BUILDING INTERACTIVE APPLICATIONS AT SCALE

DRUID · FACETJS

FANGJIN YANG · VADIM OGEIVETSKY

SOFTWARE ENGINEERS @ METAMARKETS

METAMARKETS

OVERVIEW

PROBLEM	DEALING WITH EVENT DATA
EXPLORATORY ANALYTICS	THE STORY OF DRUID
INTERACTIVE VISUALS	THE STORY OF FACETJS
NEXT STEPS	TRY IT OUT FOR YOURSELF

THE PROBLEM

THE PROBLEM

- Interactive visualizations for exploratory analytics
- Low latency queries and data ingestion
- Scalable: 500k+ events/sec, 50PB+ raw data, ~150 queries/second
- These problems exist in many industries
  - Online advertising
  - System/application metrics
  - Network traffic monitoring
  - Activity stream analysis
  - Finance

DEMO

IN CASE THE INTERNET DIDN'T WORK
PRETEND YOU SAW SOMETHING COOL

THE DATA

- Transactional/event data
- Immutable
- (Mostly) append only
- OLAP

THE QUERIES

- Business intelligence queries
- Roll-up, drill down, slice and dice, pivot
- Examples
  - Revenue over time broken down by demographic
  - Top publishers by clicks over the last month
  - Number of unique visitors broken down by any dimension
- Aggregating a set of metrics for a filtered view of a data set

THE DATABASE

- Relational databases
- Key/value stores
- Other commercial companies

RDBMS

- Common solution in data warehousing
- Many open source and commercial solutions
- Row stores
- Results
  - Scan speed: 5.5M rows/sec/core
  - 1 query over 1 week of data: 5 seconds
  - 20 queries over 1 week of data: minutes

KEY/VALUE STORES

- Pre-computation

ts	gender	age	revenue
1	M	18	$0.15
1	F	25	$1.03
1	F	18	$0.01

$\longrightarrow$

Key	Value
1	revenue=$1.19
1,M	revenue=$0.15
1,F	revenue=$1.04
1,18	revenue=$0.16
1,25	revenue=$1.03
1,M,18	revenue=$0.15
1,F,18	revenue=$0.01
1,F,25	revenue=$1.03

KEY/VALUE STORES

- Results
  - Queries are fast (lookups into maps)
  - Inflexible (not pre-computed, not available)
  - Data ingestion is slow
  - Pre-computation time is slow!
    - Limit total set of queries on ~500k events
    - With 11 dimensions: 4.5 hours on a 15-node Hadoop cluster
    - With 14 dimensions: 9 hours on a 25 node Hadoop cluster

DRUID

- Open sourced in Oct. 2012
- Growing Community
  - 52+ contributors from many different organizations
  - Many production deployments at large technology companies
- Designed for low latency ingestion and aggregation
  - Optimized to power dashboards and answer BI queries
- License: Apache 2.0, working on community governance

DRUID

- Inspired by search architecture
- Combine computation and storage
- Create immutable data structures that are highly optimized for fast aggregates and filters

DRUID - BUZZWORDS

- Distributed, column oriented, shared nothing architecture
- HA, no single point of failure
- Low latency data ingestion and exploration
- Approximate and exact calculations
- Integrates with Kafka, Samza, Storm, and Hadoop

RAW DATA

```
timestamp             publisher          advertiser  gender  country click  price
2011-01-01T01:01:35Z  bieberfever.com    google.com  Male    USA     0      0.65
2011-01-01T01:03:63Z  bieberfever.com    google.com  Male    USA     0      0.62
2011-01-01T01:04:51Z  bieberfever.com    google.com  Male    USA     1      0.45
...
2011-01-01T01:00:00Z  ultratrimfast.com  google.com  Female  UK      0      0.87
2011-01-01T02:00:00Z  ultratrimfast.com  google.com  Female  UK      0      0.99
2011-01-01T02:00:00Z  ultratrimfast.com  google.com  Female  UK      1      1.53
```

PARTITION DATA

```
timestamp             page             language  city     country  ...  added  deleted

2011-01-01T00:01:35Z  Justin Bieber    en        SF       USA           10     65
2011-01-01T00:03:63Z  Justin Bieber    en        SF       USA           15     62
2011-01-01T00:04:51Z  Justin Bieber    en        SF       USA           32     45
                   Segment 2011-01-01T00/2011-01-01T01

2011-01-01T01:00:00Z  Ke$ha            en        Calgary  CA            17     87
                   Segment 2011-01-01T01/2011-01-01T02

2011-01-01T02:00:00Z  Ke$ha            en        Calgary  CA            43     99
2011-01-01T02:00:00Z  Ke$ha            en        Calgary  CA            12     53
                   Segment 2011-01-01T02/2011-01-01T03
```

- Shard data by time
- Immutable chunks of data called “segments”

IMMUTABLE SEGMENTS

- Data stored in column orientation
- Read consistency
- One thread scans one segment
- Multiple threads can access same underlying data

INDEXES

- Builds search indexes (inverted indexes/bitmap indexes and not B-trees)
- Scan/load exactly what you need for a query

DRUID GAVE US

- Fast queries
- Arbitrarily data exploration
- Immediate insight into data
- Scalability

REMAINING PROBLEMS

- Druid's query language is JSON over HTTP
- Query language fairly low level
- Each query is designed to run very quickly
- Complex operations may require many queries
- Building meaningful visualizations can be a complex operation

FACET.JS

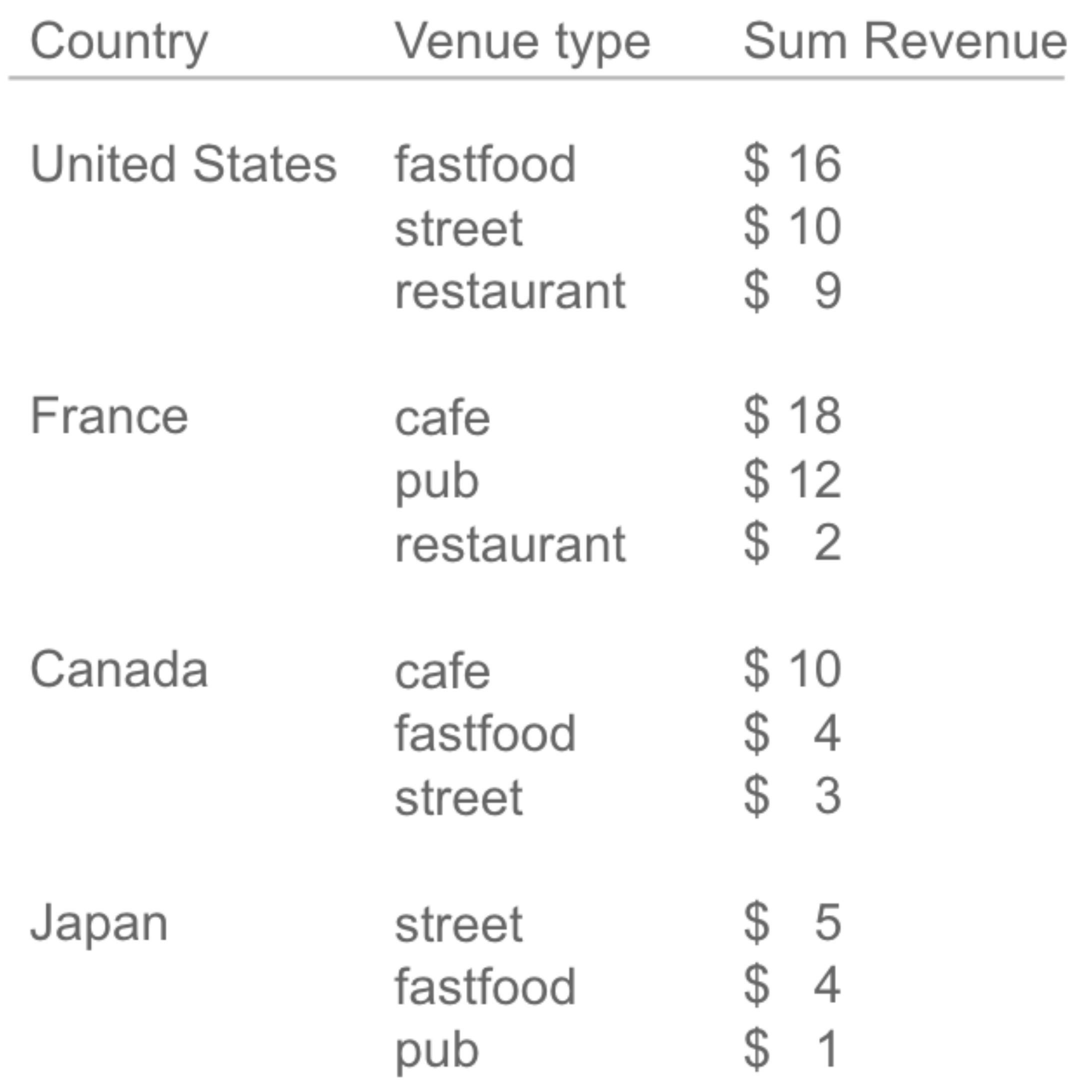
THE PROBLEM

- Datastores are designed to answer specific queries, not drive visualizations
- Lack of high-level operations needed for certain visualizations
- No good way of writing UI unit tests
- Druid specific:
  - Druid API is structured around Druid internal architecture
  - Low level queries

THE PROBLEM

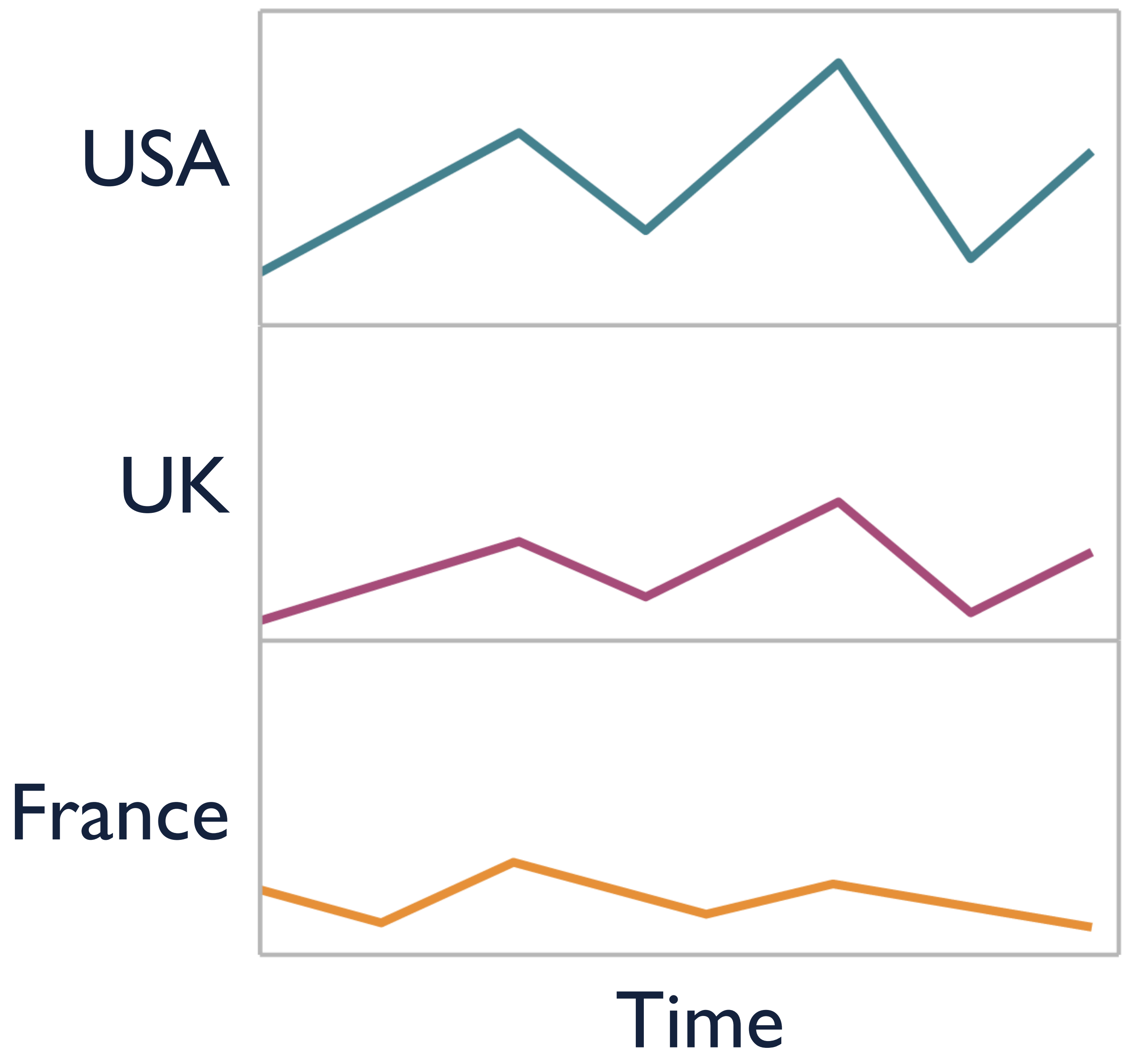
For the top four countries (by revenue), what are the top three venues?

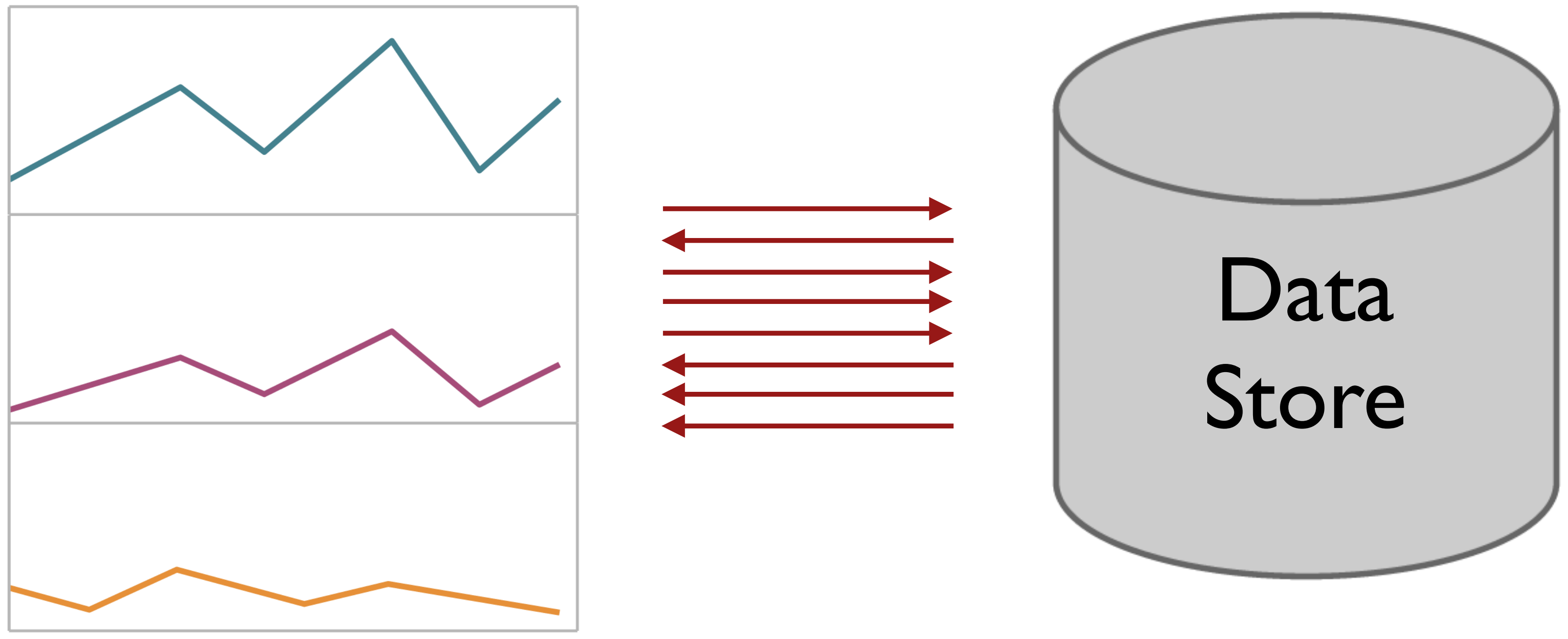
You can not answer this question with a single query.



Country	Venue type	Sum Revenue
United States	fastfood	$ 16
	street	$ 10
	restaurant	$ 9
France	cafe	$ 18
	pub	$ 12
	restaurant	$ 2
Canada	cafe	$ 10
	fastfood	$ 4
	street	$ 3
Japan	street	$ 5
	fastfood	$ 4
	pub	$ 1

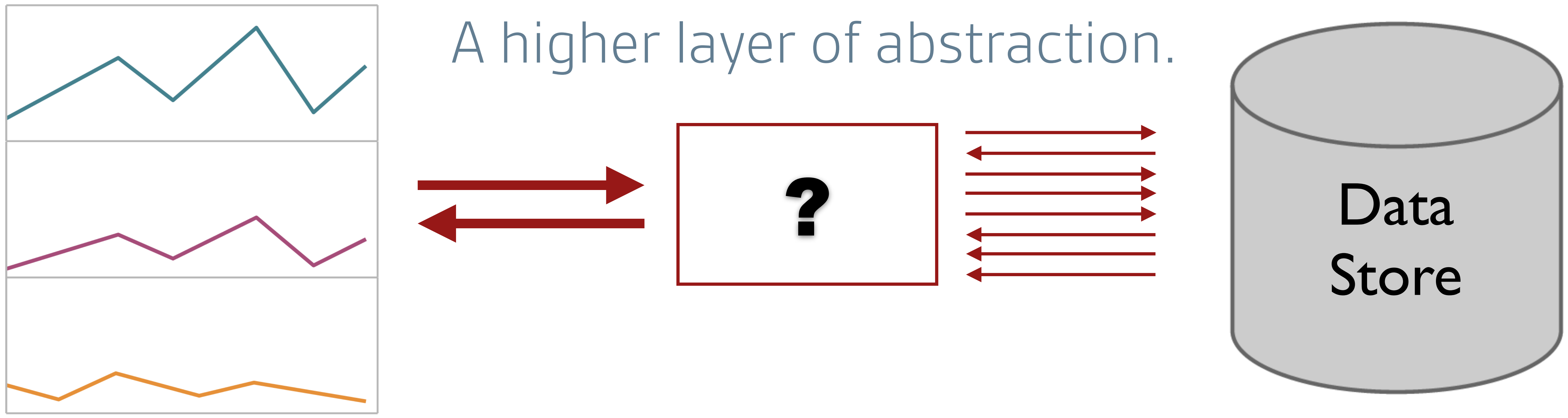
THE PROBLEM




Data
Store

WHAT IS NEEDED?

A higher layer of abstraction.

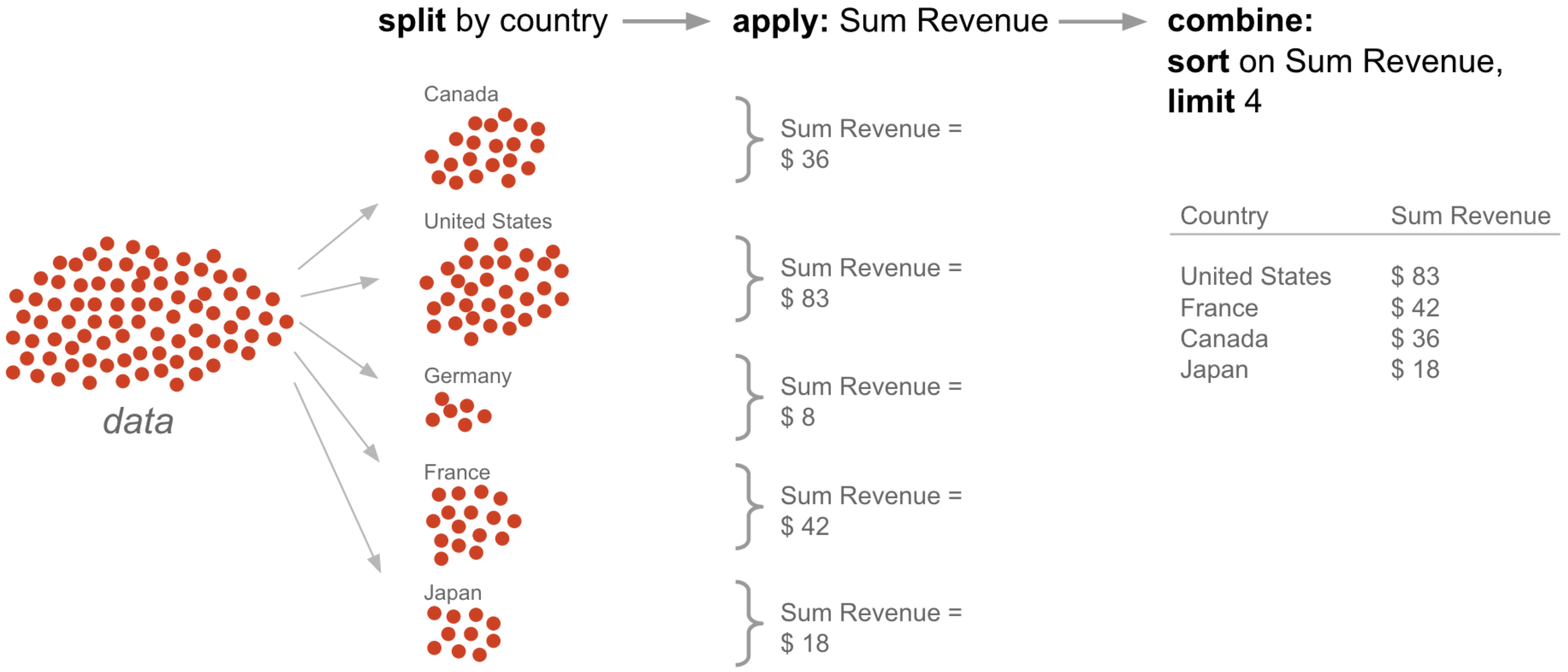


SPLIT-APPLY-COMBINE

Hadley Wickham popularized a concept called
split-apply-combine
as a way of thinking about data querying.

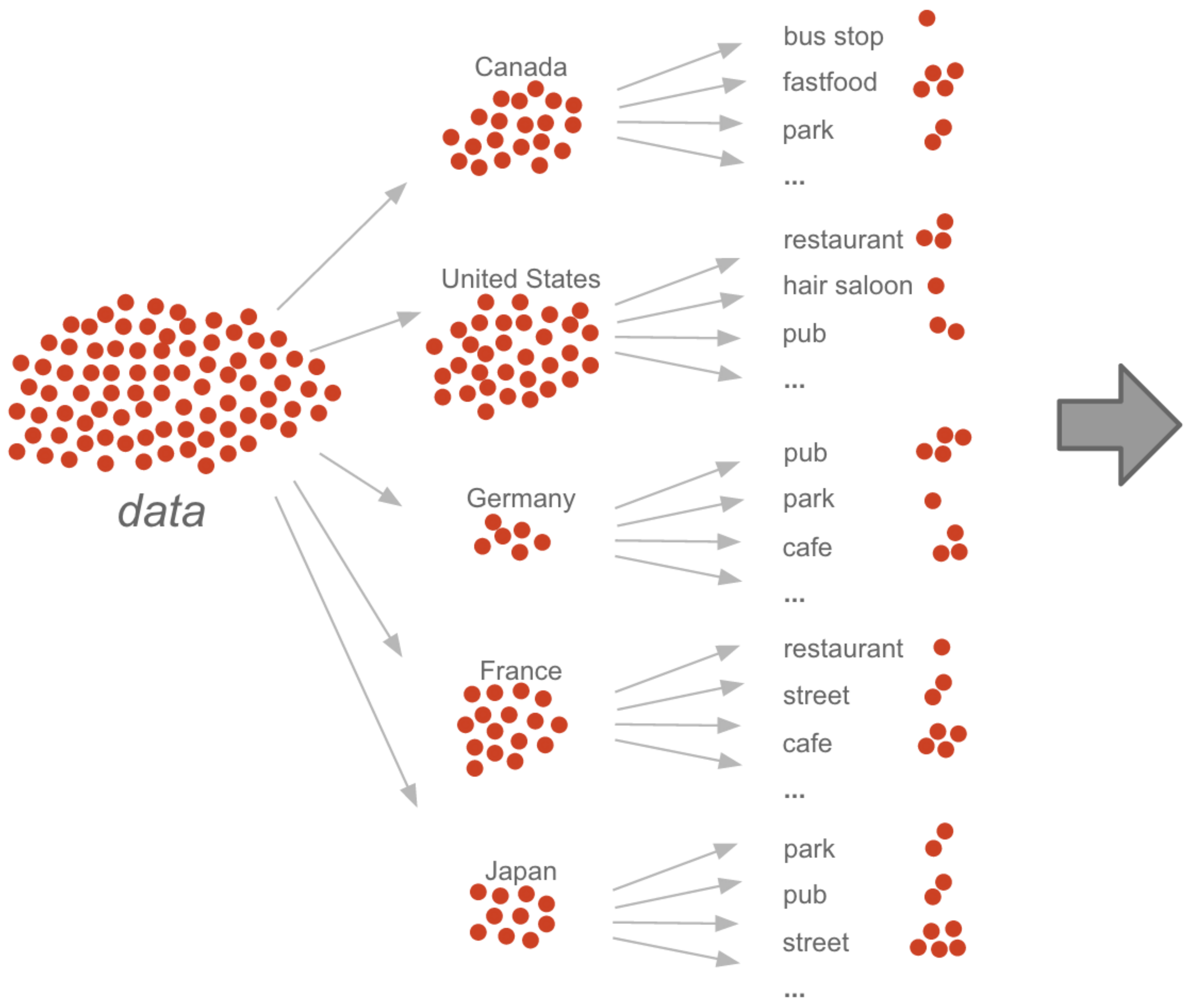
http://www.jstatsoft.org/v40/i01/paper

SPLIT-APPLY-COMBINE



Country	Sum Revenue
United States	$ 83
France	$ 42
Canada	$ 36
Japan	$ 18

SPLIT-APPLY-COMBINE

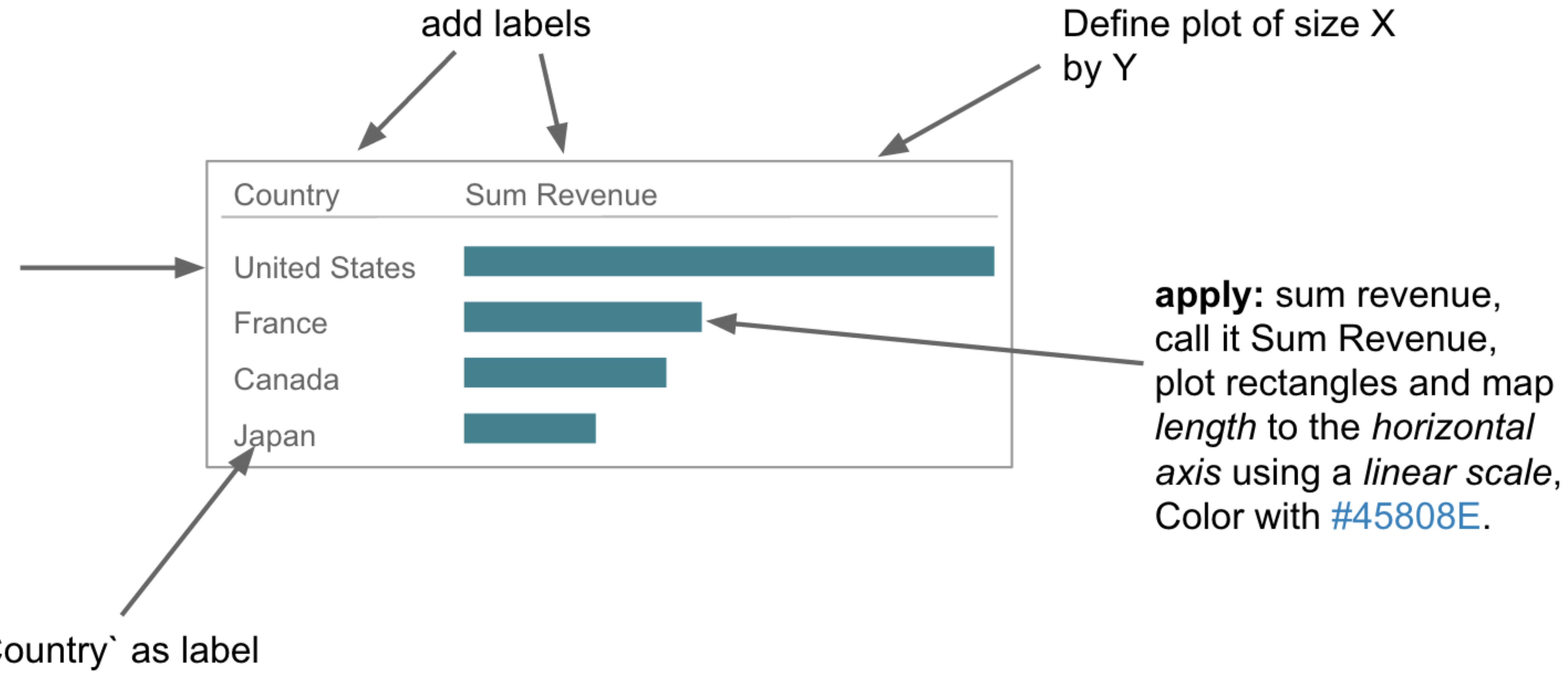


Country	Venue type	Sum Revenue
United States	fastfood	$ 16
	street	$ 10
	restaurant	$ 9
France	cafe	$ 18
	pub	$ 12
	restaurant	$ 2
Canada	cafe	$ 10
	fastfood	$ 4
	park	$ 3
Japan	street	$ 5
	fastfood	$ 4
	pub	$ 1

S-A-C IN DATA QUERIES

```
SELECT
  `country` AS "Country",
  SUM(`revenue`) AS "Sum Revenue" -- Apply: sum Revenue
FROM `myDataTable`
GROUP BY `country`                -- Split by country
ORDER BY `Sum Revenue` DESC       -- Combine by sorting on
LIMIT 4;                          -- Sum Revenue and limiting
```

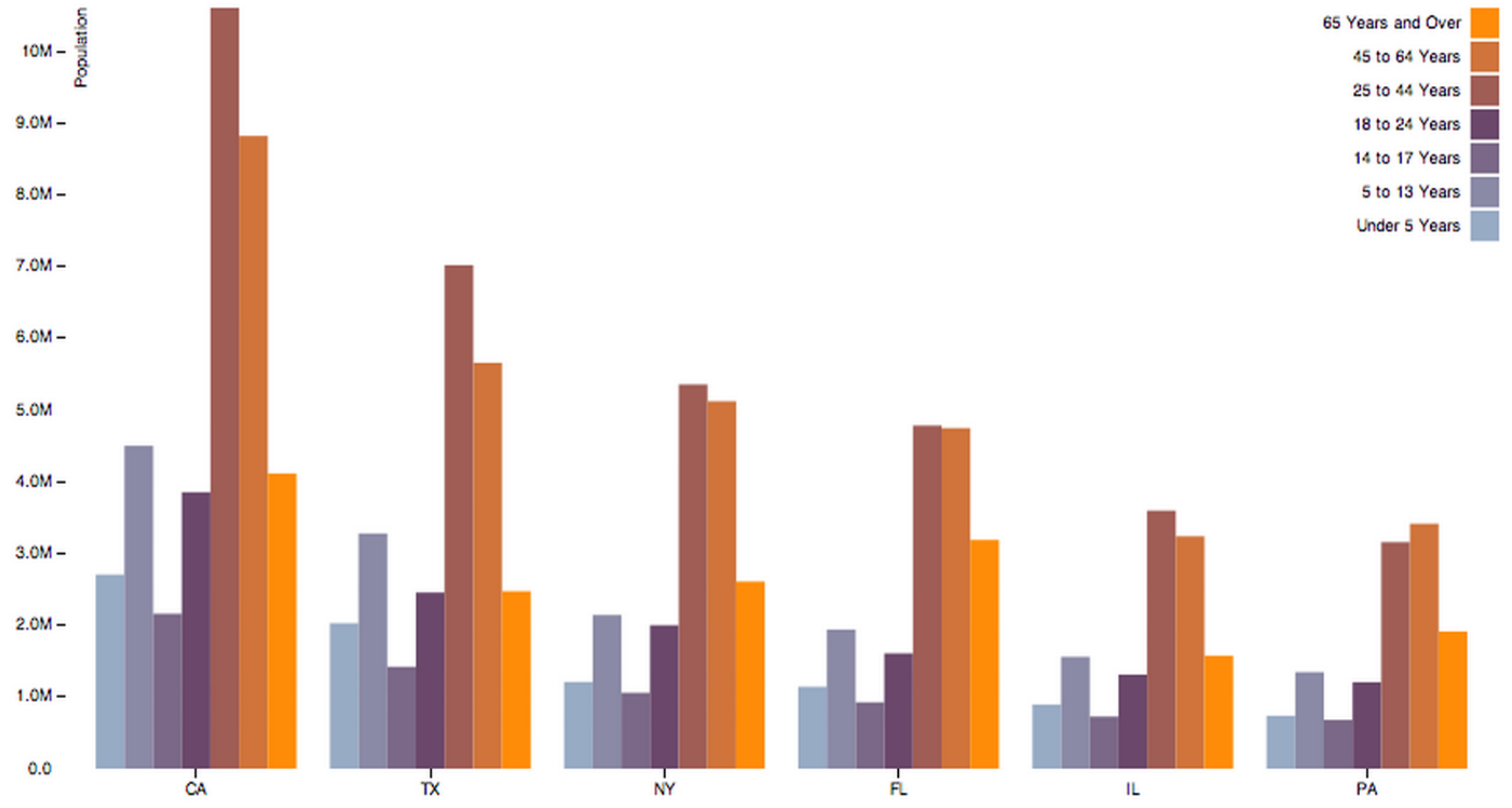
S-A-C IN VISUALIZATION


add labels
Define plot of size X by Y
split by country, **combine** by sorting desc. on Sum Revenue, map to the *vertical axis* using an *ordinal scale*.
Country
Sum Revenue
United States
France
Canada
Japan
apply: sum revenue, call it Sum Revenue, plot rectangles and map *length* to the *horizontal axis* using a *linear scale*, Color with #45808E.
Use `Country` as label

NESTED S-A-C IN VISUALIZATION

1. **split** on state
   apply sum population
   combine: sort by population,
   limit 6

2. **split** on age (bin by 5 year)
   apply sum population
   combine: sort by age



FACET.JS

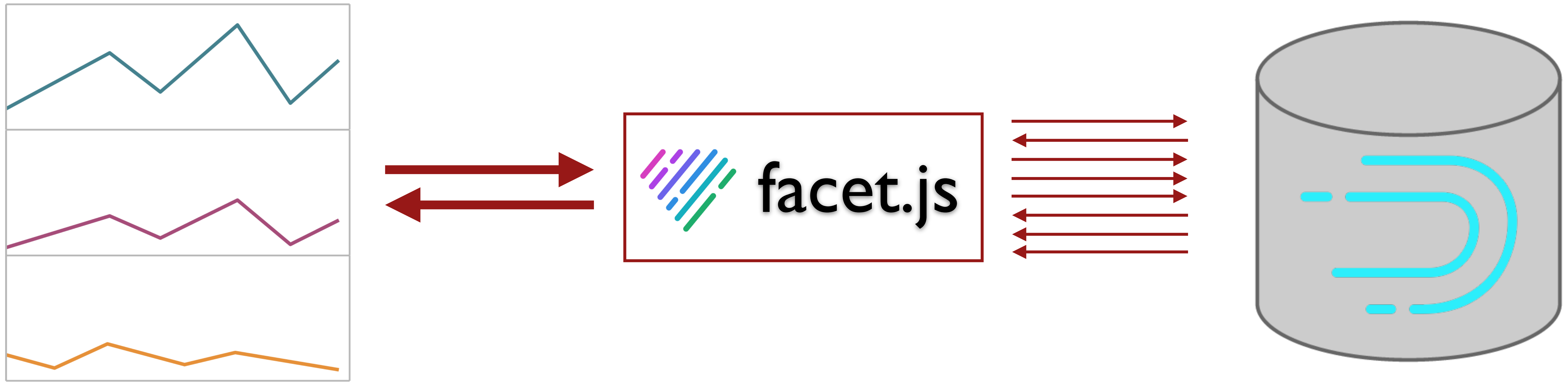
- A high level query language built on split-apply-combine
- Has query planners for Druid and MySQL
- Can compute queries natively
- Opened sourced today under the Apache 2.0 license

EXAMPLE (NESTED) QUERY

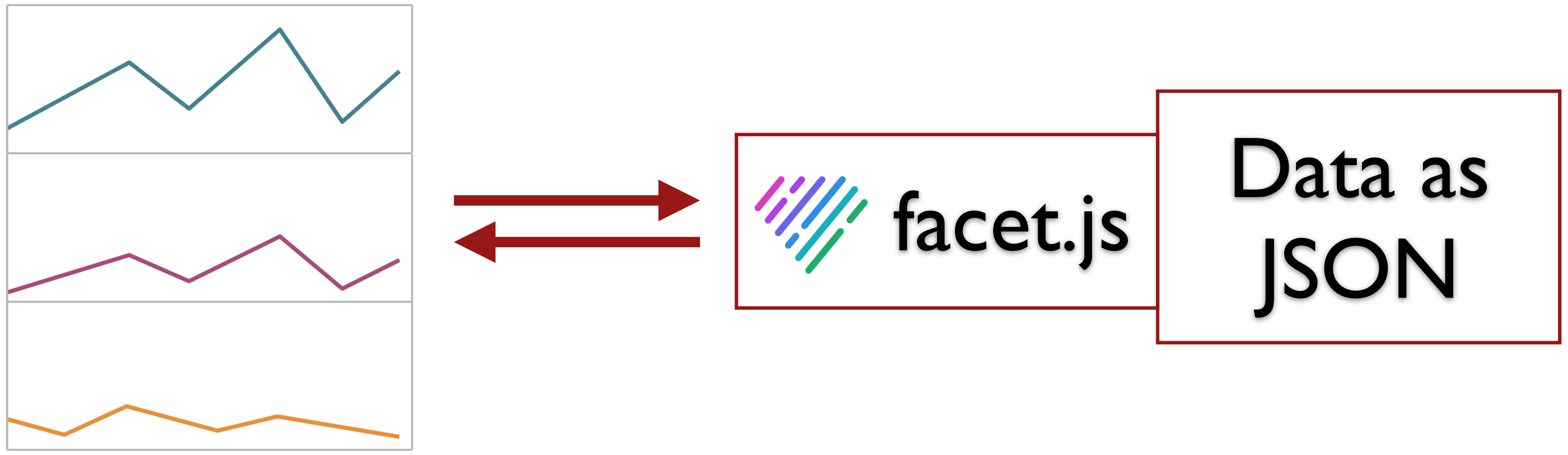
```
var populationDriver = facet.driver.druid({
  /* driver parameters */
})

var query = facet("data")
  .split("$country", 'Country')
    .apply('Revenue', '$data.sum($revenue)')
    .sort('Revenue', 'descending')
    .limit(3)
    .apply('Times',
      facet("data")
        .split("$timestamp.timeRange('PT1H', 'Etc/UTC')", 'Time')
        .apply('Revenue', '$data.sum($revenue)')
        .sort('Time', 'ascending')
    )

query.compute(populationDriver).then(function(data) {
  console.log(data)
});
```

```
[
  {
    Country: 'United States'
    Revenue: 22124
    Times: [
      {
        Time: TimeRange('2015-02-20T00:00:00', '2015-02-20T01:00:00')
        Revenue: 463
      }
      {
        Time: TimeRange('2015-02-20T01:00:00', '2015-02-20T02:00:00')
        Revenue: 245
      }
      // ...
    ]
  }
  {
    Country: 'United Kingdom'
    Revenue: 14525
    Times: [
      {
        Time: TimeRange('2015-02-20T00:00:00', '2015-02-20T01:00:00')
        Revenue: 210
      }
      {
        Time: TimeRange('2015-02-20T01:00:00', '2015-02-20T02:00:00')
        Revenue: 110
      }
      // ...
    ]
  }
  // ...
]
```

DRUID + FACET



UNIT TESTING



CONCLUSION

- Building data applications is hard
- Getting a flexible, scalable, fault tolerant system that can return results in milliseconds is difficult.
- Building a UI on top of that is painful without a good level of abstraction.
- Our solutions make it easier

THANK YOU

@DRUIDIO @FACETJS

@METAMARKETS