CS 764: Topics in Database Management Systems

Lecture 13: Distributed DBMSs

Xiangyao Yu

10/19/2020

Announcement

Project proposal due: ~~Oct 21~~ **Oct 26**

Please submit your proposal to the paper review website:

https://wisc-cs764-f20.hotcrp.com

Discussion

High-level interface like SQL

- Any programming language (functional language, python, java)
- Spark, MapReduce
- File system, network API, virtual memory, TensorFlow, PyTorch

Optimizations for storage-disaggregation architecture

- Optimize for data locality: use replica close to computation
- Higher level of consistency for OLTP than OLAP
- Offload some computation to storage (selection/projection)
- Cache intermediate results in the memory of compute nodes
- OLTP: execute select, update, insert, delete completely on storage nodes

Today's Paper: Mariposa

Mariposa: a wide-area distributed database system

Michael Stonebraker, Paul M. Aoki, Witold Litwin[1], Avi Pfeffer[2], Adam Sah, Jeff Sidell, Carl Staelin[3], Andrew Yu[4]

Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720-1776, USA

Edited by Henry F. Korth and Amit Sheth. Received November 1994 / Revised June 1995 / Accepted September 14, 1995

Abstract. The requirements of wide-area distributed database systems differ dramatically from those of local-area network systems. In a wide-area network (WAN) configuration, individual sites usually report to different system administrators, have different access and charging algorithms, install site-specific data type extensions, and have different constraints on servicing remote requests. Typical of the last point are production transaction environments, which are fully engaged during normal business hours, and cannot take on additional load. Finally, there may be many sites participating in a WAN distributed DBMS.

In this world, a single program performing global query optimization using a cost-based optimizer will not work well. Cost-based optimization does not respond well to site-specific type extension, access constraints, charging algorithms, and time-of-day constraints. Furthermore, traditional cost-based distributed optimizers do not scale well to a large number of possible processing sites. Since traditional distributed DBMSs have all used cost-based optimizers, they are not appropriate in a WAN environment, and a new architecture is required.

We have proposed and implemented an economic paradigm as the solution to these issues in a new distributed DBMS called Mariposa. In this paper, we present the architecture and implementation of Mariposa and discuss early feedback on its operating characteristics.

Key words: Databases – Distributed systems – Economic site – Autonomy – Wide-area network – Name service

[1] *Present address*: Université Paris IX Dauphine, Section MIAGE, Place de Lattre de Tassigny, 75775 Paris Cedex 16, France

1 Introduction

The Mariposa distributed database system addresses a fundamental problem in the standard approach to distributed data management. We argue that the underlying assumptions traditionally made while implementing distributed data managers do not apply to today's wide-area network (WAN) environments. We present a set of guiding principles that must apply to a system designed for modern WAN environments. We then demonstrate that existing architectures cannot adhere to these principles because of the invalid assumptions just mentioned. Finally, we show how Mariposa can successfully apply the principles through its adoption of an entirely different paradigm for query and storage optimization.

Traditional distributed relational database systems that offer location-transparent query languages, such as Distributed INGRES (Stonebraker 1986), R* (Williams et al. 1981), SIRIUS (Litwin 1982) and SDD-1 (Bernstein 1981), all make a collection of underlying assumptions. These assumptions include:

– *Static data allocation*: In a traditional distributed DBMS, there is no mechanism whereby objects can quickly and easily change sites to reflect changing access patterns. Moving an object from one site to another is done manually by a database administrator, and all secondary access paths to the data are lost in the process. Hence, object movement is a very "heavyweight" operation and should not be done frequently.

– *Single administrative structure*: Traditional distributed database systems have assumed a query optimizer which decomposes a query into "pieces" and then decides where to execute each of these pieces. As a result, site selection for query fragments is done by the optimizer. Hence, there is no mechanism in traditional systems for a site to refuse to execute a query, for example because it is overloaded or oth-

VLDB Journal 1996

Why Mariposa?

Distributed DBMSs are all designed for local-area networks (LAN)

- **Static data allocation**: data movement is heavyweight and performed manually by a database administrator
- **Single administrative structure**: centralized optimizer; no site can refuse work, even under excessive load
- **Uniformity**: optimizer assumes all sites have same hardware, network, ample disk space, etc.

Why Mariposa?

Distributed DBMSs are all designed for local-area networks (LAN)

- **Static data allocation**: data movement is heavyweight and performed manually by a database administrator
- **Single administrative structure**: centralized optimizer; no site can refuse work, even under excessive load
- **Uniformity**: optimizer assumes all sites have same hardware, network, ample disk space, etc.

Assumptions no longer true in WAN environment

- Administrator for individual sites
- Constraints on servicing remote requests
- Non-uniform hardware

Main Goals of Mariposa

Scalability to a large number of sites (10K or more)

Main Goals of Mariposa

Scalability to a large number of sites (10K or more)

Data mobility: no fixed home of data. Data fragments can move freely between sites

Main Goals of Mariposa

Scalability to a large number of sites (10K or more)

Data mobility: no fixed home of data. Data fragments can move freely between sites

No global synchronization: no forced synchronization for data updates and schema changes.

Main Goals of Mariposa

Scalability to a large number of sites (10K or more)

Data mobility: no fixed home of data. Data fragments can move freely between sites

No global synchronization: no forced synchronization for data updates and schema changes.

Local autonomy: each site has control over its resources. Query and data allocation is not done by a central authoritarian query optimizer

Main Goals of Mariposa

Scalability to a large number of sites (10K or more)

Data mobility: no fixed home of data. Data fragments can move freely between sites

No global synchronization: no forced synchronization for data updates and schema changes.

Local autonomy: each site has control over its resources. Query and data allocation is not done by a central authoritarian query optimizer

Easily configurable policies: Local database administrator can change the behavior of a Mariposa site based on user activity and data access pattern

Economics in Mariposa

Resource management is reformulated into a microeconomic framework

- Clients and servers have network bank accounts
- Users allocate budget to each query
- Broker obtains bids for a query
- Servers bids on sub-queries
- **Goal: optimize revenue**

Economics in Mariposa

Resource management is reformulated into a microeconomic framework

- Clients and servers have network bank accounts
- Users allocate budget to each query
- Broker obtains bids for a query
- Servers bids on sub-queries
- **Goal: optimize revenue**

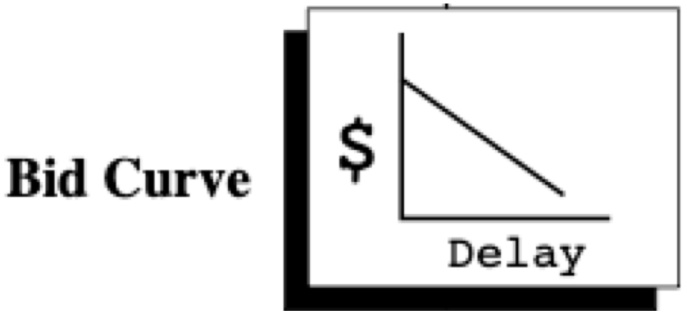
Why a microeconomic structure?

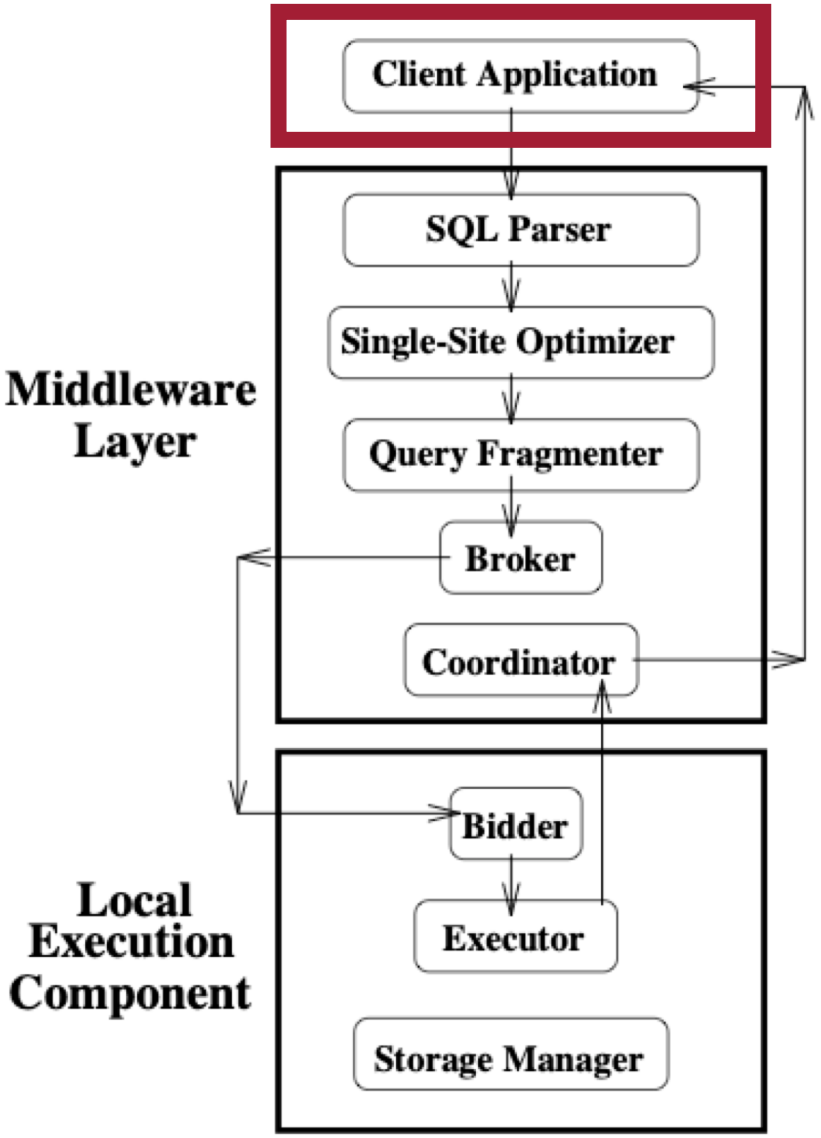
- Supports a large number of sites
- Sites can join and leave through buying and selling objects

Mariposa Architecture

Client

- Queries submitted by user applications to client site. Client site picks a query **budget** expressed as a **bid curve**

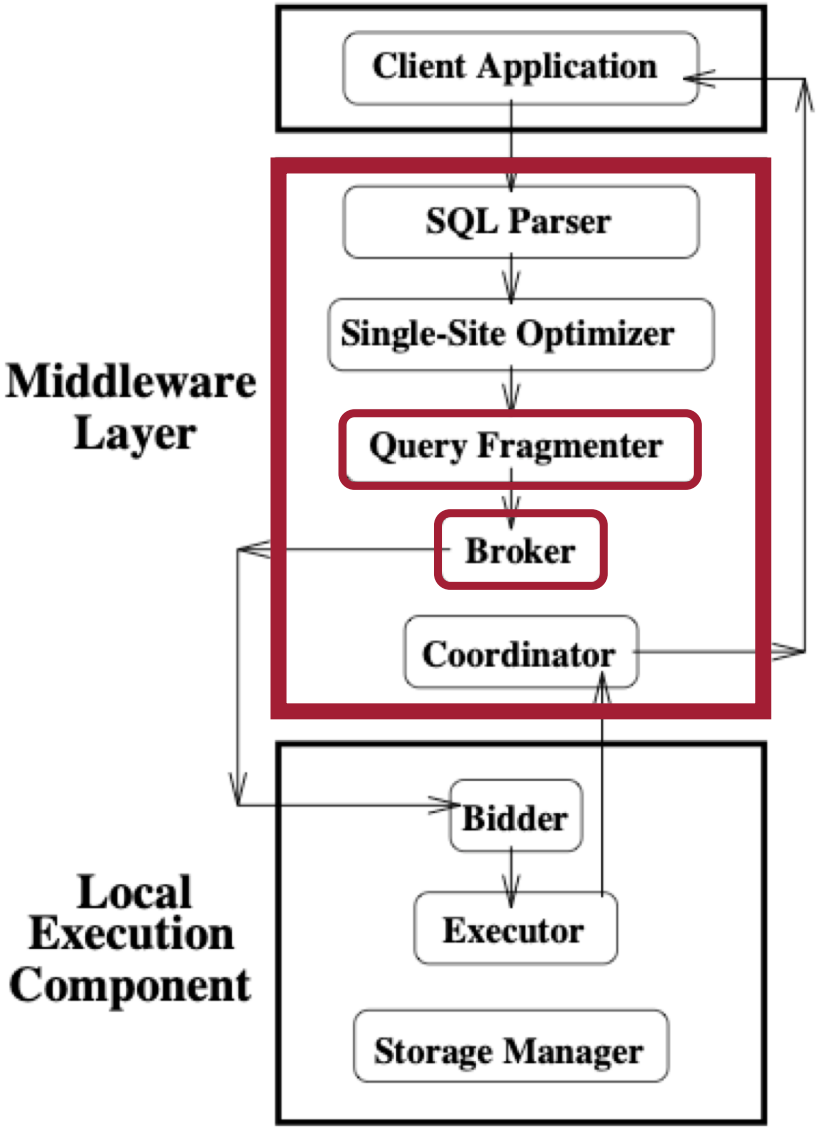




Mariposa Architecture

Middleware layer

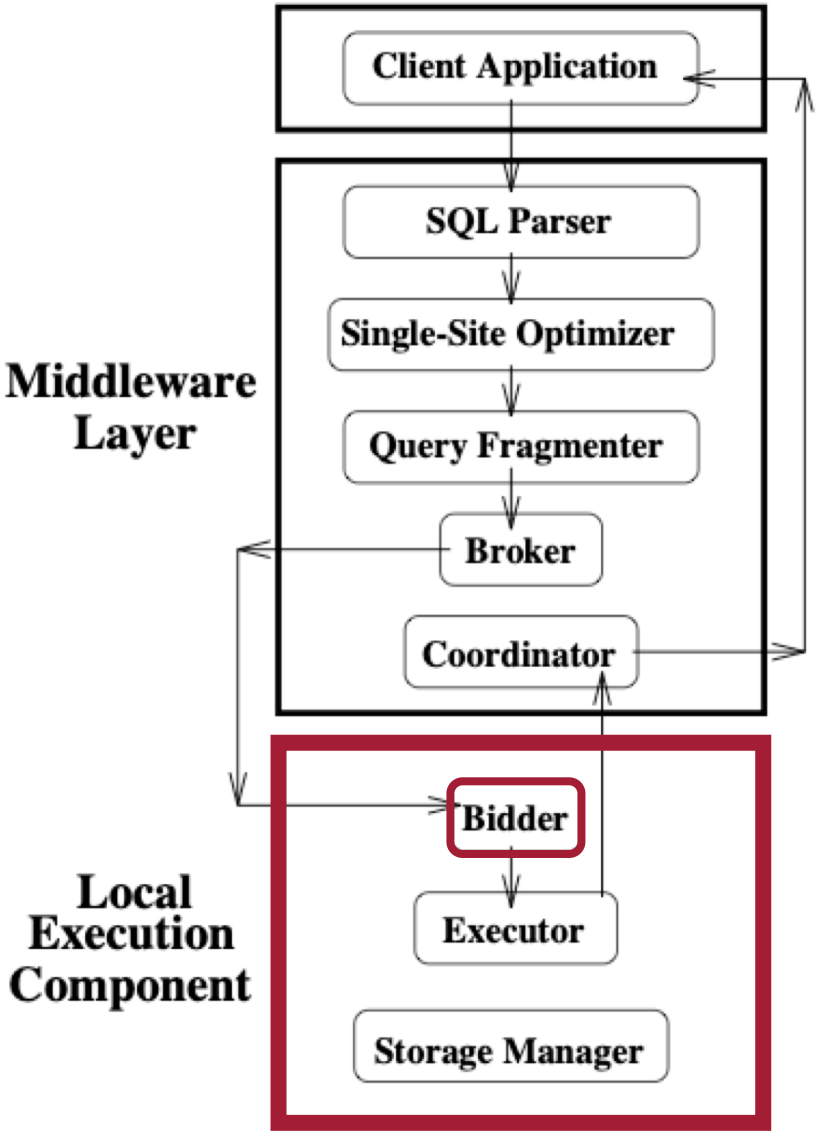
- **Parser**: request catalog information from name servers
- Conventional **query optimizer** produces a single-site query execution plan
- **Query fragmenter**: decomposes a single site plan into a fragmented query plan
- **Broker**: takes fragments and sends out bidding requests; decides which sites to accept/reject.



Mariposa Architecture

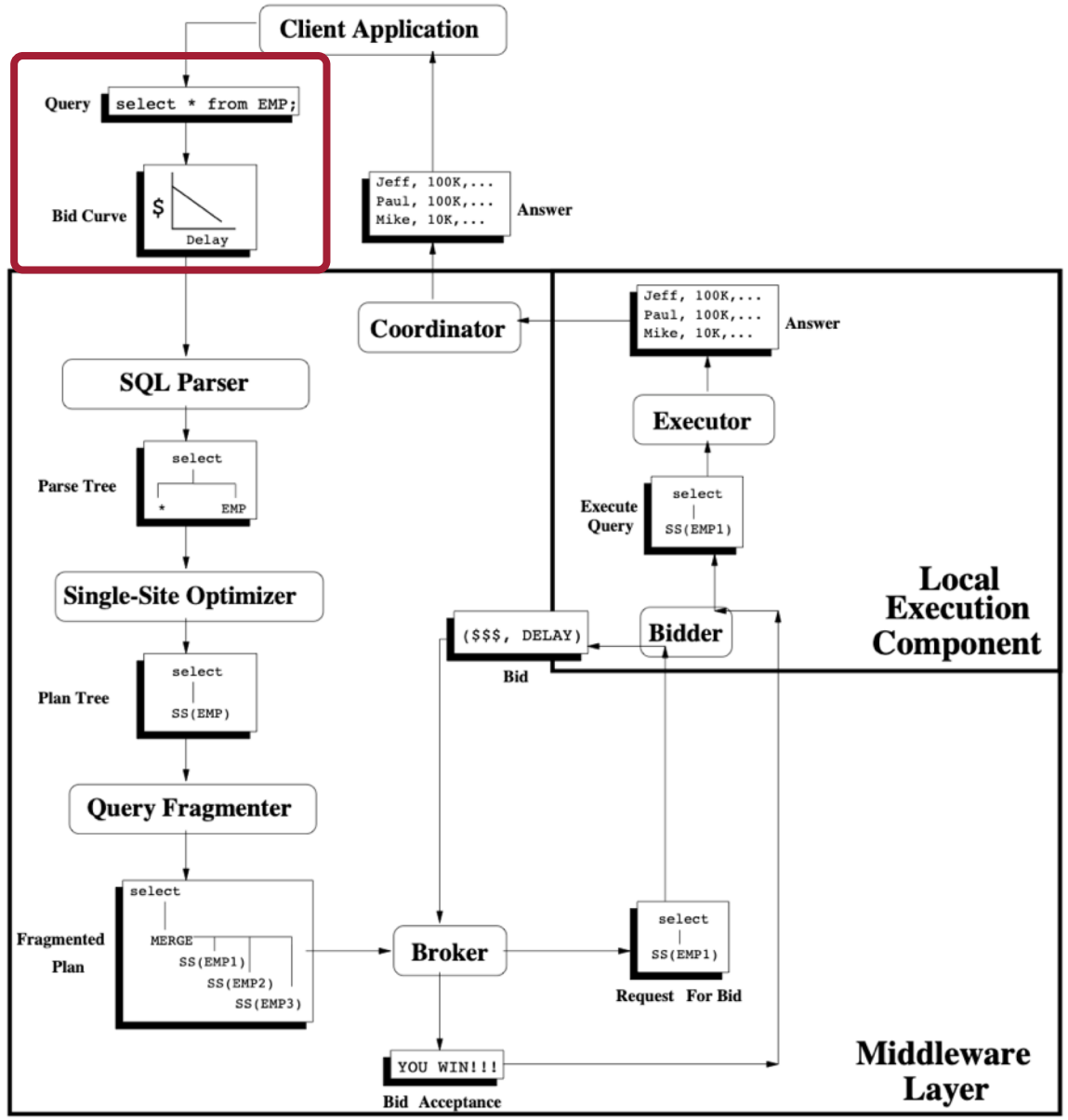
Local Execution Component

- **Bidder**: send bid price to the broker
- **Executor**: execute the query as in a conventional DBMS
- **Storage manager**: storing fragments, buying and selling fragments, splitting and coalescing fragments



Mariposa Architecture

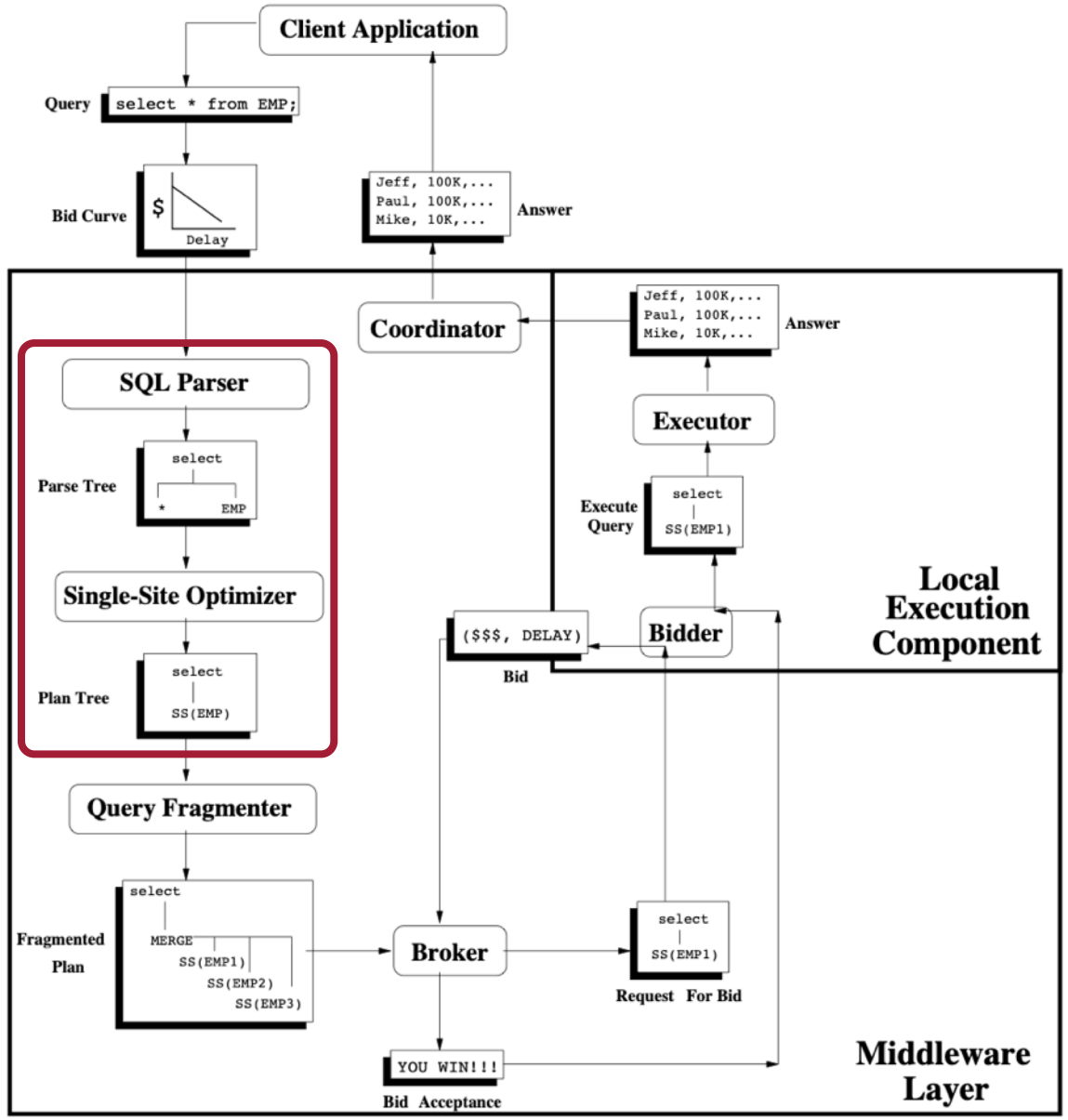
Client site picks a query **budget** expressed as a **bid curve**



Mariposa Architecture

Query parsing and single-site optimizer

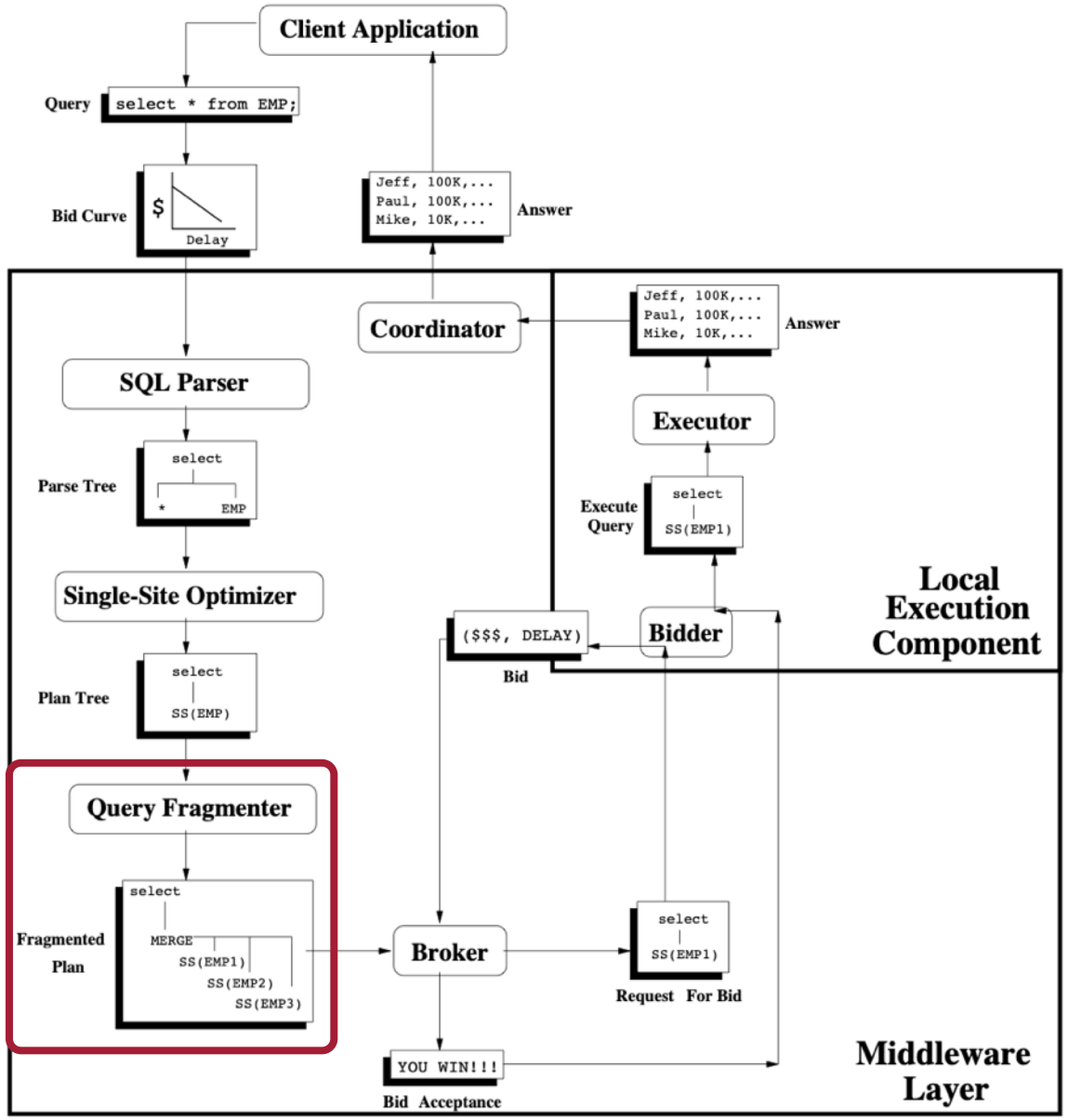
- Assume all fragments are merged and reside at a single server site



Mariposa Architecture

Query fragmenter

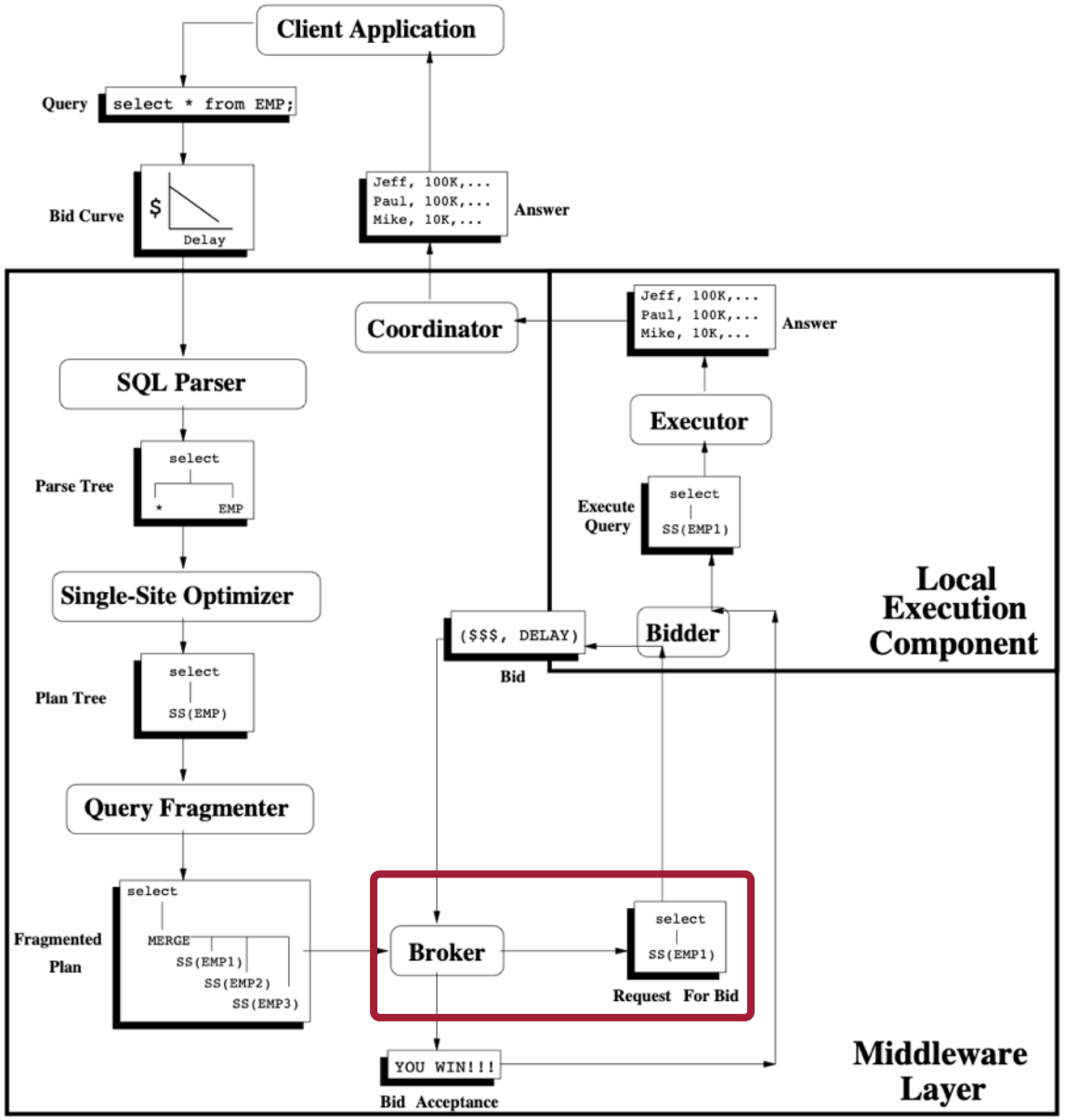
- Each table in FROM clause can be decomposed into fragments
- Fragments are partitions of tables (e.g., range, hash, or random)
- Group operations that can proceed in parallel into **query strides**. All subqueries in a stride must complete before the next stride starts



Mariposa Architecture

Broker sends bids requests

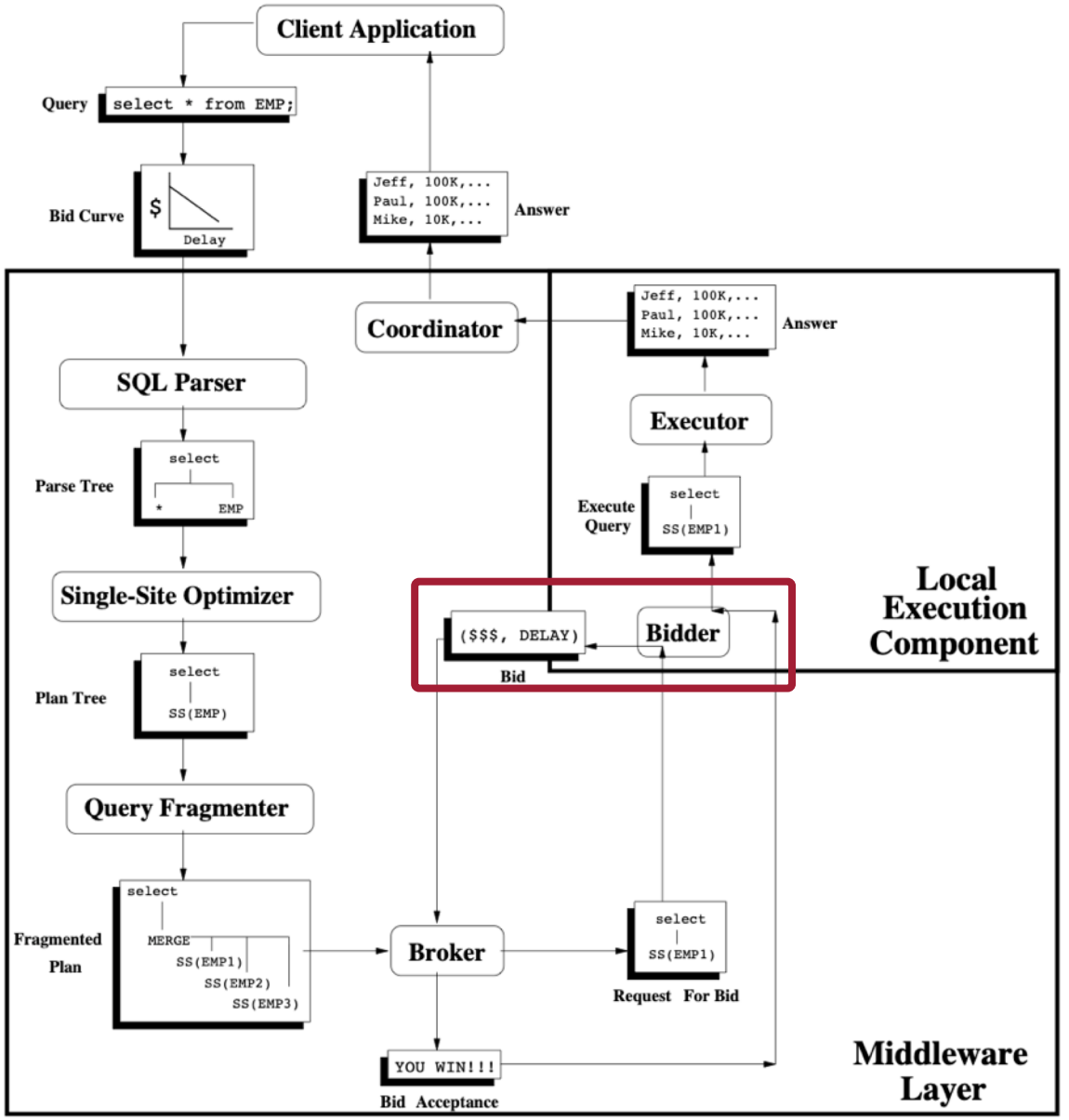
- Find processing site for each subquery (through advertisement) such that the cost and delay satisfy the budget (i.e., **bid curve**)
- **Bidding** vs. **purchase order**: For purchase order, simply send subquery to the site most likely to win the bid



Mariposa Architecture

Bidder

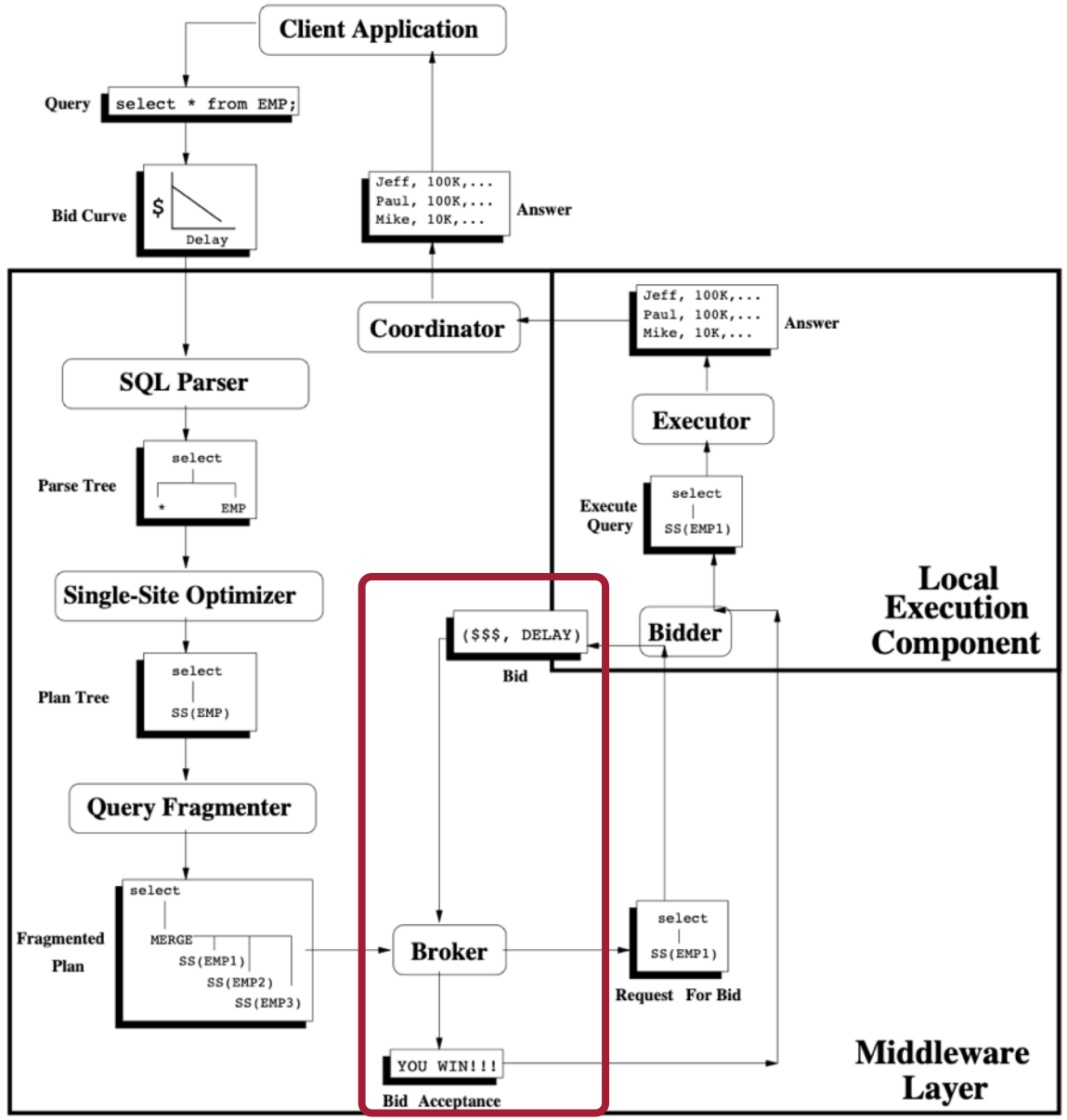
- A Bidder bids if
  1. It posseses the referenced objects (or 1 of the 2 objects for join)
  2. It has bid on a subquery whose answer is the referenced object
  3. It plans to load the object soon (e.g., object in host list)
- Actual bid depends on hardware and system load
- Send **cost** and **delay** back to broker



Mariposa Architecture

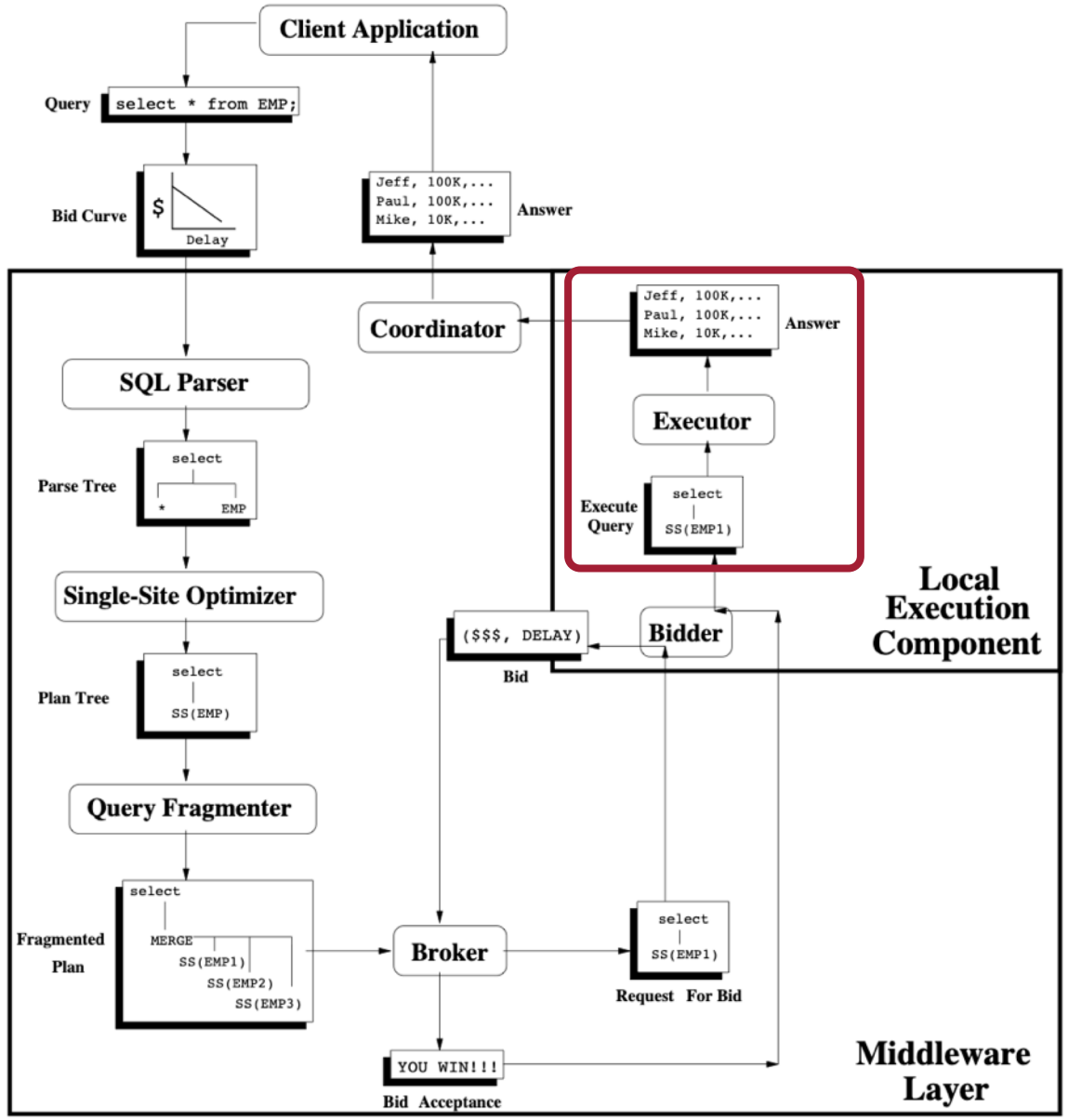
Broker picks sites

- Heuristic greedy algorithm:
  1. Find the set of sites with the smallest delay
  2. Make greedy substitutions of sites to reduce cost by increasing delay (start with the ones with greatest cost gradient)



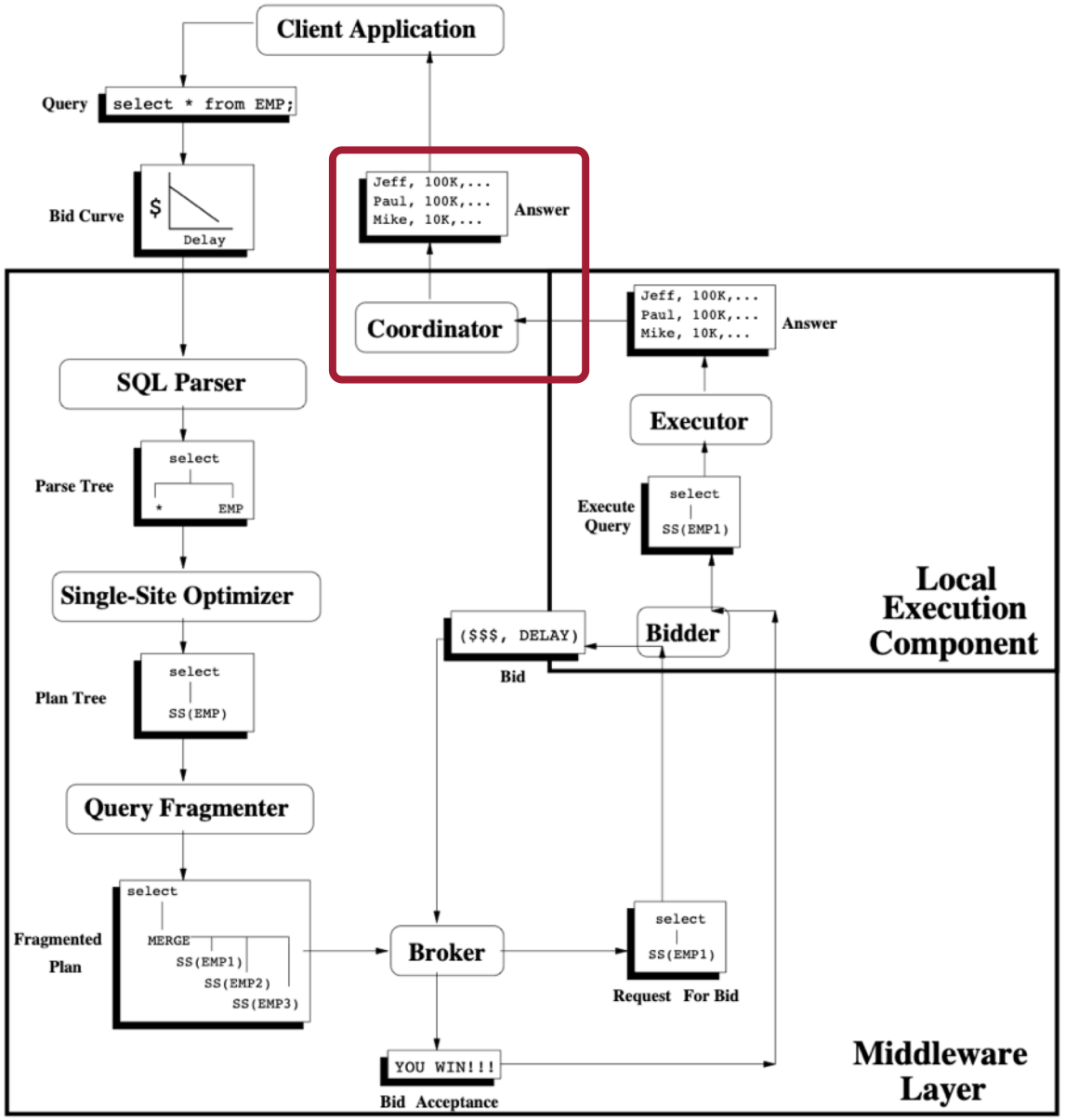
Mariposa Architecture

Local execution



Mariposa Architecture

Merge results from sites



Storage Management

Manage fragments to maximize profits in local execution component

Buying and selling fragments

- Each site tracks (size, revenue) for fragments
- Make buying/selling decision based on history (similar to cache replacement)

Splitting and coalescing

- Too few fragments hinders parallel execution
- Too many fragments lead to higher scheduling overhead
- Let the market pressures dictate the appropriate fragment size

Name Services

Decentralized name registration system

Each client/server has local name cache to resolve object names

Broker queries name server if a match is not found

Broker chooses name sever based on quality of service and cost (i.e., staleness)

Performance

Bidding overhead can be small if query execution takes a long time

		Steps					
		1	2	3	4	5	6
Elapsed time	Brokering	13.06	12.78	18.81	13.97	8.9	10.06
(s)	Total	449.30	477.74	403.61	428.82	394.3	384.04
	R1	1	1	1	1	3	3
Location of	R2	2	2	1	11	13	13
(site)	R3	13	3	3	3	3	3
	Site 1	97.6	97.6	95.5	97.2	102.3	0.0
Revenue	Site 2	2.7	2.7	3.5	1.9	1.9	1.9
(per query)	Site 3	177.9	177.9	177.9	177.9	165.3	267.7

Query performance in Mariposa improves over time

Q/A – Mariposa

Who needs a WAN database?

Used in commercial systems today?

- Cohera Corporation -> People Soft (2001) -> Oracle (2004)

Drawback of always using full name instead of common name?

Performance degradation if the query on R1, R2 and R3 runs on all the three locations?

What organizations would setup a database like this?

What if no servers bid on a query?

Security issues? Possible attacks?

Before Next Lecture

Please submit your proposal to the paper review website:

- https://wisc-cs764-f20.hotcrp.com

Submit review before next lecture

- Jeffrey Dean, Sanjay Ghemawat: MapReduce: simplified data processing on large clusters. Commun. ACM 2008.