DRYAD: DISTRIBUTED DATA-PARALLEL PROGRAMS FROM SEQUENTIAL BUILDING BLOCKS

MICHAEL ISARD, MIHAI BUDIU, YUAN YU, ANDREW BIRRELL, DENNIS FETTERLY

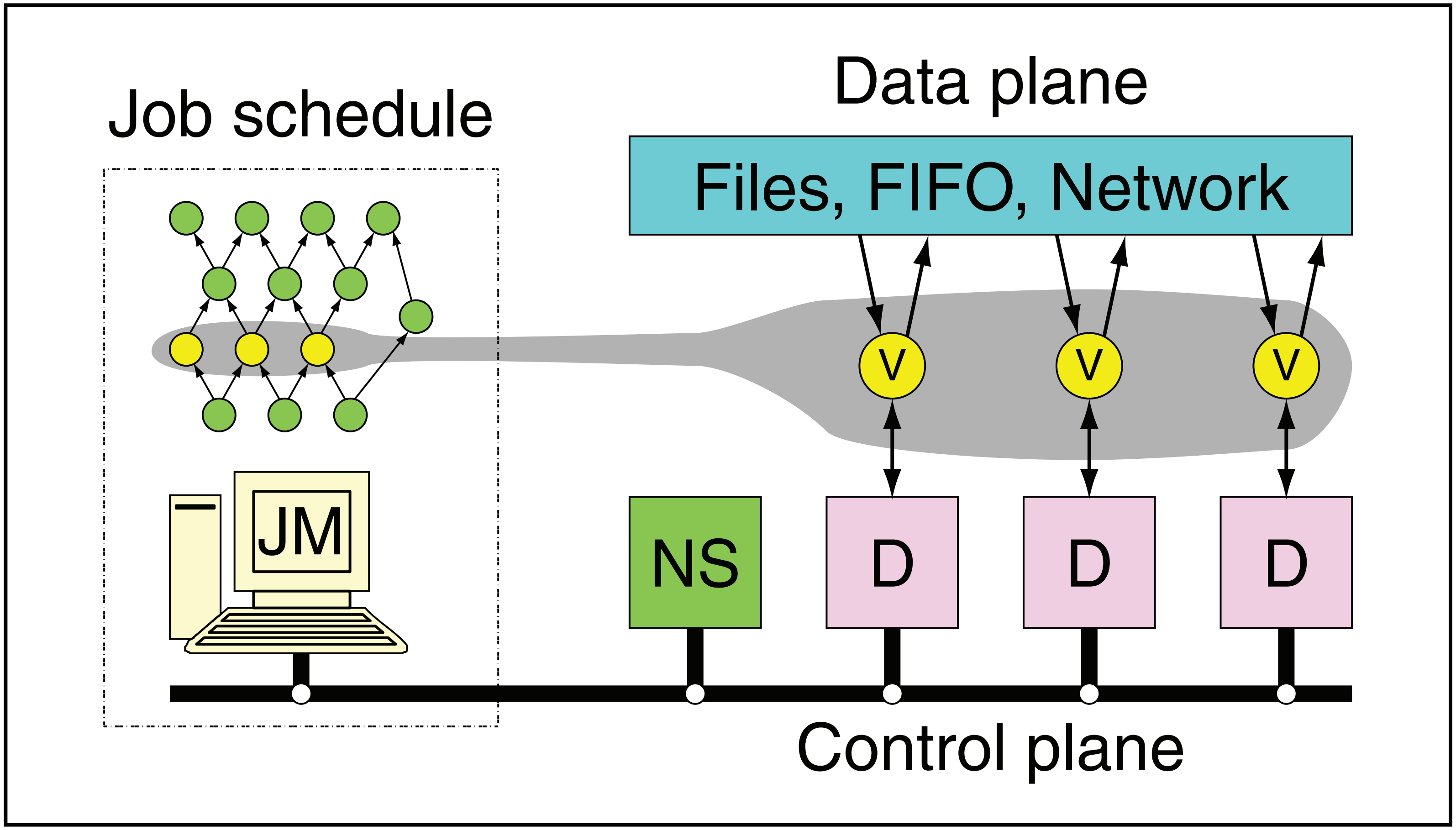
DRYAD GOALS

- **Research question: How to make it easier for programmers to express parallel and distributed program?**
- **General purpose execution environment for distributed, data-parallel applications**
  - Focuses on throughput, not latency
  - Assumes secure environment, such as a private data center
- **Automatic scheduling, distribution of data and resources, fault tolerance**

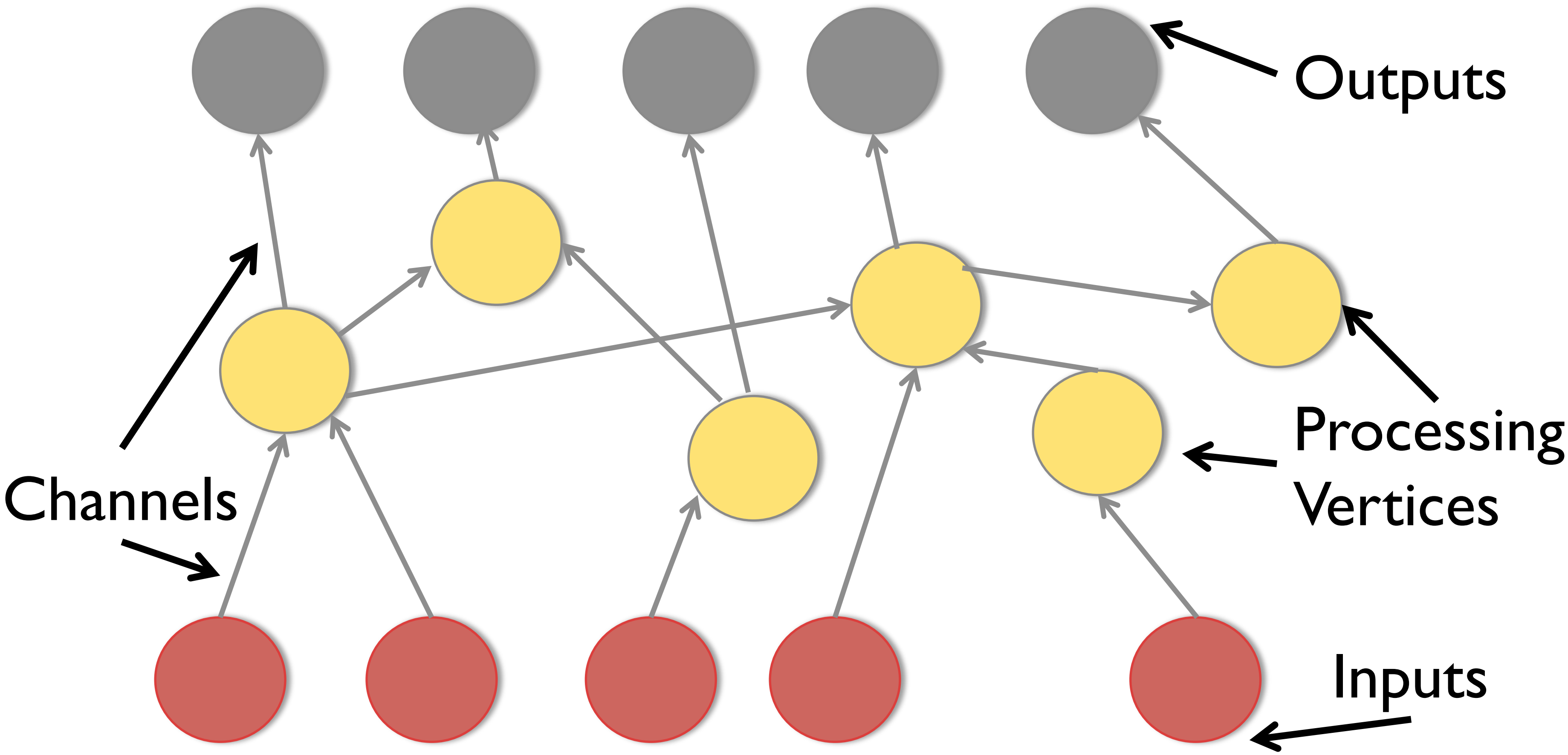
DRYAD

- Dryad is a middleware abstraction that runs programs that are represented as distributed execution graphs
- Dryad receives arbitrary graphs (DAGs) from users/ programmers
- Dryad provides mechanisms for allocating resources, scheduling computations, fault-tolerance

DRYAD RUNTIME



A DRYAD JOB: DAG

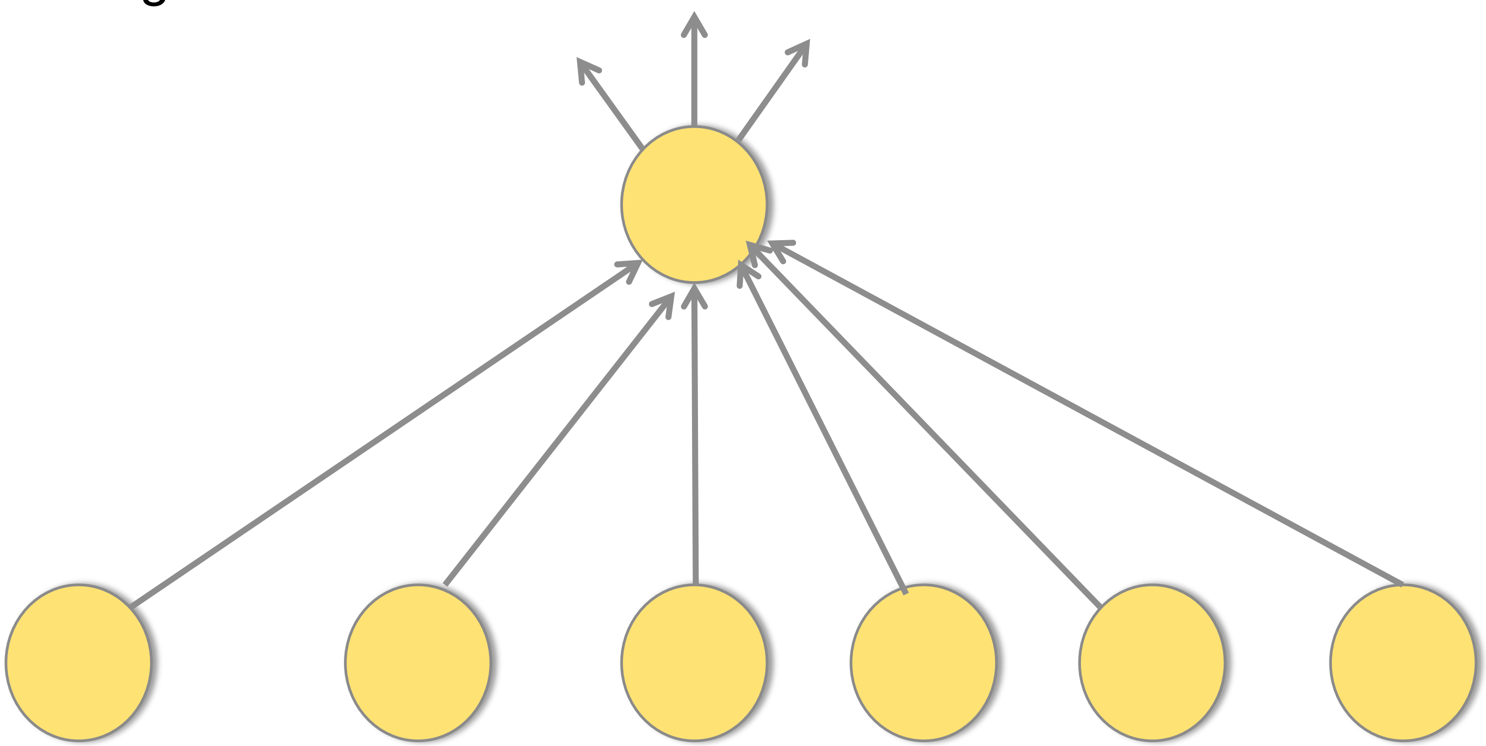


WHY A DAG?

- **Natural "most general" design point, cycles are problematic**
- **DAG supports full relational algebra**
  - Multiple inputs and outputs of different types from the same vertex
  - More general than MR, or defined special cases, no semantics included in the scheduler (just vertices to schedule)

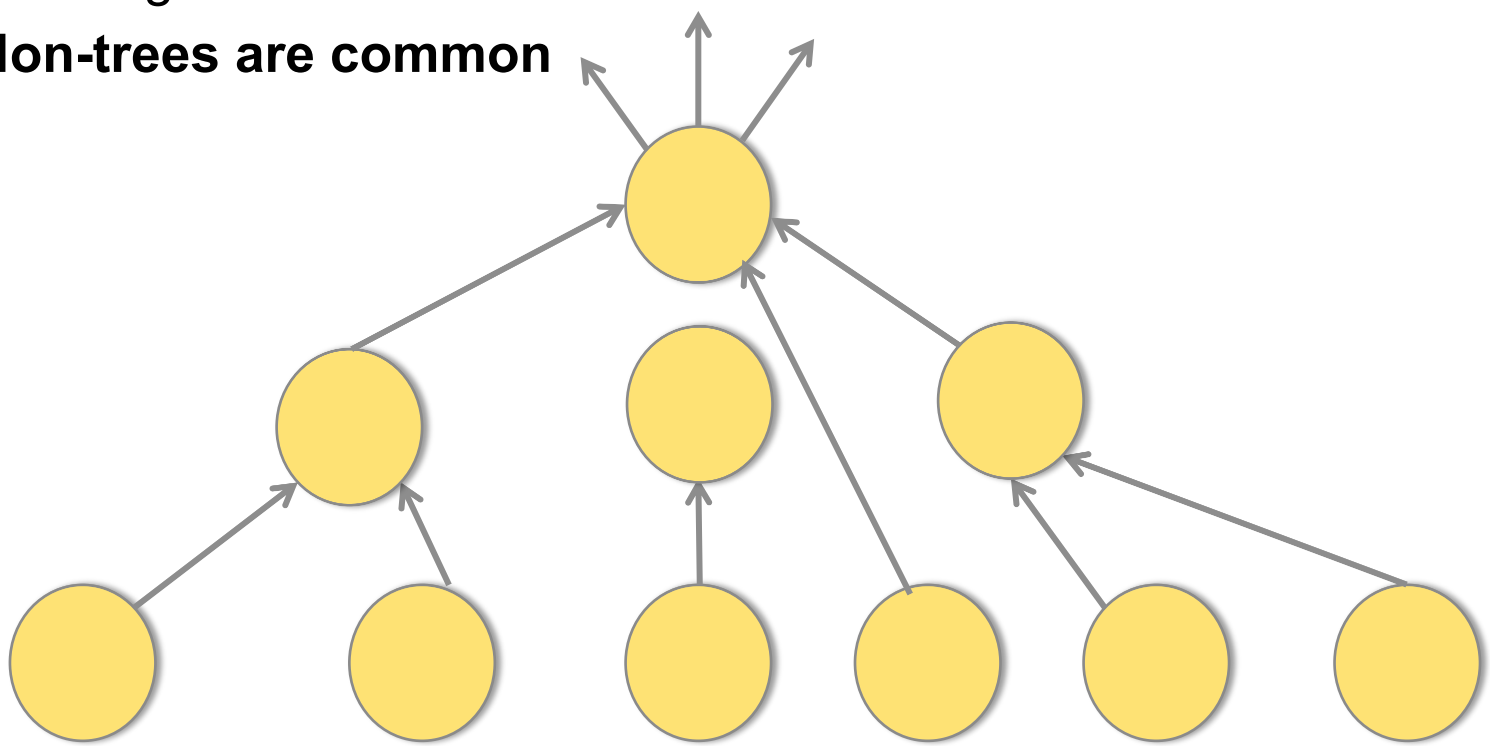
WHY A GENERAL DAG?

- **Uniform operations aren't really uniform**
  - e.g., SQL queries after dynamic optimization could look irregular.



WHY A GENERAL DAG?

- **Uniform operations aren't really uniform**
  - e.g., SQL queries after dynamic optimization could look irregular.
- **Non-trees are common**



WHY NO CYCLE?

- **No cycles makes scheduling easy:**
  - vertex can execute once all its inputs are ready
  - no deadlock
- **Fault tolerance is easy**

DYNAMIC REFINEMENT OF GRAPH

- **Application passes initial graph at start**
  - Gets callbacks on interesting events
- **Graph can be modified with some restrictions**
  - The job scheduler doesn’t itself do these modifications
  - The callbacks go to the application allowing such modifications
  - So this dynamic refinement is really an extension to Dryad

STAGE MANAGER

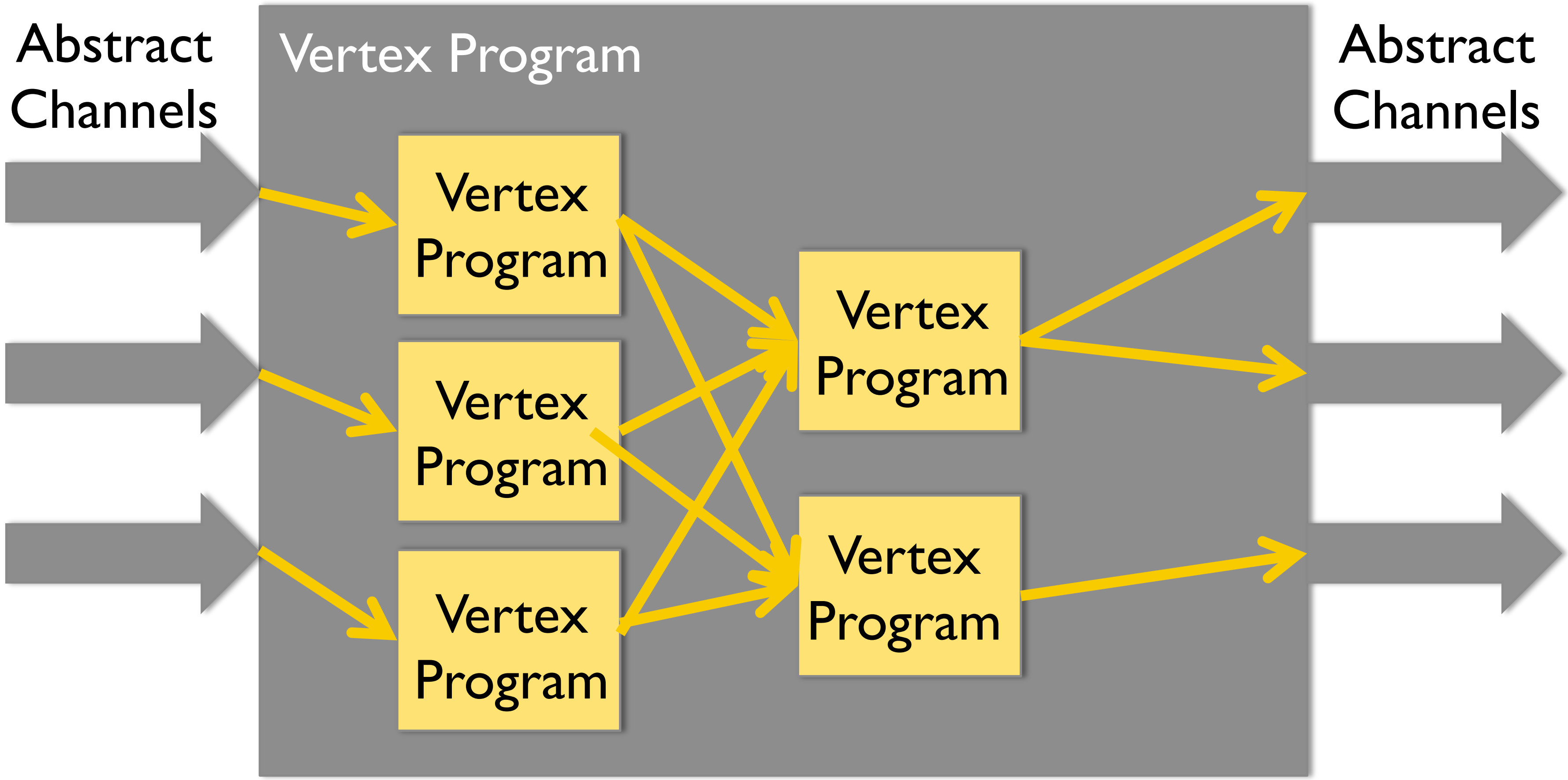
- **Every vertex has a stage manager:**
  - manages a bunch of vertices, generally grouped by function
- **A place where execution statistics are gathered and callbacks are received**
  - request re-executions of failed tasks
- **A natural place for building models for tasks executions**

CONNECTION MANAGER

- **We can overlay a graph on the stage managers**
- **Any pair of stages can be linked**
- **Gets callbacks on events in upstream stage:**
  - E.g., when vertices finish, new vertices get added
- **Most dynamic modifications happen here**

VERTEX PROGRAM

VERTEX PROGRAM



SOME CASE STUDIES

- **SkyServer DB Query**
- **Query Histogram Computation**

SKYSERVER DB QUERY

- **3-way join to find gravitational lens effect**
- **Table U: (objId, color) 11.8GB**
- **Table N: (objId, neighnorId) 41.8GB**
- **Find neighboring stars with similar colors:**
  - Join U and N to find T
    - T = U.color, N.neighborId where U.objId = N.objId
  - Join U and T to find, U.objId
    - U.objId where U.objId = T.neighborid and U.color = T.color

SKYSERVER DB QUERY

- **Manually coded the SQL plan in Dryad**

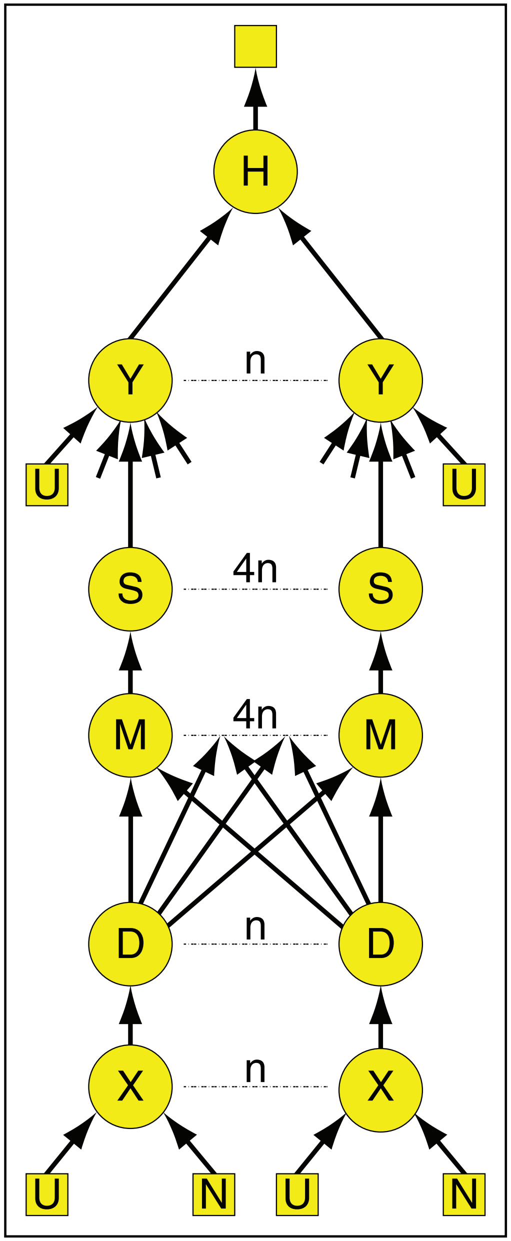
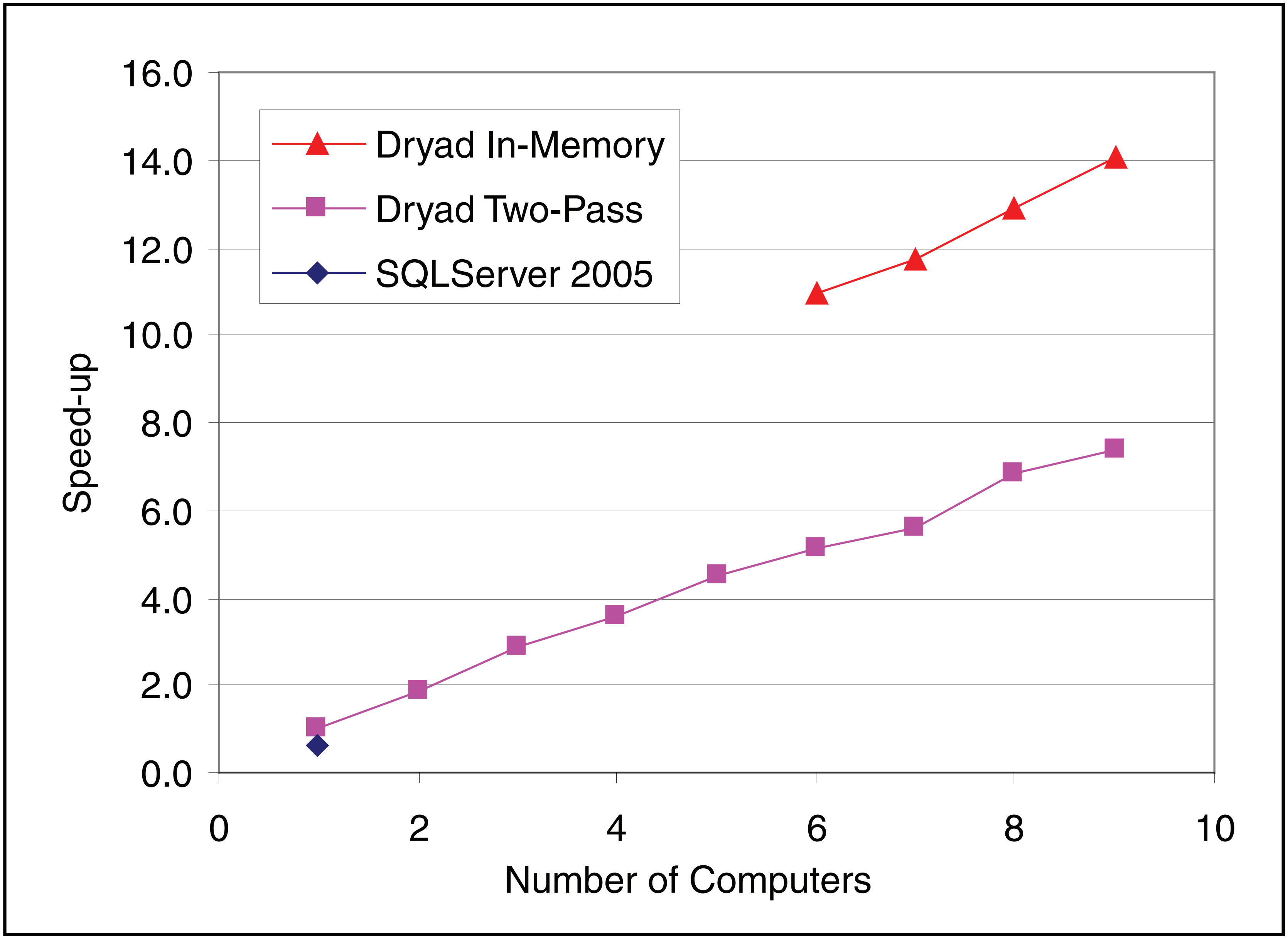


Figure 2: The communication graph for an SQL query. Details are in Section 2.1



16.0
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
Speed-up
0
2
4
6
8
10
Number of Computers
Dryad In-Memory
Dryad Two-Pass
SQLServer 2005

Optimizations done do not need any code changes, only graph manipulations!

QUERY HISTOGRAM COMPUTATION

- Input: Log file (n partitions)
- Extract queries from log partitions
- Re-partition by hash of query (k buckets)
- Compute histogram within each bucket

HISTOGRAM COMPUTATION: NAÏVE TOPOLOGY

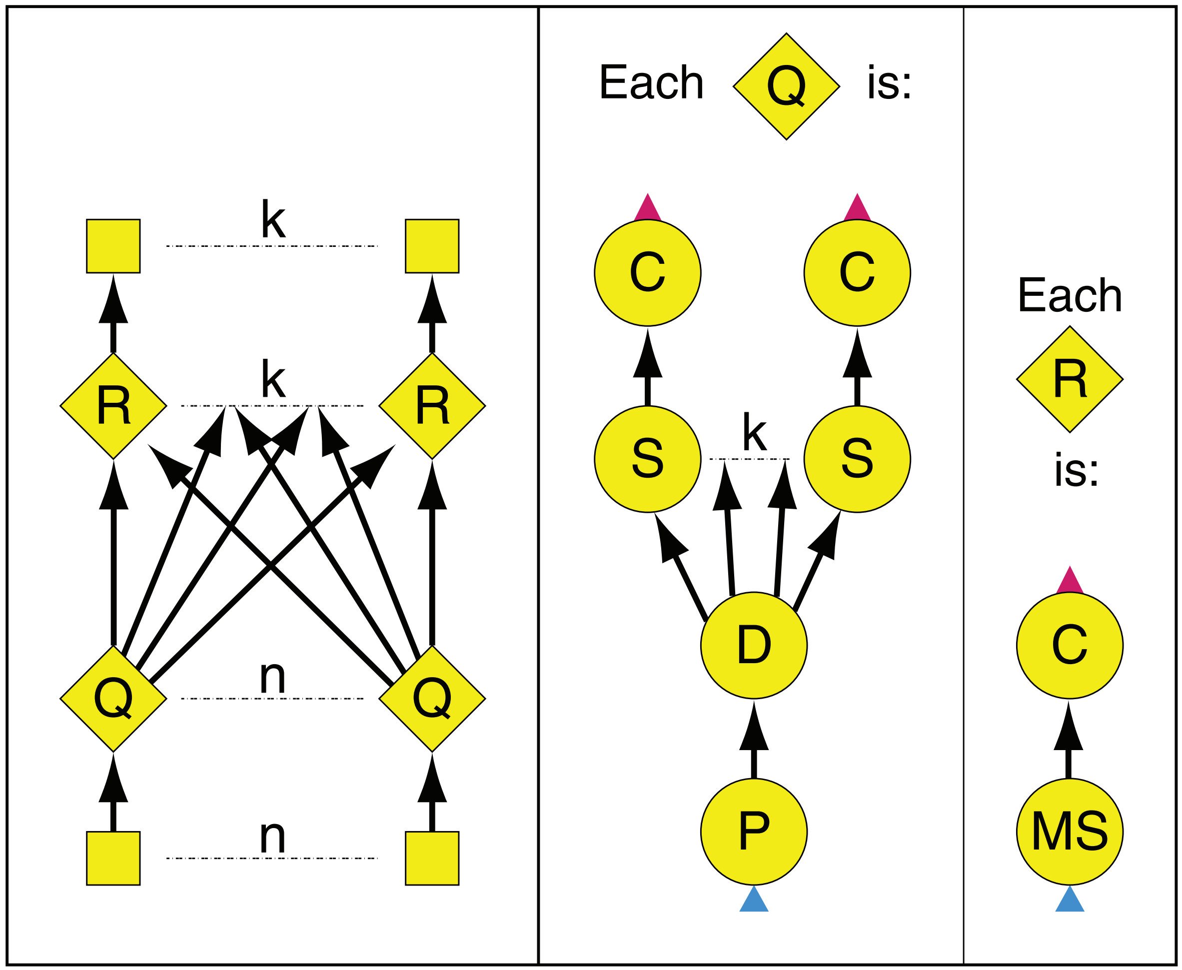
P: Parse lines

D: Hash Distribute

S: Quicksort

C: Count occurrences

MS: Merge Sort



HISTOGRAM COMPUTATION: EFFICIENT TOPOLOGY

P: Parse lines

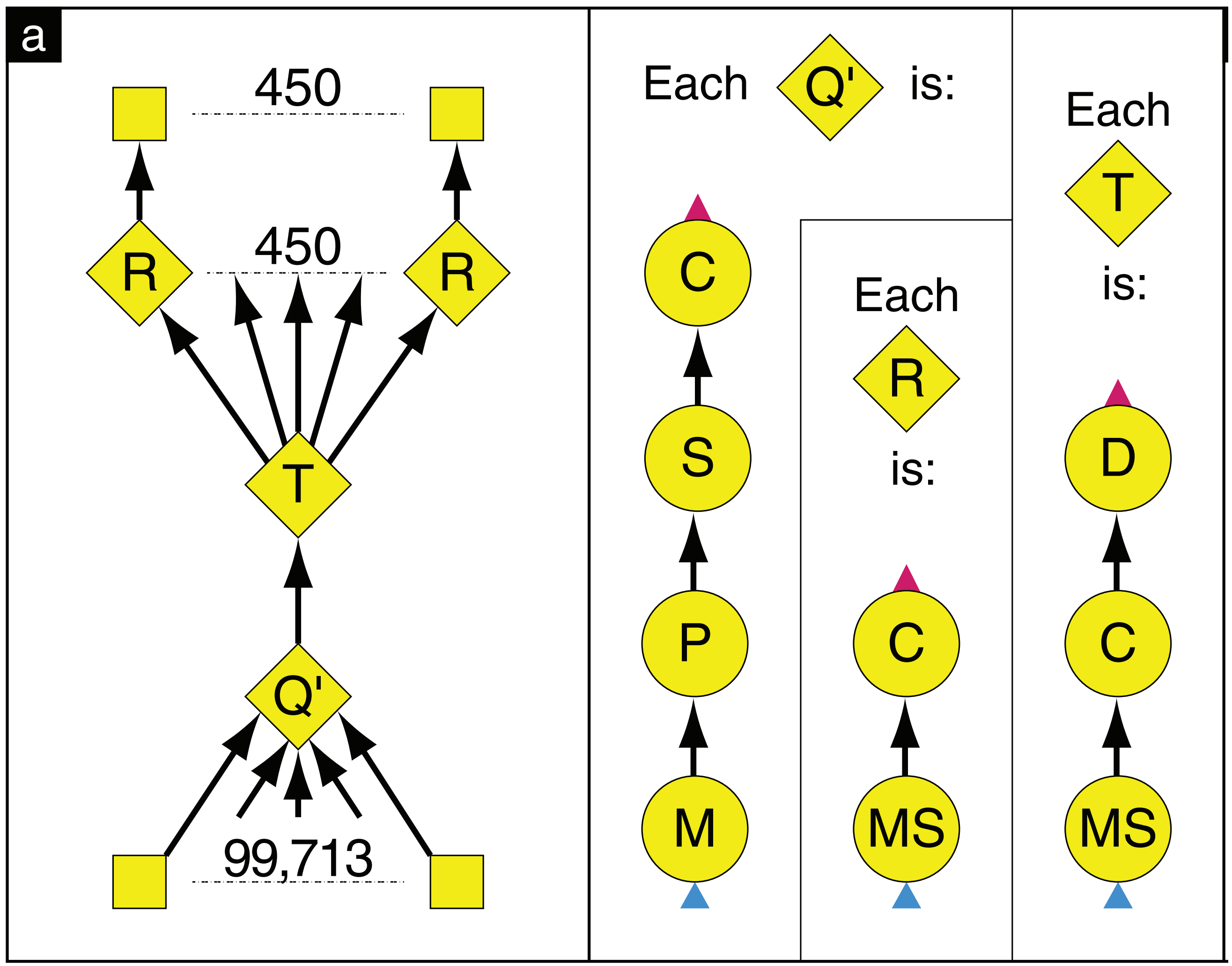
D: Hash Distribute

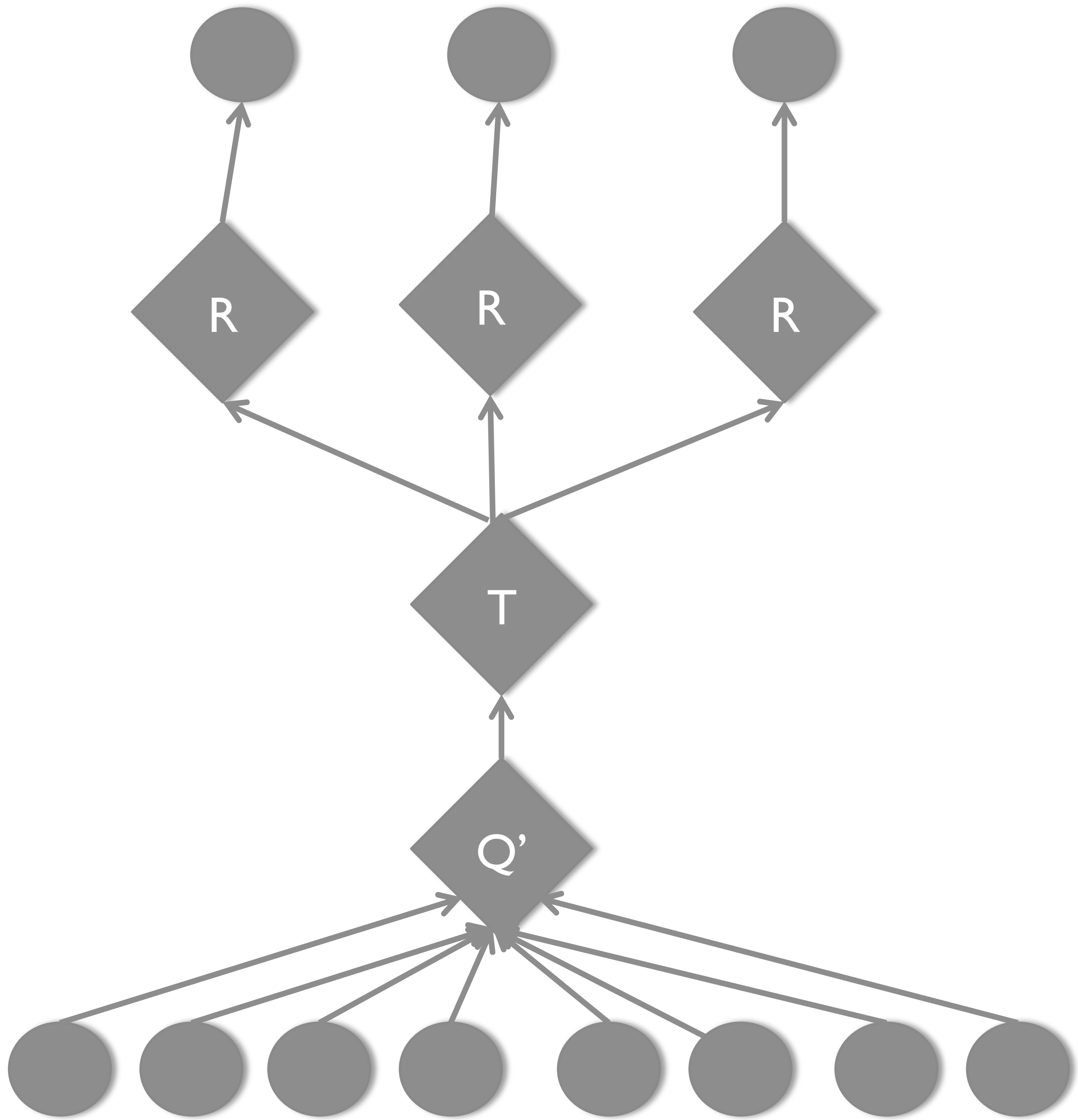
S: Quicksort

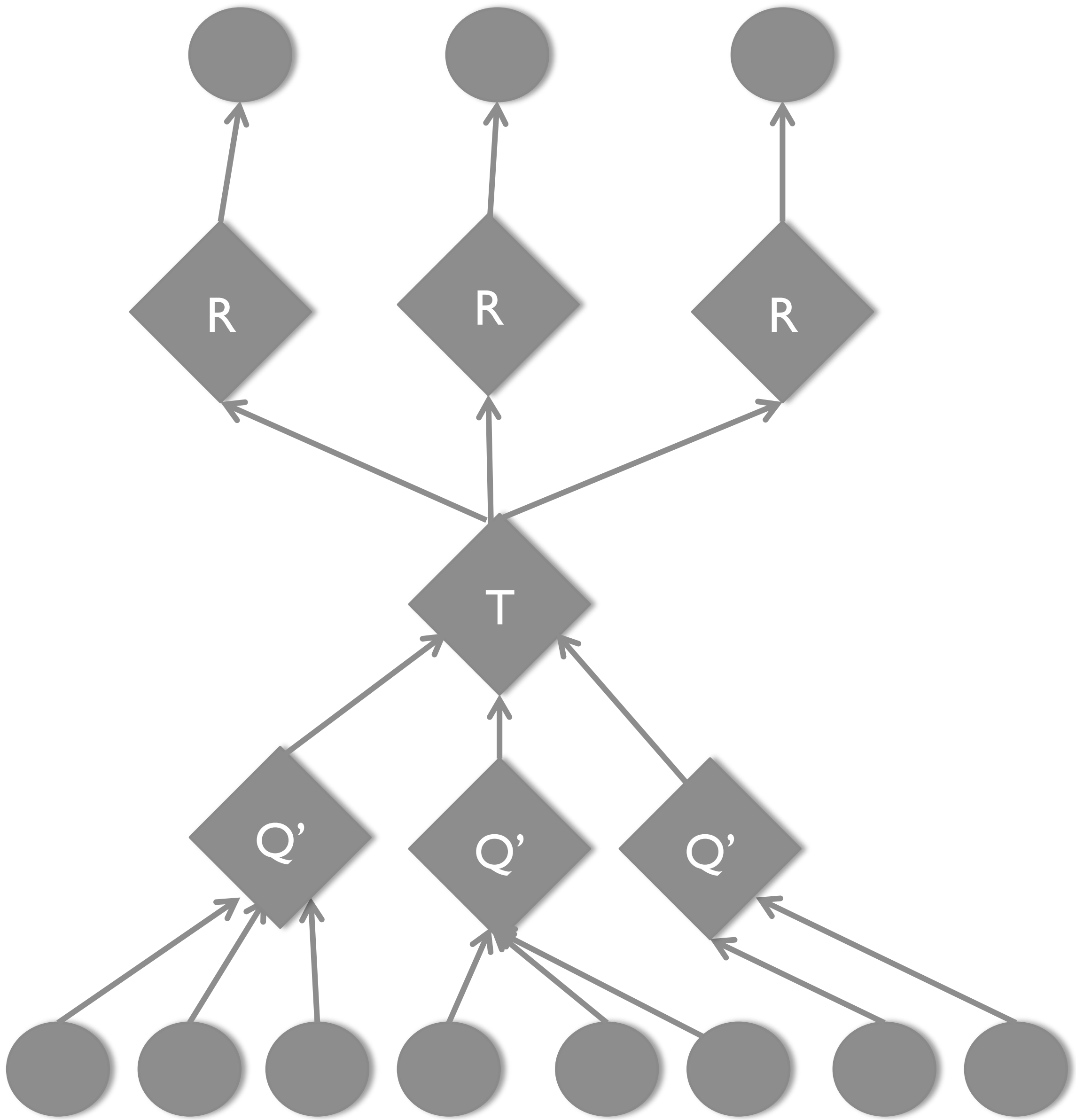
C: Count occurrences

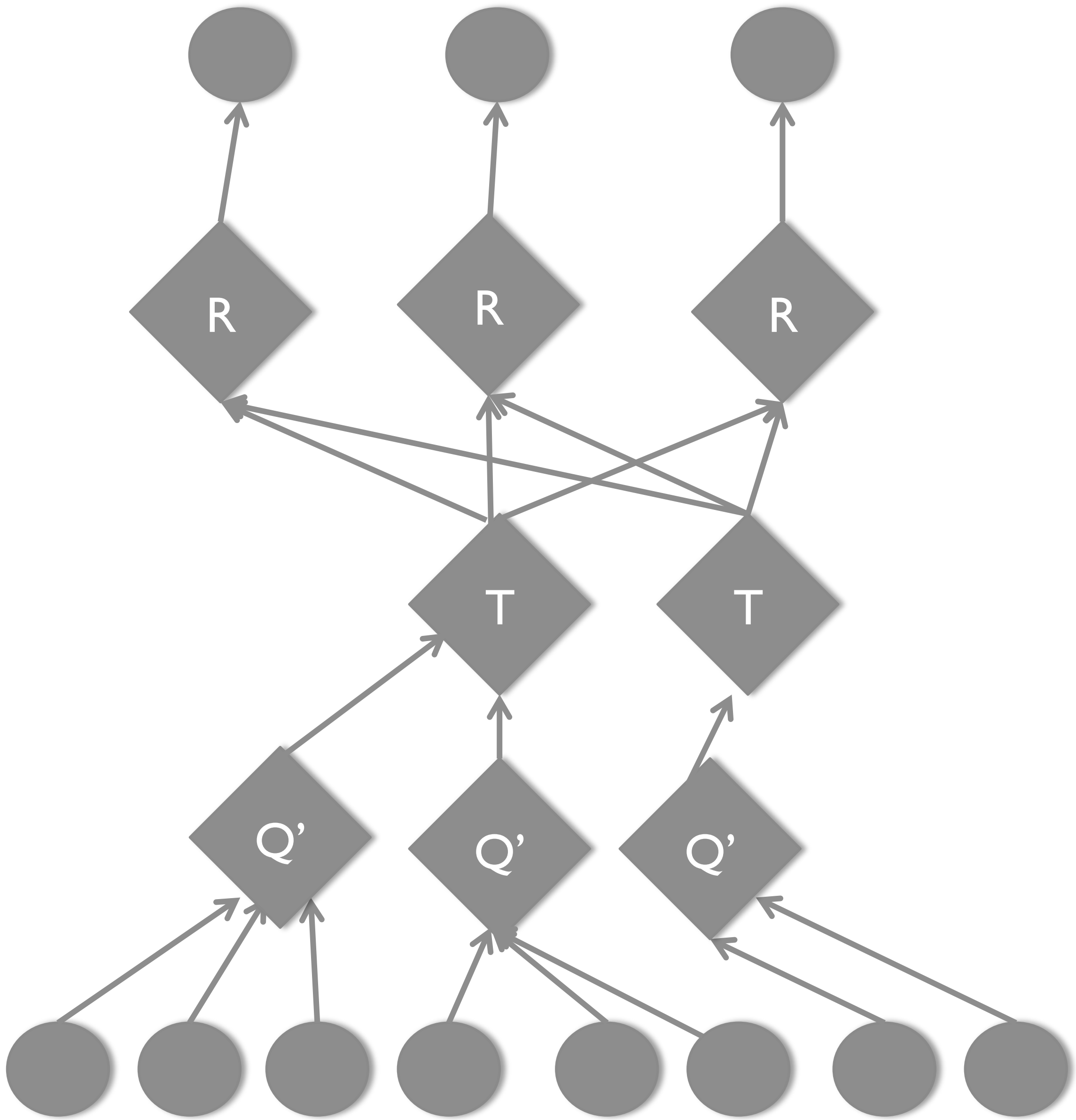
MS: Merge Sort

M: Non-deterministic Merge

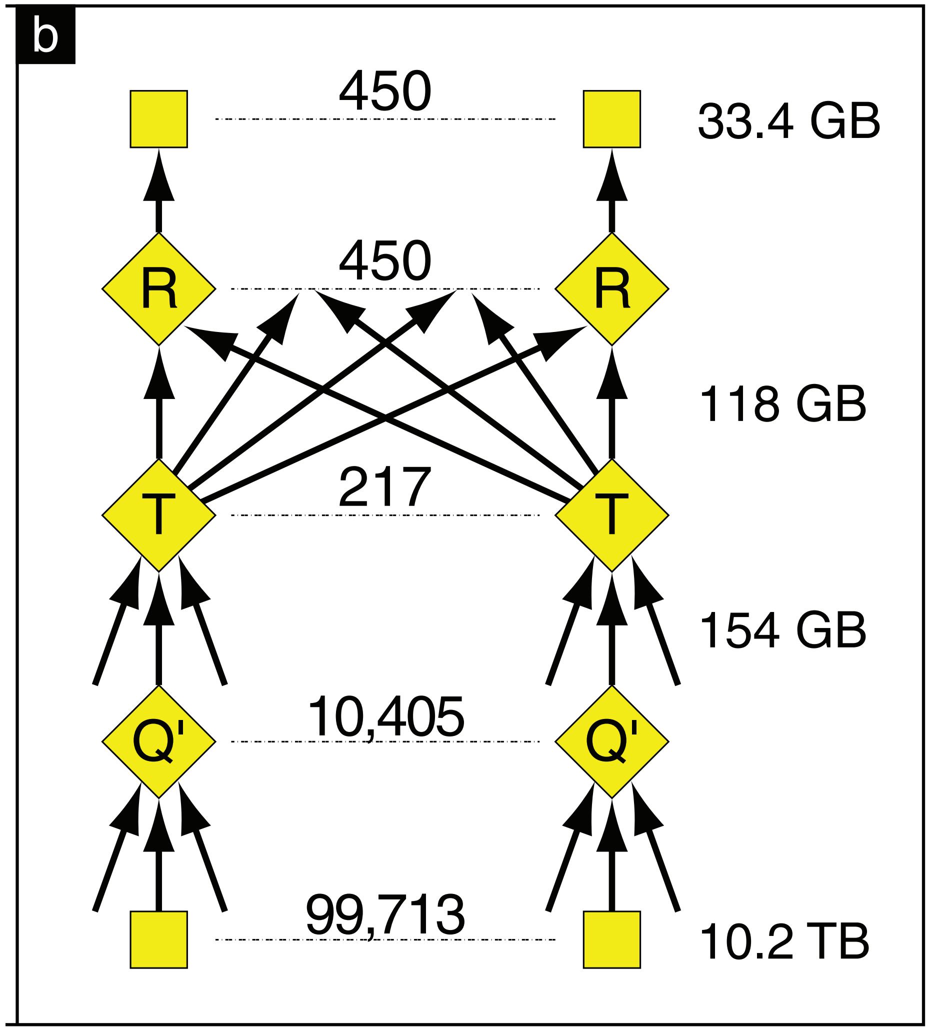



R
R
R
T
Q'


R
R
R
T
Q'
Q'
Q'


R
R
R
T
T
Q'
Q'
Q'

FINAL HISTOGRAM REFINEMENT



OPTIMIZING DRYAD APPLICATIONS

- **Application code is NOT modified**
  - Only graph manipulations
  - Users need to provide the vertex programs and the initial graph
  - Then the system optimizes it further, statically and dynamically

SMALL VS LARGE CLUSTERS

- **Small private clusters**
  - Few failures, known resources
  - ✓ Can use these sophisticated Dryad features
- **Large public clusters**
  - Unknown resources, failures
  - ➢ May not be able to use lot of the graph manipulation features as much

HIGHER-LEVEL PROGRAMMING MODELS

- **SSIS:**
  - SQLServer workflow engine, distributed
- **Simplified SQL:**
  - Perl with a few SQL like operations
- **DryadLINQ**
  - Relational queries integrated in C#
  - a front end for Dryad jobs