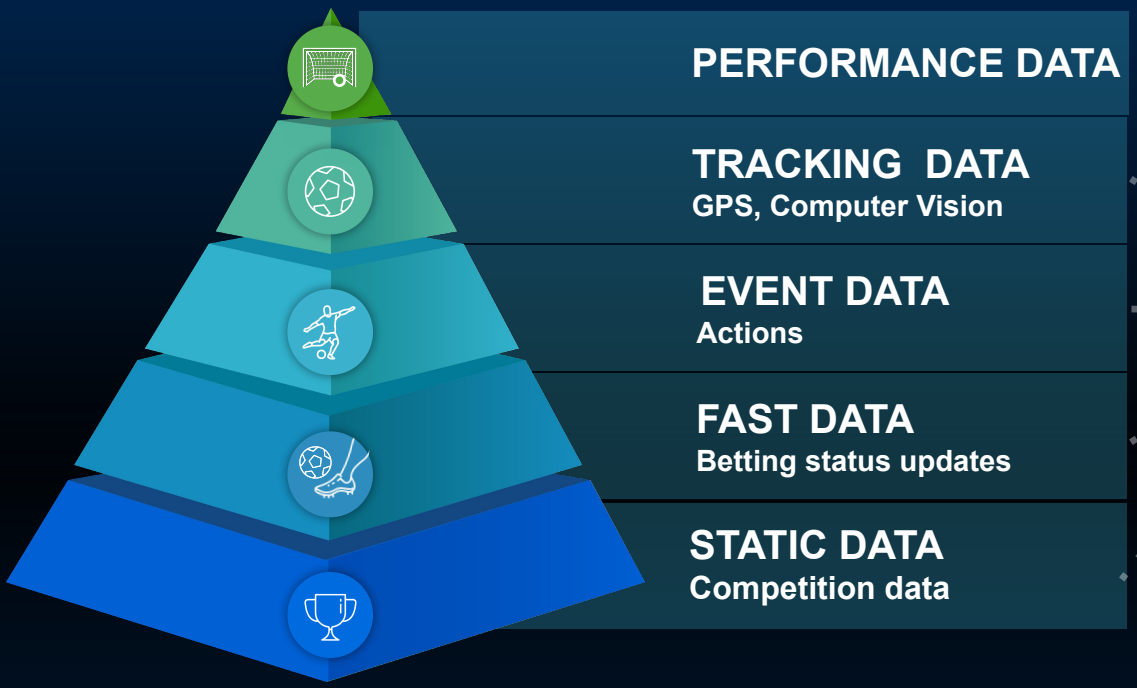
A SPORTS MODEL WITH DRUID

Paddy Green
IMGArena

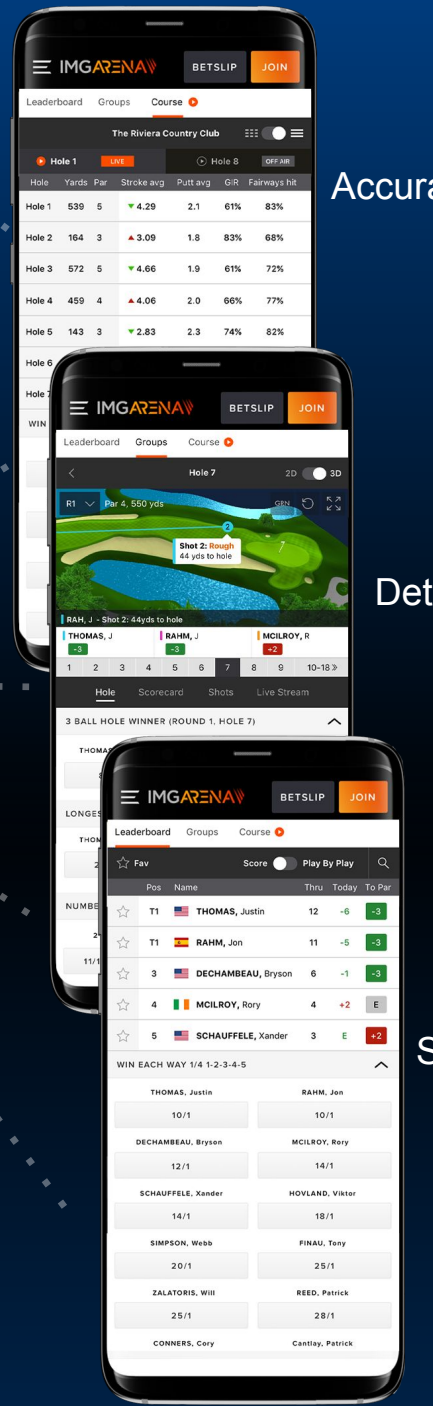
THE PROBLEM

DATA SOURCES IN SPORTS

Accelerating data science by joining the data feeds in a Data Warehouse



DATA WAREHOUSE



Accuracy of the **Event Data**

Detail of the **Tracking Data**

Speed of the **Score Data**

SPORT DATA COLLECTION DUPLICATION

Ball looks like its landed on the fairway

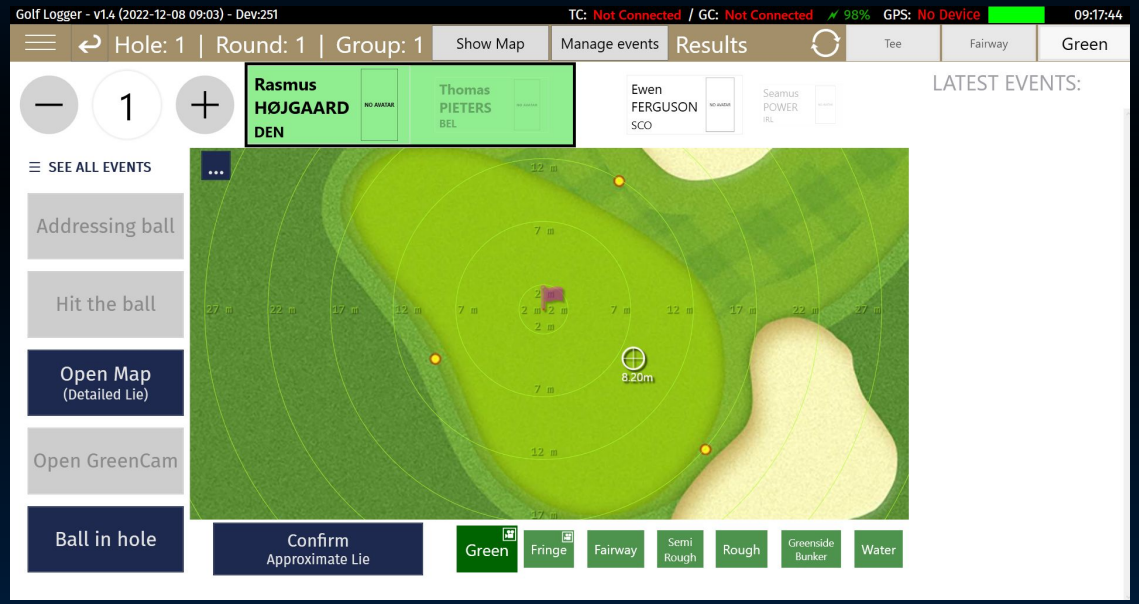
Walk over and the balls landed on a hill

Ball is blown off hill before next shot

Ball lands in the water drop behind on the tee

Distance travelled over stroke 1 is 0 feet

kafka



- Sport data is IOT data and prone to human errors, technical errors and missed packets
- Final data size is small but there can be several corrections for the same event

SPORT DATA LATENCY

Each data sources has different lengths of delay

<1s

<5s

<15s

kafka

STATIC DATA

FAST DATA

Entered by umpire or other official so prone to human error

TRACKING DATA

System delay for computations and prone to systematic errors

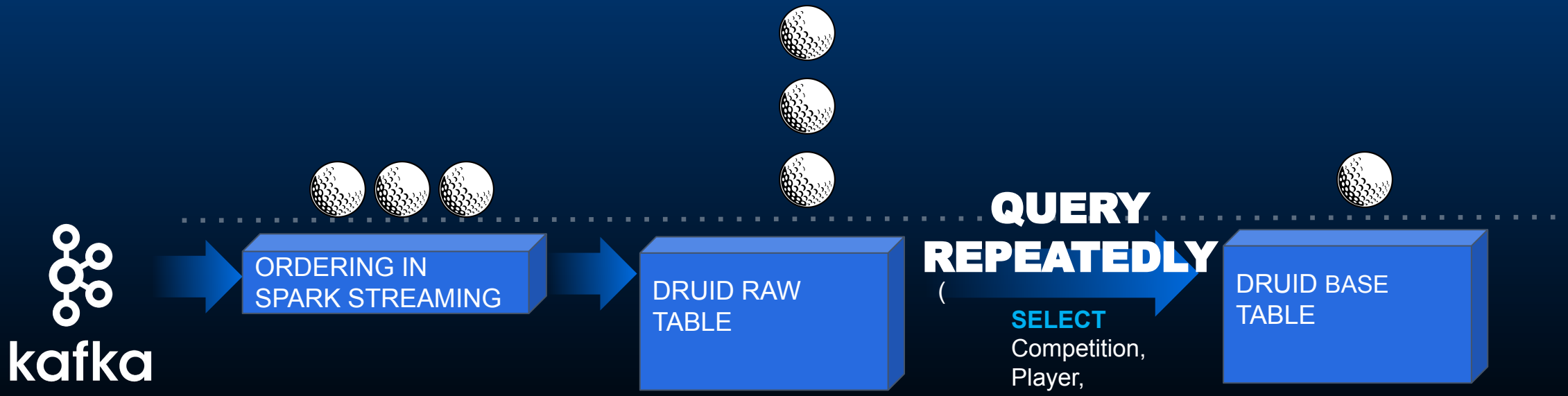
EVENT DATA

Often delayed for invigilation as derived from tracking data

- The data warehouse needs to deliver whatever information it can as quickly as it can
- These sources could be combined using spark and kafka streams jobs but we still need to version the rows that are output

STORING THE DATA IN DRUID

DEDUPLICATING EVENT DATA



- Labels can identify rows for deletion from the final table by filtering
- 2 queries are run in parallel every couple of seconds
- The first is to upsert today's segment with the latest data
  WHERE date = CAST(CURRENT_TIMESTAMP AS DATE)
- The seconds is to upsert yesterday's segment with the updates that may have come today
  WHERE date = CAST(CURRENT_TIMESTAMP AS DATE) - INTERVAL '1' DAY
+

(

```
SELECT
Competition,
Player,
Round,
Hole,
Stroke,
CAST(MILLIS_TO_TIMESTAMP(
   LATEST(TIMESTAMP_TO_MILLIS(__time), 2000)
) AS DATE) AS "date",
LATEST(SurfaceType, 200),
LATEST(DistanceToPin, 200),
….

WHERE NOT Deleted AND
__time >= CAST(CURRENT_TIMESTAMP AS DATE)
- INTERVAL '2' DAY
GROUP BY 1,2,3,4,5
```

)

BUILDING THE TABLE

Each data source needs joined into one single event level table but each competition can be monitored differently and some of the more detailed data sources won't be available

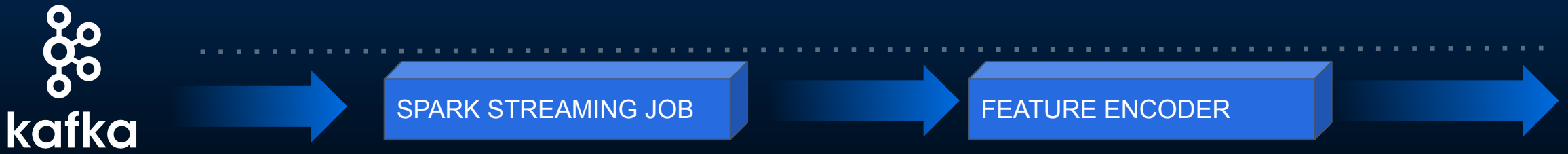
This means we require separate columns to be generated for each level of detail we receive

Time	Competition	Player	STATIC DATA	FAST DATA	EVENT DATA	TRACKING DATA
2022-01-01T10:10:01	1	A	Player name Tournament name Course name	Hole 1 finished scored 3	Landed on fairway	Drive category A
2022-01-01T10:14:11	1	A			Landed on green	
2022-01-01T10:22:00	1	A			Ball in hole	
2022-01-01T10:25:03	1	A		1 Penalty on Hole 2	Landed in water	Drive category B
2022-01-01T10:34:01	1	A			Landed on fairway	Drive category C

INTEGRATING THE TRACKING DATA

TRACKING DATA

Using a session window keyed by event we pre aggregate our tracking data and encode the information we care about

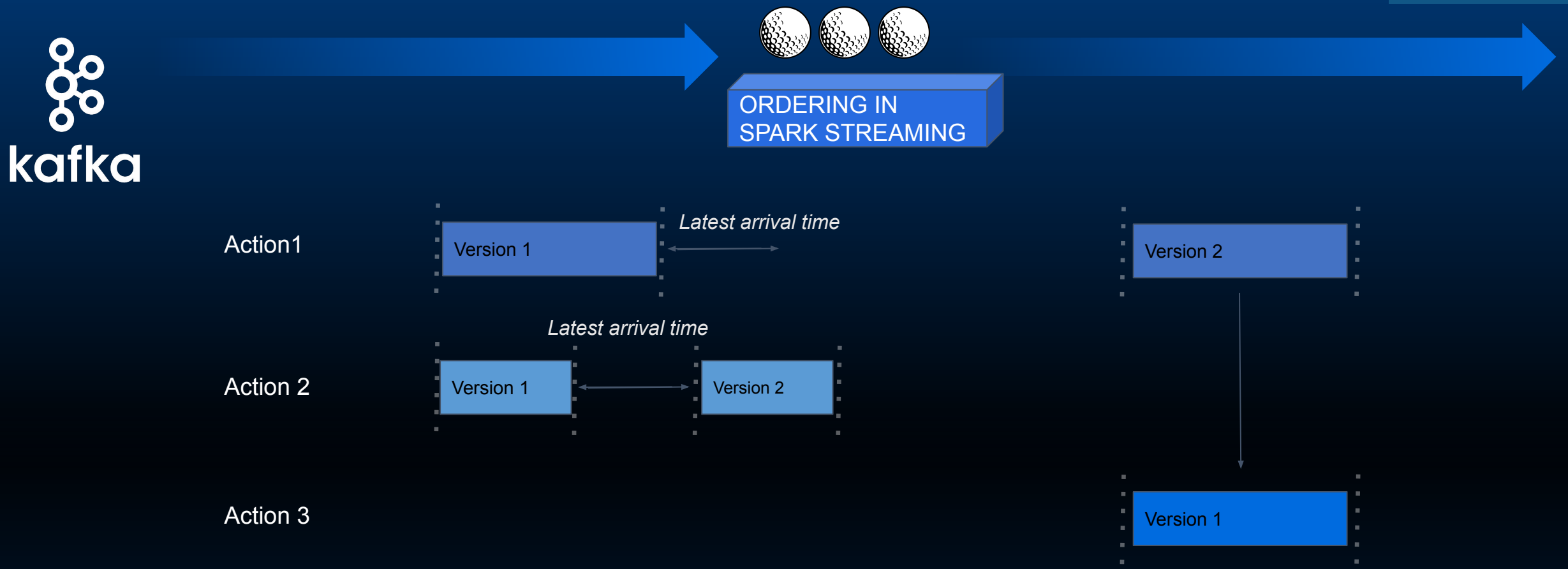


Time	
2022-01-01T10:10:01	{“event”:1, “ball”: [0.0,0.0 0.1,0.0, 0.1,0.0, 1.0,0.0, 1.2,0.0…
2022-01-01T10:12:01	{“event”:2, “ball”: [0.0,0.0 0.1,0.0, 0.1,0.0, 1.0,0.0, 1.2,0.0…

Time	
2022-01-01T10:10:01	Drive category A
2022-01-01T10:12:01	Drive category B

INTEGRATING EVENT DATA

EVENT DATA



- Keying by action (ball hit, score update ….) is not always clear enough so we also need to dedupe based on the time
- Versioning of the action are decided based on the timestamp ordering
- **Action 1** has an update that arrives after the latest arrival time so it becomes a separate action **Action 3**
- **Action 2** has an update that arrives before the latest arrival time so this action is update with a new version
-

INTEGRATING THE FAST DATA

FAST DATA

Score, Status and Important Stats

- We derive these high level columns from the **Event Data** to ensure the final version of these columns are most accurate
- In the cases where event data has not been received we convert the **Fast Data** into the same format as **Event Data** to ensure completeness of the table
- We add this to the deduplication query also

```
……
LEFT JOIN (
  SELECT ……, [SCORE+STATUS AGGREGATIONS], ……
  FROM [RAW TABLE QUERY]
  GROUP BY 1,2,3,4
)
ON player ….
```

INTEGRATING THE STATIC DATA

STATIC DATA

Names, IDs and Static information

- We derive these static values from the **Event/Fast Data** to ensure the final version of these columns are the most accurate they could be
- This **Static Data** can be integrated during deduplication via lookups

```
……
 LEFT JOIN (
   SELECT
   LOOKUP([STATIC DATA KEYS],  [STATIC DATA LOOKUP])
   FROM [RAW TABLE QUERY]
 )
ON player ….
```

QUERYING THE DATA IN DRUID

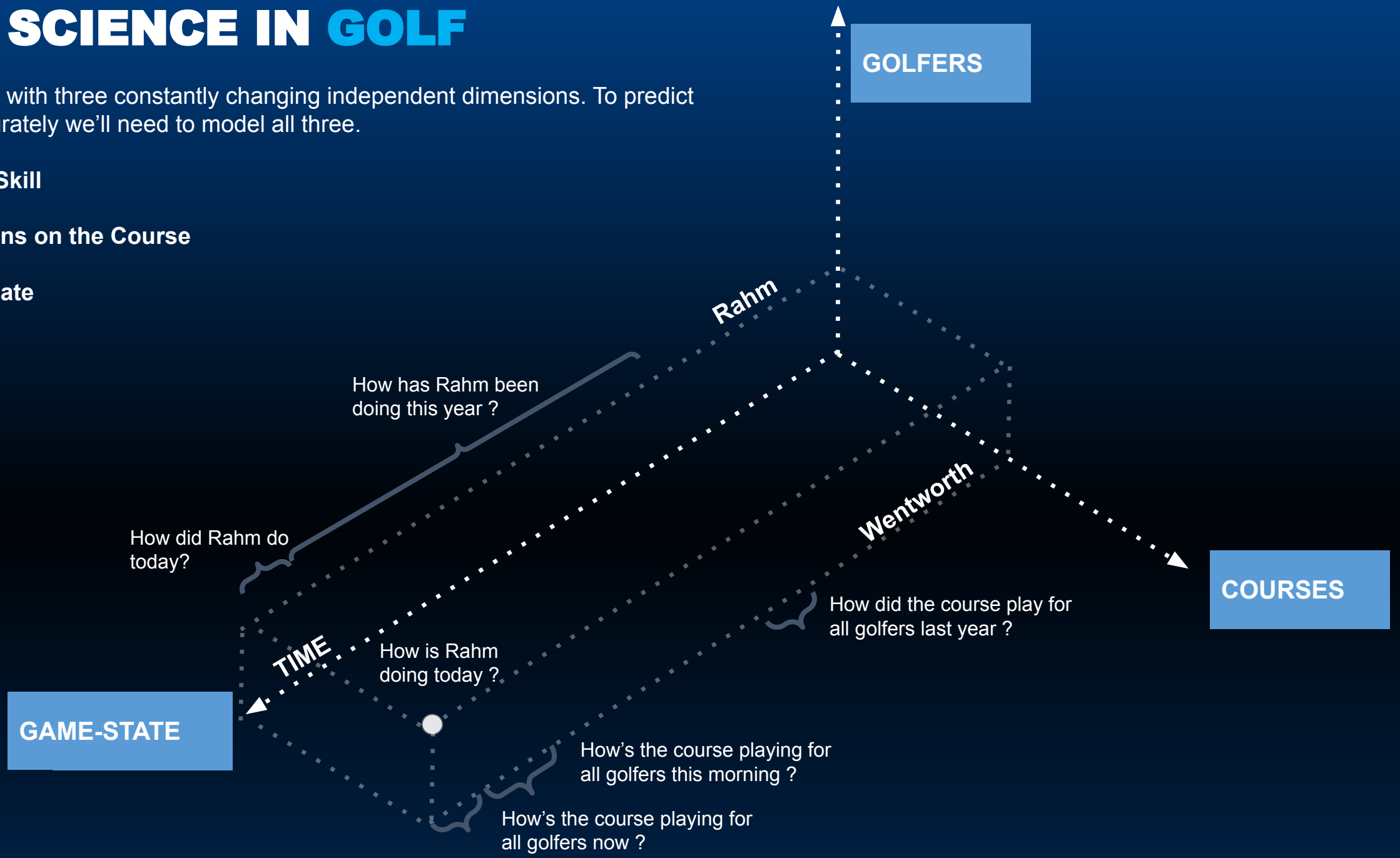
DATA SCIENCE IN GOLF

We're dealing with three constantly changing independent dimensions. To predict anything accurately we'll need to model all three.

The Golfers Skill

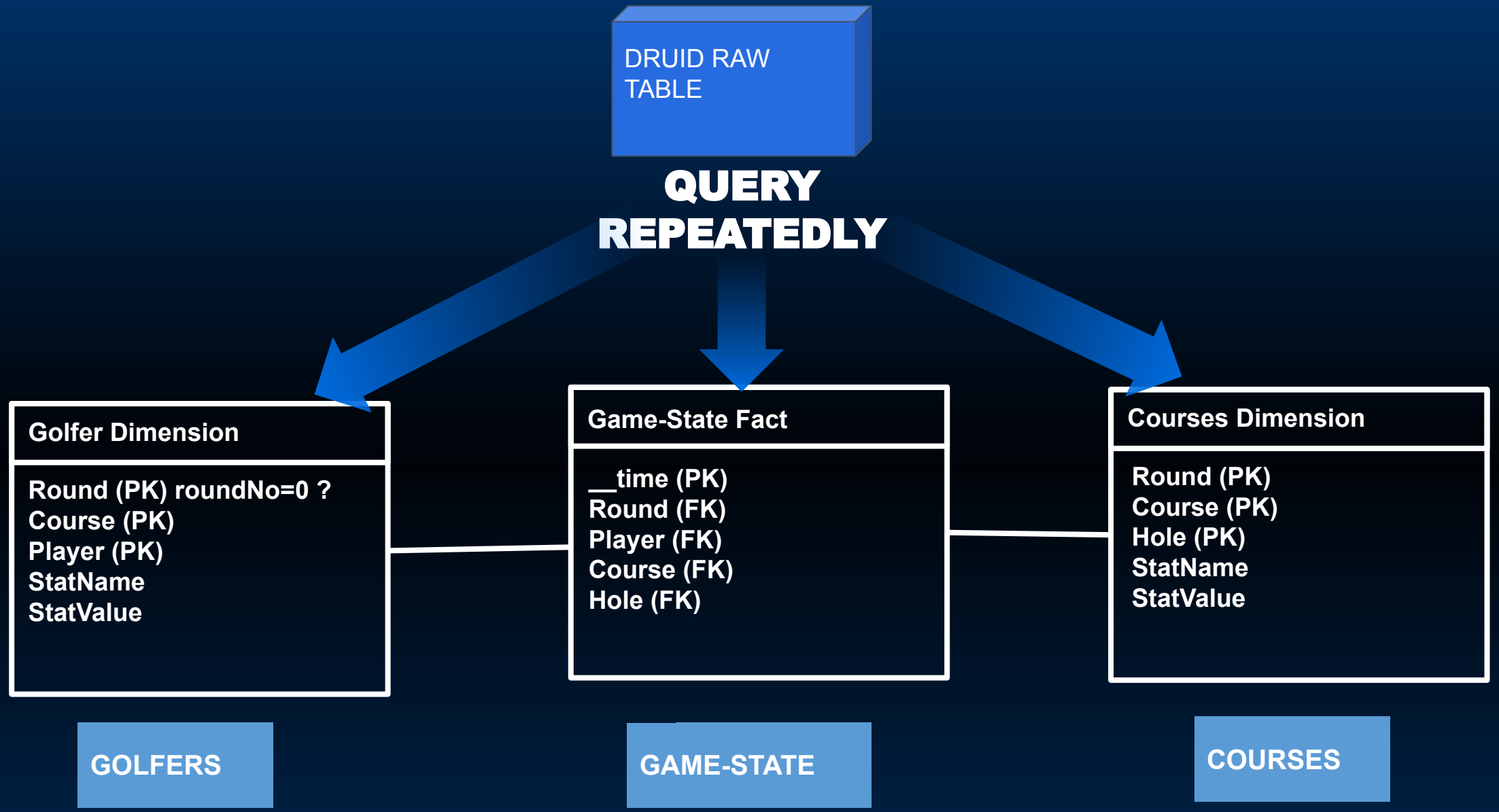
The Conditions on the Course

The Game State



A DATA STRUCTURE FOR DATA SCIENCE

Both golfer skill and course conditions will converge their values per round so we should create 3 deduplication jobs generating 3 tables instead of 1



HISTORICAL GOLFER + COURSE FEATURES

Golfer Features require weighing past data to derive current skill and also normalize observed skill based on course difficulty

- We can create these features when building our gold table
- To normalise skill we can self join to the course difficulty by adding to the deduplication query
- As for weighing the past we can self join to each time range we want to create features from

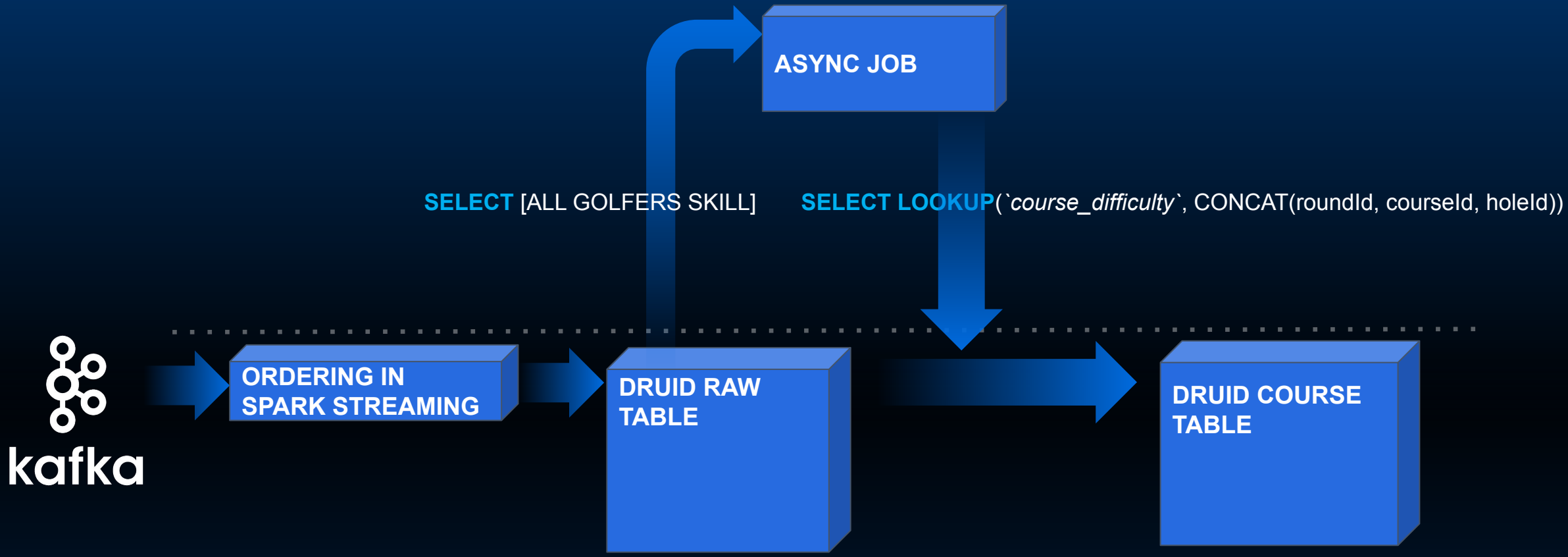

```
SELECT
 [GOLFER SKILLS]
….

FROM (
   [RAW TABLE QUERY]
)
….

LEFT JOIN (
   SELECT [GOLFER SKILLS] / [COURSE DIFFICULTY]
   FROM [RAW TABLE QUERY]
   LEFT JOIN (
      SELECT [COURSE DIFFICULTY]
      FROM [RAW TABLE QUERY]
      GROUP BY 1,2,3,4
    )
   ON round….
   WHERE __time
   GROUP BY 1,2,3
)
ON player ….
```

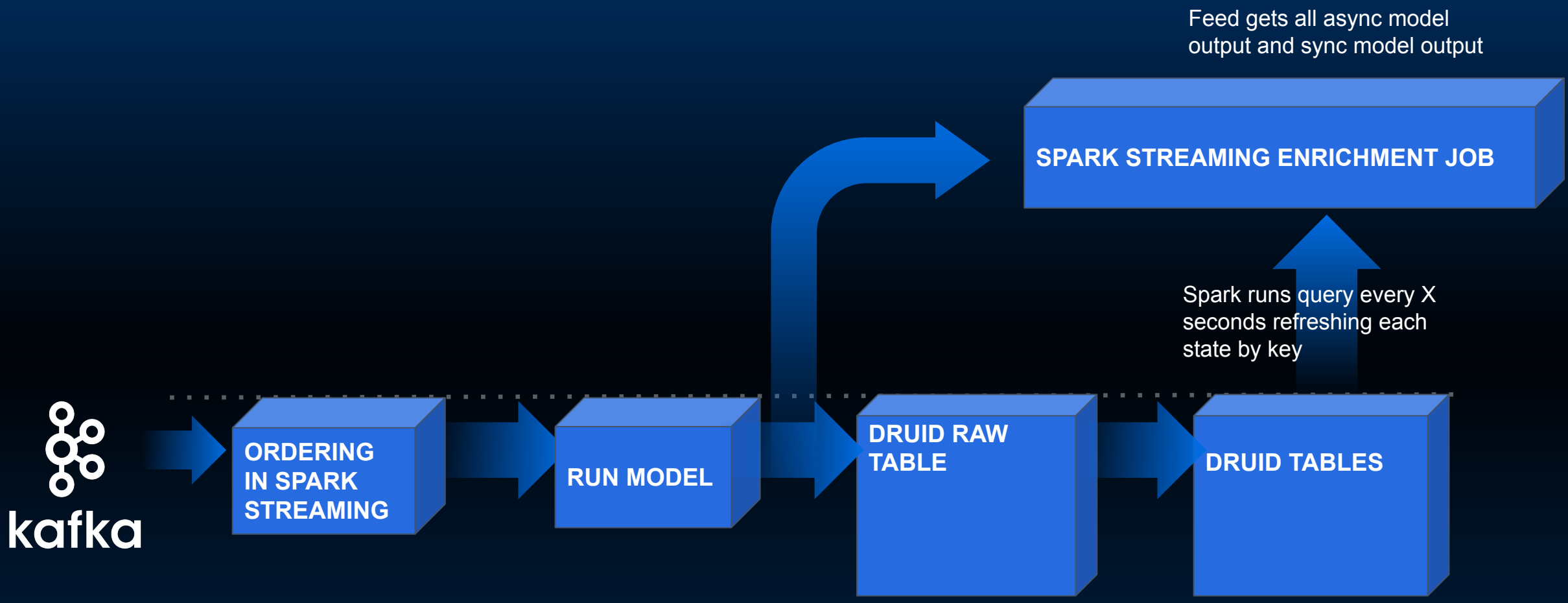
ASYNC MODELS ON RECENT GOLFER + COURSE FEATURES

Game Stat Features change slowly so models requiring them doesn't need run on every update to the data

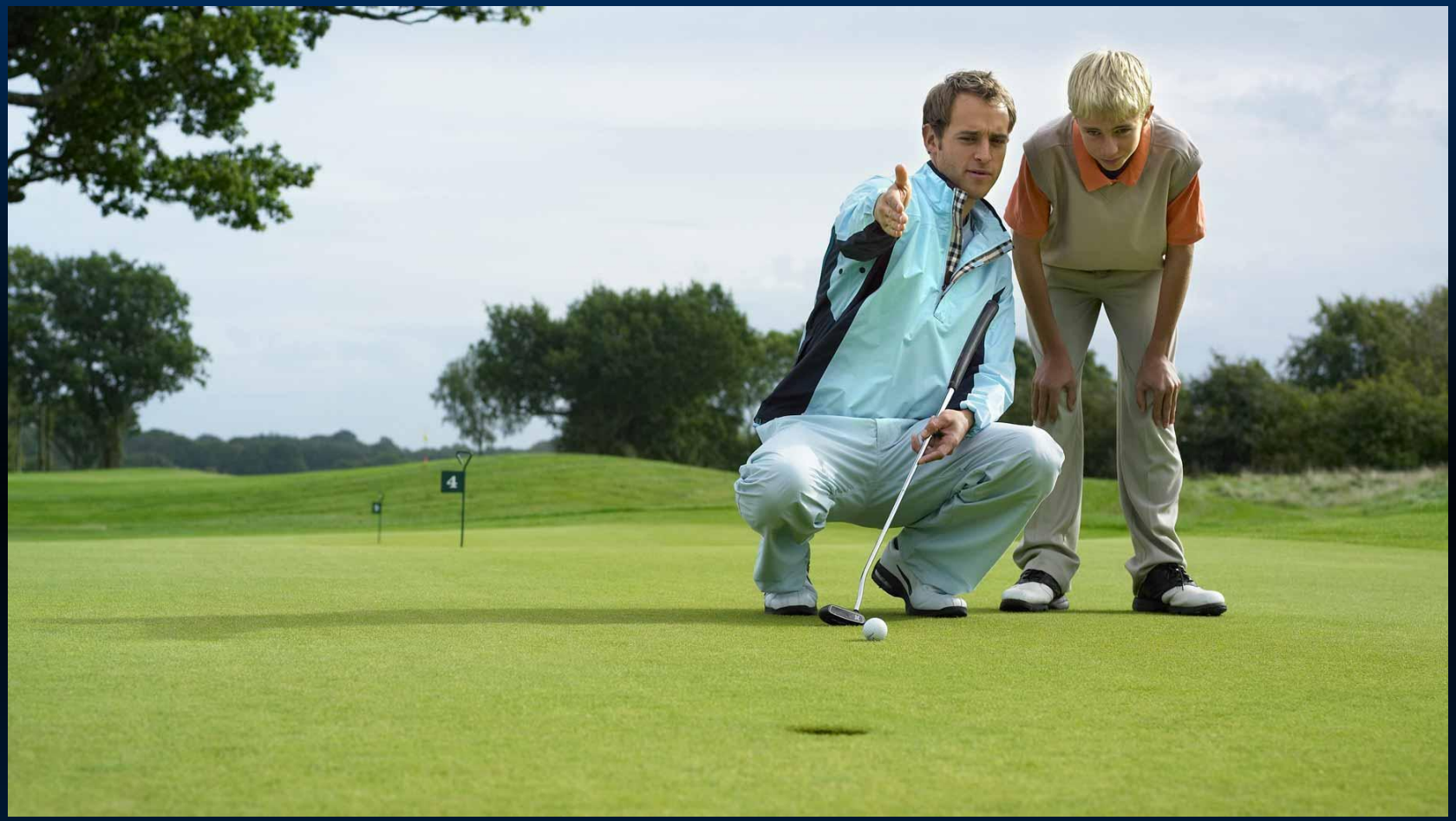


SYNC MODELS WITH GAME-STATE FEATURES

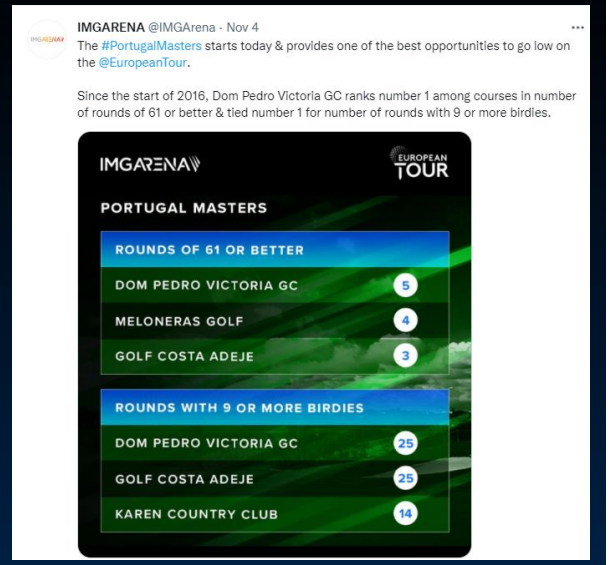
Models requiring Game State features need run in stream and results can combined at the end of the pipeline



DRUID TIPS



Subquery generated results beyond maximum[100000]



- Using apache superset we can derive almost real time graphics from a full history of a tournament or golfer
- Any large nested joins can be reduced using jinja templating

```
WHERE 1=1
{%- for filter in get_filters('tournamentId', remove_filter=True) -%}
{%- if filter.get('op') == 'IN' -%}
 AND tournamentId IN {{ filter.get('val')|where_in }}
{%- endif -%}
{%- endfor -%}
```

Apache Superset

WINDOW FUNCTIONS

How do we query the 2nd lowest scoring hole on the course ?

```
SELECT
  MV_ORDINAL(holeno_arr, MV_ORDINAL_OF(hole_score_arr, CAST(second_min_hole_score AS VARCHAR))) AS second_min_hole

FROM (
  SELECT
    ARRAY_AGG(holeNo) AS holeno_arr,
    ARRAY_AGG(hole_score) AS hole_score_arr,
    DS_GET_QUANTILE(DS_QUANTILES_SKETCH(hole_score, 32), 1.0/18.0) AS second_min_hole_score

  FROM (
    SELECT
      holeNo,
      SUBSTR(CAST(AVG(score) AS VARCHAR),1,4) as hole_score
    FROM [PRESENTATION TABLE]
    WHERE __time >= CURRENT_TIMESTAMP - INTERVAL '#' DAY
    AND tournamentId='####'
    GROUP BY 1
  )
)
```

COST OPTIMISATIONS KUBERNETES

Adding more smaller replicas increases availability and allows for cost savings through spot instances

On-Demand	Spot
Coordinator	Historical
Overloads	Middle Manager
RDS Metastore	Router
Zookeeper	Broker

```yaml
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: eks.amazonaws.com/capacityType
                operator: In
                values:
                  - SPOT
  tolerations:
    - key: "SPOT_INSTANCE"
      operator: "Equal"
      value: "true"
      effect: "NoSchedule"
  topologySpreadConstraints:
    - maxSkew: 1
      topologyKey: topology.kubernetes.io/zone
      whenUnsatisfiable: ScheduleAnyway
      labelSelector:
        matchLabels:
          app: historical
```

- Many small replicas are highly available and cheaper to run than 1 large replica
- For query nodes, adding more parallelism also helps to ensure the human errors in the data collection stage don't affect all feeds of data and only slow the golfer being recorded
- Also losing data nodes does not lose data but can occasionally slow the ingestion, this happens such small amounts of the time that its negligible in comparison to late corrections data collectors make anyway