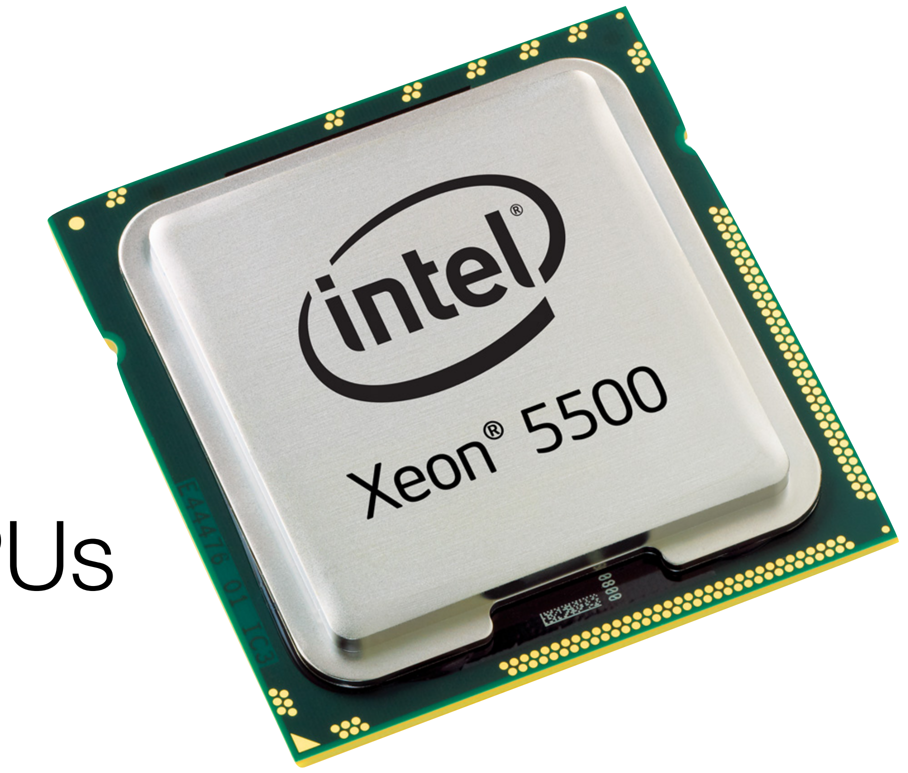
Query Processing on Heterogeneous Systems

Viktor Rosenfeld

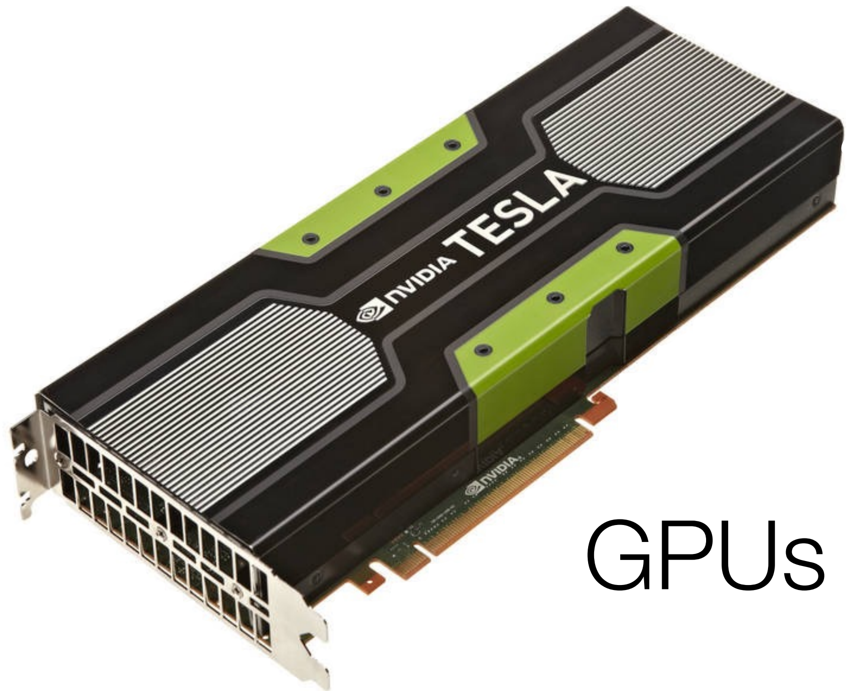
8 December 2023

Today's computing systems are highly heterogeneous.

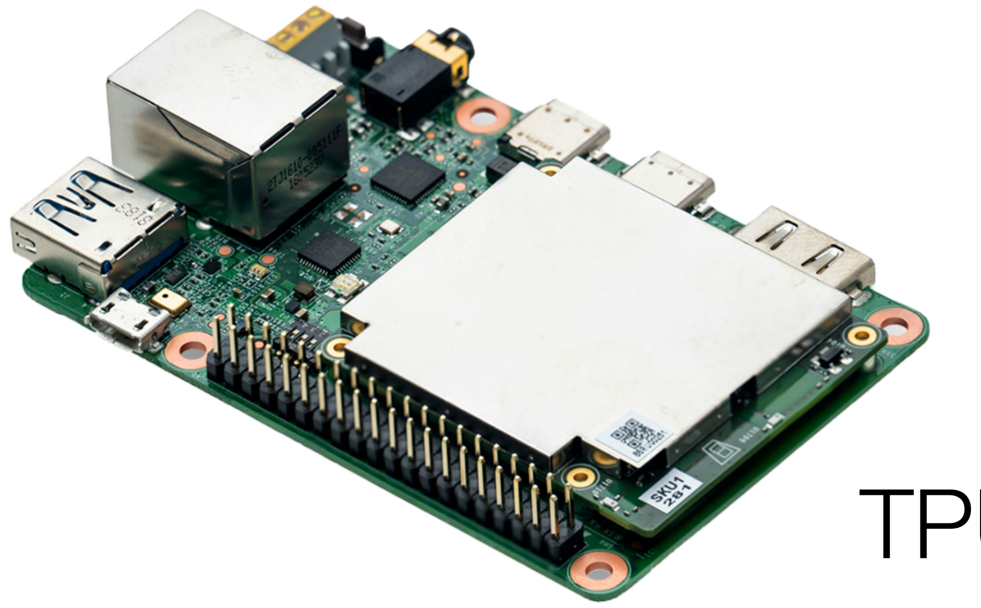
Specialized processors



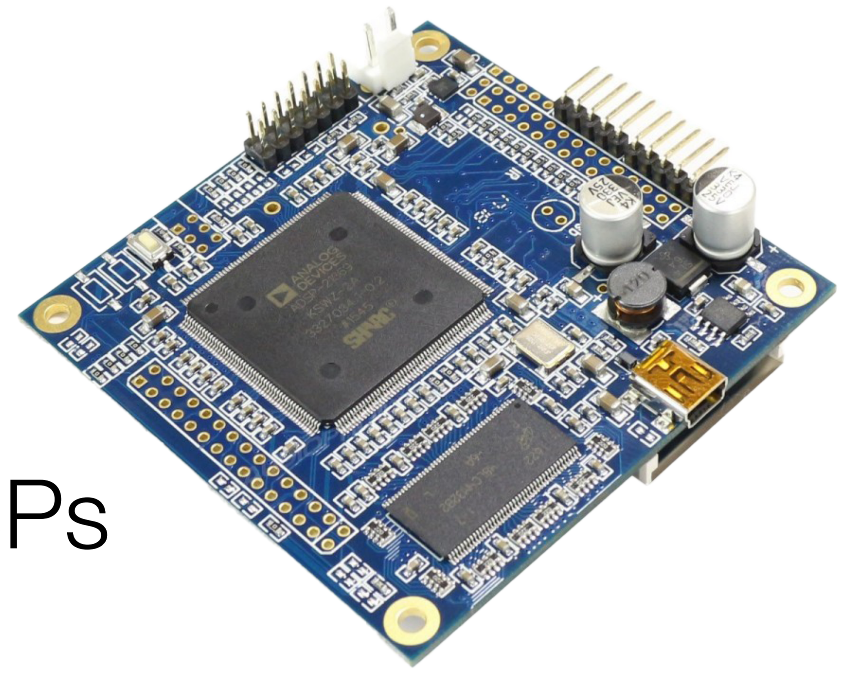
CPUs



GPUs



TPUs

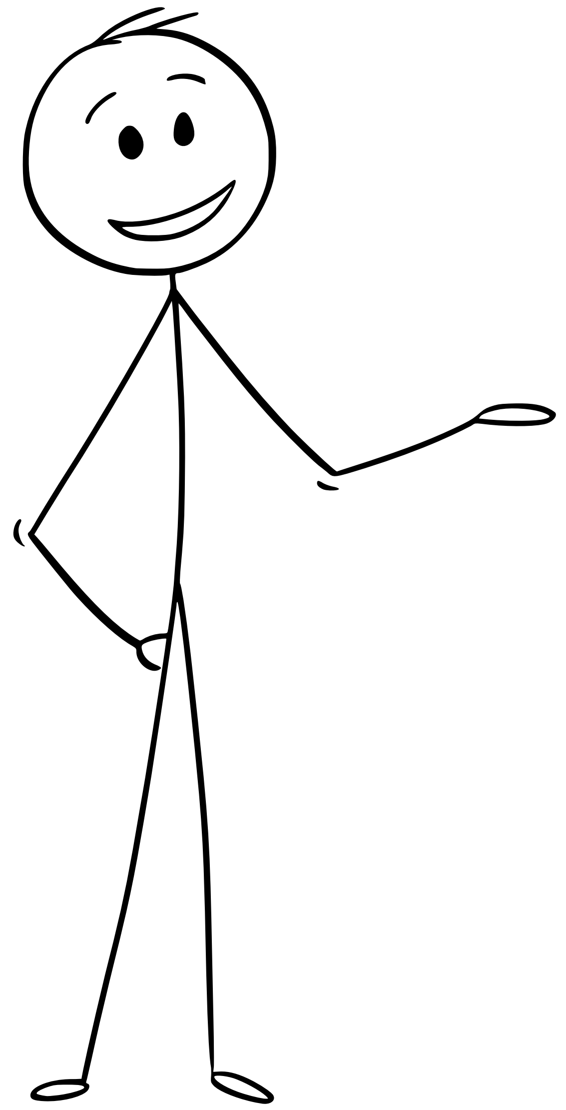


DSPs

FPGAs

Processors are optimized for different application scenarios to overcome the power wall.

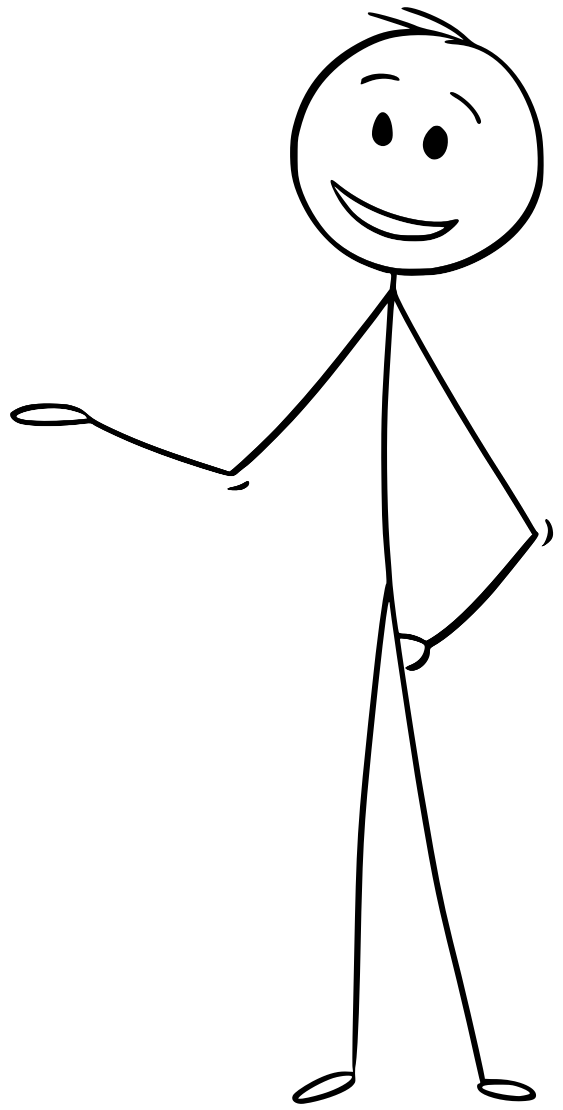
Diverse users requirements



Java

NVIDIA CUDA

Professional programmers



scikit learn

TensorFlow

Casual programmers

Programming languages have to meet users' skill set and performance requirements.

Complex data processing pipelines

Collection → Cleaning → Storage → Analysis → Querying

kafka IBM Db2 DuckDB Flink Apache Spark

Complex software ecosystems integrate many specialized tools to solve data processing tasks.

Thesis statement and goals

Heterogeneity is a benefit.

It enables the tools that allow different users to process large data sets efficiently.

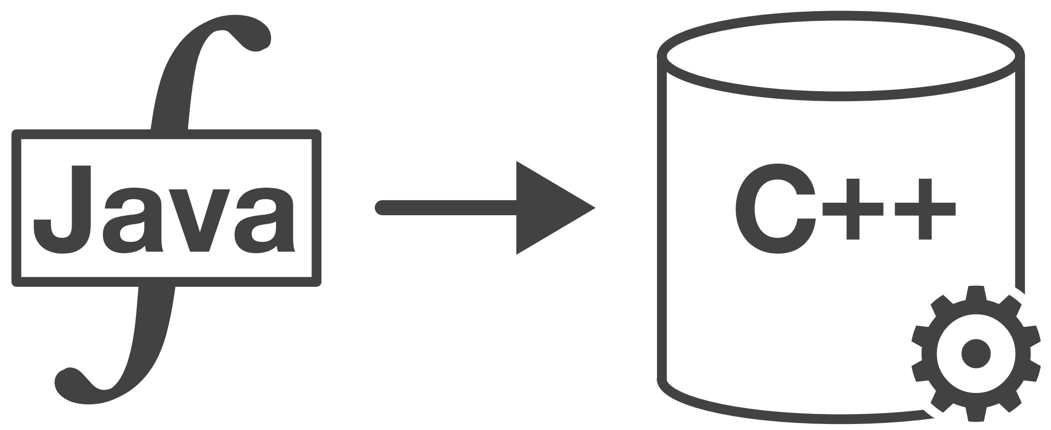
Heterogeneity is a challenge.

It increases the complexity of the computing infrastructure.

Investigate how heterogeneous hardware and software impact query processing systems.

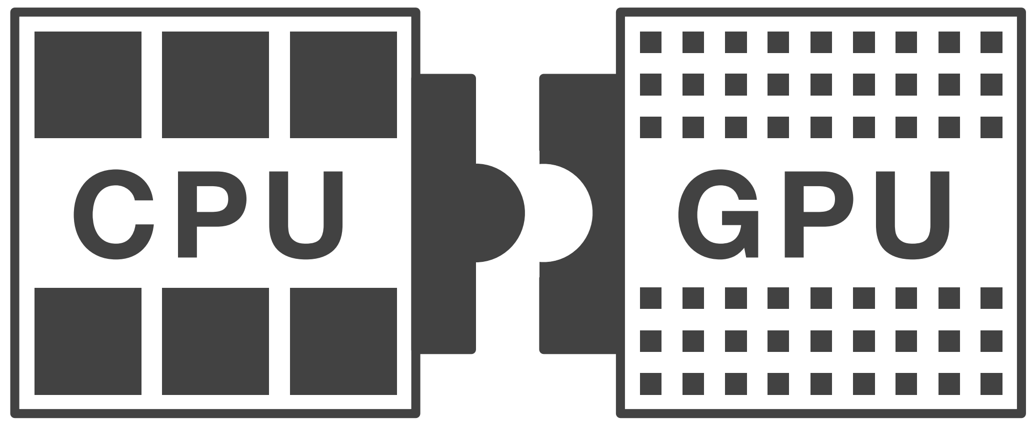
Optimize query processing performance on heterogeneous hardware and software systems.

Three scenarios



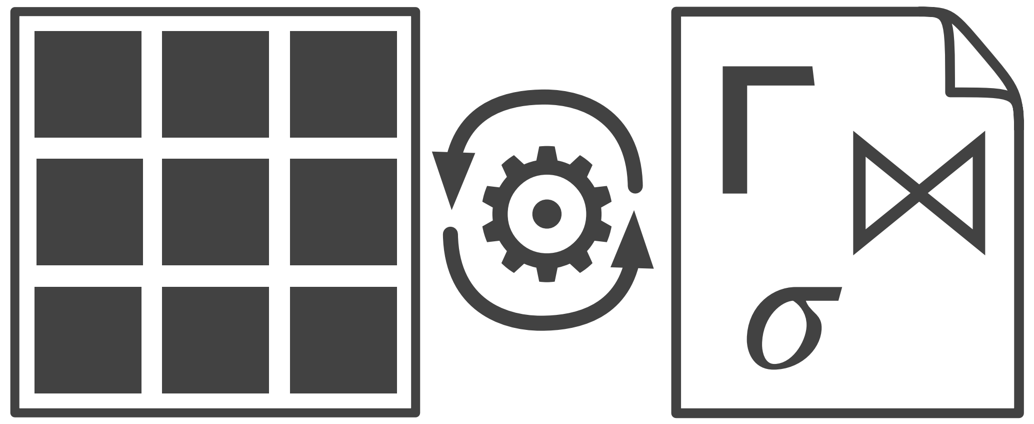
Processing Java UDFs
in a C++ Environment

ACM SoCC 2019



Query Processing on
Heterogeneous CPU/GPU Systems

ACM CSUR 2022

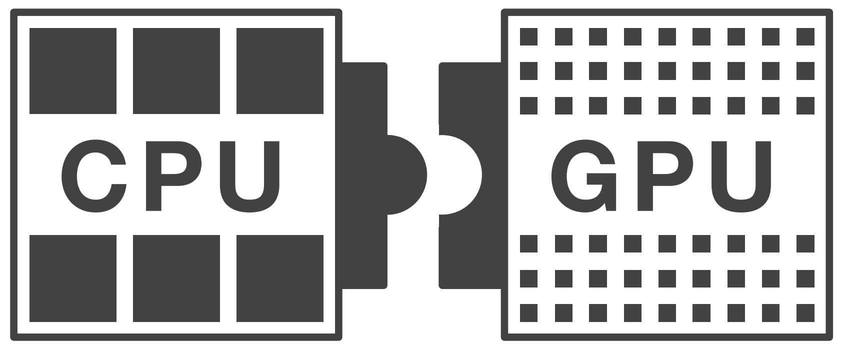


Operator Variant Tuning on
Heterogeneous Processors

ADMS 2015, DaMoN 2019

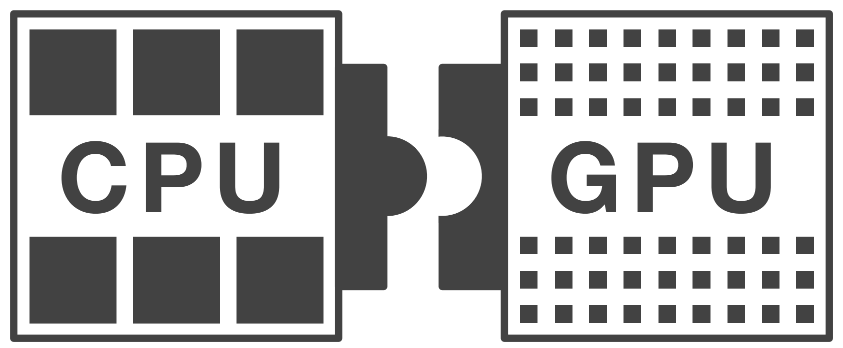
Common challenges posed by heterogeneous hardware and software

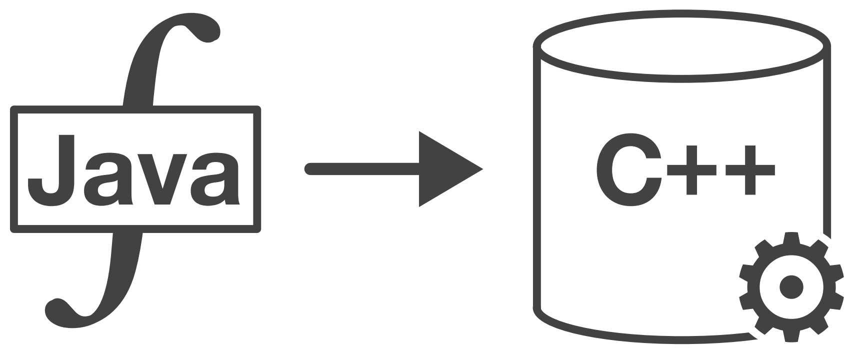
Distributing computation



Cross-platform data processing

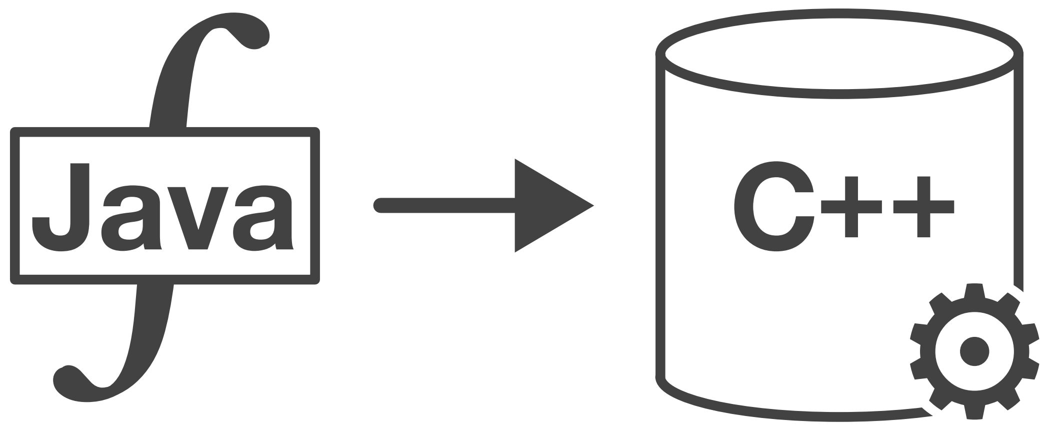
Reducing data movement





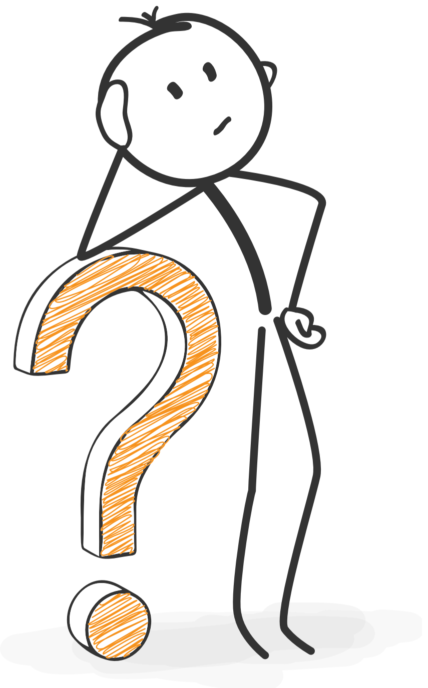
Cross-platform data processing

Techniques to address the heterogeneity of today's computing systems are applicable to a wide range of research areas and engineering tasks in query processing.



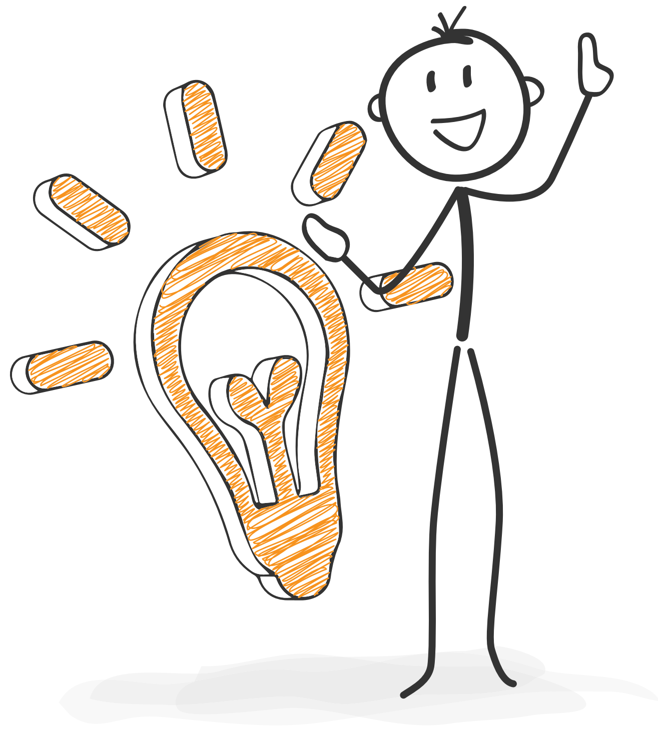
Processing Java UDFs in a C++ Environment

Extending the Apache Spark ecosystem



Add transactional processing

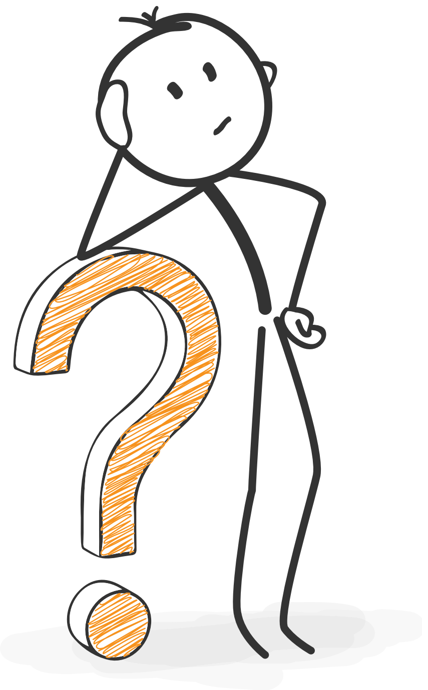
Speed up analytical processing



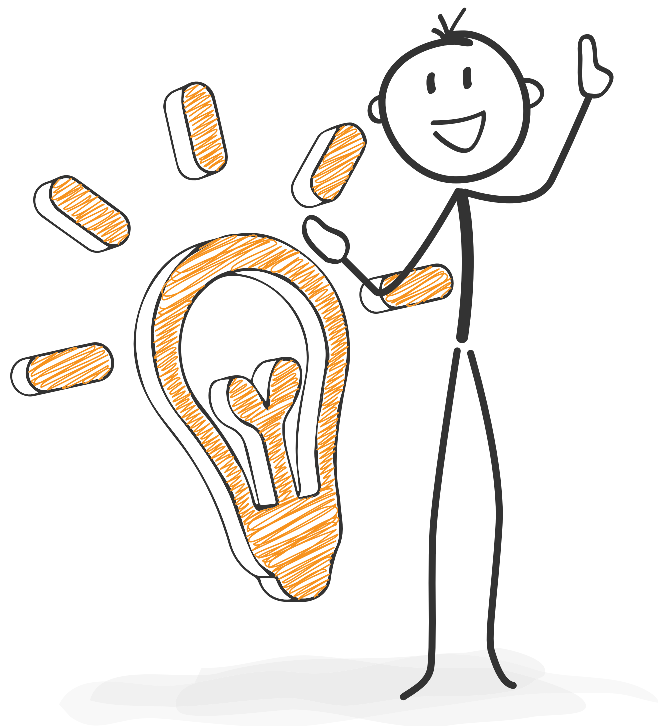
Keep Spark SQL as user frontend

Replace Spark processing engine with a C++ engine

IBM Wildfire

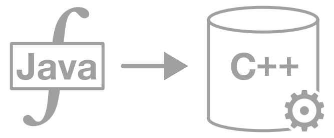


Spark SQL queries can contain arbitrary Java/Scala UDFs

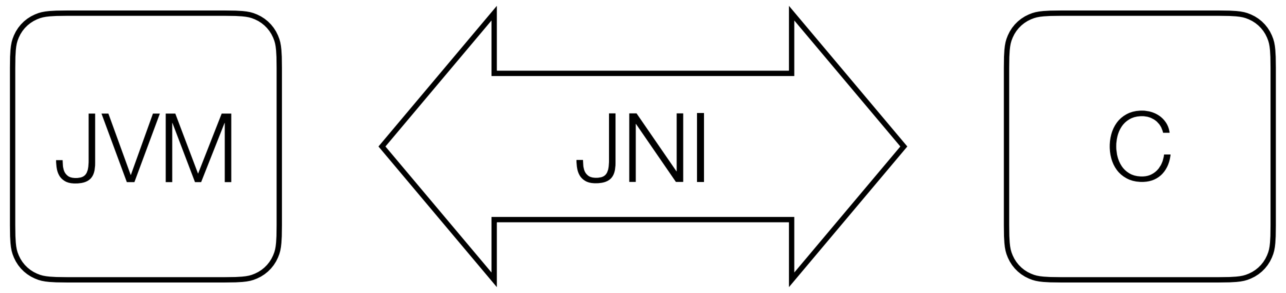


(1) Execute Java UDFs inside embedded JVM

(2) Compile Java UDFs to machine code



Java Native Interface (JNI)



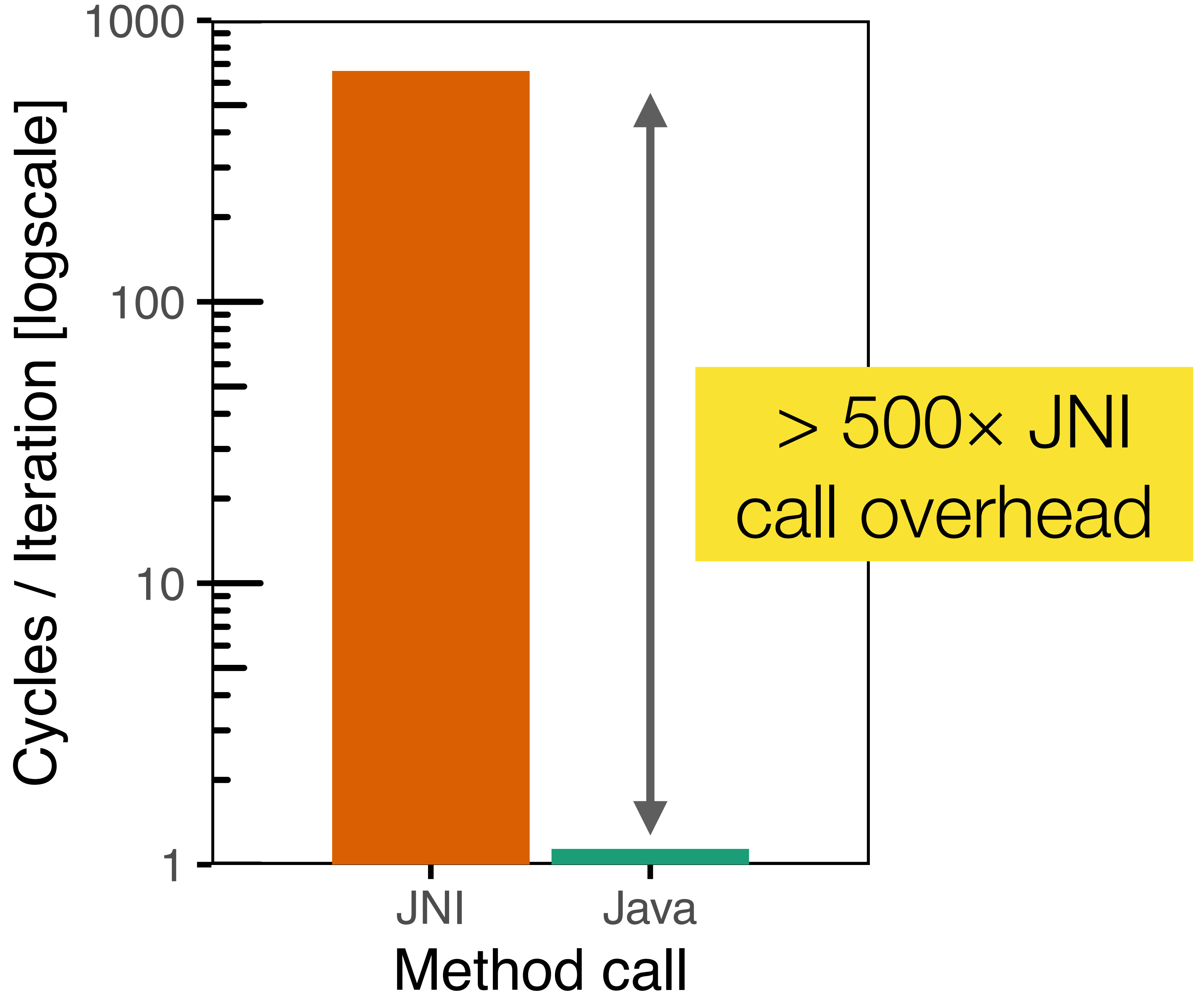
Java methods can be implemented in C

C programs can instantiate a JVM

APIs to manipulate Java objects
and to call Java methods

`(i: Int) => i`

10^9 interations

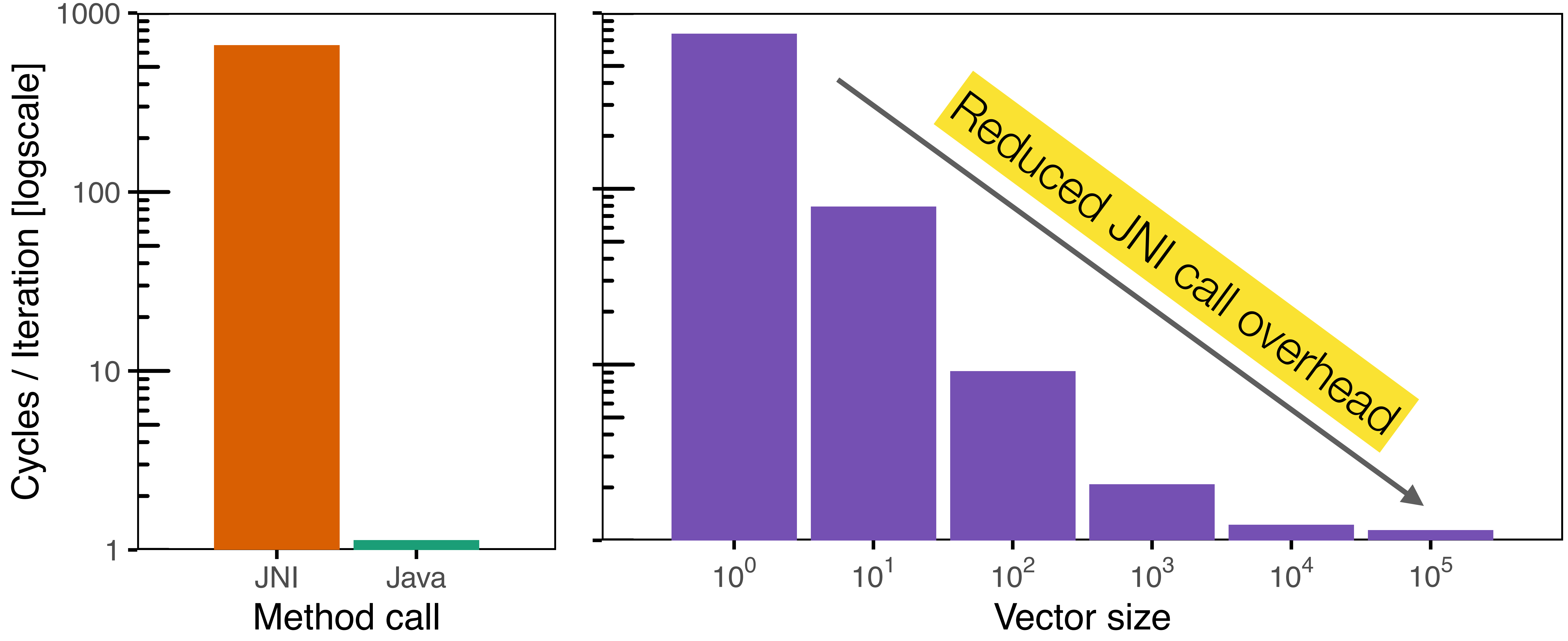


JNI call for every tuple has significant overhead.

Vectorized execution

`(i: Int) => i`

10^9 interations



Moving part of the loop inside the JVM eliminates JNI call overhead.

Vectorized execution in embedded JVM

```
output = udf.apply(input0, ..., inputN);
```

Vectorized execution in embedded JVM

```
public class StridedExecutionWrapper {

  public static void executeUdf(UdfClass udf,
                                int numRows,
                                ByteBuffer output,
                                ByteBuffer[] inputs) {

    for (int i = 0; i < numRows; ++i) {
      output.putX(udf.apply(inputs[0].getX(), ..., inputs[N].getX()));
    }

  }
}
```

Vectorized execution in embedded JVM

```
public class StridedExecutionWrapper {

  public static void executeUdf(UdfClass udf,
                                int numRows,
                                ByteBuffer output,
                                ByteBuffer[] inputs) {

    for (int i = 0; i < numRows; ++i) {
      output.putX(udf.apply(inputs[0].getX(), ..., inputs[N].getX()));
    }

  }
}
```

Vectorized execution

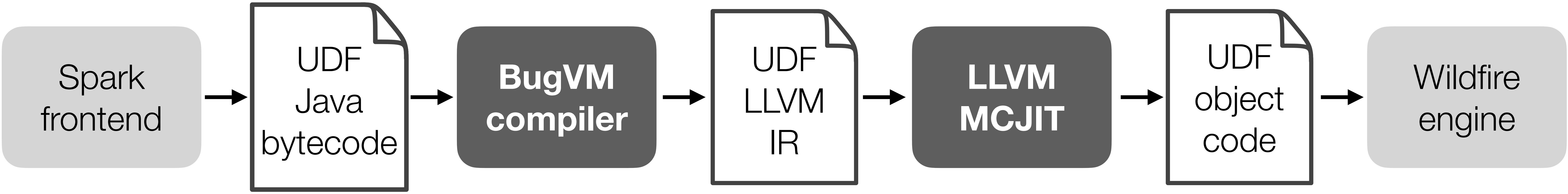
Vectorized execution in embedded JVM

```
public class StridedExecutionWrapper {

  public static void executeUdf(UdfClass udf,
                                int numRows,
                                ByteBuffer output,
                                ByteBuffer[] inputs) {

    for (int i = 0; i < numRows; ++i) {
      output.putX(udf.apply(inputs[0].getX(), ..., inputs[N].getX()));
    }

  }
}
```

Java Direct ByteBuffers reduce data copies

Vectorized execution

JIT compilation to machine code

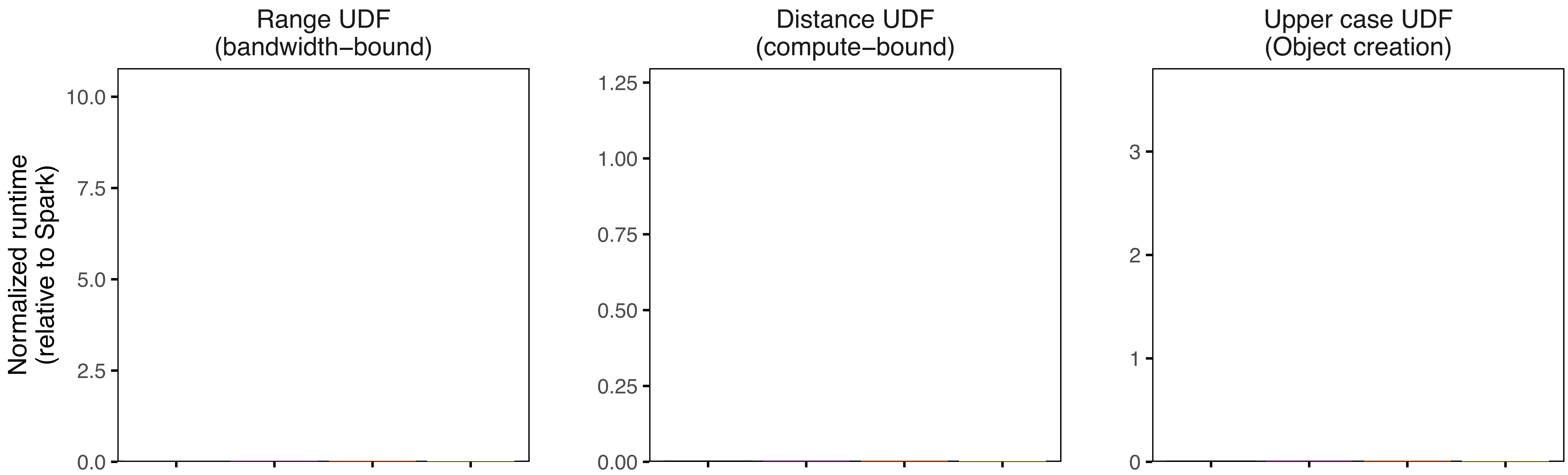


✔ No JNI call overhead

❌ No HotSpot VM optimizations

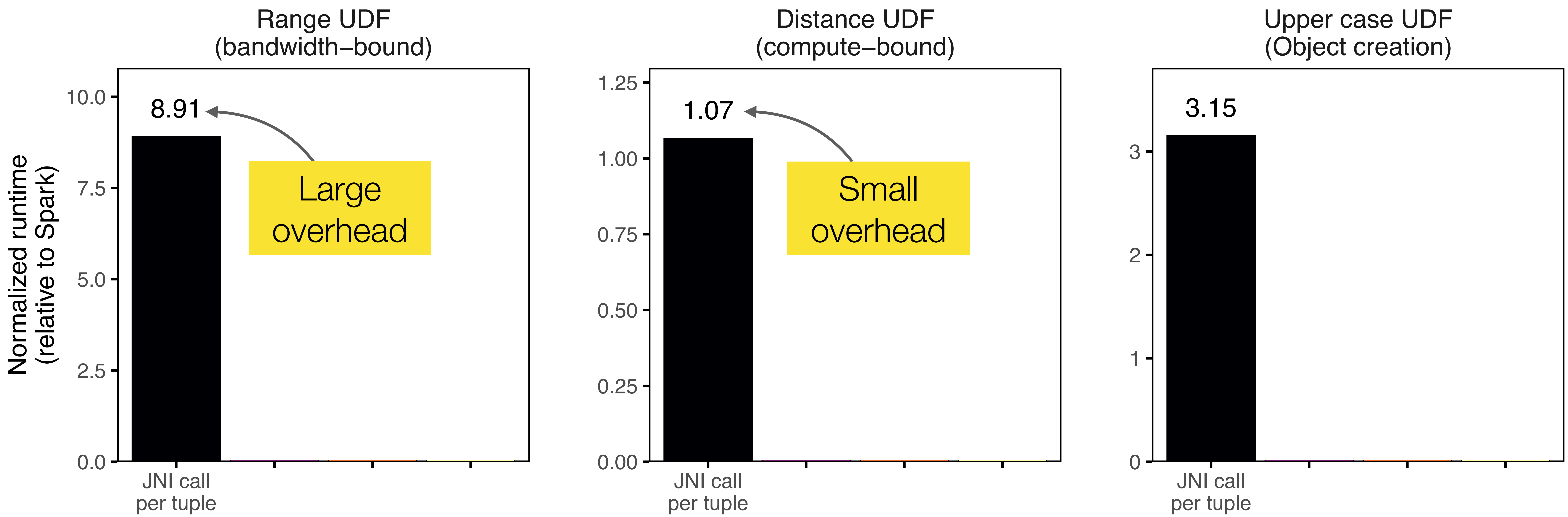
Evaluation

Runtimes relative to execution in Spark, different scales on y axes!



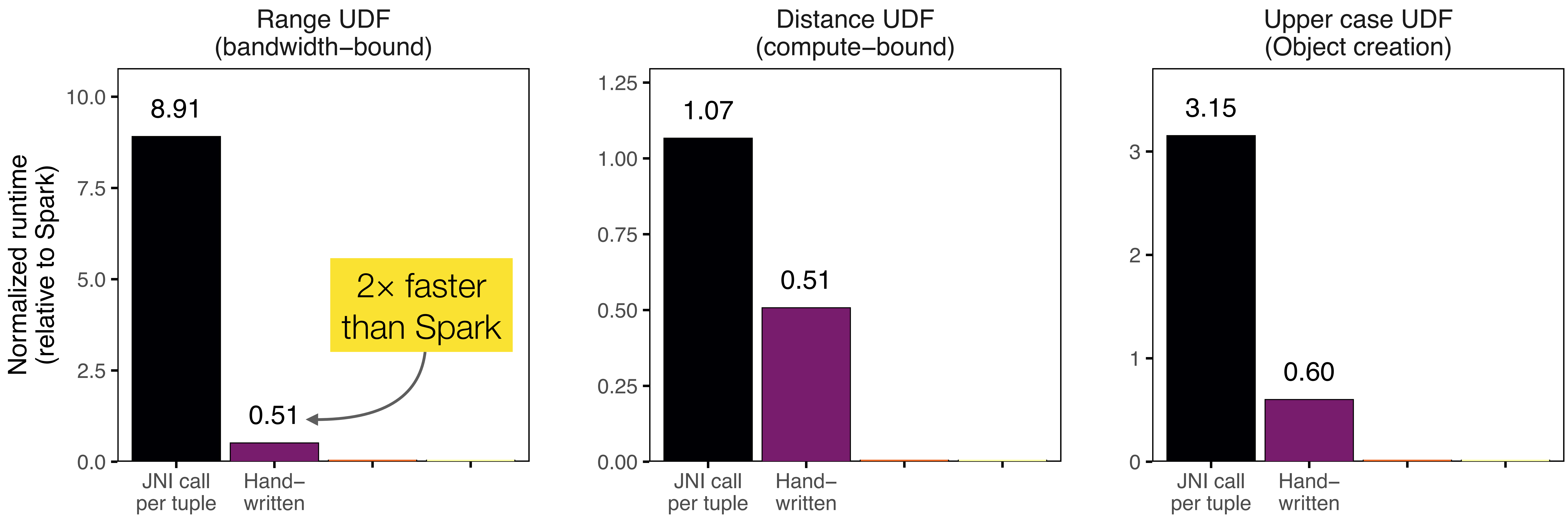
Evaluation

Runtimes relative to execution in Spark, different scales on y axes!



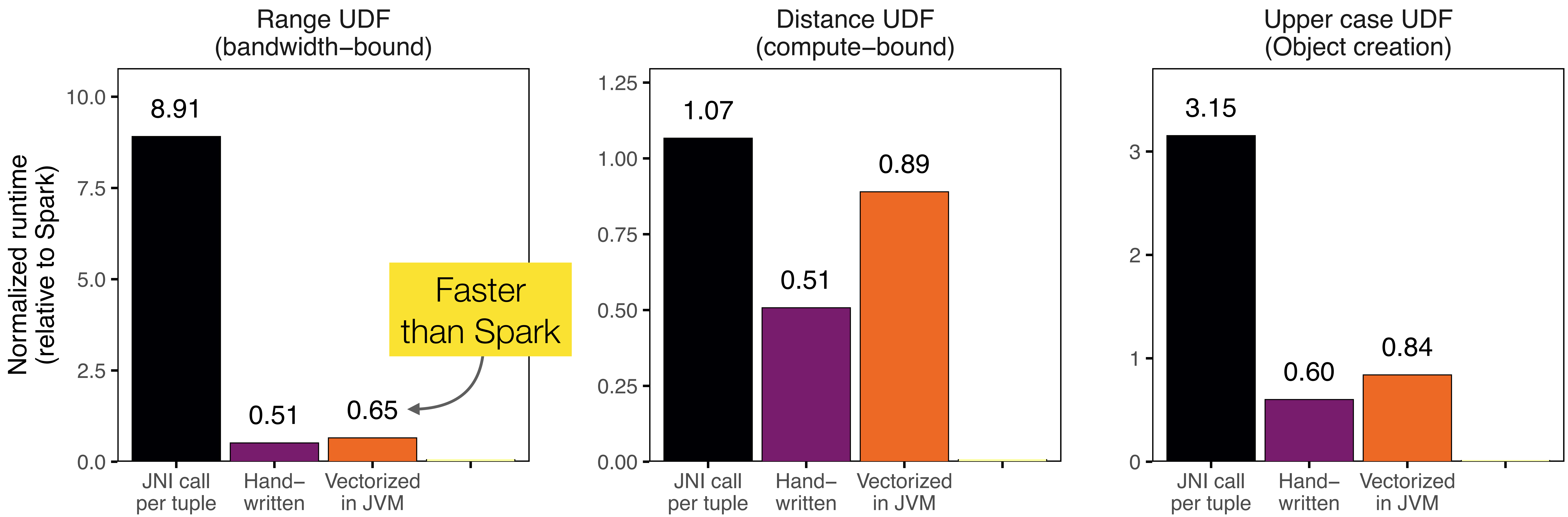
Evaluation

Runtimes relative to execution in Spark, different scales on y axes!



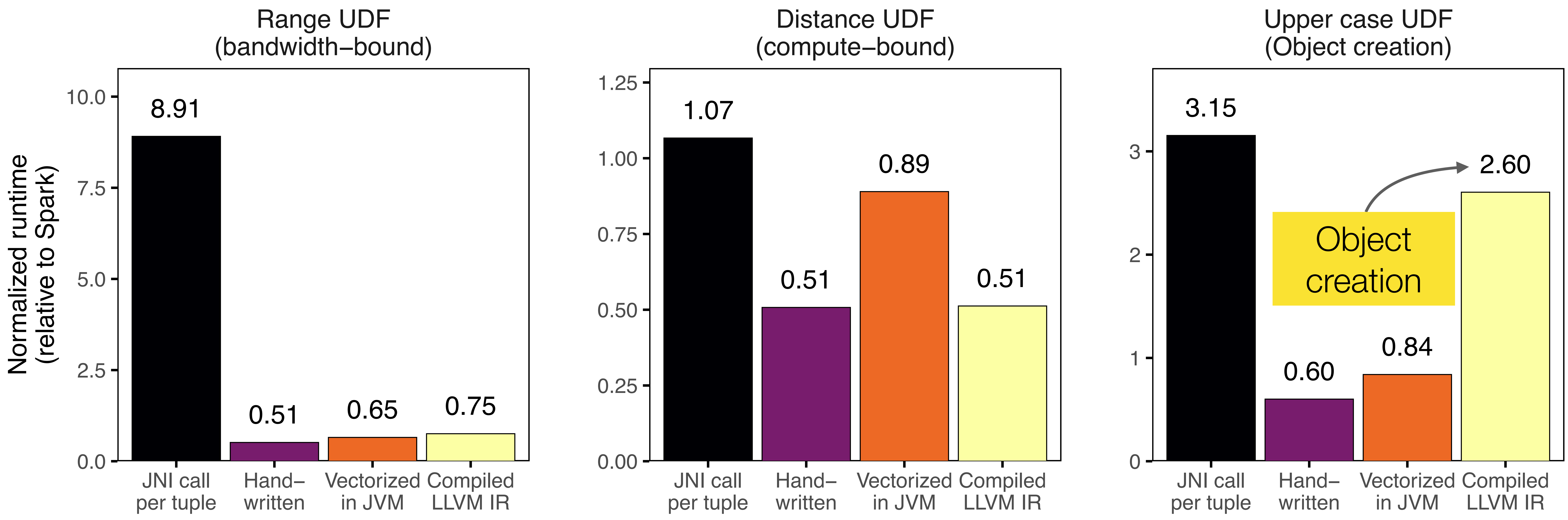
Evaluation

Runtimes relative to execution in Spark, different scales on y axes!



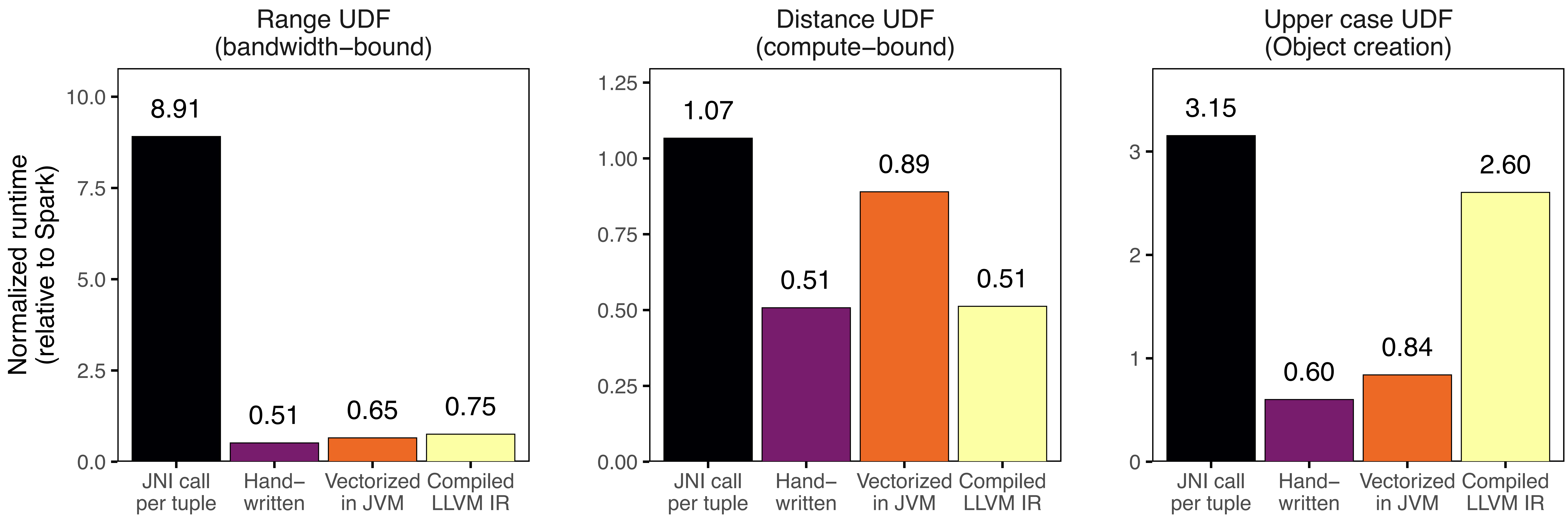
Evaluation

Runtimes relative to execution in Spark, different scales on y axes!



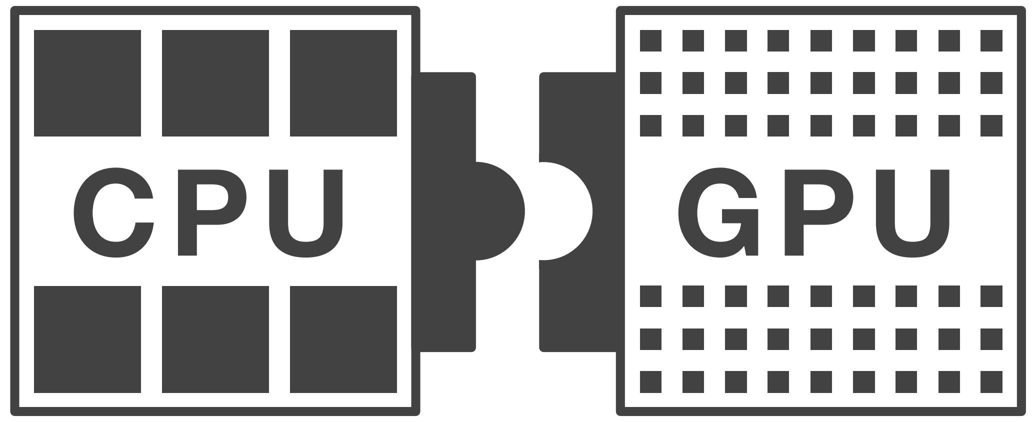
Evaluation

Runtimes relative to execution in Spark, different scales on y axes!



Vectorized execution in Wildfire is faster than execution in Spark.

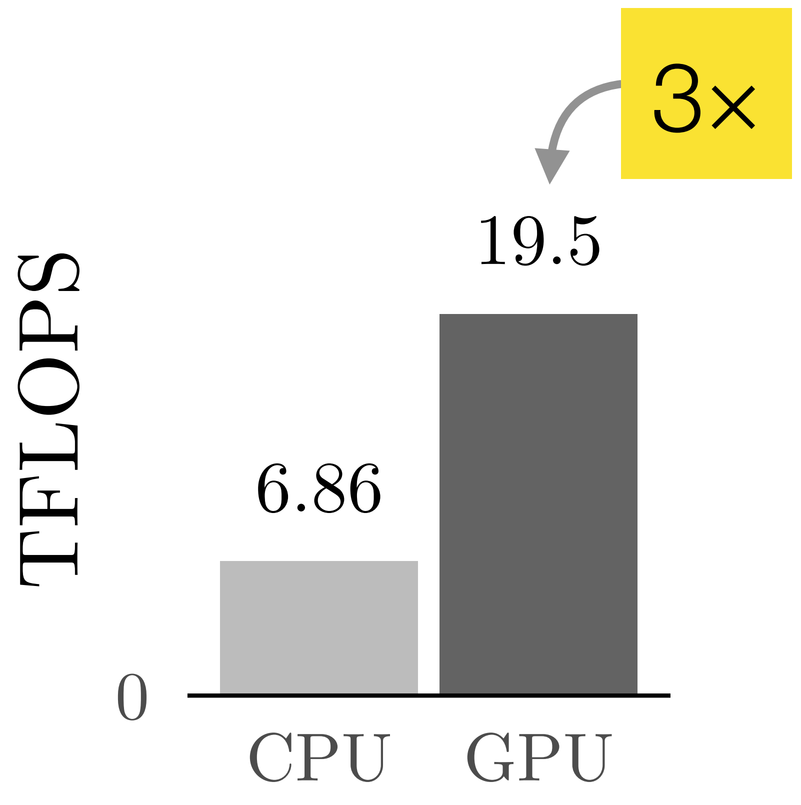
Compilation to machine code is fast if UDF is compute-heavy and does not create objects.



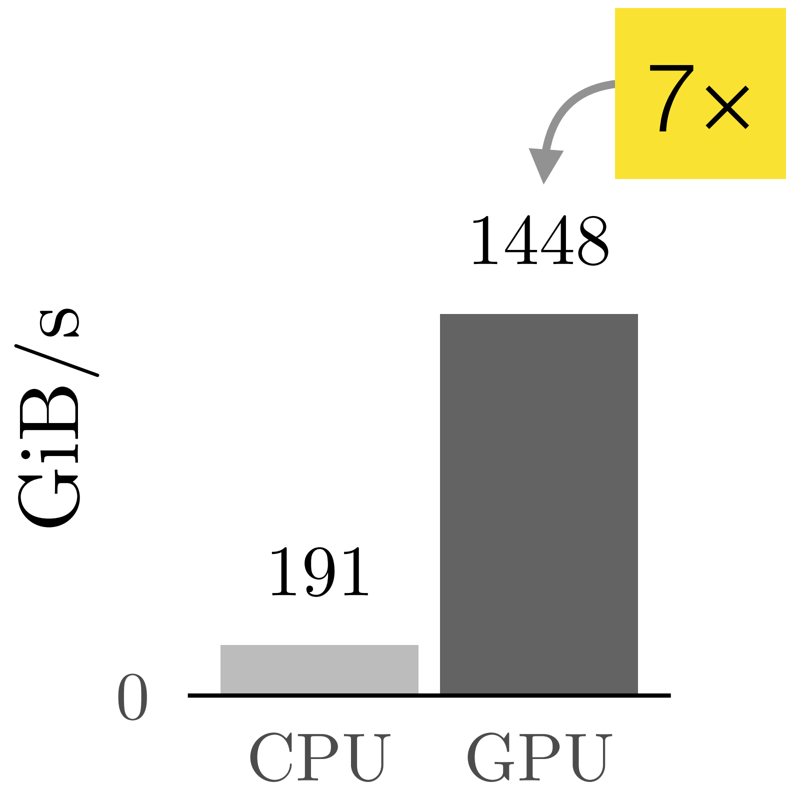
Query Processing on Heterogeneous CPU/GPU Systems

Performance characteristics of CPUs and GPUs

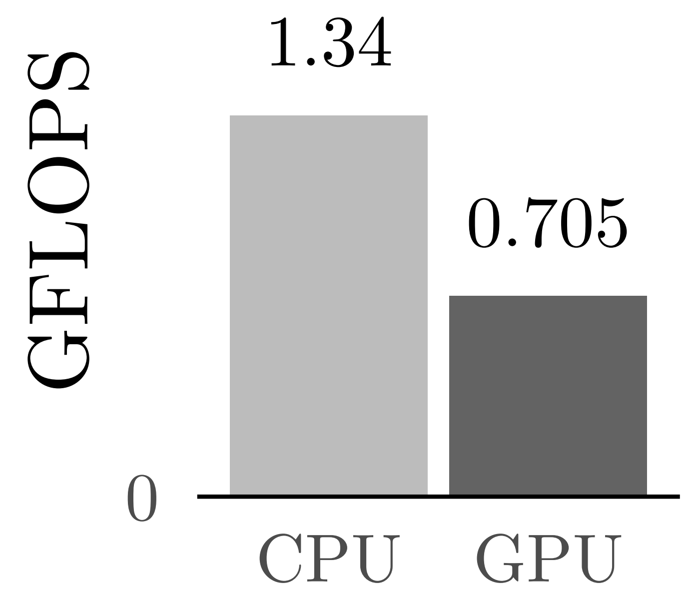
AMD EPYC 7702P vs. NVIDIA A100



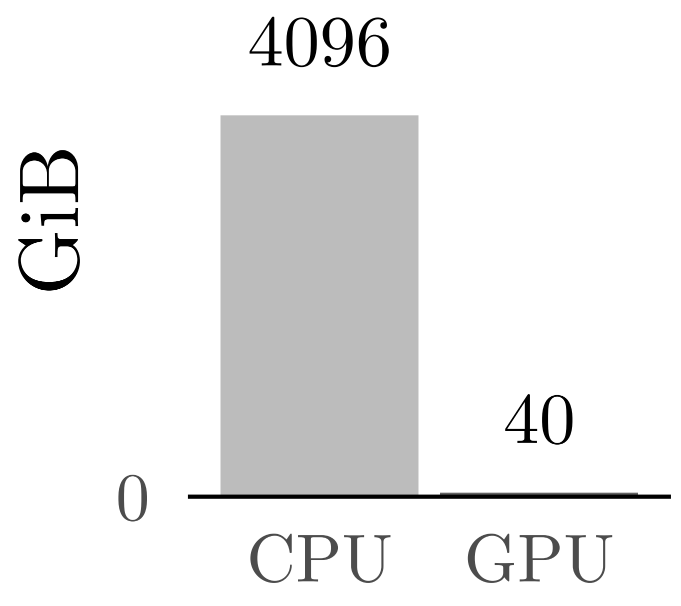
Aggregate performance



Memory bandwidth



Serial performance



Memory size

CPUs

Optimized for **single thread performance**

Extract implicit instruction parallelism

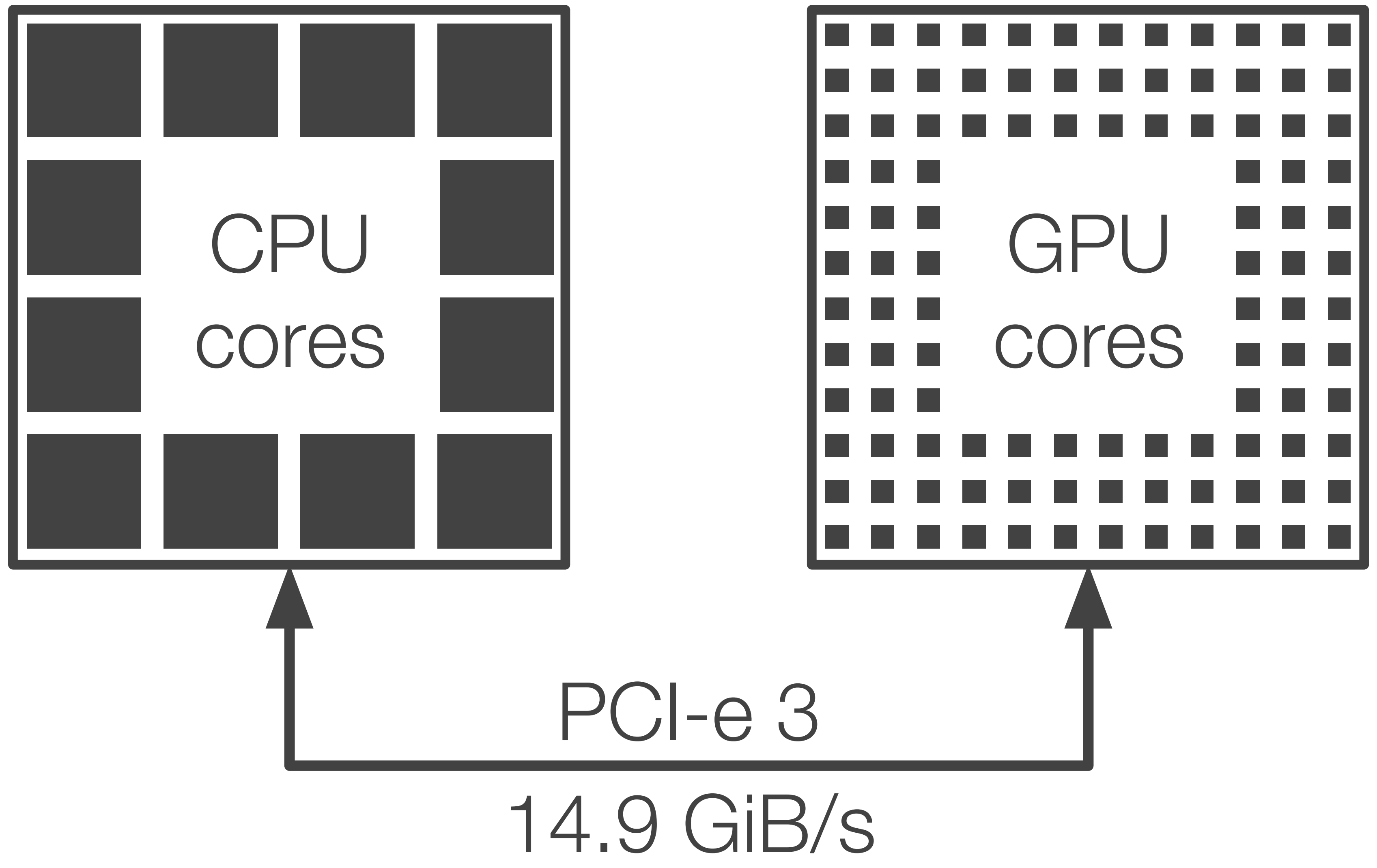
GPUs

Optimized for **throughput applications**

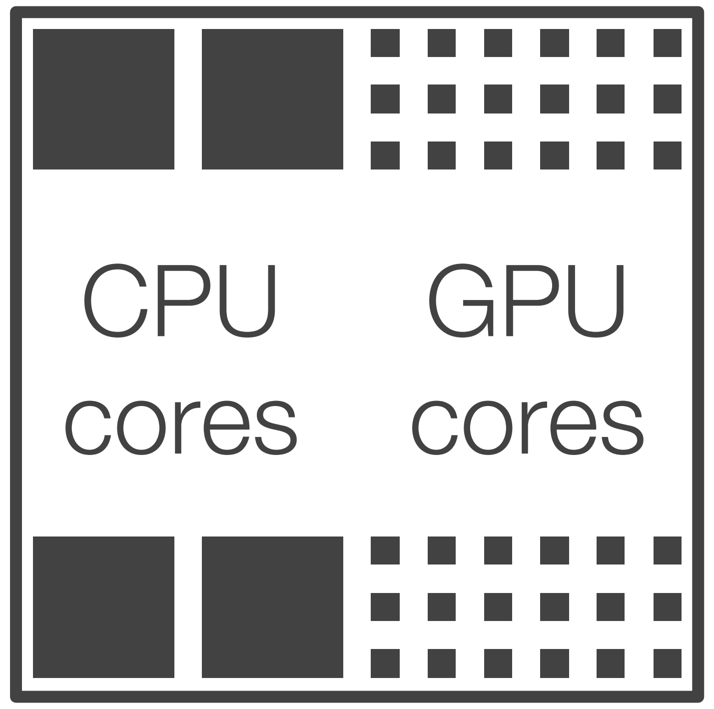
Exploit explicit data parallelism

GPU integration

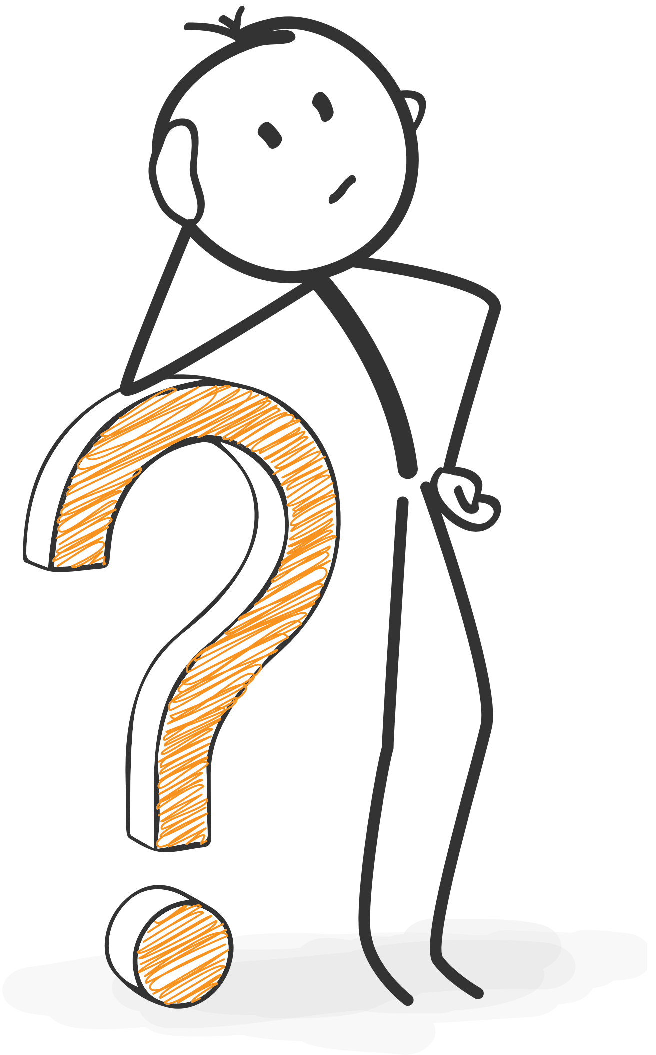
Multi-core CPU and dedicated GPU



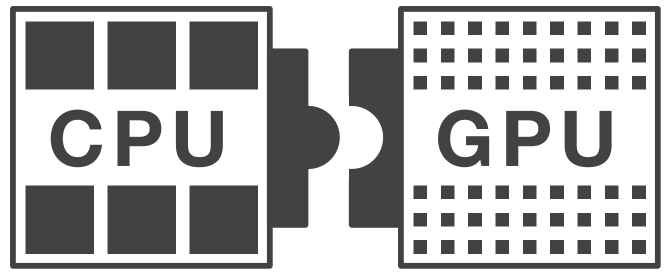
CPU and GPU cores on single die



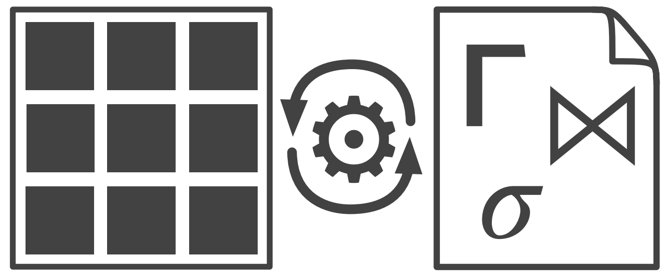
Challenges of a heterogeneous CPU/GPU query processing system



Schedule query workload on different processors



Adapt query processing code to different processors



Mitigate the data transfer bottleneck

Workload scheduling classification scheme

Processor usage – generic compute resource or specialized for specific tasks

Scheduling time – before or during query execution

Scheduling strategy – heuristics, cost models, work stealing

Workload distribution

Task granularity — what kind of tasks are scheduled

Data partitioning

Workload scheduling classification scheme

Processor usage – generic compute resource or specialized for specific tasks

Scheduling time – before or during query execution

Scheduling strategy – heuristics, cost models, work stealing, …

Workload distribution

Task granularity – what kind of tasks are scheduled

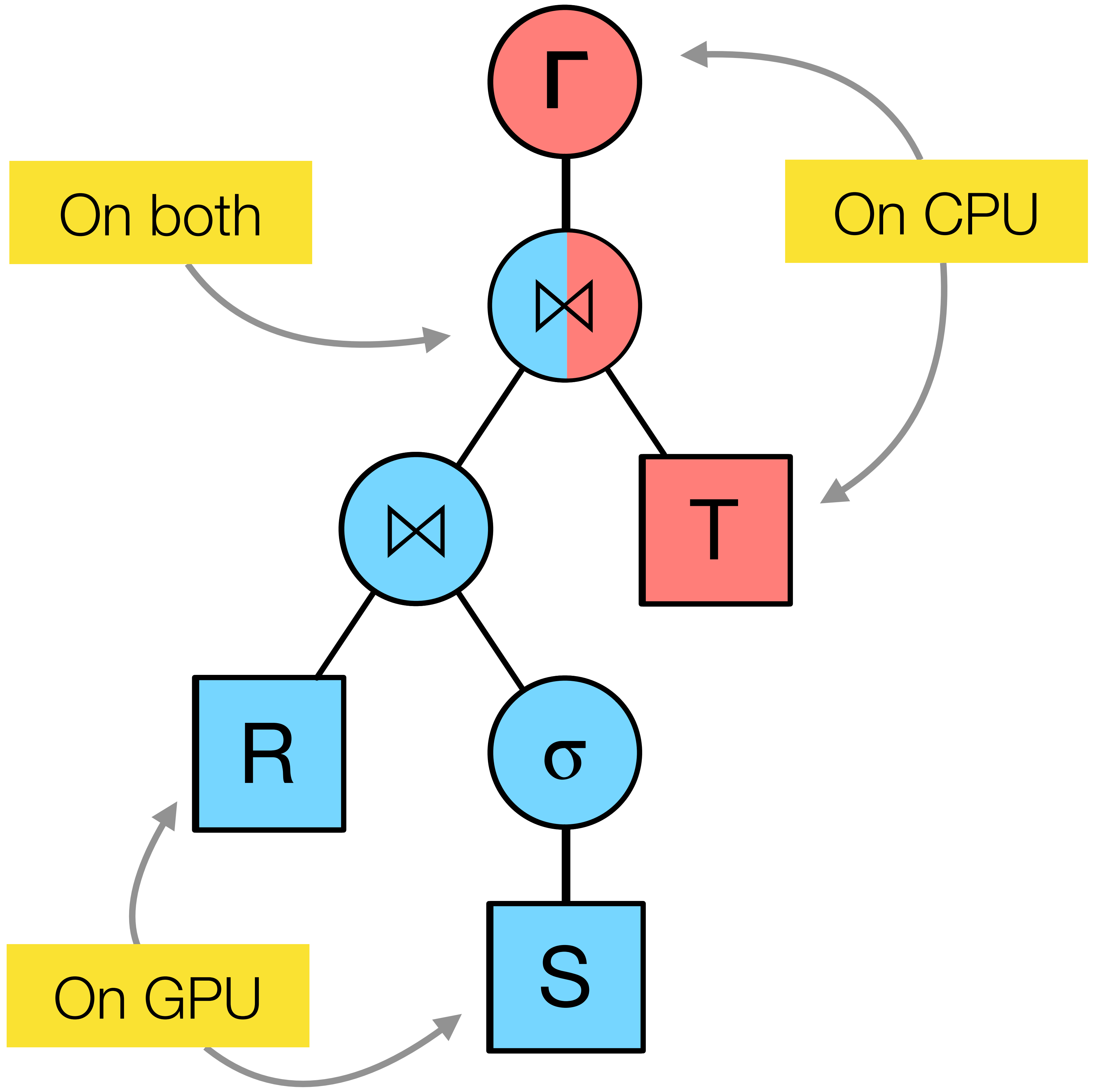
Data partitioning

Processors as generic compute resources

Tasks can be scheduled on any processor

Heuristics, cost models, work stealing

Processors are distinguished by their relative throughput



Processors as specialized resources

Developer analyzes tasks and assigns them to suitable processor

Example: Approximate & Refine [Pirk et al., ICDE 2014]

Bitwise partition of data

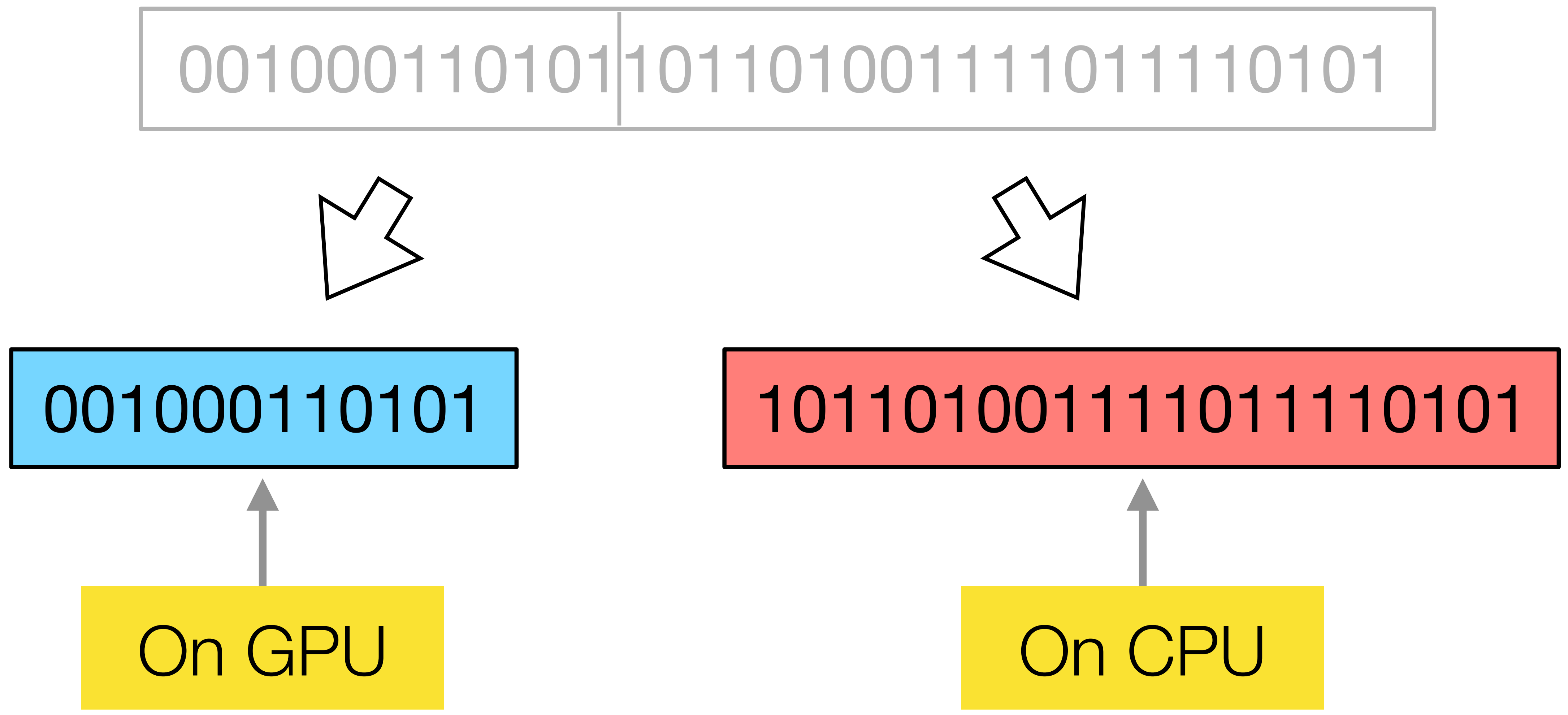
001000110101	1011010011110111110101

Processors as specialized resources

Developer analyzes tasks and assigns them to suitable processor

Example: Approximate & Refine [Pirk et al., ICDE 2014]

- Bitwise partition of data
- Place higher bits on GPU, lower bits on CPU
- Compute approximate result on GPU & refine into exact result on CPU



Publication		GPU integration	Processor usage	Scheduling time	Scheduling strategy	Workload distribution	Task granularity	Data partitioning
Full query processing systems								
Approx. & Refine	Pirk et al., 2014	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	Bits
Stat. coproc.	Heimel et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Mega-KV	Zhang et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Caldera	Appuswamy et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
Raza et al.	Raza et al., 2020	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
GDB	He et al., 2009	Dedicated	Generic	Static	Cost model	Task partitions	Operator	Tuples
CoGaDB	Breß, 2014	Dedicated	Generic	Both	Data locality, cost model	Operator placement	Operator	Columns
SABER	Koliousis et al., 2016	Dedicated	Generic	Dynamic	Load balancing, cost model	Single partition	Query	Data batch
DB2 BLU	Meraji et al., 2016	Dedicated	Generic	Dynamic	Task nature, load balancing	Task partitions	Operator	Tuples
HetExchange	Chrysogelos et al., 2019	Dedicated	Generic	Hybrid	Load balancing, data locality	Task partitions	Pipeline	Data batch
He et al.	He et al., 2014	Integrated	Hybrid	Static	Task nature, cost model	Task partitions	Primitive	Tuples
DIDO	Zhang et al., 2017	Integrated	Hybrid	Hybrid	Task nature, load balancing, cost model	Operator placement	Operator	Query batch
FineStream	Zhang et al., 2020	Integrated	Generic	Dynamic	Cost model	Operator placement	Operator	—
HERO	Karnagel et al., 2017	Both	Generic	Dynamic	Data locality, cost model	Operator placement	Primitive	—
Individual query processing tasks								
GSS	Bøgh et al., 2013	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
STIG	Doraiswamy et al., 2016	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
HB^{+}-tree	Shahvarani et al., 2016	Dedicated	Specialized	Static	Task nature, cost model	*Algorithm-specific*	—	—
Stehle et al.	Stehle et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
G-Grid	Li et al., 2018	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
GAT	Zhang et al., 2018	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Sioulas et al.	Sioulas et al., 2019	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Gubner et al.	Gubner et al., 2019	Dedicated	Hybrid	Hybrid	Task nature, load balancing	Task partitions	Operator	Key batch
SCCG	Wang et al., 2012	Dedicated	Hybrid	Dynamic	Task nature, load balancing	Task partitions	Operator	Polygon pairs
Lutz et al.	Lutz et al., 2020	Dedicated	Hybrid	Hybrid	Task nature, load balancing	Task partitions	Operator	Tuple batch
Beier et al.	Beier et al., 2012	Dedicated	Generic	Dynamic	Cost model	Single partition	—	Query batch
Bøgh et al.	Bøgh et al., 2017	Dedicated	Generic	Dynamic	Load balancing	Single partition	—	Cuboids, points
He et al.	He et al., 2013	Integrated	Generic	Static	Cost model	Task partitions	Primitive	Tuples
HELLS join	Karnagel et al., 2013	Integrated	Specialized	Static	Task nature	*Algorithm-specific*	—	—

Publication		GPU integration	Processor usage	Scheduling time	Scheduling strategy	Workload distribution	Task granularity	Data partitioning
Full query processing systems								
Approx. & Refine	Pirk et al., 2014	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	Bits
Stat. coproc.	Heimel et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Mega-KV	Zhang et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Caldera	Appuswamy et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
Raza et al.	Raza et al., 2020	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
GDB	He et al., 2009	Dedi	[illegible]	[illegible]	[illegible]	ask partitions	Operator	Tuples
CoGaDB	Breß, 2014	Dedi	[illegible]	[illegible]	[illegible]	rator placement	Operator	Columns
SABER	Koliousis et al., 2016	Dedi	[illegible]	[illegible]	[illegible]	ngle partition	Query	Data batch
DB2 BLU	Meraji et al., 2016	Dedicated	Generic	Dynamic	Task nature, load balancing	Task partitions	Operator	Tuples
HetExchange	Chrysogelos et al., 2019	Dedicated	Generic	Hybrid	Load balancing, data locality	Task partitions	Pipeline	Data batch
He et al.	He et al., 2014	Integrated	Hybrid	Static	Task nature, cost model	Task partitions	Primitive	Tuples
DIDO	Zhang et al., 2017	Integrated	Hybrid	Hybrid	Task nature, load balancing, cost model	Operator placement	Operator	Query batch
FineStream	Zhang et al., 2020	Integrated	Generic	Dynamic	Cost model	Operator placement	Operator	—
HERO	Karnagel et al., 2017	Both	Generic	Dynamic	Data locality, cost model	Operator placement	Primitive	—
Individual query processing tasks								
GSS	Bøgh et al., 2013	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
STIG	Doraiswamy et al., 2016	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
HB^+-tree	Shahvarani et al., 2016	Dedicated	Specialized	Static	Task nature, cost model	*Algorithm-specific*	—	—
Stehle et al.	Stehle et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
G-Grid	Li et al., 2018	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
GAT	Zhang et al., 2018	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Sioulas et al.	Sioulas et al., 2019	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Gubner et al.	Gubner et al., 2019	Dedicated	Hybrid	Hybrid	Task nature, load balancing	Task partitions	Operator	Key batch
SCCG	Wang et al., 2012	Dedicated	Hybrid	Dynamic	Task nature, load balancing	Task partitions	Operator	Polygon pairs
Lutz et al.	Lutz et al., 2020	Dedicated	Hybrid	Hybrid	Task nature, load balancing	Task partitions	Operator	Tuple batch
Beier et al.	Beier et al., 2012	Dedicated	Generic	Dynamic	Cost model	Single partition	—	Query batch
Bøgh et al.	Bøgh et al., 2017	Dedicated	Generic	Dynamic	Load balancing	Single partition	—	Cuboids, points
He et al.	He et al., 2013	Integrated	Generic	Static	Cost model	Task partitions	Primitive	Tuples
HELLS join	Karnagel et al., 2013	Integrated	Specialized	Static	Task nature	*Algorithm-specific*	—	—

Full query processing systems

Publication		GPU integration	Processor usage	Scheduling time	Scheduling strategy	Workload distribution	Task granularity	Data partitioning
Full query processing systems								
Approx. & Refine	Pirk et al., 2014	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	Bits
Stat. coproc.	Heimel et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Mega-KV	Zhang et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Caldera	Appuswamy et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
Raza et al.	Raza et al., 2020	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
GDB	He et al., 2009	Dedicated	Generic	Static	Cost model	Task partitions	Operator	Tuples
CoGaDB	Breß, 2014	Dedicated	Generic	Both	Data locality, cost model	Operator placement	Operator	Columns
SABER	Koliousis et al., 2016	Dedicated	Generic	Dynamic	Load balancing, cost model	Single partition	Query	Data batch
DB2 BLU	Meraji et al., 2016	Dedicated	Generic	Dynamic	Task nature, load balancing	Task partitions	Operator	Tuples
HetExchange	Chrysogelos et al., 2019	Dedicated	Generic	Hybrid	Load balancing, data locality	Task partitions	Pipeline	Data batch
He et al.	He et al., 2014	Integrated	Hybrid	Static	Task nature, cost model	Task partitions	Primitive	Tuples
DIDO	Zhang et al., 2017	Integrated	Hybrid	Hybrid	Task nature, load balancing, cost model	Operator placement	Operator	Query batch
FineStream	Zhang et al., 2020	Integrated	Generic	Dynamic	Cost model	Operator placement	Operator	—
HERO	Karnagel et al., 2017	Both	Generic	Dynamic	Data locality, cost model	Operator placement	Primitive	—
Individual query processing tasks								
GSS	Bøgh et al., 2013	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
STIG	Doraiswamy et al., 2016	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
HB$^+$-tree	Shahvarani et al., 2016	Dedicated	Specialized	Static	Task nature, cost model	*Algorithm-specific*	—	—
Stehle et al.	Stehle et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
G-Grid	Li et al., 2018	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
GAT	Zhang et al., 2018	Dedi	[illegible]	[illegible]	[illegible]	*orithm-specific*	—	—
Sioulas et al.	Sioulas et al., 2019	Dedi	[illegible]	[illegible]	[illegible]	*orithm-specific*	—	—
Gubner et al.	Gubner et al., 2019	Dedi	[illegible]	[illegible]	[illegible]	ask partitions	Operator	Key batch
SCCG	Wang et al., 2012	Dedicated	Hybrid	Dynamic	Task nature, load balancing	Task partitions	Operator	Polygon pairs
Lutz et al.	Lutz et al., 2020	Dedicated	Hybrid	Hybrid	Task nature, load balancing	Task partitions	Operator	Tuple batch
Beier et al.	Beier et al., 2012	Dedicated	Generic	Dynamic	Cost model	Single partition	—	Query batch
Bøgh et al.	Bøgh et al., 2017	Dedicated	Generic	Dynamic	Load balancing	Single partition	—	Cuboids, points
He et al.	He et al., 2013	Integrated	Generic	Static	Cost model	Task partitions	Primitive	Tuples
HELLS join	Karnagel et al., 2013	Integrated	Specialized	Static	Task nature	*Algorithm-specific*	—	—

Individual query processing tasks

Publication		GPU integration	Processor usage	Scheduling time	Scheduling strategy	Workload distribution	Task granularity	Data partitioning
Full query processing systems								
Approx. & Refine	Pirk et al., 2014	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	Bits
Stat. coproc.	Heimel et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Mega-KV	Zhang et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Caldera	Appuswamy et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
Raza et al.	Raza et al., 2020	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
GDB	He et al., 2009	Dedicated	Generic	Static	Cost model	Task partitions	Operator	Tuples
CoGaDB	Breß, 2014	Dedicated	Generic	Both	Data locality, cost model	Operator placement	Operator	Columns
SABER	Koliousis et al., 2016	Dedicated	Generic	Dynamic	Load balancing, cost model	Single partition	Query	Data batch
DB2 BLU	Meraji et al., 2016	Dedicated	Generic	Dynamic	Task nature, load balancing	Task partitions	Operator	Tuples
HetExchange	Chrysogelos et al., 2019	Dedicated	Generic	Hybrid	Load balancing, data locality	Task partitions	Pipeline	Data batch
He et al.	He						itive	Tuples
DIDO	Zha						ator	Query batch
FineStream	Zha						ator	—
HERO	Ka						itive	—
Individual query processing tasks								
GSS	Bøgh et al., 2013	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
STIG	Doraiswamy et al., 2016	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
HB$^+$-tree	Shahvarani et al., 2016	Dedicated	Specialized	Static	Task nature, cost model	*Algorithm-specific*	—	—
Stehle et al.	Stehle et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
G-Grid	Li et al., 2018	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
GAT	Zhang et al., 2018	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Sioulas et al.	Sioulas et al., 2019	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Gubner et al.	Gubner et al., 2019	Dedicated	Hybrid	Hybrid	Task nature, load balancing	Task partitions	Operator	Key batch
SCCG	Wang et al., 2012	Dedicated	Hybrid	Dynamic	Task nature, load balancing	Task partitions	Operator	Polygon pairs
Lutz et al.	Lutz et al., 2020	Dedicated	Hybrid	Hybrid	Task nature, load balancing	Task partitions	Operator	Tuple batch
Beier et al.	Beier et al., 2012	Dedicated	Generic	Dynamic	Cost model	Single partition	—	Query batch
Bøgh et al.	Bøgh et al., 2017	Dedicated	Generic	Dynamic	Load balancing	Single partition	—	Cuboids, points
He et al.	He et al., 2013	Integrated	Generic	Static	Cost model	Task partitions	Primitive	Tuples
HELLS join	Karnagel et al., 2013	Integrated	Specialized	Static	Task nature	*Algorithm-specific*	—	—

Dedicated GPUs are used as specialized compute resources

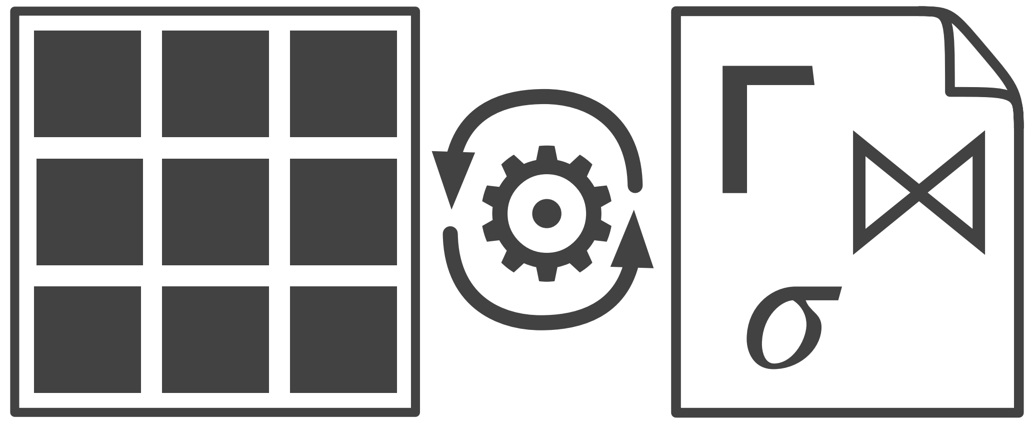
Publication		GPU integration	Processor usage	Scheduling time	Scheduling strategy	Workload distribution	Task granularity	Data partitioning
Full query processing systems								
Approx. & Refine	Pirk et al., 2014	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	Bits
Stat. coproc.	Heimel et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Mega-KV	Zhang et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Caldera	Appuswamy et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
Raza et al.	Raza et al., 2020	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
GDB	He et al., 2009	Dedicated	Generic	Static	Cost model	Task partitions	Operator	Tuples
CoGaDB	Breß, 2014	Dedicated	Generic	Both	Data locality, cost model	Operator placement	Operator	Columns
SABER	Koliousis et al., 2016	Dedicated	Generic	Dynamic	Load balancing, cost model	Single partition	Query	Data batch
DB2 BLU	Meraji et al., 2016	Dedicated	Generic	Dynamic	Task nature, load balancing	Task partitions	Operator	Tuples
HetExchange	Chrysogelos et al., 2019	Dedicated	Generic	Hybrid	Load balancing, data locality	Task partitions	Pipeline	Data batch
He et al.	He et al., 2014	Integrated	Hybrid	Static	Task nature, cost model	Task partitions	Primitive	Tuples
DIDO	Zhang et al., 2017	Integrated	Hybrid	Hybrid	Task nature, load balancing, cost model	Operator placement	Operator	Query batch
FineStream	Zhang et al., 2020	Integrated	Generic	Dynamic	Cost model	Operator placement	Operator	—
HERO	Karnagel et al., 2017	Both	Generic	Dynamic	Data locality, cost model	Operator placement	Primitive	—
Individual query processing tasks								
GSS	Bøgh et al., 2013	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
STIG	Doraiswamy et al., 2016	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
HB$^+$-tree	Shahvarani et al., 2016	Dedicated	Specialized	Static	Task nature, cost model	*Algorithm-specific*	—	—
Stehle et al.	Stehle et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
G-Grid	Li						—	—
GAT	Zha						—	—
Sioulas et al.	Sio						—	—
Gubner et al.	Gu						rator	Key batch
SCCG	Wang et al., 2012	Dedicated	Hybrid	Dynamic	Task nature, load balancing	Task partitions	Operator	Polygon pairs
Lutz et al.	Lutz et al., 2020	Dedicated	Hybrid	Hybrid	Task nature, load balancing	Task partitions	Operator	Tuple batch
Beier et al.	Beier et al., 2012	Dedicated	Generic	Dynamic	Cost model	Single partition	—	Query batch
Bøgh et al.	Bøgh et al., 2017	Dedicated	Generic	Dynamic	Load balancing	Single partition	—	Cuboids, points
He et al.	He et al., 2013	Integrated	Generic	Static	Cost model	Task partitions	Primitive	Tuples
HELLS join	Karnagel et al., 2013	Integrated	Specialized	Static	Task nature	*Algorithm-specific*	—	—

Integrated GPUs are used as generic compute resources

Publication		GPU integration	Processor usage	Scheduling time	Scheduling strategy	Workload distribution	Task granularity	Data partitioning
Full query processing systems								
Approx. & Refine	Pirk et al., 2014	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	Bits
Stat. coproc.	Heimel et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Mega-KV	Zhang et al., 2015	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
Caldera	Appuswamy et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
Raza et al.	Raza et al., 2020	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	Query type	—
GDB	He et al., 2009	Dedicated	Generic	Static	Cost model	Task partitions	Operator	Tuples
CoG[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]s
SAE[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]ch
DB2[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
HetExchange	Chrysogelos et al., 2019	Dedicated	Generic	Hybrid	Load balancing, data locality	Task partitions	Pipeline	Data batch
He et al.	He et al., 2014	Integrated	Hybrid	Static	Task nature, cost model	Task partitions	Primitive	Tuples
DIDO	Zhang et al., 2017	Integrated	Hybrid	Hybrid	Task nature, load balancing, cost model	Operator placement	Operator	Query batch
FineStream	Zhang et al., 2020	Integrated	Generic	Dynamic	Cost model	Operator placement	Operator	—
HERO	Karnagel et al., 2017	Both	Generic	Dynamic	Data locality, cost model	Operator placement	Primitive	—
Individual query processing tasks								
GSS	Bøgh et al., 2013	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
STIG	Doraiswamy et al., 2016	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	—	—
HB+-tree	Shahvarani et al., 2016	Dedicated	Specialized	Static	Task nature, cost model	*Algorithm-specific*	—	—
Stehle et al.	Stehle et al., 2017	Dedicated	Specialized	Static	Task nature	*Algorithm-specific*	[illegible]	[illegible]
G-G[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GAT[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Siou[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Gubner et al.	Gubner et al., 2019	Dedicated	Hybrid	Hybrid	Task nature, load balancing	Task partitions	Operator	Key batch
SCCG	Wang et al., 2012	Dedicated	Hybrid	Dynamic	Task nature, load balancing	Task partitions	Operator	Polygon pairs
Lutz et al.	Lutz et al., 2020	Dedicated	Hybrid	Hybrid	Task nature, load balancing	Task partitions	Operator	Tuple batch
Beier et al.	Beier et al., 2012	Dedicated	Generic	Dynamic	Cost model	Single partition	—	Query batch
Bøgh et al.	Bøgh et al., 2017	Dedicated	Generic	Dynamic	Load balancing	Single partition	—	Cuboids, points
He et al.	He et al., 2013	Integrated	Generic	Static	Cost model	Task partitions	Primitive	Tuples
HELLS join	Karnagel et al., 2013	Integrated	Specialized	Static	Task nature	*Algorithm-specific*	—	—

Dedicated GPUs should perform specialized coarse-grained tasks.

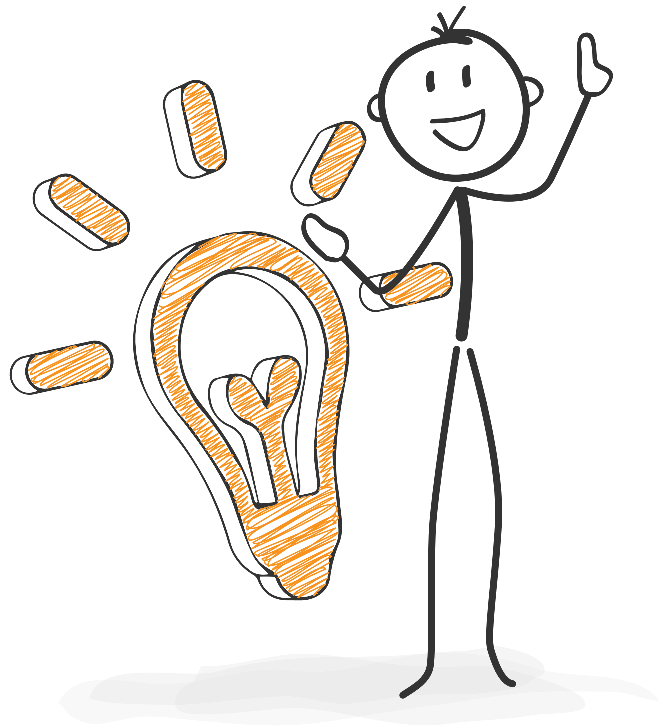
Integrated GPUs should cooperate closely with CPU.



Operator Variant Tuning on Heterogeneous Processors

Processor sensitivity

How sensitive are processors to operator implementation details?

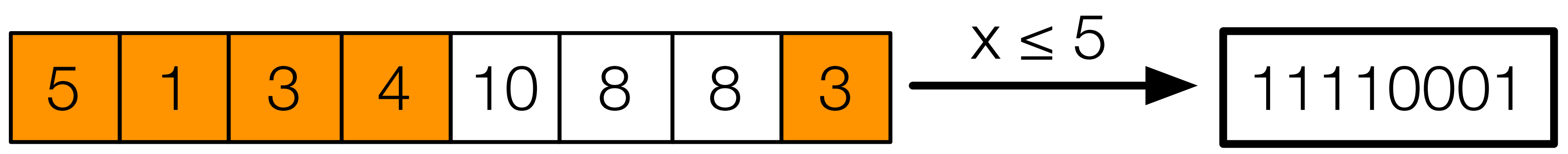


Performance analysis of selection and hash aggregation operators

Multiple CPUs, GPUs + Intel Xeon Phi coprocessor

Selection variants

Operation



Basic selection kernels — Sequential, GlobalAtomic, LocalAtomic, Reduce, Collect, Transpose

Variants — Thread configuration

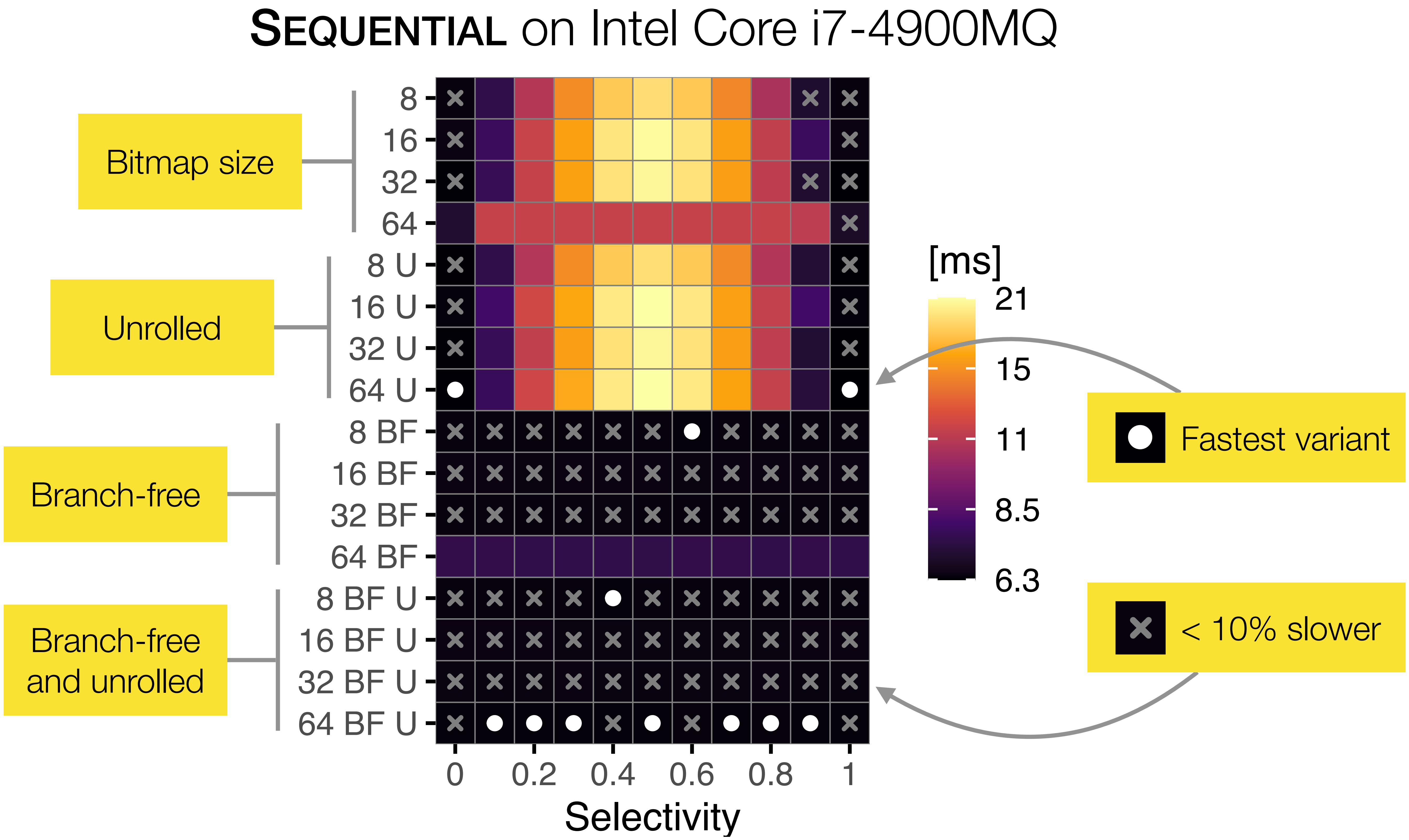
Low-level implementation parameters

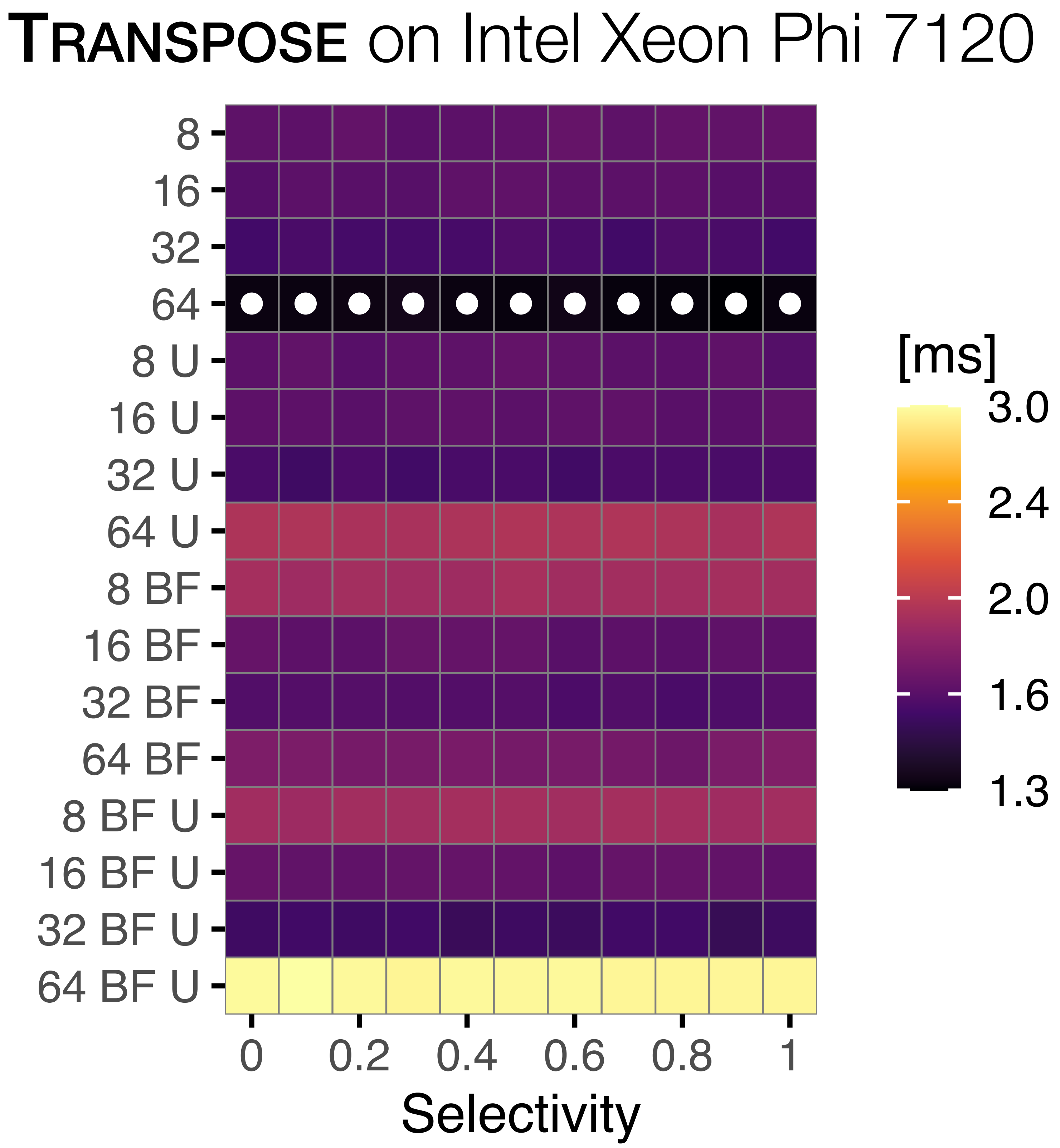
Processors — 7 CPUs, 5 GPUs, Xeon Phi

AMD, IBM, Intel, Nvidia

Selection variants performance

Selection on 32 million random integers, uniform distribution, median of 10 repetitions





No single variant performs best on every processor.

Competitive variants

At most 10% slower than the fastest variant

Processor	Number of variants	Competitive variants	Percentage	Maximum slowdown
Intel Xeon E5620	5880	1370	23 %	32
Nvidia Tesla K40m	4696	129	2.7 %	136
Intel Xeon Phi 7120	3886	6	0.15 %	147

Some processors are very sensitive to implementation details.

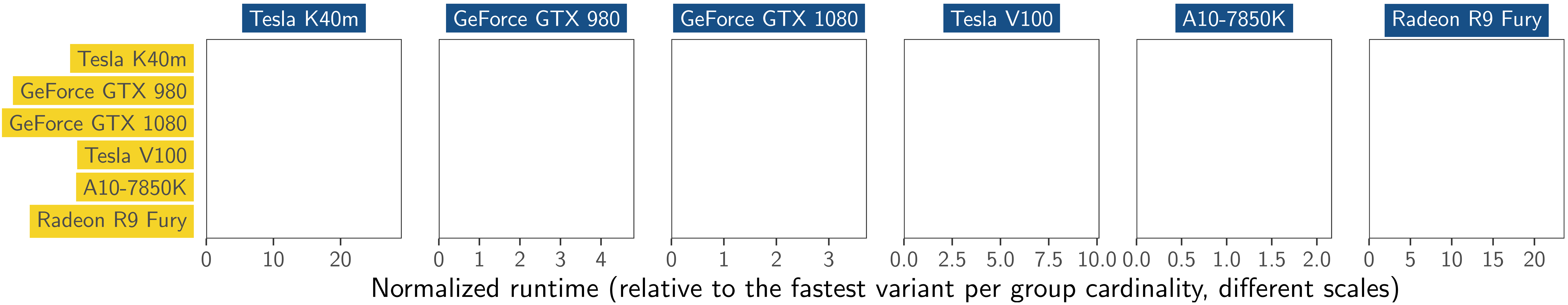
Hash aggregation variants

Parallelization strategy	SHARED, INDEPENDENT, WORKGROUPLOCAL
Variants	Thread configuration
Processors	6 GPUs
	AMD, Nvidia
	Different micro architectures

Influence of thread configuration

128 million input rows, 32-bit keys and values, Sum aggregation, group cardinality between 1 and 2^{28}

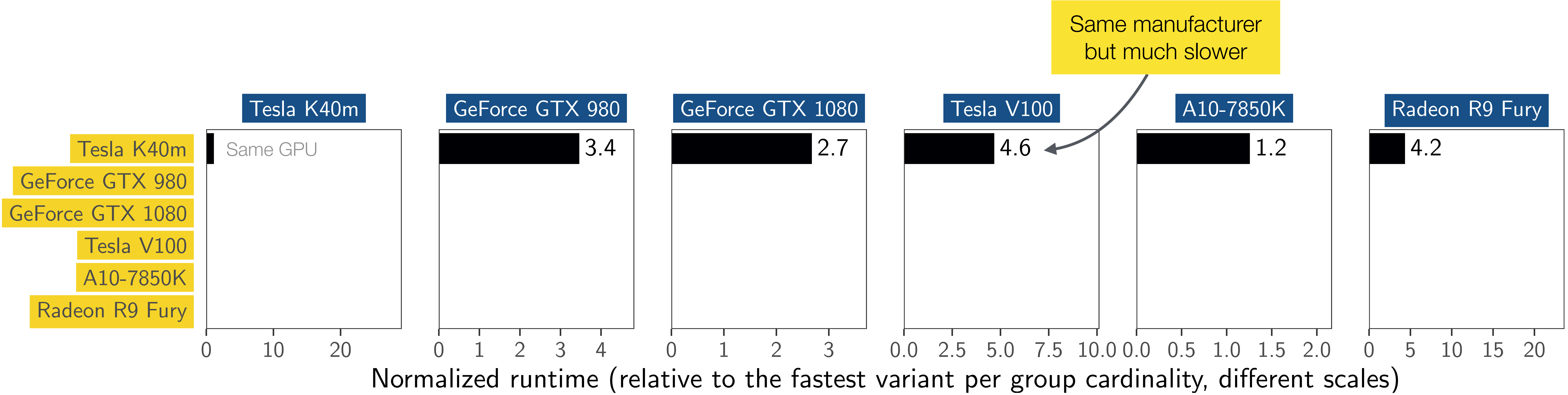
Performance penalty when a thread configuration optimized for a specific GPU (rows) is executed on another GPU (columns).



Influence of thread configuration

128 million input rows, 32-bit keys and values, Sum aggregation, group cardinality between 1 and 2^{28}

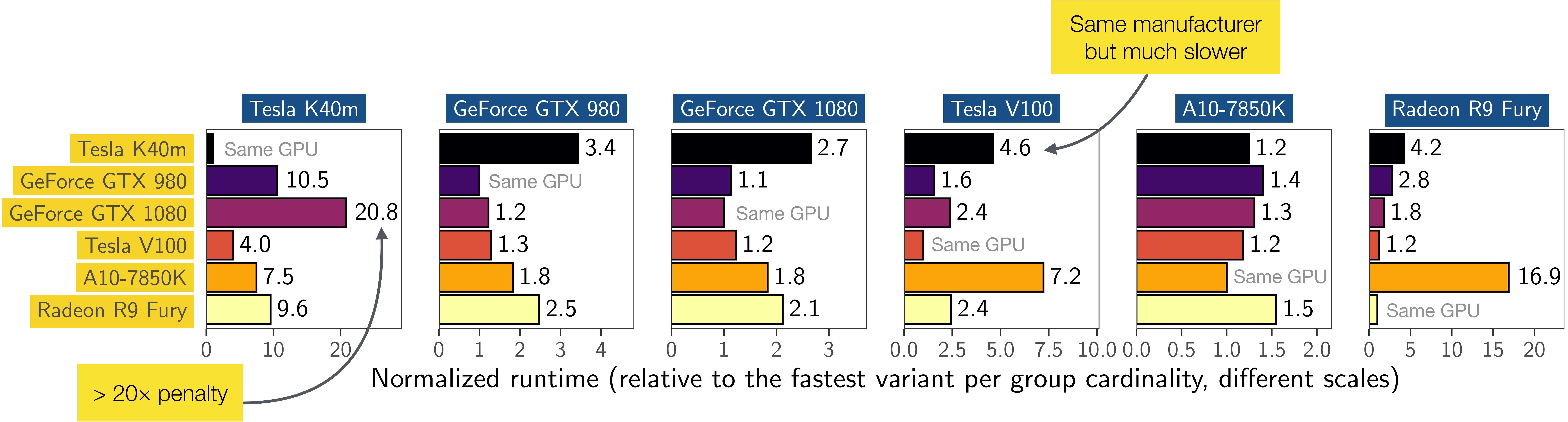
Performance penalty when a thread configuration optimized for a specific GPU (rows) is executed on another GPU (columns).



Influence of thread configuration

128 million input rows, 32-bit keys and values, Sum aggregation, group cardinality between 1 and 2^{28}

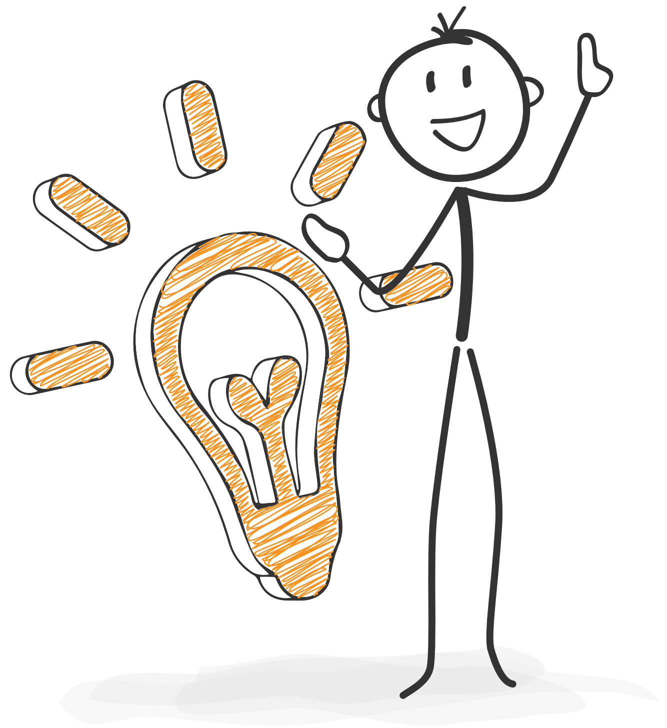
Performance penalty when a thread configuration optimized for a specific GPU (rows) is executed on another GPU (columns).



The fastest thread configuration is highly GPU-specific.

Variant tuning

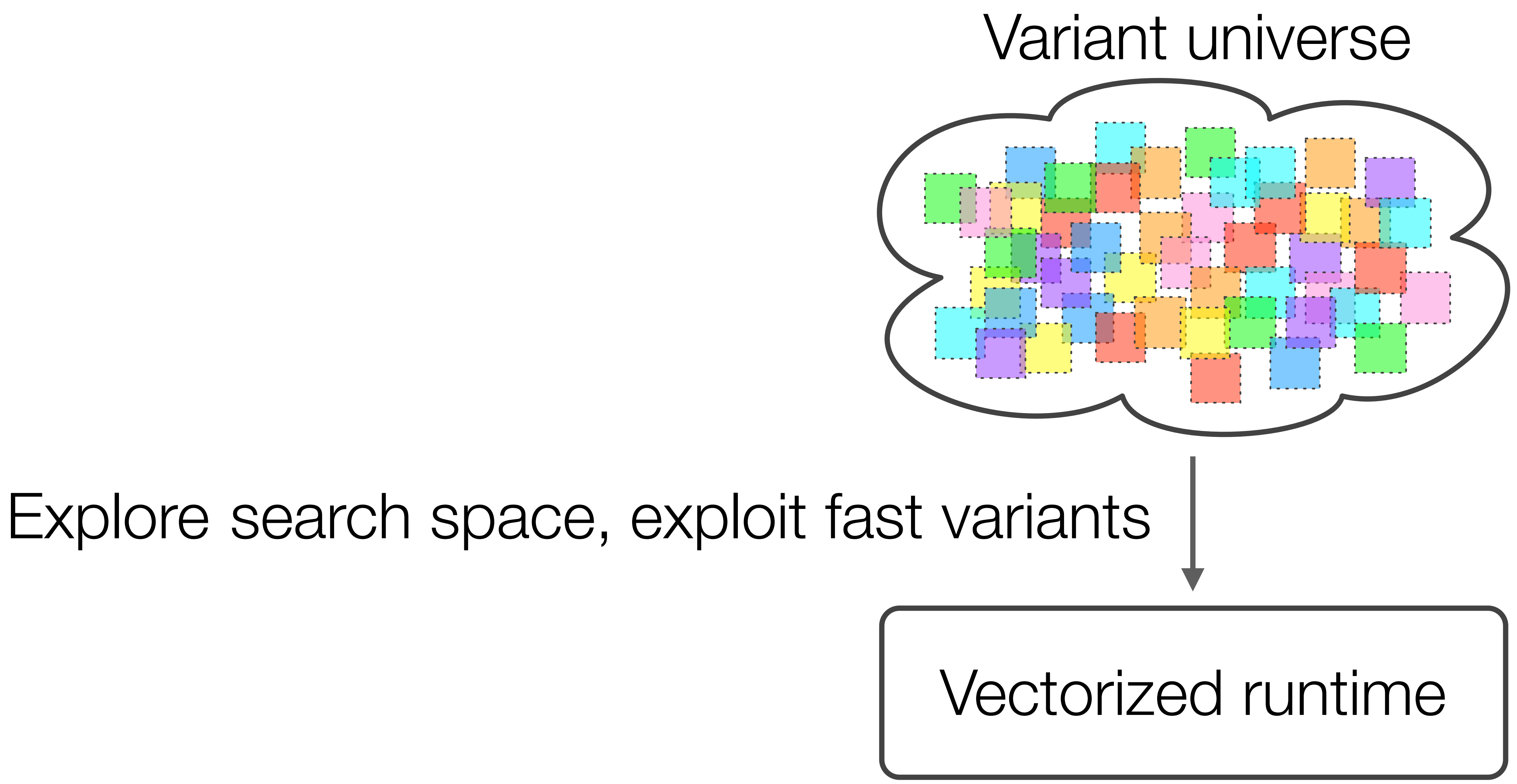
Let the database find a fast operator implementation automatically



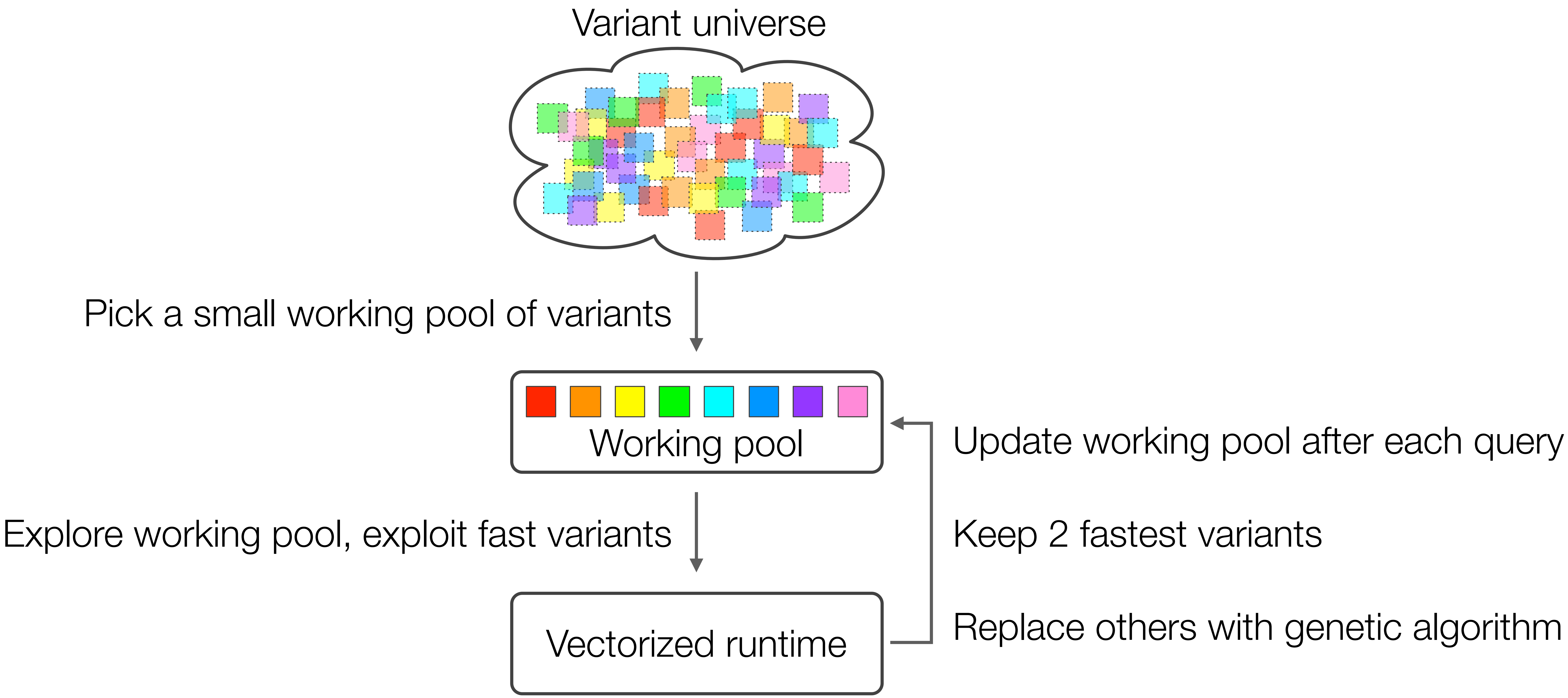
Extend micro adaptivity to handle large search spaces

Extend local search to handle runtime variation

Candidate selection in large variant spaces

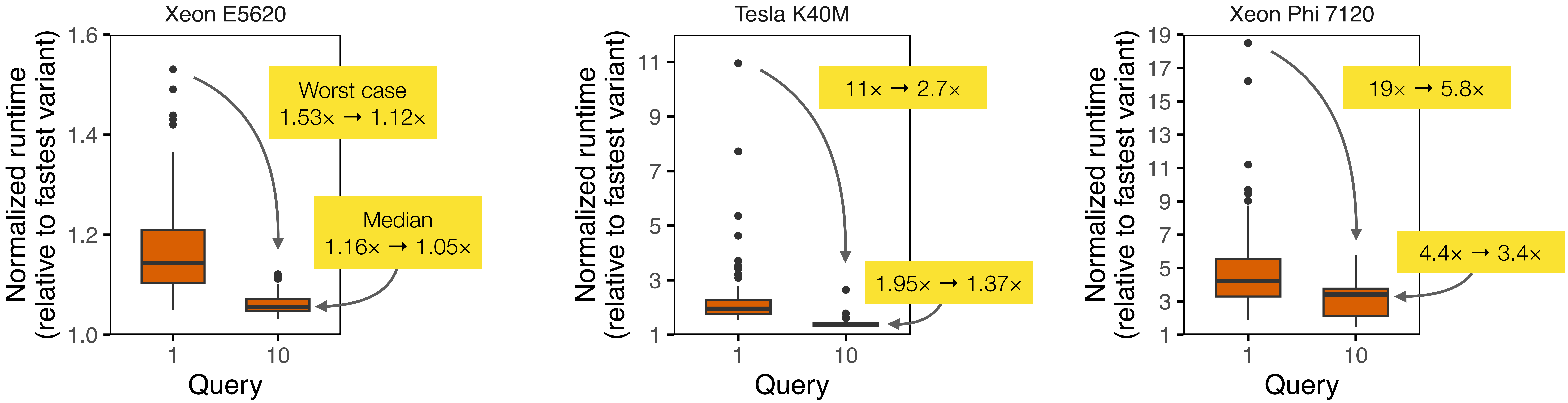


Candidate selection in large variant spaces



Evaluation

Selection variants, selectivity 0.5, 1024 blocks/query, 8 variants in pool, genetic algorithm after each query, 100 reps



23% competitive variants
→ 88% chance that initial pool contains a competitive variant

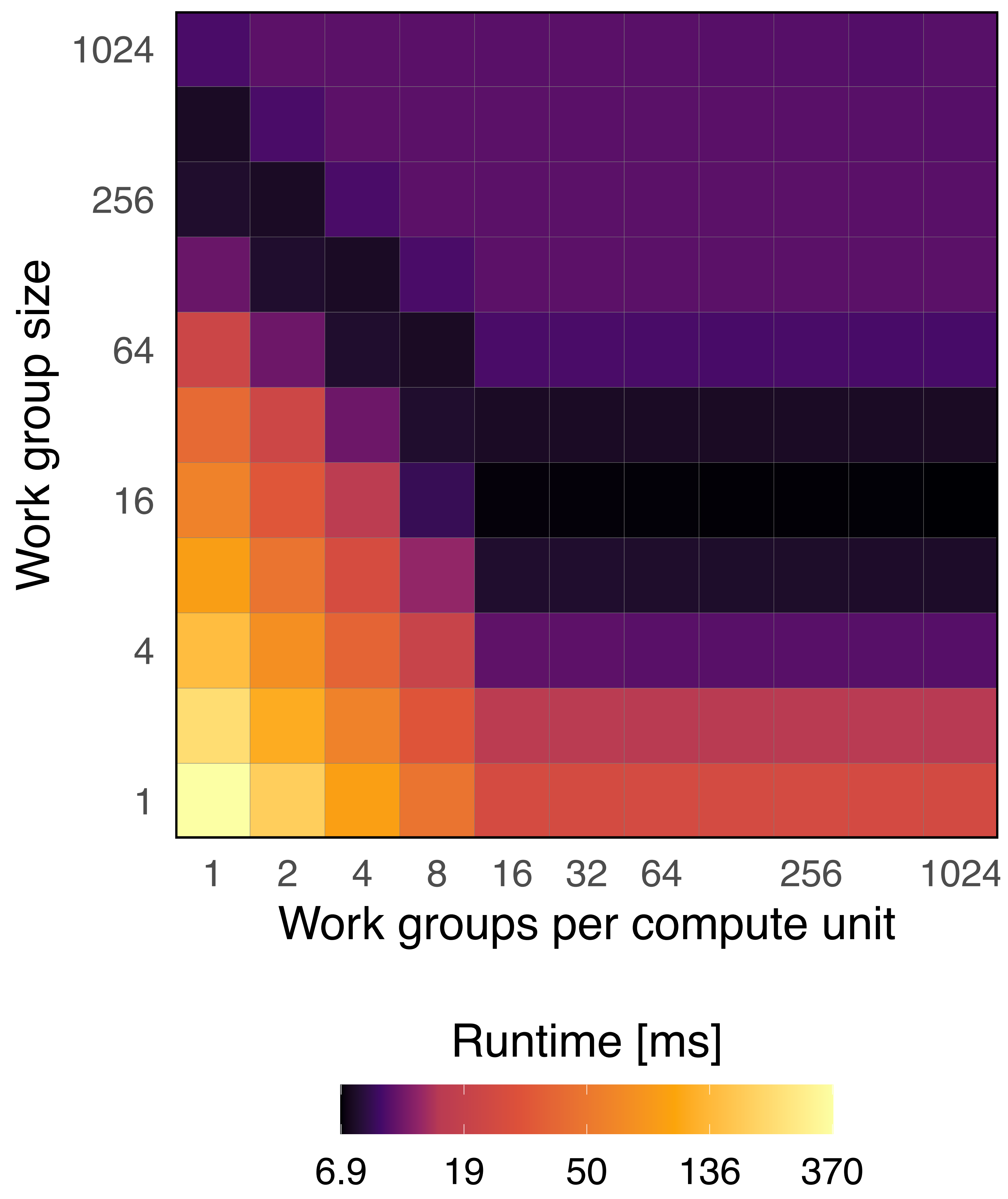
2.7% competitive variants
→ 20% chance

0.15% competitive variants
→ 1.2% chance

Variant tuning performance depends on initial working pool and number of competitive variants.

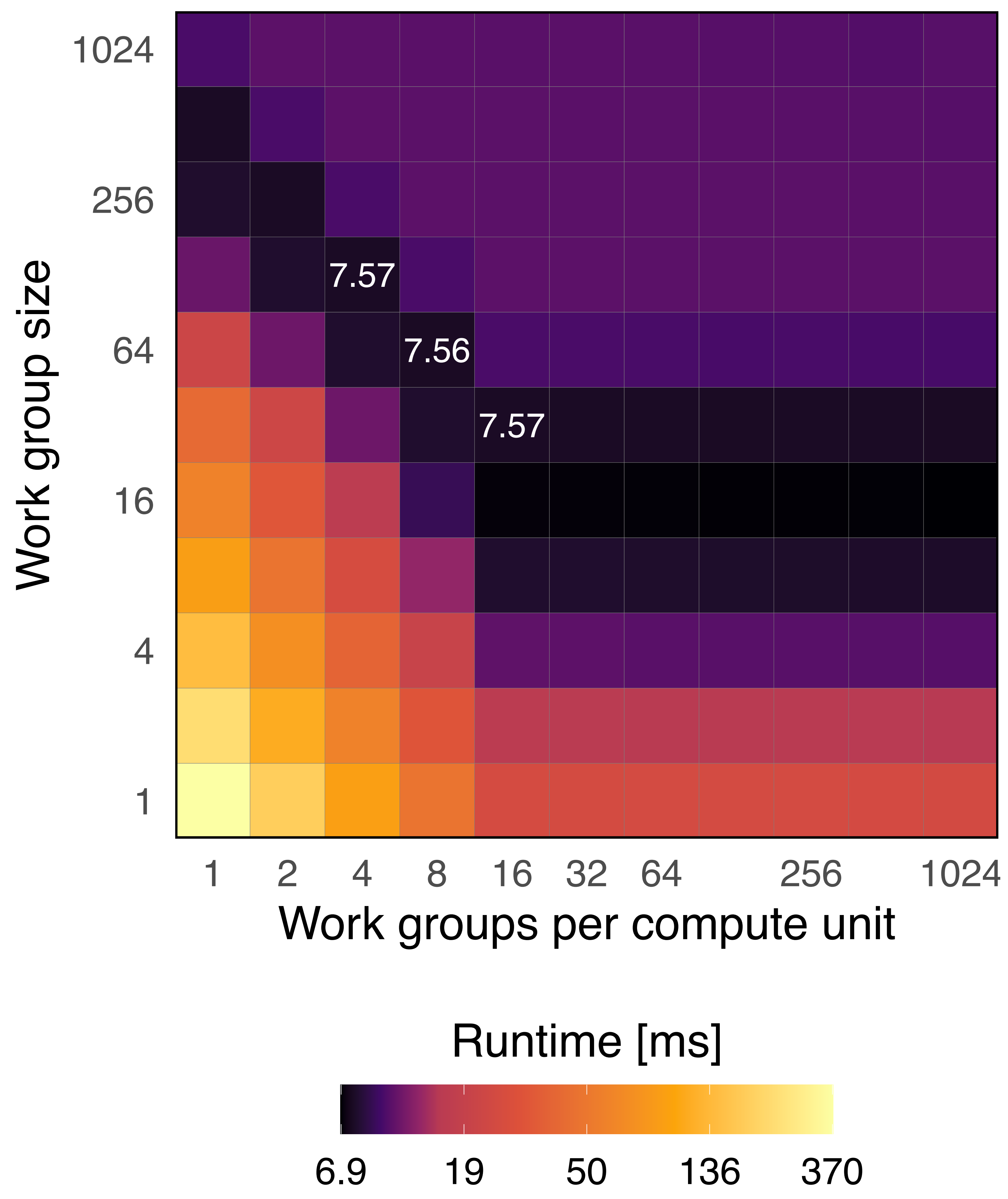
Performance plateaus

Shared aggregation, Tesla K40m, 8 million groups, 128 million input rows, 32-bit keys and values



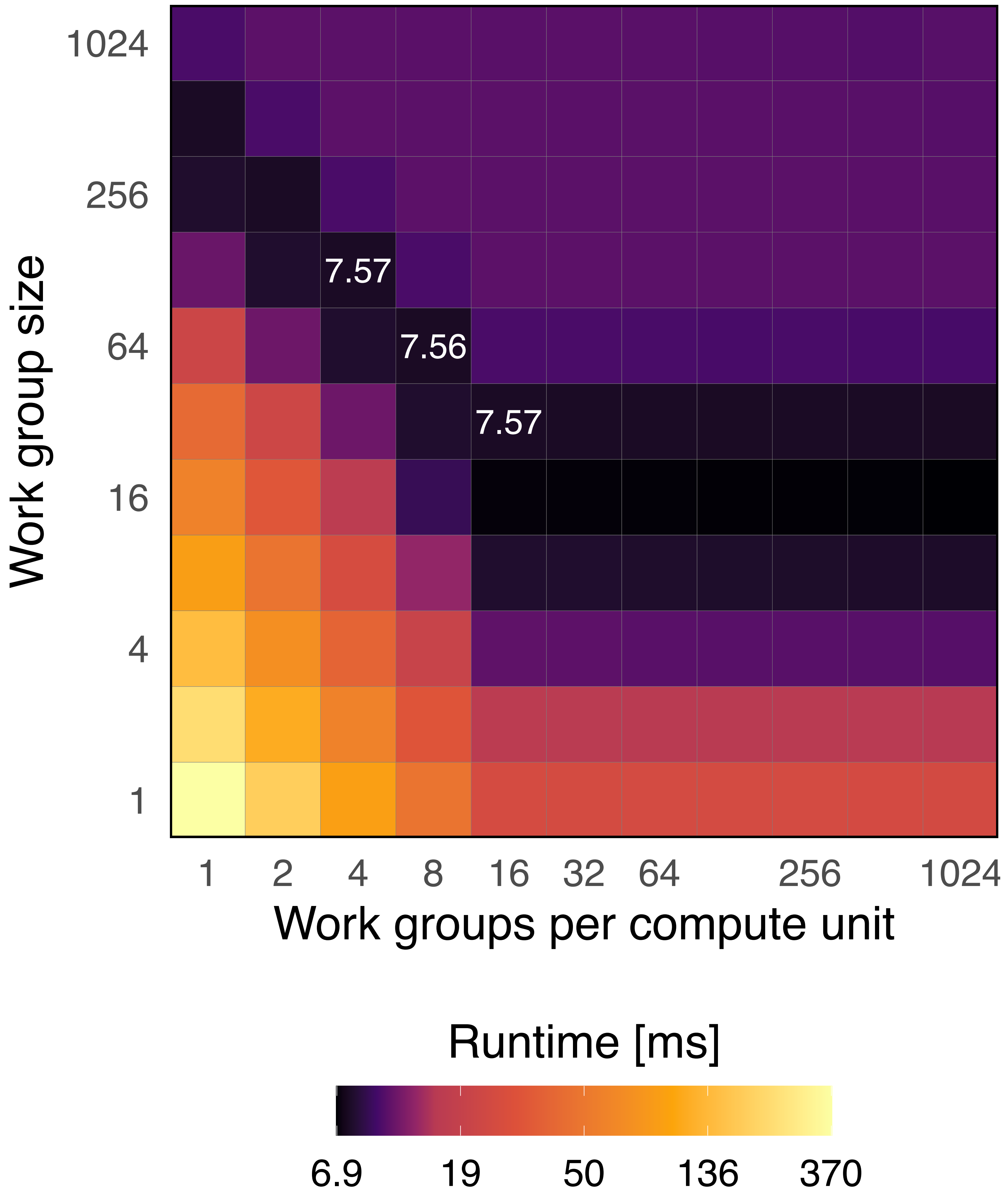
Performance plateaus

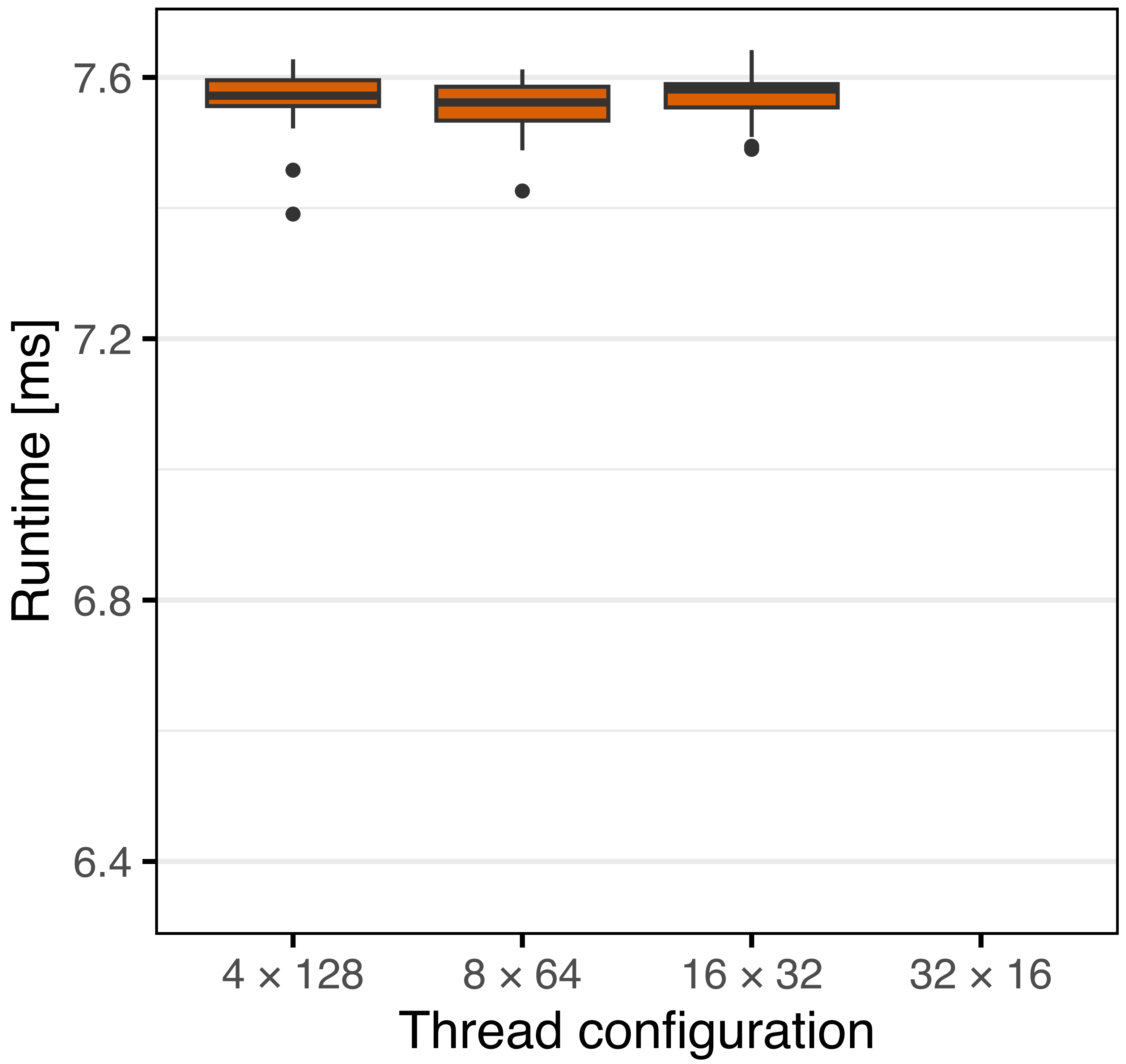
Shared aggregation, Tesla K40m, 8 million groups, 128 million input rows, 32-bit keys and values



Performance plateaus

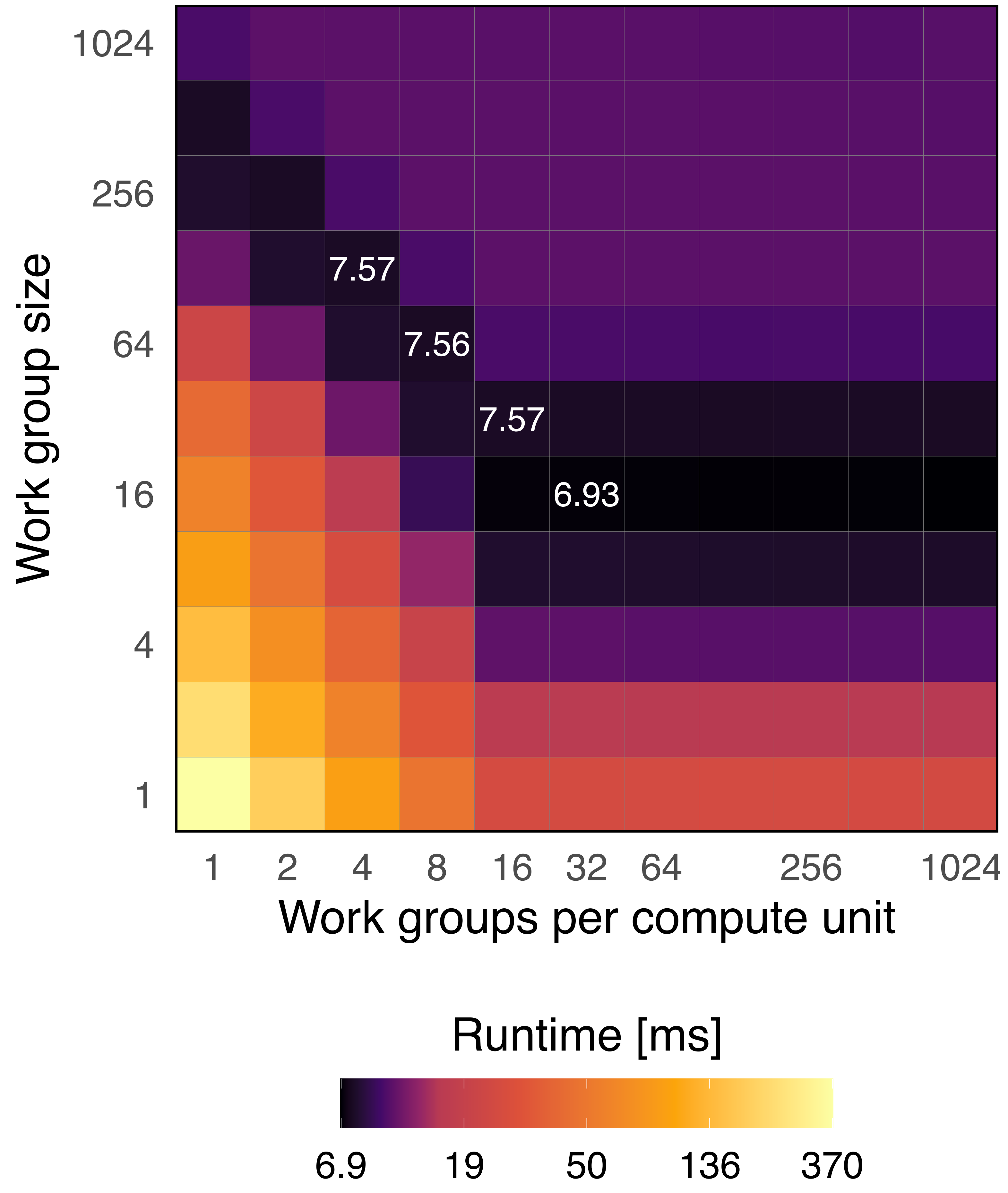
Shared aggregation, Tesla K40m, 8 million groups, 128 million input rows, 32-bit keys and values

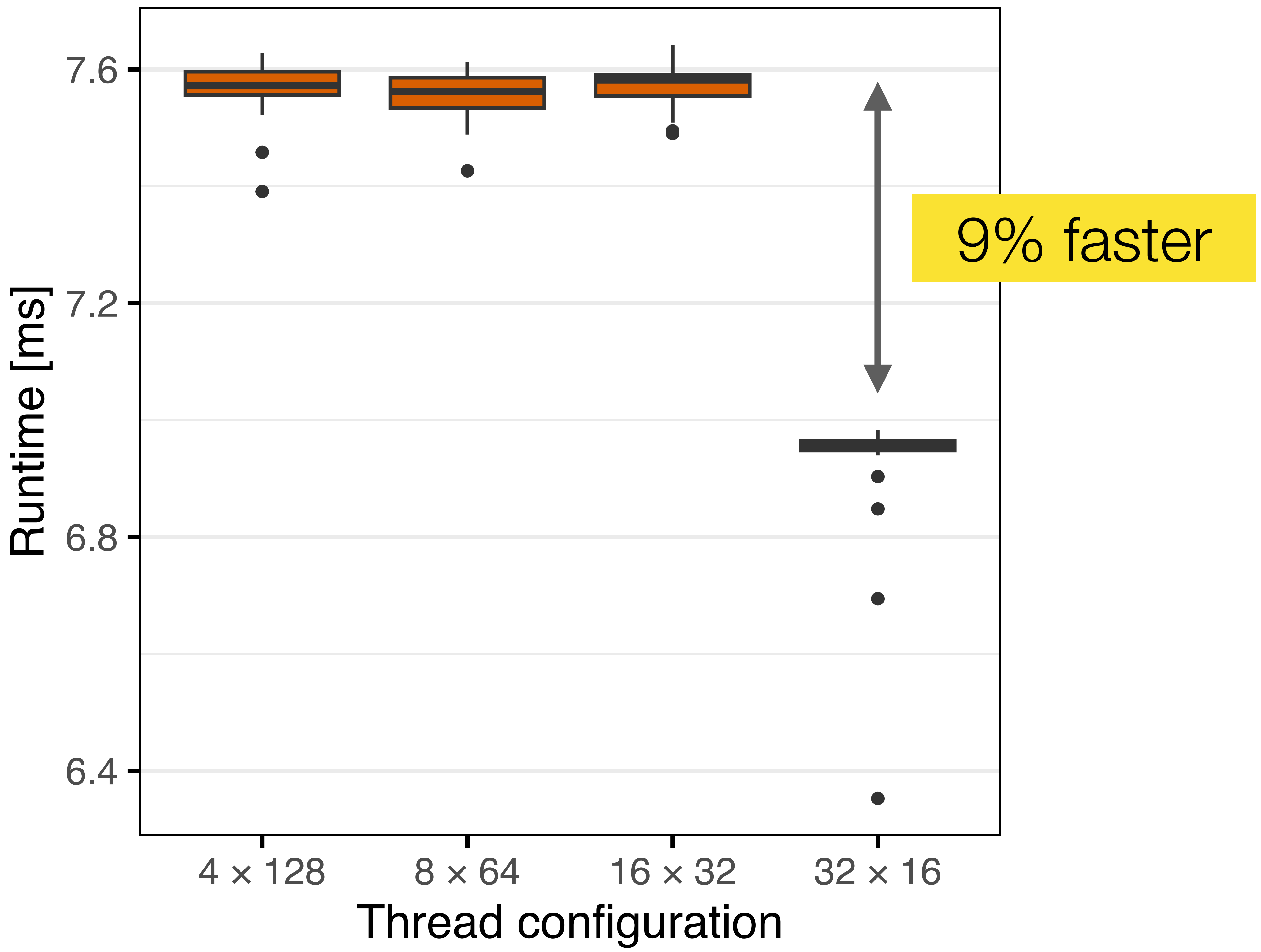




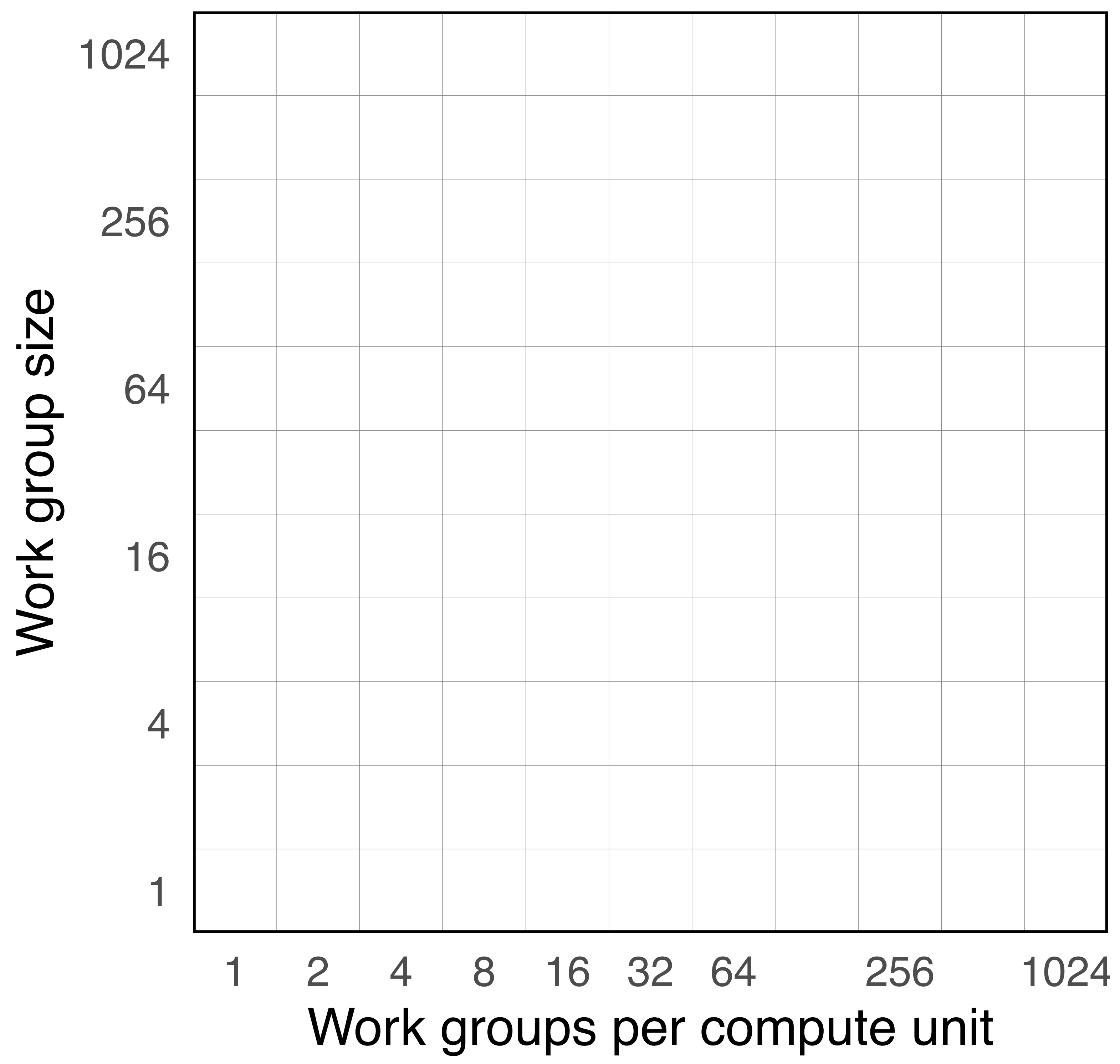
Performance plateaus

Shared aggregation, Tesla K40m, 8 million groups, 128 million input rows, 32-bit keys and values

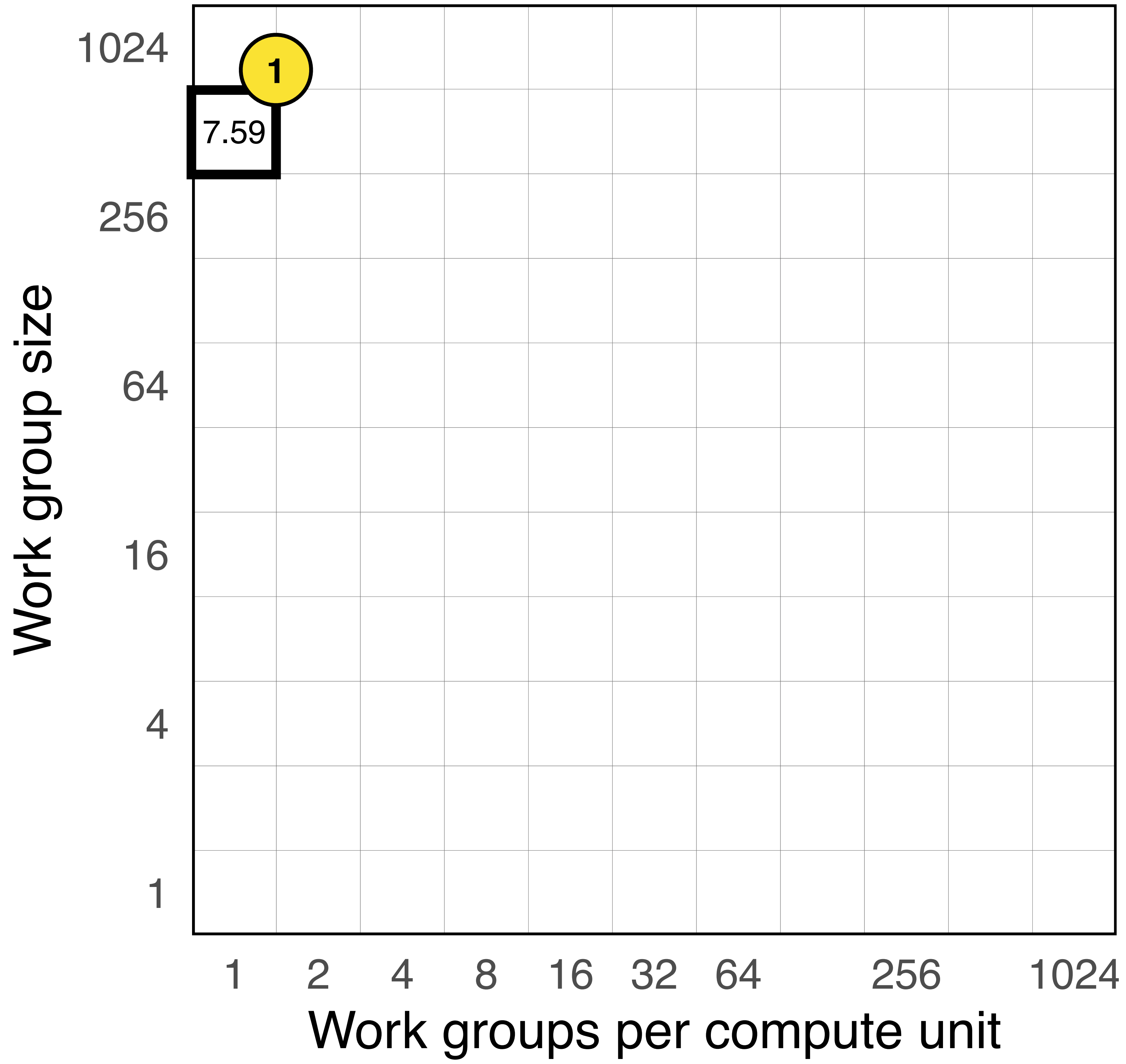




Runtime variation-aware local search

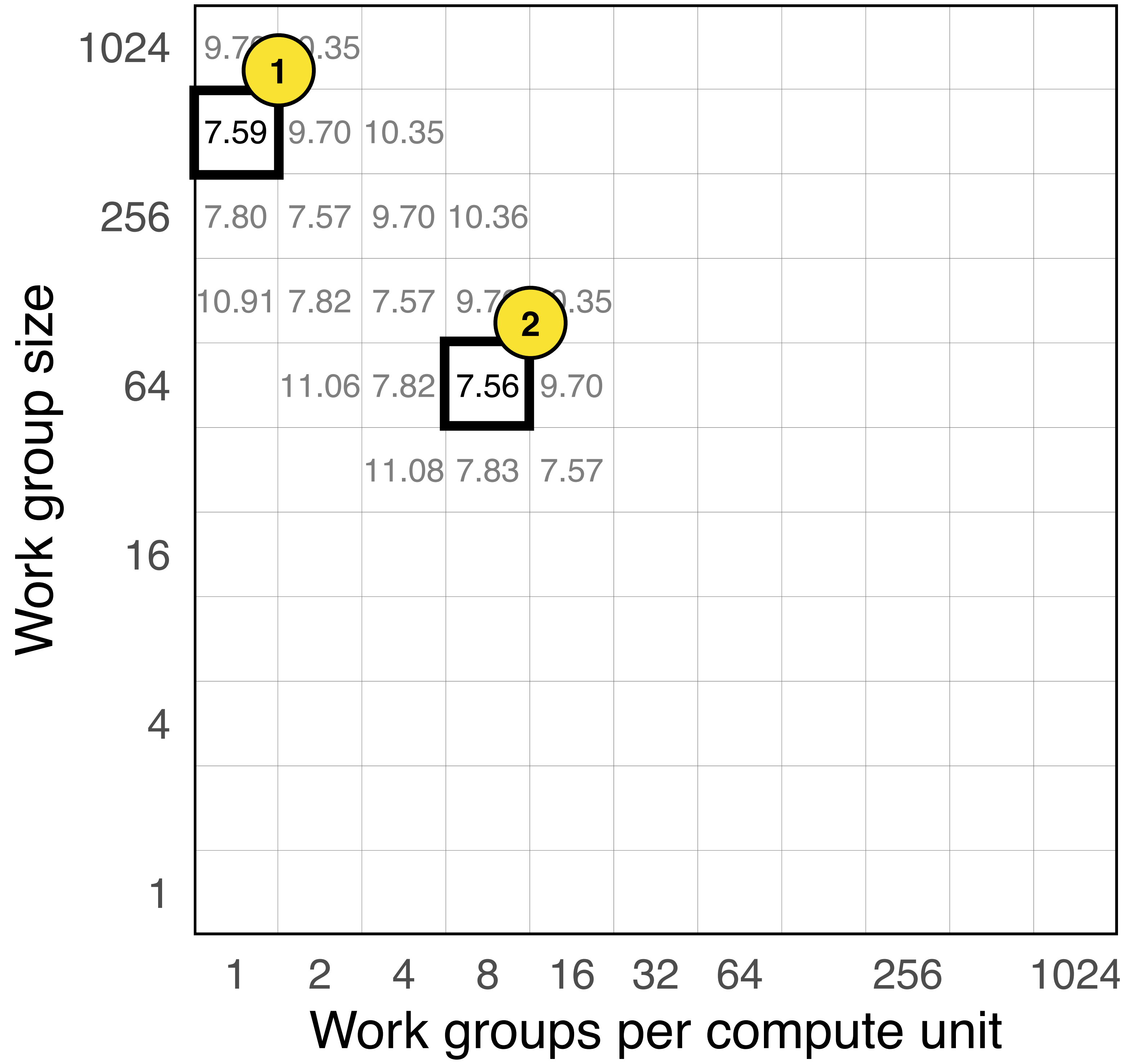


Runtime variation-aware local search



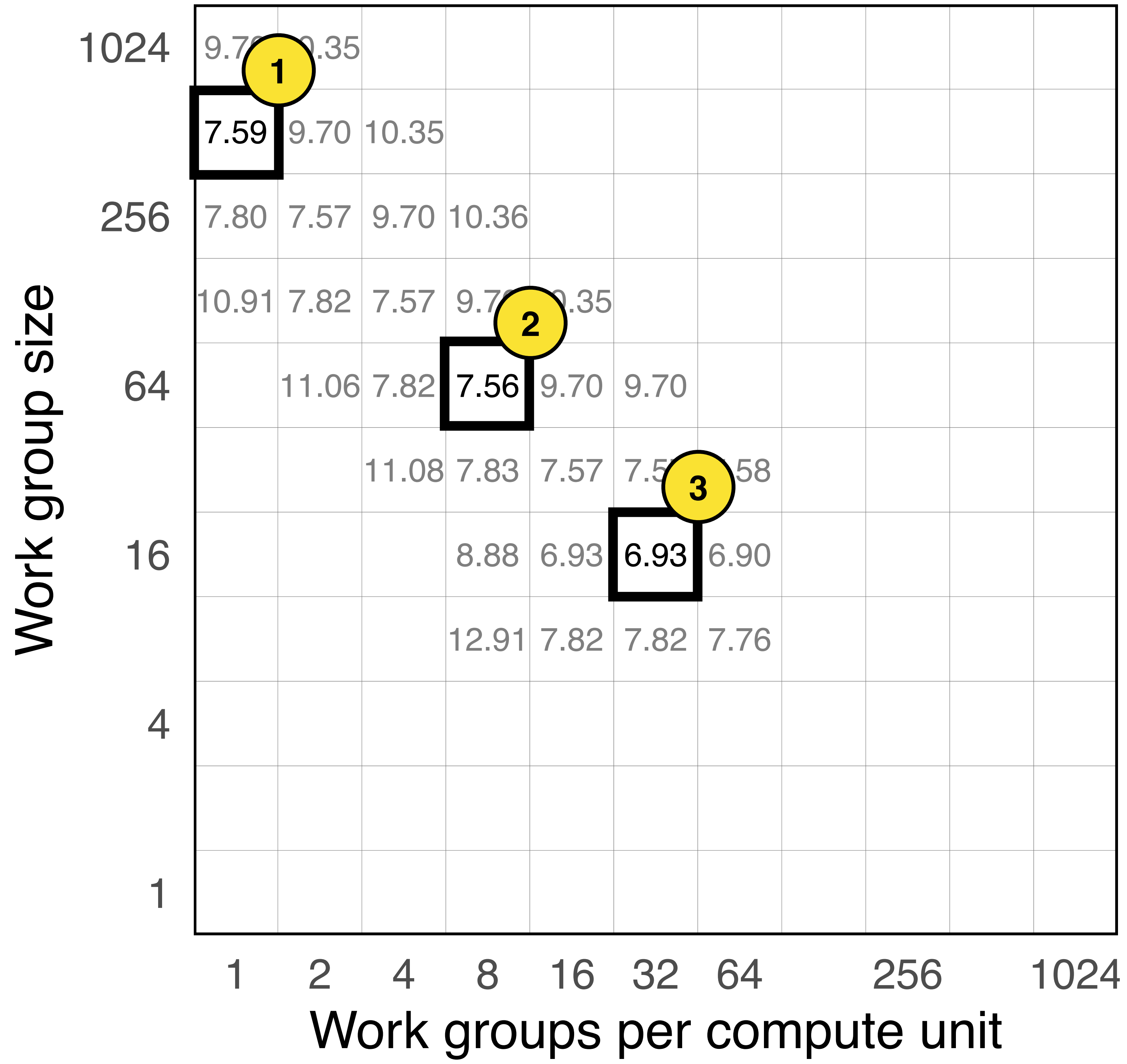
1 Start with initial thread configuration

Runtime variation-aware local search



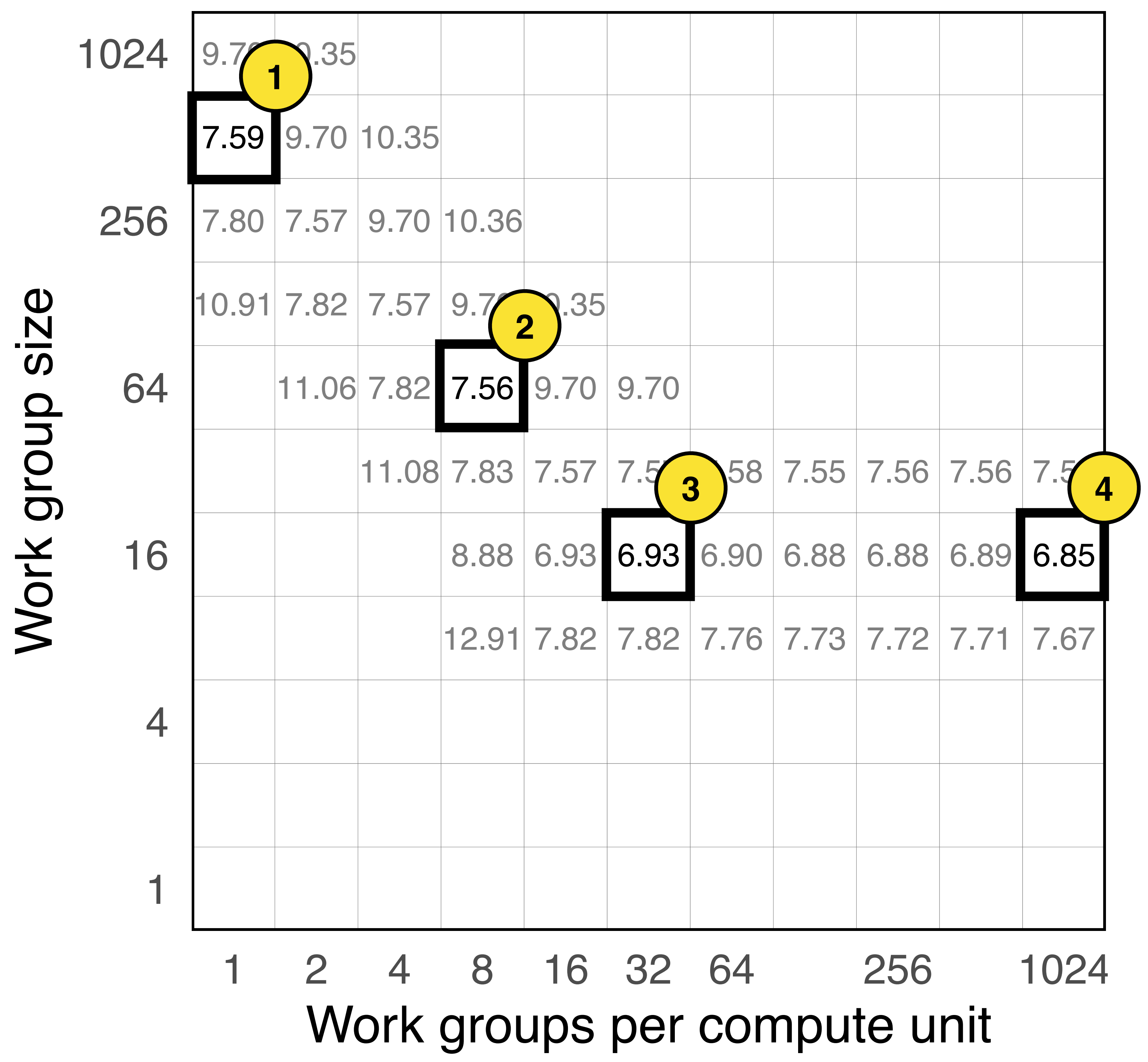
1. Start with initial thread configuration
2. Follow gradient in search space

Runtime variation-aware local search



1. Start with initial thread configuration
2. Follow gradient in search space
3. Push past performance plateaus

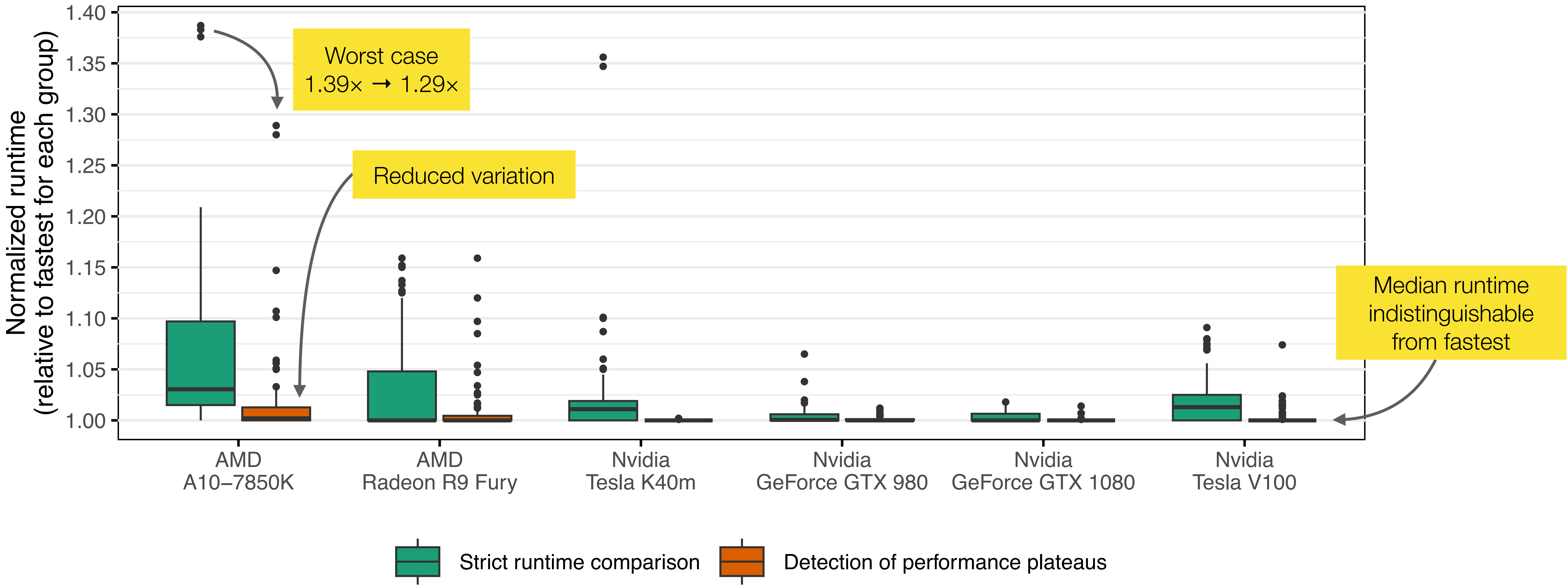
Runtime variation-aware local search



1. Start with initial thread configuration
2. Follow gradient in search space
3. Push past performance plateaus
4. Stop at minimum

Evaluation

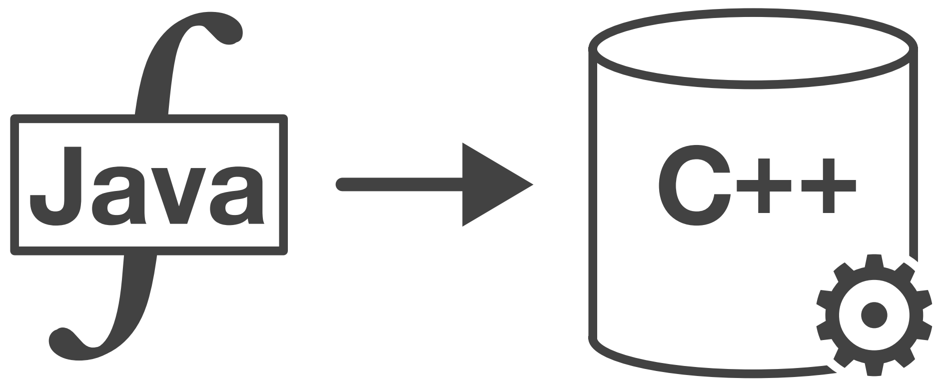
Hash aggregation variants, 128 million input rows, 32-bit keys and values, 3 samples per group cardinality



Exploiting processor and operator characteristics finds fast variants.

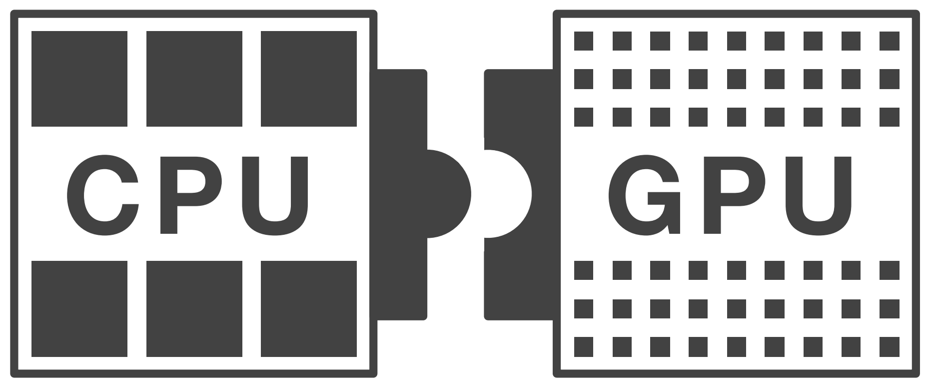
Conclusion

Summary of contributions



Processing Java UDFs in a C++ environment

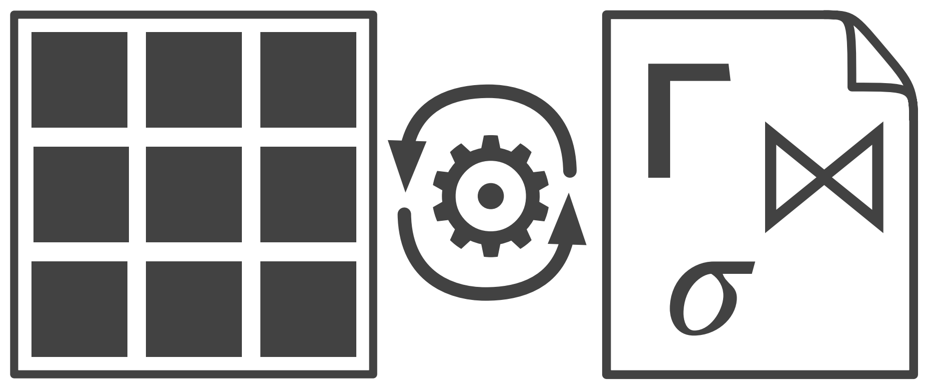
Overcome the JNI overhead when executing Java UDFs



Query Processing on Heterogeneous CPU/GPU Systems

Survey of query processing systems and individual query processing tasks

Classification scheme for workload distribution on heterogenous processors



Operator Variant Tuning on Heterogeneous Processors

Analysis of selection and hash aggregation on heterogeneous processors

Let the database find fast operator implementations automatically