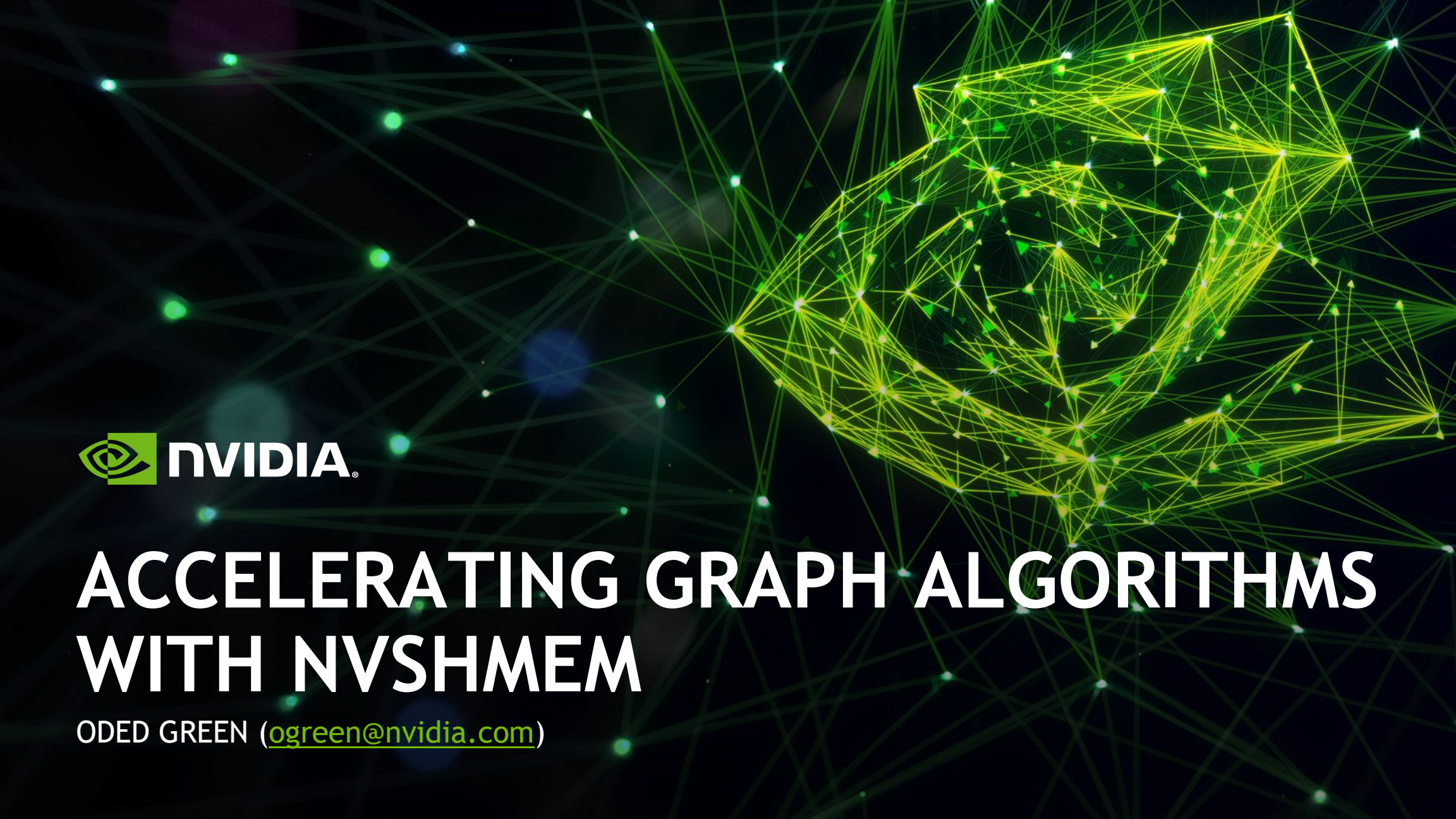


ACCELERATING GRAPH ALGORITHMS WITH NVSHMEM

ODED GREEN (ogreen@nvidia.com)

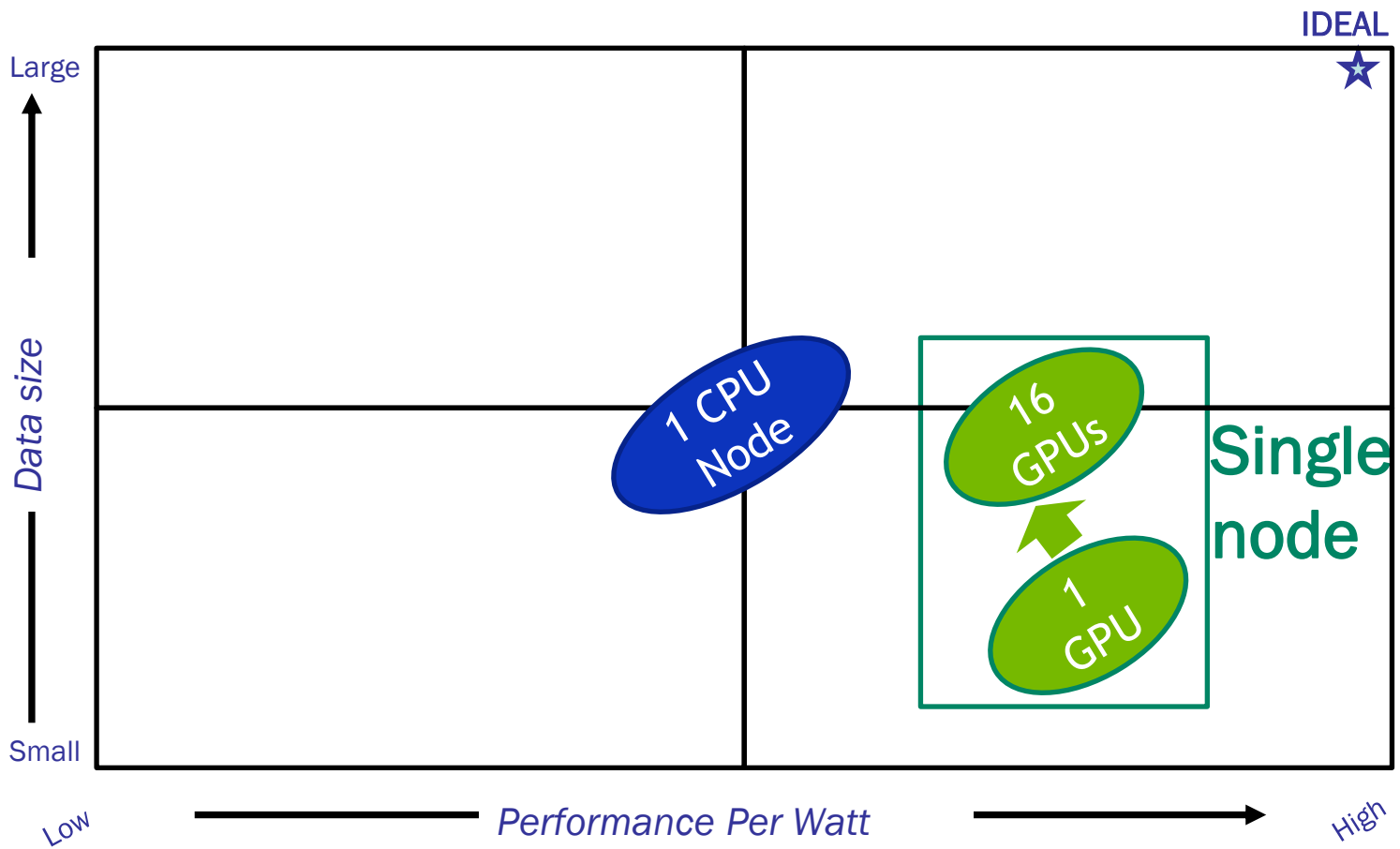
LAST YEAR AT CLASC'20

- I presented an algorithm that scaled BFS to 16 GPUs on a DGX-2 system
- Key highlights were
  - Algorithm showed how to efficiently utilize NVIDIA's NVSwitch interconnect
  - Great performance – nearly perfect linear strong-scaling speedups
  - Simple programming model – CUDA + OpenMP
  - Only one problem – limited to a single node.

PERFORMANCE VS. PROBLEM SIZE

Shared Memory Systems

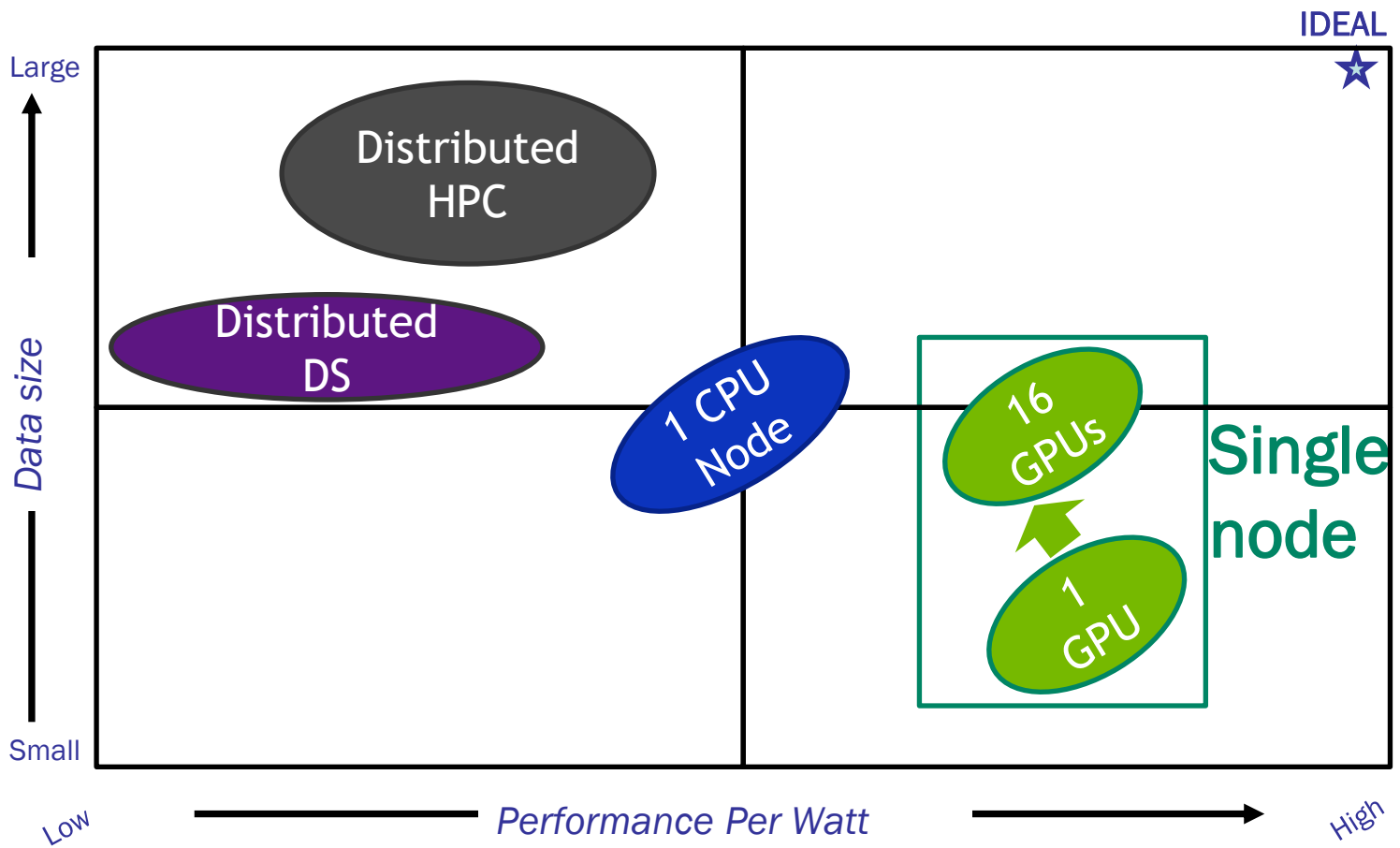
- Single node systems
- Fairly easy to program
- CPUs
  - Large memory
  - Limited compute
- One GPU
  - Limited memory
  - Lots of compute
- Multiple GPU in one node
  - Large memory
  - Massive compute



PERFORMANCE VS. PROBLEM SIZE

Distributed

- Multi node systems
- Harder to program
- HPC solutions
  - MPI+X
- Data science solutions
  - Focused on productivity



WHY DID I TRY NVSHMEM?

- Scale up and scale out problem sizes
- Simple and robust programming model (or so I was promised)
- Easy to convert OpenMP+CUDA or MPI+CUDA to NVSHMEM
  - I didn't believe it at first but saw it first hand

BENEFITS OF NVSHMEM

- NVSHMEM implements the OpenSHMEM parallel programming model for clusters of NVIDIA GPUs.
- NVSHMEM offers:
  - Lots of fine-grain primitives designed for communication/computation overlap
  - Fine-grained device-side GPU-to-GPU communication interfaces. Kernel does not need to be aborted for inter-GPU & inter-node communication.
  - Unified remote memory access & synchronization approaches
  - Integration with existing communication frameworks such as NVIDIA NCCL and transports such as UCX.

MY EXPERIENCE WITH NVSHMEM

A 6-month Journey

- Little experience with MPI. No experience with NVSHMEM or OpenSHMEM.
- Ported two different algorithms from OpenMP + CUDA to NVSHMEM
    1. BFS – used both OpenSHMEM & NVSHMEM
    2. Triangle counting (equivalent to a masked SpGEMM) – entirely using NVSHMEM
- Each algorithm took about 5-10 days to port
- Simple “ETL” took an equivalent amount of time to implement
- From single-node execution (one DGX-2 with 16 GPUs) to multi-node execution (8 DGX-1s with a combined 64 GPUs)

PERFORMANCE COMPARISON

OpenMP+CUDA Vs NVSHMEM

- OpenMP+CUDA
  - Tested on DGX-2 with 16x32GB GPUS connected via NVSwitch
  - Limited scalability on DGX-1
    - NVSWitch was a key enabler
- NVSHMEM
  - Tested on cluster of DGX-1s. 8x16GB GPUs connected via PCI-E, NVLink, and IB.
  - More resource bottlenecks (however, all of them are transparent)
- For Kron-27 graph (134M vertices and 4B edges):
  - DGX-2 (16 GPUs): 0.013sec.
  - 8 x DGX-1 (64 GPUs): 0.04sec.
- In the process of building a docker container to test both algorithms on the same hardware (DGX A100).

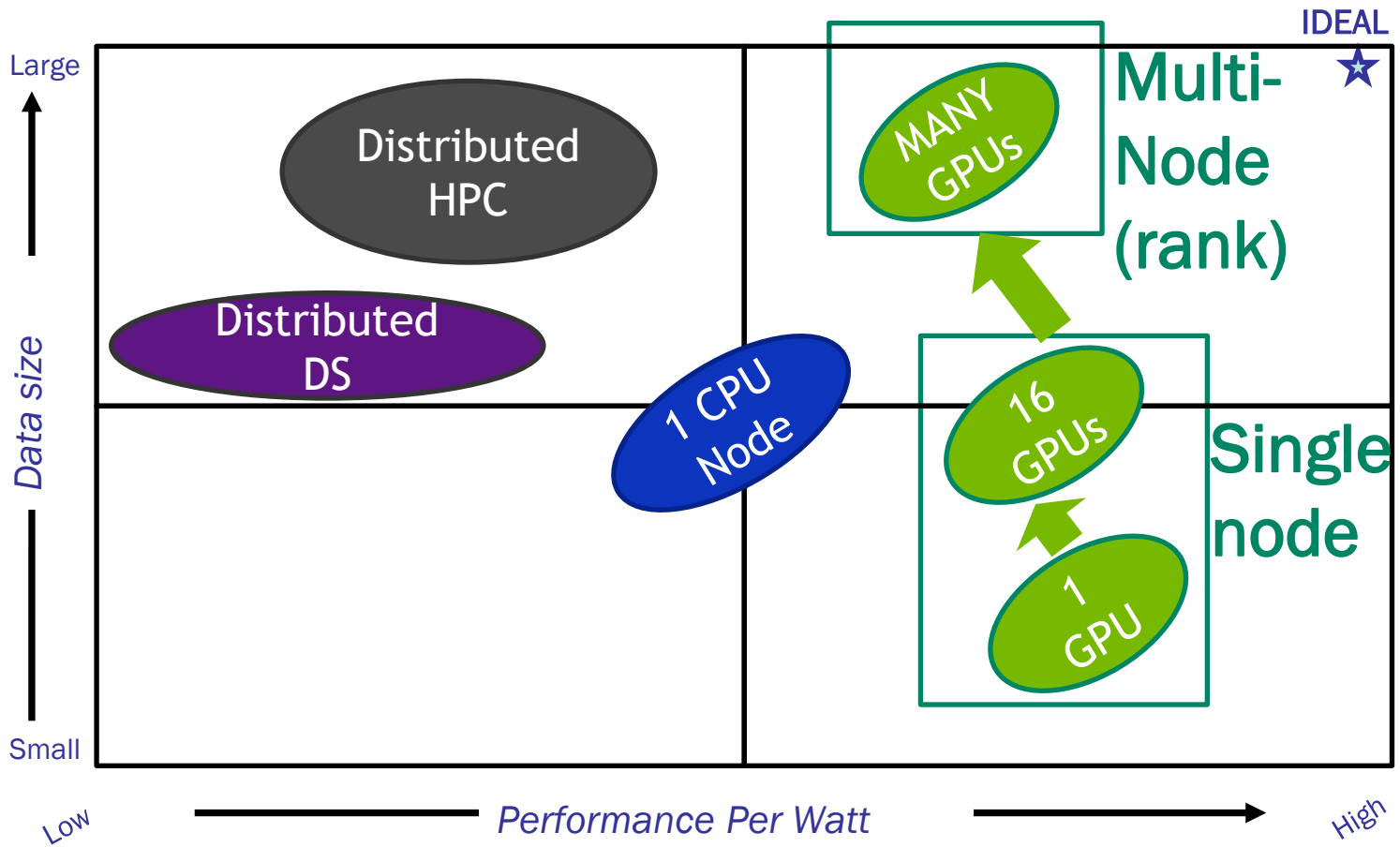
KEY INSIGHTS

- API is robust and offers lot of the communication functionality
  - Transfers can be initiated by both the host (CPU) and device (GPU)
  - Device transfers allow for fine grain transfers
- Parallelism is “implicit” similar to MPI (one process per parallel rank)
- Better off using only NVSHMEM and not mixing it with OpenSHMEM
  - Avoids some unnecessary transfers

PERFORMANCE VS. PROBLEM SIZE

Distributed with NVSHMEM

- Scales to larger problem sizes
- Utilizes GPUs capability
- Simple to program



SUMMARY

- NVSHMEM and its API shows great potential to accelerate sparse data problems
- Existing algorithms can be ported with relatively ease
- Sparse and data-dependent problems are the next frontier
  - As a community we should identify key primitives and build frameworks above the (NV)SHMEM platform

Want to learn more:

- Robert Zuppert (rzuppert@nvidia.com)
- Oded Green (ogreen@nvidia.com)

NVIDIA®