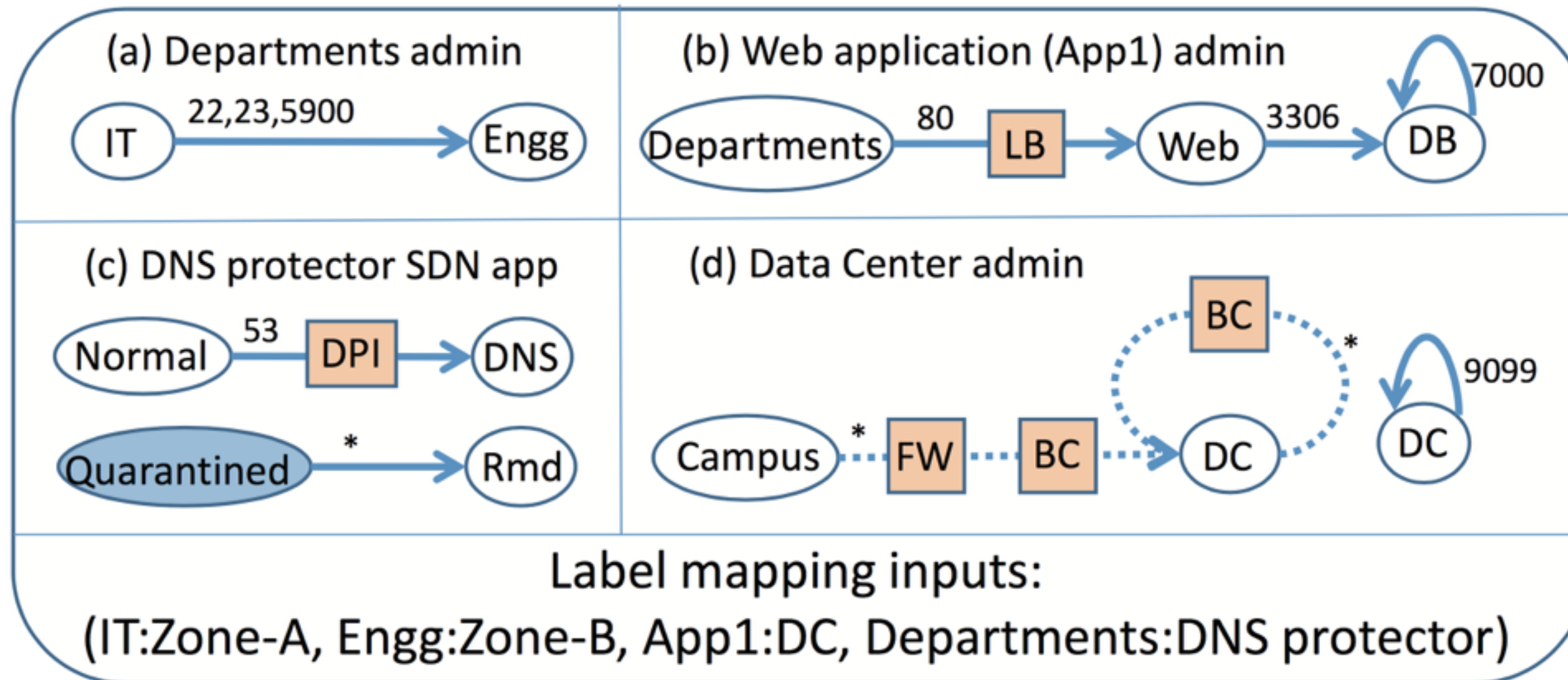
Identity -> set([])

Dependency

Conflicts



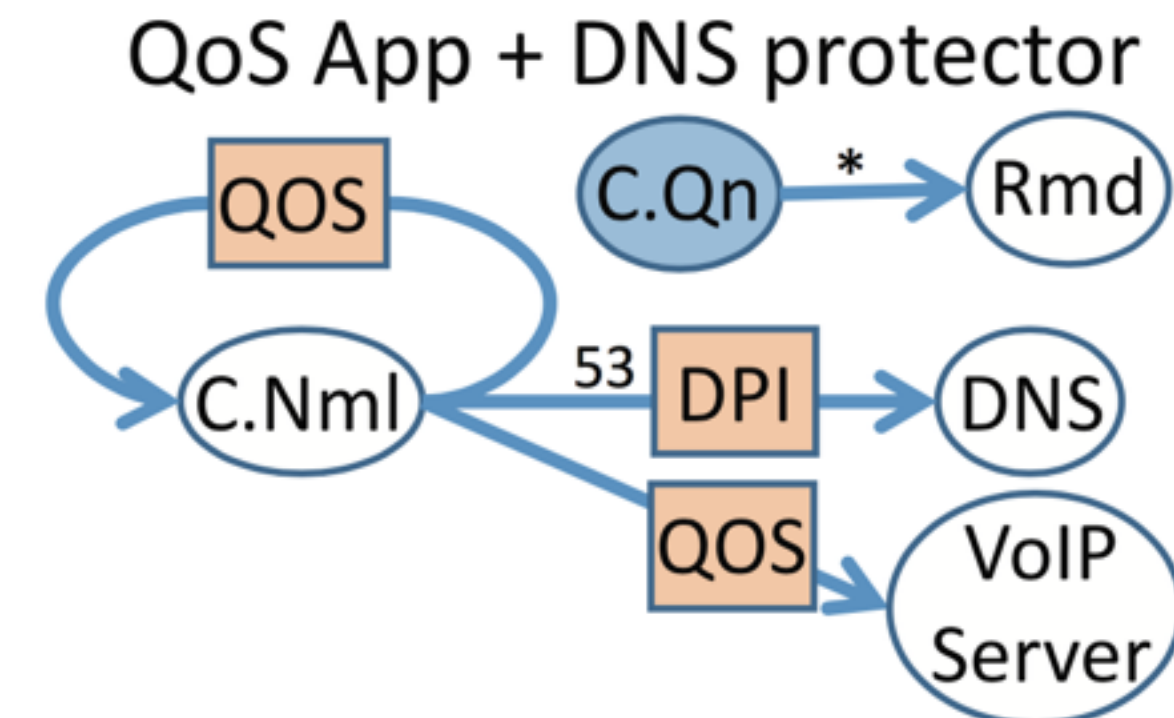
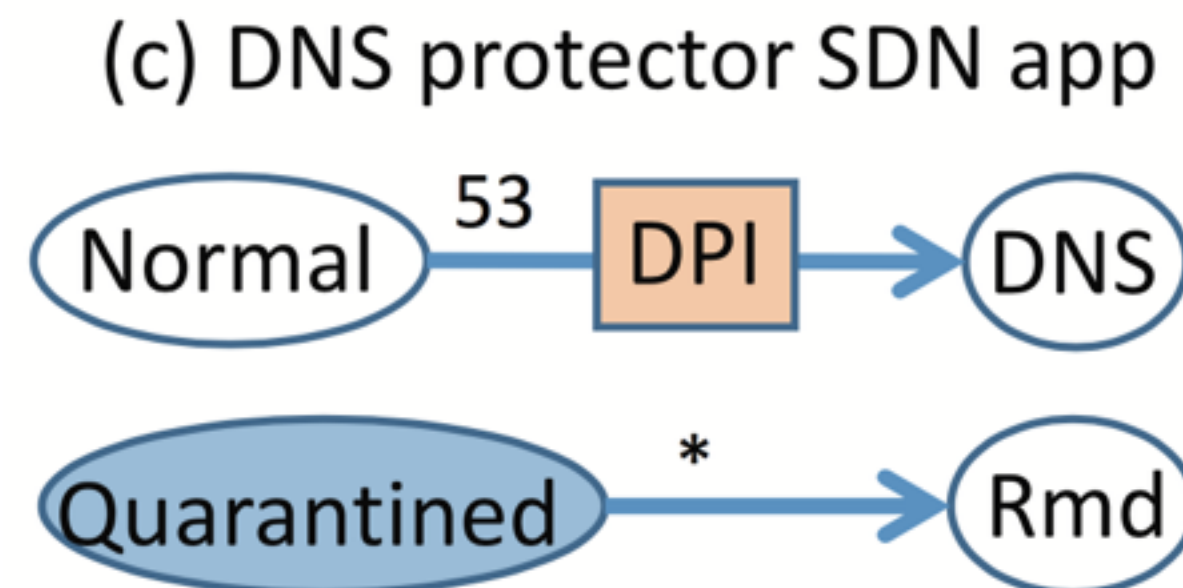
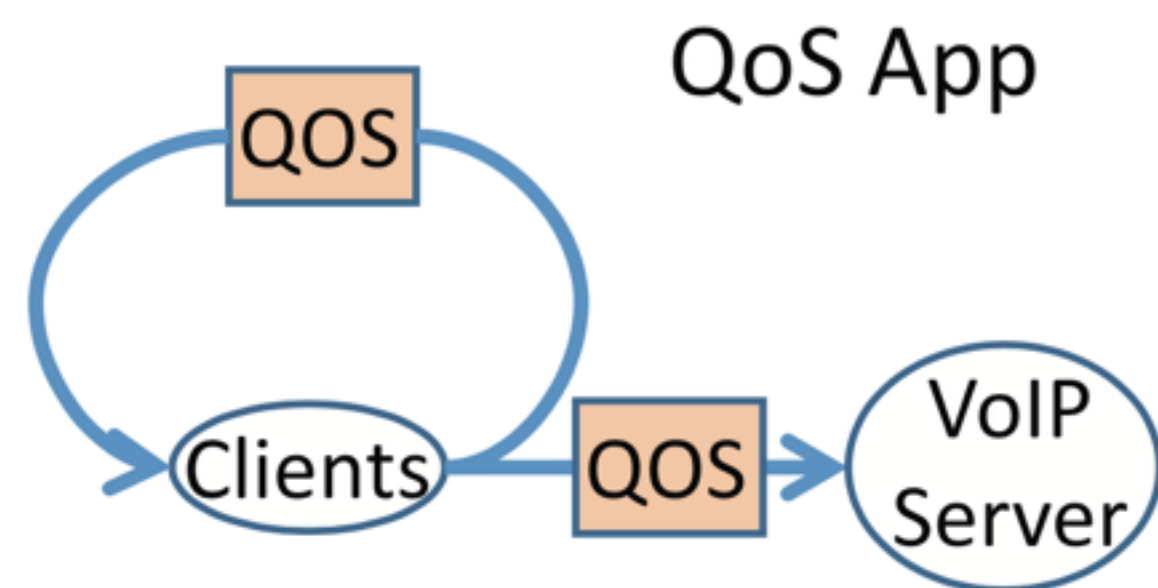
Graph Composition



Case Study

Conflict between SDN apps

- Combining SDN apps at OpenFlow rule level, coupled with prioritisation and IP address assignment, and there is no global priority coordination across the apps.
- PGA's label-based policy specification and graph composition enables eager conflict detection/resolution without requiring prioritised rules and IP addresses.



Large Enterprise

- Policies are written by network admins or application owners, manually reviewed for correctness and consistency, and then deployed in over 30+ global sites.
- Defined for 136 compartments, over 20K ACL, for 4916 EPGs.
- The sizes of EPGs vary, ranging from one IP address to over 600 non-contiguous subnets.
- Aggregating multiple policies as one edge for the same EPG pair, creating a total of 11,786 edges.

Large Enterprise

- 7% of the aggregated policies are redundant.
- 4.7% of outgoing policies are unmatched.
- 4.5% of incoming policies are unmatched.

Prototype and Evaluation

Prototype

- Abstractions
 - EPG: member variables, the set of L4 ports and endpoint IPs.
 - Function Box: extended Pyretic.
 - WhiteList: arbitrary ranges of values for different flow header fields.
- System operation
 - Deployed graph: stored as an in-memory hash table keyed with the src and dst EPGs.
 - Rule generator: A fully reactive approach instead of proactive rule compilers, installing flow rules on the edge switches within mininet.
 - Runtime verification: VeriFlow runs as a proxy between the controller and switches to detect three event types: reachability setup, blackhole (incomplete routing path) and forwarding loop.

Evaluation

- Data sets
 - D1: the synthetic running example, 8 EPGs and 11 edges
 - D2: the large enterprise dataset, 4K EPGs and over 300K edges
 - D3: D2 with randomly added function boxes, up to 3500 function boxes added
- Proactive composition
- Python, 2.5K SLOC, single thread
- 2x8 Xeon 2.6 GHz cores with 132 GB RAM
- Simple topologies with one or two switches providing connectivity to the end hosts

Evaluation

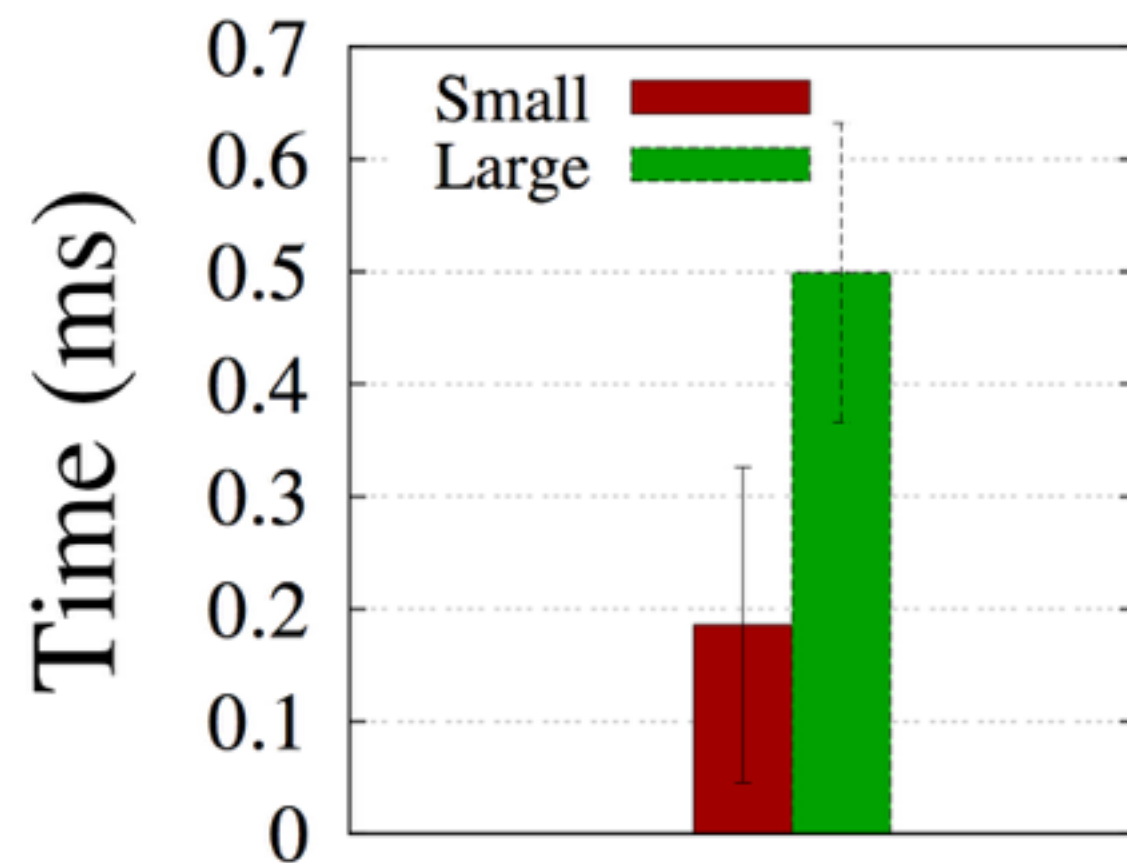


Figure 8: PGA induced additional runtime latency at the controller for D1 (Small), D2 (Large).

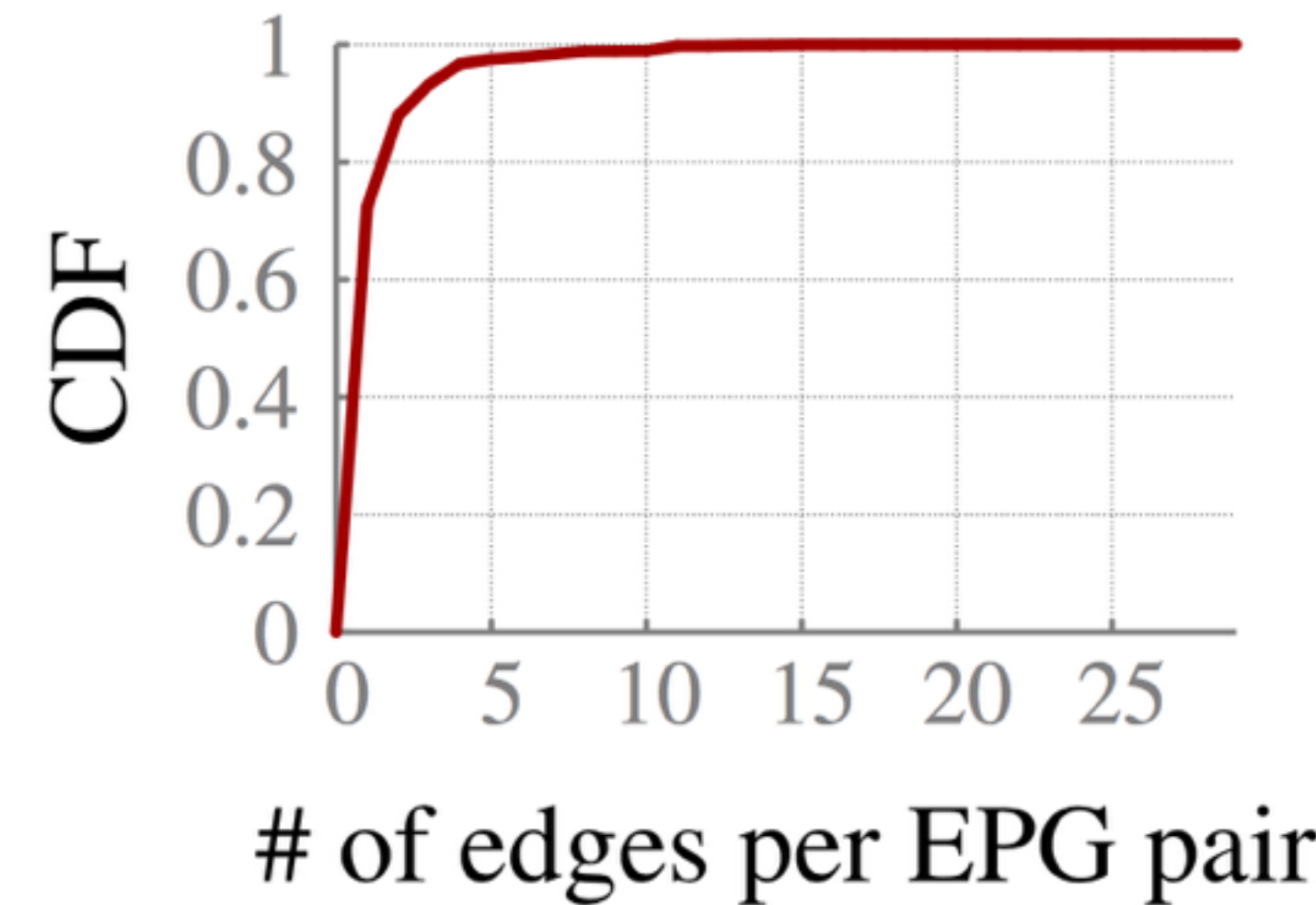


Figure 9: CDF of #edges per EPG pair in composed graph of D2.

Evaluation

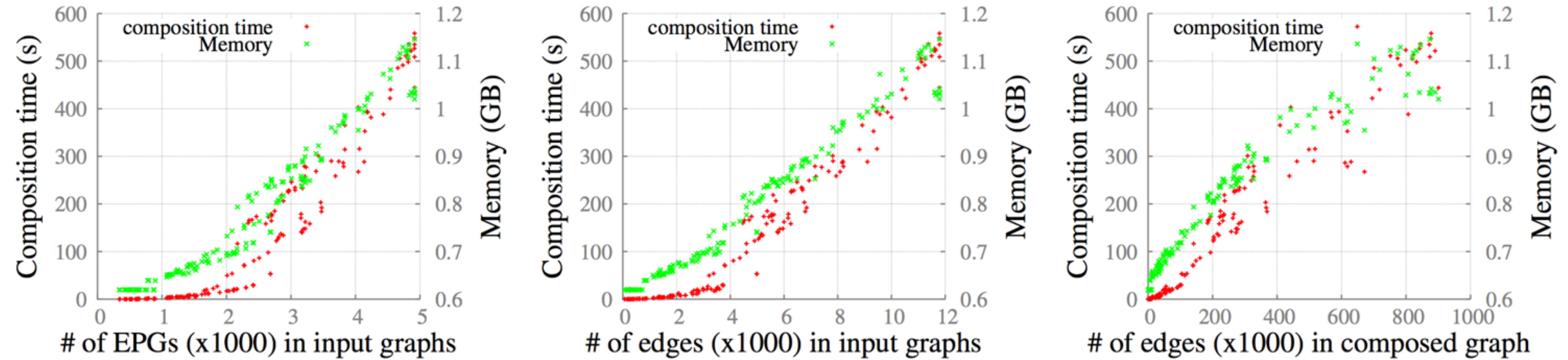


Figure 10: Scalability of PGA with D2.

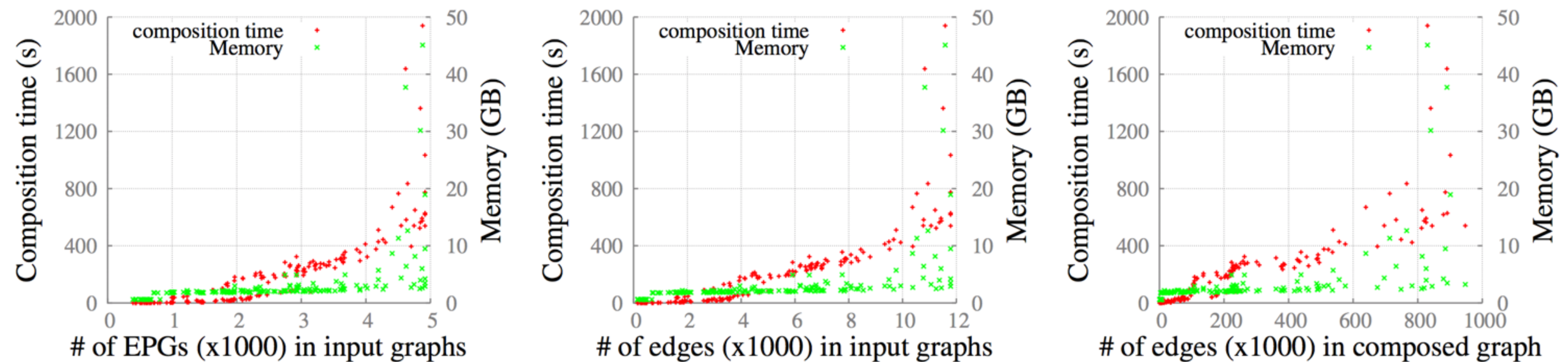


Figure 11: Scalability of PGA with D3 (D2 + function boxes).

Conclusion

- Propose an intuitive graph abstraction to express and compose policies
- The first to model the behaviour of closed middleboxes and ensure their correct behaviour in a service chain
- Future works
 - Update the network in a scalable, responsive and consistent way.
 - Support HM/VM middleboxes, verify their runtime behaviours and chain them in more flexible ways (asymmetric forward/reverse)