Comparison of MapReduce with Bulk-Synchronous Systems

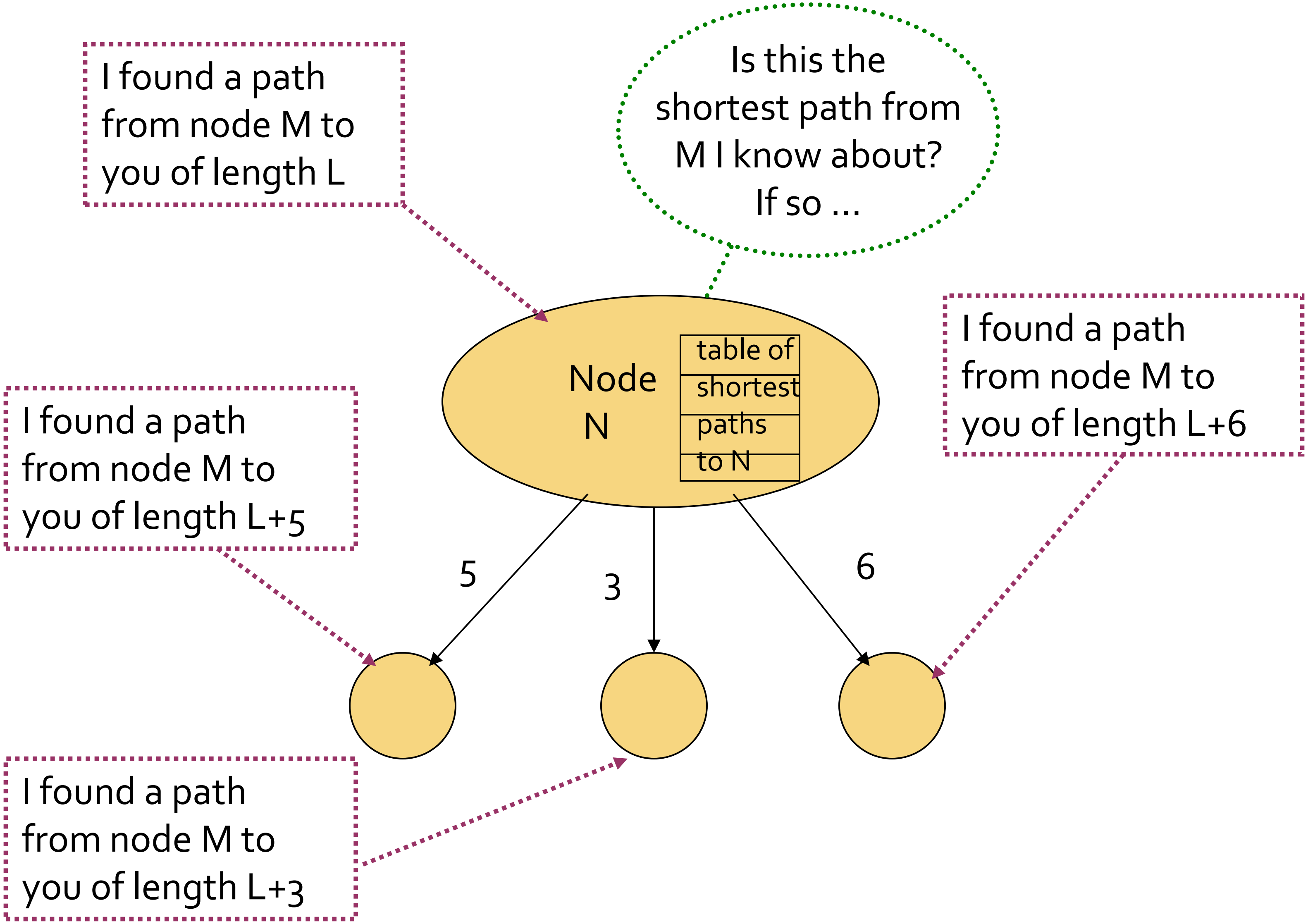
Review of Bulk-Synchronous Communication Costs Problem of Semijoin

Jeffrey D. Ullman
Stanford University

The Graph Model

- Views all computation as a recursion on some graph.
- Graph nodes send messages to one another.
  - Messages bunched into *supersteps*, where each graph node processes all data received.
  - Sending individual messages would result in far too much overhead.
- Checkpoint all compute nodes after some fixed number of supersteps.
- On failure, rolls all tasks back to previous checkpoint.

Example: Shortest Paths



Some Systems

- *Pregel*: the original, from Google.
- *Giraph*: open-source (Apache) Pregel.
  - Built on Hadoop.
- *GraphX*: a similar front end for Spark.
- *GraphLab*: similar system that deals more effectively with nodes of high degree.
  - Will split the work for such a graph node among several compute nodes.

The Tyranny of Communication

- All these systems move data between tasks.
  - It is rare that (say) a Map task feeds a Reduce task at the same compute node.
  - And even so, you probably need to do disk I/O.
- Gigabit communication seems like a lot, but it is often the bottleneck.

Two Approaches

- There is a subtle difference regarding how one avoids moving big data in MapReduce and Bulk-Synchronous systems.
- Example: join of R(A,B) and S(B,C), where:
  - A is a really large field – a video.
  - B is the video ID.
  - S(B,C) is a small number of streaming requests, where C is the destination.
- If we join R and S, most R-tuples move to the reducer for the B-value needlessly.

The Semijoin

- Might want to *semijoin* first: find all the values of B in S, and filter those (a,b) in R that are *dangling* (will not join with anything in S).
- Then Map need not move dangling tuples to any reducer.
- But the obvious approach to semijoin also requires that every R-tuple be sent by its mapper to some reducer.

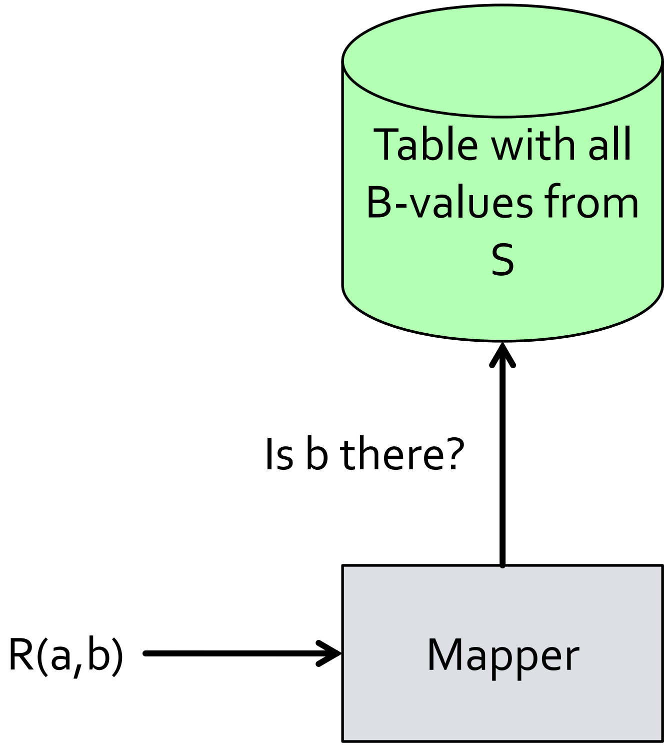
Semijoin – (2)

- To semijoin R(A,B) with S(B,C), use B as the key for both relations.
  - From R(a,b) create key-value pair (b, (R,a)).
  - From S(b,c) create key-value pair (b,S).
    - Almost like join, but you don't need the C-value.

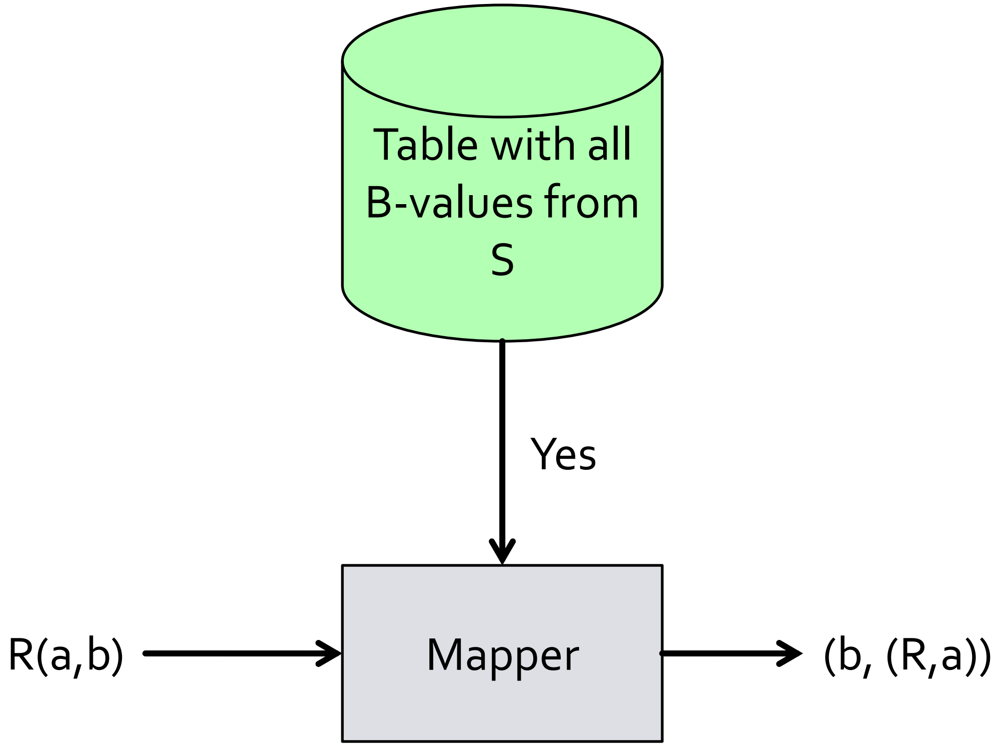
The MapReduce Solution

- Recent implementations of MapReduce allow distribution of "small" amounts of data to every compute node.
- Project S onto B to form set S' and distribute S' everywhere.
- Then, run the standard MapReduce join, but have the Map function check that (a,b) has b in S' before emitting it as a key-value pair.
  - If most tuples in R are dangling, it saves substantial communication.

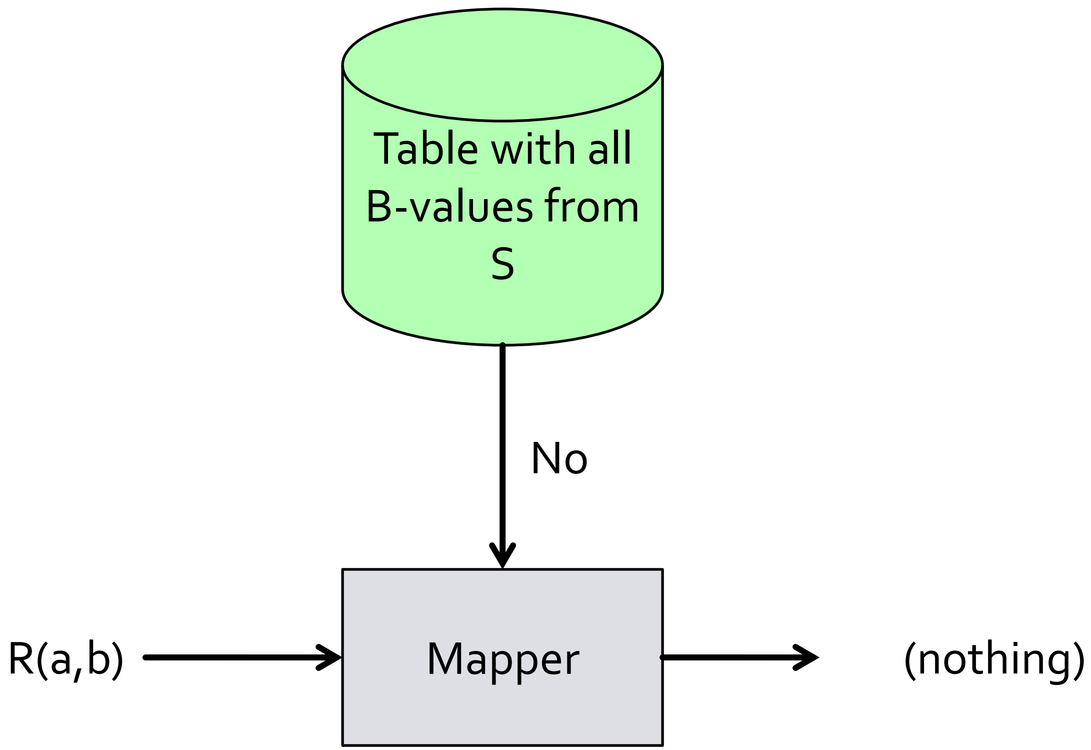
Semijoin in the Mappers



Semijoin in the Mappers – "Yes"



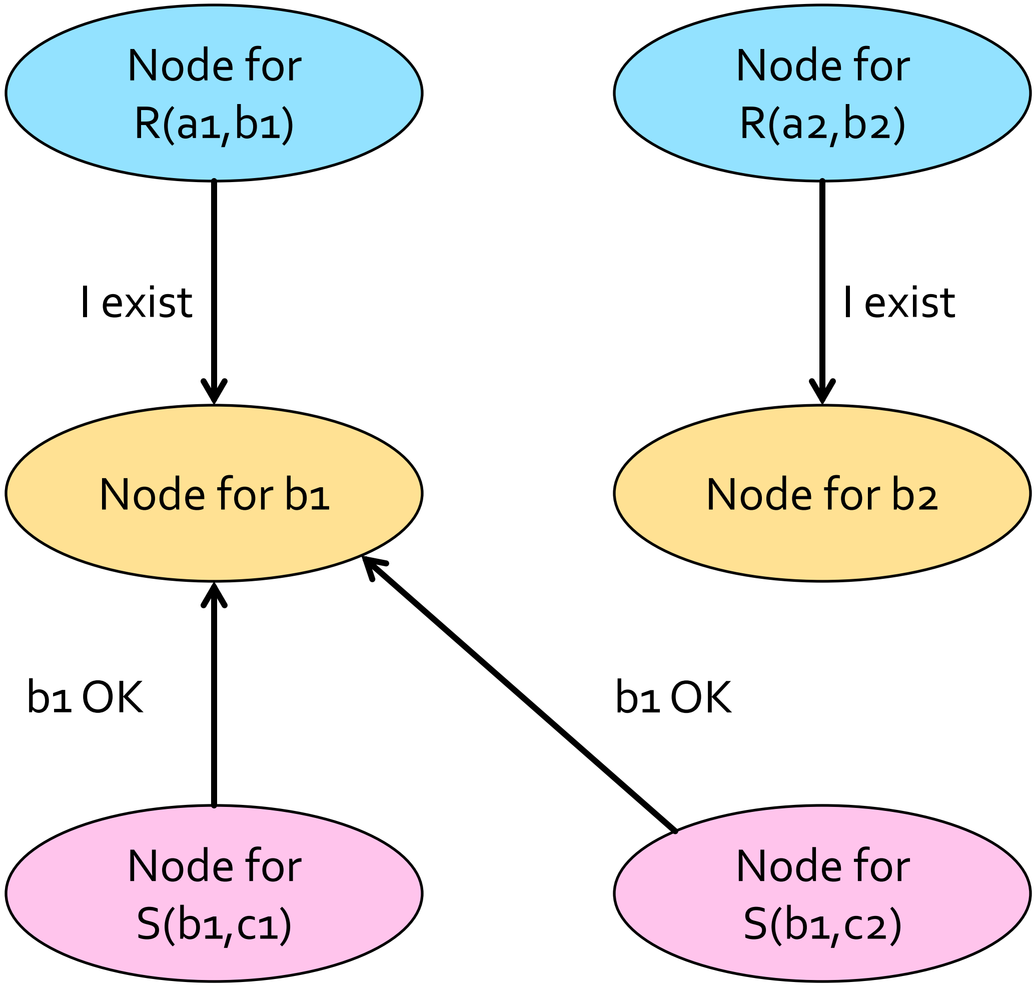
Semijoin in the Mappers – "No"



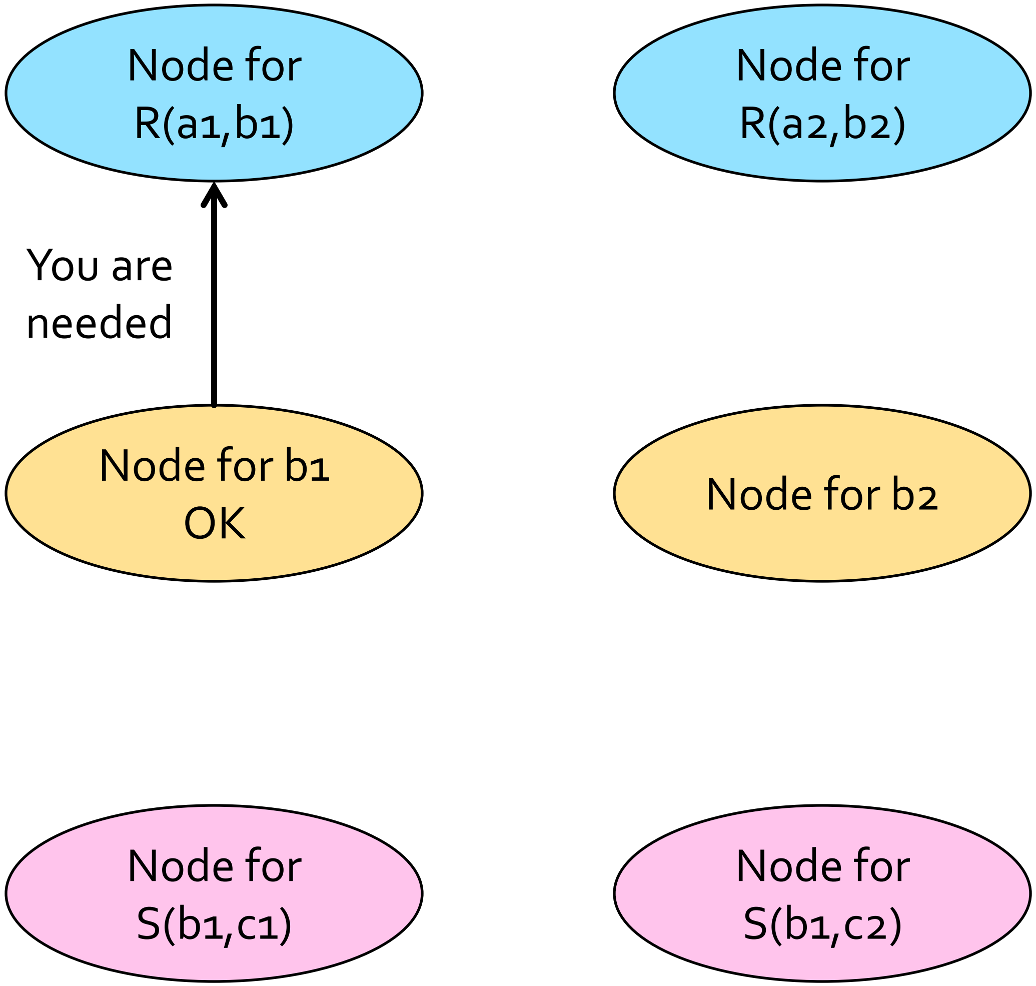
The Bulk-Synchronous Solution

- Create a graph node for every tuple, and also for every B-value.
- All tuples (b,c) from S send a message to the node for B-value b.
- All tuples (a,b) from R send a message with their node name to the node for B-value b.
- The node for b sends messages to all (a,b) in R, provided it has received at least one message from a tuple in S.
- Now, we can mimic the MapReduce join without considering dangling tuples.

Bulk-Synchronous Semijoin



Bulk-Synchronous Semijoin – (2)



Bulk-Synchronous Join Phase