VIRGINIA TECH™

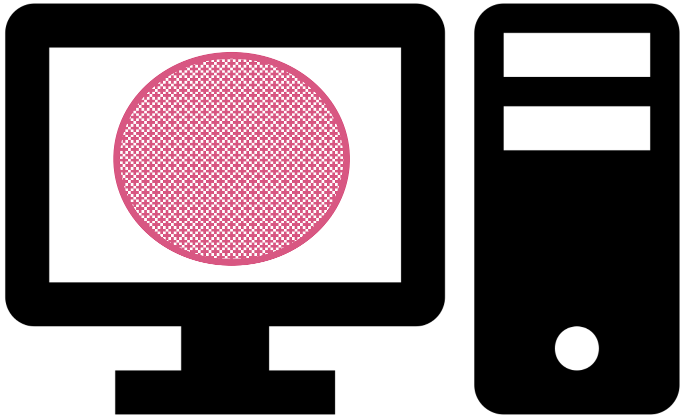
D-Goldilocks: Automatic Redistribution of Remote Functionalities for Performance and Efficiency

KIJIN AN AND ELI TILEVICH
SOFTWARE INNOVATIONS LAB

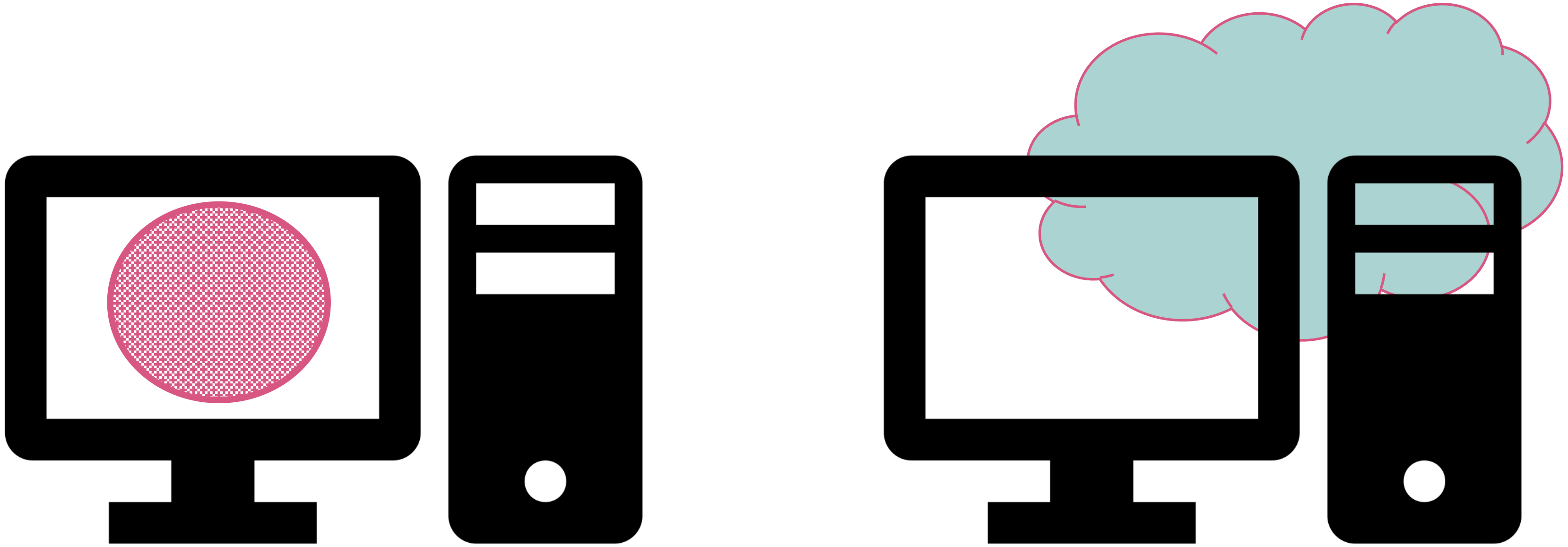
Distribution

- **Why Distributed Computing?**
  - Take advantage of remote **computing resources**
  - Improve **performance** and/or **efficiency**
- **Distribution Benefits**
  - Access superior remote resources
  - Share the **computational load**
- **Distribution Costs**
  - Communication Overhead
  - Partial failure
  - Security

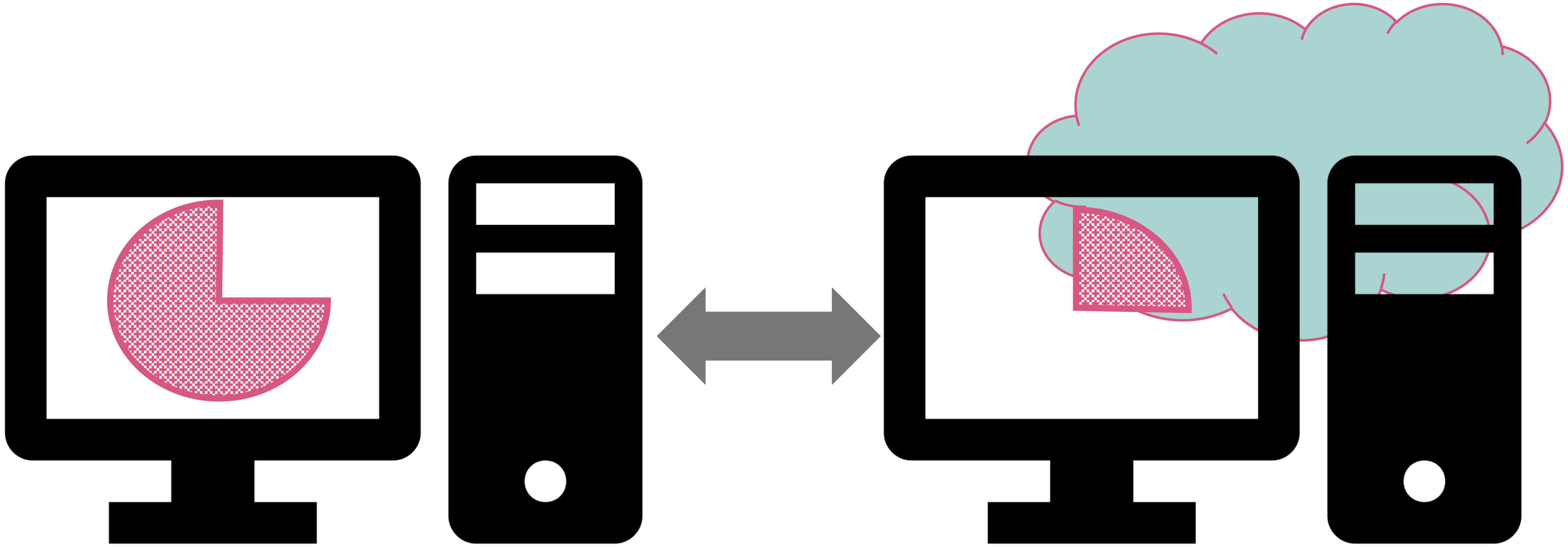
Distribution



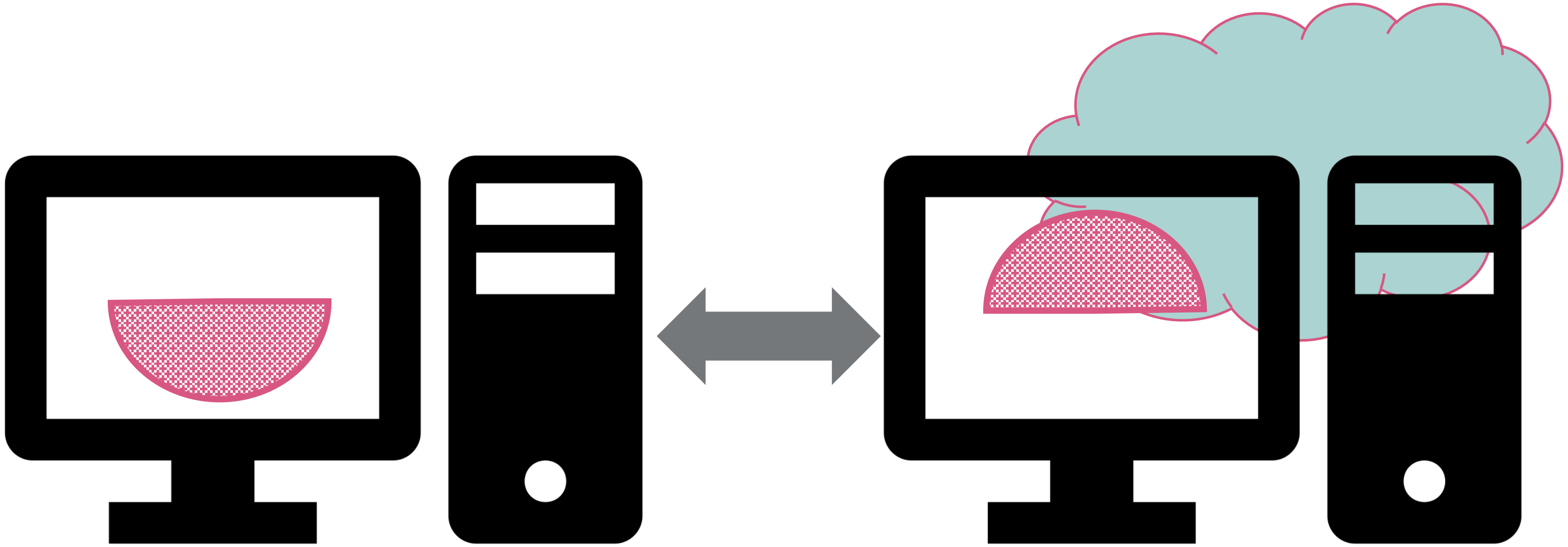
Distribution



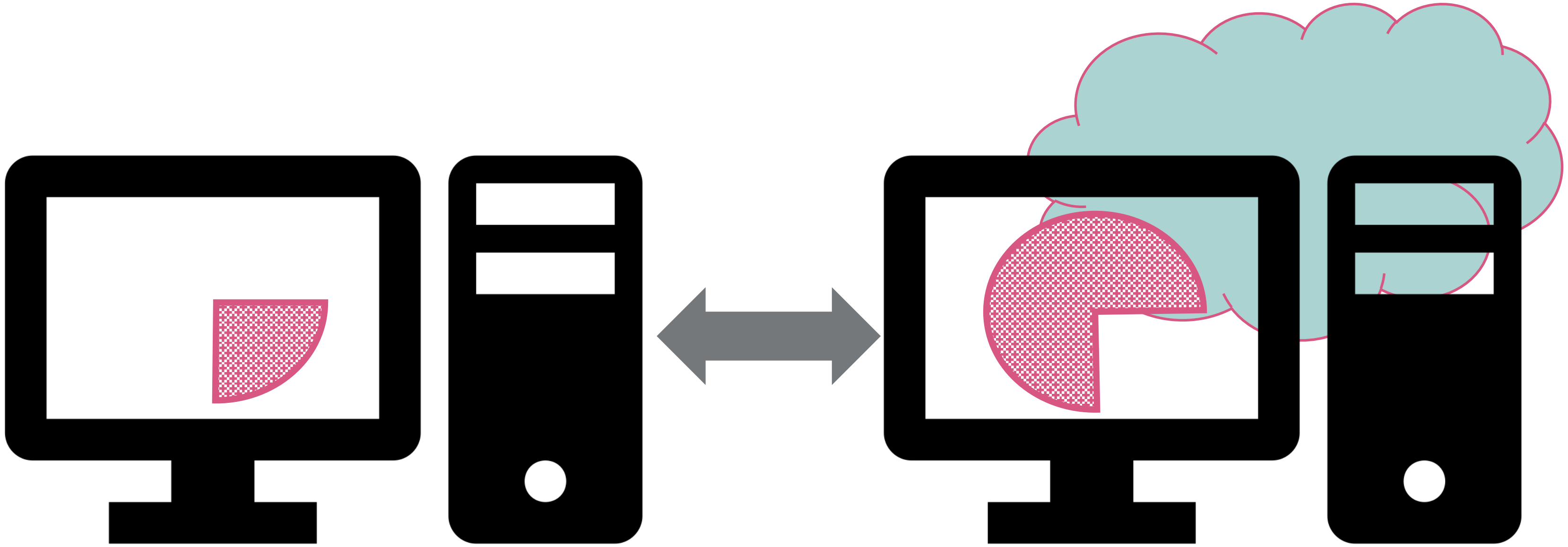
Distribution



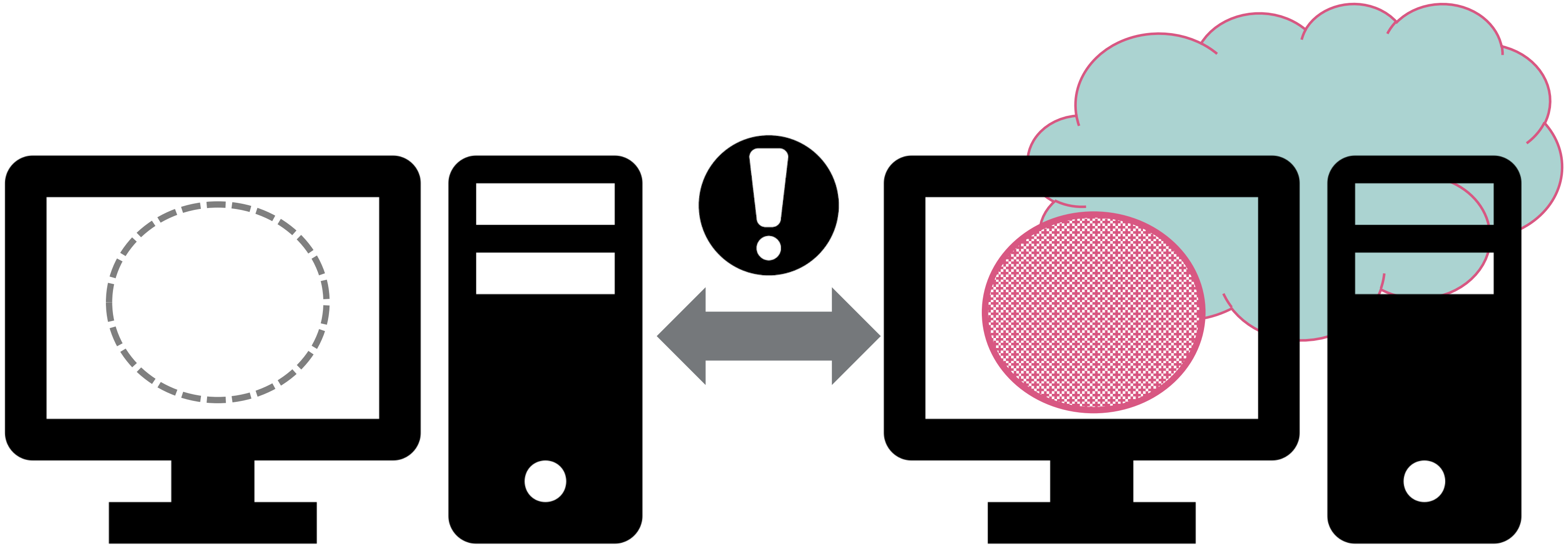
Distribution



Distribution

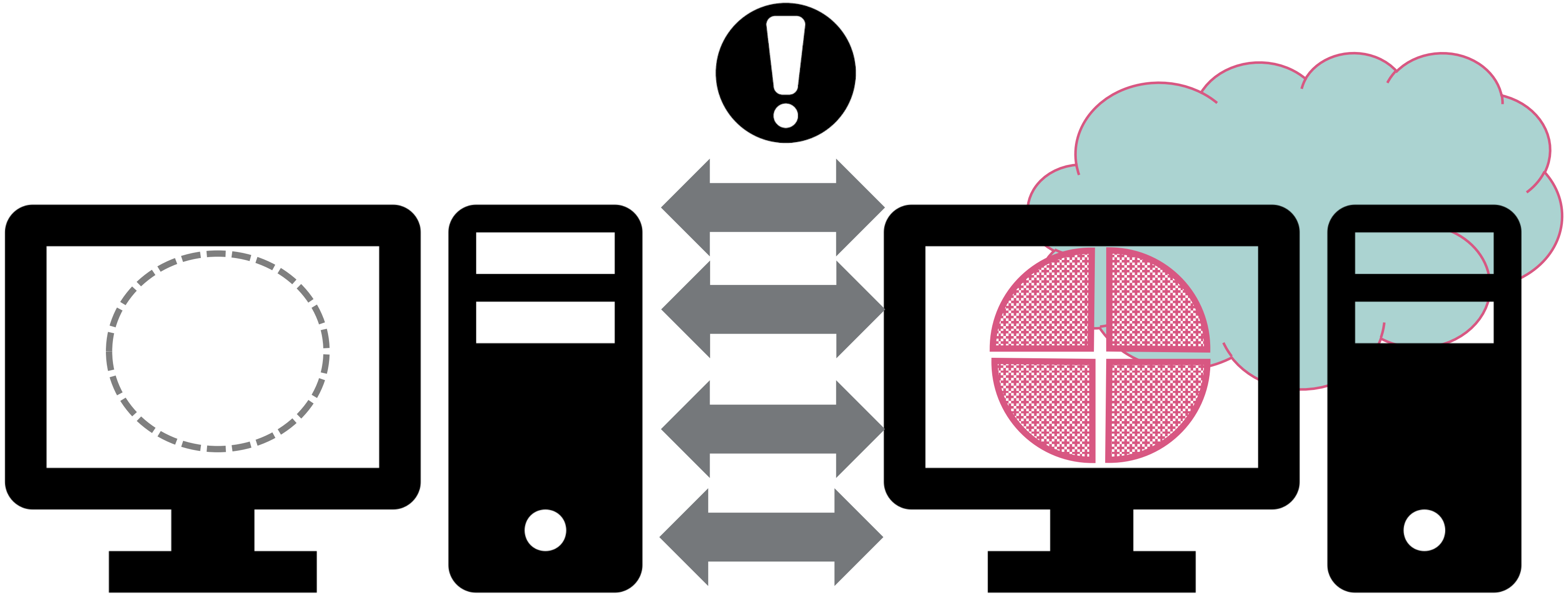


Distribution (Granularity of Remote Service)



Too Crude!
(Granularity of
Remote Service)

Distribution (Granularity of Remote Service)



(Asynchronous Executions)

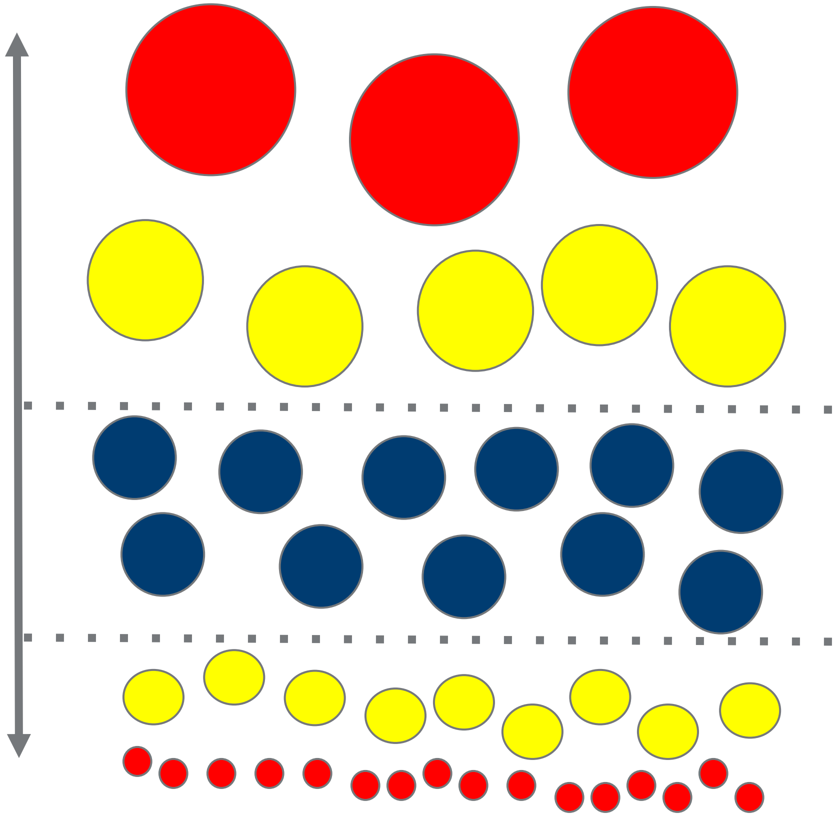
Too Fine!
(Granularity of Remote Service)

Too Much Remote Execution not always beneficial

:Nano Service Anti-pattern[Moha 2012 et. al]

Slower Speed

Low Overhead



Bulk

(Granularity of Remote Service)

Faster Speed

High Overhead

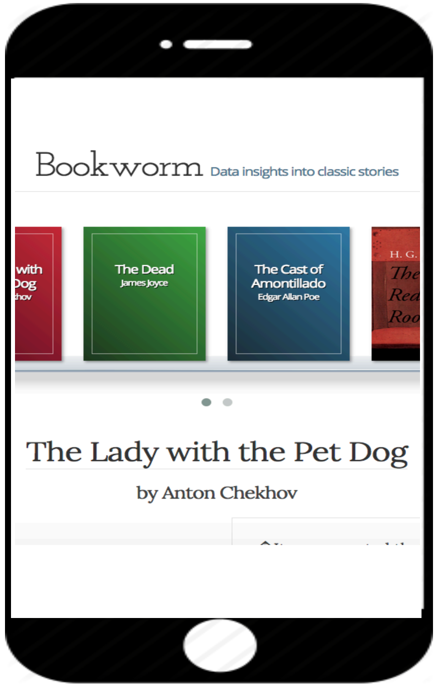
Nano

“Performance”

“Efficiency”

Motivating Real-world’s Example: Bookworm

Client-Side



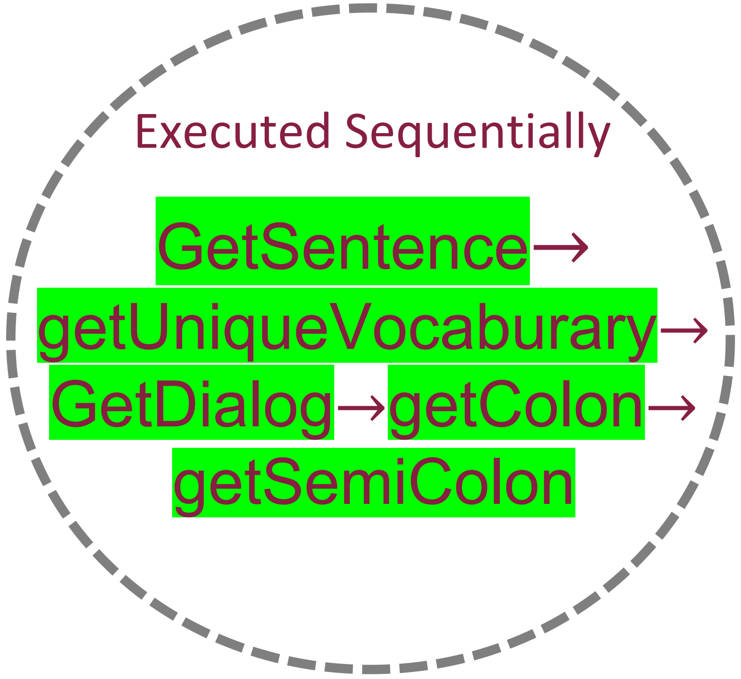
Bookworm Data insights into classic stories

The Lady with the Pet Dog

by Anton Chekhov

Server-Side

`/api/ladydog`



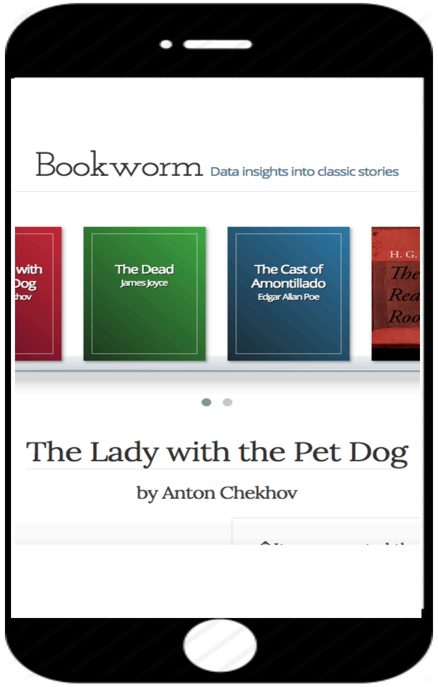
Executed Sequentially

GetSentence→
getUniqueVocaburary→
GetDialog→getColon→
getSemiColon

Initial distribution

Motivating Real-world's Example: Bookworm

Client-Side



Bookworm Data insights into classic stories

The Lady with the Pet Dog

by Anton Chekhov

"No Results Until All tasks to complete!"

Server-Side

`/api/ladydog`

Executed Sequentially

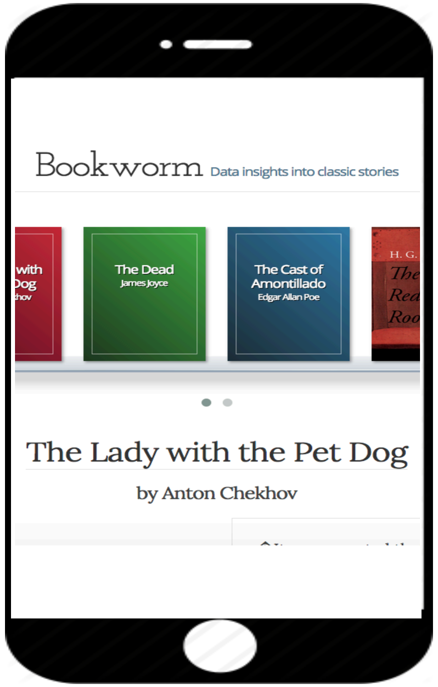
GetSentence→ getUniqueVocaburary→ GetDialog→getColon→ getSemiColon

Execute in Bulk!

Initial distribution

Motivating Real-world's Example: Bookworm

Client-Side



Bookworm Data insights into classic stories

The Lady with the Pet Dog

by Anton Chekhov

Server-Side

`/api/ladydog`

GetSentence

getUnique Vocaburary

GetDialog

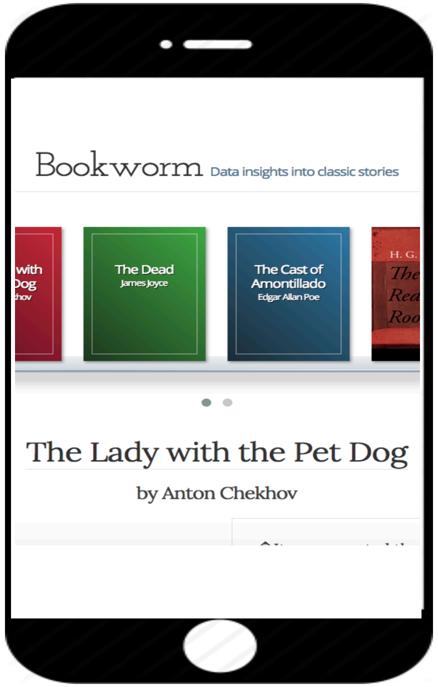
getColon

getSemiColon

Independent of each other

Motivating Real-world's Example: Bookworm

Client-Side



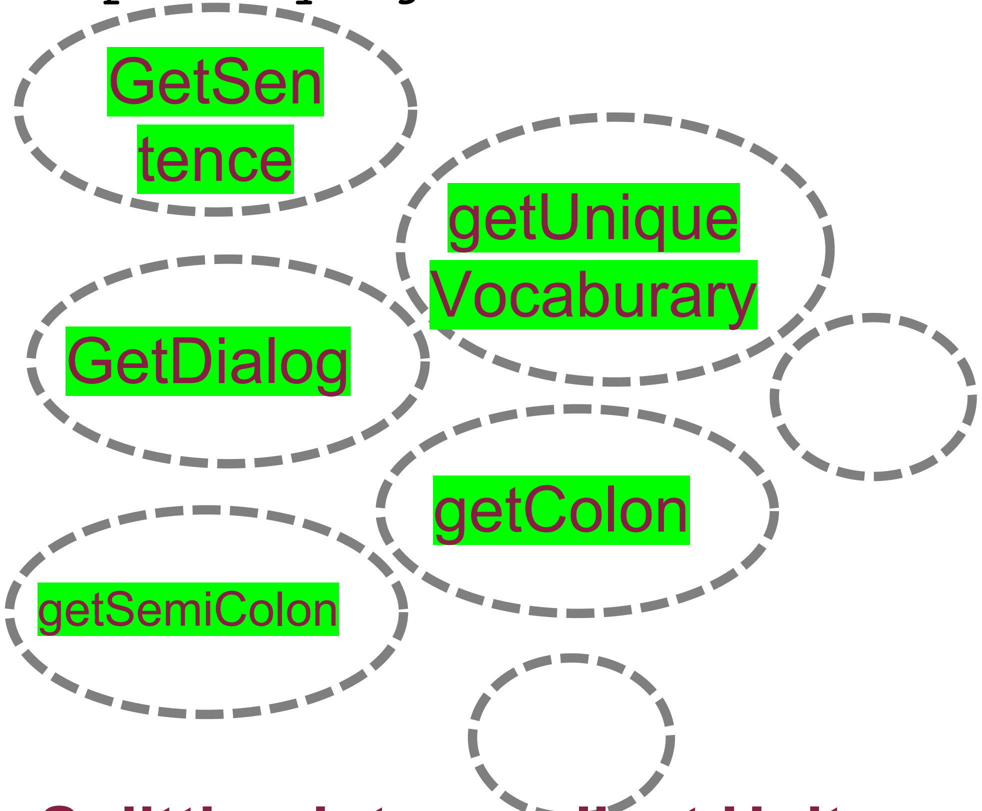
Bookworm Data insights into classic stories

The Lady with the Pet Dog

by Anton Chekhov

Server-Side

/api/ladydog



GetSentence

getUniqueVocaburary

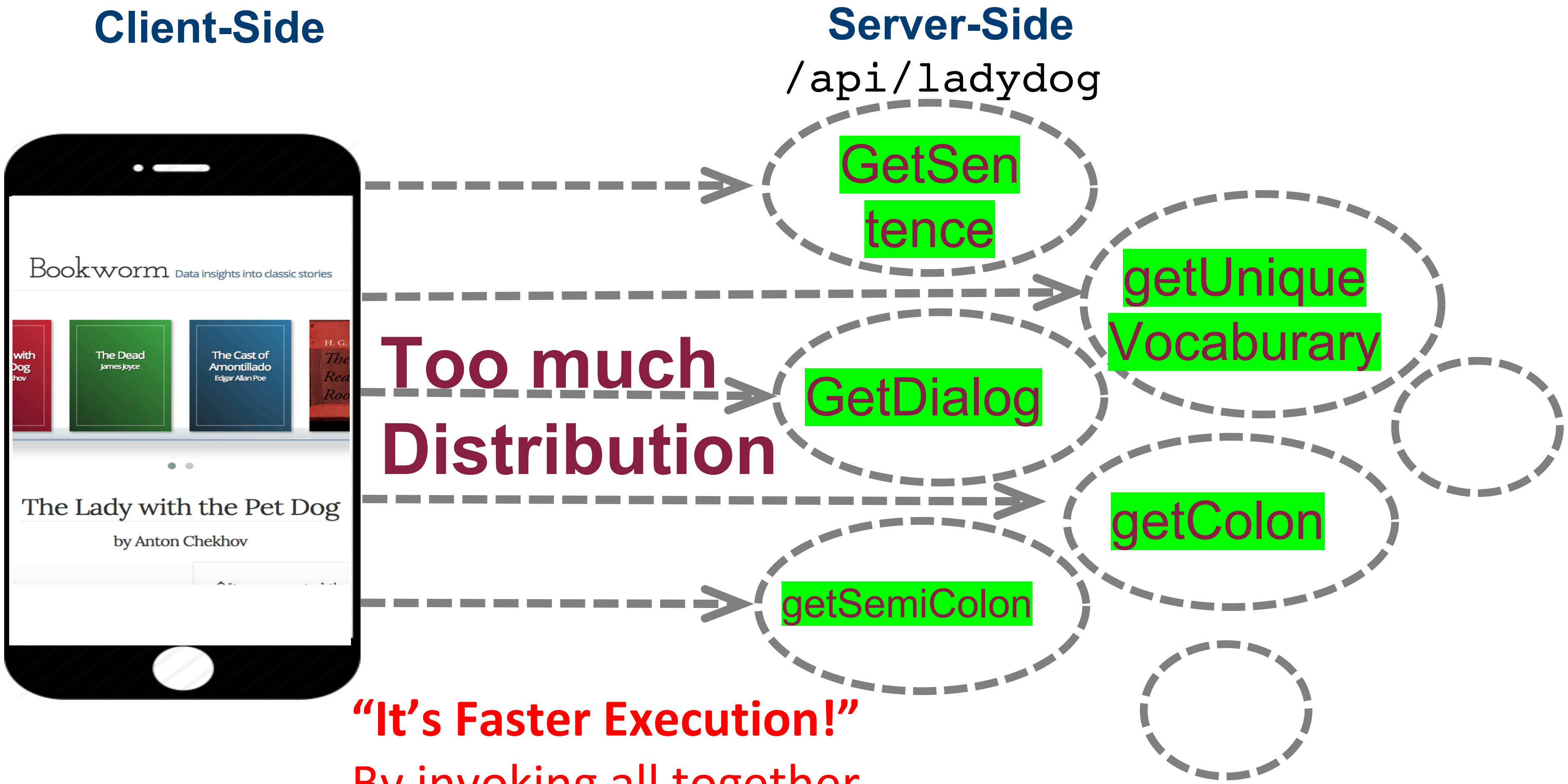
GetDialog

getColon

getSemiColon

Splitting into smallest Units to invoke Remotely

Motivating Real-world's Example: Bookworm



Too much Distribution

"It's Faster Execution!"

By invoking all together, "Asynchronously"

Goldilocks Principle

Too Crude

Too Fine

Goldilocks Principle

Too Crude

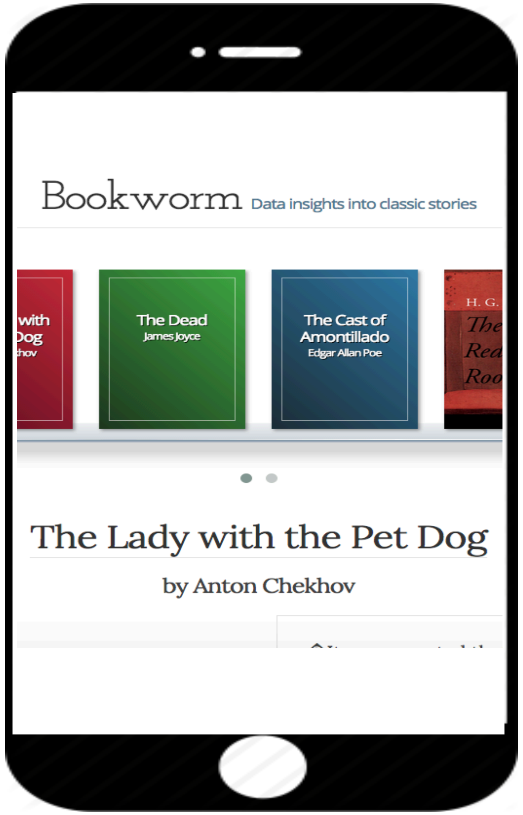
Just the right one!

Too Fine

D-Goldilocks

Client-Side

Server-Side (`/api/ladydog`)



Bookworm Data insights into classic stories

with Dog hov

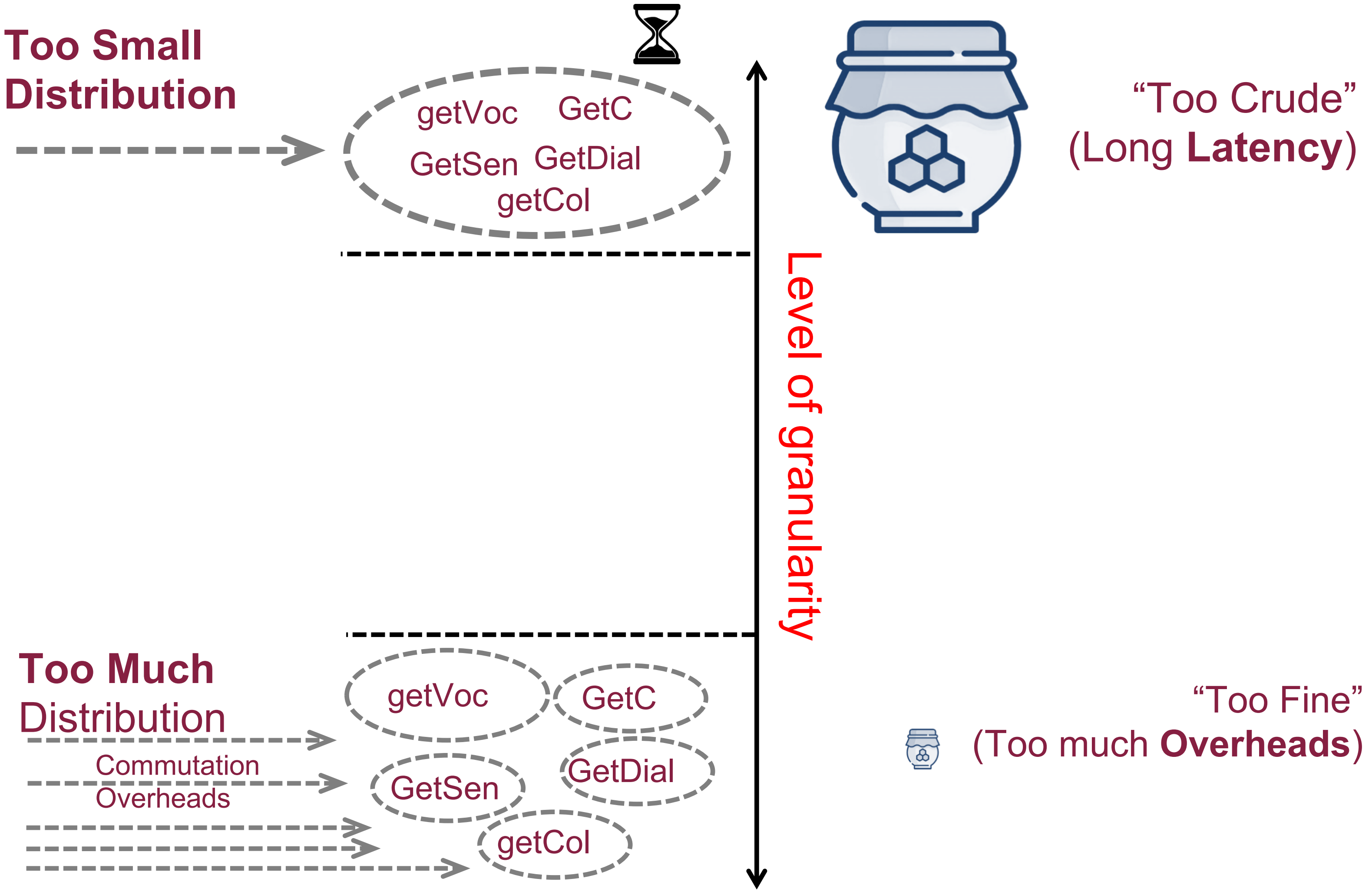
The Dead James Joyce

The Cast of Amontillado Edgar Allan Poe

H. G. The Red Roo

The Lady with the Pet Dog

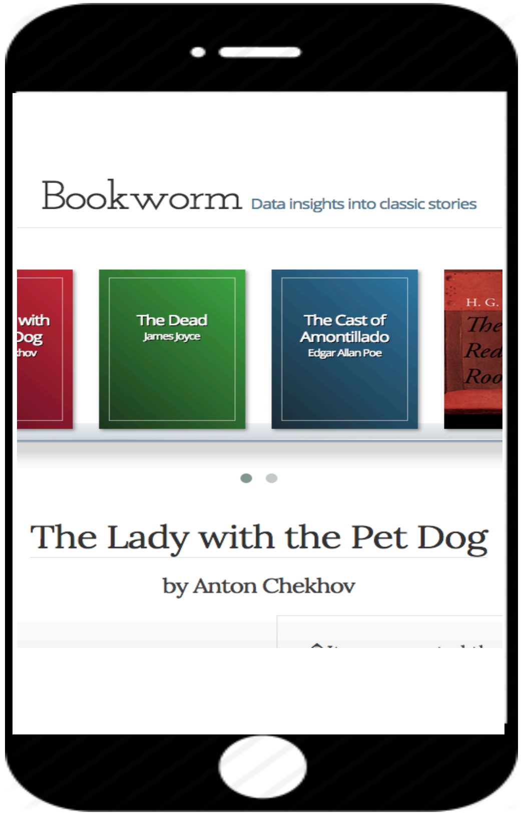
by Anton Chekhov



D-Goldilocks

Client-Side

Server-Side (`/api/ladydog`)



Bookworm Data insights into classic stories

with
Dog
hov

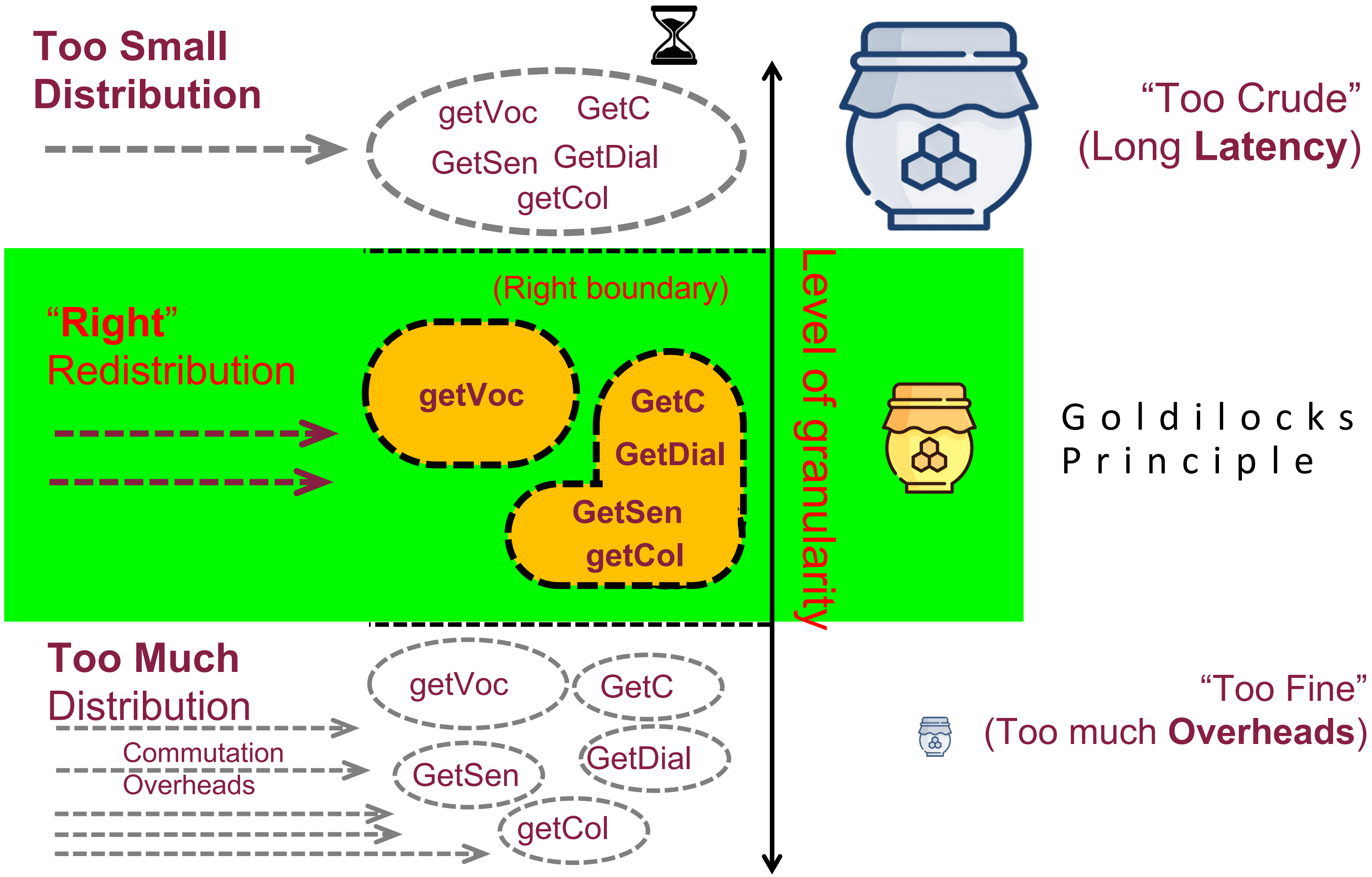
The Dead
James Joyce

The Cast of Amontillado
Edgar Allan Poe

H. G.
The
Red
Roo

The Lady with the Pet Dog

by Anton Chekhov



Too Small Distribution

getVoc GetC GetSen GetDial getCol

"Too Crude"
(Long **Latency**)

(Right boundary)

"Right" Redistribution

getVoc

GetC GetDial GetSen getCol

Level of granularity

Goldilocks Principle

Too Much Distribution

Commutation Overheads

getVoc GetC GetSen GetDial getCol

"Too Fine"
(Too much **Overheads**)

Problem Formulation

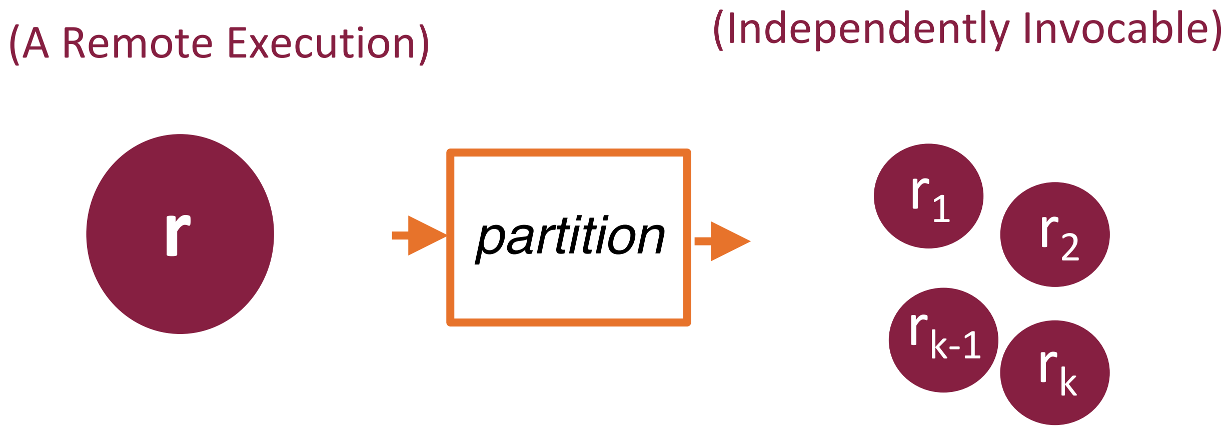
- Determine which functional distribution from the client’s standpoint would minimize the cost of distributed execution

$$C_{\text{Dist\_Exec}}(\mathbf{r}) = \alpha\cdot\underbrace{\text{latency}(\mathbf{r})}_{\substack{\text{Execution Time}\\ \text{(Performance)}}} + (1-\alpha)\cdot \underbrace{\Sigma\ \text{resource}(\mathbf{r})}_{\substack{\text{Consumed Resource}\\ \text{(Efficiency)}}}$$

Normalizing Parameter

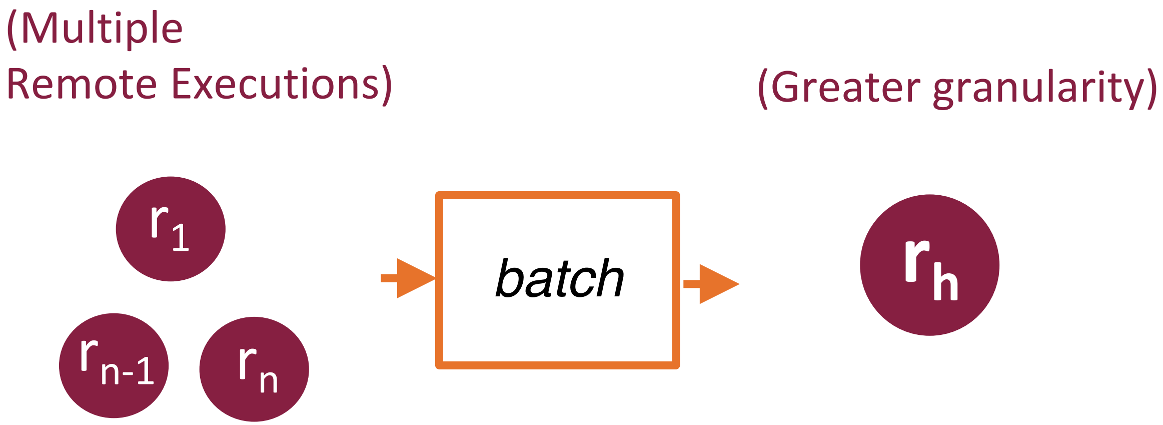
Problem Solution Outline

- Redistribution operations:
  - Partition
    - $[r_1,\ldots,r_k]$ = **partition**(r)
  - Batch
    - r_h= **batch**($[r_1,\ldots,r_n]$)



(A Remote Execution)

(Independently Invocable)

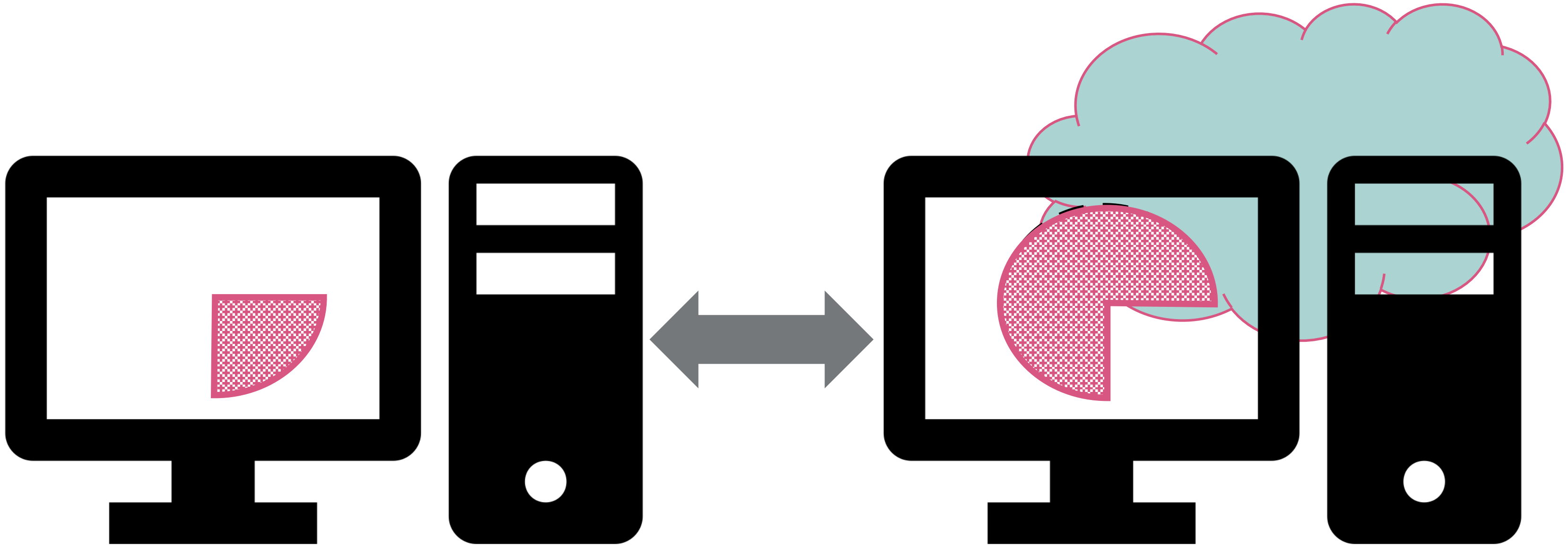


(Multiple
Remote Executions)

(Greater granularity)

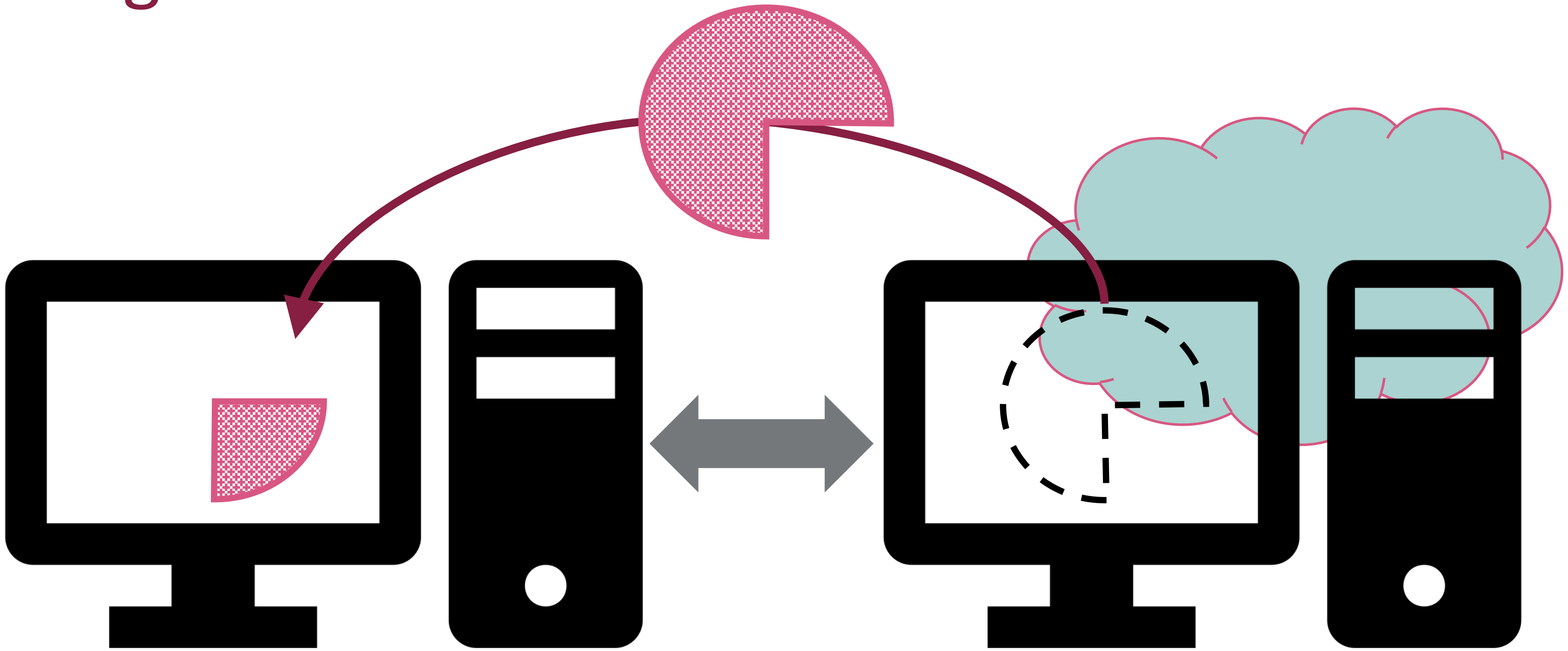
How to restructure Remote Services?

- **Client Insourcing Refactoring** [WWW '20]
  - Undoing Distribution



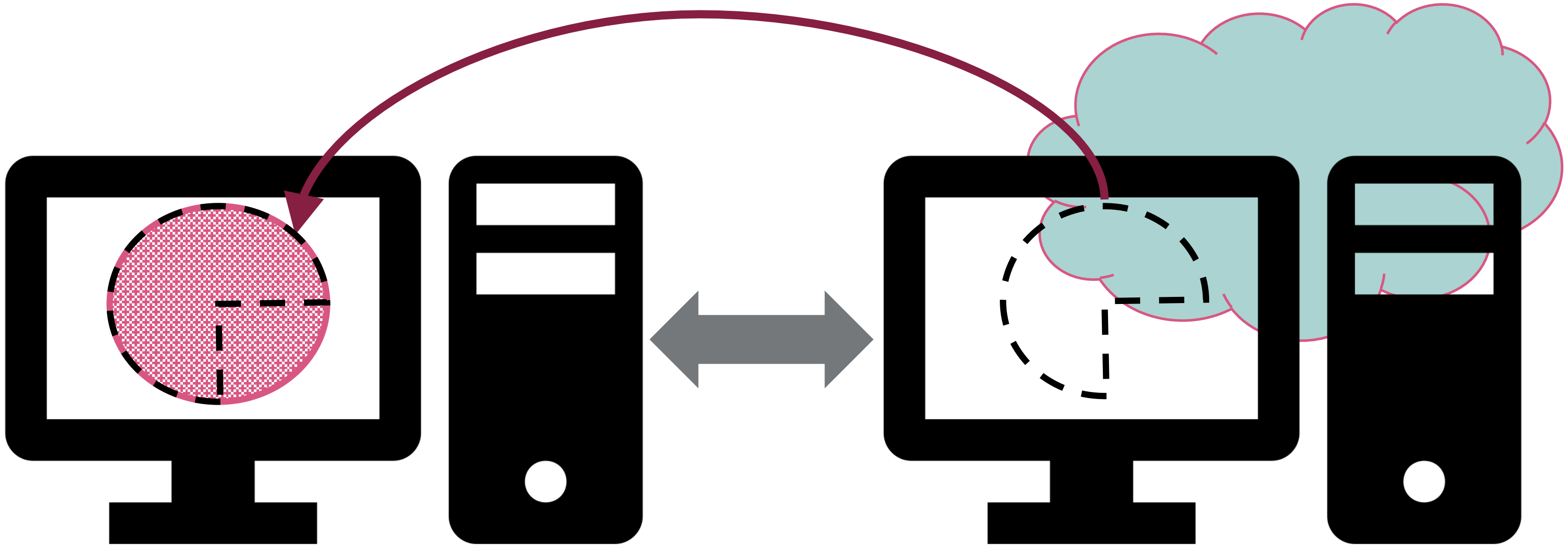
How to restructure Remote Services?

- **Client Insourcing Refactoring** [WWW ’20]
  - Undoing Distribution



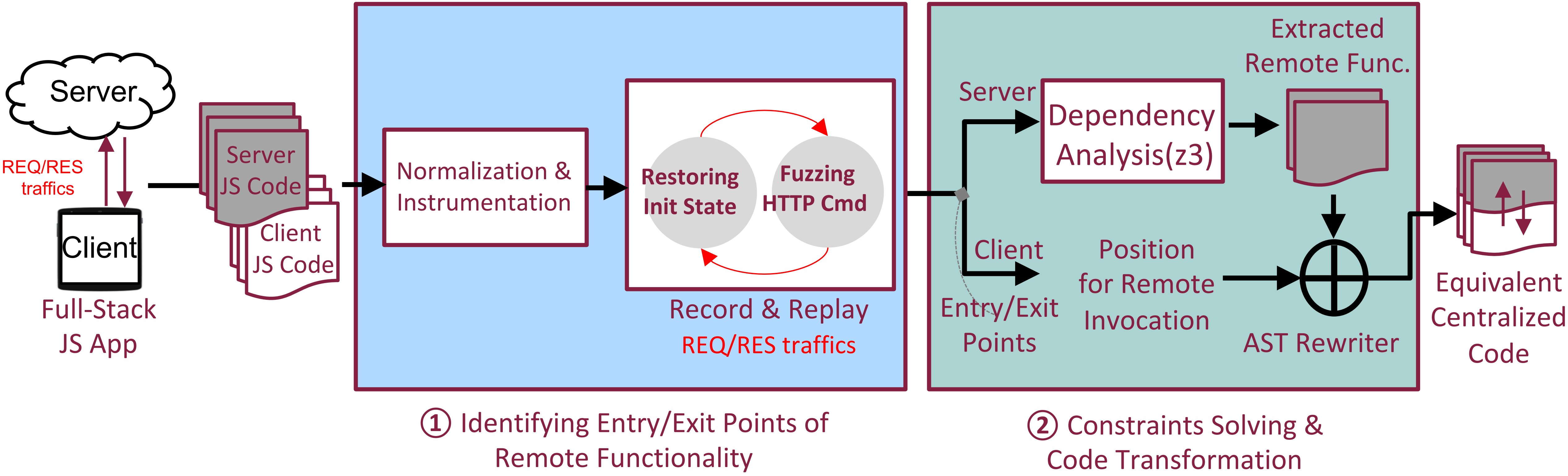
How to restructure Remote Services?

- **Client Insourcing Refactoring** [WWW ’20]
  - Undoing Distribution



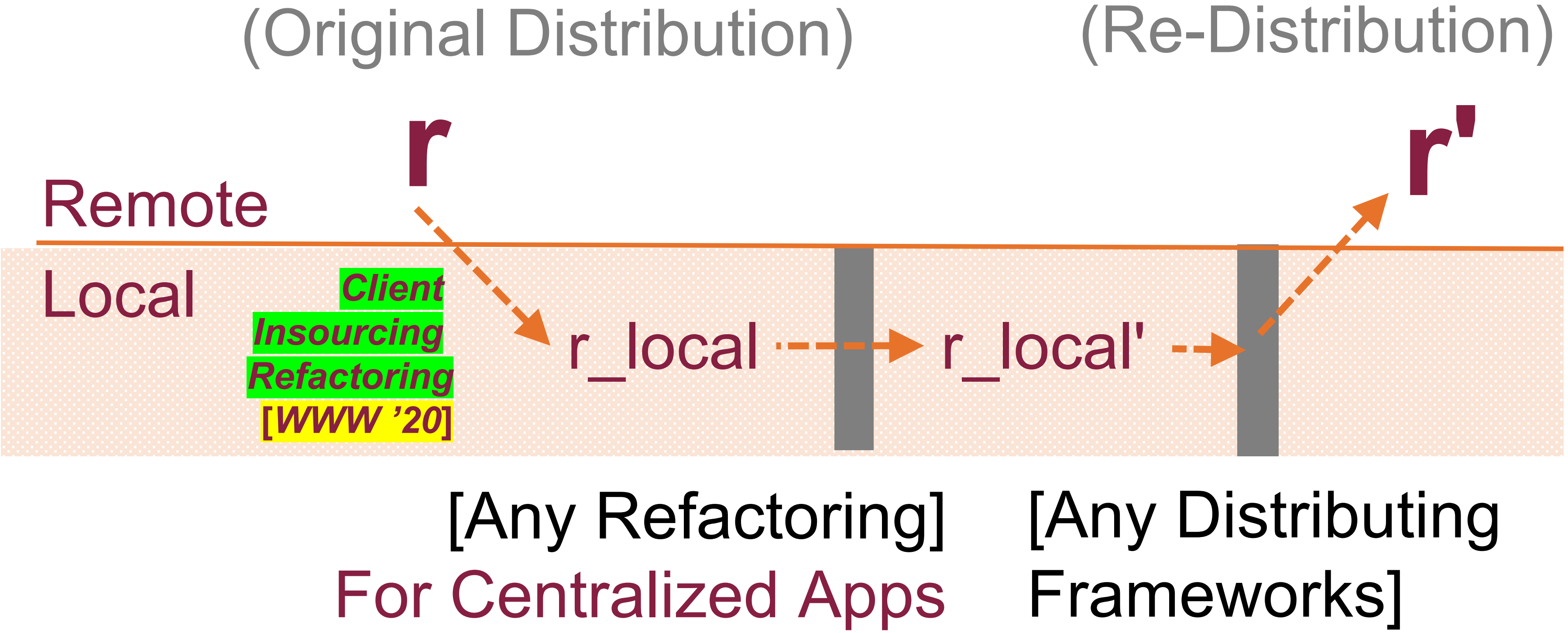
How to restructure Remote Services?

- **Client Insourcing Refactoring** [WWW ’20]



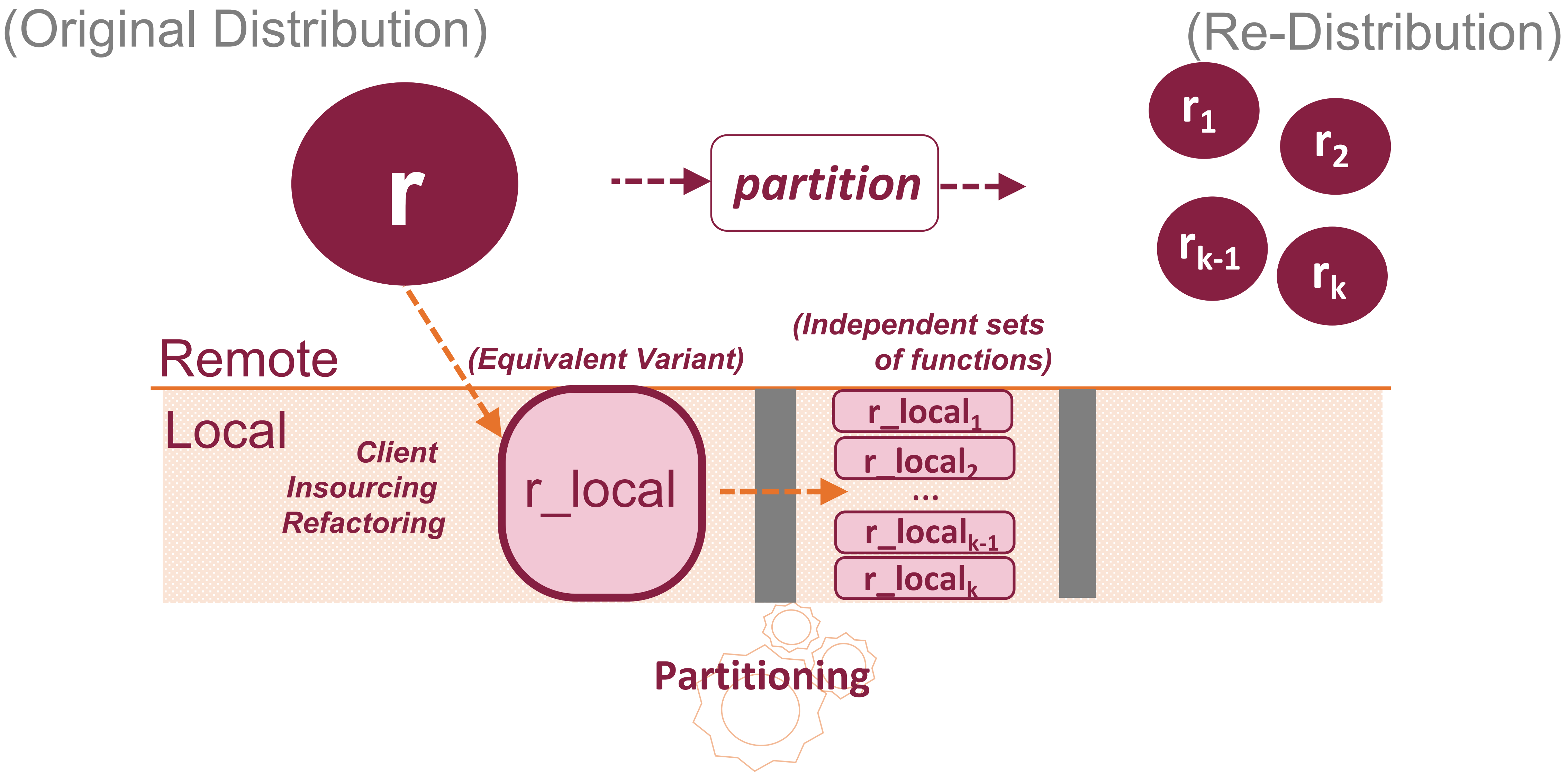
How to restructure Remote Services?

- **Client Insourcing Refactoring as re-distribution framework**

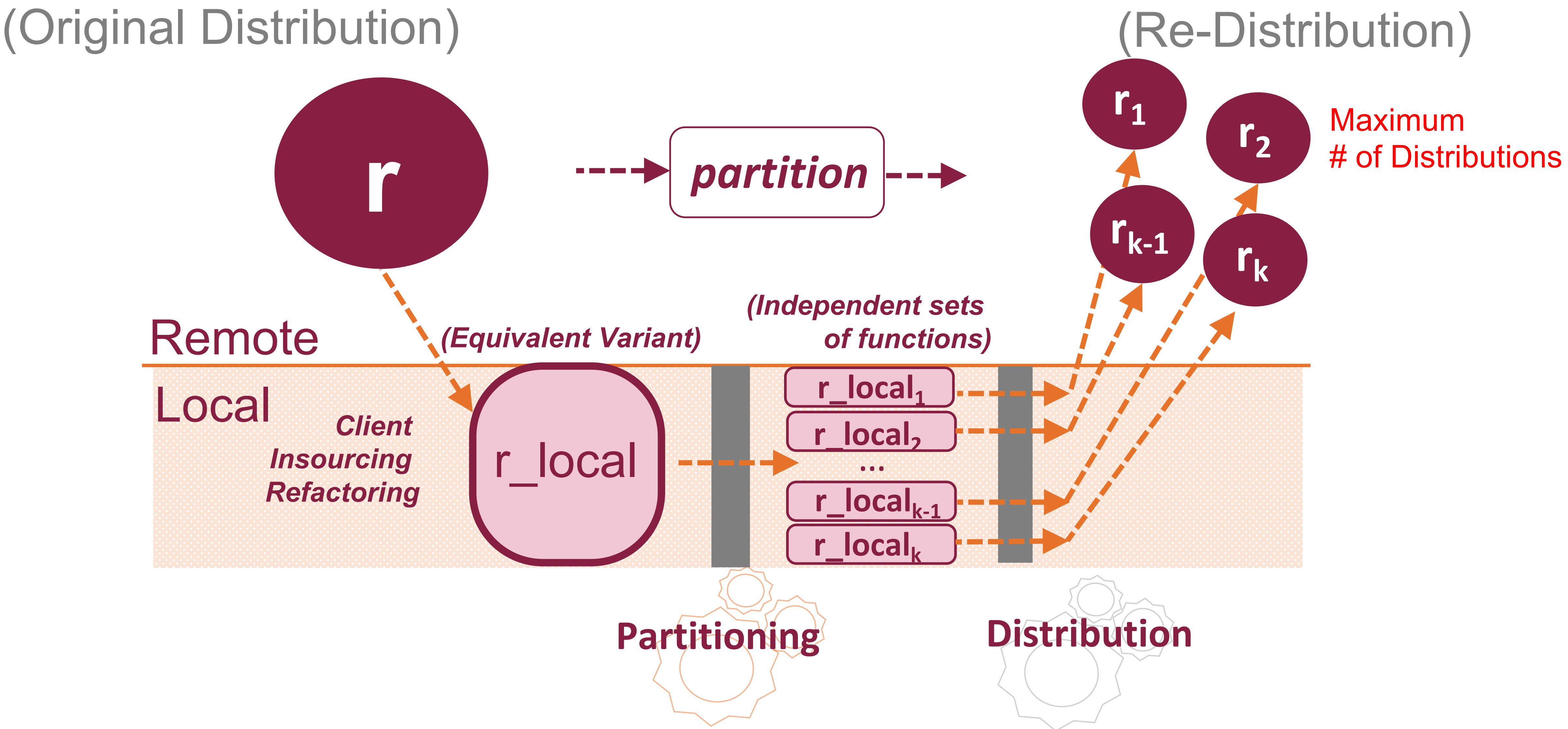


[Kwon ICDCS'13,
EXTREMEJS '12,..]

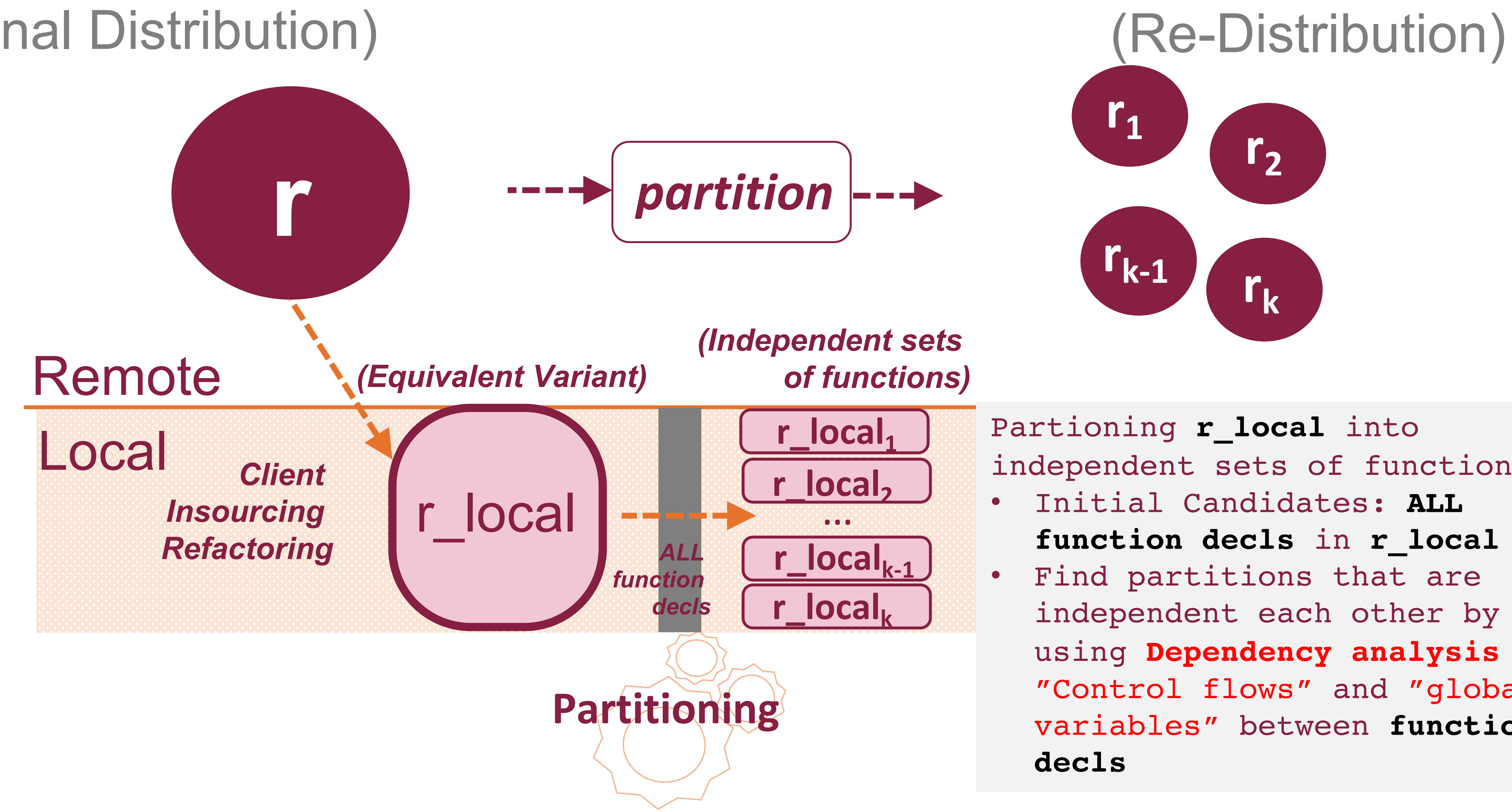
Restructuring: Partition



Restructuring: Partition



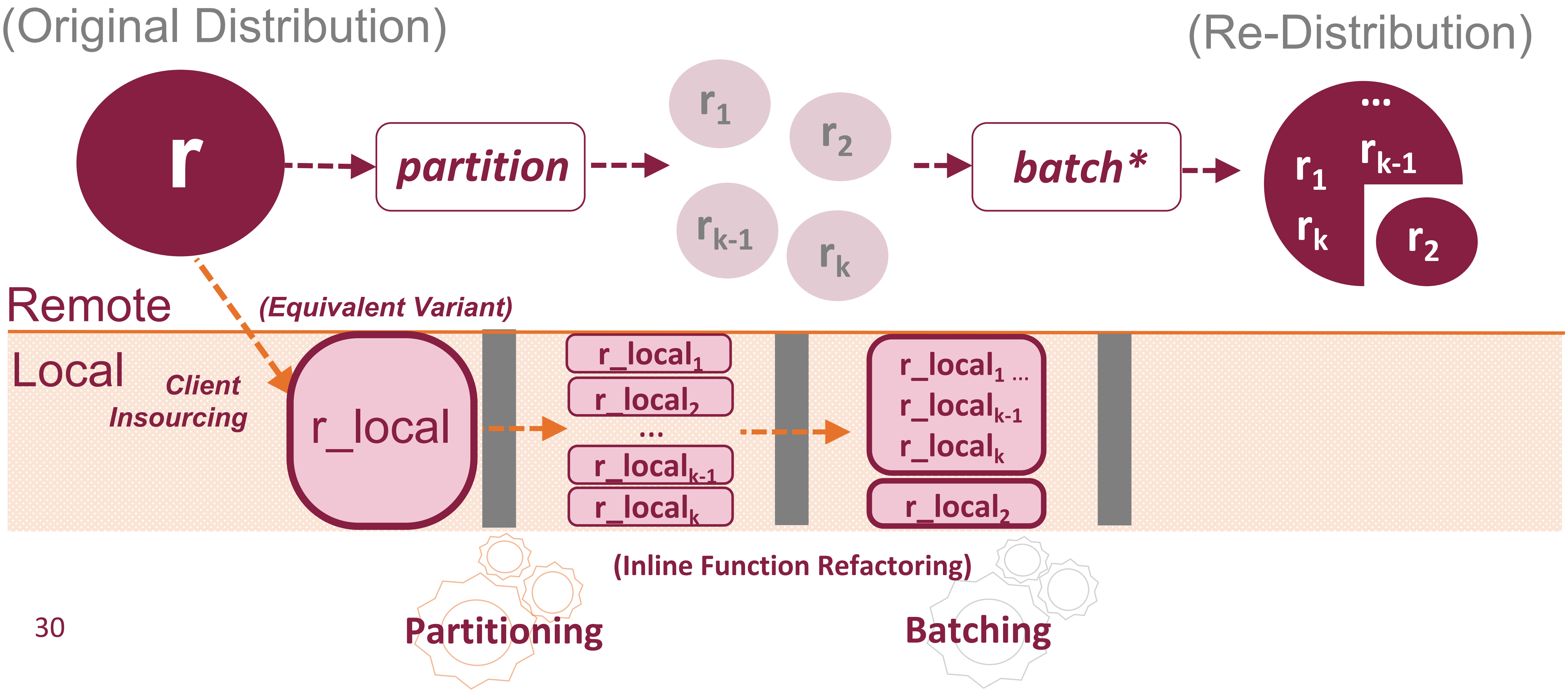
Restructuring: Partition



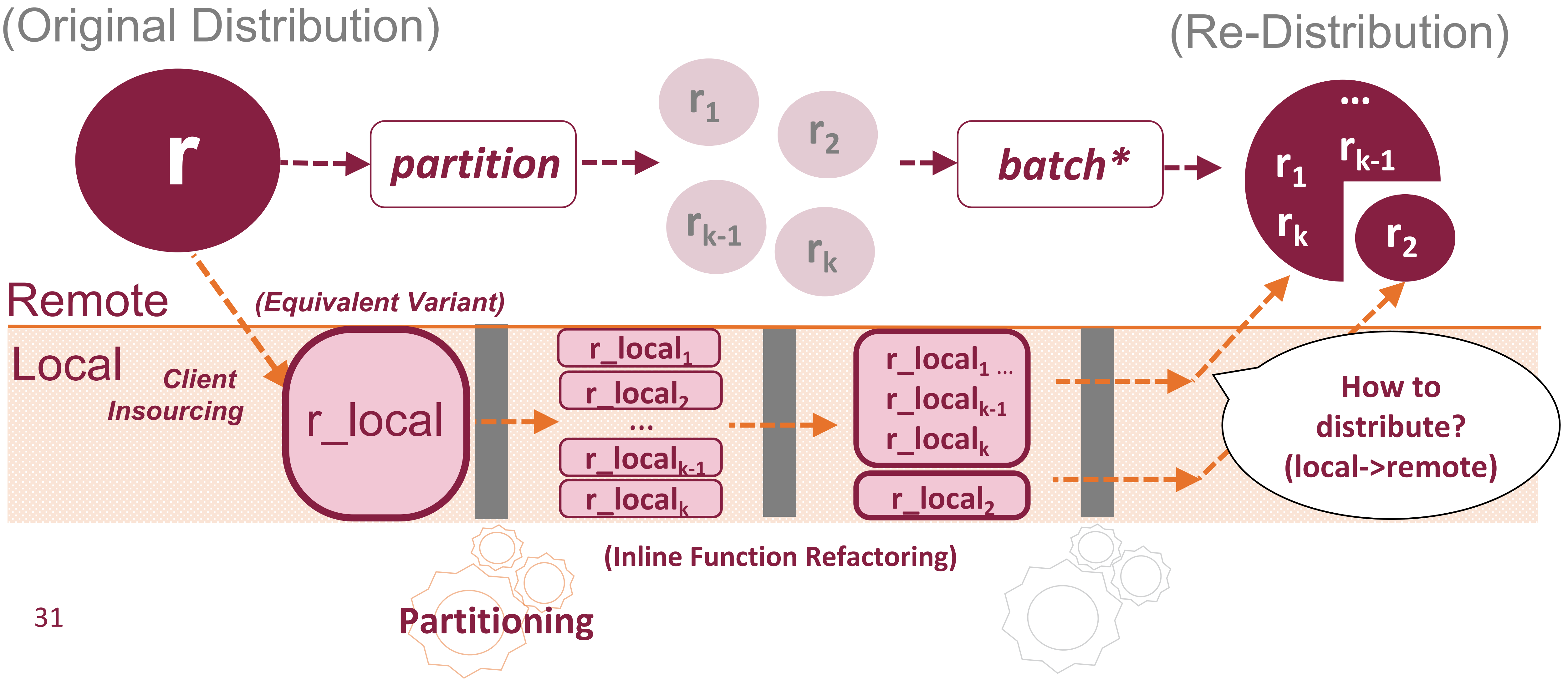
Partioning **r_local** into independent sets of functions

- Initial Candidates: **ALL function decls** in **r_local**
- Find partitions that are independent each other by using **Dependency analysis** for "Control flows" and "global variables" between **function decls**

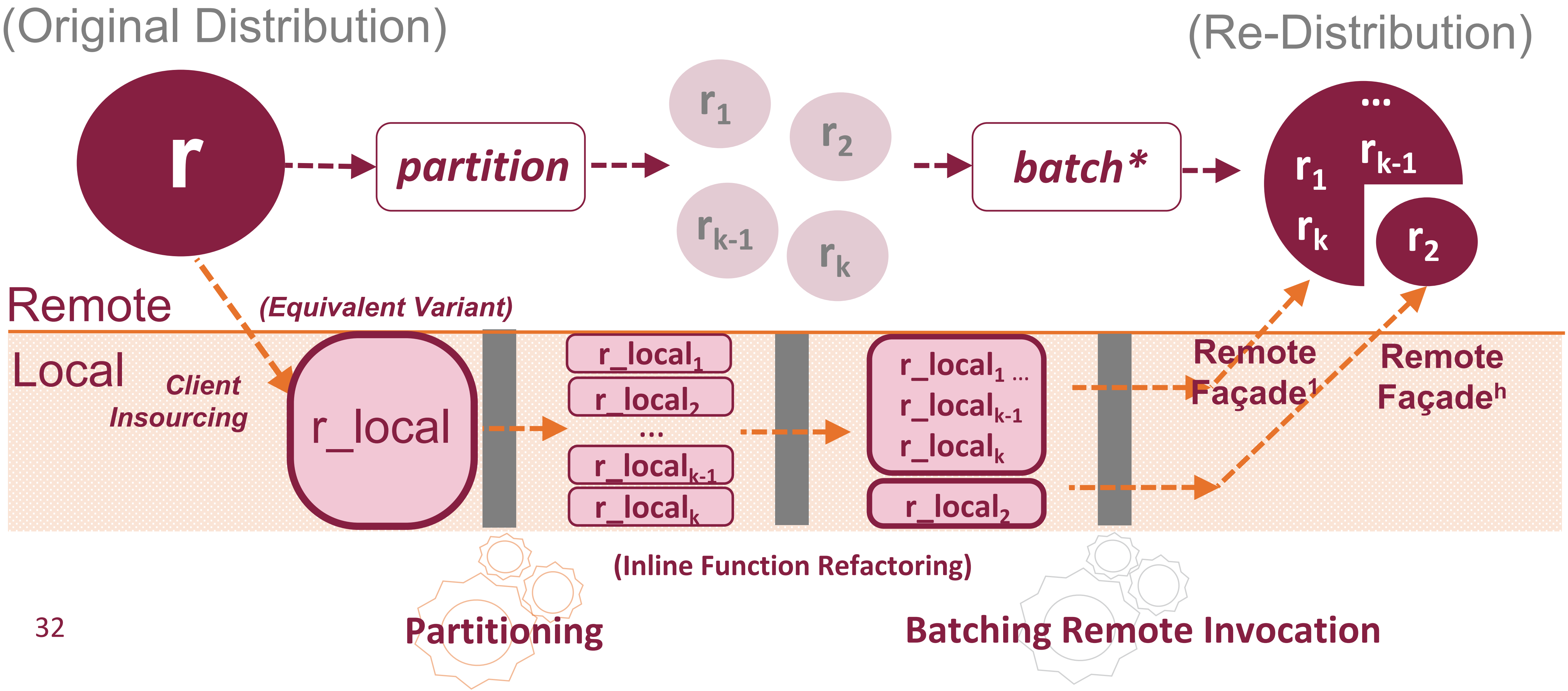
Restructuring: Partition & Batch



Restructuring: Partition & Batch

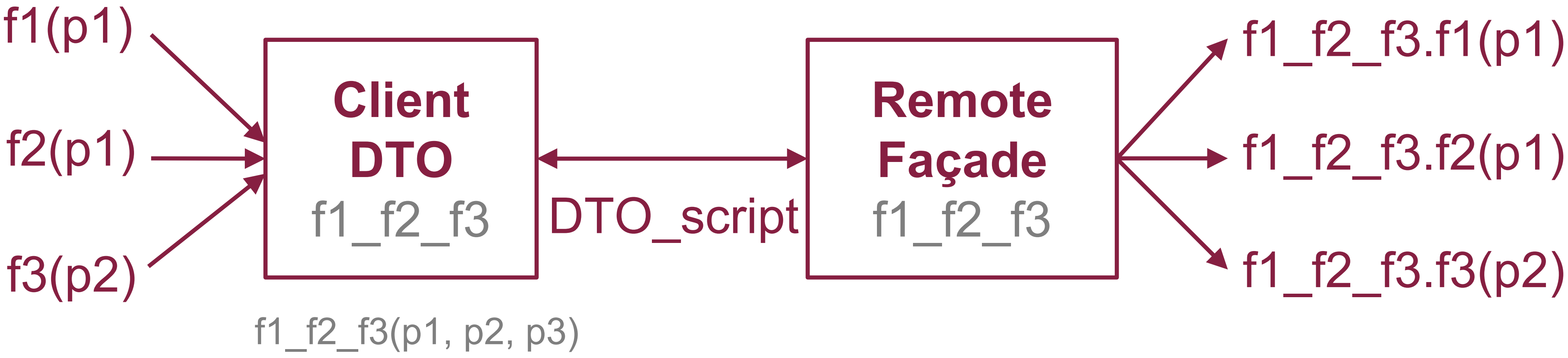


Restructuring: Partition & Batch



Batching Remote Invocation(Batch)

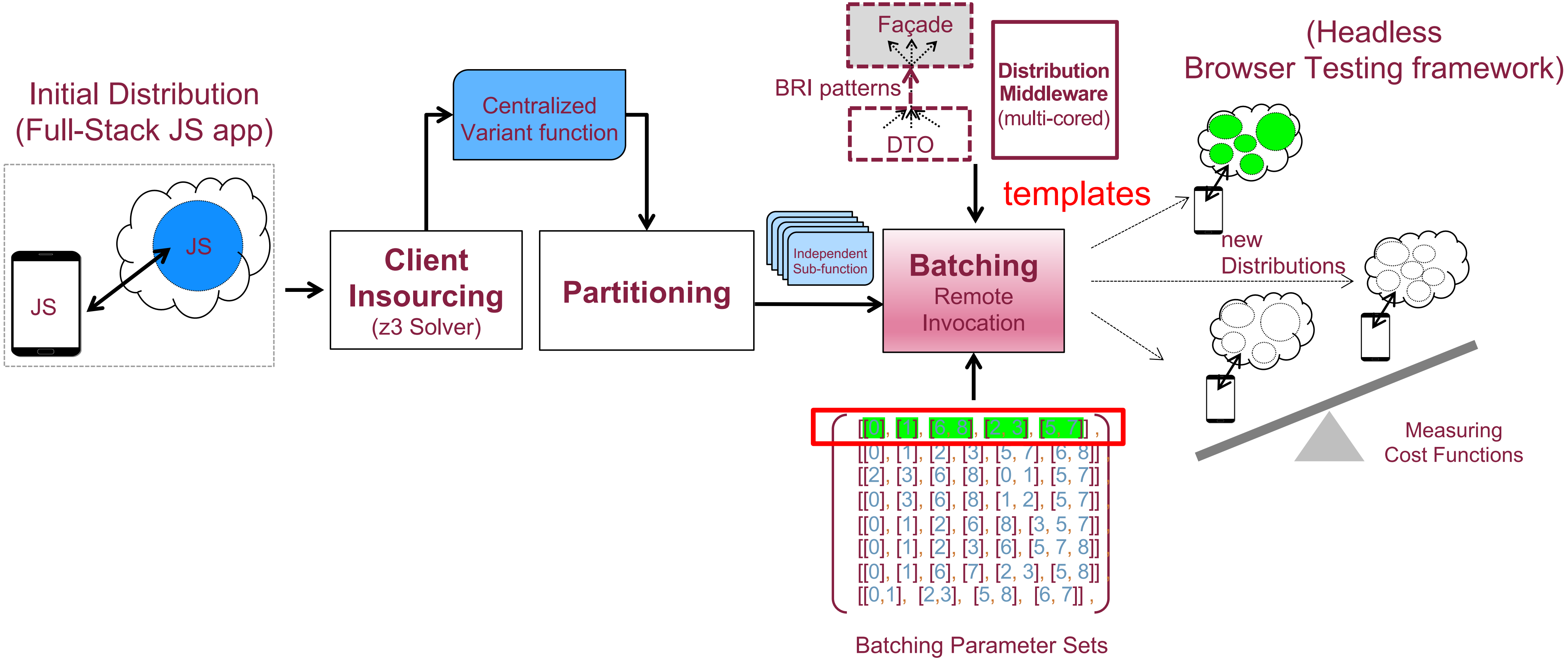
- **Distributing Programing Pattern** [Fowler '02, Ibrahim et.al ECOOP '09]



The **Client DTO** stub accumulates the fine-grained service invocations then, transfers them (parameters) in bulk.

The **remote Façade** function sequentially invokes the bundled services. Then, it combines their execution results and returns in bulk.

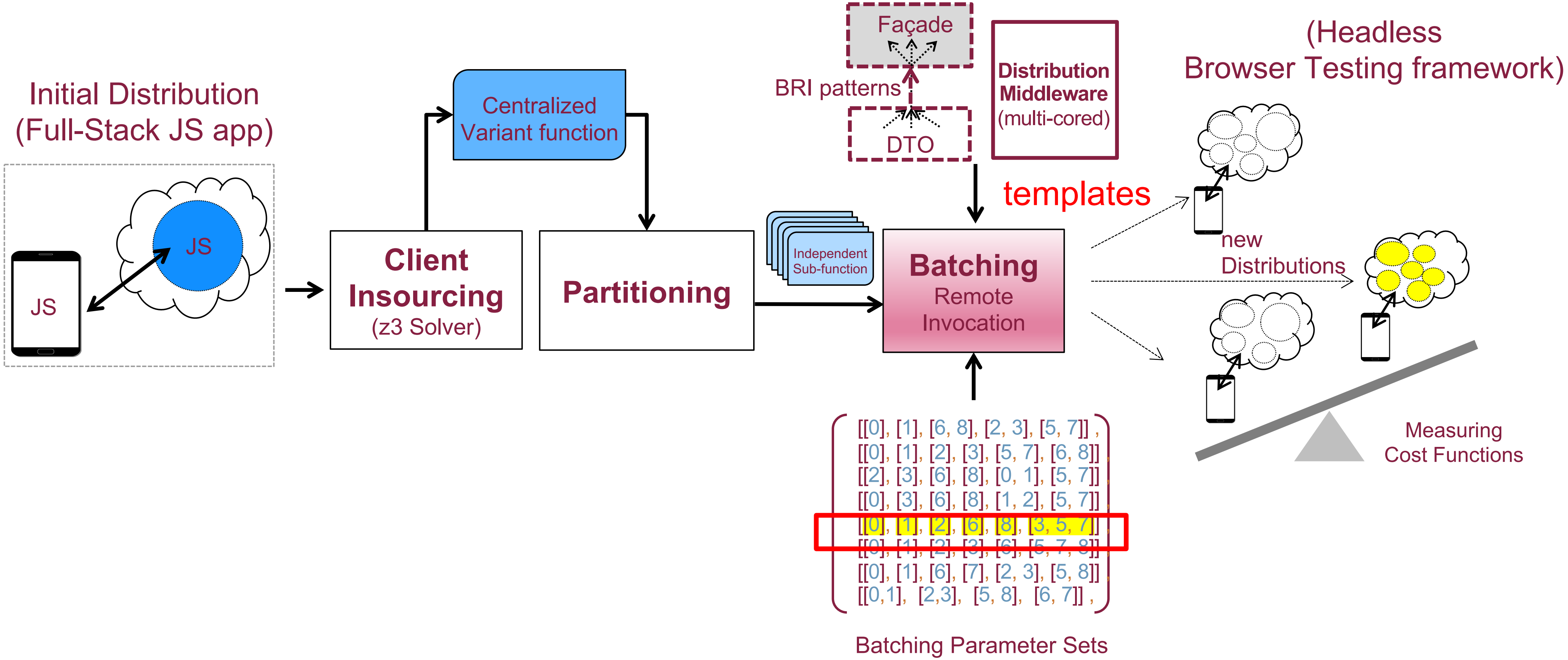
Process for D-GOLDILOCKS



[[0], [1], [6, 8], [2, 3], [5, 7]] ,
[[0], [1], [2], [3], [5, 7], [6, 8]] ,
[[2], [3], [6], [8], [0, 1], [5, 7]] ,
[[0], [3], [6], [8], [1, 2], [5, 7]] ,
[[0], [1], [2], [6], [8], [3, 5, 7]] ,
[[0], [1], [2], [3], [6], [5, 7, 8]] ,
[[0], [1], [6], [7], [2, 3], [5, 8]] ,
[[0,1], [2,3], [5, 8], [6, 7]] ,

Batching Parameter Sets

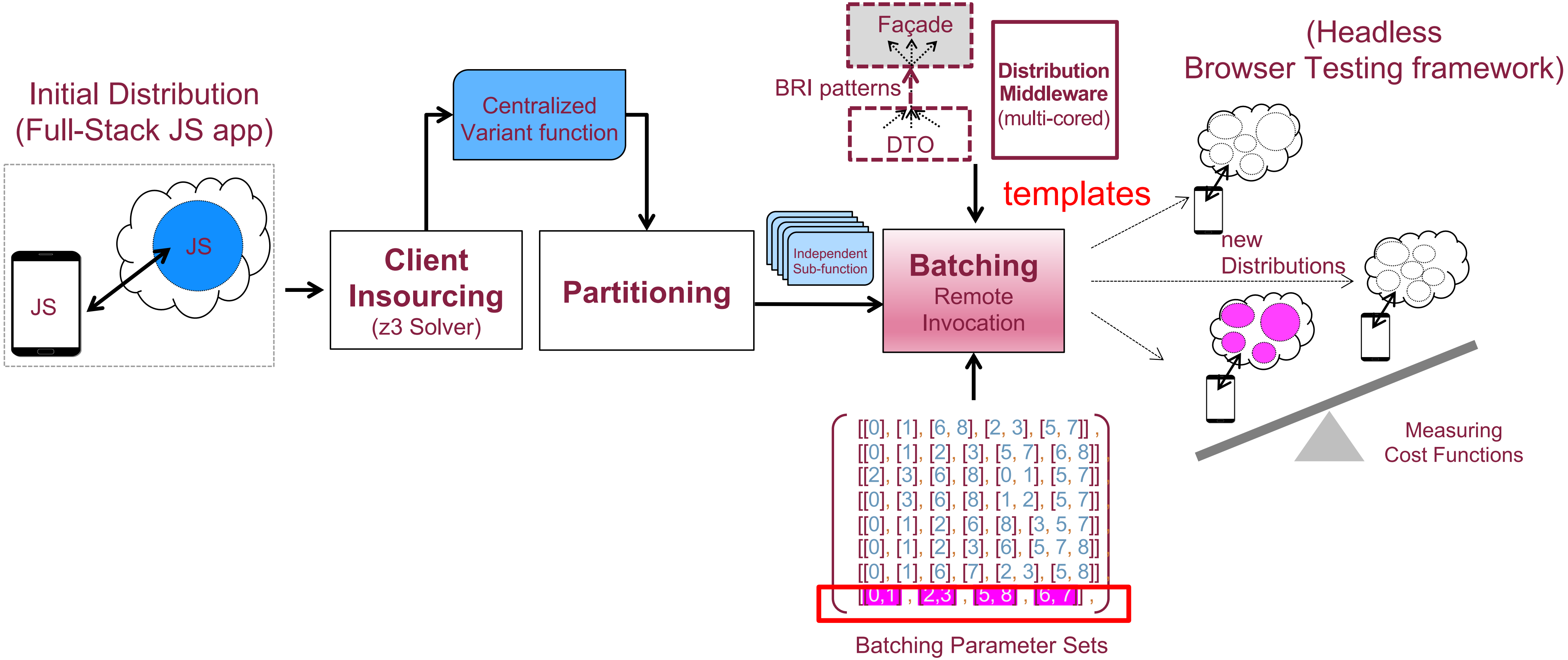
Process for D-GOLDILOCKS



[[0], [1], [6, 8], [2, 3], [5, 7]] ,
[[0], [1], [2], [3], [5, 7], [6, 8]] ,
[[2], [3], [6], [8], [0, 1], [5, 7]] ,
[[0], [3], [6], [8], [1, 2], [5, 7]] ,
[[0], [1], [2], [6], [8], [3, 5, 7]] ,
[[0], [1], [2], [3], [6], [5, 7, 8]] ,
[[0], [1], [6], [7], [2, 3], [5, 8]] ,
[[0,1], [2,3], [5, 8], [6, 7]] ,

Batching Parameter Sets

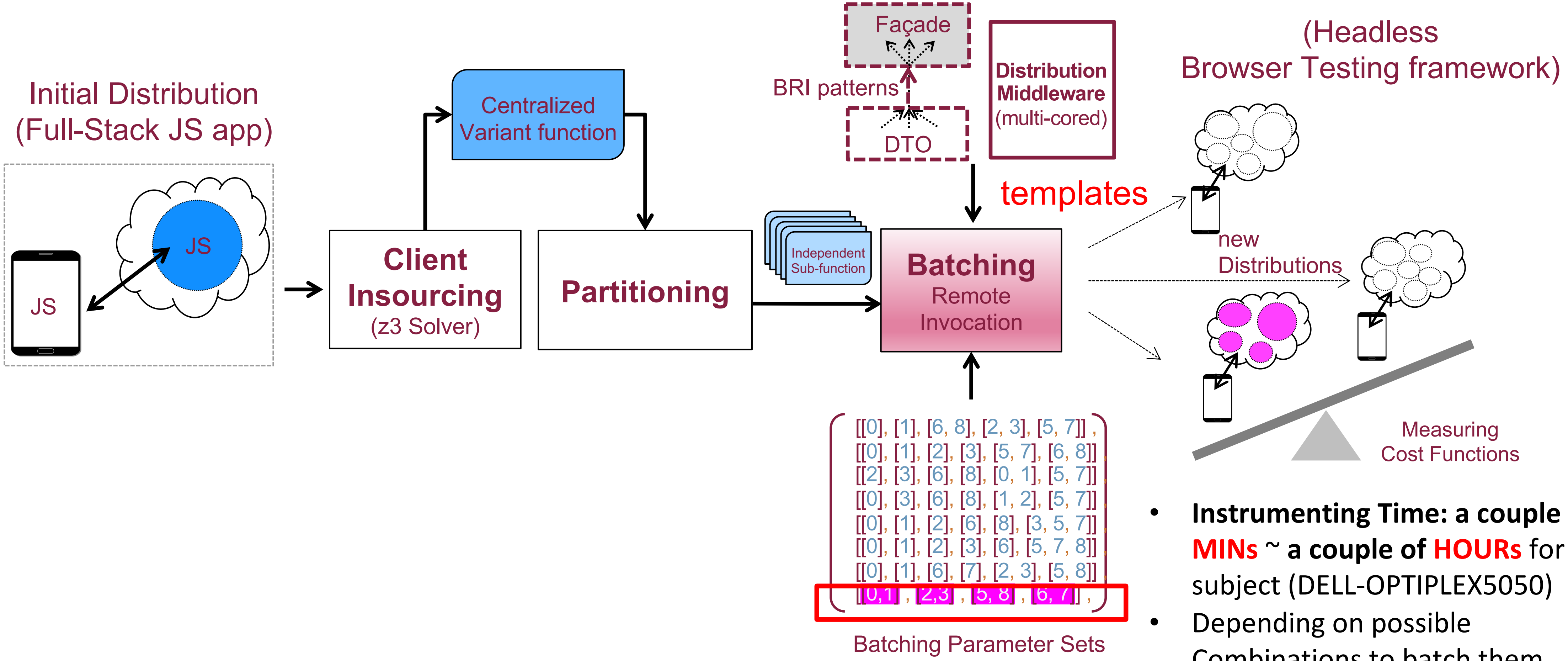
Process for D-GOLDILOCKS



```
[[0], [1], [6, 8], [2, 3], [5, 7]] ,
[[0], [1], [2], [3], [5, 7], [6, 8]] ,
[[2], [3], [6], [8], [0, 1], [5, 7]] ,
[[0], [3], [6], [8], [1, 2], [5, 7]] ,
[[0], [1], [2], [6], [8], [3, 5, 7]] ,
[[0], [1], [2], [3], [6], [5, 7, 8]] ,
[[0], [1], [6], [7], [2, 3], [5, 8]] ,
[[0,1] , [2,3] , [5, 8] , [6, 7]] ,
```

Batching Parameter Sets

Process for D-GOLDILOCKS



```
[[0], [1], [6, 8], [2, 3], [5, 7]] ,
[[0], [1], [2], [3], [5, 7], [6, 8]] ,
[[2], [3], [6], [8], [0, 1], [5, 7]] ,
[[0], [3], [6], [8], [1, 2], [5, 7]] ,
[[0], [1], [2], [6], [8], [3, 5, 7]] ,
[[0], [1], [2], [3], [6], [5, 7, 8]] ,
[[0], [1], [6], [7], [2, 3], [5, 8]] ,
[[0,1] , [2,3] , [5, 8] , [6, 7]] ,
```

Batching Parameter Sets

- **Instrumenting Time: a couple of MINs ~ a couple of HOURs** for each subject (DELL-OPTIPLEX5050)
- Depending on possible Combinations to batch them

Evaluation: Research Questions

- **RQ1:—Value:** How much **programmer effort** is **saved** by D-GOLDILOCKS's automatic redistribution operations?
- **RQ2:—Cost Model Correctness:** How applying the partition and batch operations affect the distributed execution's "**latency**" and "**consumed resources**" attributes?
- **RQ3:—Utility of Cost Model for Redistribution:** How useful is the **cost function** for guiding redistribution decisions?
- **RQ4:—Energy Consumption:** What is the effect of redistribution on the amount of **energy consumed** by the client?

Subject Full-stack JavaScript Apps
:Original Performance and Efficiency

Original ”Latency” Original ”CPU utilization”(Resource)

Remote Services	L(ms)	ΣTCPU	f^{LOC}_{CI}	f_{decl}	f^{ind}_{sub}	IDI
/api/ladypet	77.83	337	394	9	8	1.6M
/api/thedea	164.62	695	394	9	8	1.6M
/api/bigtrip	42.11	304	394	9	8	1.6M
… (Total 12 Subjects from 4 Full-Stack Apps)						
/string-fasta	29.85	328	38	5	2	76
/cflow-rec	35.43	326	49	4	3	245
/prprty/brokers	20.64	323	379	3	3	1.5K

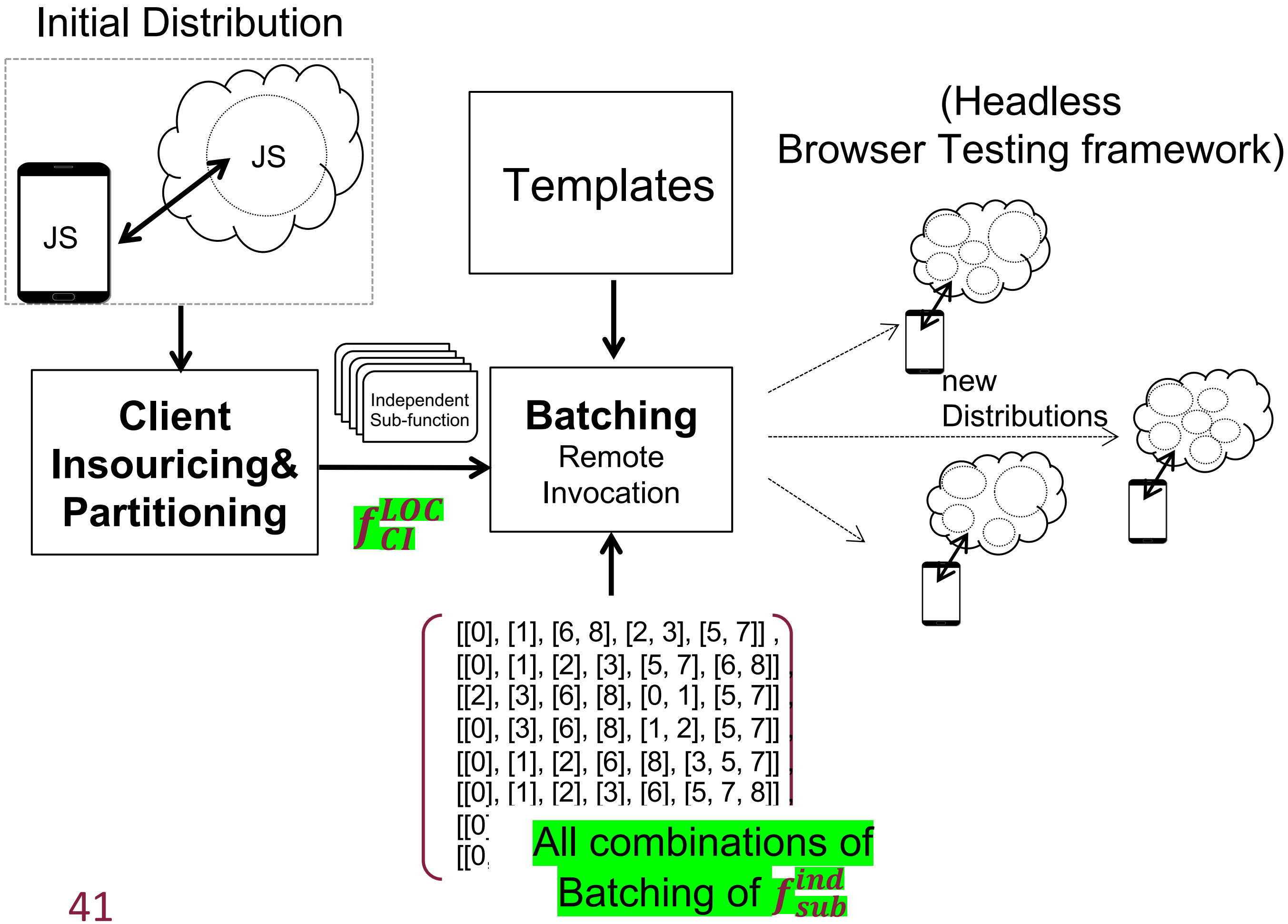
Subject Full-stack JavaScript Apps

:How many Lines of Code(f_{CI}^{LOC}) and Independent sub-functions are in the original remote functionality(**Centralized Variant**)?

Remote Services	L(ms)	ΣTCPU	f_{CI}^{LOC}	f_{decl}	f_{sub}^{ind}	IDI
/api/ladypet	77.83	337	394	9	8	1.6M
/api/thedea	164.62	695	394	9	8	1.6M
/api/bigtrip	42.11	304	394	9	8	1.6M
… (Total 12 Subjects from 4 Full-Stack Apps)						
/string-fasta	29.85	328	38	5	2	76
/cflow-rec	35.43	326	49	4	3	245
/prprty/brokers	20.64	323	379	3	3	1.5K

Subject Full-stack JavaScript Apps

:**RQ1 Value** : How much **Programmer Effort** is **Saved** by D-GOLDILOCKS’s automatic redistribution operations?



Initial Distribution

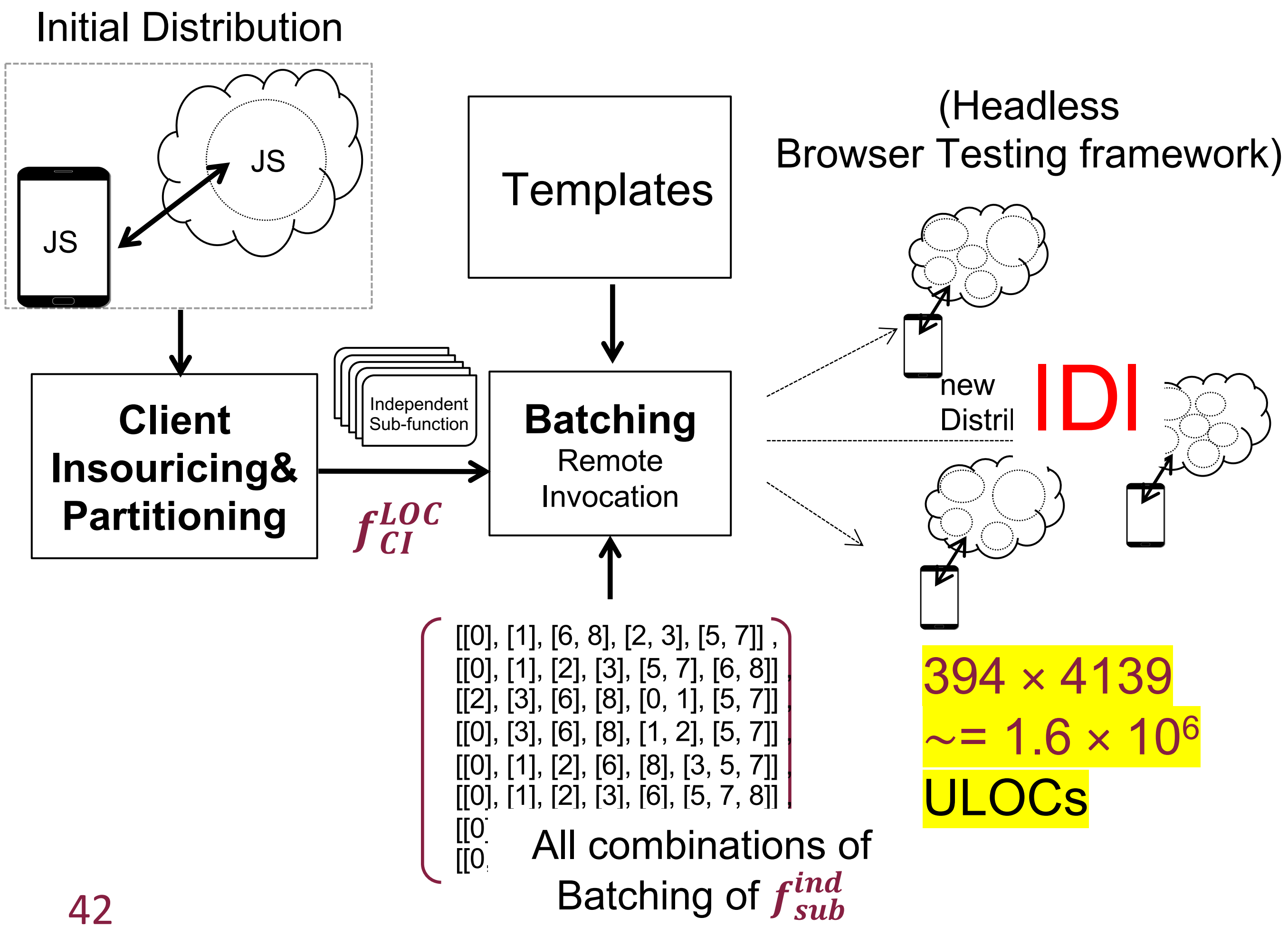
[[0], [1], [6, 8], [2, 3], [5, 7]] ,
[[0], [1], [2], [3], [5, 7], [6, 8]] ,
[[2], [3], [6], [8], [0, 1], [5, 7]] ,
[[0], [3], [6], [8], [1, 2], [5, 7]] ,
[[0], [1], [2], [6], [8], [3, 5, 7]] ,
[[0], [1], [2], [3], [6], [5, 7, 8]] ,
[[0
[[0

All combinations of
Batching of f_{sub}^{ind}

f_{CI}^{LOC}	f_{decl}	f_{sub}^{ind}	|D|
394	9	8	1.6M
394	9	8	1.6M
394	9	8	1.6M
	....		
38	5	2	76
49	4	3	245
379	3	3	1.5K

Subject Full-stack JavaScript Apps

:**RQ1** Value : How much **Programmer Effort** is **Saved** by D-GOLDILOCKS’s automatic redistribution operations?



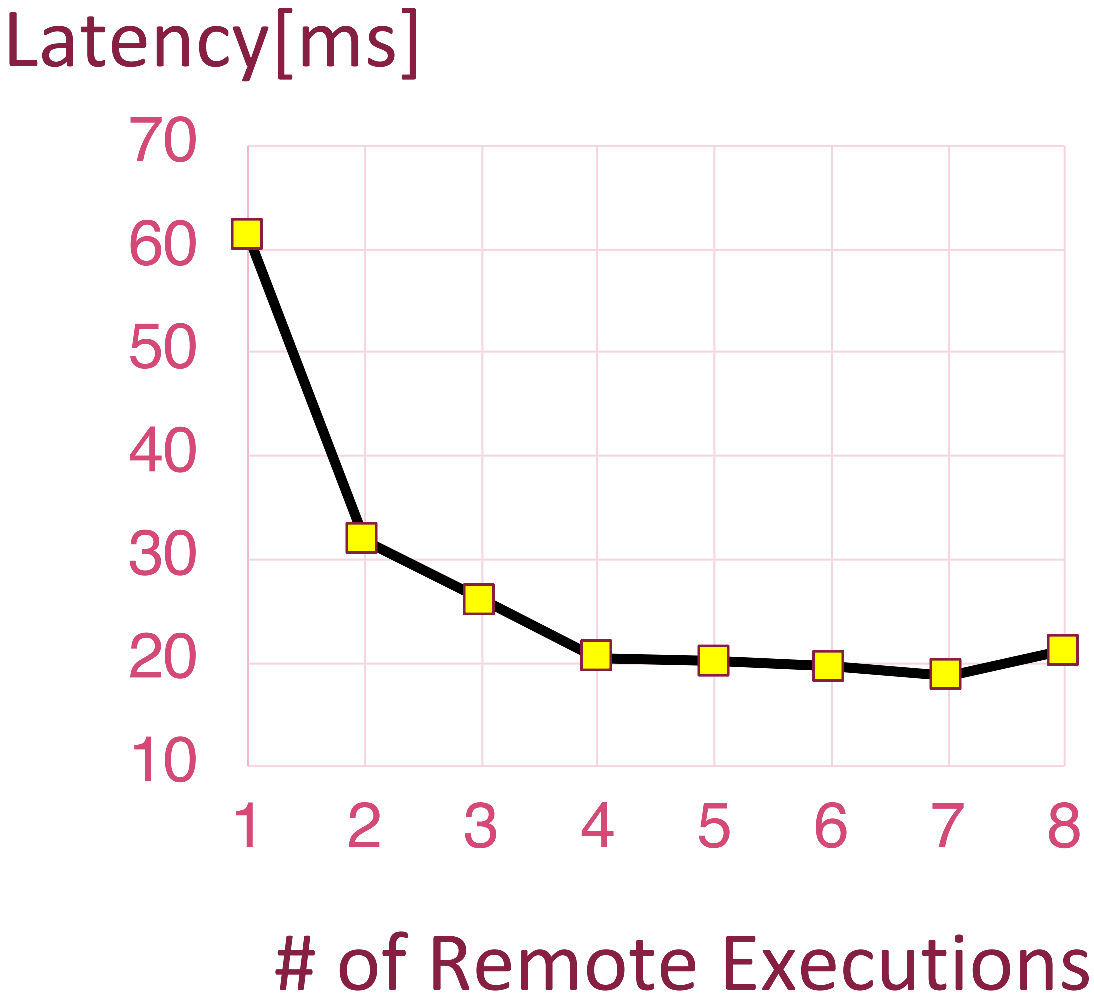
[[0], [1], [6, 8], [2, 3], [5, 7]] ,
[[0], [1], [2], [3], [5, 7], [6, 8]] ,
[[2], [3], [6], [8], [0, 1], [5, 7]] ,
[[0], [3], [6], [8], [1, 2], [5, 7]] ,
[[0], [1], [2], [6], [8], [3, 5, 7]] ,
[[0], [1], [2], [3], [6], [5, 7, 8]] ,
[[0
[[0

All combinations of Batching of f_{sub}^{ind}

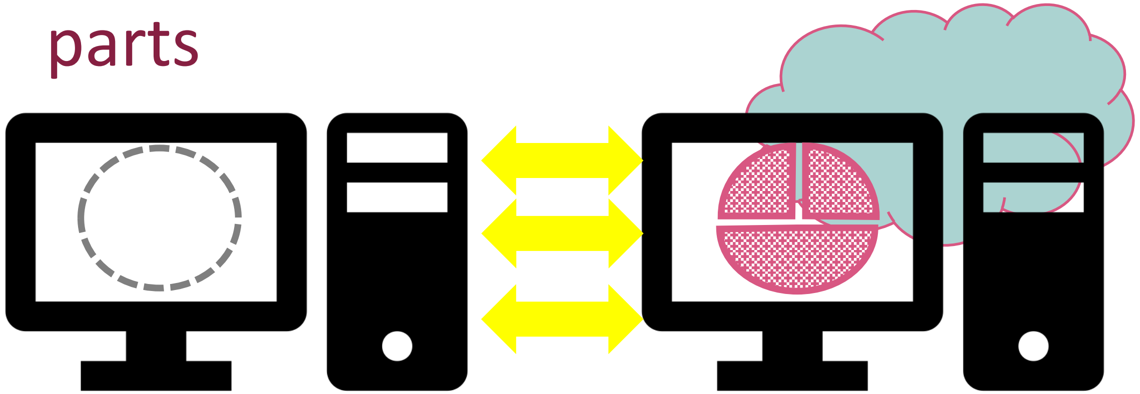
394 × 4139
~= 1.6 × 10^6
ULOCs

f_{CI}^{LOC}	f_{decl}	f_{sub}^{ind}	|D|
394	9	8	1.6M
394	9	8	1.6M
394	9	8	1.6M
	....		
38	5	2	76
49	4	3	245
379	3	3	1.5K

Latency

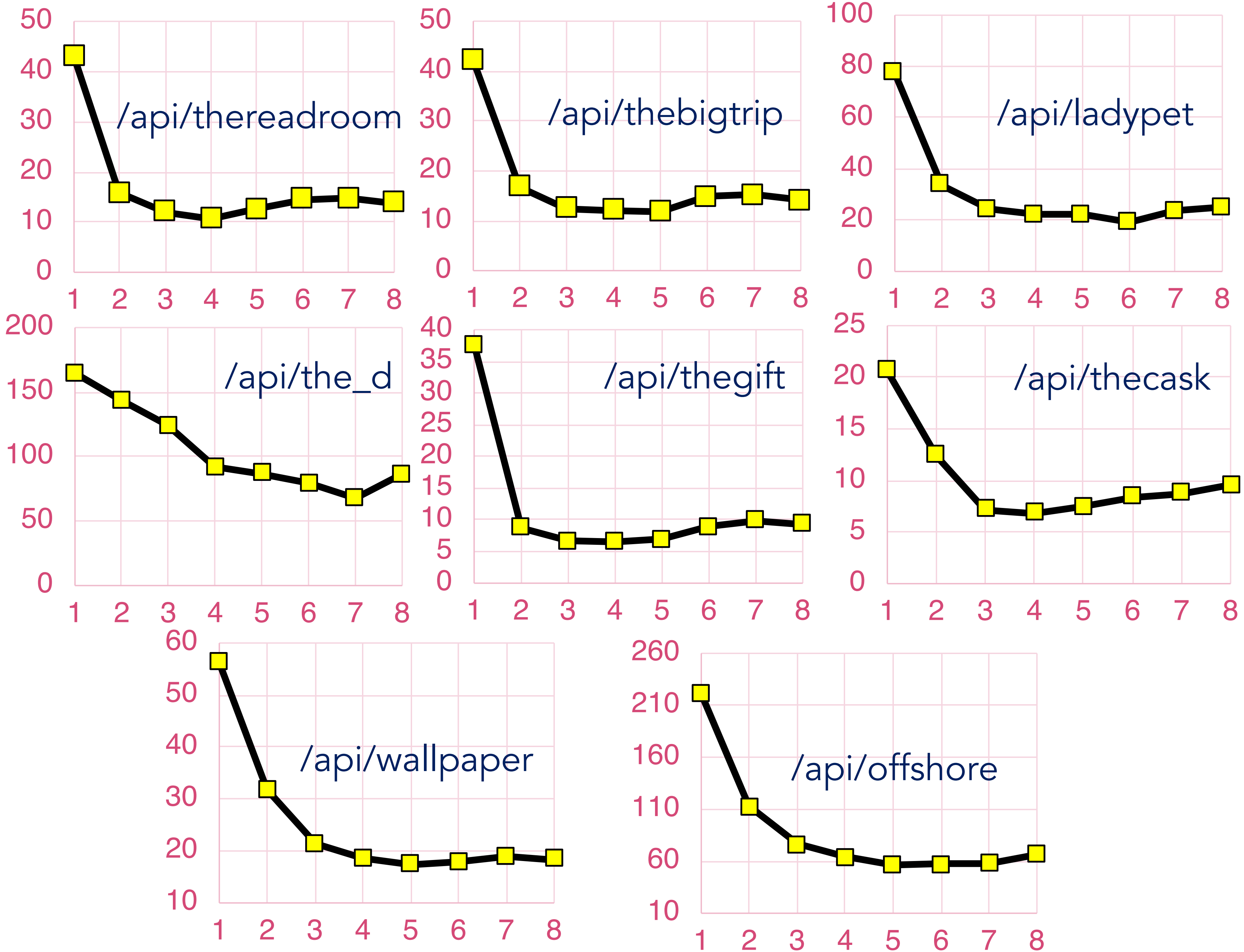


- **RQ2: Model Correctness(latency)**
- The larger the number of new remote functionalities, the smaller is the aggregate average latency ($latency(r) = \frac{1}{n}\sum_{i=1}^{n} T(r_i)$)
- Splitting a single long-running remote function into a small number of asynchronously invoked parts

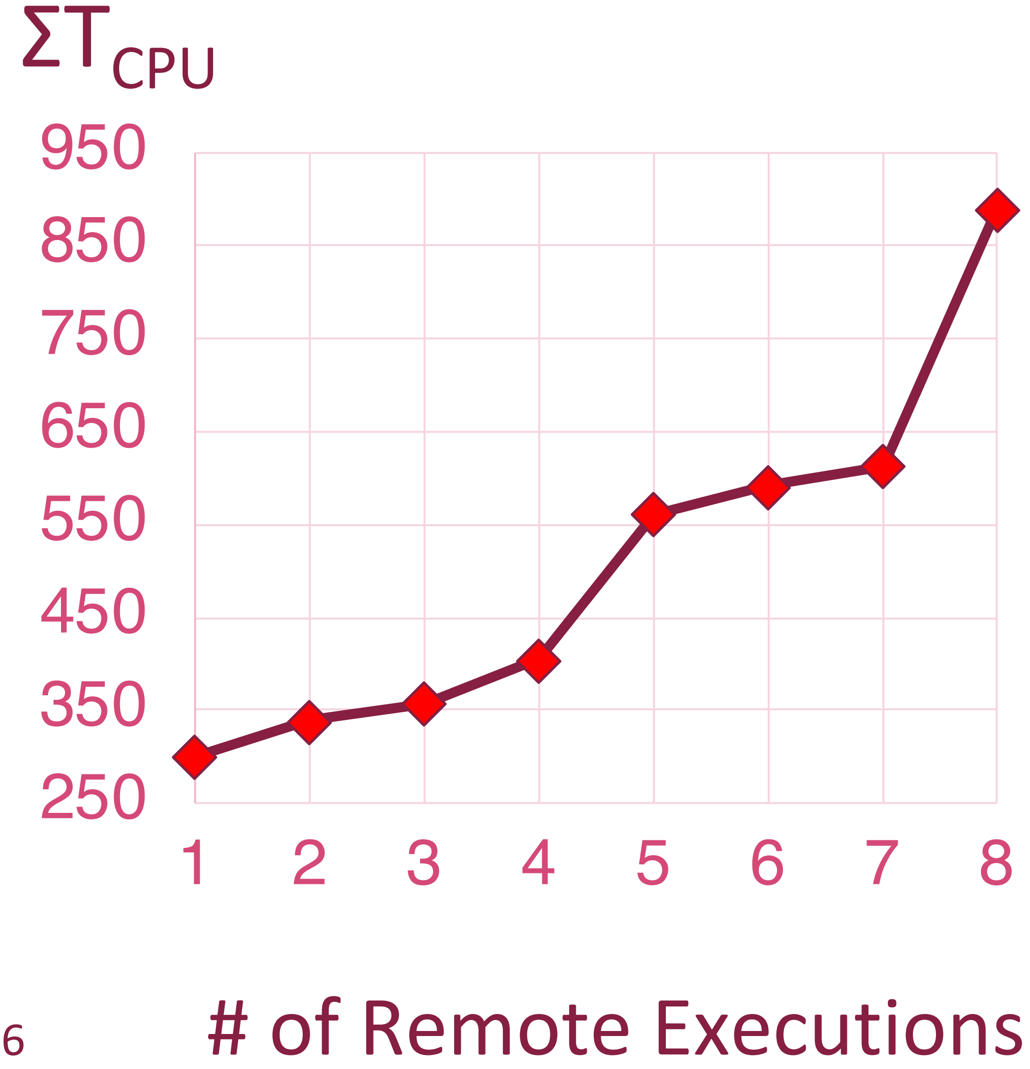


Asynchronously Invoking parts

Latency


/api/thereadroom
50
40
30
20
10
0
1 2 3 4 5 6 7 8
/api/thebigtrip
50
40
30
20
10
0
1 2 3 4 5 6 7 8
/api/ladypet
100
80
60
40
20
0
1 2 3 4 5 6 7 8
/api/the_d
200
150
100
50
0
1 2 3 4 5 6 7 8
/api/thegift
40
35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8
/api/thecask
25
20
15
10
5
0
1 2 3 4 5 6 7 8
/api/wallpaper
60
50
40
30
20
10
1 2 3 4 5 6 7 8
/api/offshore
260
210
160
110
60
10
1 2 3 4 5 6 7 8

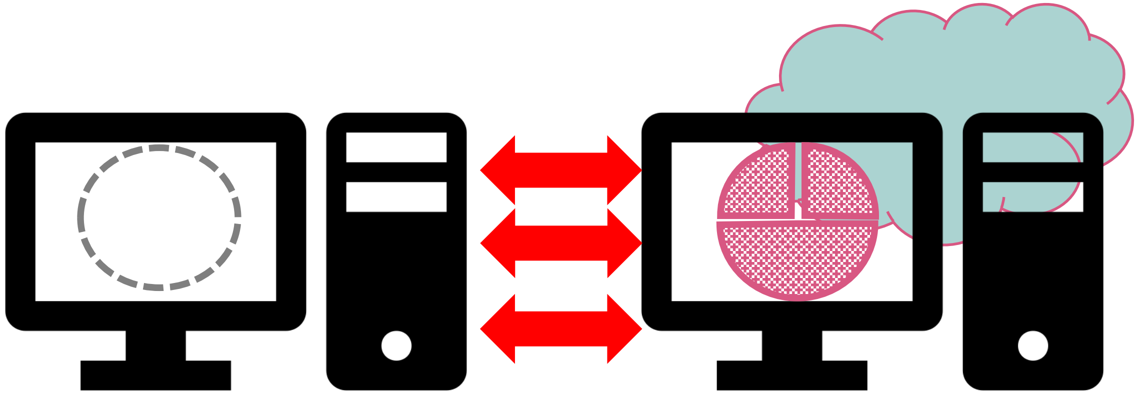
Resource(CPU Utilization)



RQ2: Model Correctness (Resources)

- We measured total CPU Utilization to invoke a remote service r: resource(r) =$\sum_{i=1}^{n} CPU(r_i)$

- **Consuming Client's Resource a lot to invoke multiple remote executions, propositionally to # of invocations**



Cost Function

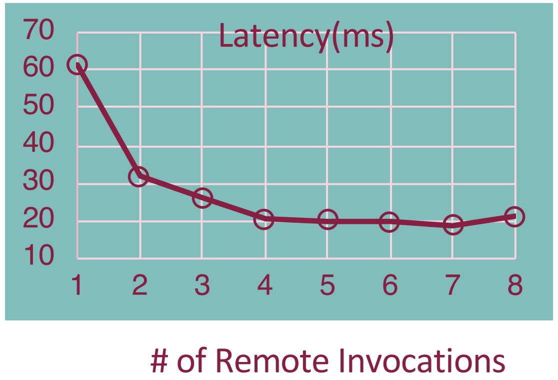
- Scaling Factor α
  - We empirically determined the required normalizing factor for the latency(milliseconds) and sum of CPU usages terms by scaling the observed latency/CPU usage ratios across all measurements $\alpha = \overline{L}/\overline{\Sigma T_{cpu}} = \mathbf{0.9281}$

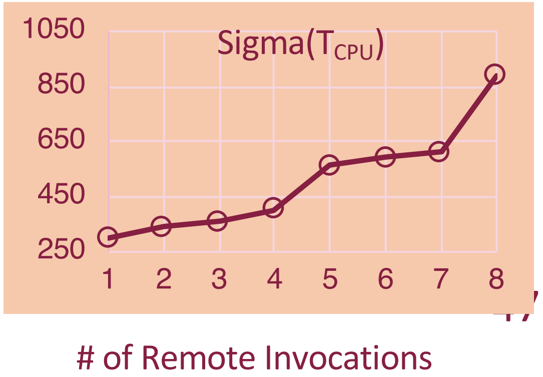
$$C_{Dist\_Exec}(\mathbf{r}) = \underline{\alpha}\cdot \mathrm{latency}(\mathbf{r}) + \underline{(1-\alpha)}\cdot \Sigma \mathrm{resource}(\mathbf{r})$$

Execution Time (Performance)

Consumed Resource (Efficiency)

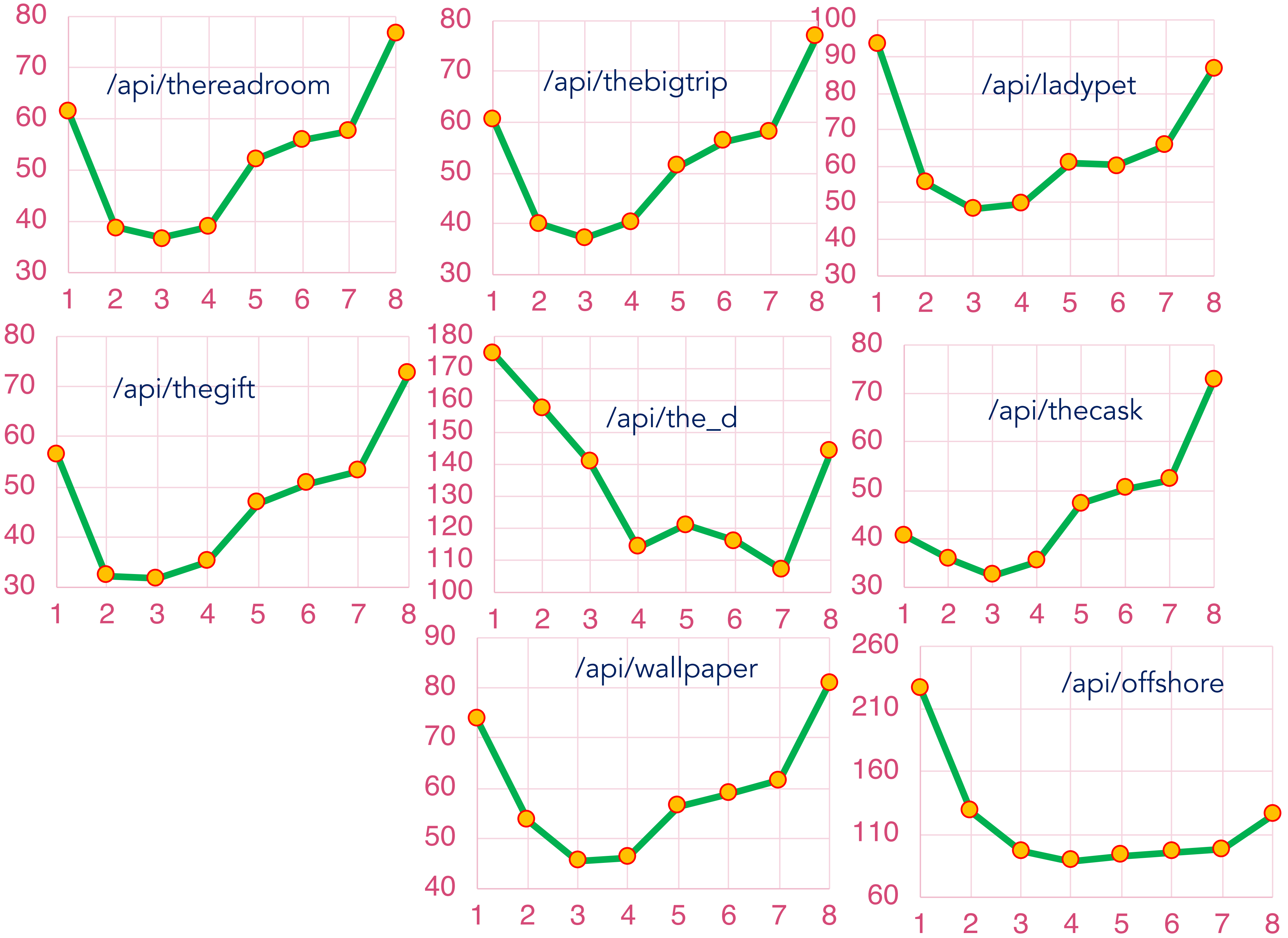
Normalizing factor





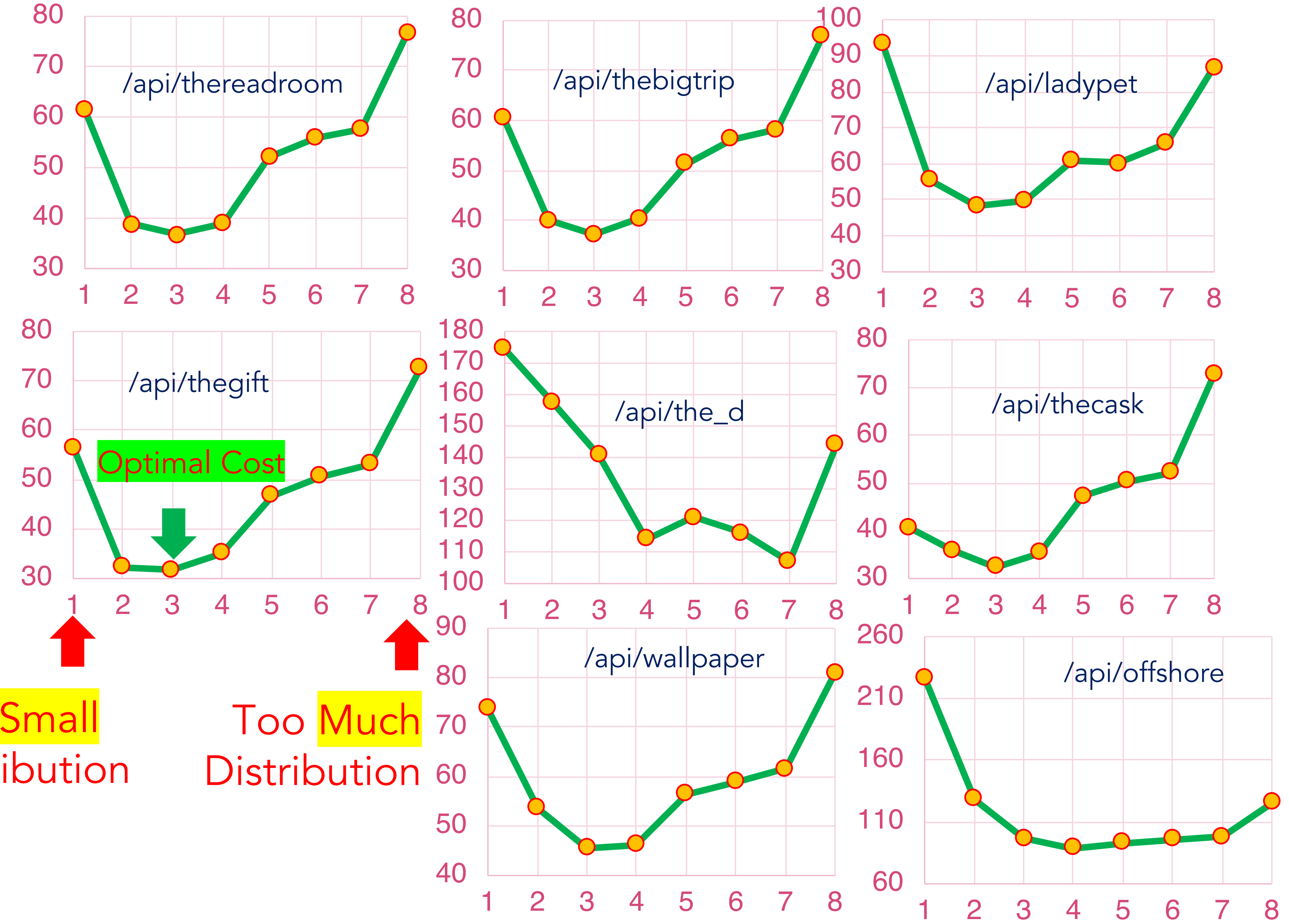
Cost Function

RQ3:—Utility of Cost Model



Cost Function

RQ3:—Utility of Cost Model



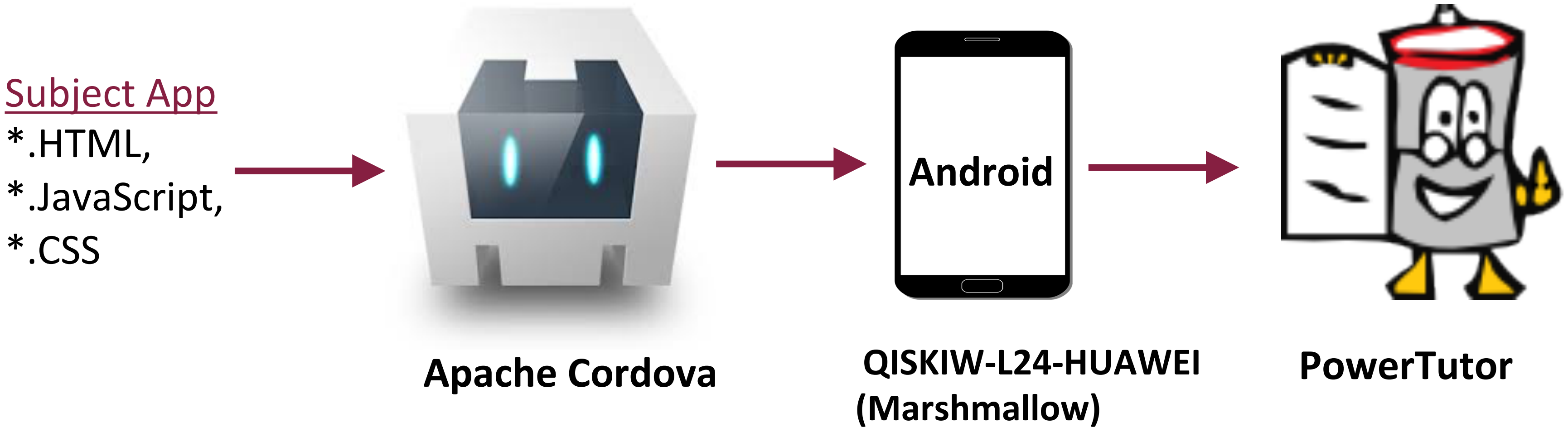
Optimal Cost

Too Small Distribution

Too Much Distribution

Energy Consumption

- **RQ4:** What is the effect of redistribution on the amount of energy consumed by the client?
  - We natively build the subject app **(BookWorm)** by using **Apache Cordova**
  - **PowerTutor** [L Zhang et.al]: a model-based energy profiler for mobile apps
  - **Energy Consumptions** (EC) for Original, Worse, and Best



Subject App
*.HTML,
*.JavaScript,
*.CSS

Apache Cordova

Android

QISKIW-L24-HUAWEI (Marshmallow)

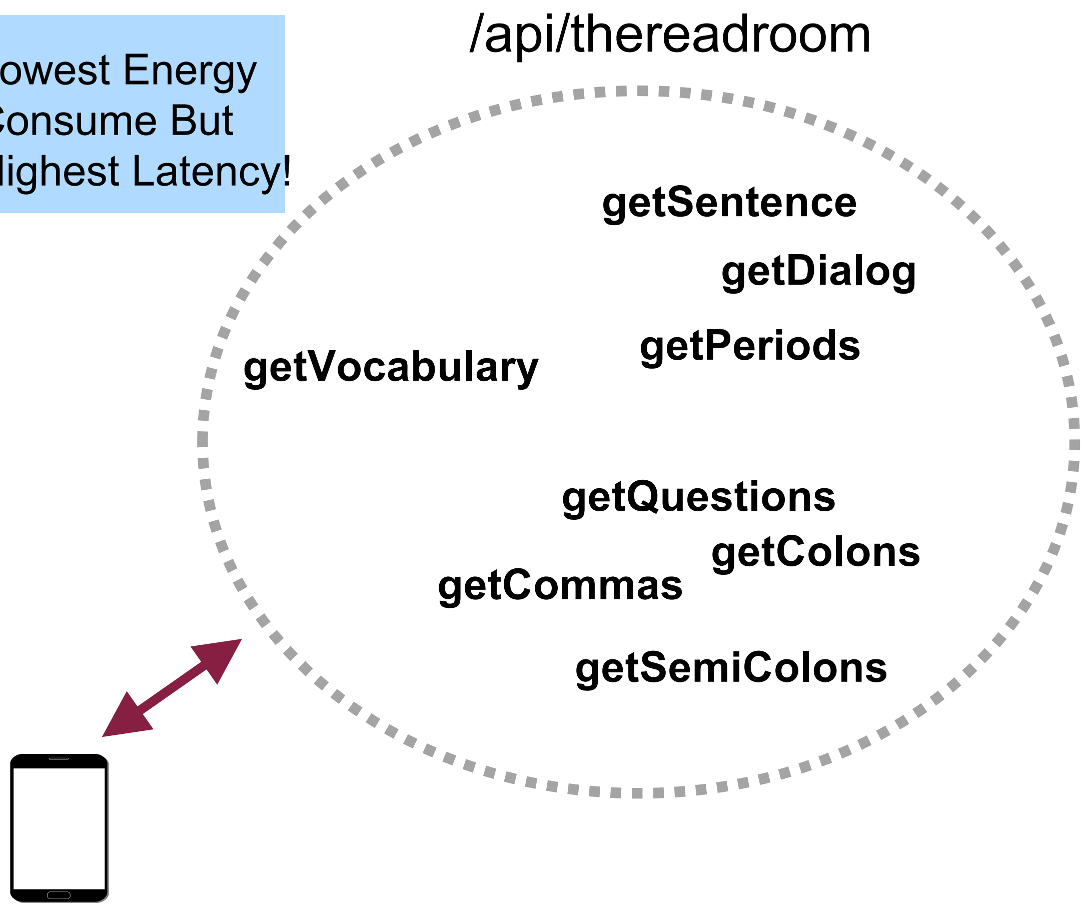
PowerTutor

Energy Consumption(Original)

- Cost Function Versus. Energy Consumption

$EC_{original\_dist}$	Original	8.4mJ
EC_{best_dist}	MIN Cost	13.4mJ
EC_{worst_dist}	MAX Cost	47.4mJ

Lowest Energy Consume But Highest Latency!

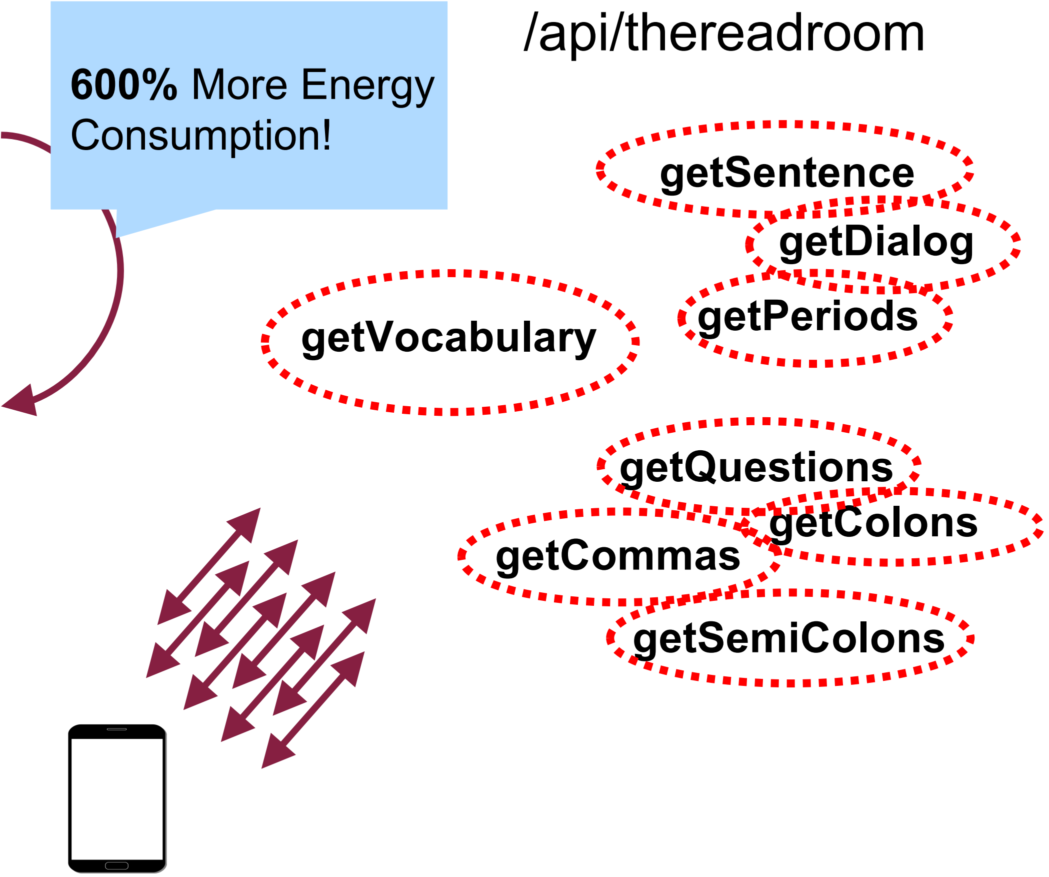


Energy Consumption(Worst)

- Cost Function Versus. Energy Consumption

$EC_{original\_dist}$	Original	8.4mJ
EC_{best_dist}	MIN Cost	13.4mJ
EC_{worst_dist}	MAX Cost	47.4mJ

600% More Energy Consumption!

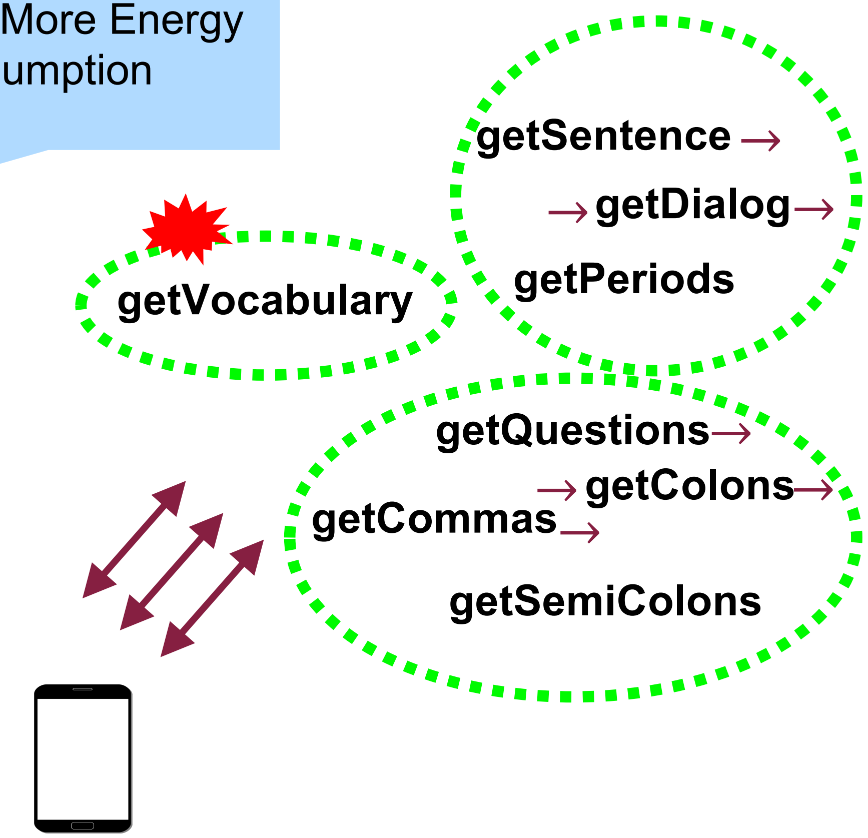


Energy Consumption(Best Dist. by D-Goldilocks)

- Cost Function Versus. Energy Consumption

$EC_{original\_dist}$	Original	8.4mJ
EC_{best_dist}	MIN Cost	13.4mJ
EC_{worst_dist}	MAX Cost	47.4mJ

50% More Energy Consumption



Conclusion

- A set of domain-specific **automatic refactorings** for **reshaping** and **redistribution**.
- A **cost function-based** heuristic for identifying how to improve the **performance** and **efficiency** of distributed apps by reshaping the original distribution, which was **too crude**.
- A systematic evaluation of our approach’s **value**, **utility**, and **efficiency** for our reference implementation **“D-Golilocks”**

Future Work

- Problem Formulation & Solution for different **Capacity** of **Clients** and **Servers** including **Network condition**

- Adaptation to **Edge Computing** for addressing their resource constraints and execution volatility

- Other types of Software **Evolution Scenarios**

Q n A

Thank you!

Appendix-1: Can multiple Executions reduce the aggregate average latency non-linearly?: $latency(r) = \frac{1}{n}\sum_{i=1}^{n} T(r_i)$

Executed **5** sub-tasks Sequentially for **1000ms**

GetSentence[1]→ getUniqueVocaburary[2]→ GetDialog[3]→getColon[4]→ getSemiColon[5]

Latency_1
= 1/5 (1000+1000+ 1000+1000+1000) = **1000 (ms)**

Executed a heavy processing function for **950ms**

getUnique Vocal

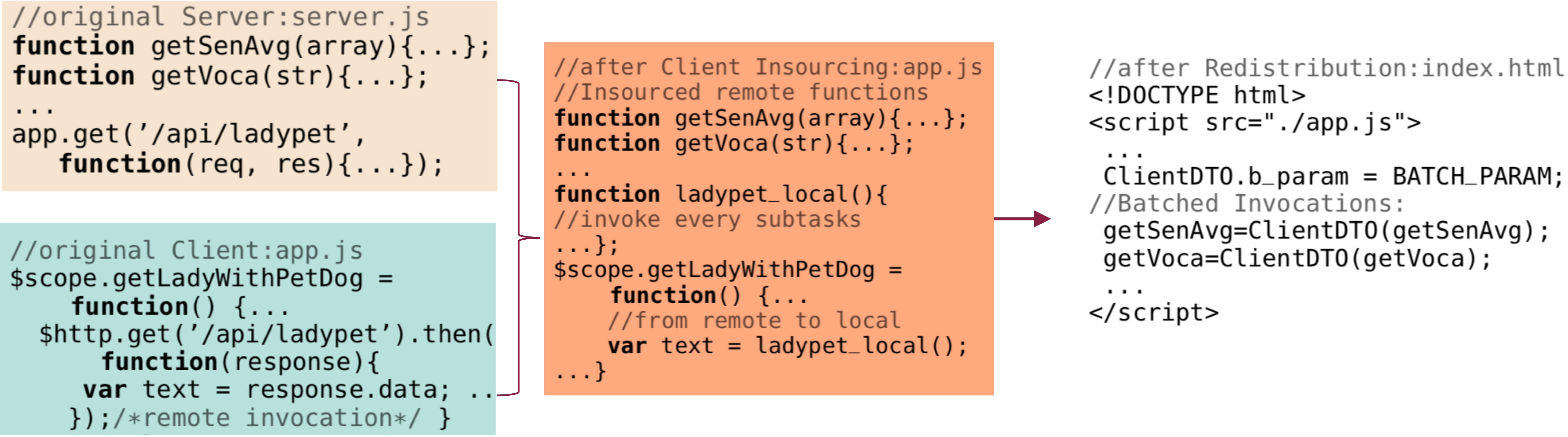
heavy processing function

Executed **4** sub-tasks Sequentially for **150ms**

getSentence[1]→ GetDialog[2]→getColon[3]→ getSemiColon[4]

Latency_2
= 1/5 (950+150+ 150+150+150) = **310 (ms)**

Appendix-2: Code-Base Example



```
//original Server:server.js
function getSenAvg(array){...};
function getVoca(str){...};
...
app.get('/api/ladypet',
    function(req, res){...});
```

```
//original Client:app.js
$scope.getLadyWithPetDog =
    function() {...
  $http.get('/api/ladypet').then(
      function(response){
     var text = response.data; ..
    });/*remote invocation*/ }
```

```
//after Client Insourcing:app.js
//Insourced remote functions
function getSenAvg(array){...};
function getVoca(str){...};
...
function ladypet_local(){
//invoke every subtasks
...};
$scope.getLadyWithPetDog =
    function() {...
    //from remote to local
    var text = ladypet_local();
...}
```

```
//after Redistribution:index.html
<!DOCTYPE html>
<script src="./app.js">
 ...
 ClientDTO.b_param = BATCH_PARAM;
//Batched Invocations:
 getSenAvg=ClientDTO(getSenAvg);
 getVoca=ClientDTO(getVoca);
 ...
</script>
```

Client
Insourcing

Batching fine-grained
service invocations