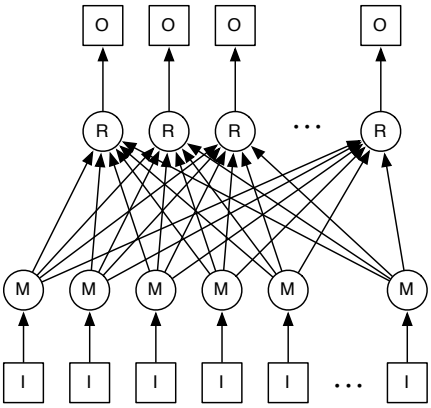
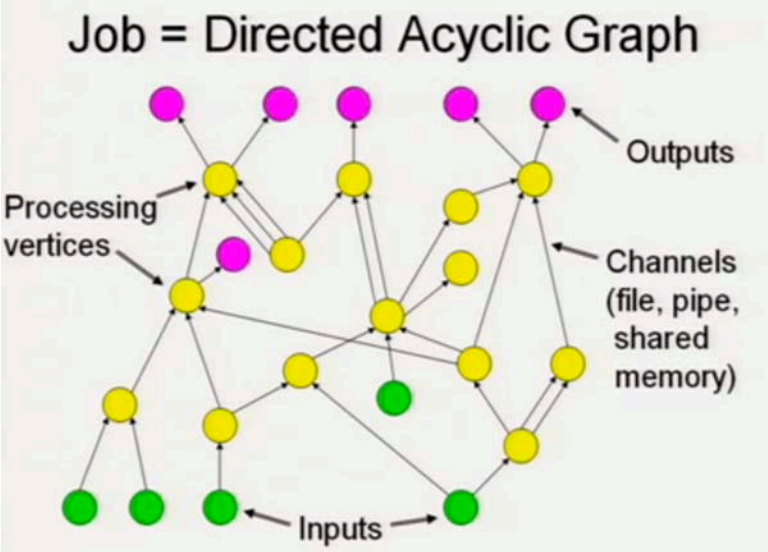
Dryad and DryadLINQ



MapReduce is great, but it lacks flexibility in the structure of computation that can be represented. One way to visualize MapReduce computations is as a graph structure (a DAG).

If you can pigeonhole your application into this structure, it will run as a MapReduce job. But, not all applications naturally fit this structure.

Dryad generalizes MapReduce to arbitrary DAGs.



Channels can be nearly anything that represents a sequence of typed items:

- temporary on-disk files
- TCP pipes
- shared memory FIFO

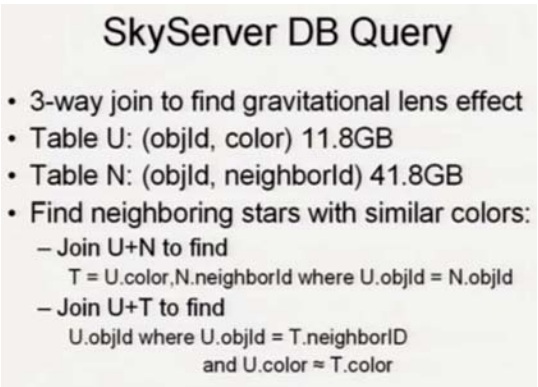
DAG is general, can be used for example to represent any SQL query (DAG supports the full relational algebra)

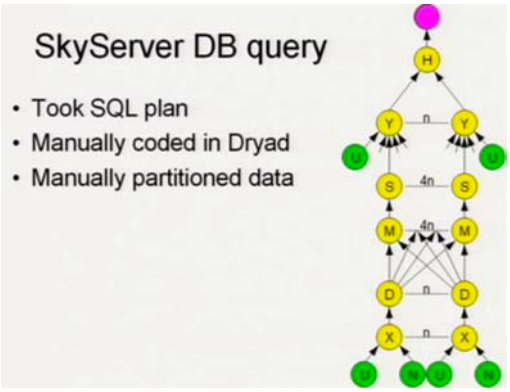
- uniform graphs become non-uniform after optimization

Very similar scheduling, fault tolerance story as mapreduce

- vertices are stateless, deterministic computations
- no cycles means that after failure, can just re-run a vertex. (if the vertex's inputs are lost, then rerun upstream vertices, transitively.)

Example:





Cool side-effect of using Dryad: optimizations can be done on the graph structure by Dryad, without understanding the semantics of the application itself.

DryadLINQ

Think of Dryad as middleware – most programmers don't want to think about manually constructing graphs. Instead, programmers will write in some higher-level language, and have their programs compiled down into Dryad graphs. Multiple higher-level "front ends" have been built by MSFT:

- SSIS – SQLServer workflow engine
- Perl + SQL
- DryadLINQ

LINQ is a way to integrate SQL-like relational queries into a C# program.