EdgStr: Automating **Client-Cloud** to **Client-Edge-Cloud** Transformation

Principle Engineer
Software Engineering Team
Samsung Research

Kijin An

Professor
CS Department
Virginia Tech

Eli Tilevich

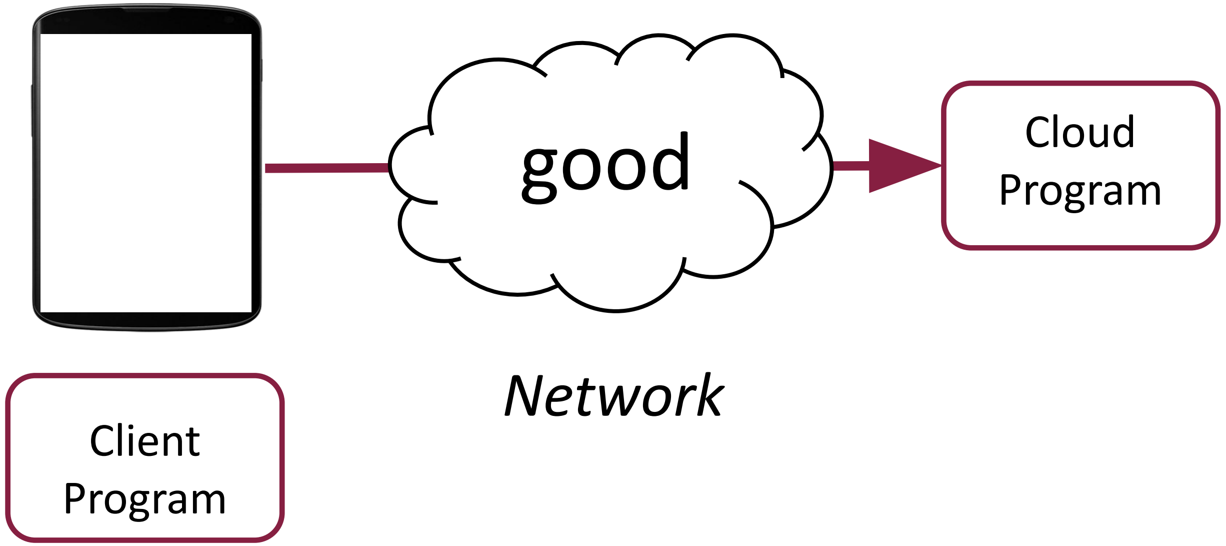
VIRGINIA TECH

Presentation Outline

- Motivation
    - State of distributed software and vision
    - Example application
- Approach for automating transformation to Client-'Edge'-Cloud
- Reference Implementation: EdgStr
- Evaluation
- Conclusions

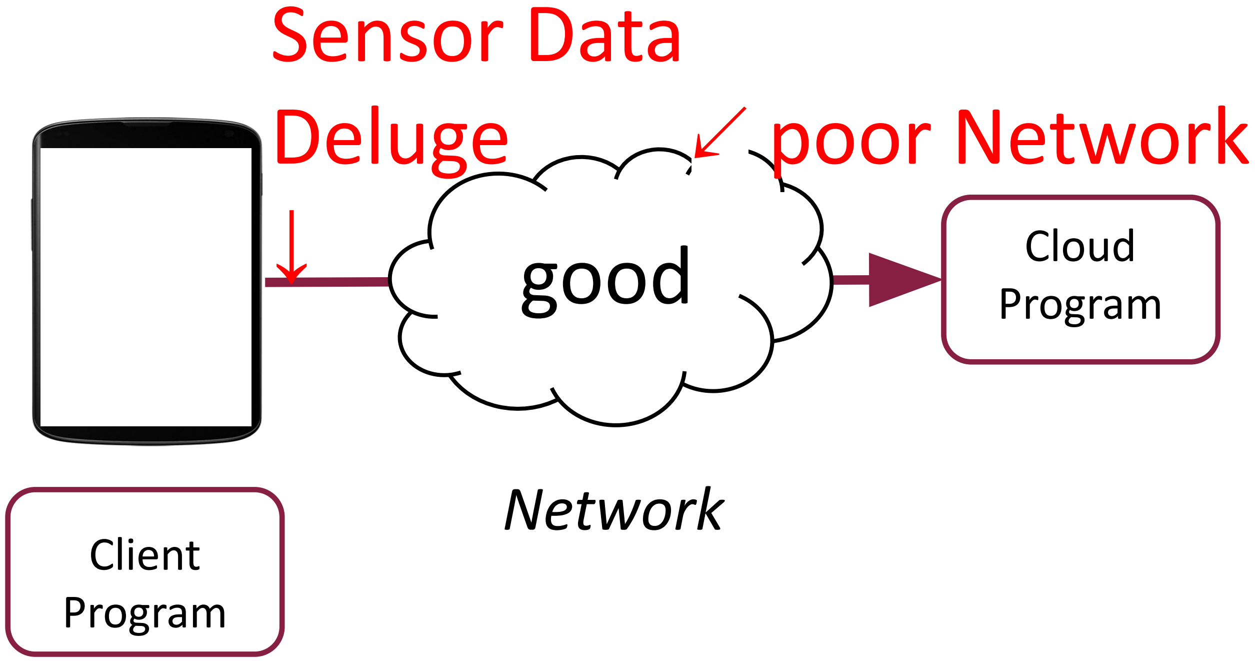
Client-Cloud Architecture (2-tier)

- Cloud-Client predominant
    - Cloud Infrastructure: Powerful
    - Network: Fast
- Conventional 2-tier no longer meets performance and resource utilization requirements of modern apps



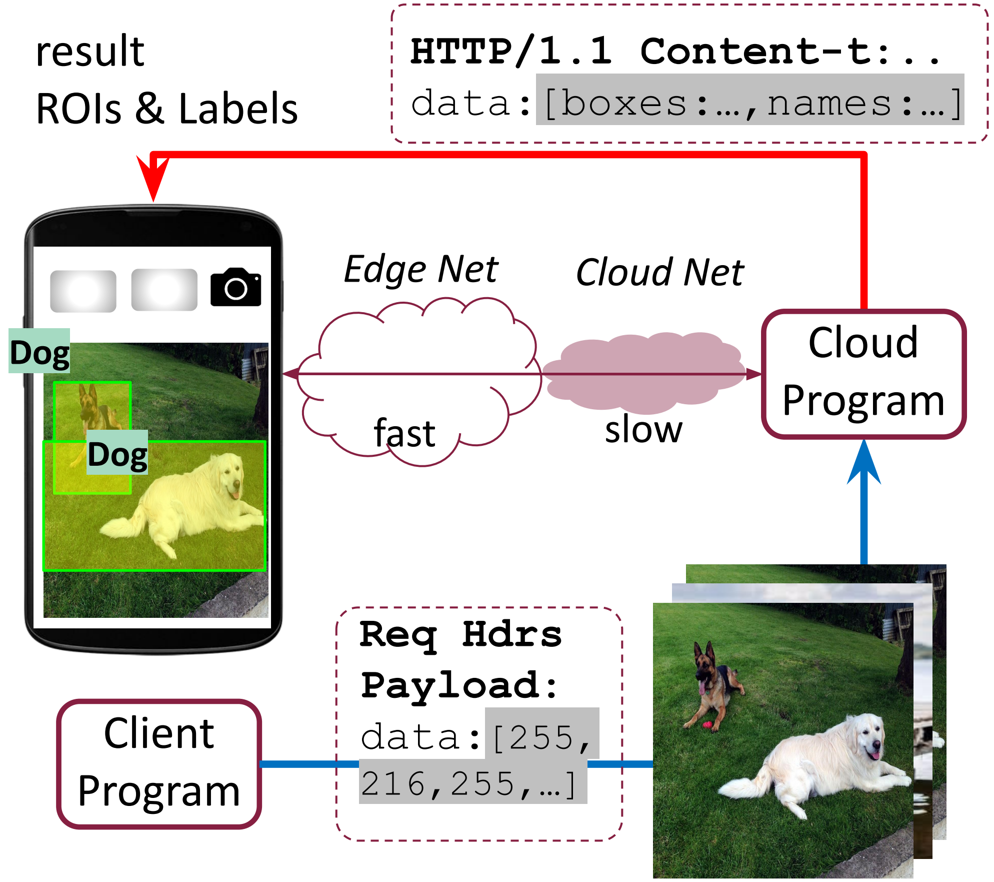
Client-Cloud Architecture (2-tier)

- Cloud-Client
  - Cloud Infrastructure: Powerful
  - Network:Fast
- What if:
  - Network: Slow & unreliable
  - Sensor Data: Massive (“sensor deluge”)
  - □ Increased Latency



Motivating Example
(firebase-objdet)

- Client-Cloud program (/predict, detect objects in the cloud)



```
HTTP/1.1 Content-t:..
data:[boxes:…,names:…]
```

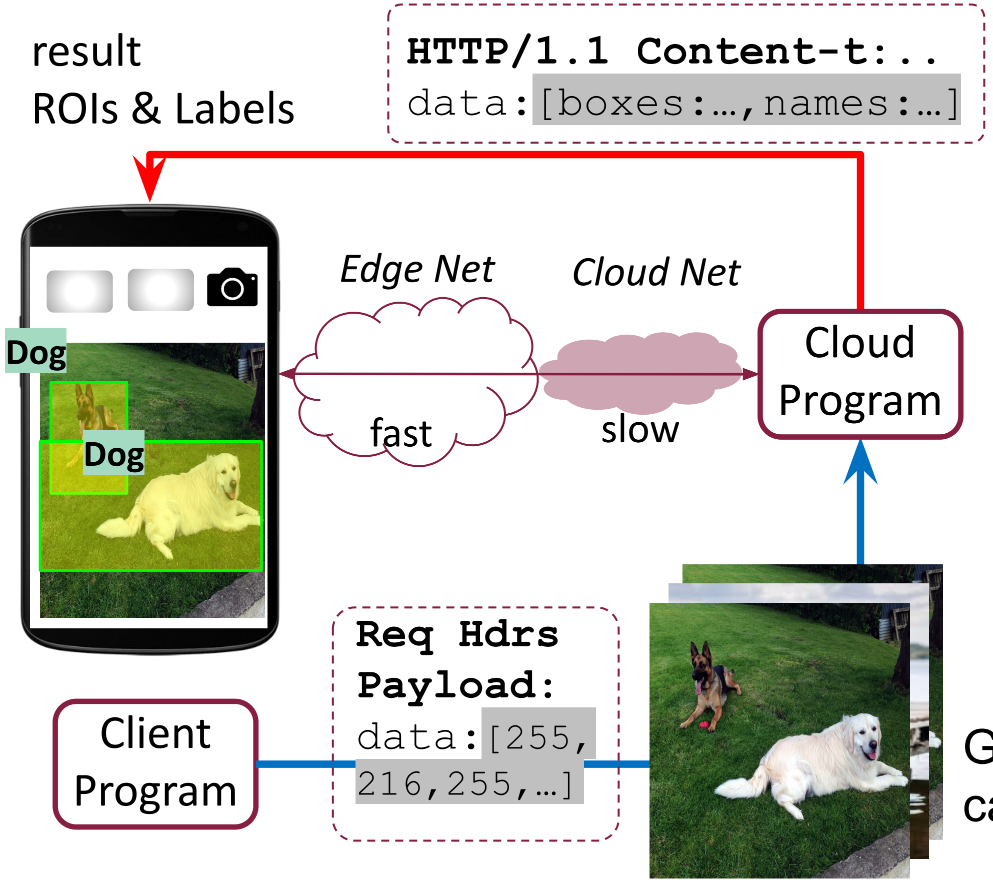
```
Req Hdrs
Payload:
data:[255,
216,255,…]
```

Sensor Data Deluge!

Galaxy S24 Ultra or iPhone 15 Pro can capture a photo of **12MBytes**

Motivating Example
(firebase-objdet)

- Client-Cloud program

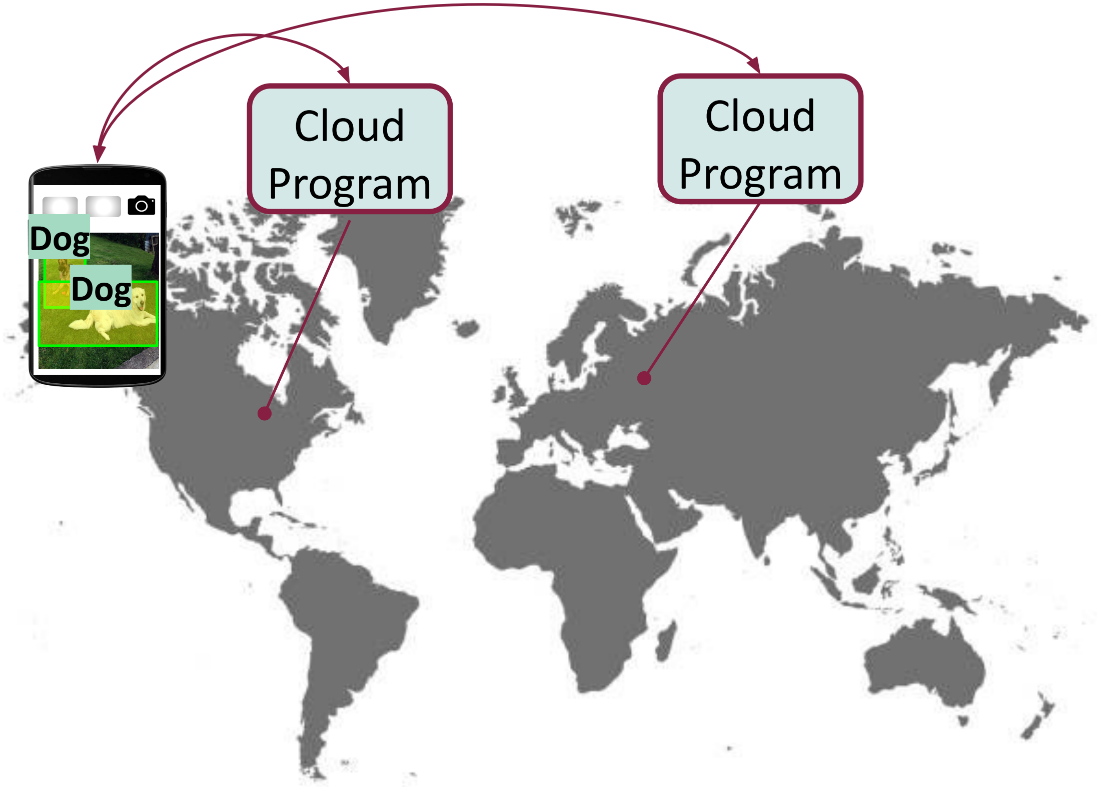


```
HTTP/1.1 Content-t:..
data:[boxes:…,names:…]
```

```
Req Hdrs
Payload:
data:[255,
216,255,…]
```

Galaxy S24 Ultra or iPhone 15 Pro can capture a photo of **12MBytes**

RTT across different/same continents are different from
<span>:An Order of magnitude between them!</span>

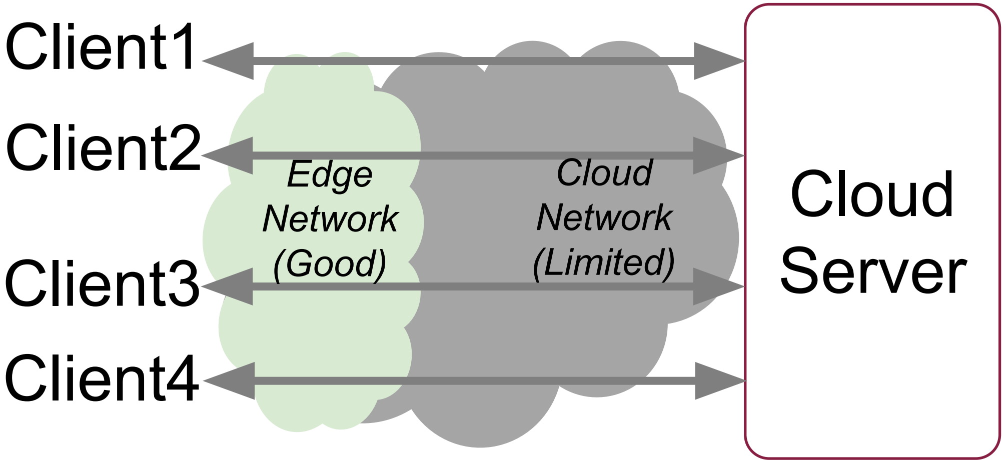


[Installed Cloud Programs differently on *Heroku* platform]

Transforming 2-tier into 3-tier architecture

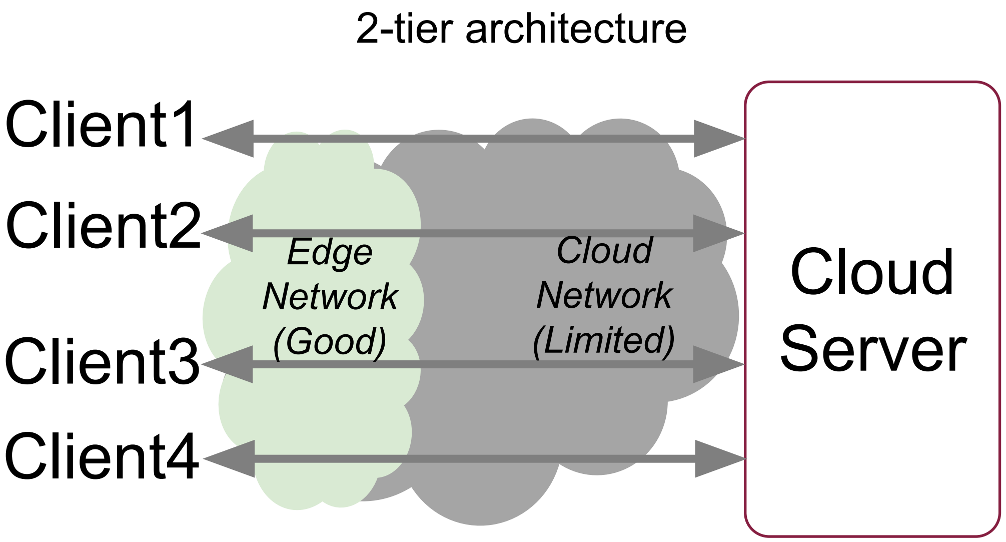
- Edge-based processing benefits

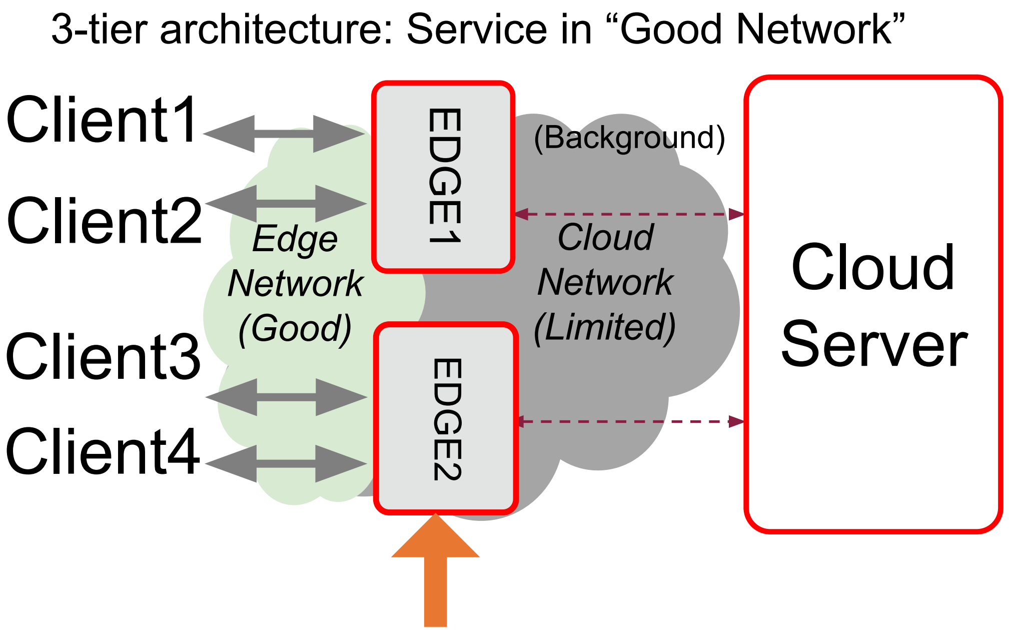
2-tier architecture



Transforming 2-tier into 3-tier architecture

- Benefit from edge-based processing





"Replicas" of Cloud program

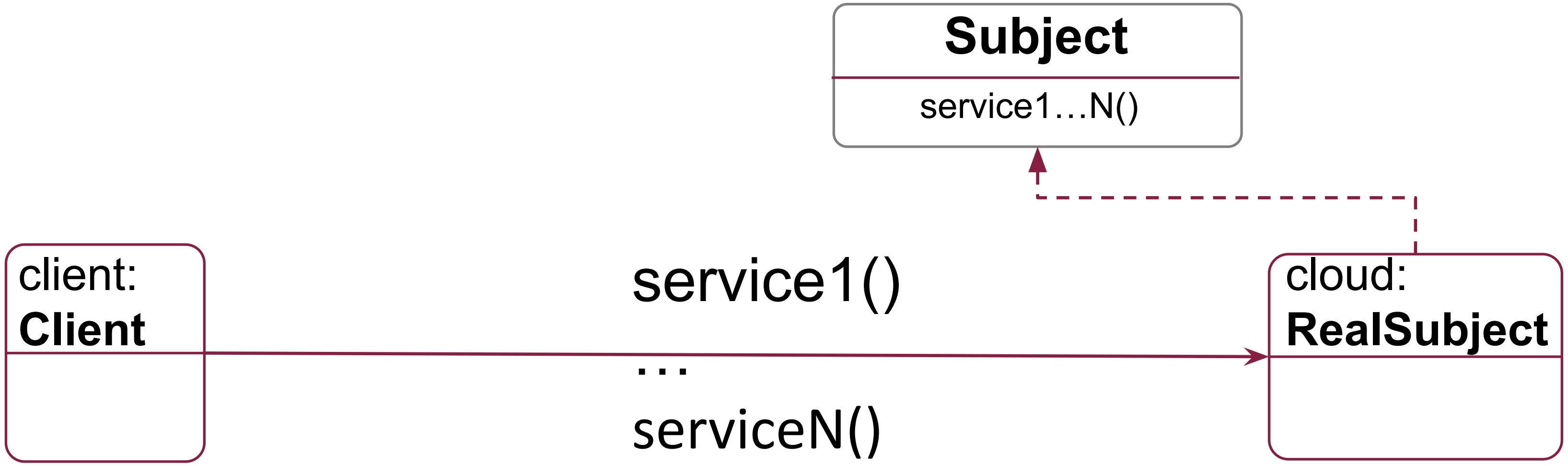
- We understand the benefits, but how to automate the transformation?

Approach Overview

- Replicate a cloud-based service on edge devices
- Select the portion functionality to replicate that improves performance
- Provide eventual consistency by relying on CRDT
- Load-balance to a cluster of edge devices for scalability and throughput

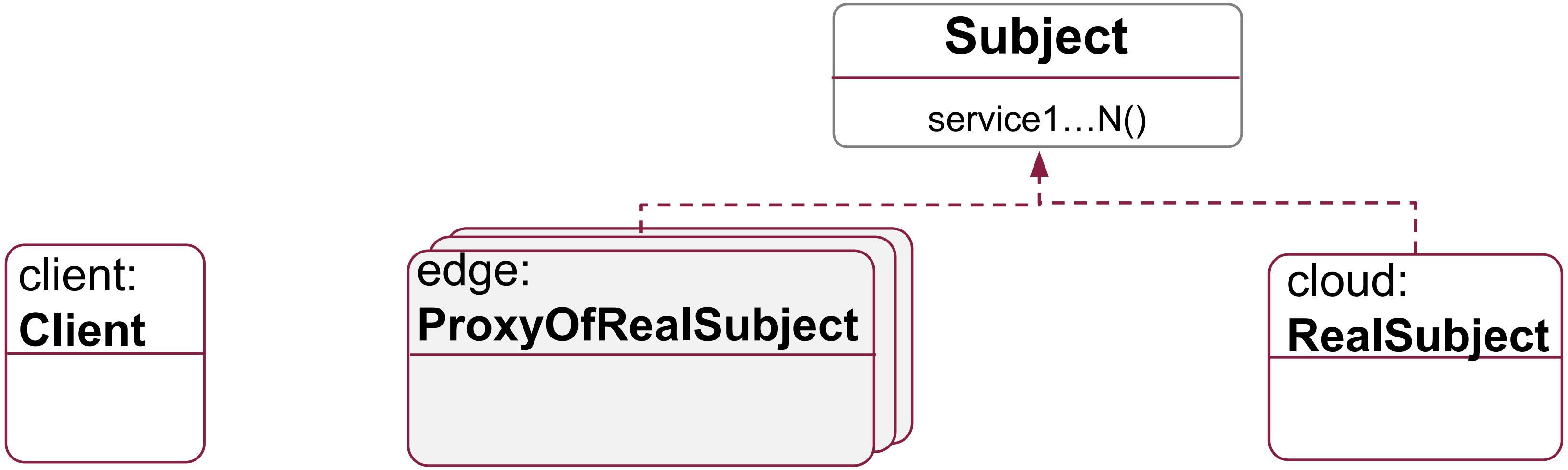
Edge Processing via Tailed Proxy Pattern

- Proxy Pattern: Client makes request to Proxy (edge replicas)



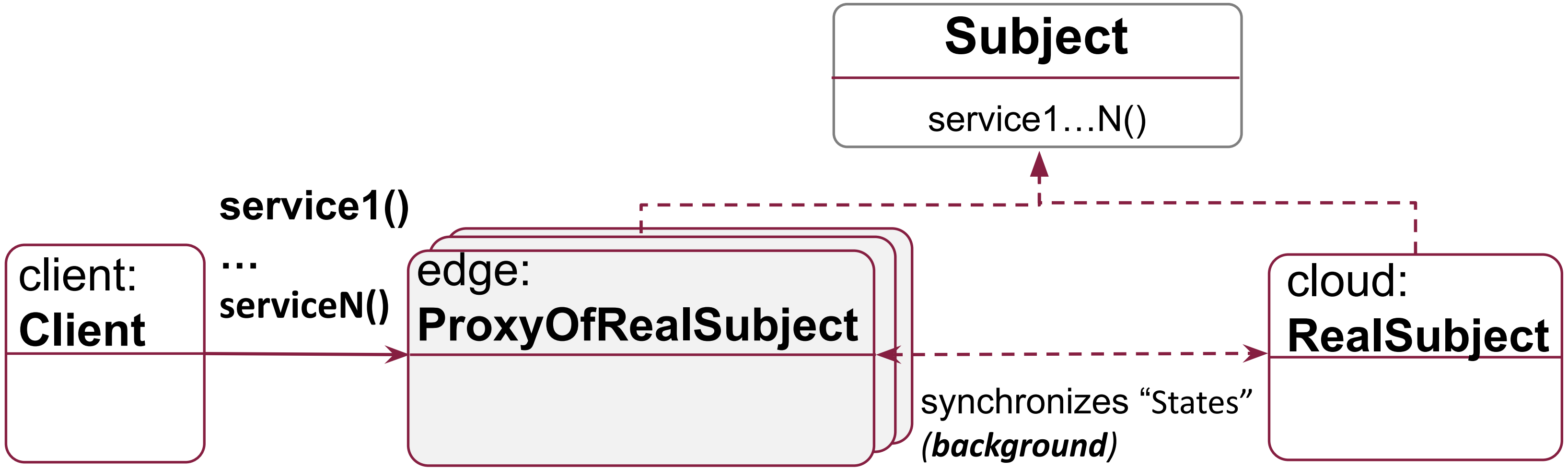
Edge Processing via Tailed Proxy Pattern

- Proxy Pattern: Client makes request to Proxy (edge replicas)



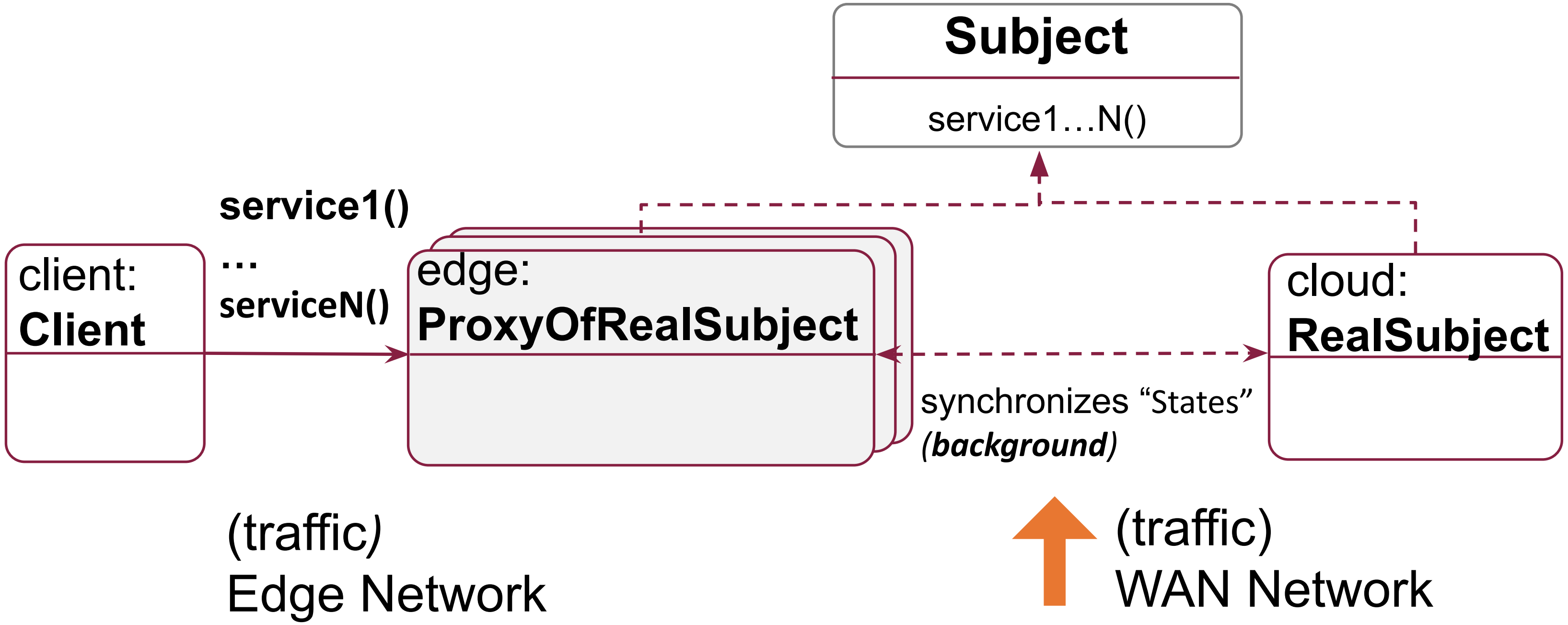
Edge Processing via Tailed Proxy Pattern

- Proxy Pattern: Client makes request to Proxy (edge replicas)



Edge Processing via Tailed Proxy Pattern

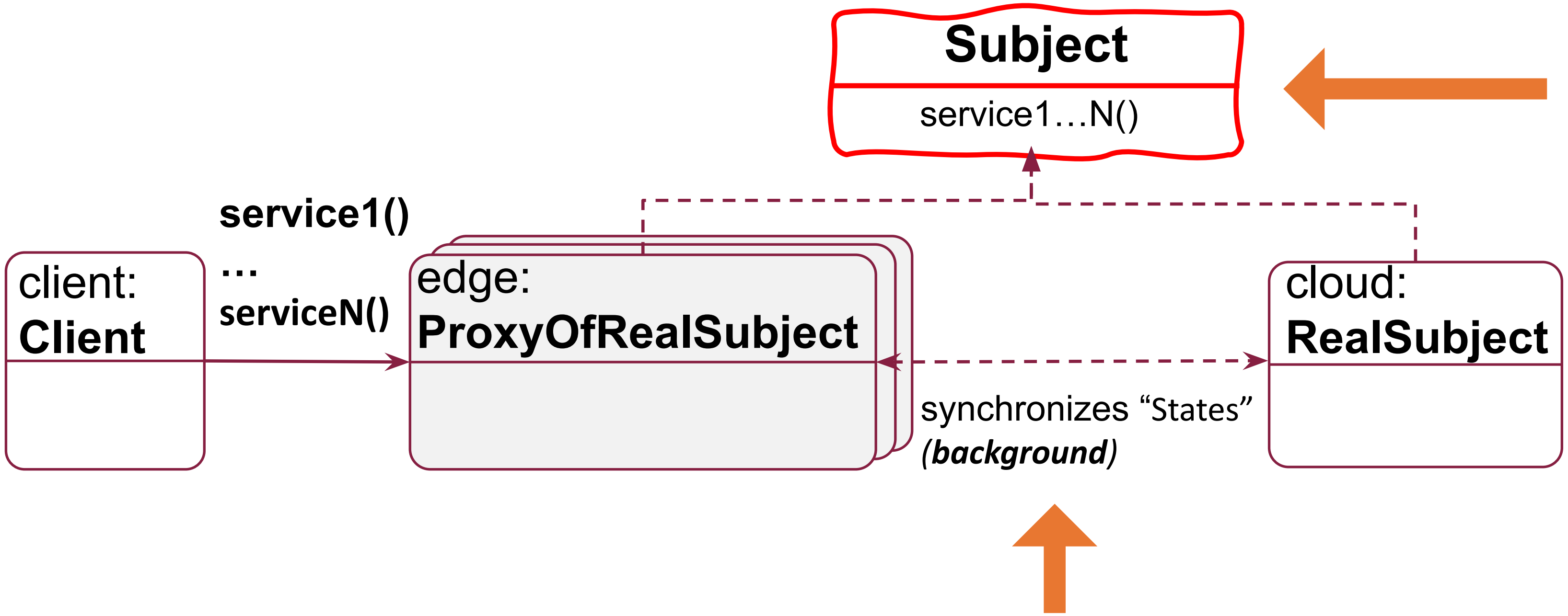
- Proxy Pattern: Client makes request to Proxy (edge replicas)



<u>relaxed consistency</u>:
synchronized in a background process without interfering with main functionality

Edge Processing via Tailed Proxy Pattern

- Proxy Pattern: Client makes request to Proxy (edge replicas)



Subject

service1…N()

client:
Client

service1()
…
serviceN()

edge:
ProxyOfRealSubject

cloud:
RealSubject

synchronizes "States"
(background)

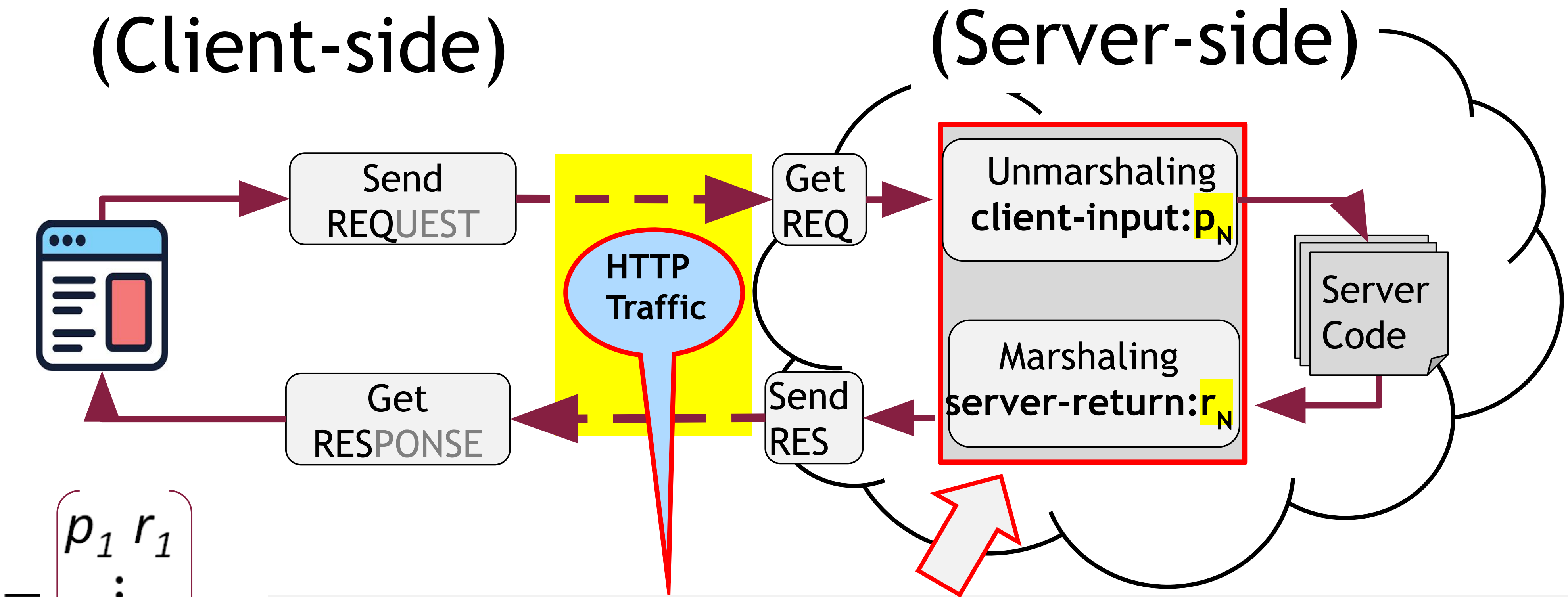
Program Analysis & Transformation:
How to identify and extract required subject functionalities in Cloud program?
: carefully choosing to benefit from edge based processing!

relaxed consistency:
synchronized in a background process without interfering with main processing

EdgStr: Automated Transformation

Identify and extract required subject functionalities in Cloud program

- Identify Subject s_1, s_2,…s_N from by capturing HTTP traffic



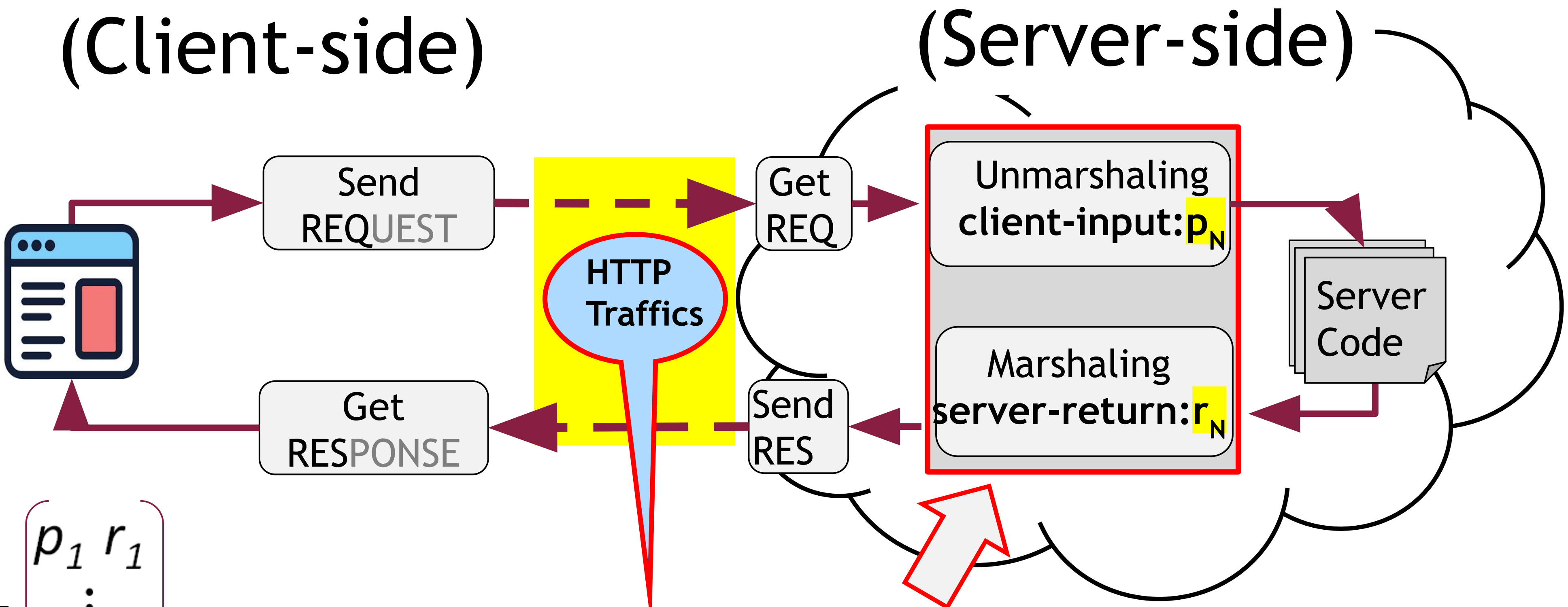
$$\begin{bmatrix} s_1 \\ \vdots \\ s_N \end{bmatrix} = \begin{bmatrix} p_1\ r_1 \\ \vdots \\ p_N\ r_N \end{bmatrix}$$

1. **Identify Subject:** Decode {client-parameter, server-return} □ $S_N(p_N, r_N)$

EdgStr: Automated Transformation

Identify and extract required subject functionalities in Cloud program

- **Extracting** "functionality" from Cloud program:

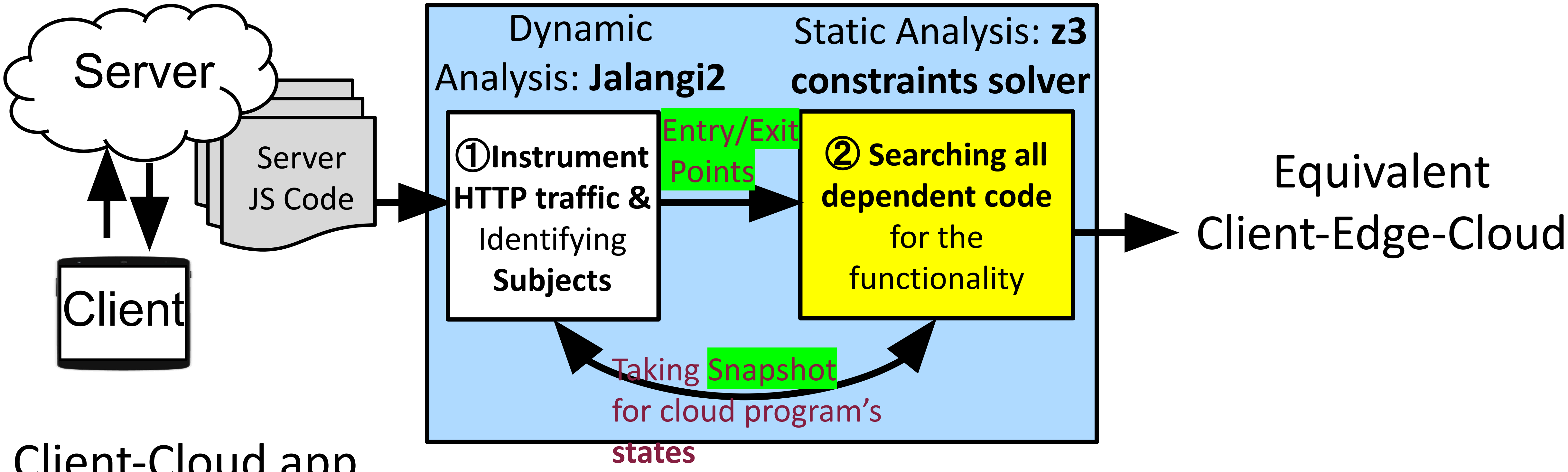


$$\begin{bmatrix} s_1 \\ \vdots \\ s_N \end{bmatrix} = \begin{bmatrix} p_1\ r_1 \\ \vdots \\ p_N\ r_N \end{bmatrix}$$

1. **Identify Subject:** Decode {client-parameter, server-return} □ $S_N(p_N, r_N)$
2. **Extract functionality:** Analyze S_N from Cloud Program based on p_N and r_N

EdgStr: Overall Process

- Dynamic and Static Analysis for Cloud Program



Server

Server JS Code

Client

Dynamic Analysis: **Jalangi2**

Static Analysis: **z3 constraints solver**

①**Instrument HTTP traffic &** Identifying **Subjects**

Entry/Exit Points

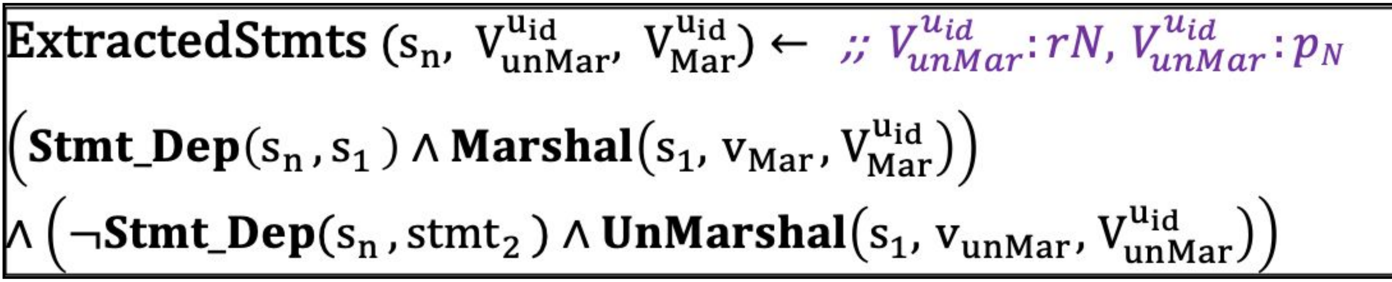
② **Searching all dependent code** for the functionality

Taking Snapshot for cloud program's **states**

Equivalent Client-Edge-Cloud

Client-Cloud app

EdgStr: Overall Process

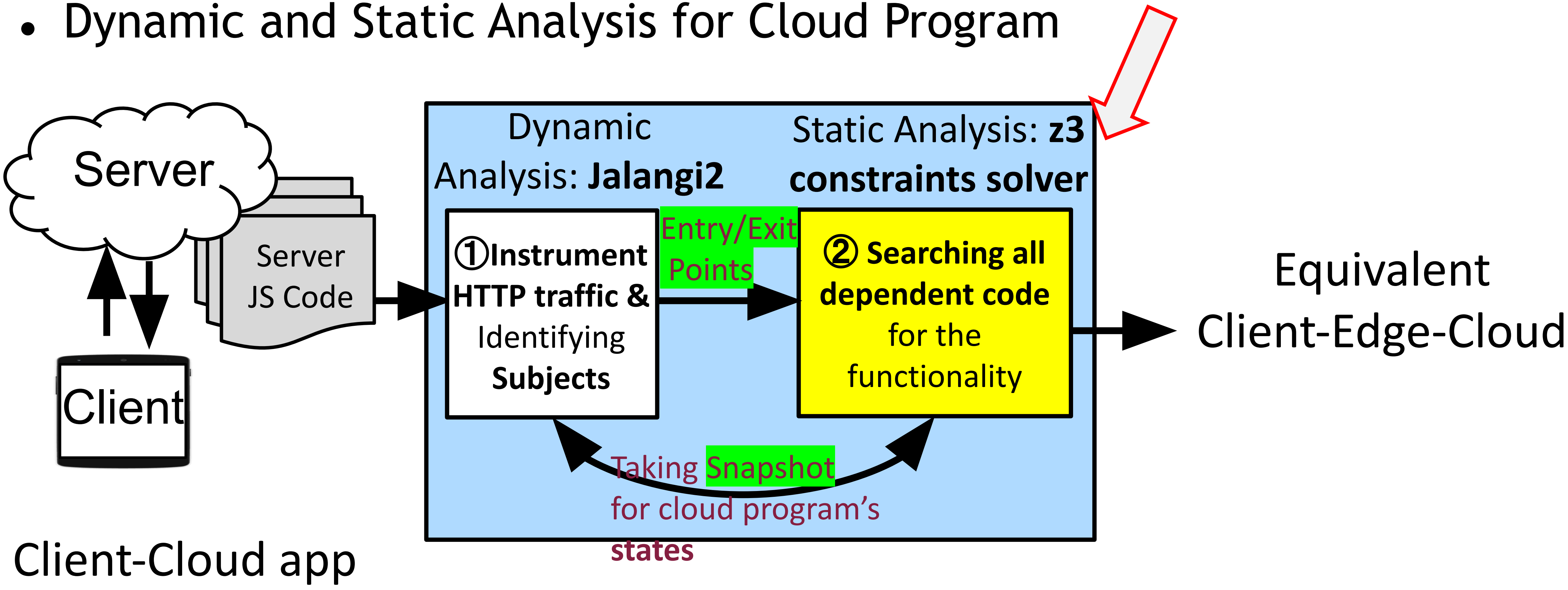


$$\mathbf{ExtractedStmts}\,(s_n,\ V^{uid}_{unMar},\ V^{uid}_{Mar}) \leftarrow \ ;;\ V^{uid}_{unMar}\!:rN,\ V^{uid}_{unMar}\!:p_N$$

$$\big(\mathbf{Stmt\_Dep}(s_n, s_1) \wedge \mathbf{Marshal}(s_1,\ v_{Mar},\ V^{uid}_{Mar})\big)$$

$$\wedge\ \big(\neg\mathbf{Stmt\_Dep}(s_n, stmt_2) \wedge \mathbf{UnMarshal}(s_1,\ v_{unMar},\ V^{uid}_{unMar})\big)$$

- Dynamic and Static Analysis for Cloud Program



Server

Server JS Code

Client

Client-Cloud app

Dynamic Analysis: **Jalangi2**

Static Analysis: **z3 constraints solver**

①**Instrument HTTP traffic &** Identifying **Subjects**

Entry/Exit Points

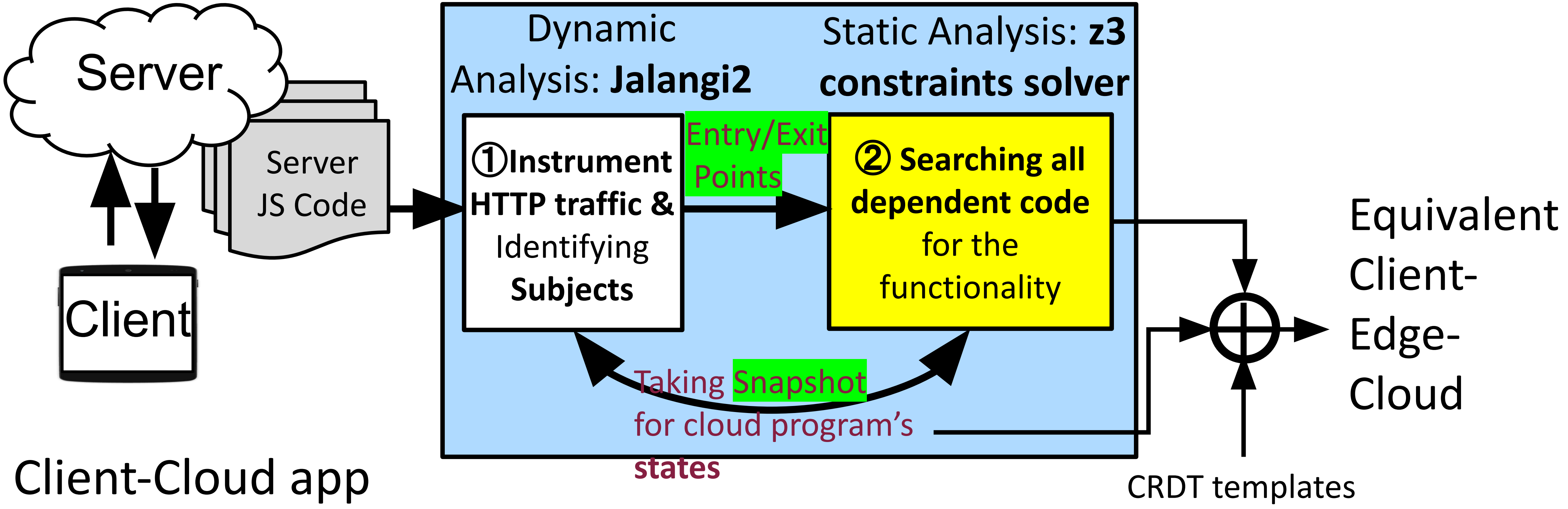
② **Searching all dependent code** for the functionality

Taking Snapshot for cloud program's **states**

Equivalent Client-Edge-Cloud

EdgStr: Overall Process

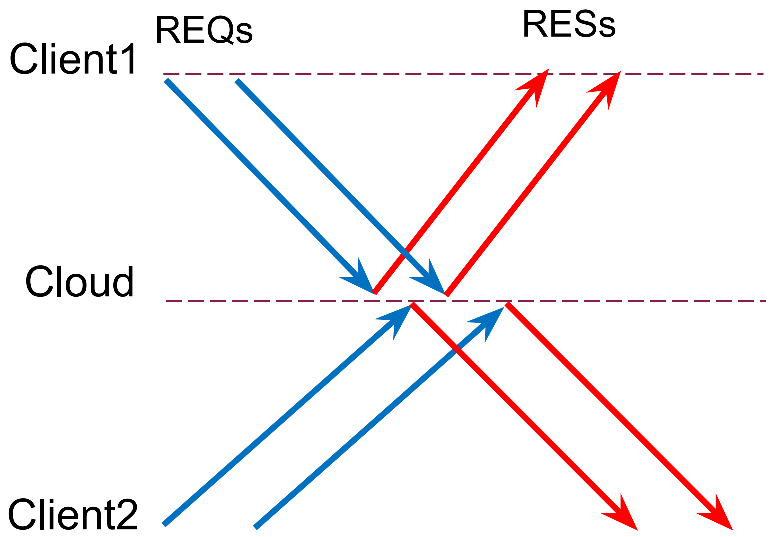
- Dynamic and Static Analysis for Cloud Program



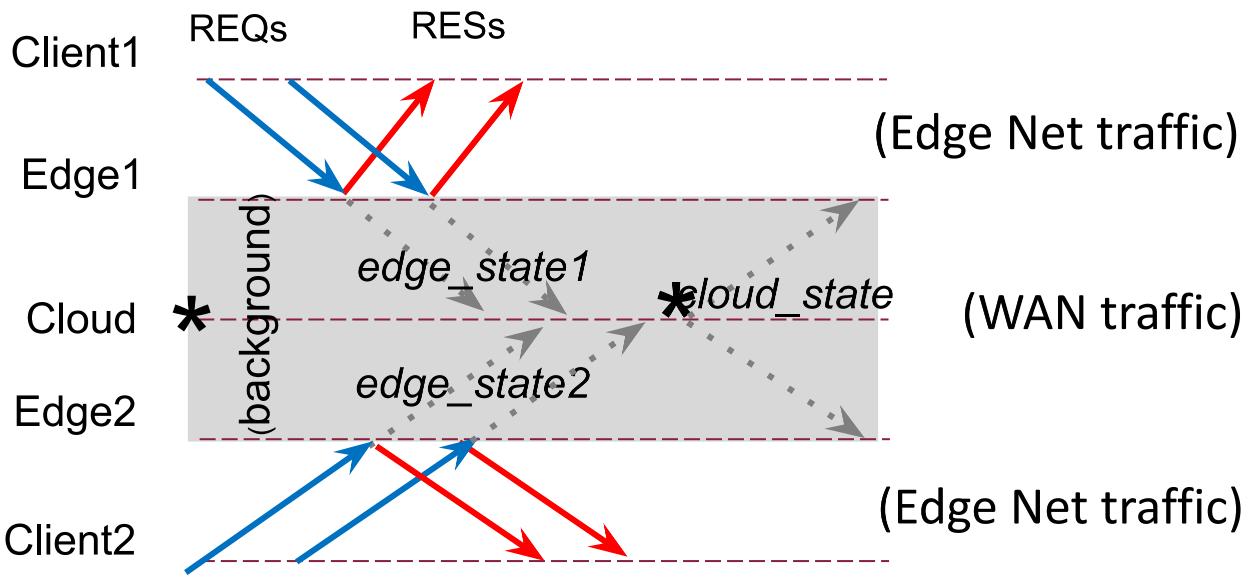
Server
Server JS Code
Client
Client-Cloud app
Dynamic Analysis: **Jalangi2**
Static Analysis: **z3 constraints solver**
①**Instrument HTTP traffic &** Identifying **Subjects**
Entry/Exit Points
② **Searching all dependent code** for the functionality
Taking Snapshot for cloud program's **states**
CRDT templates
Equivalent Client-Edge-Cloud

States are synchronized: Between Cloud and Edge Replicas

- Eventually Consistency Sync. with CRDT for read or write operations across edge replicas and original cloud



Original
Client-Cloud app



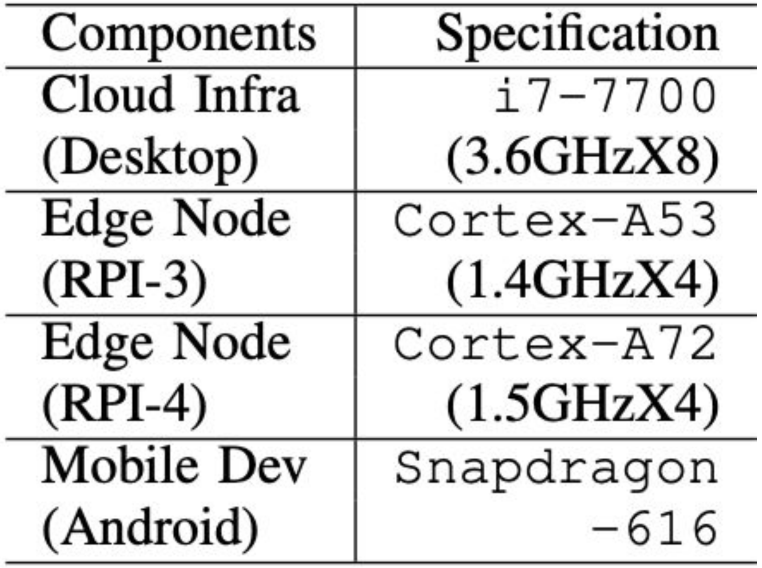
Transformed
Cloud-Edge-Client

Evaluation

- 7 open-source distributed apps (42 remote services)
- Edge Node Setup: RPI-3s and RPI-4s

- **RQ1. Correctness**
- **RQ2. Performance**
- **RQ3. Efficiency**

(comparison with related works)



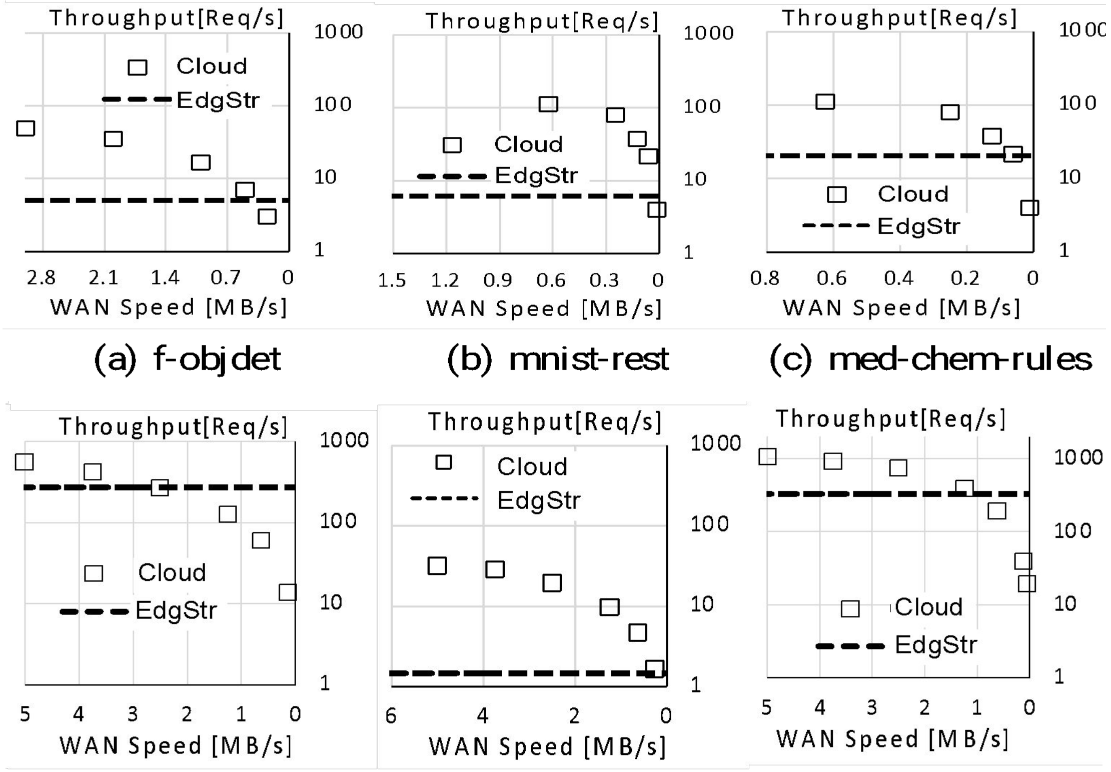
Components	Specification
Cloud Infra (Desktop)	i7-7700 (3.6GHzX8)
Edge Node (RPI-3)	Cortex-A53 (1.4GHzX4)
Edge Node (RPI-4)	Cortex-A72 (1.5GHzX4)
Mobile Dev (Android)	Snapdragon -616

Evaluation (RQ1. Correctness)

- **42/42 was correctly transformed**
  - Given (p_1 , ...p_n) sent to the original service OS and the replicated service RS, check if Ros == Rrs

Evaluation (RQ2: Throughput)

- **Benefit of Edge-based execution** in subjects with
  - Relatively heavy upload/download
  - Low computational loads



(a) f-objdet

(b) mnist-rest

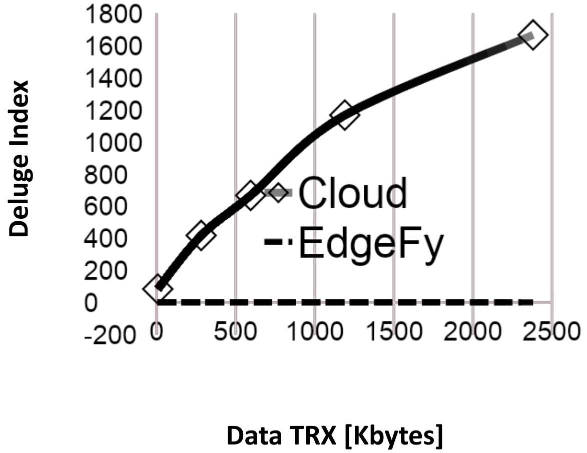
(c) med-chem-rules

(d) ionic2-realty-rest

(e) Bookworm

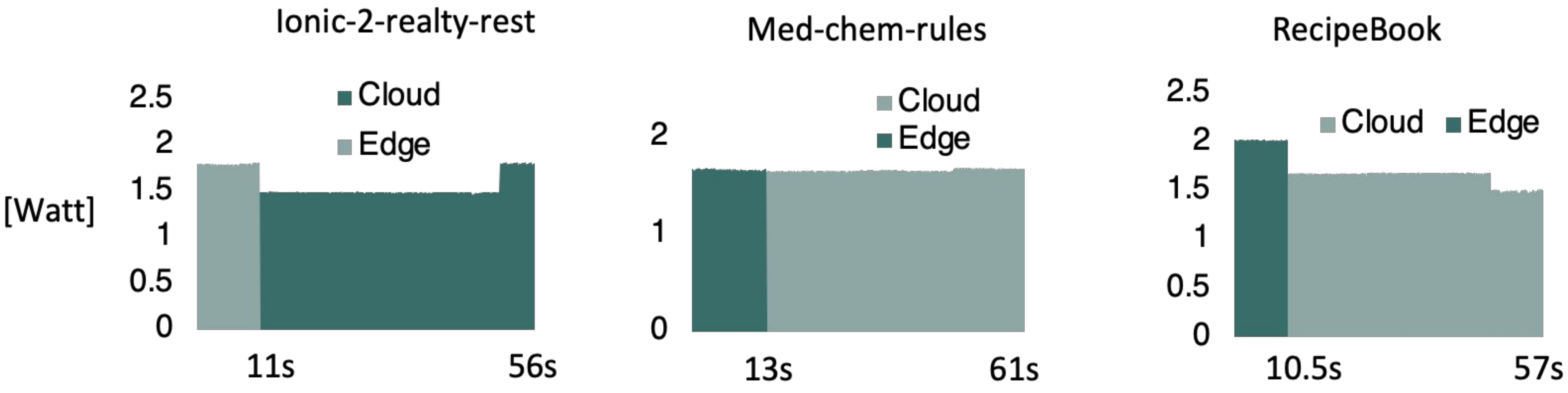
(f) RecipeBook

- **Deluge Index** (ΔNet/ΔTput)
  - The volumes of transmitted data over WAN almost did NOT affect EdStr's throughput



Evaluation (RQ2. Energy Consumption in Client Device)

- The longer it takes to execute a cloud-based, the more client device will end up consuming



Ionic-2-realty-rest

Med-chem-rules

RecipeBook

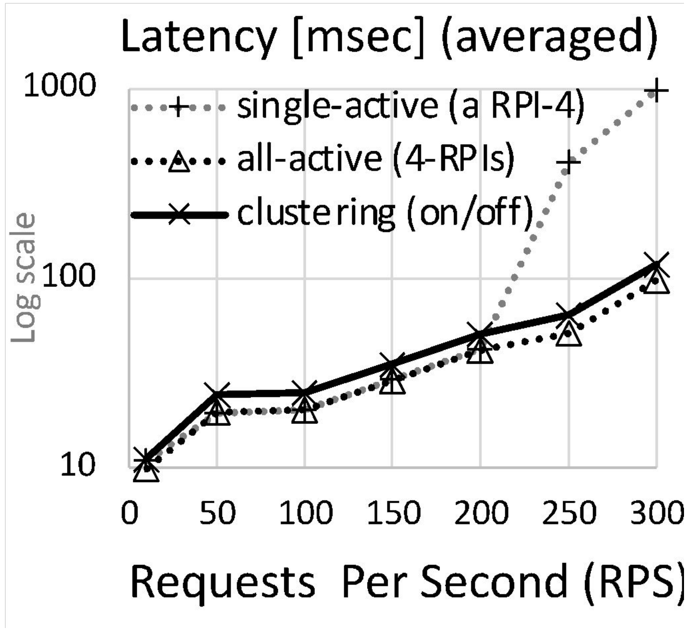
We used Trepn Profiler to measure the consumed energy in Android Device

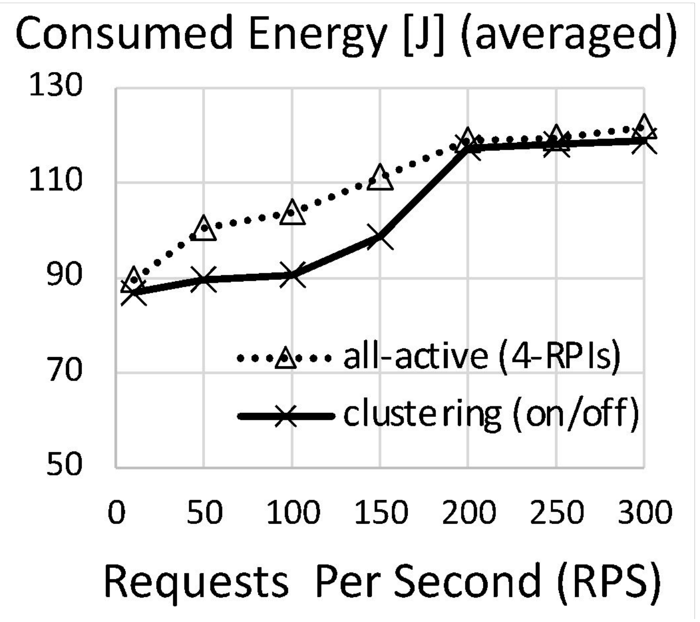
Evaluation (RQ2. Scalability and Elasticity of Edge-based processing)

- Built a cluster using 4 RPIs: distributing clients’ requests to available edge replicas

2 RPI-3s and 2 PRI-4s

- Load balancer shuts on or off the RPIs based on service utilization (clustering on/off)

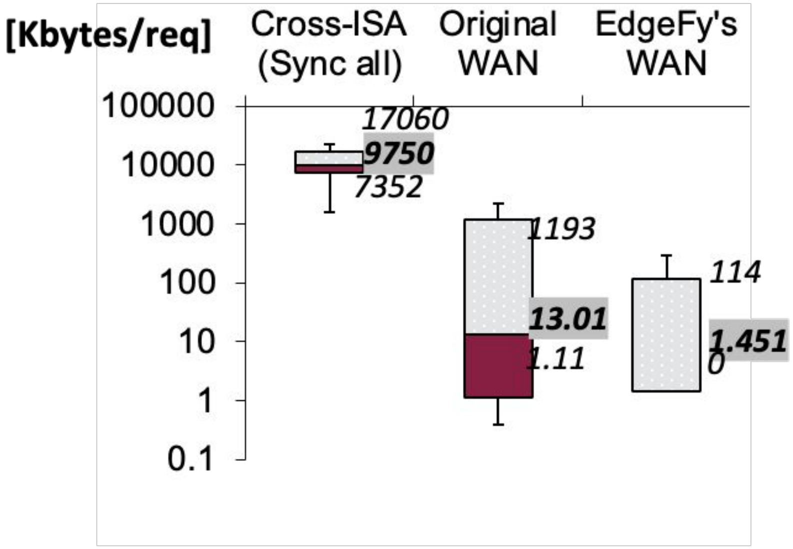




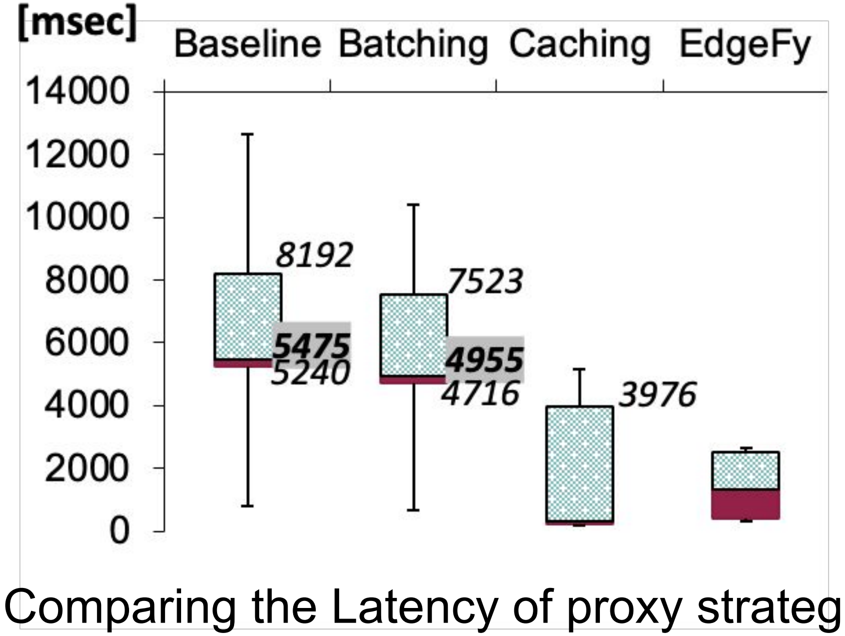
Active replicas gradually changed from 4 to 1, reducing overall consumed energy by as much as **12.96%**

Evaluation (RQ3. effectiveness of EdgStr's sync and proxying strategy)

- **Cross-ISA** offloading systems [25,26,27] inefficiently **syncs all states** of cloud program
- Proxy Caching [28,29] benefits in **read-mostly services**
- Batching [31,32] only **reduces WAN traffics** through request aggregation



Sync Overhead and WAN traffic analysis



Comparing the Latency of proxy strategies

Cross-ISA vs EdgStr: EdgStr minimizes the amount of synchronization traffic over WAN by synchronizing only the *modifiable* parts of the replicated service state.

Conclusion and Q/A

- We described and evaluated EdgStr’s advanced program analysis and transformation techniques
  - from 2-tier client-cloud to 3-tier client-edge-cloud

- Applying EdgStr to representative distributed mobile apps introduces the performance benefits of edge processing, without the high costs of manual program transformation

Applicability & Limitation for **EdgStr**

- Subject: Cloud Services (targeting important domain in Node.js)
- RESTful HTTP protocols
  - Executions: HTTP Request/Response, GET/POST/...
  - What else? Socket.IO, gRPC, ...
- ☞ **Cloud Server State Replications**
  - DataBase with **SQL**, **Files**, and **global variables**
  - What else? (Future work) framework specific Data Structures or **ML Models**
    - Federated Learning for replicating **ML Models** across cloud and edges