Online data processing with S4 and Omid*

Flavio Junqueira

Microsoft Research, Cambridge

* Work done while in Yahoo! Research

Big Data defined

Wikipedia

In information technology, big data[1][2] is a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.

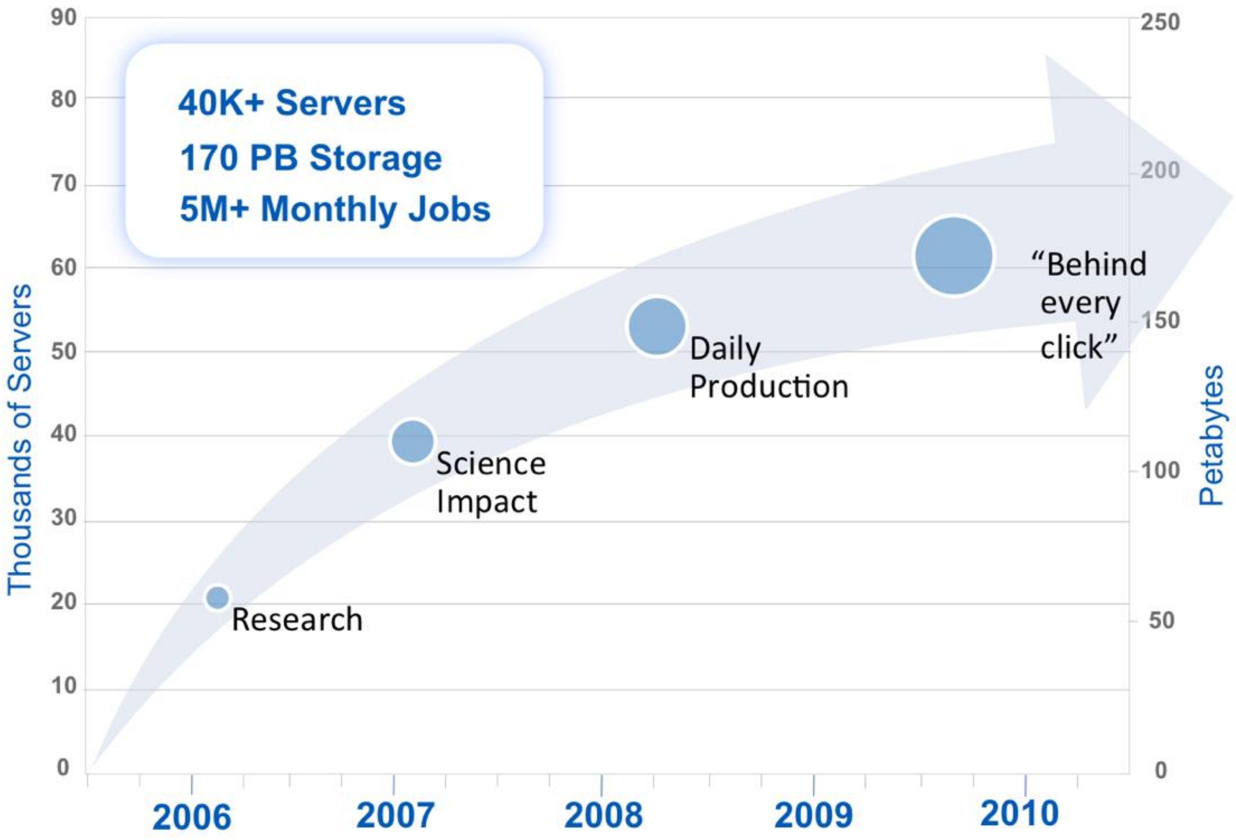
Hortonworks

A Big Data system has four properties:

- *It uses **local storage** to be fast but inexpensive*
- *It uses clusters of **commodity hardware** to be inexpensive*
- *It uses **free software** to be inexpensive*
- *It is **open source** to avoid expensive **vendor lock-in***

http://hortonworks.com/blog/big-data-defined/
http://hortonworks.com/blog/big-data-defined-part-deux-value-definition/

Hadoop @ Yahoo!



Eric Baldeschwieler @IBM Big Data, May 2011

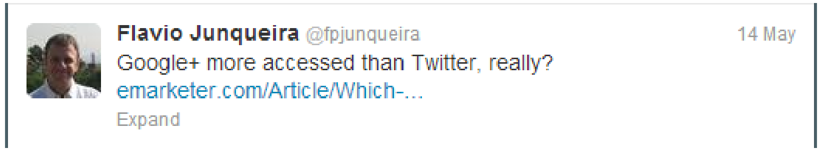
Context: Back in 2008

- Needed scalable real-time processing
  - Direct feedback
  - Optimization
  - Adaptation
- Use case
  - Ad ranking with clickthrough analysis
- Solution
  - Distributed stream processing platform
  - At the time
    - No generic platform available
    - Research project

Source: unbounce.com

Stream Processing Platform

- Enables applications that process streams of events



Flavio Junqueira @fpjunqueira 14 May
Google+ more accessed than Twitter, really?
emarketer.com/Article/Which-...
Expand

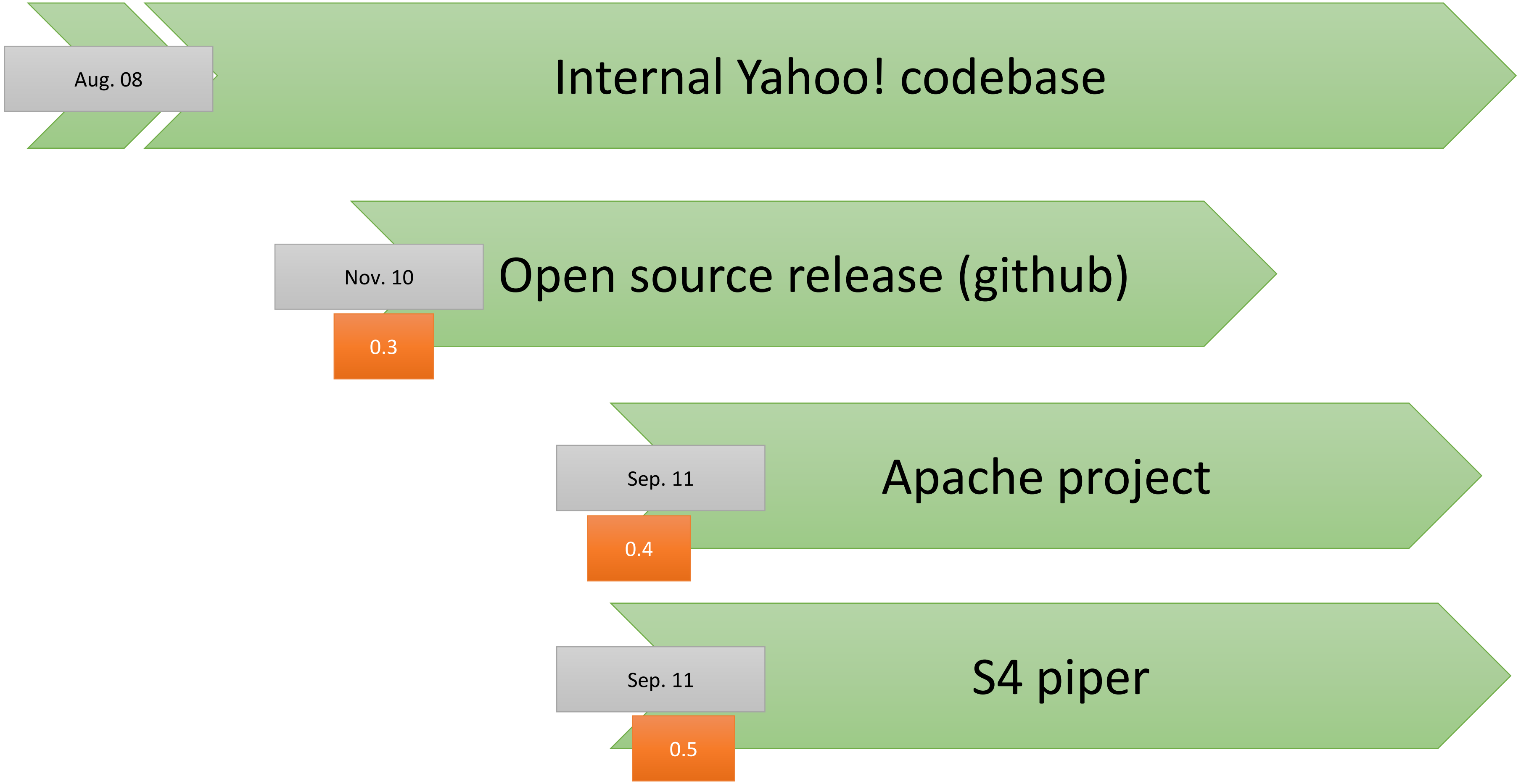
Source: ji-make.com

- Desirable properties
  - Online meaning low-latency
  - Best effort
  - Scalable
  - Fault tolerance (perhaps limited)
  - Flexible

S4: Simple Scalable Streaming System

http://incubator.apache.org/s4

S4 Evolution



Aug. 08

Internal Yahoo! codebase

Nov. 10

0.3

Open source release (github)

Sep. 11

0.4

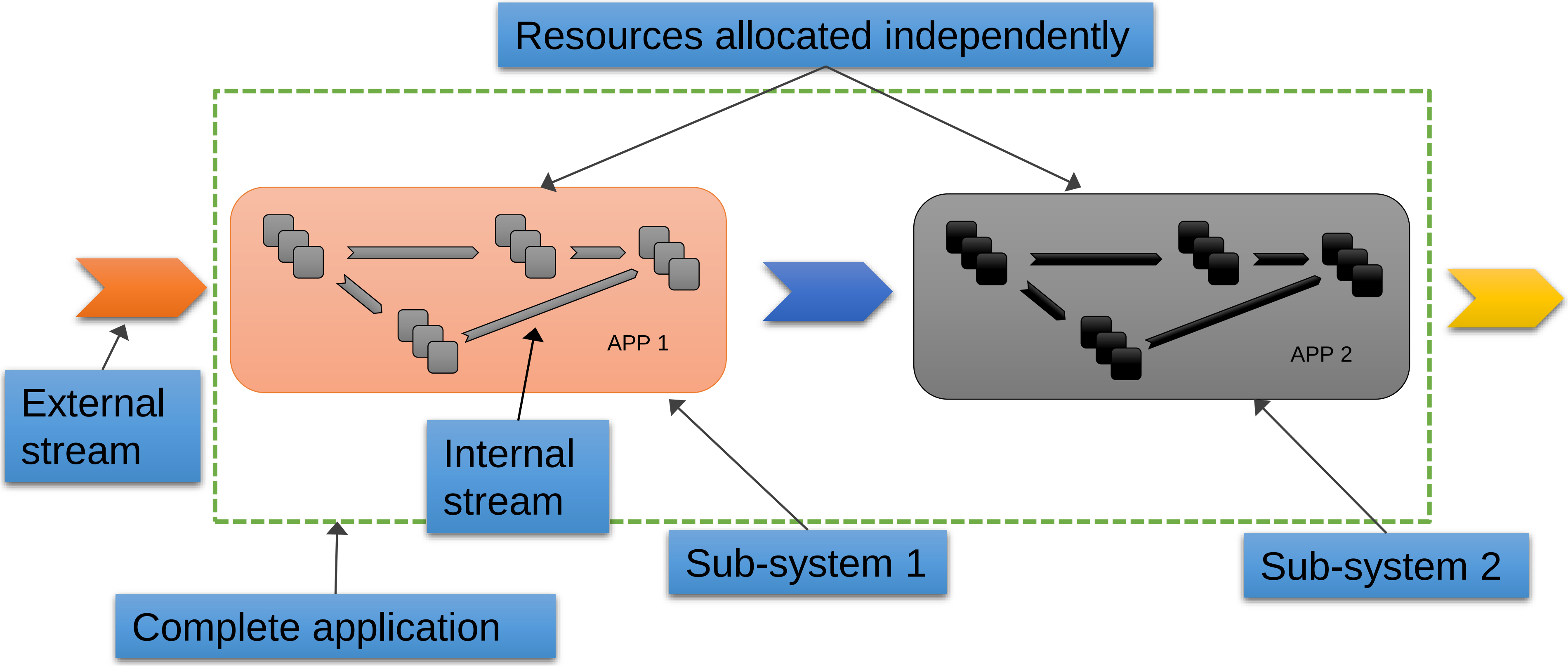
Apache project

Sep. 11

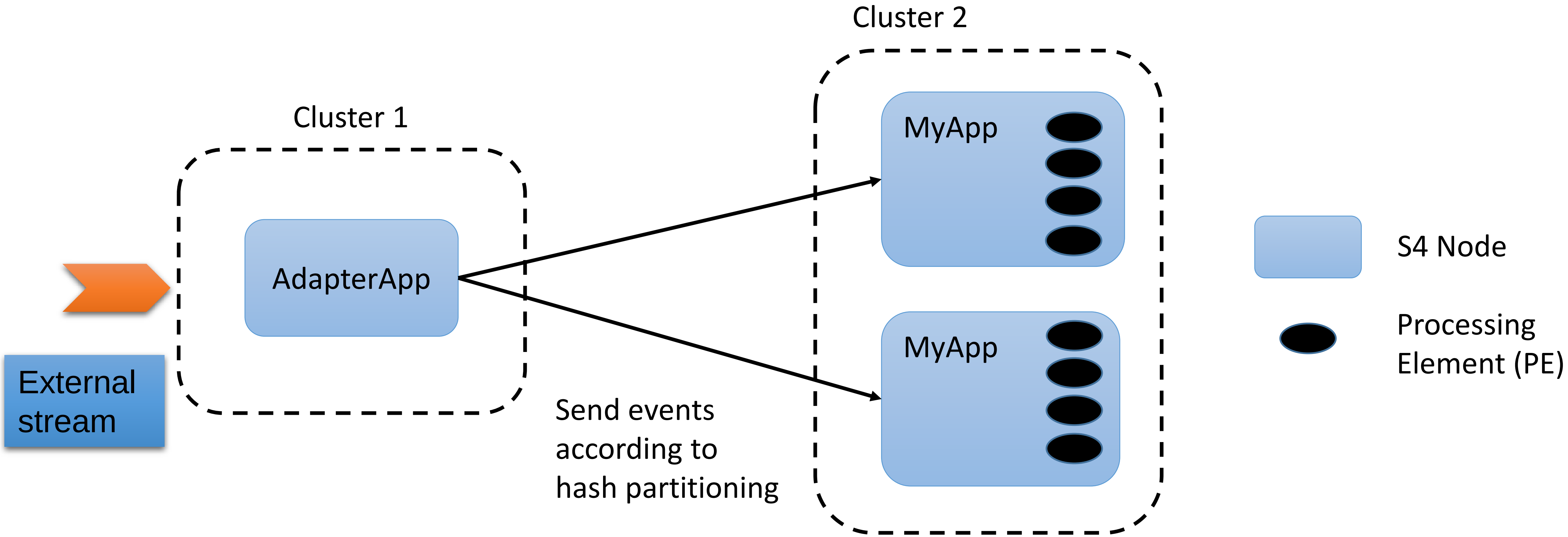
0.5

S4 piper

System overview



App



Cluster 1

Cluster 2

External stream

AdapterApp

MyApp

MyApp

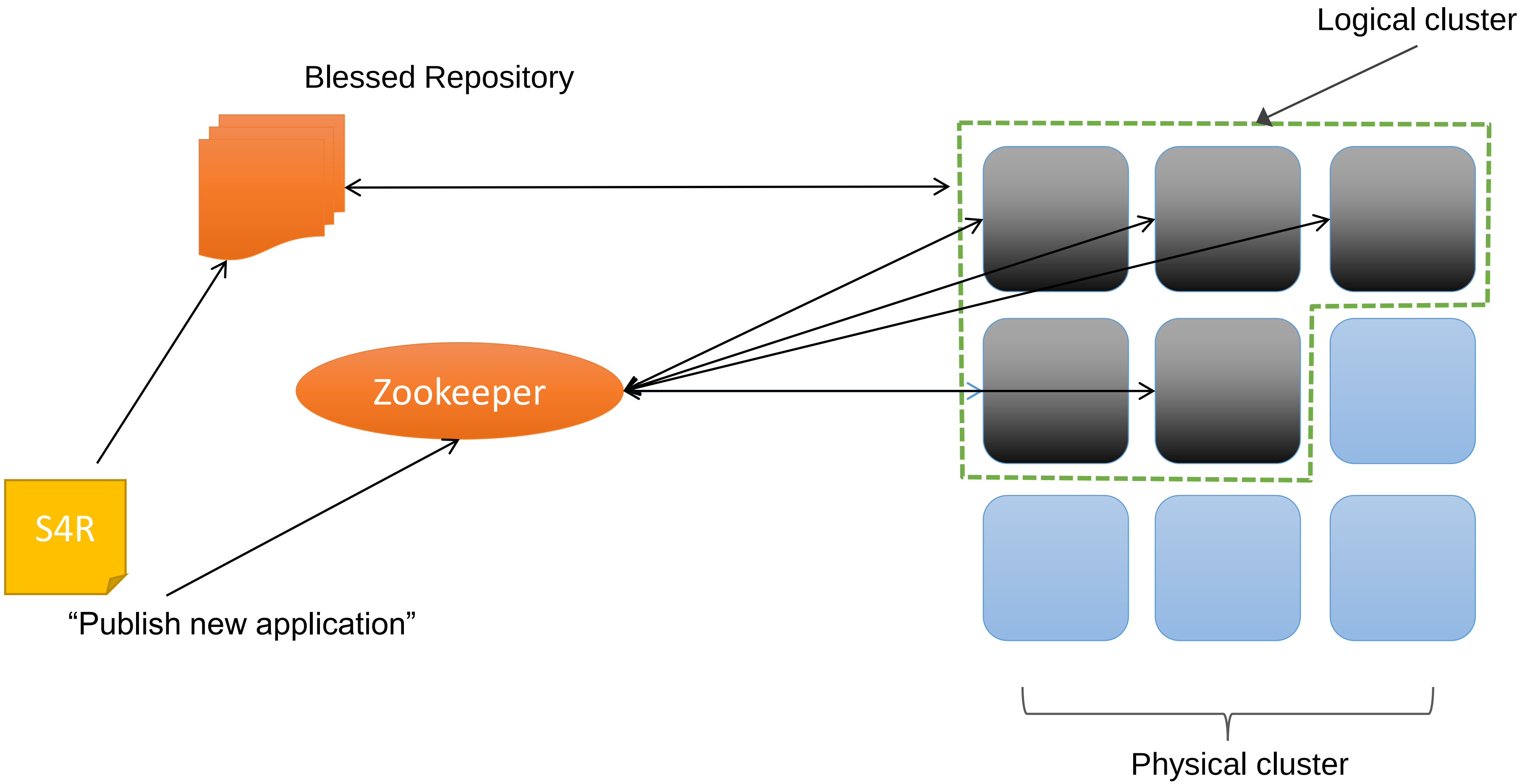
Send events according to hash partitioning

S4 Node

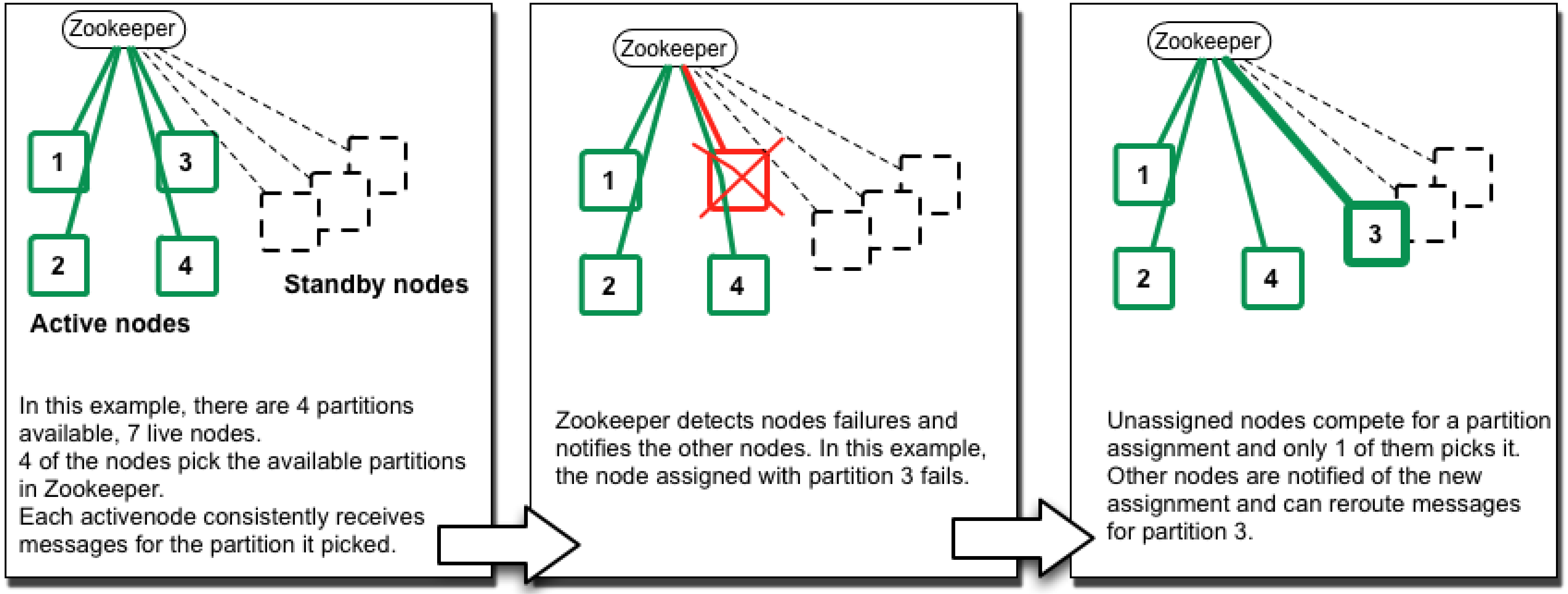
Processing Element (PE)

- App defines keys
- One PE per key

Deployment


Logical cluster
Blessed Repository
Zookeeper
S4R
"Publish new application"
Physical cluster

Fault tolerance: Fail-over



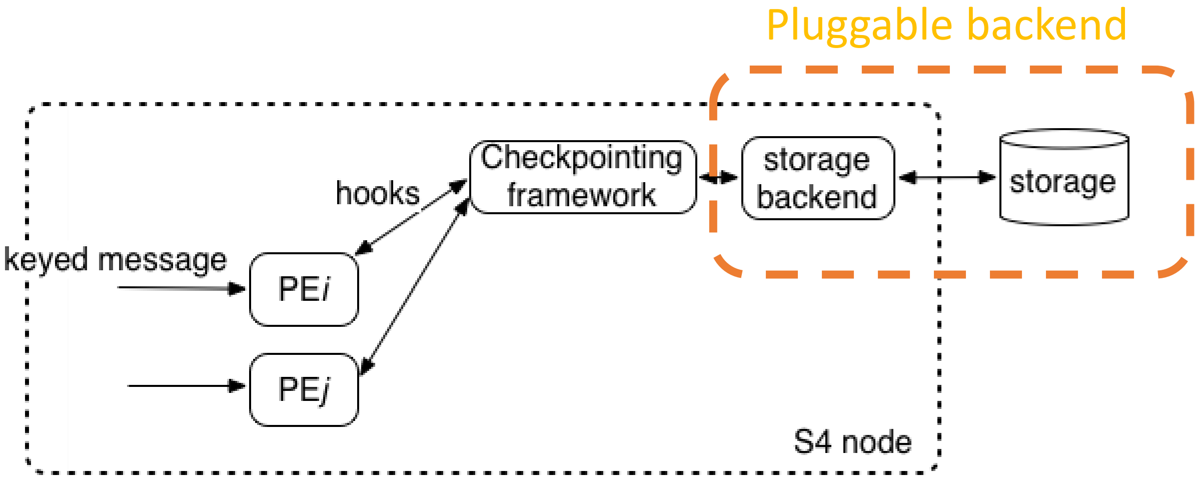
In this example, there are 4 partitions available, 7 live nodes.
4 of the nodes pick the available partitions in Zookeeper.
Each activenode consistently receives messages for the partition it picked.

Zookeeper detects nodes failures and notifies the other nodes. In this example, the node assigned with partition 3 fails.

Unassigned nodes compete for a partition assignment and only 1 of them picks it. Other nodes are notified of the new assignment and can reroute messages for partition 3.

http://incubator.apache.org/s4/doc/0.6.0/fault_tolerance/

Fault tolerance: Checkpointing



http://incubator.apache.org/s4/doc/0.6.0/fault_tolerance/

- Uncoordinated and Asynchronous checkpoints
- Lazy recovery
  - PE state recovered upon message

- Scheme is lossy
- Prevents loss of state accumulated over extended periods

Writing an app

Skeleton of an app

- HelloInputAdapter
  - Events from external source
- HelloApp
  - Creates topology
  - Connects adapter to first PE
- HelloPE
  - Process events

Skeleton of an app

- HelloInputAdapter
  - Events from external source
- HelloApp
  - Creates topology
  - Connects adapter to first PE
- HelloPE
  - Process events

```
public class HelloInputAdapter extends AdapterApp {
 …
   @Override
   protected void onStart() {

     …
          Event event = new Event();
          event.put("name", String.class, line);
          getRemoteStream().put(event);
          connectedSocket.close();

     …
   }

…
}
```

Skeleton of an app

- HelloInputAdapter
  - Events from external source
- HelloApp
  - Creates topology
  - Connects adapter to first PE
- HelloPE
  - Process events

```
public class HelloApp extends App {
    …
    @Override
    protected void onInit() {
        // create a prototype
        HelloPE helloPE = createPE(HelloPE.class);
        // Create a stream that listens to the "names" stream and
        // passes events to the helloPE instance.
        createInputStream("names", new KeyFinder<Event>() {
            @Override
            public List<String> get(Event event) {
                return Arrays.asList(new String[] { event.get("name") });
            }
        }, helloPE);
    }
  …
}
```

Skeleton of an app

- HelloInputAdapter
  - Events from external source
- HelloApp
  - Creates topology
  - Connects adapter to first PE
- HelloPE
  - Process events

```
public class HelloPE extends ProcessingElement {
…
    public void onEvent(Event event) {
        System.out.println("Hello " + (seen ? "again " : "") +
event.get("name") + "!");
     }
…
}
```

S4 Piper

Lessons learned

Lessons from initial design

- State loss upon node crash
- Rigid communication layer
  - UDP only
  - No retransmission, no flow control
- Hard to use/debug/deploy
  - Subjective, but that’s the overall feeling
- Isolated applications
- No regression tests

S4 Piper improvements

- Dynamic coupling of applications
  - Via a simple registration scheme
- Communication via TCP
  - Throttling
  - Retransmission and flow control
- Fault tolerance
  - Checkpointing
  - Node failover

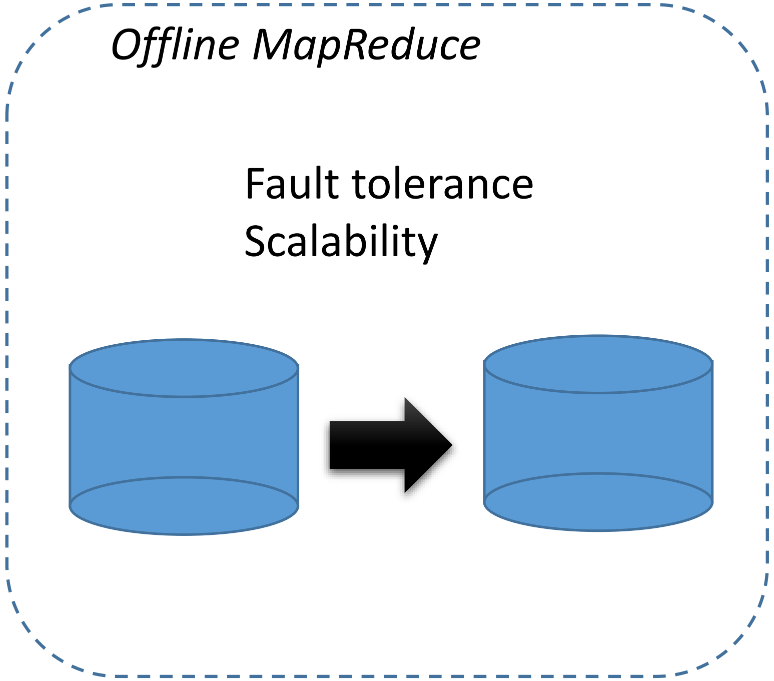
Other ways of achieving low latency?

Omid project: https://github.com/yahoo/omid

Context

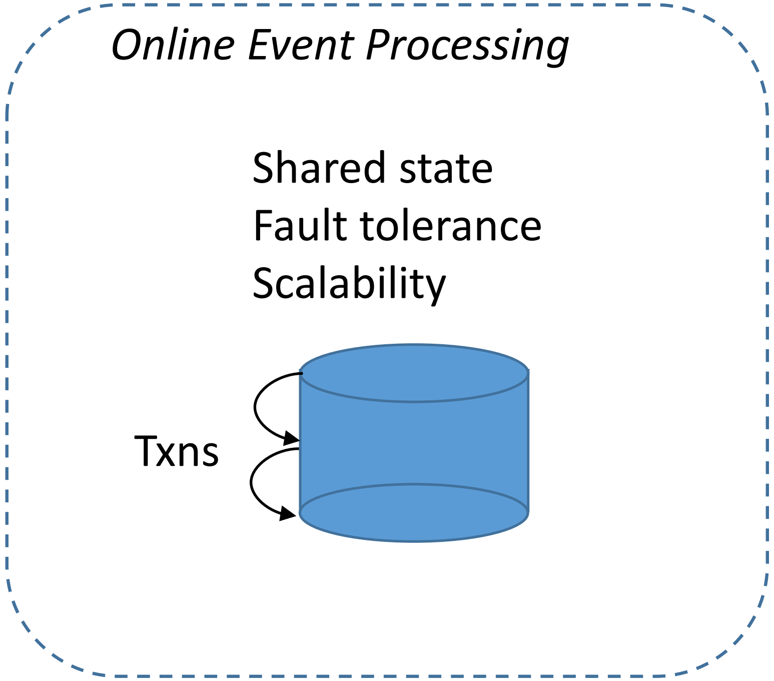
- Incremental processing a la Percolator (Google, OSDI 2010)
    - Distributed transactions
    - Observers
    - Bigtable
- Use case
    - Search index
    - Online updates
    - Crawl to index in 5s
- Omid is about transactions…

Why transactions?



Offline MapReduce

Fault tolerance
Scalability



Online Event Processing

Shared state
Fault tolerance
Scalability

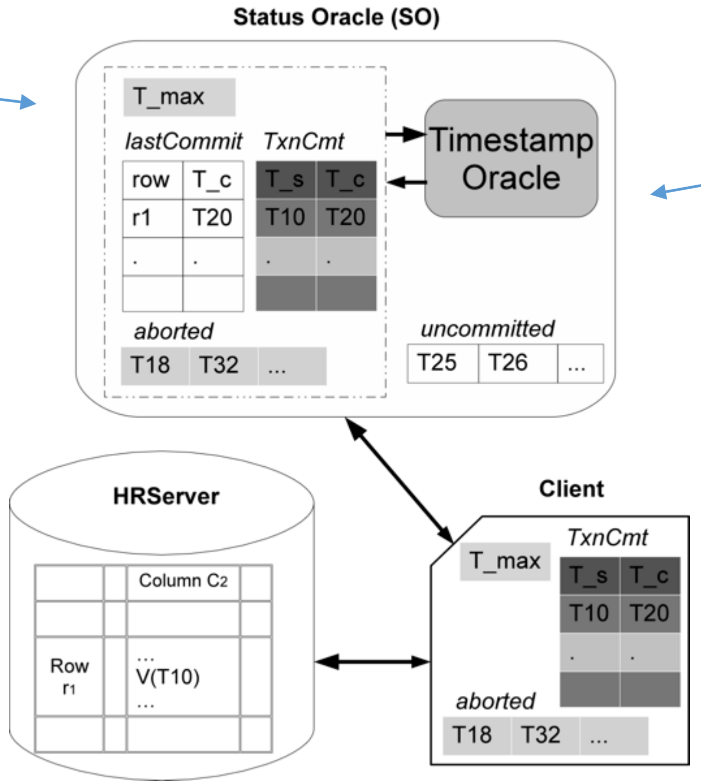
Txns

How does it differ from S4 stream processing?

- S4
  - Data lives in memory
  - Access to databases is expensive
- Incremental processing
  - Computation is close to the data
  - Higher latency
- Omid
  - Targets lower latency for transactions

Omid architecture

WAL for recoverability



row	T_c
r1	T20
.	.

T_s	T_c
T10	T20
.	.

T18	T32	...

T25	T26	...

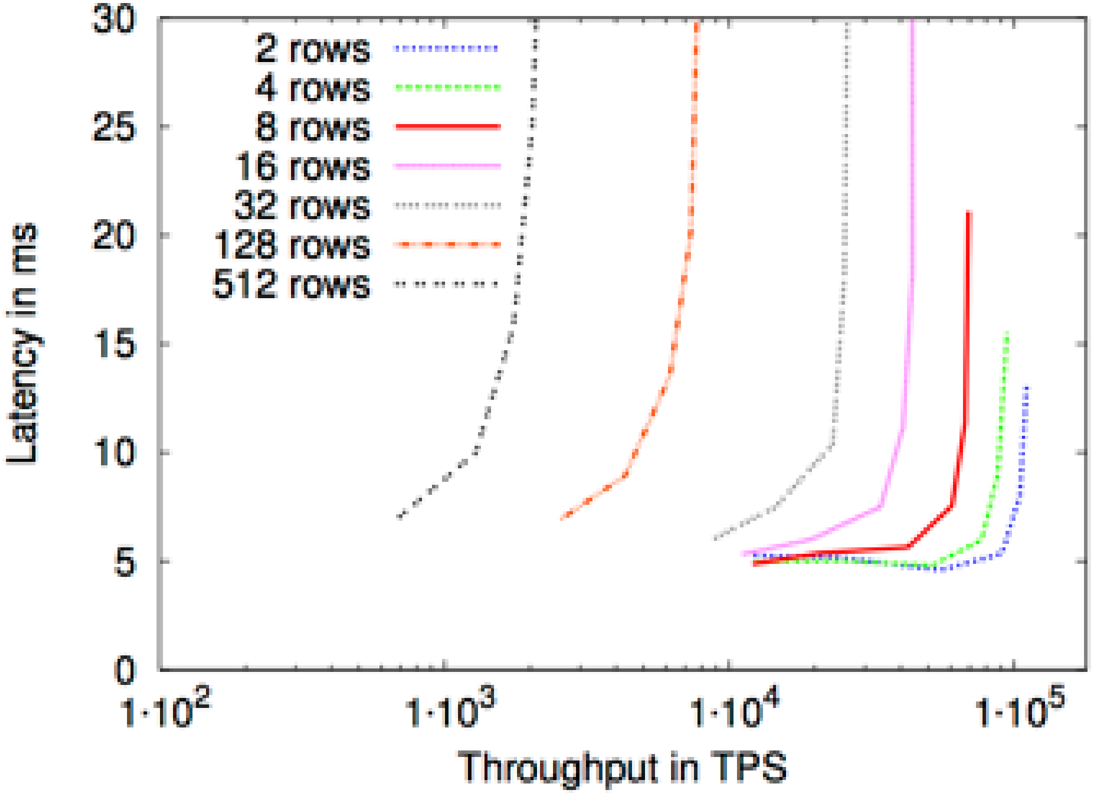
		Column C2	
Row r1		... V(T10) ...	

- Centralized Status Oracle
- Lock-free scheme

Runs on unmodified HBase

- Status replicated to clients
- Replication reduces SO load

Throughput vs. Latency



Use case

- News recommendation system
- Users with similar interests are clustered
- Upon a new article
  - Check which clusters might be interested in that article
  - Recommend article to users in the cluster
- Problems txns solve
  - Concurrent operations reconfiguring the clusters
  - Queries while clusters are being reconfigured

http://www.slideshare.net/DanielGmezFerro/omid-efficient-transaction-management-and-incremental-processing-for-hbase-hadoop-summit-europe-2013

Wrap up

Online processing

- Goal
  - Receive events
  - Make them ready for consumption fast
- Two techniques
  - Stream processing
    - Events processed against small amount of local memory
    - Very low latency (+250k events/node/s)
  - Incremental processing
    - Shared state in the form of a datastore
    - Events processed against the datastore
    - Higher latency

Acknowledgements

- S4
  - Matthieu (lead developer)
  - Daniel Gómez Ferro
  - Leo Neumeyer
  - Kishore Gopalakrishna

S4 project: http://incubator.apache.org/s4

- Omid
  - Daniel Gómez Ferro (lead developer)
  - Maysam Yabandeh
  - Ivan Kelly
  - Ben Reed

Omid project: https://github.com/yahoo/omid