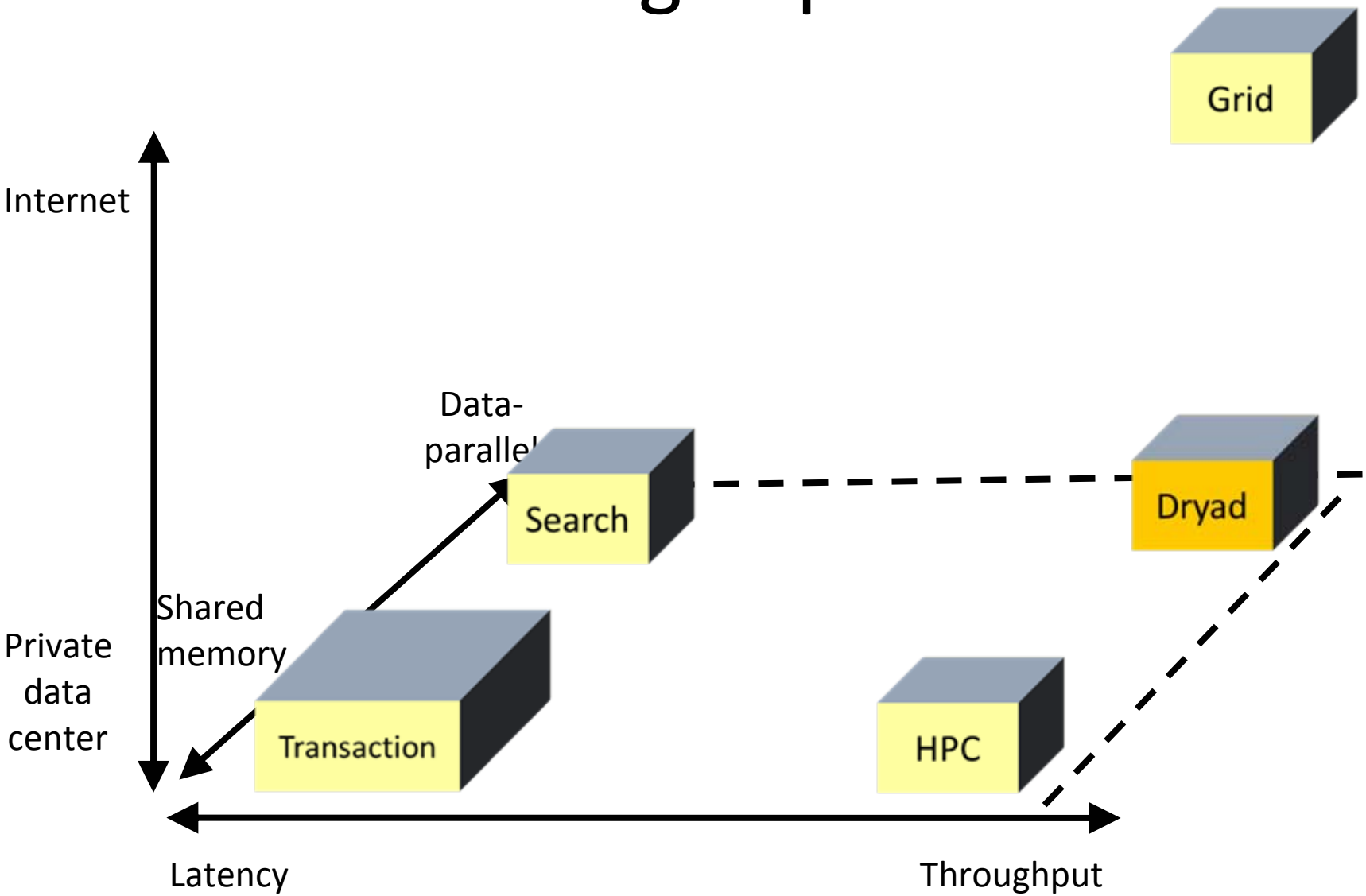
Cluster Computing with DryadLINQ

Mihai Budiu
Microsoft Research Silicon Valley

IEEE Cloud Computing Conference
July 19, 2008

Goal

Design Space

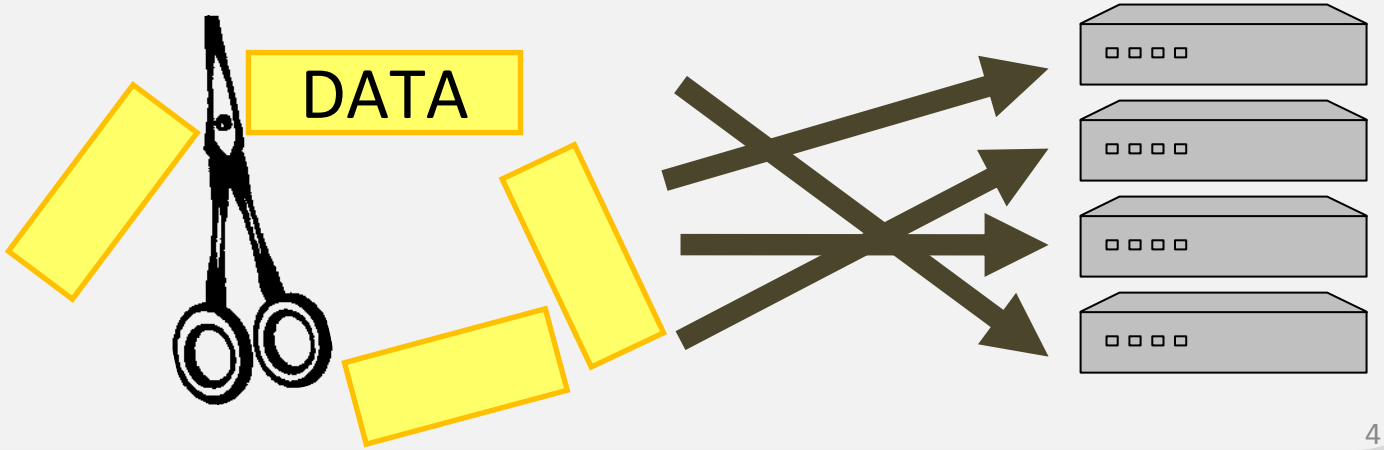


Data Partitioning

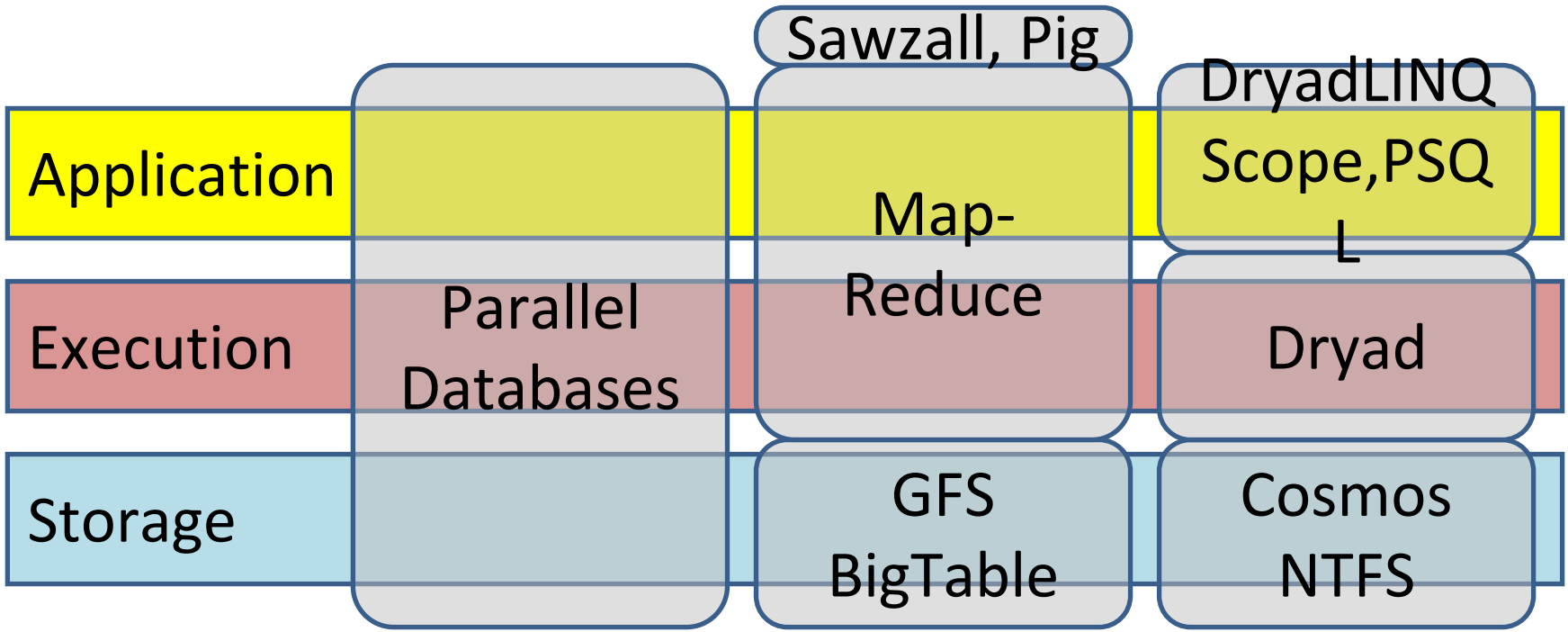
DATA

RAM

DATA



Data-Parallel Computation



Outline

- Introduction
- Dryad
- DryadLINQ
- Applications

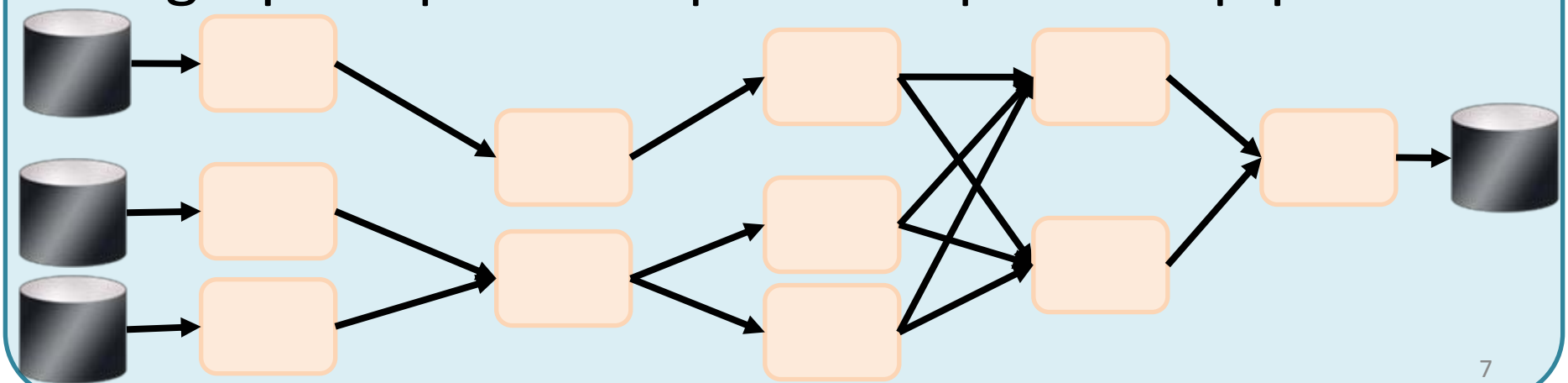
2-D Piping

- Unix Pipes: 1-D

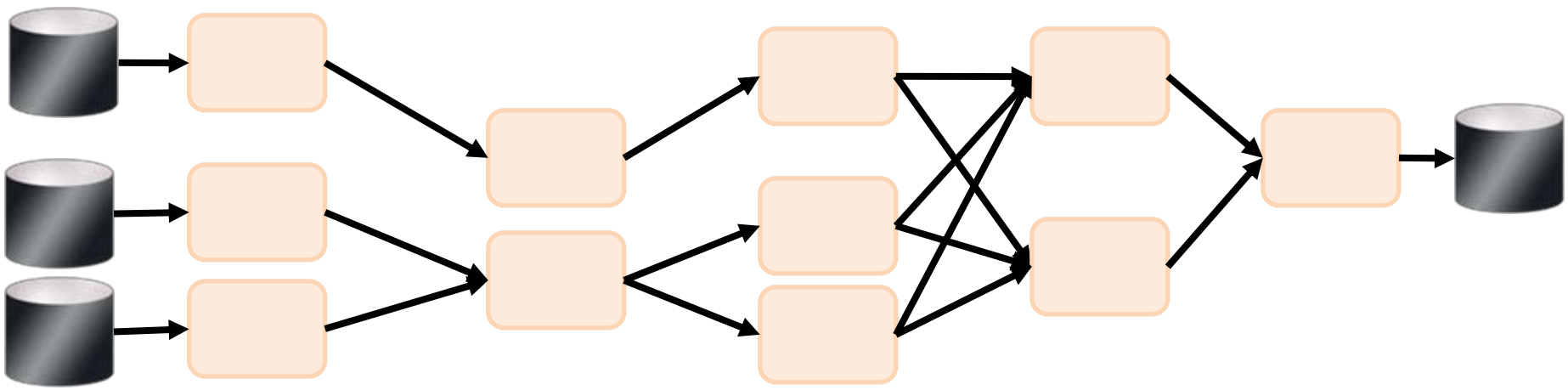
grep | sed | sort | awk | perl

- Dryad: 2-D

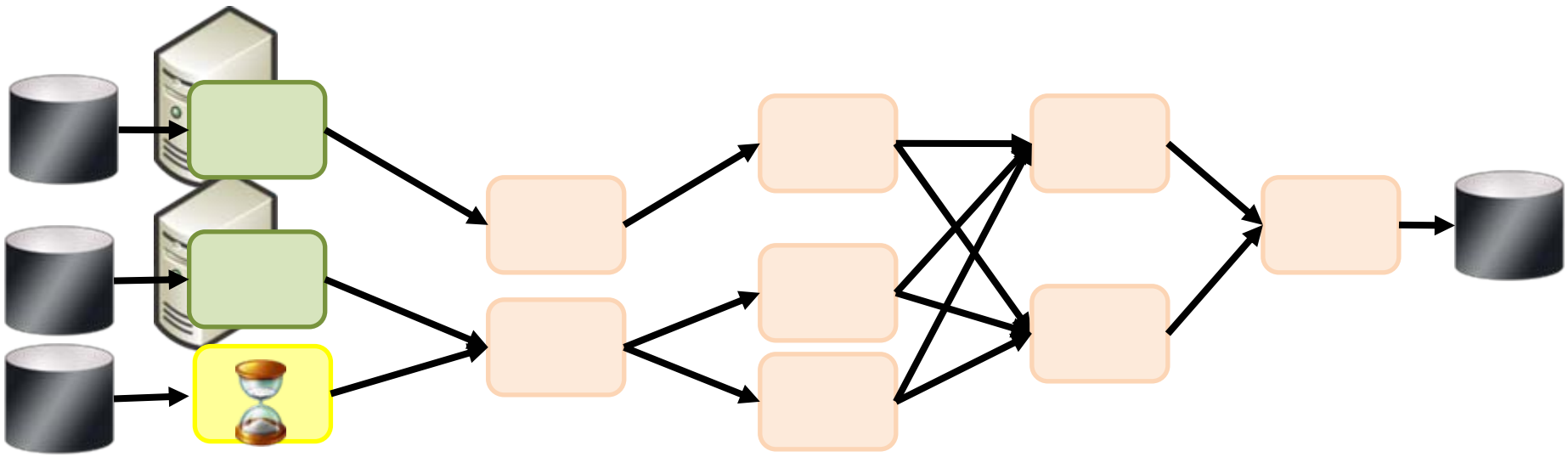
grep^{1000} | sed^{500} | sort^{1000} | awk^{500} | perl^{50}



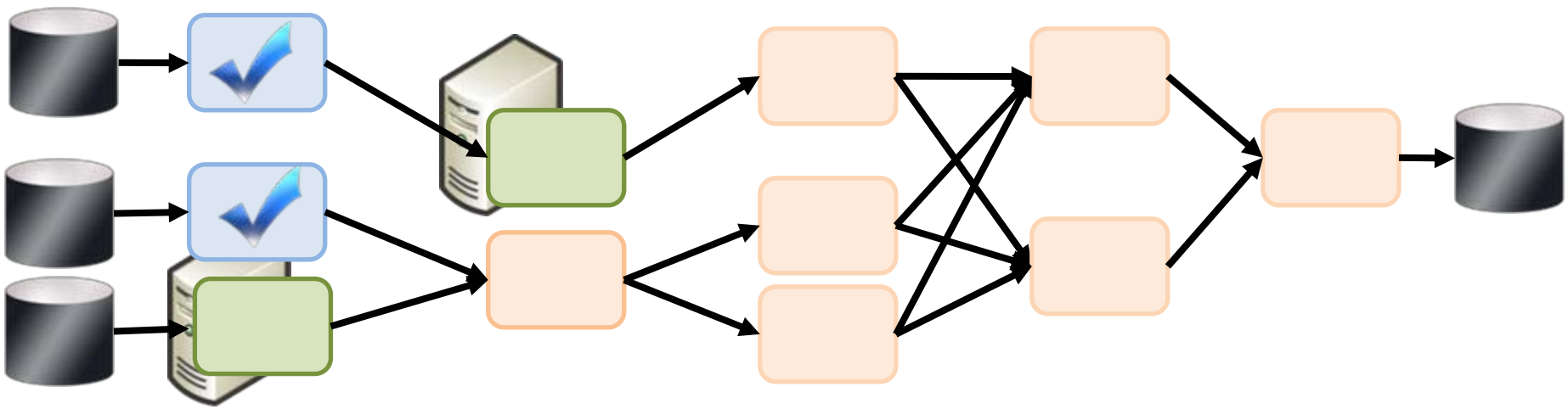
Virtualized 2-D Pipelines



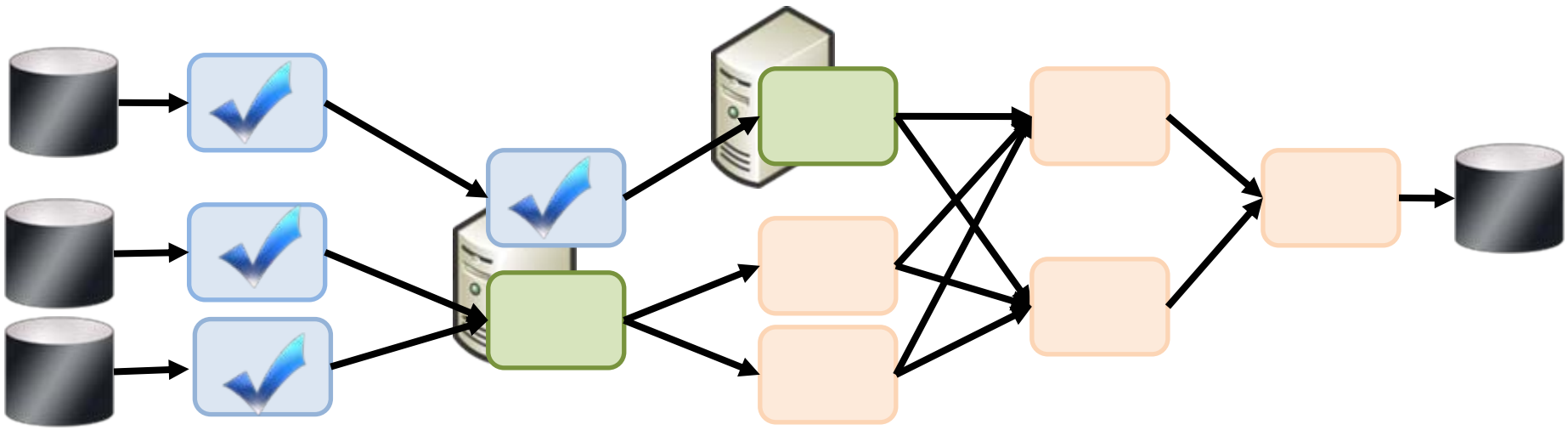
Virtualized 2-D Pipelines



Virtualized 2-D Pipelines

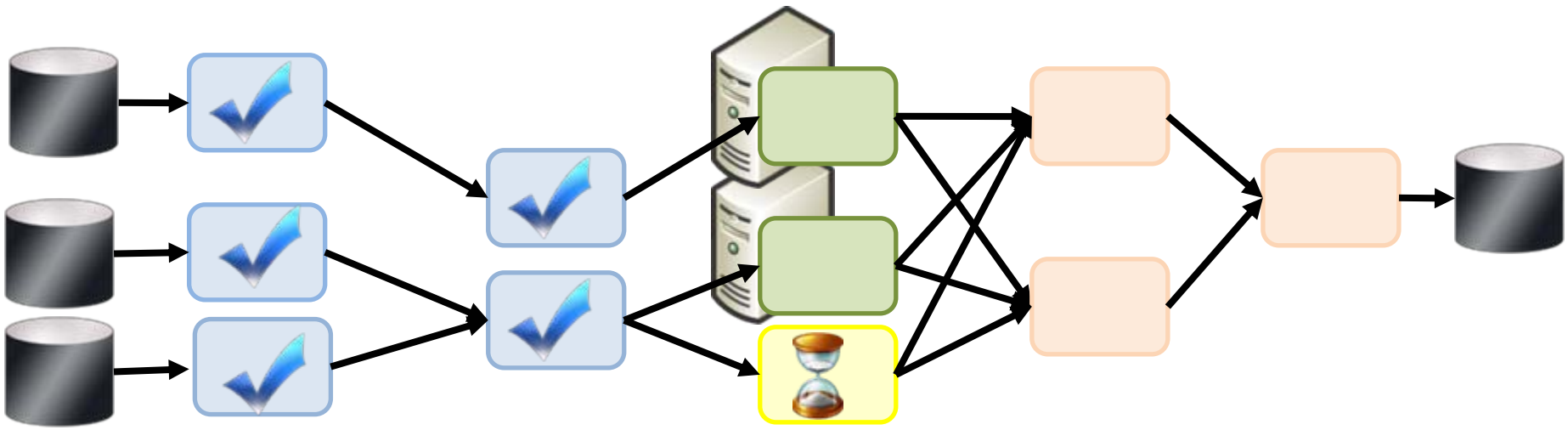


Virtualized 2-D Pipelines

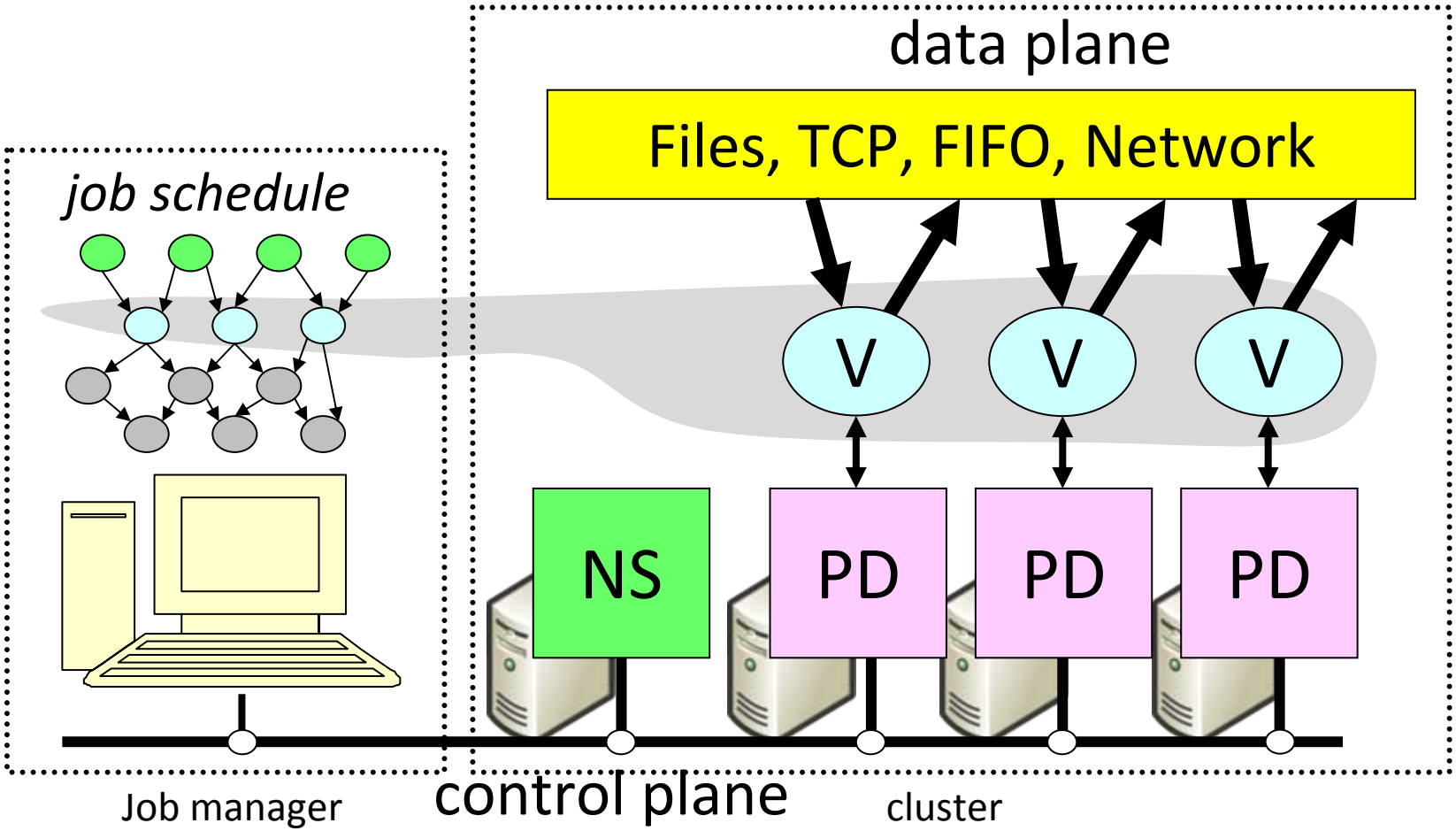


Virtualized 2-D Pipelines

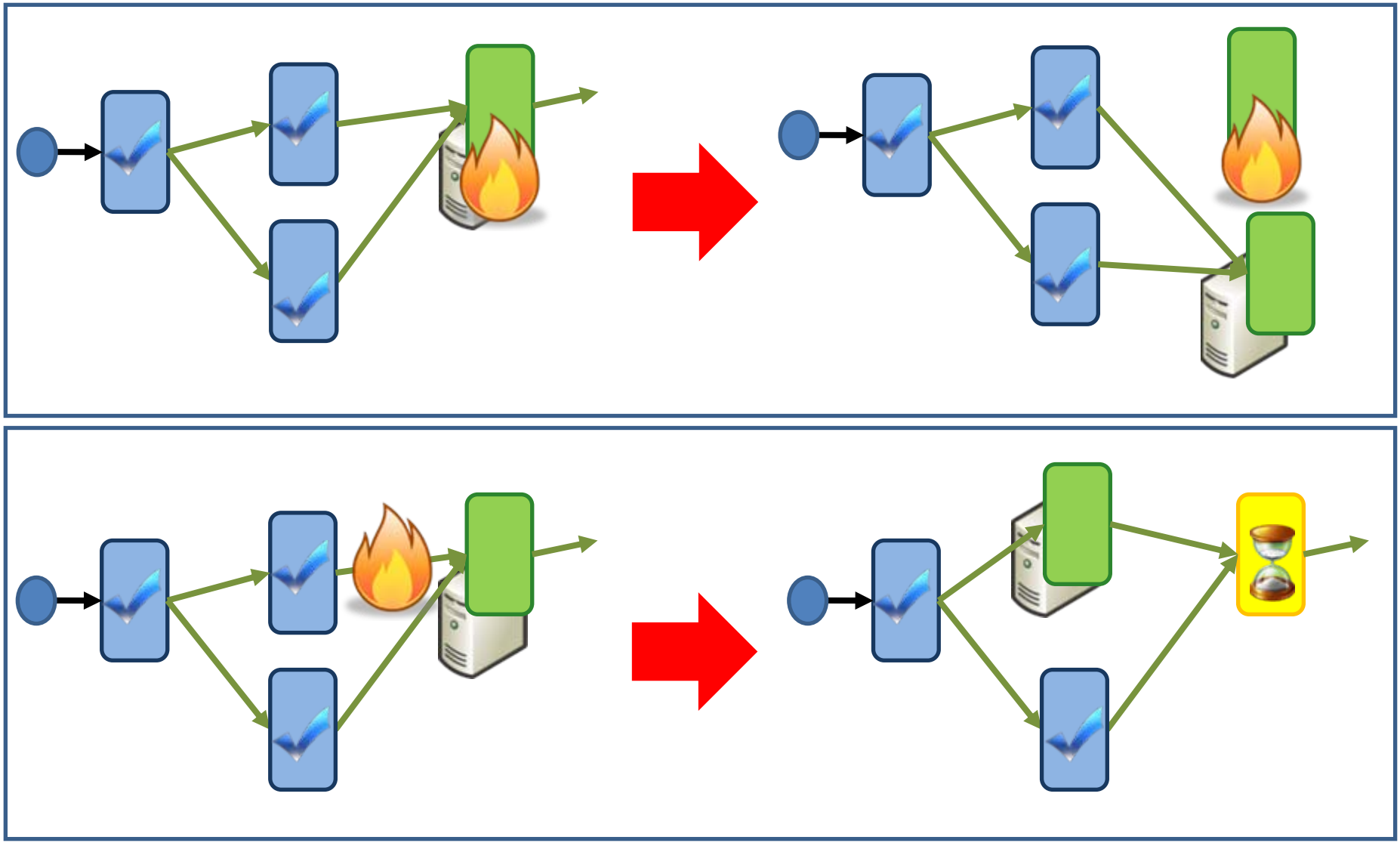
- 2D DAG
- multi-machine
- virtualized



Architecture



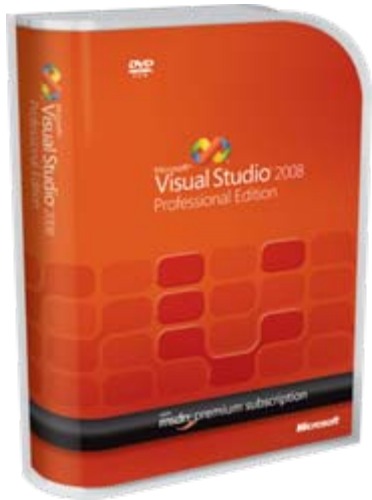
Fault Tolerance



Outline

- Introduction
- Dryad
- DryadLINQ
- Applications

DryadLINQ

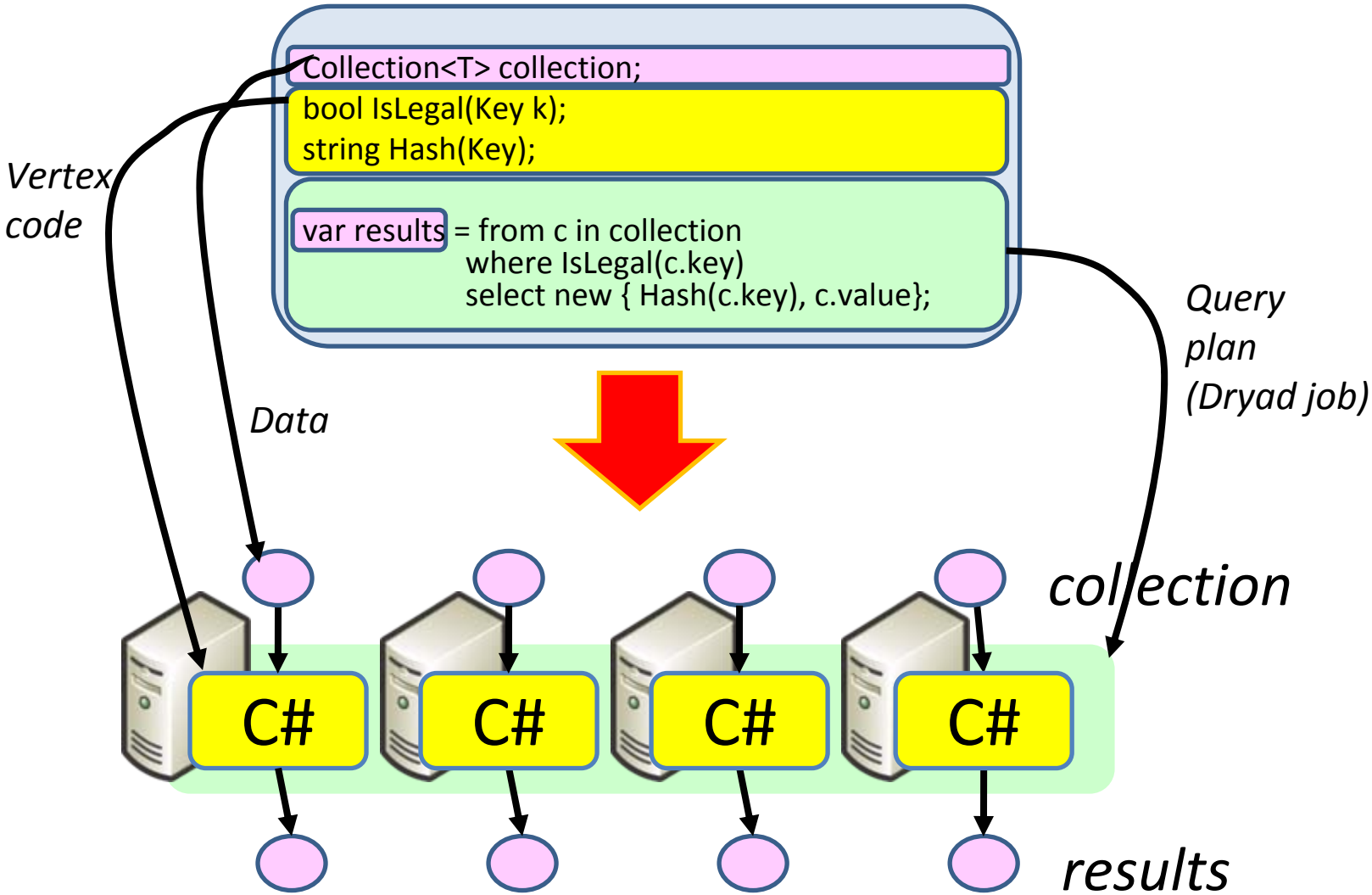


Dryad

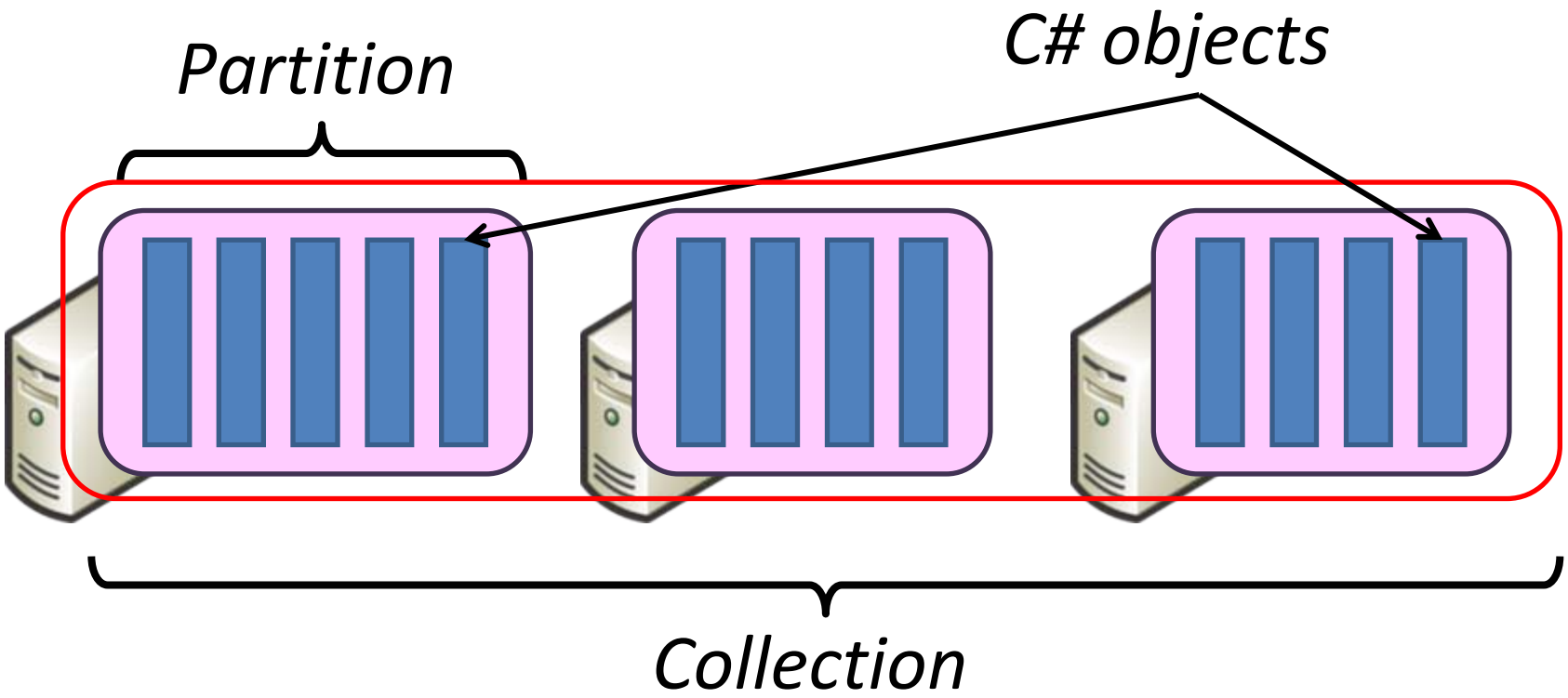
LINQ = C# + Queries

```
Collection<T> collection;
bool IsLegal(Key);
string Hash(Key);
var results = from c in collection
              where IsLegal(c.key)
              select new { Hash(c.key), c.value};
```

DryadLINQ = LINQ + Dryad

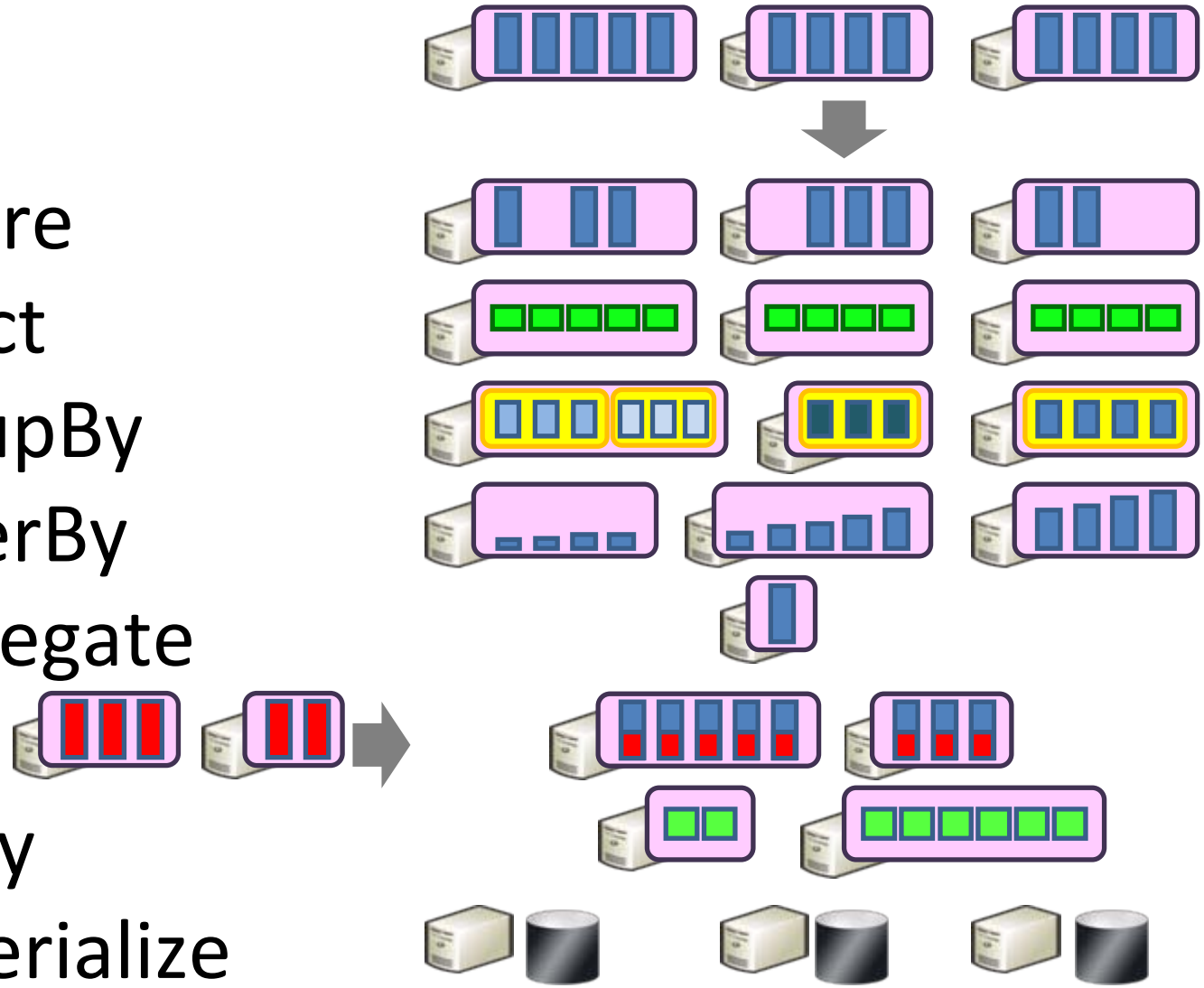


Data Model

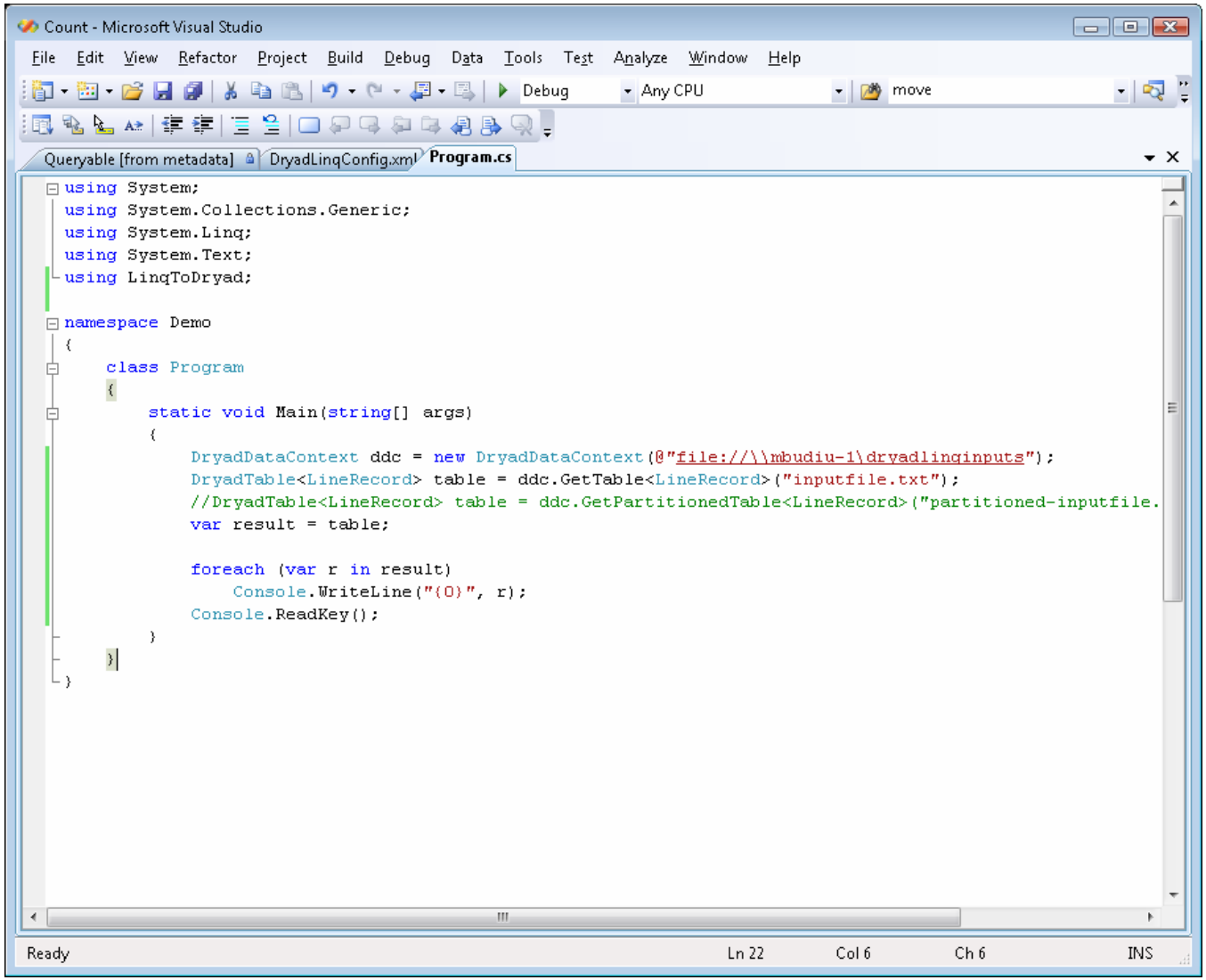


Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Materialize



Demo


Count - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools Test Analyze Window Help
Debug
Any CPU
move
Queryable [from metadata]
DryadLinqConfig.xml
Program.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using LinqToDryad;
namespace Demo
{
class Program
{
static void Main(string[] args)
{
DryadDataContext ddc = new DryadDataContext(@"file://\\mbudiu-1\dryadlinqinputs");
DryadTable<LineRecord> table = ddc.GetTable<LineRecord>("inputfile.txt");
//DryadTable<LineRecord> table = ddc.GetPartitionedTable<LineRecord>("partitioned-inputfile.
var result = table;
foreach (var r in result)
Console.WriteLine("{0}", r);
Console.ReadKey();
}
}
}
Ready
Ln 22
Col 6
Ch 6
INS

Done

Example: Histogram

```
public static IQueryable<Pair> Histogram(
      IQueryable<LineRecord> input, int k)
{
    var words = input.SelectMany(x => x.line.Split(' '));
    var groups = words.GroupBy(x => x);
    var counts = groups.Select(x => new Pair(x.Key, x.Count()));
    var ordered = counts.OrderByDescending(x => x.count);
    var top = ordered.Take(k);
    return top;
}
```

“A line of words of wisdom”
[“A”, “line”, “of”, “words”, “of”, “wisdom”]
[[“A”], [“line”], [“of”, “of”], [“words”], [“wisdom”]]
[{“A”, 1}, {“line”, 1}, {“of”, 2}, {“words”, 1}, {“wisdom”, 1}]
[{“of”, 2}, {“A”, 1}, {“line”, 1}, {“words”, 1}, {“wisdom”, 1}]
[{“of”, 2}, {“A”, 1}, {“line”, 1}]

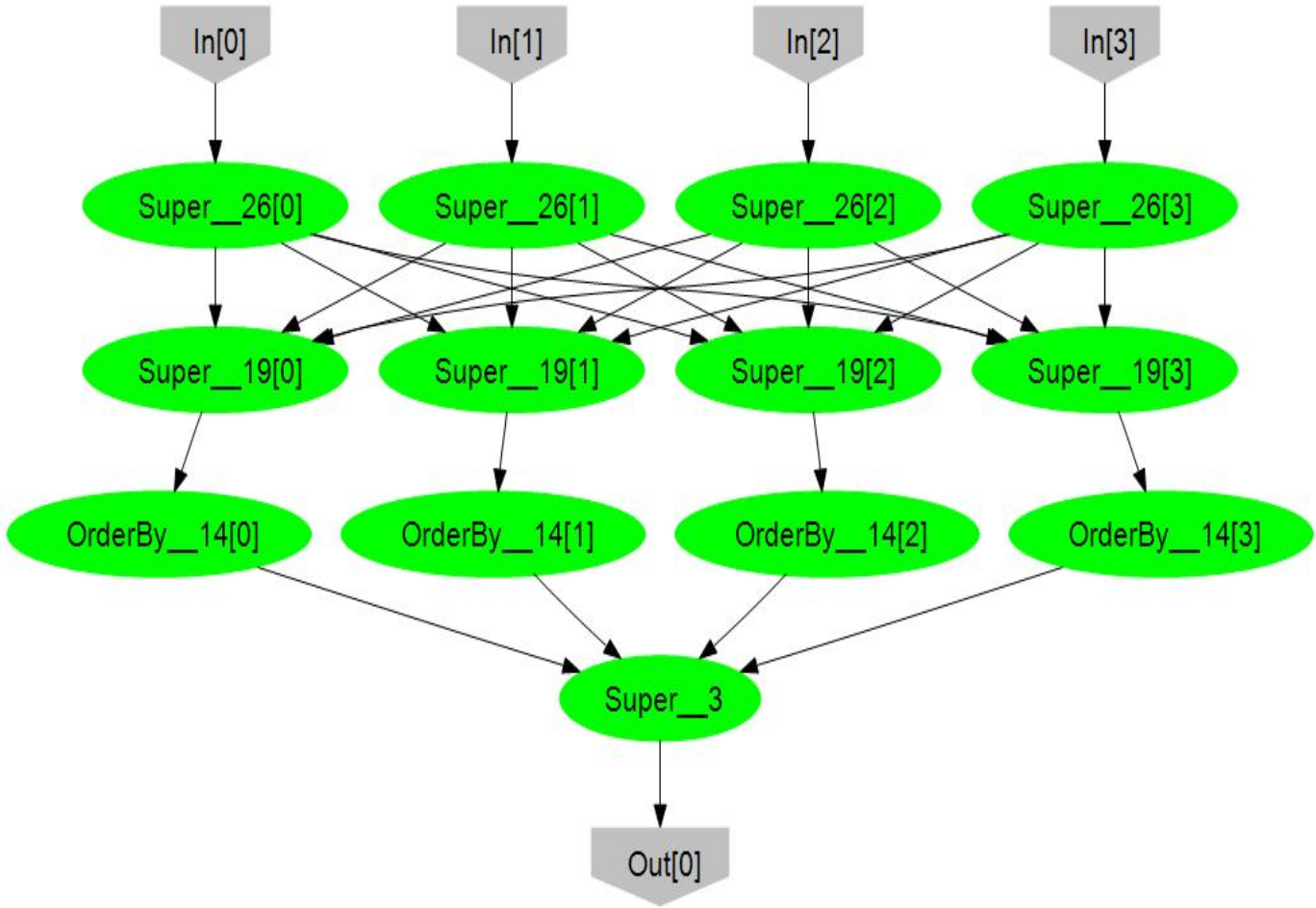
Histogram Plan

SelectMany
HashDistribute

Merge
GroupBy
Select

OrderByDescending
Take

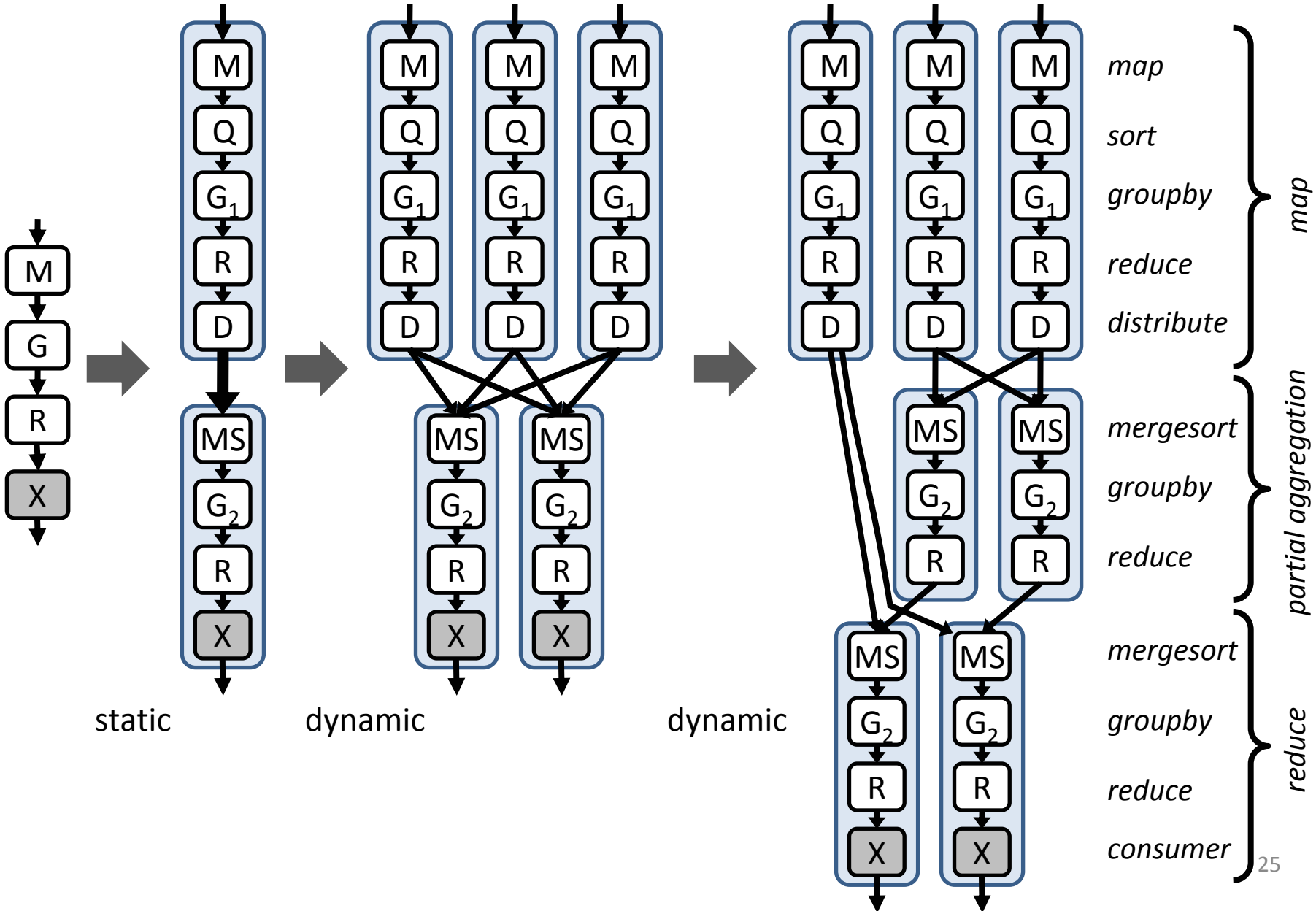
MergeSort
Take



Map-Reduce in DryadLINQ

```
public static IQueryable<S> MapReduce<T,M,K,S>(
    this IQueryable<T> input,
    Expression<Func<T, IEnumerable<M>>> mapper,
    Expression<Func<M,K>> keySelector,
    Expression<Func<IGrouping<K,M>,S>> reducer)
{
    var map = input.SelectMany(mapper);
    var group = map.GroupBy(keySelector);
    var result = group.Select(reducer);
    return result;
}
```

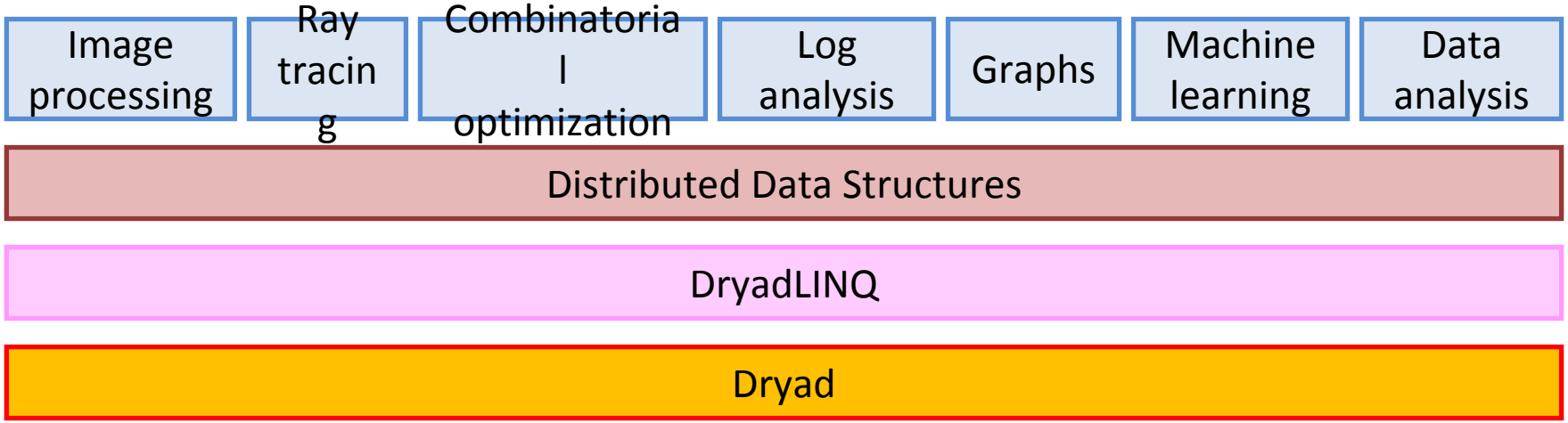
Map-Reduce Plan



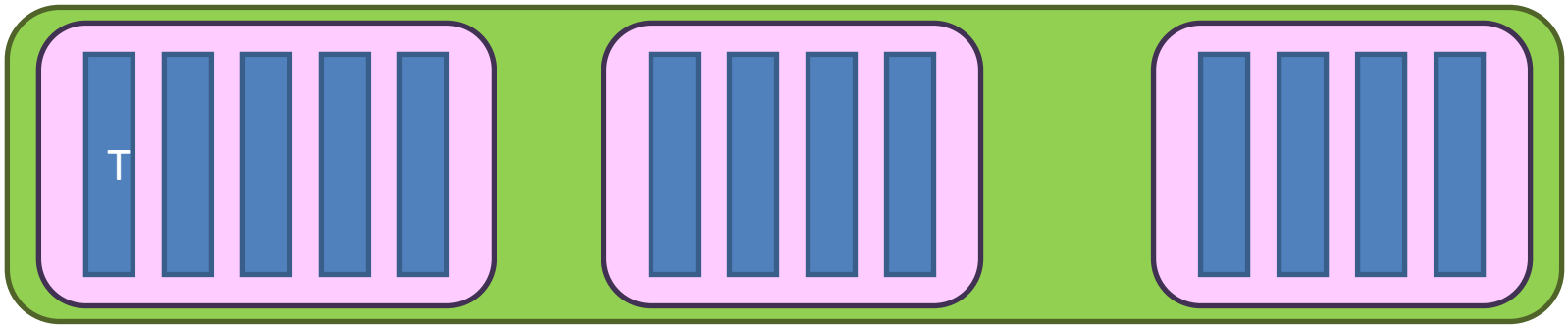
Outline

- Introduction
- Dryad
- DryadLINQ
- Applications

Applications

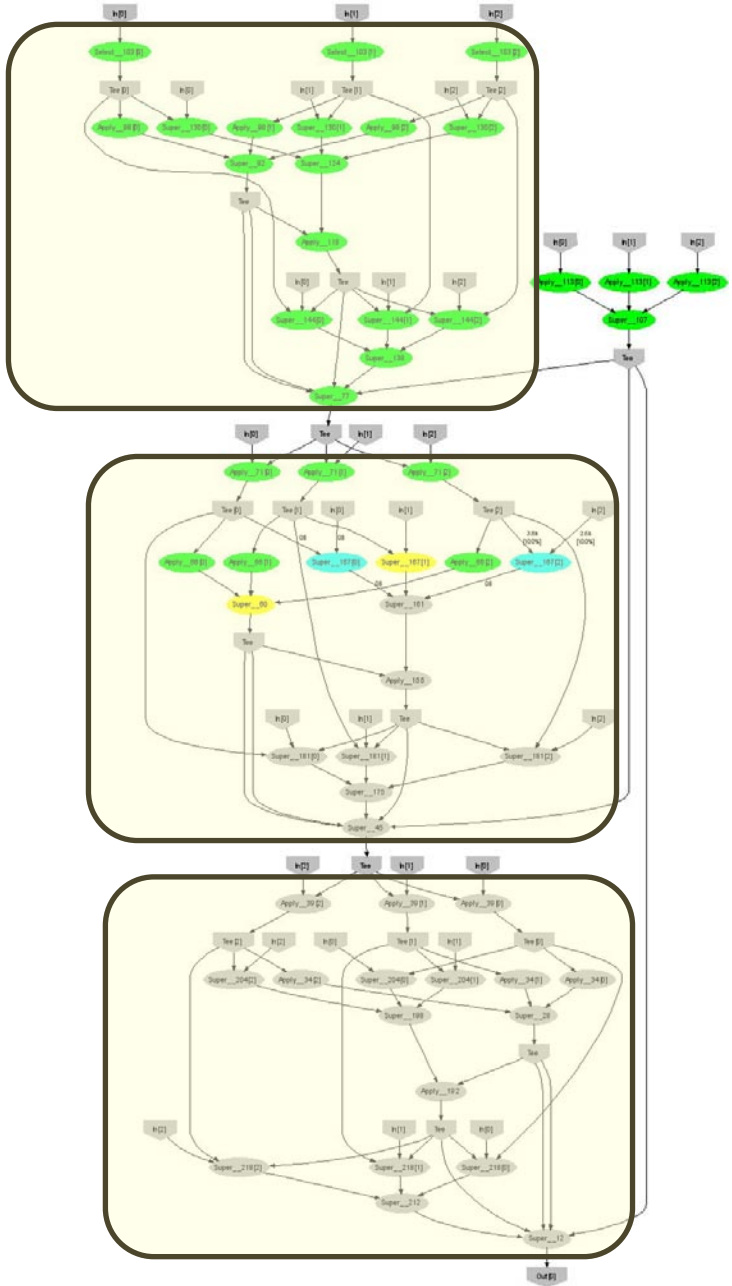


E.g: Linear Algebra



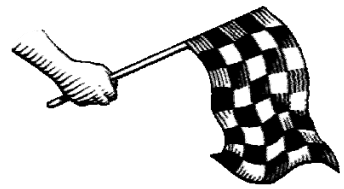
$$\{ T, U, V \} = \{ \Re, \Re^m, \Re^{m \times n} \}$$

Expectation Maximization (Gaussians)



- 160 lines
- 3 iterations shown

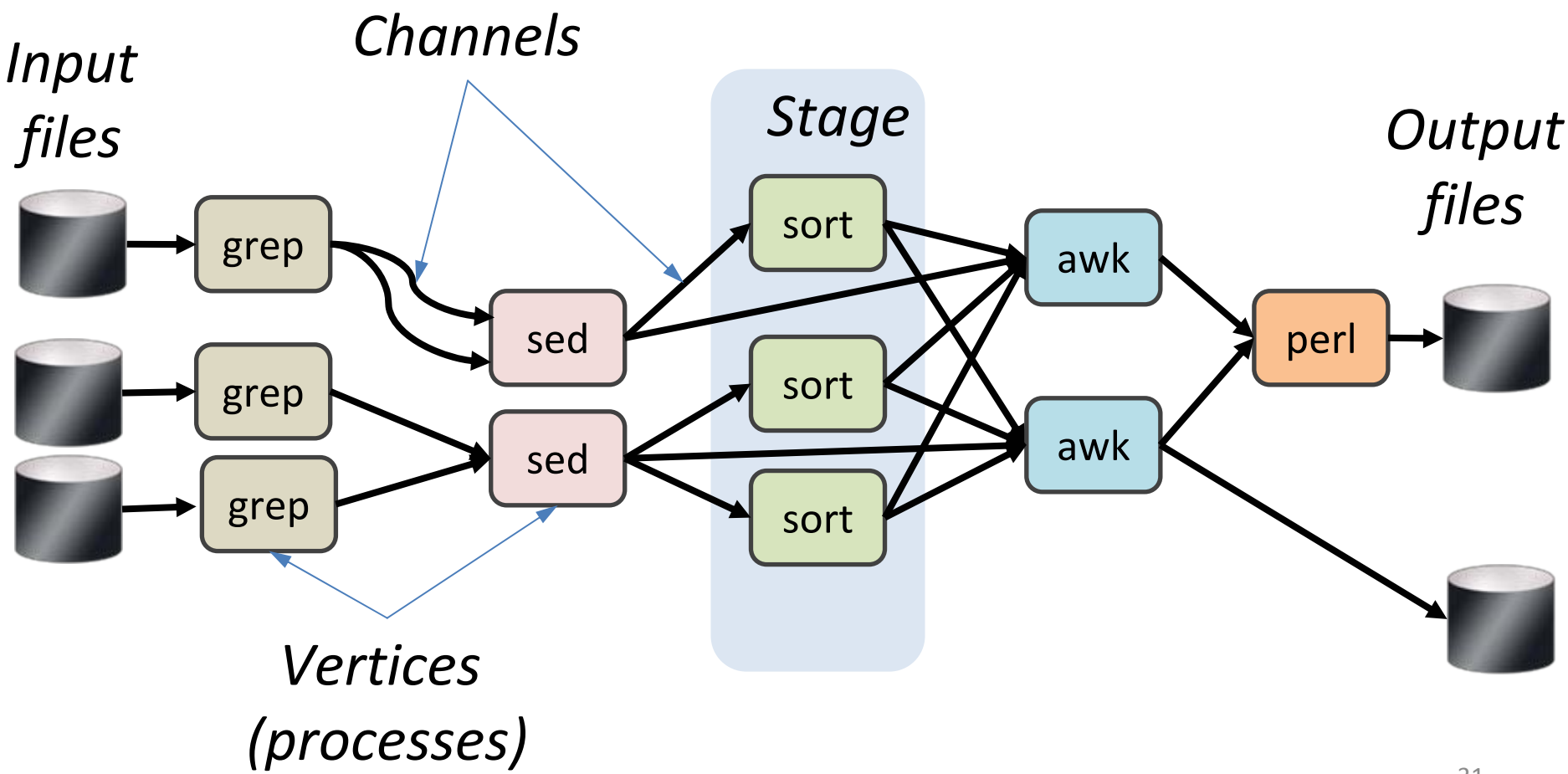
Conclusions



- Dryad = distributed execution environment
- Supports rich software ecosystem

- DryadLINQ = Compiles LINQ to Dryad
- C# objects and declarative programming
- .Net and Visual Studio
  for distributed programming

Dryad Job Structure



Linear Regression

- Data

$$x_t \in \Re^n, y_t \in \Re^m \qquad t \in \{1,\ldots,n\}$$

- Find

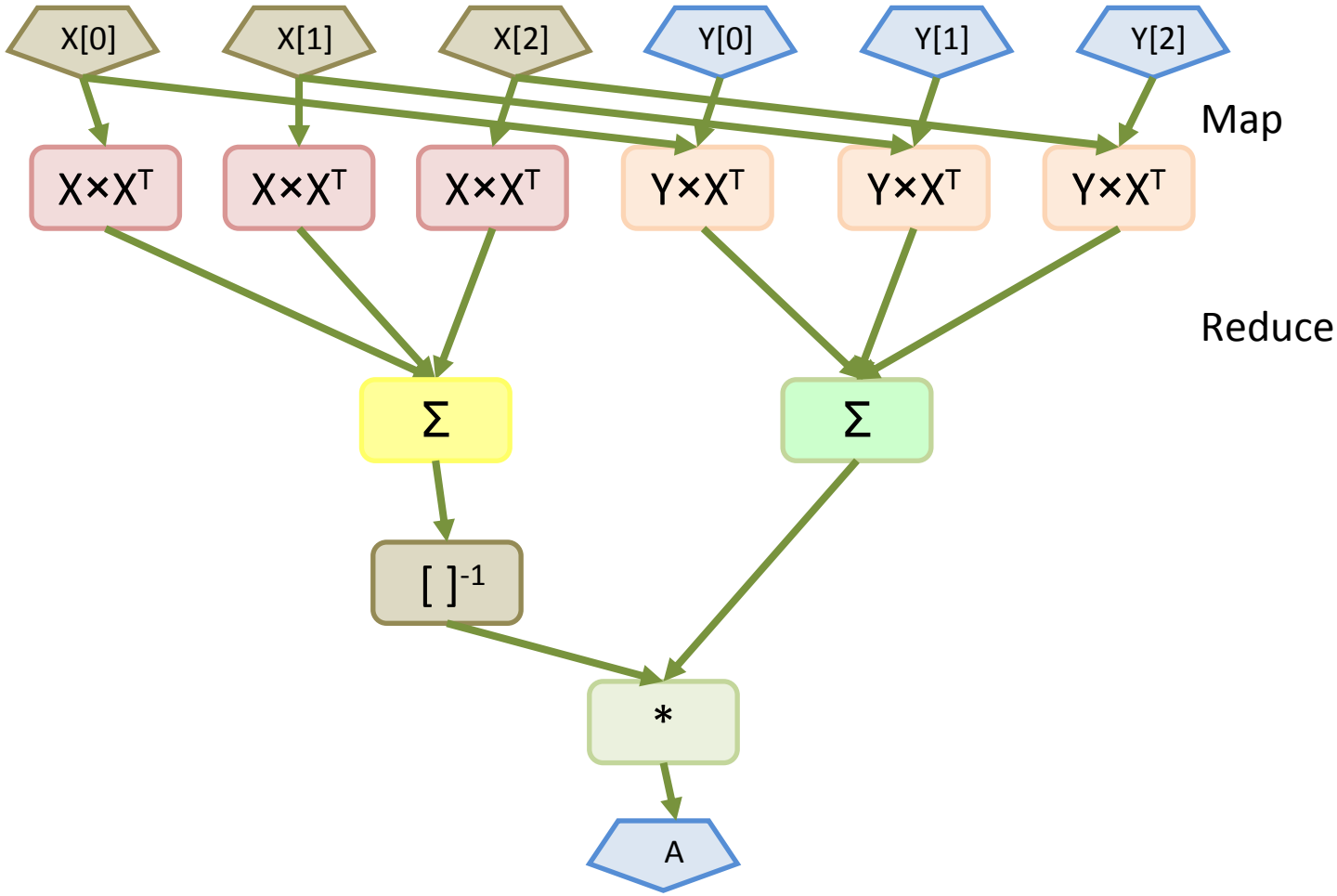
$$A \in \Re^{n \times m}$$

- S.t.

$$Ax_t \approx y_t$$

Analytic Solution

$$A = \left(\sum_t y_t \times x_t^T\right)\left(\sum_t x_t \times x_t^T\right)^{-1}$$

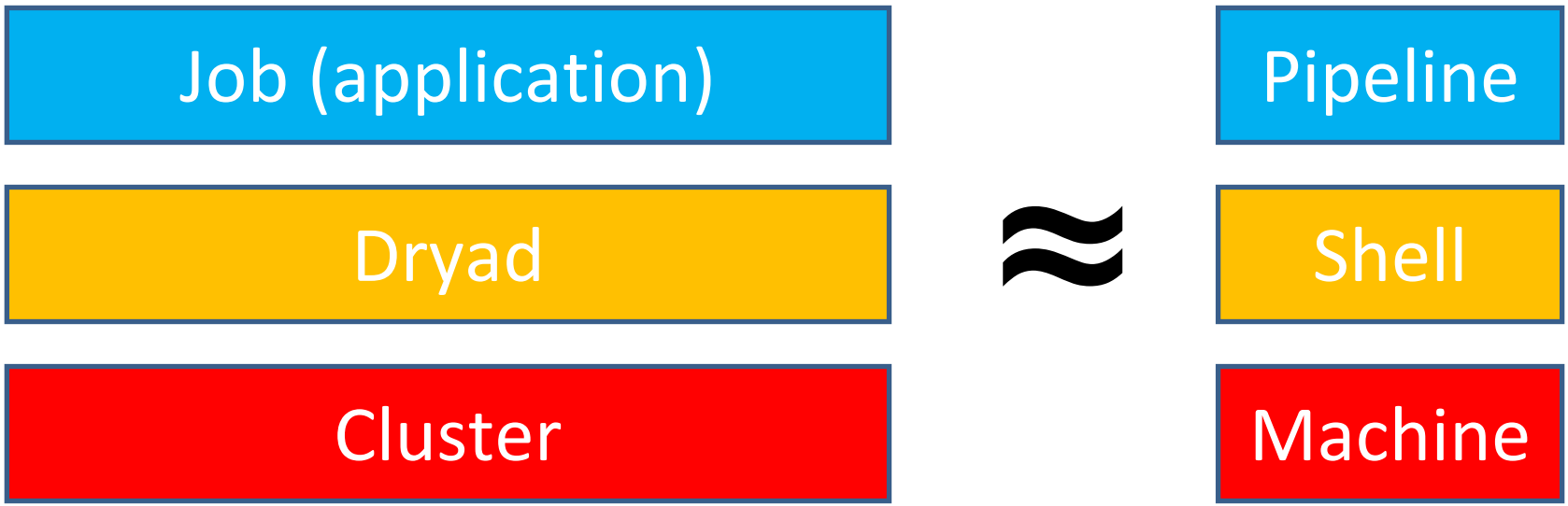


Linear Regression Code

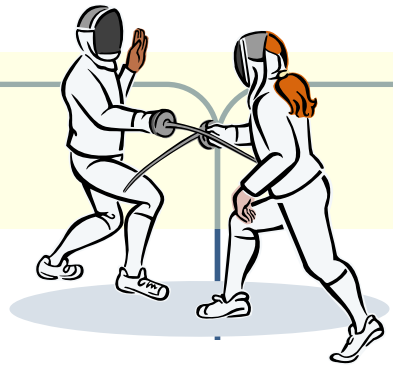
$$A = (\sum_t y_t \times x_t^T)(\sum_t x_t \times x_t^T)^{-1}$$

```
Vectors x = input(0), y  = input(1);
Matrices xx = x.Map(x, (a,b) => a.OuterProd(b));
OneMatrix xxs = xx.Sum();
Matrices yx = y.Map(x, (a,b) => a.OuterProd(b));
OneMatrix yxs = yx.Sum();
OneMatrix xxinv = xxs.Map(a => a.Inverse());
OneMatrix A = yxs.Map(xxinv, (a, b) => a.Mult(b));
```

Dryad = Execution Layer



Dryad

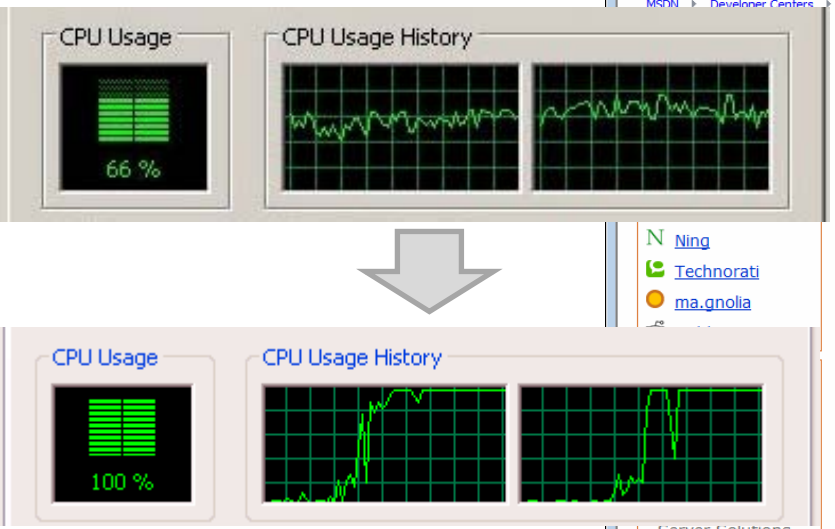


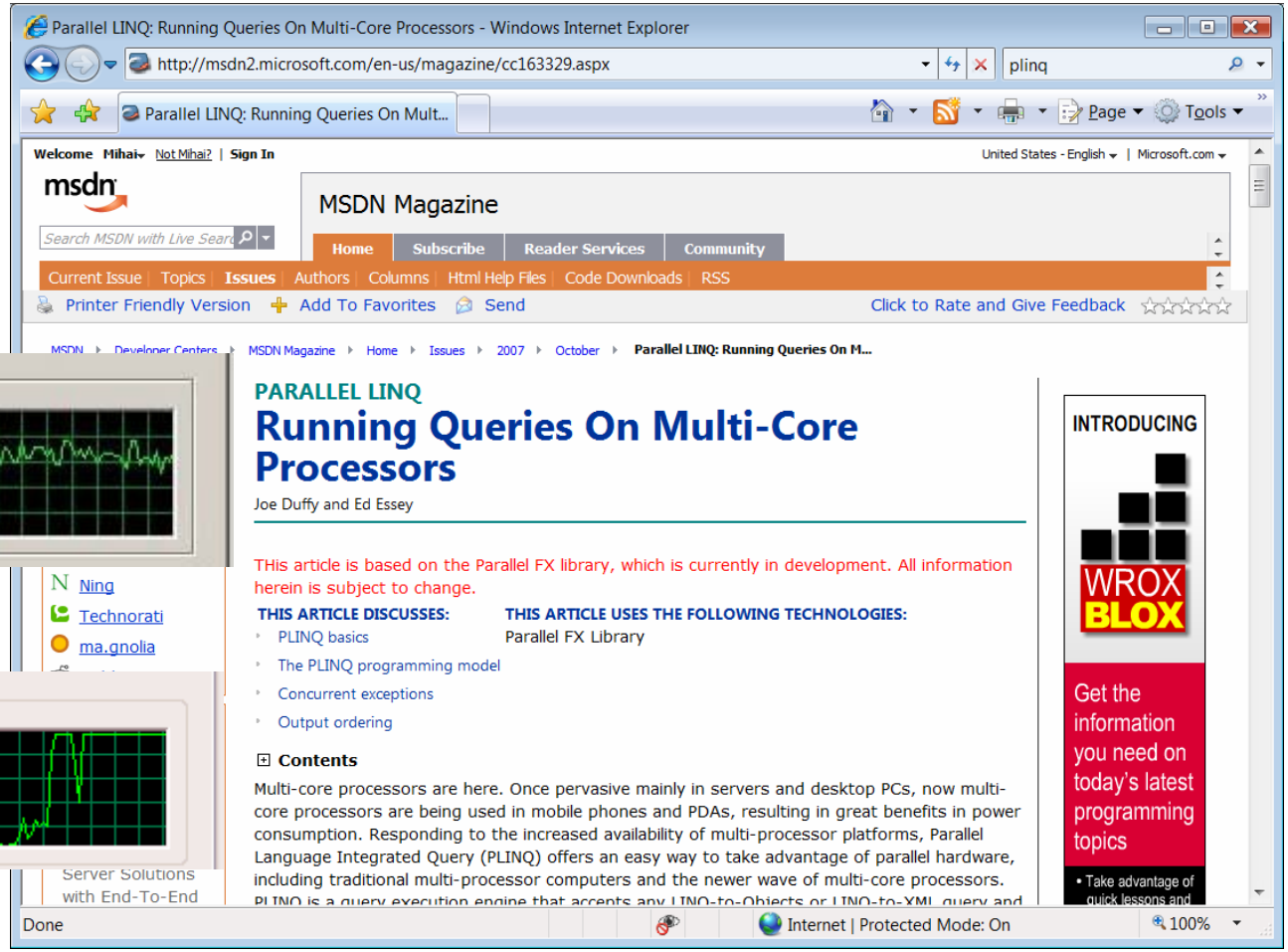
Map-Reduce

- Many similarities

• Execution layer • Job = arbitrary DAG • Plug-in policies • Program=graph gen.	• Exe + app. model • Map+sort+reduce • Few policies • Program=map+reduce
• Complex (⬆features) • New (< 2 years) • Still growing • Internal	• Simple • Mature (> 4 years) • Widely deployed • Hadoop

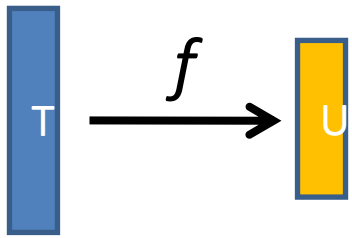
PLINQ



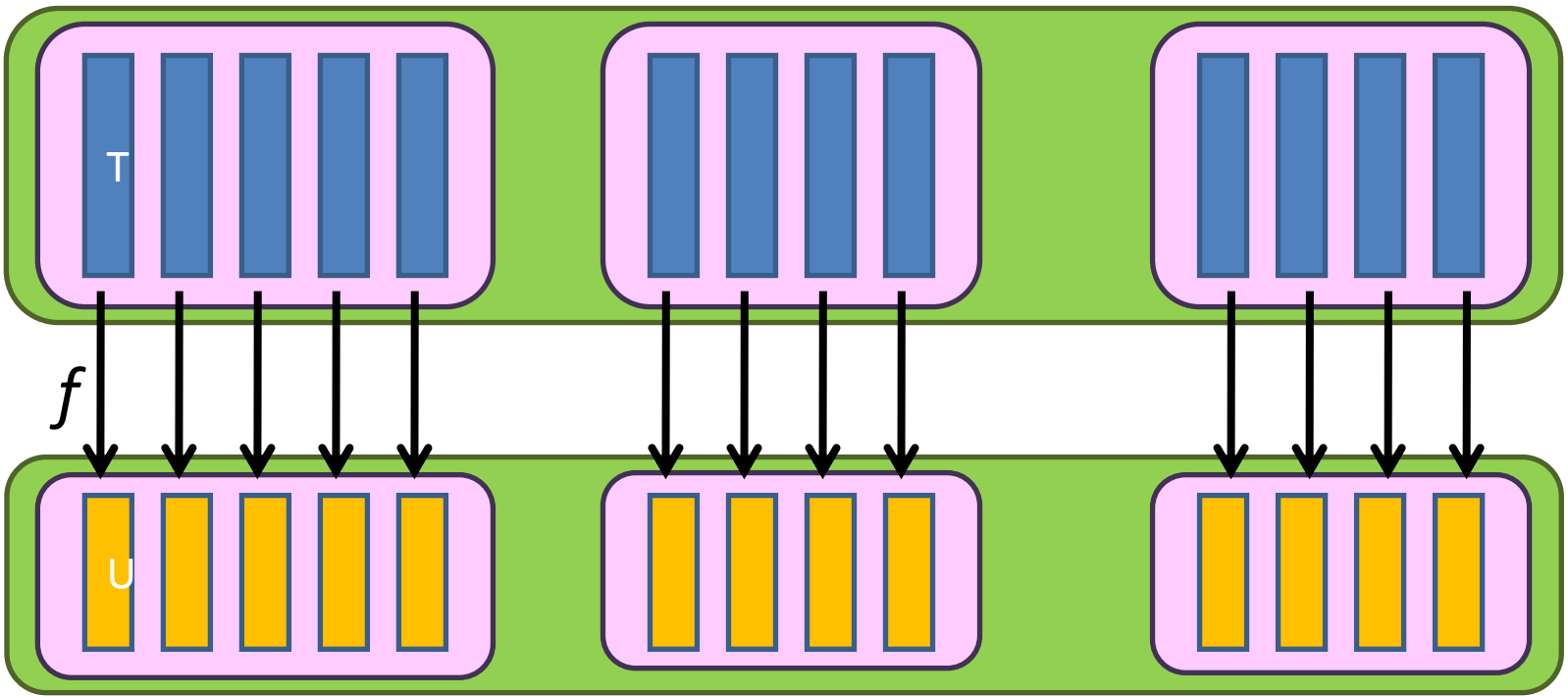



```
public static IEnumerable<TSource>
            DryadSort<TSource, TKey>(IEnumerable<TSource> source,
                                     Func<TSource, TKey> keySelector,
                                     IComparer<TKey> comparer,
                                     bool isDescending)
{
        return source.AsParallel().OrderBy(keySelector, comparer);
}
```

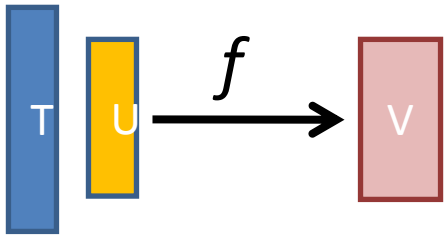
Operations on Large Vectors: Map 1

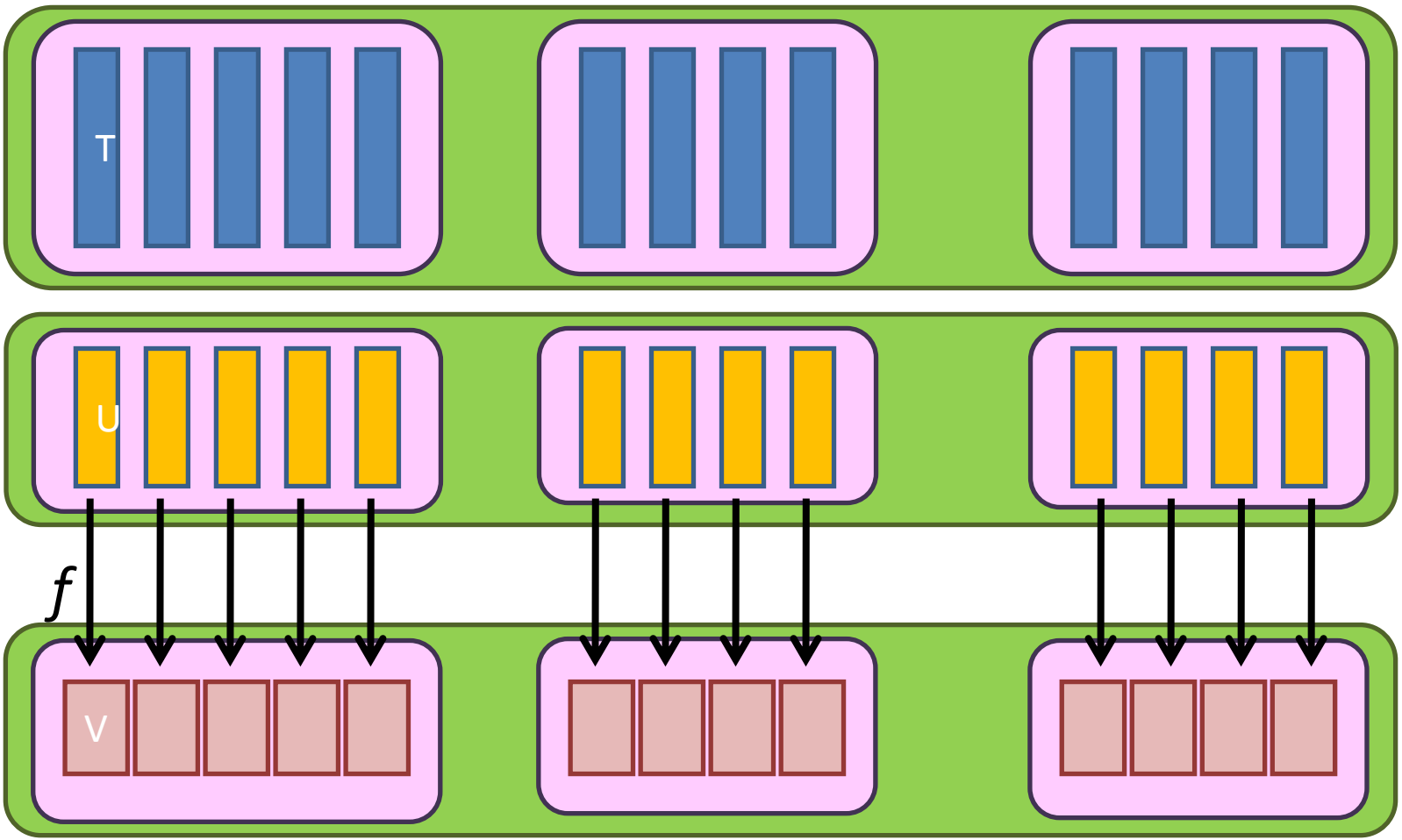


f preserves partitioning

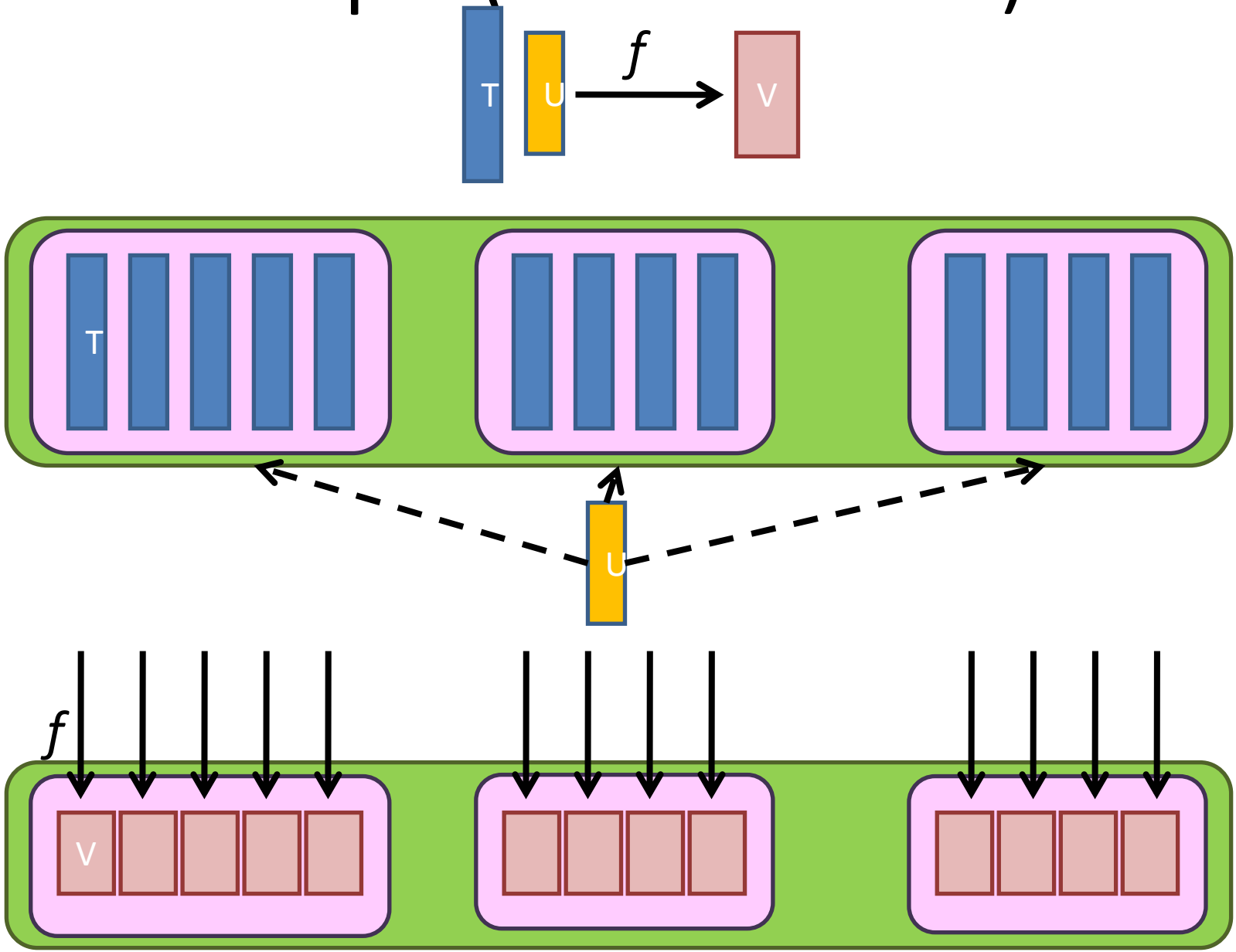


Map 2 (Pairwise)

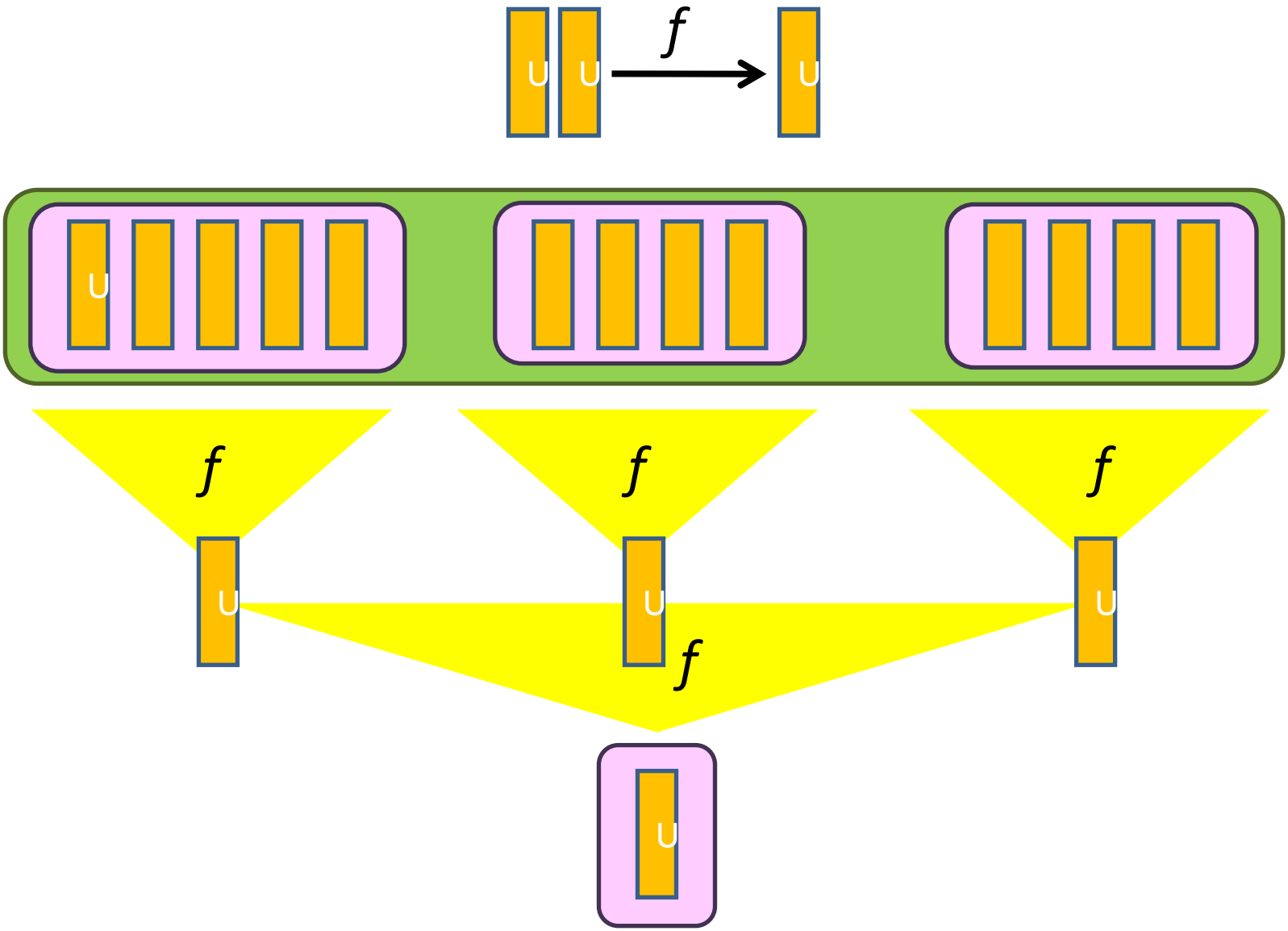




Map 3 (Vector-Scalar)



Reduce (Fold)



Software Stack