The Future of Apache Storm

Hadoop Summit 2016 - Dublin

P. Taylor Goetz, Hortonworks
@ptgoetz

About Me

- Tech Staff @ Hortonworks
- PMC Chair, Apache Storm
- ASF Member
- PMC, Apache Incubator
- PMC, Apache Arrow
- PMC, Apache Kylin
- Mentor/PPMC, Apache Eagle (Incubating)
- Mentor/PPMC, Apache Metron (Incubating)
- Mentor/PPMC, Apache Apex (Incubating)

Apache Storm 0.9.x

Storm moves to Apache

Apache Storm 0.9.x

- First official Apache Release
- Storm becomes an Apache TLP
- 0mq to Netty for inter-worker communication
- Expanded Integration (Kafka, HDFS, HBase)
- Dependency conflict reduction (It was a start ;))

Apache Storm 0.10.x

Enterprise Readiness

Apache Storm 0.10.x

- Security, Multi-Tenancy
- Enable Rolling Upgrades
- Flux (declarative topology wiring/configuration)
- Partial Key Groupings

Apache Storm 0.10.x

- Improved logging (Log4j 2)
- Streaming Ingest to Apache Hive
- Azure Event Hubs Integration
- Redis Integration
- JDBC Integration

Apache Storm 1.0

Maturity and Improved Performance

Release Date: April 12, 2016

Pacemaker

Heartbeat Server

Pacemaker

- Replaces Zookeeper for Heartbeats
- In-Memory key-value store
- Allows Scaling to 2k-3k+ Nodes
- Secure: Kerberos/Digest Authentication

Pacemaker

- Compared to Zookeeper:
  - Less Memory/CPU
  - No Disk
  - Spared the overhead of maintaining consistency

Distributed Cache API

Distributed Cache API

- Topology resources:
  - Dictionaries, ML Models, Geolocation Data, etc.
- Typically packaged in topology jar
  - Fine for small files
  - Large files negatively impact topology startup time
  - Immutable: Changes require repackaging and deployment

Distributed Cache API

- Allows sharing of files (BLOBs) among topologies
- Files can be updated from the command line
- Allows for files from several KB to several GB in size
- Files can change over the lifetime of the topology
- Allows for compression (e.g. zip, tar, gzip)

Distributed Cache API

- Two implementations: LocalFsBlobStore and HdfsBlobStore
- Local implementation supports Replication Factor (not needed for HDFS-backed implementation)
- Both support ACLs

Distributed Cache API

Creating a blob:

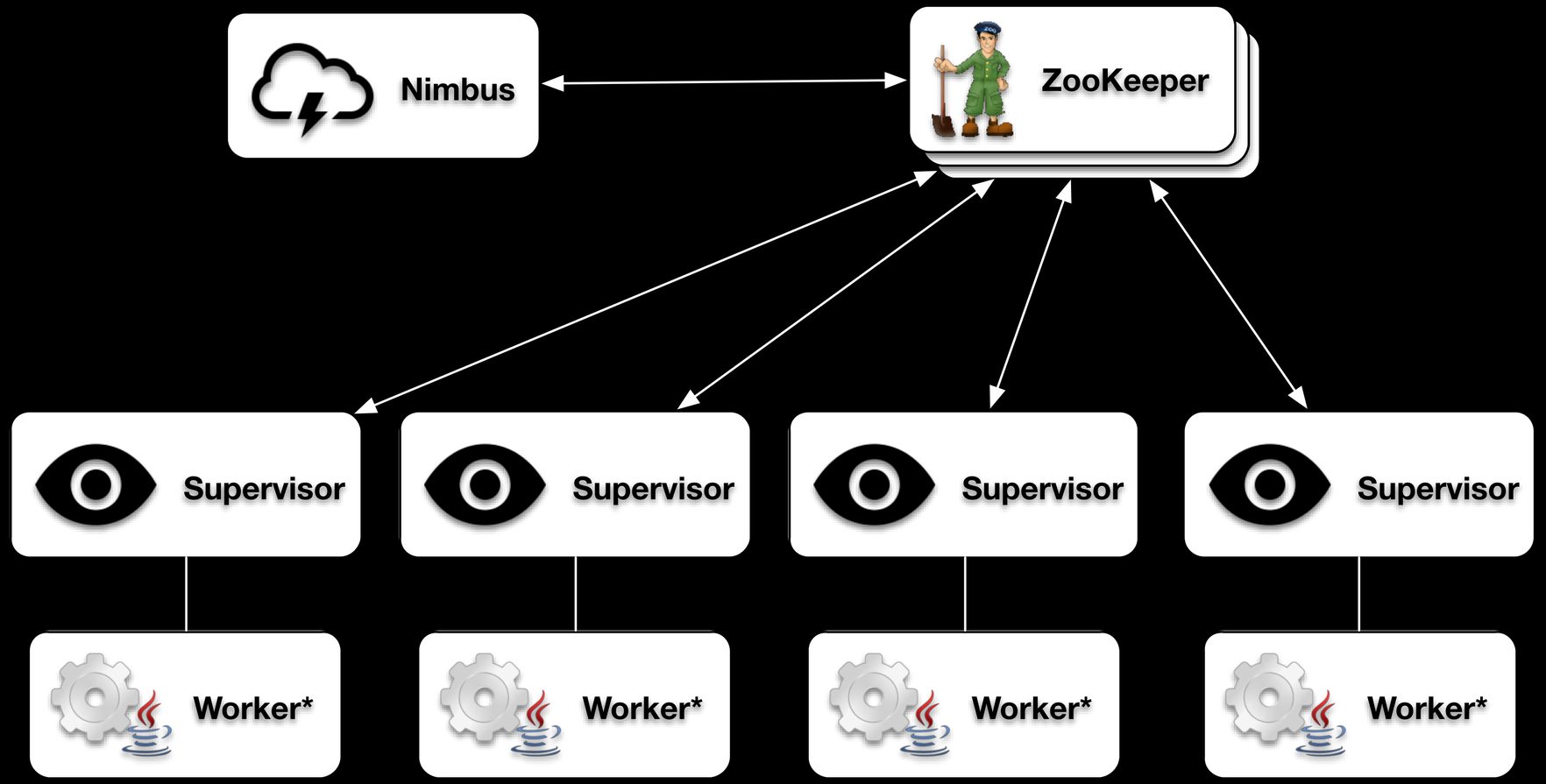
```
storm blobstore create --file dict.txt --acl o::rwa
--repl-fctr 2 key1
```

Making it available to a topology:

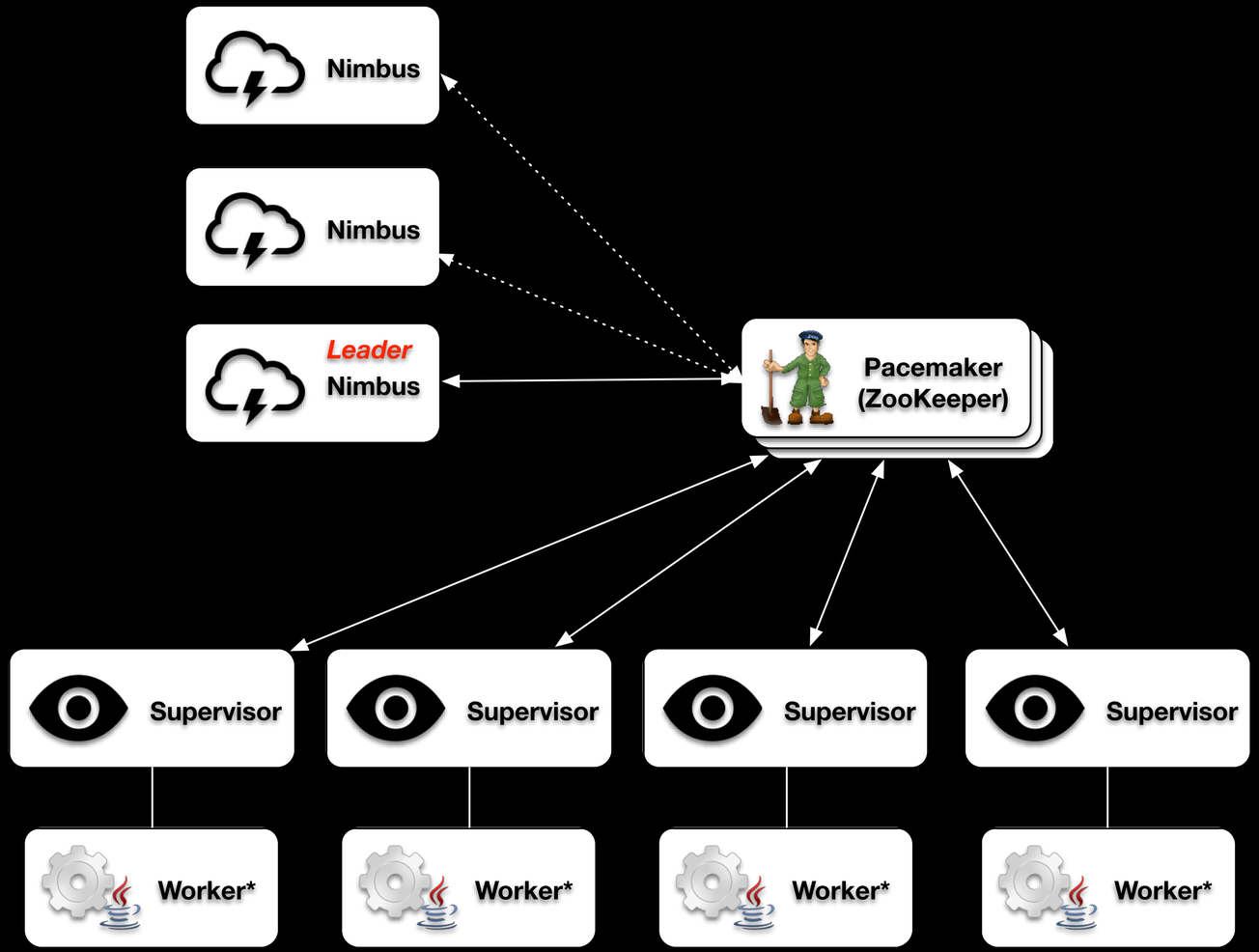
```
storm jar topo.jar my.topo.Class test_topo -c
topology.blobstore.map='{"key1":
{"localname":"dict.txt", "uncompress":"false"}}'
```

High Availability Nimbus

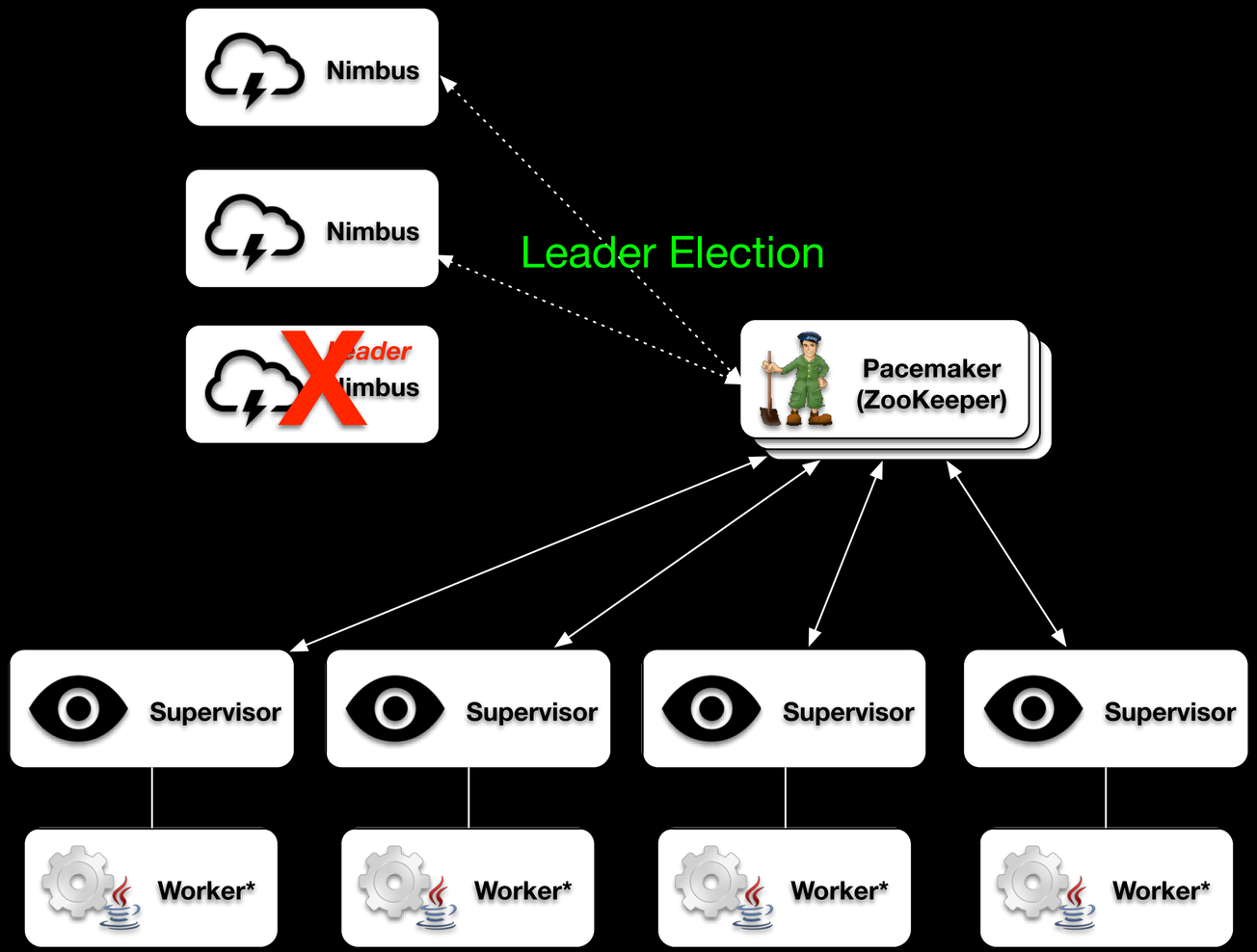
Before HA Nimbus



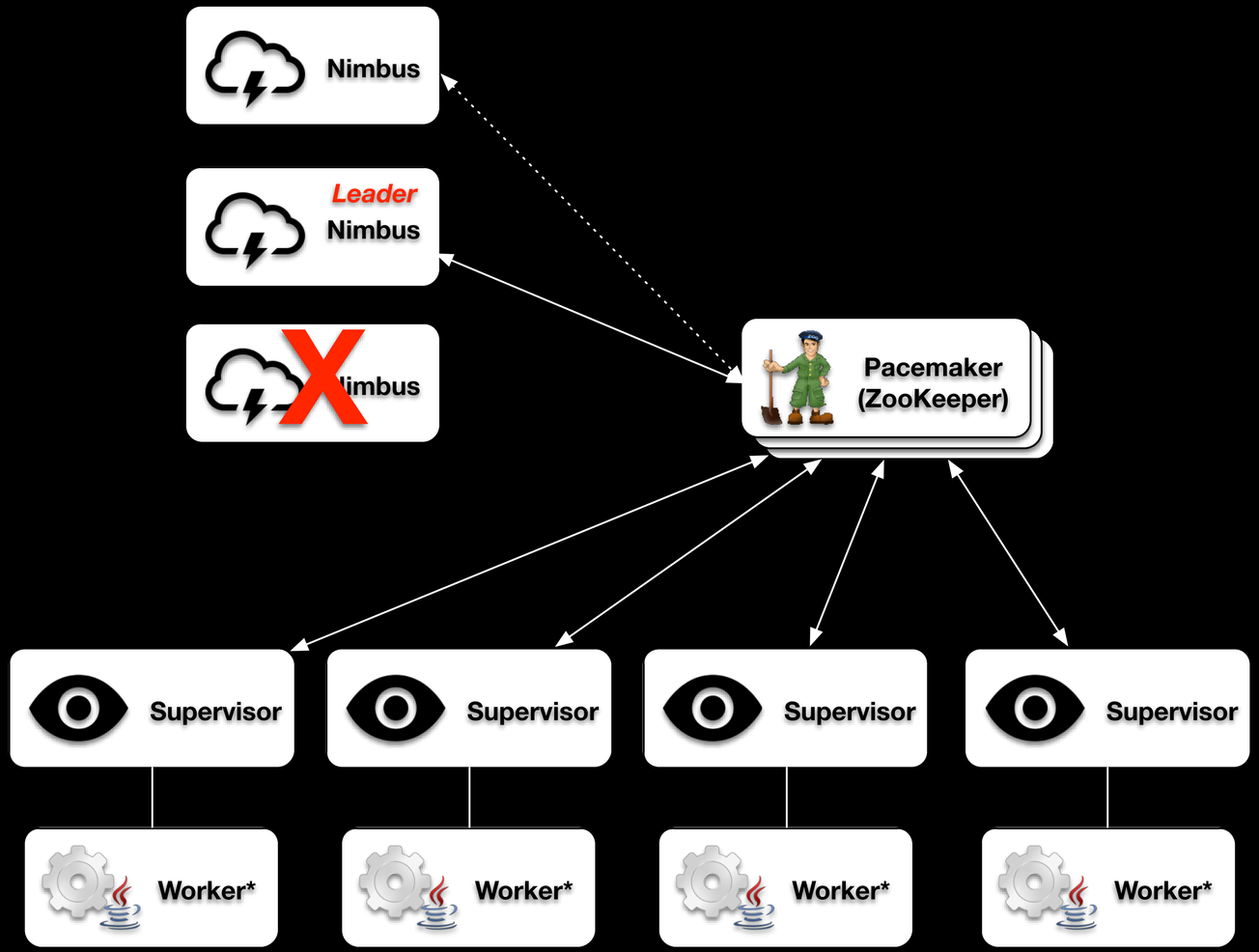
HA Nimbus


Nimbus
Nimbus
Leader
Nimbus
Pacemaker
(ZooKeeper)
Supervisor
Supervisor
Supervisor
Supervisor
Worker*
Worker*
Worker*
Worker*

HA Nimbus - Failover



HA Nimbus - Failover



HA Nimbus

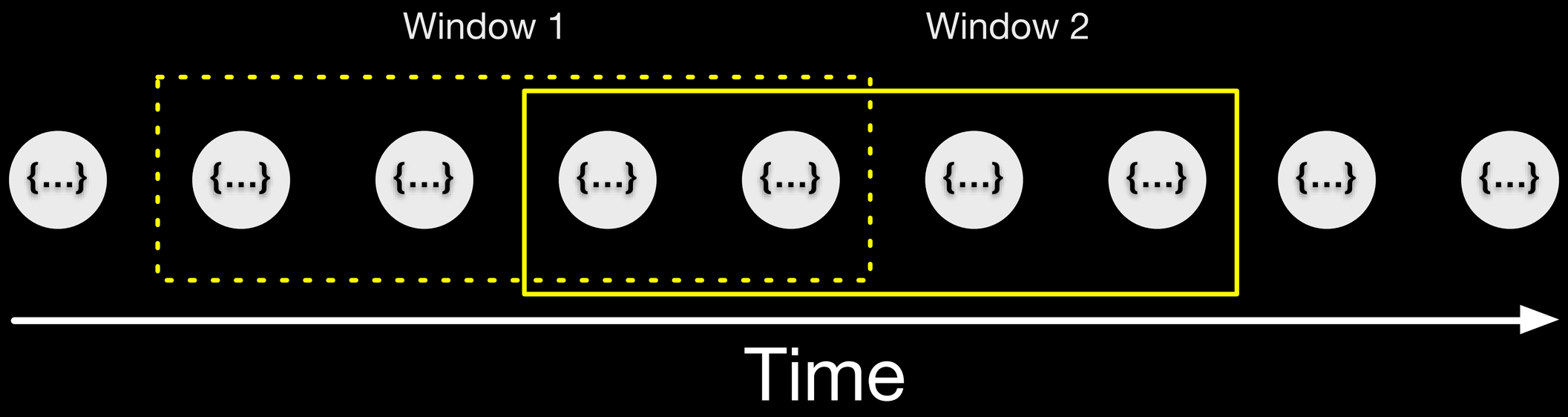
- Increase overall availability of Nimbus
- Nimbus hosts can join/leave at any time
- Leverages Distributed Cache API
- Topology JAR, Config, and Serialized Topology uploaded to Distributed Cache
- Replication guarantees availability of all files

Native Streaming Windows

Streaming Windows

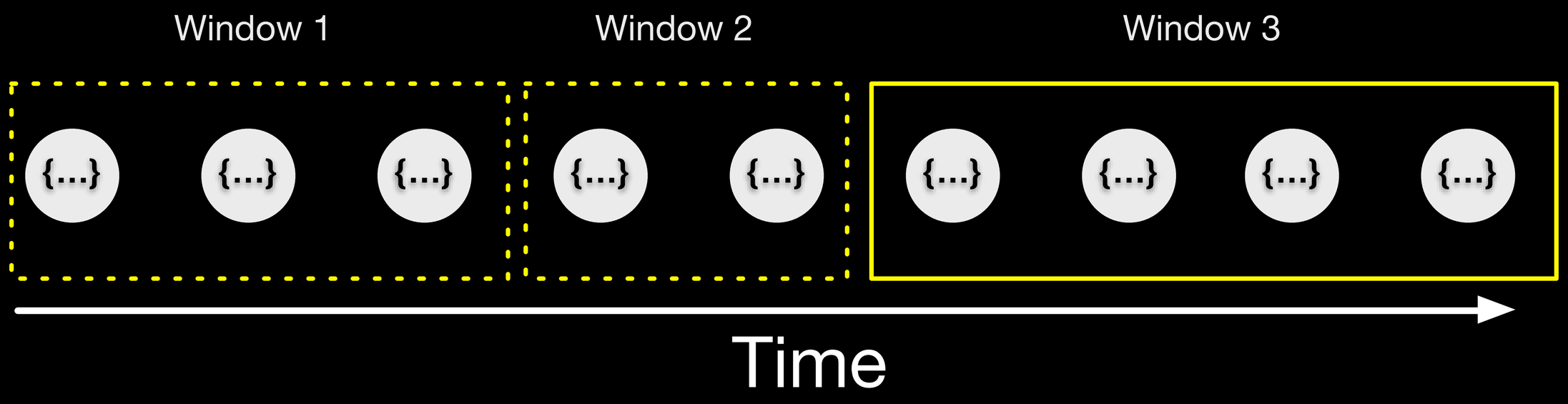
- Specify Length - Duration or Tuple Count
- Slide Interval - How often to advance the window

Sliding Windows



Windows can overlap

Tumbling Windows



Windows do not overlap

Streaming Windows

- Timestamps (Event Time, Ingestion Time and Processing Time)
- Out of Order Tuples
- Watermarks
- Window State Checkpointing

Sate Management

Stateful Bolts with Automatic Checkpointing

State Management

What you *see.*

State Management

```
public class WordCountBolt extends BaseStatefulBolt<KeyValueState> {

    private KeyValueState wordCounts;
    private OutputCollector collector;

    public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
        this.collector = collector;
    }

    public void initState(KeyValueState state) {
        this.wordCounts = state;
    }

    public void execute(Tuple tuple) {
        String word = tuple.getString(0);
        Integer count = (Integer) wordCounts.get(word, 0);
        count++;
        wordCounts.put(word, count);
        collector.emit(new Values(word, count));
    }
}
```

State Management

```
public class WordCountBolt extends BaseStatefulBolt<KeyValueState> {

    private KeyValueState wordCounts;
    private OutputCollector collector;

    public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
        this.collector = collector;
    }

    public void initState(KeyValueState state) {
        this.wordCounts = state;
    }

    public void execute(Tuple tuple) {
        String word = tuple.getString(0);
        Integer count = (Integer) wordCounts.get(word, 0);
        count++;
        wordCounts.put(word, count);
        collector.emit(new Values(word, count));
    }
}
```

Initialize State

State Management

```
public class WordCountBolt extends BaseStatefulBolt<KeyValueState> {

    private KeyValueState wordCounts;
    private OutputCollector collector;

    public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
        this.collector = collector;
    }

    public void initState(KeyValueState state) {
        this.wordCounts = state;
    }

    public void execute(Tuple tuple) {
        String word = tuple.getString(0);
        Integer count = (Integer) wordCounts.get(word, 0);
        count++;
        wordCounts.put(word, count);
        collector.emit(new Values(word, count));
    }
}
```

Read/Update State

Sate Management

Automatic Checkpointing

State Management

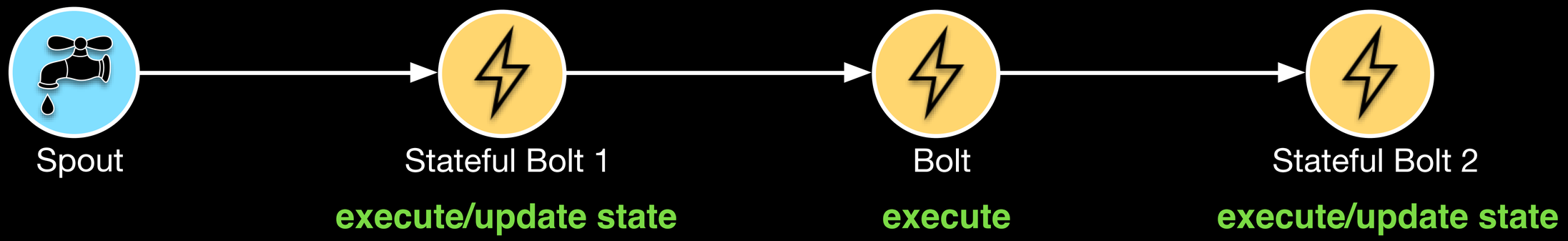
Checkpointing/Snapshotting

- Asynchronous Barrier Snapshotting (ABS) algorithm [1]
- Chandy-Lamport Algorithm [2]

[1] http://arxiv.org/pdf/1506.08603v1.pdf
[2] http://research.microsoft.com/en-us/um/people/lamport/pubs/chandy.pdf

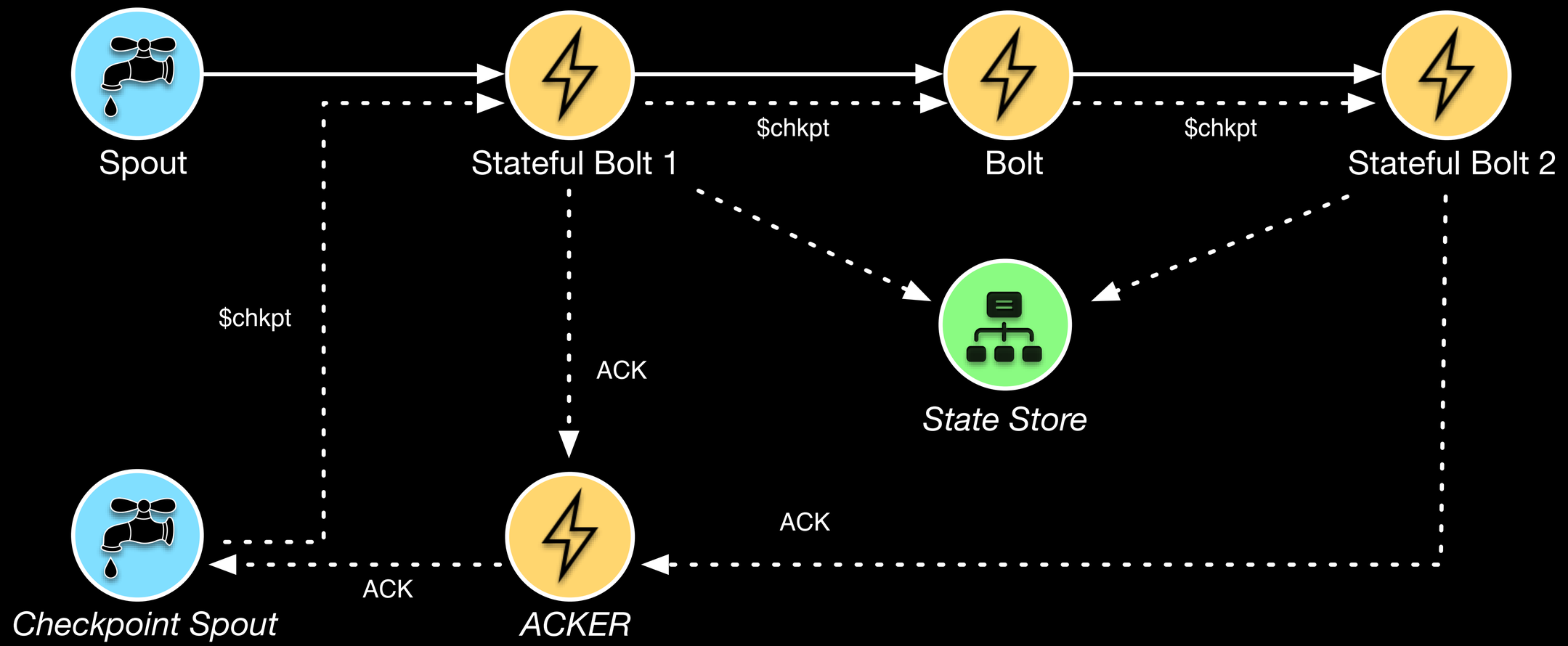
Storm State Management

Checkpointing/Snapshotting: What you *see.*



Storm State Management

Checkpointing/Snapshotting: What you *get.*



Automatic Back Pressure

Automatic Back Pressure

- In previous Storm versions, the only way to throttle topologies was to enable ACKing and set `topology.spout.max.pending`.
- If you don't require at-least-once guarantees, this imposed a significant performance penalty.**

** In Storm 1.0 this penalty is drastically reduced (more on this later)

Automatic Backpressure

- High/Low Watermarks (expressed as % of buffer size)
- Back Pressure thread monitors buffers
- If High Watermark reached, slow down Spouts
- If Low Watermark reached, stop throttling
- All Spouts Supported

Resource Aware Scheduler (RAS)

Resource Aware Scheduler

- Specify the resource requirements (Memory/CPU) for individual topology components (Spouts/Bolts)
- Memory: On-Heap / Off-Heap (if off-heap is used)
- CPU: Point system based on number of cores
- Resource requirements are ***per component instance*** (parallelism matters)

Resource Aware Scheduler

- CPU and Memory availability described in `storm.yaml` on each supervisor node. E.g.:

```
supervisor.memory.capacity.mb: 3072.0
supervisor.cpu.capacity: 400.0
```

- Convention for CPU capacity is to use 100 for each CPU core

Resource Aware Scheduler

Setting component resource requirements:

```
SpoutDeclarer spout =  builder.setSpout("sp1", new TestSpout(), 10);
//set cpu requirement
spout.setCPULoad(20);
//set onheap and offheap memory requirement
spout.setMemoryLoad(64, 16);

BoltDeclarer bolt1 = builder.setBolt("b1", new MyBolt(), 3).shuffleGrouping("sp1");
//sets cpu requirement.  Not neccessary to set both CPU and memory.
//For requirements not set, a default value will be used
bolt1.setCPULoad(15);

BoltDeclarer bolt2 = builder.setBolt("b2", new MyBolt(), 2).shuffleGrouping("b1");
bolt2.setMemoryLoad(100);
```

Storm Usability Improvements

Enhanced Debugging and Monitoring of Topologies

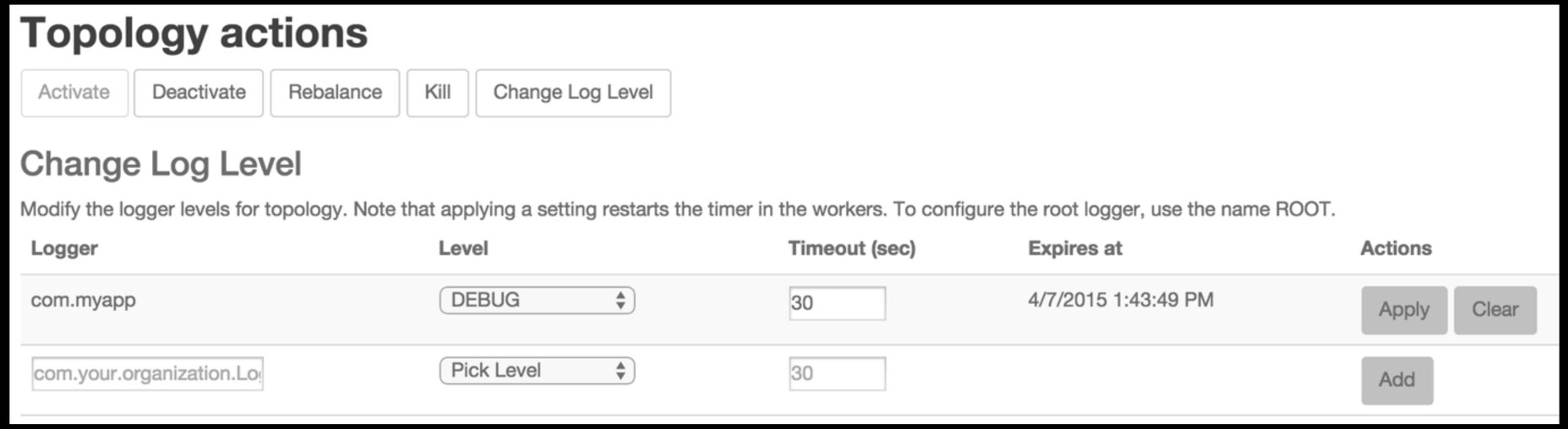
Dynamic Log Level Settings

Dynamic Log Levels

- Set log level setting for a running topology
- Via Storm UI and CLI
- Optional timeout after which changes will be reverted
- Logs searchable from Storm UI/Logviewer

Dynamic Log Levels

Via Storm UI:



Dynamic Log Levels

Via Storm CLI:

```
./bin/storm set_log_level [topology name] -l
[logger_name]=[LEVEL]:[TIMEOUT]
```

Tuple Sampling

- No more debug bolts or Trident functions!
- In Storm UI: Select a Topology or component and click “Debug”
- Specify a sampling percentage (% of tuples to be sampled)
- Click on the “Events” link to view the sample log.

Distributed Log Search

- Search across all log files for a specific topology
- Search in archived (ZIP) logs
- Results include matches from all Supervisor nodes

Dynamic Worker Profiling

- Request worker profile data from Storm UI:
  - Heap Dumps
  - JStack Output
  - JProfile Recordings
- Download generated files for off-line analysis
- Restart workers from UI

Supervisor Health Checks

- Identify Supervisor nodes that are in a bad state
- Automatically decommission bad nodes
- Simple shell script
- You define what constitutes “Unhealthy”

New Integrations

- Cassandra
- Solr
- Elastic Search
- MQTT

Integration Improvements

- Kafka
- HDFS Spout
- Avro Integration for HDFS
- HBase
- Hive

Before I forget...

Performance

Up to 16x faster throughput.

Realistically 3x -- Highly dependent on use case

> 60% Latency Reduction

Bear in mind performance varies widely depending on the use case.

The most important benchmarks are the ones ***you do***.

Storm 2.0

Underway @ Apache

Clojure to Java

Broadening the contributor base

Clojure to Java

Alibaba JStorm Contribution

Questions?

Thank you!

P. Taylor Goetz, Hortonworks
@ptgoetz