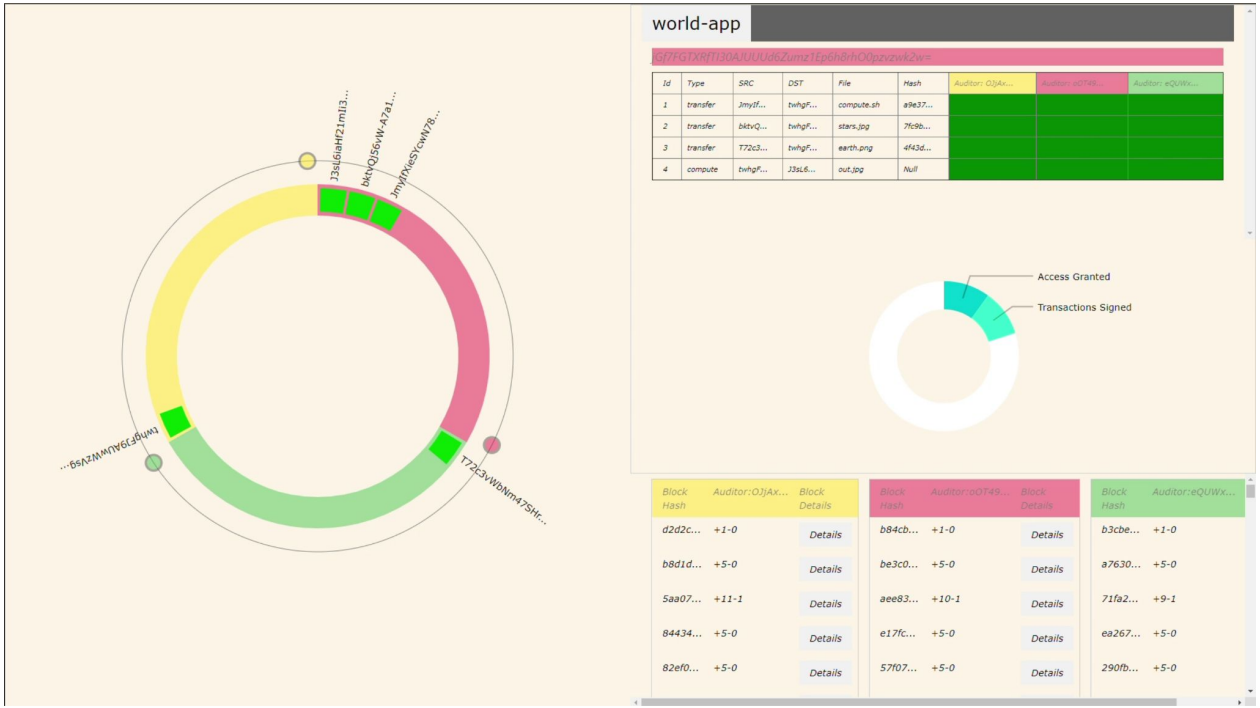
On Multilateral Agreements And Multidomain Applications

Reggie Cushing
r.s.cushing@uva.nl

Story so far...

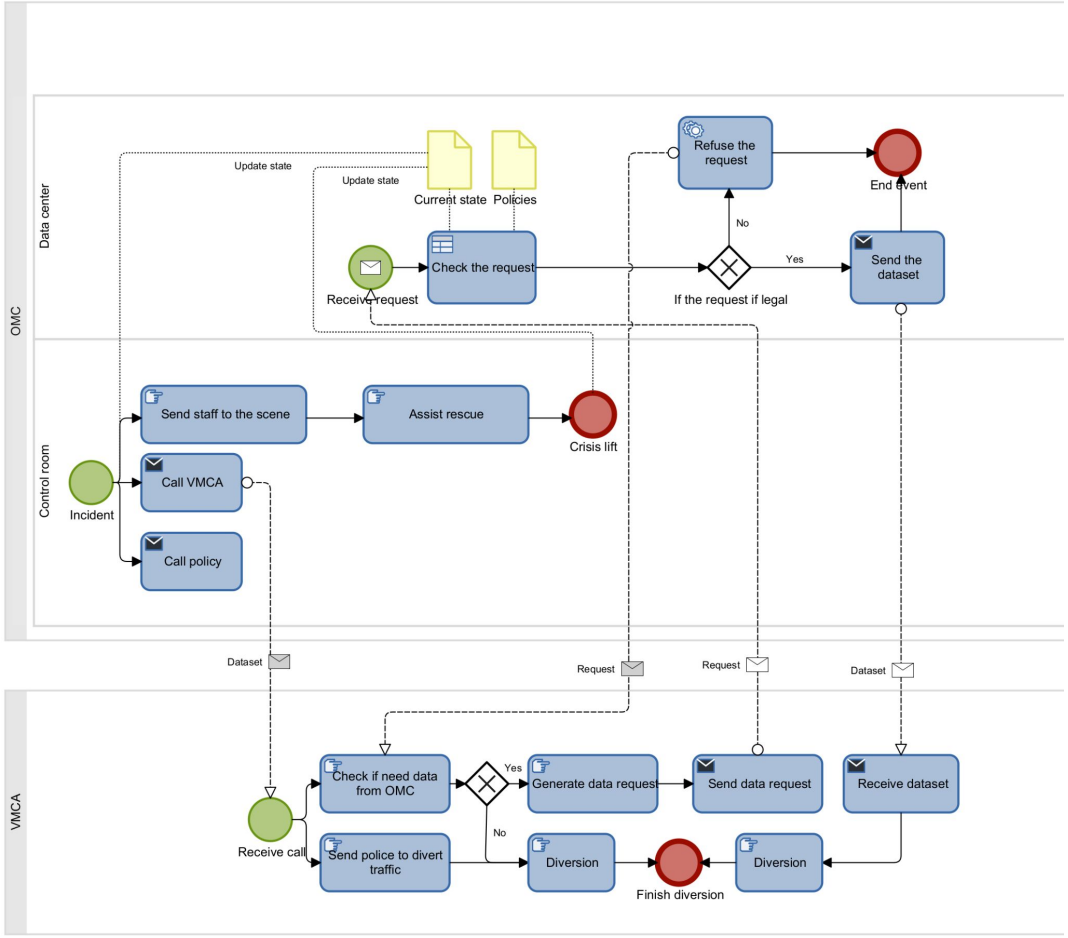


- **Actors** = Containers
- Actors cryptographically addressed
- Multidomain communication through MQ using actor keys as topics.
- **Auditor** actors give permission to actors to carry out actions
- **Planner** actors encapsulate the notion of a workflow
    - Planners coordinate with auditors to execute workflow

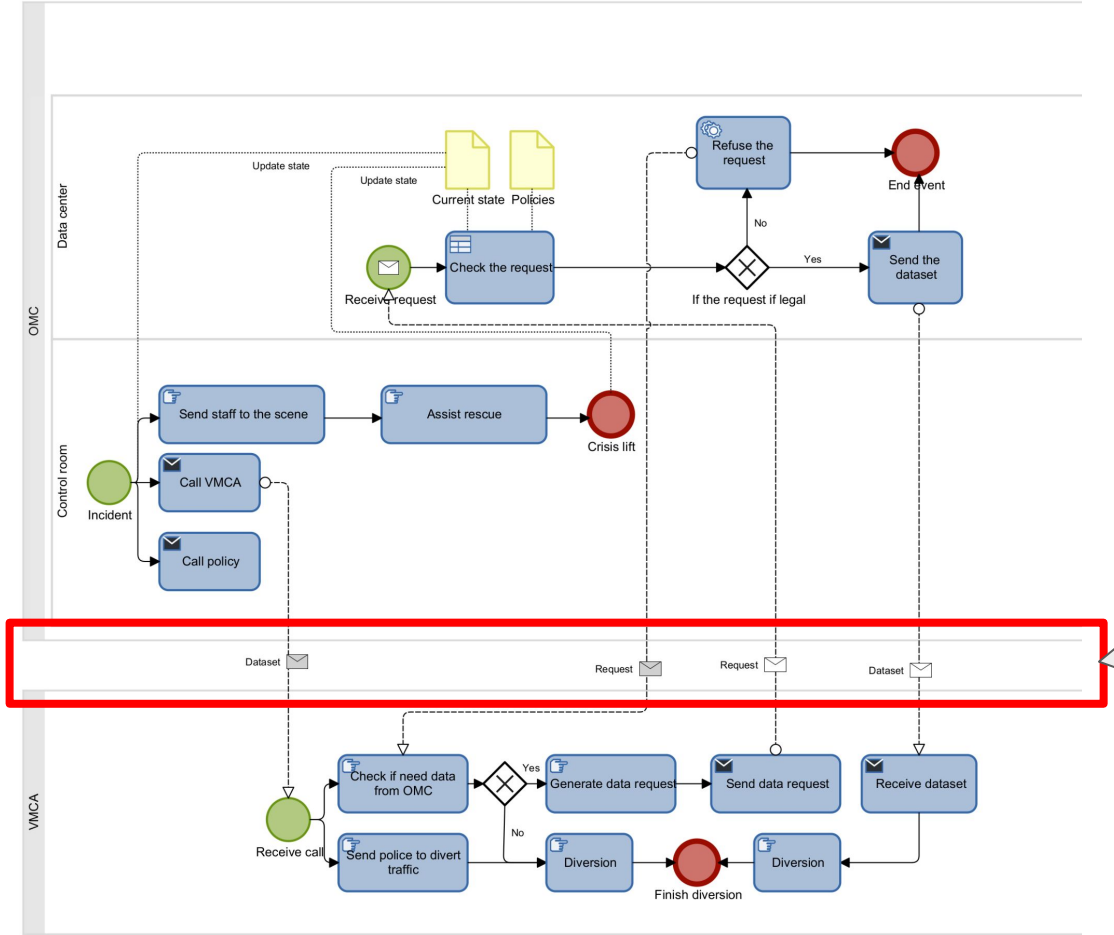
Moving forward...multi-domain coordination

- A multidomain application is a workflow whereby the (data|control)flow crosses domain boundaries.
- Domain boundaries are controlled through rules/agreements derived from policies.
- A use case can be considered as having multiple facets.
  - The application functional components (functions)
  - The data assets
  - The coordination logic (controlflow)
- Controlflow is a program in itself that is *owned* by multiple domains.
- The challenge is:
  - *How to execute a control program owned by multiple domains?*

ArenA use-case multi-domain process model in BiZZdesign

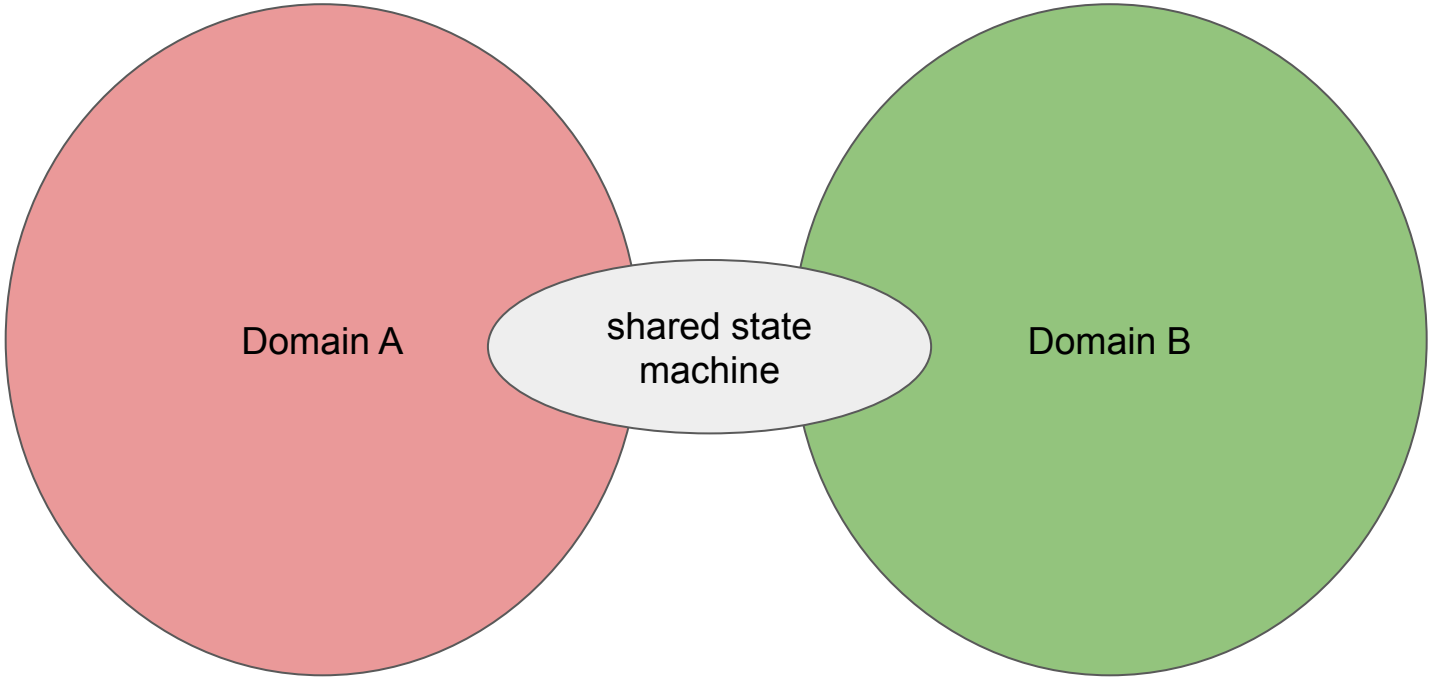


ArenA use-case multi-domain process model in BiZZdesign

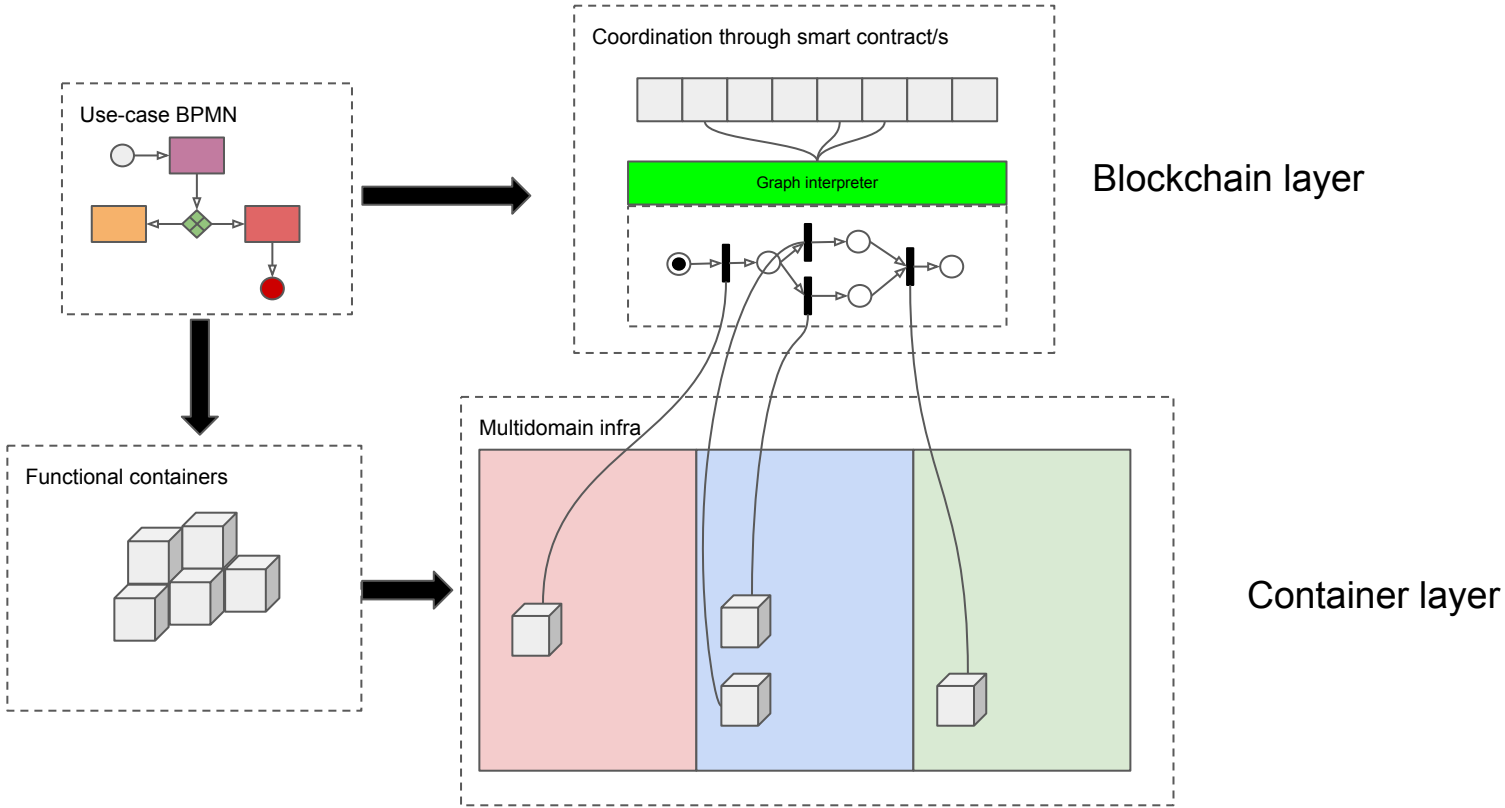


- Track, control, coordinate cross-border processes.
- Traditionally a **static** layer using API keys etc.
- In a marketplace we propose a **programmable** layer.
- We need to capture and coordinate these set of rules in a transparent and secure way.
- We propose state machines to keep track of the state of the border.
- Each party/domain updates the state machine thus signaling the other parties to take action.

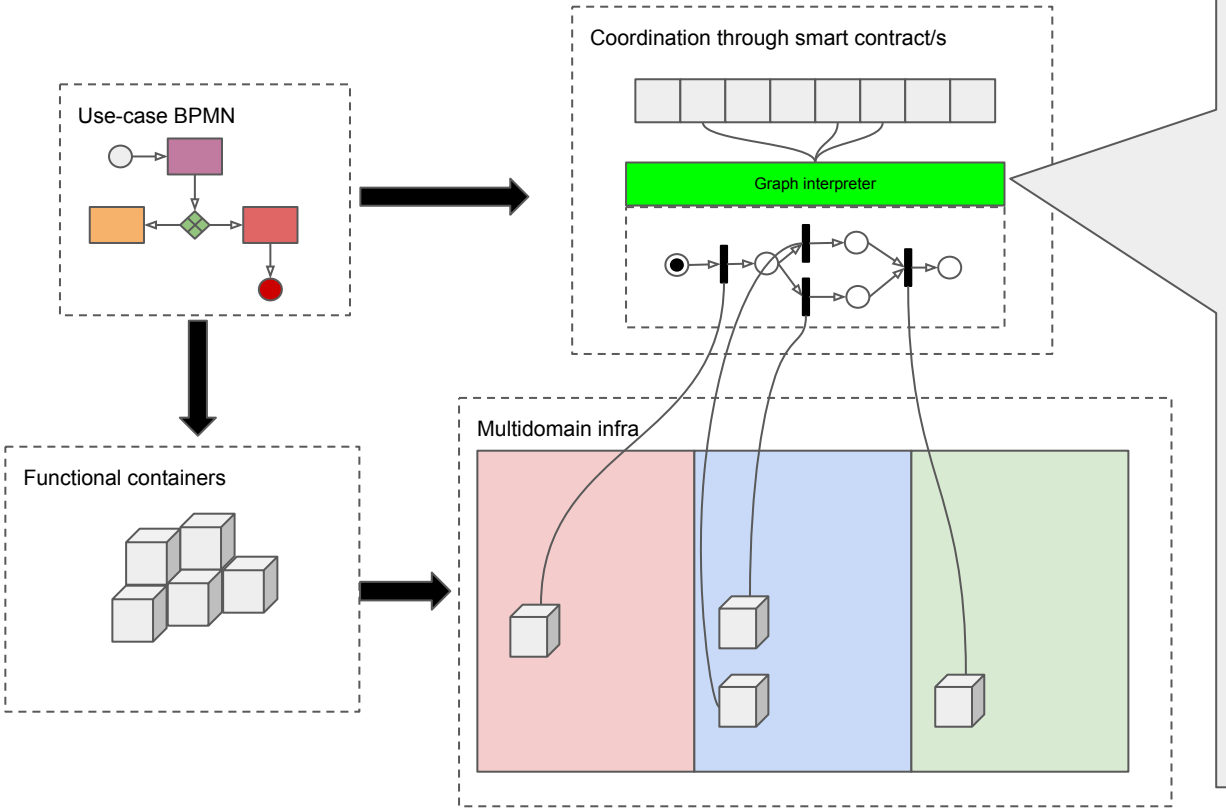
Shared state machine



Process model to infrastructure

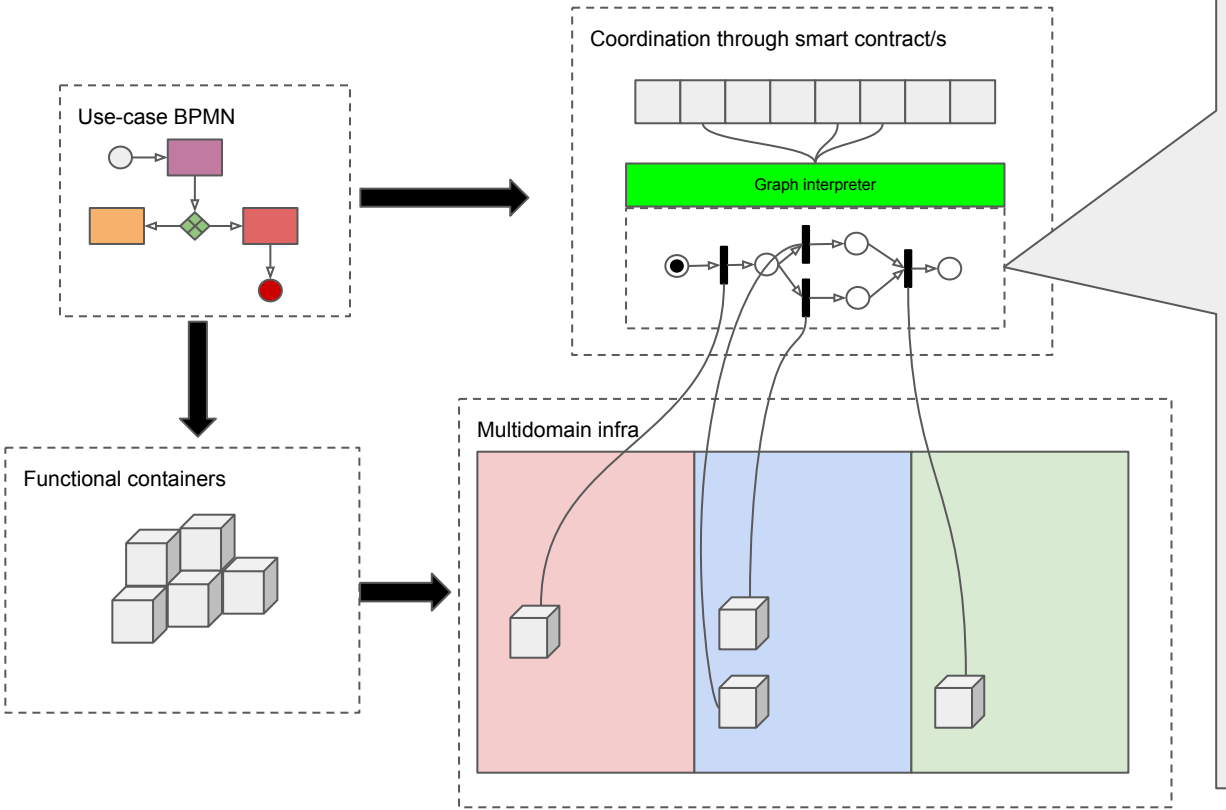


Process model to infrastructure



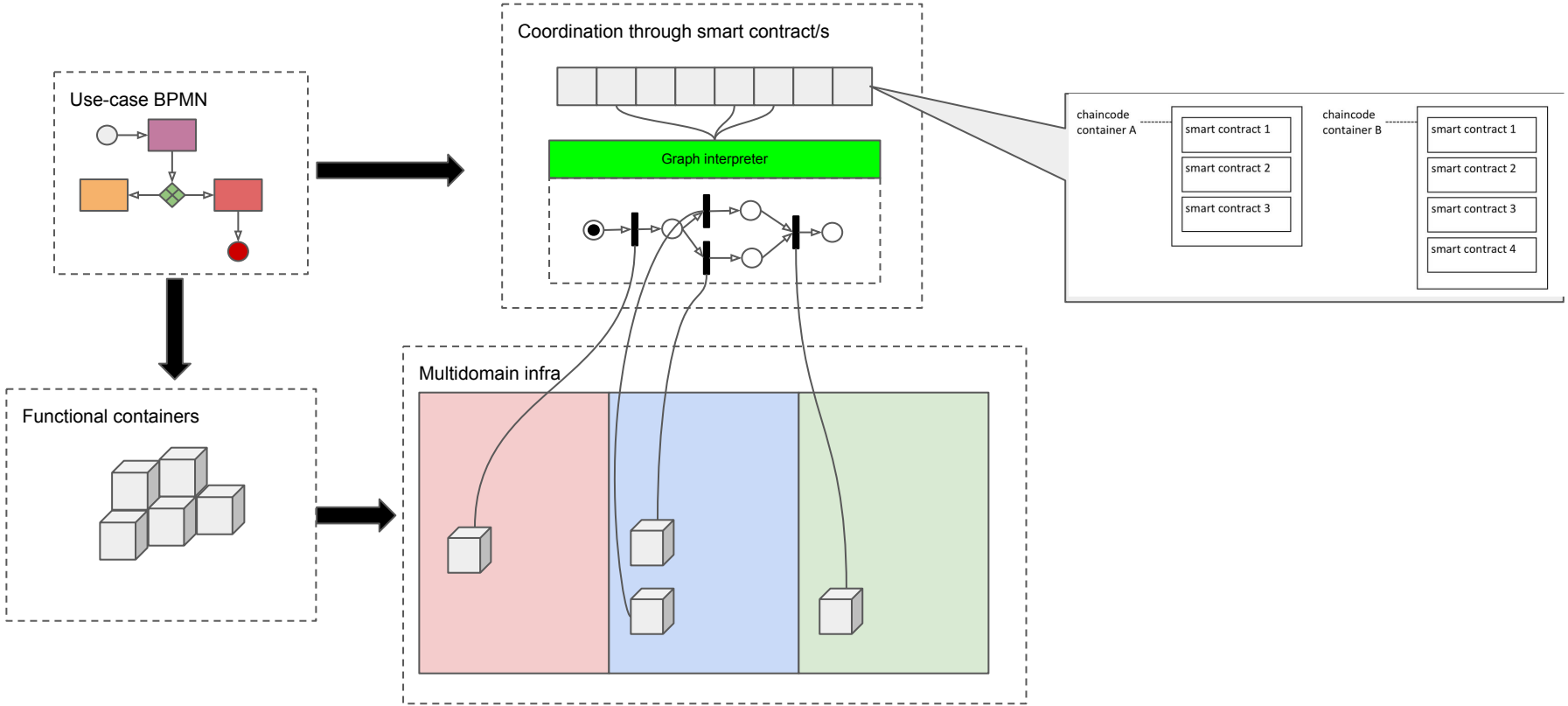
- **Generic** dataflow/petrinet executor running on a blockchain i.e. every peer is running the executor.
- Domains/actors are assigned a set of tokens.
- Actors define functions as a task with token input, token outputs and webhooks to interact with the outside world.
- So actors own tokens and tasks
- A task needs certain amount of tokens to **fire**
- Blockchain transactions copy tokens between actors.
- When a task has enough input tokens it will **fire** which in turn generates blockchain events.
- Containers monitor the ledger to trigger a process inside a container (the task).
- The container will make blockchain transactions to signal the task is completed and move the state machine.

Process model to infrastructure



- a petrinet that regulates how multi parties collaborate.
- the rationel is that a party can only perform a certain task given a certain context.
- Context is a set of multi domain preconditions that have to met.
- The movement of tokens within the graph changes the context.
- Tasks running outside the their agreed context are deemed as illegal.
- The approach should be generic enough that it can be applied to event driven scenarios.
- Petrinets have strong mathematical foundations.
- Can be analysed for behaviour and structure properties e.g. reachability, boundedness liveness, reversibility, coverability...

Process model to infrastructure



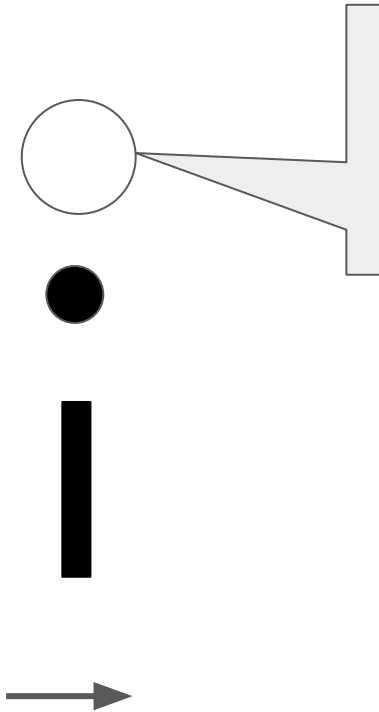
Beneath the blockchain buzz words; a computer scientist's view

- Is a distributed database.
- Instead of storing the DB data, store the transactions the made the data.
- Data **'asset|token'** is cryptographically signed data struct by users **'owners'**.
- Changing owner's signature of data is a **'transaction'.**
- Users have pki keys. **'accounts|wallets'**.
- Use a linked list to store the transactions **'blockchain'**.
- Reference(hash) the previous list's recordset **'block'** in the new block.
- Multiple nodes need to agree on recordset order **'consensus'**.
- Multiple nodes can rebuild the data from the linked list.
- Since multiple nodes can do *something* then they can also run scripts **'smart contracts'**.
- End result is a distributed network that can run deterministic scripts to manipulate a shared linked list where records are owned by different users.

Blockchain primitives

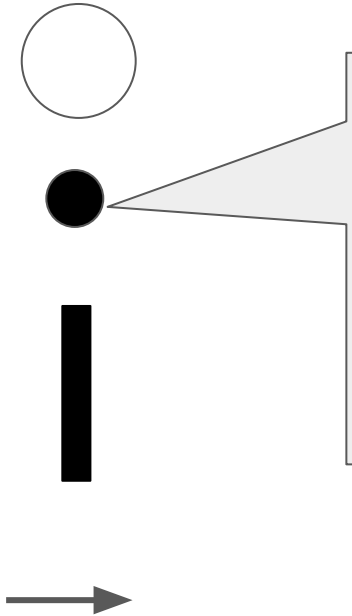
- Participants
  - Users with an x509 cert given by a CA peer on the network.
- Assets
  - User defined data structs owned by a participant.
  - **Cryptographically signed** data structs.
- Transactions
  - Move assets between participants
- Chaincode(smart contracts)
  - Javascrip/go/java programs to create programs with these primitives.
  - The chaincode runs on all/multiple peers of the network
  - Transactions are recorded in the DB(Ledger)
- The challenge:
  - How to map the controlflow program to a chaincode.
  - Make it generic.
  - How to interface actors to the chaincode (we want actors to affect state changes in the controlflow)

Petrinet to blockchain mapping



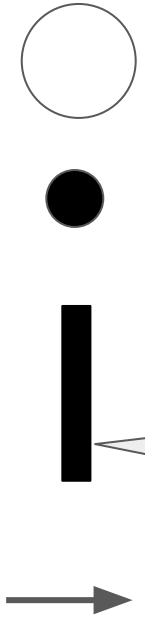
- A place receives is a placeholder for tokens.
- It is owned by a domain.
- Can be represented as an Asset.

Petrinet to blockchain mapping



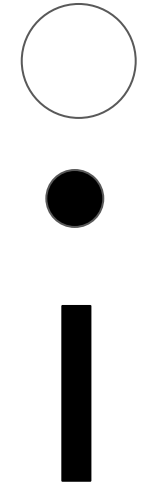
- Tokens are passed between places.
- They are owned by domains.
- They are represented as assets.
- Tokens change ownership when moved between places.
- As is with web tokens, tokens also represent authorization. A function can only execute if it has the correct tokens from the different domains.

Petrinet to blockchain mapping



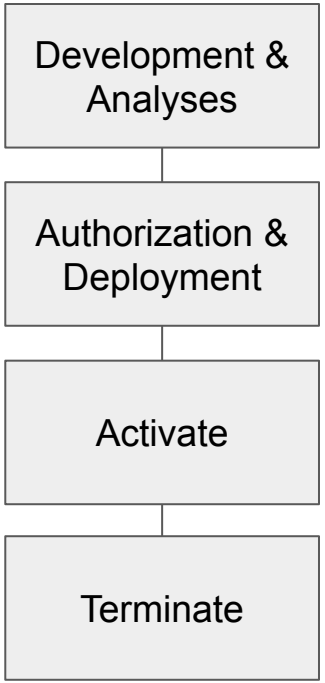
- Transitions are what move tokens between places.
- They are represented as an asset.
- They are owned by domains.
- They map to container functions.
- A transition fire implies a container function execution.

Petrinet to blockchain mapping



- Arrows show the control flow of the network.
- They indicate the required input tokens for a transition and the number of output tokens.
- A transition (container function) fires when the required input tokens are ready.

Petrinet life cycle



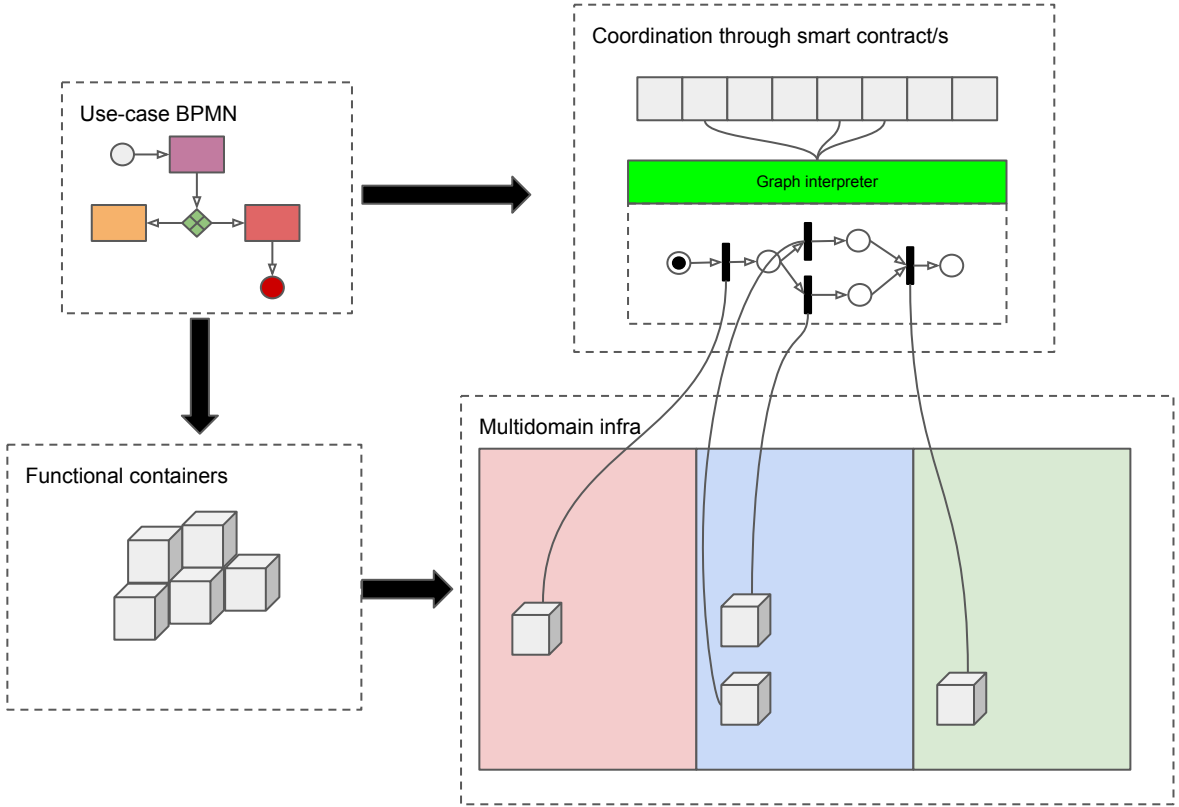
Develop the petrinets as ‘smart contracts’. Analyse petrinets. We can only deploy once to a blockchain.

Deployment needs authorization from multiple peers on the network. This will need an audit layer to authorize deployments.

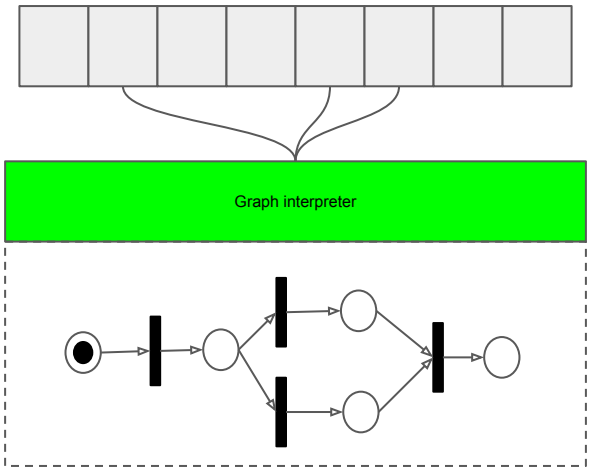
Once deployed it is in a start state. Moving from the start state activates the petrinet.

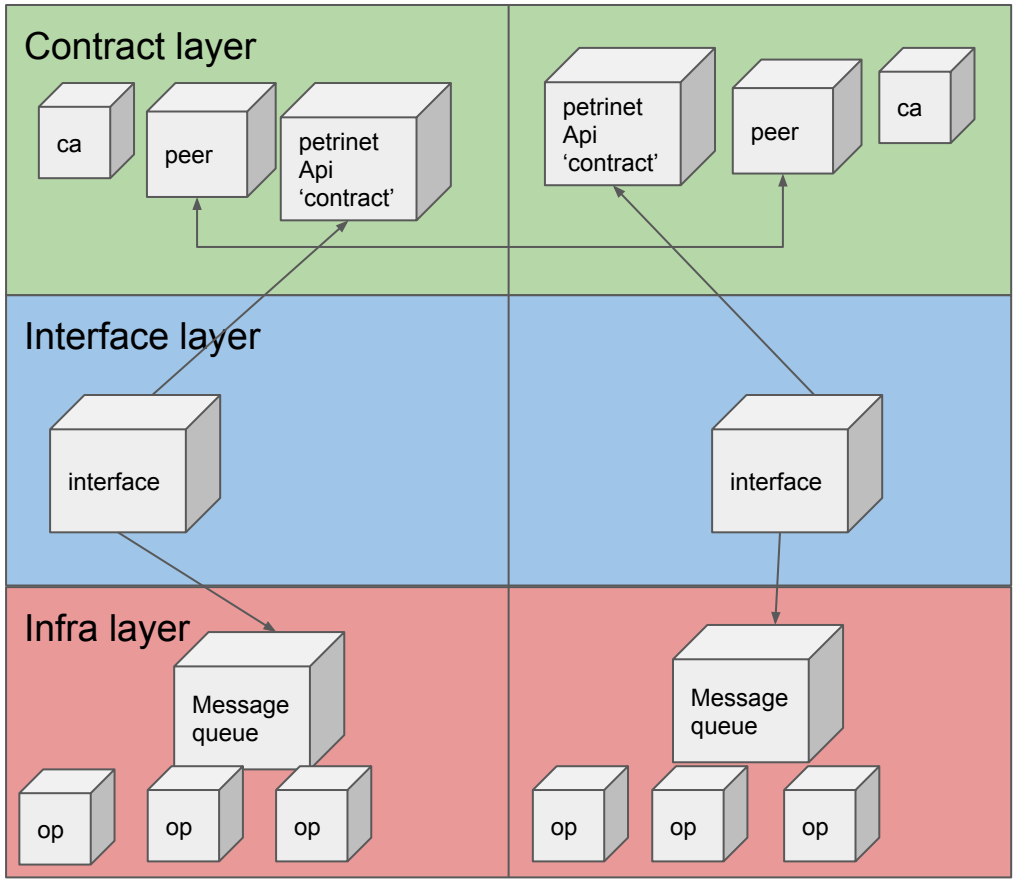
A petrinet can terminate it can not move to any other state.

Architecture



Architecture





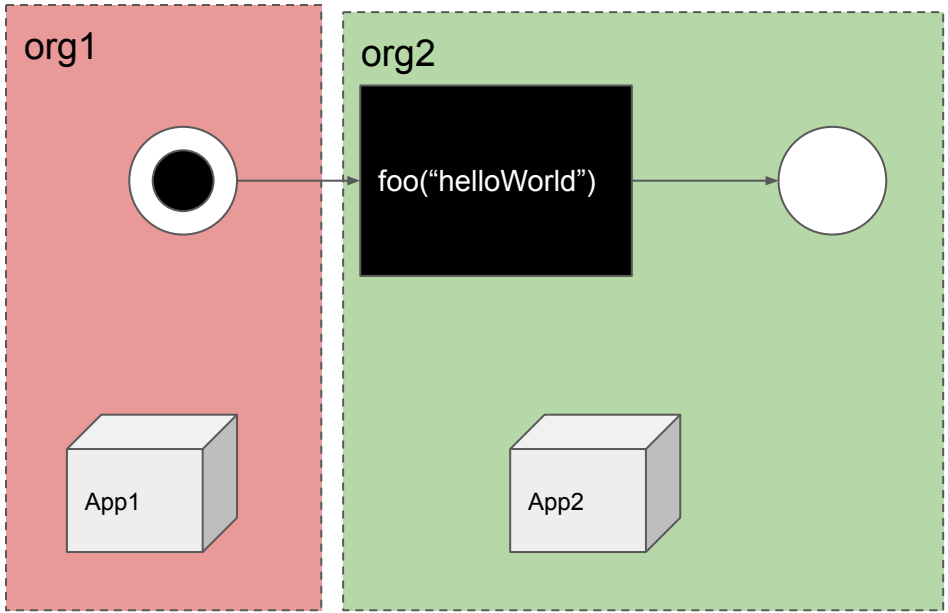
Petrinet contract API

- Create|Update|DeleteToken
  - Struct { id, issuer, owner, color }
  - Updates ledger
- Create|Update|DeletePlace
  - Struct { id, issuer, owner, tokens:[ ] }
  - Updates ledger
- Create|Update|DeleteTransition
  - Struct {id, issuer, owner, functionURI, state}
  - Updates ledger
- Create|Update|DeleteNet
  - Struct {id, issuer, owner, accepts:[], arcs: [ ] }
  - Updates ledger
- AcceptNet
  - Update net with domain keys that accept the petrinet.
- PutToken
  - Puts a token in a place **OR** transfers token to place owner **AND** generate event
  - Checks if transition is ready to fire
  - Generate fire event

Interface application

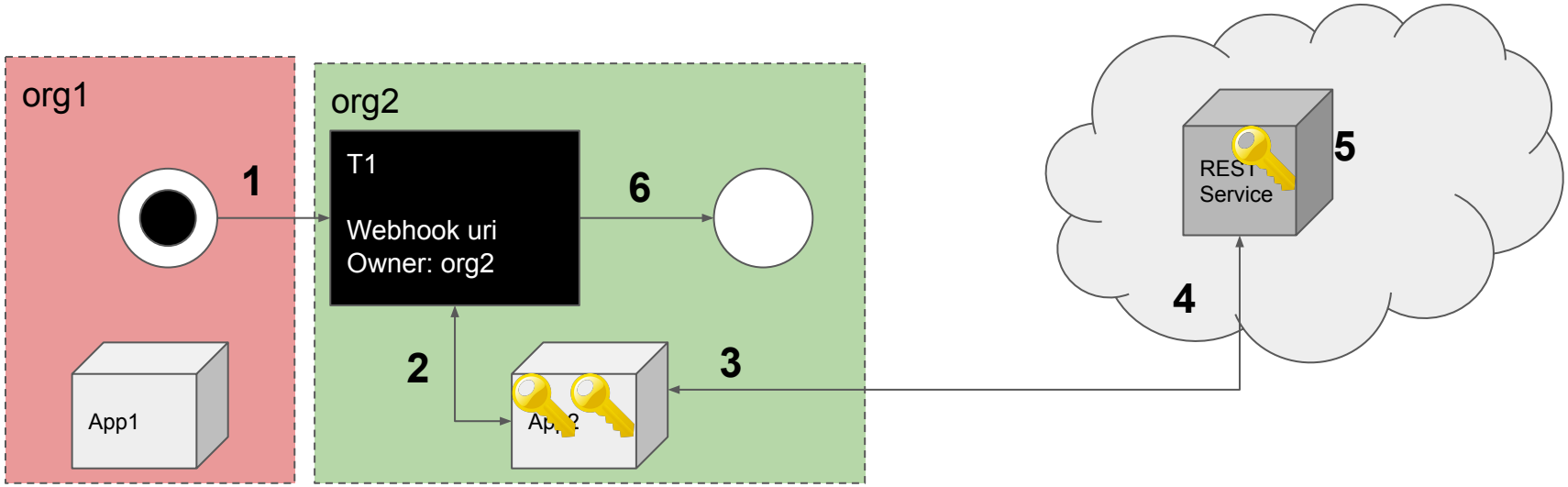
- Build wallet
  - User keys from domain CA
  - Enroll user as part of domain
- Connect to blockchain node
- Creates domain transitions, places and tokens.
- Creates petrinets
- Moves tokens, updates places.
- Listens for events (transition firing)
  - Call infrastructure **operations** through message queue
  - Move tokens

Simple Petrinet



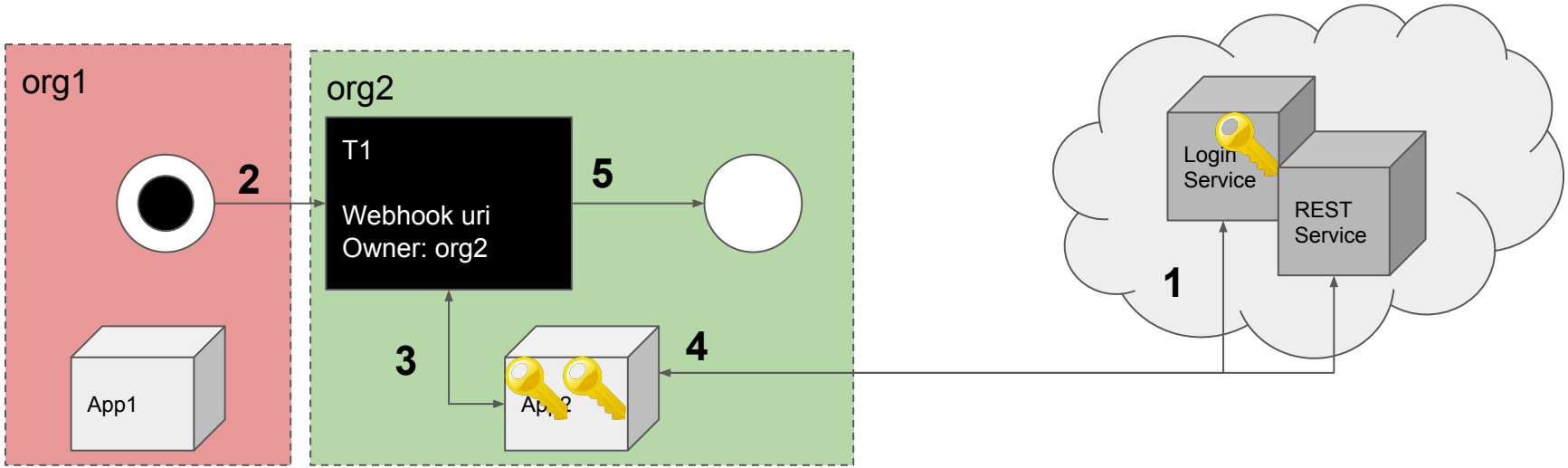
- **App1**: CreateToken, CreatePlace
- **App2**: CreateTransition, CreatePlace
- **App1**: CreateNet, event
- **App2**: AcceptNet
- **App1**: PutToken
- **App2**: Listen for *Fire* event
- **App2**: Message Infra to run foo()
- **App2**: Wait response
- **App2**: PutToken

Generalized Webhook interface - 1



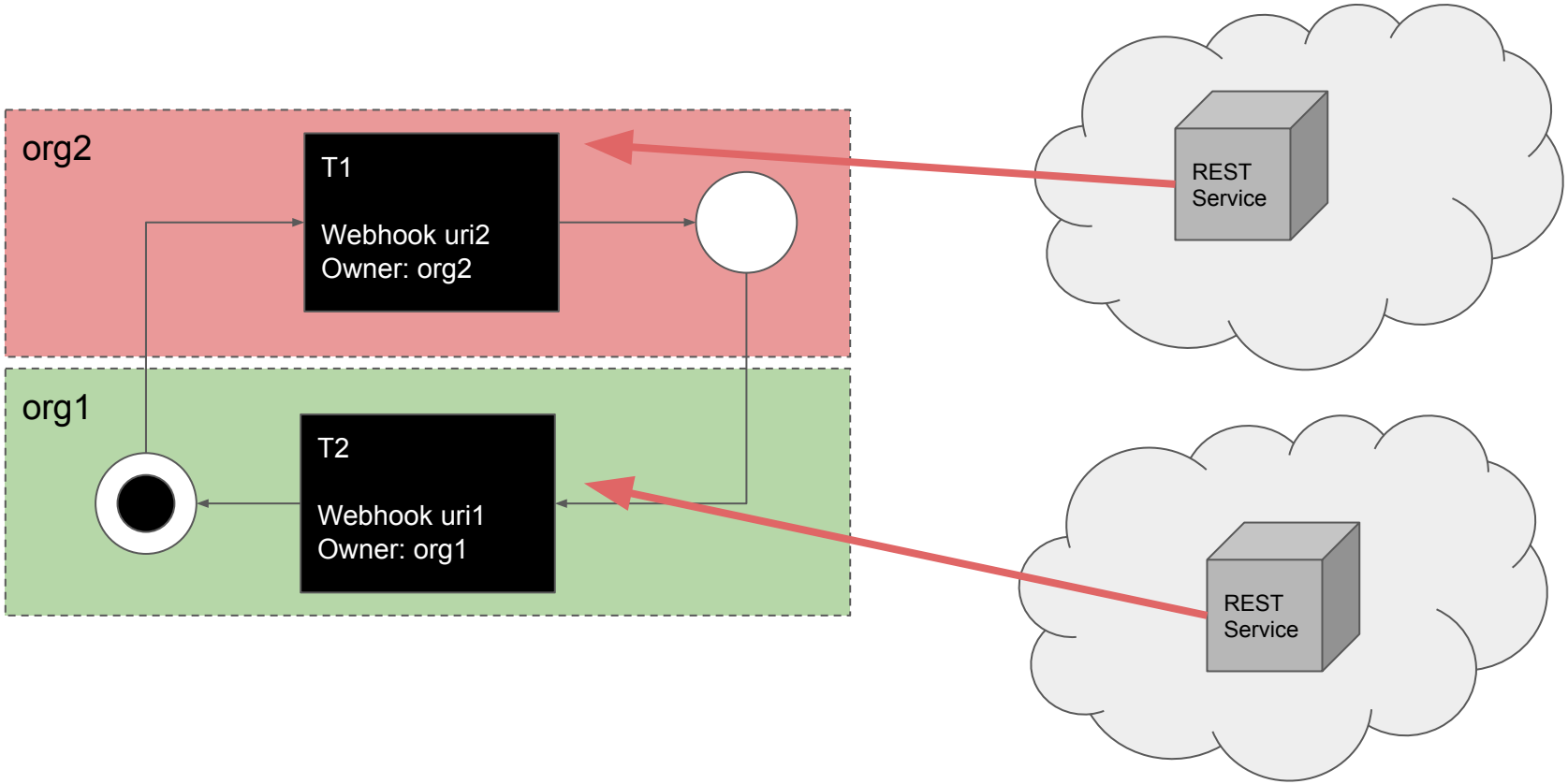
1. **T1** fires, event generated on blockchain.
2. **App2** reads event; is owner of **T1** (has keys).
3. Generates a JSON web token using owner private key to sign the web token.
4. Calls the uri encoded in **T1** asset, passing the web token in the header.
5. The trusts **org2** public key; authorizes call from **App2** by validating web token.
6. **App2** changes petrinet state.

Generalized Webhook interface - 2

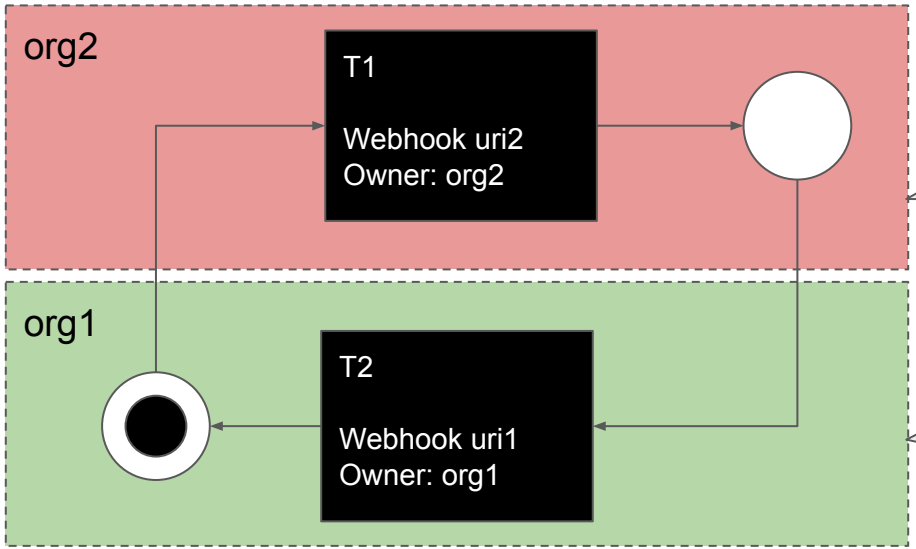


1. App2 obtains a JWT (JSON web token) from remote service; saves it locally.
2. **T1** fires; event generated on blockchain.
3. App2 read event; is owner of T1 asset.
4. Matches JWT to T1; calls service with JWT in header.
5. Changes petrinet state.

Webhook loop example



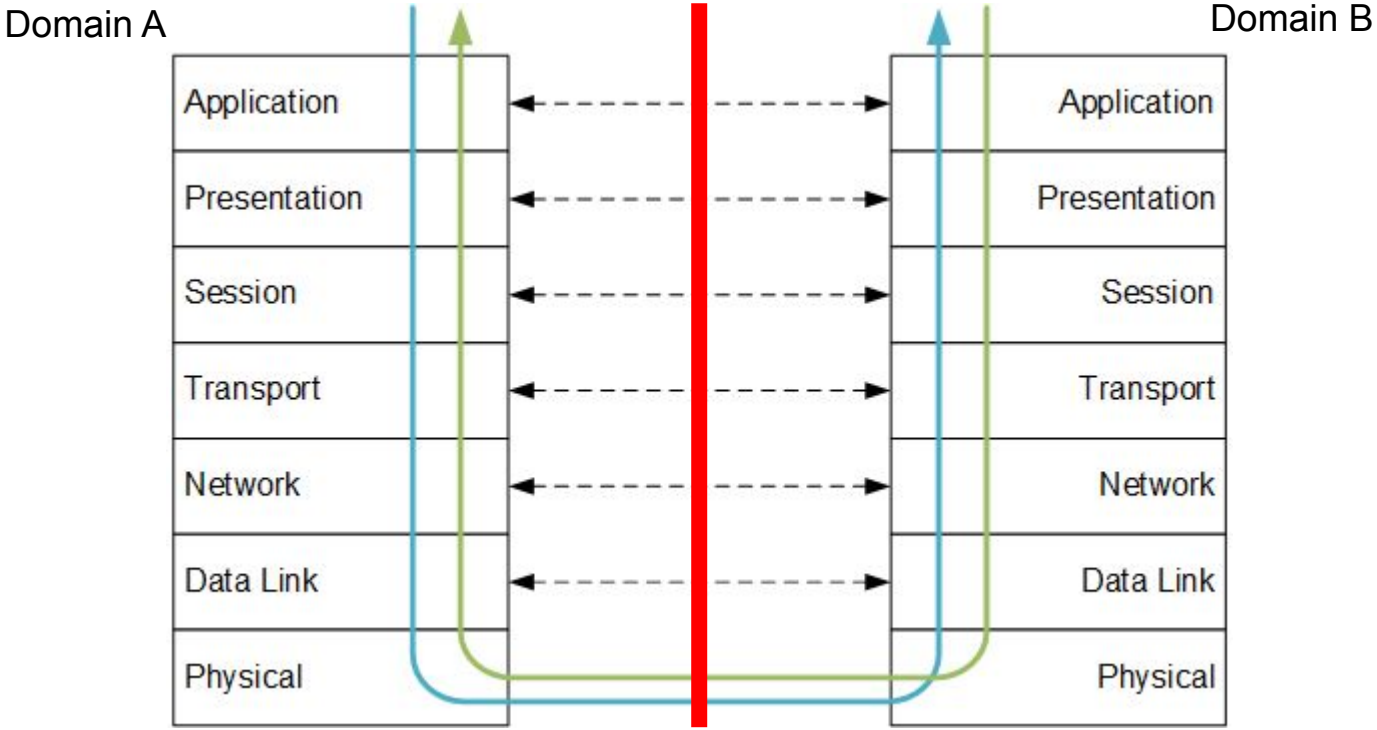
Webhook loop example



1. CREATE Place asset.
2. CREATE Transition asset.
3. LISTEN for events.
4. ACCEPT Petrinet.
5. EXEC T1.
6. CREATE Token
7. MOVE Token.
8. GOTO 3

1. CREATE Place asset.
2. CREATE Transition asset.
3. CREATE Token asset.
4. CREATE Petrinet.
5. MOVE initial Token.
6. LISTEN for events.
7. EXEC T2.
8. CREATE and MOVE Token.
9. GOTO 6

Data-specific fullstack multi-domain setup



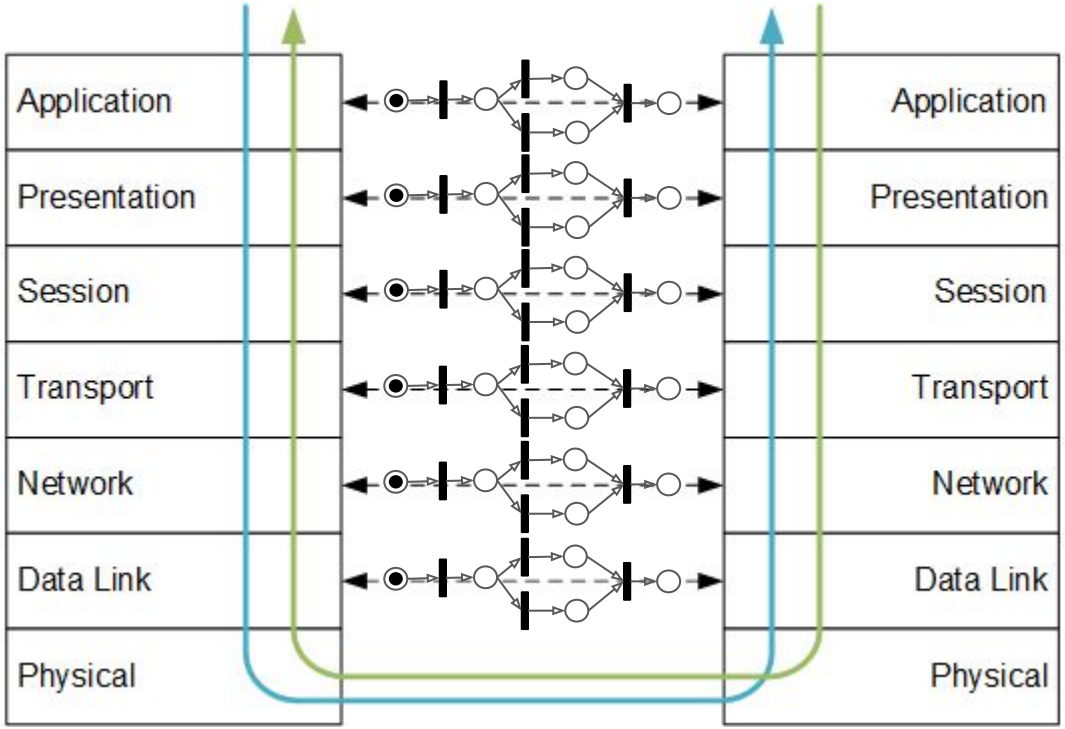
Use-cases - bottom stack

- Programmable networks (multi-domain)
  - Route setup (source routing)
    - Control PCEs in different domains
  - Network Function Chaining
    - Setup chains that span multiple domains
  - Multi-domain P4 setups.
- Overlays
  - Application-specific VPNs
    - Application events that trigger underlying connection setup.
  - P2p setups.

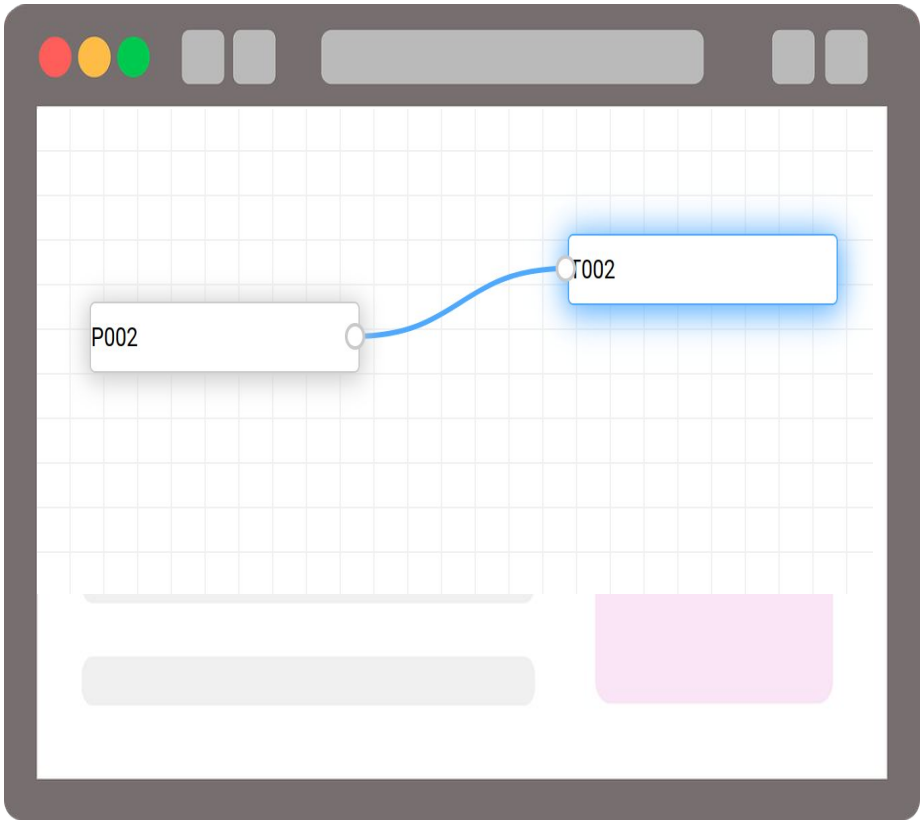
Use-cases - top stack

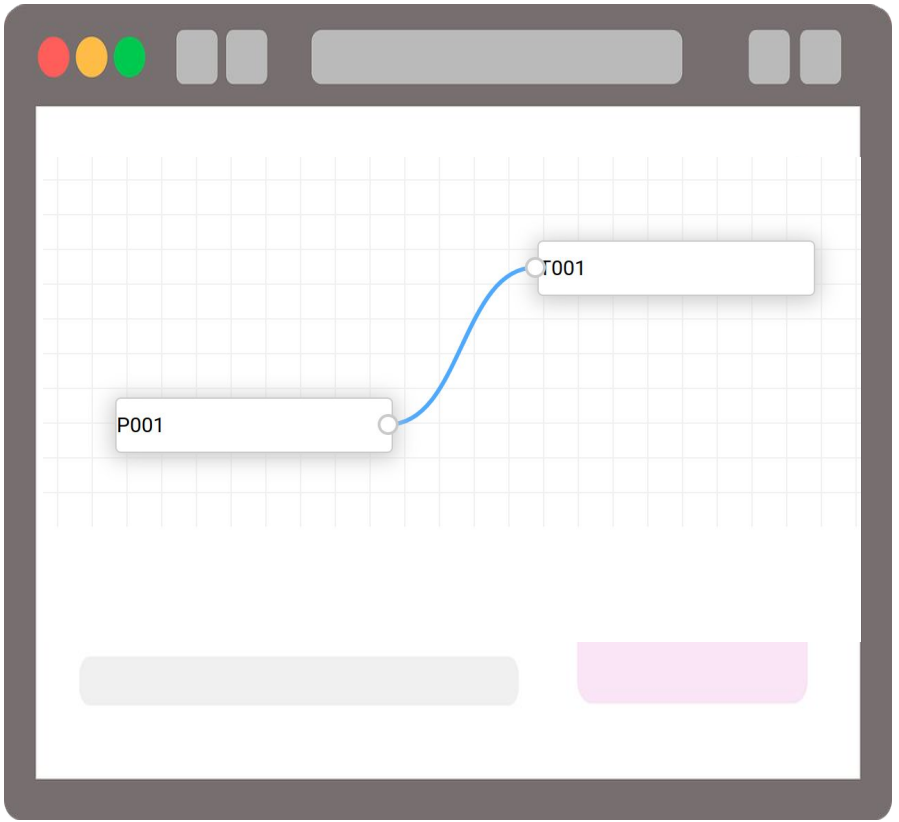
- Services
  - Application specific network services and endpoint parameter tuning.
    - Gridftp
    - UTD
    - FDT
  - Partner backend integrations
    - Control how bilateral agreements translate to multi-backend access.
  - QoS agreements
    - Scaling services to meet QoS
- Applications
  - DTNs
    - Type of data prioritization between DTNs.
    - Cache reservation setup on DTNs
  - Data quality agreements
    - What is transfered? Data specific quality checks E.g. time-series data quality could be sampling rate etc.

Data-specific fullstack multi-domain setup

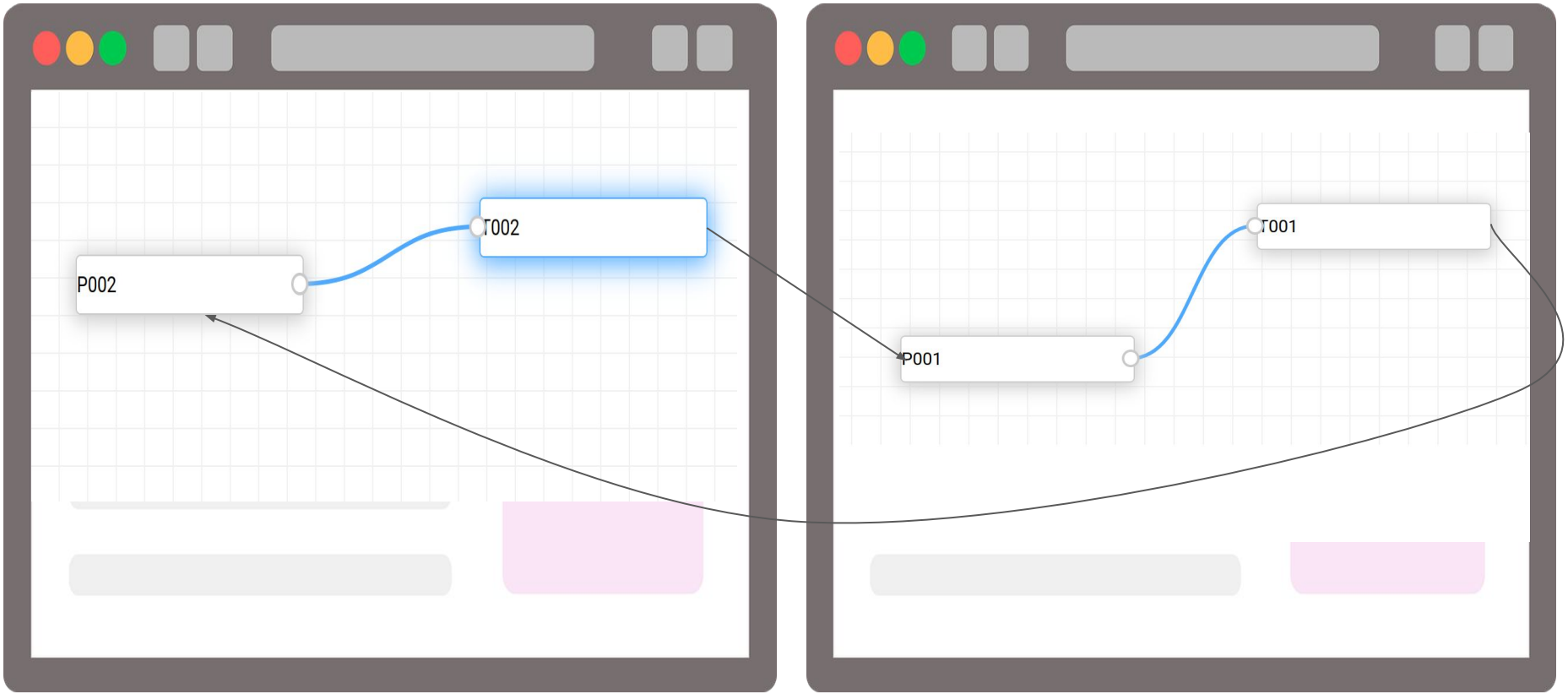


UI - connecting sub petrinets





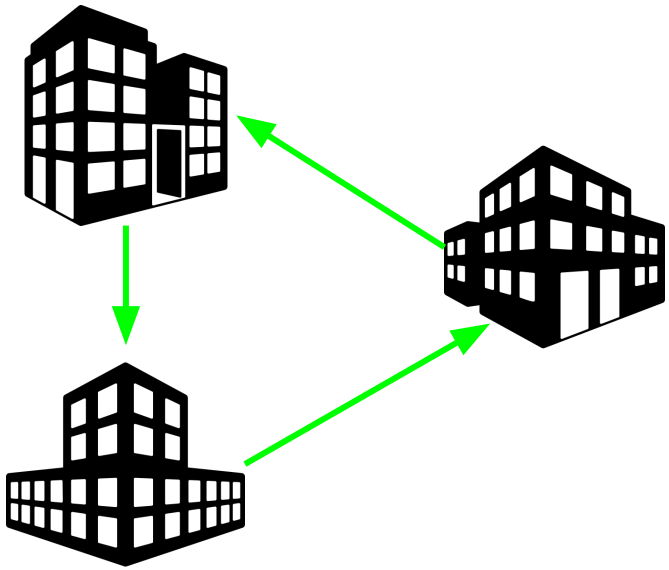
UI - connecting sub petrinets



DDOS usecase

- Simulate a network using Kathara emulator which uses docker containers as nodes in the network.
- Create a petrinet to describe the scenario that “when an attack happens in domain A, notify domains B,C,D”
- Have Kathara router containers listen for the notifications and block IPs of attacking nodes so that attack is blocked close to the source.
- Undo the block.

Multi-domain application coordination using Petrinets in smart contracts



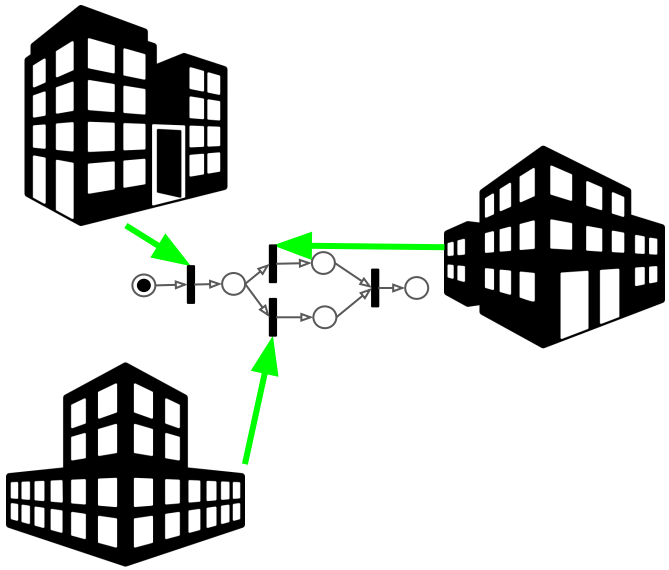
Motivation:

Multi-domain applications are characterized by applications such as workflows that cross domain boundaries.

The motivation for such applications is in mutual benefit for all parties to collaborate. For example airline industries, healthcare, smart cities.

The new set of challenges that this setup introduces revolve mainly around enforcement of agreed multilateral contracts and minimizing risks due to exposure.

In this work we propose to encode the application agreement as a smart contract using Petrinet as a model to track state changes.



Motivation:

Multi-domain applications are characterized by applications such as workflows that cross domain boundaries.

The motivation for such applications is in mutual benefit for all parties to collaborate. For example airline industries, healthcare, smart cities.

The new set of challenges that this setup introduces revolve mainly around enforcement of agreed multilateral contracts and minimizing risks due to exposure.

In this work we propose to encode the application agreement as a smart contract using Petrinet as a model to track state changes.

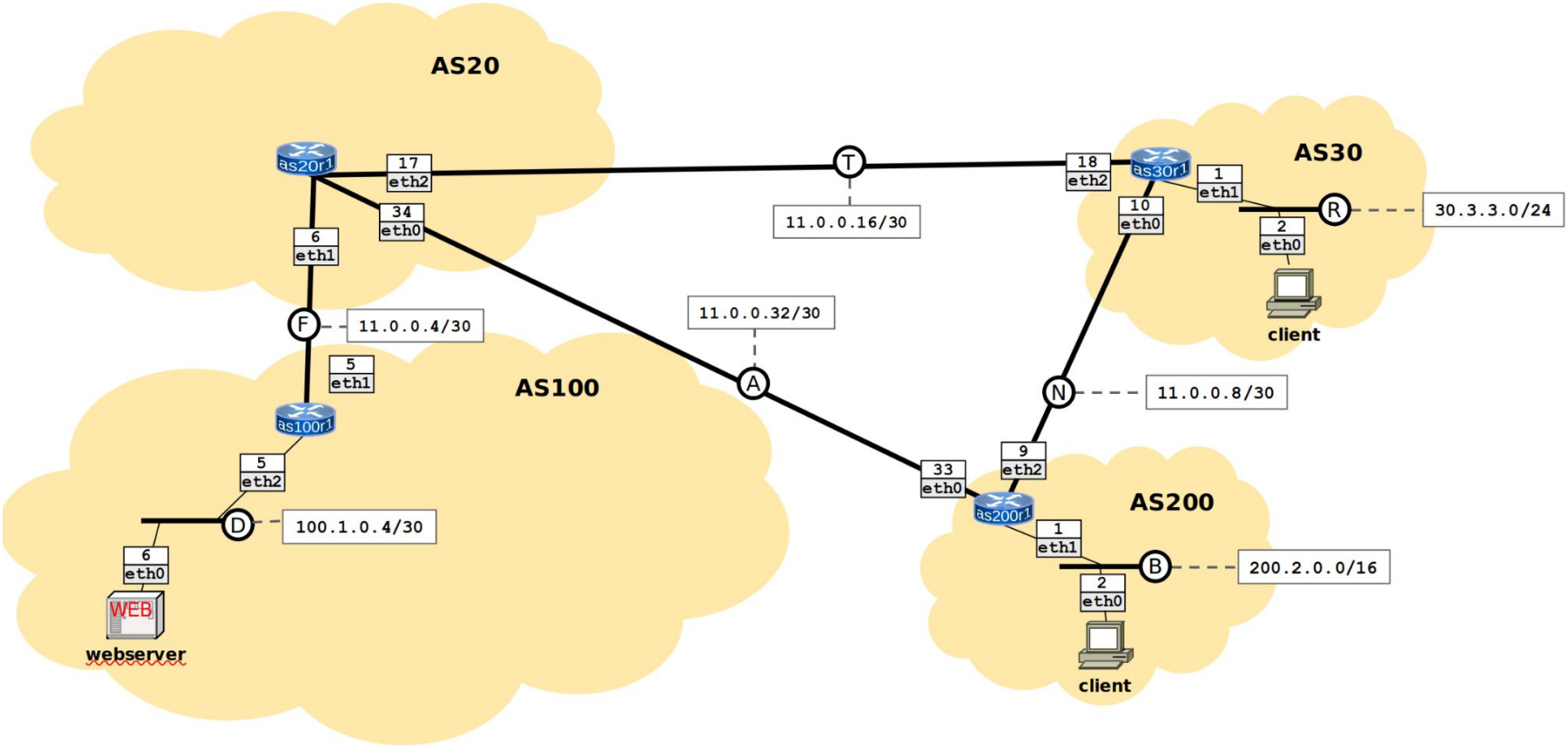
Demo setup

- Collaboration between Internet domains in a DOS attack.
- We emulate a simple Internet with 4 ASs and assign them to 3 domains.
- We create a Hyperledger across the 3 domains.
- The application says that:

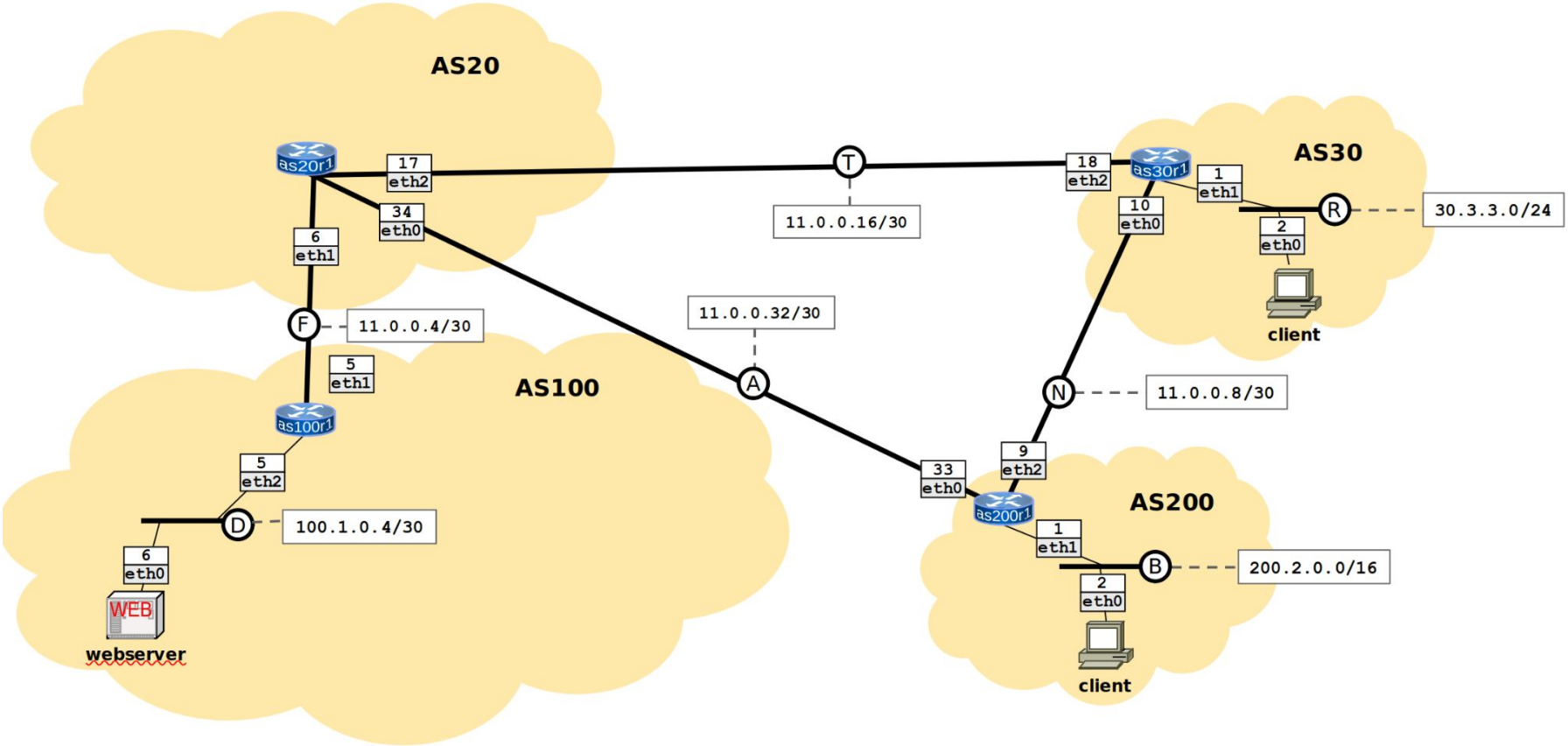
  "*If any domain detects a DOS it will inform the others.The others will inturn block the IP on their side.*"

- This is encoded as a Petrinet using smart contracts.

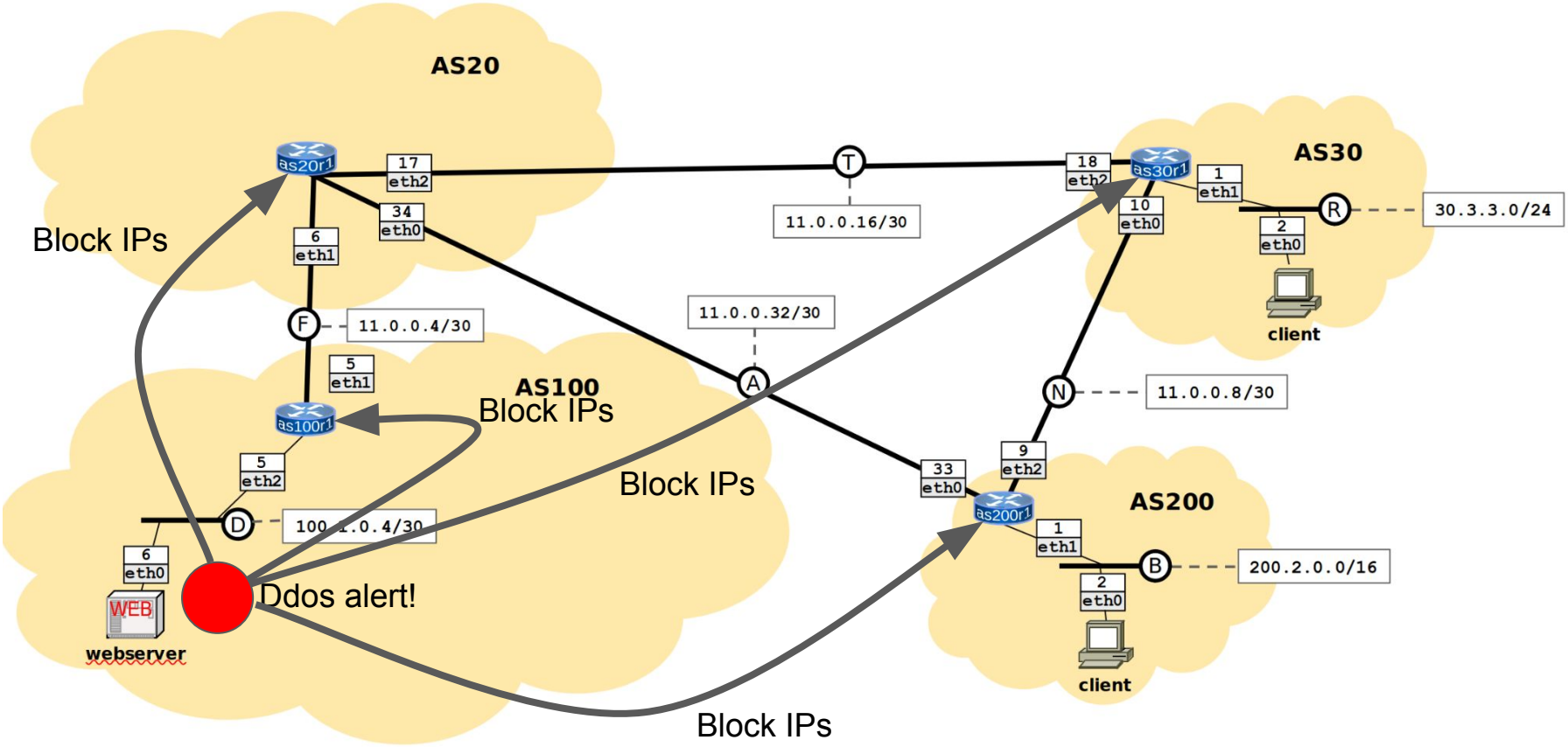
Network setup using Kathara emulator.
Setup involves 4 ASs, a webserver and 2 clients.


AS20
AS30
AS100
AS200
as20r1
as30r1
as100r1
as200r1
17
eth2
34
eth0
6
eth1
18
eth2
1
eth1
10
eth0
2
eth0
5
eth1
5
eth2
6
eth0
9
eth2
33
eth0
1
eth1
2
eth0
T
R
F
A
N
D
B
11.0.0.16/30
30.3.3.0/24
11.0.0.4/30
11.0.0.32/30
11.0.0.8/30
100.1.0.4/30
200.2.0.0/16
client
client
WEB
webserver

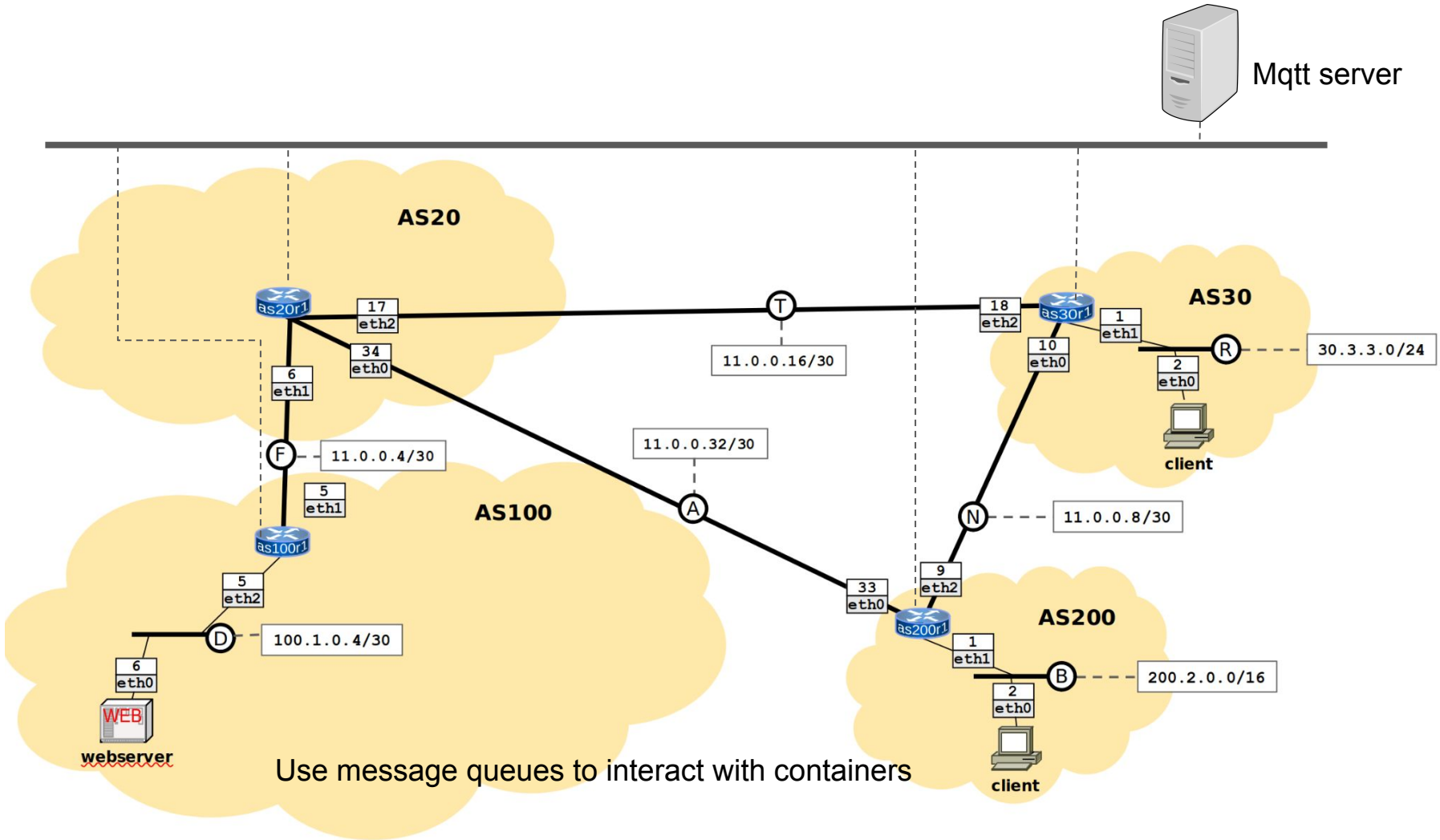
Devices are emulated as containers.
Collision domains as separate networks.

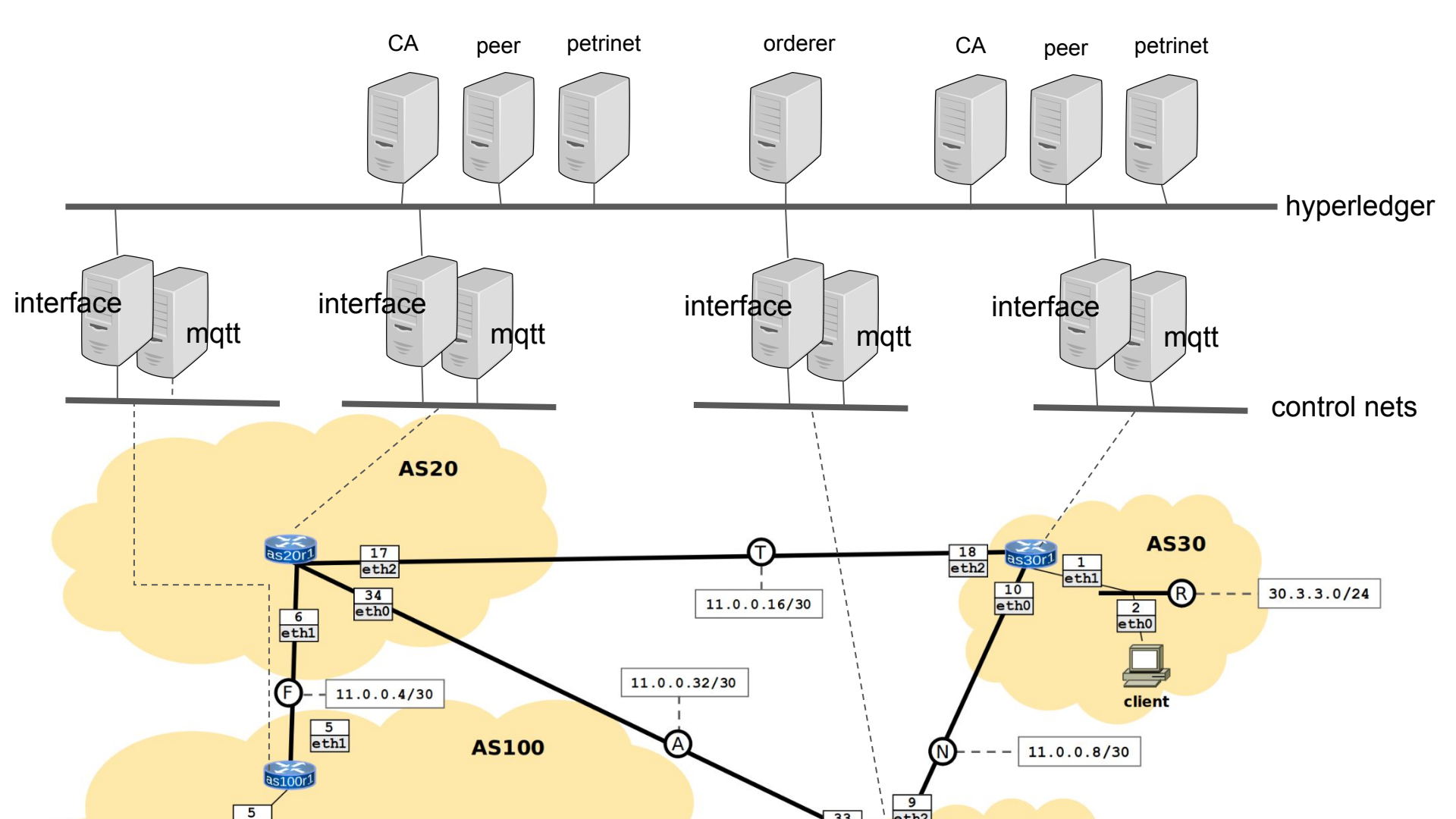

AS20
AS30
AS100
AS200
as20r1
as30r1
as100r1
as200r1
17
eth2
34
eth0
6
eth1
18
eth2
1
eth1
10
eth0
2
eth0
5
eth1
5
eth2
6
eth0
9
eth2
33
eth0
1
eth1
2
eth0
T
R
F
A
N
D
B
11.0.0.16/30
30.3.3.0/24
11.0.0.4/30
11.0.0.32/30
11.0.0.8/30
100.1.0.4/30
200.2.0.0/16
client
client
WEB
webserver

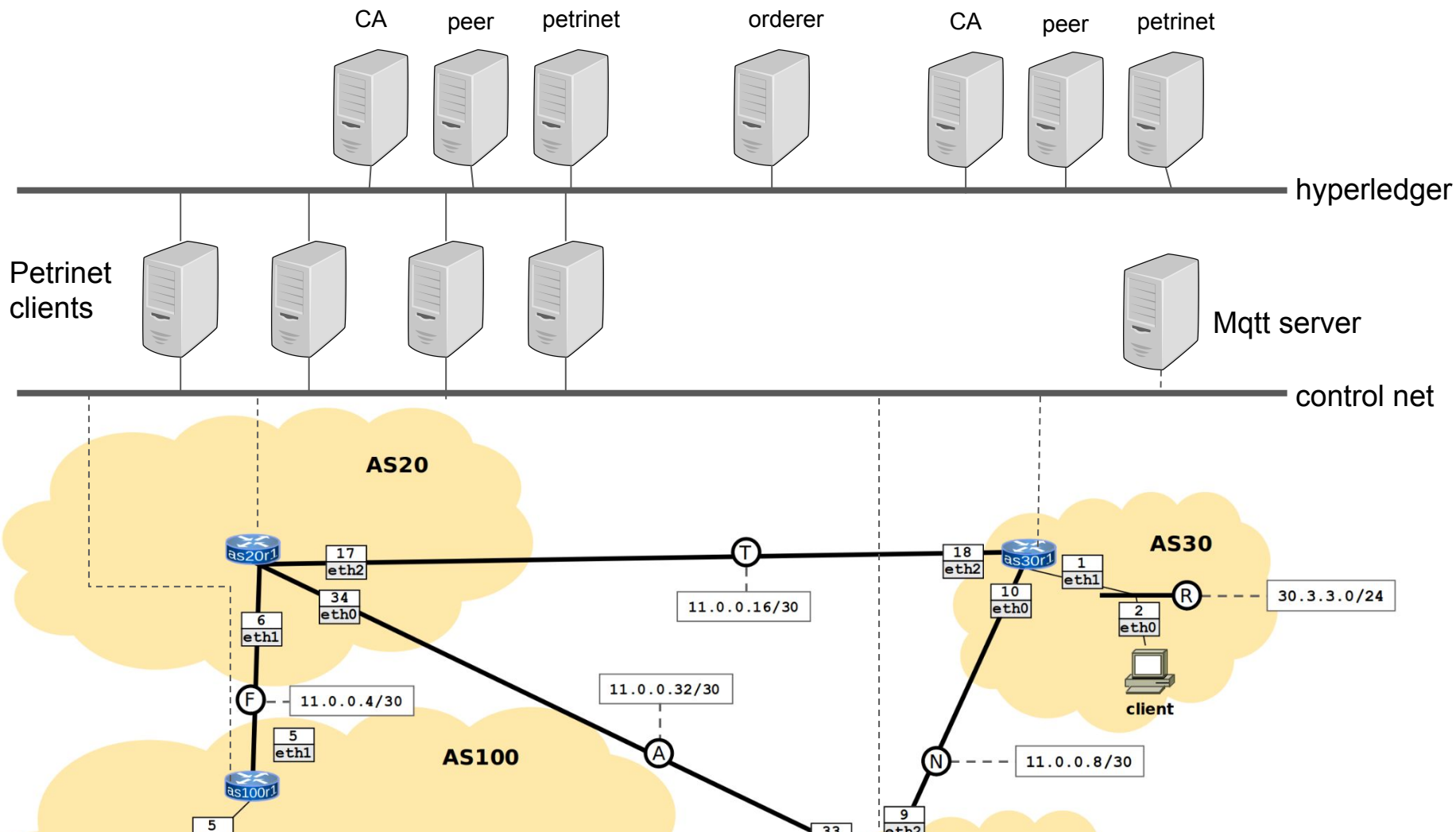
Multi-domain scenario:
Collaborative DOS response,
The web server is attacked by a client and AS
Collaborate to block the IP



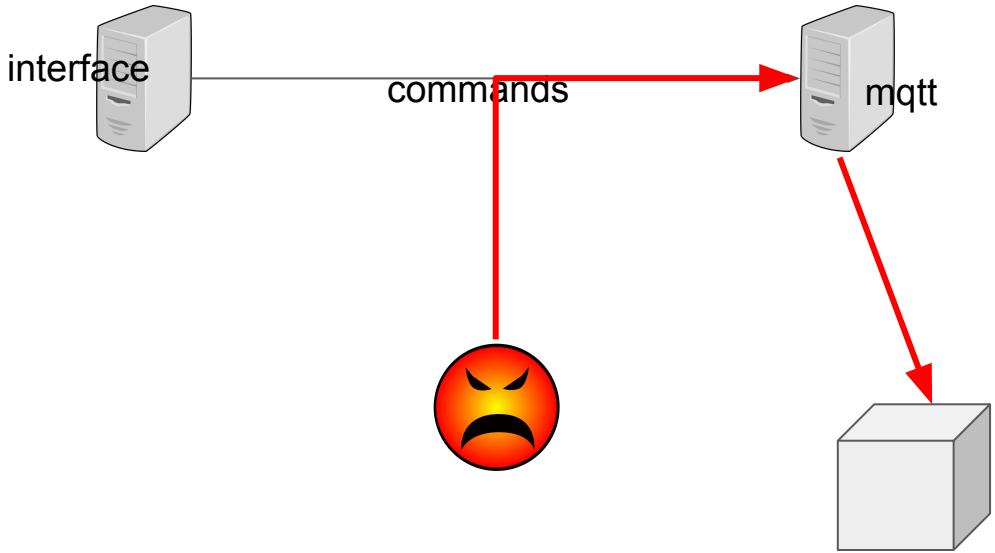
Attack scenario: web server is being attacked.
Notify other domains so they can block the IPs


Mqtt server
AS20
AS30
AS100
AS200
as20r1
as30r1
as100r1
as200r1
17
eth2
34
eth0
6
eth1
18
eth2
10
eth0
1
eth1
2
eth0
5
eth1
5
eth2
6
eth0
9
eth2
33
eth0
1
eth1
2
eth0
T
R
F
A
N
D
B
11.0.0.16/30
30.3.3.0/24
11.0.0.4/30
11.0.0.32/30
11.0.0.8/30
100.1.0.4/30
200.2.0.0/16
client
client
WEB
webserver
Use message queues to interact with containers

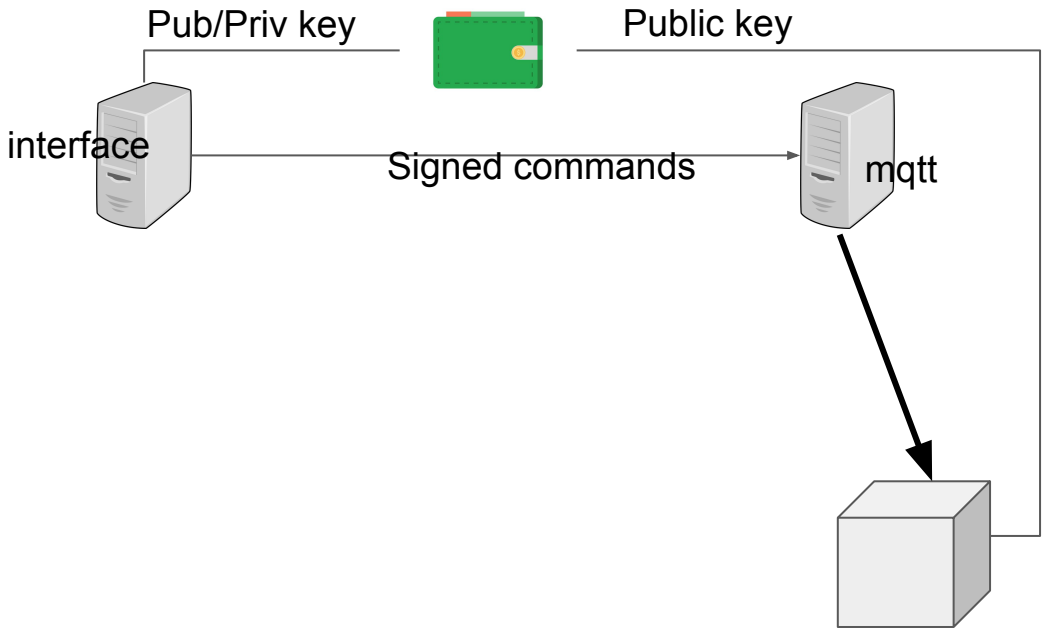

CA
peer
petrinet
orderer
CA
peer
petrinet
hyperledger
interface
mqtt
interface
mqtt
interface
mqtt
interface
mqtt
control nets
AS20
AS30
AS100
as20r1
as30r1
as100r1
17
eth2
34
eth0
6
eth1
18
eth2
10
eth0
1
eth1
2
eth0
5
eth1
5
T
11.0.0.16/30
R
30.3.3.0/24
client
F
11.0.0.4/30
A
11.0.0.32/30
N
11.0.0.8/30
9


CA
peer
petrinet
orderer
CA
peer
petrinet
hyperledger
Petrinet clients
Mqtt server
control net
AS20
AS30
AS100
as20r1
as30r1
as100r1
17 eth2
34 eth0
6 eth1
18 eth2
10 eth0
1 eth1
2 eth0
5 eth1
9 eth2
5
T
F
A
N
R
11.0.0.16/30
11.0.0.4/30
11.0.0.32/30
11.0.0.8/30
30.3.3.0/24
client

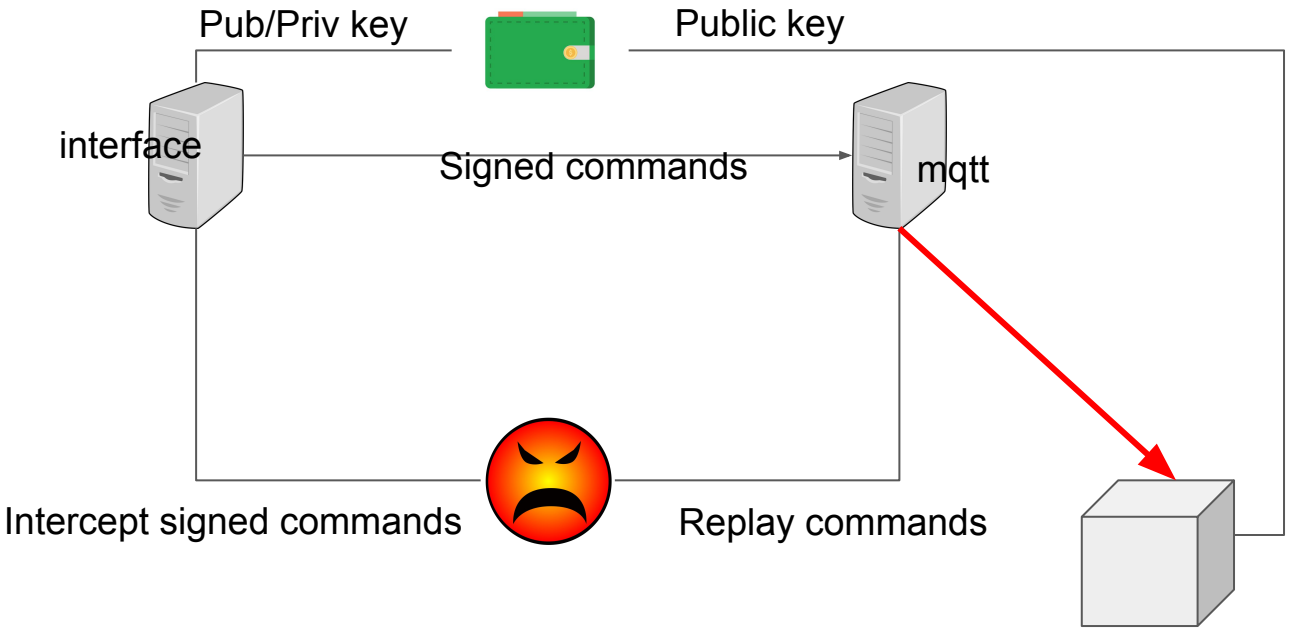
Interface to mqtt security



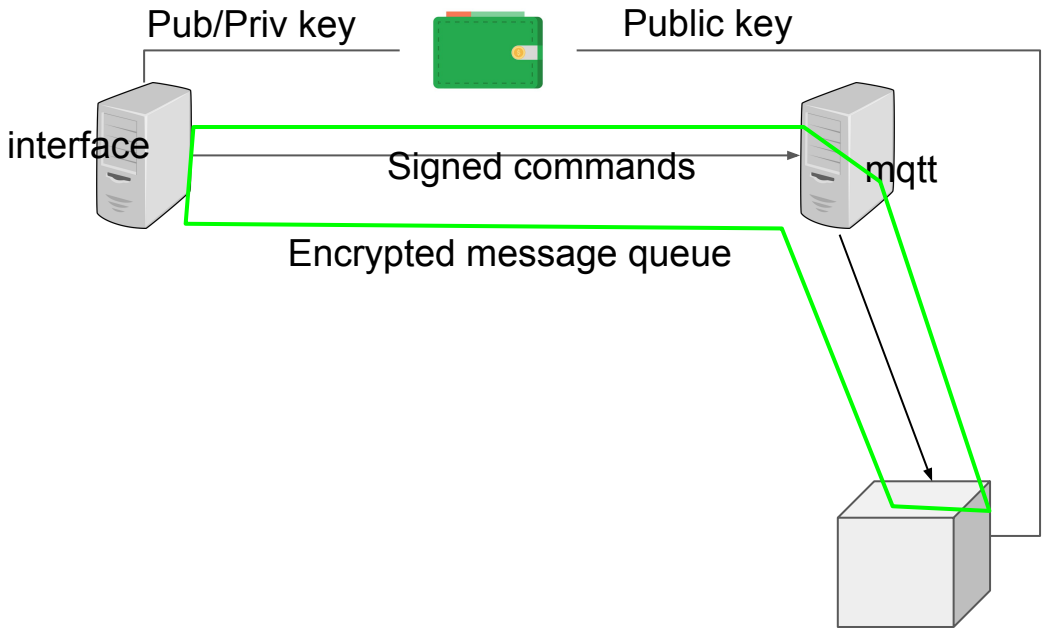
Interface to mqtt security



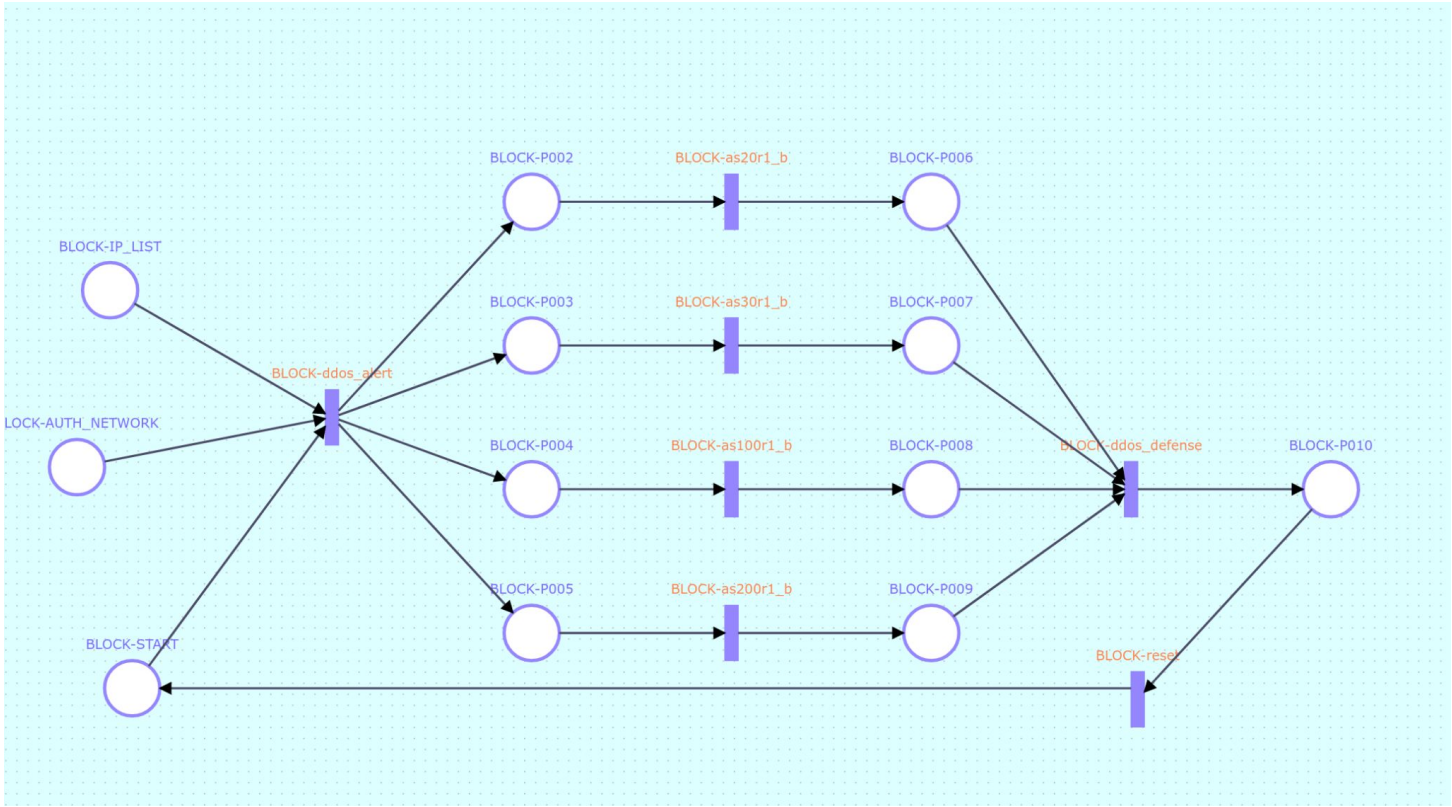
Interface to mqtt security



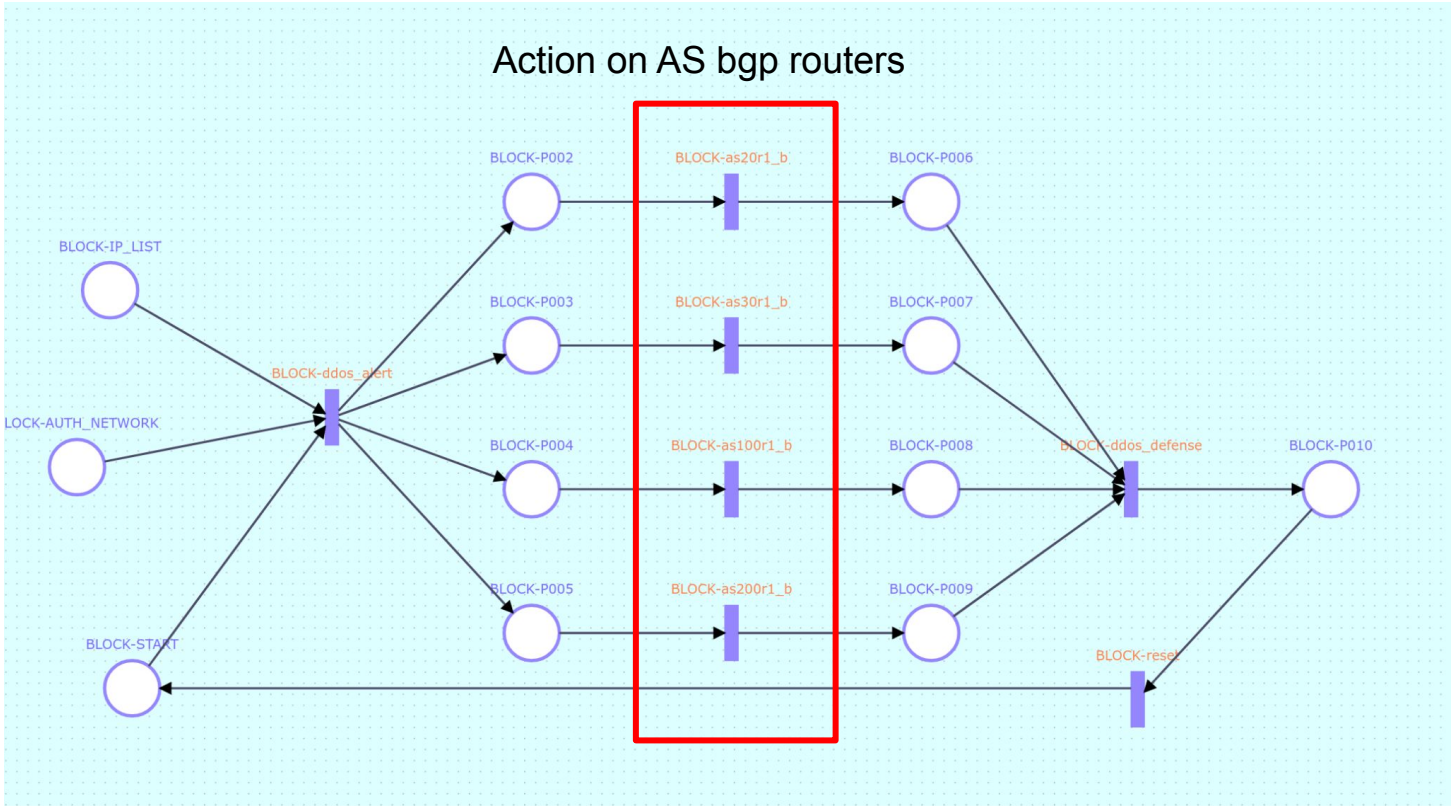
Interface to mqtt security



Demo ddos defence contract



Demo ddos defence contract



Demo ddos defence contract