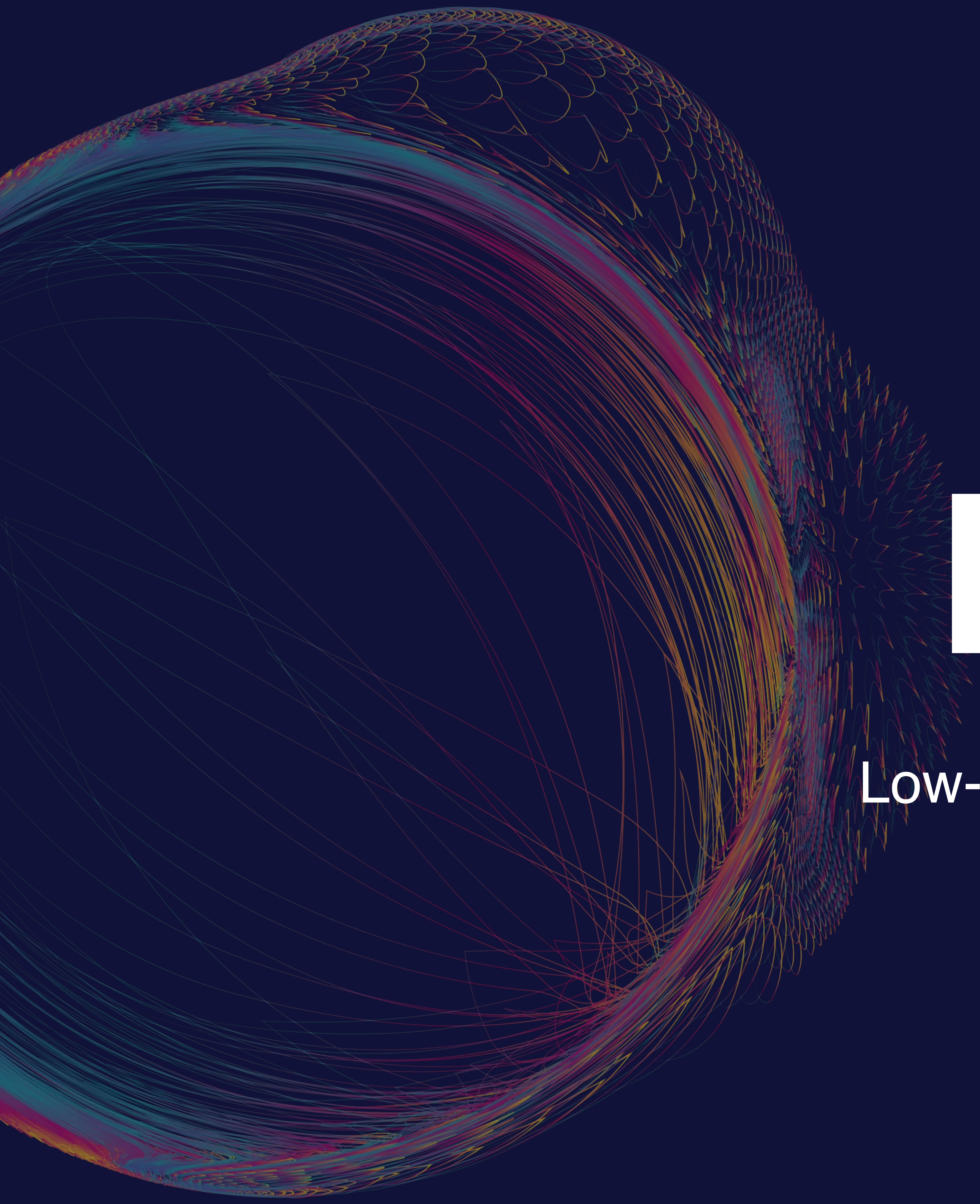


Hypi.

Low-code, GraphQL, Serverless Platform

2019

IMCS June 2019

Courtney Robinson

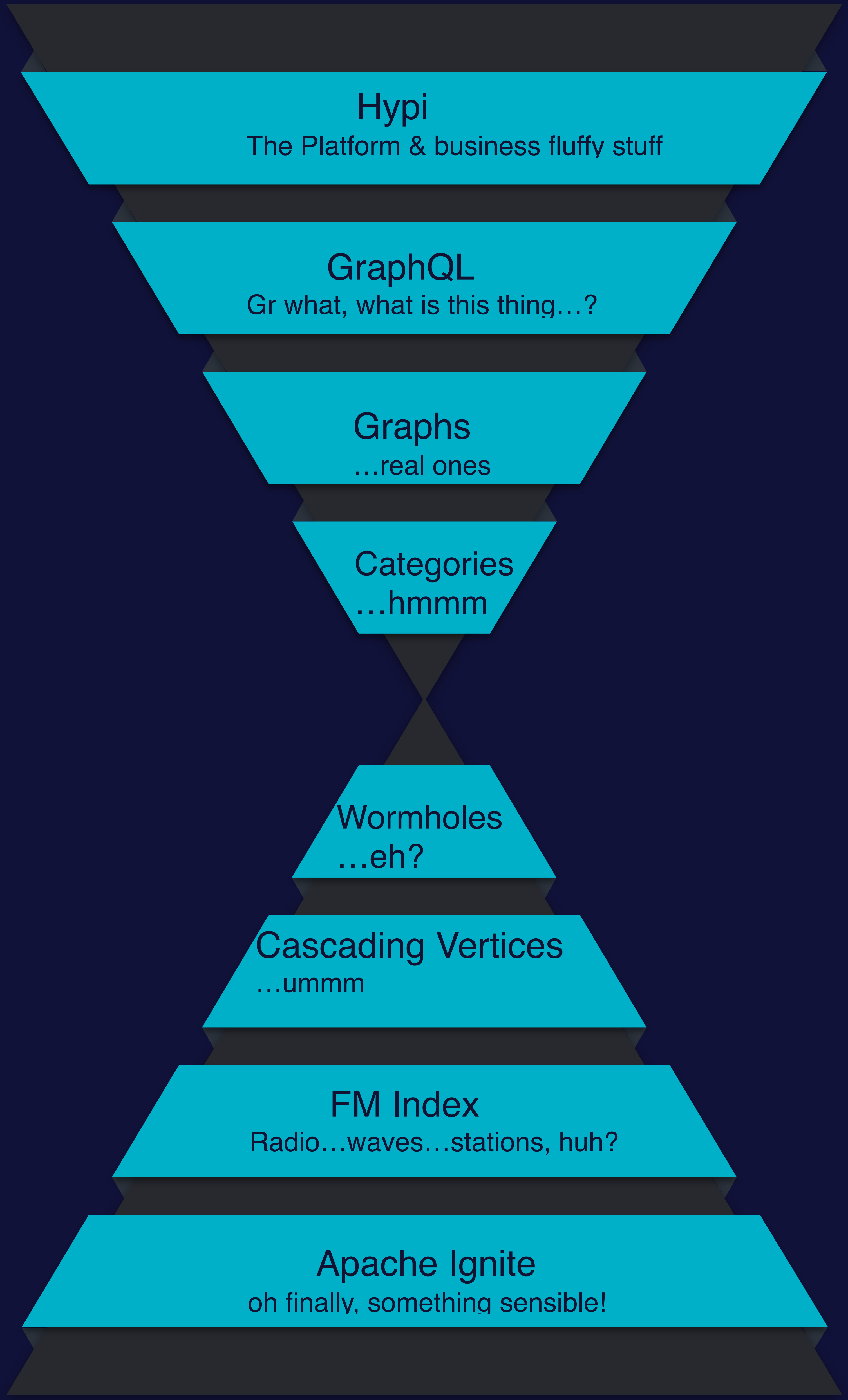
Founder & CEO of Hypi; Jack of all trades and worse PhD student ever…so let’s skip the hard questions

Hypi.

The Descent

We’ll start out easy and work our way down.

…and hopefully back out again



Hypi
The Platform & business fluffy stuff

GraphQL
Gr what, what is this thing…?

Graphs
…real ones

Categories
…hmmm

Wormholes
…eh?

Cascading Vertices
…ummm

FM Index
Radio…waves…stations, huh?

Apache Ignite
oh finally, something sensible!

Hypi.

The Core Team

Rochelle Singh	Courtney Robinson	Damion Robinson	Jennicka Buckingham	Pawel Ungier
CTO	CEO	HEAD OF PRODUCT	HEAD OF BRAND	HEAD OF SALES

Hypi.

A Little Bit About Hypi

One API, any platform

Hypi takes data model and in seconds turn it into a highly available, distributed, serverless backend API.

Takes project development down to a fraction of the time.

Includes serverless functions with built-in storage and Identity and Access Management (UMA ish).

Hypi Hyper Cloud enables development against a single API to integrate with any public or private cloud.

Hypi.

What is it?

- Serveless Functions
- On Demand Service Provisioning
- Service & Resource sharing
- Low code, no code Applications

In short, Hypi gives all the benefits of grid computing but reduces the complexity & cost of running the “conventional” way.

Hypi.

What does that mean?

- Hypi. has storage
- It has compute
- It has authorisation
- It is scalable (just add more nodes)
- It is extensible

The Platform

Hypi is a declarative platform.

It lets you declare a desired end state and Hypi figures out how to get to that state.

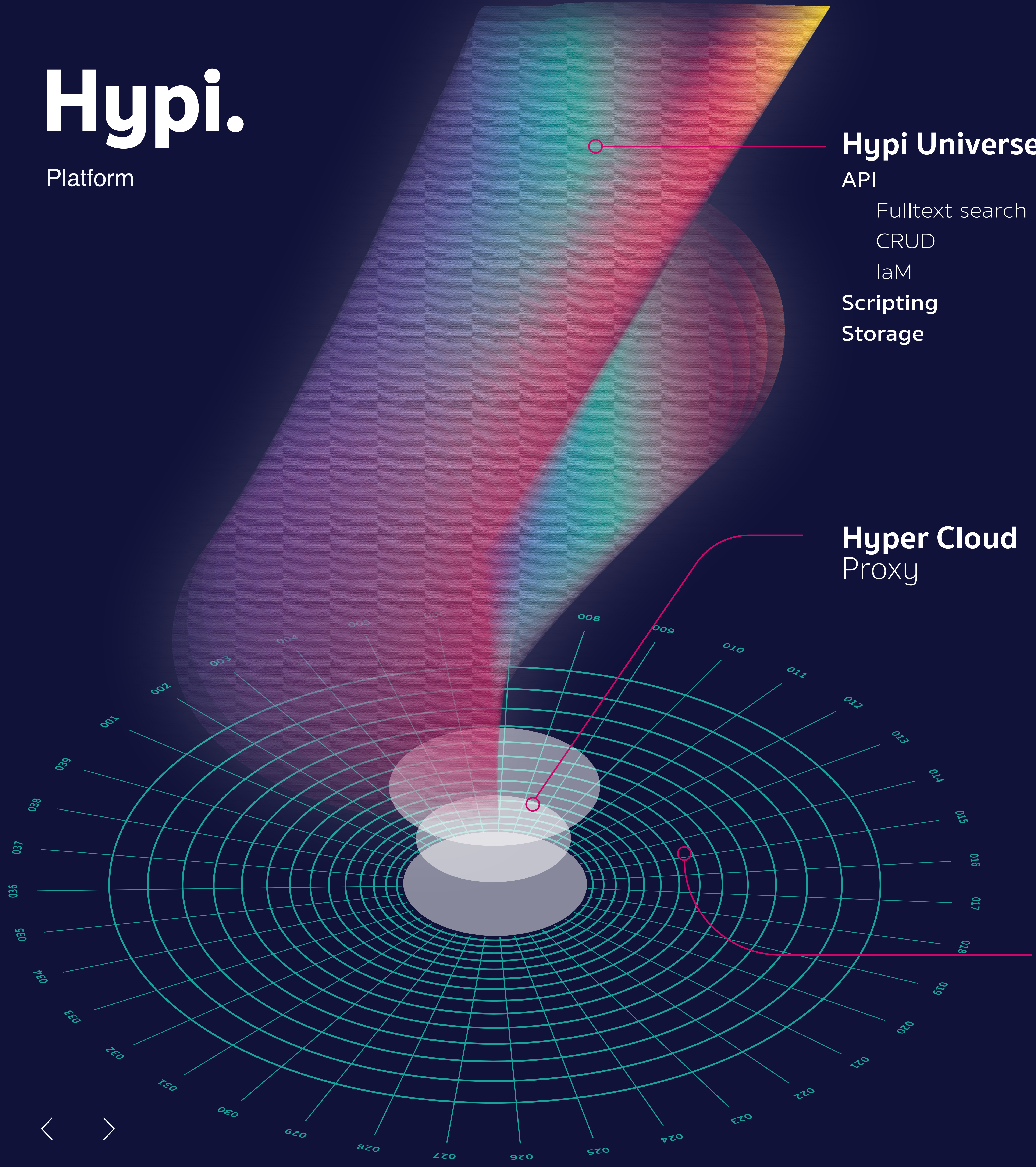
Hypi Universe has a core set of features baked into the Hypi services.

Hyper Cloud builds our Delta Grid enabling automatic integration with services (Hypi provided or custom integrations).

This lean combination drastically reduces development time, if a project’s model and UI can be prototyped in a day, the platform lets you ship it in a day!

Hypi.

Platform



Hypi Universe

API

- Fulltext search
- CRUD
- IaM

Scripting

Storage

Hyper Cloud

Proxy

Delta Grid

Hypi Universe

Api

Auto generated from a GraphQL model, one consistent API for core and multi-cloud services

Fulltext search

allows data to be "Indexed" so that it can be searched against

Scripting

Allows submission of JavaScript, entire Java classes, single Java functions or single Java expressions that can be executed before or after CRUD functions or associated with custom GraphQL functions

CRUD

Create, Read, Update and Delete (+ trash) APIs

IaM

Identity and Access management todefine organisation structure, groups, policies and permissions

Storage

Simple APIs to upload files of any kind that can be downloaded or otherwise used later.

Hyper Cloud

Proxy

Allows the definition of application secrets/credentials that are needed to access 3rd party APIs. The third party APIs together form the Hypi Delta Grid

Delta Grid

Machine Learning

OCR Entity Extraction - Allows extraction and identification of contents from images

Facial Recognition - Facial verification, identification, age detection, gender and emotions.

General (Ignite/Tensorflow) - Custom machine learning based on Tensorflow. Preprocessing, Partition Based Dataset, Linear Regression, K-Means Clustering, Genetic Algorithms, Multilayer perceptron, Decision Trees, k-NN Classification, k-NN Regression, SVM Binary Classification, SVM Multi-class Classification.

Video processing

Per 1K mins stored/viewed (Cloudflare) - billed per 1K minutes stored and viewed

Per GB stored/transferred - billed per GB stored/transferred

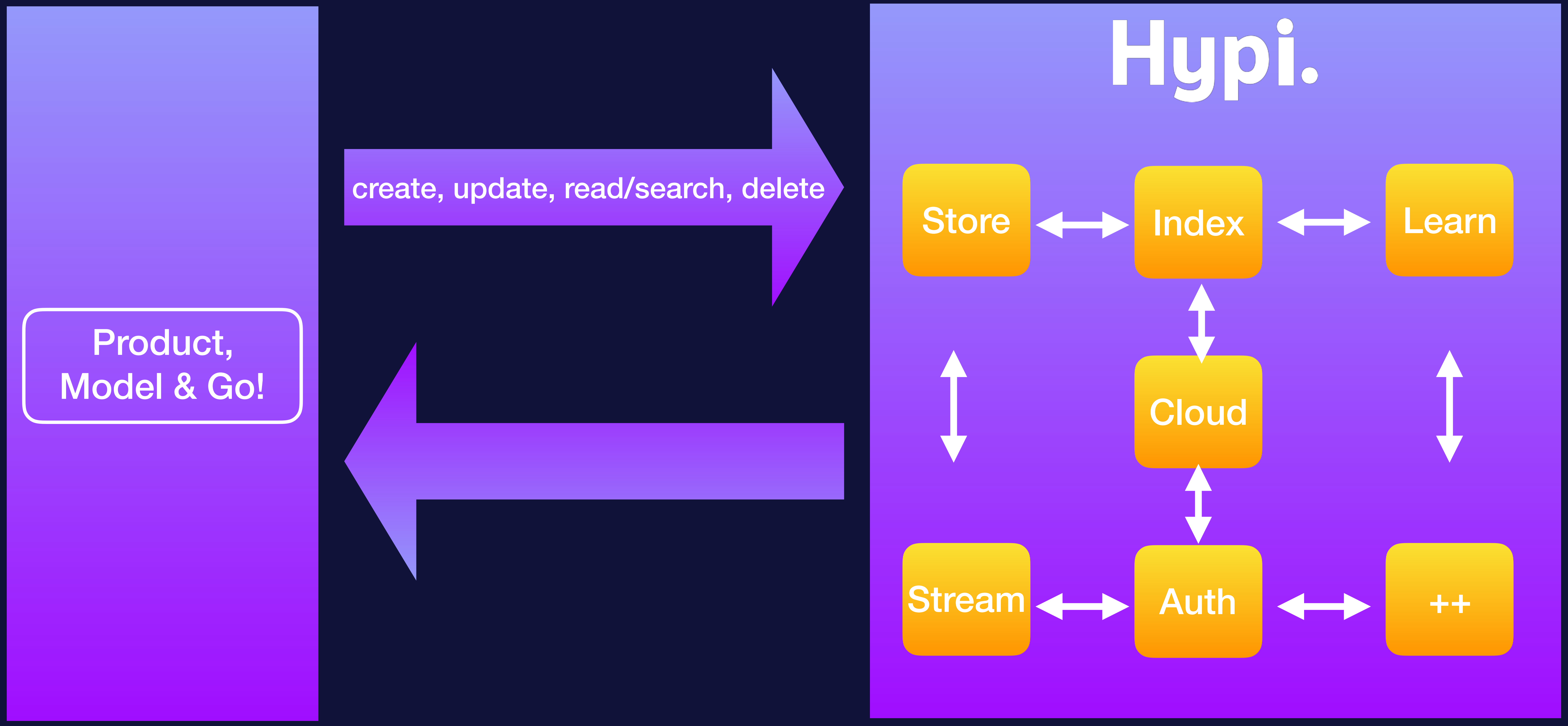
Payment Processing

Allows apps to collect credit/debit card payements

Stripe
SIBS
PayPal
Braintree
Square

Hypi.

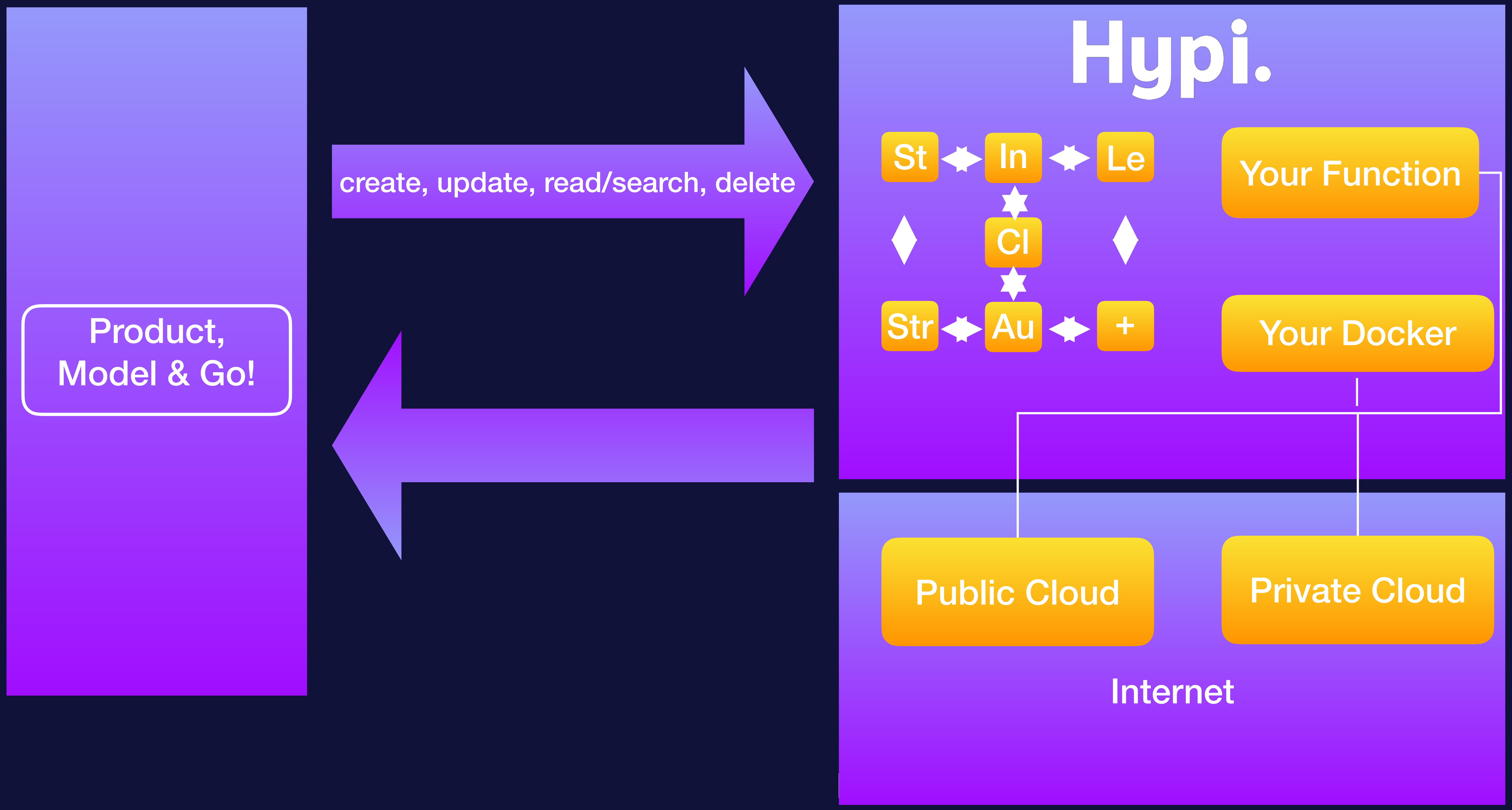
For any Hypi Application



- ✓Storage
- ✓Compute
- ✓Authorisation
- Extensible

Hypi.

Extensible



✓Storage

✓Compute

✓Authorisation

✓Extensible

Enough of that, on to the reason we’re all here…the how… how do we do it?

Magic!

Joking

...probably

GraphQL

- Declarative, type based framework, language, standard…may be easier to say what it isn’t
- Expressive, any model that can be expressed through an OOP object model can be expressed with GraphQL
- Succinct, one of the points FB sells it on. Useful in low/expensive bandwidth situations
- Flexible, use directives to add features/semantics
- Growing adoption, can hardly be dismissed as a fad anymore

Let’s build a todo app

Possible features:

1. Create todo item
2. Complete todo item
3. Add comments to todo items
4. Search for todo items
5.Paginate through todo items
6. Trash todo items
7. Add attachments to todo items
8. Create groups of todo items
9. Share individual todo items
10. Share groups
11.Delete todo items
12.Delete groups

For this talk we will focus on

1. Create todo item
2. Complete todo item
3. Add comments to todo items
4. Search for todo items

Hypi.

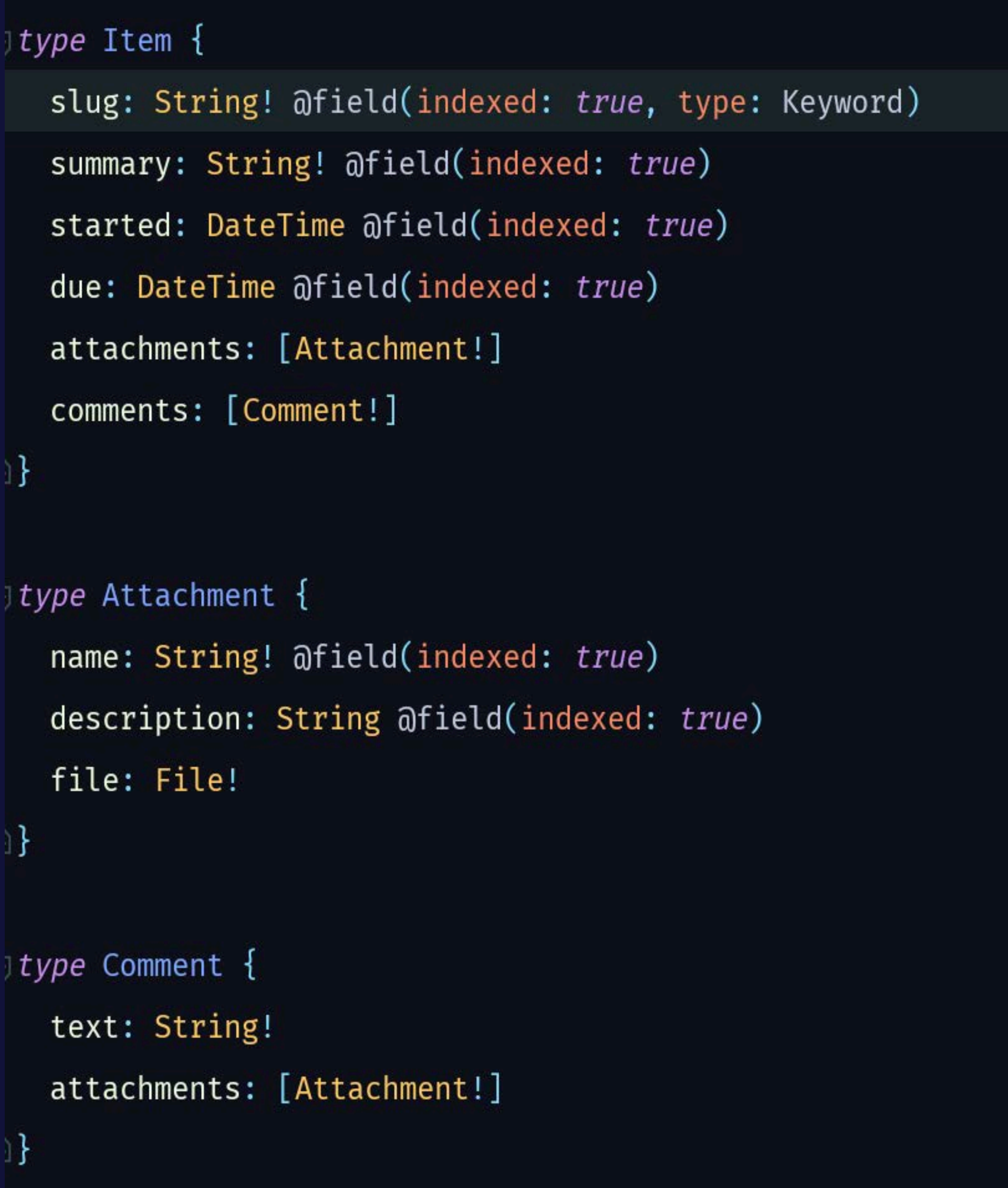
What does it look like?

For this talk we will focus on

1. Create todo item
2. Complete todo item
3. Add comments to todo items
4. Search for todo items

...I lied a little
From this model, you can already do all of these

1. Paginate through todo items
2. Trash todo items
3. Add attachments to todo items
4. Create groups of todo items
5. Share individual todo items
6. Share groups
7. Delete todo items
8. Delete groups



```
type Item {
  slug: String! @field(indexed: true, type: Keyword)
  summary: String! @field(indexed: true)
  started: DateTime @field(indexed: true)
  due: DateTime @field(indexed: true)
  attachments: [Attachment!]
  comments: [Comment!]
}

type Attachment {
  name: String! @field(indexed: true)
  description: String @field(indexed: true)
  file: File!
}

type Comment {
  text: String!
  attachments: [Attachment!]
}
```

What did you see?

Hypi saw relations
Relations means graph

…Graph means categories, categories means graph, graph means categories, categories…well, you get the idea

Only a few slides in and we’re already in recursive hell

Let’s get real

Graphs in review

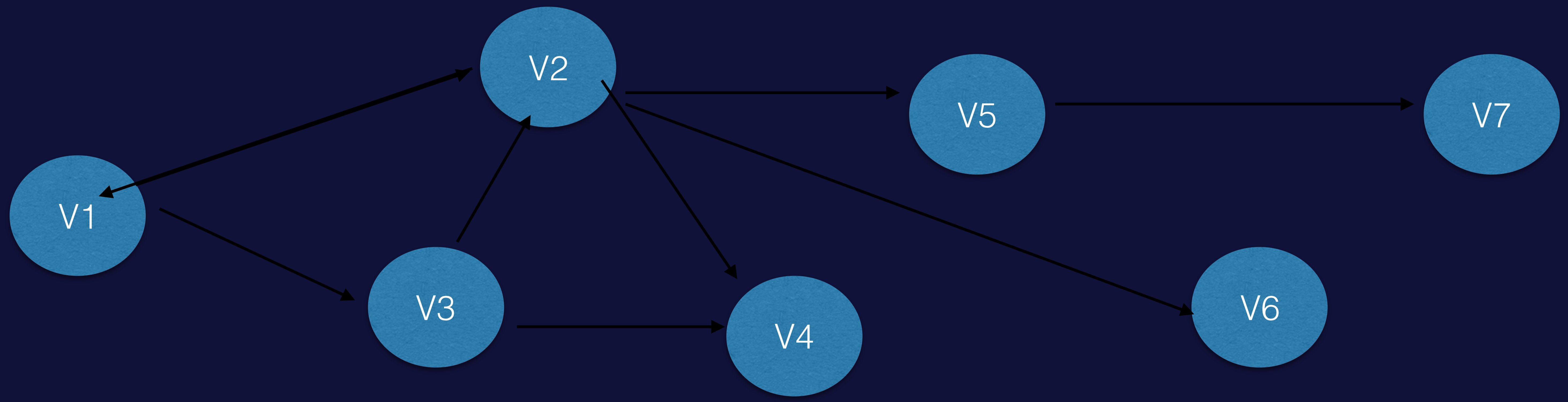
A **graph** G is made up of a set of vertices and edges,

G = (V,E)

A **Vertex** is a single datum within a graph.

An **edge** connects two vertices.

A **property** is a key-value pair on an edge or vertex.



Distributed systems

CAP theorem anyone?

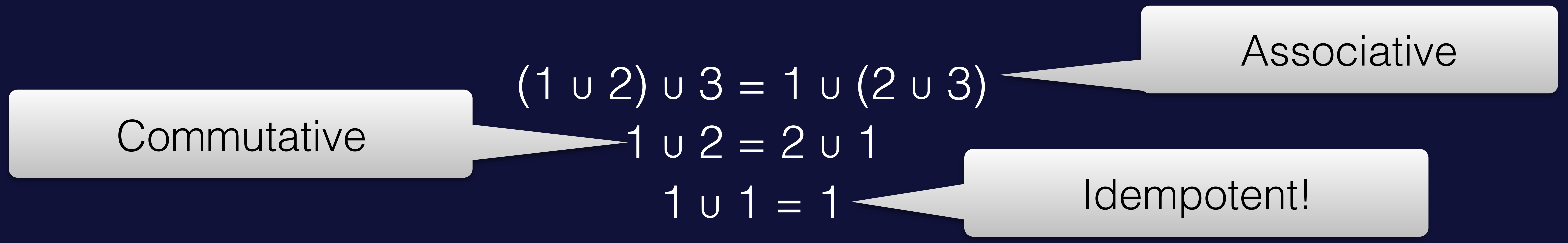
Consistency, Availability & Partition tolerance…choose two?

It’s a hard life, so we choose…discipline.

Draw upon some set theory to take advantage of a winning combination.

1. Commutativity
2. Idempotence
3. Associativity

For more checkout CRDTs, in particular, how join-semi lattice is used



Associative

$(1 \cup 2) \cup 3 = 1 \cup (2 \cup 3)$

Commutative

$1 \cup 2 = 2 \cup 1$

$1 \cup 1 = 1$

Idempotent!

Bare in mind for later

$$\{a,b,c,d\} \;:\Leftrightarrow \{a,b\} \cup \{c,d\}$$

Category Theory

at least the bit I didn't get bored of anyway…

- Think of a category as a collection of objects with arrows between them with the 3 properties

   1. Composition
   2. Identity
   3. Associativity

Wait…didn't you just call those something else?

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g\circ f"'] \arrow[loop left, "1_A"] & B \arrow[d, "g"] \arrow[loop right, "1_B"] \\
& C \arrow[loop right, "1_C"]
\end{tikzcd}$$

Basic category theory becomes the basis for describing distributed graph computations.

Interesting because things that hold true in category theory generally holds true when graph computing is reasoned about with it.

Put it all together

and you get…

Distributed Graph Computing

…he claims

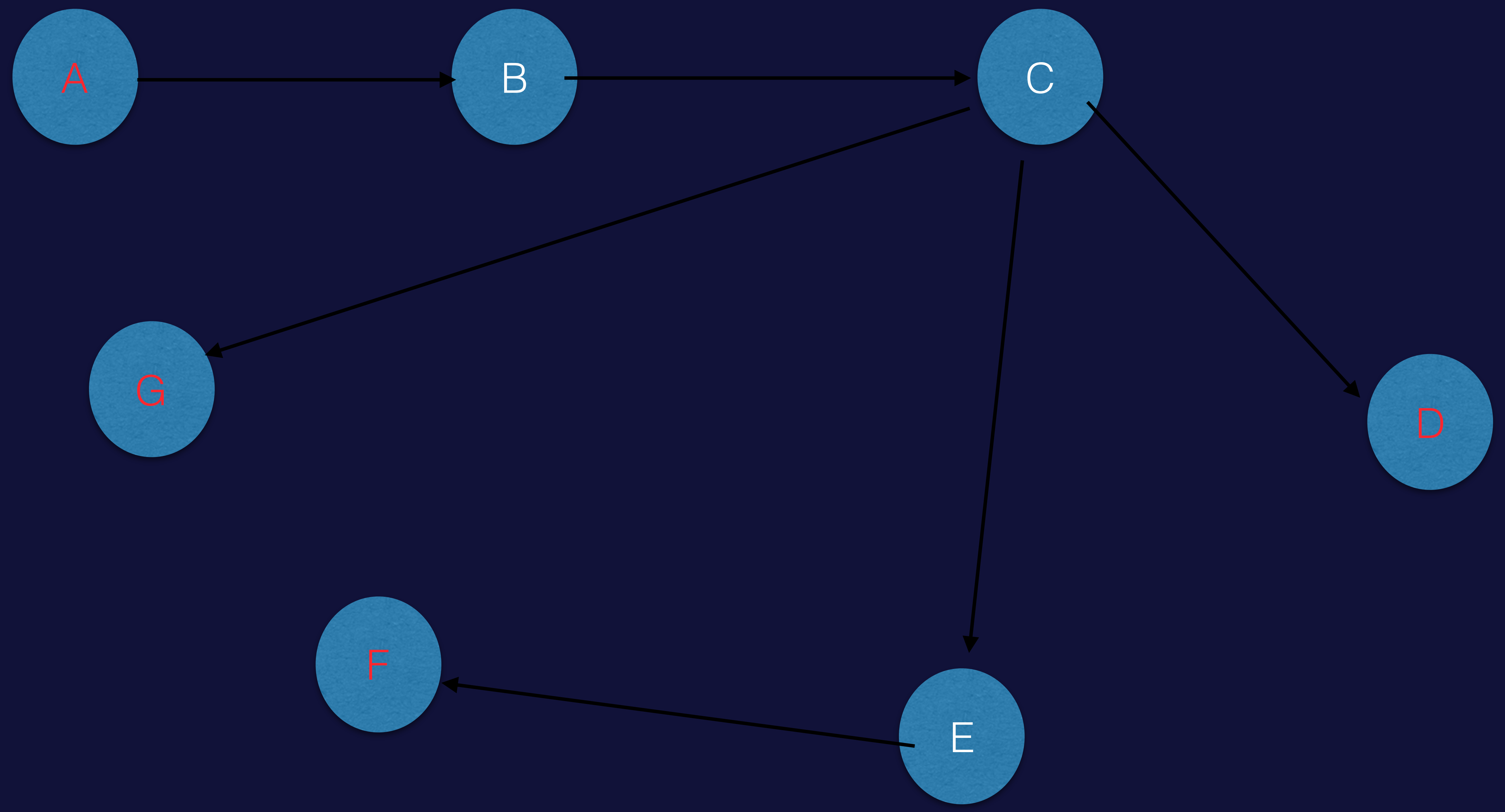
Hypi.

Remember this?
Look at G of F, it more or less says the same thing

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g\circ f"'] \arrow[loop left, "1_A"] & B \arrow[d, "g"] \arrow[loop right, "1_B"] \\
 & C \arrow[loop right, "1_C"]
\end{tikzcd}$$

Wormhole traversals

brought to you by CR…get it?



Graphs can get pretty big. Big enough not to fit one a single machine.

Imagine red letters are on different drives or machines.

Imagine the graph was immutable…

At its simplest, wormhole traversals enables jumping from A to G or any other of the vertices in red.

The cost?

1. ~7% disk overhead for 20 - 35% speedup.
2. ~5 - 15% configurable memory overhead for an additional 13-27% speedup.

Hypi.

Cascading vertices

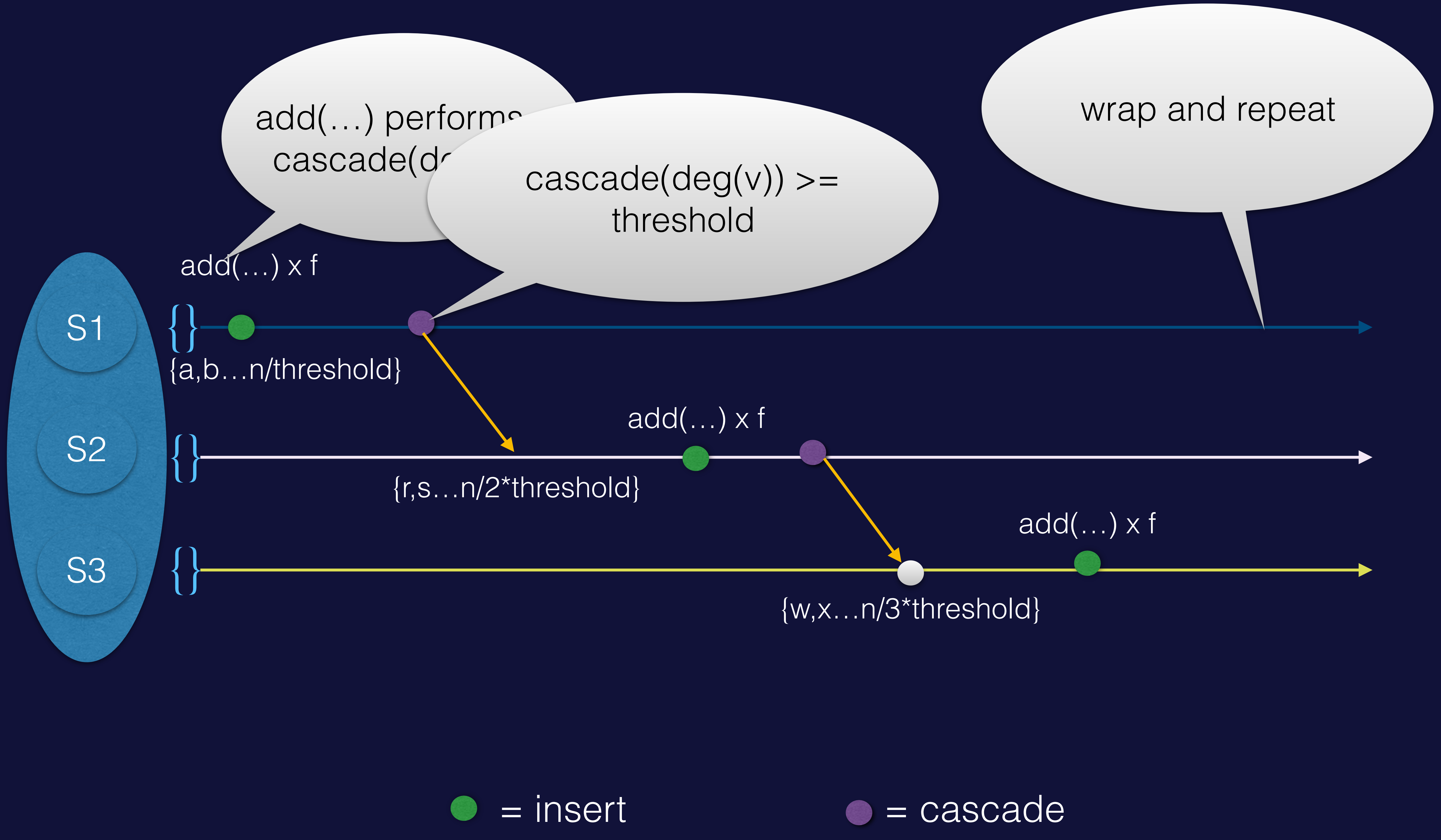
Power to the vertex!

Graphs can get pretty big…I said that already…

Vertices can get pretty big, big enough not to fit on a single machine.

Promise I’m not just repeating myself…the graph is.

- “Cascading vertices” is a technique for partitioning
  - Addresses the power law distribution
  - The edges of a vertex cascade over multiple servers
- Twitter followers as an example e.g. Obama, massive vertex
- Simple threshold base cascading
  - Impl. based on vertex degree
  - Experimenting with ML base placements



Remember this?

$$\{a,b,c,d\} \;:\Leftrightarrow\; \{a,b\} \cup \{c,d\}$$

That is to say, some arbitrary set S if split into n parts can be unioned to obtain the equivalent original set

If it matters to you, the important thing is isomorphism i.e. structural equivalence. It matters both here and in wormhole traversals.

FM Indexes

or how we hang this all together

Succinct data structure i.e. space "close" to the information-theoretic lower bound

Hypi version combines

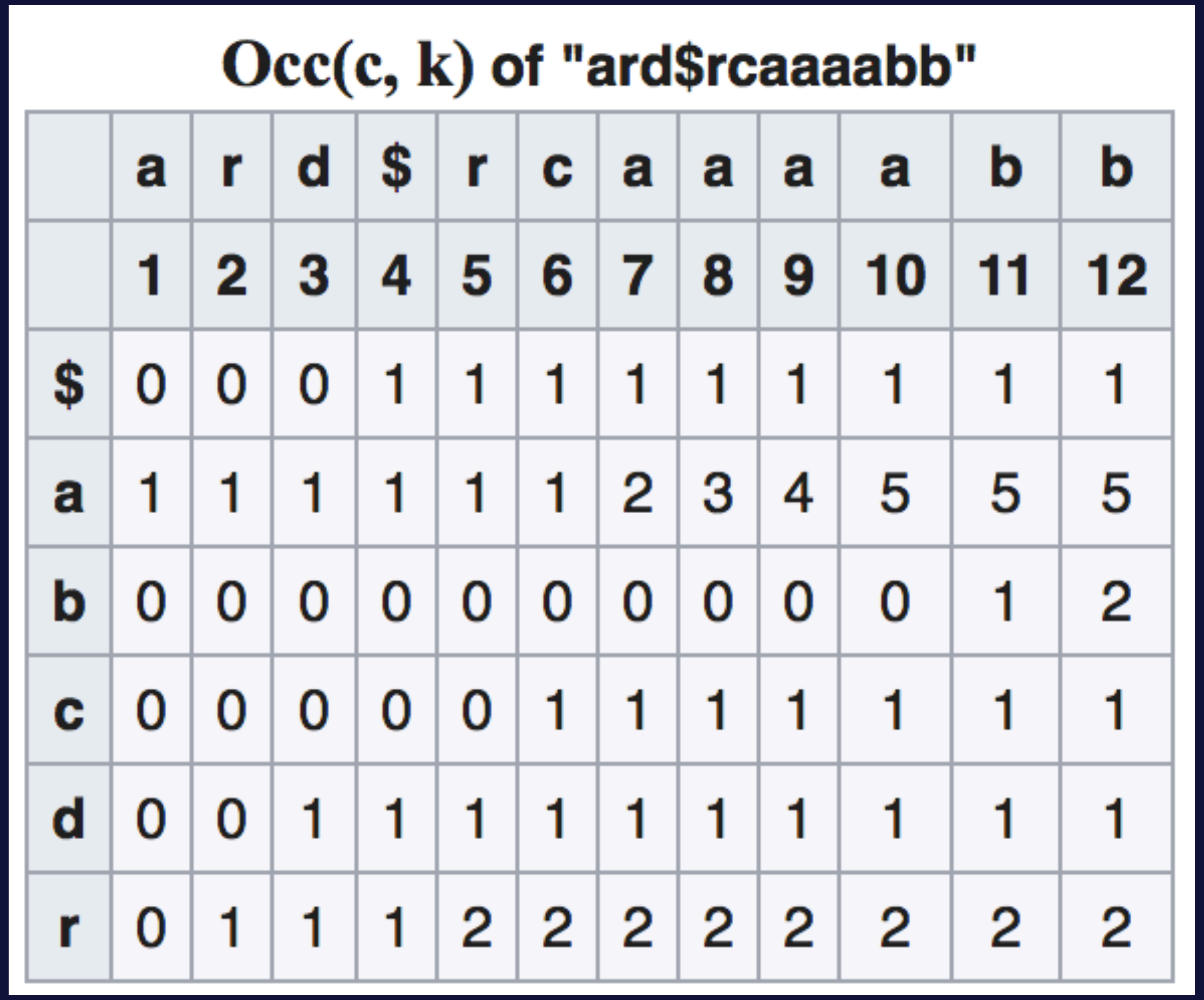
1. Radix Trie
2. Burrows-Wheeler transform
3. Huffman encoding

As a basis for a new in memory encoding.

No need to deserialise compressed/encoded data to use

Still get prefix traversals i.e. given this vertex, find all connected vertices
In addition, enables O(k) reply to "are these two edges connected" where k is length of input (UUID in our case)

From Wikipedia



Occ(c, k) of "ard$rcaaaabb"

	a	r	d	$	r	c	a	a	a	a	b	b
	1	2	3	4	5	6	7	8	9	10	11	12
$	0	0	0	1	1	1	1	1	1	1	1	1
a	1	1	1	1	1	1	2	3	4	5	5	5
b	0	0	0	0	0	0	0	0	0	0	1	2
c	0	0	0	0	0	1	1	1	1	1	1	1
d	0	0	1	1	1	1	1	1	1	1	1	1
r	0	1	1	1	2	2	2	2	2	2	2	2

Hypi.

Ignite, bringing it all together

whoohoo, we’re back!

Hypi implemented using KV APIs for caches instead of SQL APIs.

Recent project with:

1.2+ billion vertices, 7+ billion edges

10ms 99 percentile query time

only 15 servers, 500GB RAM and nearly 3TB disk usage.

Graphs

Implicit through the GraphQL model

Wormholes

An optimisation that allows you to skip vertices during traversal

Cascading Vertices

Partitioning of super-vertices

FM Index

It's like a BloomFilter for Graphs...kinda

Ignite: How we hook in

- Affinity runs
  - use Lucene for indexing
  - FM index for relationships, falling back to Lucene
- Ignite’s affinity keys are used to implement vertex cascading
  - We get relatively slow writes (sometimes read before write)

Some key points

- Every GraphQL type results in one Ignite cache
- Each Ignite cache has one lucene index and one RocksDB database
- Each Ignite cache is shared if two tenants have the same GraphQL type name
  - Dedicated tenant caches are planned for Q4 2019
- Each RocksDB database is also shared
  - Each tenant gets a RocksDB Column Family
- Relationship references are stored in the target Lucene index
- FMIndex partially rebuilt from disk references on startup then rest is populated on demand

Instant CRUD API

```
type Item {
slug: String
summary: String!
comments: [Comment!]
}
```

→

```
findItem(arcql: String): [Item!]
createItem(values: [Item!]!): [Item!]
updateItem(values: [Item!]!): [Item!]
deleteItem(arcql: String!): [Item!]
trashItem(arcql: String!): [Item!]
```

Arc Query Language i.e. Arc QL

Simple, intuitive, familiar!

<query> <sort> <from> <limit>

FROM ‘<pagination-cursor>’

SORT fieldName ASC I DESC

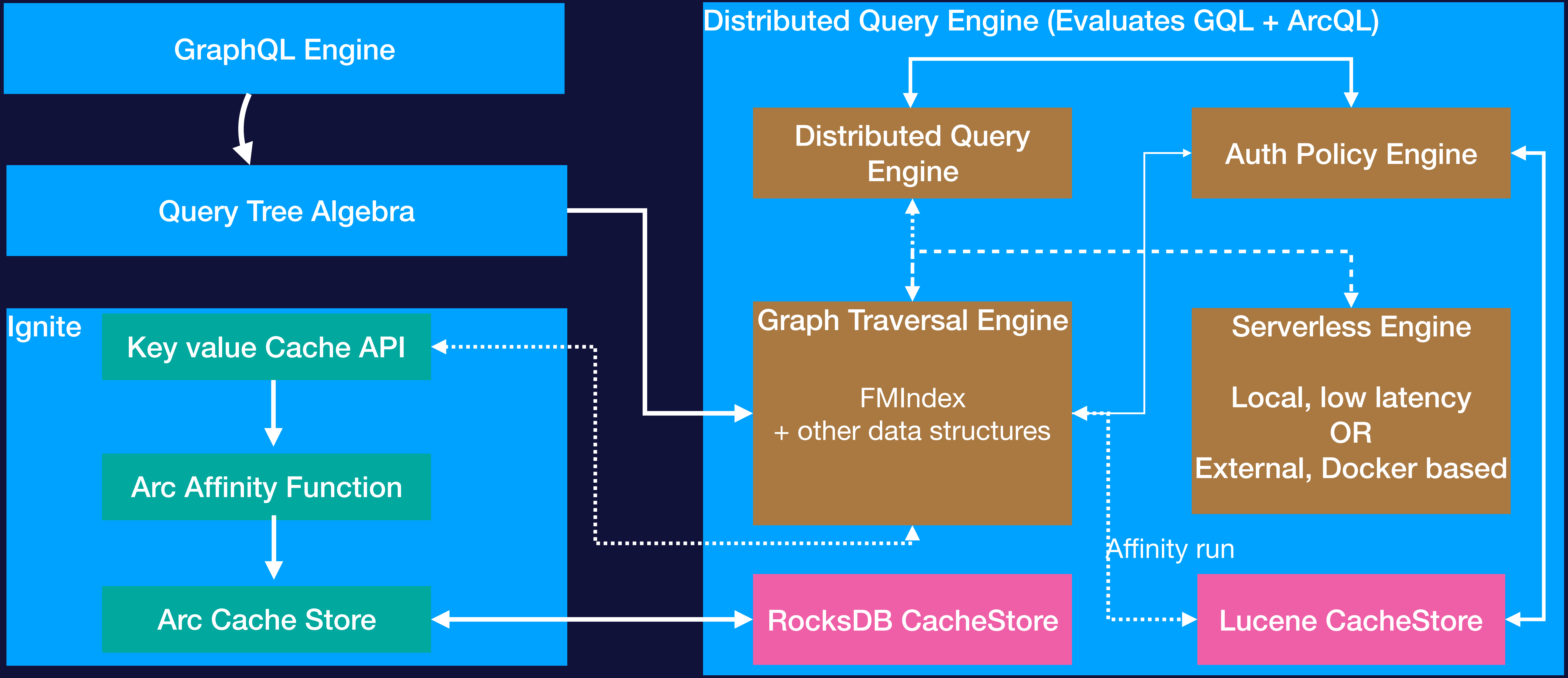
LIMIT <N>

Query types

- **Term** - fieldName = ‘value’
- **Phrase** - fieldName ~ ‘some value’
- **Prefix** - fieldName ^ ‘music’
- **Wildcard** - fieldName * ‘mu?ic*’
- **Fuzzy** - ~fieldName~ ‘name’
- **Range** - fieldName IN [0, 100)
- **Match all** - *
- EXIST
- NOT EXIST
- INNER JOIN (implicit e.g. a.b.c = 'xyz'
- LEFT JOIN
- REFS FROM...WHERE (optional)
- link
- unlink
- subscribe (for realtime updates on IDs and near real time on queries)

Hypi.

Arc OS - Platform Architecture



GraphQL Engine

Query Tree Algebra

Ignite

Key value Cache API

Arc Affinity Function

Arc Cache Store

Distributed Query Engine (Evaluates GQL + ArcQL)

Distributed Query Engine

Auth Policy Engine

Graph Traversal Engine

FMIndex
+ other data structures

Serverless Engine

Local, low latency
OR
External, Docker based

Affinity run

RocksDB CacheStore

Lucene CacheStore

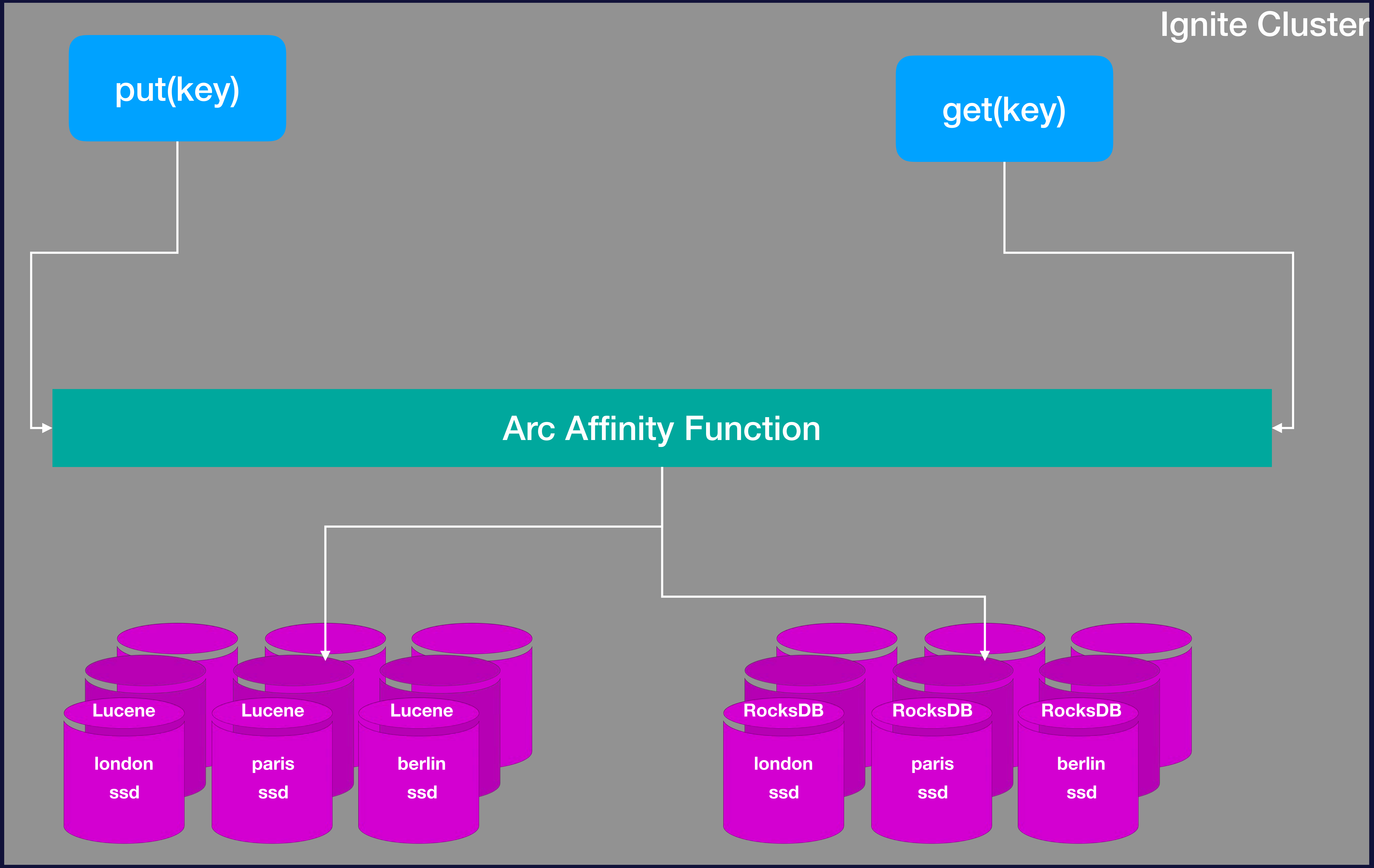
Hypi.

Arc OS - Affinity Function & Query Routing

Query & Data Routing

- f : key => partition
- Rendezvous hashing based on
  - Type of key
  - Node requirements
  - Cache name

- Double query required to filter
- Average <5ms to do both



‹ ›

Thank you

Hypi cloud service will be in public beta June 2019.
courtney.robinson@hypi.io for an invite, 3 months free use.

There was a lot glossed over here…any questions?