CS 839: Design the Next-Generation Database

Lecture 8: Accelerator

Xiangyao Yu

2/13/2020

Announcements

Next lecture will be a guest lecture given by Dr. Goetz Graefe from Google

Title: Sort-based query processing

Abstract: Since the mid-1980s, common wisdom has favored hash-based algorithms over sort-based algorithms for joins and for duplicate removal. This includes other binary operations, e.g., outer joins and intersection, and other unary operations, e.g., grouping and aggregation. Over the last few years, the presenter has convinced himself of the opposite. He will present some of his considerations and arguments.

Bio: Goetz has worked on database research and product development for many years. He is best known for query processing, which has been deployed in millions of Microsoft SQL Server deployments for over two decades and honored by ACM SIGMOD with the 2017 Edgar F. Codd Innovations Award, and for surveys on query execution, query optimization, sorting, b-tree indexes, concurrency control, logging, and recovery.

Discussion Highlights

Transactions on GPU

- Pros: More parallelism; higher memory bandwidth; good for read-only transactions
- Cons: Limited memory; ACID over SIMT (e.g., logging latency, concurrency control scalability, etc.); scratchpad hard to use;

Overcome GPU problems:

- **Avoiding PCIe bottleneck**: better scheduler for data movement (only warm data in GPU, use GPU when results set if small); GPUDirect; compress data transfer;
- **Handling limited GPU memory**: virtual memory for GPU; HBM to CPU; compress data in GPU; multi-GPU; Hybrid CPU-GPU system;

Heterogeneous hardware

- Opportunities: Workload specific optimizations; minimize data transfer cost; massive parallelism
- Challenges: complex scheduling, load balancing; hard to program; difficult failure handling; complex coordination among devices with difference architecture;

Today's Paper

Q100: The Architecture and Design of a Database Processing Unit

Lisa Wu Andrea Lottarini Timothy K. Paine Martha A. Kim Kenneth A. Ross

Columbia University, New York, NY

{lisa,lottarini,martha,kar}@cs.columbia.edu/tkp2108@columbia.edu

Abstract

In this paper, we propose Database Processing Units, or DPUs, a class of domain-specific database processors that can efficiently handle database applications. As a proof of concept, we present the instruction set architecture, microarchitecture, and hardware implementation of one DPU, called Q100. The Q100 has a collection of heterogeneous ASIC tiles that process relational tables and columns quickly and

It goes on to describe big data analytics as not just important for business, but essential. The article emphasized that analyses must process large *volumes* of a wide *variety*, and at real-time or nearly real-time *velocity*. With the big data technology and services market forecast to grow from $3.2B in 2010 to $16.9B in 2015 [23], and 2.6 exabytes of data created each day [28], it is imperative for the research community to develop machines that can keep up with this data

ASPLOS 2014

Some slides borrowed from the original talk at
http://arcade.cs.columbia.edu/q100-asplos14-slides.pdf

Today’s Agenda

Instruction set architecture

Microarchitecture

Evaluation

Domain-Specific Accelerators

Graphics workloads	->	GPU (Graphics Processing Unit)
Artificial intelligence	->	TPU (Tensor Processing Unit)
Database	->	DPU (Database Processing Unit)

Q100

Accelerator for analytical queries (Not transactions)

Hardware support for relational operators

- Join
- Aggregation
- Sort
- Select

Processing data as streams

Combination of spatial and temporal instructions

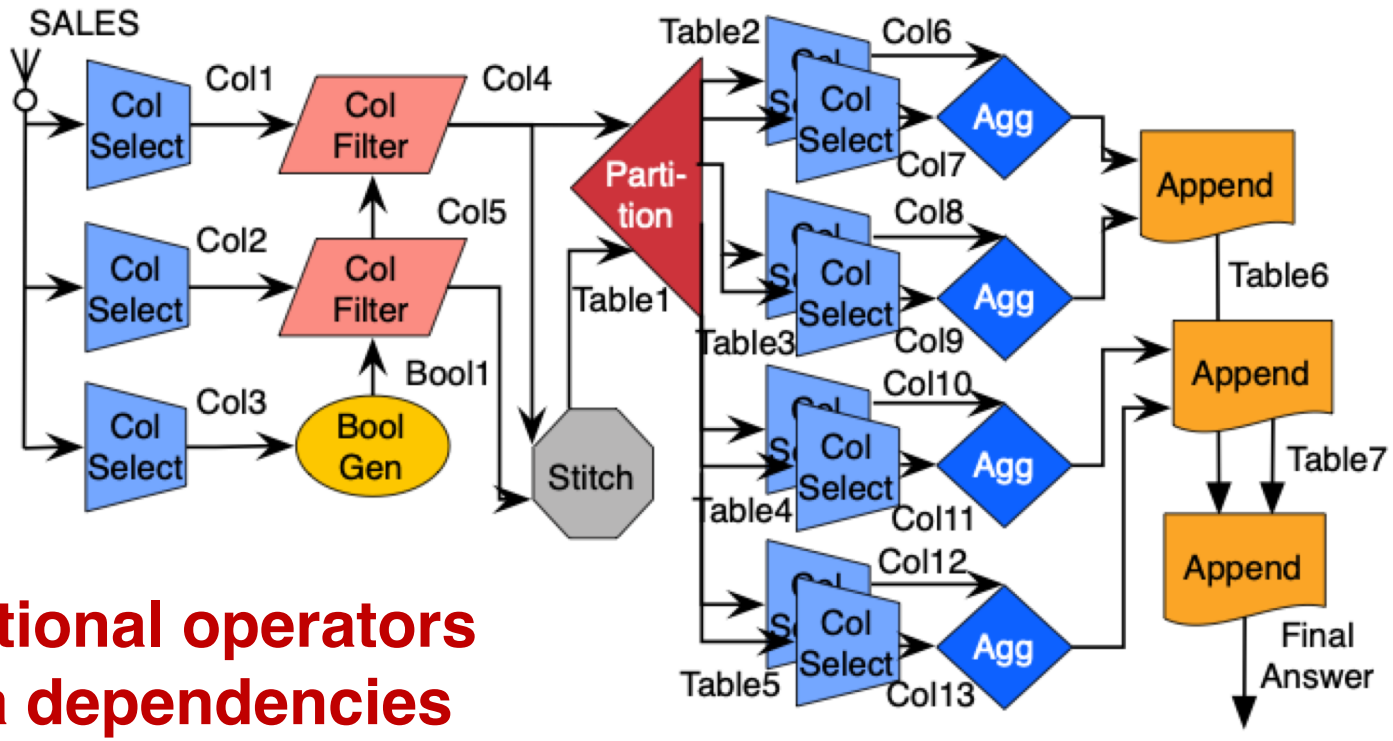
Instruction Set Architecture

Example Query

```
SELECT S_SEASON,
       SUM(S_QUANTITY) as SUM_QTY
FROM   SALES
WHERE  S_SHIPDATE <= '1998-12-01' – INTERVAL '90' DAY
GROUP BY S_SEASON
ORDER BY S_SEASON
```

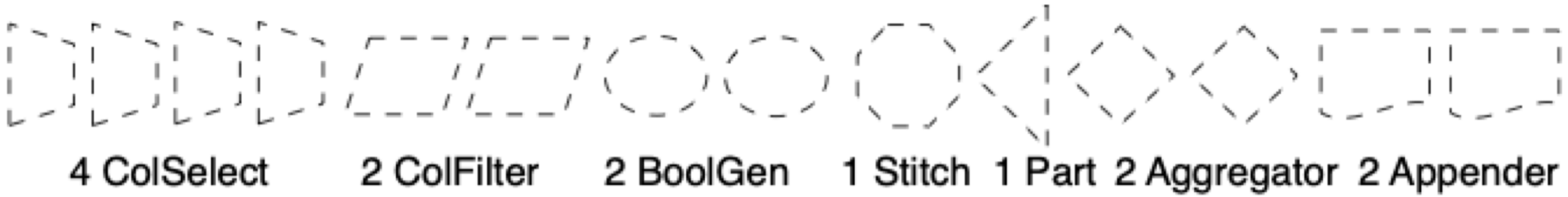
Spatial Instruction Plan

```
SELECT S_SEASON,
       SUM(S_QUANTITY) as SUM_QTY
FROM   SALES
WHERE  S_SHIPDATE <= '1998-12-01' – INTERVAL '90' DAY
GROUP BY S_SEASON
ORDER BY S_SEASON
```



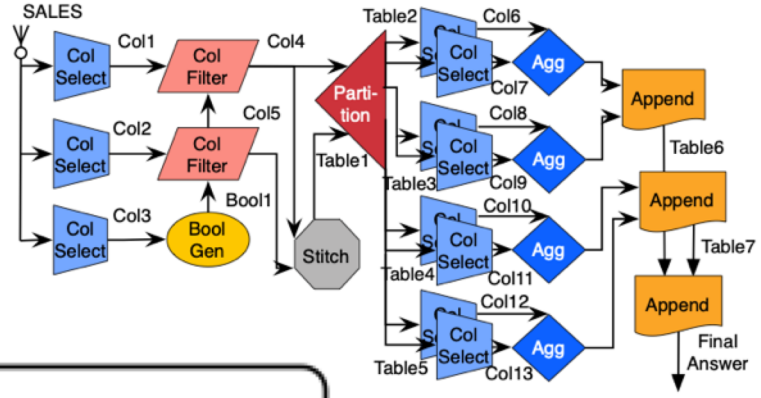
Nodes = relational operators
Edges = data dependencies

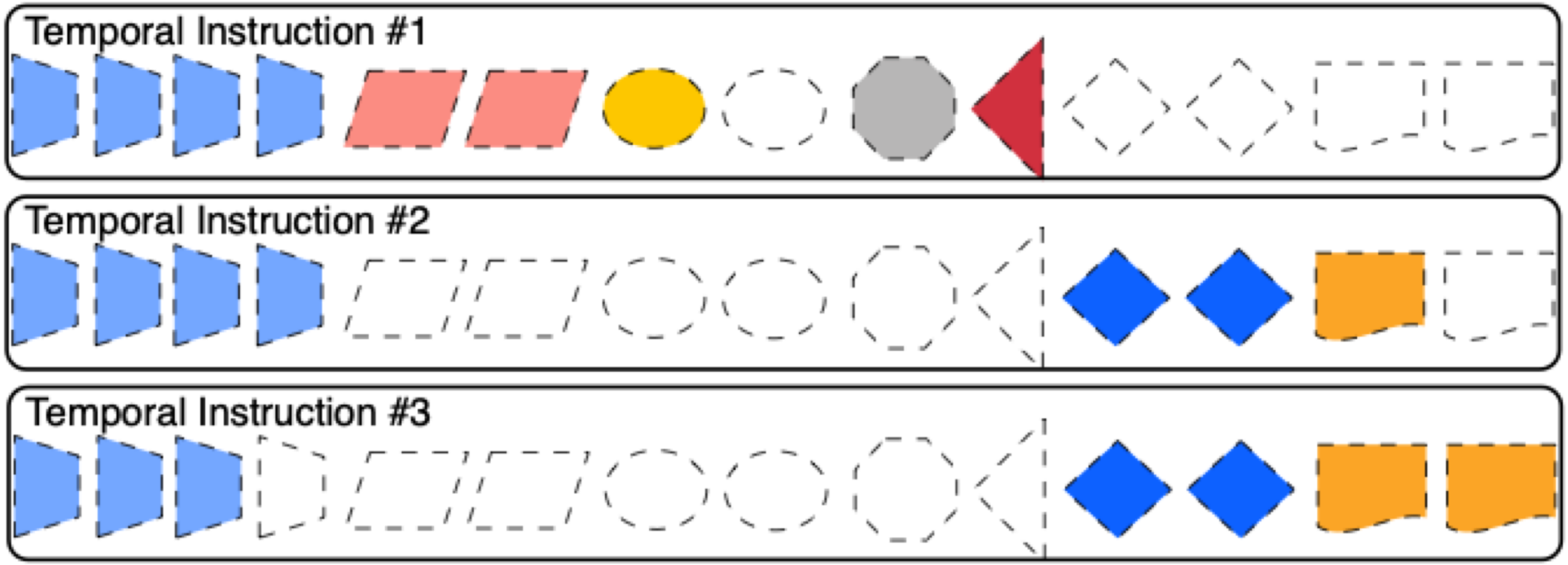
Resource Profile



Main memory

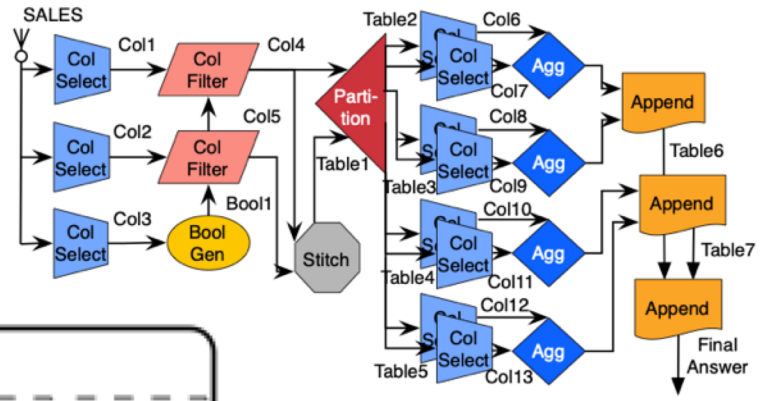
Temporal Instructions



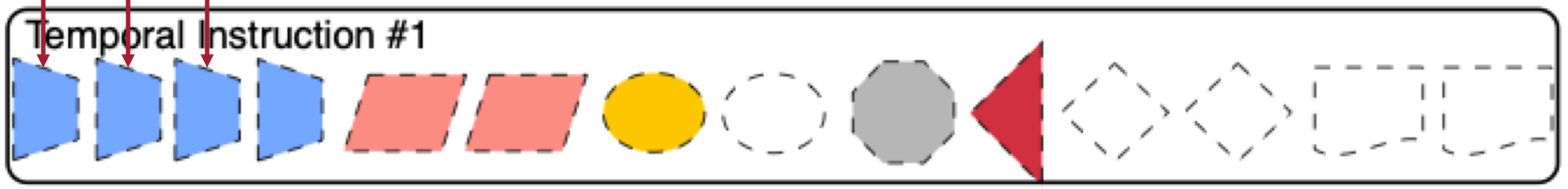


Main memory

Temporal Instructions

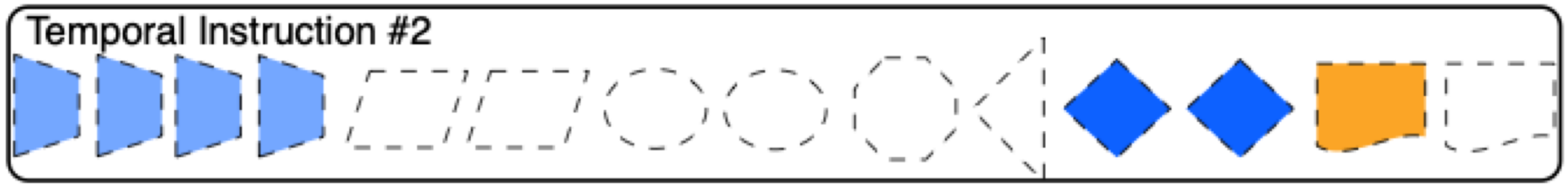


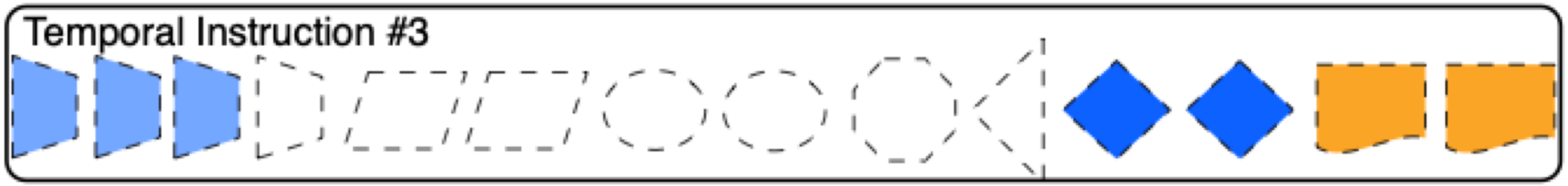
SALES Table



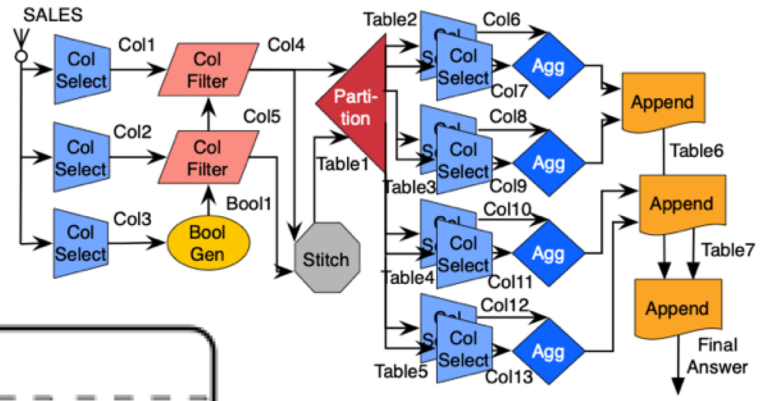
Temp column

Partitioned tables

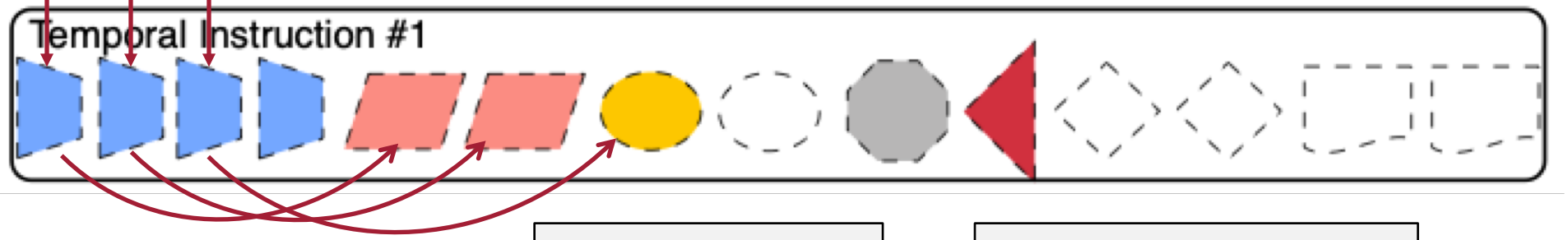




Temporal Instructions

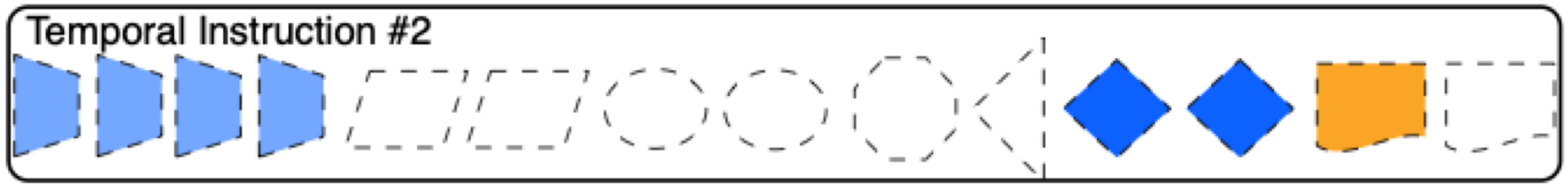


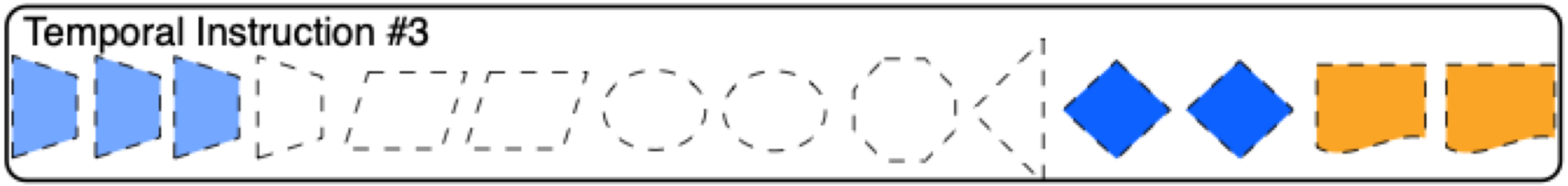
SALES Table



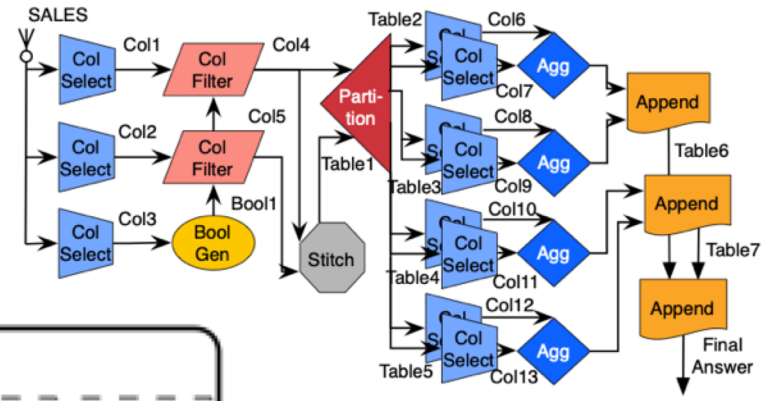
Temp column

Partitioned tables

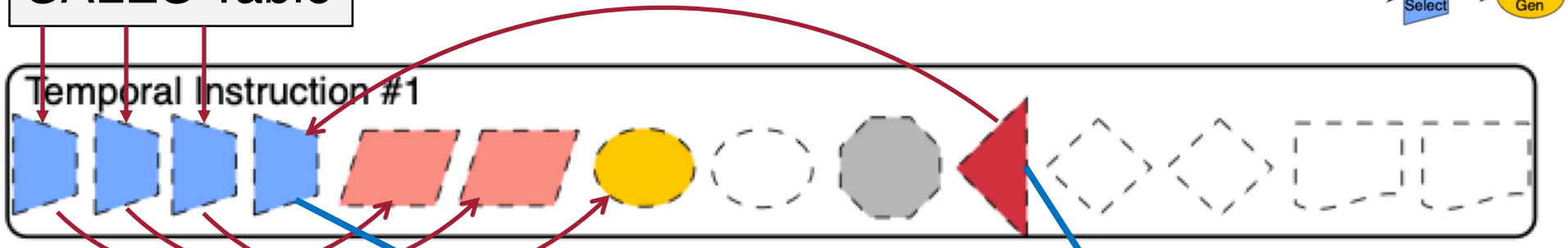




Temporal Instructions

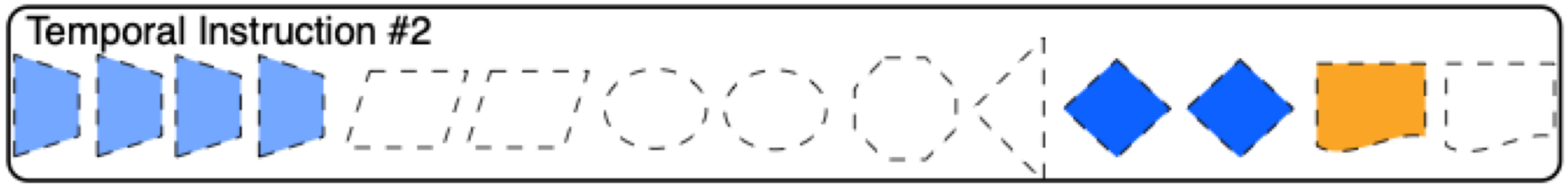


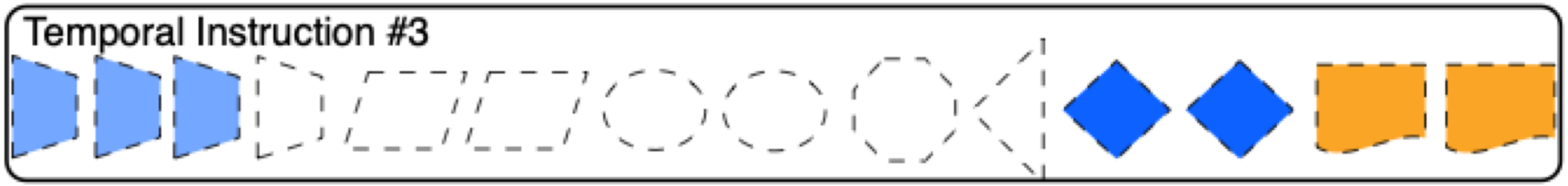
SALES Table



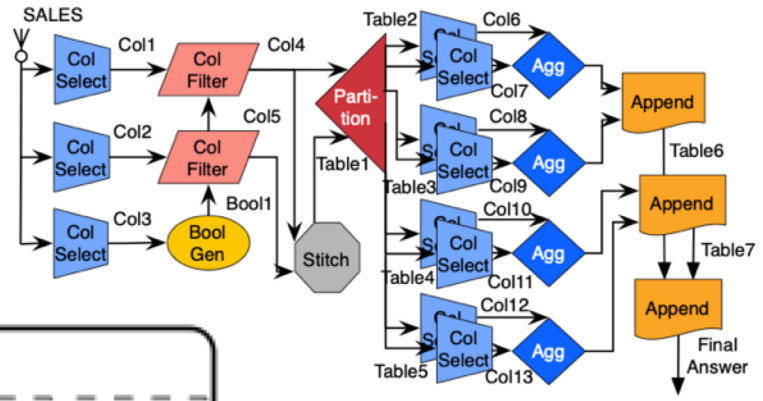
Temp column

Partitioned tables





Temporal Instructions



SALES Table

Temporal Instruction #1

Temp column

Partitioned tables

Temporal Instruction #2

Temporal Instruction #3

Microarchitecture

Functional Tiles

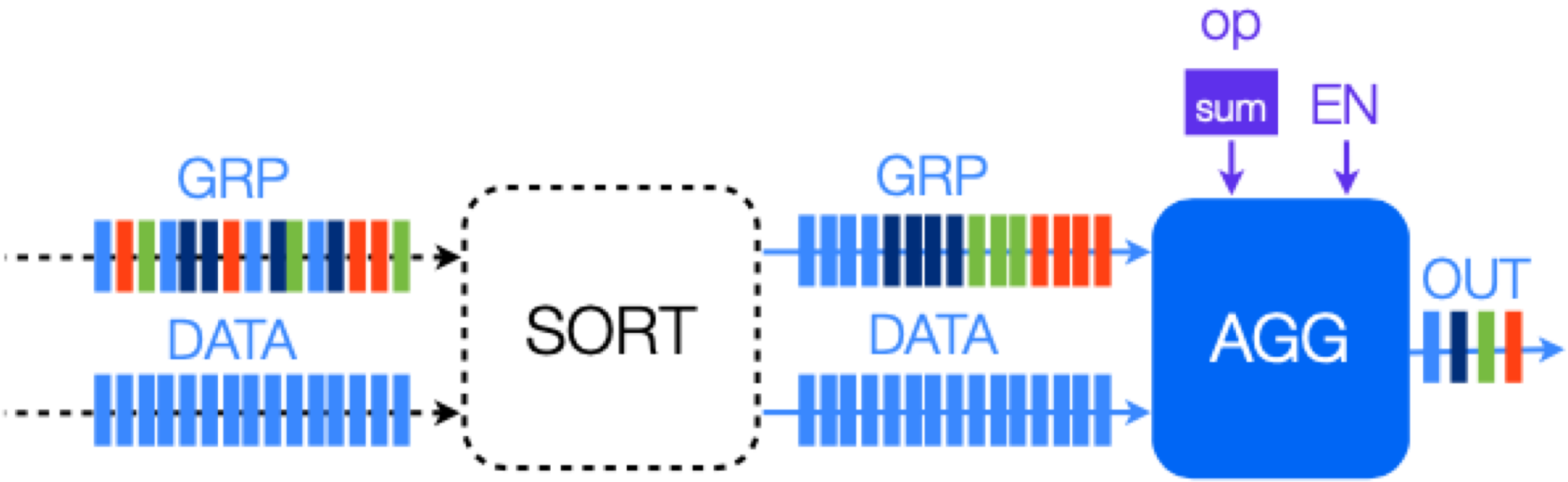
Functional

- Aggregator: both group_by and aggregate columns are sorted
- ALU
- BoolGen: compare two columns and generate bit vector
- ColFilter: select values from a column based on a bit vector
- Joiner: Inner-equijoin (hash or merge join?)
- Partitioner: range-partition input column
- Sorter: bitonic sort for 1024 records

Auxiliary

- Apend: Append two tables with the same schema
- ColSelect: extract column from table
- Concat: concatenate two columns
- Stitch: produce a table based on multiple input columns

Functional Tiles – E.g., Aggregator



Functional Tiles

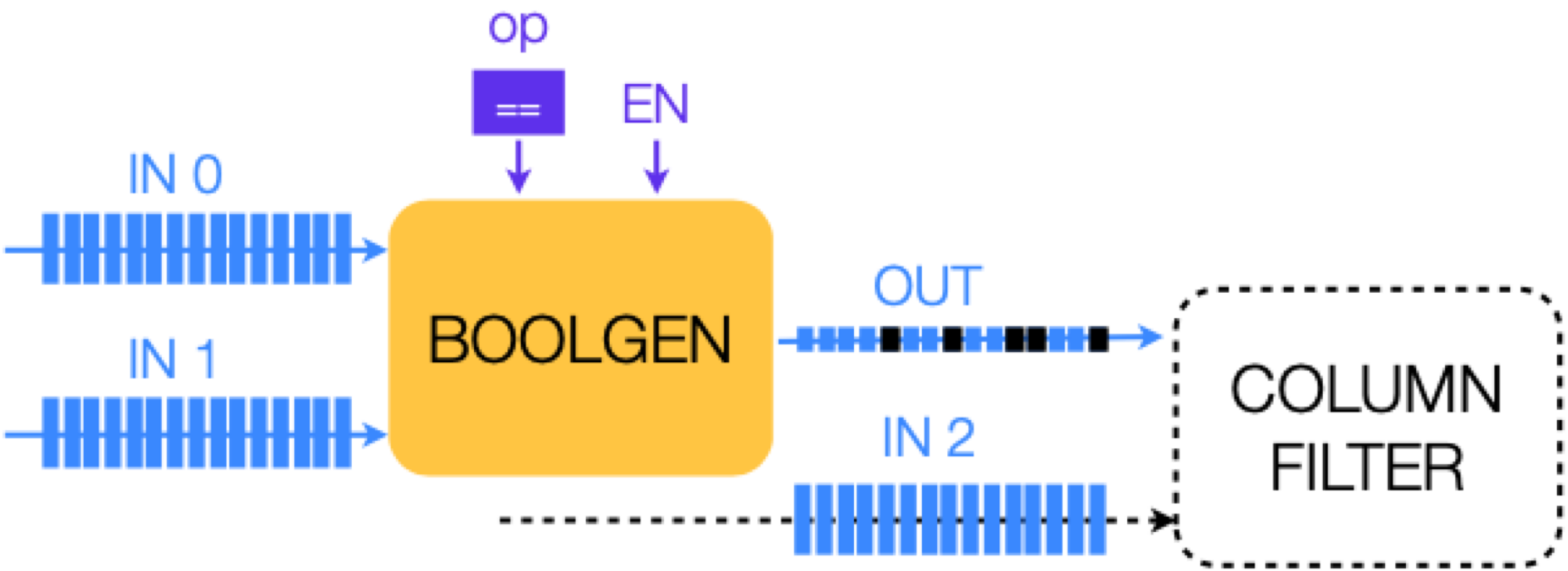
Functional

- Aggregator: both group_by and aggregate columns are sorted
- ALU
- BoolGen: compare two columns and generate bit vector
- ColFilter: select values from a column based on a bit vector
- Joiner: Inner-equijoin (hash or merge join?)
- Partitioner: range-partition input column
- Sorter: bitonic sort for 1024 records

Auxiliary

- Apend: Append two tables with the same schema
- ColSelect: extract column from table
- Concat: concatenate two columns
- Stitch: produce a table based on multiple input columns

Functional Tiles – E.g., Boolean Generator



WHERE s_shipdate >= '2013-01-01'

Functional Tiles

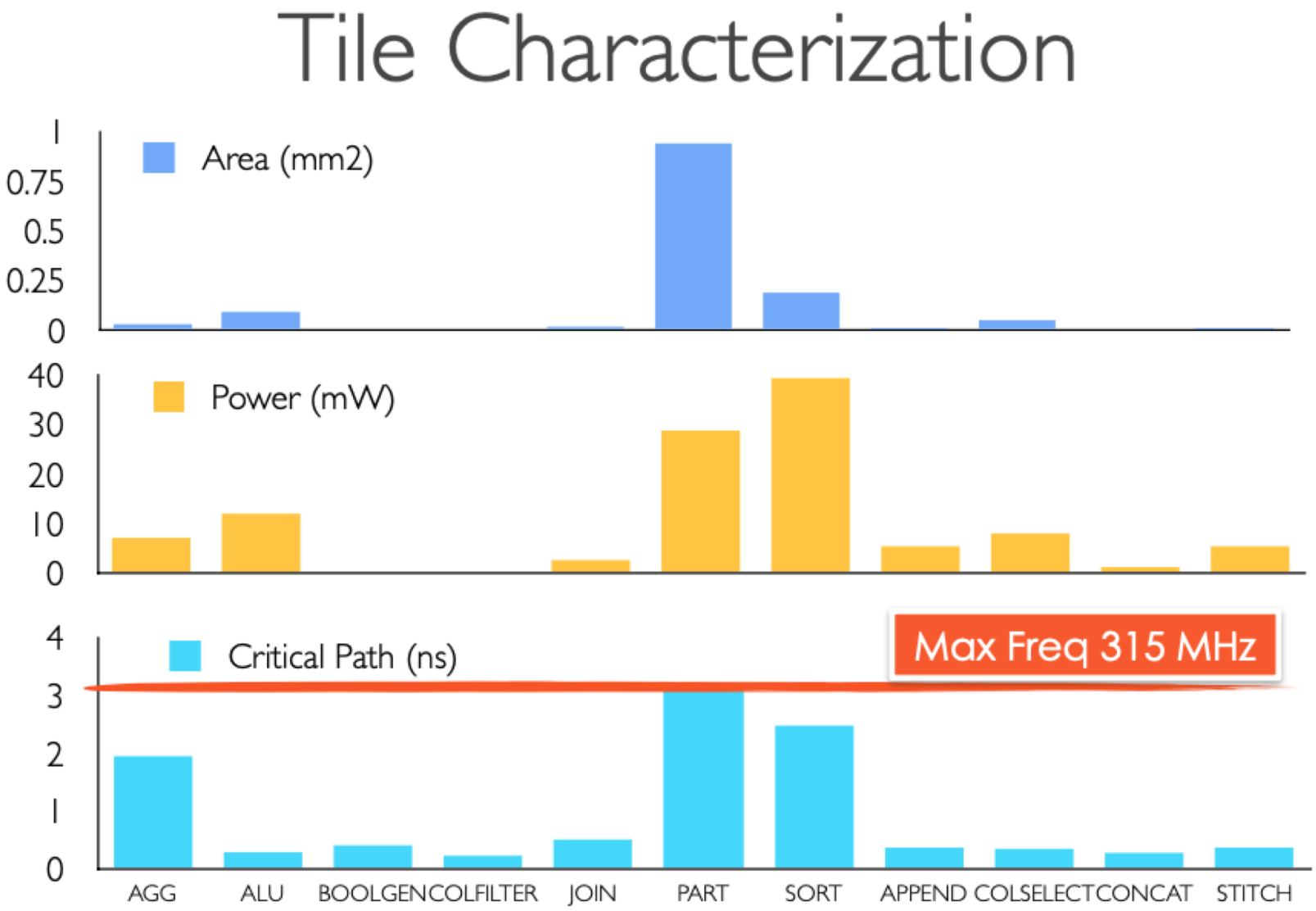
Functional

- Aggregator: both group_by and aggregate columns are sorted
- ALU
- BoolGen: compare two columns and generate bit vector
- ColFilter: select values from a column based on a bit vector
- Joiner: Inner-equijoin (hash or merge join?)
- Partitioner: range-partition input column
- Sorter: bitonic sort for 1024 records

Auxiliary

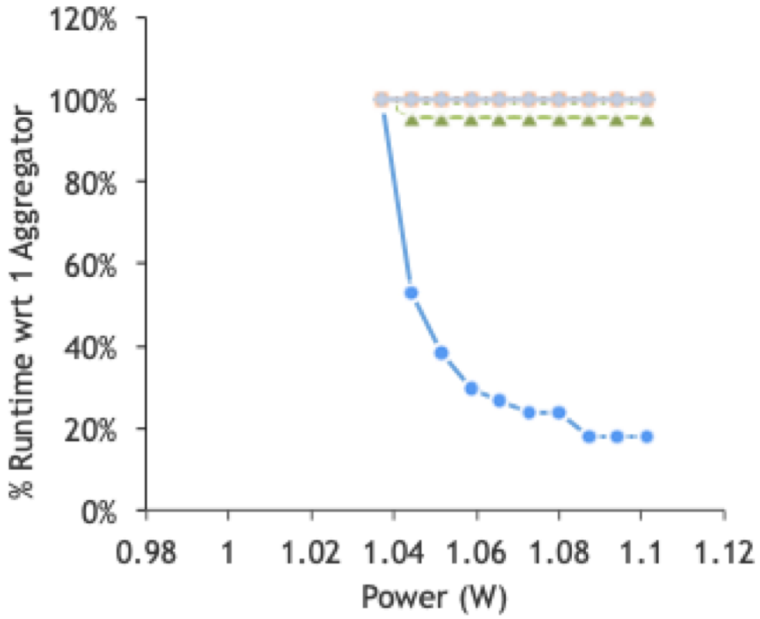
- Apend: Append two tables with the same schema
- ColSelect: extract column from table
- Concat: concatenate two columns
- Stitch: produce a table based on multiple input columns

Tile Characterization

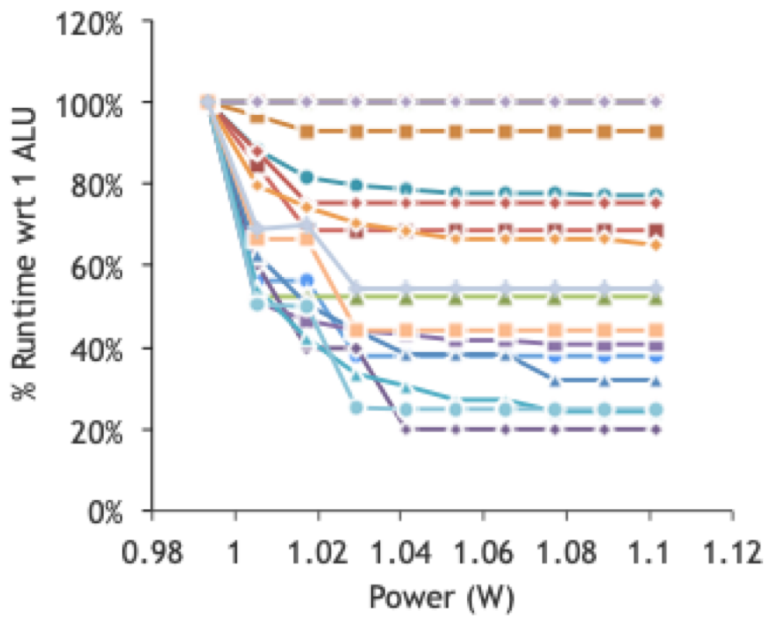


Tile Count Sensitivity

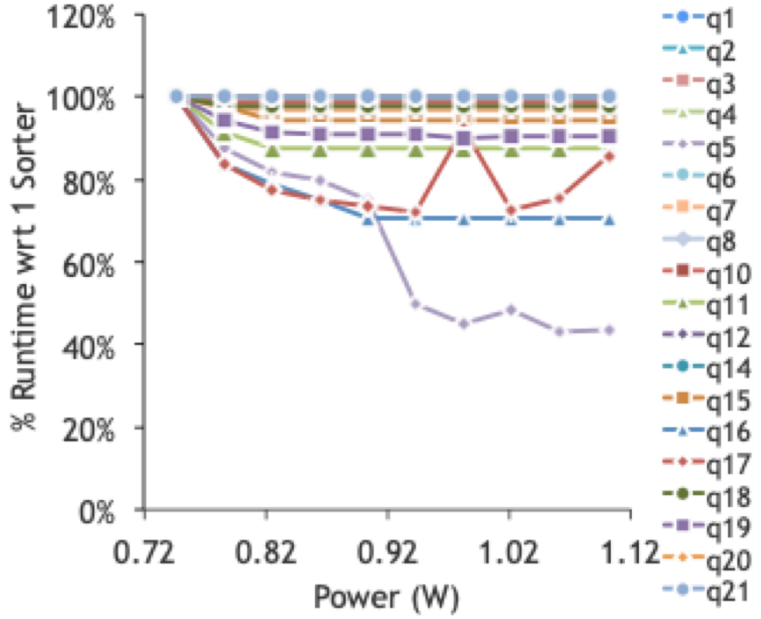
How many tiles do we need?



Aggregator

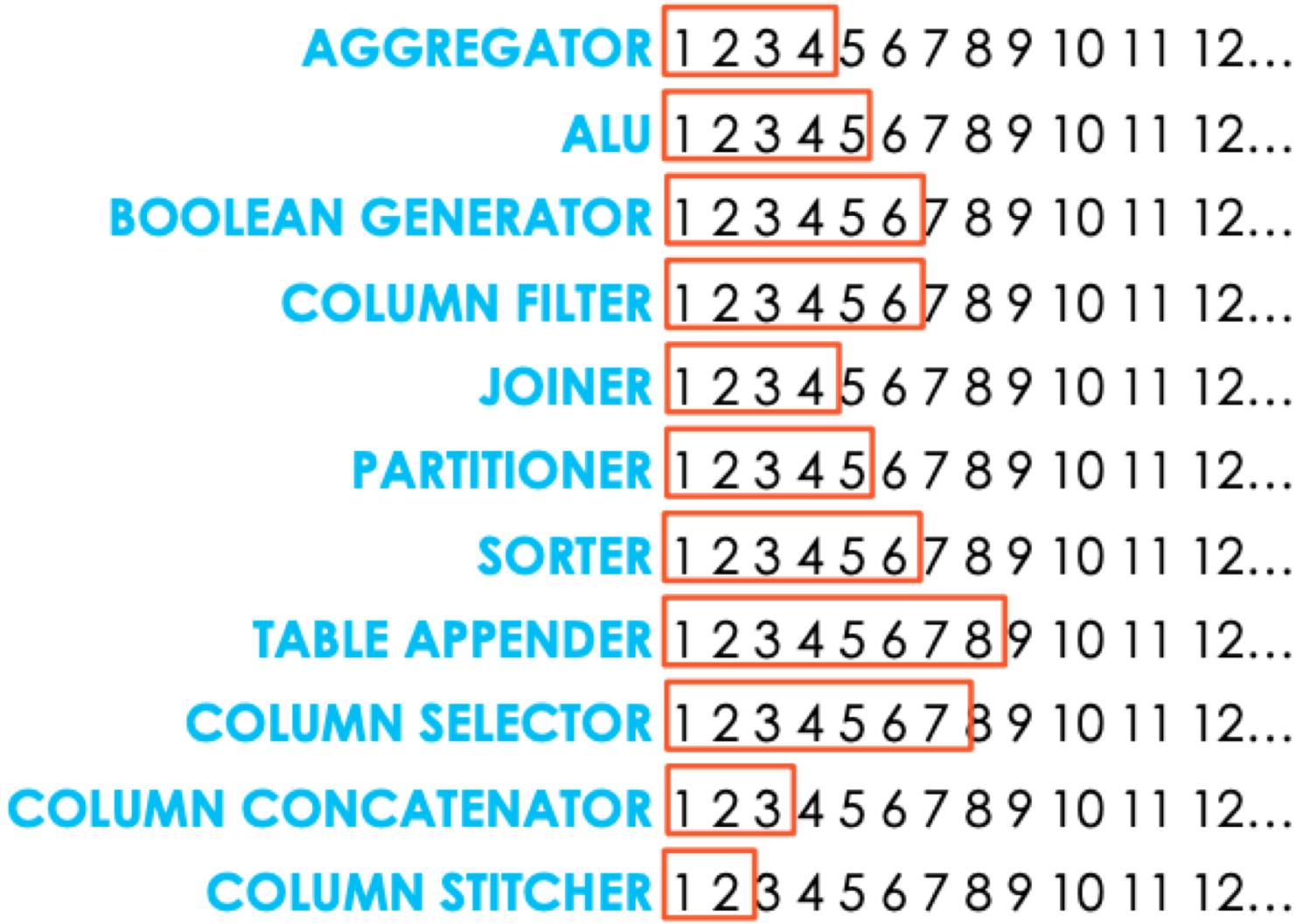


ALU



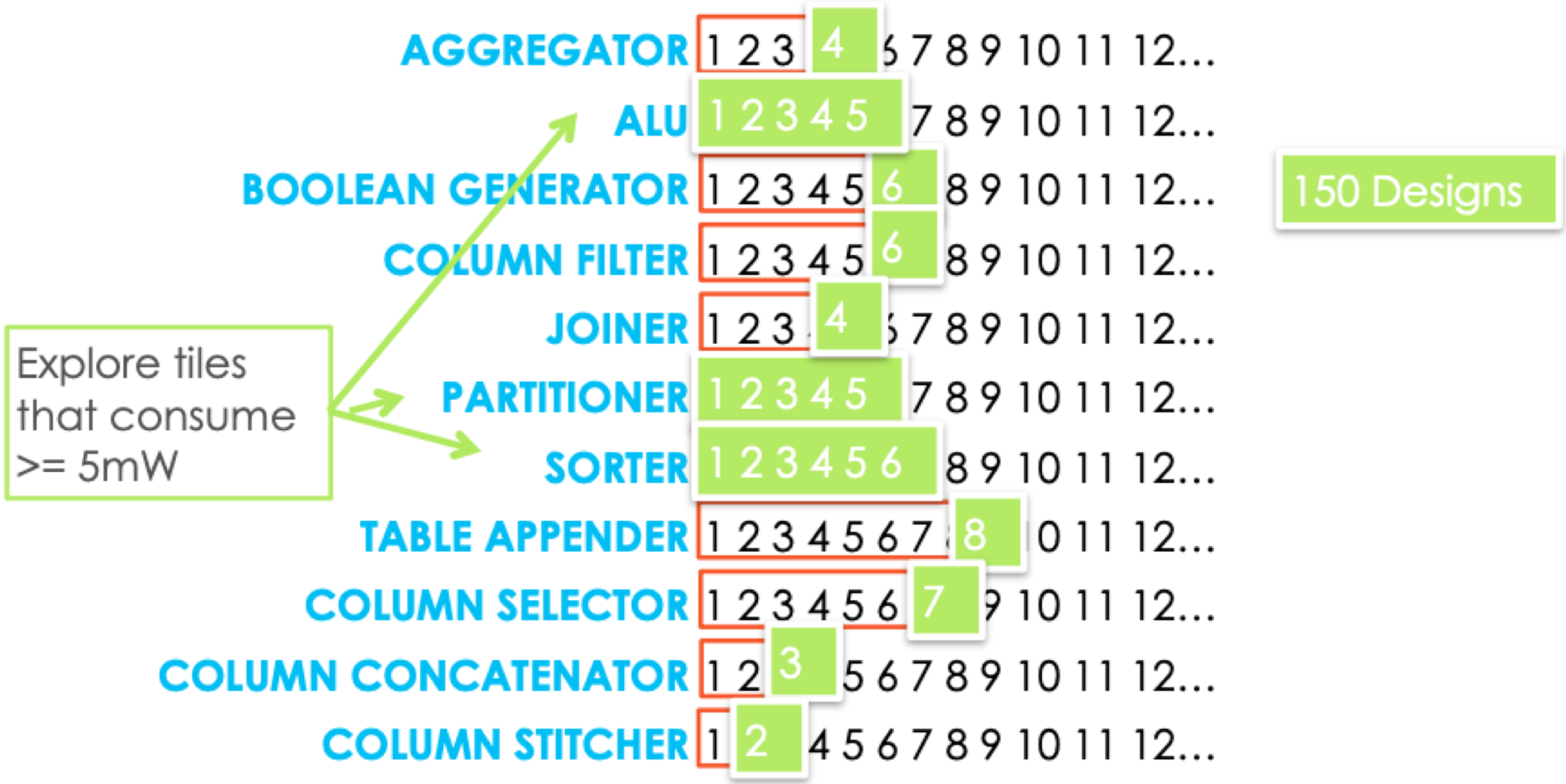
Sorter

Bounded Design Space

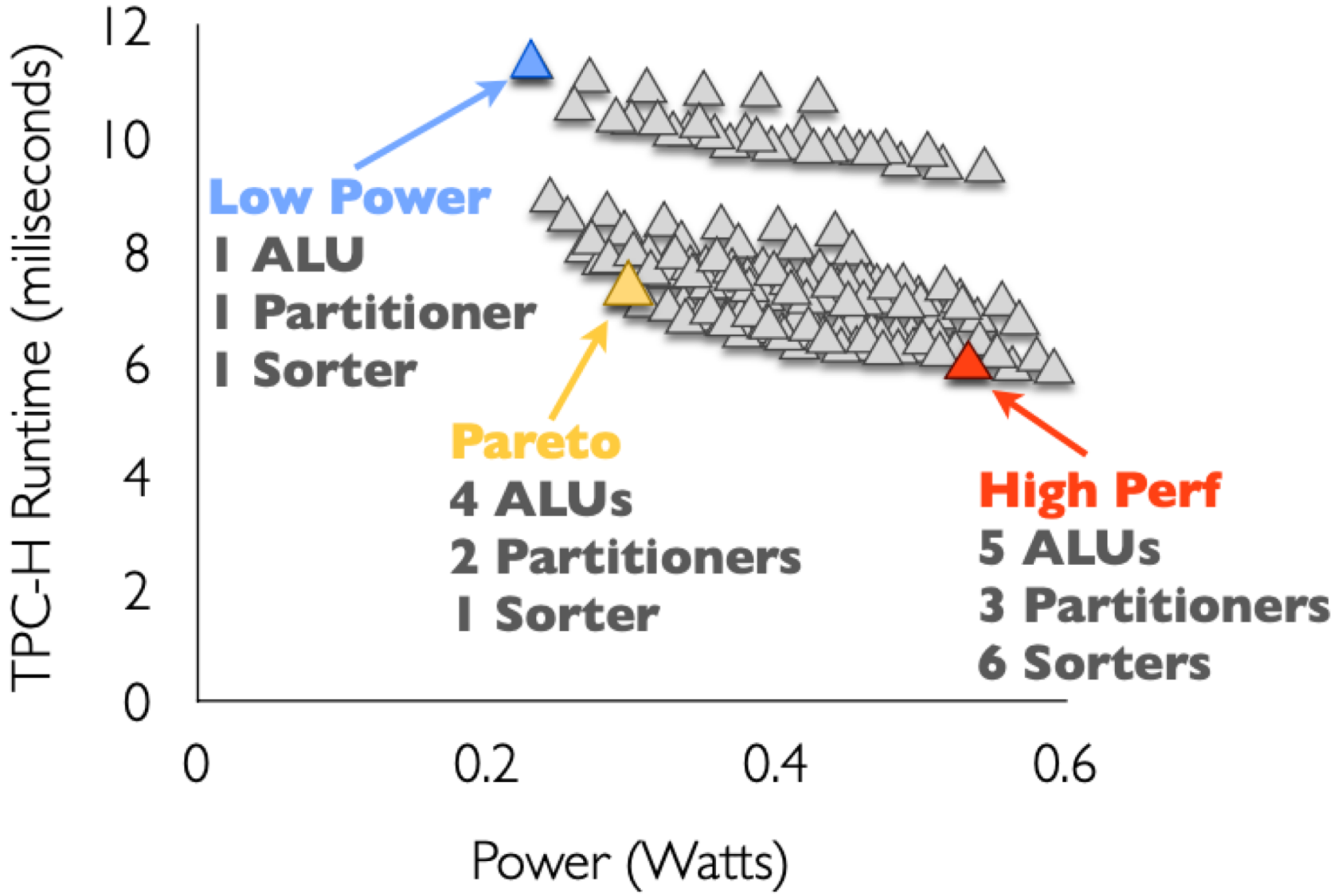


2.9 Million Designs!!

Bounded Design Space

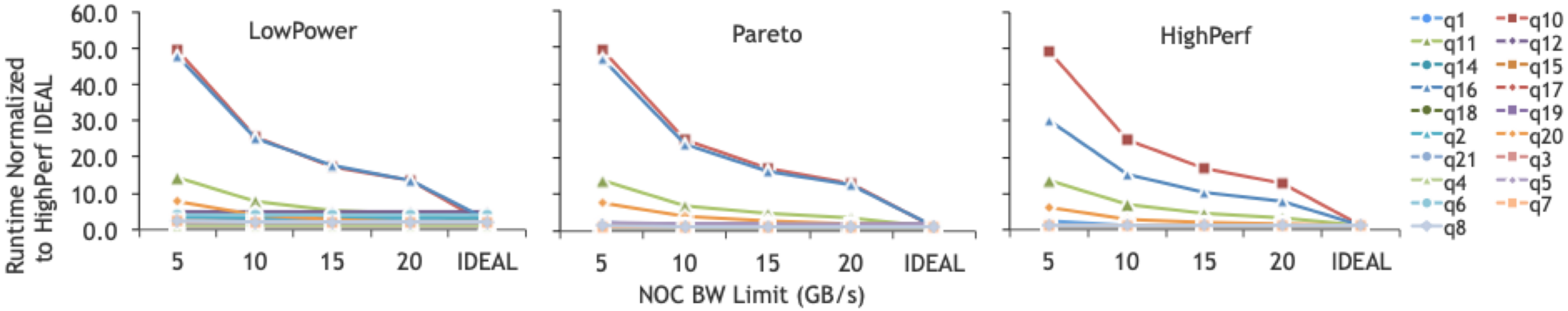


Designs for Further Evaluation

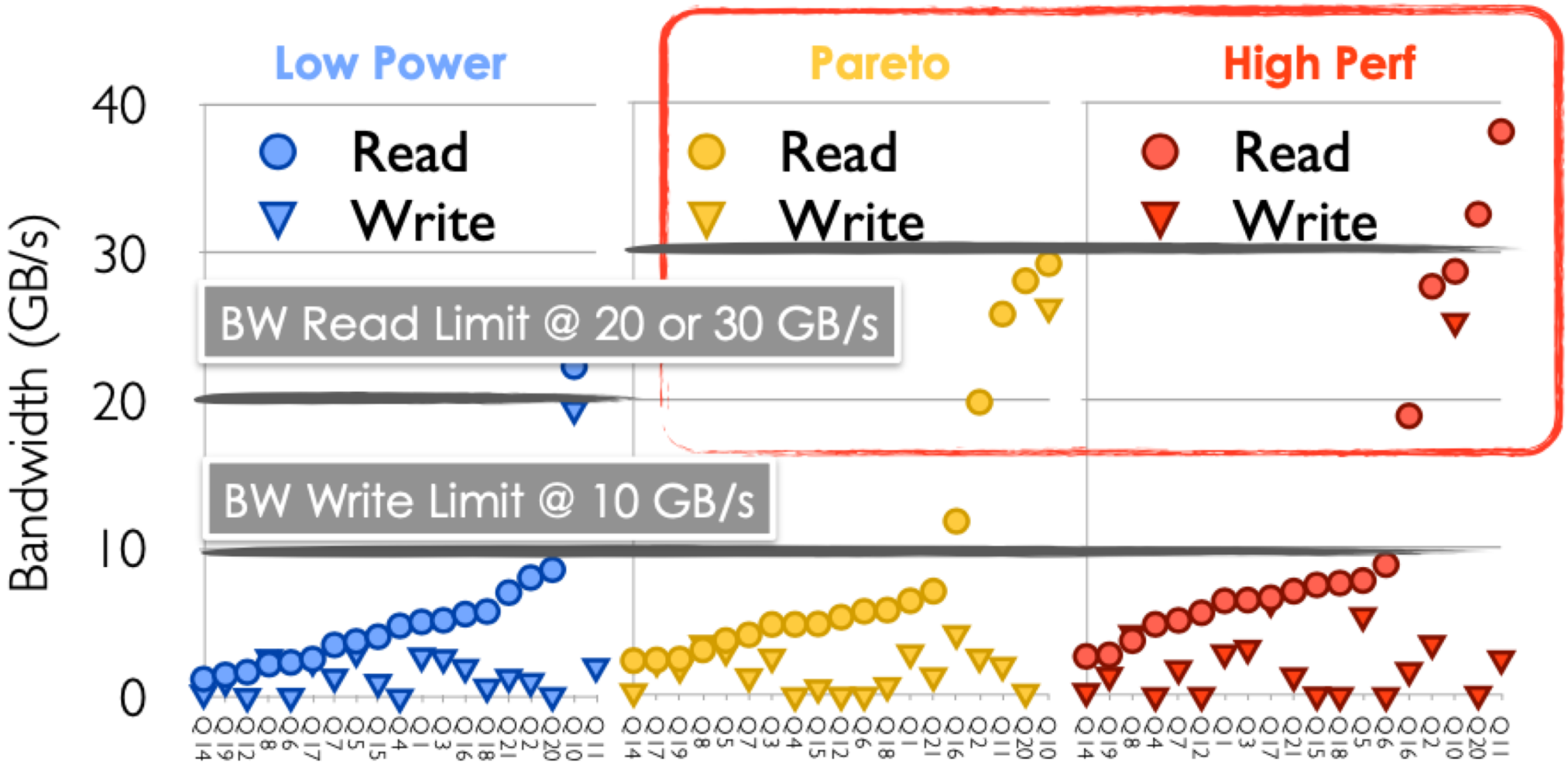


Network on Chip (NoC) Bandwidth

Performance sensitivity to NoC bandwidth
NoC limit @ 6.3 GB/s (scaled down from Intel TeraFlop)



Memory Bandwidth



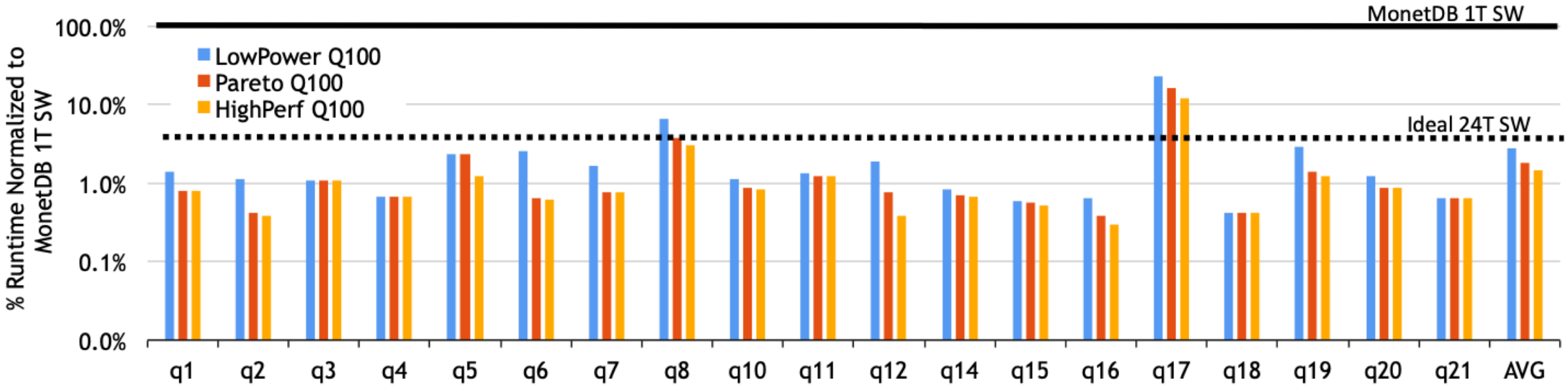
Evaluation

Methodology

System Configuration	
Chip	2X Intel E5-2430 6C/12T, 2.2 GHz, 15 MB LLC
Memory	32 GB per chip, 3 Channels, DDR3
Max Memory BW	32 GB/sec per chip
Max TDP	95 $Watts$ per chip
Lithography	32 nm

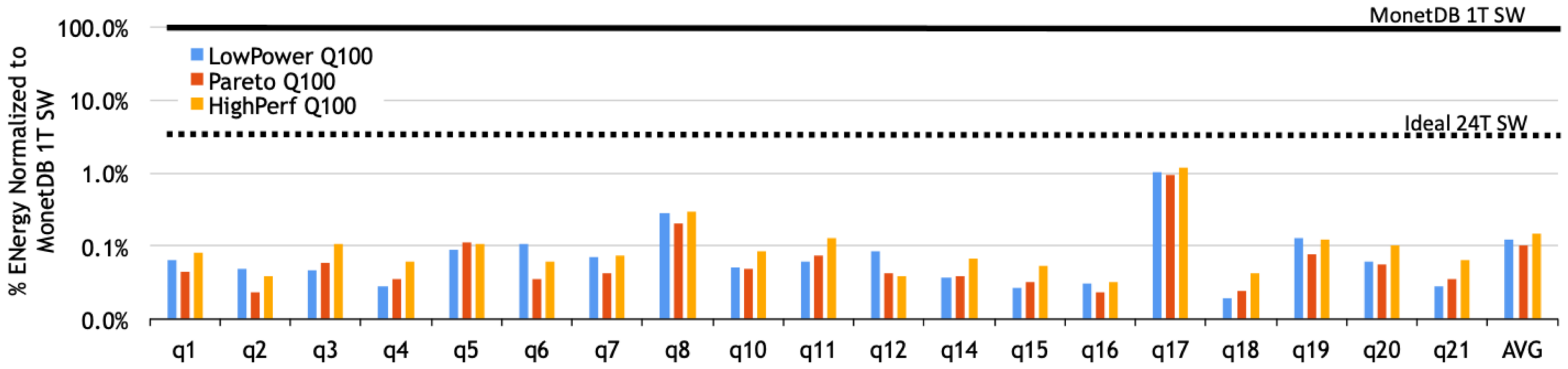
- Used in-house simulator to model the performance and energy consumption of Q100

Q100 vs. MonetDB – Runtime



37X – 70X performance improvement over single-threaded MonetDB

Q100 vs. MonetDB – Energy



691X – 983X energy efficiency improvement over single-threaded MonetDB

Summary

Q100 is an efficient domain-specific accelerator for analytical database workloads

ISA exploits parallelism and streaming efficiencies

At < 15% area and power of a Xeon core, a Q100 device gets exceptional performance and energy efficiency

Accelerators – Q/A

Why is Q100 more energy efficient?

What's the state-of-the-art in this space? Is Q100 commercially available?

Why not explore all possible software solutions first?

Group Discussion

From the hardware perspective, what are the key ideas that lead to the speedup of Q100 over MonetDB? In your opinion, how would Q100 compare to an optimized CPU-based database?

What are the limitations of Q100?

Based on this lecture (Q100) and last lecture (GPU), to build a high performance database accelerator, what are the most important optimization goals? (e.g., computation power, DRAM bandwidth, etc)

Before Next Lecture

Submit discussion summary to https://wisc-cs839-ngdb20.hotcrp.com

- **Deadline: Wednesday 11:59pm**

Submit review for

- [optional] New algorithms for join and grouping operations
- [optional] Modern B-tree techniques