

# Deploying remote GPU virtualization with rCUDA

**Federico Silla**

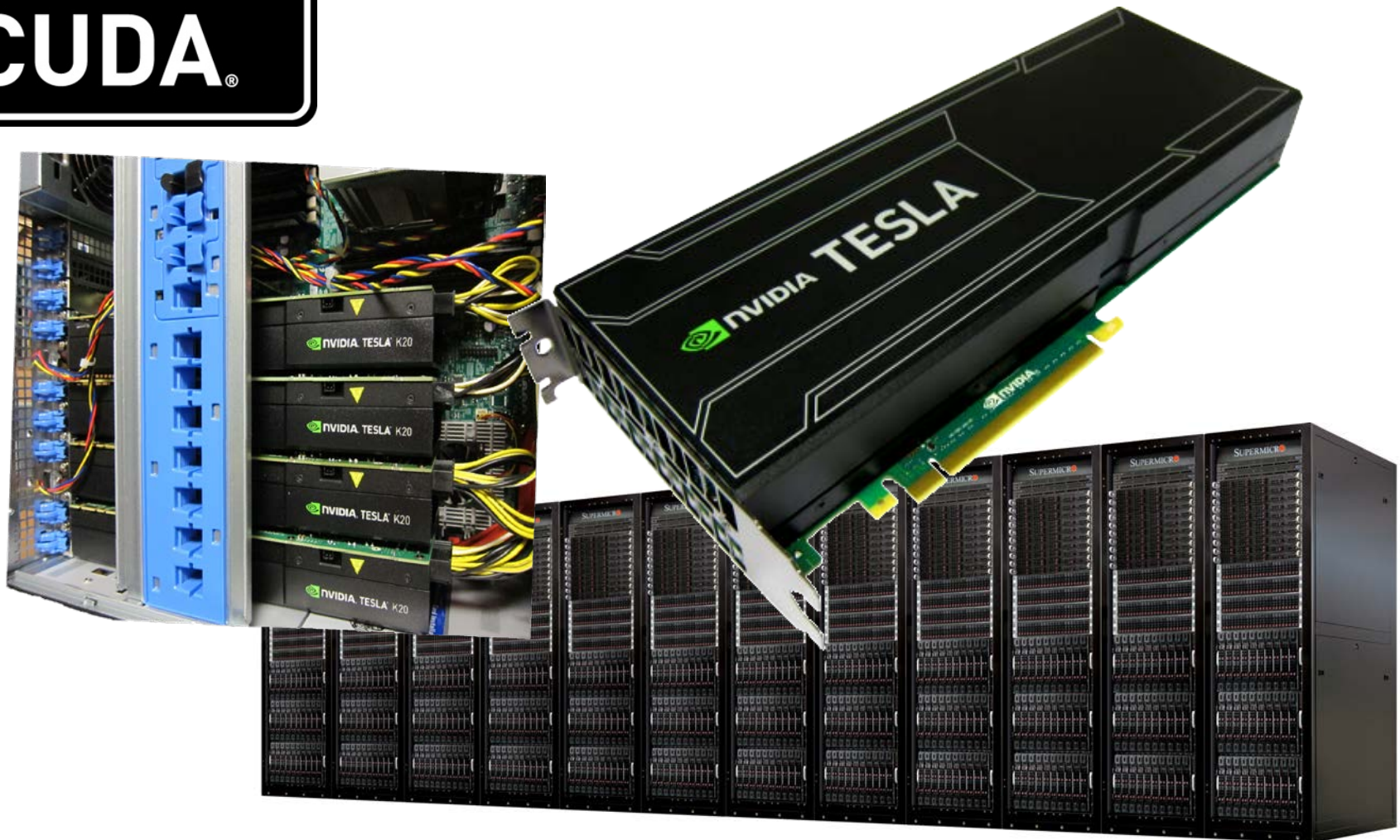
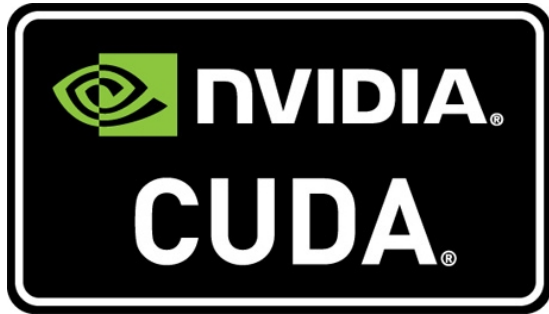
Technical University of Valencia  
Spain

# 1st

What is "*remote GPU virtualization*"?

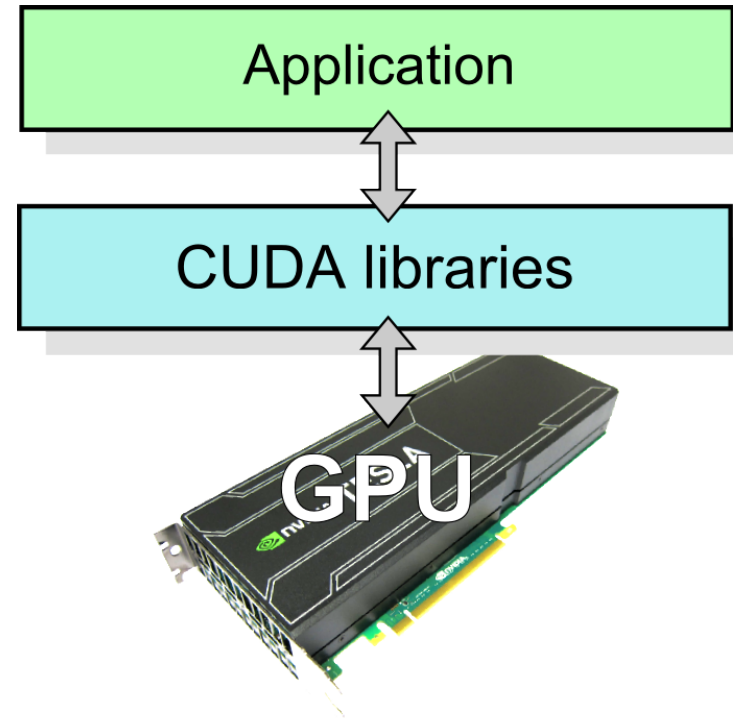
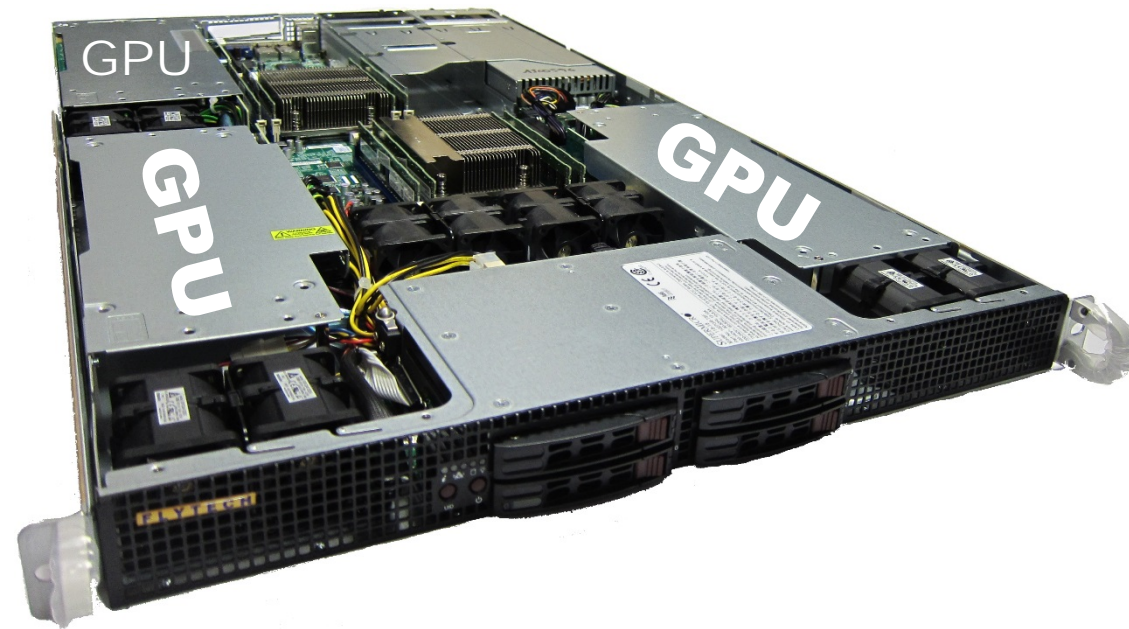


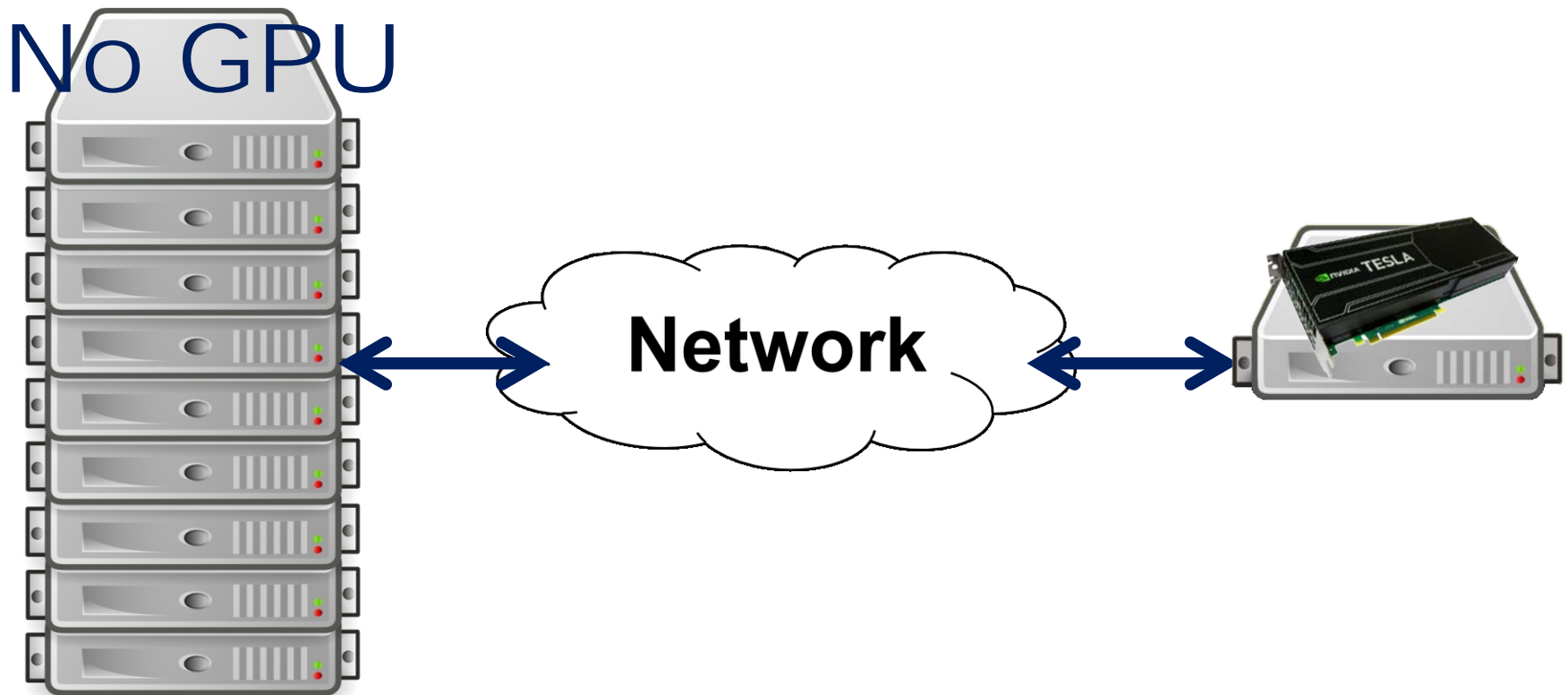
*It deals with GPUs, obviously!*



# Basics of GPU computing

## Basic behavior of CUDA

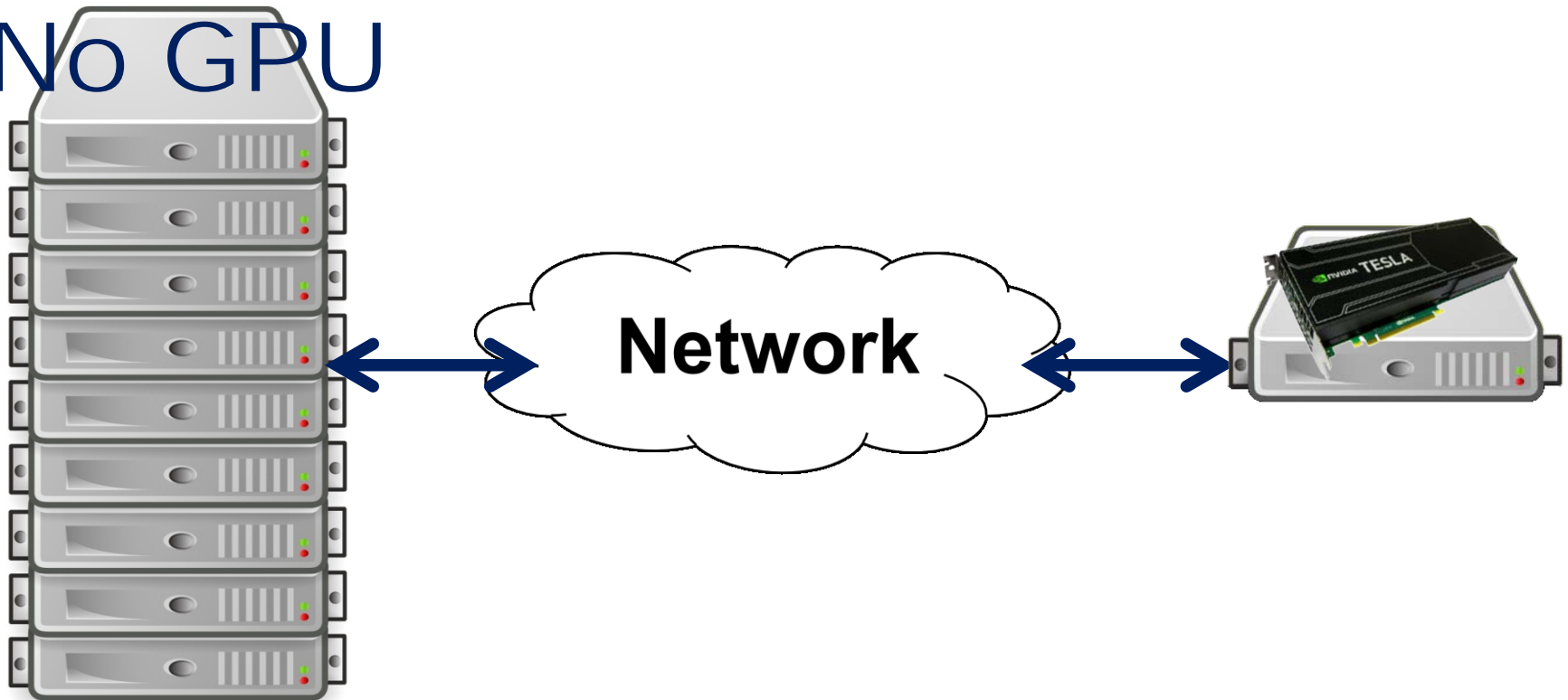




# Remote GPU virtualization

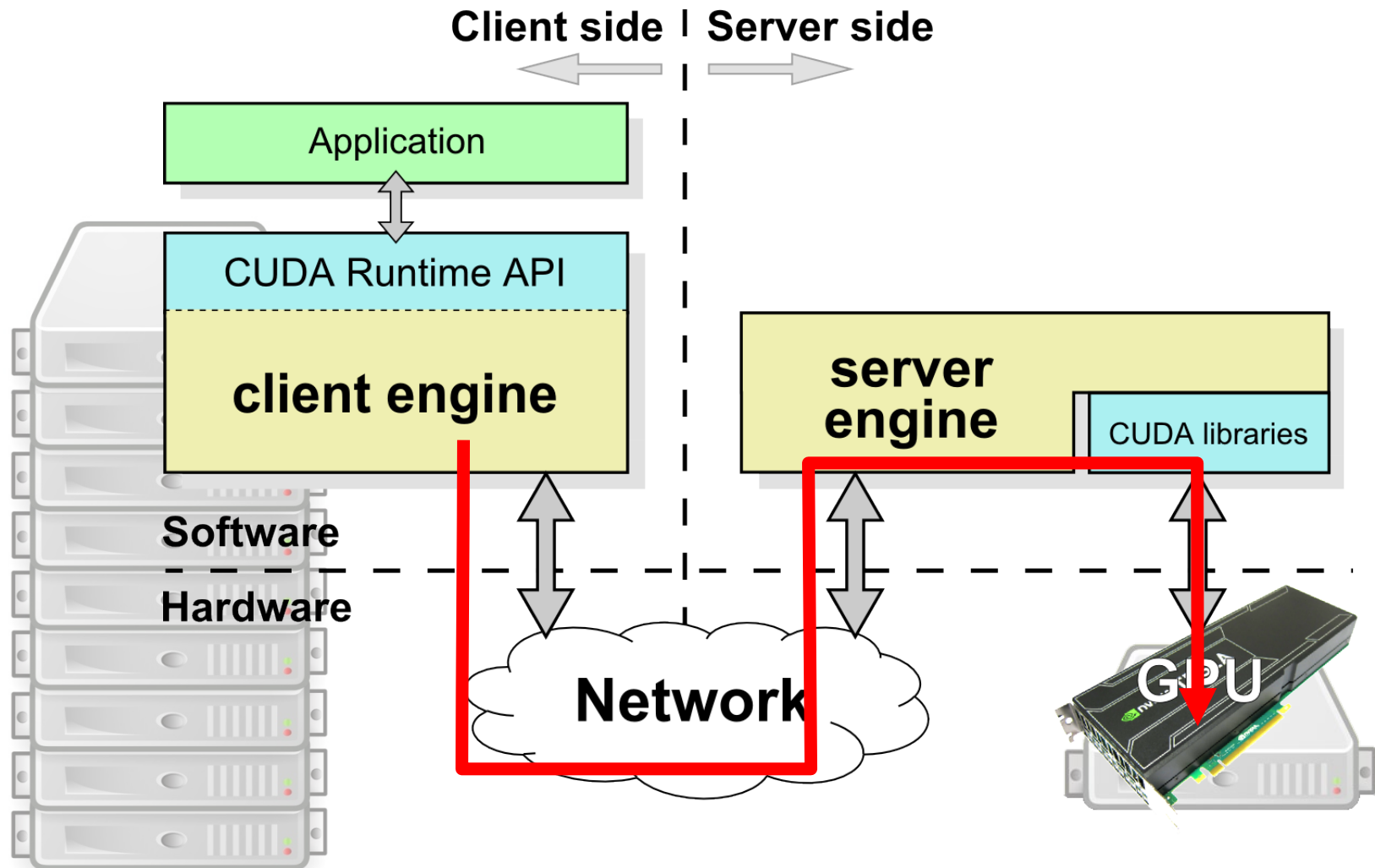
A **software** technology that enables **a more flexible use of GPUs** in computing facilities

No GPU

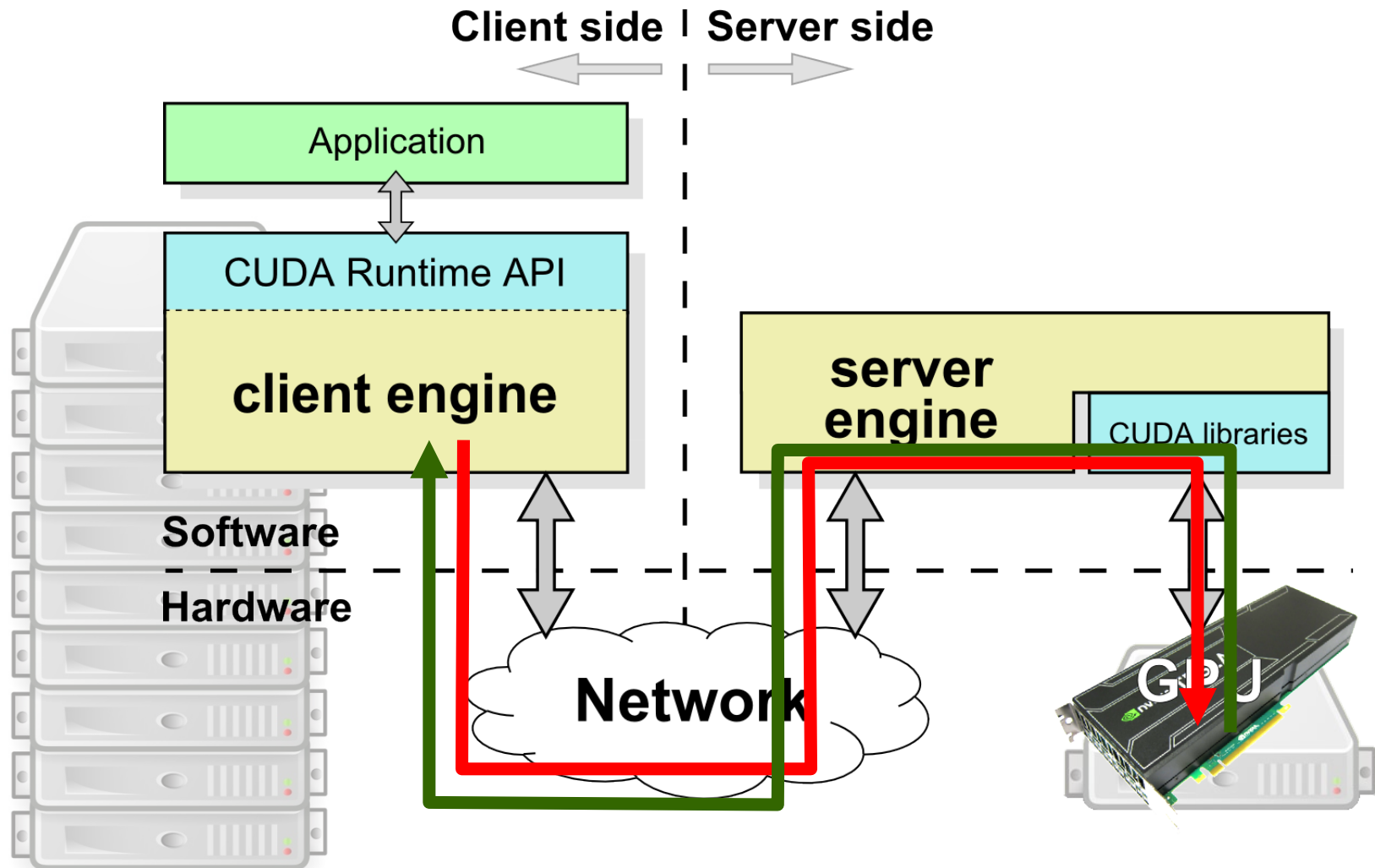




# Basics of remote GPU virtualization



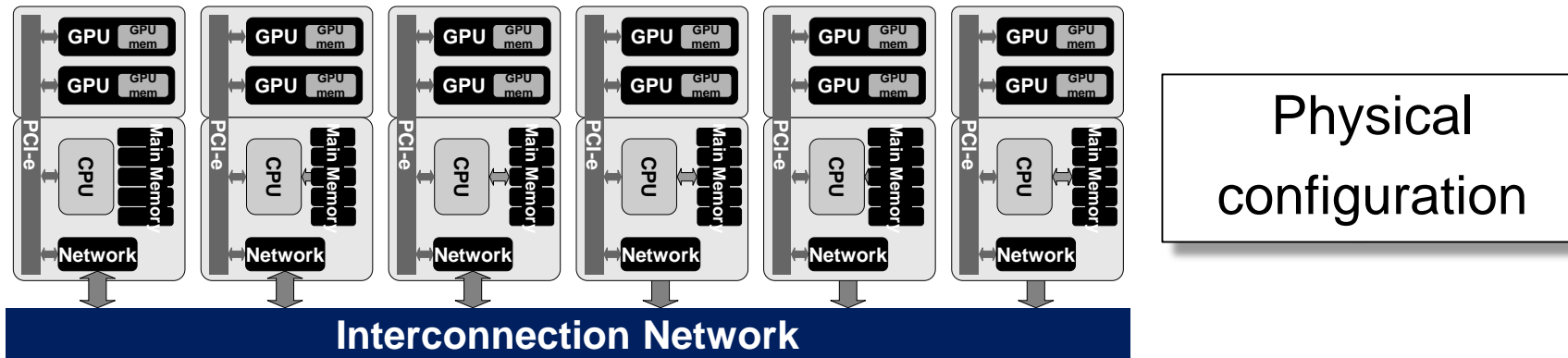
# Basics of remote GPU virtualization



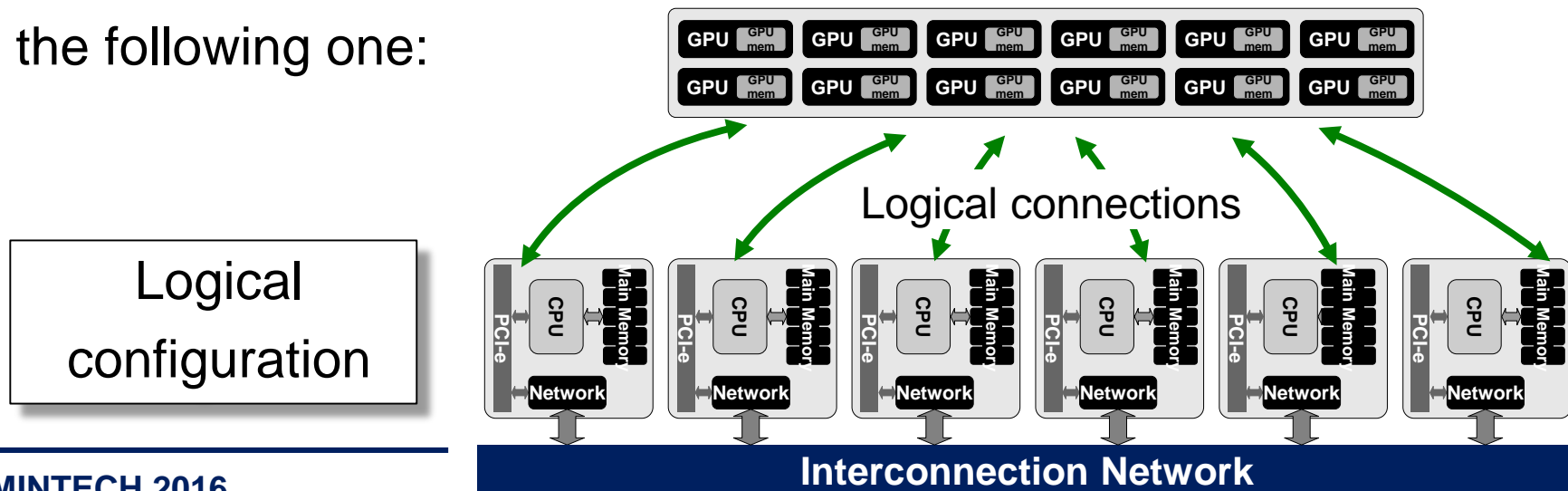


# Remote GPU virtualization envision

- Remote GPU virtualization allows a new vision of a GPU deployment, moving from the usual cluster configuration:



to the following one:



# 2nd

Why is "*remote GPU virtualization*" needed?



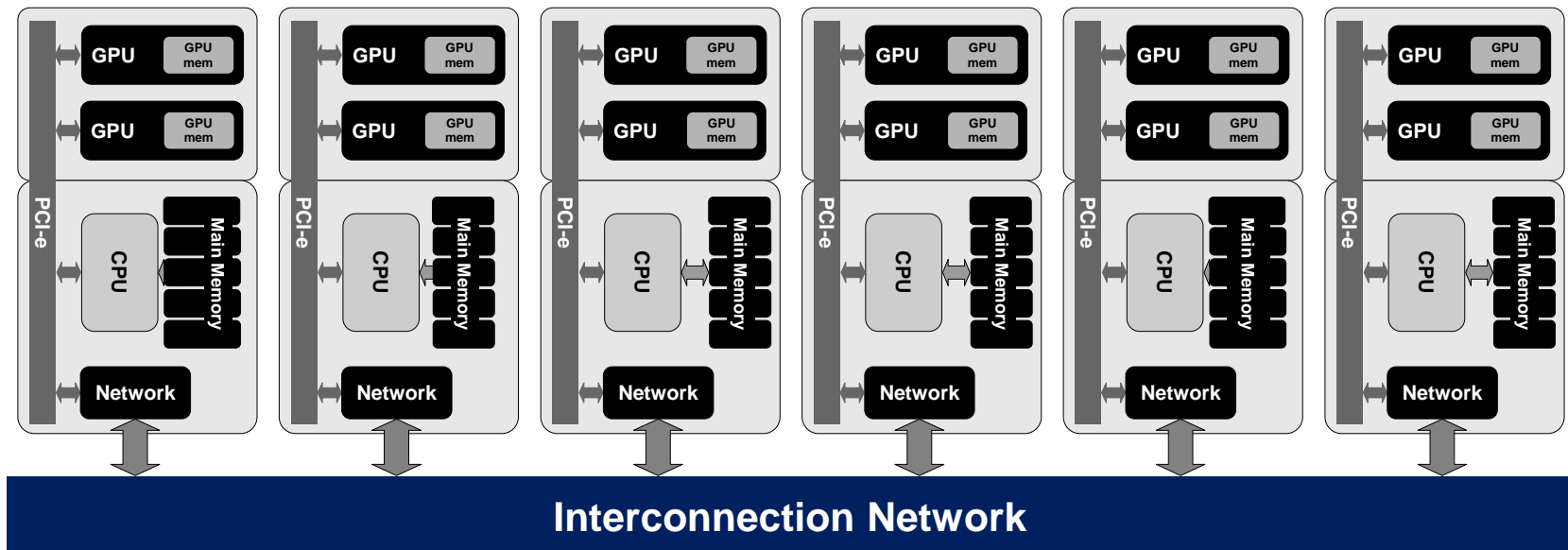


Which is the problem  
with GPU-enabled  
clusters?



# Characteristics of GPU-based clusters

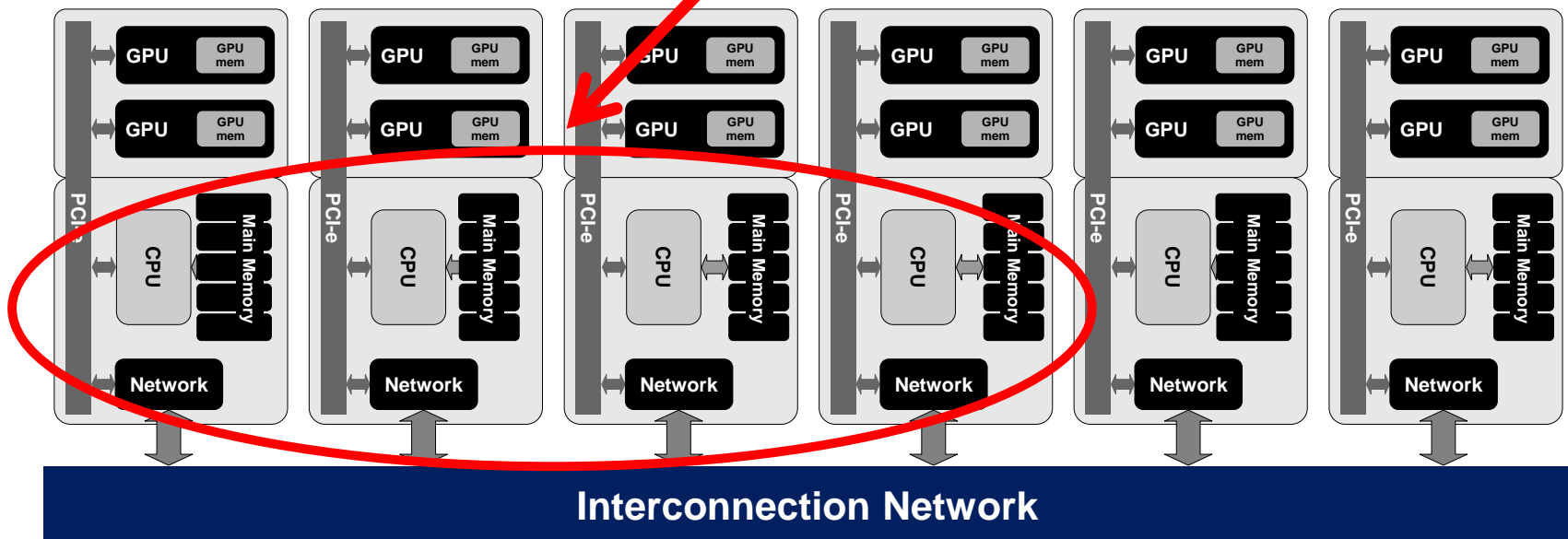
- A GPU-enabled cluster is a set of independent self-contained nodes that leverage the **shared-nothing approach**:
  - Nothing is directly shared among nodes (MPI required for aggregating computing resources across the cluster, **included GPUs**)
  - **GPUs can only be used within the node they are attached to**



# First concern with accelerated clusters

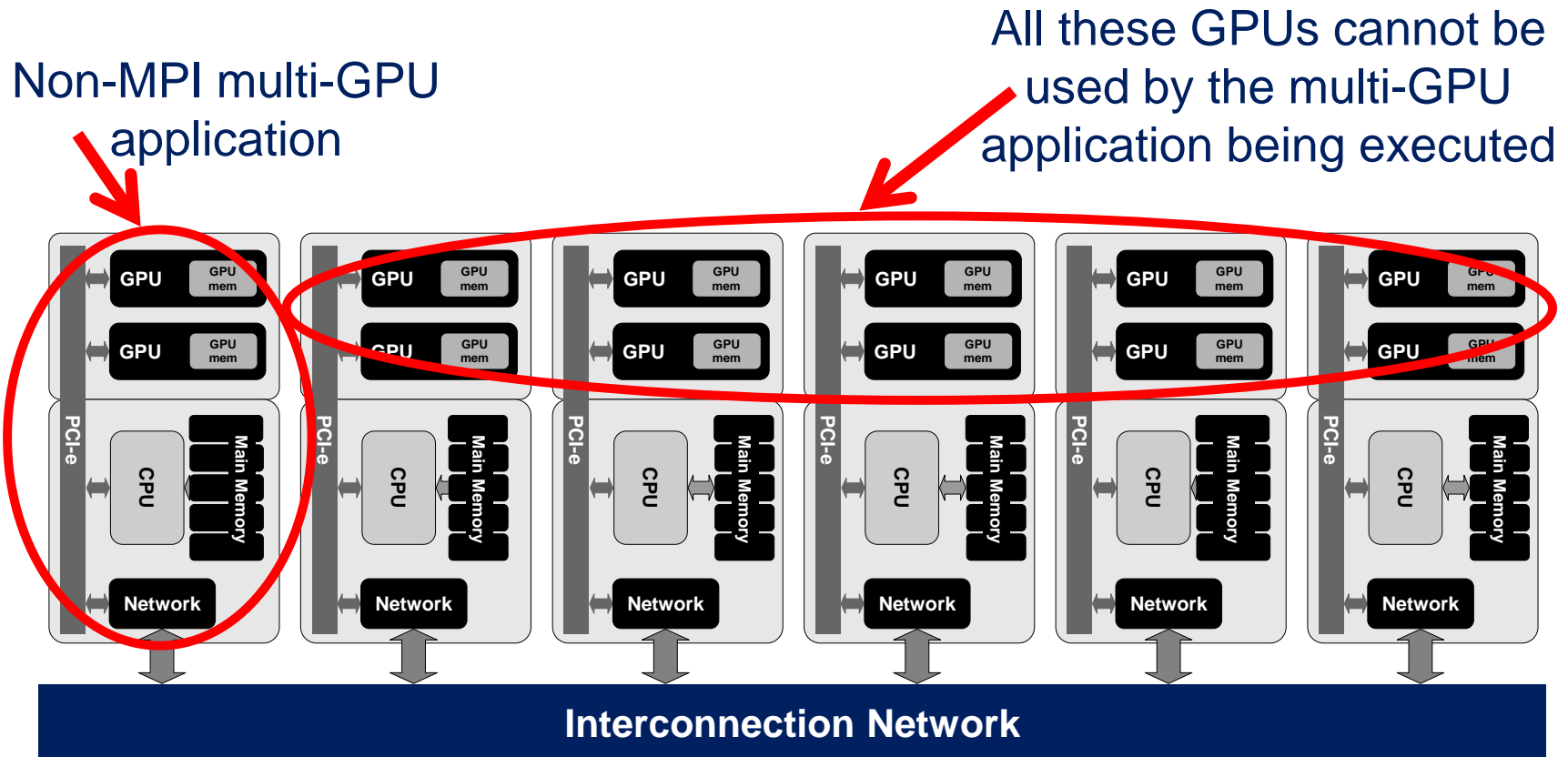
- Applications can only use the GPUs located within their node:
  - Non-accelerated applications **keep GPUs idle** in the nodes where they use all the cores

A CPU-only application spreading over these four nodes would make their GPUs unavailable for accelerated applications



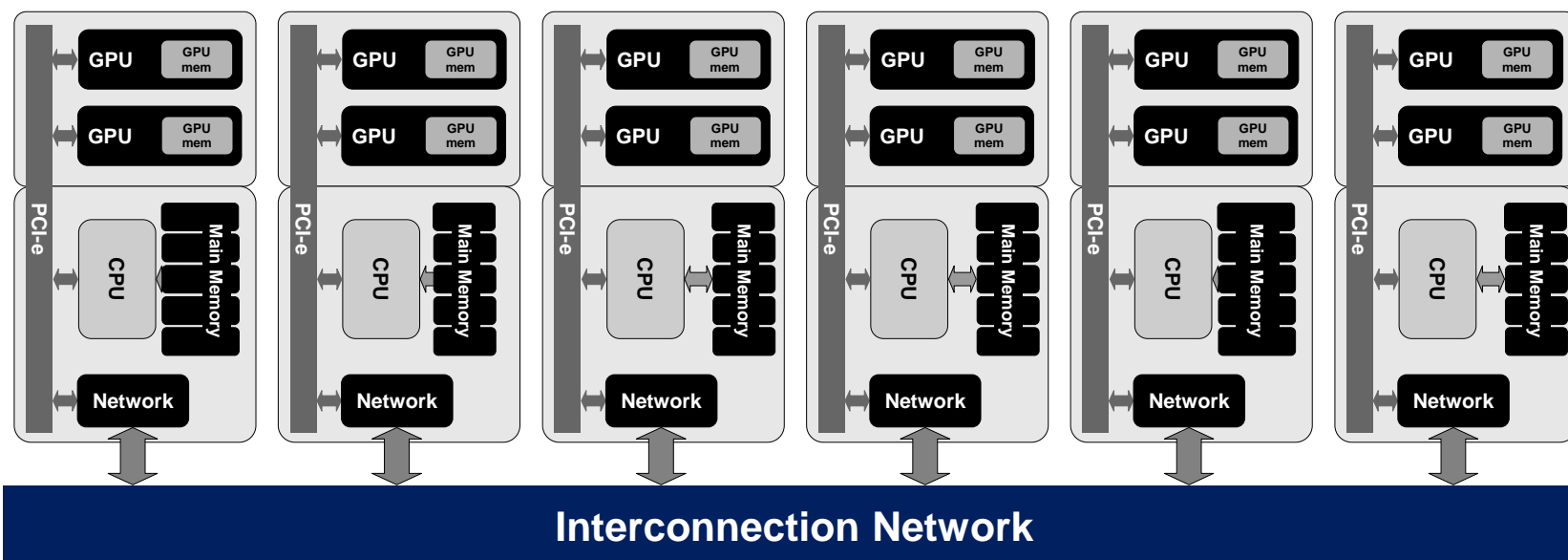
# Second concern with accelerated clusters

- Applications can only use the GPUs located within their node:
  - non-MPI multi-GPU applications running on a subset of nodes cannot make use of the tremendous GPU resources available at other cluster nodes (even if they are idle)

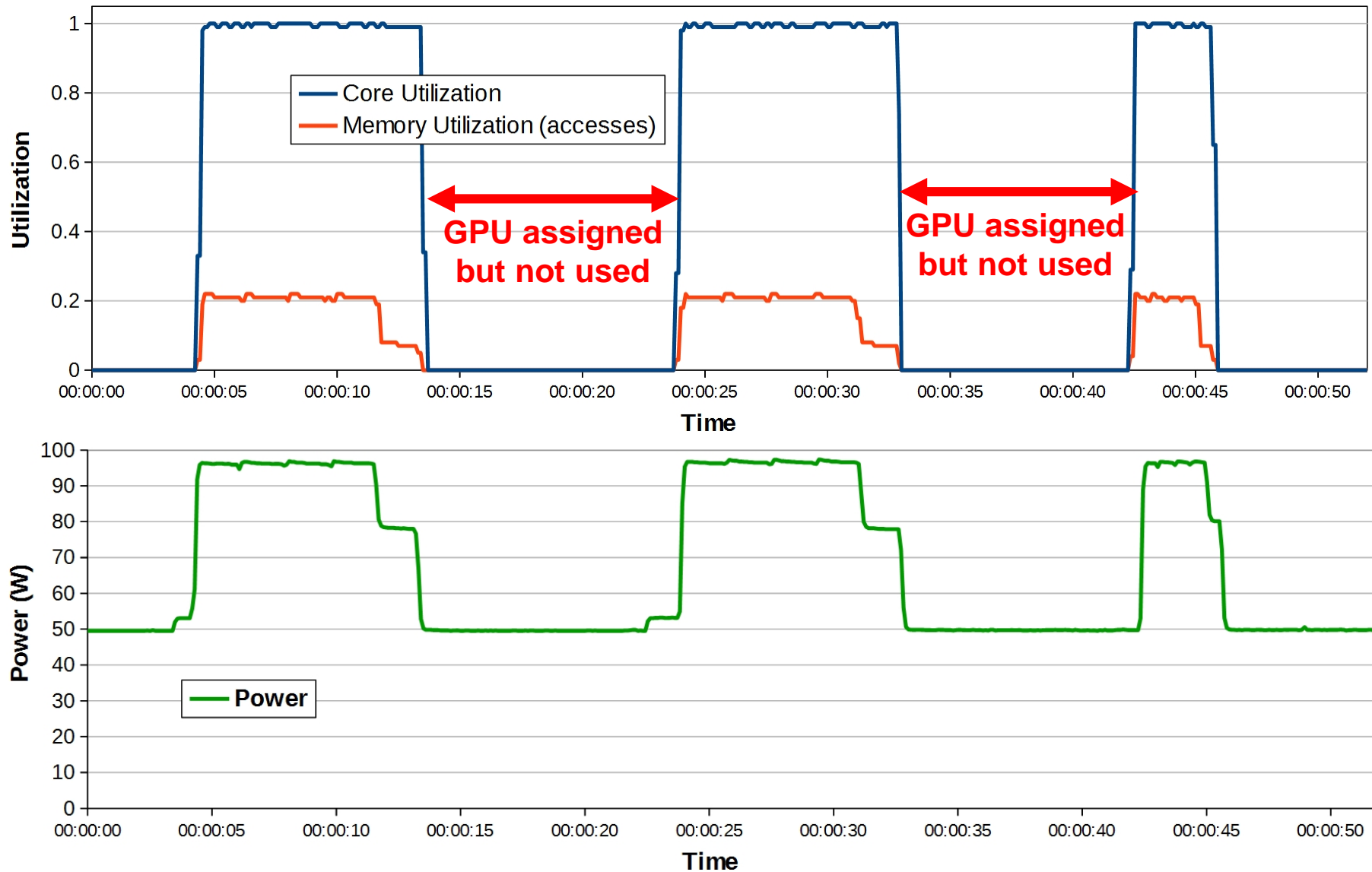




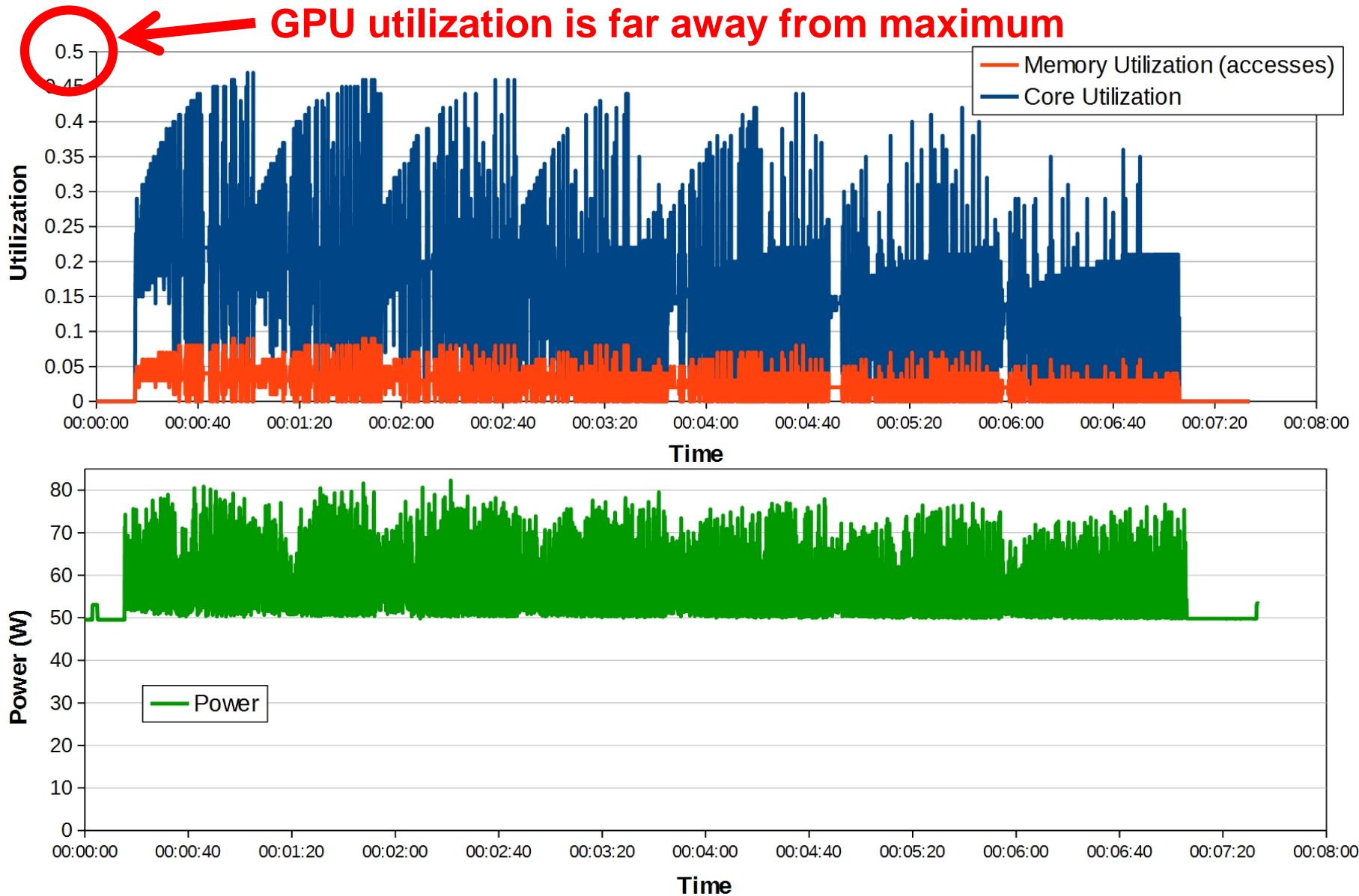
- Do applications **completely squeeze** the GPUs available in the cluster?
  - When a GPU is assigned to an application, computational resources inside the GPU may not be fully used
    - Application presenting low level of parallelism
    - CPU code being executed (**GPU assigned  $\neq$  GPU working**)
    - GPU-core stall due to lack of data
    - etc ...



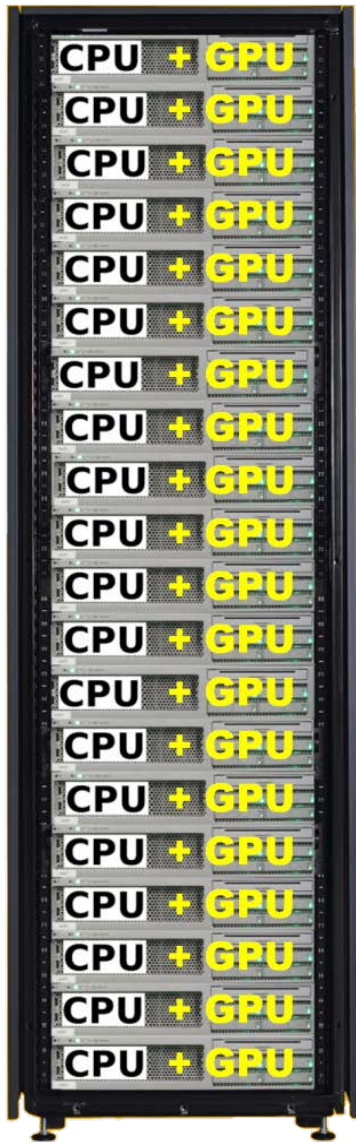
# GPU usage of GPU-Blast



# GPU usage of CUDA-MEME



# Why performance in GPU clusters is lost?



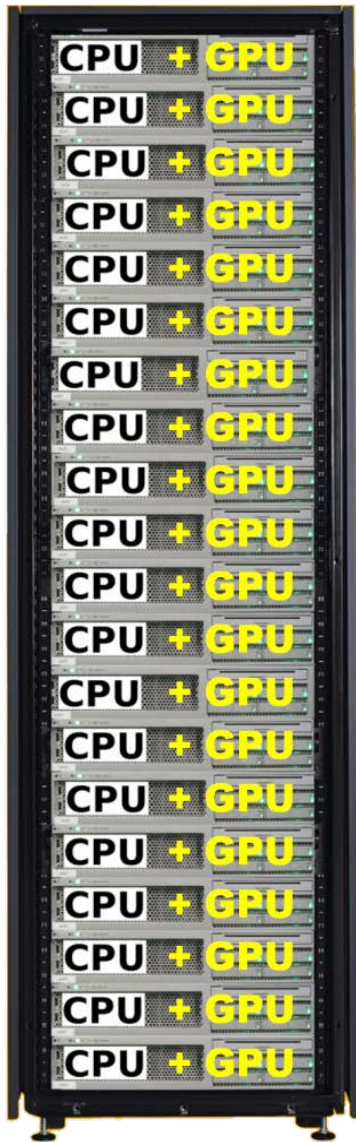
## In summary ...

- There are scenarios where GPUs are available but cannot be used
- Accelerated applications do not make use of GPUs 100% of the time

## In conclusion ...

- GPU cycles are lost, thus reducing cluster performance

# *We need something more in the cluster*

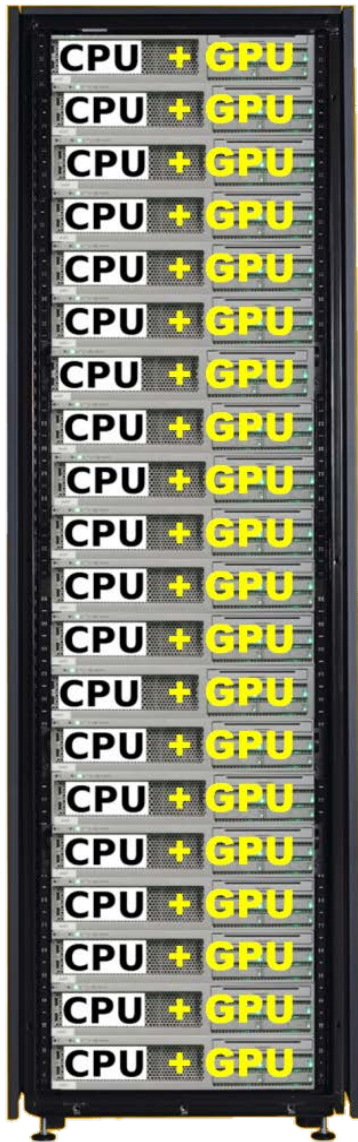


The current model for using GPUs is too rigid

**What is missing is ...**

**... some flexibility for using the GPUs in the cluster**

# *We need something more in the cluster*



The current model for using GPUs is too rigid

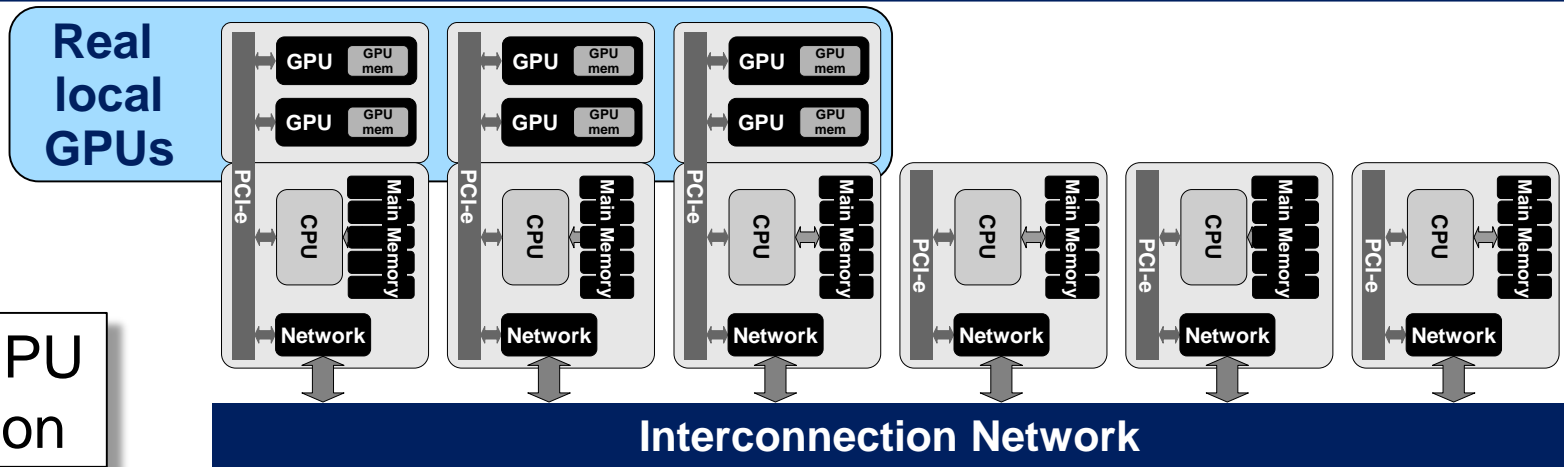
**What is missing is ...**

**... some flexibility for using the GPUs in the cluster**

A way of seamlessly sharing GPUs across nodes in the cluster  
(**remote GPU virtualization**)

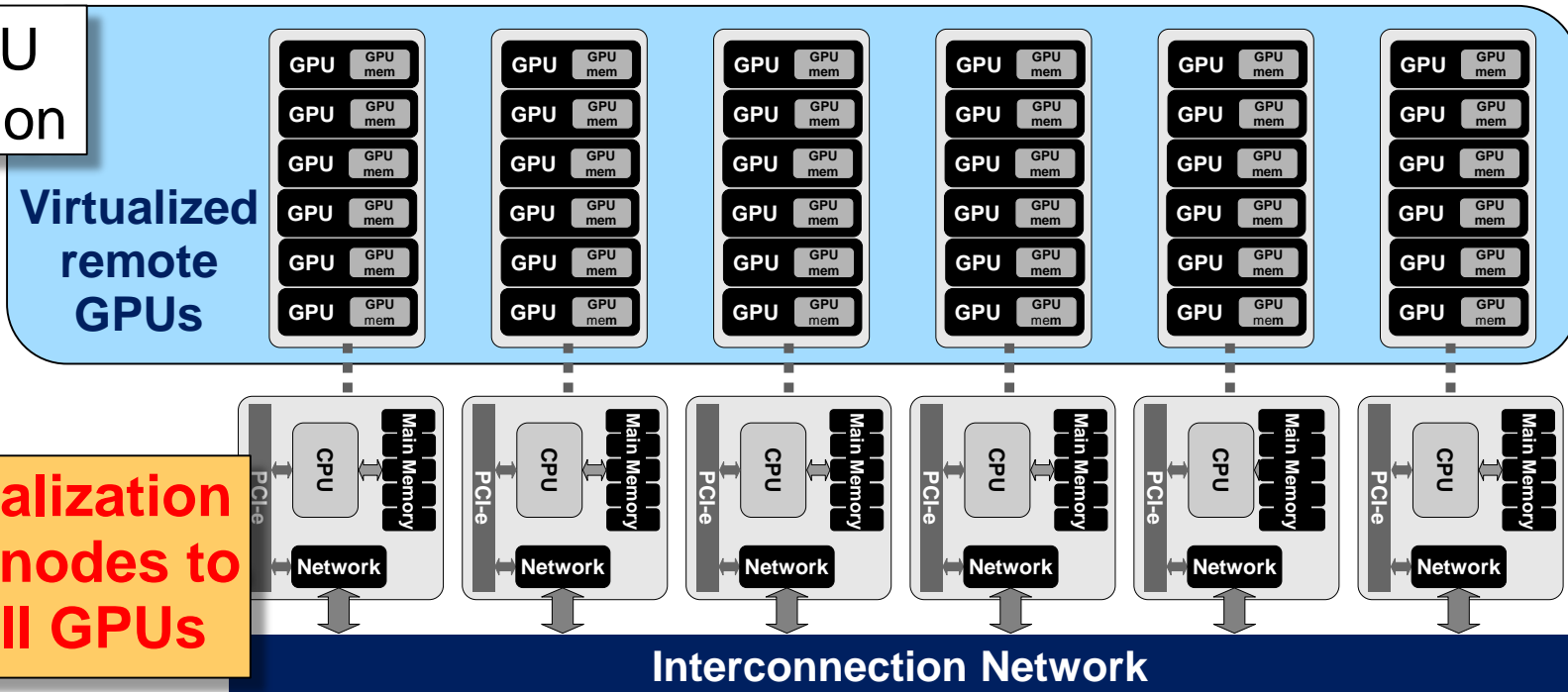


## *Remote GPU virtualization envision*



## Without GPU virtualization


**With GPU**  
virtualization



**GPU virtualization  
allows all nodes to  
access all GPUs**

# Remote GPU virtualization frameworks

- Several efforts have been made to **implement** remote GPU virtualization during the last years:

<ul style="list-style-type: none"> <li>• <b><u>rCUDA</u></b> (CUDA 7.0)</li> <li>• GVirtuS (CUDA 3.2)</li> <li>• DS-CUDA (CUDA 4.1)</li> </ul>	 <div>Publicly available</div>
<ul style="list-style-type: none"> <li>• vCUDA (CUDA 1.1)</li> <li>• GViM (CUDA 1.1)</li> <li>• GridCUDA (CUDA 2.3)</li> <li>• V-GPU (CUDA 4.0)</li> </ul>	<div>NOT publicly available</div>

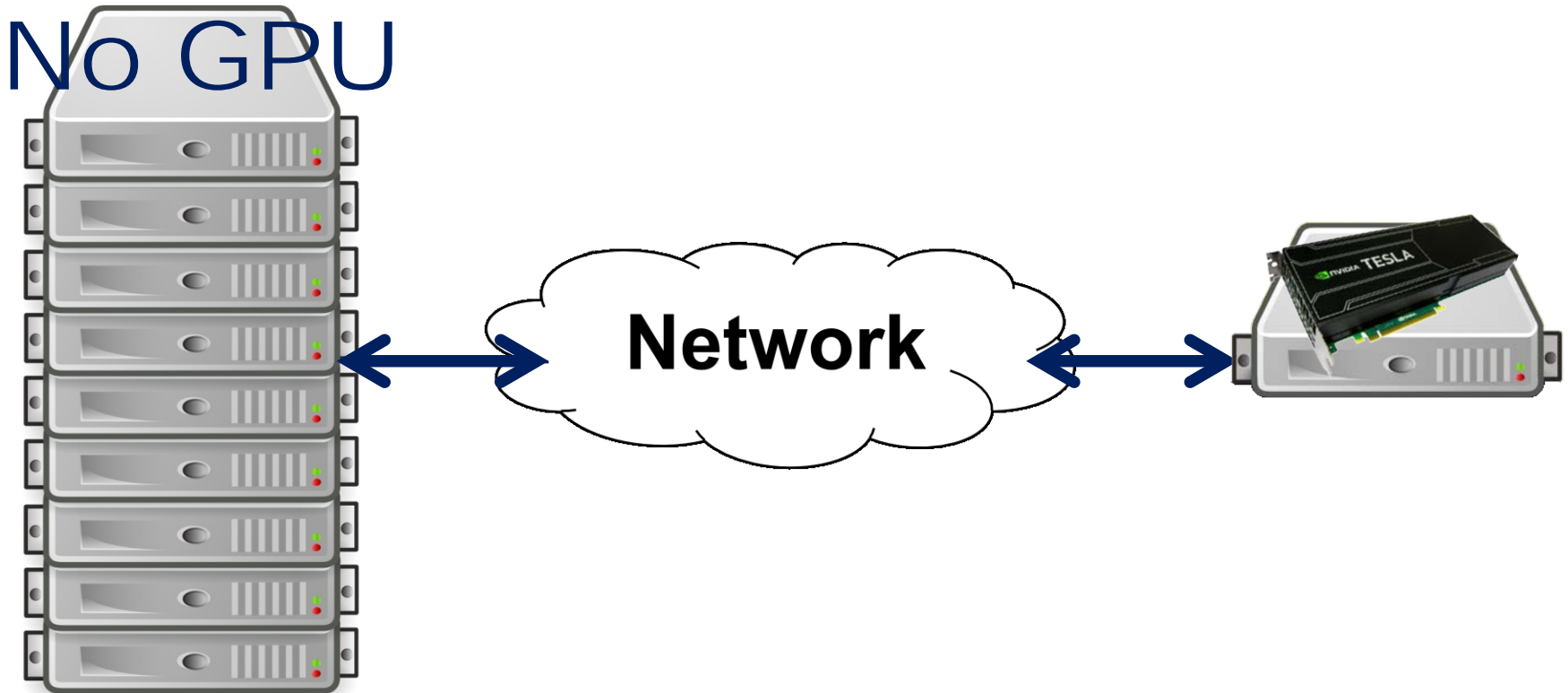
# 3rd

Cons of "*remote GPU virtualization*"?

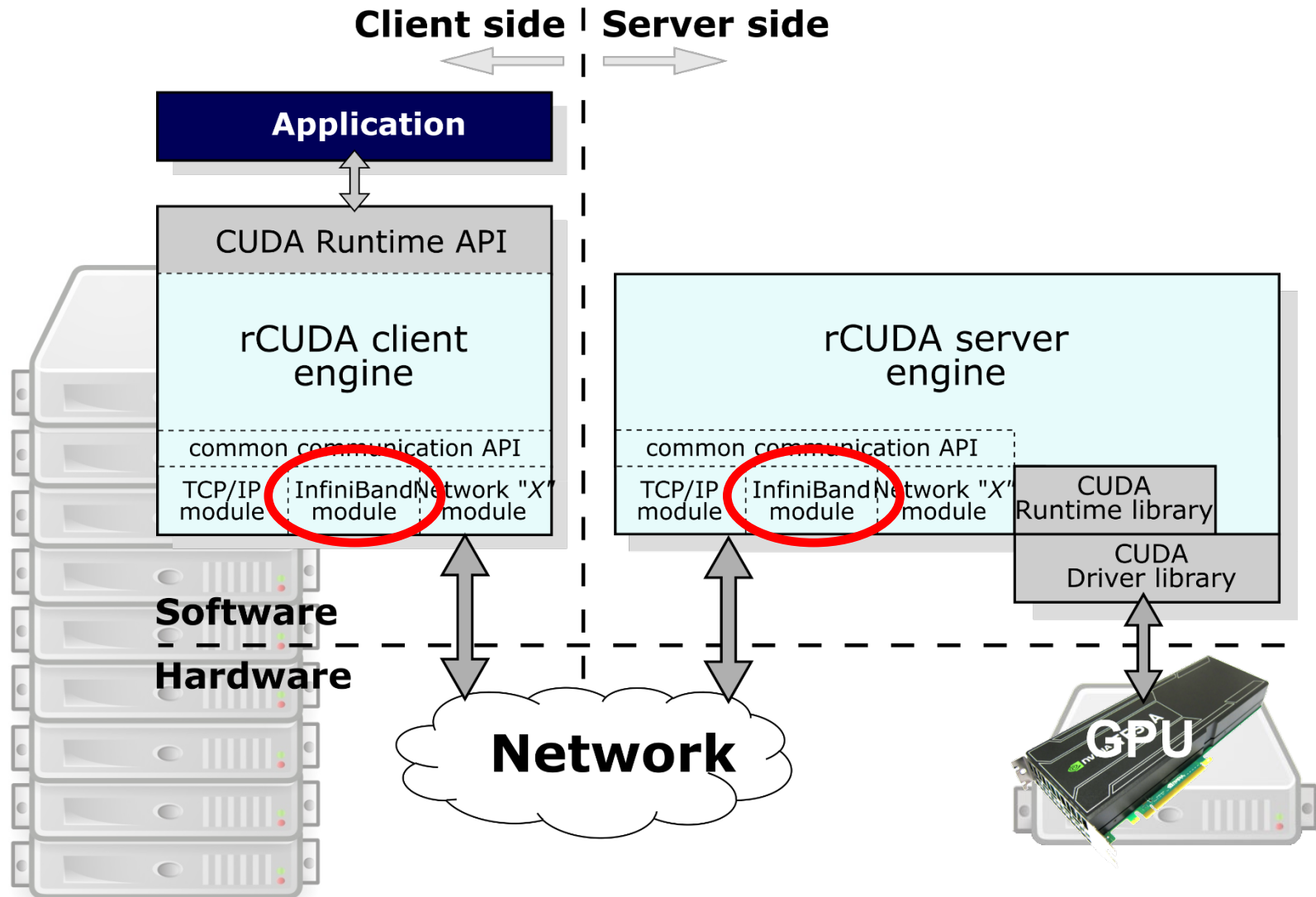


# Problem with remote GPU virtualization

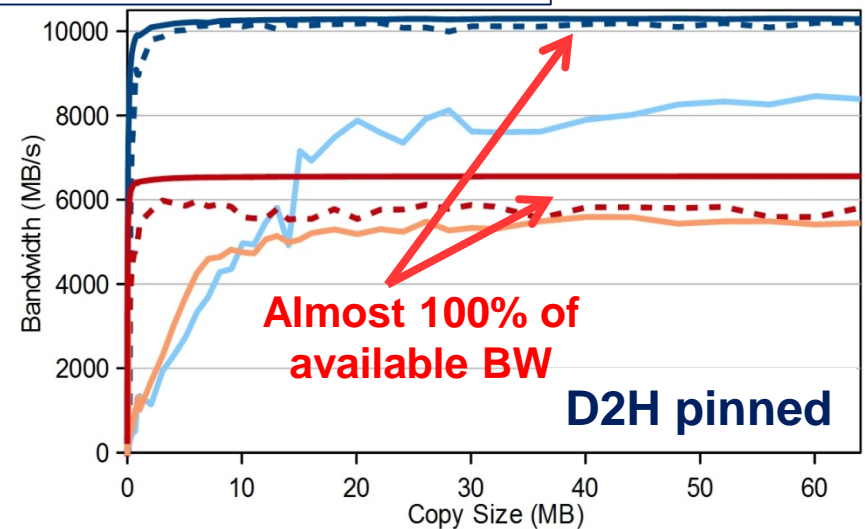
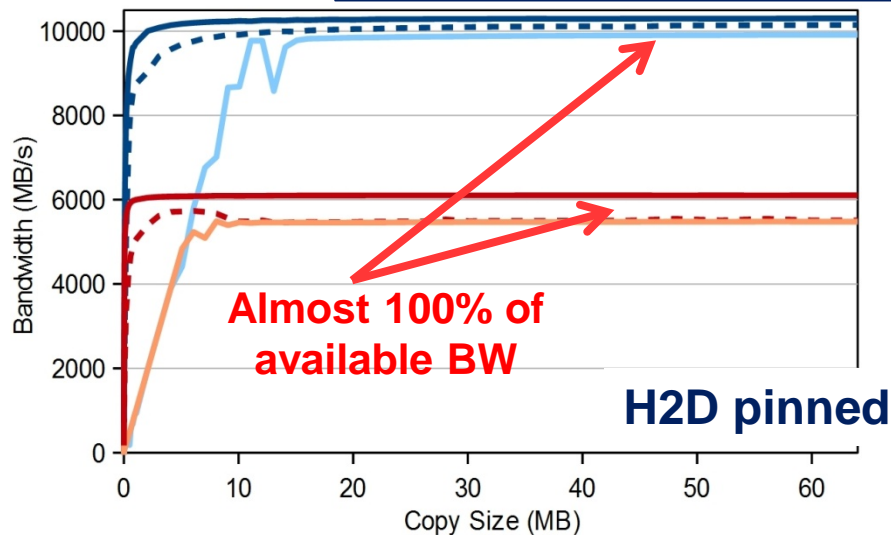
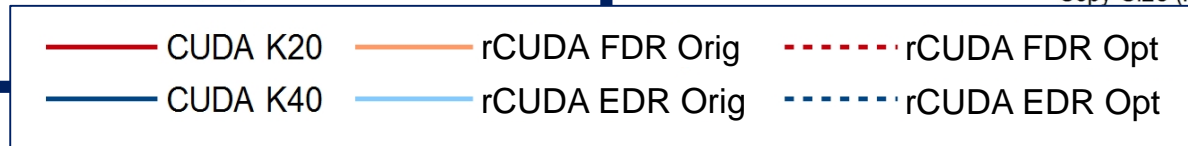
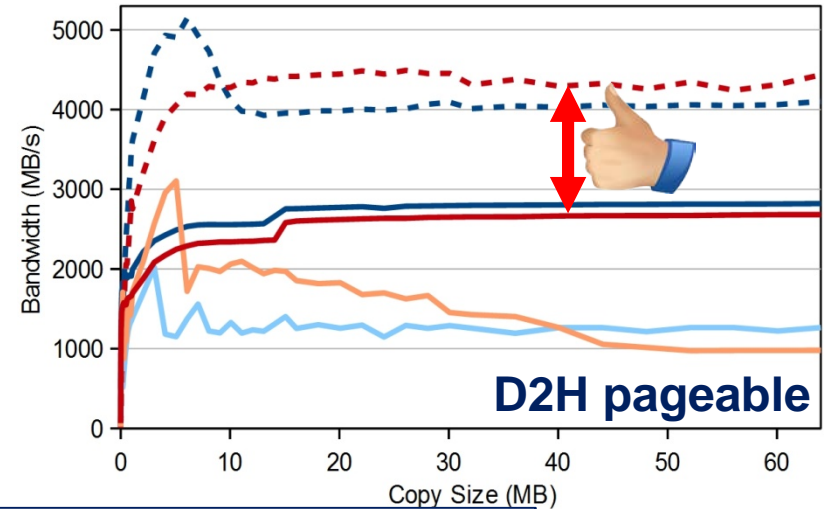
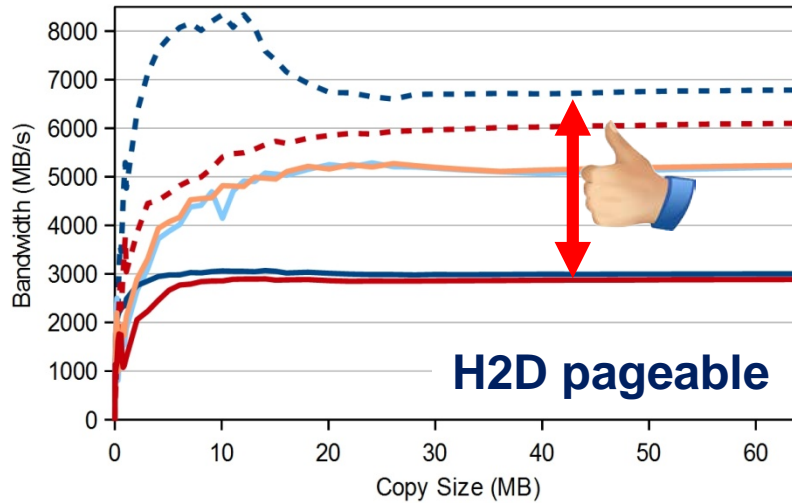
The main drawback of GPU virtualization is the reduced bandwidth to the remote GPU



# Using InfiniBand networks



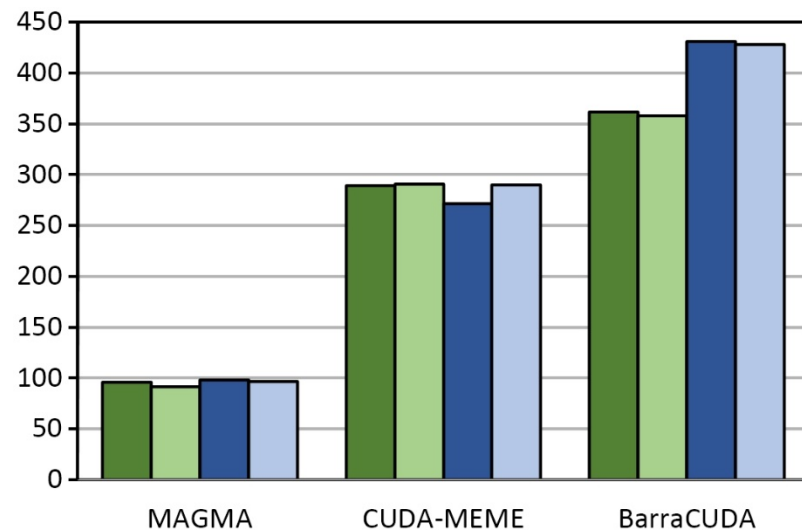
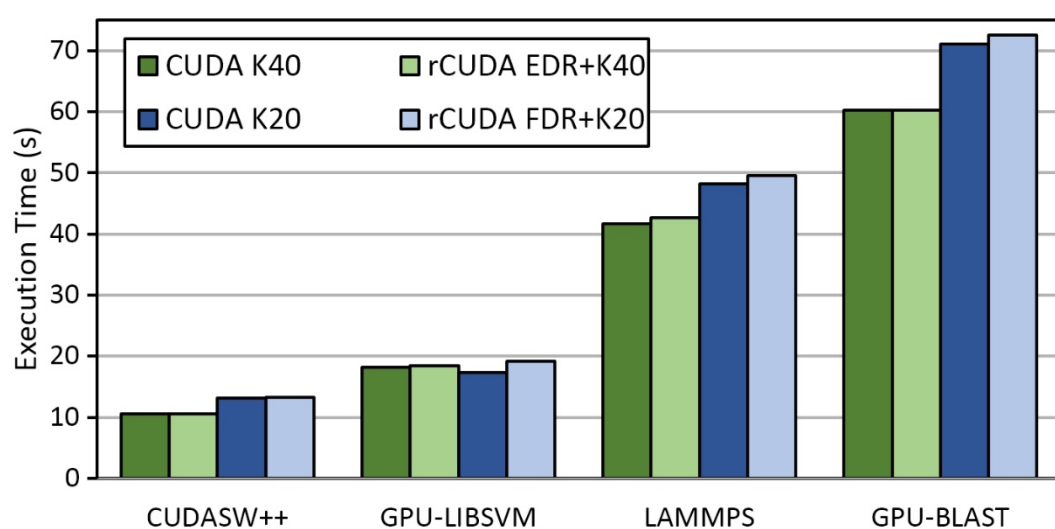
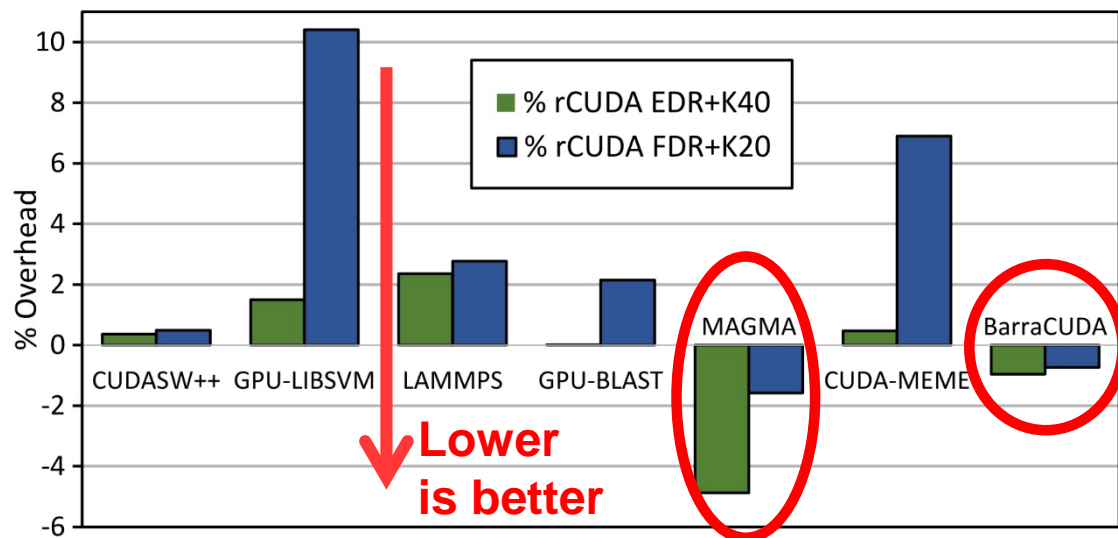
# Optimized transfers within rCUDA





# *rCUDA optimizations on applications*

- Several applications executed with CUDA and rCUDA
  - K20 GPU and FDR InfiniBand
  - K40 GPU and EDR InfiniBand



# 4th

Pros of "*remote GPU virtualization*"?

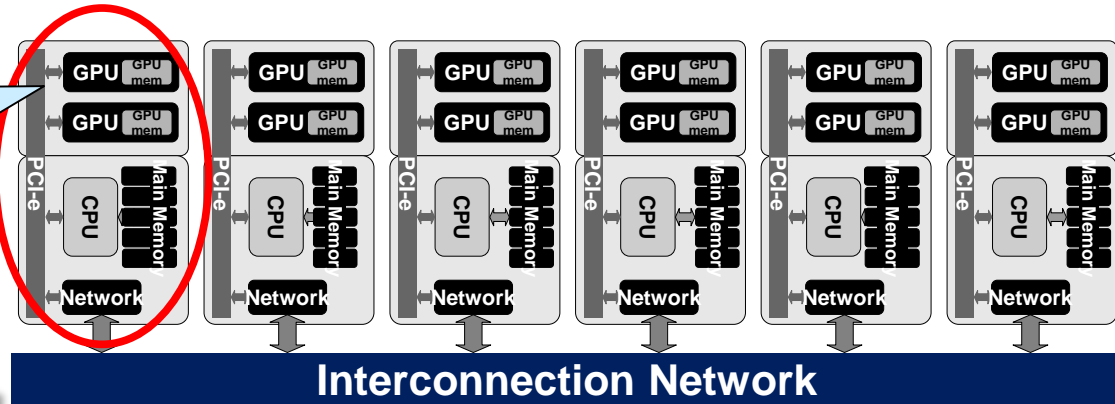


# 1: more GPUs for a single application

- GPU virtualization is useful for multi-GPU applications

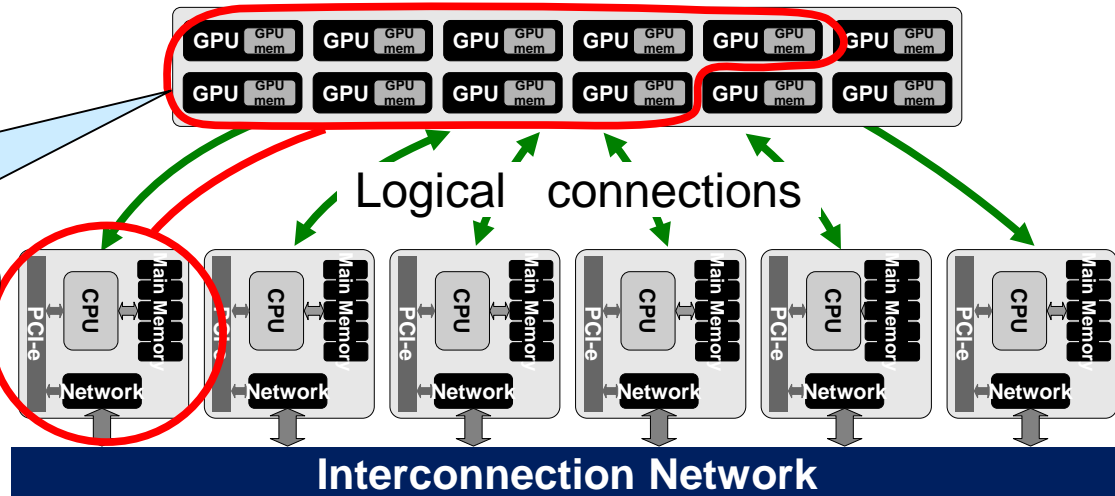
Only the GPUs in the node can be provided to the application

Without GPU virtualization



With GPU virtualization

Many GPUs in the cluster can be provided to the application



# 1: more GPUs for a single application

64  
GPUs!

```

bsc19421@nvb127:~
./deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 64 CUDA Capable device(s)

Device 0: "Tesla M2090"
  CUDA Driver Version / Runtime Version      5.0 / 5.0
  CUDA Capability Major/Minor version number: 2.0
  Total amount of global memory:              6144 MBytes (6442123264 bytes)
  (16) Multiprocessors x ( 32) CUDA Cores/MP: 512 CUDA Cores
  GPU Clock rate:                            1301 MHz (1.30 GHz)
  Memory Clock rate:                         1848 Mhz
  Memory Bus Width:                          384-bit
  L2 Cache Size:                             786432 bytes
  Max Texture Dimension Size (x,y,z)          1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)
  Max Layered Texture Size (dim) x layers      1D=(16384) x 2048, 2D=(16384,16384) x 2048
  Total amount of constant memory:             65536 bytes
  Total amount of shared memory per block:     49152 bytes
  Total number of registers available per block: 32768
  Warp size:                                  32
  Maximum number of threads per multiprocessor: 1536
  Maximum number of threads per block:         1024
  Maximum sizes of each dimension of a block:  1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:    65535 x 65535 x 65535
  Maximum memory pitch:                       2147483647 bytes
  Texture alignment:                          512 bytes
  Concurrent copy and kernel execution:        Yes with 2 copy engine(s)
  Run time limit on kernels:                   No
  Integrated GPU sharing Host Memory:          No
  Support host page-locked memory mapping:     No
  Alignment requirement for Surfaces:          Yes
  Device has ECC support:                      Disabled
  Device supports Unified Addressing (UVA):     Yes
  Device PCI Bus ID / PCI location ID:         2 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

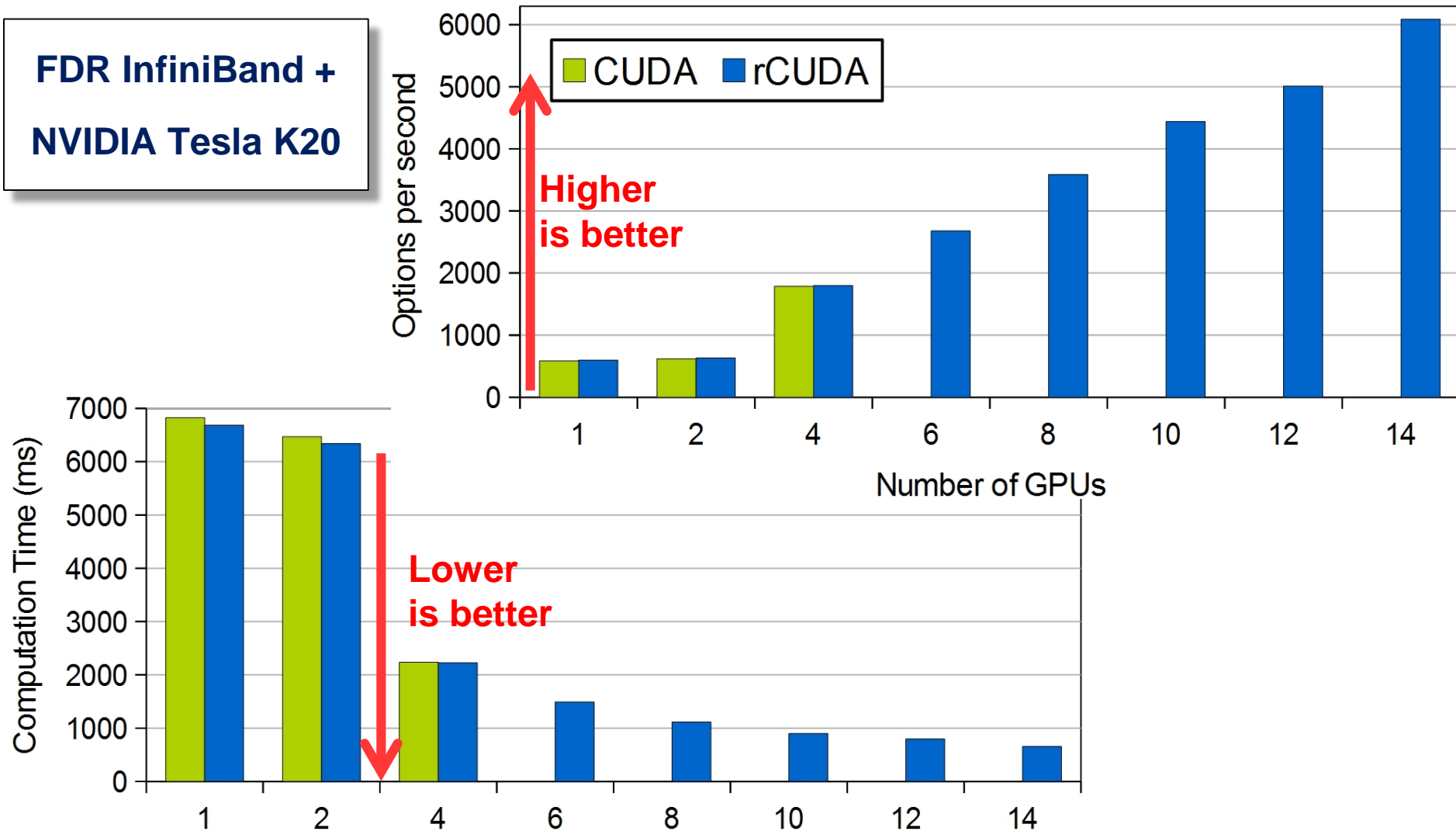
Device 1: "Tesla M2090"
  CUDA Driver Version / Runtime Version      5.0 / 5.0

```

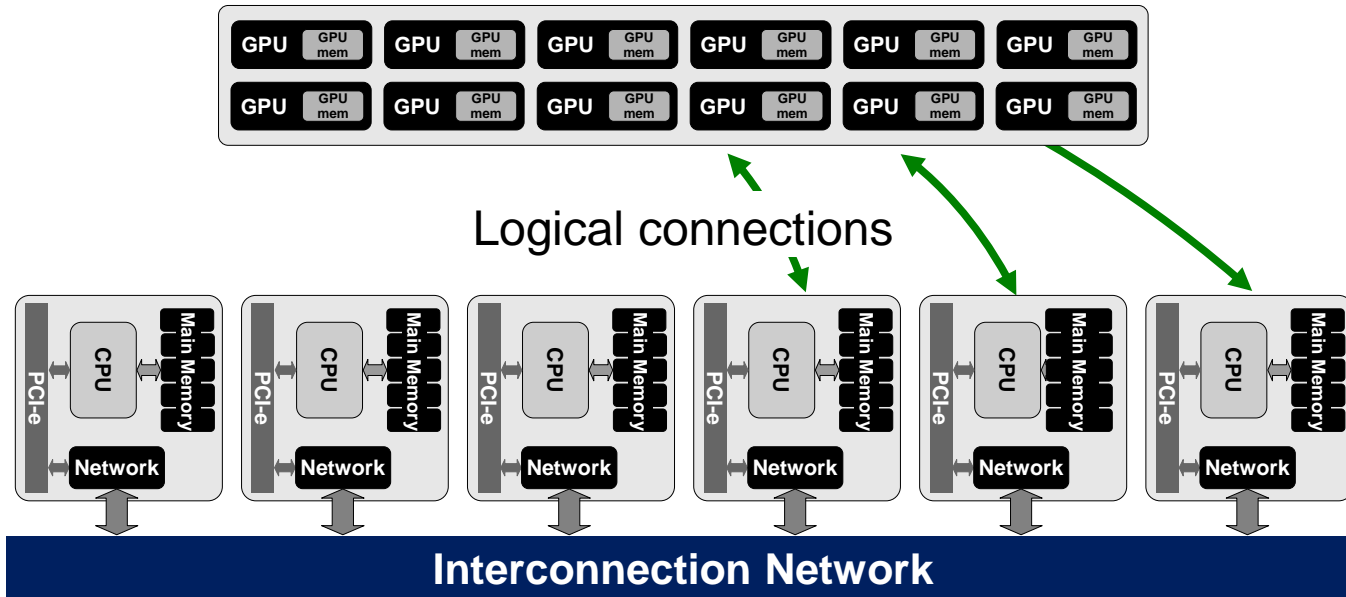
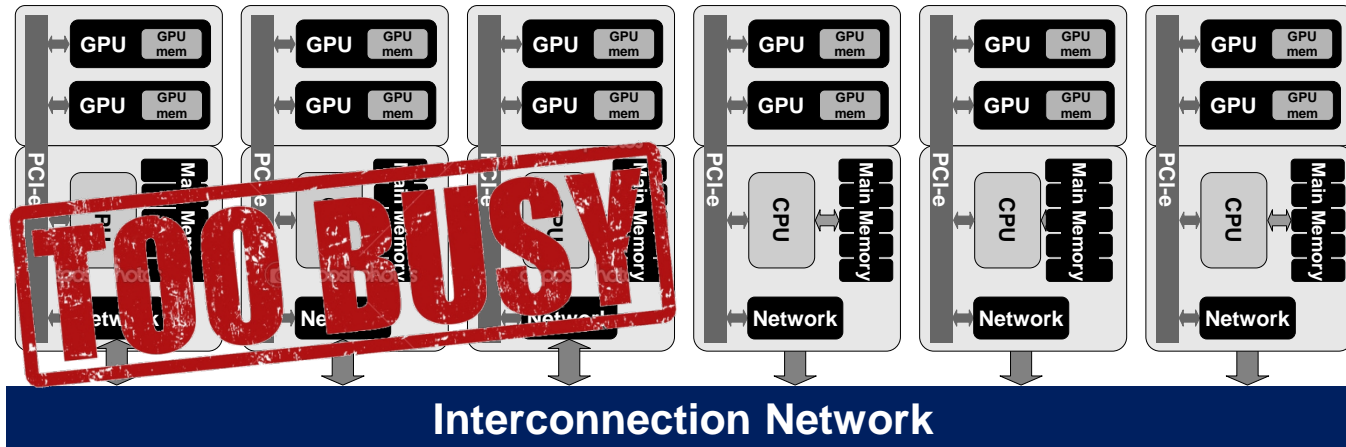
# 1: more GPUs for a single application

- MonteCarlo Multi-GPU (from NVIDIA samples)

**FDR InfiniBand +  
NVIDIA Tesla K20**

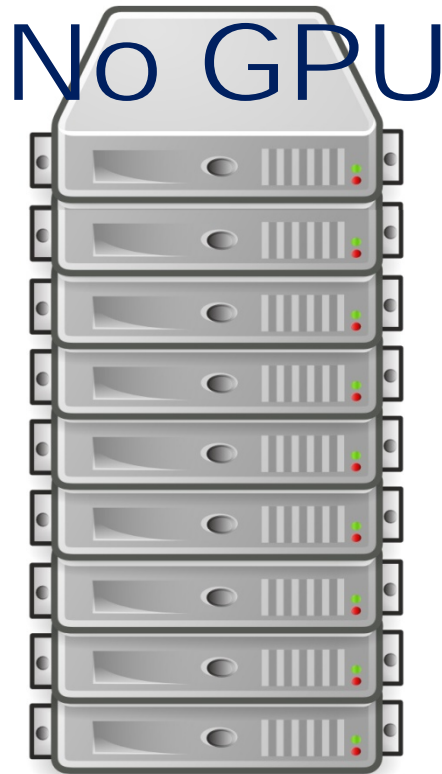


# 2: busy CPU cores do not block GPUs



# 3: cheaper cluster upgrade

- Let's suppose that a cluster without GPUs needs to be upgraded to use GPUs



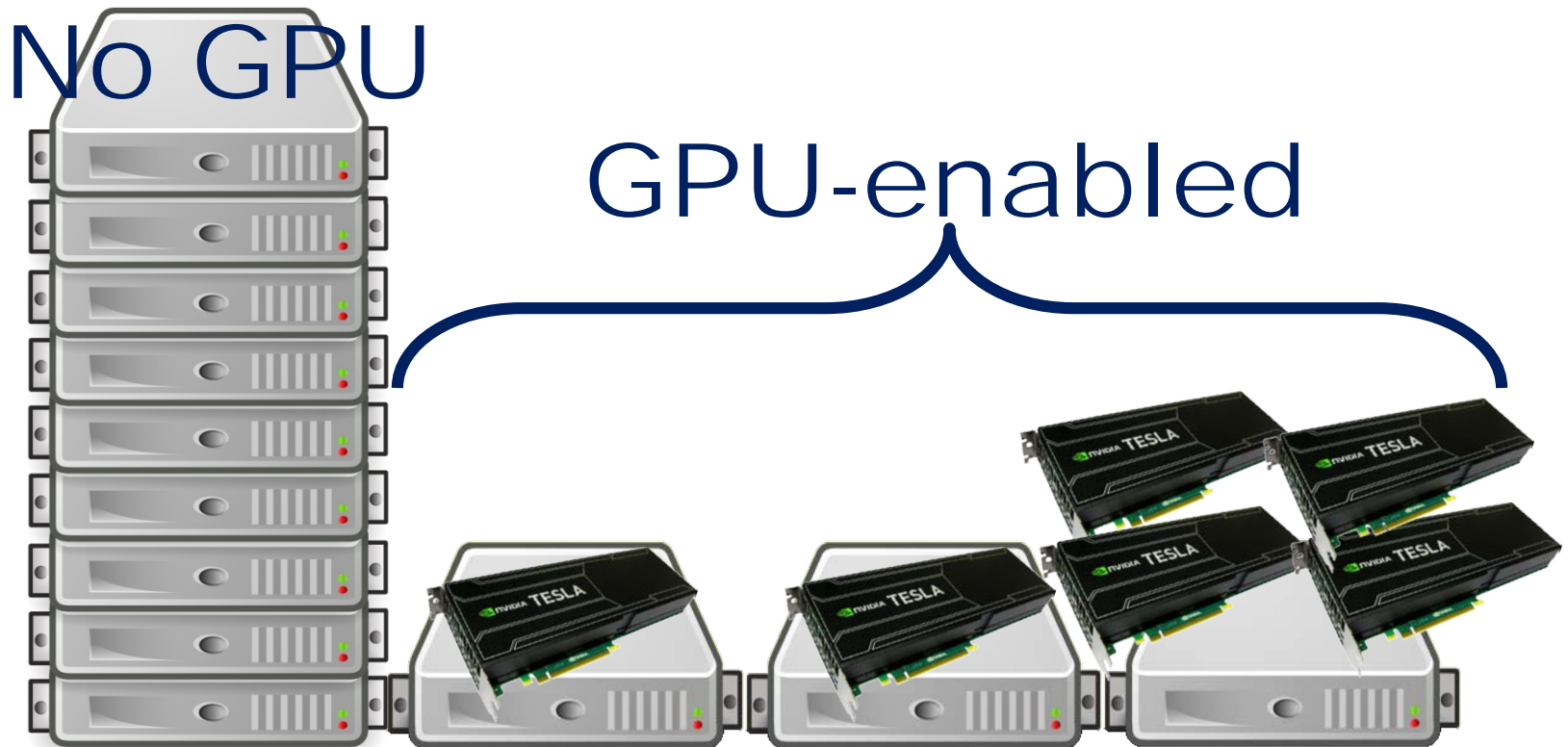
- GPUs require large power supplies**
  - Are power supplies already installed in the nodes large enough?
- GPUs require large amounts of space**
  - Does current form factor of the nodes allow to install GPUs?

The answer to both questions is usually **“NO”**



# 3: cheaper cluster upgrade

**Approach 1:** augment the cluster with some CUDA GPU-enabled nodes → **only those GPU-enabled nodes can execute accelerated applications**

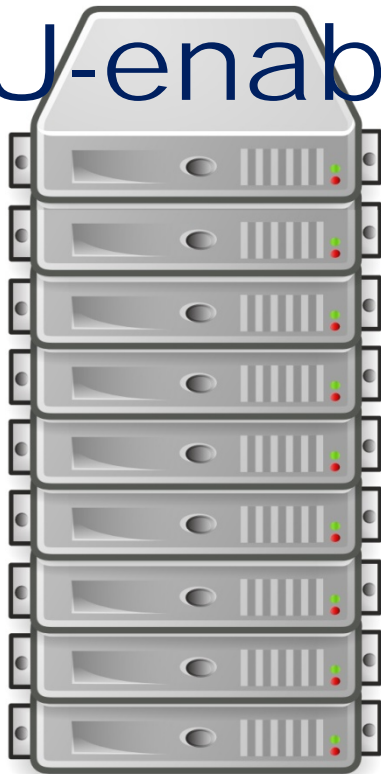


### 3: cheaper cluster upgrade

**Approach 2:** augment the cluster with some rCUDA servers → **all nodes can execute accelerated applications**



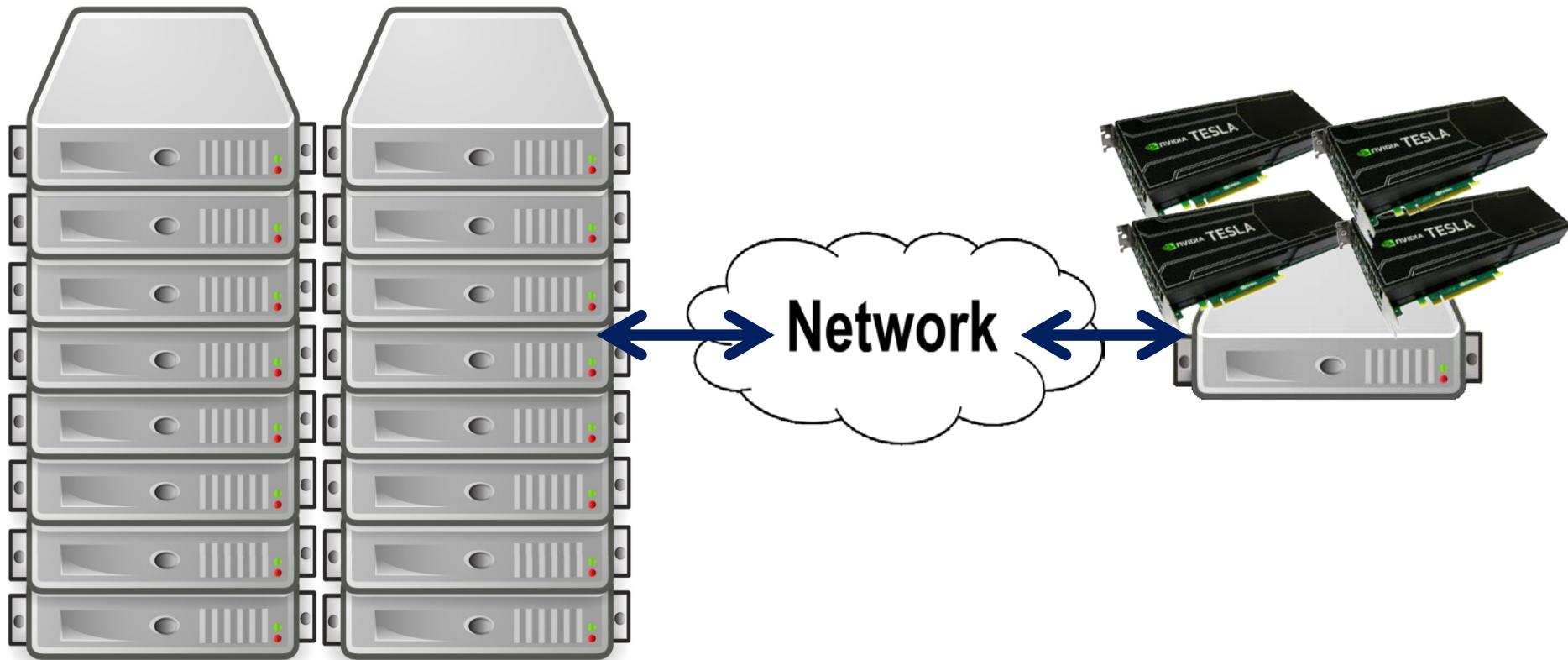
GPU-enabled



# 3: cheaper cluster upgrade

- Dual socket E5-2620v2 Intel Xeon + 32GB RAM + K20 GPU
- FDR InfiniBand based cluster

**16 nodes without GPU + 1 node with 4 GPUs**



# 3: cheaper cluster upgrade

- Applications used for tests:

- GPU-Blast (21 seconds; 1 GPU; 1599 MB)
- LAMMPS (15 seconds; 4 GPUs; 876 MB)
- MCUDA-MEME (165 seconds; 4 GPUs; 151 MB)
- GROMACS (167 seconds)

Set 1

Short execution time

- NAMD (11 minutes)
- BarraCUDA (10 minutes; 1 GPU; 3319 MB)
- GPU-LIBSVM (5 minutes; 1GPU; 145 MB)
- MUMmerGPU (5 minutes; 1GPU; 2804 MB)

Set 2

Long execution time

Non-GPU

- Three workloads:

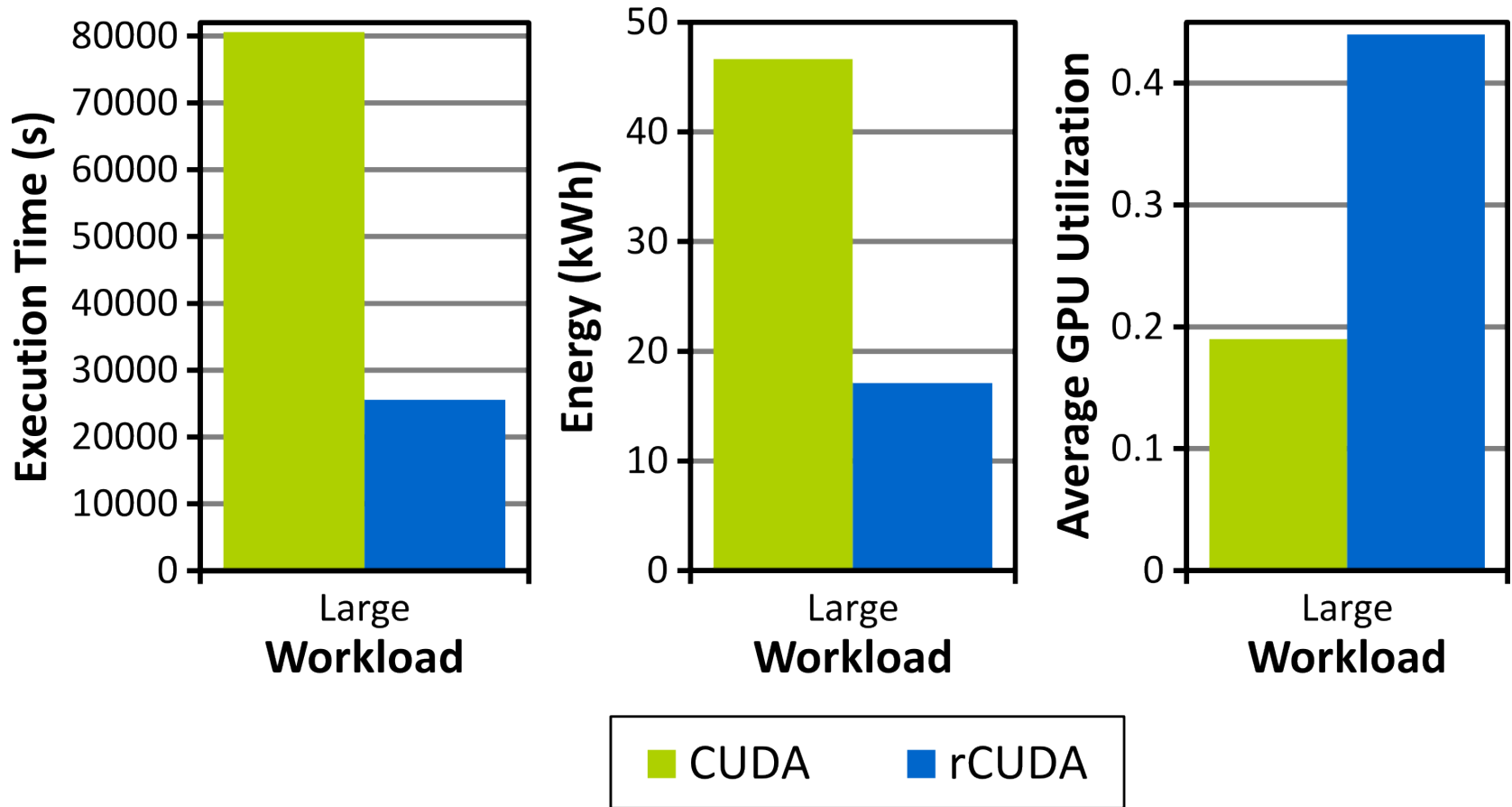
- Set 1
- Set 2
- Set 1 + Set 2

- Three workload sizes:

- Small (100 jobs)
- Medium (200 jobs)
- Large (400 jobs)

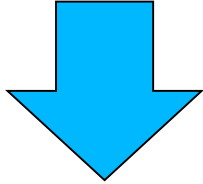


# 3: cheaper cluster upgrade



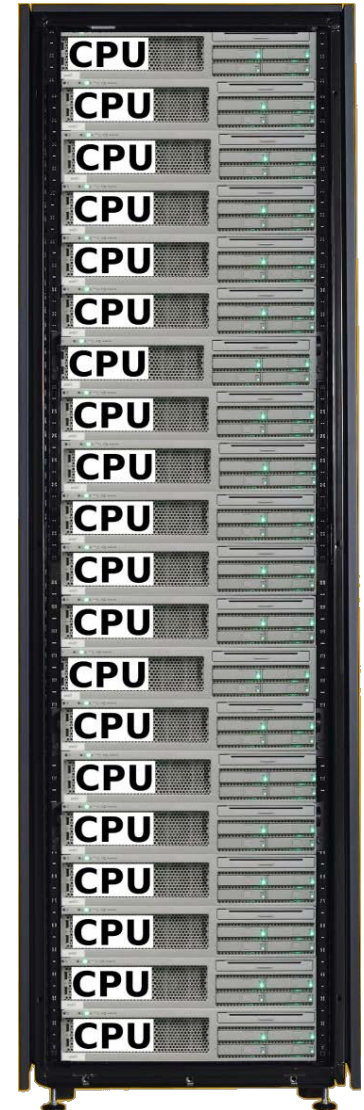
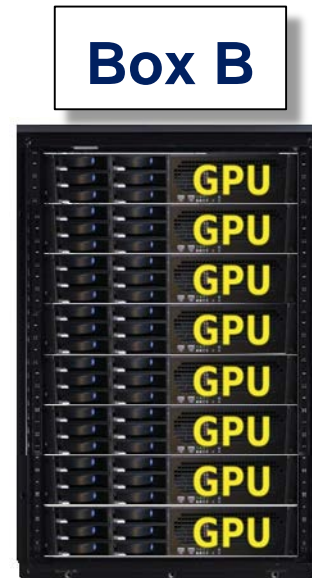
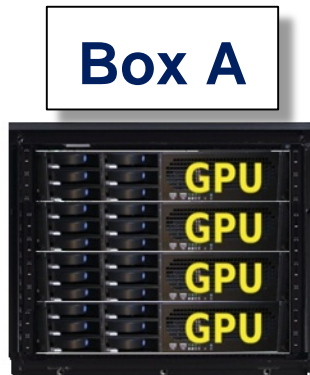
# 4: GPU task migration

- Box A has **4 GPUs** but only **one** is **busy**
- Box B has **8 GPUs** but only **two** are **busy**



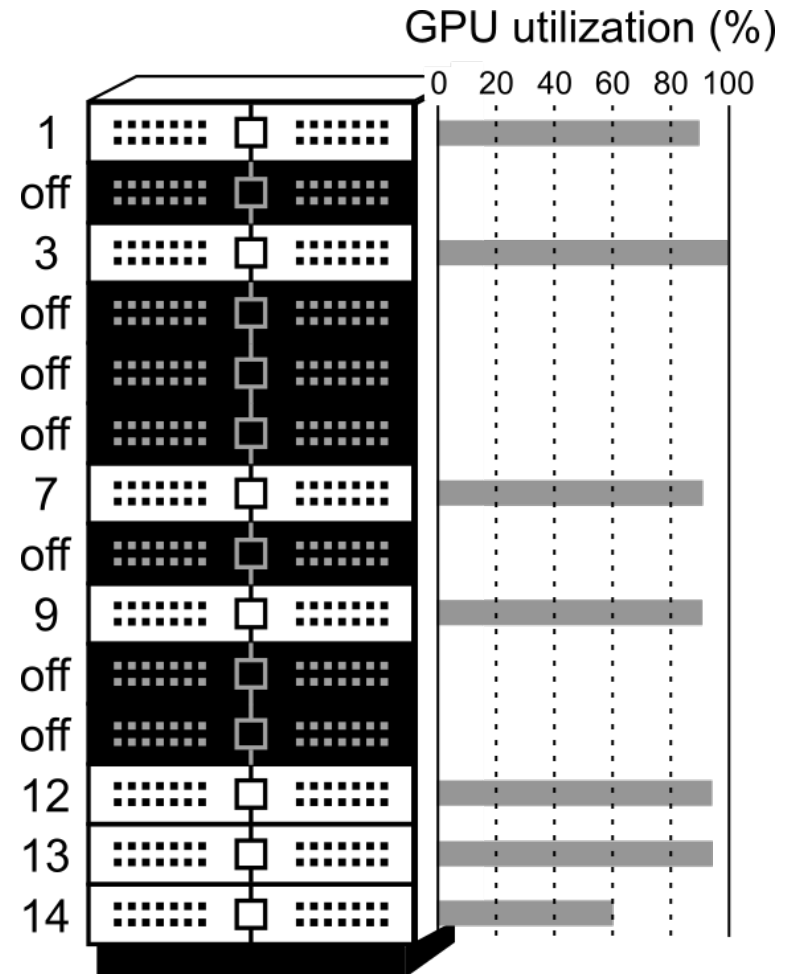
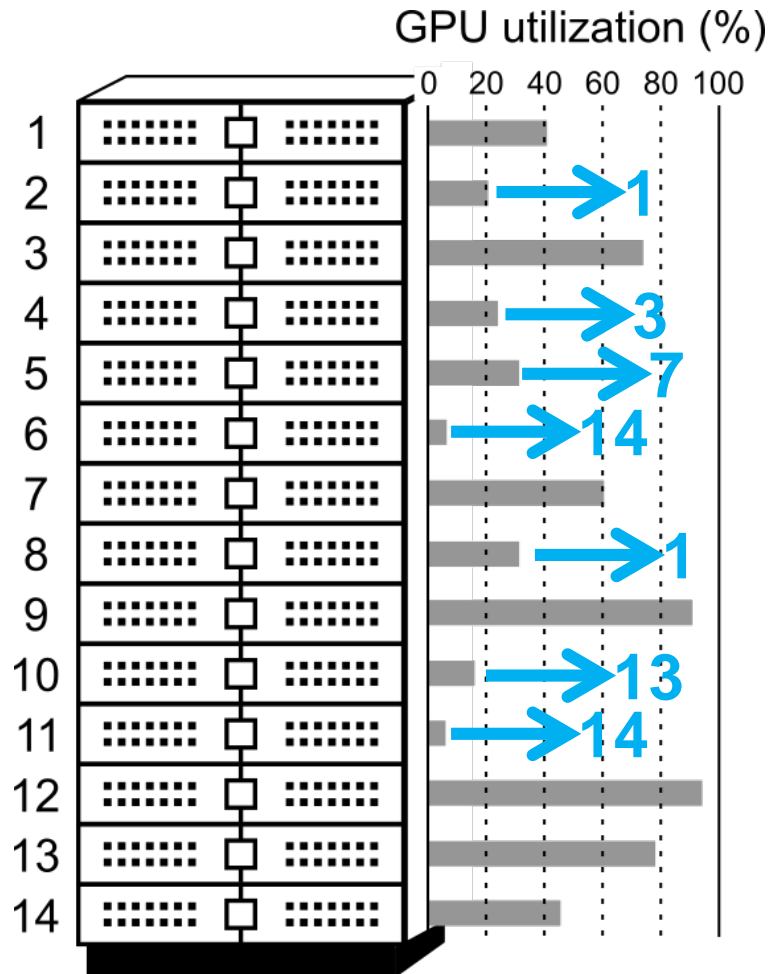
1. Move jobs from Box B to Box A and switch off Box B
2. Migration should be transparent to applications (decided by the global scheduler)

**Migration is performed  
at GPU granularity**





# 4: GPU task migration



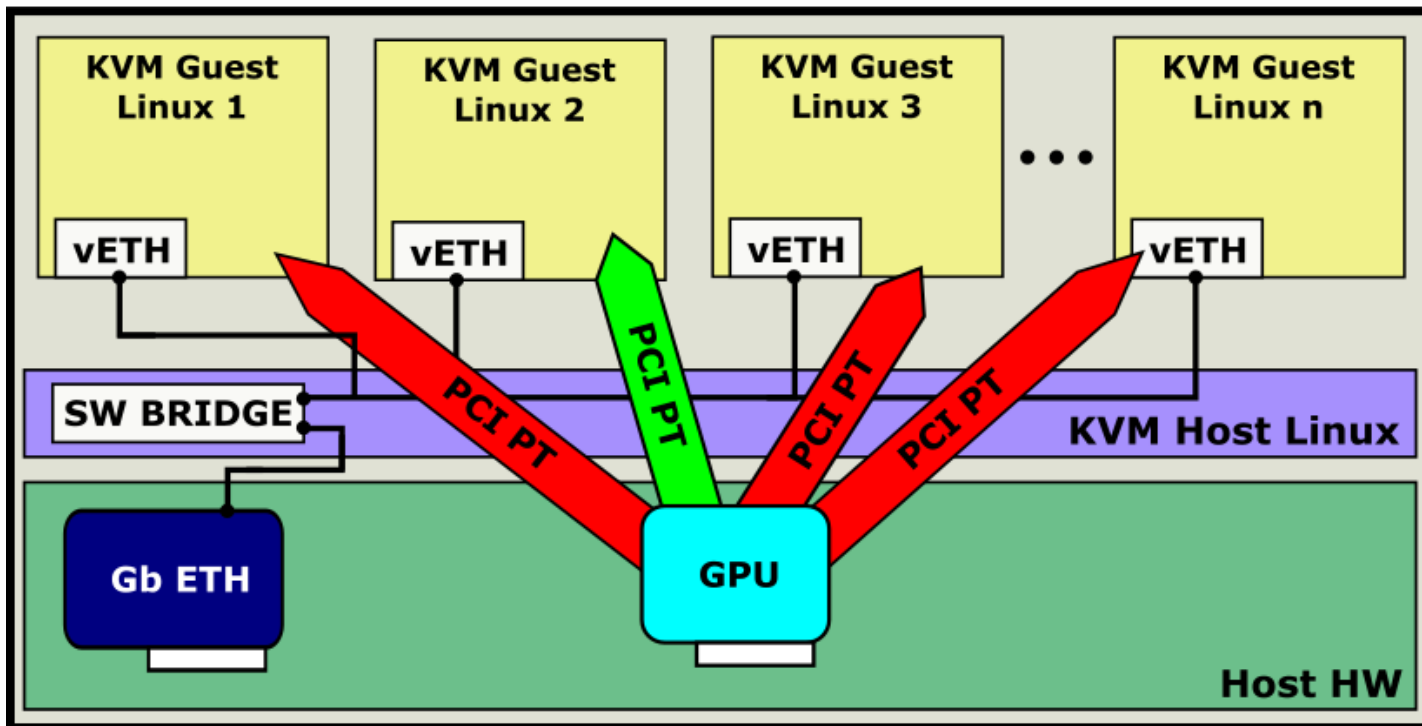
**Job granularity instead of GPU granularity**



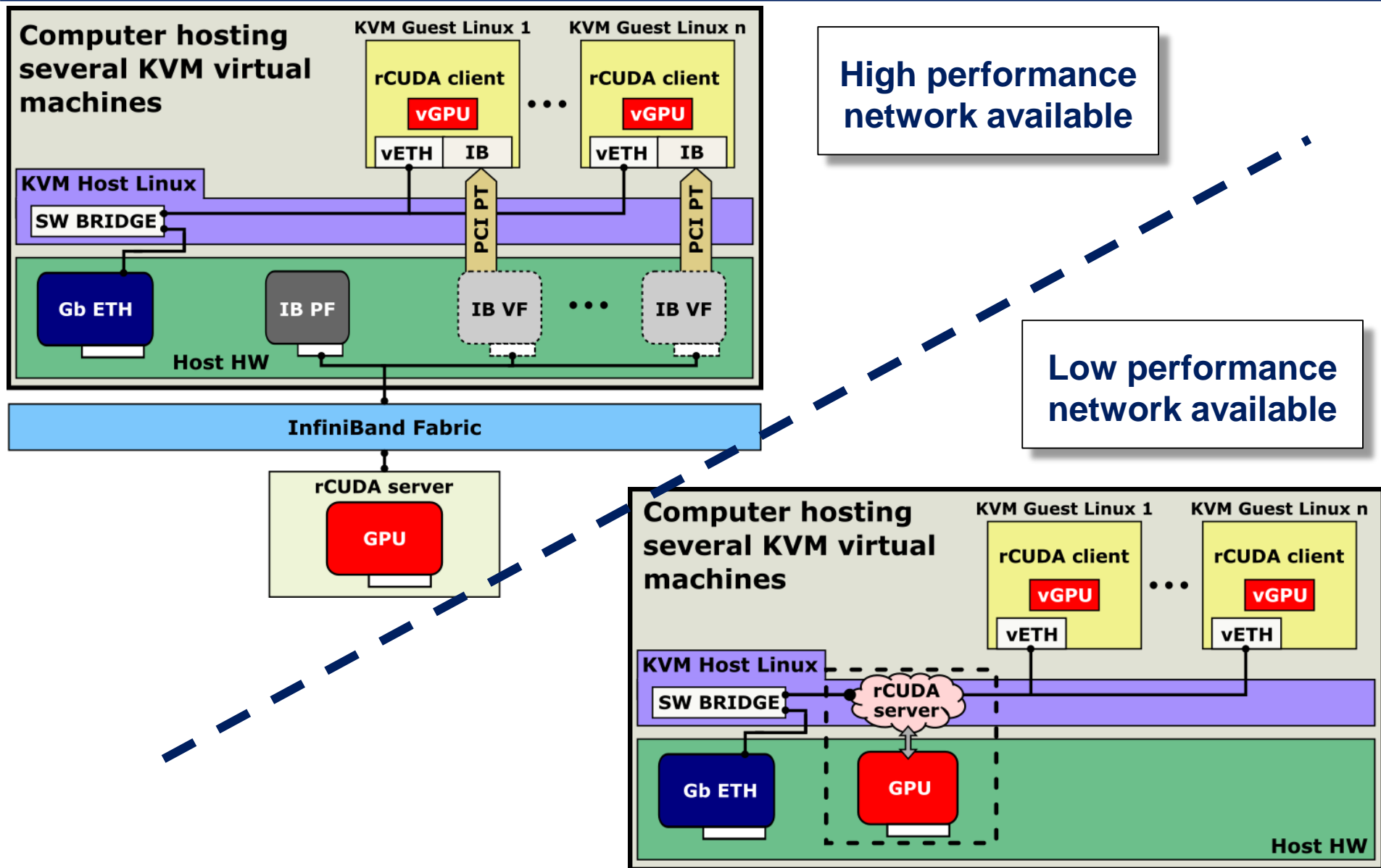
# 5: virtual machines can easily access GPUs

- The GPU is assigned by using PCI passthrough **exclusively to a single virtual machine**
- Concurrent usage of the GPU is not possible

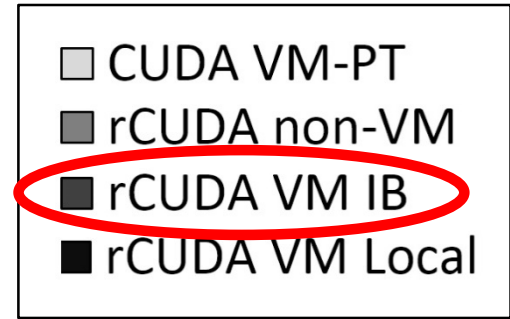
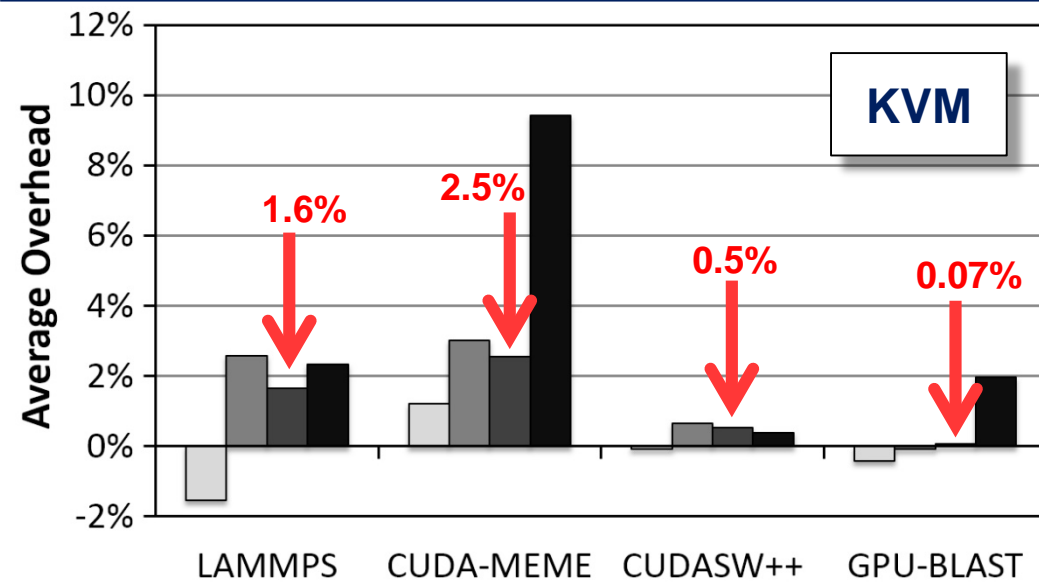
## Computer hosting several KVM virtual machines



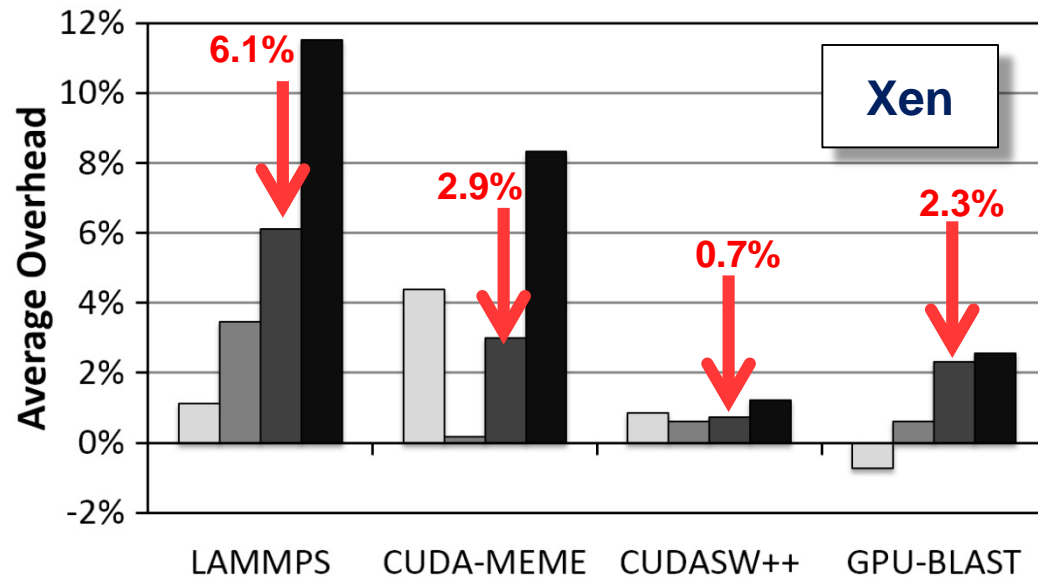
# 5: virtual machines can easily access GPUs



# 5: virtual machines can easily access GPUs



**FDR InfiniBand + K20 !!**



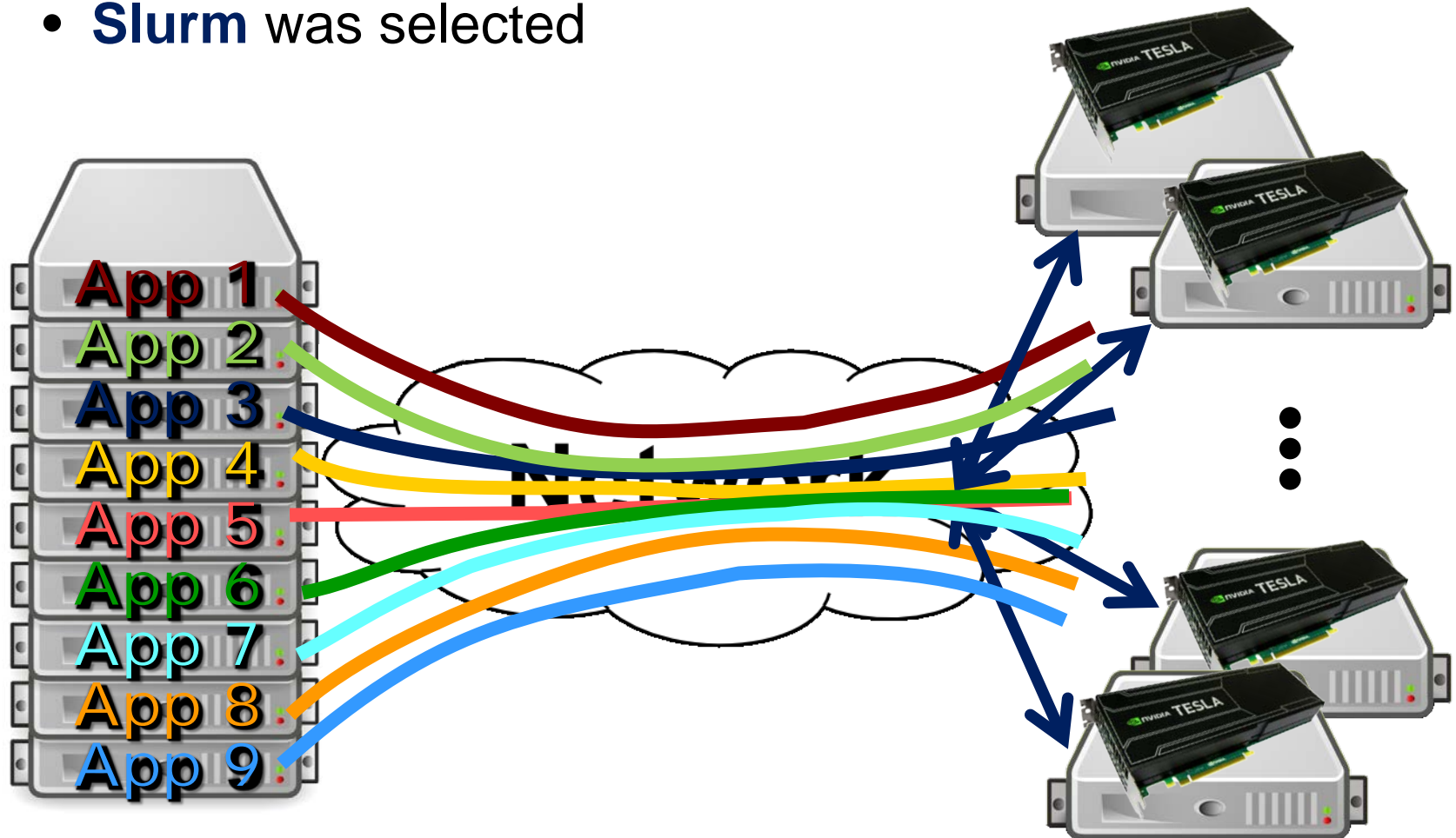
# 5th

What happens at  
the cluster level?



# *rCUDA at cluster level ... Slurm*

- **GPUs can be shared** among jobs running in remote clients
  - Job scheduler required for coordination
    - **Slurm** was selected



# Applications for studying rCUDA+Slurm

- Applications used for tests:

Non-GPU

- GPU-Blast (21 seconds; 1 GPU; 1599 MB)
- LAMMPS (15 seconds; 4 GPUs; 876 MB)
- MCUDA-MEME (165 seconds; 4 GPUs; 151 MB)
- GROMACS (167 seconds)

Set 1

Short execution time

- NAMD (11 minutes)
- BarraCUDA (10 minutes; 1 GPU; 3319 MB)
- GPU-LIBSVM (5 minutes; 1 GPU; 145 MB)
- MUMmerGPU (5 minutes; 1 GPU; 2804 MB)

Set 2

Long execution time

- Three workloads:

- Set 1
- Set 2
- Set 1 + Set 2

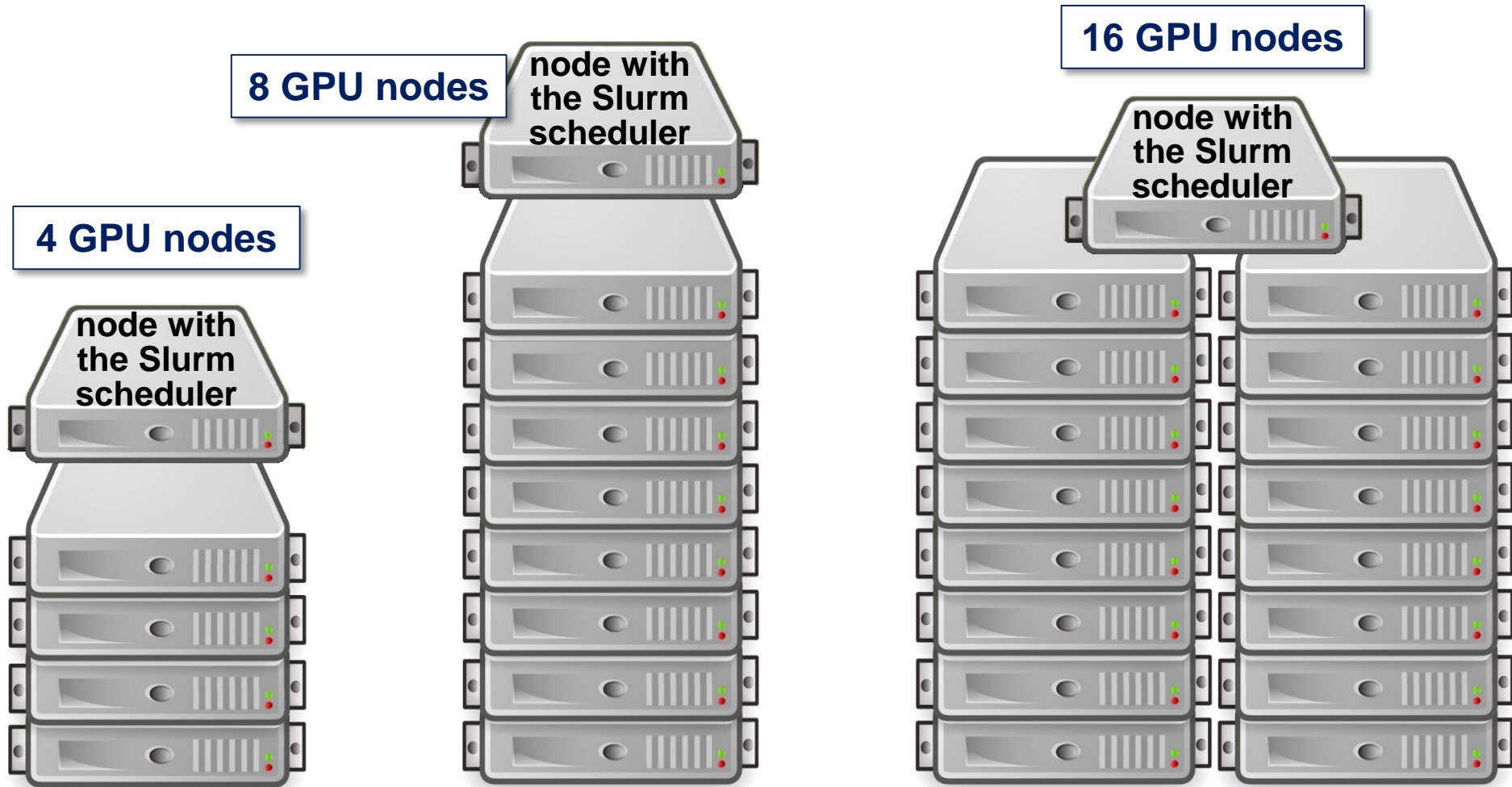
- Three workload sizes:

- Small (100 jobs)
- Medium (200 jobs)
- Large (400 jobs)



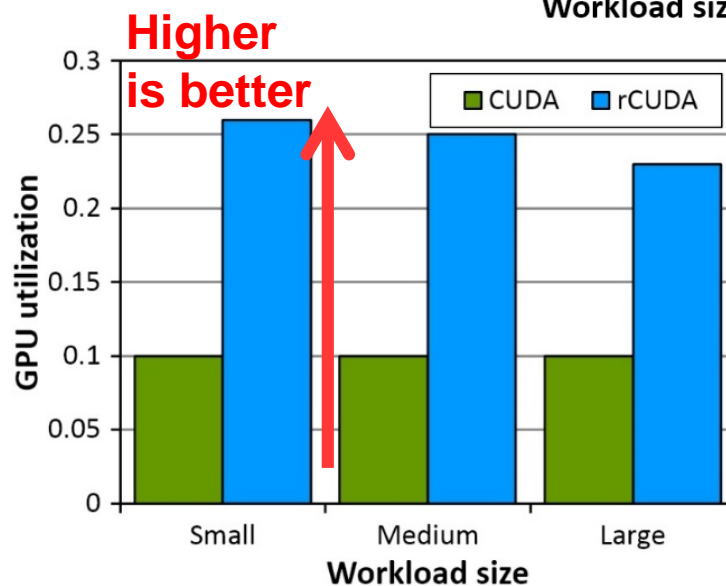
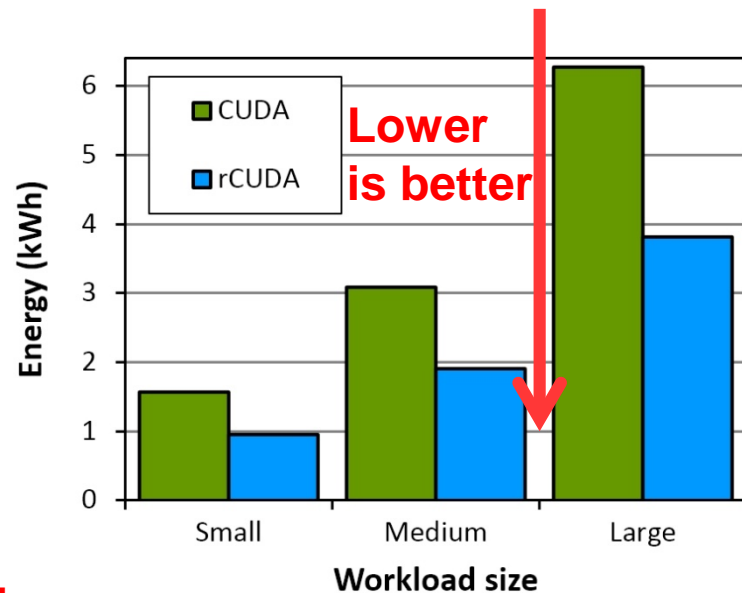
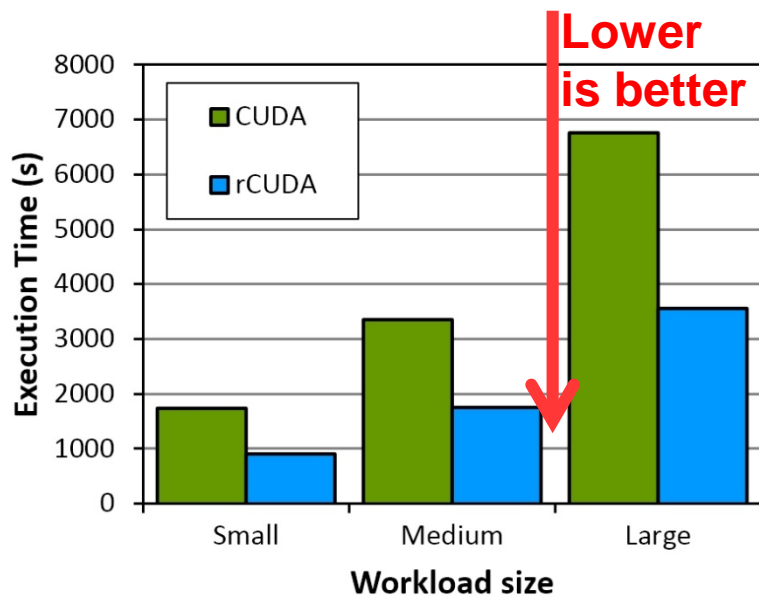
# Test bench for studying *rCUDA+Slurm*

- Dual socket E5-2620v2 Intel Xeon + 32GB RAM + K20 GPU
- FDR InfiniBand based cluster



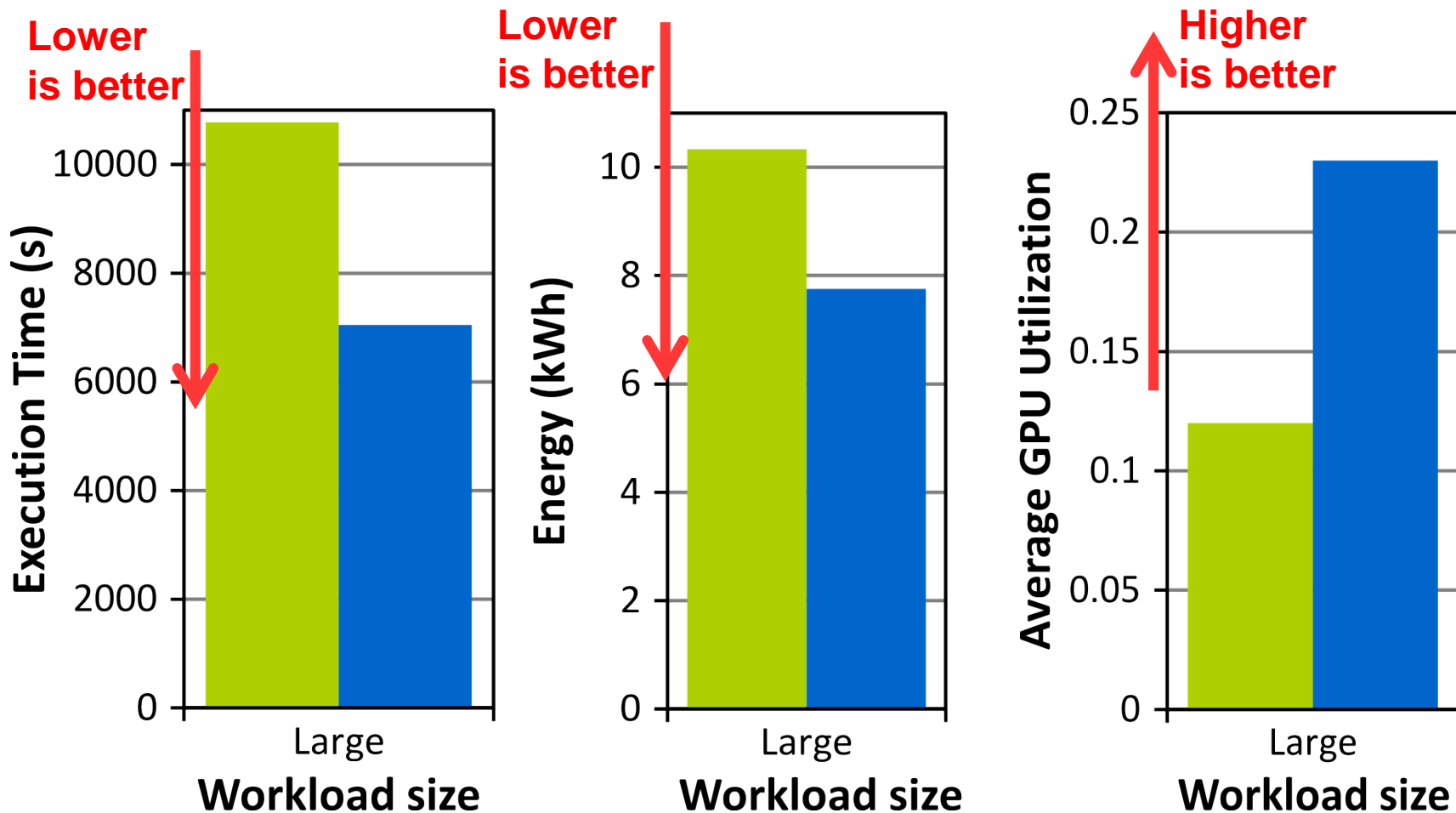


# Increased cluster performance with Slurm



**Results for 16 nodes  
with Set 1**

# Increased cluster performance with Slurm



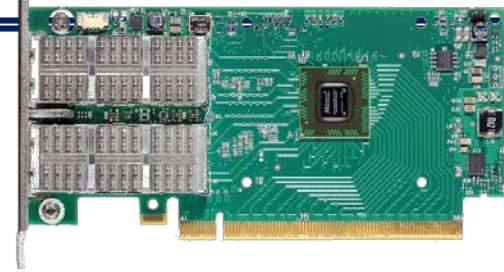
**Results for 16 nodes  
with Set 1+2**

■ CUDA ■ rCUDA

# 6th

... in summary ...





- **Cons:**

1. Reduced bandwidth to remote GPU (really a concern??)

- **Pros:**

1. Many GPUs for a single application
2. Concurrent GPU access to virtual machines
3. Increased cluster throughput
4. Similar performance with smaller investment
5. Easier (cheaper) cluster upgrade
6. Migration of GPU jobs
7. Reduced energy consumption
8. Increased GPU utilization



rCUDA is a development by Technical University of Valencia



**Get a free copy of rCUDA at**  
**<http://www.rcuda.net>**

**More than 650 requests world wide**



**rCUDA is a development by Technical University of Valencia**



# Thanks!

# Questions?

rCUDA is a development by Technical University of Valencia