Processing “BIG-DATA” In Real Time

Yanai Franchi , Tikal


Happy
Birthday

Vacation to Barcelona

NON-END-RER-MU ONLY/DTE CHG M
YR150/RFD MYR300
KULKUL
08SEP07
18/LX6U20
IATA BSP
MSIAN HARMONY TOUR
N TRAVEL
KUALA LUMPUR /MY
20300626 /LL/U756/1

	KUALA LUMPUR	MU	5024N	20DEC	1645	OKYEE45			03FEB20K
O	SHANGHAI PUDONG	MU	5023N	06JAN	1000	OKYEE45			03FEB20K
	KUALA LUMPUR		VOID	VOID					
	TO --VOID--		VOID	VOID					
	TO --VOID--								

MYR 5776KUL MU SHA843.85MU KUL843.85NUC1687.70END ROE3.4224

51MY
42CN
314YQINVABT
KUL07010

MYR 6183
77657

1 781

Printed by IATA/ICAO, CHINA

After a Long Travel Day

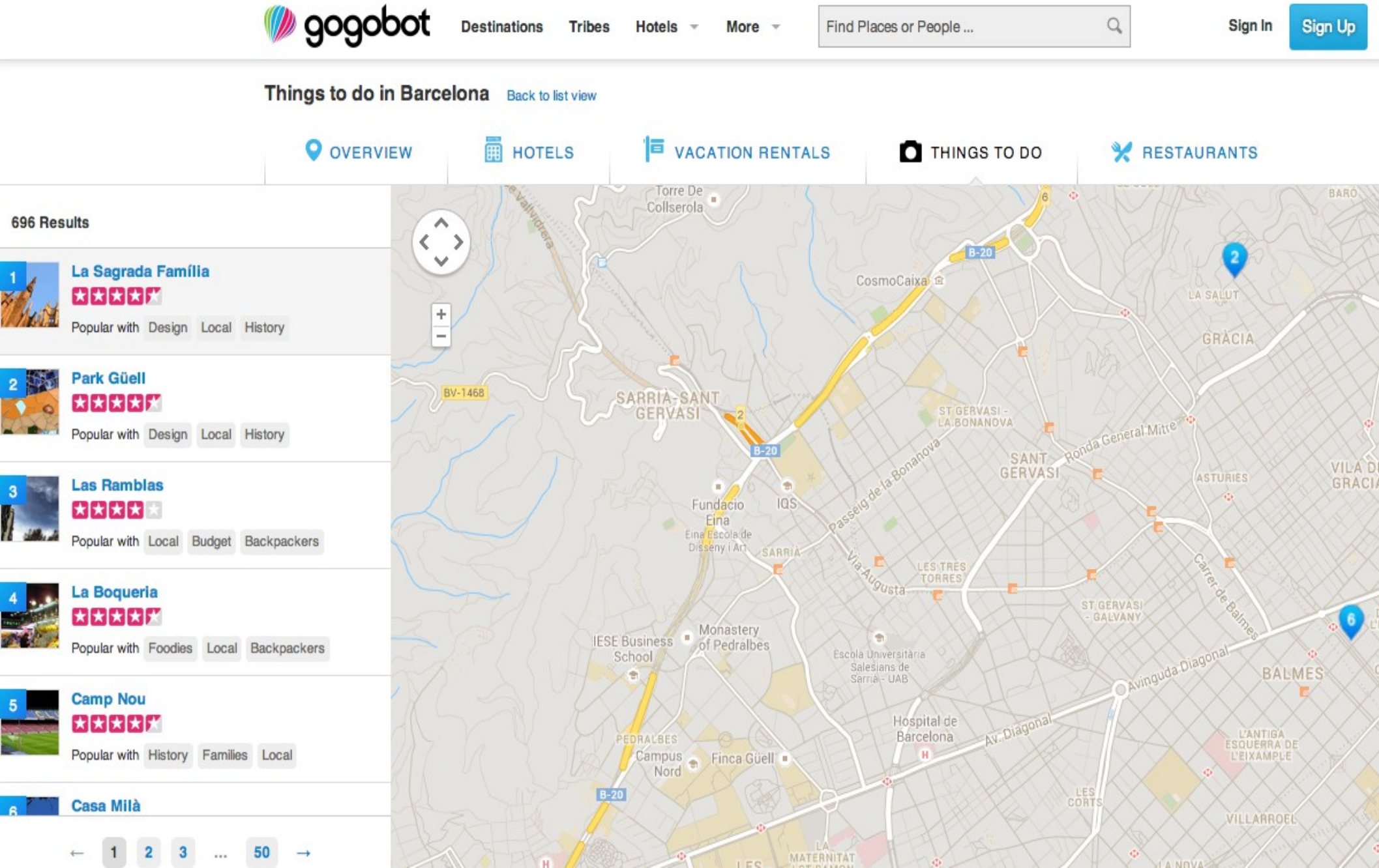
Going to a Salsa Club

Best Salsa Club

NOW

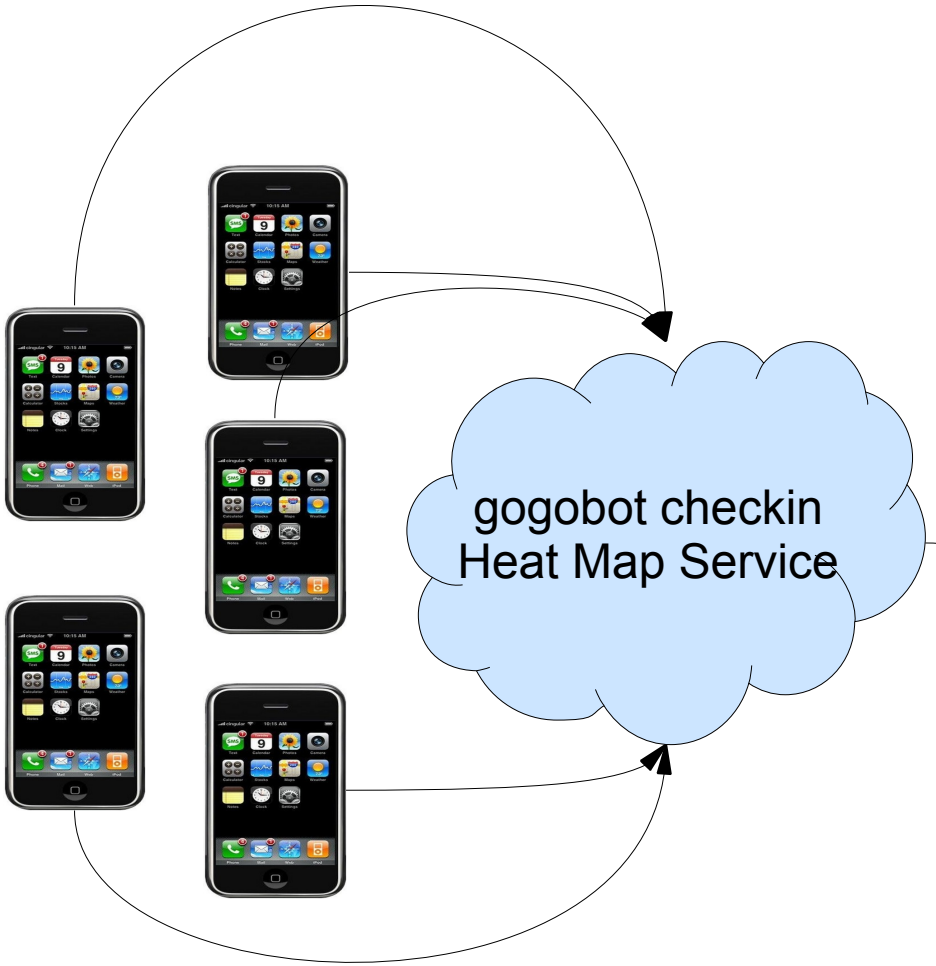
- Good Music
- Crowded – Now!

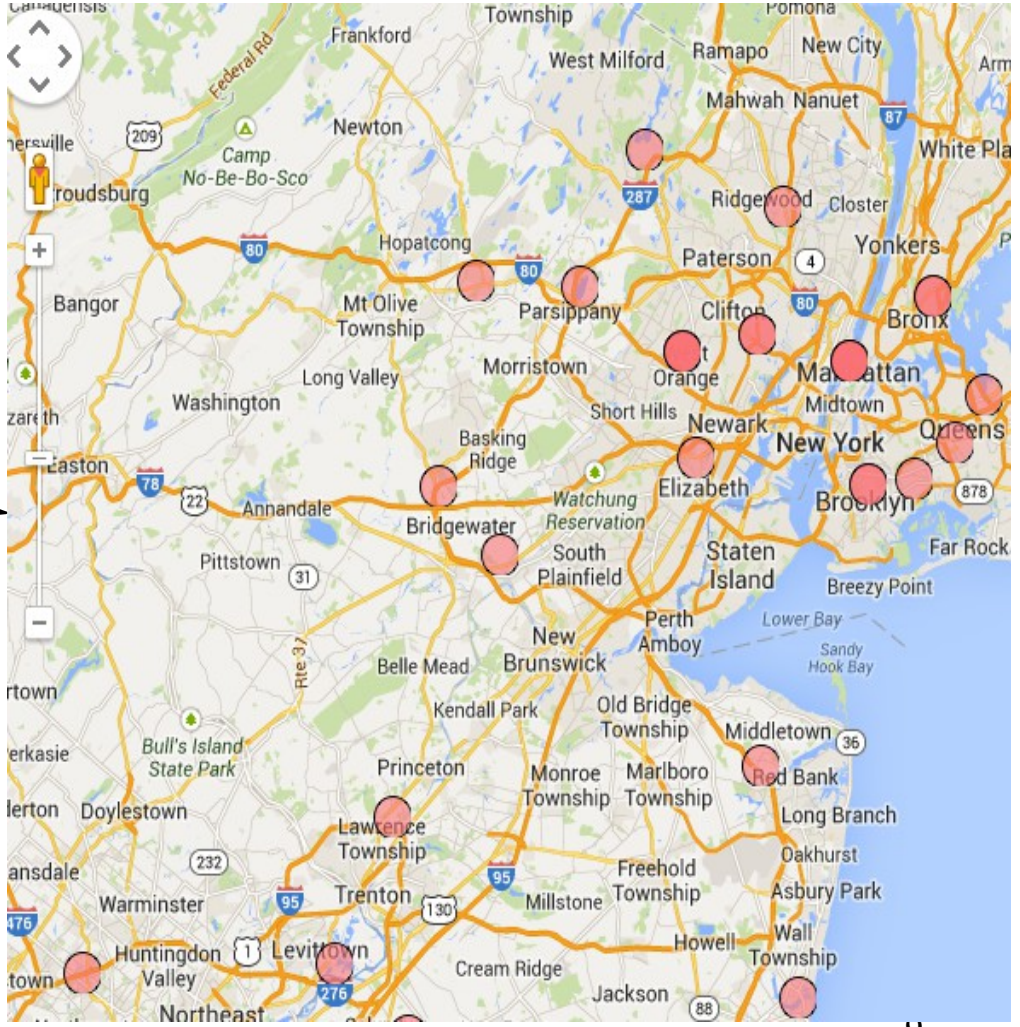
Same Problem in "gogobot"




Do it
Realtime!

Lets' Develop
“Gogobot Checkins Heat-Map”

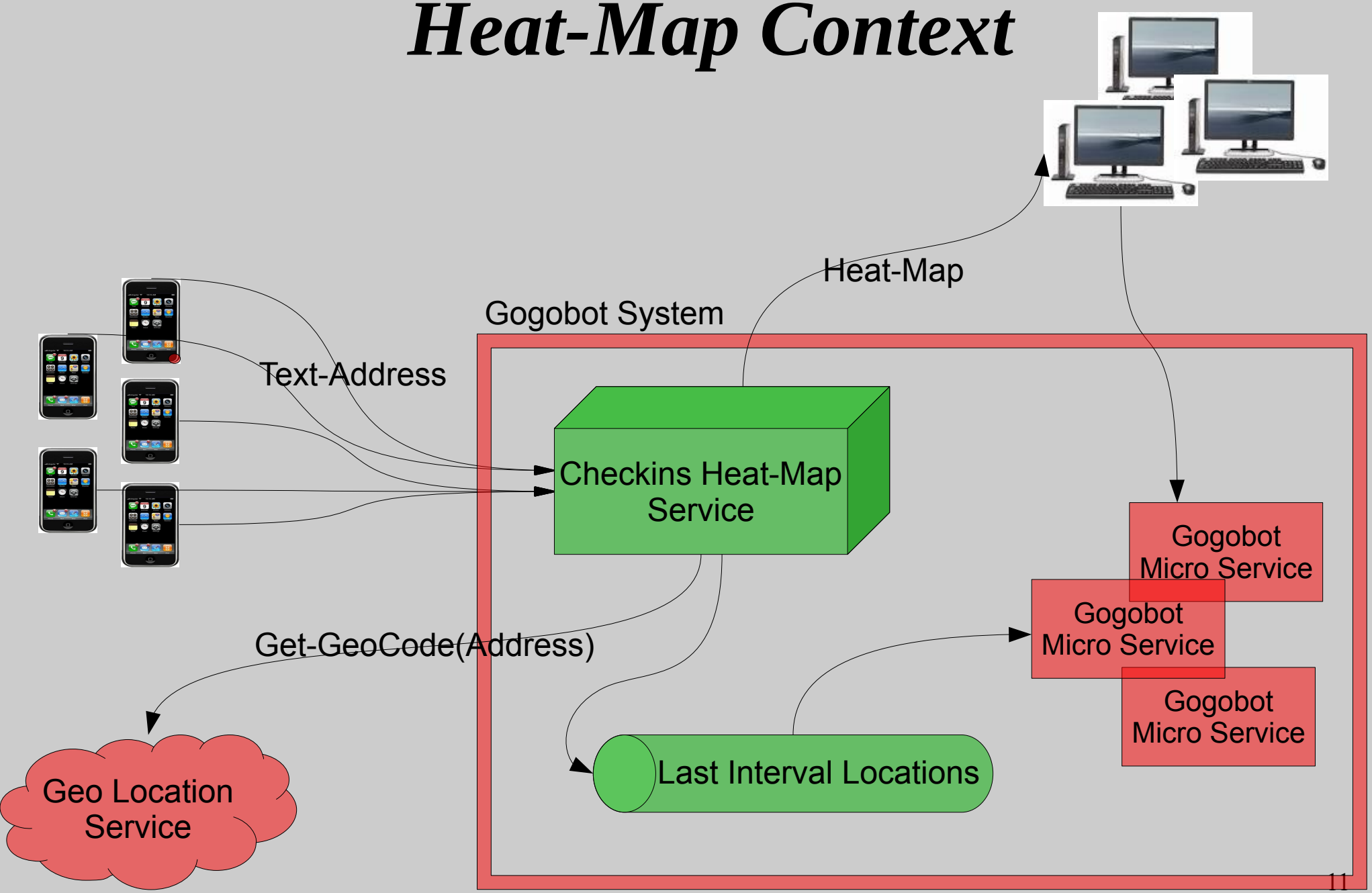




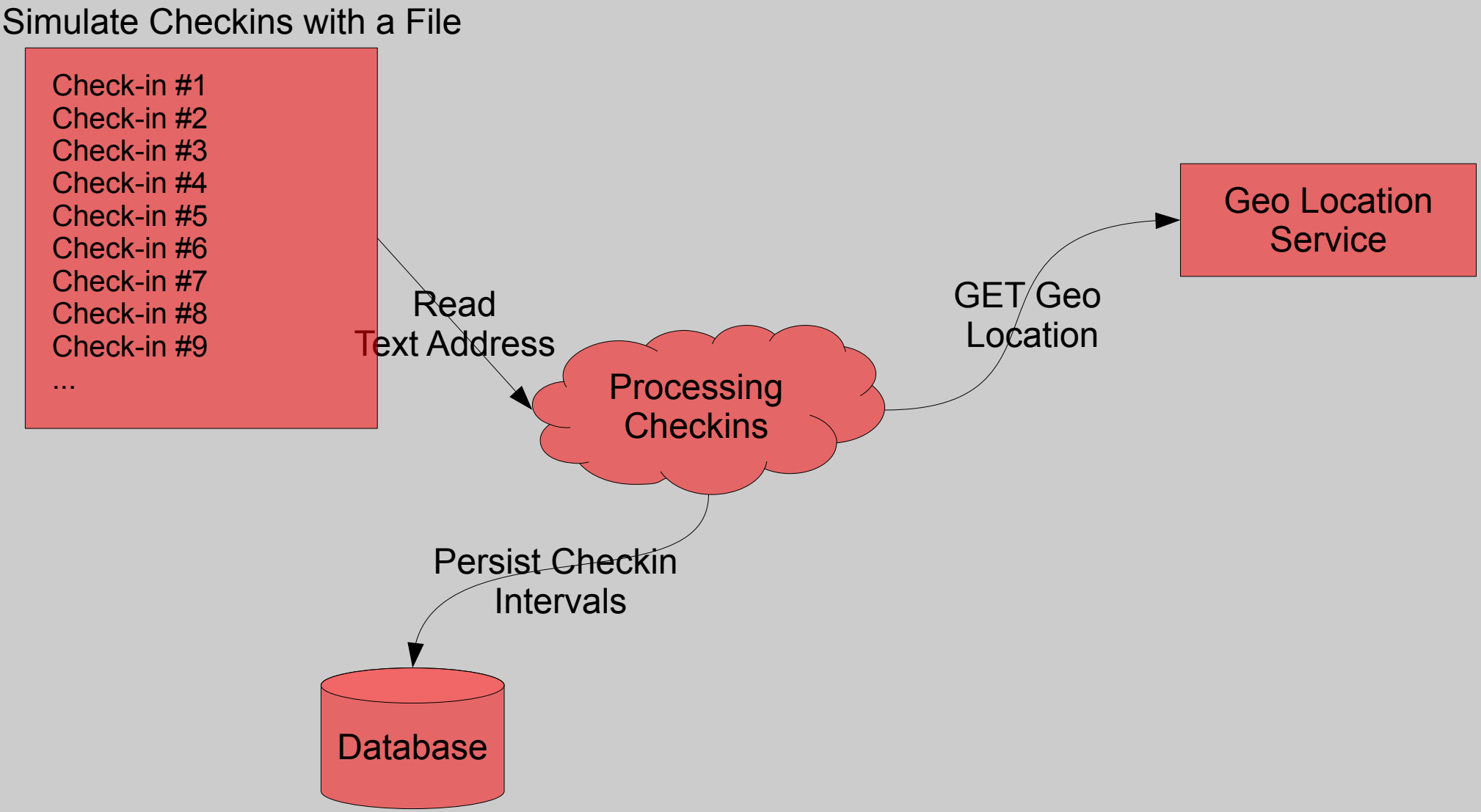
Key Notes

- Collector Service - Collects checkins as text addresses
  - We need to use GeoLocation Service
- Upon elapsed interval, the last locations list will be displayed as Heat-Map in GUI.
- Web Scale service – 10Ks checkins/seconds all over the world (imaginary, but lets do it for the exercise).
- Accuracy – Sample data, NOT critical data.
  - Proportionately representative
  - Data volume is large enough to compensate for data loss.

Heat-Map Context



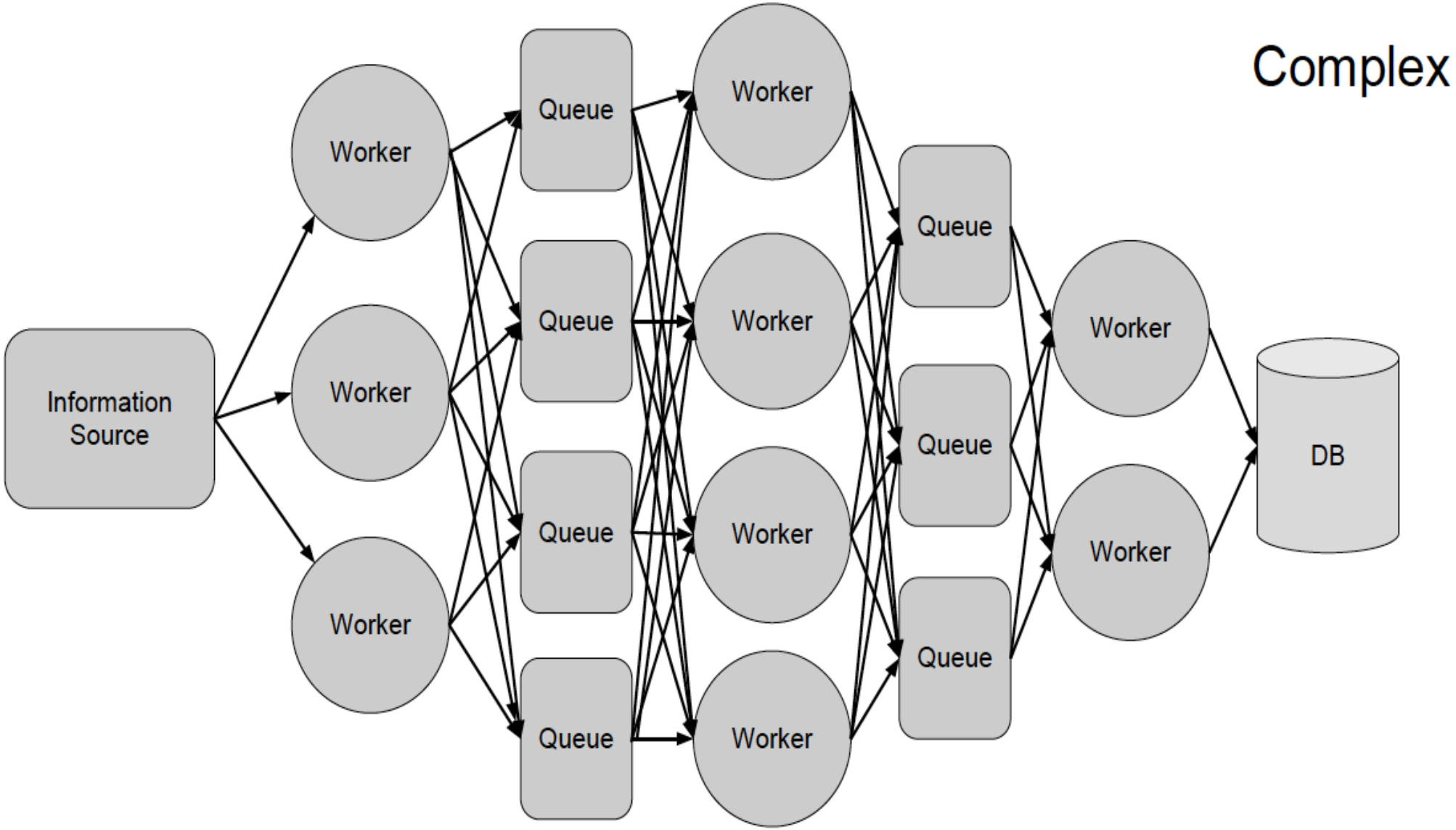
Plan A


Simulate Checkins with a File
Check-in #1
Check-in #2
Check-in #3
Check-in #4
Check-in #5
Check-in #6
Check-in #7
Check-in #8
Check-in #9
...
Read
Text Address
Processing
Checkins
GET Geo
Location
Geo Location
Service
Persist Checkin
Intervals
Database


Tons of Addresses
Arriving Every Second

Architect - First Reaction...

Second Reaction...




Developer
First
Reaction


Second
Reaction

Problems ?

- Tedious: Spend time conf guring where to send messages, deploying workers, and deploying intermediate queues.
- Brittle: There's little fault-tolerance.
- Painful to scale: Partition of running worker/s is complicated.

What We Want ?

- Horizontal scalability
- Fault-tolerance
- No intermediate message brokers!
- Higher level abstraction than message passing
- “Just works”
- Guaranteed data processing (not in this case)

Apache Storm

- ✔Horizontal scalability
- ✔Fault-tolerance
- ✔No intermediate message brokers!
- ✔Higher level abstraction than message passing
- ✔“Just works”
- ✔Guaranteed data processing

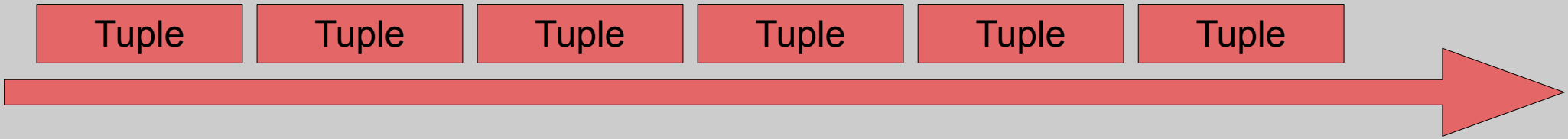

Anatomy of Storm

What is Storm ?

- CEP - Open source and distributed realtime computation system.
  - Makes it easy to reliably process unbounded streams of tuples
  - Doing for realtime processing what Hadoop did for batch processing.

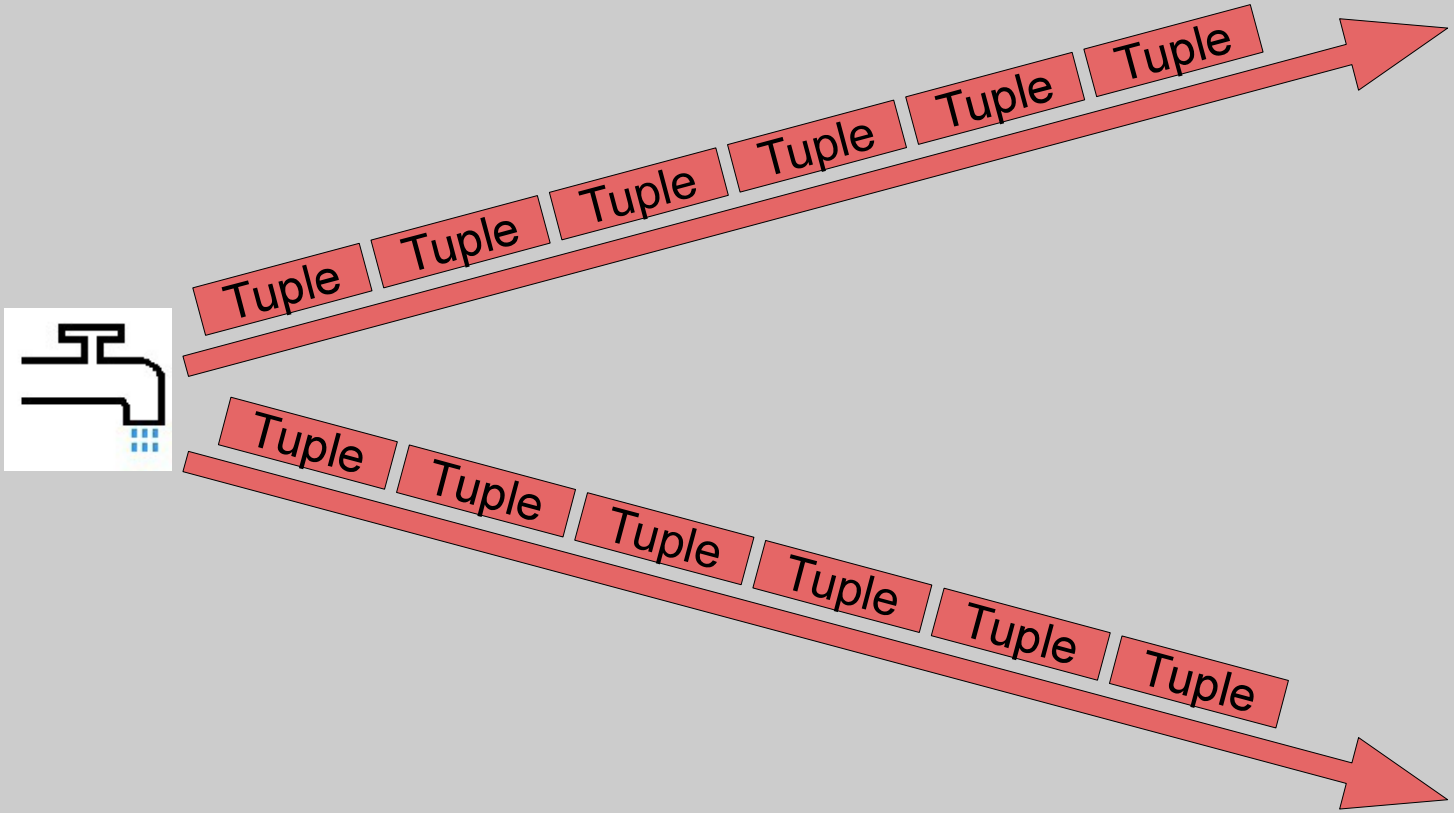
- Fast - 1M Tuples/sec per node.
  - It is scalable,fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

Streams



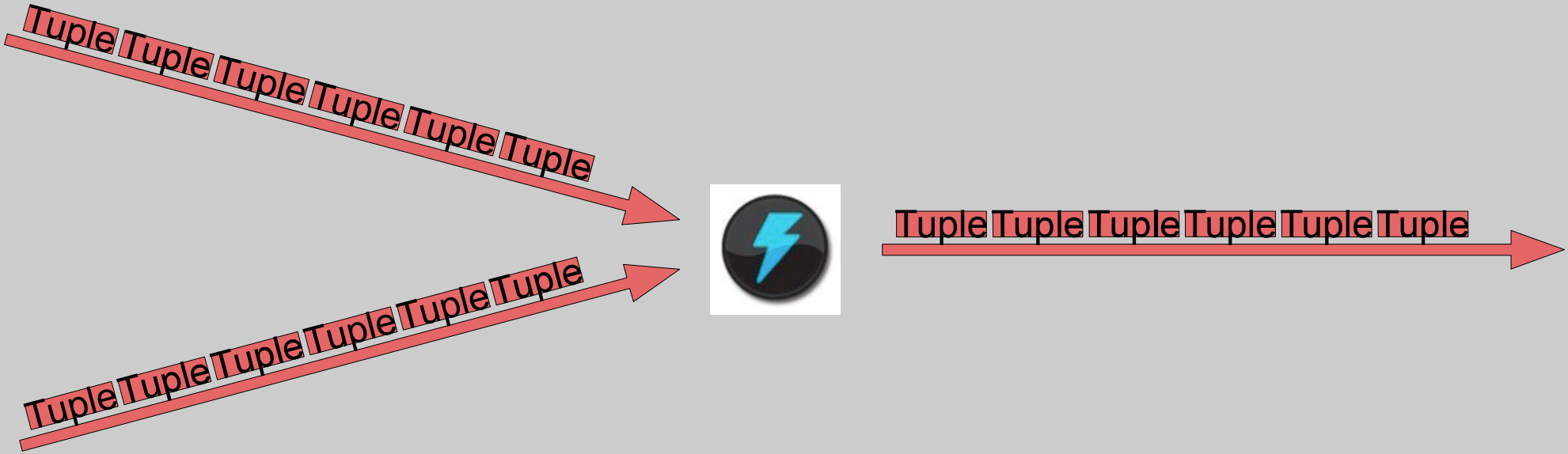
Unbounded sequence of tuples

Spouts



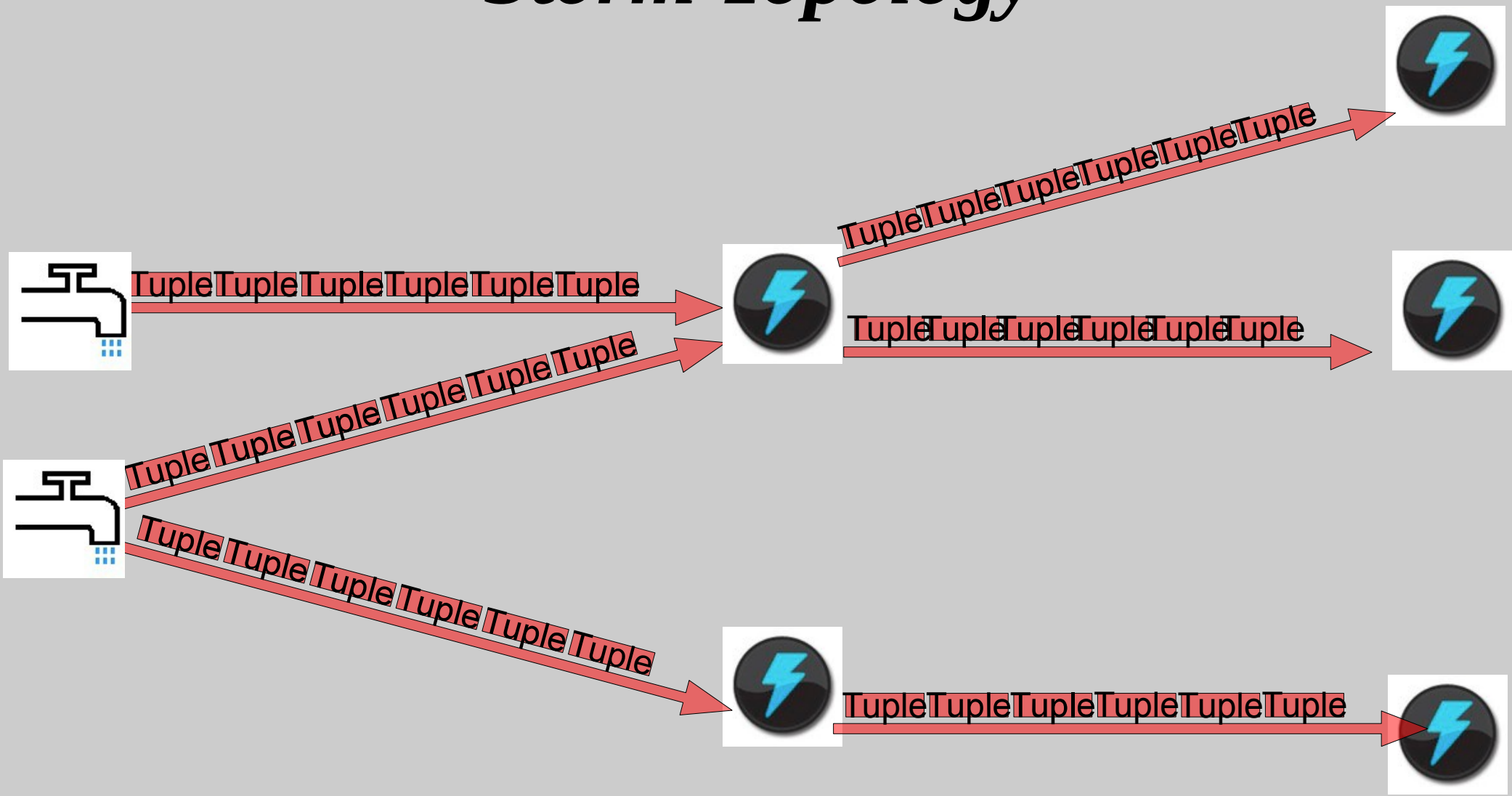
Sources of Streams

Bolts



Processes input streams and produces new streams

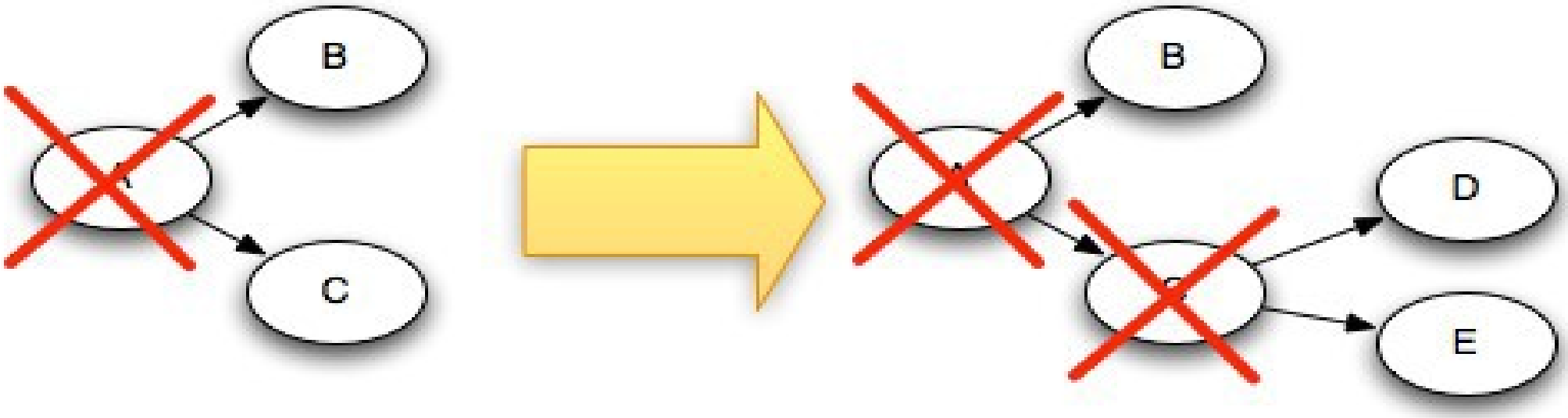
Storm Topology



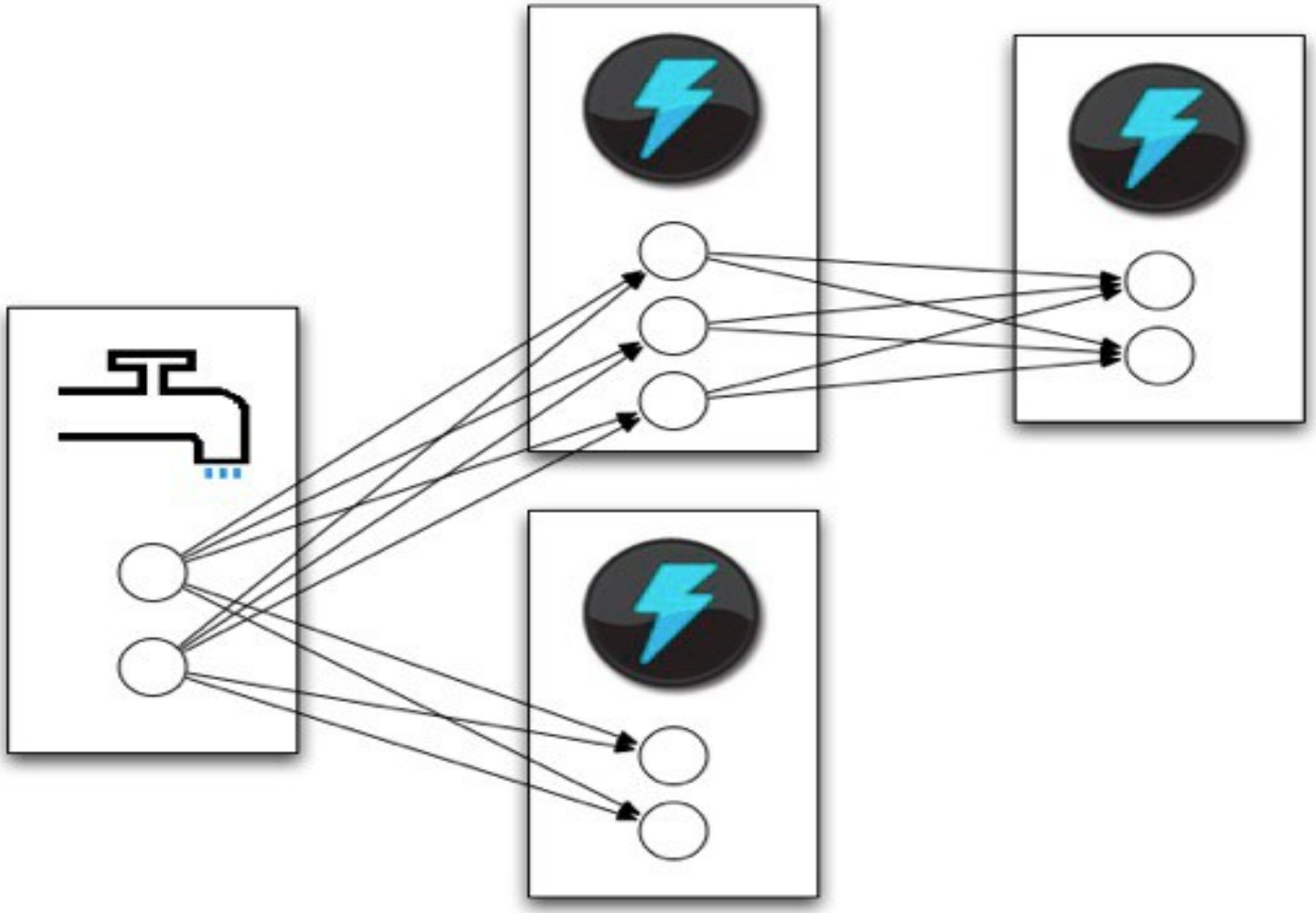
Network of spouts and bolts

Guarantee for Processing

- Storm guarantees the full processing of a tuple by tracking its state
- In case of failure, Storm can re-process it.
- Source tuples with full “acked” trees are removed from the system

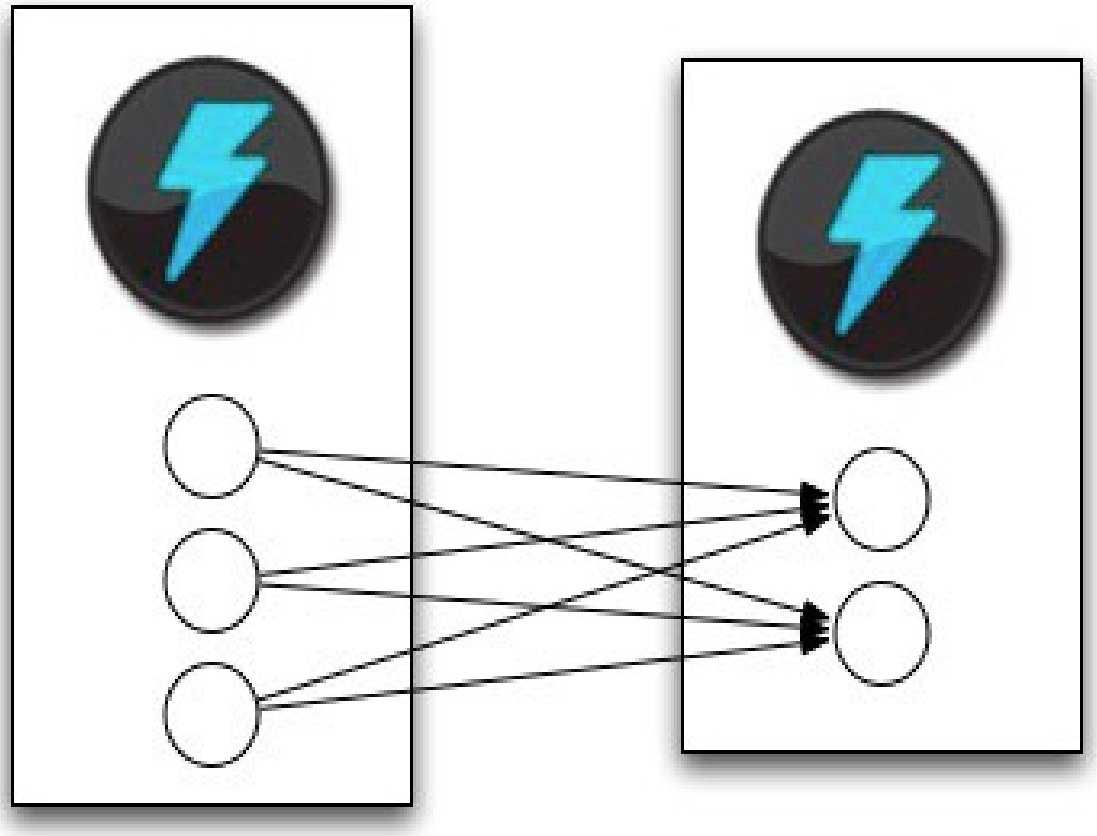


Tasks (Bolt/Spout Instance)



Spouts and bolts execute as many tasks across the cluster

Stream Grouping

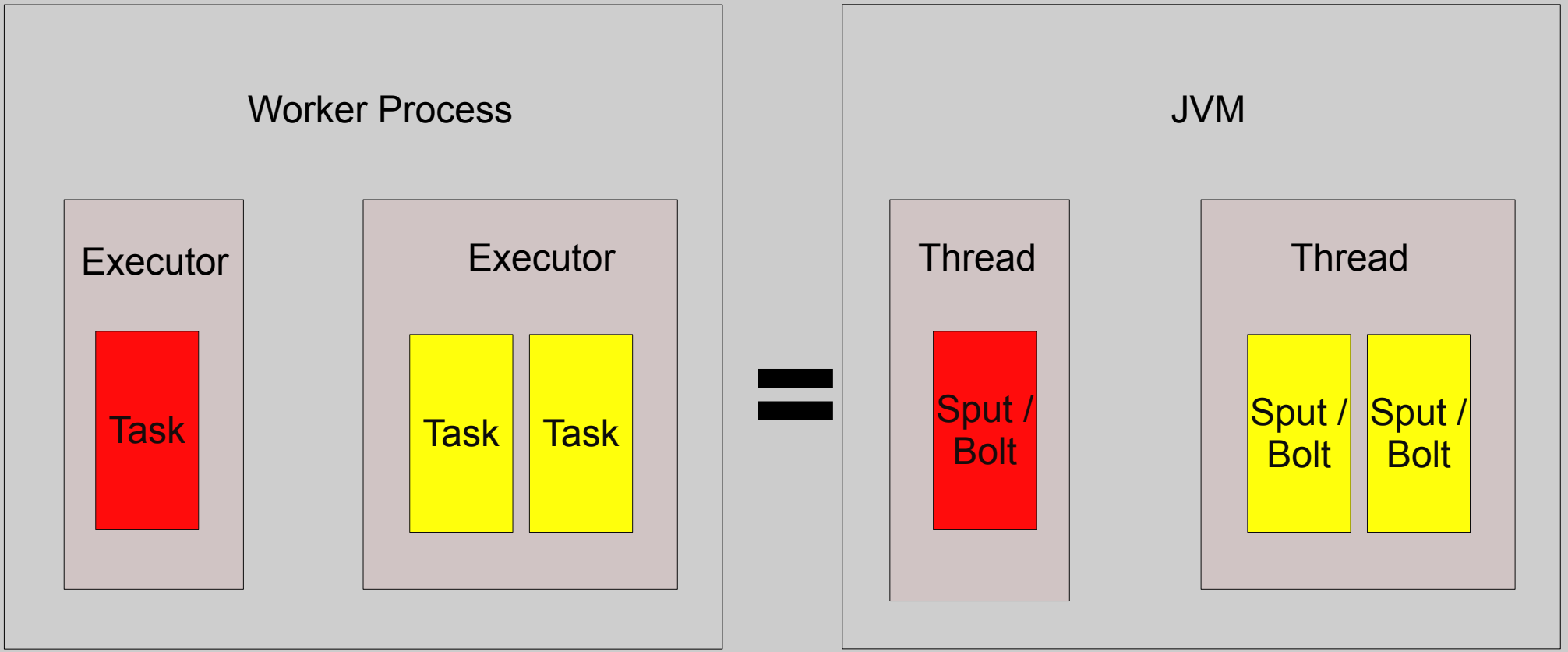


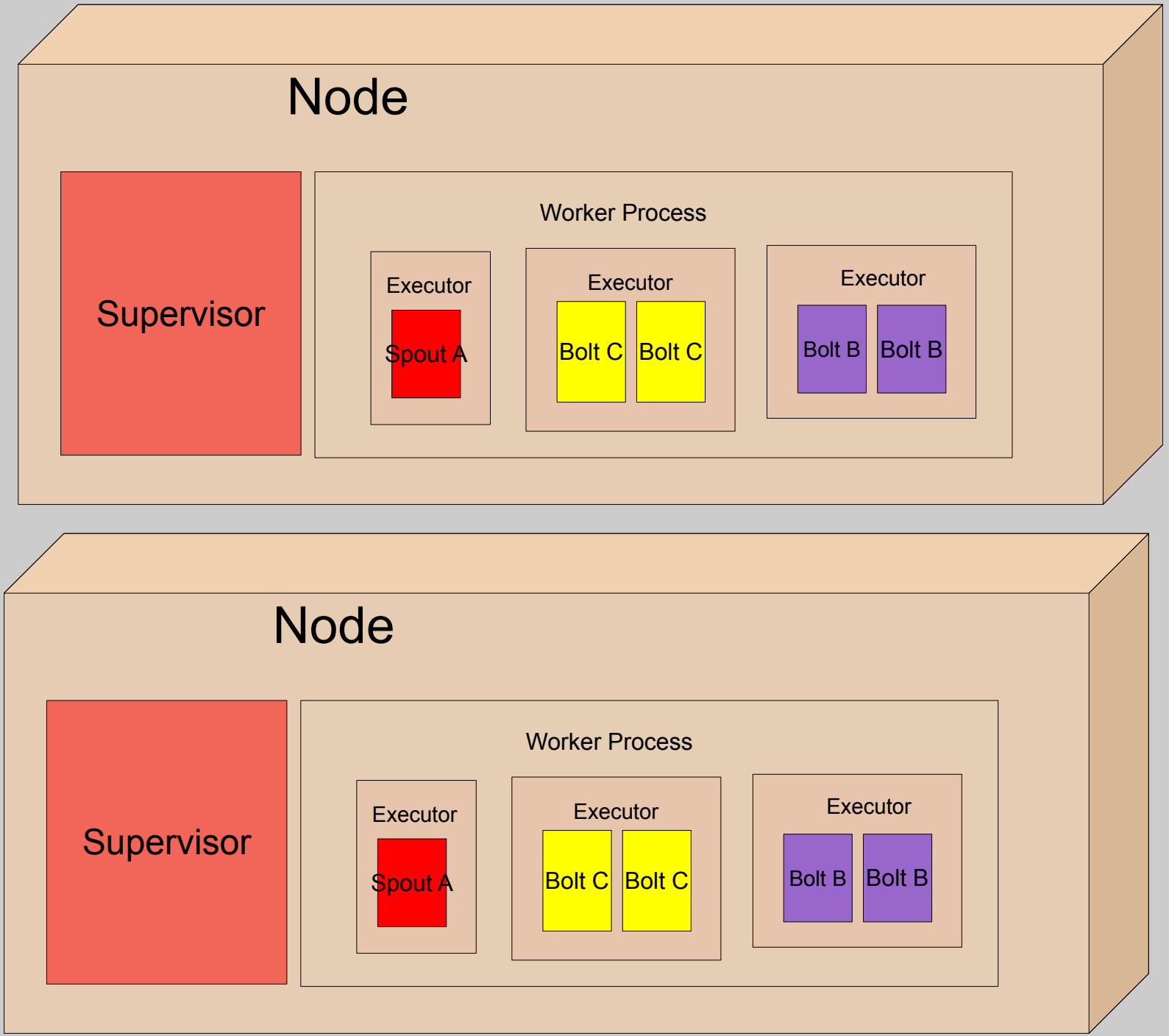
When a tuple is emitted, which task (instance) does it go to?

Stream Grouping

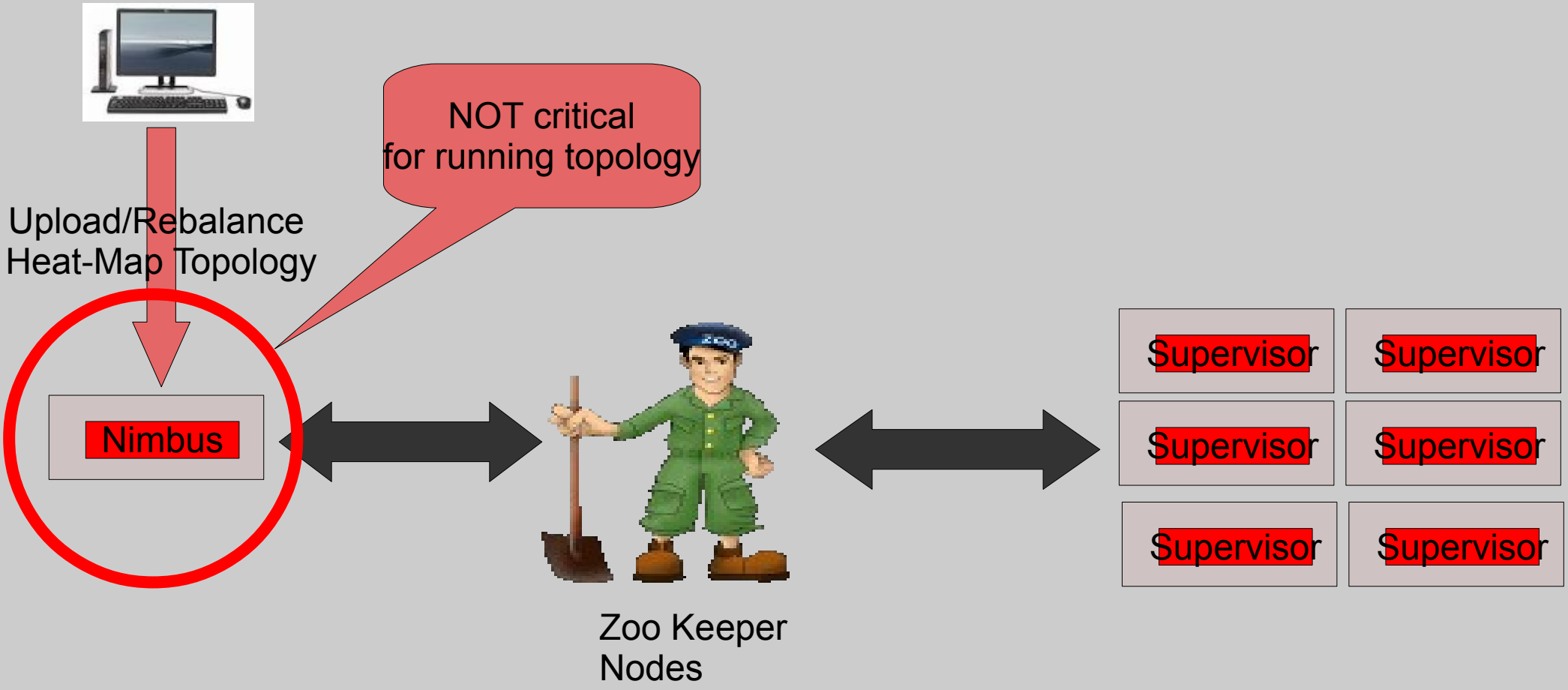
- Shuff e grouping: pick a random task
- Fields grouping: consistent hashing on a subset of tuple f elds
- All grouping: send to all tasks
- Global grouping: pick task with lowest id

Tasks , Executors , Workers



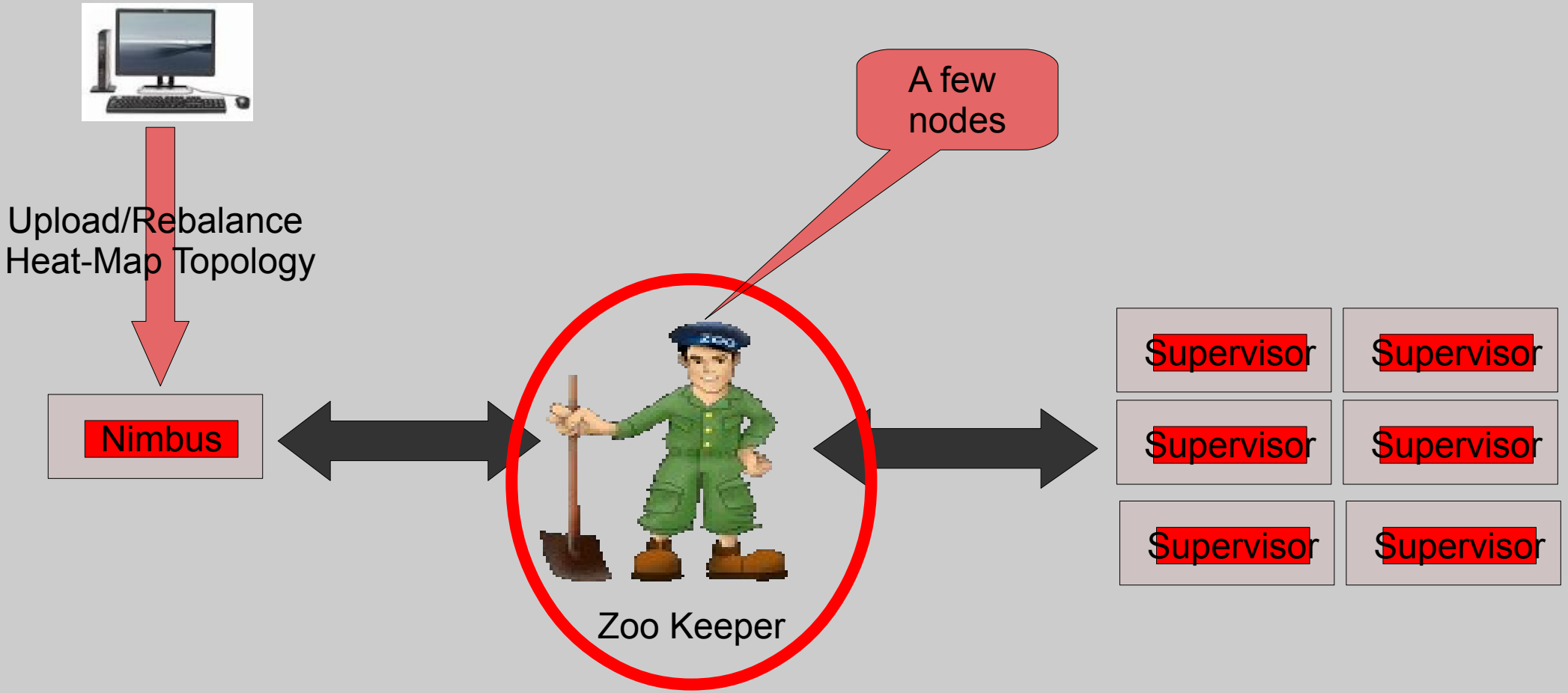

Node
Supervisor
Worker Process
Executor
Spout A
Executor
Bolt C
Bolt C
Executor
Bolt B
Bolt B
Node
Supervisor
Worker Process
Executor
Spout A
Executor
Bolt C
Bolt C
Executor
Bolt B
Bolt B

Storm Architecture



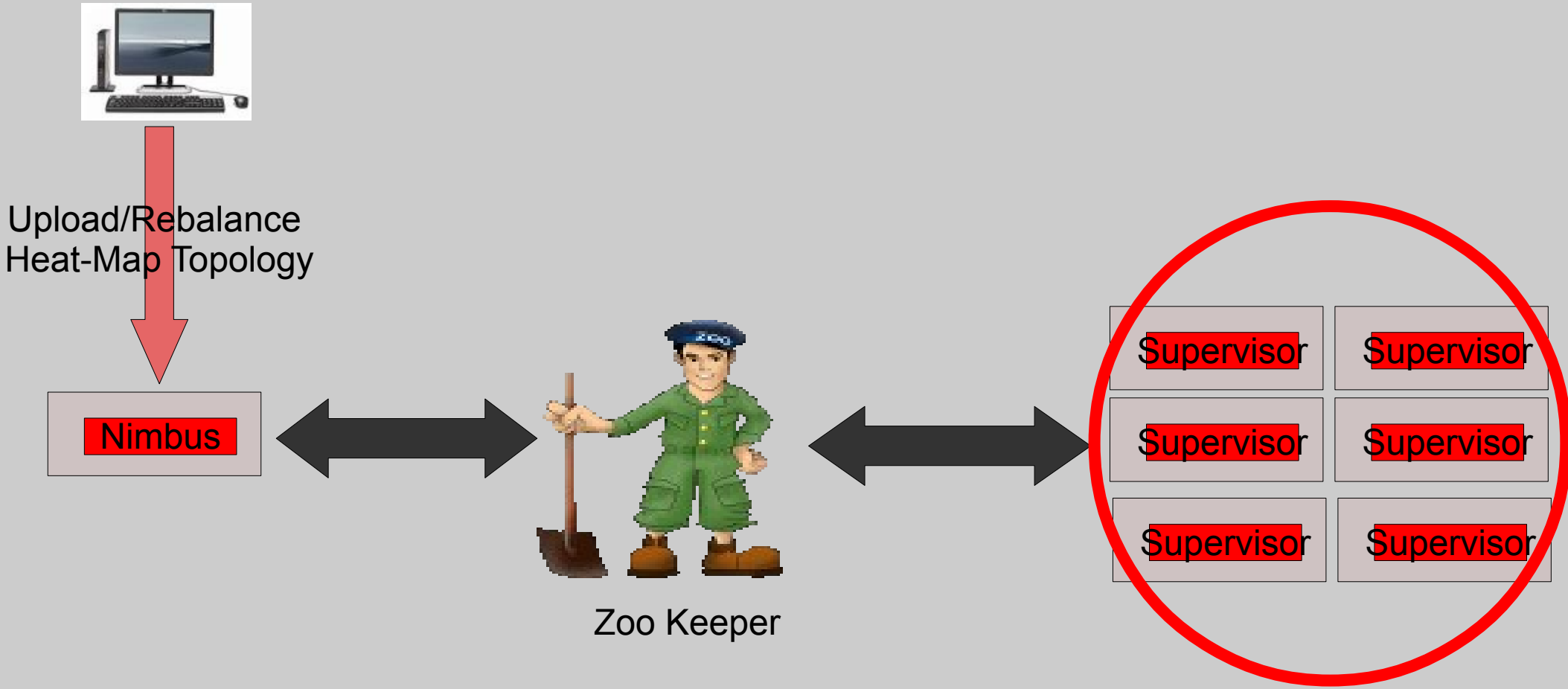
Master Node
(similar to Hadoop JobTracker)

Storm Architecture



Used For Cluster Coordination

Storm Architecture



Run Worker Processes

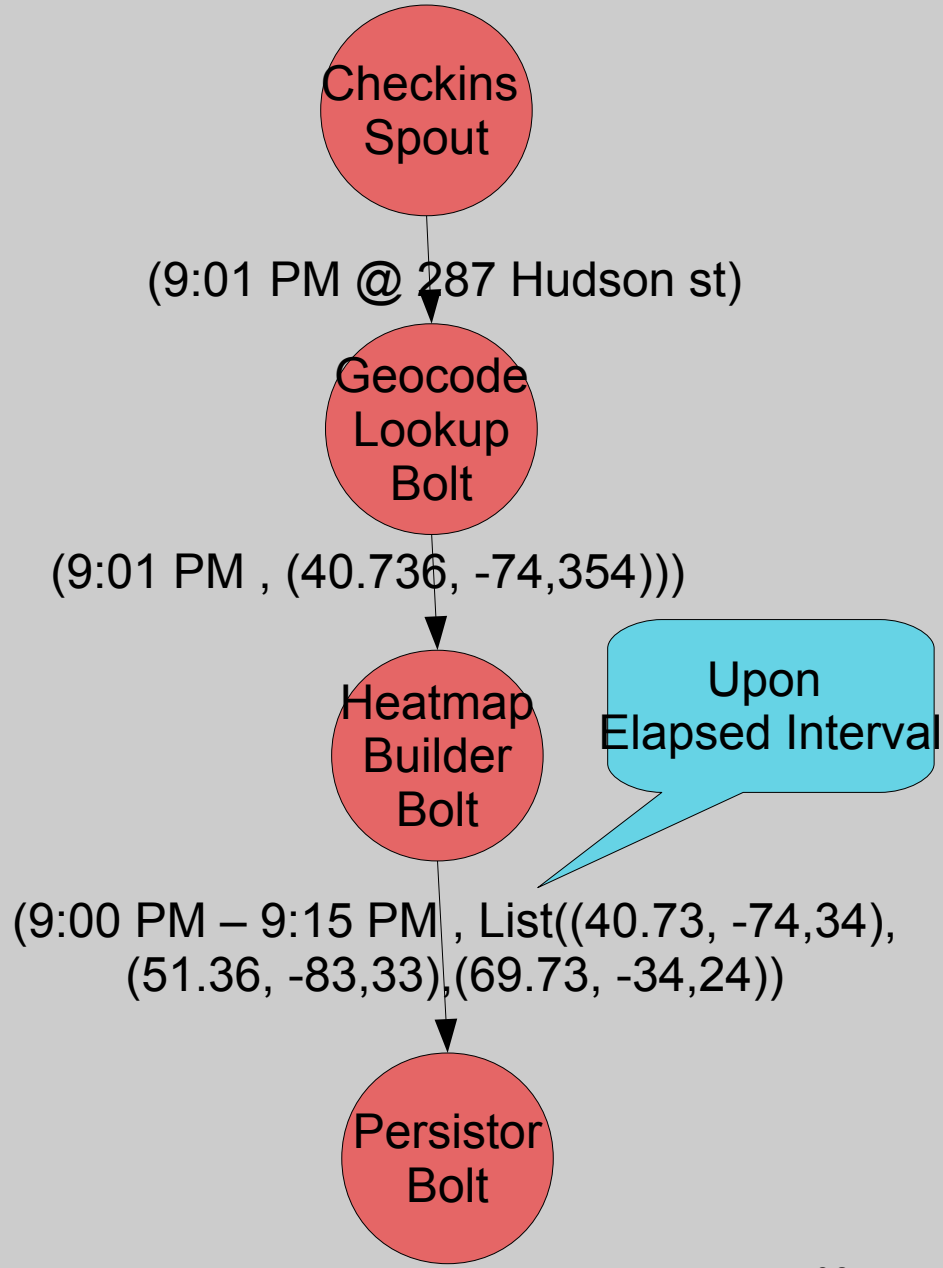
Assembling Heatmap Topology

HeatMap Input/Output Tuples

- Input Tuples: Timestamp and Text Address :
  - (9:00:07 PM , “287 Hudson St New York NY 10013”)

- Output Tuple: Time interval, and a list of points for it:
  - (9:00:00 PM to 9:00:15 PM, List((40.719,-73.987),(40.726,-74.001),(40.719,-73.987))

Heat Map Storm Topology



Checkins Spout

```
public class CheckinsSpout extends BaseRichSpout {
    private List<String> sampleLocations;
    private int nextEmitIndex;
    private SpoutOutputCollector outputCollector;


    @Override
    public void open(Map map, TopologyContext topologyContext,
                                    SpoutOutputCollector spoutOutputCollector) {
        this.outputCollector = spoutOutputCollector;
        this.nextEmitIndex = 0;
        sampleLocations = IOUtils.readLines(
            ClassLoader.getSystemResourceAsStream("sanple-locations.txt"));
    }

    @Override
    public void nextTuple() {
        String address = checkins.get(nextEmitIndex);
        String checkin = new Date().getTime()+"@ADDRESS:"+address;
        outputCollector.emit(new Values(checkin));
        nextEmitIndex = (nextEmitIndex + 1) % sampleLocations.size();
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("str"));
    }
}
```

We hold state
No need for thread safety

Been called iteratively by Storm

Declare output fields

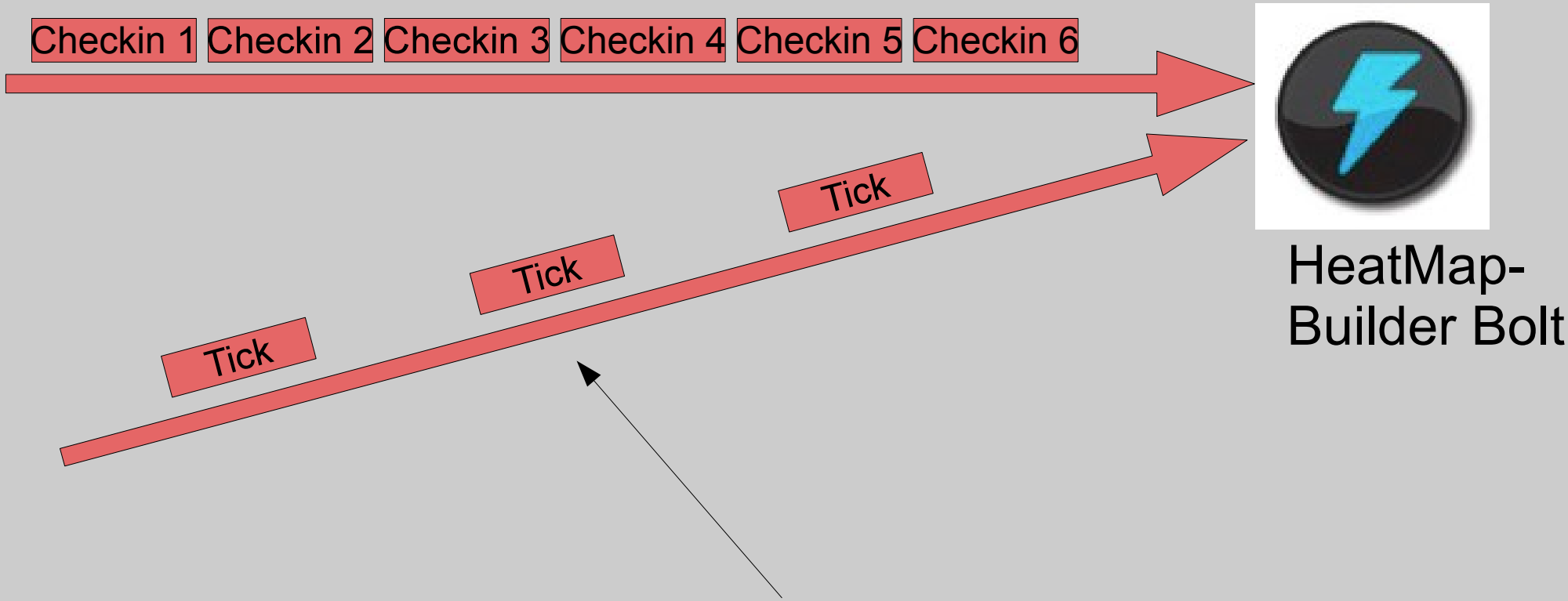
Geocode Lookup Bolt

```
public class GeocodeLookupBolt extends BaseBasicBolt {
    private LocatorService locatorService;

    @Override
    public void prepare(Map stormConf, TopologyContext context) {
        locatorService = new GoogleLocatorService();
    }

    @Override
    public void execute(Tuple tuple, BasicOutputCollector outputCollector) {
        String str = tuple.getStringByField("str");
        String[] parts = str.split("@");
        Long time = Long.valueOf(parts[0]);
        String address = parts[1];

        LocationDTO locationDTO = locatorService.getLocation(address);
        if(checkinDTO!=null)
            outputCollector.emit(new Values(time,locationDTO) );
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer fieldsDeclarer) {
        fieldsDeclarer.declare(new Fields("time", "location"));
    }

}
```

Get Geocode,
Create DTO

Tick Tuple – Repeating Mantra

Two Streams to Heat-Map Builder



On tick tuple, we f ush our Heat-Map

Tick Tuple in Action

```
public class HeatMapBuilderBolt extends BaseBasicBolt {
    private Map<String, List<LocationDTO>> heatmaps;

    @Override
    public Map<String, Object> getComponentConfiguration() {
        Config conf = new Config();
        conf.put(Config.TOPOLOGY_TICK_TUPLE_FREQ_SECS,  60 );
        return conf;
    }

    @Override
    public void execute(Tuple tuple, BasicOutputCollector outputCollector) {
        if (isTickTuple(tuple)) {
            // Emit accumulated intervals
        } else {
            // Add check-in info to the current interval in the Map
        }
    }

    private boolean isTickTuple(Tuple tuple) {
       return  tuple.getSourceComponent().equals(Constants.SYSTEM_COMPONENT_ID)
         &&  tuple.getSourceStreamId().equals(Constants.SYSTEM_TICK_STREAM_ID);
    }
```

Hold latest intervals

Tick interval

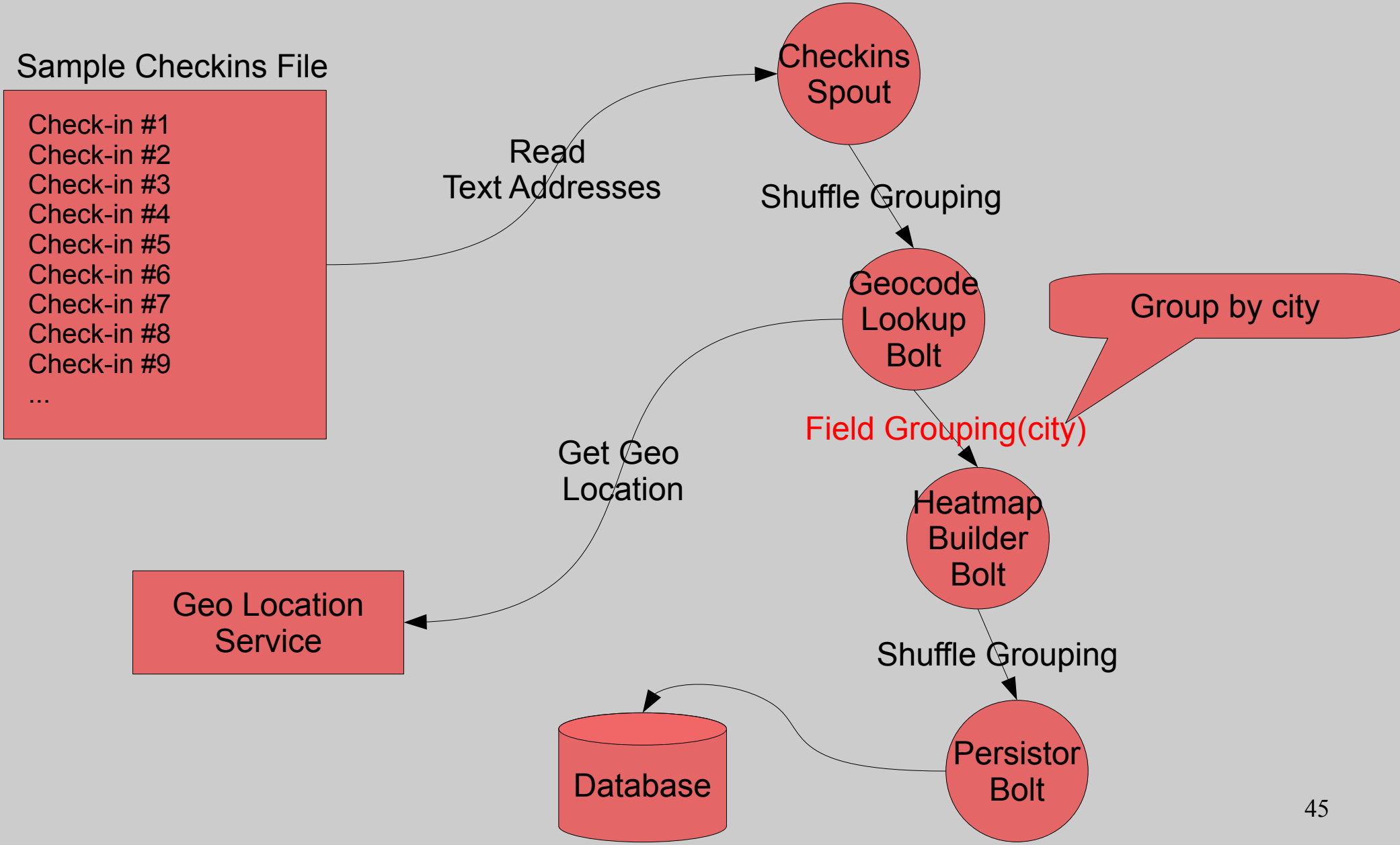
Persister Bolt

```
public class PersistorBolt extends BaseBasicBolt {
    private Jedis jedis;

    @Override
    public void execute(Tuple tuple, BasicOutputCollector outputCollector) {

      Long timeInterval = tuple.getLongByField("time-interval");
      String city = tuple.getStringByField("city");
      String locationsList = objectMapper.writeValueAsString
                                    ( tuple.getValueByField("locationsList"));

      String dbKey = "checkins-" + timeInterval+"@"+city;


      jedis.setex(dbKey, 3600*24 ,locationsList);

      jedis.publish("location-key", dbKey);

    }
}
```

Persist in Redis for 24h

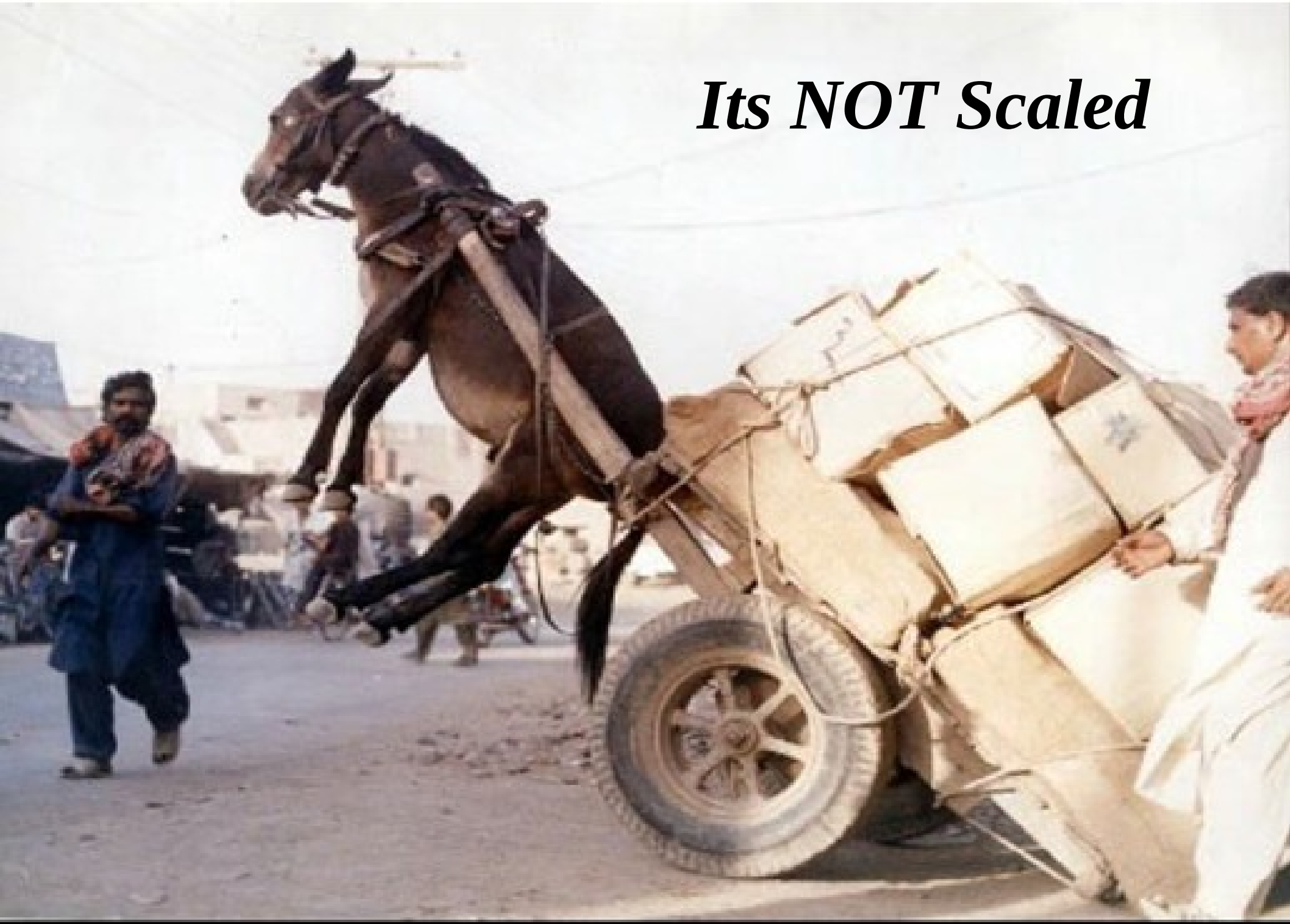
Publish in Redis channel for debugging

Transforming the Tuples



Heat Map Topology

```
public class LocalTopologyRunner {
    public static void main(String[] args) {
        TopologyBuilder builder = buildTopolgy();
        StormSubmitter.submitTopology(
                    "local-heatmap", new Config(), builder.createTopology());

    }


    private static TopologyBuilder buildTopolgy() {
      topologyBuilder builder = new TopologyBuilder();

      builder.setSpout("checkins", new CheckinsSpout());

      builder.setBolt("geocode-lookup", new GeocodeLookupBolt() )
                    .shuffleGrouping("checkins");

      builder.setBolt("heatmap-builder", new HeatMapBuilderBolt() )
            .fieldsGrouping("geocode-lookup", new Fields("city"));

      builder.setBolt("persistor", new PersistorBolt() )
                        .shuffleGrouping("heatmap-builder");

         return builder;
    }
}
```


Its NOT Scaled


FAIL.
NMOABC.com

Scaling the Topology

```
public class LocalTopologyRunner {
conf.setNumWorkers(20);
    public static void main(String[] args) {
        TopologyBuilder builder = buildTopolgy();
        Config conf = new Config();
        conf.setNumWorkers(2);
        StormSubmitter.submitTopology(
                    "local-heatmap", conf, builder.createTopology());
    }



    private static TopologyBuilder buildTopolgy() {
      topologyBuilder builder = new TopologyBuilder();

      builder.setSpout("checkins", new CheckinsSpout(), 4 );

      builder.setBolt("geocode-lookup", new GeocodeLookupBolt() , 8 )
                        .shuffleGrouping("checkins").setNumTasks(64);

      builder.setBolt("heatmap-builder", new HeatMapBuilderBolt() , 4)
             .fieldsGrouping("geocode-lookup", new Fields("city"));

      builder.setBolt("persistor", new PersistorBolt() , 2 )
                        .shuffleGrouping("heatmap-builder").setNumTasks(4);

         return builder;
```

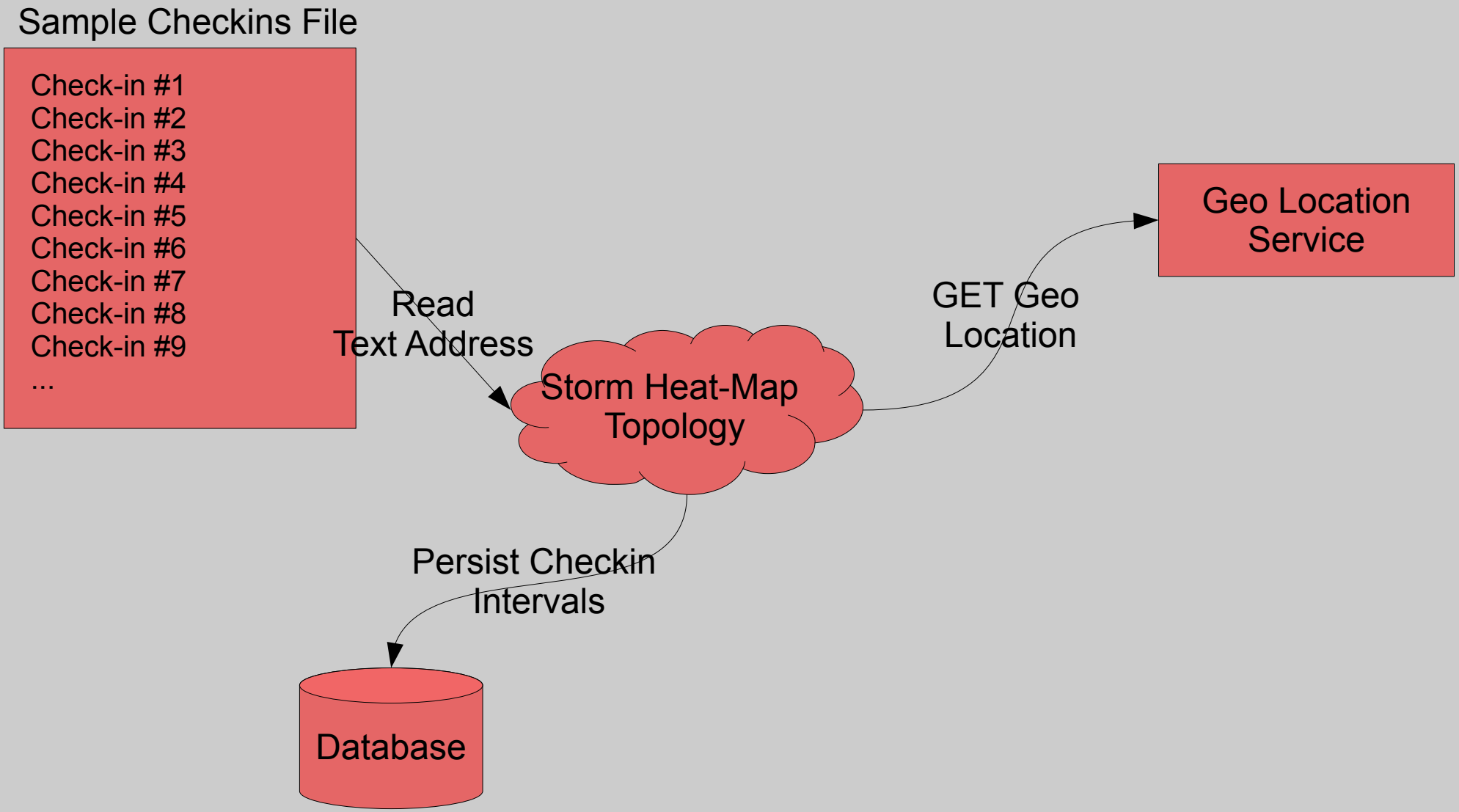
Set no. of workers

Parallelism hint

Increase Tasks For Future


Demo

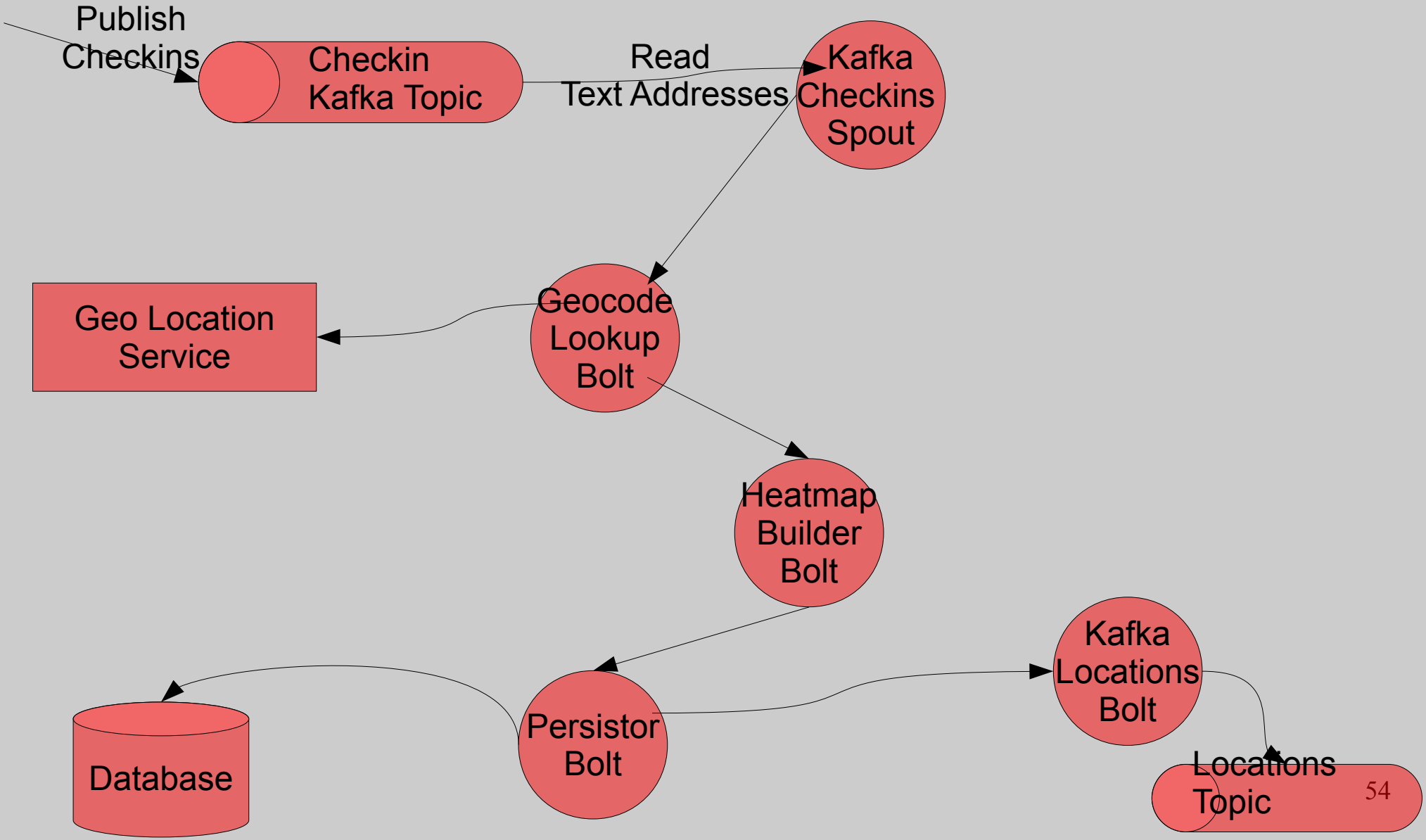
Recap – Plan A


Sample Checkins File
Check-in #1
Check-in #2
Check-in #3
Check-in #4
Check-in #5
Check-in #6
Check-in #7
Check-in #8
Check-in #9
...
Read
Text Address
Storm Heat-Map
Topology
GET Geo
Location
Geo Location
Service
Persist Checkin
Intervals
Database

We have something working

Add Kafka Messaging

Plan B - Kafka Spout&Bolt to HeatMap



HornetQ

ActiveMQ

Why Kafka?

When we have....
Aren't they Good?

ØMQ

RabbitMQ
Open Source Enterprise Messaging

*Apache ActiveMQ, JBoss HornetQ, Zero MQ, RabbitMQ are respective brands of Apache Software Foundation, JBoss Inc, iMatix Corporation, and Vmware Inc.

They all are Good

But not for all use-cases

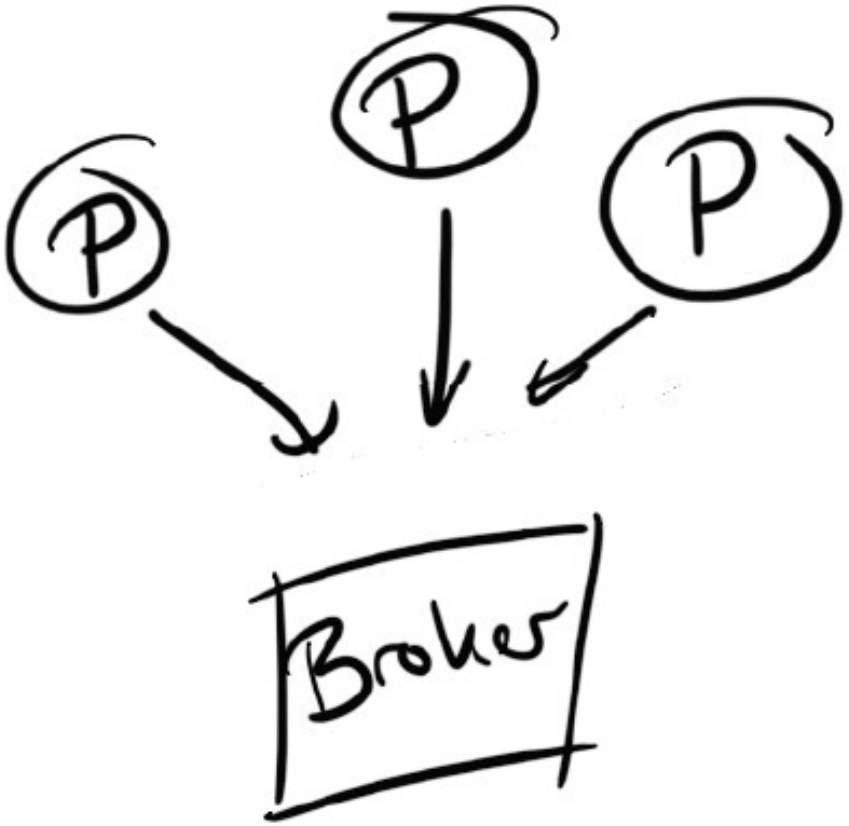
Kafka

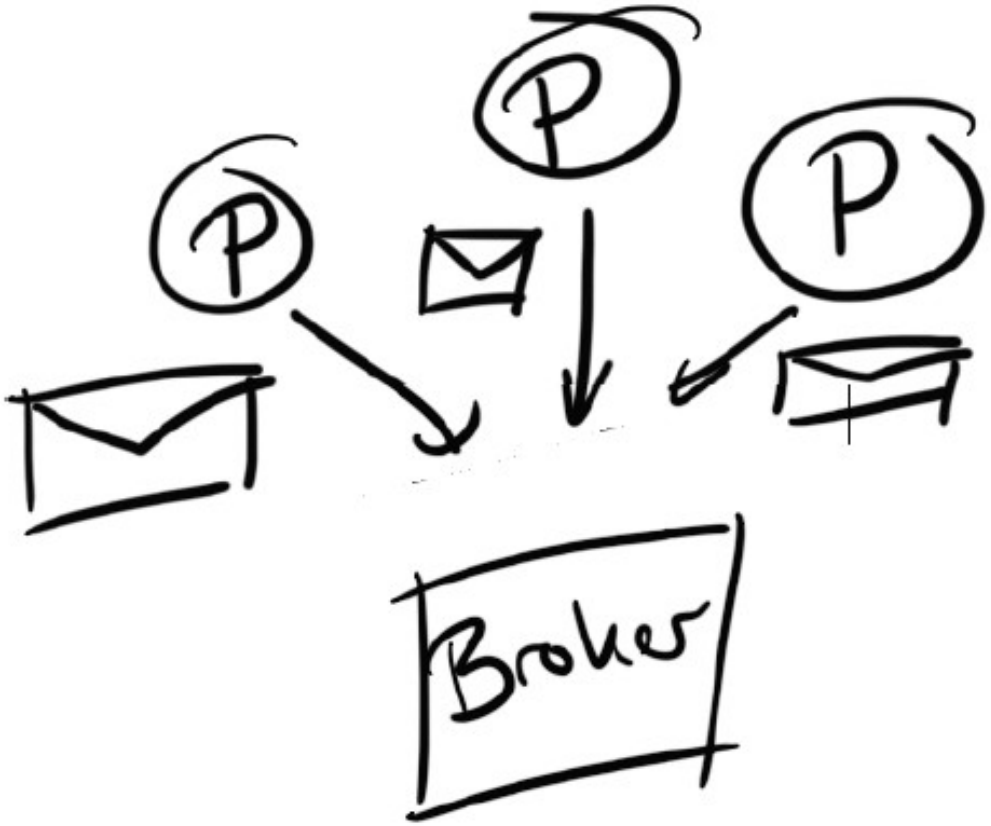
A little introduction

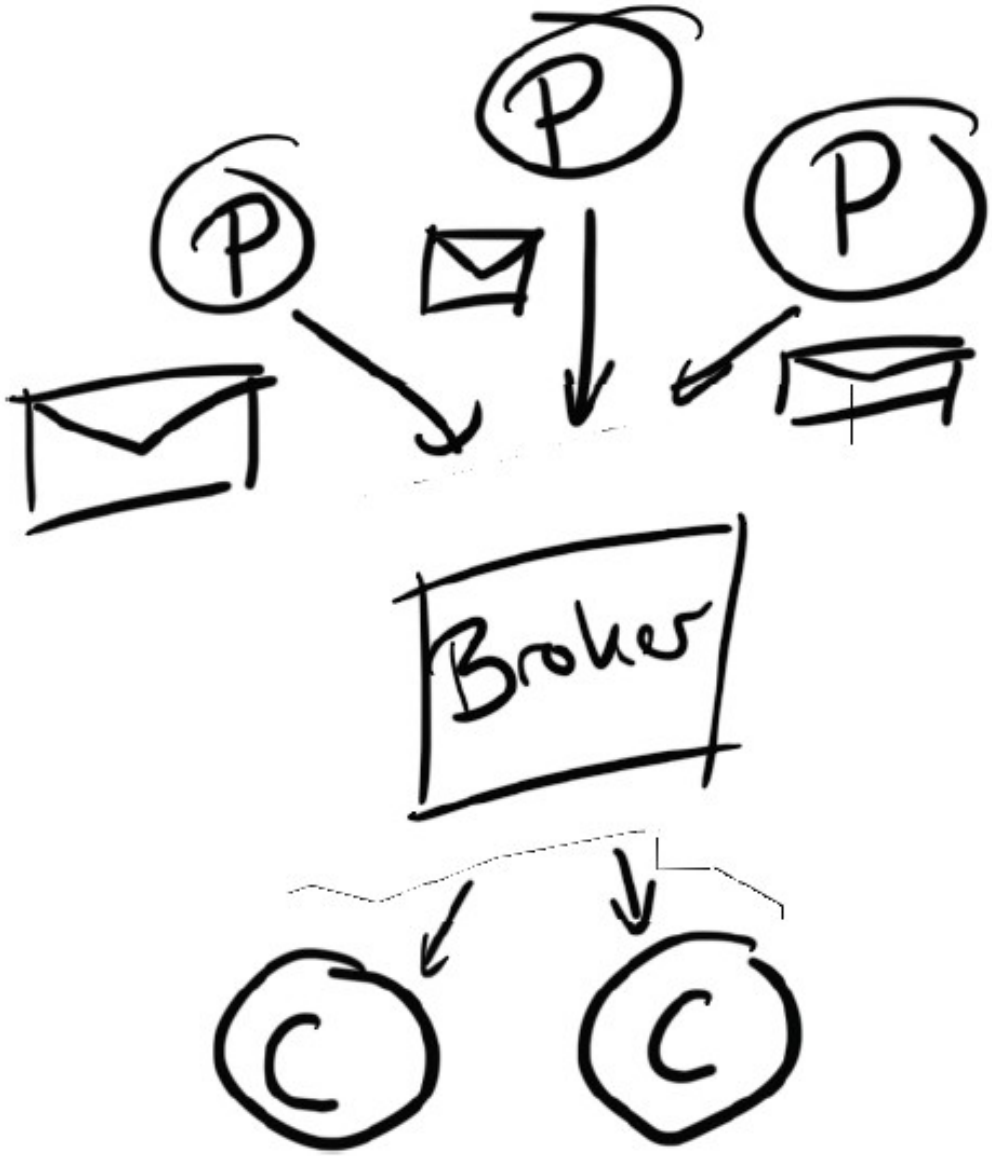

LinkedIn

Pub-Sub Messaging System

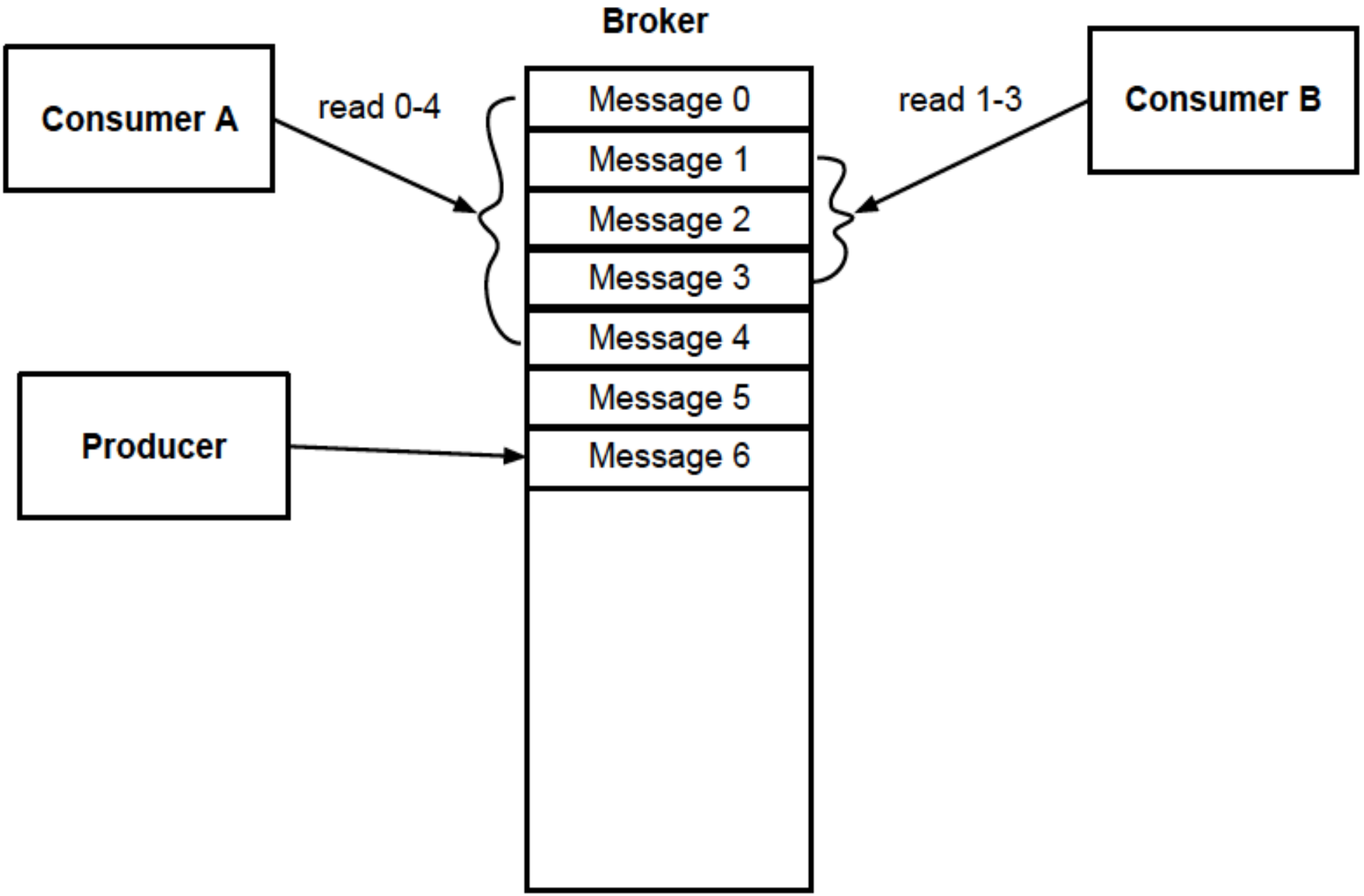

Broker


P
P
P
Broker

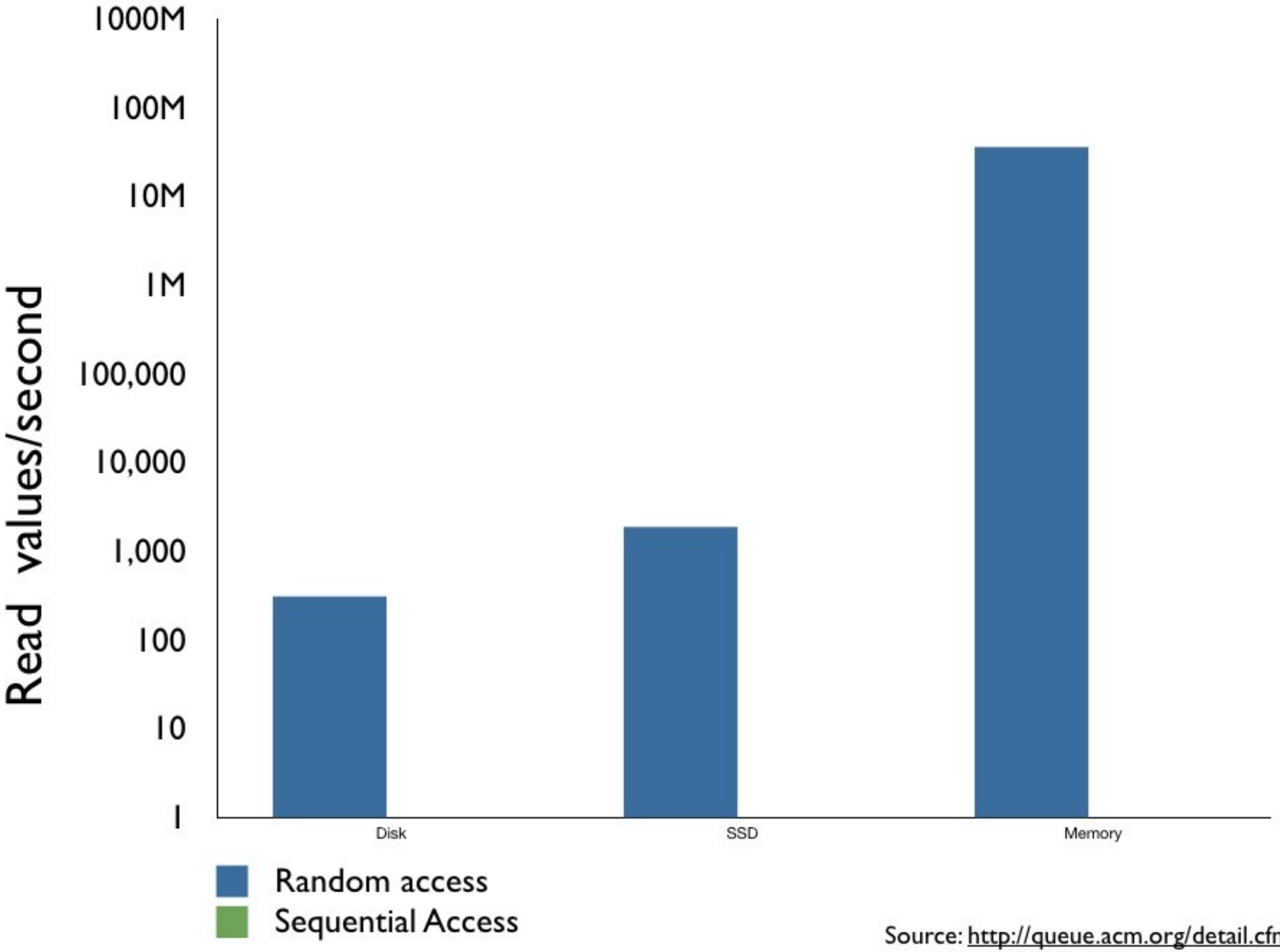

P
P
P
Broker


P
P
P
Broker
C
C

Doesn't Fear the File System

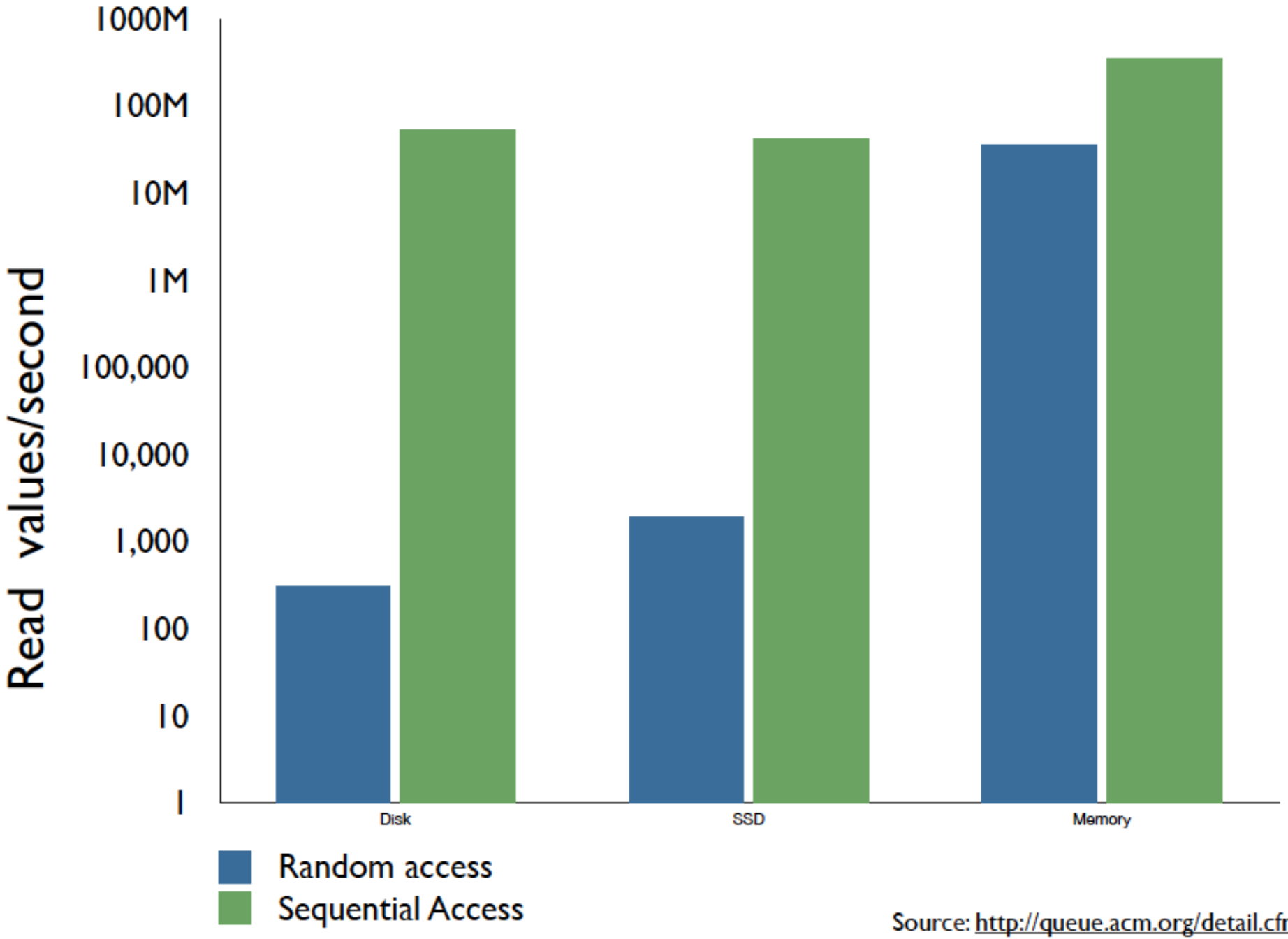


Disk/Memory Performance



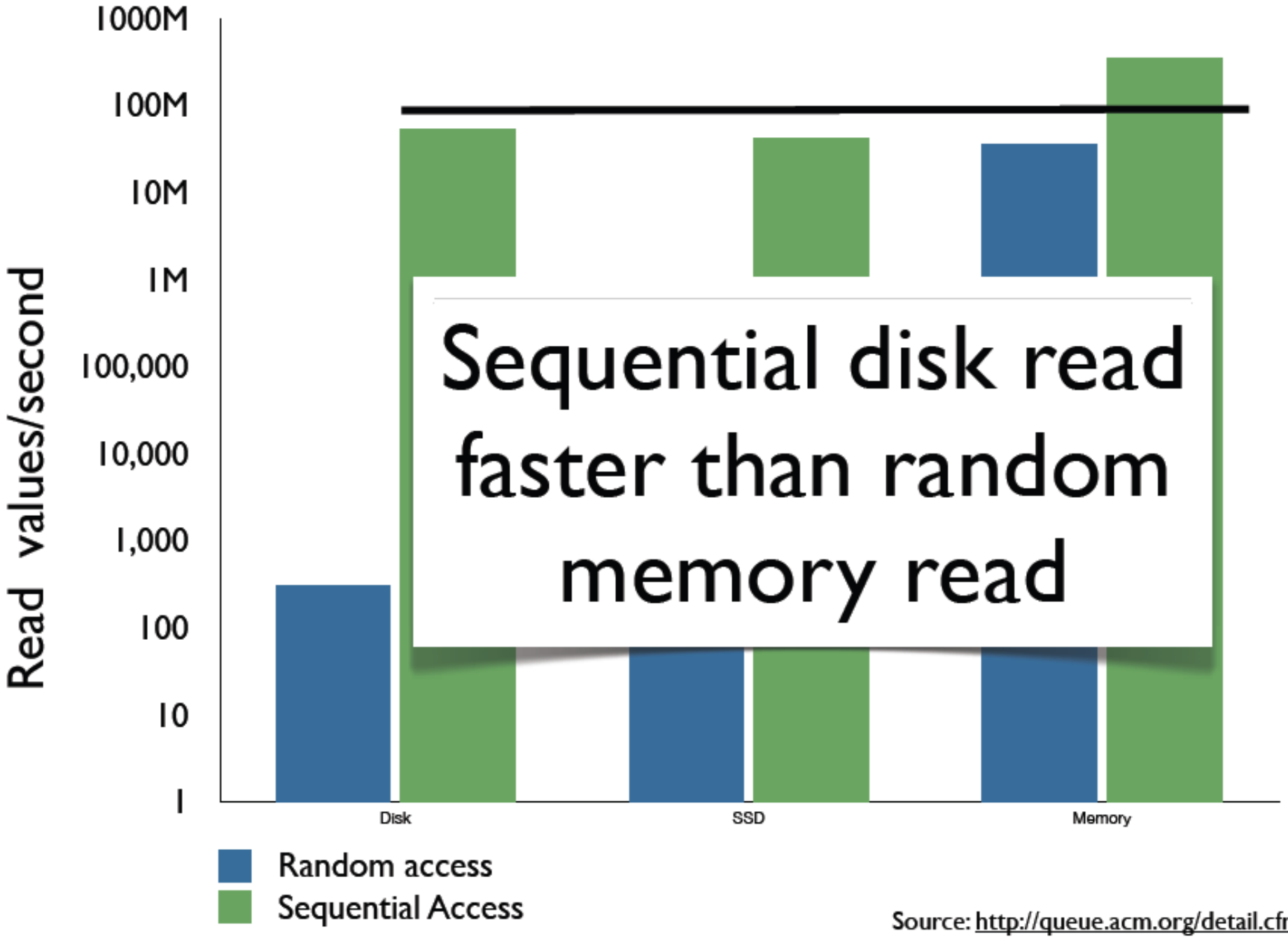
Source: http://queue.acm.org/detail.cfm?id=1563874

Disk/Memory Performance



Source: http://queue.acm.org/detail.cfm?id=1563874

Disk/Memory Performance

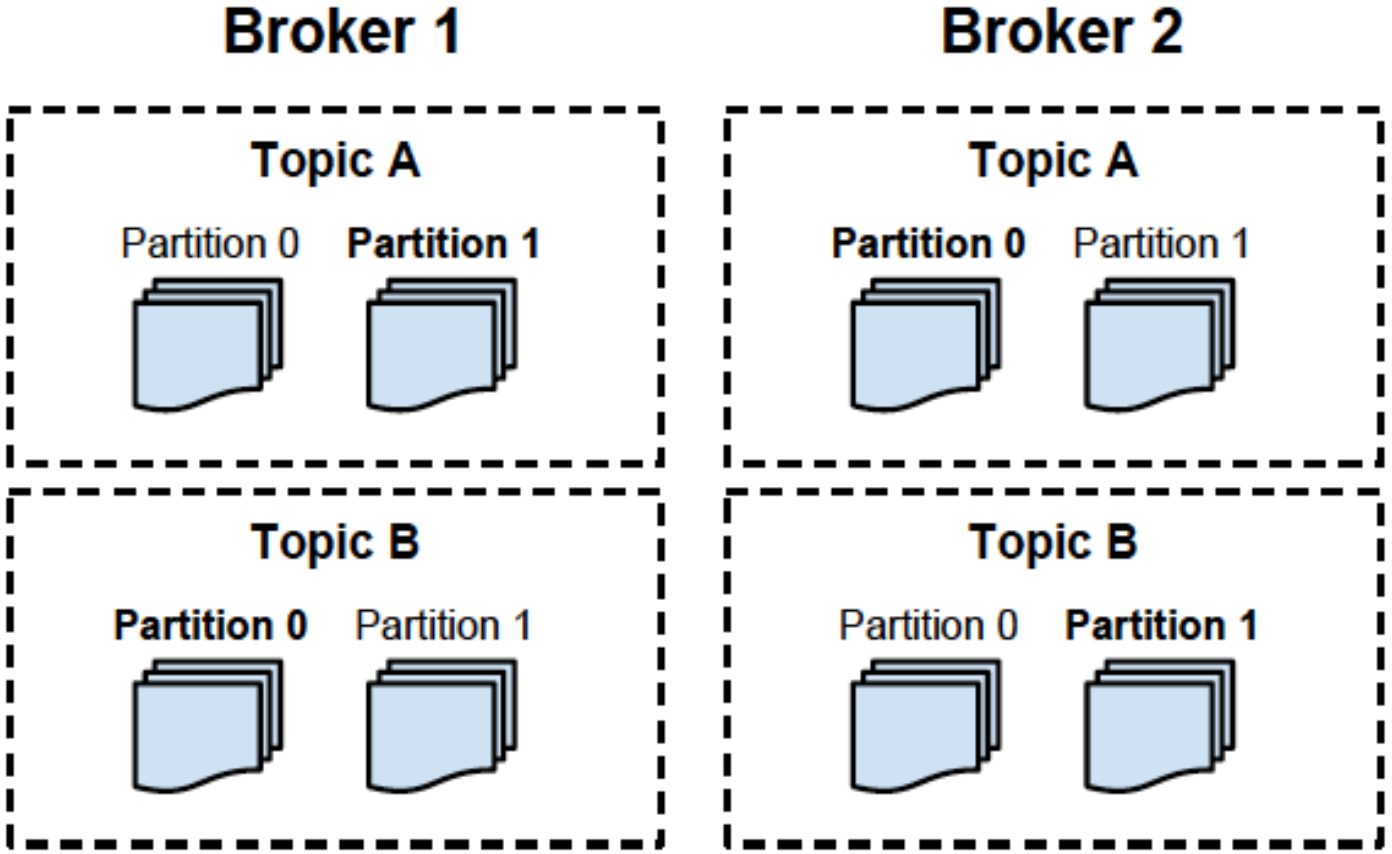


Sequential disk read faster than random memory read

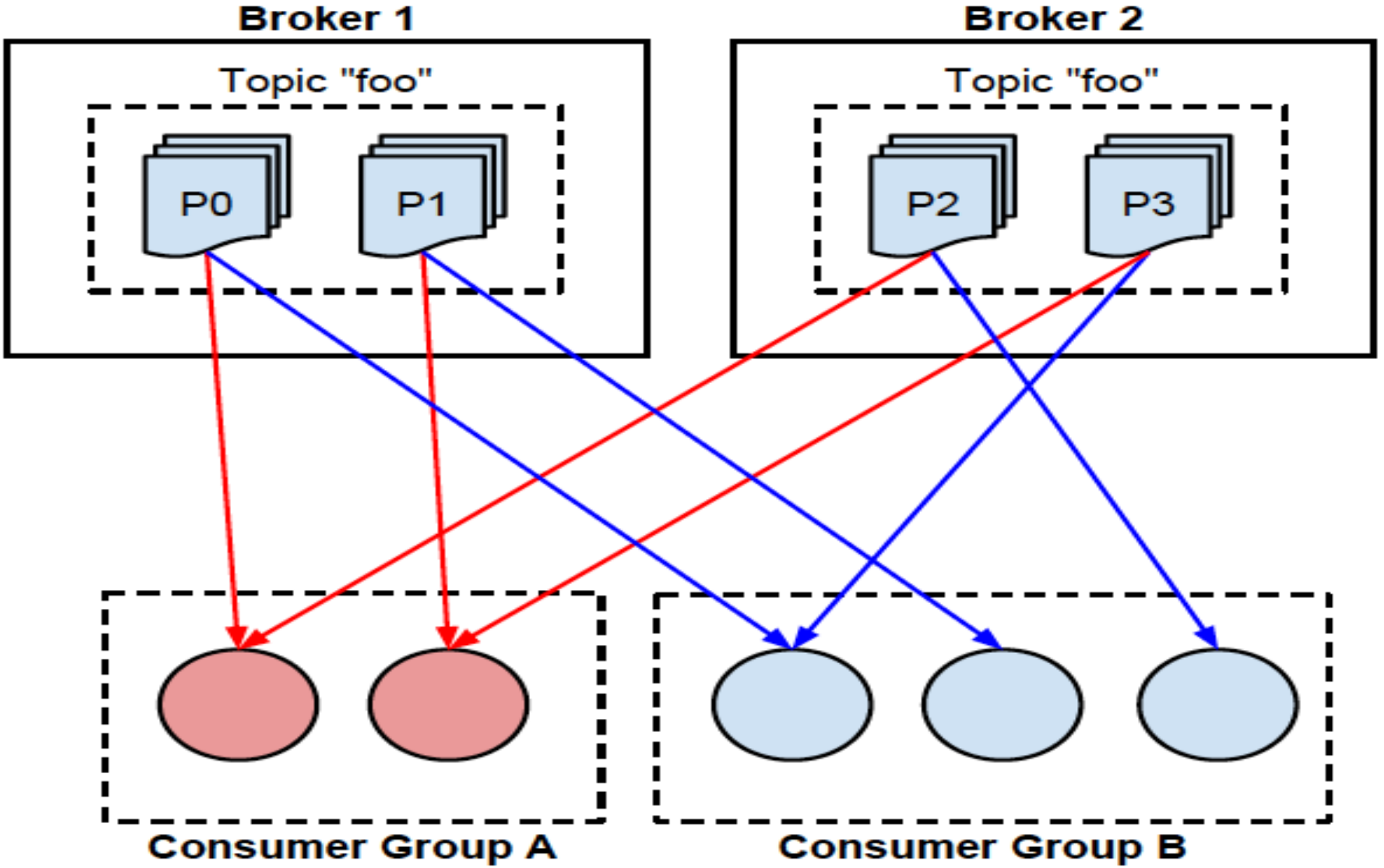
Source: http://queue.acm.org/detail.cfm?id=1563874

Topics

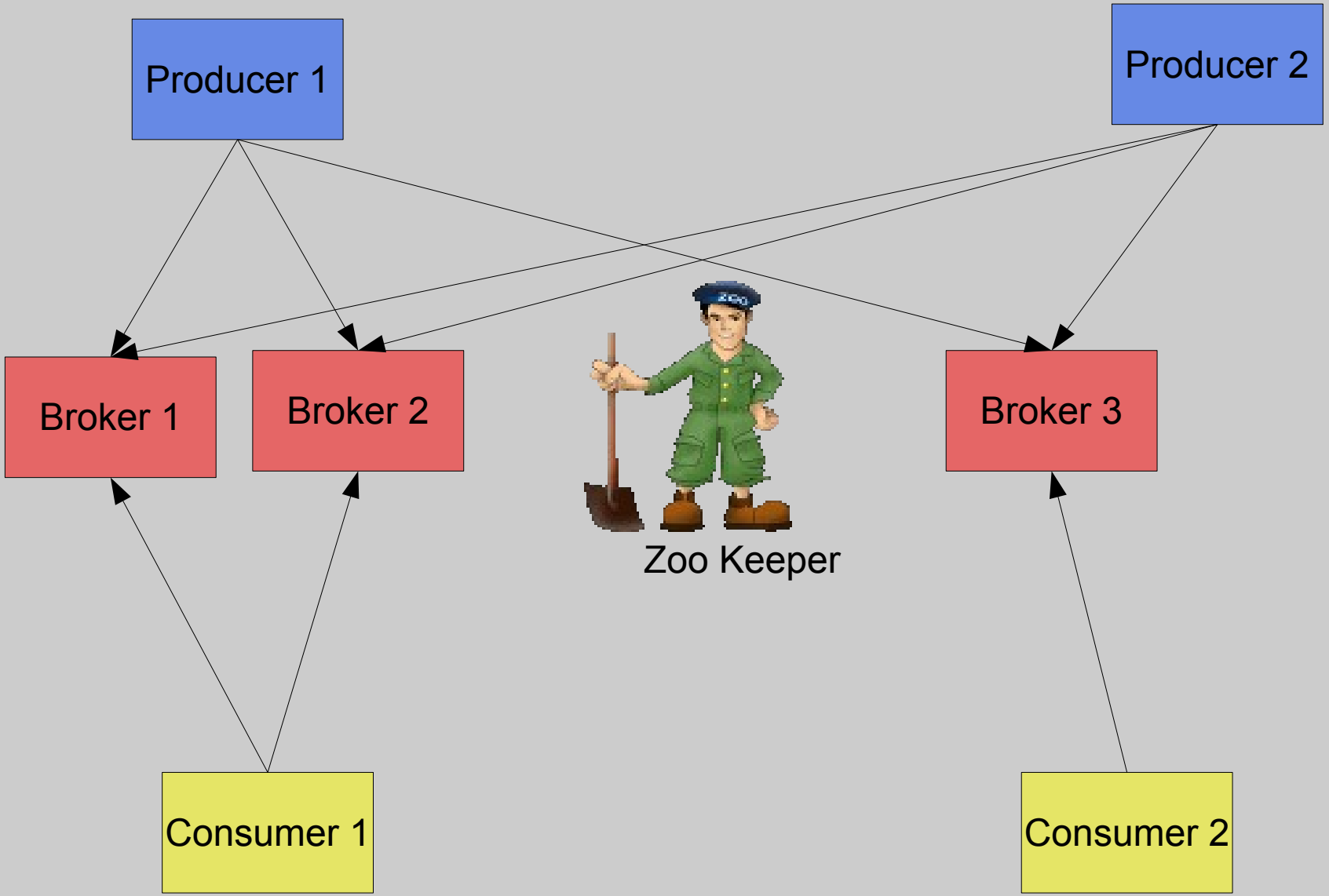
- Logical collections of partitions (the physical f les).
- A broker contains some of the partitions for a topic

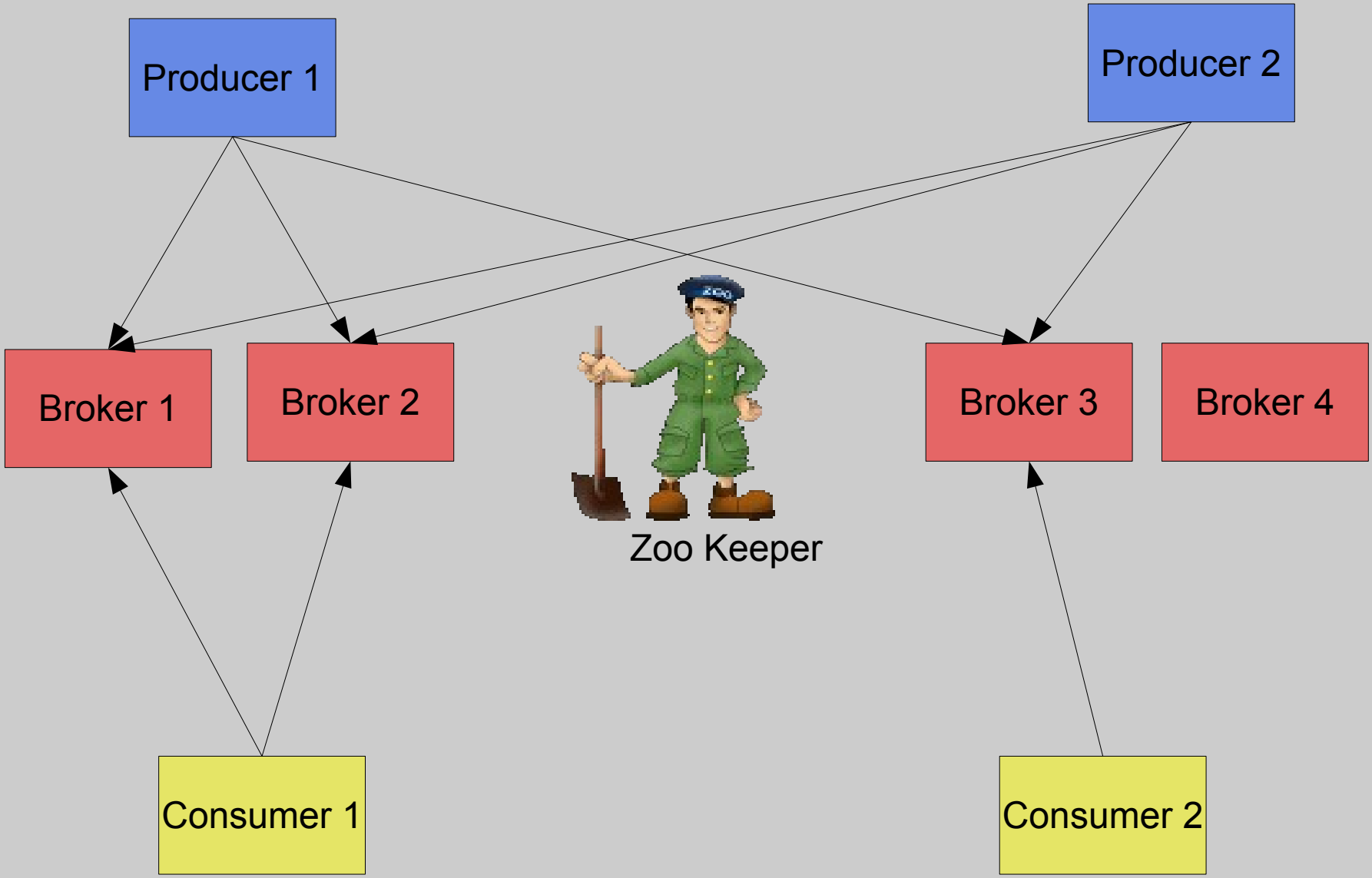


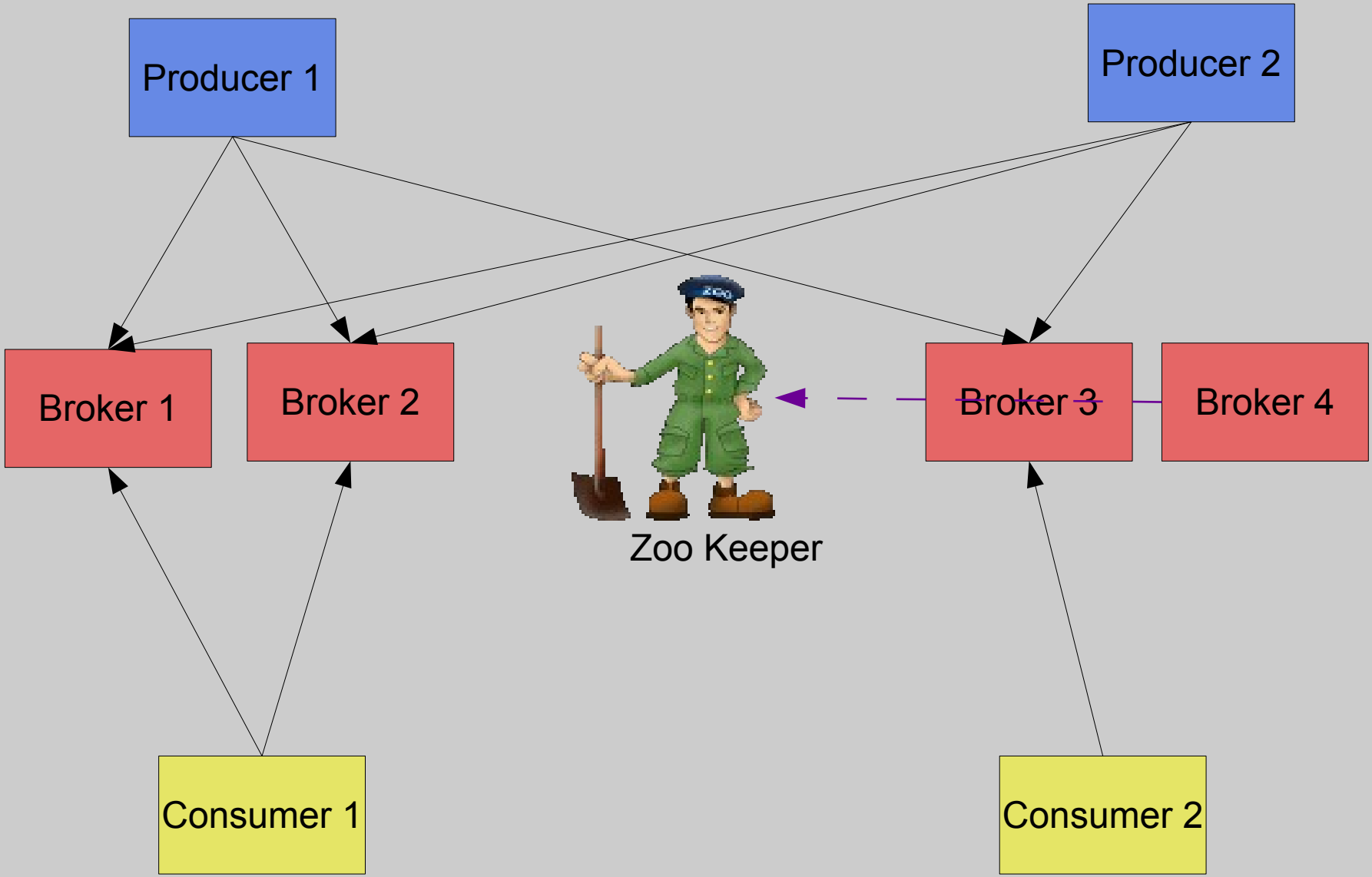
A partition is Consumed by <u>Exactly One</u> Group's Consumer

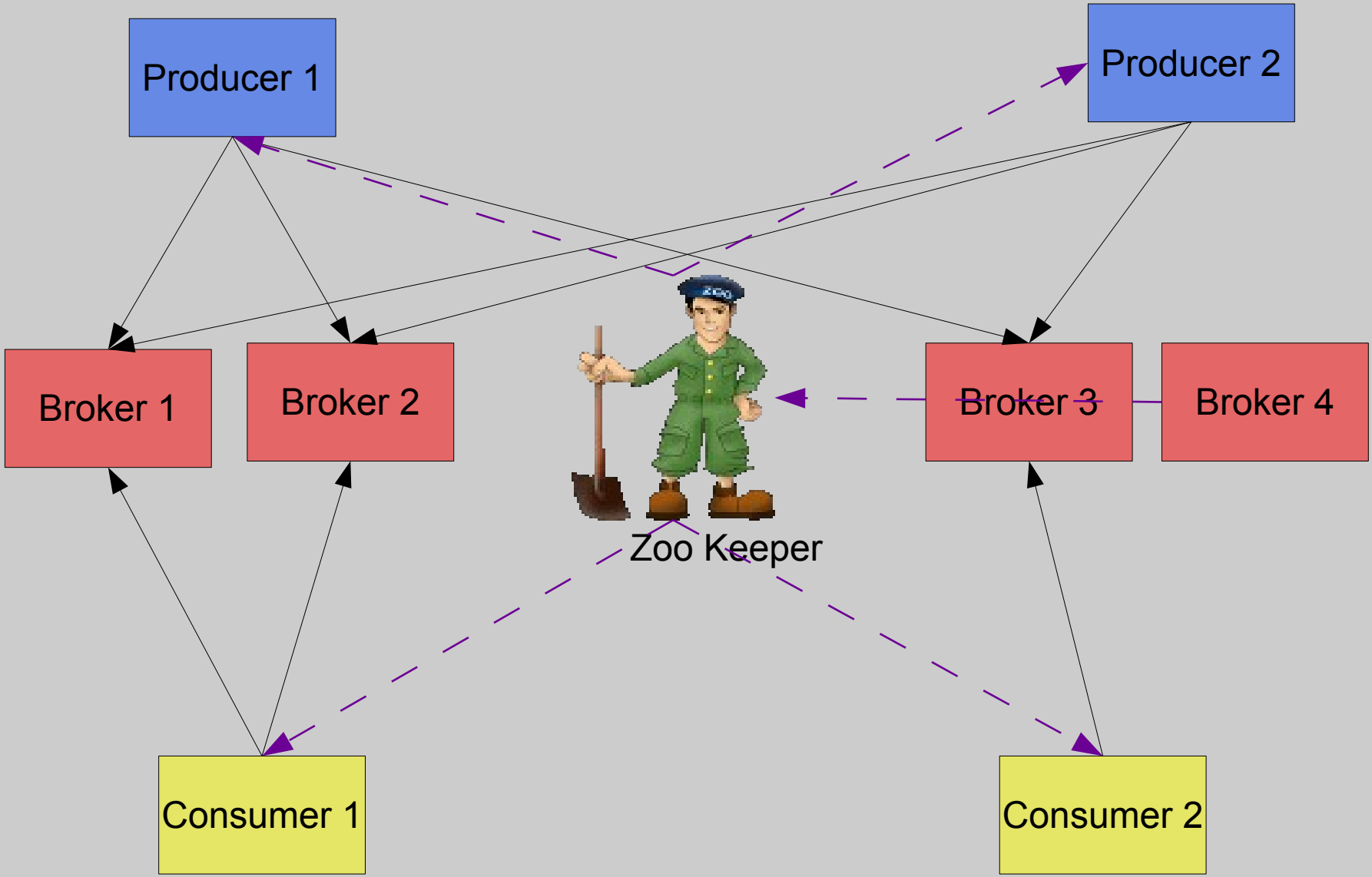


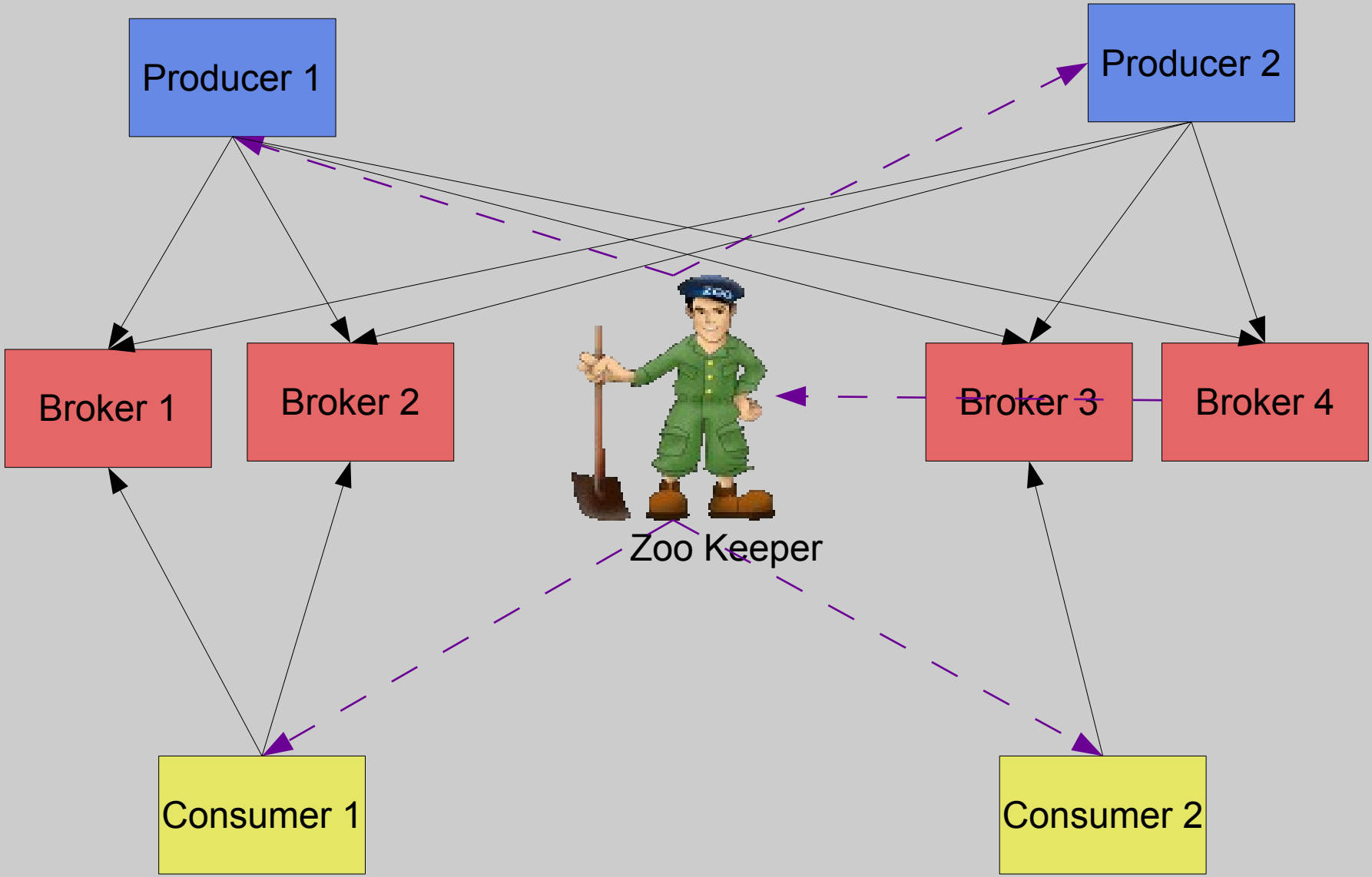
Distributed & Fault-Tolerant

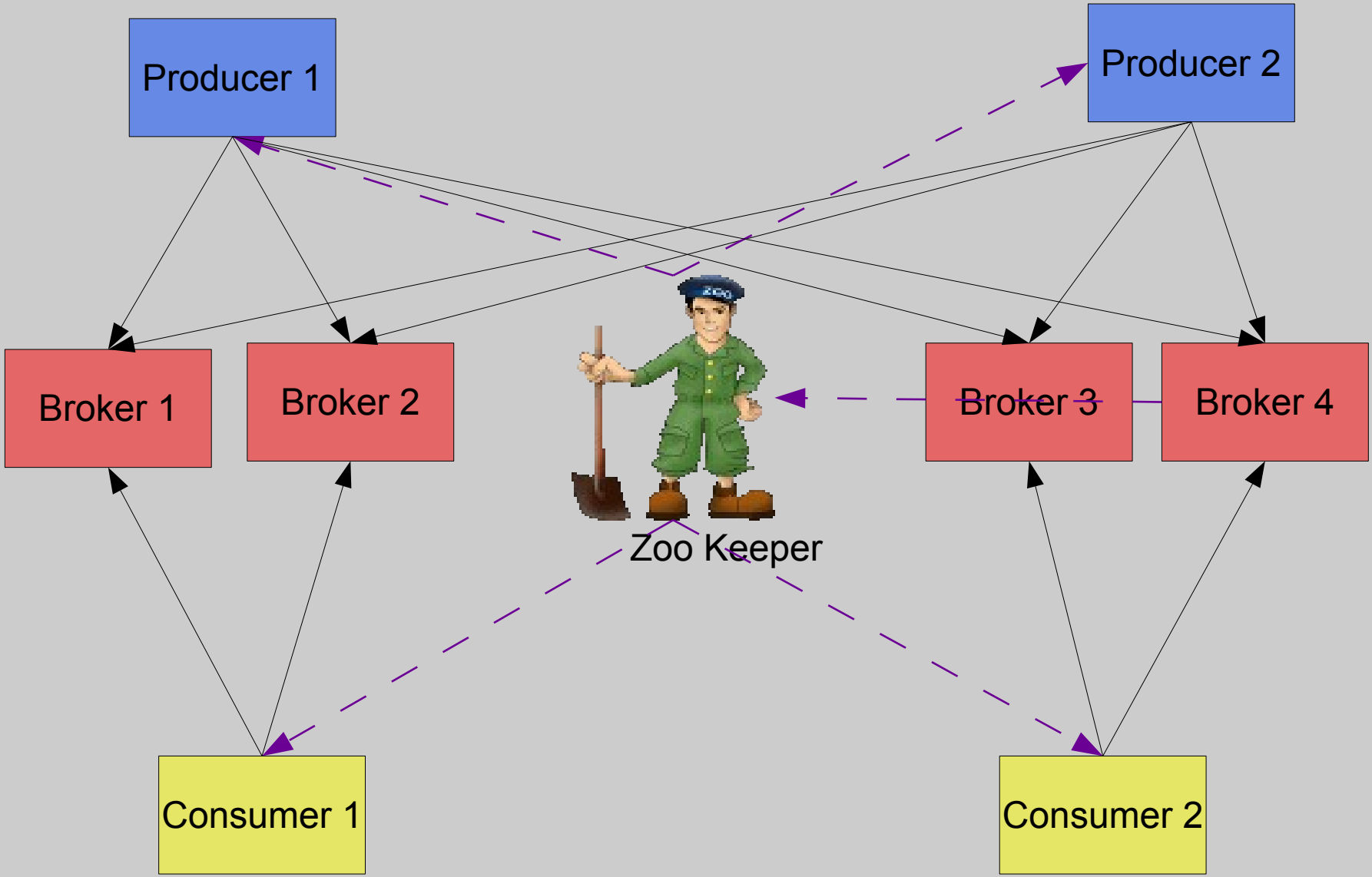

Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Zoo Keeper
Consumer 1
Consumer 2

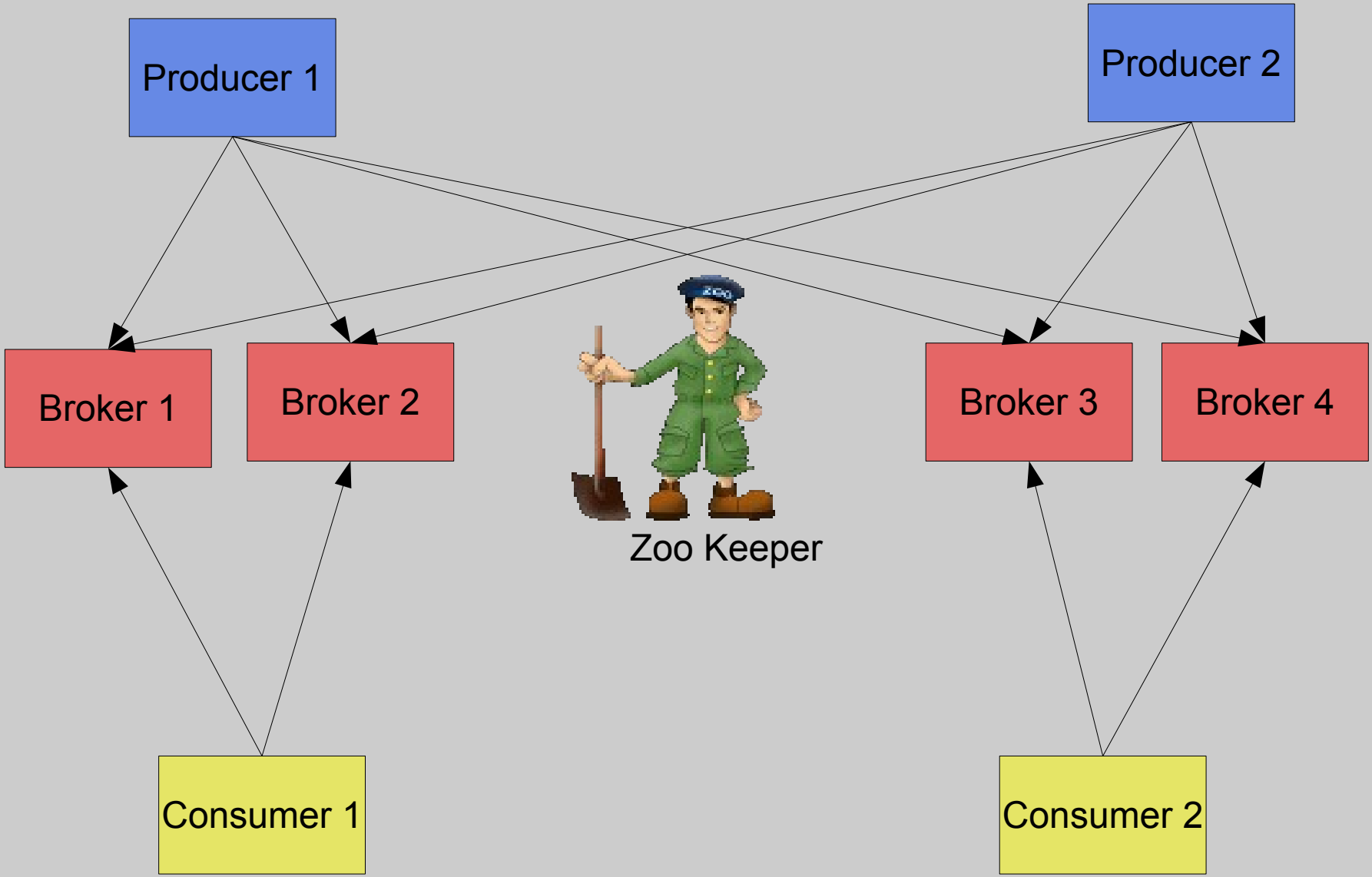

Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Broker 4
Zoo Keeper
Consumer 1
Consumer 2

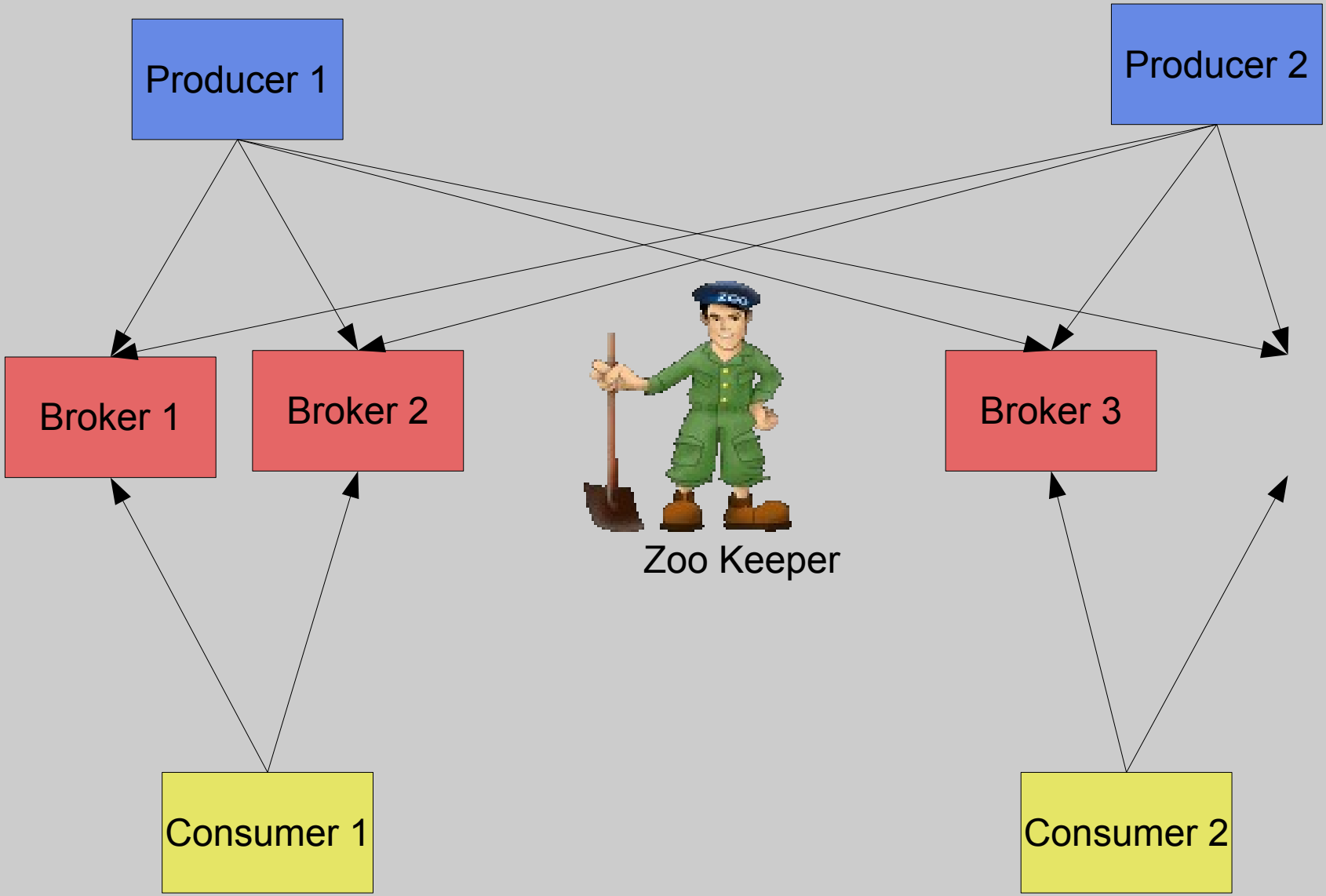

Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Broker 4
Zoo Keeper
Consumer 1
Consumer 2

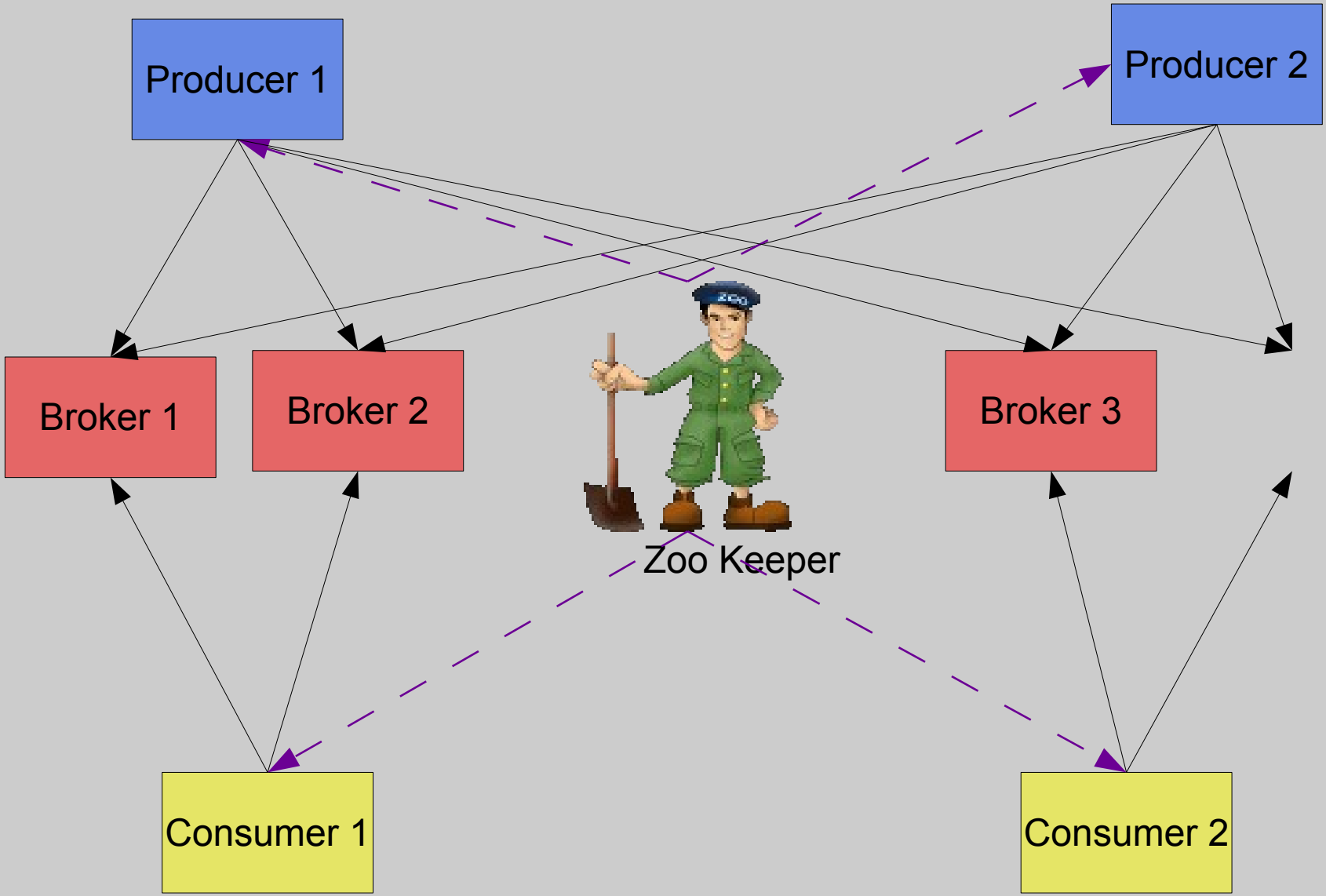

Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Broker 4
Zoo Keeper
Consumer 1
Consumer 2

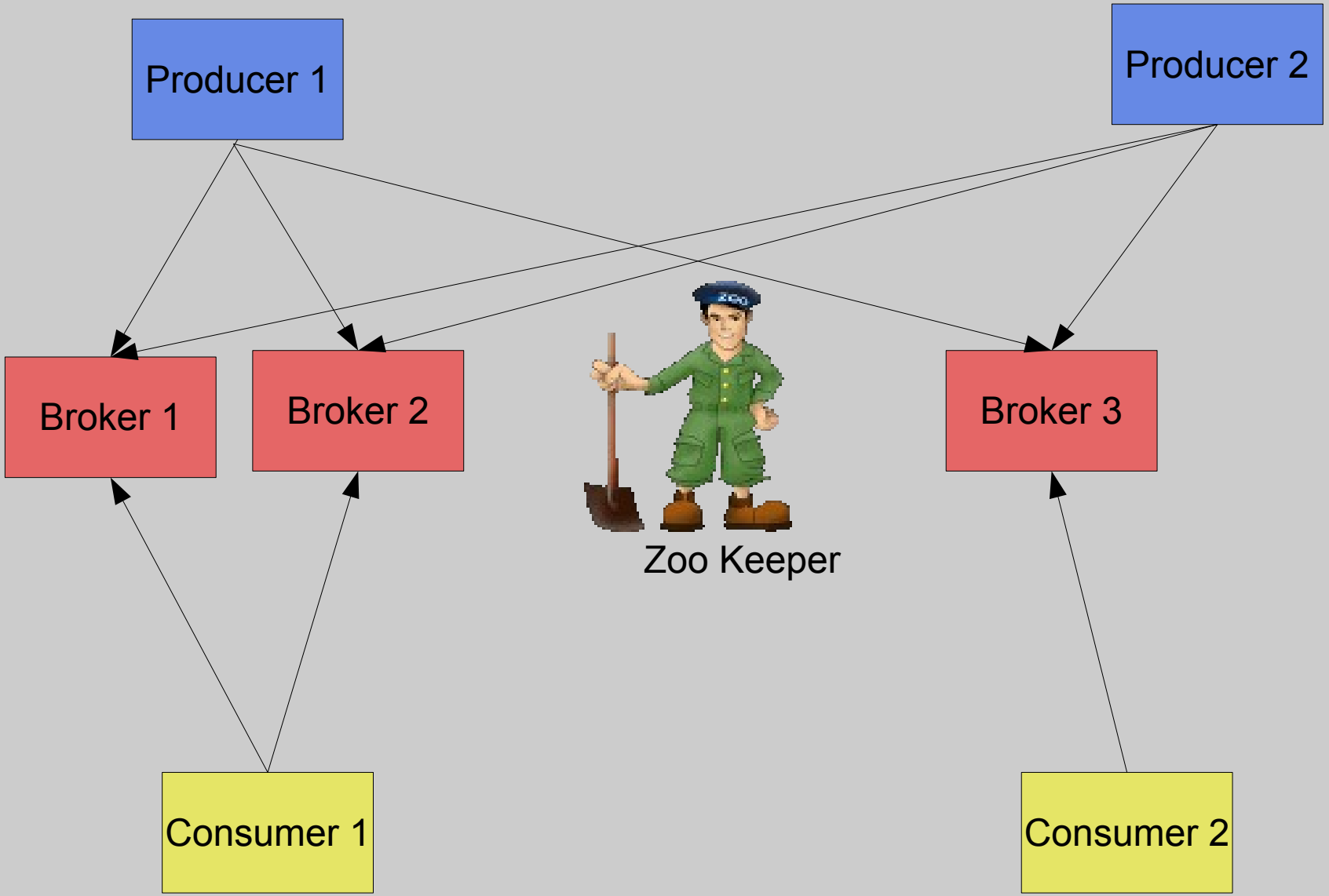

Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Broker 4
Zoo Keeper
Consumer 1
Consumer 2

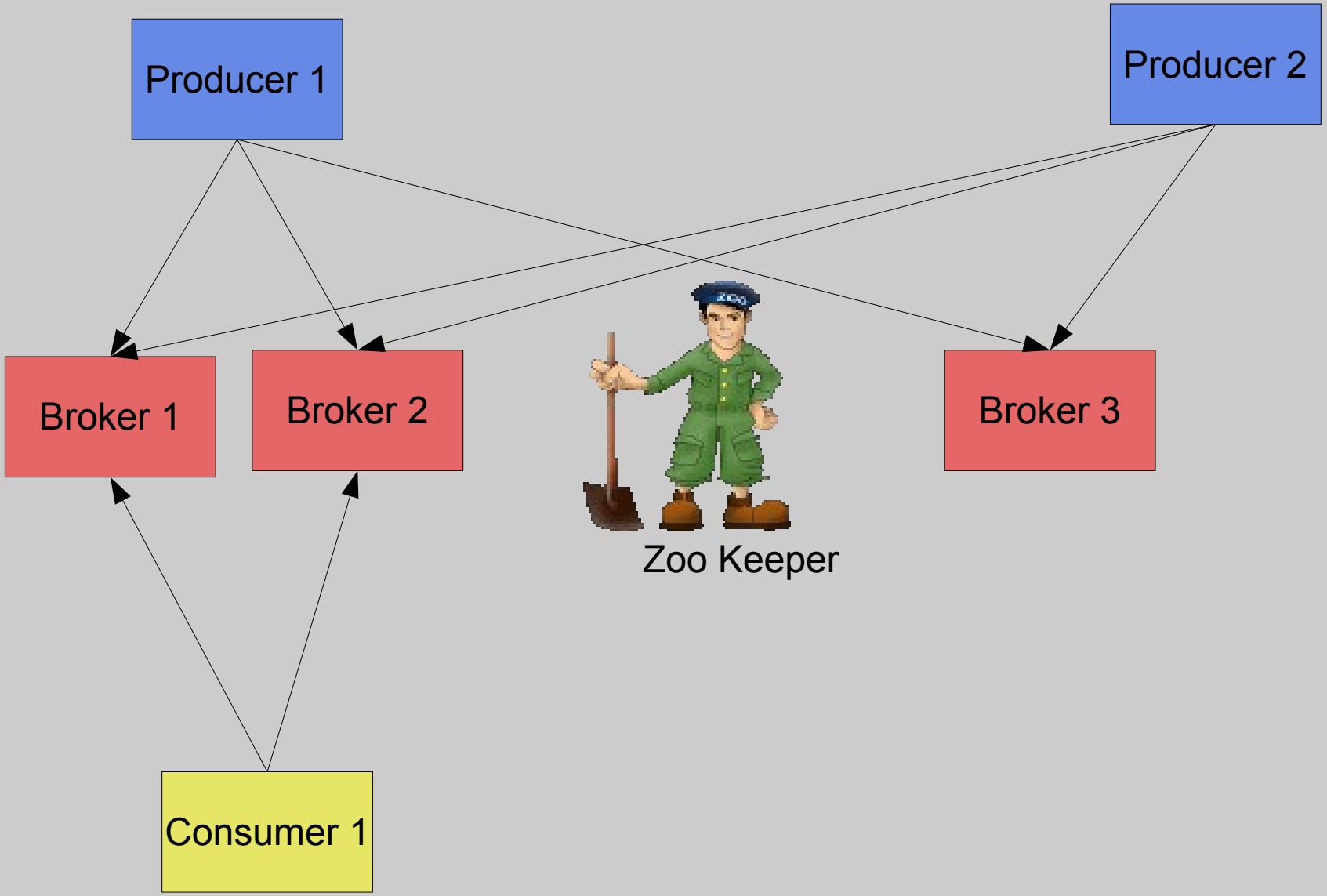

Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Broker 4
Zoo Keeper
Consumer 1
Consumer 2

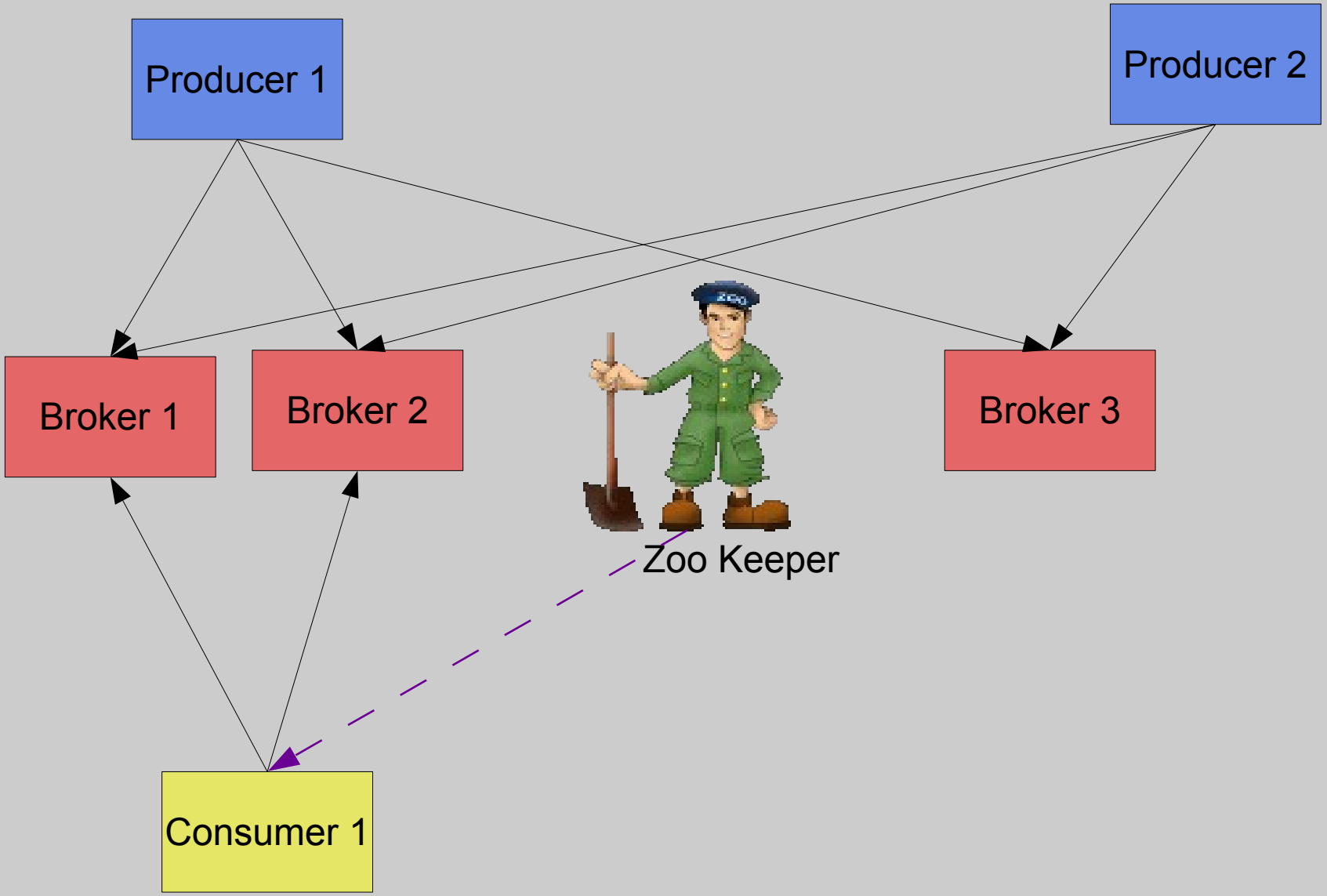

Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Broker 4
Zoo Keeper
Consumer 1
Consumer 2

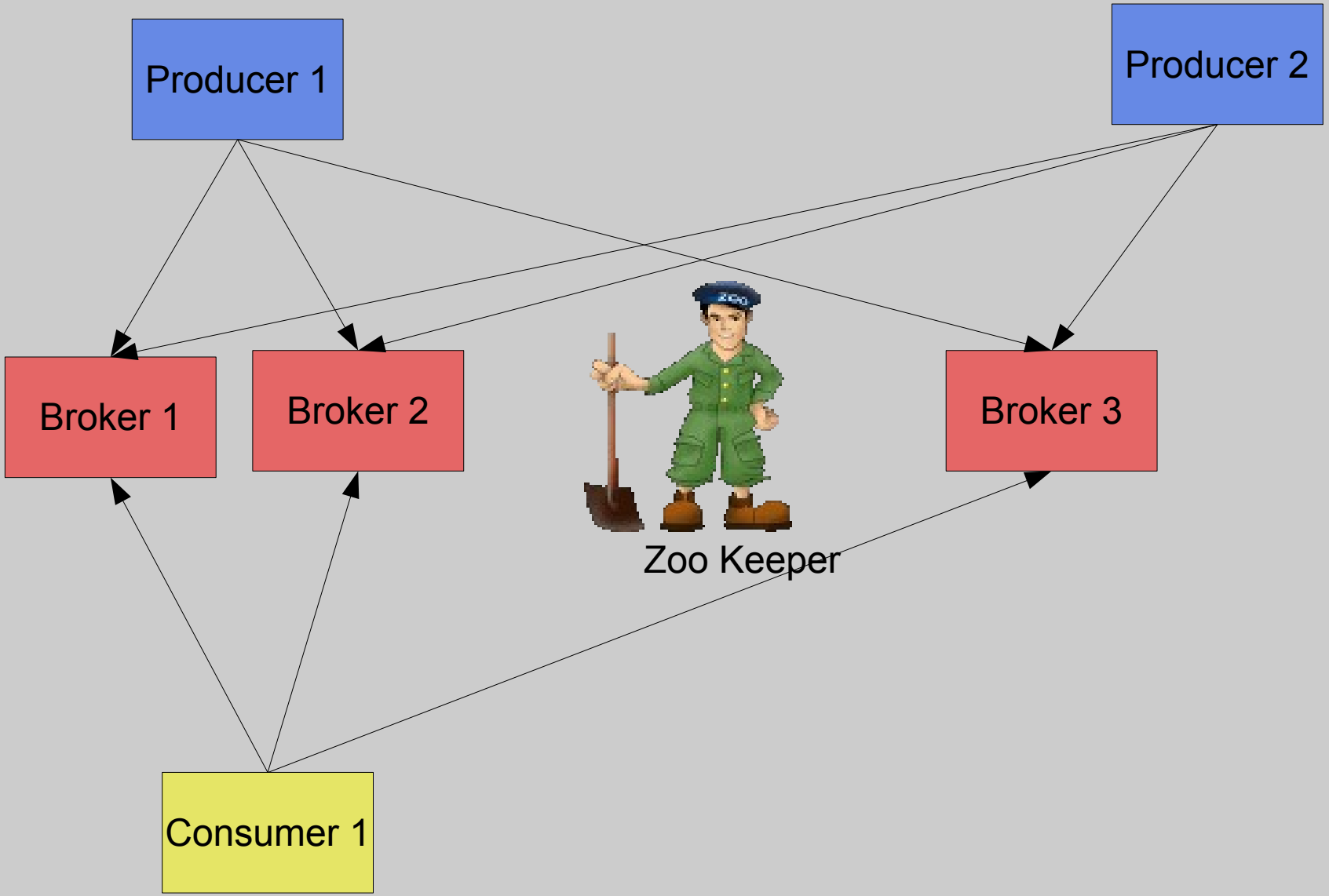

Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Zoo Keeper
Consumer 1
Consumer 2


Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Zoo Keeper
Consumer 1
Consumer 2

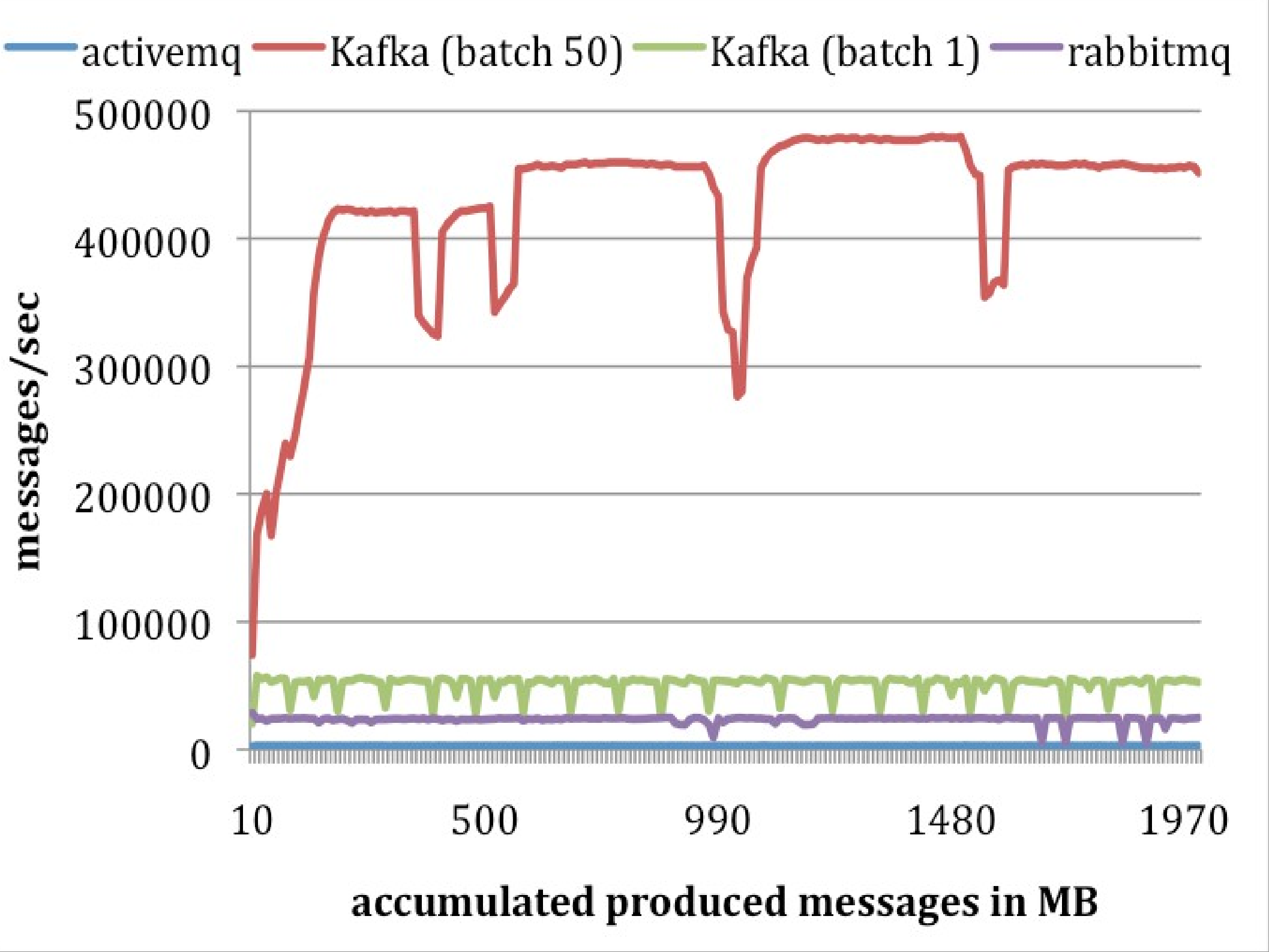

Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Zoo Keeper
Consumer 1
Consumer 2

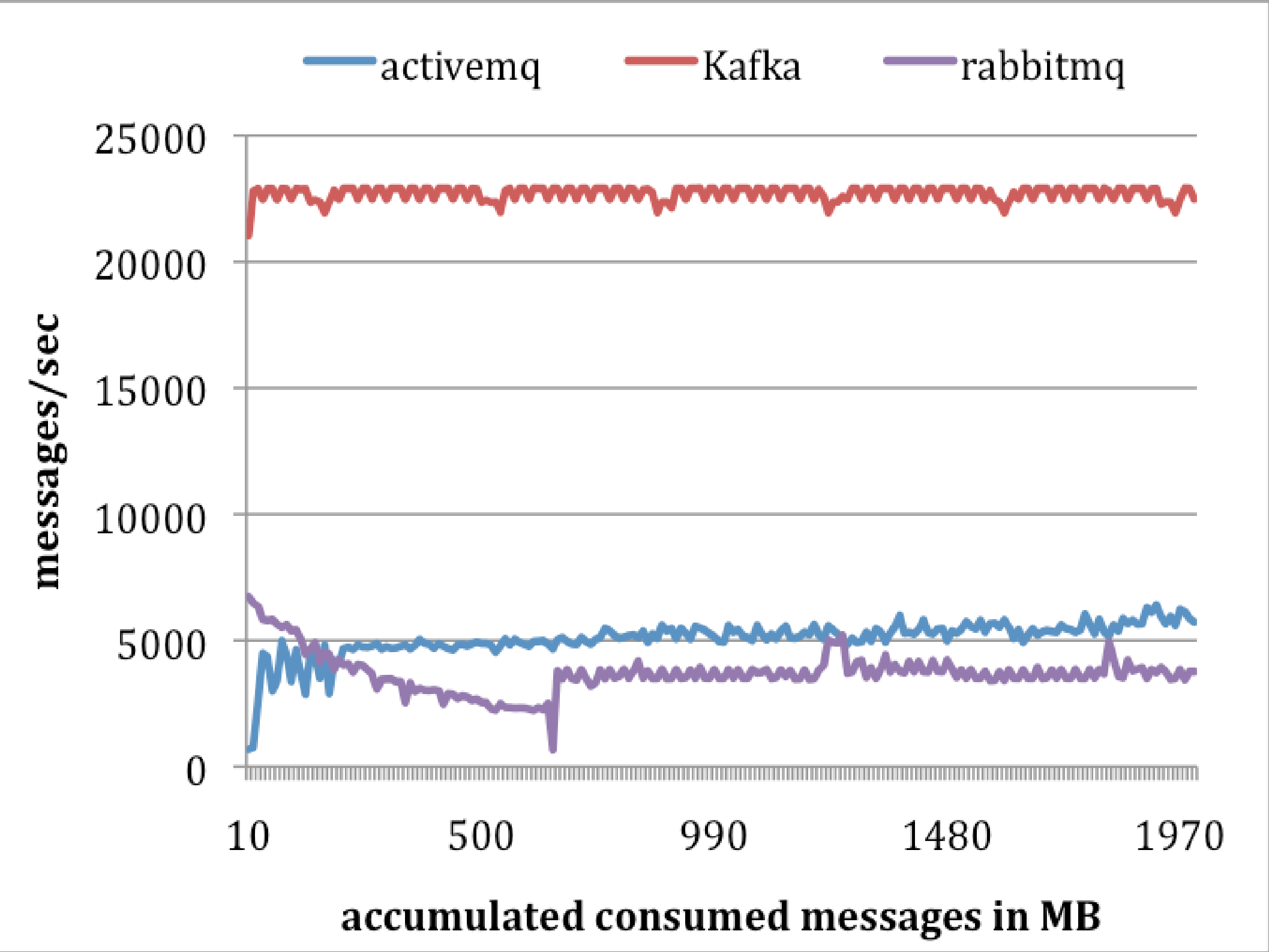

Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Zoo Keeper
Consumer 1


Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Zoo Keeper
Consumer 1


Producer 1
Producer 2
Broker 1
Broker 2
Broker 3
Zoo Keeper
Consumer 1

Performance Benchmark
1 Broker
1 Producer
1 Consumer


activemq
Kafka (batch 50)
Kafka (batch 1)
rabbitmq
500000
400000
300000
200000
100000
0
messages/sec
10
500
990
1480
1970
accumulated produced messages in MB


activemq
Kafka
rabbitmq
25000
20000
15000
10000
5000
0
messages/sec
10
500
990
1480
1970
accumulated consumed messages in MB

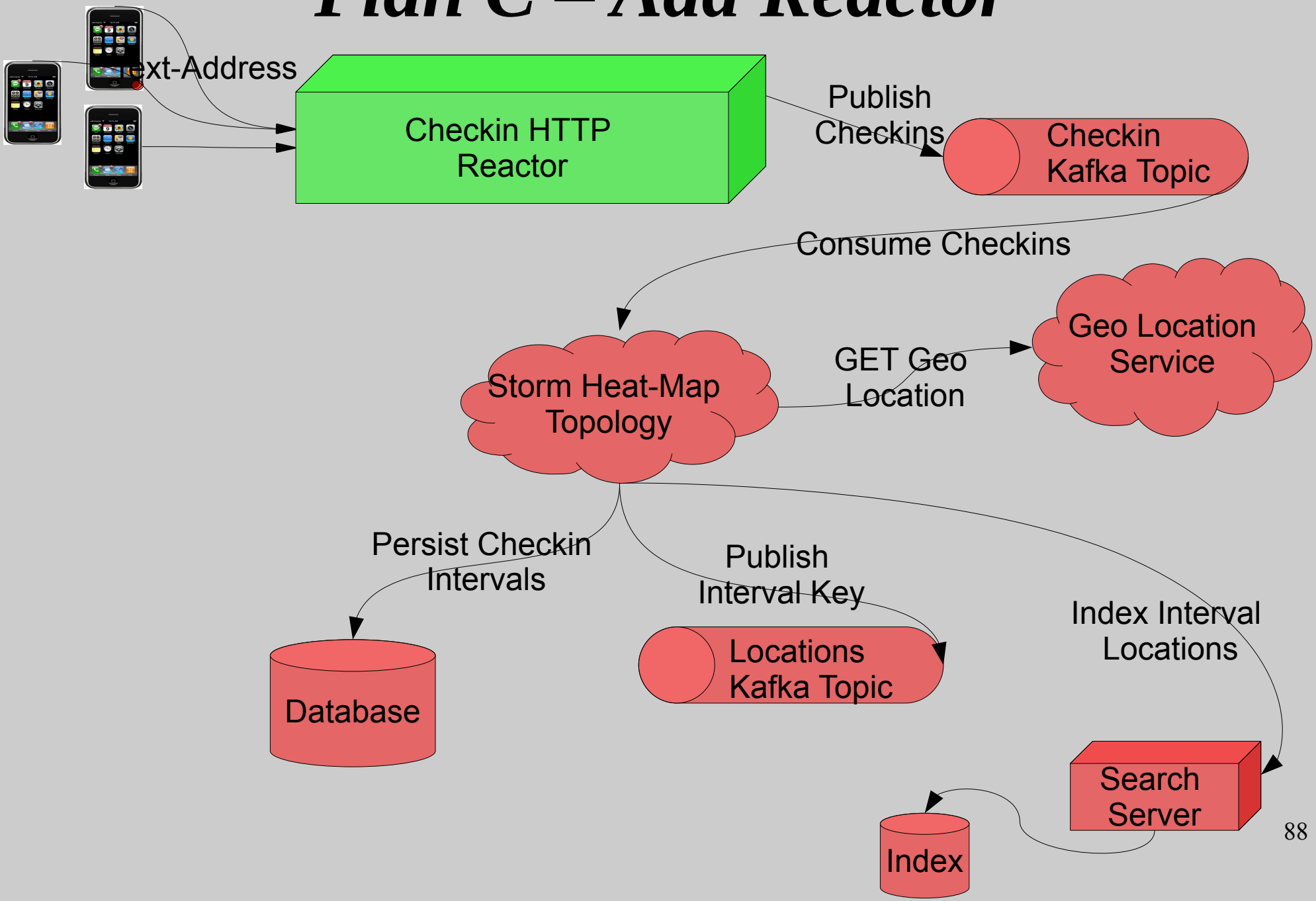
Add Kafka to our Topology

```
public class LocalTopologyRunner {
   ...

 private static TopologyBuilder buildTopolgy() {

         ...
        builder.setSpout("checkins", new KafkaSpout(kafkaConfig));

        ...
        builder.setBolt("kafkaProducer", new KafkaOutputBolt
              (  "localhost:9092",
                 "kafka.serializer.StringEncoder",
                 "locations-topic"))
                .shuffleGrouping("persistor");

         return builder;
    }
}
```

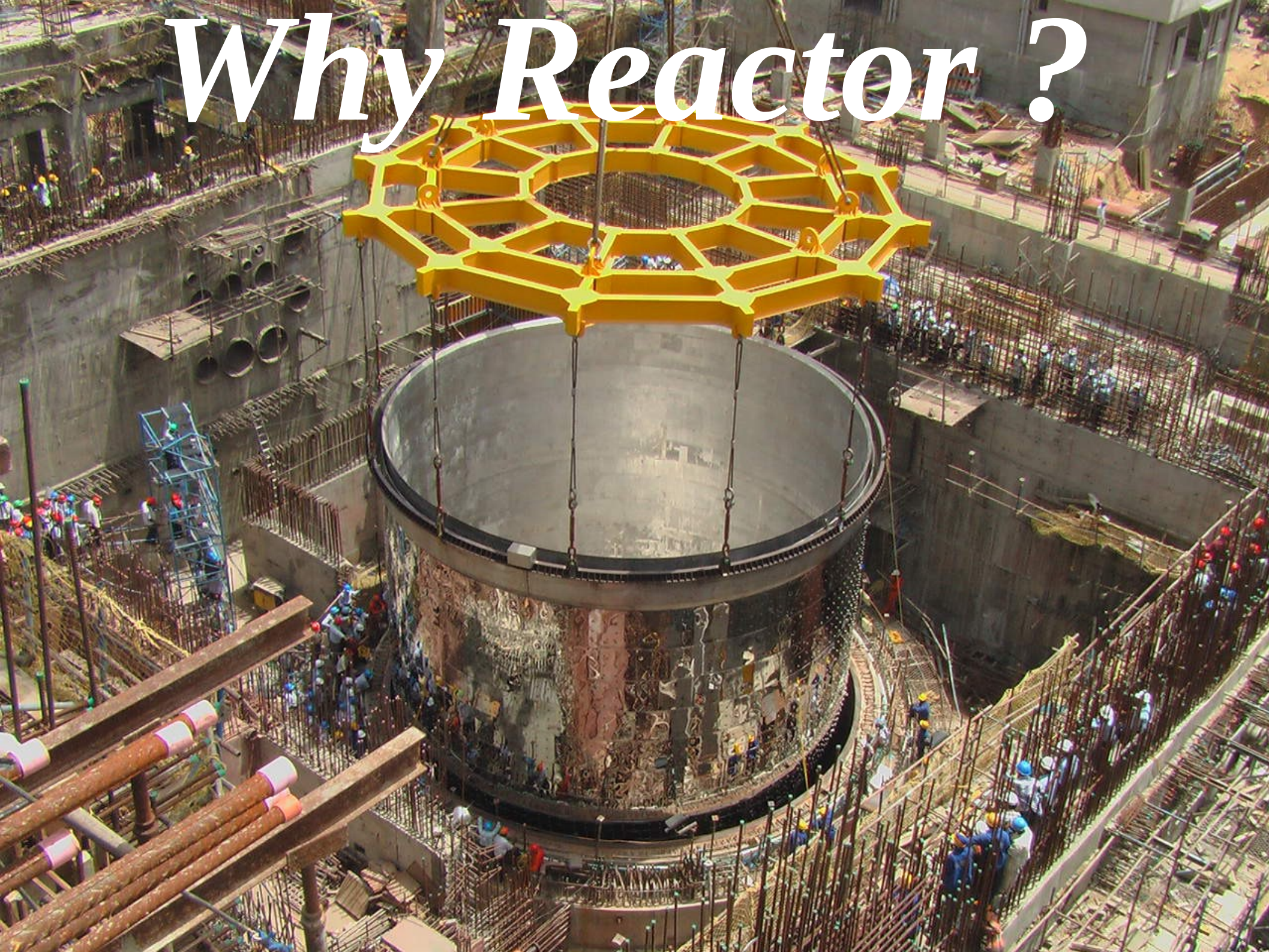
Kafka Spout

Kafka Bolt

Plan C – Add Reactor



Why Reactor ?



C10K Problem

2008:
Thread Per Request/Response

Reactor Pattern Paradigm

Application registers handlers

...events trigger handlers

Reactor Pattern – Key Points

- Single thread / single event loop
- EVERYTHING runs on it
- You MUST NOT block the event loop
- Many Implementations (partial list):
  - Node.js (JavaScrip), EventMachine (Ruby), Twisted (Python)... and Vert.X

Reactor Pattern Problems

- Some work is naturally blocking:
  - Intensive data crunching
  - 3rd-party blocking API’s (e.g. JDBC)
- Pure reactor (e.g. Node.js) is not a good f t for this kind of work!

Vert.x is a lightweight, high performance application platform for the JVM that's designed for modern mobile, web, and enterprise applications.

Polyglot

Write your application components in Java, JavaScript, CoffeeScript, Ruby, Python or Groovy...

... or mix and match several programming languages in a single app.

Simplicity

...without being simplistic.

Simple, powerful, APIs enable you to write non-blocking network enabled applications with ease. No complex configuration or boilerplate required.

Scalability

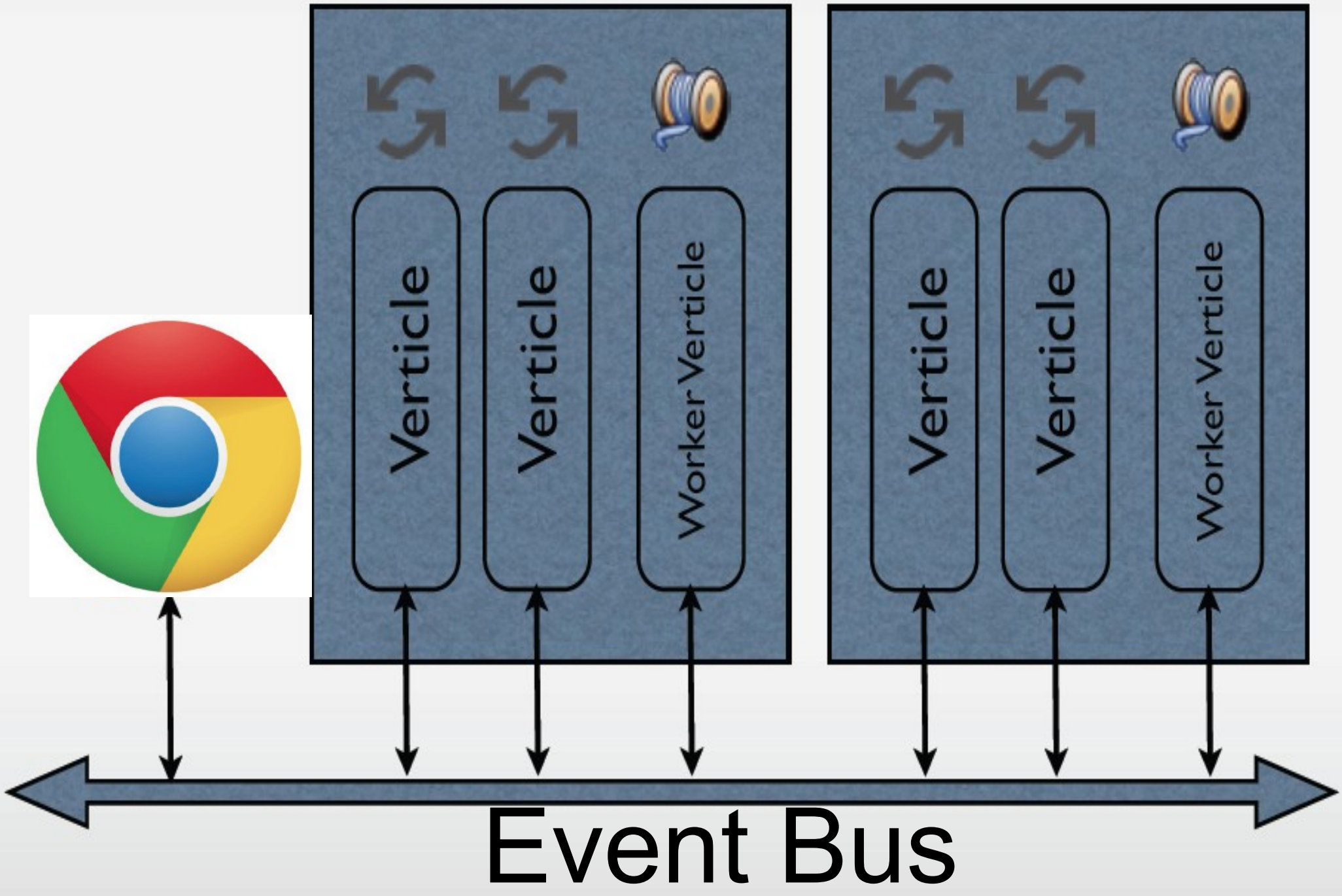
Scales using messaging passing to efficiently utilise your server cores.

Uses non blocking I/O to serve many connections with minimal threads.

Concurrency

Simple *actor-like* concurrenc model frees you from the pitf of traditional multi-threaded programming.

Vert.X Architecture



Vert.X Goodies

- Growing Module
- Repository
- web server
- Persistors (Mongo, JDBC, ...)
- Work queue
- Authentication
- Manager
- Session manager
- Socket.IO
- TCP/SSL servers/clients
- HTTP/HTTPS servers/ clients
- WebSockets support
- SockJS support
- Timers
- Buffers
- Streams and Pumps
- Routing
- Asynchronous File I/O

Node.JS vs Vert.X

Node.js vs Vert.X

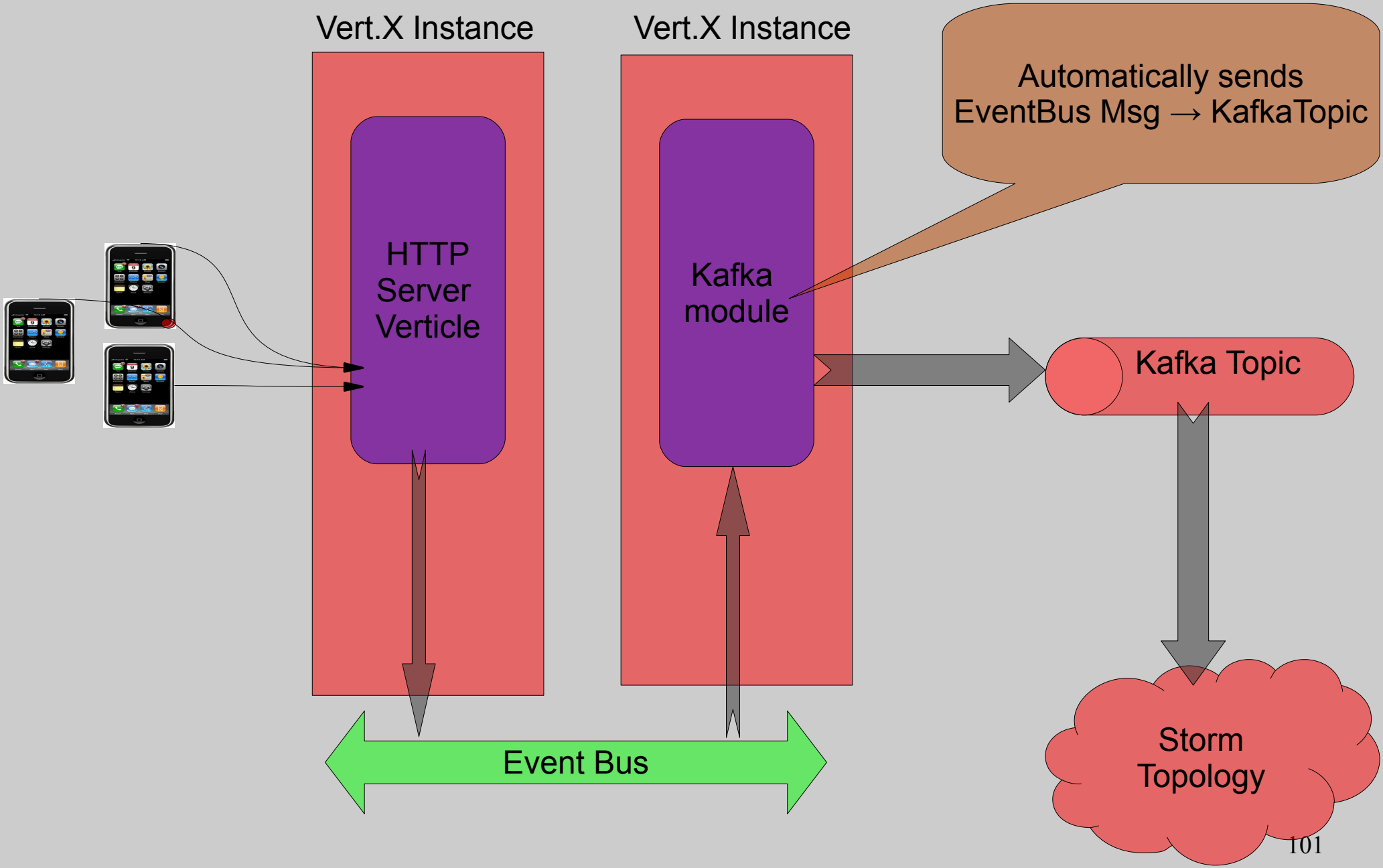
- Node.js
  - JavaScript Only
  - Inherently Single Threaded
  - No help much with IPC
  - All code MUST be in Event loop
- **Vert.X**
  - Polyglot (JavaScript, Java, Ruby, Python...)
  - Leverages JVM multi-threading
  - Nervous Event Bus
  - Blocking work can be done off the event loop

Node.js vs Vert.X Benchmark

http://vertxproject.wordpress.com/2012/05/09/vert-x-vs-node-js-simple-http-benchmarks/

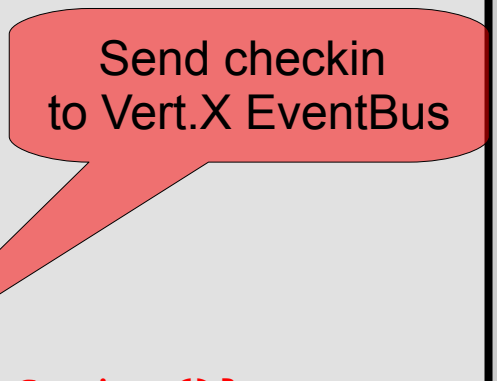
AMD Phenom II X6 (6 core), 8GB RAM, Ubuntu 11.04

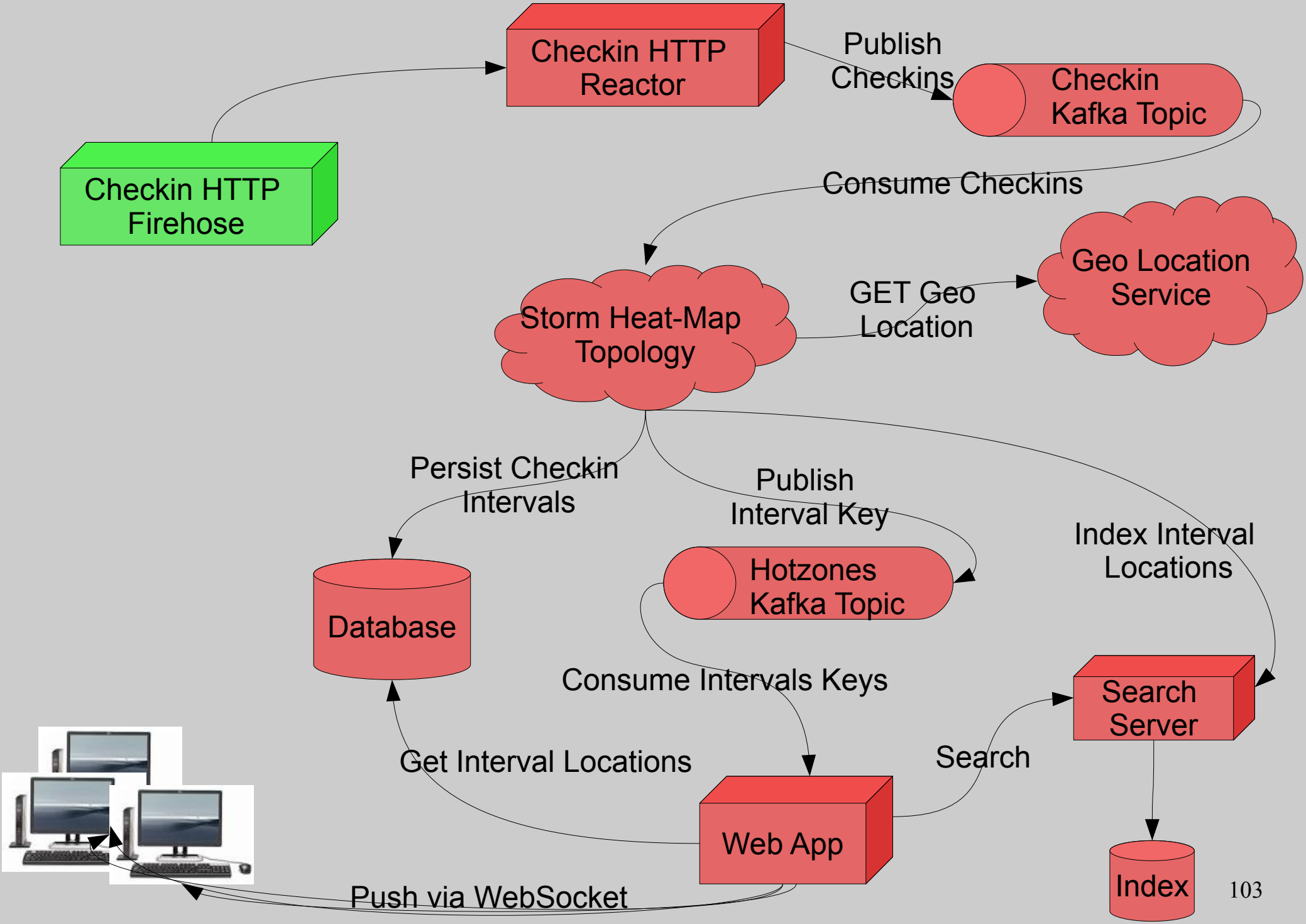
HeatMap Reactor Architecture



Heat-Map Server – Only 6 LOC !

```
var vertx = require('vertx');
var container = require('vertx/container');
var console = require('vertx/console');
var config = container.config;



vertx.createHttpServer().requestHandler(function(request) {
    request.dataHandler(function(buffer) {
        vertx.eventBus.send(config.address, {payload:buffer.toString()});
    });
    request.response.end();
}).listen(config.httpServerPort, config.httpServerHost);

console.log("HTTP CheckinsReactor started on port "+config.httpServerPort);
```




Checkin HTTP Reactor
Publish Checkins
Checkin Kafka Topic
Checkin HTTP Firehose
Consume Checkins
Geo Location Service
GET Geo Location
Storm Heat-Map Topology
Persist Checkin Intervals
Publish Interval Key
Index Interval Locations
Hotzones Kafka Topic
Database
Consume Intervals Keys
Search Server
Get Interval Locations
Search
Web App
Index
Push via WebSocket


Funmunch.com
Demo

Summary

When You go out to Salsa Club

- Good Music
- Crowded

More Conclusions..

- Storm – Great for real-time BigData processing. Complementary for Hadoop batch jobs.
- Kafka – Great messaging for logs/events data, been served as a good “source” for Storm spout
- Vert.X – Worth trial and check as an alternative for reactor.


Thanks