Advanced features
for
Multi-formalism modelling
with
Java Modelling Tools

Giuseppe Serazzi
Politecnico di Milano, Italy
giuseppe.serazzi@polimi.it

Joint work with:
Giuliano Casale, Lulai Zhu
Imperial College London, UK

28/11/19

Outline

- the JMT suite of tools
- Fork/Join (Queueing Networks, QN)
  - timeout of computing time
  - approximate computing
- Place/Transition (Petri Nets, PN)
- Multi-formalism models (QN + PN)
  - core allocation in HPC system
  - Adaptive Car Navigation Service
- What’s Next

Java Modelling Tools (JMT)

The project

- co-developed by Politecnico di Milano and Imperial College London
- 63+K downloads, open source http://jmt.sourceforge.net

Supported models

- Queueing Systems
- Queueing Networks (QN)
  - Product-form
  - Extended (fork/join, blocking, priorities, …)
- Petri Nets (PN)
  - Stochastic Petri Nets (SPN)
  - Generalized Stochastic Petri Nets (GSPN)
  - Coloured Petri Nets (CPN)
- Multiformalism models (QN + PN)

JMT website http://jmt.sourceforge.net

DEIB -Politecnico di Milano - Italy

Imperial College London

Department of Computing - UK

Coordinators: G.Casale, G.Serazzi

Java Modelling Tools - JMT

Main Menu

- Introduction
- Download JMT
- Requirements
- JSIM*graph*
- JSIM*wiz*
- JMVA
- JABA
- JWAT
- JMCH
- Documentation
- License

manual, papers, video tutorials

Links

- Need Help?
- Report a Bug
- Request a New Feature
- Contribute a Patch
- Sourceforge Page

SOURCEFORGE

Introduction

Project Description

Java Modelling Tools (JMT) is a suite of applications developed by Politecnico di Milano and Imperial College London and released under GPL license.

The project aims at offering a comprehensive framework for performance evaluation, system modeling with analytical and simulation techniques, capacity planning and workload characterization studies.

The current stable version of the suite includes six Java applications:

1. JSIM*graph* - Queueing network and Petri net simulator with graphical user interface
2. JSIM*wiz* - Queueing network and Petri net simulator with wizard-based user interface
3. JMVA - Mean Value Analysis and Approximate solution algorithms for queueing network models
4. JABA - Asymptotic Analysis and bottlenecks identification of queueing network models
5. JWAT - Workload characterization from log data
6. JMCH - Markov chain simulator

Versions

Download: **JMT 1.0.4** (Released 2019-Sep-04).

Release announcements: please subscribe to the jmt-announce list.

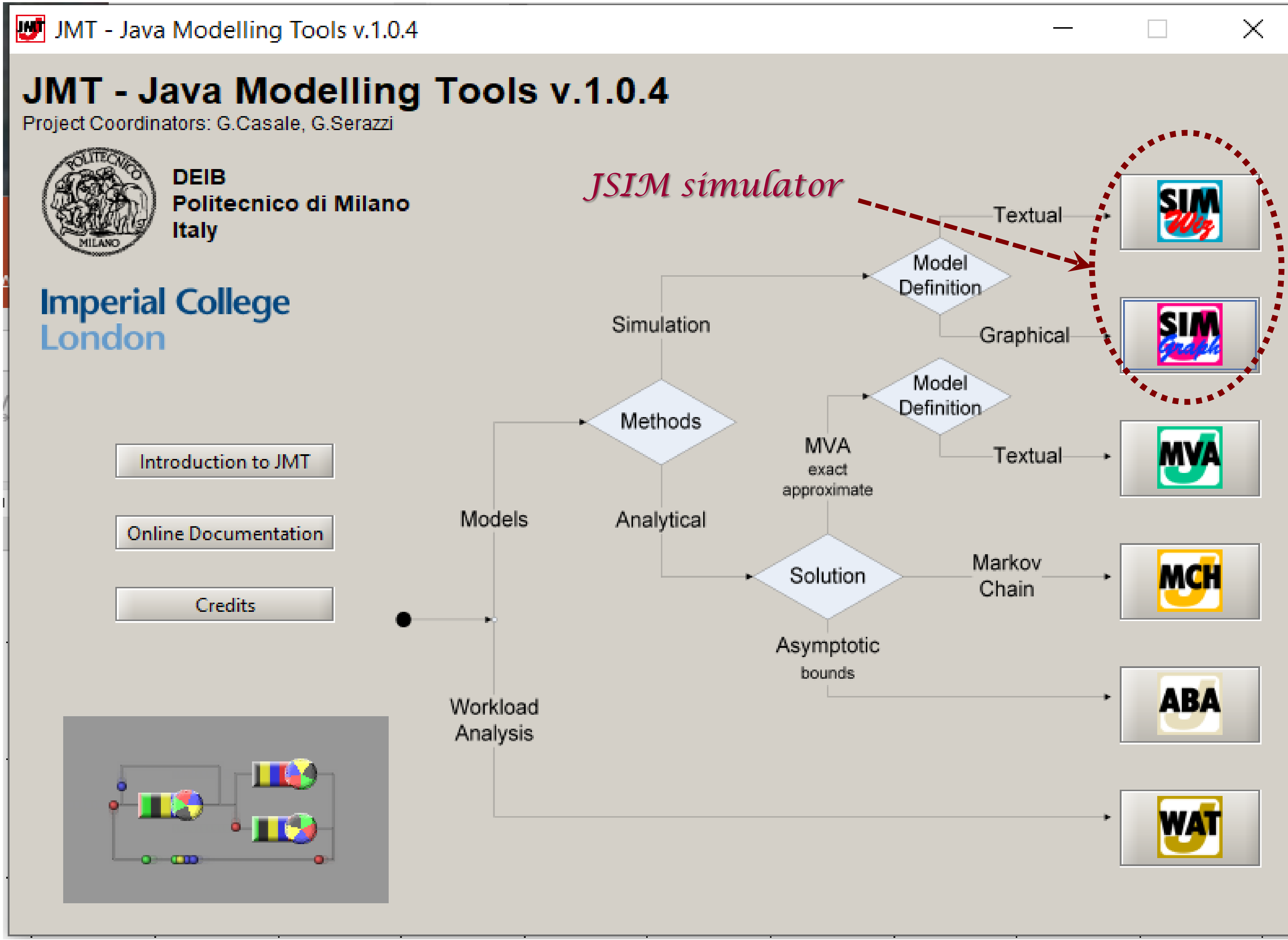
New users: new to JMT? Check out the manual and our video tutorials.

Reference

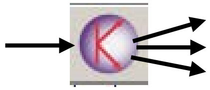
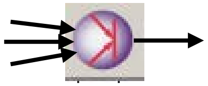
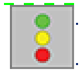
If you are using JMT for scientific papers, academic lectures, project reports, or technical documents, please cite:

M.Bertoli, G.Casale, G.Serazzi.

JMT start screen http://jmt.sourceforge.net

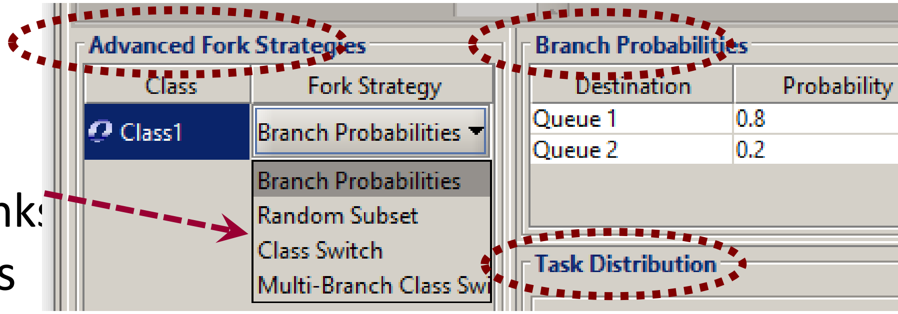


Fork/Join (QN)

Fork  Join  Semaphore 

- Fork
  - jobs split into tasks (with same id of the parent job) executed in parallel

  advanced fork policies:
  - Branch prob.: randomize output links and no. of tasks per link
  - Random subset: choose n-out-of-k output links
  - Class Switch: assign new class to forked tasks



- Join
  - fires a job when all tasks forked from the same parent job are executed

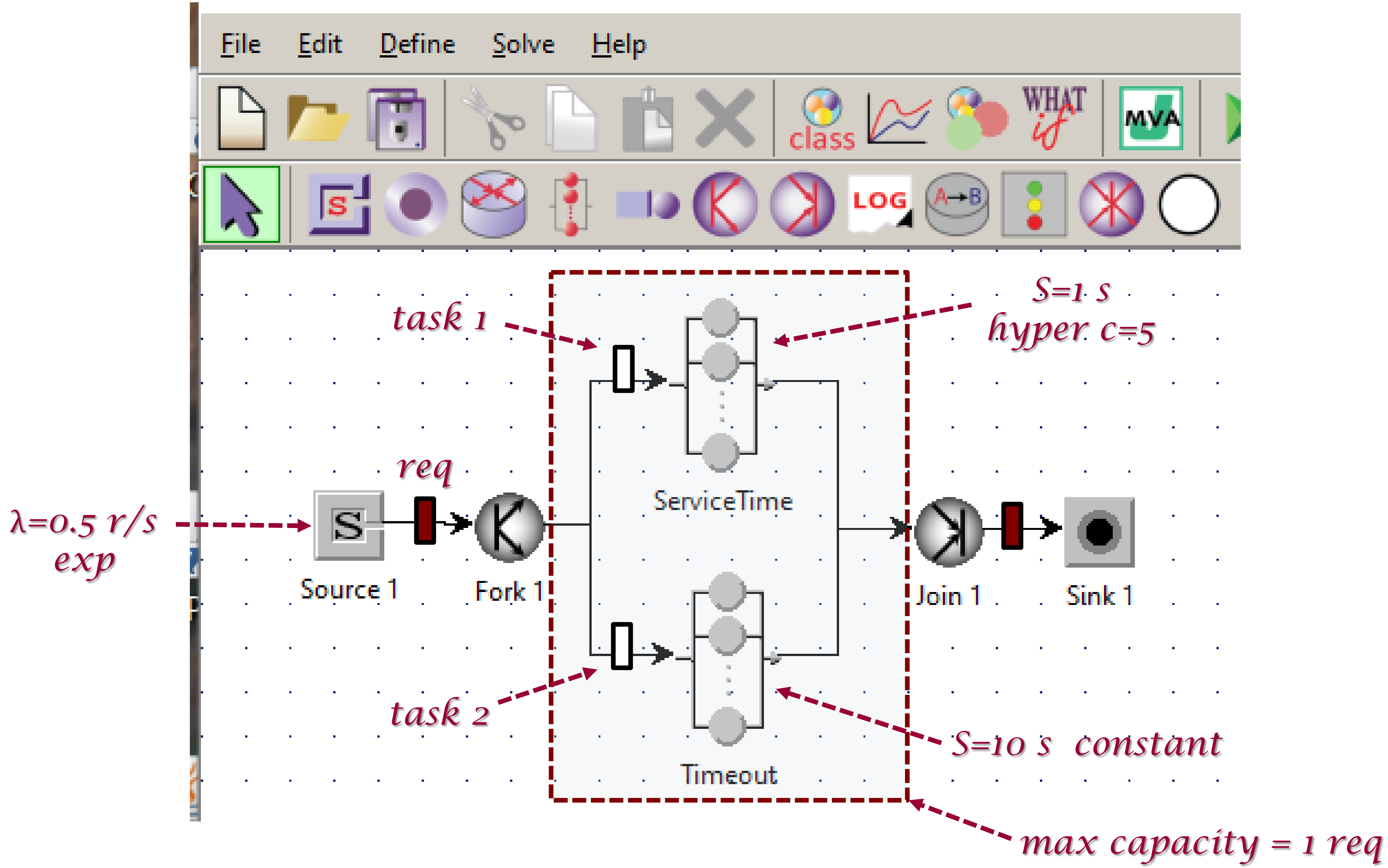
  advanced join policies:
  - Quorum: wait a subset of tasks of the same parent job
  - Guard: like quorum but requires a given class mix (multiclass workload)
- F/J region
  - finite capacity on no. of jobs, multiple join
  - Semaphore : block the first N tasks of the same job, when N is reached unblock all

timeout on computation time – the problem

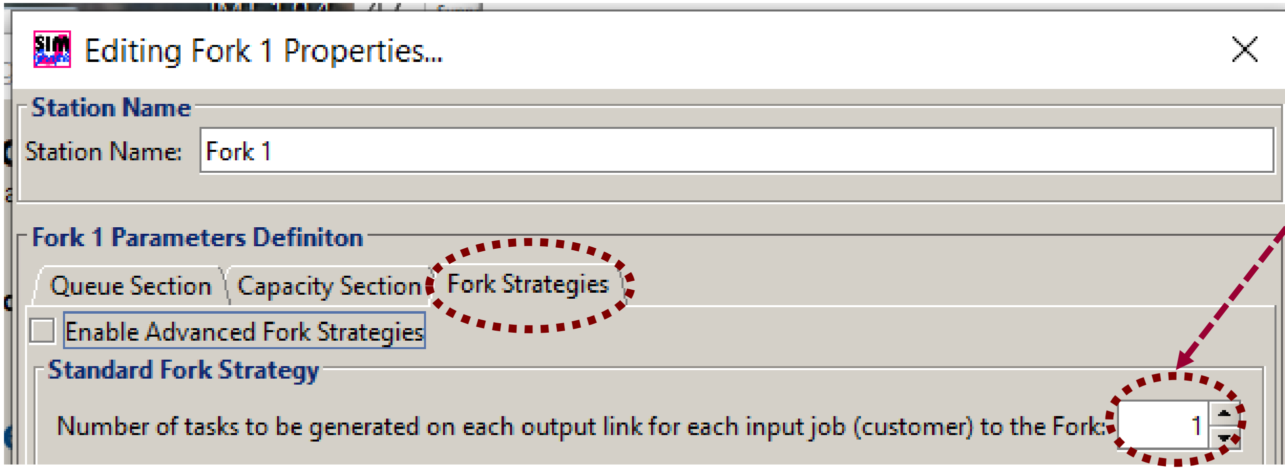
- the service demands of the algorithms executed by a genetic program are highly variable
- a limit (timeout) to the computation time of each algorithm must be set in order to complete the program execution in a given amount of time
- the timeout can be described by a constant value or other statistical metrics: mean, std.dev, distribution (independence from distributions)
- investigate the behavior of the response time of each algorithm as a function of the timeout

- the model
  - F/J region with capacity limited to one request
  - two Delay stations: one for service demands, one for timeout
  - the Fork has two output links, one for each Delay
  - job split into two tasks, one for each output link
  - Join waits only the first task executed (Quorum=1)

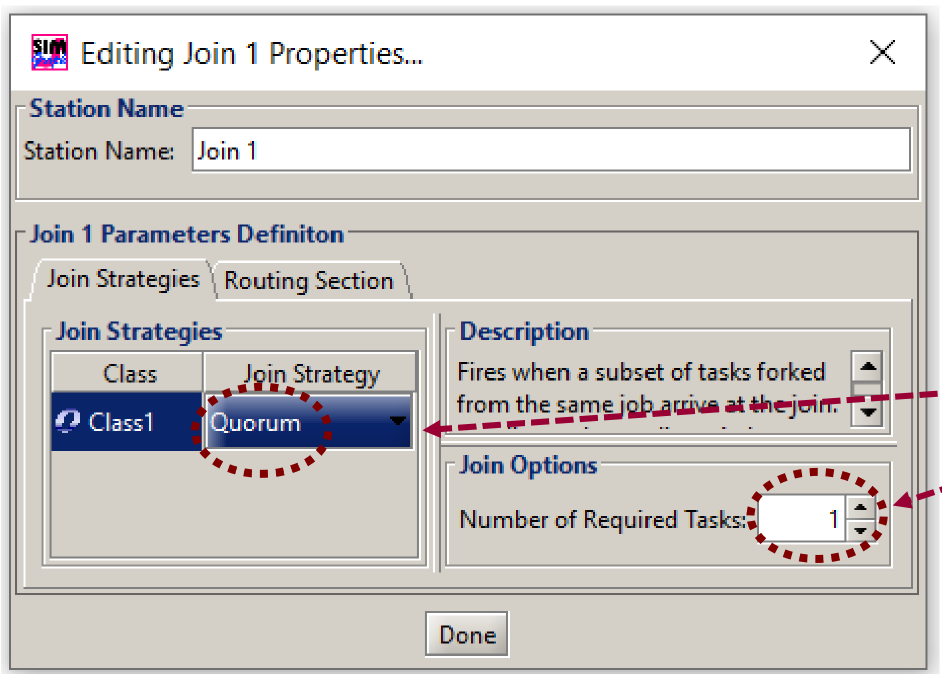
timeout – the JSIM model



timeout – Fork/Join parameters

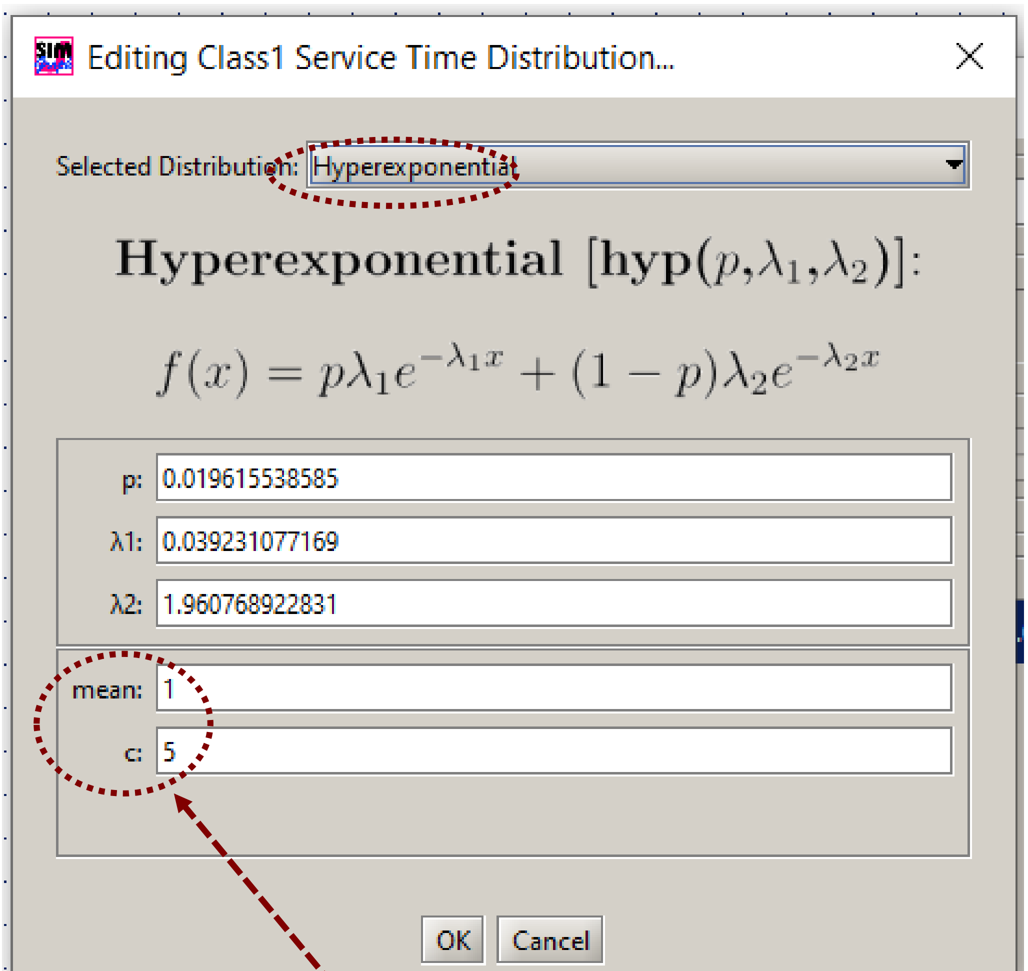


- a req generate one task per link
- the max number of req in the F/J section is 1

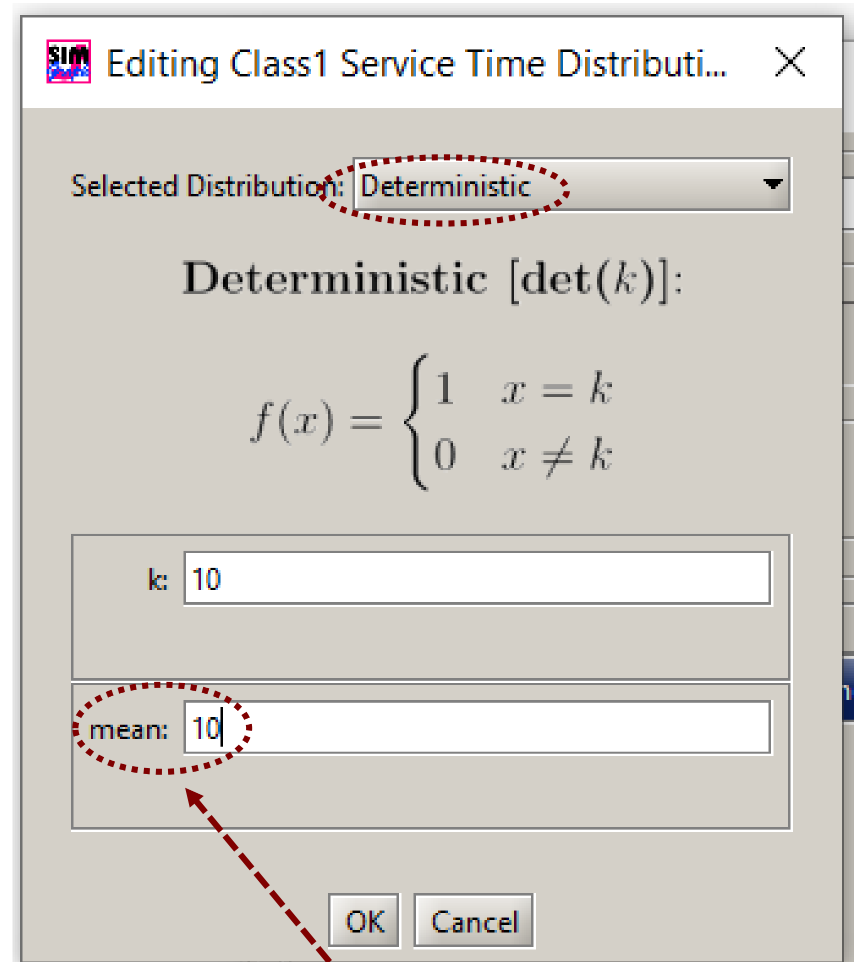


- *Join Quorum=1*: the first task that complete the execution exit the Join

timeout – service times of the two Delays

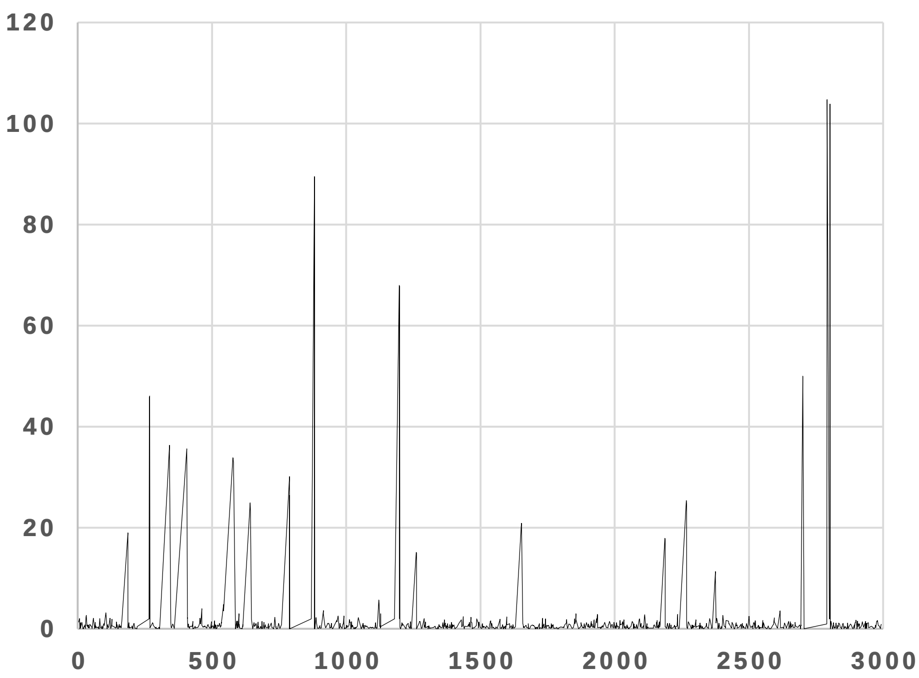


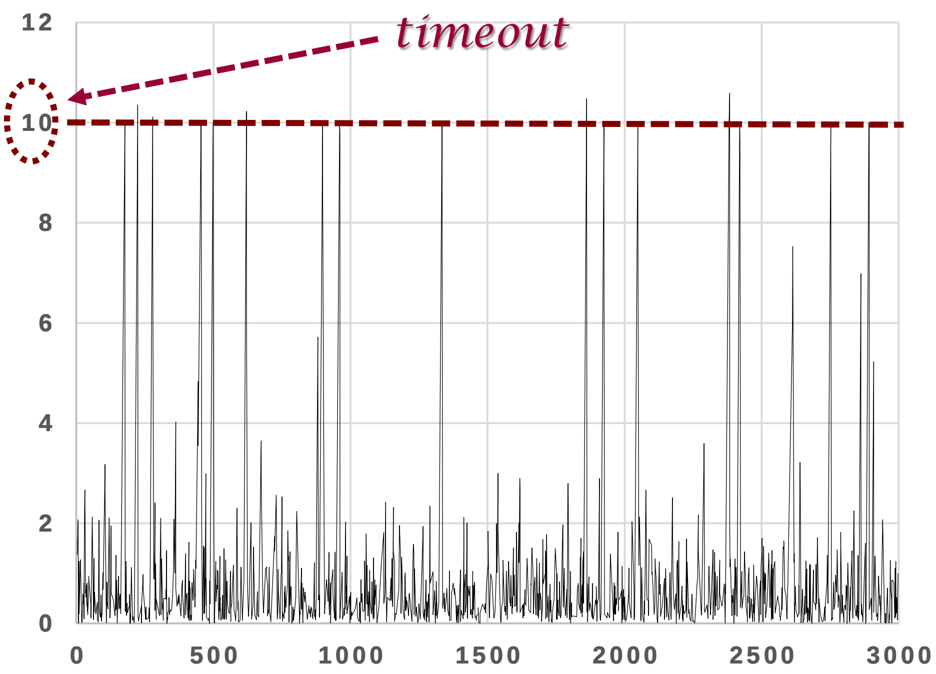
Service times hyper-exp distr.
S=1 s coeff.var.=5



timeout value
constant = 10 sec

timeout – service times of requests





service times without control

service times with timeout

timeout – response times Fork/Join *vs* timeout

Fork Join Response Time

Average response time for each selected class in each selected Fork/Join section.

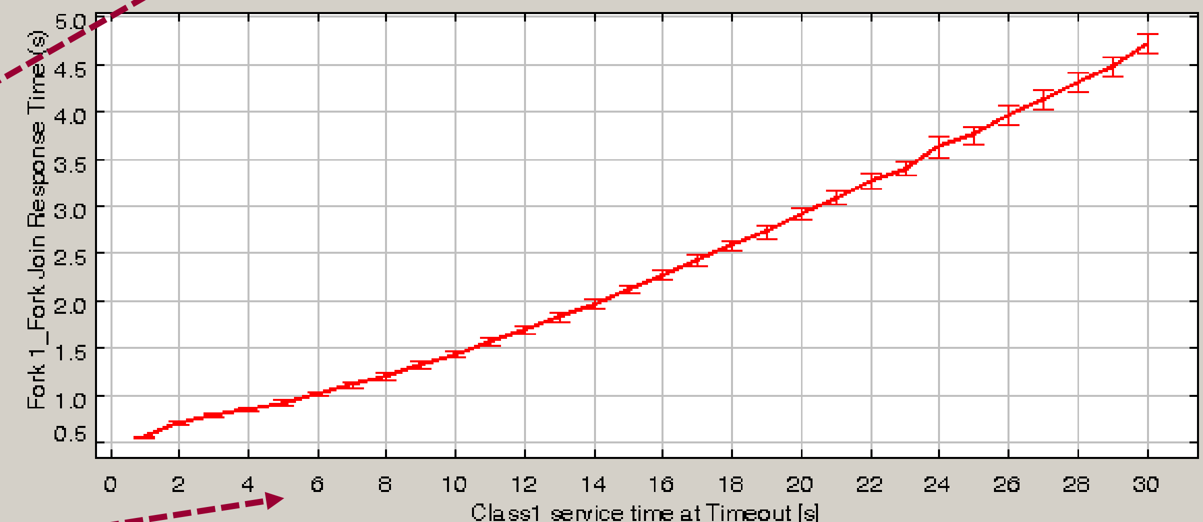
n.o of JSIM invocations

Description

Station Name:	Fork 1 (Fork Join)
Class Name:	All classes
Conf.Int. / Max Rel.Err. (0-1):	0.99 / 0.03
Models:	30
X min:	1
X max:	30
Y min:	0.539
Y max:	4.826

Plot

☑ Show confidence interval range



timeout 1÷30 sec

Simulation Results

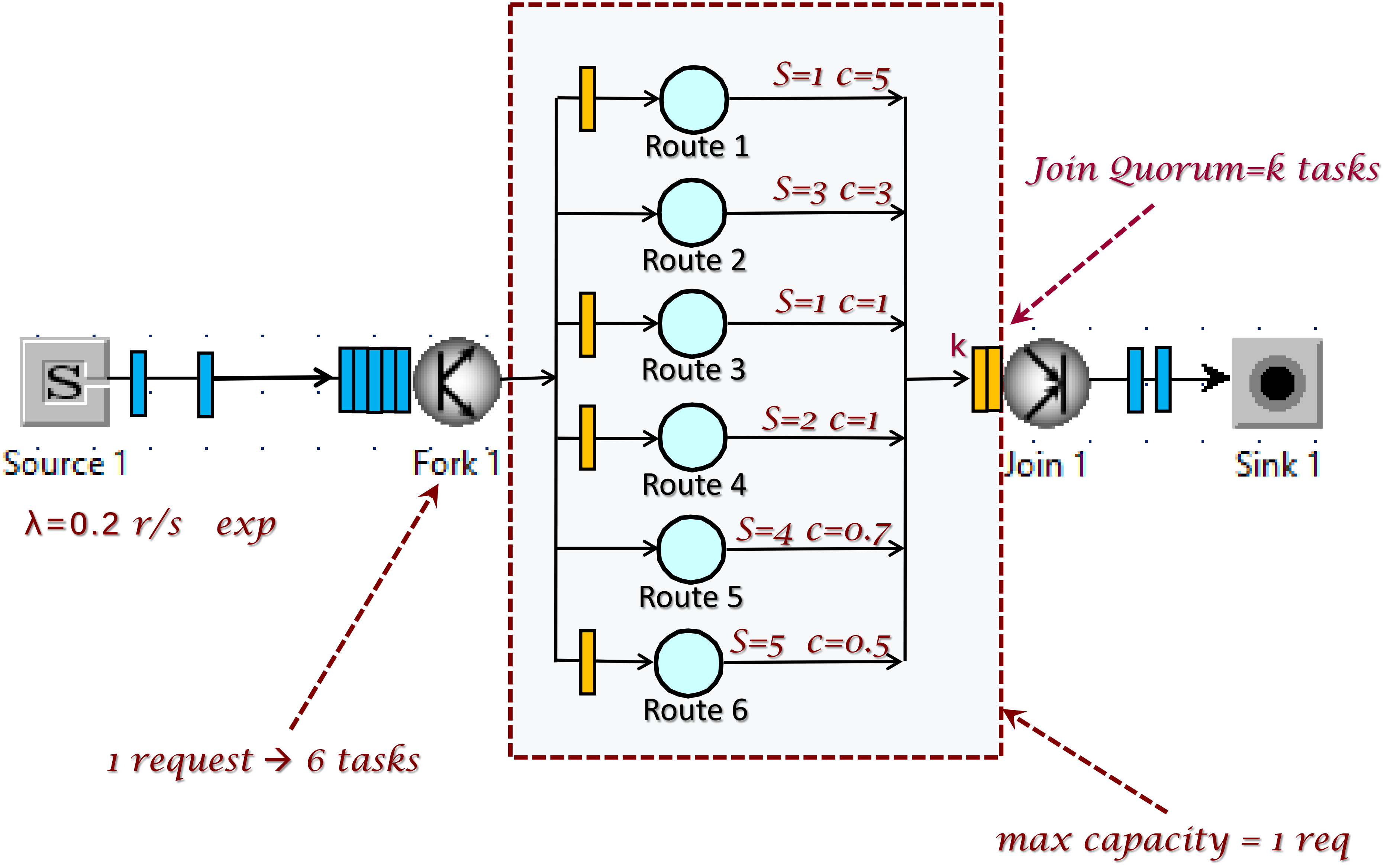
*	)s	21.000 s	22.000 s	23.000 s	24.000 s	25.000 s	26.000 s	27.000 s	28.000 s	29.000 s	30.000 s
Mean Value (s)	9	3.0853	3.2729	3.3960	3.6276	3.7516	3.9673	4.1267	4.3109	4.4779	4.7207
Max (s) (Conf.Int.)	9	3.1578	3.3557	3.4645	3.7361	3.8502	4.0649	4.2261	4.4051	4.5878	4.8261
Min (s) (Conf.Int.)	3	3.0128	3.1900	3.3276	3.5192	3.6531	3.8696	4.0272	4.2166	4.3679	4.6153

approximate computing: Alternative Route Planning

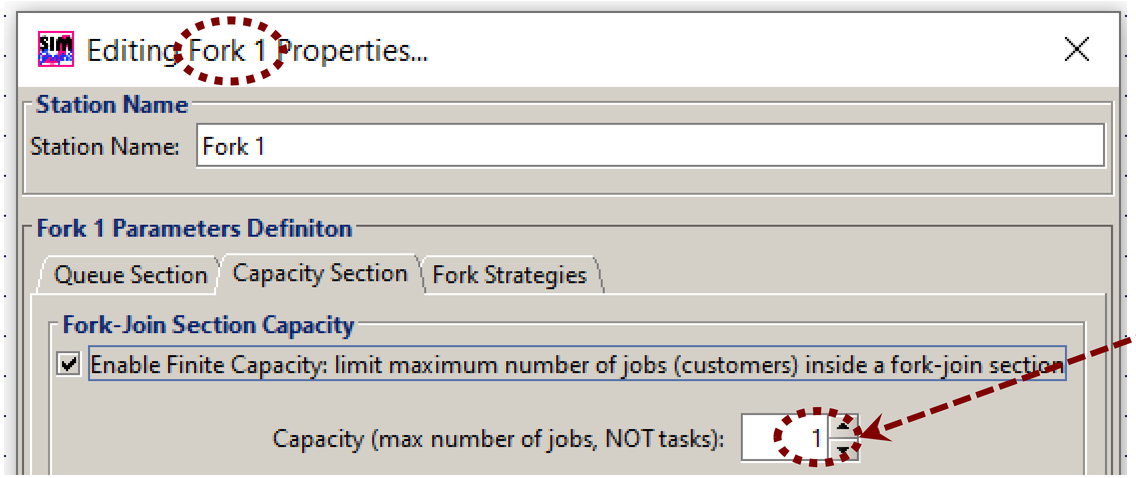
- Alternative Route Planning: several algorithms are executed in parallel to compute N=6 alternative routes from source to destination in a Car Navigation System (highly variable computation times: travelling time, municipality policies: no city center, limited traffic areas, street closed, ...)
- to limit the computation time of the best route to R<=3 sec, sub-optimal solutions (approximate) are computed using a subset k of the N routes
- impact on the best route computation time *wrt* variability of the computation times and the subset size k

- the model
  - F/J region with capacity limited to one request
  - 6 delay stations, one for each algorithm
  - a request is split into N=6 tasks in output from Fork (1 task per link)
  - Join waits the complete execution of only the first k tasks

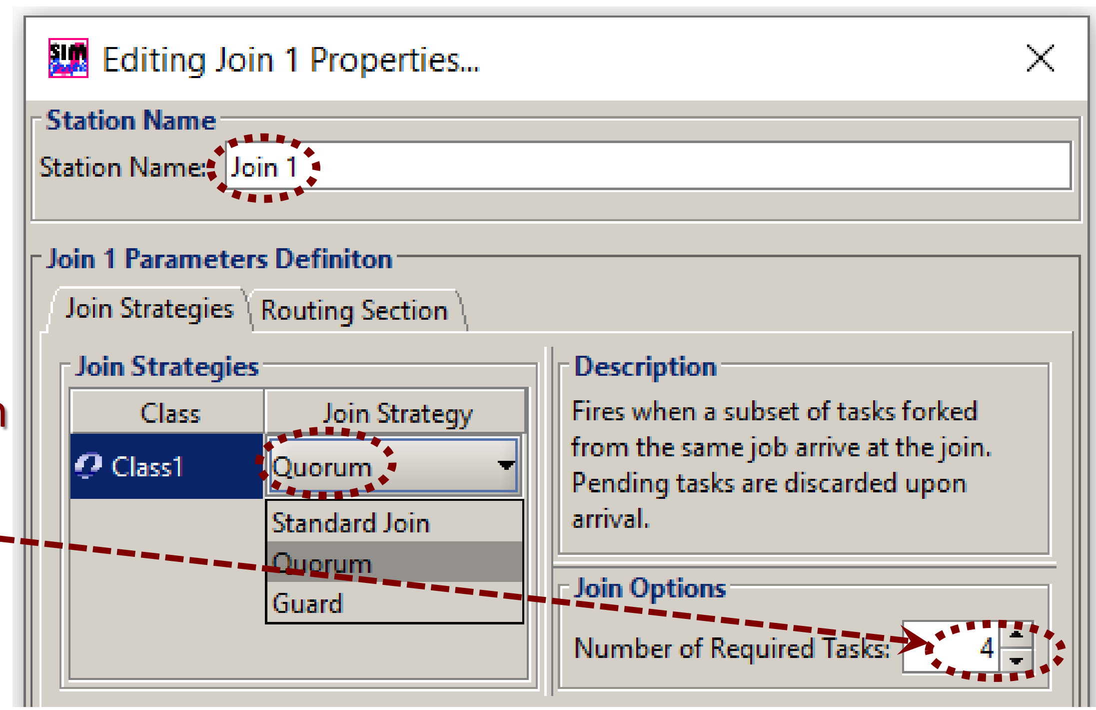
approx. comp.– the model



approx. comp. - Fork/Join parameters

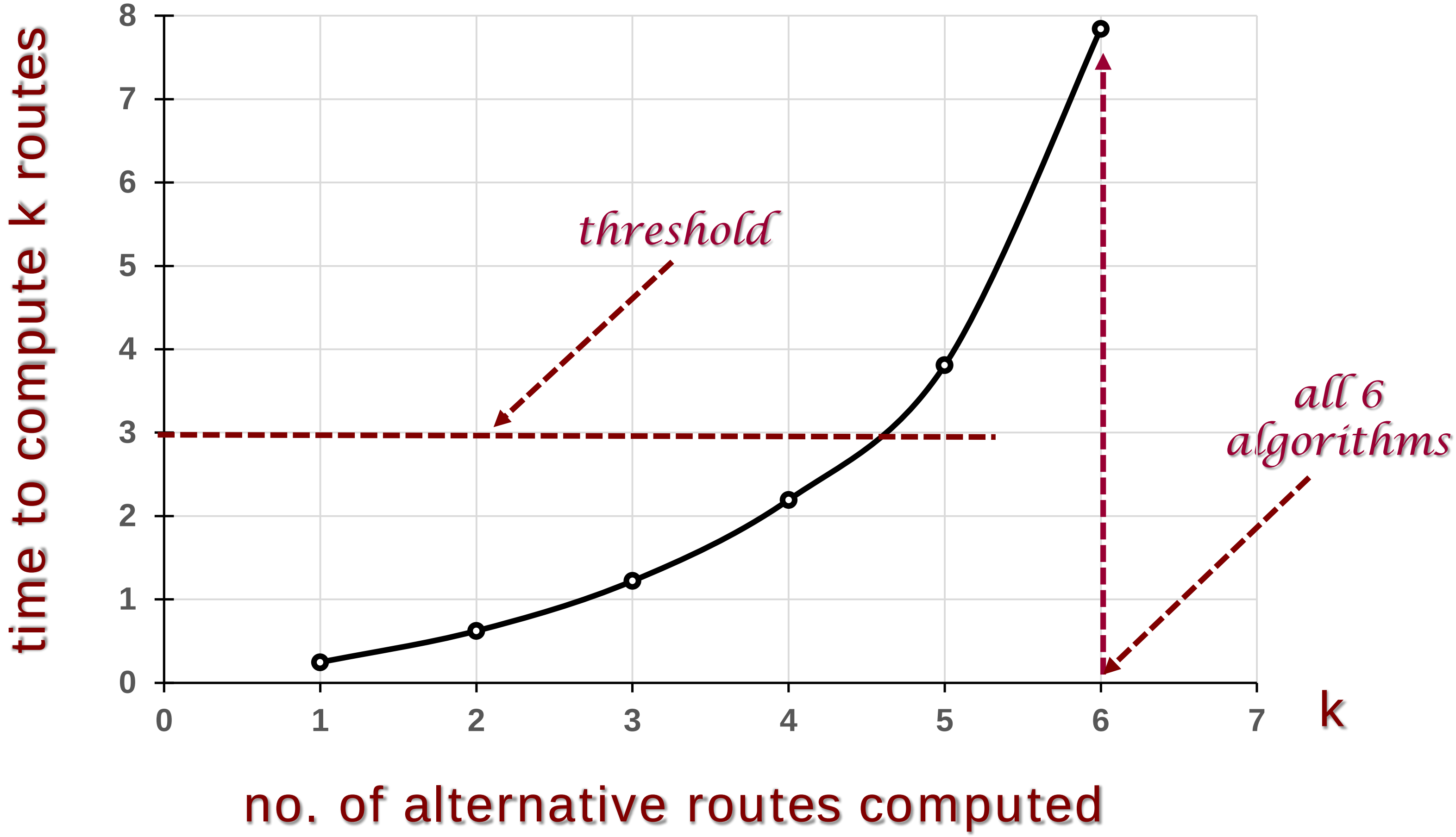


- each req generate one task per output link
- the max number of req in the F/J section is 1



Join Quorum=4: wait the computation of the first 4 routes then fire

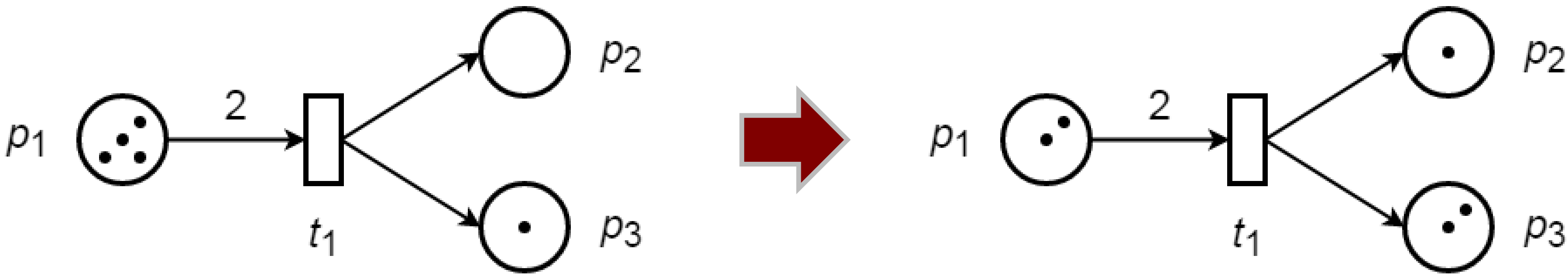
approx. comp. – computation time of k routes



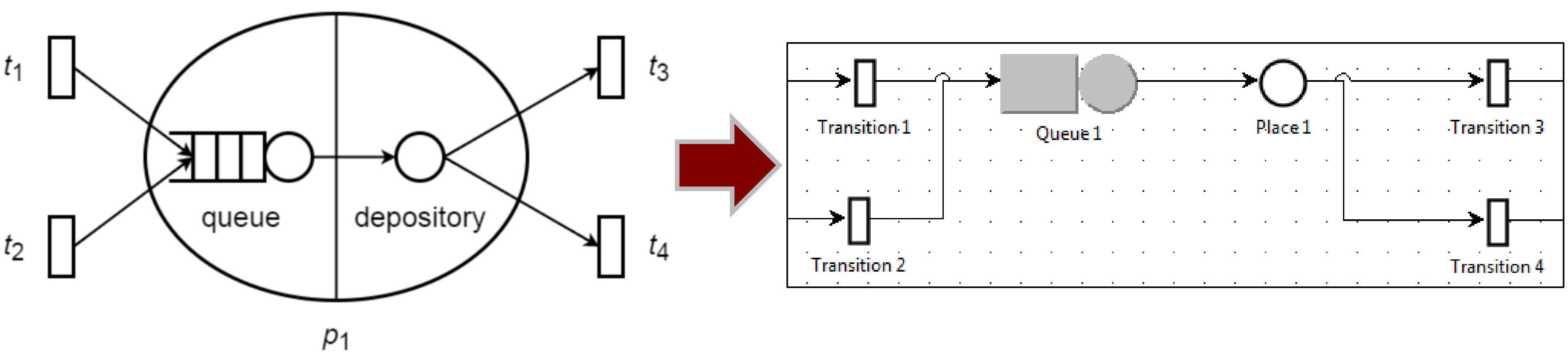
Place – Transition (PN)

PN elements supported

- Places and Transitions (PN)

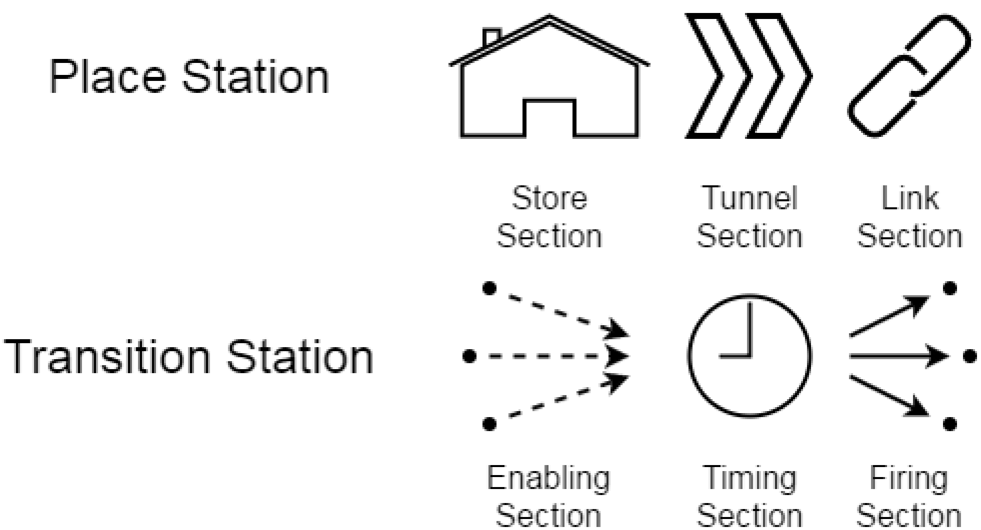


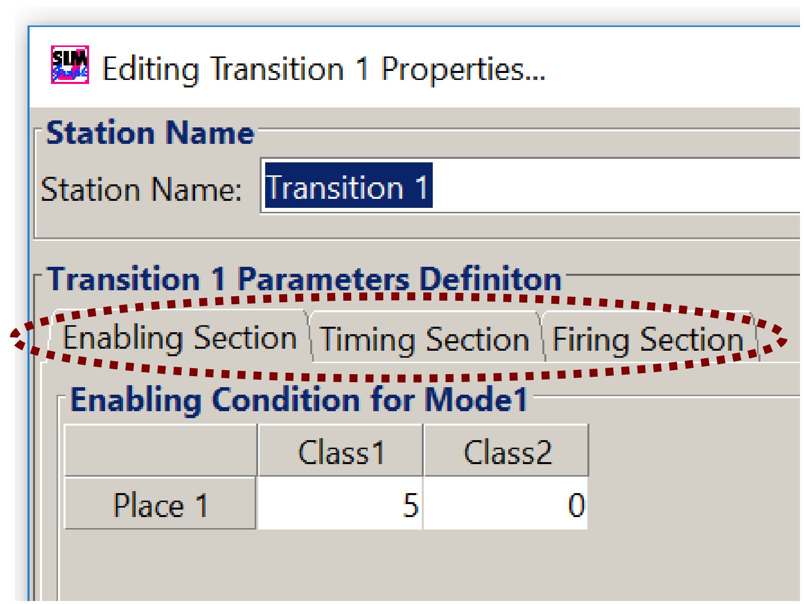
- Multi-formalism models (QN + PN)



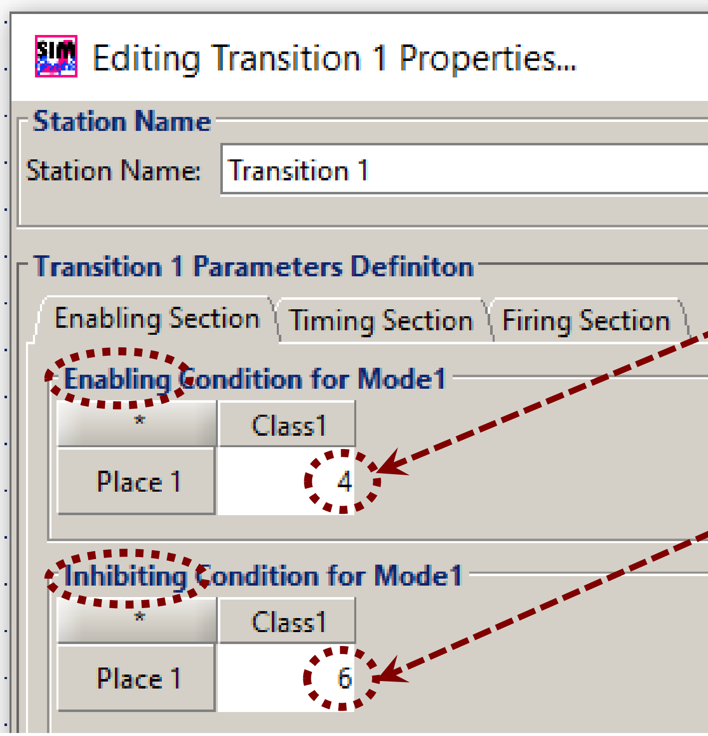
PN sections & modes

- JMT design paradigm extends to PN elements
  - Enabling, Inhibiting, Timing, Firing sections
- Mode: a rule to activate and fire a transition





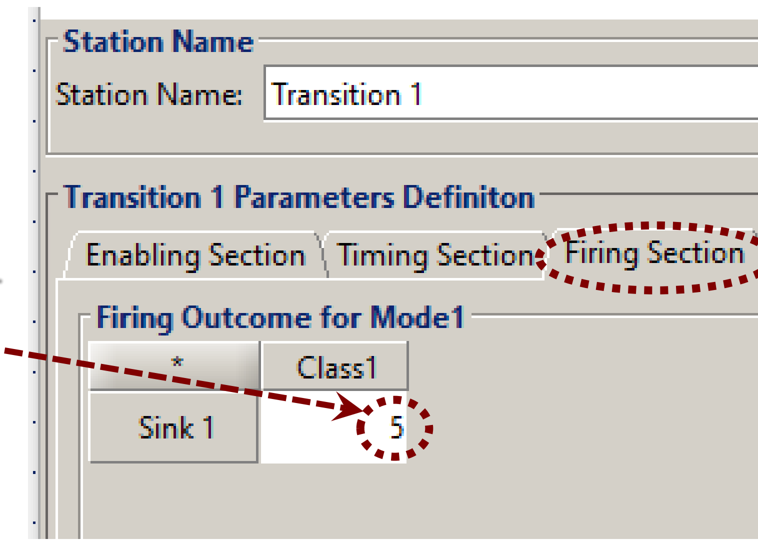
PN - transition parameters



enable firing when 4 class1 requests are arrived (4 tokens of colour class1)

inhibit the fire when 6 or more requests are arrived

when enabled, send 5 class1 requests to Sink1

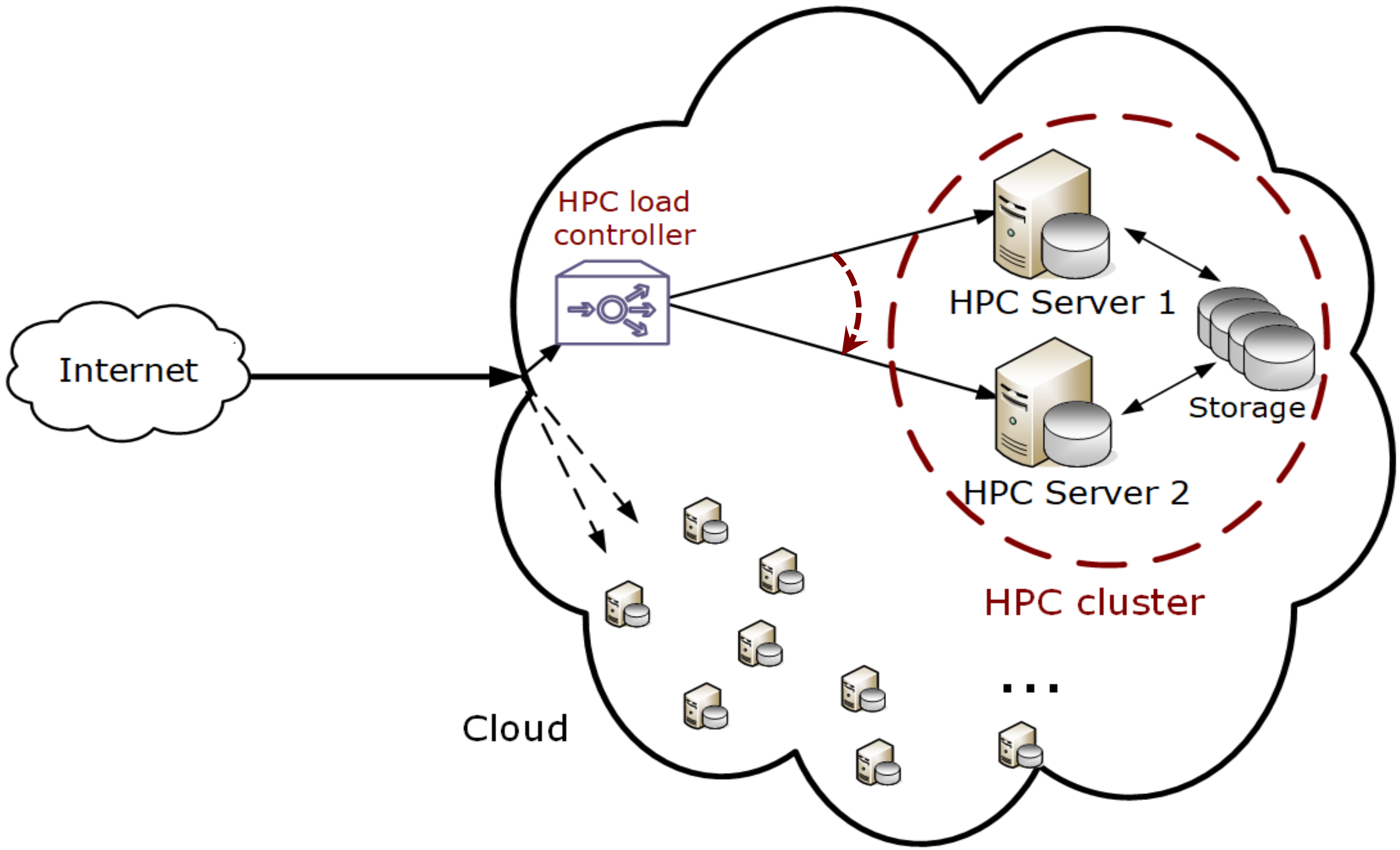


Multi-formalism models (QN+PN)

HPC cluster on cloud: core allocation and dynamic routing

- a cloud provider offers a HPC service through a cluster infrastructure with two powerful HPC multicore systems (HPC Server1, HPC Server2) sharing a high performance storage and networking components
- the HPC app considered (find nearest available parking space) is compute intensive and the number of cores it uses depends on the number of requests in execution (one request per core assuming a single-thread core)
- there is a high variability of arrival traffic of requests and compute demands
- arriving requests to the cluster are initially routed to HPC Server1
- to avoid decreasing the performance of other apps running on HPC Server1, the number of cores that can be allocated to this app is limited to N
- when this threshold is reached, new incoming requests are dynamically routed to HPC Server2 (where an instance of the same app is running)

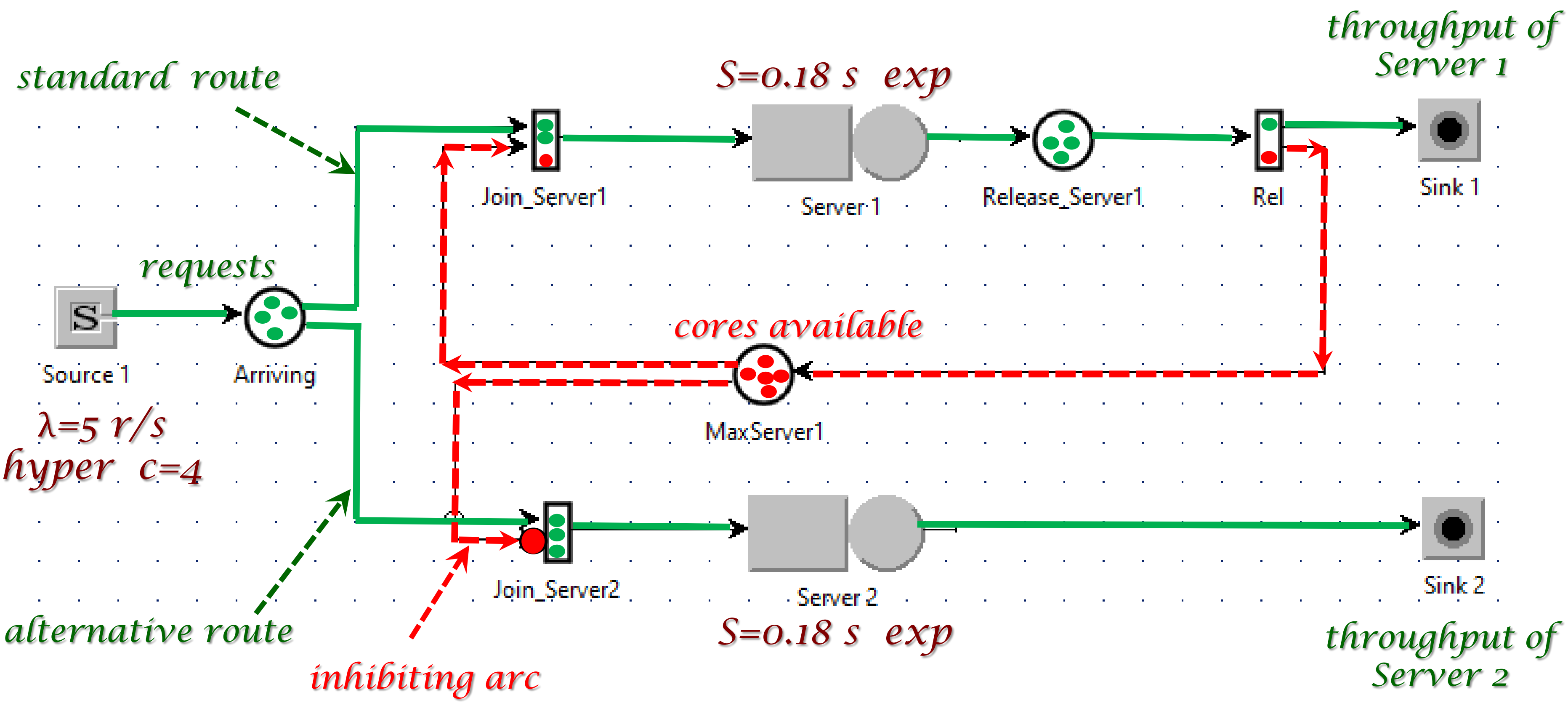
HPC cluster on cloud: the infrastructure



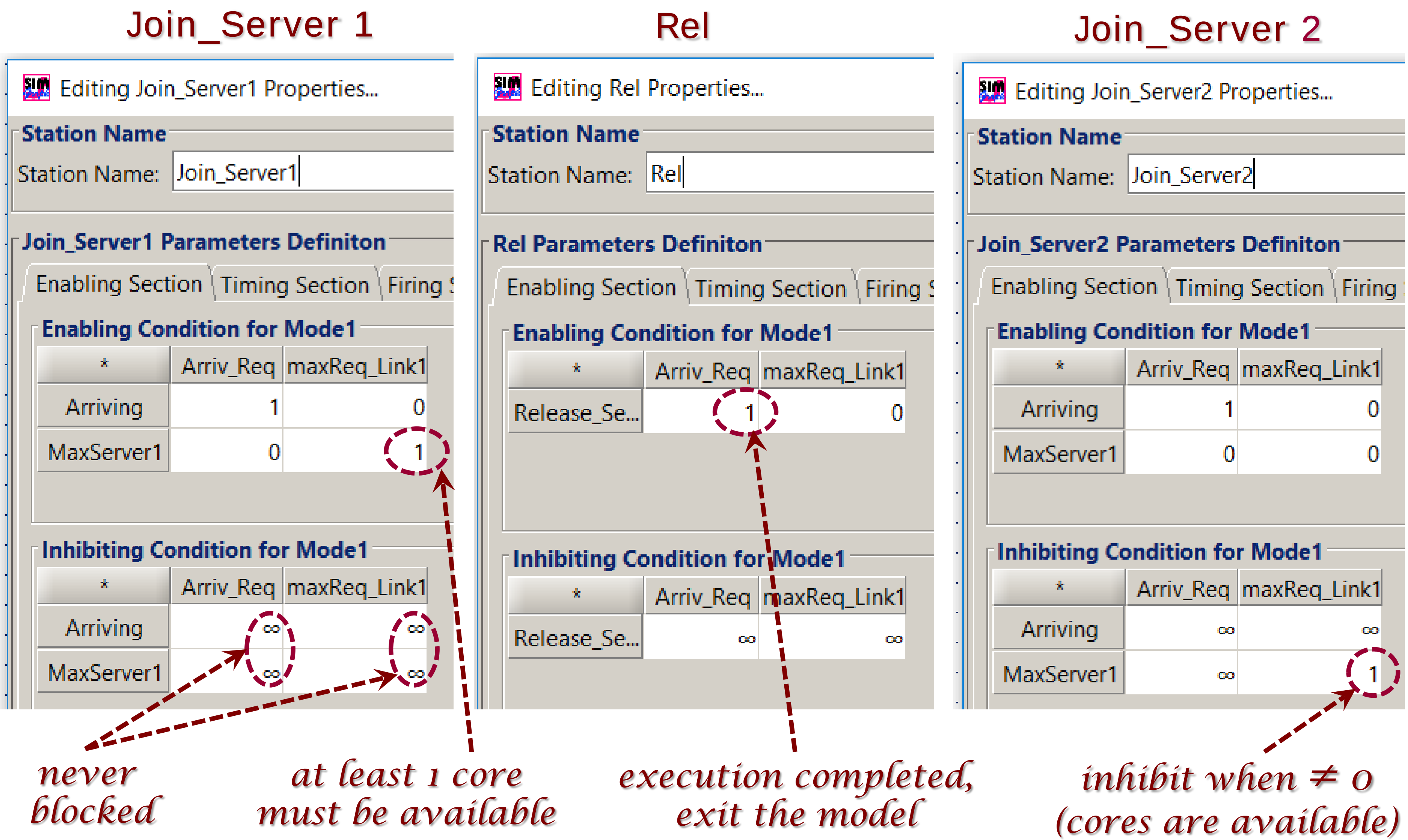
- GOAL: evaluate response times and throughput of the requests executed on the HPC cluster *wrt* the constraint on the maximum number of cores it can use in HPC Server1 (in the range 1-190)

HPC cluster: the model

- 2 classes of customers: requests (open) and cores (closed)

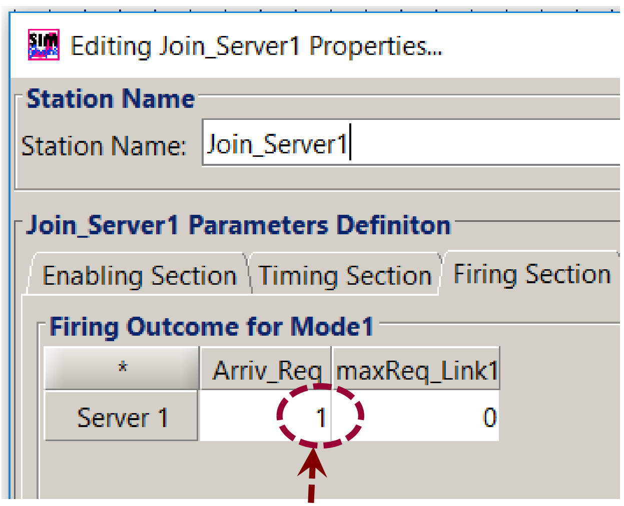


HPC cluster: transitions enabling/inhibiting conditions



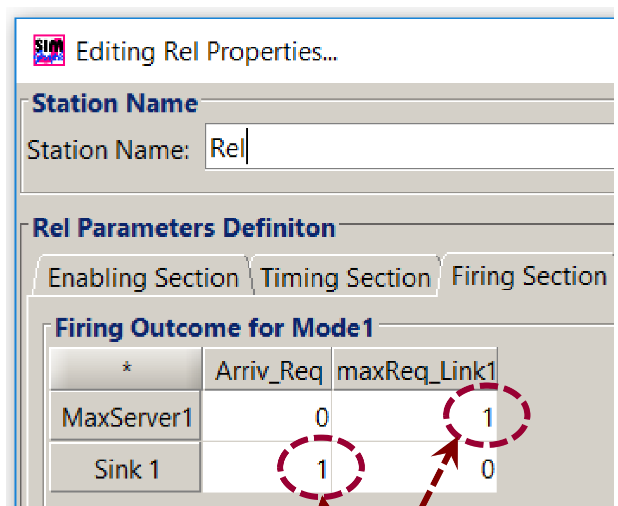
HPC cluster: transitions firing outcome

Join_Server 1



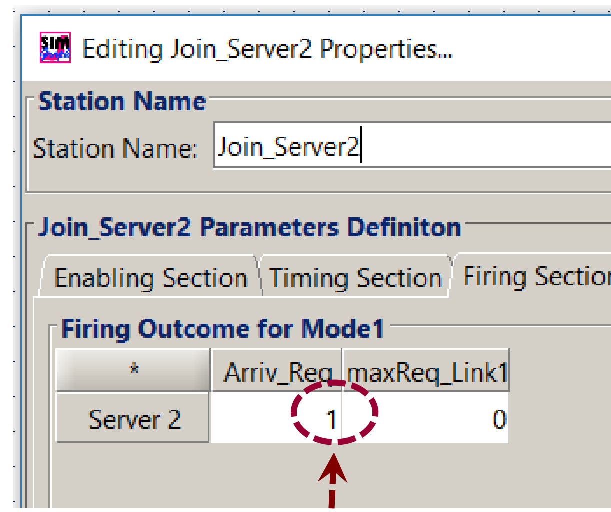
1 request sent to Server 1 (when enabled)

Rel



1 request completed and exit + 1 core becomes available (when enabled)

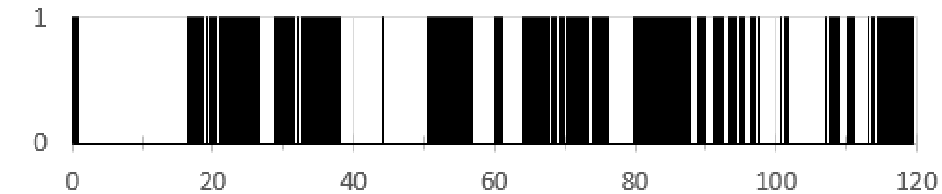
Join_Server 2



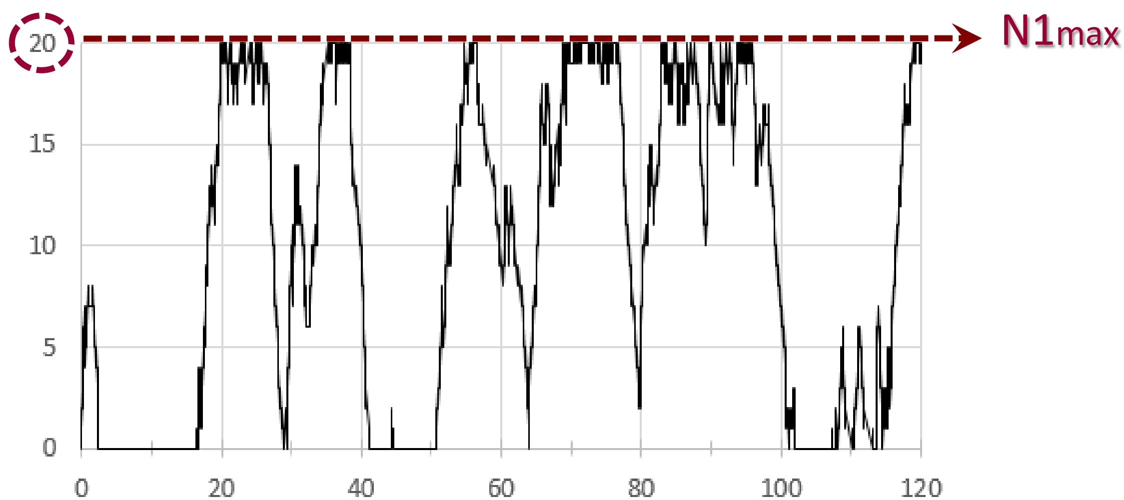
1 request sent to Server 2 (when enabled)

HPC cluster: behavior of core allocation ($N1_{max}$=20)

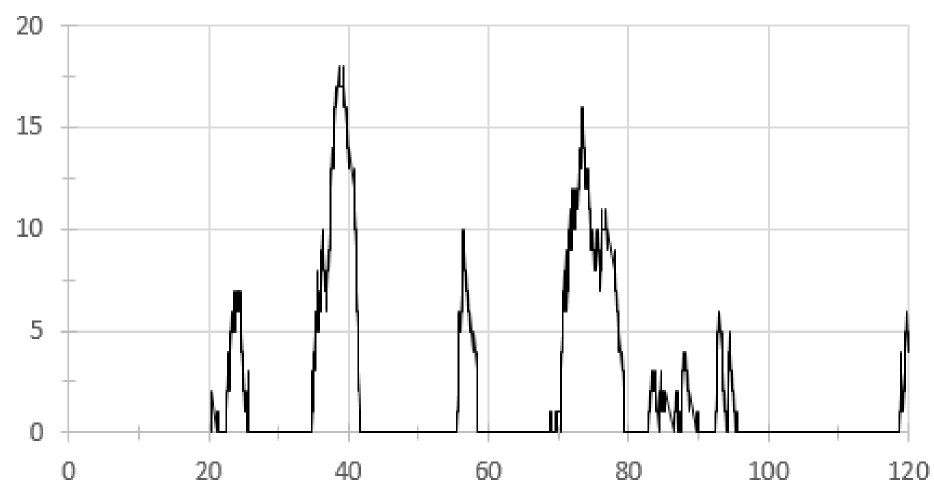
arriving requests



cores allocated of Server 1

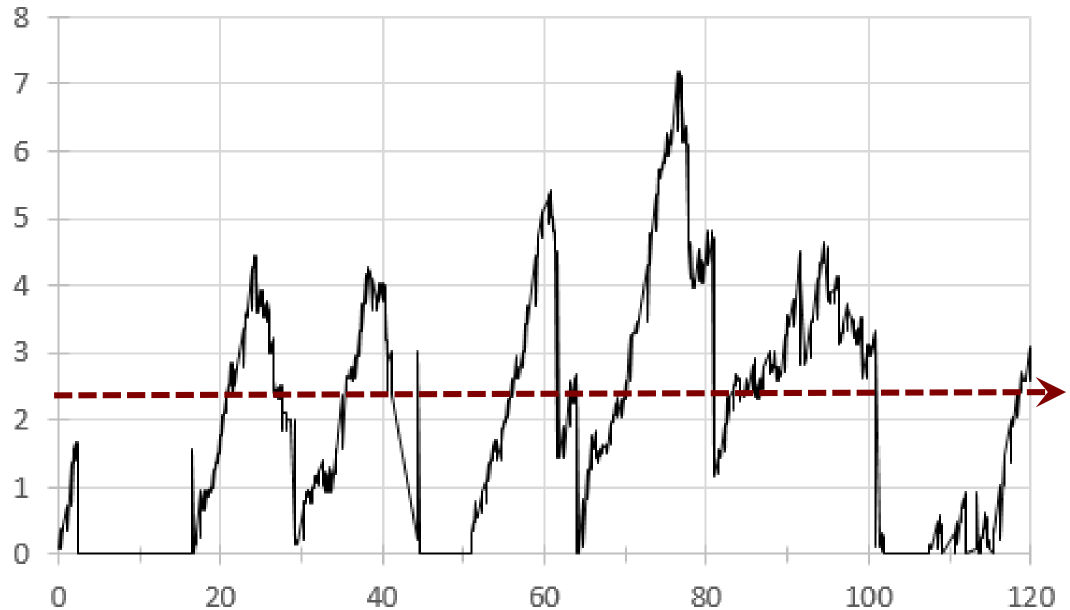


cores allocated of Server 2



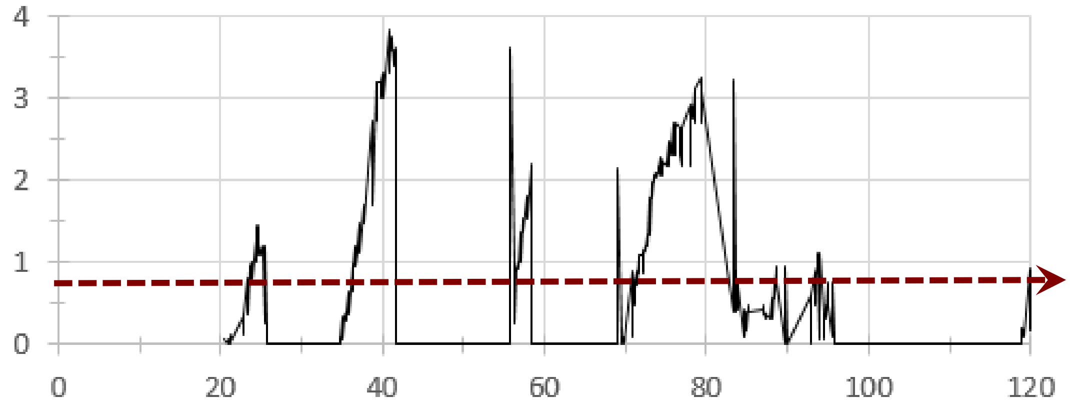
HPC cluster: behavior of response times ($N1_{max}=20$)

Server 1



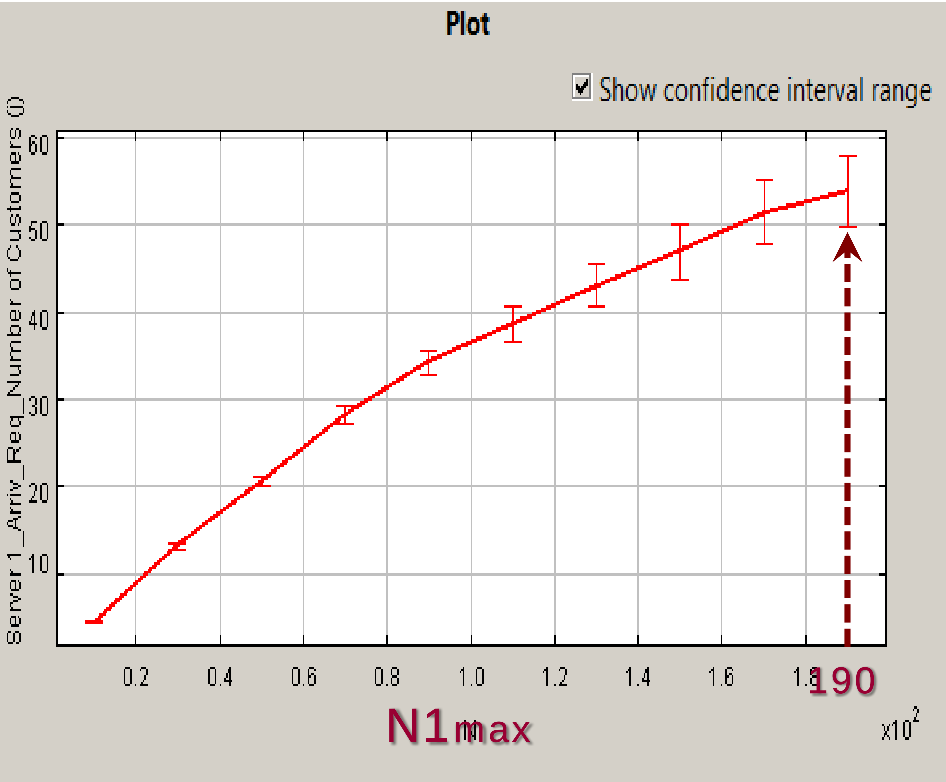
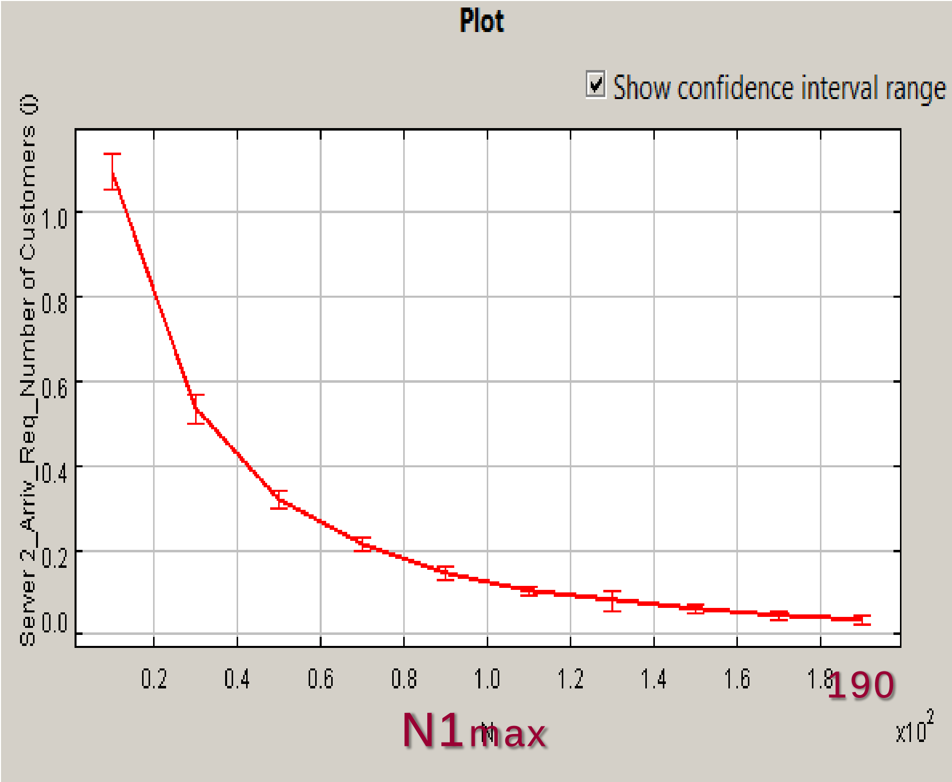
mean $R_{Server1}$ = 2.20 s

Server 2



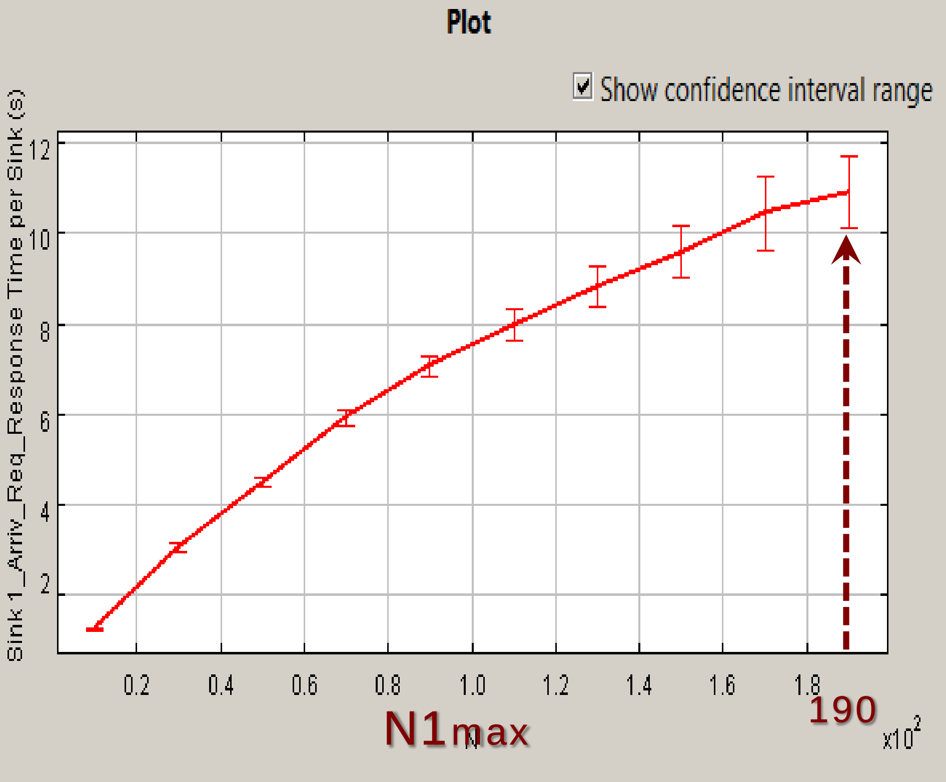
mean $R_{Server2}$ = 0.78 s

HPC cluster: mean no. of cores allocated *vs* N1max

Server 1	*Server 2*
	
max number of cores available on Server 1	*max number of cores available on Server 1*

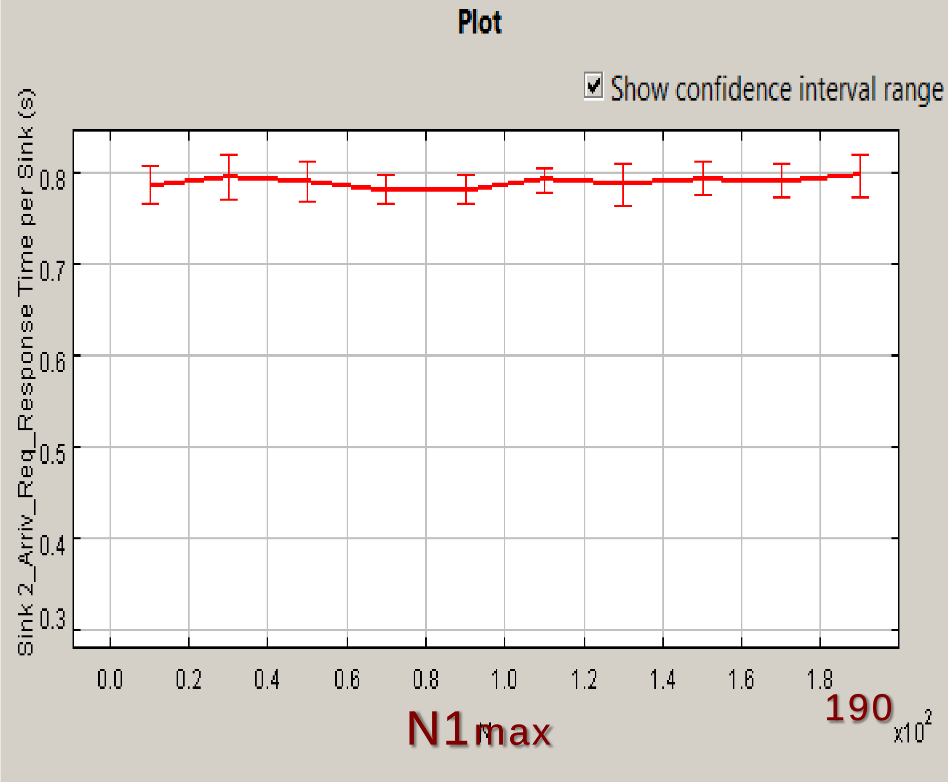
HPC cluster: response times of Server1 – Server2 *vs* N1max

Server 1



max number of cores available on Server 1

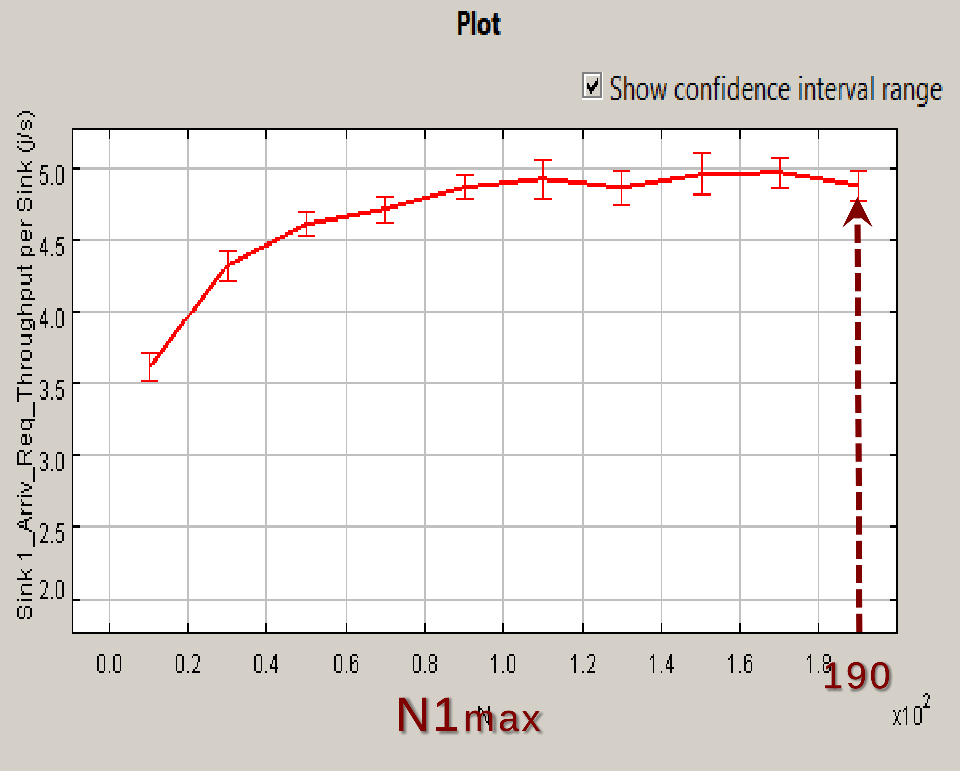
Server 2



max number of cores available on Server 1

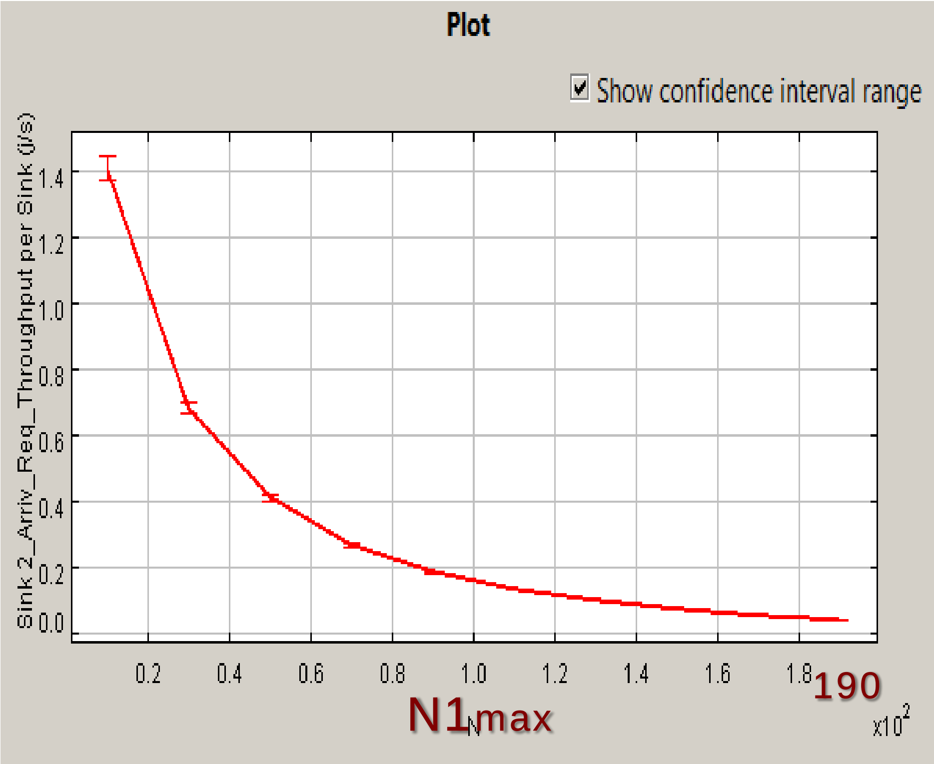
HPC cluster: throughput of Server1 – Server2 *vs* N1max

Server 1



max number of cores available on Server 1

Server 2



max number of cores available on Server 1

Adaptive Car Navigation Service for Smart Cities

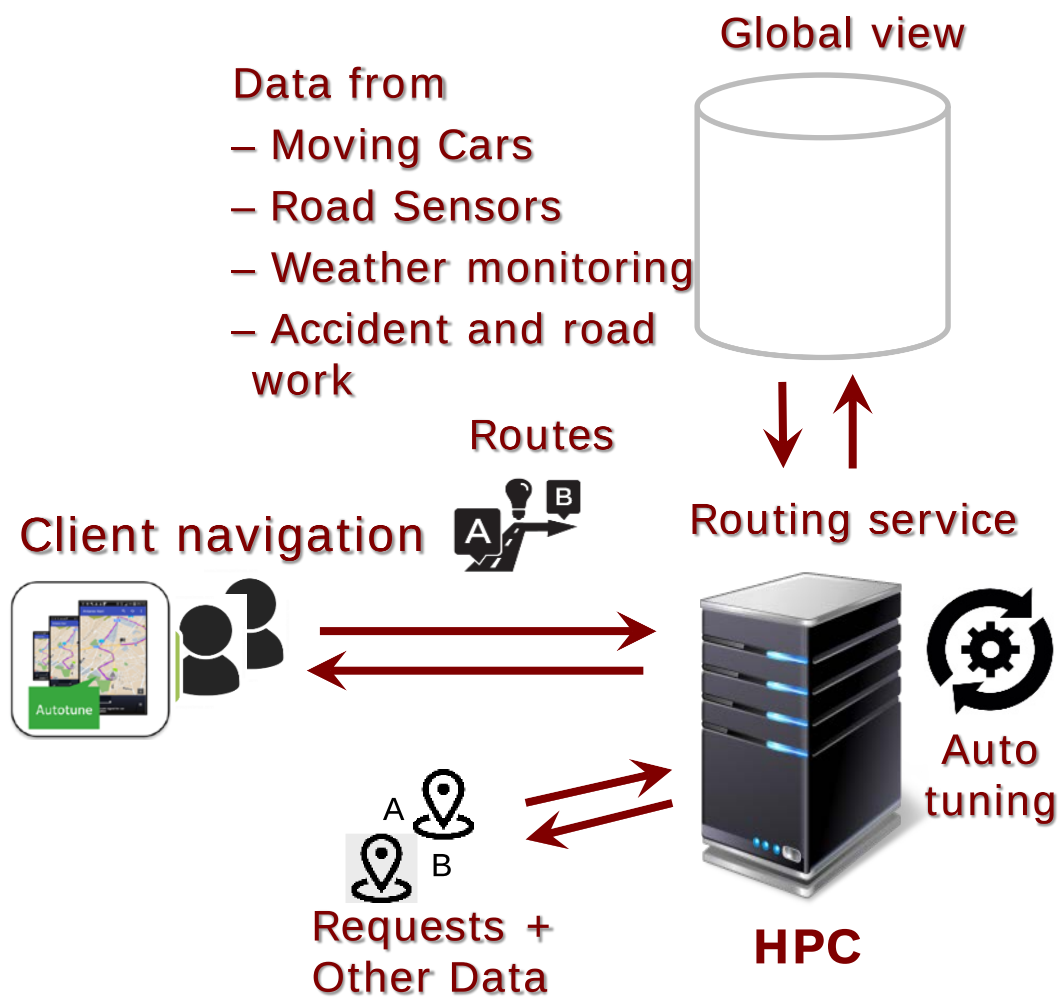
L.Arcari, M.Gribaudo, G.Palermo, G.Serazzi, Performance Driven Analysis for an Adaptive Car Navigation Service on HPC Systems, ISCIS19, Paris, 2019, EU Project 671623 FET-HPC-ANTAREX

- provide optimal routes to hundred thousands of drivers operating in the city area

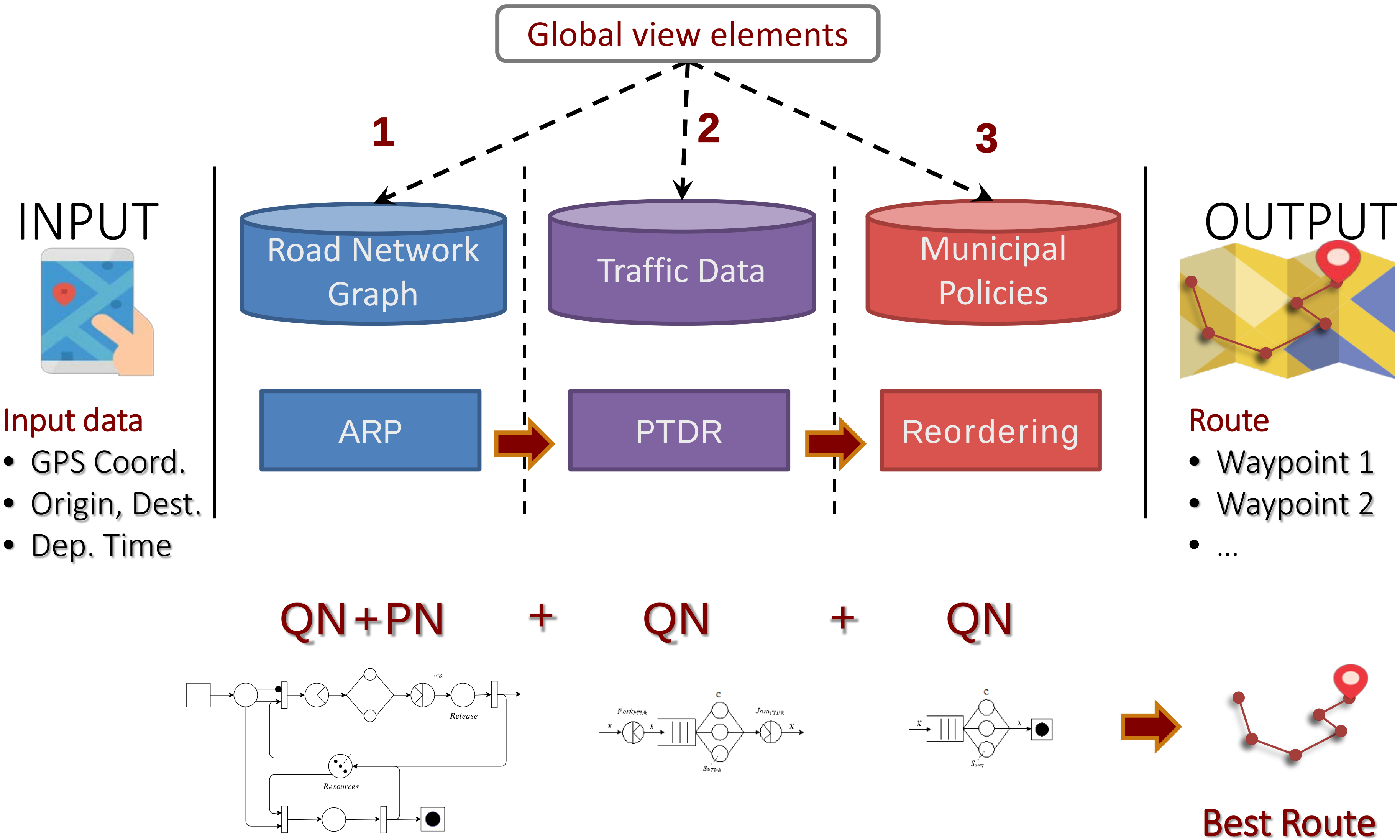
Requirements:

- intelligent routing based on calculation of traffic view state
- traffic global view calculation & optimization based on data fusion from several sources
- different layers of adaptivity

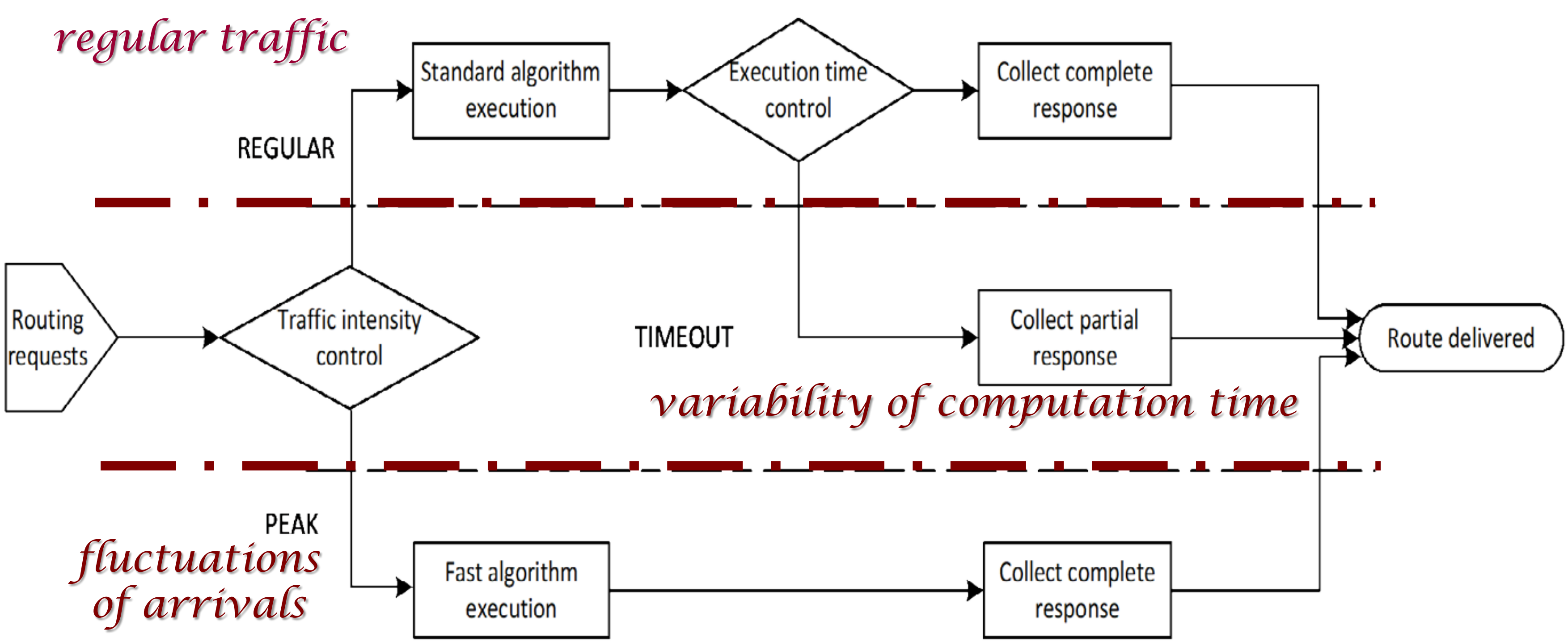
GOAL: evaluate the **optimal number of cores** of the HPC infrastructure to achieve the target performance of **R≤1 sec**.



Car Navig. Serv.: three stages of the application

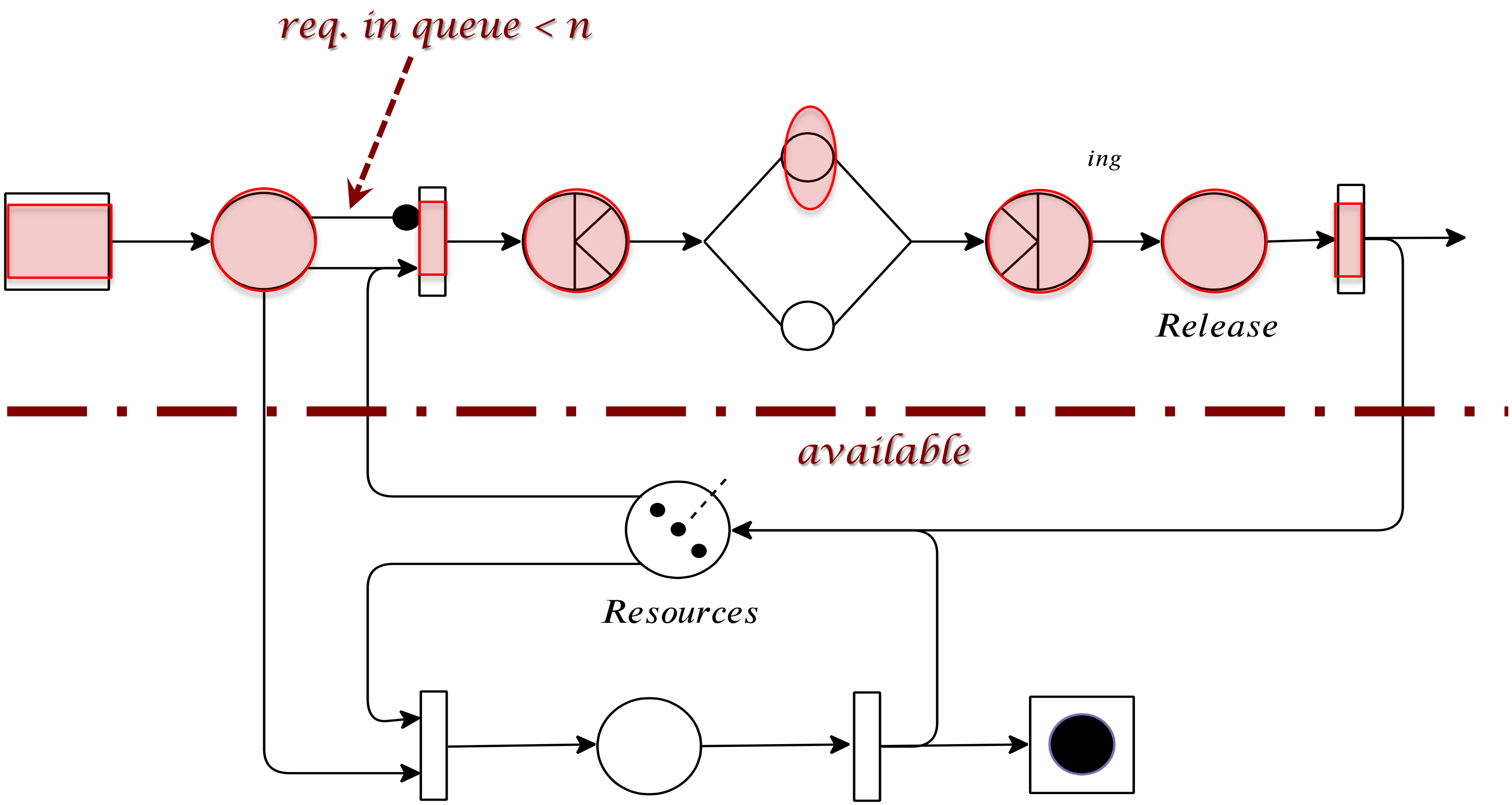


Car Navig. Serv.: Adaptivity controls

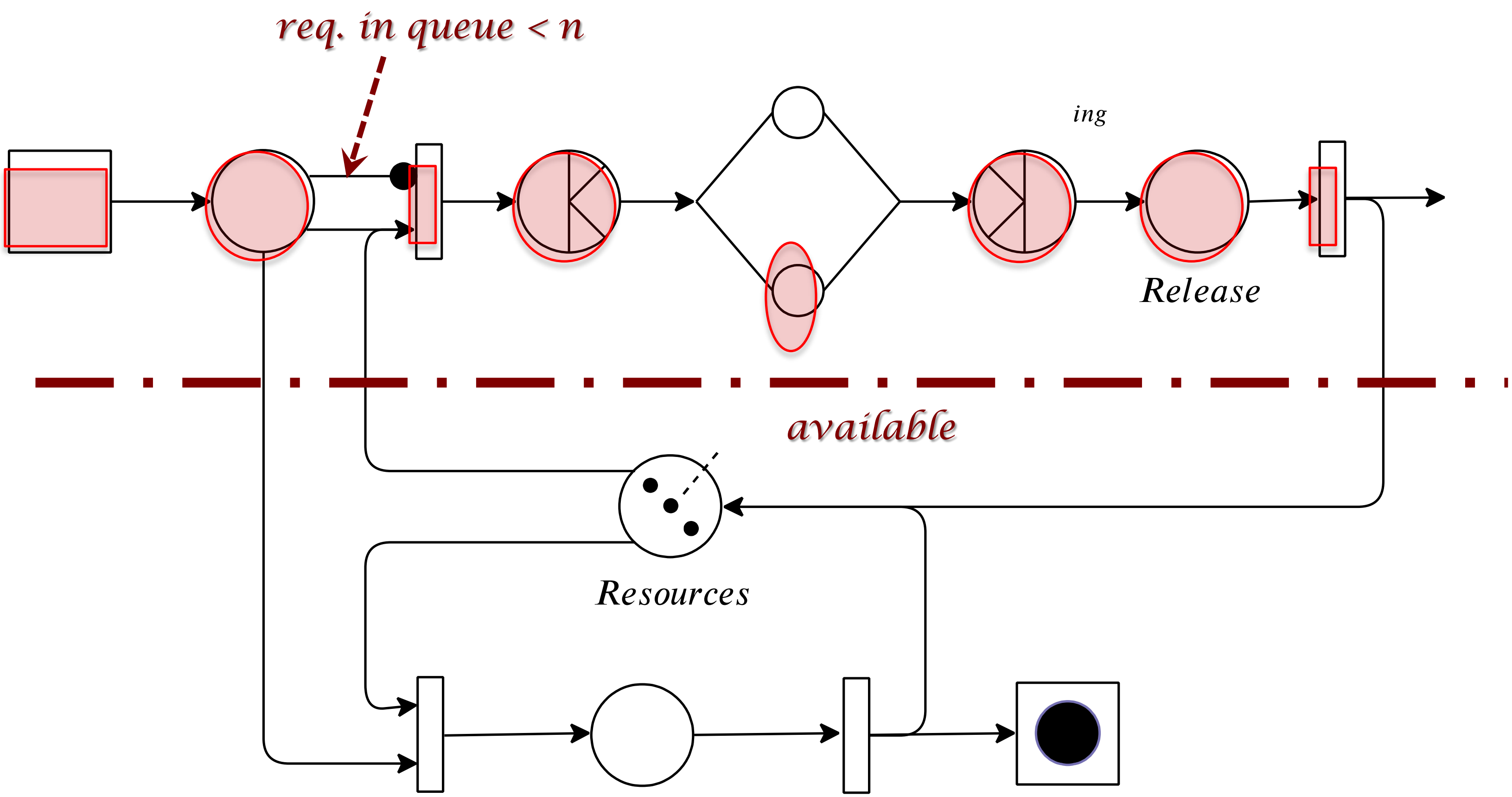


- the application may apply **sub-optimal** solutions when the system is **overloaded**, or when the computation time of a request is **too long**

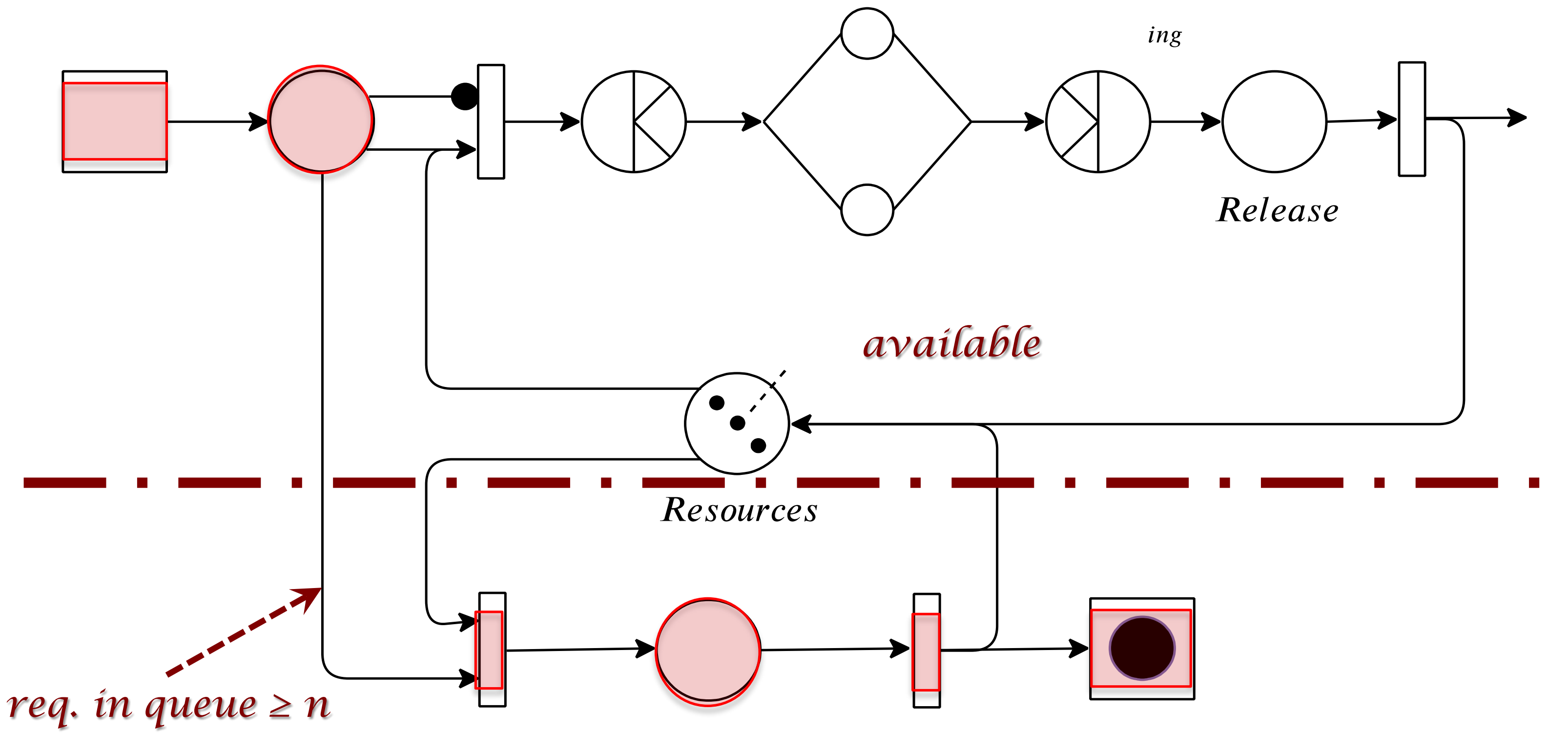
Car Navig. Serv.: ARP stage – Regular service



Car Navig. Serv.: ARP stage – requests with timeout



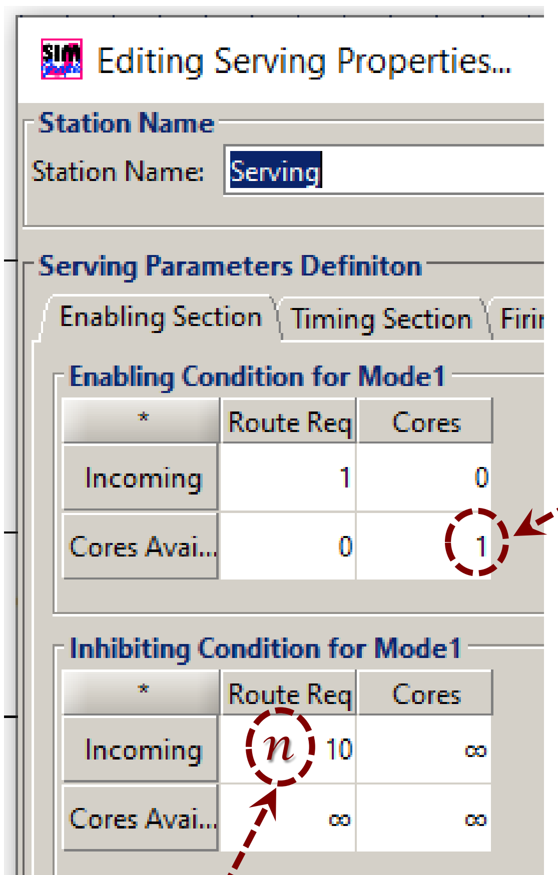
Car Navig. Serv.: Fast algorithm when overloaded



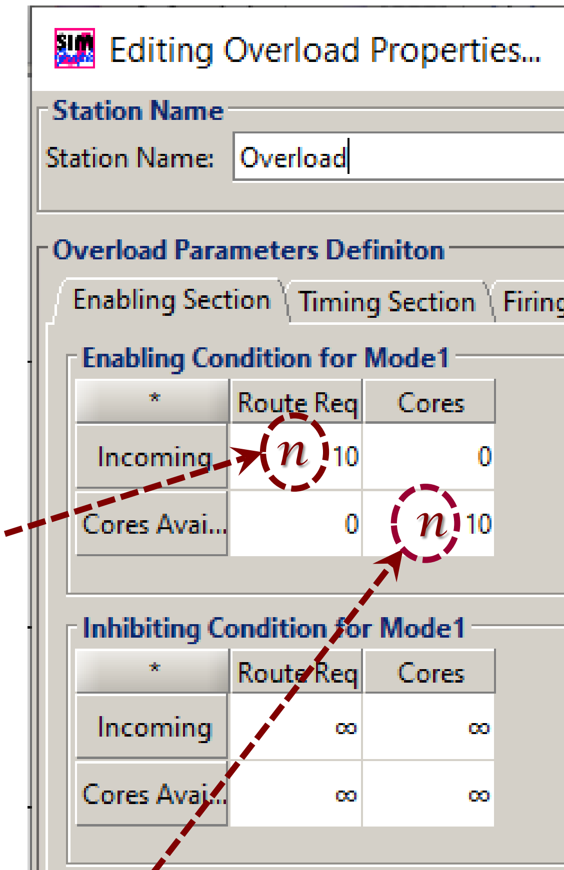
- **Fast lane**: system privileges speed over quality, giving a better performance at the expense of the effectiveness of the suggested routes

Car Navig. Serv.: transitions enab/inhib conditions

Serving



Overload



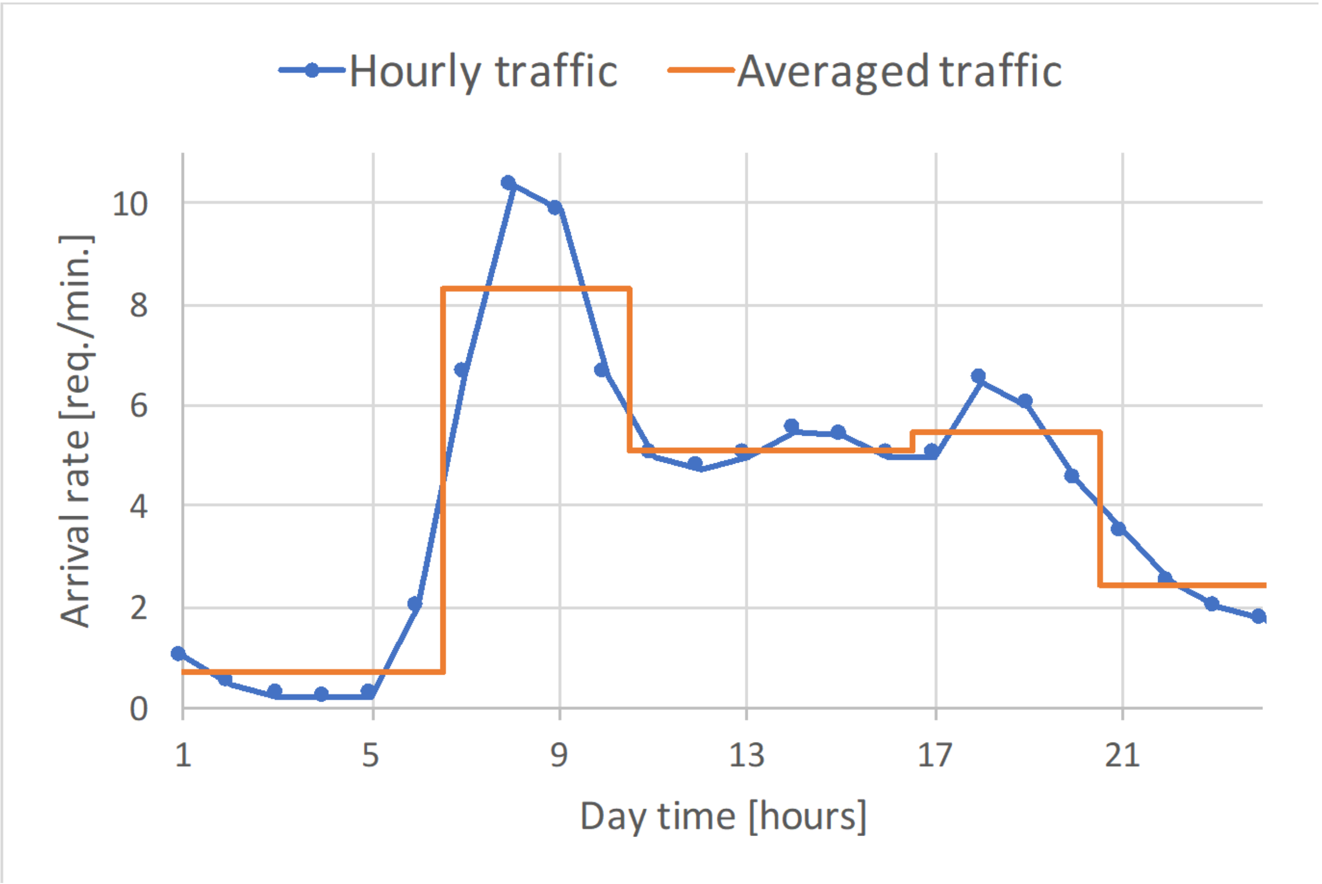
at least 1 core available

enabling when requests in queue are ≥n

block when requests in queue are ≥n

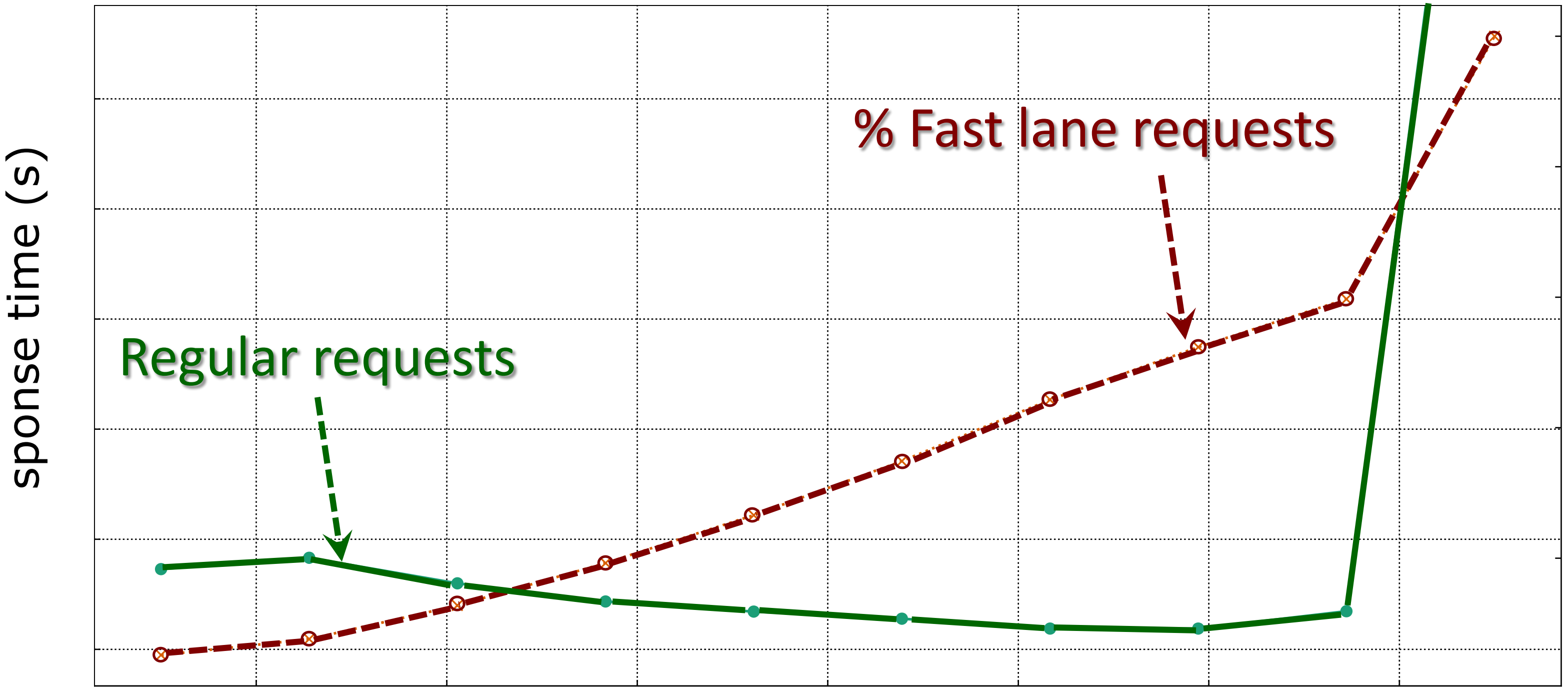
at least n core available

Car Navig. Serv.: workload



- routing requests traffic in Milano Urban Area (1 day)
  2.5 Million trips per day

Car Navig. Serv.: Regular and Fast lane response time



Car Navig. Serv.: some Results

Time Slot	ARP	PTDR	Reordering	Total cores
01 - 06	51	10	1	**62**
07 - 10	584	111	2	**697**
11 - 16	373	71	1	**445**
17 - 20	393	75	1	**469**
21 - 24	200	38	1	**239**

What's Next

What's next

- Impatient customers (debugged)
- Parallel What-if (debugged)
- What-if – new control parameters
- New scheduling algorithms
- Blackbox modeling with predefined targets
  - With multiclass workload find the optimal load that minimize Response time or maximize throughput, Energy saving in a cloud, adaptive allocation of Virtual Machines, ...
- ... ...