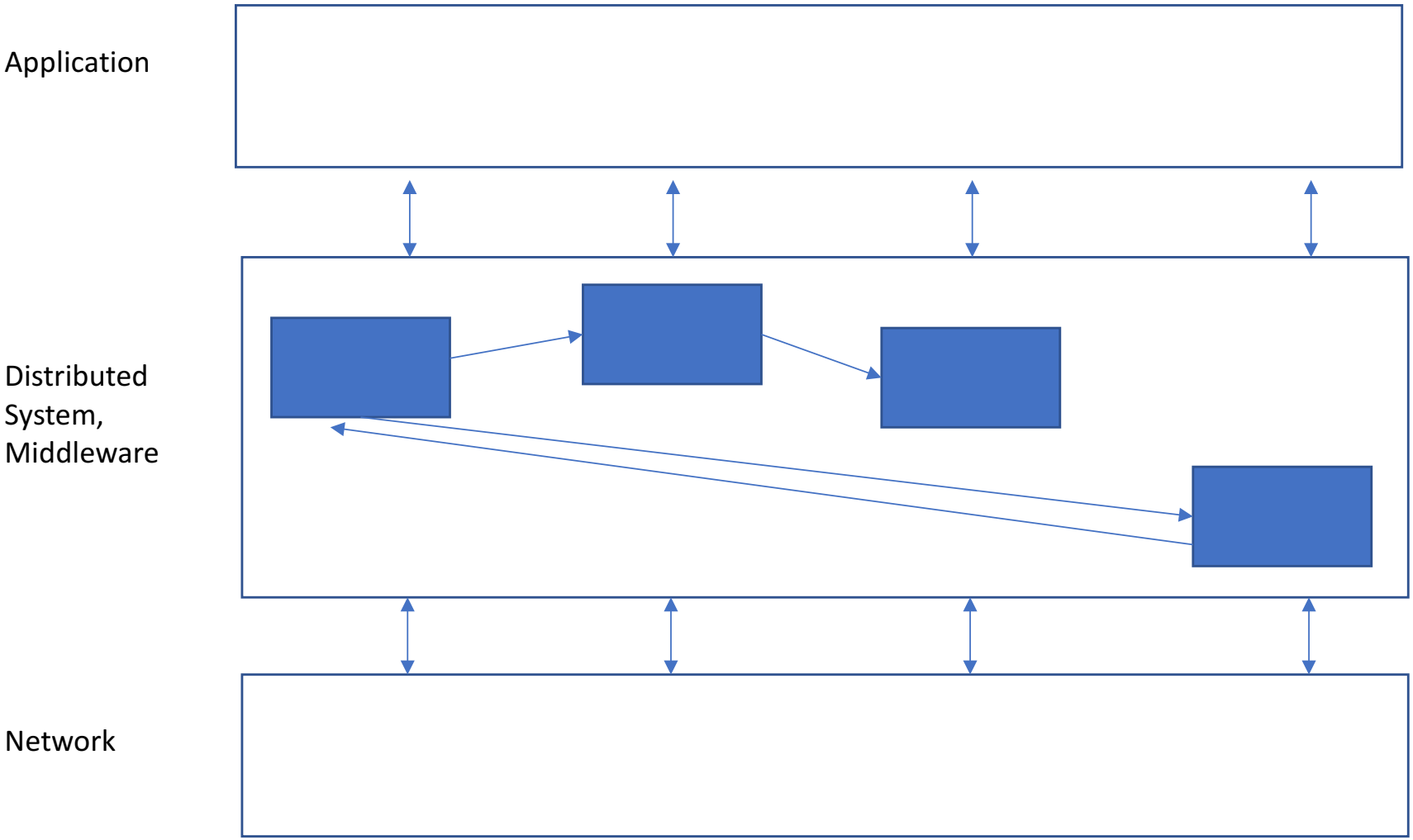
Bettina Kemme
School of Computer Science

Cesar Cañas, Joerg Kienzle, Suhail Kandanur, Kaiwen Zhang, Mona El Saadawy, Maximilian Schiedermeier, Shiquan Zhang

McGill University

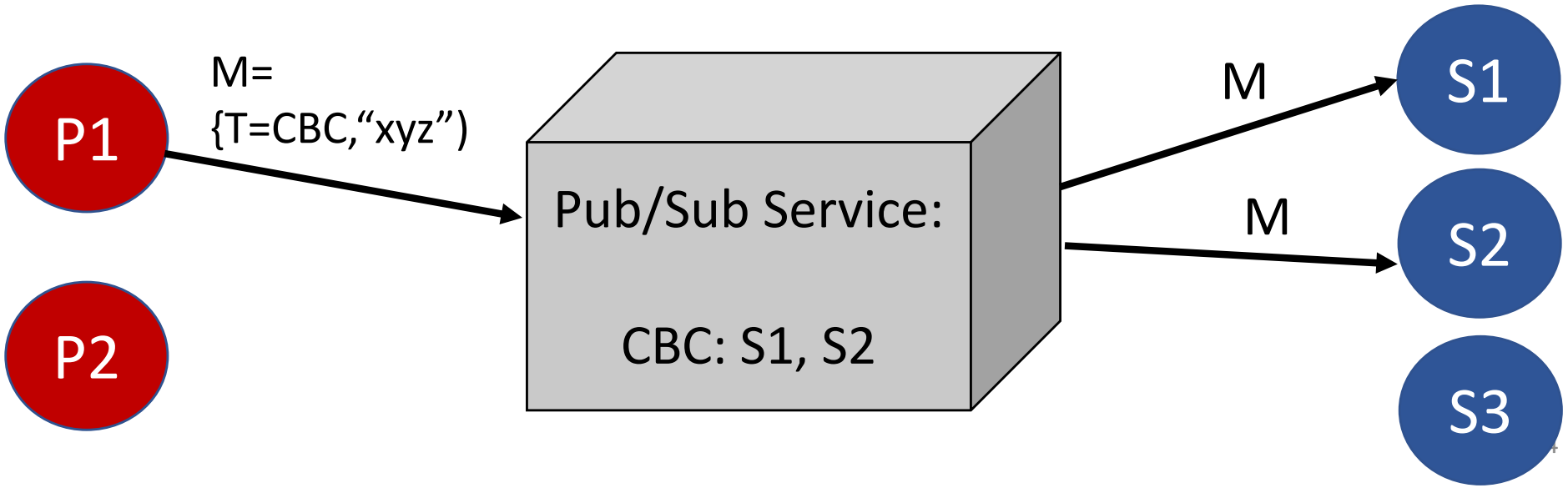
Upreach and Downreach



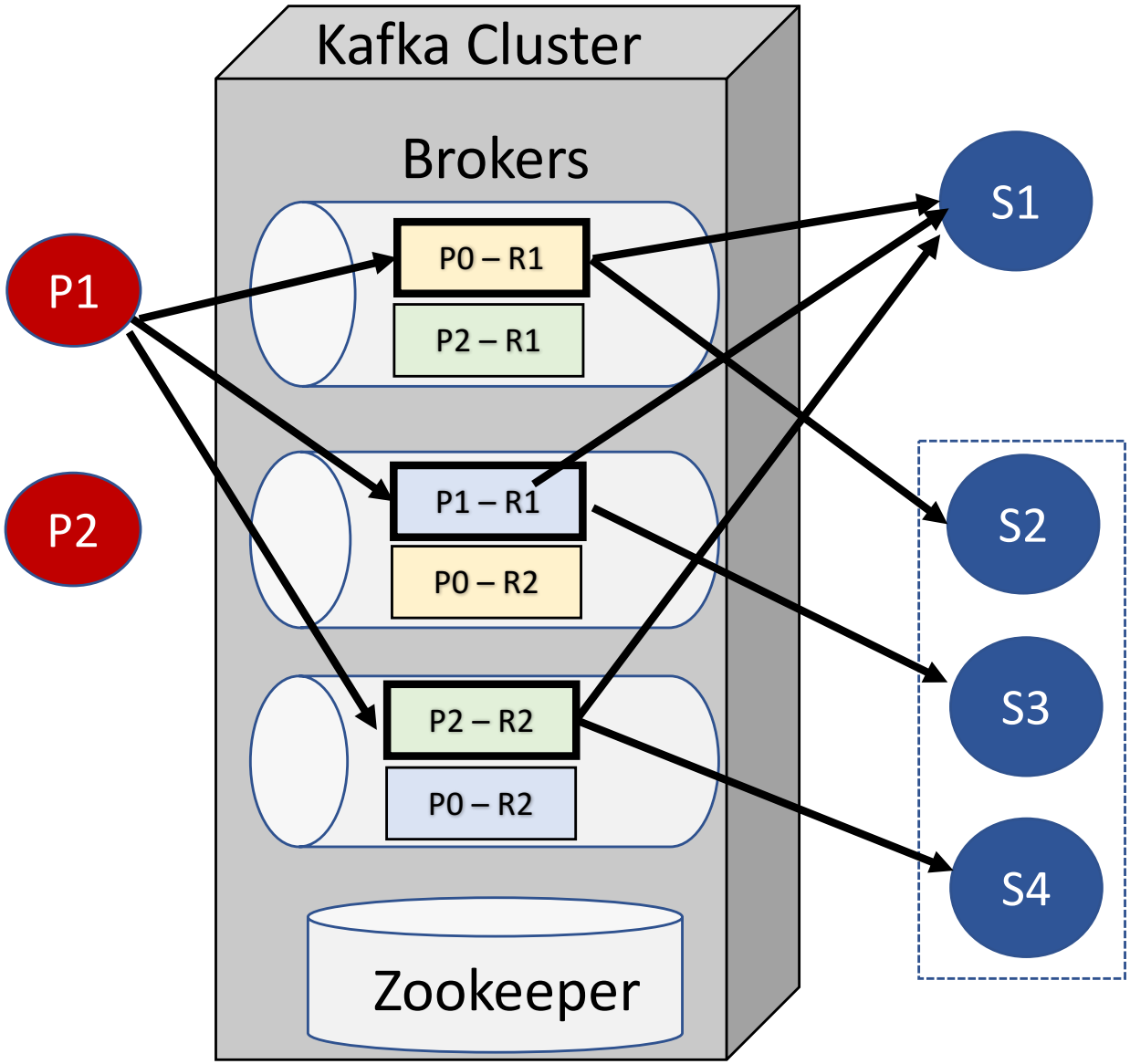
Graph - based pub / sub

Pub/Sub Systems

- Topic based
- Content based
  - Subscriptions are queries over publication content
    - Attributes/value and filters
    - XML and XQuery
    - RDF graphs and graph queries



System Research



- Scalability
- Availability & Fault Tolerance
- Low Latency
- Delivery guarantees

- Many techniques from research
  - Quorums, Zookeeper, commit-log, pull vs. push…..
- Many research prototypes
  - Pipelining, component-based, …

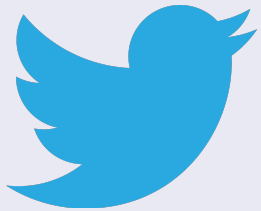
Applications

Traffic alert systems

Weather alert systems

Mobile notif. frameworks

Social networks



Chat/IM systems

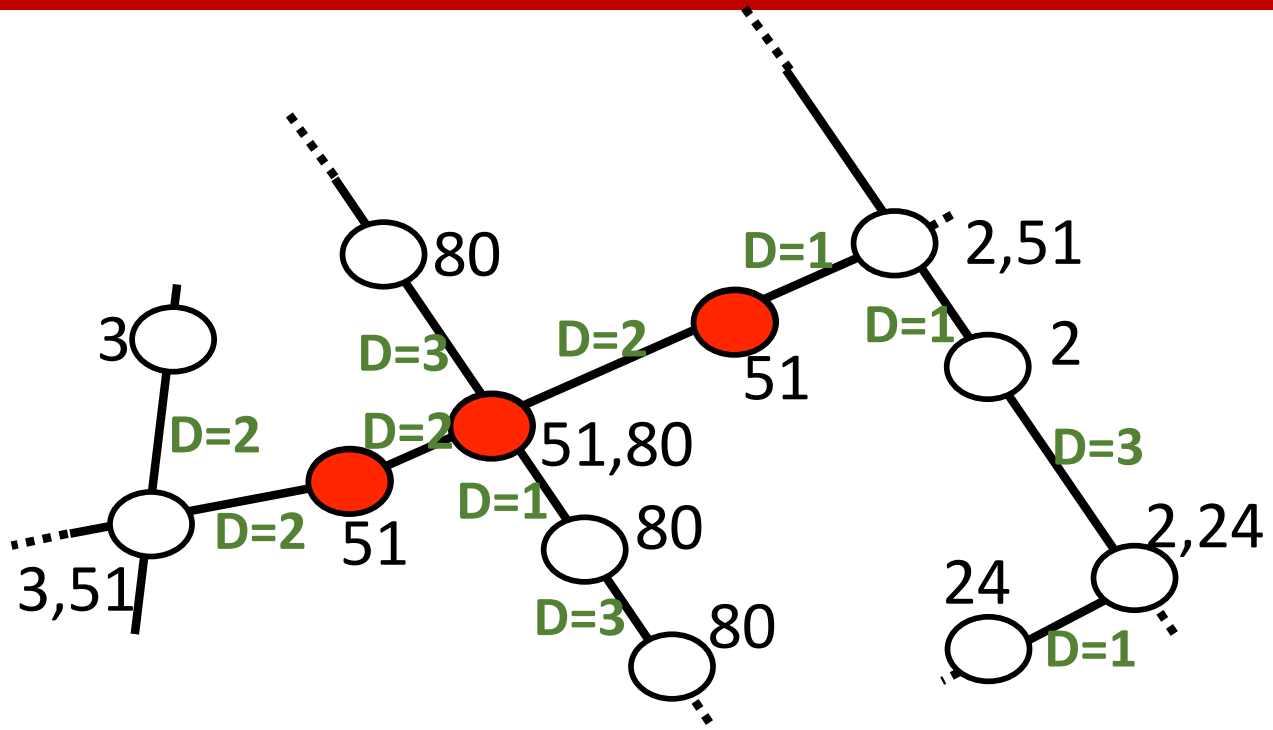
Multiplayer Games

Extending the core functionality of pub/sub

Understanding the application and its needs drive the development

1. GraPS
   - Merging Data Management with pub/sub
2. Evolving subscriptions
   - Enable fast-changing interest
3. CacheDOCS
   - Mixing caching with pub/sub

Application Graph: Transit Network



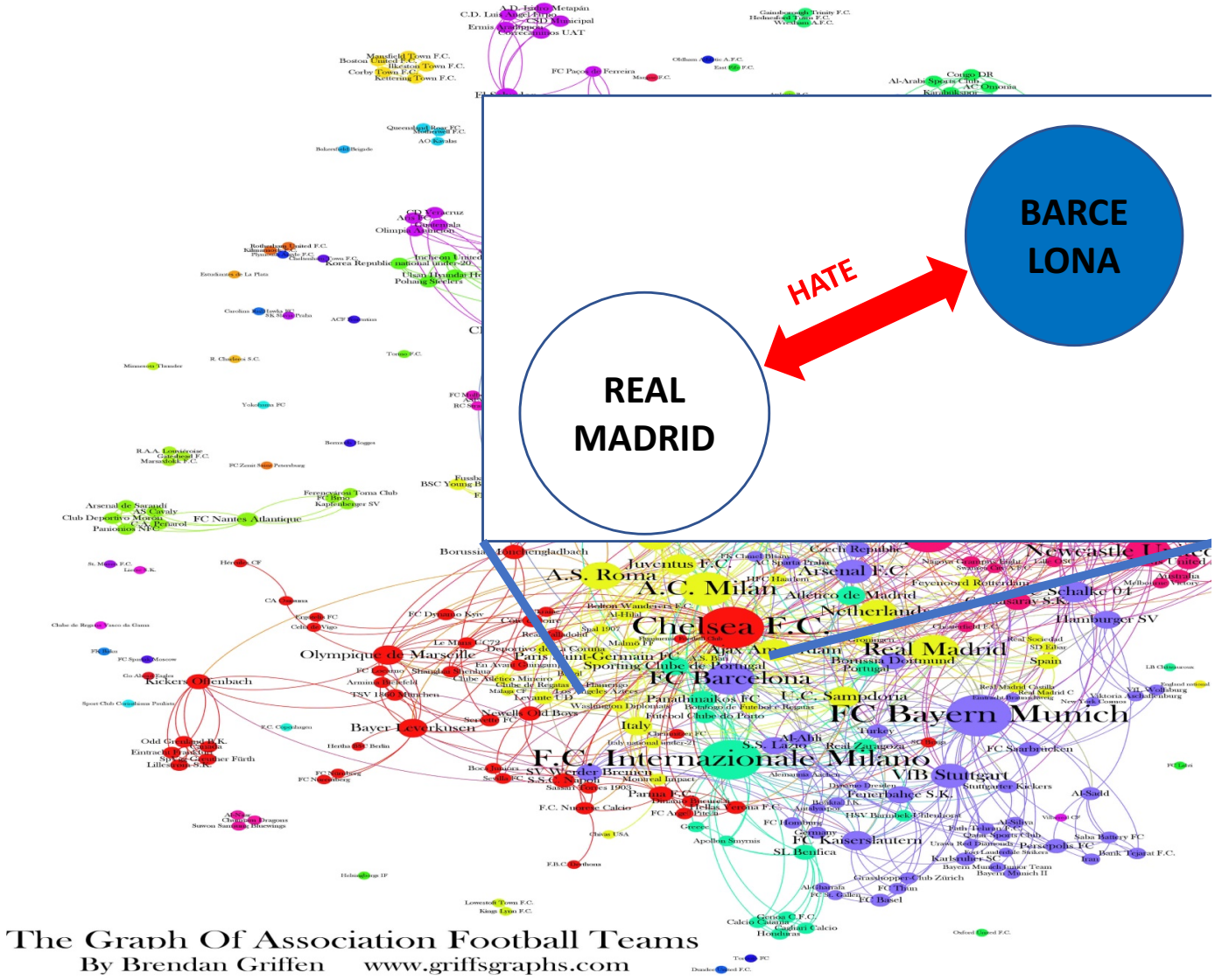
- Subscriptions:
  - The 3 stops before my stop
  - Max Distance of 6 minutes from my stop
- Publications: Whenever a bus arrives at a bus stop

Application Graphs: Street Maps

Traffic Monitoring

- Publications: Traffic Information on each segment
- Subscriptions: Path to be travelled

Knowledge Graphs: Soccer Teams

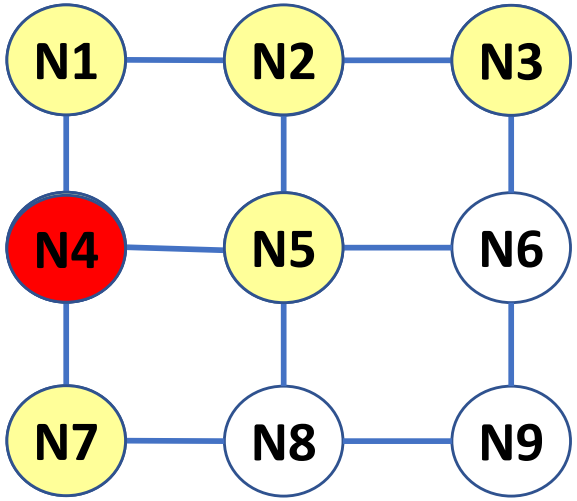


- Publication:
  - Info about a certain team
  - Published on a node
- Subscriptions:
  - all teams my team is related to
  - Graph query

Graph-based pub/sub

- The application domain is represented as a graph or multiple graphs that are stored as meta-information in our system.

Graph G1



- **Subscriptions:**
  - Expressed as graph-query
  - Returns a sub-graph

```
Subscribe (hopDistance (N1,  2));
```

- **Publications:**
  - On a node / edge
  - Graph-query returning sub-graph

```
Publish (N4, msg);
```

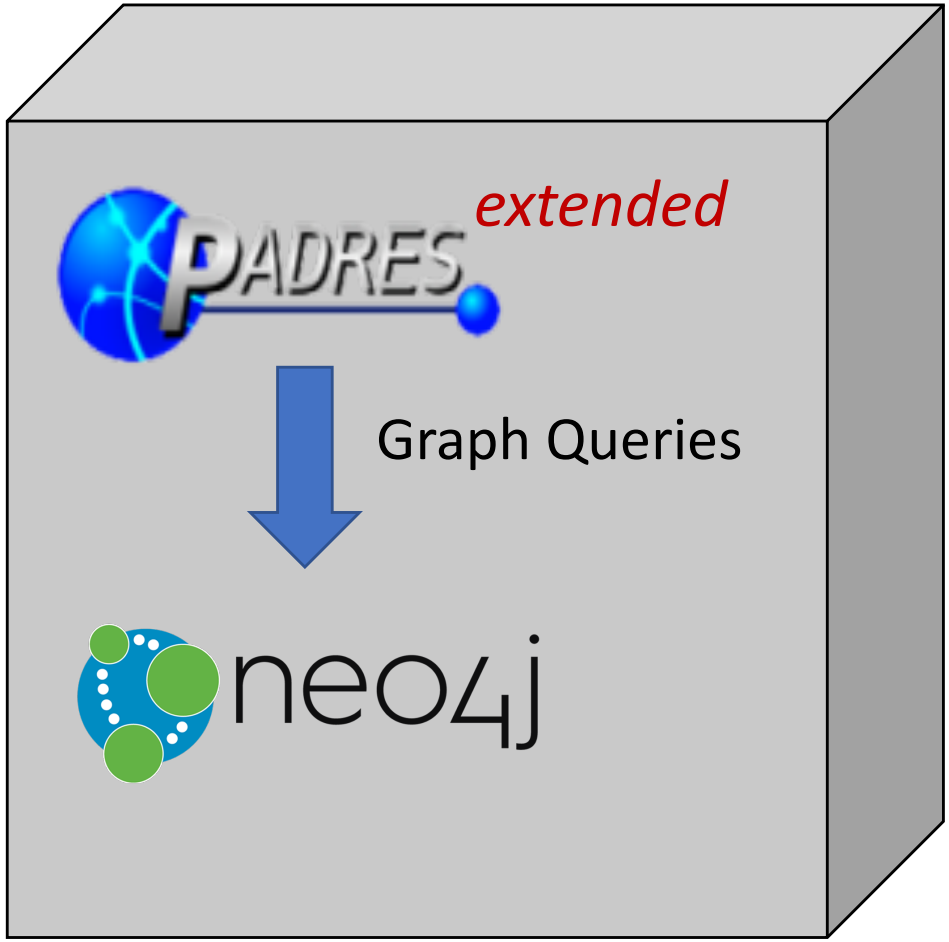
- **Match:**
  - Sub-graph overlap

11

Graps Implementation

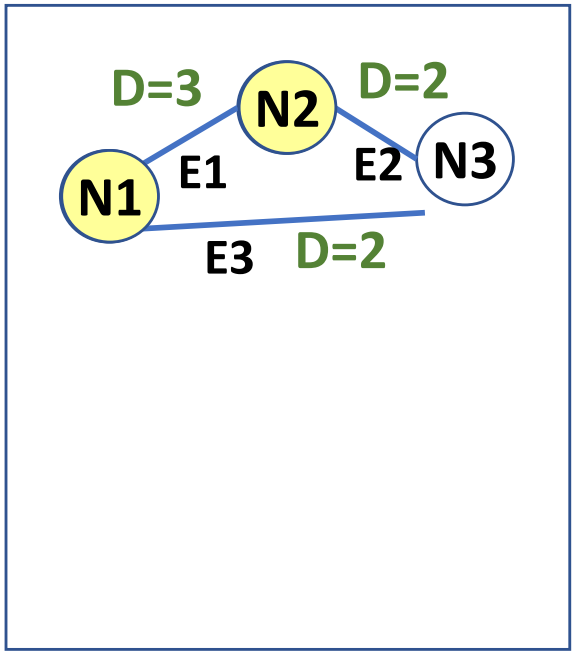
- Publish
- Subscribe
- unsubscribe

- Store/delete Graph
- Update Graph



- Pub/sub engine
- Graph DBS backend

Updating the Graph

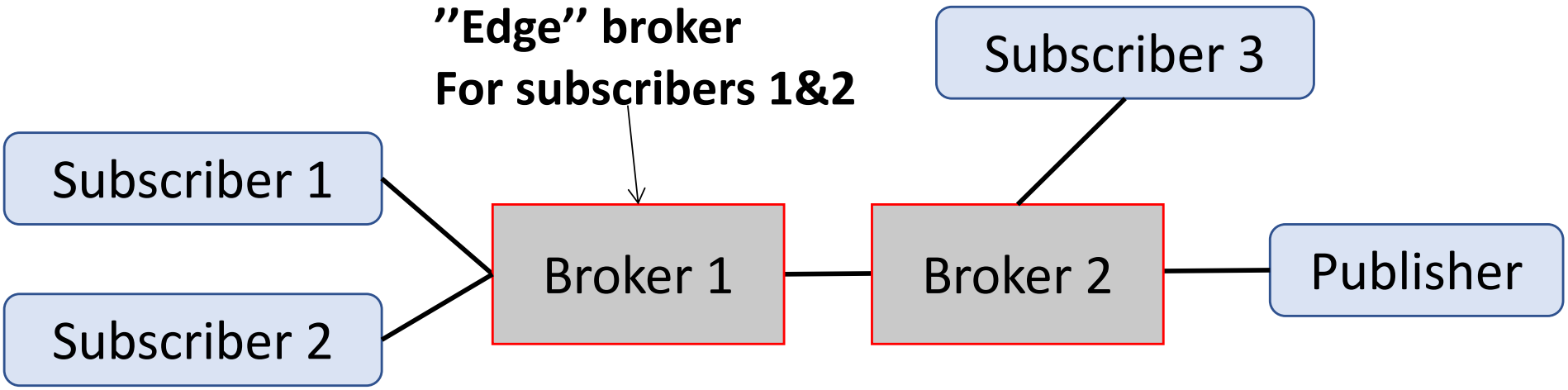


Subscribe(shortestPath (N1, N2))

SetEdgeDistance(E1, 5)

- Subscriptions automatically updated when graph changes
- Multiple Updates at the same time

Multi-Broker Systems



Graph replicated, subscribers and publishers distributed

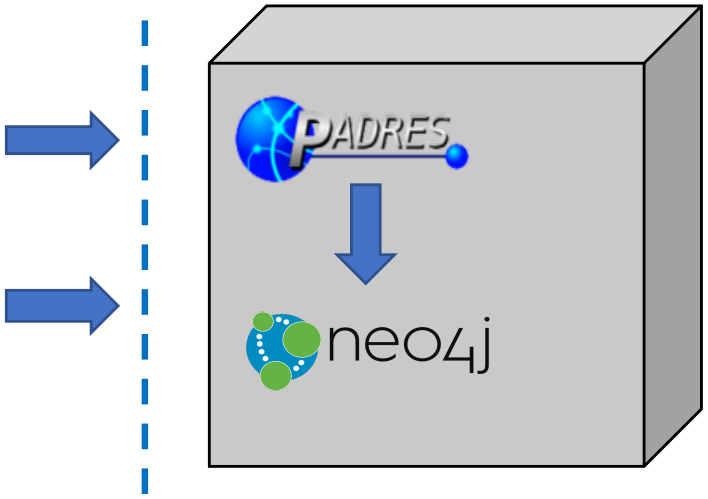
* how to know to which broker to forward a publication
* how to update the graph consistently

Or graph distributed, subscribers replicated?

* what about shortest paths that span multiple brokers

GraPS Summary

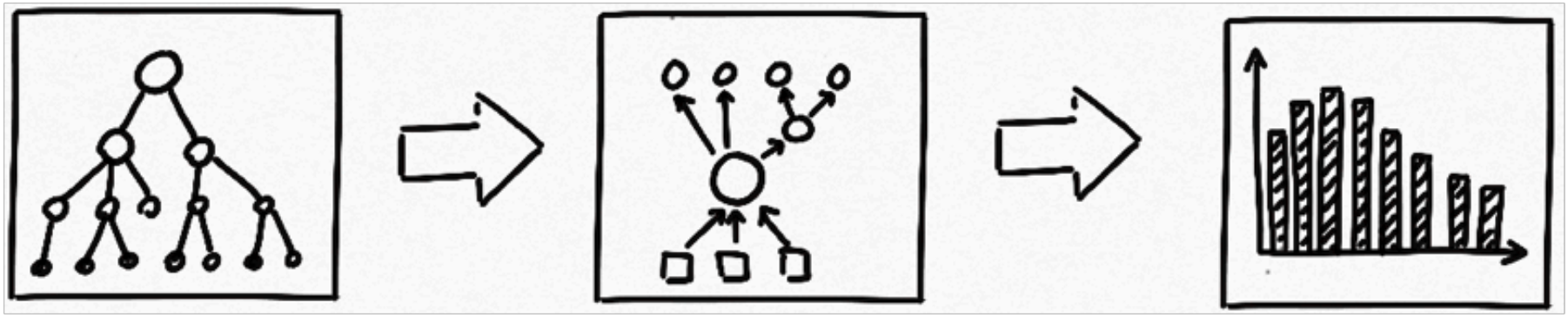
- Data Management *AND* pub/sub
- Scalability through distribution and replication
- Consistency of graph updates



Towards a reuse-driven design of Micro-services

Towards a reuse-driven design of Micro-services

How would you design a Micro-Service system ?



1. Select Micro-Service concerns & features to reuse

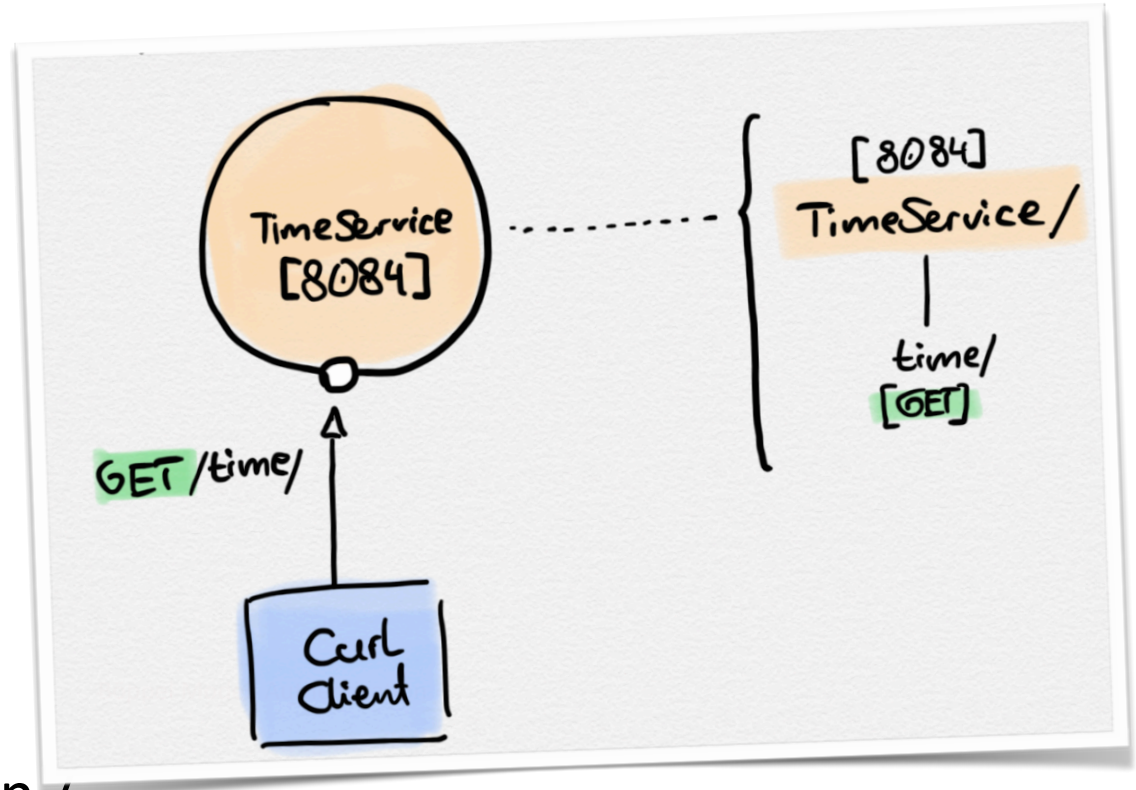
2. Infer Micro-Service Architecture

3. Examine properties

The described process requires accurate models for each stage.

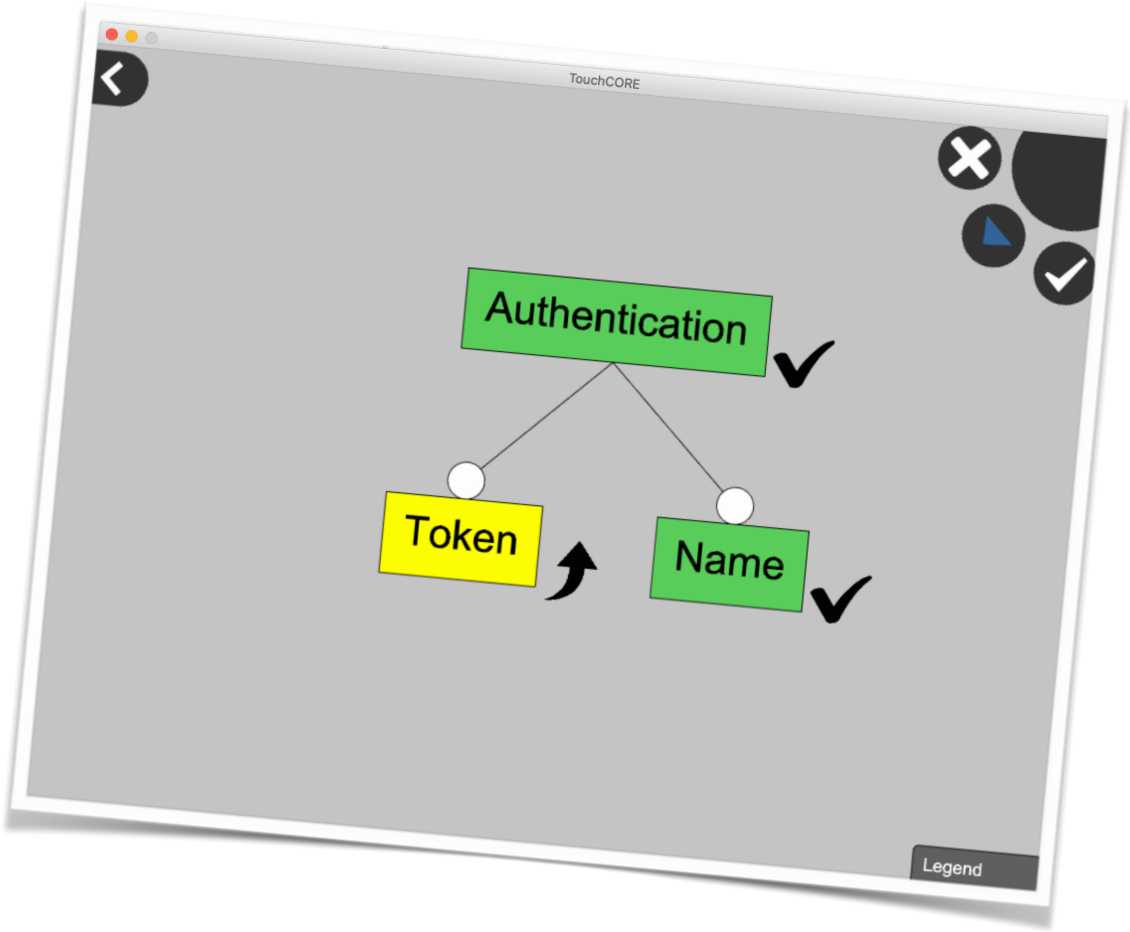
Architecture and Context Example

- Architecture
  - *TimeService*
  - *APIs: [Get] /time/*
  - *Dependencies: Client needs TimeService*
- Context
  - e.g. 50 clients, 1 request/10 ms
  - Deployment information / ports.



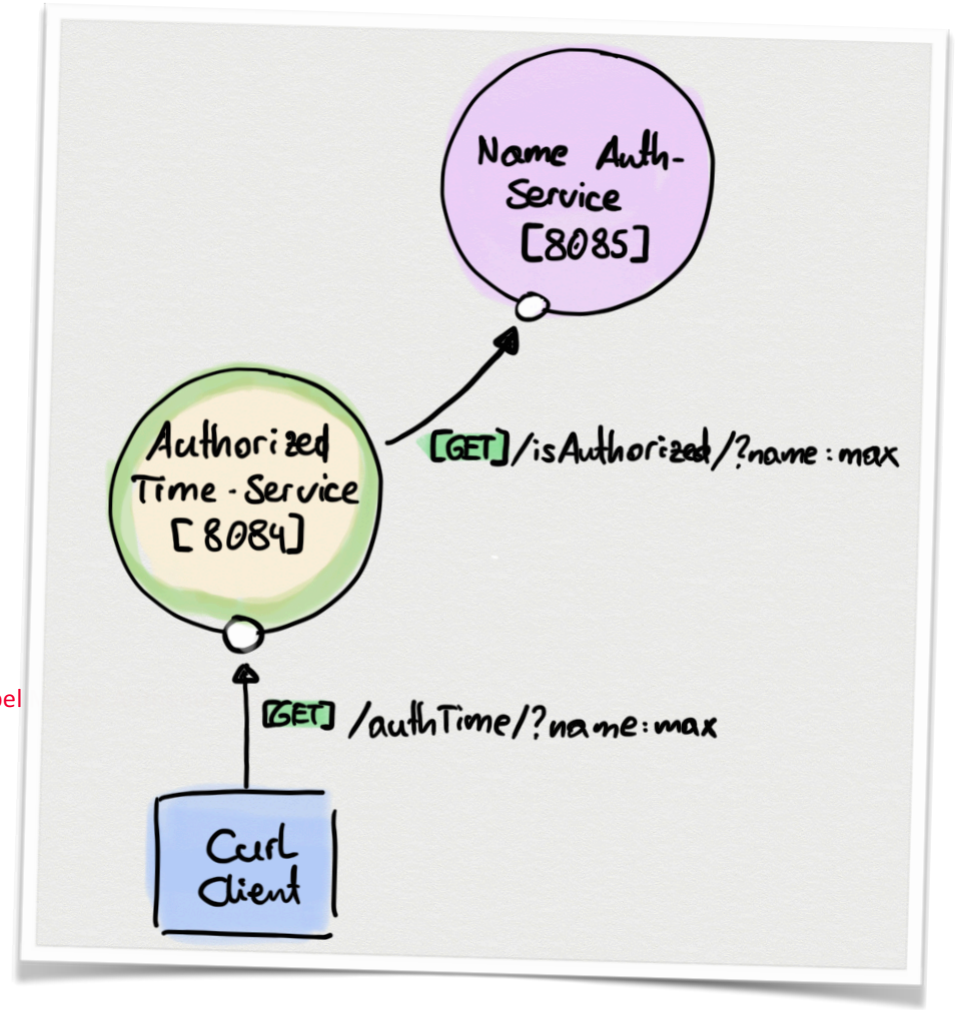
Plug-and-play: Authentication

- Concern: Authentication
- Feature Options:
  - Access Blocking
  - Authentication Means
    - Token
    - Name
  - Auto Logoff



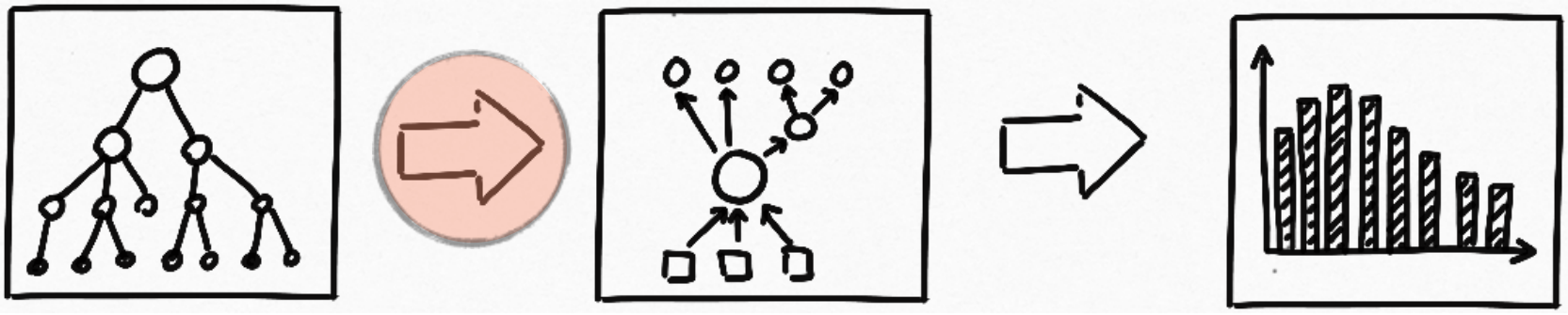
Architecture and Context Example

- Architecture
  - *TimeService, NameAuthService*
  - *APIs: [Get] /authTime/, /isAuthorized/*
  - *Dependencies: Client needs TimeService needs AuthService*
- Context
  - e.g. 50 clients, 1 request/10 ms
  - Deployment information / ports.



Böppel

Translating Reuse Selections



1. Select Micro-Service concerns & features to reuse

2. Infer Micro-Service Architecture

3. Examine properties

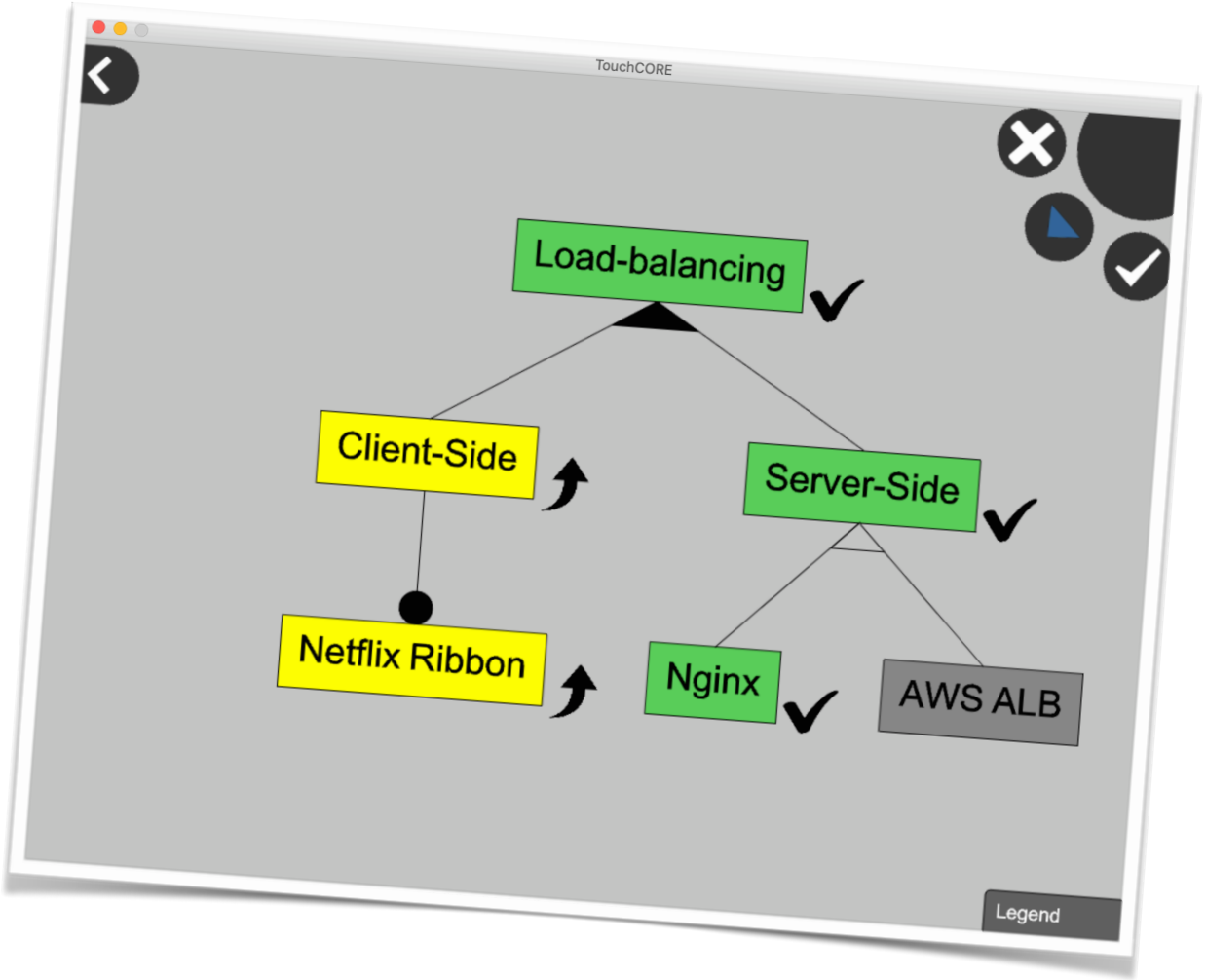
- Not all selections translate directly into services!
  - Authentication -> New extra service
  - Encryption -> New arrangement

Functional vs. Non-functional

- Functional
  - Time-service, Authentication Service
- Non-functional
  - Load-balancing, fault-tolerance, encryption
- Or is it really so easy to distinguish between the two?

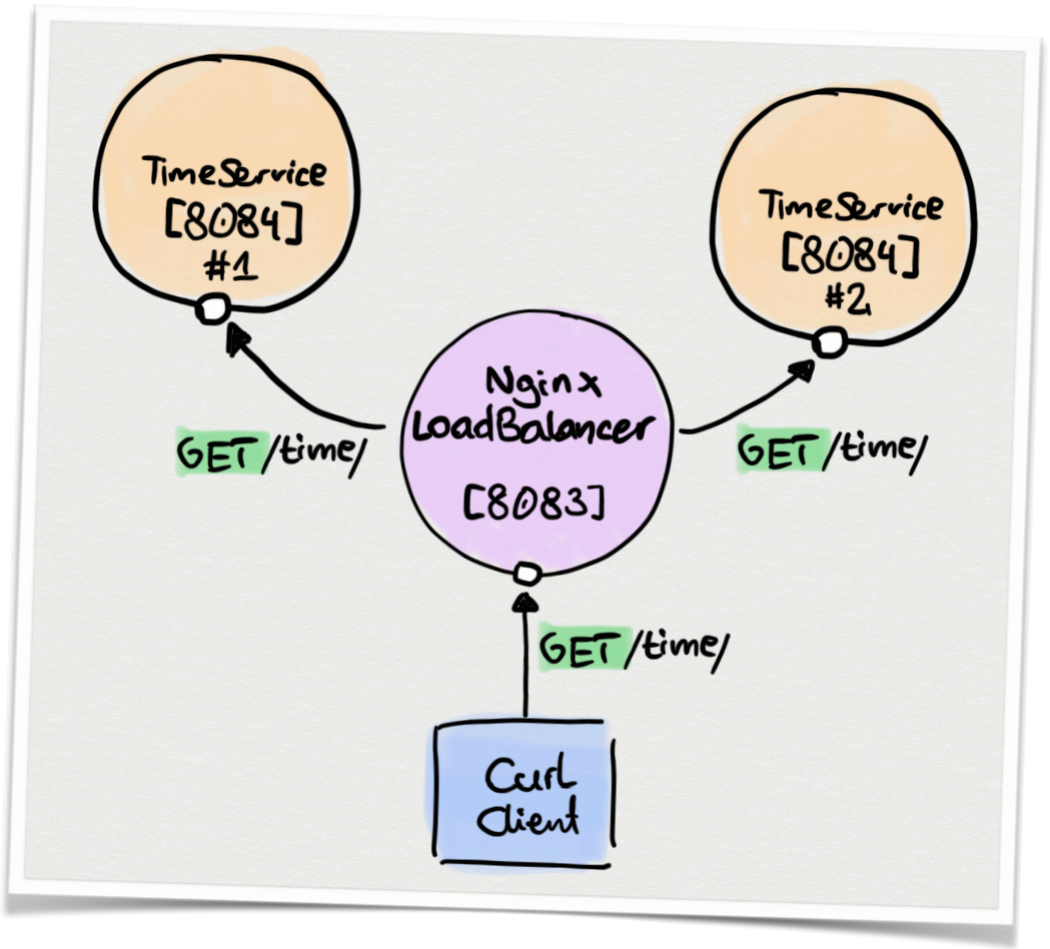
Example 2: Load Balancing

- Concern: Load Balancing
- Feature Options:
  - Client Side
    - Netflix Ribbon
  - Server Side
    - Nginx
    - AWS ELB

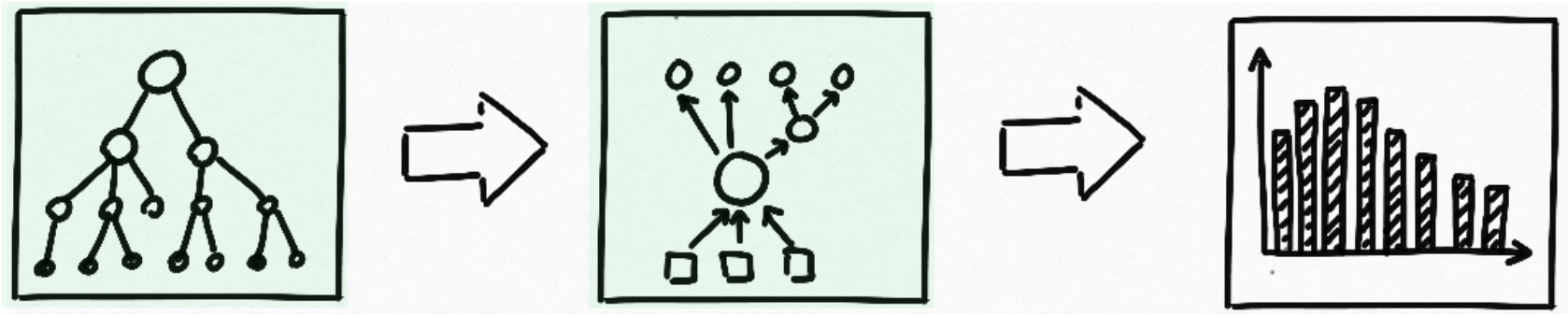


Example 2: Load Balancing

- Architecture
  - *LoadBalancer, 2x TimeService*
  - *APIs: [Get] /time/*
  - *Dependencies: Client needs LoadBalancer needs TimeService*
- Context
  - 50 Clients, 1 call / 10 ms.
  - Deployment information / ports.



The reuse approach



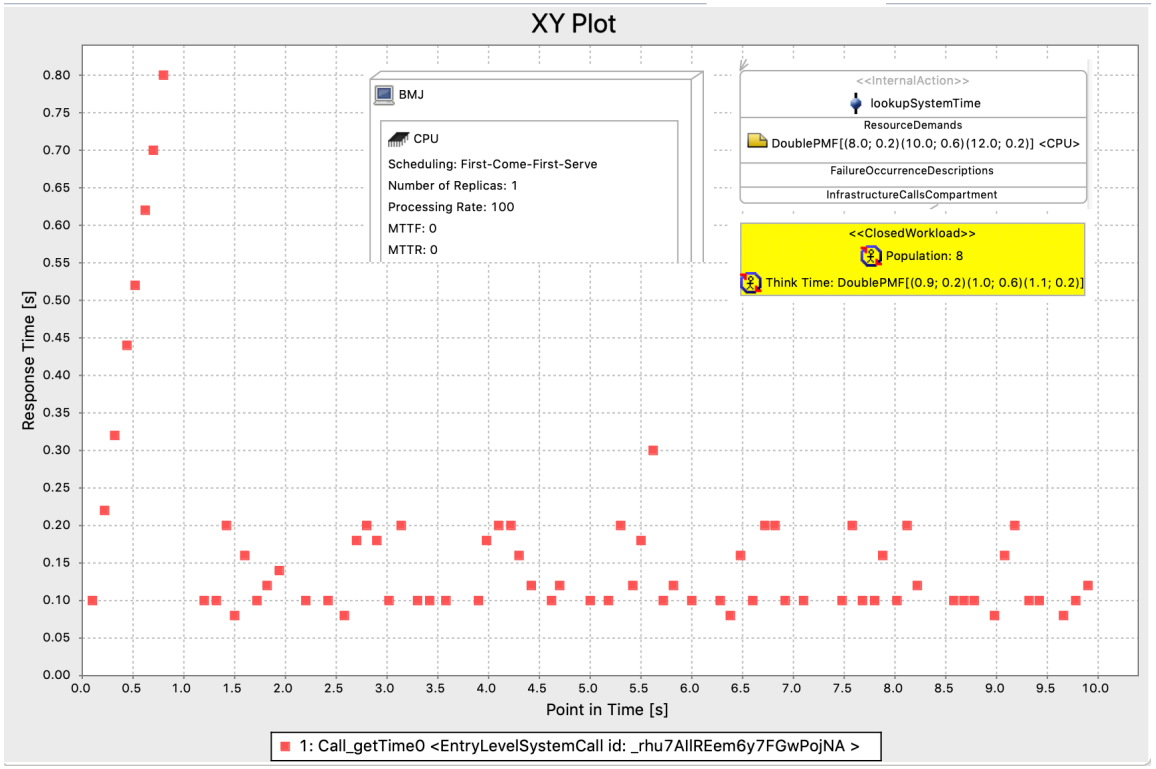
1. Select Micro-Service concerns & features to reuse

2. Infer Micro-Service Architecture

3. Examine properties

Architecture Evaluation

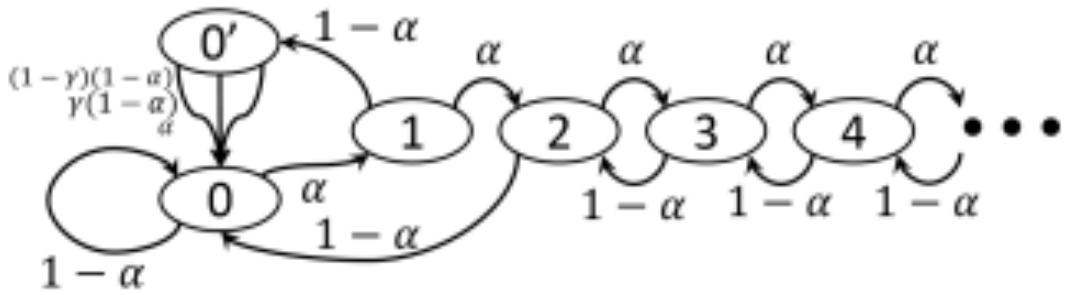
- Quantitative properties
  - Performance
  - Cost
- Qualitative properties
  - Reliability
  - Security
  - Maintainability



Modeling rational behavior in Blockchain mining

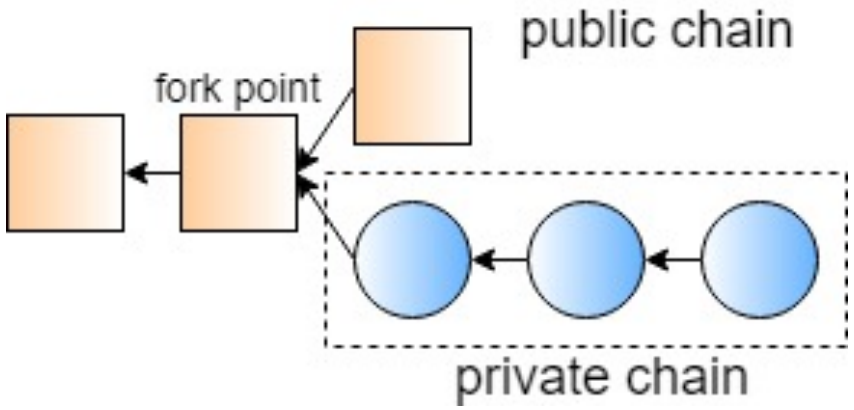
Rational Mining Behavior in Blockchain

- In most Proof-of-Work (PoW) blockchain systems, including Bitcoin, *honest miners* always choose to mine on top of the longest chain.
- *Rational miners* might choose an alternative strategy if the expected reward is eventually larger than the proportion of their mining power.
- Eyal et al.[1] describes *selfish mining* as one form or rational mining.
- They use a 1D Markov model to specifically analyze the profitability when there is one selfish miner (or mining pool) and one honest miner (all honest are grouped together as one large honest miner).
- The strategy is profitable when the mining power is between 1/4 and ½.

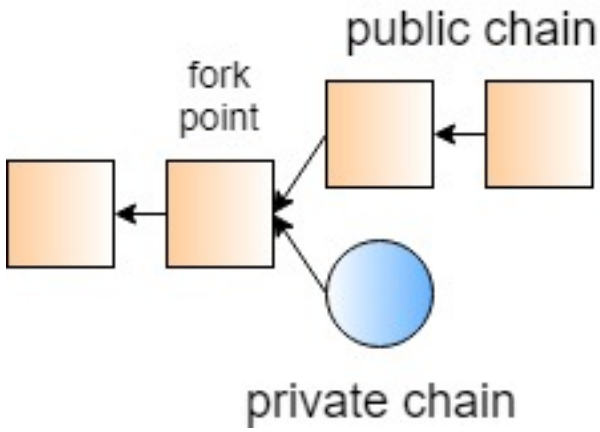


Model other strategies?

- Selfish Mining
  - Does not release the newly mined block immediately but waits until the length of the public chain catches up with the private chain
  - Gives up when the public chain is longer
  - Gains higher proportion of rewards by overwriting the public chain
  - Wastes the mining power of honest miners

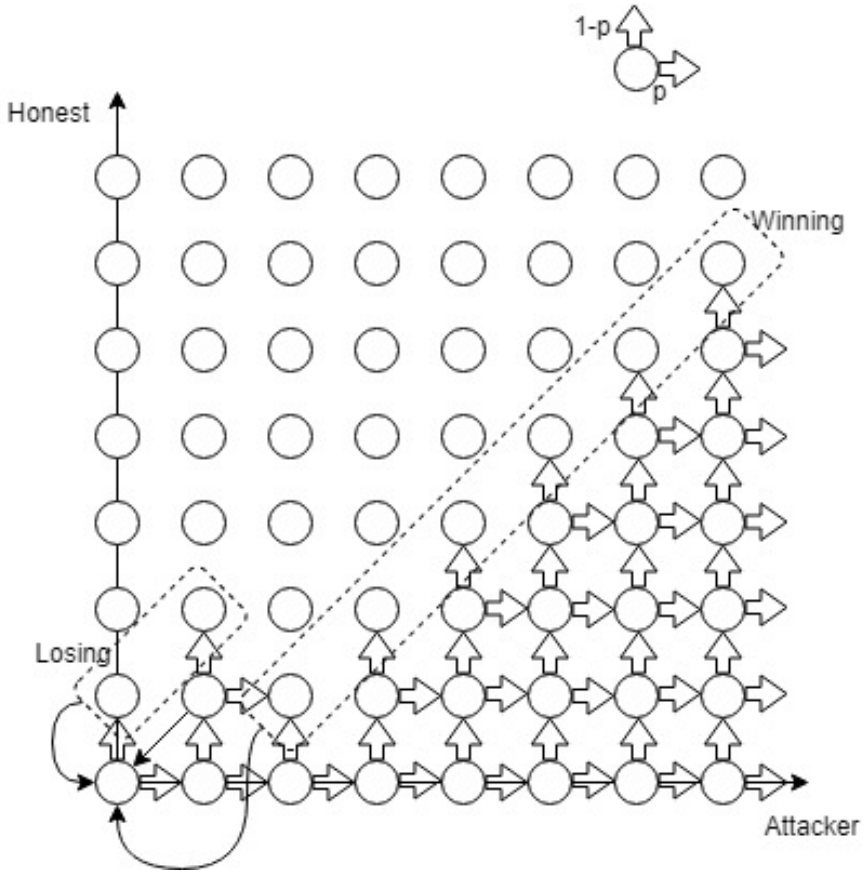


- Stubborn Mining
  - Does not give up even if lagging behind the public chain
  - Overwrites the public chain by chance
  - wastes the mining power of honest miners



2D Markov Model: Selfish Mining

- Selfish mining



- When $p > \frac{1-\gamma}{3-2\gamma}$, $\mathbf{E}[r_A] > p$

2D Markov Model: Stubborn Mining

When $p > 43.0\%$,
$\mathrm{E}[r_A] > p$

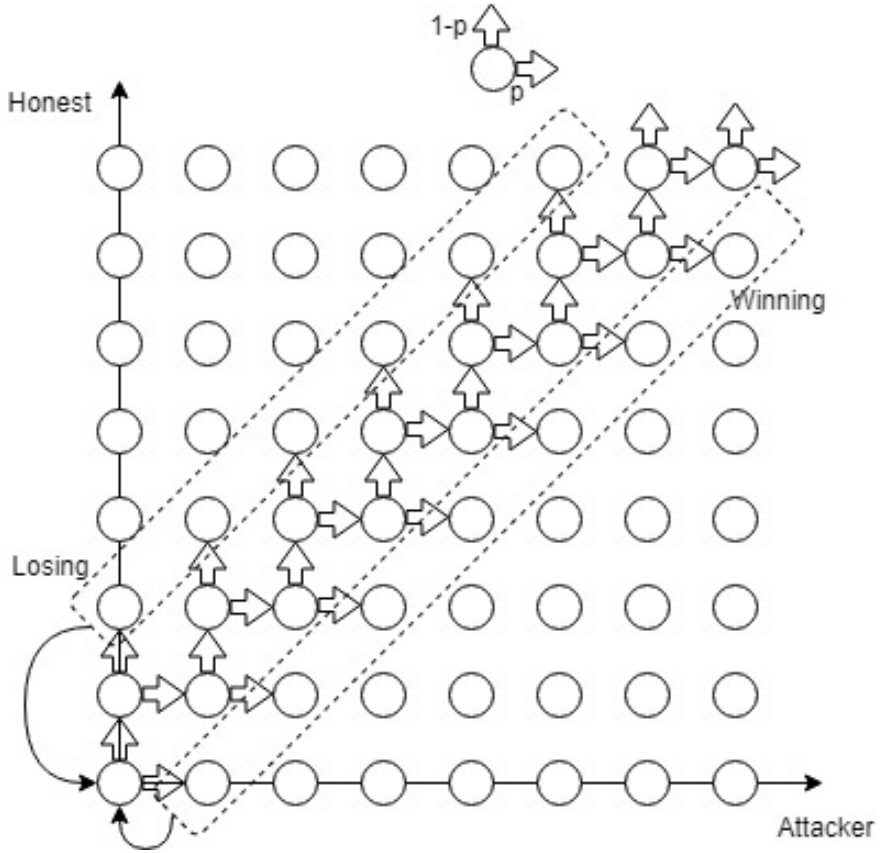


Fig. 6 2D Markov Model of Naive Stubborn Mining

Simulation of Multi-player game

- Assumption: more than one selfish miner
- Assumption realistic:

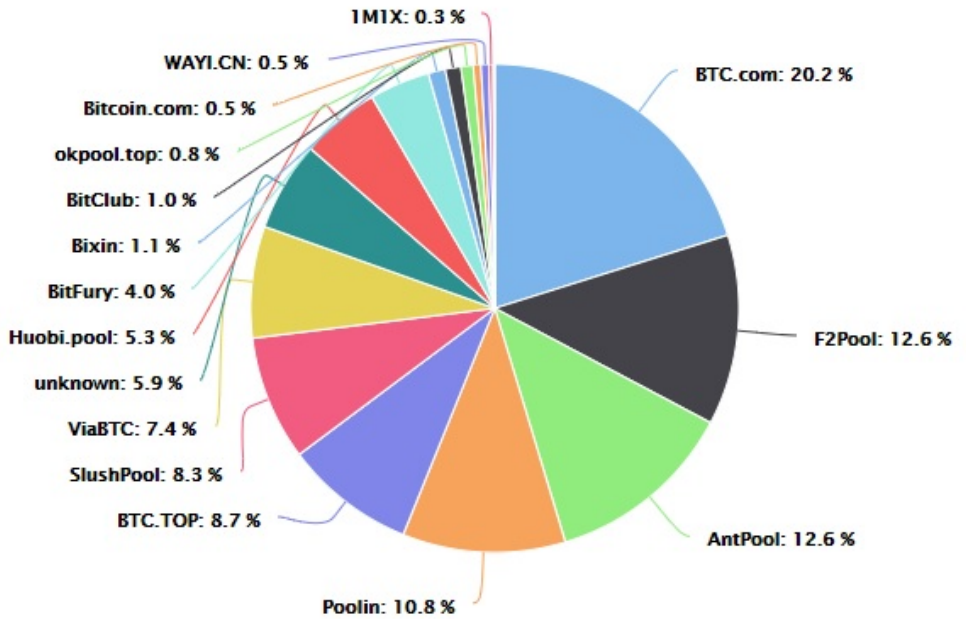
  Other miners can detect that there is a selfish attack but cannot identify the attacker.

  Selfish miners themselves will not publish their identity to avoid being expelled

  Rational miners, when detecting that somebody else is selfish, might start being selfish themselves.

- Simulator for 3-player game

  2 selfish miners, one honest miner



Results

- Rewards of different miners by the mining power of attacker 1/ attacker 2 (heatmap)

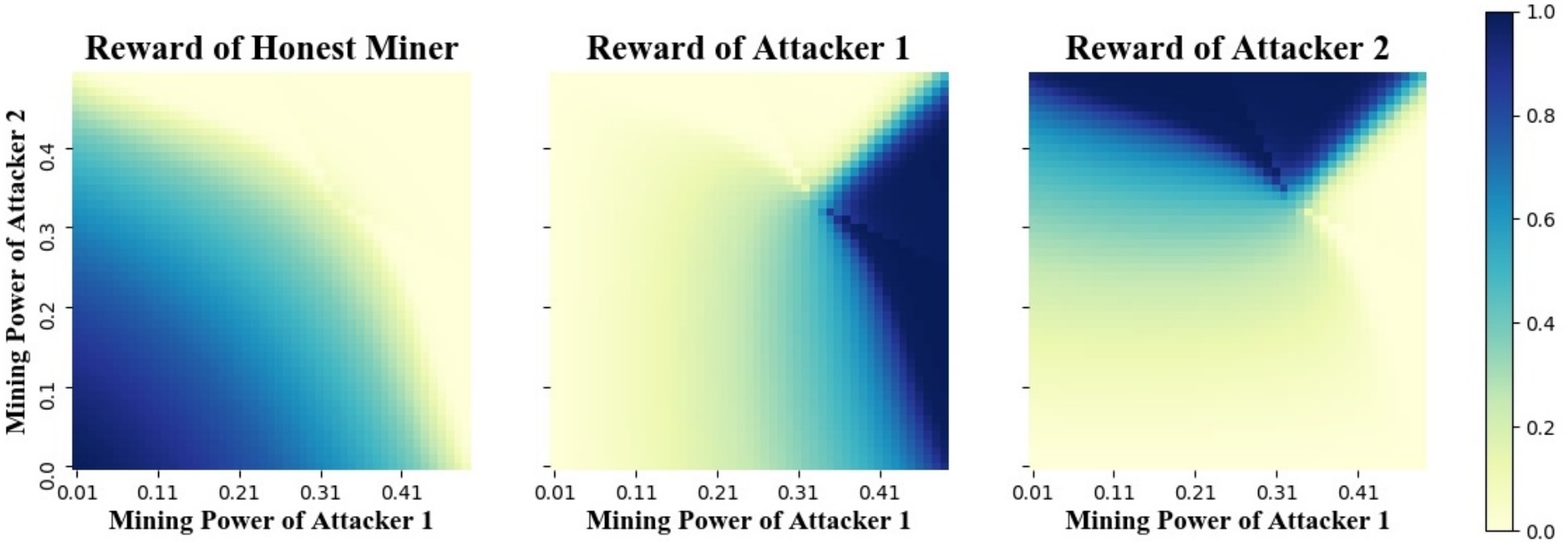


Fig. 10 Rewards of Honest Miner and Attackers in 3-player Game

3-player game

- There exists a small area where it can be profitable for both attackers simultaneously, for example, when $(p_H, p_{A1}, p_{A2}) = (0.5, 0.25, 0.25)$, the profitability of each miner is $(Profit_H, Profit_{A1}, Profit_{A2}) = (0.4565, 0.2718\ 0.2717)$.
- It won't be profitable if the attacker's mining power is under 20%.
- The reward of the first attacker slightly increases when the second attacker comes in. And it decreases drastically when the second attacker becomes dominant.
- The reward for the attackers would be larger if they worked together as one big attacker. That is, finding other selfish miners is more beneficial compared to attacking alone.

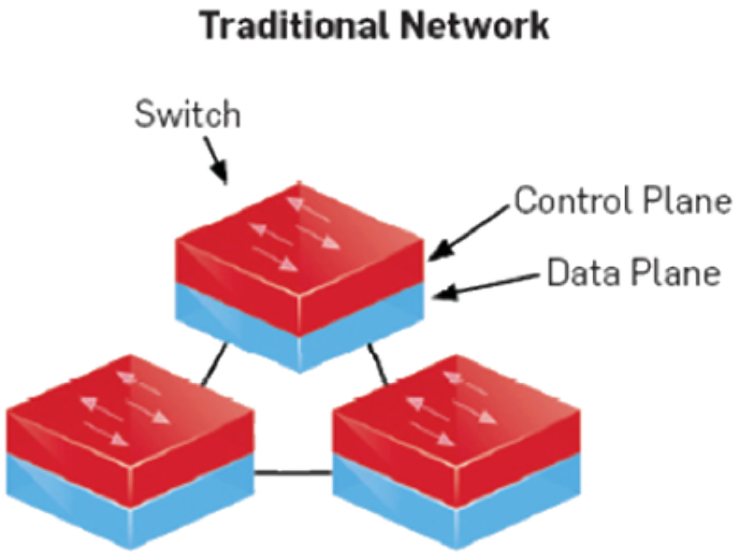
Open Data Science

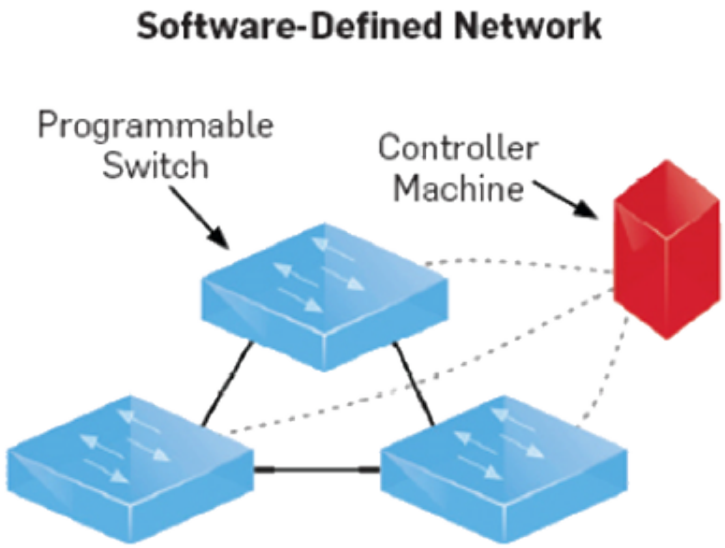
- FAIR Principle of Data Management:
  - Findable
  - Accessible
  - Interoperable
  - Reusable
- Open Science
  - Collaborative
  - Transparency
  - Reproducibility
    - Data Transformations performed by distributed programs
- Tools needed
  - Distributed Data Management
  - Large-scale workflow based computations (flavor Spark)
  - Development kitchen
  - Project Provenance

Application Driven System Development

- Application centric view
- Detect new application needs
  - Data centric (graph)
  - Mining behavior in bitcoins
  - Open Data Science
- Weaving functional and non-functional modeling
- Find generic solution
- Make it work

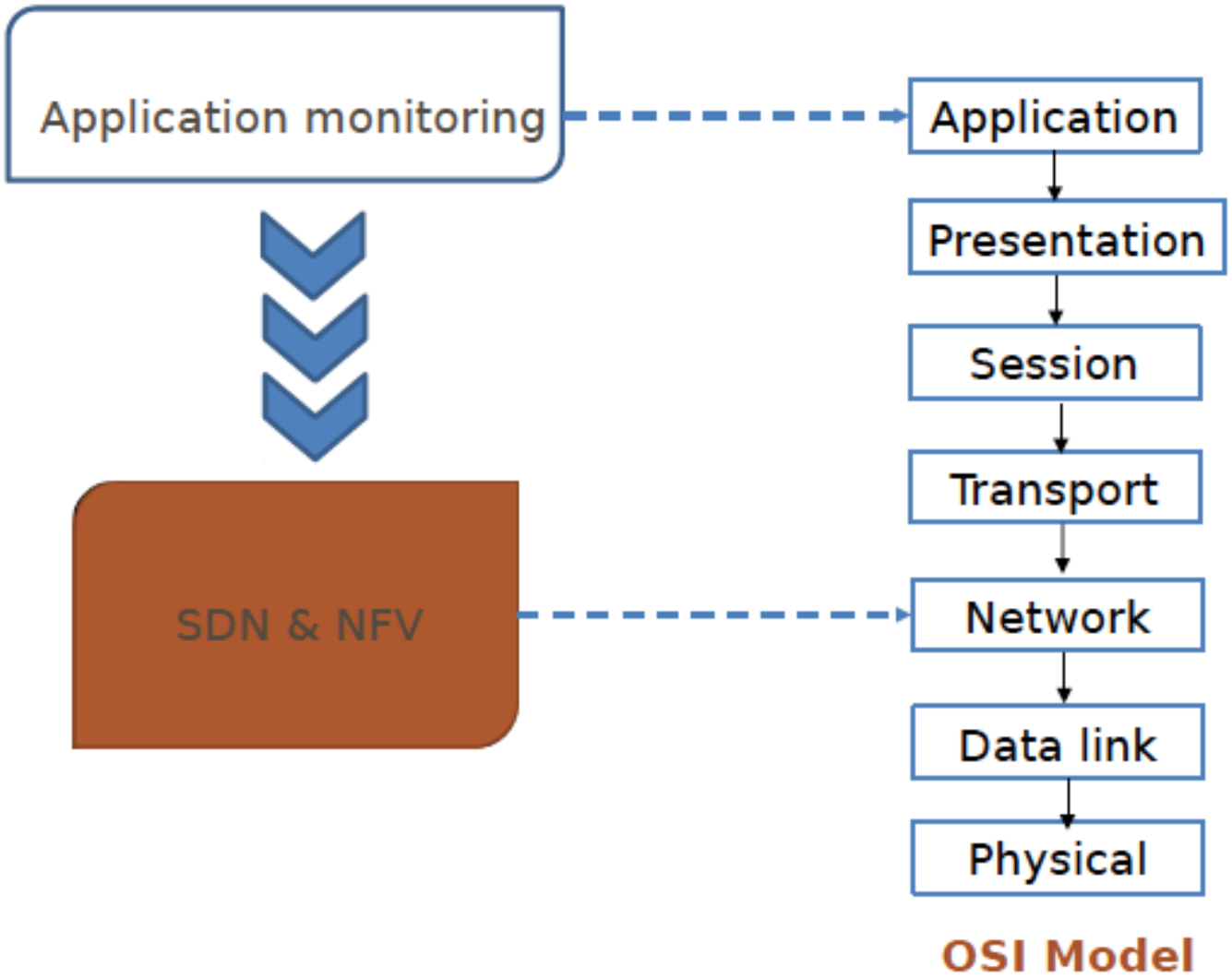
Network-level Monitoring-aaS

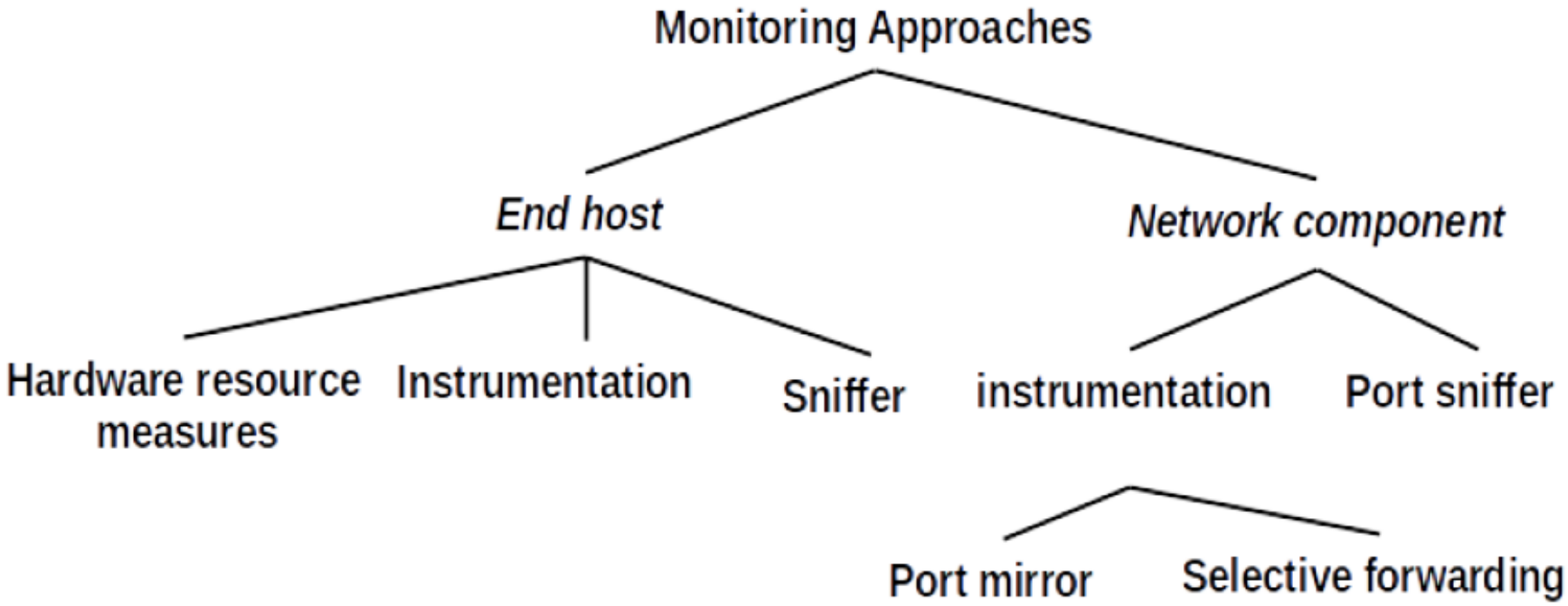




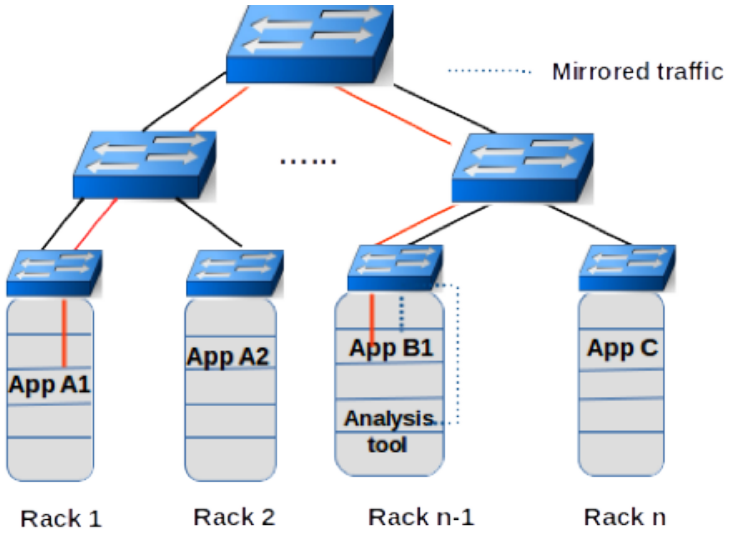
Datacomm company website: https://www.datacomm.co.id/en/telco/sdn/

Research Objective

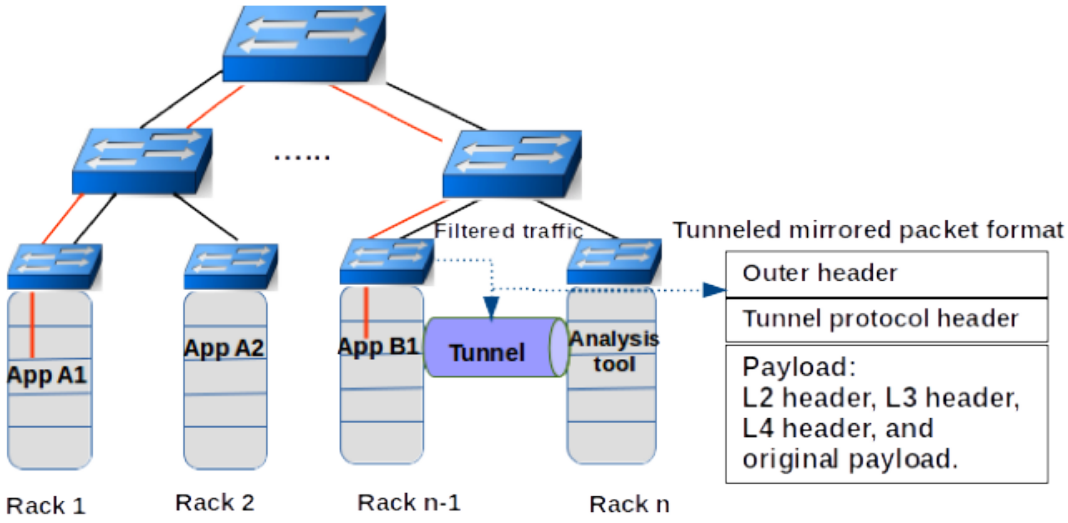



Monitoring Approaches
End host
Network component
Hardware resource measures
Instrumentation
Sniffer
instrumentation
Port sniffer
Port mirror
Selective forwarding

Monitoring options in the Network

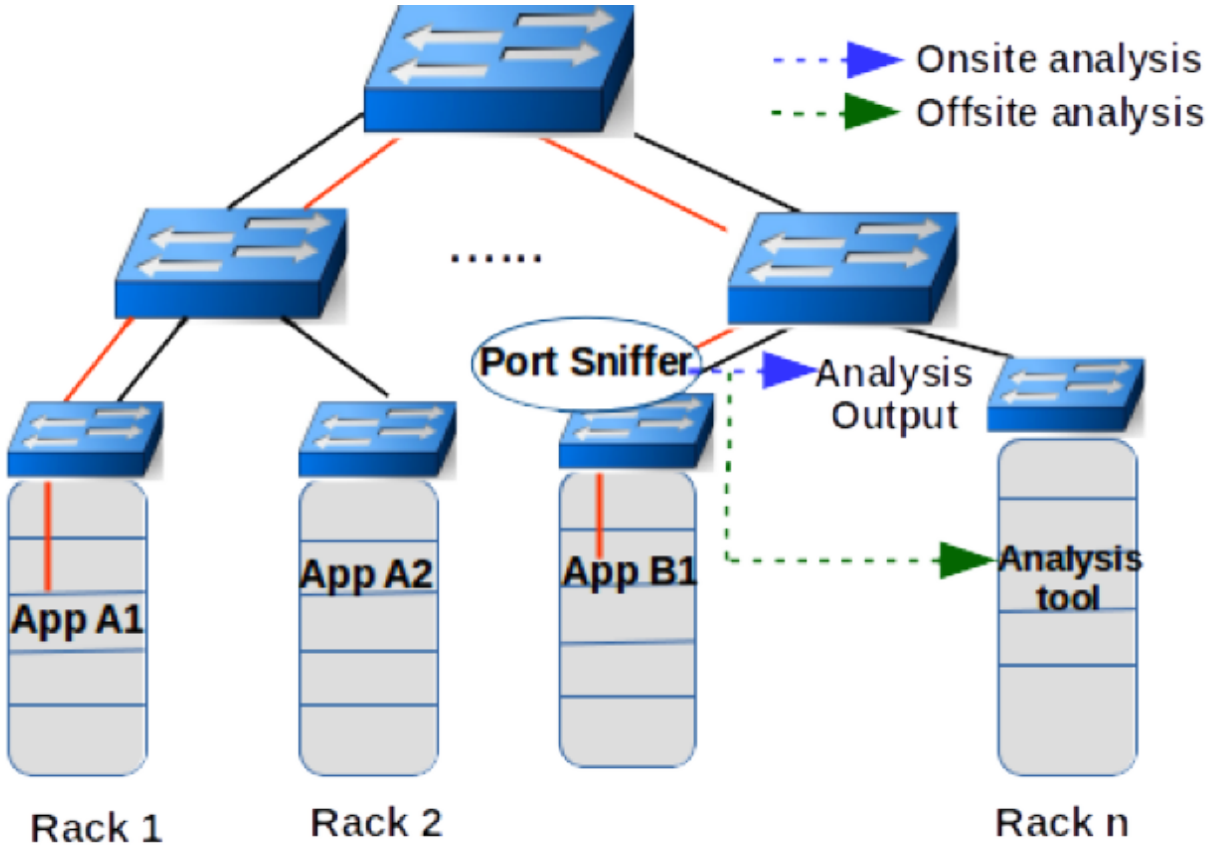


(A) Port/Selective mirroring



(B) Tunneled selective mirroring

Proposed port sniffer



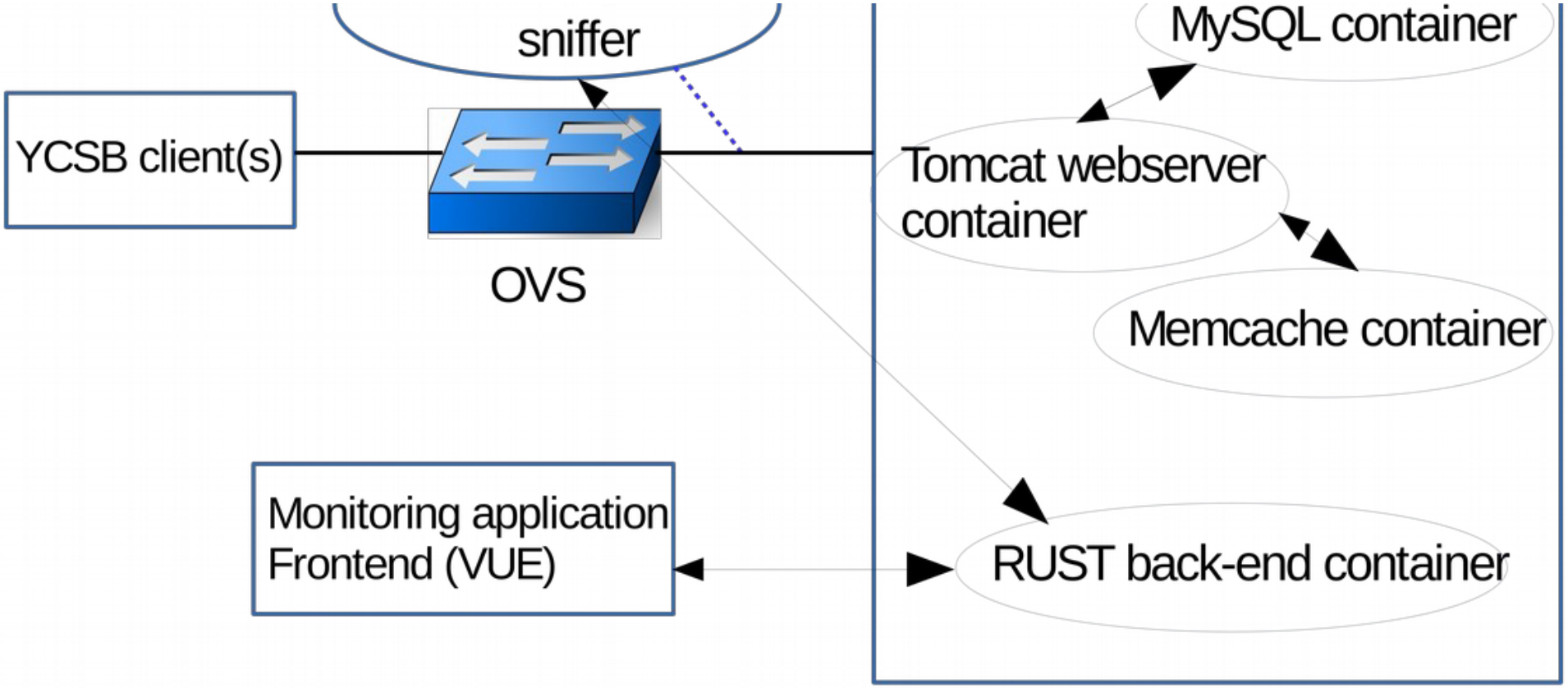
mirrored packet from port sniffer

TCP/UDP header
IP header
Ethernet header
Payload: selected information from original packet header and data.

What to monitor

- Request response time
- Throughput
  - Overall
  - Per object/method/client
- Number of different clients
- Error Rate
- ?

Current Prototype



Teaching and Training

Teaching and Training Interests

- Distributed Systems Course
  - What are ”classic topics”
  - What are new developments that are a “must”
  - What are the skills to be learned
- Distribution, consistency, dependability everywhere
  - How to coordinate with other courses
- What should a PhD student all do
  - Internships?
  - Collaboration?
  - Supervision?