QCon SAN FRANCISCO

Scaling Up Performance Benchmarking

-with SPECjbb2015

Anil Kumar –
Runtime Performance Architect @Intel,
OSG Java Chair

Monica Beckwith –
Runtime Performance Architect @Arm,
Java Champion

Auto scaling

FaaS

Serverless

Frameworks

Microservices

Scalable thread pool like Fork-Join

Containers / VMs

Elastic Cloud

kubernetes

Scalable processors

* All trademarks are the property of their respective owners

Performance Engineering

1970 - 2020

Performance at scale

Scaling out a “poor single node” performance is waste of $$$$$!

☹

☹ ☹

☹ ☹ ☹

☹ ☹ ☹ ☹

Scaling out an “optimal single node” performance requires coordination like an orchestra !

☺

At scale, even 1% gain worth $$$$$!

Agenda

Not being covered today:

FaaS (Function as a Service) or Serverless or Microservices etc.

Being covered:

Modelling a complex backend of e-commerce enterprise (5 minutes)

Scaling from the beginning (5 minutes)

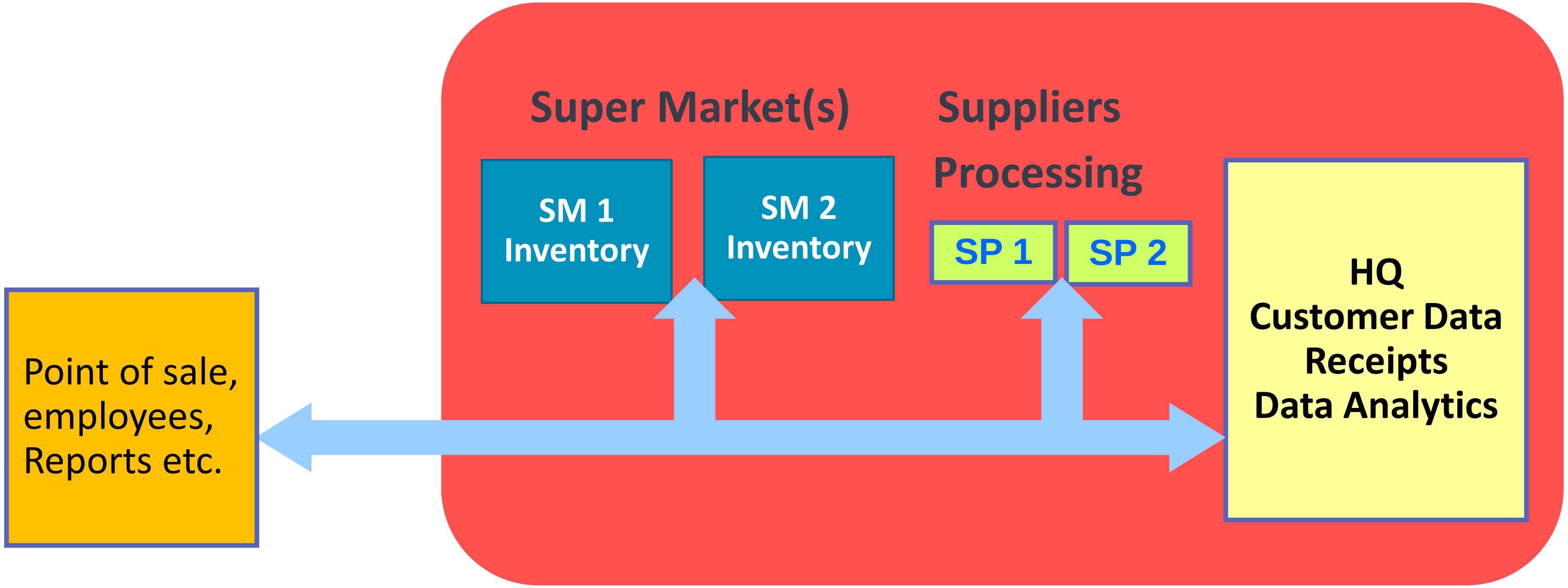
Architecture

Telemetry / observation points and metrics

Interesting data from scale up and scale out (15 minutes)

Take away(s) (5 minutes)

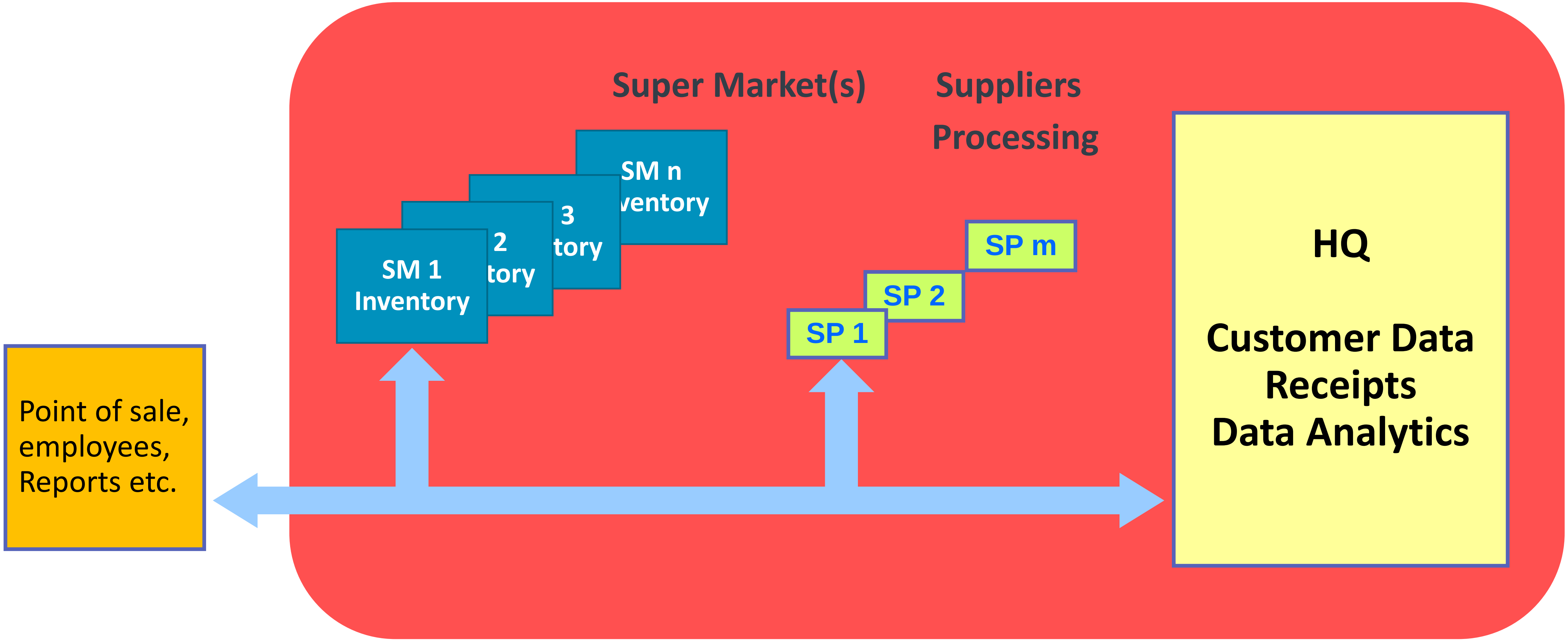
Architecture: Modelling backend of e-commerce enterprise



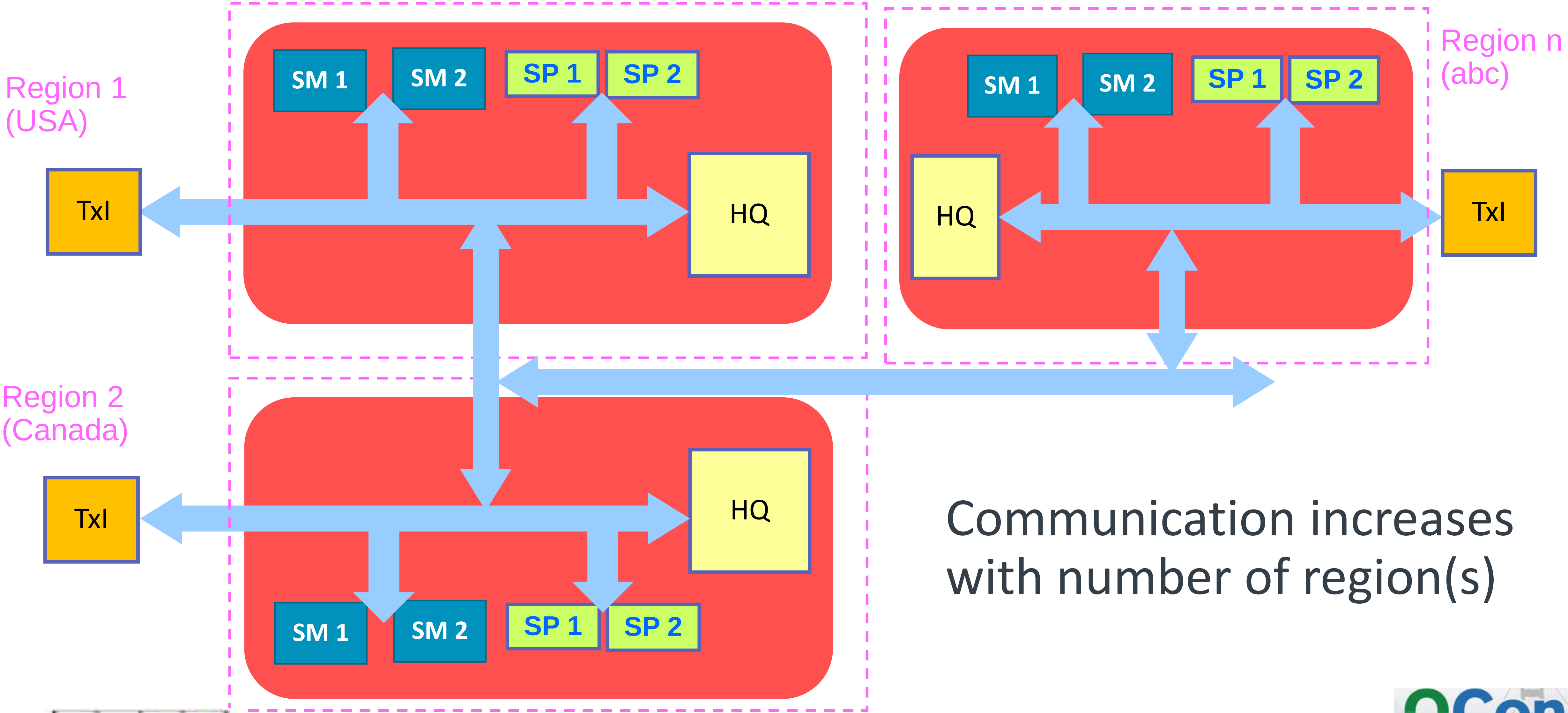
Flexibility of modules like SM, SP and different type of data

Scale up

More modules like SM, SP, inventory, user data etc.



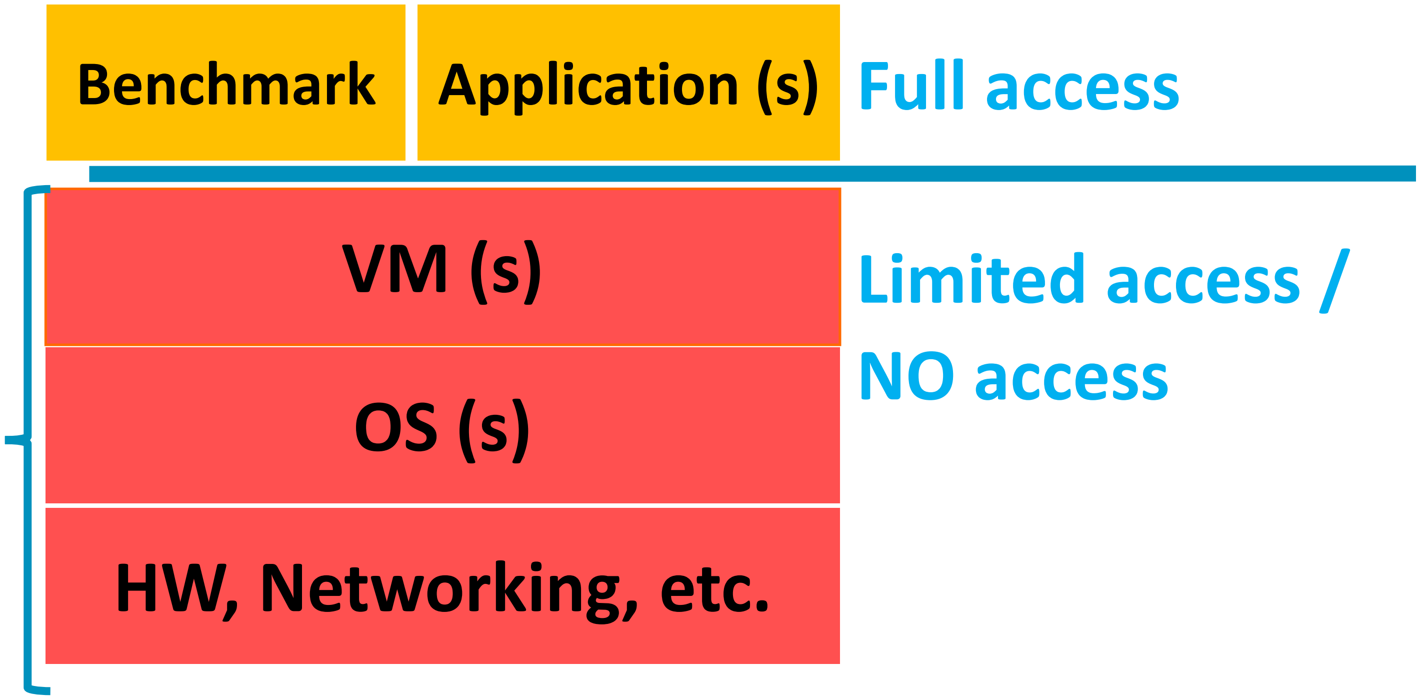
Scale out



Communication increases with number of region(s)

Why should anyone care ?

Expected benchmark behavior can help in performance estimation / debug / scaling



Benchmark

Application (s)

Full access

VM (s)

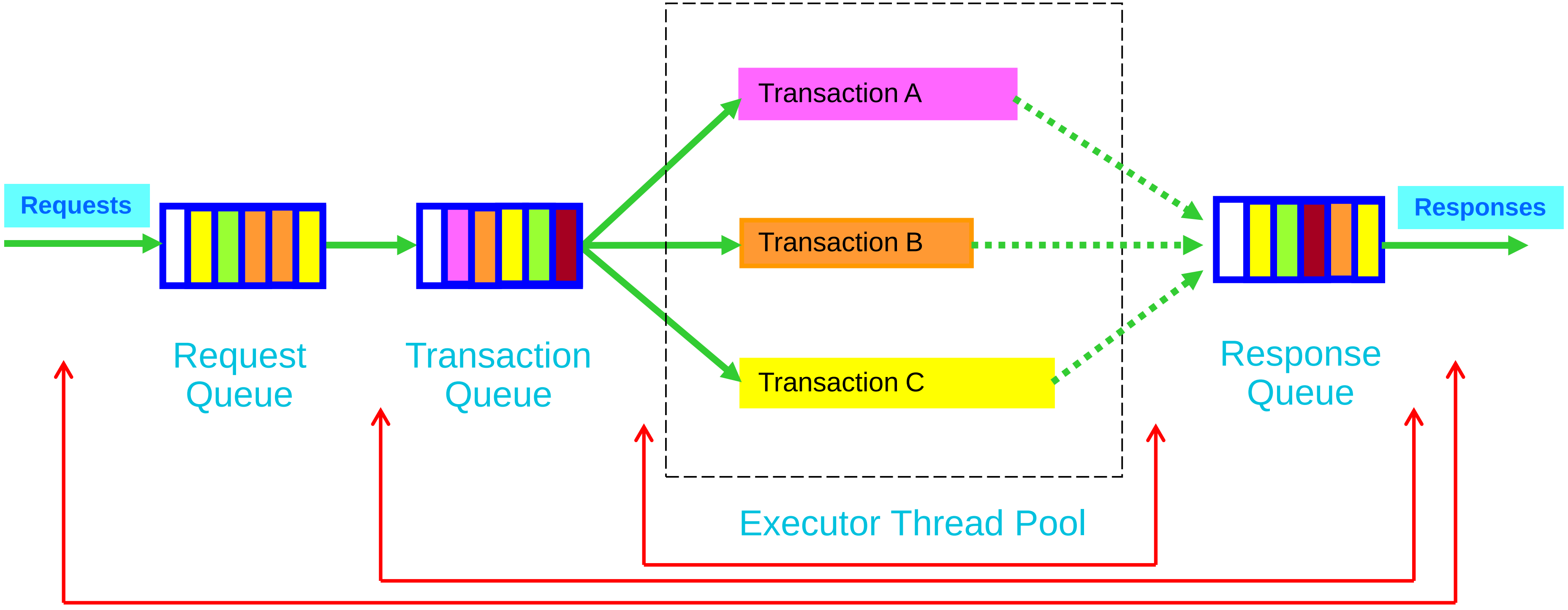
OS (s)

HW, Networking, etc.

Limited access / NO access

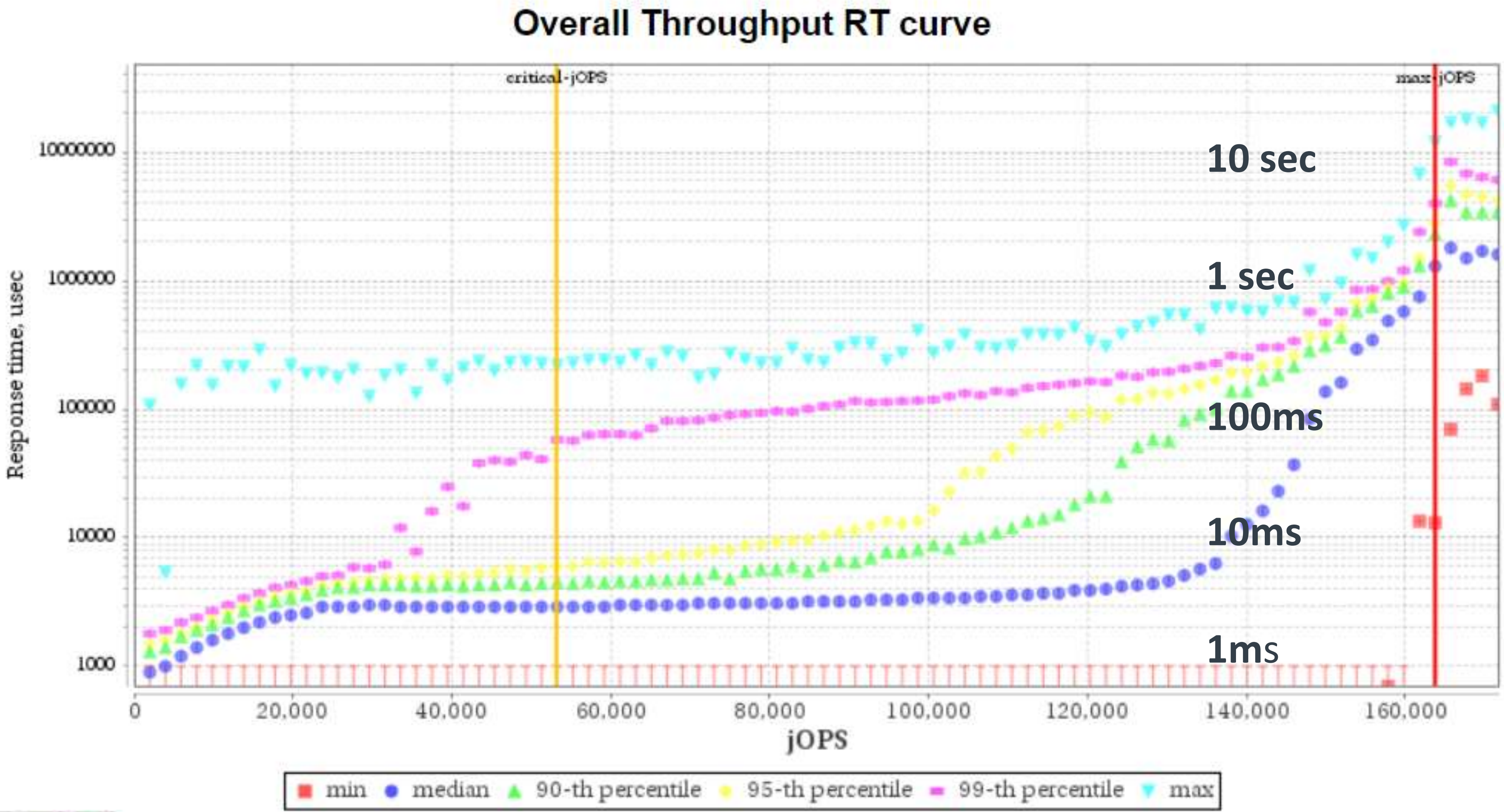
Measuring response time

Be sure what you're measuring *is* the response time you're interested in

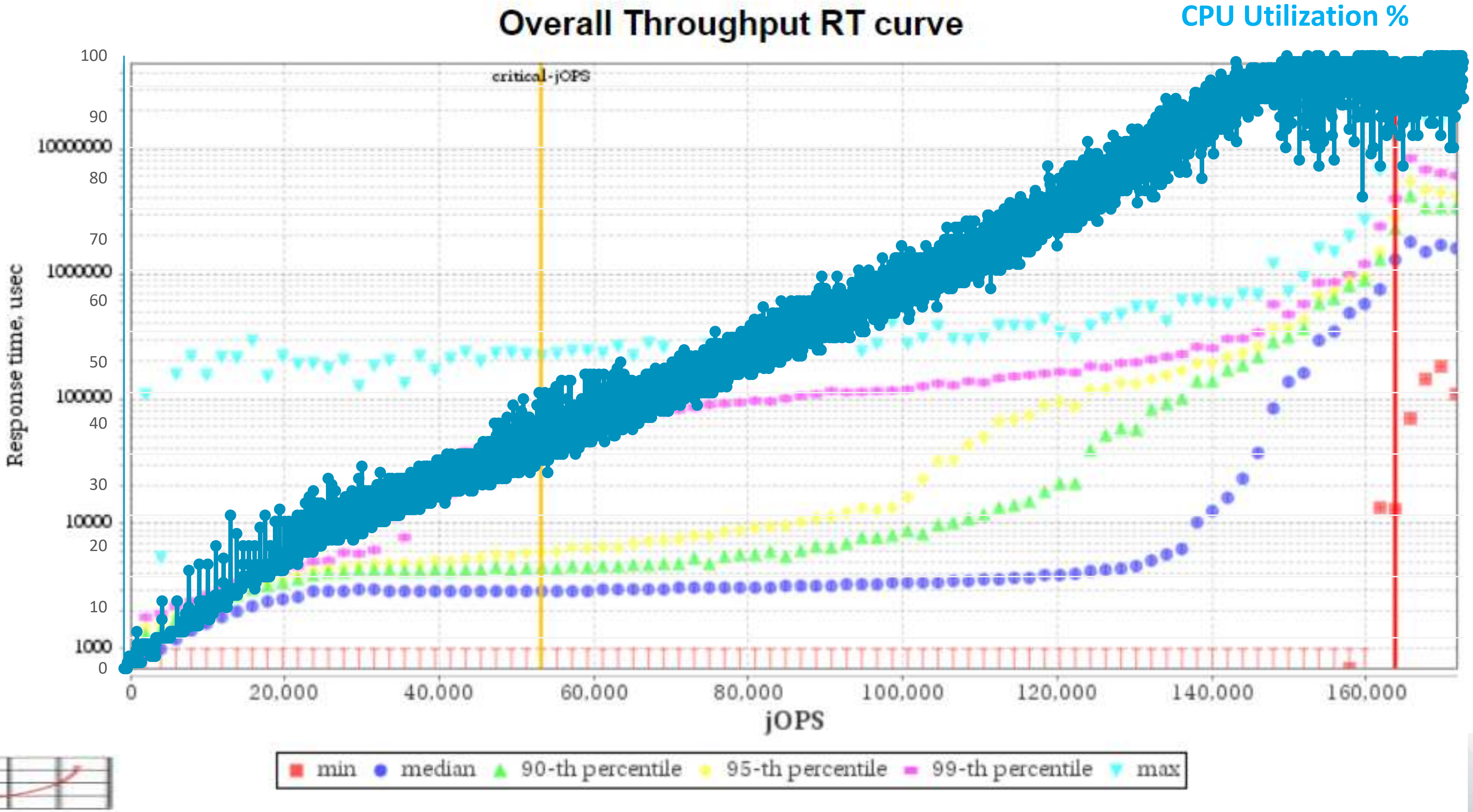


Measuring response time from request made to response received?

One typical run



CPU % utilization as load increases



HW infrastructure focus

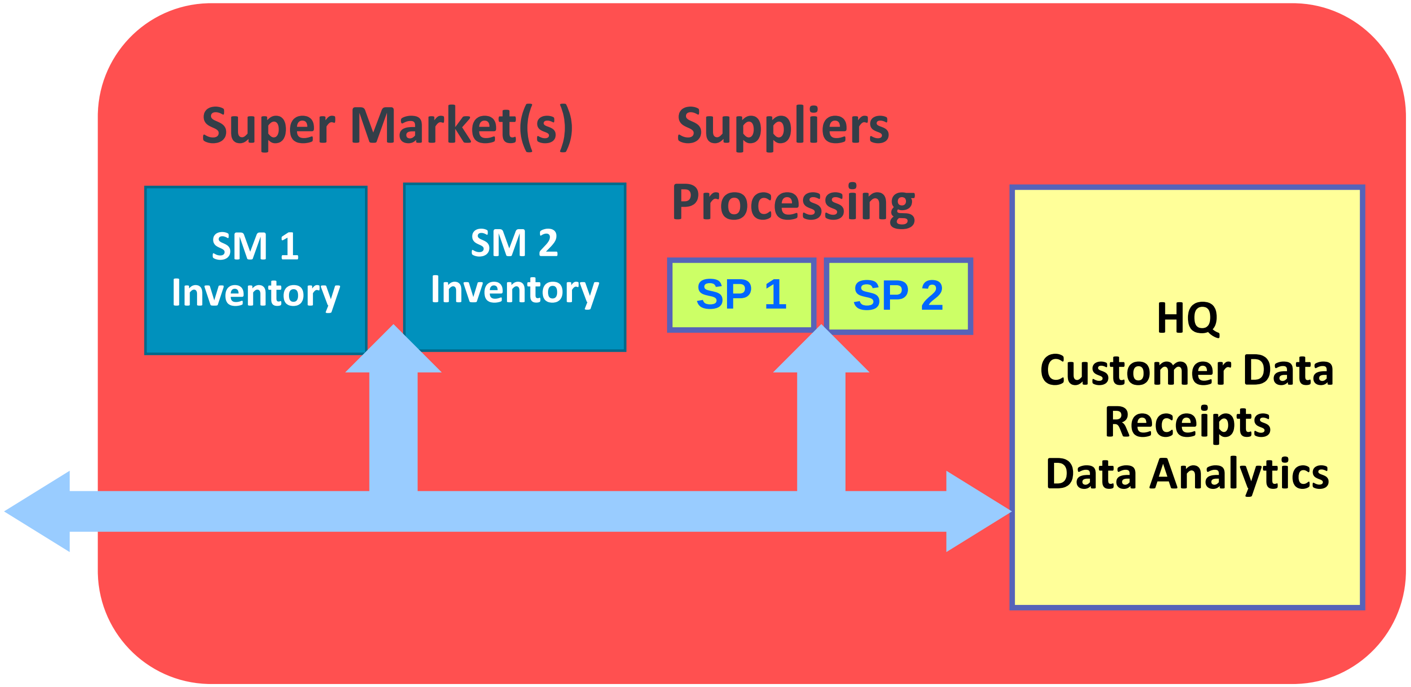
✓ Stand alone or #blade servers with network

✓ Racks or #blade servers with high bandwidth network

✓ ### of CPUs / memory as SINGLE OS image

X #blades with offload to GPU, FPGA etc. (Local or Shared)

SW architecture for scaling

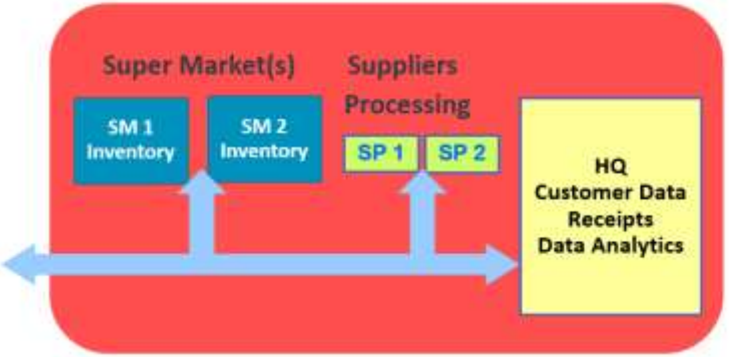


- Modules
- Thread pools
- Queues
- Data structures
- Communication
- Telemetry and Metrics

All modules can be deployed within one instances or separately !

Scale up: modules and thread pools

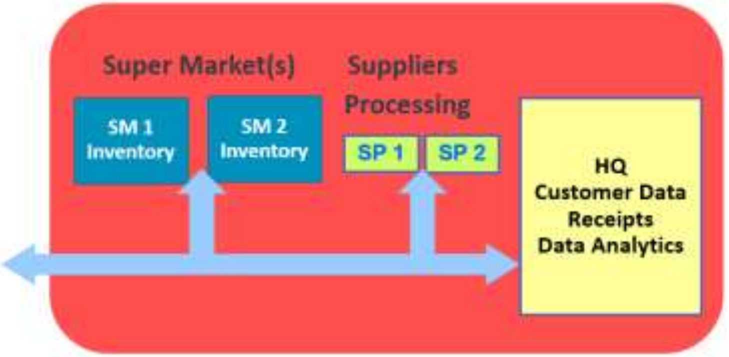
- Cost of modularity similar to microservices,
  - Serialization / deserialization
  - Data sharing



- Fork-Join thread pools
  - Auto scaling with bounded values

Scale up: queues and data structures

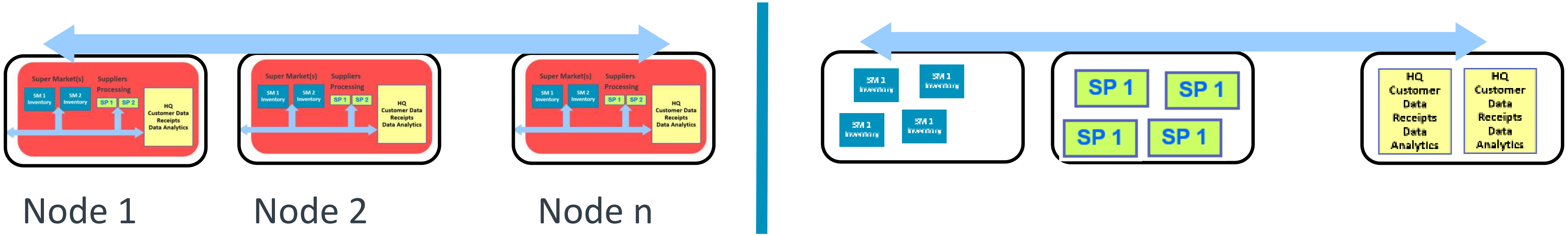
- Queue design very critical
  - Different type of requests in separate queues
  - Important messages not waiting in long queues



- Data structures
  - Concurrency with scaling important at high throughput

Scale out: scale up + communication + telemetry

- Telemetry and efficient aggregation
- Low latency and high bandwidth communication
- Node topology deployment strategy



Problem Statement

Scaling Up a System is Not Easy …

So What Do We Do?

Scale Out!

~~Divide~~ Distribute and conquer!

Advantages:

- Cheaper commodity hardware
- Deploy Nodes/VMs/Containers/Infrastructure as needed

Potential issues to consider – orchestration/networking!

2 Backends

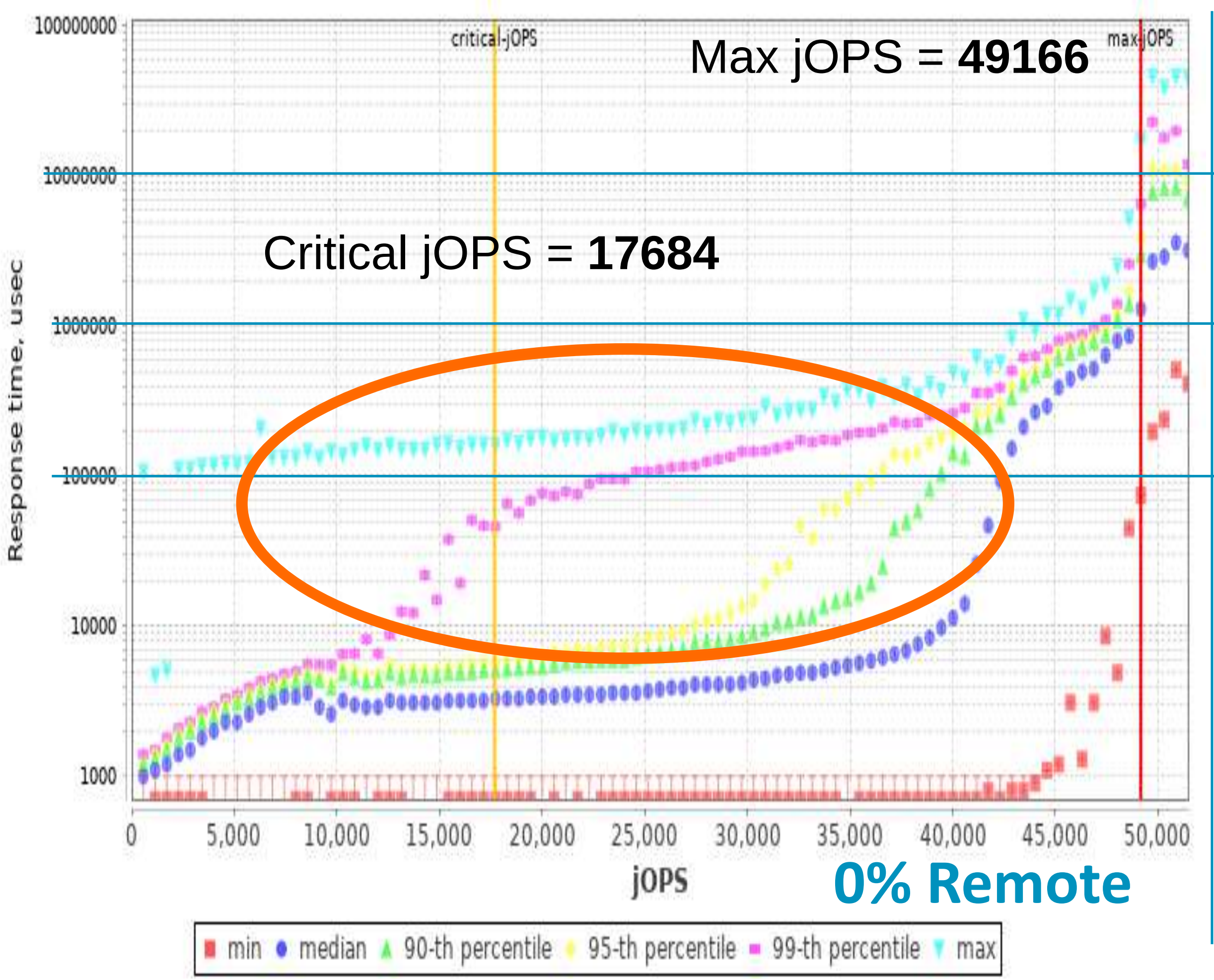
Networking Traffic:

0% Remote vs 50% Remote

specjbb.sm.replenish.localPercent	100 vs 50
specjbb.customer.RemoteCustomerShare	0.0 vs 0.2

Network Traffic Comparison

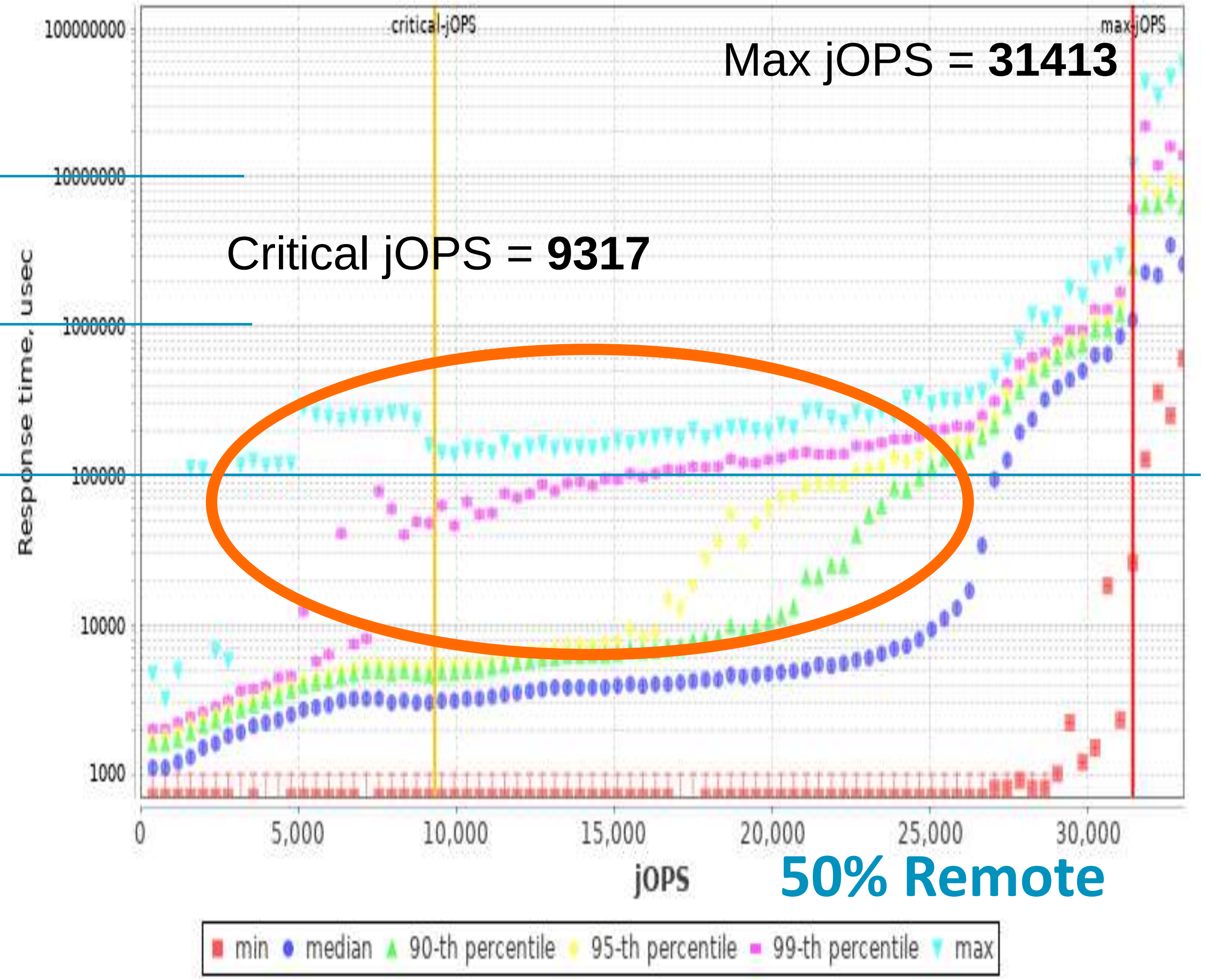
Remote traffic effects on SLA



Max jOPS = **49166**

Critical jOPS = **17684**

0% Remote



Max jOPS = **31413**

Critical jOPS = **9317**

50% Remote

Problem Statement

Scaling Up a System is Not Easy …

Problem Statement

Scaling Up a System is Not Easy …

Some brave-hearts still attempt it!

Why?	Approach?	How?
To increase injection rate/transactions/users/clients	HW	Add memory to provide more heap
	SW	Choose a different Garbage Collection algorithm
To optimize CPU cores/SMT usage	SW	Optimize task scheduler

Potential issues to consider – SLA constraints

Scenarios That We Will Cover Today

Scenarios	Why?	How?
Scenario 1	Increase injection rate	Increase heap
Scenario 2	Increase injection rate	Choose a different Garbage Collection algorithm
Scenario 3	Optimize CPU cores/SMT usage	Optimize task scheduler

Scenario 1

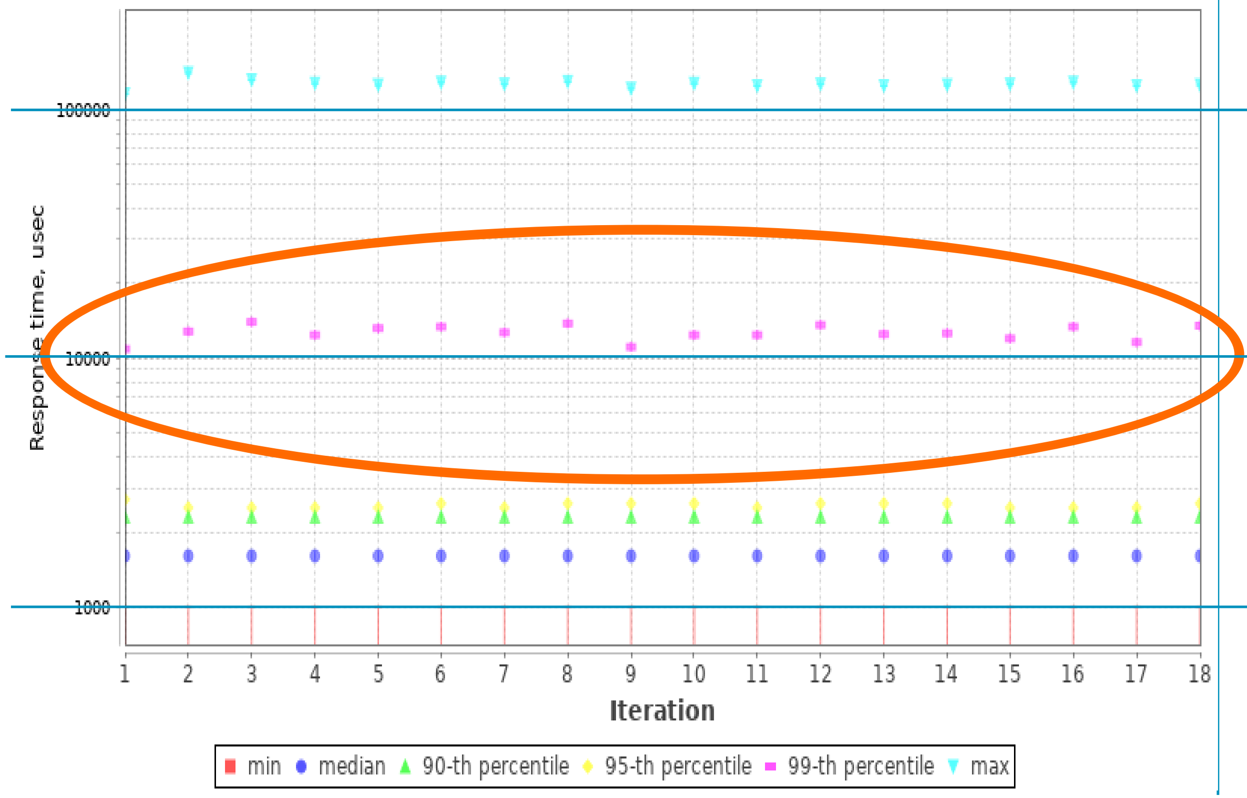
Increase Injection Rate and Heap Sizes

Check your SLA Constraints!

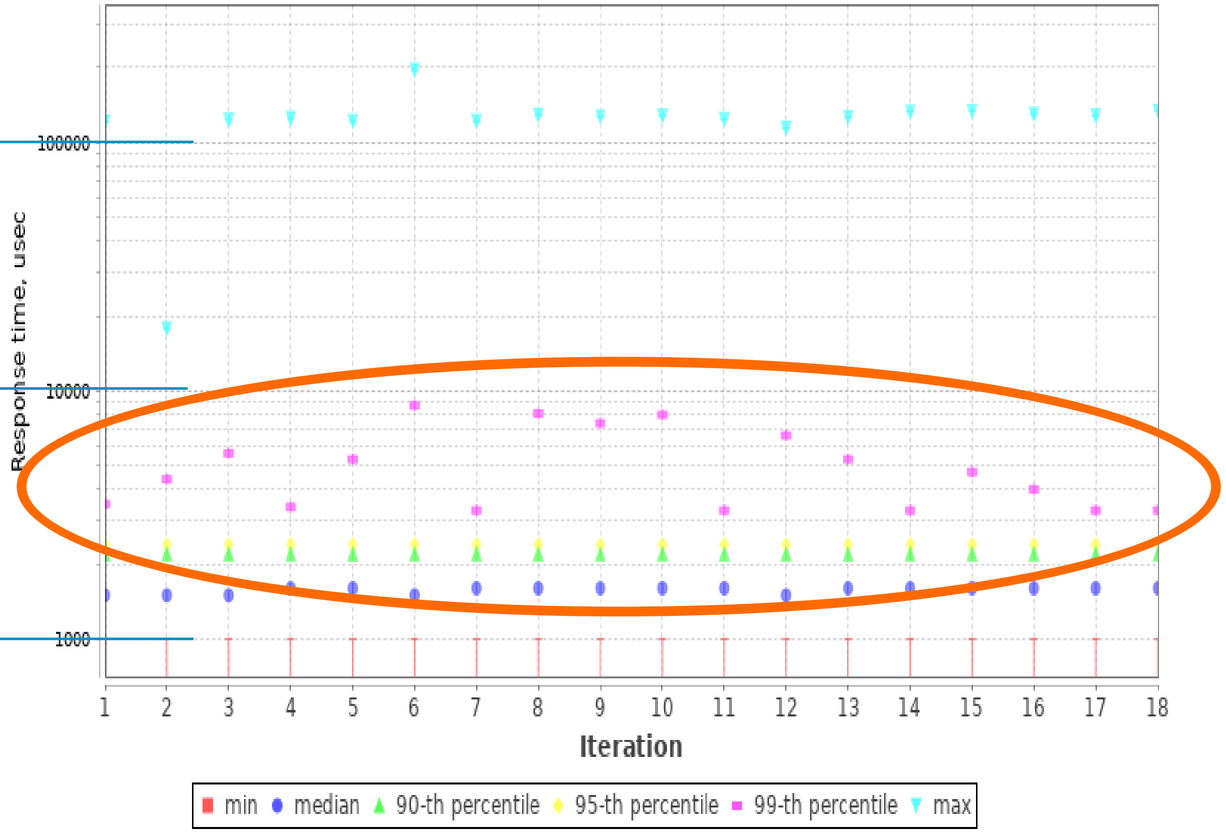
Heap Comparison: 10GB vs 30GB @ 10K Injection Rate

How heap size affects your SLAs

10GB



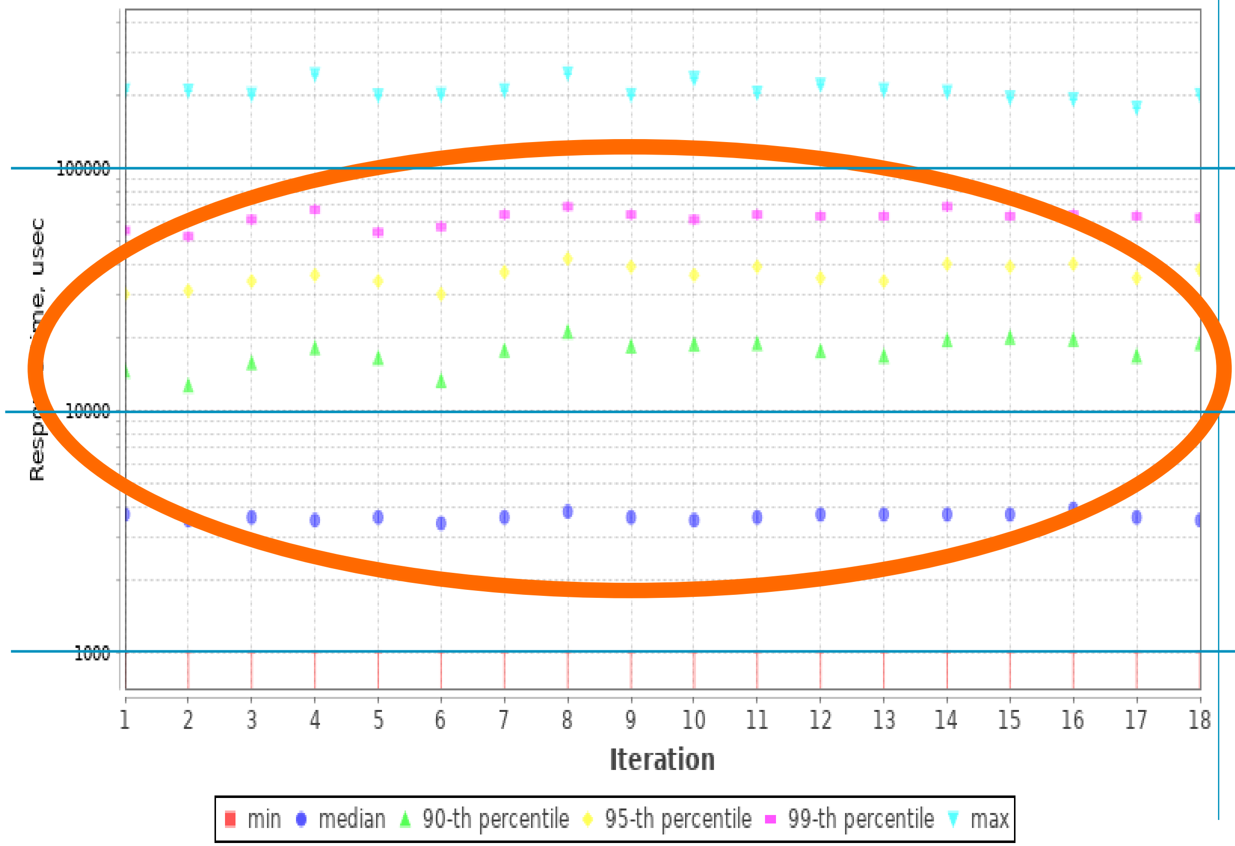
30GB



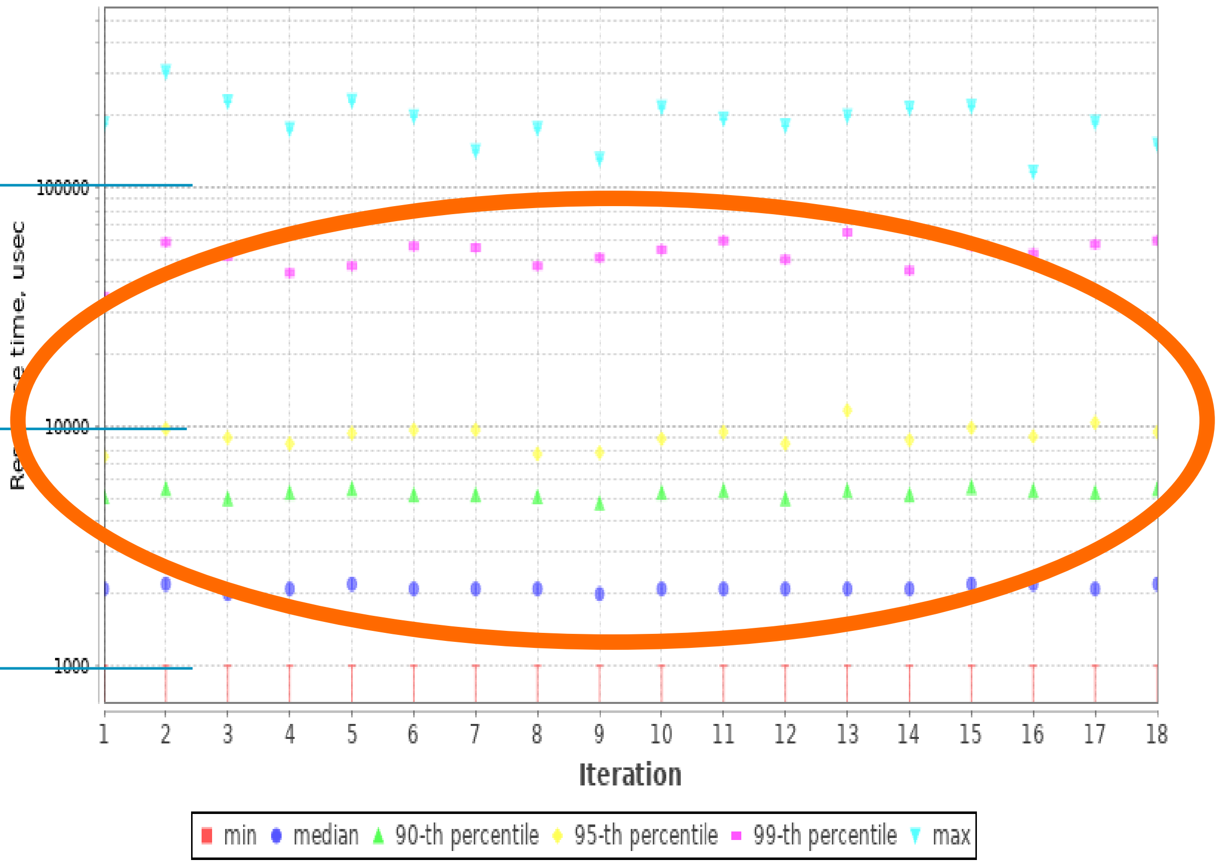
Heap Comparison: 10GB vs 30GB @ 30K Injection Rate

How heap size affects your SLAs

10GB



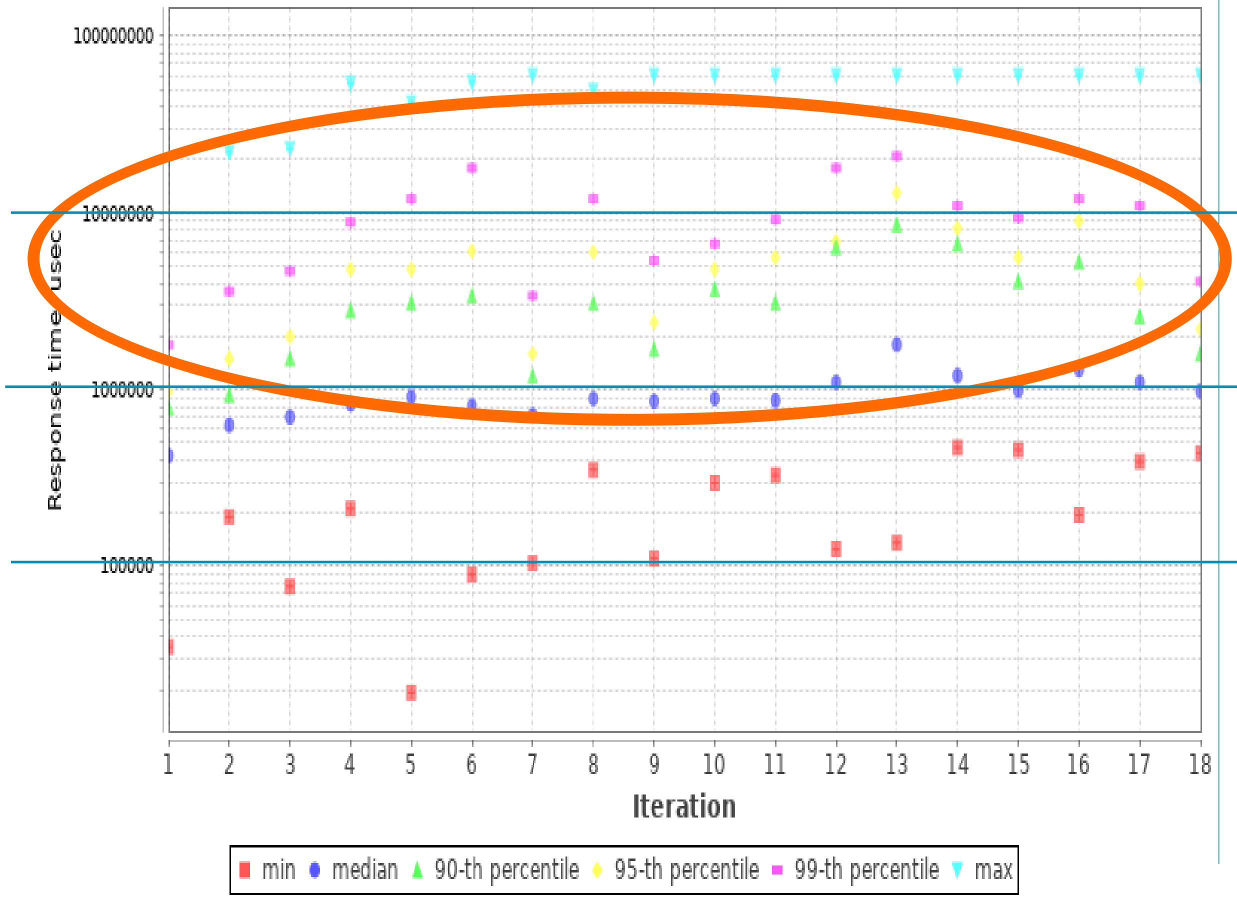
30GB



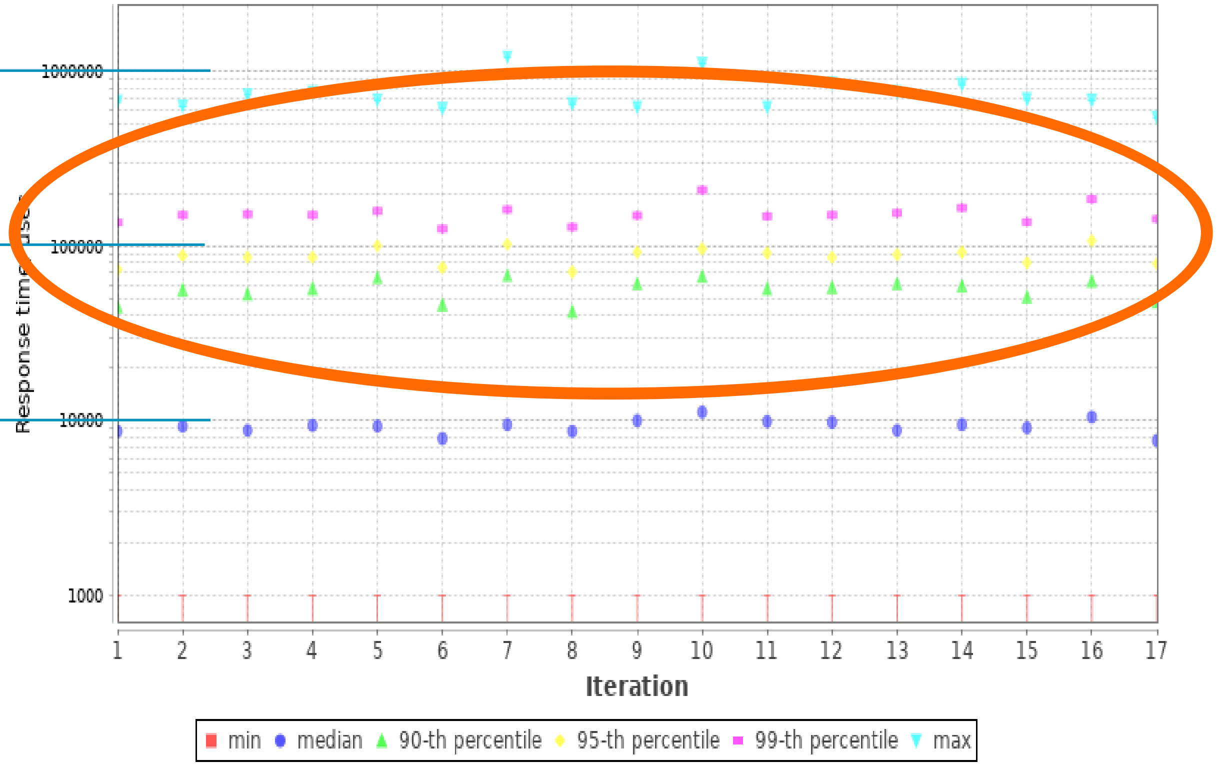
Heap Comparison: 10GB vs 30GB @ 50K Injection Rate

How heap size affects your SLAs

10GB



30GB



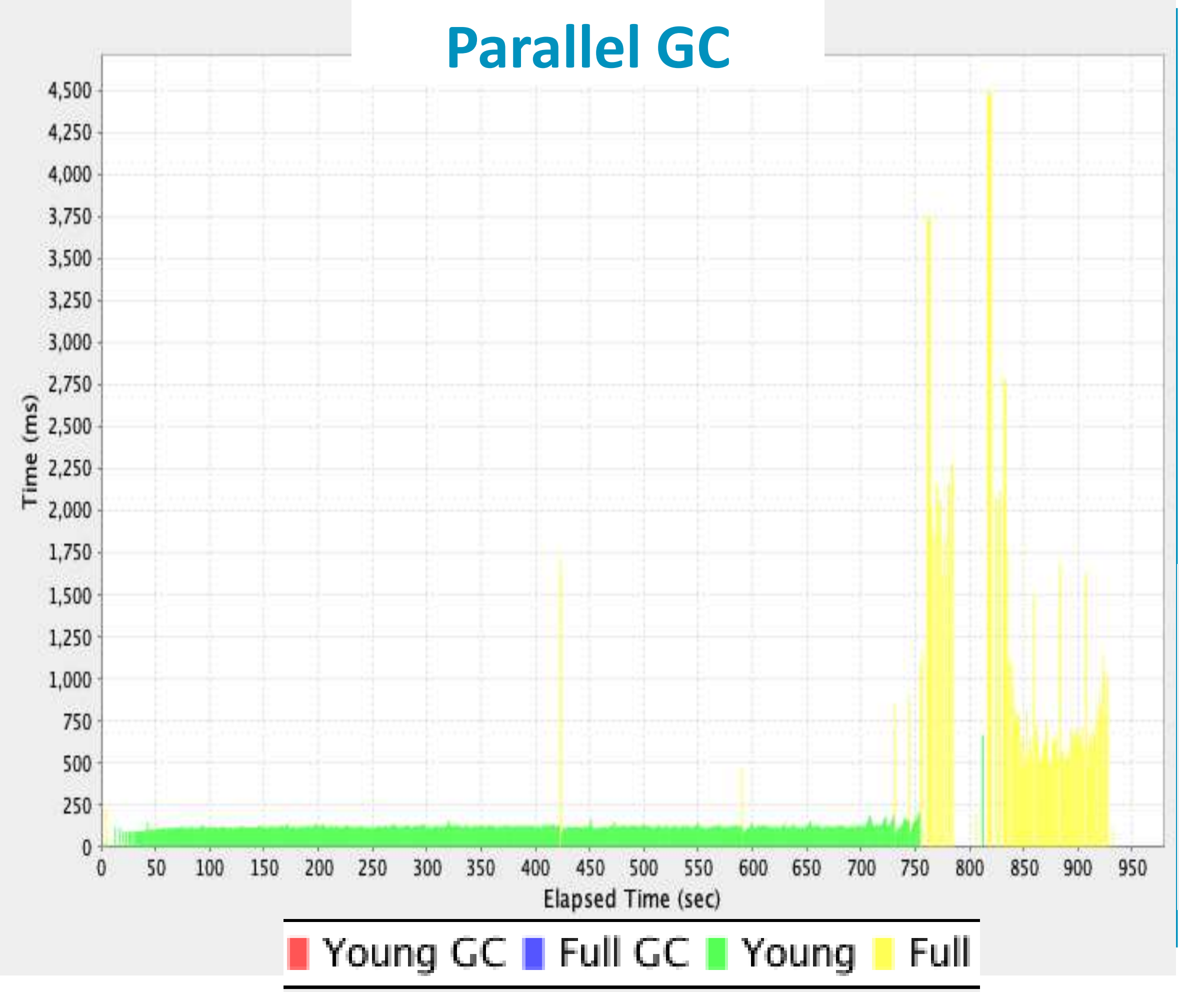
Scenario 2

Increase Injection Rate by Choosing a Better Suited GC Algorithm

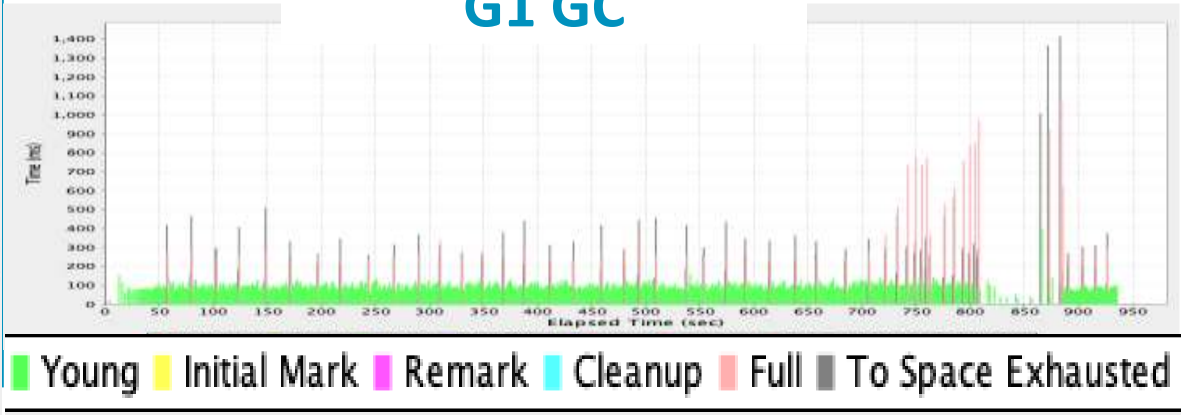
Check your SLA Constraints!

GC Comparison: @ 10GB Heap @ 50K Injection Rate

Comparing GC Pauses



Parallel GC



G1 GC

GC Comparison: @ 10GB Heap @ 50K Injection Rate

Comparing GC Overhead and Worst Case Pauses

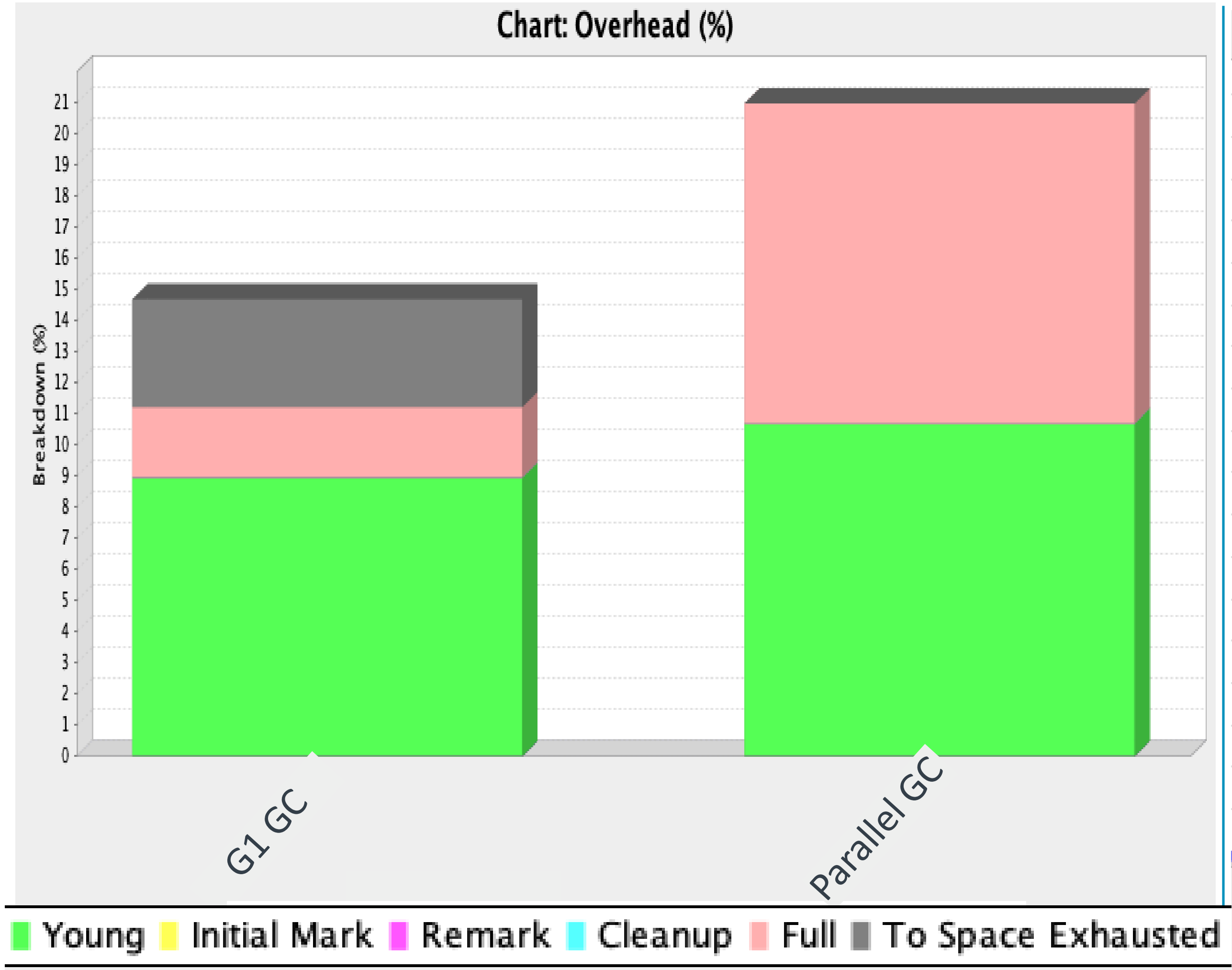


Chart: Overhead (%)

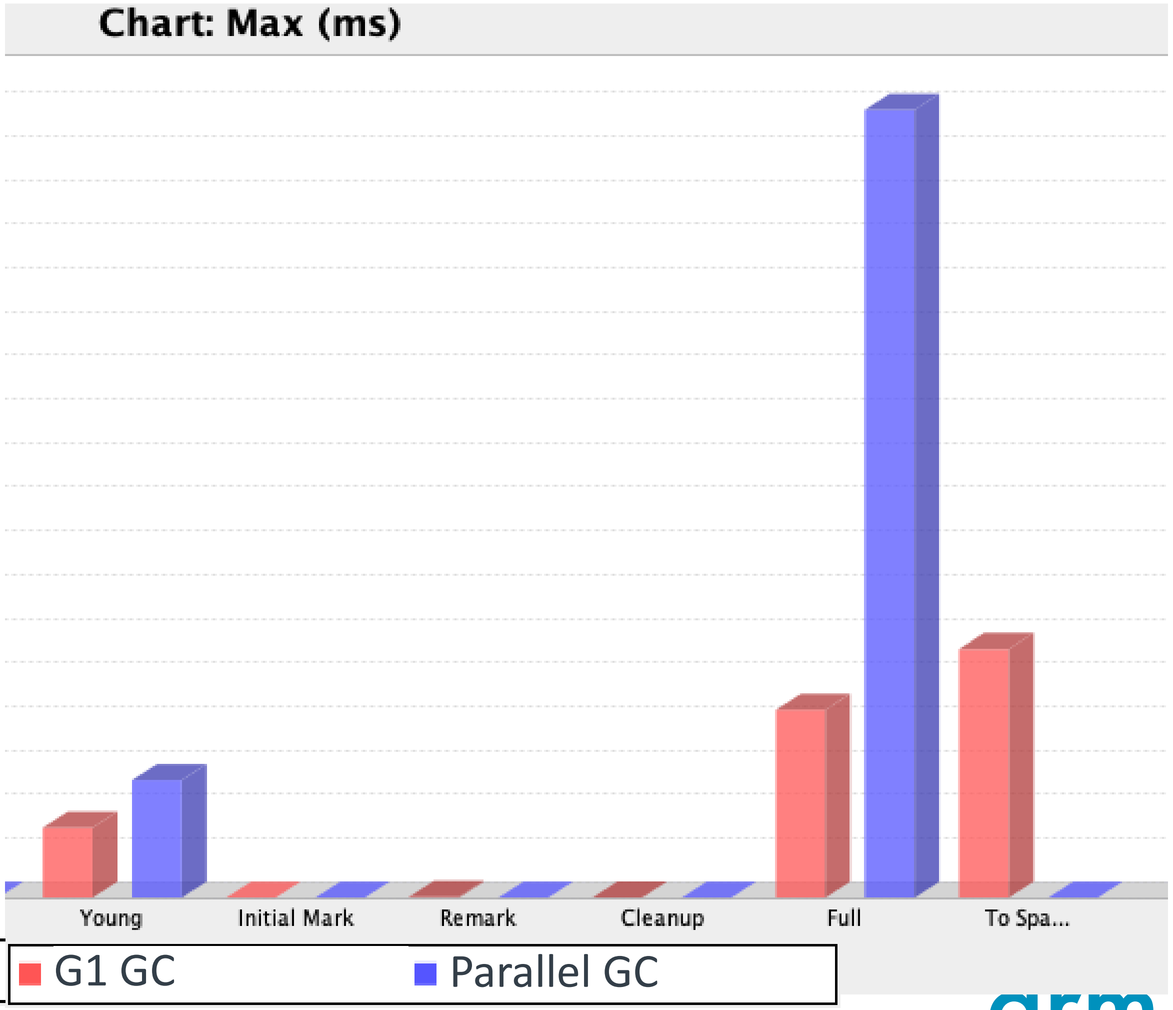


Chart: Max (ms)

Scenario 3

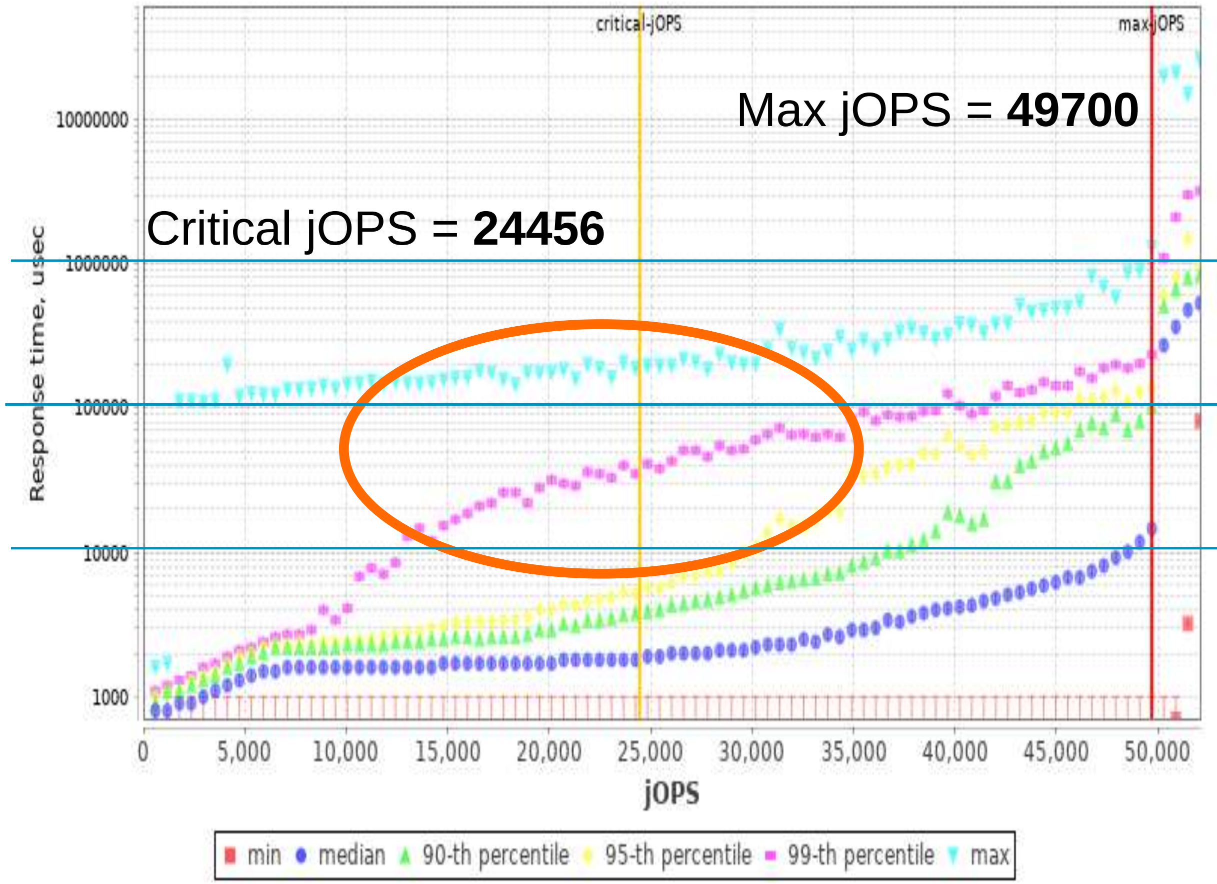
Increase CPU Usage by Optimizing Task Scheduler

Check your SLA Constraints!

Fork Join Pool Scheduler Comparison: 1x vs 2x (of SMT)

Optimizing CPU usage

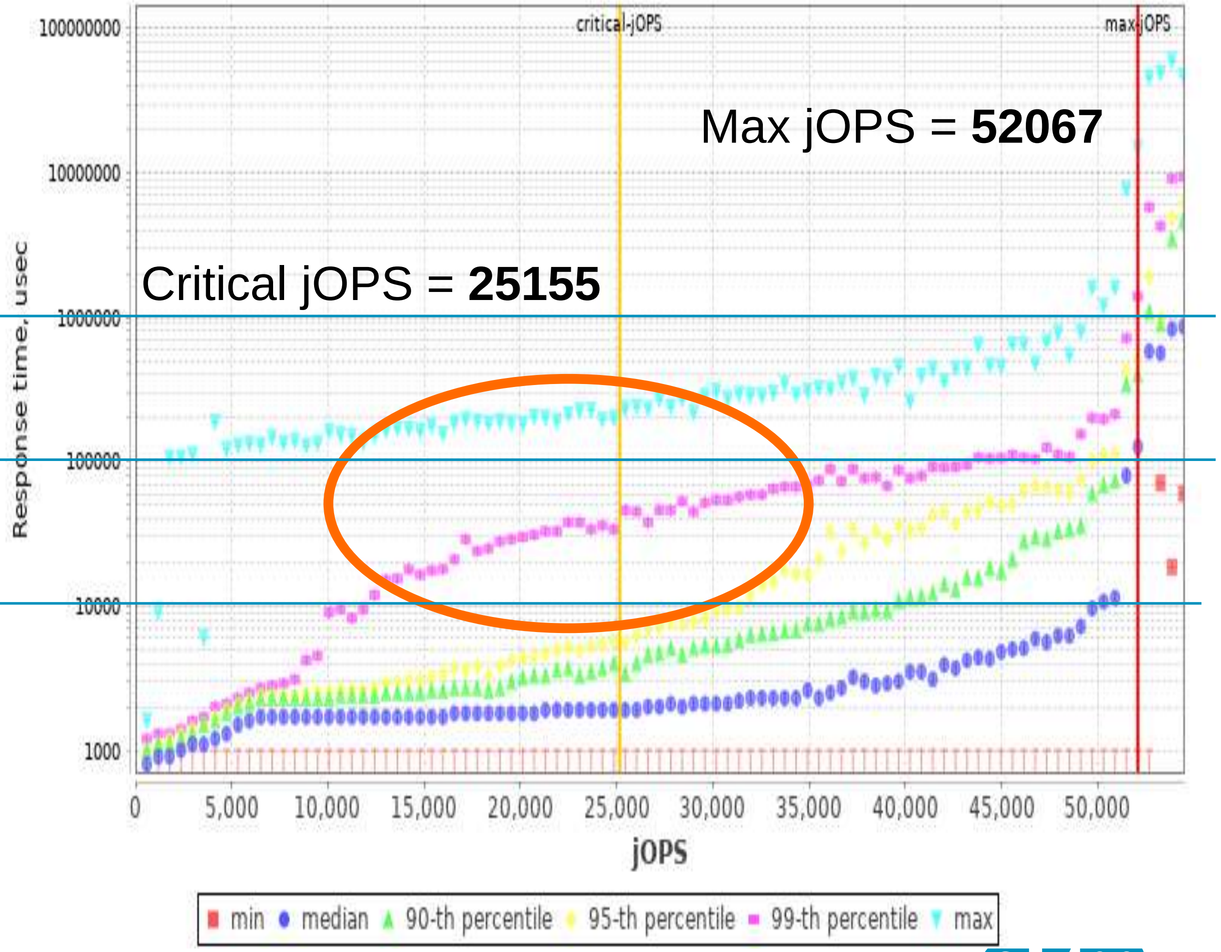
FJP == SMT count



Max jOPS = **49700**

Critical jOPS = **24456**

FJP = 2xSMT count



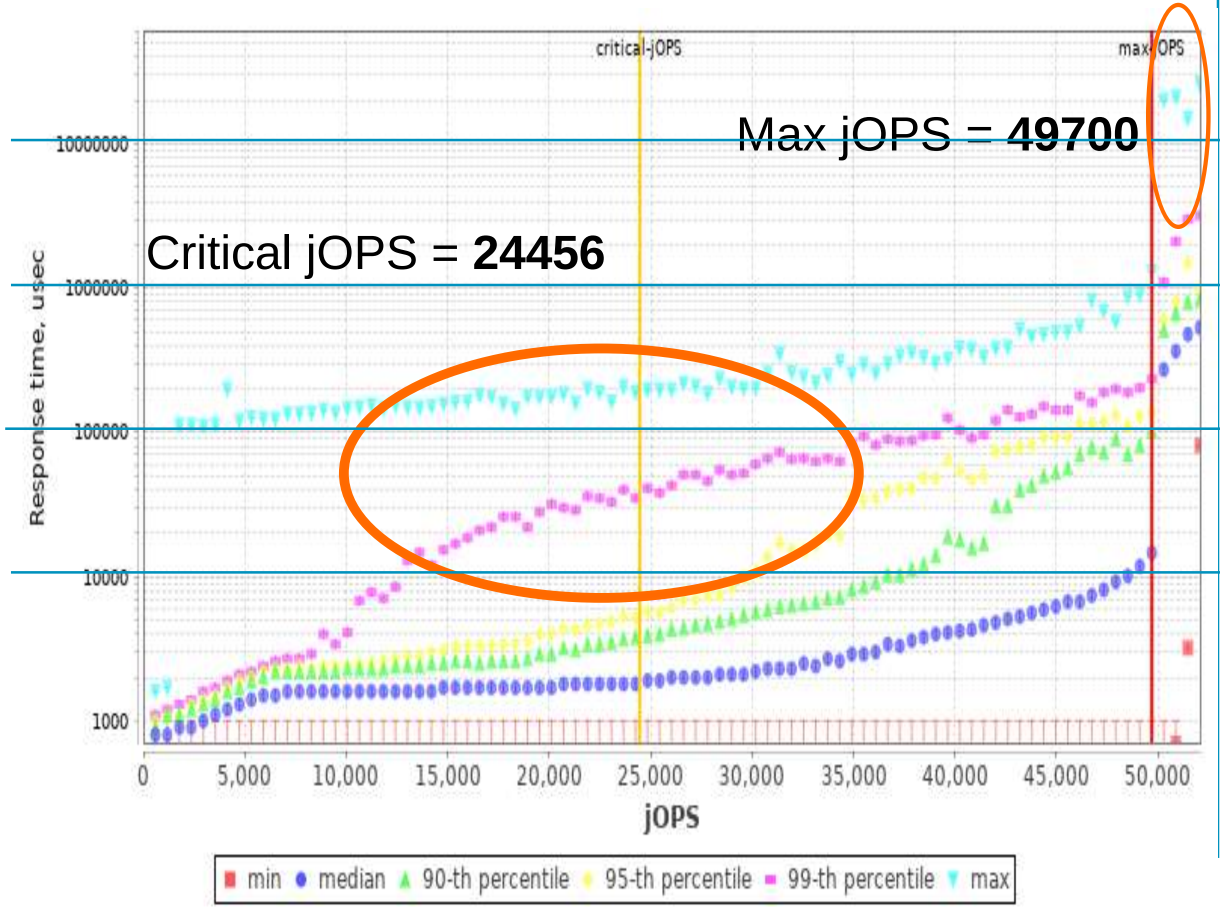
Max jOPS = **52067**

Critical jOPS = **25155**

Fork Join Pool Scheduler Comparison: 1x vs 2x (of SMT)

Optimizing CPU usage

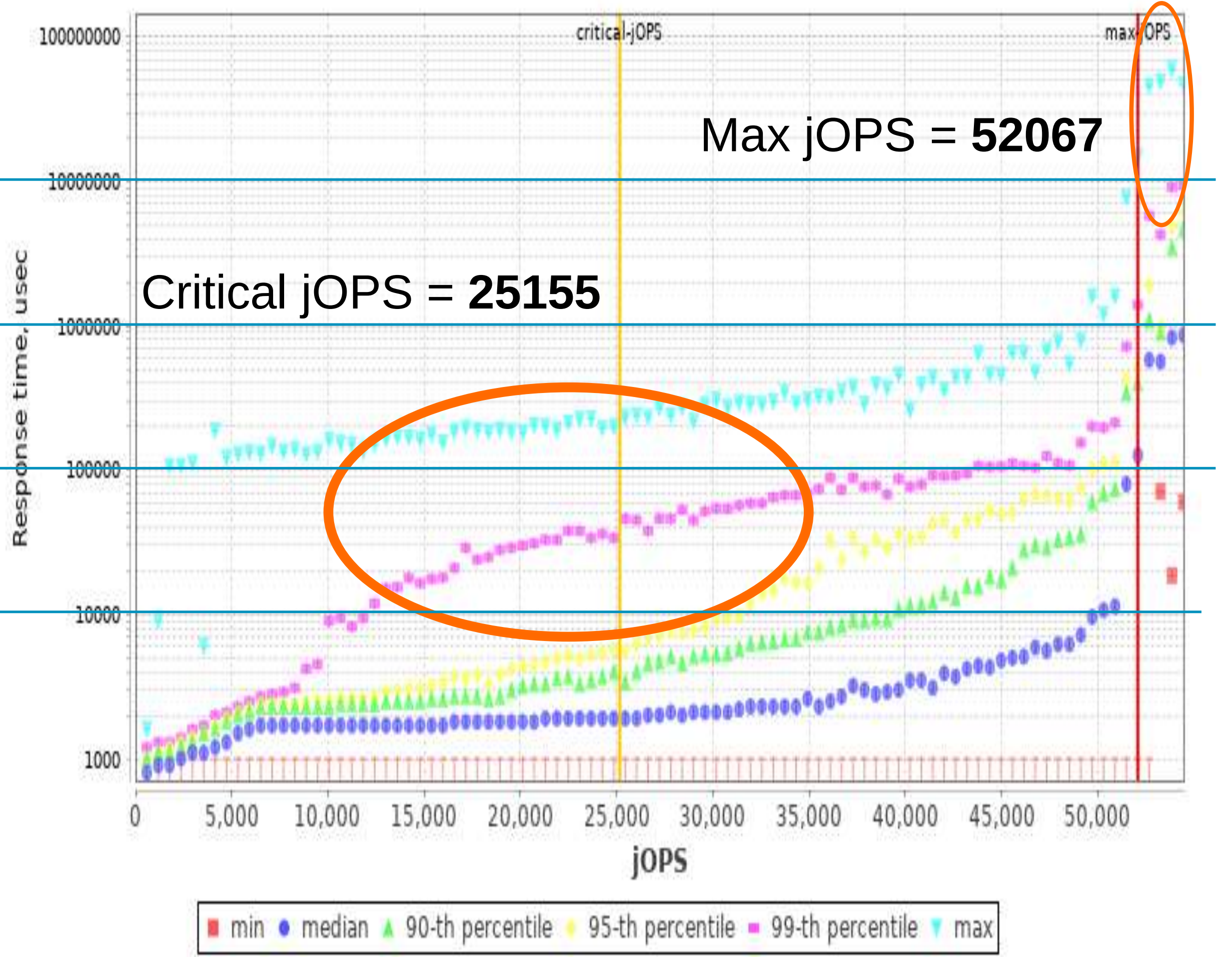
FJP == SMT count



Max jOPS = **49700**

Critical jOPS = **24456**

FJP = 2xSMT count



Max jOPS = **52067**

Critical jOPS = **25155**

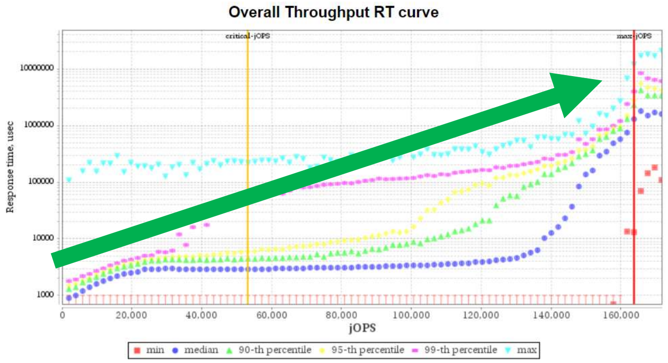
Summary - If ifs and buts were candies and nuts …

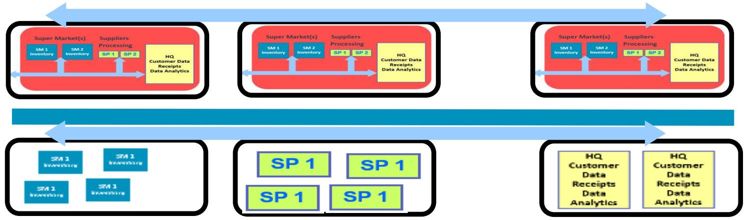
Scaling Up a System Can Be Easy …

If…	When…
We check our SLA constraints	Increasing the heap
	Choosing GC algorithms
	Optimizing CPU usage

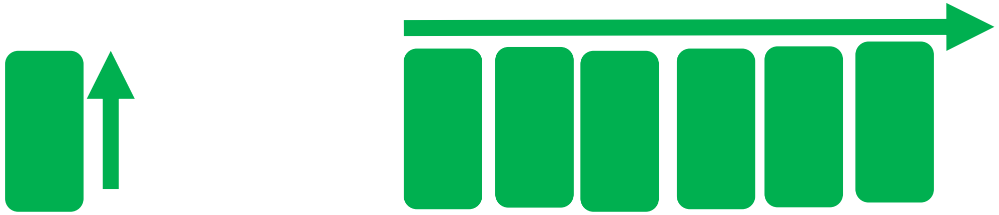
Conclusions:

- Scale UP:
  - Telemetry and correlation
  - Estimate of performance gain
  - Footprint and SLA
- Scale OUT
  - Telemetry and correlation
  - Cost of orchestration and weigh throughput vs latency
  - $$$$$ for scaling out vs. throughput meeting SLA





Scale up performance benchmarking



Scale Up → Scale Out → Away to

(Telemetry)

(Hawaii)