Dryad and DryadLINQ

Yuan Yu

Microsoft Research Silicon Valley

Joint work with
Andrew Birrell, Mihai Budiu, Jon Currey, Úlfar Erlingsson,
Dennis Fetterly, Pradeep Kumar Gunda, Michael Isard

Dryad and DryadLINQ

DryadLINQ provides automatic query plan generation
Dryad provides automatic distributed execution

DryadLinq - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Data Tools Test Analyze Window Help

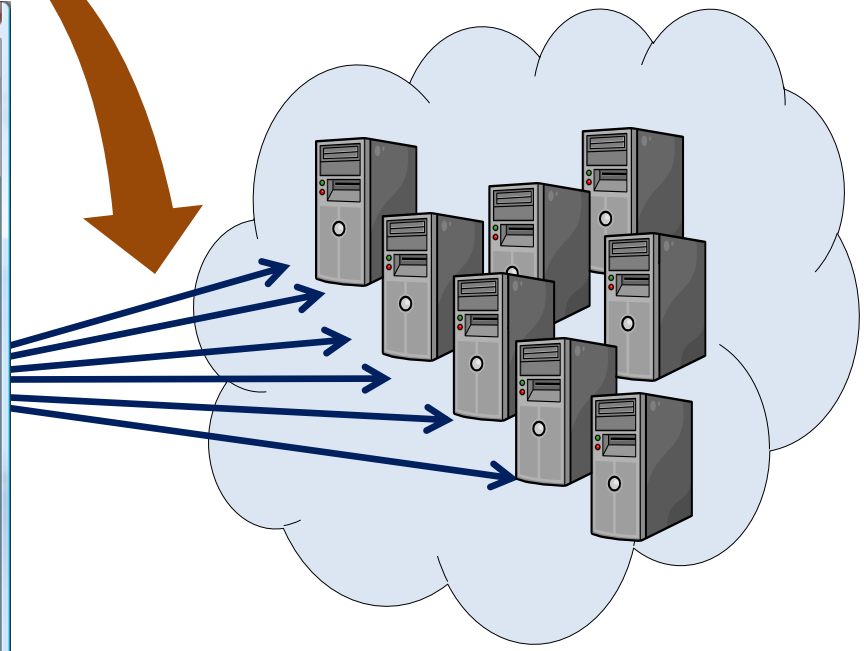
Release | Any CPU

PageRank.cs | Start Page | Object Browser | DryadQueryNode.cs | DryadLinqGlobals.cs

PageRank | ComputePageRank(IQueryable<Page> pages, IQueryab

```
using System;
using System.Collections.Generic;
using System.Linq;

class PageRank
{
    static IQueryable<Rank> ComputePageRank(IQueryable<Page> pages,
                                            IQueryable<Rank> ranks)
    {
        for (int iter = 0; iter < iterations; iter++)
        {
            // join pages with ranks, and disperse updates
            var updates = from page in pages
                          join rank in ranks on page.name equals rank.name
                          select page.Distribute(rank);

            // re-accumulate.
            ranks = from list in updates
                    from rank in list
                    group rank.rank by rank.page into g
                    select new Rank(g.Key, g.Sum());
        }
        return ranks;
    }
```

Build succeeded | Ln 19 | Col 38 | Ch 38 | INS



Availability

Dryad/DryadLINQ on HPC is available as a free download from:

http://research.microsoft.com/en-us/collaboration/tools/dryad.aspx

- *DryadLINQ (in source) & Dryad (in binary)*
- *With tutorials, programming guides, sample codes, libraries, and a community site on Microsoft Connect*

Outline

- Programming model and demo
- Dryad and DryadLINQ overview
- Applications
- Lessons and conclusions

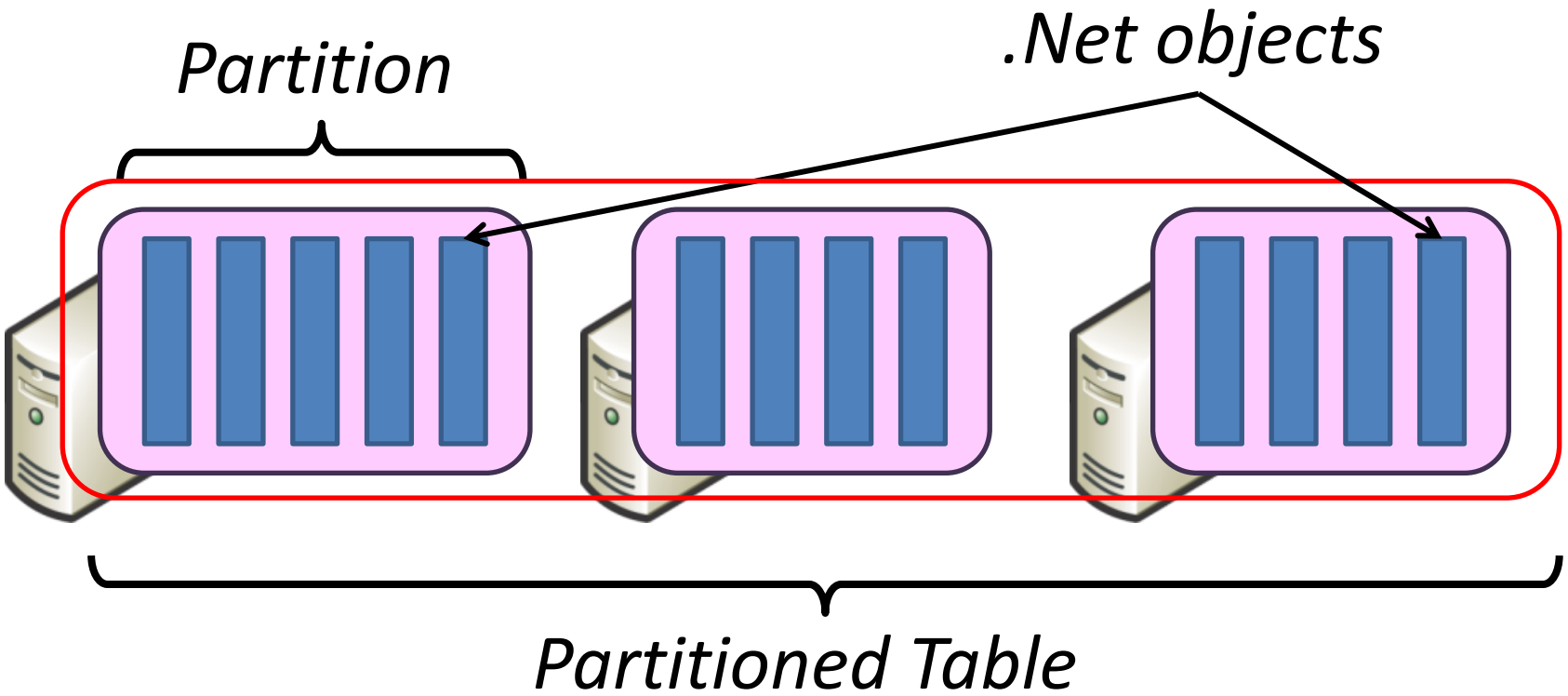
The Programming Model

- Use a cluster as if it is a single computer
  - Sequential, single machine programming abstraction
  - Same program runs on single-core, multi-core, or cluster
  - Familiar programming languages
    - C#, VB, F#, IronPython, …
  - Familiar development environment
    - Visual Studio and .NET

LINQ

- Microsoft's Language INtegrated Query
  - Available in .NET3.5 and Visual Studio 2008
- A set of operators to manipulate datasets in .NET
  - Support traditional relational operators
    - Select, Join, GroupBy, Aggregate, etc.
  - Integrated into .NET programming languages
    - Programs can invoke operators
    - Operators can invoke arbitrary .NET functions
- Data model
  - Data elements are strongly typed .NET objects
  - Much more expressive than relational tables
    - For example, nested data structures

DryadLINQ Data Model



Partitioned table exposes metadata information

- type, partition, compression scheme, serialization, etc.

Demo

- It is just programming
  - The same familiar programming languages, development tools, libraries, etc.

K-means in DryadLINQ

```
public class Vector {
   public double[] entries;
   [Associative]
   public static Vector operator +(Vector v1, Vector v2) {  … }
   public static Vector operator -(Vector v1, Vector v2) { … }
   public double Norm2() { …}
}
```

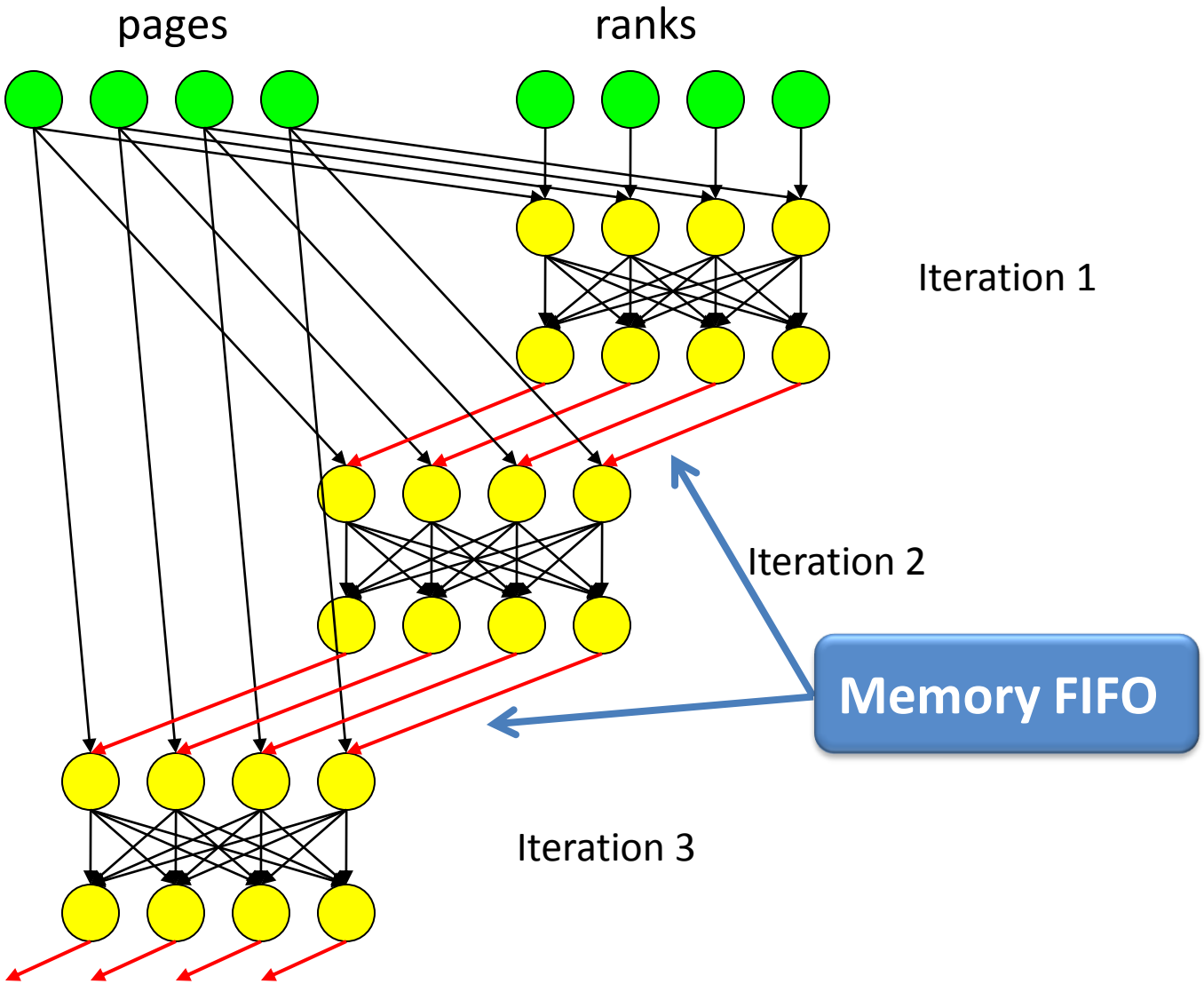
```
public static Vector NearestCenter(Vector v, IEnumerable<Vector> centers) {
   return centers.Aggregate((r, c) => (r - v).Norm2() < (c - v).Norm2() ? r : c);
}
public static IQueryable<Vector> Step(IQueryable<Vector> vectors, IQueryable<Vector> centers) {
   return vectors.GroupBy(v => NearestCenter(v, centers))
                 .Select(group => group.Aggregate((x,y) => x + y) / group.Count());
}

var vectors = PartitionedTable.Get<Vector>("dfs://vectors.pt");
var centers = vectors.Take(100);
for (int i = 0; i < 10; i++)
   centers = Step(vectors, centers);
centers.ToPartitionedTable<Vector>("dfs://centers.pt");
```

PageRank in DryadLINQ

```
public static IQueryable<Rank> Step(IQueryable<Page> pages,
                                    IQueryable<Rank> ranks) {
   // join pages with ranks, and disperse updates
   var updates = from page in pages
                 join rank in ranks on page.name equals rank.name
                 select page.Disperse(rank);

   // re-accumulate.
   return from list in updates
          from rank in list
          group rank.rank by rank.name into g
          select new Rank(g.Key, g.Sum());
}

var pages = PartitionedTable.Get<Page>("dfs://pages.pt");
var ranks = pages.Select(page => new Rank(page.name, 1.0));

// repeat the iterative computation several times
for (int iter = 0; iter < n; iter++) {
   ranks = Step(pages, ranks);
}

ranks.ToPartitionedTable<Rank>("dfs://ranks.pt");
```

```
public struct Page {
   public UInt64 name;
   public Int64 degree;
   public UInt64[] links;

   public Page(UInt64 n, Int64 d, UInt64[] l) {
      name = n; degree = d; links = l; }

   public Rank[] Disperse(Rank rank) {
      Rank[] ranks = new Rank[links.Length];
      double score = rank.rank / this.degree;
      for (int i = 0; i < ranks.Length; i++) {
         ranks[i] = new Rank(this.links[i], score);
      }
      return ranks;
   }
}

public struct Rank {
   public UInt64 name;
   public double rank;

   public Rank(UInt64 n, double r) {
      name = n; rank = r; }
}
```

Multi-Iteration PageRank



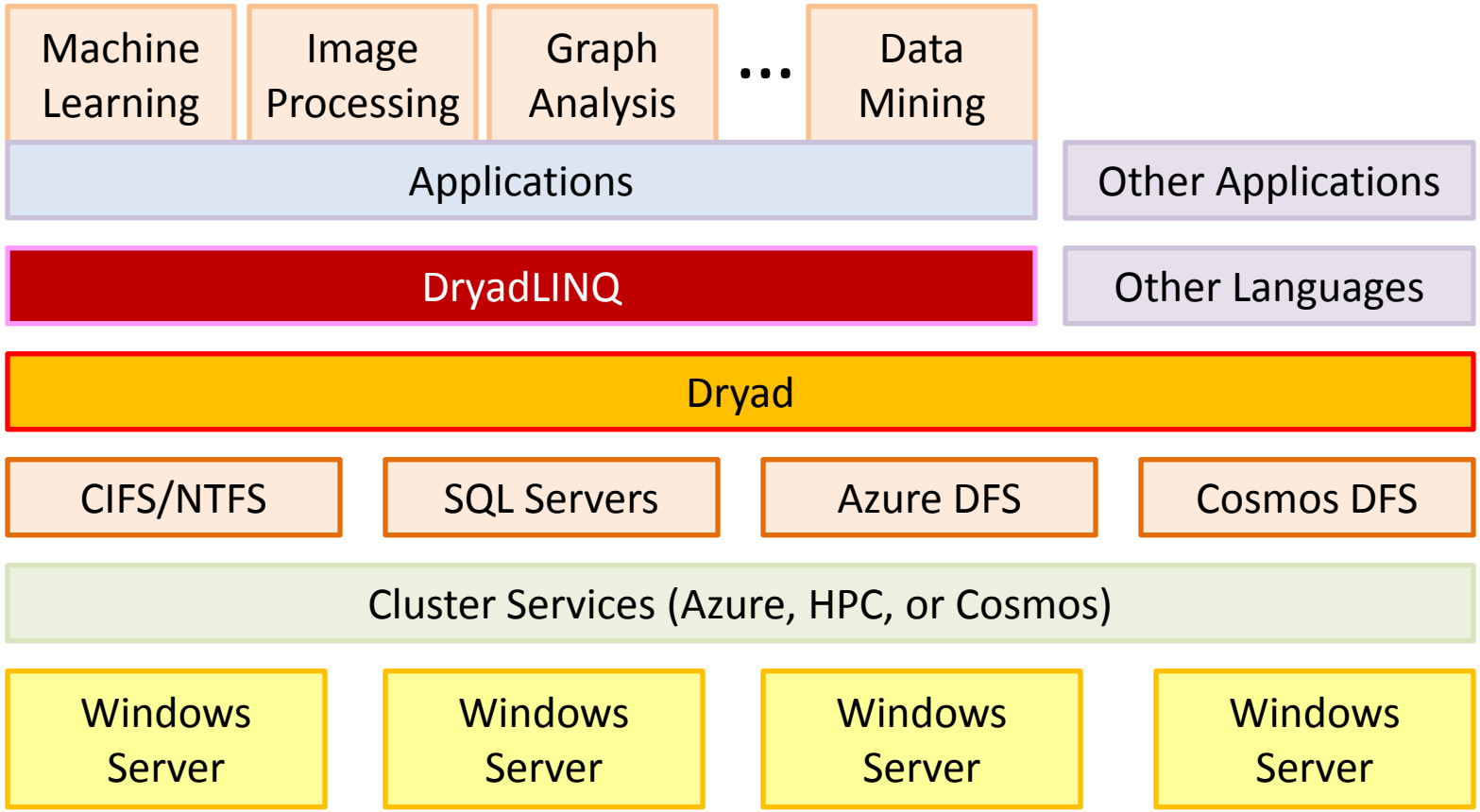
MapReduce in DryadLINQ

```
MapReduce(source,          // sequence of Ts
          mapper,          // T -> Ms
          keySelector,     // M -> K
          reducer)         // (K, Ms) -> Rs
{
    var map = source.SelectMany(mapper);
    var group = map.GroupBy(keySelector);
    var result = group.SelectMany(reducer);
    return result;     // sequence of Rs
}
```

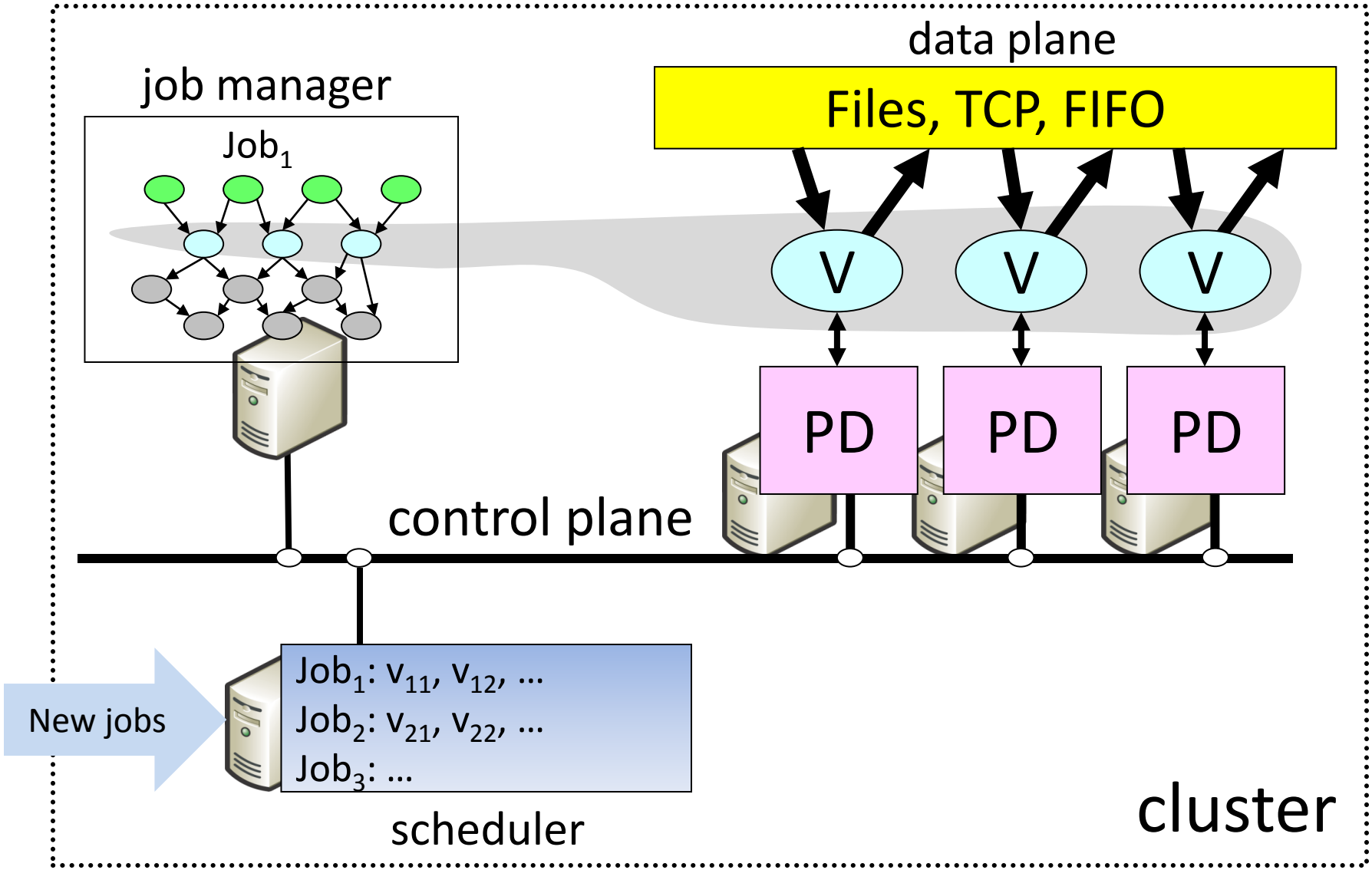
Outline

- Programming model and demo
- **Dryad and DryadLINQ overview**
- Applications
- Conclusions

Software Stack



Dryad System Architecture

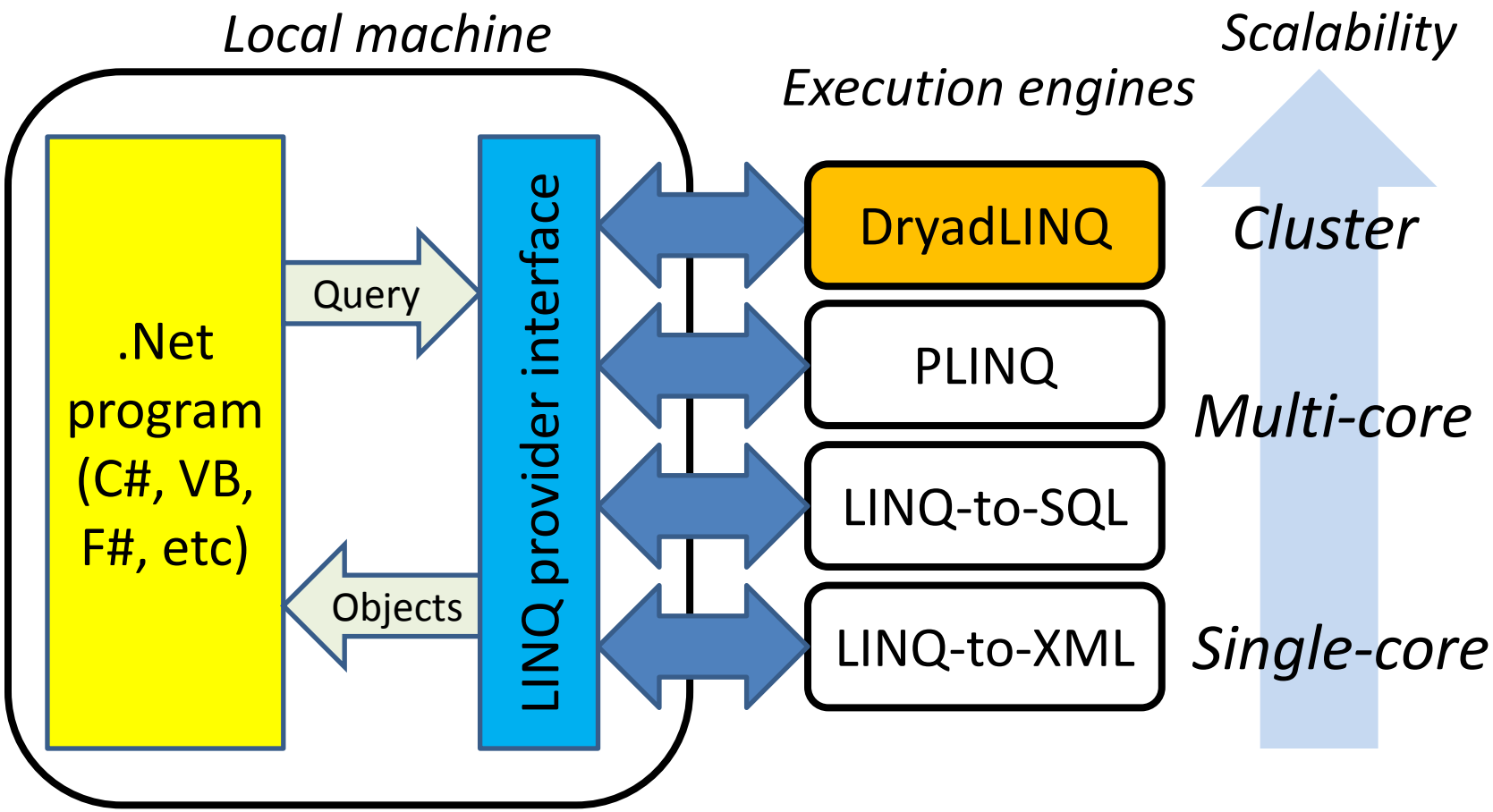


Dryad

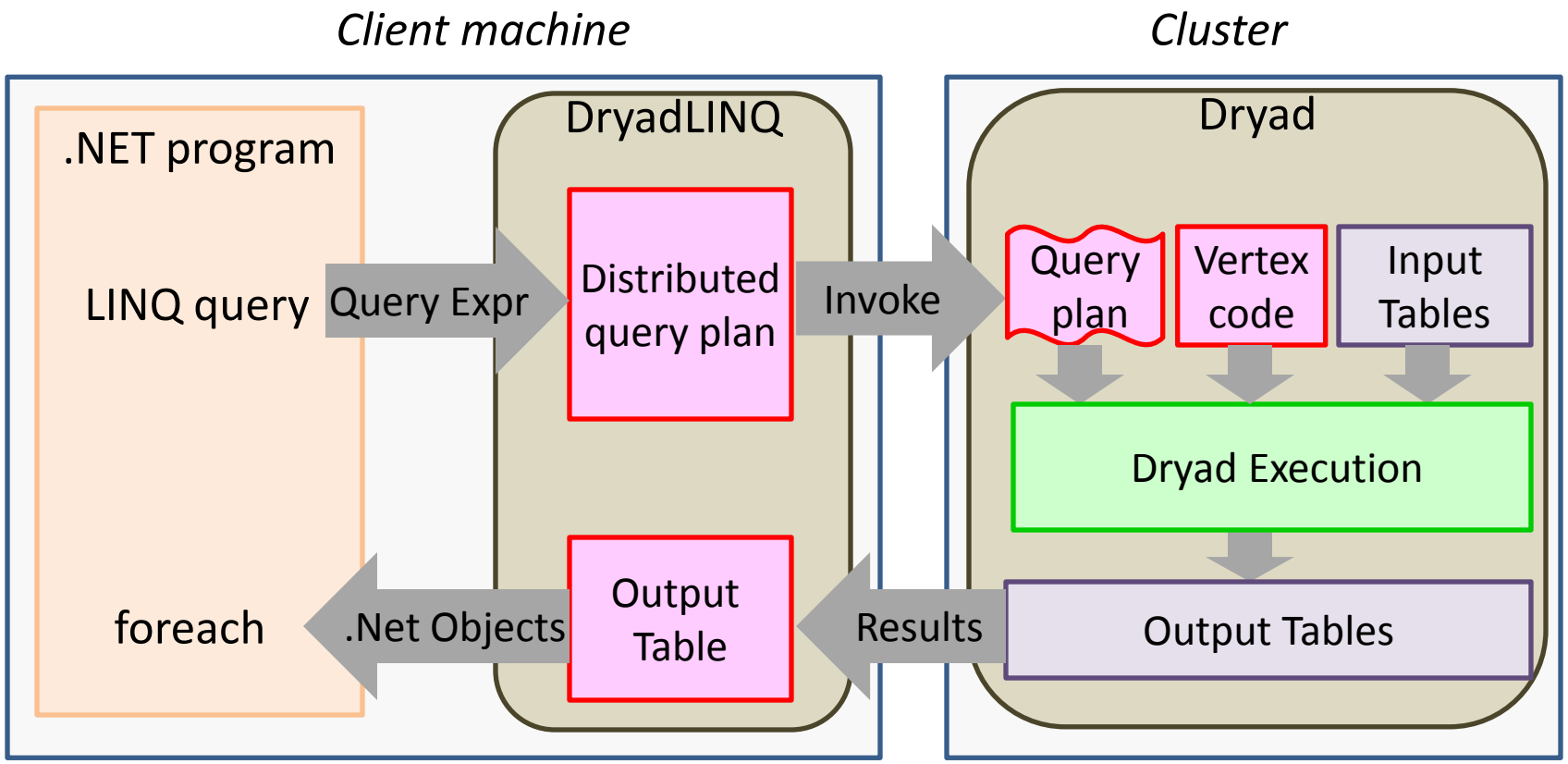
- Provides a general, flexible execution layer
  - Dataflow graph as the computation model
    - Can be modified by runtime optimizations
  - Higher language layer supplies graph, vertex code, serialization code, hints for data locality, …
- Automatically handles distributed execution
  - Distributes code, routes data
  - Schedules processes on machines near data
  - Masks failures in cluster and network
  - Fair scheduling of concurrent jobs

LINQ Framework

- Extremely open and extensible



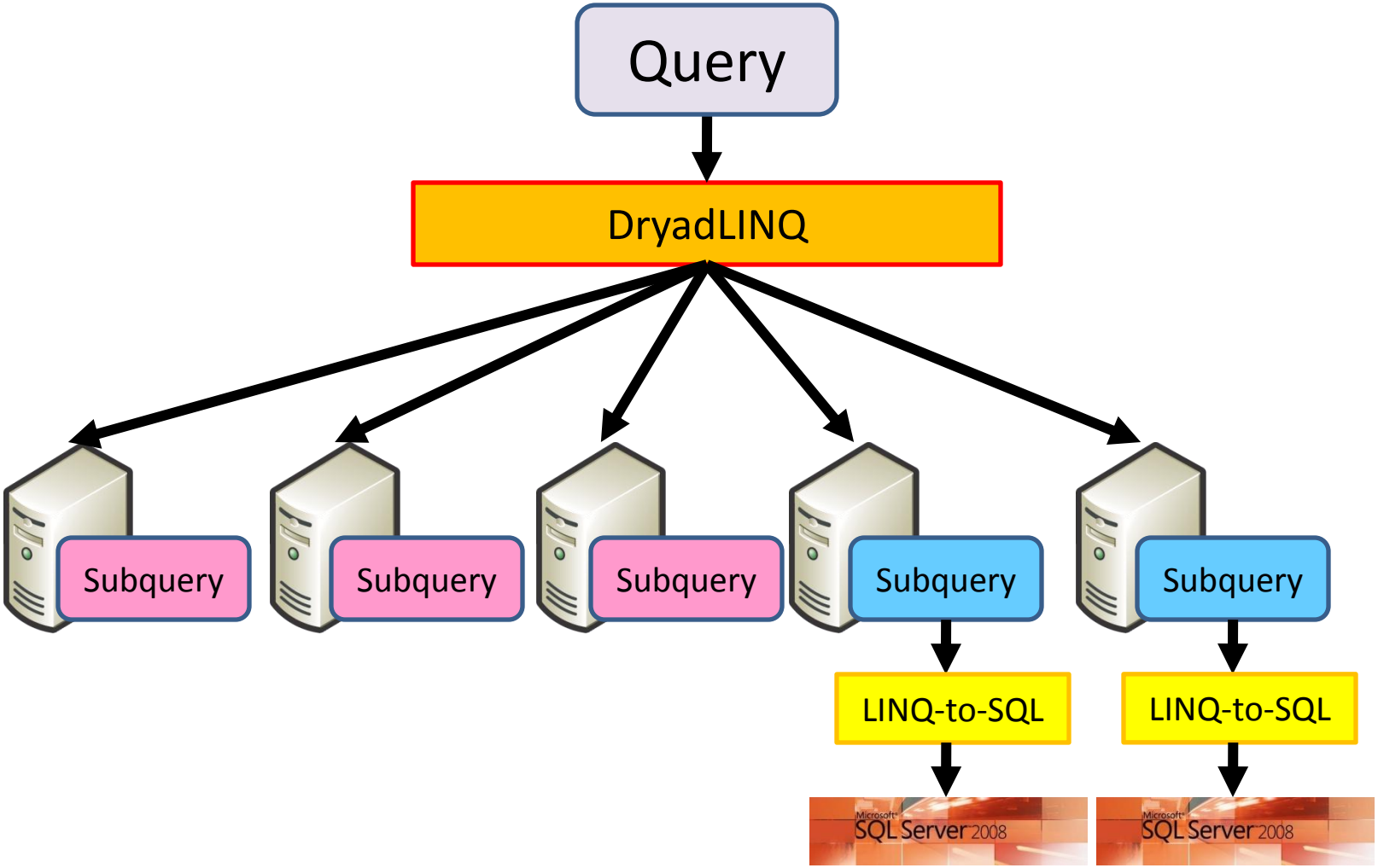
DryadLINQ System Architecture



DryadLINQ

- Distributed execution plan generation
  - Static optimizations: pipelining, eager aggregation, etc.
  - Dynamic optimizations: data-dependent partitioning, dynamic aggregation, etc.
- Vertex runtime
  - Single machine (multi-core) implementation of LINQ
  - Vertex code that runs on vertices
  - Data serialization code
  - Callback code for runtime dynamic optimizations
  - Automatically distributed to cluster machines

Combining with SQL



Applications

- Dryad has been in production use since 2006
  - The execution engine for Bing analytics
  - Runs on >> 10^4 machines
  - Runs on clusters with > 3000 machines
  - Processes many petabytes of data daily
- Dryad is the execution engine for DryadLINQ

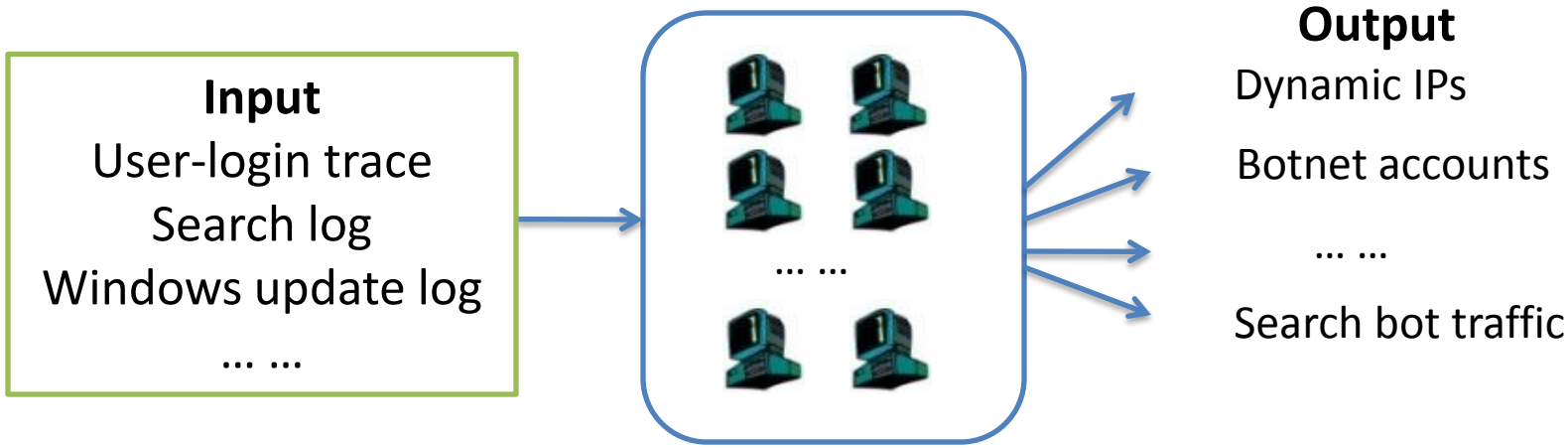
Examples of DryadLINQ Applications

- Data mining
  - Analysis of service logs for network security
  - Analysis of Windows Watson/SQM data
  - Cluster monitoring and performance analysis
- Graph analysis
  - Accelerated Page-Rank computation
  - Road network shortest-path preprocessing
- Image processing
  - Image indexing
  - Decision tree training
  - Epitome computation
- Simulation
  - light flow simulations for next-generation display research
  - Monte-Carlo simulations for mobile data
- eScience
  - Machine learning platform for health solutions
  - Astrophysics simulation

Network Security

Yinglian Xie, Fang Yu et al

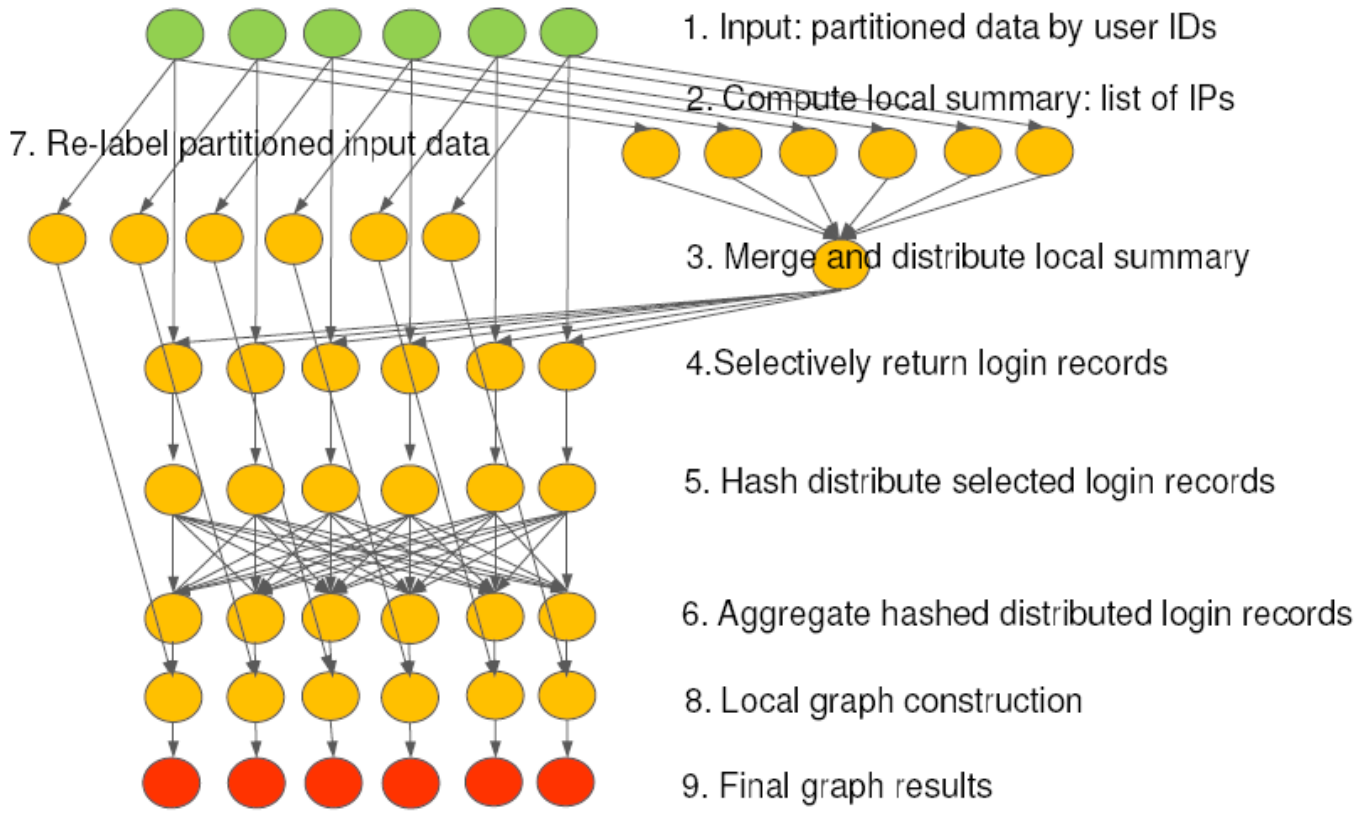
- **Large-scale data mining for network security**
  - Analyze huge amount of service logs: TBs of data
  - Find correlated activities and global patterns
- **Current applications**
  - **UDMap:** automatic dynamic IP address identification
  - **BotGraph:** large-scale spamming botnet detection
  - **HostTracker:** infer host-IP binding in the Internet
  - **SBotMiner:** large-scale search-bot identification



BotGraph: Spamming Botnet Detection

- **A graph-based approach to attack detection**
  - Construct user-user graph to capture bot-account correlations
    - Input data: Hotmail user login data
  - Identify 26M bot-accounts with a low false positive rate with two month data
- **DryadLINQ-based implementation**
  - Graph construction/analysis: 10^8 nodes and 10^{11} edges
  - Graph construction : 1.5 hours
  - Connected component analysis: process a graph of 8.6 billion edges in 7 minutes

Graph Construction

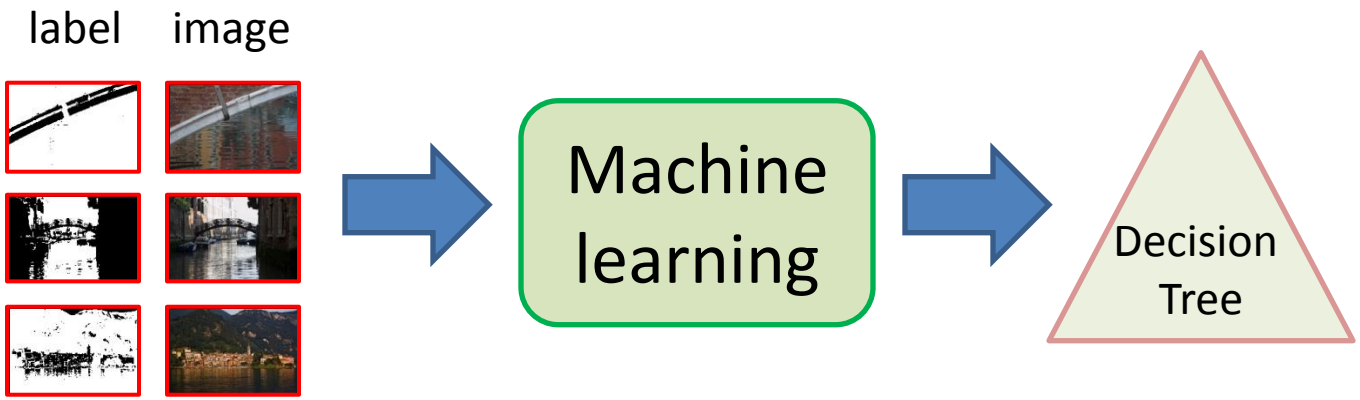


- Total I/O read/write: 11.5 TB
- Total processing vertices: 1600
- Different types of data joins and broadcasts

Decision Tree Training

Mihai Budiu, Jamie Shotton et al

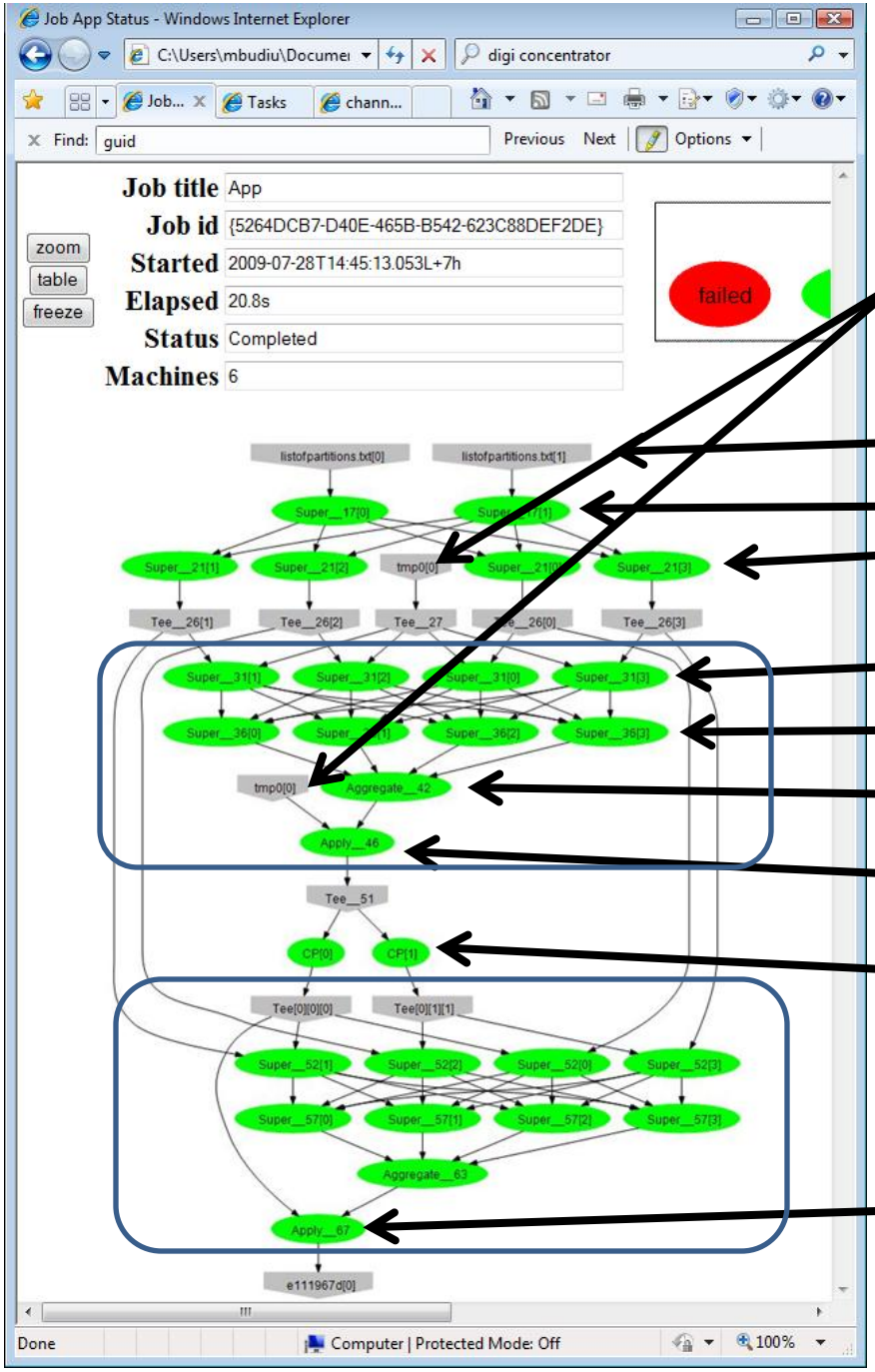
Learn a decision tree to classify pixels in a large set of images



1M images x
10,000 pixels x
2,000 features x
2^{21} tree nodes

Complexity $>10^{20}$ objects

Sample Execution plan



Initial empty tree

Image inputs, partitioned

Read, preprocess

Redistribute

Histogram

Regroup histograms on node

Compute new tree layer

Compute new tree

Broadcast new tree

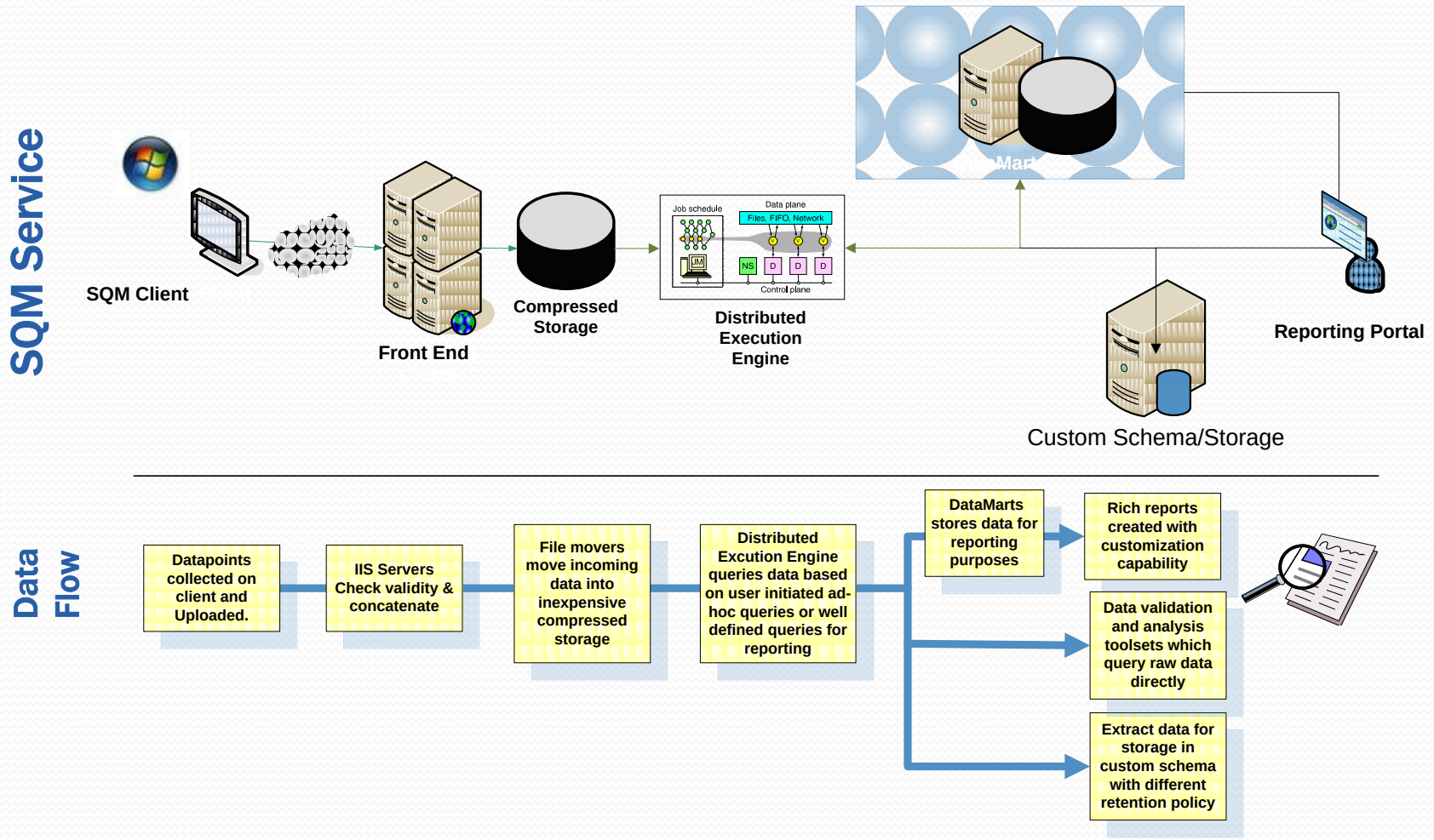
Final tree

Application Details

- Workflow = 37 DryadLINQ jobs
- Checkpoint results between jobs
- 12 hours running time on 235 machines
- More than 100,000 processes
- More than 100 days of CPU time
- Recovers from several failures daily
- 34,000 lines of .NET code

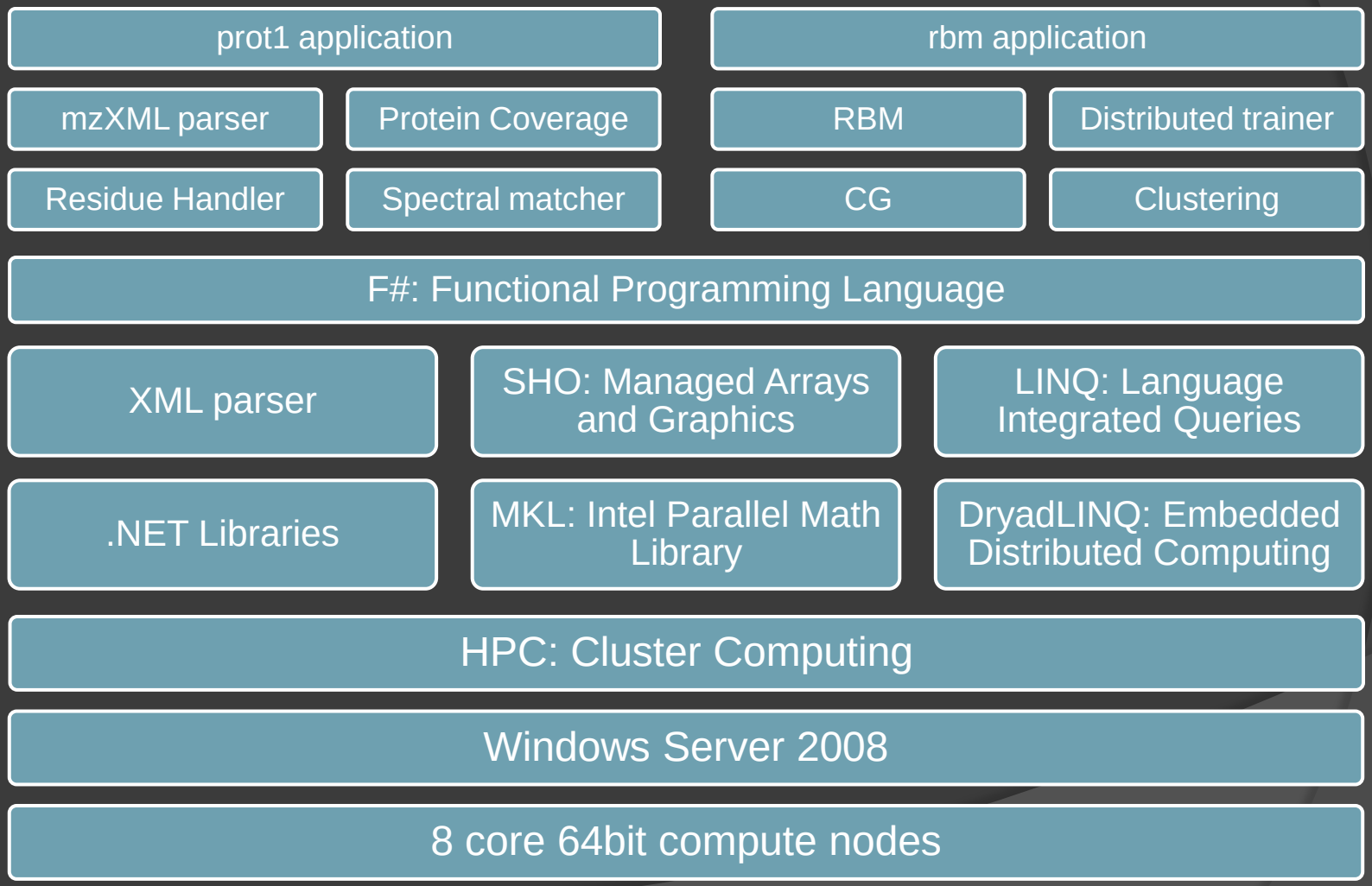
Windows SQM Data Analysis

Michal Strehovsky, Sivarudrappa Mahesh et al



Machine Learning Platform

Dave Wecker et al



Summary

- Single unified programming environment
  - Unified data model and programming language
  - Direct access to IDE and libraries
  - Powerful and expressive execution engine and programming model
- An open and extensible system
  - Very easy to write a new LINQ provider for your app domain
    - Existing ones: LINQ-to-XML, LINQ-to-SQL, …
  - Dryad/DryadLINQ scales out all of them!