Large-scale Incremental Data Processing with Change Propagation

Pramod Bhatotia

Alexander Wieder, Istemi Ekin Akkus, Rodrigo Rodrigues, Umut A. Acar

MPI-SWS, Germany

USENIX HotCloud 2011

Large-scale Data Processing

- Need to repeatedly process evolving data-sets
  - For Web search PageRank is re-computed for every crawl
- Online data-sets evolve slowly
  - Successive Yahoo! Web crawls change by 0.1% to 10%
- Need for incremental computations
  - Instead of re-computing from scratch

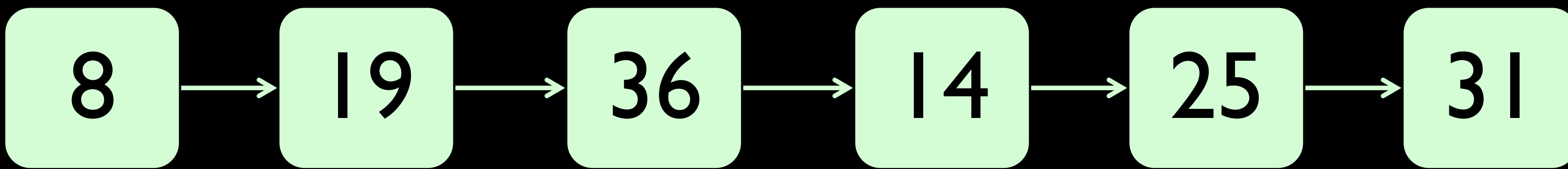
Incremental Data Processing

- Systems for incremental processing
  - Google Percolator [OSDI'10]
  - Yahoo! CBP [SoCC'10]
- Drawbacks of these systems
  - Adopt a new programming model
  - Require implementation of dynamic algorithms

Incremental Data Processing

- Systems for incremental processing
  - Google Percolator [OSDI'10]
  - Yahoo! CBP [SoCC'10]
- Drawbacks of these systems
  - Adopt a new programming model
  - Require implementation of dynamic algorithms

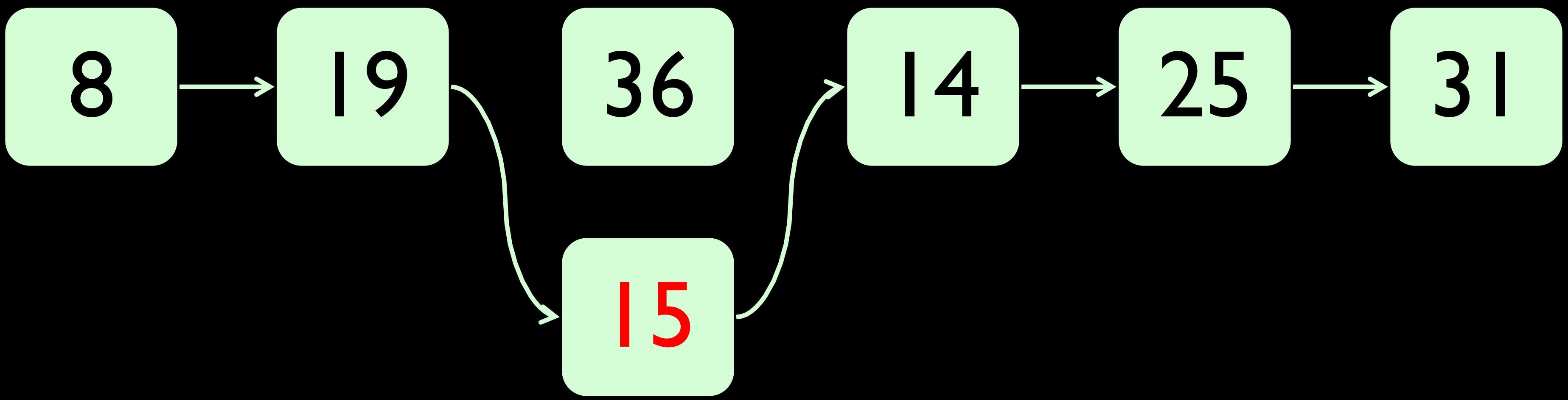
Example of a Static Algorithm



Compute the maximum element in a list

Scan the list and compute max in $O(n)$

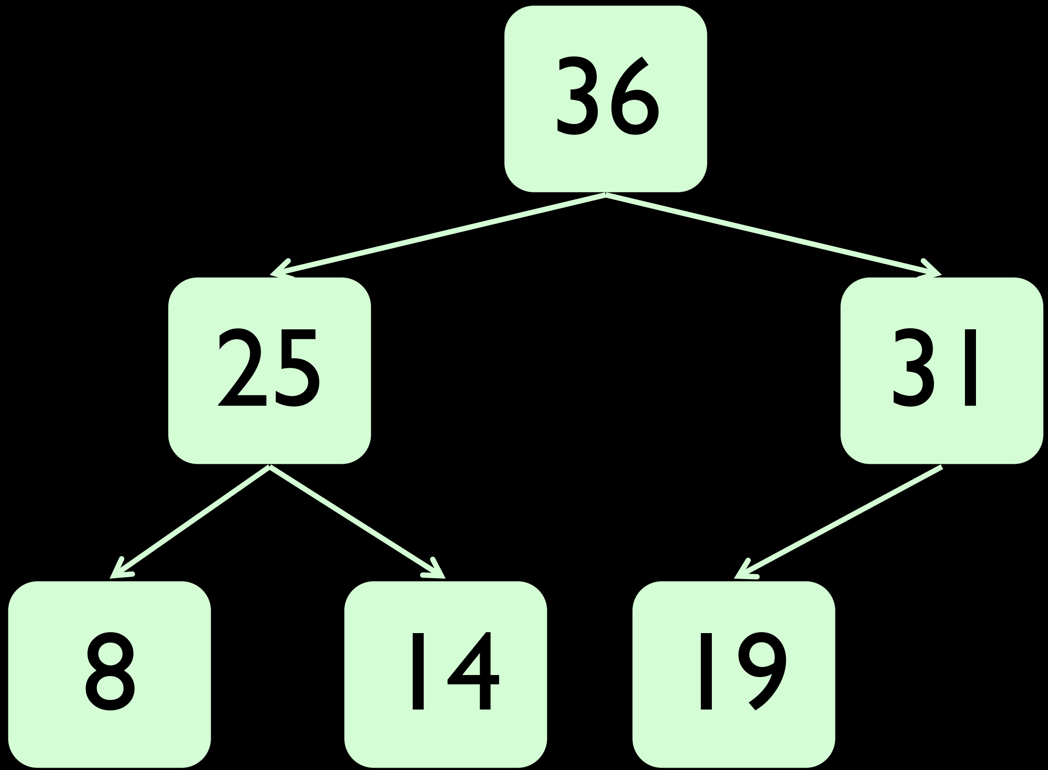
Static Algorithm with Input Change



Modify the input and find the max

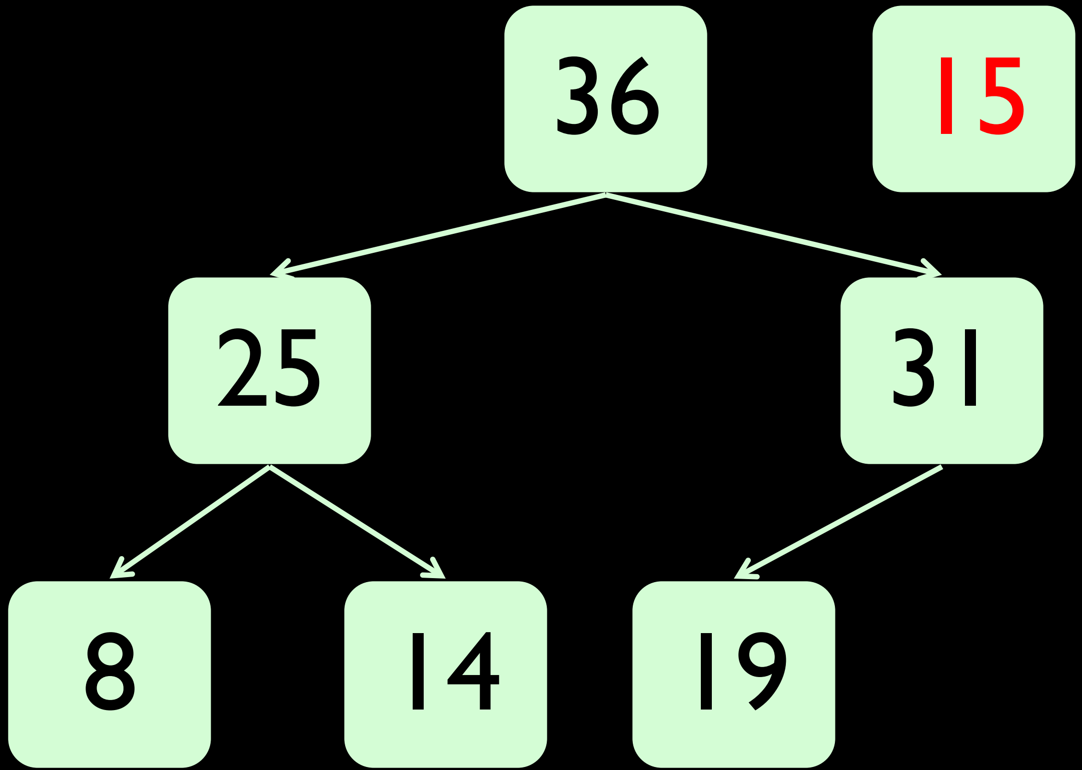
Static algorithms re-computes from scratch: O(n)

Example of a Dynamic Algorithm



maintain maximum heap

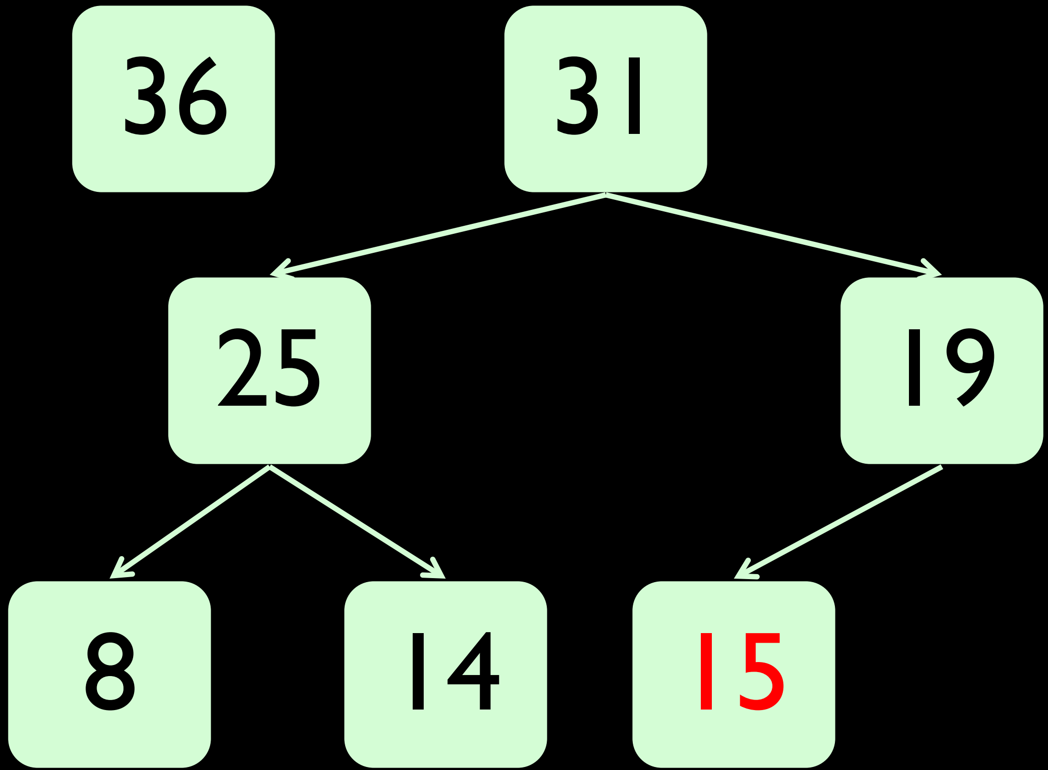
Example of a Dynamic Algorithm



Incremental updates in O(logn)

Asymptotically faster than the static algorithm

Example of a Dynamic Algorithm



Incremental updates in O(logn)

Asymptotically faster than the static algorithm

Static vs Dynamic

Algorithm	Simplicity	Efficiency
Linked list (Static)	**Easy**	**O(n)**
Heap (Dynamic)	**Hard**	**O(log n)**

Goals

- Retain the simplicity of static algorithms
- Achieve the efficiency of dynamic algorithms

Can we meet these goals in distributed systems?

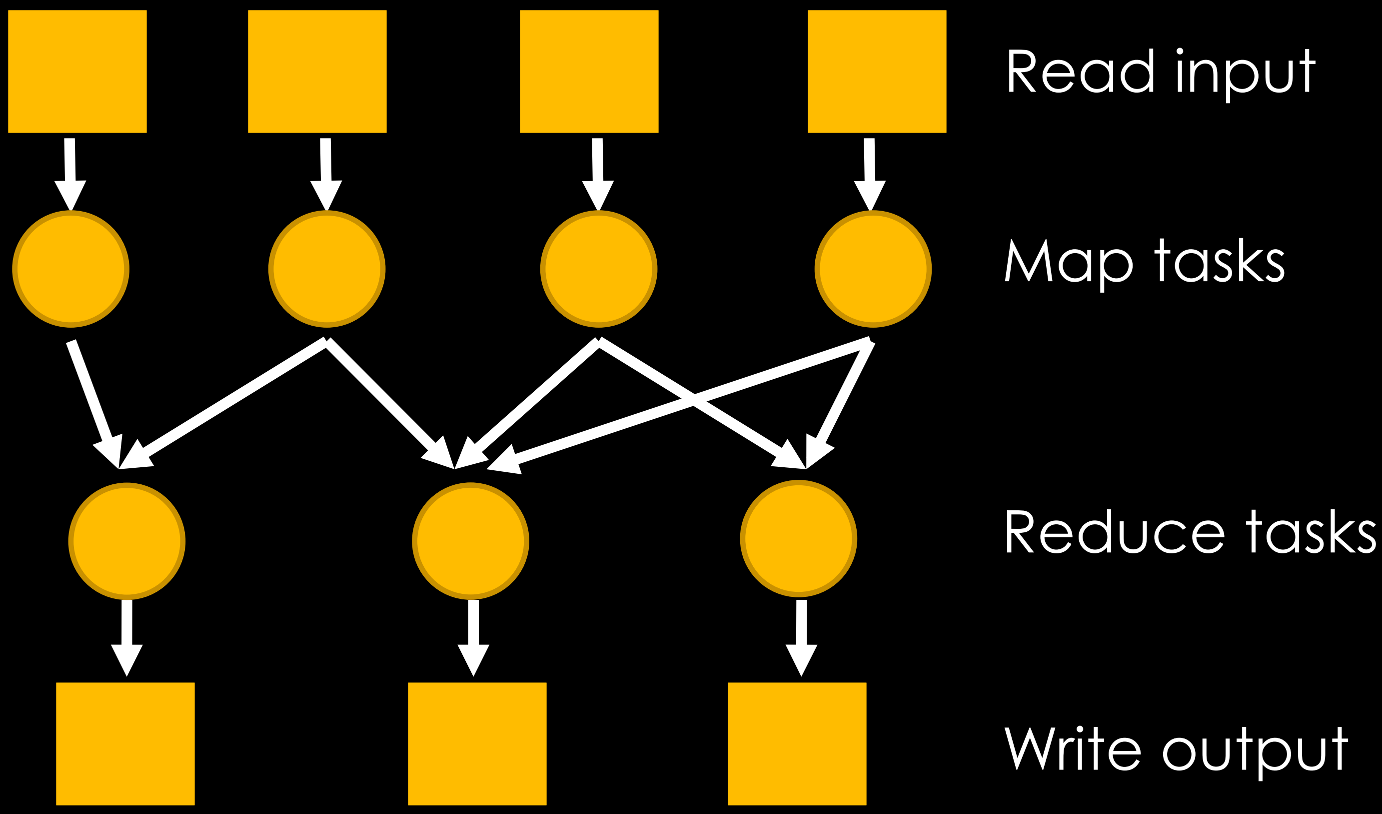
This talk : MapReduce

Our Approach

- Take an unmodified MapReduce program
- Automatically make it incremental

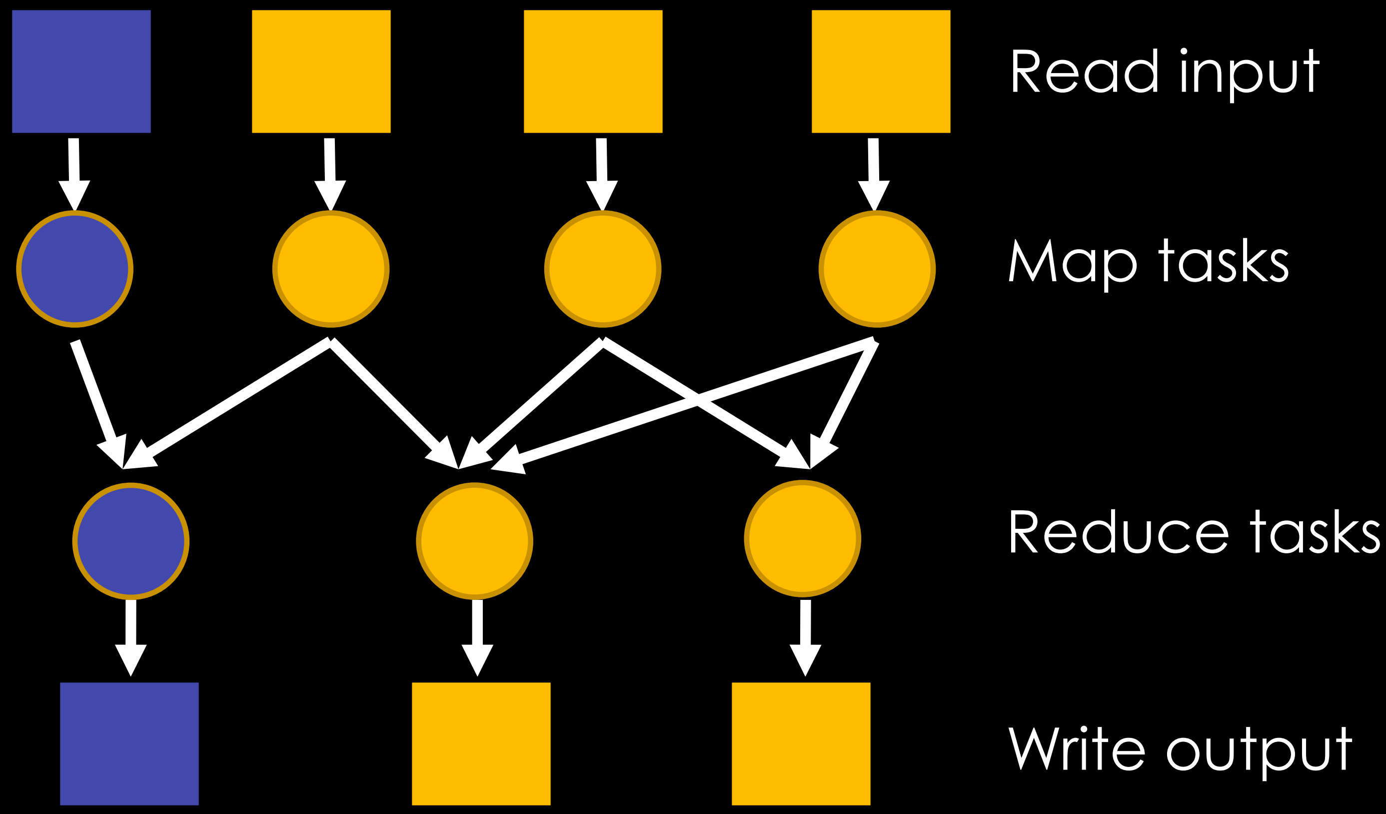
- Basic principle: Self-adjusting computations
  - Break computation into sub-computations
  - Memoize the results of sub-computations
  - Track dependencies between input and computation
  - Re-compute only the parts affected by changes

MapReduce with Change Propagation



Changes propagate through dependence graph

MapReduce with Change Propagation



Changes propagate through dependence graph

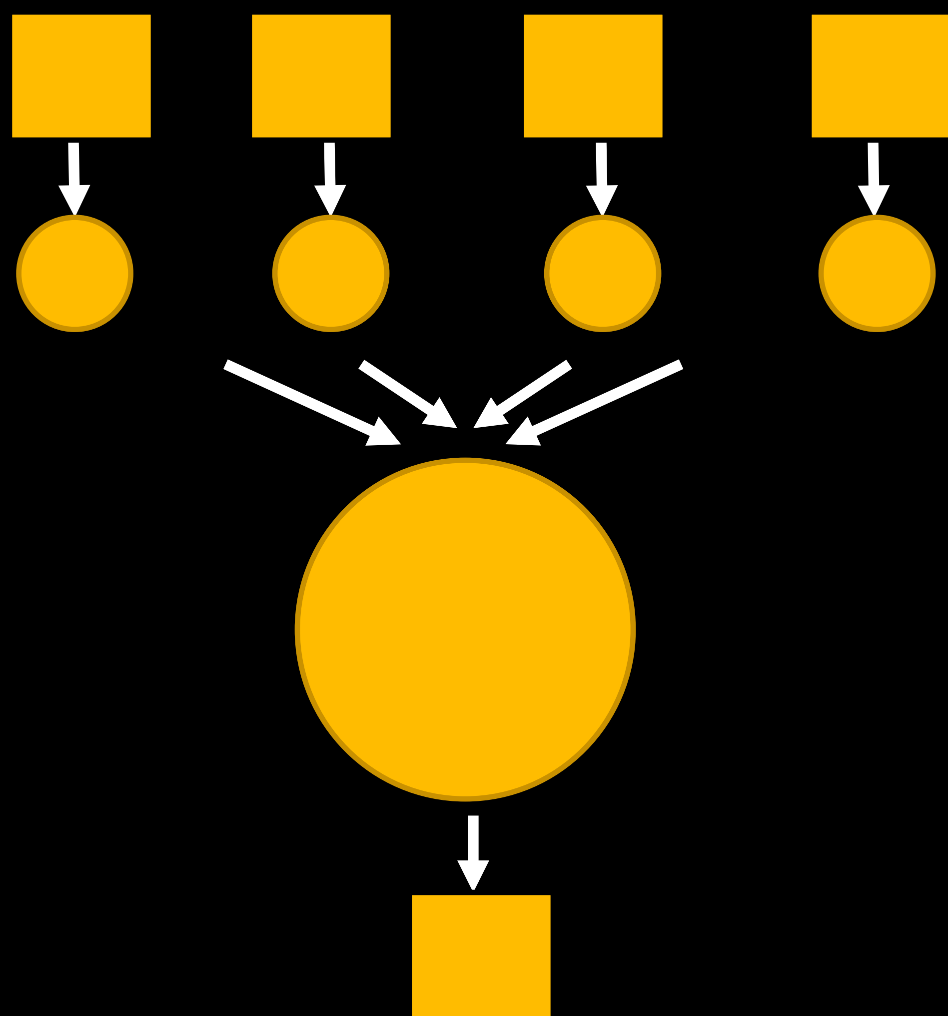
Challenges

- How to efficiently detect insertion/deletion ?
- How to minimize data movement ?
- How to perform fine-grained updates ?

Challenges

- How to efficiently detect insertion/deletion ?
- How to minimize data movement ?
- How to perform fine-grained updates ?

How to control granularity of Reduce ?



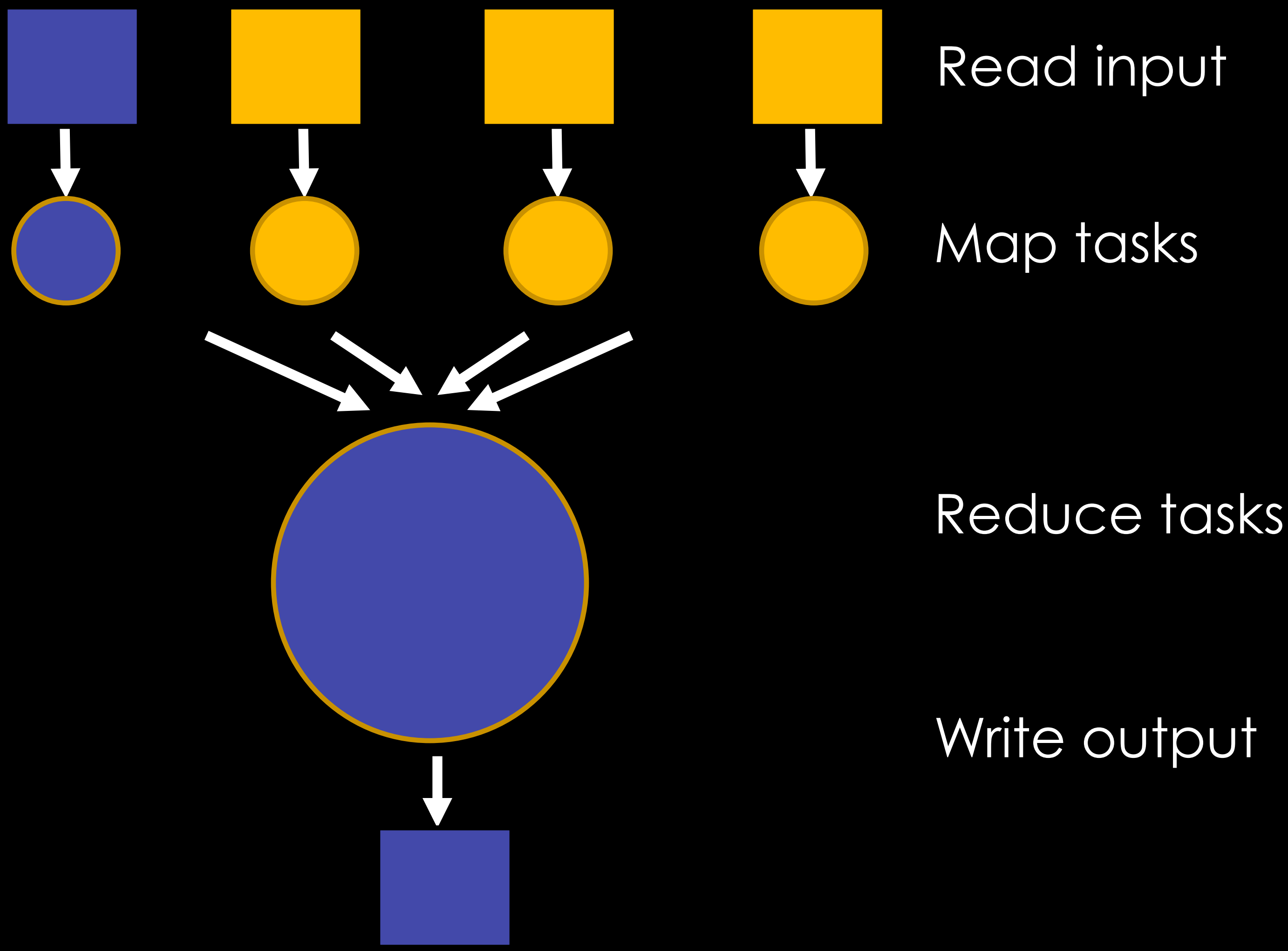
Read input

Map tasks

Reduce tasks

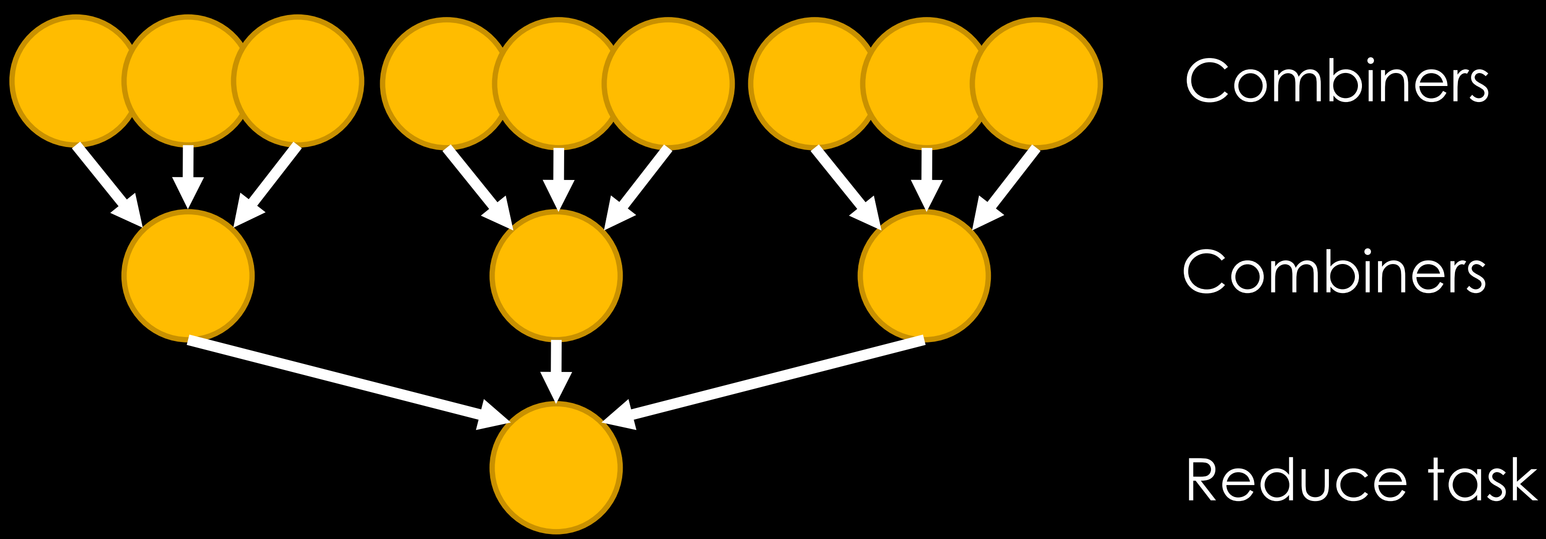
Write output

How to control granularity of Reduce ?

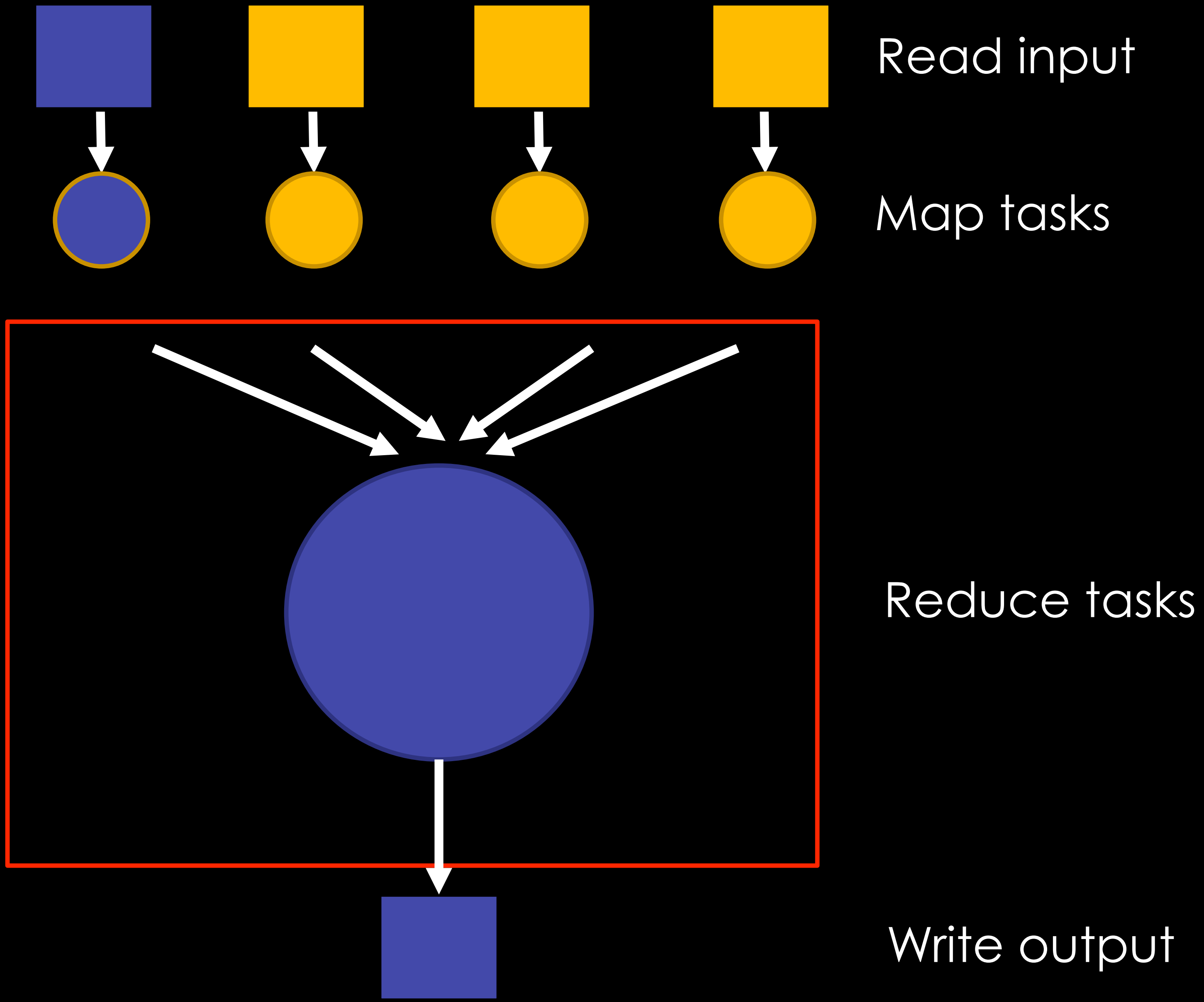


Controlling Reduce Granularity

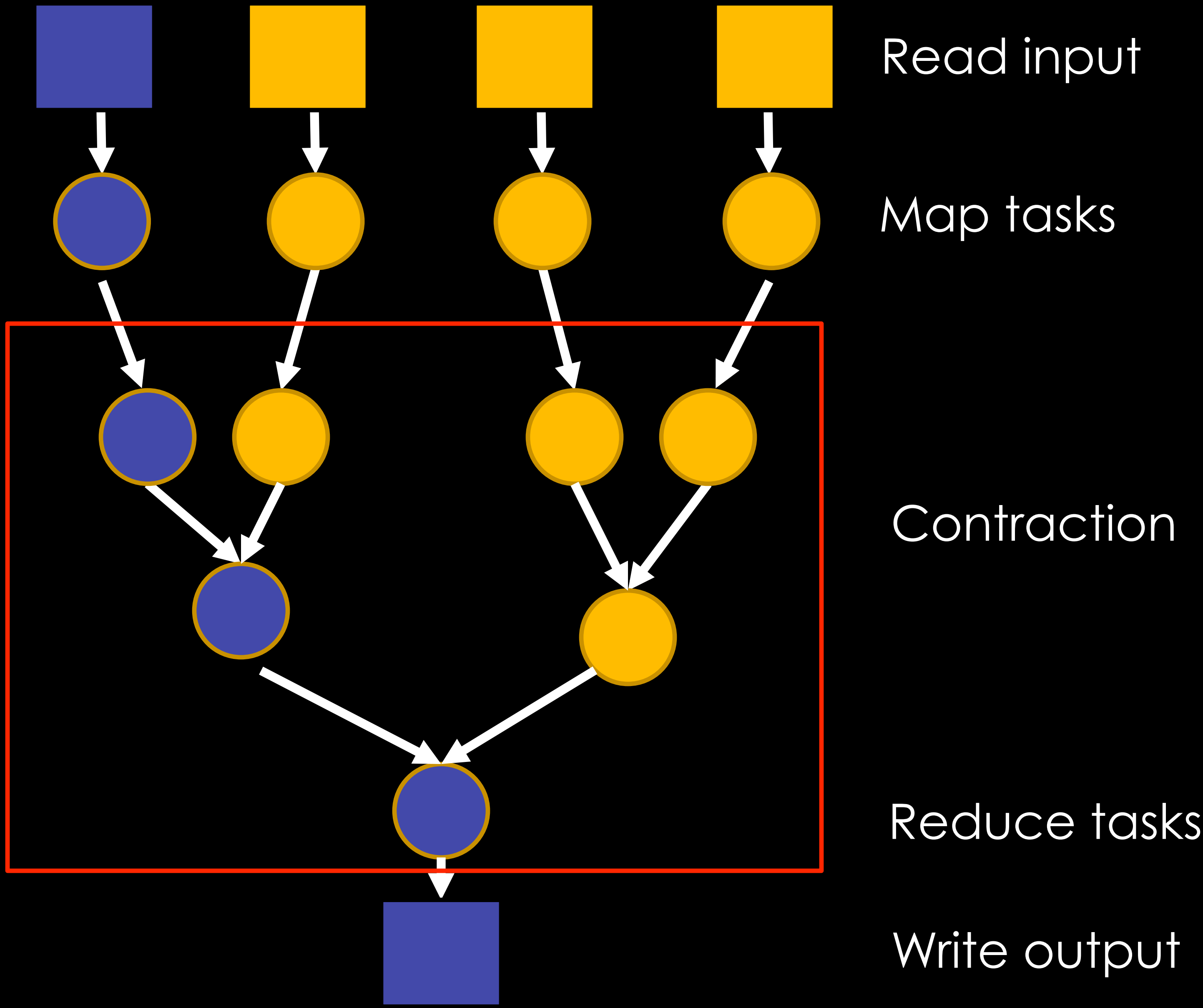
- Leverage Combiners: pre-processing of Reduce
  - Co-located with Map task for local reduction
- Use them to break up Reduce work



Contraction Phase: Tree of Combiners



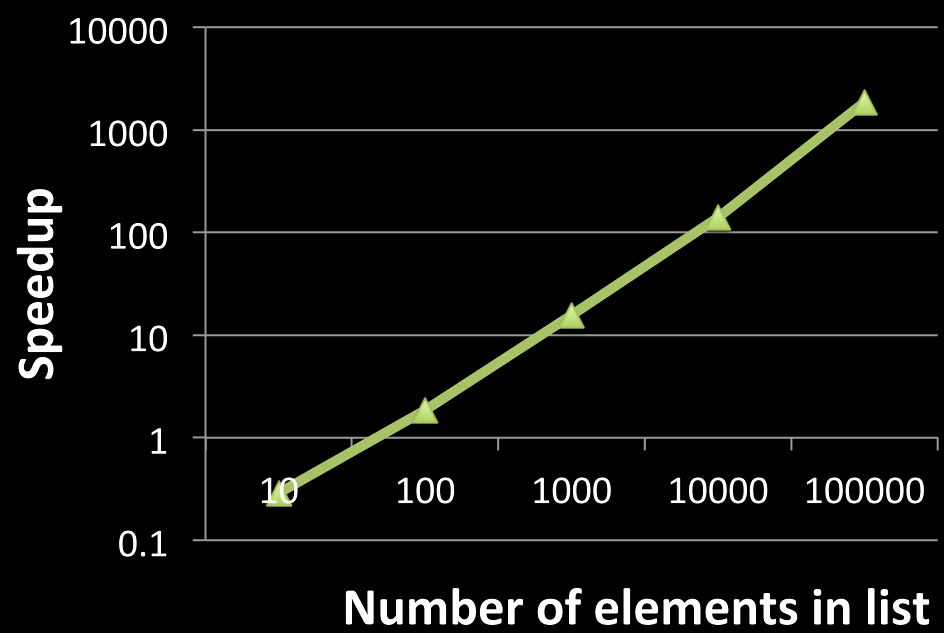
Contraction Phase: Tree of Combiners



Evaluation: Proof-of-concept

- Single-node MapReduce with change propagation
- Computing maximum for a list with single modification

$$\text{SpeedUp} = \frac{\text{Run-time for computing from scratch}}{\text{Run-time for incremental computation}}$$



Asymptotic gains with increase in size of data-set

Summary

Goals:

- Retain the simplicity of static algorithms
- Achieve the efficiency of dynamic algorithms

This talk:

- How to achieve these goals in MapReduce

Future:

- Apply principles to broad class of data processing systems