The Client Insourcing Refactoring to Facilitate the Re-engineering of Web-Based Applications

KIJIN AN

APRIL 28, 2021

Committee Members

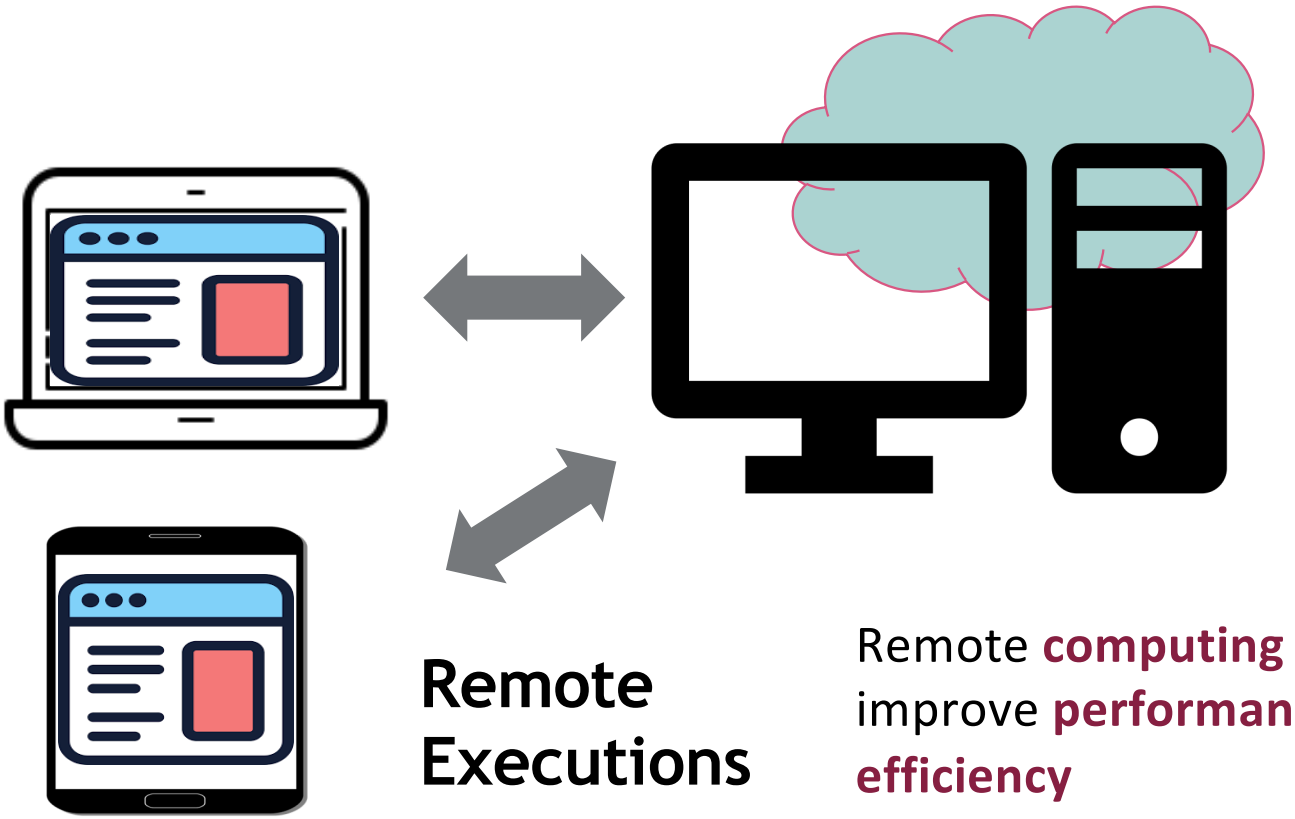
Eli Tilevich, *Virginia Tech* (Chair)
Godmar Back, *Virginia Tech*
Walter Binder, *University of Lugano*
Xun Jian, *Virginia Tech*
Francisco Servant, *Virginia Tech*

VIRGINIA TECH™

Dissertation Outline

- **Distributed Applications** and **Evolutionary Modifications**
- Motivation of **Client Insourcing Refactoring**
- **Applicability** and **Limitations** of Client Insourcing
- Client Insourcing and its **Applications**
- **Future work** directions
- Conclusion

Distributed Applications

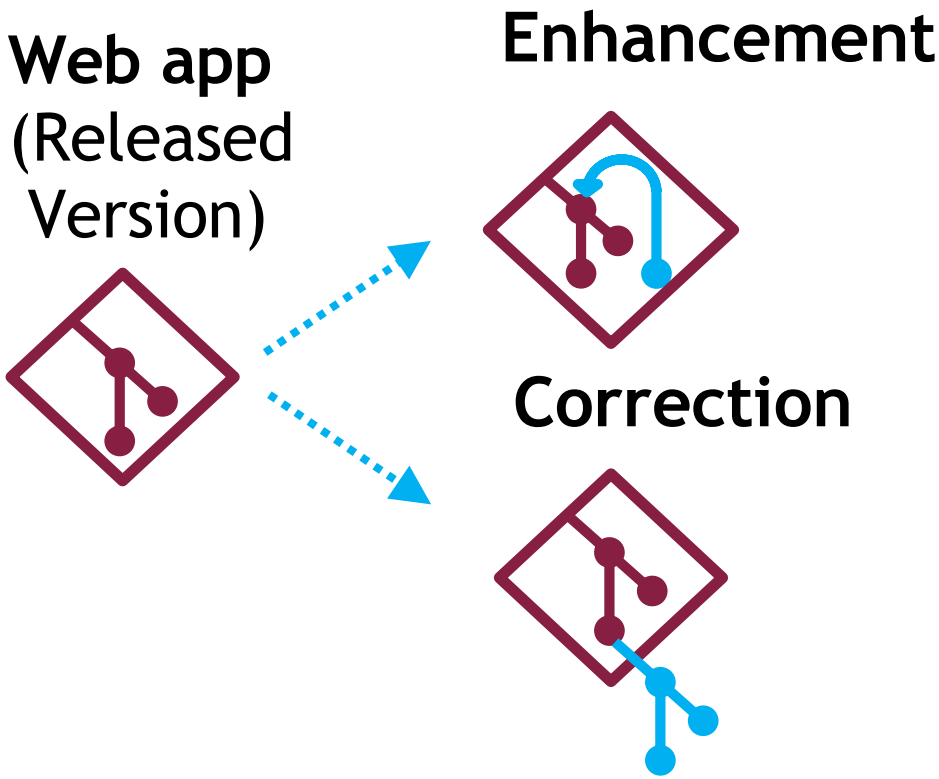


Remote Executions

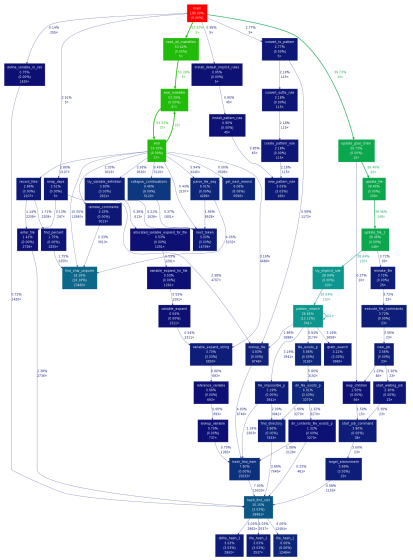
Remote **computing resources** improve **performance** and/or **efficiency**

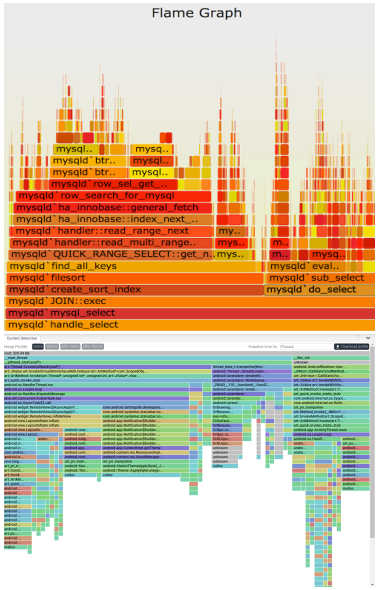
Evolving Distributed Web Apps

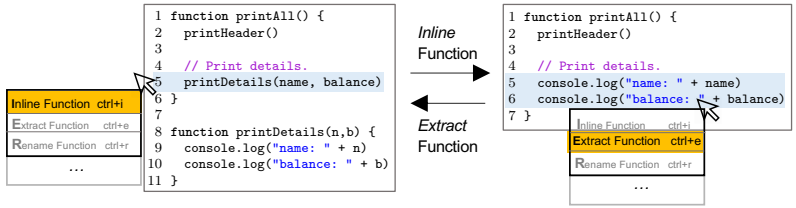
- Why hard and error-prone?
  - Complex control flows (middleware libraries and frameworks)
  - Server misconfigurations
  - Network volatility



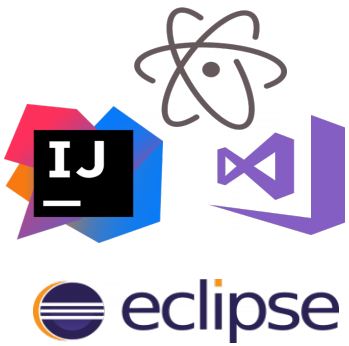
Re-engineering Tools



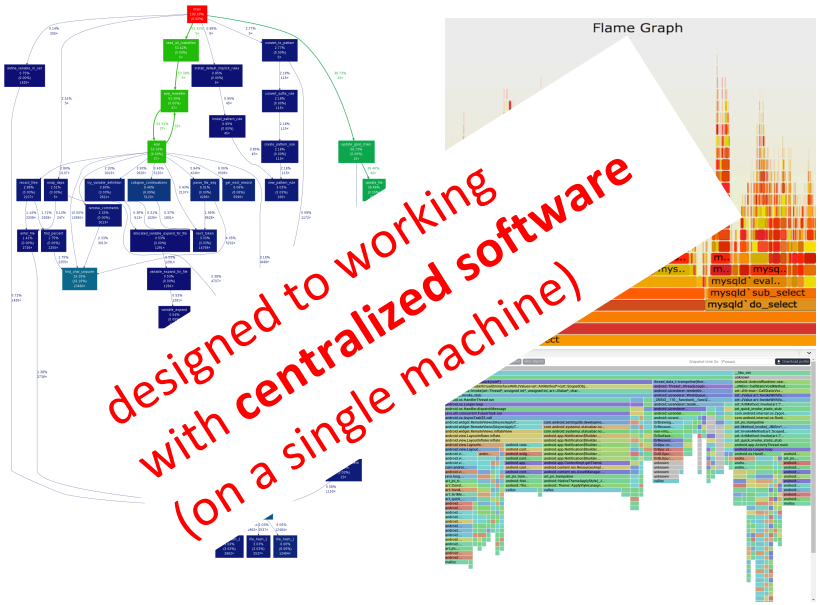


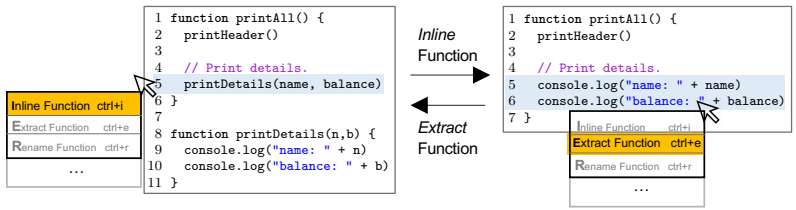


- Memory/CPU Profilers, Debuggers
- Refactoring Tools
  - rename, extract/inline functions, etc.
- Being parts of modern IDEs



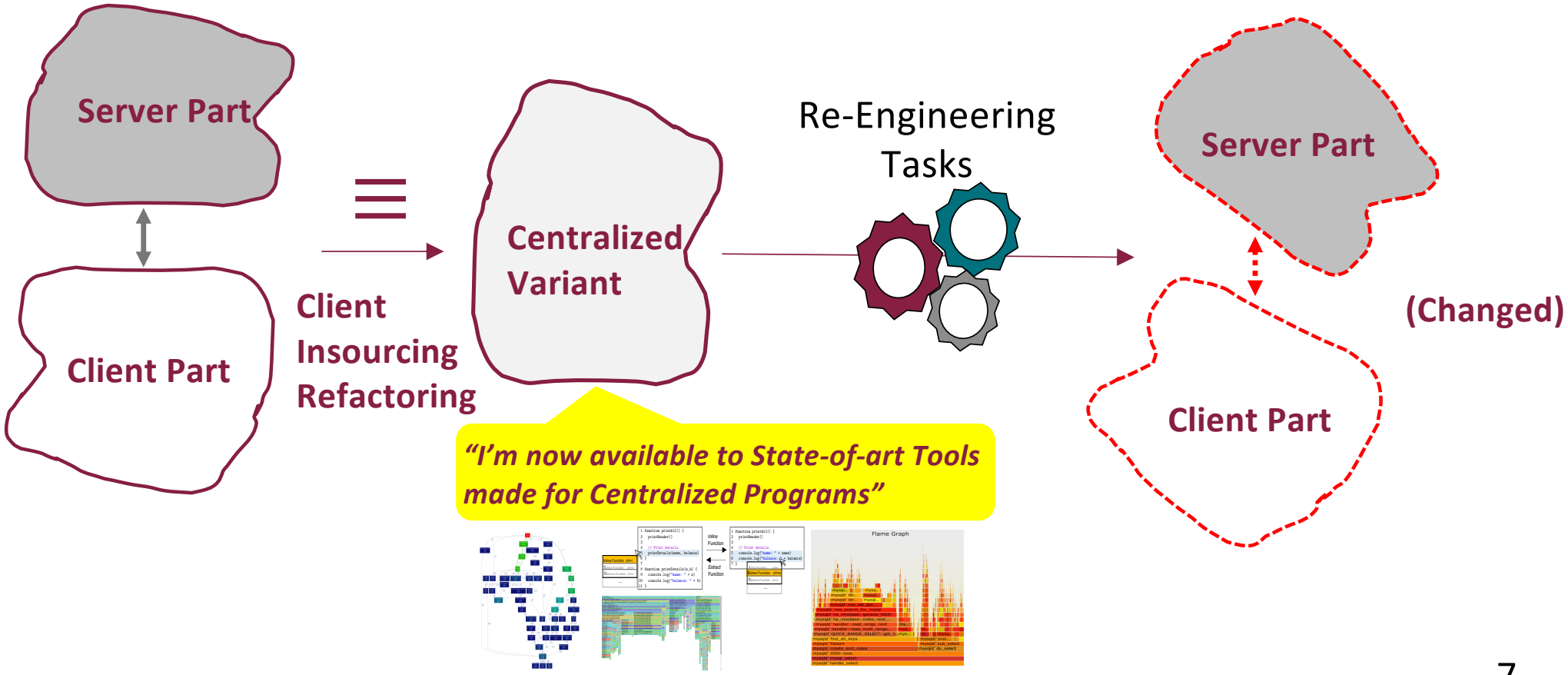
Re-engineering Tools





- Memory/CPU Profilers
- Refactoring Tools
  - renaming, extract/inline functions, etc
- Major Disconnect between **modern software apps** and mature **software re-engineering tools**
- How about Research frameworks?
  - Cloud Refactoring [23], ZQ Compiler [21], CCReplay [17], BrowserSix [42]
  - Mostly working on Centralized programs

Client Insourcing Refactoring



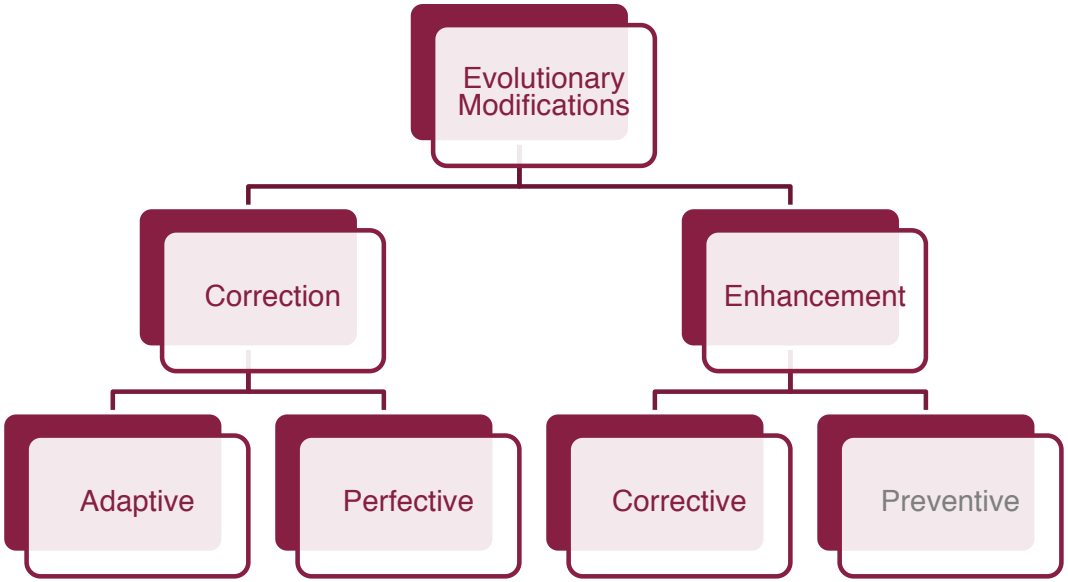
Dissertation Contributions

- Introduce new refactoring: *Client Insourcing*
- Demonstrate the value and utility of Client Insourcing in
  - Correcting
  - Enhancing
  - Optimizing

  of Distributed Web Applications

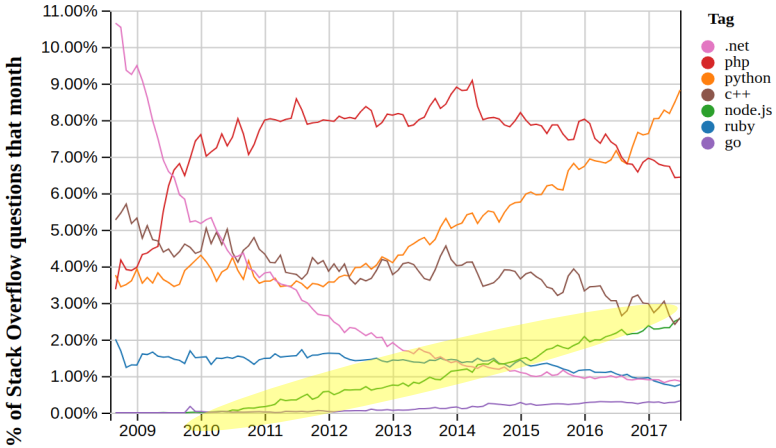
Applications of Client Insourcing

- Re-engineering Distributed Web Apps with Client Insourcing
- Dissertation is based on
  1. Client Insourcing Refactoring [WWW 2020]
  2. **Debugging** Full-Stack Apps [ICWE 2019]
  3. Correcting **Distribution Granularity** [SANER 2020]
  4. **Adapting** Full-Stack Apps for Responsiveness [ICWE 2021]



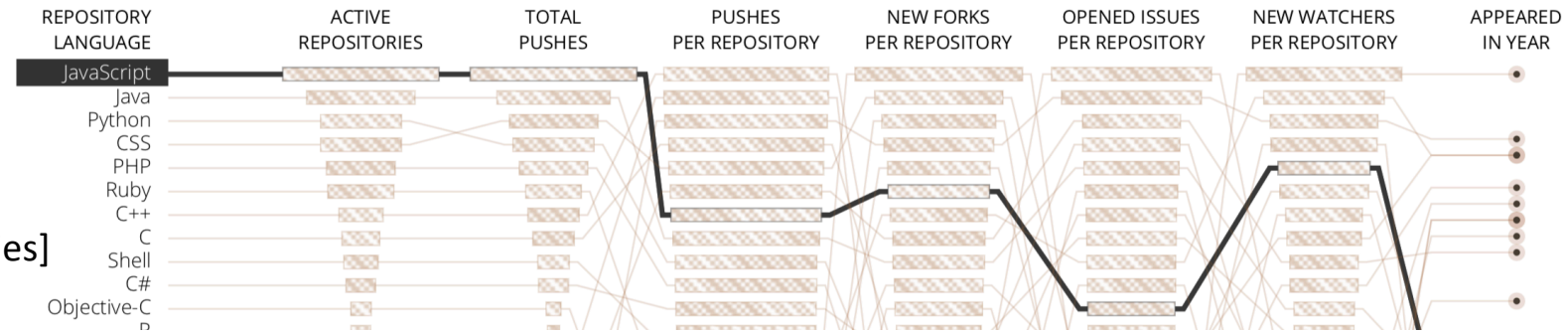
Applicability & Limitation of Client Insourcing

- ☞ **Subject Domain**
  - Full-Stack JavaScript apps (Node.js Backend)
  - To catch up with rapidly growing user base, the backends follow a lower load time development
  - Netflix, Uber, LinkedIn,…
  - Popular in Backend and open source



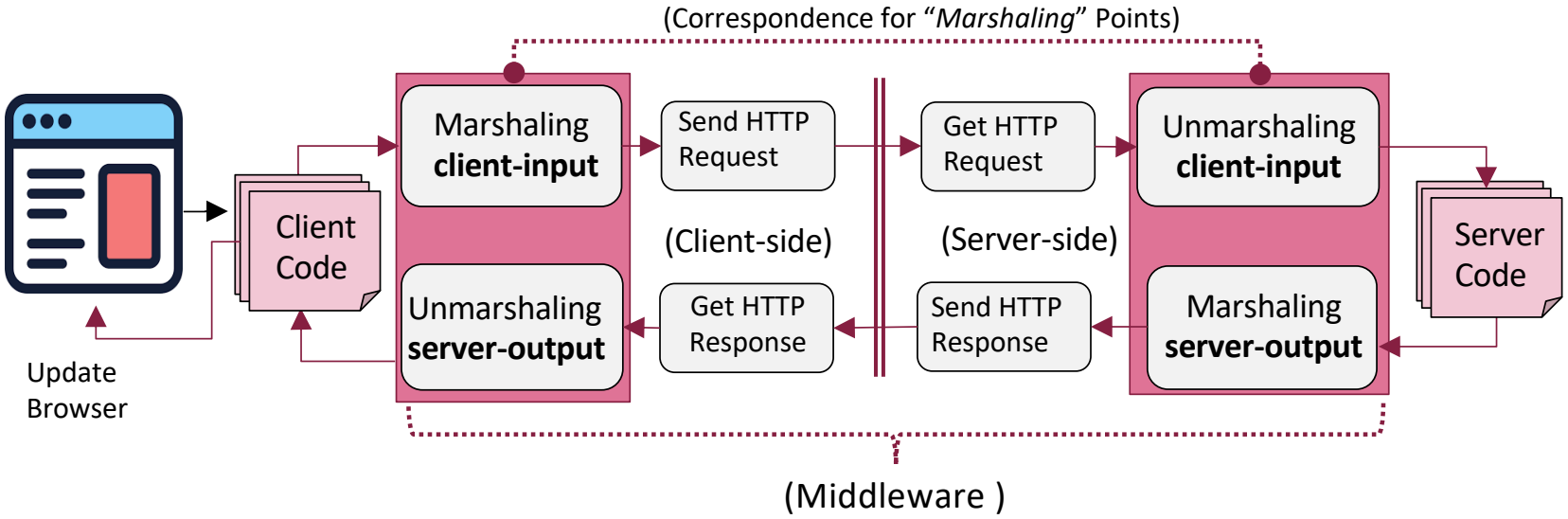
[Trend in JS Backend Developments]

[Trend in Github Repositories]



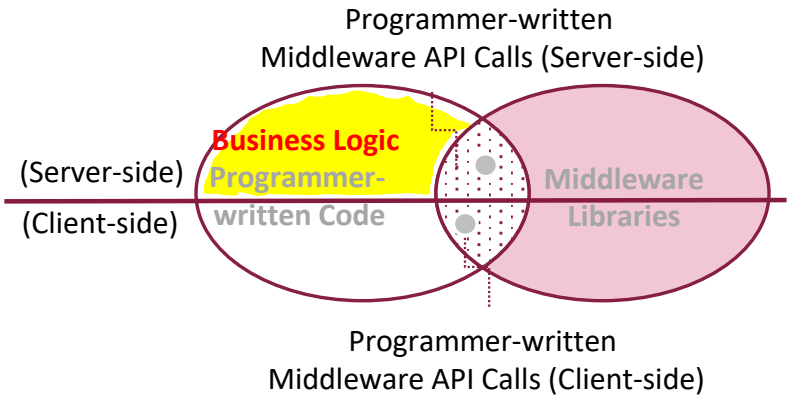
Applicability & Limitation of Client Insourcing

- **Subject Domain: Full-Stack JS apps**
- ☞ **RESTful HTTP protocols**
  - Executions: HTTP Request/Response, GET/POST/...
  - What else? Socket.IO, gRPC, ...



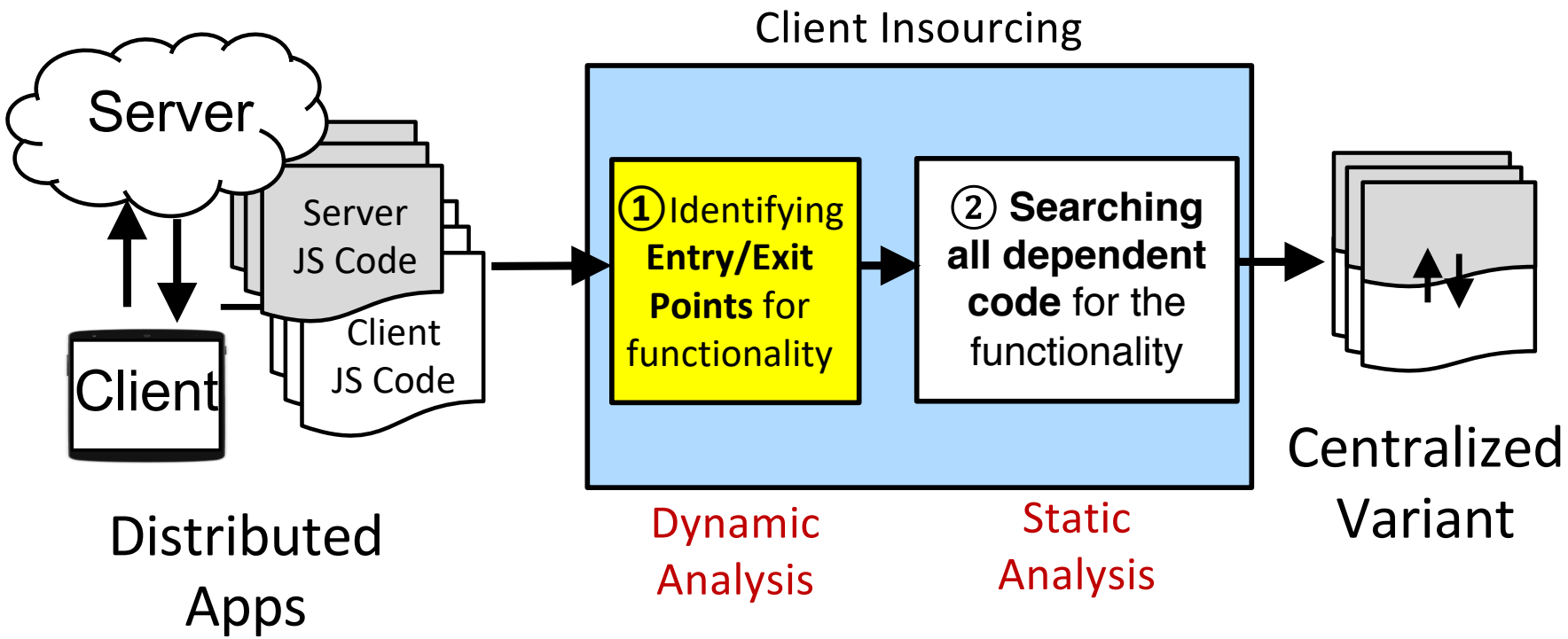
Applicability & Limitation of Client Insourcing

- **Subject Domain: Full-Stack JS apps**
- **RESTful HTTP protocols**
  - Executions: HTTP Request/Response, GET/POST/...
  - What else? Socket.IO, gRPC, ...
- ☞ **Insourcing Business Logic only**
  - What else? *Failure/Exception handling Logics*
- ☞ **Server State Isolations/Replications**
  - Database with **SQL**, **Files**, and **global variables**
  - What else? *Framework specific CRUD/Data Structures*



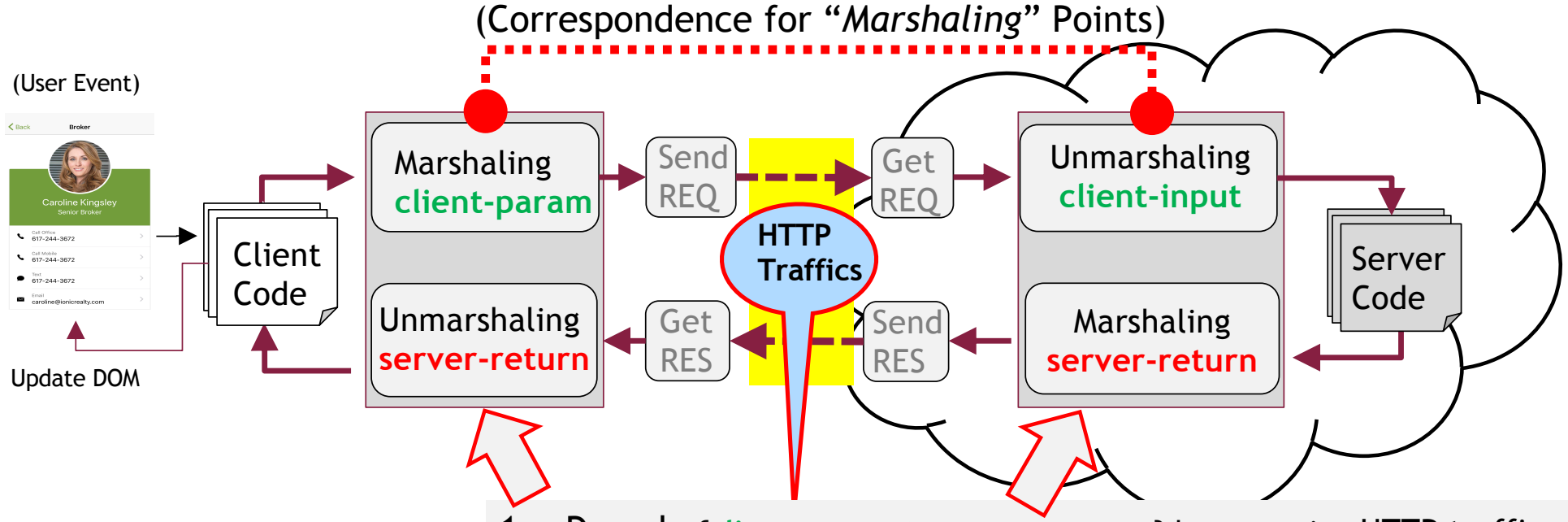
Client Insourcing Refactoring [WWW 2020]

- Phrase1: Identify Entry/Exit Points of Remote Functionality



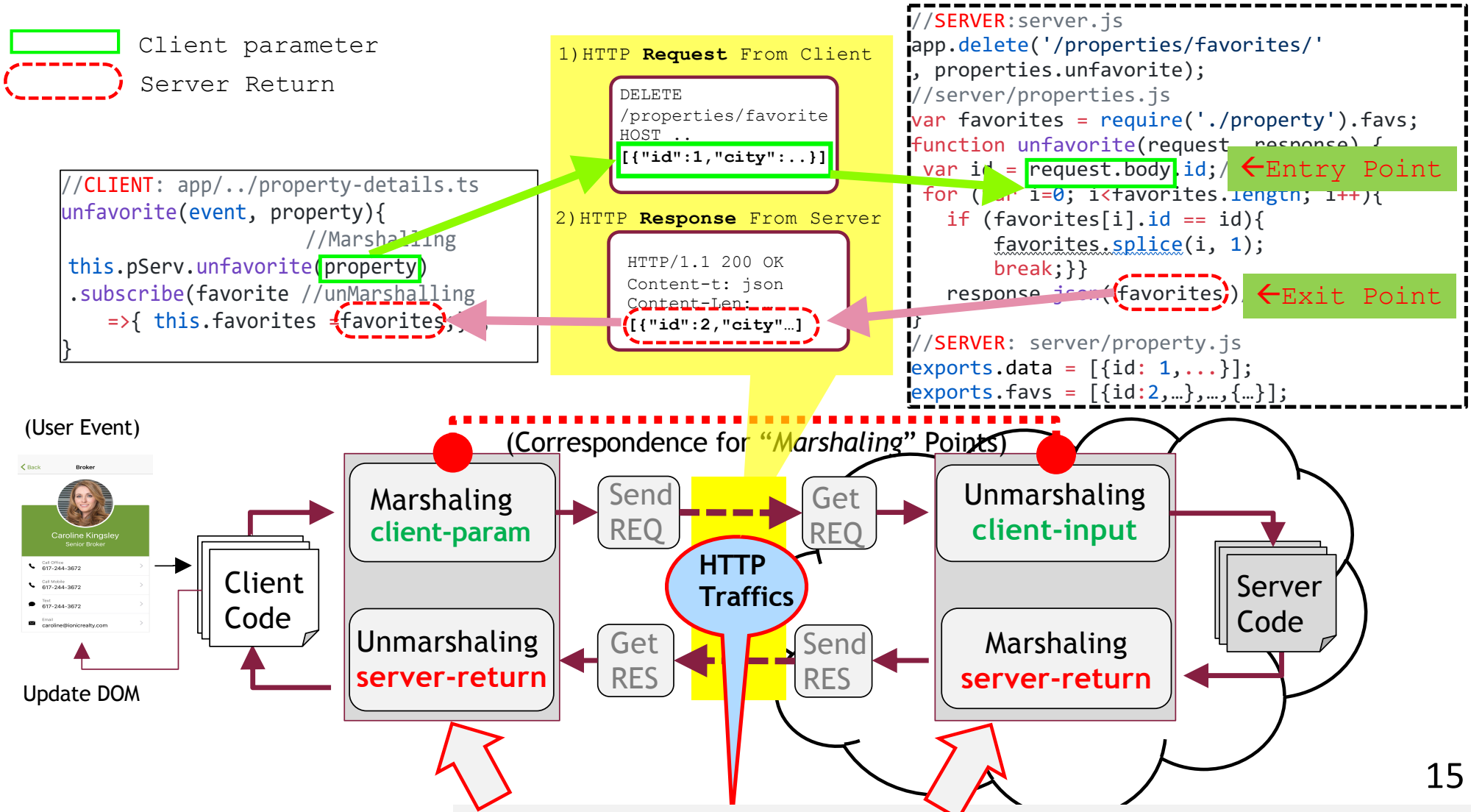
Client Insourcing Refactoring [WWW 2020]

- Execution Model for Distributed App: *Marshaling*/*UnMarshaing*
  - Identify Entry/Exit Points of Remote Execution both in Client/Server



(Client-side)

1. Decode {client-parameter, server-return} by capturing HTTP traffics
2. Instrument code parts that RWs these values



15

(Client-side)

1. Decode {client-parameter, server-return} by capturing HTTP traffics
2. Instrument code parts that RWs these values (jalangi)

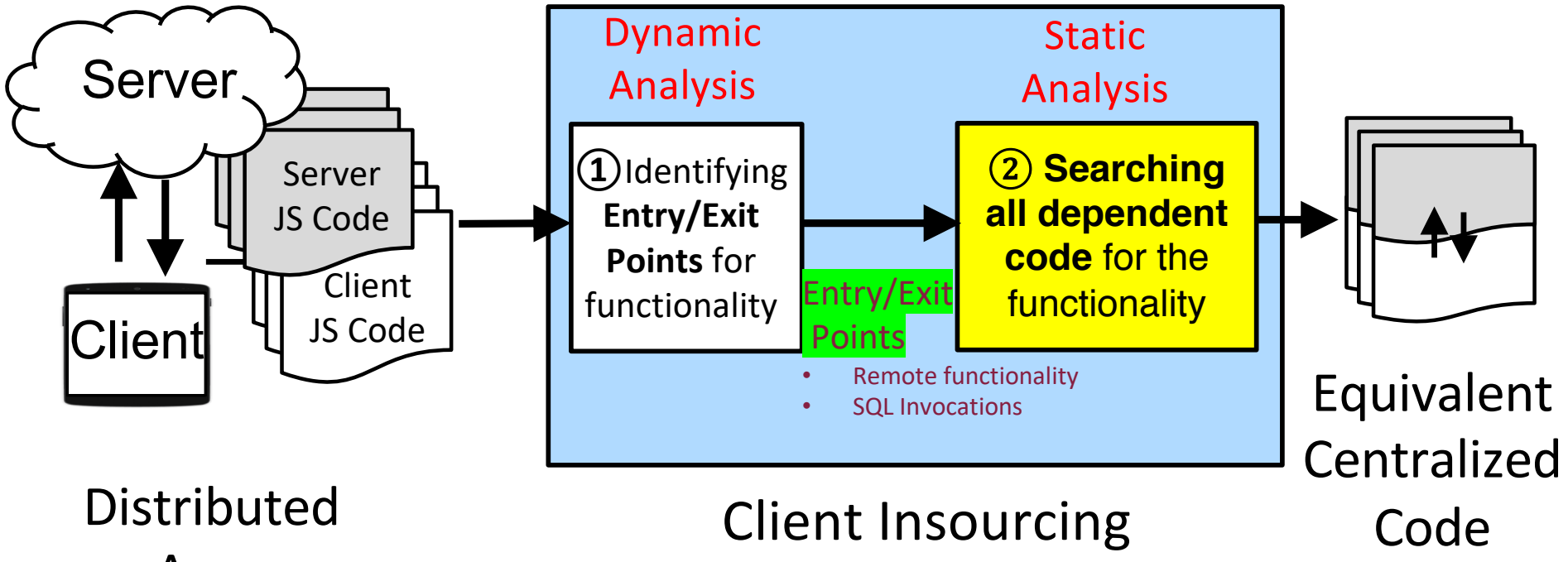
Client Insourcing Refactoring [WWW 2020]

- Marshaling its **response** output differently, **Entering** the remote ftn through a **different point**

- 1. Achieving the **Idempotency** Executions (jalangi framework)
  - Testing HTTP REQ **changes** the server **states**
  - “Restore” ops, interleaving between executions
- 2. **Fuzzing** HTTPs (de/encoding HTTP records)
  - Fuzzing the original HTTP for built-in primitive types
  - e.g. 1->90001, “str1”->“str1_JSRCIStr”

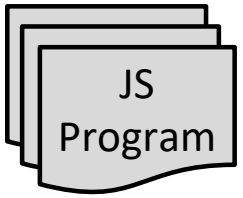
Client Insourcing Refactoring [WWW 2020]

- Phrase2: Searching all Dependent JavaScript Statements



Client Insourcing Refactoring [WWW 2020]

- Searching all dependent JS code in Entry/Exit points
- Extending *Declarative approach for JavaScript* Analysis
  - **Logically** representing/analyzing **JS program** by means of *facts* and *rules*
  - **GATEKEEPER** [USENIX Security'09]: Point-to-Analysis
  - **JSDep** [FSE'15]: JS Stmts Dependency Analysis (extension of GATEKEEPER)



s_1 → ***var*** *v_a*="*DATA*";
s_2 → ***fuction*** abc(){..
var *v_b*=*v_a*;
}

→

Facts

$\mathbf{Write}(s_1, v\_a)$
$\mathbf{Read}(s_2, v\_a)$
$\mathbf{Write}(s_2, v\_b)$ ...

;;Generated from a JS Program

Rules

$\mathbf{DataDep}(stmt_1, stmt_2) \leftarrow$
$\mathbf{Read}(stmt_1, v_1) \wedge \mathbf{Write}(stmt_2, v_1)$

Query $\mathbf{DataDep}(stmt_1, s_2)$
>> s_1

Client Insourcing Refactoring [WWW 2020]

- Detecting Un-/Marshaling Point in Server Program

;;Rule for Detecting Entry Point at the Server

$$\mathbf{UnMarshal}(\mathrm{stmt}_1, \mathrm{v}_{\mathrm{unMar}}, \mathrm{V}^{\mathrm{u_{id}}}_{\mathrm{unMar}}) \leftarrow \mathbf{Write}(\mathrm{stmt}_1, \mathrm{v}_{\mathrm{unMar}}) \wedge \mathbf{Ref}(\mathrm{v}_{\mathrm{unMar}}, \mathrm{V}^{\mathrm{u_{id}}}_{\mathrm{unMar}})$$

Generated Fact From JSDep

Generated Fact From Stage 1

;;Rule for Detecting Exit Point at the Server

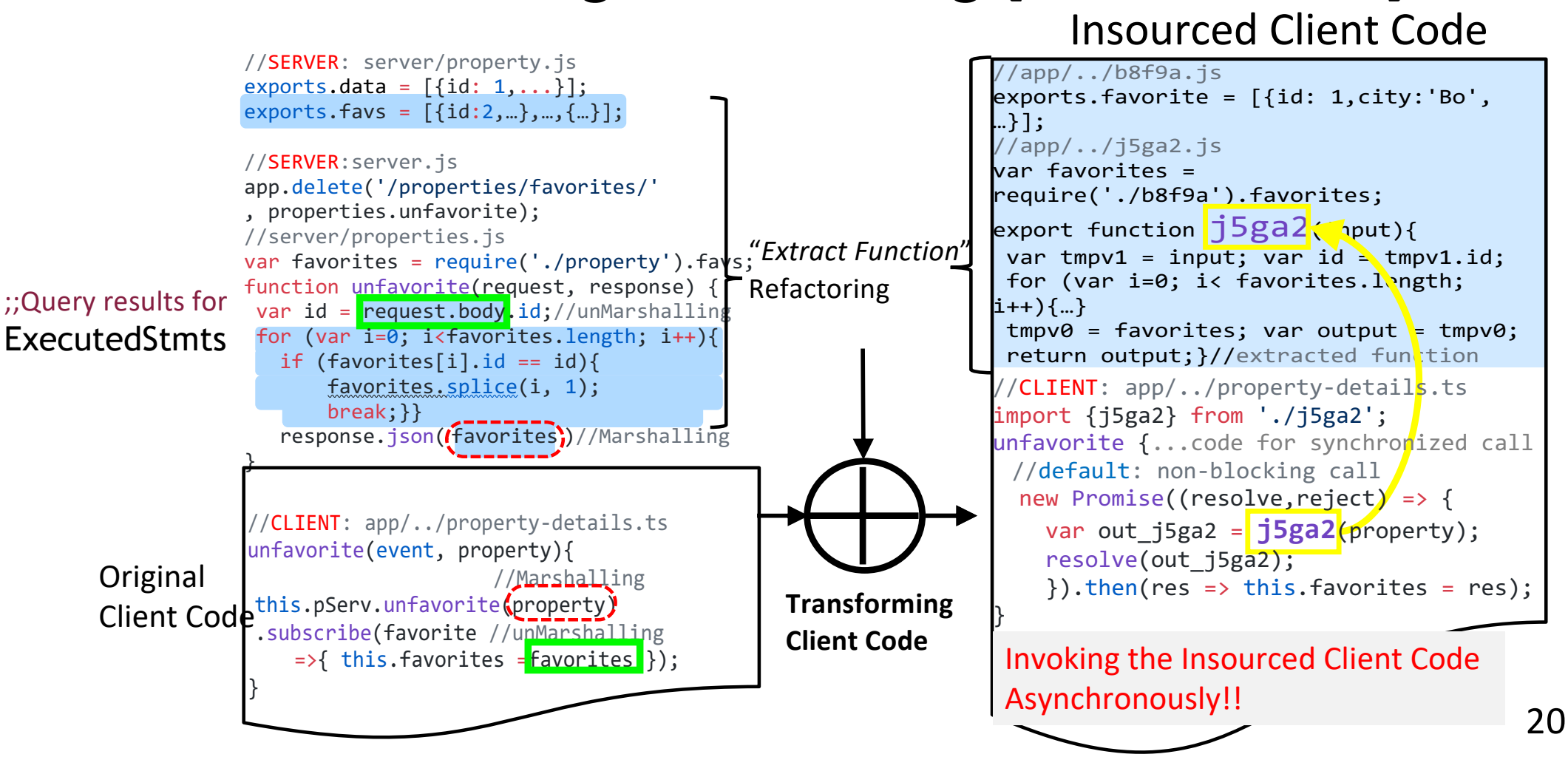
$$\mathbf{Marshal}(\mathrm{stmt}_1, \mathrm{v}_{\mathrm{Mar}}, \mathrm{V}^{\mathrm{u_{id}}}_{\mathrm{Mar}}) \leftarrow \mathbf{Write}(\mathrm{stmt}_1, \mathrm{v}_{\mathrm{Mar}}) \wedge \mathbf{Ref}(\mathrm{v}_{\mathrm{Mar}}, \mathrm{V}^{\mathrm{u_{id}}}_{\mathrm{Mar}})$$

- Querying JavaScript Statements to extract the functionality

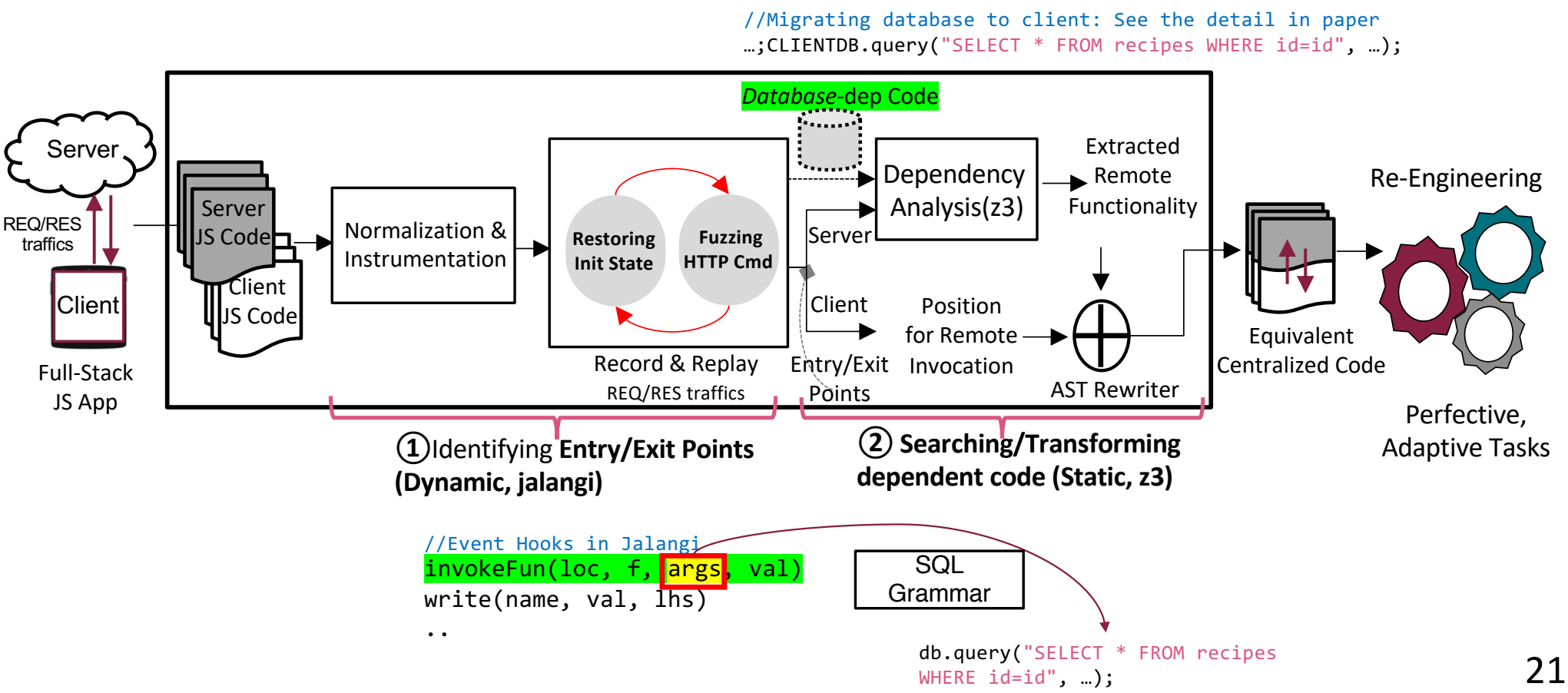
;;Find all Statements that are dependent in Entry and Exit Points

$$\mathbf{ExecutedStmts}(\mathrm{stmt}_n, \mathrm{V}^{\mathrm{u_{id}}}_{\mathrm{unMar}}, \mathrm{V}^{\mathrm{u_{id}}}_{\mathrm{Mar}}) \leftarrow \left(\mathbf{DataDep}(\mathrm{stmt}_n, \mathrm{stmt}_1) \wedge \mathbf{Marshal}(\mathrm{stmt}_1, \mathrm{v}_{\mathrm{Mar}}, \mathrm{V}^{\mathrm{u_{id}}}_{\mathrm{Mar}})\right) \wedge \left(\neg\mathbf{DataDep}(\mathrm{stmt}_n, \mathrm{stmt}_2) \wedge \mathbf{UnMarshal}(\mathrm{stmt}_1, \mathrm{v}_{\mathrm{unMar}}, \mathrm{V}^{\mathrm{u_{id}}}_{\mathrm{unMar}})\right)$$

Client Insourcing Refactoring [WWW 2020]



Overall Process: Client Insourcing Refactoring



Evaluation:
RQ1. **Effort Saved** by Refactoring Operations

Subjects (T1,T2,T3)	$HTTP_{Methods}$	Services	$C\&P/M$ (ULOC)
recipebook (AngularJS, Express,MySQL)	GET	/recipes	22/45
	GET/PUT/DEL	/recipes:id	72/172
	POST	/ingts	25/48
	GET/PUT/DEL	/ingts:id	74/207
	POST	/directions	26/57
	GET/PUT/DEL	/directions:id	60/130
DonutShop (Ajax, Express,knex)	GET/POST	/donuts	22/88
	GET/POST/DEL	/donuts:id	29/155
	GET/POST	/employee	20/71
	GET/POST/DEL	/employee:id	29/138
	GET/POST	/shops	16/83
	GET/DEL	/shops:id	19/128
res-postgresql (axios,restify, Postgres)	GET/POST	/user	22/71
	GET/PUT/DEL	/user	40/120
shopping-cart (Angular2,Express)	GET/POST/DEL	/cart-items	79/130
realty_rest (Angular2,Express)	GET/POST/DEL	/prprts/favs	34/73
	POST	/prprts/likes	291/304
	GET	/prprts	284/297
	GET	/prprts:id	287/300
	GET	/brokers	86/99
	GET	/brokers:id	90/103
med-chem (fetch,koa.js ,knex)	GET	/hbone	9K/9K
	GET	/molecular	9K/9K
BrownNode (AngularJS ,Express)	GET	/u/search	37/65
	GET	/u/search/id	36/64
Bookworm (JQuery, Express)	GET	/ladywithpet	394/409
	GET	/thedea	394/409
	GET	/theredroom	394/409
	GET	/thegift	394/409
	GET	/wallpaper	394/409
	GET	/offshore	394/409
	GET	/bigtripup	394/409
	GET	/amont	394/409
ConfApp (Angular2 ,Express)	GET	/findpeakers	13/66
	GET	/findSpeaker	15/68
	GET	/findSessions	43/117
	GET	/findSession	46/119
Emp_Dir (Angular2 ,Express)	GET	/employees	22/44
	GET	/employees/id	38/60
Total	**61**		**24K/26K**

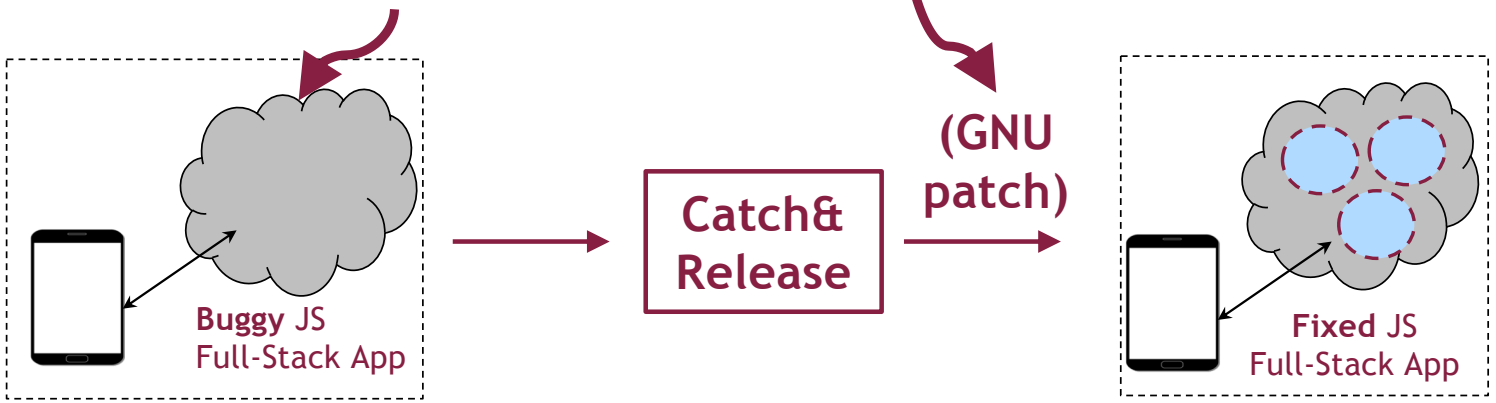
Stateful Subjects

- **61** subjects from **10** full-stack Apps, written with popular JavaScript Libaries
  - 2 tiers or 3 tiers (with database)
  - HTTP protocols with different Commands

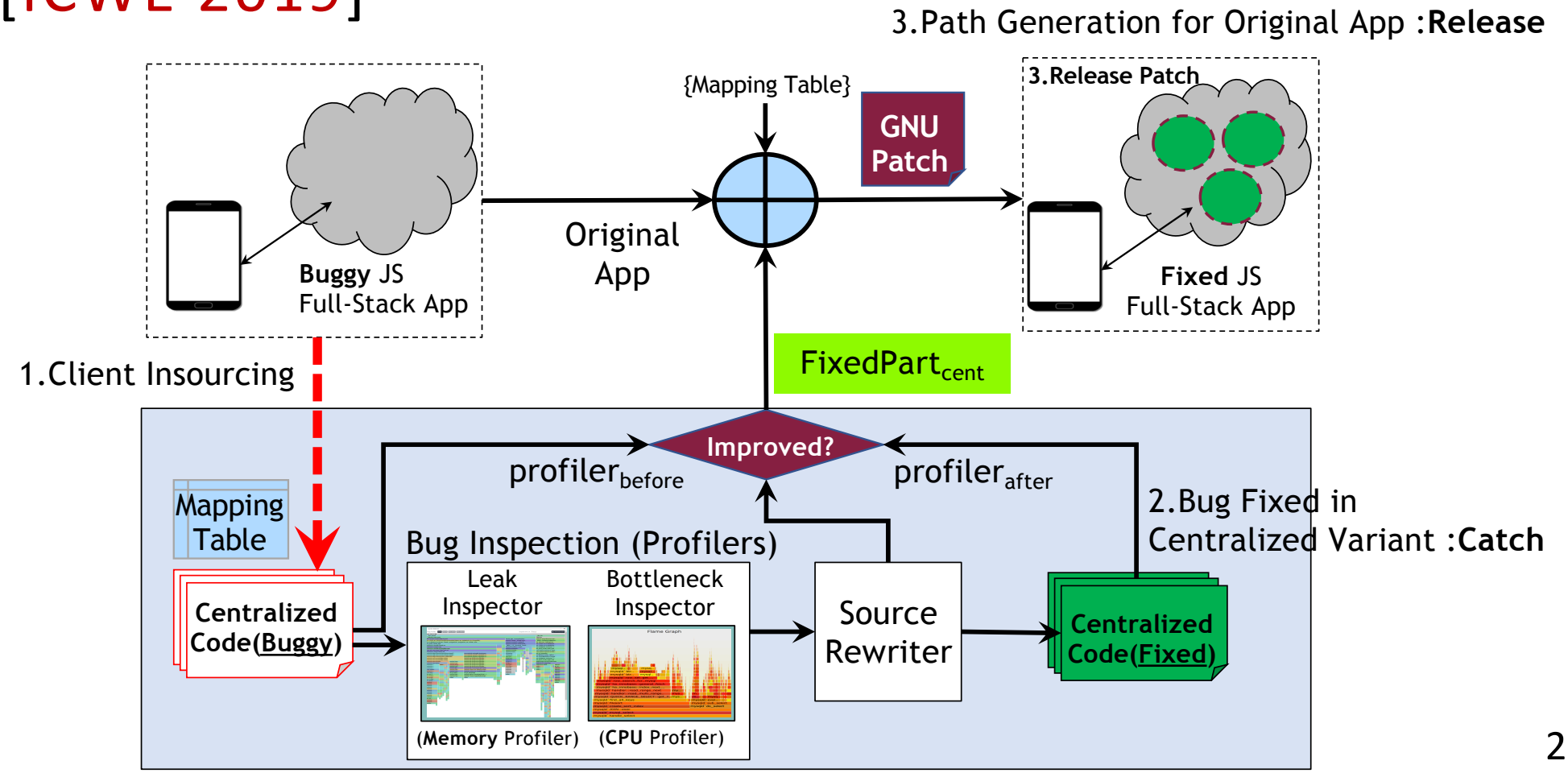
Application #1: *Catch & Release*: Debugging [ICWE 2019]

```
//server/main.js...
function getObjsInArray(obj, array) {
 var foundObjs = [];
 for (var i=0; i<array.length;i++){
  for(var prop in obj) {
   if(obj.hasOwnProperty(prop)) {
    if(obj[prop]===array[i][prop])
    {
      foundObjs.push(array[i]);
      break;
   }}}}
 return foundObjs;
}
```

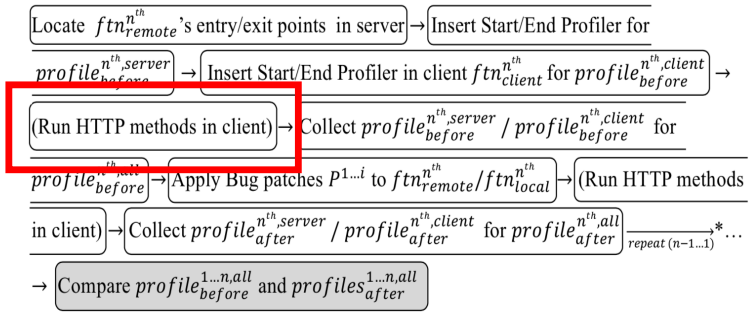
```
5,18c1,16
<(original code for getObjsInArray)
---
> function getObjsInArray(obj, array) {
>  var foundObjs = [];
>  var keys = Object.keys(obj);
>  for (var i=0; i < array.length; i++){
>    for (var j = 0, l = keys.length; j <
  l; j++) {
>    var key = keys[j];
>    if (obj[key] === array[i][key]) {
>      foundObjs.push(array[i]);
>      break;
> }}}
> return foundObjs;
> }
```

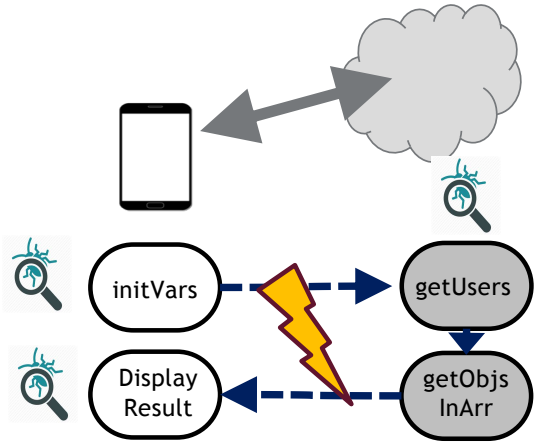


Application #1: *Catch & Release*: Debugging [ICWE 2019]

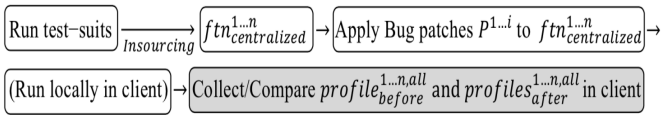


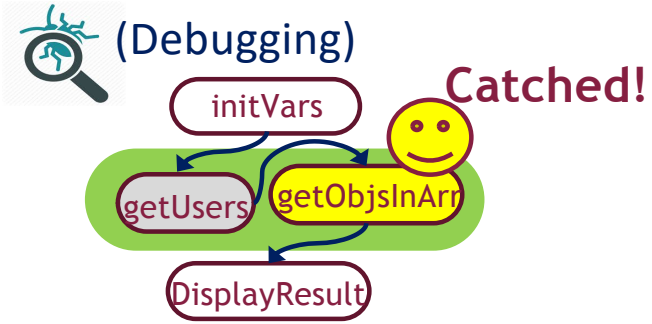
Catch & Release: Decreased Debugging Efforts



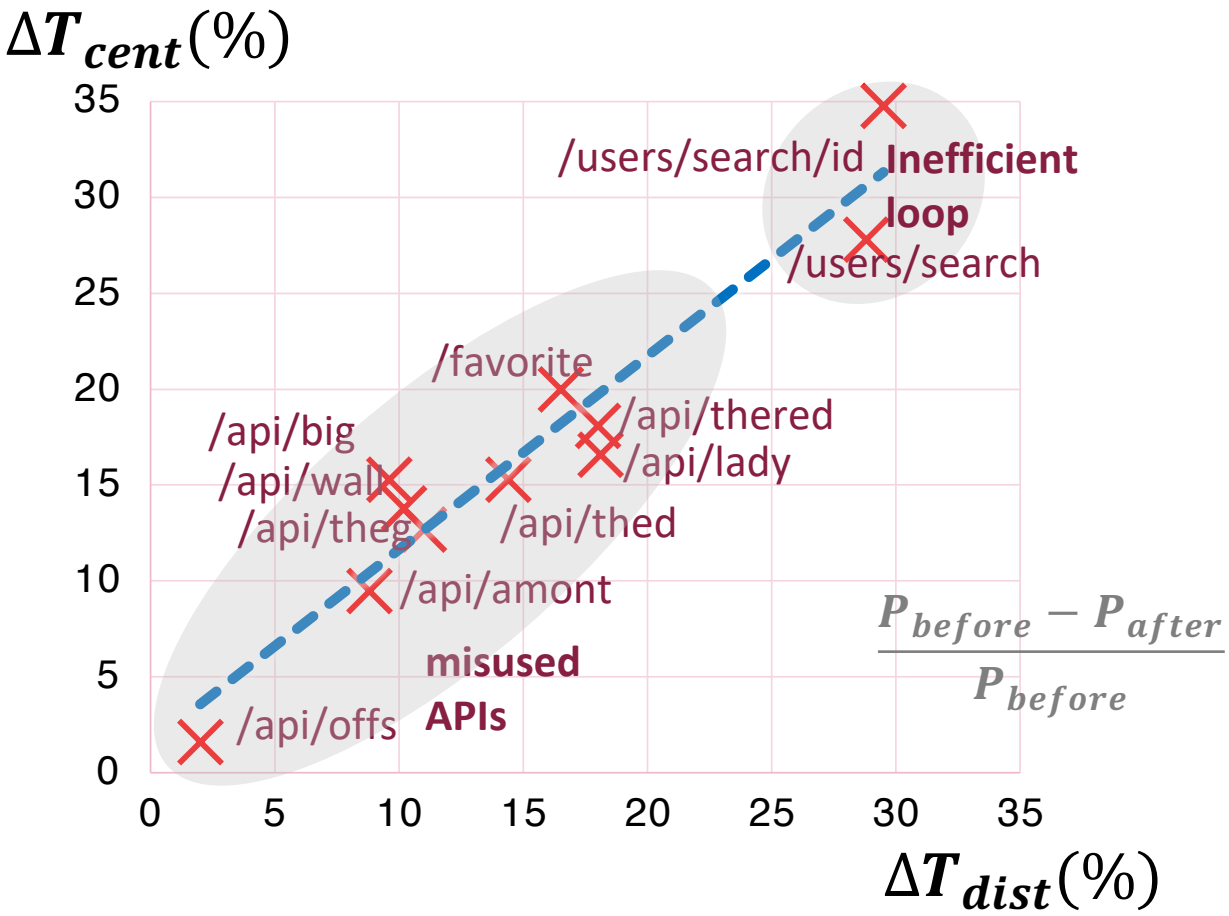


- CanDoR **reduces** the **complexity** of **debugging** distributed apps.
- **90%** Reduced Time to execute Debugging Task in Centralized Variant



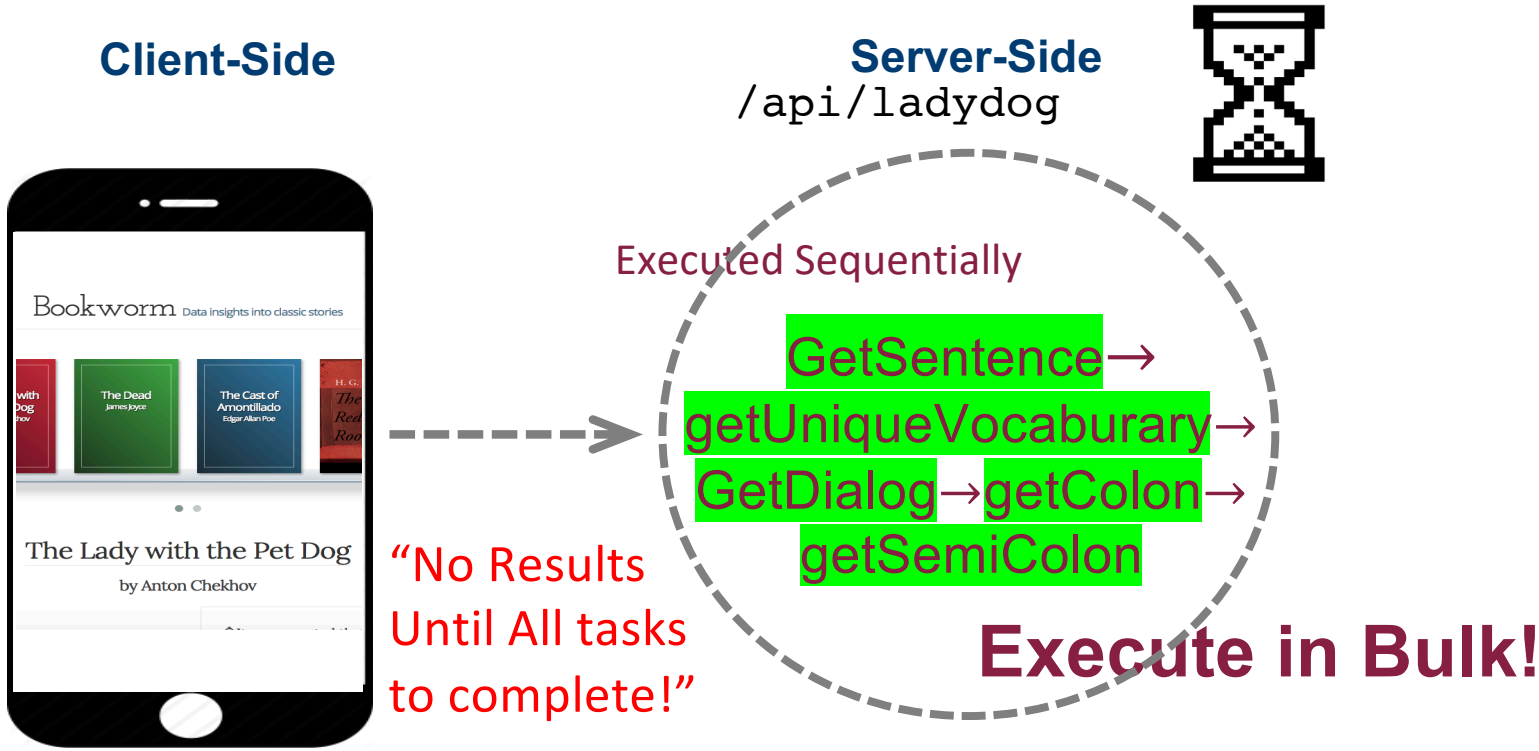


Catch & Release: Regression Analysis[ICWE19, WWW20]



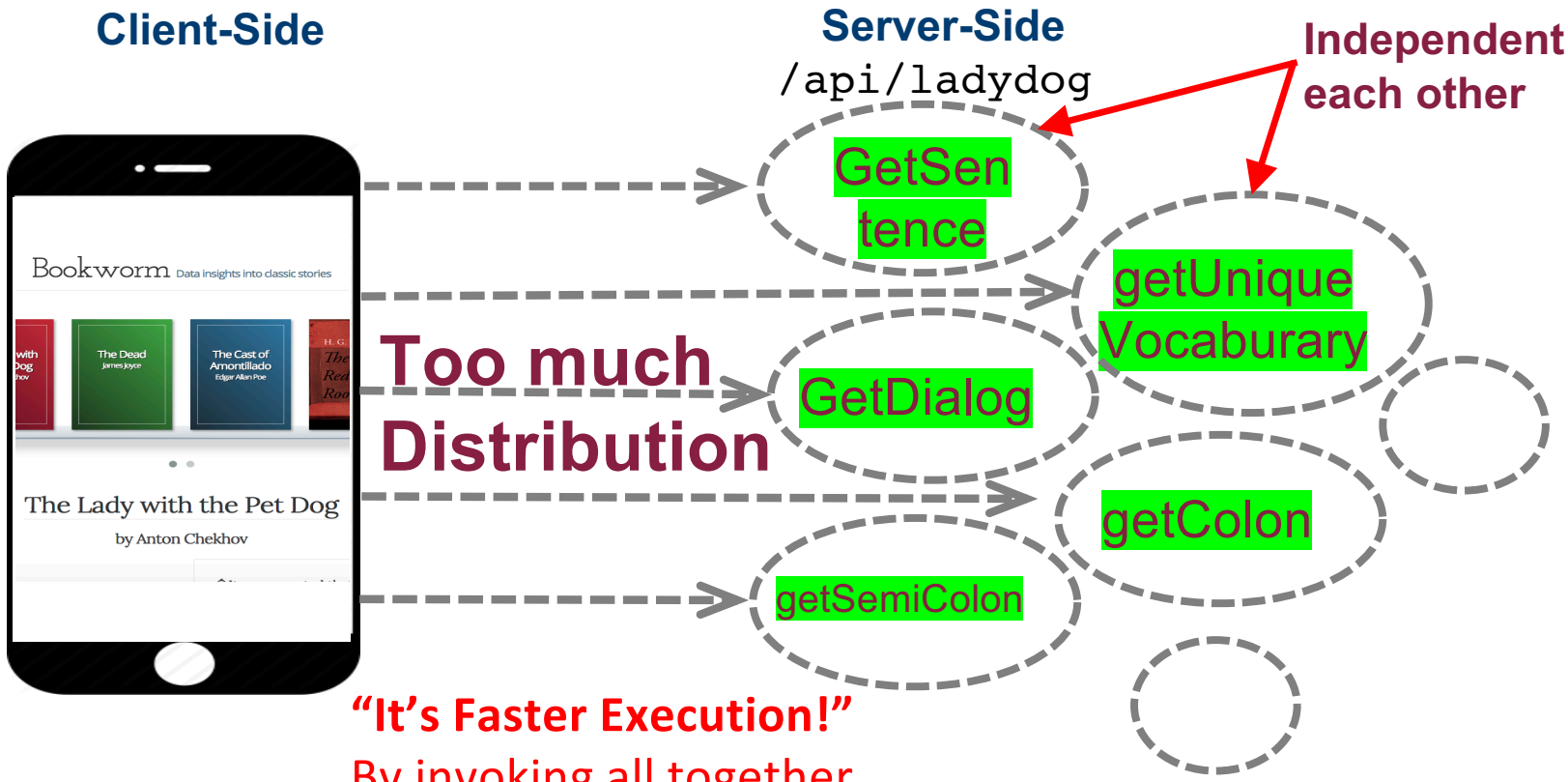
- The result shows that ΔT_{dist} and ΔT_{cent} are almost perfectly **correlated**.
- So **centralized variants** can indeed serve as reliable and convenient **proxies** for **performance debugging**.

Application #2:
D-Goldilocks: Correcting Distribution [SANER 2020]

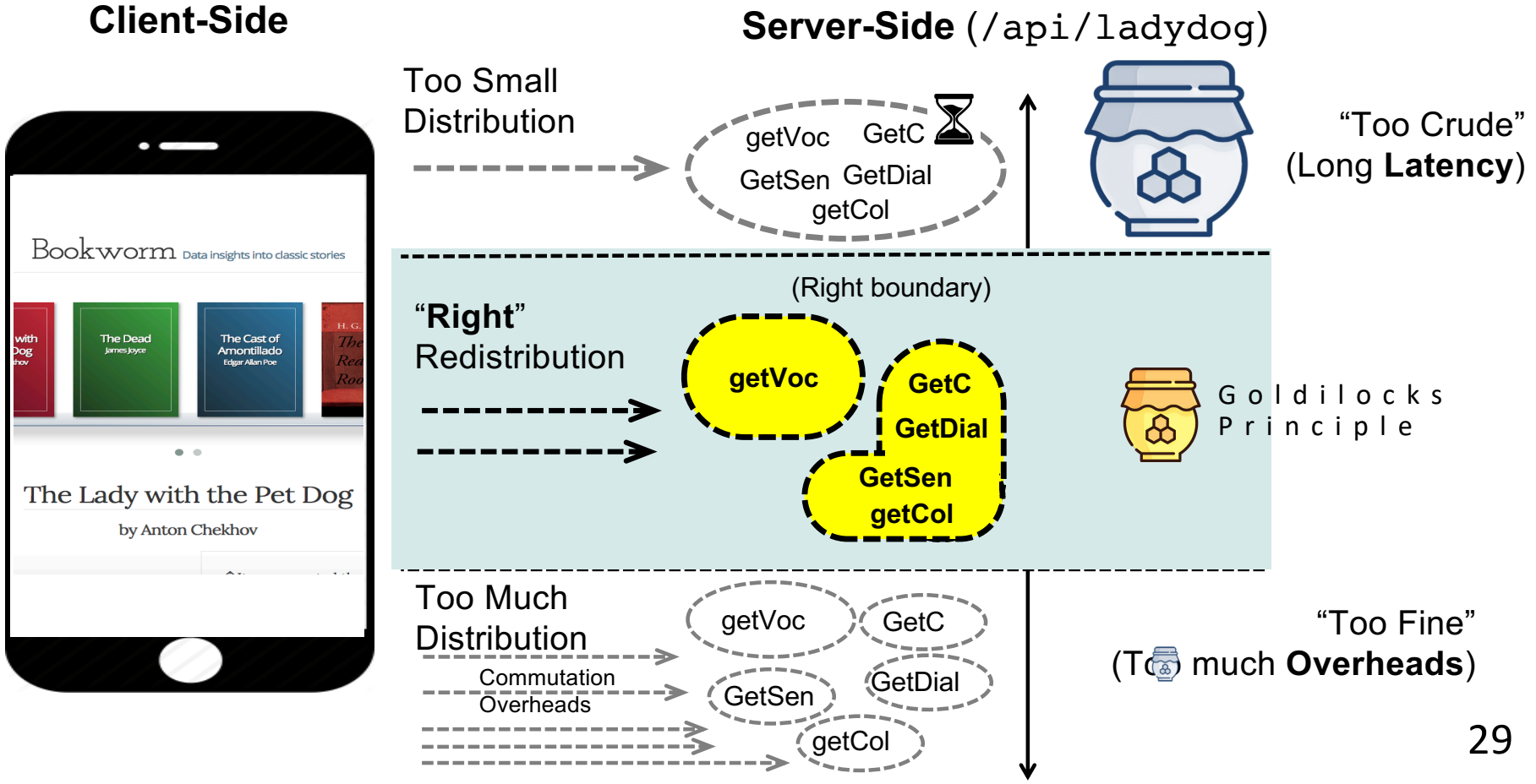


Initial distribution

Application #2:
D-Goldilocks: Correcting Distribution [SANER 2020]



Application #2:
D-Goldilocks: Correcting Distribution [SANER 2020]



D-Goldilocks: Problem Formulation

- Determine which functional distribution from the client's standpoint would minimize the cost of distributed execution

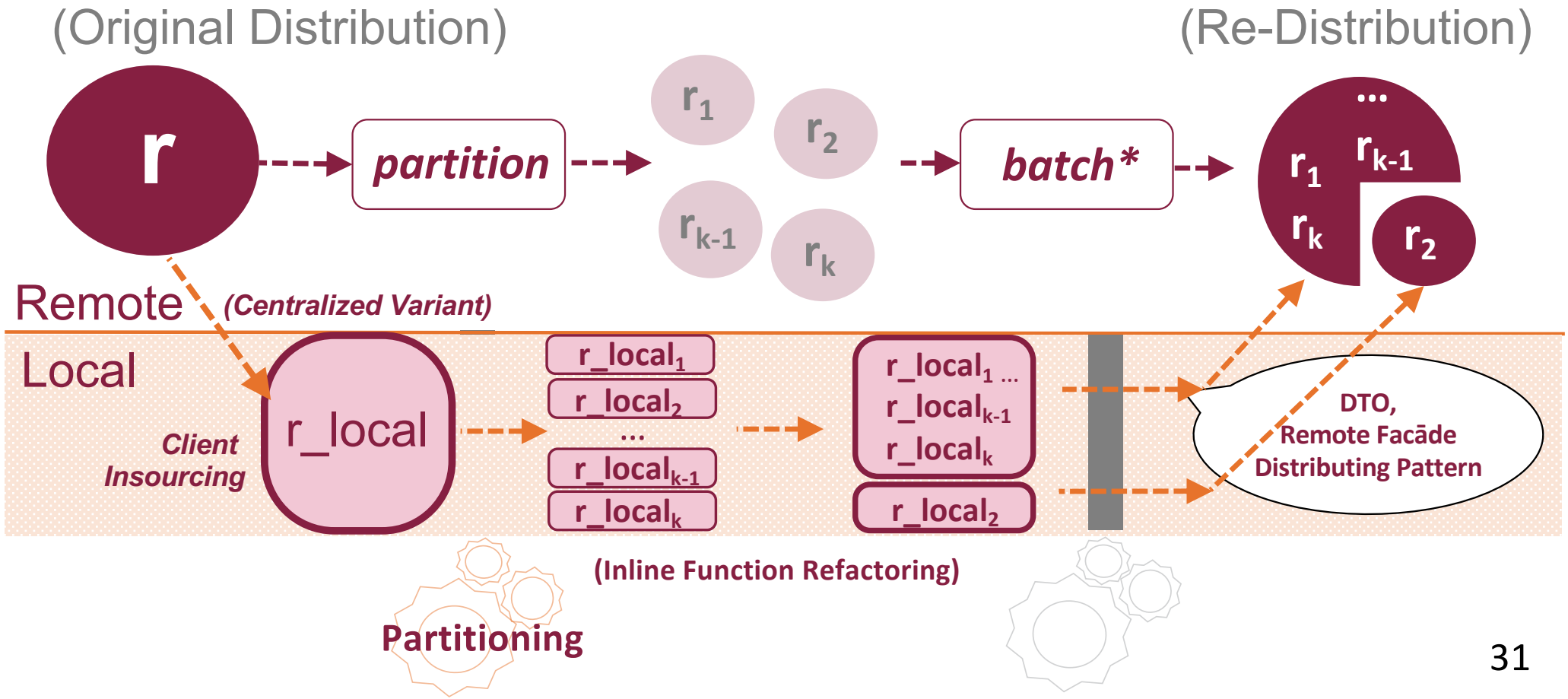
$$C_{Dist\_Exec}(\mathbf{r}) = \alpha \cdot latency(\mathbf{r}) + (1-\alpha) \cdot \Sigma\, resource(\mathbf{r})$$

Execution Time (Performance)

Consumed Resource (Efficiency)

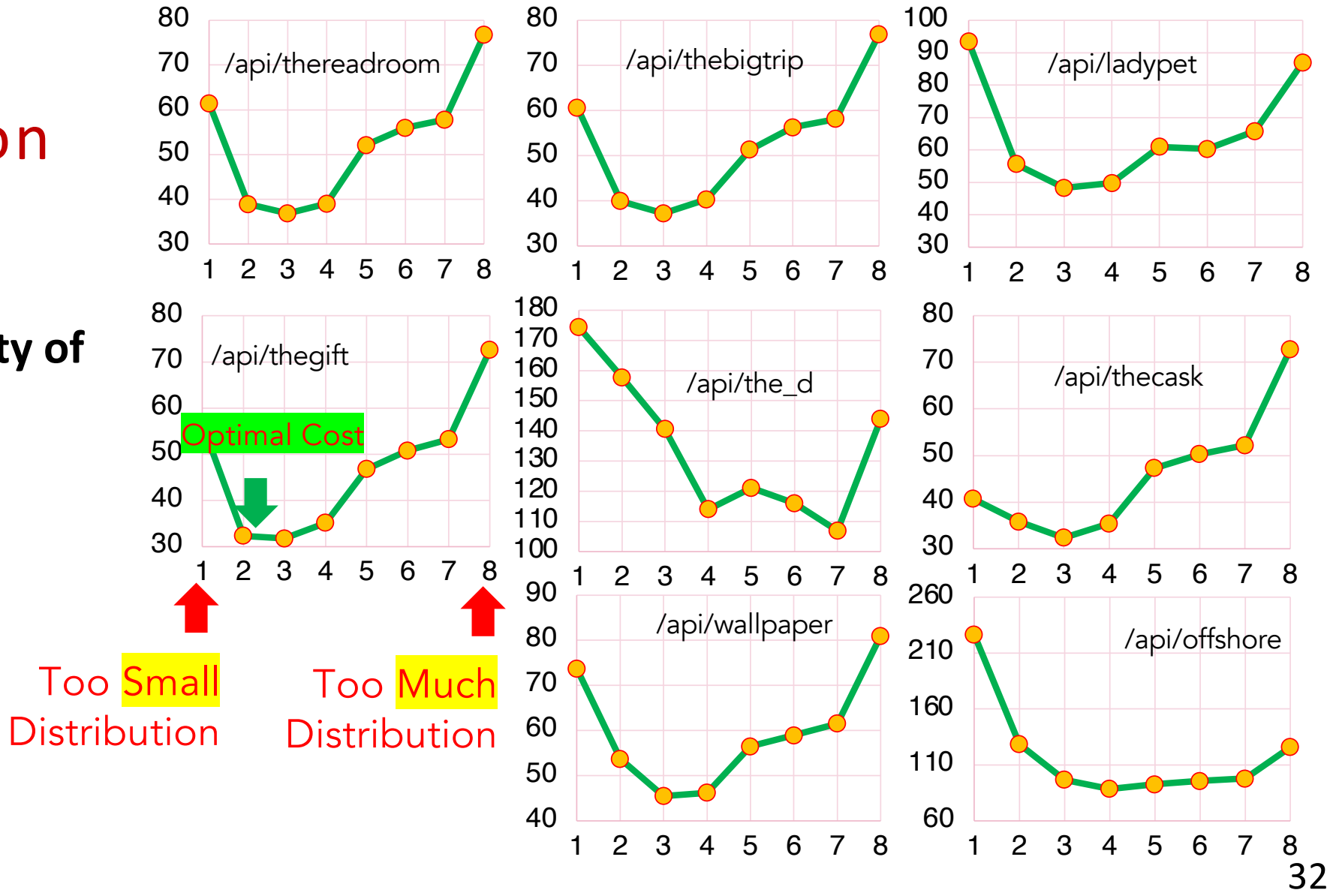
Scale Parameter

D-Goldilocks's Solution: Insourcing->Partition-> Batch Remote Invocation



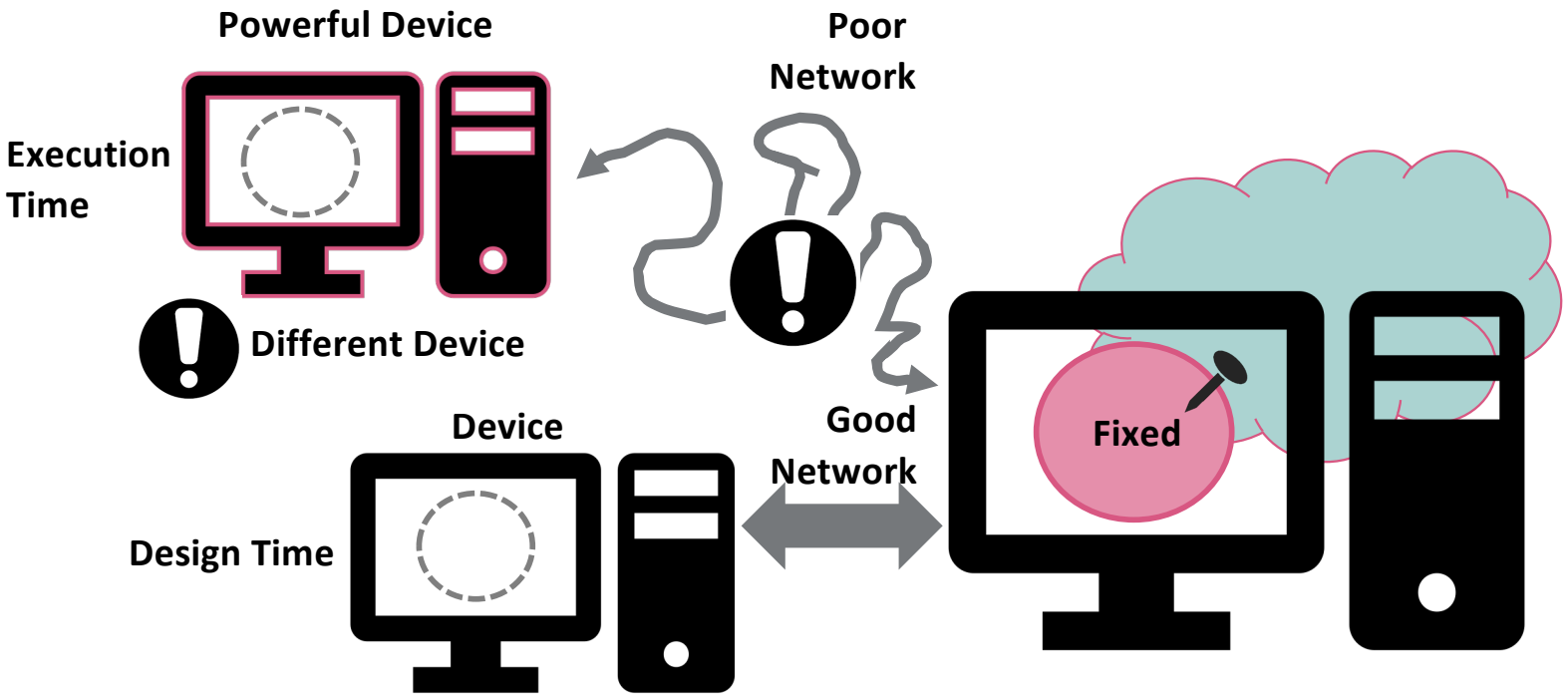
Cost Function

RQ3:—Utility of Cost Model



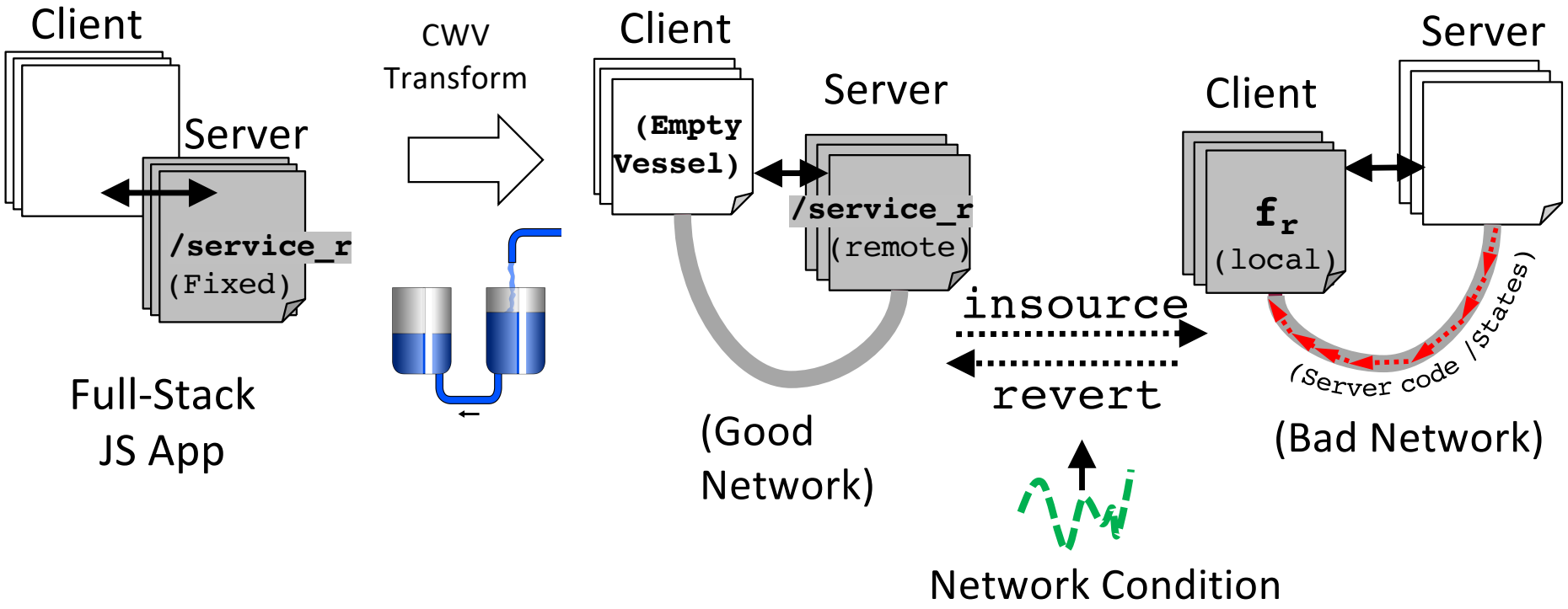
Application #3: *Communicating Web Vessels(CWV)* [ICWE 2021]

- Design and Execution Time Mismatch – Mobile App

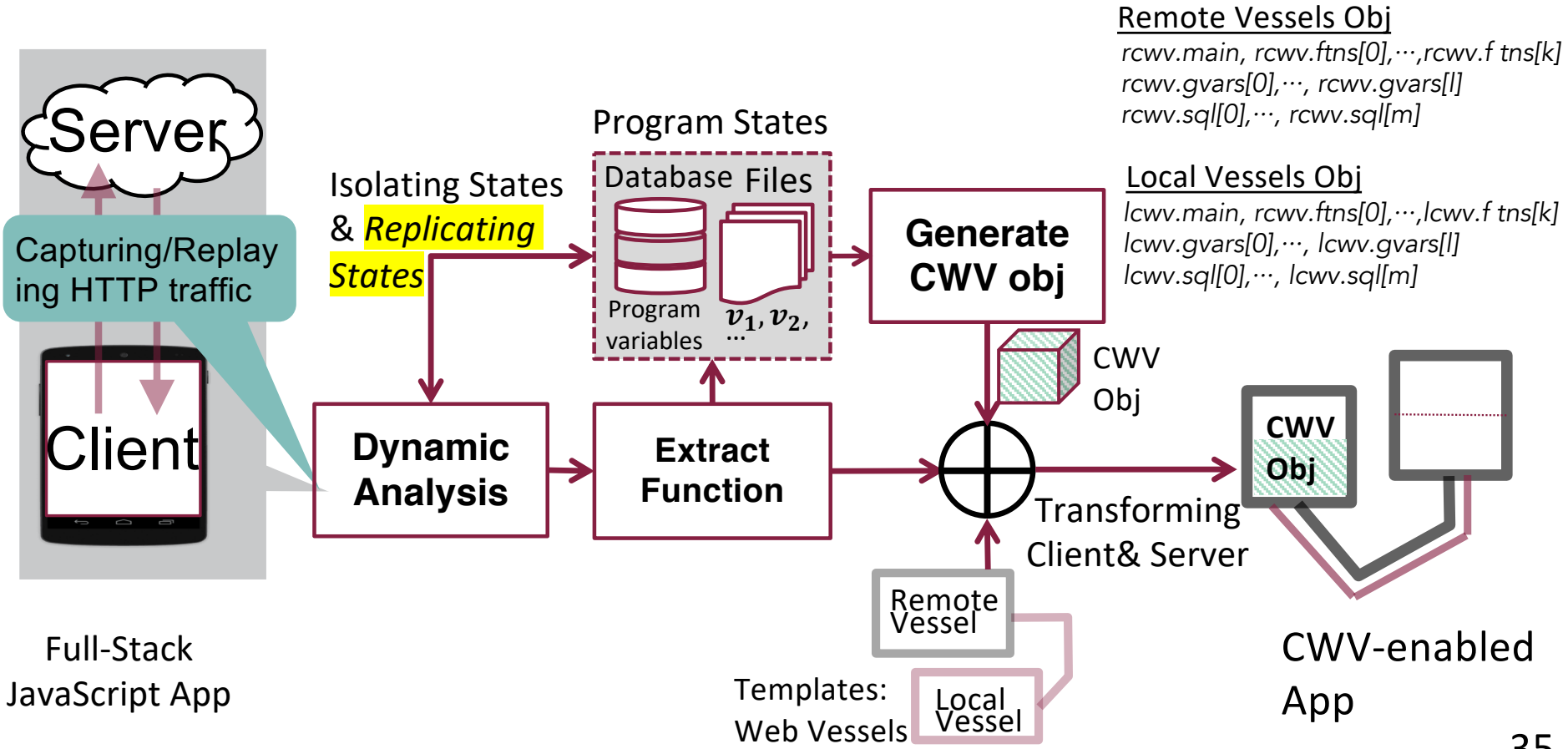


Application #3: *Communicating Web Vessels(CWV)* [ICWE 2021]

- Adaptively *Insource* or *Revert* based on Execution Conditions



Application #3: *Communicating Web Vessels(CWV)*

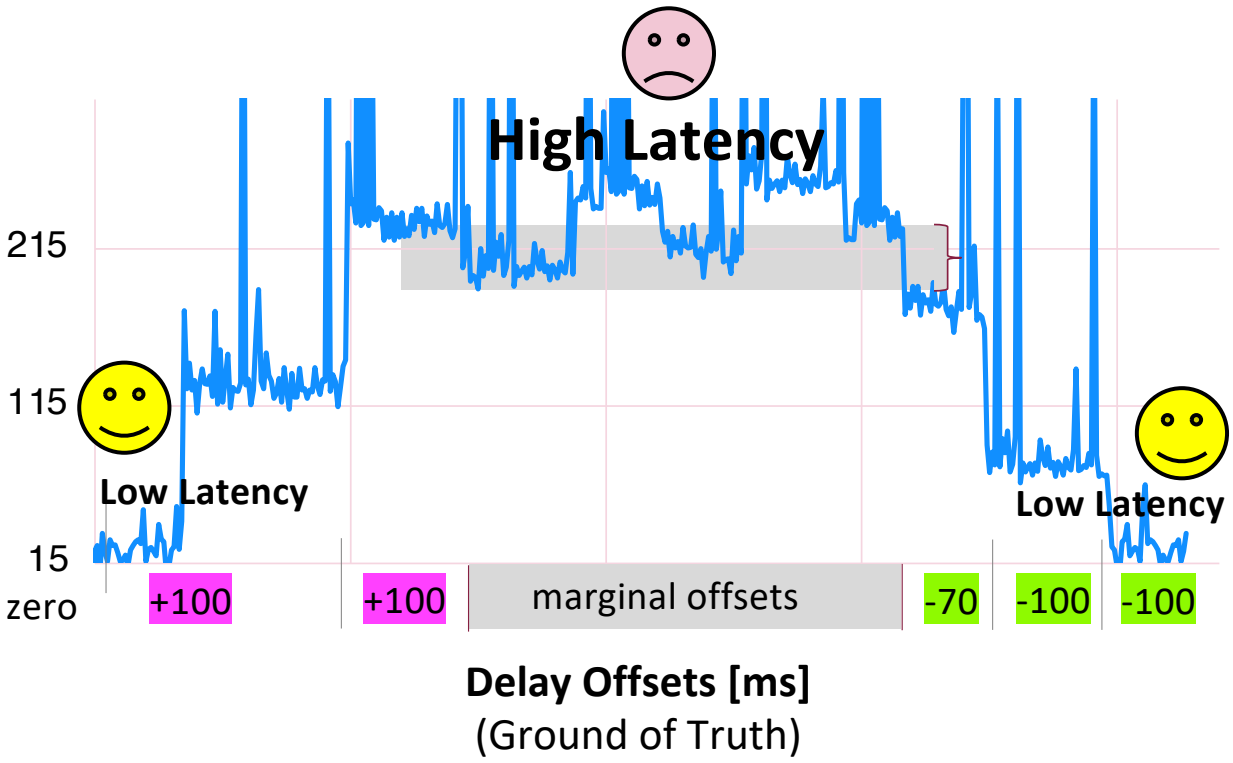


Application #3: *Communicating Web Vessels(CWV)*

CWV-enabled Client

1. **Monitoring Network Condition** by Remote Exec info.
2. Removing Fluctuations with an Adaptive Filter
3. Determining Switching Points by Remote Exec – Local Exec

$$RT(f_r) = \begin{cases} T_{server}(f_r) + RTT^{net} & \text{Remote Exec.} \\ T_{client}(f_r) & \text{Local Exec.} \end{cases}$$

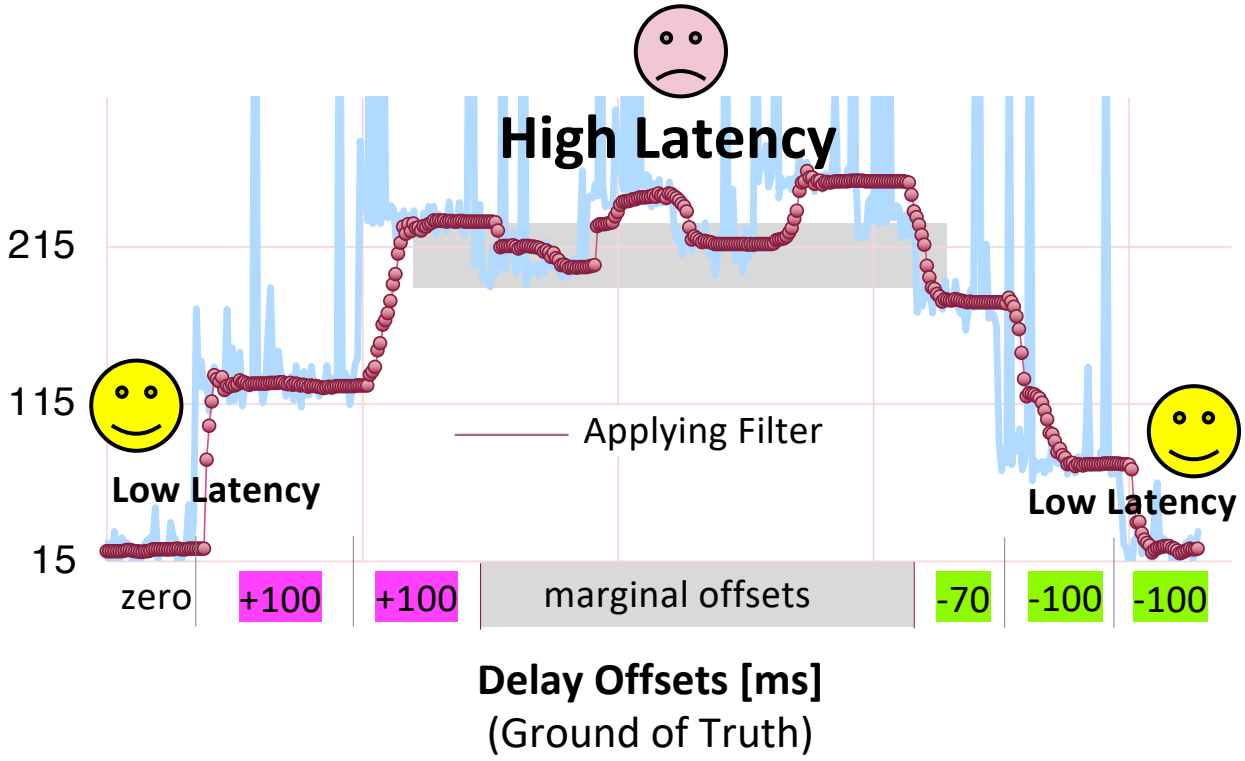


Application #3: *Communicating Web Vessels(CWV)*

CWV-enabled Client

1. **Monitoring Network Condition** by Remote Exec info.
2. **Removing Fluctuations** with an Adaptive Filter
3. Determining Switching Points by Remote Exec – Local Exec

$$RT(f_r) = \begin{cases} T_{server}(f_r) + RTT^{net} & \text{Remote Exec.} \\ T_{client}(f_r) & \text{Local Exec.} \end{cases}$$

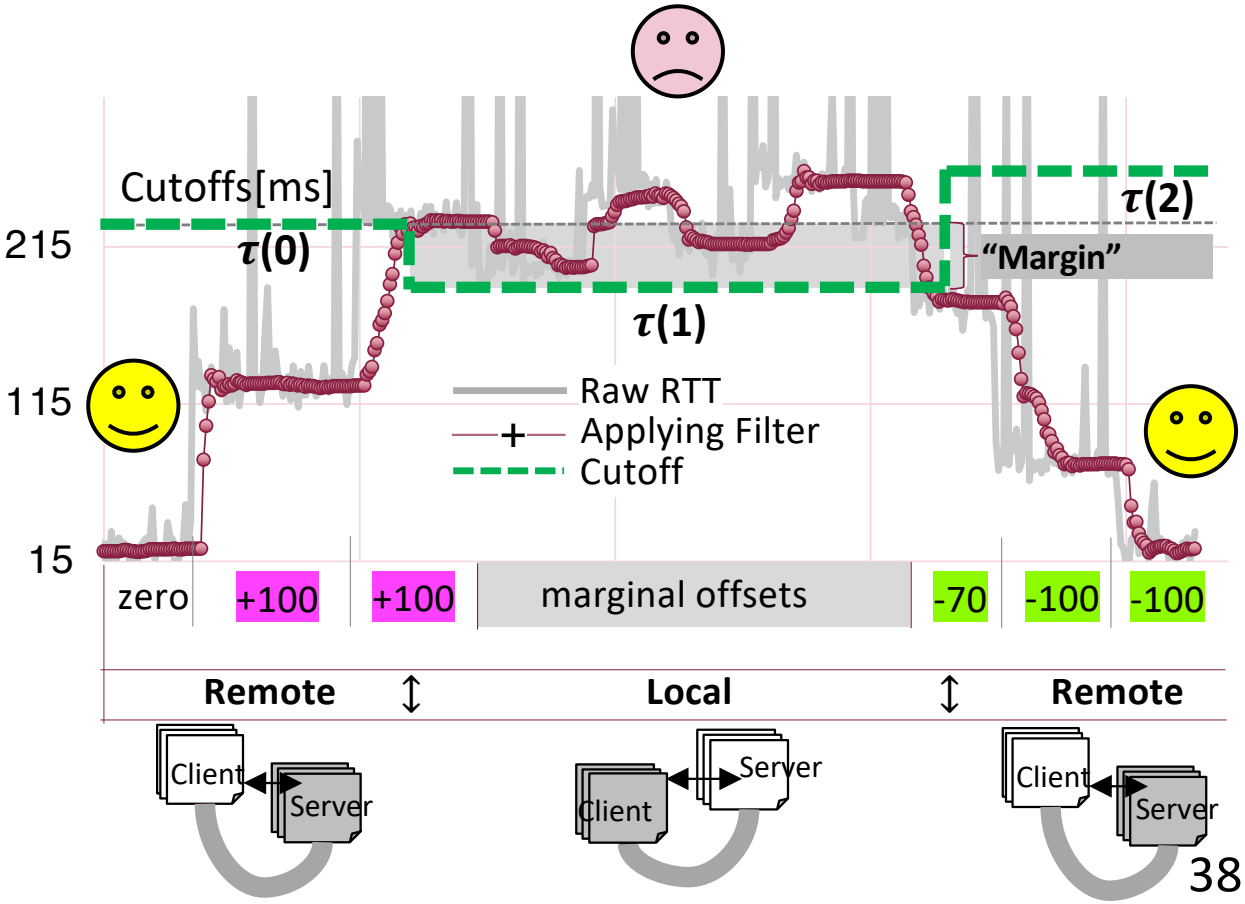


Application #3: *Communicating Web Vessels(CWV)*

CWV-enabled Client

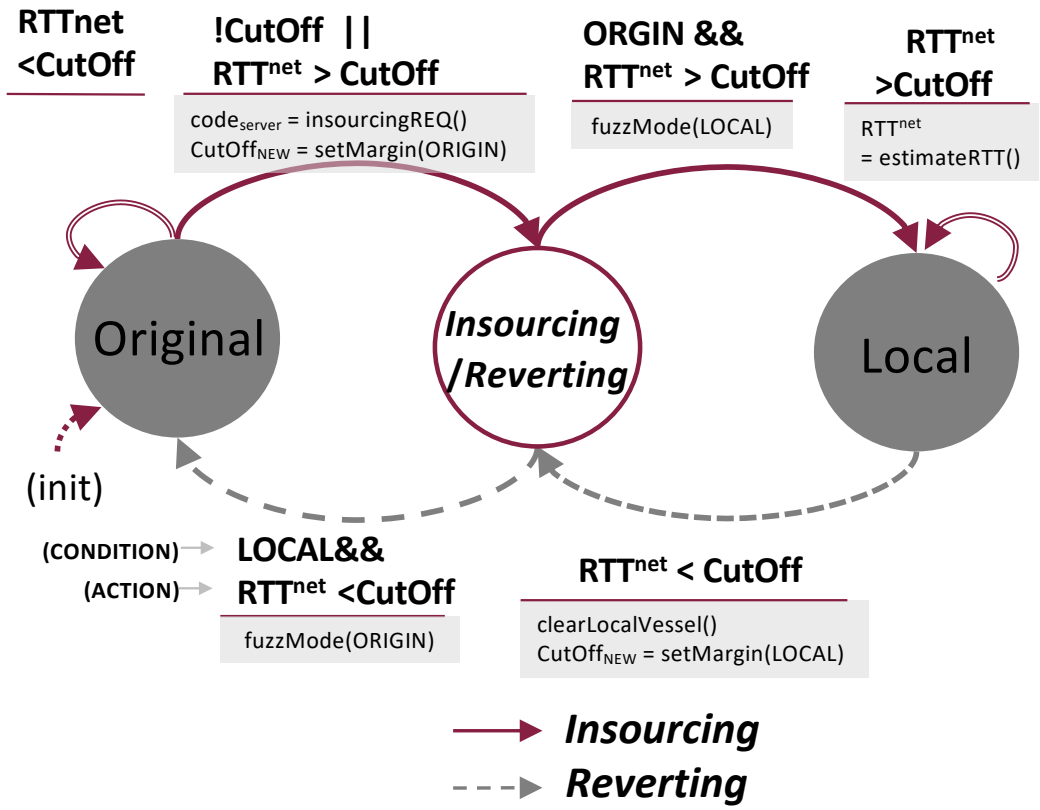
1. **Monitoring Network Condition** by Remote Exec info.
2. **Removing Fluctuations** with an Adaptive Filter
3. **Determining Switching Points** by Remote Exec – Local Exec

$$RT(f_r) = \begin{cases} T_{server}(f_r) + RTT^{net} & \text{Remote Exec.} \\ T_{client}(f_r) & \text{Local Exec.} \end{cases}$$



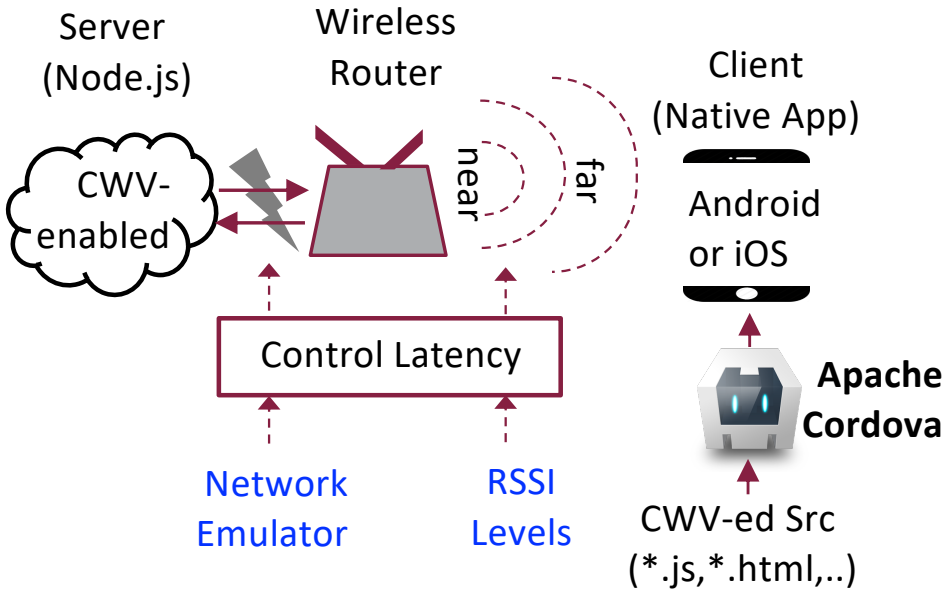
Application #3: *Communicating Web Vessels(CWV)*

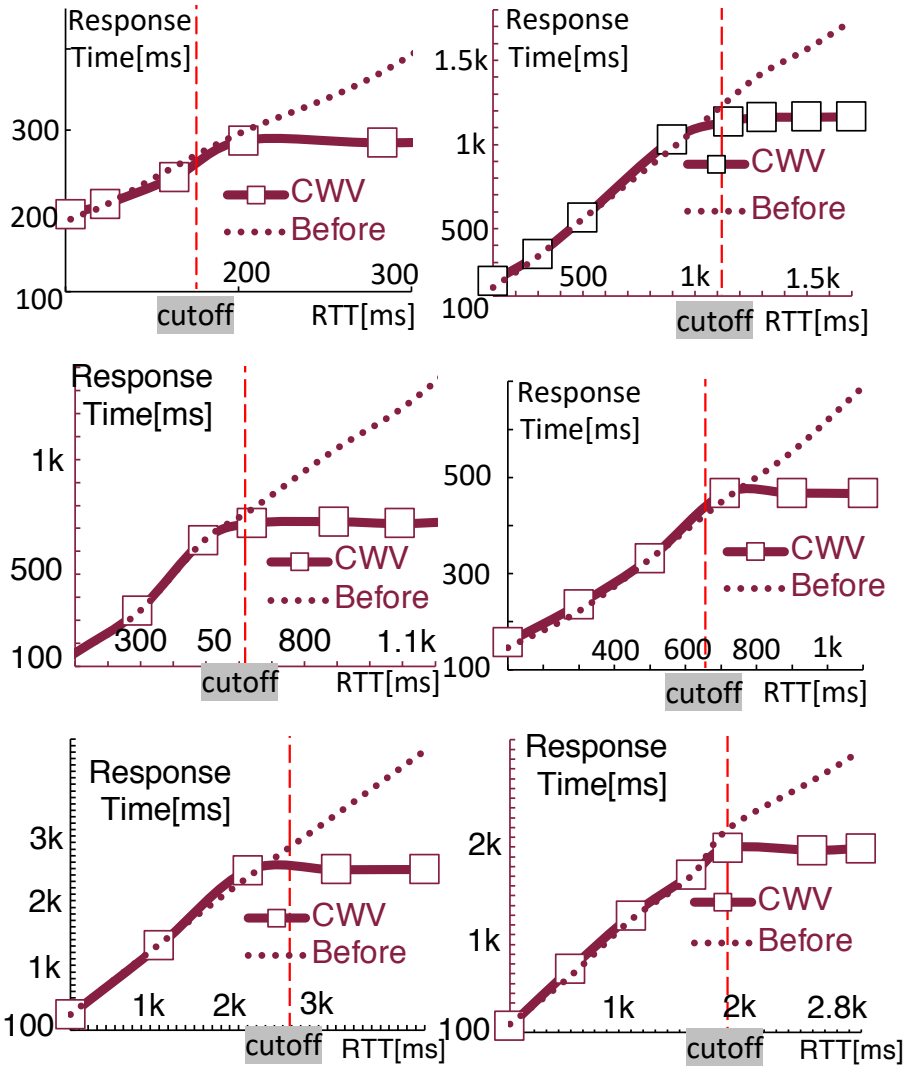
- Implemetations Web Vessels
  - Reducing transmitting Overhead
    - Initially Attempting Insourcing remote vessel (rcwv). After that, only changed part is transmitted
    - Ignoring Temporal or Marginal changes via Adaptive Filter and Margin Param
  - Sandboxing the Local Vessels
    - iframe sandbox
  - Lightweight DB for client browser



Application #3: *CWV*

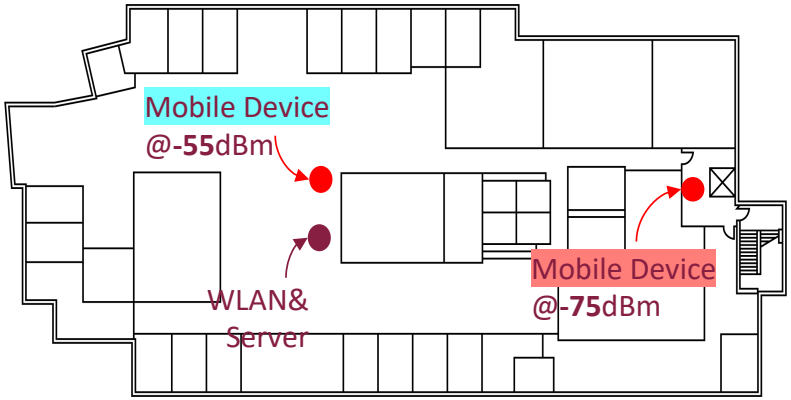
- Response time on different Nets
  - Original versus CWV-enabled apps



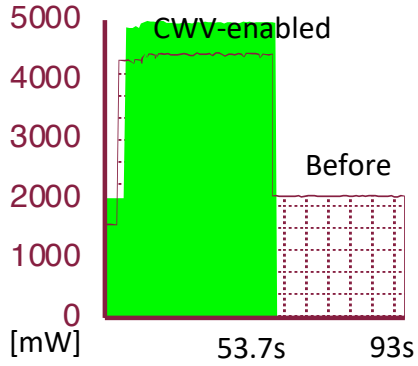


Application #3: *CWV*

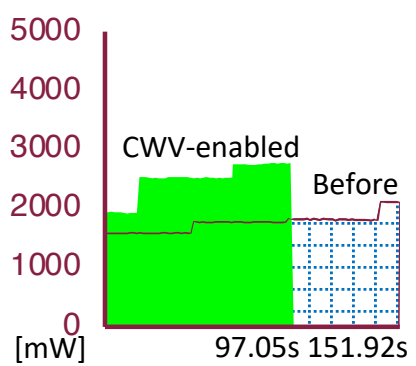
- Energy Consumed under different network conditions
  - Original versus CWV-enabled apps



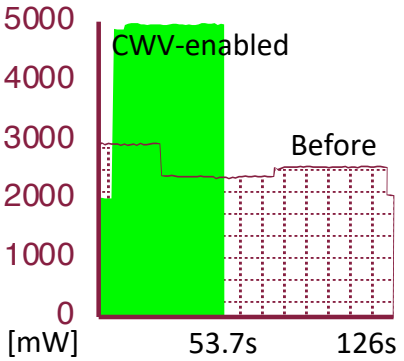
Environment: 28x14m^2



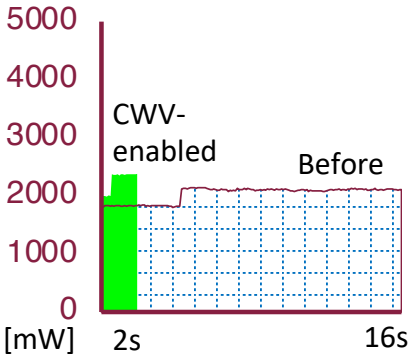
(a) /string-fasta



(b) /api/thedea



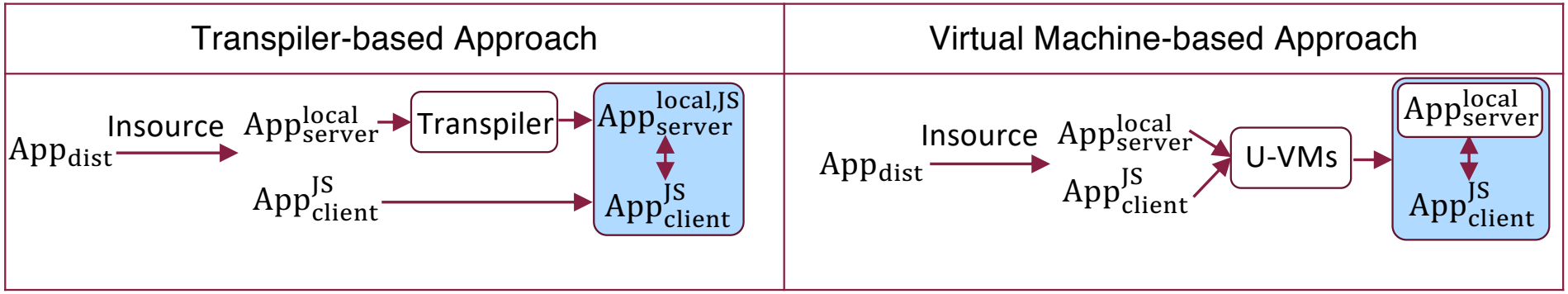
(c) /api/thelady



(d) /hbone

Future Work Direction

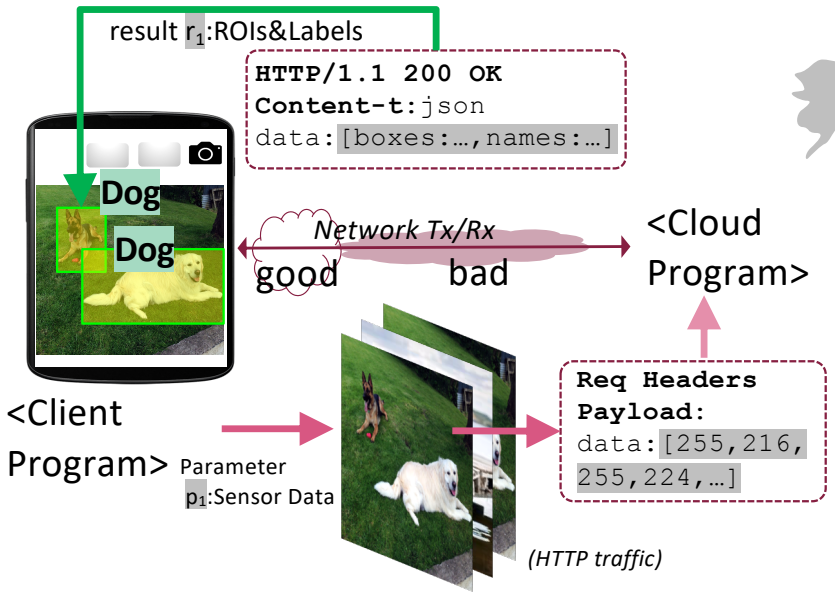
- 1. Transitioning Partial Stack to Full Stack
  - Increasing Applicability of Client Insourcing

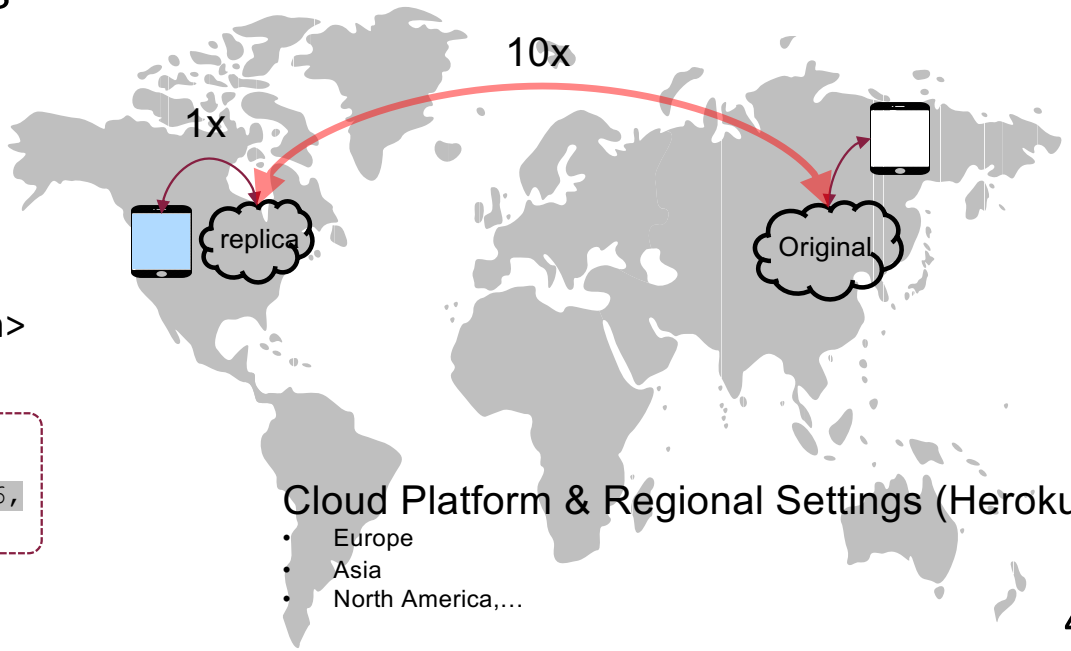


- 2. Edge Refactoring
  - Replicating Cloud Apps for Edge-based Services
    - From Client<->Cloud To Client<->**Edge**<->Cloud architecture
  - Rejected from ICDCS 2021 with 2 Accepts

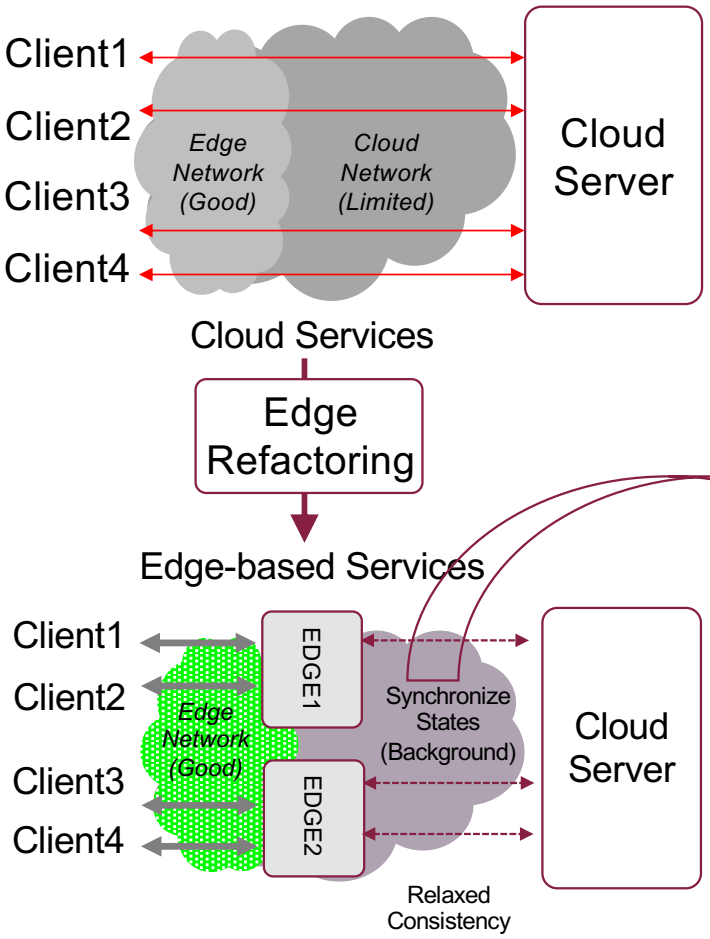
Future work --Edge Refactoring

- Locality of Cloud-based services
  - RTT across different continents: one order of magnitude
- The network transmission bottlenecks (Sensor Data Deluge)
  - Negate all performance advantages

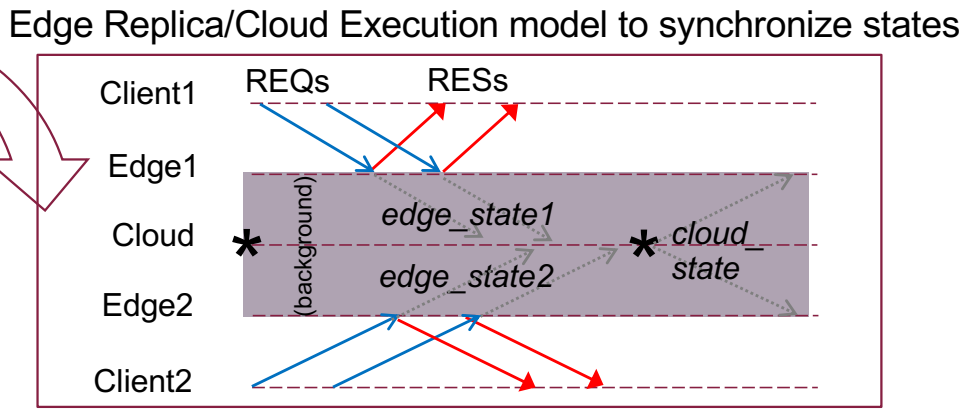




Future work ---Edge Refactoring



- 1. Replicating {$state_{init}$, ftn_{init}} of Cloud Service
- 2. Synchronizing States between Cloud and Edge Replicas
- **RQ1**. Correctness of our Execution Model: proving it with Isabelle HOL framework
- **RQ2**. Performance: Throughput, Latency



Conclusion

- Introduce a new Refactoring: ***"Client Insourcing"***
- Demonstrate the value and utility of ***"Client Insourcing"***
  - **Saving Effort** for Programmer to Analyze and Modify Distributed Apps
  - **Optimizing** Distributed Apps
    - Granularity, Performance, Efficiency, Response Time
  - **Evolving/Maintaining** Distributed Apps
    - *Adaptive*: CW-Vessels, D-Goldilocks
    - *Perfective*: Automated Sandboxing, D-Goldilocks
    - *Corrective:* Catch & Release

Q&A and Thanks

[Dissertation]

The Client Insourcing Refactoring to Facilitate the Re-engineering of Web-Based Applications

- Introduction
- Background
- *Chapter 1. Client Insourcing*
- *Chapter 2. D-Goldilocks*
- *Chapter 3. Catch & Release*
- *Chapter 4. Comm. Web Vessels*
- Related Work
- Future Work
- Conclusion

No.	Paper		Conference	Area	
1.	Client Insourcing		**Web Conference 2020** (19%, 217/1129)	Web Engineering	1st Author/2
2.	D-Goldilocks		**SANER 2020** (21%, 42/199)	Software Engineering	1st Author/2
3.	Catch & Release (CanDoR)		**ICWE 2019** (25%, 26/106)	Web Engineering	1st Author/2
4.	Comm. Web Vessels		**ICWE 2021** (22%, 25/118)	Web Engineering	1st Author/2
5.	Doctoral Symposium	Paper1	**Web Conference 2020**	Web Engineering	1st Author/1
6.		Paper2	**ICWE 2019**	Web Engineering	1st Author/1
7.	Project1	Paper1	**MobileSoft 2018** (Nominated for the Best Paper)	Software Engineering	1st Author/3
8.	Project2	Paper1	**GPCE 2018**	Software Engineering	2nd Author/3
9.		Paper2	**Journal of Computer Lang** (Nominated for the Best Paper)	Software Engineering	2nd Author/3

Appendix