The Approximate Filter, Join, and GroupBy

David Rodriguez, Scott Sitar, Dhia Mahjoub
Cisco Systems

About Us

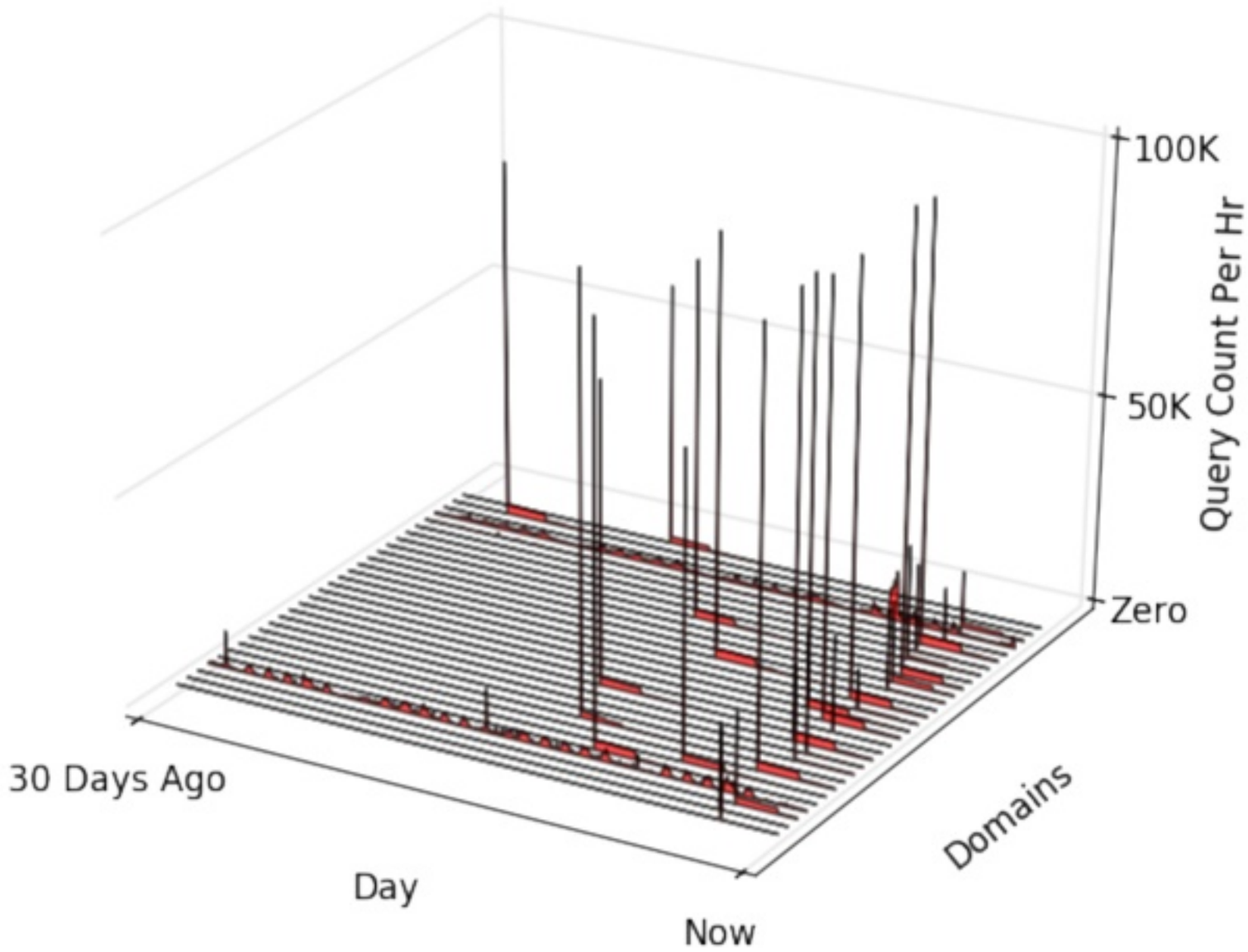
- David Rodriguez [davrodr3@cisco.com, @dvidrdgz]
  Data Scientist / Security Researcher. MA Mathematics.

- Scott Sitar [ssitar@cisco.com]
  Engineer Tech Lead. Hadoop Based Technologies.
  Mathematics.

- Dhia Mahjoub [dmahjoub@cisco.com, @dhialite]
  Head of Security Research for Cisco Umbrella. PhD.
  Mathematics.

One Problem

Motivating Problem: group similar spikes



"Key, simple idea : Partition data into buckets by hashing, then analyze."

Michael Mitzenmacher

Some Uses of Hashing in Networking Problems

Overview

Outline

- Primitives Filter, Join, GroupBy

  examples: clickstream and cpu usage

- Non-Primitive Filter, Join, GroupBy

  examples: signal clustering

- Locality Sensitive Hashing (LSH)

  examples: cosine measure (minhash if time permits)

- Application: Botnet Detection

Primitives

Clickstream Counts

```
public static void main(String[] args) throws Exception {

    // set up the execution environment
    final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

    // fake clickstream
    DataSet<String> text = env.fromElements("http://www.google.de/login",
            "http://www.google.de/search?q=flink+forward",
            "http://www.google.de/search?q=simple+string",
            "http://www.google.de/search?q=simple+count"
    );

    DataSet<Tuple2<String, Integer>> counts =
            // extract words embedded in url
            text.flatMap(new URLExtractor())
                    // group over words extracted
                    .groupBy(0)
                    .sum(1);

    // execute and print result
    counts.print();
}
```

- Identify most frequent words occurring in clickstreams

Questions

```
public static final class URLExtractor implements FlatMapFunction<String, Tuple2<String, Integer>> {

    @Override
    public void flatMap(String value, Collector<Tuple2<String, Integer>> out) {

        String[] tokens = value.toLowerCase().split("(\\/|\\+|\\=|\\?|\\:)");

        for (String token : tokens) {
            if (token.length() > 0) {
                out.collect(new Tuple2<String, Integer>(token, 1));
            }
        }
    }
}
```

- If this URLExtractor is used, what is the output of the previous program?

Challenges

```
// fake clickstream
DataSet<String> text = env.fromElements("http://www.google.de/login",
         "http://www.google.de/search?q=flink+forward",
         “http://www.google.de/search?q=simple+string",
         “http://www.google.de/search?q=simple+count",
         “https://berlin.flink-forward.org/kb_sessions“
);
```

- Group similar browsing history: e.g. *google searches*.
- In General, we need an unsupervised method of detecting similar browsing patterns.

CPU Usage

```
public static void main(String[] args) throws Exception {

    // set up the execution environment
    final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

    // fake cpu usage
    DataSet<Integer> text = env.fromElements(1, 10, 100, 20, 15,
            20000, 20001, 10001, 1000, 99, 98, 10, 2
    );

    DataSet<Tuple2<Integer, Integer>> counts =
            // discretize values
            text.flatMap(new Histogram())
                    // group over buckets in histogram
                    .groupBy(0)
                    .sum(1);

    // execute and print result
    counts.print();
}
```

- Identify most frequent buckets occurring in cpu usage

Questions

```
public static final class Histogram implements FlatMapFunction<Integer,
        Tuple2<Integer, Integer>> {

    @Override
    public void flatMap(Integer value, Collector<Tuple2<Integer, Integer>>
            out) {

        Double value_log = Math.log10(value);
        Integer bucket =(int)Math.floor(value_log);

        out.collect(new Tuple2<Integer, Integer>(bucket, 1));
    }
}
```

- If this Histogram is used, what is the output of the previous program?

Challenges

```
// fake cpu usage
DataSet<Tuple2> text3 = env.fromElements(
        new Tuple2(10, 0),
        new Tuple2(10, 1),
        new Tuple2(5, 5),
        new Tuple2(4, 5)
);
```

- Group multivariate signal readings.
- In General, we need an unsupervised method of detecting similar readings.

Approximate

Signal Clustering

```
public static void main(String[] args) throws Exception {

    // set up the execution environment
    final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

    // gen fake signals with event name
    DataSet<Event> text = env.fromElements(
            new Event("flinker1", new Signal(100, 0, 100)),
            new Event("flinker2", new Signal(100, 0, 101)),
            new Event("flinker3", new Signal(50, 0, 50)),
            new Event("unflinker1", new Signal(100, 0, 50)));

    DataSet<OutPreview> counts =
            // create LSH hash by random hyperplanes
            text.flatMap(new RandHyperplanes())
                    // group by hash signature
                    .groupBy(0)
                    .sum(2);

    // execute and print result
    counts.print();
}
```

- Identify similar signals: similar wave forms possibly dilated or contracted

Questions

```
// gen fake signals with event name
DataSet<Event> text = env.fromElements(
        new Event("flinker1", new Signal(100, 0, 100)),
        new Event("flinker2", new Signal(100, 0, 101)),
        new Event("flinker3", new Signal(50, 0, 50)),
        new Event("unflinker1", new Signal(100, 0, 50)));
```

- Are (100, 0, 100) and (100, 0, 101) similar?
- Are (100, 0, 100) and (50, 0, 50) similar?
- Are (100, 0, 100) and (100, 0, 50) similar?
- In General, we need an unsupervised method of detecting similar readings.

Locality Sensitive Hashing - LSH

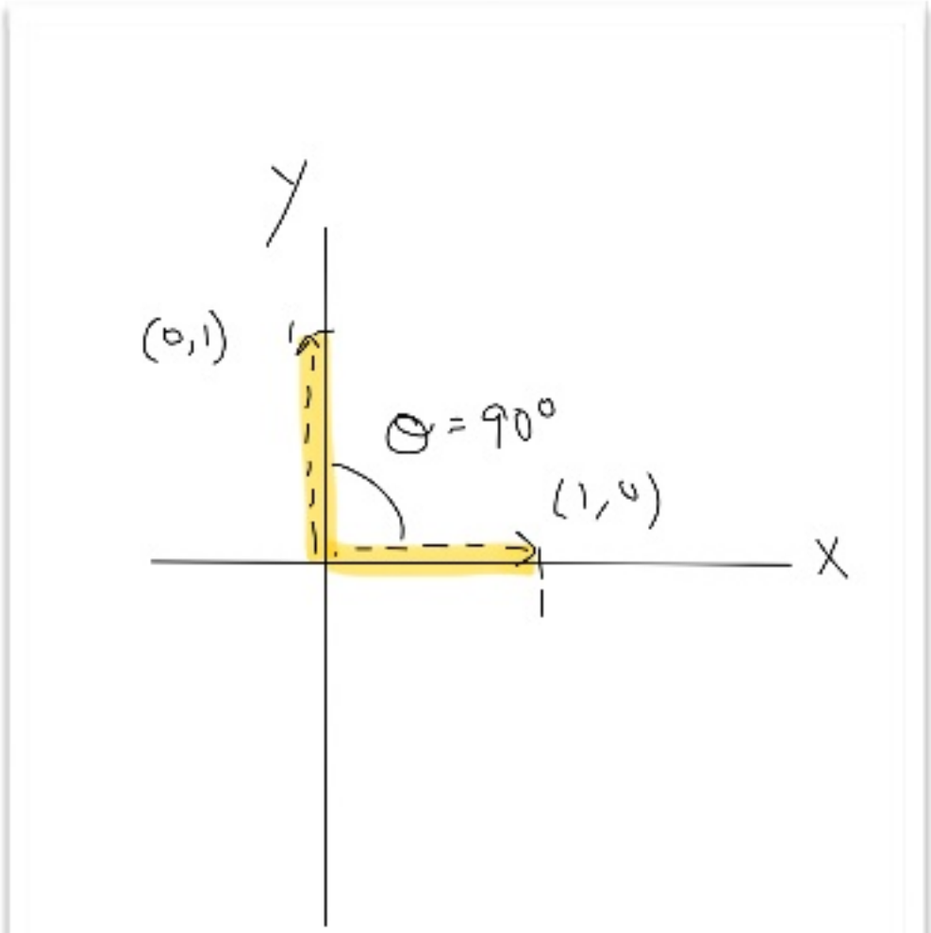
Cosine Similarity

- Given two vectors:
  (1, 0) = x and (1, 0) = y

- Define angle between and from calculus:

  theta = arccos(x \dot y / (||x|| * ||y||))

- theta is a distance measure:
  1. d(x,x) = 0
  2. d(x,y) = d(y,x)
  3. d(x,y) >= 0 w/ 0 <= theta <= 180



Theta and Hashes

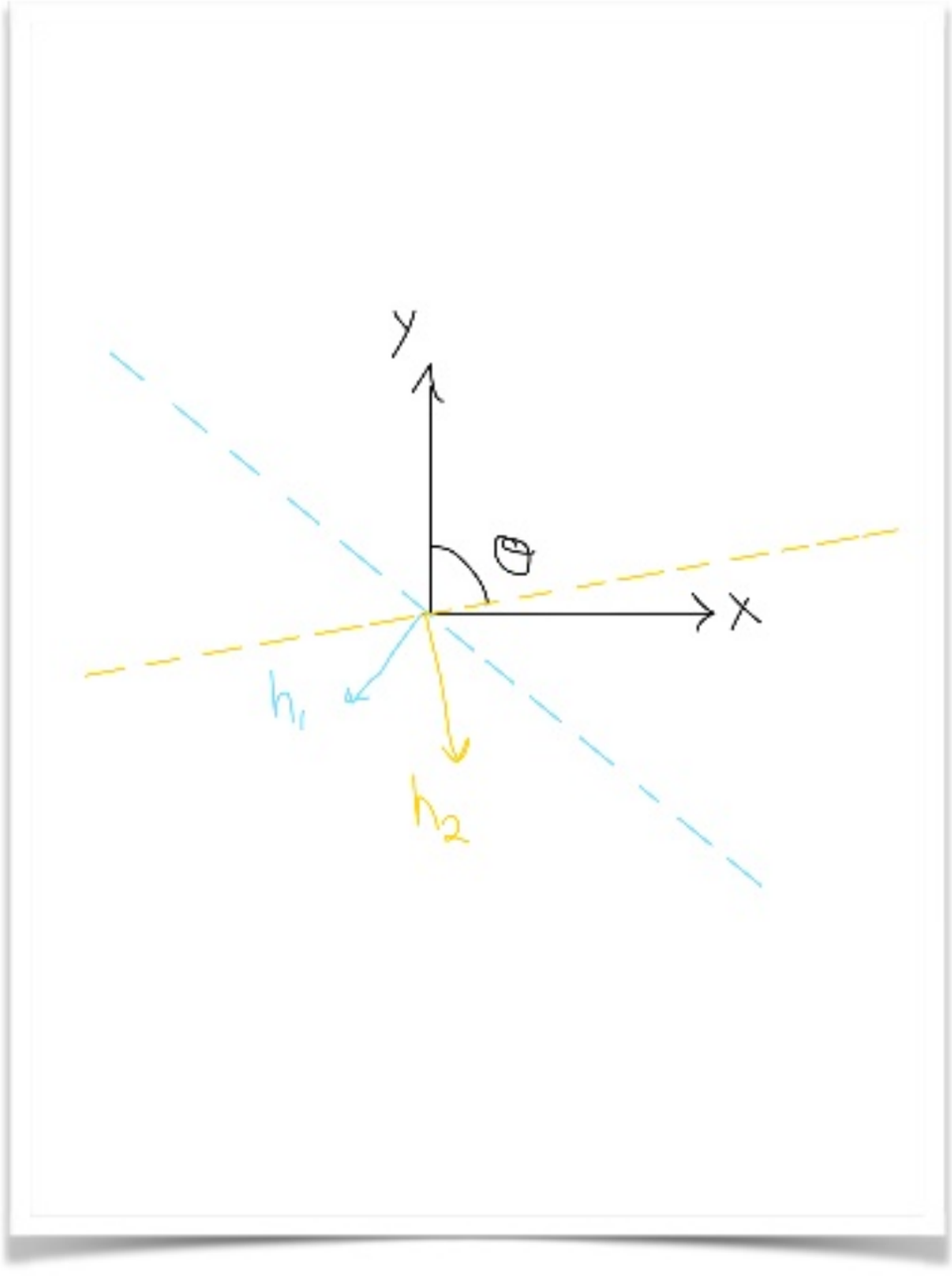
- Define a function f_1:

  0 , if sgn(h_1 \dot x) > 0

  1 , otherwise

- Then:

  P(f_1(x) = f_2(y)) = 1-\theta / 180

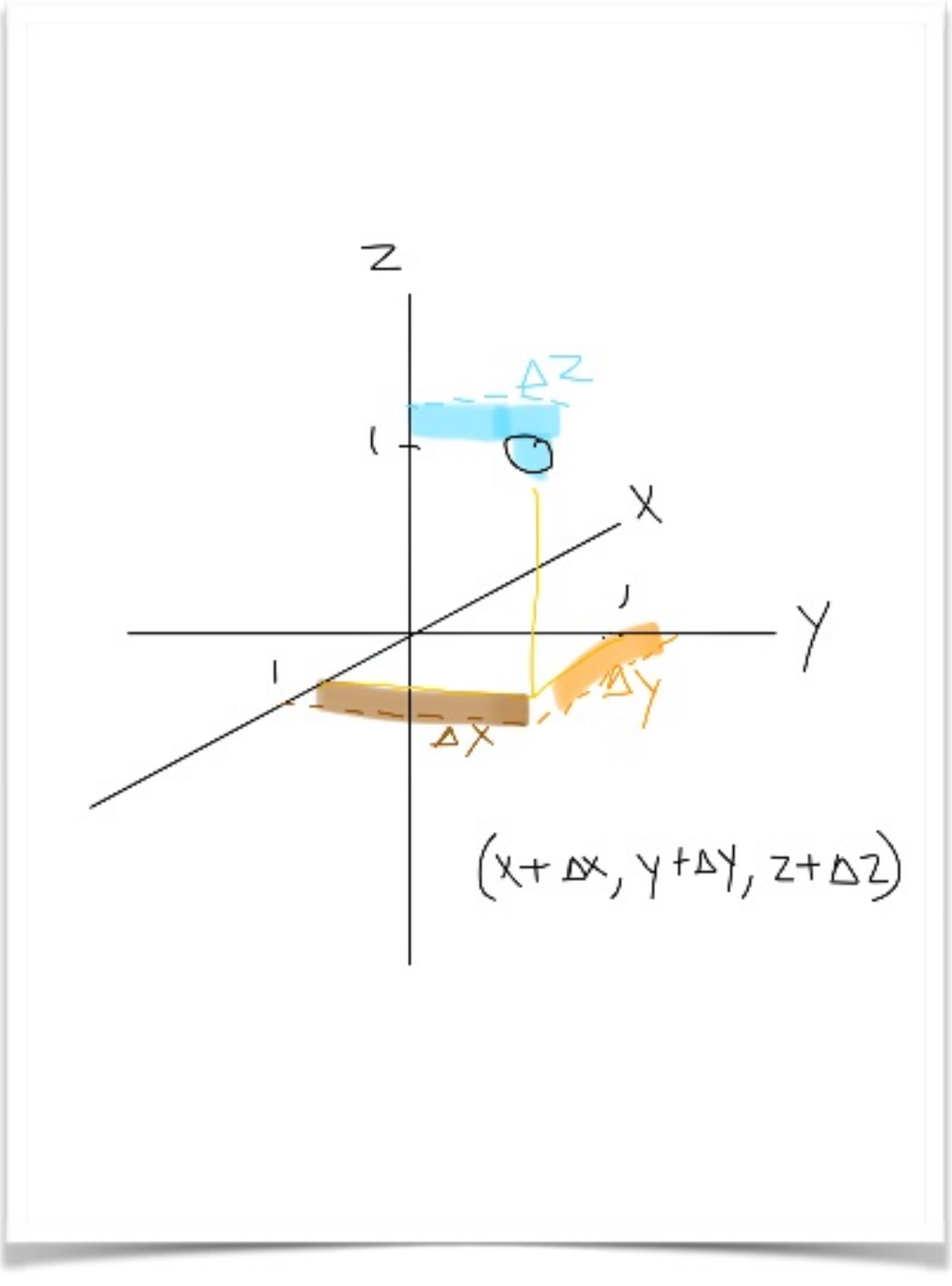


Theta and Deltas

- Are (100, 0, 100) and (100, 0, 101) similar?

- Are (100, 0, 100) and (50, 0, 50) similar?

- Are (100, 0, 100) and (100, 0, 50) similar?



p-Stable Hashes

- family H of hash functions is said to be (d_1, d_2, p_1, p_2) - sensitive

if for any x and y in a possible space of points

1. if d(x,y)<= d_1, then P(h(x)=h(y)) >= p_1
2. if d(x,y)>= d_2, then P(h(x)=h(y)) <= p_2

Finding the Number of Hyperplanes

- Let:

  (20°, 120°, 1 - 20°/180°, 1 - 120°/180°)

  for one hyperplane.

- We can determine the **number of hyperplanes**.

theta	(1-(1-p)^2)^2
20°	0.0440
40°	0.1560
60°	0.3086
80°	0.4779
100°	0.6439

Signal Clustering

Flink Main

```
public class ApproximateSignals {

    public static class Event {
        String name;
        Signal signal;

        public Event() {}

        public Event(String name, Signal signal) {
            this.name = name;
            this.signal = signal;
        }
    }

    public static class Signal extends Tuple3<Integer, Integer, Integer> {
        Signal(Integer x, Integer y, Integer z) {
            super(x, y, z);
        }
    }

    public static class OutPreview extends Tuple3<String, String, Integer> {
        OutPreview(String hash, String preview, Integer count) {
            super(hash, preview, count);
        }
    }

    public static void main(String[] args) throws Exception {

        // set up the execution environment
        final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

        // get input data
        DataSet<Event> text = env.fromElements(
                new Event("flinker1", new Signal(100, 0, 100)),
                new Event("flinker2", new Signal(100, 0, 101)),
                new Event("flinker3", new Signal(50, 0, 50)),
                new Event("unflinker2", new Signal(100, 0, 50)));

        DataSet<OutPreview> counts =
                // create LSH hash by random hyperplanes
                text.flatMap(new RandHyperplanes())
                        // group by hash signature
                        .groupBy(0)
                        .sum(2);
        // execute and print result
        counts.print();
    }

    public static final class RandHyperplanes implements FlatMapFunction<Event,
            OutPreview> {

        @Override
        public void flatMap(Event event, Collector<OutPreview> out) {
            String hash = HyperplaneSignature.matVProd(event.signal);
            String name = event.name;
            out.collect(new OutPreview(hash, name, 1));
        }
    }
}
```

Hyperplane Generation

```
public class HyperplaneSignature {
    static Integer[][] hyperplanes = {{-1, 1, 1},
            {-1, -1, 1},
            {1, 1, 1},
            {1, -1, 1},
            {-1, 1, -1},
            {1, -1, -1}};

    static Integer product(Integer[] w, Integer[] v) {
        Integer n = w.length;
        Integer sum = 0;
        for (int i = 0; i < n; i++) {
            sum += w[i] * v[i];
        }
        return sum;
    }

    static String sign(Integer value) {
        if (value < 0.0) {
            return "a";
        } else {
            return "b";
        }
    }

    public static String matVProd(Tuple3 v) {
        Integer[] vhat = {(Integer) v.f0, (Integer) v.f1, (Integer) v.f2};
        String signature = "";
        for (Integer[] hyperplane : hyperplanes) {
            Integer magnitude = product(hyperplane, vhat);
            signature += sign(magnitude);
        }
        return signature;
    }
}
```

Hyperplane Generation - *alternative*

```scala
import breeze.linalg._
import scala.util.Random


object LSH {

  val hyperplanes_n = 10
  val series_len = 3
  val hyperplanes = __initHyperplanes

  def __initHyperplanes: DenseMatrix[Double] = {
    val planes = (1 to hyperplanes_n).map(x =>
      (1 to series_len).map(y =>
        Random.nextGaussian()
      )
    )
    DenseMatrix(planes.map(_.toArray): _*)
  }

  def hash(volume: DenseVector[Double]): Array[Int] = {
    val mult = hyperplanes * volume

    mult
      .toArray
      .map({ x =>
        if (x > 0) 0 else 1
      })
  }
}
```

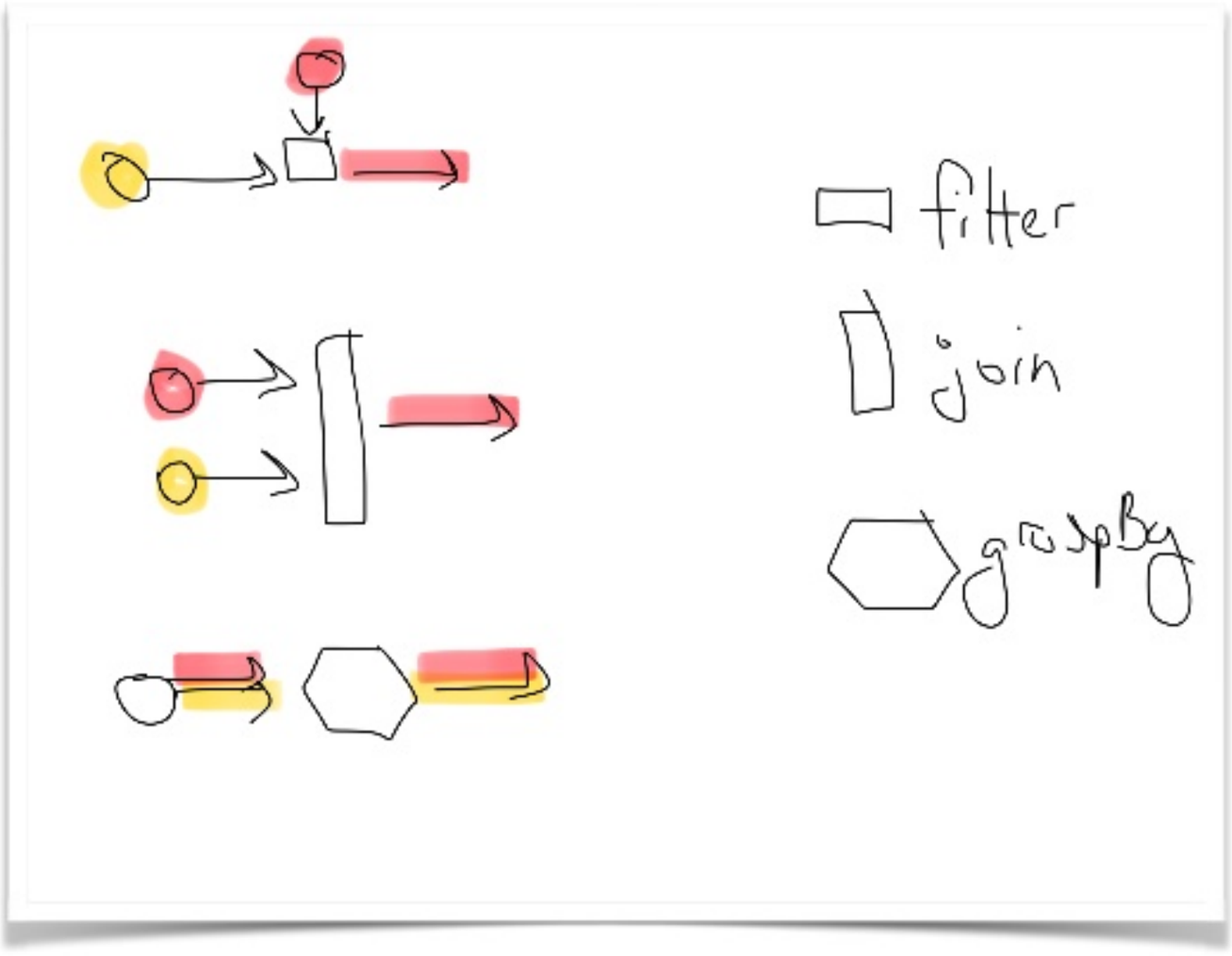
Signal Clusters

```
(aabbab,unflinker1,1)

(bbbbab,flinker3,3)


Program execution finished
Job with JobID 9658d0fb4037ec75e24b244479ef4b39 has finished.
Job Runtime: 469 ms
```

- Surprised?

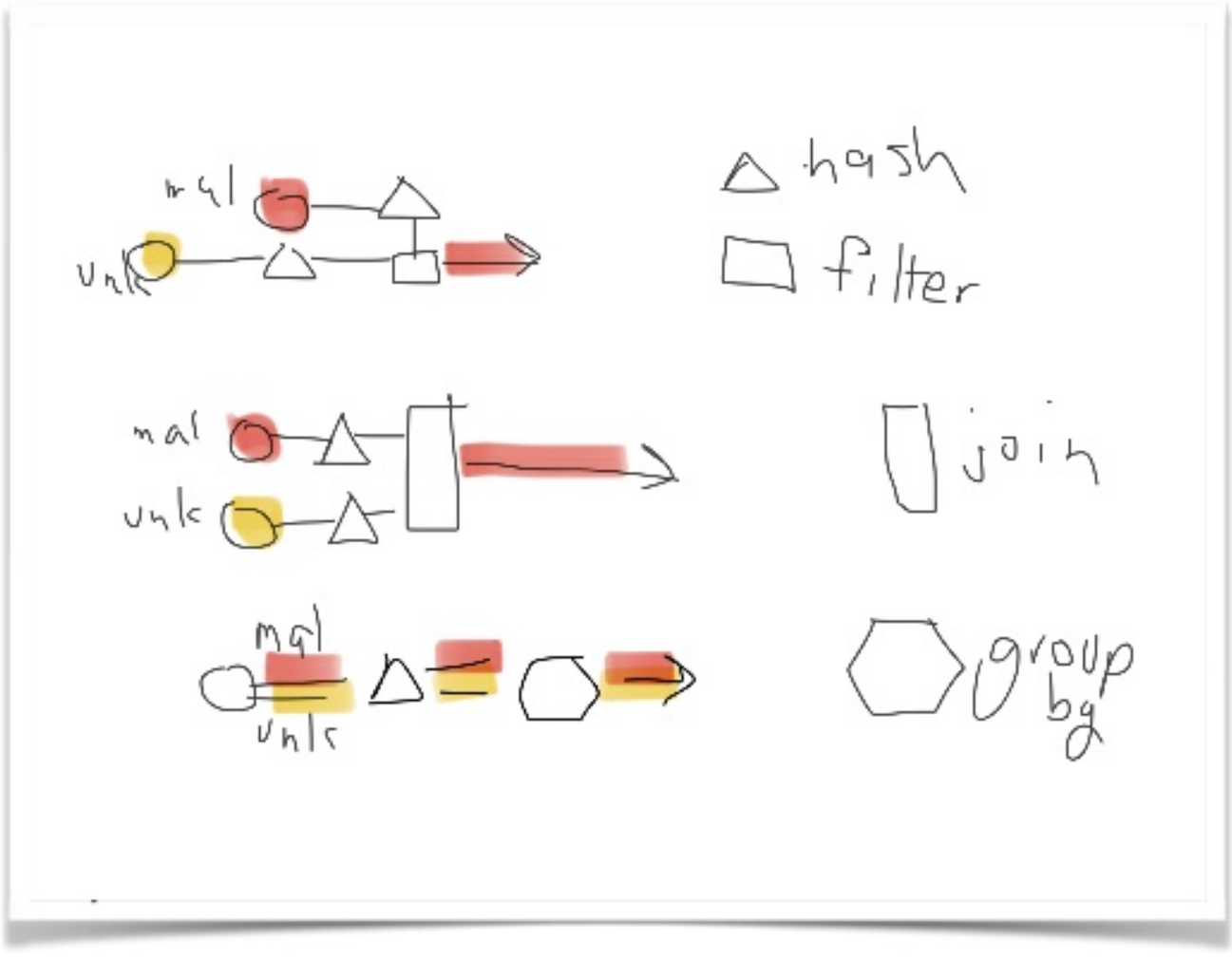
Pipelines

Traditional Pipelines



- Count events, group related items, etc.

Adversarial Pipelines



- Pattern recognition, similarity searching, cohort analysis

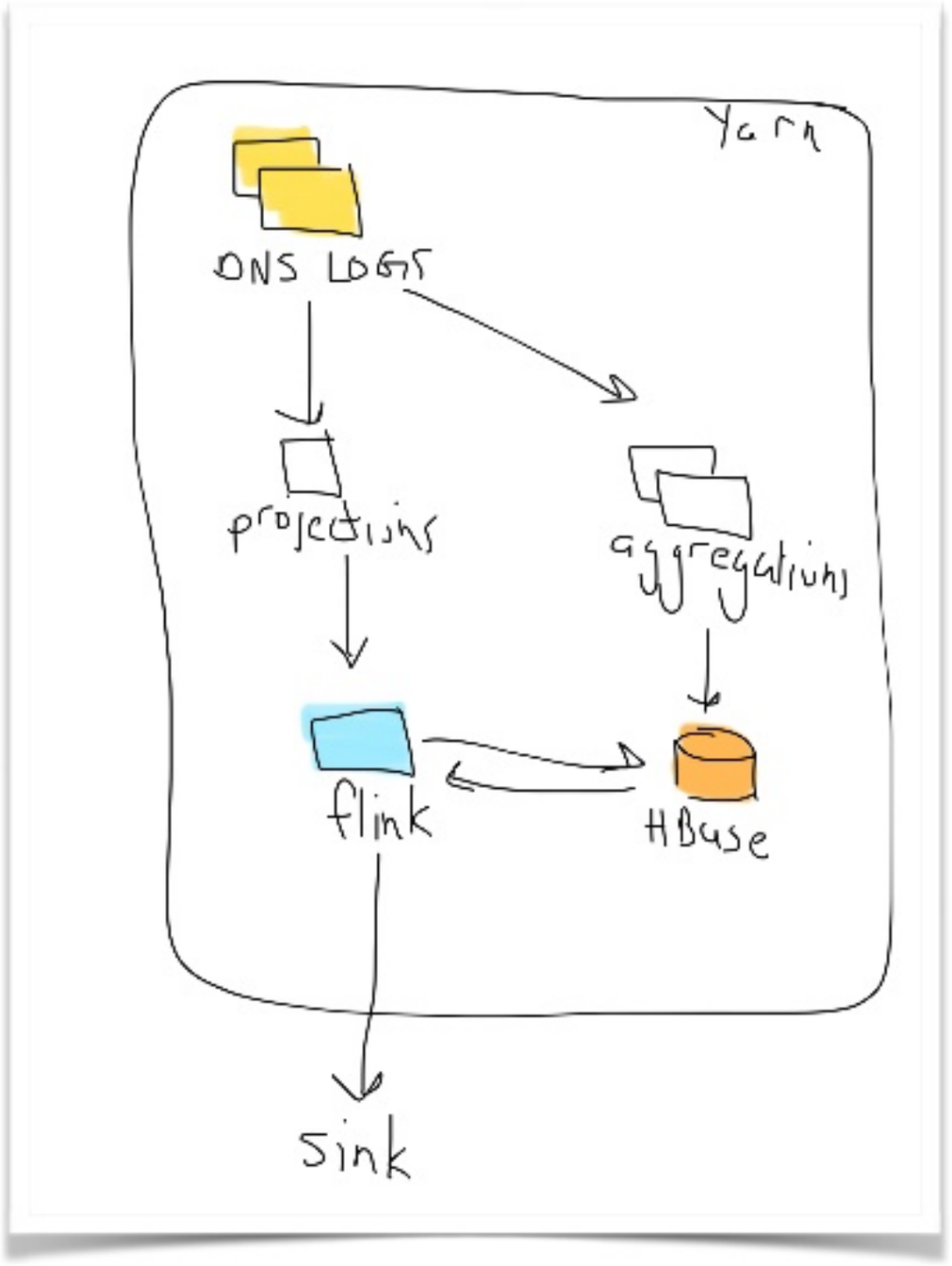
Adversarial Join

```
public class ApproximateArraysJoin {

    public static class Event {
        String name;
        Signal signal;

        public Event() {
        }

        public Event(String name, Signal signal) {
            this.name = name;
            this.signal = signal;
        }
    }

    public static class Signal extends Tuple3<Integer, Integer, Integer> {
        Signal(Integer x, Integer y, Integer z) {
            super(x, y, z);
        }
    }

    public static void main(String[] args) throws Exception {

        // set up the execution environment
        final StreamExecutionEnvironment env =
                StreamExecutionEnvironment.getExecutionEnvironment();

        DataStream<Event> control = env.fromElements(
                new Event("badflinker", new Signal(99, 0, 100)),
                new Event("badflinker!", new Signal(77, 0, 75)));

        // get input data
        DataStream<Event> data = env.fromElements(
                new Event("flinker1", new Signal(100, 0, 100)),
                new Event("flinker2", new Signal(100, 0, 101)),
                new Event("flinker3", new Signal(50, 0, 50)),
                new Event("unflinker2", new Signal(100, 0, 50)));


        DataStream<String> result = control
                .broadcast()
                .connect(data)
                .flatMap(new RandHyperplaneCoFlatMap());
        result.print();
        env.execute();
    }

    public static final class RandHyperplaneCoFlatMap implements
            CoFlatMapFunction<Event, Event, String> {
        HashSet blacklist = new HashSet();

        @Override
        public void flatMap1(Event event, Collector<String> out) {
            String hash = HyperplaneSignature.matVProd(event.signal);
            blacklist.add(hash);
            out.collect("added to blacklist " + hash + " " + event.name);
        }

        @Override
        public void flatMap2(Event event, Collector<String> out) {
            String hash = HyperplaneSignature.matVProd(event.signal);
            if (blacklist.contains(hash)) {
                out.collect("alas! caught another bad one " + hash + " " + event.name);
            } else {
                out.collect("bummer! this isn't bad " + hash + " " + event.name);
            }
        }
    }
}
```

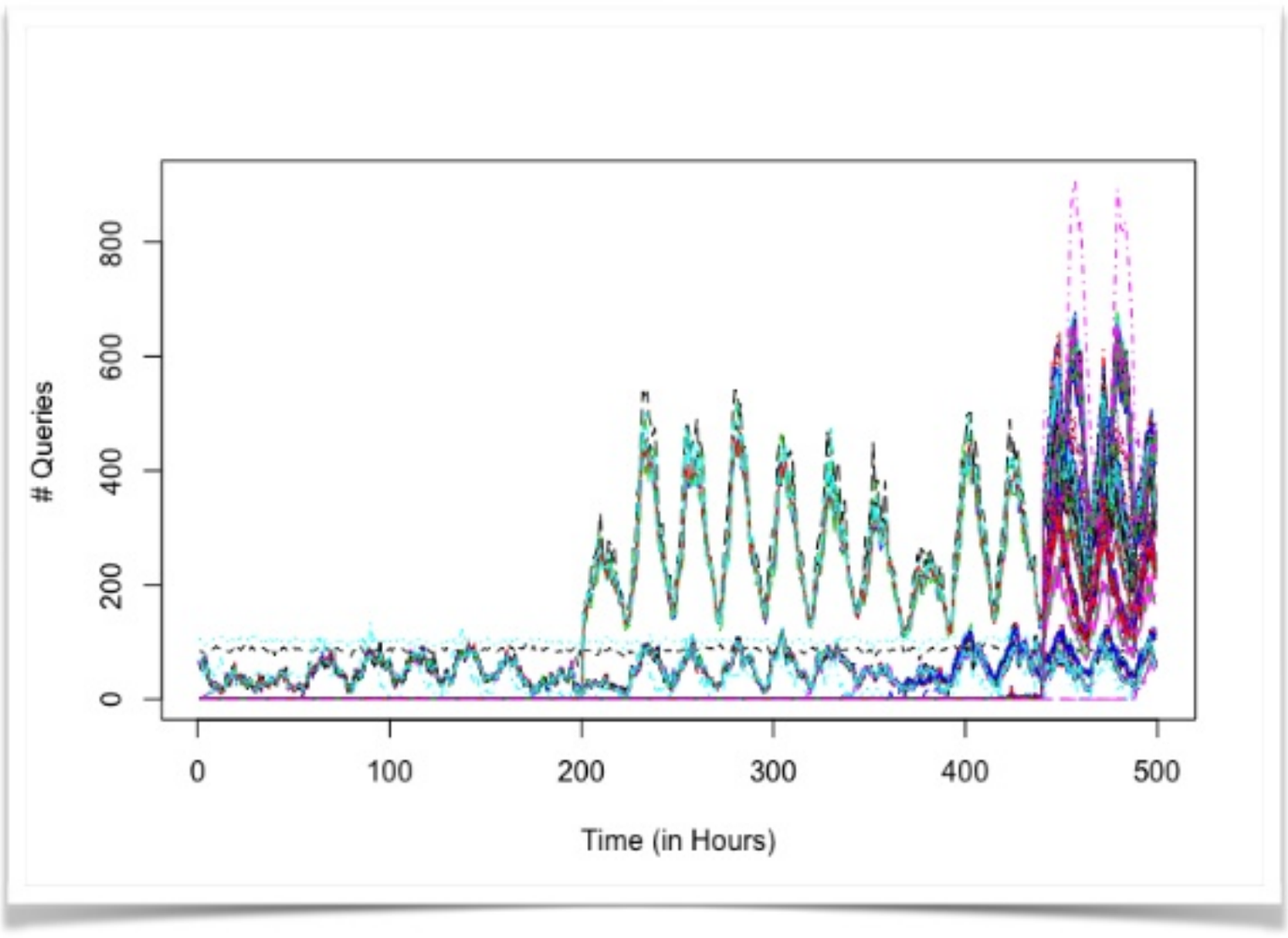
Application

Architecture

- Decouples & Plays nice with legacy MapReduce oozie workflows
- Projections enable passing:
    - User-Item Connected Components of Graphs
- Achieves
    - 5 Million + domains per batch
    - Fetching signals with 700+ readings
    - Subdomain aggregation
    - Cohort Analysis

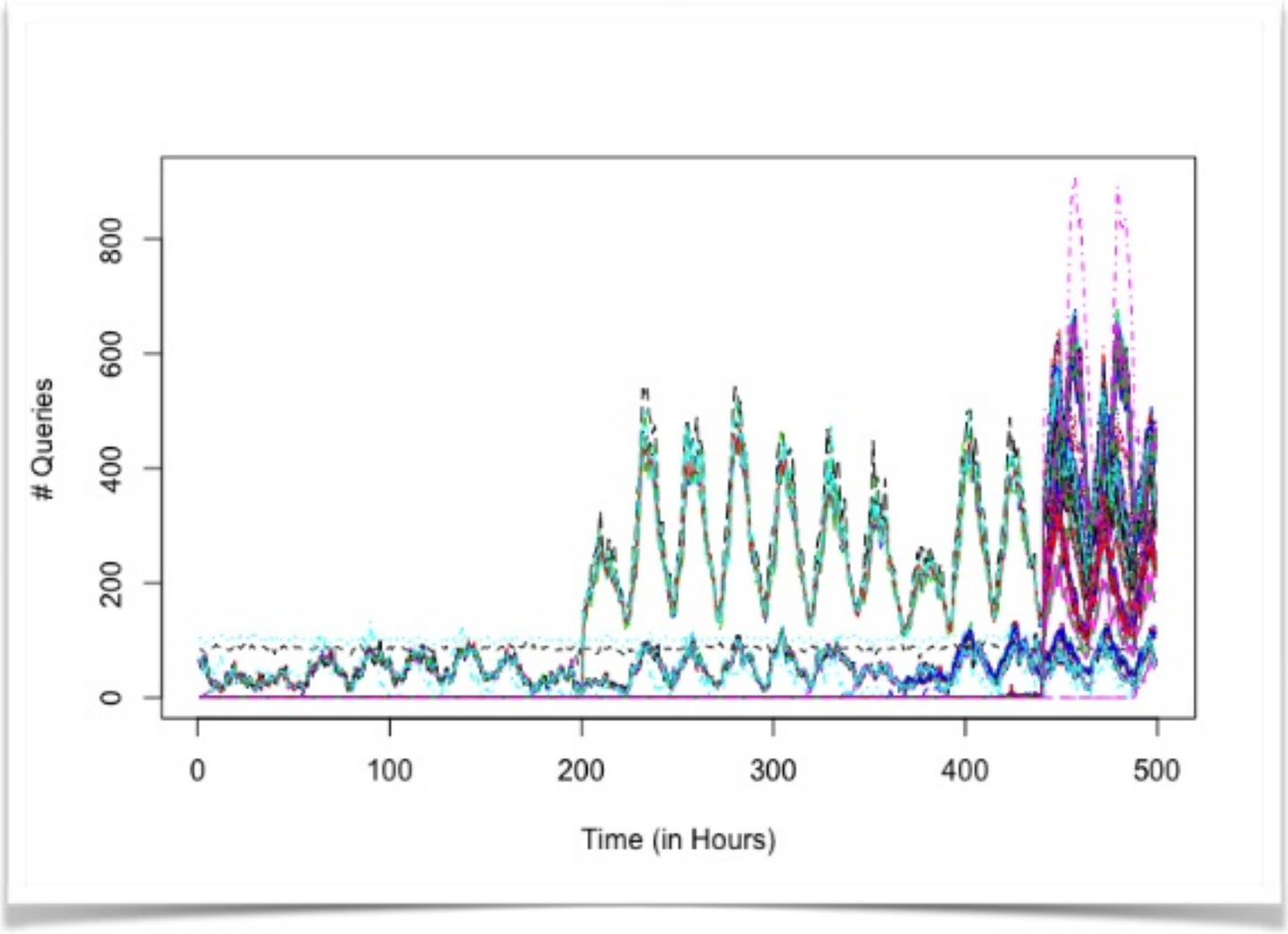


Detecting Botnets



- 200 Randomly sampled Algorithmically Generated Domains from: Pykspa, Necurs, Suppobox, Pushdo, Zeus Gameover.

Detecting Botnets



Pushdo

zyawafdeqer.kz
pokhuwyad.kz
pelzobdath.kz
zohevesvylu.kz
bosutymcure.kz

Necurs

qcakimrtsrarcqjcutxmb.cc
ltcolhdcvv.nf
memeugampjhdbnwhxxsa.nu
vexbklktsbfyuv.la
uyndateoeqshbeytpwwb.jp

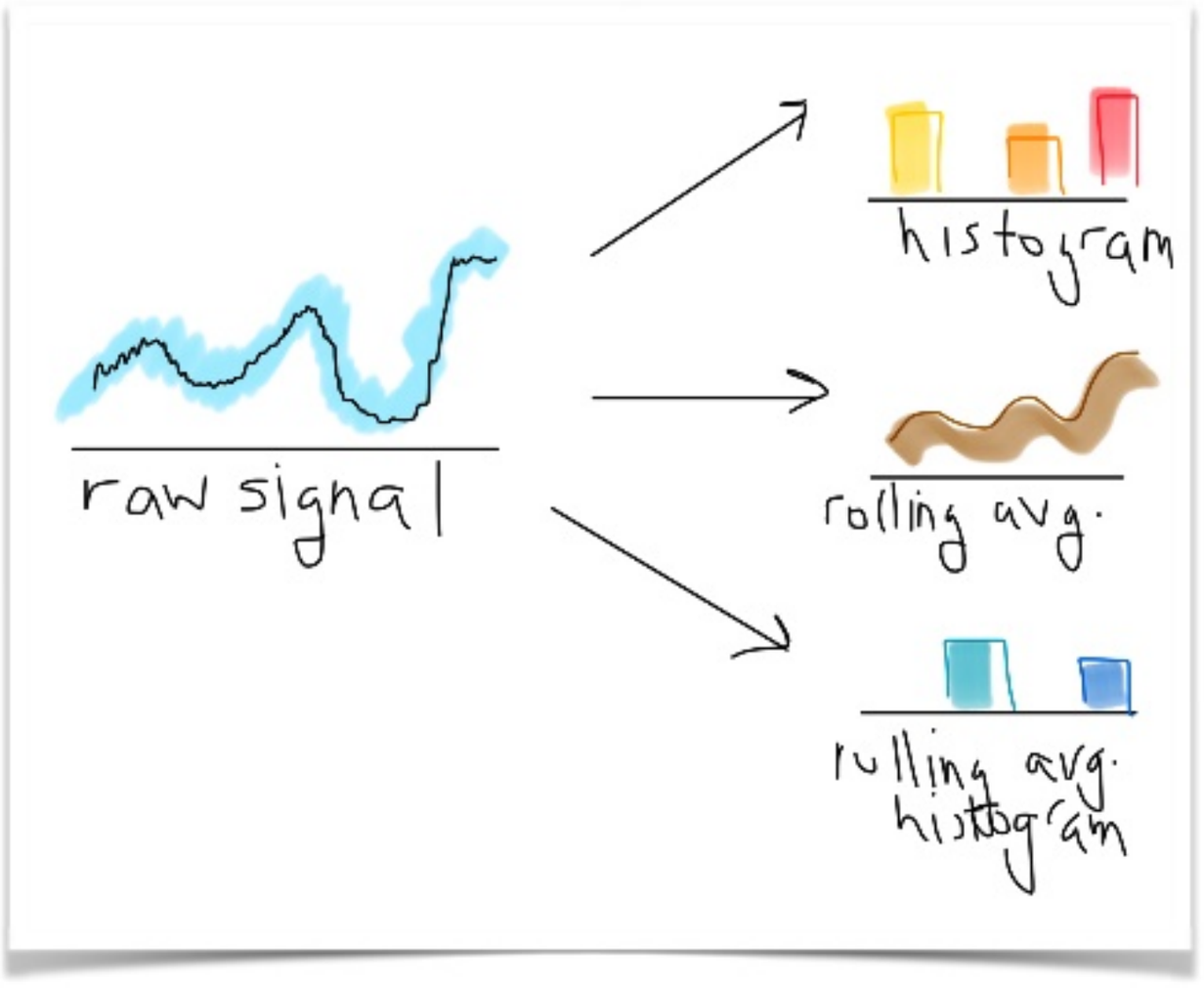
Gameover Zeus

xzaulqbqojdyzpdykzhojduhml.net
mjtyphaealldutxcdscnrh.org
dyugcqmzzhkduzplteabqgidfa.info
pjscugmuaynbfqwoqcqcfmhtfyo.net
ividkvrcfehoveavsljaelvro.com

Pykspa

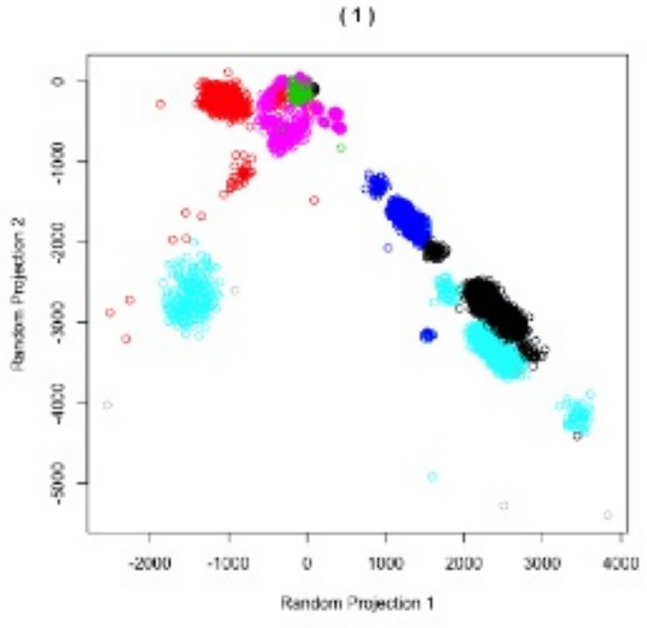
nemynulcnx.net
aszzfvzqb.net
oahebrlytur.net
nalznbmmhvn.com
lhdajsdenwb.com

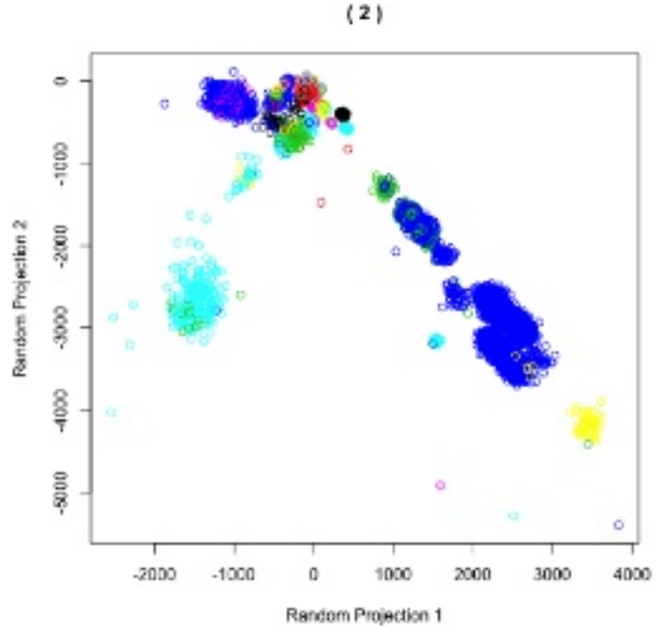
Signal Pre-Processing

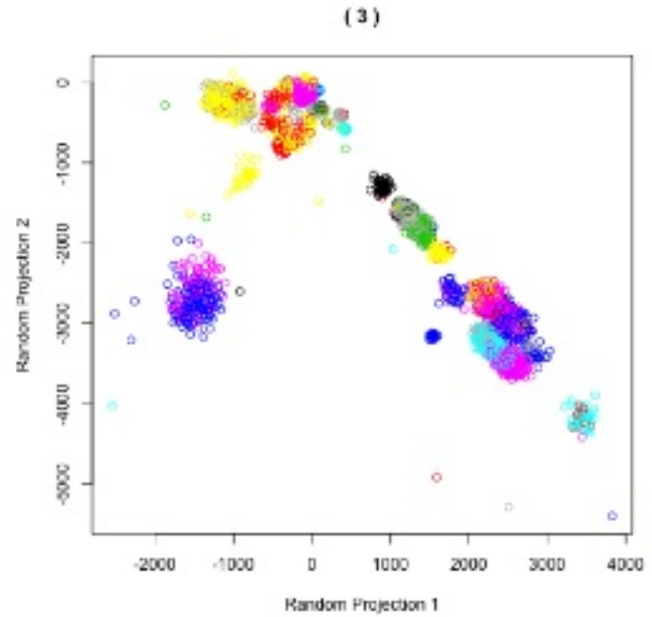


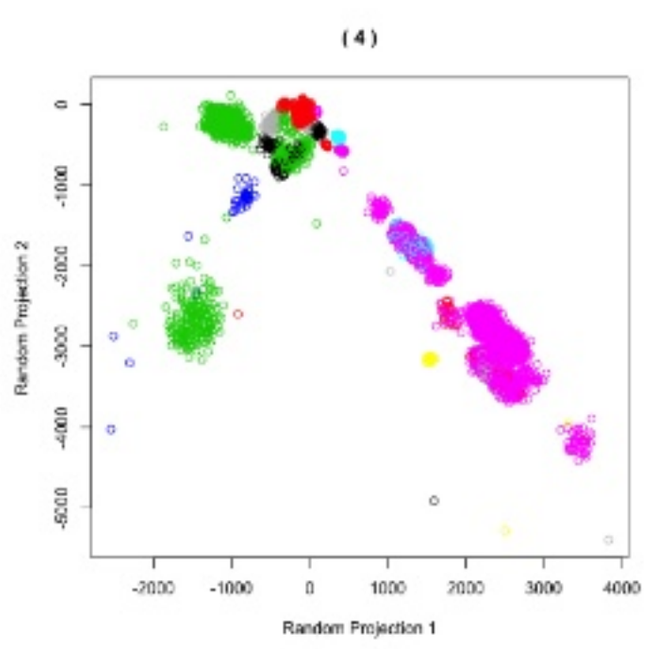
- The idea: test whether more compressed versions of the signals could be stored.

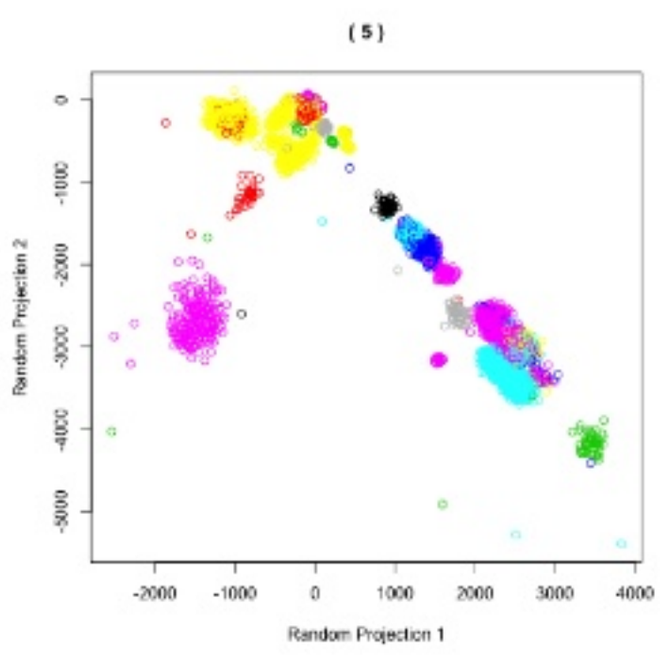
Detecting Botnets











- Assigning each of the 8,475 signals a cluster from different LSH & pre-processing techniques. We label then randomly project the 500-dimensional signals into 2-dimensions. The plots are as follows: (1) Hand labeled domains into 9 families, (2) raw-counts hashed then clusters formed, (3) histogram of signals hashed then clustered, (4) rolling-average of signals hashed then clustered, (5) rolling-average-histogram of signals hashed then clusters formed.

Recap

Outline: recap

- Primitives Filter, Join, GroupBy
  examples: clickstream and cpu usage

- Non-Primitive Filter, Join, GroupBy
  examples: signal clustering

- Locality Sensitive Hashing (LSH)
  examples: cosine measure

- Application: Botnet Detection

The End

Bibliography

Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA.

Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.

S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," Proc. Intl. Conf. on Data Engineering, 2006.

A.Z. Broder, M. Charikar, A.M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," ACM Symposium on Theory of Computing, pp. 327–336, 1998.

A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," Comm. ACM 51:1, pp. 117– 122, 2008.

Appendix

Flink Main

```
public class ApproximateStringsNew {
    public static class OutPreview extends Tuple3<String, String, Integer> {
        OutPreview(String hash, String preview, Integer count) {
            super(hash, preview, count);
        }
    }

    public static void main(String[] args) throws Exception {

        // set up the execution environment
        final ExecutionEnvironment env =
ExecutionEnvironment.getExecutionEnvironment();

        // get input data
        DataSet<String> text = env.fromElements(
                "http://www.google.de/login",
                "http://www.google.de/search?q=flink+forward",
                "http://www.google.de/search?q=simple+string",
                "http://www.google.de/search?q=simple+count"
        );

        DataSet<Tuple3<String, String, Integer>> counts =
                text.flatMap(new MinHasher())
                        .groupBy(0)
                        .sum(2);

        // execute and print result
        counts.print();
    }

    public static final class MinHasher implements FlatMapFunction<String,
            Tuple3<String, String, Integer>> {

        @Override
        public void flatMap(String value, Collector<Tuple3<String, String, Integer>>
                out) {
            // simple url word extractor
            String[] tokens = value.toLowerCase().split("(\\/|\\+|\\=|\\?|\\:)");
            String hash = MinHasherStatic.minHash(tokens);

            out.collect(new Tuple3<String, String, Integer>(hash,
                    value.toLowerCase(), 1));
        }
    }
}
```

Hyperplane Generation

```
public class MinHasherStatic {
    static int VOCAB_SIZE = 12;
    static int KHASHES = 2;

    static private List<Map<String, Integer>> kHashes = setup();

    static private List<Map<String, Integer>> setup() {
        Map<String, Integer> hm1 = new HashMap<>();
        hm1.put("http", 1);
        hm1.put("www", 7);
        hm1.put("google", 6);
        hm1.put("de", 5);
        hm1.put("login", 4);
        hm1.put("search", 2);
        hm1.put("q", 9);
        hm1.put("flink", 10);
        hm1.put("forward", 3);
        hm1.put("simple", 8);
        hm1.put("string", 11);
        hm1.put("count", 12);

        Map<String, Integer> hm2 = new HashMap<>();
        hm2.put("http", 8);
        hm2.put("www", 9);
        hm2.put("google", 12);
        hm2.put("de", 3);
        hm2.put("login", 4);
        hm2.put("search", 11);
        hm2.put("q", 7);
        hm2.put("flink", 10);
        hm2.put("forward", 5);
        hm2.put("simple", 1);
        hm2.put("string", 2);
        hm2.put("count", 6);

        List<Map<String, Integer>> hashes = new ArrayList<>();
        hashes.add(hm1);
        hashes.add(hm2);

        return hashes;
    }

    static String minHash(String[] words) {
        Integer[] mh = new Integer[KHASHES];
        Arrays.fill(mh, VOCAB_SIZE + 1);

       for (String word : words) {
            for (int i = 0; i < KHASHES; i++) {
                Map<String, Integer> m = kHashes.get(i);
                if (m != null){
                    if (m.containsKey(word)){
                        int h = m.get(word);
                        int c = mh[i];
                        if (h < c) {
                            mh[i] = h;
                        }
                    }

                }

            }
        }
        return Arrays.toString(mh);
    }
}
```

ClickStream Clusters

```
([1, 1],http://www.google.de/search?q=simple+count,2)

([1, 4],http://www.google.de/login,1)

([1, 5],http://www.google.de/search?q=flink+forward,1)

Program execution finished
Job with JobID 0f8fac5481bb2ccd1acd772dca0759cc has finished.
Job Runtime: 698 ms
```

- Surprised?