Customer Behaviour Analytics: Billions of Events to one Customer-Product Graph

Strata London, 12th November 2013
Presented by Paul Lam

About Paul Lam

Joined uSwitch.com as first Data Scientist in 2010

- developed internal data products
- built distributed data architecture
- team of 3 with a developer and a statistician

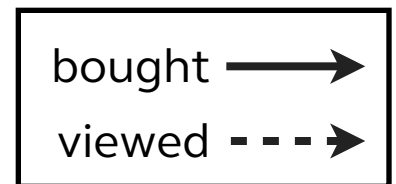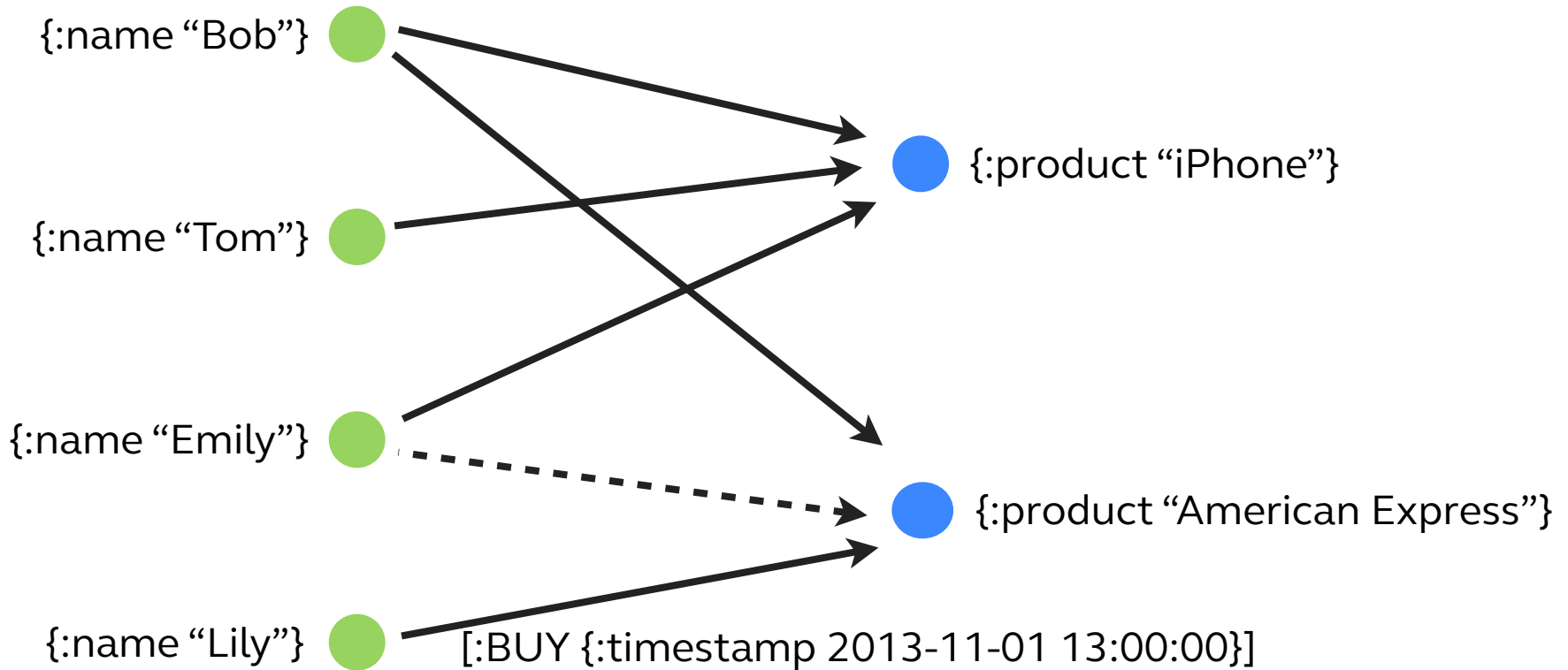Code contributor to various open source tools

- Cascalog, a big data processing library built on top of Cascading (comparable to Apache Pig)
- Incanter, a statistical computing platform in Clojure

Author of *Web Usage Mining: Data Mining Visitor Patterns From Web Server Logs** to be published in late 2014
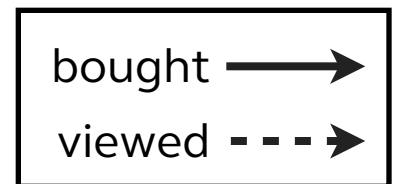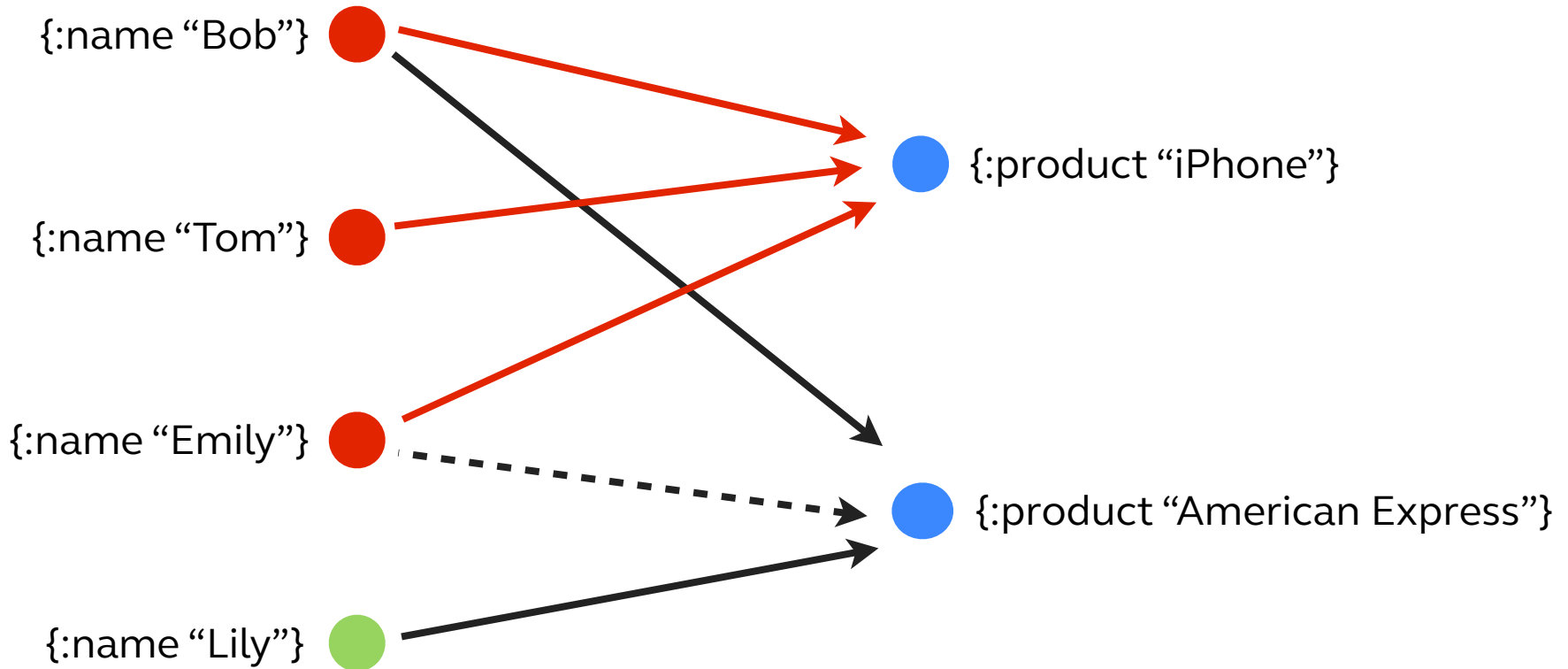
*tentative title

What is it

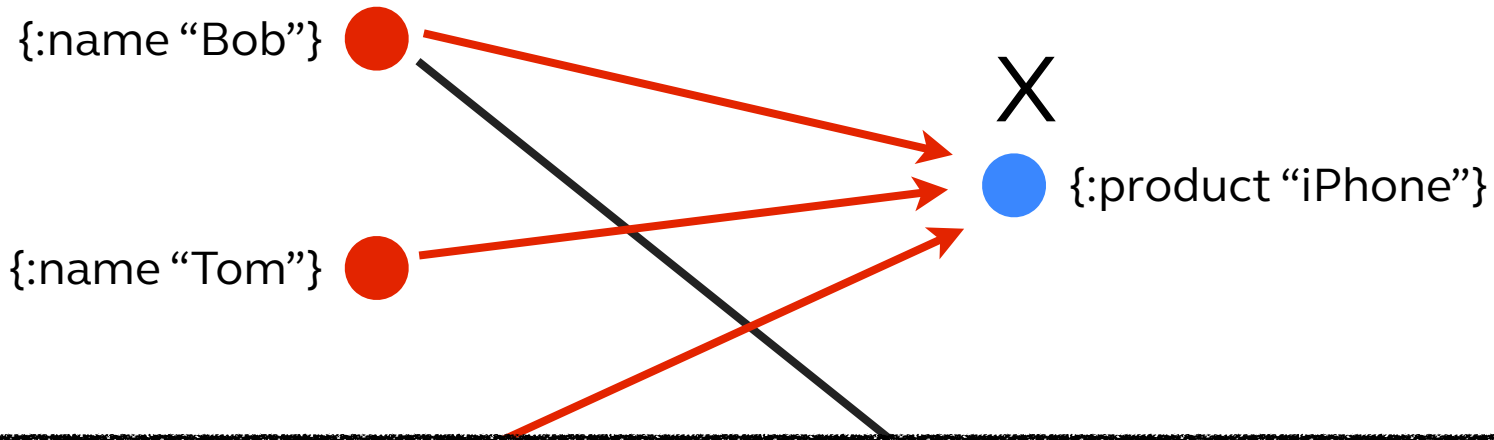# Customer-Product Graph

{:name "Bob"} ●

{:name "Tom"} ●

{:name "Emily"} ●

{:name "Lily"} ● [:BUY {:timestamp 2013-11-01 13:00:00}]

● {:product "iPhone"}

● {:product "American Express"}

bought ——→
viewed ╌╌→

Question: Who bought an iPhone?

Query: Who bought an iPhone?

{:name "Bob"}

{:name "Tom"}

X

{:product "iPhone"}

```
START
x=node:node_auto_index(product='iPhone')

MATCH (person)-[:BUY]->(x)

RETURN person
```

Neo4j
the graph database

Question: What else did they buy?
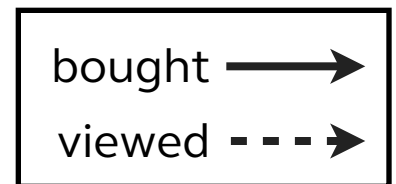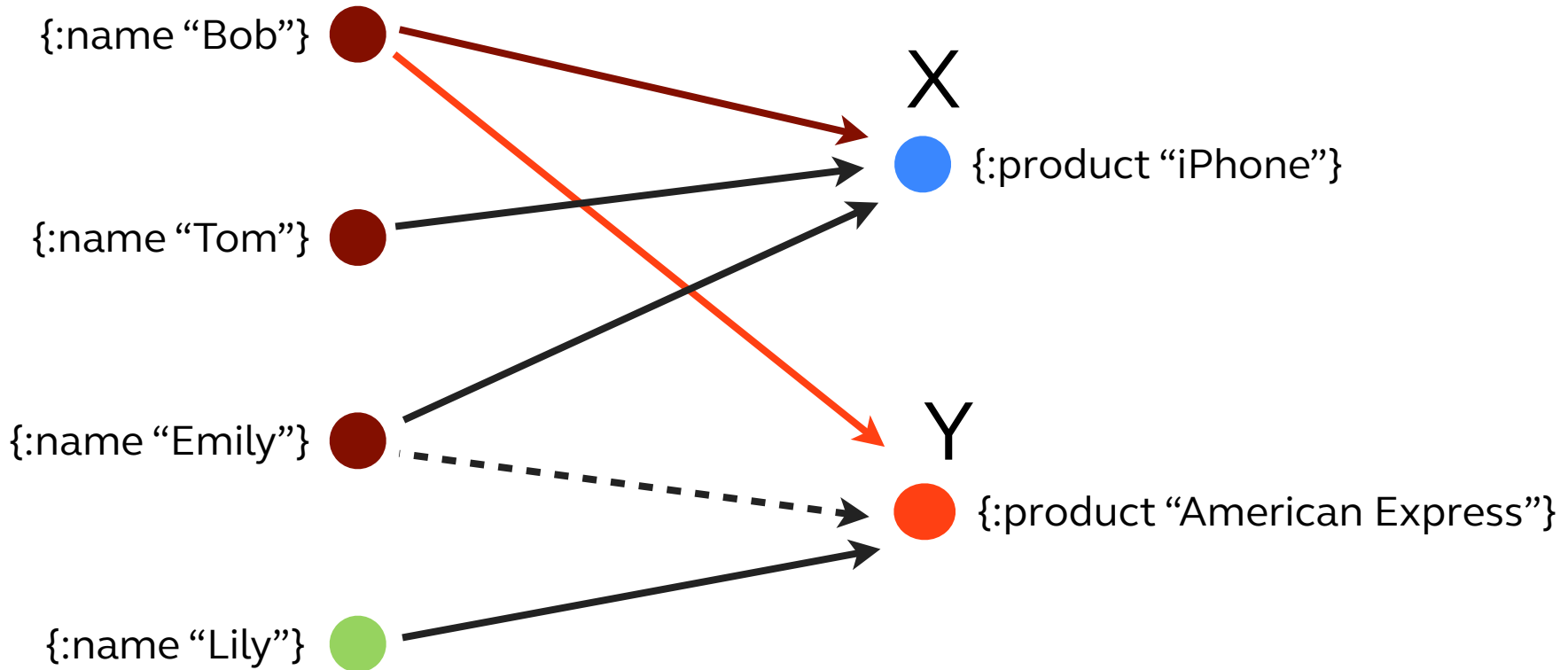
Query: What else did they buy?
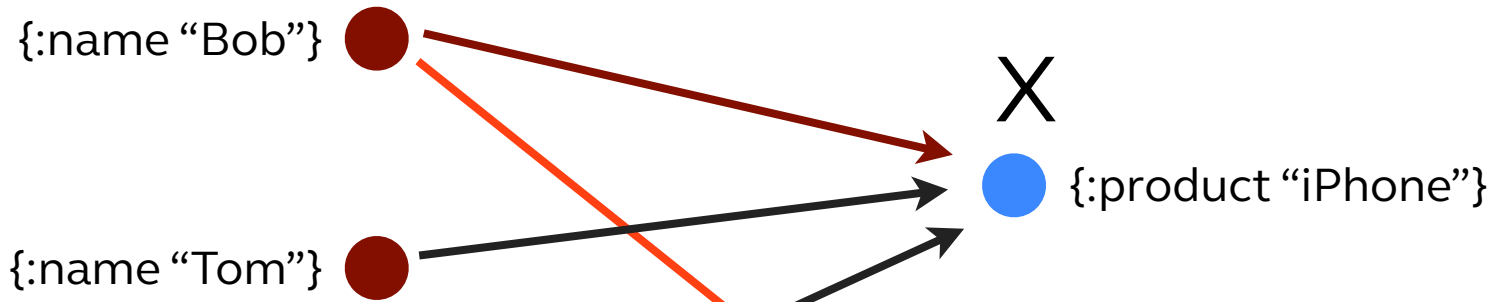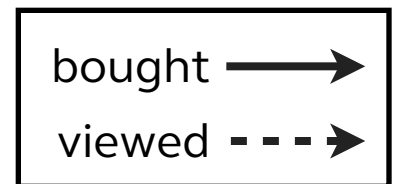


```
START x=node:node_auto_index(product='iPhone')

MATCH

   (x)<-[:BUY]-(person)-[:BUY]->(y)

RETURN y
```

Hypothesis: People that buy X has interest in Y
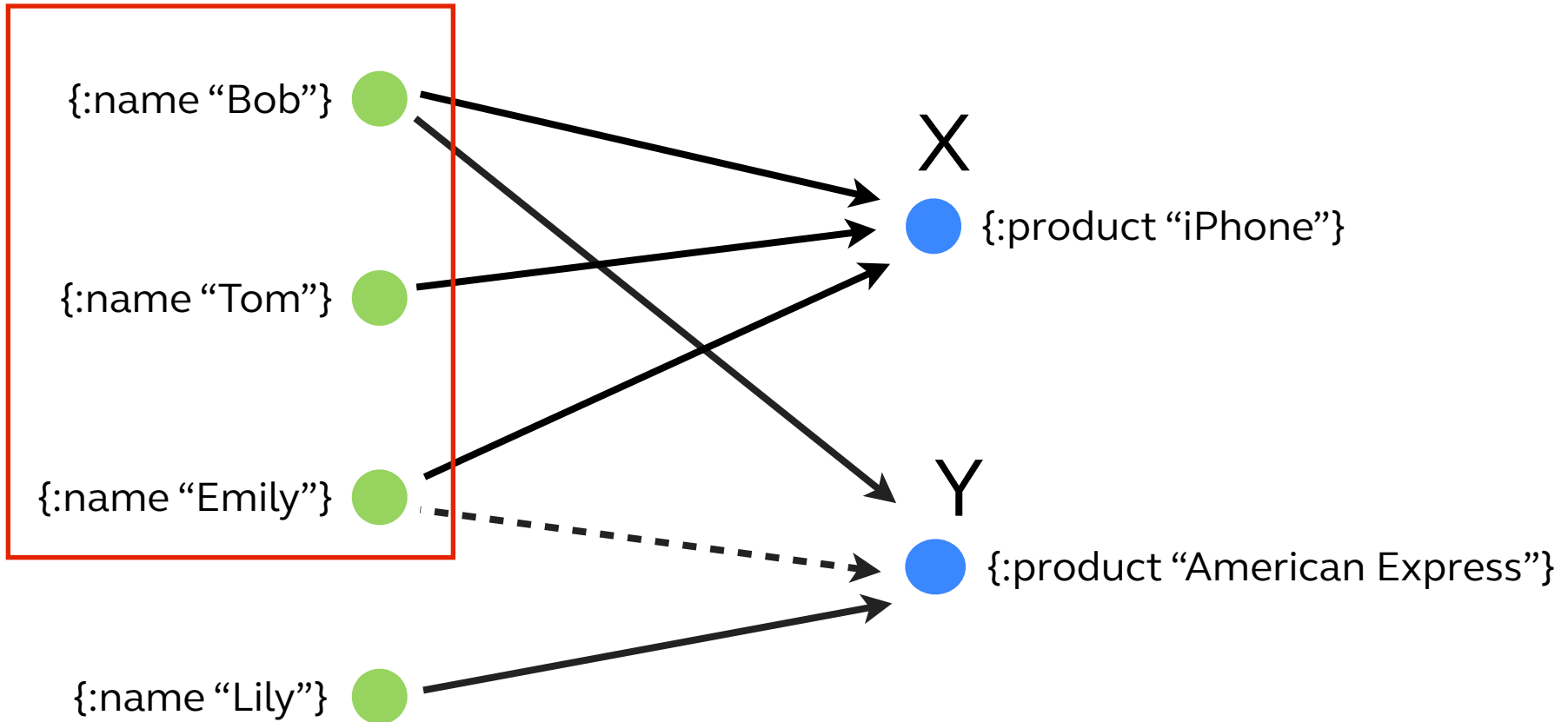
{:name "Bob"}

{:name "Tom"}

{:name "Emily"}

{:name "Lily"}

X

{:product "iPhone"}

Y

{:product "American Express"}

bought ⟶
viewed ⤏

Query: Who to recommend Y

Product Recommendation by Reasoning Example

Interactive demo at http://bit.ly/customer_graph

1. Start with an idea
2. Trace to connected nodes
3. Identify patterns from viewpoint of those nodes
4. Repeat from #1 until discovering actionable item
5. Apply pattern

Challenge: Event Data to Graph Data

User ID	Product ID	Action
Bob	iPhone	Bought
Tom	iPhone	Bought
Emily	iPhone	Bought
Bob	AE	Bought
Emily	AE	Viewed
Lily	AE	Bought
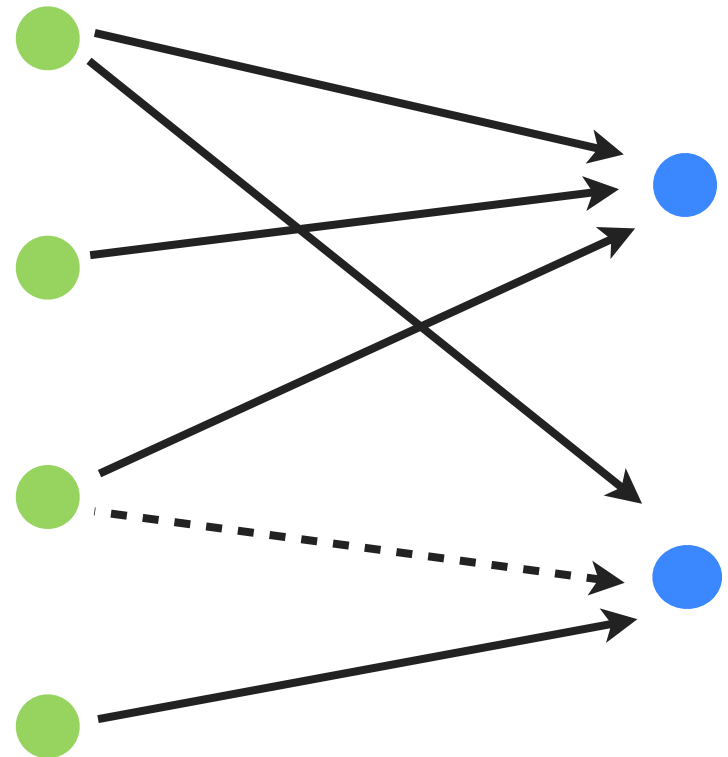
Why should you care

Customer Journey

Purchase Funnel

Interest

Desire
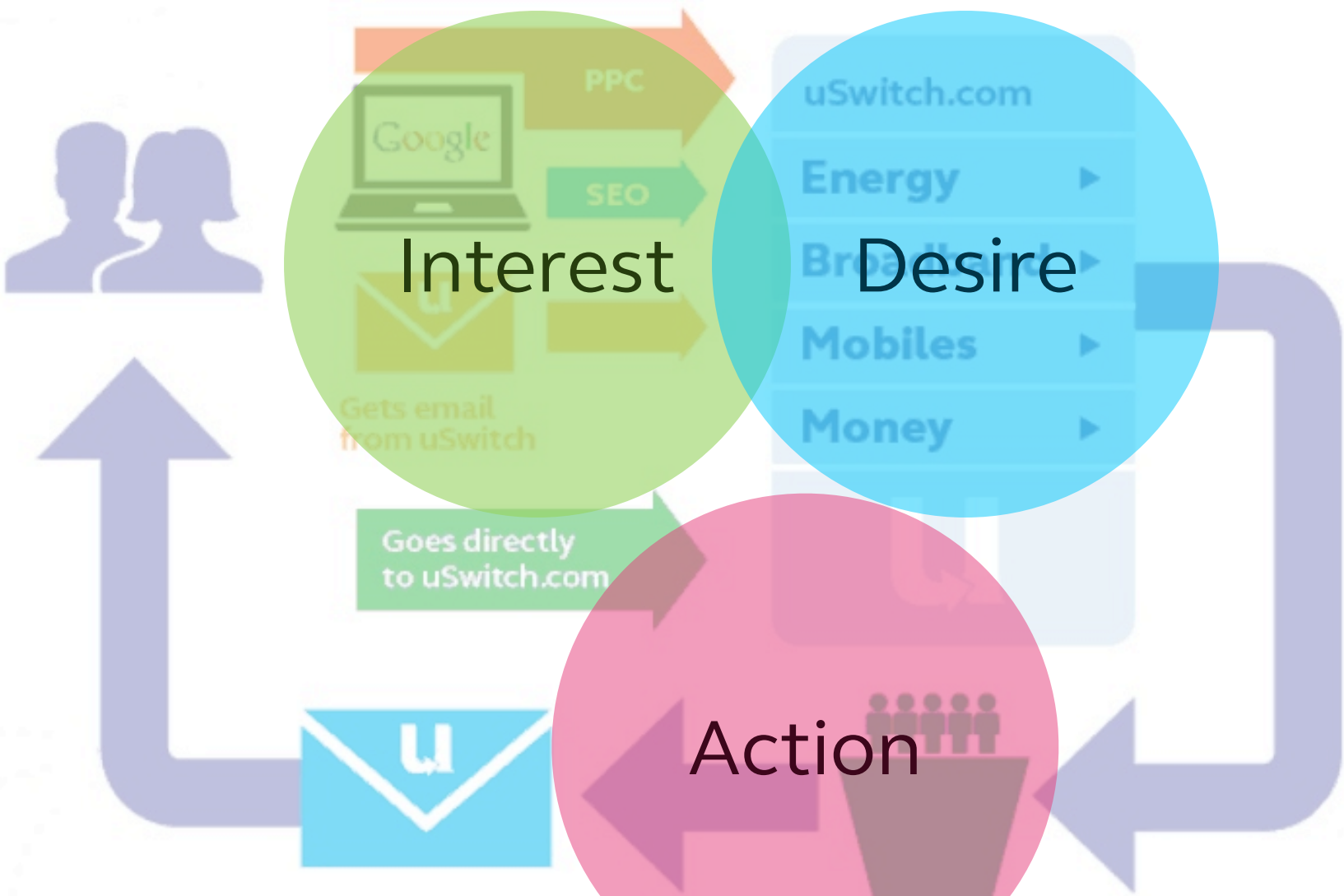
Action

Understanding Customer Intents

Interest

Desire
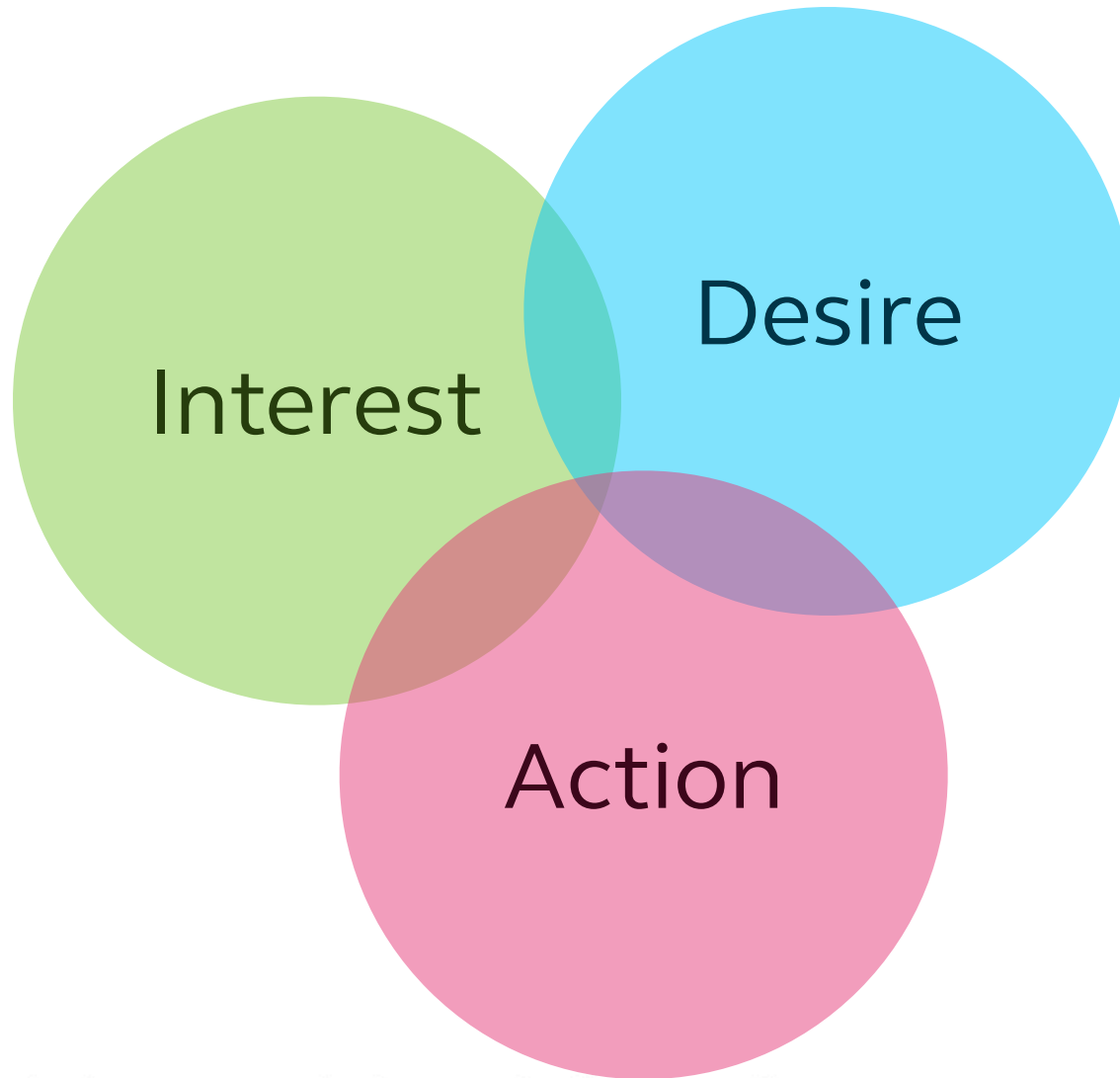
Action

Customer Experience as a Graph

http://insideintercom.io/criticism-and-two-way-streets/
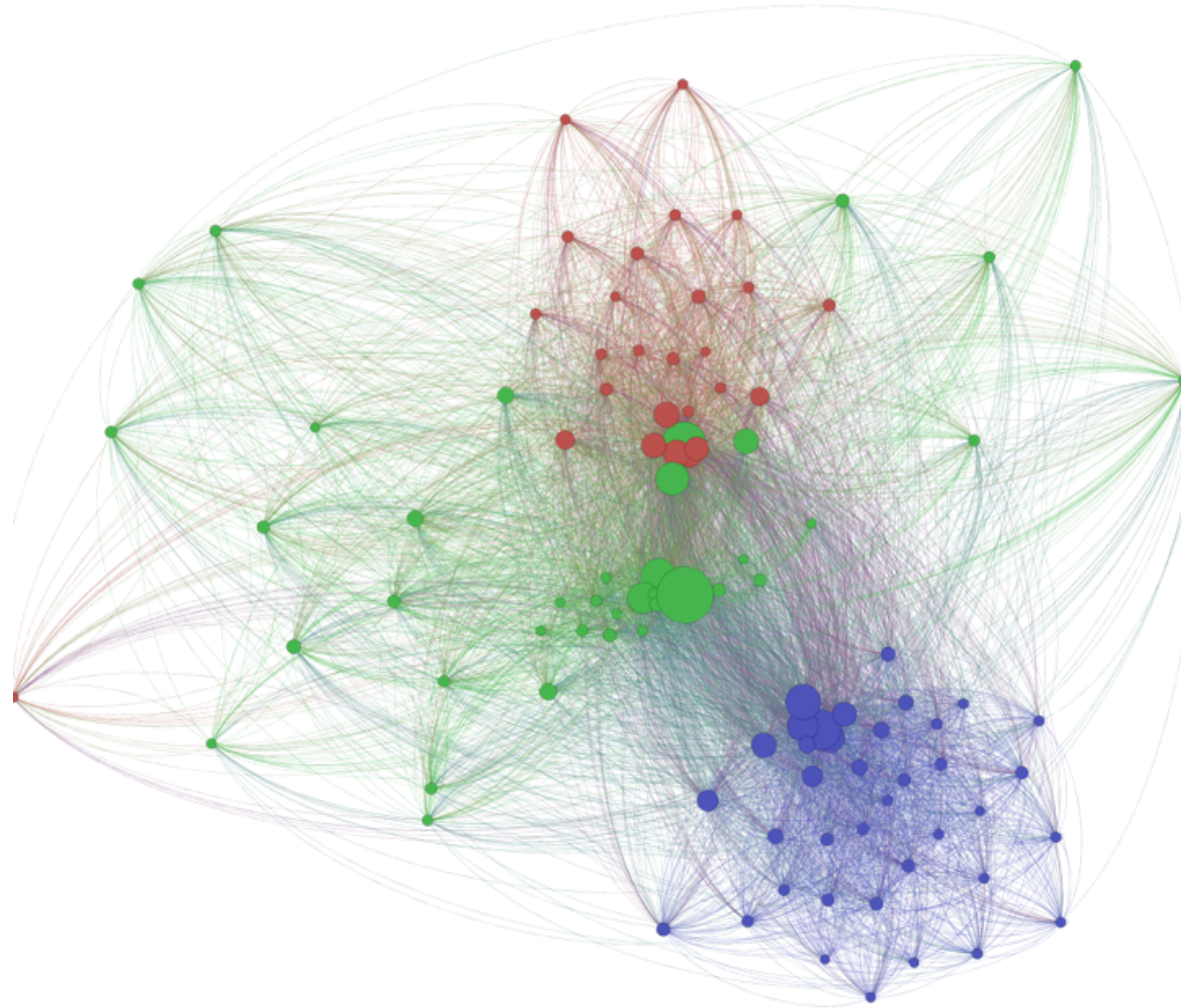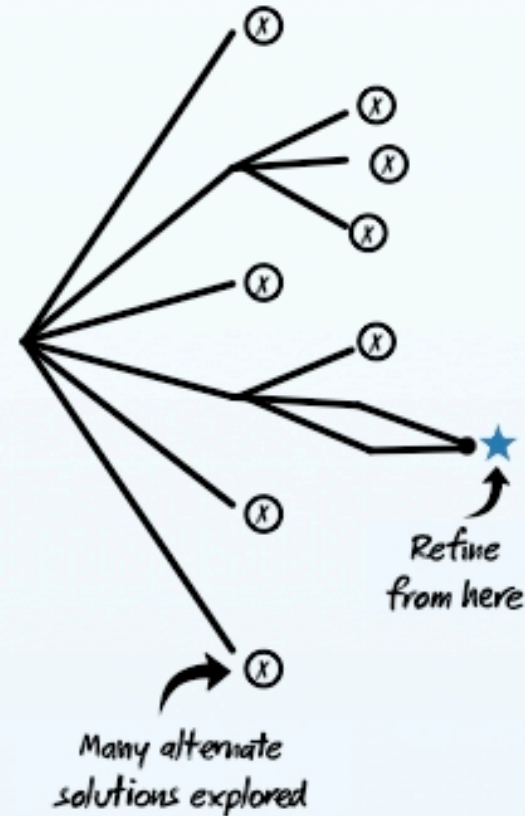
A Feedback System to Drive Profit
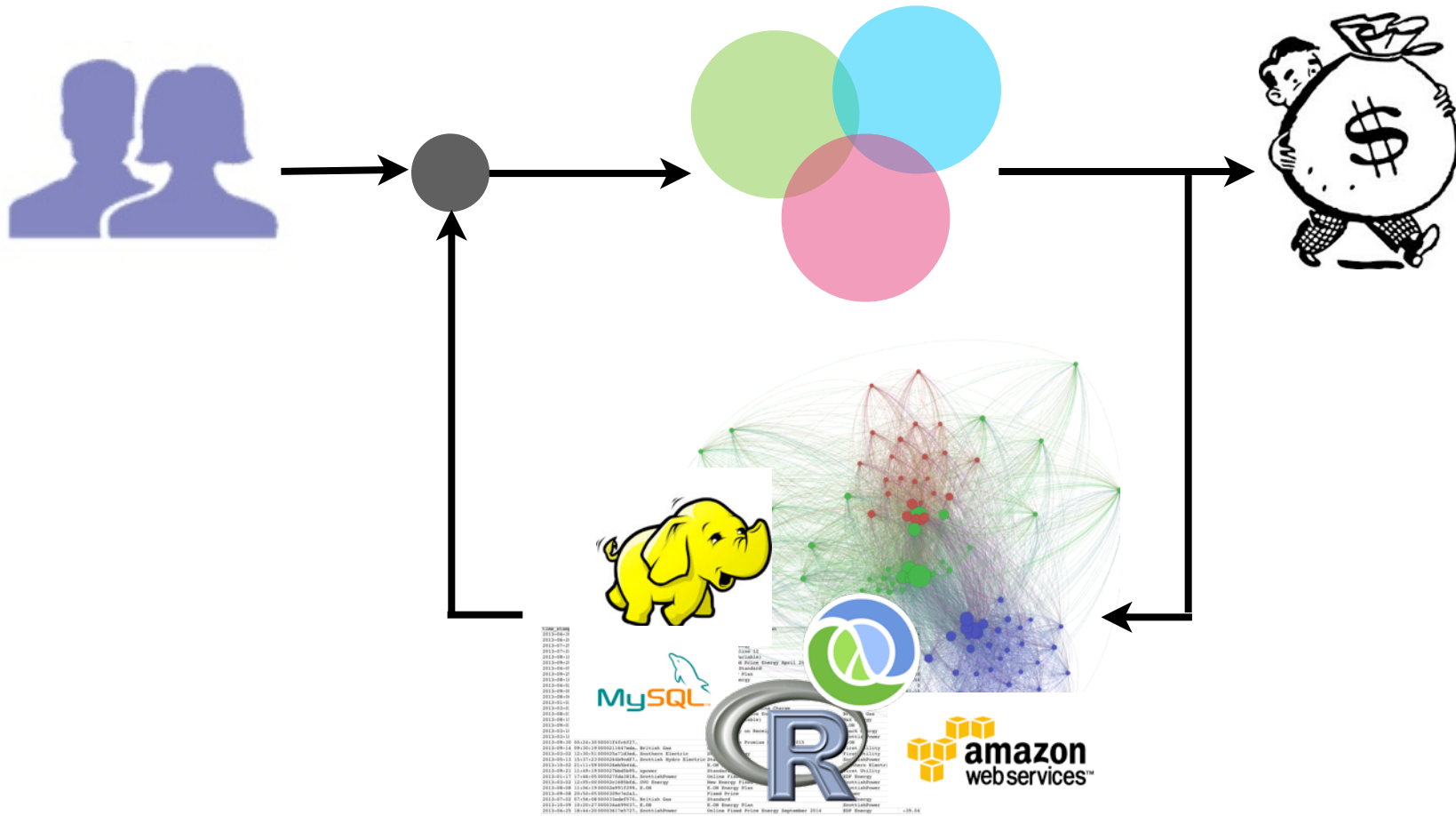
Minimise effort between Q & A

Question

Answer

Data Interrogation
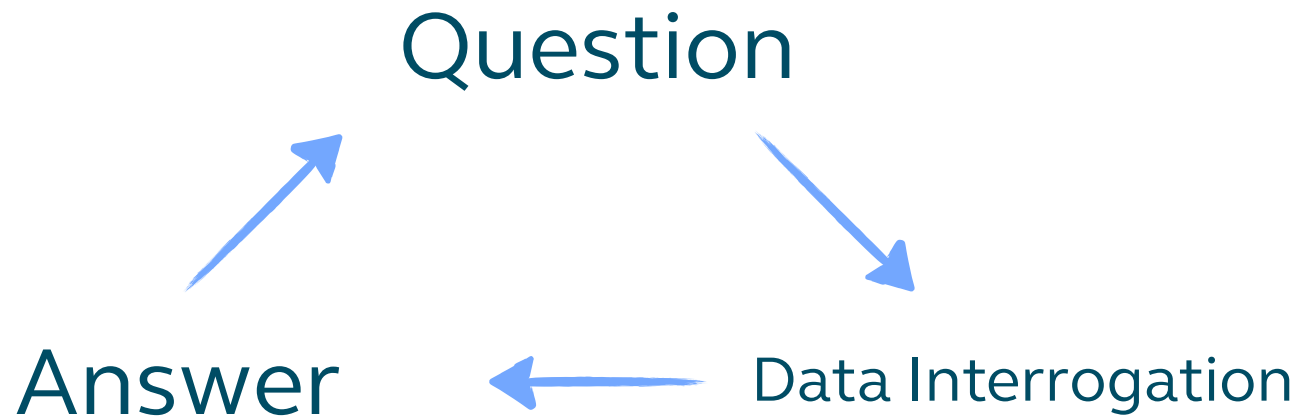
One Approach: Make data querying easier

Query = Function(Data) [1]

~ Function(Data Structure)

[1] Figure 1.3 from *Big Data* (preview v11) by Nathan Marz and James Warren

Data Structure: Relations versus Relations

aka **Edges**

Sale ID	User ID	Product ID	Profit
1	1	1	£100
2	1	2	£50
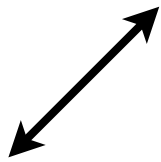
User ID	Name
1	Bob
2	Emily

Product ID	Name
1	iPhone
2	A.E.

{:profit £100}

{:profit £50}
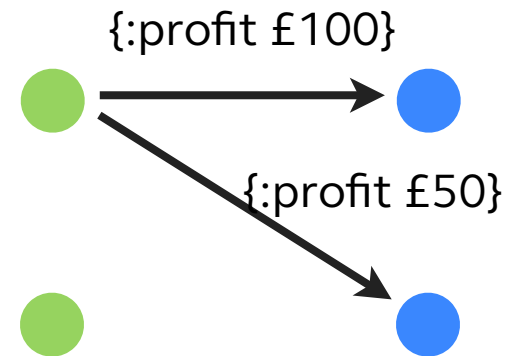
Using the right database for the right task

	RDBMS	Graph DB
Data	Attributes	Entities and relations
Model	Record-based	Associative
Relation	By-product of normalisation	First class citizen
Example use	Reporting	Reasoning

How does it work

User actions as time-stamped records

time

HDFS

Paul Ingles, "User as Data", Euroclojure 2012
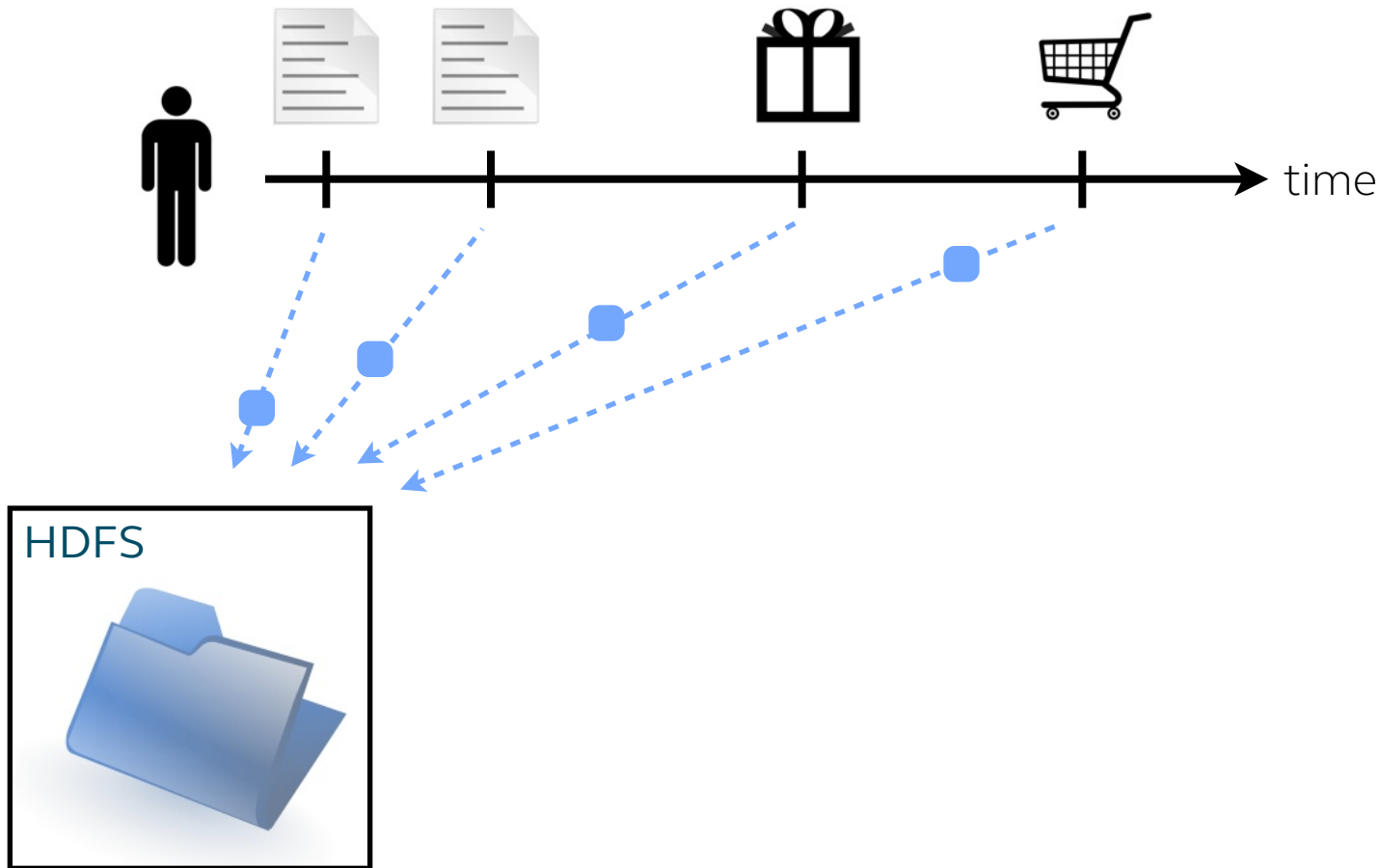
Our User Event to Graph Data Pipeline
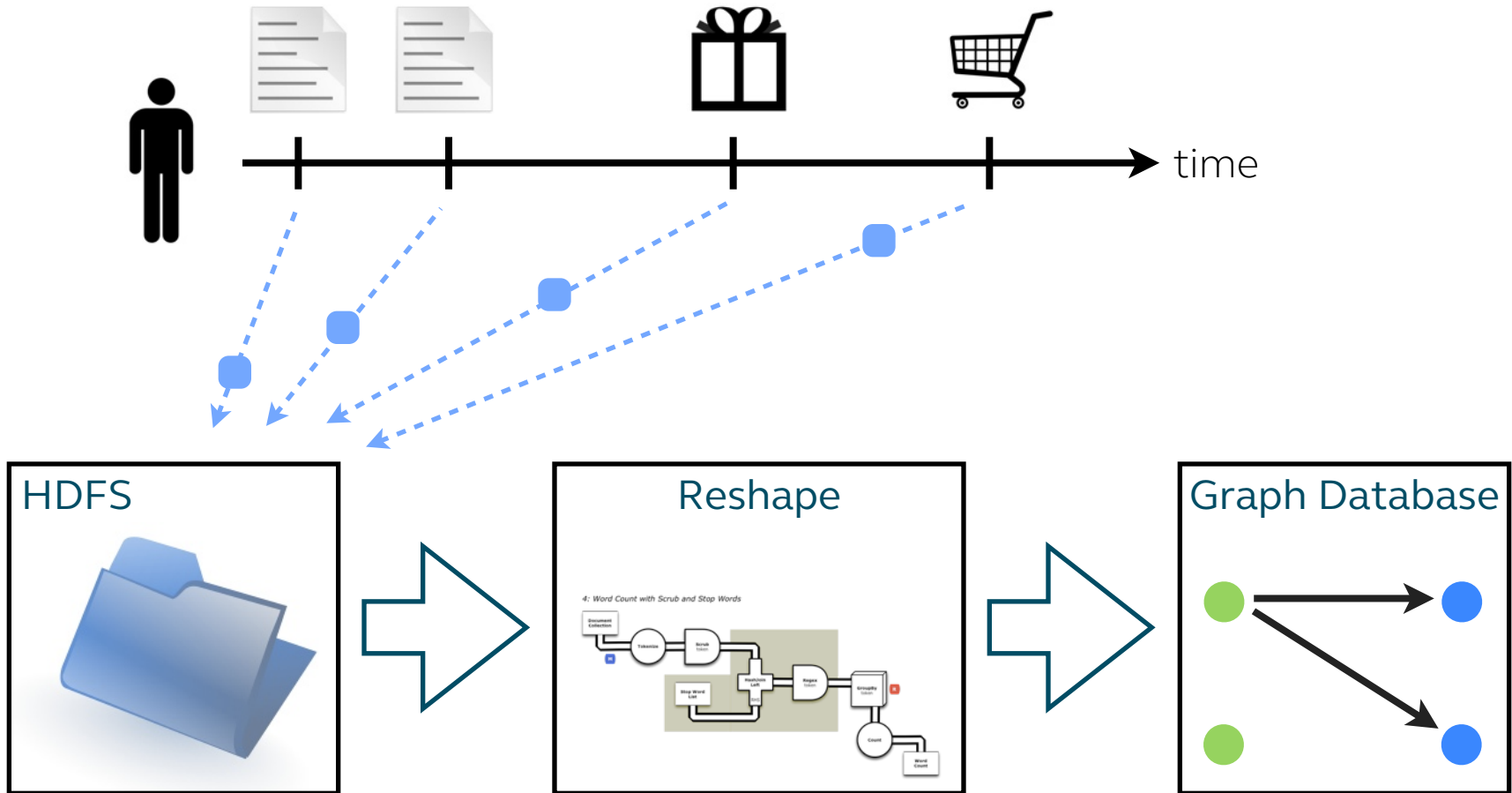
From Input to Output with Hadoop, aka ETL step

1. interface

2. input data

3. output data

4... The 80%

HDFS

Reshape

4: Word Count with Scrub and Stop Words

Graph Database
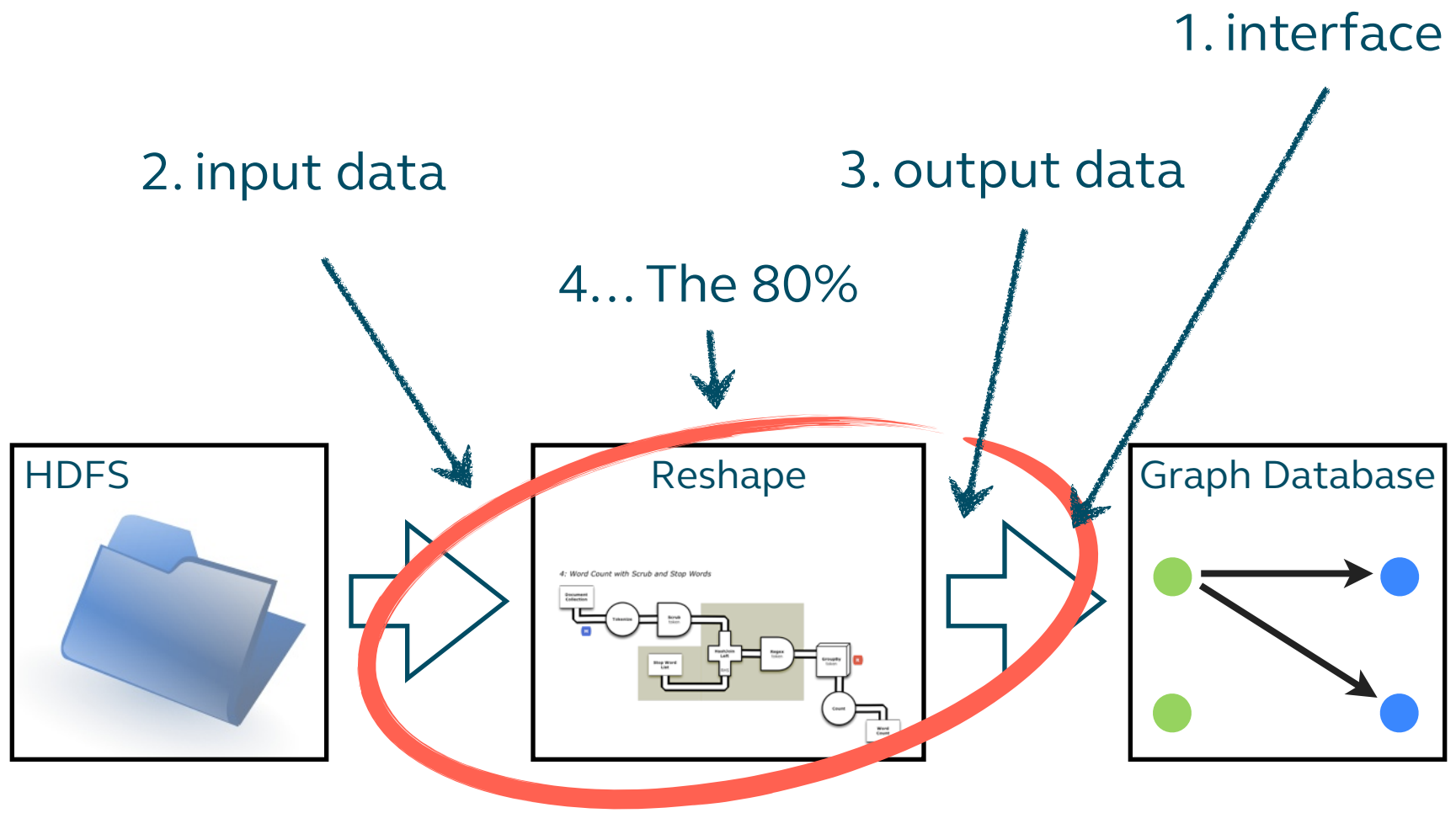
Hadoop interface to Neo4J

- Cascading-Neo4j tap [1]
- Faunus Hadoop binaries [2]
- CSV files*
- etc.

[1] http://github.com/pingles/cascading.neo4j
[2] http://thinkaurelius.github.io/faunus/
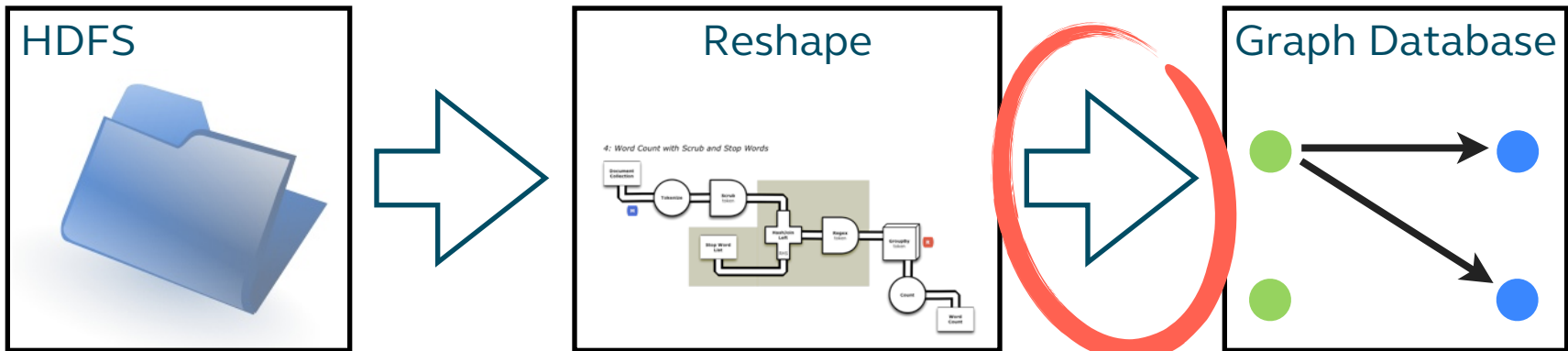
Input data stored on HDFS

User	Timestamp	Viewed Page	Referrer
Paul	2013-11-01 13:00	/homepage/	google.com
Paul	2013-11-01 13:01	/blog/	/homepage/

User	Timestamp	Viewed Product	Price	Referrer
Paul	2013-11-01 13:04	iPhone	£500	/blog/

User	Timestamp	Purchased	Paid	Attrib.
Paul	2013-11-01 13:05	iPhone	£500	google.com

User	Landed	Referral	Email
Paul	2013-11-01 13:00	google.com	paul.lam@uswitch.com

Nodes and Edges CSVs to go into a property graph

Node ID	Properties
1	{:name "Paul", email: "paul.lam@uswitch.com"}
2	{:domain "google.com"}
3	{:page "/homepage/"}
…	…
5	{:product "iPhone"}

From	To	Type	Properties
1	2	:SOURCE	{:timestamp "2013-11-01 13:00"}
1	3	:VIEWED	{:timestamp "2013-11-01 13:00"}
…	…	…	…
1	5	:BOUGHT	{:timestamp "2013-11-01 13:05"}

Records to Graph in 3 Steps

^ *importable CSV*

1. Design graph

2. Extract Nodes

3. Build Relations

Step 1: Designing your graph

{:name "Paul"
 :email "paul.lam@uswitch.com"}

:viewed {:timestamp … }

:bought {:timestamp … }

:viewed {…}

{:page "homepage"}

{:product "iPhone"}

:viewed {…}

{:page "blog"}

:source {:timestamp … }

{:domain "google.com"}
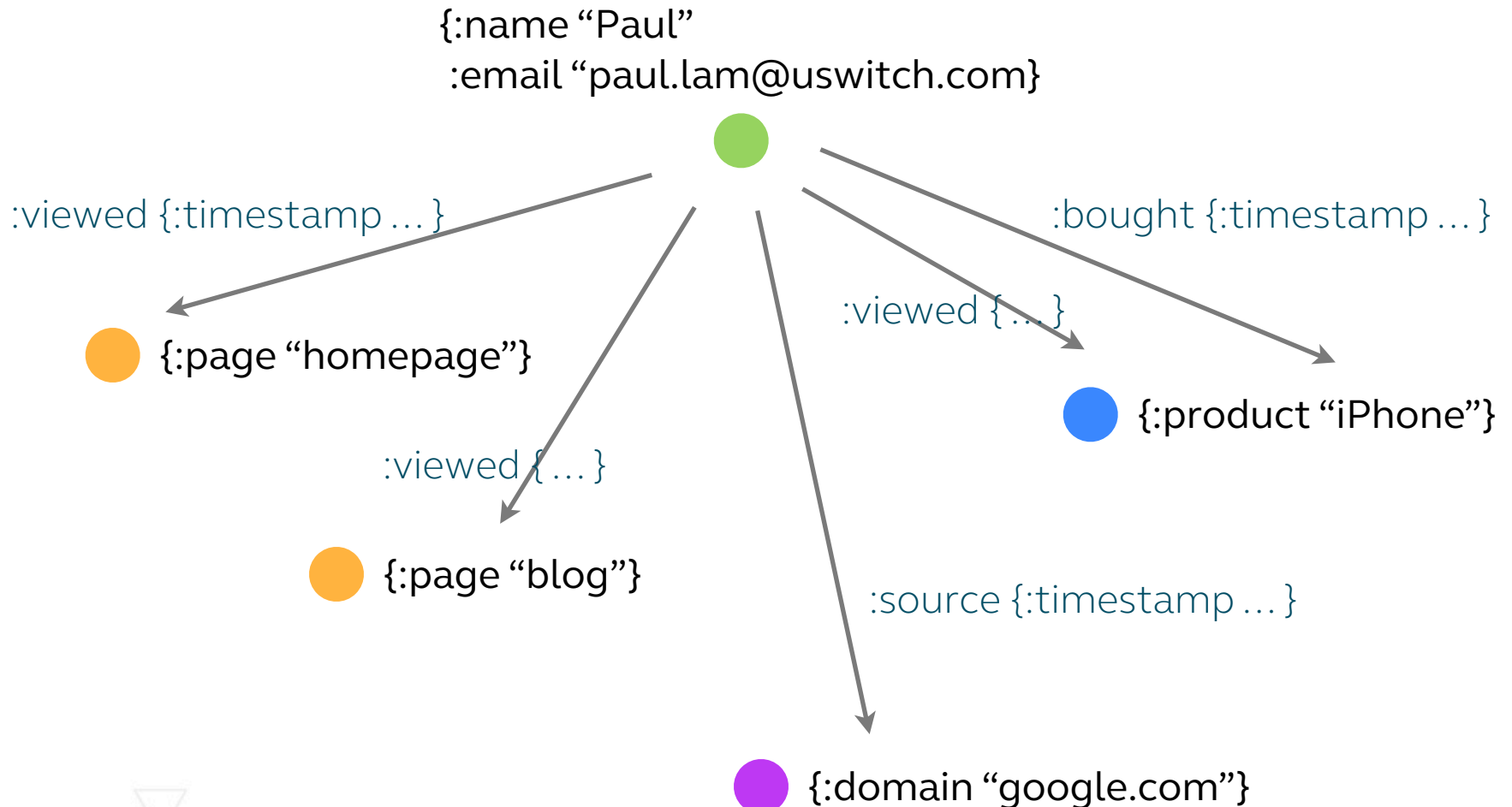
Step 2: Extract list of entity nodes

User	Timestamp	Viewed Page	Referrer
Paul	2013-11-01 13:00	/homepage/	google.com
Paul	2013-11-01 13:01	/blog/	/homepage/

User	Timestamp	Viewed Product	Price	Referrer
Paul	2013-11-01 13:04	iPhone	£500	/blog/

User	Timestamp	Purchased	Paid	Attrib.
Paul	2013-11-01 13:05	iPhone	£500	google.com

User	Landed	Referral	Email
Paul	2013-11-01 13:00	google.com	paul.lam@uswitch.com

Step 3: Building node-to-node relations

User	Timestamp	Viewed Page	Referrer
Paul	2013-11-01 13:00	/homepage/	google.com
Paul	2013-11-01 13:01	/blog/	/homepage/

User	Timestamp	Viewed Product	Price	Referrer
Paul	2013-11-01 13:04	iPhone	£500	/blog/

User	Timestamp	Purchased	Paid	Attrib.
Paul	2013-11-01 13:05	iPhone	£500	google.com

User	Landed	Referral	Email
Paul	2013-11-01 13:00	google.com	paul.lam@uswitch.com

Do this across all customers and products

Use your data processing tool of choice:

- Apache Hive
- Apache Pig
- Cascading   and more …
    - Scalding
    - Cascalog
- Spark
- your favourite programming language

Paco Nathan, "The Workflow Abstraction", Strata SC, 2013.

Cascalog code to build user nodes

- 145 lines of Cascalog code in production
- a couple hundred lines more of utility functions
- build entity nodes and meta nodes
- sink data into database with Cascading-Neo4j Tap

```
(def cc-visit-uscc-src
  "Users in credit cards nodes"
  (let [email-src (newest-record-only (named-select
                                        emails-generator
                                        ["?timestamp" "?uscc" "?email" "!opt-in"])
                                       ["?uscc"] ["?email" "!opt-in"])
        dob-src   (newest-record-only (named-select
                                        date-of-birth-entered-generator
                                        ["?timestamp" "?uscc" "?age"])
                                       ["?uscc"] ["?age"])]
    (<- (set-vars-type cc-visit-uscc-fields (var-kwd :unground) "!email" "!optin" "!age")
        ((select-fields staged-web-log-generator ["?uscc" "?request-path"]) ?uscc ?requestpath)
        (email-src _ ?uscc !!email !!optin)
        (dob-src _ ?uscc !!age)
        (match? [#"^/credit-cards.*"] ?requestpath)
        (:distinct true))))
```

Cascalog code to build user nodes

- 145 lines of Cascalog code in production
- a couple hundred lines more of utility functions
- build entity nodes and meta nodes
- sink data into database with Cascading-Neo4j Tap

```clojure
(def cc-visit-uscc-src
  "Users in credit cards nodes"
  (let [email-src (newest-record-only (named-select
                                        emails-generator
                                        ["?timestamp" "?uscc" "?email" "!opt-in"])
                                        ["?uscc"] ["?email" "!opt-in"])
        dob-src   (newest-record-only (named-select
                                        date-of-birth-entered-generator
                                        ["?timestamp" "?uscc" "?age"])
                                        ["?uscc"] ["?age"])]
    (<- (set-vars-type cc-visit-uscc-fields (var-kwd :unground) "!email" "!optin" "!age")
        ((select-fields staged-web-log-generator ["?uscc" "?request-path"]) ?uscc ?requestpath)
        (email-src _ ?uscc !!email !!optin)
        (dob-src _ ?uscc !!age)
        (match? [#"^/credit-cards.*"] ?requestpath)
        (:distinct true))))
```

Code to build user to product click relations

- 160 lines of Cascalog code in production
- + utility functions
- build direct and categoric relations
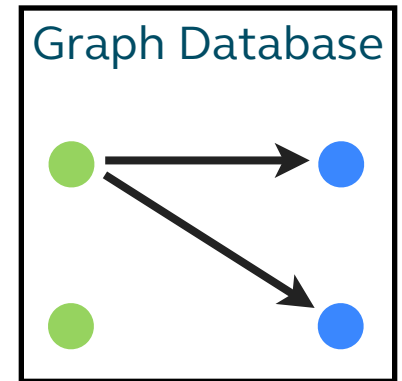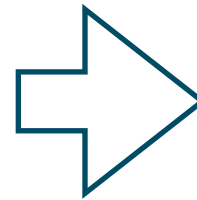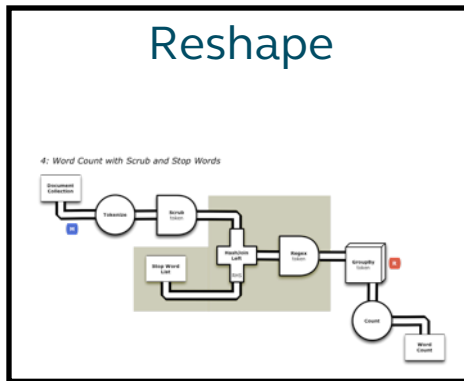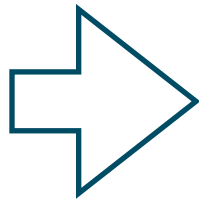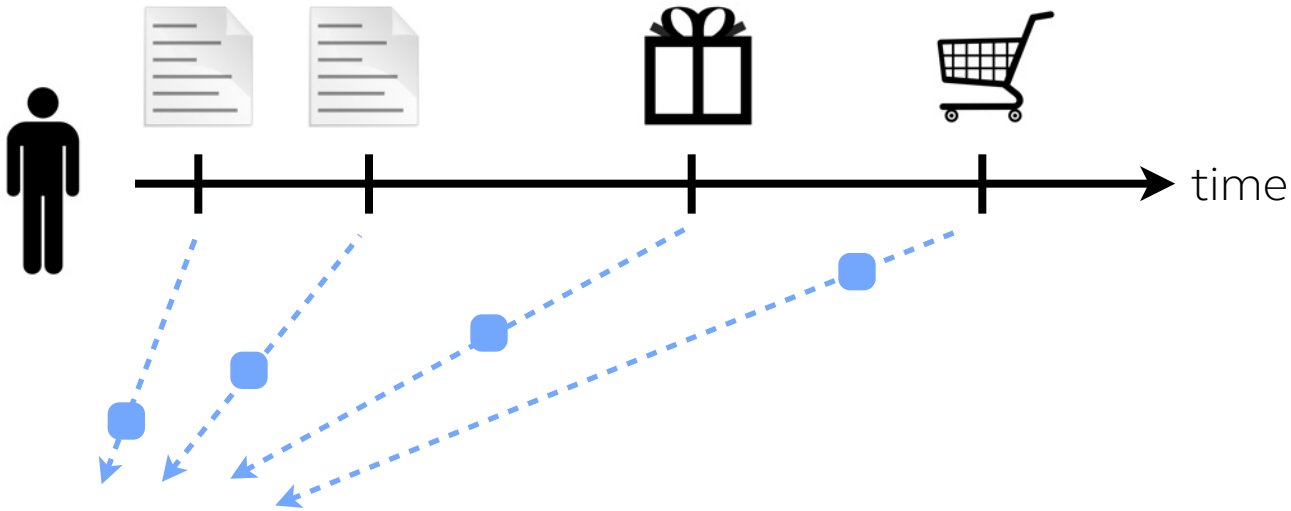- sink data with Cascading-Neo4j Tap

```
(defn cc-clicked-rel-query
  "Link user to product clicked"
  [clickthrough-src product-src users-src]
  (<- cc-clicked-rel-fields
      (clickthrough-src
          ?timestamp ?uscc ?uniqueid !category !position)
      (product-src :>> credit-cards-fields)
      (users-src :>> cc-visit-uscc-fields)
      (identity "CLICKED" :> ?type)
      (:distinct true)))
```

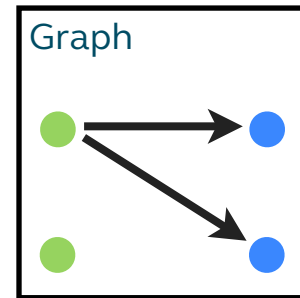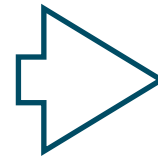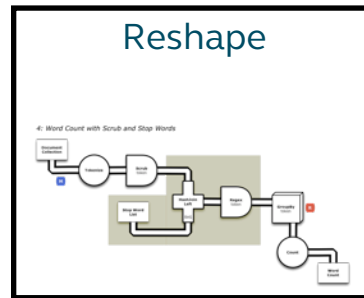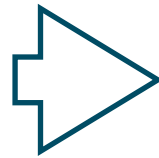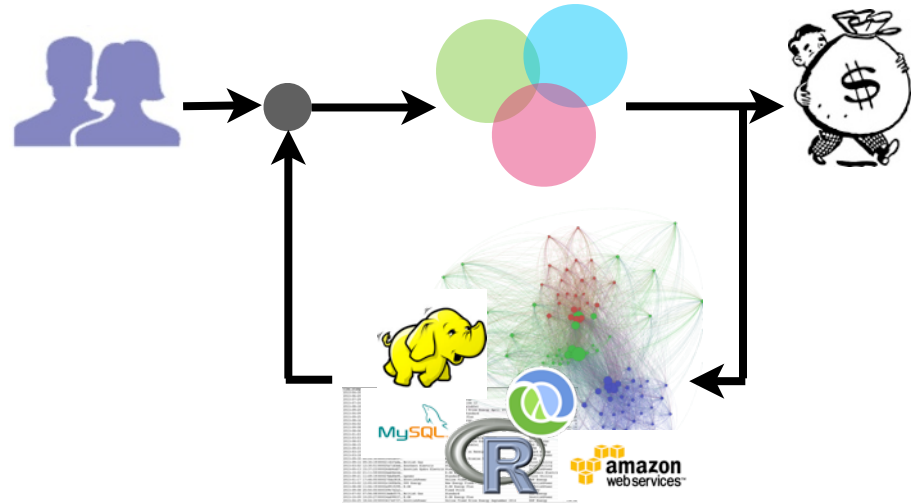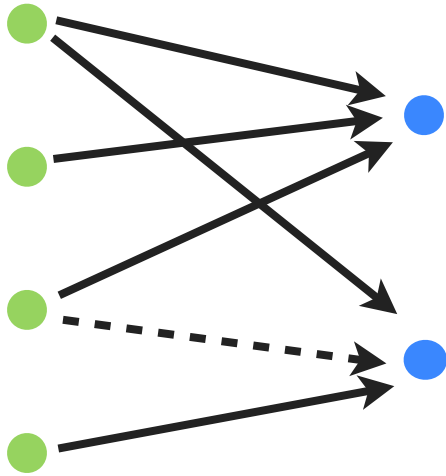# Code to build user to product click relations

- 160 lines of Cascalog code in production
- + utility functions
- build direct and categoric relations
- sink data with Cascading-Neo4j Tap

```
(defn cc-clicked-rel-query
  "Link user to product clicked"
  [clickthrough-src product-src users-src]
  (<- cc-clicked-rel-fields
      (clickthrough-src
        ?timestamp ?uscc ?uniqueid !category !position)
      (product-src :>> credit-cards-fields)
      (users-src :>> cc-visit-uscc-fields)
      (identity "CLICKED" :> ?type)
      (:distinct true)))
```

Our User Event to Graph Data Pipeline

HDFS

Reshape

Graph Database

time

Summary

HDFS

Reshape

4: Word Count with Scrub and Stop Words

Graph

Contact

Paul Lam, data scientist at uSwitch.com

Email: paul@quantisan.com

Twitter: @Quantisan