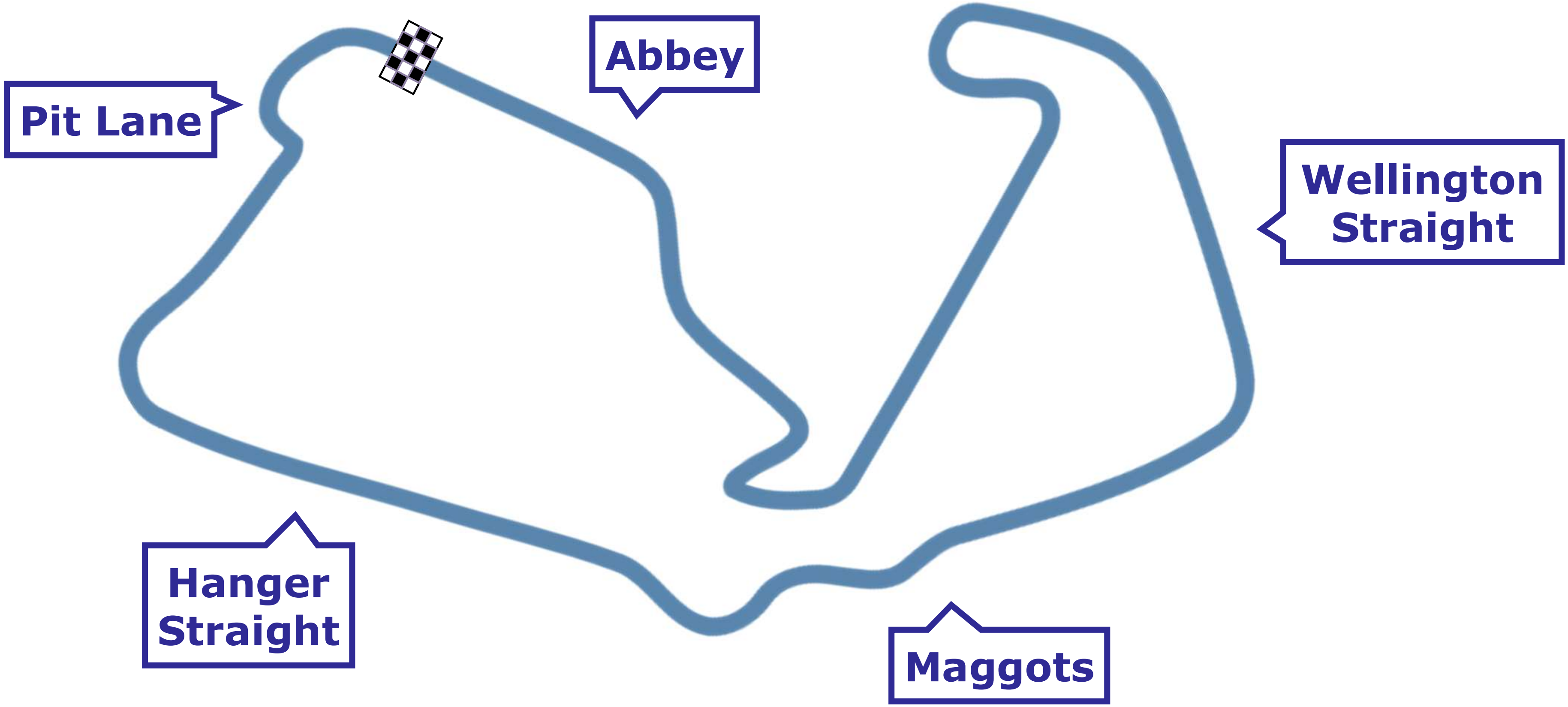
High-Speed Processing & SQL

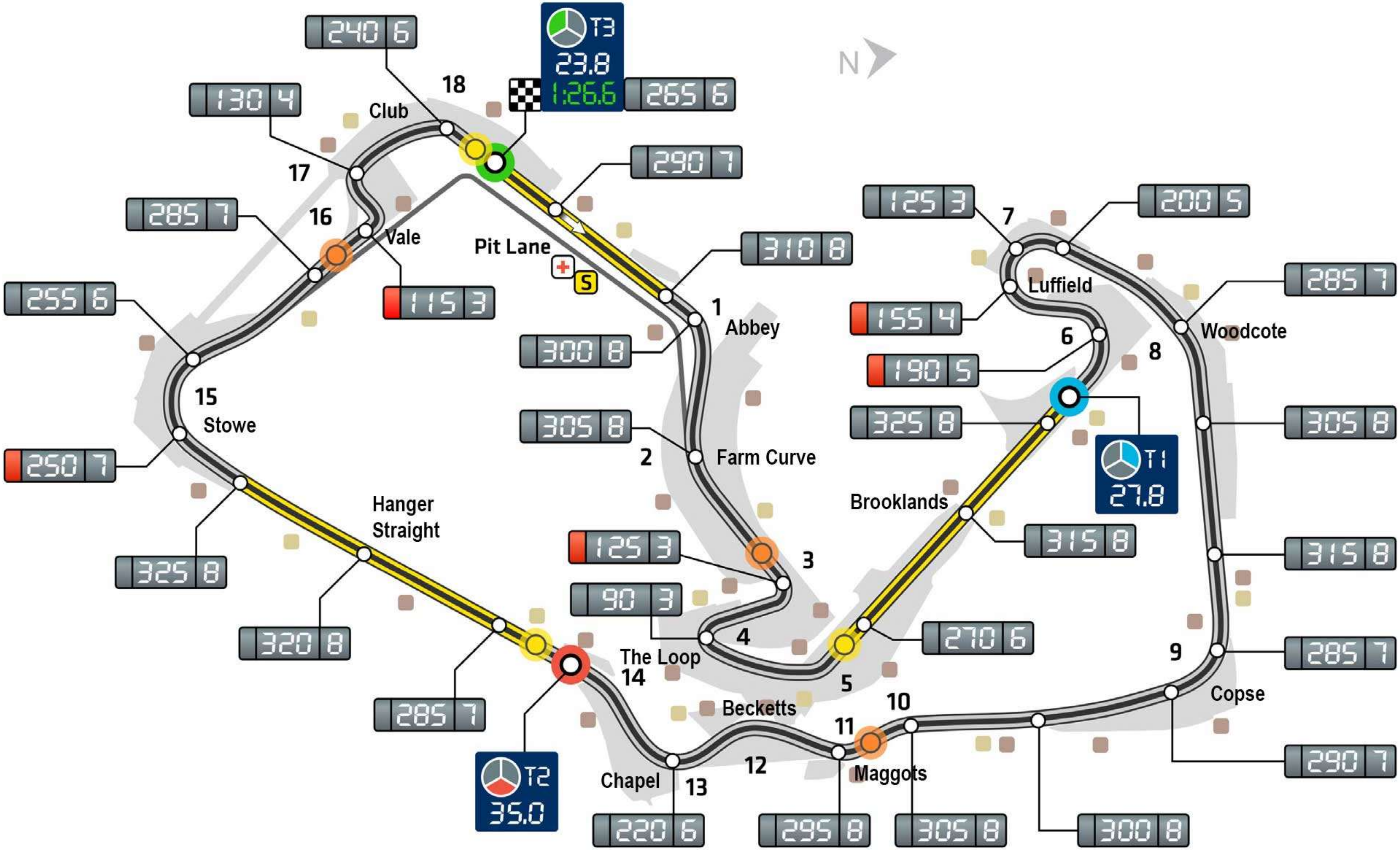
InterSystems UKI Summit

October 19, 2022

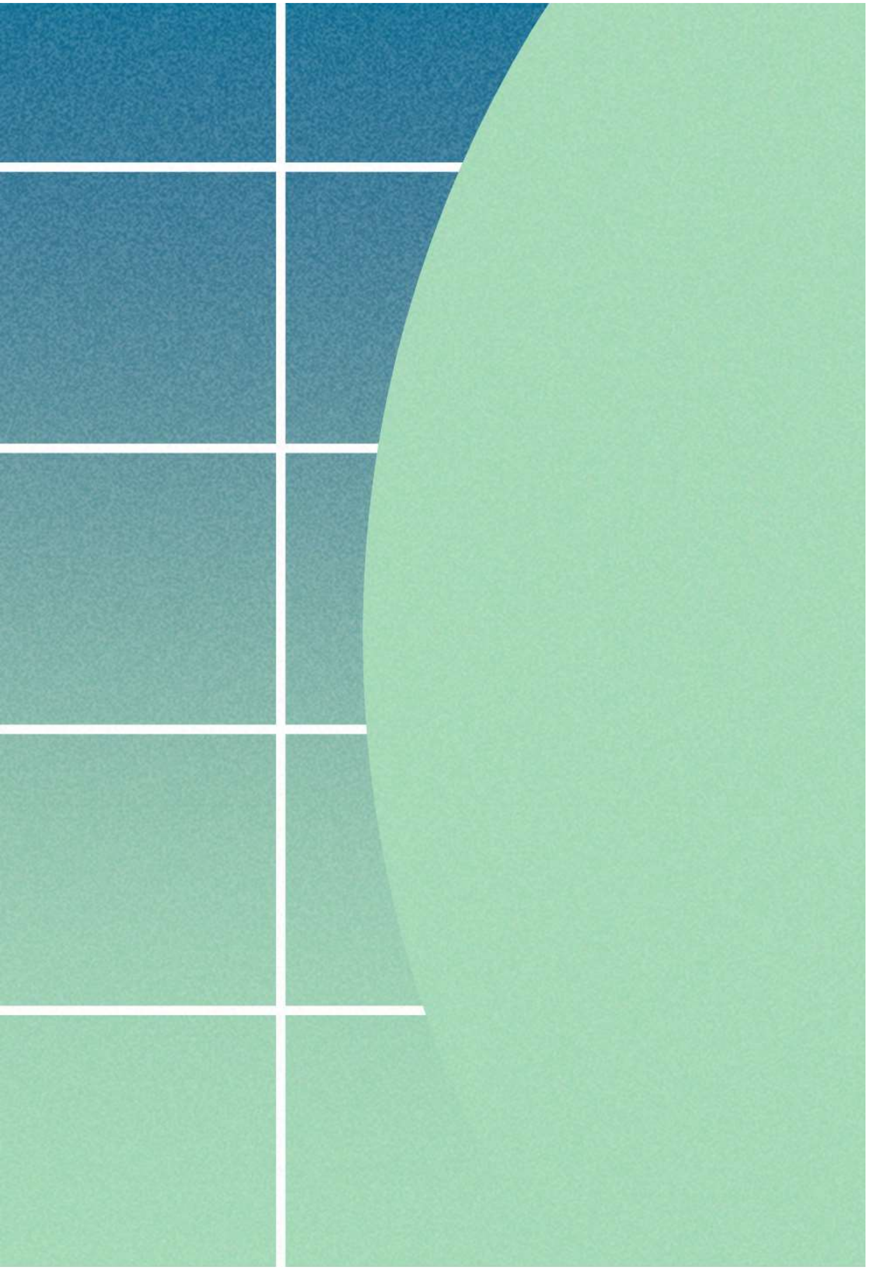
InterSystems®
Creative data technology

High Speed – UKI Edition





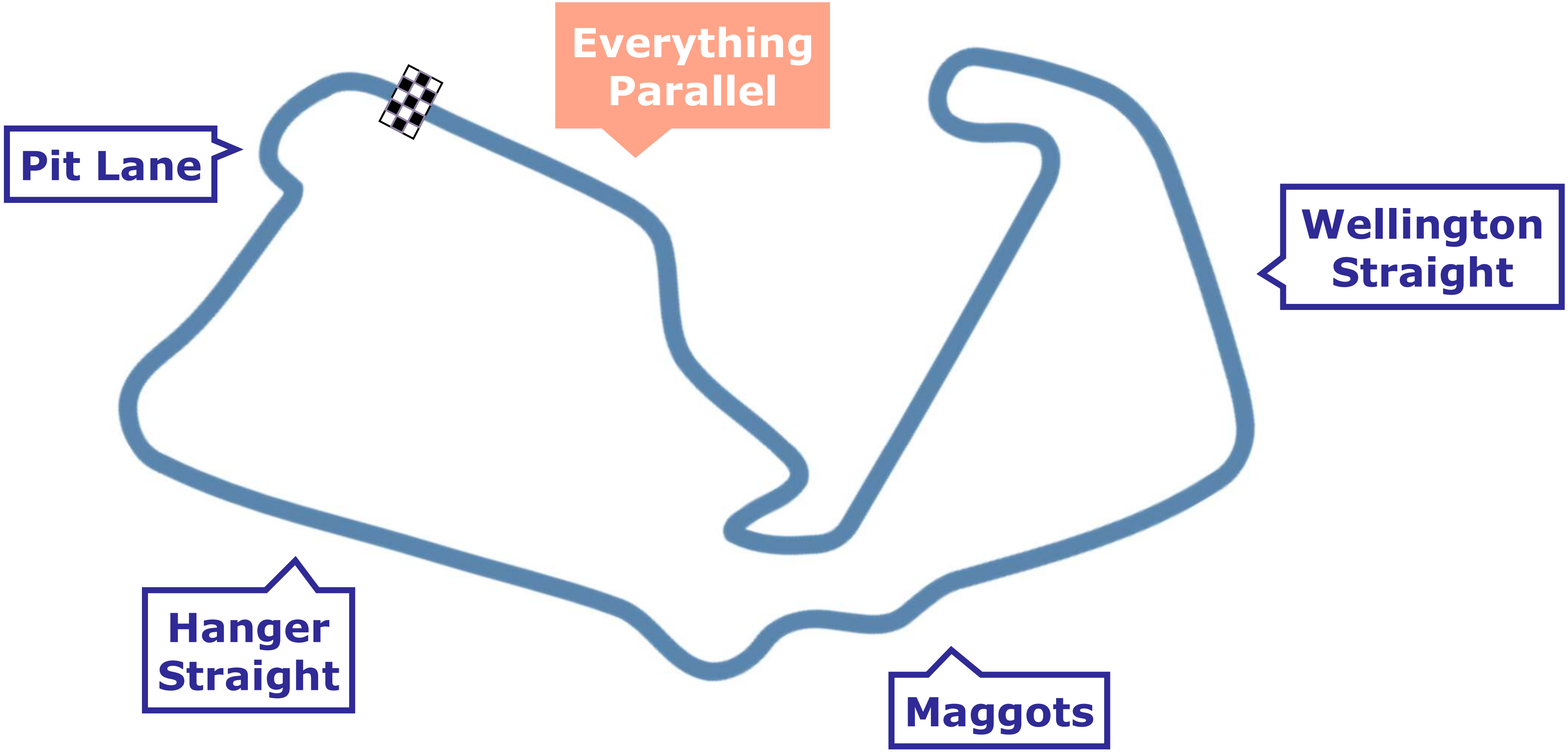
240 6
130 4
18
Club
T3
23.8
1:26.6
265 6
N
290 7
17
285 7
16
Vale
Pit Lane
S
310 8
1
Abbey
255 6
115 3
300 8
125 3
7
200 5
Luffield
285 7
Woodcote
155 4
6
8
190 5
325 8
T1
27.8
305 8
15
Stowe
305 8
2
Farm Curve
250 7
Brooklands
Hanger
Straight
315 8
315 8
325 8
125 3
3
90 3
320 8
4
270 6
285 7
The Loop
14
5
9
285 7
Copse
Beckets
10
285 7
11
T2
35.0
Chapel
13
12
Maggots
290 7
220 6
295 8
305 8
300 8



Abbey: Everything Parallel

High-Speed Processing & SQL

High Speed – UKI Edition



Everything Parallel

Single Car

Multi Car

Everything Parallel – Application

Available Today

- Auto-parallelized SQL query execution
- Always-parallel MDX query execution
- Work Queue Manager API for custom code
- Workload distribution across shards

Coming Soon

- Tuned heuristics for auto-parallelized SQL
- Auto-parallelized ingestion with `LOAD DATA` command

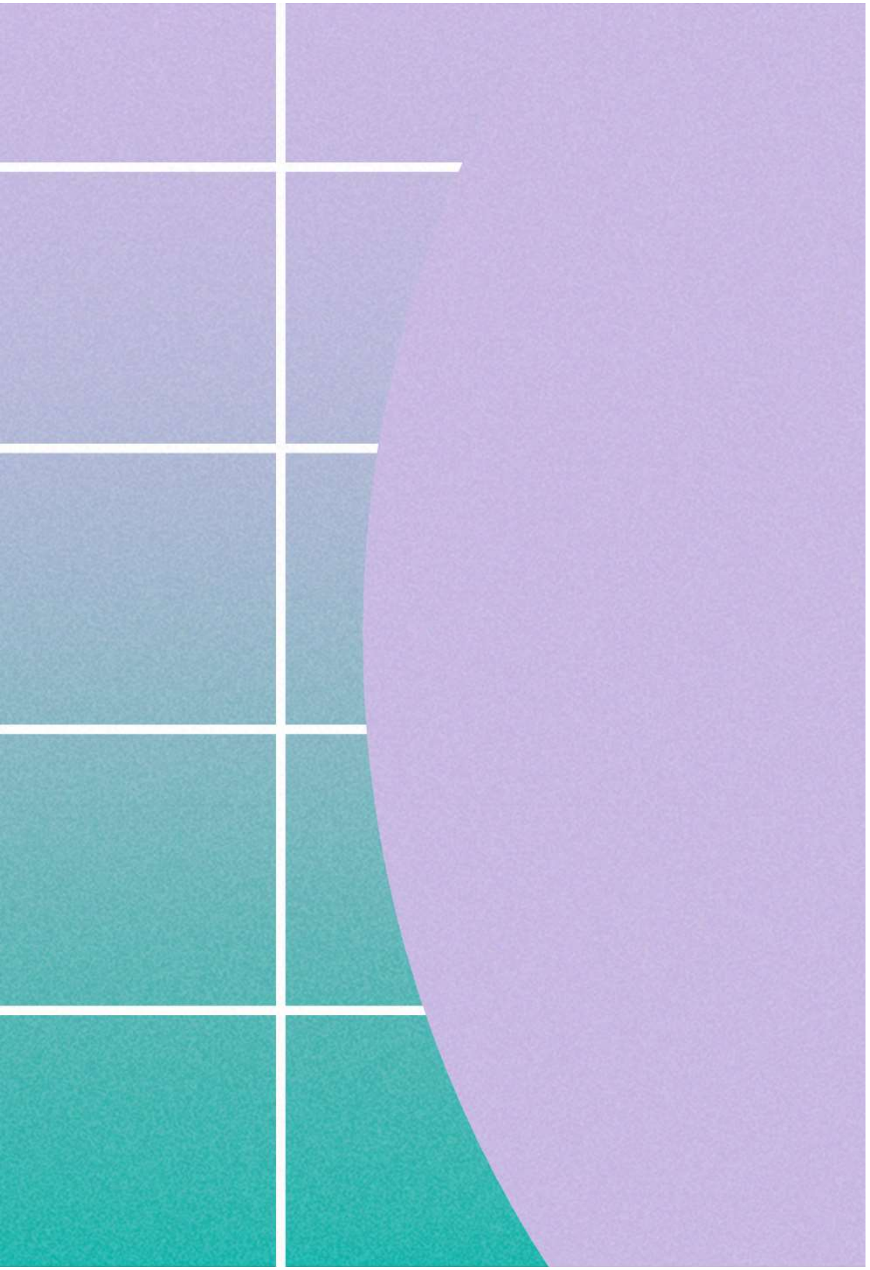
Everything Parallel – Kernel

Available Today

- Auxiliary Write Daemons participate in Write Image Journaling, complementing async IO – up to 4x throughput
- Multi-process dejournaling – easily doubles mirroring throughput

Coming Soon

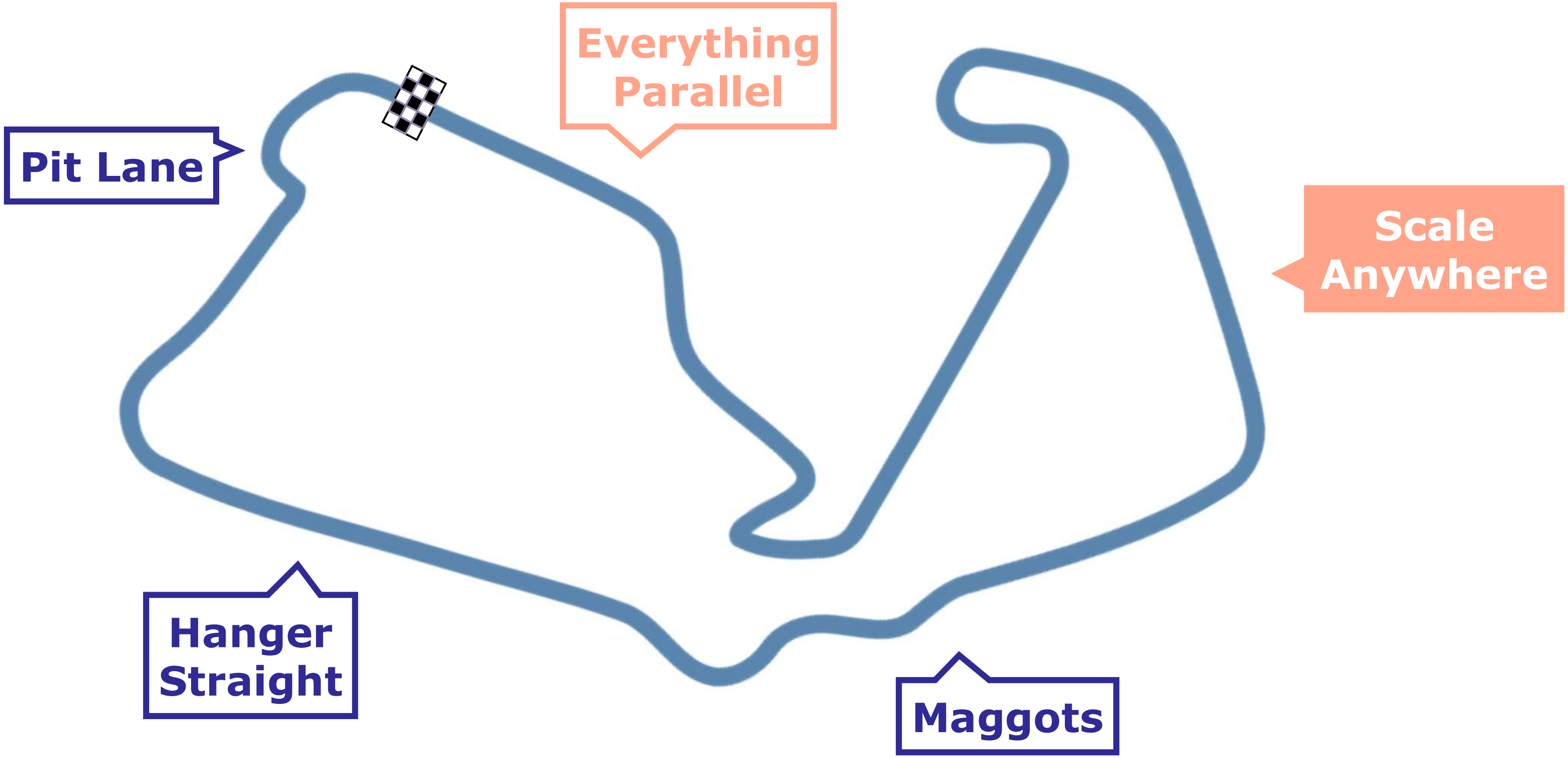
- Multi-process Online Backup



Wellington Straight: Scale Anywhere

High-Speed Processing & SQL

High Speed – UKI Edition



Everything Parallel

Pit Lane

Scale Anywhere

Hanger Straight

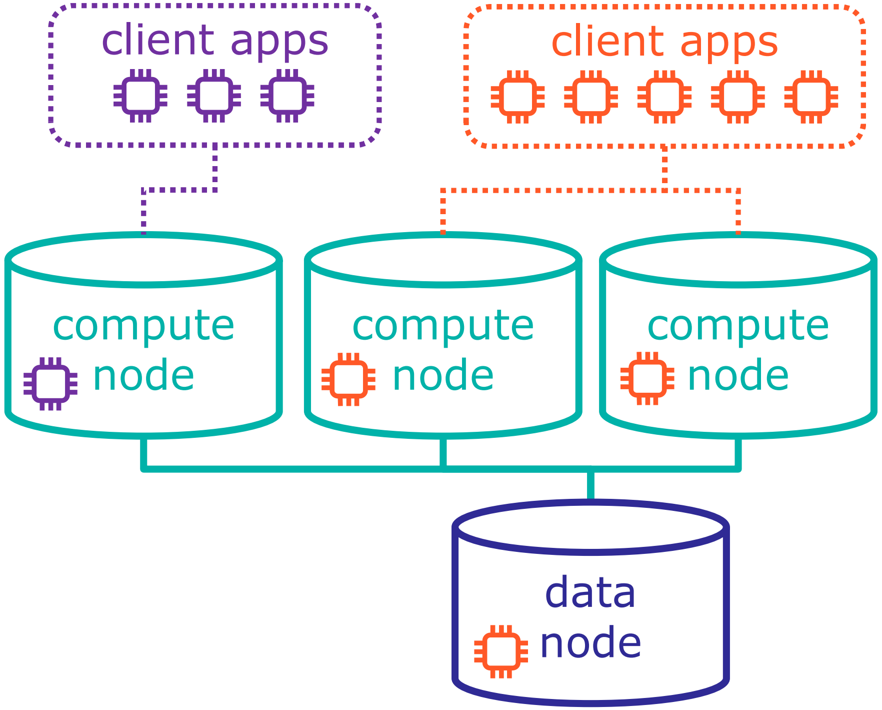
Maggots

Scaling Compute

Enterprise Cache Protocol

- Fully Transparent
- Fully Elastic
- Easy to Organize

Recent work: increased efficiency at ultra-high scale



Scaling Compute – Recent Lab Testing

Goal: identify application and system-level bottlenecks beyond the current horizon

- 100 compute nodes
- over 4000 CPU cores

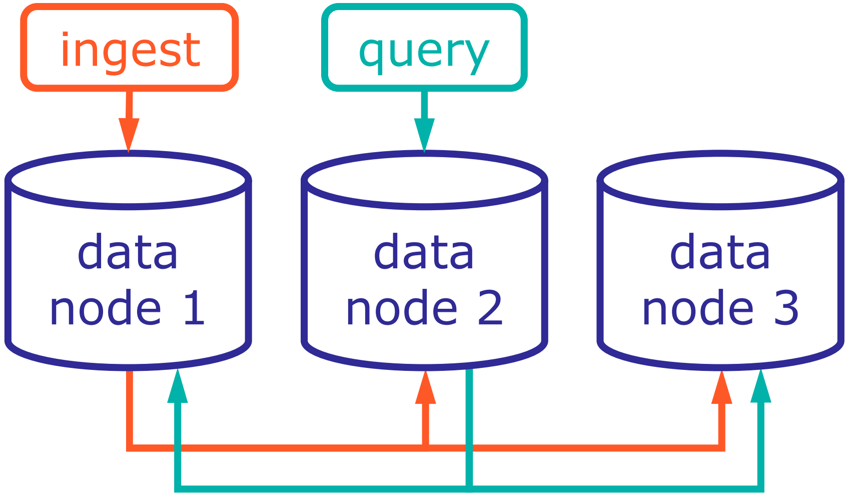
Outcome: Achieved 600M grefs/s of sustained load.

- IRIS keeps pushing the limits to get the most out of your infrastructure investment.

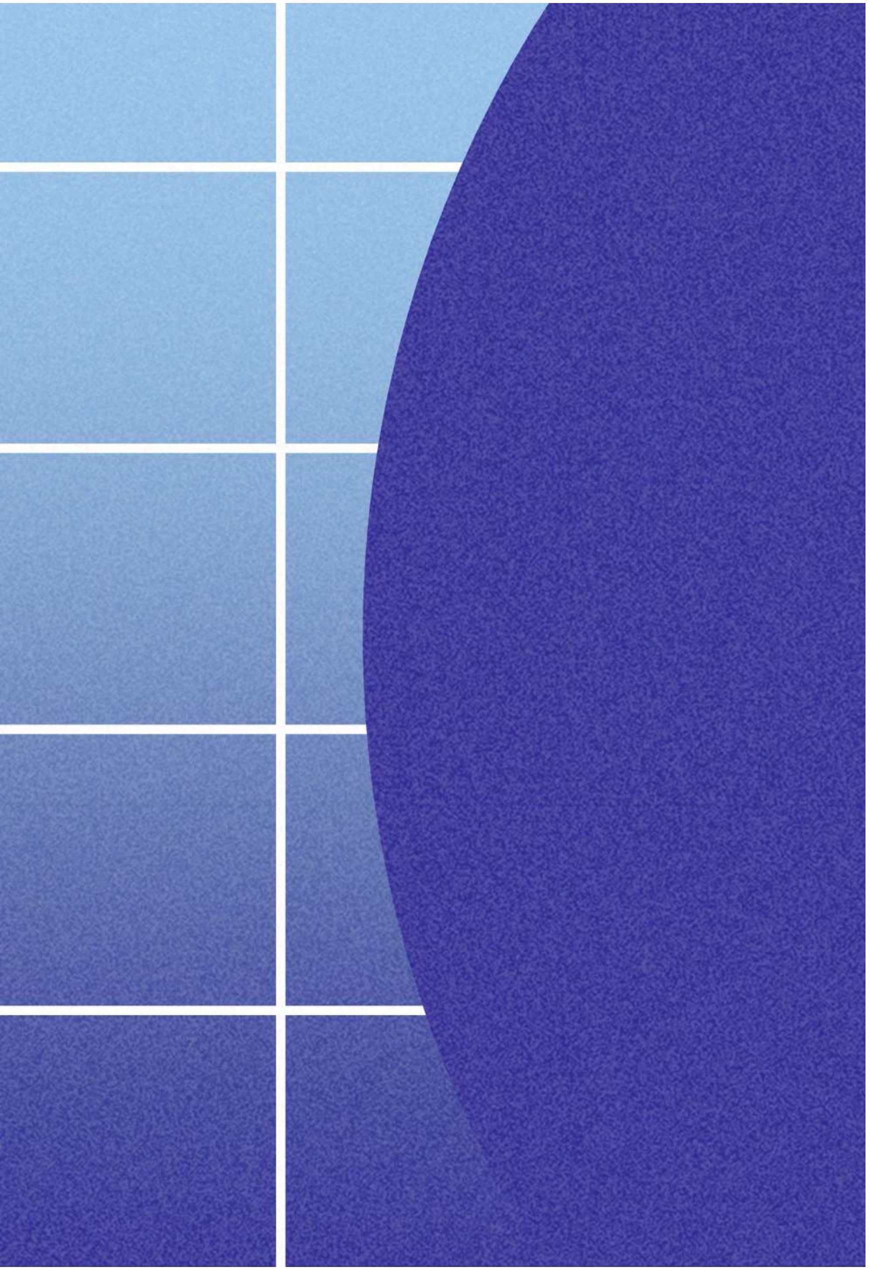
Scaling Data

Sharding

- Transparent Data Management
- Transparent Query Management



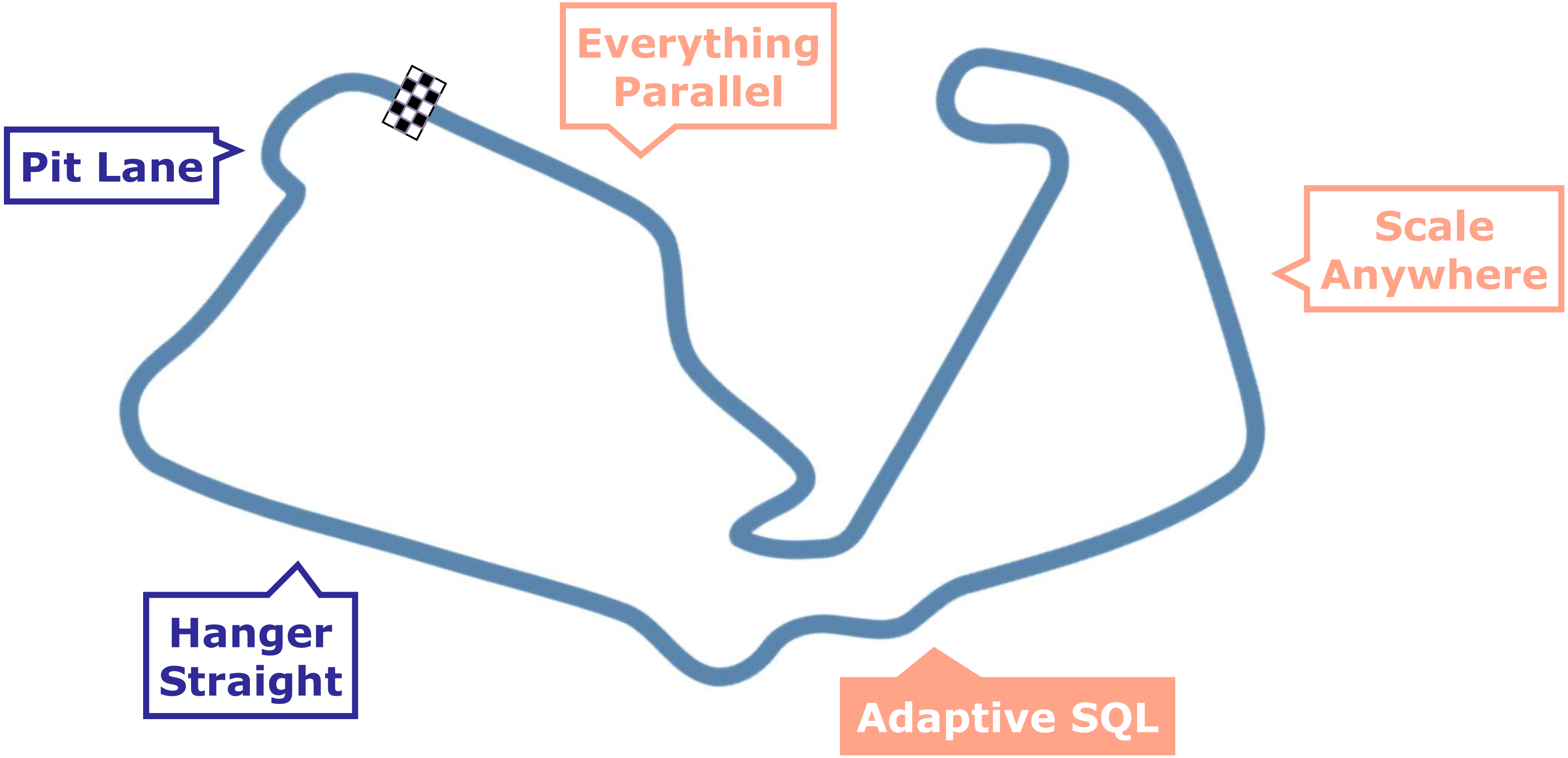
Recent work: elasticity in the data tier and improved schema design flexibility



Maggots: Adaptive SQL

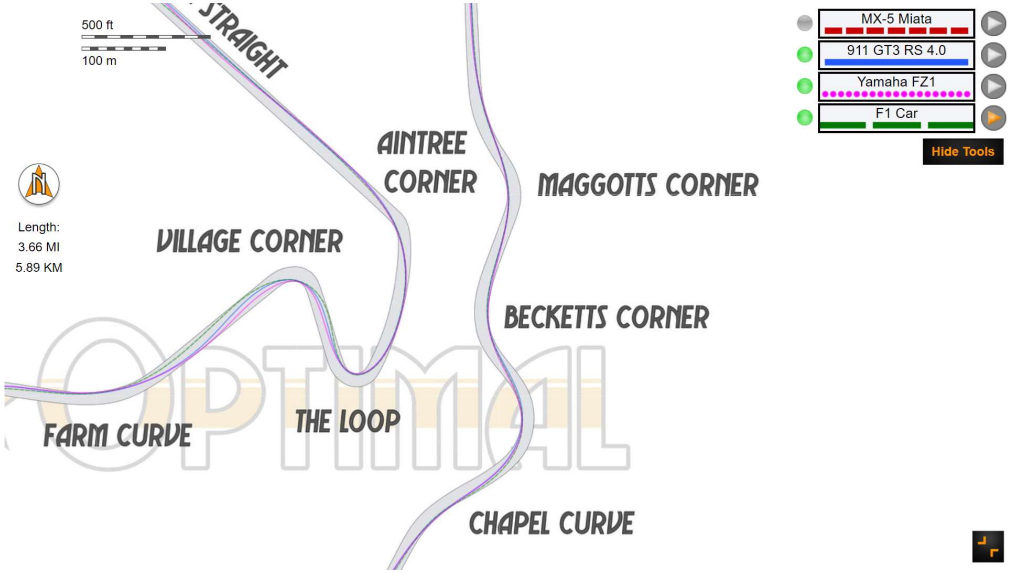
High-Speed Processing & SQL

High Speed – UKI Edition



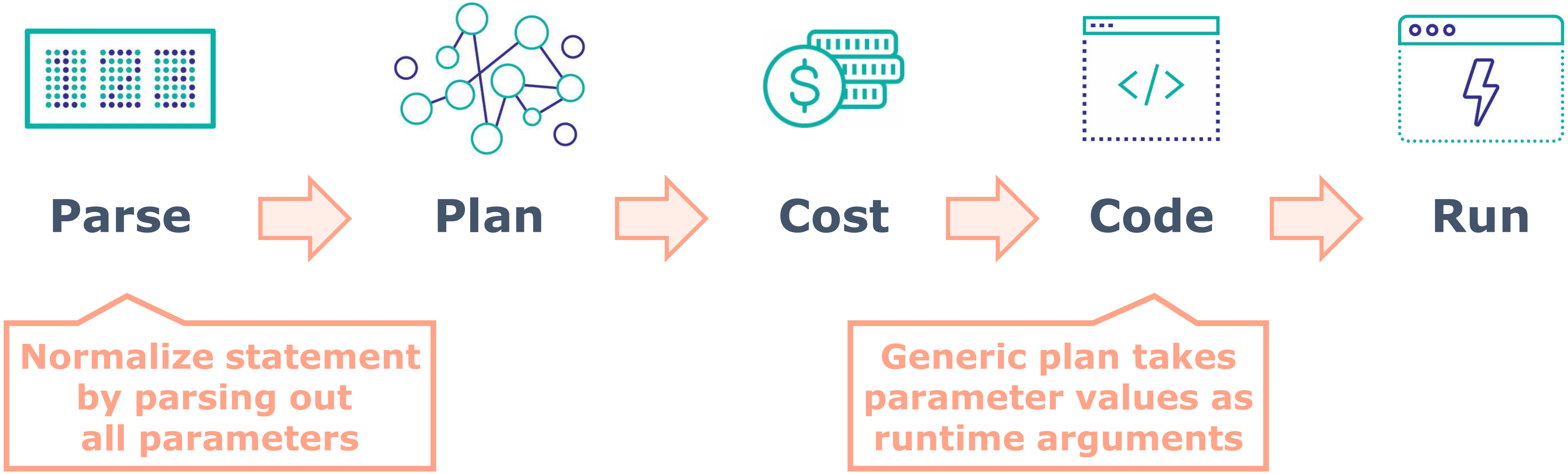
Race Planning

The Plan



The Race

SQL Processing

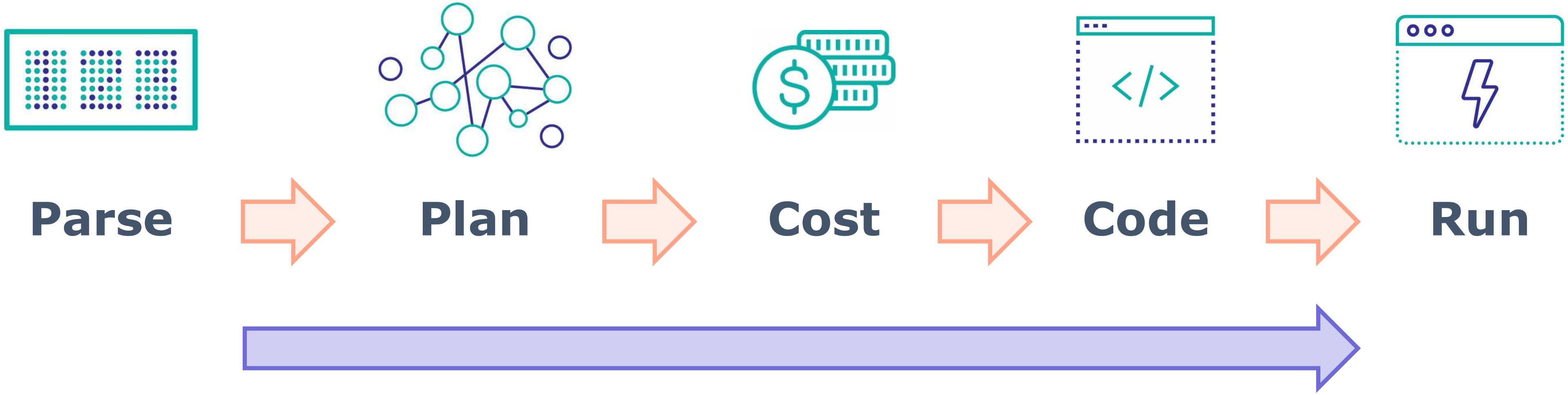


Parse → Plan → Cost → Code → Run

Normalize statement by parsing out all parameters

Generic plan takes parameter values as runtime arguments

SQL Processing



Parse

Plan

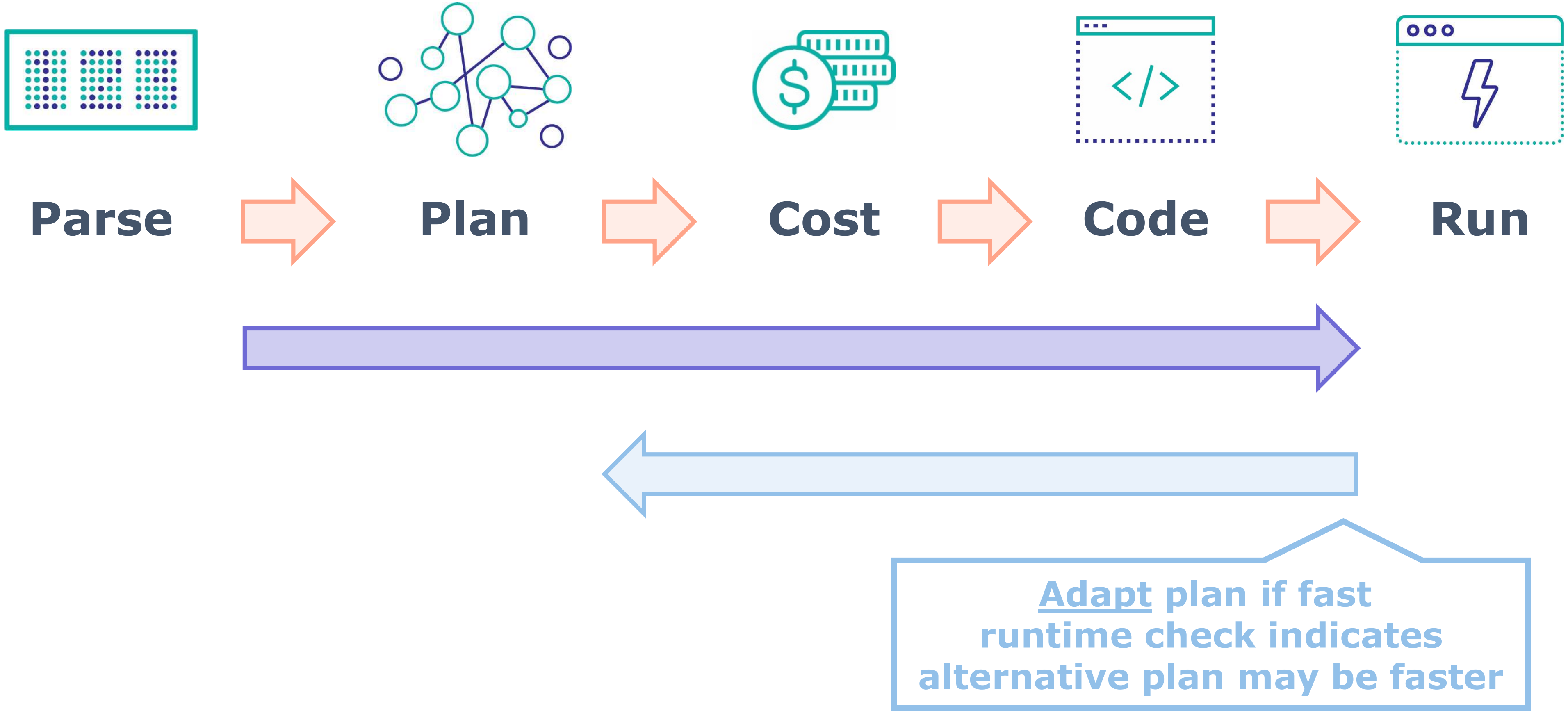
Cost

Code

Run

Reuse generic plan and code at next invocation

SQL Processing



Parse

Plan

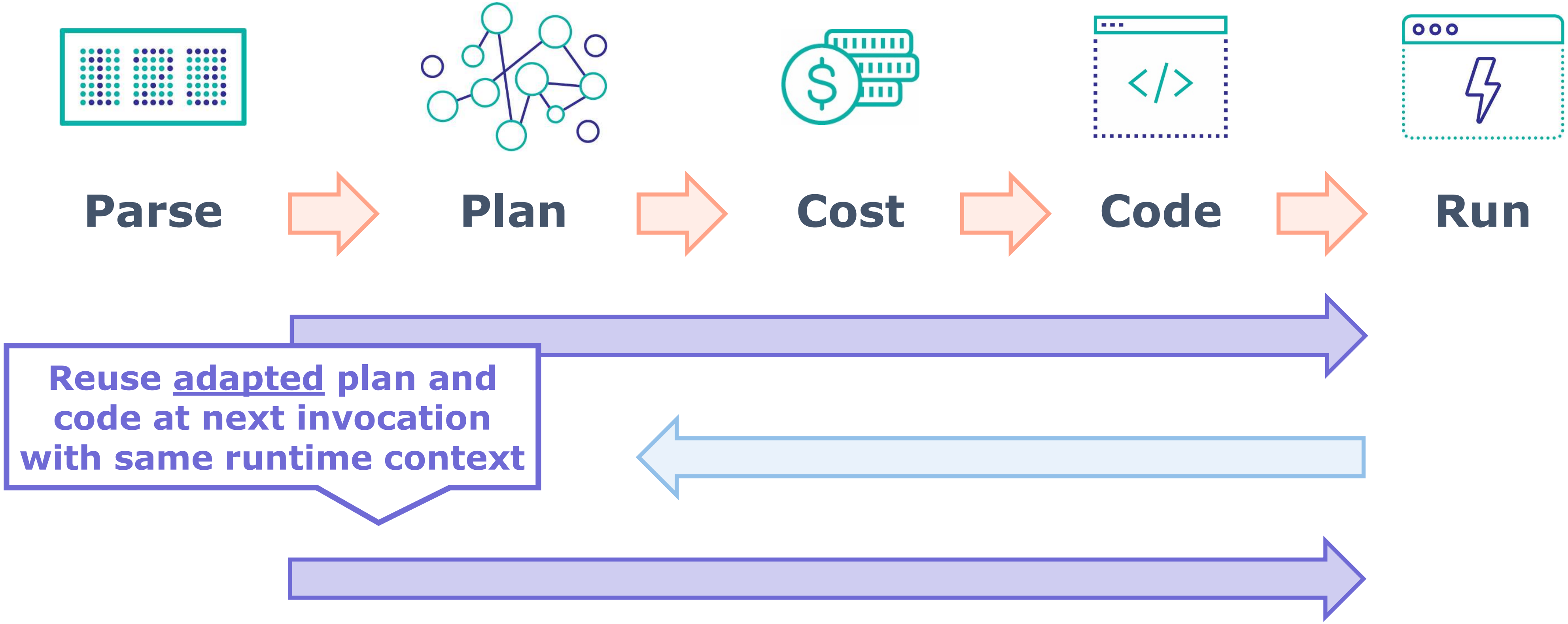
Cost

Code

Run

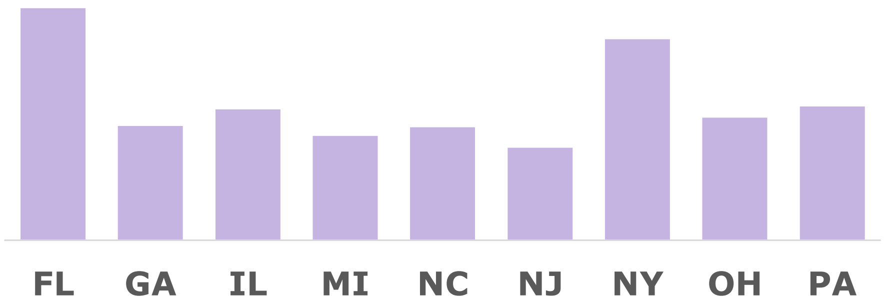
Adapt plan if fast
runtime check indicates
alternative plan may be faster

SQL Processing

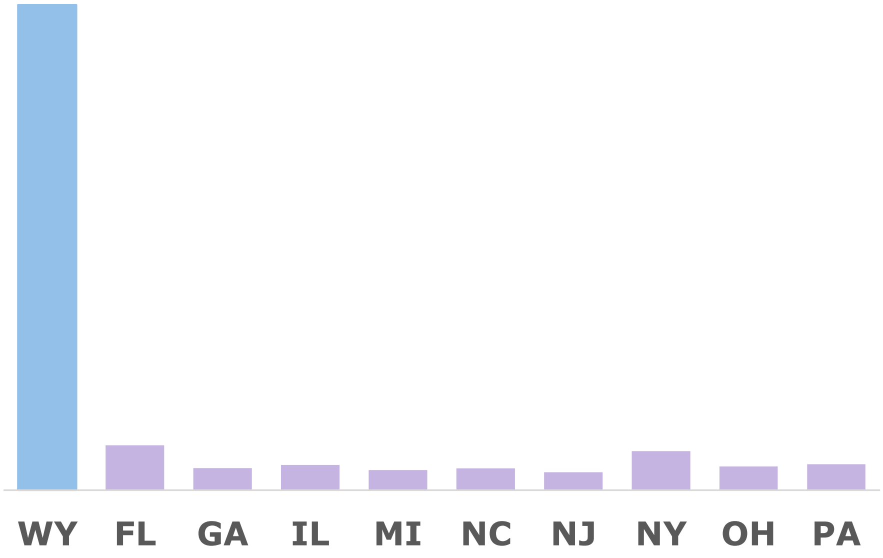


Reuse adapted plan and code at next invocation with same runtime context

Outlandish Parties



National retailer



Wyoming retailer

Outlandish Parties

Outliers are field values with a disproportionately high frequency

- Outliers are very common in real-world data and (used to be) a common source of unlucky query plans.
- IRIS registers outliers and their selectivity separately in the table stats

Adaptive Planning

InterSystems IRIS SQL’s **RunTime Plan Choice** checks parameter values for re-planning opportunities before running the default plan:

- Outlier values:

  ```
  … FROM log WHERE level = 'INFO'
  ```

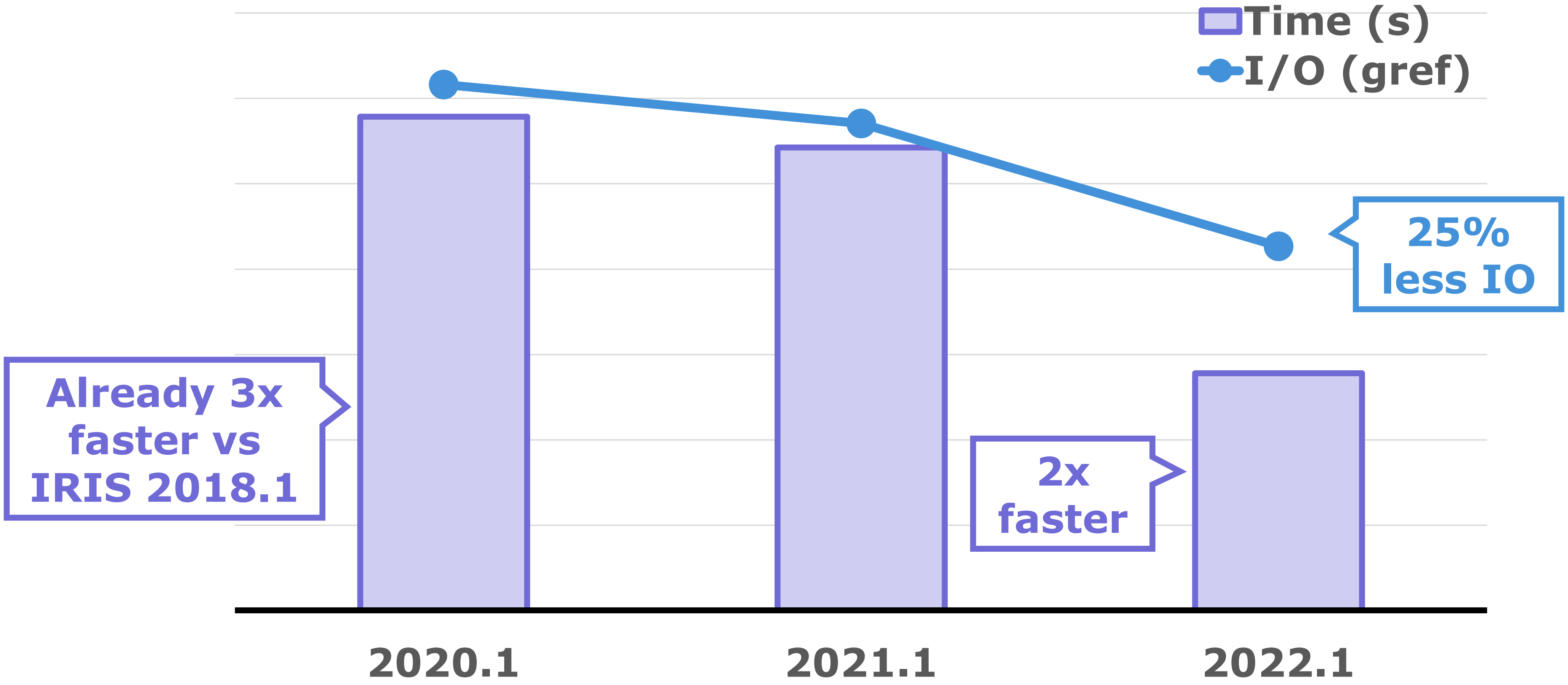
- Range selectivity:

  ```
  … FROM log WHERE dt > '5/5/22'
  ```

- Truth conditions:

  ```
  … FROM log WHERE (1 = 0 AND …)
  ```

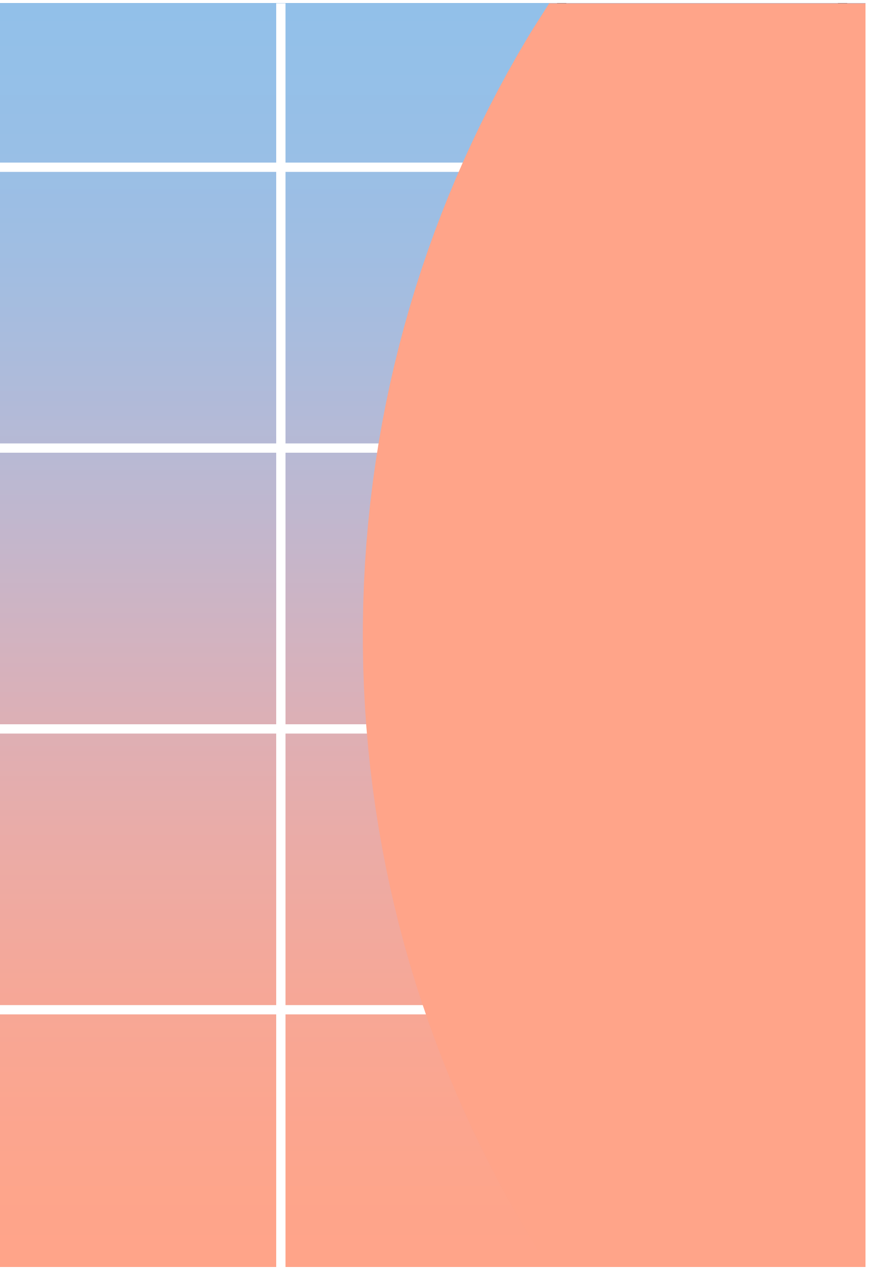
Adaptive Planning – Customer Benchmark



Already 3x faster vs IRIS 2018.1

2x faster

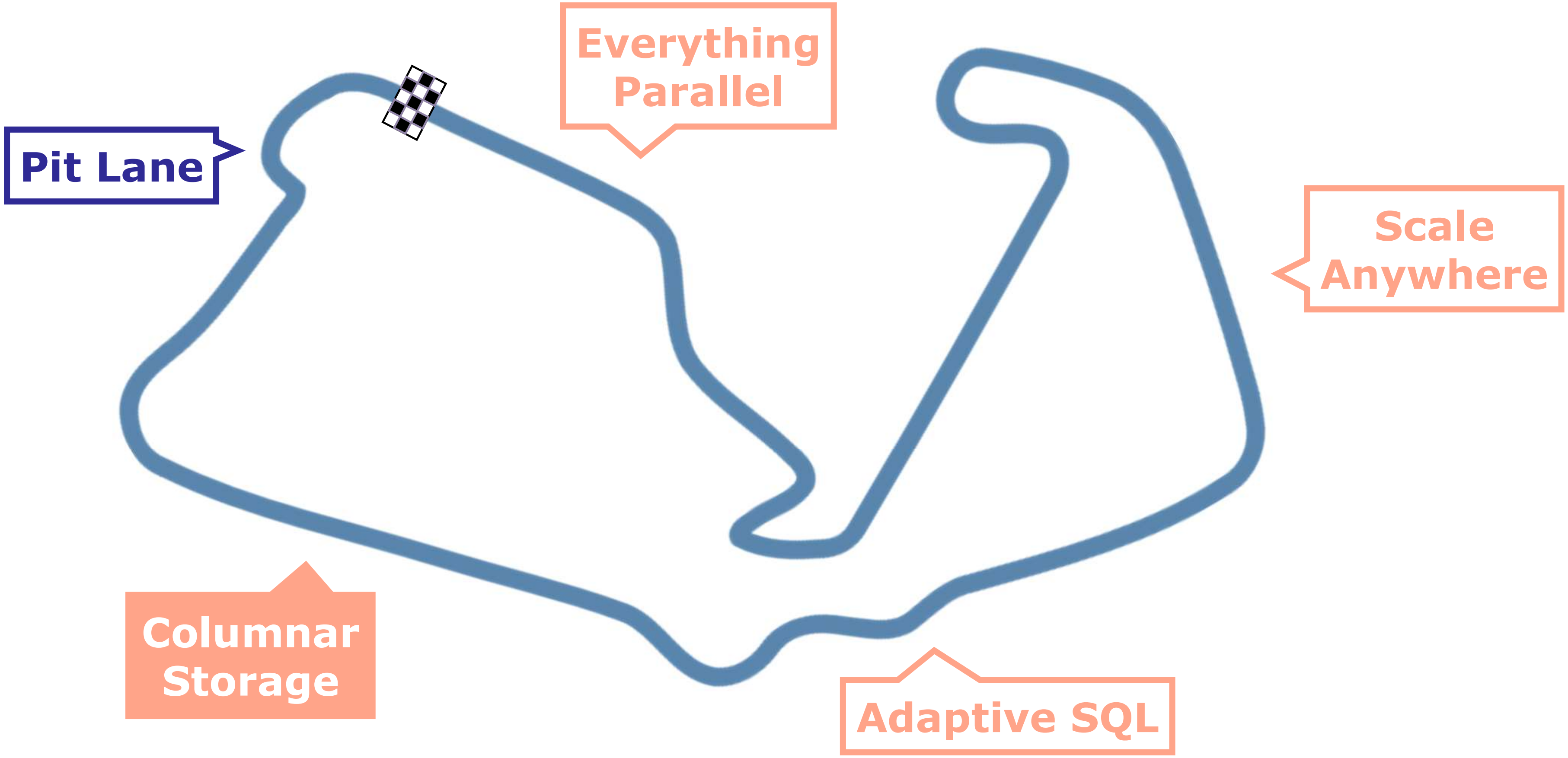
25% less IO



Hanger Straight: Columnar Storage

High-Speed Processing & SQL

High Speed – UKI Edition



Applications

```
SELECT TOP 10 * FROM tx WHERE acct = 123
   ORDER BY txTime DESC

START TRANSACTION

INSERT INTO tx (txTime, acct, type, amount)
   VALUES ( NOW(), 123, 'DEBIT', -1000 );
INSERT INTO tx (txTime, acct, type, amount)
   VALUES ( NOW(), 456, 'CREDIT', 1000 );

UPDATE acct SET balance = balance – 1000
   WHERE ID = 123;
UPDATE acct SET balance = balance + 1000
   WHERE ID = 456;

COMMIT
```

Applications

Fast row inserts & updates

Full row retrieval

Focus on **latency**

Store data how it’s used: **row by row**

Analytics

```
  SELECT MONTH(txTime) AS TxMonth,
         type          AS TxType,
         AVG(amount)   AS AverageAmount,
         MAX(amount)   AS MaxAmount

    FROM tx

   WHERE acct = 123
         AND txTime > DATEADD('YY', -1, NOW())

GROUP BY MONTH(txTime),
         type;


LOAD DATA FROM FILE '/tmp/20221018-tx.csv' INTO tx;
```

Analytics

Complex queries on large tables

Returning aggregates, not rows

Focus on **throughput**

Analytics

Complex queries on large tables

Returning aggregates, not rows

Focus on **throughput**

Store data how it’s used: **column by column**

A Bitmap of Pioneering

Bitmap Indices pioneered the key concepts needed for efficient analytical query processing

- Pack info for many rows in one IO
- Operate on many rows in one function call

Regular Index	Bitmap Index
^idx(“ABC”, 1) = “” ^idx(“ABC”, 3) = “” ^idx(“ABC”, 4) = “” ^idx(“DEF”, 2) = “” ^idx(“DEF”, 5) = “” ^idx(“DEF”, 64001) = “” ^idx(“DEF”, 64002) = “”	^idx(“ABC”, 1) = $bit(1, 3, 4) ^idx(“DEF”, 1) = $bit(2, 5) ^idx(“DEF”, 2) = $bit(1, 2)

A Bitmap of Pioneering

`$bit` is a dedicated string-based datatype for bit sequences, used in bitmap indices

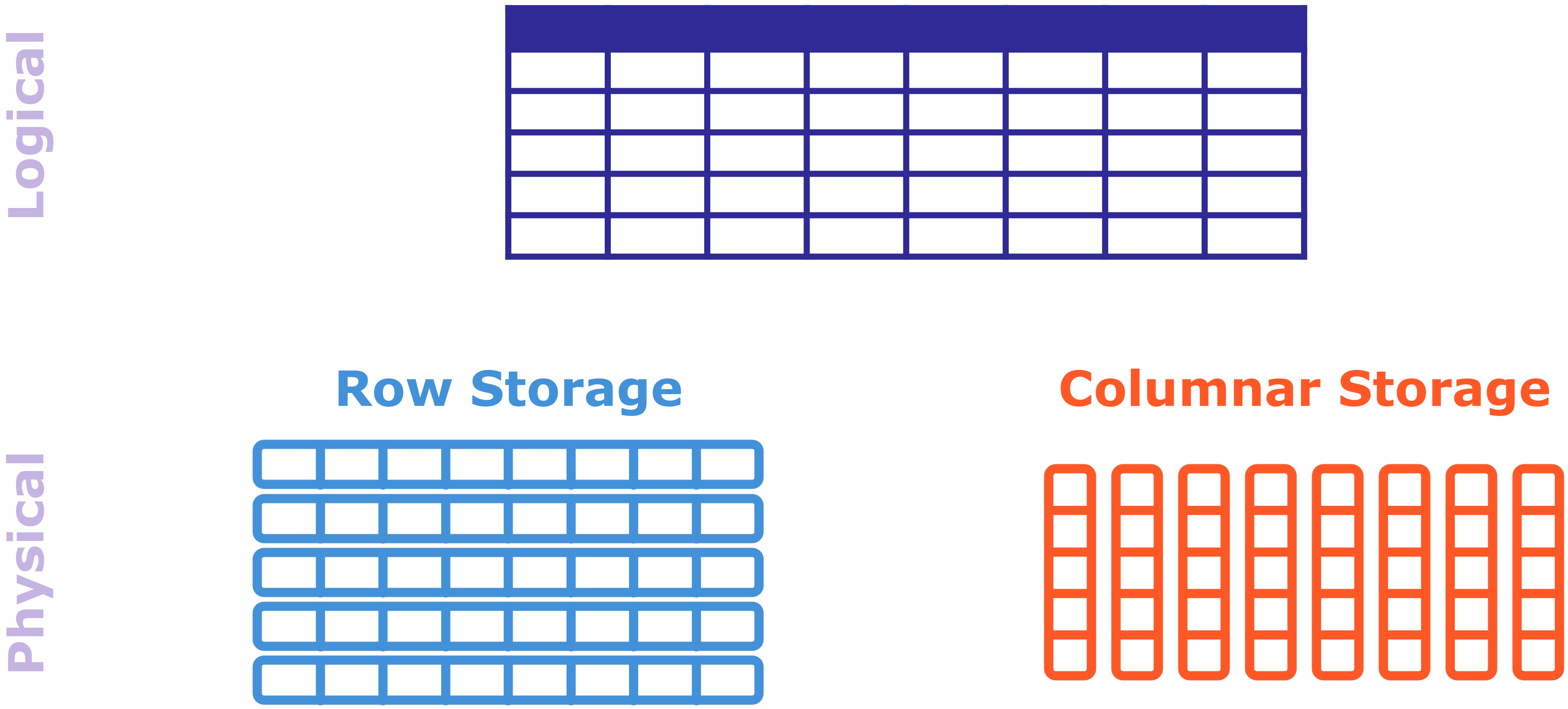
Optimized Storage

- Flexible internal structure for `$bit` enables compression
- Optimal 64k chunk size empirically shown to work well

Optimized Compute

- Dedicated operations for Boolean logic & traversal
- Support for atomic updates, ECP and journaling

Optimized Storage



Logical

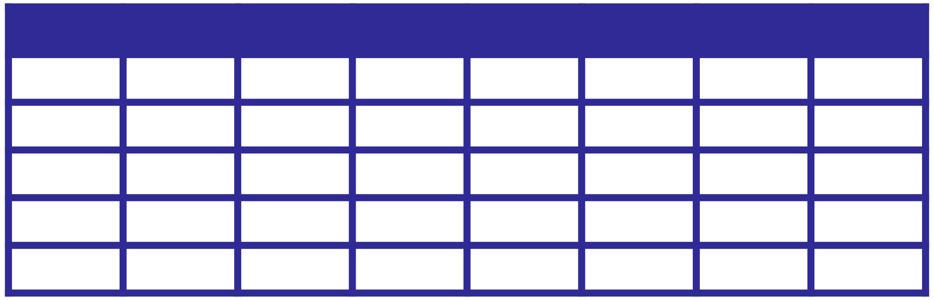
Physical

Row Storage

Columnar Storage

Optimized Storage

Logical



Physical

Row Storage

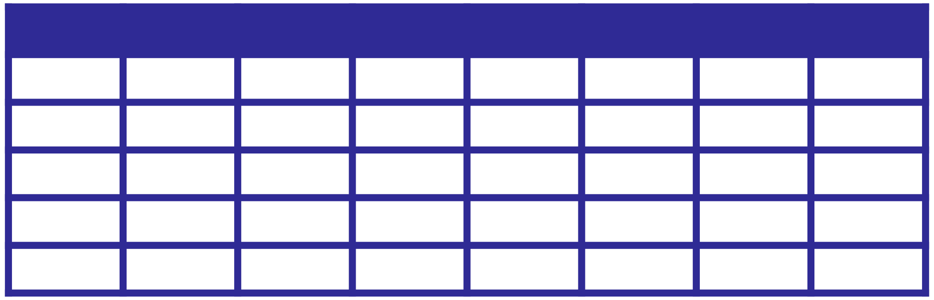
```
^d(1) = $list(“abc”, 9, 1.23, …)
^d(2) = $list(“abc”, 8, 2.1, …)
^d(3) = $list(“def”, 7, 3.45, …)
^d(4) = $list(“ghi”, 6, <null>, …)
^d(5) = $list(“xyz”, 5, 9.99, …)
```

Columnar Storage

```
^d.V1(1) = $vector(<string>: “abc”,
  “abc”, “def”, “ghi”, “xyz”, …)
^d.V2(1) = $vector(<integer>: 9, 8, 7,
  6, 5, …)
^d.V3(1) = $vector(<decimal>: 1.23, 2.1,
  3.45, <null>, 9.99, …)
```

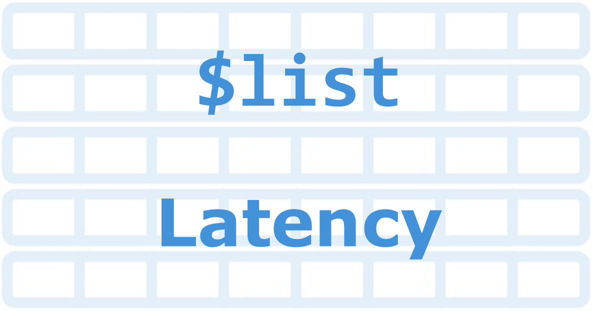
Optimized Storage

Logical



Physical

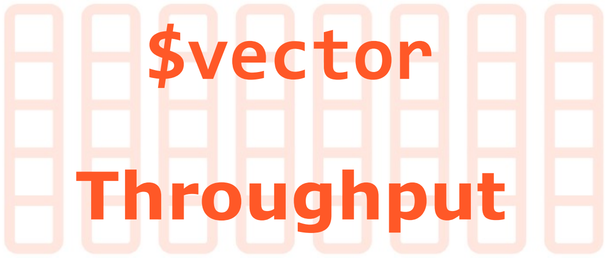
Row Storage



`$list`

Latency

Columnar Storage

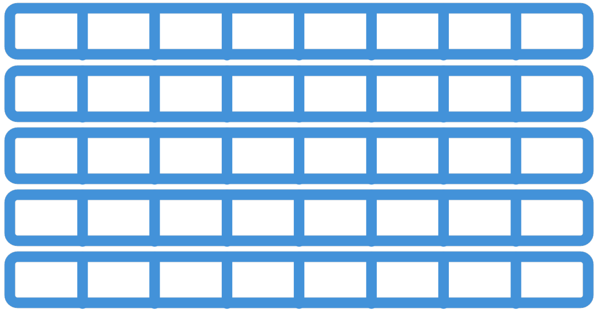


`$vector`

Throughput

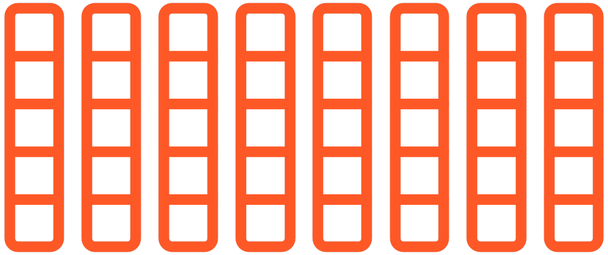
Optimized Storage

Row Storage



- Clustered for point IO: **Latency**
- Packed in **$list** format for dynamic access: **Flexibility**
- Cache all fields for few rows: **Transactions**

Columnar Storage



- Clustered for bulk IO: **Throughput**
- Packed in **$vector** format for predictable access: **Throughput**
- Cache selected fields for many rows: **Throughput**

$vector

New internal data type for storing large arrays of same-datatype values

Efficient handling of **sparse data**

- Internal distinction between dense and sparse regions using run-length encoding

Efficient **datatype-specific encodings**

- Dictionary encoding for strings
- Adaptive scale for integers

Support for **atomic and bulk updates**

- Including ECP and journaling

Optimized Compute

Modern CPUs love **tight loops**:

```
for (i = 0; i < BUF_SIZE; i++) { c[i] = a[i] + b[i]; }
```

SIMD units keep getting wider & supporting more operations

Compilers getting better at **auto-vectorization**

Operating directly on **encoded data**:

- RLE: `int sum(RLUnit u) { return u.length * u.value; }`
- Leverage dictionaries where possible

$vectorop()

New set of dedicated internal functions for operating on $vector data

- Tight loops, auto-vectorized, SIMD, RLE, ...: ☑

Functions for aggregates, filters, groupings, ...

```
set i = “”, sum = 0
for {
  s i = $order(^d.V1(i), 1, col1)  q:i=“”
  s filter = $vectorop(“=”, col1, “abc”)
  s sum = sum + $vectorop(“sum”, ^d.V2(i), filter)
}
```

Optimized Processing

$vector Operations

Handle 64k values at a time

- Leverage encoding scheme

Exploit modern CPU strengths

- Tight loops
- SIMD instructions
- Auto-vectorization

Vectorized SQL Processing

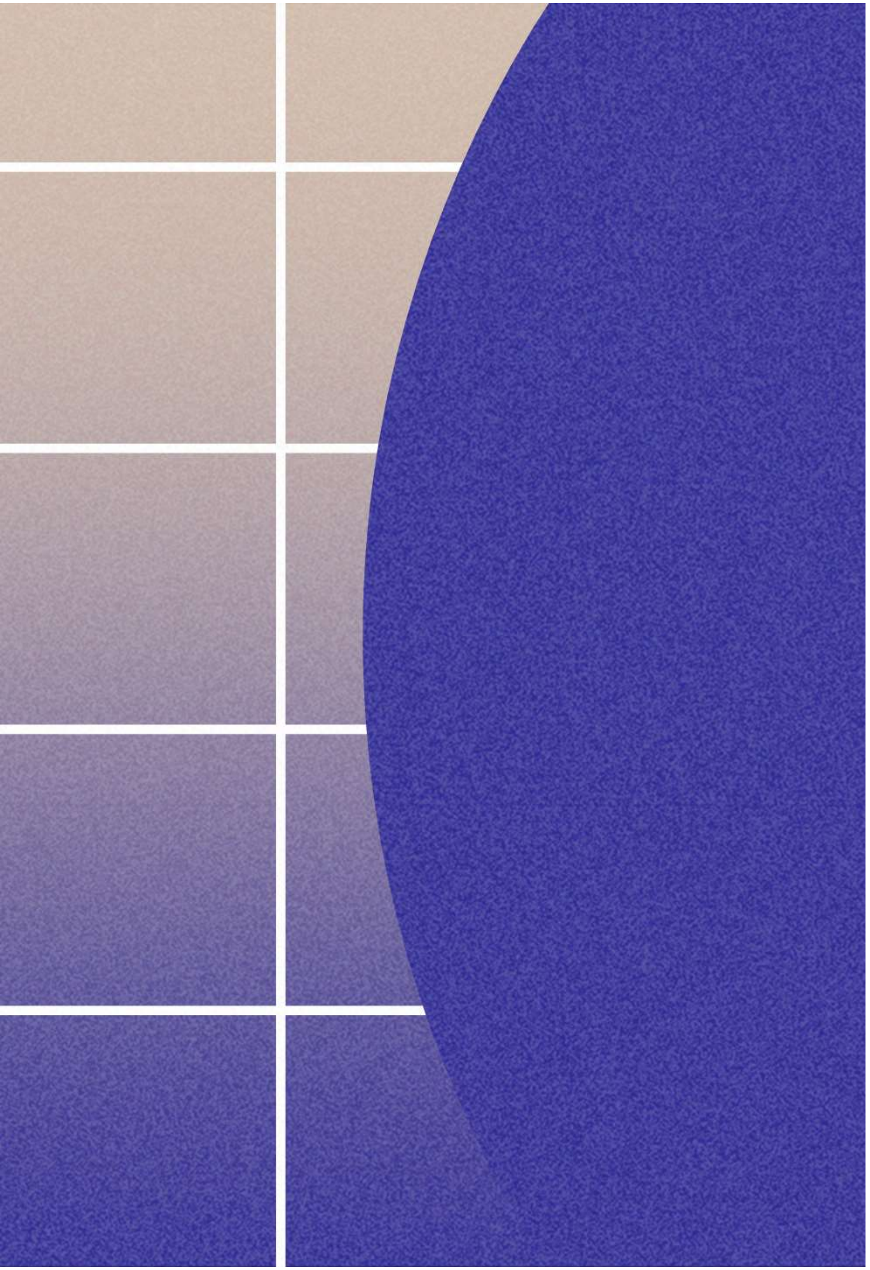
Leverage columnar layout

- Push $vector chunks throughout query processing
- Only read required columns
- Late row materialization

Adaptive Parallel Execution

We’re making Analytical Queries 10x Faster

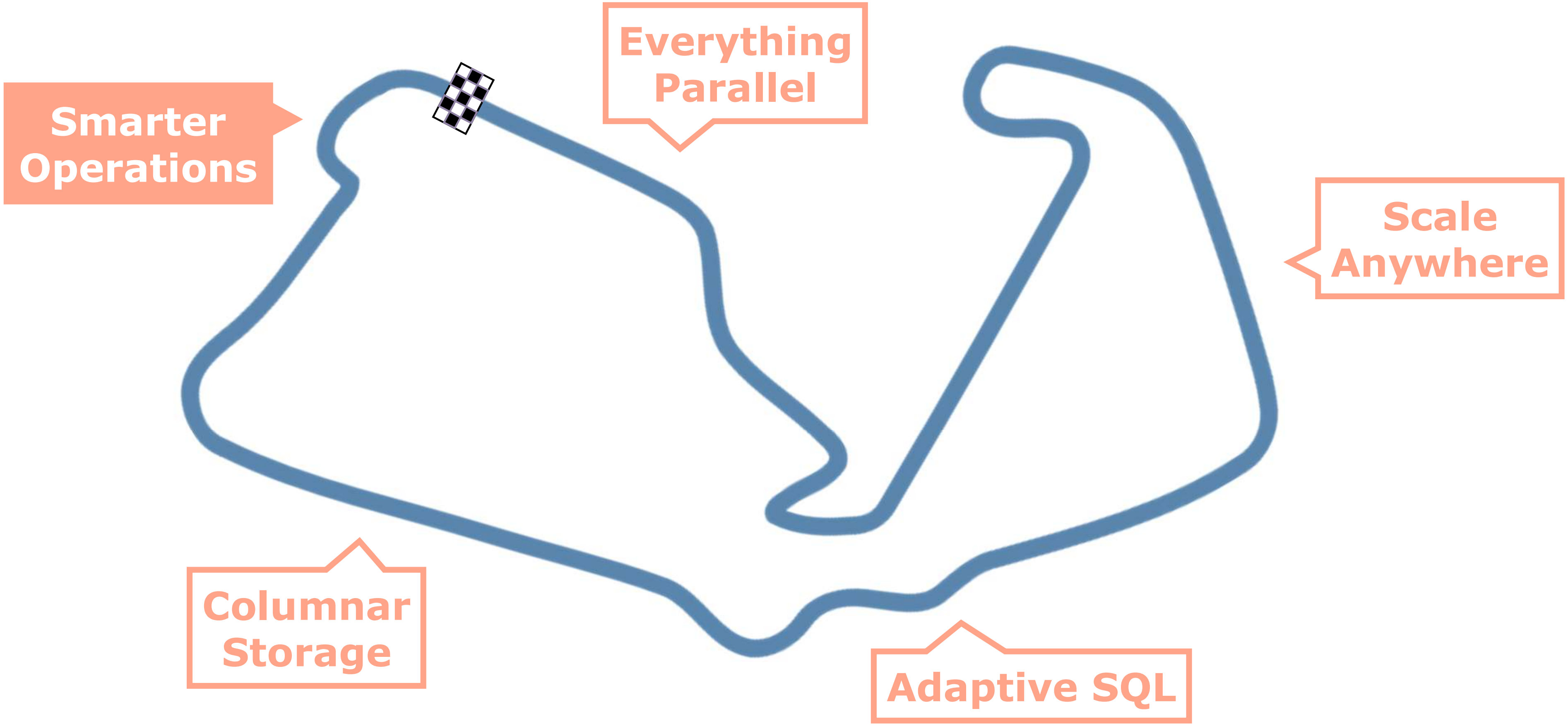
Available from IRIS 2022.2 as an Experimental Feature



Pit Lane: Smarter Operations

High-Speed Processing & SQL

High Speed – UKI Edition



Smarter Operations

Everything Parallel

Scale Anywhere

Columnar Storage

Adaptive SQL

Smarter Operations

Tired of juggling Tires?

Let us be your pit crew!

Best Practices

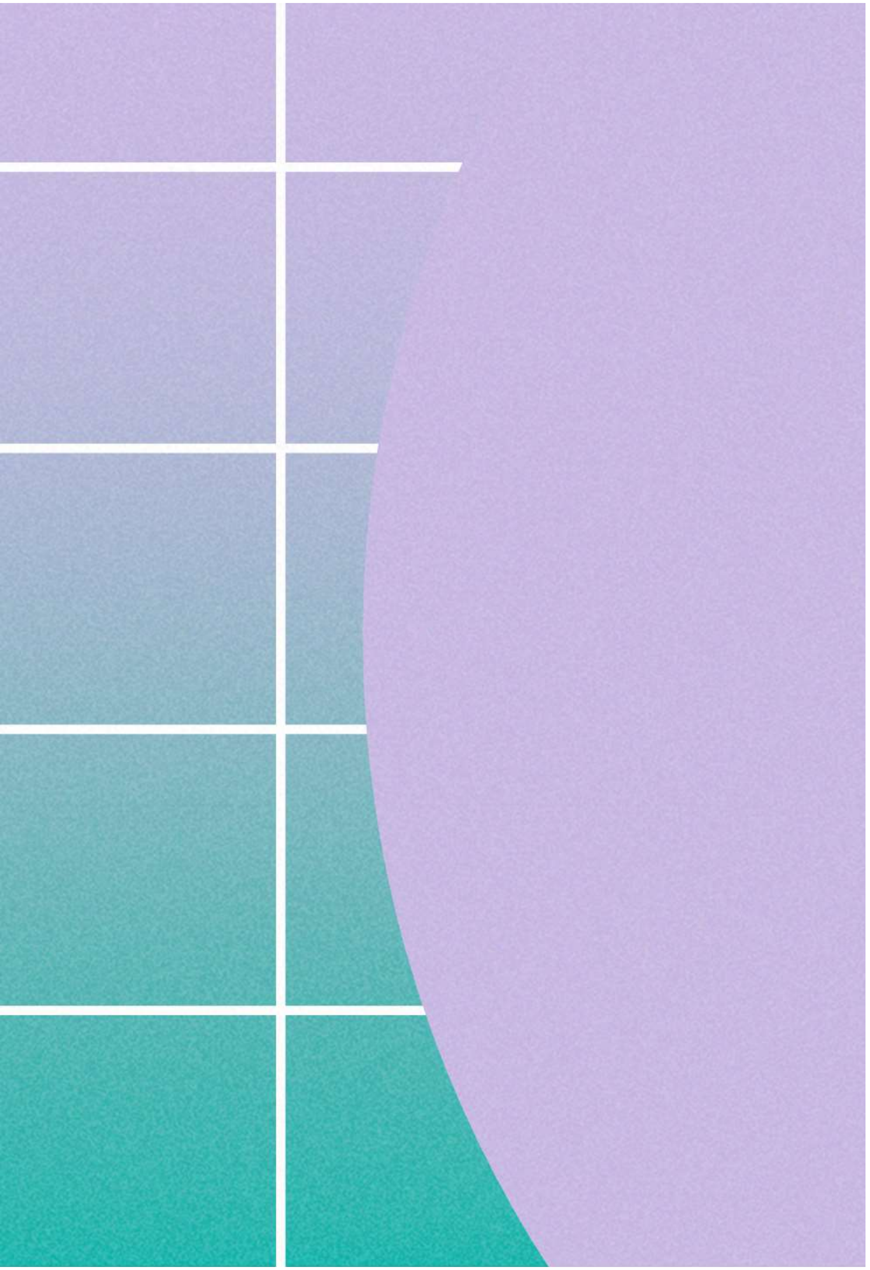
- Agile deployment (K8s)
- Observability & Monitoring
- Security

Managed Services

- IRIS & IRIS for Health
- Health Connect Cloud

Full SaaS

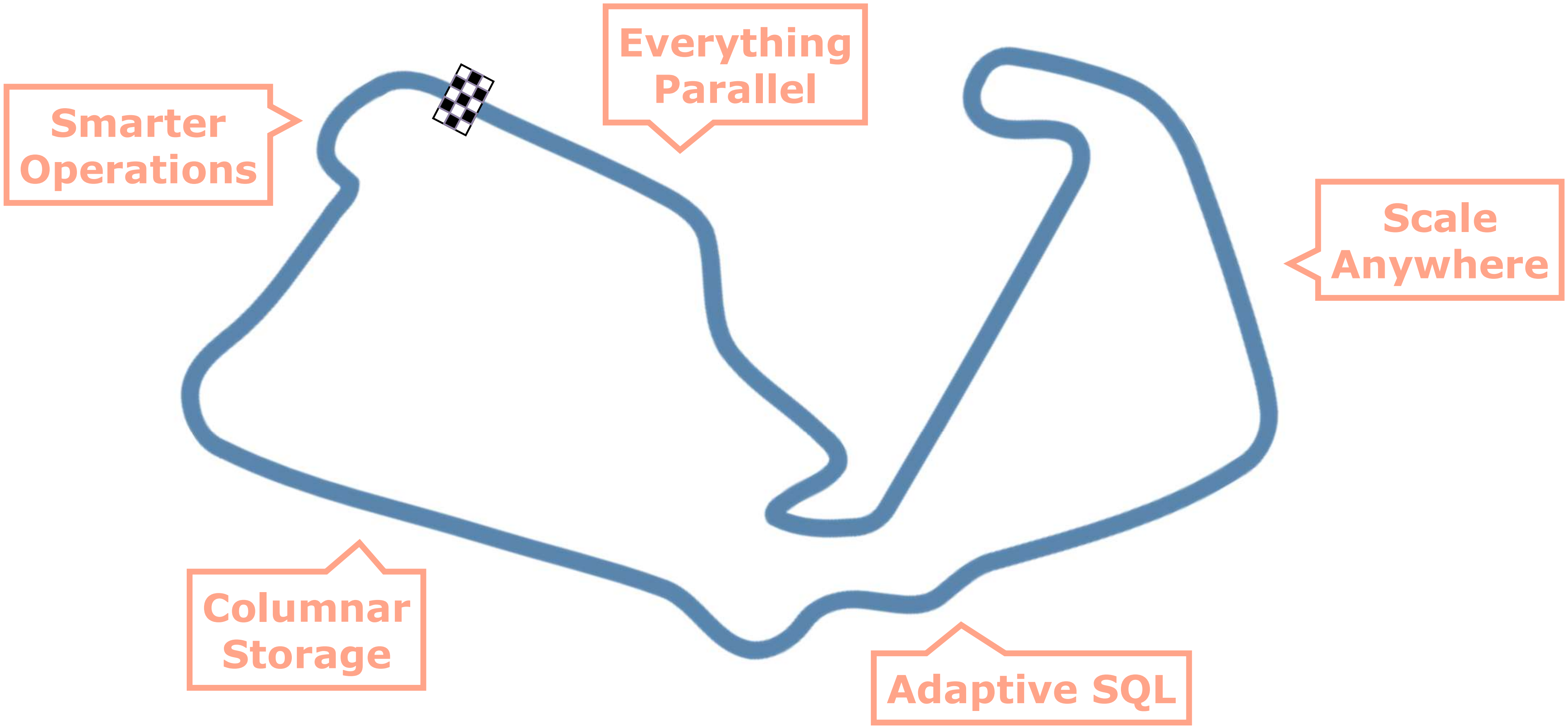
- IRIS Cloud SQL
- IRIS Cloud IntegratedML
- FHIR Server
- FHIR SQL Builder
- FHIR Transformation Service



Wrapping Up

High-Speed Processing & SQL

High Speed – UKI Edition



Smarter Operations

Everything Parallel

Scale Anywhere

Columnar Storage

Adaptive SQL

Wrapping up

Needles are for Moving

Application Transparency

Let us be your pit crew

Thank You

InterSystems®
Creative data technology