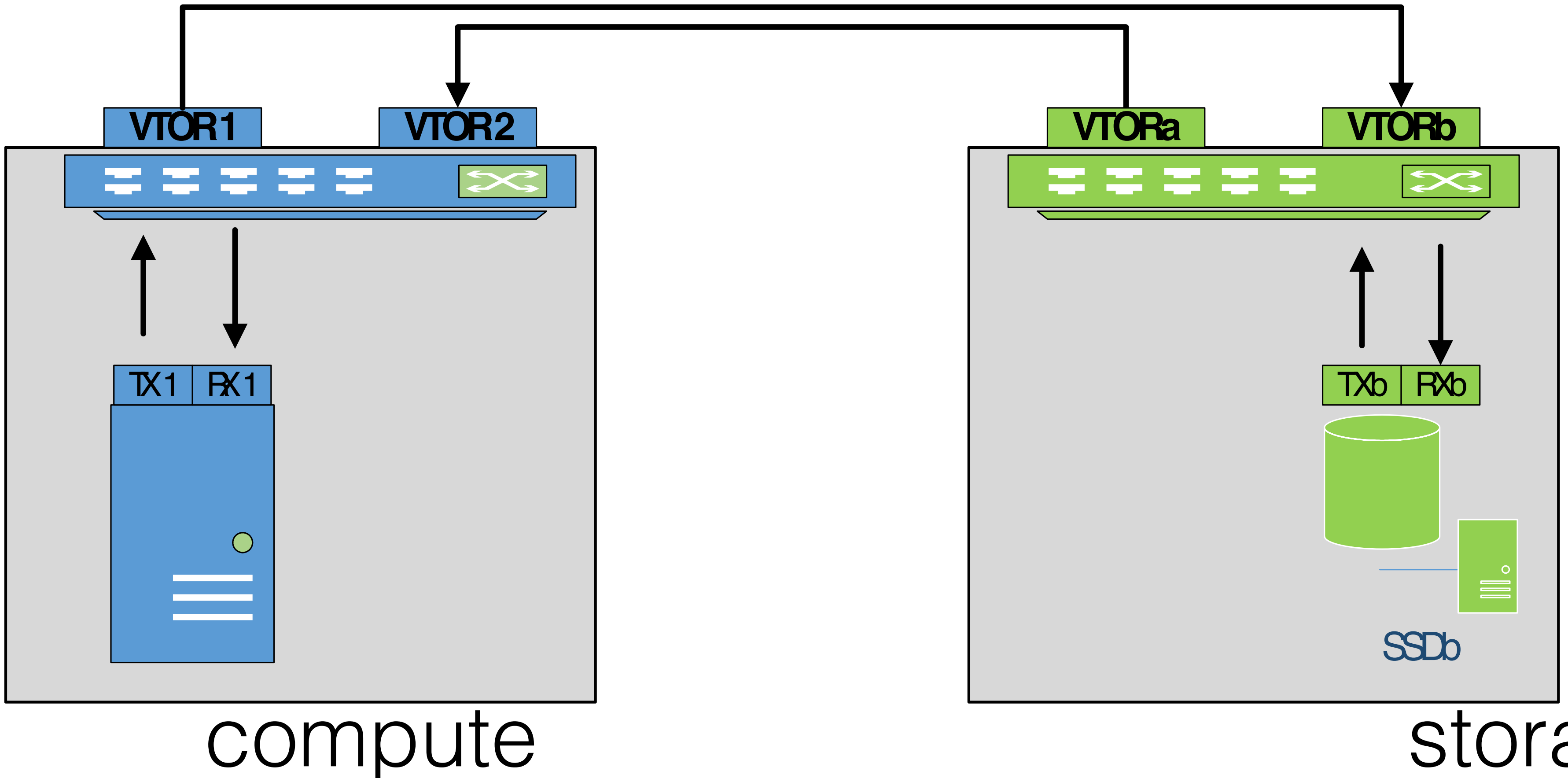
DC-DRF : Adaptive Multi-Resource Sharing at Public Cloud Scale

ACM Symposium on Cloud Computing 2018

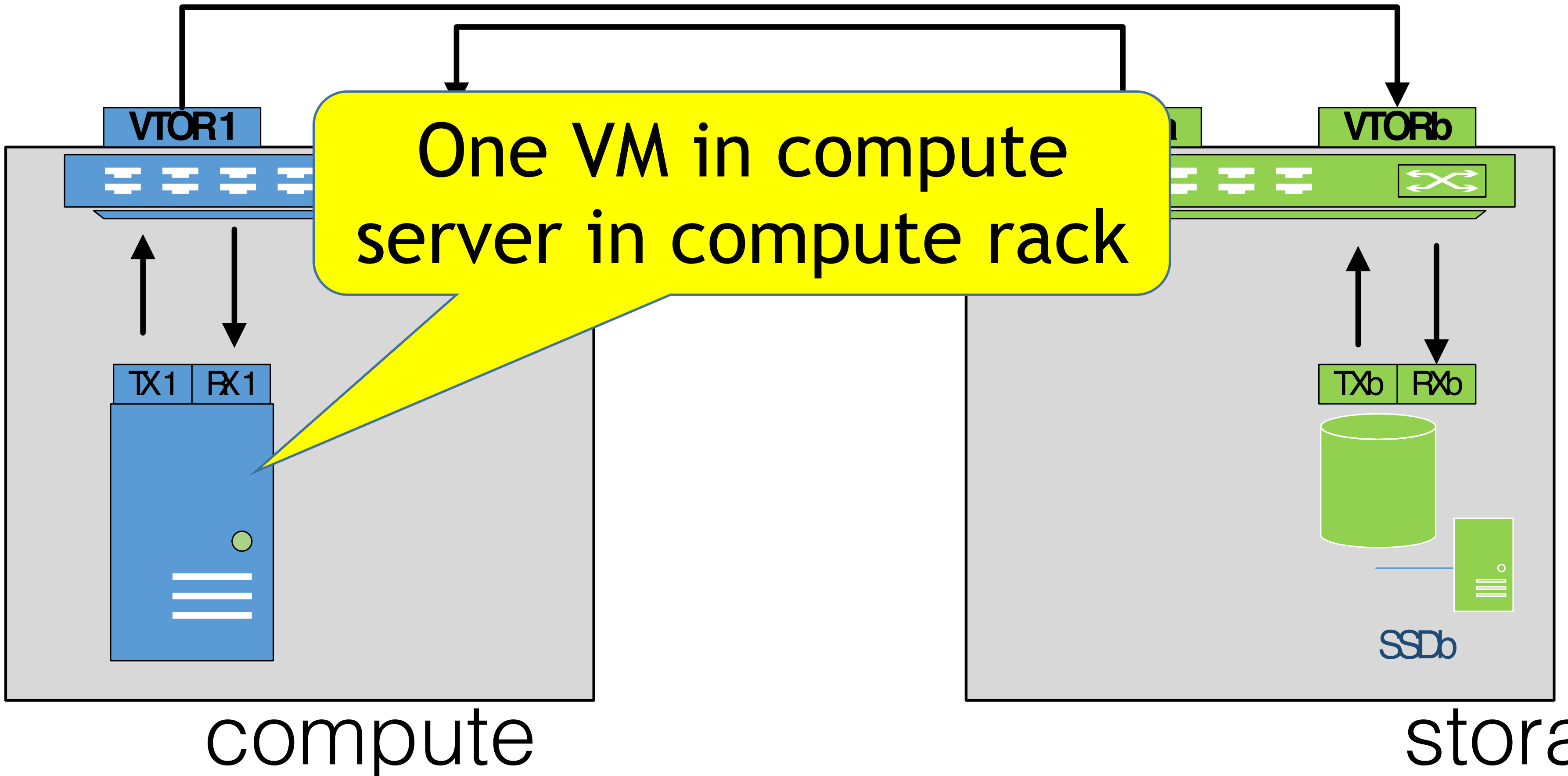
Ian A Kash, Greg O'Shea, Stavros Volos

Public Cloud DC hosting enterprise customers
O(100K) servers, mostly small tenants

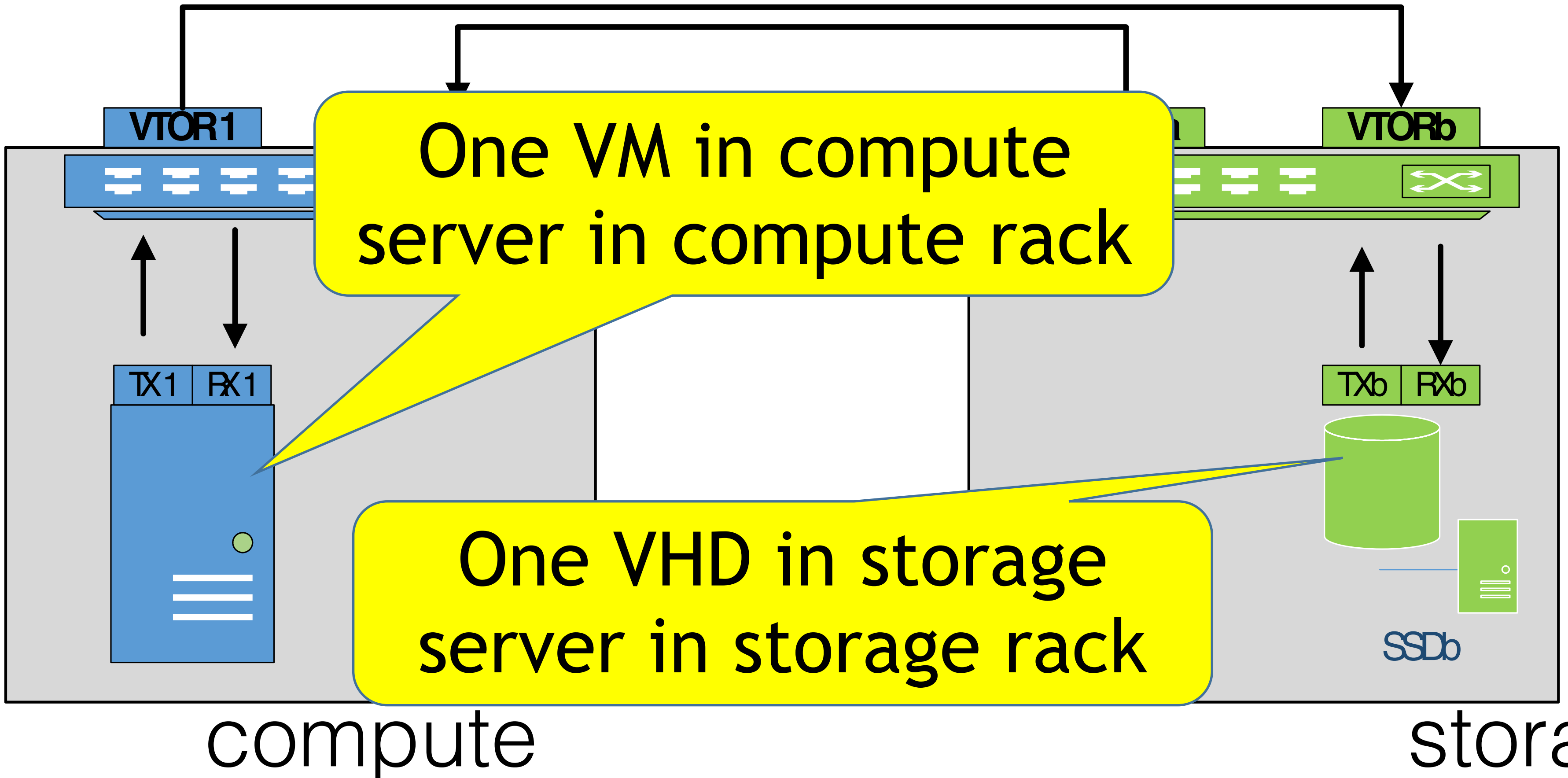
Small customer : one VM accessing storage



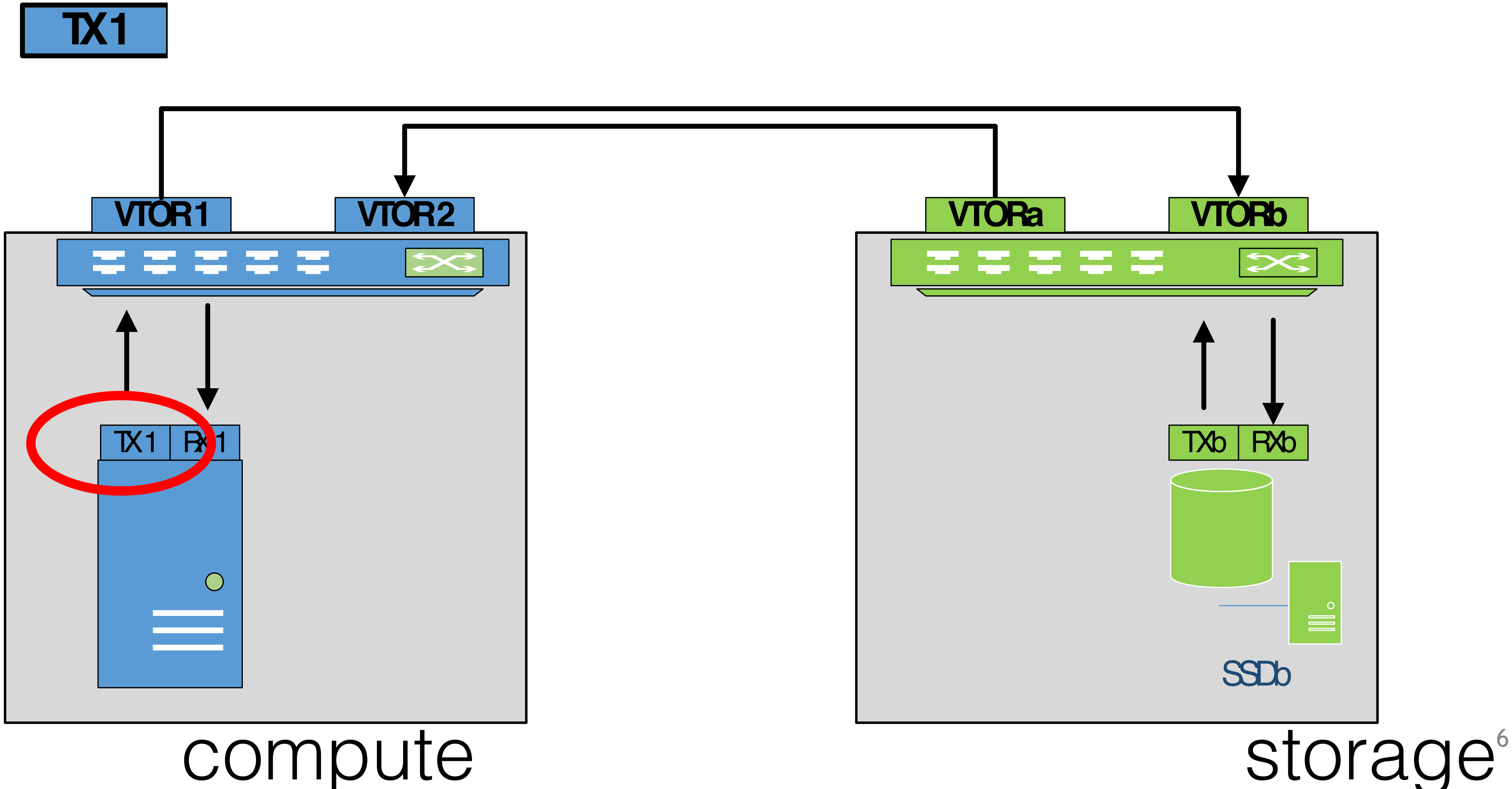
Small customer : one VM accessing storage



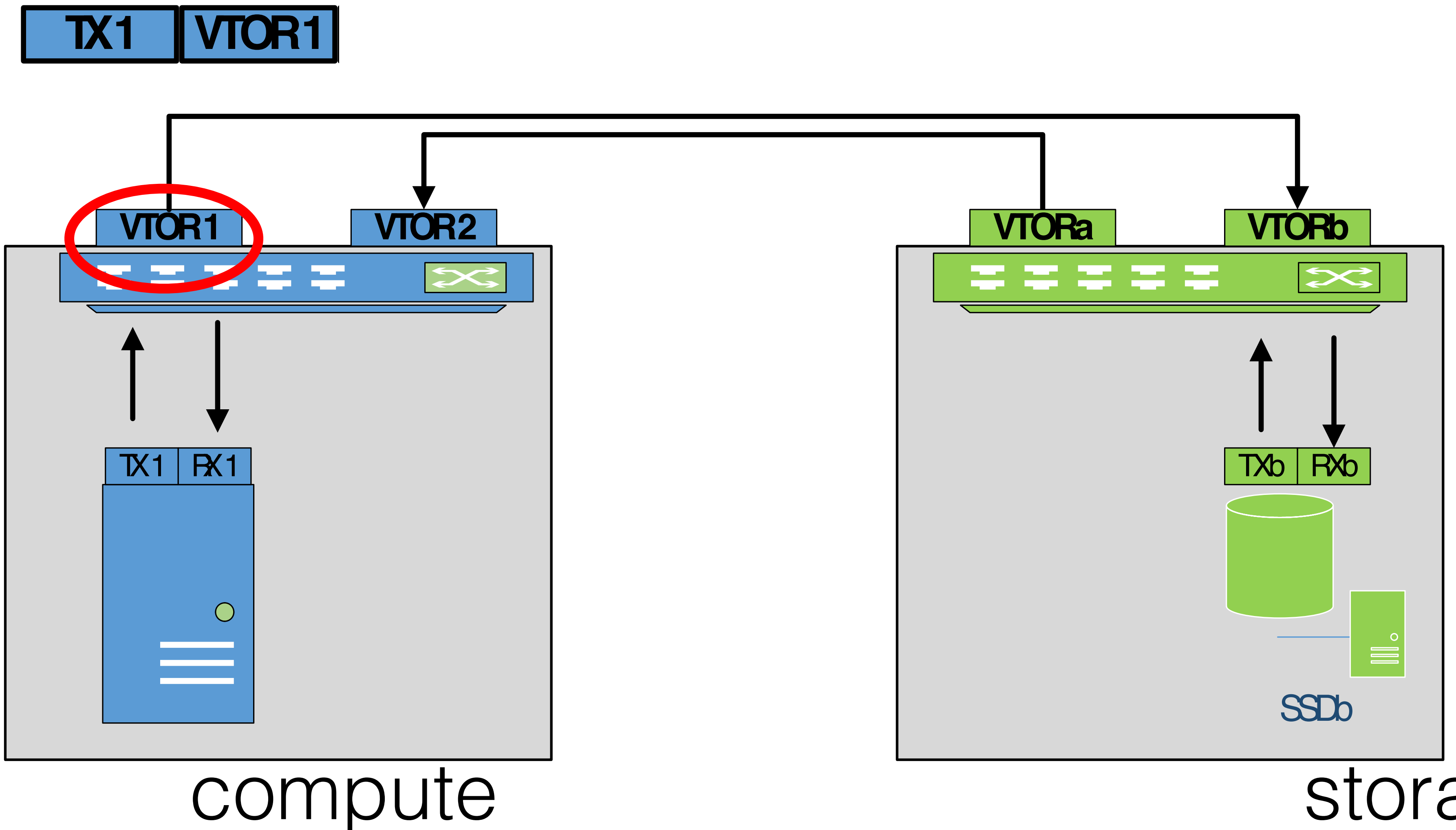
Small customer : one VM accessing storage



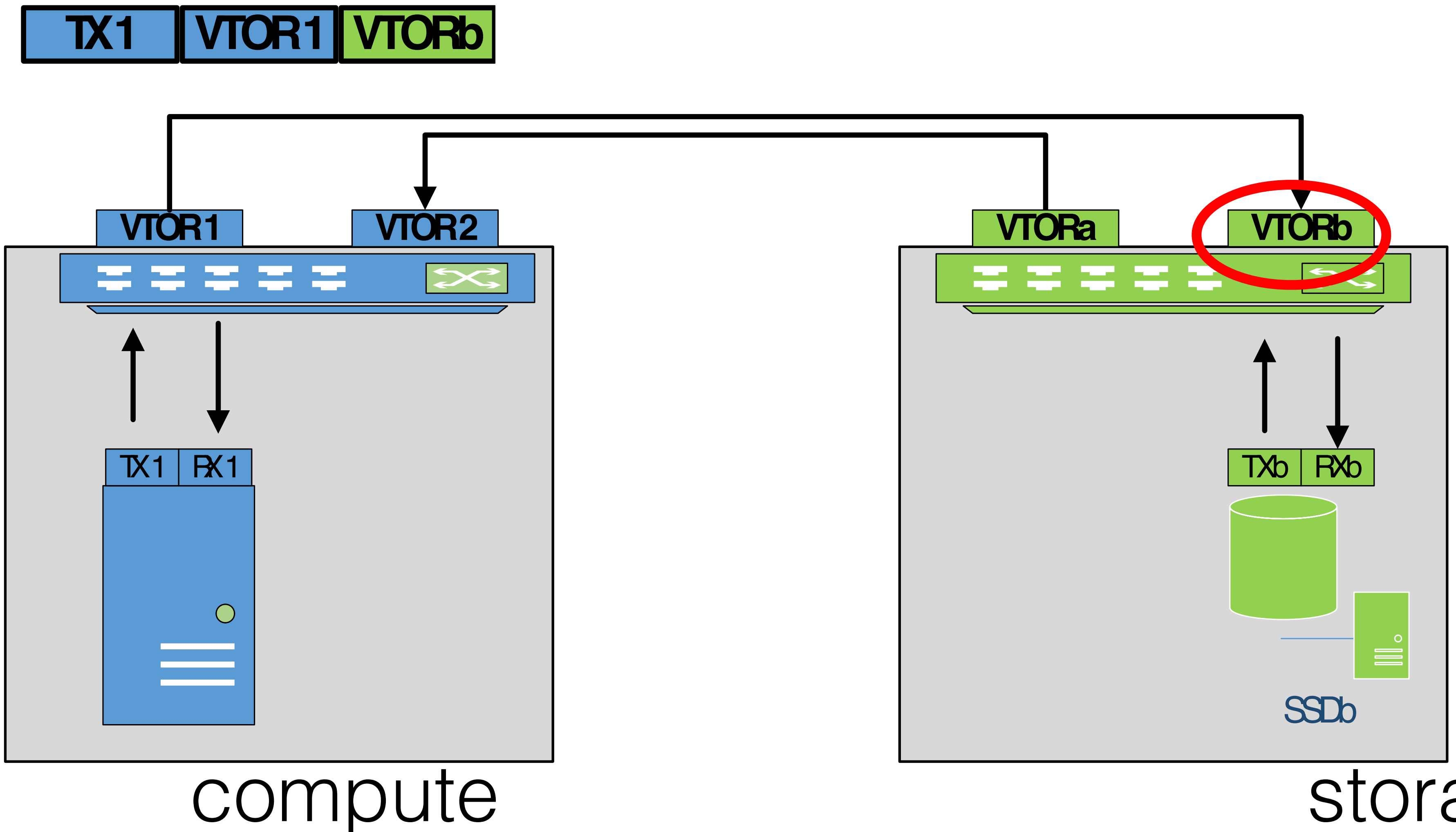
Small customer : one VM accessing storage



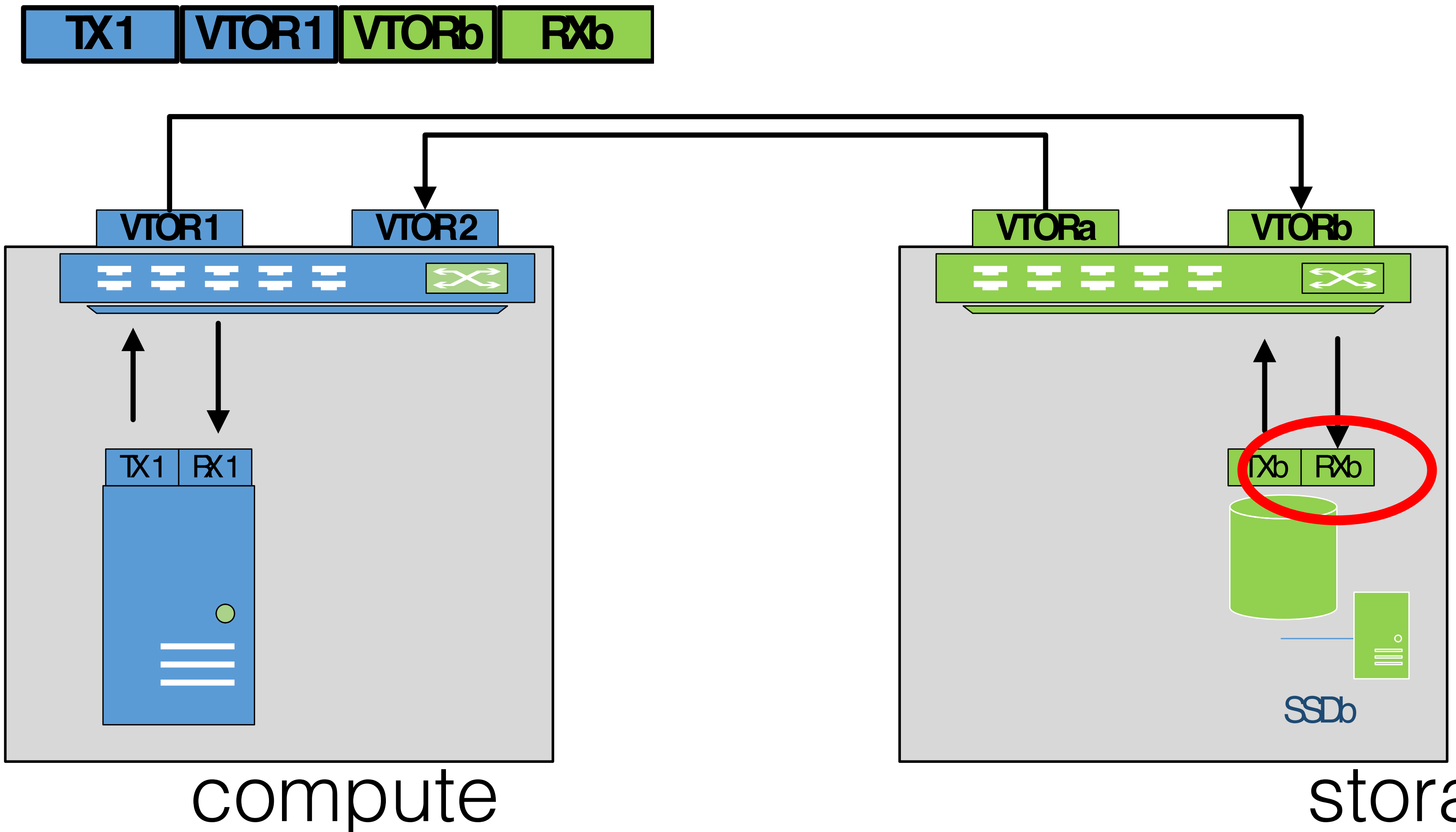
Small customer : one VM accessing storage



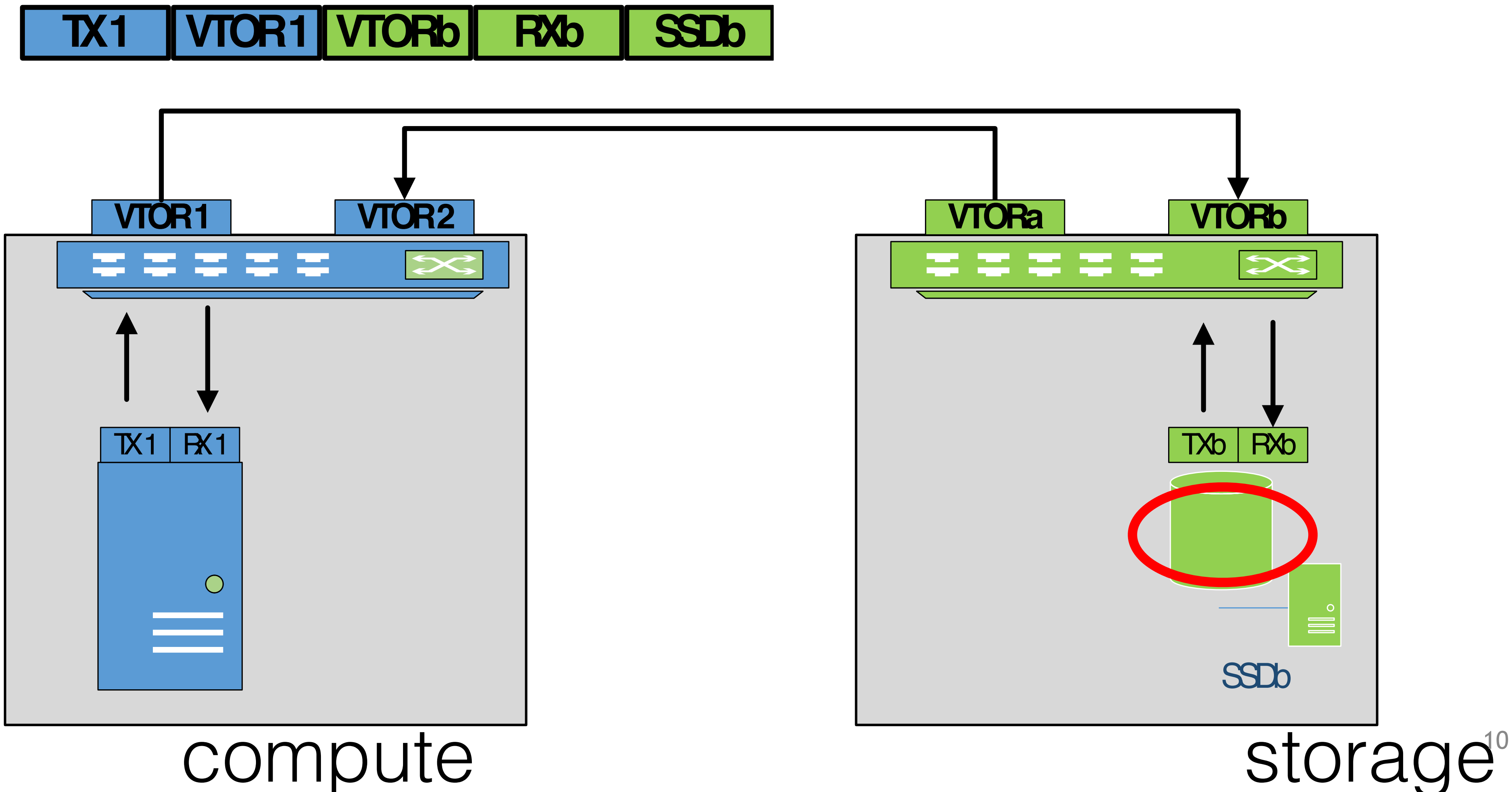
Small customer : one VM accessing storage



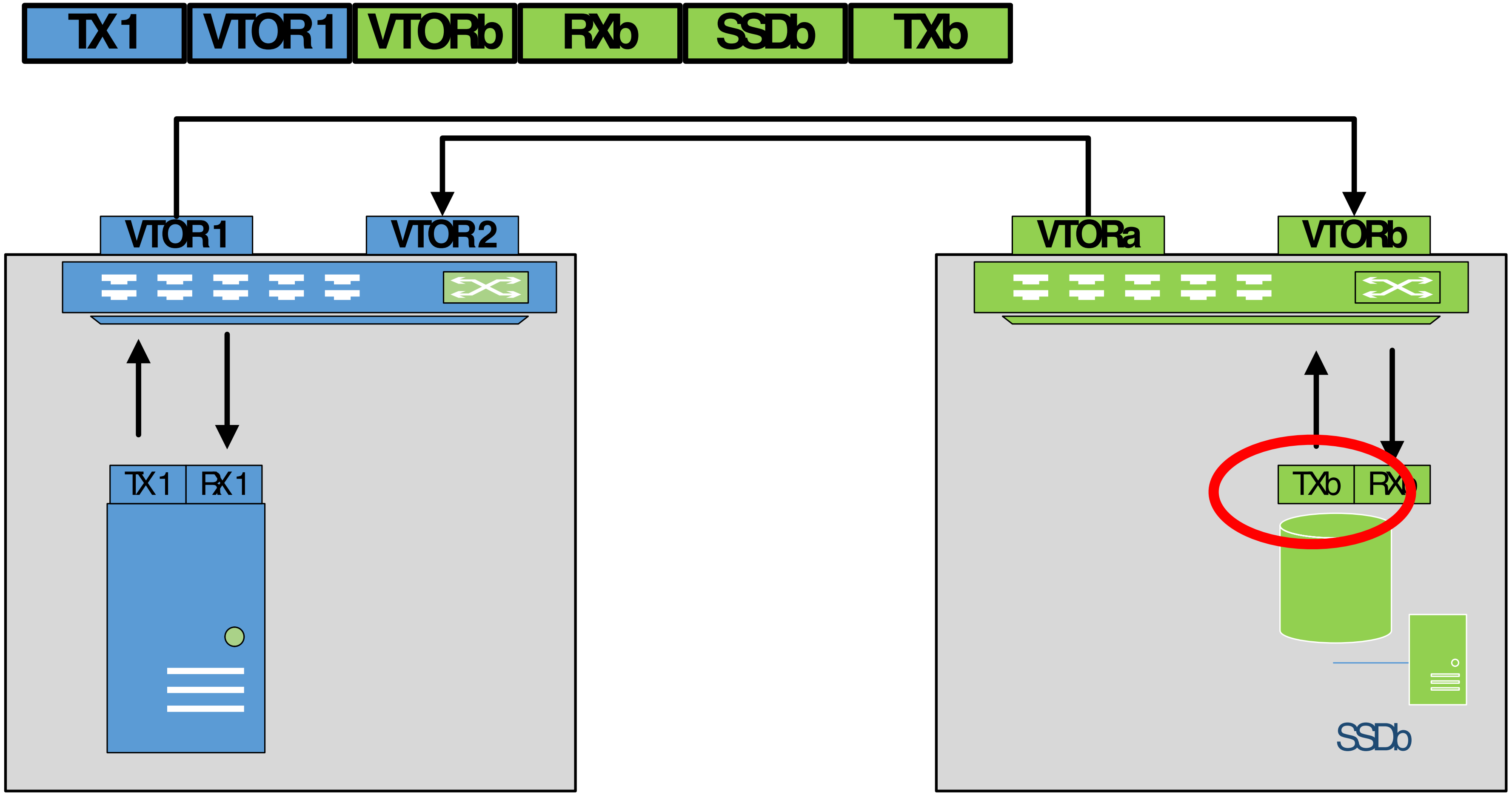
Small customer : one VM accessing storage



Small customer : one VM accessing storage

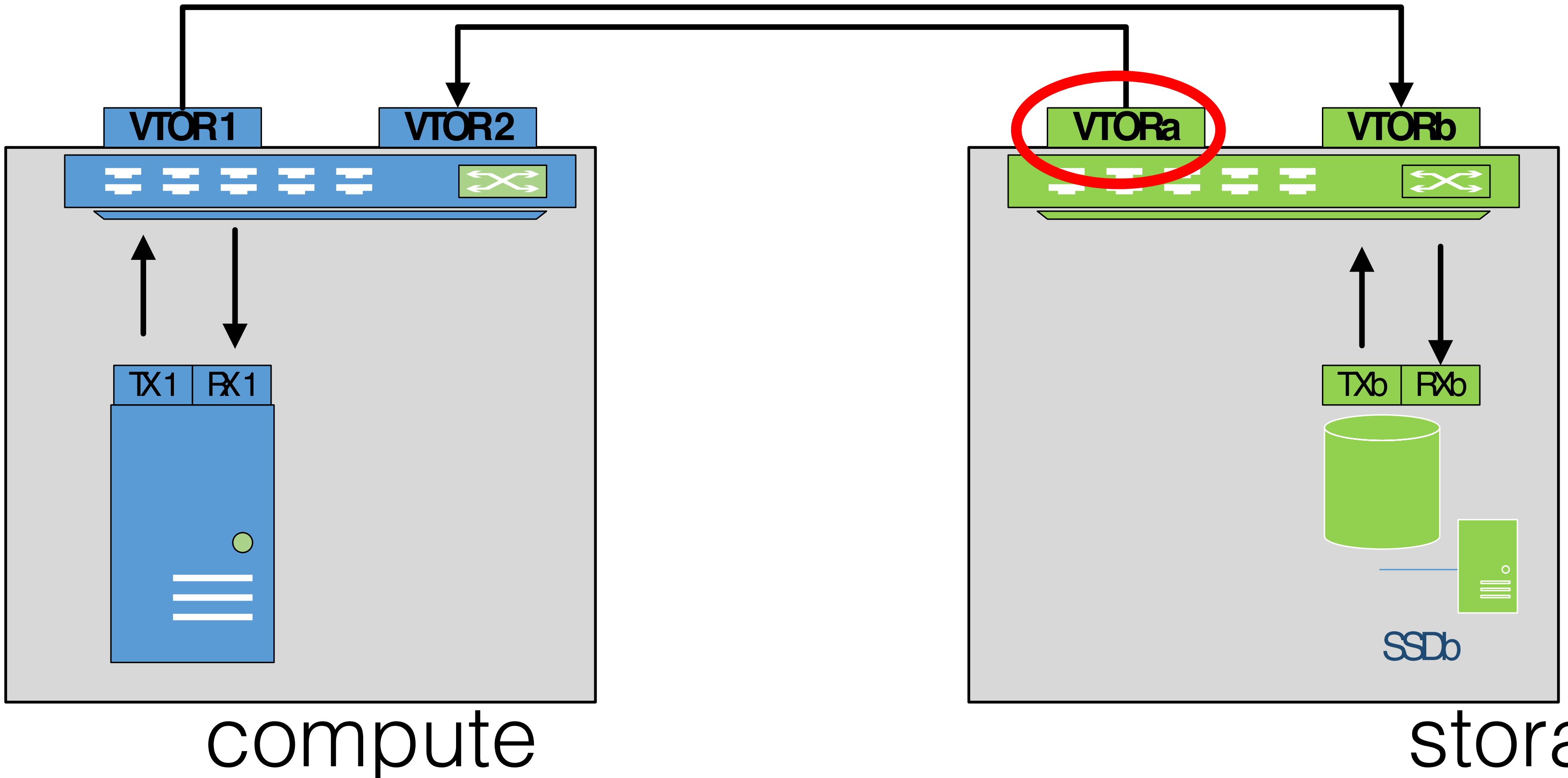


Small customer : one VM accessing storage

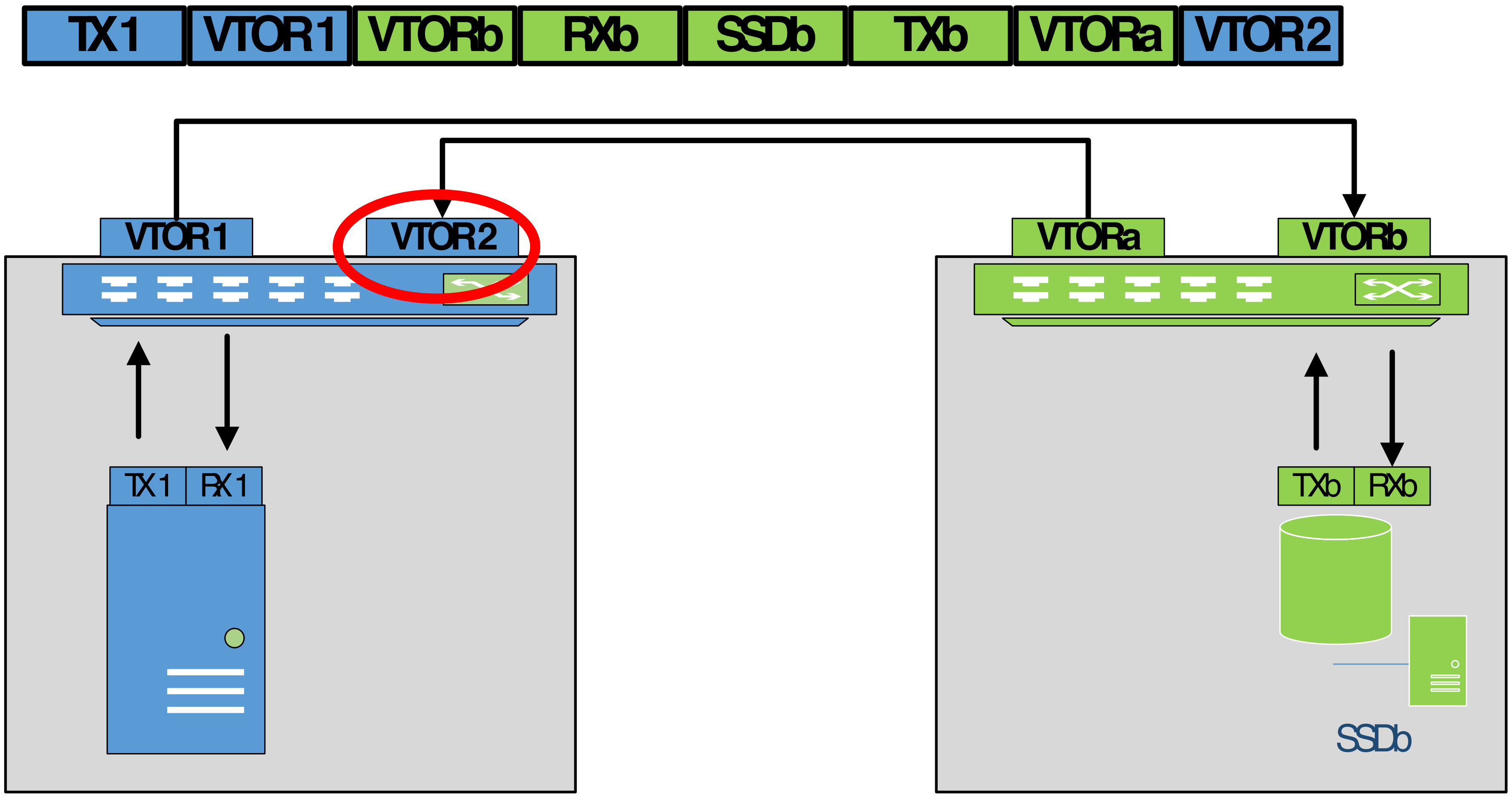


Small customer : one VM accessing storage

TX1	VTOR1	VTORb	RXb	SSDb	TXb	VTORa



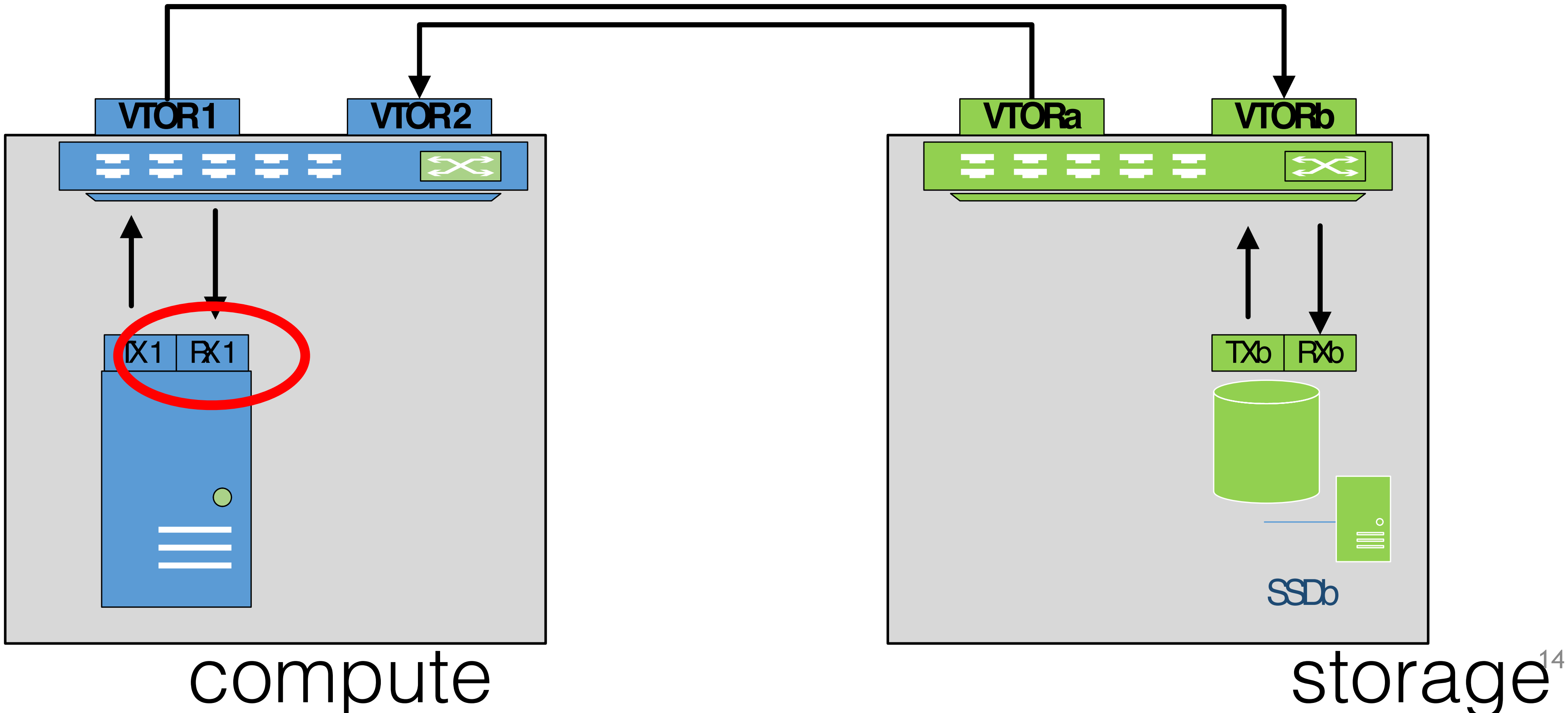
Small customer : one VM accessing storage



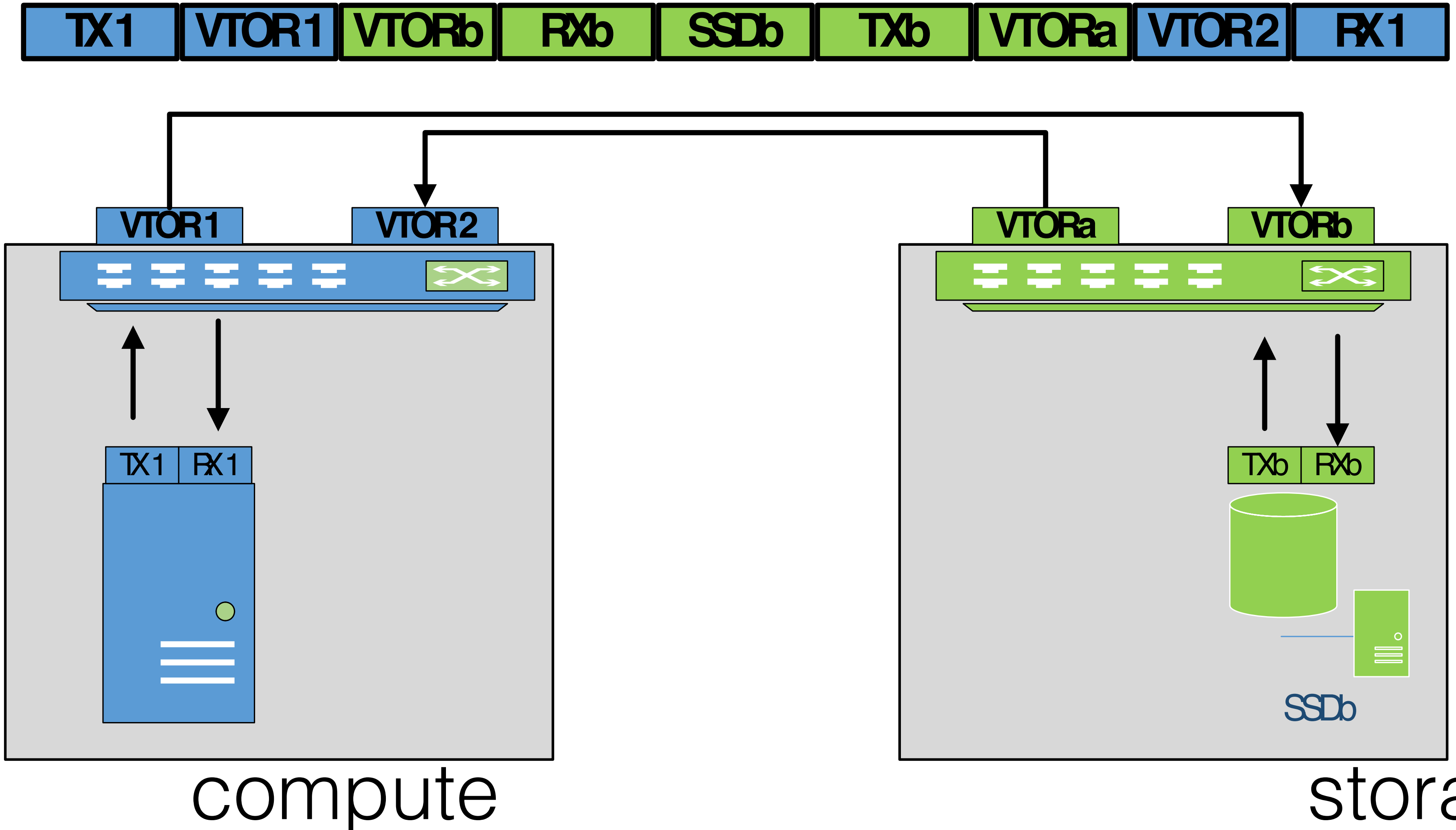
compute storage

Result: a multi-resource “demand vector”

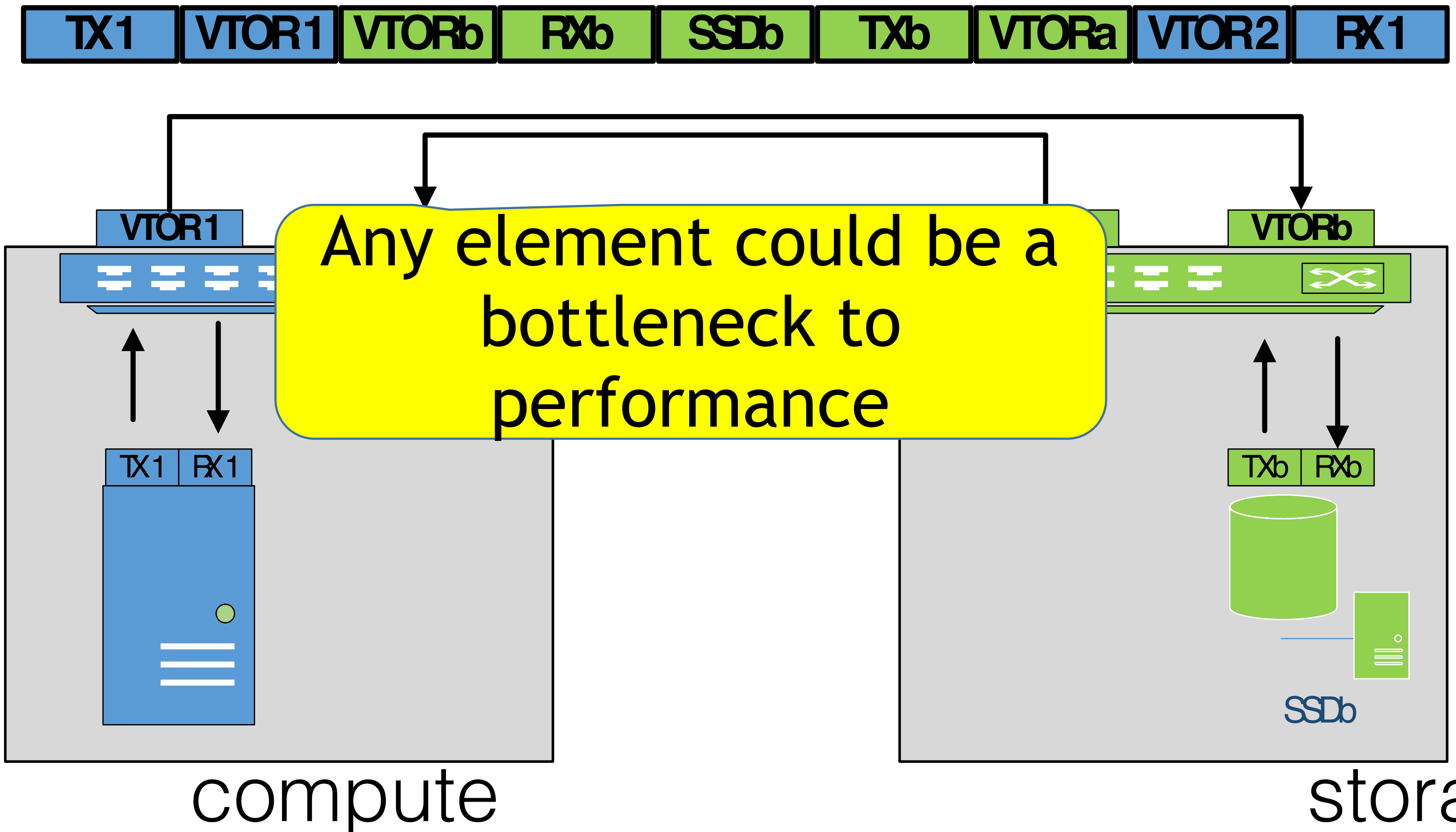
TX1	VTOR1	VTORb	RXb	SSDb	TXb	VTORa	VTOR2	RX1



Encodes resource id and proportions



Encodes resource id and proportions



Demand vectors form a sparse demand matrix

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	-	-	-	-	-	-	-	-
n_1	-	-	-	-	-	-	-	-	-	-
n_2	-	-	-	-	-	-	-	-	-	-
n_3	-	-	-	-	-	-	-	-	-	-
n_4	-	-	-	-	-	-	-	-	-	-
n_5	-	-	-	-	-	-	-	-	-	-
n_6	-	-	-	-	-	-	-	-	-	-
n_7	-	-	-	-	-	-	-	-	-	-
n_8	-	-	-	-	-	-	-	-	-	-
n_9	-	-	-	-	-	-	-	-	-	-

Columns are shared physical resources

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	-	-	-	-	-	-	-	-
n_1	-	-	-	-	-	-	-	-	-	-
n_2	-	-	-	-	-	-	-	-	-	-
n_3	-	-	-	-	-	-	-	-	-	-
n_4	-	-	-	-	-	-	-	-	-	-
n_5	-	-	-	-	-	-	-	-	-	-
n_6	-	-	-	-	-	-	-	-	-	-
n_7	-	-	-	-	-	-	-	-	-	-
n_8	-	-	-	-	-	-	-	-	-	-
n_9	-	-	-	-	-	-	-	-	-	-

Rows are tenants' demand vectors

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	-	-	-	-	-	-	-	-
n_1	-	-	-	-	-	-	-	-	-	-
n_2	-	-	-	-	-	-	-	-	-	-
n_3	-	-	-	-	-	-	-	-	-	-
n_4	-	-	-	-	-	-	-	-	-	-
n_5	-	-	-	-	-	-	-	-	-	-
n_6	-	-	-	-	-	-	-	-	-	-
n_7	-	-	-	-	-	-	-	-	-	-
n_8	-	-	-	-	-	-	-	-	-	-
n_9	-	-	-	-	-	-	-	-	-	-

Shown as fractions of a resource

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	-	-	-	-	-	-	-	-	-	-
n_2	-	-	-	-	-	-	-	-	-	-
n_3	-	-	-	-	-	-	-	-	-	-
n_4	-	-	-	-	-	-	-	-	-	-
n_5	-	-	-	-	-	-	-	-	-	-
n_6	-	-	-	-	-	-	-	-	-	-
n_7	-	-	-	-	-	-	-	-	-	-
n_8	-	-	-	-	-	-	-	-	-	-
n_9	-	-	-	-	-	-	-	-	-	-

Large and very sparse matrix

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-									13
n_4	-									31
n_5	-									13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56

DC matrix 100K by 100K
Rows mostly empty

Provider has multi-resource allocation problem

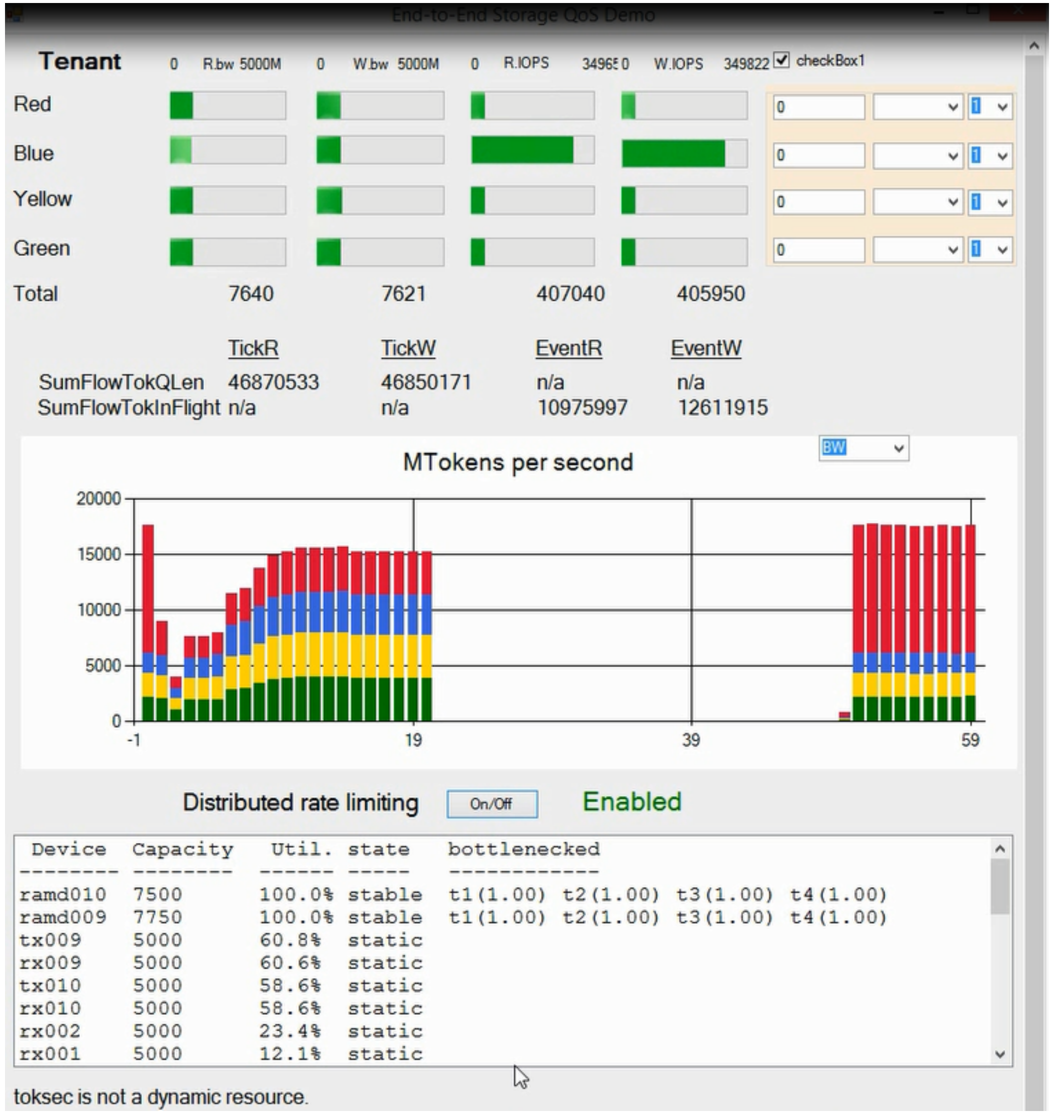
- Goal: maintain acceptable service level for all tenants
  - Acceptable means always “willing to pay”
  - Avoid abrupt performance collapse for any tenant
  - Assuming aggressive (noisy) neighbors and oversubscription
- DC-DRF builds on existing multi-resource algorithms
  - DRF [Ghodsi et al, NSDI’11]
  - EDRF [Parkes et al., EC2012]
- Challenging at DC scale: EDRF iterates and is

Systems aspects

Systems challenges

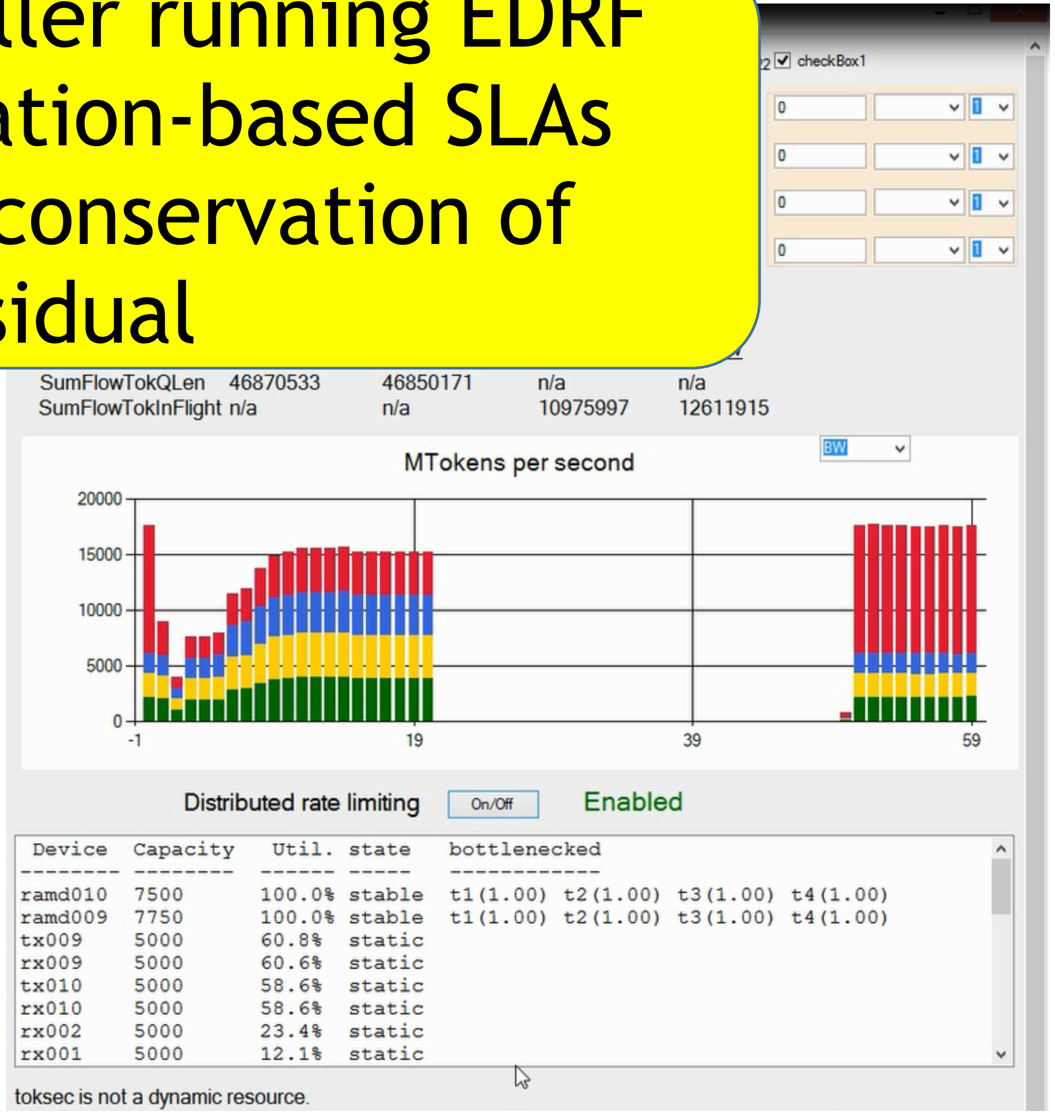
- How to capture multi-resource demand vectors?
- How to enforce multi-resource allocations?
- DRF implies central SDN-like controller – good or bad?
    - Good: Simpler algorithm and global view
    - Bad: EDRF at Public Cloud DC scale

SIGCOMM 2015 demonstration

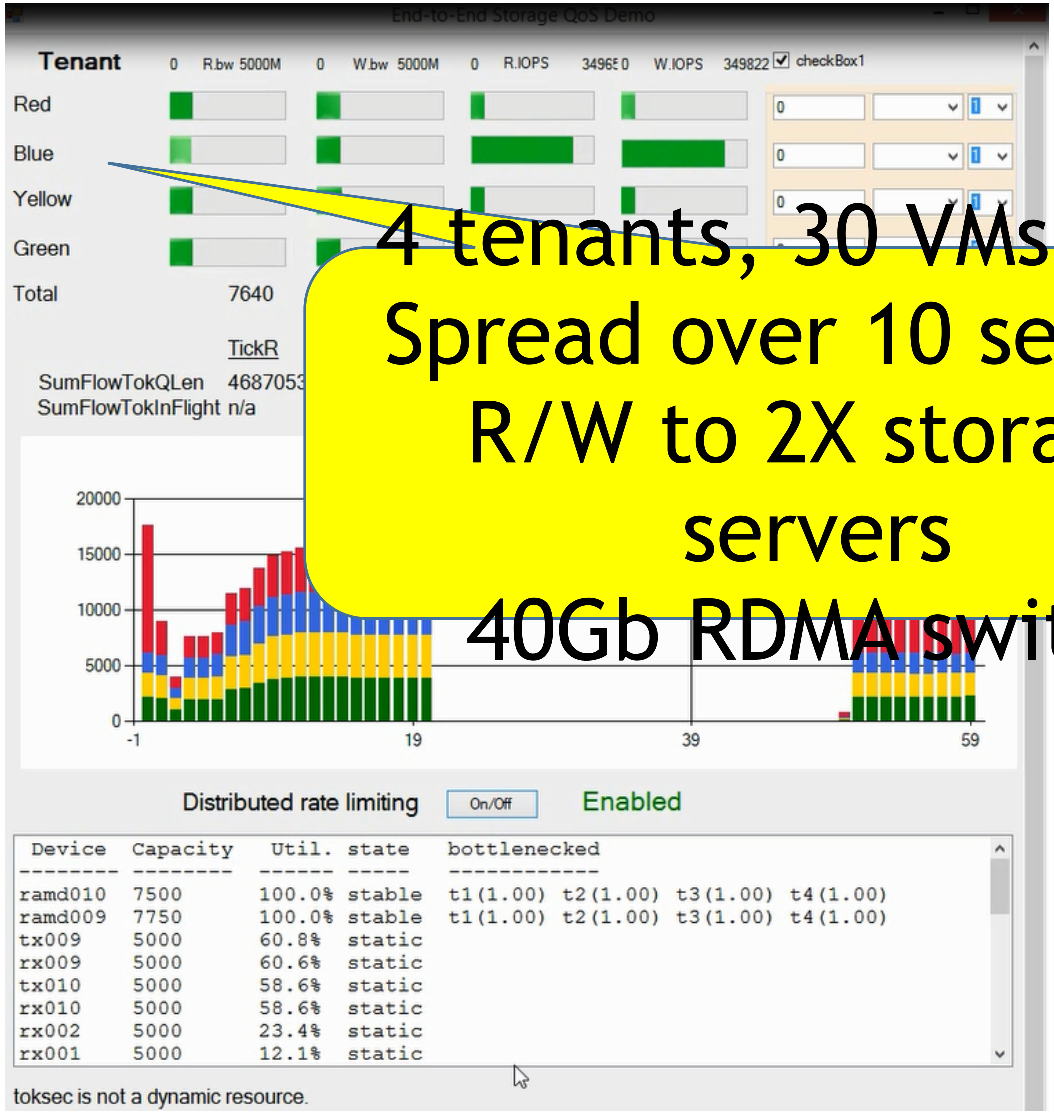


SIGCOMM 2015 demonstration

Central controller running EDRF
Pass1: reservation-based SLAs
Pass2: work conservation of residual



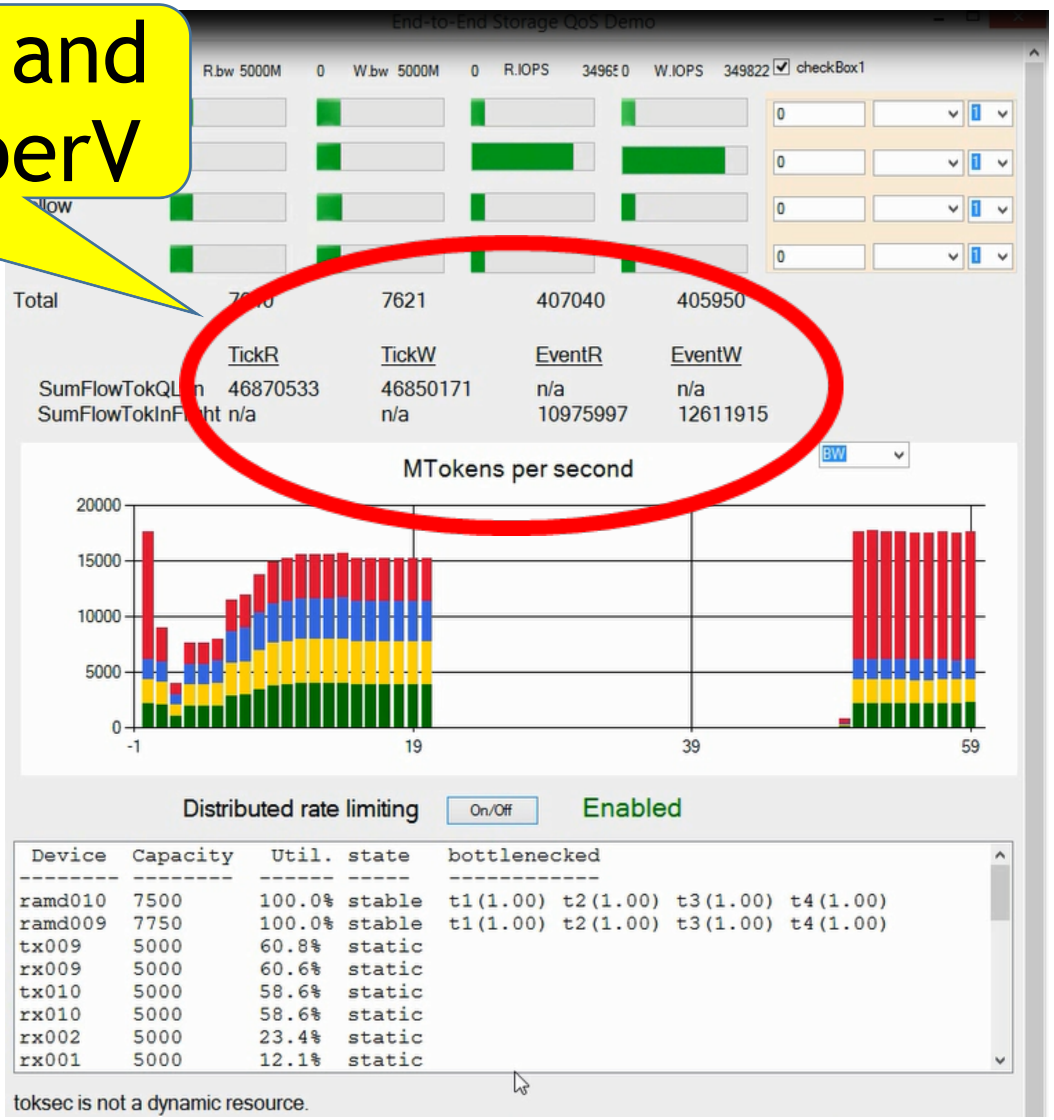
SIGCOMM 2015 demonstration



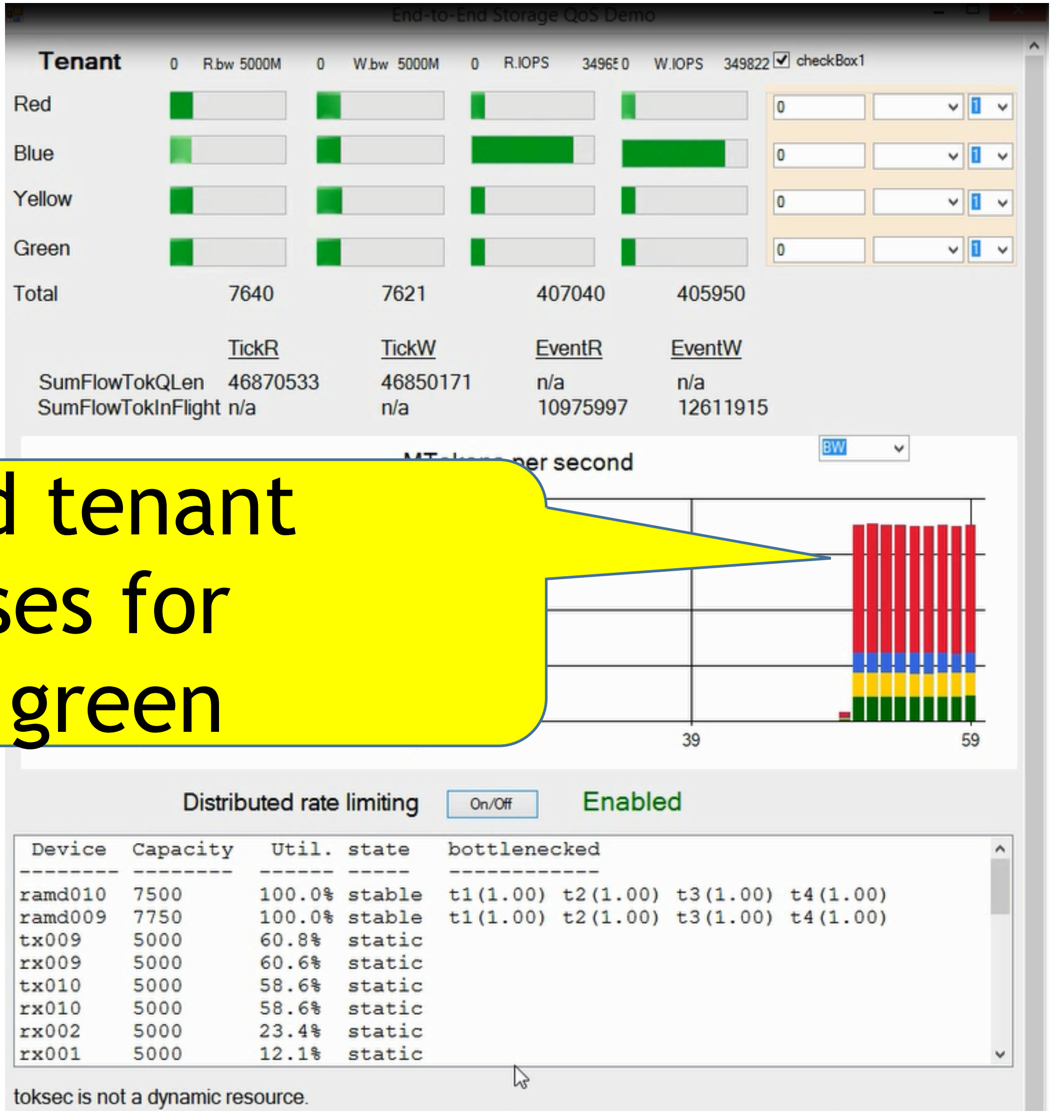
4 tenants, 30 VMs each
Spread over 10 servers
R/W to 2X storage servers
40Gb RDMA switch

SIGCOMM 2015 demonstration

Demand estimation and enforcement in HyperV



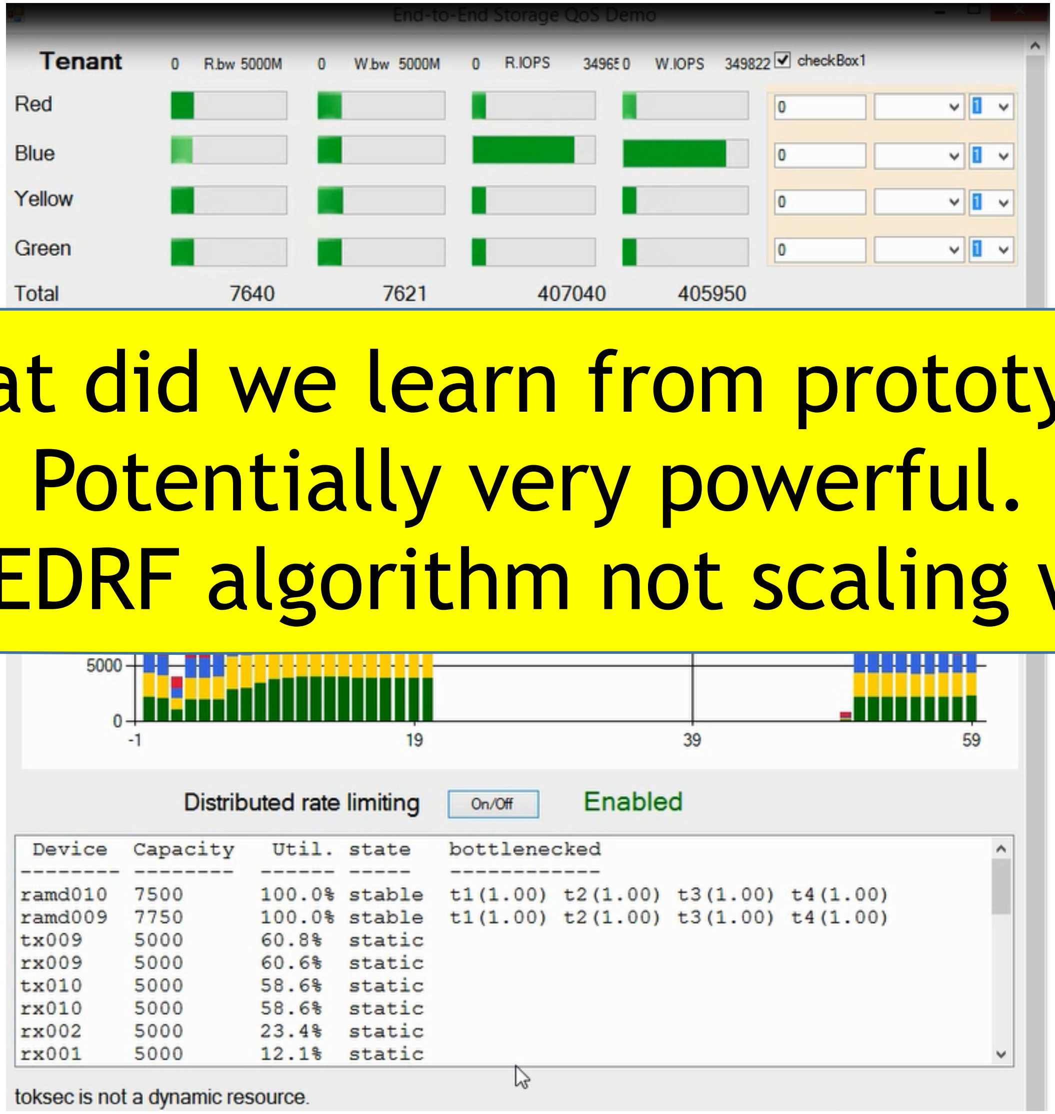
SIGCOMM 2015 demonstration



Aggressive red tenant
Perf. collapses for
blue,yellow,green

video

SIGCOMM 2015 demonstration



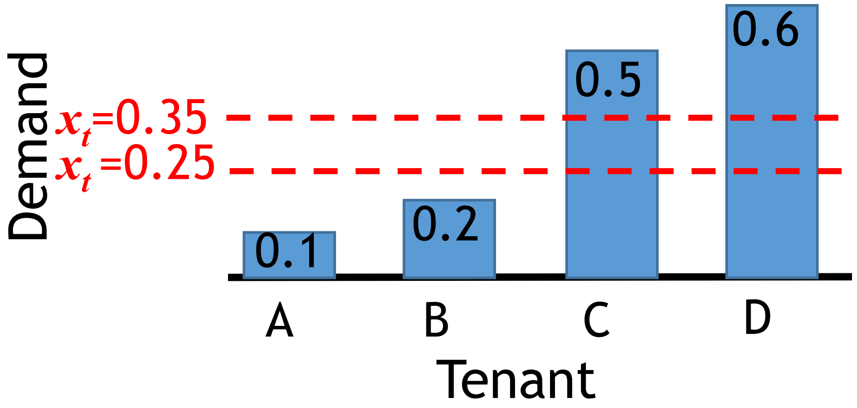
What did we learn from prototype?
Potentially very powerful.
But EDRF algorithm not scaling well.

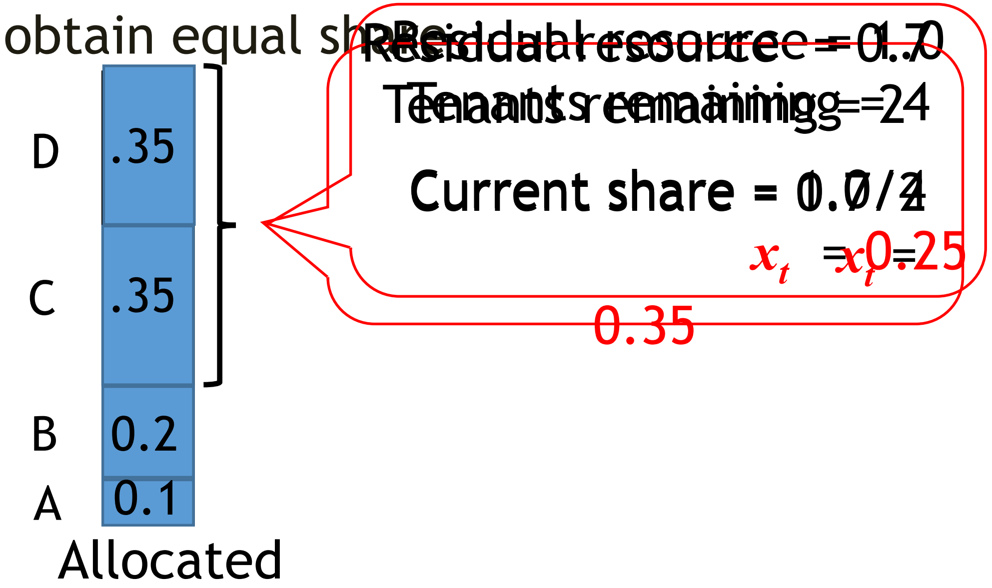
The algorithms

- to understand DC-DRF first understand EDRF
- to understand DRF first understand max-min

Max-Min fairness : mice before elephants

- Maximize the minimum allocation across competing tenants
- Allocate fractions of a single shared resource based on demand
  - No tenant gets a larger fraction than its demand
  - Tenants with unsatisfiable demand obtain equal share





How to handle multiple resources?

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-	.41	.20	.12	.13	.09	.23	1.0	.23	.13
n_4	-	1.0	-	.30	-	.23	.55	-	-	.31
n_5	-	.41	.09	.12	1.0	.64	.23	.20	.13	.13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56

Dominant Resource Fairness (DRF)

- For each tenant identifies its Dominant Resource
  - The resource of which it demands the largest fraction
- Apply max-min fairness across dominant shares
  - Maximize smallest dominant share in system
  - Then second smallest, and so on…
  - Think : find the smallest mouse across all columns

Demand vectors normalized by Dominant Resource

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-	.41	.20	.12	.13	.09	.23	1.0	.23	.13
n_4	-	1.0	-	.30	-	.23	.55	-	-	.31
n_5	-	.41	.09	.12	1.0	.64	.23	.20	.13	.13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56

Maximize (max-min) smallest dominant share

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-	.41	.20	.12	.13	.09	.23	1.0	.23	.13
n_4	-	1.0	-	.30	-	.23	.55	-	-	.31
n_5	-	.41	.09	.12	1.0	.64	.23	.20	.13	.13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56
x_{tr} =	.37	.33	.35	.45	.35	.40	.33	.63	.246	.30

Find residual resource with smallest x_{tr}

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-	.41	.20	.12	.13	.09	.23	1.0	.23	.13
n_4	-	1.0	-	.30	-	.23	.55	-	-	.31
n_5	-	.41	.09	.12	1.0	.64	.23	.20	.13	.13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56
x_{tr} =	.37	.33	.35	.45	.35	.40	.33	.63	.246	.30

Use x_{r8} to allocate at every resource

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-	.41	.20	.12	.13	.09	.23	1.0	.23	.13
n_4	-	1.0	-	.30	-	.23	.55	-	-	.31
n_5	-	.41	.09	.12	1.0	.64	.23	.20	.13	.13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56
x_{tr} =	.37	.33	.35	.45	.35	.40	.33	.63	.246	.30

Eliminate r_8 if residual capacity hits zero

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-	.41	.20	.12	.13	.09	.23	1.0	.23	.13
n_4	-	1.0	-	.30	-	.23	.55	-	-	.31
n_5	-	.41	.09	.12	1.0	.64	.23	.20	.13	.13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56
x_{tr} =	.37	.33	.35	.45	.35	.40	.33	.63	.246	.30

And eliminate tenants demanding r_8

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-	.41	.20	.12	.13	.09	.23	1.0	.23	.13
n_4	-	1.0	-	.30	-	.23	.55	-	-	.31
n_5	-	.41	.09	.12	1.0	.64	.23	.20	.13	.13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56
x_{tr} =	.37	.33	.35	.45	.35	.40	.33	.63	.246	.30

Next round: find new smallest x_{tr} and so on…

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-	.41	.20	.12	.13	.09	.23	1.0	.23	.13
n_4	-	1.0	-	.30	-	.23	.55	-	-	.31
n_5	-	.41	.09	.12	1.0	.64	.23	.20	.13	.13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56

result : allocation matrix

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	.35	-	-	-	-	-	-	.32
n_1	.23	-	.12	-	-	-	-	-	.25	-
n_2	.13	.25	-	.07	.08	.06	.14	-	.14	.08
n_3	-	.10	.05	.03	.03	.02	.06	.25	.06	.03
n_4	-	.35	-	.10	-	.08	.19	-	-	.11
n_5	-	.10	.02	.03	.25	.16	.06	.05	.03	.03
n_6	.08	-	.02	.03	.25	.16	.06	.05	.03	.03
n_7	-	-	-	-	-	-	.35	-	-	.20
n_8	-	-	.14	.16	.05	.08	.03	.02	.25	.06
n_9	.22	.07	.11	.16	.05	.08	.03	.02	.25	.14

result : allocation matrix

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	.35	-	-	-	-	-	-	.32
n_1	.23	-	.12	-	-	-	-	-	.25	-
n_2	.13	.25	-	.07	.08	.06	.14	-	.14	.08
n_6	.08	-	.02	.03	.25	.16	.06	.05	.03	.03
n_7	-	-	-	-	-	-	.35	-	-	.20
n_8	-	-	.14	.16	.05	.08	.03	.02	.25	.06
n_9	.22	.07	.11	.16	.05	.08	.03	.02	.25	.14

Issue: EDRF is iterative. Sparsity implies slow elimination of tenants.

DC-DRF algorithm

Goal

- Monitor and adjust shares at 10-30 second intervals
  - Resource demands variation in datacentre traces [Angel et al., OSDI’14]
  - Using demands plausibly realistic of Public Cloud DC

DC-DRF: two tactics to improve scalability

1. Algorithmic: extending EDRF
   - Operate to a time deadline chosen by operator (“control interval”)
   - Variable degree of approximation: trading resource utilization for time
2. HPC: maximize rate of computation
   - Parallel where possible
   - Optimize for thread and NUMA locality
   - SIMD vector instructions

Algorithm: inner and outer loops

OuterLoop(time t)   // runs one per control interval

Initialize demand matrix for this interval

Set approximation control variable   [0, 1]

  timeOut = InnerLoop()

  if elapsed time exceeds t then return true

  Eliminate a resource when 1- full // e.g. =0.01 at 99%

  Resources and tenants eliminated earlier and in fewer rounds

if (timeOut) then increase() else decrease()

Tactic #2 : HPC

- Goal: minimize value of required to meet deadline
- Minimize error due to approximation and maximize utilization
- Do this by extracting as much perf as we can from the platform.

Parallelism : resource tiles over large sparse matrix

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-	.41	.20	.12	.13	.09	.23	1.0	.23	.13
n_4	-	1.0	-	.30	-	.23	.55	-	-	.31
n_5	-	.41	.09	.12	1.0	.64	.23	.20	.13	.13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56

Alternating with tenant tiles

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31
n_3	-	.41	.20	.12	.13	.09	.23	1.0	.23	.13
n_4	-	1.0	-	.30	-	.23	.55	-	-	.31
n_5	-	.41	.09	.12	1.0	.64	.23	.20	.13	.13
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56

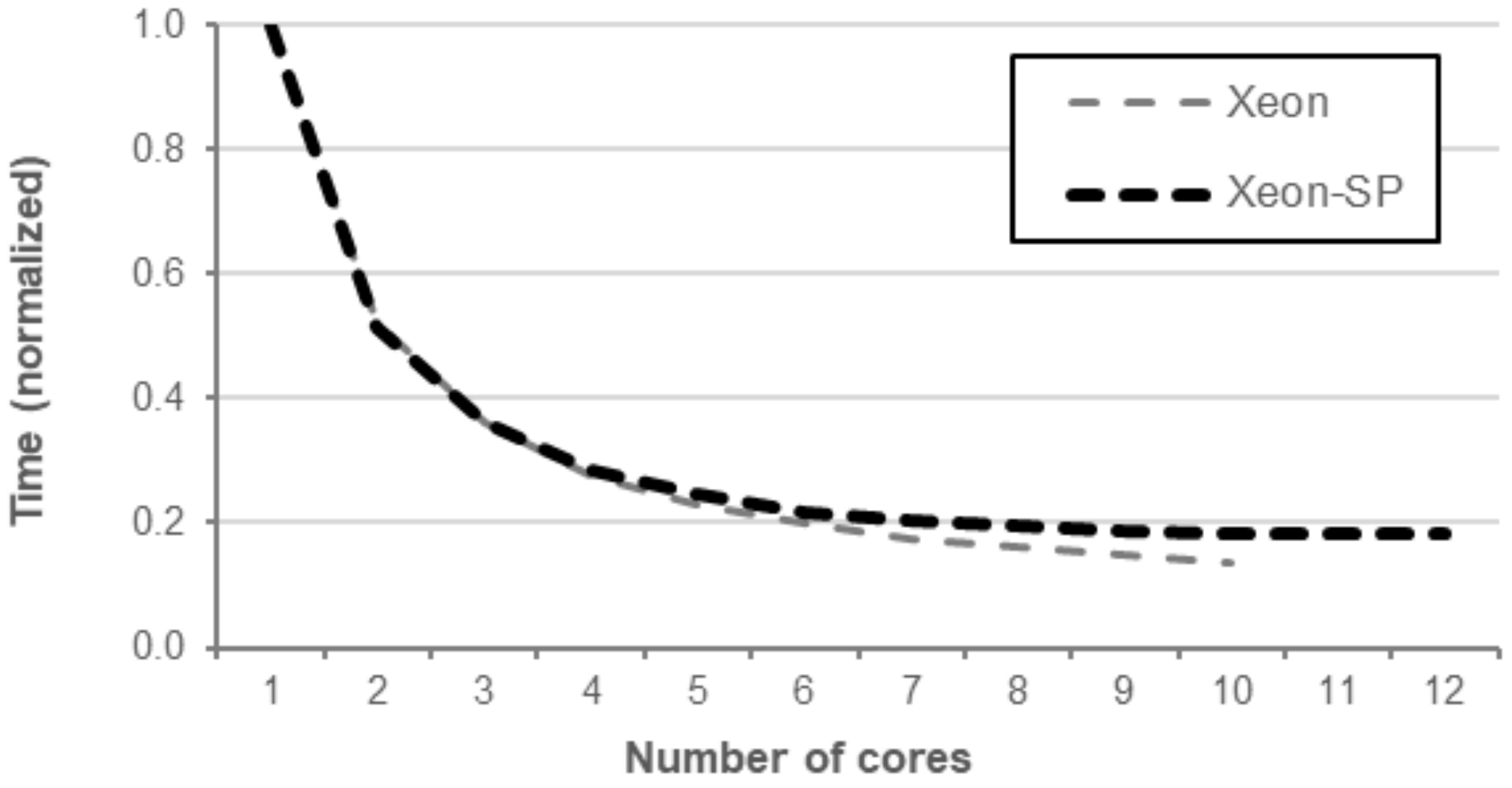
Alternating with tenant tiles

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_0	-	-	1.0	-	-	-	-	-	-	.92
n_1	.95	-	.47	-	-	-	-	-	1.0	-
n_2	.54	1.0	-	.30	.33	.23	.55	-	.56	.31

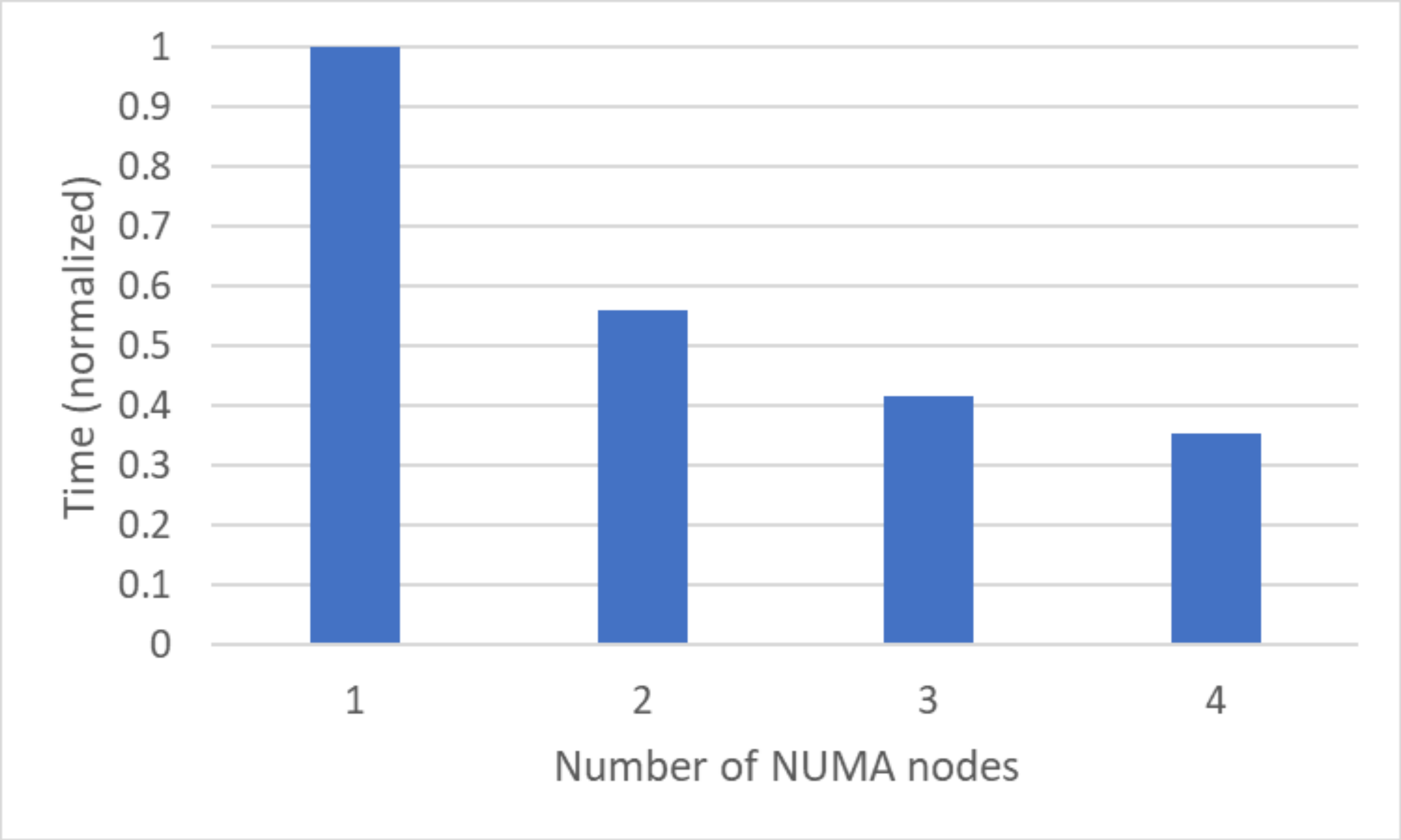
Carefully cache-aligned memory and bespoke mem barriers make this lock-free.

	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
n_6	.32	-	.09	.12	1.0	.64	.23	.20	.13	.13
n_7	-	-	-	-	-	-	1.0	-	-	.57
n_8	-	-	.56	.64	.20	.32	.13	.09	1.0	.23
n_9	.90	.27	.45	.64	.20	.32	.13	.09	1.0	.56

Single socket parallelisation



NUMA-aware aggregation and mem allocation



SIMD: AVX512 vector instruction set

```
__mm512_vindex vindex_512 = _MM512_LOAD_VINDEX(*ptr);
__m512r mu_tr = _mm512_i32gather_pr(vindex_512,pScratchR);
mu_tr = _mm512_add_pr(mu_tr, A_irt);
_mm512_mask_i32scatter_pr(pScratchR, m, vindex_512, mu_tr);
```

SIMD: AVX512 vector instruction set

```
__mm512_vindex vindex_512 = _MM512_LOAD_VINDEX(*ptr);
__m512r mu_tr = _m  m512_i32gather_pr(vindex_512,pScratchR);
mu_tr = _mm512_add_p  mu_tr, A_irt);
_mm512_mask_i32scatte  pScratchR, m, vindex_512, mu_tr);
```

Identify 16 values in 100K array

SIMD: AVX512 vector instruction set

```
__mm512_vindex vindex_512 = _MM512_LOAD_VINDEX(*ptr);
__m512r mu_tr = _mm512_i32gather_pr(vindex_512,pScratchR);
mu_tr = _mm512_add_pr(mu_tr, A_irt);
_mm512_mask_i32scatter_pr(pScratchR, m, vindex_512, mu_tr);
```

Pull them all into 512-bit register.

SIMD: AVX512 vector instruction set

```
__mm512_vindex vindex_512 = _MM512_LOAD_VINDEX(*ptr);
__m512r mu_tr = _mm512_i32gather_pr(vindex_512,pScratchR);
mu_tr = _mm512_add_pr(mu_tr, A_irt);
_mm512_mask_i32scatter_pr(pScratchR, m, vindex_512, mu_tr);
```

Perform arithmetic on them all at once.

SIMD: AVX512 vector instruction set

```
__mm512_vindex vindex_512 = _MM512_LOAD_VINDEX(*ptr);
__m512r mu_tr = _mm512_i32gather_pr(vindex_512,pScratchR);
mu_tr = _mm512_add_pr(mu_tr, A_irt);
_mm512_mask_i32scatter_pr(pScratchR, m, vindex_512, mu_tr);
```

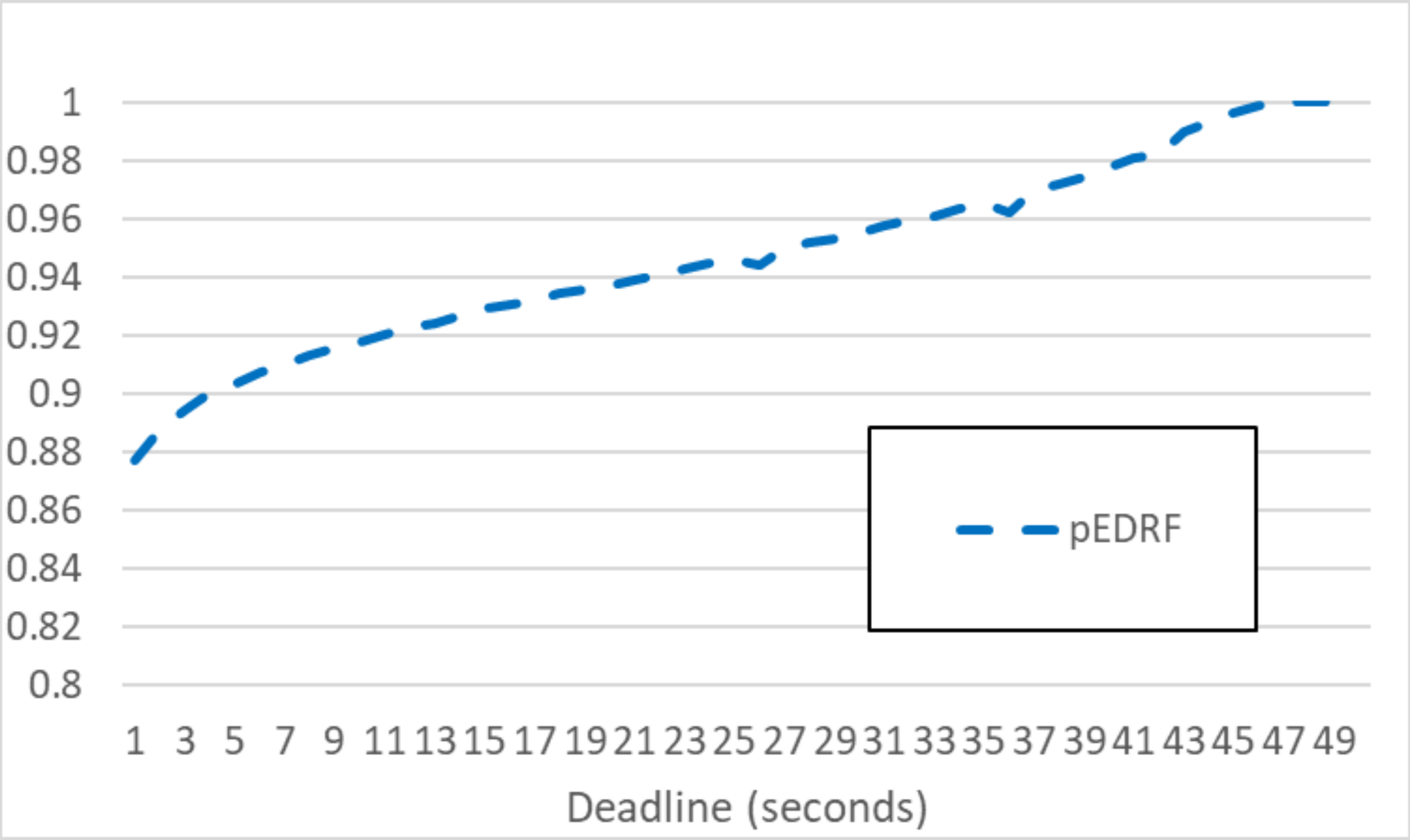
Scatter them back into 100K array

Evaluation

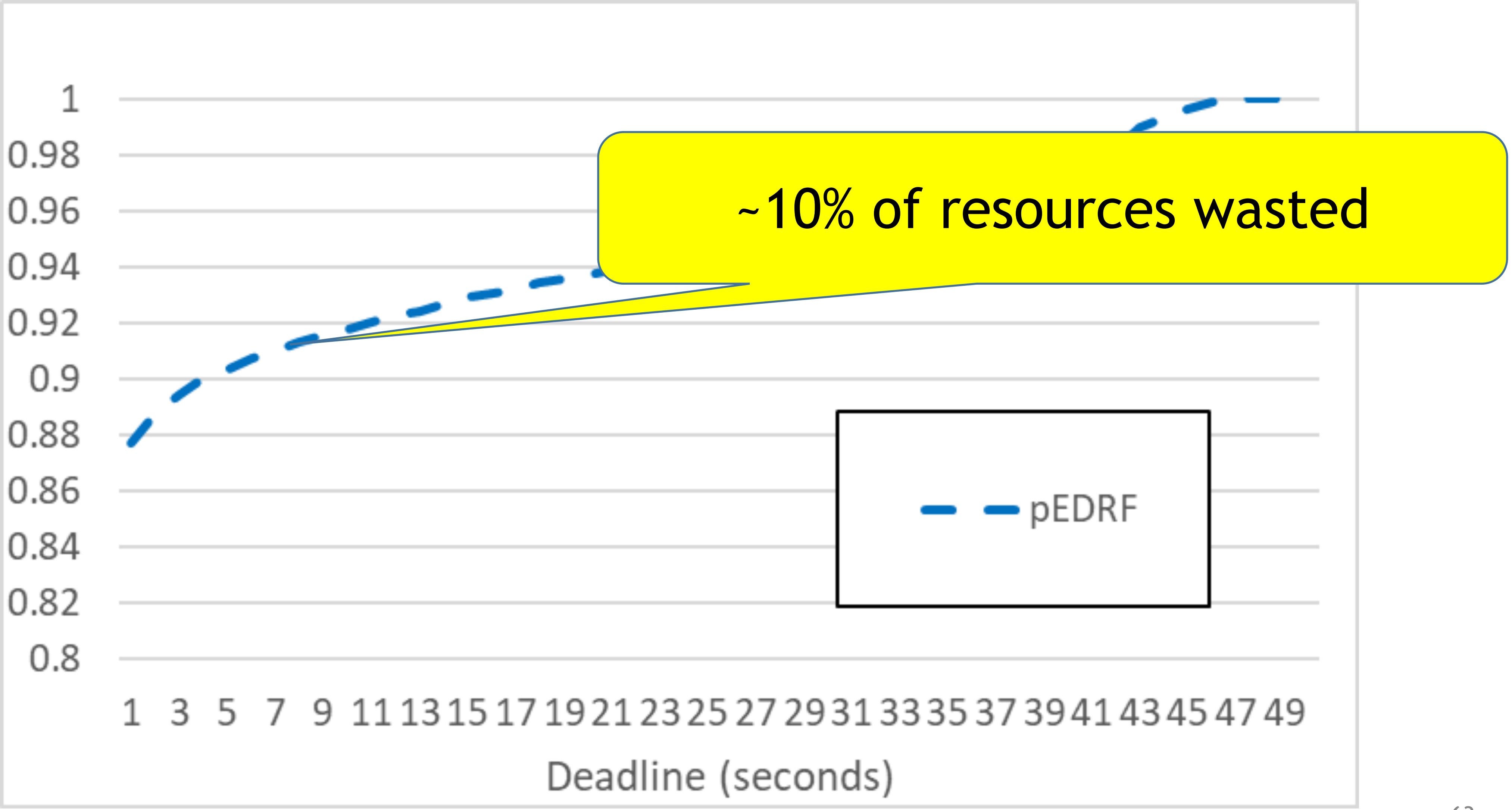
Approach

- Method: synthetic demands based on Azure traces [Cortez, SOSP'17]
  - Synthetic demand for 100K resources X 1M tenants
  - Demand vector sizes [2,128] from truncated Gaussian (most tenants small)
- Deadline for DC-DRF: 8 seconds
- Compare to baseline single-threaded EDRF in unbounded time
- Show overall results and breakdown
  - DC-DRF: both approximation and HPC
  - sDC-DRF: approximation only
  - pEDRF: HPC only, finish at deadline

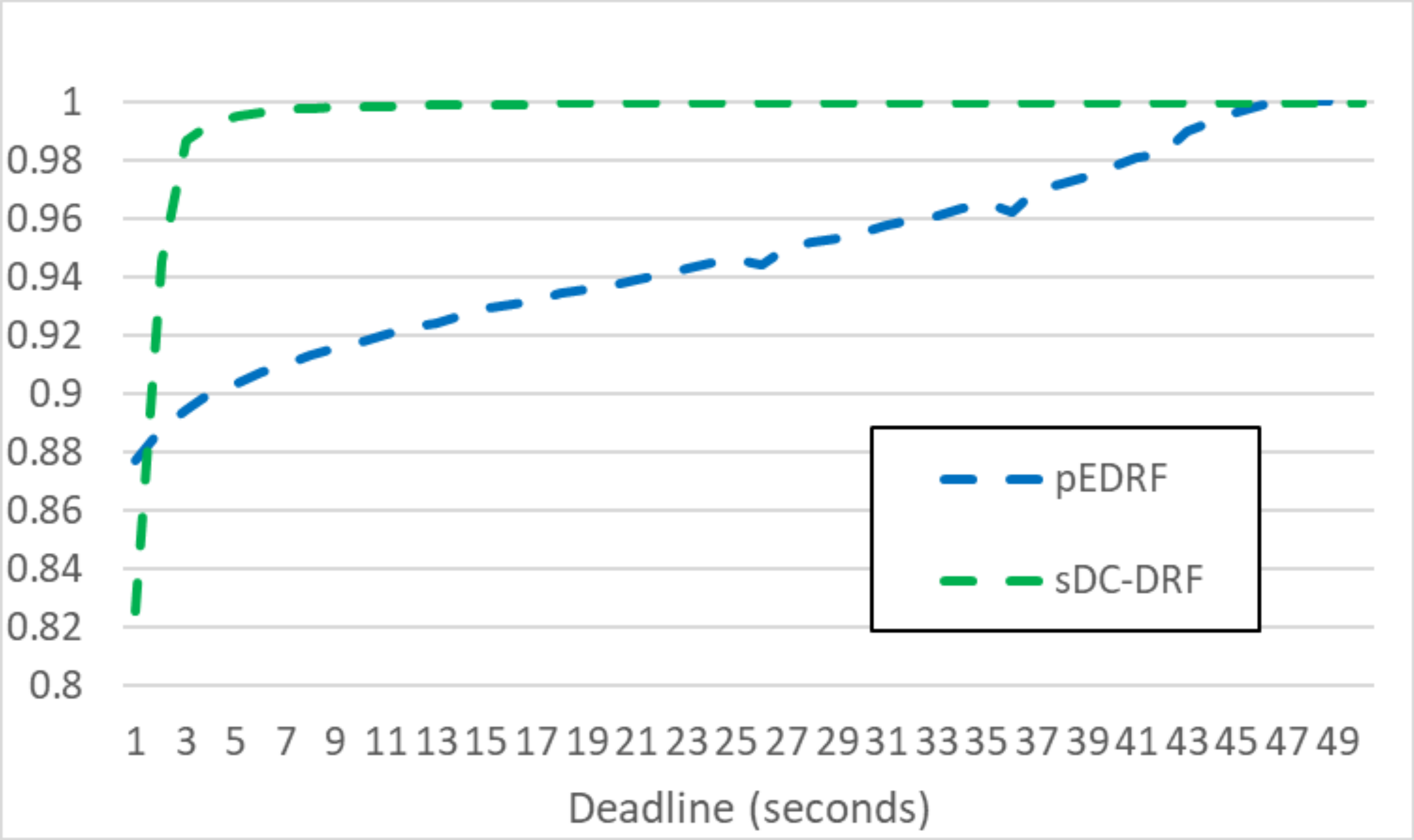
Utilization relative to baseline



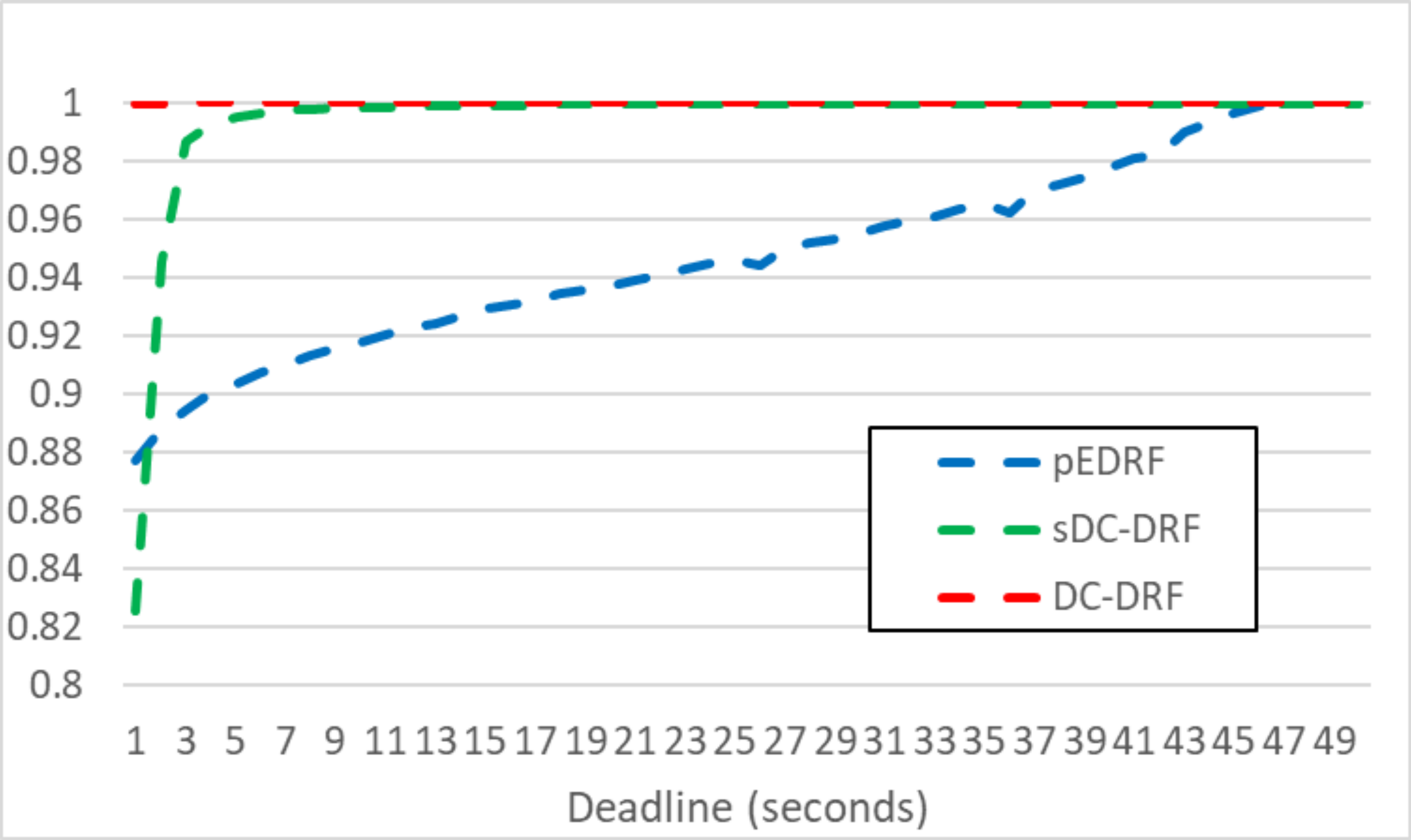
Utilization relative to baseline



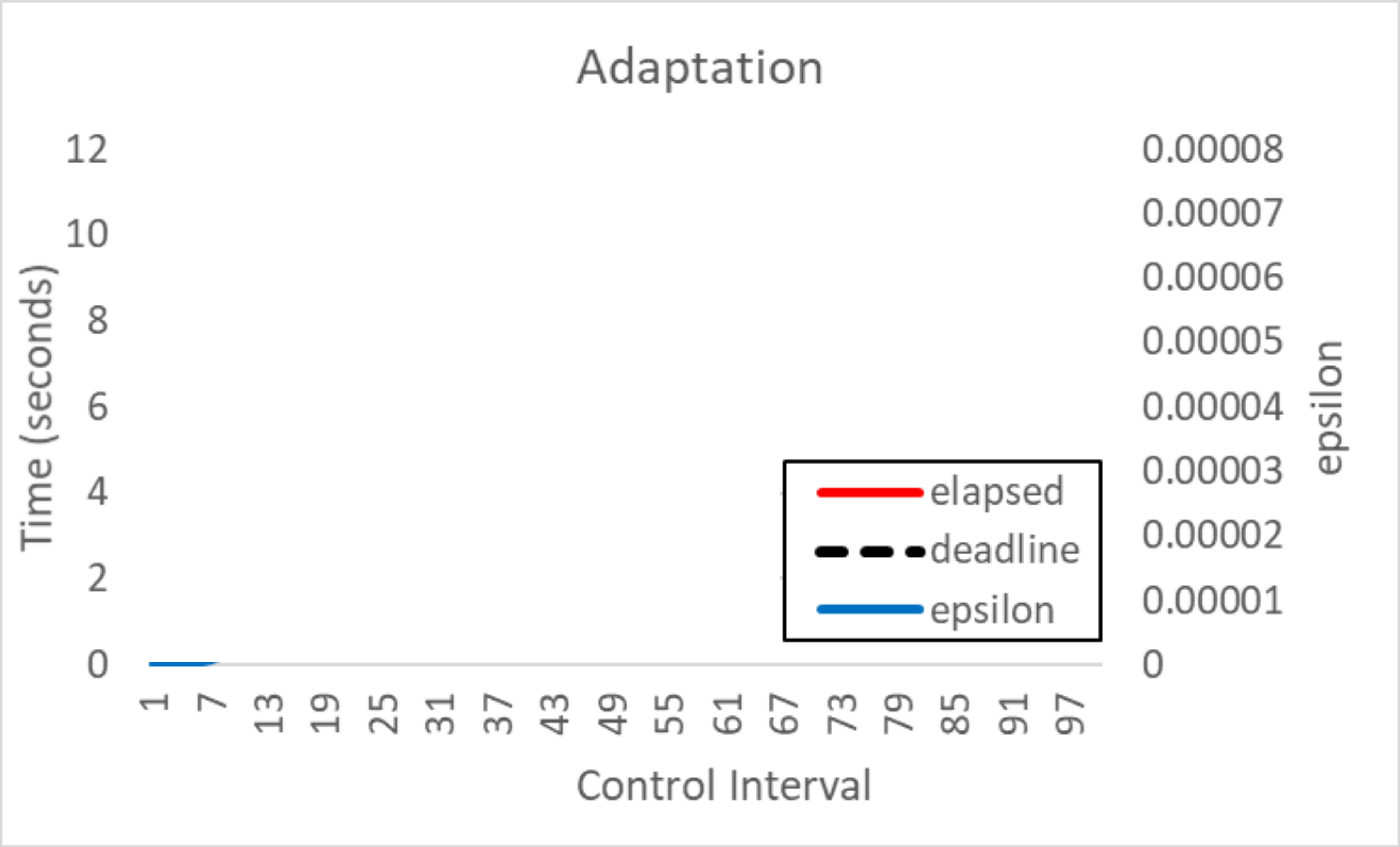
Utilization relative to baseline



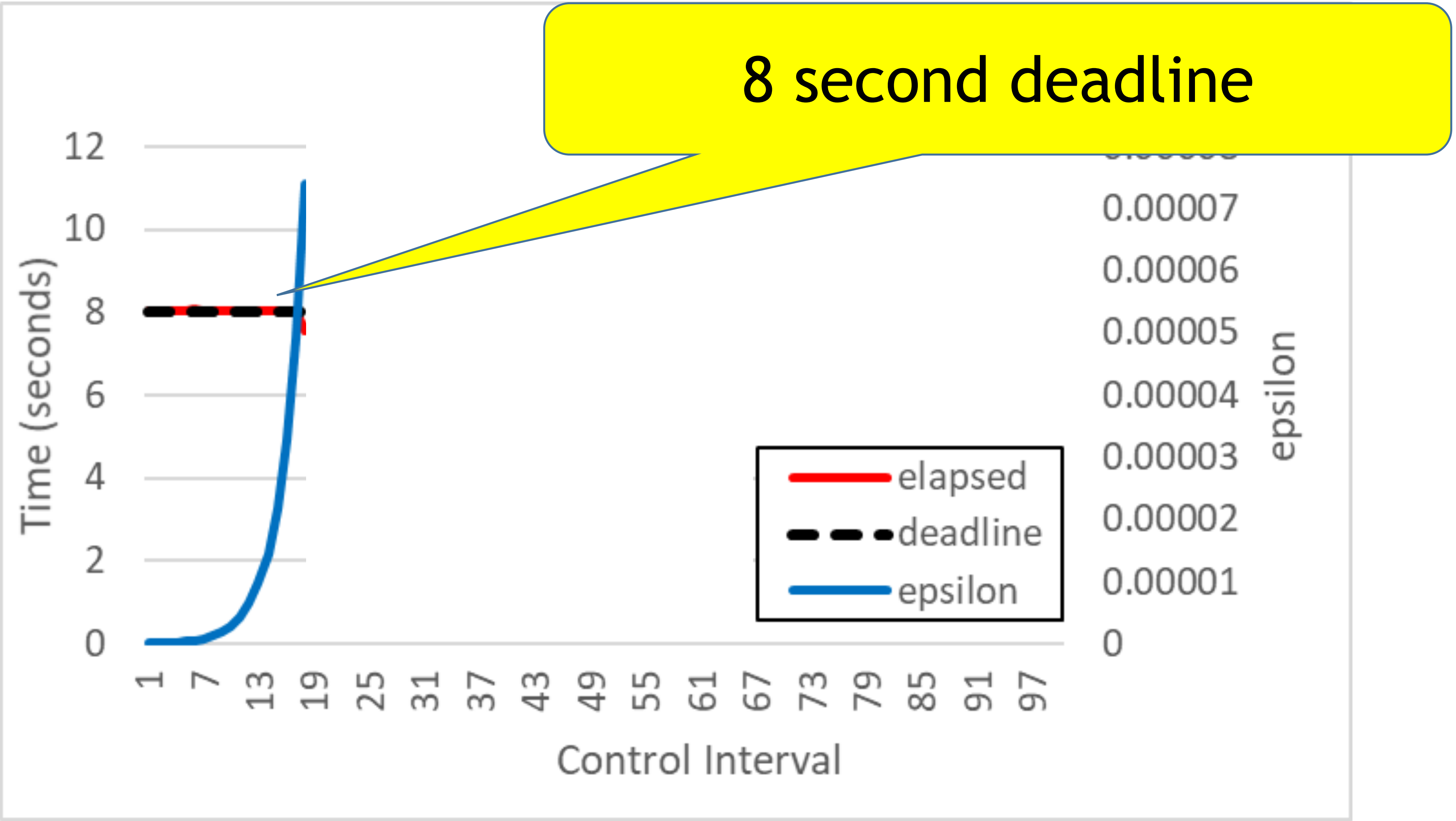
Utilization relative to baseline



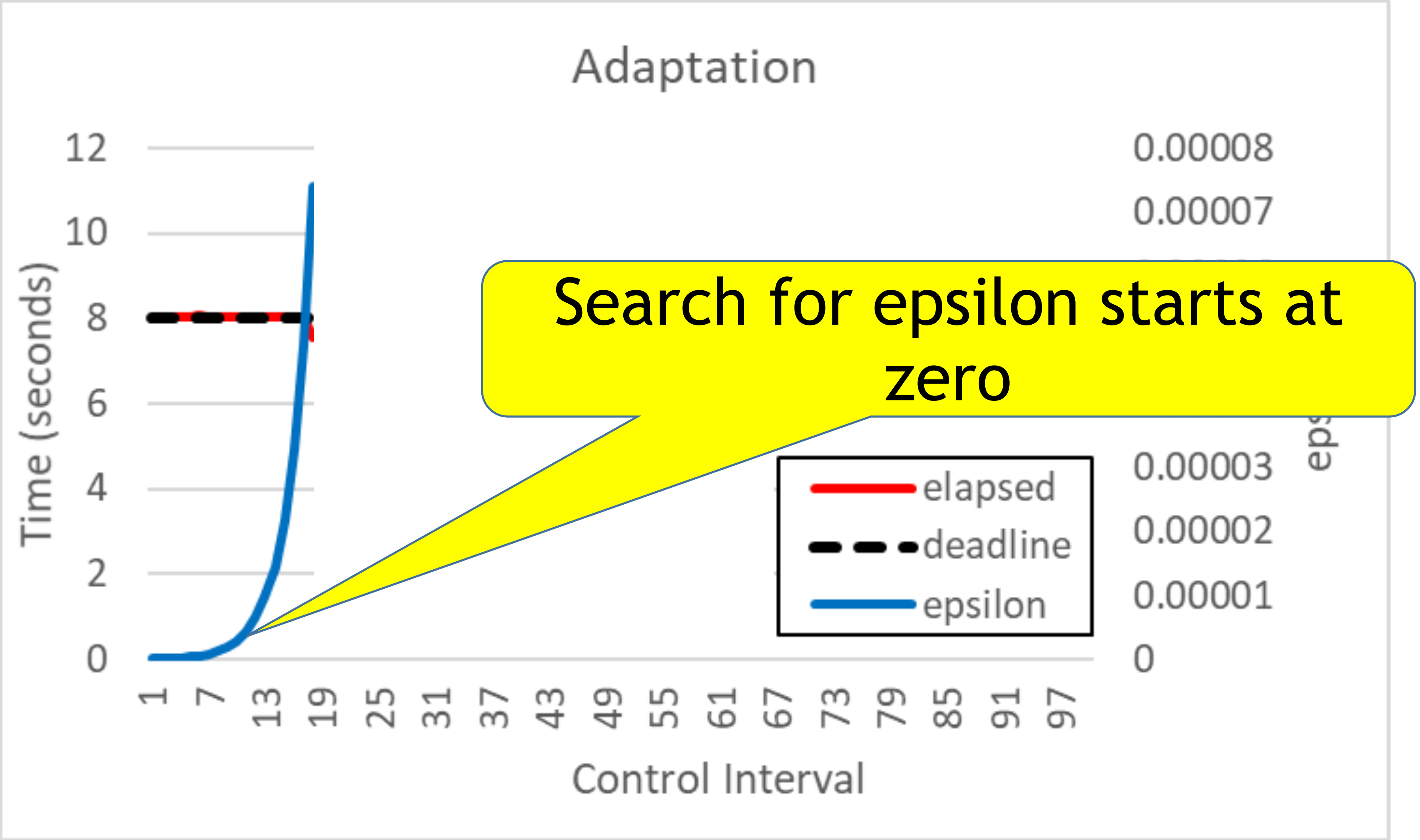
Outer loop : adapting epsilon



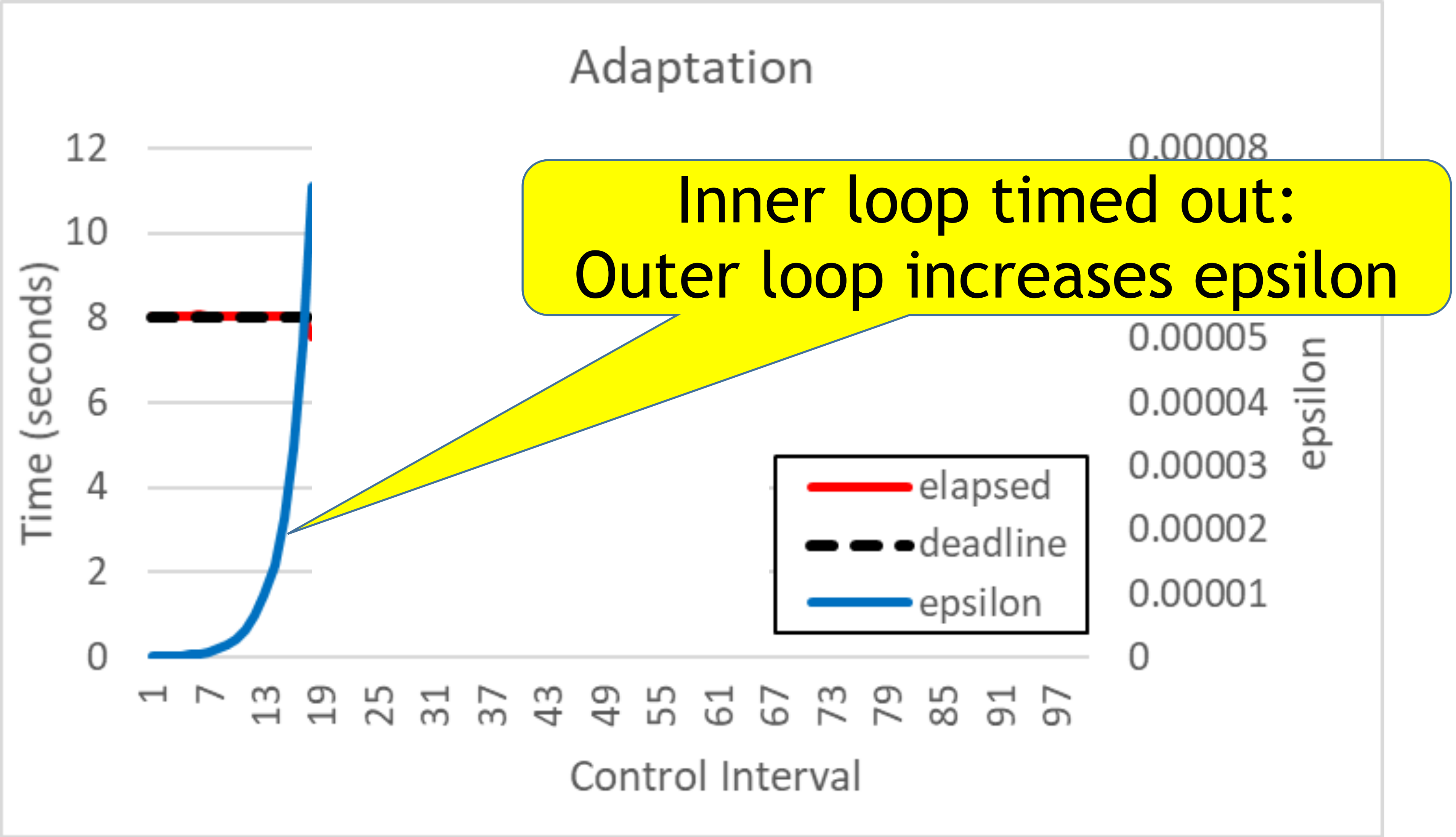
Outer loop : adapting epsilon



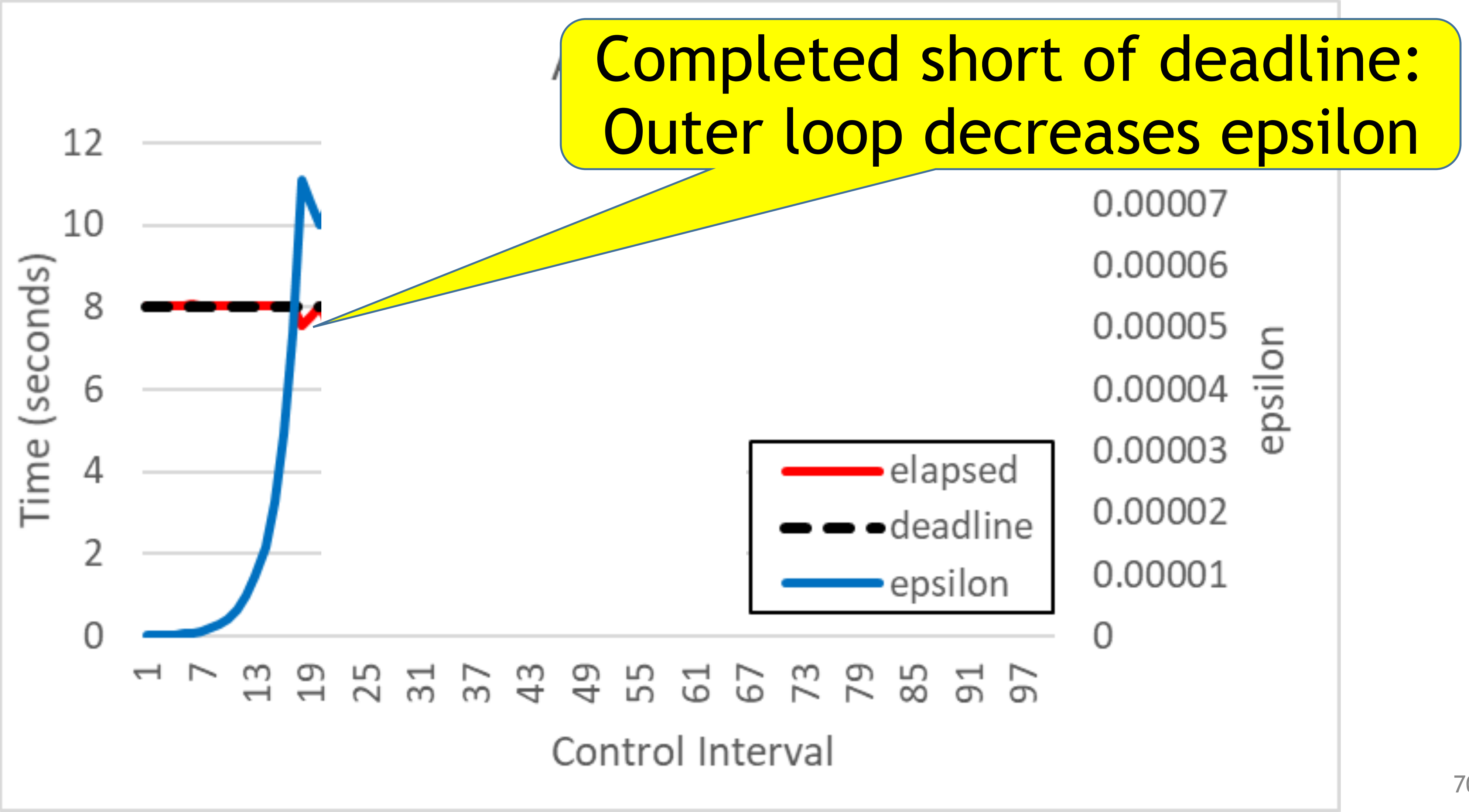
Outer loop : adapting epsilon



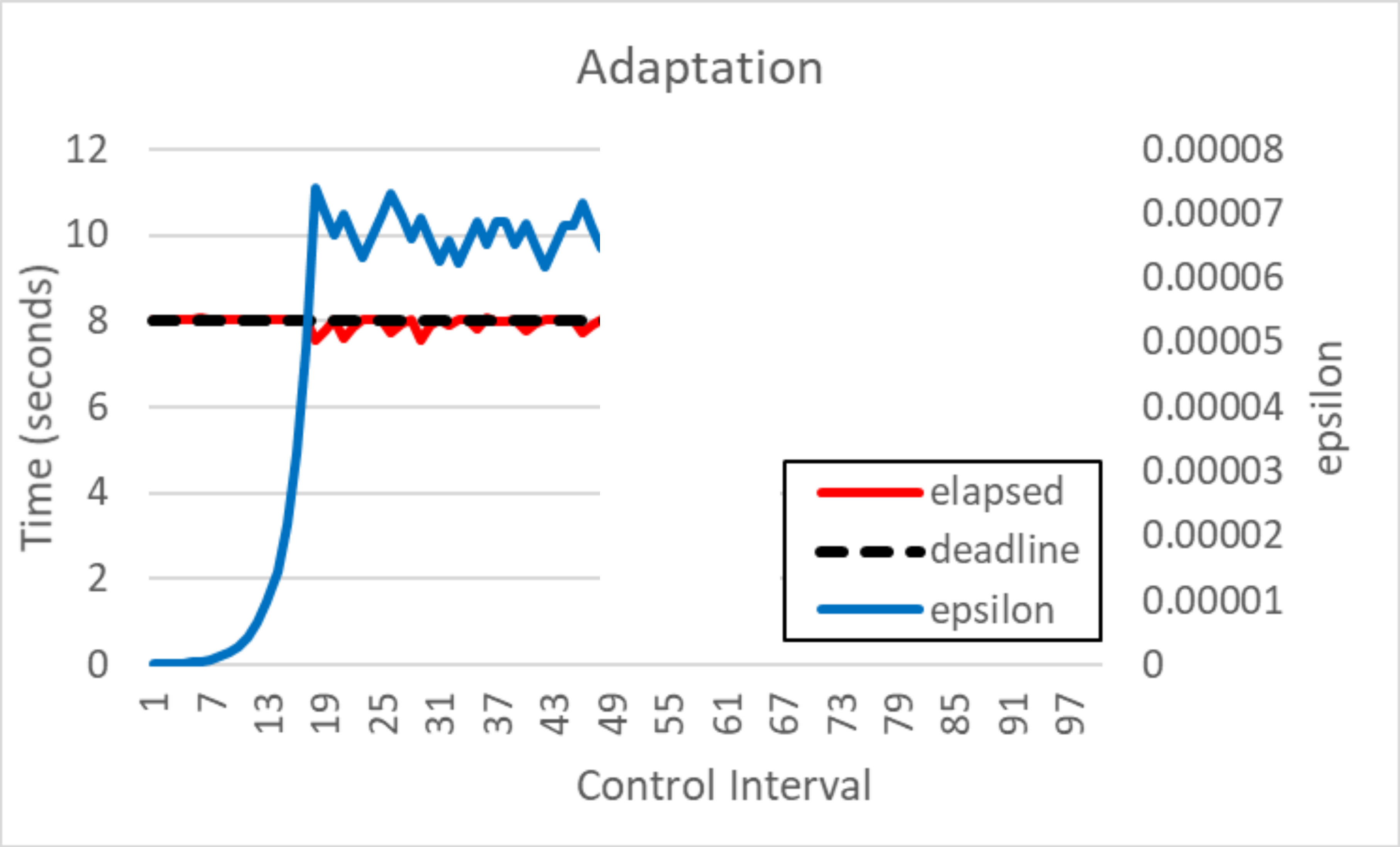
Outer loop : adapting epsilon



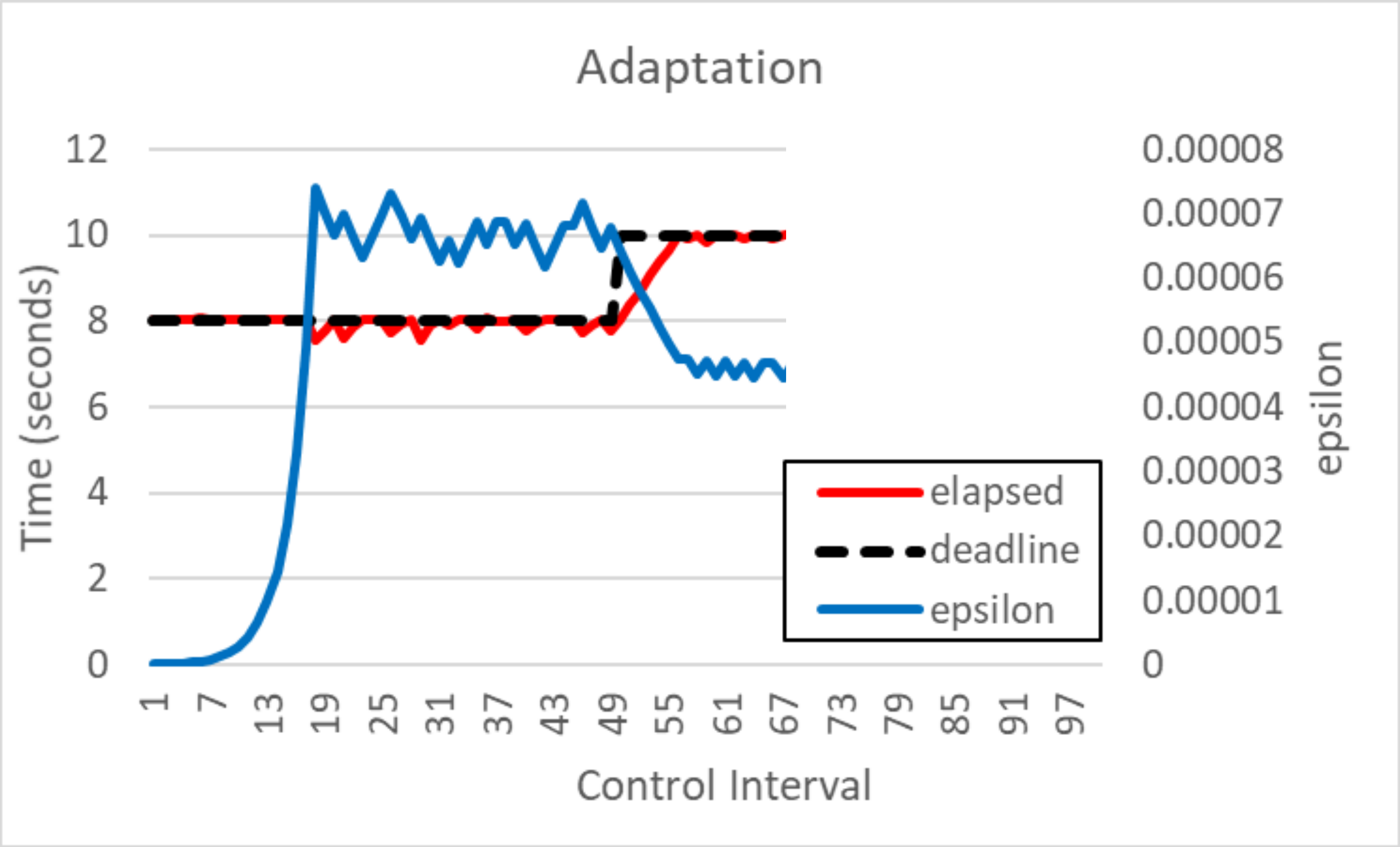
Outer loop : adapting epsilon



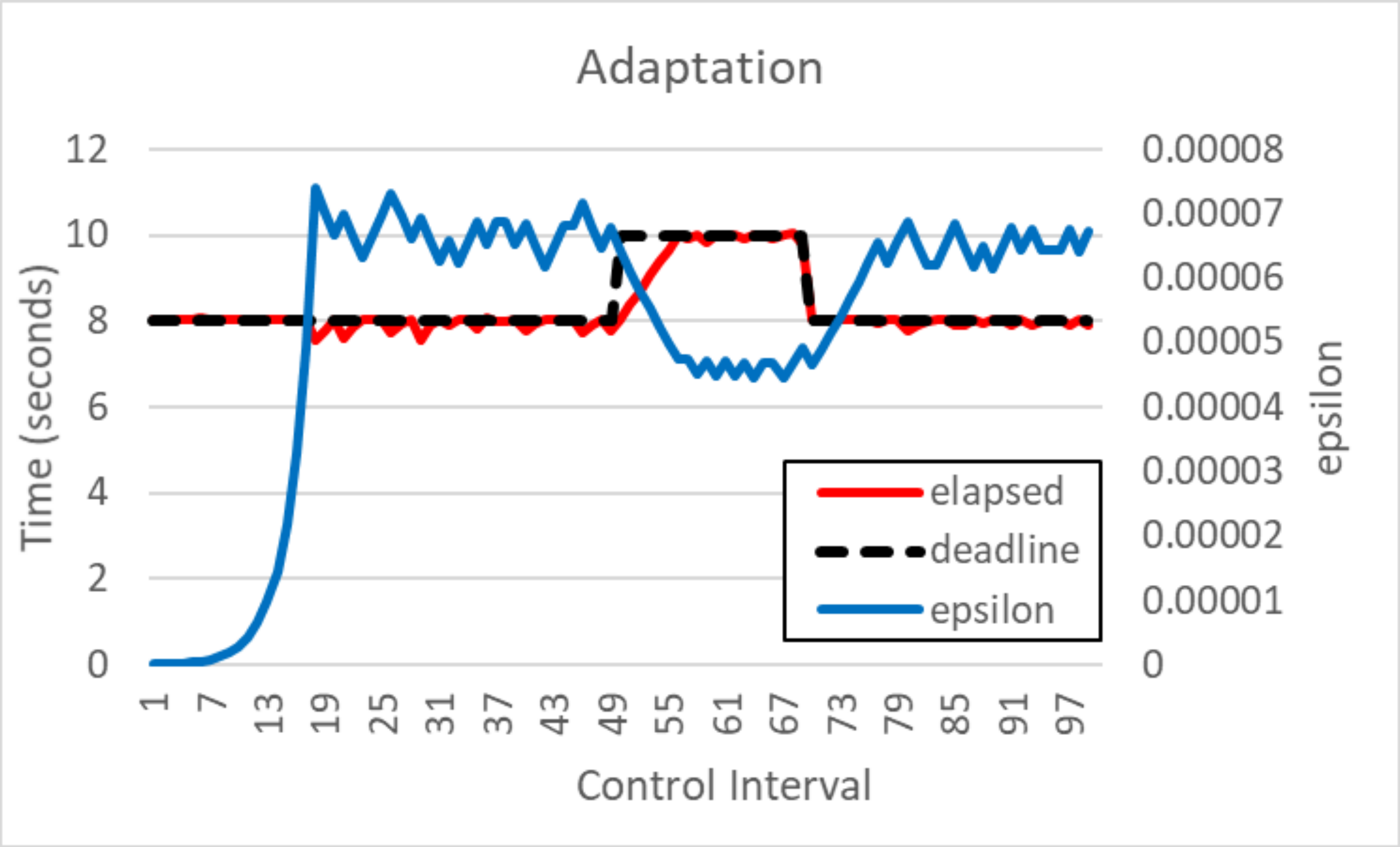
Outer loop : adapting epsilon



Outer loop : adapting epsilon



Outer loop : adapting epsilon



Summary

	EDRF	DC-DRF	util. drop/ baseline
1Mx100K:	15 mins	8 secs	0.0065%

Summary

	EDRF	DC-DRF	util. drop/ baseline
1Mx100K:	15 mins	8 secs	0.0065%
1Mx1M:	129 mins	8 secs	0.06%

Conclusion

DC-DRF enables multi-resource allocation to be calculated at Public Cloud scale in bounded time.

Thankyou

Backup video from demo at SIGCOMM'15

- 4 x tenants with 3 VMs each on 10 compute servers
- Accessing 2X RAMD storage servers over RDMA
- Demand estimation and vector rate limiters in Hyper-V drivers
- Central controller using EDRF algorithm in two passes
  - per-tenant aggregate reservation and intra-tenant work conservation
  - Inter-tenant work conservation

Specification

Outer loop : find to meet deadline

while true do

// ingest latest observed demands…

Inner loop : approximation of EDRF

while do

…fication

Iterate until done or timeout

Inner loop : approximation of EDRF

while do

while true do

// ingest latest observed demands…

Specification

Smallest for this round

Ou

nner loop : approximation of EDRF

vhile do

while true do

// ingest latest observed demands…

Specification

Outer loop : find to meet deadline

while true do

// ingest latest observed demands…

Inner loop : approximation of EDRF

while do

trades utilization for speed

Specification

Outer loop : find to meet deadline

while true do

// ingest latest observed demands...

Inner loop : approximation of EDRF

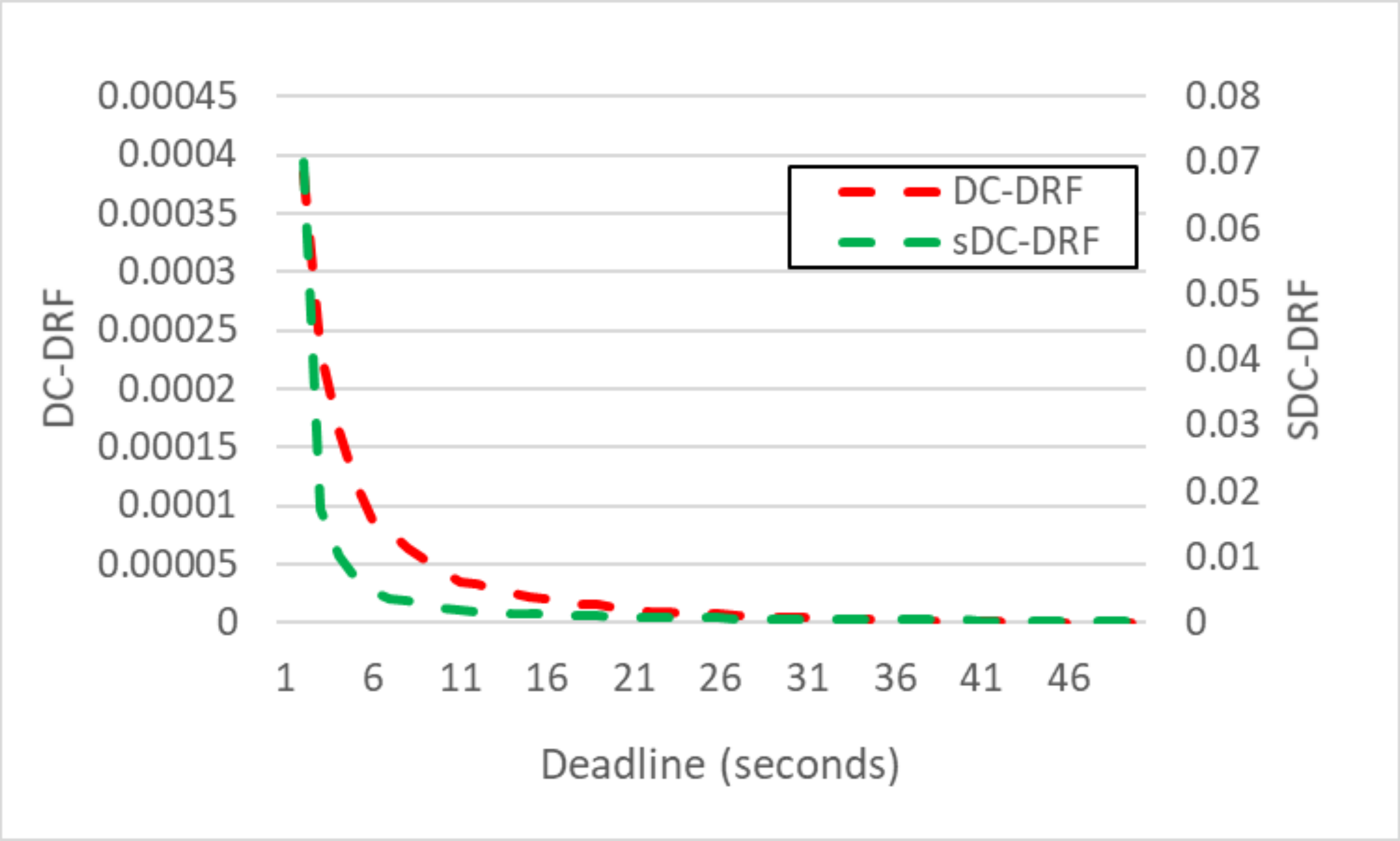
while do

Adjust to meet deadline

DRF fairness properties

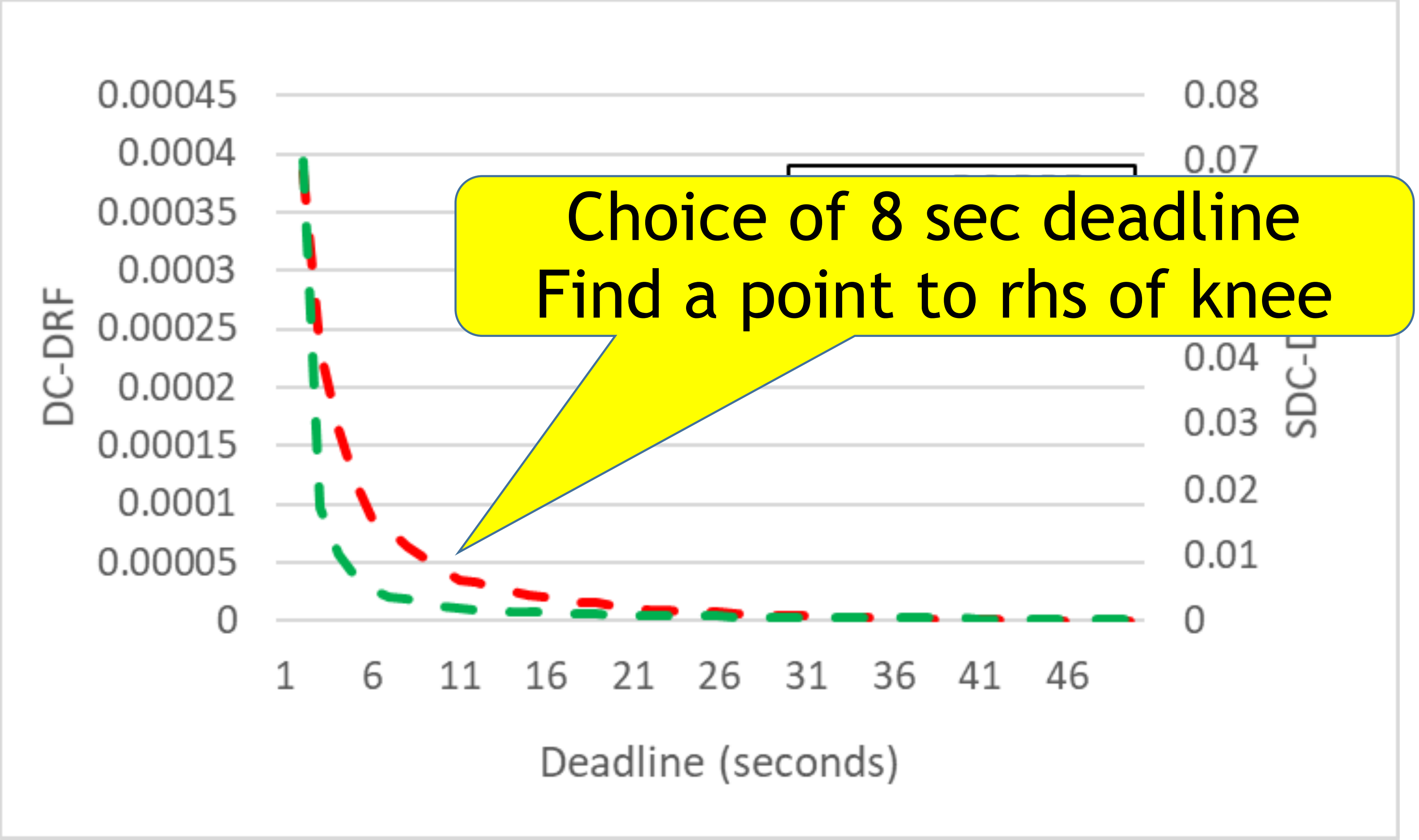
- Formal fairness properties:
  - Sharing incentive : no tenant would prefer a simple resource partitioning
  - Strategy-proof : no benefit from falsified demands
  - Envy-free : no tenant would prefer another tenant’s allocation
  - Pareto-fairness : increasing one tenant decreases another

epsilon



0.00045
0.0004
0.00035
0.0003
0.00025
0.0002
0.00015
0.0001
0.00005
0
DC-DRF
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
SDC-DRF
1
6
11
16
21
26
31
36
41
46
Deadline (seconds)
DC-DRF
sDC-DRF

epsilon


0.00045
0.0004
0.00035
0.0003
0.00025
0.0002
0.00015
0.0001
0.00005
0
DC-DRF
0.08
0.07
0.04
0.03
0.02
0.01
0
1
6
11
16
21
26
31
36
41
46
Deadline (seconds)
Choice of 8 sec deadline
Find a point to rhs of knee

What worked in our prototypes

- Distribute enforcement mechanisms into edge hypervisors
  - Classification, demand estimation, rate limiters
- Central SDN-like controller calculating shares
  - Simpler algorithm: easier to build confidence
  - Complete information beats partial views (think: B4 and SWAN)
- For detail see
  - IoFlow: single-resource Max-Min [Thereska et al., SOSP'13]
  - Pulsar: multi-resource EDRF [Angel et al., OSDI'14] :
  - Filo : distributed EDRF [Marandi at al., USENIX ATC 16]