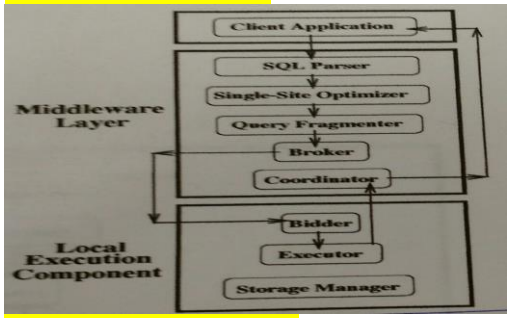
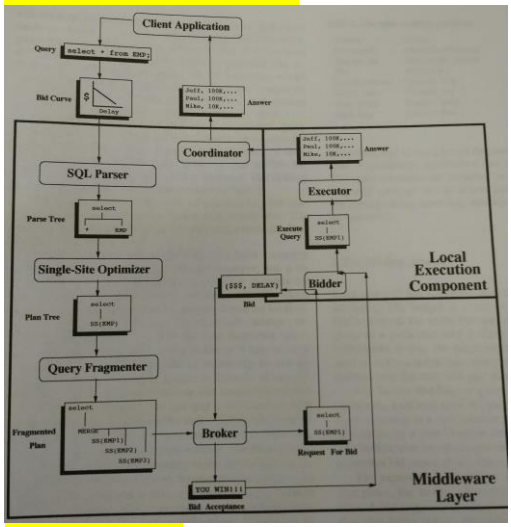
Best R* strategy – Bloom Joins || Optimizers model system performance for some subset of alternatives, taking into consideration a subset of the relevant factors – this introduces error || Distributed Query: SQL data manipulation statement that references tables at sites other than the query site – the site to which an application program is submitted. This site serves as the master site. Sites other than the master site which store a table referenced in the query are called apprentice sites || **Total cost:** W_{CPU}(# of instructions) + W_{IO}(# of IO) + W_{MSG}(# of MSG) + W_{BYT}(# of BYT) || EXPLAIN: describe the optimizers access plan. COLLECT: collect the values from internal counters FORCE OPTIMIZER: override the optimizer's choice of plans || W strategy is always better then F. In W, msg costs are very less of total (~5%) where as in F it is~80% || -M- merge scan join; -N- nested loop join || Ship outer to the inners' site, return the results to outers' site – outer will be small & results will be small & can use inners' indexes → BEST PLAN || DISTRIBUTED join is better then LOCAL join bcoz – availability and twice as much buffer space then local query || *Tables S and T @ Site 1 and 2 respectively, with 1 as query site.* Dynamically created temp index on inner: **1)** Scan and ship T to 1. **2)** Store and create temp index on T at 1. (No IO as T is already in buffer) **3)** Execute best plan for a local join at 1. SEMI JOIN: **1)** Sort both S and T on the join column to get S' & T'. **2)** Read S'.a @site 1, eliminating duplicates and send the results to site 2. **3)** @site 2, select the tuples that match S'.a to get T'' and ship T'' to 1. **4)** @1, merge-join S' and T''. BLOOM JOIN (BJ): **1)** Generate BfS @1. **2)** Send BfS to 2. **3)** Scan T @2, hashing T.b with same hash function as point 1. If bit hashed to is '1', send that tuple to 1 as stream T'. **4)** @1, join T' and S. || **BJ is better coz** bit tables used by BJ will be typically smaller than the join-column values transmitted by semi joins. Also BJ reduces the size of inner tables at early stage to save local costs. || **BJ is better then semi-joins coz** SJ incurs higher local processing costs to essentially perform second join at B's site, compared to simple scan of B in no particular order to do the filtering. || Shipping the entire inner table to join site & storing it there dominates the F, which incurs prohibitive per msg costs for each outer tuple even in high-speed networks.

MARIPOSA (MP) is a sol^{n} proposed to replace traditional cost based optimizers, as they do not scale well to a large number of processing sites. REASON – COLLECTION OF FAULTY ASSUMPTIONS: ***1)*** *Single administrative structure* (site selection for query fragments done by optimizers) ***2)*** *Static Data Allocation* (objects can't move quickly to change sites) ***3)*** *Uniformity* (all processors and network connections are the same & any joins can be done at any site). || MP PRINCIPLES: **1)** Scalability to large # of cooperating sites. **2)** Data Mobility **3)** No global synchronization **4)** Total local autonomy **5)** Easily configurable policies || MP BASICS: All MP clients & servers have an account with a network bank. User allocates a budget in the currency of this bank to each query. Goal of query processing system is to solve the query within allotted budget by contracting with different MP sites to perform portion of the query. A **broker** administers each query. MP provides powerful mechanisms for specifying the behavior of each site. Sites must decide which objects to buy & sell & which queries to bid on. **Bidder** and **storage manager** makes these decisions. ||

MP ARCHITECTURE



MP COMMUNICATION



Bid Curve B(t): how much user is willing to pay. **Parser:** parse the incoming query, performing name resolution and authorization. **SSO:** Take query in form of parse tree from parser and prepare single-site QEP. **Fragmenter:** Use the location info previously obtained from the name server, to decompose the single site plan into a fragmented query plan (FQP). Group the operations that can proceed in parallel into query strides. **Broker:** Take FQP & send out requests for bids to various sites. Notify the winning site by sending bid acceptance. **Coordinator:** coordinate the execution of resulting query strides. Assemble the partial results and return the final answers to the user processes. ||