Lecture 23: Vectorized Execution & Course Retrospective


Logistics

- No class on **Nov 18**
- Project final reports (2+ pages with GitHub link) due on **Nov 18**
  - **5% extra credits for all non-trivial submissions**
- In-class presentations on **Nov 20**
  - **10% extra credit for those selected for in-class presentations**

Recap

- Columnar Storage
- Compression
- Compressed Columnar Storage

Lecture Overview

- Vectorized Execution
- Course Retrospective

Vectorized Execution

Limitations of Tuple-at-a-time Processing

- Each tuple incurs the cost of:
  - Function calls between operators.
  - Deserializing, interpreting, and processing the tuples.
  - Doesn't leverage the CPU's ability to process batches of data efficiently.
  - Results in frequent pipeline stalls and cache misses.

Vector-at-a-Time Processing

SIMD Instructions

Limitations of Tuple-at-a-time Processing

FUNCTION CALLS

NO DATA-LEVEL PARALLELISM

PIPELINE STALLS + CACHE MISSES

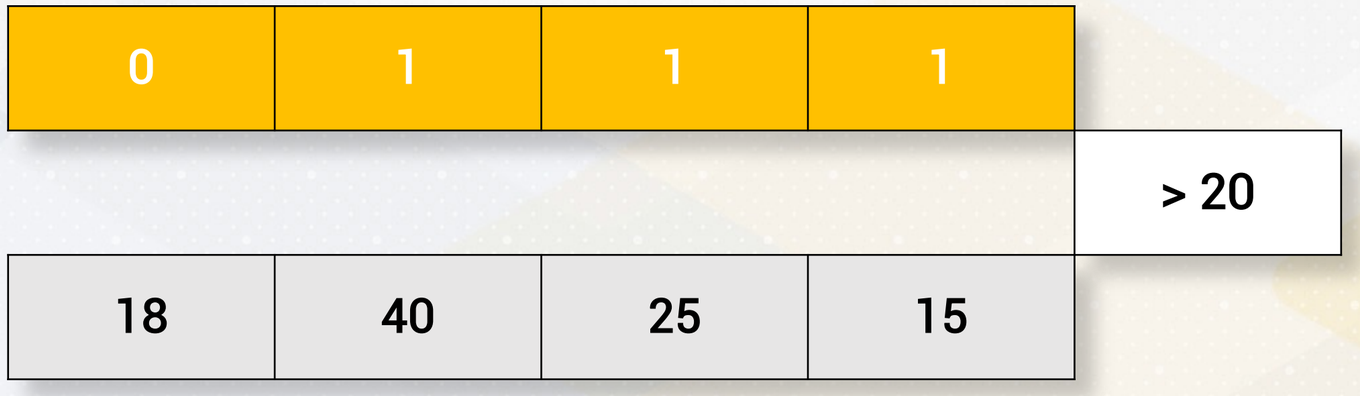
COLUMN VALUE	18	40	25	15
CONSTANT	> 20	> 20	> 20	> 20

Vector-at-a-time Processing

Process a vector of tuples at a time to reduce overhead of function calls

Use SIMD (Single Instruction, Multiple Data)

A single instruction operates on multiple data points simultaneously



SIMD in Query Execution

SIMD Use Cases

- Filtering
- Aggregations
- Compression

SIMD Operations

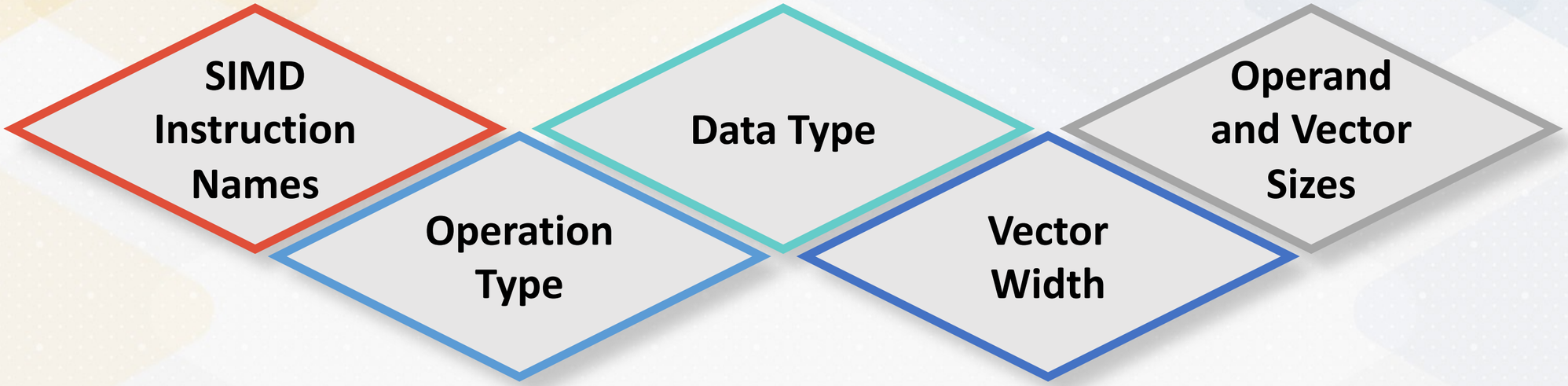
- Vector loads
- Filter masks
- Horizontal reduction

Filter timestamps using SIMD

```
int32x4_t ts_vec = vld1q_s32(&data.timestamps[i]); // Load timestamps


uint32x4_t mask = vandq_u32(vcgeq_s32(ts_vec, 1),
                            vcleq_s32(ts_vec, end)); // Mask for range
```

SIMD Instruction Naming



vaddq_f32	Addition Operation	32-bit floating-point numbers	Quad-word (4 floats)
vcgeq_s32	Compare greater than or equal to	Operates on signed 32-bit integers.	64-bit vector (2 integers)

Filter timestamps using SIMD

vld1q_s32	18	40	25	15	
vcgeq_s32					> 20
	0	1	1	1	
vcleq_s32					< 30
	1	0	1	1	
vandq_u32					> 20 AND < 30
	0	0	1	0	

Scalar vs SIMD Execution

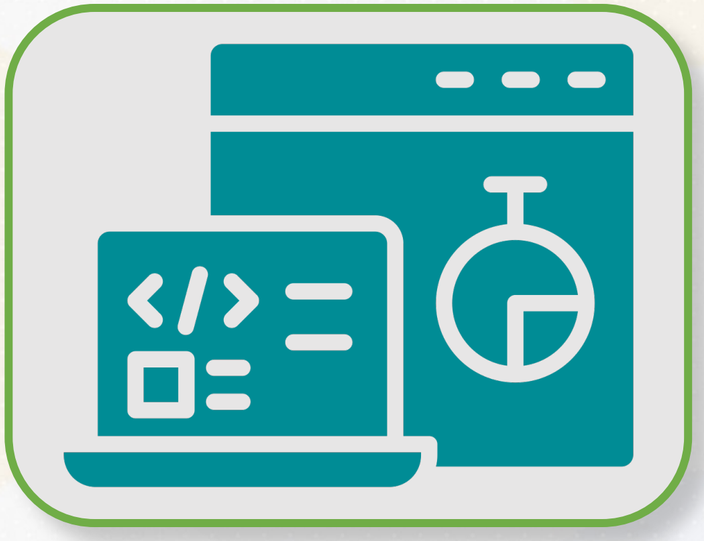
Scalar Execution

Repeated Instruction for Each Element

SIMD Execution

Batch Execution with Single Instruction

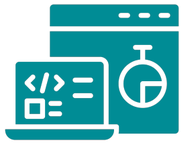
Benefits of SIMD Execution #1



- Instruction-Level Efficiency
- Vectorized Instructions
- N instructions for N data points
- N/W instructions for SIMD Vector

Benefits of SIMD Execution #2

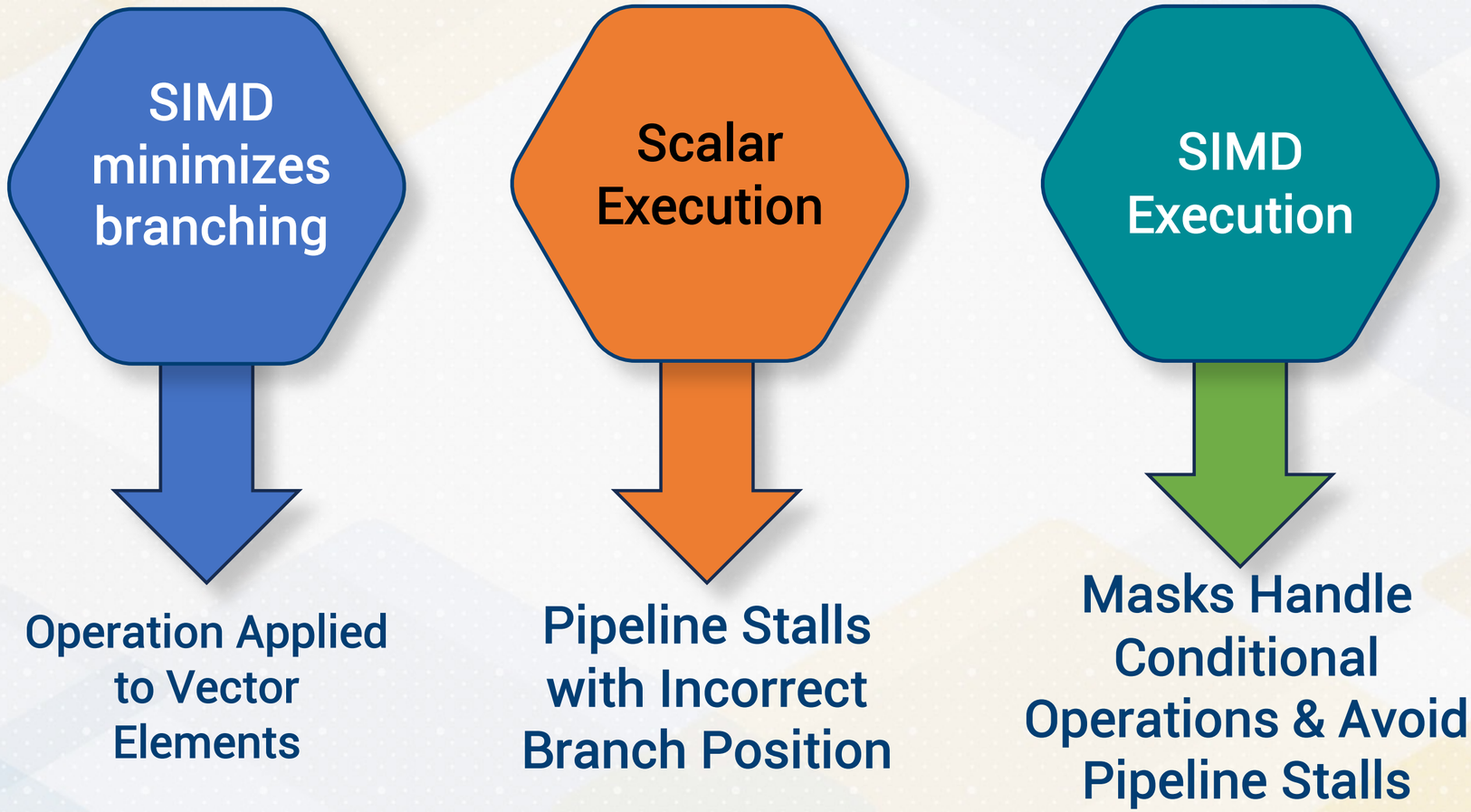
Contiguous Memory



SIMD Operations

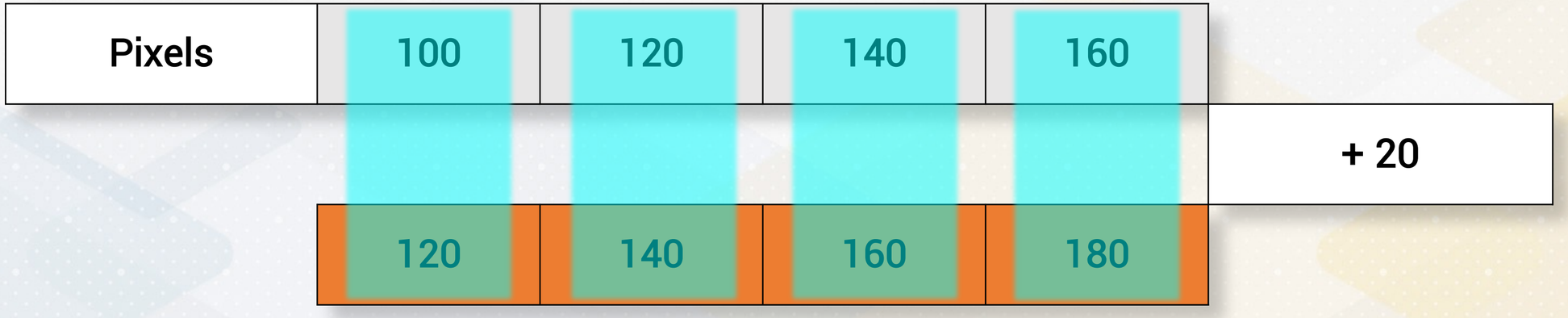
Cache Line Usage

Benefits of SIMD Execution #3



History of SIMD #1

1960s-1970s	**Supercomputers and Scientific Computing**	SIMD first used in systems like **ILLIAC IV** for scientific workloads	Permits simultaneous operations on multiple data points
1980s-1990s	**Multimedia Processing**	MMX (Intel, 1996): Integer operations for multimedia	Example: Increase the brightness of an image's pixels by a constant value



History of SIMD #2

2000s	**Integration in General-Purpose CPUs**	SIMD became a standard in consumer CPUs	SSE/AVX (Intel): Wide registers for floats and integers	NEON (ARM): Optimized for embedded and mobile devices
2010s-Present	**Acceleration for Analytical Workloads**	SIMD now powers modern databases and big data systems	Vectorized query execution (e.g., Apache Arrow)	

Sensor Data Analysis

```
struct SensorData {
  int *timestamps;     // Contiguous array of timestamps
  float *temperatures; // Contiguous array of temperatures


  SensorData(size_t count) {
    timestamps = static_cast<int *>(aligned_alloc(16, sizeof(int) * count));
    temperatures =
        static_cast<float *>(aligned_alloc(16, sizeof(float) * count));
  }
};
```

Sensor Data Generation

```
void generateData(SensorData &data, int count) {
  std::random_device rd;
  std::mt19937 gen(rd());
  std::normal_distribution<float> temp_dist(25.0, 5.0);
  std::uniform_int_distribution<int> time_dist(1, 5);


  int timestamp = START_TIMESTAMP;
  for (int i = 0; i < count; ++i) {
    data.timestamps[i] = timestamp;
    data.temperatures[i] = temp_dist(gen);
    timestamp += time_dist(gen); // Increment timestamp
  }
}
```

SIMD Query

SIMD Query

Task: Calculate the average temperature within a timestamp range

Timestamps	1	2	5	7
Temperature	25.5	26.0	27.2	28.3

> 2	0	1	1	0
< 6	1	1	1	0
> 2 AND < 6	0	1	1	0

Masked Temps	0	26.0	27.2	0
Sum	53.2			

SIMD Query: Loading Data

Timestamps	1	2	5	7
Temperature	25.5	26.0	27.2	28.3

```
int32x4_t ts_vec = vld1q_s32(&data.timestamps[i]);       // Load 4 timestamps

float32x4_t temp_vec = vld1q_f32(&data.temperatures[i]); // Load 4 temperatures
```

SIMD Query: Filtering Timestamps

> 2	0	1	1	0
< 6	1	1	1	0
> 2 AND < 6	0	1	1	0

```
uint32x4_t in_range_mask =
    vandq_u32(vcgeq_s32(ts_vec, vdupq_n_s32(startTimestamp)),
              vcleq_s32(ts_vec, vdupq_n_s32(endTimestamp)));
```

SIMD Query: Mask Application

Masked Temps	0	26.0	27.2	0

```
float32x4_t masked_temps = vmulq_f32(temp_vec, vcvtq_f32_u32(in_range_mask));

sum_vec = vaddq_f32(sum_vec, masked_temps);
```

SIMD Query: Final Steps

Sum	53.2			

```
float total_sum = vaddvq_f32(sum_vec);
for (int i = count - (count % 4); i < count; ++i) {
  if (data.timestamps[i] >= startTimestamp &&
      data.timestamps[i] <= endTimestamp) {
    total_sum += data.temperatures[i];
  }
}
```

Advanced SIMD: Hash Join Algorithm

```
// Load probe keys
int32x4_t probe_keys = vld1q_s32(&probe_table.keys[i]);
// Compute hash
int32x4_t hash_vec =
    vmodq_s32(vmulq_s32(probe_keys, hash_multiplier), hash_table_size);
// Gather hash table values
int32x4_t hash_table_vals = vld1q_s32(&hash_table[hash_vec]);
// Compare keys
uint32x4_t match_mask = vceqq_s32(probe_keys, hash_table_vals);
```

What's Next?

Looking Ahead: What's Next?

- CS 6423 (Advanced Database Implementation)
  - Logging and Recovery
  - Concurrency Control
  - Query Optimization
- Building on this course
  - Deeper insights into **how databases optimize for efficiency**.
  - Expanding your knowledge of **system-level guarantees**.

Logging and Recovery

- **Example:** A system crash during a transfer:

```
--- Money transfer from Account 1 to Account 2
BEGIN TRANSACTION
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
--- System crash
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
END TRANSACTION
```

Logging and Recovery

- Mechanisms to restore the database to a consistent state after crashes.
- **Write-Ahead Logging (WAL)**
  - Log changes before applying them.
- **Checkpointing and Crash Recovery**
  - Periodically save the database state to reduce recovery time.
  - Redo/Undo logs to reconstruct committed transactions and roll back uncommitted ones.

Concurrency Control

- **Example:** Two users simultaneously trying to update the same row

```
--- Deposit from User 1 into Account 1
BEGIN TRANSACTION
UPDATE accounts SET balance = balance + 100 WHERE id = 1;
END TRANSACTION

--- Deposit from User 2 into Account 1
BEGIN TRANSACTION
UPDATE accounts SET balance = balance + 100 WHERE id = 1;
END TRANSACTION
```

Concurrency Control

- Ensuring **correctness** and **consistency** when multiple users access the database simultaneously.
- **Locks and Latches**
  - Types of locks (shared, exclusive); Deadlock detection and prevention.
- **Multi-Version Concurrency Control (MVCC)**
  - Readers don't block writers; writers don't block readers.
- **Isolation Levels**
  - Read Committed, Repeatable Read, Serializable.

Query Optimization

- **Example:** Push filters before the join. Use an indexed join when available.

```sql
--- Orders from customers based in Atlanta
SELECT *
FROM orders
JOIN customers ON orders.customer_id = customers.id
WHERE customers.city = 'Atlanta';
```

Query Optimization

- Selecting the **best execution plan** for a query.
- Goal: Minimize **execution cost** (e.g., time, memory, I/O).
- **Execution Plans:** Logical vs. physical plans.
- **Cost Models:** Estimating costs for different plans.
- **Heuristics and Rules:** Simplifying query plan tree
- Advanced Techniques
  - Dynamic Programming (e.g., System R algorithm).
  - Cardinality Estimation

Course Feedback & Project Presentations

- Please share your feedback via CIOS
- +1% extra credit for entire class if we get 80%+ participation
- No class on **Nov 18**
- In-class presentations on **Nov 21**
- Tentatively prepare a 5-minute presentation!

Course Retrospective

Takeaways from the Course

- Let's take a step back and reflect on what you've accomplished.
- Systems programming is challenging.
  - Delving into internals teaches attention to detail.
  - It forces understanding of how things work under the hood.
- Foundational systems knowledge beyond databases.
  - Threading, memory management, and I/O.
  - You now have tools to approach any system-level problem.
  - Do reflect on how much you've learned and grown as a programmer.

Big Ideas from the Course

- Database Systems Are Awesome
  - They solve real-world problems elegantly.
  - But they are **not** magic!
- Abstractions Are “Magic”
  - Elegant abstractions are the "magic" enabling usability and performance
- Declarativity Rules
  - Declarative query models make complex systems usable.
  - Taken to the extreme -- Google search or ChatGPT query!

Big Ideas from the Course

- Building Systems Is More Than Hacking
  - It's about design, principles, and reusability.
- Recurring System Design Principles
  - Recognizing motifs like optimizing for the common case (caching), work avoidance (indexing), composability and modularity (operator framework).

Periodic Table of System Design Principles

Towards a Periodic Table of Computer System Design Principles

JOY ARULRAJ, Georgia Institute of Technology

System design is often taught through domain-specific solutions specific to particular domains, such as databases, operating systems, or computer architecture, each with its own methods and vocabulary. While this diversity is a strength, it can obscure cross-cutting principles that recur across domains. This paper proposes a preliminary "periodic table" of system design principles distilled from several domains in computer systems. The goal is a shared, concise vocabulary that helps students, researchers, and practitioners reason about structure and trade-offs, compare designs across domains, and communicate choices more clearly. For supporting materials and updates, please refer to the repository at: https://github.com/jarulraj/periodic-table.

https://github.com/jarulraj/periodic-table

You can contribute to Database Systems!

- Database Systems is evolving; you can contribute to its history!