UNIVERSITY OF CAMBRIDGE

# Integrating Scale Out and Fault Tolerance in Stream Processing using Operator State Management

with Raul Castro Fernandez*
Matteo Migliavacca+ and Peter Pietzuch*
*Imperial College London, +Kent Univerisity

# Big data …

**… in numbers:**

- 2.5 billions on gigabytes of data every day (source IBM)
- LSST telescope, Chile 2016, 30 TB nightly

**… come from everywhere:**


- web feeds, social networking
- mobile devices, sensors, cameras
- scientific instruments
- online transactions (public and private sectors)

**… have value:**

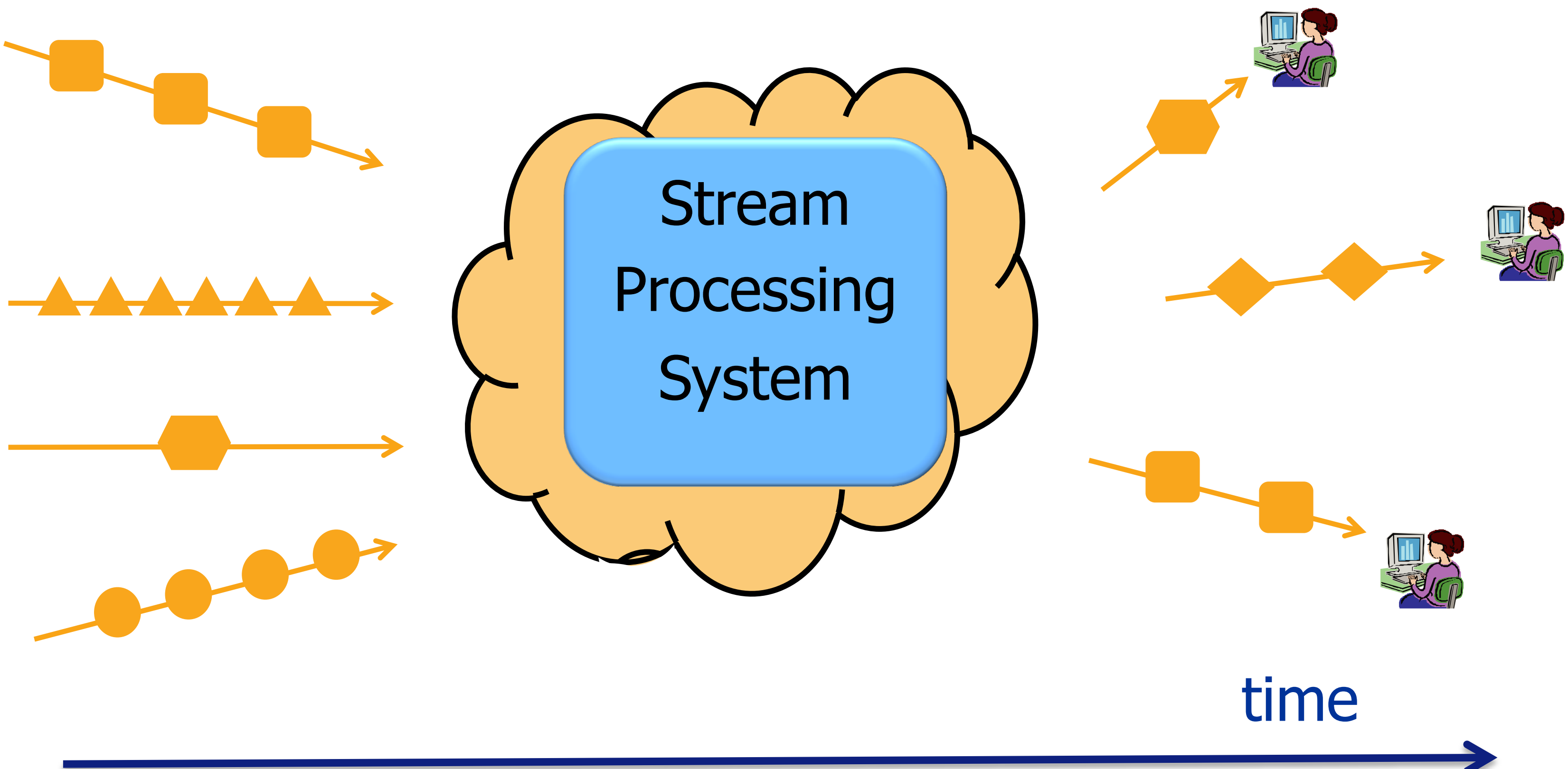
- Global Pulse forum for detecting human crises internationally
- real-time big data analytics in UK £25 billions → £216 billions in 2012-17
- recommendation applications (LinkedIn, Amazon)

☛ processing infrastructure for big data analysis

# A black-box approach for big data analysis

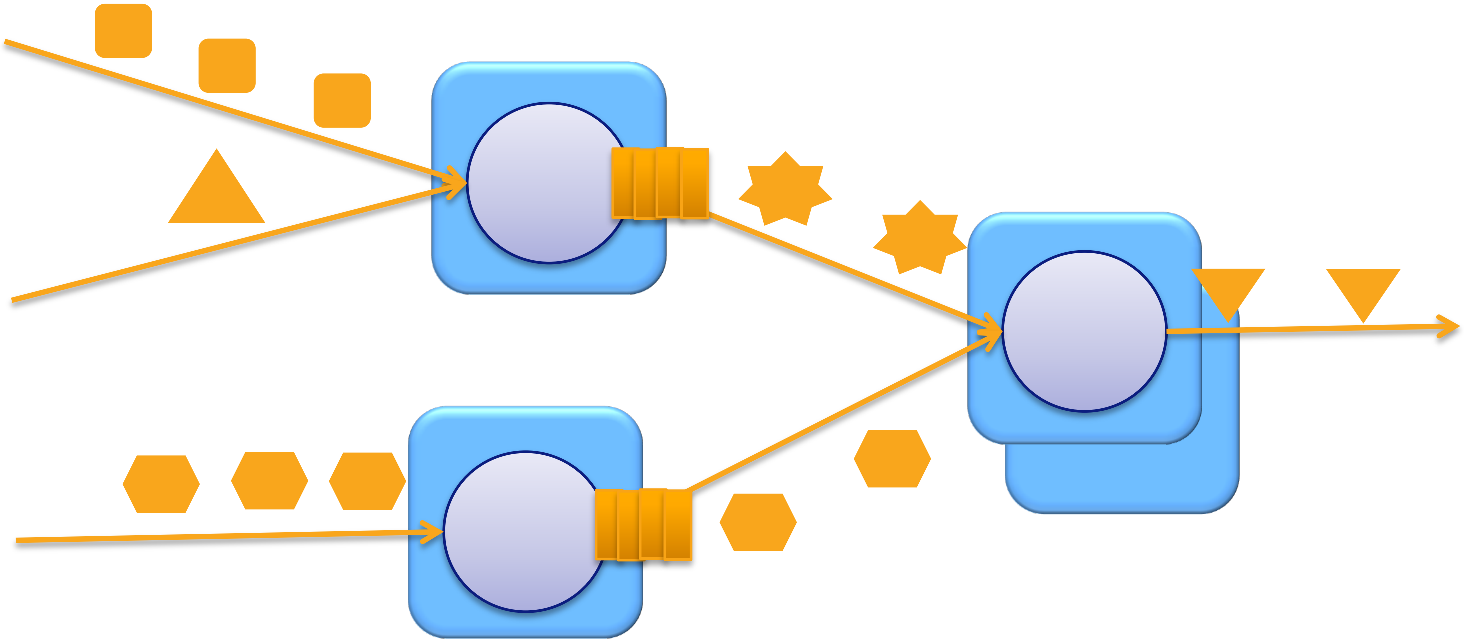
- users issue analysis ***queries*** with real-time semantics
- ***streams*** of data updates, ***time-varying rates***, generated in ***real-time***
- ***streams of result data***

✓ processing in ***near real-time***



# Distributed Stream Processing System

- queries consist of **operators** (join, map, select, ..., UDOs)
- operators form graphs
- operators process **streams of tuples** on-the-fly
- operators span nodes



# Elastic DSPSs in the Cloud

**Real-time big data analysis challenge traditional DSPS:**

**?** what about continuous workload surges?

**?** what about real-time resource allocation to workload variations?

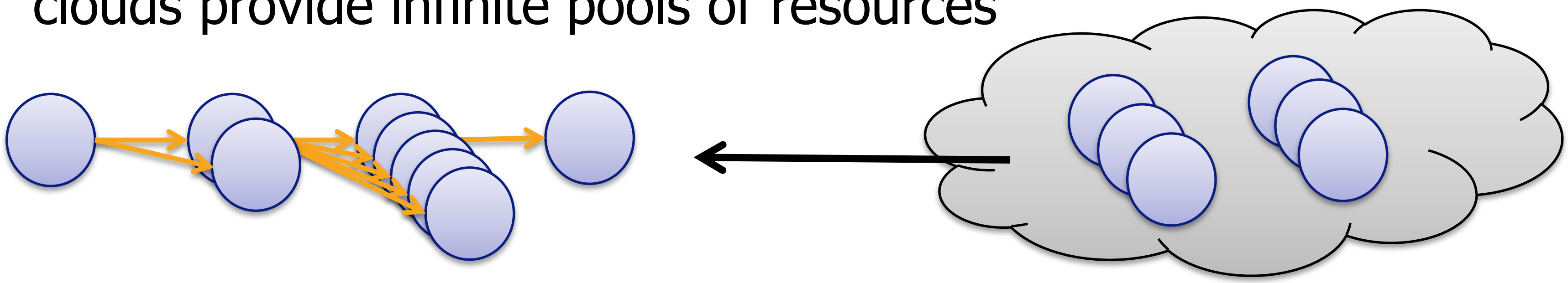
**?** keeping the state correct for *stateful* operators?

**Massively scalable , cloud-based DSPSs [SIGMOD 2013]**

1. gracefully handles **stateful** operators' state
2. operator **state management** for **combined** scale out and fault tolerance
3. SEEP system and evaluation
4. related work
5. future research directions

# Stream Processing in the Cloud

clouds provide infinite pools of resources



**?** How do we build a stream processing platform in the Cloud?

**Intra-query parallelism:**

- provisioning for workload peaks unnecessarily conservative

☛ **dynamic scale out:** increase resources when peaks appear

**Failure resilience:**

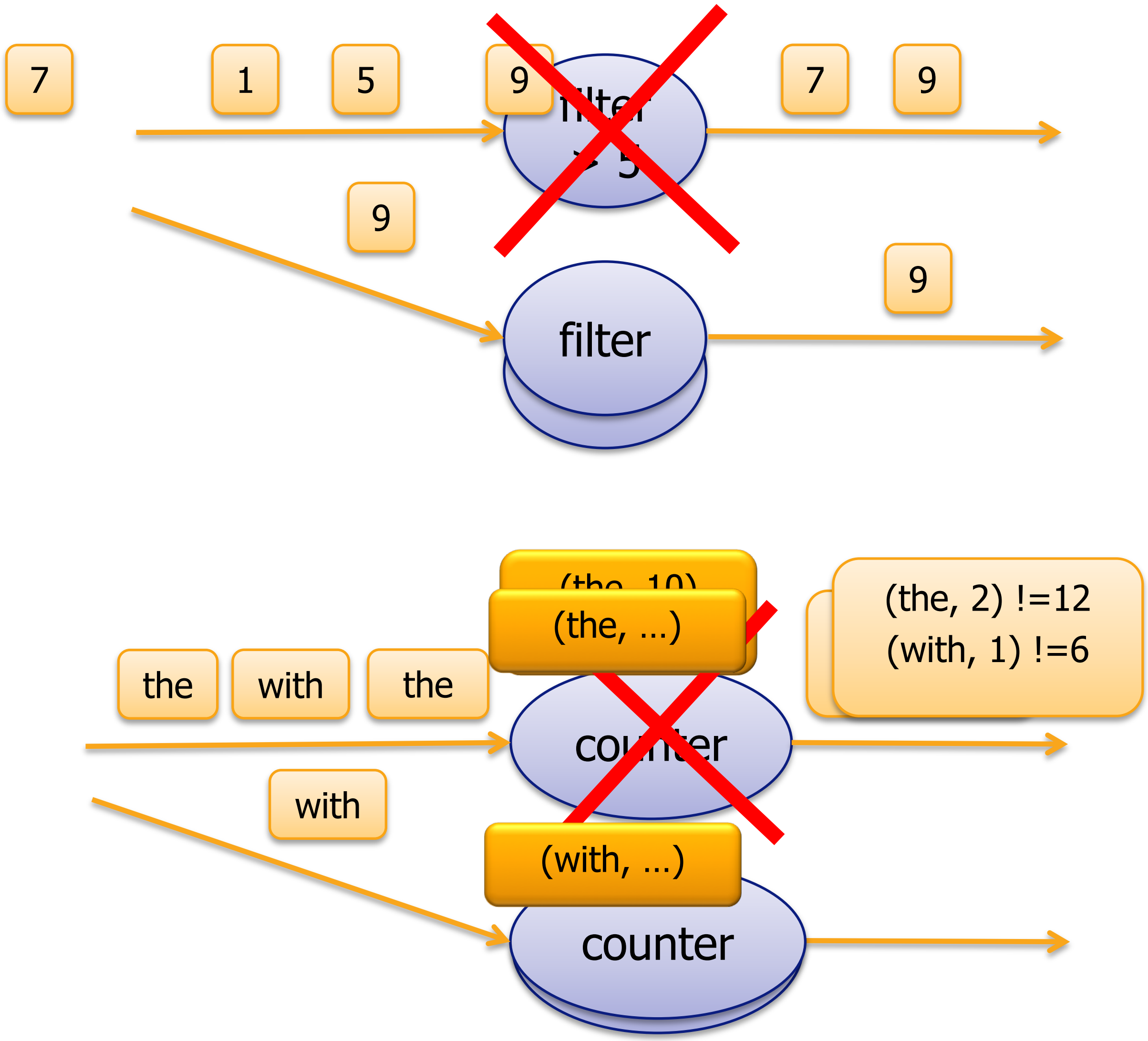
- active fault-tolerance needs 2x resources
- passive fault-tolerance leads to long recovery times

☛ **hybrid fault-tolerance:** low resource overhead with fast recovery

☛ Both mechanisms must support **stateful** operators

# Stateless vs Stateful Operators

**operator state:** a summary of past tuples' processing




**stateless:**

- ✓ failure recovery
- ✓ scale out

**stateful:**

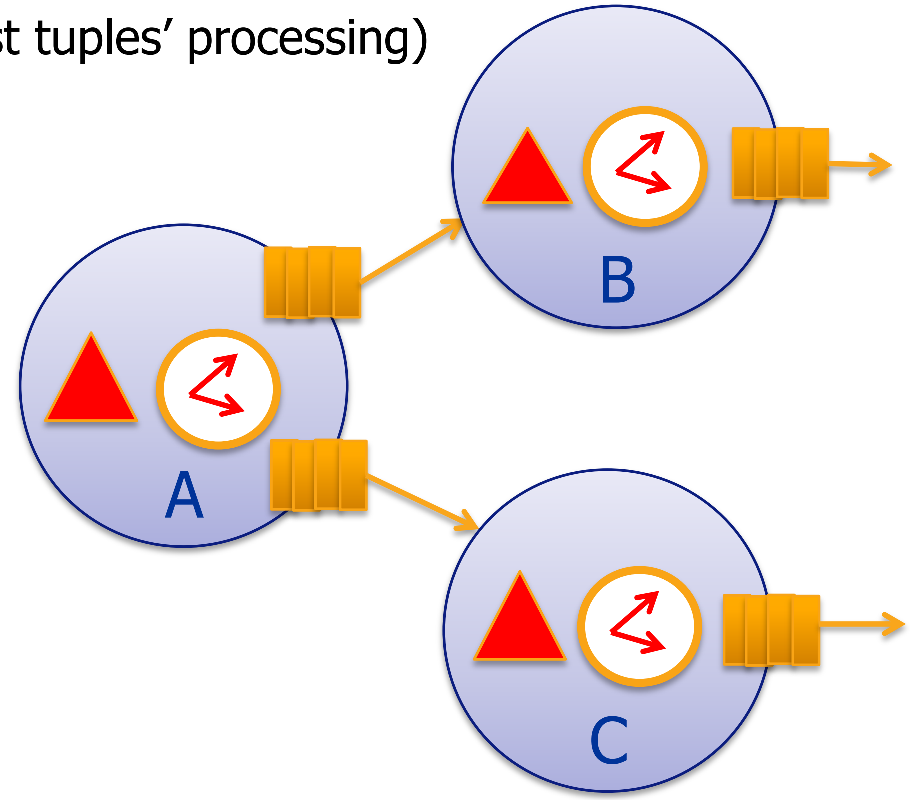
- × failure recovery
- × scale out

# State Management

**processing state**: (summary of past tuples' processing)

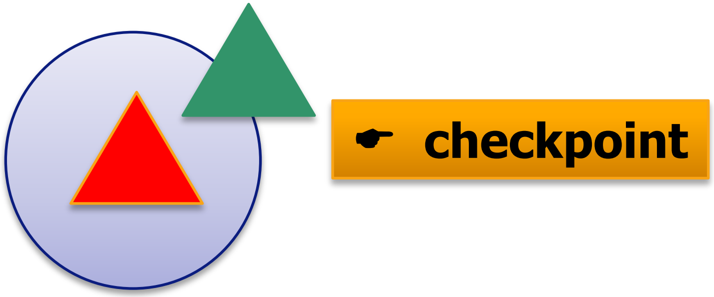
**routing state**: (routing of tuples)

**buffer state**: (tuples)

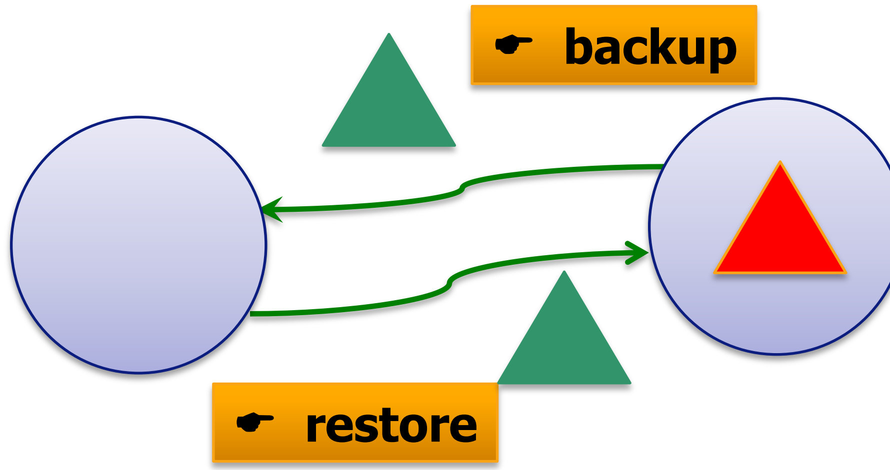



- operator state is an external entity managed by the DSPS
- **primitives** for state management
- **mechanisms** (scale out, failure recovery) on top of primitives
- **dynamic reconfiguration** of the dataflow graph

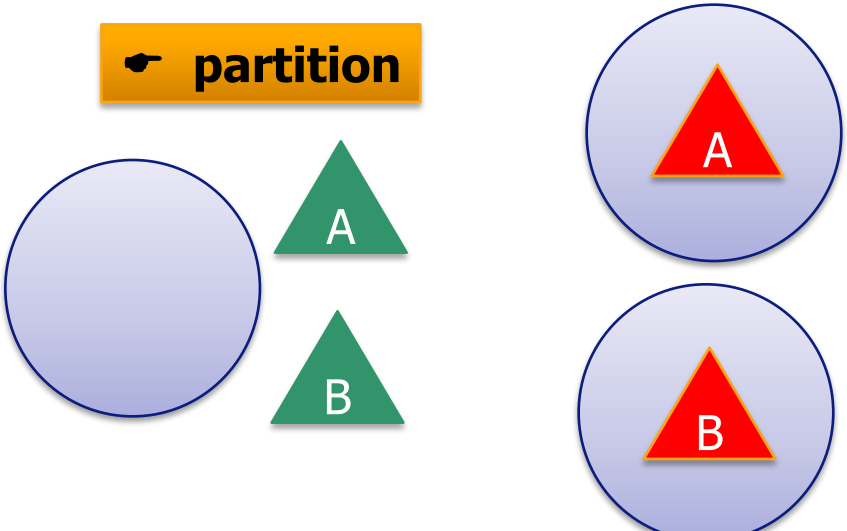
# State Management Primitives




takes snapshot of state and makes it externally available




moves copy of state from one operator to another

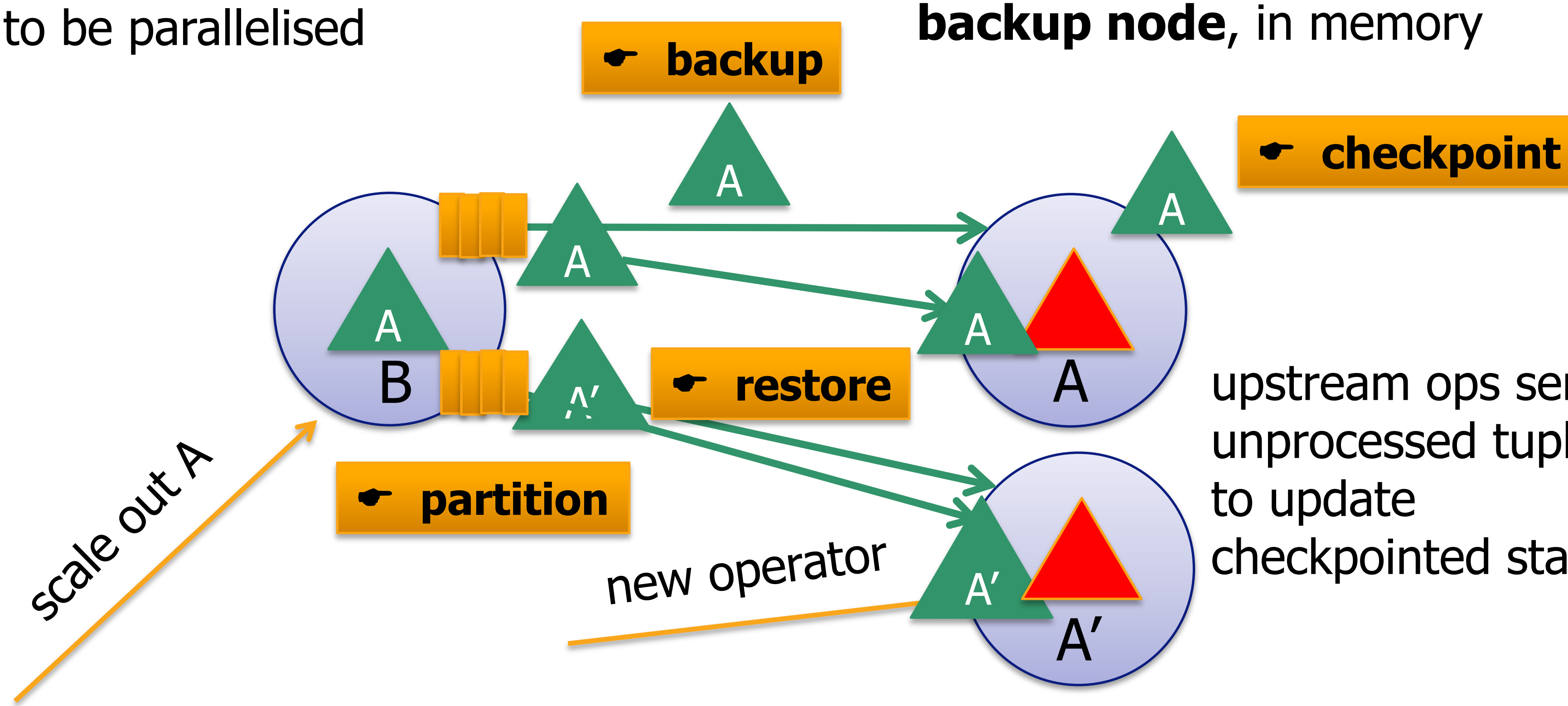



splits state in a semantically correct fashion for parallel processing

# State Management Scale Out, Stateful Ops

backup node already has state of operator to be parallelised

periodically, stateful operators checkpoint and back up state to designated **upstream backup node**, in memory

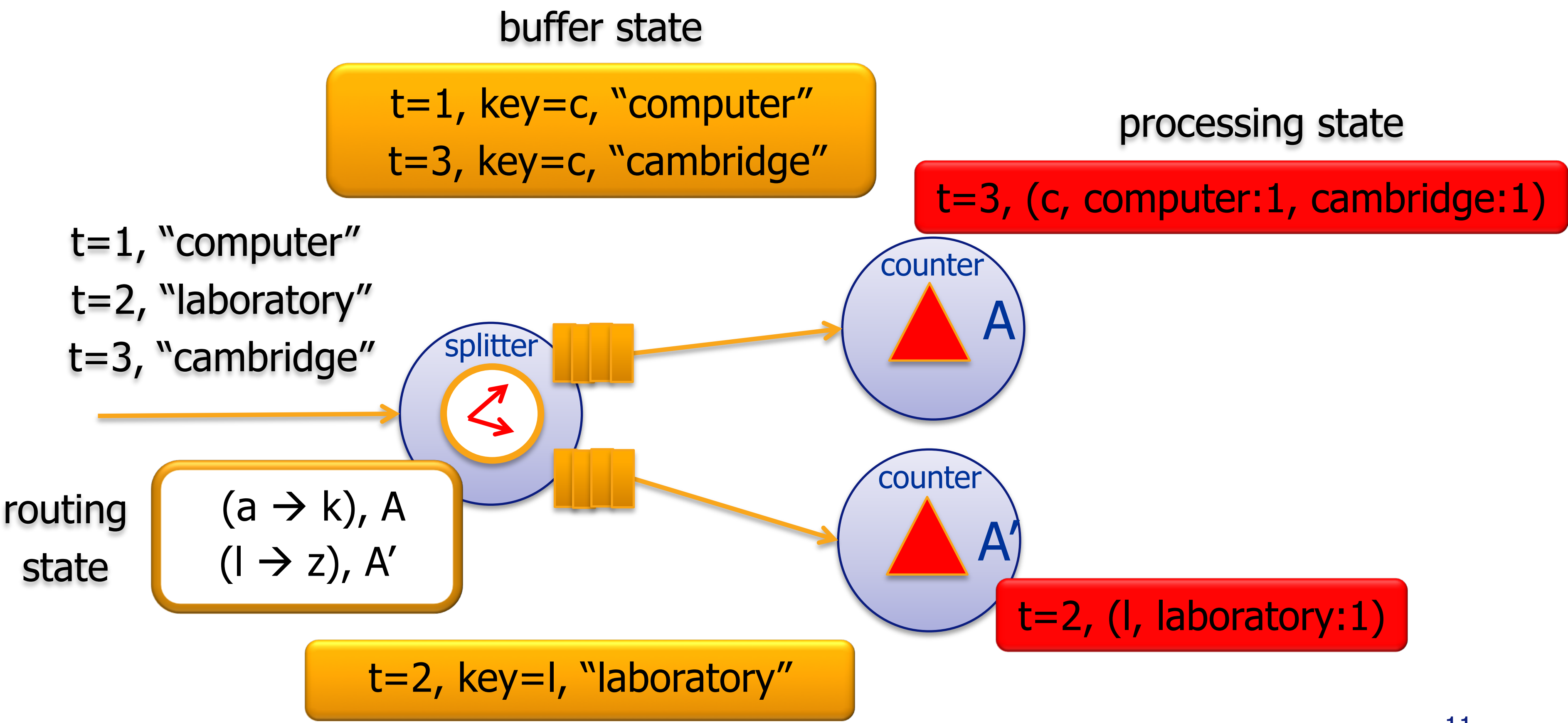



upstream ops send unprocessed tuples to update checkpointed state

☛ How do we partition **stateful** operators?

# Partitioning Stateful Operators

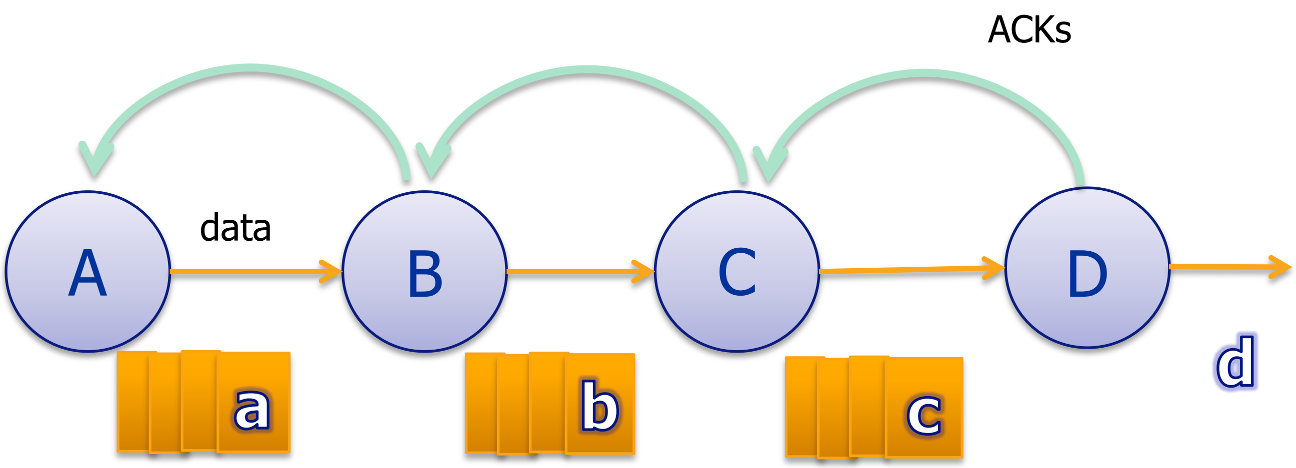
1. Processing state modeled as (key, value) dictionary
2. State partitioned according to key k of tuples
3. Tuples will be routed to correct operator as of k



# Passive Fault-Tolerance Model

recreate operator state by replaying tuples after failure:

- **upstream backup**: sends acks upstream for tuples processed downstream



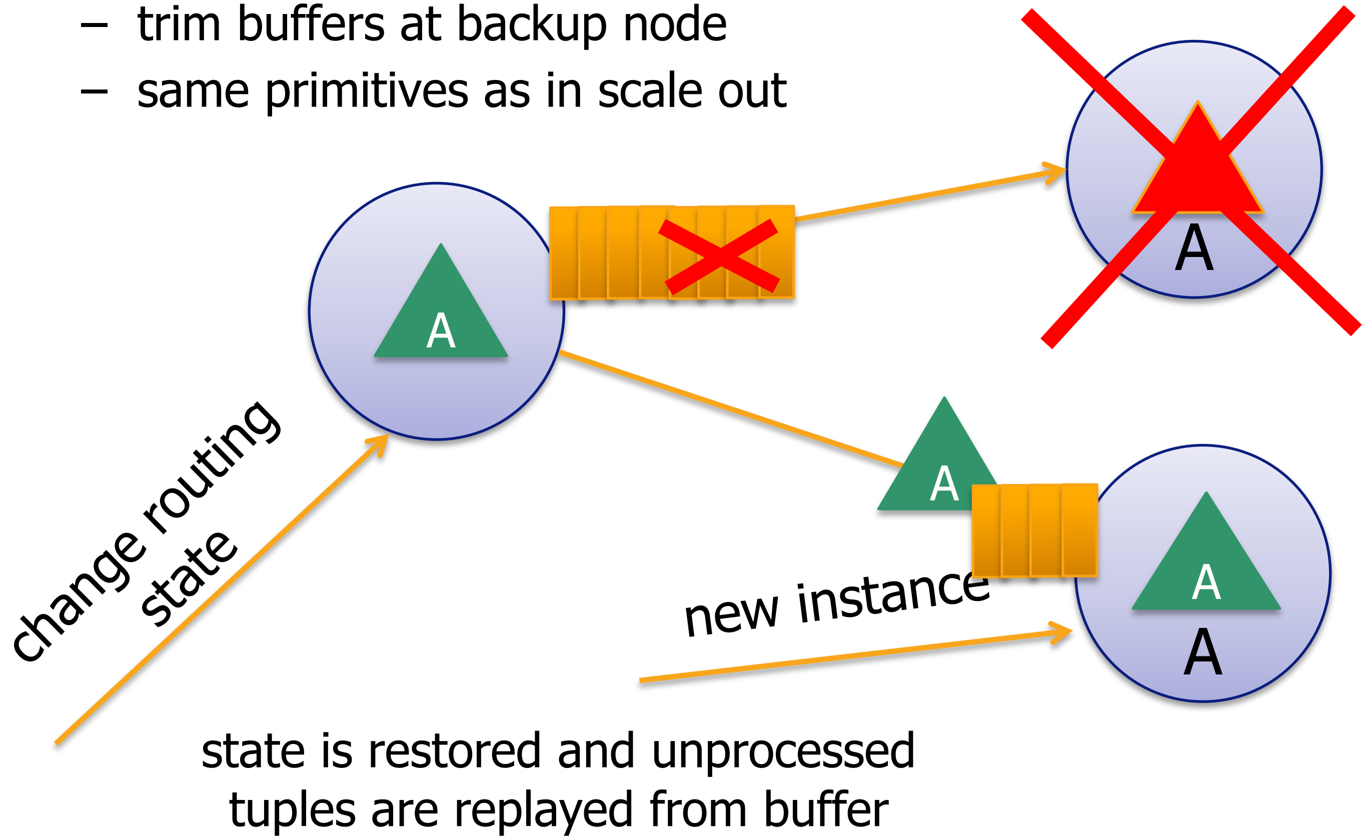

may result in long recovery times due to large buffers:

- system is reprocessing streams after failure ➔ inefficient

# Recovering using State Management (R+SM)

**Benefit from state management primitives:**

- use periodically backed up state on upstream node to recover faster
- trim buffers at backup node
- same primitives as in scale out

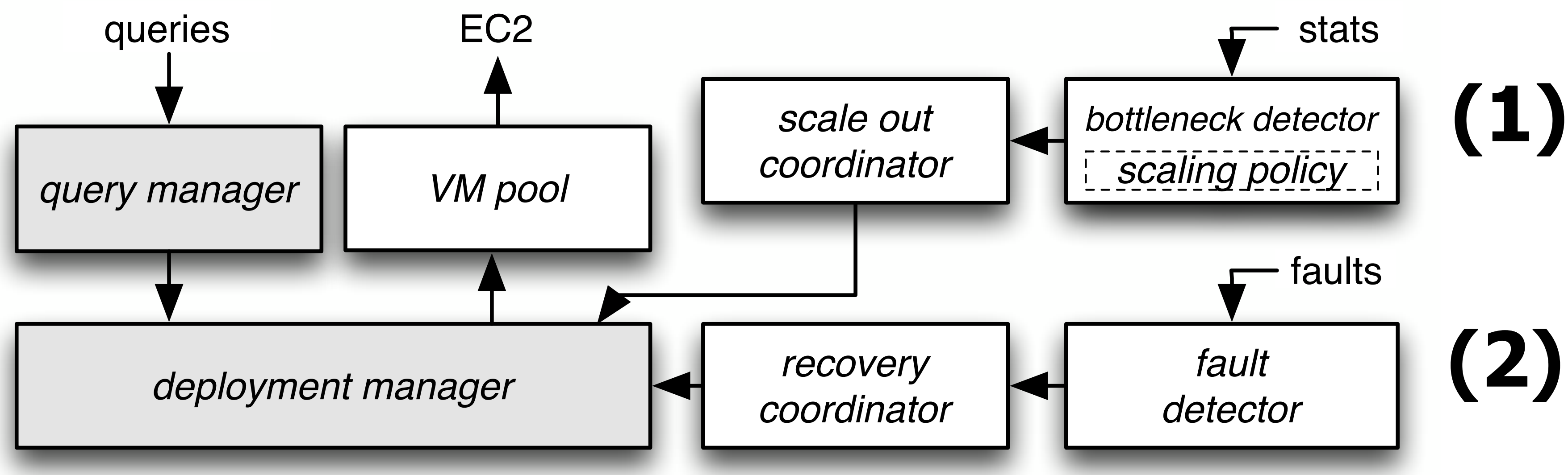



☛ same primitives for parallel recovery

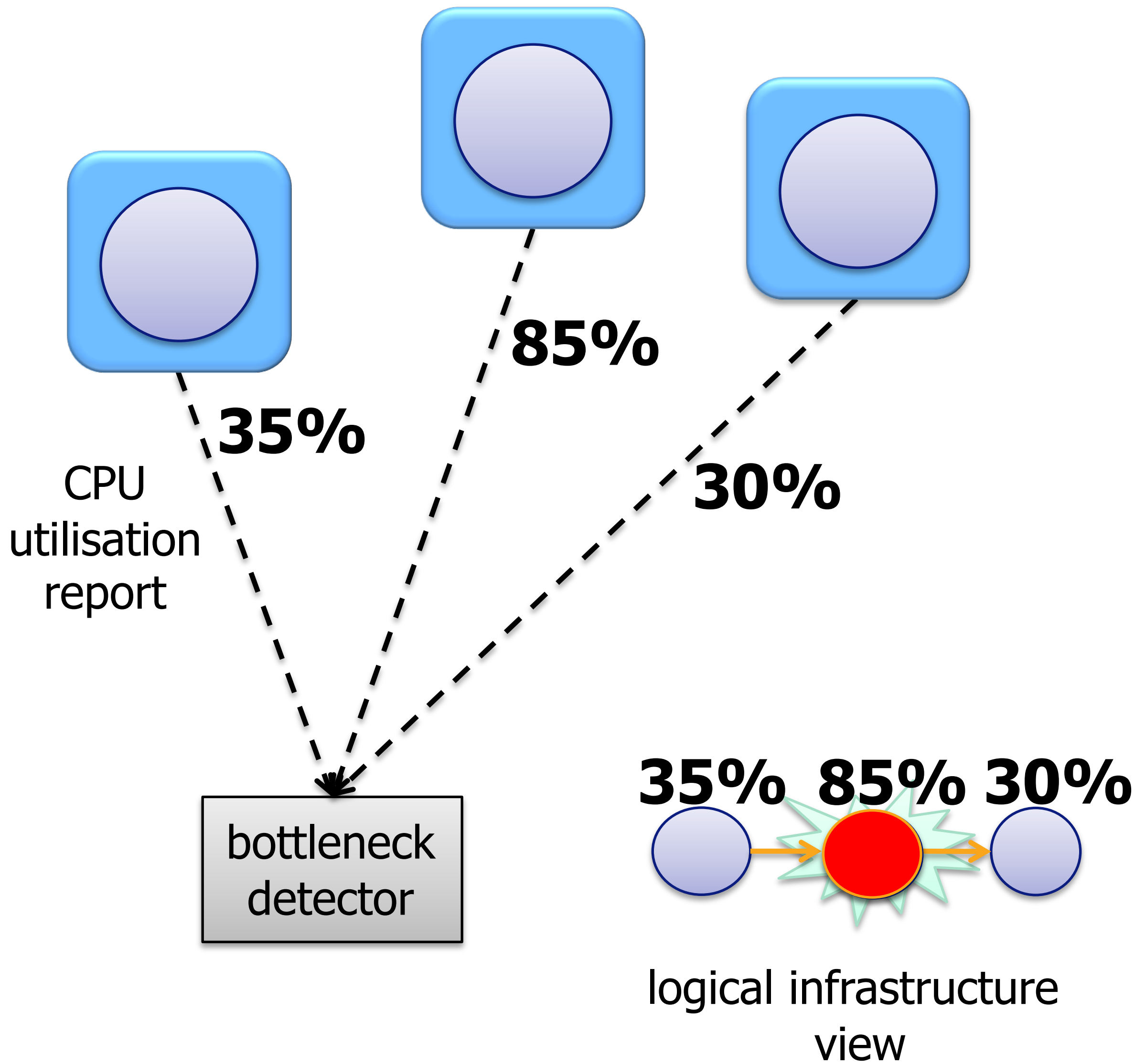
# State Management in Action: SEEP

**(1) dynamic Scale Out**: detect bottleneck , add new parallelised operator

**(2) failure Recovery**: detect failure, replace with new operator

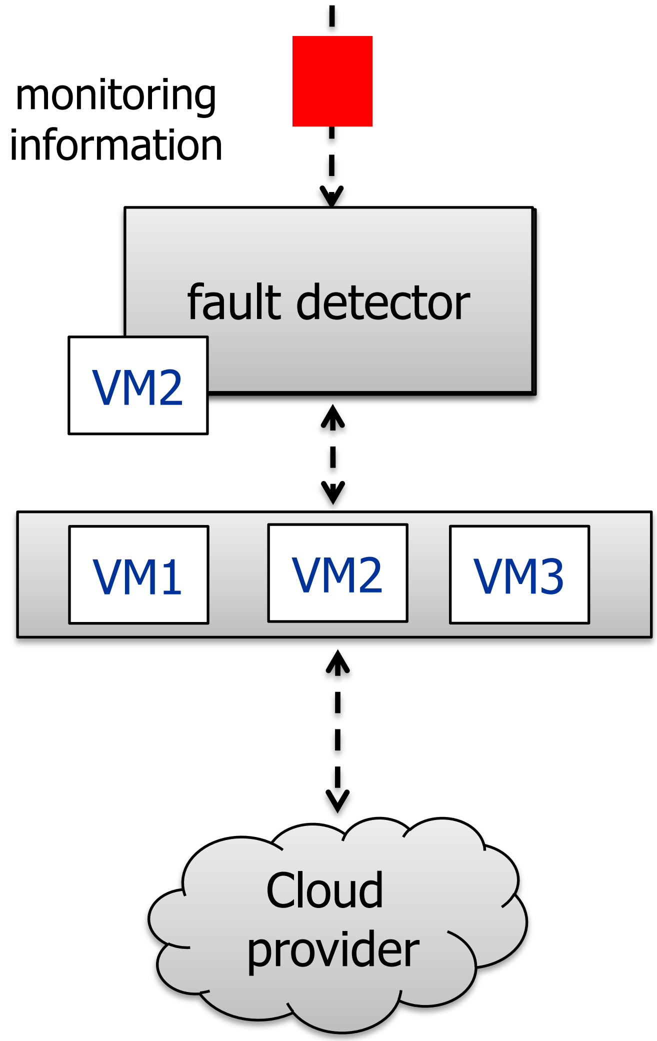


# Dynamic Scale Out: Detecting bottlenecks



# The VM Pool: Adding operators

**problem**: allocating new VMs takes minutes...




**virtual machine pool**

(dynamic pool size)

add new VM to pool

provision VM from cloud (order of mins)

# Experimental Evaluation

**Goals:**

- investigate effectiveness of **scale out** mechanism
- recovery time after failure using **R+SM**
- overhead of **state management**

**Scalable and Elastic Event Processing (SEEP):**

- implemented in Java; Storm-like data flow model

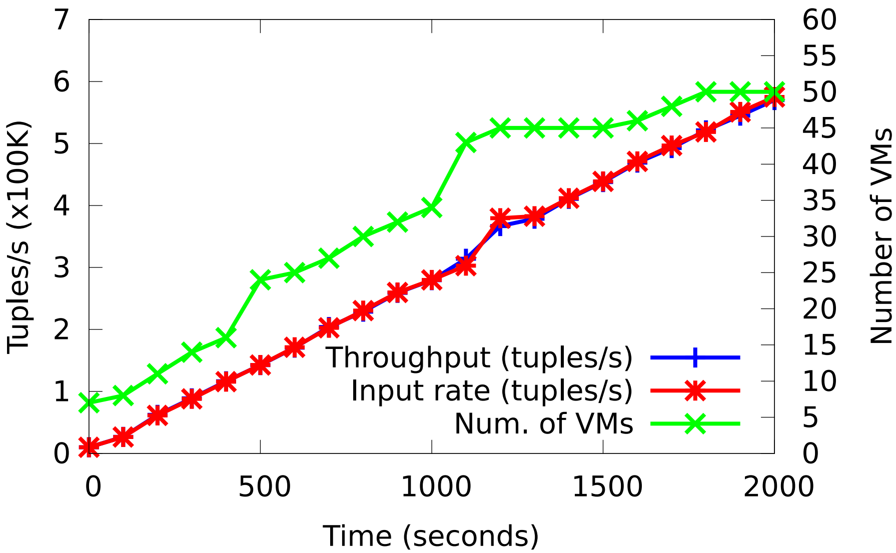
**Sample queries + workload**

- **Linear Road Benchmark** (LRB) to evaluate scale out [VLDB'04]
  - provides an increasing stream workload over time
  - query with 8 operators, 3 are stateful; SLA: results < 5 secs
- **Windowed word count query** (2 ops) to evaluate fault tolerance
  - induce failure to observe performance impact

**Deployment on Amazon AWS EC2**

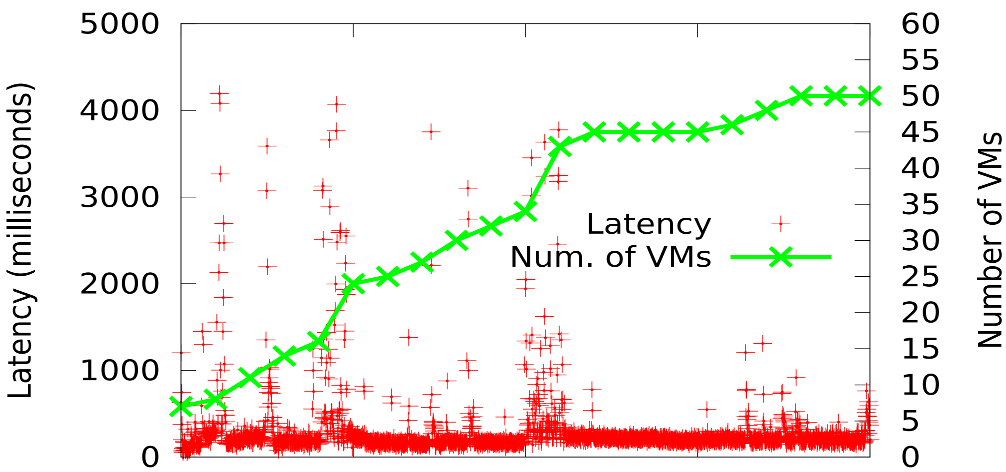
- sources and sinks on high-memory double extra large instances
- operators on small instances

# Scale Out: LRB Workload




**scales to load factor L=350 with 50 VMs on Amazon EC2**
(automated query parallelisation, scale out policy at 70%)

**L=512 highest result [VLDB'12]**
(hand-crafted query on cluster)




**scale out leads to latency peaks, but remains within LRB SLA**

☛ SEEP scales out to increasing workload in the Linear Road Benchmark

# Conclusions

**Stream processing will grow in importance:**

- handling the data deluge
- enables real-time response and decision making

**Integrated approach for scale out and failure recovery:**

- operator state an independent entity
- primitives and mechanisms

**Efficient approach extensible for additional operators:**

- effectively applied to Amazon EC2 running LRB
- parallel recovery