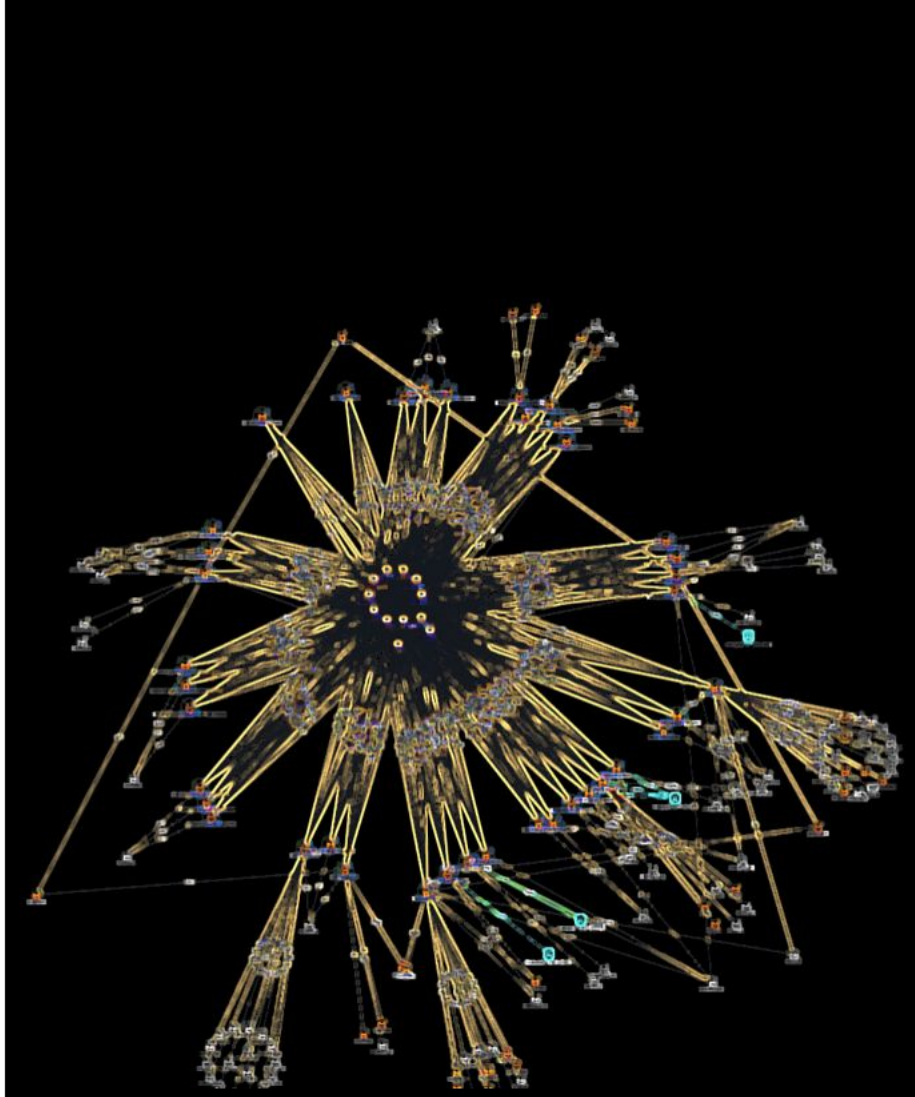
NetSlicer:

Automated and Traffic-Pattern Based Application Clustering in Datacenters

Liron Schiff [1], Ofri Ziv [1], Manfred Jaeger [2] and Stefan Schmid [2,3]

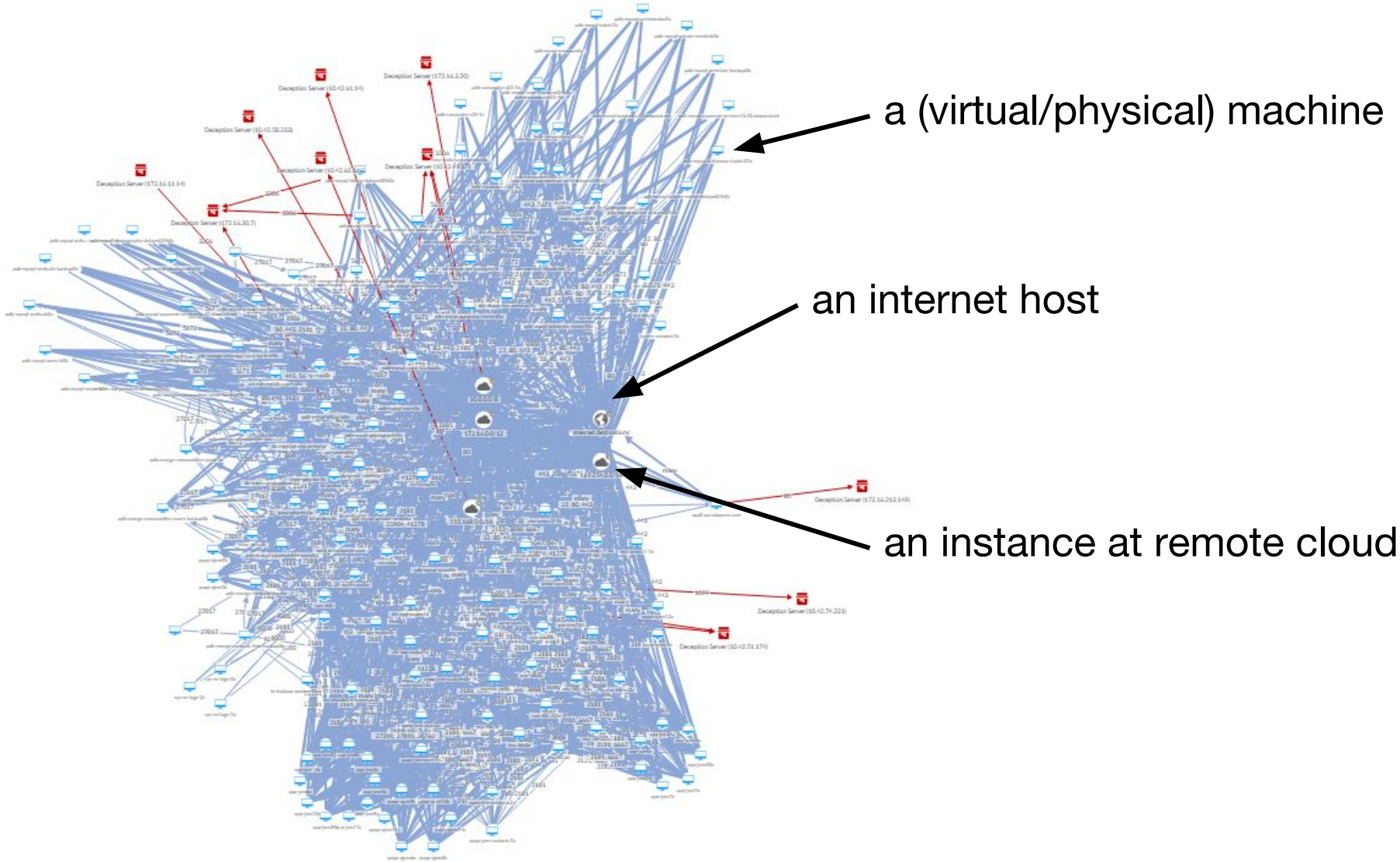
[1)]GuardiCore, Israel [2)]AAU, Denmark [3)]Univie, Austria



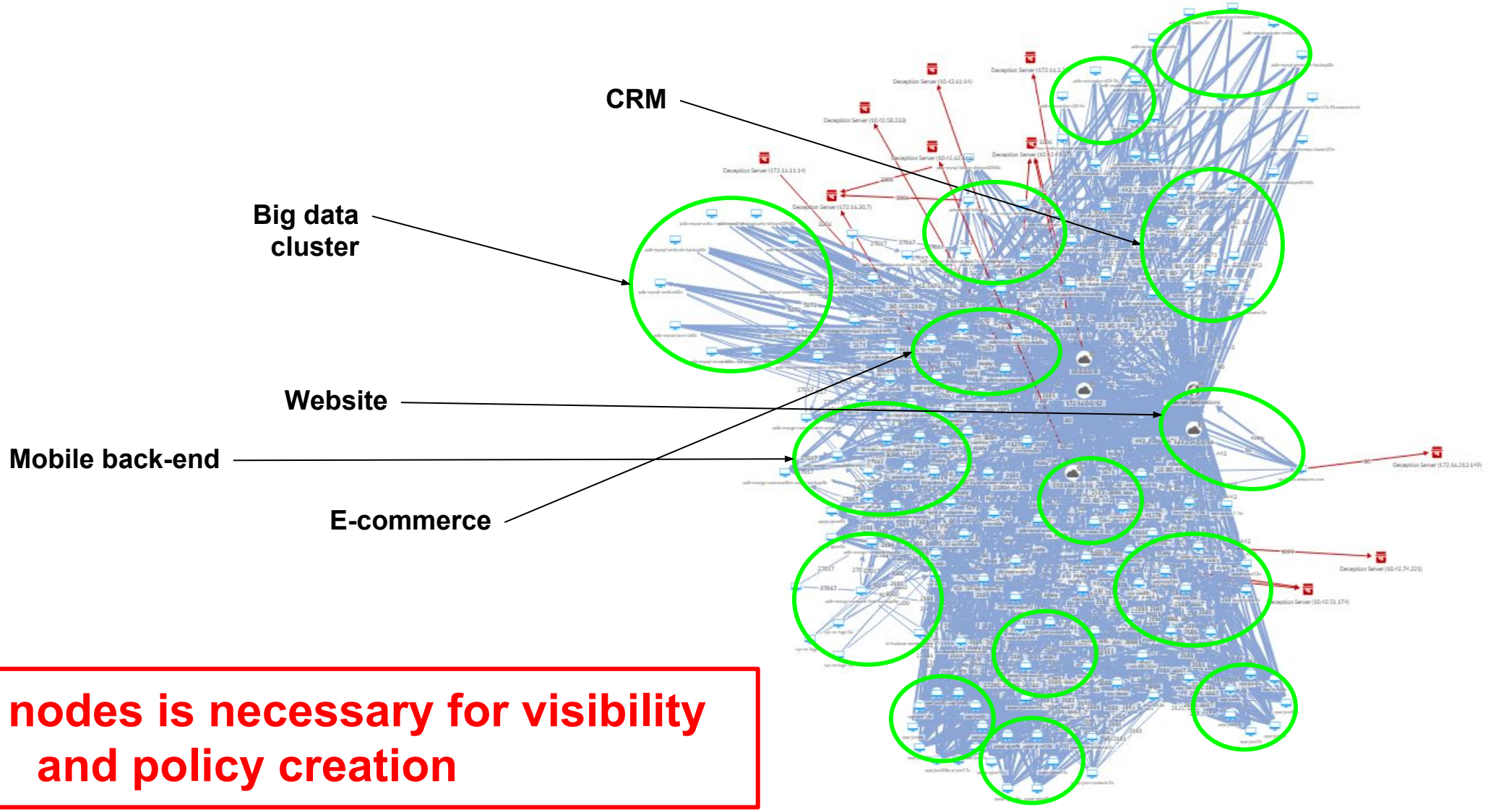
Agenda

- The problem
- Our ML approach
- Challenges we encountered
- The NetSlicer algorithm
- Evaluation
- Use case

The network is the sum of its applications



The network is the sum of its applications

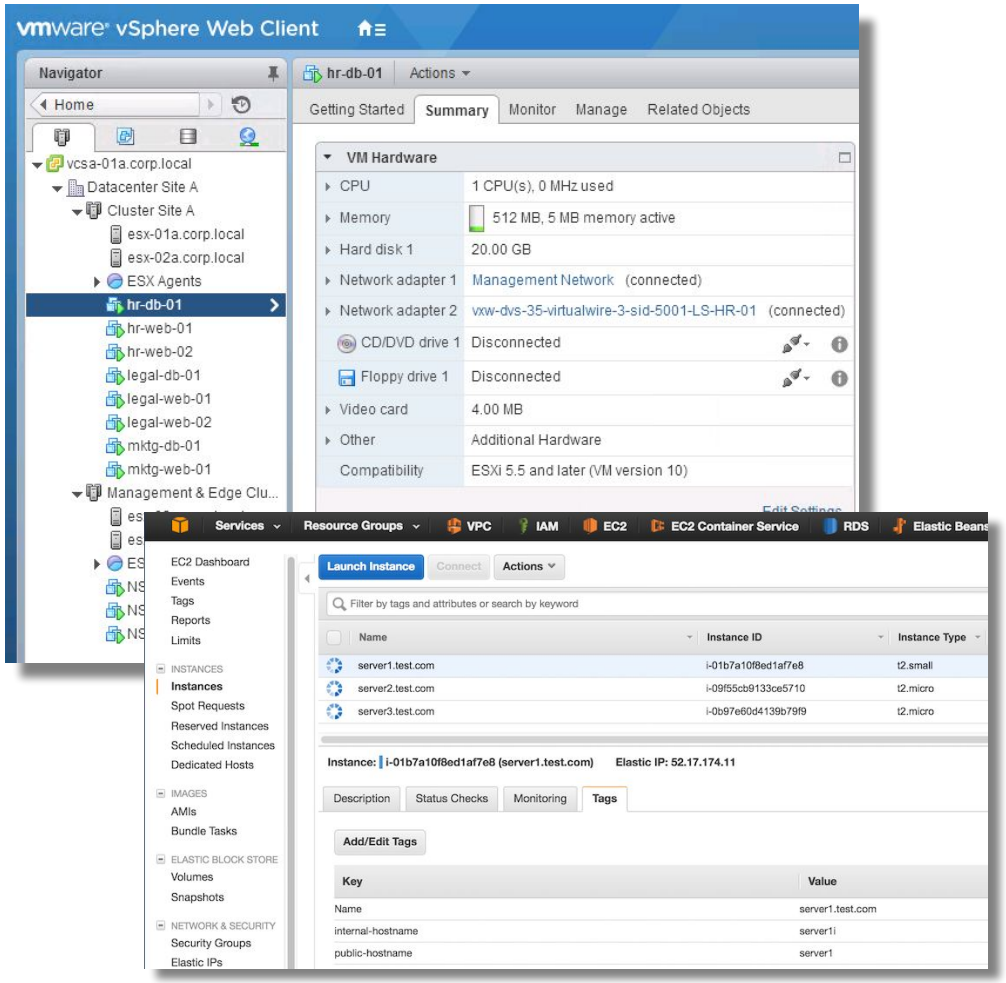


Grouping nodes is necessary for visibility and policy creation

Leveraging existing data to group nodes

Many definitions, many sources

- Orchestration (e.g., tags, names, desc.)
- Inventory (e.g., CMDB)
- DevOps (e.g., puppet recipes, ansible scripts)



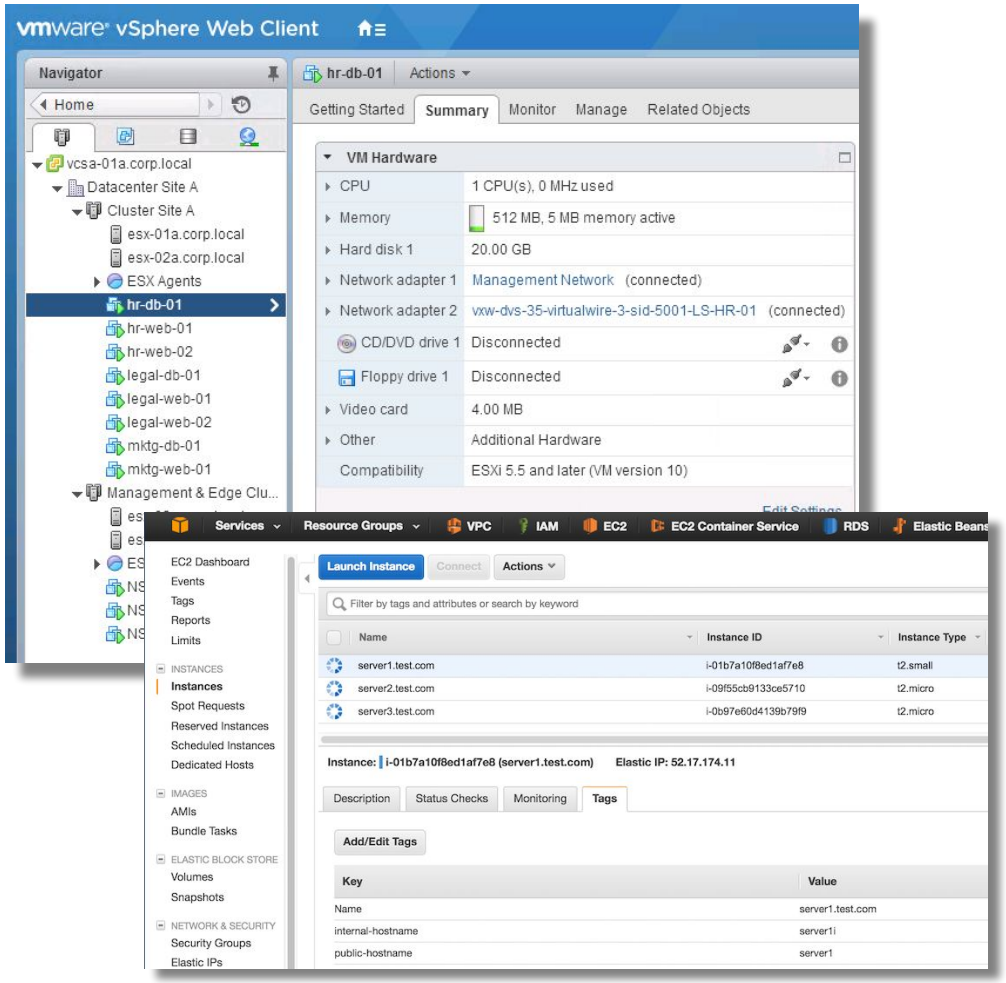
Leveraging existing data to group nodes

Many definitions, many sources

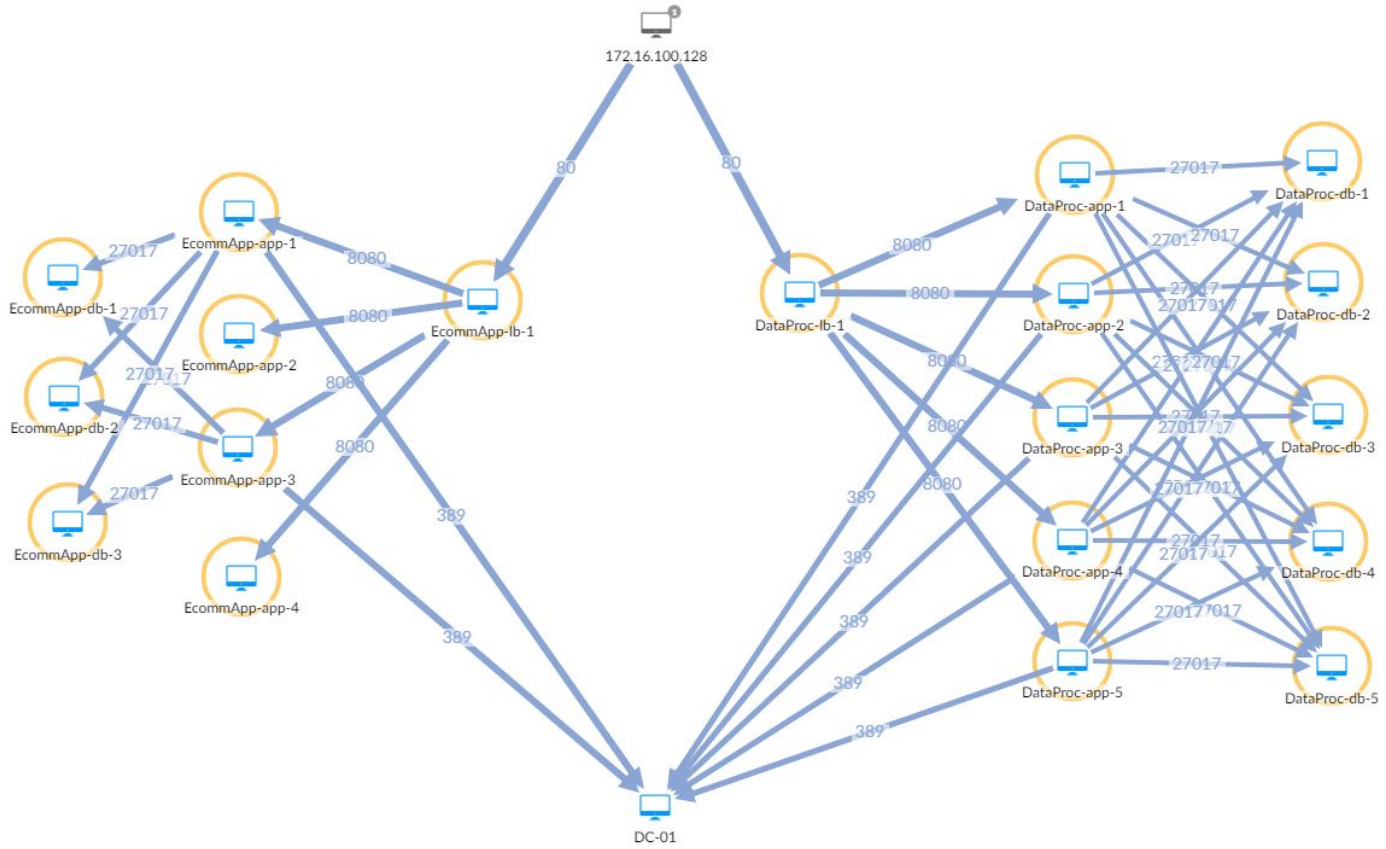
- Orchestration (e.g., tags, names, desc.)
- Inventory (e.g., CMDB)
- DevOps (e.g., puppet recipes, ansible scripts)

Limited by

- Availability (not always defined)
- Relevance (may not be up to date)
- Consistency (may not be standardized)



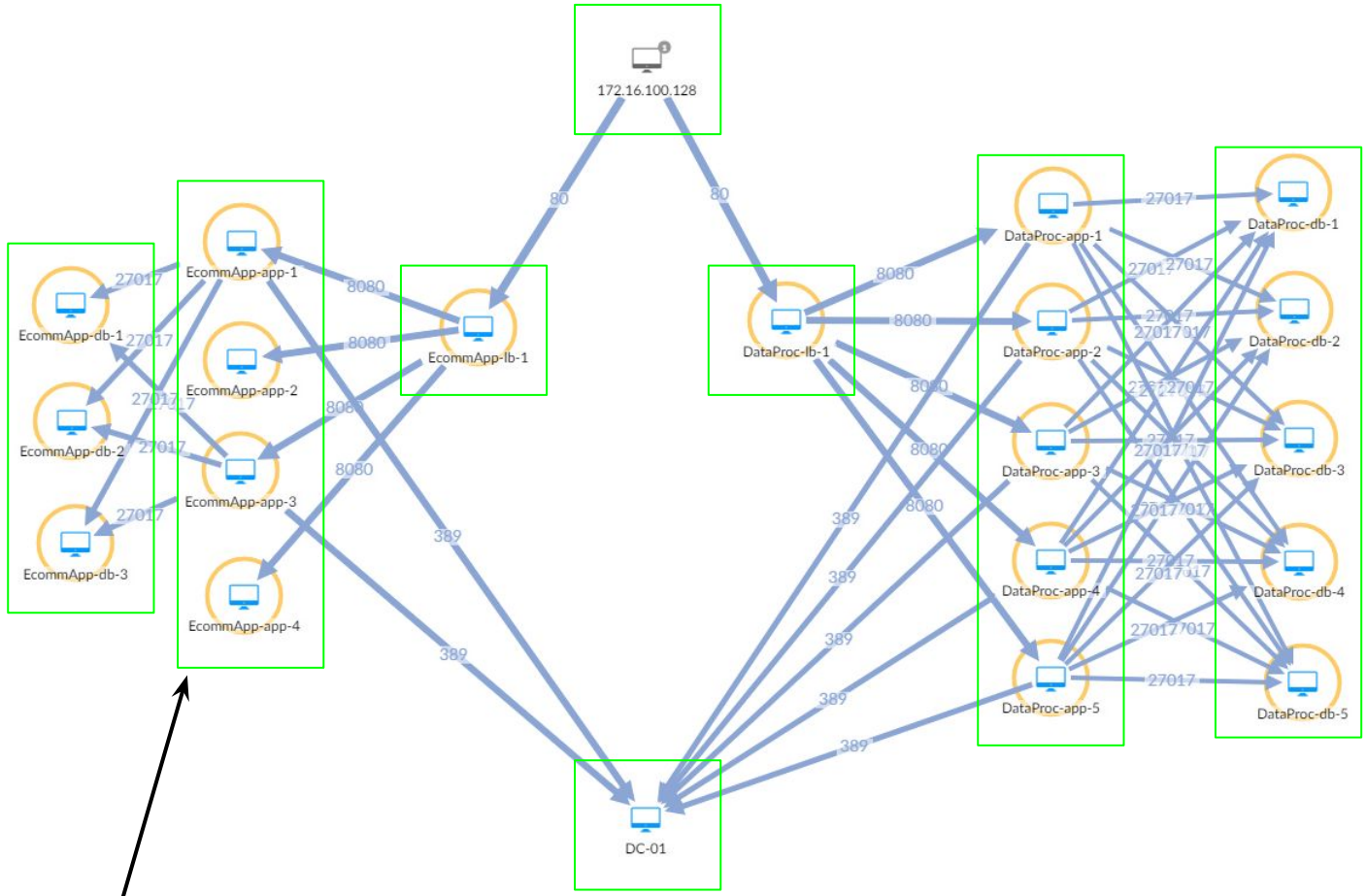
Machine Learning Based Labeling



GuardiCore

Machine Learning Based Labeling

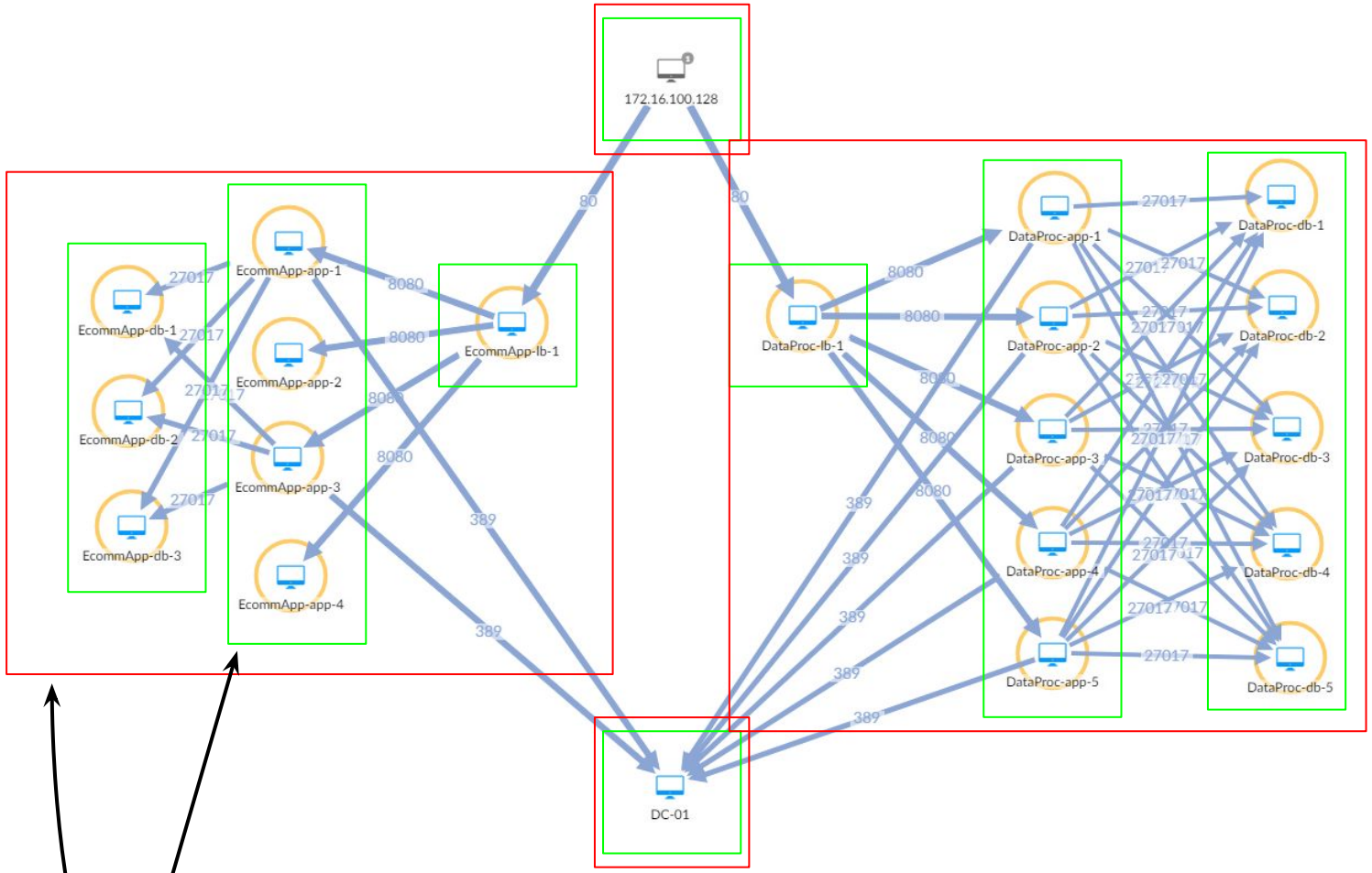
- Cluster nodes into tiers



Tier := a set of nodes with similar connections

Machine Learning Based Labeling

- Cluster nodes into tiers
- Cluster tiers into apps

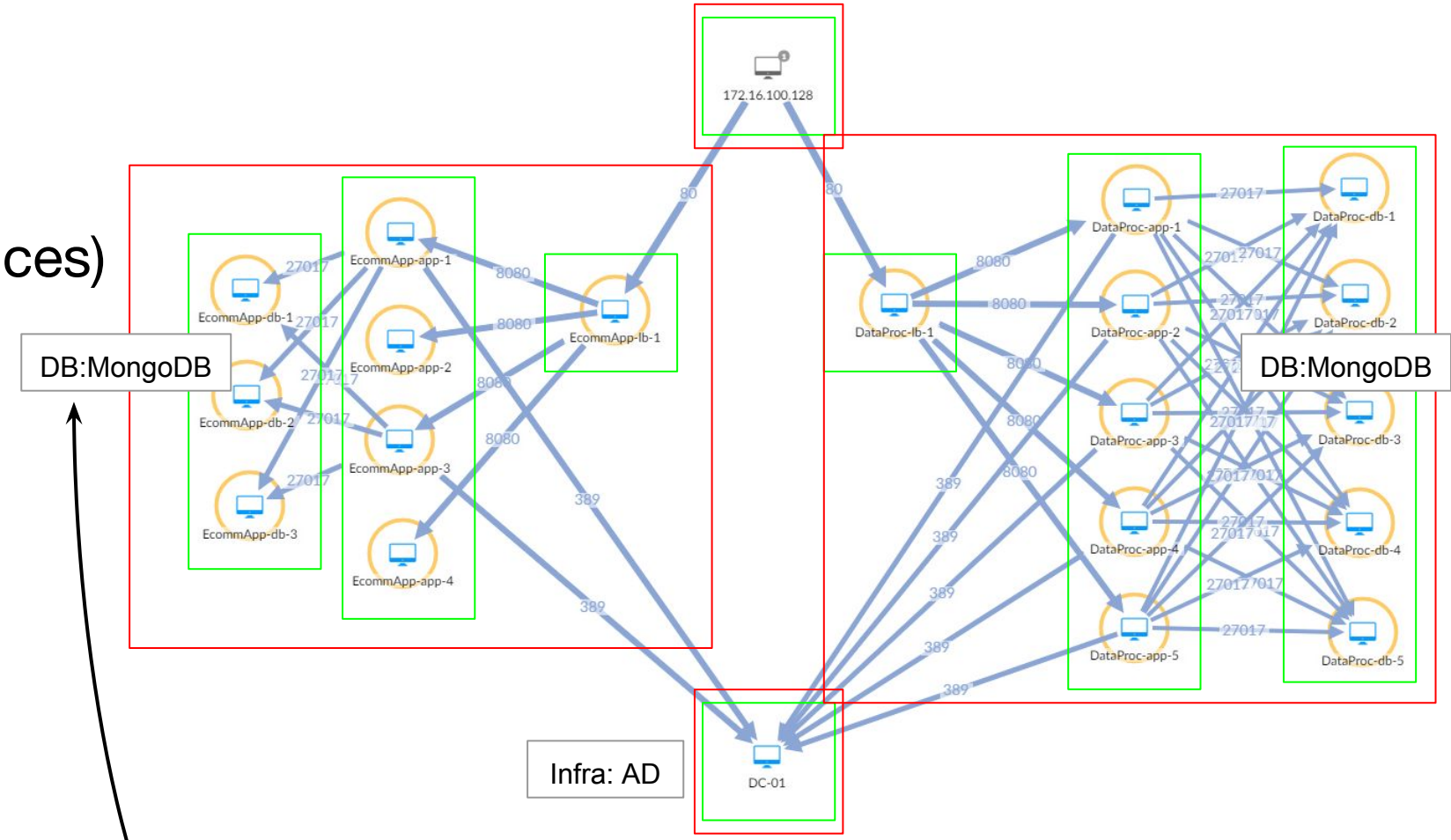


Tier := a set of nodes with similar connections

App := a set of tightly coupled tiers

Machine Learning Based Labeling

- Cluster nodes into tiers
- Cluster tiers into apps
- Identify roles (Infras, services)



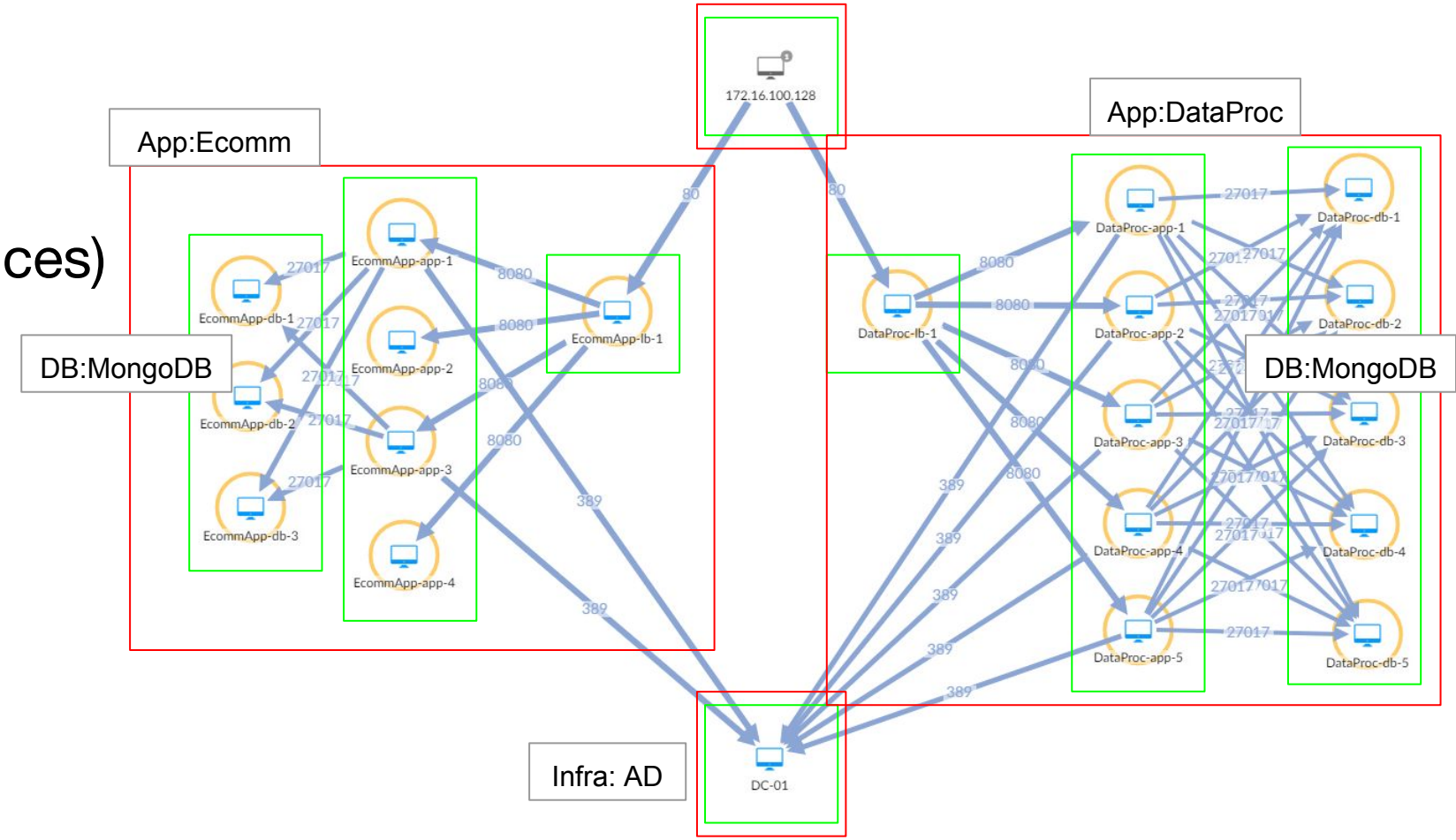
Tier := a set of nodes with similar connections

App := a set of tightly coupled tiers

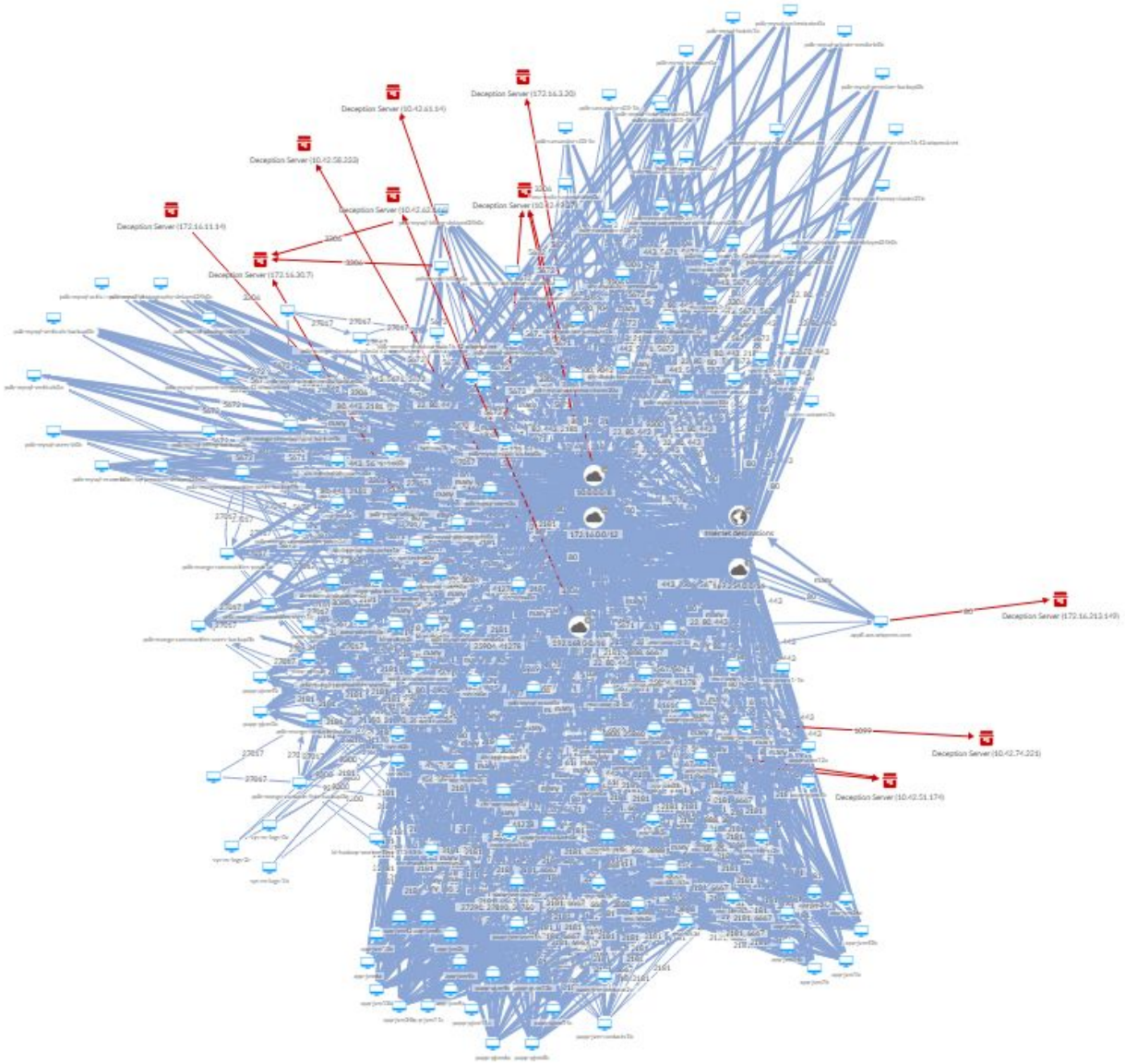
Role := a functionality type (may repeat across the network)

Machine Learning Based Labeling

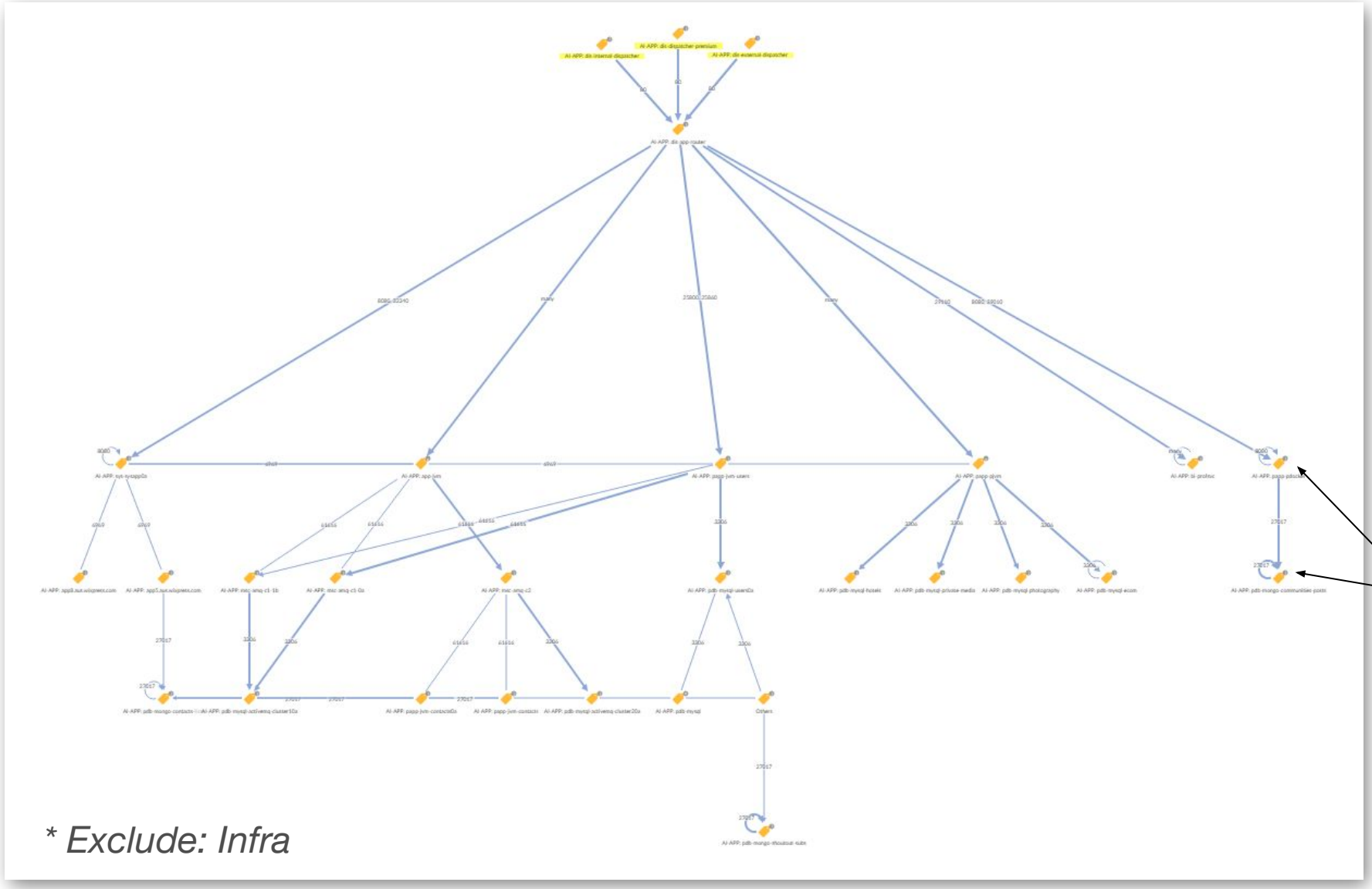
- Cluster nodes into tiers
- Cluster tiers into apps
- Identify roles (Infras, services)
- Generate (simple) labels
  - Based on services
  - Based on machine names



Example - ungrouped



Example - grouped by application



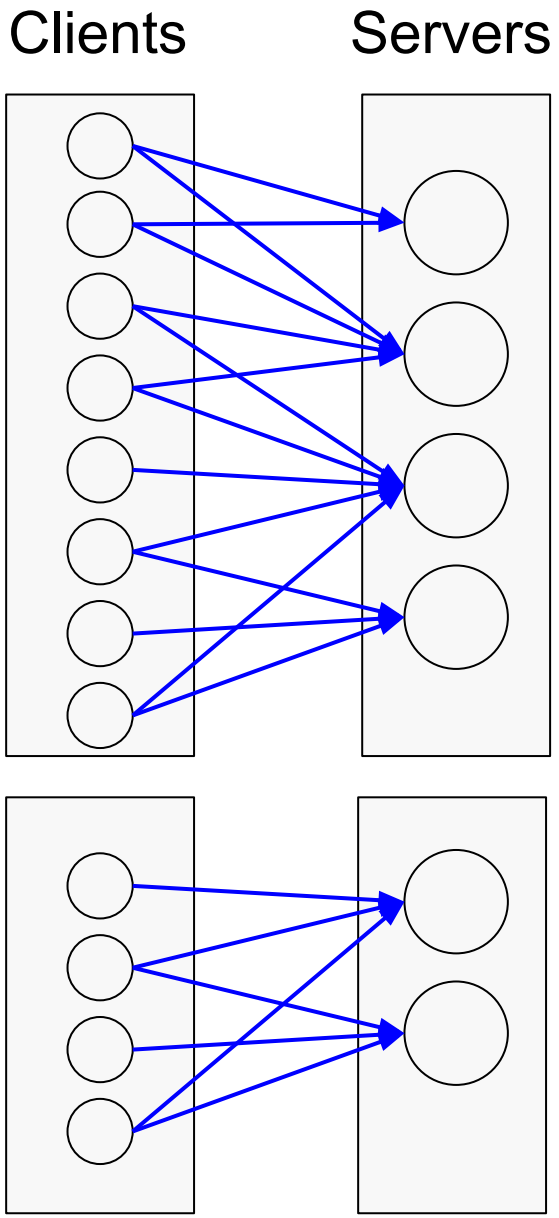
groups of nodes

** Exclude: Infra*

Why is it challenging?

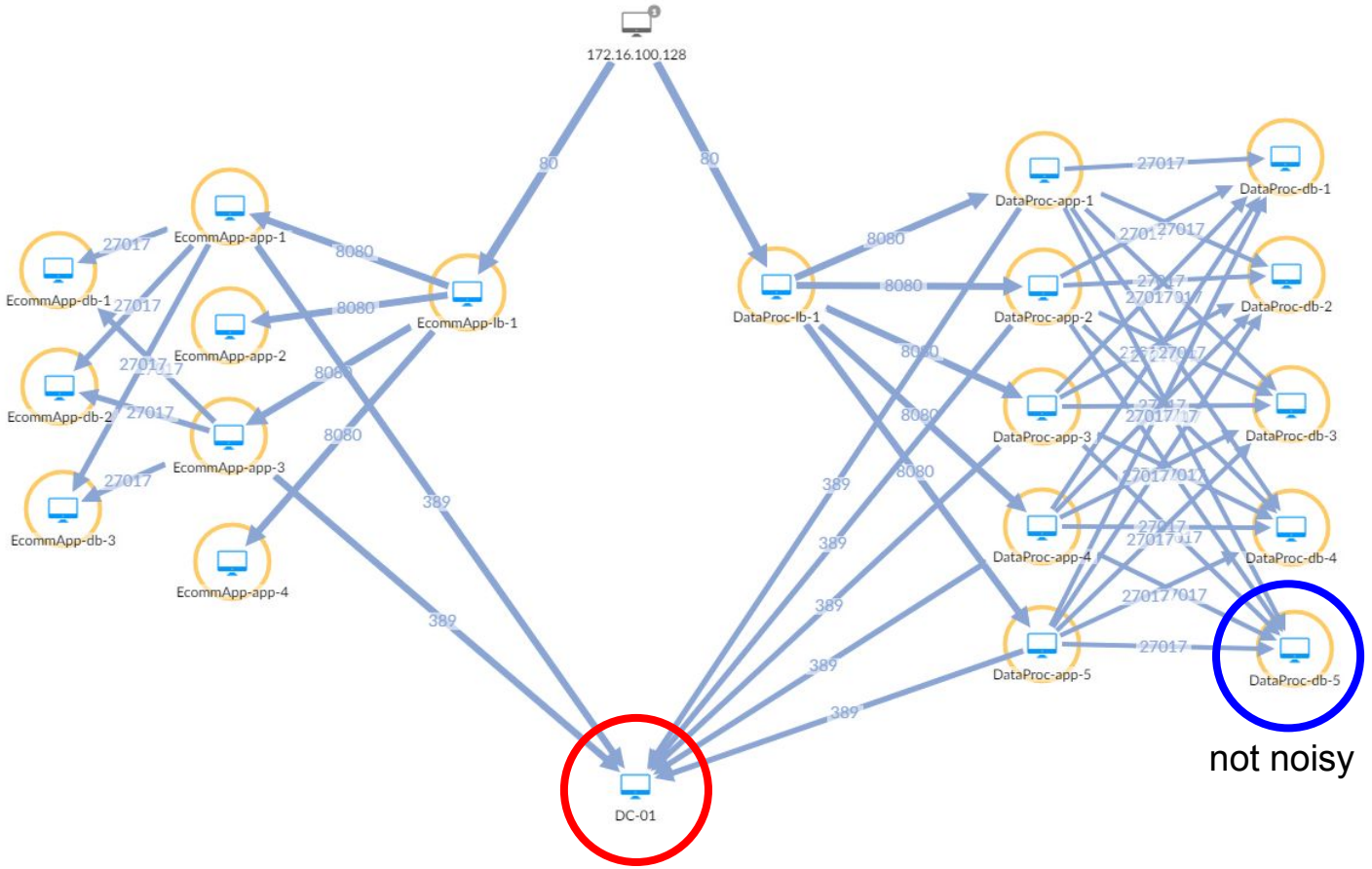
Why is it challenging?

- Load balancing
  - Same tier nodes are not “identical”
  - Different servers may serve different clients



Why is it challenging?

- Load balancing
- Infras and monitors
  - Add “noisy” connections
  - Over-connected network



GuardiCore

Big-DAMA 2018

Why is it challenging?

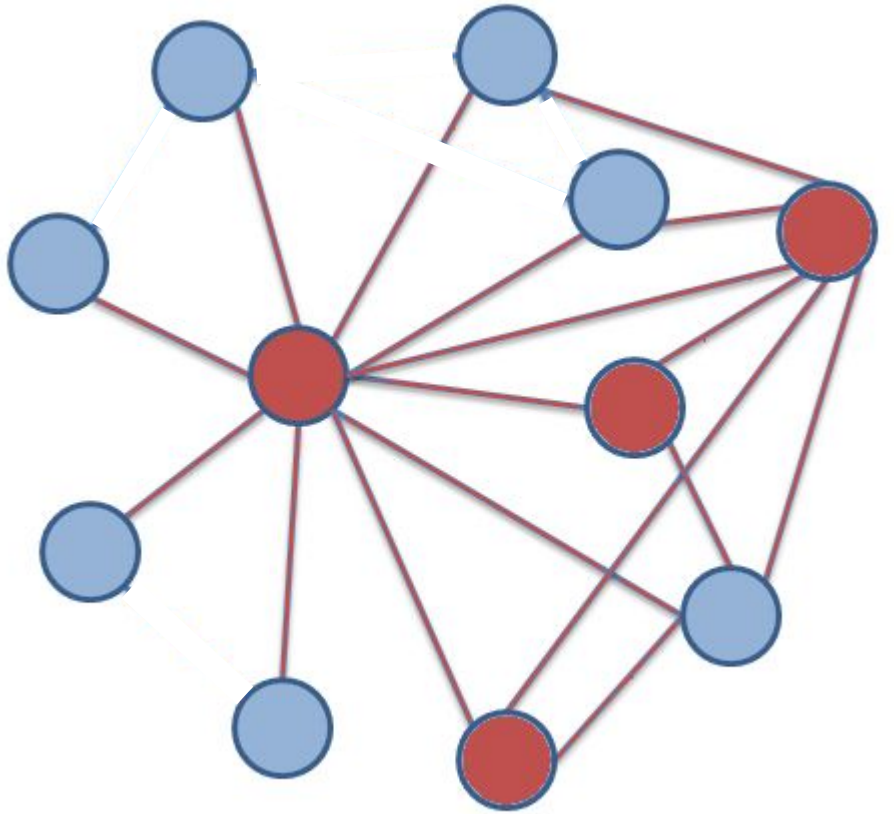
- Load balancing
- Infras and monitors
- Non-standard ports
  - Customized and random port numbers
  - Harder to identify apps and filter noise

Services
Certificate Services (CertSvc) Distributed File System (Dfs) Event Log Exchange Server Fax Service File Replication Local Security Authority (LSASS) Netlogon Remote Storage Terminal Services ...

Protocols
DCOM FTP HADOOP RPC ...

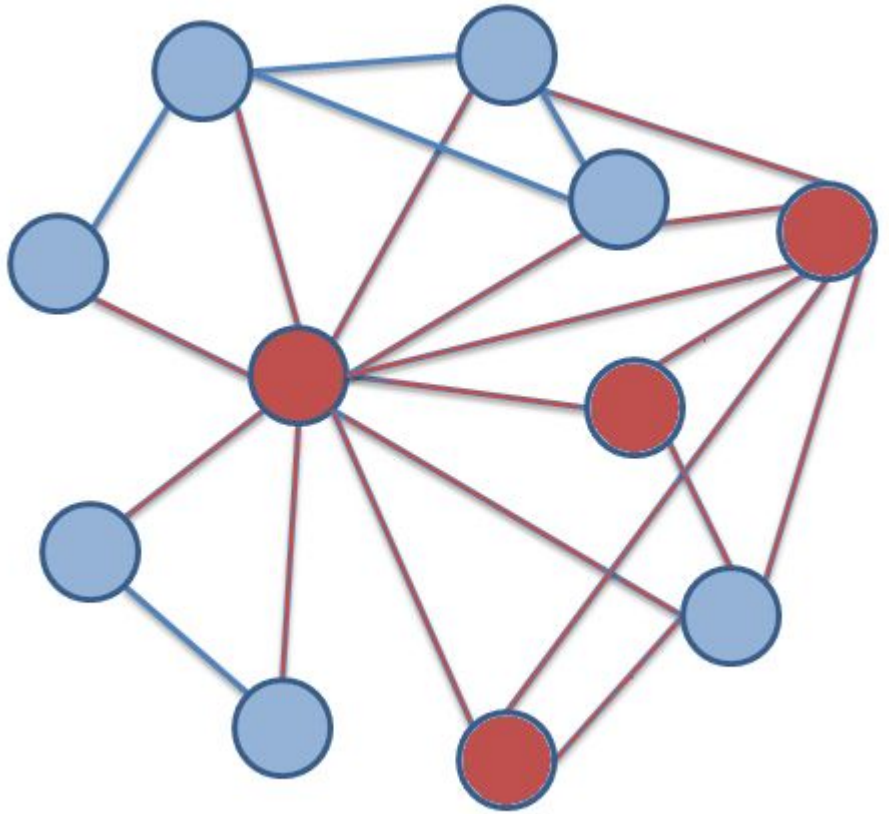
Why is it challenging?

- Load balancing
- Infras and monitors
- Non-standard ports
- Uncovered nodes
  - Missing connection / L7 data



Why is it challenging?

- Load balancing
- Infras and monitors
- Non-standard ports
- Uncovered nodes
  - Missing connection / L7 data



Why is it challenging?

- Load balancing
- Infras and monitors
- Non-standard ports
- Uncovered nodes
- Big data
  - Tens of thousands of nodes (and millions of links)
  - (almost) unlimited external internet hosts

Table 2. Number of data centres by size category and growth rates

Data centres category	No of data centres (2013)	Change in No of data centres in 2008-2013
Server Cabinets (3-10 m^2)	30500	-8%
Server Rooms (11-100 m^2)	18100	+/- 0%
Small Data centres (101-500 m^2)	2150	+23%
Medium Data centres (501-5000 m^2)	280	+27%
Large Data centres (over 5000 m^2)	70	+40%

Source: Hintemann and Clausen, 2014

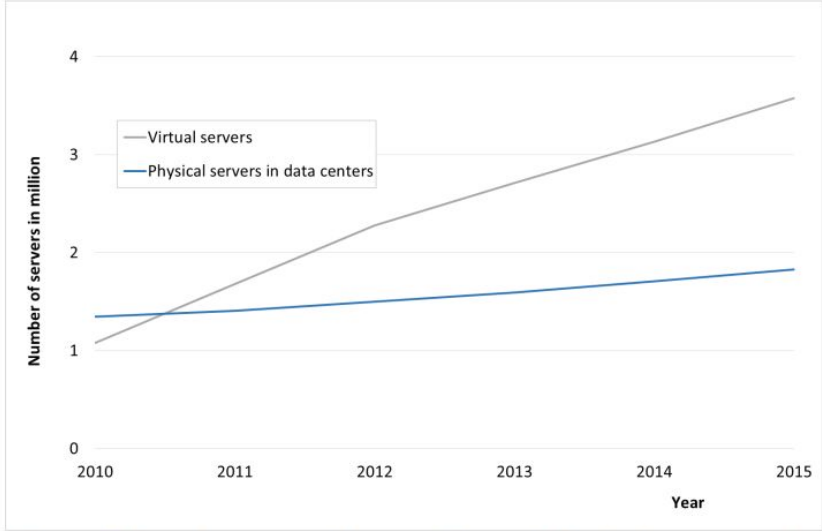
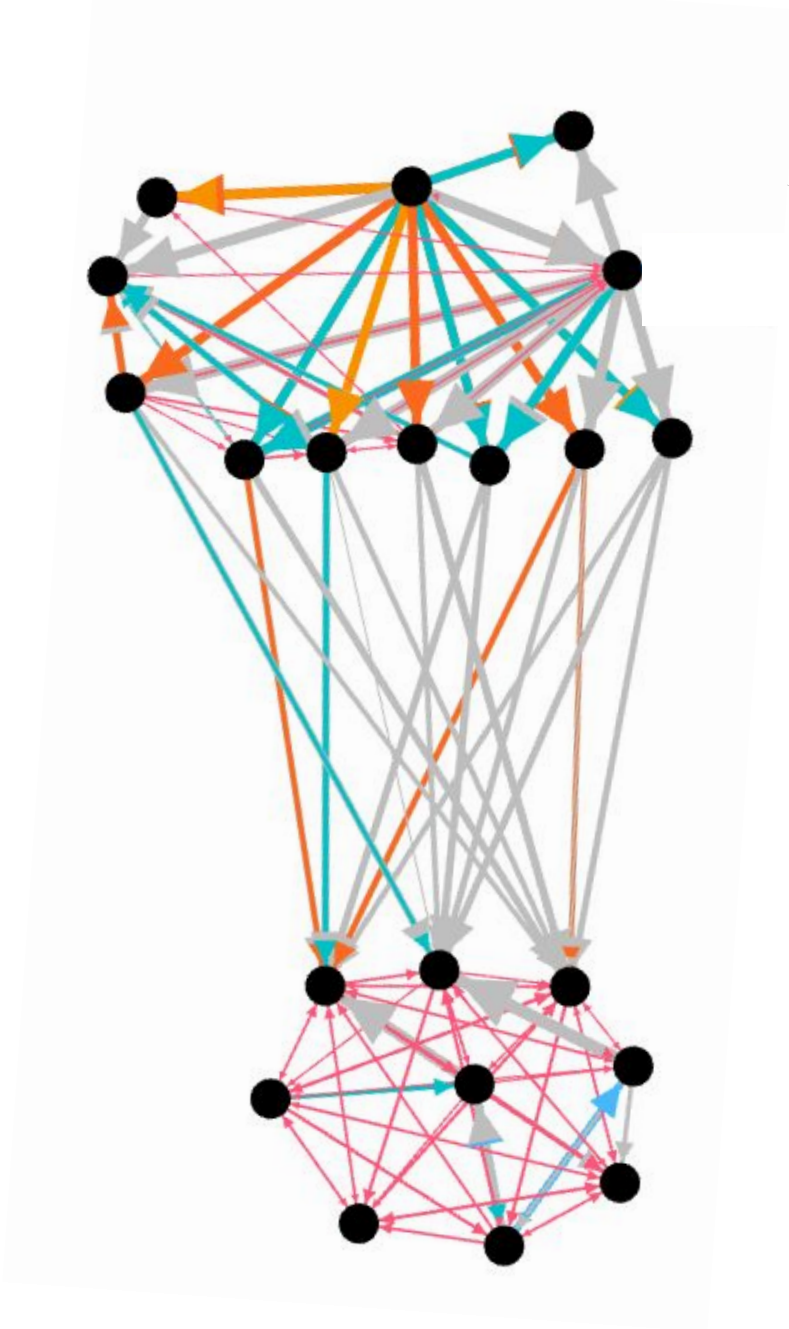


Figure 2: *Development of the numbers of physical servers and virtual servers in data centers in Germany (Source: calculations by Borderstep on the basis of Techconsult eanalyzer)*

Introducing the NetSlicer Algorithm

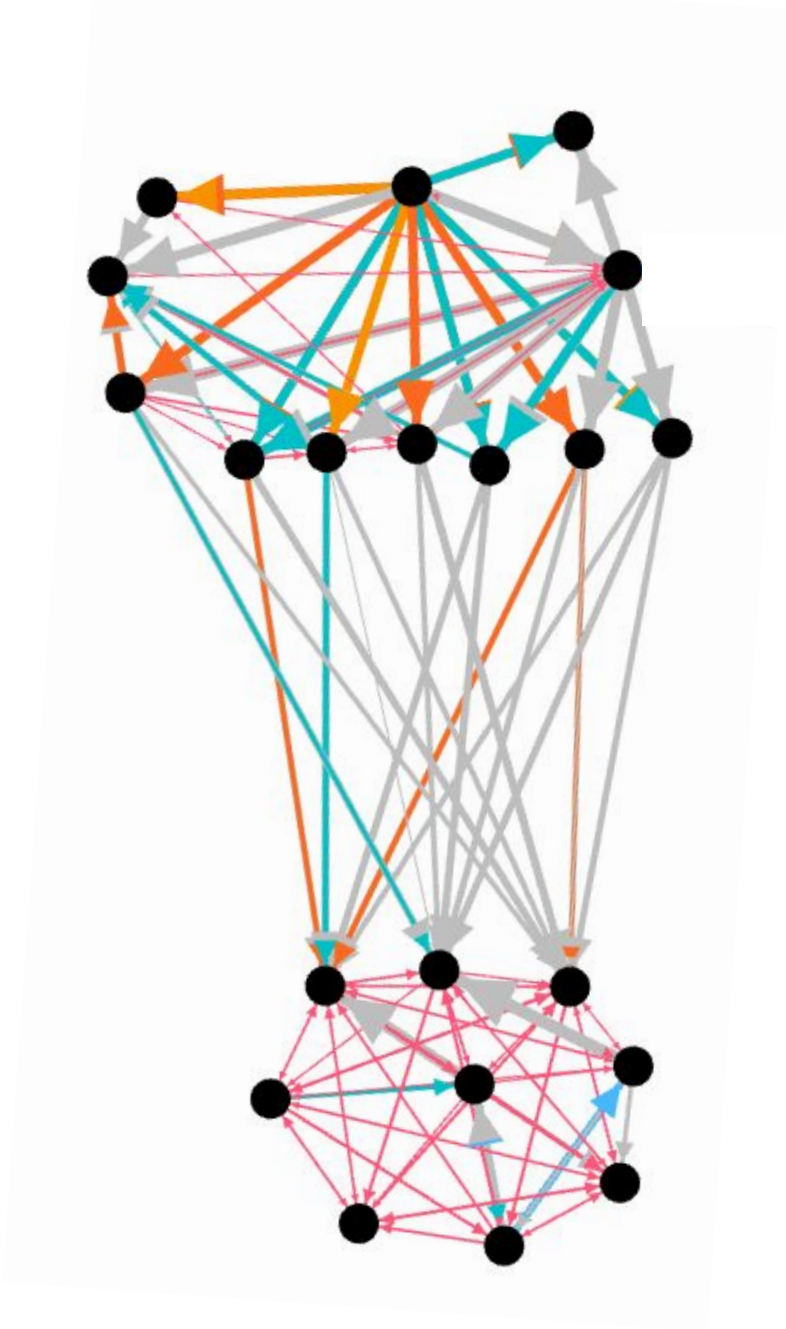
Introducing the NetSlicer Algorithm

- Start with an annotated graph
  - each link represents connections at a single port



Introducing the NetSlicer Algorithm

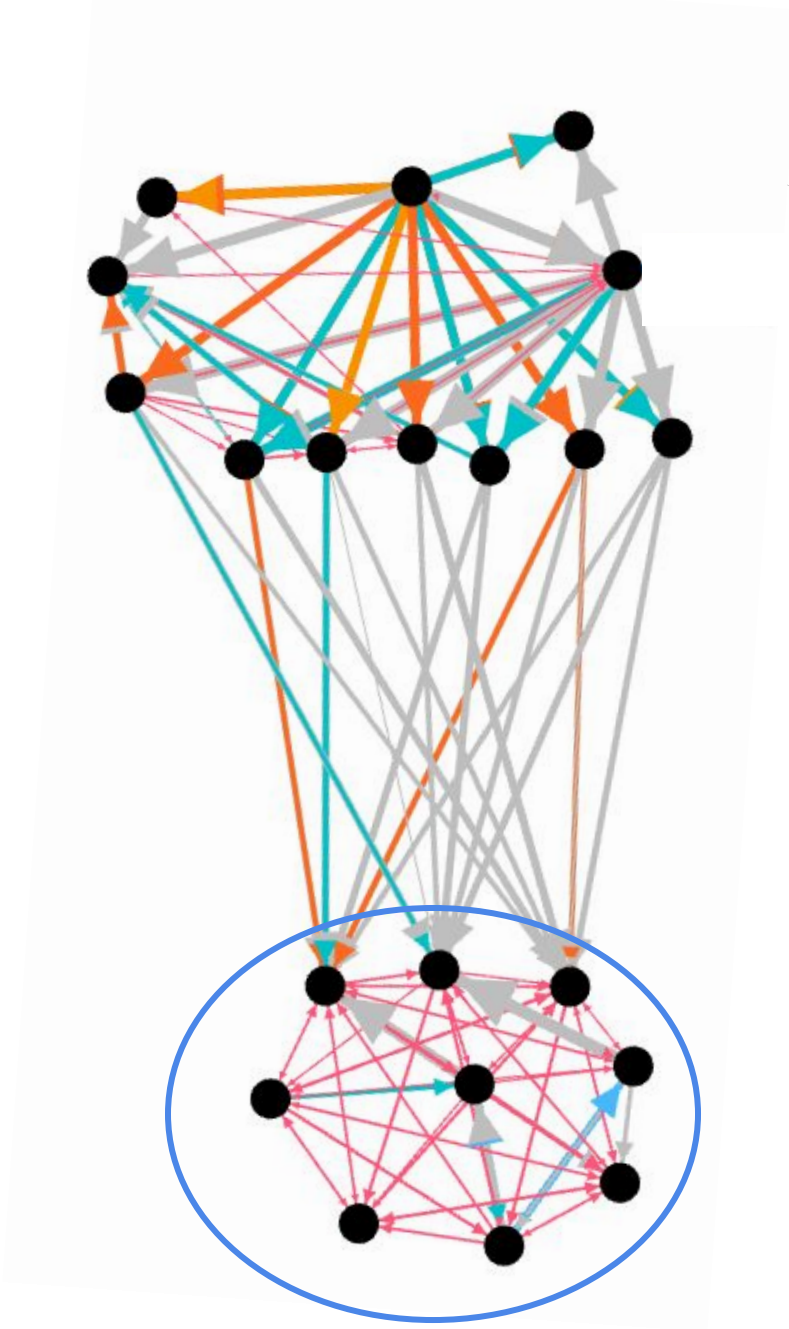
Merge nodes at 3 phases:



Introducing the NetSlicer Algorithm

Merge nodes at 3 phases:

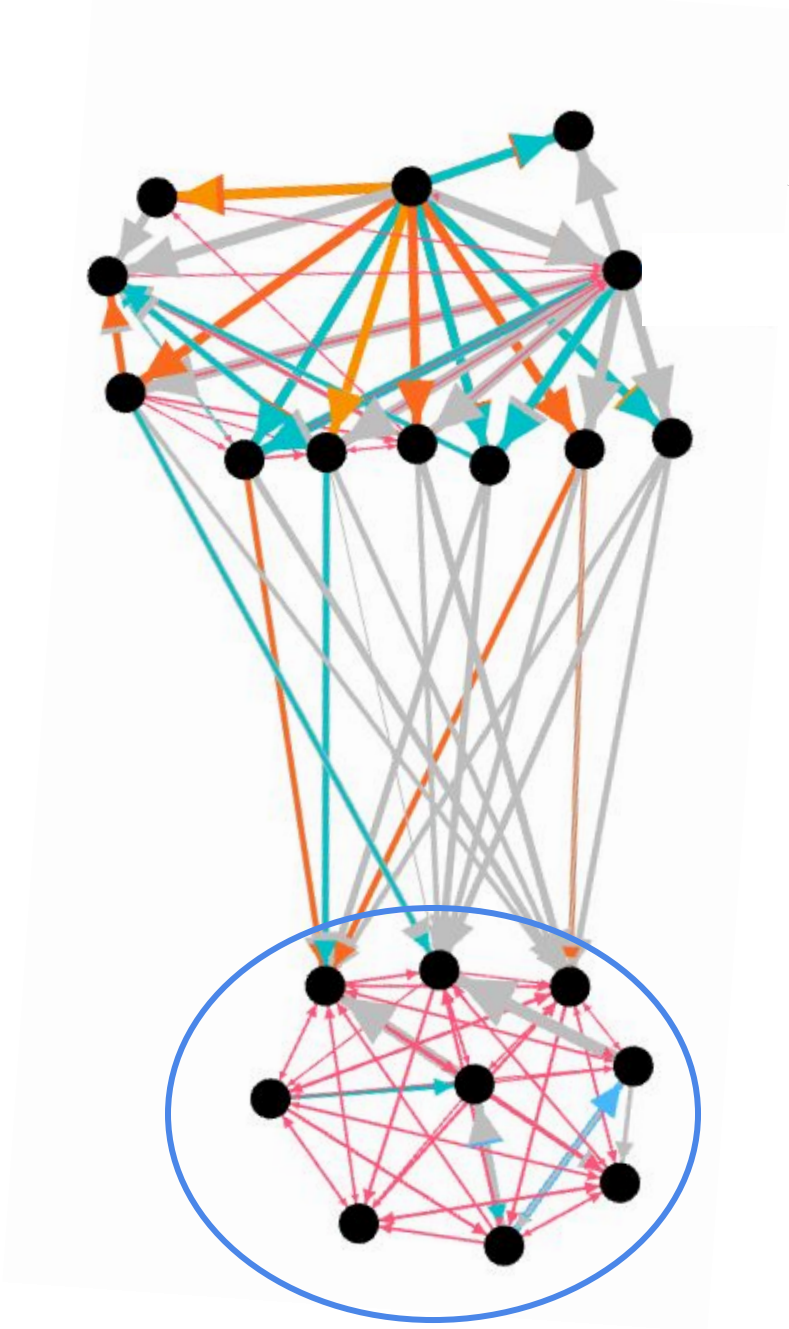
- Detect “clusters”
  - Strongly connected components



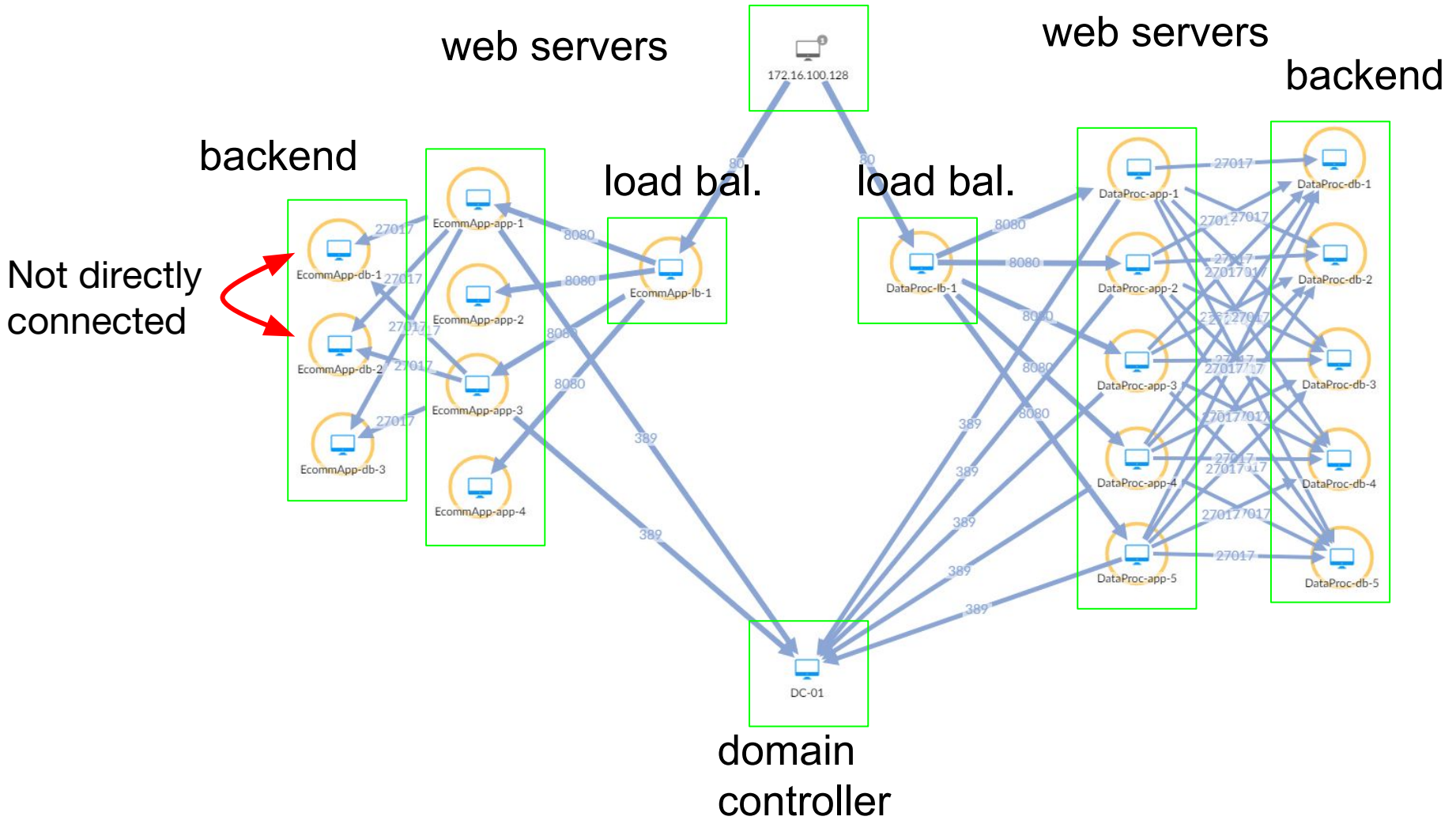
Introducing the NetSlicer Algorithm

Merge nodes at 3 phases:

- Detect “clusters”
  - Strongly connected components
- Detect tiers
  - based on neighborhood similarity
- Detect applications
  - based on weighted connectivity

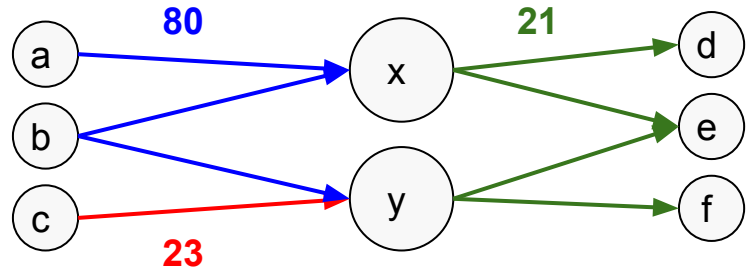


Detecting Tiers



A customized similarity measure

- Endpoint definition *(u,p,t)*∈**Φ**:
  - *u*: a node id
  - *p*: a service port number
  - *t*: type (ⓒ:client / ⓢ:server)

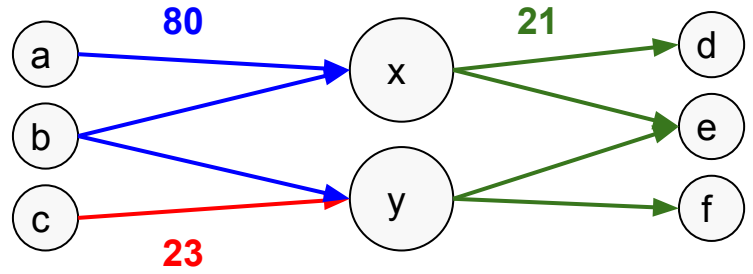


node id	connected endpoints
x	(a,80,ⓒ), (b,80,ⓒ), (d,21,ⓢ), (e,21,ⓢ)
y	(c,23,ⓒ), (b,80,ⓒ), (f,21,ⓢ), (e,21,ⓢ)
b	(x,80,ⓢ), (y,80,ⓢ)

A customized similarity measure

- Endpoint definition $(u,p,t) \in \Phi$:
  - u: a node id
  - p: a service port number
  - t: type (ⓒ:client / ⓢ:server)

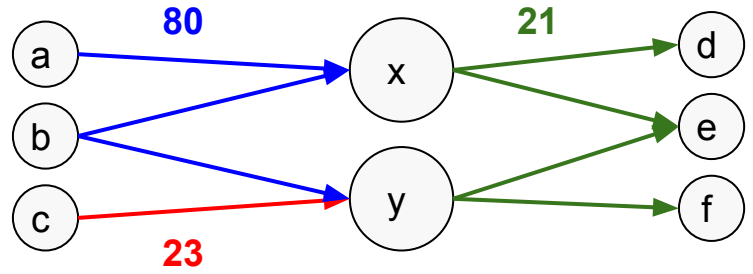
- Weight function $W_v: \Phi \rightarrow R+$
  - set weights to endpoint with respect to v
  - E.g., $W_v(\phi)=1$ if ϕ connected with v, o.w. 0



node id	connected endpoints
x	(a,80,ⓒ), (b,80,ⓒ), (d,21,ⓢ), (e,21,ⓢ)
y	(c,23,ⓒ), (b,80,ⓒ), (f,21,ⓢ), (e,21,ⓢ)
b	(x,80,ⓢ), (y,80,ⓢ)

A customized similarity measure

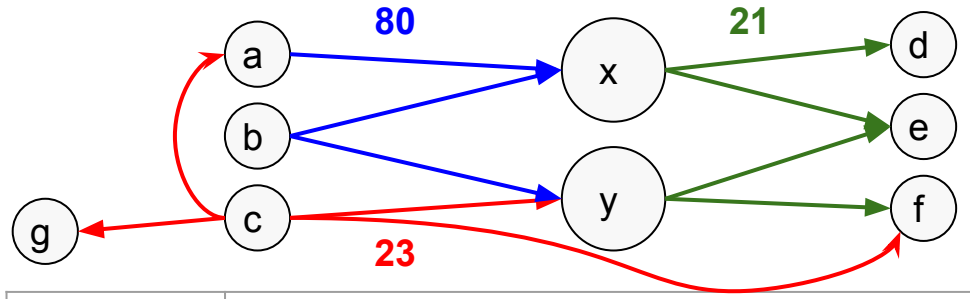
- Similarity by normalized dot product:
  - $Sim(u, v) = \sum_{\phi \in \Phi} W_u(\phi) W_v(\phi) / (|W_u| \cdot |W_v|)$,
  - where $|W_x| = \sqrt{\sum_{\phi \in \Phi} W_x(\phi)^2}$.



node id	connected endpoints
x	(a,80,Ⓒ), (b,80,Ⓒ), (d,21,Ⓢ), (e,21,Ⓢ)
y	(c,23,Ⓒ), (b,80,Ⓒ), (f,21,Ⓢ), (e,21,Ⓢ)
b	(x,80,Ⓢ), (y,80,Ⓢ)

A customized similarity measure

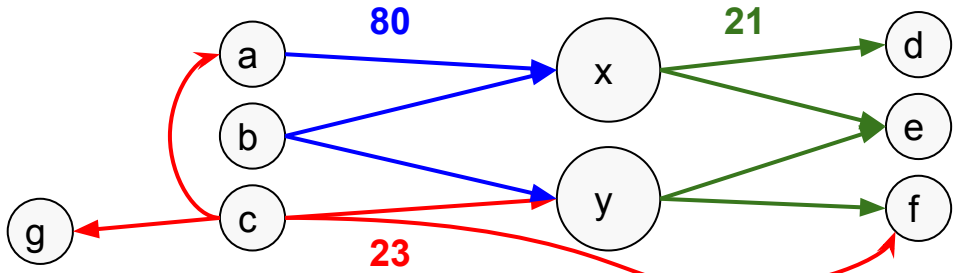
- Similarity by normalized dot product:
  - $Sim(u, v) = \sum_{\phi \in \Phi} W_u(\phi) W_v(\phi) / (|W_u| \cdot |W_v|),$
  - where $|W_x| = \sqrt{\sum_{\phi \in \Phi} W_x(\phi)^2}.$
- Consider noisiness of endpoints
  - $noise(\phi) = \max_{\substack{u,v \\ \text{connected to } \phi}} (1 - Sim(u,v))$
  - $W_v(\phi) = (1\text{-}noise(\phi))^{\alpha}$



node id	connected endpoints
x	(a,80,Ⓒ), (b,80,Ⓒ), (d,21,Ⓢ), (e,21,Ⓢ)
y	(c,23,Ⓒ), (b,80,Ⓒ), (f,21,Ⓢ), (e,21,Ⓢ)
b	(x,80,Ⓢ), (y,80,Ⓢ)

A customized similarity measure

- Similarity by normalized dot product:
  - $Sim(u, v) = \sum_{\phi \in \Phi} W_u(\phi) W_v(\phi) / (|W_u| \cdot |W_v|)$,
  - where $|W_x| = \sqrt{\sum_{\phi \in \Phi} W_x(\phi)^2}$.
- Consider noisiness of endpoints
  - $noise(\phi) = \max_{\substack{u,v \\ \text{connected to } \phi}} (1 - Sim(u,v))$
  - $W_v(\phi) = (1\text{-}noise(\phi))^{\alpha}$
- Similarity and noise are computed in several rounds

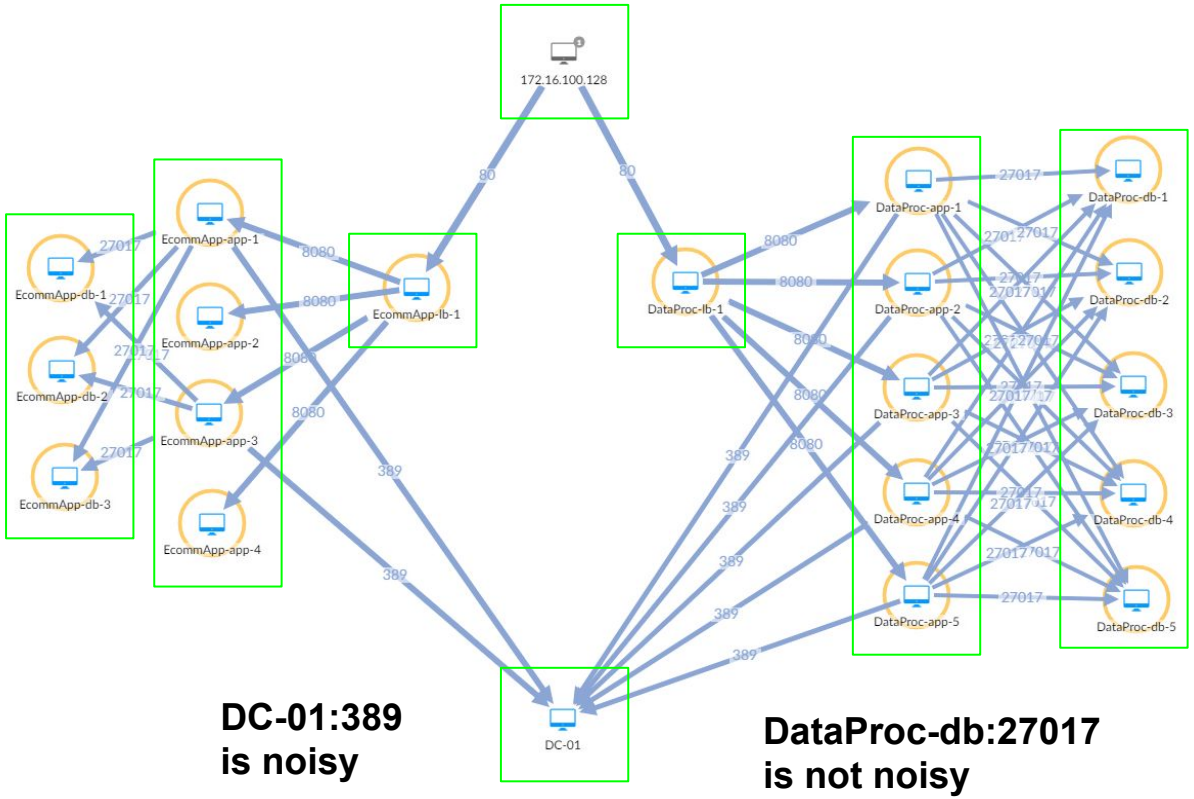


node id	connected endpoints
x	(a,80,Ⓒ), (b,80,Ⓒ), (d,21,Ⓢ), (e,21,Ⓢ)
y	(c,23,Ⓒ), (b,80,Ⓒ), (f,21,Ⓢ), (e,21,Ⓢ)
b	(x,80,Ⓢ), (y,80,Ⓢ)

Detecting Tiers

An iterative process

- computes similarity and noise
- merges similar nodes
- stops when no similar pair found

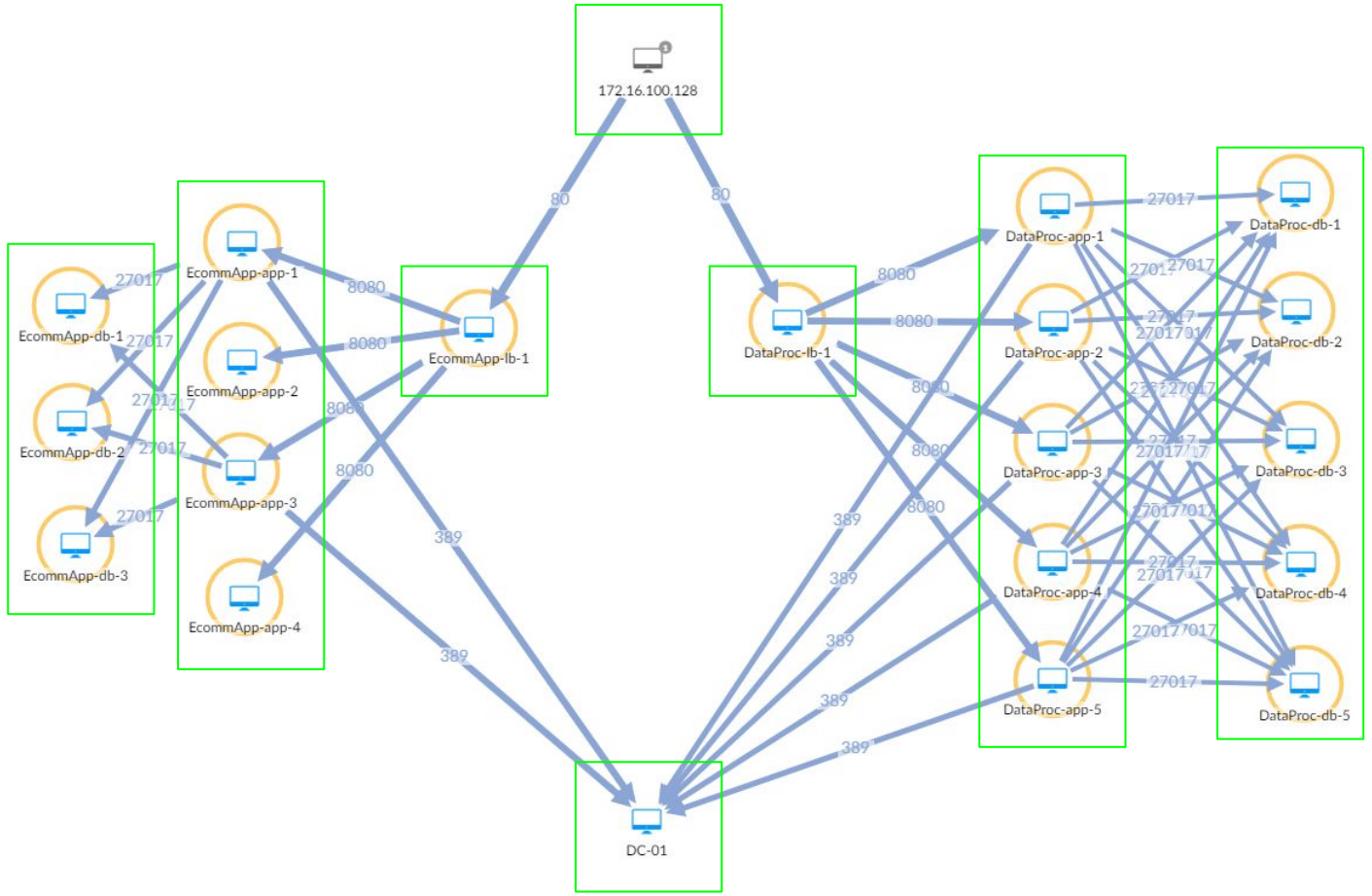


DC-01:389 is noisy

DataProc-db:27017 is not noisy

Detecting Applications

- From tiers to applications

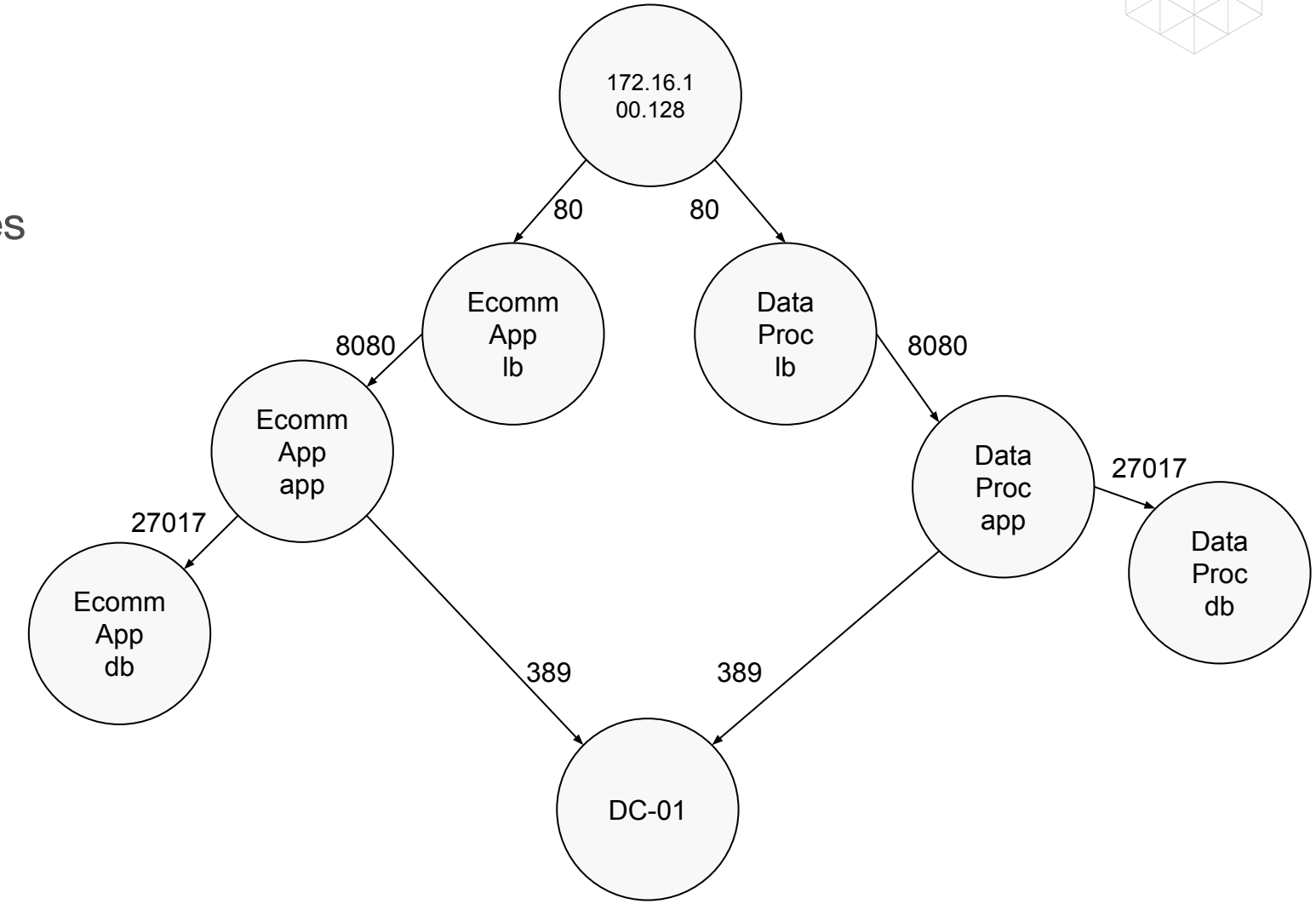


GuardiCore

Big-DAMA 2018

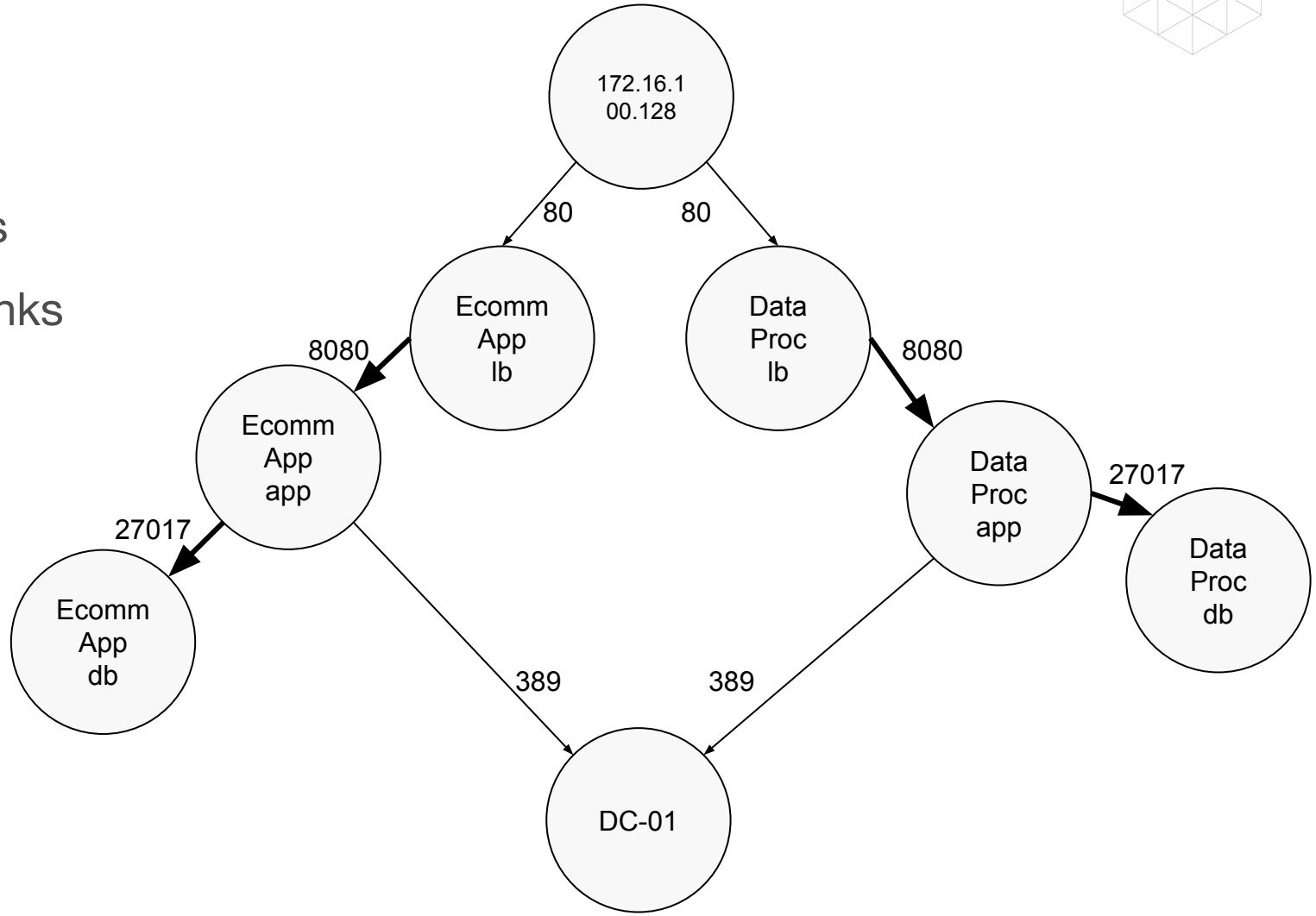
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes



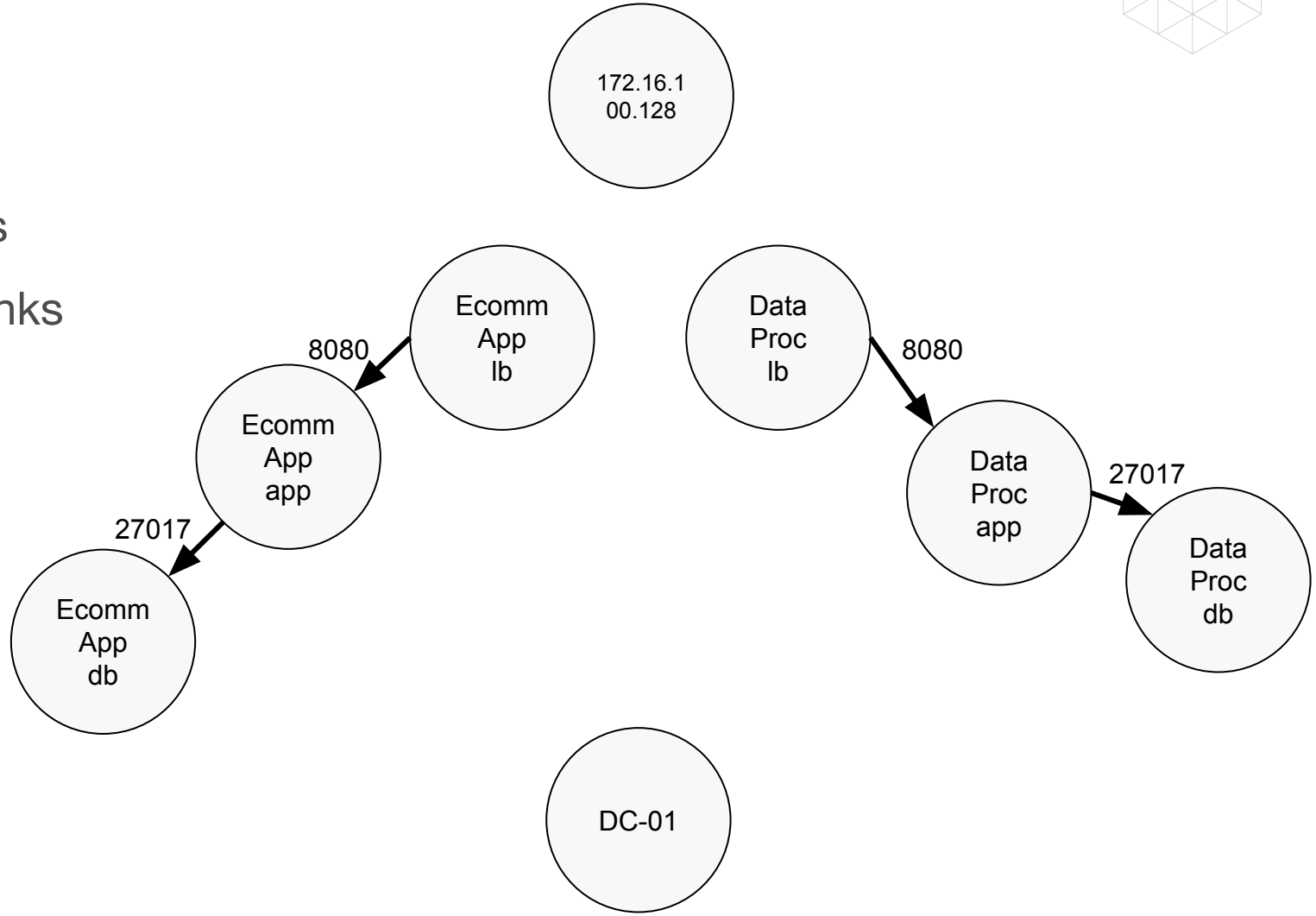
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links



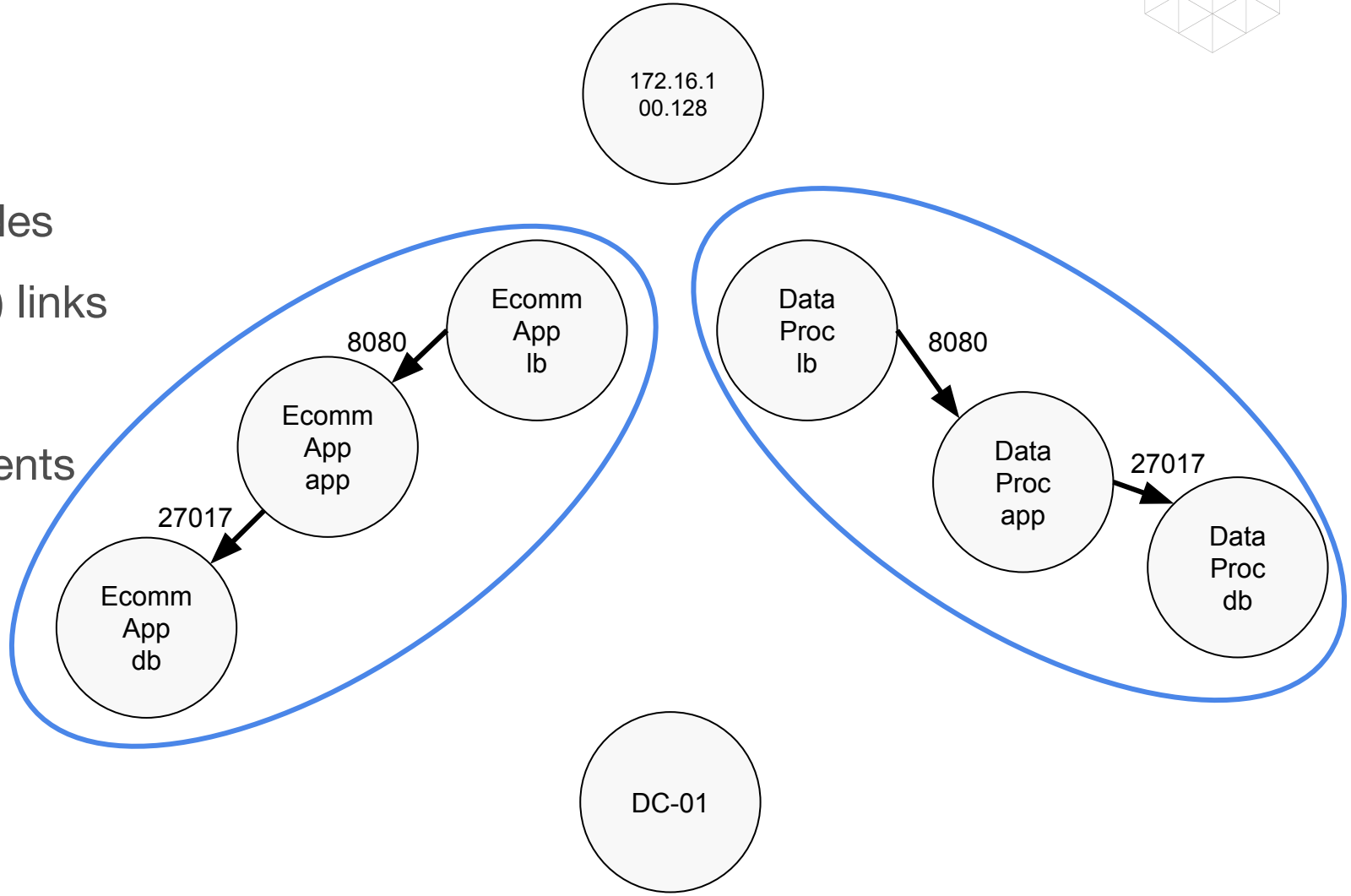
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links
  - Ignore weak/infra links



Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links
  - Ignore weak/infra links
  - Group connected components



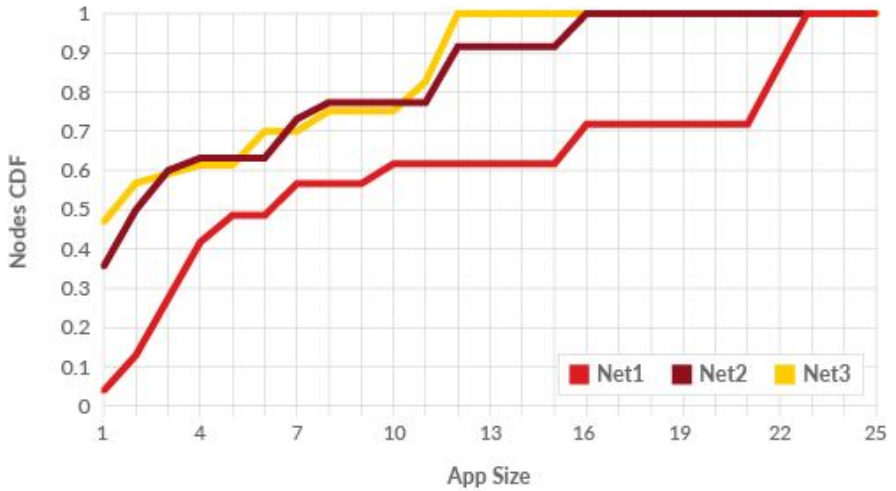
Implementation

- Programed in Python
- Parallel design
- Optional initial noise configuration
  - for better and faster results
  - e.g., super noisy ports and nodes
- Modular
  - any clustering alg. for tiers and apps (given similarity values)

Evaluation

- Datasets:
  - 3 datacenter networks
  - monitored at core VMs
  - ground truth for monitored
  - available online

	net1	net2	net3
dominat OS			
#nodes	**3003**	**2507**	**4588**
#monitored	153	143	174
#unmonitored	2850	2364	4414
ground-truth apps	**40**	**58**	**85**

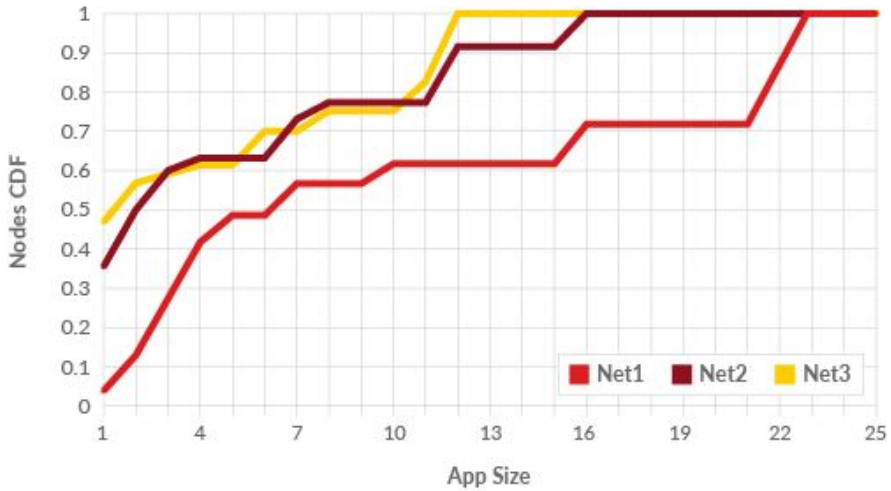


Distribution of nodes to app sizes

Evaluation

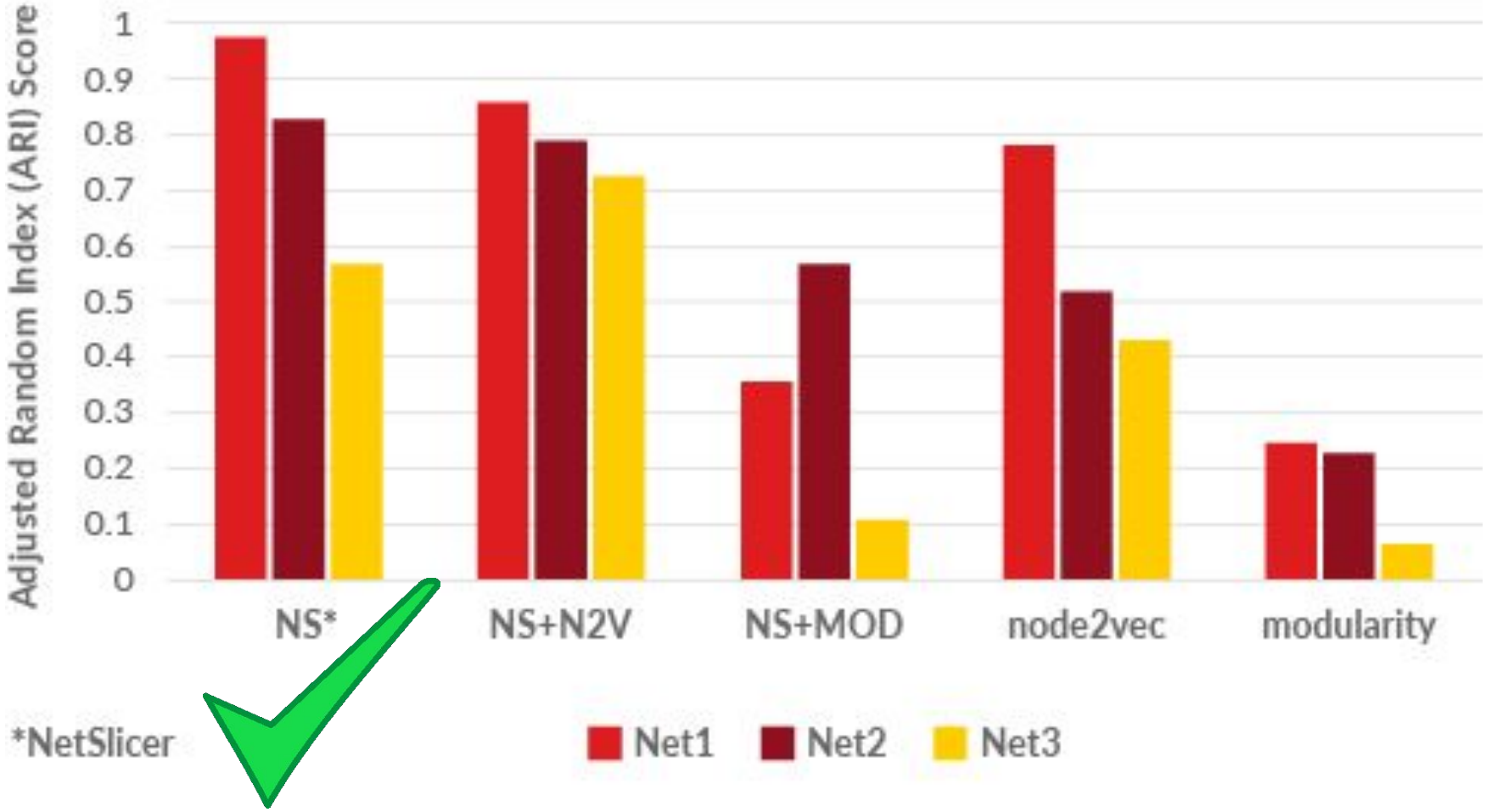
- Datasets:
  - 3 datacenter networks
  - monitored at core VMs
  - ground truth for monitored
  - available online
- Compared with:
  - louvain modularity
  - node2vec (+ HAC/HDBSCAN)
  - and mixtures
- Scoring:
  - Adjusted Random Index (ARI) from ground truth

	net1	net2	net3
dominat OS			
#nodes	**3003**	**2507**	**4588**
#monitored	153	143	174
#unmonitored	2850	2364	4414
ground-truth apps	**40**	**58**	**85**



Distribution of nodes to app sizes

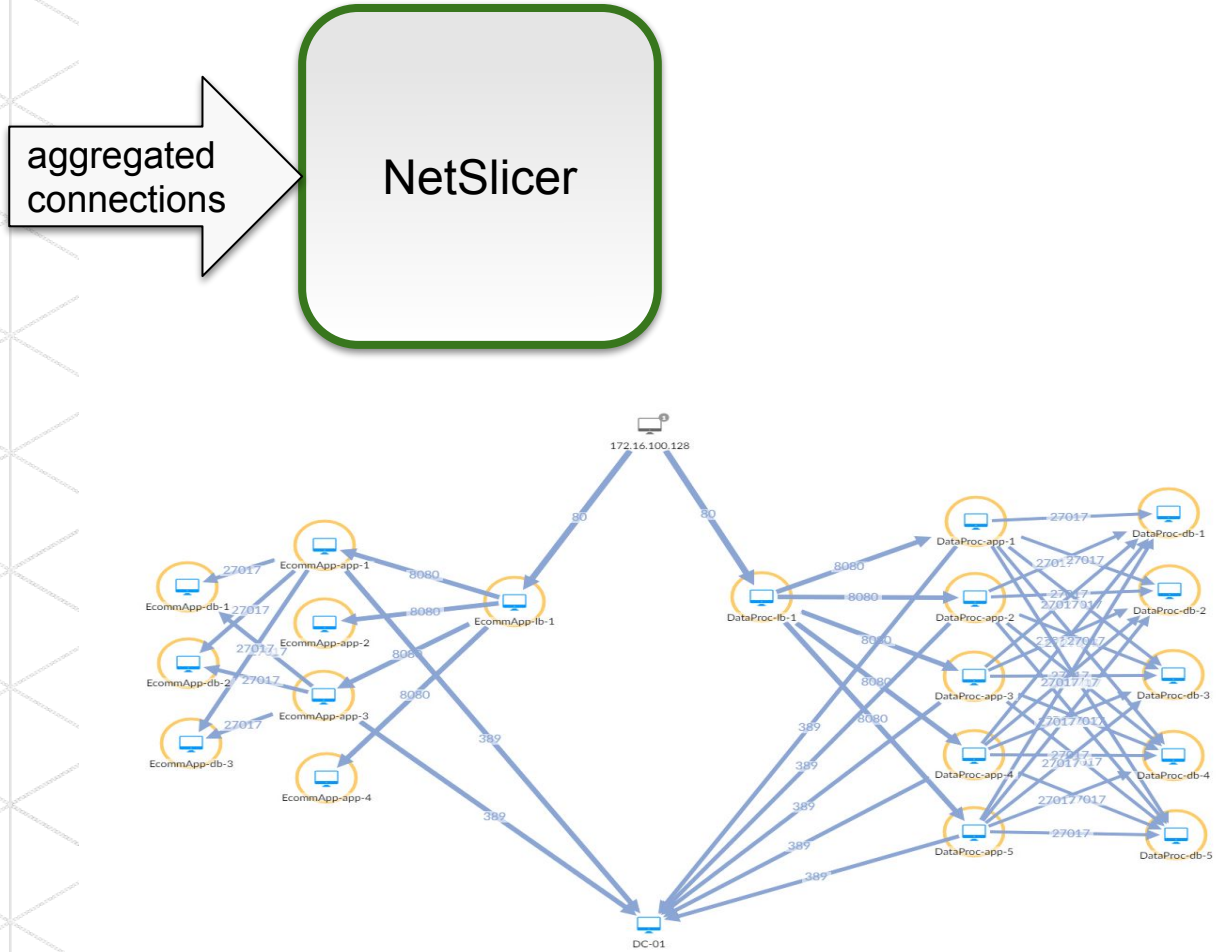
Results



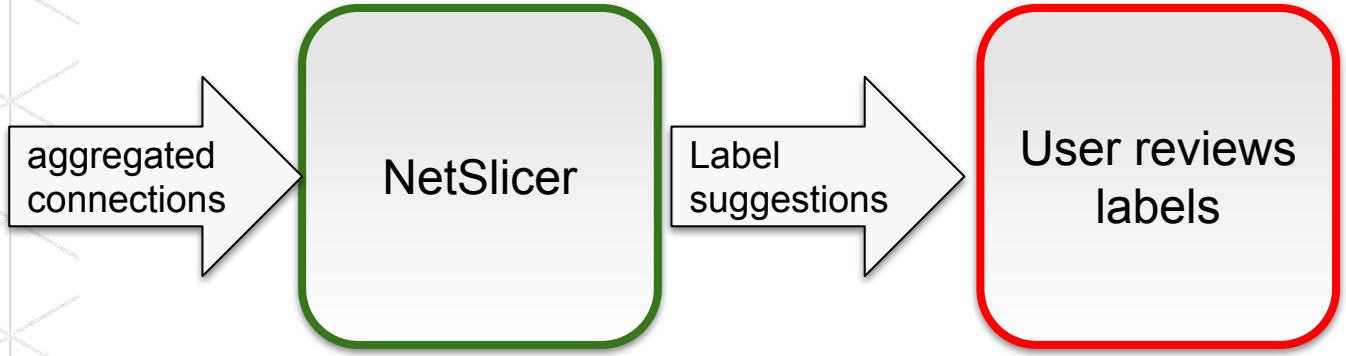
Analyst Compatible

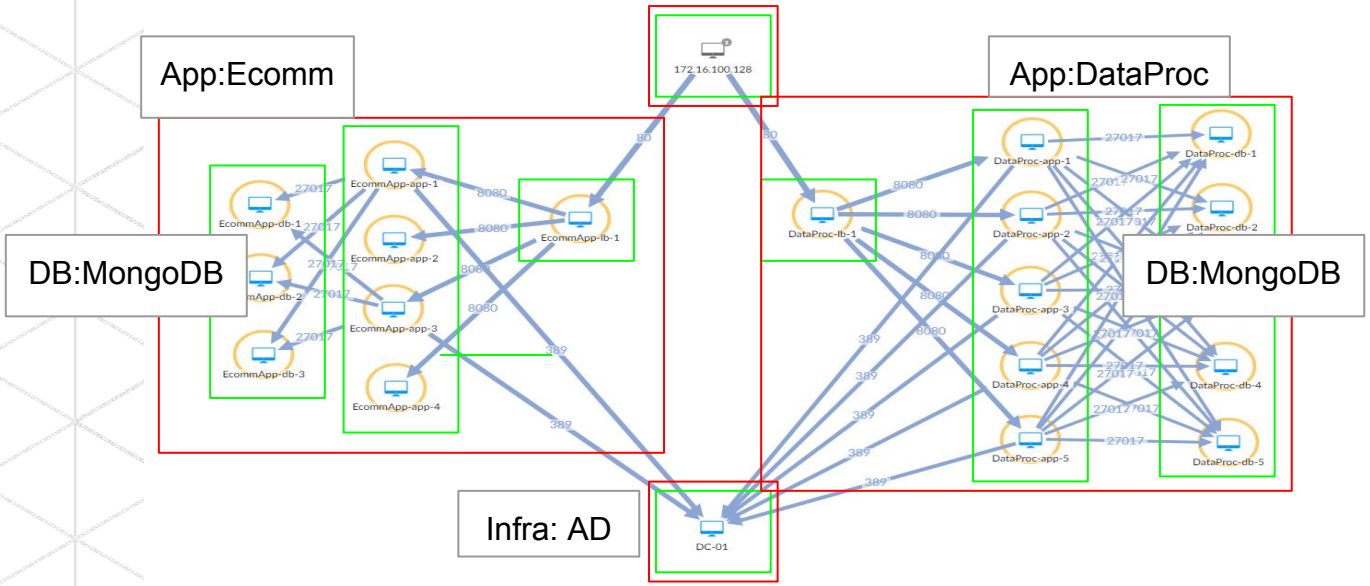
Use case: Microsegmentation Workflow

Use case: Microsegmentation Workflow

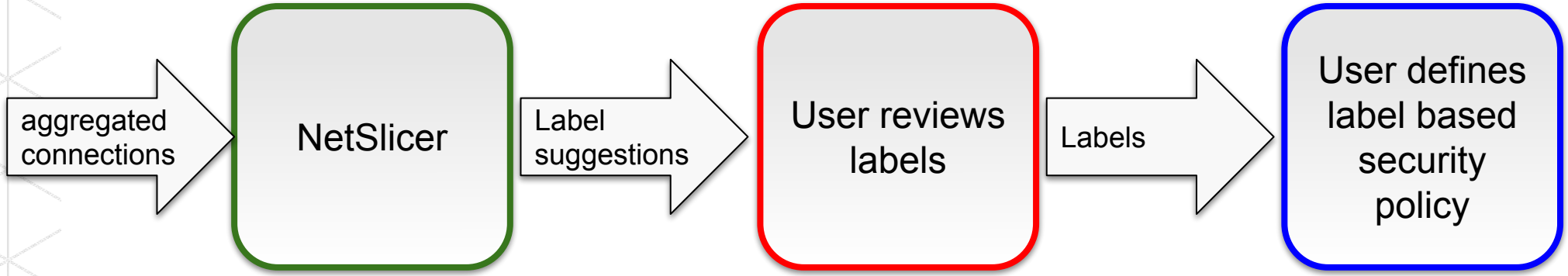


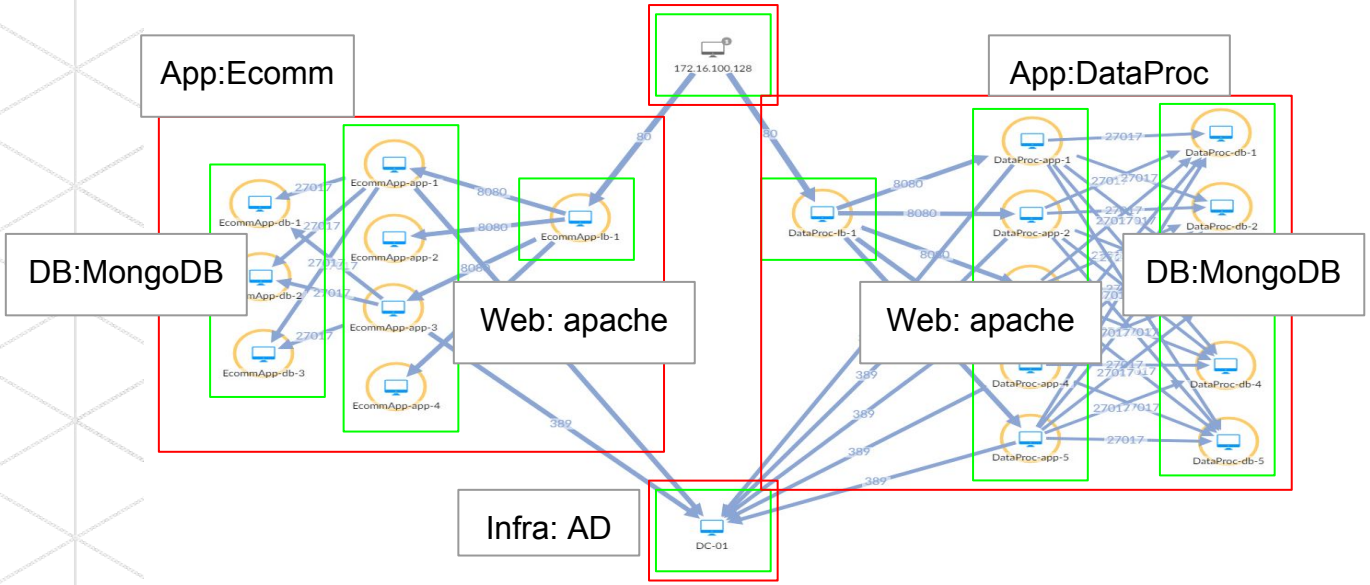
Use case: Microsegmentation Workflow





Use case: Microsegmentation Workflow

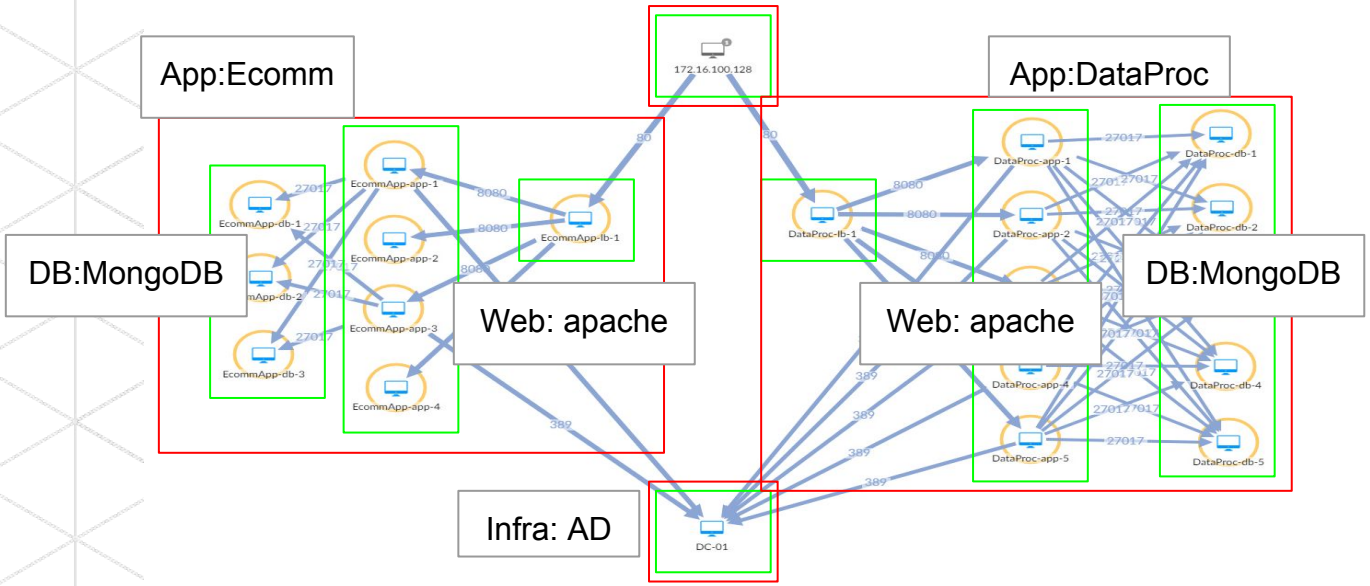




GuardiCore

Big-DAMA 2018

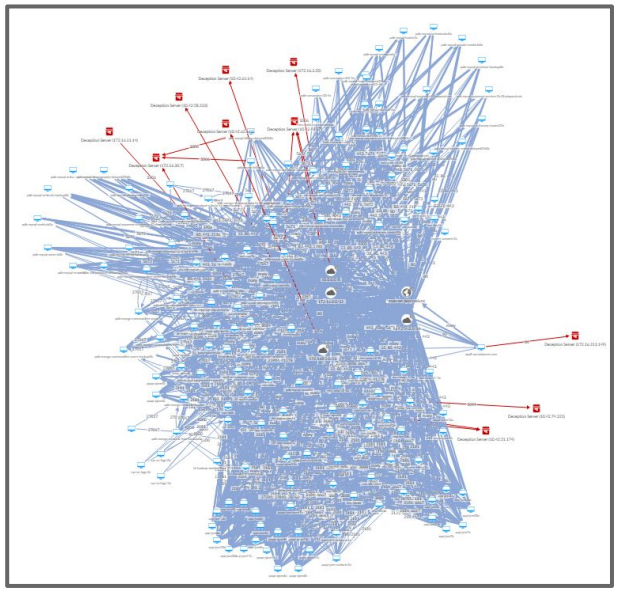
Use case: Microsegmentation Workflow

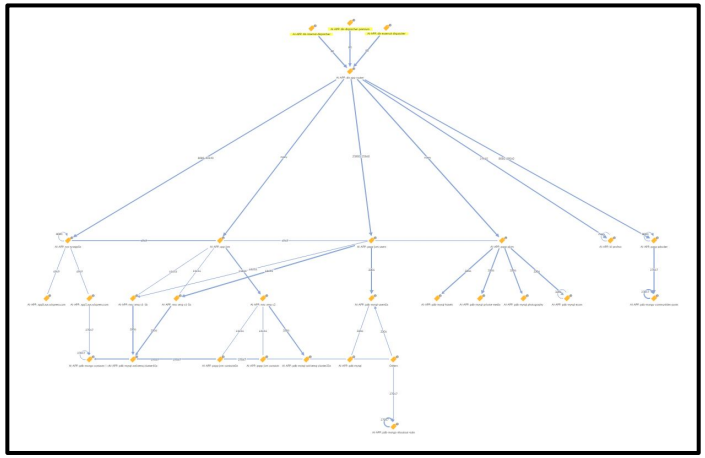


src	dst	action
Ecomm.apache	Ecomm.Mongo:27018	allow
Ecomm	AD:389	allow
Users	DataProc:80	allow
...	...	...

The NetSlicer Algorithm

- Customized similarity
- Considers real life scenarios
- Promising initial results
- Parallel and modular design



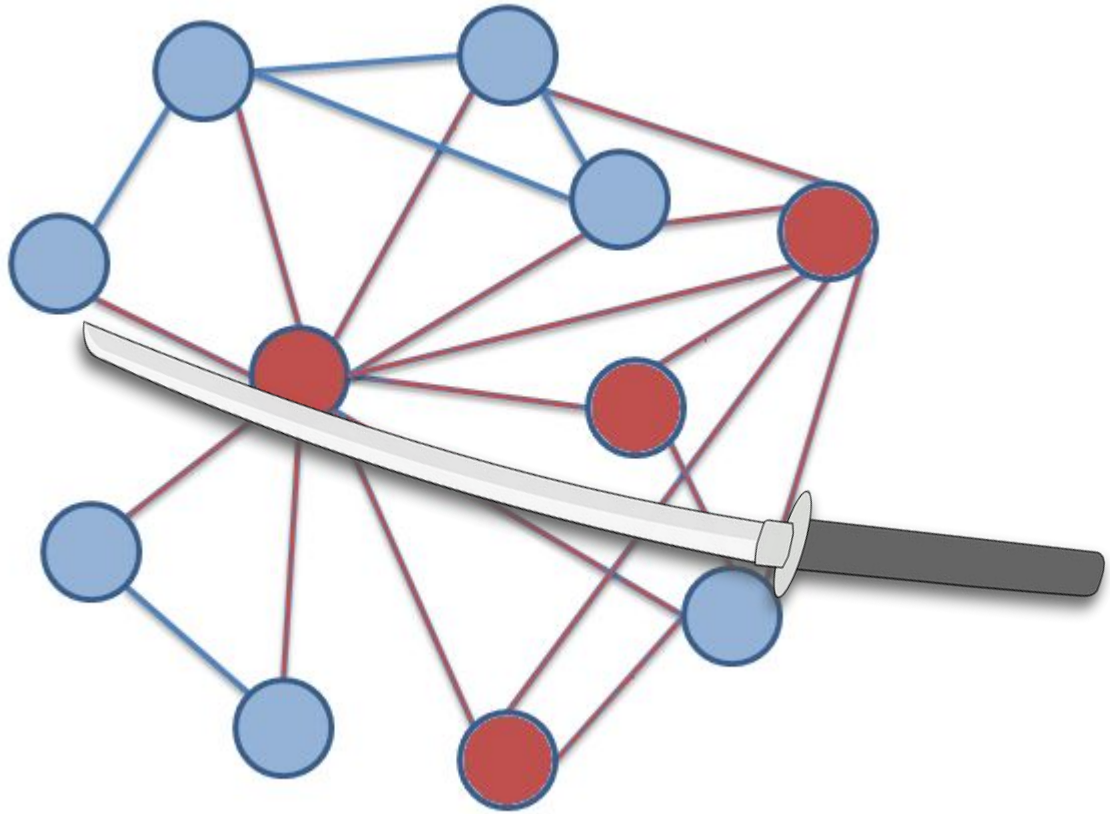


Future work

- More datasets
- Clustering processes
  - (multiple apps per server)
- Mixing with more clustering algs.
- Convergence time bounds

GuardiCore

Questions?



datasets: https://www.guardicore.com/labs/datacenter-traces/
more works: https://www.guardicore.com/labs/research-academic/
me: liron.schiff@guardicore.com

Architecture