Imperial College London

Minimising Application Deployment Cost Using Spot Cloud Resources

Daniel J. Dubois

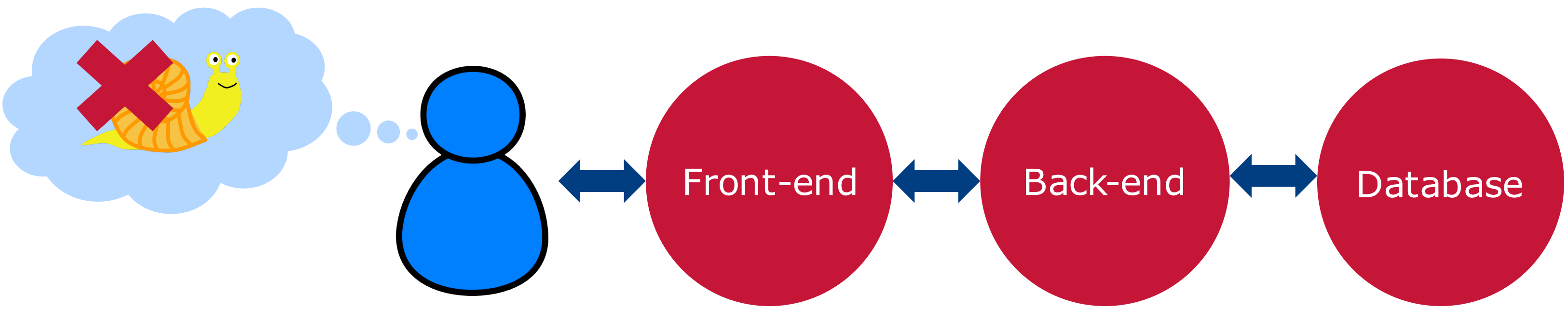
AESOP research group

daniel.dubois@imperial.ac.uk

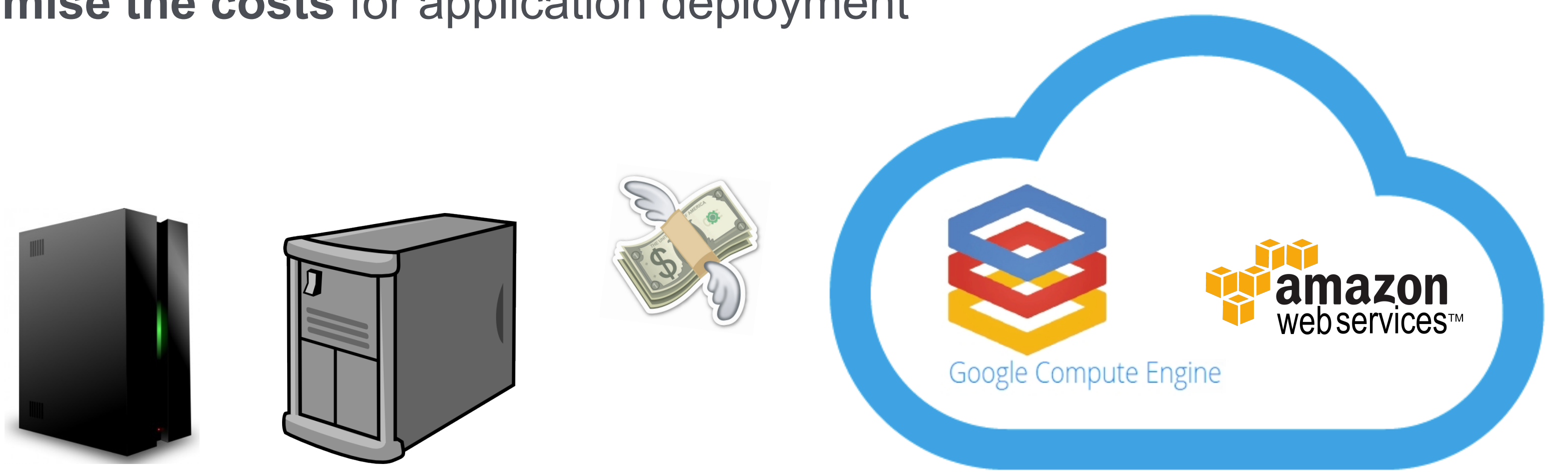
RA Symposium – DoC – Imperial College London – 14 June 2016

Context

Enterprise applications with **quality requirements**



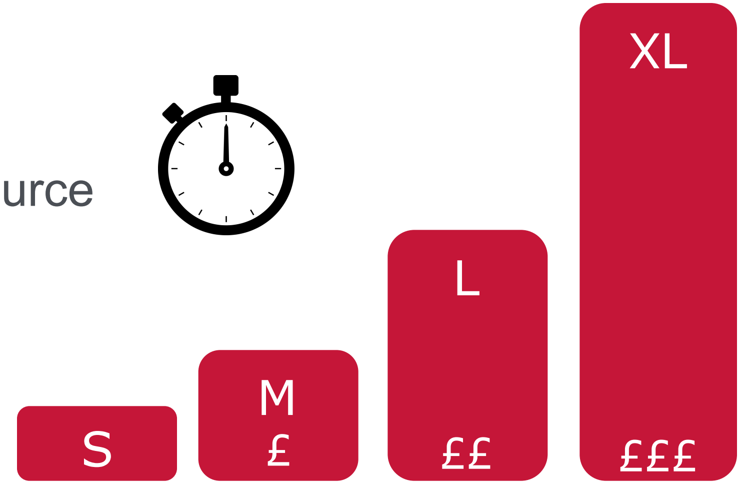
Minimise the costs for application deployment



How to save on costs? Cloud Computing!

Pay for the **time** you use a resource

Different **sizes** of resources



Different **providers**

Google Compute Engine

amazon web services™

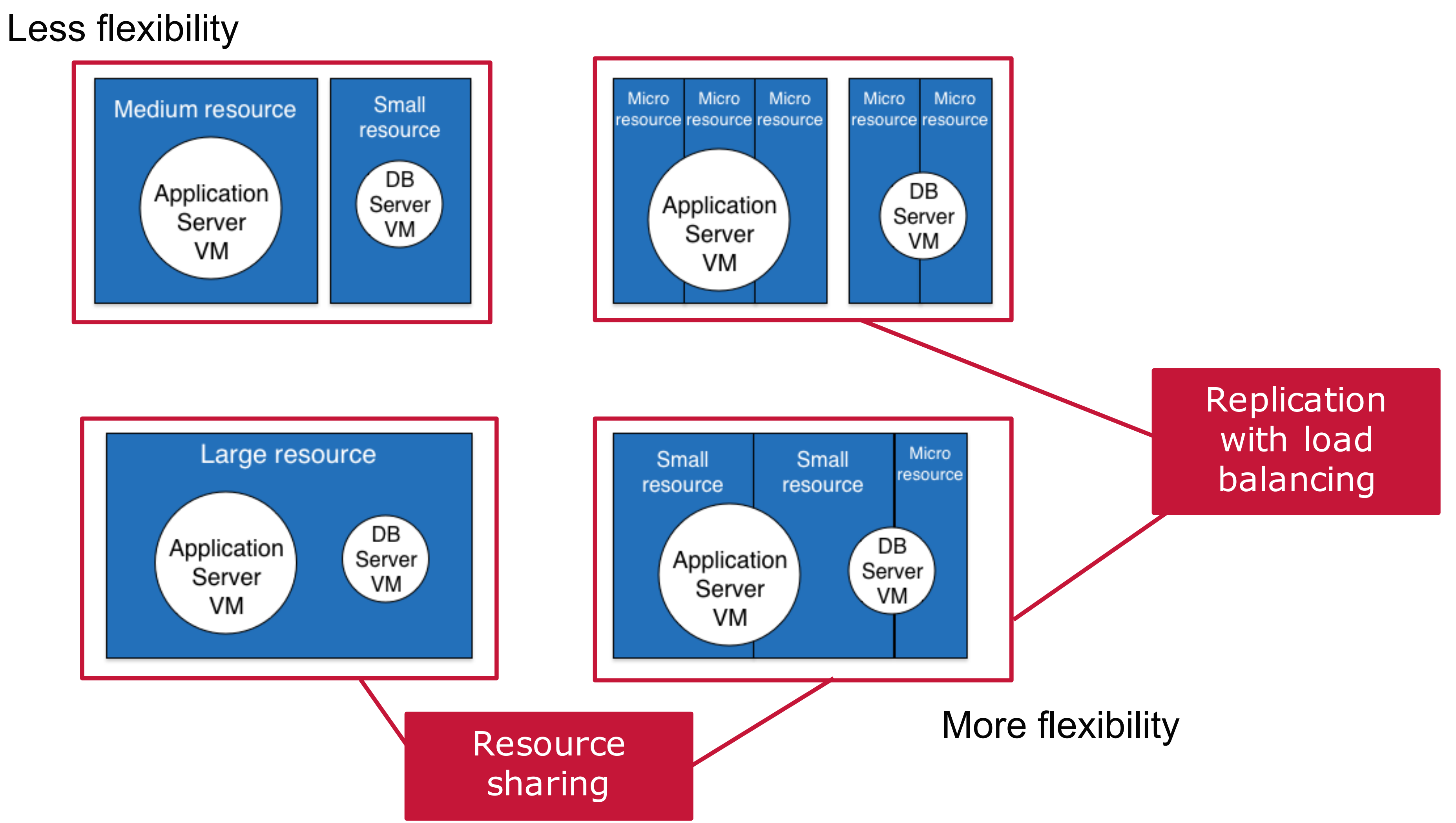
Microsoft Azure

Different **pricing** strategies: ON DEMAND vs SPOT

The decision space is huge!

Imperial College London

Deployment Possibilities

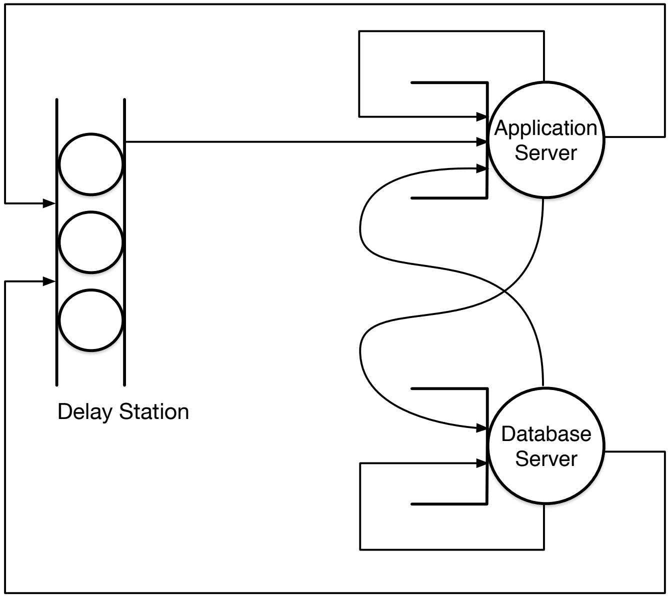


Imperial College London

Application Model

Closed Multi-class **Queueing Network:**

- Exponentially distributed service times



Application constraints:

- $maxMRT_k$: max. response time for each class
- $maxRTP_{k,u}$: max. response time in the u-th percentile

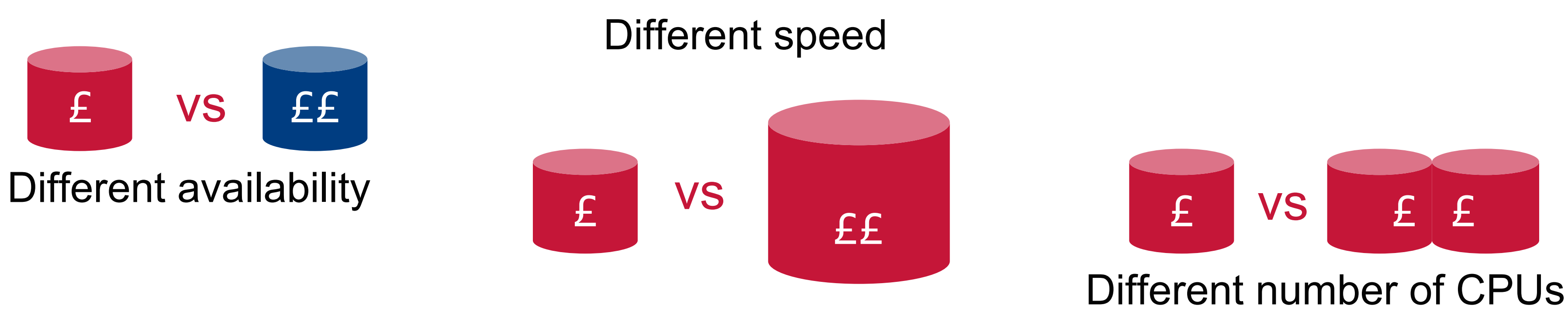
⇔ SLO

Service Level Objectives

Resource Model

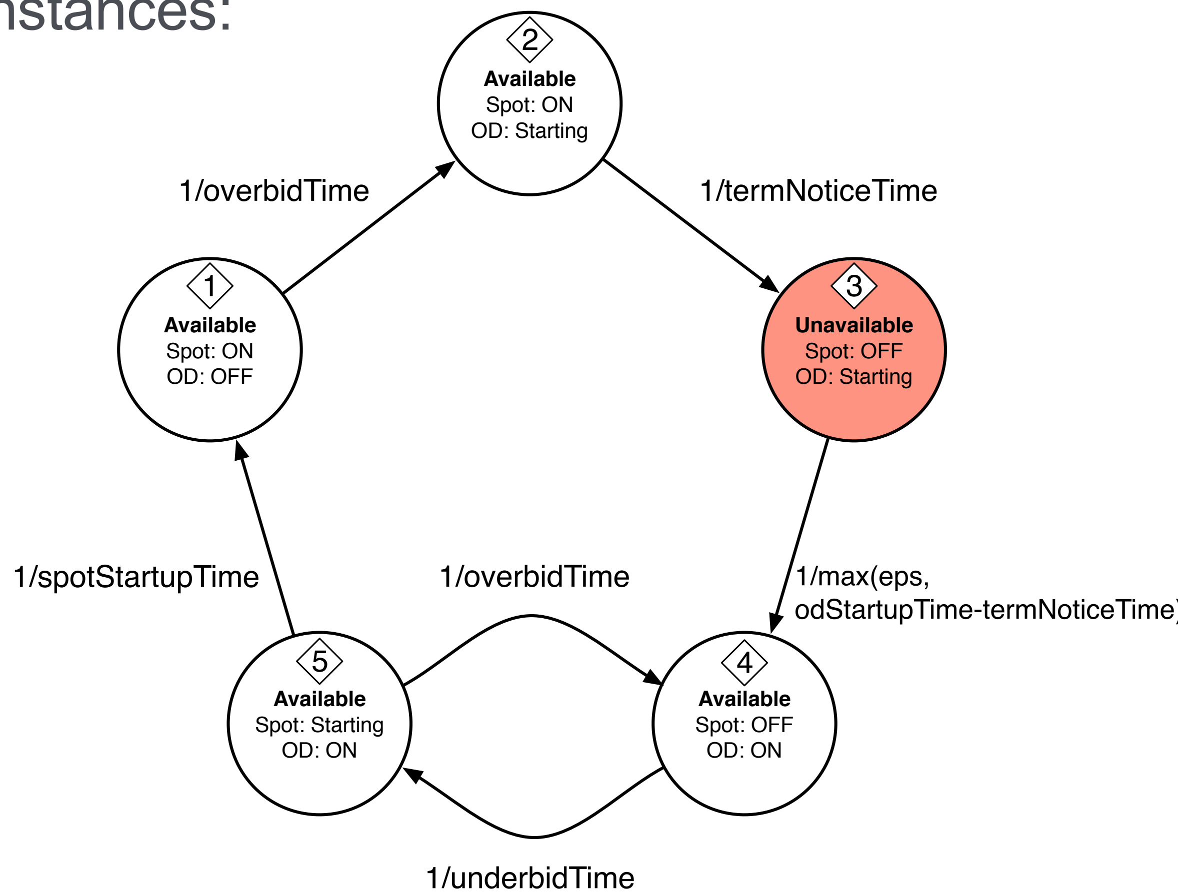
Parameters for **each** resource (case for Amazon spot resources)

- Speed of the resource (e.g., Amazon ECU)
- Number of processors
- On demand price
- Optimal bid price to obtain a certain level of availability
- Expected cost of the resource when bidding the bid price
- ...



Performance Prediction: Random Environment for Spot Instances

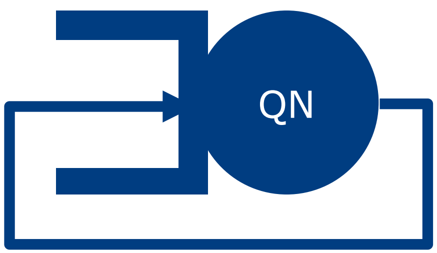
Use a Continuous-time Markov Chain to represent unreliable Spot instances:

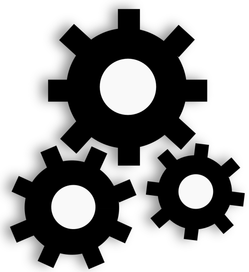


Performance Prediction: Evaluating the QN

We use this existing tool: LINE solver*

- Solves the QN using a fluid approximation
- Supports Random Environments



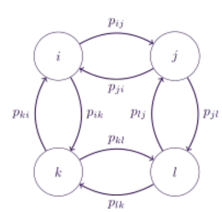


LINE Solver*

MRT_k: max. response time for each class

$RTP_{k,u}$: max. response time in the u-th percentile

Random Environment (Markov Chain of QN evolution)



* http://line-solver.sf.net

Decision Parameters: Resource Type Vector

Which resources to choose?

$t=[t_y]$ → resource type vector

Example: t_1=small, t_2=large

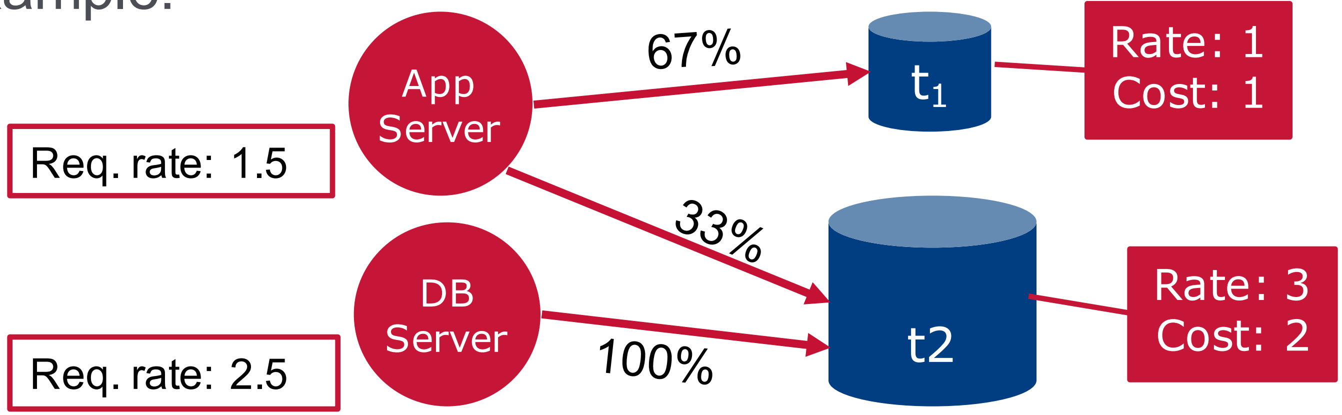
Total cost: 3

Decision Parameters: Allocation Matrix

How are application components allocated to resources?

$D=[d_{m,y}]$ → allocation matrix of component **m** to resource **y**

Example:



D=

	t_1	t_2
App Server	1	0.5
DB Server	0	2.5

Optimisation Problem

Goal:

- **Minimise the total cost**

$$\min \sum_{y=1,\ldots,Y} \hat{c}_y$$

Subject to:

- **Speed constraints of the chosen resources**
- **Service Level Objectives**

$$\sum_{m=1,\ldots,M} d_{m,y} \leq \hat{\lambda}_y, \forall y$$

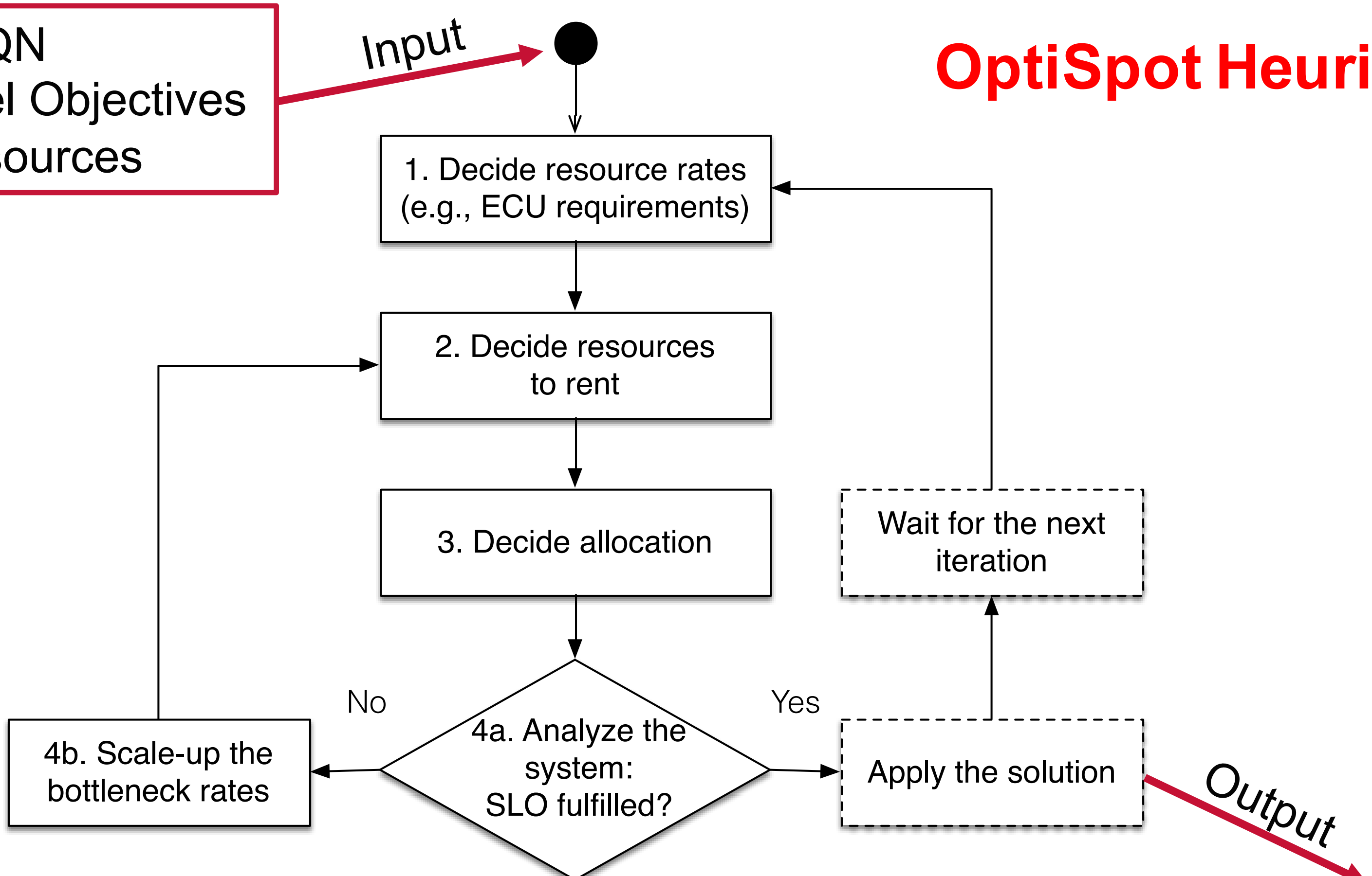
$$MRT_k(D) \leq maxMRT_k, \forall k$$

$$RTP_{u,k}(D) \leq maxRTP_{u,k}, \forall u, \forall k$$

Adaptation: iterative process

Application QN
Service Level Objectives
Available resources

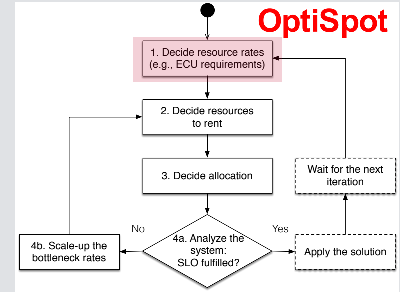
OptiSpot Heuristic



Cloud resources to instantiate (**t vector**)
Allocation of the application components to the resources (**D matrix**)

Imperial College London

Step 1: Deciding Resource Rates



Input

Application Model (Queueing Network)

Constraints on the response time

Output

Service rates for each resource fulfilling the constraints

Problem (NLP)

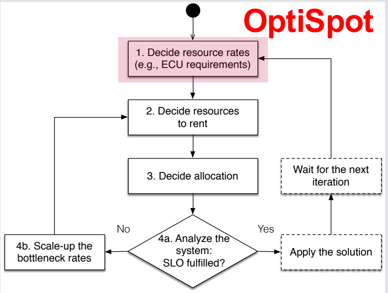
Minimise the rate of each resource

Subject to: **Service Level Objectives**

$$\min \sum_{m=1,\dots,M} \hat{\mu}_m$$

$$\text{s.t.} \quad MRT_{k,}(\hat{\mu}) \leq maxMRT_k, \forall k$$

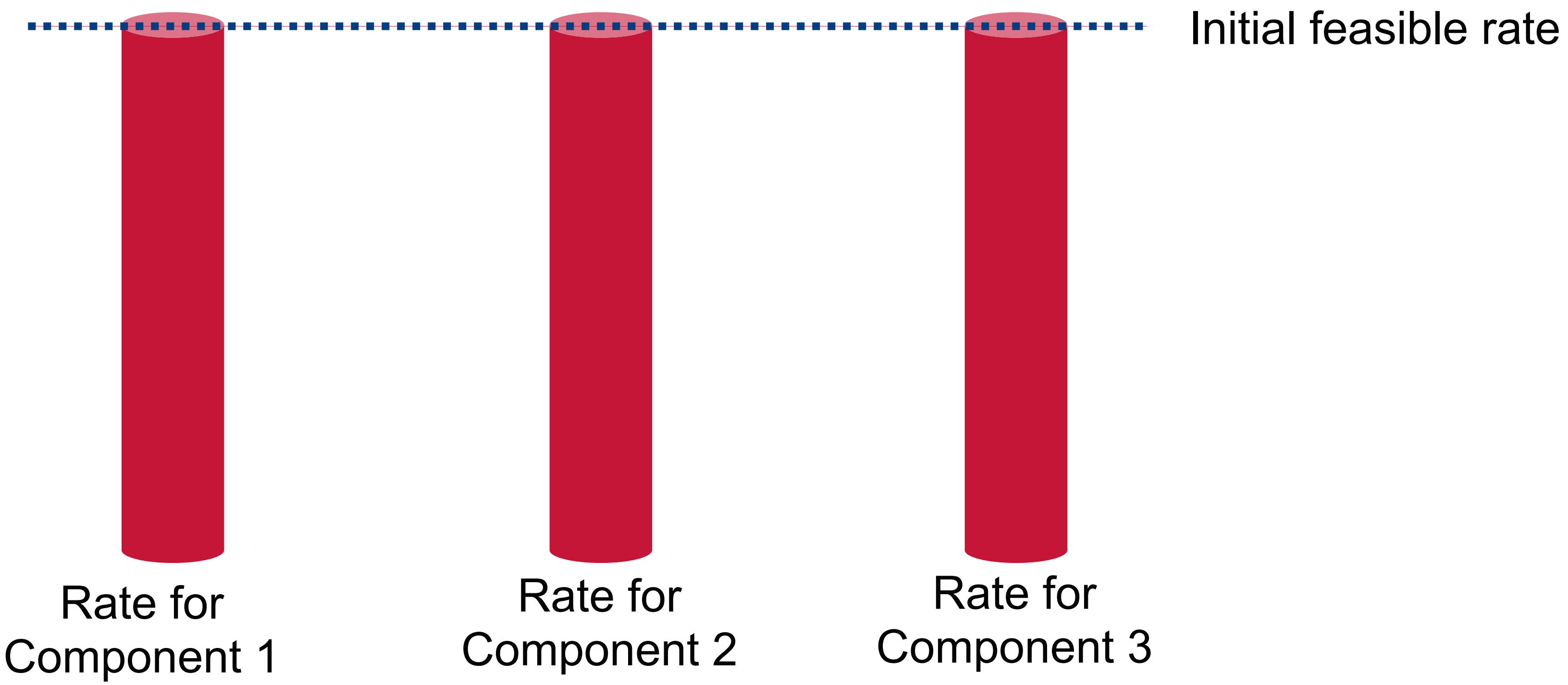
$$RTP_{u,k}(\hat{\mu}) \leq maxRTP_{u,k}, \forall u, \forall k$$



Step 1: Deciding Resource Rates

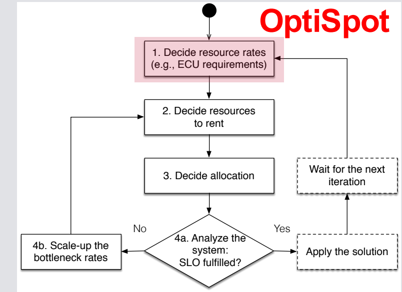
Idea:

- Start with high feasible resource rates
- Scale-down the rates of all components until reaching the SLO boundary



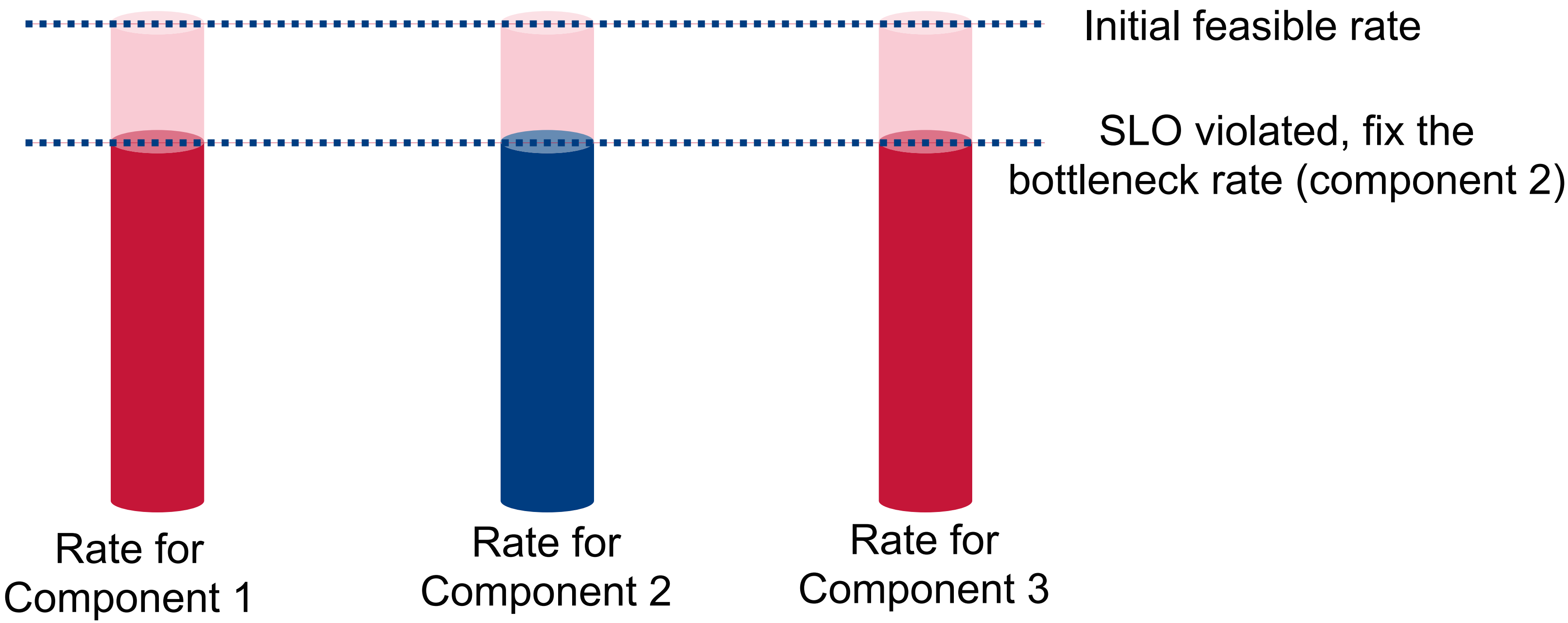
Imperial College London

Step 1: Deciding Resource Rates

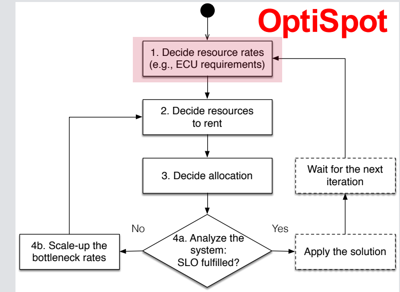


Idea:

- Start with high feasible resource rates
- Scale-down the rates of all components until reaching the SLO boundary



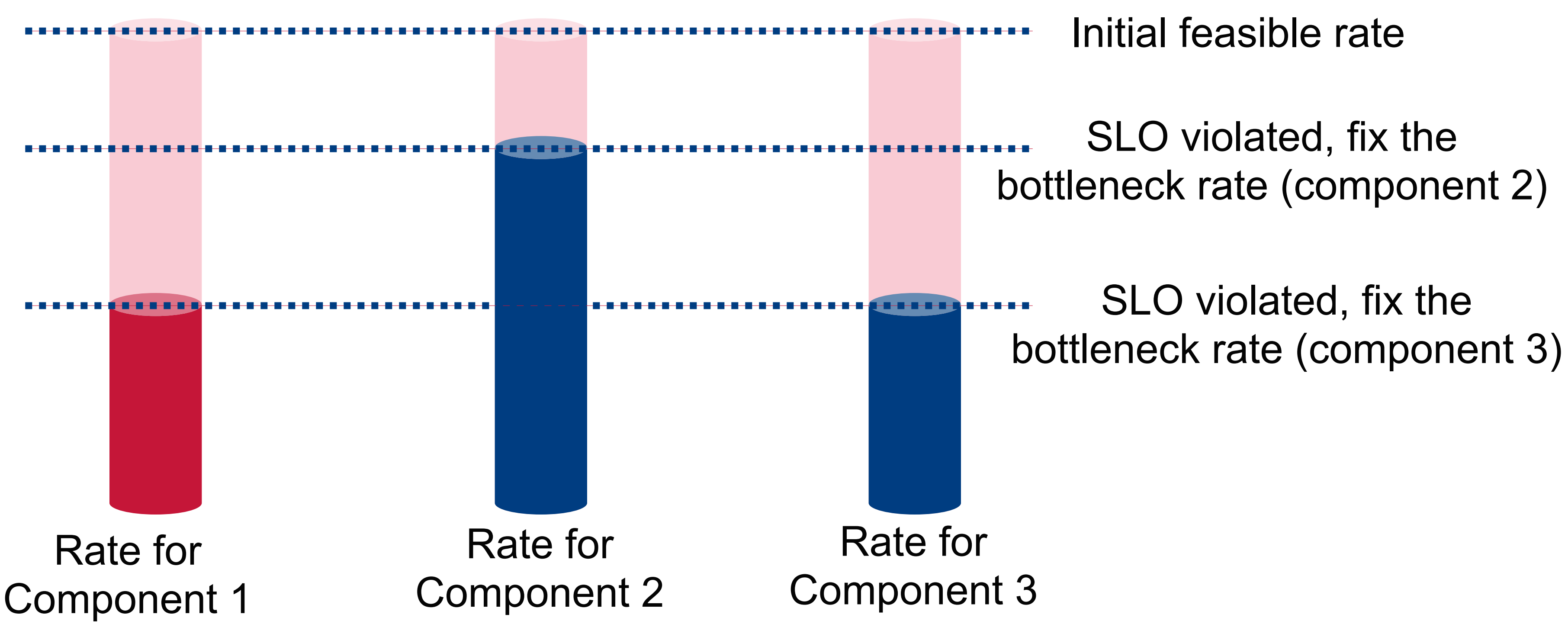
Imperial College London



Step 1: Deciding Resource Rates

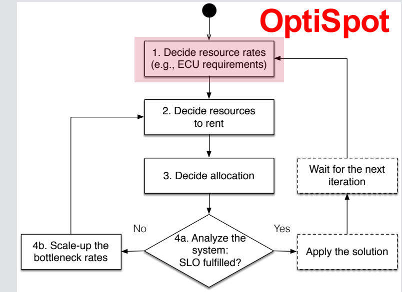
Idea:

- Start with high feasible resource rates
- Scale-down the rates of all components until reaching the SLO boundary



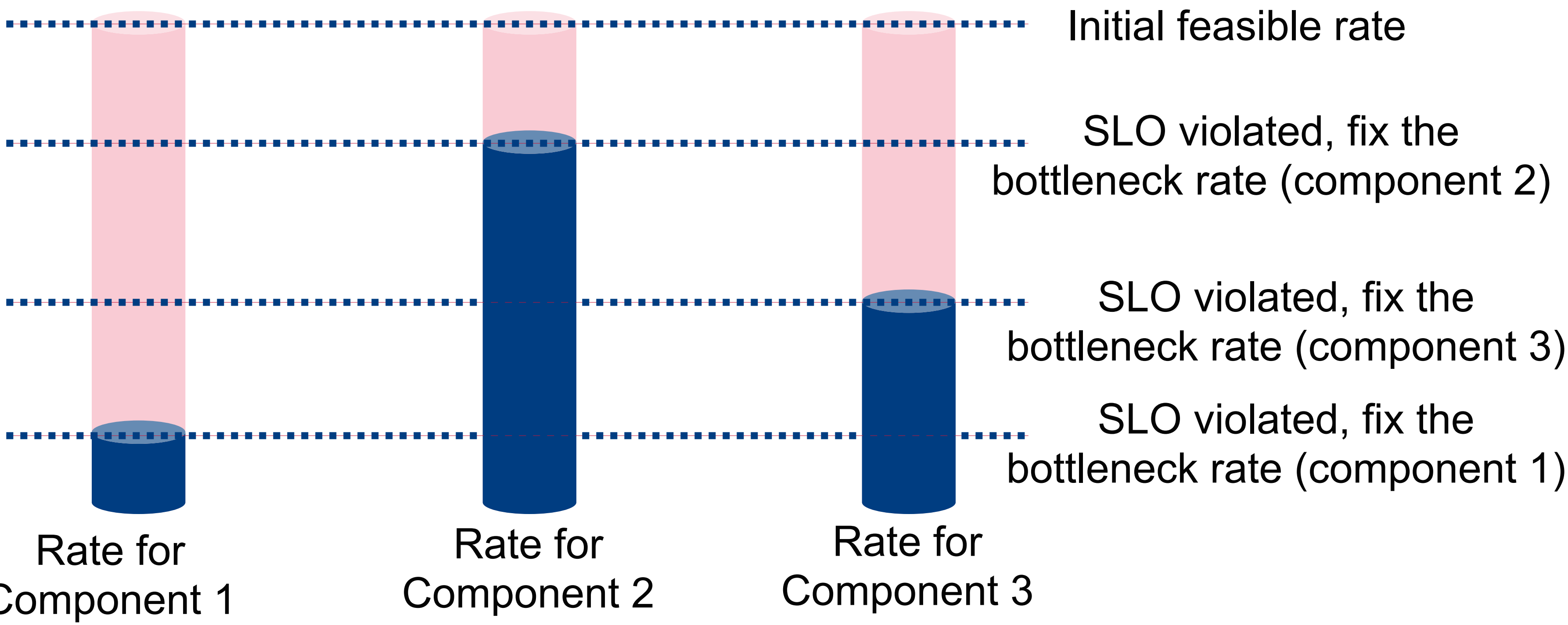
Imperial College London

Step 1: Deciding Resource Rates



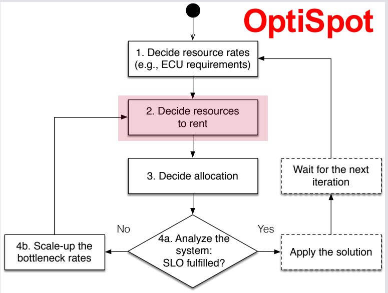
Idea:

- Start with high feasible resource rates
- Scale-down the rates of all components until reaching the SLO boundary



Imperial College London

Step 2: Deciding the Resources to Rent



Find the **cheapest** way to provide the required resource rates

Input

- Sum of the resource rates found at STEP 1
- Available cloud resources (characteristics and costs)

Output

- List of resources to be rented (**t vector**)

Problem (ILP)

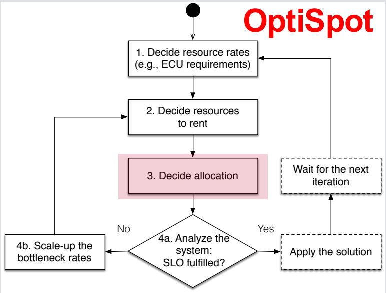
- **Minimise the cost of rented resources**
- Subject to:
  fulfilment of component service rate requirement

$$\min \sum_{y=1,\dots,Y} \hat{c}_y$$

$$\text{s.t.} \sum_{y\in 1,\dots,Y} \hat{\lambda}_y \geq \sum_{m\in 1,\dots,M} \hat{\mu}_m$$

Imperial College London

Step 3: Deciding Resource Allocation



Map application components to the resources to rent such that:

- Number of partitioned components is minimised
- Each component has enough resources allocated

Input

- Component rates (output of STEP 1)
- Rented resources (**t vector**, output of STEP 2)

Output

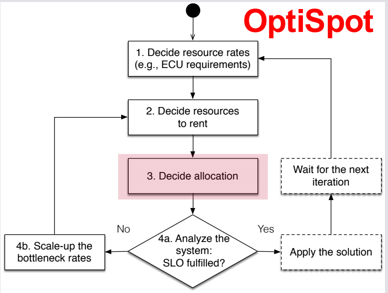
- Allocation matrix **D**

Idea

- Assign the component with the largest non-allocated rate to the resource with the largest unused rate

Imperial College London

Step 3: Deciding Resource Allocation



Component 1 (required rate: 10)

Resource 1 (available rate: 6)

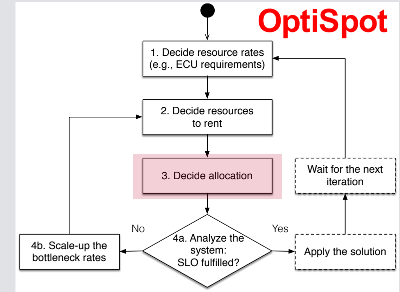
Resource 2 (available rate: 6)

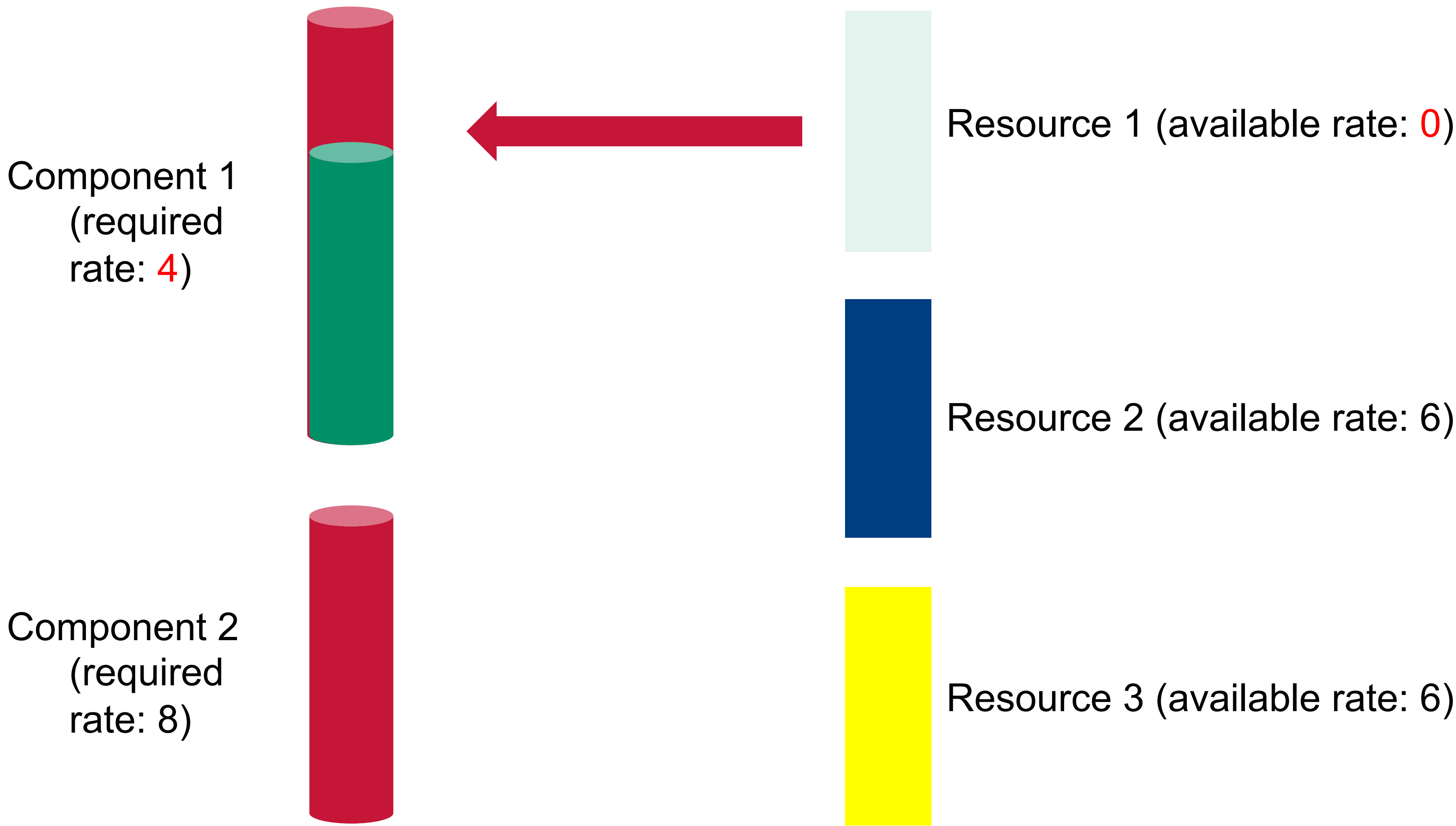
Component 2 (required rate: 8)

Resource 3 (available rate: 6)

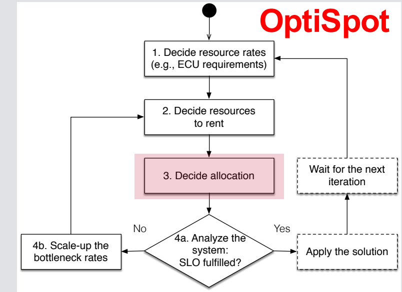
Imperial College London

Step 3: Deciding Resource Allocation

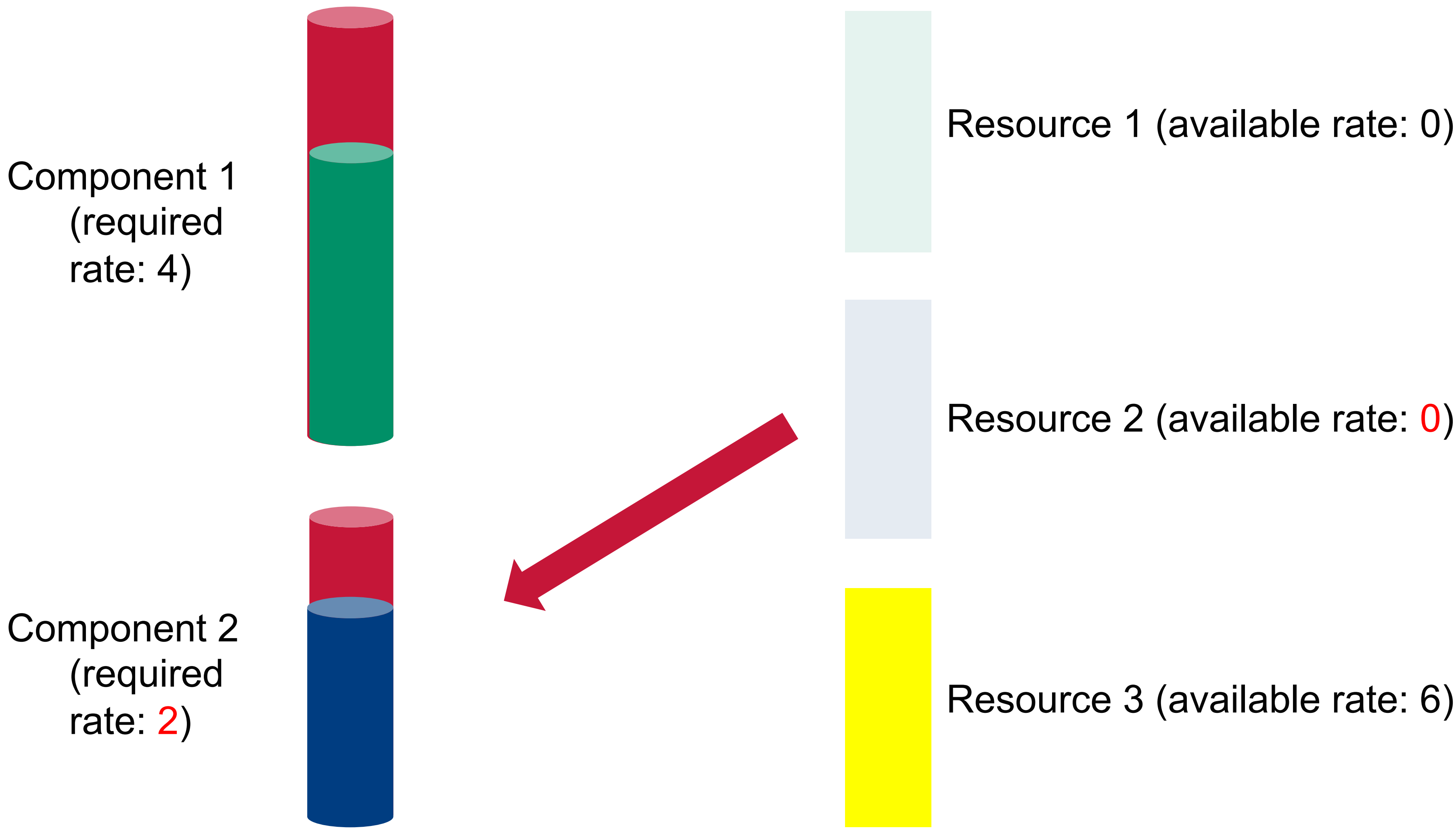




Imperial College London

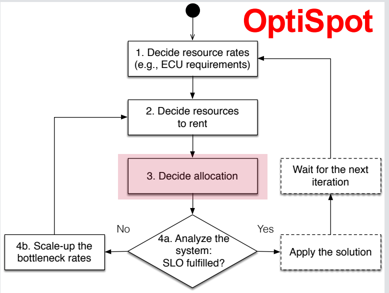


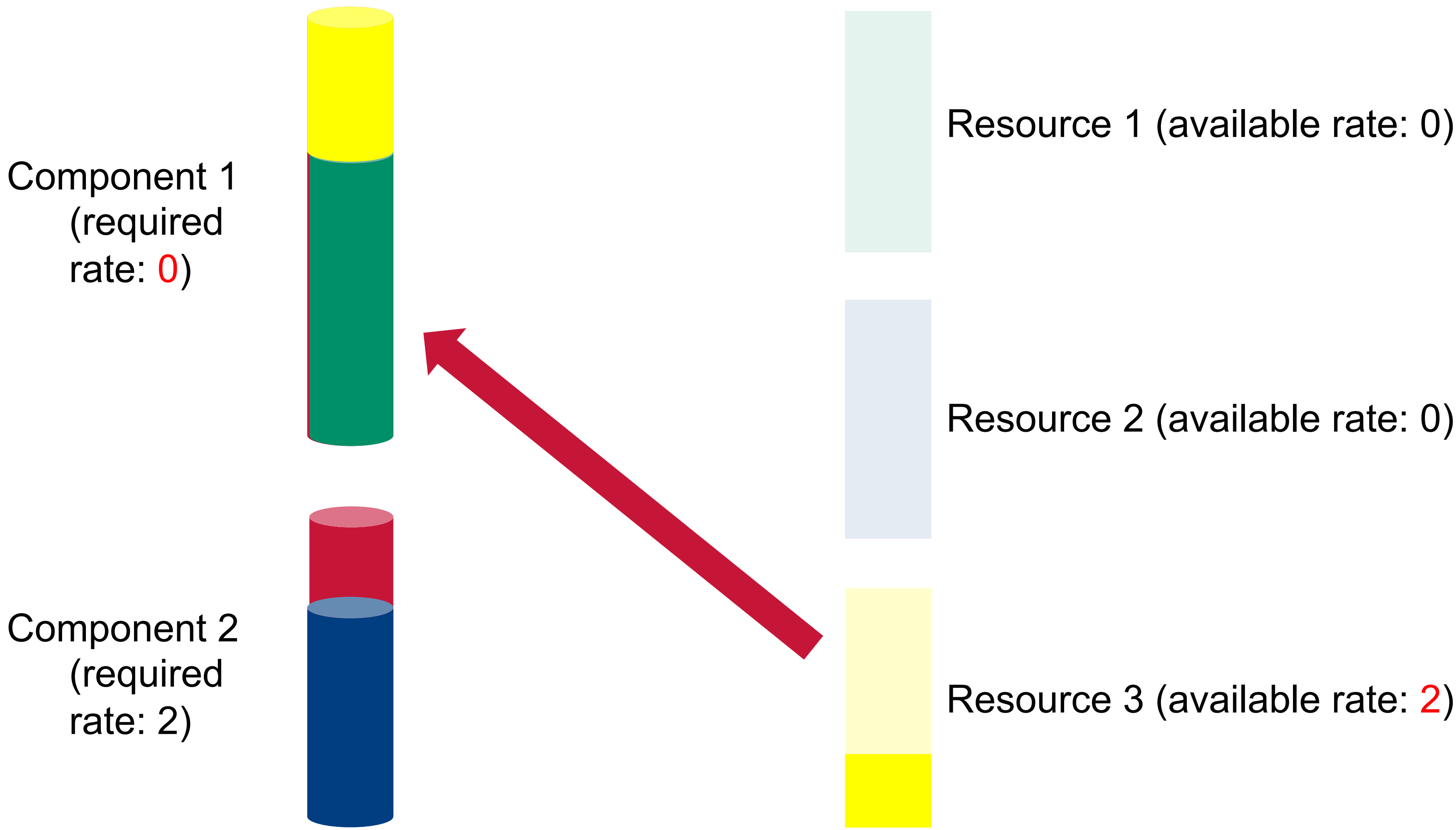
Step 3: Deciding Resource Allocation



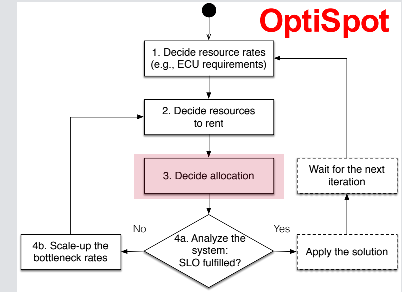
Imperial College London

Step 3: Deciding Resource Allocation

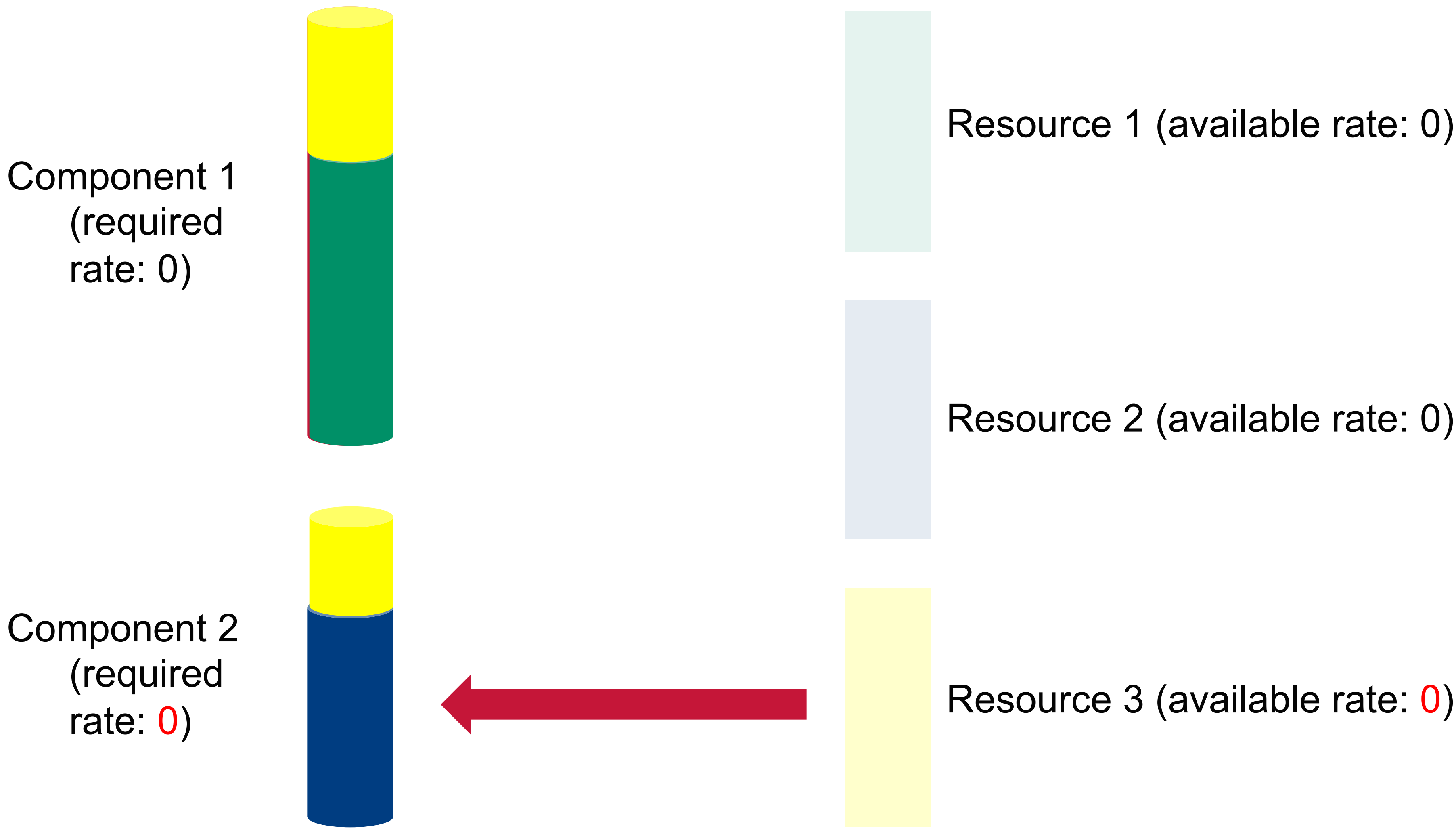




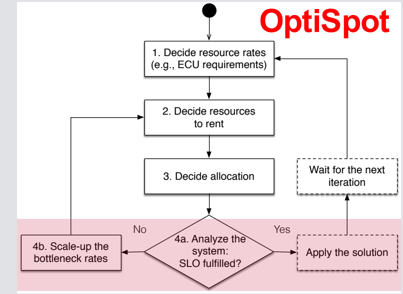
Imperial College London



Step 3: Deciding Resource Allocation



Step 4: Check the solution

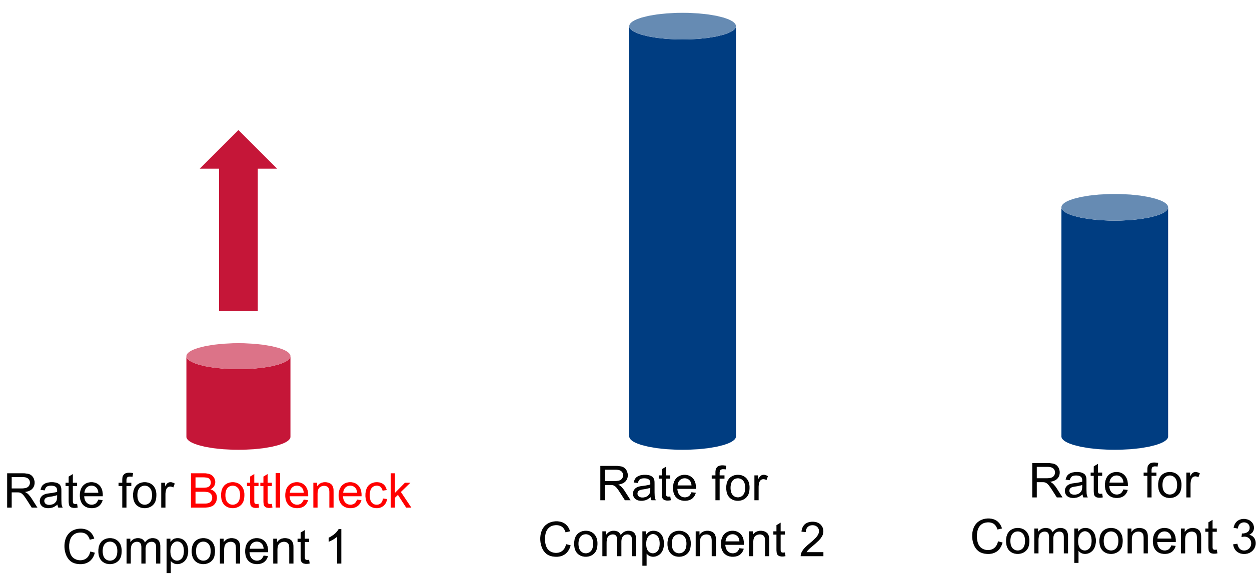


Is the SLO fulfilled?

YES: return the solution found

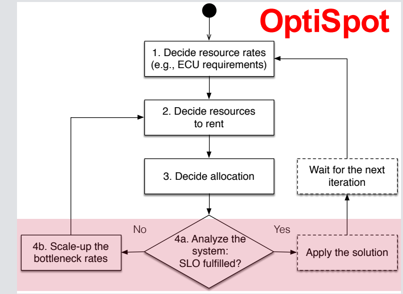
NO:

- Identify the **bottleneck** (components that causes that major SLO violation)
- scale-up the bottleneck component and restart from STEP 2



Imperial College London

Step 4: Check the solution

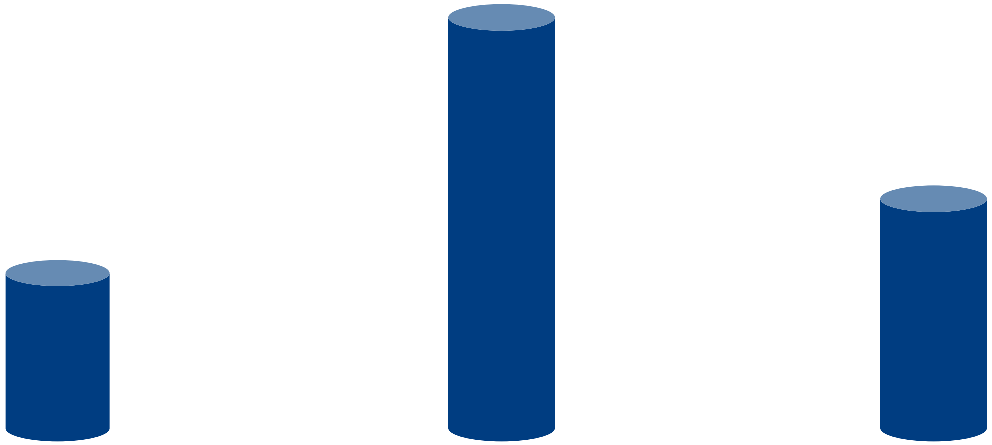


Is the SLO fulfilled?

YES: return the solution found

NO:

- Identify the **bottleneck** (components that causes that major SLO violation)
- scale-up the bottleneck component and restart from STEP 2

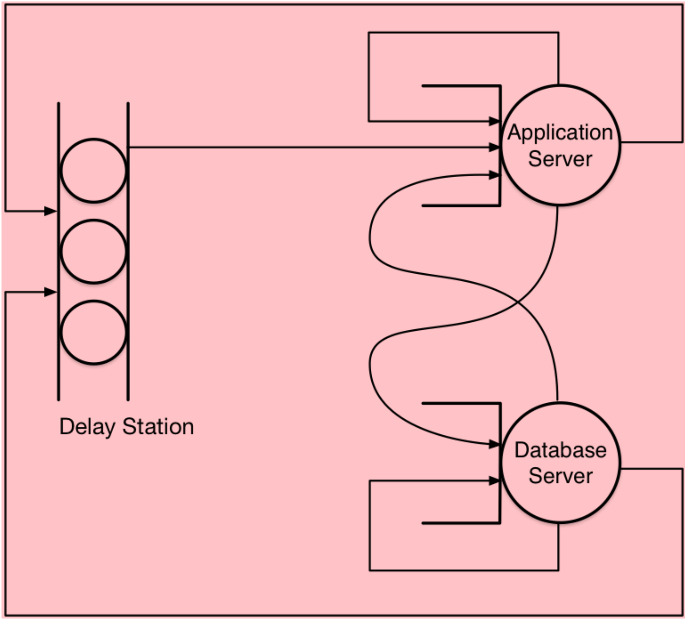


Case study 1: SAP ERP Application

Real application with real measurements

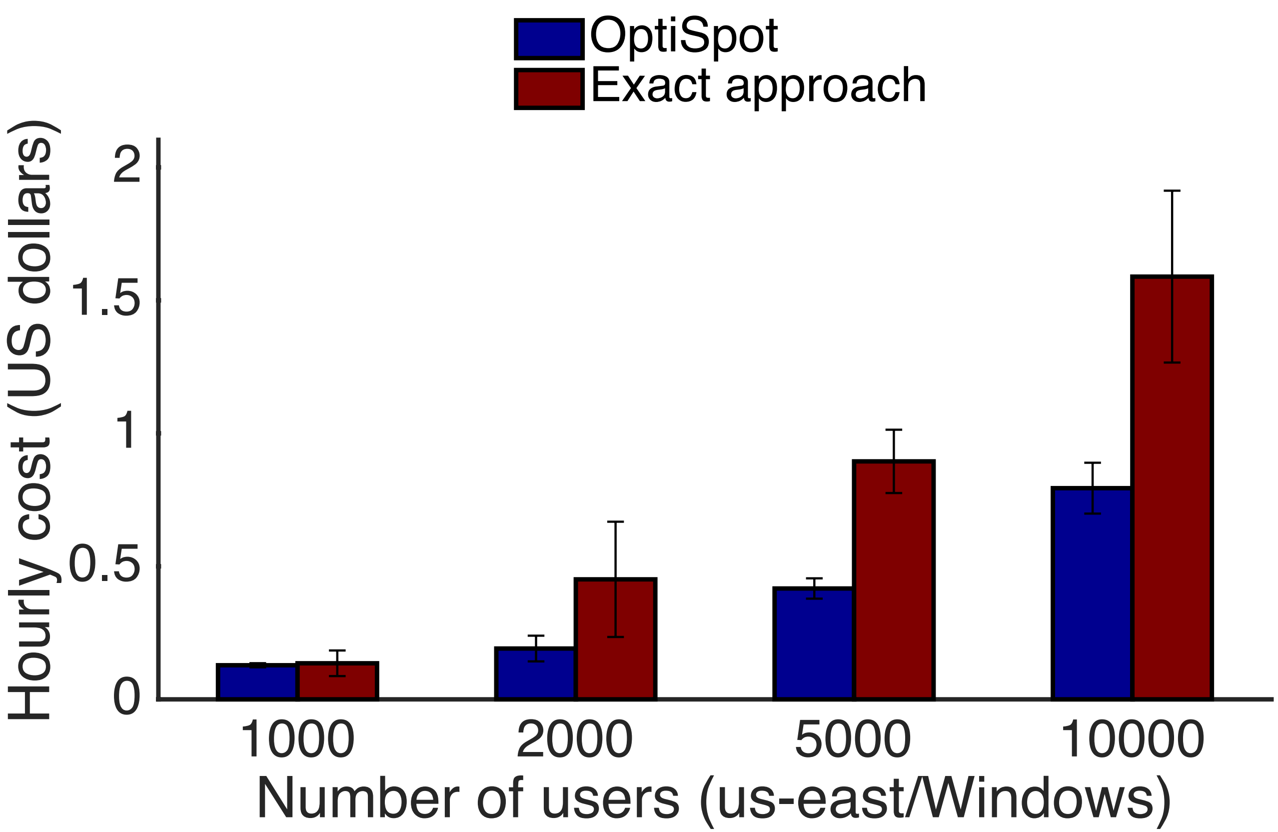
1 Application Server
1 DB Server

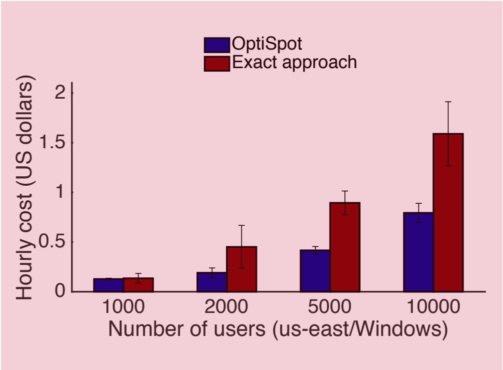
QN description publicly available

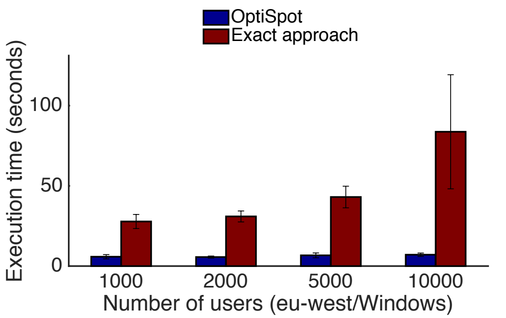


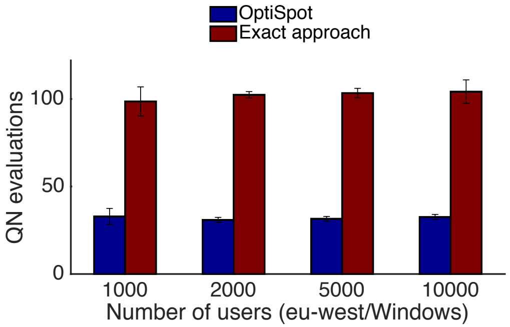
Compare our approach VS an exact approach based on a generic non-linear solver provided by MATLAB (fmincon using interior-point method)

Varying the Number of Users (hourly cost)



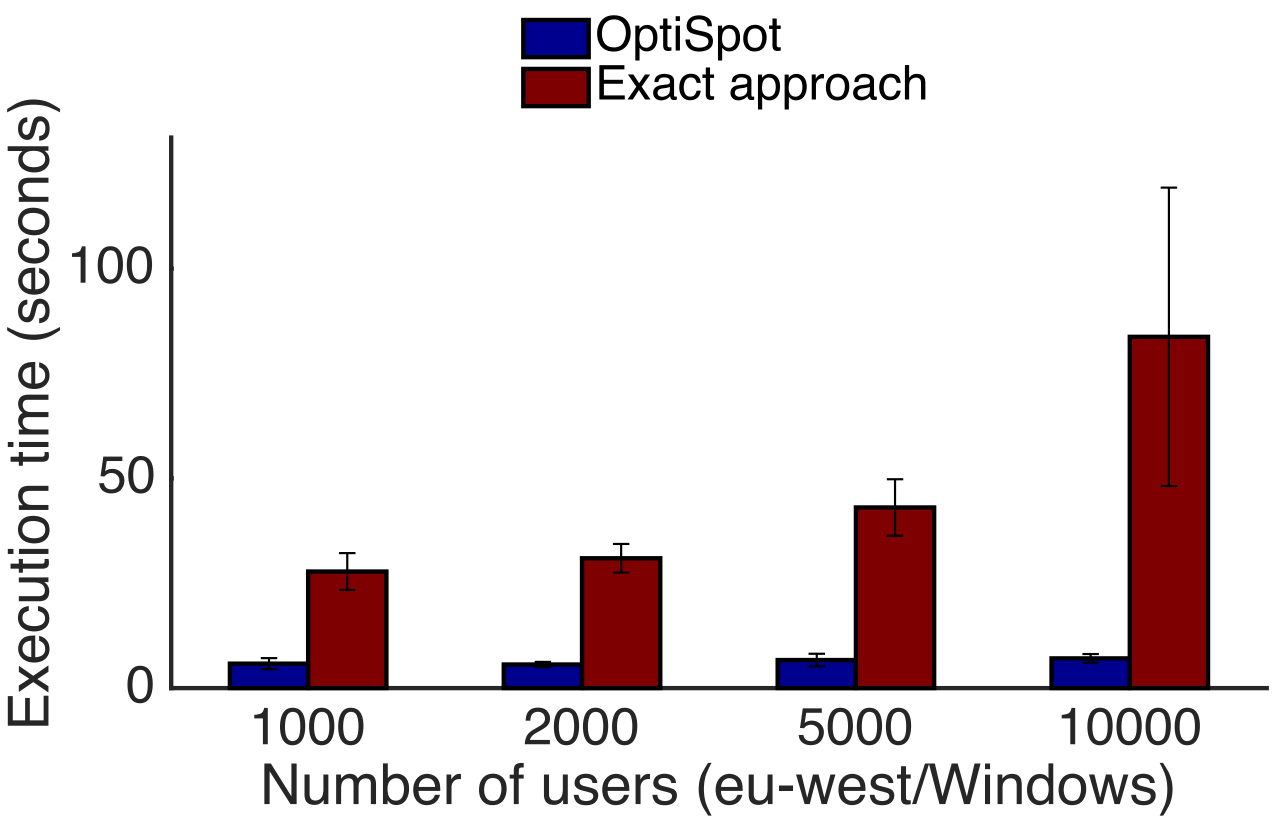


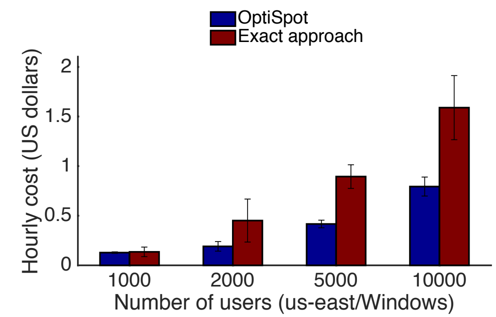


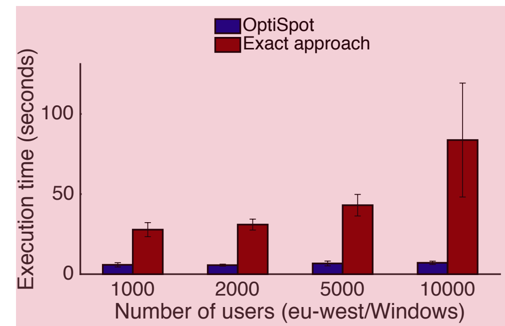


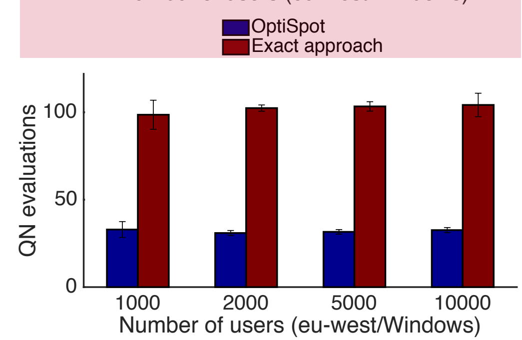
Imperial College London

Varying the Number of Users (execution time)



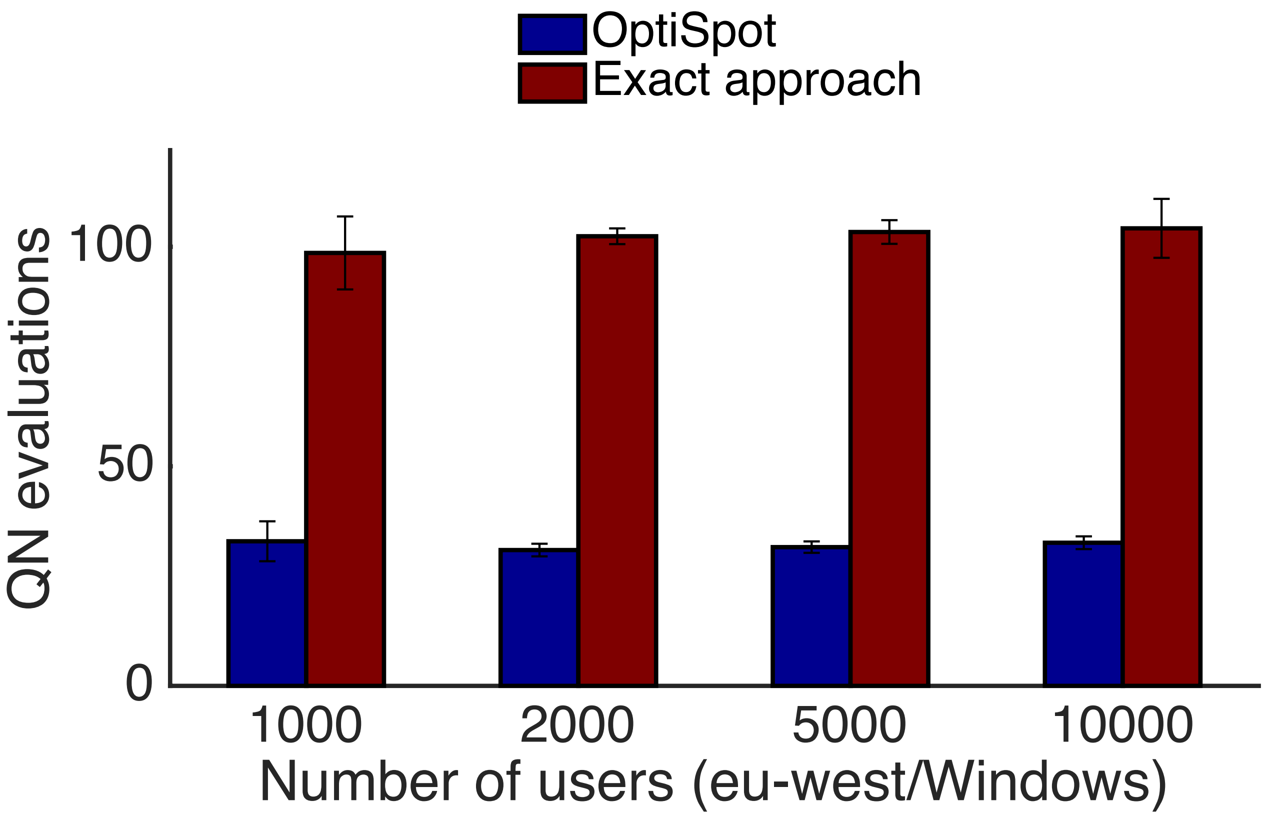


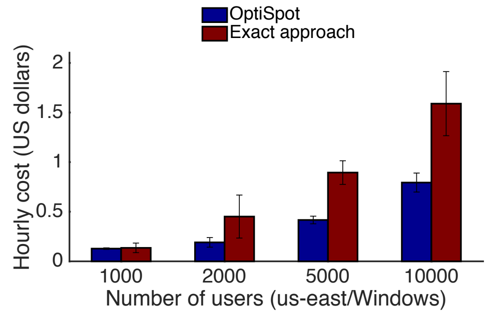


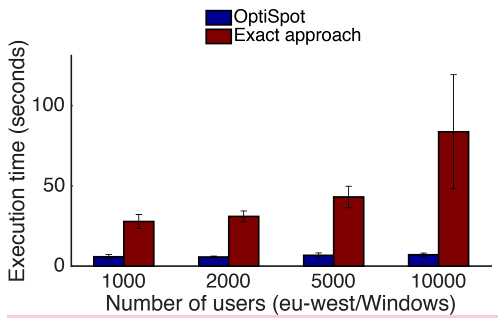


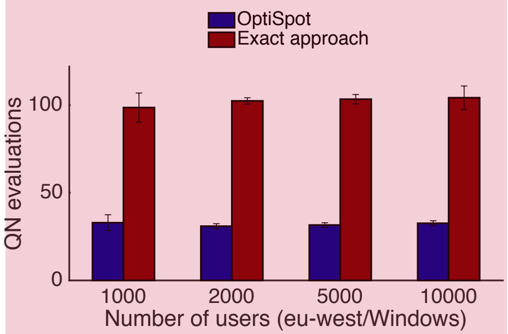
Imperial College London

Varying the Number of Users (QN evaluations)





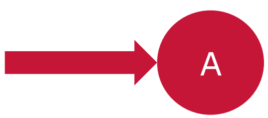




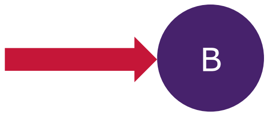
Model Improvement: Application Refactoring

Refactoring the Application Model (i.e., the QN) to improve the results

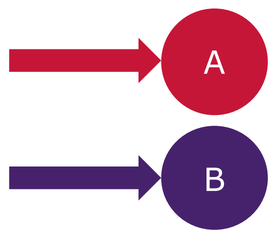
Software component replacement



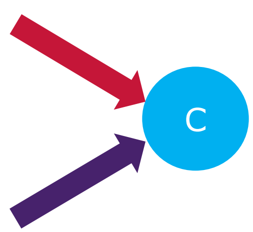
Component A is replaced by B



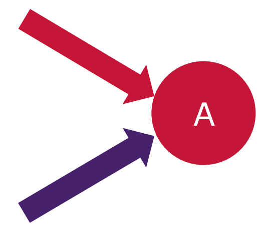
Software component merge



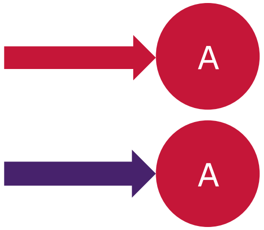
Components A and B are merged into C



Software component reassignment



Functional separation of component A into two replica



Imperial College London

Case study 2: SPECjAppServer Benchmark Application

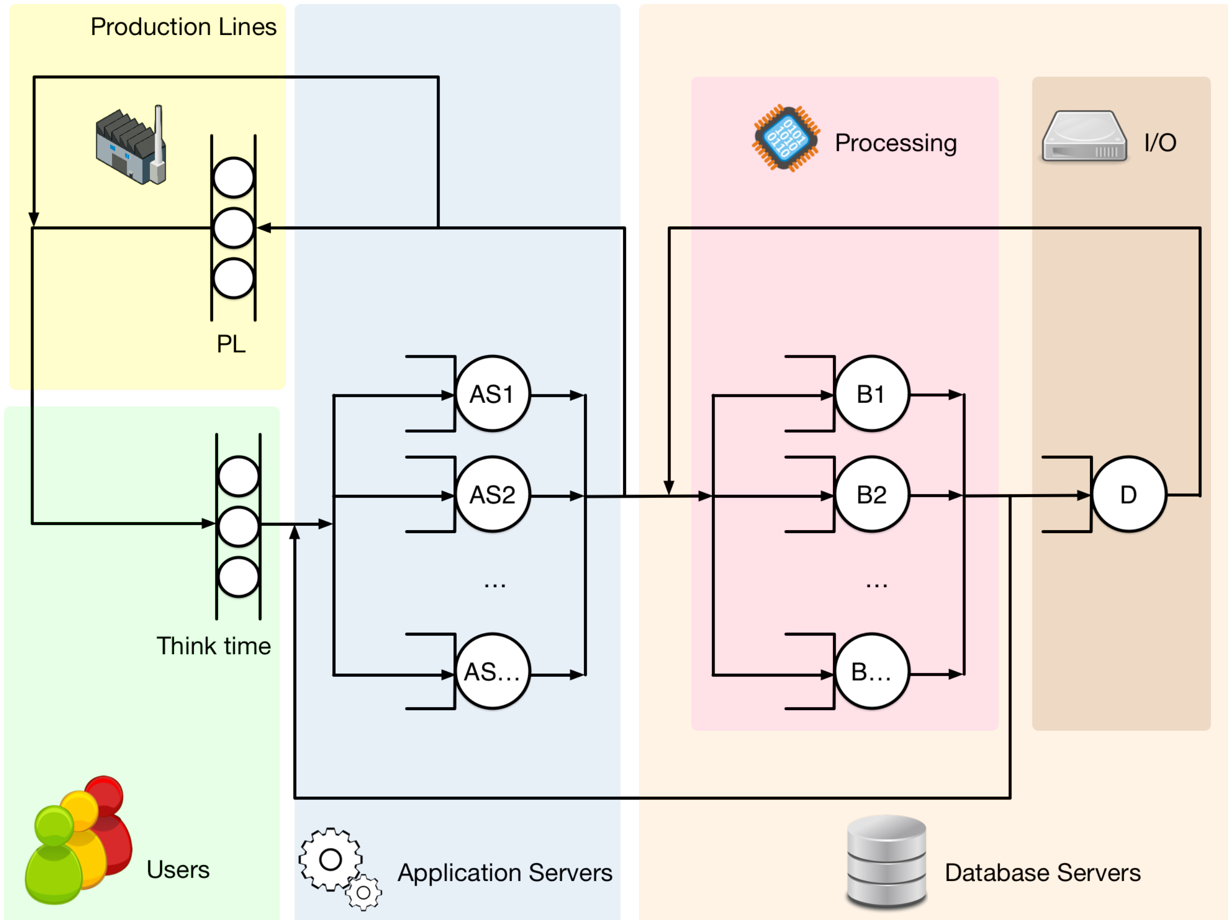
Enterprise-level business-to-business e-commerce benchmark

1 Application Server

1 DB Processor

1 DB I/O

QN description publicly available

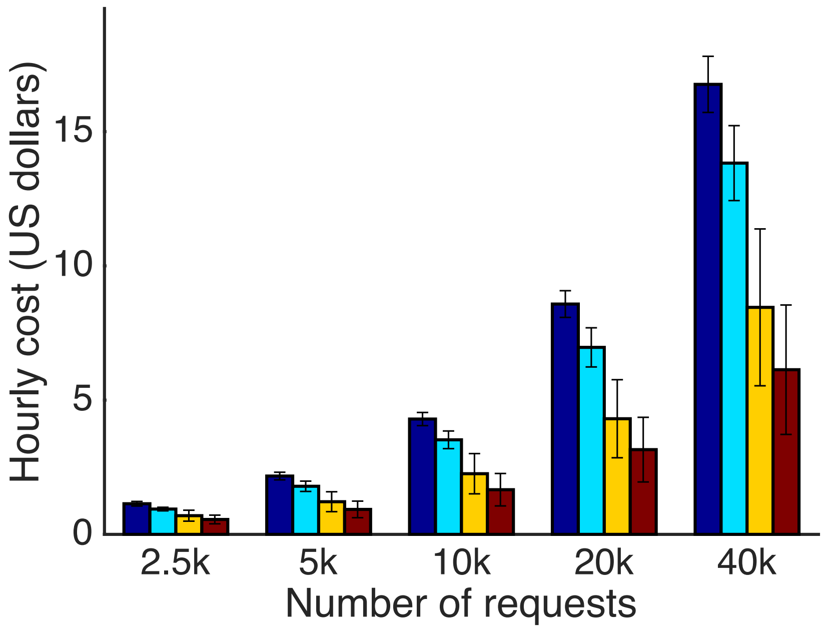


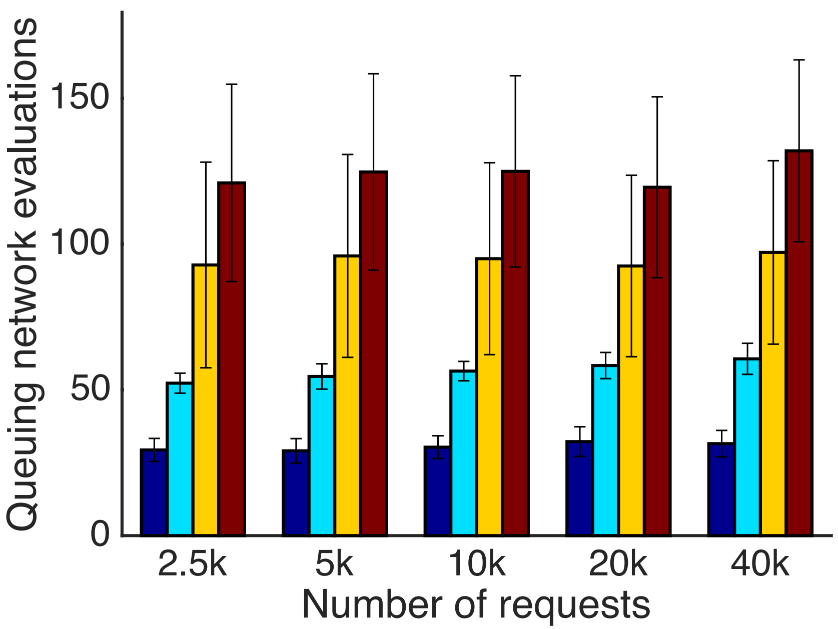
Application Refactoring: Some Results

Some experiments:

1. Only OptiSpot
2. OptiSpot with Software replacement refactoring
3. OptiSpot with Software reassignment refactoring
4. OptiSpot with Both software refactorings

+ Refactorings scale well
- More QN evaluations are needed





Conclusions

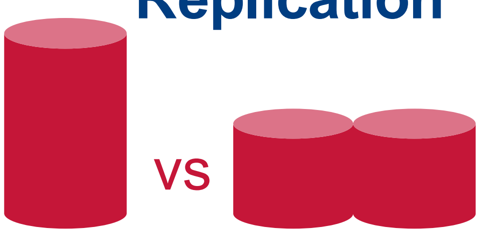
Cost-aware approach to support provisioning and allocation decisions

- Decide which resources to rent and what to deploy on them
- Random environment representation for spot and preemptible resources
- Model-driven Application Refactoring

Lightweight approach for a complex problem → **Run-time adaptation** → **Allocation Deallocation Migration Replication**

Future challenges

- Runtime adaptation experiments
- Burstable instances
- Multidimensional requirements



Different number of CPUs
Same number of ECUs

Imperial College London

Thank You!