a stripe talk

# Aggregator: MapReduce in the type system

by Dan Frank | @danielhfrank

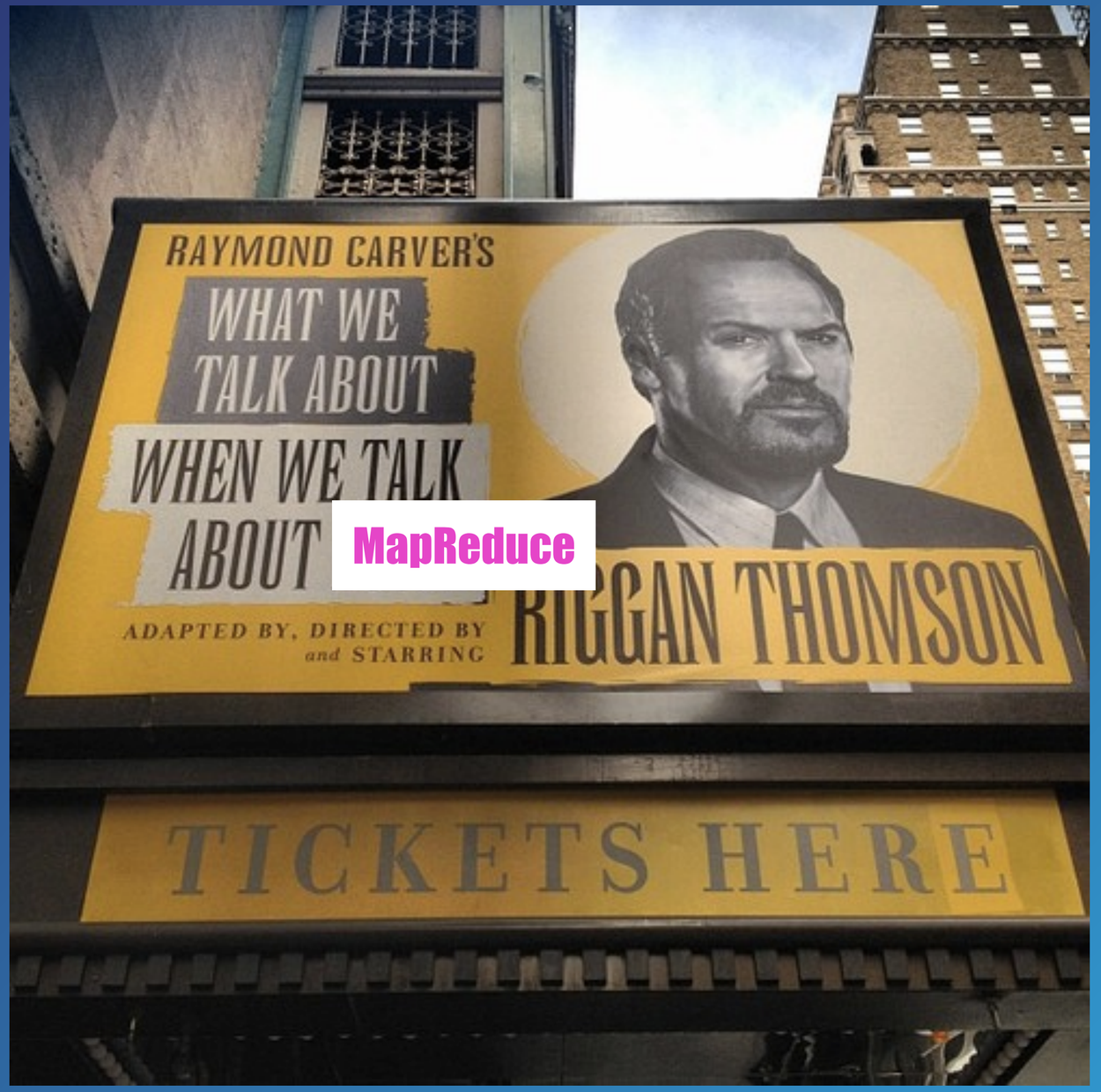

stripe

stripe

# Data at Stripe

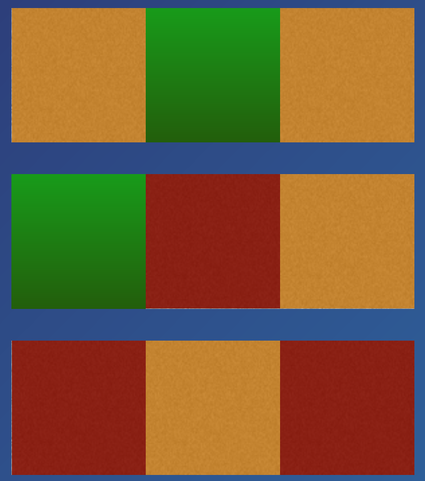
- Units: cards, users, customers, transactions
- Product usage analytics for business / product development, machine learning for fraud prevention, and much more
- Today: Use Aggregator on this data to predict fraud

stripe


RAYMOND CARVER'S
WHAT WE TALK ABOUT
WHEN WE TALK ABOUT
MapReduce
ADAPTED BY, DIRECTED BY and STARRING
RIGGAN THOMSON
TICKETS HERE

stripe

# Raw Data



# Properties

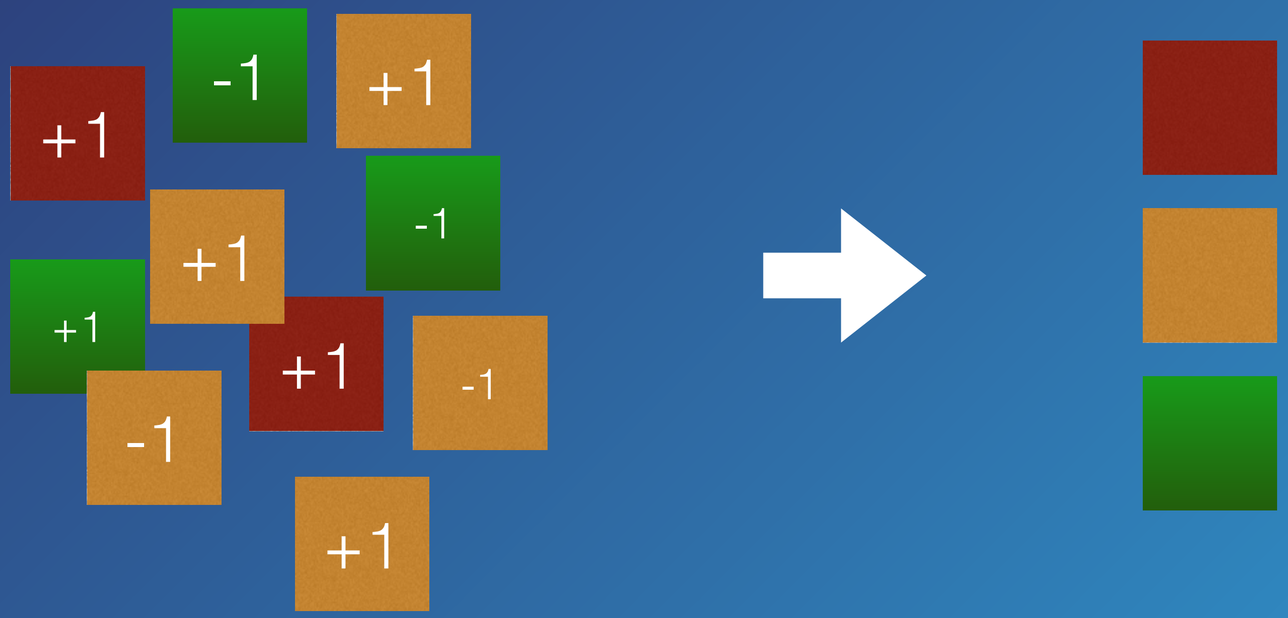
stripe

# Further Reduction


Max = 4
Mean = 3
Std = 0.82

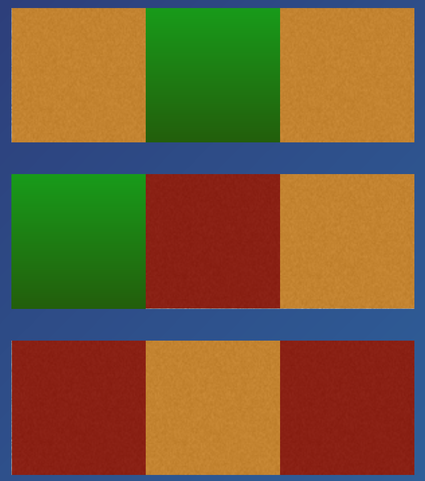
stripe

# Example: Comment Ranking



stripe

# Raw Data



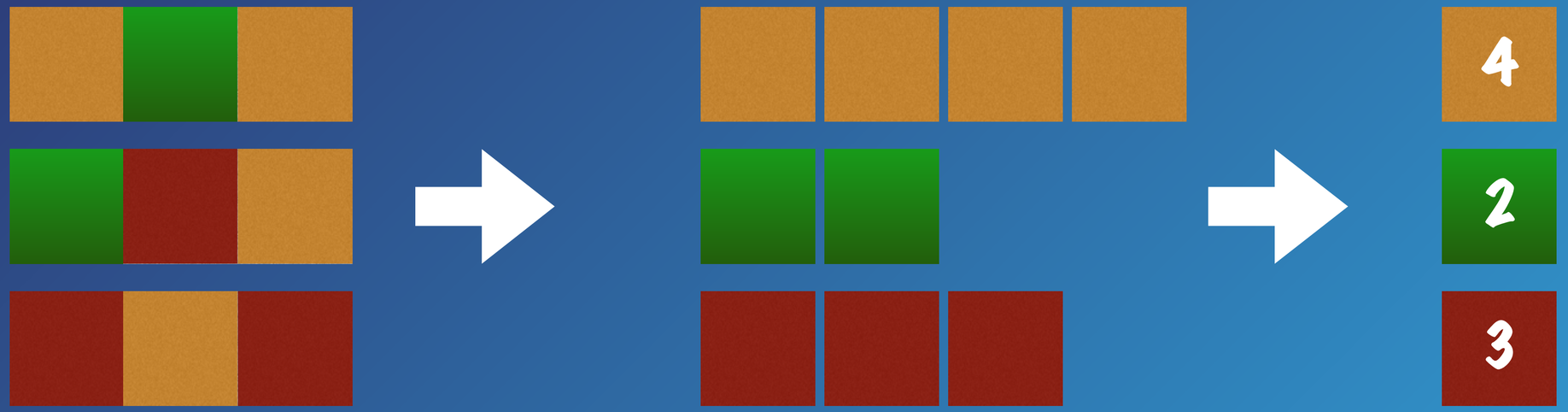
# Properties

stripe

# Example: How many charges per merchant?

```
1  SELECT merchant, COUNT(*)
2  FROM charges
3  GROUP BY 1
```

stripe



stripe

```
map(K1,V1) => (K2,V2)
reduce(K2,Seq[V2]) => (K3,V3)
```

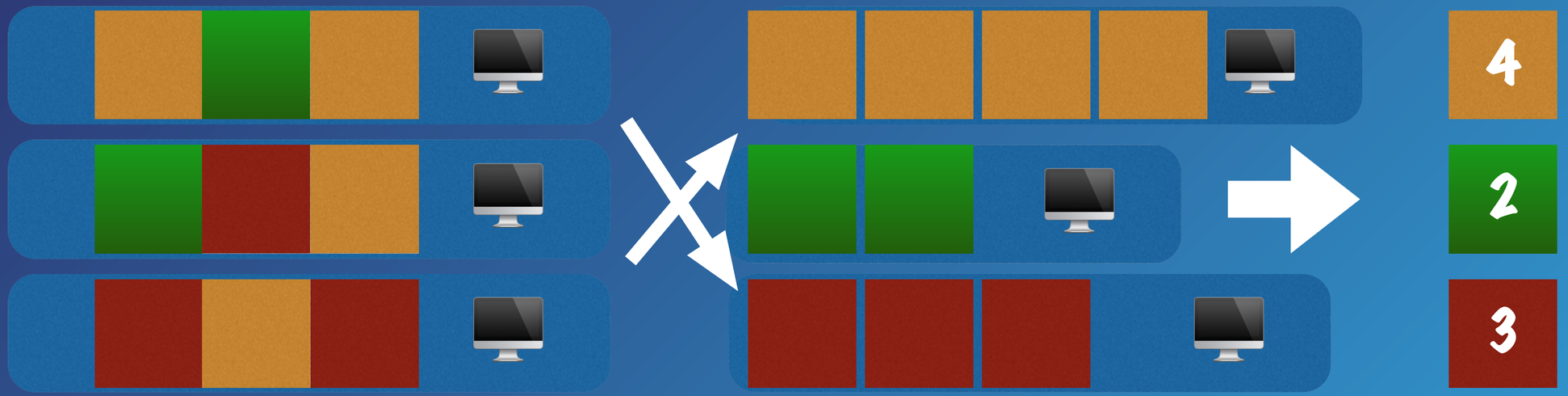
stripe

# Example: How many charges per merchant?

```
1  def map(charge):
2      yield (charge['merchant'], 1)
3
4  def reduce(key, values):
5      yield (key, sum(values))
```

stripe

# Why MapReduce?



stripe

# Breaking the decomposition boundary

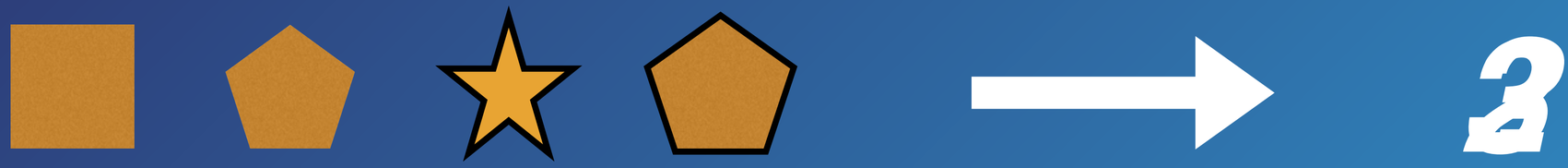
(Orange, 1)

stripe

# The "real" components

- Group data points by entities of interest (keys)
- Reduce a sequence of data points to a quantity of interest

stripe



stripe

# Example: How many cards per merchant?

```
1  SELECT merchant, COUNT(DISTINCT(card))
2  FROM charges
3  GROUP BY 1
```

stripe

# Example: How many cards per merchant?

```
1  def map(_, charge):
2      yield (charge['merchant'], charge['card'])
3
4  def reduce(key, values):
5      yield (key, len(set(values)))
```

stripe

# Aggregator

- Scala trait in Algebird: "Abstract Algebra for Scala"
- Use types to guide composition and reuse
- Agnostic to execution platform

stripe

# Example: How many cards per merchant?

```
Seq[Charge] => Int
```

stripe

# Computation type

```
Seq[Charge] => Set[Card] => Int
```

stripe

```
val chars = List('S', 'C', 'A', 'L', 'A')

chars.map(_ => 1).reduce(_ + _) // => 5

chars.map(Set(_)).reduce(_ ++ _).size // => 4
```

stripe

# Aggregator Components

```
Aggregator[A,B,C]

  prepare: A => B
reduce: (B, B) => B
  present: B => C
```

stripe

stripe

# Example: How many cards per merchant?

```
1 class MerchantCardCount extends Aggregator[Charge, Set[Card], Long] {
2     def prepare(a: Charge) = Set(a.card)
3
4     def reduce(b1: Set[Card], b2: Set[Card]) = b1 + b2
5
6     def present(b: Set[Card]) = b.size
7 }
```

stripe

# The Next Level

- Composition, reuse, and the "standard library"
- Aggregator execution patterns
- Aggregator and fraud at Stripe

stripe

# Leveraging the standard library

```
Aggregator.uniqueCount[Card]
```

```
[Card, Set[Card], Int]
```

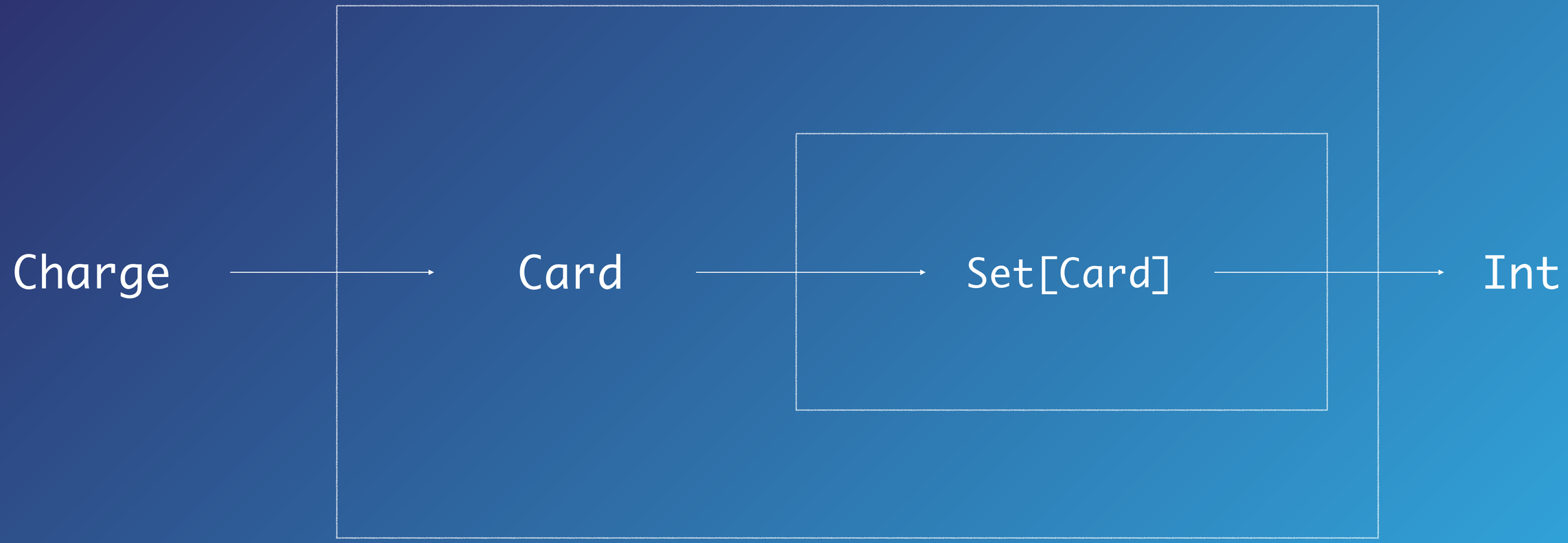
stripe

# composePrepare

```
Aggregator.uniqueCount[Card]
    .composePrepare(a: Charge => a.card)
```

```
[Charge, Set[Card], Int]
```

stripe



stripe

# *Really* leveraging the standard library

```
SetSizeAggregator[Card](10) // bits for hll
    .composePrepare(a: Charge => a.card)
```

```
[Charge, HLL[Card] , Int]
```

stripe

Approx Set Size -> HyperLogLog

Approx Item Count -> CountMinSketch

Approx Quantiles -> QTree

Heavy Hitters, Decayed Value, etc..

stripe

# Aggregator Execution

```
apply(inputs: TraversableOnce[A]): C
```

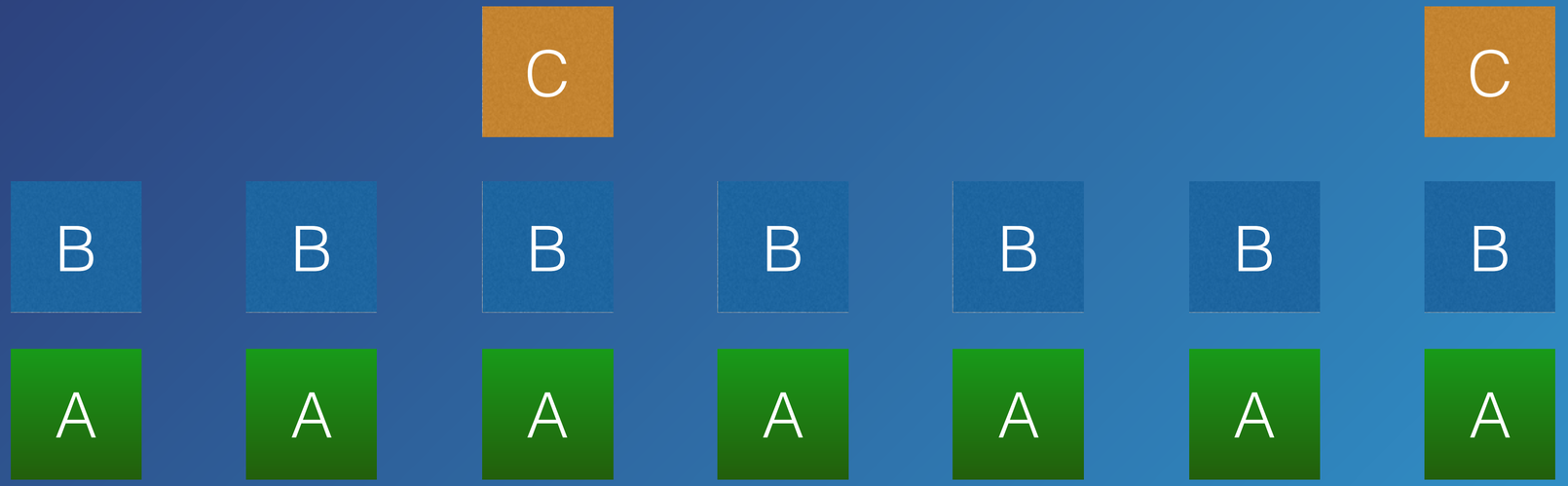
stripe

# MapReduce with Aggregator

```
def map(v: V) = (key(v), V)

def reduce(k: K, vs: Seq[V]) = agg(vs)
```

stripe

# More execution



stripe

# Fraud Prevention at Stripe

- Merchants as bad actors
- Detect via activity on Stripe (charges)

stripe

# Predictive Model

When evaluating a merchant, compute score like:

```
sum(weight1*feature1 + weight2*feature2 + ...)
```

stripe

Static Features: Location, Time of Day...

Dynamic Features: Mean Charge Amount, Number of Cards

stripe

# Feature: Number of international cards

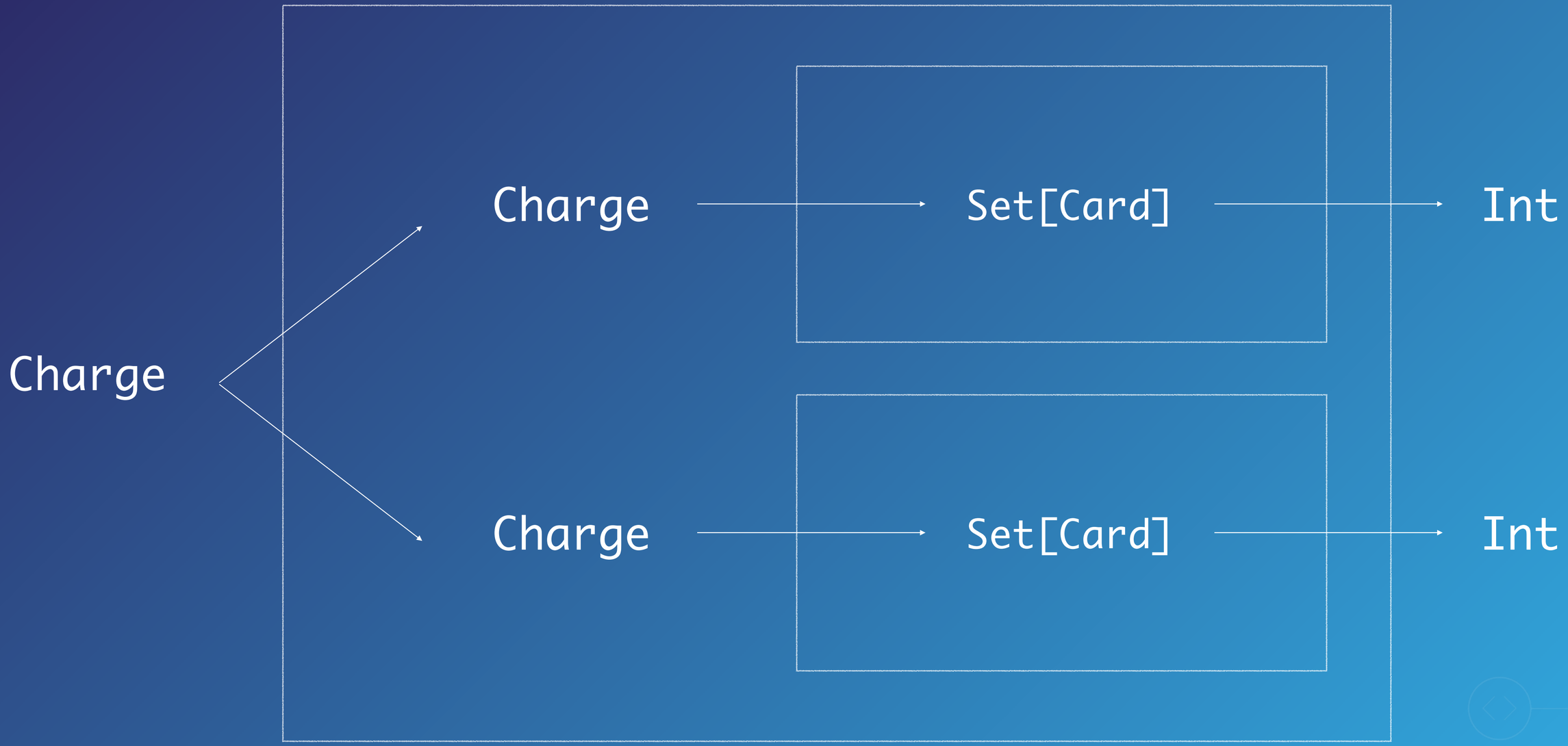
```
def internationalCard(c: Charge): Option[Card] = ???

SetSizeAggregator[Card](10) // our normal SetSize
    .sumBefore() // make it work with Option
    .composePrepare(a: Charge => a.internationalCard)
```

stripe

# Composing Aggregators

```
val intlRatio: Aggregator[A, (B1, B2), (C1, C2)] =
    from2(intlCards, totalCards)
```

stripe

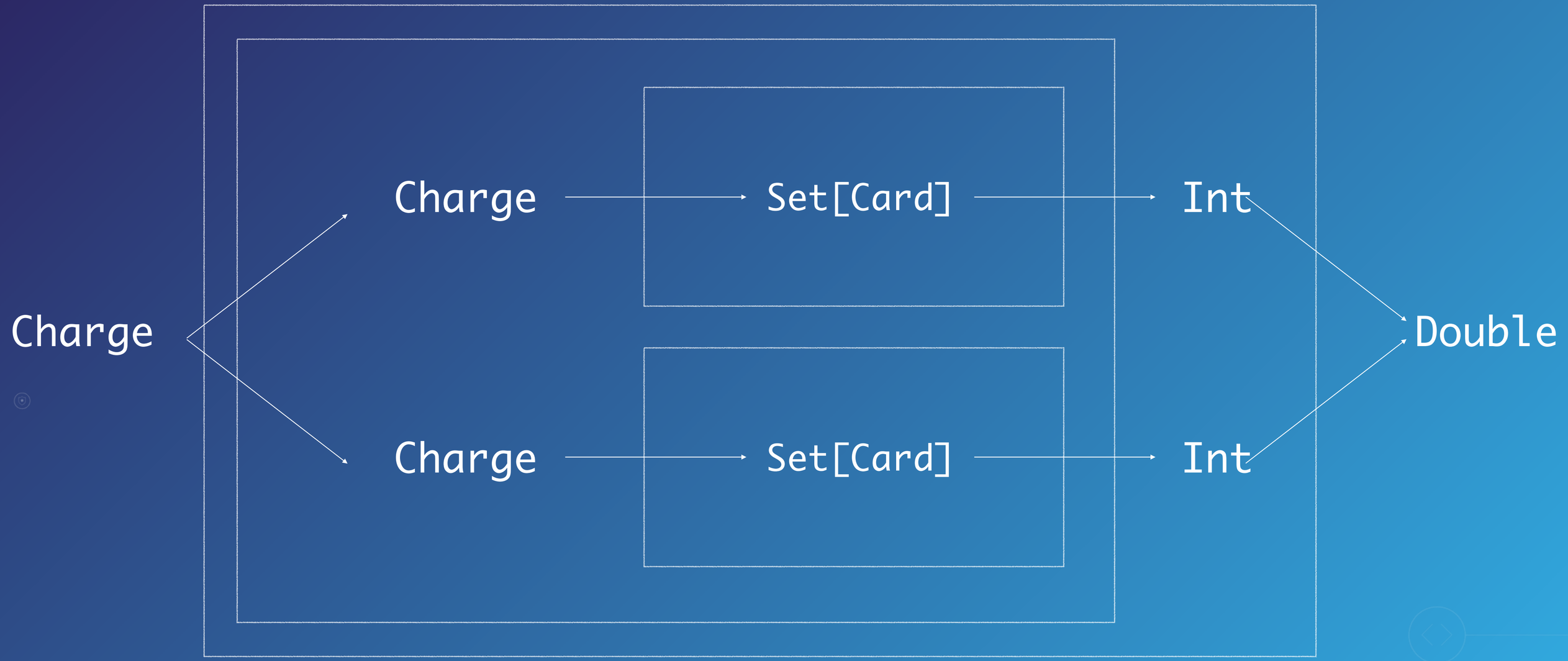


stripe

# andThenPresent

```
val intlRatio: Aggregator[A, (B1, B2), Double] =
    from2(intlCards, totalCards)
    .andThenPresent(case (intl,total) => intl / total)
```

stripe



stripe

# More features, no problem

```
val features = from2(intlRatio, prepaidRatio)
    .andThenPresent(toJson(_))
```

stripe

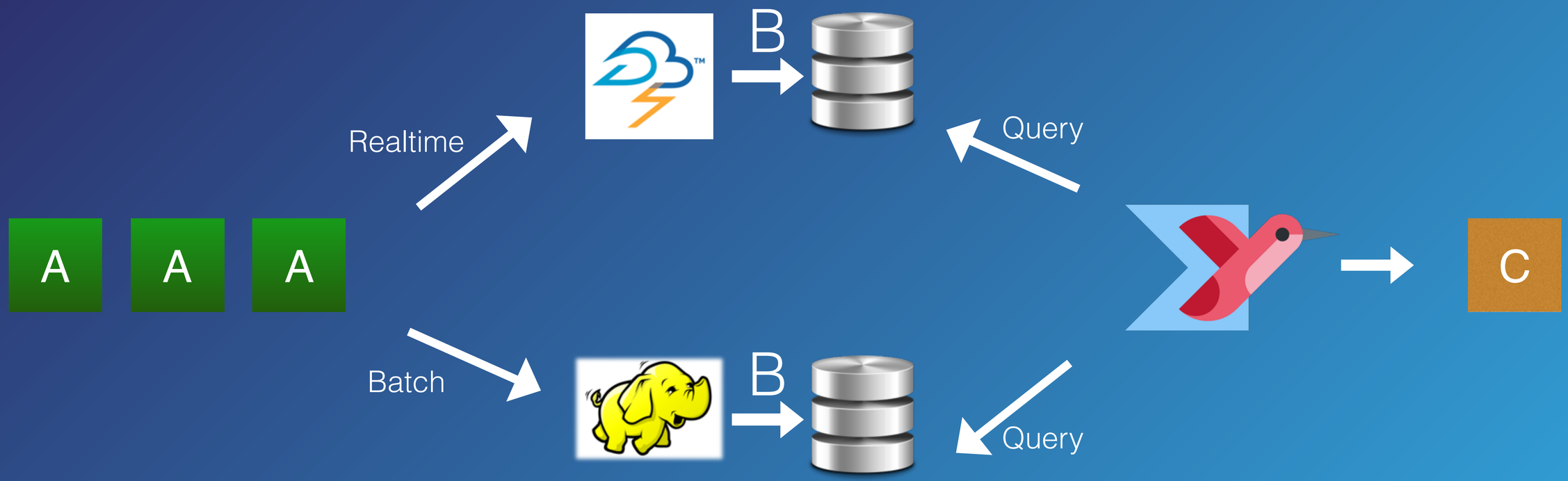
# Fraudsters Don't Run on Cron

stripe

# Beyond Hadoop

- Summingbird: run MapReduce-like code on Storm or Hadoop
- Combine results from two platforms at read-time
- Run complex calculations with Algebird + Aggregator

stripe



stripe

# Recap

- Reduction (and MapReduce) at heart of data analysis
- Aggregator defines reduction in the type system
- (De)composability encourages reuse
- Execution-agnosticism encourages flexibile deployments

stripe


https://stripe.com/jobs/

stripe

# Thanks.

Dan Frank | @danielhfrank | df@stripe.com