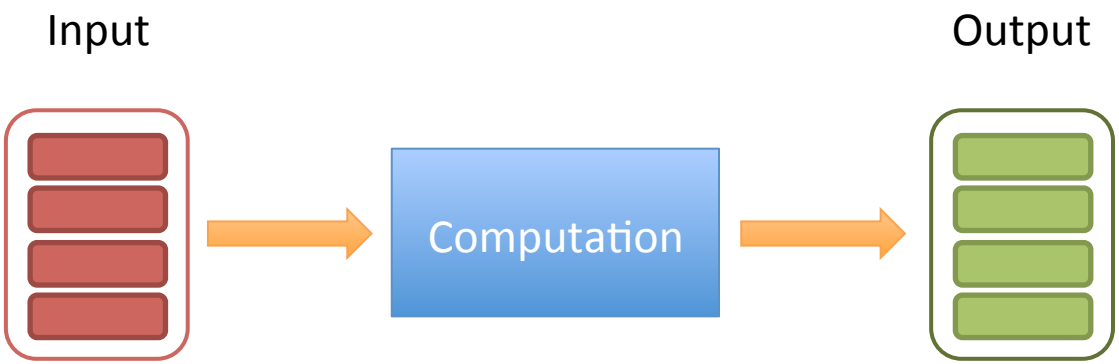
Large-scale computation without sacrificing expressiveness

Sangjin Han Sylvia Ratnasamy

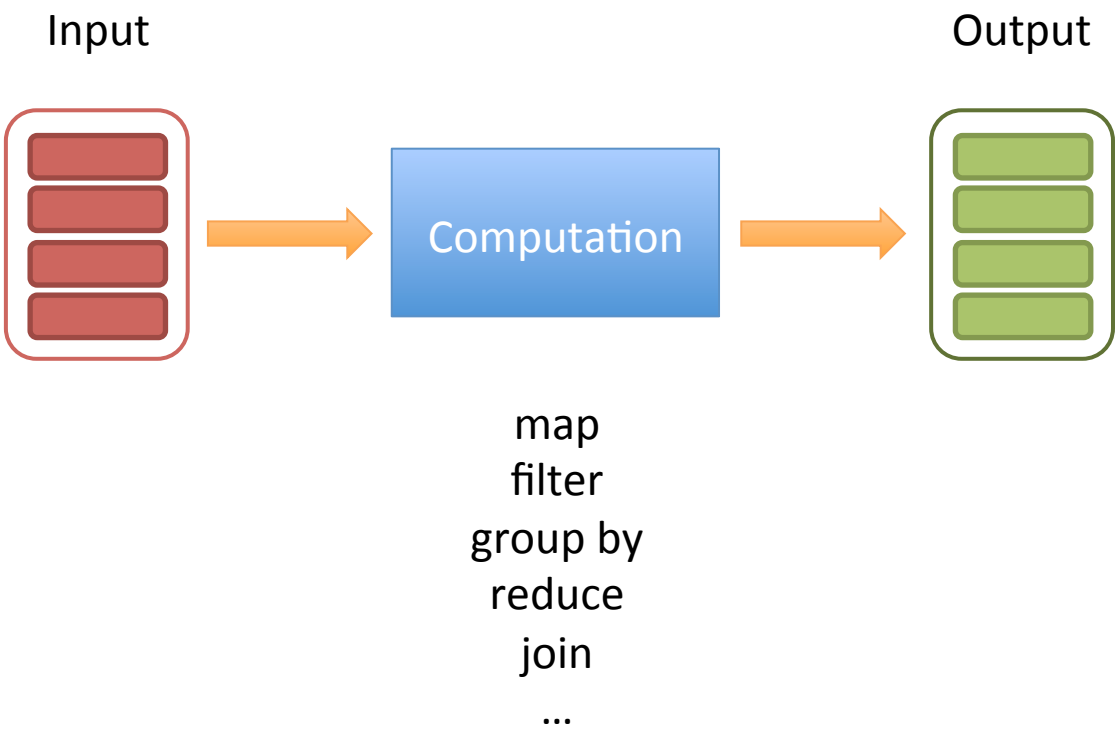
UC Berkeley

Review: MapReduce and Friends



map
filter
group by
reduce
join
…

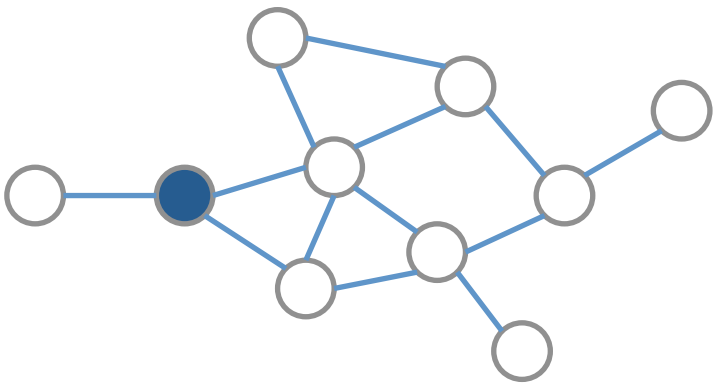
Review: MapReduce and Friends



Observation 1: Bulk transformation of immutable data
(no fine-grained updates)

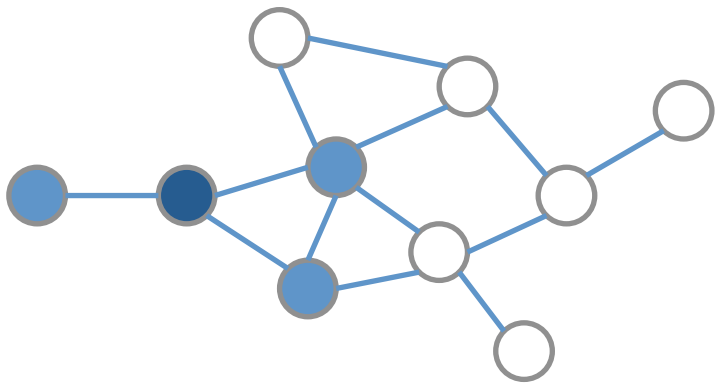
Example 1: Sparse Operations

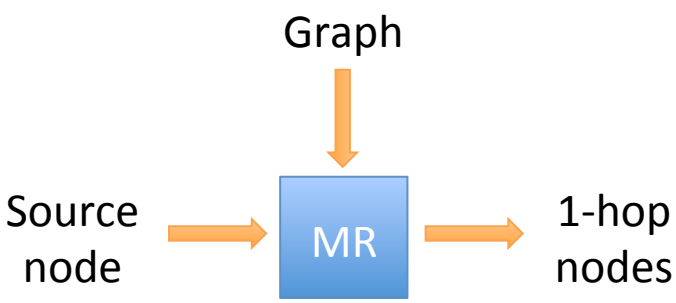
- k-hop reachability with iterative MapReduce



Example 1: Sparse Operations

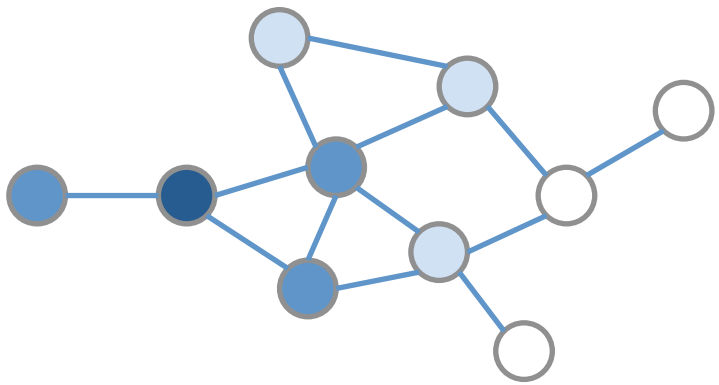
- k-hop reachability with iterative MapReduce

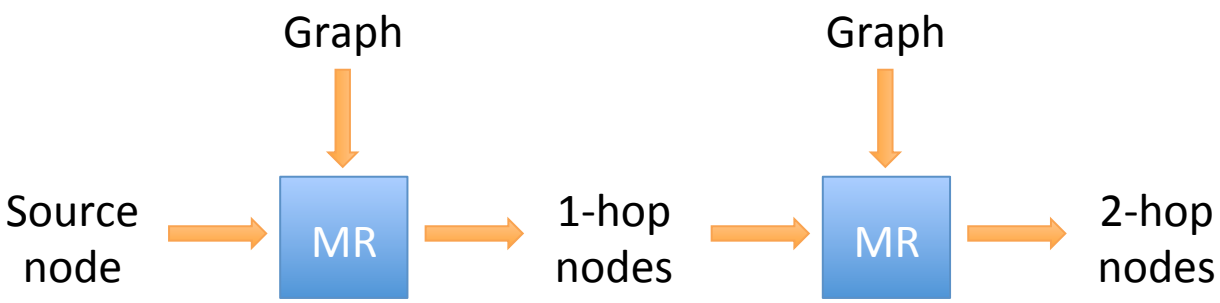




Example 1: Sparse Operations

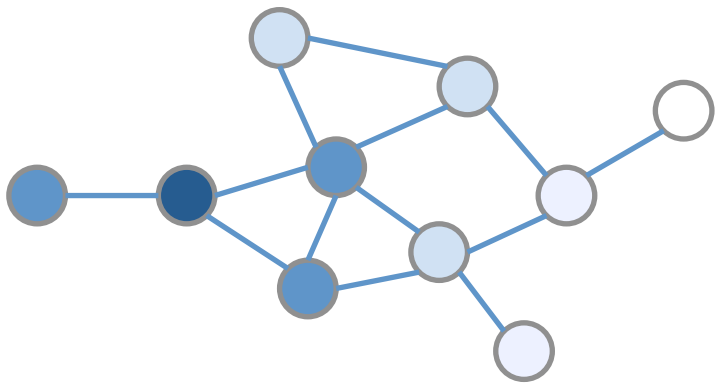
- k-hop reachability with iterative MapReduce

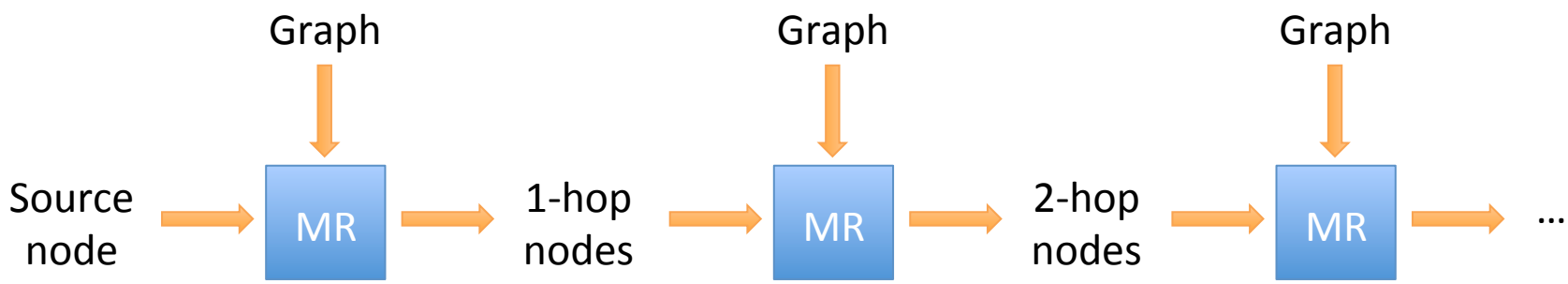




Example 1: Sparse Operations

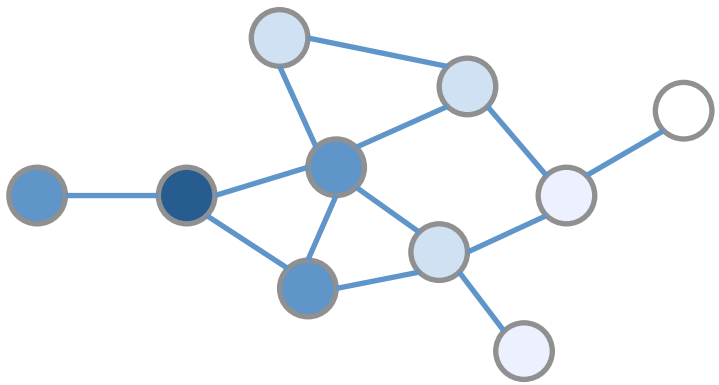
- k-hop reachability with iterative MapReduce

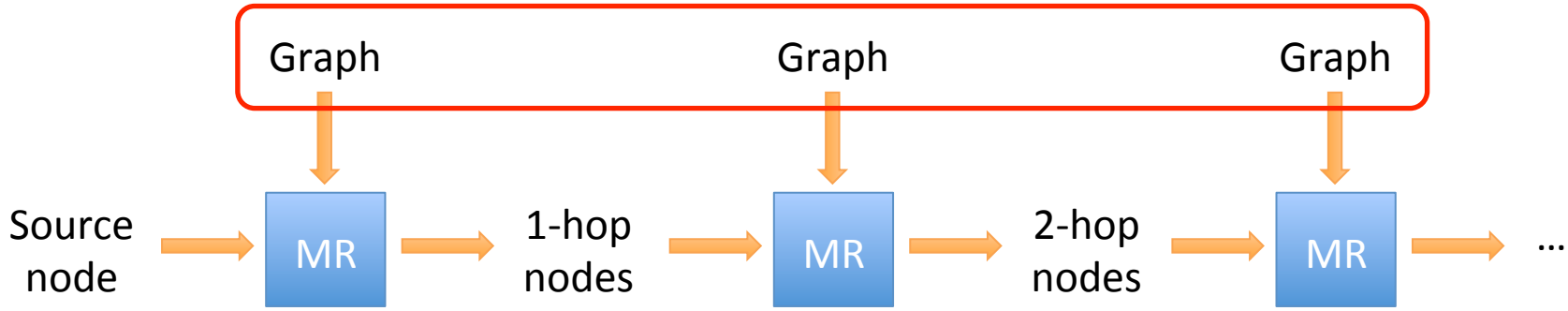




Example 1: Sparse Operations

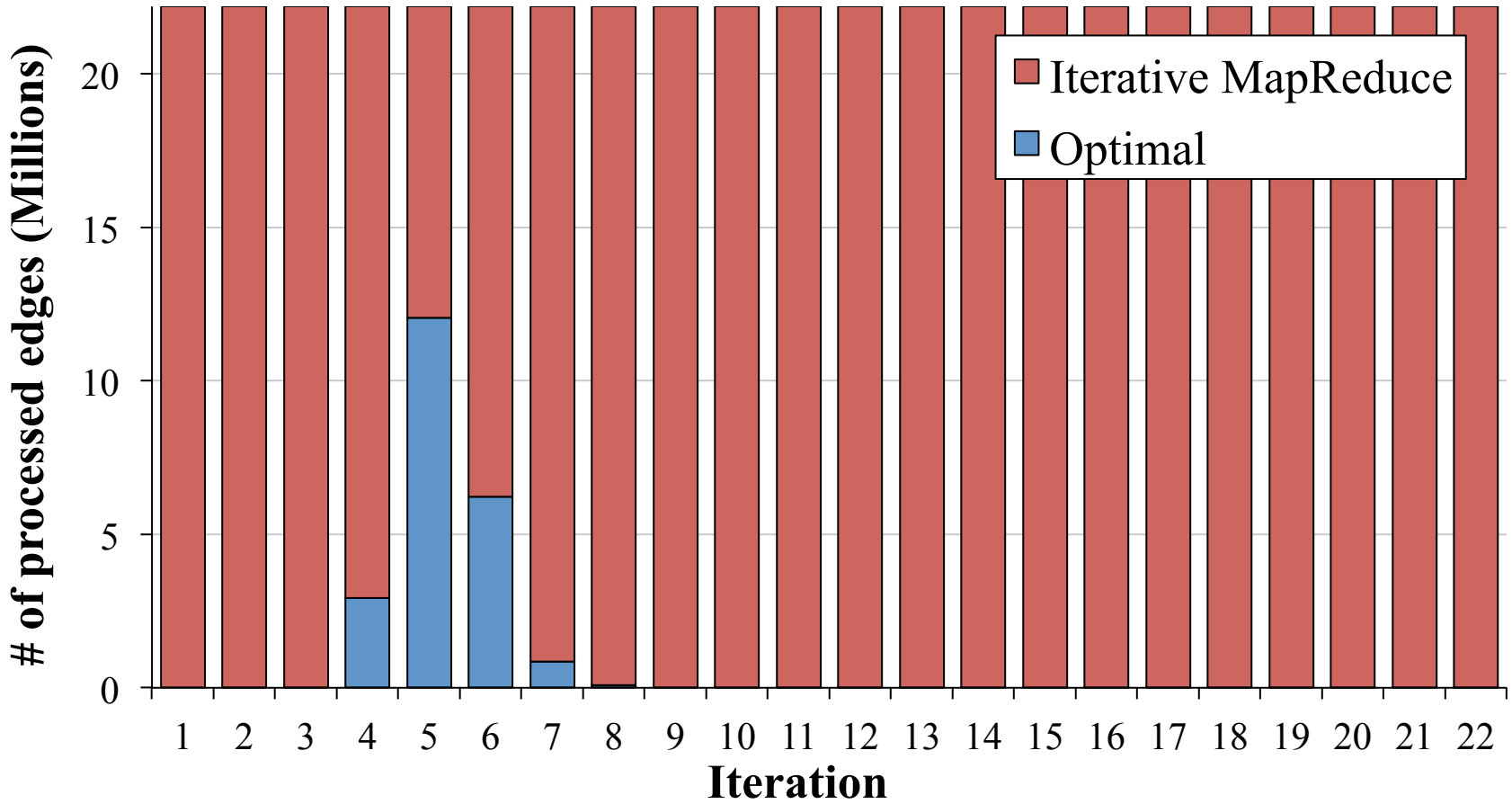
- k-hop reachability with iterative MapReduce





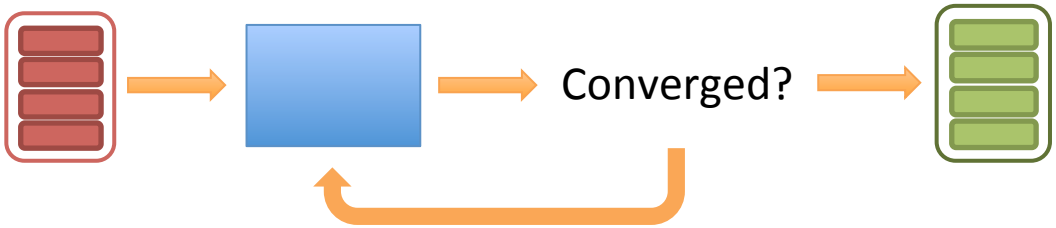
Example 1: Sparse operations

- k-hop reachability with iterative MapReduce

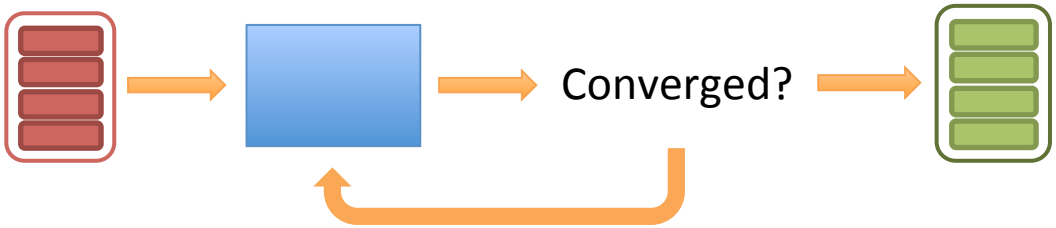


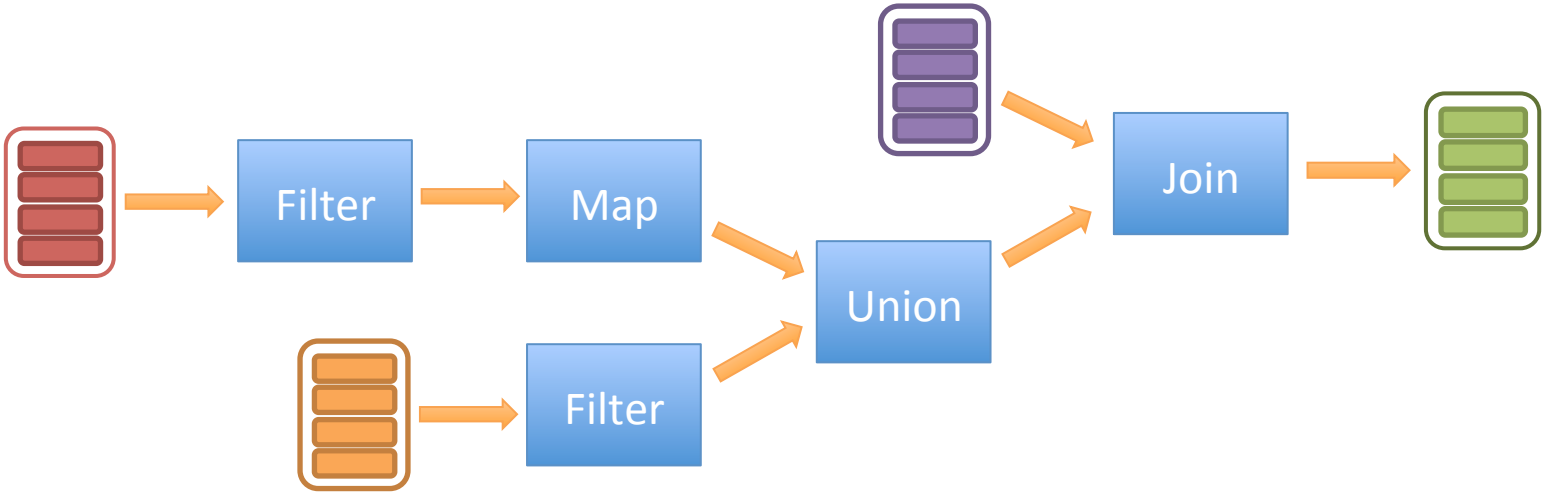
Internet router topology graph (1.7M nodes, 22.2M edges)

Review: MapReduce and Friends (cont'd)

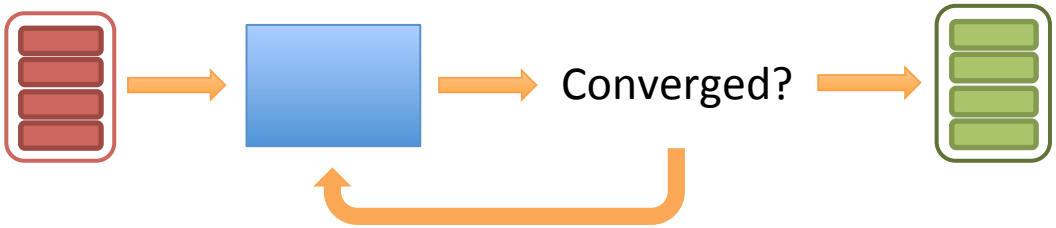


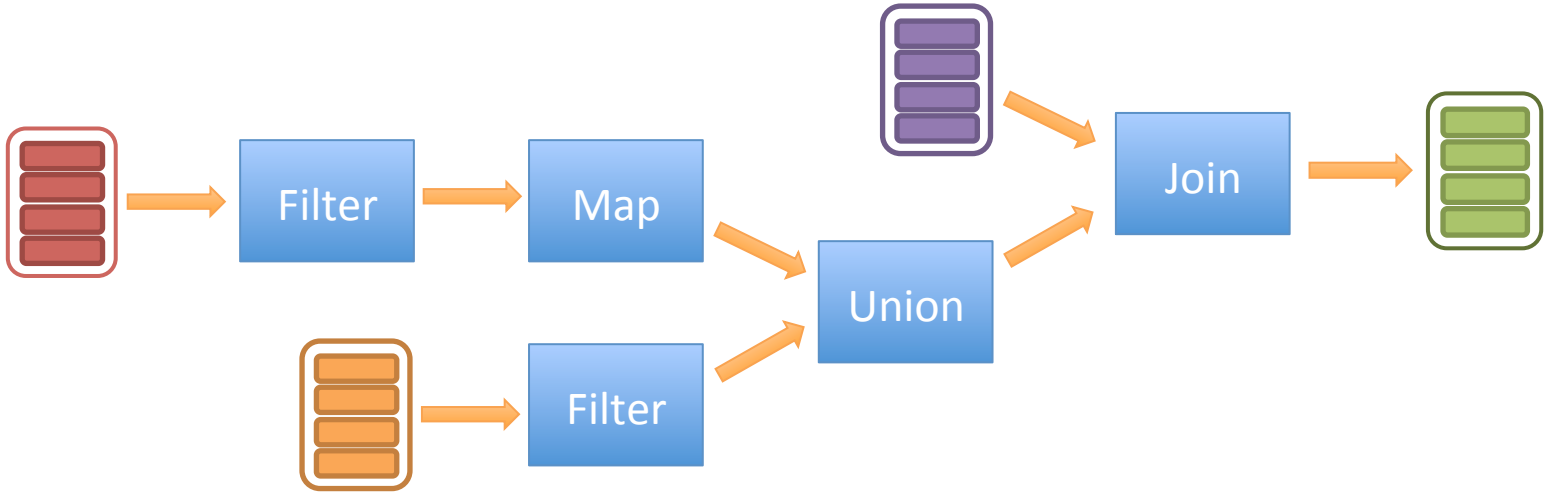
Review: MapReduce and Friends (cont'd)





Review: MapReduce and Friends (cont'd)





Observation 2: Static dataflow
(no data-dependent control flow)

Example 2: Irregular parallelism

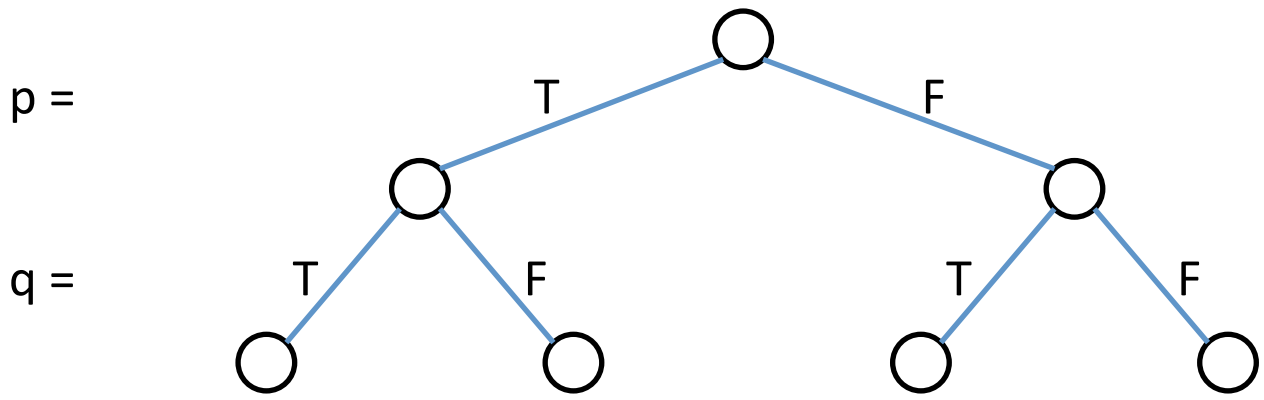
- Parallel SAT solver

$$E = (p \vee !q) \wedge (!p \vee r \vee s) \wedge (q \vee !s \vee !t) \wedge (!p \vee s) \wedge \cdots$$

Example 2: Irregular parallelism

- Parallel SAT solver

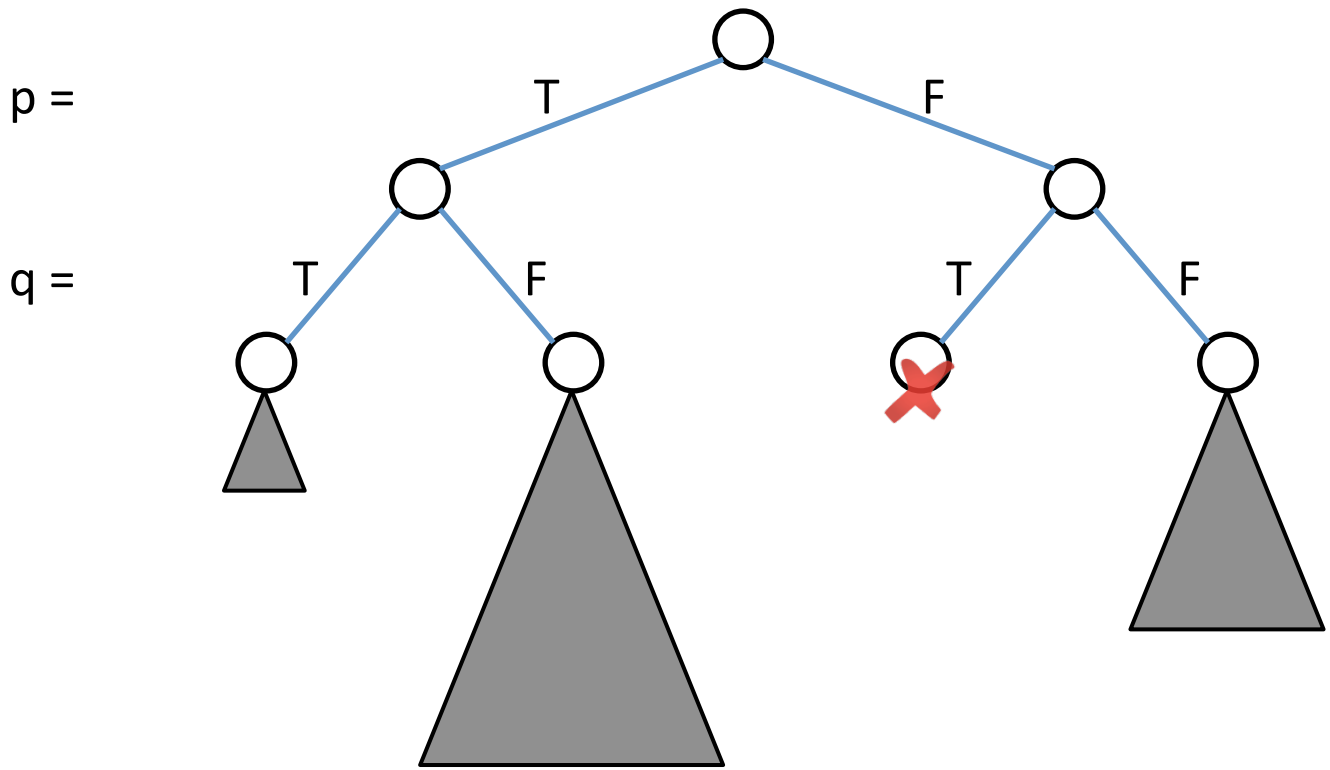
$$E = (p \vee !q) \wedge (!p \vee r \vee s) \wedge (q \vee !s \vee !t) \wedge (!p \vee s) \wedge \cdots$$



Example 2: Irregular parallelism

- Parallel SAT solver

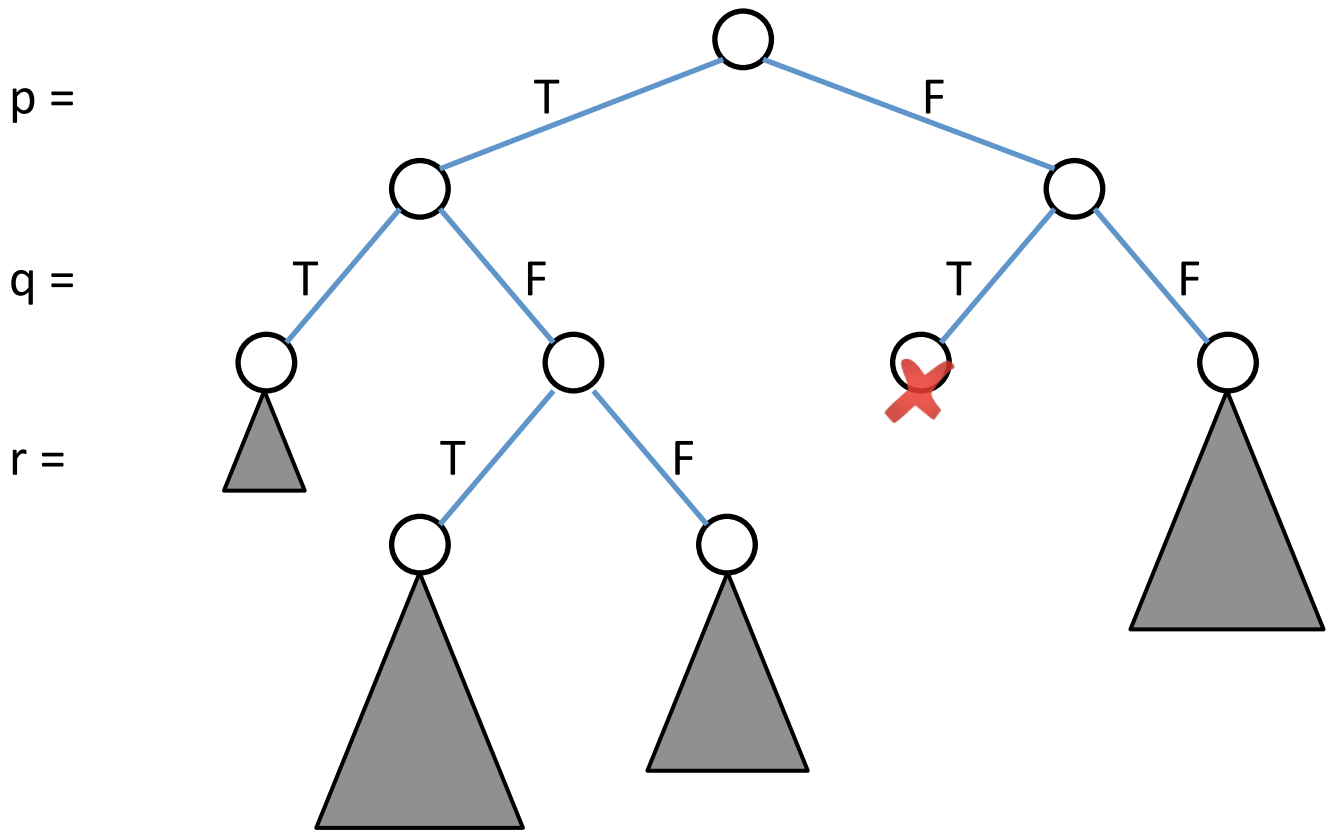
$$E = (p \vee !q) \wedge (!p \vee r \vee s) \wedge (q \vee !s \vee !t) \wedge (!p \vee s) \wedge \cdots$$



Example 2: Irregular parallelism

- Parallel SAT solver

$$E = (p \vee !q) \wedge (!p \vee r \vee s) \wedge (q \vee !s \vee !t) \wedge (!p \vee s) \wedge \cdots$$



MapReduce-like frameworks assume:

1. Bulk transformation of immutable data

2. Static dataflow

~~Existing frameworks assume:~~

Our work:

1. ~~Bulk transformation of immutable data~~

   Fine-grained operations on mutable data

2. ~~Static dataflow~~

   Dynamic, data-dependent control flow

Yet we still want elastic scalability and fault tolerance

Spinning a small twist to Linda

CELIAS PROGRAMMING MODEL

Programming model =

data model + computation model

Data Models for Mutable Shared Memory

Data Models for Mutable Shared Memory

Global address space: UPC, X10, Fortress…

Too low level

Data Models for Mutable Shared Memory

Global address space: UPC, X10, Fortress…

Too low level

Key-value tables: RAMCloud, Dynamo, Piccolo…

Key	Value
…	…

Limited lookup ability

Consistency concerns

Data Models for Mutable Shared Memory

Global address space: UPC, X10, Fortress…

Too low level

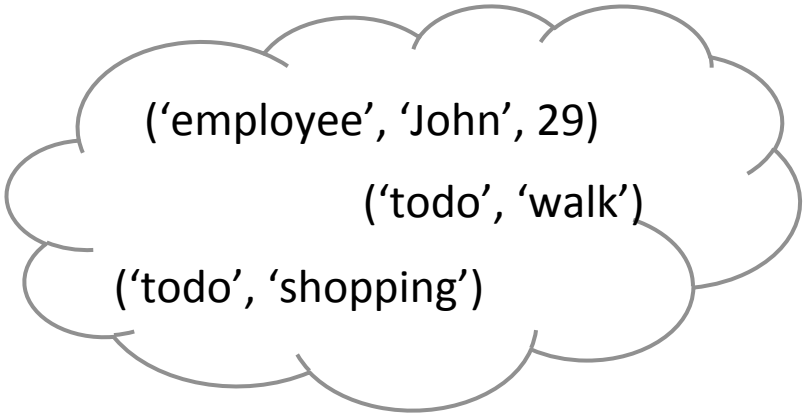
Key-value tables: RAMCloud, Dynamo, Piccolo…

Key	Value
…	…

Limited lookup ability

Consistency concerns

Tuplespace: Linda



Flexible lookup with any attributes

Individual tuples are immutable

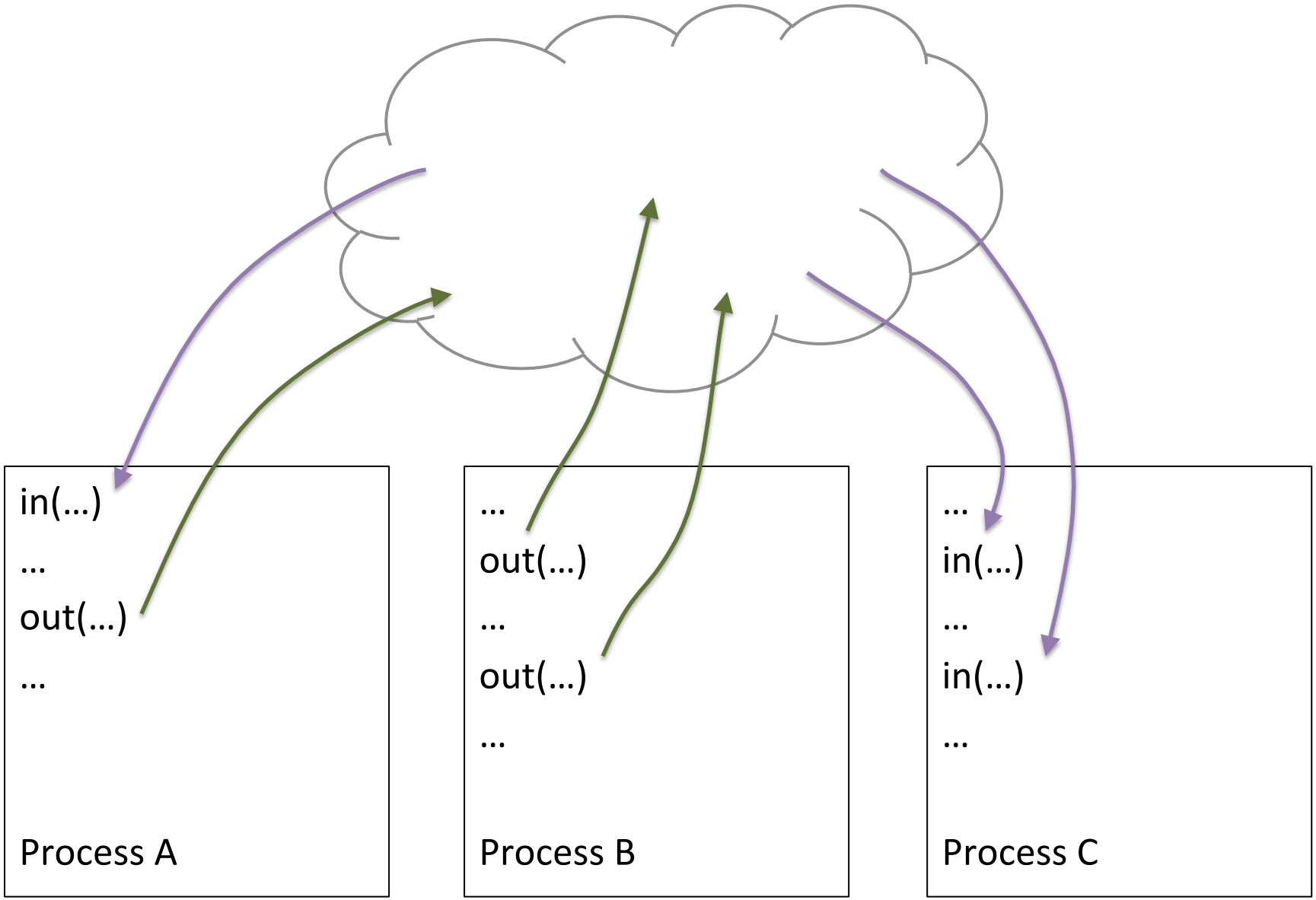
Programming model =

data model + computation model

Linda =

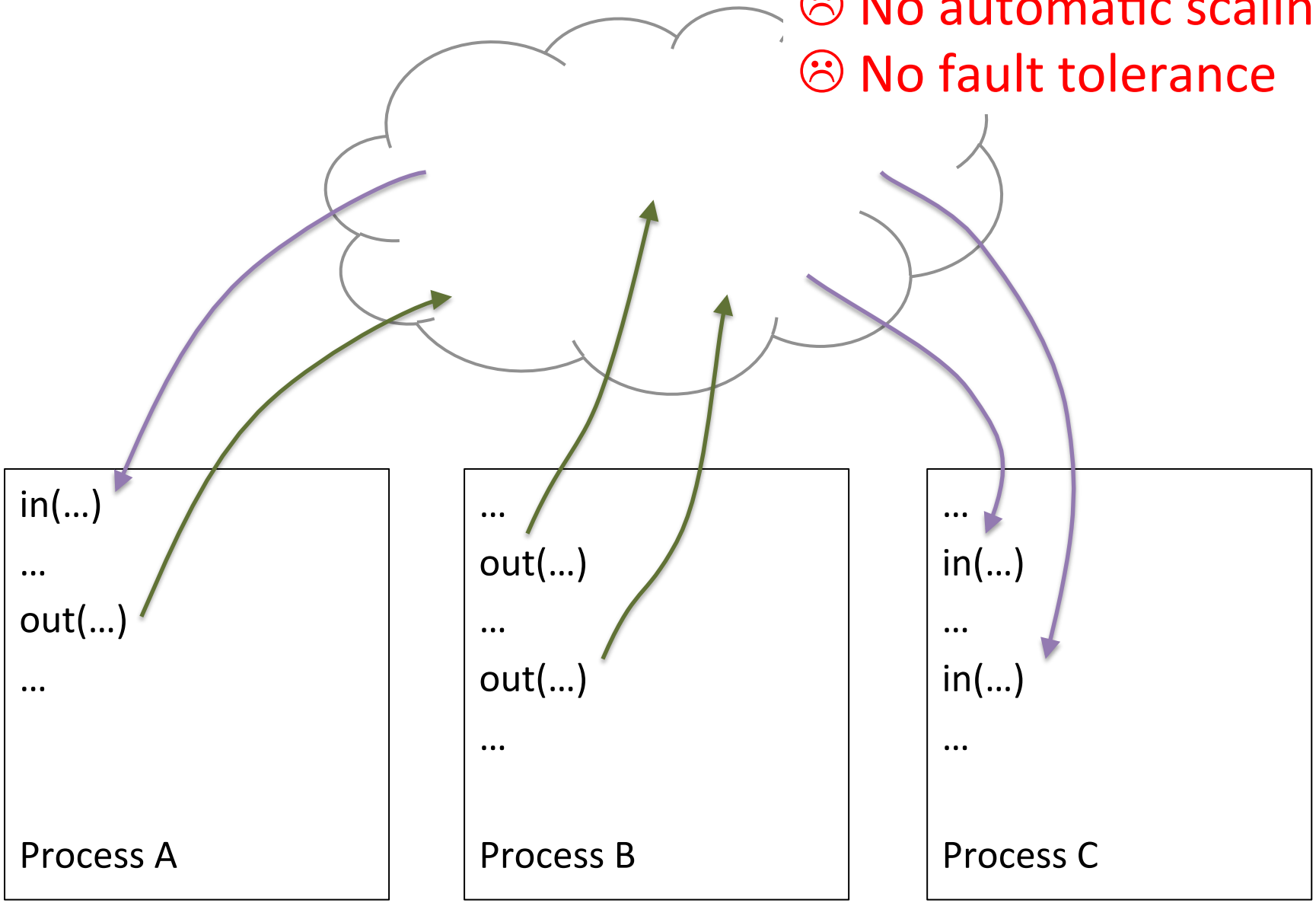
Tuplespace + Linda processes

Linda Processes



Linda Processes

☹ No automatic scaling
☹ No fault tolerance



Programming model =

data model + computation model

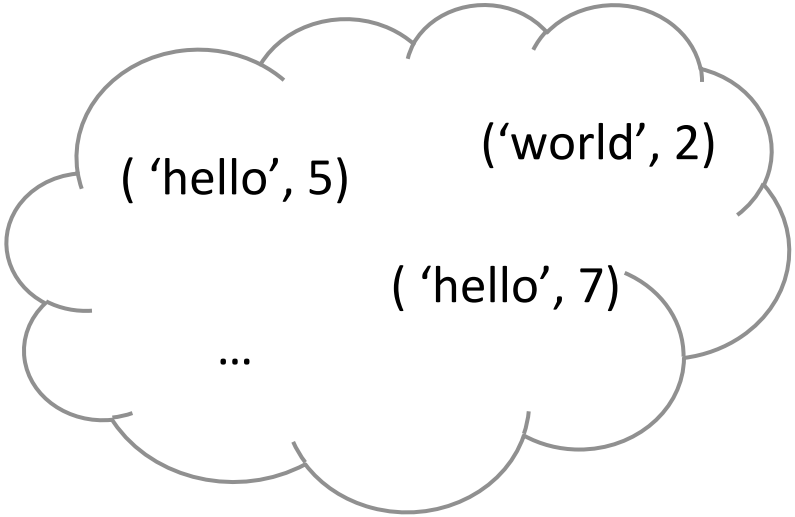
Linda =

Tuplespace + Linda processes

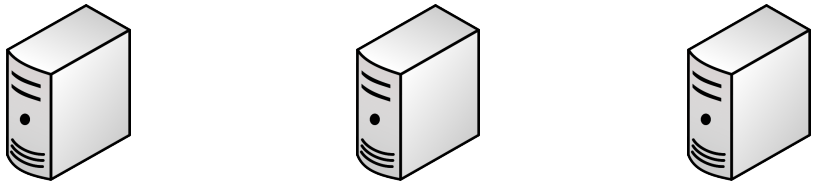
Celias =

Tuplespace + microtasks

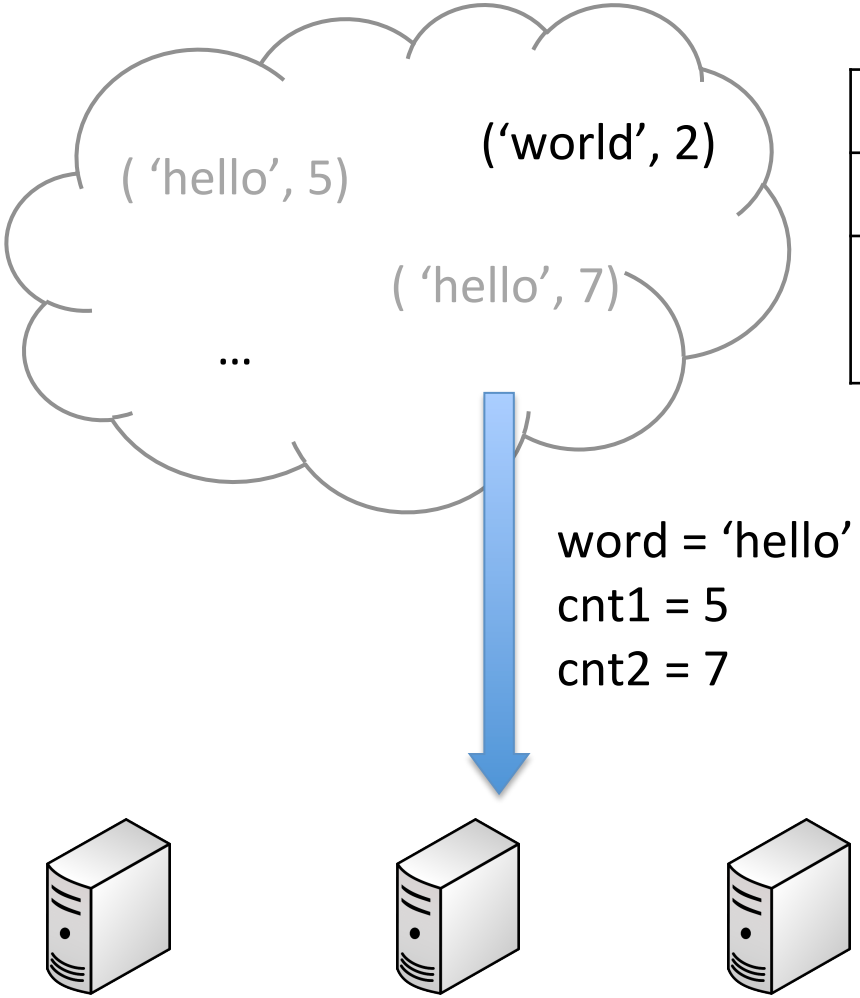
Microtasks



Function wordcount()	
Signature	(?word, ?cnt1), (?word, ?cnt2)
Code	sum := cnt1 + cnt2 emit (word, sum)



Microtasks

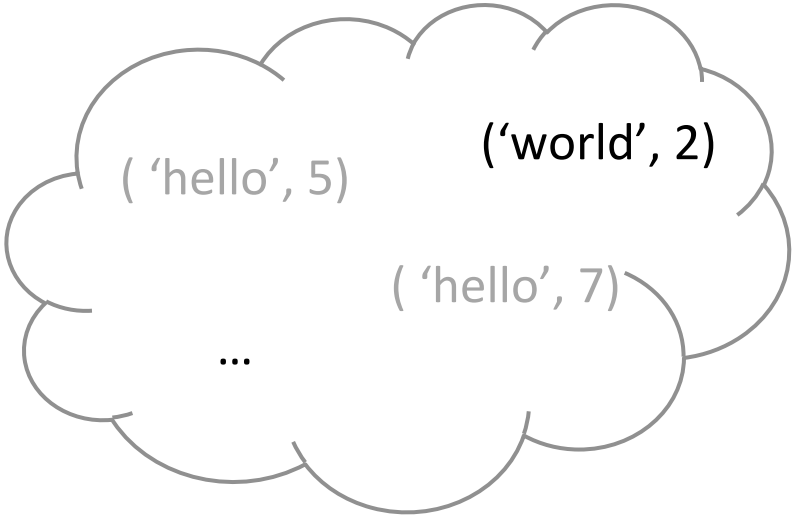


Function wordcount()	
Signature	(?word, ?cnt1), (?word, ?cnt2)
Code	sum := cnt1 + cnt2 emit (word, sum)

When a signature matches:

1. microtask launch

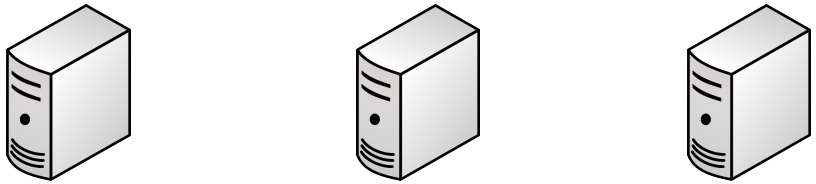
Microtasks



Function wordcount()	
Signature	(?word, ?cnt1), (?word, ?cnt2)
Code	sum := cnt1 + cnt2 emit (word, sum)

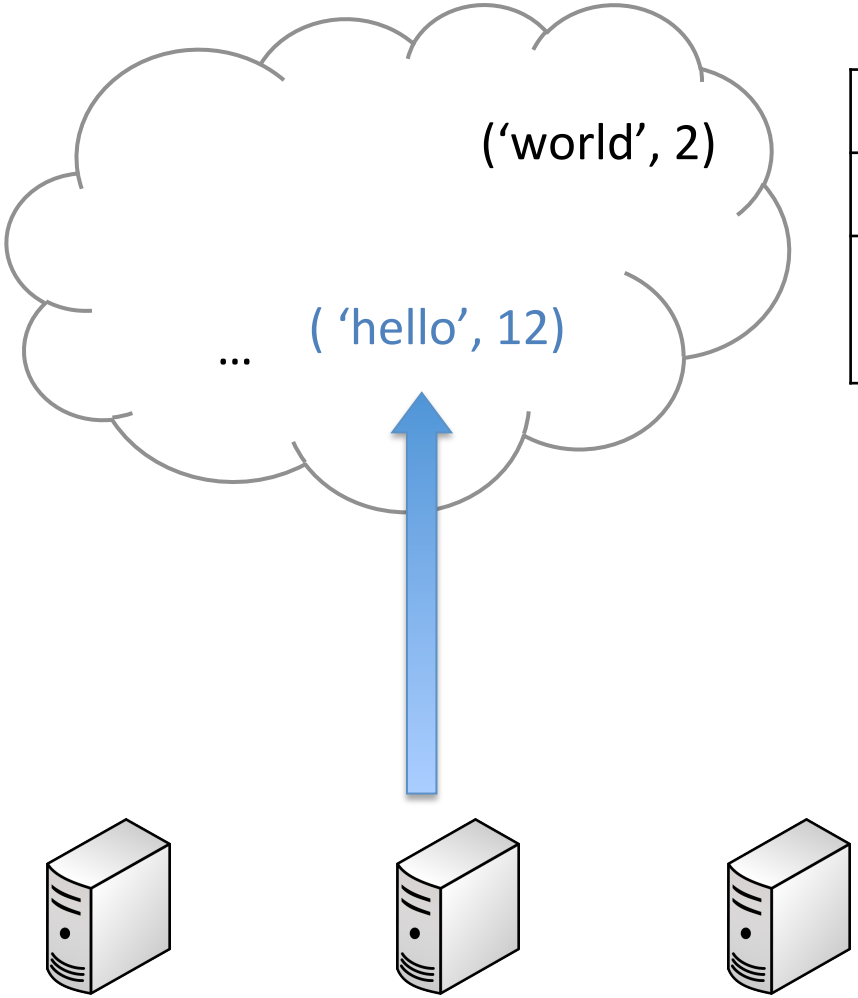
When a signature matches:

1. microtask launch
2. code execution



5 + 7 = ??

Microtasks



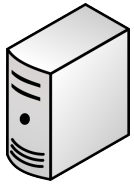
Function wordcount()	
Signature	(?word, ?cnt1), (?word, ?cnt2)
Code	sum := cnt1 + cnt2 emit (word, sum)

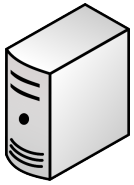
When a signature matches:

1. microtask launch
2. code execution
3. atomic replacement

Two functions: add() and multiply()

$$(A + B) \times (C + D)$$

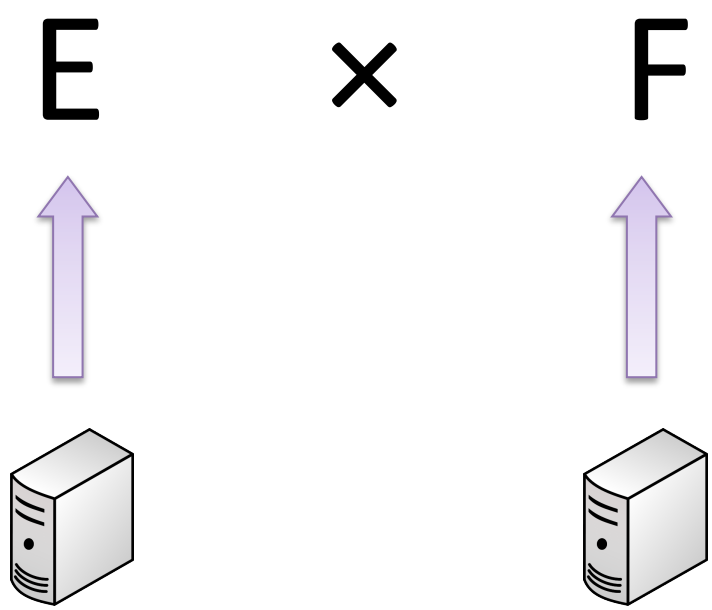




Two functions: add() and multiply()

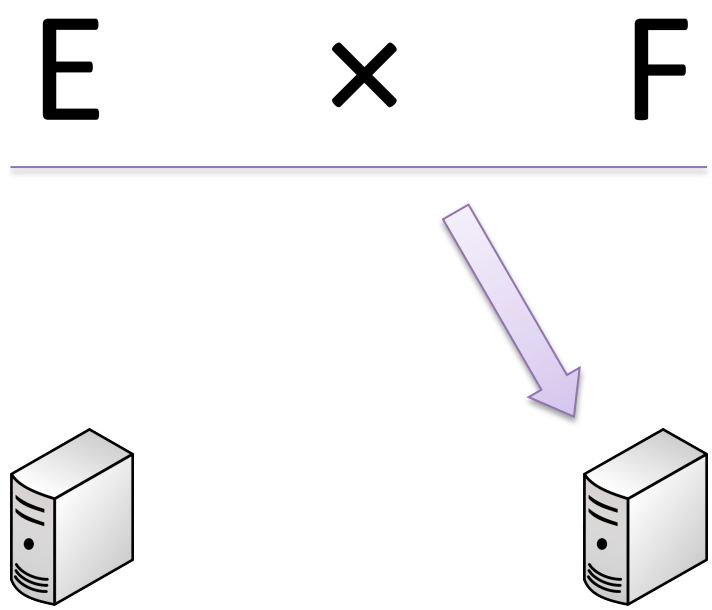
$$(A + B) \times (C + D)$$

Two functions: add() and multiply()



☺ Automatic scaling

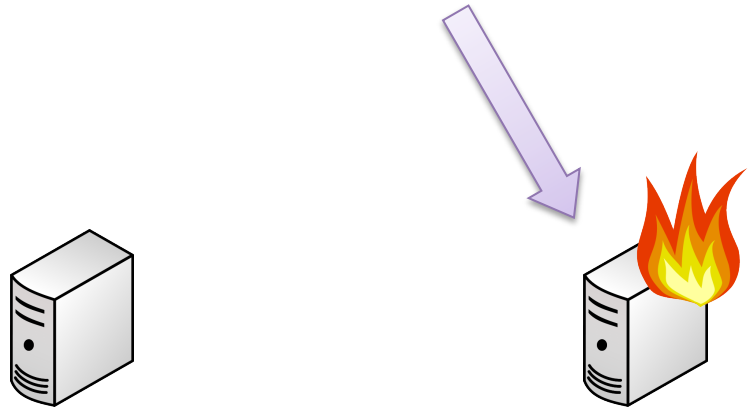
Two functions: add() and multiply()



☺ Automatic scaling

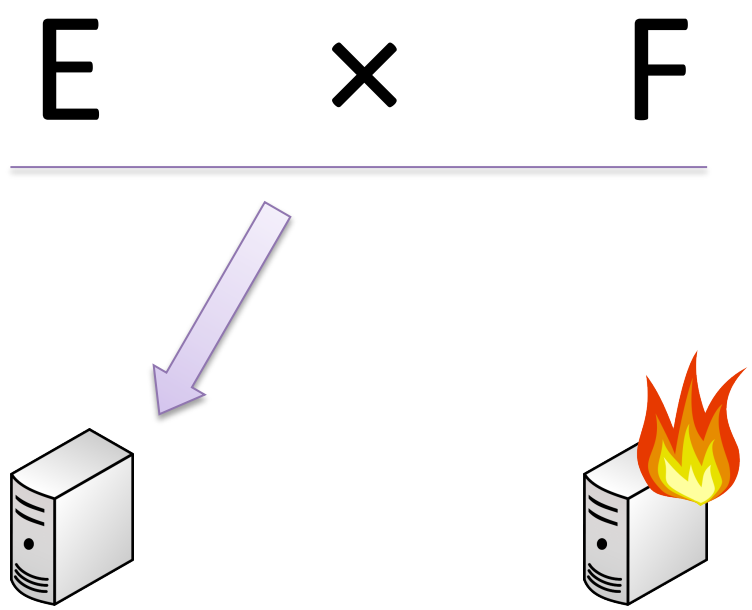
Two functions: add() and multiply()

E × F



☺ Automatic scaling

Two functions: add() and multiply()



☺ Automatic scaling

☺ Fault tolerance

More Examples in the Paper…

- MapReduce
  - Celias is ~~Turing-complete~~ MapReduce-complete!
  - without any artificial sync. barriers
- Single-source shortest path
  - Pregel-style graph processing
- Quicksort
  - Recursive control flow

Summary

- MapReduce-like frameworks are not suitable for algorithms with:
  - Sparse/incremental/fine-grained computation
  - Dynamic dataflow

- Celias comes to our rescue, yet it is also
  - automatically scalable
  - fault tolerant

Open Questions

- Microtask abstraction: good enough? went too far?
- Feasibility of an efficient implementation
  - Reliable tuplespace
  - Signature matching
  - Microtask transactions
- … what is a killer app of Celias?
- <Your questions here>