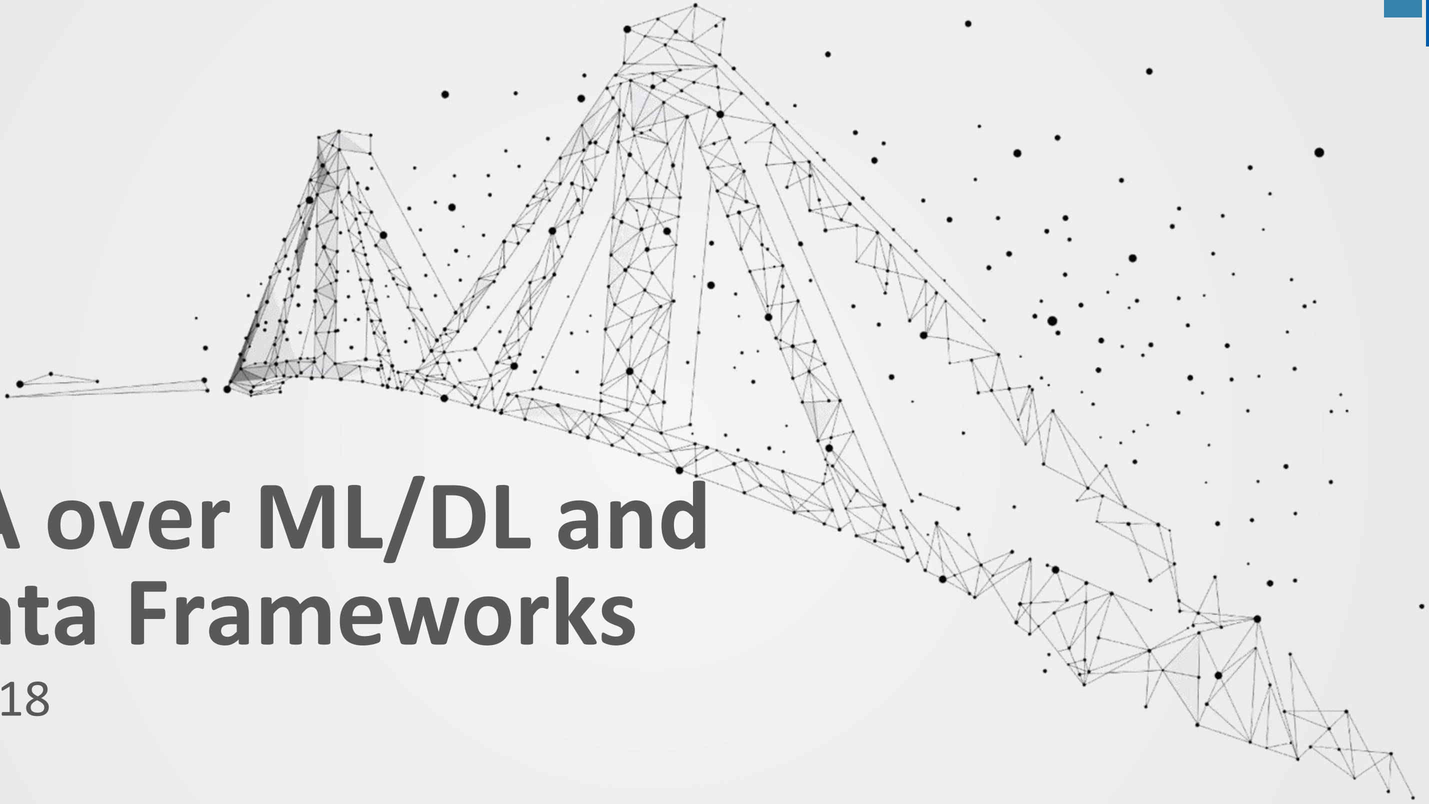
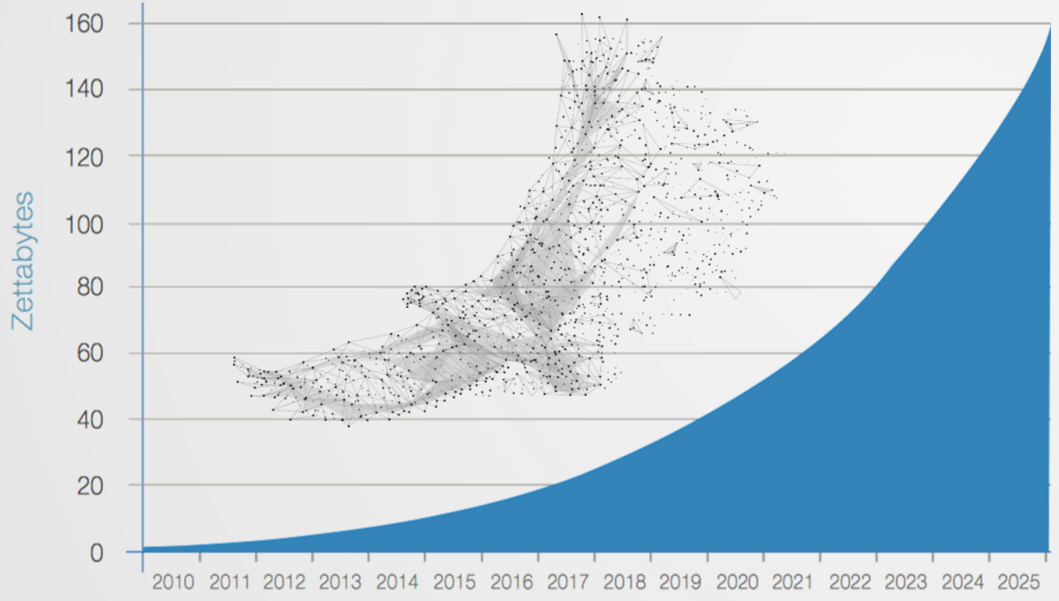


RDMA over ML/DL and Big Data Frameworks

SC Asia 2018

Ido Shamay

Exponential Data Growth Everywhere



- Cloud
- HPC
- Big Data
- Security
- Internet of Things
- Enterprise
- Storage
- Machine Learning
- Business Intelligence

Neural Networks Complexity Growth

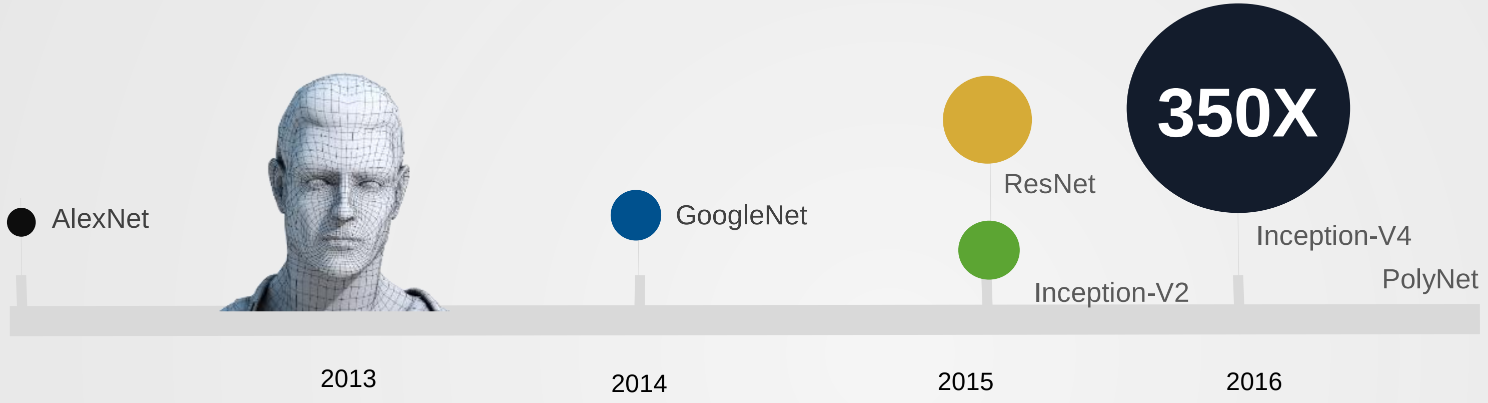
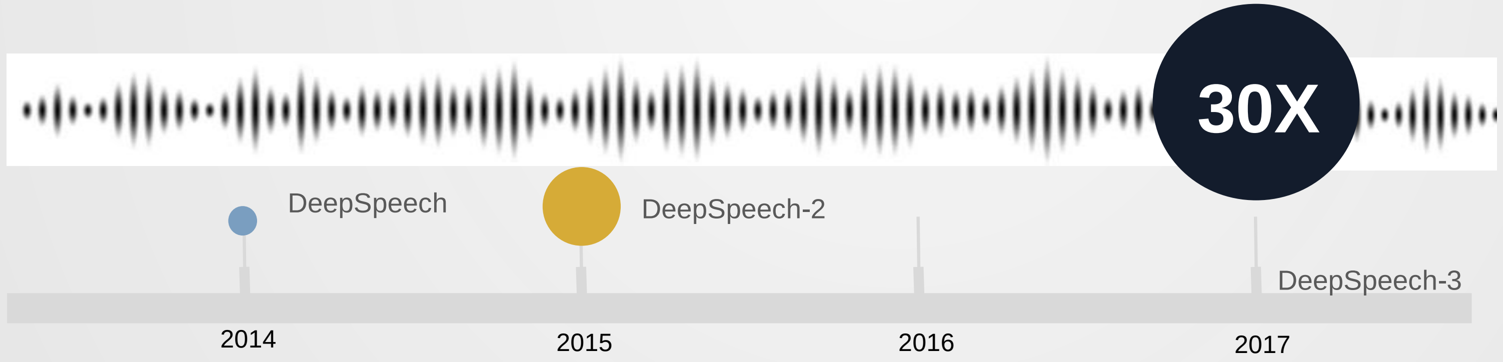


Image Recognition



Speech Recognition

Distributed Training use case

- Training with large data sets and the ever increasing neural networks can take long time
  - In some cases even weeks

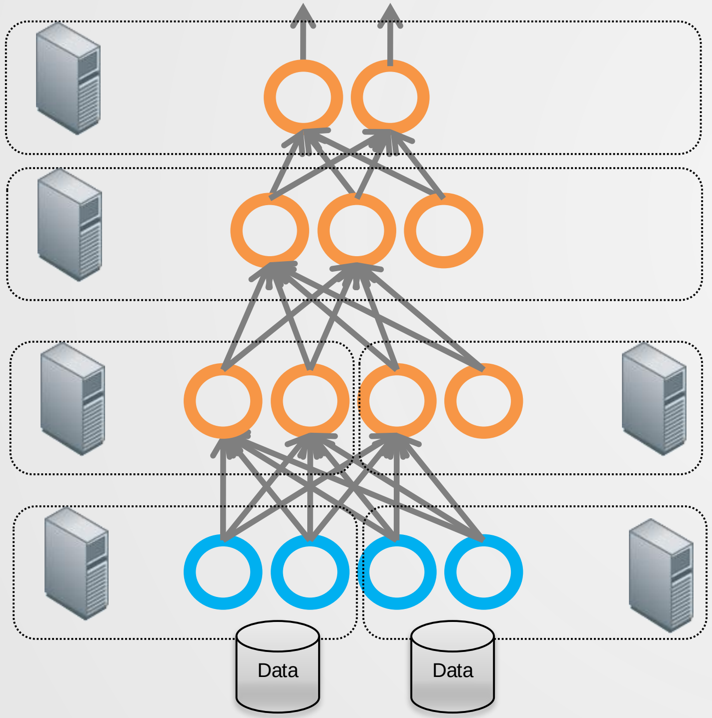
- In many cases training need to happen frequently
  - Model development and tuning
  - Real life use cases may require retraining regularly

- Accelerate training time by scale out architecture
  - Add workers (nodes) to reduce training time
  - Like in HPC.

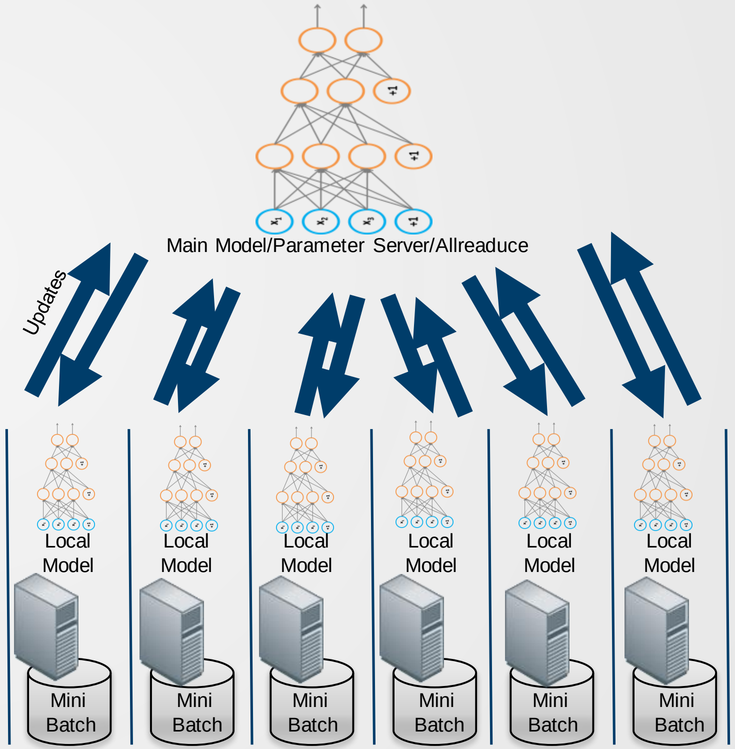
- 2 type of parallelism are now popular:
  - Data parallelism
  - Model parallelism

Model and Data Parallelism

Model Parallelism

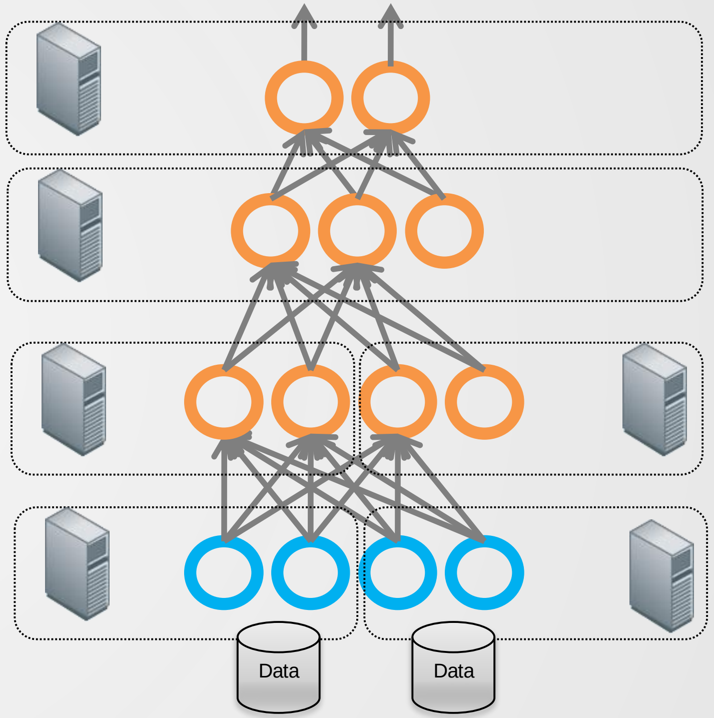


Data Parallelism



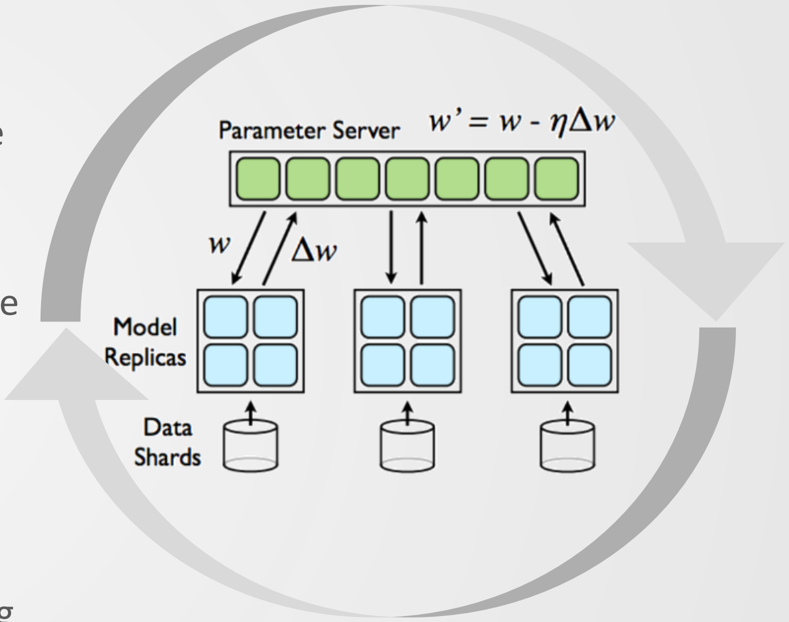
Model Parallelism

- Models size is limited by the compute engine (GPU for example)
- In some cases the model doesn't fit in compute
  - Large models
  - Small compute such as FPGAs
- Model parallelism slice the model and run each part on different compute engine
- Networking become a critical element
- High bandwidth, low latency are mandatory



Data Parallelism

- Data Parallelism communication is network intensive
  - Vast amount of data is distributed to a lot of different compute elements.
  - Training engines need to coordinate their neural networks weights and resynchronized with each other to get a cumulative benefit.
  - That synchronization is a performance bottleneck and requires High bandwidth, as models become larger and larger
    - Number of weights is increasing in exponential growth.
  - Usually characterized with Bursts on the network, as computing elements are synchronized.



Efficient Networking is a Key to Enable Data Parallelism

Data Parallelism

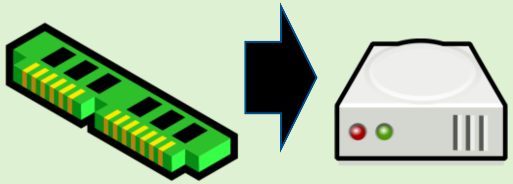
- Synchronous:
  - Workers start the training step together (synchronized) with the same variables.
  - Each worker compute gradients towards the global variables with its data (mini batch).
  - Gradients are synchronized between all workers
    - Either by sending it to parameter servers, which then average it.
    - Or using collectives operations.
  - The averaged gradient is enforced on the global parameters
  - Everyone starts a new step.
  - Effect on the training is that of the accumulated mini batch.

- Asynchronous:
  - Workers fetch model variables independently.
  - Each worker compute gradient it updates the global variables.
  - Very stochastic process.

Spark’s Shuffle use case

- Shuffling is the process of redistributing data across partitions (AKA repartitioning)

Shuffle Write

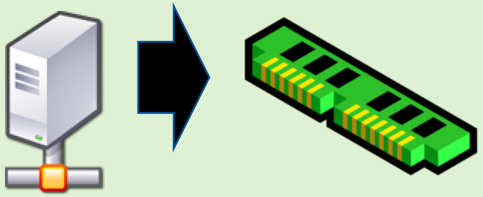


Worker nodes write their intermediate data blocks (Map output) into local storage, and list the blocks by partition

Broadcast block locations

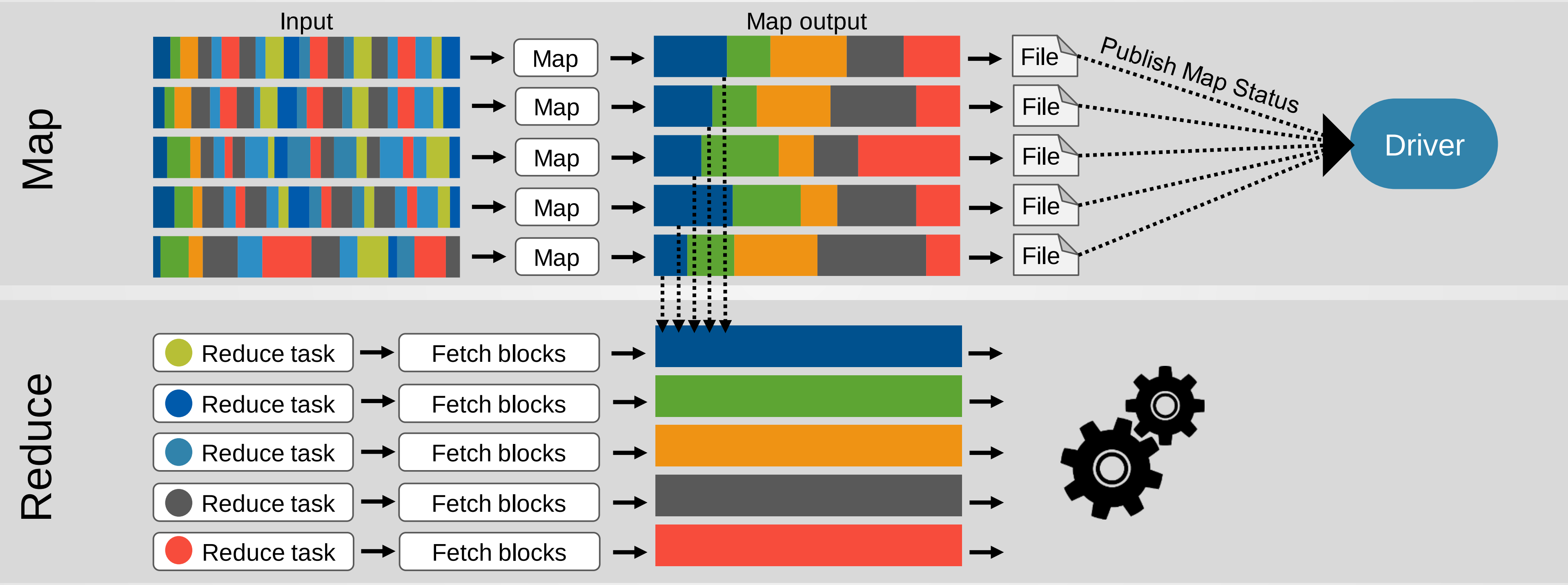
The master node broadcasts a combined list of blocks, grouped by partition

Shuffle Read



Each reduce partition fetches all of the blocks listed under its partition – from various nodes in the cluster

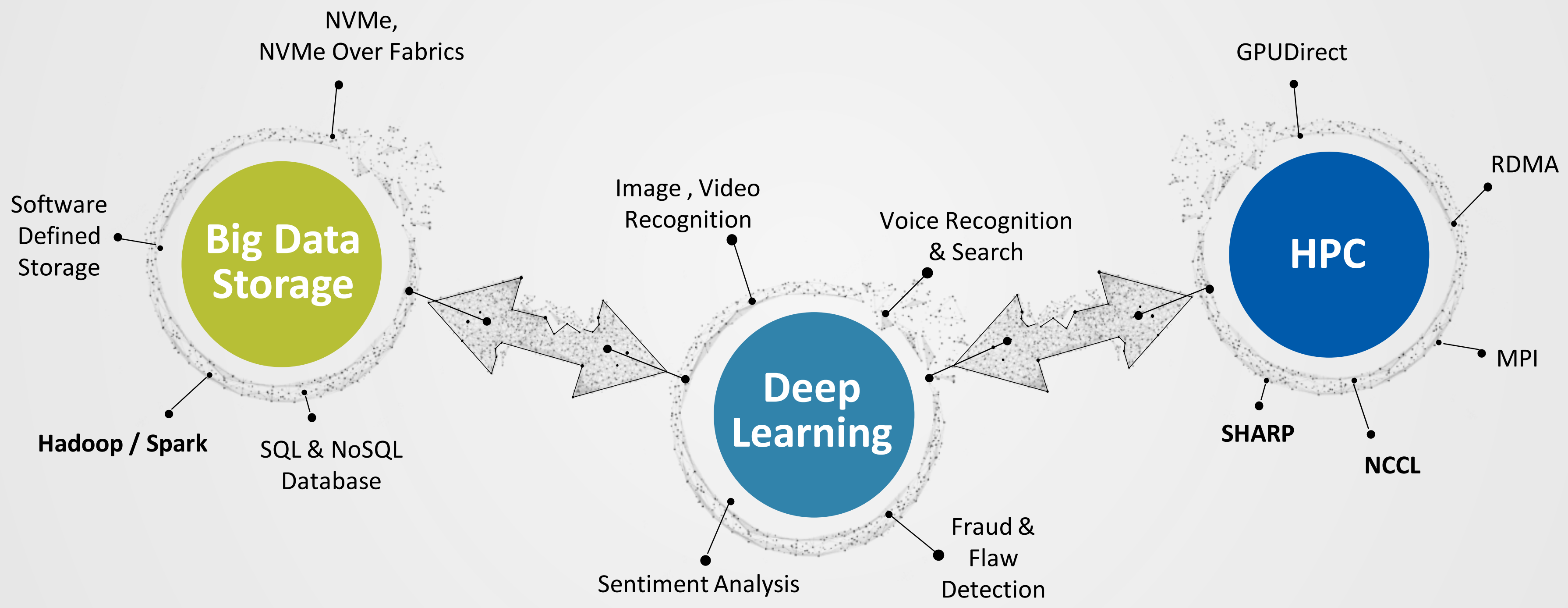
Spark’s Shuffle use case



Three Major Computing Categories

- Scientific Computing
  - Message Passing Interface (MPI), including MPI + OpenMP, is the Dominant Programming Model
  - Many discussions towards Partitioned Global Address Space (PGAS)
    - UPC, OpenSHMEM, CAF, UPC++ etc.
  - Hybrid Programming: MPI + PGAS (OpenSHMEM, UPC)
  - Relatively small-sizes message (order of kilobytes)
  - CPU-based communication buffers
- Deep Learning
  - Caffe, CNTK, TensorFlow, and many more
  - Widespread popularity of accelerators like Nvidia GPUs
  - Most of the frameworks are exploiting GPUs to accelerate training
  - Diverse range of applications –Image Recognition, Cancer Detection, Self-Driving Cars, Speech Processing etc.
  - Unusually large message sizes (order of megabytes)
  - Most communication based on GPU buffers
- Big Data/Enterprise/Commercial Computing
  - Focuses on large data and data analysis
  - Spark and Hadoop (HDFS, HBase, MapReduce)

Three Major Computing Categories



How do those applications looks like ?

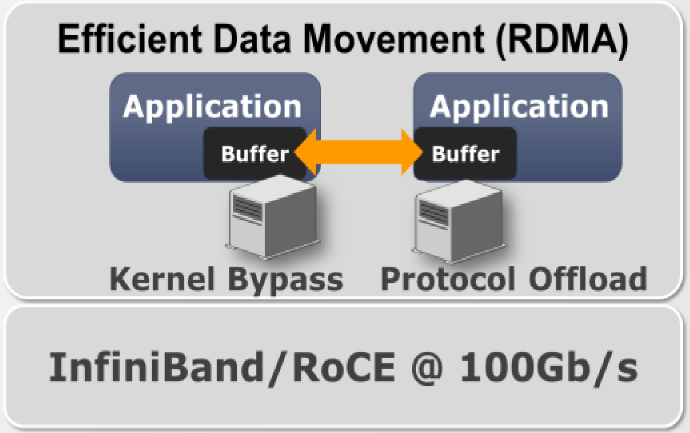
- Most of the time application is doing computation
  - Which is what actually interests the users
- Parallel communication and computation
  - Start non-blocking send / receive
  - Compute the data for next phase
  - Wait for completion of all requests
- Typical application flow:
  - Init:
    - Read input file
    - Decompose the problem
  - Work Loop:
    - Compute
    - Data exchange
  - Finalize:
    - Write output file

The secret sauce

- Low Latency (< 1 usec)
- High Bandwidth (> 100 Gb/s)
- Scalability
  - Efficient support for communications on large
- OS bypass (direct access to the hardware from the user level)
- Remote Direct Memory Access (avoid memory copies in communication on stack)
  - Read, Write, Atomics
- Offloads
  - Collective operations, support for non-contiguous data, GPU-Direct, Peer-Direct, tag-matching, etc.
  - Low software overheads
- Low memory footprint (as much as possible)
- Performance portable APIs

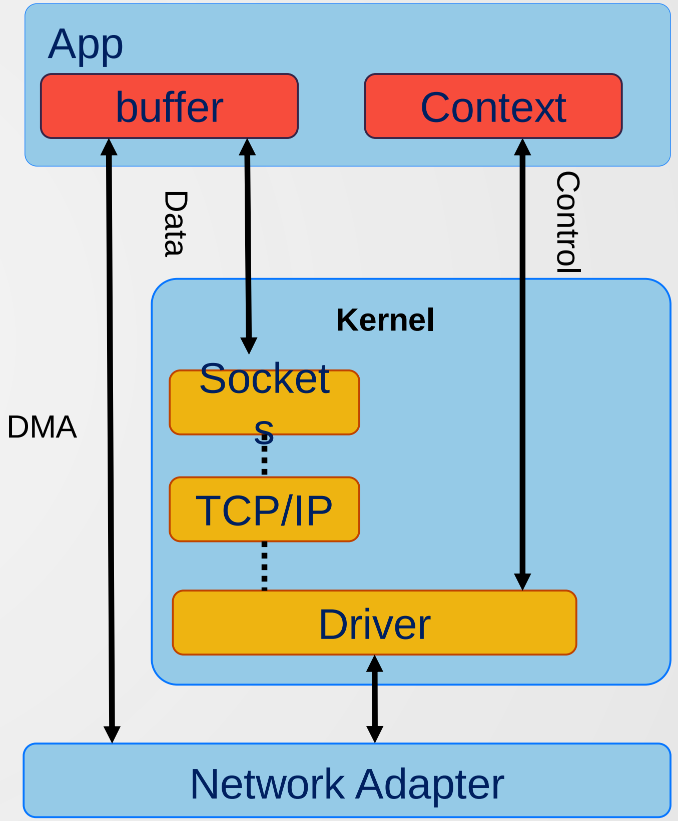
What Is RDMA?

- Stands for “Remote Direct Memory Access”
- Advanced transport protocol (same layer as TCP and UDP)
  - Modern RDMA comes from the Infiniband L4 transport specification.
  - Full hardware implementation of the transport by the HCAs.
  - Flow control and reliability is offloaded in hardware.
- Remote memory READ/WRITE semantics (one sided) in addition to SEND/RECV (2 sided)
- RoCE: RDMA over Converged Ethernet
  - The Infiniband transport over UDP encapsulation.
  - Available for all Ethernet speeds 10 – 100G
- Verbs: Low level abstract description of the functionality for RDMA programming.
  - Control path and data path verbs.
  - Several APIs – main one is libibverbs – standard Linux Verbs API.
  - Same abstraction for IB/RoCE/iWARP.
- Supported on almost all mid-range/high-end network adapters
  - Growing cloud support



RDMA Verbs basics

- Uses Kernel bypass / direct user space access
  - Supports Zero-copy – no need to copy user application buffers to a dedicated NIC buffers.
  - Application may scatter/gather to upper layer applications buffers directly.
  - Direct user space - hardware interface which bypasses the kernel and TCP/IP in data path
- Sub-microsecond latency.
- CPU utilization – CPU not involved in the DMA operations.
- High Bandwidth.



The Transport types

- Queue Pair is the actual object that transfers data
  - It encapsulates both Send and Receive Queue
  - A QP represent a real HW resource
  - Connecting QPs – manually or through rdma-cm (defined in Infiniband transport specification).

- UD – Unreliable Datagram
  - implements only two-sided communication semantics.
  - Single MTU at a time.
  - Connectionless – single QP per process (AV in send/recv)

- RC – Reliable Connected
  - Support RDMA and large messages in transport
  - Limited number of RC channels, to preserve HCA resources - Every channel is an RC QP

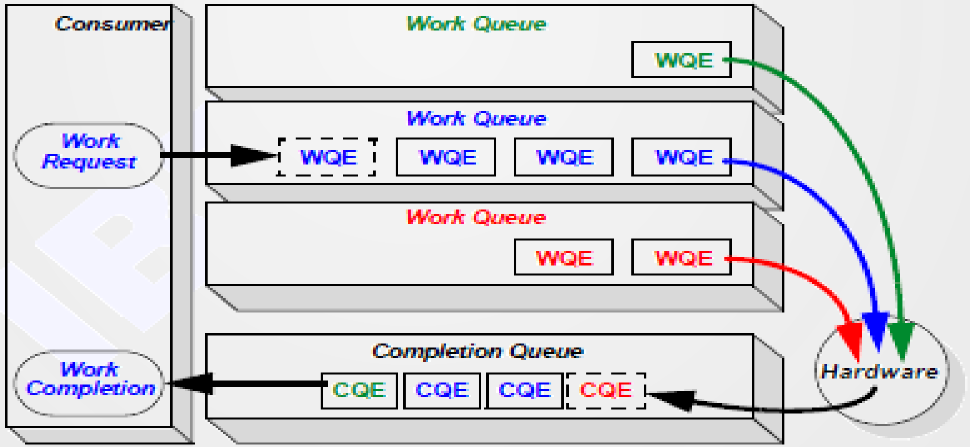
- DC - Dynamically Connected (Mellanox)
  - dynamically creates and destroys connections
  - memory consumption close to the level of
  - UD, while offering memory semantics, as RC

The IB Transport Layer

- The InfiniBand transport uses the queue pair (QP) model
  - send and a receive queue are used for issuing and receiving Messages respectively.

- A work request is submitted to these queues, where the hardware can read it to perform the communication.

- Additionally, a completion queue is associated with each queue pair for notification of communication completion.



Memory management in RDMA applications

- Communication over RDMA Verbs requires all memory regions that are accessed by the hardware to be registered (the famous ibv_reg_mr verb).
  - Pinned and mapped for the hardware as physical address by the IB core drivers.

- Memory Region is a virtually contiguous memory block that was registered, i.e. prepared for work with RDMA.
  - Any memory buffer in the process' virtual space can be registered
  - May specify which permission to allow (remote/local).

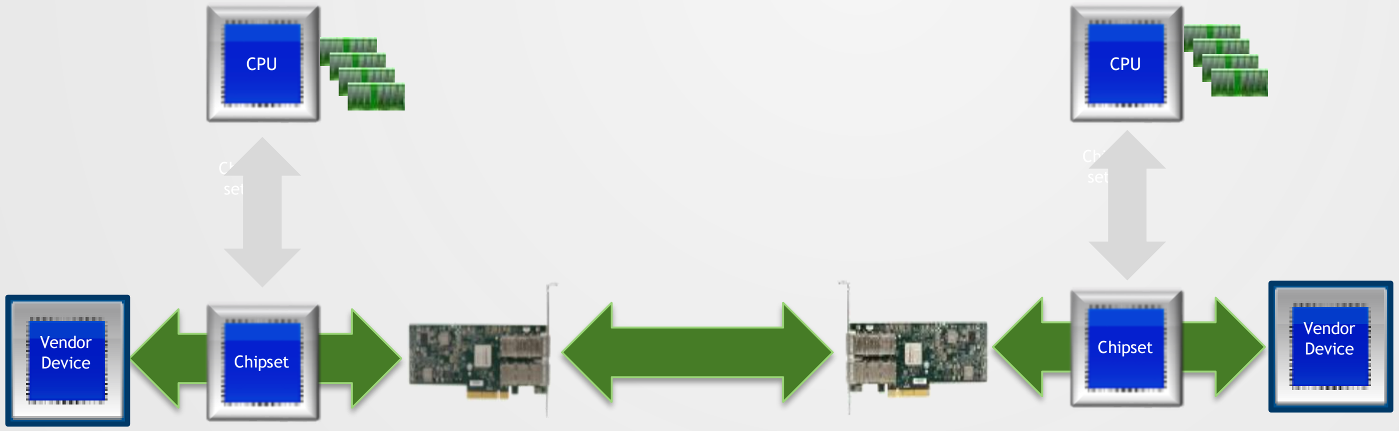
- After a successful memory registration, two keys are being generated: lkey and rkey
- In order to alleviate the overheads of memory registration, short messages can be inline in the work requests, whereas larger messages can take advantage of a zero-copy protocol.

- This strategy means that the work request gets only a description of the memory buffer and later reads the data directly from the buffer, without any CPU involvement.

- Some advanced features (Mellanox)
  - ODP – on demand paging avoidpinningthepagesofregisteredmemoryregions
  - UMR - support direct local and remote noncontiguous memory access

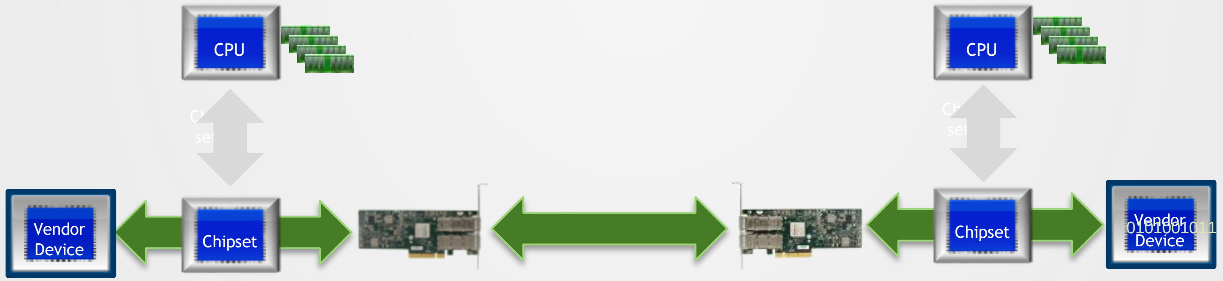
Peer to Peer communication

- Direct data transfer between PCI-e devices without the need to use main memory
  - No need for temporary storage in CPU memory.
- Also control of peers directly from other peer devices
  - Accelerate transfers between different PCI-E devices
- Improve latency, system throughput, CPU utilization, energy usage
  - Cut out the middleman



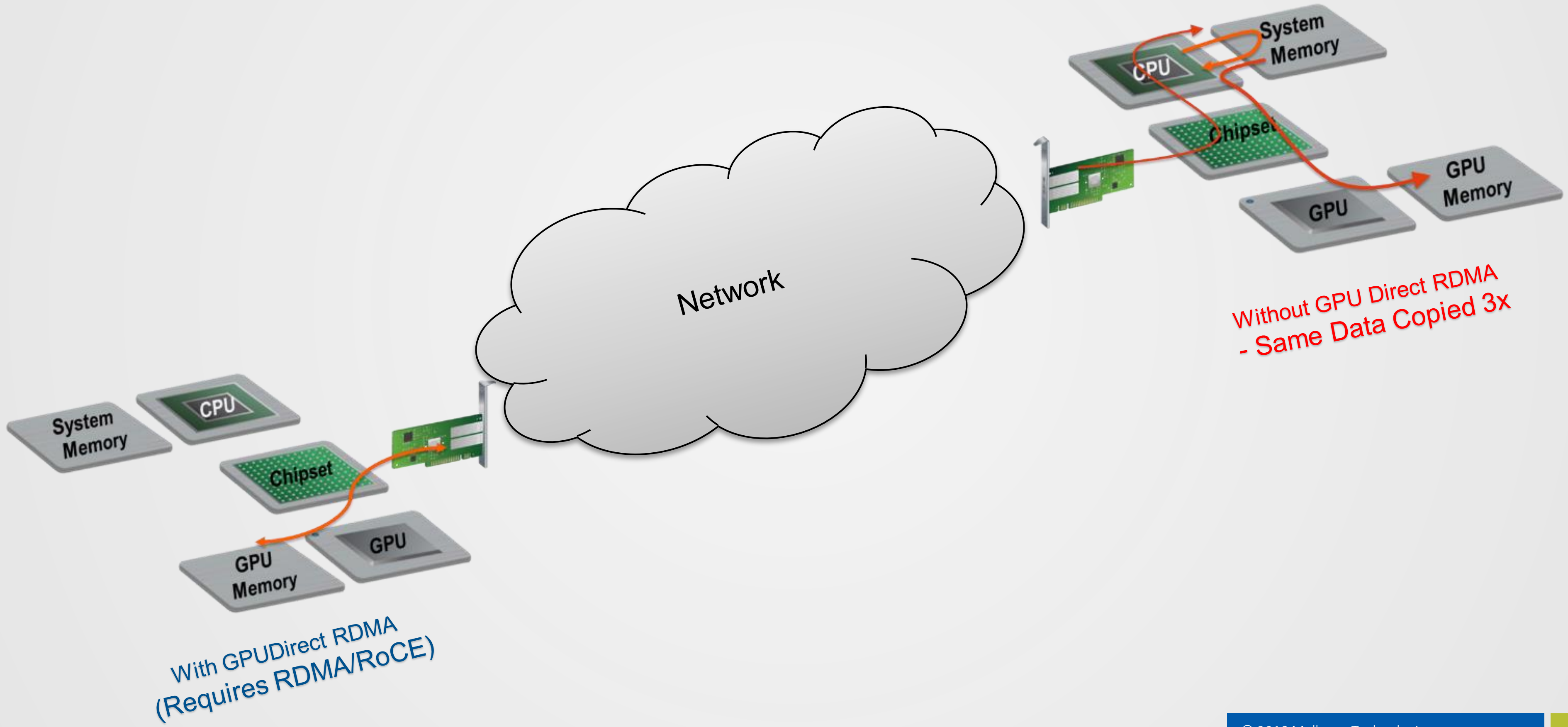
Mellanox PeerDirect™

- Natively supported by Mellanox OFED
- Supports peer-to-peer communications between Mellanox adapters and third-party devices
- No unnecessary system memory copies & CPU overhead
- Enables GPUDirect™ RDMA, GPUDirect™ ASYNC, ROCm and others
- InfiniBand and RoCE



Designed for Deep Learning Acceleration

NVIDIA GPUDirect™ RDMA



Without GPU Direct RDMA
- Same Data Copied 3x

With GPUDirect RDMA
(Requires RDMA/RoCE)

Example

```
cudaMalloc(&s_buf_d, size);
s_buf_h = malloc(size, …);
mr = ibv_reg_mr(pd, s_buf_h, size, …);
.
.
// Want to send the CUDA buffer
cudaMemcpy(s_buf_h, s_buf_d, size..);
wr.sg_list->addr = s_buf_h;
wr.sg_list->lkey = mr.lkey;
wr.sg_list->length = size;
ibv_post_send(qp, &wr, …);
```

```
cudaMalloc(&s_buf_d, size);
mr = ibv_reg_mr(s_buf_d, size, …);
.
.
.
wr.sg_list->addr = s_buf_d;
wr.sg_list->lkey = mr.lkey;
wr.sg_list->length = size;
ibv_post_send(qp, &wr, …)
```

How does it work ?

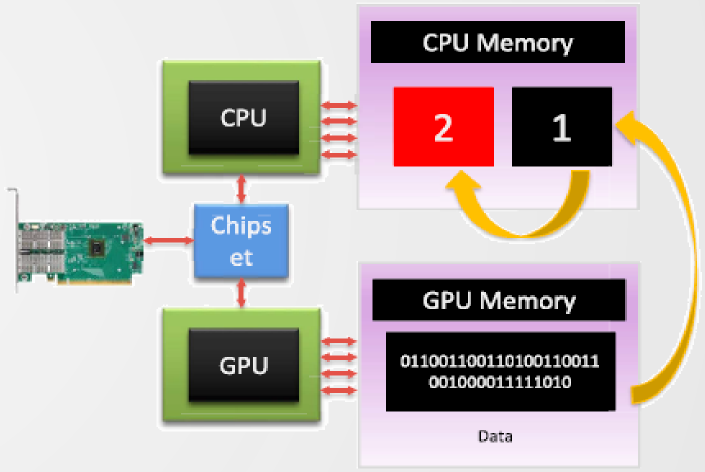
- Allow ibv_reg_mr() to register peer memory.
  - Peer devices implement new kernel module – io_peer_mem
  - Register with RDMA subsystem - ib_register_peer_memory_client()

- io_peer_mem implements the following callbacks :
  - acquire() – detects whether a virtual memory range belongs to the peer
  - get_pages() – asks the peer for the physical memory addresses matching the memory region
  - dma_map() – requests the bus addresses for the memory region

- Matching callbacks for release: dma_unmap(), put_pages() and release()

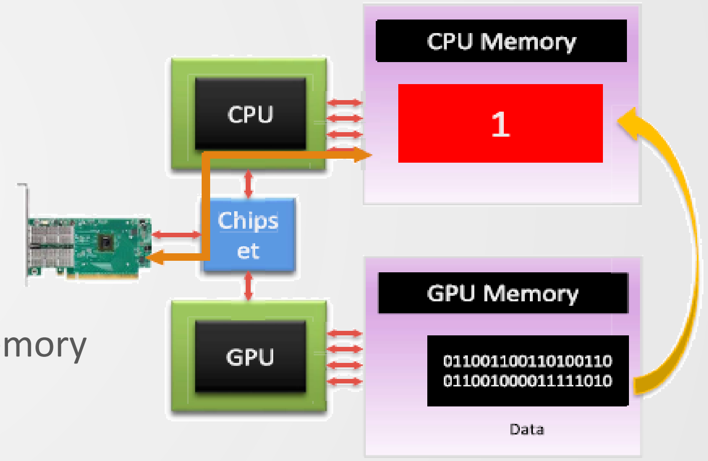
Before GPUDirect™

- GPUs use driver-allocated pinned memory buffers for transfers
- RDMA use user pinned buffers for communication
- It was impossible for RDMA drivers to pin memory allocated by the GPU
- Userspace needed to copy data between the GPU driver's system memory region and the RDMA memory region



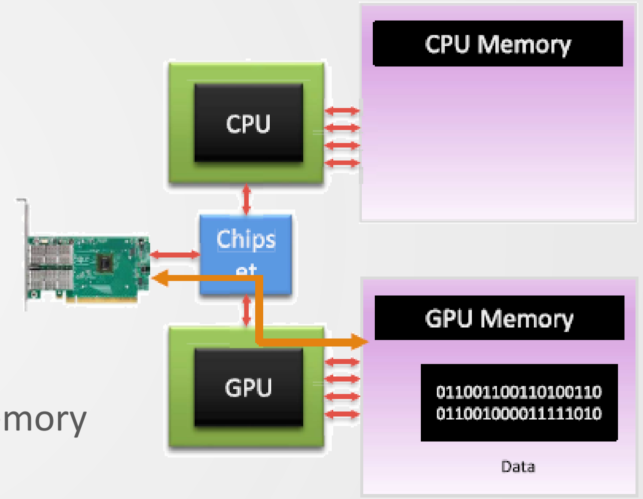
GPUDirect™ / GPUDirect™ P2P

- GPU and RDMA device share the same “pinned” buffers
- GPU copies the data to system memory
  RDMA device sends it from there
- Eliminate the need to make a redundant copy in CUDA host memory



GPUDirect™ RDMA / PeerDirect™

- CPU synchronizes between GPU tasks and data transfer
- HCA directly accesses GPU memory
- Direct path for data exchange
- Eliminate the need to make a redundant copy in host memory

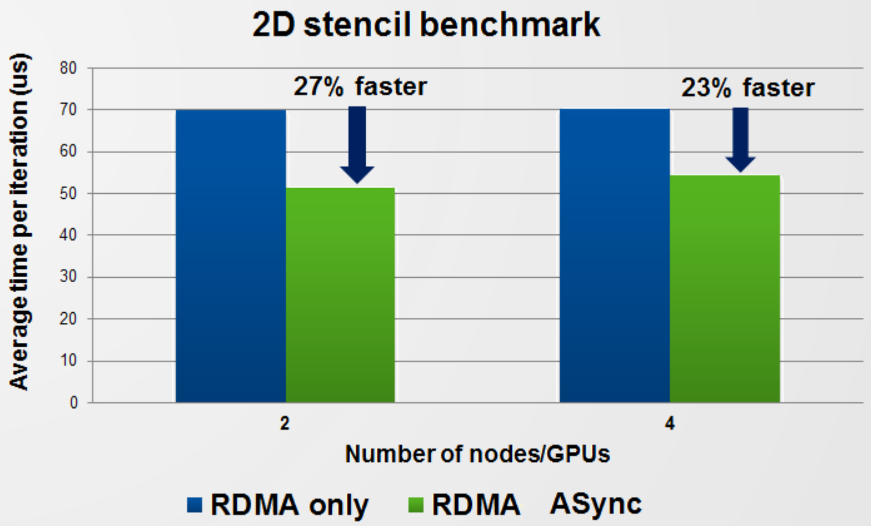


GPUDirect™ ASYNC

- GPUDirect RDMA (3.0) – direct data path between the GPU and Mellanox interconnect
  - Control path still uses the CPU
    - CPU prepares and queues communication tasks on GPU
    - GPU triggers communication on HCA
    - Mellanox HCA directly accesses GPU memory

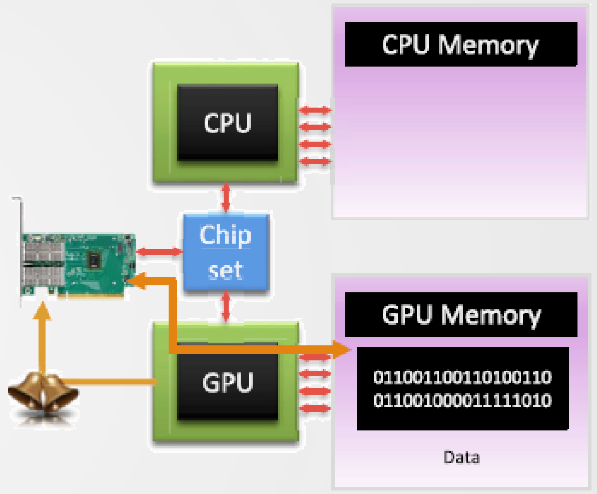
- GPUDirect ASYNC (GPUDirect 4.0)
  - Both data path and control path go directly between the GPU and the Mellanox interconnect

Maximum Performance
For GPU Clusters



GPUDirect™ Async / PeerDirect™ Async

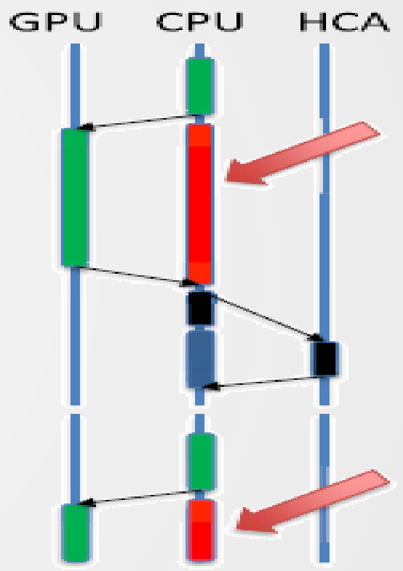
- Control the HCA from the GPU
  - Enable batching of multiple GPU and communication tasks
- • Reduce latency
- • Reduce CPU utilization
- • Light weight CPU
- • Less power



- CPU prepares and queues compute and communication tasks on GPU
- GPU triggers communication on HCA
- HCA directly accesses GPU memory

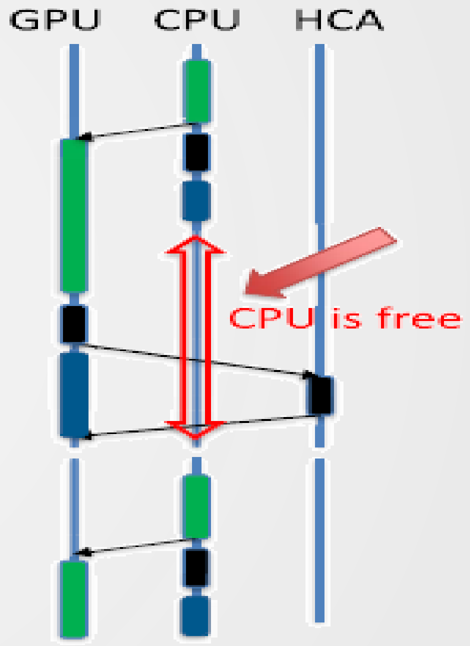
Code example GPUDirect™ RDMA

```
while(fin) {
    gpu_kernel <<<… , stream>>>(buf);
     cudaStreamSynchronize(stream);
     ibv_post_send(buf);
     ibv_poll_cq(cqe);
}
```



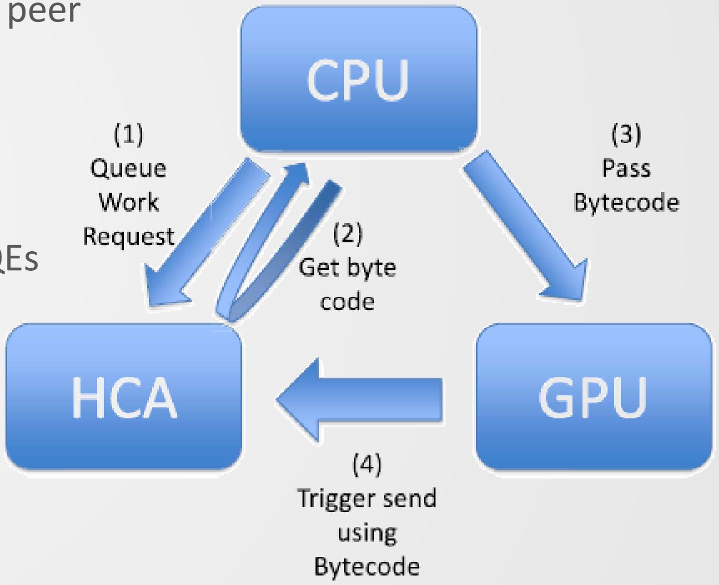
Code example GPUDirect™ ASYNC

```
while(fin) {
  gpu_kernel <<<… , stream>>>(buf);
  gds_stream_queue_send(stream, qp, buf);
  gds_stream_wait_cq(stream, cqe);
}
```



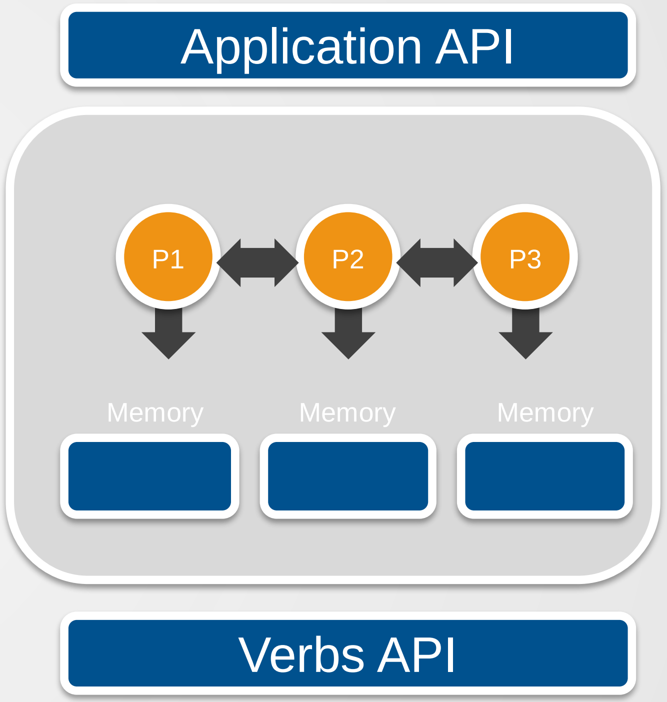
How does it work ?

- Create a QP ->Mark it for PeerDirect Async -> Associate it with the peer
- Post work requests using ibv_post_send()
  - Doorbell is not ringed
- Use ibv_peer_commit_qp() to get bytecode for committing all WQEs
  - currently posted to the send work queue
- Queue the translated bytecode operation on the peer
- Peer executes the operations after generating outgoing data



So now what ?

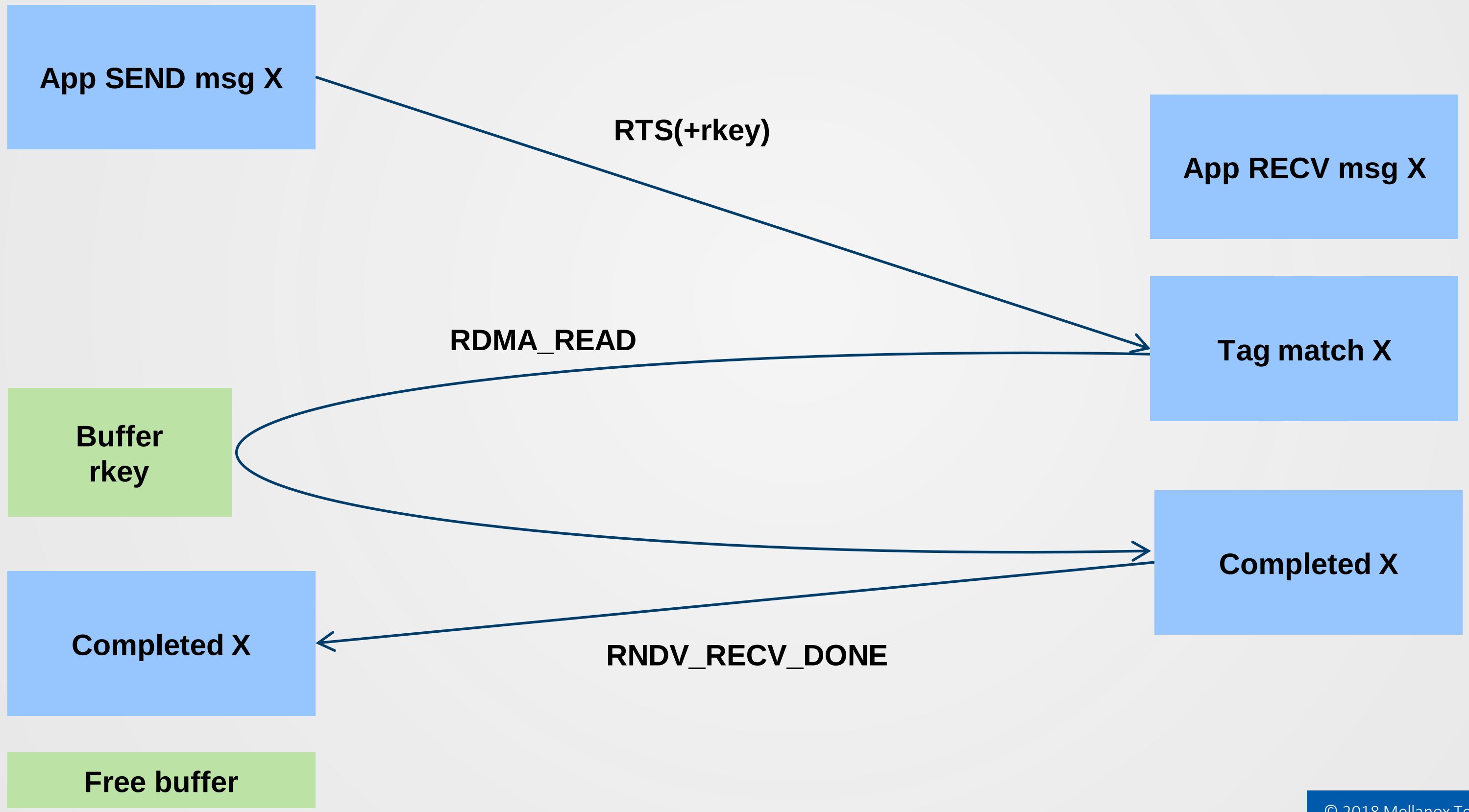
- So RDMA is really a P2P memory semantics programing.
  - Also the 2 sided SEND/RECV must be in order.
  - Application needs to know what/when came from the RDMA and use it when it wants.
- Applications (sometimes – depends on its nature) need message passing semantics protocol on top of the RDMA.
- So application usually need to implement messaging protocol over the RDMA.
  - So operations are based on identifiers of the application (like in MPI tags)
  - Sender specifies which message to send when ready.
  - Receiver specifies which message it expect to receive in a given moment.



Point to Point message protocols (MPI)

- Eager
  - Sender sends the tag+data to receiver
  - Send() is completed immediately*
    - (unless sender is using zero-copy)
  - Receiver must store the data until it’s matched
  - May exhaust memory on the receiver
  - Typically used for small-medium messages
  - Usually means no GPUDirect RDMA
- Rendezvous
  - Sender sends only the tag to the receiver
  - Send() is not completed
  - Receiver requests the data from the sender when it matches the tag
  - Data can be stored directly to the receiver buffer, since it’s already matched
    - use RDMA for zero-copy receive
  - Typically used for large messages
- Configurable threshold for protocol switch

Rendezvous example



Protocol threshold calculation: rendezvous vs. eager (From MXM)

- Estimate performance of each protocol
  - Based on: wire speed, latency overhead, registration cost, …
- Find the point where one protocol is better than another
- The intersection point is the threshold

For InfiniBand EDR:

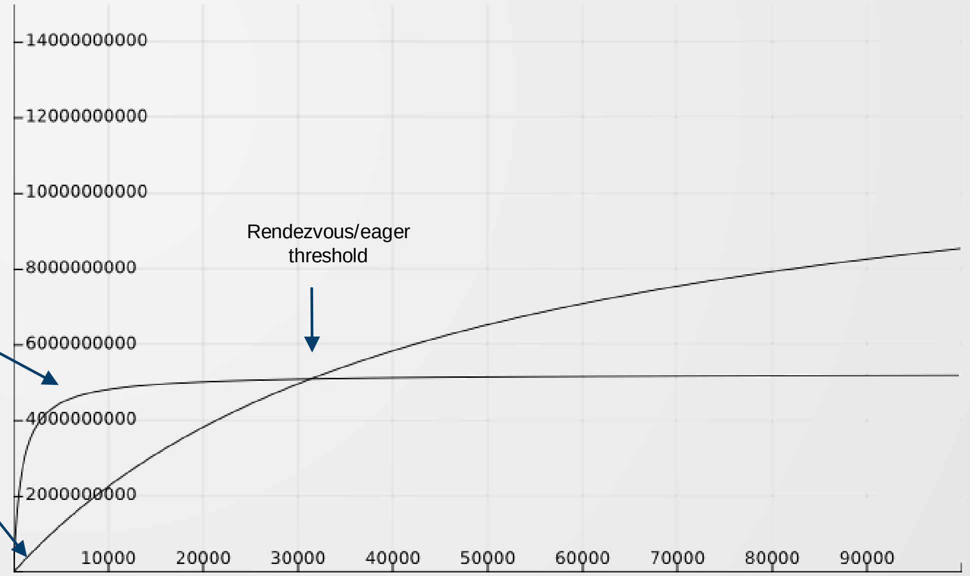
- eager zcopy bandwidth (x=message size):

$$\frac{x}{\underbrace{90\times10^{-9}}_{\text{memreg cost}} + \underbrace{\frac{x}{5000\times2^{20}}}_{\text{memcpy bw}} + \underbrace{75\times10^{-9}}_{\text{send overhead}}}$$

- Rendezvous RDMA bandwidth:

$$\frac{x}{\underbrace{2\times90\times10^{-9}}_{\text{memreg cost}} + \underbrace{4\times800\times10^{-9}}_{\text{fabric latency}} + \underbrace{3\times75\times10^{-9}}_{\text{send overhead}} + \underbrace{\frac{x}{11794.23\times2^{20}}}_{\text{wire speed}}}$$

- Threshold: ~30k



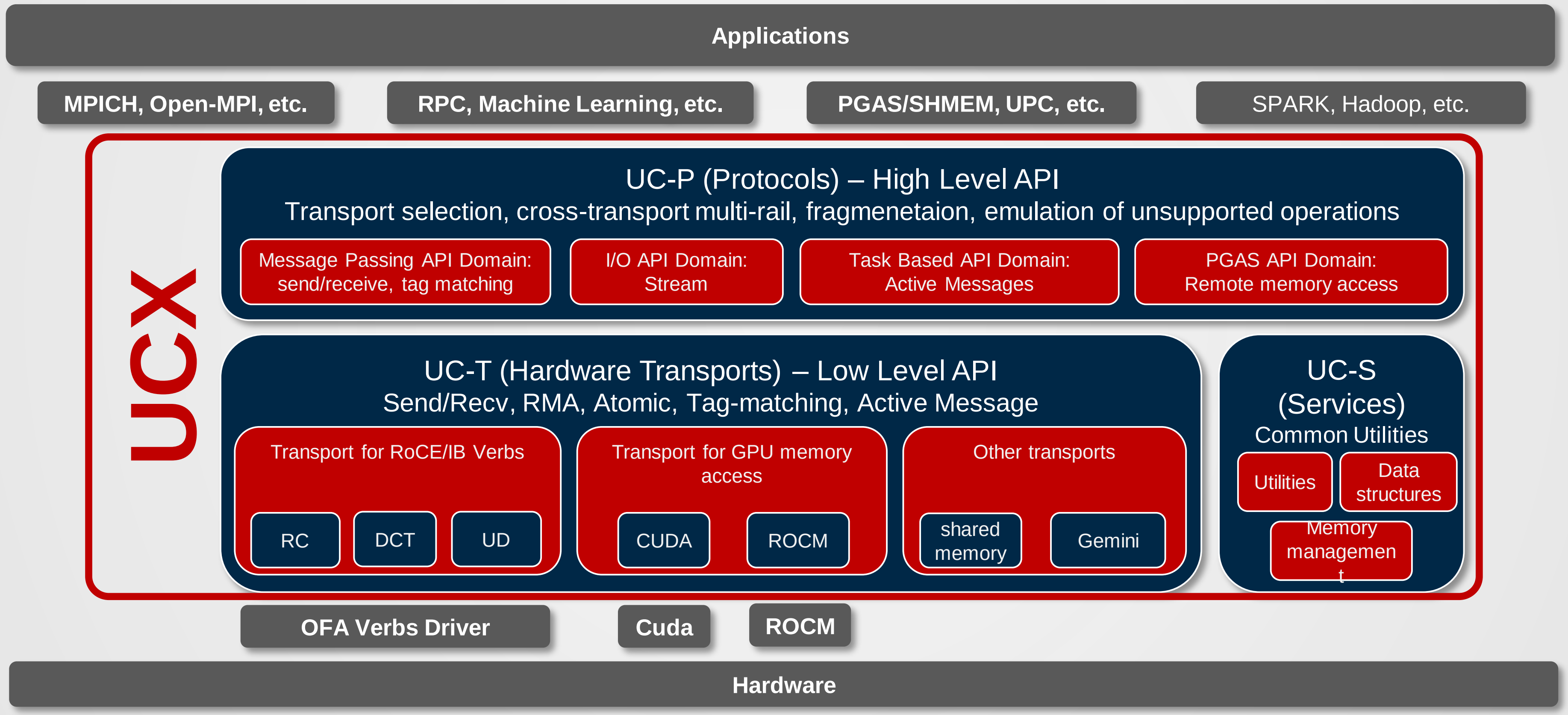
Common pitfalls

- Rendezvous and Zero-copy are different things
  - The relation between them is that zero-copy on *receiver* side can be done only with rendezvous (or hardware assisted tag matching)
- Scale
  - Which transport to use ?
  - Maybe hold set of multiple transport resources per use case ?
- Zero copy can be slower
  - Zero copy improves CPU usage, not bandwidth*
  - Whether zero copy is useful or not depends on the application.
- GPU Direct RDMA edge cases
  - Not in all cases GPUDirect RDMA zero copy is better
  - Depends on the topology of the peers (GPU and HCA), can also depends on the message size.
- Memory
  - All buffers must be registered, how many buffers to use.
  - Bounce buffers may be needed.

UCX – Unified Communication X

- So maybe we need all this expertise in one place ?
- Collaboration between industry, laboratories, and academia
- Open-source production grade communication framework for HPC and data-driven applications
- Goal is to enable the highest performance through co-design of software-hardware interfaces
- Simple and consistent API
- Protocols and transports are selected by capabilities and performance estimations, rather than hard-coded definitions.

UCX stack



Applications
MPICH, Open-MPI, etc.
RPC, Machine Learning, etc.
PGAS/SHMEM, UPC, etc.
SPARK, Hadoop, etc.
UCX
UC-P (Protocols) – High Level API
Transport selection, cross-transport multi-rail, fragmenetaion, emulation of unsupported operations
Message Passing API Domain: send/receive, tag matching
I/O API Domain: Stream
Task Based API Domain: Active Messages
PGAS API Domain: Remote memory access
UC-T (Hardware Transports) – Low Level API
Send/Recv, RMA, Atomic, Tag-matching, Active Message
Transport for RoCE/IB Verbs
RC
DCT
UD
Transport for GPU memory access
CUDA
ROCM
Other transports
shared memory
Gemini
UC-S (Services)
Common Utilities
Utilities
Data structures
Memory managemen t
OFA Verbs Driver
Cuda
ROCM
Hardware

UCP API – send/receive

- Non-blocking send:

```
ucs_status_ptr_t ucp_tag_send_nb(ucp_ep_h ep, const void
*buffer, size_t count, callback cb,
                                              ...)
```

- Non-blocking receive:

```
ucs_status_ptr_t ucp_tag_recv_nb(ucp_worker_h worker,
void *buffer, size_t count, callback cb
                                              ...);
```

Collectives

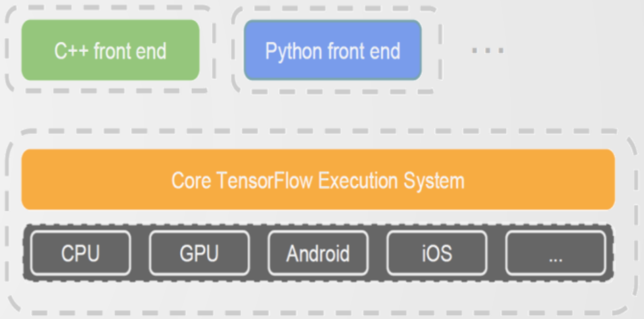
- So RDMA only support a message passing from one process to another. In this sense, these commands are called point-to-point communication commands.
- In many other occasions, we want to let one process to send data to all the other processes, or gather data from all the processes to one process.
  These operations are called collective communications.
- In theory, collective communication can be achieved by a combination of point-to-point communications, but using collective communications make the code simpler and maybe faster.
- There are numerous algorithms to perform every collective operations (active academic research)
  - Optimized for different scales / message sizes / networks
- Examples: Recursive doubling, Pairwise exchange, Fain-in, fan-out, Reduce-scatter/Allgather, Linear, Bruck, etc..
- Mellanox HCOLL - hierarchical collectives, based on point-to-point, multicast, core-direct.

All Major Machine Learning Frameworks Support RDMA

- TensorFlow
  - Native RDMA support distributed TensorFlow:
    - “verbs” – Donated primarily by Yahoo
    - “gdr” – with GPUDirect RDMA - Donated Bairen Yi (HKUST).
  - Via OpenMPI and NCCL2 for Horovod distribution.
- Caffe2: Support with GLOO
  - Supports peer to peer and collectives with native RDMA
- Microsoft Cognitive Toolkit: Native support of MPI (OpenMPI)
- Paddle Paddle: Native support
- NVIDIA NCCL: Native and NCCL2
- Spark – RDMA support (add spark logo)

TensorFlow

- TensorFlow is an open source software library for numerical computation using data flow graphs
- Second generation deep learning system from Google. (based on DistBeleif framework)
- Front-end in python/C++ for model definition.
- Runtime backend in C++, linear algebra packages and GPU support.
- Model design transparent to infrastructure/scale:
  - Abstracts away the underlying hardware.
  - Same program for different infrastructure.
- Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them.
- Graph is all Operations and Tensors (and maybe also devices)



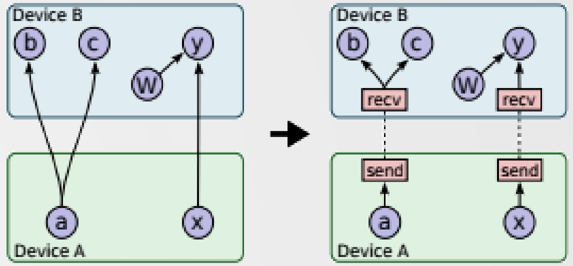
Distributed TensorFlow

- Tensor flow has built in support for distributed execution.

- Sessions - Actual runtime lifting of a graph.
  - Translate graph definition to executable operation distributed across devices (compute resources). Devices – CPU/GPUs.
  - Nodes/operations placement on devices.
  - May add SEND/RECEIVE nodes – where a Tensor traverse two different devices.

- Running a TensorFlow cluster:
  - Running TensorFlow server program, with one or more worker, on each node.
  - Use the devices scatters around the cluster in the TensorFlow graph.
  - TensorFlow server class can be implemented by others to add extra support
    - Like in the case of the RDMA implementations.
    - All the RDMA plugins use the default GRPC layer by adding the RDMA backbone and only implement the RendezvousMgr interface.

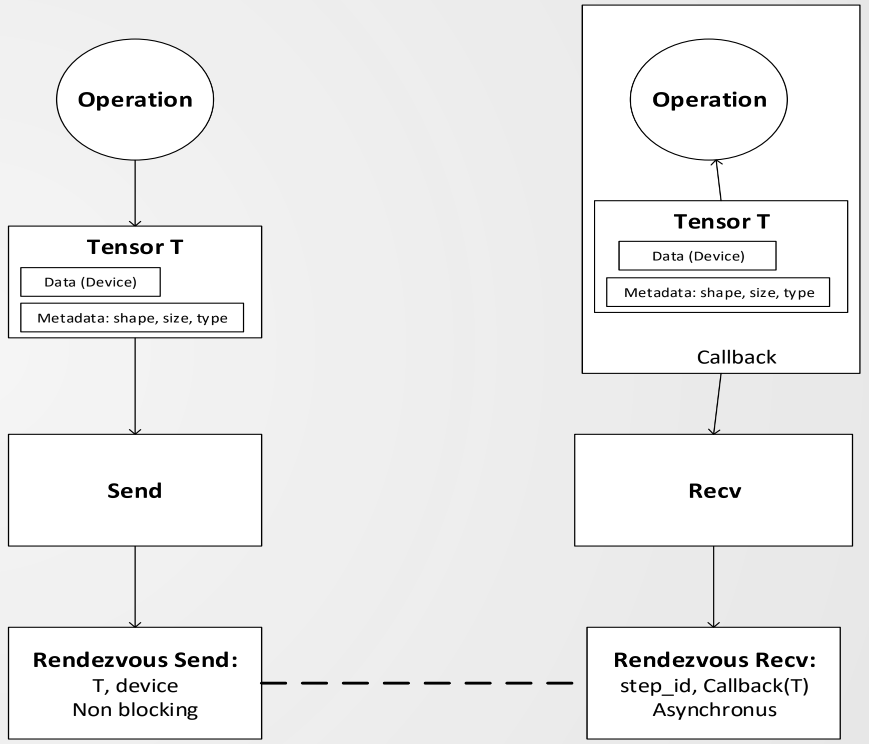
- The SEND/RECV nodes communicate with each other by an abstract *Rendezvous* interface, supplied to the SEND/RECV kernels in their construction phase by the session.



The TensorFlow Rendezvous Interface

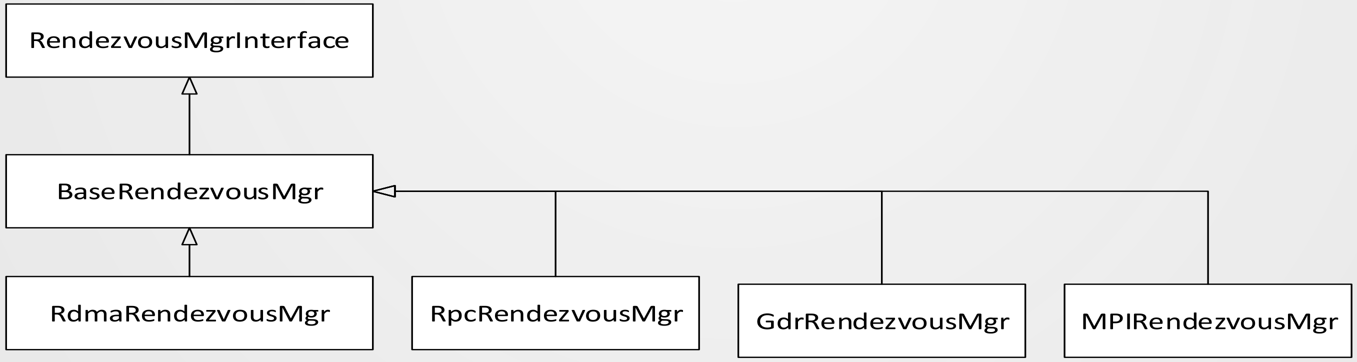
- A non-blocking Send operation, which receive the Tensor it needs to send and its device information (which device holds this tensor, memory allocator for this tensor memory).

- An asynchronous Receive operation which is supplied with a callback function (to be called when the Tensor is ready) and device information of where to place this Tensor when calling the callback.


Operation
Tensor T
Data (Device)
Metadata: shape, size, type
Send
Rendezvous Send: T, device Non blocking
Operation
Tensor T
Data (Device)
Metadata: shape, size, type
Callback
Recv
Rendezvous Recv: step_id, Callback(T) Asynchronus

The RendezvousMgr Interface

- TensorFlow allows for external implementations of RendezvousMgr to be “plugged” in.
- The SEND/RECV nodes use the above functionally of the abstract Rendezvous interface, without really knowing which Rendezvous implementation is being used (Implementation is chosen by the session).
- Interface allows proprietary implementations of Rendezvous SEND/RECV and essentially defers the entire Tensor passing mechanism (default is GRPC).

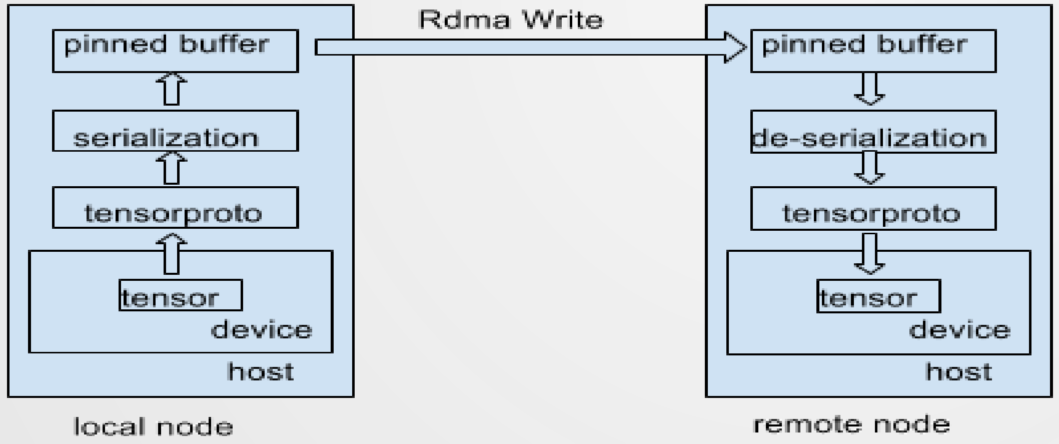


First verbs implementation example

- First implementation was not using zero copy and used eager mode.
  - Motivation was to save the memory registration operations.

- GPUDirect RDMA couldn't have work since it worked only with pinned CPU buffer.

- That means there is always 1 copy for CPU tensors, and 2 copies for GPU tensors.



TensorFlow GPUDirect RDMA

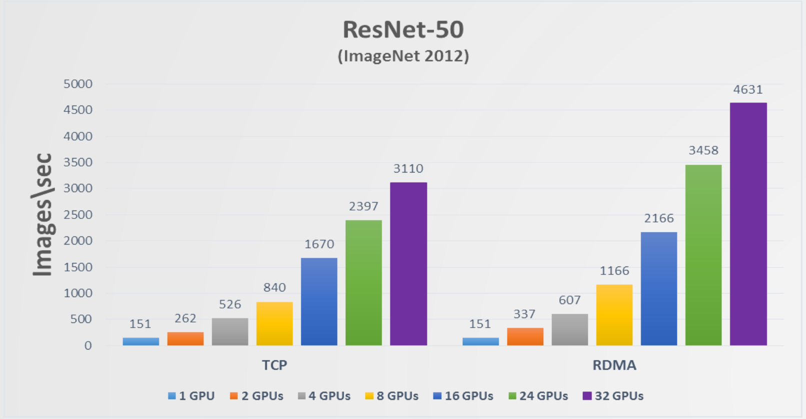
- In order to apply GPUDirect RDMA 0 copies approach:
  - Reduce the CPU tensors copies to 0 (always)
  - GPU tensors copies to 0 for RDMA compatible GPUs, and 1 for other GPUs.

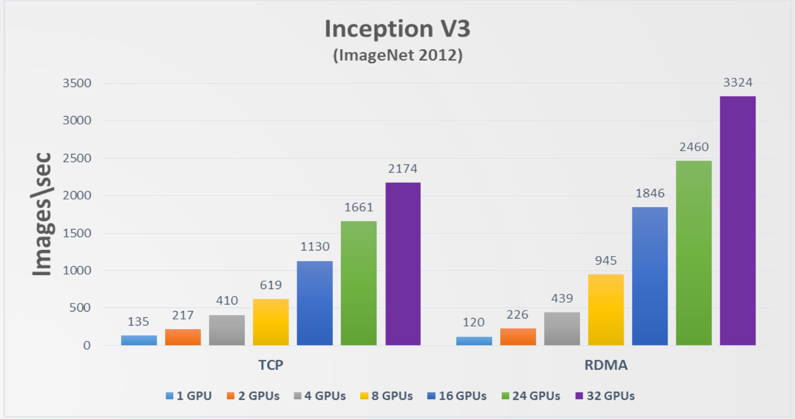
- No longer allocate a fixed CPU buffer per tensor, since we want the write to be done directly to the result Tensor’s buffer.

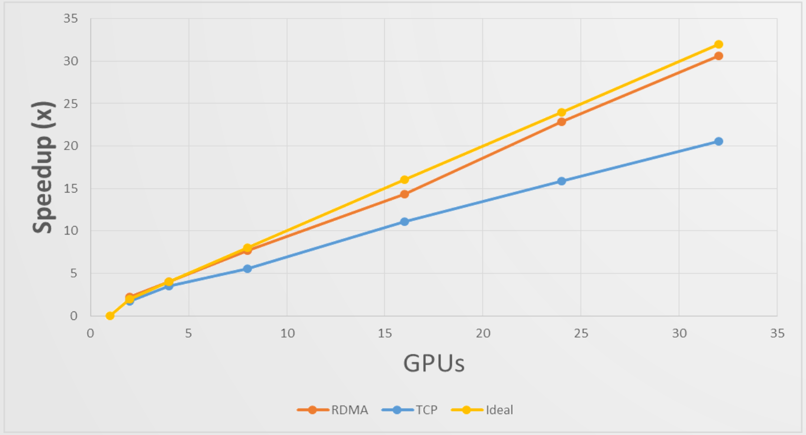
- The result Tensor is allocated on the receiver side **BEFORE** it sends the request. This way the remote address and rkey can be delivered inside the request itself

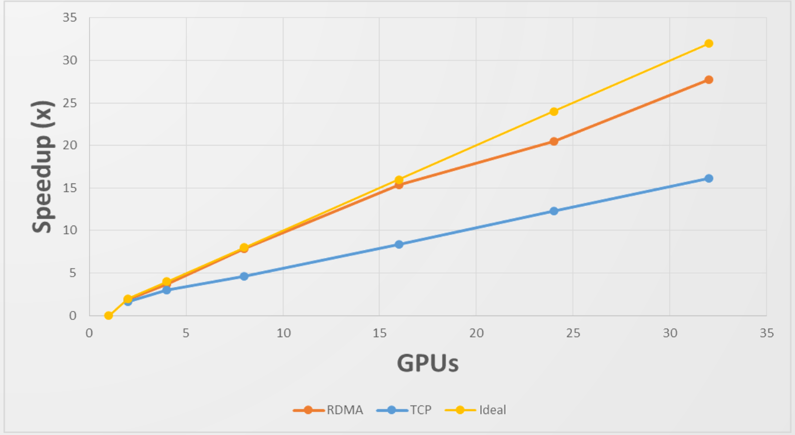
- GDR’s (Bairen Wi) allocation theme:
  - the result Tensor is allocated on an already registered memory region, whether it is a CPU tensor or a RDMA compatible GPU Tensor.
  - The Sent Tensor was already allocated at the moment of the SEND operation.
  - So we can inherit and implement a new memory allocator which will do the registration automatically On new GPU allocations, which usually are being allocated in big chunks (good for performance).

Example Results – Distributed TensorFlow



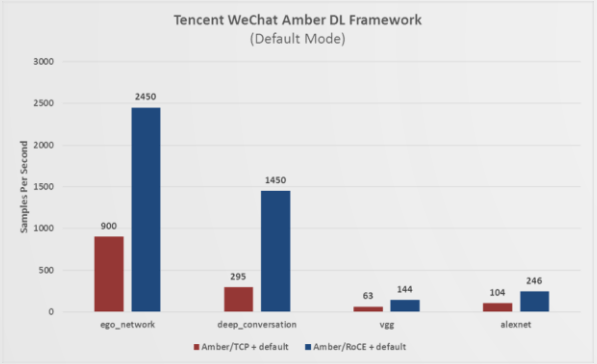


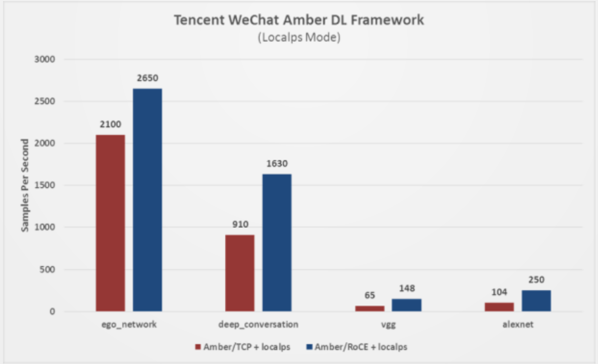


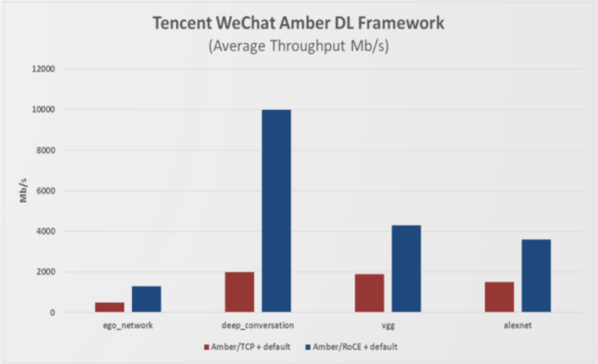


Tencent Amber Deep Learning Framework

- Provides **2X** bandwidth than TCP in VGG
- Delivers **2.5X** performance than TCP in localps mode
- Delivers **5X** performance than TCP in default mode
- Provides **5X** bandwidth than TCP in ego_network
- Linear scale up for CNN, VGG16 and AlexNet models





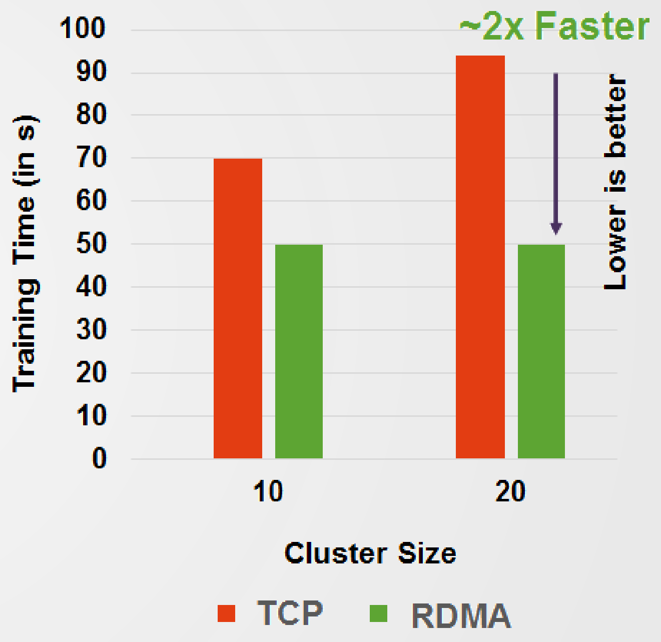


Mellanox RoCE Provides 2.5X Better Performance over TCP
Delivers 5X Better Network Bandwidth and Linear Scalability

2X Acceleration for Baidu

- Machine Learning Software from Baidu
  - Usage: word prediction, translation, image processing
- RDMA (GPUDirect) speeds training
  - Lowers latency, increases throughput
  - More cores for training
  - Even better results with optimized RDMA

PaddlePaddle

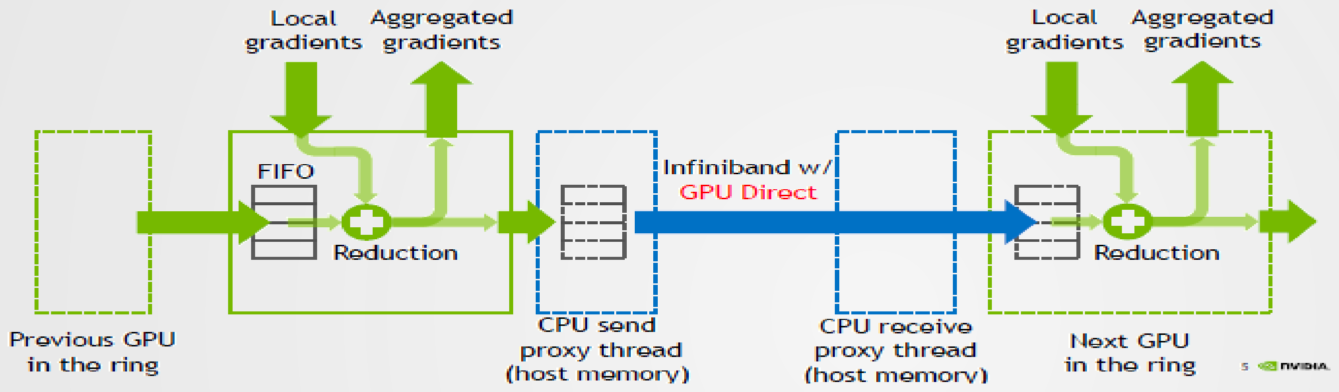


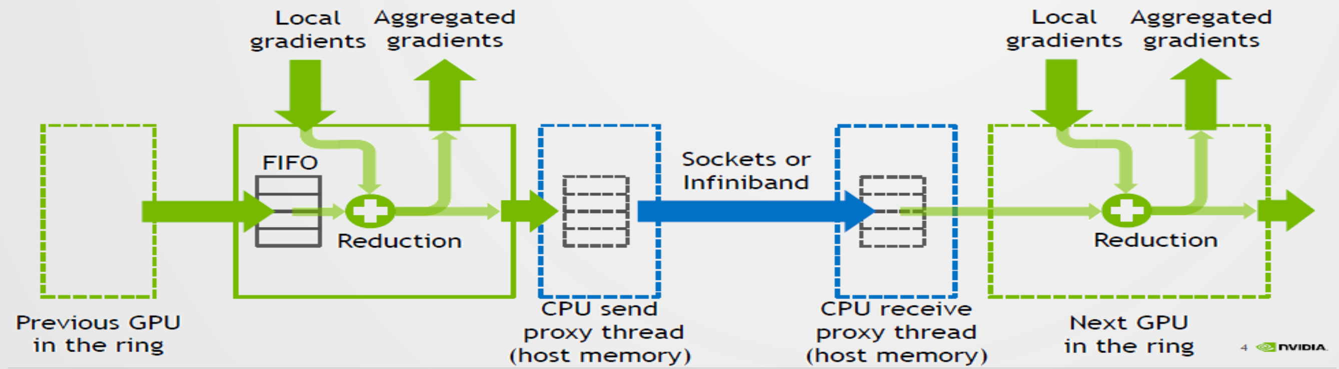
~2X Acceleration for Paddle Training with RDMA

NCCL2.0 with RDMA

- Optimized collective communication library between CUDA devices.
- Easy to integrate into frameworks, as well as for traditional HPC (MPI)
- Runs on the GPUs using asynchronous CUDA kernels.
- Operates on CUDA pointers, operations tied to CUDA stream.
- Support Point-to-Point communications and collectives inter-node communications.
  - Inter-node communication using Sockets or Infiniband verbs, with multi rail support, topology detection and automatic use of GPUDirect RDMA.
- Optimal combination of NVLink,PCIe and network interfaces to maximize bandwidth and create rings across node.

NCCL2 GPUDirect RDMA example

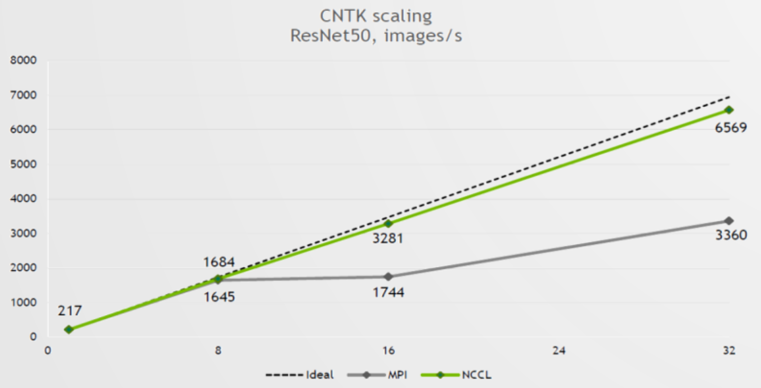


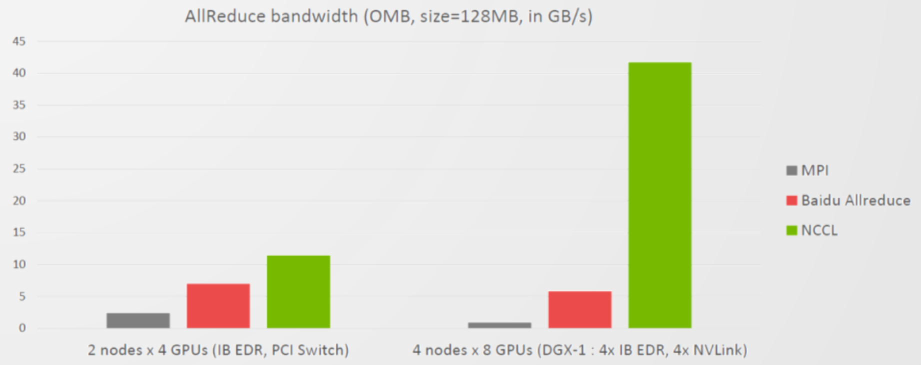


Mellanox Accelerates NVIDIA NCCL 2.0

50% Performance Improvement

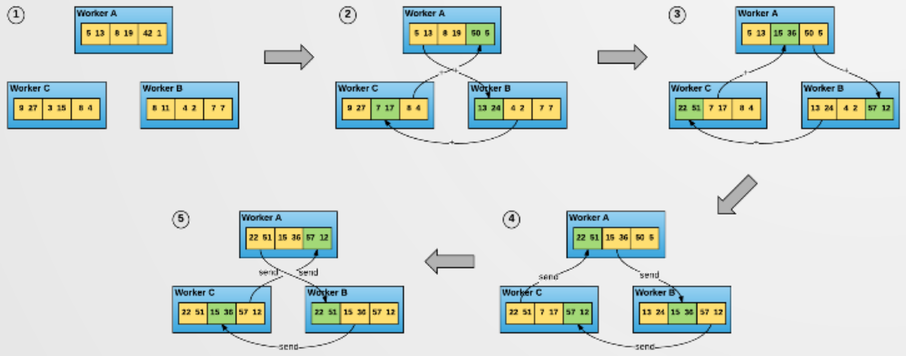
with NVIDIA® DGX-1 across 32 NVIDIA Tesla V100 GPUs Using InfiniBand RDMA and GPUDirect™ RDMA

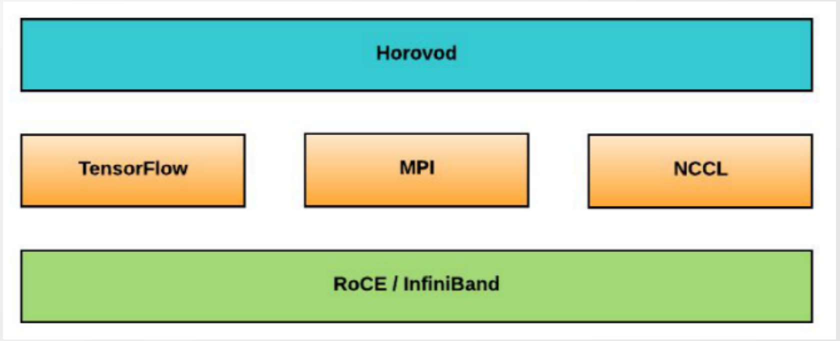




Horovod

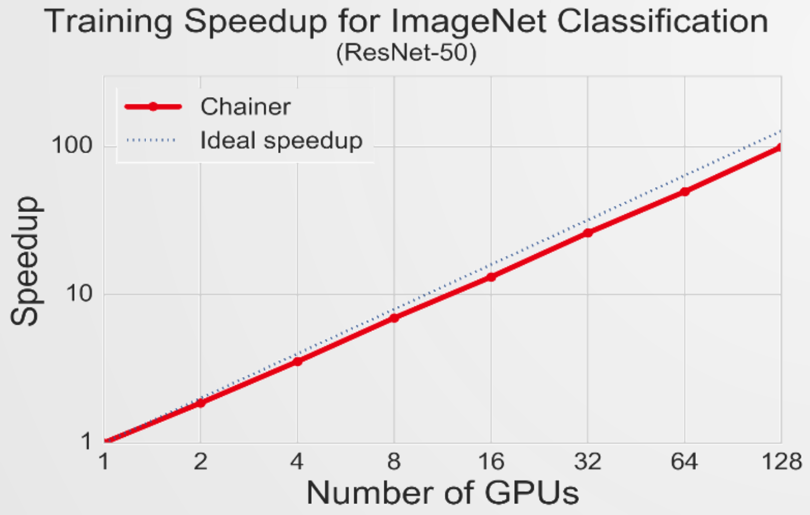
- Distributed training framework for TensorFlow
- Inspired by work of Baidu, Facebook, et al.
- Uses bandwidth-optimal communication protocols
  - Makes use of RDMA (Both RoCE or InfiniBand)
- Seamlessly installs on top of TensorFlow via pip install horovod
- Named after traditional Russian folk dance where participants dance in a circle with linked hands

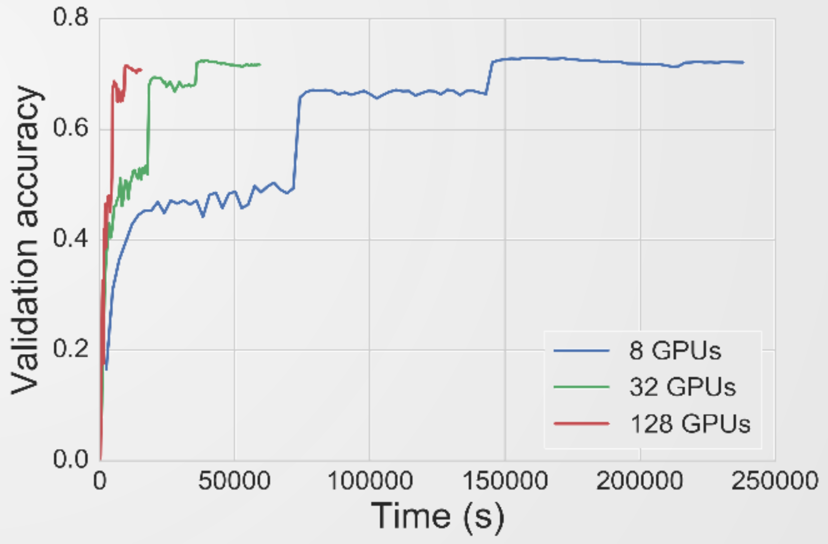




ChainerMN Depends on InfiniBand

- ChainerMN depends on MPI for inter-node communication
- NVIDIA® NCCL library is then used for intra-node communication between GPUs
- Leveraging InfiniBand results in near linear performance
- Mellanox InfiniBand allows ChainerMN to achieve ~72% accuracy.





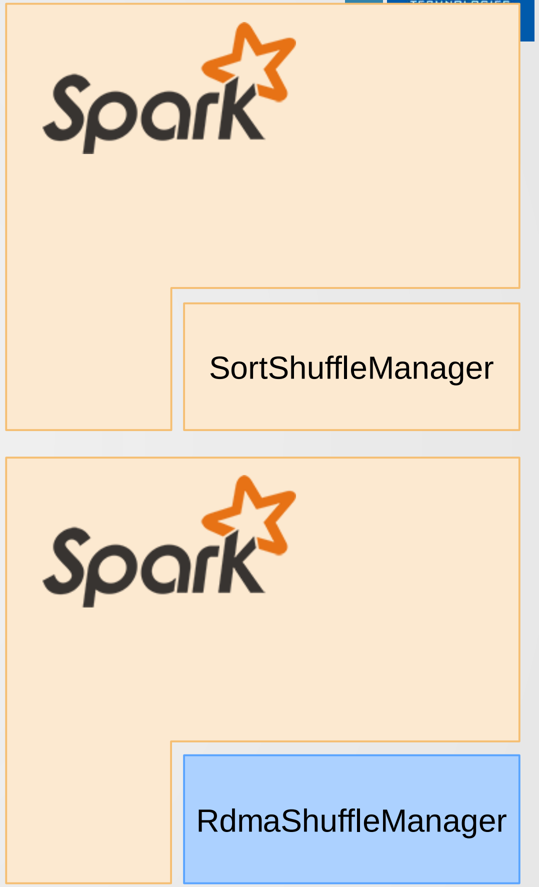
Source: http://chainer.org/general/2017/02/08/Performance-of-Distributed-Deep-Learning-Using-ChainerMN.html

ShuffleManager Plugin

- Spark allows for external implementations of ShuffleManagers to be plugged in
  - Configurable per-job using: “spark.shuffle.manager”

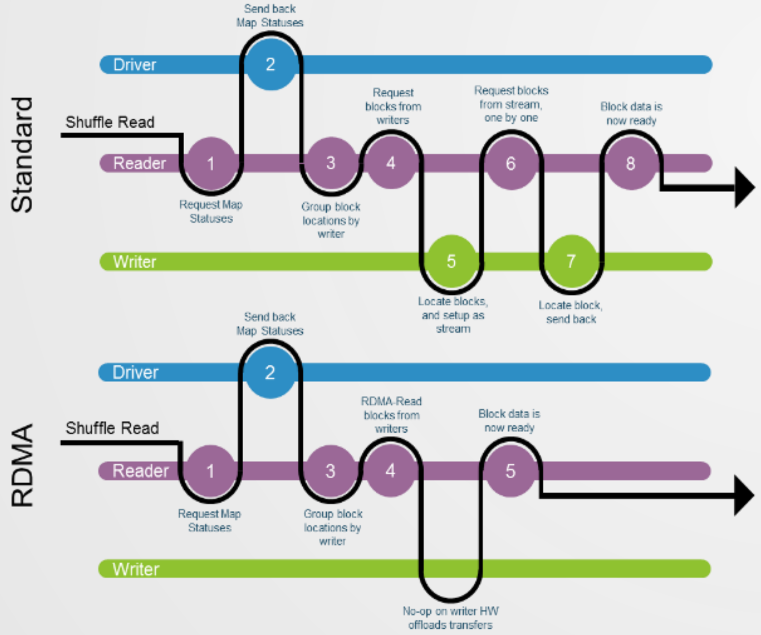
- Interface allows proprietary implementations of Shuffle Writers and Readers, and essentially defers the entire Shuffle process to the new component

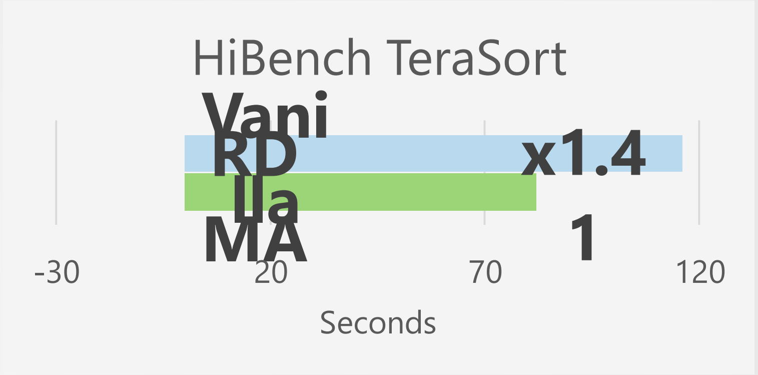
- SparkRDMA utilizes this interface to introduce RDMA in the Shuffle process

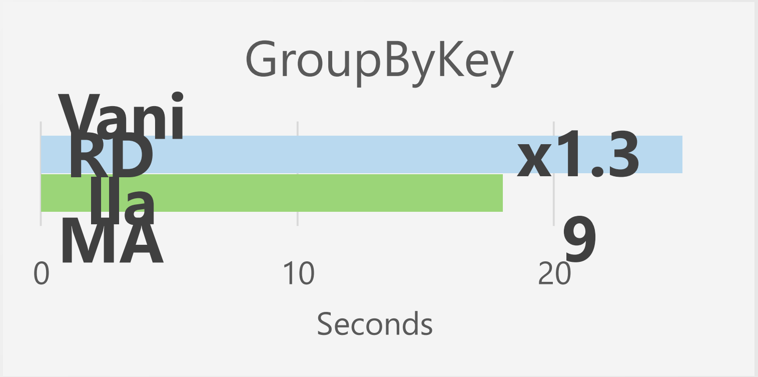


SparkRDMA – Accelerate Apache Spark by 40%

- Shuffle Manager plugin for Apache Spark
- RDMA acceleration for Spark Shuffle transfers
- Open-source on GitHub: https://github.com/Mellanox/SparkRDMA
- Easy and seamless deployment, on a per-job basis


Standard
Shuffle Read
Driver
Reader
Writer
Send back Map Statuses
Request Map Statuses
Group block locations by writer
Request blocks from writers
Locate blocks, and setup as stream
Request blocks from stream, one by one
Locate block, send back
Block data is now ready
RDMA
Shuffle Read
Driver
Reader
Writer
Send back Map Statuses
Request Map Statuses
Group block locations by writer
RDMA-Read blocks from writers
No-op on writer HW offloads transfers
Block data is now ready





HDFS+RDMA

- All-new implementation of RDMA acceleration for HDFS
  - Implements a new DataNode and DFSClient
  - Data transfers are done in zero-copy, with RDMA

- Lower CPU, lower latency, higher throughput
- Efficient memory utilization

- Initial support:
  - Hadoop: HDFS 2.6
  - Cloudera: CDH 5.10

- Future:
  - Erasure coding offloads on HDFS 3.X
  - NVMeF

- 1.0 GA is scheduled for end of Q1 2018
  - WRITE operations over RDMA
  - READ operations still carried over TCP in this version

RDMA Proven Advantages

- RDMA delivers performance advantage over traditional TCP
- Machine Learning and HPC platforms share the same interconnect needs
- Scalable, flexible, high performance, high bandwidth, end-to-end connectivity
- Standards-based and supported by the largest eco-system
- Supports all compute architectures: x86, Power, ARM.
- Native Offloading architecture
- GPUDirect RDMA accelerations
- Backward and future compatible

Thank You