Data Processing on Modern Hardware

Jana Giceva

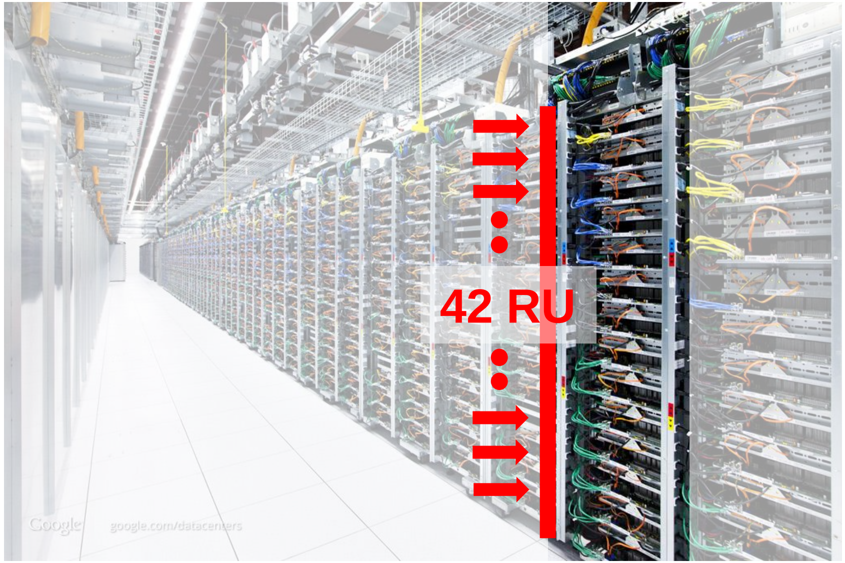
Lecture 10: Rack-scale data processing

Rack-scale

- What is a rack?

Rack-scale

- What is a rack?
  - The rack is the new unit of deployment in data centers
  - Sweet spot between a single-server and cluster deployments
  - It has 42 units (rack-units – RU) that host the compute resources



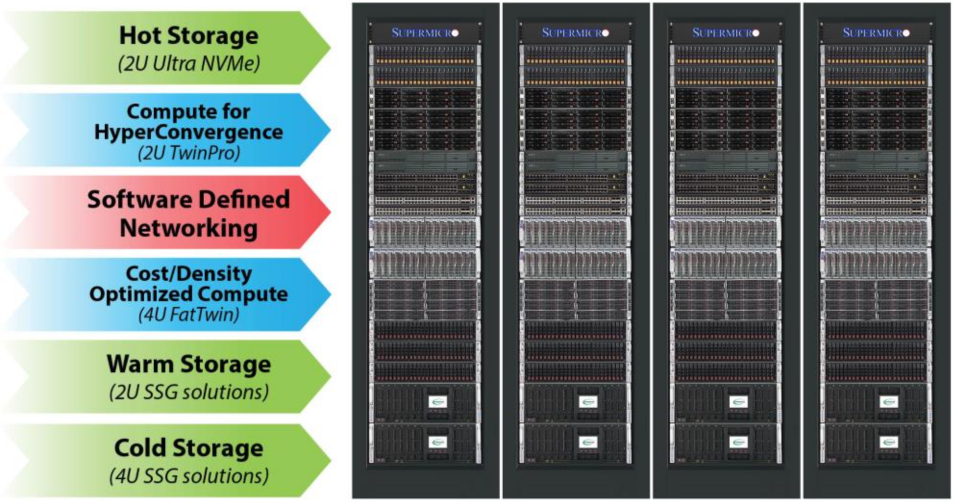
What’s in a Rack-scale computer?

Rack-scale computer (pre-packaged)

Compute:
- standard compute
- accelerators

Storage:
- hot / warm / cold disks

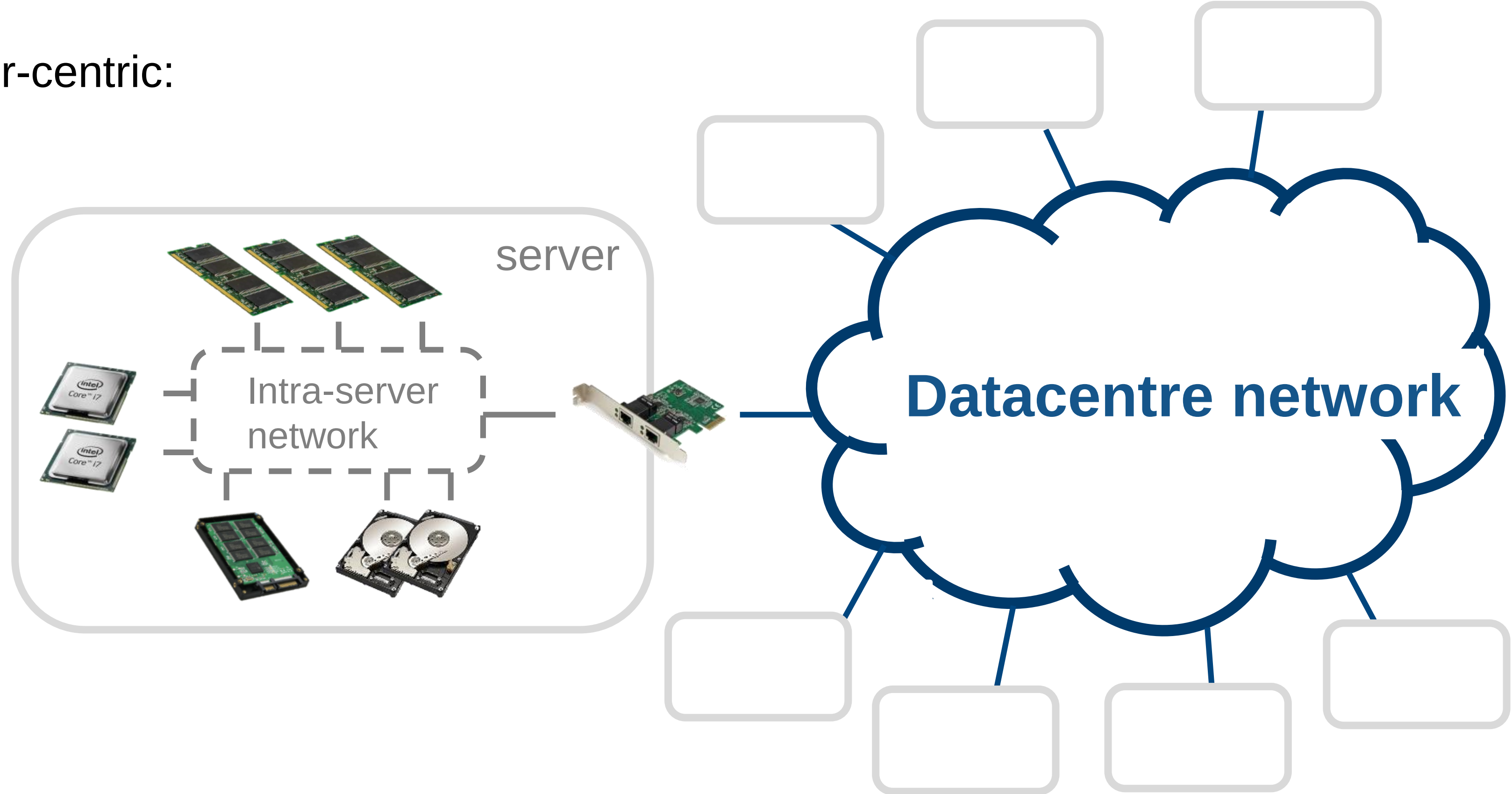
Networking:
- interconnect
- software defined networking



img src: Supermicro RSD

From server- to resource-centric datacentre design

Server-centric:



From server- to resource-centric datacentre design

Towards resource-centric

Past: physical aggregation

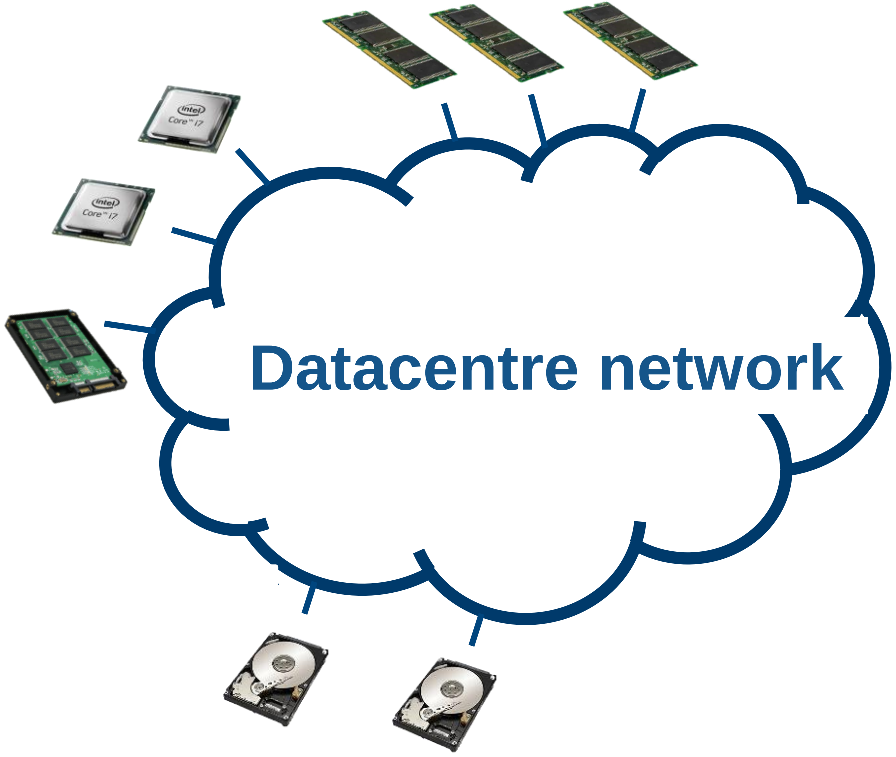
- shared power, cooling, rack-management

Now: fabric integration

- fast ***rack***-wide ***interconnect***

Future goal: resource disaggregation

- pooled compute, storage, memory resources



Today's scale within a rack computer

- We already have *scale* within a rack itself.

Machine	#cores
AMD SeaMicro *SM15000-64*	2'048
HP Moonshot *Redstone*	11'520
Boston Viridis	7'680

Machine	Memory
AMD SeaMicro *SM15000-XE*	8 TB
HP Moonshot *Redstone*	11.25 TB

Machine	Network
EDR Mellanox	100 Gbps
Intel silicon photonics	100-400 Gbps

- And increasing *heterogeneity* of resources

AMD Rack P47 – 1 PetaFLOP of compute at FP32 single precision

CPU	GPU	Memory	Network
20x AMD EPYC 7601	80x Radeon Instinct	10 TB DDR4	2x36 port EDR switch (100 Gbps)

Rack-scale *computing*

- How do we implement applications for a rack computer?

- How do we manage these resources?

- What is the failure model? How do we achieve fault tolerance?

Data appliances – among the first rack-scale apps

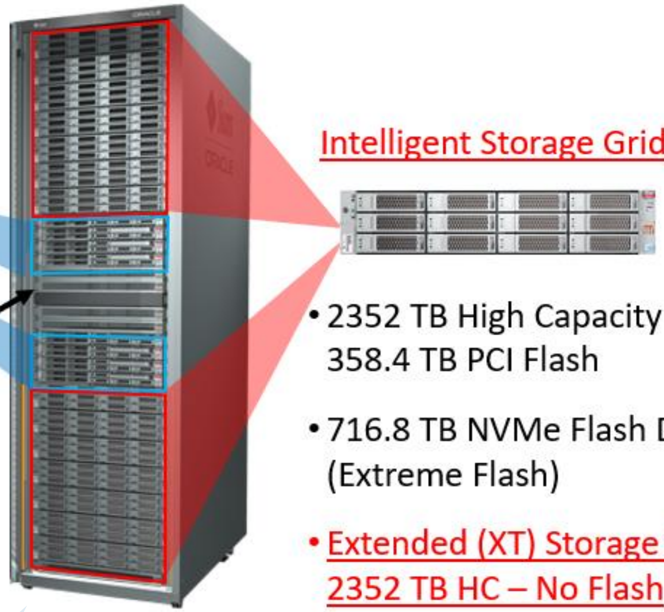
Oracle's Exadata
rack-scale data analytics
engine since 2008

Database Grid

- Exadata database servers (X8-2) , 384 cores Intel Xeon 8250 processor
- Up to 12TB Physical Memory

InfiniBand Network

- Redundant 40Gb/s (QDR) IB N/W
- Unified server & storage network



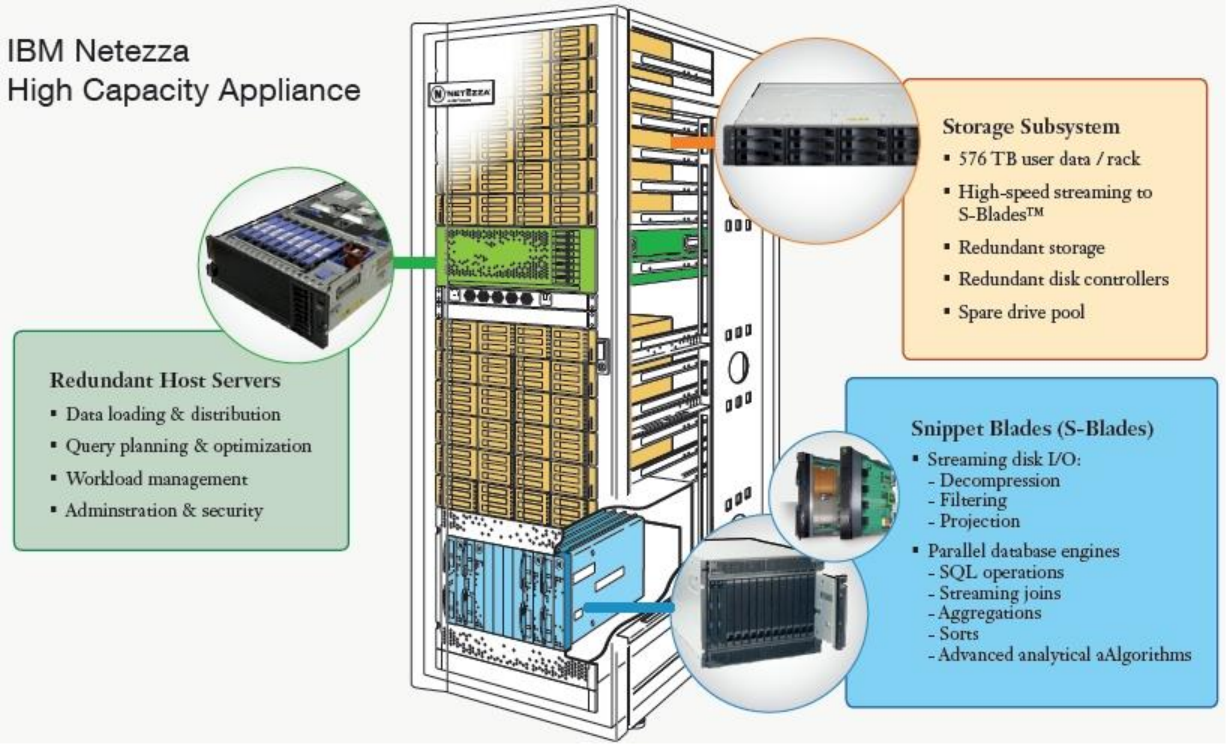
X8-2 Exadata Full Rack

Intelligent Storage Grid

- 2352 TB High Capacity disk & 358.4 TB PCI Flash
- 716.8 TB NVMe Flash Drive (Extreme Flash)
- Extended (XT) Storage Server – 2352 TB HC – No Flash
- Data mirrored across storage servers

Exadata X8M offers:
- RDMA over RoCE enabling 100Gb/s
- Persistant memory for new shared storage acceleration tier

Data appliances – among the first rack-scale apps



IBM Netezza
Heterogeneous appliance incorporating FPGA blades

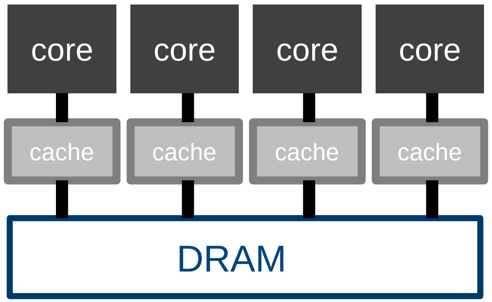
Figure from 2011

How do we program with remote memory?

Parallel architectures

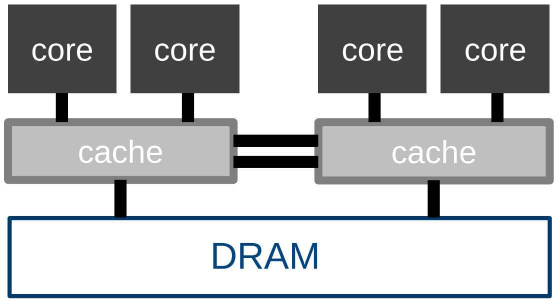
Today's Laptops

Uniform memory access (UMA)



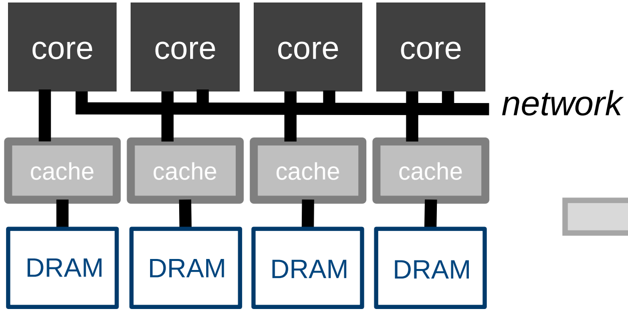
Today's Servers

Non-uniform memory access (NUMA)



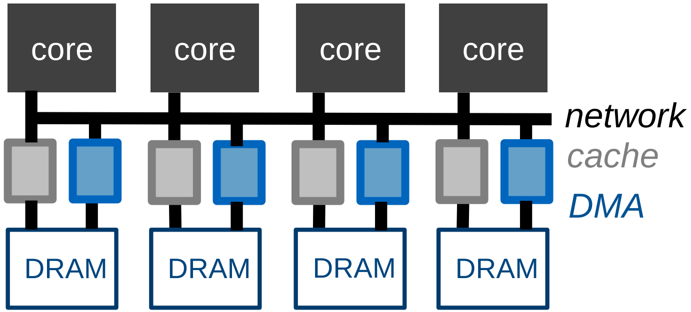
Yesterday's clusters

Time-division multiplexing



Today's clusters

Remote direct-memory access



Programming models

Shared memory programming

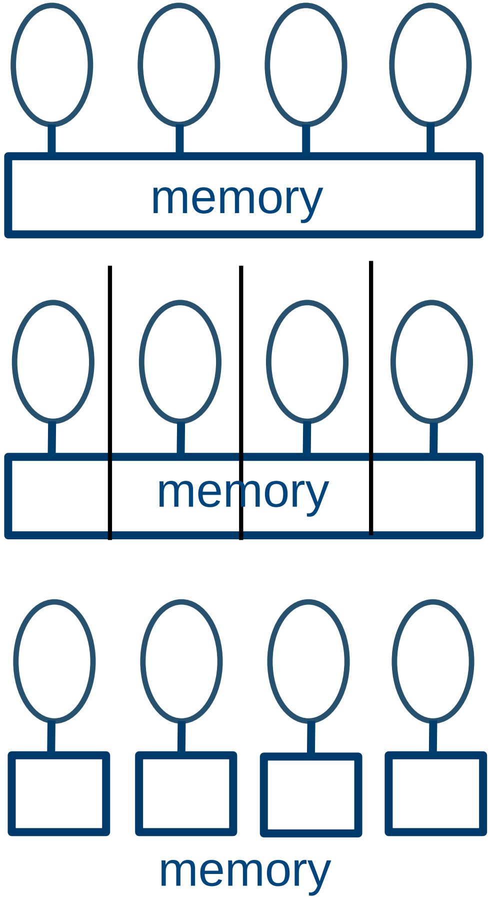
- shared address space
- implicit communication
- cache-coherent NUMA
- e.g., pthreads or OpenMP

(Partitioned) global address space

- Remote Memory Access
- Remote vs. local memory (e.g., ncc NUMA)

Distributed memory programming

- Explicit communication (e.g., with messages)
- Message passing



Remote Memory Access (RMA)

- Shared memory programming abstraction
- Access to remote memory region through ***explicit read*** and ***write*** operations.
- Similar to programming ***non-cache-coherent*** machines:
  - data needs to be ***explicitly loaded*** into the cache-coherency domain before it can be used
    - e.g., loaded in a register
  - changes to data have to ***explicitly flushed*** back to the source
    - so that the modifications become visible in the remote machine.

- The one-sided operations can optionally notify the remote process of an RMA access.

- Some implementations support ***atomic operations***:
  - fetch-and-add and
  - compare-and-swap

- RMA has been adopted by many libraries such as ibVerbs and MPI-3.

Partitioned Global Address Space (PGAS)

- PL concept for writing parallel applications for large distributed memory machines.

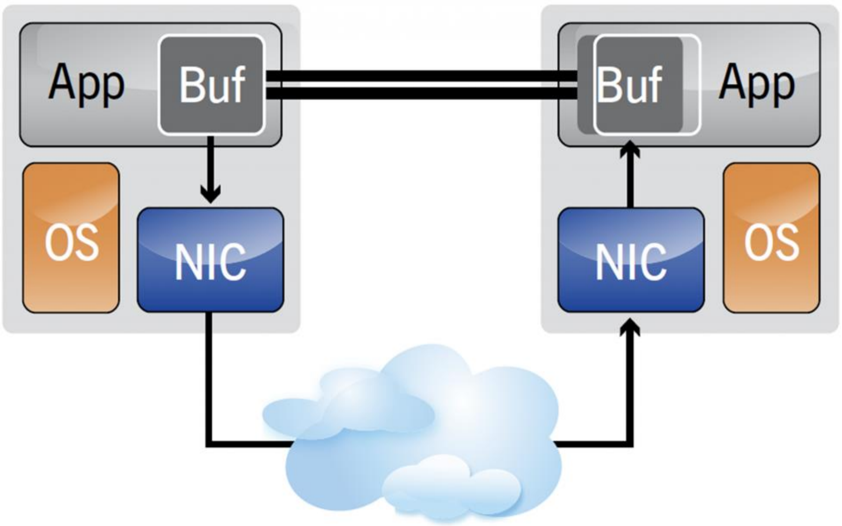
- Assumes a single global memory address space, partitioned across all the processes.

- The programming model differentiates between ***local*** and ***remote*** memory.
  - The ***compiler*** adds the necessary code to ***implement a remote*** variable ***access***.
  - From a programming perspective, a remote variable can be assigned to a local variable or register.
  - The ***developer*** needs to be ***aware*** of the ***implicit data movement*** when accessing shared variable.
  - Careful ***NUMA-like optimizations*** are ***required*** for ***high-performance***.

A popular approach – RDMA

- Remote Direct Memory Access

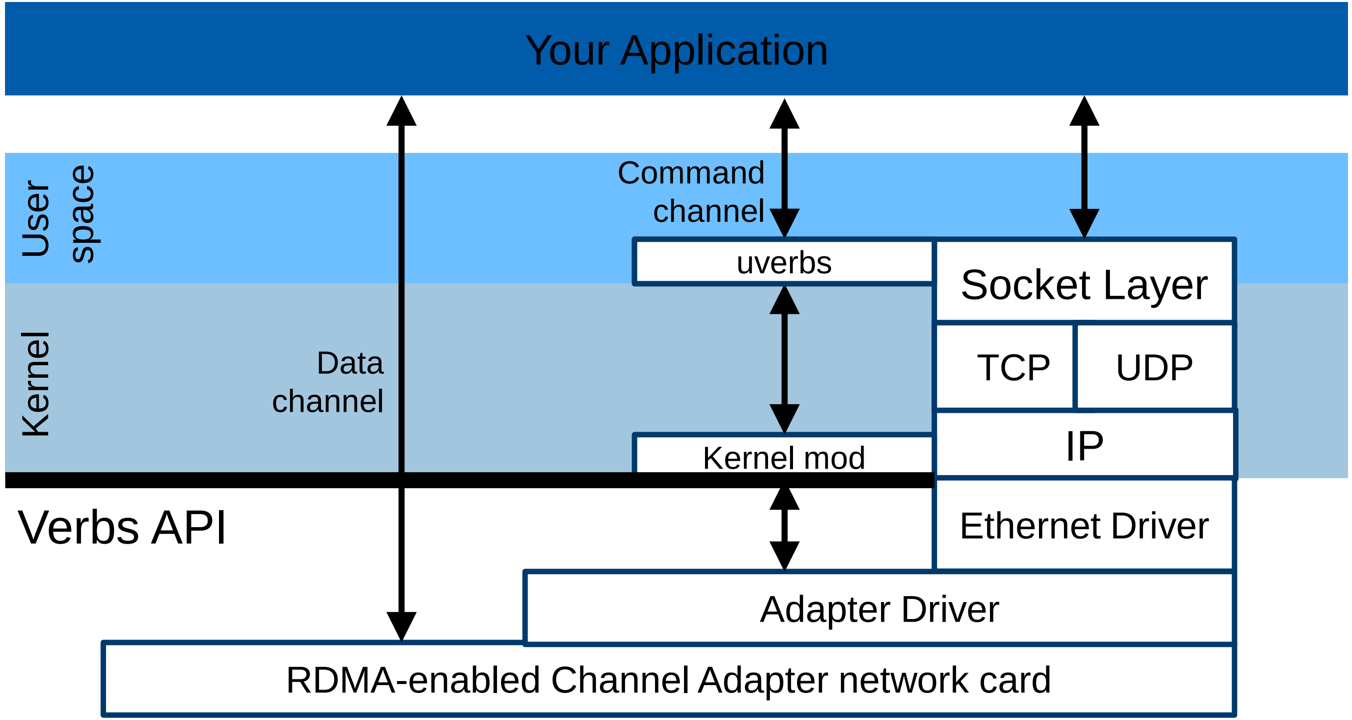
- RDMA is a HW mechanism through which the network card can directly access all or parts of the main memory of a remote node without involving the CPU.



RDMA properties

- Bypass the CPU → low CPU utilization
- Bypass the OS kernel → no interrupts, no context switching
- Zero-copy data → low memory bus contention
- Message based transactions
- Asynchronous operations → overlapping communication and computation

Traditional TCP/IP sockets vs RDMA



src: InfiniBand Trade Association: Introduction to IB for end users

"Expanding" the Memory hierarchy

registers

10'000 Gbps, 1ns

CPU cache

500 Gbps, 50 ns

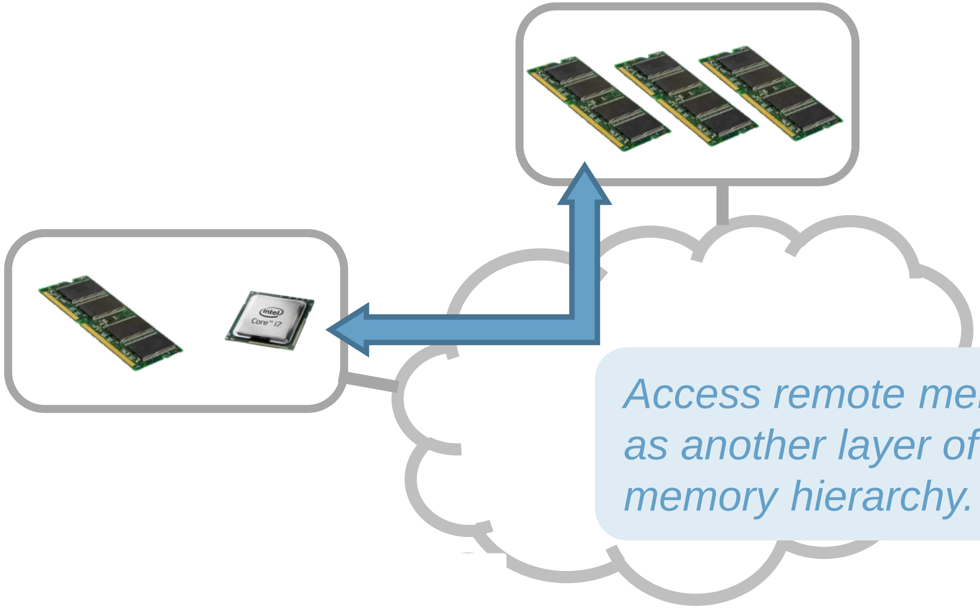
Local memory

100 Gbps, 5'000 ns

Remote memory

1-10 Gbps, 50'000 ns

SSD / HDD



Access remote memory as another layer of the memory hierarchy.

Microsoft Research showed that using Remote Memory (and RDMA) improves the latency of TPC-H and TPC-DS queries by 2-100x

Li et al. [SIGMOD 2016]

RDMA in research

High Performance Computing is the home research domain for RDMA

Databases

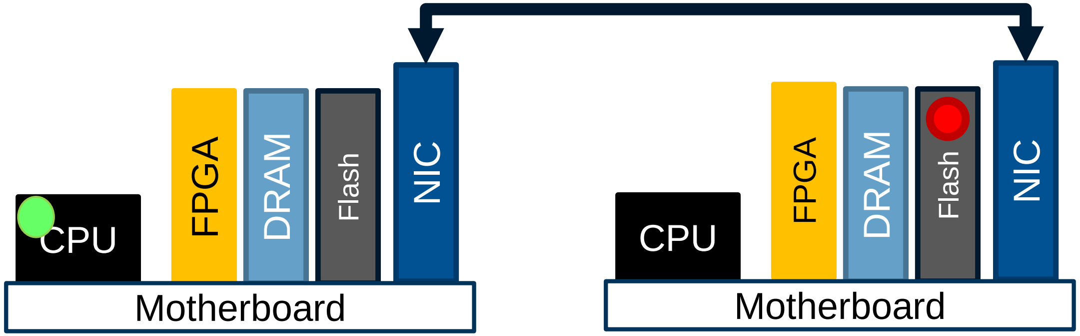
- ***Distributed transactions***
  FaSST [OSDI'16], FaRM [NSDI'14, SOSP'15], DrTM [SOSP'15], Tell [SIGMOD'15], NAM-DB [VLDB'17]
- ***RDMA KV-stores***
  RAMCloud [FAST'11, SOSP'11, SOSP'15], HERD [SIGCOMM'14], Pilaf [ATC'13]
- ***Distributed query processing***
  Barthels et al. [SIGMOD'15], Frey et al. [ICDCS'10], Rödiger et al. [ICDE'16]
- ***Accelerating RDBMS with RDMA***
  Li et al. [SIGMOD'16], BatchDB [SIGMOD'17]

Operating Systems

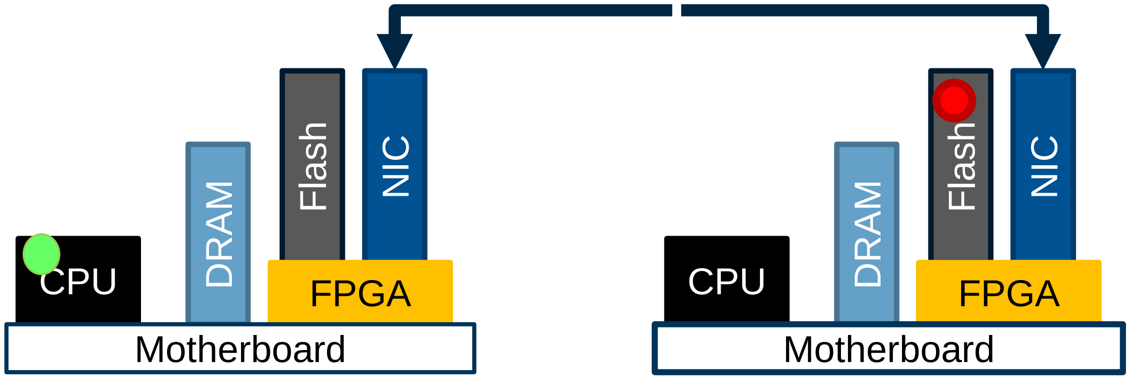
- Data-centres / Rack-scale computing: LITE [OSDI'17]

What about remote storage?

Traditionally:
- Accessing remote storage requires traversing the whole system stack.
- But, hardware and software latencies are additive.



Future:
- Intelligent storage
- BlueDBM [ISCA’15]
- Ibex [VLDB’14]



RDMA basics

Setting up the RDMA data channels

Buffers need to be ***registered*** with the ***network card*** before used

During the registration process:

- ***Pin memory*** so that it cannot be swapped by the Operating System.
- ***Store*** the ***address translation*** information ***in the NIC***.
- ***Set permissions*** for the memory region.
- ***Return*** a ***remote*** and ***local key***, which are used by the adapters when executing the RDMA operations.

Work Queues

RDMA communication is based on a set of three queues

- ***Send***
- ***Receive***
- ***Completion***

work queues, always created as a Queue Pair (QP)

The ***send*** and ***receive*** queues are there to schedule the ***work*** to be done.

A ***completion*** queue is used to ***notify*** when the work has been completed.

Queue Elements

Applications issue a job using a ***work request*** or a ***work queue element***

A work request is a small ***struct*** with a ***pointer to a buffer***:

- In a *send queue* – it's a pointer to a message to be sent.
- In a *receive queue* – it's shows where an incoming message should be placed.

Once a work request has been completed, the adapter creates a ***completion queue element*** and enqueues it in the *completion queue*.

RDMA’s network stack overview

Application	Posts work requests to a queue Each work request is a message, a unit of work
(Verbs interface)	Verbs interface – allows the application to request services
RDMA adapter driver	Maintains the work queues Manages address translation Provides completion and even mechanisms
RDMA-supporting NIC and network protocols	Transport layer: reliable/unreliable, datagram, etc. Packetizes messages Implements the RDMA protocol Implements end-to-end reliability Assures reliable delivery

src: InfiniBand Trade Association: Introduction to IB for end users

Network protocols supporting RDMA

InfiniBand (IB)

- QDR 4x – 32 Gbps
- FDR 4x – 54 Gbps
- EDR 4x – 100 Gbps

RoCE – RDMA over Converged Ethernet

- 10 Gbps
- 40 Gbps

iWARP – internet Wide Area RDMA Protocol

RDMA is just a *mechanism*

Does ***not*** specify the ***semantics*** of a data transfer

RDMA networks support two types of memory access models:

One sided – RDMA read and write + atomic operations
Two sided – RDMA send and receive

RDMA Send and Receive

Traditional message passing where ***both*** the ***source*** and the ***destination*** processes are ***active***ly involved in the communication.

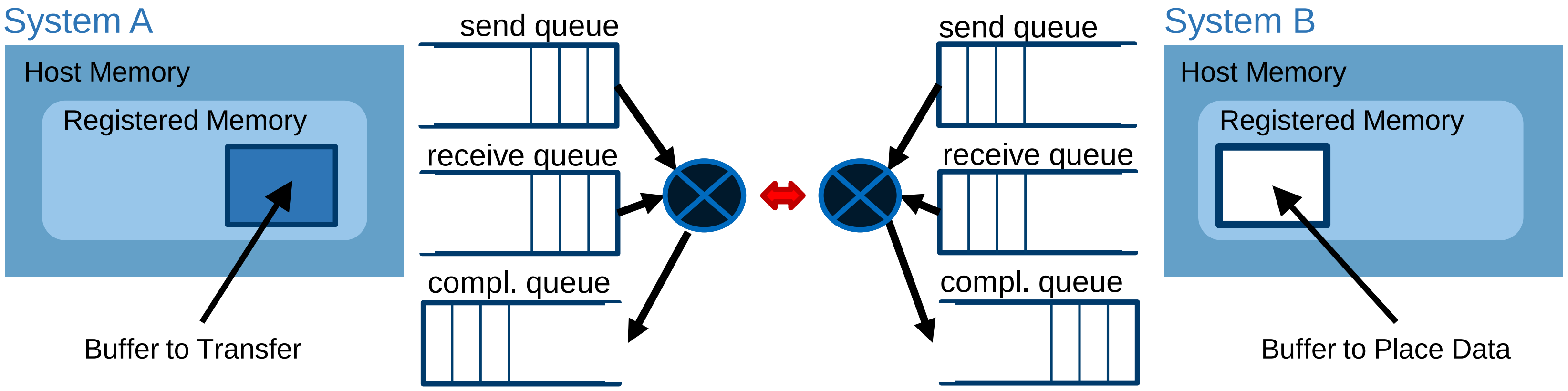
Both need to have ***create***d their queues:

- A ***queue pair*** of a ***send*** and a ***receive*** queue.
- A ***completion queue*** for the queue pair.

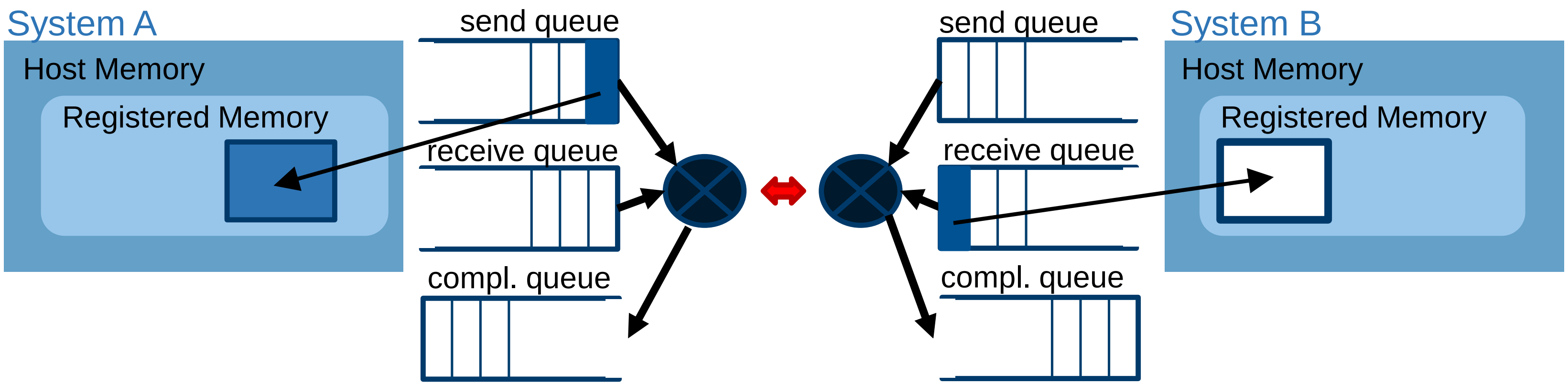
Sender's work request has a pointer to a buffer that it wants to send. The WQE is enqueued in the send queue.

Receiver's work request has a pointer to an empty buffer for receiving the message. The WQE is enqueued in the receive queue.

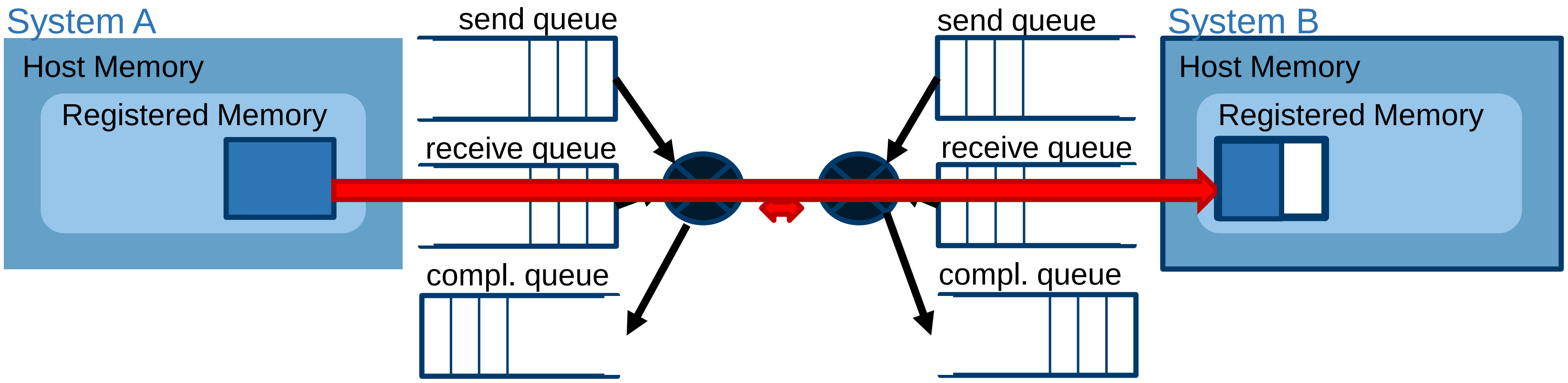
Example RDMA send



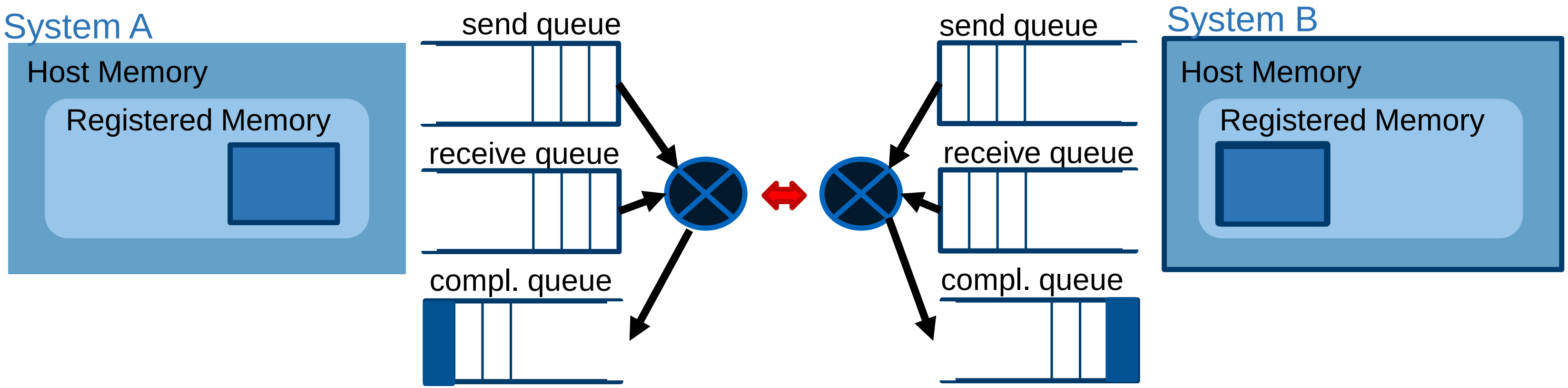
Example RDMA send



Example RDMA send



Example RDMA send



RDMA Read and Write

Only the ***sender*** side is ***active***; the ***receiver*** is ***passive***.

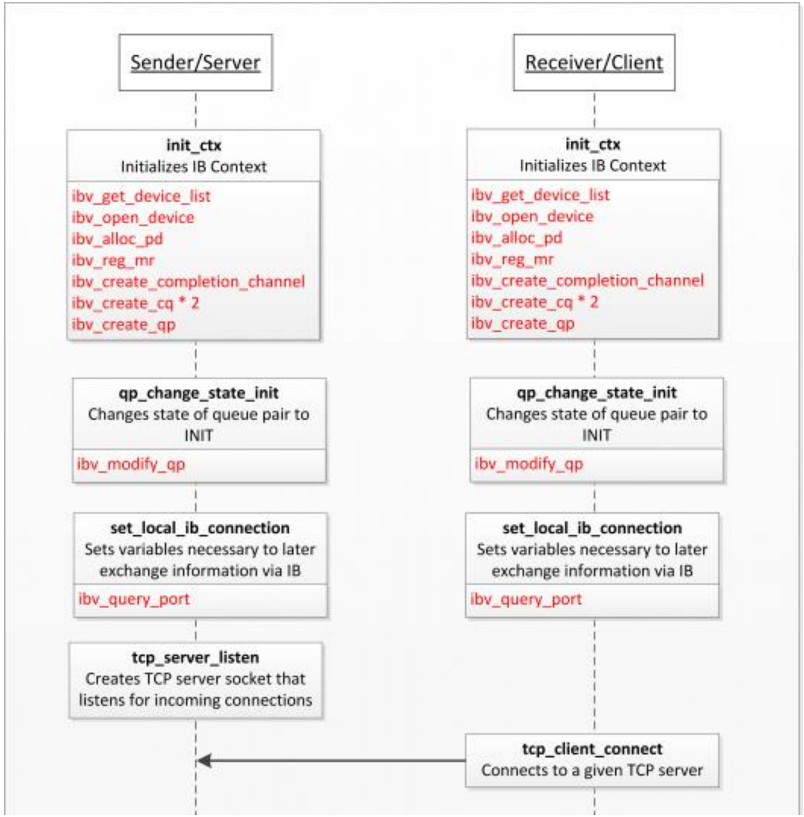
The passive side issues no operation, uses no CPU cycles, gets no indication that a "read" or a "write" happened.

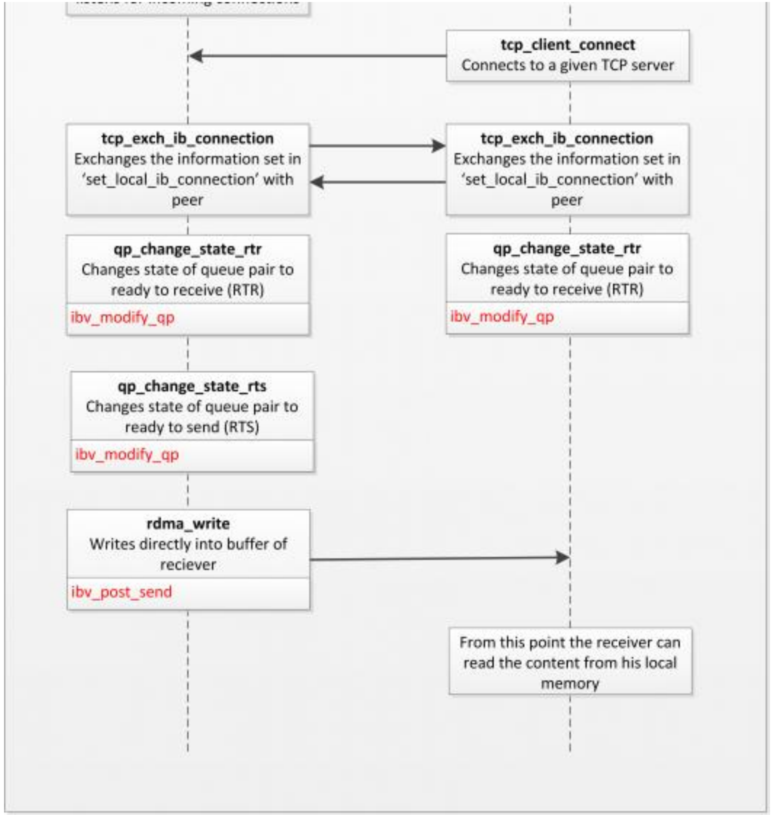
To issue an RDMA ***read*** or a ***write***, the work request ***must include:***

1. the ***remote*** side's ***virtual memory address*** and
2. the ***remote*** side's ***memory registration key***.

The active side must obtain the passive side's address and key beforehand. Typically, the traditional RDMA send/receive mechanisms are used.

Using the verbs API

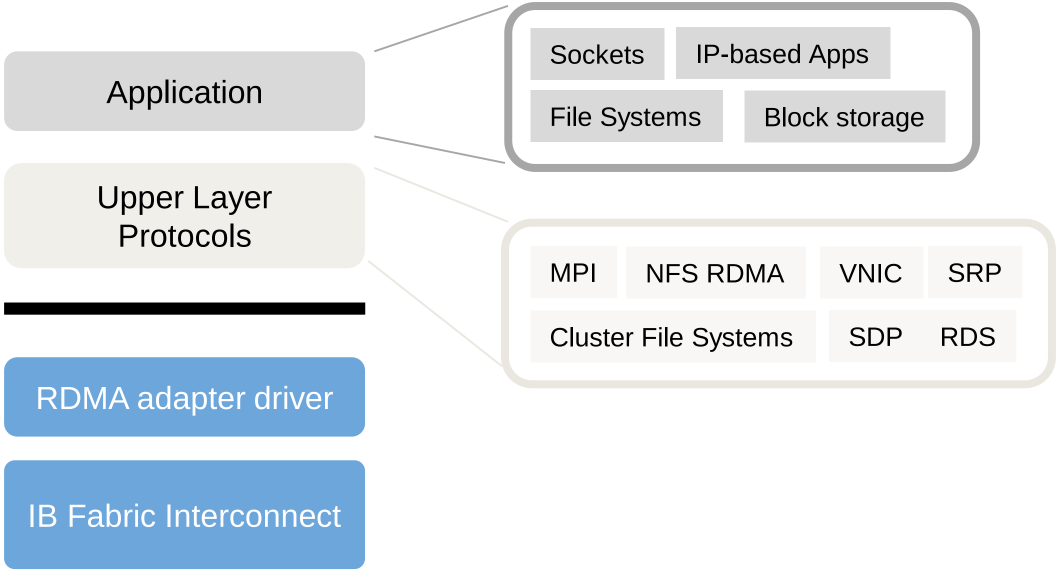




src: https://blog.zhaw.ch/icclab/infiniband-an-introduction-simple-ib-verbs-program-with-rdma-write/

Challenges of using RDMA

Added extra complexity for the developer to use the Verbs API



MPI : Message Passing Interface

- Widely used in HPC
- Example: OpenMPI, MVAPICH, Intel MPI, etc.

File Systems:

- Lustre – parallel distributed FS for Linux
- NFS_RDMA – Network FS over RDMA

src: InfiniBand Trade Association: Introduction to IB for end users

RDMA References

- IB trade introduction https://cw.infinibandta.org/document/dl/7268
- First steps for programming with IB verbs https://thegeekinthecorner.wordpress.com/2010/08/13/building-an-rdma-capable-application-with-ib-verbs-part-1-basics/
- Figures from https://zcopy.wordpress.com/category/getting-started/
- More details at http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf

Overview of our EDR cluster

- **EDR InfiniBand**
- **36-port** Mellanox switch
- **18 nodes cluster** (EDR NICs)

- **1 server with 4 Xeon E5-5660 v4 processors:**
  - 64 cores (128 with HT enabled)
  - 512 GB RAM
  - 2 EDR NICs, 1 x 10G NIC, 1 x 1G NIC

- **8 servers with 2 Xeon E5-2630 v4 processors:**
  - 20 cores (40 with HT enabled)
  - 32 GB RAM
  - 2 EDR NICs

RDMA-based joins

Good practices

- Memory region registration cost increases with the number of registered pages
  - To avoid pinning large parts of main memory, we need efficient buffer management
  - The algorithm should reuse existing RDMA-enabled buffers as much as possible

- RDMA requires asynchronous communication:
  - to prevent processor cores from becoming idle, we need to overlap computation with communication.

- Accessing remote memory is slower than local memory, even with RDMA
  - Again, it is important to hide the network latency, by interleaving computation and communication

- Watch out for NUMA effects in RDMA-based algorithms
  - Only threads local to the NUMA node where the buffers are registered should communicate

RDMA-version of the Radix Join

1. Histogram computation phase

- All threads within the same machine compute a histogram over the input data
- They exchange their histograms and combine them into one machine-level histogram providing an overview of the data residing on a particular machine
- The machine-level histograms are then exchanged over the network → compute the global histogram
  - Global overview of the partition sizes
  - Necessary size of the buffers to be allocated to store the data to be sent/received over the network

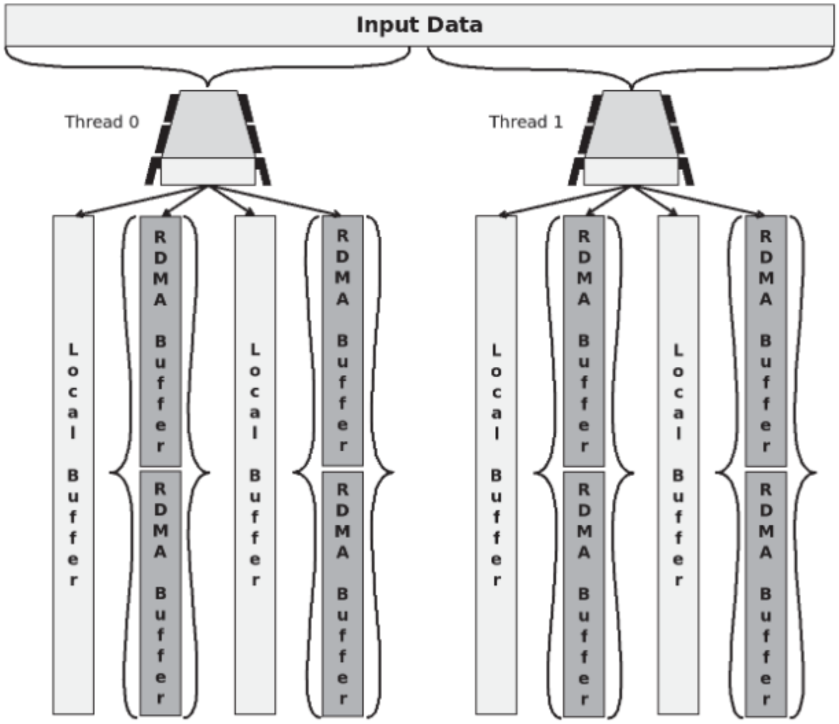
Src: Barthels et al. Rack-scale In-memory Join Processing using RDMA. SIGMOD 2015

RDMA-version of the Radix Join

2. Partitioning phase

- Distinguish between two types of partitioning passes:
  - Network partitioning pass, which interleaves computation of the partitions with network transfer
  - Local partitioning pass, which partitions the data locally to ensure that the partitions fit in the caches

- Network partitioning pass:
  - Pool of RDMA-enabled buffers
  - While the content of one buffer is transmitted over the network, populate another one.
  - All buffers are private to each thread, to avoid the need of synchronization



Src: Barthels et al. Rack-scale In-memory Join Processing using RDMA. SIGMOD 2015

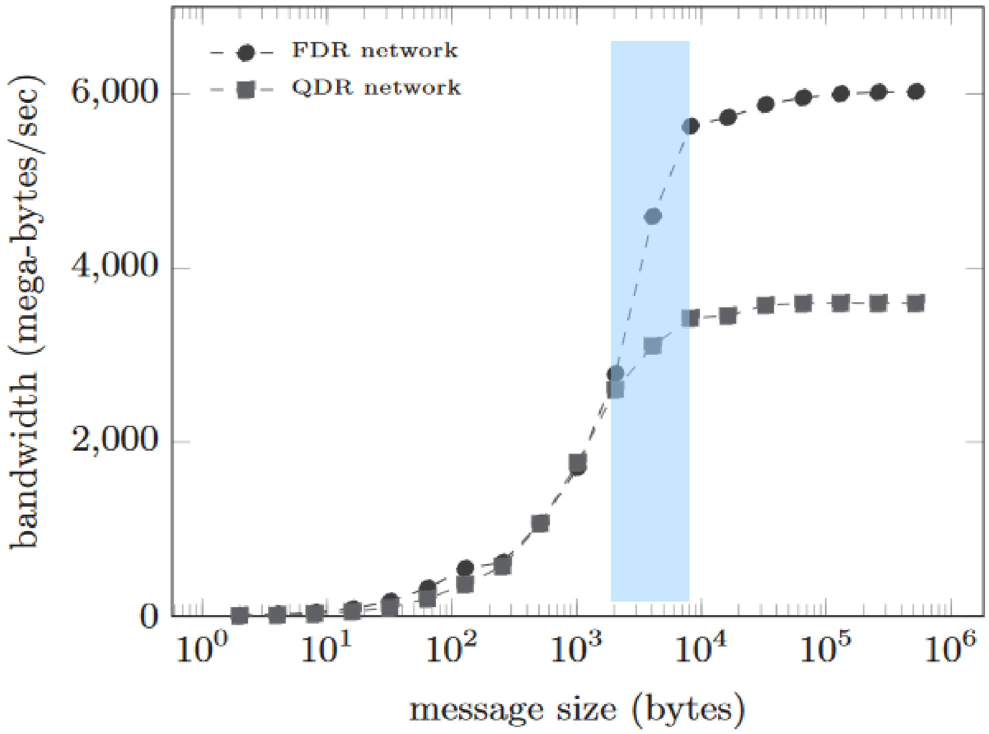
RDMA-version of the Radix Join

3. Build and Probe

- The matching partitions of both relations should by now be on the same machine and cache-size resident.
- The build phase populates a hash table and the corresponding partition of the output relation probes it
- The matching results are either written out to a local buffer or to an RDMA-enabled buffers, depending on the location where the results will be further processed.
- The RDMA-enabled buffer is transmitted over the network once it is full. The buffer can be reused once the network operation has completed.

Src: Barthels et al. Rack-scale In-memory Join Processing using RDMA. SIGMOD 2015

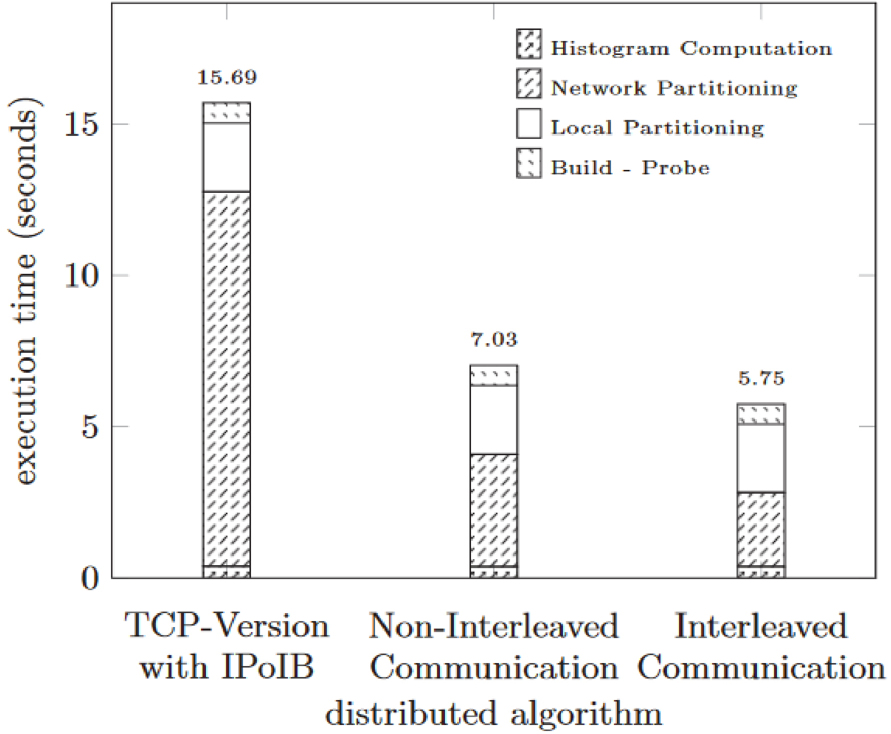
RDMA performance intrinsics



The best performance can only be achieved after a certain message size!

Src: Barthels et al. Rack-scale In-memory Join Processing using RDMA. SIGMOD 2015

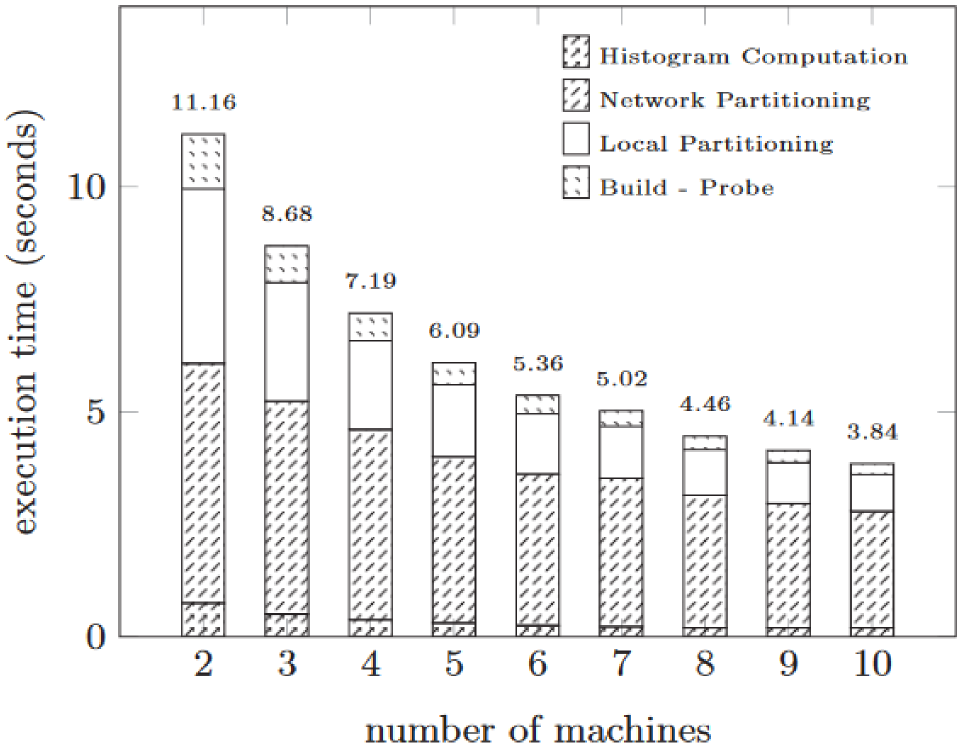
RDMA-based join performance



- Set-up: Joining 2x2048M tuples. Running on 4 machines with 32 CPUs (FDR cluster).
- Just using RDMA already brings significant performance improvements over a traditional IP-based network stack.
- Interleaving computation with communication brings additional 20% improvement.
- The network- and local- partitioning are the most expensive operations.

Src: Barthels et al. Rack-scale In-memory Join Processing using RDMA. SIGMOD 2015

RDMA-based join performance



- Set-up: speed-up experiment, measuring the execution time for a 2048x2038M tuples on a variable number of machines (QDR cluster).
- With the increasing number of machines, the network-partitioning phase becomes the dominant performance bottleneck.
  - Speed-up of only 2.9 when scaling from 2 to 10 machines.
  - A larger fraction of the data needs to be transmitted over the network.
  - Additional congestion over the network.

Src: Barthels et al. Rack-scale In-memory Join Processing using RDMA. SIGMOD 2015