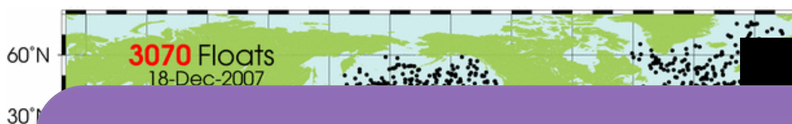
Skew and Failures during Parallel Data Processing

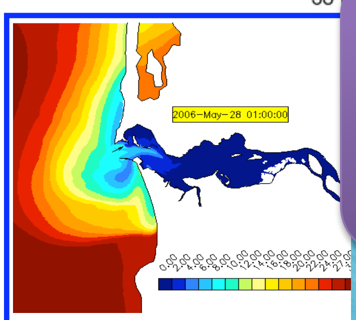
Magdalena Balazinska
UNIVERSITY OF WASHINGTON
http://www.cs.washington.edu/homes/magda/

Science is Facing a Data Deluge!

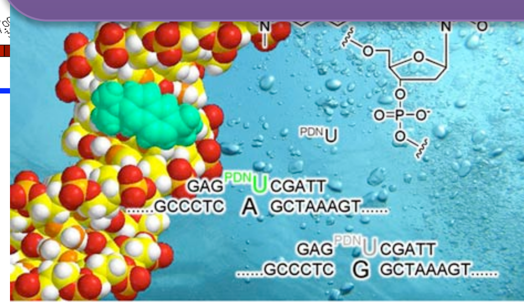
- Astronomy: High-resolution, high-frequency sky surveys (SDSS, LSST)
- Medicin
- Biology:
- Oceanog ... tellites
- Etc.

Scientists need new tools and techniques to effectively analyze all this data!





This need extends beyond science!



Nuage and CQMS Projects

http://nuage.cs.washington.edu/ and http://cqms.cs.washington.edu/

Goal:
- Big-data analytics
- Cloud computing
- Emphasis on scientific apps

SciDB
- Parallel Array DBMS

High-Performance Big-Data Analytics (Nuage)
- **SkewReduce**: Skew resistant proc. of complex functions [SSDBM 2010, SOCC 2010]
- **FTOpt**: Fault-tolerance optimization in parallel systems [SIGMOD 2011]
- **HaLoop**: Support for iterative MapReduce processing [VLDB 2010)
- **SciDB**: Parallel array-based system [SIGMOD 2010, SIGMOD 2011]

Easier Analytics (CQMS/Nuage)
- **ParaTimer**: Progress estimation for MapReduce DAGs [SIGMOD 2010, ICDE 2010]
- **SnipSuggest**: Context-Aware Auto-completion for SQL [VLDB 2011]
- **PerfXPlain**: Performance Debugging for MapReduce Jobs [VLDB 2012]

Acknowledgments

- SkewReduce is joint work with **YongChul Kwon** (UW), Bill Howe (UW), and Jerome Rolia (HP Labs)

- FTOpt is joint work with **Prasang Upadhyaya** and YongChul Kwon (UW)

SkewReduce Motivation

- Scientists need more than relational algebra
  - Complex analytics (e.g., data clustering)
  - Complex objects (e.g., points in 3D or 4D space)
- MapReduce is an attractive solution
  - Easy API, declarative layer, seamless scalability, …
  - User provides 2 functions: map and reduce
  - Map: Read input one record at a time and process
  - Reduce: Aggregate the output of Map

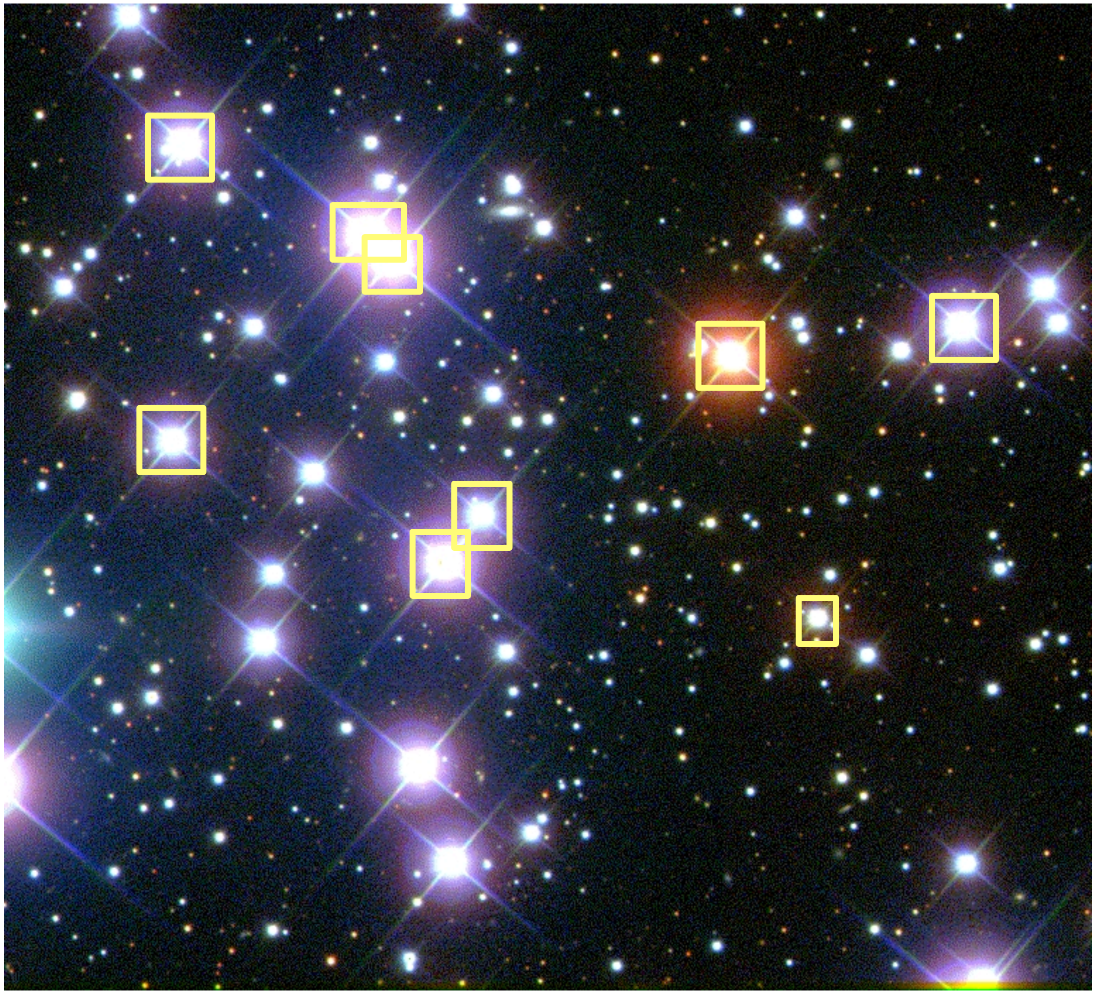
Motivation (continued)

- But it is hard to
  - Express complex algorithms and
  - Get high performance (e.g., 14 h vs. 1.5 h)

- SkewReduce:
  - Toward scalable ***feature extraction analysis***

Example 1: Extracting Celestial Objects

- Input
  - { (x,y,r,g,b,ir,uv,…) }
    - Coordinates
    - Light intensities
    - …
- Output
  - List of *celestial objects*
    - Star
    - Galaxy
    - Planet
    - Asteroid
    - …



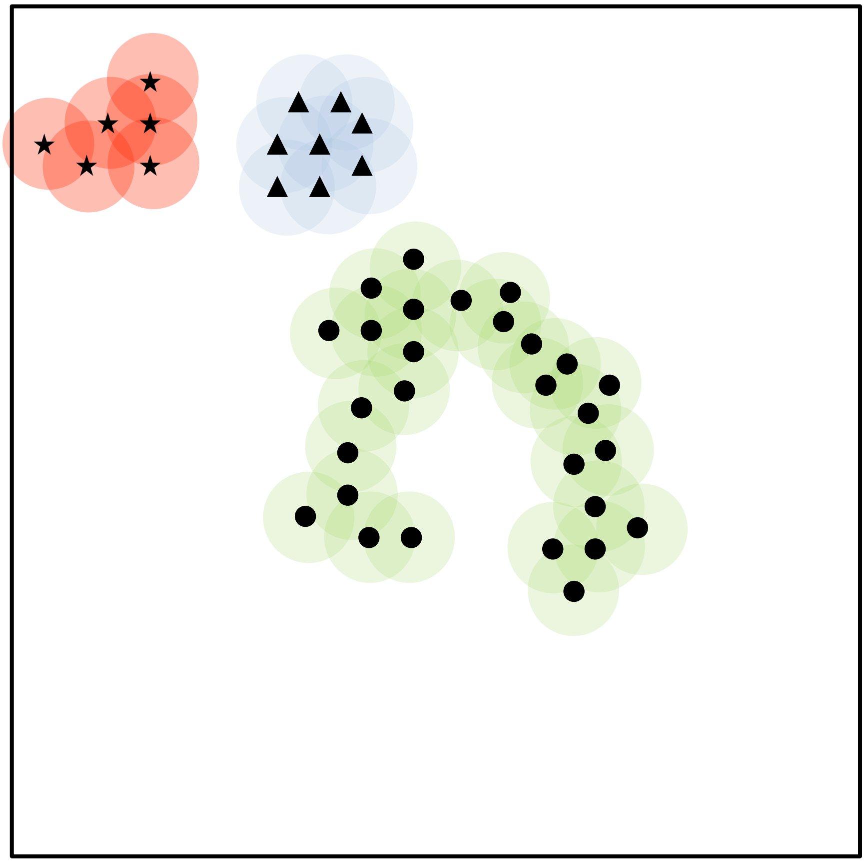
M34 from Sloan Digital Sky Survey

Example 2: Friends of Friends

- Simple clustering algorithm

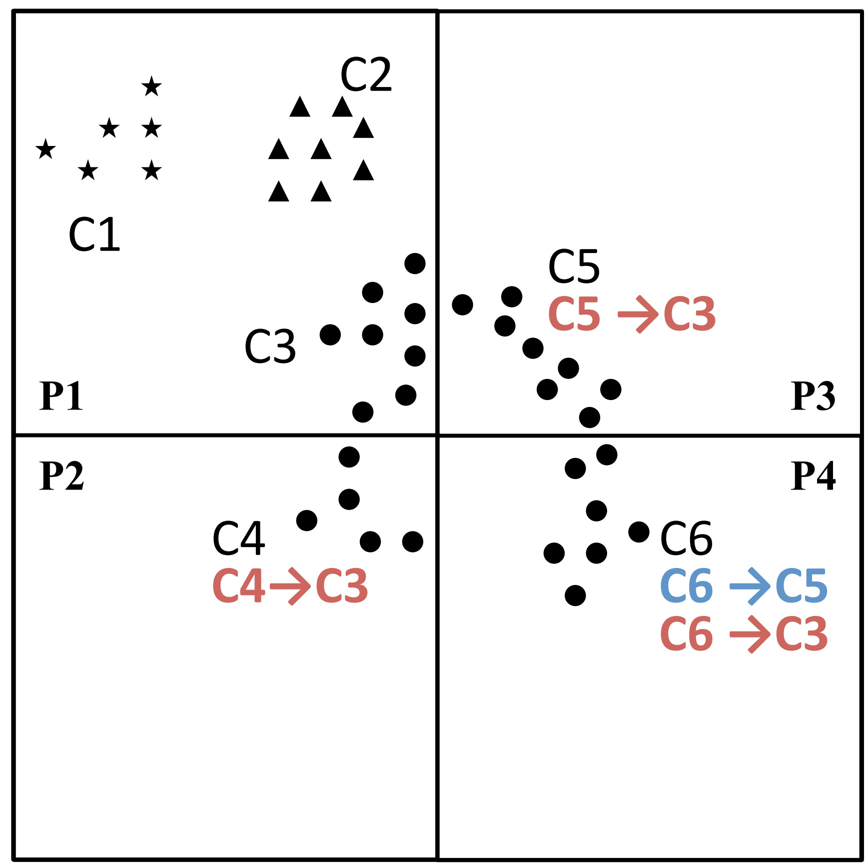
- Input:
  - Points in multi-dimensional space
- Output:
  - List of clusters
  - Original data annotated with cluster ID

- Friend
  - Point within a distance threshold
- Friends of Friends
  - Transitive closure of Friend relation

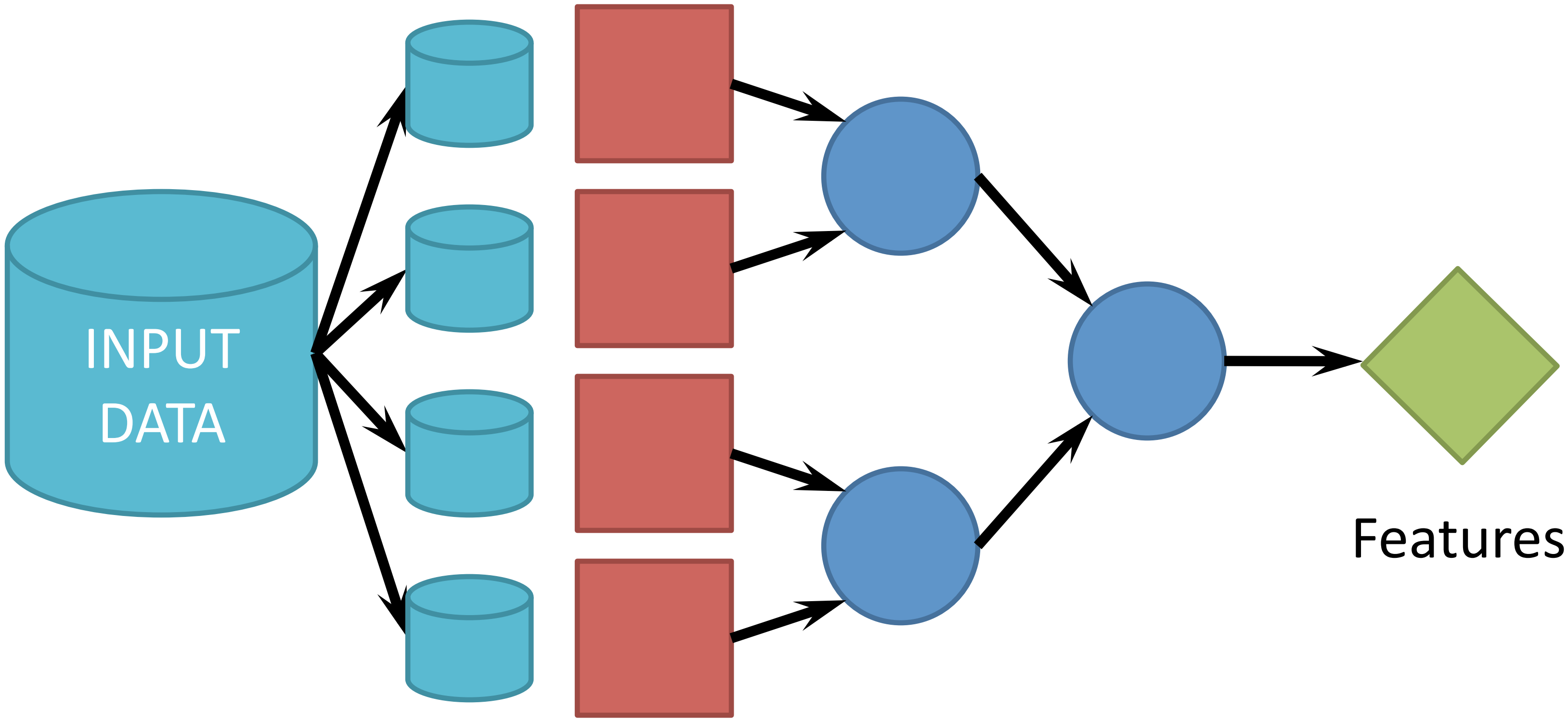


Parallel Friends of Friends

- Partition
- Local clustering
- Merge
  - P1-P2
  - P3-P4
- Merge
  - P1-P2-P3-P4
- Finalize
  - Annotate original data

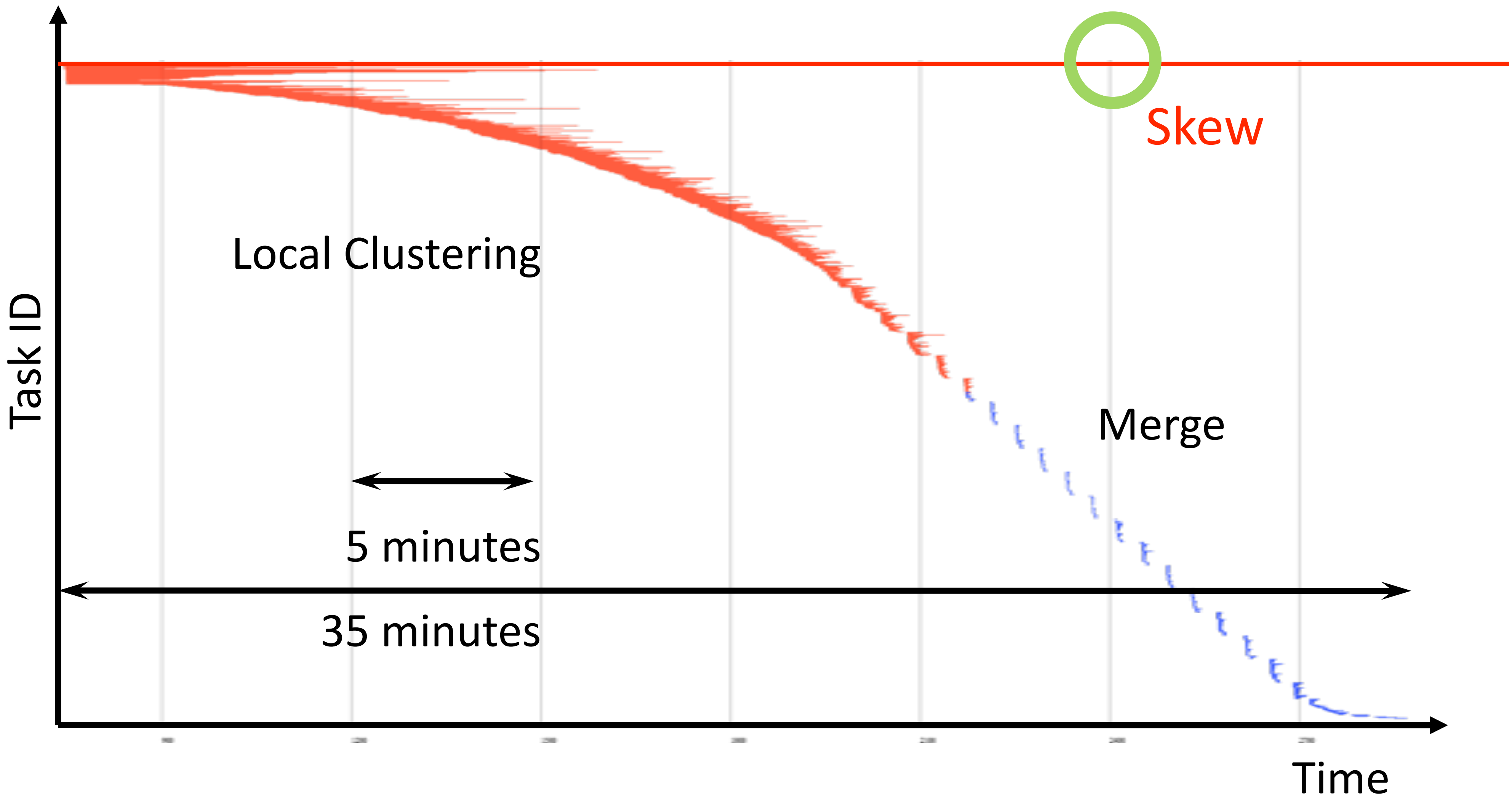


Parallel Feature Extraction



- *Partition* multi-dimensional input data
- *Extract* features from each partition — **Map**
- *Merge* (or reconcile) features — **Hierarchical Reduce**
- *Finalize* output

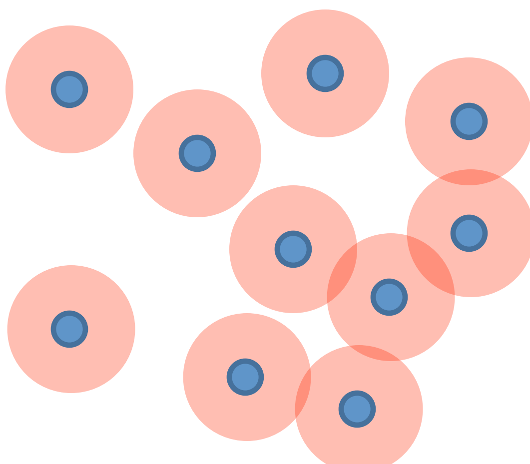
Problem: Skew



- The top red line runs for 1.5 hours

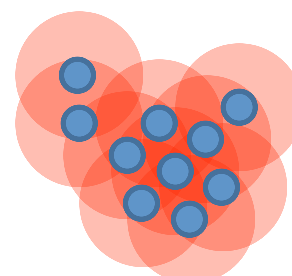
Unbalanced Computation: Skew

- Computation skew
  - Characteristics of algorithm
    - Same amount of input data != Same runtime



O(N log N)

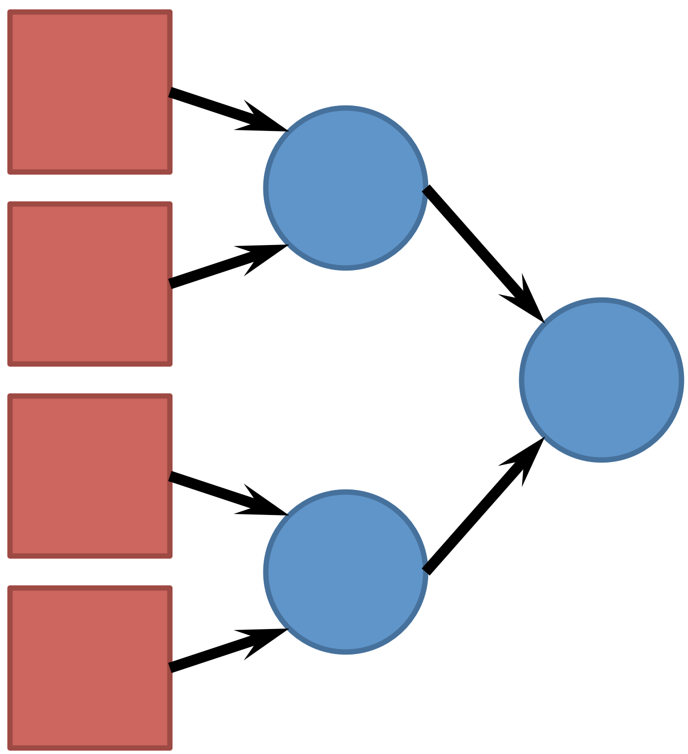
0 neighbors per particle



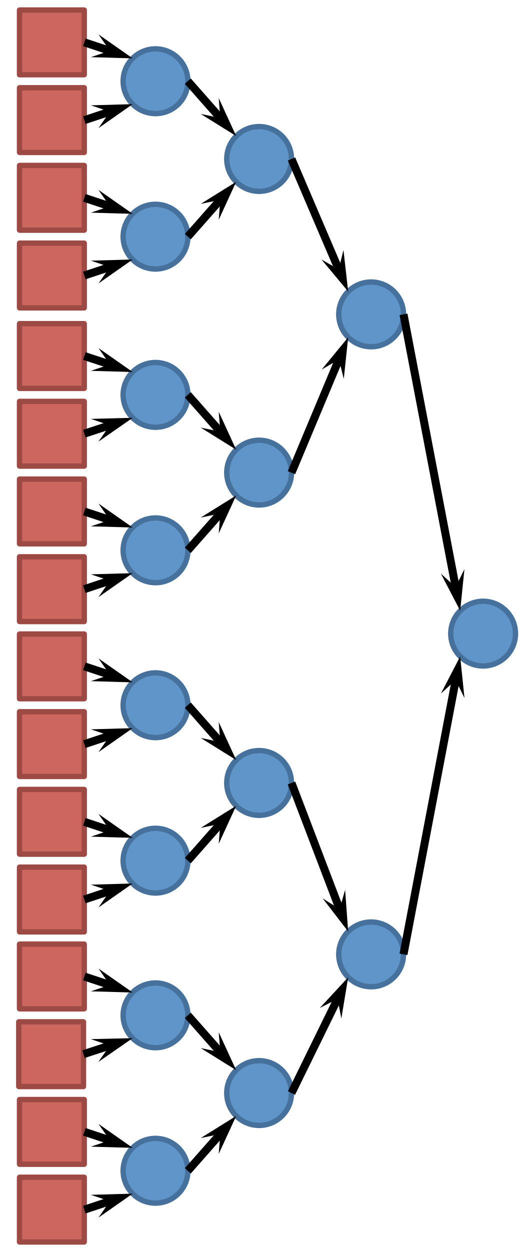
~ $O(N^2)$

~ N neighbors per particle

Solution 1?
Micro partitions



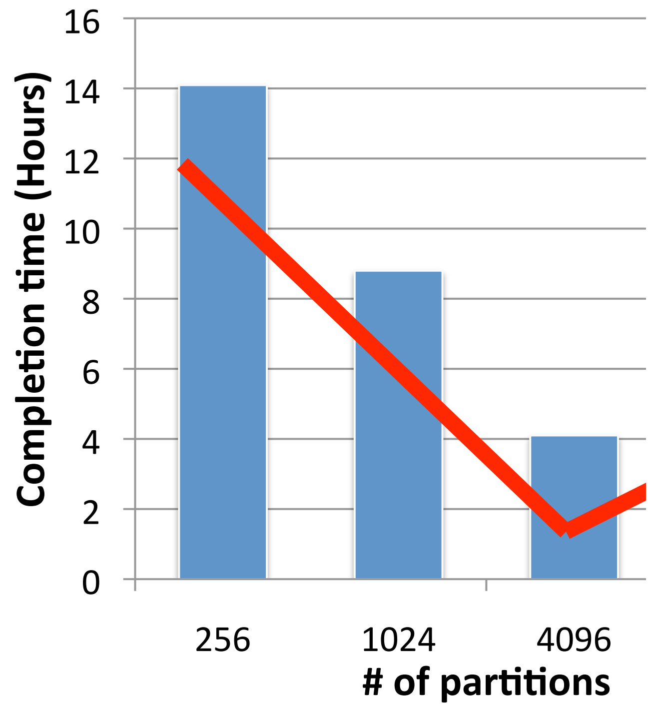
- Assign tiny amount of work to each task to reduce skew



How about having micro partitions?

- It works!
- But
- Overhead!

- To find sweet spot, need to try different granularities!

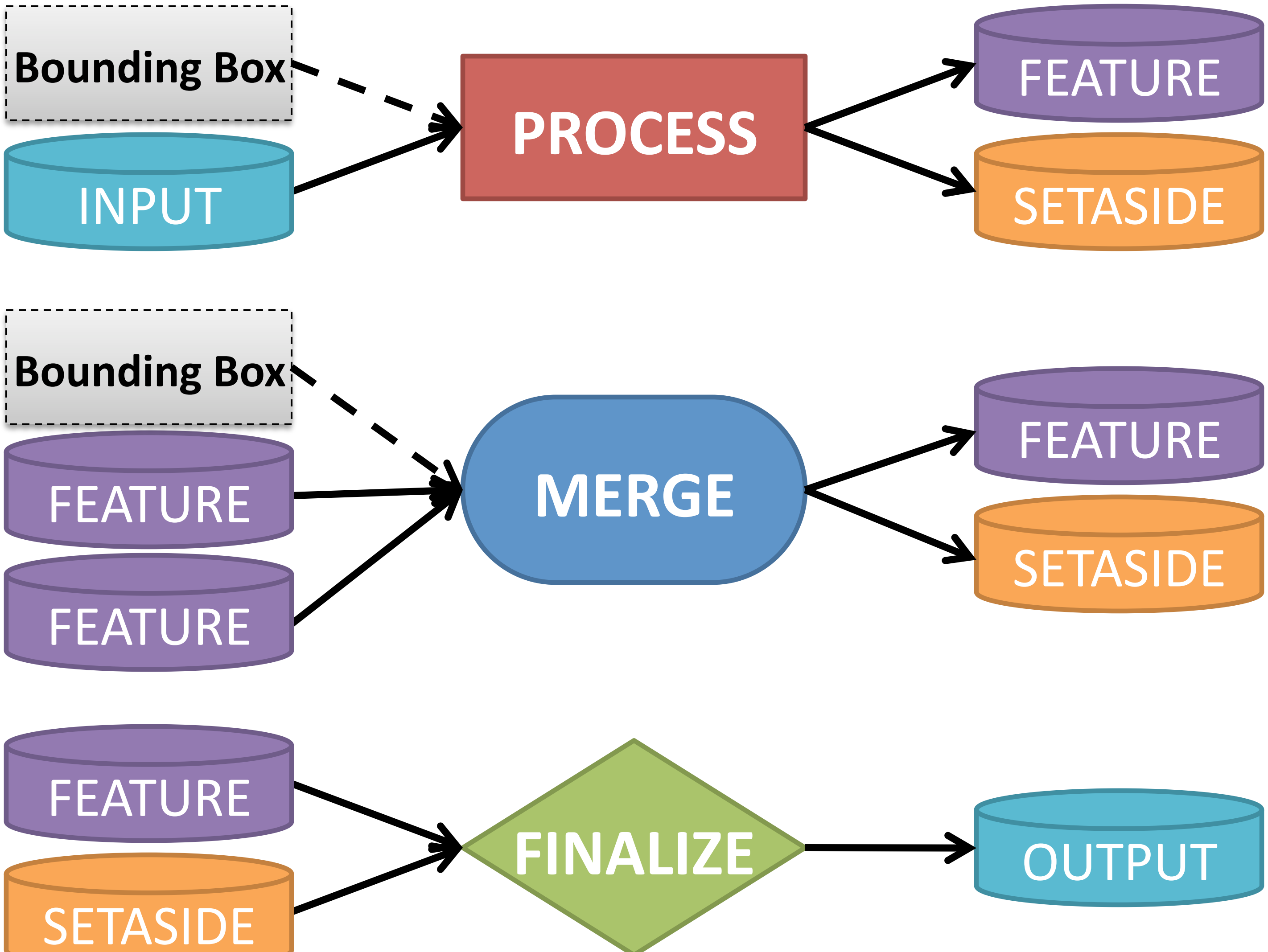


Can we find a good partitioning plan without trial and error?

SkewReduce

- An API for expressing feature-extracting applications
- A system built on top of Hadoop
  - Implements the API
  - Executes applications in a shared-nothing cluster
- An optimizer for automatically partitioning data
- Evaluation on astronomy and oceanography data

SkewReduce API



SkewReduce API

- Facilitates expression of feature extracting funcs
  - Input: set of points in a multidimensional space
  - Output: features and points labeled with their features

Input data

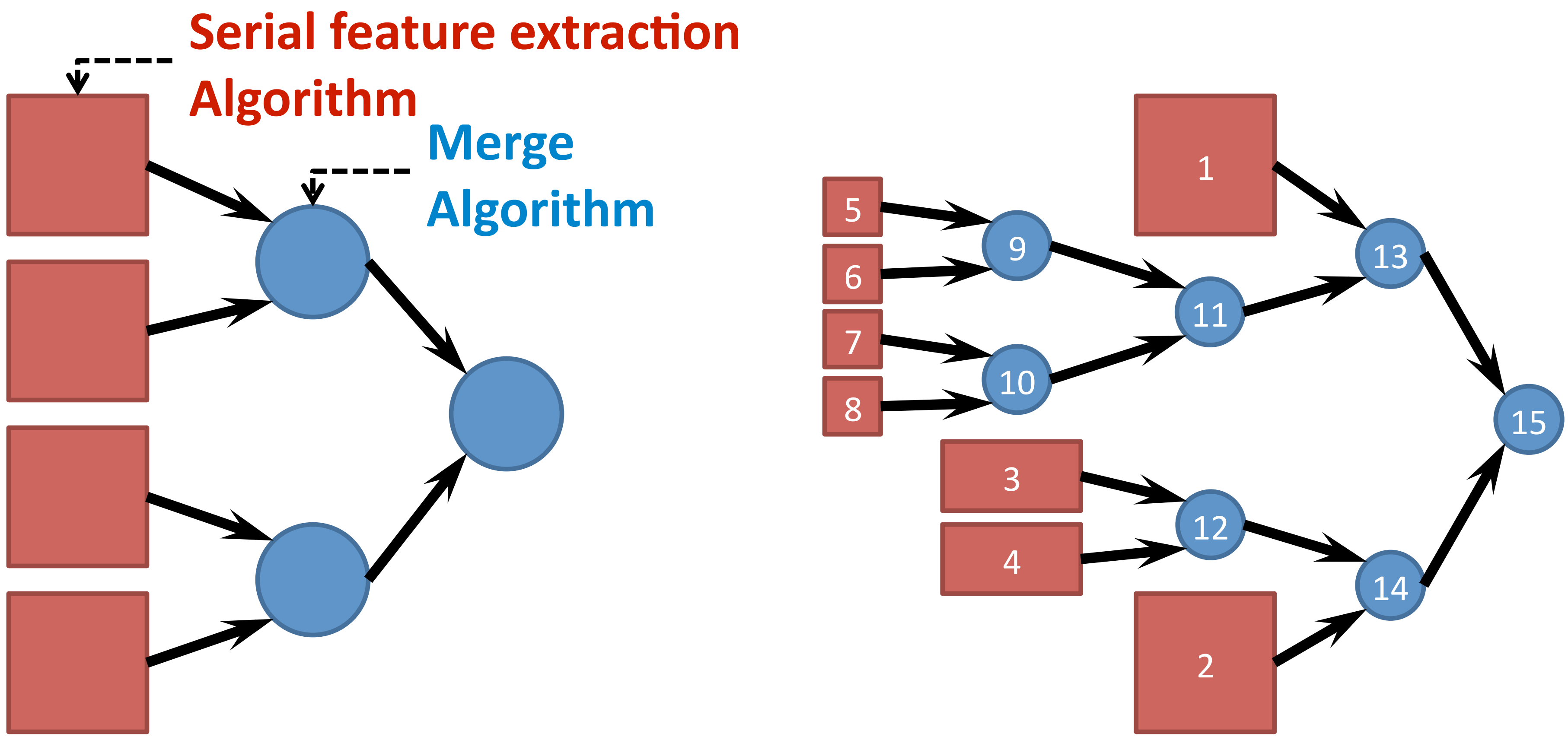
Features and extra info for merge

- Serial feature extraction function
  - process :: < Seq. of T> -> <F, Seq. of S>

Initial data labeled with features

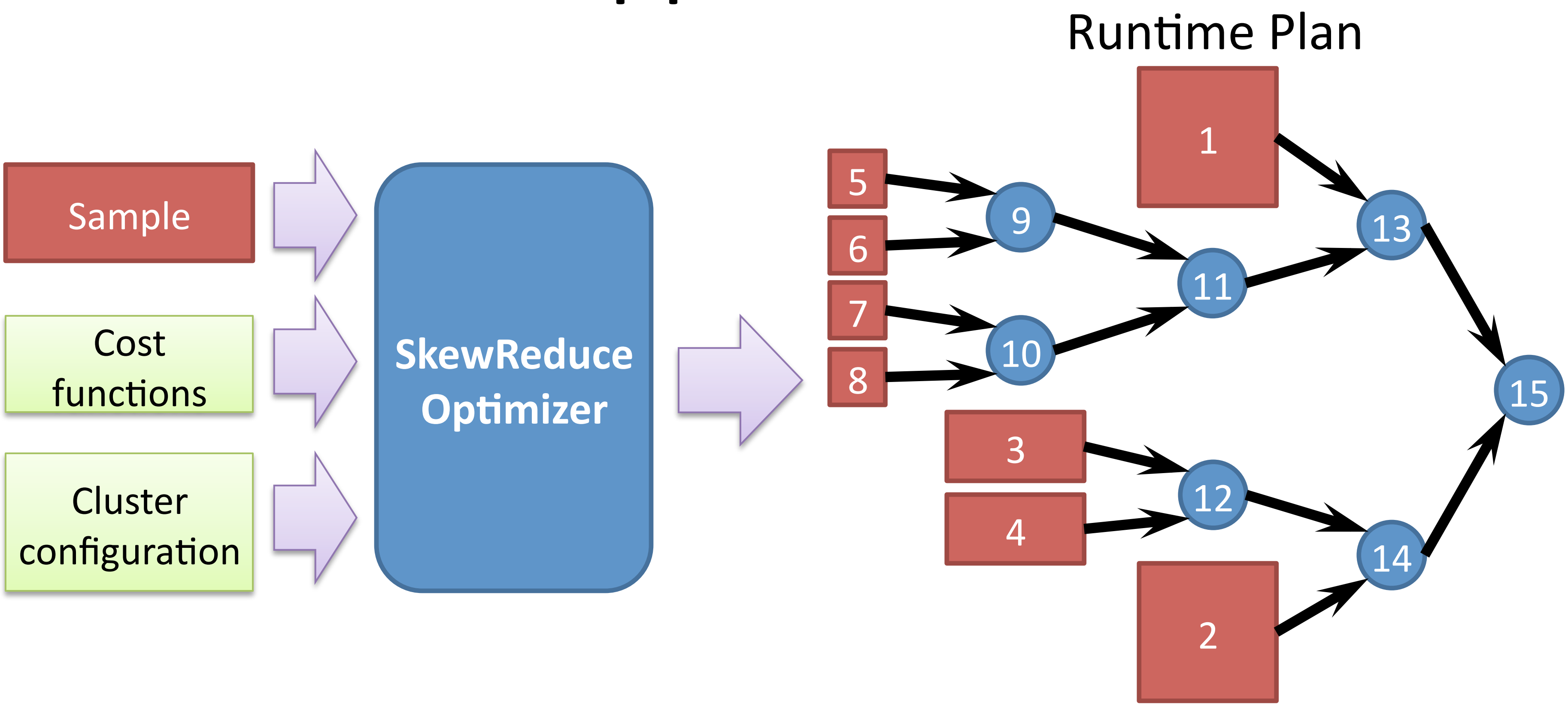
- Reconcile/merge extracted features
  - merge :: <F, F> -> <F; Seq. of S>
- Perform any final re-labeling as needed
  - finalize :: <F, S> -> <Seq. of Z>

Partition Optimization



- Two key algorithms: Extract features & Merge
- Can we *automatically* find *a good partition plan* and *schedule*?

Approach

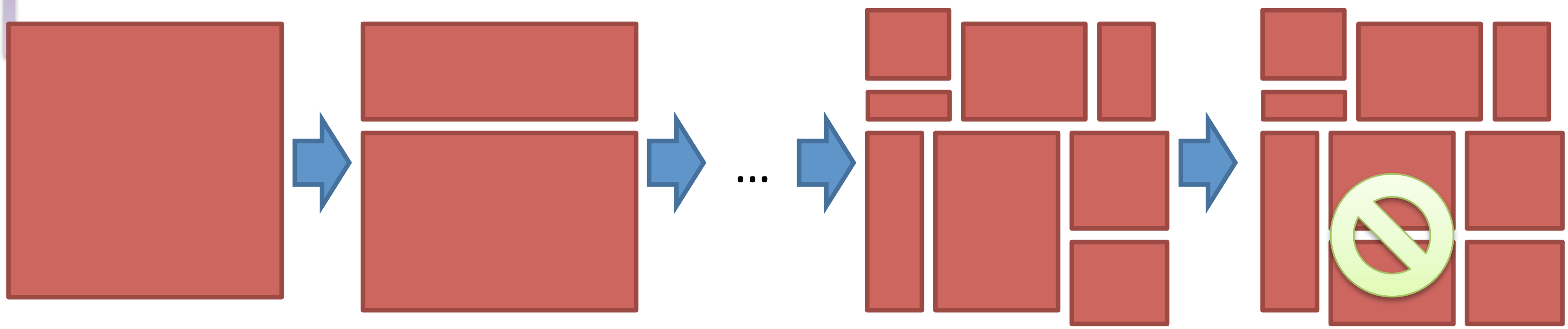


- ***Goal**: minimize expected total runtime*
- SkewReduce runtime plan
  - Bounding boxes for data partitions
  - Schedule

User-Provided Cost Functions

- Cost functions for feature extraction *and* merge:
  - CostProcess(Bounding box b, sample s, rate r) → cost
  - CostMerge(b1, s1, r1, b2, s2, r2) → cost
- Example cost function for FoF:
  - Build a 3D histogram of the sample data
  - Compute sum of squares of frequencies
- Should satisfy 2 properties: **fidelity** and **boundedness**
  - Fidelity: Lower cost means lower processing time
  - Boundedness: Can we scale costs into runtimes?

Partition Plan Guided By Cost Functions

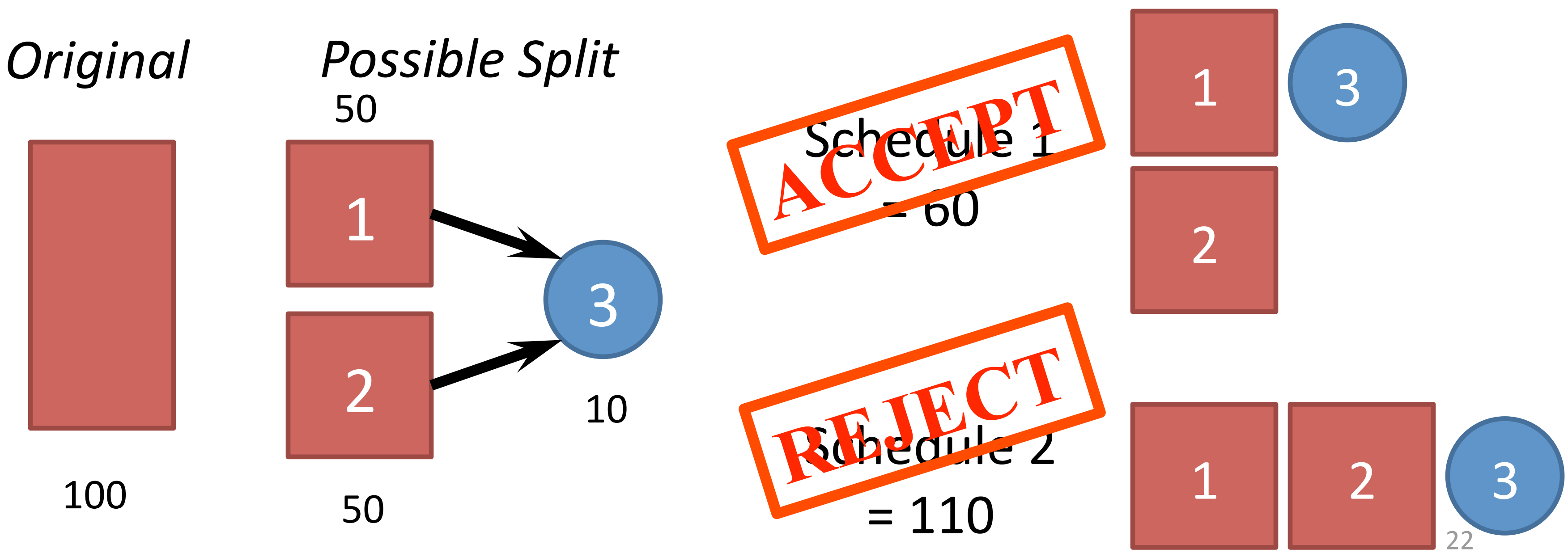


Cost functions serve to make two decisions

- **<u>How</u>** (axis, point) to split a partition
  - Ideally, want to split costs of partition in half
  - Approach: sampling, binary search, or incremental
- **<u>When</u>** to stop partitioning
  - Can the new set of partitions lead to a faster runtime?
  - *Must check actual expected schedule*

Search Partition Plan

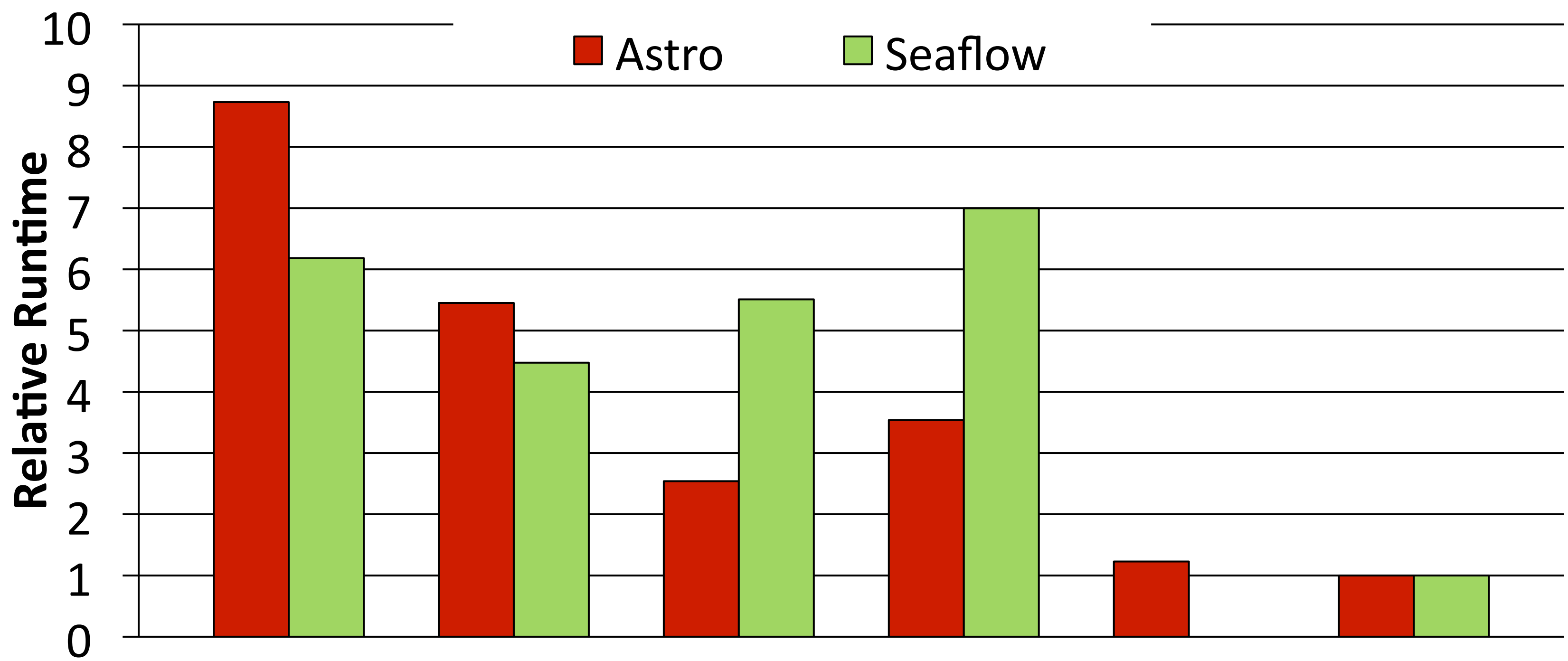
- Greedily split if total expected runtime improves
  - Search the best split (axis, point)
  - Evaluate costs for subpartitions and merge
  - Estimate new runtime



Evaluation

- 8 node cluster
  - Dual quad core, 16 GB RAM
  - Hadoop 0.20.1 + custom patch in MapReduce API
- Distributed Friends of Friends
  - Astro: Gravitational simulation snapshot
    - 900 M particles
  - Seaflow: flow cytometry survey
    - 59 M observations

Does SkewReduce work?



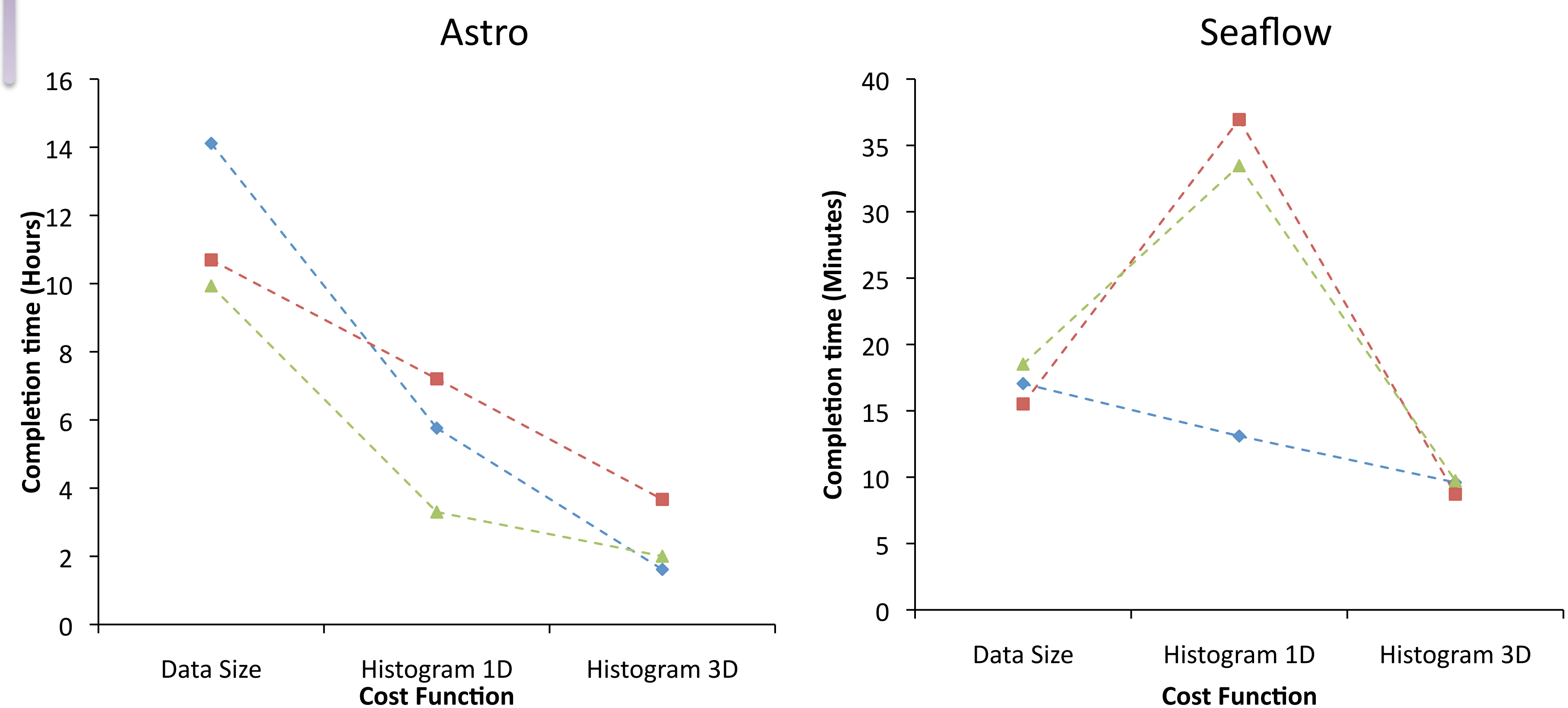
Coarse	Fine	Finer	Finest	Manual	SkewReduce	
14.1	8.8	4.1	5.7	2.0	1.6	Hours
87.2	63.1	77.7	98.7	-	14.1	Minutes

- Static plan yields 2 ~ 8 times faster running time

Cost Functions

- Data Size
  - the number of data items in a partition
- Histogram 3D
  - Model spatial index traversal pattern
  - Construct equi-width 3D histogram
  - Cost = sum of square of frequencies
- Histogram 1D
  - 1D version of Histogram 3D

Fidelity of Cost Functions



- Higher fidelity = Better performance
- Seaflow -- overestimation

SkewReduce Summary

- Scientific analysis should be *easy to write*, *scalable*, and with a *predictable performance*

- SkewReduce
  - API to faciliate expression of feature extracting funcs
  - Scalable execution
  - High-performance in spite of skew
    - Cost-based partition optimization using a data sample

Next Steps: SkewTune

- Key ideas:
  - Ask nothing from the developer
  - Make skew handling completely transparent
  - Mitigate skew at runtime
- Key approach:
  - As long as everyone is doing useful work, do nothing
  - If resources idle, re-partition longest task remaining only
  - Initial results: 4X time improvements!

Failures

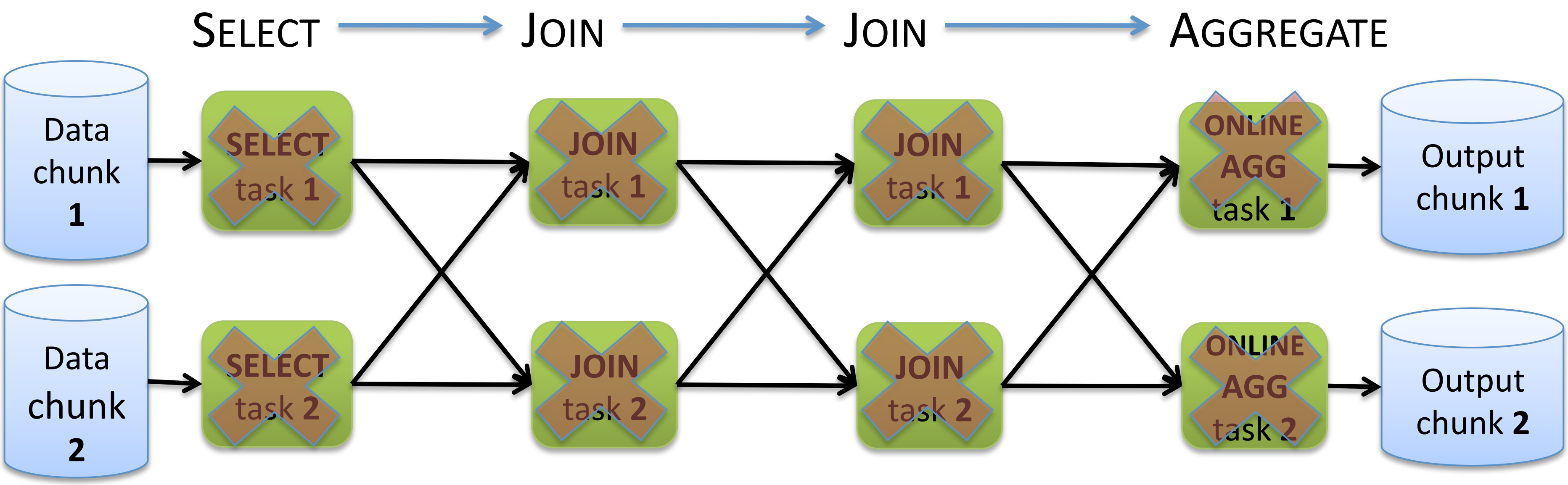
Failures in Big-Data Analysis

- At large scale, failures are the norm!
- Average of **5 worker deaths*** per **MapReduce job**

* Availability in Globally Distributed Storage Systems. OSDI 2010.

Fault Tolerance Approach 1

Streaming Query Execution **(Parallel Databases)**

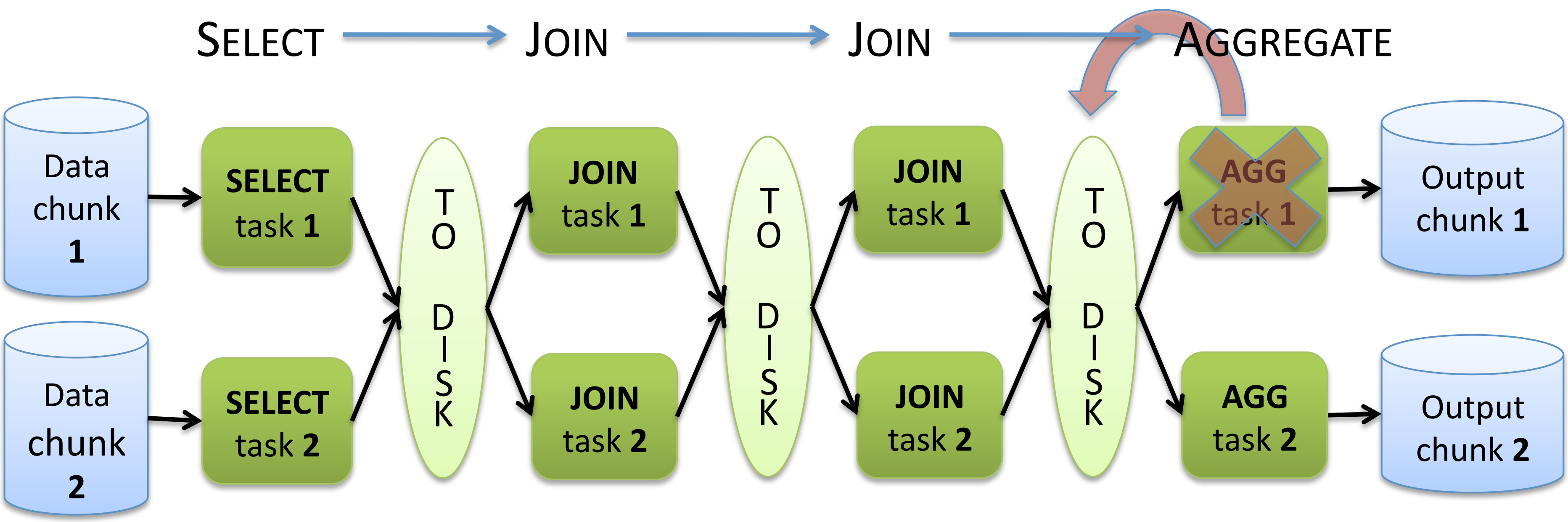


Incremental results possible with **"online"** operators

Failures are **costly**

Fault Tolerance Approach 2

Blocking Query Execution **(MapReduce)**



Inexpensive failure recovery

Blocking fault-tolerance prevents incremental results

Overhead of materialization

Bottom line

Desiderata:

1. Incremental output
2. Fast completion time **with failures**

- How can we achieve this?
  - **Use non-blocking** fault-tolerance techniques
  - **SKIP:** Skipping over un-needed inputs
  - **MATERIALIZE:** Non-blocking materialization
  - **CHECKPOINT:** Incremental checkpoints

- Each technique has a tradeoff. Which **ones** to use?

Tradeoff in Fault-tolerance Techniques

Skeleton Query Processing Engine

- Uses **TCP connection** to connect different operators
- Developed in **JAVA** using **Apache MINA**
- **Pluggable** fault-tolerance algorithms

17 machines. **8** cores of 2.5 GHz. **16 GB** RAM. **Two** 7.2K RPM

Exclusive access to cores and disks

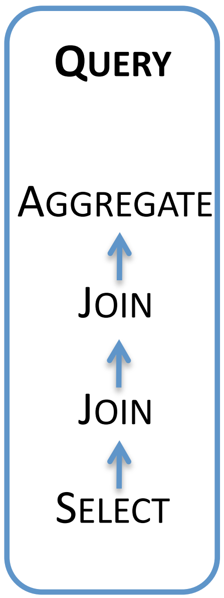
Each operator assigned **an equal number** of cores and disks

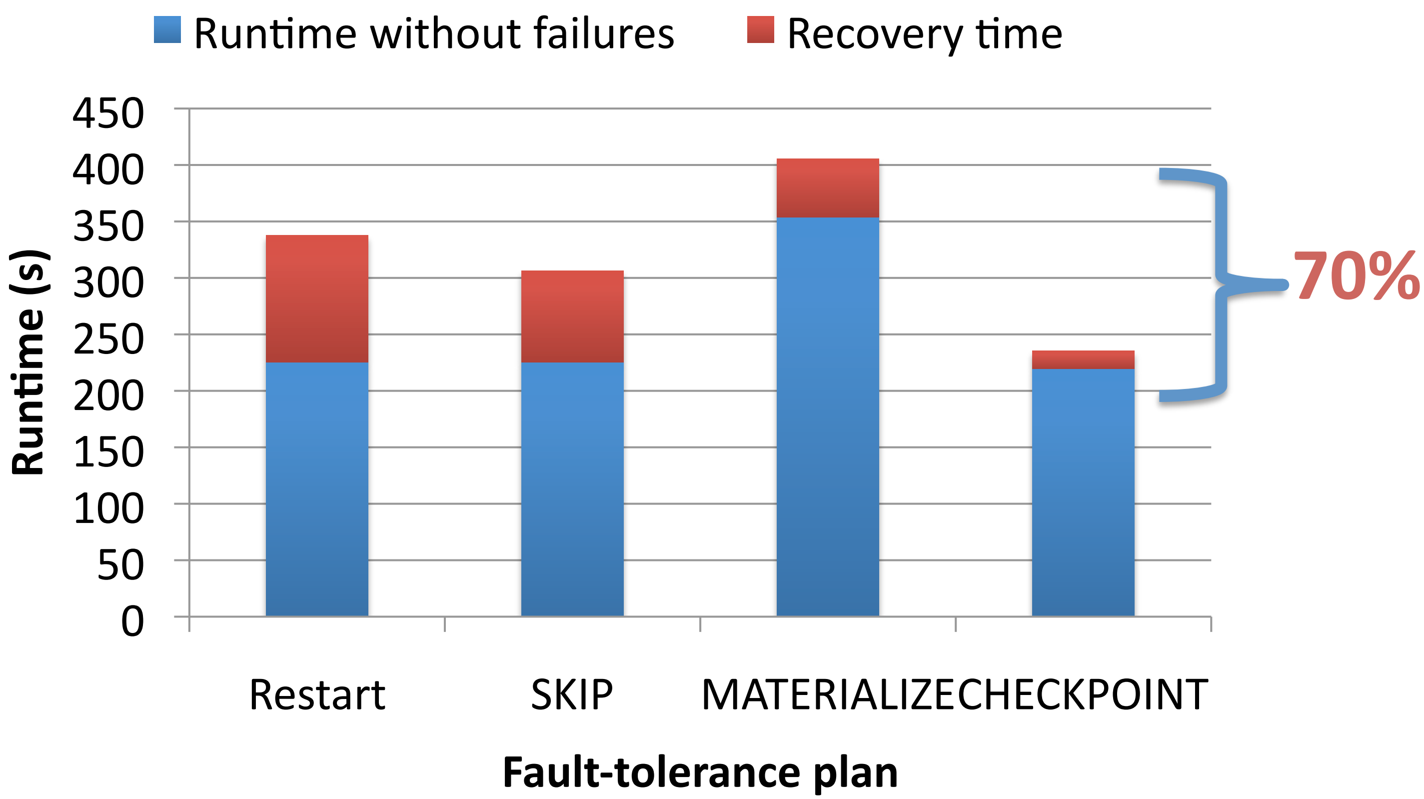
One failure in experiments

Half-way through the **normal runtime**.

For ***n*** operator query, execute ***n*** times, fail **a different** operator each time.

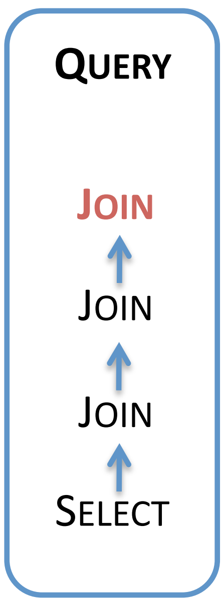
Fault-Tolerance Strategy Performance

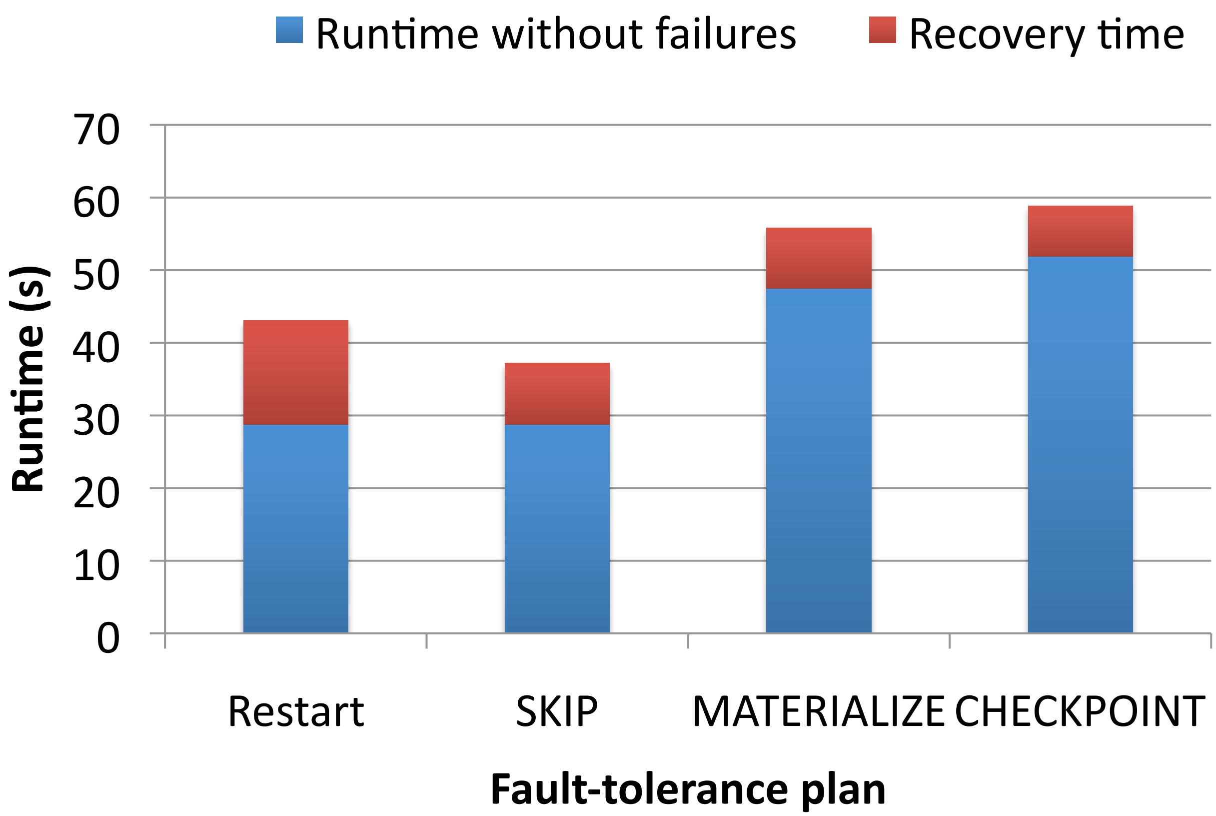




Does One Strategy Always Win?

Is the optimal strategy to checkpoint?





Takeaway

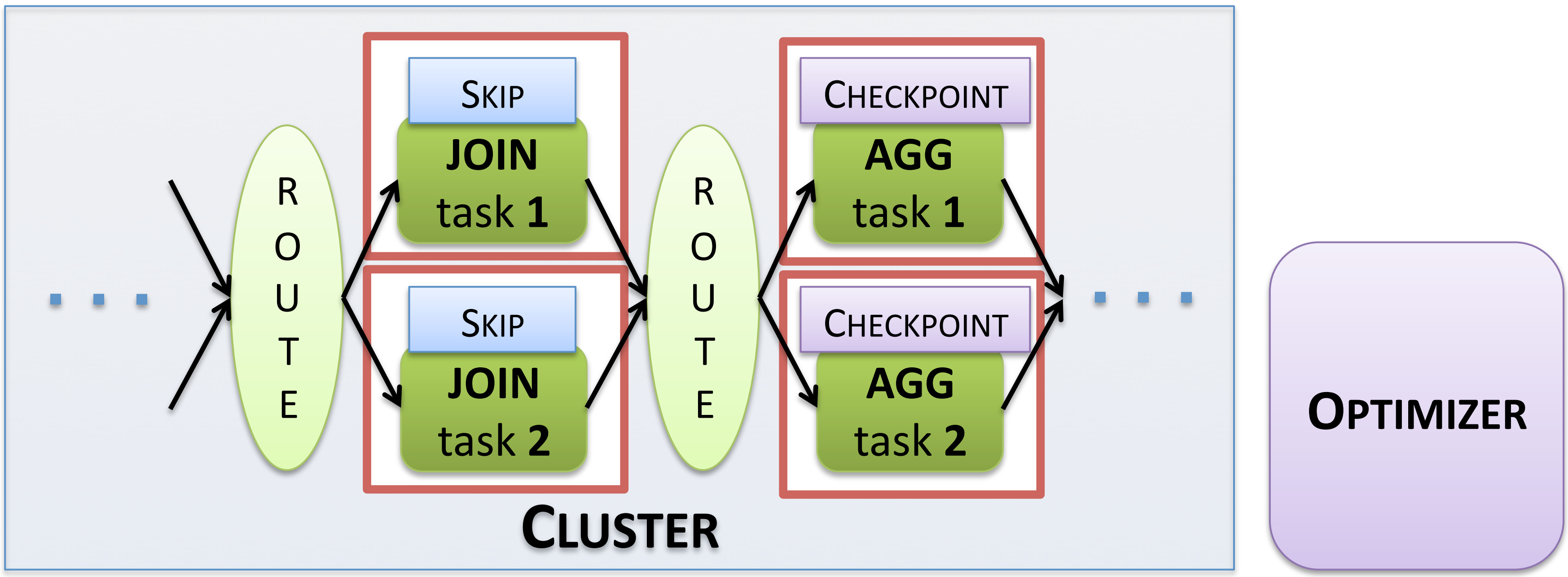
Is the optimal strategy to checkpoint?

Depends on the query!

To choose from **multiple fault-tolerance plans** is useful!

In fact, choose at the **granularity of operators**
and not **just queries**

FTOpt Components



FTOPT: approach to enable **heterogeneous fault-tolerance** and **automatic strategy selection**

Interaction Protocol
Offline Optimizer

Novel contributions

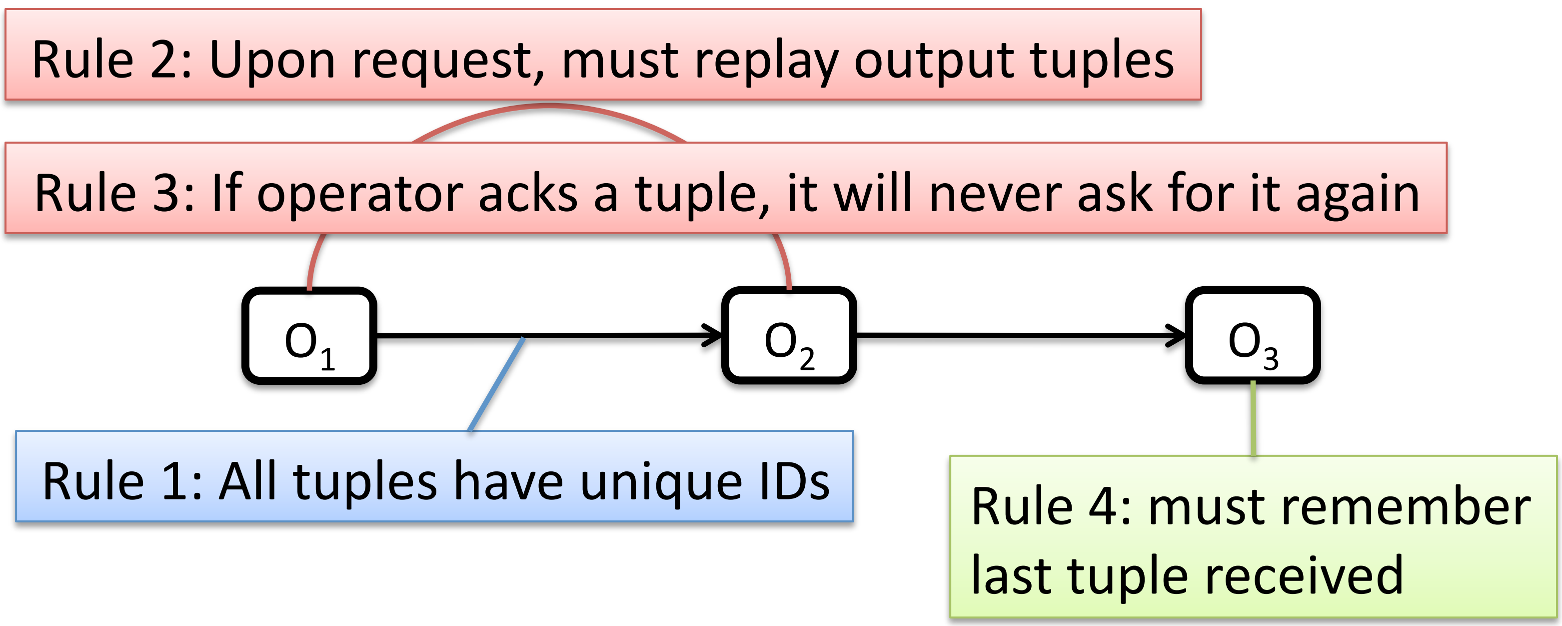
Interaction Protocol

Aim: Make **pipeline-of-operators** fault-tolerant
using operator-level fault-tolerance

Solution: Abstract the **local** fault-tolerance **properties**
required for **global** correctness

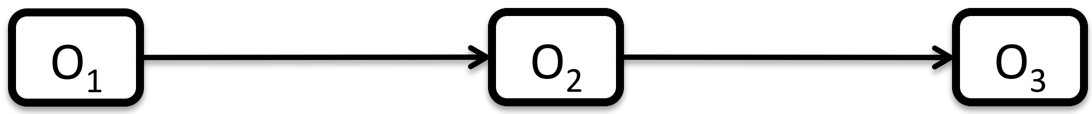
Heterogeneous Fault-Tolerance Protocol

- Want each operator to pick preferred FT strategy
- Simple protocol based on ideas from homogeneous FT:



- Operators are now individually recoverable
- Data can keep flowing without interruption

How Failure Recovery Works?



Step 1: Operator O_2 crashes and restarts

Step 2: O_2 asks O_3 for last tuple it received — Rule 4, Rule 1

Step 3: Optionally, O_2 recovers any saved state from stable storage

Step 4: O_2 asks O_1 to replay any needed data — Rule 1, Rules2 & 3

Step 5: O_2 sends only new data to O_3

Optimization Program

Objective is the **expected total runtime:**

T = **Blocking Delay + Processing time**

+ **E[# of failures * recovery time]**

Challenge: Need **accurate** estimates of **execution times** under **normal operation** and during **recovery**

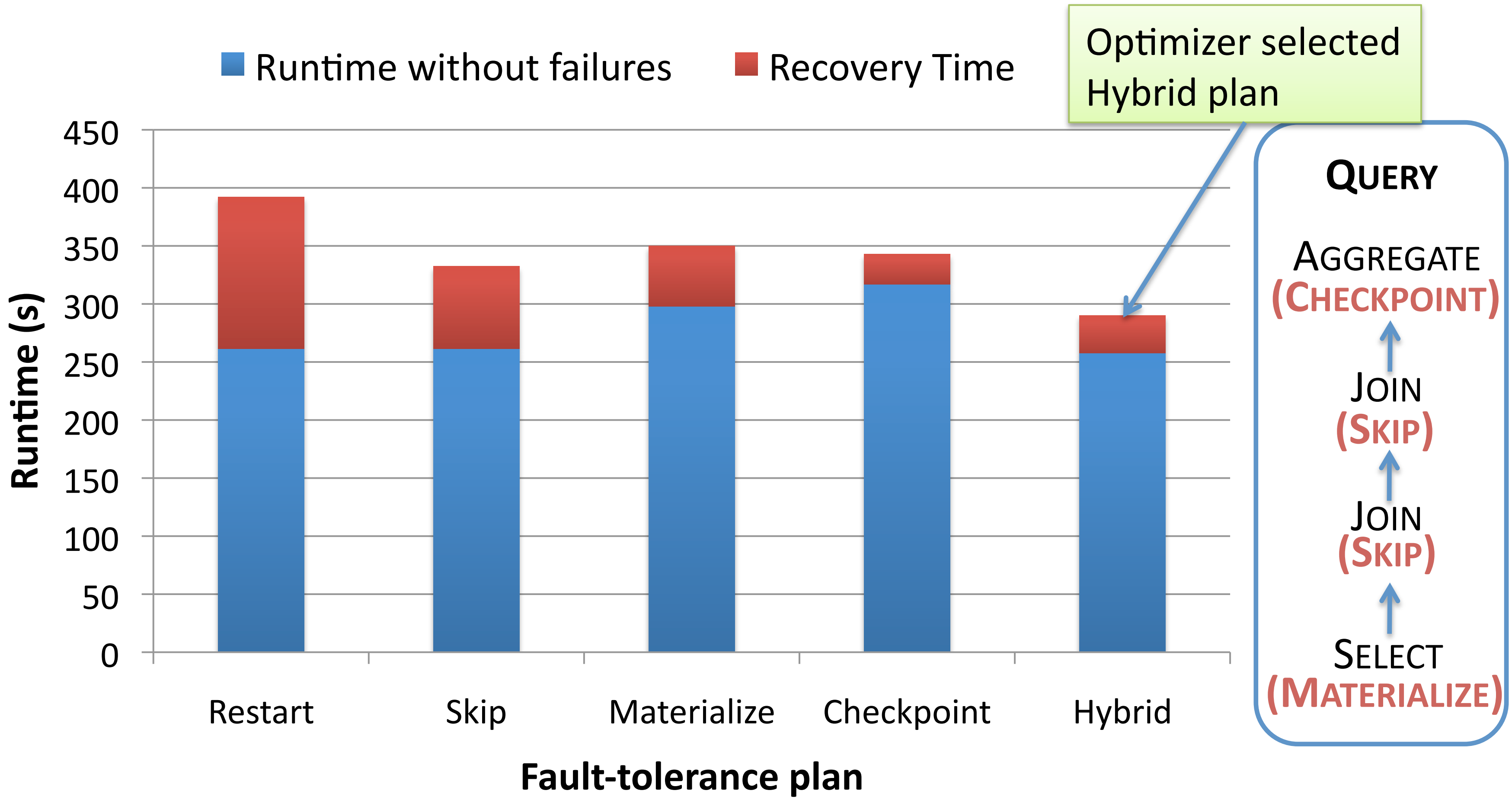
Why is this hard?

Operator behavior can be **non-linear**

Fault-tolerance can be **non-linear**

Solution: model operators using **convex constraints**

Heterogeneous Fault-Tolerance



Hybrid plan is **21% better** than any uniform plan and **33% better** than restart

FTOpt Summary

FTOpt

- **Protocol:** Support fault-tolerance **mix-and-match**
- **Optimizer:** Cost-based optimization

Runtime differences of **up to 70%**!

Extra details in the paper:

- **Operator models** within convex constraints framework
- **User-Defined Operators** (UDOs)
- Protocol implementation and **extensibility**
- Additional experiments: **sensitivity**, impact of **UDOs**, etc.

Conclusion

- Big-data analytics plays important role today
  - In science
  - In industry
- Many challenges to big-data analytics
  - Today we discussed skew and failures
  - Nuage project strives for high-performance analytics
  - CQMS project studies usability aspects
- Please visit our websites for more information!

http://nuage.cs.washington.edu/ and http://cqms.cs.washington.edu/