intuit turbotax quickbooks mint

Intuit API: Caching at the Edge

October 31, 2019

Philip Russell

Meet Your Speaker

- Staff Software Engineer at Intuit
- Services APIs & Infrastructure
- Compilers, Language/API Frameworks and Distributed Systems

Let's talk About...

- ❑ GraphQL at Intuit
- ❑ Caching
- ❑ Control Mechanisms
- ❑ Eventing and Caches

GraphQL at Intuit

Intuit Scale

1800+ Entities (Query-able Types)

10000+ Expanded Types

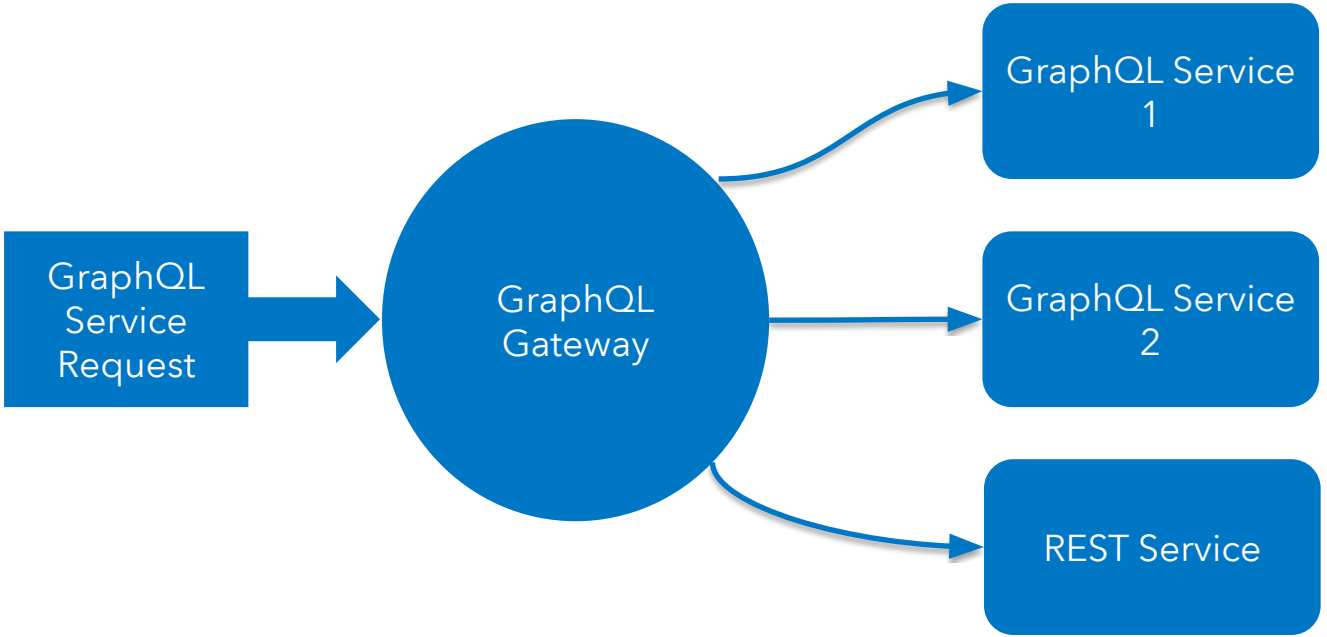
> 1 billion Requests / Day

100+ Product Teams

Thousands of Services

Question: how do we find & Query N different Services?

Solution 1: GraphQL Gateway



Problems at Scale

Assume 100 different Services

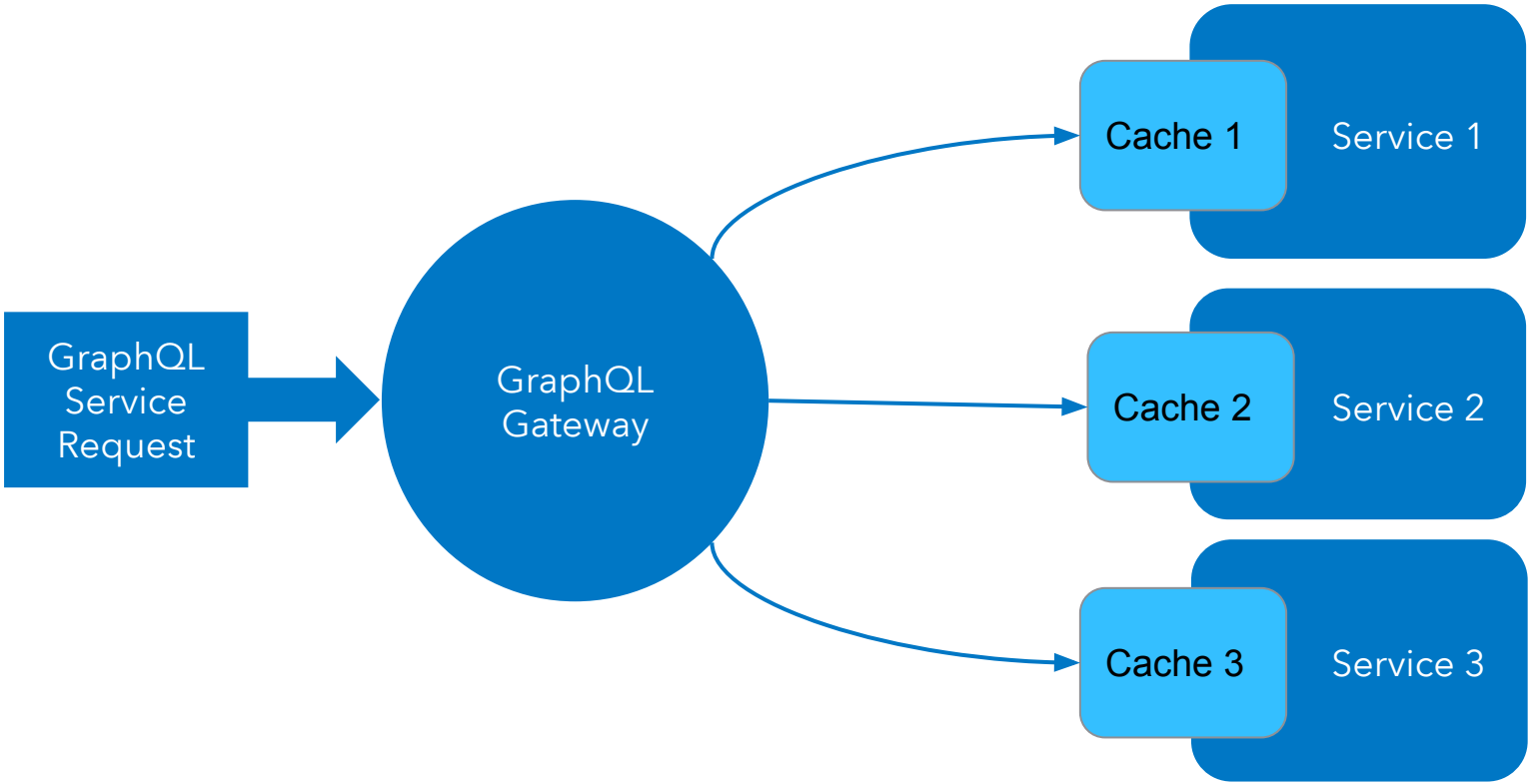
Clients (assuming Access Control) can Query all 100

N+1 Problem! 🥵

Solution 1: collate common fields/sub-objects to the same Service

Solution 2: cache everything, mitigate effects of N+1

Solution 2: Cache at Services



Problems with Service-Level Caching

100 different Services -> 100 different Caching layers

No way to coordinate them!

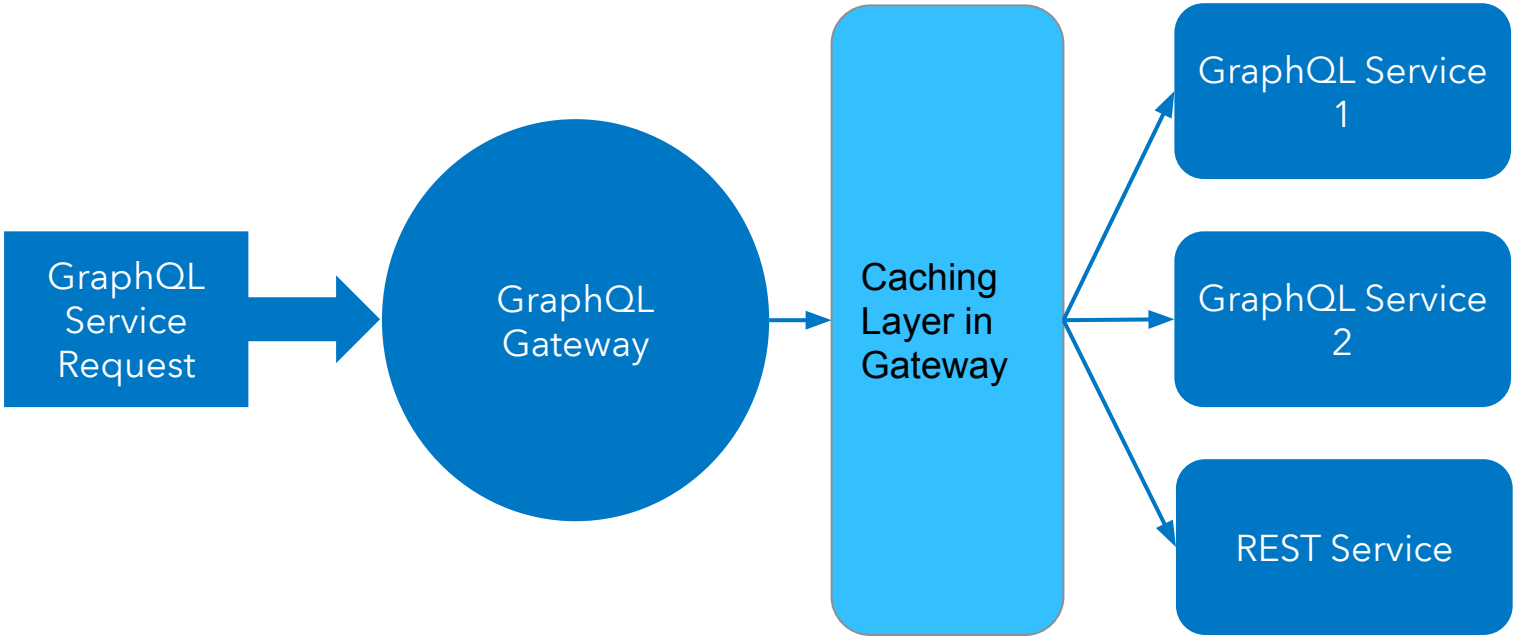
Cache for Every Service

Duplicate and Disparate Data

How to Solve: Cache at the Edge

Common Data Layer

Solution 3: Cache at Edge



Caching at the Edge

Just Request/Response Caching not good enough

Query1 goes to Service 1+2, Query2 goes to Service1+3

Naive approach just caches Response, duplicate data!

Matters at scale

Is there a way to Store and Find common components?

Solution: Overlapping Response Caching

Overlapping Response Caching

Basic Idea

Query 1

```
film(filmID :"1") {
  ...
  characterConnection {
    characters {
      name
      id
      homeworld {
        name
        diameter
        rotationPeriod
      }
    }
}
```

Query 2

```
person(id : "cGVvcGxlOjE=") {
      …
      name
      id
      homeworld {
        name
        diameter
        rotationPeriod
      }
}
```

How can we compare Query Components?
We need a common Key across Queries

ObjectId

How can we identify Objects in the Graph?

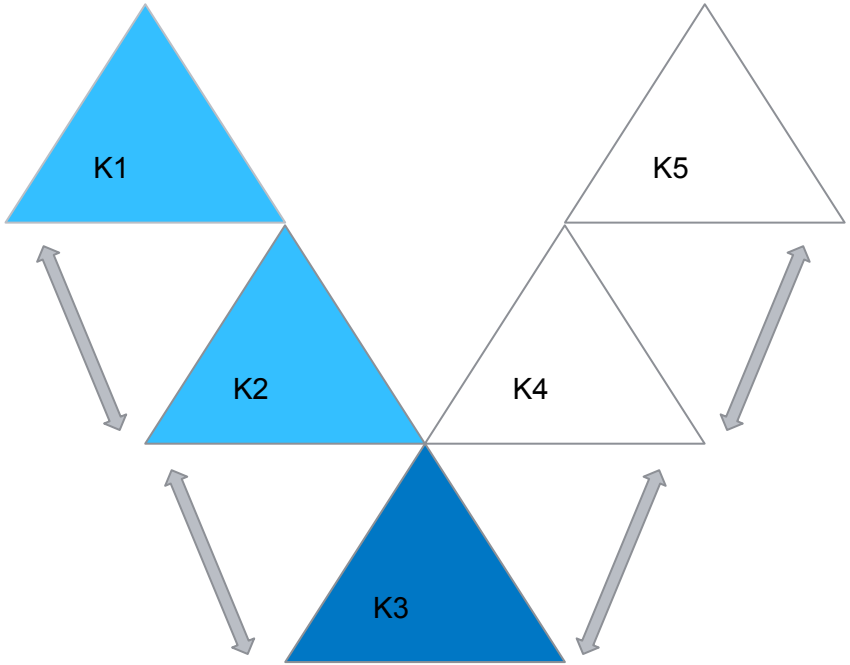
Need an ObjectId

Same Across Requests

Needs to be a Globally Unique Id

ObjectId = Hash(Domain, Type, LocalId)

Graph with Keys



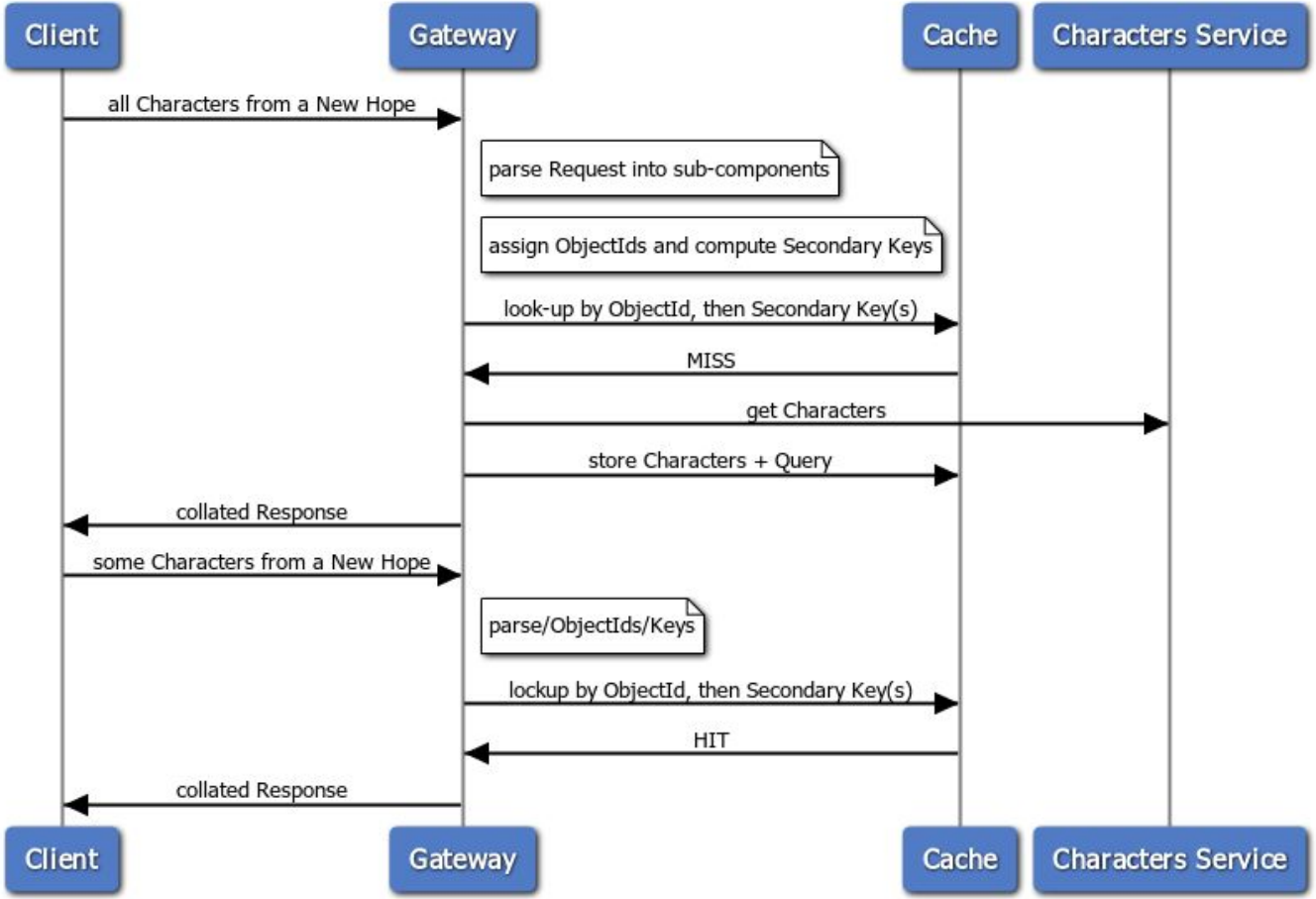
Use GraphQL framework to parse Request into AST

Use a Visitor to annotate the AST with ObjectIds

Store each Type Instance in a Table

K3 is the same for both Requests!

Cache Miss/Set across Queries



Direct Lookups

```
person(objectId : “1234") {
        …
        name
        id
        homeworld {
          name
          diameter
          rotationPeriod
        }
}
```

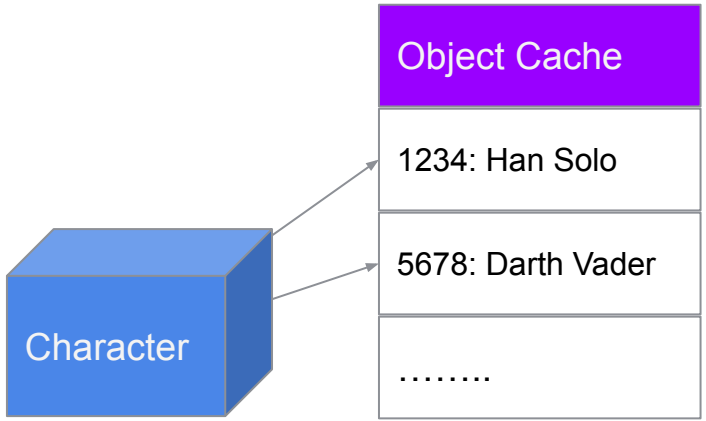
SELECT FROM Person WHERE objectId = ‘djQuM…’

Direct Id lookup => Primary Key

Id is an ObjectId, so just use that for Cache Set/Get

Works for single-object case, what about multiple objects? Non-id Keys?

Cache Keys: Primary Key Case



Direct Query-by-Id => we have the ObjectId

ObjectId is Primary Key for Get/Set

Is a Leaf Node in the Graph-in-Cache

More Complicated Queries

```
film(objectId :"abcd") {
  ...
  characterConnection {
    characters {
      name
      id
      homeworld {
        name
        diameter
        rotationPeriod
      }
  }
}
```

Cases to Handle

SELECT FROM Characters WHERE ParentKey = ‘abcd’

SELECT FROM Characters WHERE id IN (‘1234’, ‘5678’, ...)

SELECT FROM Characters WHERE ParentKey IN (‘abcd’, ‘efgh’, ...)

ParentKey is the ForeignKey => pointer to parent in the Graph

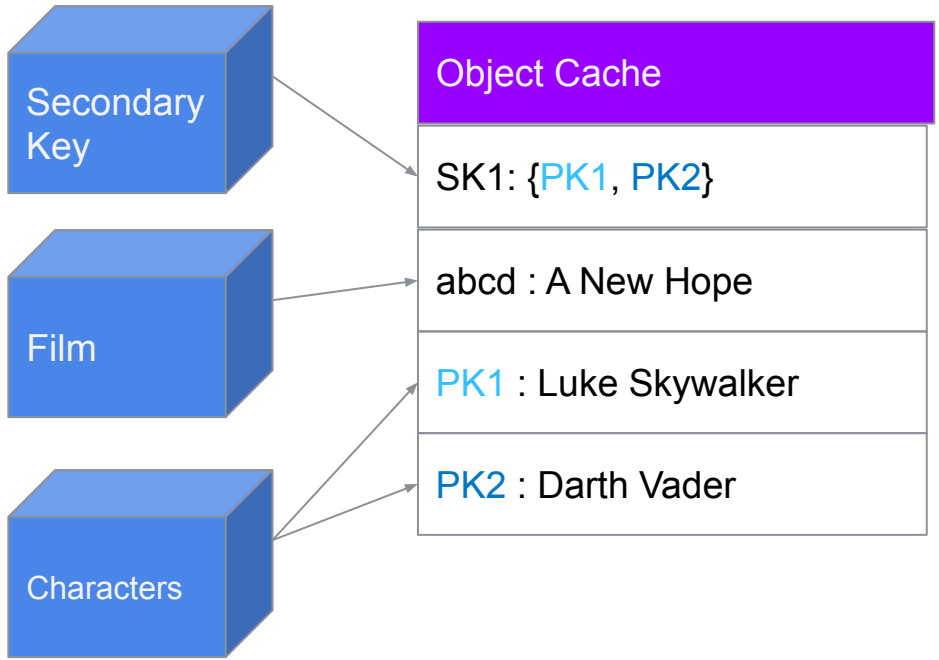
Cache Keys: Secondary Keys

Cases: == ForeignKey, IN ForeignKeySet, IN PrimaryKeySet

Can cover all of them using the same Secondary Key format

SK = Hash(Type, FieldKey, FieldValue)

SELECT Characters WHERE ParentKey (Film) == abcd => Hash(Product, ParentKey, ObjectId(abcd))



Secondary Keys Example

Q1: SELECT Characters WHERE ParentKey = abcd

MISS, Data Response is:

```
{
  “data” : [
    { “name” : “LukeSkywalker”, “object_id” : 1234},
    { “name” : “Darth Vader”, “object_id” : 5678}
  ],
  “parent_key” : “abcd”
}
```

Three Entries:

1. 1234 : Luke Skywalker
2. 5678 : Darth Vader
3. SK(Character, ‘ParentKey’, ‘abcd’) : {1234, 5678}

Q2: SELECT FROM Characters WHERE object_id = 1234 -> HIT

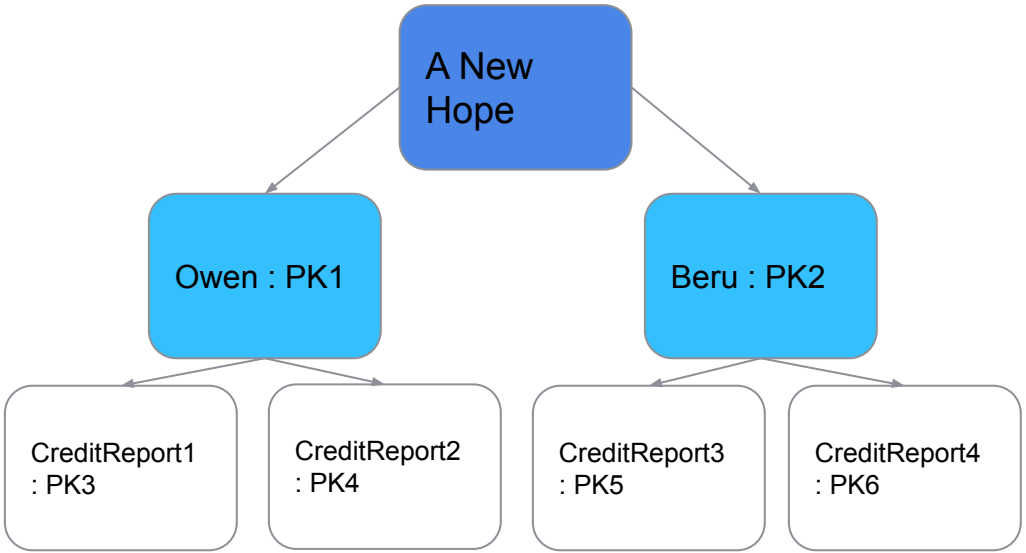
Q3: SELECT FROM Characters WHERE object_id = 5678 -> HIT

Q4: SELECT FROM Characters WHERE object_id IN {1234, 5678} -> HIT

Q5: SELECT FROM Characters WHERE ParentKey = abcd - HIT

Graph in the Cache

```
{
  film(filmID :"1") {
    characterConnection {
      characters {
        ... on Person {name, CreditReport}
      }
...
```



Object Cache
SK1: {PK1, PK2}
PK1 : Owen
PK2 : Beru
SK2: {PK3, PK4}
PK3: CreditReport1
PK4: CreditReport2
SK3: {PK5, PK6}
PK5: CreditReport3
PK6: CreditReport4

Comparing Queries across Requests

```
{
  film(filmID :"1") {
    planetConnection {
      totalCount
    }
  characterConnection {
    characters {
      name
      birthYear
      homeworld {
        name
      }
...
```

>=

```
{
  film(filmID :"1") {
    characterConnection {
      characters {
        name
      }
    }
  }
}
```

- S1 = {name, birthYear, homeworld.name}
- S2 = {name}
- S1.isSuperSet(S2) is True!
- Absolutely must store original Query with ObjectIds to do this comparison
- Increases Cache Utilization! If we Cache Query with N fields, all 2^N field combinations can be served from Cache
- If ask for SuperSet, then go to DataService and overwrite Cache Entries
- If using additional Input Variables, have to canonicalize (ie sort) them and store for later comparisons in the original Query

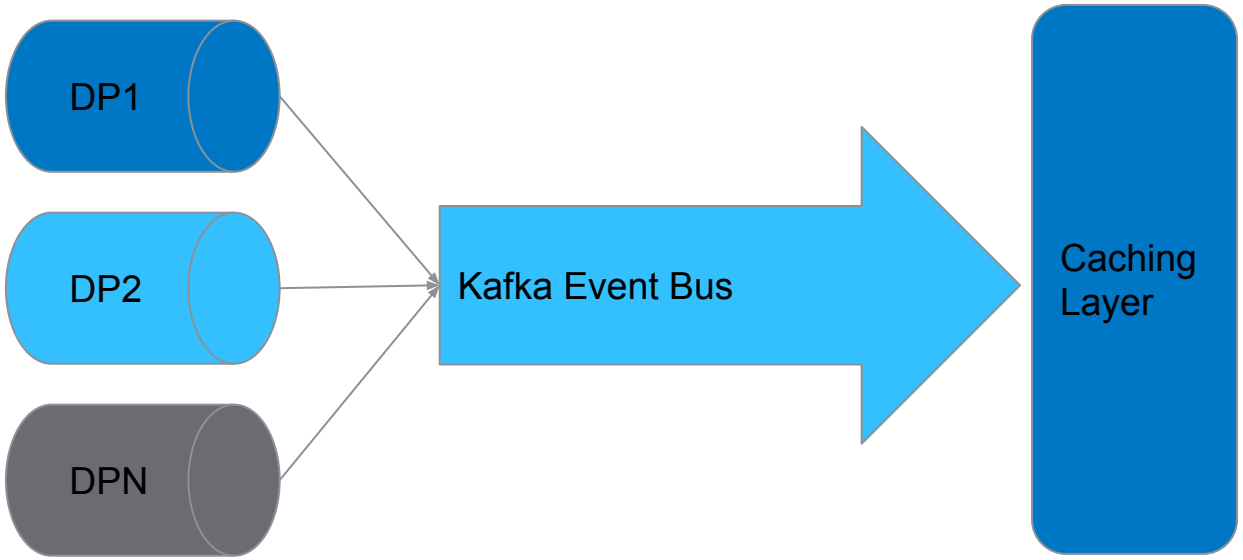
Cache-Control

Data Service controls their Cached Data (it's theirs…). How?

Use standard Cache-Control Header

Cache-Control value	Who sets it?	Effect
no-cache	Client	Get data from Provider and bypass the Cache.
no-store	Data Service	Do not set data in Cache
max-age>=0	Data Service	Set Response Data for specified amount of time. 0 'abused' to mean Purge Entry

Caching and Events (Future)



Overlapping Response Caching is about improving Cache Hit Rate. What about Data Freshness? Data Normalization?

Solutions: enforce common interface across the stack via code generation. Collate Events with same Id inside time “buckets” for data SnapShot.

Sum-Up: Main Points

ObjectId that is Consistent Across Requests (Globally Unique Id) for each Type

If Select by Id, just use the ObjectId as the Primary Key

If Select with ParentKey or IN Clauses, construct the Secondary Key(s)

Forms an in-Cache "Shadow Graph" of Data comprised of ObjectIds, relationships between them and original Queries

Common Components between Requests can be served from Cache

If |RQ1| >= |RQ2|, serve Data from Cache (consider Variables too)

Use Cache-Control to dynamically control Cache behavior