Accelerated Data Management Systems Through Real-Time Specialization

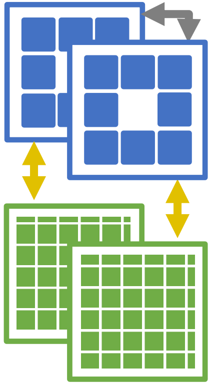
Anastasia Ailamaki

for Periklis Chrysogelos, Aunn Raza, Panos Sioulas, and the DIAS team

DATA-INTENSIVE APPLICATIONS AND SYSTEMS

EPFL

The game changers



Hardware

Workloads

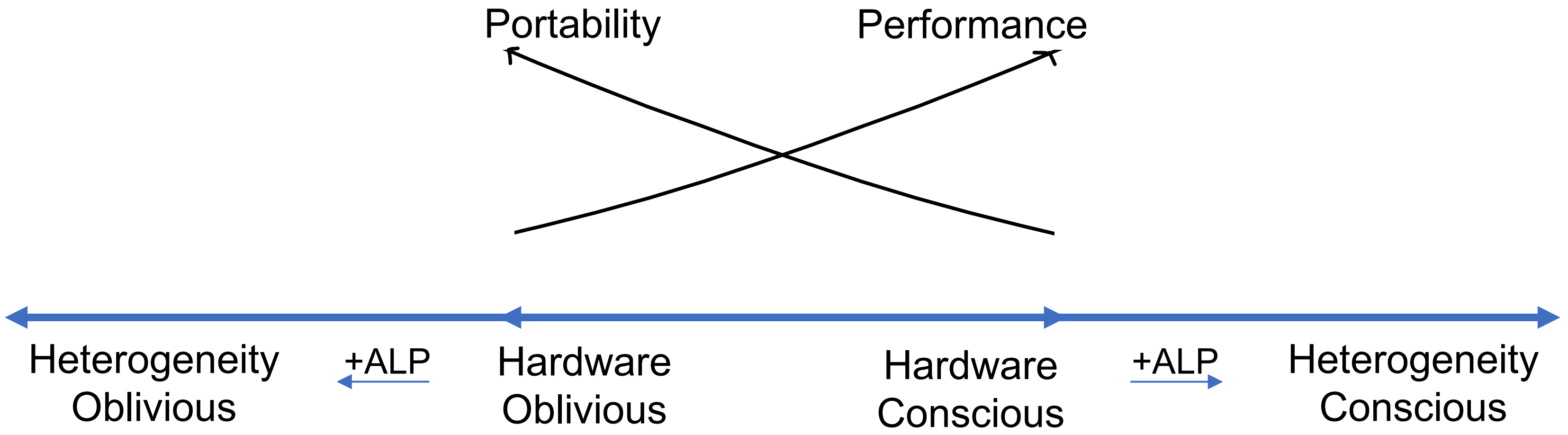
Heterogeneous and underutilized

Complex and unpredictable

Agile data management engines

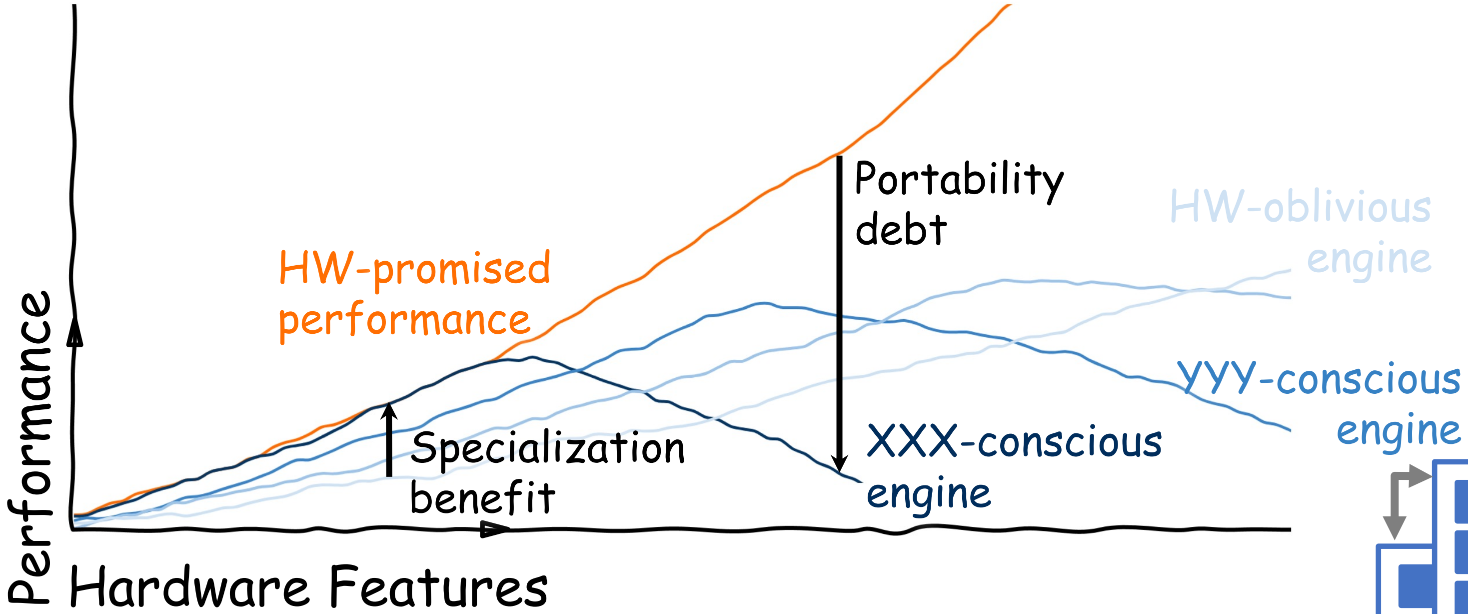
Portability

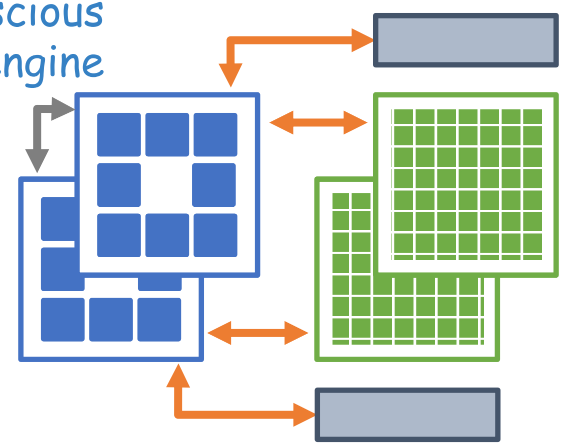
Performance



Where's the sweet spot?

Analytics on heterogeneous hardware

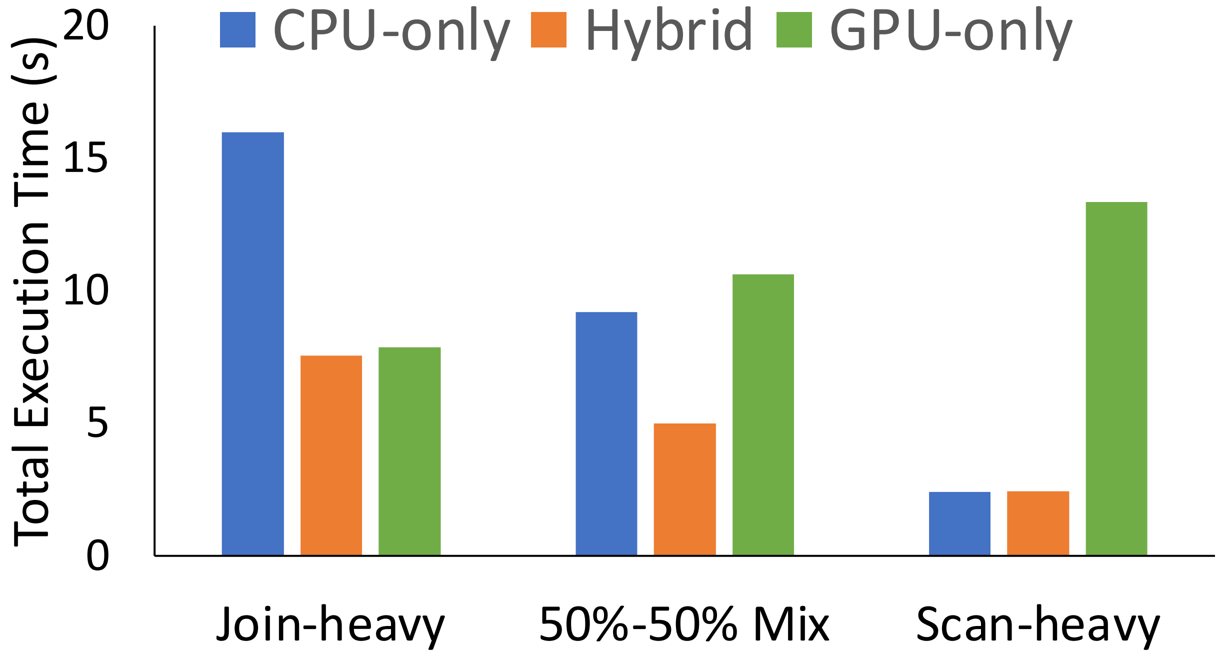




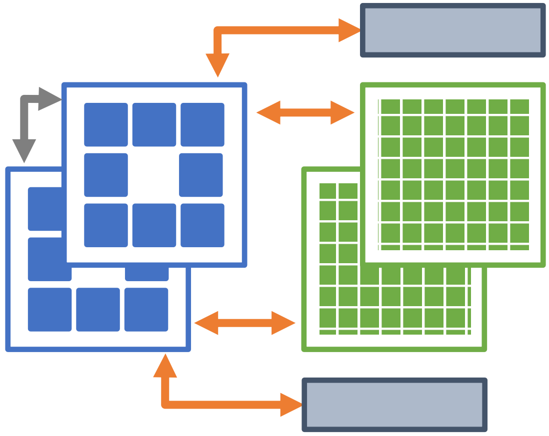
Device specialization carries portability debt

AiAS EPFL

Analytics on heterogeneous hardware



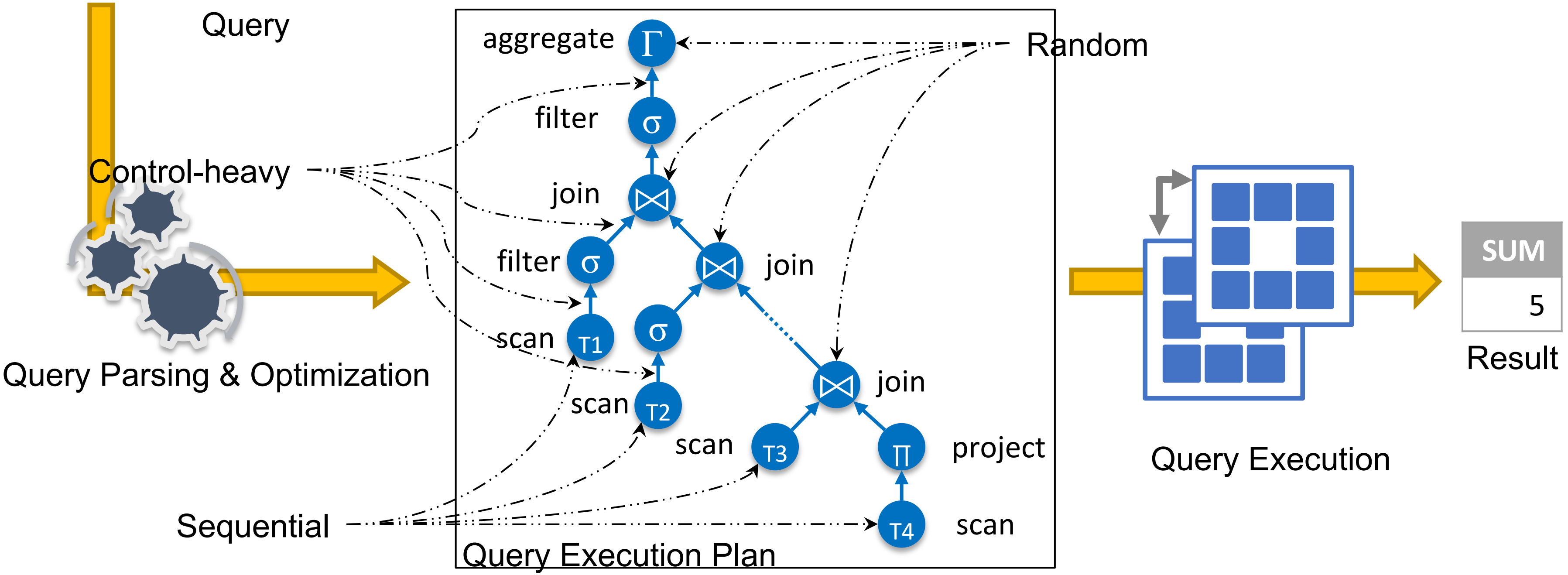
2x Intel(R) Xeon(R) Gold 5118 CPU
2x Mellanox MT27800 100G IB NIC
2x NVIDIA Tesla V100S GPU
13 Queries, CPU-resident data
96-144GB working set/query
Join-heavy: SPJ{1-4}Aggr
Scan-heavy: Scan-Aggr
4 servers



Device specialization carries portability debt

Lifetime of a Query

```
SELECT SUM(T3.c * T4.d)
FROM T1, T2, T3, T4, …
WHERE T1.f < 50 AND T1.a = T2.t1_id AND …
```



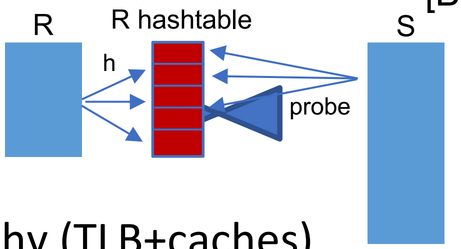
Hardware-conscious Analytics?

Traditional	CPU-optimized	Relies on
Random-access	radix-(join/group by)	High cache-size-to-thread ratio
Control-heavy	vector-at-a-time	High cache-size-to-thread ratio
T4 Sequential scan	parallelism/inter-socket atomics	Efficient inter-socket operations

Won’t work on GPUs

A fast equi-join algorithm

[Boncz et al. VLDB1999]



Radix-join

Partition both inputs

Size partition fanout based on memory hierarchy (TLB+caches)

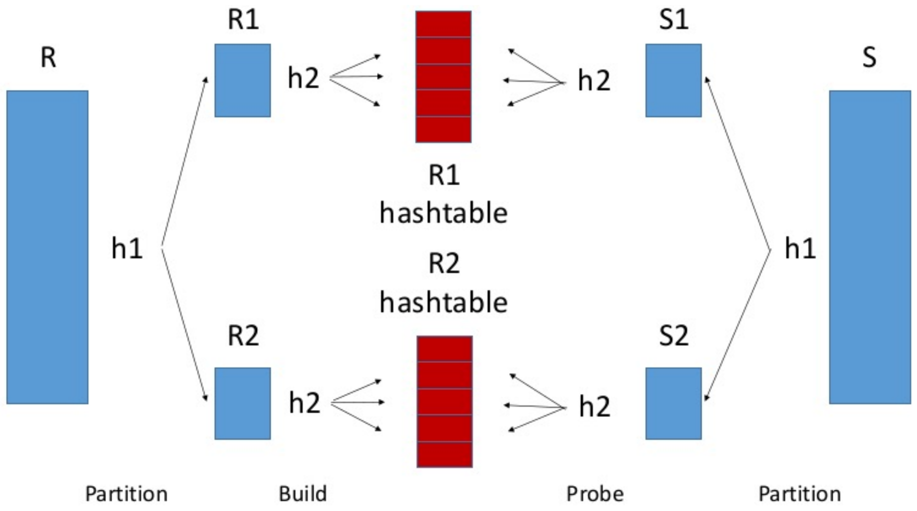
Assuming sufficient cache-to-thread ratio

GPU memory hierarchy

Low cache-to-thread ratio

Software and hardware-managed caches

But collaborative thread execution



Think differently for GPUs!

GPU-aware radix-join

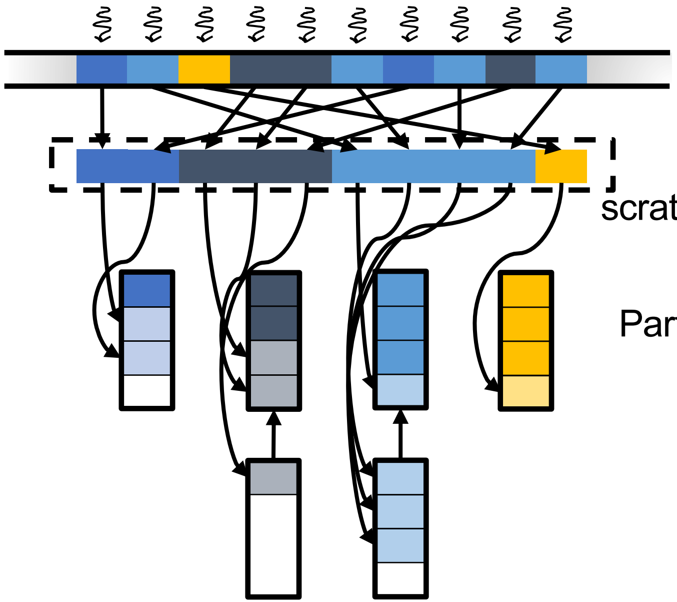
[ICDE2019]

Collaboratively partition per GPU thread block

- Amortize radix cluster maintenance
- Rely on big register files and thread overlapping
- Avoid random accesses to GPU memory

Stage partition output in scratchpad

- Irregular access patterns through scratchpad
- Coalesce writes through shared memory
- Multiple threads “complete” a cache line



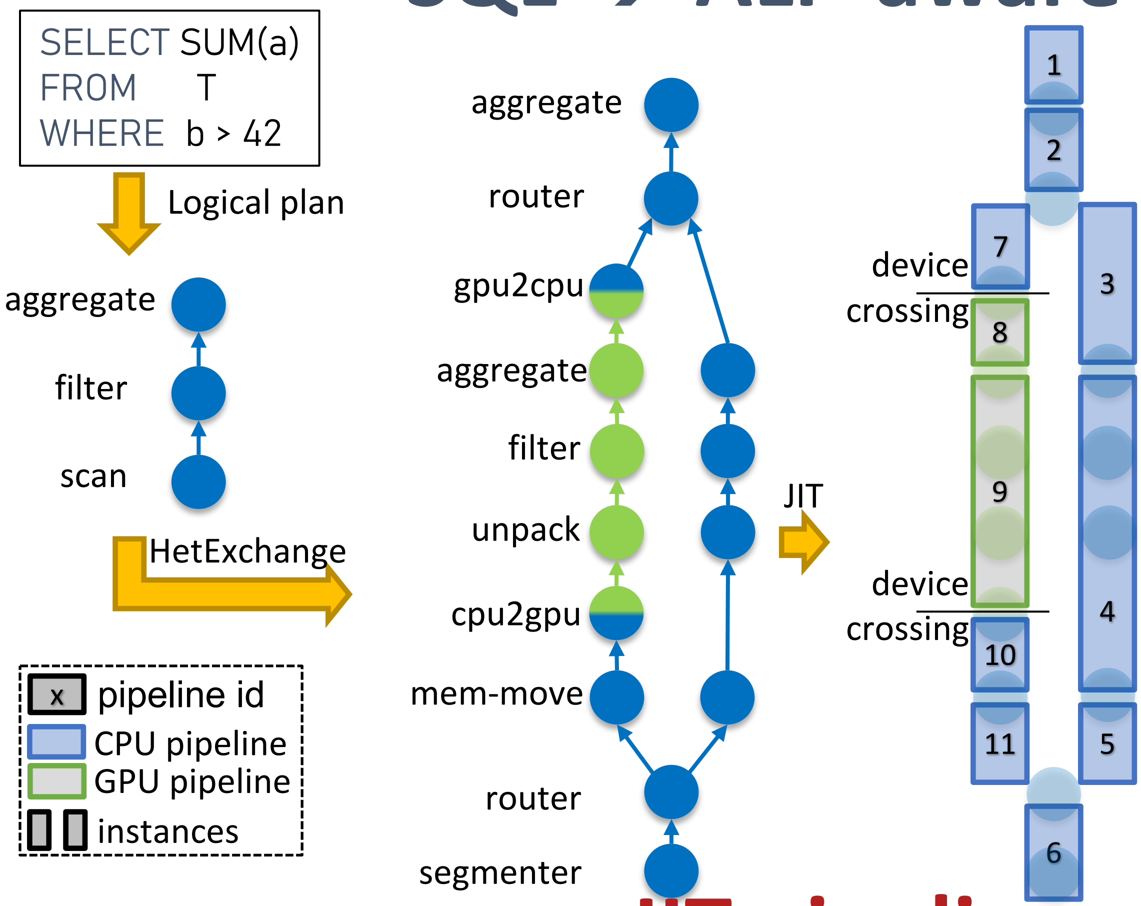
3.6x speedup

Accelerator-conscious Analytics

Traditional	CPU-optimized	CPU-GPU
⋈ Random-access	radix-(join/group by)	Tune operators to memory hierarchy specifics
Control-heavy	vector-at-a-time	Code fusion & specialization for fast composability
T4 Sequential scan	parallelism/inter-socket atomics	Encapsulate heterogeneity and balance load

SQL → ALP-aware code

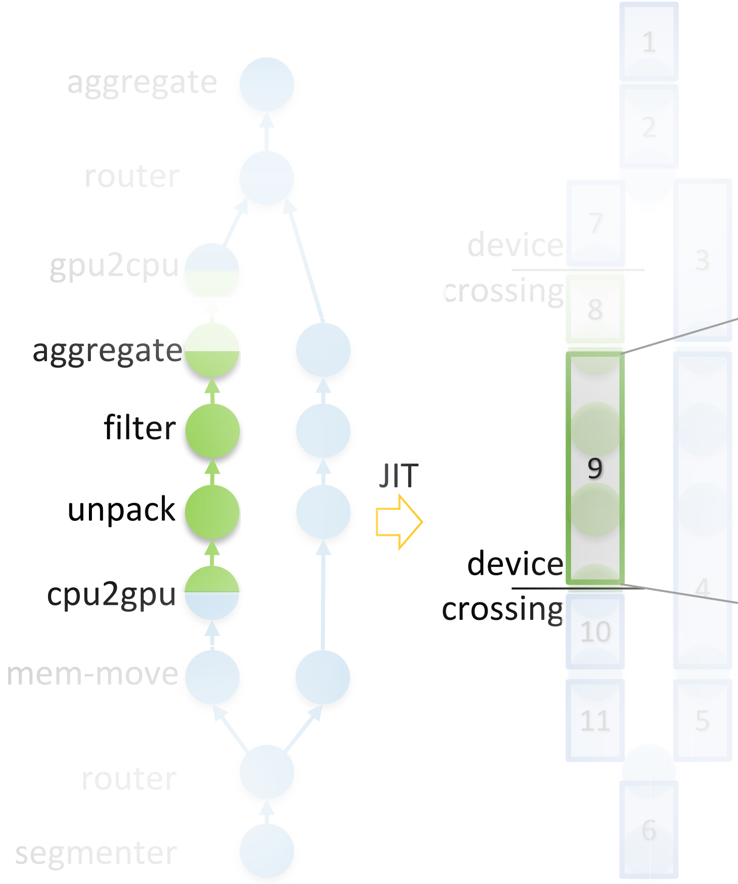
[VLDB2019]



JIT pipelines

JIT Code Generation for OLAP in GPUs

[VLDB2019]

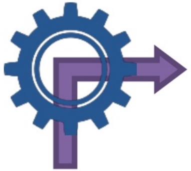



```
def unpack_filter_reduce(data_block, N,
    state)
  local_acc ← 0
  for i = threadIdInWorker to N − 1 with
      step #threadsInWorker
    t ← data_block[i]
    if t.a > 42
      local_acc ← local_acc + t.b
  nh_acc ← neighborhood_reduce(local_acc)
  if thread neighborhood leader
    atomic_add(state.acc, nh_acc)
```

Device providers

[VLDB2019]

CPU Provider

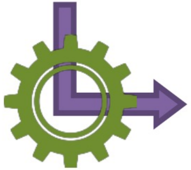


```
function ufr_cpu(data_block, N, state)
 local_acc ← 0
 for i = 0 to N − 1
   t ← data_block[i]
   if t.a > 42
     local_acc ← local_acc + t.b
 nh_acc ← local_acc
 state.acc ← state.acc + nh_acc
```

```
def unpack_filter_reduce(data_block, N,
    state)
  local_acc ← 0
  for i = threadIdInWorker to N − 1 with
      step #threadsInWorker
    t ← data_block[i]
    if t.a > 42
      local_acc ← local_acc + t.b
  nh_acc ← neighborhood_reduce(local_acc
  if thread neighborhood leader
    atomic_add(state.acc, nh_acc)
```

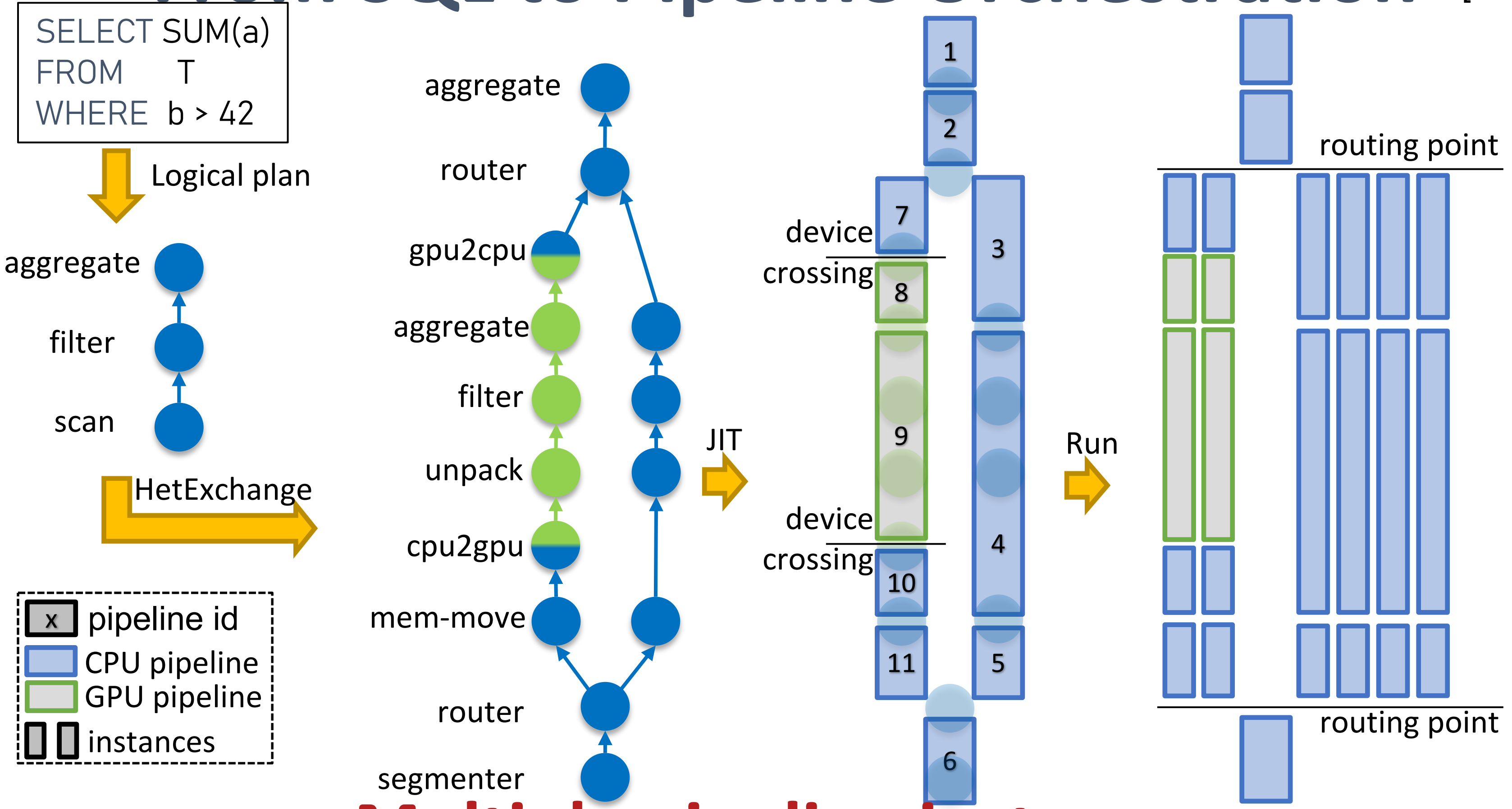
```
gpu_kernel ufr_gpu(data_block, N, state)
 local_acc ← 0
 for i = threadIdInGrid to N − 1 with
     step gridSize
   t ← data_block[i]
   if t.a > 42
     local_acc ← local_acc + t.b
 nh_acc ← threadblock_reduce(local_acc)
 if threadId = 0
   atomicAdd(state.acc, nh_acc)
```

GPU Provider



Inject target-specific info

From SQL to Pipeline Orchestration [VLDB2019]



Multiple pipeline instances

JIT data flow inspection

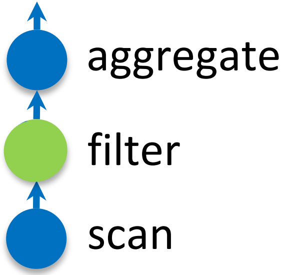
[VLDB2019]

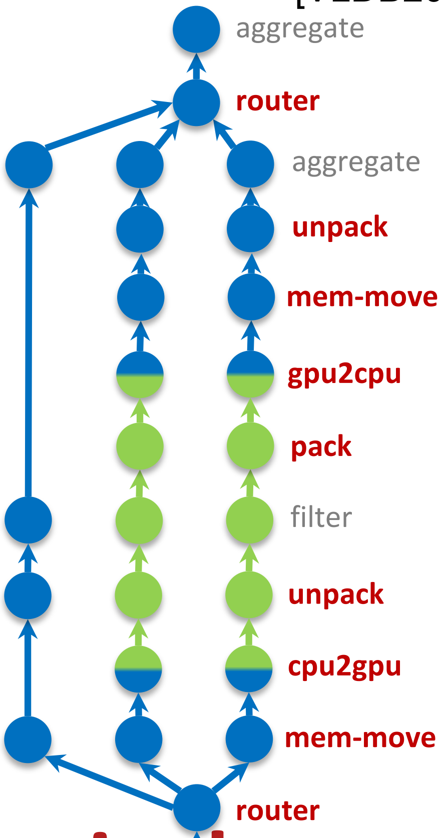
Decouple data- from control-flow

Encapsulate trait conversions into operators

Inspect flows to load-balance

Flow	Scope	Trait
Control	Delegation	Heterogeneous Parallelism
	Routing	Homogeneous Parallelism
Data	Transfer	Data Locality
	Granularity	Execution Granularity

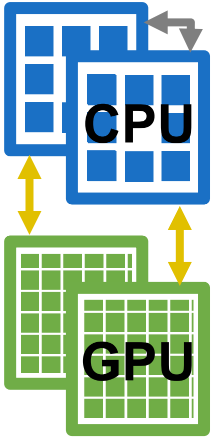


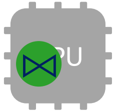


Distribute load to devices adaptively

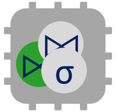
Abstractions for fast CPU-GPU analytics

[CIDR2019]

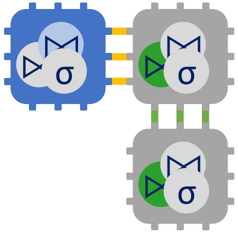


intra-operator 

- Operator tuning is µ-architecture specific
- Tune operators to memory hierarchy specifics

intra-device 

- Portability clashes with specialization
- Inject target-specific info using codegen



inter-device

- Limited device inter-operability
- Encapsulate heterogeneity and balance load

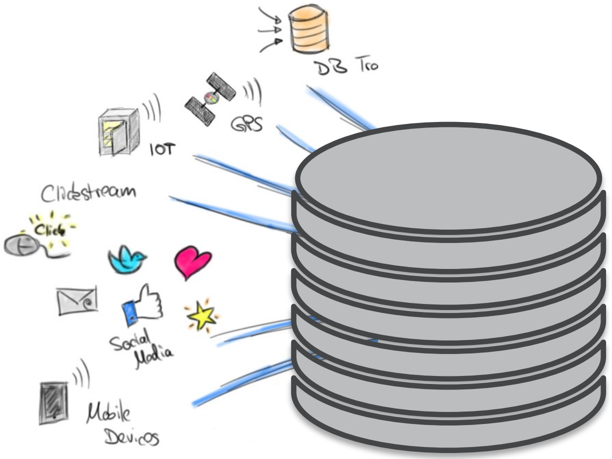
Selective obliviousness

The game changers

Heterogeneous and underutilized

Complex and unpredictable

Specialized OLTP & OLAP Systems



Fresh Data

Transactions

OLTP

Extract-Transform-Load

Analytics

OLAP

Data freshness bounded by ETL latency

Hybrid Transactional and Analytical Processing

OLTP: task-parallel 

– High rate of short-lived transactions
– Mostly point accesses (high data access locality)

OLAP: data-parallel 

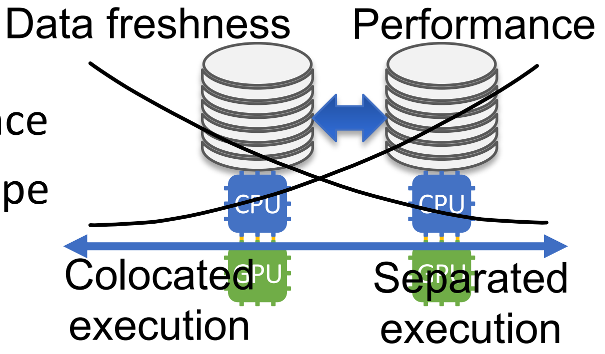
– Few, but long-running queries
– Scans large parts of database

Align tasks & hardware to improve utilization

HTAP: Chasing 'locality of freshness'

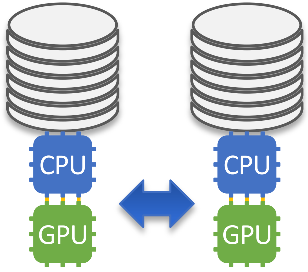
Static OLAP-OLTP assignment

– Unnecessary tradeoff between interference and performance
– Pre-determined resource assignment based on workload type
– Wasteful data consolidation and synchronization



Real-time, Adaptive scheduling of HTAP workloads

– Specialize to requirements and data/freshness-rates
– Workload-based resource assignment
– Pay-as-you-go snapshot updates



Task placement based on resource usage

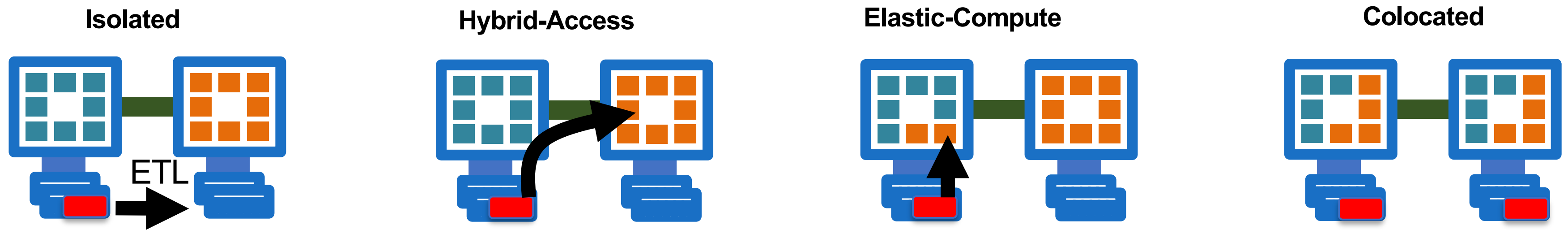
Workload Isolation & Fresh Data Throughput

OLTP OLAP

Fresh Data

Interference ←→ performance

Pre-determined resource assignment

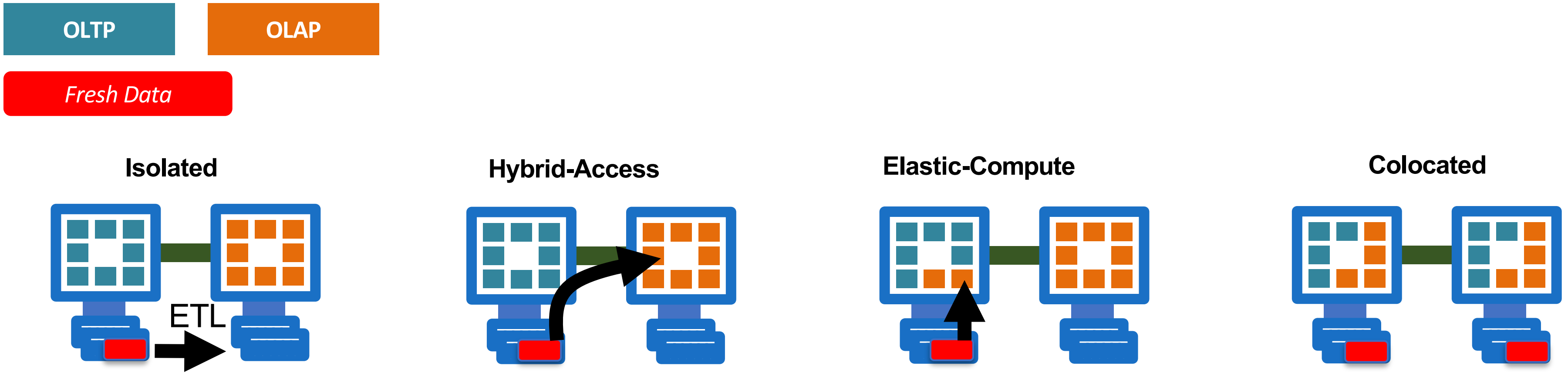


Fresh Data Access Bandwidth →

← Independent execution (isolation)

no extreme is good

Workload Isolation & Fresh Data



Real-time: Adaptive scheduling of HTAP workloads

- Specialize to requirements and amount of unconsolidated data
- Workload-based resource assignment
- Pay-as-you-go snapshot updates

Task placement & consolidation based on

Caldera: HTAP on CPU-GPU Servers

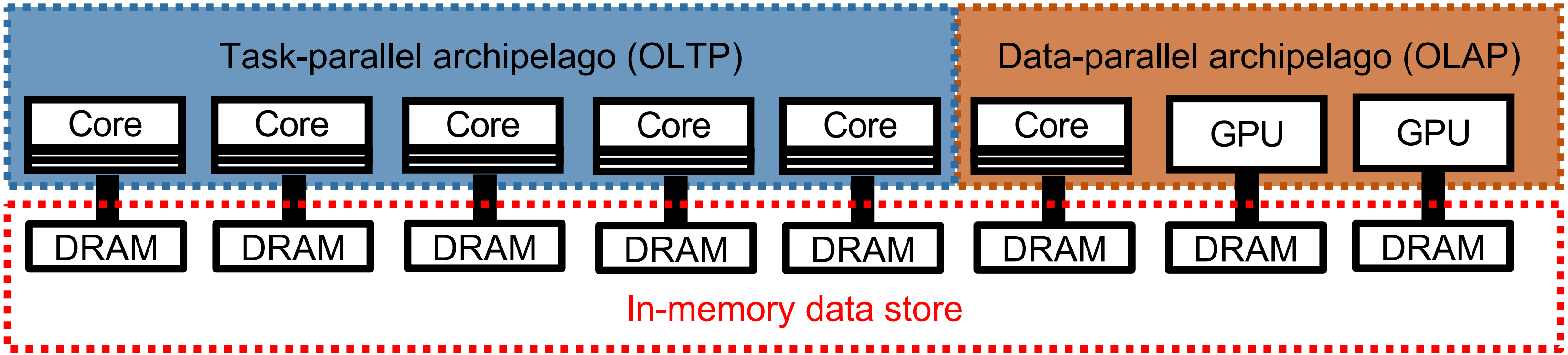
[CIDR2017]

Store data in shared memory

Run OLTP workloads on task-parallel processors

Run OLAP workloads on data-parallel processors

– On-demand copy-on-write snapshots in shared memory



Adaptively scaling resources with load

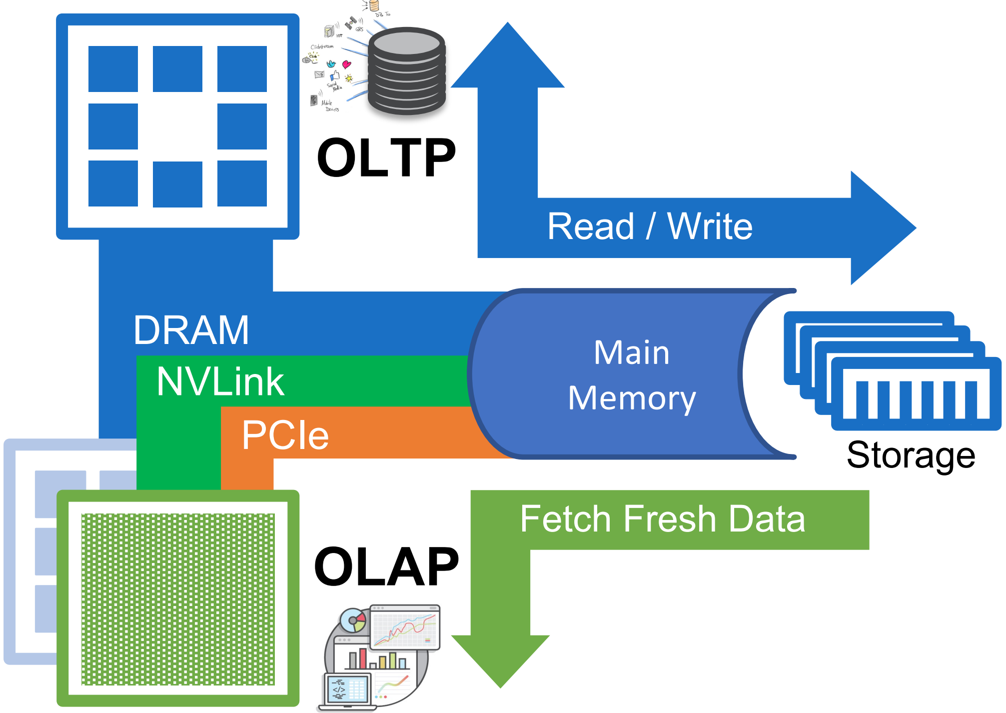
GPU Accesses Fresh Data from CPU Memory

[CIDR2017, CIDR2020, SIGMOD2020]

OLTP generates fresh data on CPU Memory

Data access protected by concurrency control

OLAP needs to access fresh data



snapshot isolation for OLAP w/o CC overheads

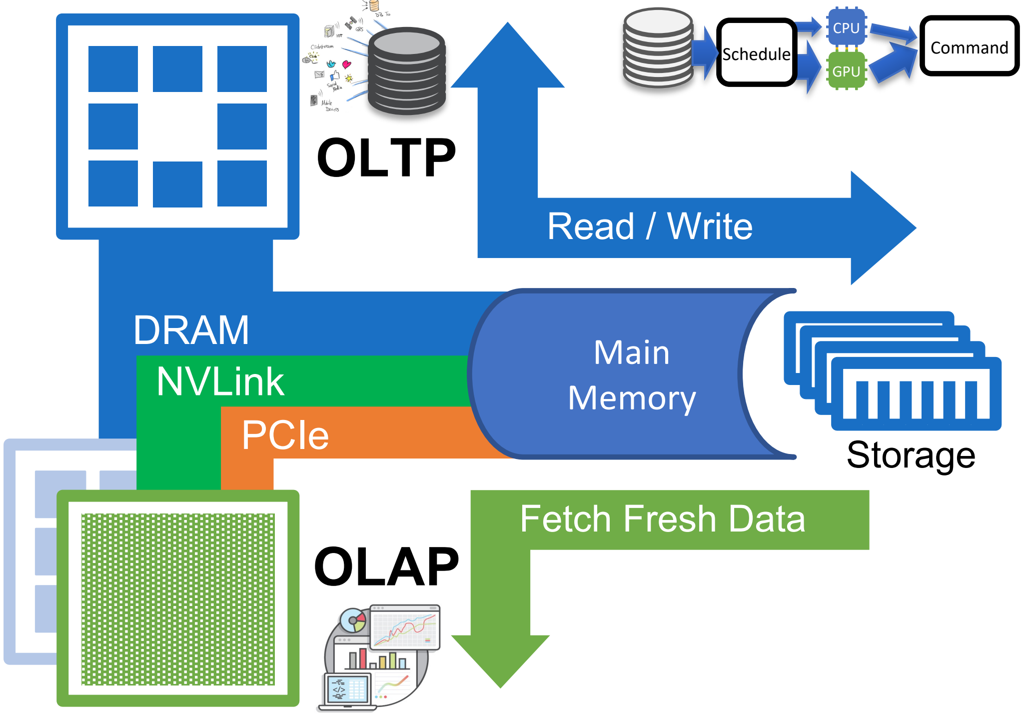
GPU Accesses Fresh Data from CPU Memory

[CIDR2017, CIDR2020, SIGMOD2020]

OLTP generates fresh data on CPU Memory

Data access protected by concurrency control

OLAP needs to access fresh data



snapshot isolation for OLAP w/o CC overheads

Increasing workload complexity

Diverse modern data problems

– IOT, OCR, ML, NLP, Medical, Mathematics etc…

Commercial AI/ML

DBMS catch-up for popular functionality

– Human effort and big delays

– Oblivious to out-of-DBMS workflows

Augmented analytics

Vast resource of libraries

– Authored by domain experts, used by everybody

– Loose library-to-data-sources integration and optimization

Conversational analytics and NLP

Combination of IoT and analytics

Need for systems that can "learn" new functionality

Network looks like a single machine

Similar intra-/inter-server interconnect bandwidth

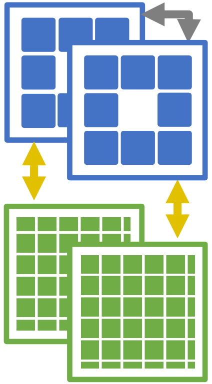
Local memories and NUMA effects across devices

CPU-GPU: Capacity-Throughput

Heterogeneous interconnected devices across

Intelligent Real-time Systems

Hardware



Workloads

A solution is only as efficient as its least adaptive component.