Real-time Analytics with HBase

Alex Baranau, Sematext International

(short version)

About me

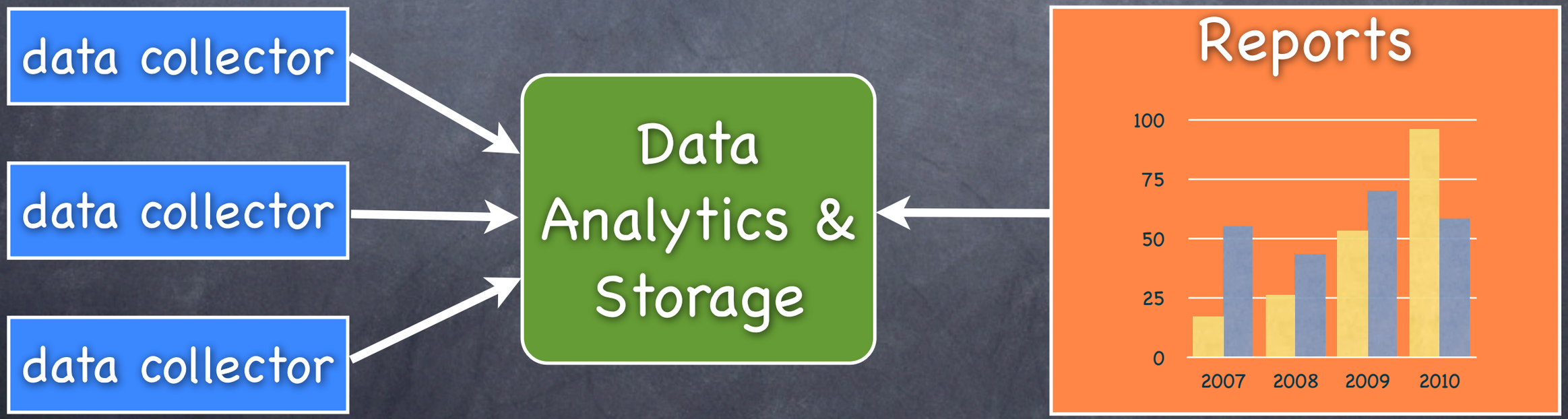
- Software Engineer at Sematext International
- http://blog.sematext.com/author/abaranau
- @abaranau
- http://github.com/sematext (abaranau)

Plan

- Problem background: what? why?
- Going real-time with append-only updates approach: how?
- Open-source implementation: how exactly?
- Q&A

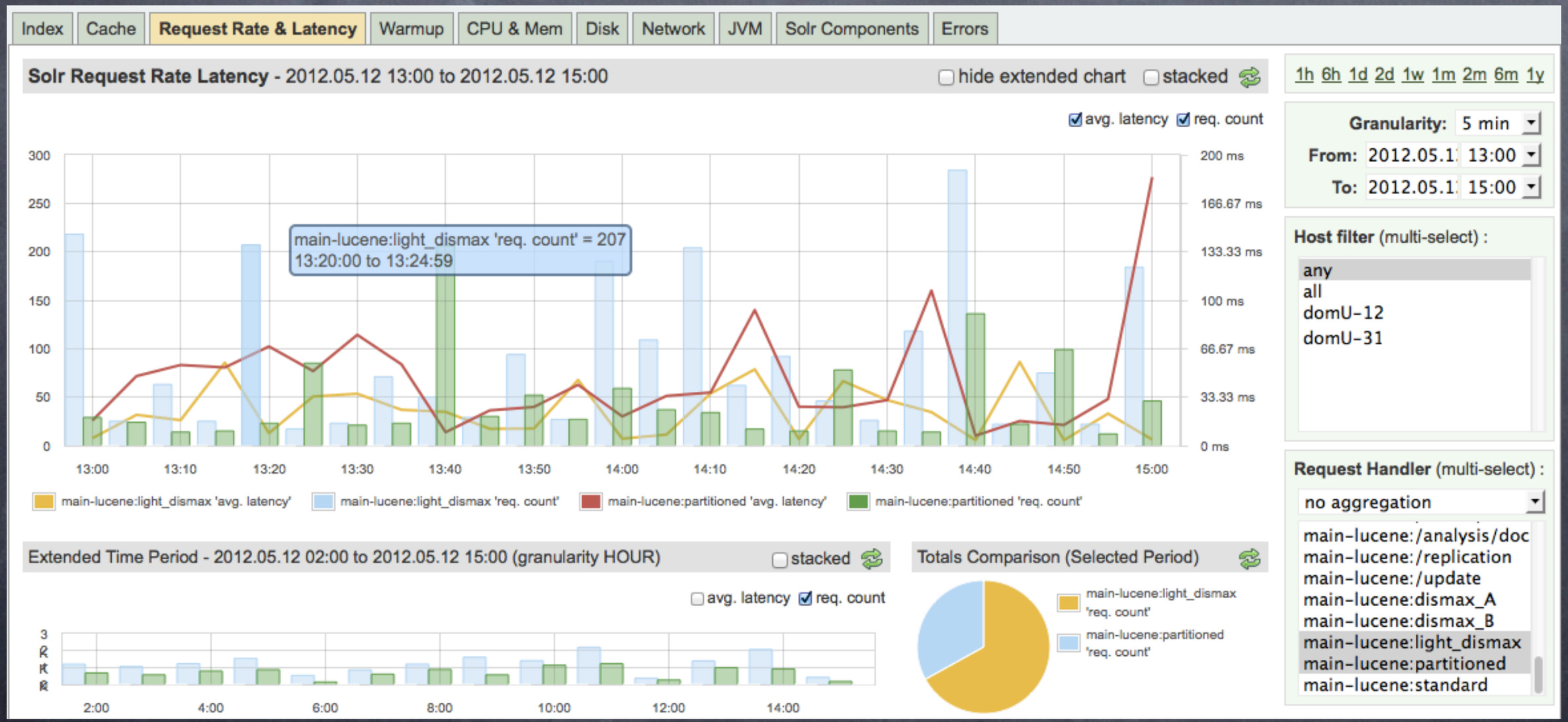
Background: our services

- Systems Monitoring Service (Solr, HBase, ...)
- Search Analytics Service



Background: Report Example

Search engine (Solr) request latency

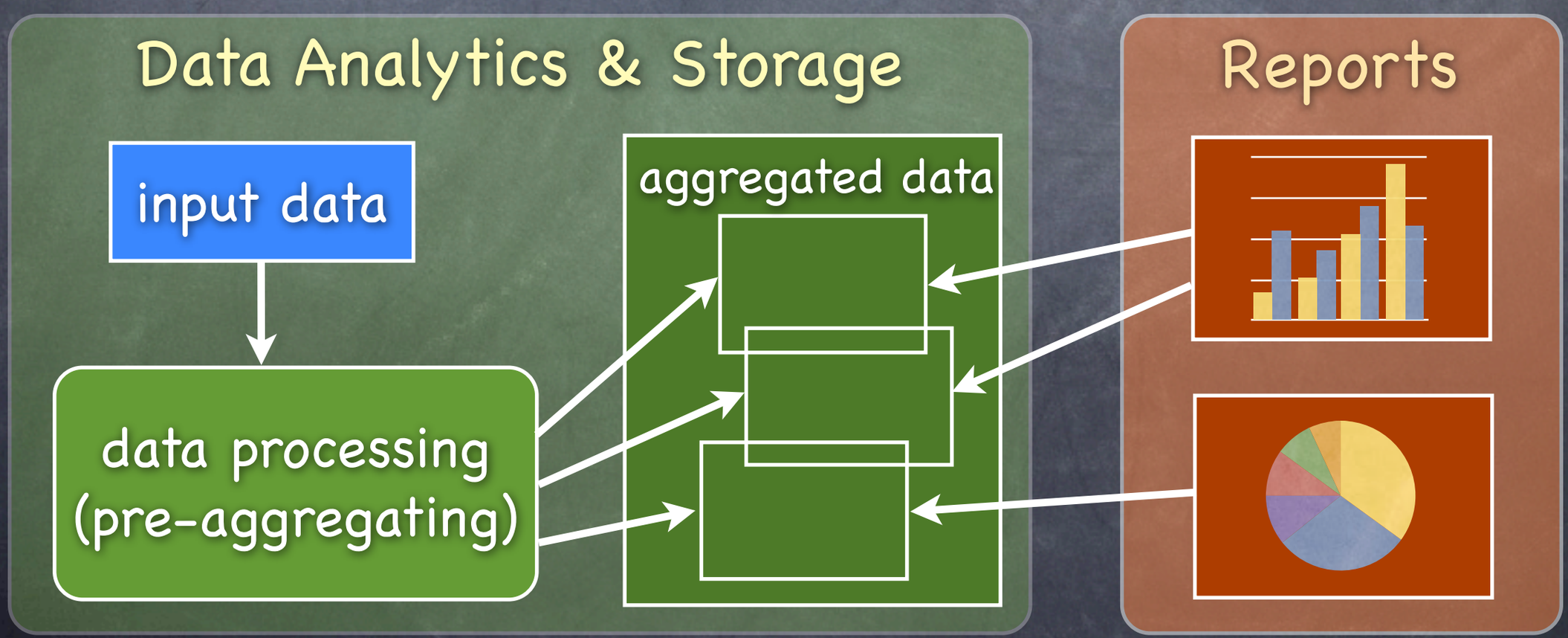


Background: requirements

- High volume of input data
- Multiple filters/dimensions
- Interactive (fast) reports
- Show wide range of data intervals
- Real-time data changes visibility
- No sampling, accurate data needed

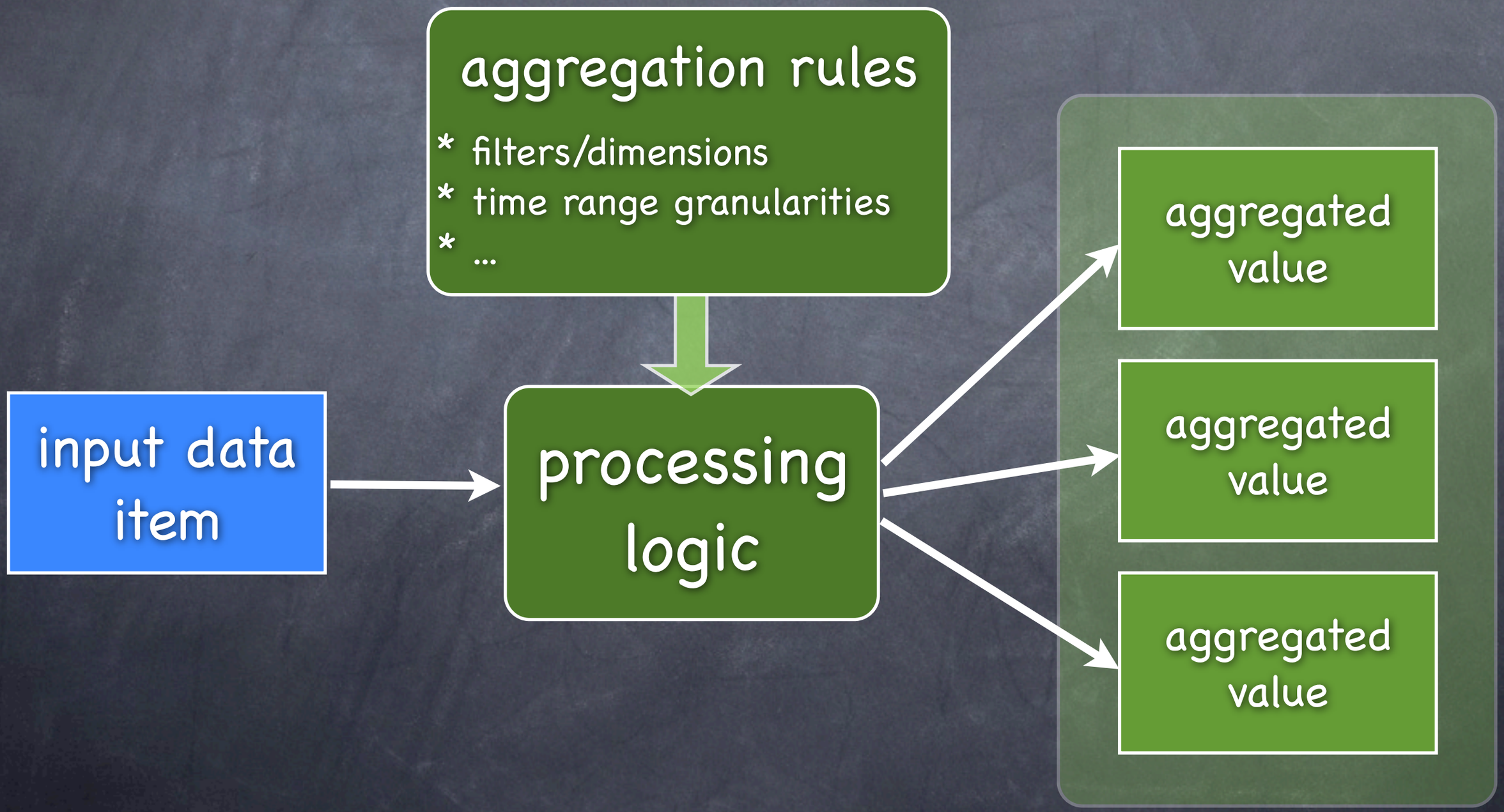
Background: serve raw data?

- simply storing all data points doesn't work
  - to show 1-year worth of data points collected every second 31,536,000 points have to be fetched
- pre-aggregation (at least partial) needed



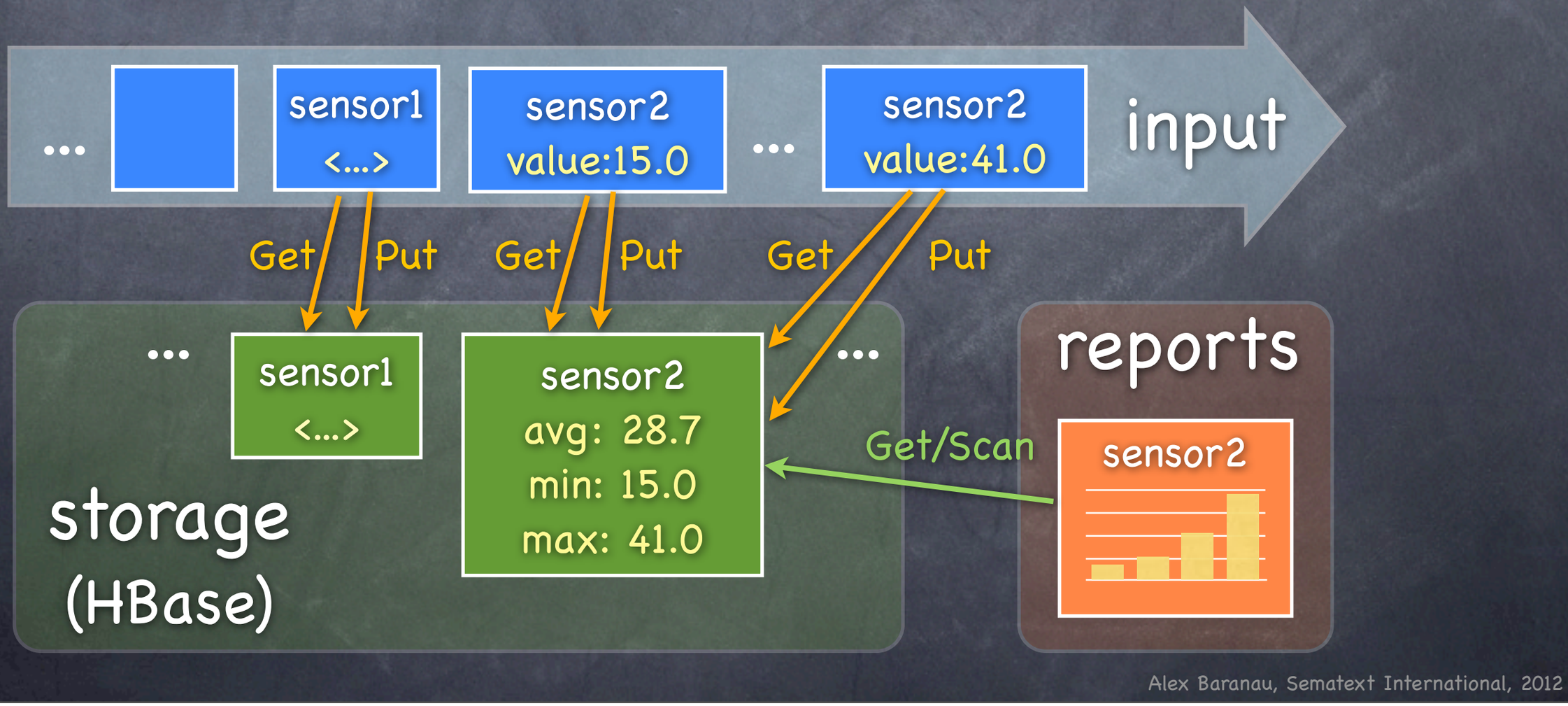
Background: pre-aggregation

OLAP-like Solution



Background: RMW updates are slow

- more dimensions/filters -> greater output data vs input data ratio
- individual ready-modify-write (Get+Put) operations are slow and not efficient (10-20+ times slower than only Puts)



Background: batch updates

- More efficient data processing: multiple updates processed at once, not individually
- Decreases aggregation output (per input record)
- Reliable, no data loss in case of failures

- Not real-time
- If done frequently (closer to real-time), still a lot of costly Get+Put update operations
- Handling of failures of tasks which partially wrote data to HBase is complex

Going Real-time

with

Append-based Updates

Append-only: main goals

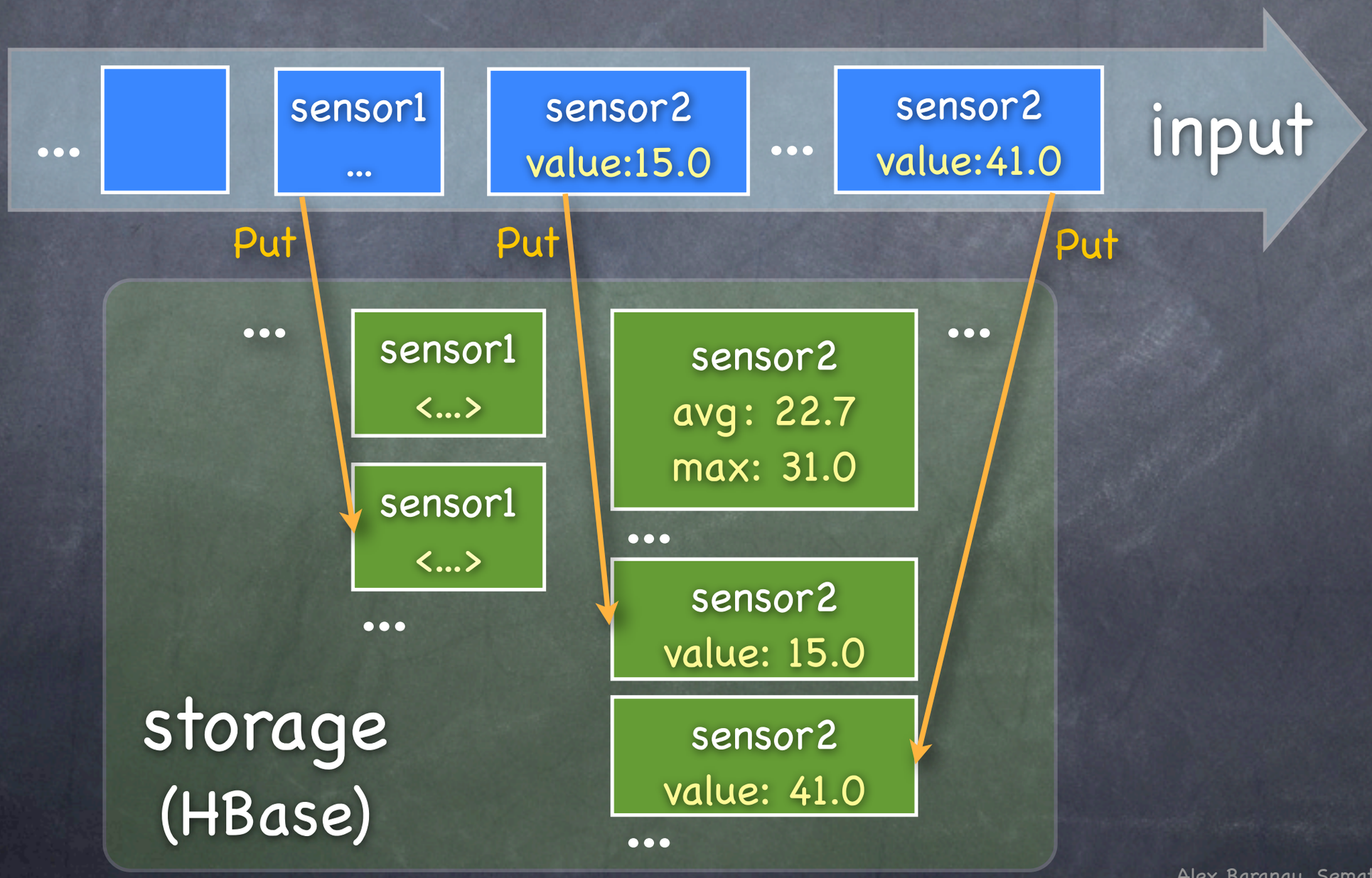
- Increase record update throughput
- Process updates more efficiently: reduce operations number and resources usage
- Ideally, apply high volume of incoming data changes in real-time
- Add ability to roll back changes
- Handle well high update peaks

Append-only: how?

1. Replace read-modify-write (Get+Put) operations at write time with simple append-only writes (Put)
2. Defer processing of updates to periodic jobs
3. Perform processing of updates on the fly only if user asks for data earlier than updates are processed.

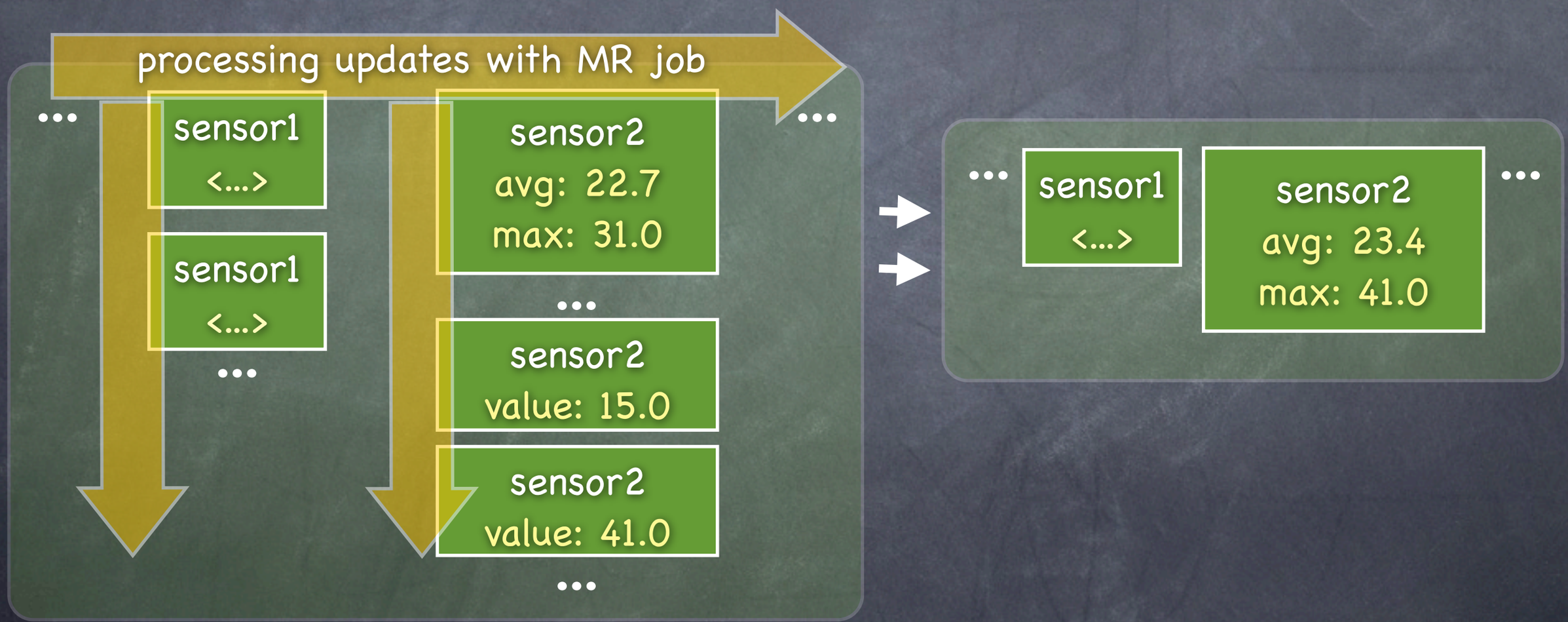
Append-only: writing updates

1 Replace update (Get+Put) operations at write time with simple append-only writes (Put)



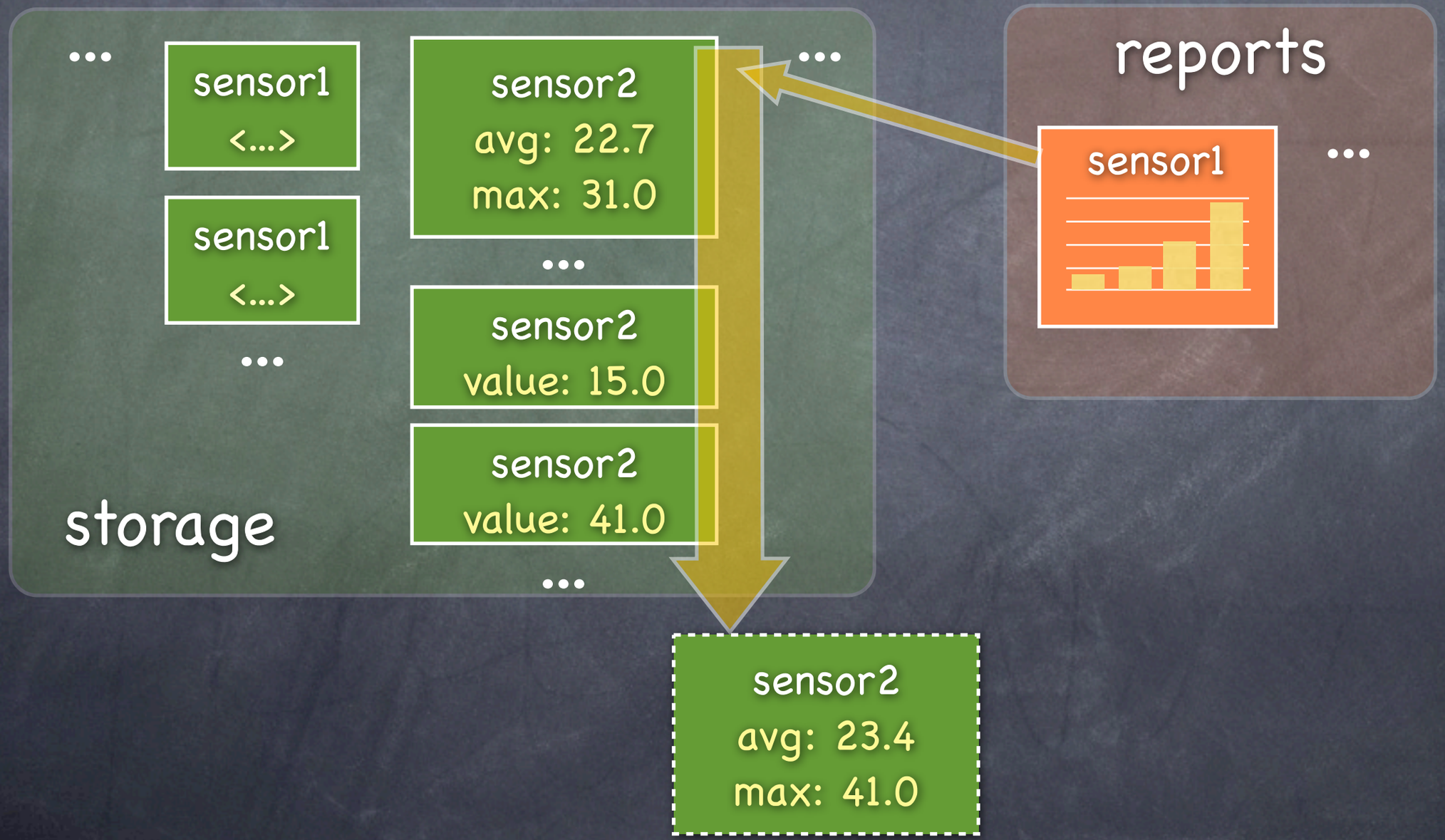
Append-only: writing updates

2 Defer processing of updates to periodic jobs



Append-only: writing updates

3 Perform aggregations on the fly if user asks for data earlier than updates are processed



Append-only: benefits

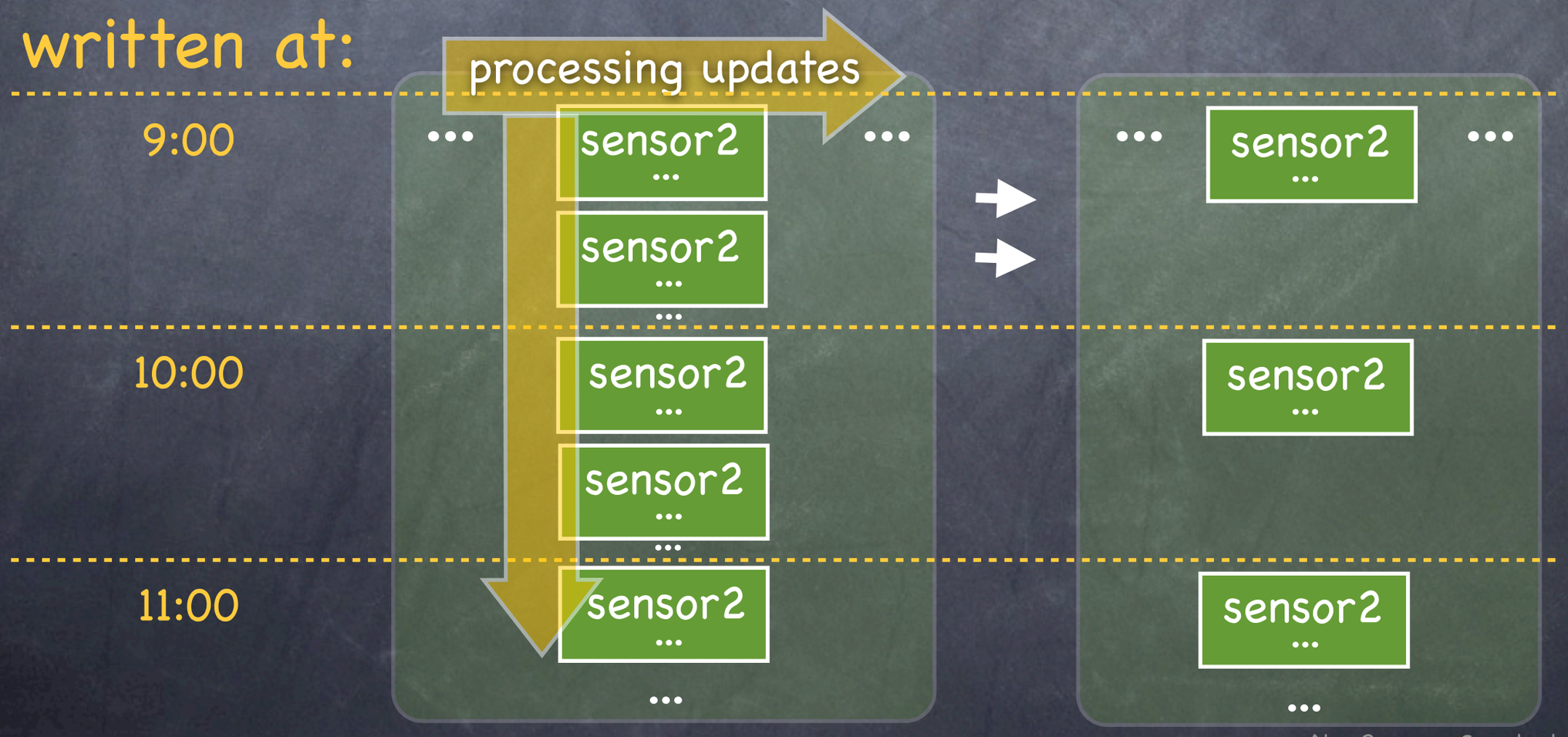
- High update throughput
- Real-time updates visibility
- Efficient updates processing
- Handling high peaks of update operations
- Ability to roll back any range of changes
- Automatically handling failures of tasks which only partially updated data (e.g. in MR jobs)
- Update operation becomes idempotent & atomic, easy to scale writers horizontally

Append-only: efficient updates

- To apply N changes:
  - N Get+Put operations replaced with
  - N Puts and 1 Scan (shared) + 1 Put operation
- Applying N changes at once is much more efficient than performing N individual changes
  - Especially when updated value is complex (like bitmaps), takes time to load in memory
  - Skip compacting if too few records to process
- Avoid a lot of redundant Get operations when large portion of operations - inserting new data

Append-only: rollback

- Rollbacks are easy when updates were not processed yet (not merged)
- To preserve rollback ability after they are processed (and result is written back), updates can be compacted into groups



Append-only: idempotency

- Using append-only approach helps recover from failed tasks which write data to HBase
  - without rolling back partial updates
  - avoids applying duplicate updates
  - fixes task failure with simple restart of task
- Note: new task should write records with same row keys as failed one
  - easy, esp. given that input data is likely to be same
- Very convenient when writing from MapReduce
- Updates processing periodic jobs are also idempotent

Append-only: cons

- Processing on the fly makes reading slower
- Looking for data to compact (during periodic compactions) may be inefficient
- Increased amount of stored data depending on use-case (in 0.92+)

Append-only updates implementation

HBaseHUT

HBaseHUT: Overview

- Simple
- Easy to integrate into existing projects
  - Packed as a singe jar to be added to HBase client classpath (also add it to RegionServer classpath to benefit from server-side optimizations)
  - Supports native HBase API: HBaseHUT classes implement native HBase interfaces
- Apache License, v2.0

HBaseHUT: Overview

- Processing of updates on-the-fly (behind ResultScanner interface)
  - Allows storing back processed Result
  - Can use CPs to process updates on server-side
- Periodic processing of updates with Scan or MapReduce job
  - Including processing updates in groups based on write ts
- Rolling back changes with MapReduce job

HBaseHUT: API overview

Writing data:

```
Put put = new Put(HutPut.adjustRow(rowKey));
// ...
hTable.put(put);
```

Reading data:

```
Scan scan = new Scan(startKey, stopKey);
ResultScanner resultScanner =
    new HutResultScanner(hTable.getScanner(scan),
                         updateProcessor);

for (Result current : resultScanner) {...}
```

HBaseHUT: API overview

Example UpdateProcessor:

```
public class MaxFunction extends UpdateProcessor {
  // ... constructor & utility methods

  @Override
  public void process(Iterable<Result> records,
                      UpdateProcessingResult result) {
    Double maxVal = null;

    for (Result record : records) {
      double val = getValue(record);
      if (maxVal == null || maxVal < val) {
        maxVal = val;
      }
    }

    result.add(colfam, qual, Bytes.toBytes(maxVal));
  }
}
```

HBaseHUT: Next Steps

- Wider CPs (HBase 0.92+) utilization
  - Process updates during memstore flush
- Make use of Append operation (HBase 0.94+)
- Integrate with asynchbase lib
- Reduce storage overhead from adjusting row keys
- etc.
- Contributors are welcome!

Qs?

- http://github.com/sematext/HBaseHUT
- http://blog.sematext.com
- @abaranau
- http://github.com/sematext (abaranau)
- http://sematext.com, we are hiring! ;)
- there will be a longer version of the presentation on the web