Access Control Policies Across Abstraction Layers

Lorenzo Ceragioli
Università di Pisa

Supervisors

Pierpaolo Degano
Università di Pisa

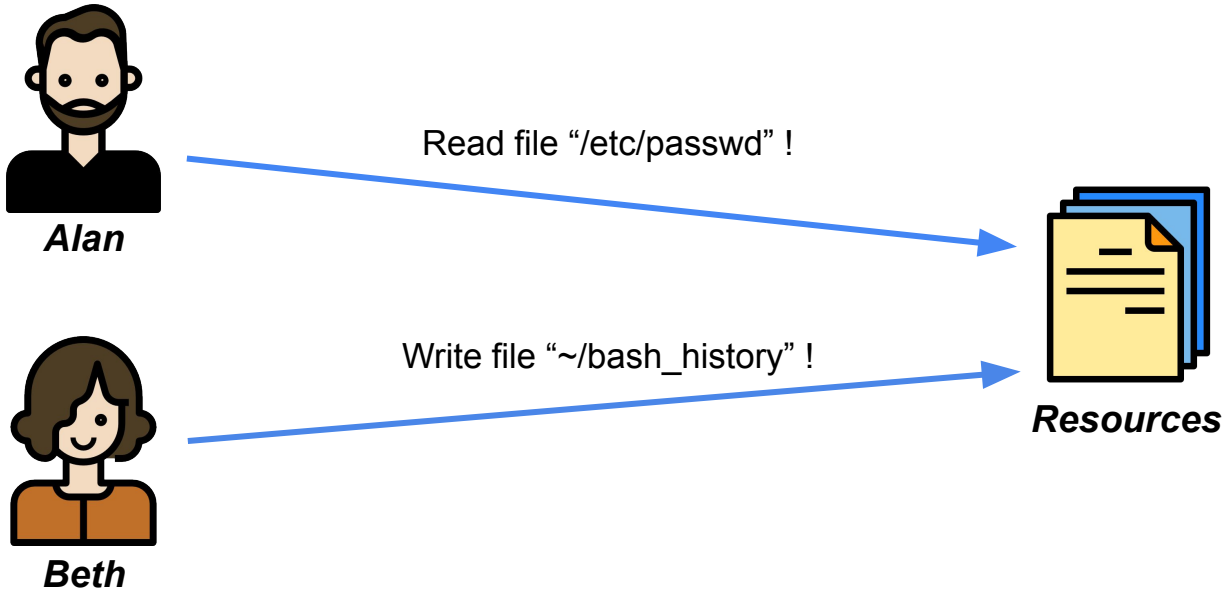
Letterio Galletta
IMT, Lucca

Referees

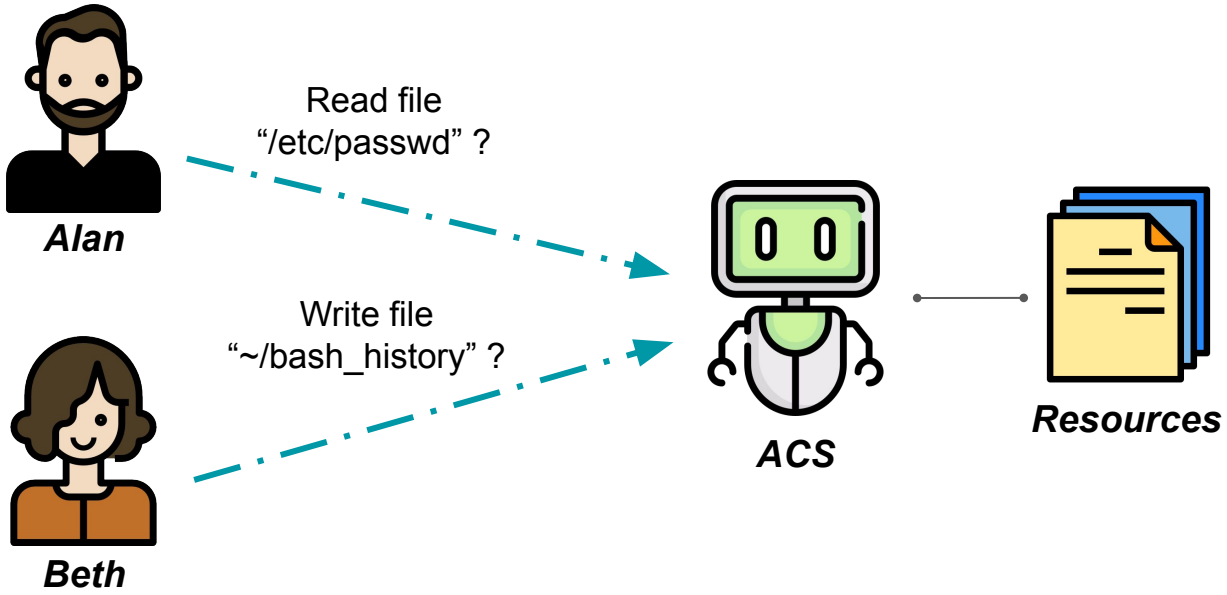
David Basin
ETH Zurich

Rosario Pugliese
Università degli Studi di Firenze

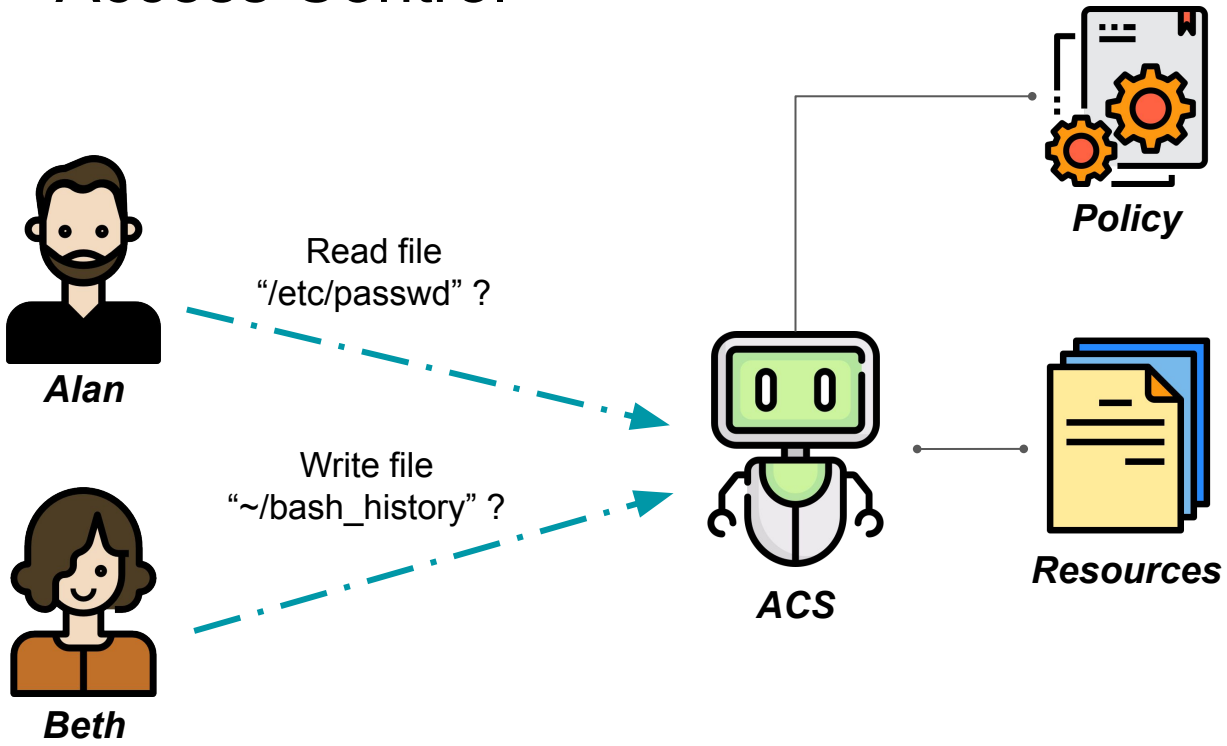
Recap - Access Control



Recap - Access Control



Recap - Access Control



Access Control - Where?

- **Networks**: Firewalls
- **Web**: XACML
- **Social Networks**: ReBAC
- **Operating Systems**: ACLs, SELinux
- **Medium - Large Enterprises**: RBAC
- ...

Tasks

- **Collecting Requirements**
- **Defining a Specifications**
- **Coding the Configuration**
- **Verification and Analysis**
- **Testing**
- **Update (specifications and Configuration)**

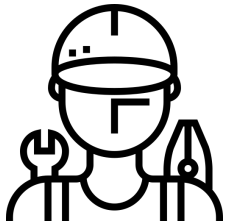
Tasks - Abstraction Layers

- **Collecting Requirements**
- **Defining a Specifications**
- **Coding the Configuration**
- **Verification and Analysis**
- **Testing**
- **Update (specifications and Configuration)**

High Level

Specification Language
Easy to Write & Read
Cannot Run

Low Level



Configuration Language
Hard to Write & Read
Actually Run

Problems

Manual coding is

- error prone - misunderstanding
  - of the specifications – e.g. ignore corner cases
  - of the configuration – e.g. low level intricacy
- expensive

Configurations and Specifications may change over time

Specifications may be impossible to implement

… we propose different solutions for mitigating these problems on three contexts

Two-way Translation Based Solution

- Compilation & Decompilation
  - Grant coherence
  - Automatise Coding & Analysis
- Support configuration and specification changes
- Low Level configuration is automatically produced, but can also be modified by hand

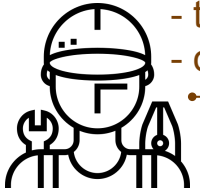
- specify
- verify
- update

High Level Language

compile

project limitations

decompile



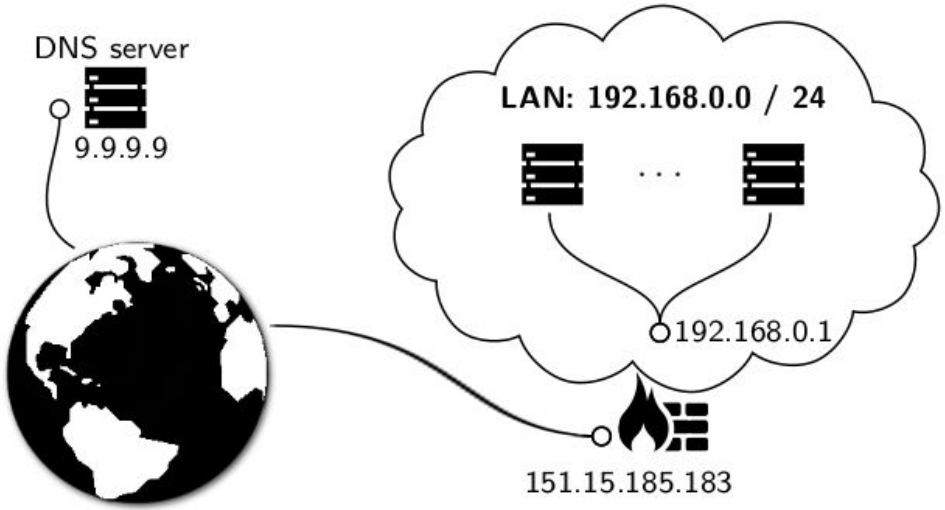
- tune
- test
- optimize

Low Level Language

Firewalls

On boundaries of the networks, **filter** and **translate** packets (NAT)

Different low level languages (*iptables*, *pf*, *ipfw*). Difficult to read and write, with low level details like shadowing and tags



"Connection from internal hosts to a DNS Server are redirected to 9.9.9.9"

FWS/F2F

- specify
- verify
- update

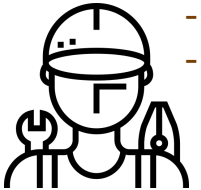
FWQL

- Function over IP packets
- SQL like interface

compile

project limitations

decompile



- tune
- test
- optimize

Firewall Languages

- iptables, ipfw, pf, …
- tag, shadowing, …

FWS/F2F

- specify
- verify
- update

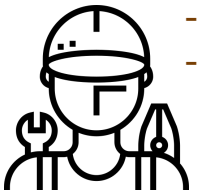
FWQL

- Function over IP packets
- SQL like interface

compile

project limitations

decompile



- tune
- test
- optimize

Firewall Languages

- iptables, ipfw, pf, …
- tag, shadowing, …

"Connection from internal hosts to a DNS Server are redirected to 9.9.9.9"

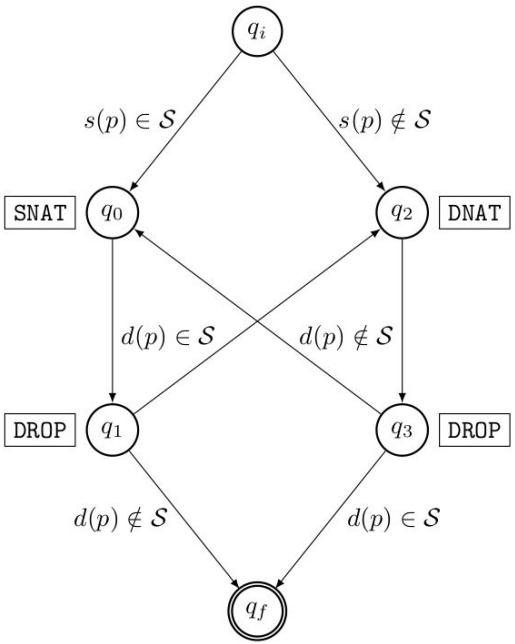
```
update t_dst = 9.9.9.9 in TAB where
     ( ( srcIp = Internal)
          and dstPort = DNS )
```

```
*nat
-A PREROUTING -p udp -s 192.168.0.0/24
     -- dport 53 -j DNAT -- to 9.9.9.9

*filter
-A FORWARD -m state -- state
     ESTABLISHED -j ACCEPT
-A INPUT -m state -- state
     ESTABLISHED -j ACCEPT
-A FORWARD -p udp -s 192.168.0.0/24
     --dport 53 -j ACCEPT
-A INPUT -p udp -s 192.168.0.0/24
     --dport 53 -j ACCEPT
```

Intermediate Firewall Configuration Language - IFCL

System Evaluation Algorithm



Configuration

Ruleset: list of pairs (Predicate, Action)

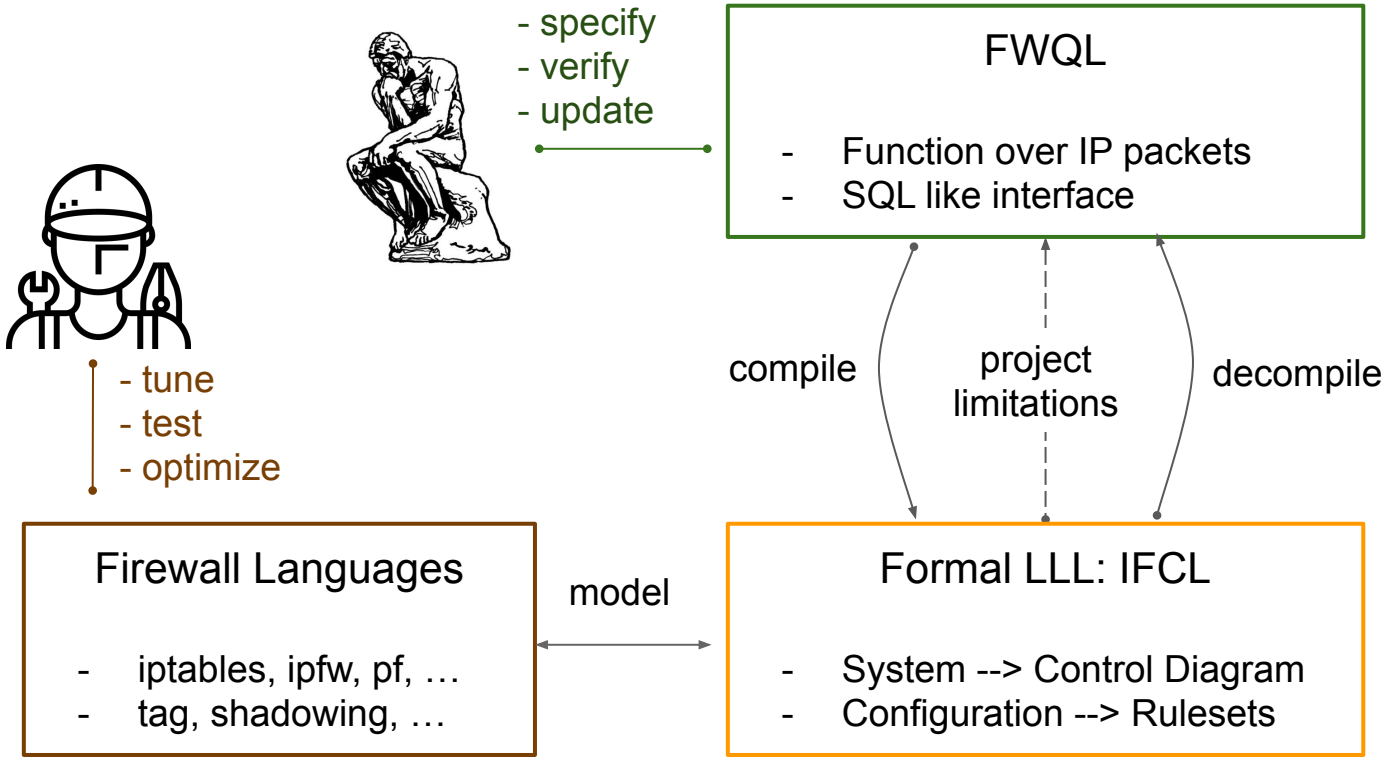
Action in

ACCEPT	RETURN
DROP	NAT(n_d, n_s)
CALL(R)	MARK(m)
GOTO(R)	CHECK-STATE(X)

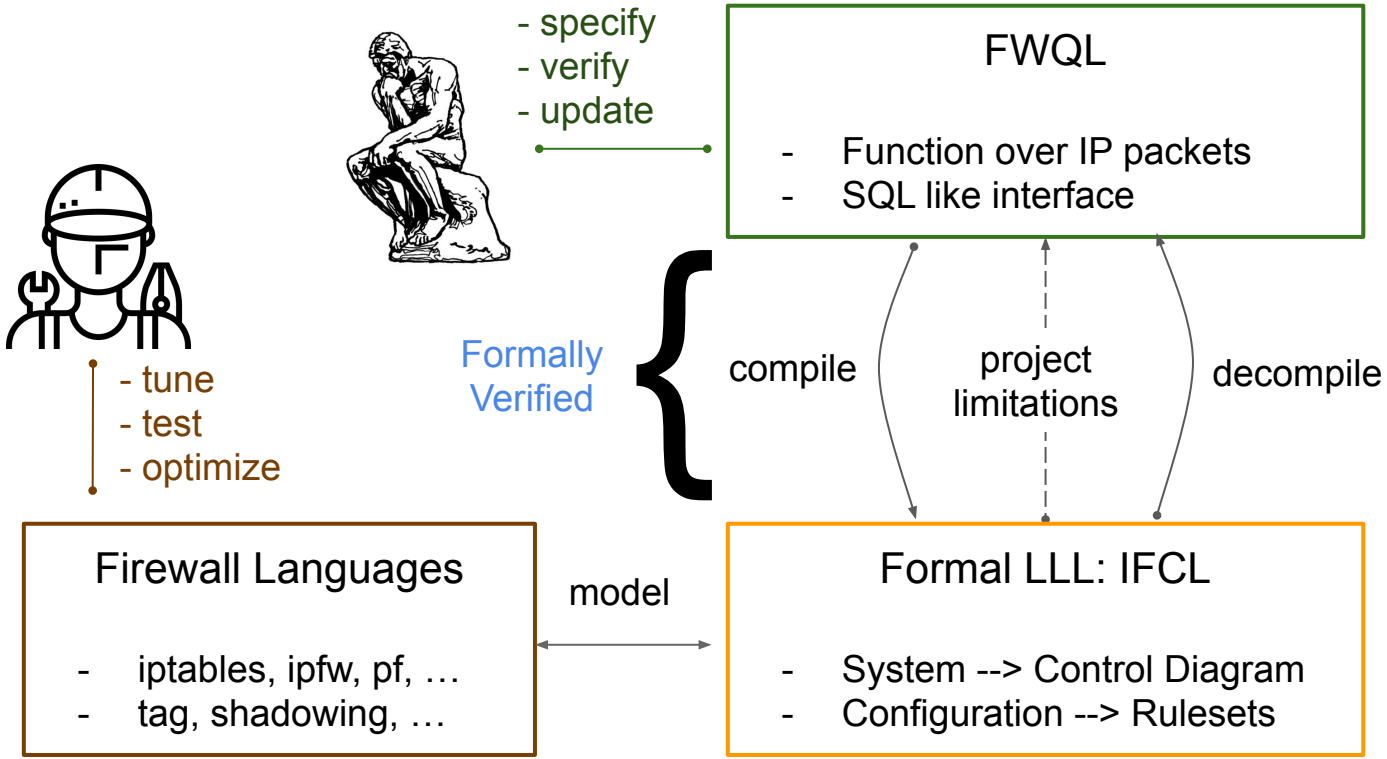
Rulesets Association

$$c_{pf}(q_i) = R \qquad c_{pf}(q_0) = R_{snat} \qquad c_{pf}(q_2) = R_{dnat}$$
$$c_{pf}(q_f) = R \qquad c_{pf}(q_1) = R_{fout} \qquad c_{pf}(q_3) = R_{finp}$$

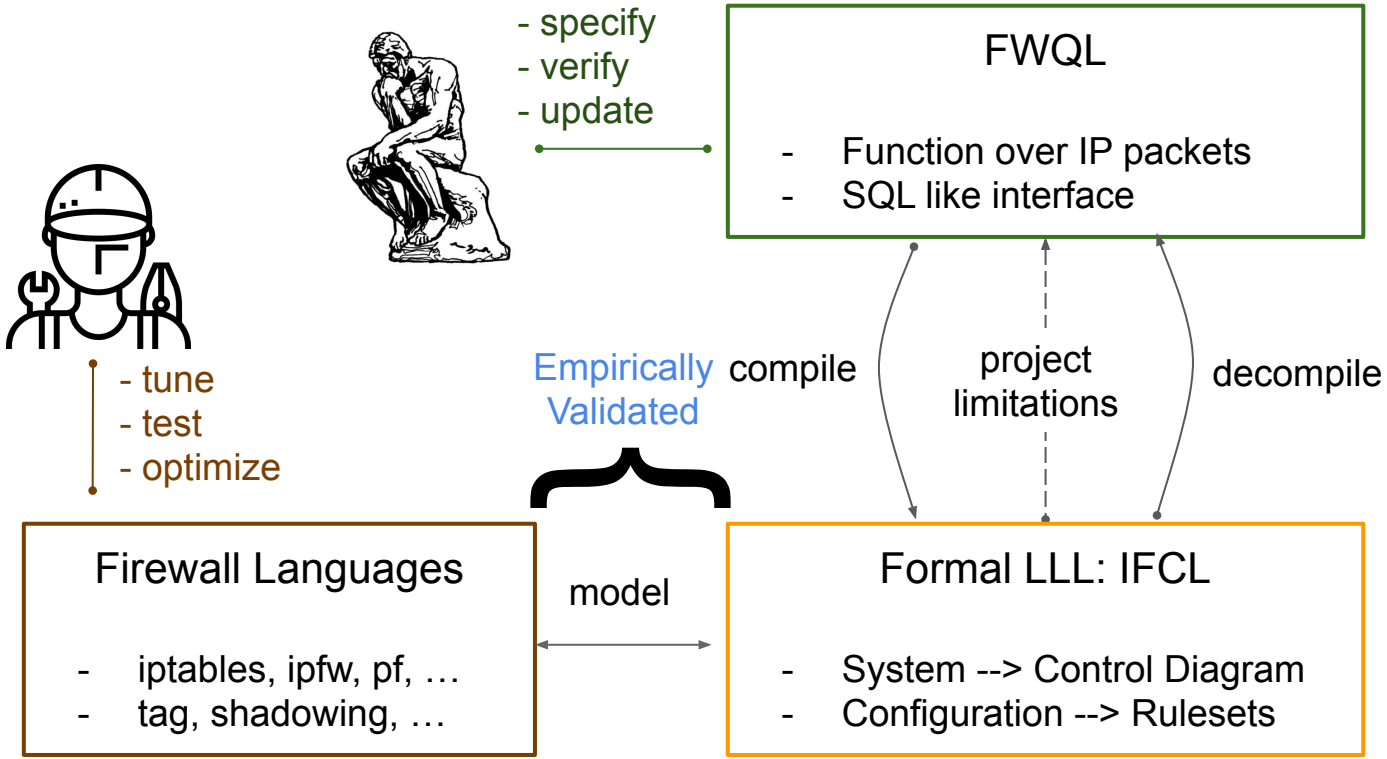
FWS/F2F - Tool


- specify
- verify
- update
FWQL
- Function over IP packets
- SQL like interface
compile
project limitations
decompile
- tune
- test
- optimize
Firewall Languages
- iptables, ipfw, pf, …
- tag, shadowing, …
model
Formal LLL: IFCL
- System --> Control Diagram
- Configuration --> Rulesets

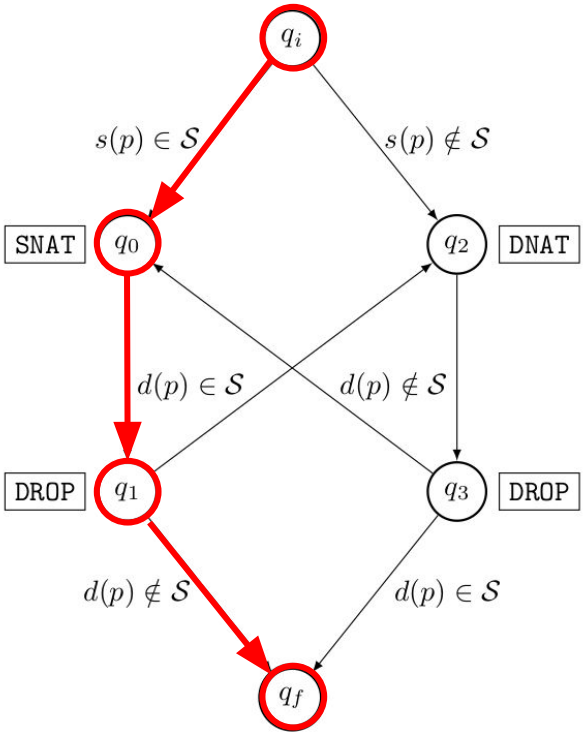
FWS/F2F - Tool


- specify
- verify
- update
FWQL
- Function over IP packets
- SQL like interface
- tune
- test
- optimize
Formally Verified
compile
project limitations
decompile
Firewall Languages
- iptables, ipfw, pf, …
- tag, shadowing, …
model
Formal LLL: IFCL
- System --> Control Diagram
- Configuration --> Rulesets

FWS/F2F - Tool


- specify
- verify
- update
FWQL
- Function over IP packets
- SQL like interface
- tune
- test
- optimize
Empirically Validated
compile
project limitations
decompile
Firewall Languages
- iptables, ipfw, pf, …
- tag, shadowing, …
model
Formal LLL: IFCL
- System --> Control Diagram
- Configuration --> Rulesets

Expressivity Problem

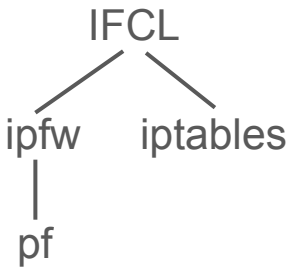


Individual Expressivity

pf cannot apply Destination NAT (**DNAT**) on packets following the path in red

Functional Expressivity

packet p accepted with SNAT
packet p' dropped
what if p after SNAT is equal to p' in q_1?



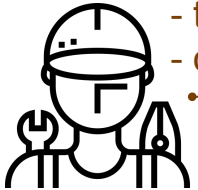
Verification Based Solution

- Configuring by hand
- Verification procedure guarantees coherence between high and low level
- Support specification and configuration changes
- When compilation would be risky (security critical low level details)

- specify
- verify
- update

High Level Language

verify coherence



- configure
- test
- optimize

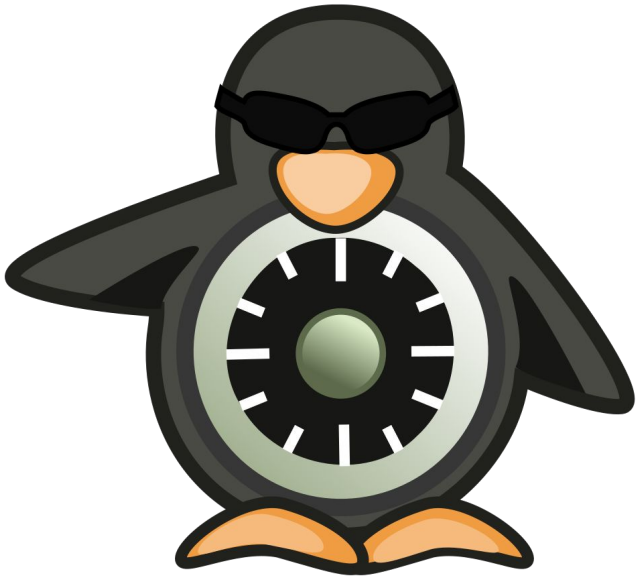
Low Level Language

SELinux CIL

SELinux policy defines mandatory access control for the applications, processes, and files on a Linux system.

Used from Servers to Android devices

CIL allows to structure configurations using macros and blocks

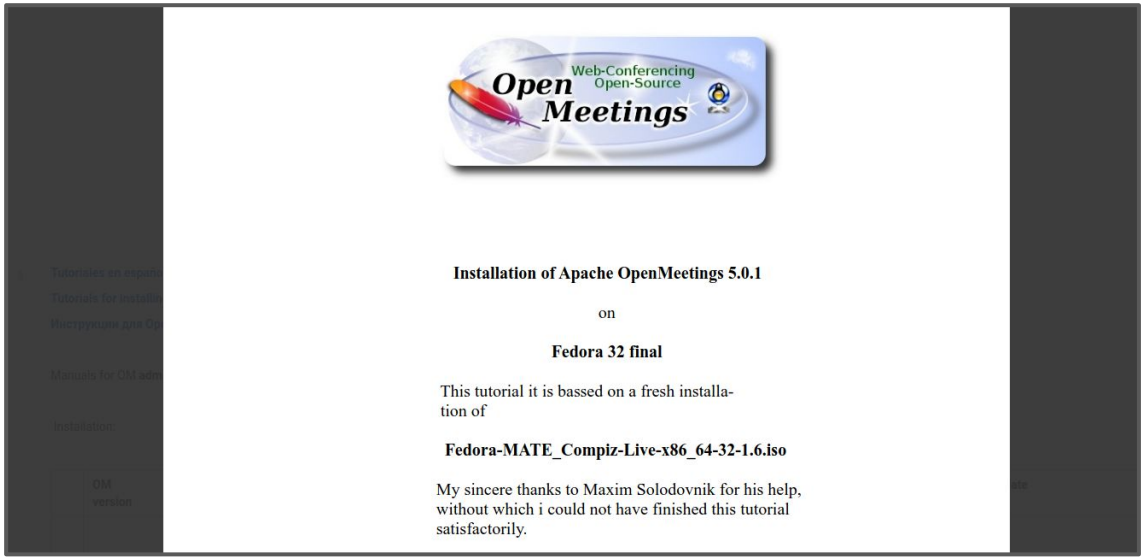


SELinux - Notoriously a Nightmare

- OS entities and operations are numerous and varied
- Configurations are huge

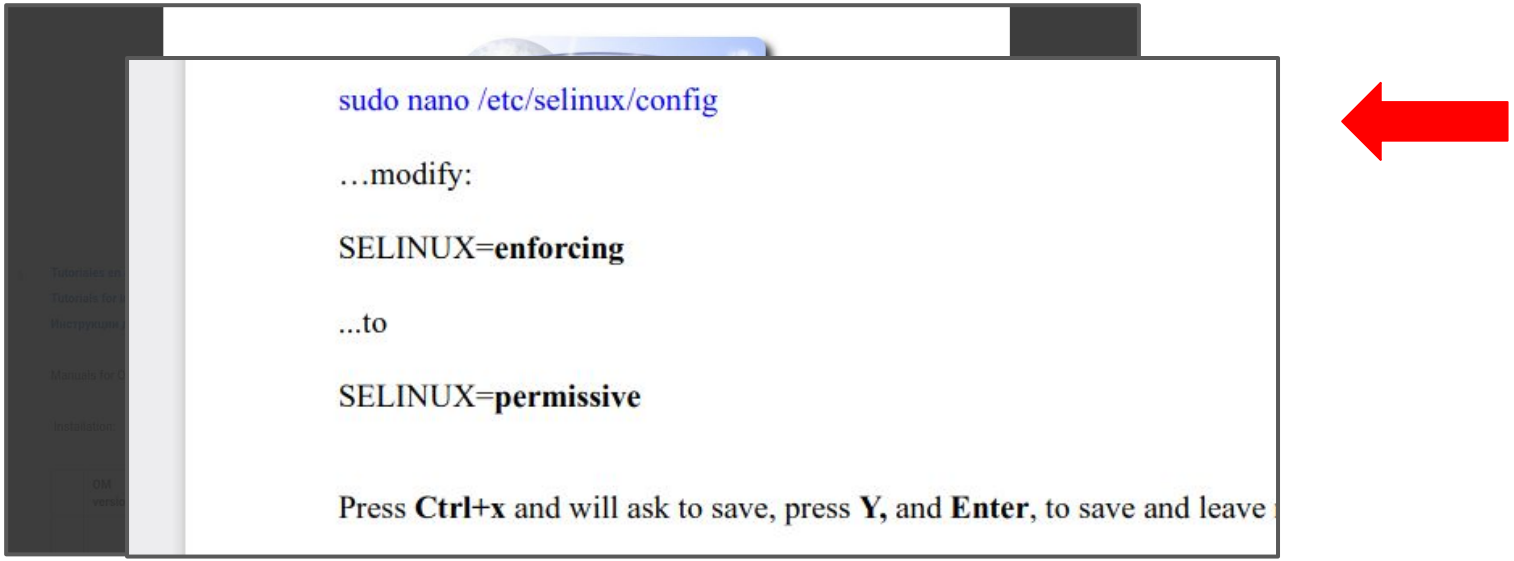
SELinux - Notoriously a Nightmare

- OS entities and operations are numerous and varied
- Configurations are huge



SELinux - Notoriously a Nightmare

- OS entities and operations are numerous and varied
- Configurations are huge



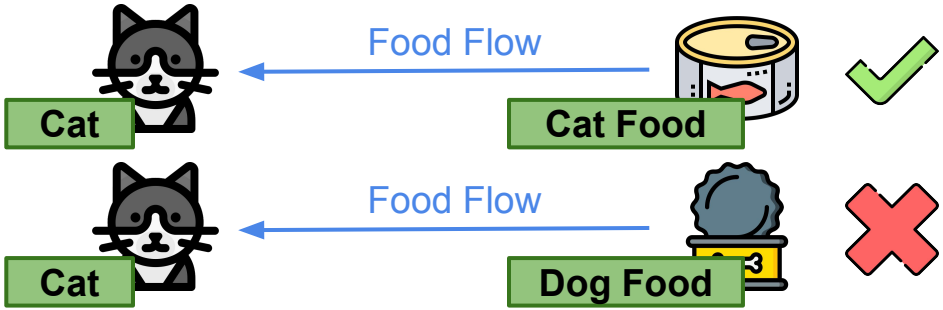
SELinux - Low Level Configurations

- Every part of the OS is associated with Types
- A set of Operations are defined
- Rules “Type x can perform Operation a on Type y”

Types: Dog, Cat, Dog Bowl, Cat Bowl

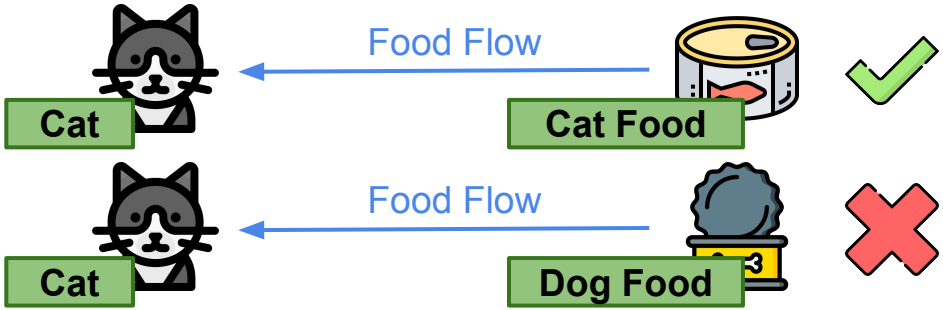
SELinux - High Level Specifications

Flow properties:

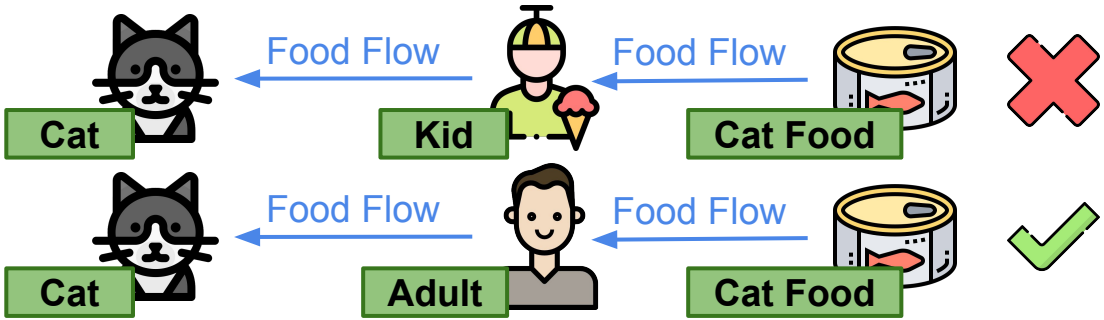


SELinux - High Level Specifications

Flow properties:

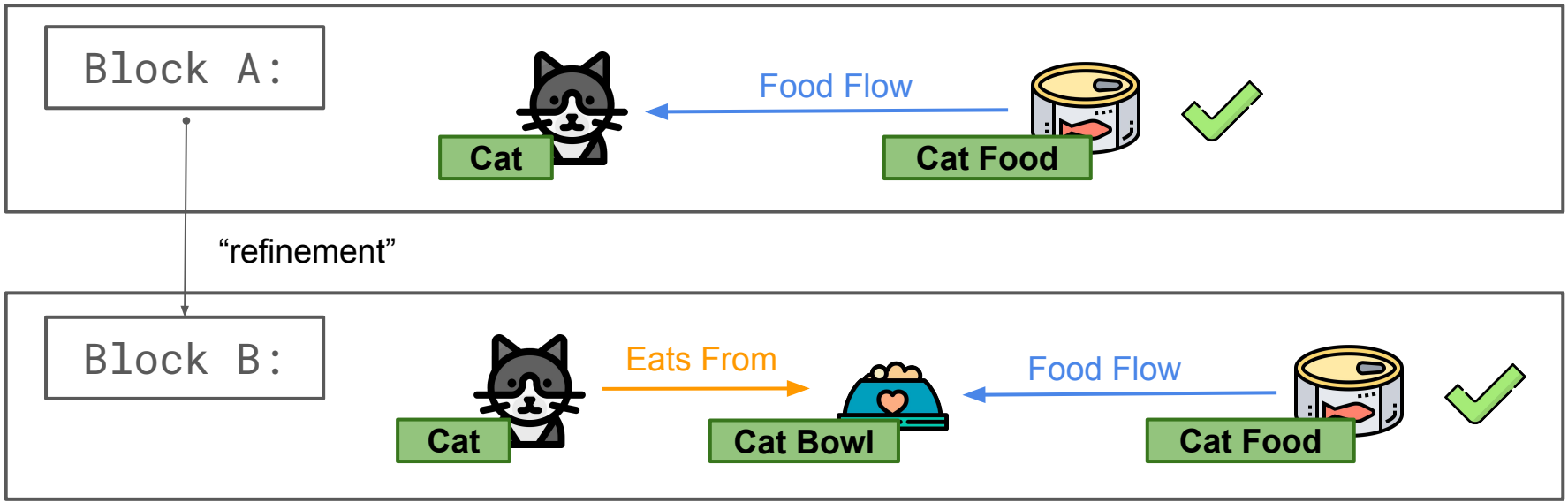


Intransitive Flow Properties:



SELinux - High Level Specifications

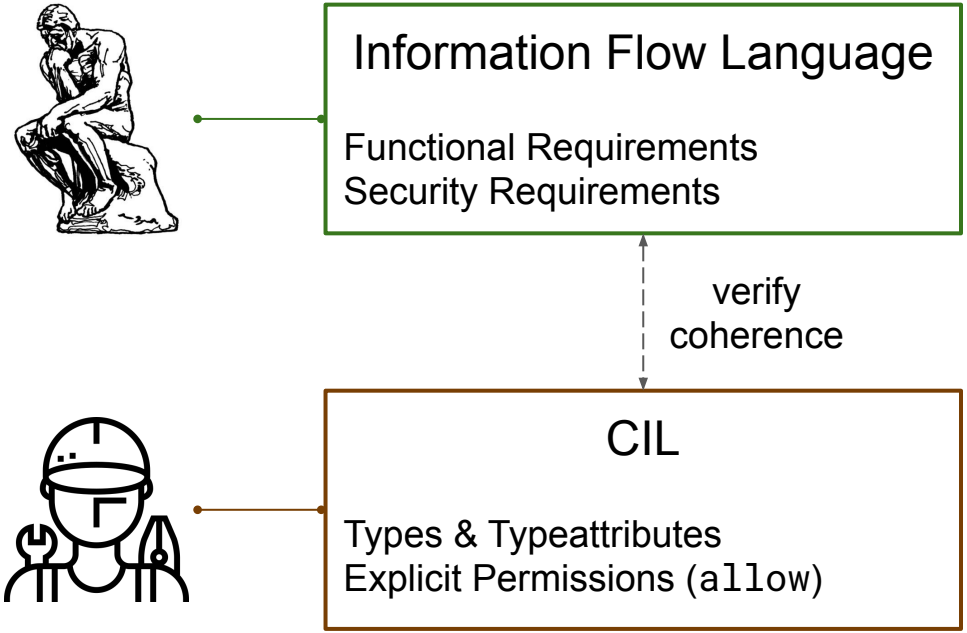
Flow properties allow Policy Engineering:



SELinux IFCIL

IFCIL extends CIL with IFL requirements that are first class citizens

A verification procedure grants that the actual permissions satisfies the requirements



IFCIL - Example

```
(macro anonymize((type x) (type y))
    (type anon)
    (allow anon x (file (read)))
    ;IFL; (S1) x +> y : x > anon +> y ;IFL;)

(type DB)
(type http)
(type net)

;IFL; (F1) DB +> http +> net ;IFL;
;IFL; (F2) net +> http +> DB ;IFL;

(call anonymize(DB net))

(allow http anon (file (read)))
(allow http DB (file (write)))
(allow http net (file (read write)))
```

IFCIL - Example

CIL

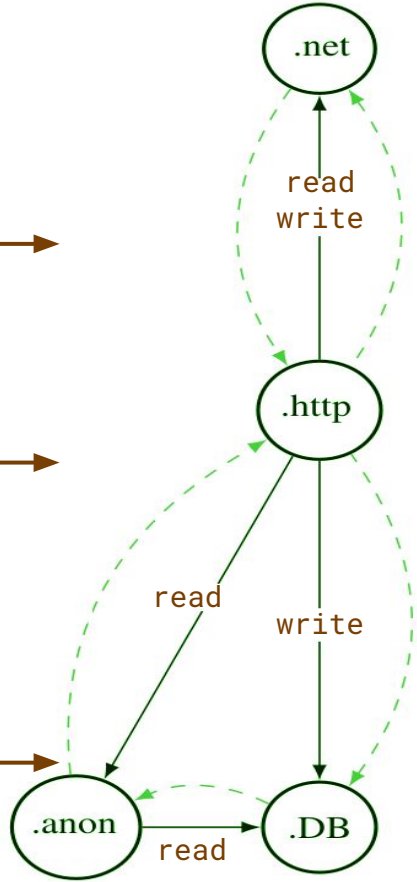
```
(macro anonymize((type x) (type y))
    (type anon)
    (allow anon x (file (read)))
    ;IFL; (S1) x +> y : x > anon +> y ;IFL;)
```

```
(type DB)
(type http)
(type net)
```

```
;IFL; (F1) DB +> http +> net ;IFL;
;IFL; (F2) net +> http +> DB ;IFL;
```

```
(call anonymize(DB net))

(allow http anon (file (read)))
(allow http DB (file (write)))
(allow http net (file (read write)))
```

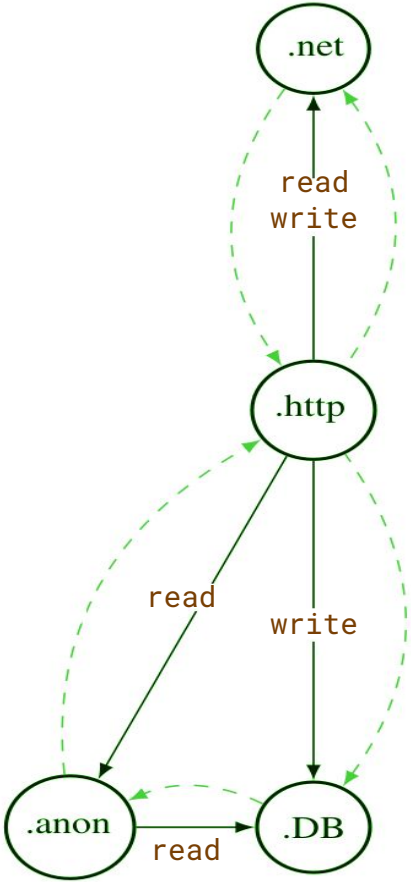


IFCIL - Example

```
;IFL; (S1) DB +> net : DB > anon +> net ;IFL;
;IFL; (F1) DB +> http +> net ;IFL;
;IFL; (F2) net +> http +> DB ;IFL;
```

IFCIL encoded as NuSMV configuration file :

- Permissions as Kripke Transition System
- Requirements as LTL formulas



One-way Translation Based Solution

- Users interact with the High Level, only tools interact with the Low Level representation
- Automatise simple but error-prone tasks
- Prevent misunderstanding due to different languages
- Support specification changes

- specify
- verify
- update

High Level Language

compile

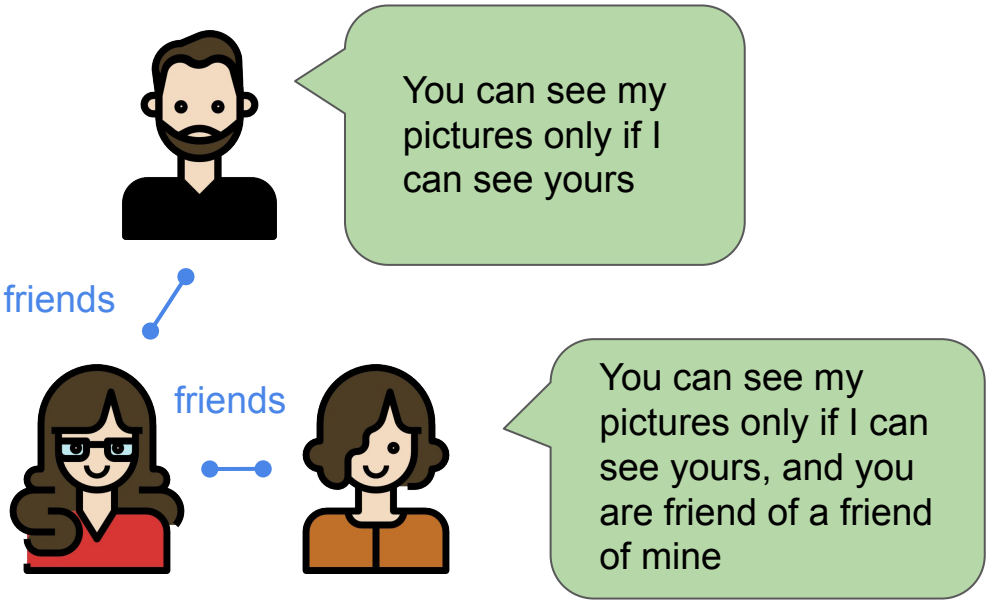
Low Level Language

Collaborative Environments

Users own resources and decide their AC policies

Traditional AC cannot express exchange conditions

New feature: AC decisions based on what the owner gets in return



Resources

Infinite or Reusable

- Private Data on Social Networks
- Files on a File Sharing Platform
- Read-only Accesses

Finite and Not Reusable

- Non Fungible Tokens
- Cryptocurrencies
- Memory Storage
- Computing Power
- Physical Assets

Resources

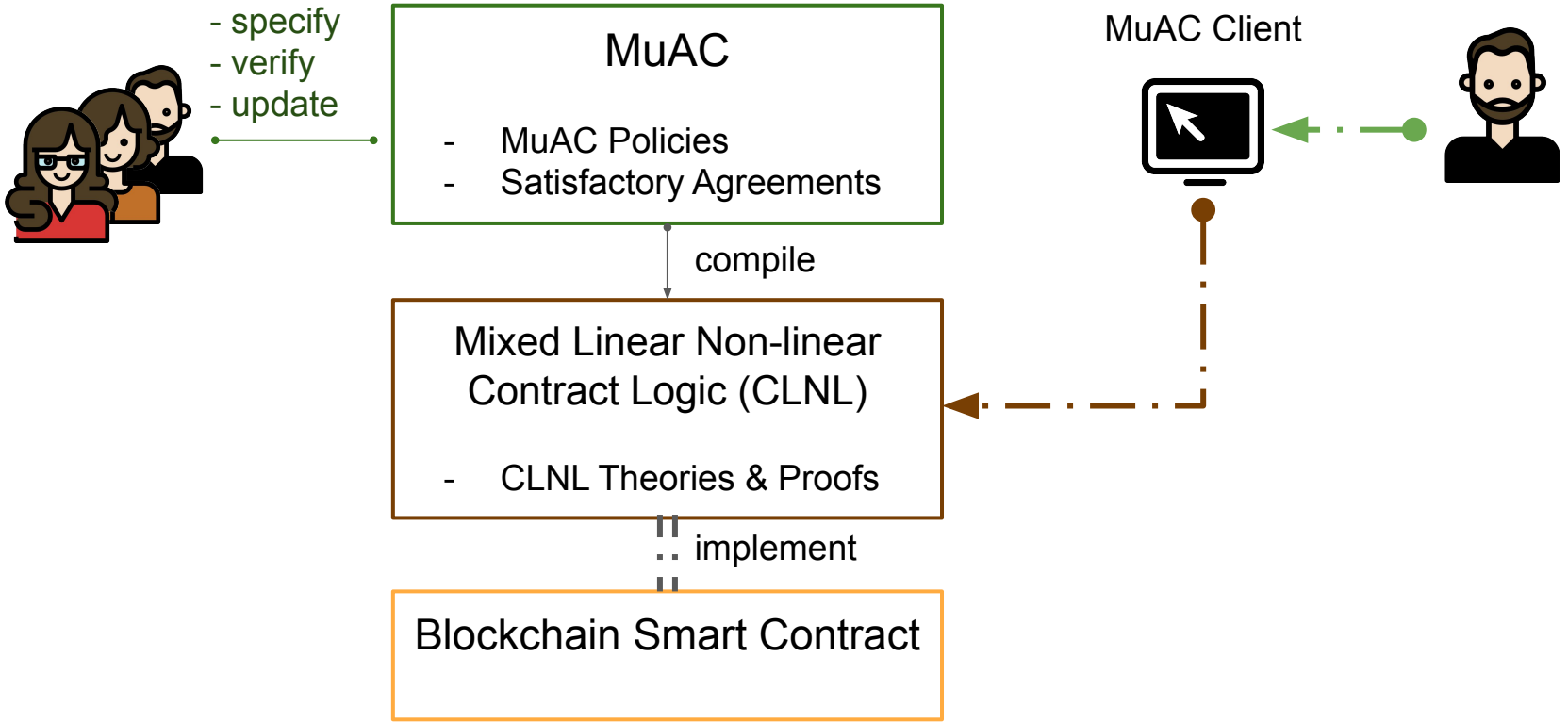
Infinite or Reusable

- Private Data on Social Networks
- Files on a File Sharing Platform
- Read-only Accesses

Finite and Not Reusable

- Non Fungible Tokens
- Cryptocurrencies
- Memory Storage
- Computing Power
- Physical Assets

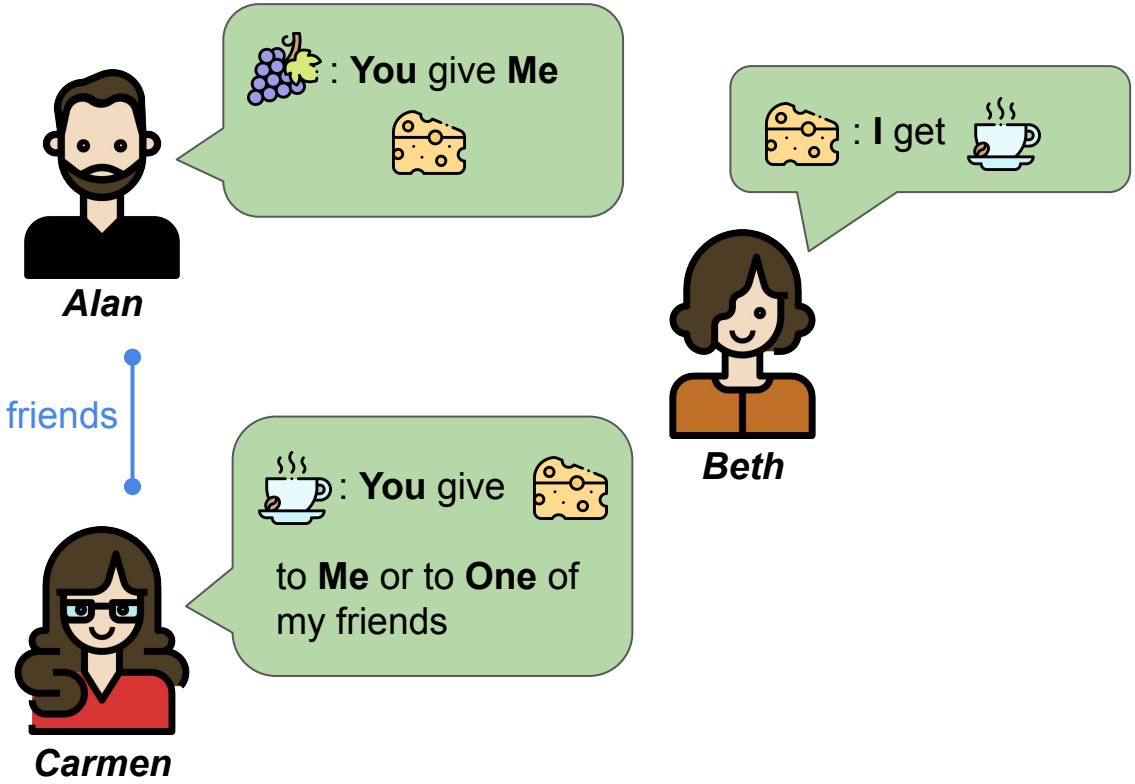
MuAC


- specify
- verify
- update
MuAC
- MuAC Policies
- Satisfactory Agreements
compile
Mixed Linear Non-linear Contract Logic (CLNL)
- CLNL Theories & Proofs
implement
Blockchain Smart Contract
MuAC Client

MuAC Policies

Users define their policies in isolation.

Conditions about what other users must give in order to obtain the permission for a given resource.



CLNL - Computing Agreements

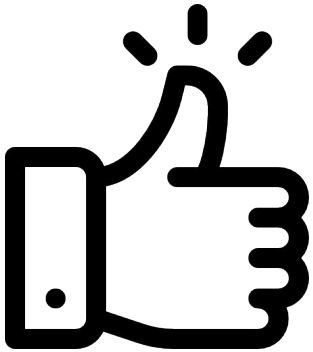
Alan gives  to **Beth** if **Beth** gives  to **Alan**

Beth gives  to **Alan** if **Alan** gives  to **Beth**

CLNL - Computing Agreements

Alan gives  to **Beth** if **Beth** gives  to **Alan**

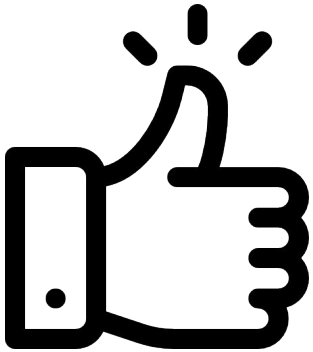
Beth gives  to **Alan** if **Alan** gives  to **Beth**



CLNL - Computing Agreements

Alan gives  to **Beth** if **Beth** gives  to **Alan**

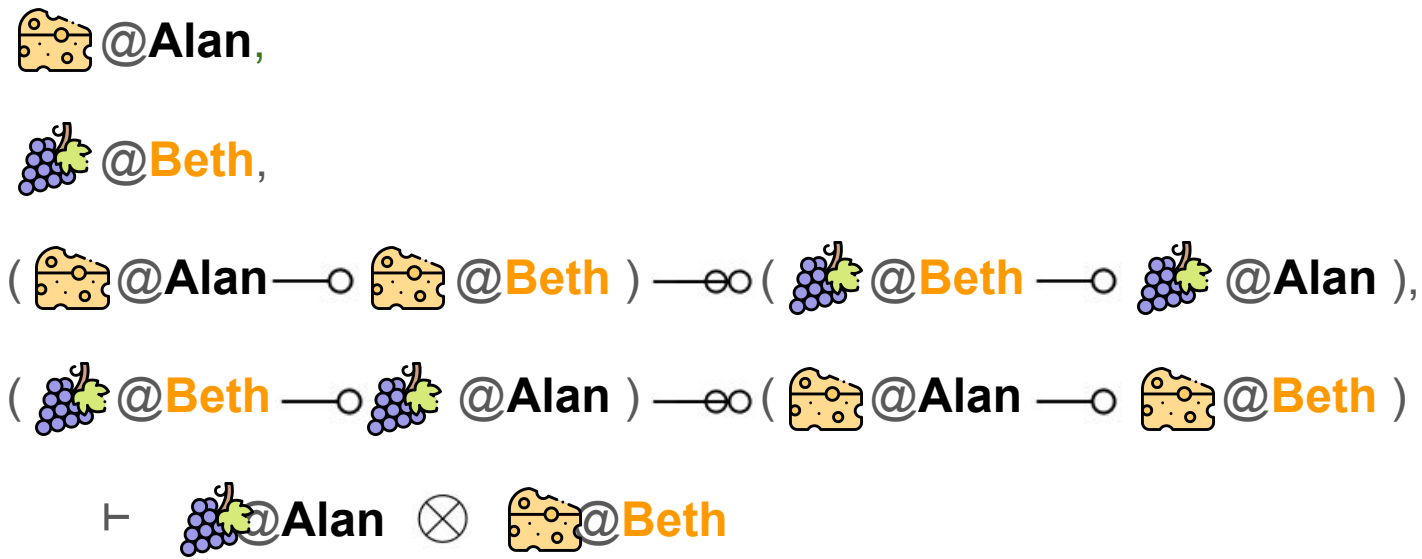
Beth gives  to **Alan** if **Alan** gives  to **Beth**



Classical Logic Does not Work!

$$a \Rightarrow b,\ b \Rightarrow a \not\vdash a \wedge b$$

CLNL - Computing Agreements



CLNL - Computing Agreements

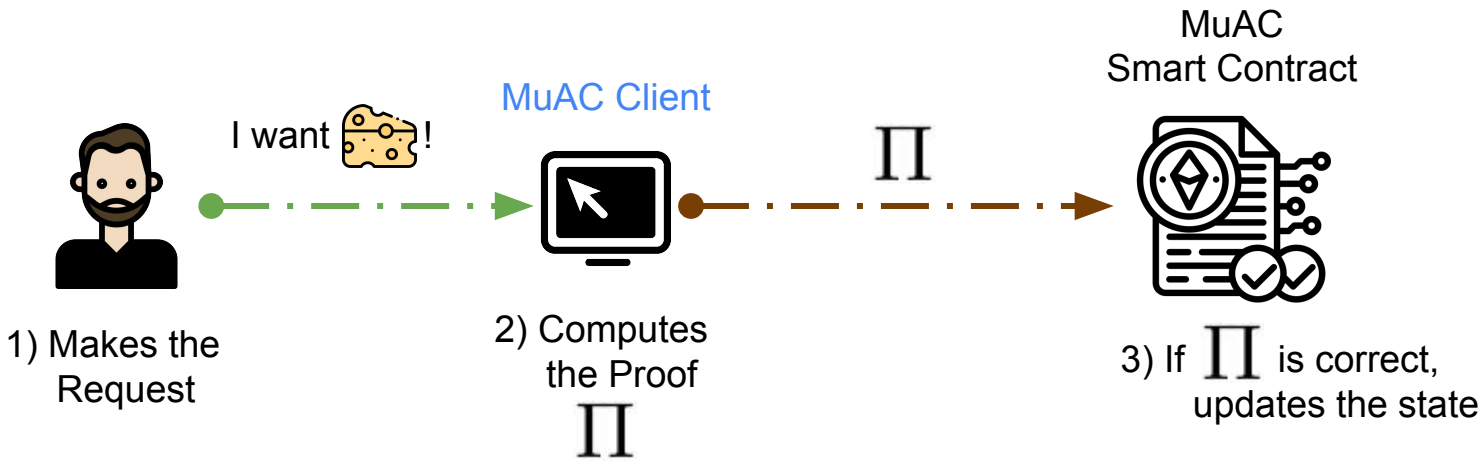
Assuming a request from **Beth** for 

An agreement is mutually satisfactory **iff** a CLNL proof exists

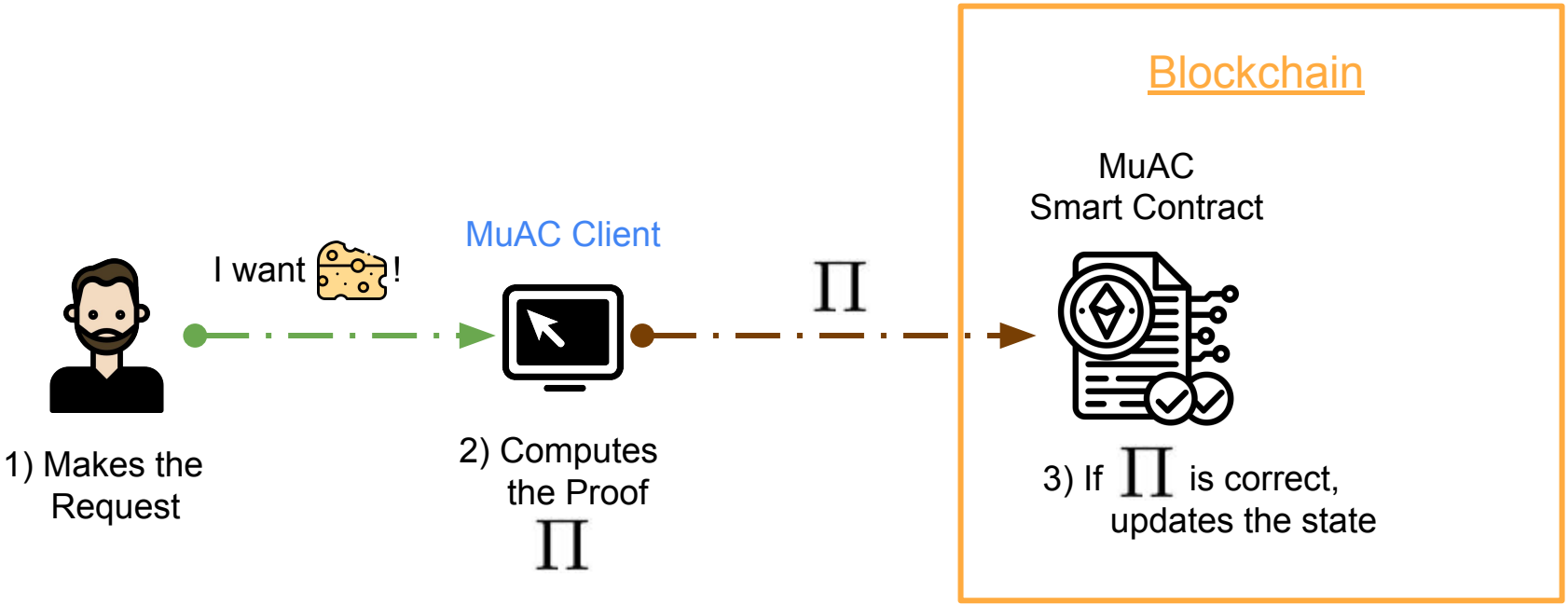
[[Policies]], [[Actual State]] ⊢ [[New State (where **Beth** has ]]

Algorithm for finding such a proof (on a computational fragment)

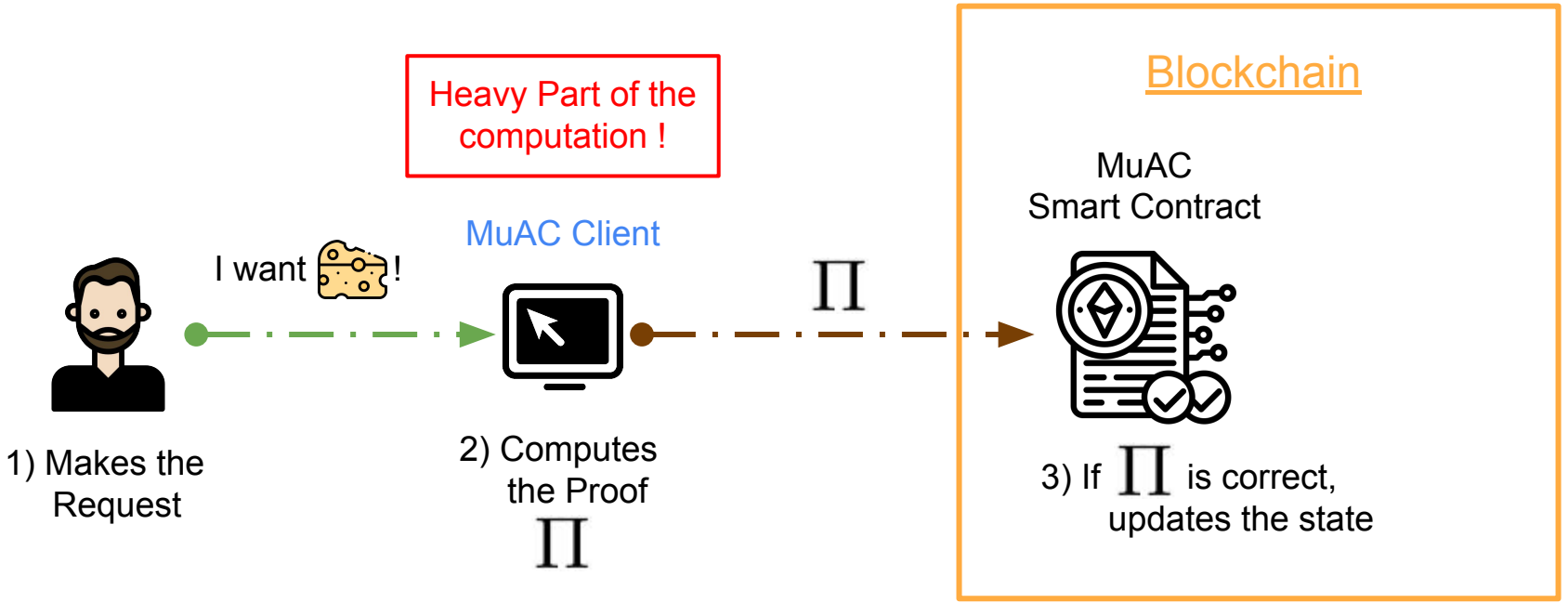
MuAC as a Smart Contract for Exchanging NFTs



MuAC as a Smart Contract for Exchanging NFTs

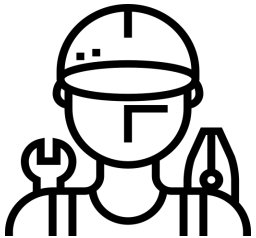


MuAC as a Smart Contract for Exchanging NFTs



Concluding Remarks - Two-Layers Approach…

High Level



Low Level

Granting Coherence

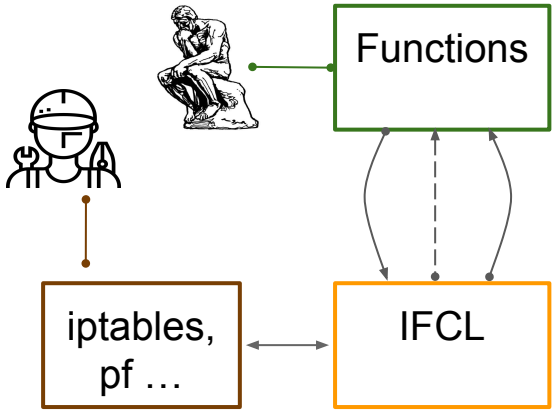
Translation Based

- **one-way** : low level details in charge of tools
- **two-way** : low level details in charge of both humans and tools

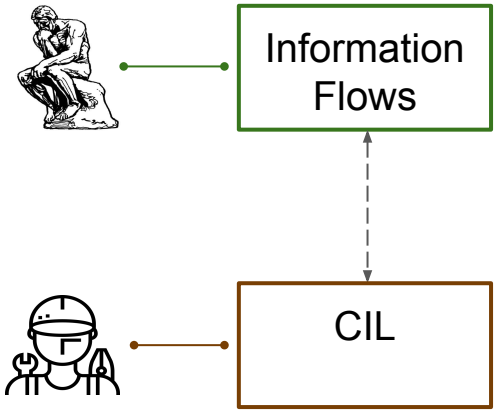
Verification Based :
low level details in charge of humans

…Three Solutions for Three Contexts

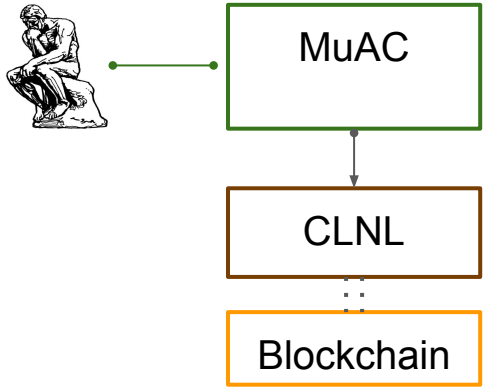
Network Security
Firewalls



System Security
SELinux



Collaborative Environments
Access Grants
Exchanges



Publications

- L. Ceragioli, L. Galletta, M. Tempesta, **From Firewalls to Functions and Back**, *ITASEC 2019*
- L. Ceragioli, P. Degano, L. Galletta, **Are All Firewall Systems Equally Powerful?**, *PLAS@CCS 2019*
- L. Ceragioli, P. Degano, L. Galletta, **Checking the Expressivity of Firewall Languages**, *The Art of Modelling Computational Systems 2019 - LNCS11760*
- C. Bodei, L. Ceragioli, P. Degano, R. Focardi, L. Galletta, F. Luccio, M. Tempesta, L. Veronese, **FWS: Analyzing, Maintaining and Transcompiling Firewalls,** *Journal of Computer Security 29(1) - 2021*

Publications

- L. Ceragioli, P. Degano, L. Galletta, **MuAC: Access Control Language for Mutual Benefits**, *ITASEC 2020*
- L. Ceragioli, P. Degano, L. Galletta, **Can my Firewall System Enforce this Policy?**, *Computers & Security 117 2022*
- L. Ceragioli, L. Galletta, P. Degano, D. Basin, **IFCIL: An Information Flow Configuration Language for SELinux**, *CSF 2022* – `submitted`

Future Work - Extending our Proposals

- Networks
  - Networks with multiple Firewalls
  - Software Defined Networks
- Systems
  - Other CIL features (Roles, MLS)
  - Combination of policies written in different languages
- Collaborative Environments
  - Numbered Resources (currencies)
  - Negative Conditions (conflict of interest)

Future Work - Incrementality and Compositionality

- Translation based solutions
  - Preserve low-level details when compiling
- Verification based solutions
  - Modules related information flows
  - Instant feedback on requirements violations while writing code