University of Southampton

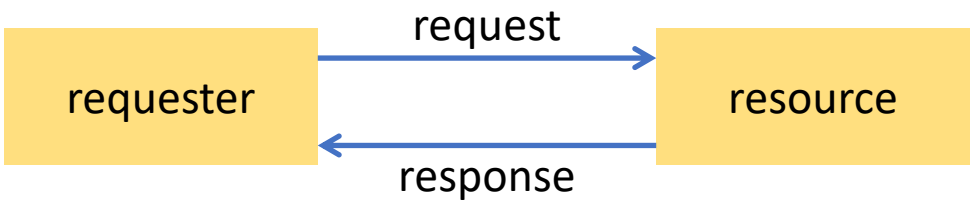
An Event-B Formal Model for Access Control and Resource Management of Serverless Apps

Mehmet Said Nur Yagmahan
msny1y17@soton.ac.uk

Abdolbaghi Rezazadeh
ra3@ecs.soton.ac.uk

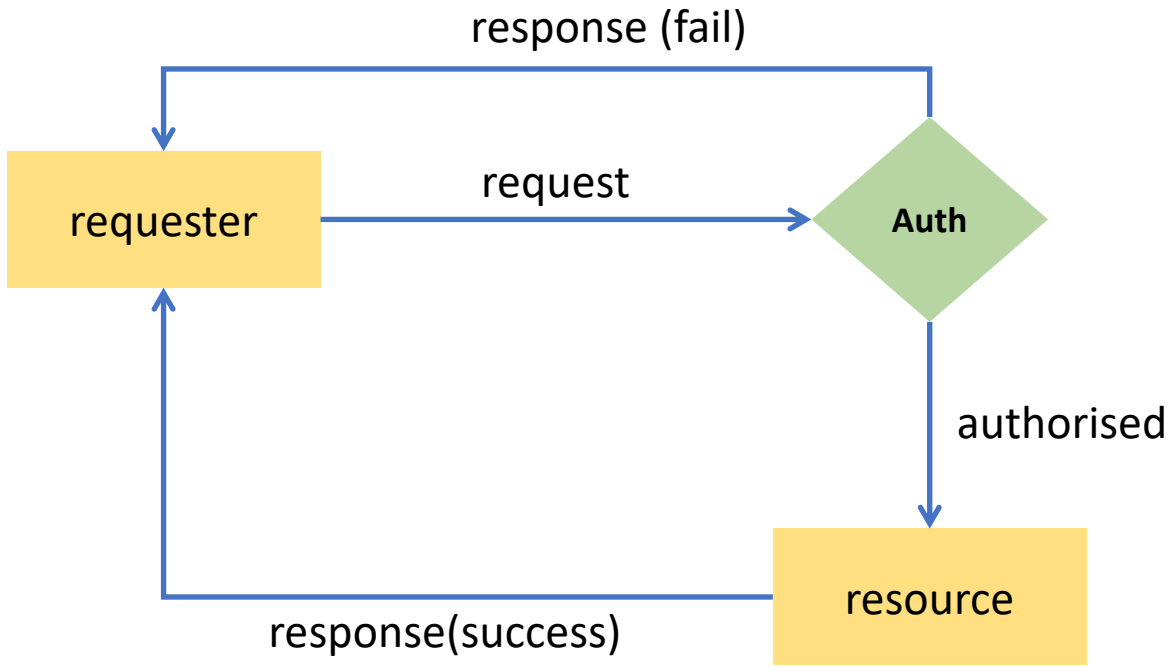
Michael Butler
m.j.butler@soton.ac.uk

Resource Management in Cloud-native Systems



- In cloud, everything is resource.
- Resource = DB table, function, network, permission policy and so on
- To access resource, requests are made

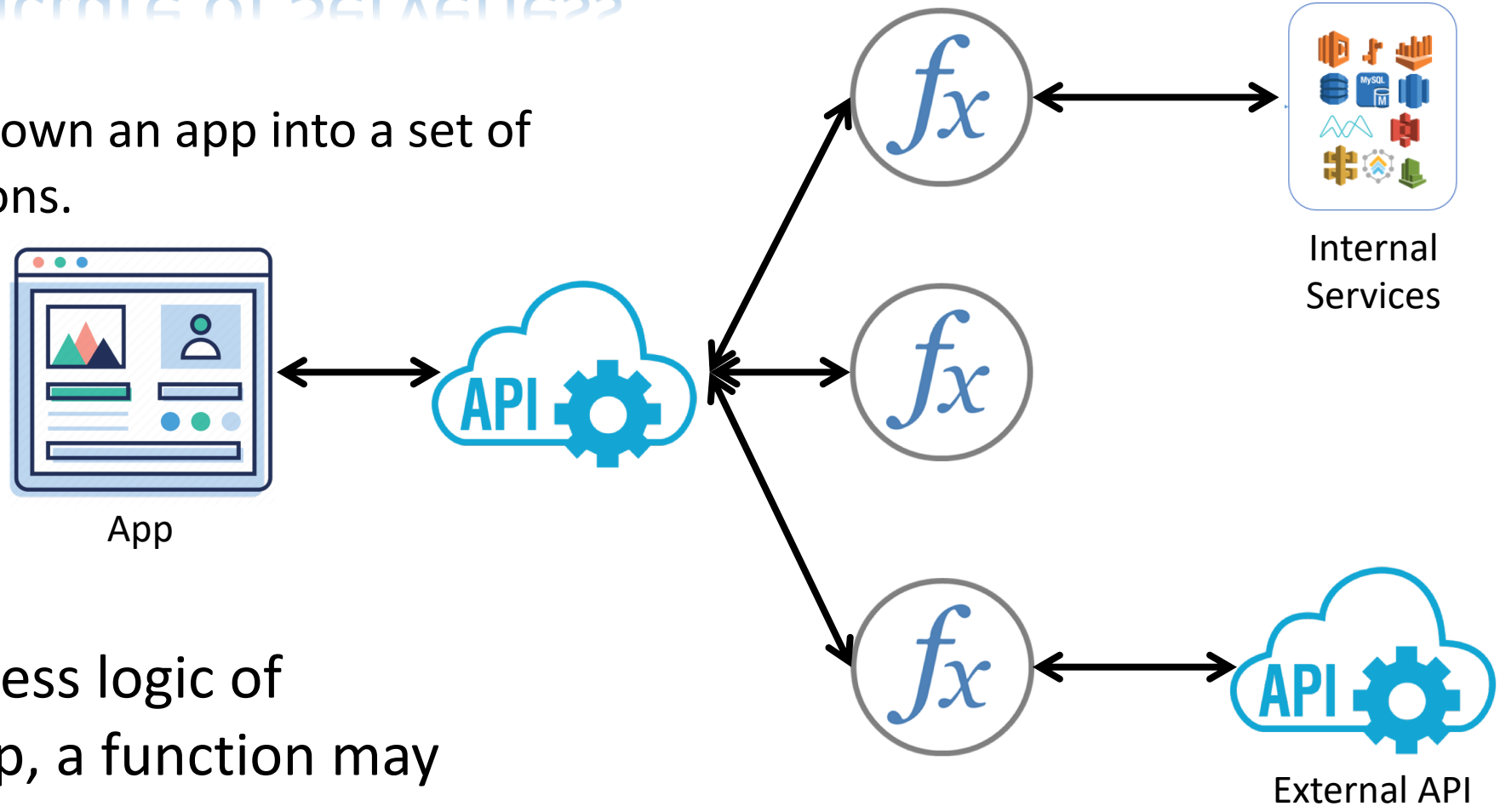
Request Authorization



- Each request from an authenticated entity should go through authorisation mechanism
- The request can be accepted only if the requester have proper authorisation

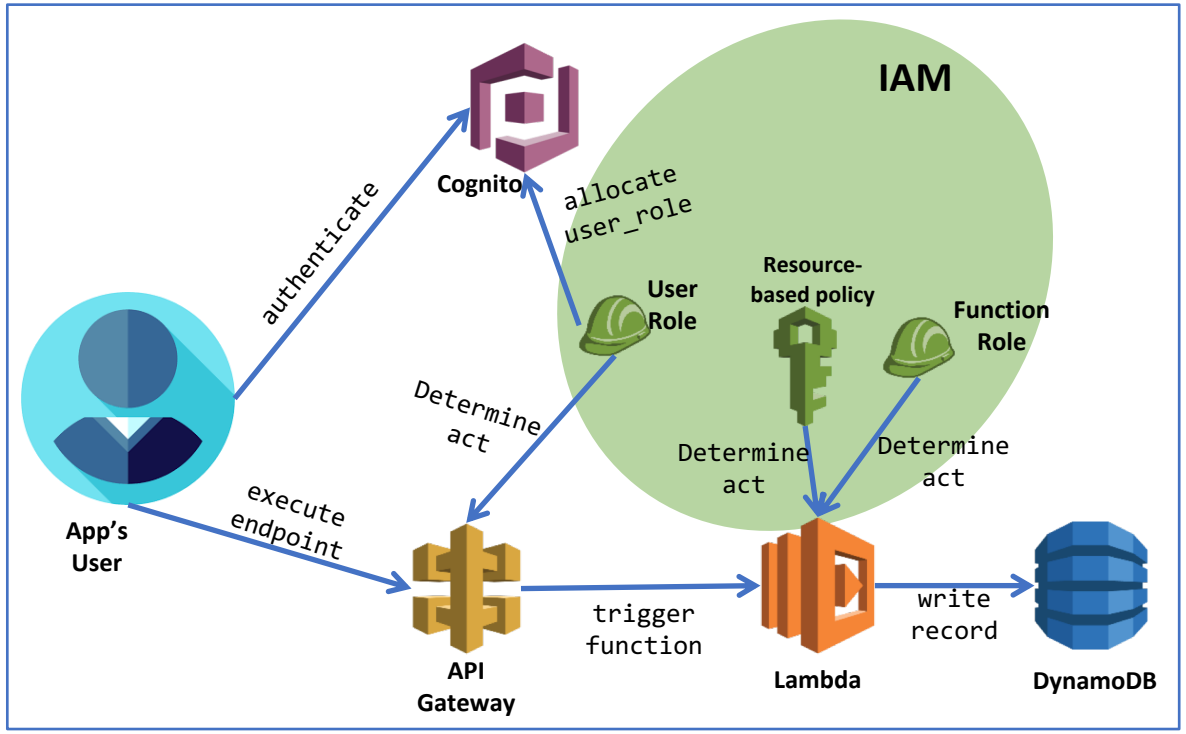
A Basic Structure of Serverless

- … to break down an app into a set of small functions.



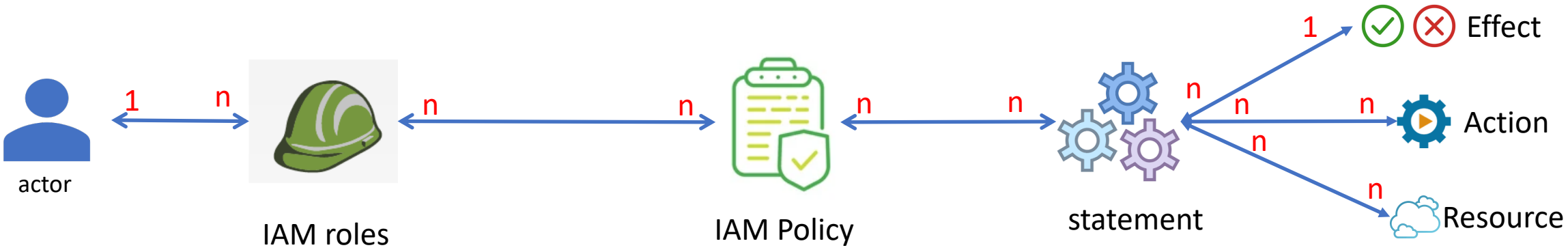
- To fulfil business logic of serverless app, a function may
  - connect to other internal services
  - make calculation and response
  - connect to a third-party external API

An AWS-based Serverless Structure



- **Cognito** : It manages app users and makes association between *them and IAM roles*..
- **API Gateway :** It is used to create **RESTful APIs** that *map to* the **application's functionalities**.
- **Lambda** : It allows to create **lambda functions** that a piece of code that processes a task
- **DynamoDB** : It is a fully managed **database service** (NoSQL). It is used for storing and managing **application's data**.
- **IAM** : It provides role and policy that allow cloud-native app developer to configurate access control in his account

Structure of Permission in AWS




```
{ "Version": "2012-10-17",
    "Statement": [
       {   "Effect": “Allow”,
            "Action": [ “dynamodb:PutItem" ],
            "Resource": [
“arn:aws:dynamodb:[region]:[account-id]:table/project” ,

“arn:aws:dynamodb:[region]:[account-
id]:table/department“ ,

“arn:aws:dynamodb:[region]:[account-id]:table/employee”
                    ]
}      }
```

The **statement** says that:
* **Adding a record to department, employee, or project table in DB is allowed.**

- **Action**
- **Resource**
- **Effect**

The Problem - Complexity

University of Southampton

- Resources and services are
  - Highly distributed
  - Scalable
  - Dynamically allocated/deallocated
- The complexity in Authorisation mechanism
  - multi-layer structure
  - accessible as a resource (role, policies)

- Any inconsistency or security issue in Access Control policy :
  - may lead to misconfiguration
  - may make the serverless system more vulnerable.
  - may lead to unauthorised access to resources

Why these problems are important :

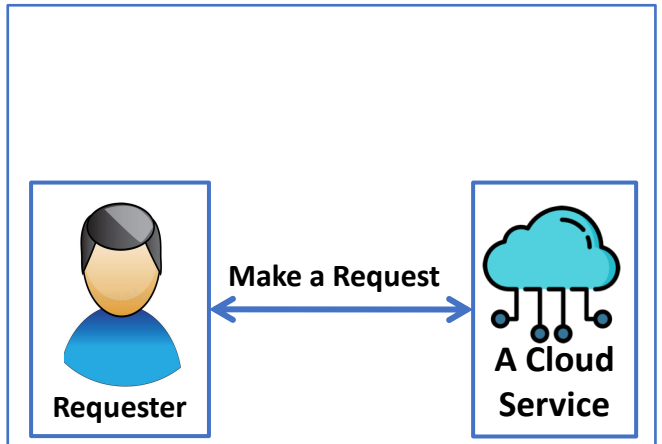
- (1) 65% of cloud incidents were because of customer misconfiguration
- (2) The popularity Serverless Architecture:
  - * 93% of enterprises use cloud services
  - * top trend is Serverless (75% growth rate)

(1) Flexera, “2019 state of the cloud from rightscale," Right Scale, Tech. Rep., 2019. [Online]. Available: https://media.exera.com/documents/ rightscale-2019-state-of-the-cloud-report-from-exera.pdf

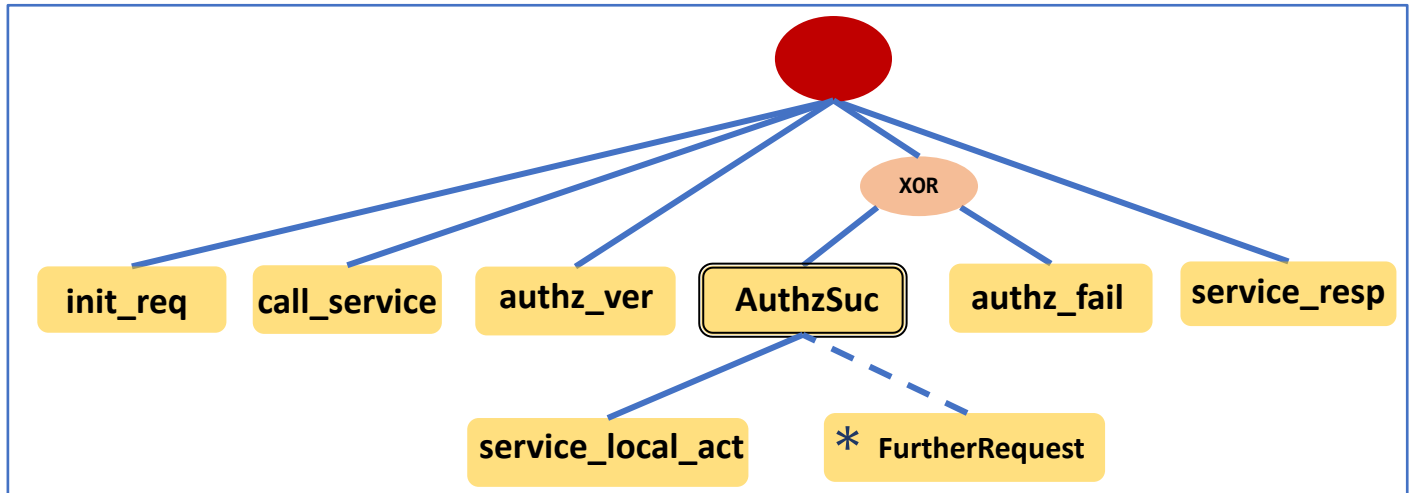
(2) P. A. Networks, “Unit 42 cloud threat report," 2020. [Online]. Available: https://www.paloaltonetworks.com/prisma/unit42-cloud-threat-research

FM Patterns : RHP

University of Southampton



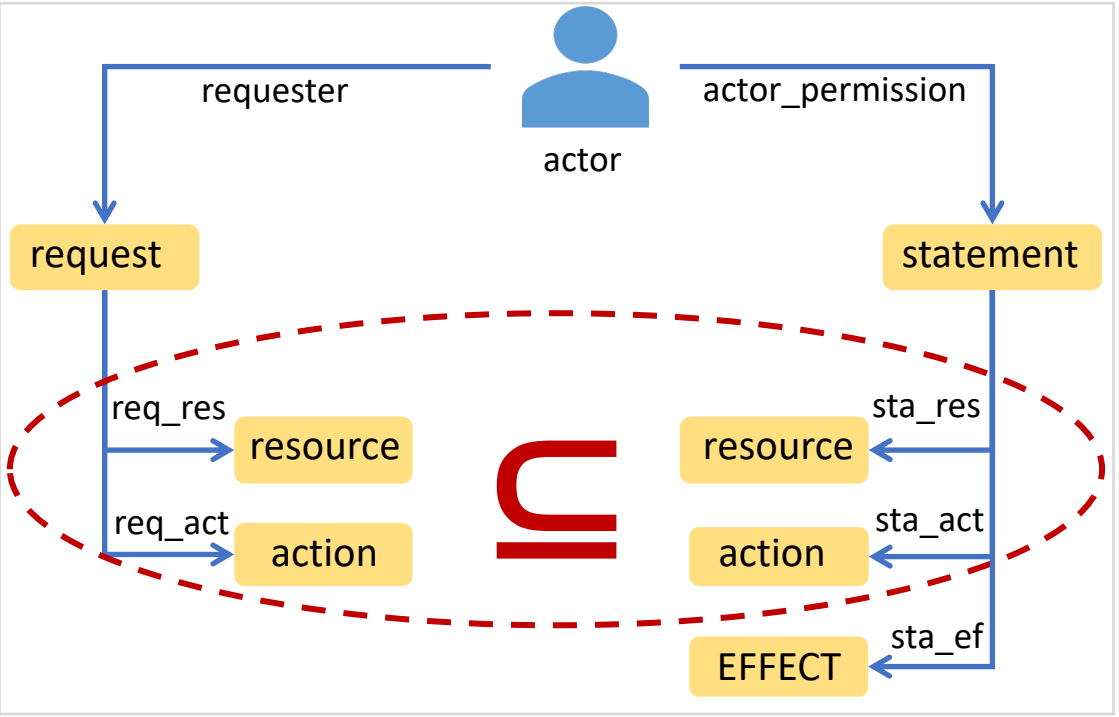
A) Request to a Cloud Service



B) A Generic Pattern for Request Handling in a Cloud-Native Apps

FM Patterns : Authorization Mechanism

Deterministic Authorizer



A) Authorization for actor's requests

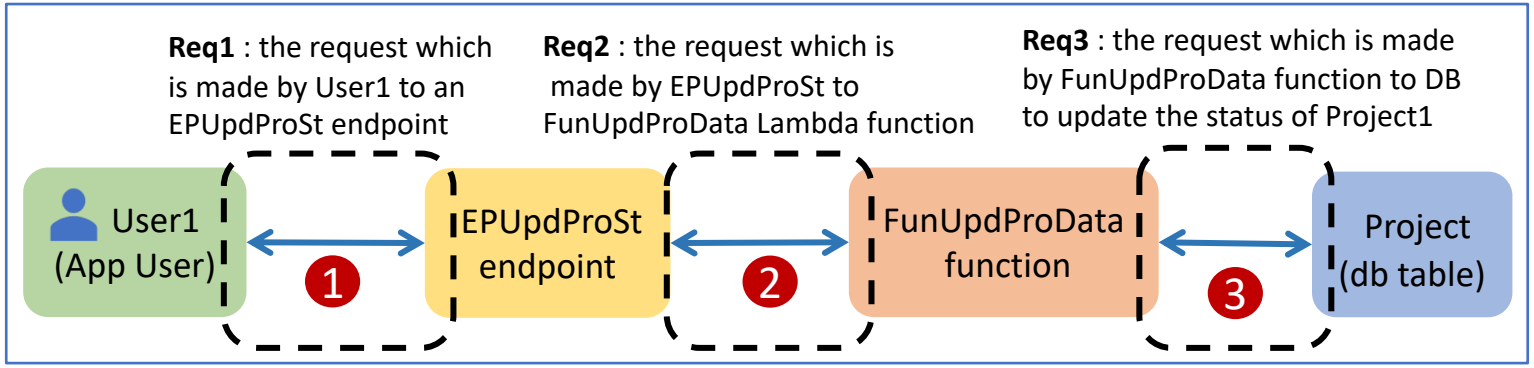
```
event authz_ver_actor_permit ...
any req   pm //permission statement
when
...
@grd2_1 requester(req) ∈ actor
@grd2_3 pm = {st / st ∈ statement ∧
      st ∈ actor_permission[{requester(req)}] ∧
      req_res(req) ∈ sta_res[{st}] ∧
      req_act(req) ∈ sta_act[{st}]}
@grd2_4 ∃st·st∈pm ∧ sta_ef(st)=Allow
@grd2_5 ∀st·st∈pm ⇒ sta_ef(st) ≠ Deny
...
then
@act2_1 req_authz(req) ≔ Allow
end
```

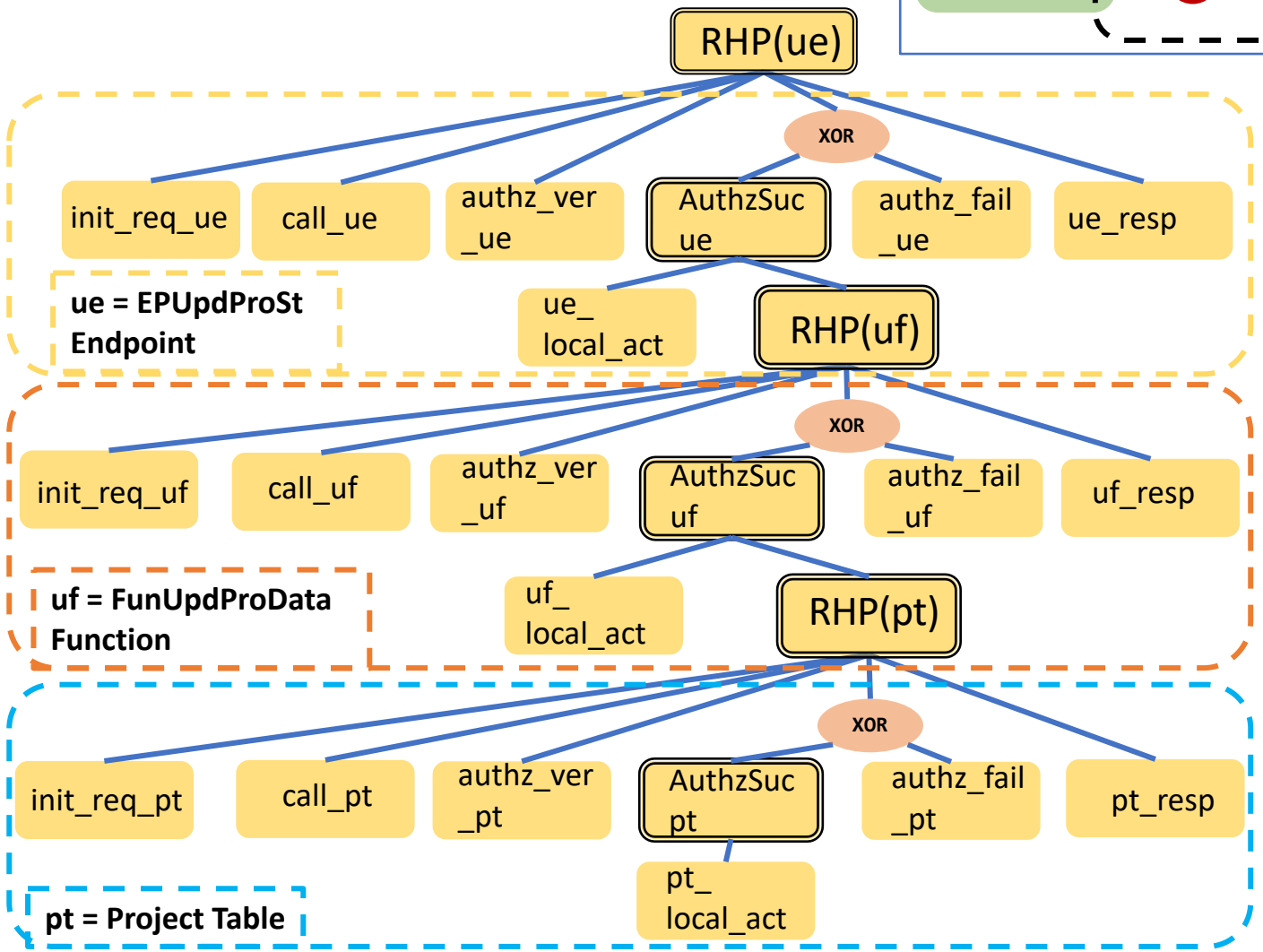
A) Permit case for Actors

```
@inv_authz ∀r·r∈request ∧ req_authz(r)=Allow ∧ requester(r)∈ actor ⇒
        (∃s·s∈statement ∧
           requester(r)↦s ∈ actor_permission ∧
           s↦req_res(r) ∈ sta_res ∧
           s↦req_act(r) ∈ sta_act ∧
           sta_ef(s)= Allow)
```

Case Study Scenario

"Updating Project Status"
in "Project Management System"





Contribution

- Modelling request handling process in the AWS using graphical notation and Event-B Formalism,
  - **RHP** -> Pattern to model a **request life-cycle**
  - **Reusing RHP** Pattern for multiple request to model a **functionality** of a **serverless app**

- Developing guards and invariants to prevent any inconsistencies in policy to achieve effective authorization and security.
  - **Event-B formalism** for **Authorization Mechanism**

Future Work

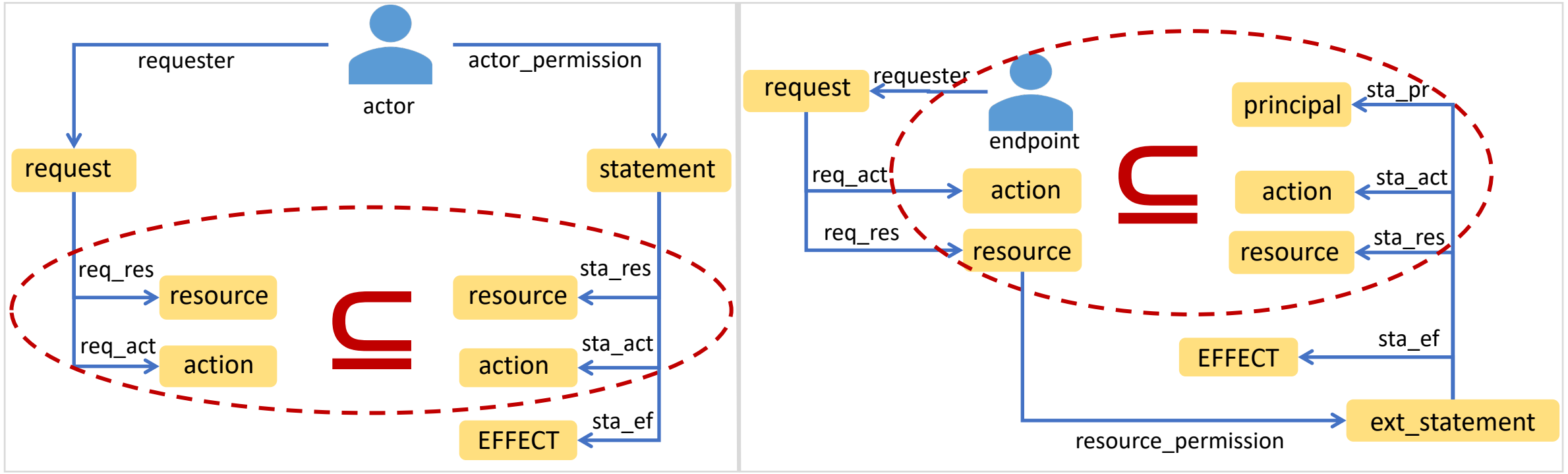
University of Southampton

- to model more case studies from different domains to show wider usability/usefulness of our approach
- add Authentication aspects in our modelling approach to formalize a more comprehensive approach to cloud native security
- To develop a RODIN extension to generate Event-B model from the patterns
- To research on other cloud environments to develop our approach

University of Southampton

Thanks

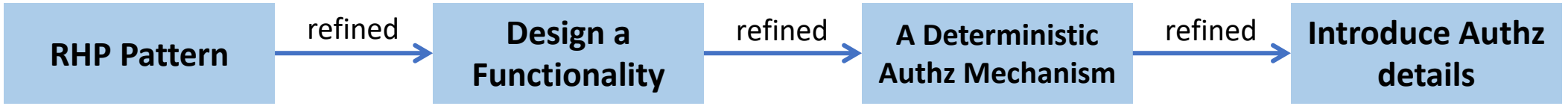
Deterministic Authorizer



A) Authorization for actor's requests

B) Authorization for endpoint's requests

Model a Functionality



Abstract

To introduce request execution life-cycle

1. Refinement

To introduce request sequentially to model a functionality

2. Refinement

To introduce permission and corresponding relation

3. Refinement

To introduce role and policy