Tigon

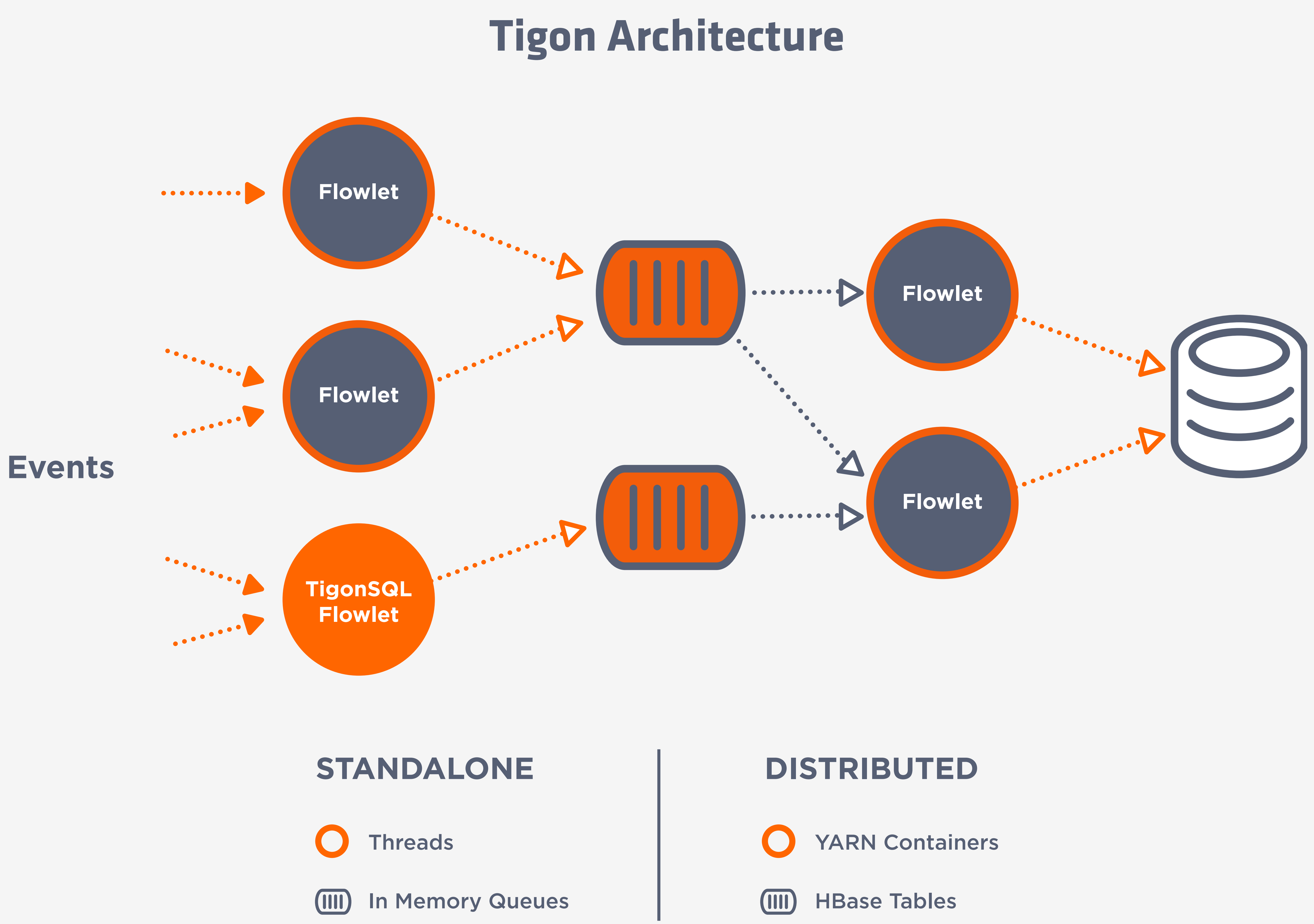
High Throughput Transactional Stream Processing

Terence Yim (@chtyim)

Who We Are

- Create open source software than provides simple access to powerful technologies
- Cask Data Application Platform (http://cdap.io) CDAP
  - A platform runs on top of Hadoop to provide data and application virtualization
    - Virtualization of data through logical representations of underlying data
    - Virtualization of applications through application containers
    - Services and tools that enable faster application development and better operational control in production
- Coopr (http://coopr.io) COOPR
  - Clusters with a Click
  - Self-service, template-based cluster provisioning system

Tigon Architecture



- Basic unit of execution is called ***Flow***
  - A Directed Acyclic Graph (DAG) of ***Flowlets***
- Multiple modes of execution
  - Standalone
    - Useful for testing
  - Distributed
    - Fault tolerant, scalable

Execution Model

- **Distributed mode**
  - Runs on YARN through Apache Twill
  - One YARN container per one flowlet instance
  - One active thread per flowlet instance
  - Flowlet instances can scale dynamically and independently
    - No need to stop Flow
- **Standalone mode**
  - Single JVM
  - One thread per flowlet instance
  - Queues are in-memory, not really persisted

Flowlet

- Processing unit of a Flow
- Flowlets within a Flow are connected through ***Distributed Queue***
- Consists of one or more ***Process Method***(s)
  - User defined Java method
  - No restriction on what can be done in the method
- A ***Process Method*** in a Flowlet can be triggered by
  - Dequeue objects emitted by upstream flowlet(s)
  - Repeatedly triggered by time delay
    - Useful for polling external data (Twitter Firehose, Kafka, …)
- Inside Process Method, you can emit objects for downstream Flowlet(s)

Word Count

```
public class WordSplitter extends AbstractFlowlet {
  private OutputEmitter<String> output;

  @Tick(delay=100, unit=TimeUnit.MILLISECONDS)
  public void poll() {
    // Poll tweets from Twitter Firehose
    // ...
    for (String word : tweet.split("\\s+")) {
      output.emit(word);
    }
  }
}
```

Word Count

```
public class WordCounter extends AbstractFlowlet {

  @ProcessInput
  public void process(String word) {
    // Increments count for the word in HBase
  }
}
```

Word Count

```
public class WordCountFlow implements Flow {
  @Override
  public FlowSpecification configure() {
    return FlowSpecification.Builder.with()
      .setName("WordCountFlow")
      .setDescription("Flow for counting words)
      .withFlowlets()
        .add("splitter", new WordSplitter())
        .add("counter", new WordCounter())
      .connect()
        .from("splitter").to("counter")
      .build();
  }
}
```

Data Consistency

- Node dies
- Process method throws Exception
  - Transient IO issues (e.g. connection timeout)
- Conflicting updates
  - Writes to the same cell from two instances

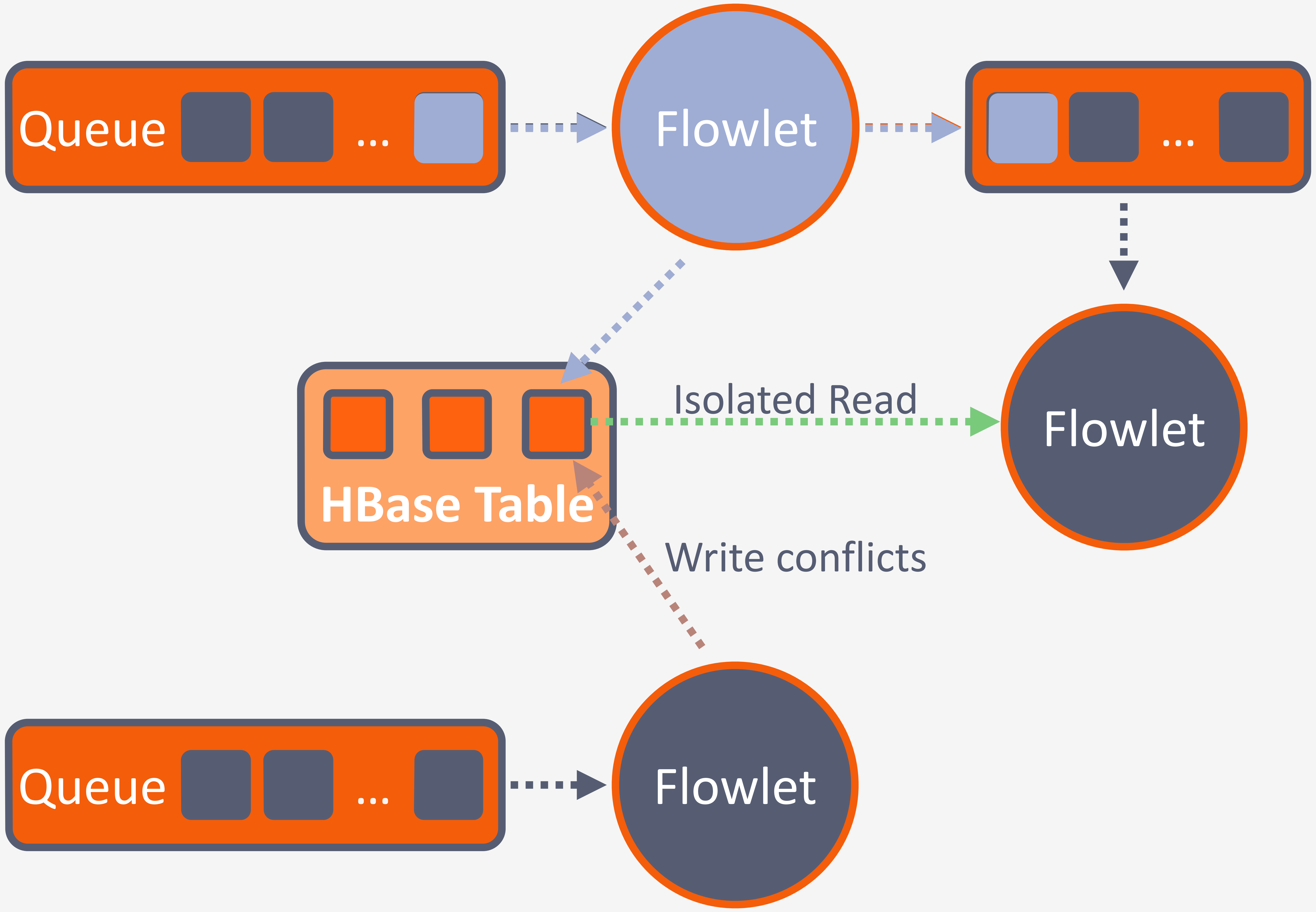
Data Consistency

- Resume from failure by replaying queue
  - At least once
    - Data logic be idempotent
    - Program handles rollback / skipping
  - At most once
    - Lossy computation
  - Exactly once
    - Ideal model for data consistency as if failure doesn’t occurred

- How about data already been written to backing store?

Flowlet Transaction

Transaction to the rescue



Tigon and HBase

- Tigon uses HBase heavily
  - Queues are implemented on HBase Tables
  - Optionally integrated with HBase as user data stores

- HBase has limited support on transaction
  - Has atomic cell operations
  - Has atomic batch operations on rows within the same region
  - NO cross region atomic operations
  - NO cross table atomic operations
  - NO multi-RPC atomic operations

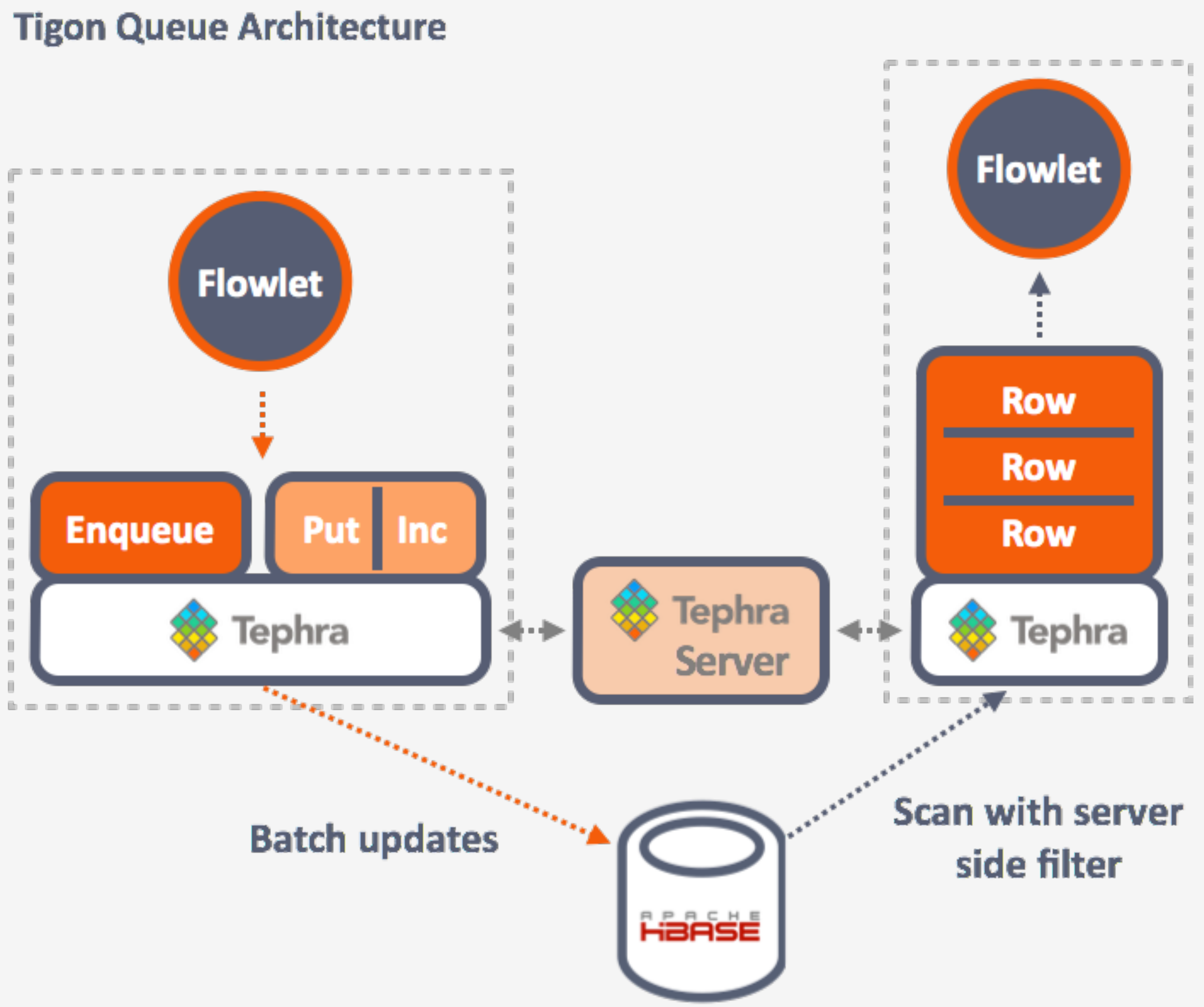
Tephra on HBase

- Tephra (http://tephra.io) Tephra
  - Brings ACID to HBase
    - Extends to multi-rows, multi-regions, multi-tables
  - Multi-Version Concurrency Control
    - Cell version = Transaction ID
      - All writes in the same transaction use the same transaction ID as version
      - Reads isolation by excluding uncommitted transactions
  - Optimistic Concurrency Control
    - Conflict detection at commit time
      - No locking, hence no deadlock
      - Performs good if conflicts happens rarely

Flowlet Transaction

- Transaction starts before dequeue
- Following actions happen in the same transaction
    - Dequeue
    - Invoke Process Method
    - States updates
        - Only if updates are integrated with Tephra (e.g. ***Queue*** and ***TransactionAwareHTable*** in Tephra)
    - Enqueue
- Transaction failure will trigger rollback
    - Exception thrown from Process Method
    - Write conflicts

Distributed Transactional Queue



- Persisted transactionally on HBase
- One row per queue entry
- Enqueue
  - Batch updates at commit time
  - Commits together with user updates
- Dequeue
  - Scans for uncommitted entries
  - Marks entries as processed on commit
- Coprocessor
  - Skipping committed entries on dequeue scan
  - Cleanup consumed entries on flush/compact

Transaction Failure

- Rollback cost may be high, depends on what triggers the failure
  - User exception
    - Most likely low as most changes are still in local buffer
  - Write conflicts
    - Relatively low if conflicts are detected before persisting
    - High if changes are persisted and conflicts found during the commit phase
- Flowlet optionally implements the ***Callback*** interface to intercept transaction failure
  - Decide either retry or abort the failed transaction
- Default is to retry with limited number of times (Optimistic Concurrency Control)
  - Max retries is setting through the ***@ProcessInput*** annotation.

Performance Tips

- Runs more Flowlet instances
- Dequeue Strategy
  - @HashPartition
    - Hash on the write key to avoid write conflicts
- Batch dequeue
  - Use ***@Batch*** annotation on Process Method
  - More entries will be processed in one transaction
    - Minimize IO and transaction overhead

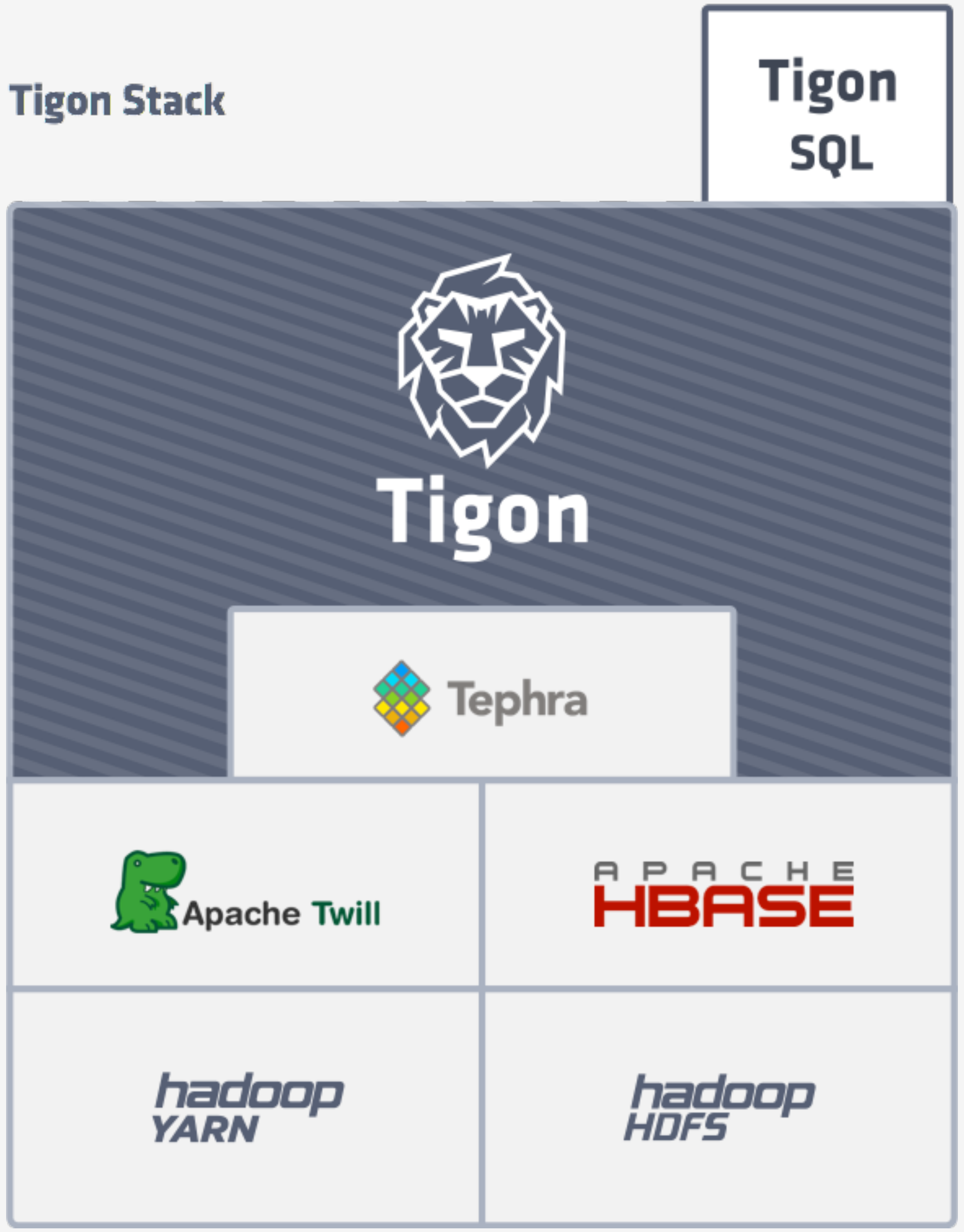
Word Count

```
public class WordSplitter extends AbstractFlowlet {
 private OutputEmitter<String> output;

 @Tick(delay=100, unit=TimeUnit.MILLISECONDS)
 public void poll() {
  // Poll tweets from Twitter Firehose
  // ...
  for (String word : tweet.split("\\s+")) {
   output.emit(word, "key", word);   // Hash by the word
  }
 }
}
```

Word Count

```
public class WordCounter extends AbstractFlowlet {

  @ProcessInput
  @Batch(100)
  @HashPartition("key")
  public void process(String word) {
    // ...
  }
}
```

Summary



- Real-time stream processing framework
- Exactly once processing guarantees
  - Transaction message queue on ***Apache HBase***
- Transactional storage integration
  - Through **Tephra** transaction engine
- Executes on ***Hadoop YARN***
  - Through ***Apache Twill***
- Simple Java Programmatic API
  - Imperative programming
  - Data model through Java Object

Road map

- Partitioned queue
  - Better scalability, better performance
  - Preliminary tests shows 100K events/sec on 8 nodes cluster with 10 flowlet instances
    - Linearly scalable
- Drain / cleanup queue
  - Better controls for upgrade
- Supports more programming languages
- External logging and metrics system integration
- More source Flowlet types
  - Kafka, Twitter, Flume…

Contributions

- Web-site: http://tigon.io
- Tigon in CDAP: http://cdap.io
- Source: https://www.github.com/caskdata/tigon
- Mailing lists
  - tigon-dev@googlegroups.com
  - tigon-user@googlegroups.com
- JIRA
  - http://issues.cask.co/browse/TIGON