Spyre: A Resource Management Framework for Container-based Clouds

Karthick Rajamani, Alexandre Ferreira, Juan Rubio
Optimized Cloud Infrastructure, IBM Research

Wes Felter
IBM Cloud Innovation Lab

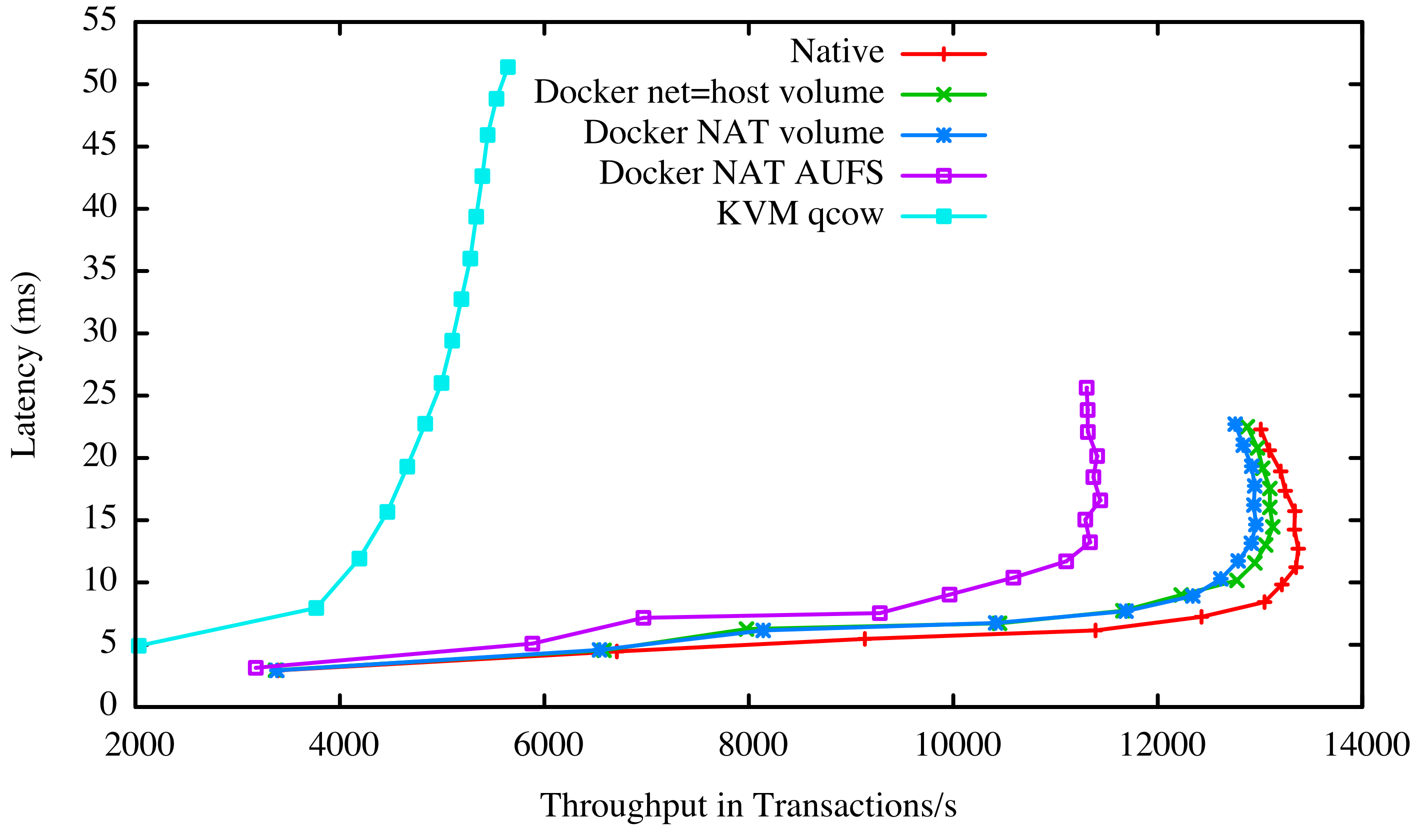
{karthick,apferrei,rubioj,wmf}@us.ibm.com

Overview

- What is Spyre?
- Resource management with Spyre
- Performance evaluation
- Status and next steps
- Extending Tenant SLA models – discussion

Containers offer better performance than VMs

Sysbench with MySQL



Source: An Updated Performance Comparison of Virtual Machines and Linux Containers – Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio

What is Spyre?

Optimized foundation for the container-based cloud

- Containers are fundamental unit of computation (not container in VM)
- Superior resource isolation and performance (tail latency) for tenant/performance-sensitive services – resource-isolated slices.
- Support resource-sharing among containers used as side-cars (running within same slice).
- Avoid multi-tenant dockerd issue – each client (slice) can have their own dockerd.
- Can be used with any container eco-system – we have experimented to date with Docker.

Spyre Goals

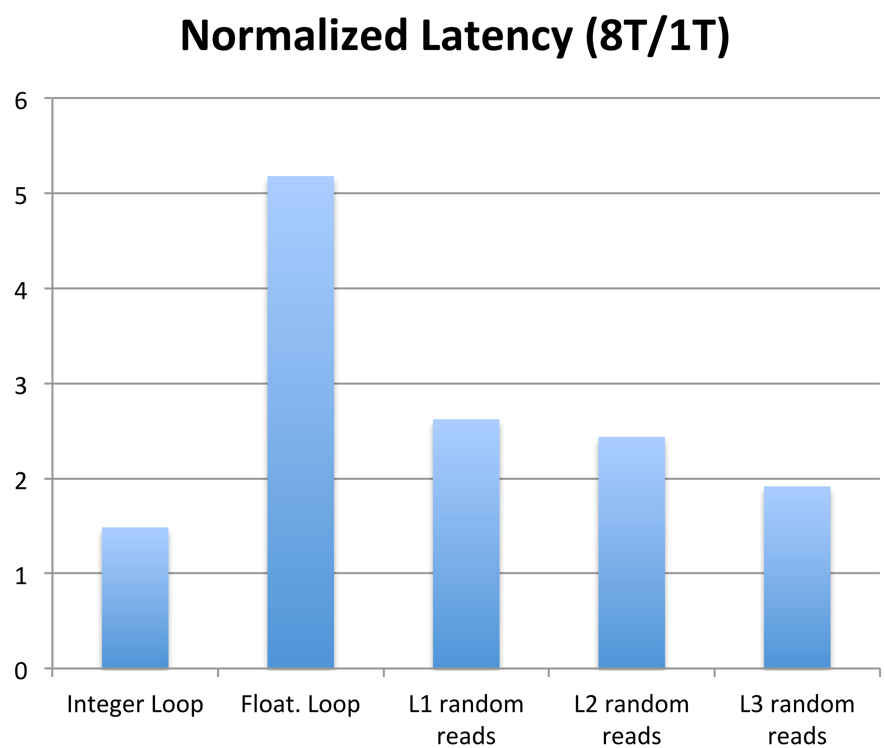
- Predictable performance (including tail)
  - Strong isolation (e.g., dedicated physical cores) with *slices*
  - Allocate resources using *real* units (say Ghz not abstract compute units)
  - Unique use/configuration of cgroups
- Vertical scaling
  - Grow containers while running (e.g., add cores/RAM)
  - Subletting: spot market (like a CloudBnB)
- High performance
  - Base unit is containers
  - Optimize storage & network I/O

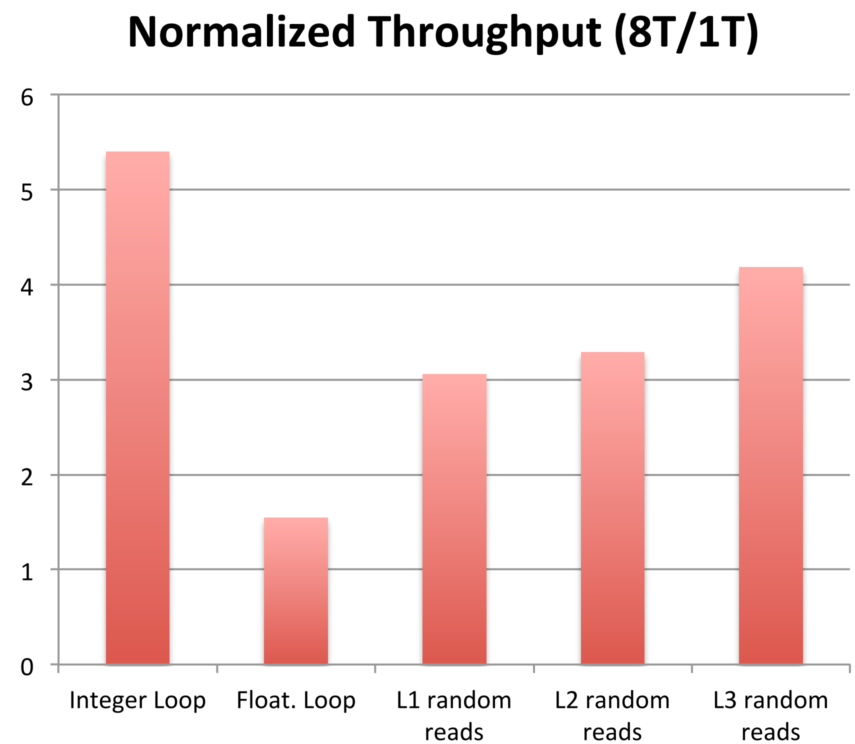
    e.g., eliminate NAT and replace AUFS with block storage

Resource management with Spyre

- Key concept: Slices
  - Dedicated resources for predictable/guaranteed performance
    - Dedicated physical cores
    - Dedicated RAM

Challenges to sharing core resources





- Shared cores result in variable impact on performance
  - Significant, difficult to predict impact for tenant workload
  - Difficult to predict returns for provider

- Data taken on POWER8 processor which has dedicated L1, L2, L3 cache per core

Resource management with Spyre

- Key concept: Slices
  - Dedicated resources for predictable performance
    - Dedicated physical cores
    - Dedicated RAM
  - Guaranteed minimum network bandwidth
  - Multiple vNICs, IP addresses, block storage (optional)

IAAS Customer View - Slice

Name	Type	cores	RAM (GB)	L3 (MB)	Net. BW	Price/hr
BDW-2GB-HT	Broadwell 2GHz	1/4	2	-	0.31	$0.04
BDW-4GB-1T	Broadwell 2GHz	1/2	4	-	0.63	$0.06
BDW-8GB-1C	Broadwell 2GHz	1	8	1.5	1.25	$0.10
BDW-16GB-2C	Broadwell 2GHz	2	16	3.0	2.50	$0.22
BDW-24GB-3C	Broadwell 2GHz	3	24	4.5	3.75	$0.33
BDW-32GB-4C	Broadwell 2GHz	4	32	6.0	5.00	$0.44
BDW-40GB-5C	Broadwell 2GHz	5	40	7.5	6.25	$0.55
BDW-48GB-6C	Broadwell 2GHz	6	48	9.0	7.50	$0.66
BDW-56GB-7C	Broadwell 2GHz	7	56	10.5	8.75	$0.77
BDW-64GB-8C	Broadwell 2GHz	8	60	12.0	10.00	$0.80
P8-4GB-2T	Power8 3.x GHz	1/4	4	-	0.25	$0.06
P8-8GB-4T	Power8 3.x GHz	1/2	8	-	0.5	$0.11
P8-16GB-1C	Power8 3.x GHz	1	16	8.0	1	$0.20

Observations from other work

- ISCA 2015 – "Heracles: Improving Resource Efficiency at Scale – David Lo et al."
  - Latency critical workloads need dedicated/isolated resources, distinct from those allowed to be assigned for batch workloads
- Microservices require stronger focus around component-service tail latencies
  - Increased probability of impact on composite service latency.
  - https://engineering.linkedin.com/performance/who-moved-my-99th-percentile-latency - Richard Hsu and Cuong Tran

Spyre-slice frame-work of value also to latency-sensitive cloud services.

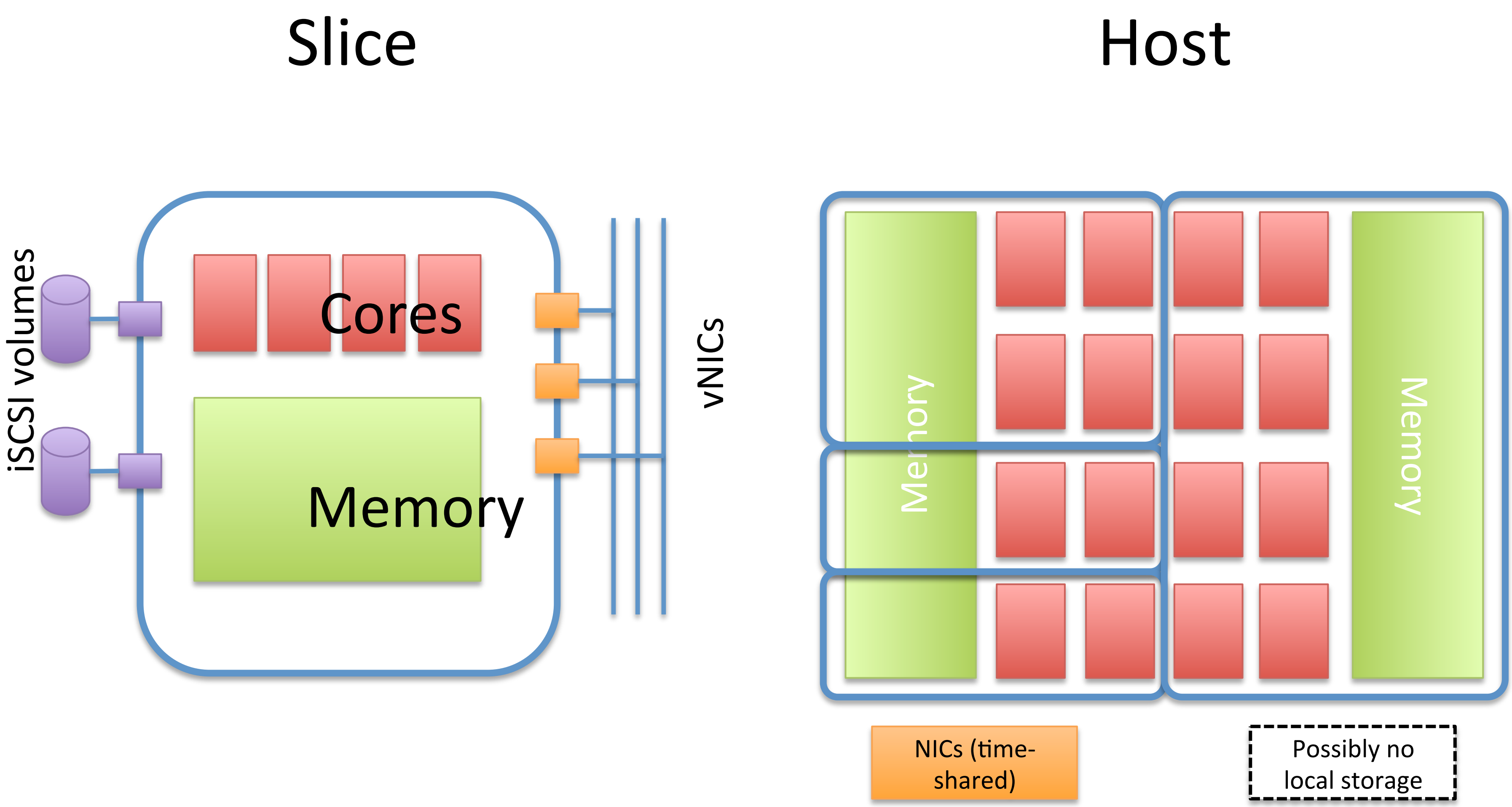
Resource management with Spyre

- Key concept: Slices
  - Dedicated resources for predictable performance
    - Dedicated physical cores
    - Dedicated RAM
  - Guaranteed minimum network bandwidth
  - Multiple vNICs, IP addresses, block storage (optional)

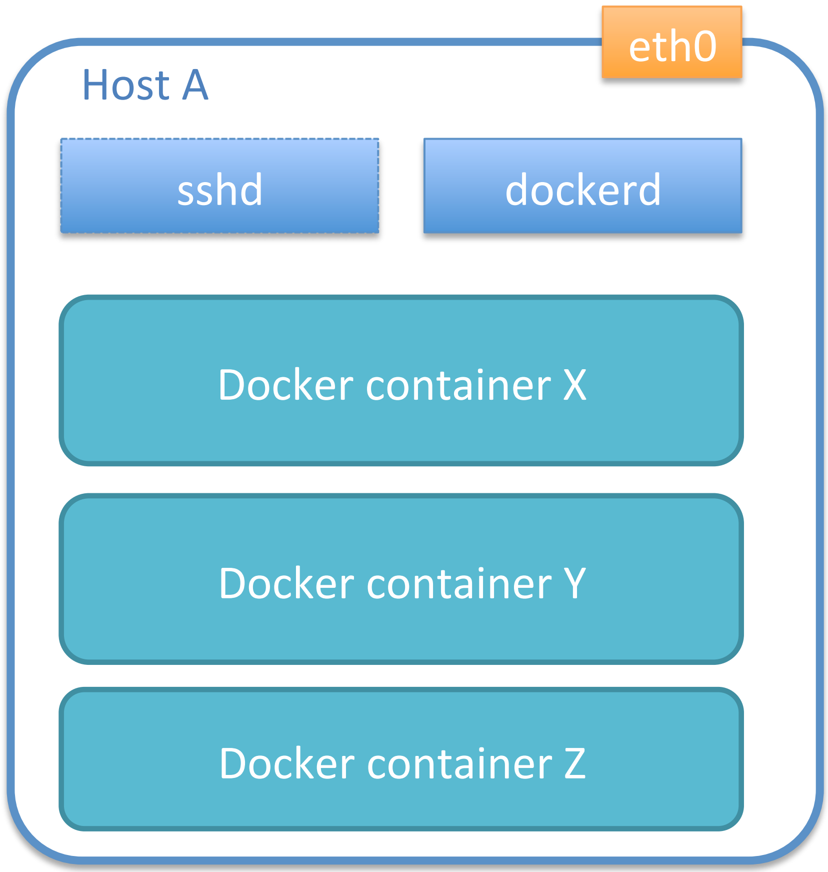
- Implemented using cgroups & systemd units
  - Note: systemd does not yet support dedicated cores (cpusets), custom script implements it.

- Multiple containers per slice (similar to Kubernetes *pod*/Carina *segment*)
  - Allows intra-customer sharing of resources

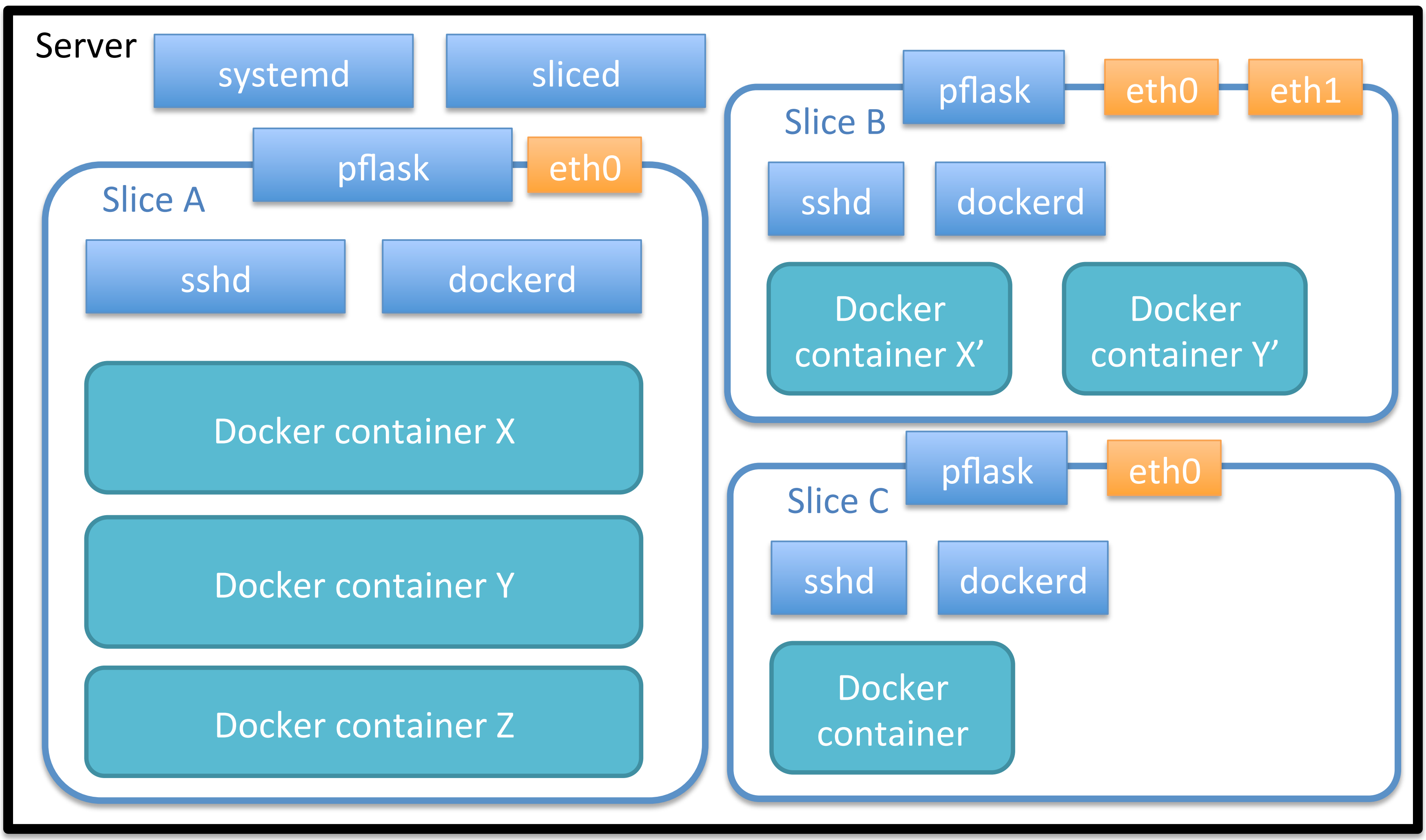
Resource View



Slice — tenant view



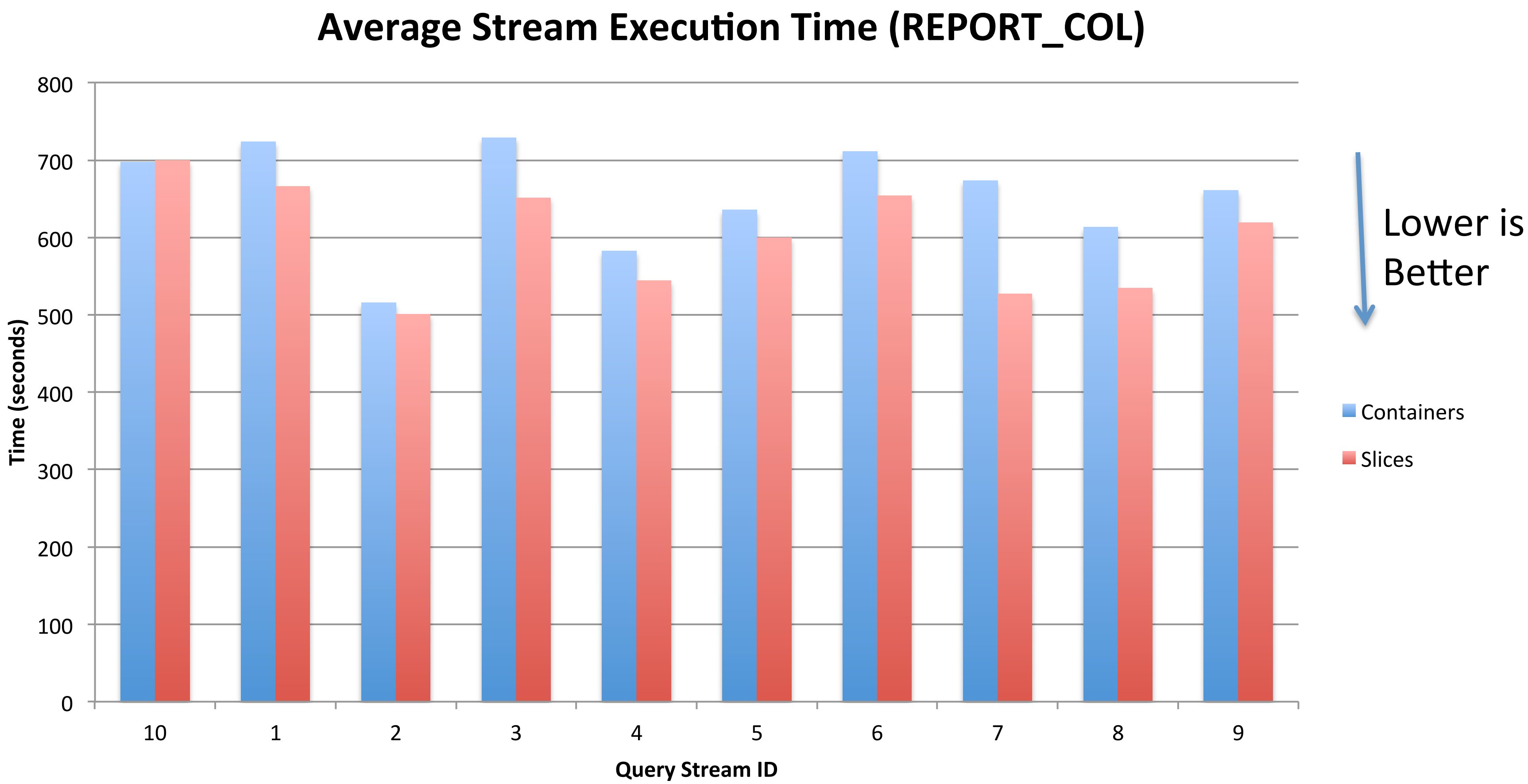
Host — software view



Slice benefit analysis with an in-memory database workload

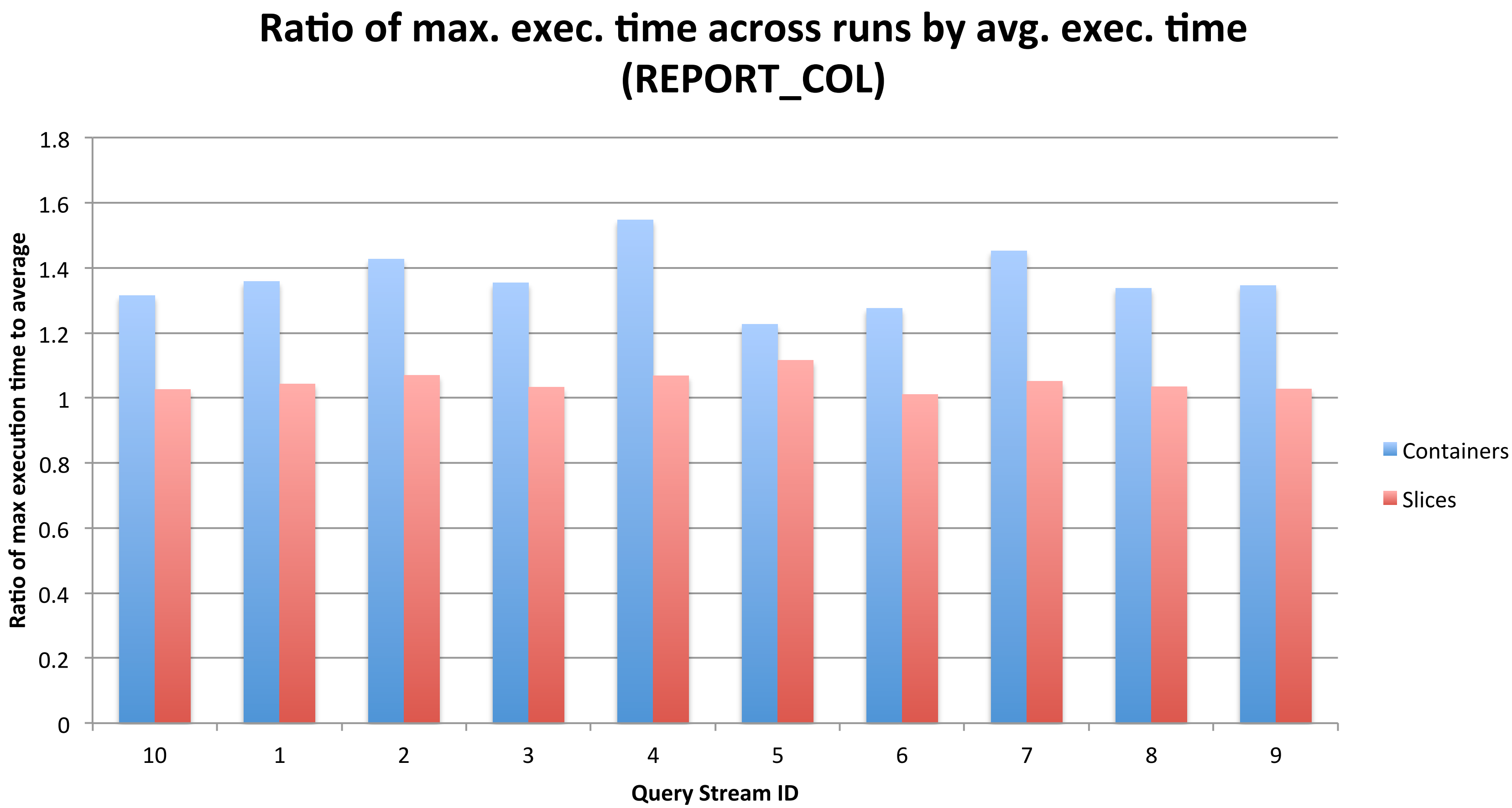
- DB2 BLU (in-memory database)
  - AGG_COL
    - includes up to 10 concurrent streams of SQL queries
    - used to emulate background, interfering job
  - REPORT_COL
    - includes up to 10 concurrent streams of SQL queries
    - used to emulate foreground job.
- 4 instances of 100GB datasets with 3 REPORT_COL and 1 AGG_COL executed concurrently in
  - 4 Docker containers on Host
  - 4 Docker containers, each within own slice (6-core) on Host
- 24-core POWER8-S824 machine (2 6-core dies per socket, 2 sockets) with 512GB of memory spread evenly among the dies.
- All databases are resident on iSCSI volumes
- 2 runs done for both container-only and containers-within-slice scenarios
  - 6 data points for REPORT_COL, 2 data points for AGG_COL for each scenario

REPORT_COL average



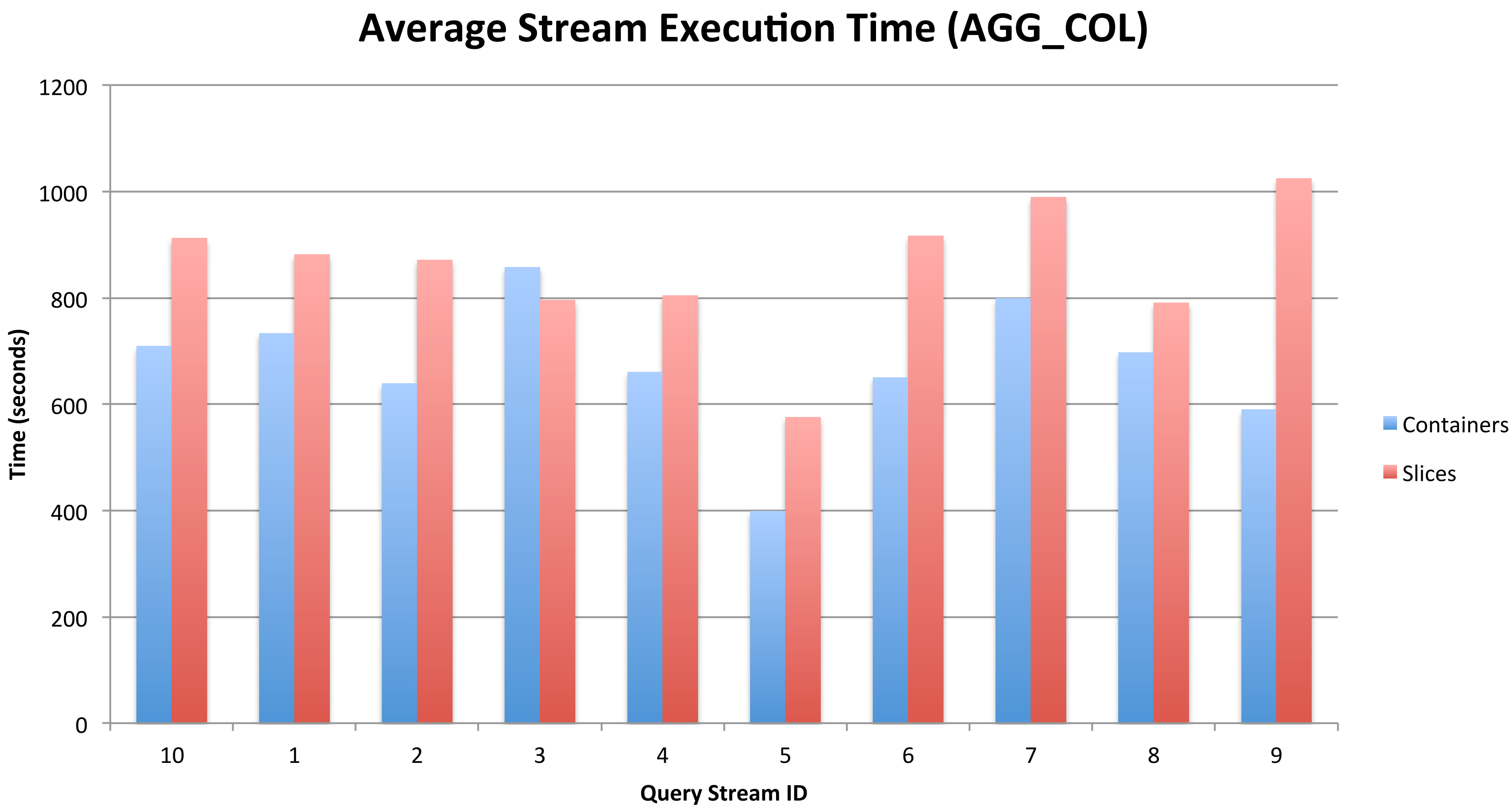
Average better or same with slices.

REPORT_COL worst-case performance



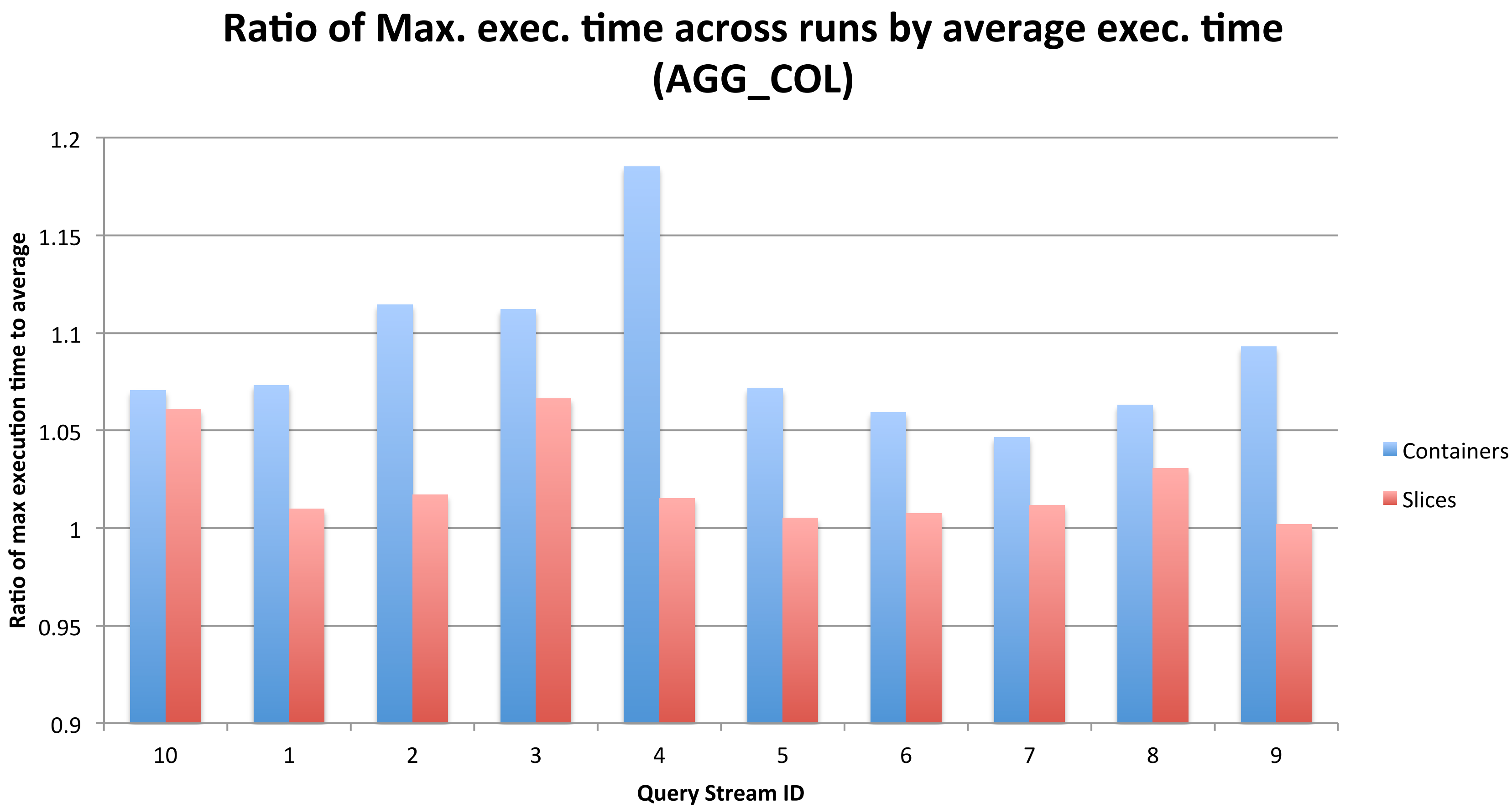
Slices improve worst-case performance i.e. lower tail latency (lower ratio of max to average)

AGG_COL average



AGG_COL benefits from stealing resources, i.e., sees lower performance when constrained within slice.

AGG_COL worst-case performance



Lower variation of runtimes with slices.
caveat: only two data points behind each bar.

Spyre Status

sliced in Linux on x86 and POWER:

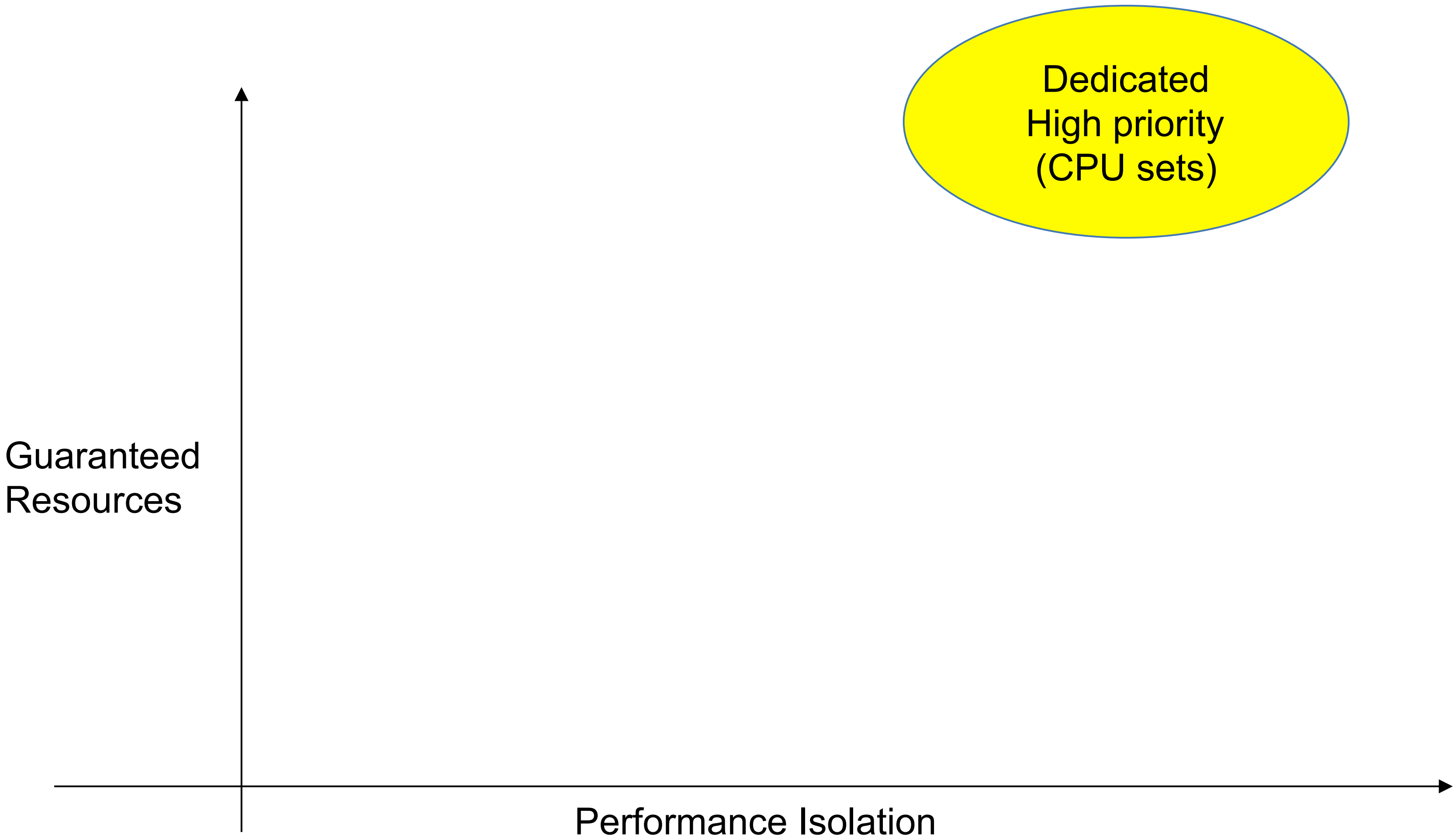
- Interface
  - Simple REST API supporting slice create, query, *resize*, delete, and to query system for resources available/free
  - Returns and accepts JSON
- Capability
  - Provides CPU (core, cache) isolation
  - Automatic memory affinity with CPU
  - Vertical scaling
- Implementation
  - Python
  - Systemd, cgroups, cpuset
  - Pflask for outer container
  - Slice has own IP, ssh access with public key

Opening project to community

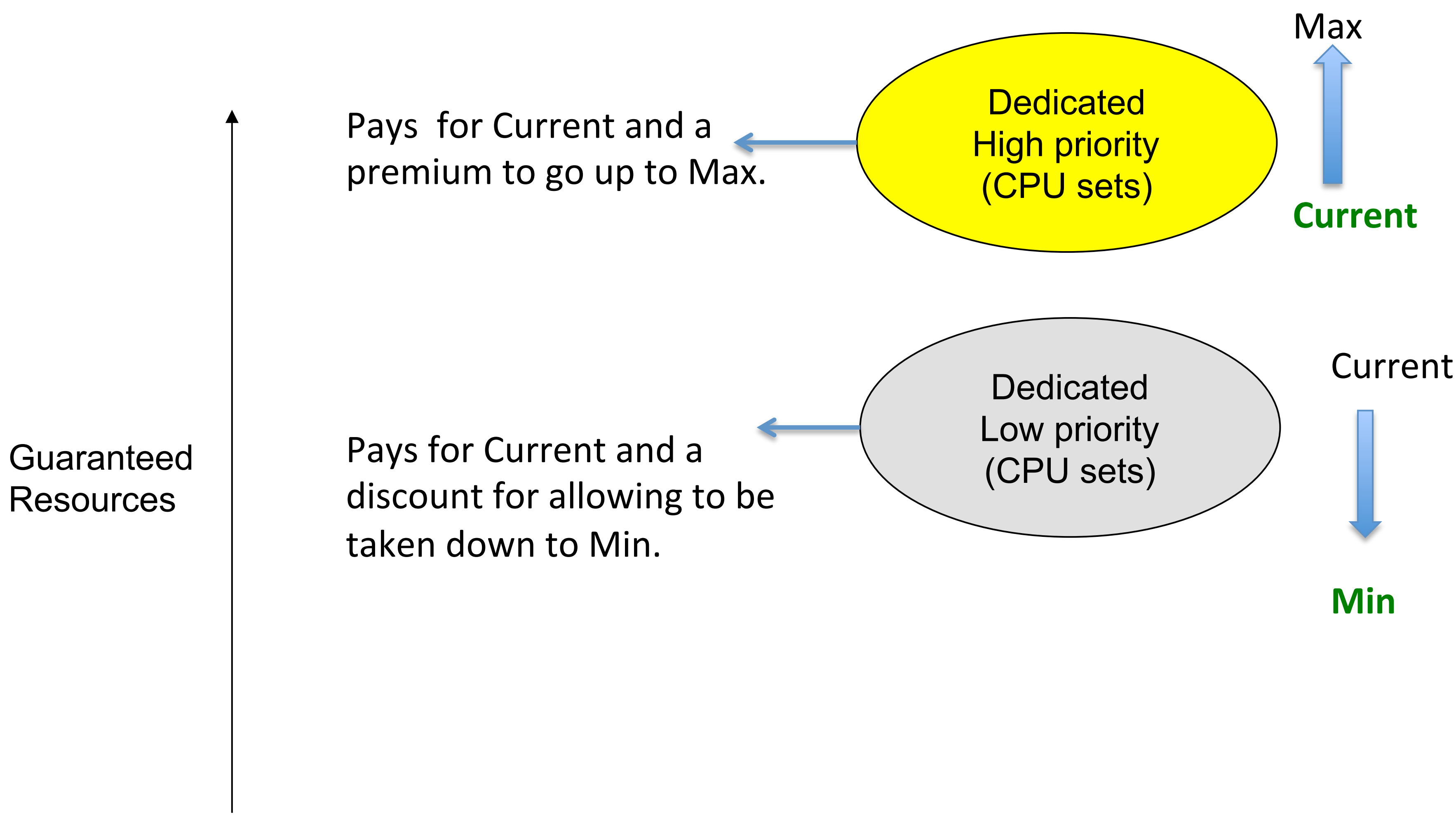
Next Steps

- *Spyred* implementation for stand-alone cluster.
- Memory bandwidth control (IBM POWER8) and shared-cache control (Intel Haswell+) – hardware-specific.
- Networking design and network bw control work.
- Storage design.
- Integration with broader eco-system: Machine +Swarm, Kubernetes, Mesos… (?)
- *Extending tenant SLA models*

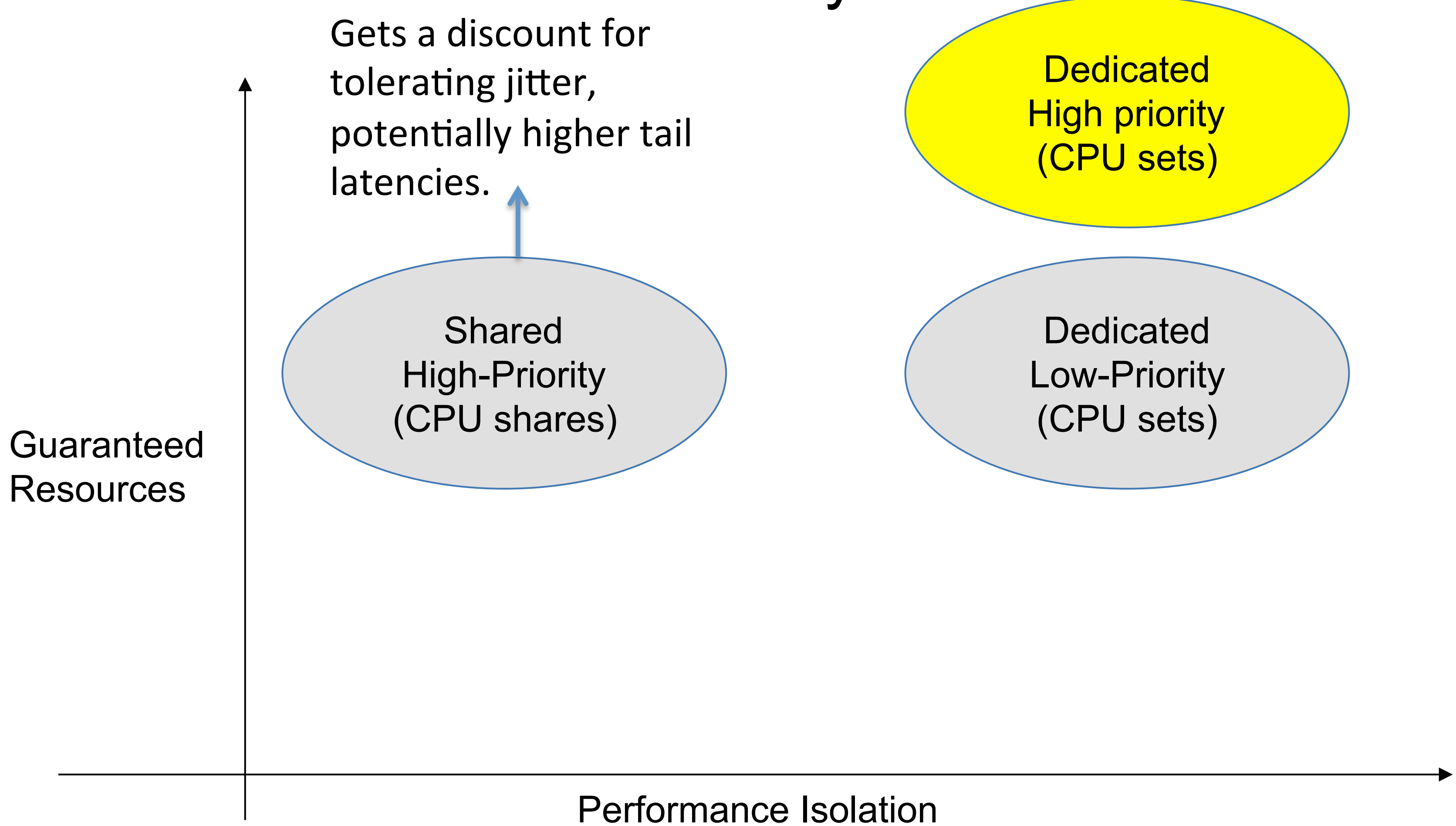
Extending Tenant-Slice service models



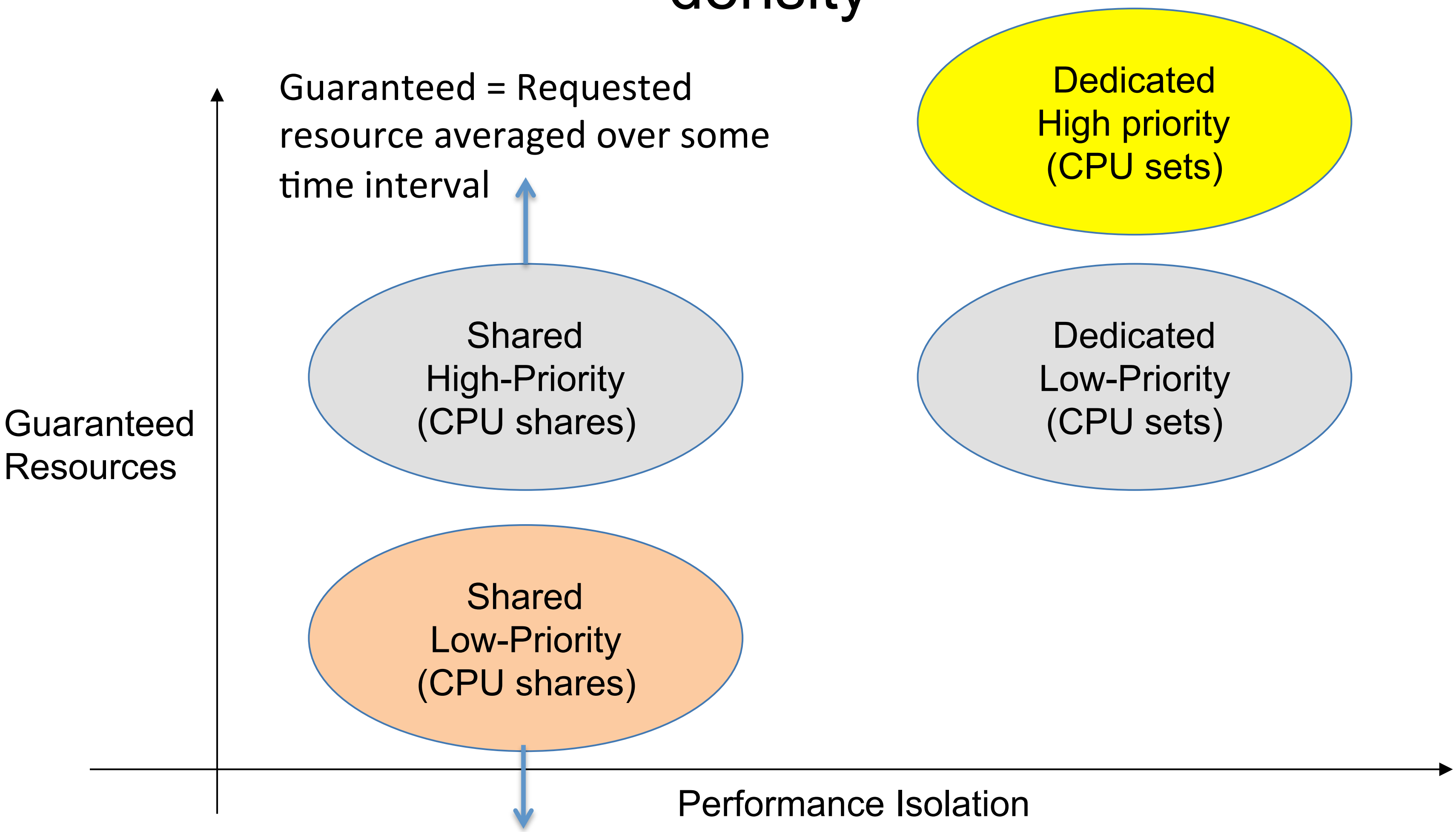
Extending Tenant-Slice service models – Vertical Resizing



Extending Tenant-Slice service models – Increasing density



Extending Tenant-Slice service models – Increasing density



Guaranteed = Requested resource when occasionally active (enables provider to overcommit); gets discount for not needing requested resources all the time.

Discussion

- How important is dynamic resizing ability – both growing up and ability to pay for a lower minimum?
- Is dynamic resizing applicable to memory
  - Can applications deal with some of their allocated memory being moved to swap?
  - Will high-speed swap (SSD/NVME backed) help?
- If a system supports both dedicated and shared is there need for high/low priority sub-classes?
- Any user classes not covered by these models?
- Any other comments, questions?

Thank you

IBM Research is hiring in Cloud Infrastructure and Data centers area.

If interested please contact me,

Email: karthick@us.ibm.com