When MPPDB Meets GPU:

An Extendible Framework for Acceleration

Laura Chen, Le Cai, Yongyan Wang

www.huawei.com

HUAWEI TECHNOLOGIES CO., LTD.

HUAWEI

Background: Heterogeneous Computing

Hardware Trend

- **CPU** stops growing with Moore's Law
- **Fast development of GPU**
  - › Since 1993, GPU performance speed up 2.8X per year
  - › GPU computation power and bandwidth can be 39~50X of CPU

Computation (TFLOPS):
- Intel：1
- NAVIDIA(8x)：52.8

Bandwidth(GB/s)：
- Intel：68
- NAVIDIA(8x):2692

GPU-accelerated DB

- **Positioning：** a differentiator for high-end market, for workload that is sensitive to response time, support real-time computing and decision making
- **State-of-Art Research:** join, sort, group-by, etc.
- **Speedup from end-to-end:** (for complicated analytical queries)
  - › Main-stream GPU： 5 ~10X
  - › High-end GPU: 10 ~ 20X

goai
GPU OPEN ANALYTICS INITIATIVE

SQREAM TECHNOLOGIES

BLAZINGDB

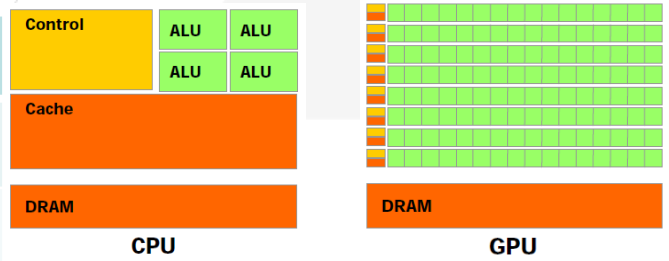
MAPD

kinetica

Background: GPU vs. CPU

Features Comparison

	GPU	CPU
Model	Nvidia TITAN X	Intel Xeon E7-8890 v4
Architecture	Pascal	Broadwell
Launch	Aug-2016	Jun-2016
# of cores	**3584 (simple)**	24 (functional)
Core clock	1.5 GHz	2.2 GHz, up to 3.4 GHz
Peak Flops (single precision)	11 TFLOPS	1689.6 GFLOPS (with AVX2)
DRAM size	12 GB, GDDR5X (384 bits bus)	768 GB/socket, DDR4
Memory band	**480 GB/s**	85 GB/s
Power consumption	250 W	165 W
Price at launch	$1200	$7,174



- GP compute unit
- Include Cache and Control

- Data parallelism
- Mostly computation unit

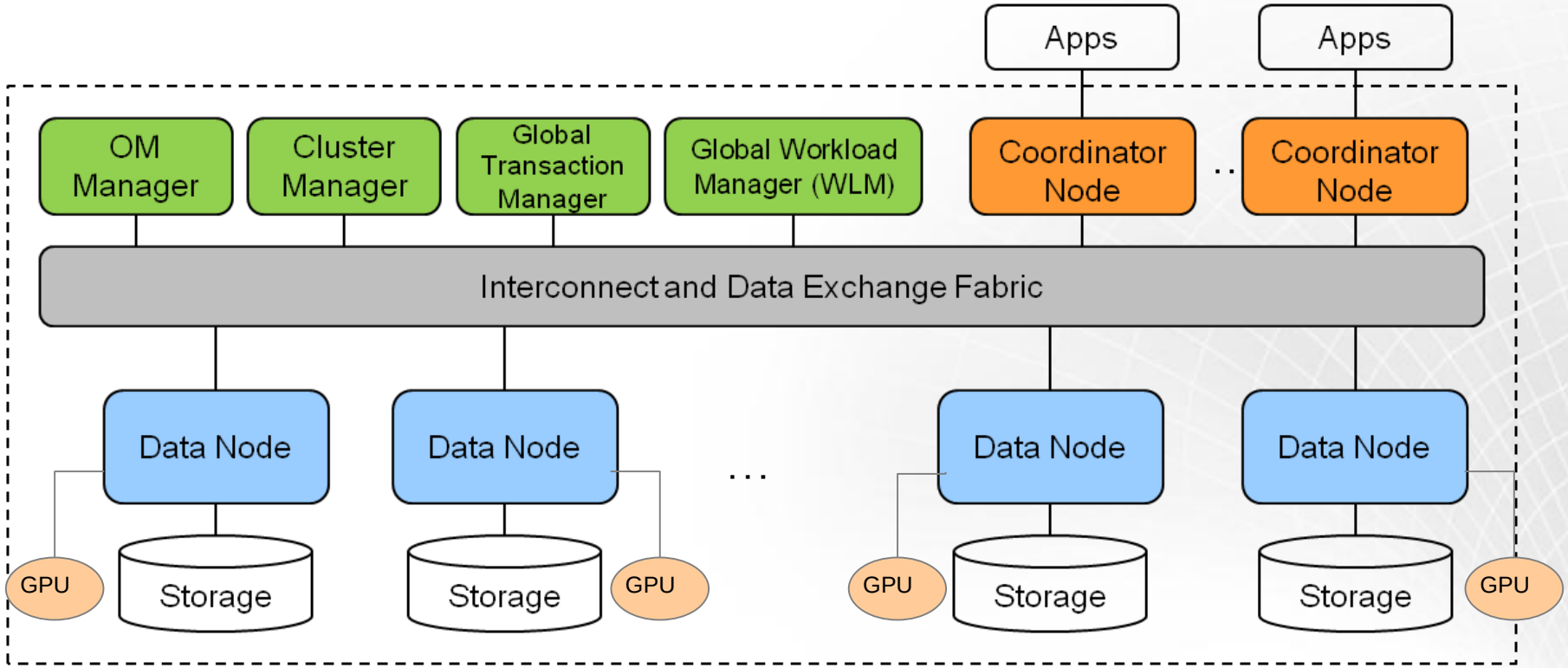
- Massive parallel cores
- Much higher DRAM bandwidth
- Better price/performance ratio
- Efficient for:
  - Parallel floating point processing
  - SIMD ops
  - High comp. per mem access
  - Agility in multi-threading
- Not efficient for:
  - Complex control logic
  - Logic operations
  - Random access, mem-intensive ops

Optimization Strategies Comparison

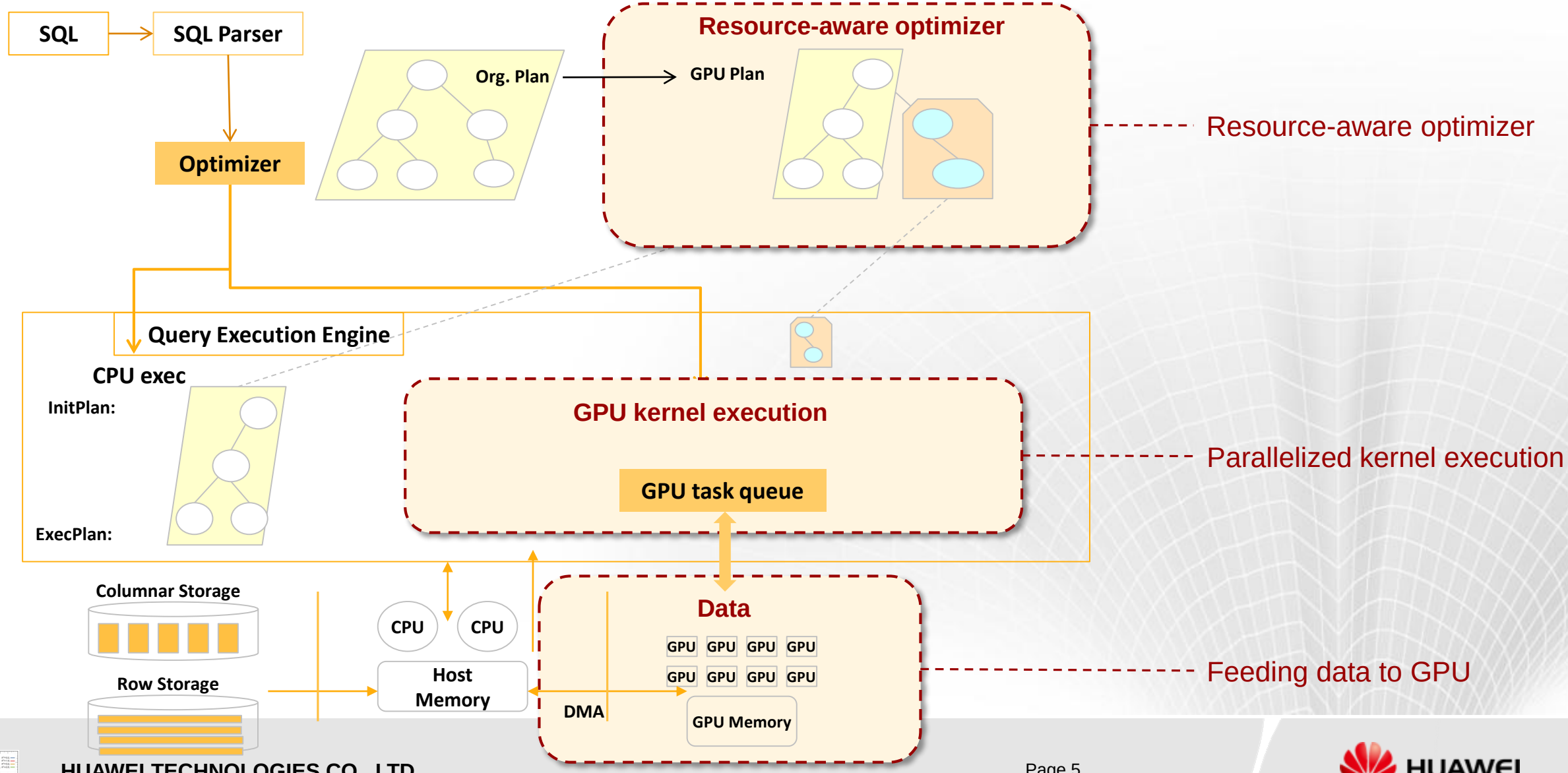
	GPU	CPU
Parallelism	Fine-grained massive data parallelism - Thousands of cores - Hide memory latency	- Multi-threaded: coarse-grained task parallelism - SIMD: fine-grained data parallelism (max. 512-bit SIMD width)
Memory access pattern	Coalesced: high memory bandwidth with high latency e.g., < 10 GB/sec for random > 100 GB/sec for coalesced	Sequential
Cache usage	Programmable L1 cache (up to 48 KB) on each multi-processor - The key to improve data locality	(Almost) Transparent to programmers
Global sync	Not supported	Feasible
Other concerns	- Branch divergence - Slow PCI-E bandwidth - Relatively small memory size	

When MPPDB Meets GPU

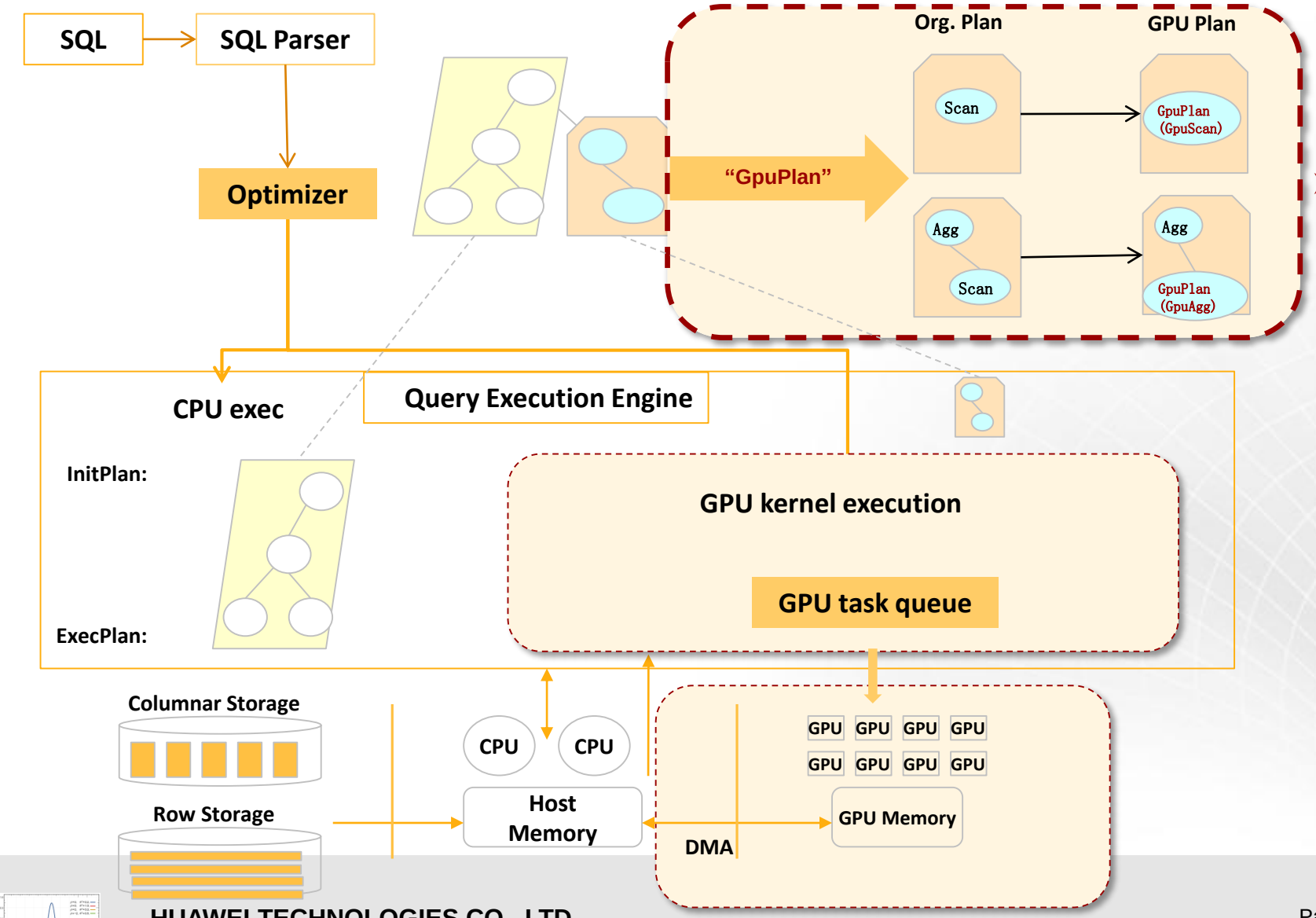
- MPPDB: share-nothing architecture, massively parallel processing
- Fully utilize CPU multi-core: SMP, etc.
- Hybrid approach: worth to offload compute-intensive operators to GPU? How?



Extendible Framework for Hardware Acceleration



Resource-Aware Optimizer

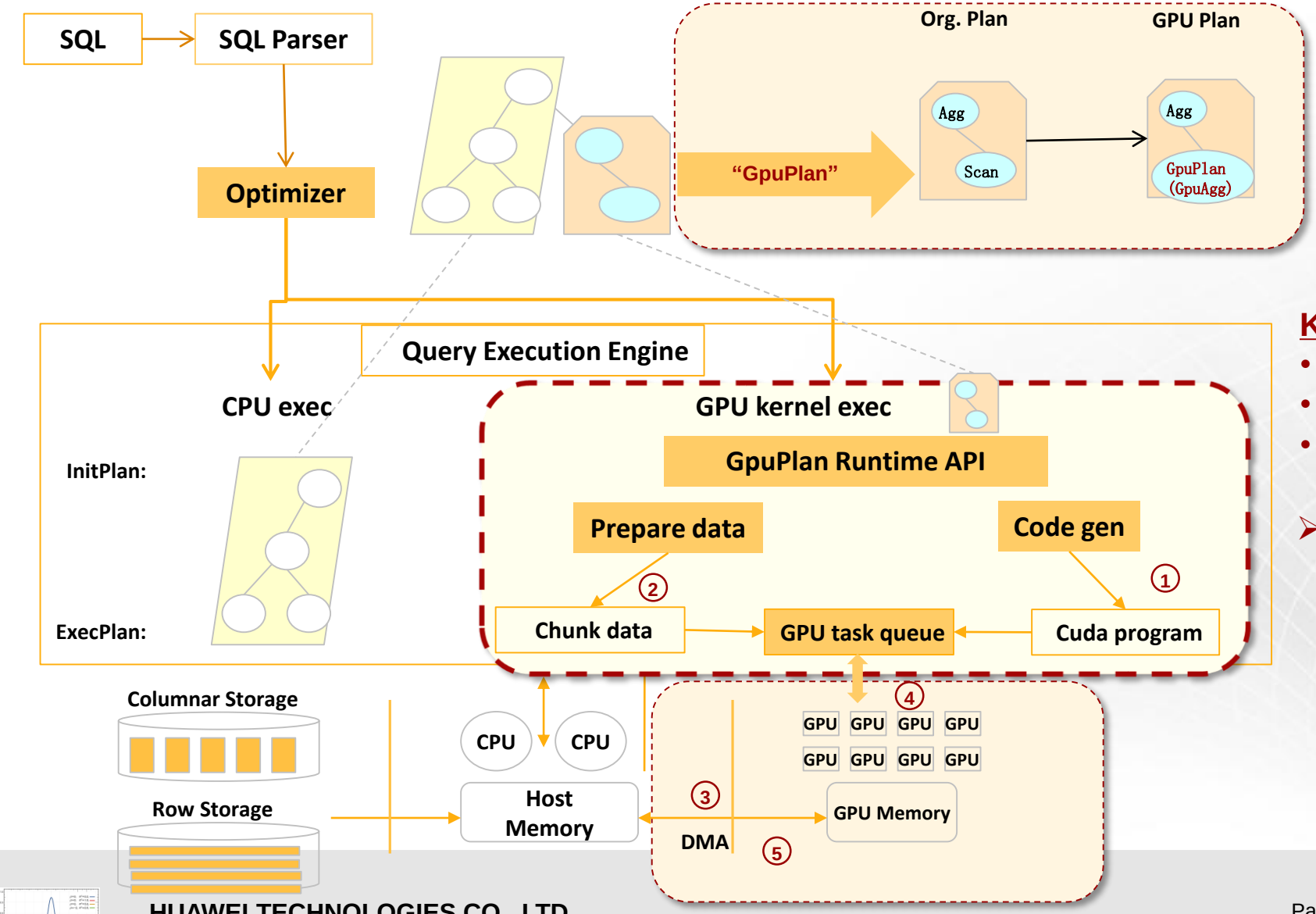


Get the best out of both CPU and GPU

➢ **Plan Time:**

- Enhanced cost-model to determine either use CPU or GPU plan, push which operators to GPU.
- Wrap GPU plan nodes in unified "GpuPlan" node
- Use plan hook to hook up to other parts of plan
- Interface between plan nodes stay same
- GpuPlan is the connector for GPU to the original runtime engine
- Same API can be applied to operators such as Scan, Agg, Join, Sort, and can connect to other hardware (FPGA)

Parallelized GPU Kernel Execution



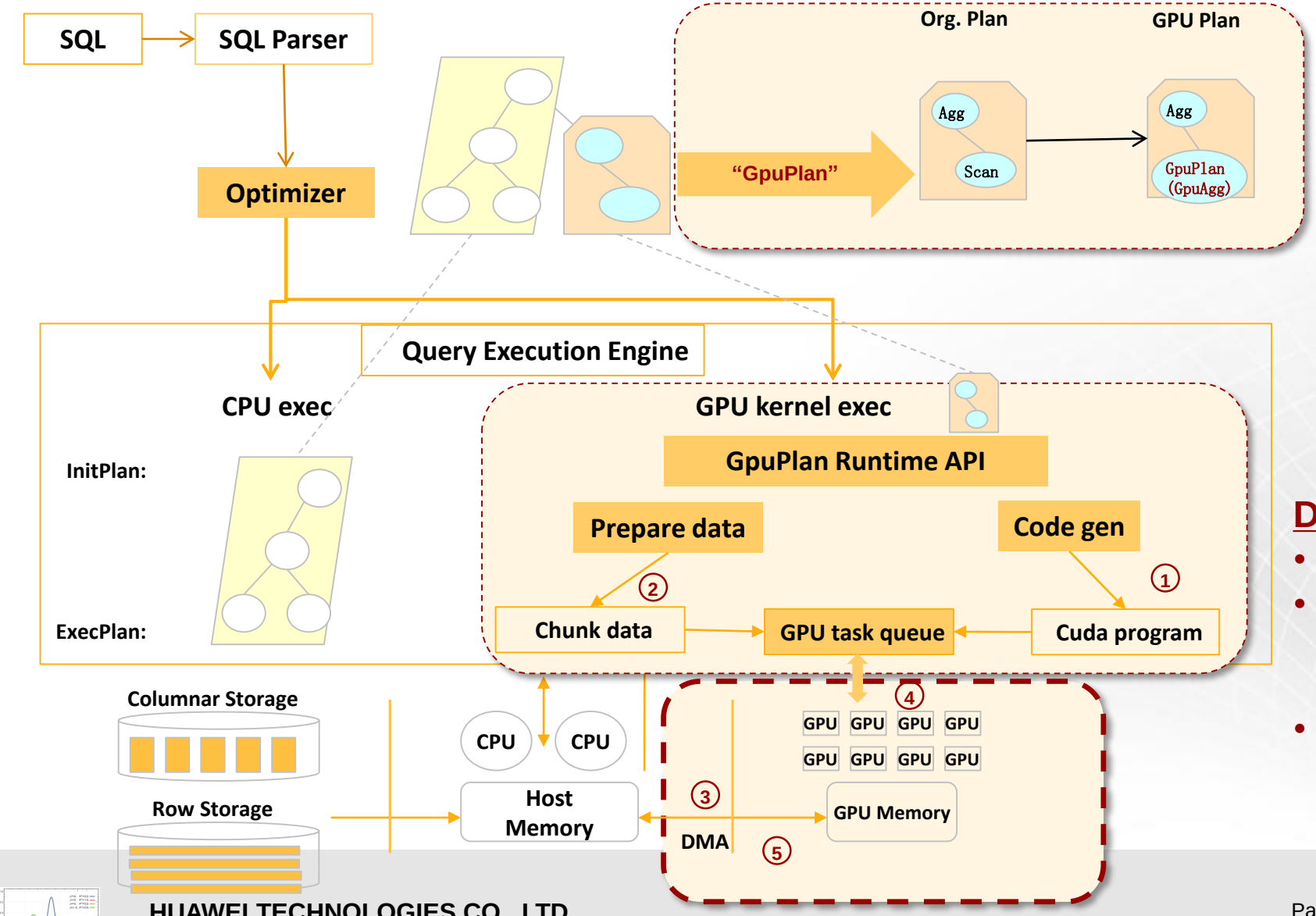
Kernel algorithms on GPU:

- Fine-grained massive data parallelism
- Coalesced access pattern
- JIT compilation

➢ **Exec Time:**

1. Code gen to generate CUDA program on-the-fly (JIT compile)
2. Prepare chunk size of data for GPU tasks, load to DMA buffer
3. Kick asyn DMA over PCI-E
4. Launch GPU kernel program
5. Write back results

Feeding Data to GPU

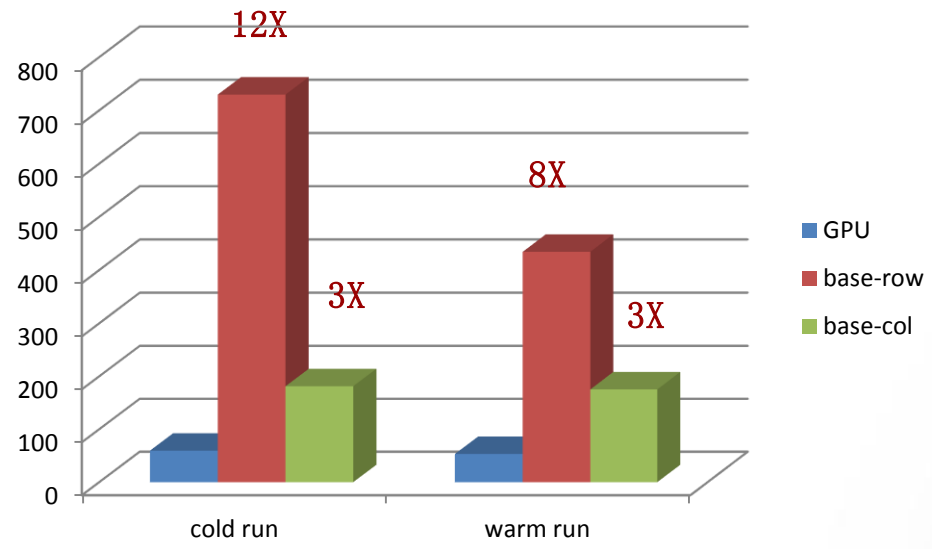


Data flow to GPU:

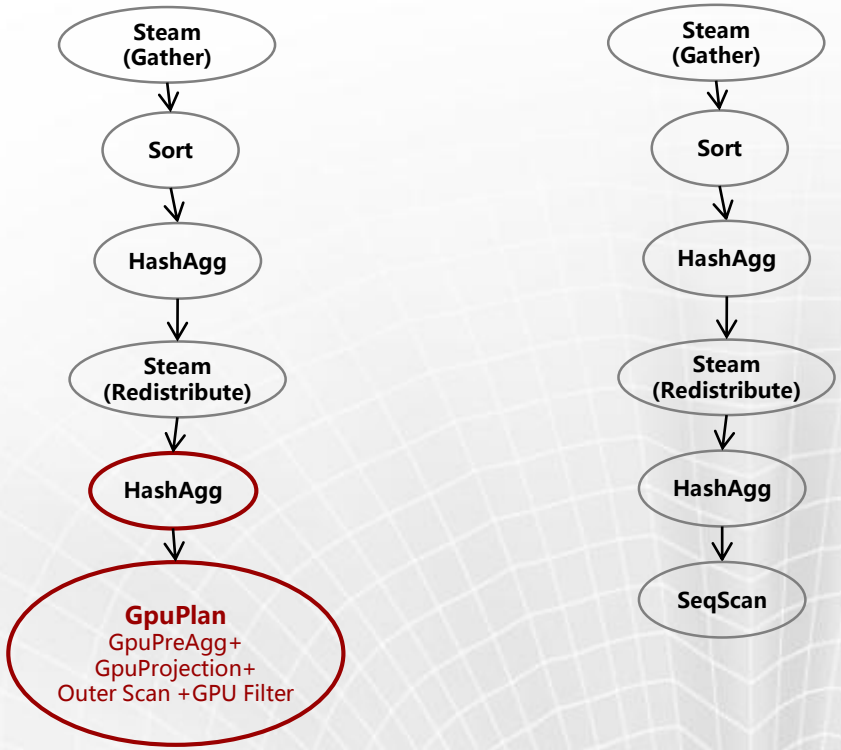
- Assemble chunk size data
- Through PCI-e bus (16GB/s):
  - HDD/SSD → mem →GPU mem
  - SSD → GPU mem
- Through high-speed connect:
  - NVLink (5~12x), CAPI, CCIX

Example: TPC-H Q1

TPC-H Q1 100x (with 1 Nvidia GTX 1080)



TPCH-Q1 : GPU PLAN CPU PLAN




```
QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Streaming (type: GATHER)  (cost=440338.85..440335.24 rows=6 width=36) (actual time=8534.439..8534.440 rows=4 loops=1)
  Node/s: All datanodes
  ->  Sort  (cost=440334.85..440334.86 rows=6 width=36) (actual time=[8205.003,8205.003]..[8205.321,8205.321], rows=4)
        Sort Key: l_returnflag, l_linestatus
        Sort Method: quicksort  Memory: 25kB
        ->  HashAggregate  (cost=440334.27..440334.83 rows=6 width=36) (actual time=[8204.954,8204.955]..[8205.269,8205.270], rows=4)
              Group By Key: l_returnflag, l_linestatus
              ->  Streaming(type: REDISTRIBUTE)  (cost=220167.14..220167.59 rows=12 width=36) (actual time=[8128.316,8204.796]..[8163.807,8205.152], rows=8)
                    Spawn on: All datanodes
                    ->  HashAggregate  (cost=220167.14..220167.24 rows=12 width=36) (actual time=[5906.548,5906.552]..[5932.072,5932.076], rows=8)
                          Group By Key: l_returnflag, l_linestatus
                          ->  GpuPlan (GpuPreAgg) on lineitem  (cost=10671.20..108150.40 rows=0 width=72) (actual time=[5824.801,5906.320]..[5851.165,5931.877], rows=336)
                                Reduction: Local + Global
                                GPU Projection: l_returnflag, l_linestatus, l_quantity, l_extendedprice, l_discount, l_tax
                                Outer Scan: lineitem (never executed)
                                GPU Filter: (l_shipdate <= '1998-11-28 00:00:00'::timestamp without time zone)
```

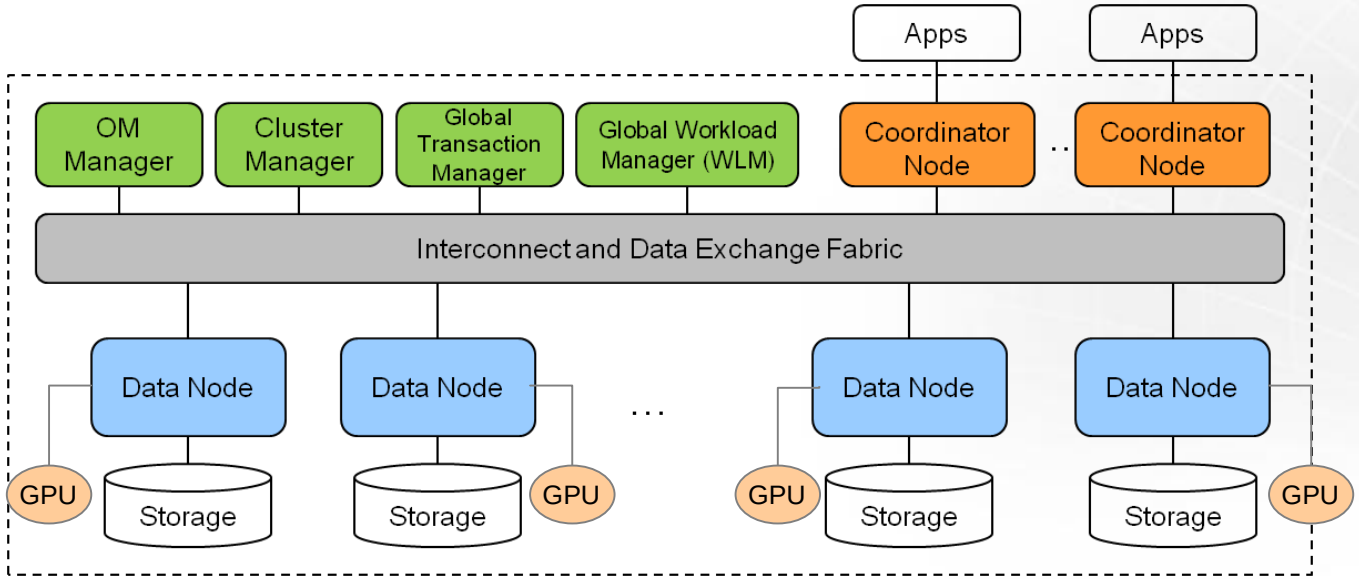
More Challenges for GPU on MPPDB

How to share GPU resources by multi-tasks concurrently?

- Abstract GPU mem to enable context-switch
- Resource pool of logical GPU units

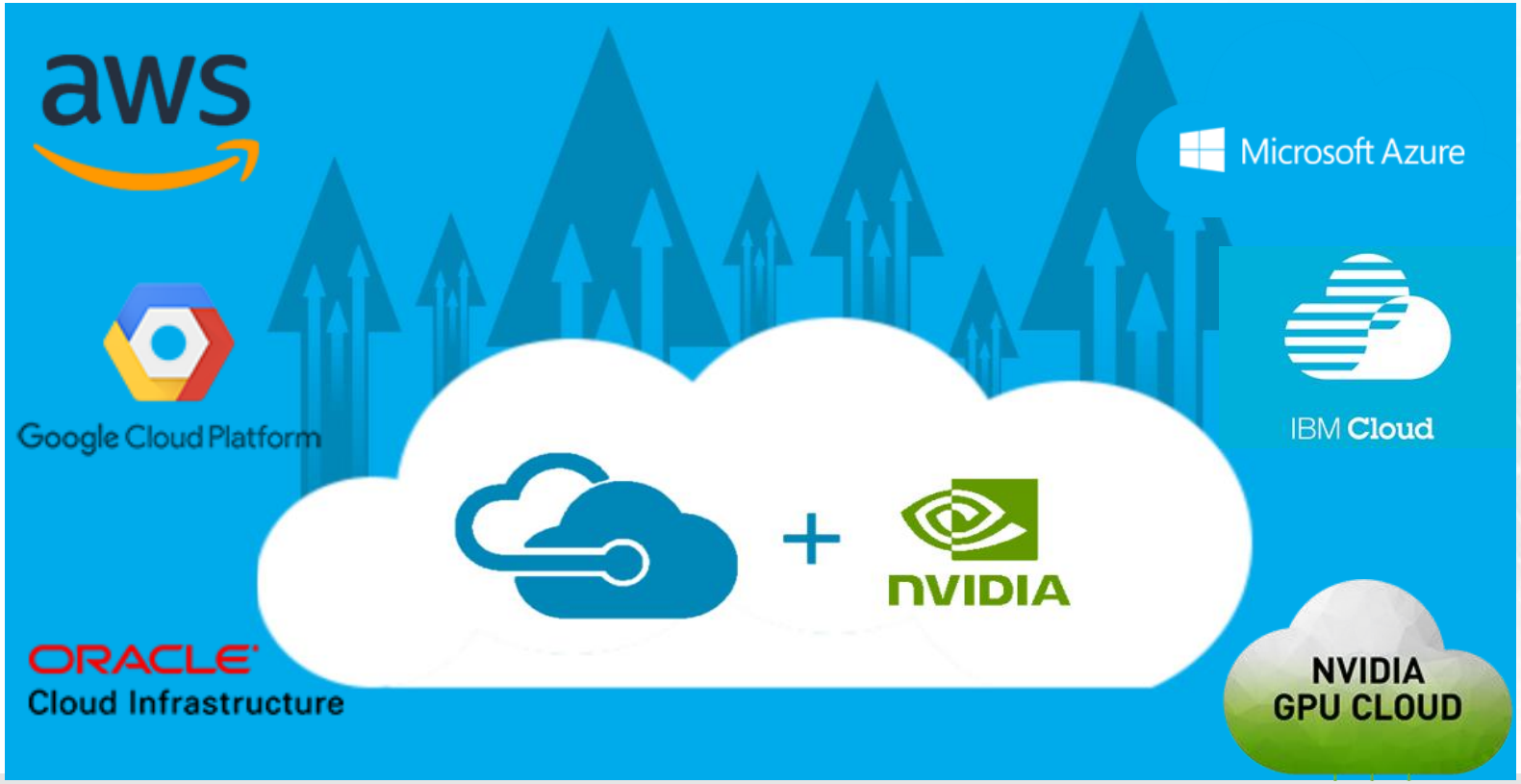
How to optimize performance when MPPDB shuffles data between nodes?

- Overlap data transfer with computation with enhanced pre-fetching
- Enable faster network protocol to lower network latency, bypass CPU



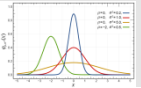
GPU on Cloud

How to deploy GPUs on cloud database?
How does data transfer overlap with computation in Cloud environment?
What about GPU clusters?



Take-Away

- **GPU is for massive data parallelism**
- **GPU can be used to speed up analytical queries**
- **Extendible acceleration framework requires:**
  - Resource-aware optimizer
  - Parallelized GPU kernel
  - Overlap data transfer with computation



Thank you!