

Systems for Data-Intensive Cluster Computing

Mihai Budiu

Microsoft Research, Silicon Valley

ALMADA Summer School

Moscow, Russia

August 2 & 6, 2013



About Myself



- <http://budiu.info/work>
- Senior Researcher at Microsoft Research in Mountain View, California
- Ph.D. in Computer Science from Carnegie Mellon, 2004
- Worked on reconfigurable hardware, compilers, security, cloud computing, performance analysis, data visualization

Lessons to remember



Will use this symbol throughout the presentation to point out fundamental concepts and ideas.

This talk is not about specific software artifacts. It is a talk about principles, illustrated with some existing implementations.



BIG DATA

500 Years Ago



Tycho Brahe
(1546-1601)



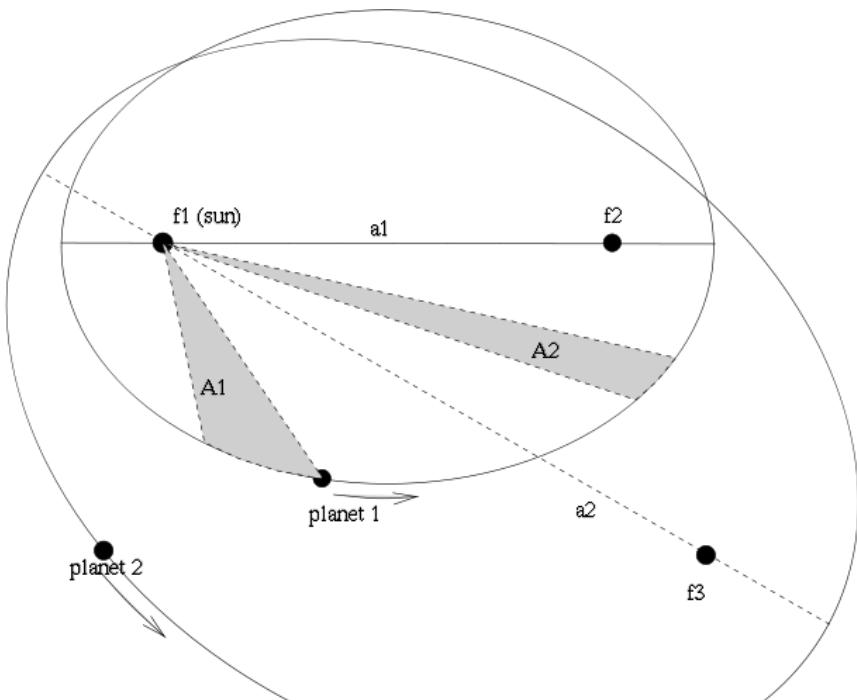
Johannes Kepler
(1571-1630)

The Laws of Planetary Motion

TYCHONIS BRAHE LIB. I
TABVLA PARALLAXIVM SOLARIVM IN CIRCULO VERTICALI
AD EIVS A TERRA REMOTIONEM TRIPLEM.

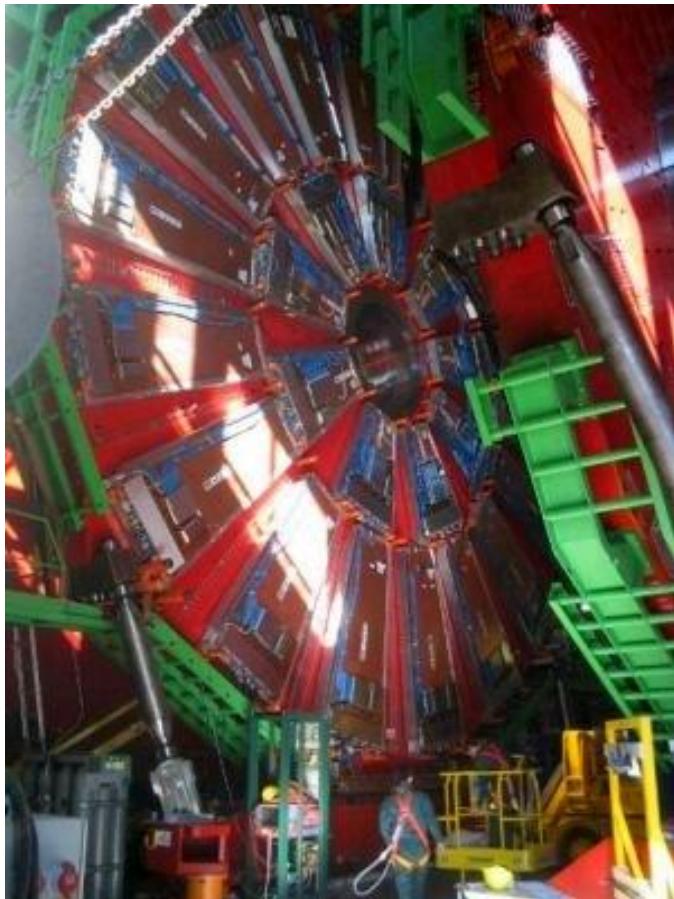
A.	Maxi:			Mediā			Min:			$\frac{\Delta}{\pi}$			Maxi:			Mediā			$\frac{\Delta}{\pi}$			Maxi:			Mediā			$\frac{\Delta}{\pi}$		
	I.	II.	III.	I.	II.	G.	I.	II.	G.	I.	II.	G.	I.	II.	G.	I.	II.	G.	I.	II.	G.	I.	II.	G.	I.	II.	G.	I.	II.	G.
0	2	54	3	0	3	7	30	2	30	2	36	2	42	60	1	27	1	30	1	33	1	39	1	42	1	45	1	48	1	51
1	2	54	3	0	3	7	31	2	28	2	34	2	40	61	1	25	1	28	1	31	1	38	1	41	1	44	1	47	1	50
2	2	54	3	0	3	7	32	2	27	2	32	2	38	62	1	23	1	25	1	28	1	37	1	40	1	43	1	46	1	49
3	2	54	3	0	3	7	33	2	25	2	30	2	37	63	1	19	1	22	1	25	1	35	1	38	1	41	1	44	1	47
4	2	53	2	59	3	6	34	2	23	2	29	2	35	64	1	16	1	19	1	22	1	32	1	35	1	38	1	41	1	44
5	2	53	2	59	3	6	35	2	22	2	27	2	33	65	1	13	1	16	1	19	1	30	1	33	1	36	1	39	1	42
6	2	53	2	59	3	6	36	2	20	2	25	2	31	66	1	10	1	14	1	17	1	29	1	32	1	35	1	38	1	41
7	2	52	2	58	3	5	37	2	18	2	23	2	29	67	1	8	1	11	1	14	1	28	1	31	1	34	1	37	1	40
8	2	52	2	58	3	5	38	2	17	2	21	2	27	68	1	5	1	8	1	11	1	27	1	30	1	33	1	36	1	39
9	2	51	2	57	3	4	39	2	15	2	19	2	25	69	1	2	1	5	1	8	1	26	1	29	1	32	1	35	1	38
10	2	51	2	57	3	4	40	2	13	2	18	2	23	70	0	59	1	2	1	5	1	8	1	25	1	28	1	31	1	34
11	2	50	2	56	3	3	41	2	11	2	16	2	21	71	0	56	0	59	0	62	1	24	1	27	1	30	1	33	1	36
12	2	50	2	56	3	3	42	2	9	2	14	2	19	72	0	53	0	56	0	59	1	23	1	26	1	29	1	32	1	35
13	2	49	2	55	3	2	43	2	7	2	12	2	17	73	0	50	0	53	0	56	1	22	1	25	1	28	1	31	1	34
14	2	48	2	54	3	1	44	2	5	2	9	2	15	74	0	47	0	49	0	52	1	21	1	24	1	27	1	30	1	33
15	2	48	2	54	3	0	45	2	3	2	7	2	12	75	0	45	0	46	0	48	1	20	1	23	1	26	1	29	1	32
16	2	47	2	53	2	59	46	2	1	2	5	2	10	76	0	42	0	43	0	45	1	19	1	22	1	25	1	28	1	31
17	2	46	2	52	2	58	47	1	39	2	3	2	8	77	0	39	0	40	0	41	1	18	1	21	1	24	1	27	1	30
18	2	46	2	51	2	58	48	1	37	2	0	2	5	78	0	36	0	37	0	38	1	17	1	20	1	23	1	26	1	29
19	2	45	2	50	2	57	49	1	35	1	58	2	3	79	0	33	0	34	0	35	1	16	1	19	1	22	1	25	1	28
20	2	44	2	50	2	56	50	1	32	1	56	2	0	80	0	30	0	31	0	32	1	15	1	18	1	21	1	24	1	27
21	2	43	2	49	2	55	51	1	30	1	54	1	58	81	0	27	0	28	0	29	1	14	1	17	1	20	1	23	1	26
22	2	42	2	48	2	53	52	1	47	1	51	1	55	82	0	24	0	25	0	26	1	13	1	16	1	19	1	22	1	25
23	2	41	2	46	2	52	53	1	45	1	48	1	52	83	0	21	0	21	0	22	1	12	1	15	1	18	1	21	1	24
24	2	40	2	45	2	50	54	1	43	1	46	1	50	84	0	18	0	18	0	19	1	11	1	14	1	17	1	20	1	23
25	2	38	2	44	2	49	55	1	40	1	43	1	47	85	0	15	0	15	0	16	1	10	1	13	1	16	1	19	1	22
26	2	37	2	43	2	47	56	1	35	1	41	1	45	86	0	12	0	12	0	13	1	9	1	12	1	15	1	18	1	21
27	2	35	2	41	2	45	57	1	35	1	39	1	42	87	0	9	0	9	0	9	1	8	1	11	1	14	1	17	1	20
28	2	33	2	39	2	44	58	1	32	1	36	1	39	88	0	6	0	6	0	6	1	5	1	8	1	11	1	14	1	17
29	2	31	2	37	2	43	59	1	30	1	33	1	36	89	0	3	0	3	0	3	1	2	1	5	1	8	1	11	1	14
30	2	30	2	36	2	42	60	1	27	1	30	1	33	90	0	0	0	0	0	0	1	2	1	5	1	8	1	11	1	14

Tycho's measurements

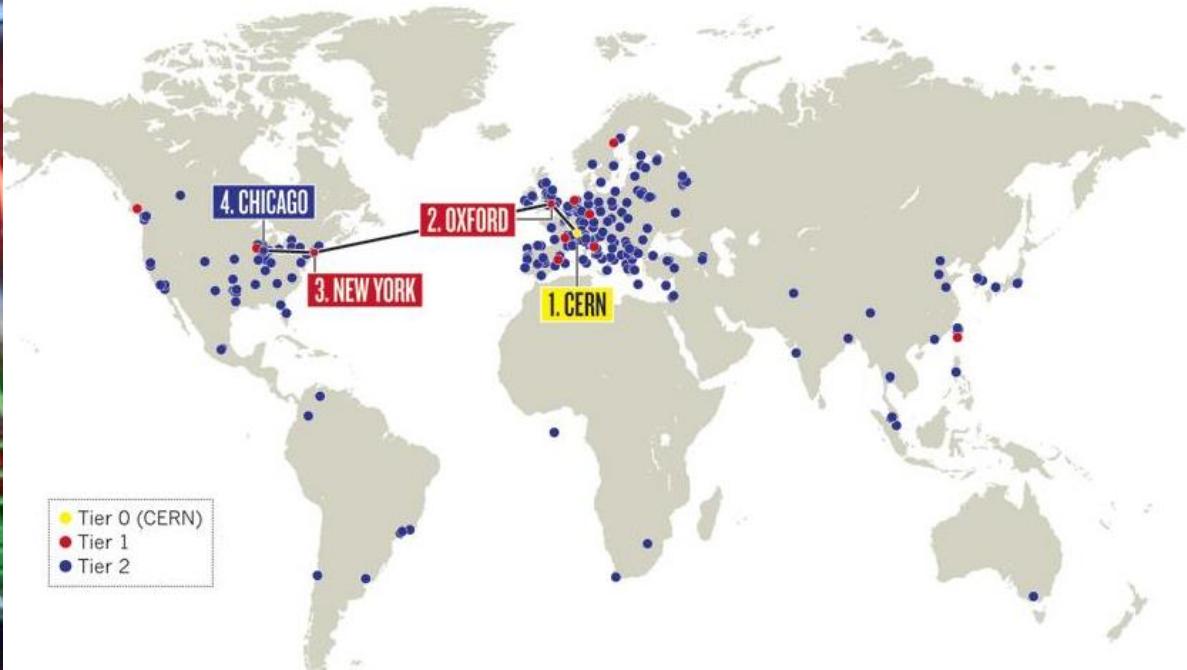


Kepler's laws

The Large Hadron Collider

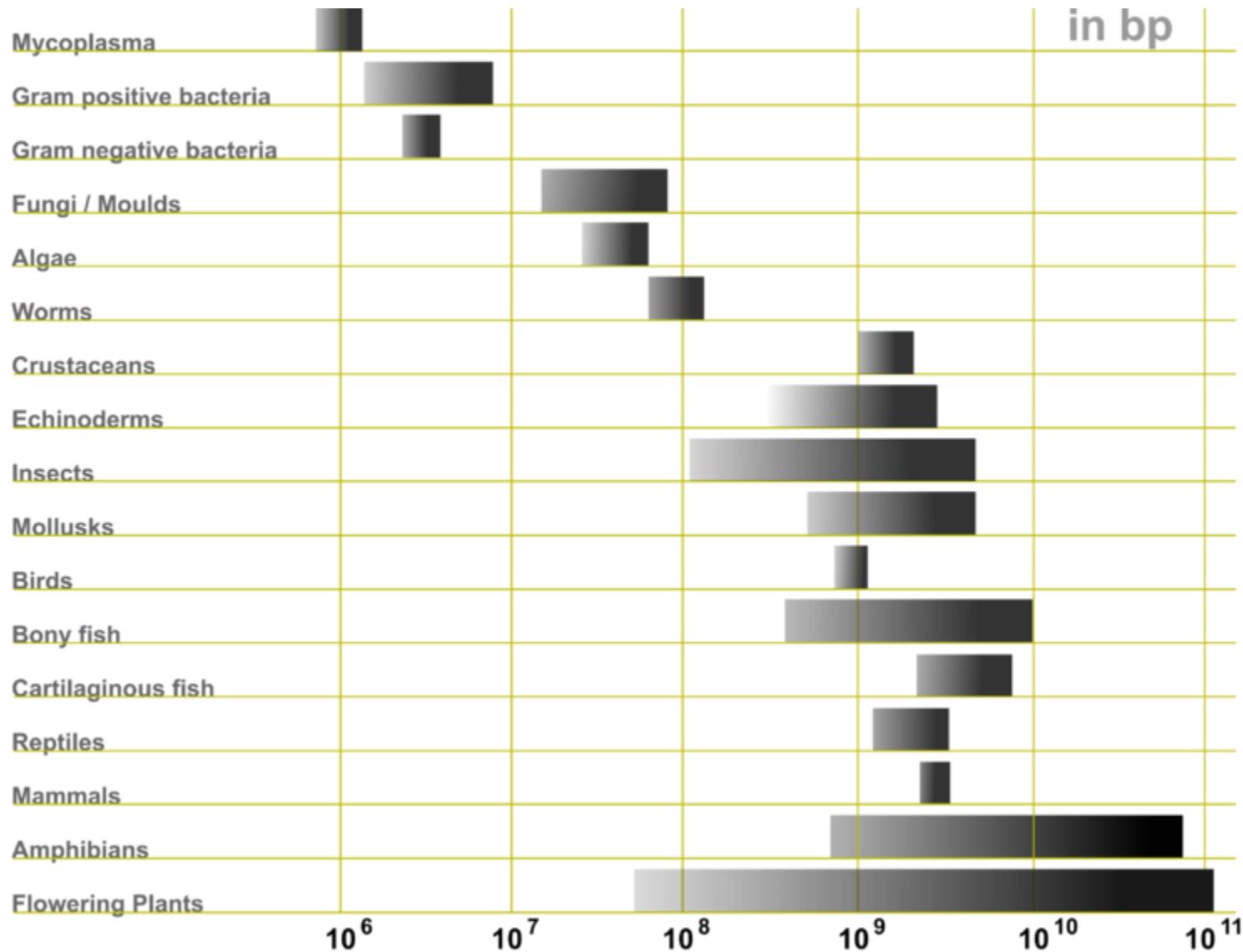


25 PB/year

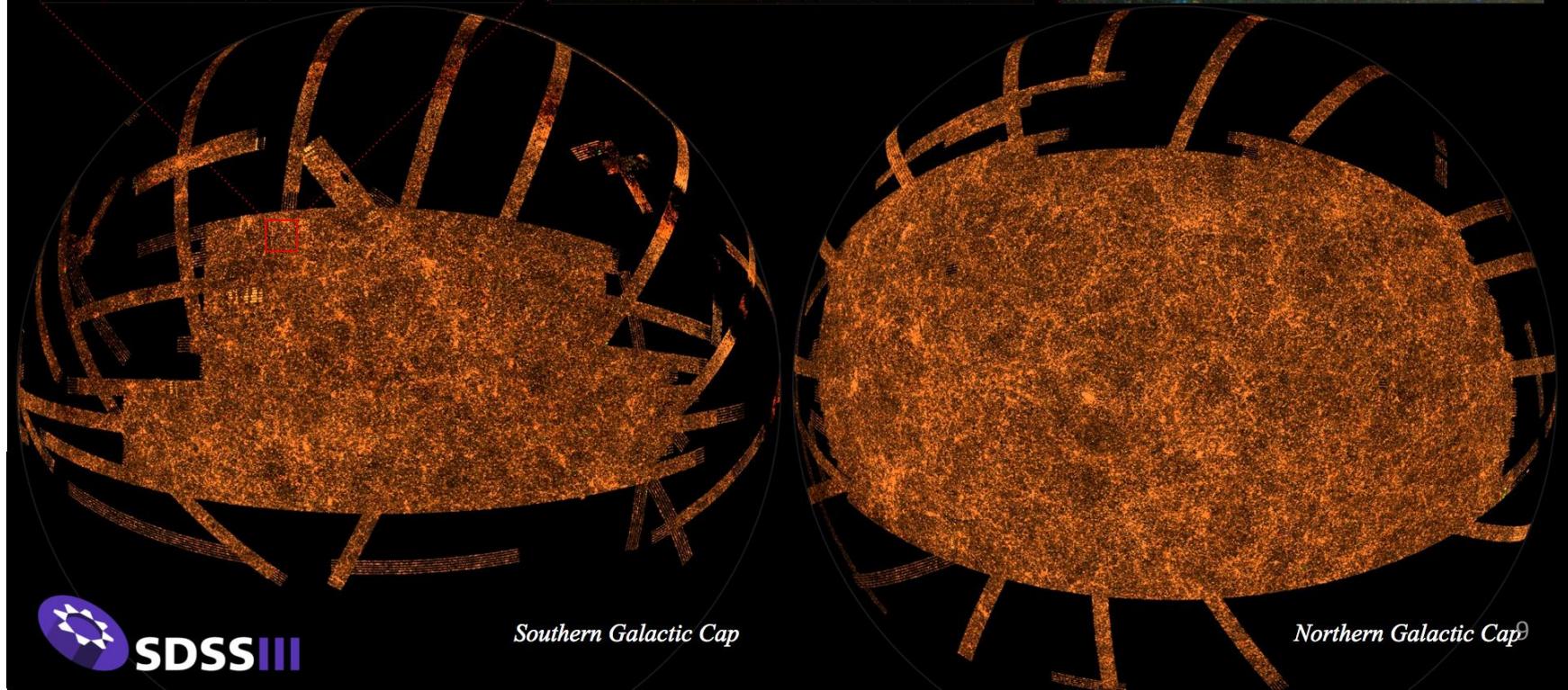
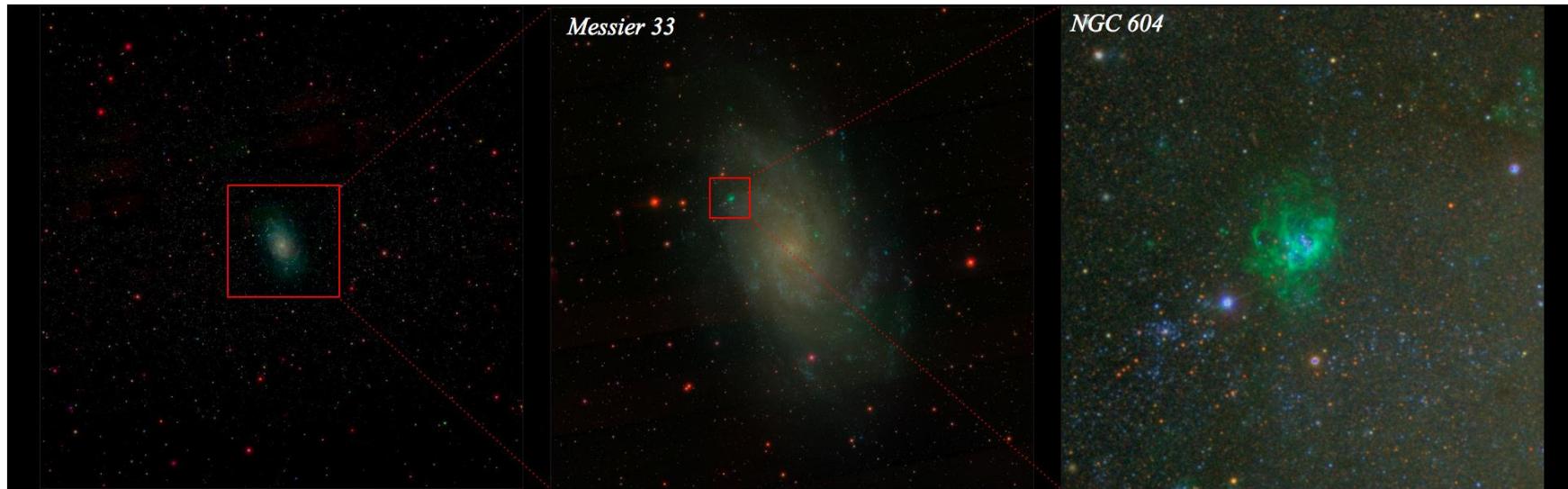


WLHC Grid:
200K computing cores

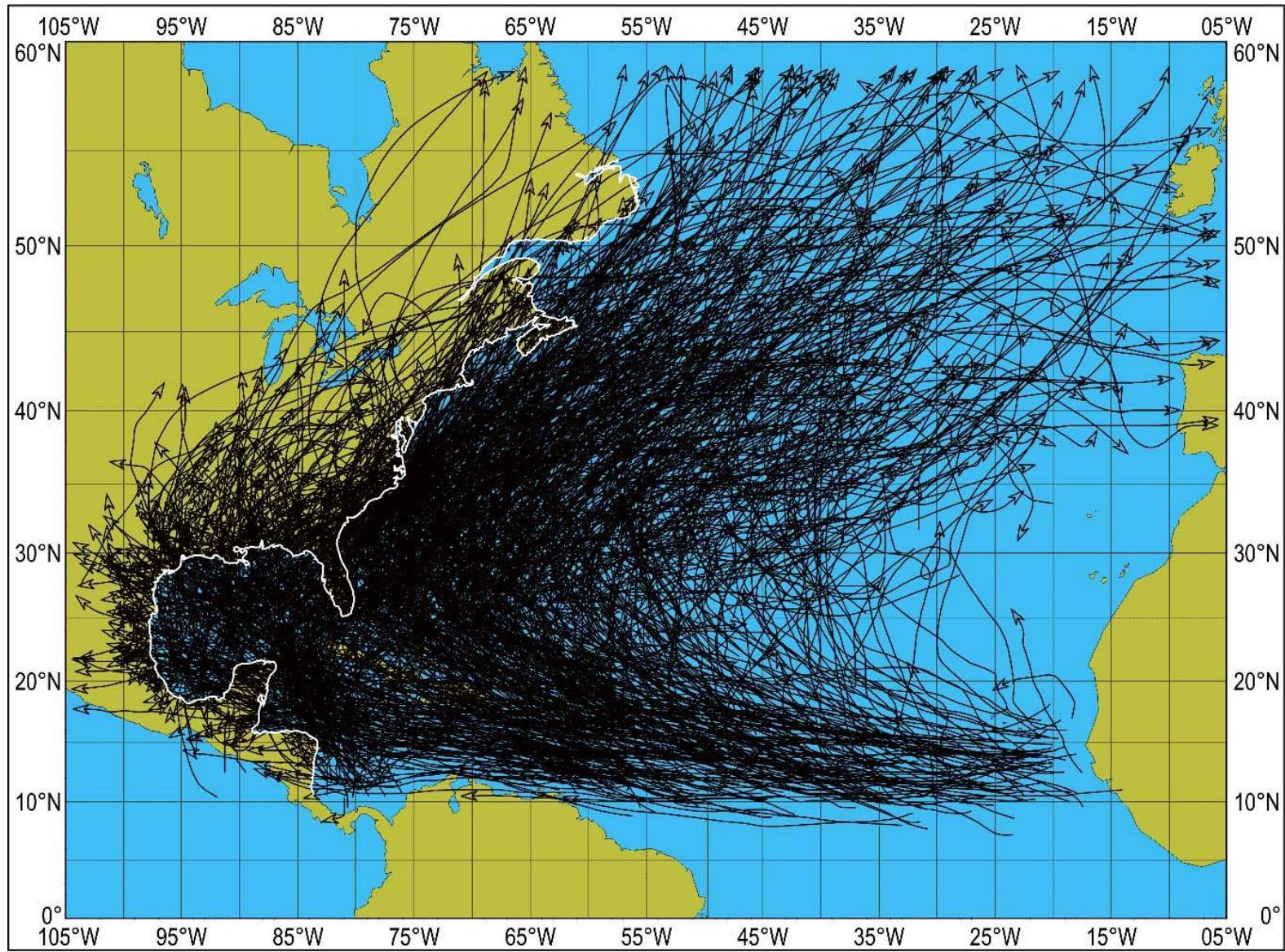
Genetic Code



Astronomy

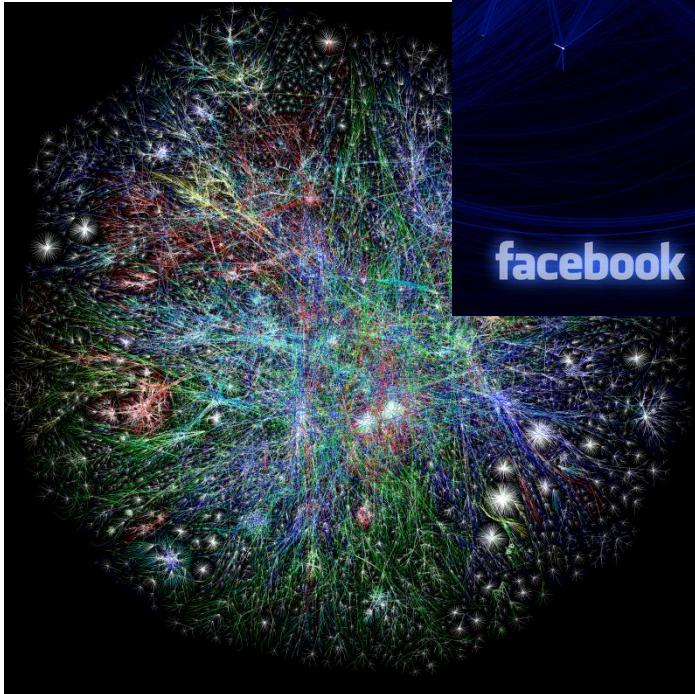


Weather



NORTH ATLANTIC TROPICAL STORMS AND HURRICANES, 1851-2004 (1325 STORMS)
NOAA

The Webs



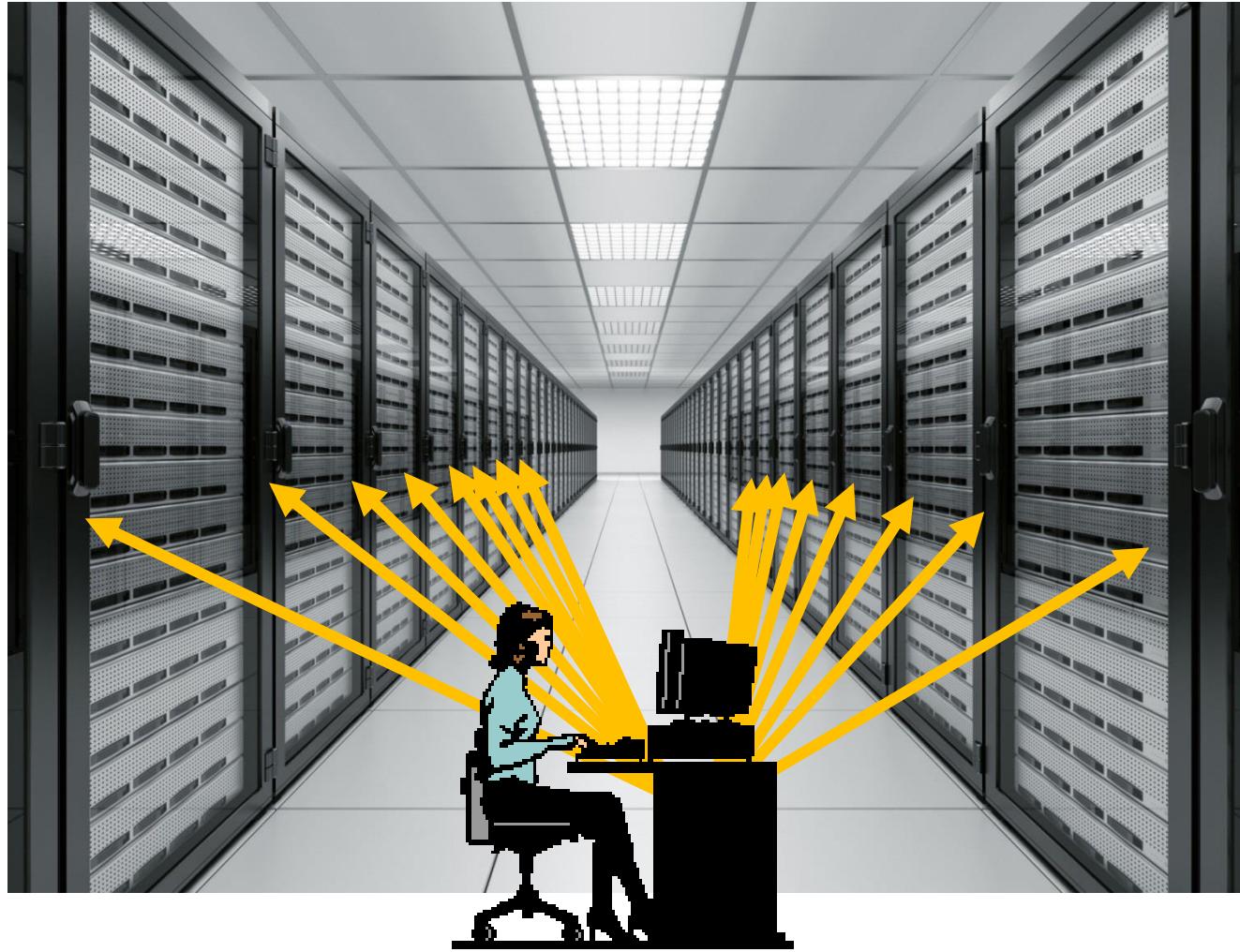
Facebook friends graph

Internet

Big Data



Big Computers



Presentation Structure

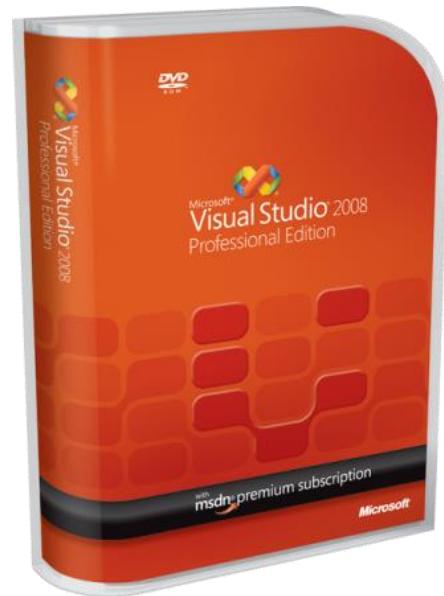
- Friday
 - Introduction
 - LINQ: a language for data manipulation; homework
 - A Software Stack for Data-Intensive Manipulation (Part 1)
 - Hardware, Management, Cluster, Storage, Execution
- Tuesday
 - Thinking in Parallel
 - A Software Stack for Data-Intensive Manipulation (Part 2)
 - Language, Application
 - Conclusions

LINQ: LANGUAGE-INTEGRATED QUERY



LINQ

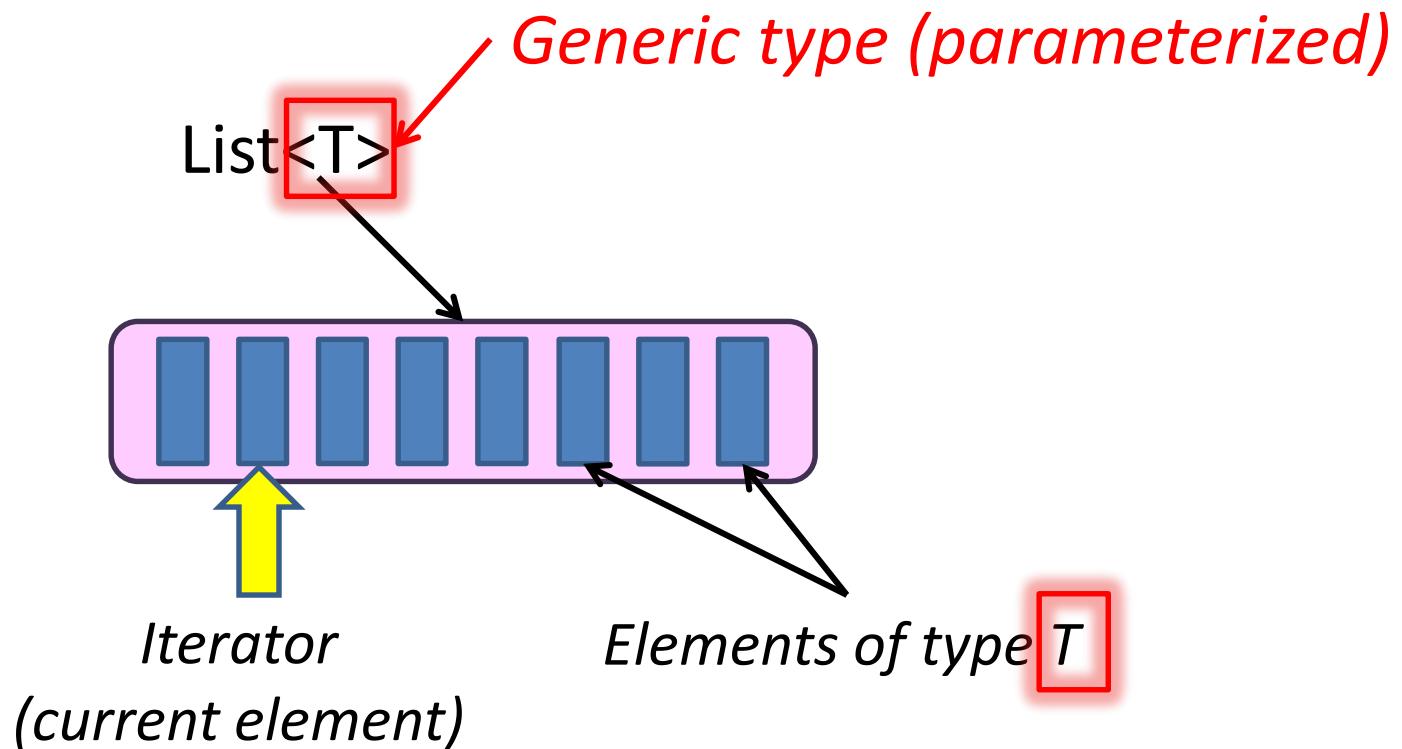
- Language extensions to .Net
 - (We will use C#)
- Strongly typed query language
 - Higher-order query operators
 - User-defined functions
 - Arbitrary .Net data types



LINQ Availability

- On Windows: Visual Studio 2008 and later, including VS Express (free)
 - <http://www.microsoft.com/visualstudio/eng/downloads#d-2012-editions>
- On other platforms: open-source  **mono**
 - http://mono-project.com/Main_Page
- Java Streams --- planned in Java 8 --- are similar

Collections



LINQ = .Net+ Queries

```
Collection<T> collection;
```

```
bool IsLegal(Key);
```

```
string Hash(Key);
```

```
var results = collection.
```

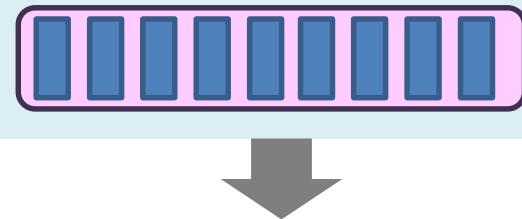
```
    Where(c => IsLegal(c.key)).
```

```
    Select(c => new { Hash(c.key), c.value});
```



Essential LINQ Summary

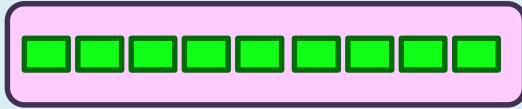
Input



Where (filter)



Select (map)



GroupBy



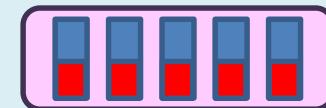
OrderBy (sort)



Aggregate (fold)

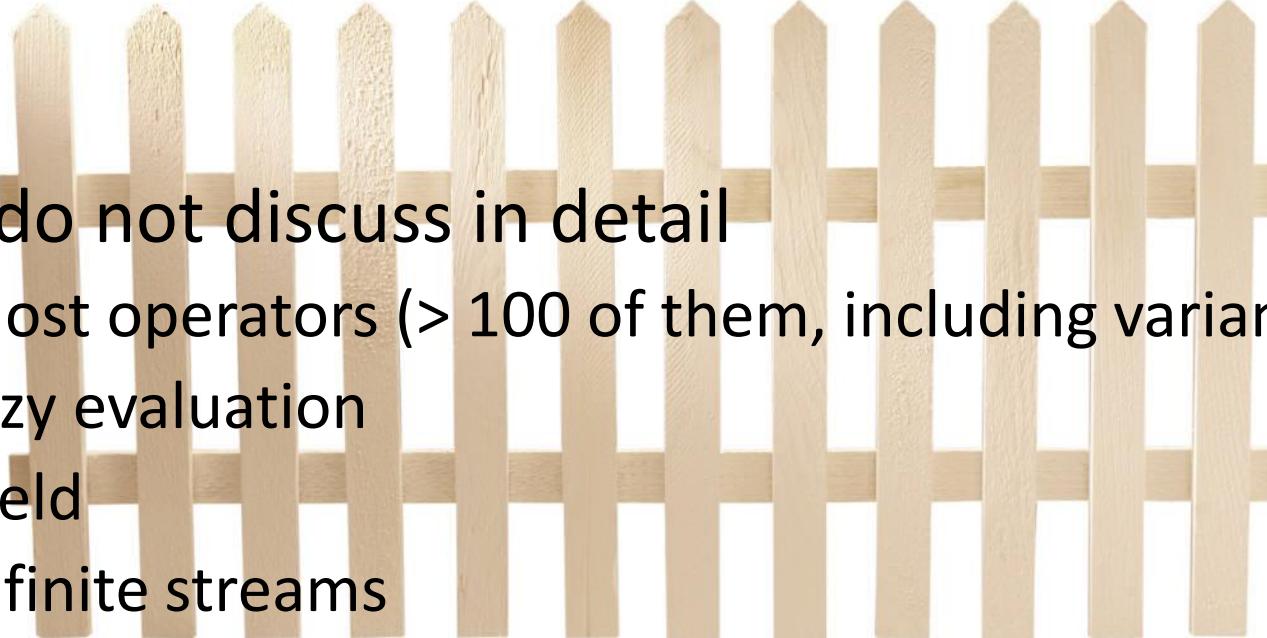


Join



Tutorial Scope

- We only discuss essential LINQ

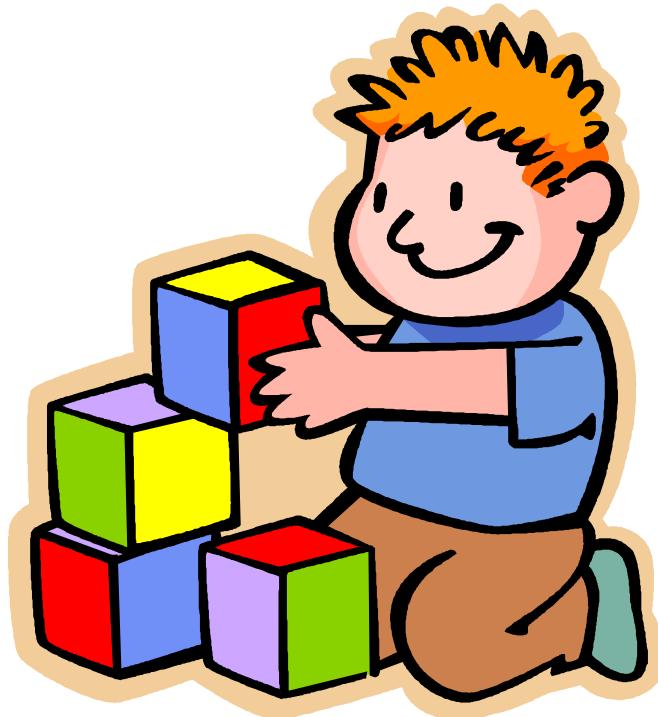
- 
- We do not discuss in detail
 - most operators (> 100 of them, including variants)
 - lazy evaluation
 - yield
 - Infinite streams
 - IQueryable, IObservable

Running the Code



- Code can be downloaded at
<http://budiu.info/almada-code.zip>
- Includes the code on all the following slides
(file Slides.cs)
- Includes the homework exercises (details later)
- Includes testing data for the homework
- Contains a Visual Studio Solution file
TeachingLinq.sln
- compileMono.bat can be used for Mono

Useful .Net Constructs



Tuples

```
 Tuple<int, int> pair = new  
   Tuple<int, int>(5, 3);  
  
 Tuple<int, string, double> triple =  
   Tuple.Create(1, "hi", 3.14);  
  
 Console.WriteLine("{0} {1} {2}",  
   pair, triple, pair.Item1);
```

Result: (5, 3) (1, hi, 3.14) 5

Anonymous Functions



- $\text{Func}\langle I, O \rangle$
 - Type of function with
 - input type I
 - output type O
- $x \Rightarrow x + 1$
 - Lambda expression
 - Function with argument x , which computes $x+1$

Using Anonymous Functions

```
Func<int, int> inc = x => x + 1;  
Console.WriteLine(inc(10));
```

Result: 11

Functions as First-Order Values

```
static T Apply<T>(  
    T argument,  
    Func<T, T> operation)  
{  
    return operation(argument);  
}
```

```
Func<int, int> inc = x => x + 1;  
Console.WriteLine(Apply(10, inc));
```

Result: 11

Multiple Arguments

```
Func<int, int, int> add =  
(x, y) => x + y;
```

```
Console.WriteLine(add(3,2));
```

Result: 5

Collections



IEnumerable<T>

```
IEnumerable<int> data =  
    Enumerable.Range(0, 10);  
  
foreach (int n in data)  
    Console.WriteLine(n);
```

```
Program.Show(data);
```



*Helper function I wrote
to help you debug.*

Result: 0,1,2,3,4,5,6,7,8,9

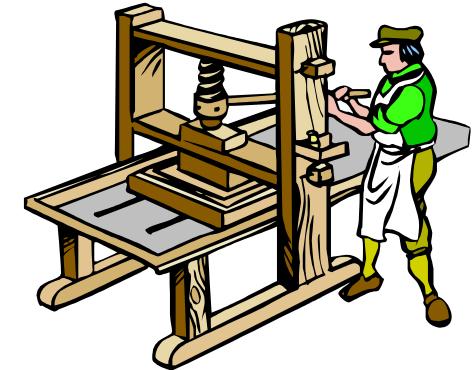
IEnumerables everywhere

- `List<T>` : `IEnumerable<T>`
- `T[]` : `IEnumerable<T>`
- `string` : `IEnumerable<char>`
- `Dictionary<K,V>` :
`IEnumerable<KeyValuePair<K,V>>`
- `ICollection<T>` : `IEnumerable<T>`
- Infinite streams
- User-defined



Printing a Collection

```
static void Print<T>(  
    IEnumerable<T> data)  
{  
    foreach (var v in data)  
        Console.Write("{0} ", v);  
    Console.WriteLine();  
}
```



LINQ Collection Operations

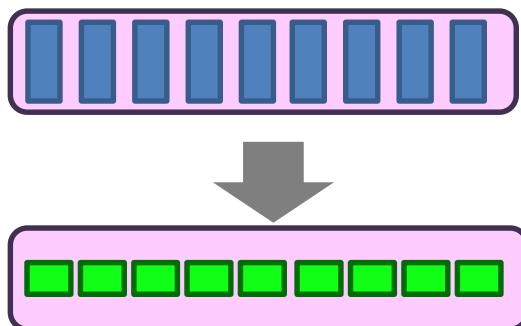


Select

Input and output are both collections.

```
IEnumerable<int> result =  
    data.Select(d => d + 1);  
  
Show(result);
```

Note: everyone else calls this function “Map”



Data: 0,1,2,3,4,5,6,7,8,9

Result: 1,2,3,4,5,6,7,8,9,10

Select can transform types

```
var result = data.Select(  
    d => d > 5);  
  
Show(result);
```

Result: False, False, False, False, False, False, True, True, True, True

Separation of Concerns

```
var result = data.Select(d => d>5);
```

Operator applies to *any* collection

User-defined function (UDF)
Transforms the elements



Chaining operators

```
var result = data  
    .Select(d => d * d)  
    .Select(d => Math.Sin(d))  
    .Select(d => d.ToString());
```



All these are collections.

```
Result: "0.00","0.84","-0.76","0.41","-0.29","-0.13","-0.99","-0.95","0.92","-0.63"
```

Generic Operators

IEnumerable<Dest>

Output type

Select<Src, Dest>(

this IEnumerable<Src>,

Input type

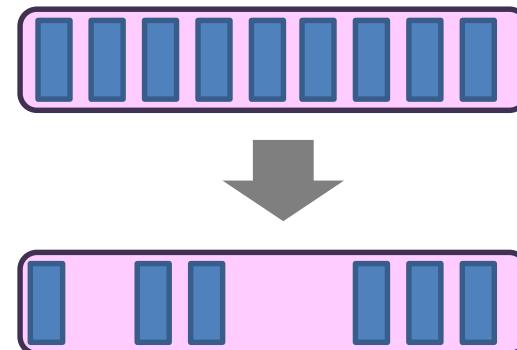
Func<Src, Dest>)

Transformation function

Filtering with Where

```
var result = data.Where(  
    d => d > 5);  
  
Show(result);
```

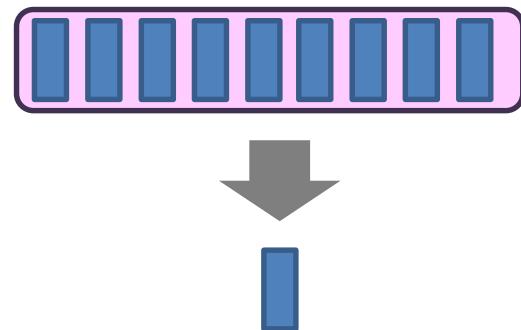
Result: 6,7,8,9



Aggregations

```
var ct = data.Count();  
var sum = data.Sum();  
Console.WriteLine("{0} {1}", ct, sum);
```

Result: 10 45



General Aggregations

```
var sum = data.Aggregate(  
    (a, b) => a + b);  
Console.WriteLine("{0}", sum);
```

Result: 45

$((((((((0 + 1) + 2) + 3) + 4) + 5) + 6) + 7) + 8) + 9$

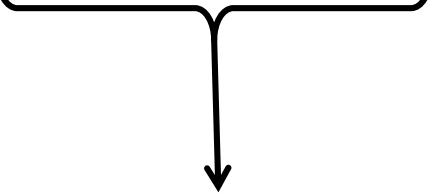
Aggregations with Seeds

```
var sum = data.Aggregate(  
    "X", // seed  
    (a, b) => string.Format(  
        "( {0} + {1} )", a, b));  
  
Console.WriteLine("{0}", sum);
```

Result: “((((((((X + 0) + 1) + 2) + 3) + 4) + 5) + 6) + 7) + 8) + 9)”

Values can be Collections

```
var lines = new string[]  
    { "First string",  
      "Second string" };  
  
IEnumerable<int> counts =  
    lines.Select(d => d.Count());  
Show(counts);
```



(d as IEnumerable<char>).Count()

Result: 12,13

Flattening Collections with SelectMany

```
IEnumerable<string> words =  
    lines.SelectMany(  
        d => d.Split(' '));
```

```
IEnumerable<string[]> phrases =  
    lines.Select(  
        d => d.Split(' '));
```

SelectMany

```
var lines = new string[]  
    { "First string",  
      "Second string" };  
lines.SelectMany(d => d.Split(' '));
```

Result: { "First", "string", "second", "string" }

```
lines.Select(      d => d.Split(' '));
```

Result: { {"First", "string"}, {"second", "string"} }

This is a collection of collections!

GroupBy

```
IEnumerable<IGrouping<int, int>> groups
```

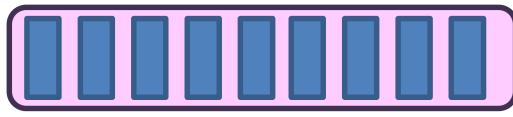
```
= data.GroupBy(d => d%3);
```

Key function

```
Result: 0 => {0,3,6,9}
```

```
    1 => {1,4,7}
```

```
    2 => {2,5,8}
```



IGrouping

```
var result =  
    data.GroupBy(d => d%3)  
        .Select(g => g.Key);
```

Result: 0,1,2

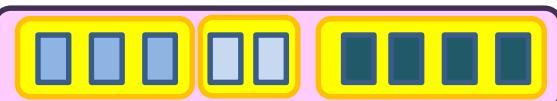
Groups are Nested Collections

IGrouping< TKey, TVals > : IEnumerable< TVals >

```
var result =  
    data.GroupBy(d => d%3)  
        .Select(g => g.Count());
```

LINQ computation on each group

Result: 4,3,3



Sorting

```
var sorted =  
    lines.SelectMany(l=>l.Split(' '))  
        .OrderBy(l => l.Length);  
Show(sorted);
```

Sorting key

Result: "First", "string", "Second", "string"

Joins

```
var L = Enumerable.Range(0, 4);
```

```
var R = Enumerable.Range(0, 3);
```

```
var result = L.Join(R, | => | % 2,  
r => (r + 1) % 2,  
(|, r) => | + r);
```

	L	0	1	2	3	4
R	keys	0	1	0	1	0
0	1		1+0=1		3+0=3	
1	0	0+1=1		2+1=3		4+1=5
2	1		1+2=3		3+2=5	
3	0	0+3=3		2+3=5		4+3=7

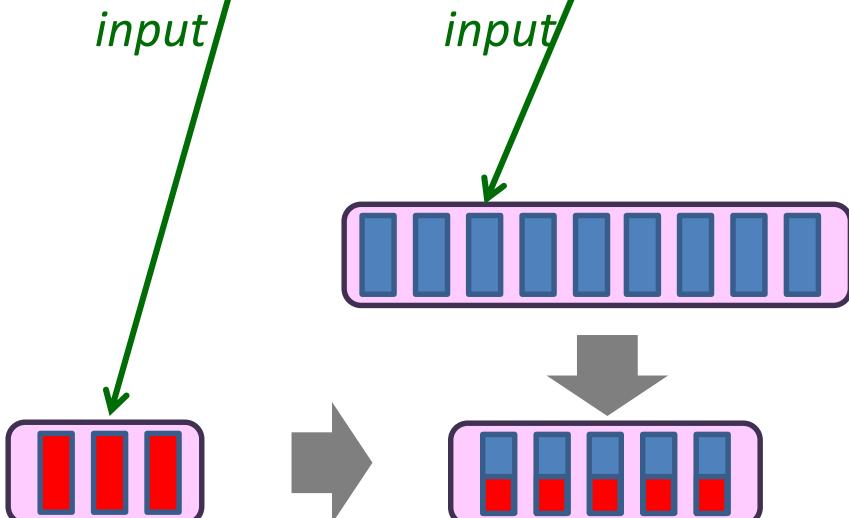
result

Join Example

```
var common =  
    words.Join(data, s => s.Length, d => d,  
               (s, d) => new  
               { length = d, str = s });
```

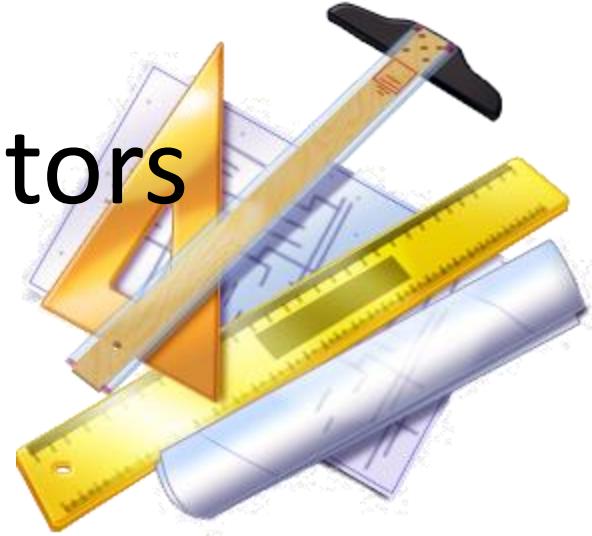
Left input

Right input



Result: {
 { length = 5, str = First }
 { length = 6, str = string }
 { length = 6, str = Second }
 { length = 6, str = string }
}

Other Handy Operators



- `collection.Max()`
- `left.Concat(right)`
- `collection.Take(n)`
- Maximum element
- Concatenation
- First n elements

Take

First few elements in a collection

```
var first5 = data  
    .Select(d => d * d)  
    .Take(5);
```

Result: 0,1,4,9,16

Homework



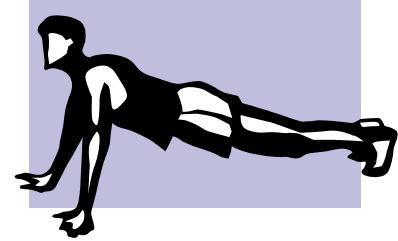
- <http://budiu.info/almada-code.zip>
- You only need the LINQ operations taught in this class
- Solutions should only use LINQ (no loops)
- Fill the 12 functions in class Homework, file Homework.cs
- Testing data given in Program.cs; Main is there
- Calling ShowResults(homework, data) will run your homework on the data (in Program.cs)
- You can use the function Program.Show() to debug/display complex data structures

- Slides.cs: code from all these slides
- Program.cs: main program, and sample testing data
- Tools.cs: code to display complex C# objects
- Homework.cs: skeleton for 12 homework problems

You must implement these 12 functions.

- (Solutions.cs: sample solutions for homework)

Exercises (1)



1. Count the number of words in the input collection (use the supplied `SplitStringIntoWords` function to split a line into words)
2. Count the number of unique (distinct) words in the input collection. “And” = “and” (ignore case).
3. Find the most frequent 10 words in the input collection, and their counts. Ignore case.

Exercises (2)



4. From each line of text extract just the vowels ('a', 'e', 'i', 'o', 'u'), ignoring letter case.
5. Given a list of words find the number of occurrences of each of them in the input collection (ignore case).
6. Generate the Cartesian product of two sets.

Exercises (3)



7. Given a matrix of numbers, find the largest value on each row. (Matrix is a list of rows, each row is a list of values.)
8. A sparse matrix is represented as triples (colNumber, rowNum, value). Compute the matrix transpose. (Indexes start at 0.)
9. Add two sparse matrices.

Exercises (4)



10. Multiply two sparse matrices.

11. Consider a directed graph (V, E) .

Node names are numbers. The graph is given by the set E , a list of edges $\text{Tuple}<\text{int}, \text{int}\rangle$.

Find all nodes reachable in exactly 5 steps starting from node 0.

Exercises (5)



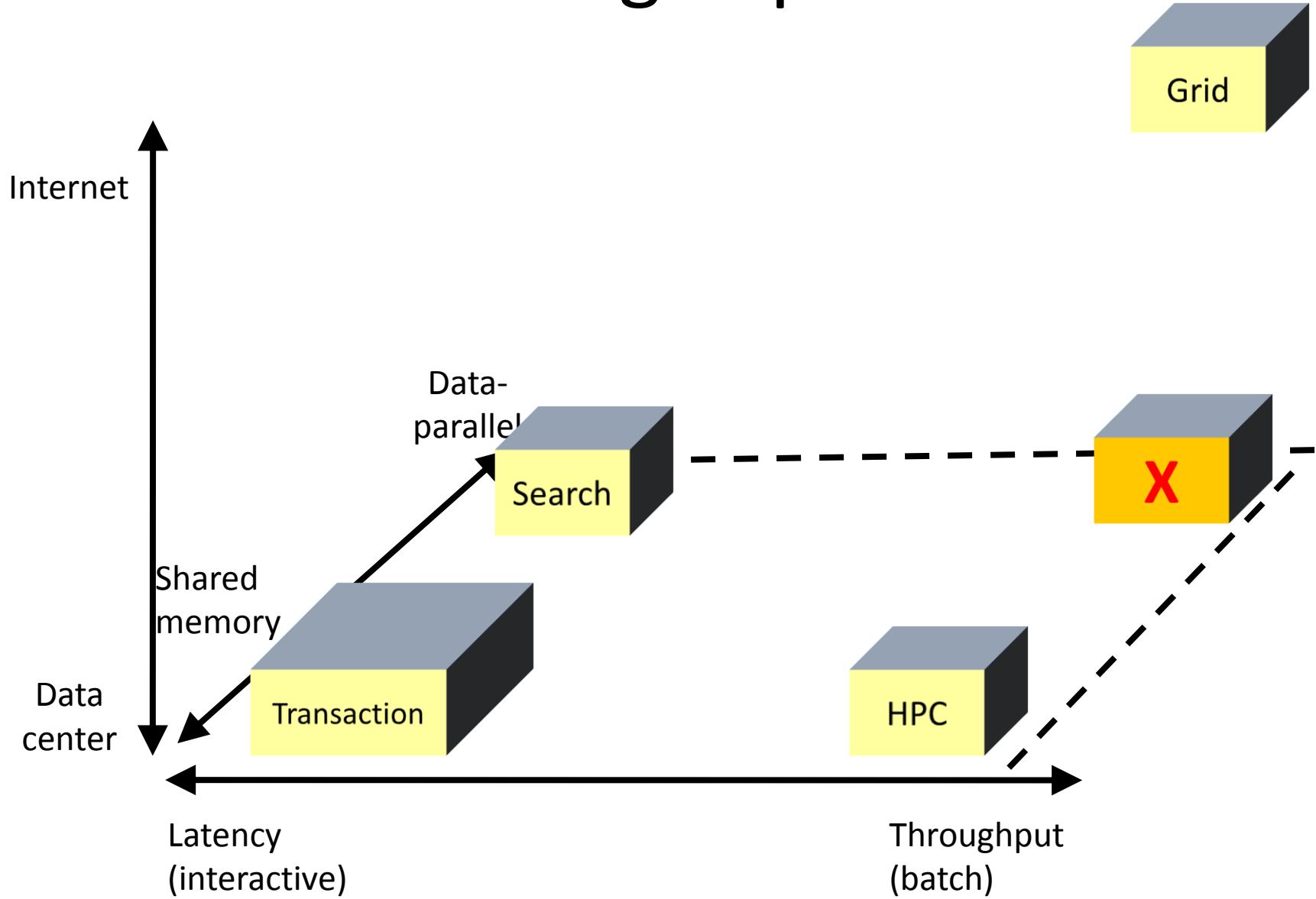
12. Compute two iteration steps of the pagerank of a directed graph (V, E) . Each node $n \in V$ has a weight $W(n)$, initially $1/|V|$, and an out-degree $D(n)$ (the number of out-edges of n). Each algorithm iteration changes weights of all nodes n as follows:

$$W'(n) = \sum_{(m,n) \in E} W(m)/D(m)$$

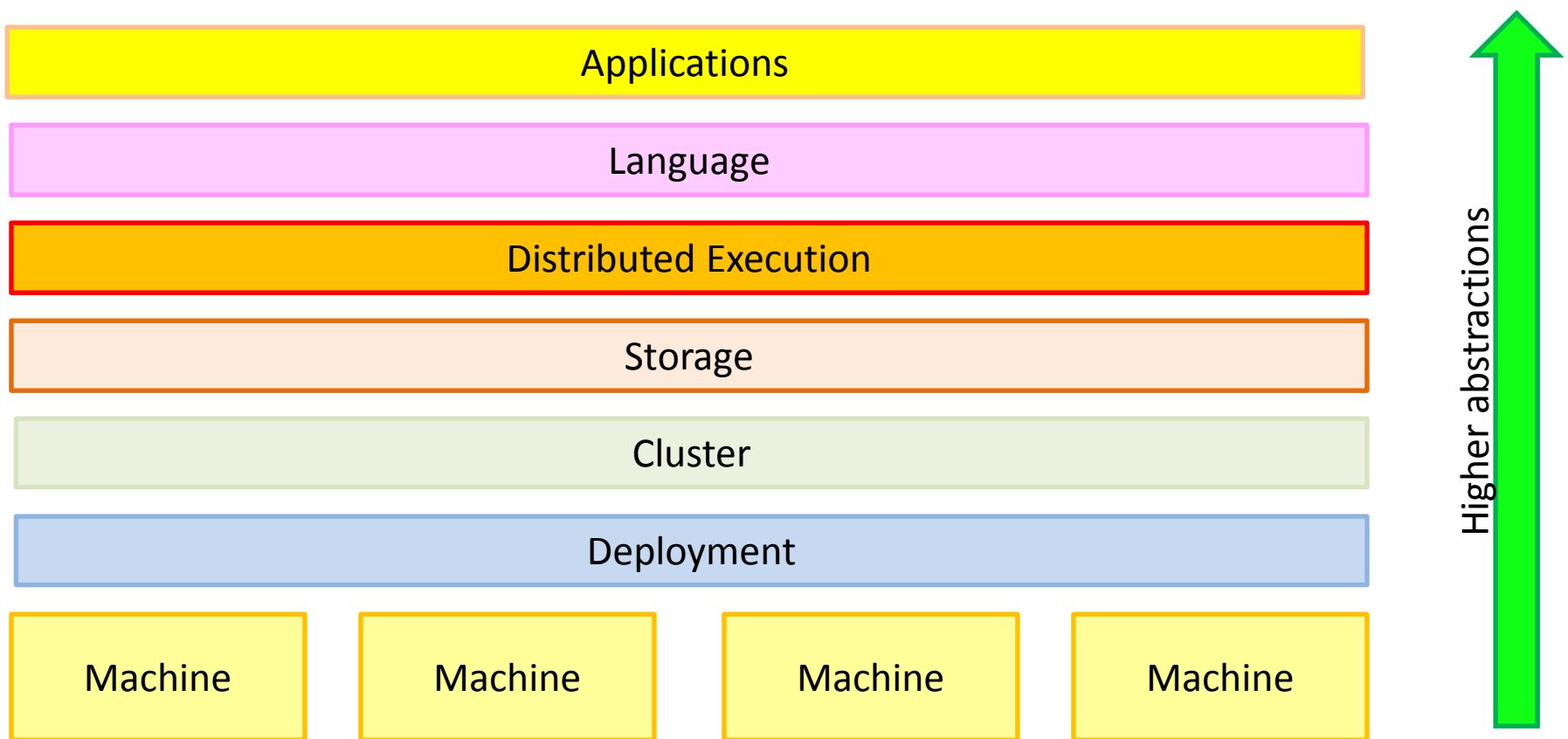


Crunching Big Data

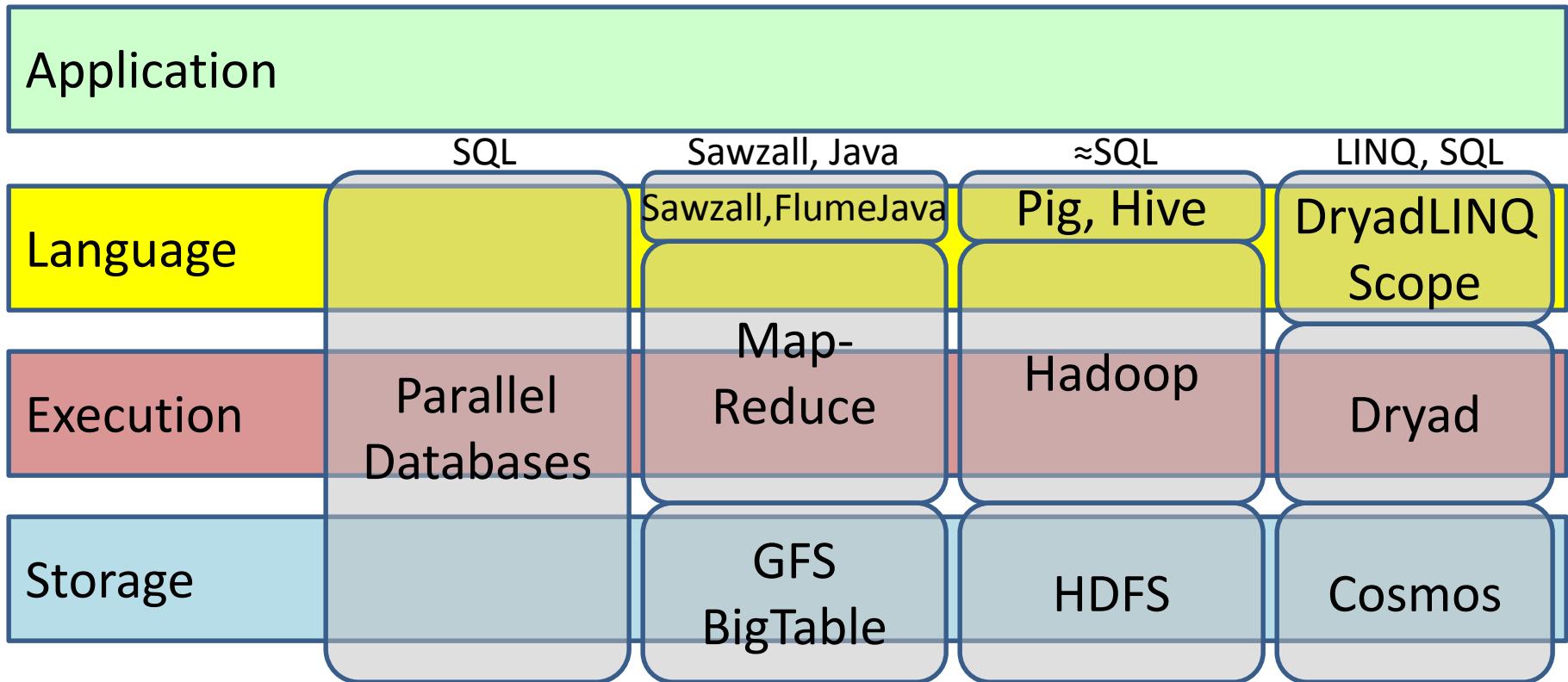
Design Space

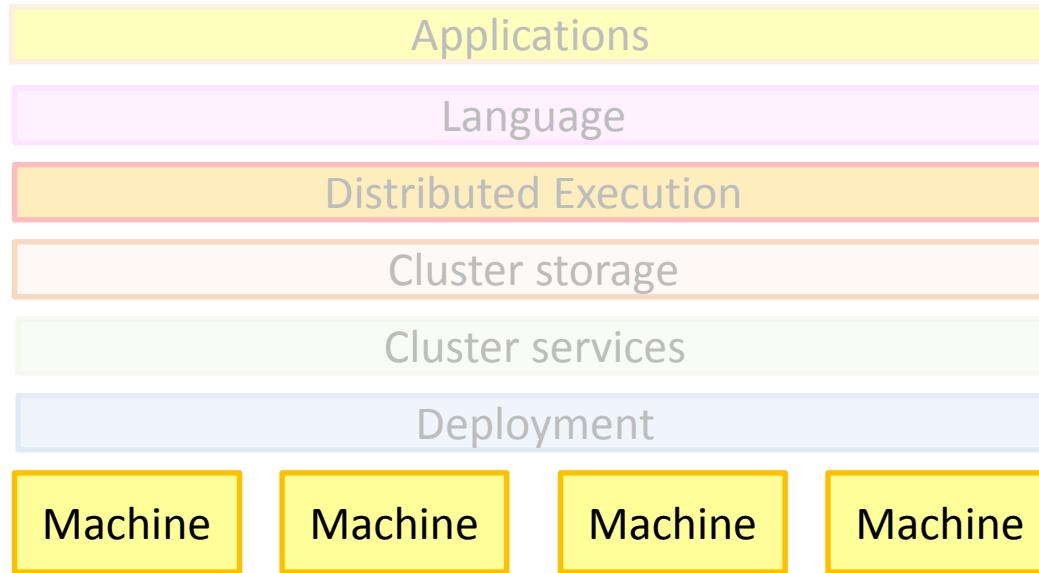


Software Stack



Data-Parallel Computation

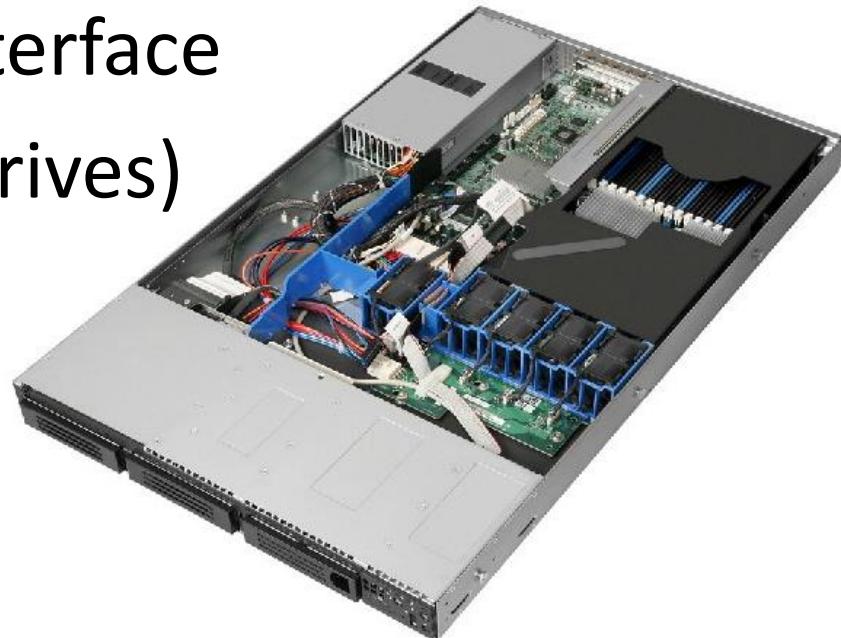




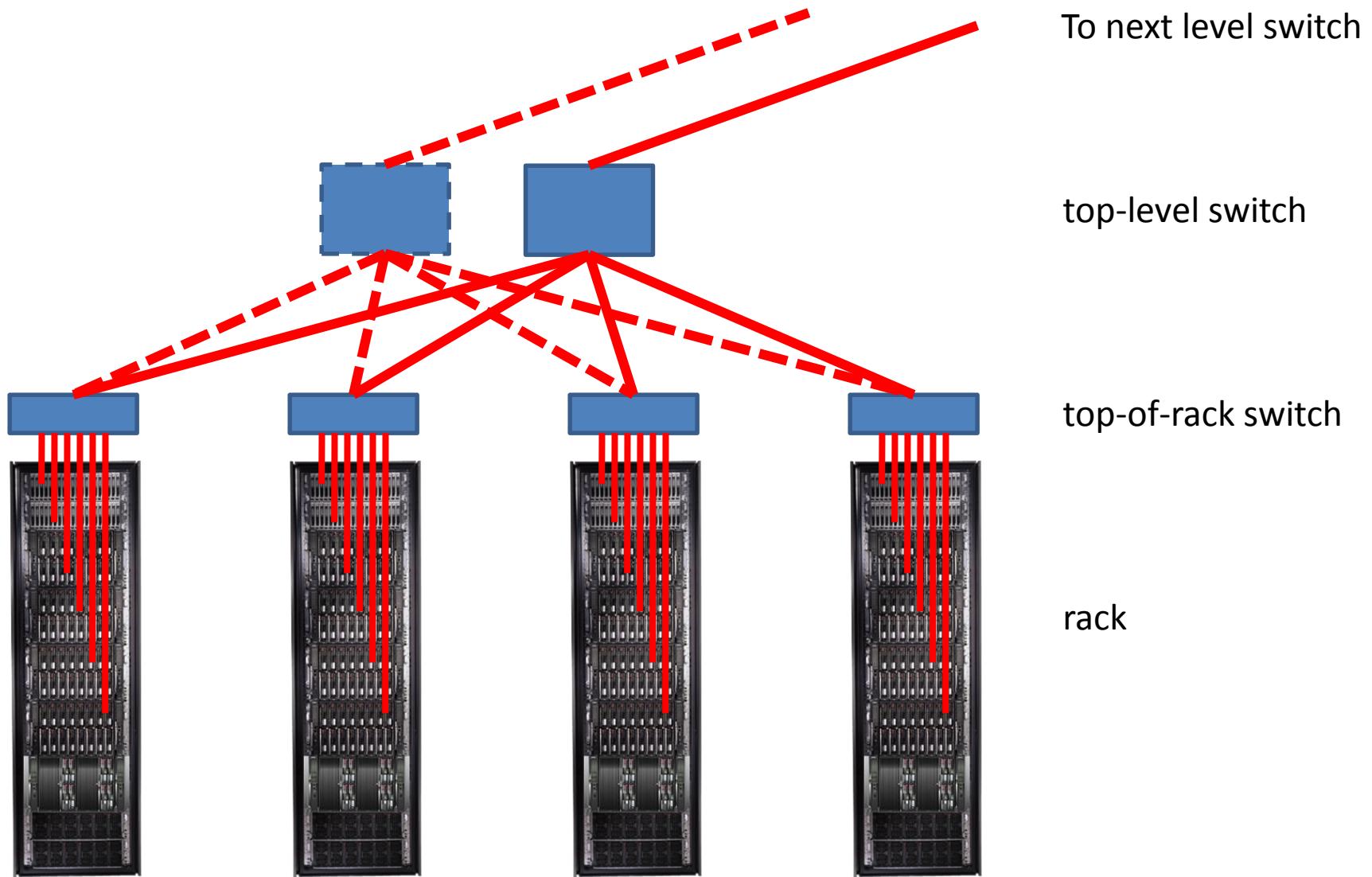
CLUSTER ARCHITECTURE

Cluster Machines

- Commodity server-class systems
- Optimized for **cost**
- Remote management interface
- Local storage (multiple drives)
- Multi-core CPU
- Gigabit+ Ethernet
- Stock OS

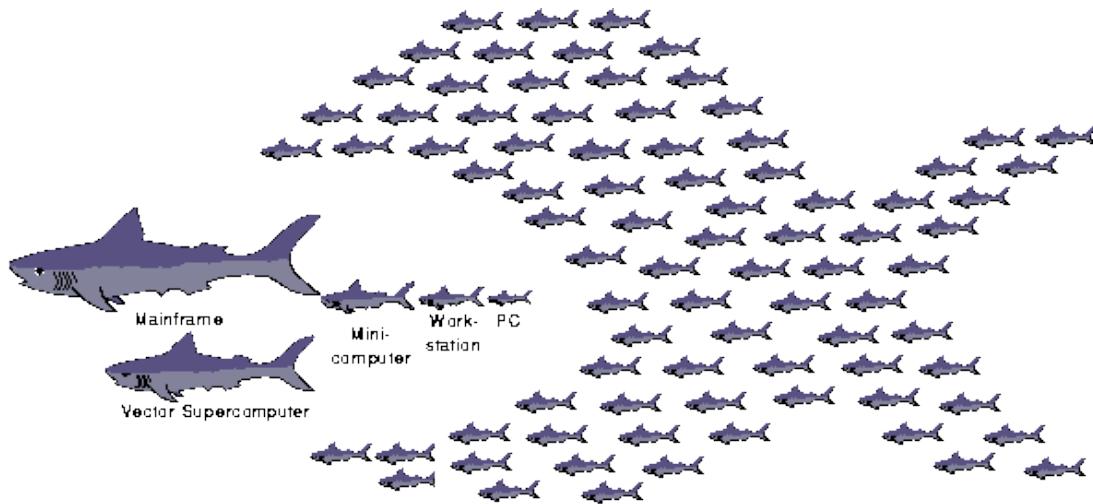


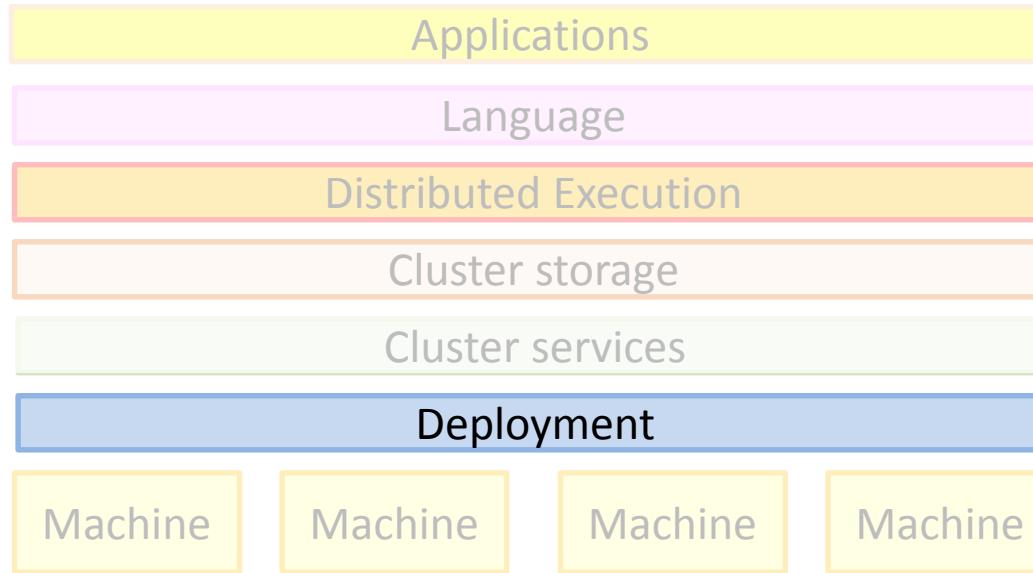
Cluster network topology



The secret of scalability

- Cheap hardware
- Smart software
- Berkeley Network of Workstations ('94-'98)
<http://now.cs.berkeley.edu>





DEPLOYMENT: AUTOPILOT

[Autopilot: Automatic Data Center Management](#), Michael Isard, in
Operating Systems Review, vol. 41, no. 2, pp. 60-67, April 2007

Autopilot goal



- Handle automatically routine tasks
- Without operator intervention



Autopiloted System

Autopilot control



Application

Autopilot services



Recovery-Oriented Computing

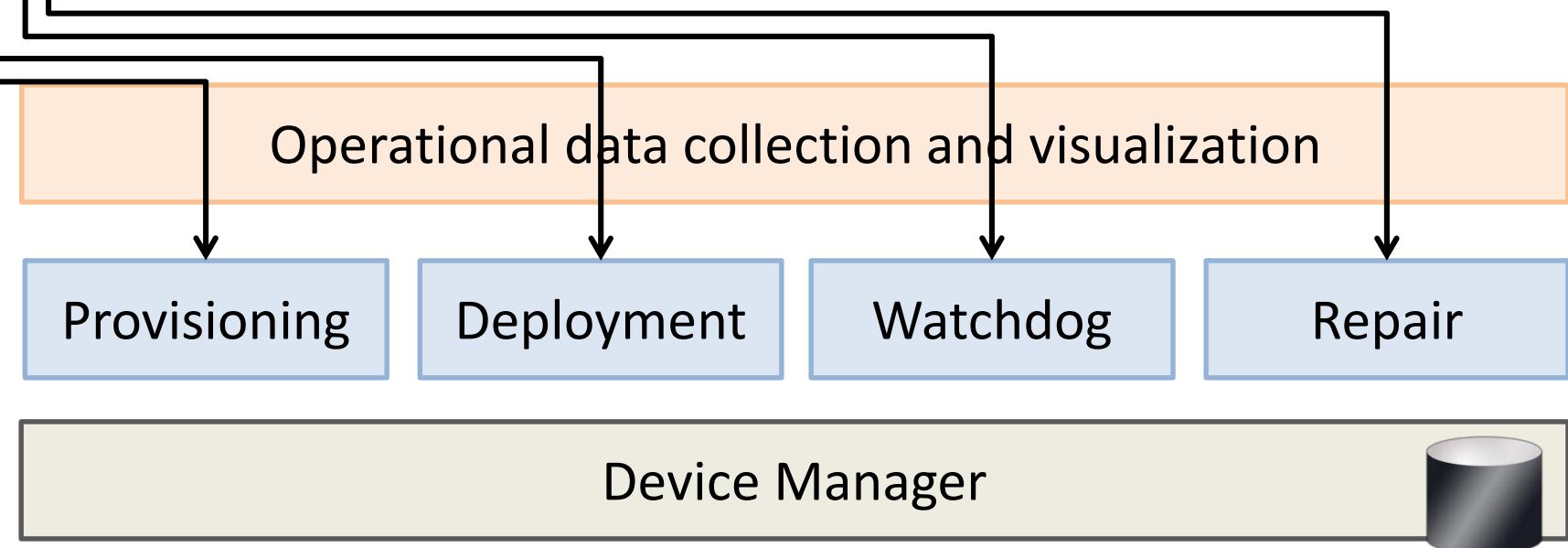
- Everything will eventually fail
- Design for failure
- Crash-only software design
- <http://roc.cs.berkeley.edu>



Brown, A. and D. A. Patterson. Embracing Failure: A Case for Recovery-Oriented Computing (ROC). *High Performance Transaction Processing Symposium*, October 2001.

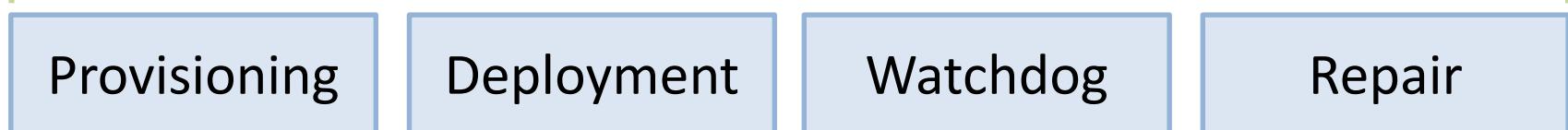
Autopilot Architecture

- Discover new machines; netboot; self-test
- Install application binaries and configurations
- Monitor application health
- Fix broken machines



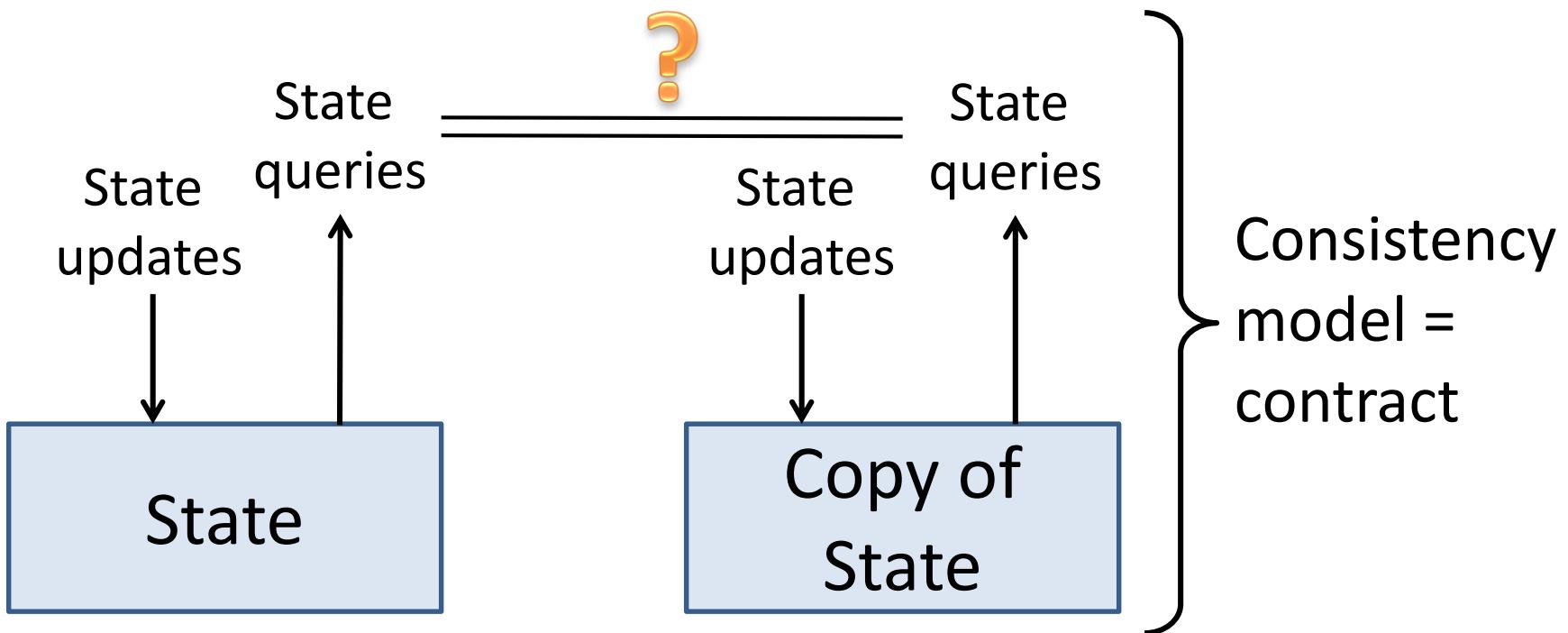
Distributed State

*Distributed state
Replicated
Weakly consistent*



*Centralized state
Replicated
Strongly consistent*

Problems of Distributed State

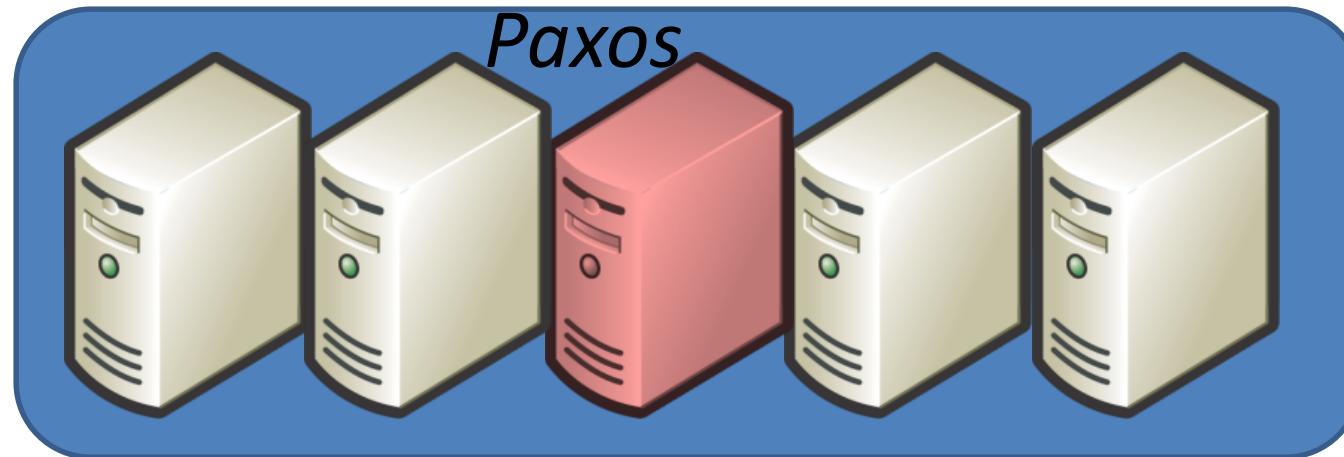


Centralized Replicated Control

- Keep essential control state centralized
- Replicate the state for reliability
- Use the Paxos consensus protocol
(Zookeeper is the open-source alternative)



Leslie Lamport. The part-time parliament. ACM Transactions on Computer Systems, 16(2):133-169, May 1998.



Consistency Models



Strong consistency

- Expensive to provide
- Hard to build right
- Easy to understand
- Easy to program against
- Simple application design



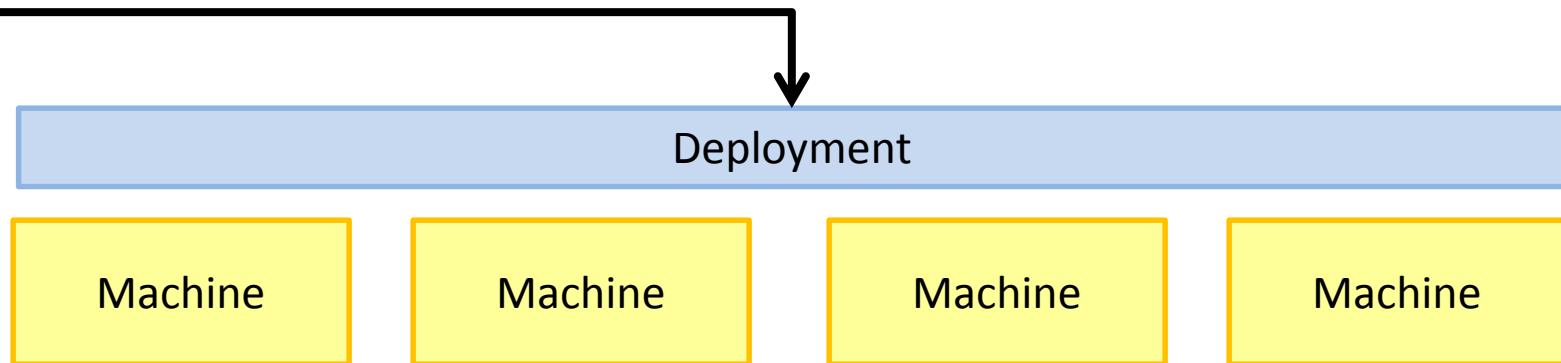
Weak consistency

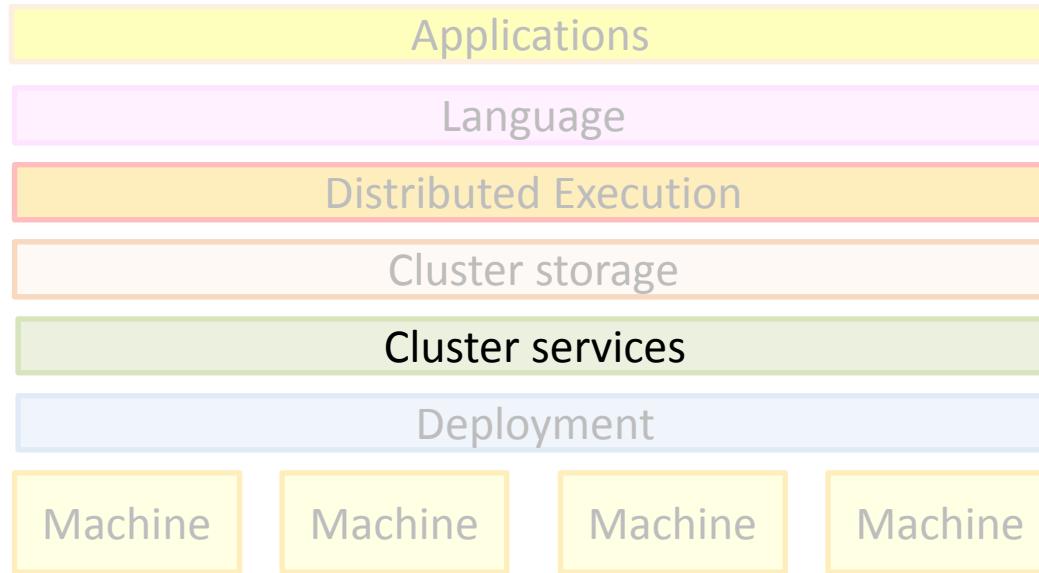


- Increases availability
- Many different models
- Easy to misuse
- Very hard to understand
- Conflict management in application

Autopilot abstraction

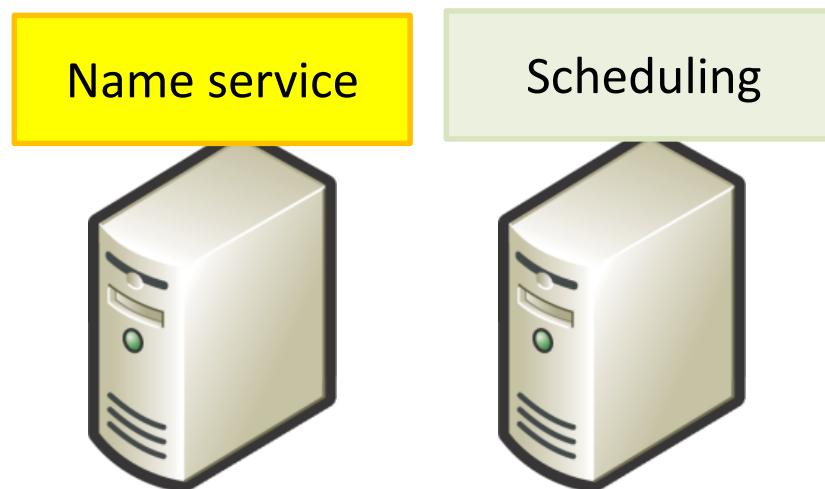
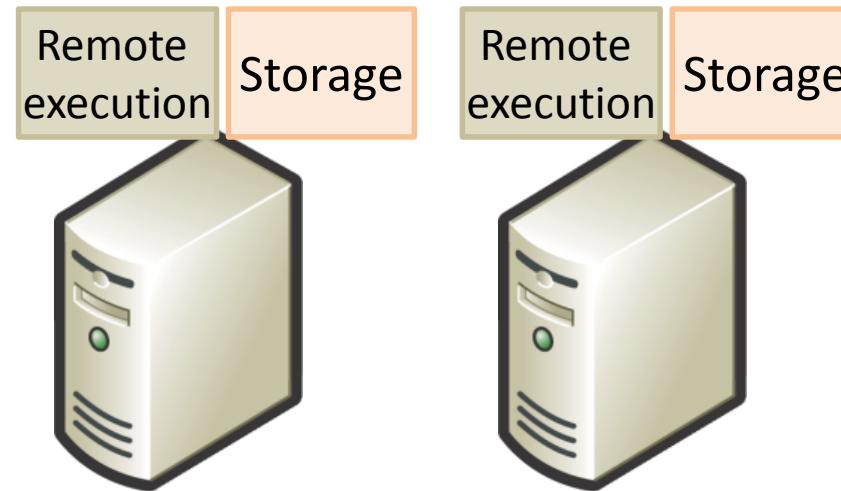
Self-healing machines





CLUSTER SERVICES

Cluster Machines



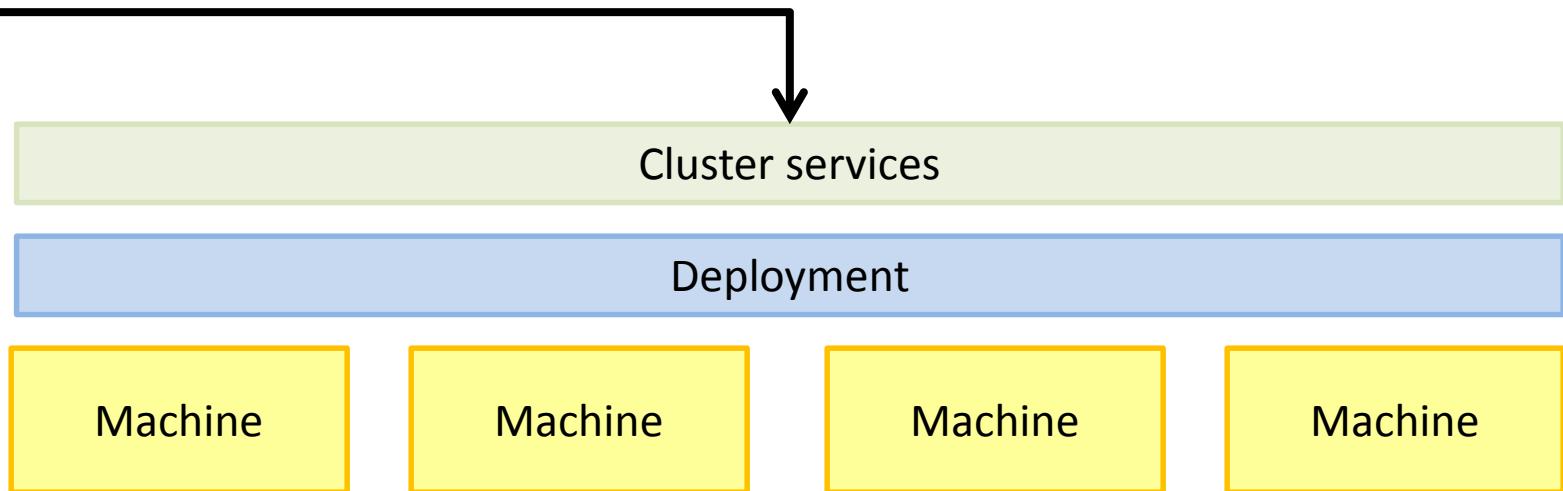


Cluster Services

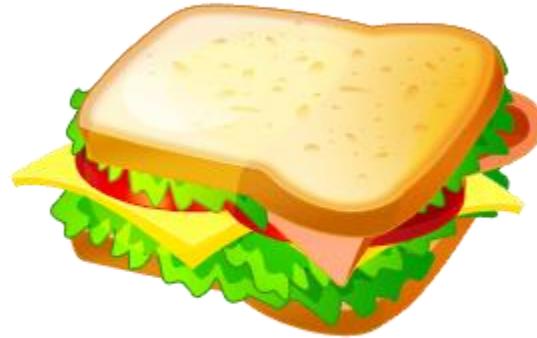
- Name service: discover cluster machines
- Scheduling: allocate cluster machines
- Storage: file contents
- Remote execution: spawn new computations

Cluster services abstraction

Reliable specialized machines

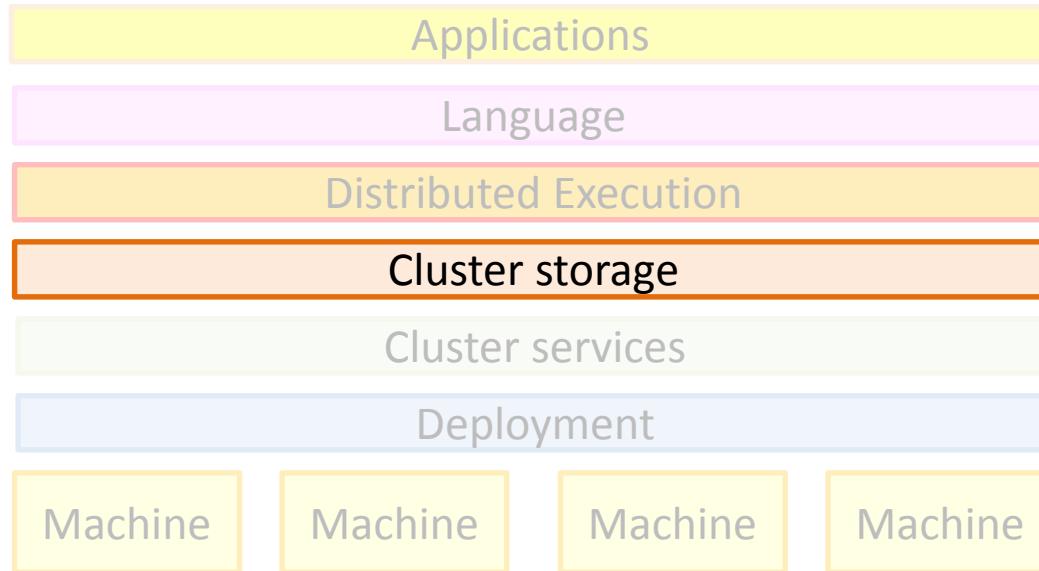


Layered Software Architecture



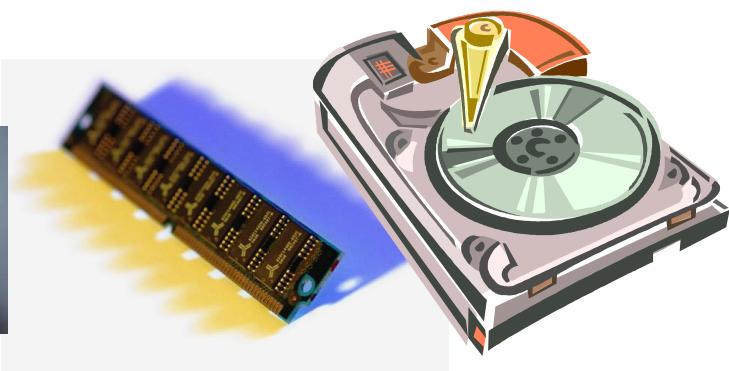
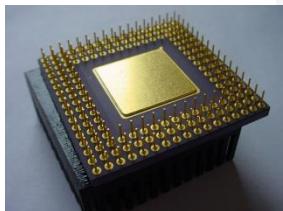
- Simple components
- Software-provided reliability
- Versioned APIs
- Design for live staged deployment





DISTRIBUTED STORAGE

Bandwidth hierarchy



Cache

RAM

Local
disks

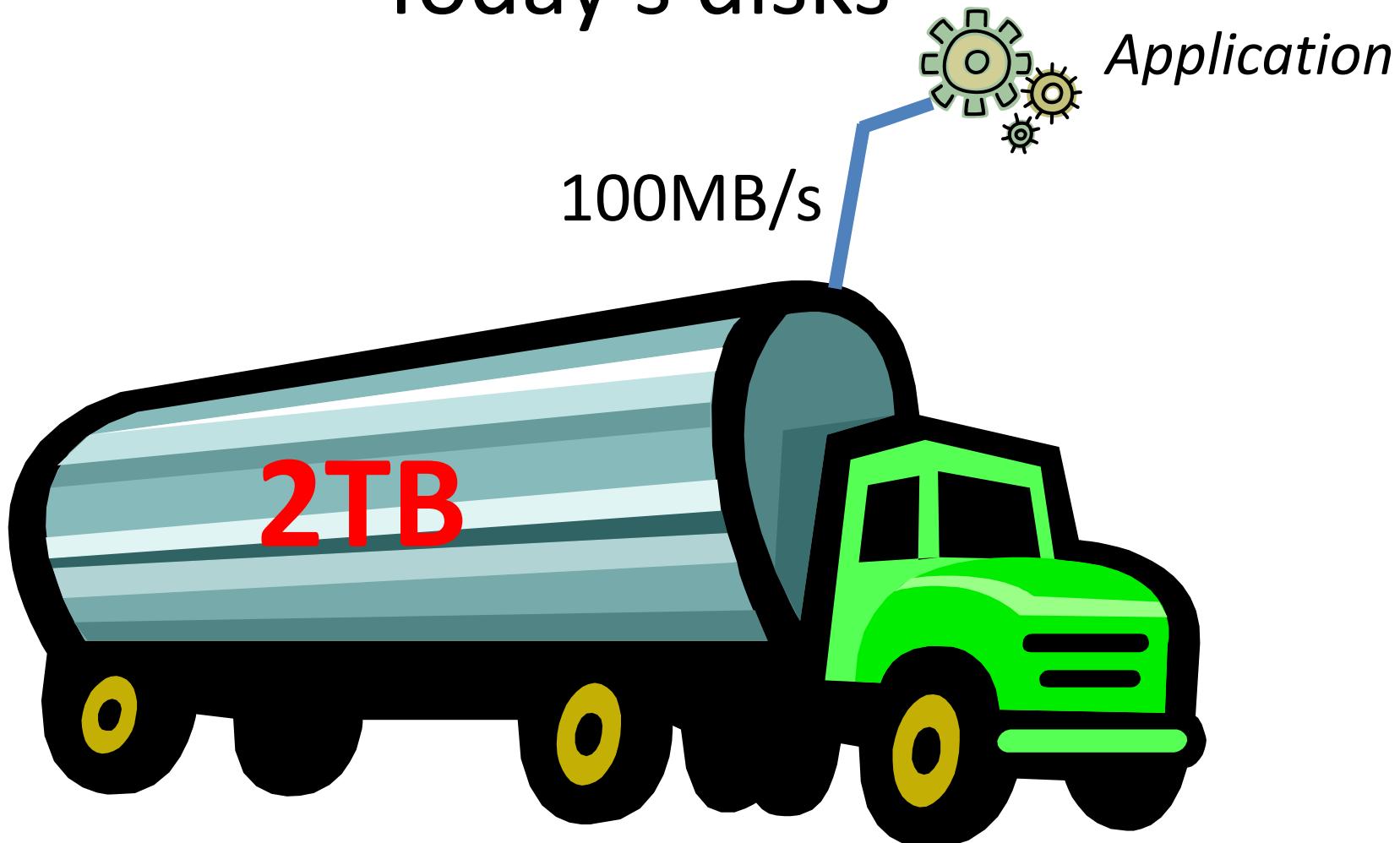
Local
rack

Remote
rack

Remote
datacenter

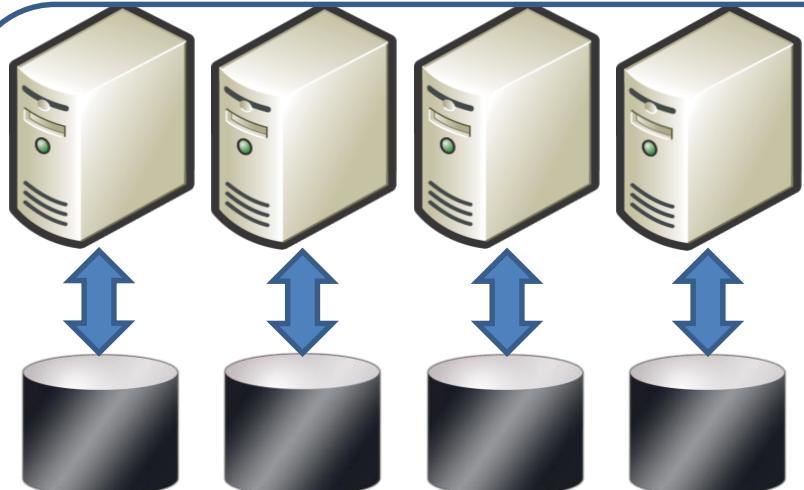


Today's disks



Storage bandwidth

- Expensive
- Fast network needed
- Limited by network b/w



- Cheap network
- Cheap machines
- Limited by disk b/w

Time to read 1TB (sequential)



- $1 \text{ TB} / 100\text{MB/s} = 3 \text{ hours}$

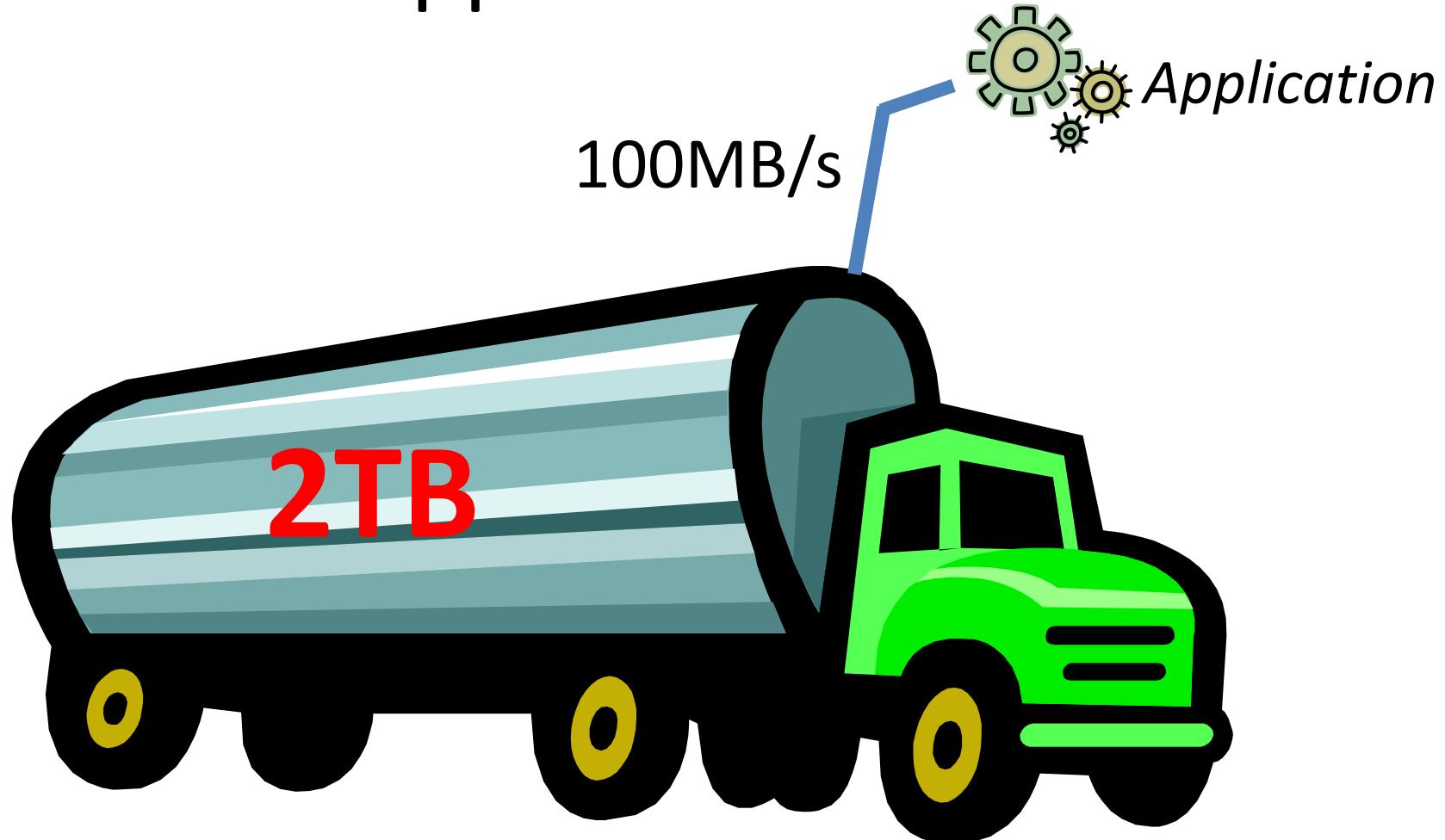


- $1 \text{ TB} / 10 \text{ Gbps} = 40 \text{ minutes}$

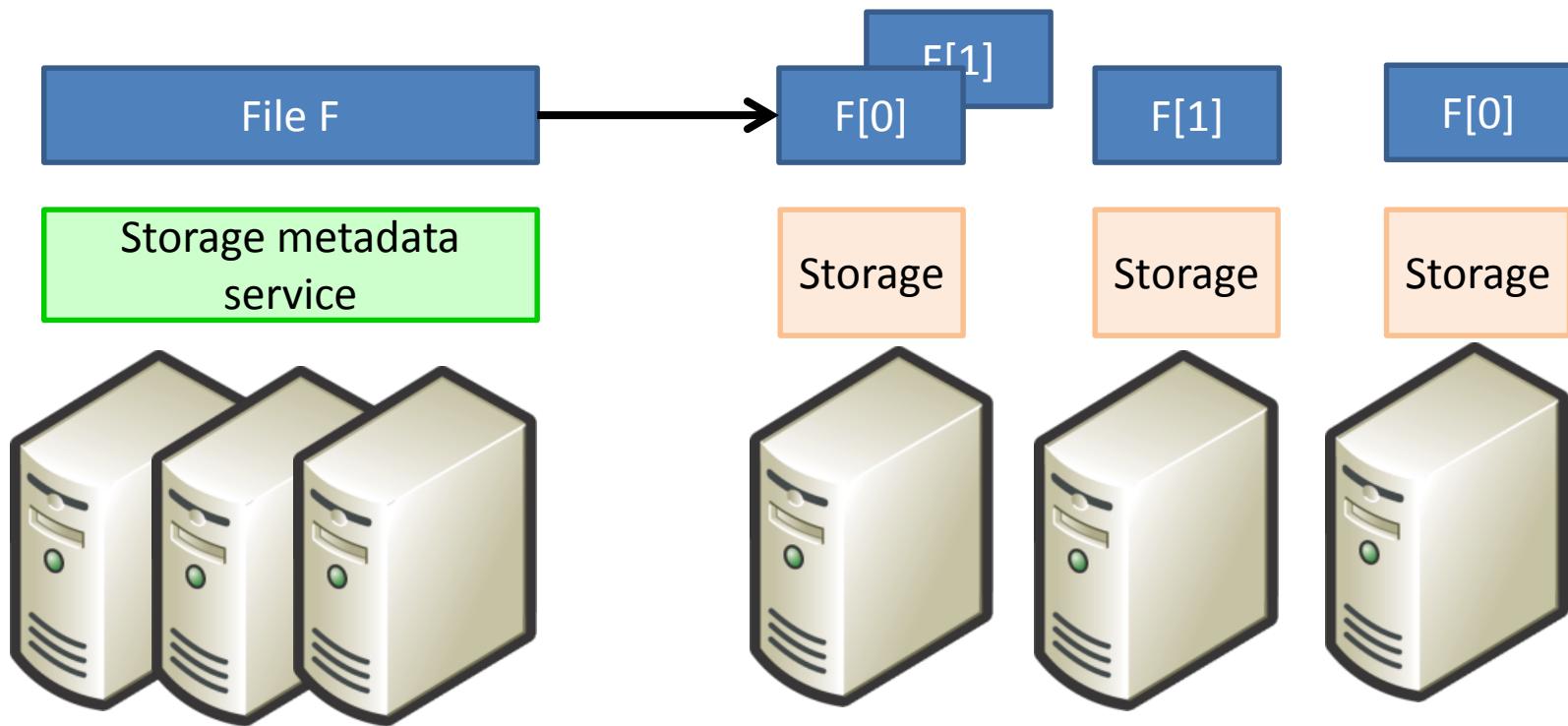


- $1 \text{ TB} / (100 \text{ MB/s/disk} \times 10000 \text{ disks}) = 1\text{s}$
- $(1000 \text{ machines} \times 10 \text{ disks} \times 1\text{TB/disk} = 10\text{PB})$

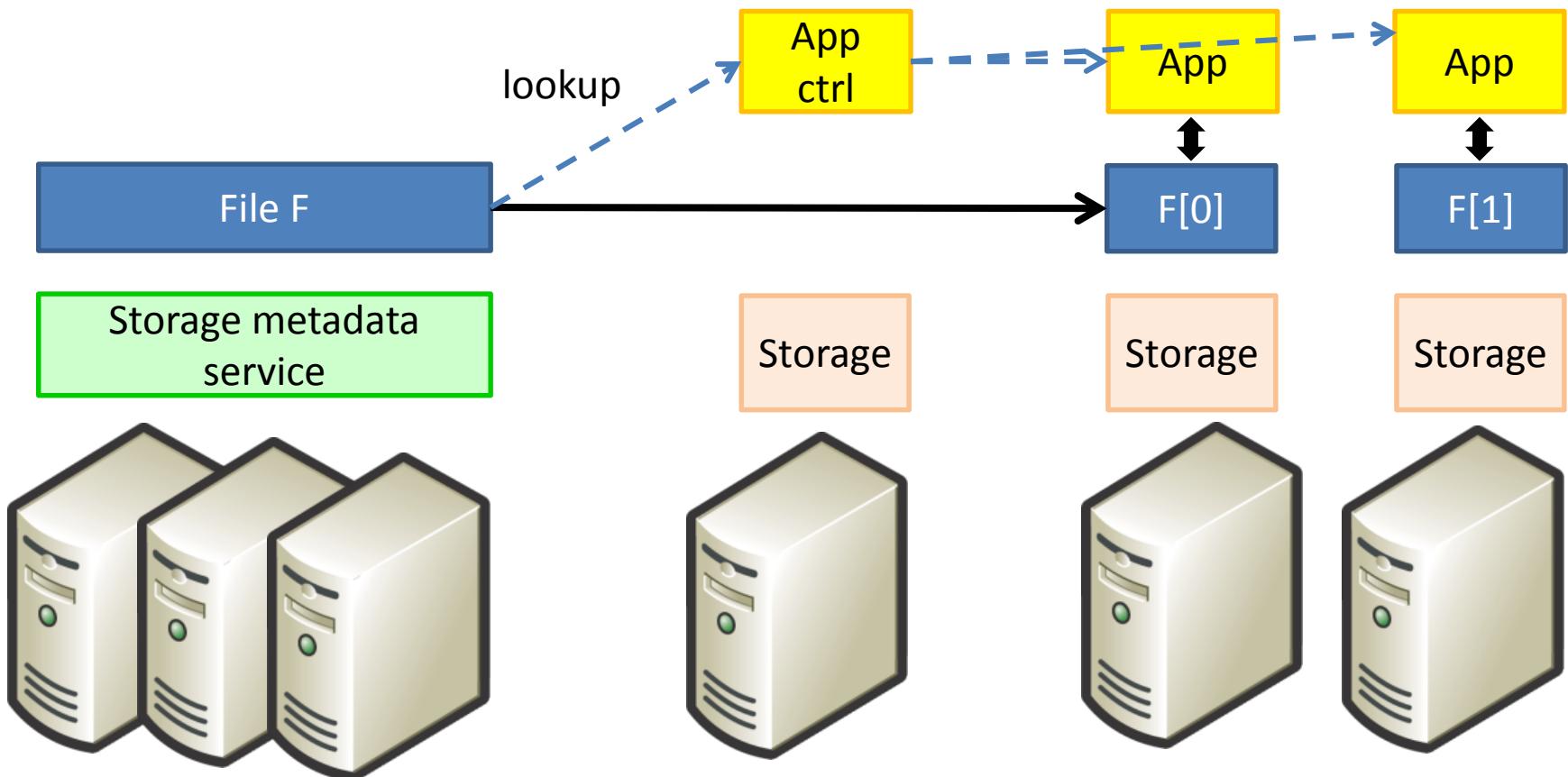
Send the application to the data!



Large-scale Distributed Storage

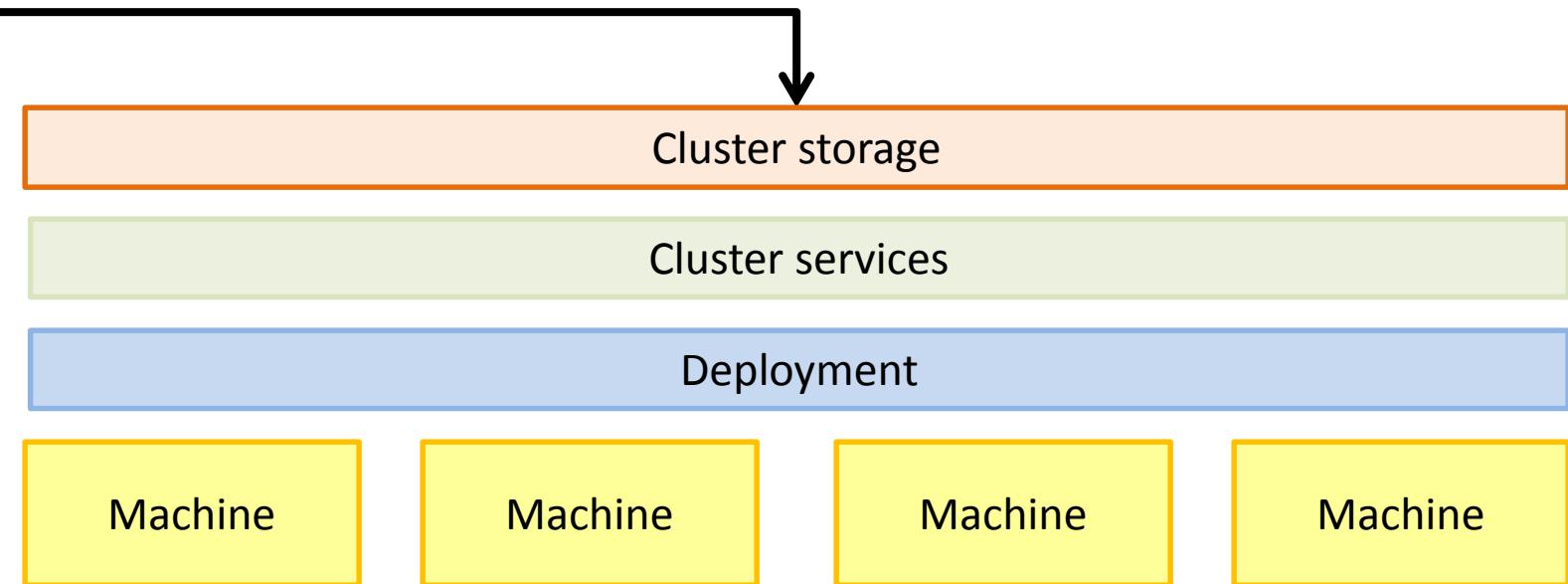


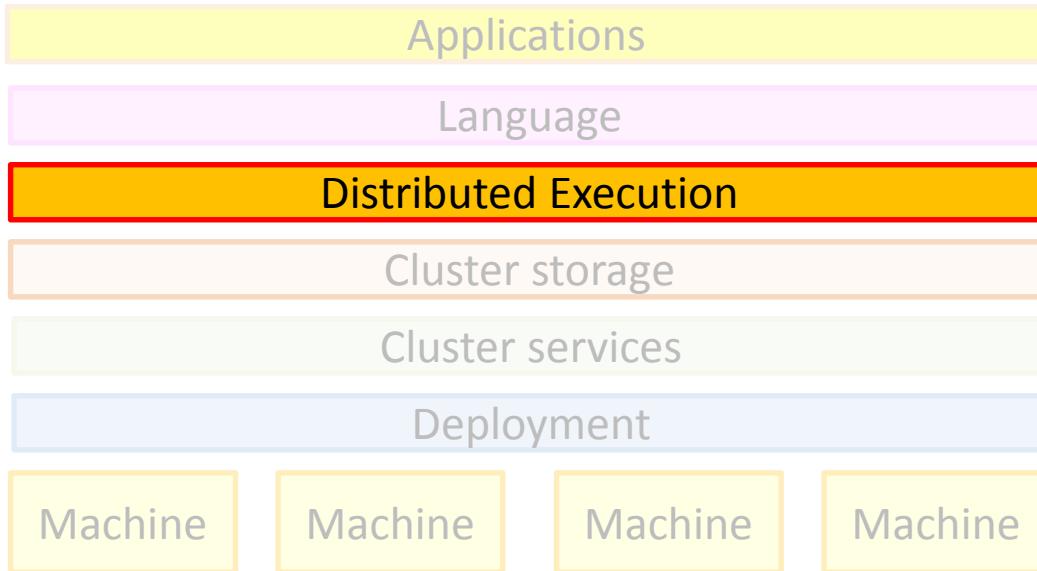
Parallel Application I/O



Cluster Storage Abstraction

Set of reliable machines with a global filesystem





DISTRIBUTED EXECUTION: DRYAD



Dryad painting by
Evelyn de Morgan

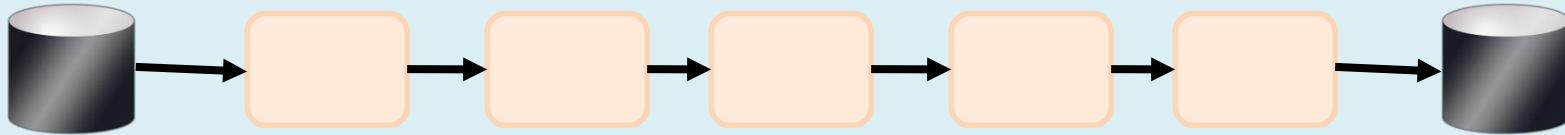
Dryad = Execution Layer



2-D Piping

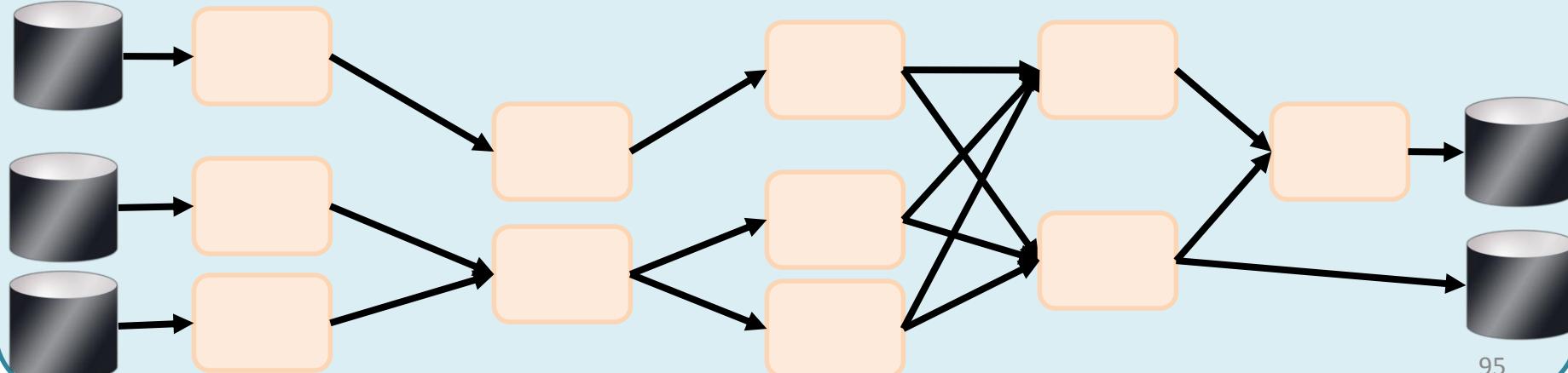
- Unix Pipes: 1-D

grep | sed | sort | awk | perl

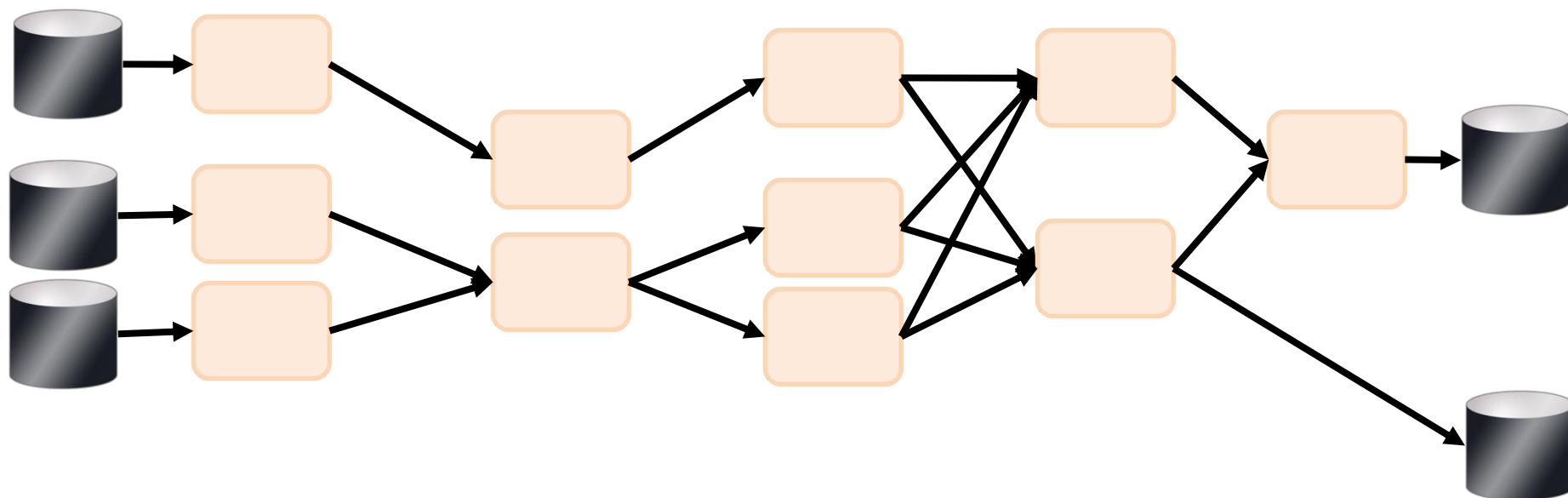


- Dryad: 2-D

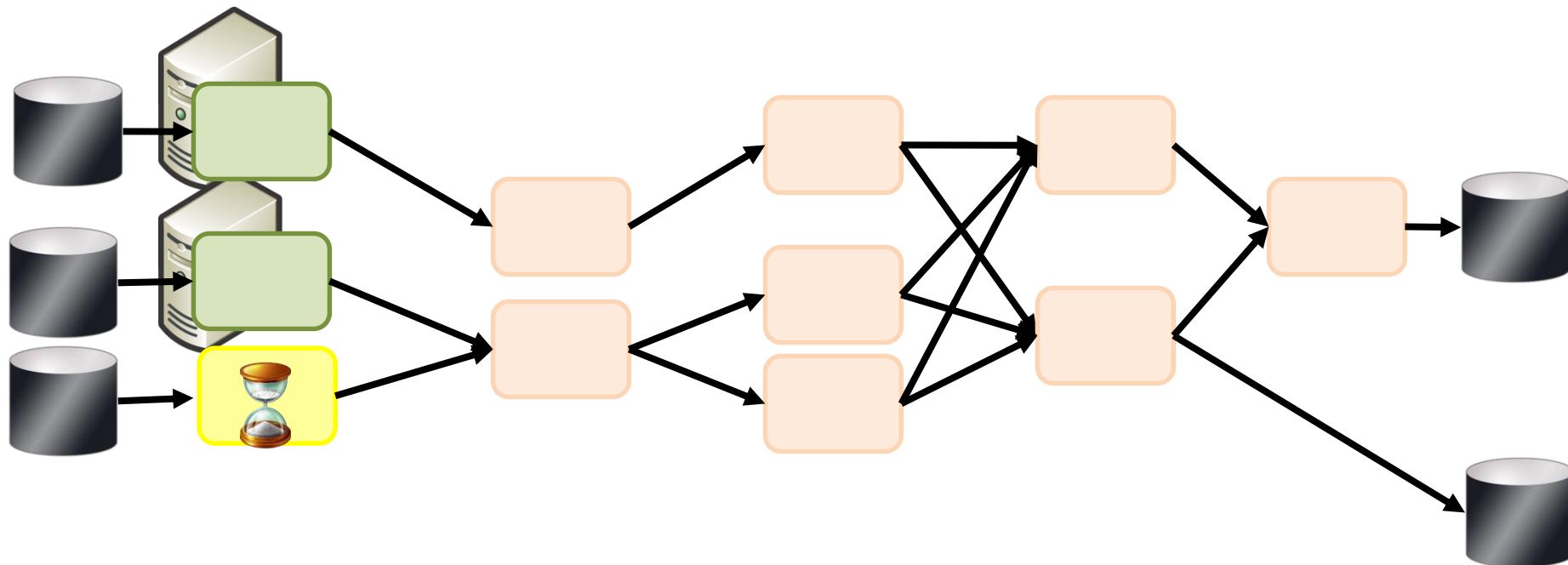
grep¹⁰⁰⁰ | sed⁵⁰⁰ | sort¹⁰⁰⁰ | awk⁵⁰⁰ | perl⁵⁰



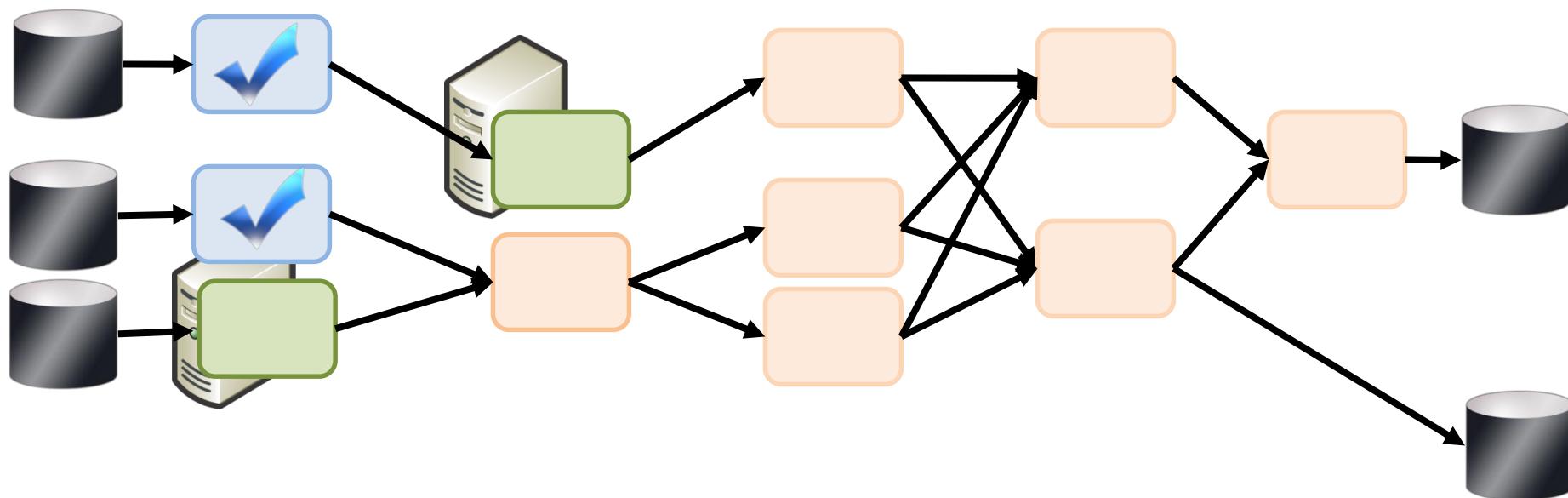
Virtualized 2-D Pipelines



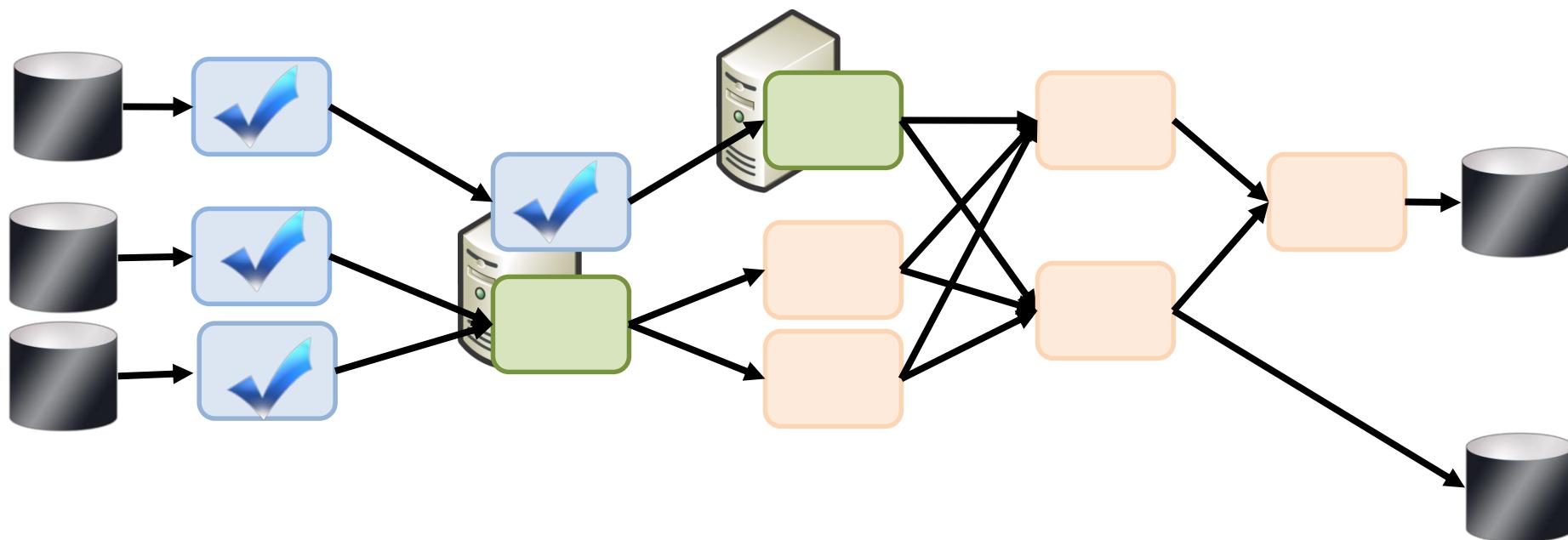
Virtualized 2-D Pipelines



Virtualized 2-D Pipelines

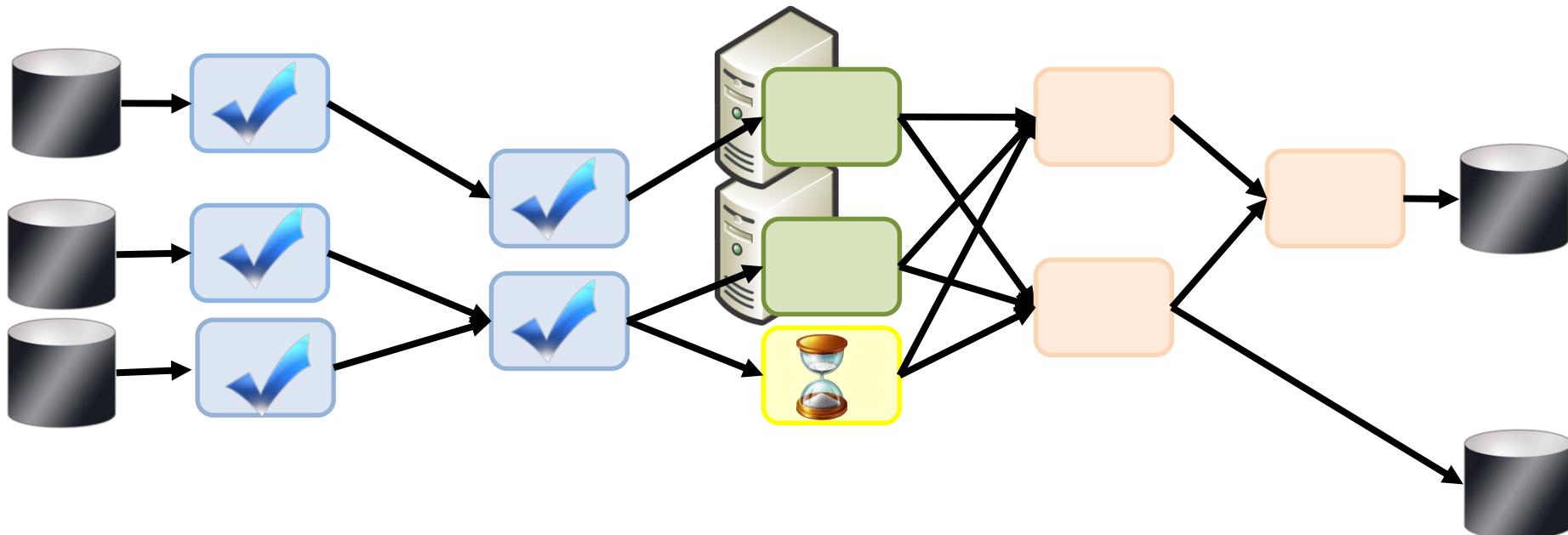


Virtualized 2-D Pipelines

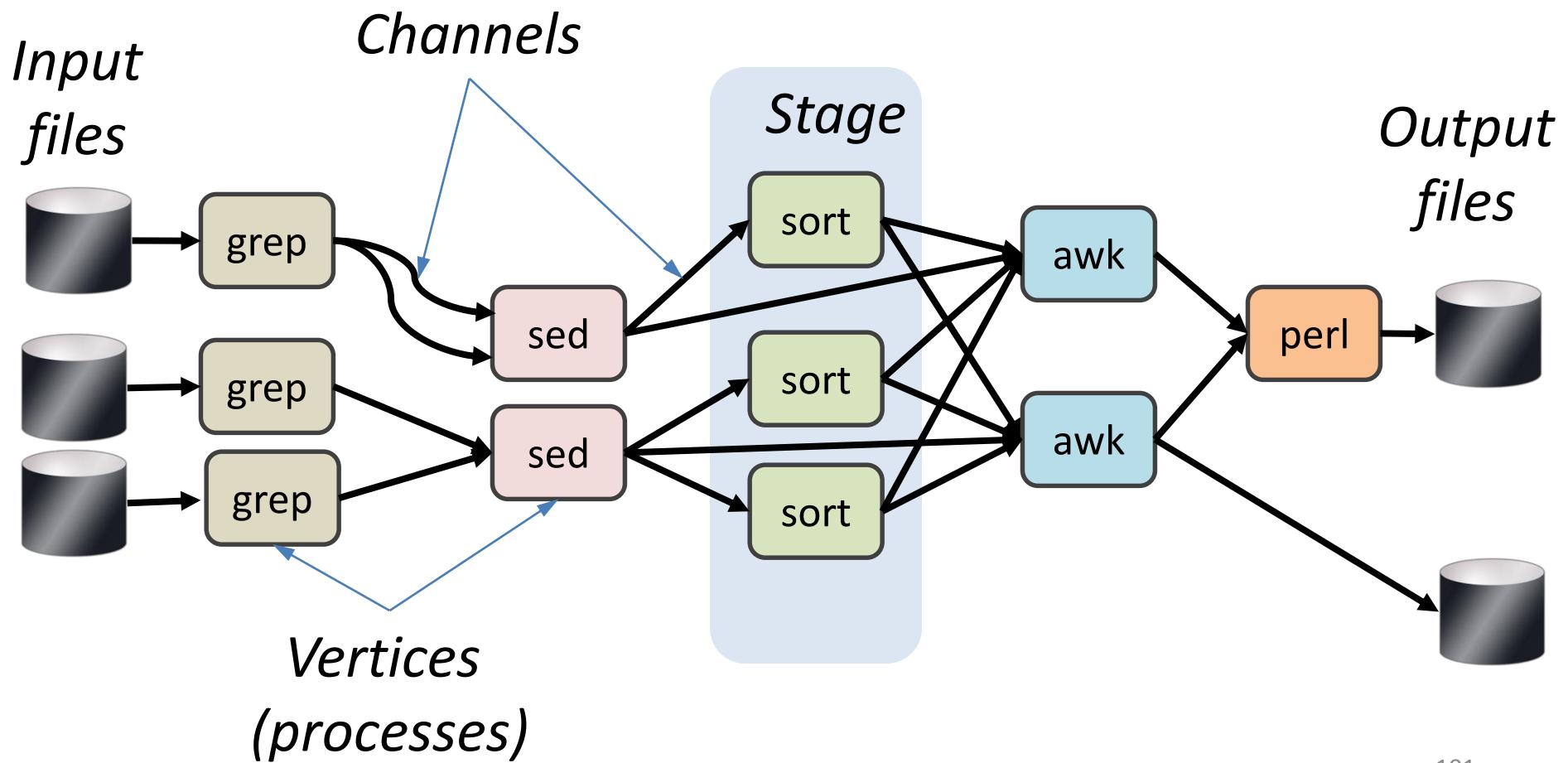


Virtualized 2-D Pipelines

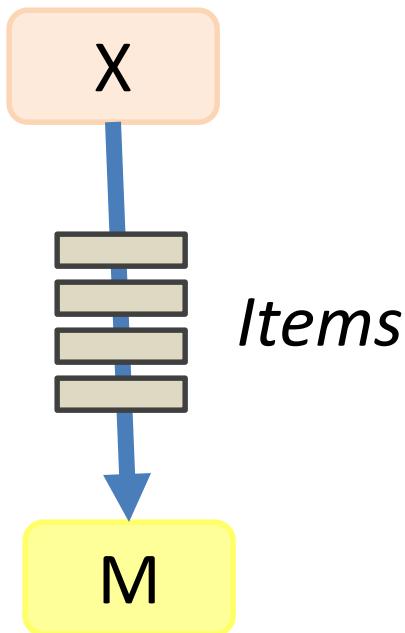
- 2D DAG
- multi-machine
- virtualized



Dryad Job Structure



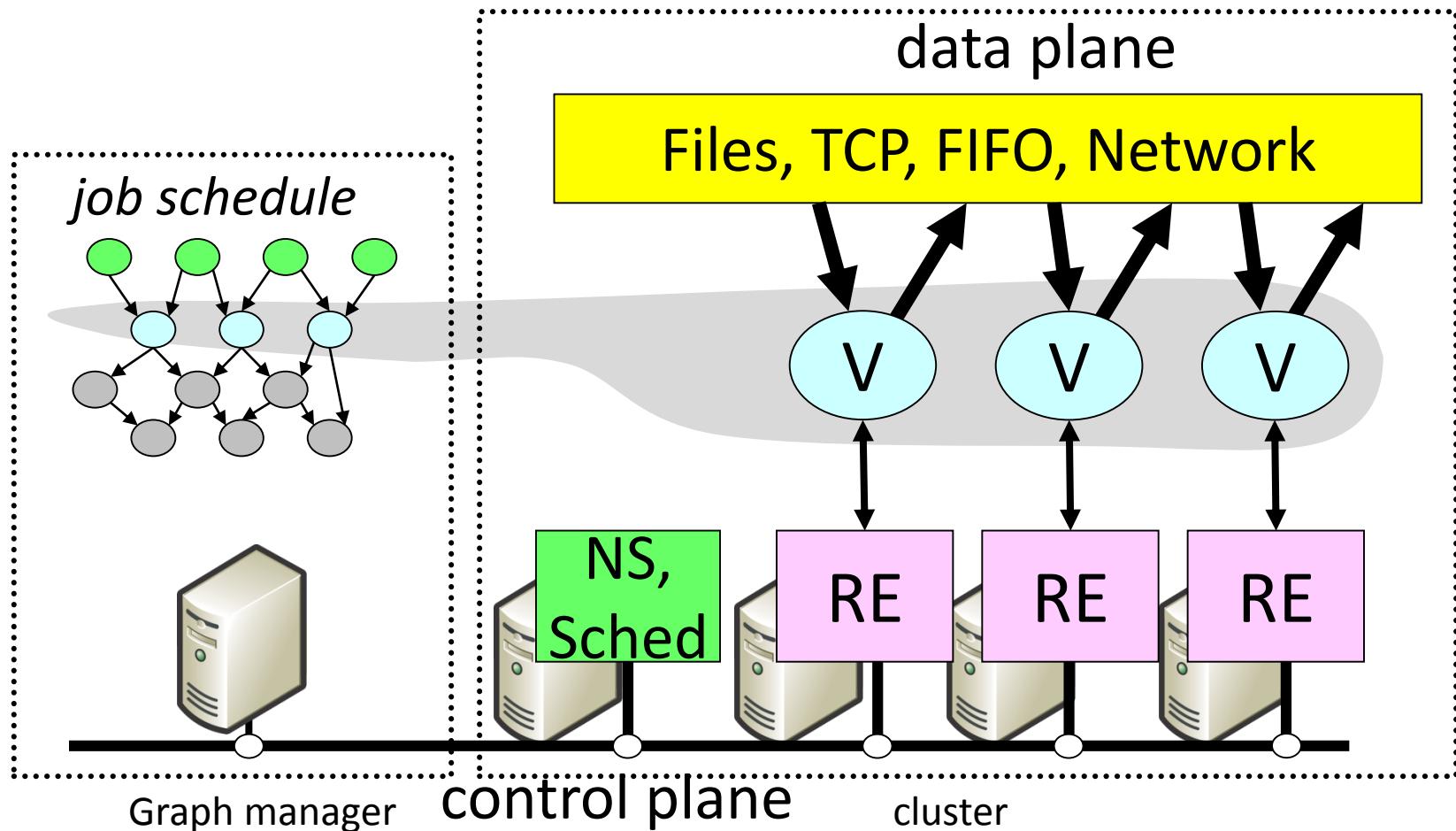
Channels



Finite streams of items

- files
- TCP pipes
- memory FIFOs

Dryad System Architecture



Separate Data and Control Plane

- Different kinds of traffic
 - Data = bulk, pipelined
 - Control = interactive
- Different reliability needs



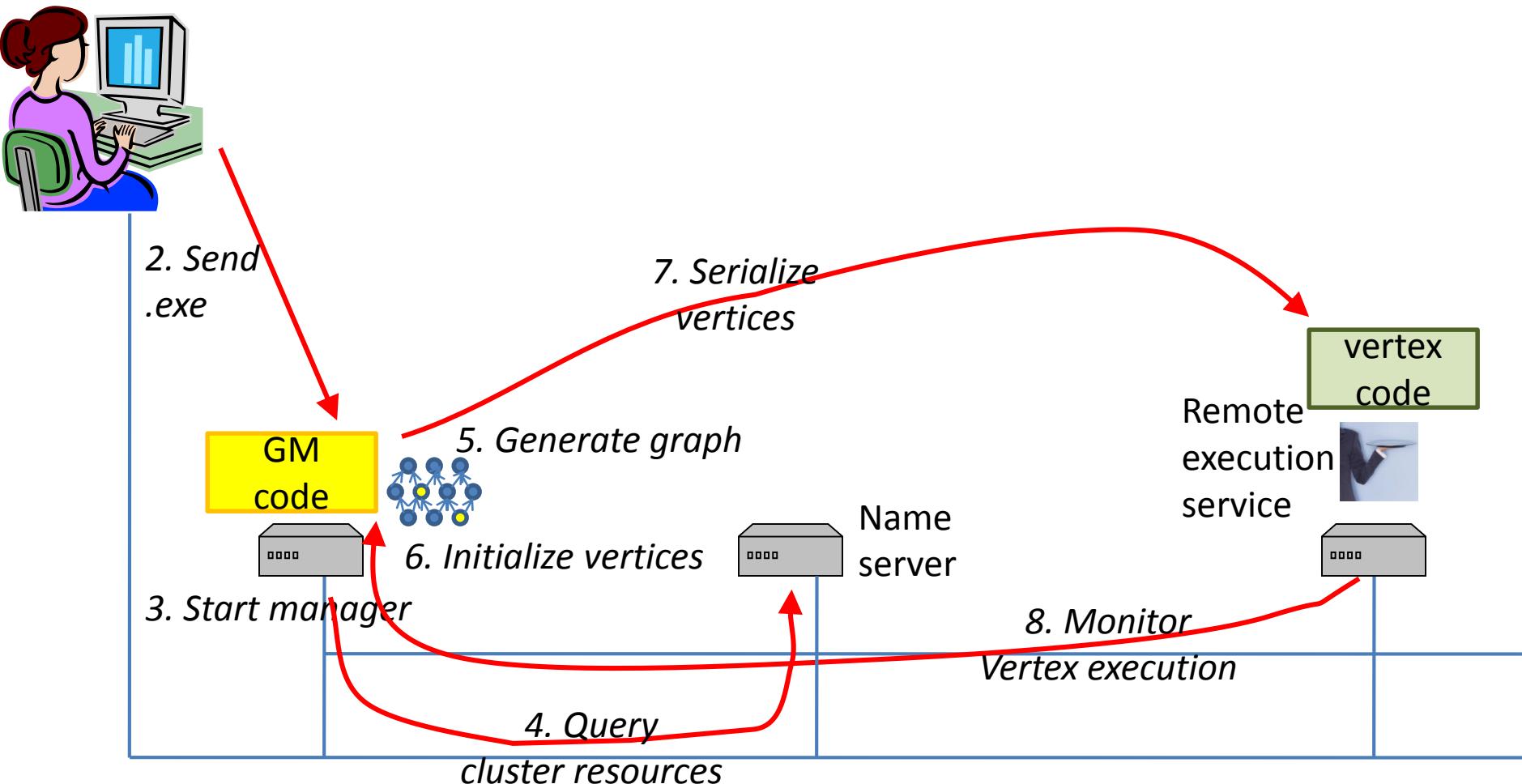
Centralized control

- Manager state is not replicated
- Entire manager state is held in RAM
- Simple implementation
- Vertices use leases:
no runaway computations on manager crash
- Manager crash causes complete job crash



Staging

1. Build

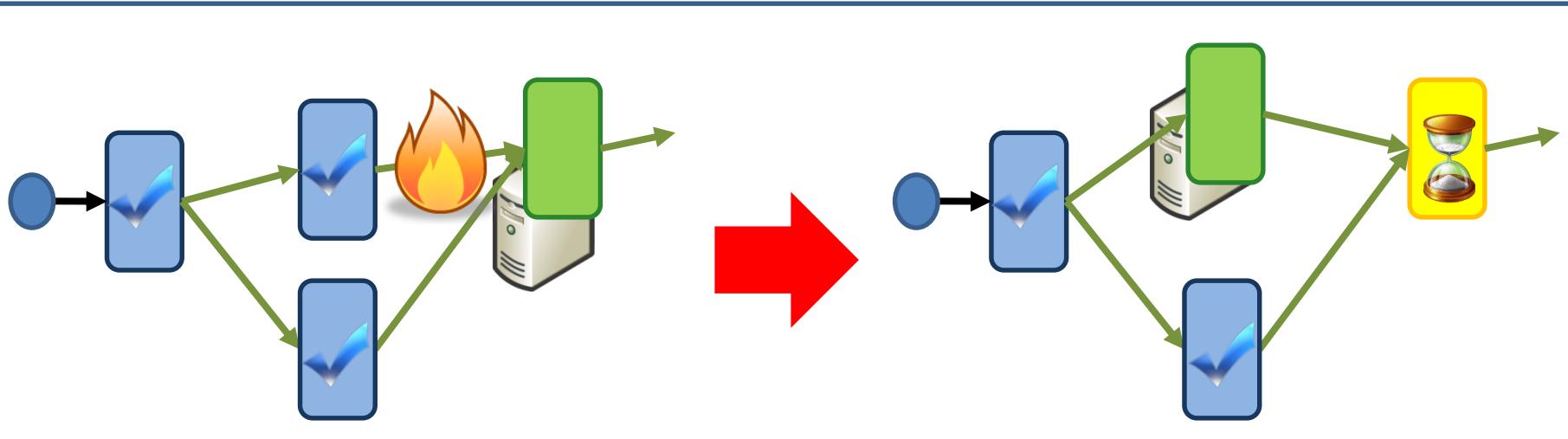
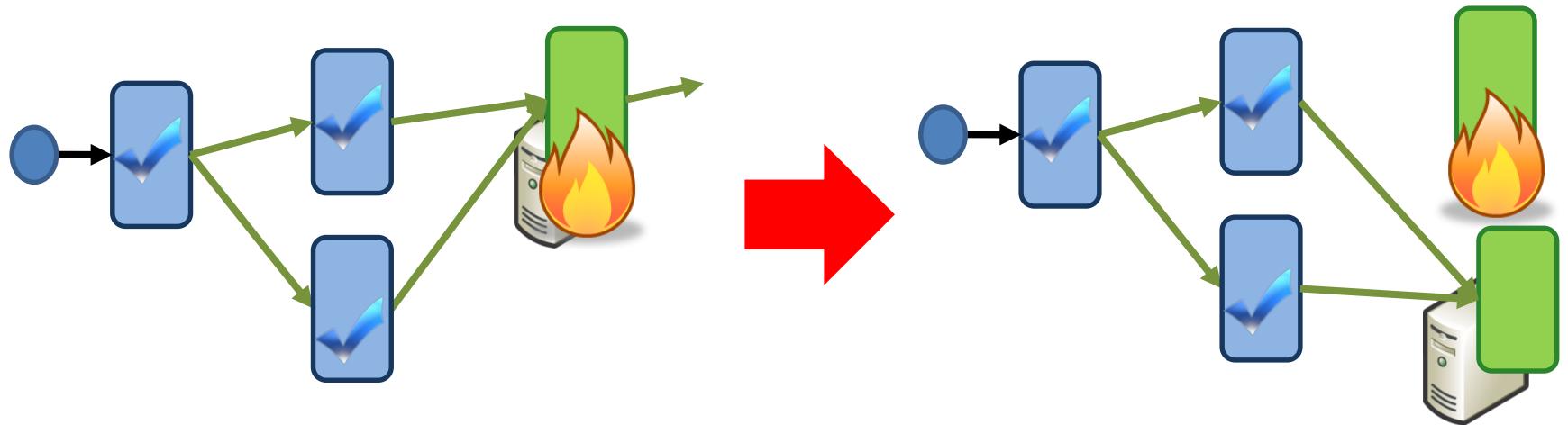


Scaling Factors

- Understand how fast things scale
 - # machines << # vertices << # channels
 - # control bytes << # data bytes
- Understand the algorithm cost
 - $O(\# \text{ machines}^2)$ acceptable, but $O(\# \text{ edges}^2)$ not
- Every order-of-magnitude increase will reveal new bugs



Fault Tolerance



Danger of Fault Tolerance

- Fault tolerance can mask defects in other software layers
- Log fault repairs
- Review the logs periodically



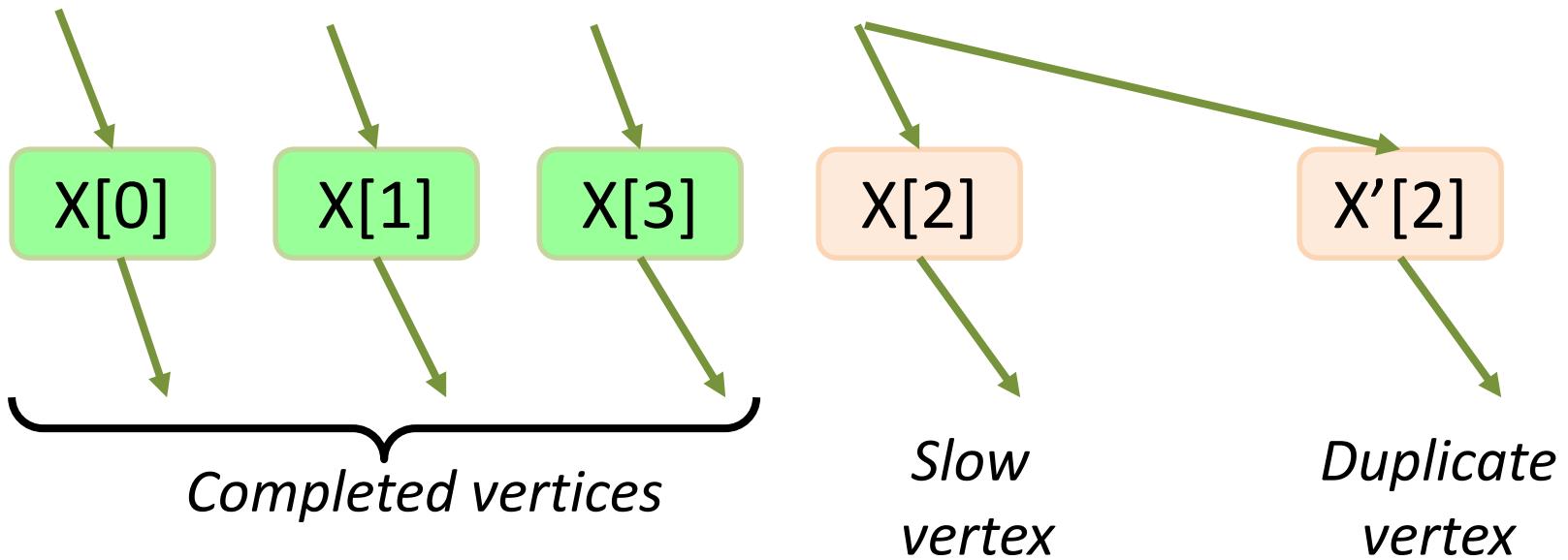
Failures



- Fail-stop (crash) failures are easiest to handle
- Many other kinds of failures possible
 - Very slow progress
 - Byzantine (malicious) failures
 - Network partitions
- Understand the failure model for your system
 - probability of each kind of failure
 - validate the failure model (measurements)



Dynamic Graph Rewriting



Duplication Policy = $f(\text{running times}, \text{data volumes})$

Dynamic Aggregation



static



rack #

dynamic

Separate policy and mechanism

- Implement a powerful and generic mechanism
- Leave policy to the application layer
- Trade-off in policy language:
power vs. simplicity





Policy vs. Mechanism



- Application-level
- Most complex in C++ code
- Invoked with upcalls
- Need good default implementations
- DryadLINQ provides a comprehensive set
- Built-in
 - Scheduling
 - Graph rewriting
 - Fault tolerance
 - Statistics and reporting

Dryad Abstraction

Reliable machine running distributed jobs with
“infinite” resources



Distributed Execution

Cluster storage

Cluster services

Deployment

Machine

Machine

Machine

Machine

Thinking in Parallel



Computing



Definition of computer (n)

bing.com · Bing Dictionary

com·put·er [kəm pyoȯtər] 

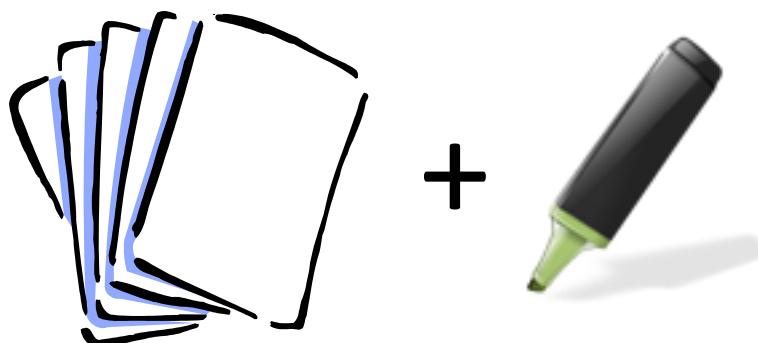
1. electronic data processor: an electronic device that accepts, processes, stores, and outputs data at high speeds according to programmed instructions
2. somebody who computes: somebody who calculates numbers or amounts using a machine

Synonyms: processor, CPU, mainframe, supercomputer, workstation, PC, laptop, notebook, palmtop

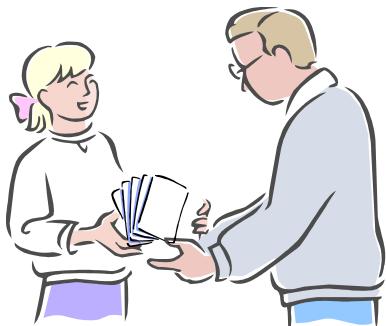
Logistics



= input data



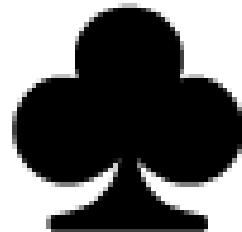
= output data



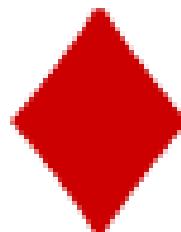
= communication
is expensive

Playing Cards

suits



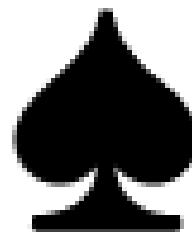
Clubs



Diamonds



Hearts



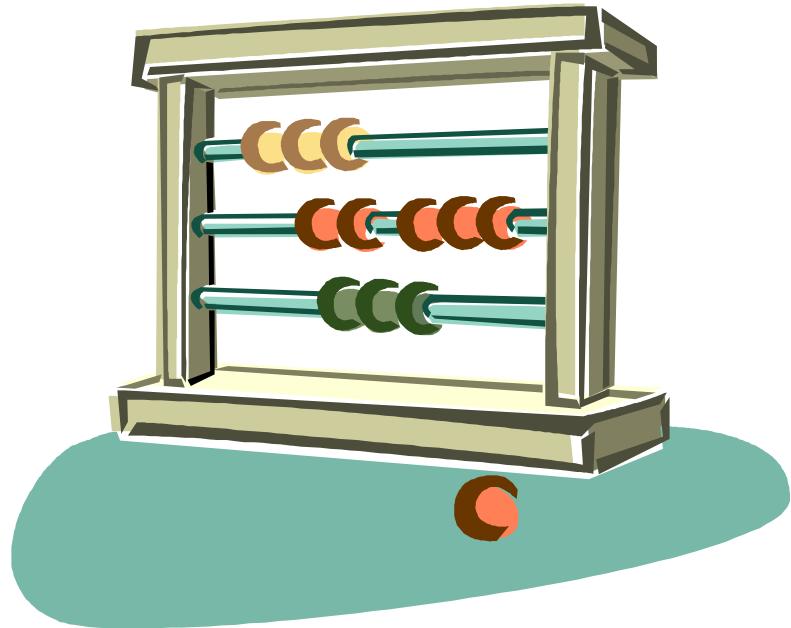
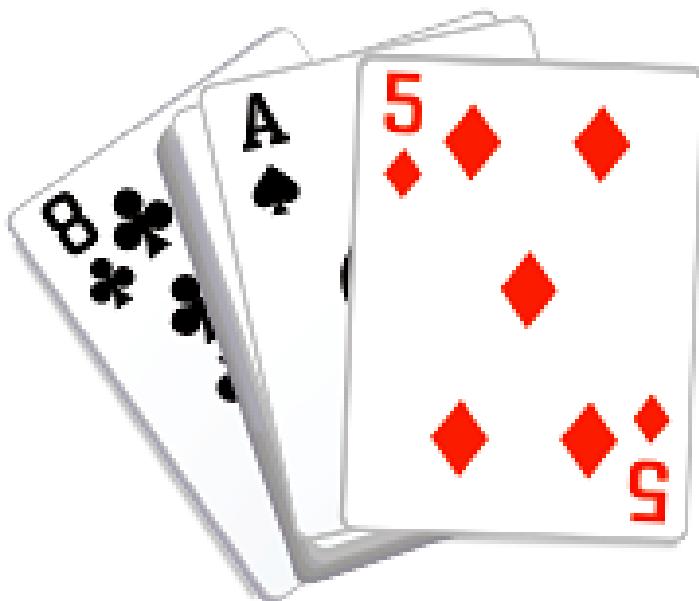
Spades



2,3,4,5,6,7,8,9,10,J,Q,K,A

our order

Count the number of cards

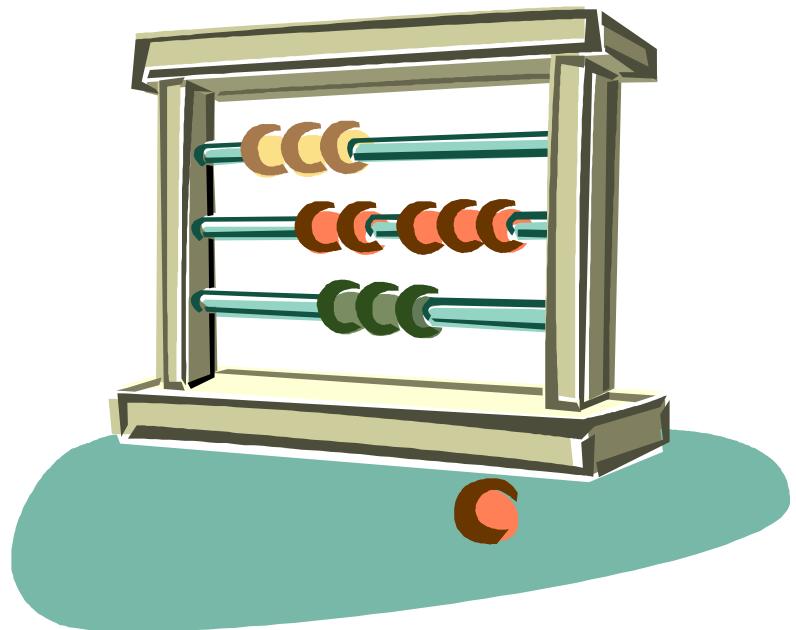


Counting, or aggregation

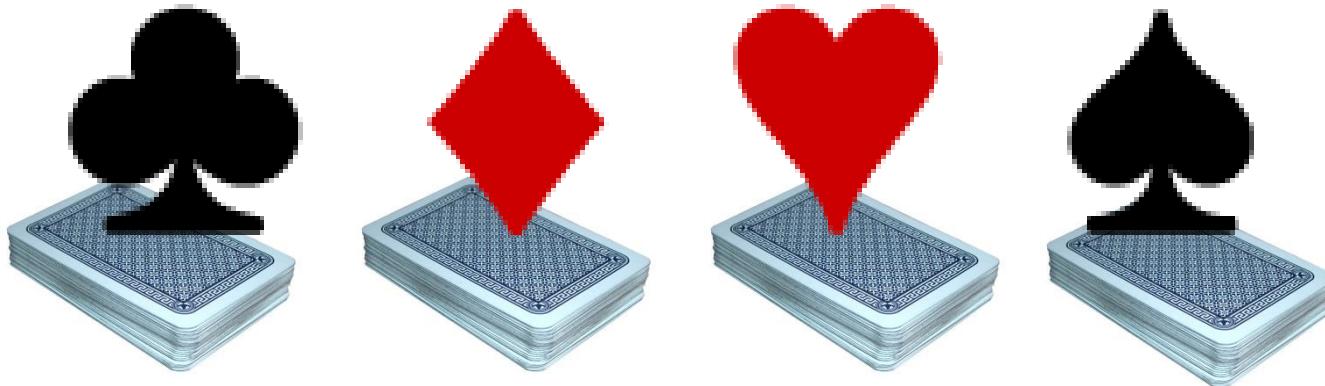
Keep only the even cards

2,3,4,5,6,7,8,9,10,J,Q,K,A

Count the Number of Figures



Group cards by suit

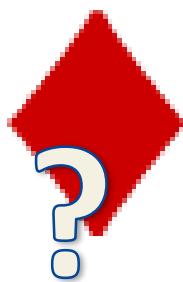


Which cards are missing?

2,3,4,5,6,7,8,9,10,J,Q,K,A



Number of cards of each suit



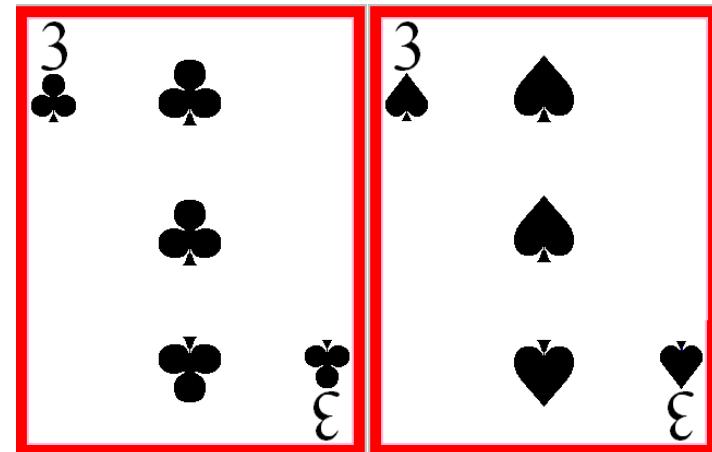
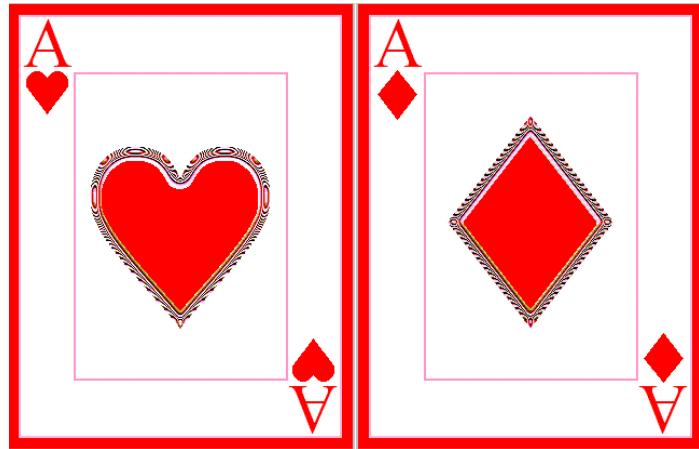
Sort the cards

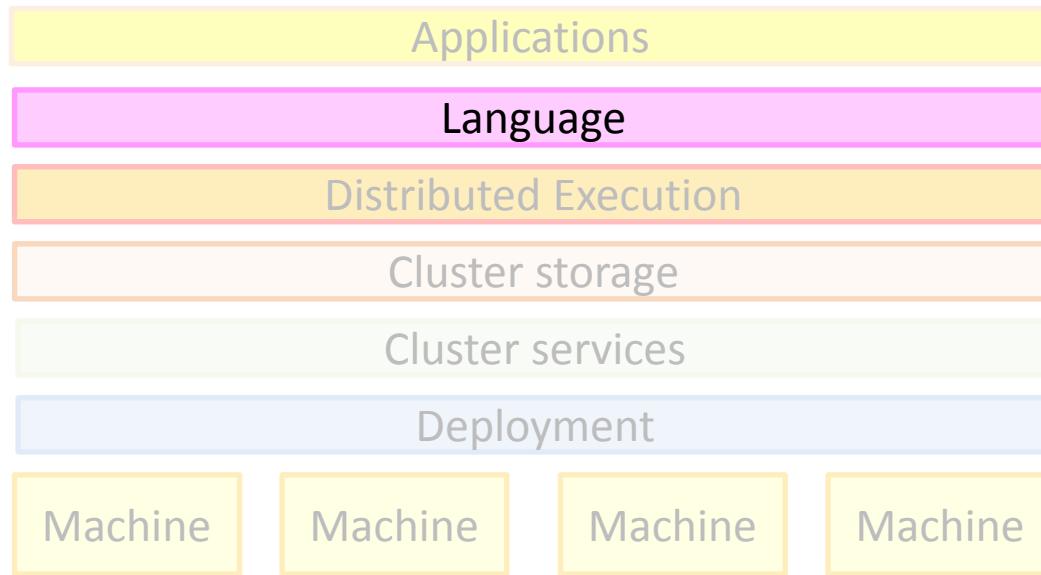


lexicographic
order

2,3,4,5,6,7,8,9,10,J,Q,K,A

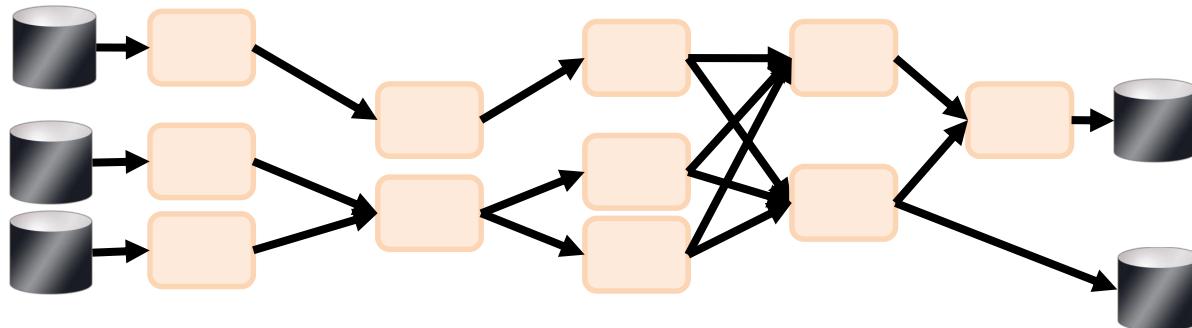
Cards with same color & number



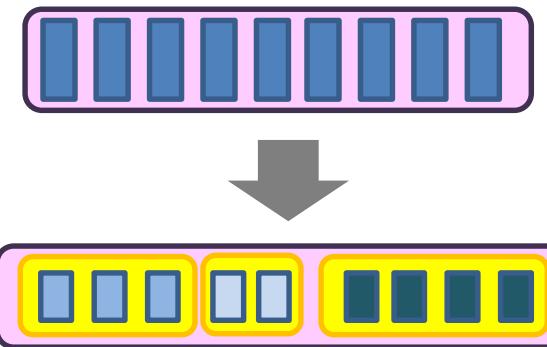


DRYADLINQ

DryadLINQ = Dryad + LINQ

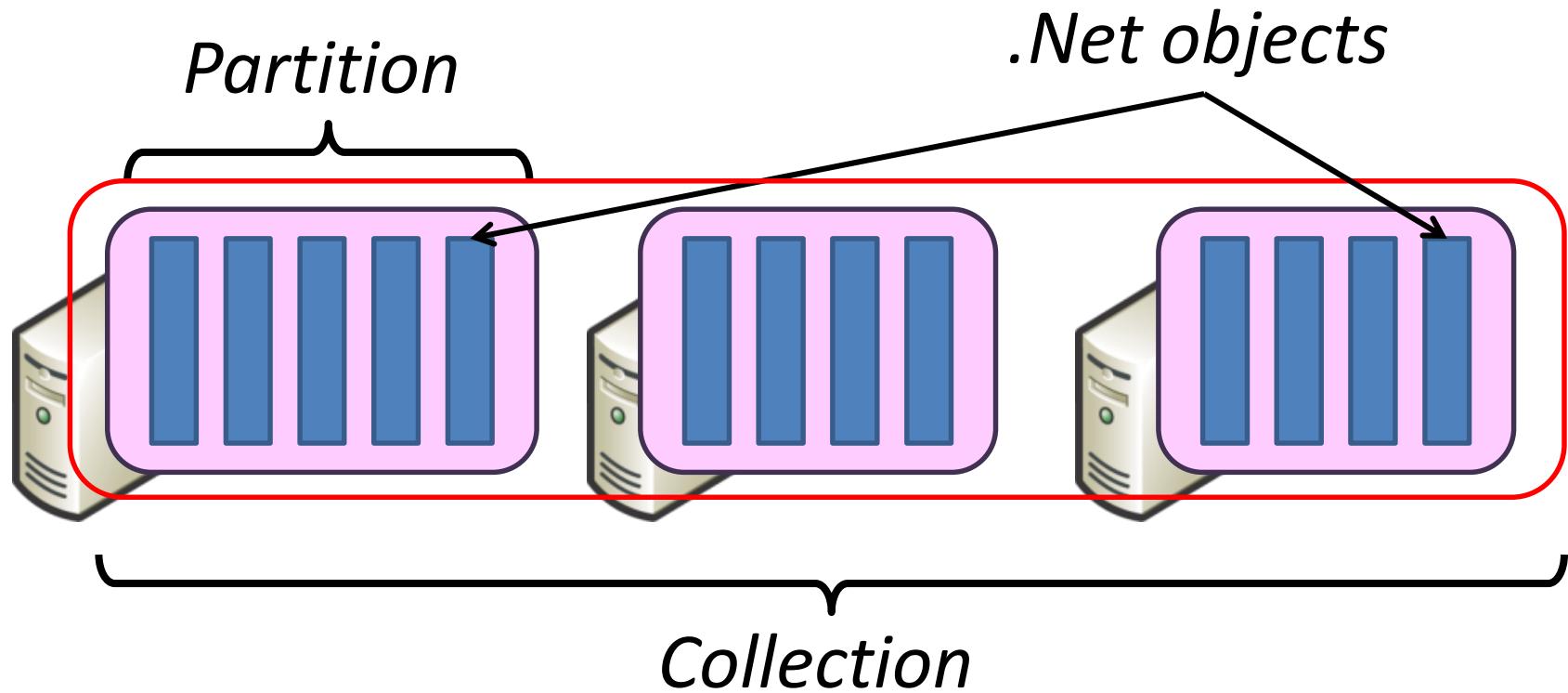


Distributed computations

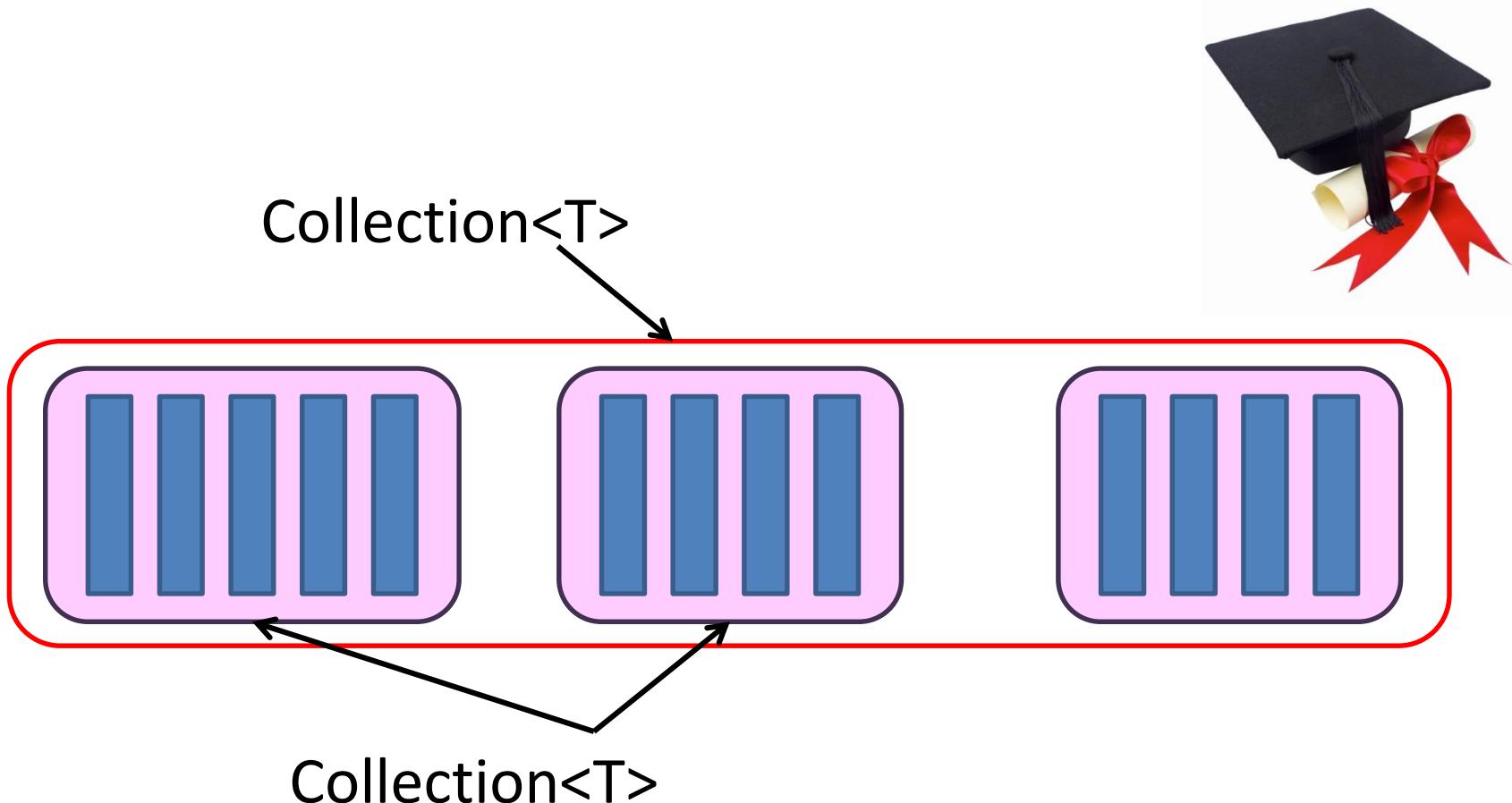


Computations on collections

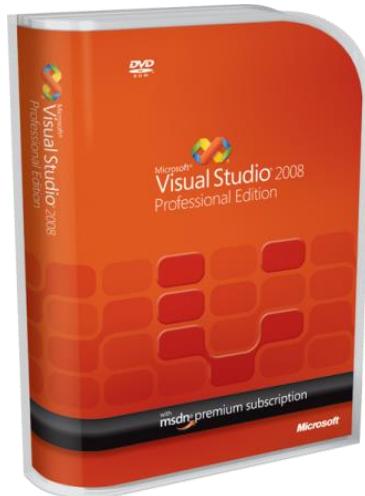
Distributed Collections



Collection = Collection of Collections



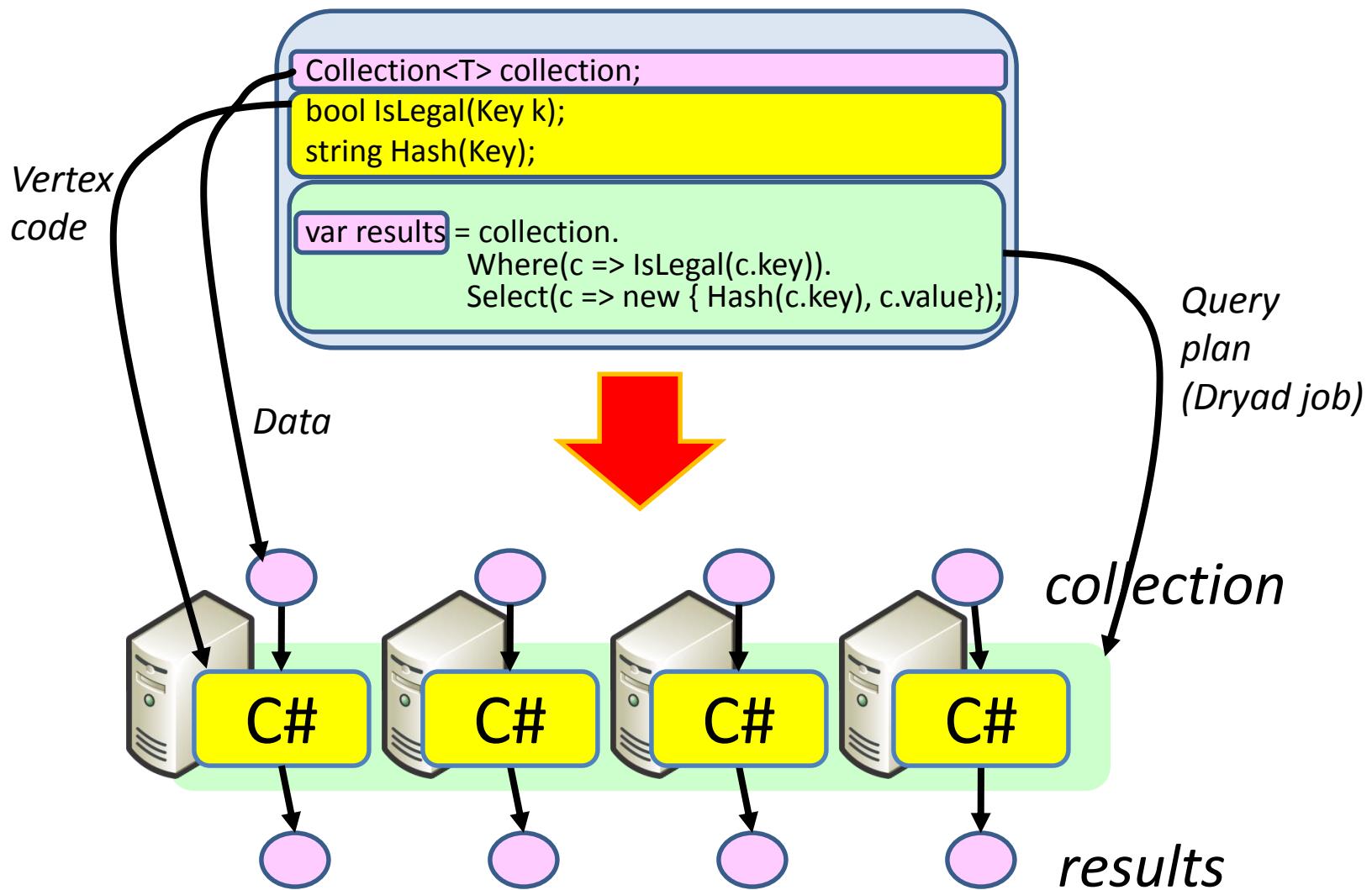
LINQ => DryadLINQ



Dryad



DryadLINQ = LINQ + Dryad



DryadLINQ source code

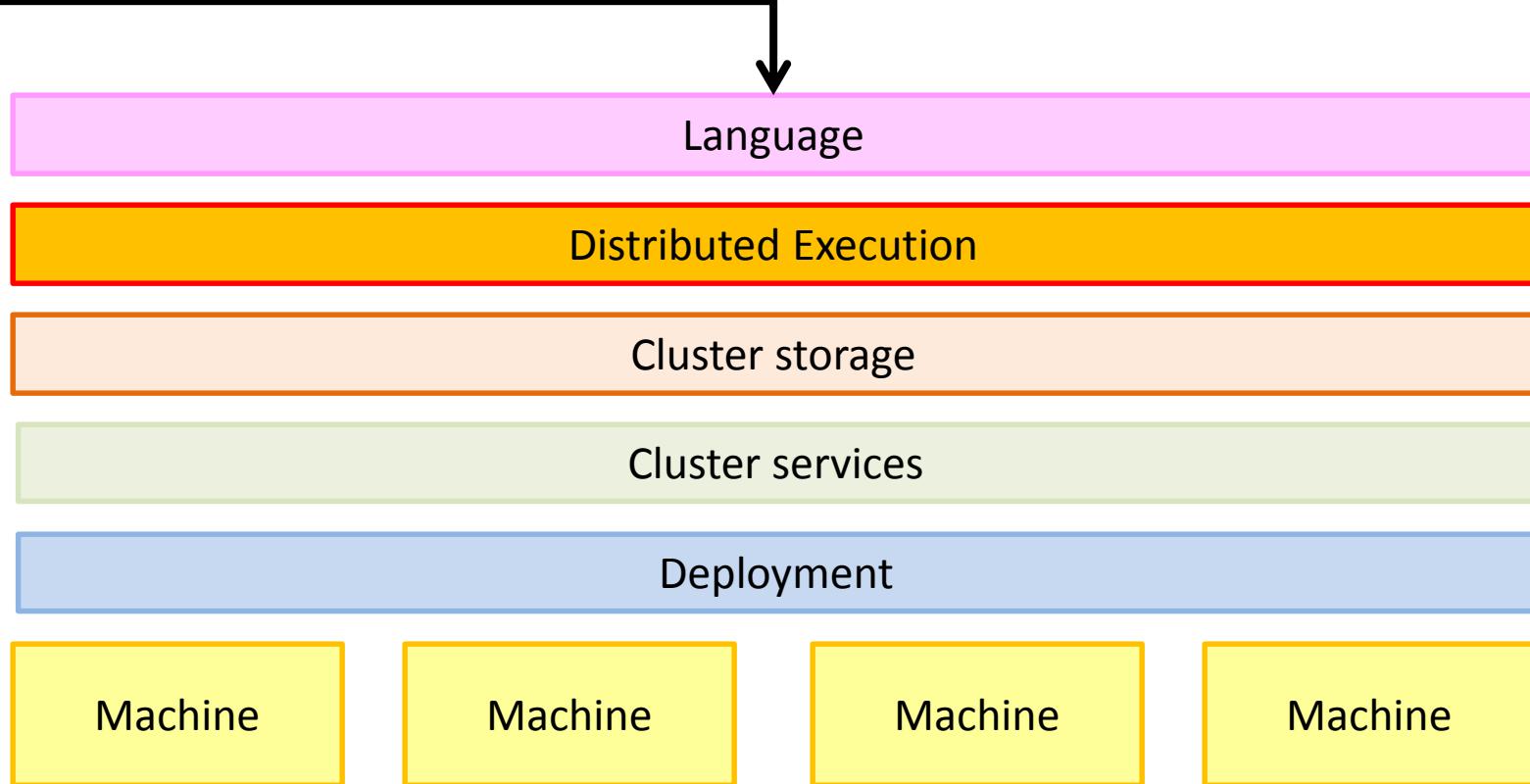
- <https://github.com/MicrosoftResearchSVC/Dryad>
- Apache license
- Runs on Hadoop YARN
- Research prototype code quality

The screenshot shows the GitHub interface for the repository 'MicrosoftResearchSVC / Dryad'. At the top, there's a navigation bar with links for 'Explore', 'Features', 'Enterprise', and 'Blog'. Below the navigation is a search bar with the placeholder 'Search or type a command'. To the left of the search bar is a dropdown menu labeled 'This repository'. The main area displays the repository name 'MicrosoftResearchSVC / Dryad' next to a small icon of a document with a star.

This is a research prototype of the Dryad and DryadLINQ data-parallel processing frameworks running on Hadoop YARN.

DryadLINQ Abstraction

.Net/LINQ with “infinite” resources



Demo

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "Count - Microsoft Visual Studio". The menu bar includes File, Edit, View, Refactor, Project, Build, Debug, Data, Tools, Test, Analyze, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Print. The status bar at the bottom shows "Ready", "Ln 23", "Col 1", "Ch 1", and "INS".

The code editor displays a C# program named Program.cs:

```
using System;
using System.Linq;
using LinqToDryad;

namespace Count
{
    public class Program
    {
        static void ShowOnConsole<T>(IQueryable<T> data)...

        static void Main(string[] args)
        {
            config

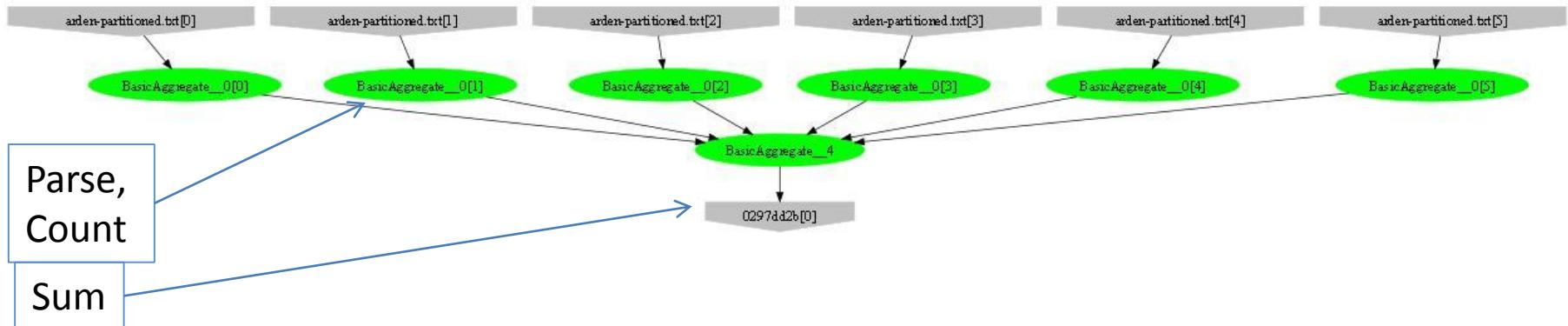
            PartitionedTable<LineRecord> table = PartitionedTable.Get<LineRecord>(fileDir + smallfile);
            var result = table.SelectMany(l => l.line.Split(' '))
                .GroupBy(w => w[0])
                .Select(g => g.Count())
                .OrderBy(c => -c);

            ShowOnConsole(result);
            //Console.WriteLine("{0}", result);
            Console.ReadKey();
        }
    }
}
```

The code uses LINQ to query a PartitionedTable and output the results to the console. The "Main" method reads command-line arguments, configures the table, processes it, and then reads a key from the console.

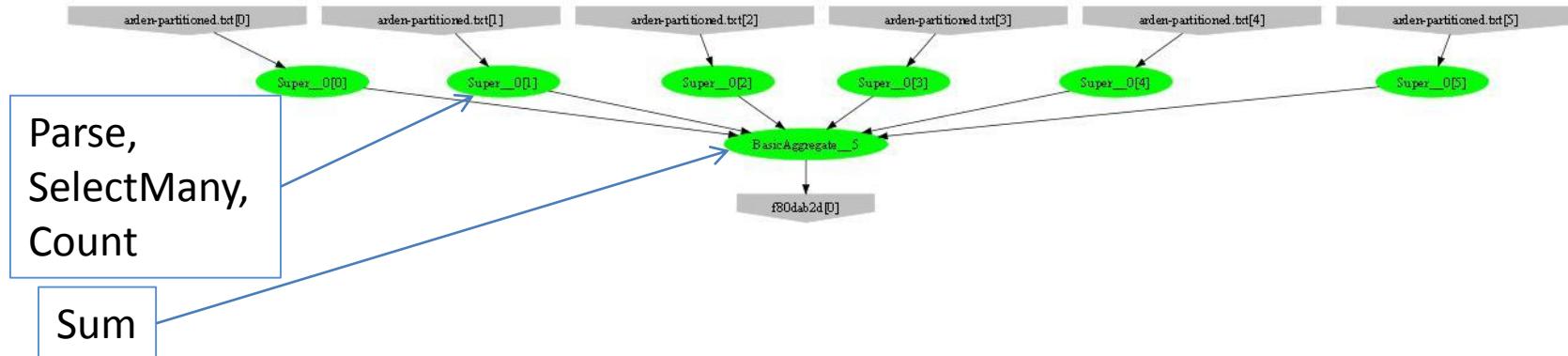
Example: counting lines

```
var table = PartitionedTable.Get<LineRecord>(file);
int count = table.Count();
```



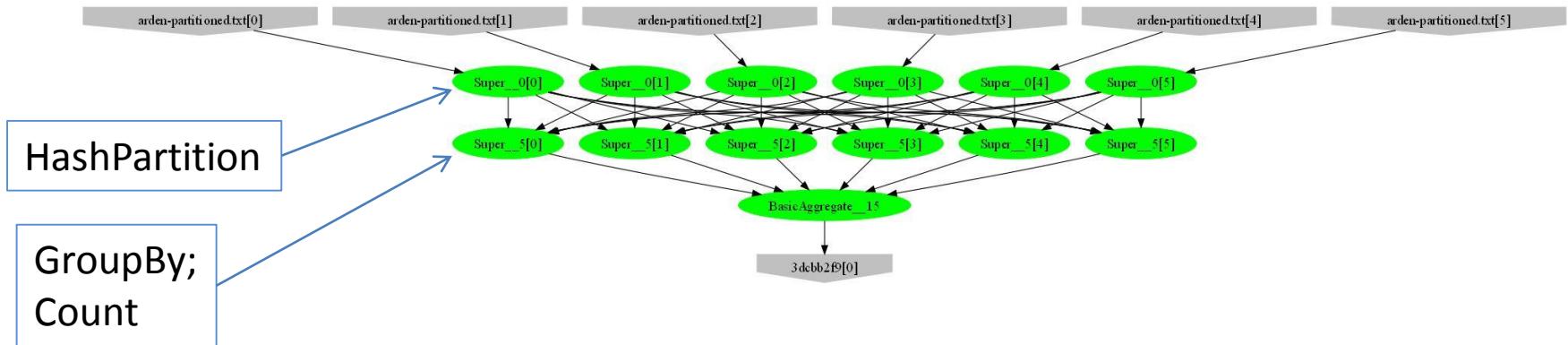
Example: counting words

```
var table = PartitionedTable.Get<LineRecord>(file);
int count = table
    .SelectMany(l => l.line.Split(' '))
    .Count();
```



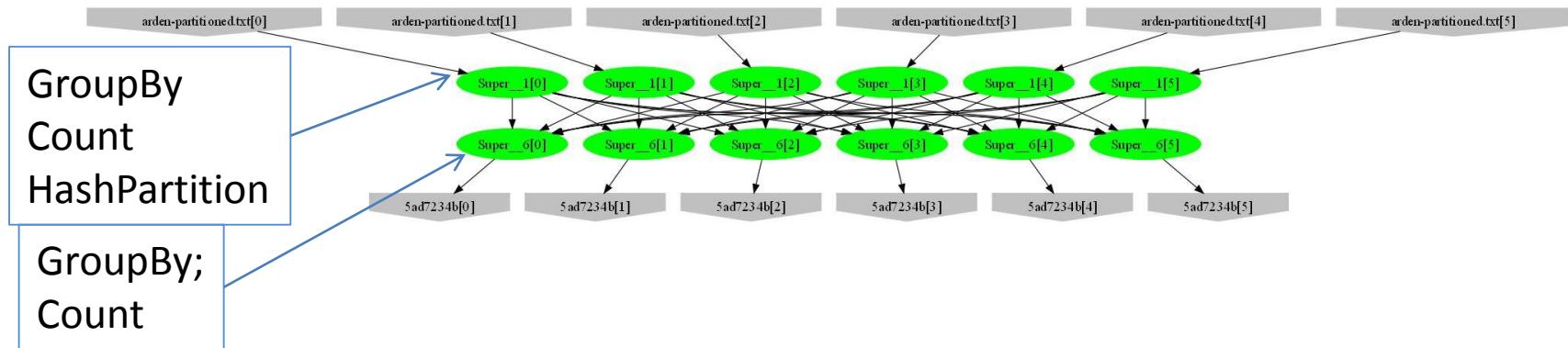
Example: counting unique words

```
var table = PartitionedTable.Get<LineRecord>(file);
int count = table
    .SelectMany(l => l.line.Split(' '))
    .GroupBy(w => w)
    .Count();
```



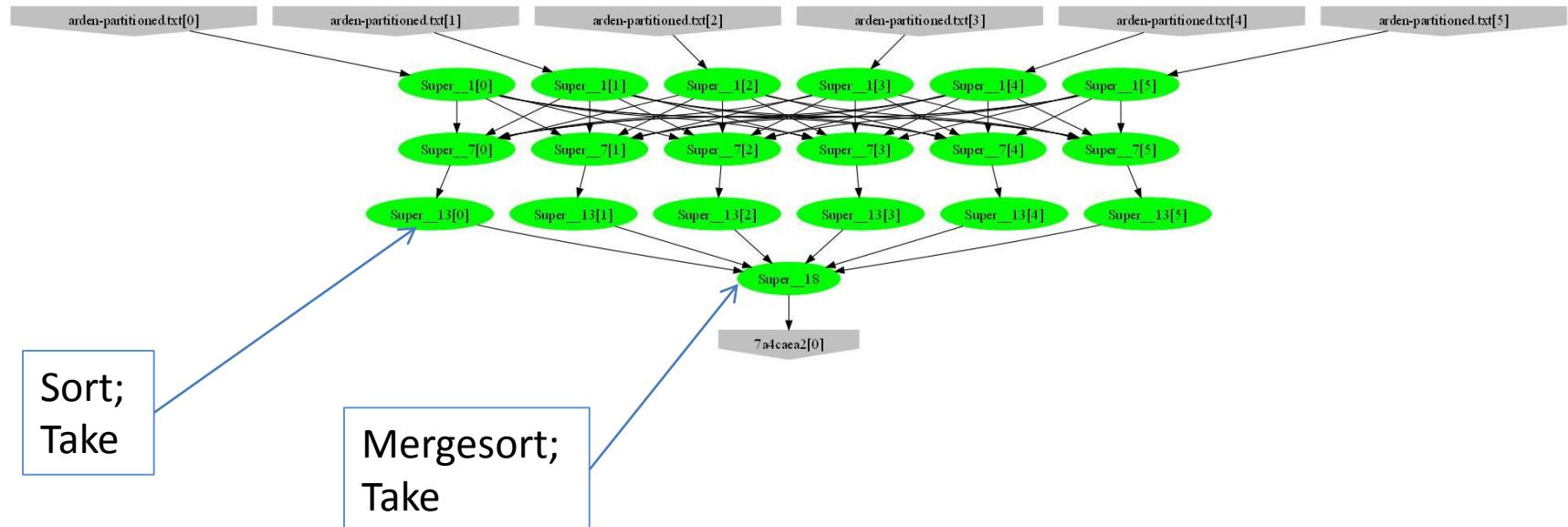
Example: word histogram

```
var table = PartitionedTable.Get<LineRecord>(file);
var result = table.SelectMany(l => l.line.Split(' '))
    .GroupBy(w => w)
    .Select(g => new { word = g.Key, count = g.Count() });
```



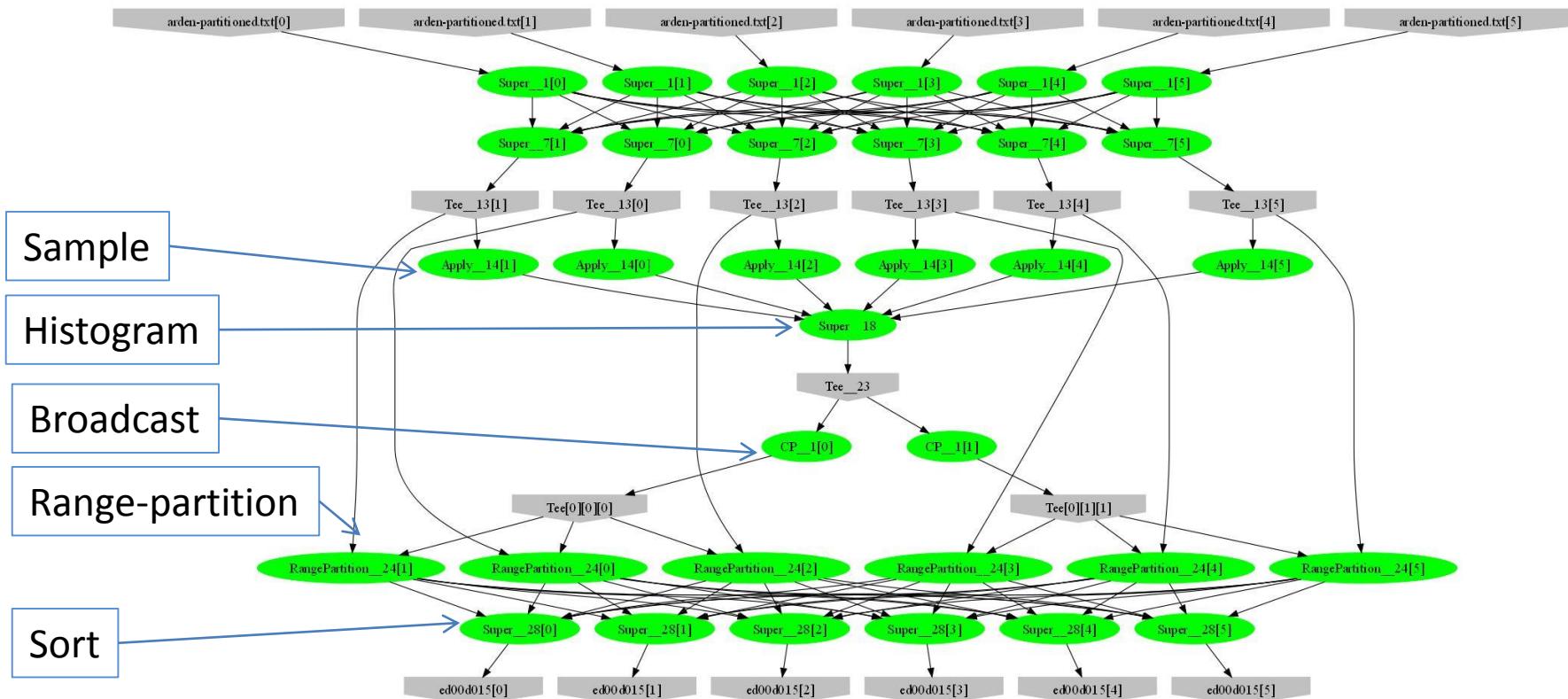
Example: high-frequency words

```
var table = PartitionedTable.Get<LineRecord>(file);
var result = table.SelectMany(l => l.line.Split(' '))
    .GroupBy(w => w)
    .Select(g => new { word = g.Key, count = g.Count() })
    .OrderByDescending(t => t.count)
    .Take(100);
```



Example: words by frequency

```
var table = PartitionedTable.Get<LineRecord>(file);
var result = table.SelectMany(l => l.line.Split(' '))
    .GroupBy(w => w)
    .Select(g => new { word = g.Key, count = g.Count() })
    .OrderByDescending(t => t.count);
```

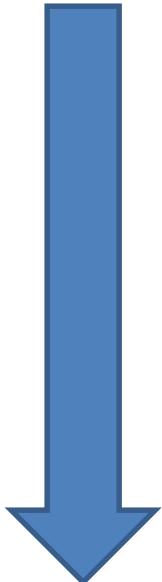


Example: Map-Reduce

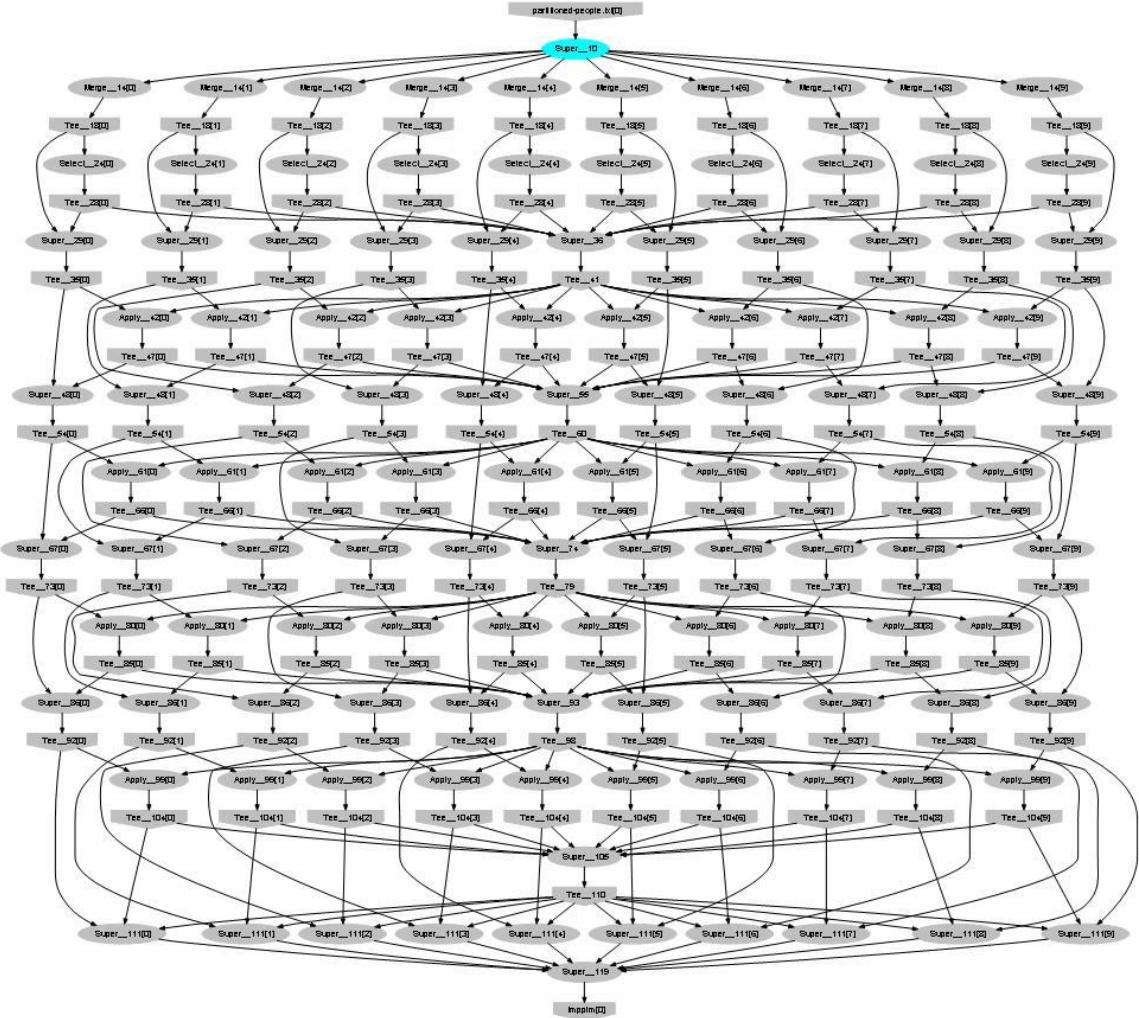
```
public static IQueryable<S>
MapReduce<T,M,K,S>(
    IQueryable<T> input,
    Func<T, IEnumerable<M>> mapper,
    Func<M,K> keySelector,
    Func<IGrouping<K,M>,S> reducer)
{
    var map = input.SelectMany(mapper);
    var group = map.GroupBy(keySelector);
    var result = group.Select(reducer);
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

Probabilistic Index Maps

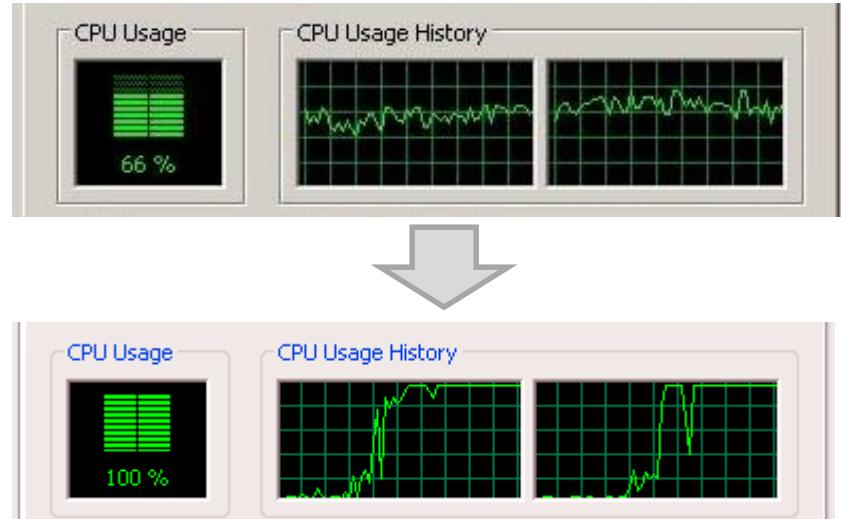
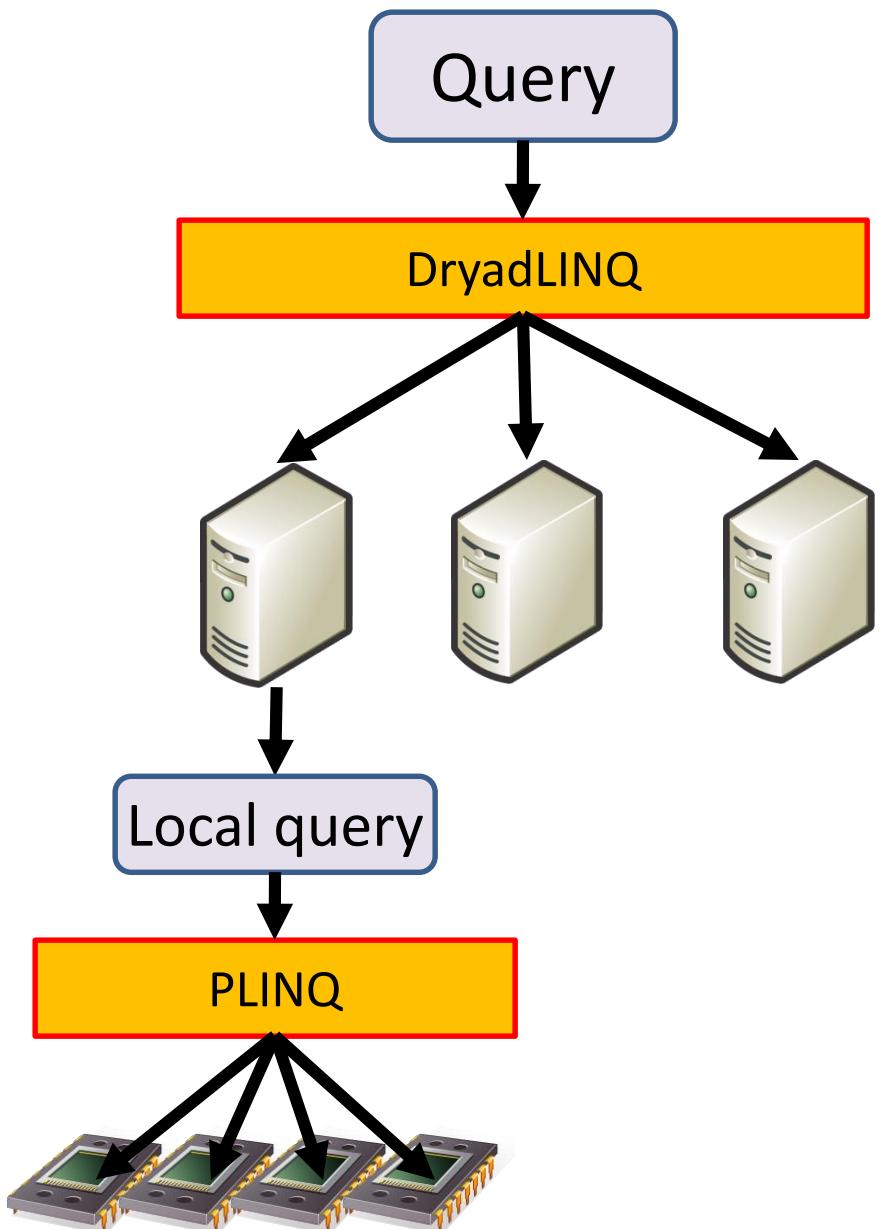
Images



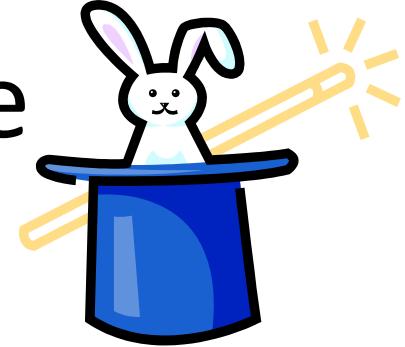
features



Parallelizing on Cores

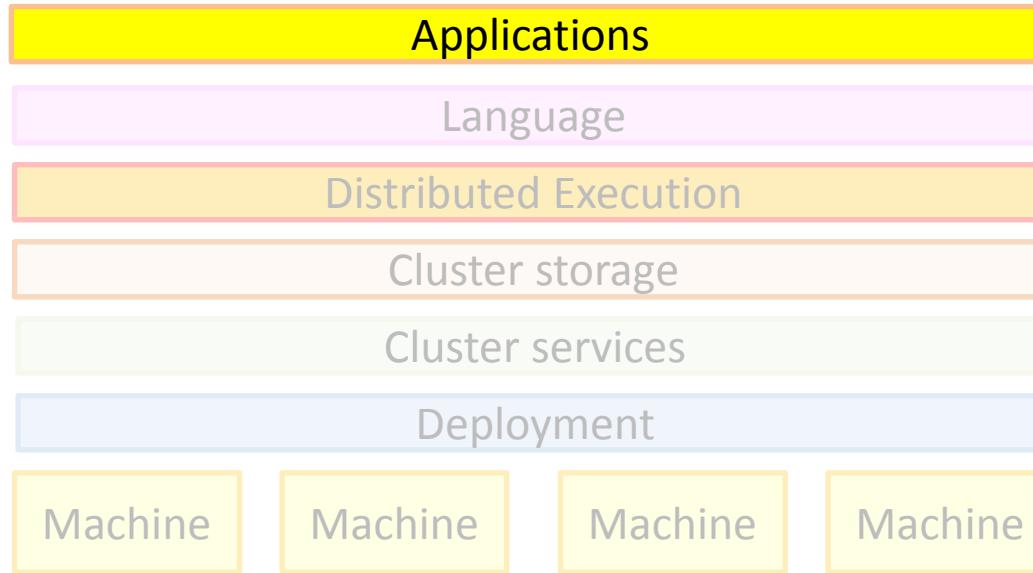


More Tricks of the trade



- Asynchronous operations hide latency
- Management using distributed state machines
- Logging state transitions for debugging
- Compression trades-off bandwidth for CPU





BUILDING ON DRYADLINQ

Job Visualization

Job: INDEX [updated@9/9/2010 11:19:12 AM]

ObjectName	Value
StartJMTTime	9/9/2010 11:05:45 AM
LastUpdateTime	9/9/2010 11:19:12 AM
EndTime	9/9/2010 11:09:05 AM
RunningTime	00:03:20.1511260

Plan Static Auto Refresh

```

graph TD
    A[1999 x OSUAIPIP_allweb_bindings.pf] --> B((1999 x Apply__123))
    B --> C[1999 x Apply__129]
    C --> D[1999 x 75416bc0]
    D --> E{JobManager}
  
```

Stage: Apply__123

ObjectName	Value
TotalVertices	2250
CreatedVertices	0
StartedVertices	0
FailedVertices	92

Vertex: Apply__123[0] Show: stdout

ObjectName	Value
State	Successful
WorkDirectory	\\\sherwood-122\dyaddata\pn\Processes\6AB9CA68-BA5E-4D94-A0...
DataRead	-1
DataWritten	-1

Find prev next filter X

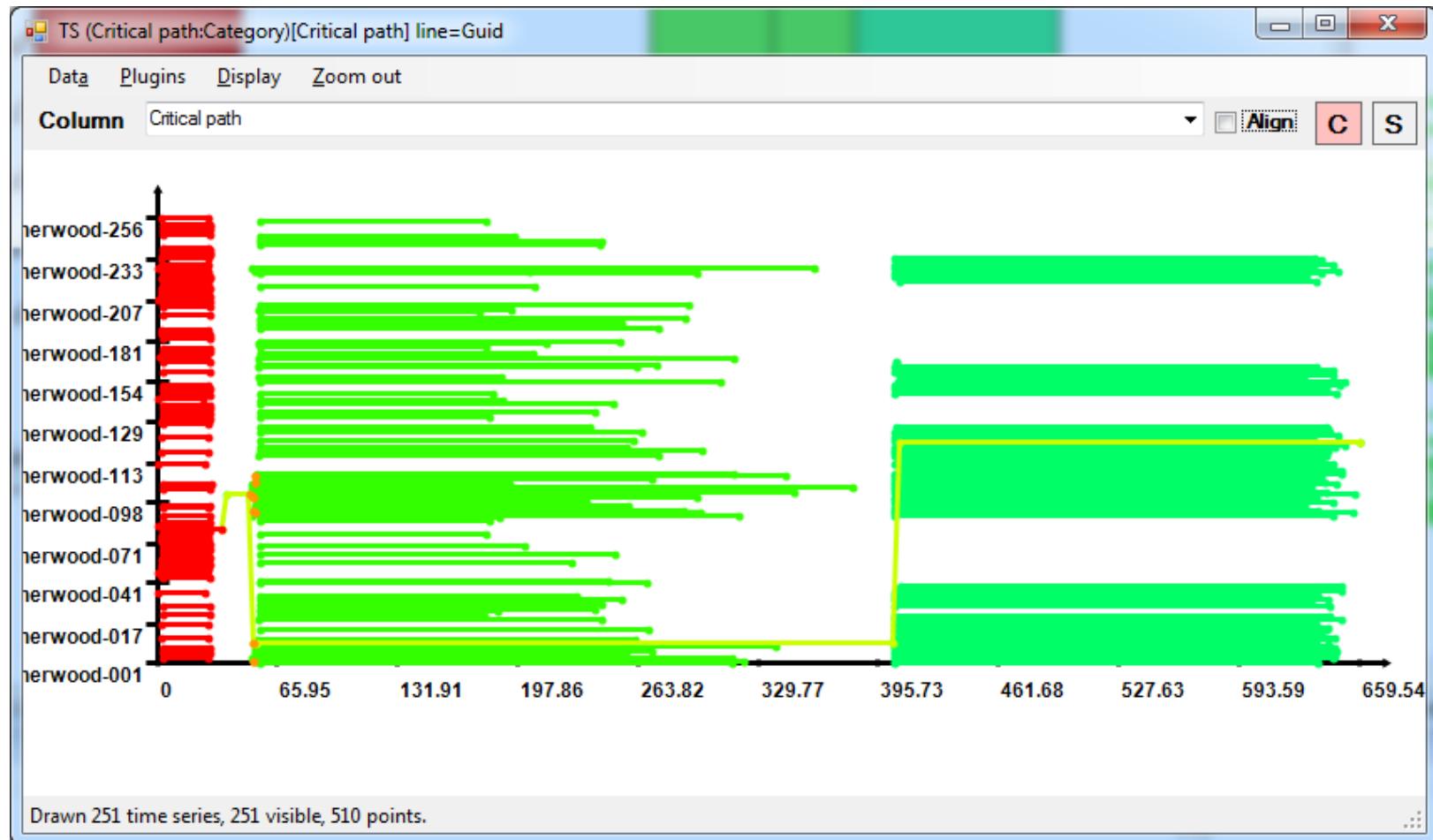
```

Starting ManagedWrapperVertex Main with 4 arguments
ManagedWrapperVertex: 00000000ED2430 1 1
ManagedWrapperVertex: Calling LinqToDryad.DryadLinq__Vertex.Apply__123
ManagedWrapperVertex: Binding to runtime
ManagedWrapperVertex: Starting managed runtime
ManagedWrapperVertex: Managed runtime started
ManagedWrapperVertex: Invoking CLR.
DryadLinq: Vertex Apply__123 started at 09/09/2010 11:05:53.882
DryadLinq: Read 336980 records from DryadChannel[0] from 09/09/2010 11:05:54.
DryadLinq: Input channel 0 was closed.
DryadLinq: Async writer with buffer size 1024
DryadLinq: Output channel 0 was closed.
DryadLinq: Wrote 29905 records to DryadChannel[0]
DryadLinq: Vertex Apply__123 completed at 09/09/2010 11:05:55.851
ManagedWrapperVertex: Cleaning up NativeInfo at 00000000ED2430
The client did not completely read channels: {}
WrapperNativeInfo read 16020792 bytes from channel 0.

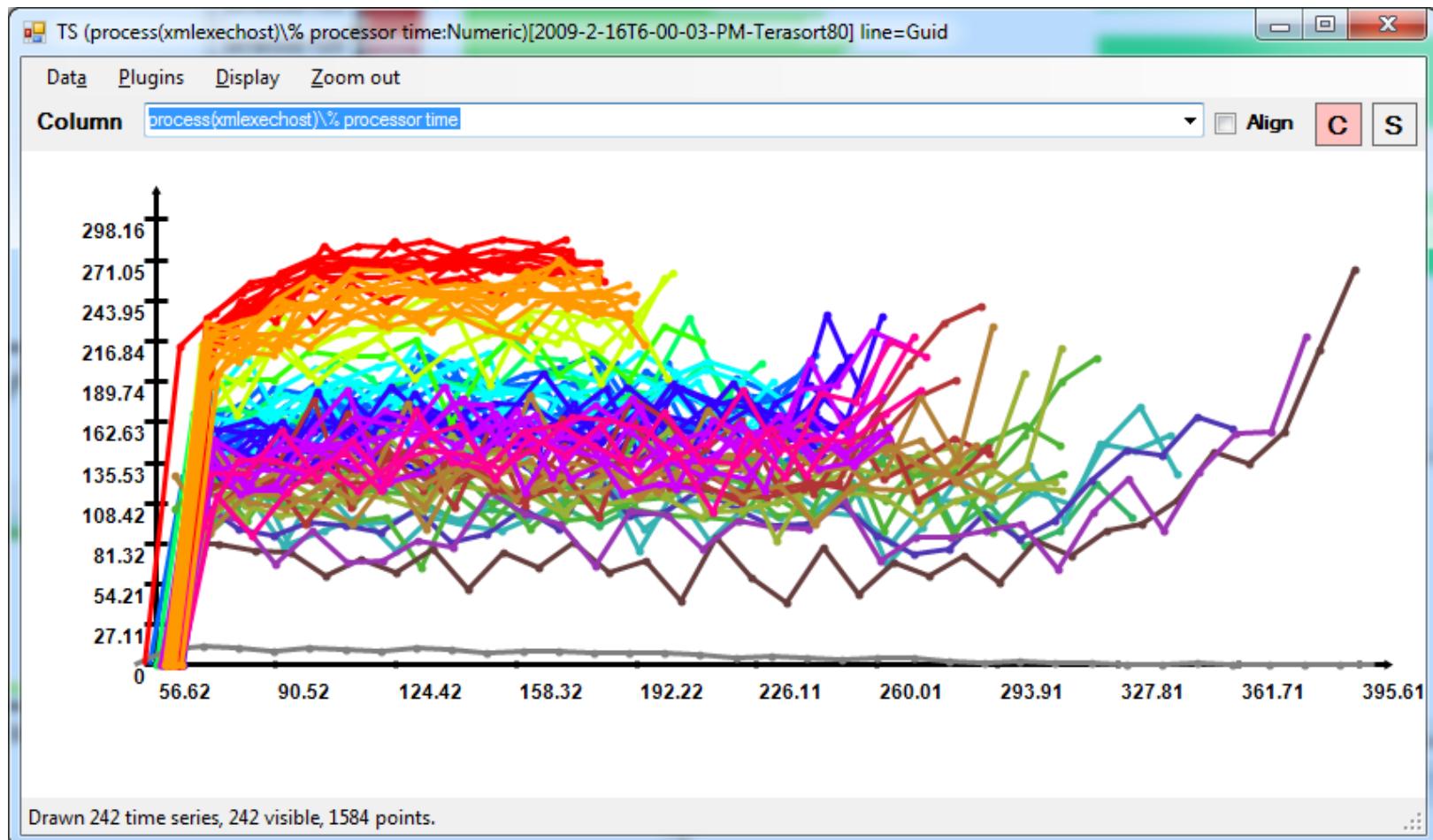
WrapperNativeInfo wrote 239240 bytes to channel 0.
WrapperNativeInfo read 16020792 bytes from all channels.
WrapperNativeInfo wrote 239240 bytes to all channels.

Doing nothing. 0 pending activities.
  
```

Job Schedule



CPU Utilization



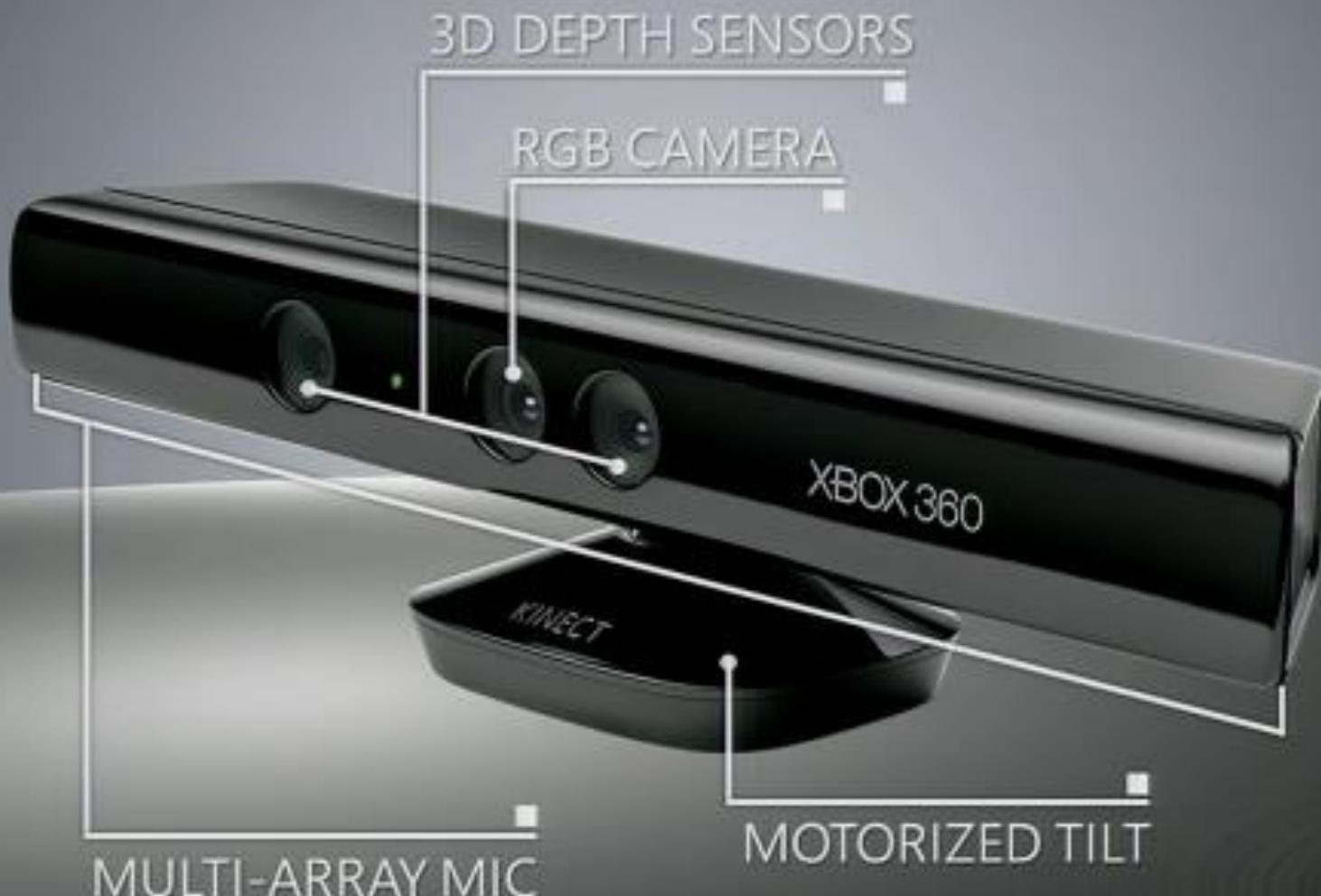
So, What Good is it For?



KINECT™

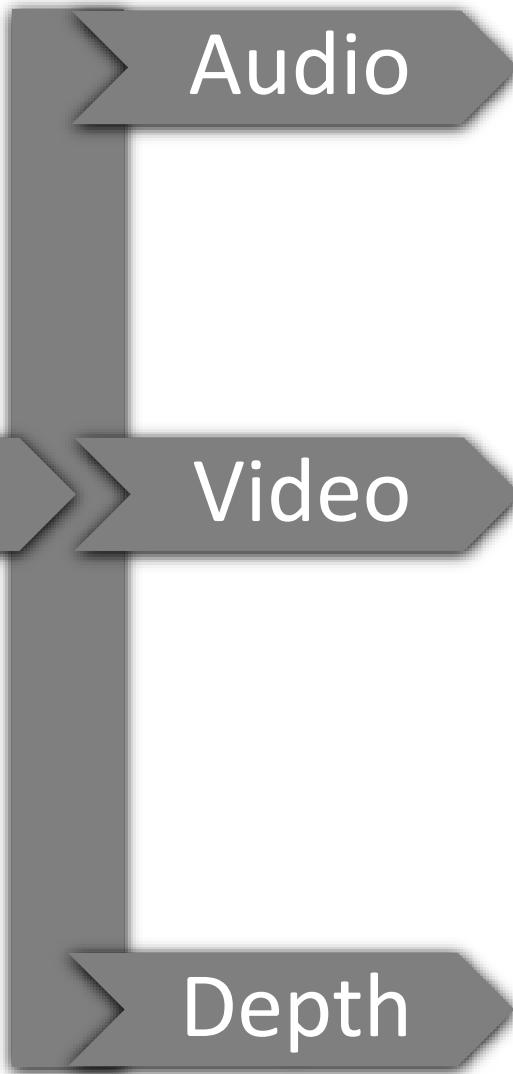


Input Device

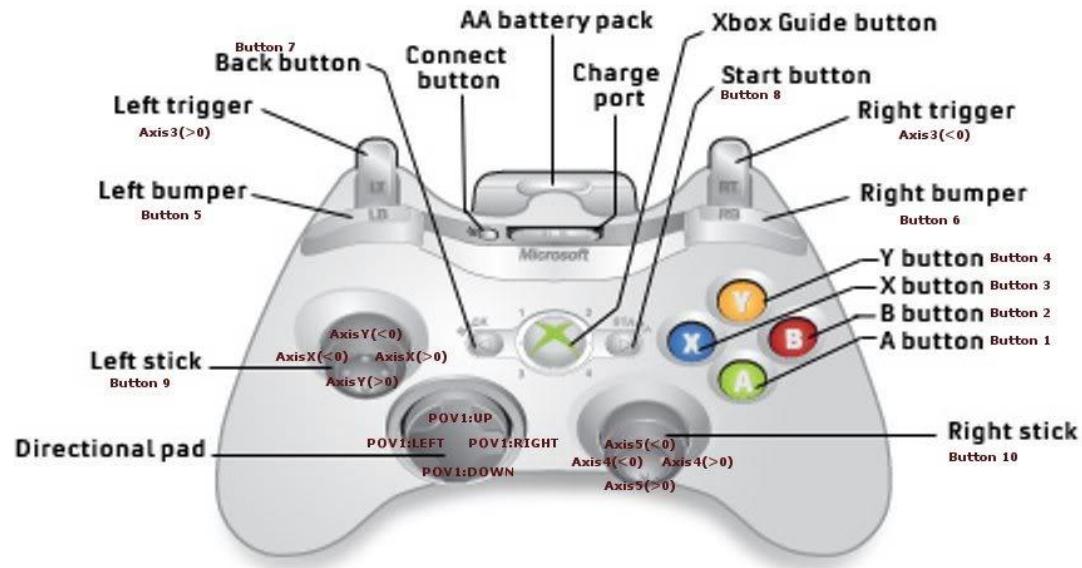


XBOX
LIVE

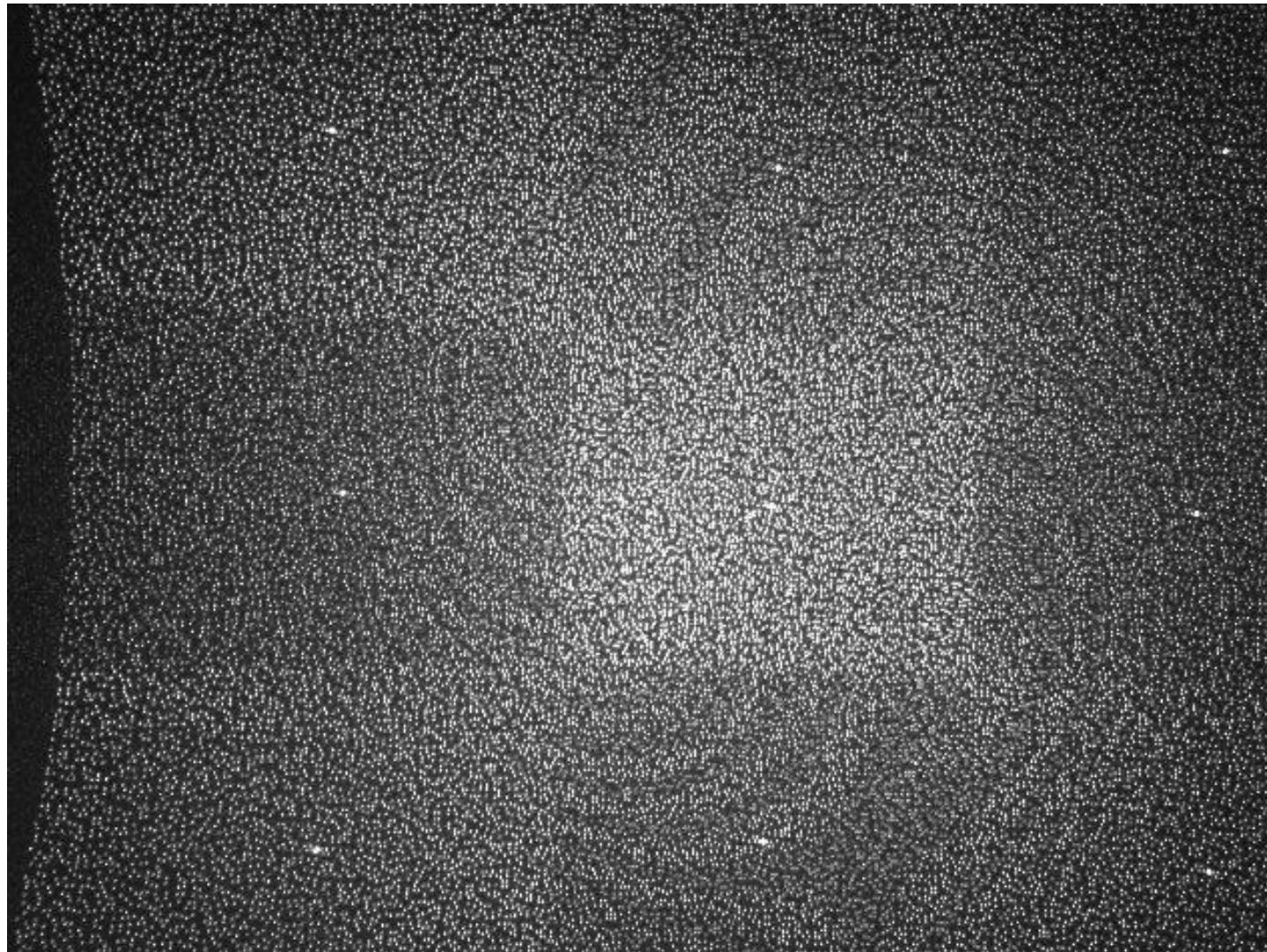
Data Streams



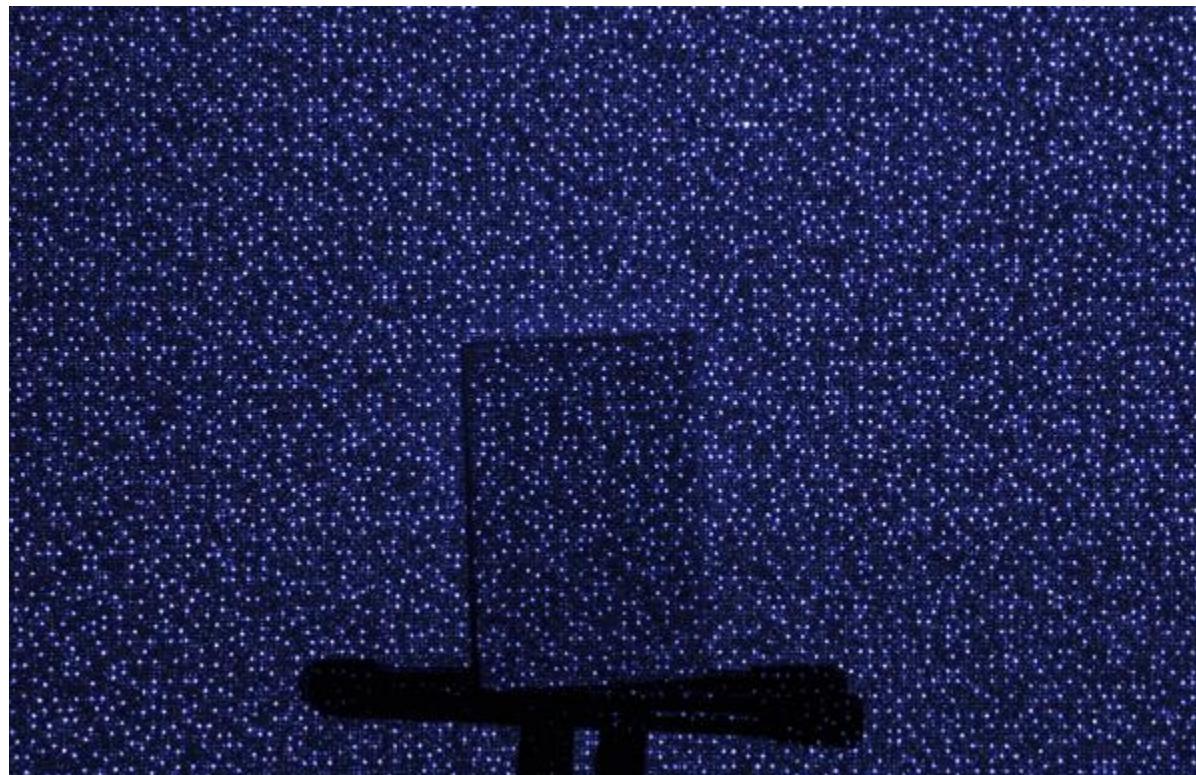
“the Controller”



Projected IR pattern



Depth computation



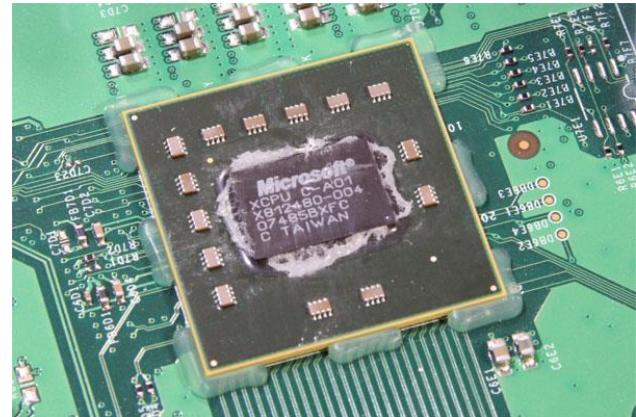
Source: <http://nuit-blanche.blogspot.com/2010/11/using-kinect-for-compressive-sensing.html>

The Body Tracking Problem



XBox 360 Hardware

- Triple Core PowerPC 970, 3.2GHz
- Hyperthreaded, 2 threads/core
- 500 MHz ATI graphics card
- DirectX 9.5
- 512 MB RAM
- 2005 performance envelope
- Must handle
 - real-time vision AND
 - a modern game

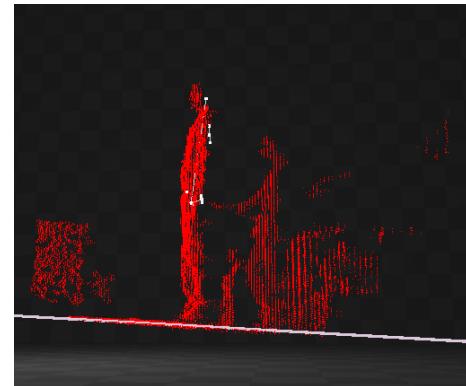


Source: <http://www.pcper.com/article.php?aid=940&type=expert>

Tracking Pipeline



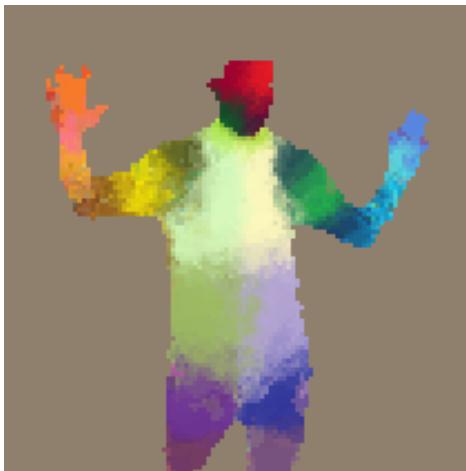
Sensor



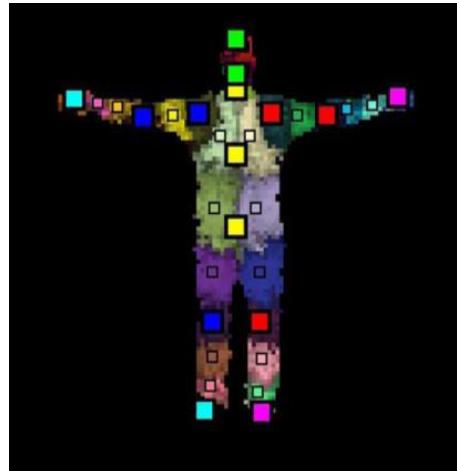
Depth map



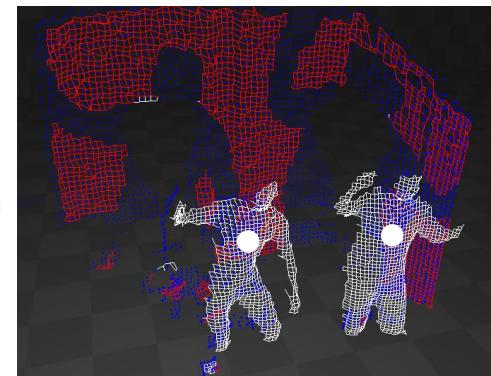
Background elimination
Player segmentation



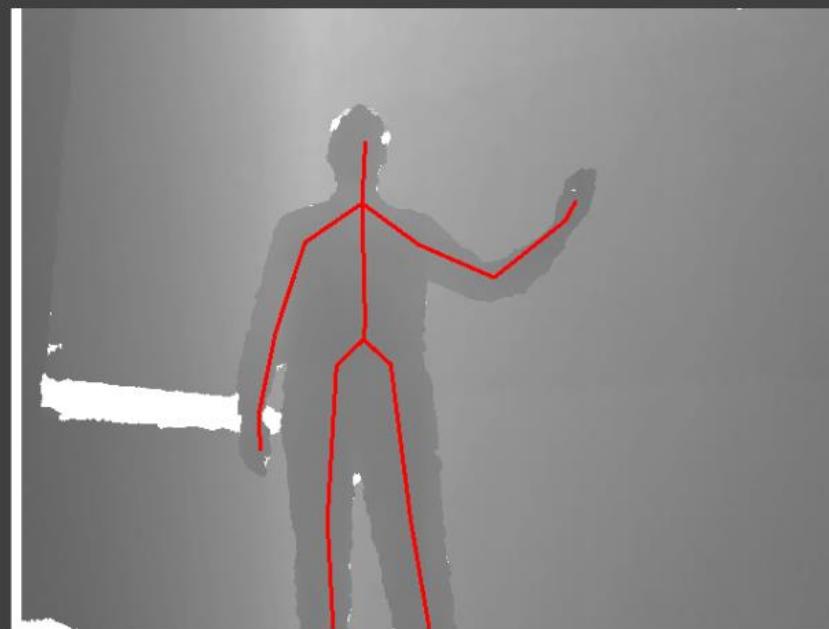
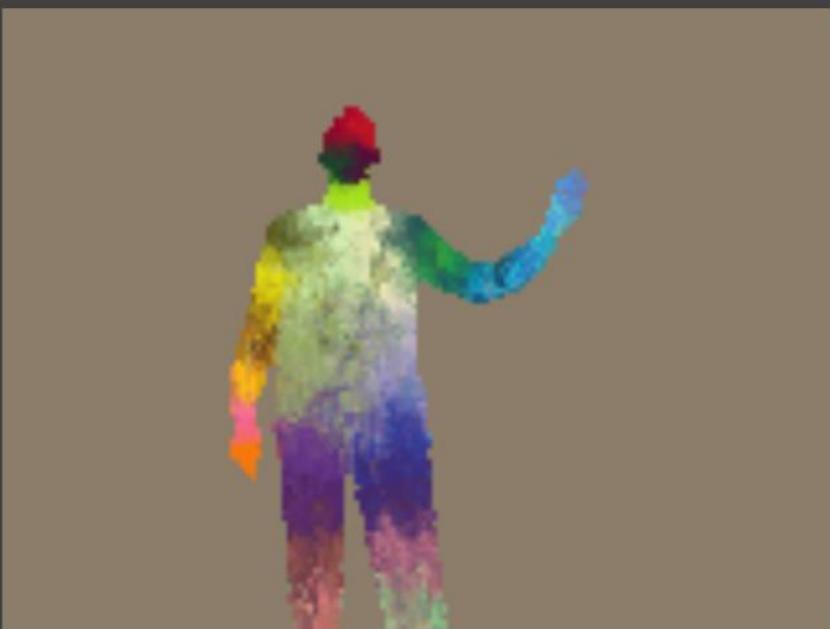
Body Part Classifier



Body Part
Identification



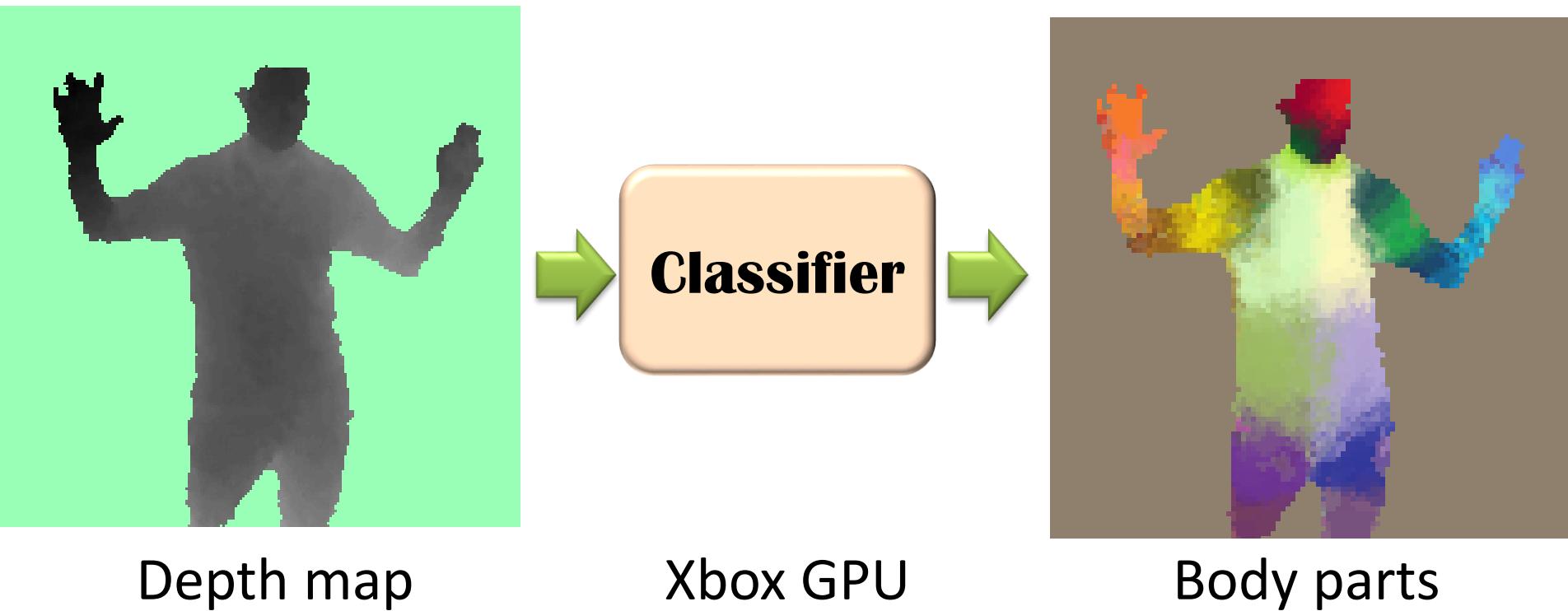
Skeleton



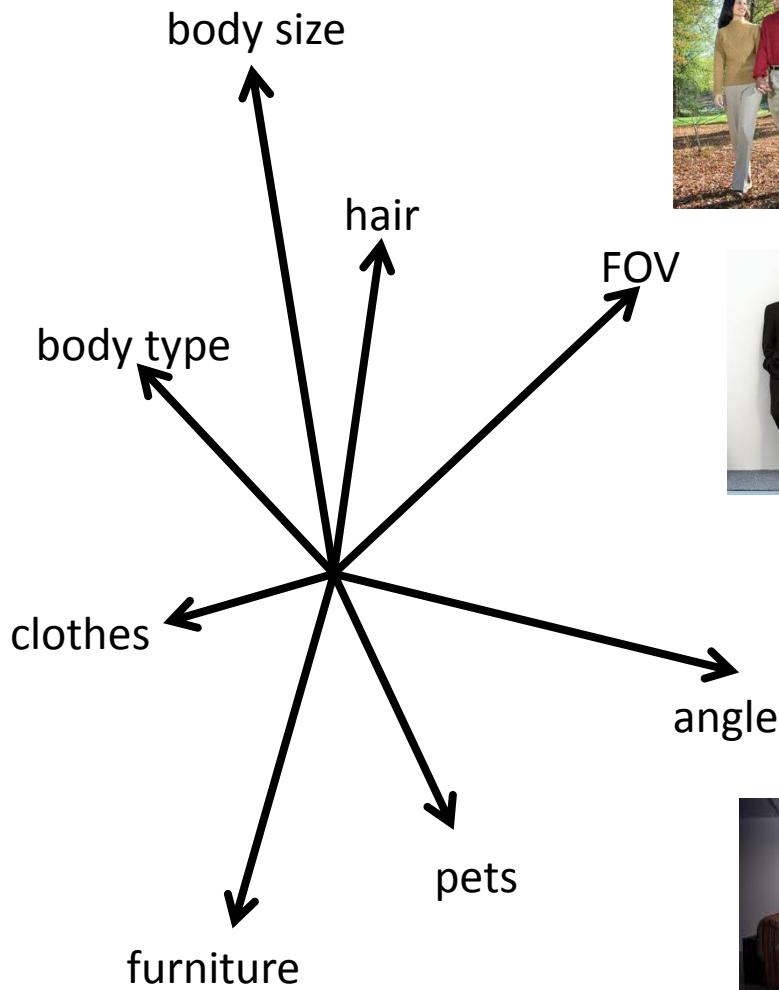
RGB: 30.0 fps

Depth: 30.0 fps

From Depth Map to Body Parts



What is the Shape of a Human?

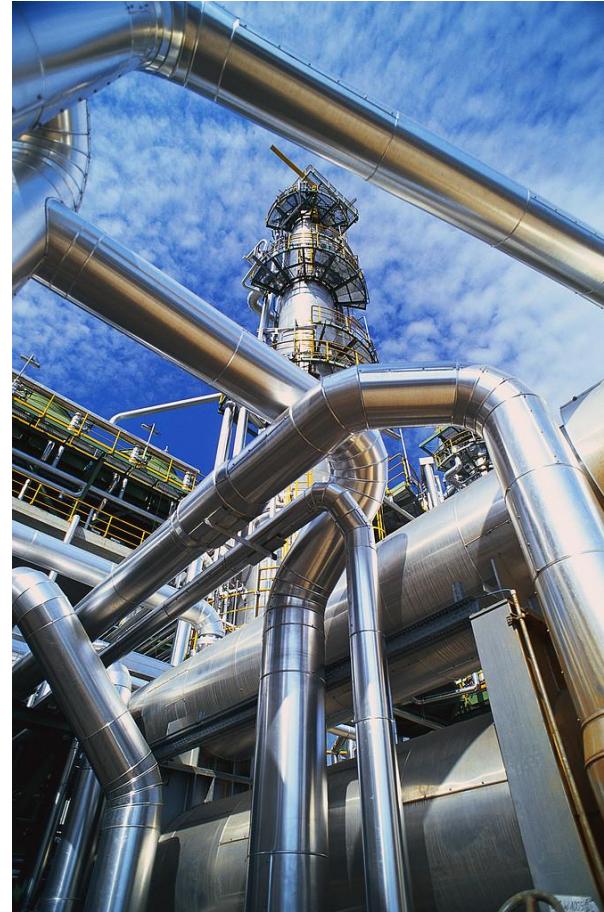


How do You Recognize a Human?



“nose”

Learn from Examples



Machine learning



Examples



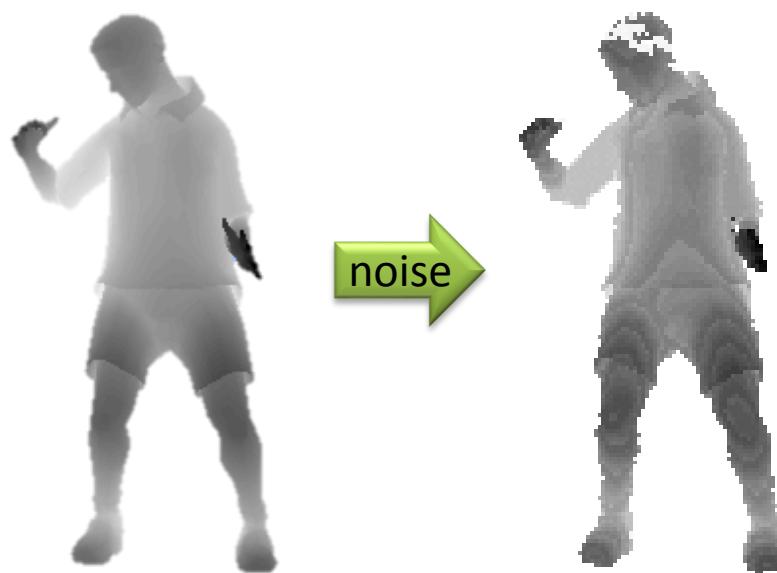
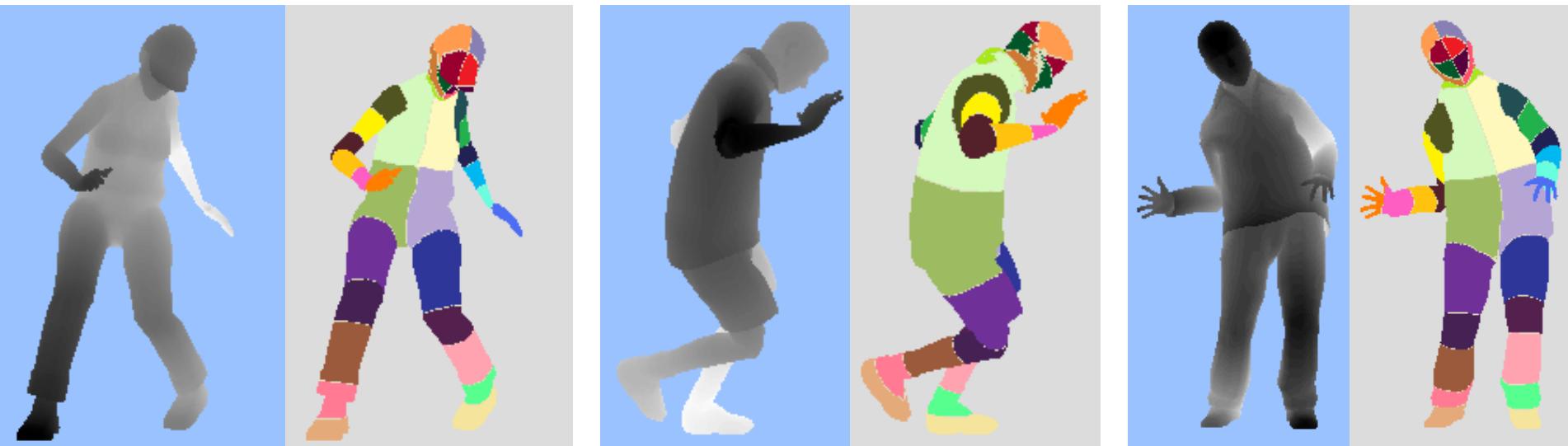
Motion Capture (Mocap)



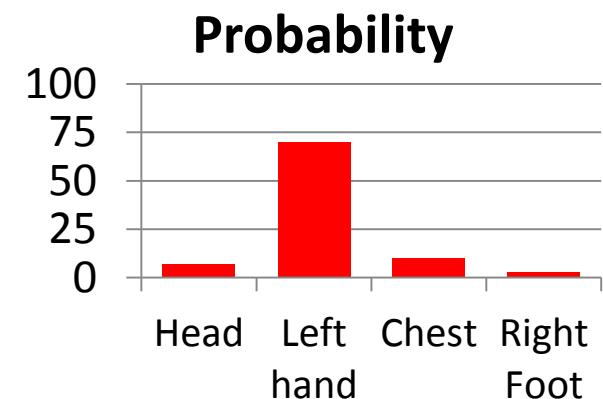
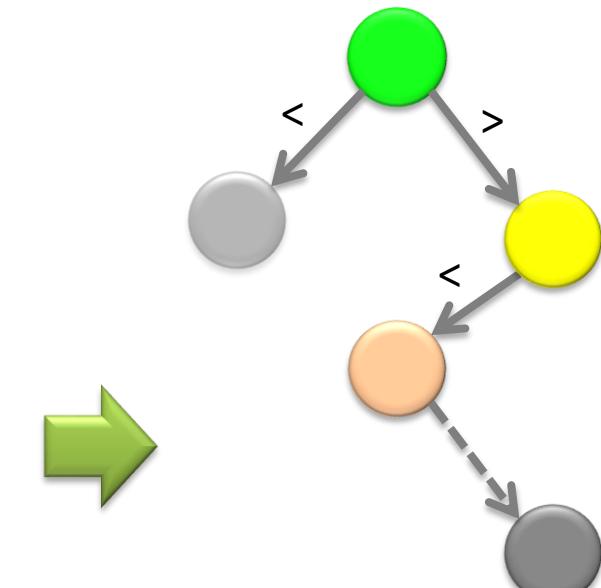
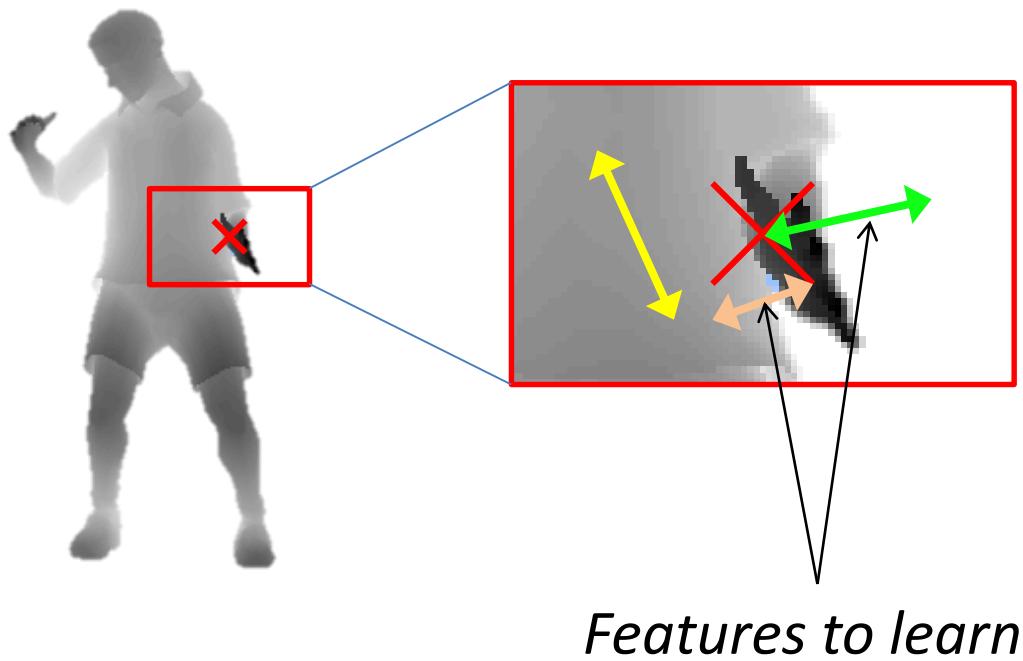
Rendered Avatar Models



Ground Truth Data

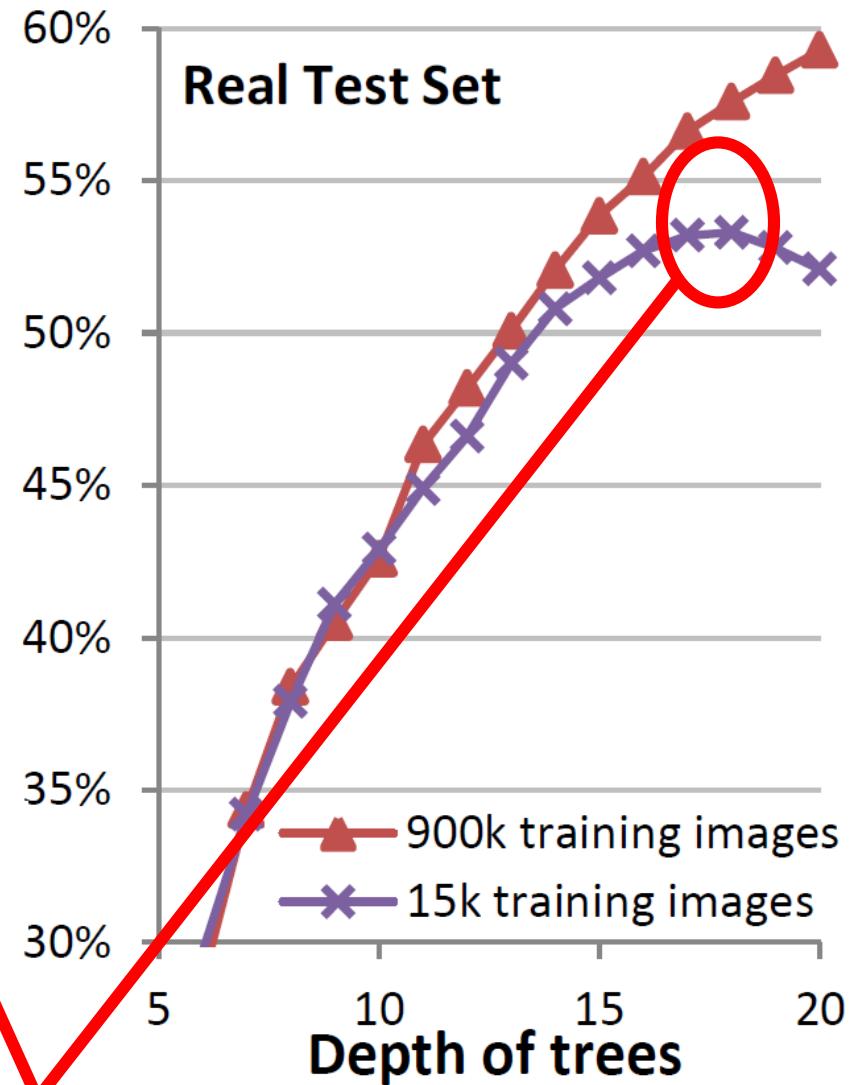
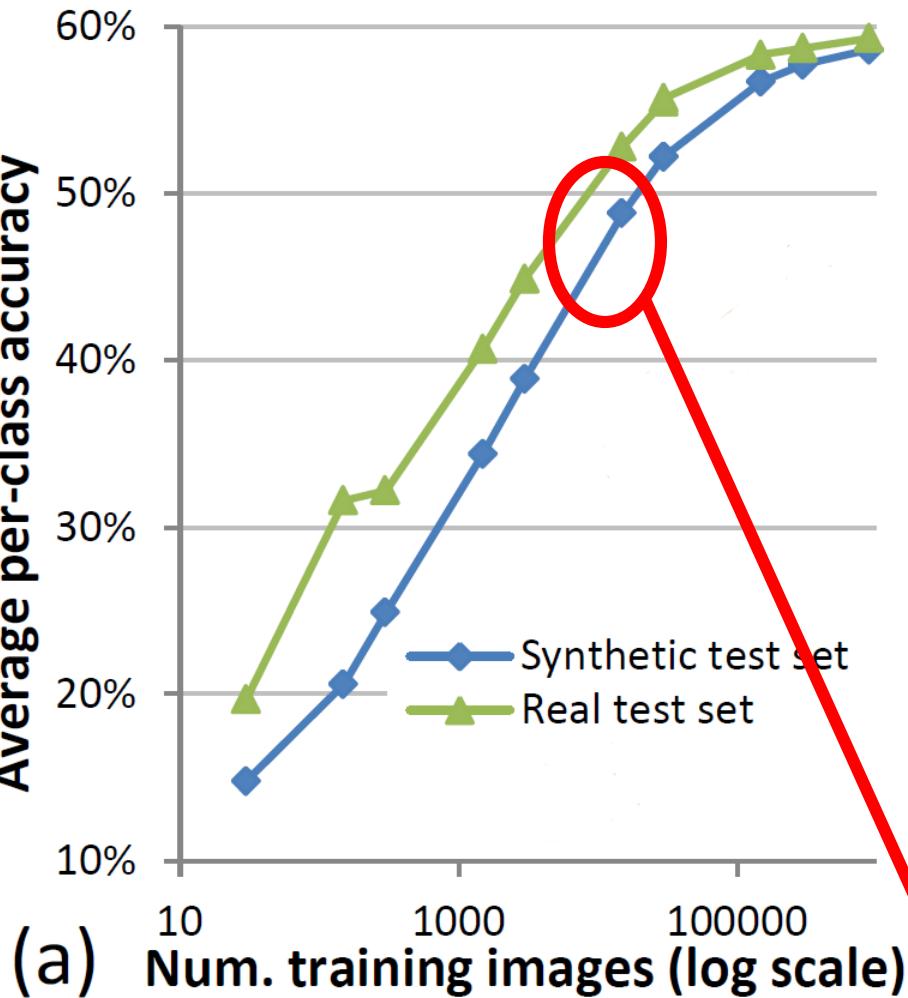


Decision Trees



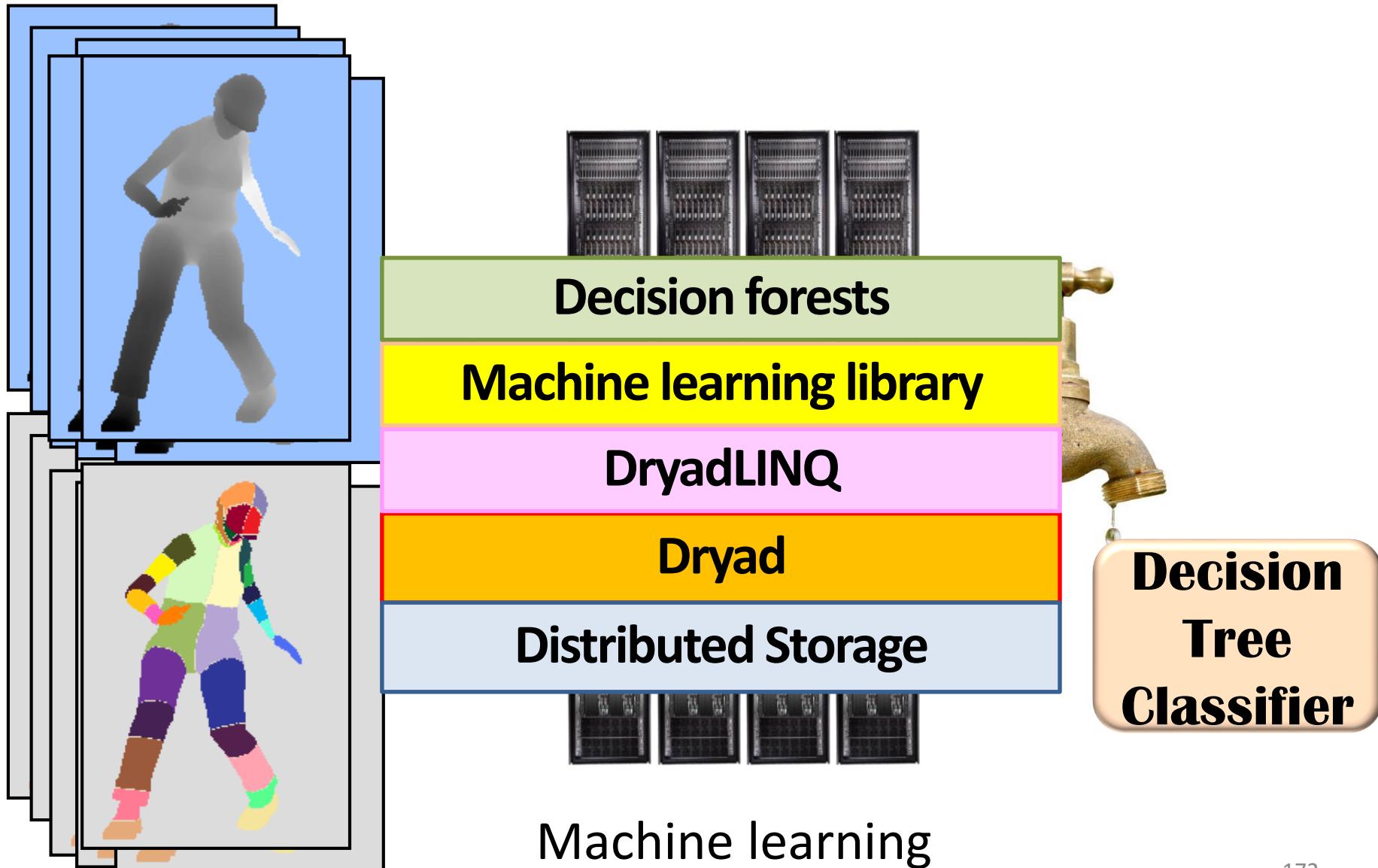
Accuracy

Average per-class accuracy

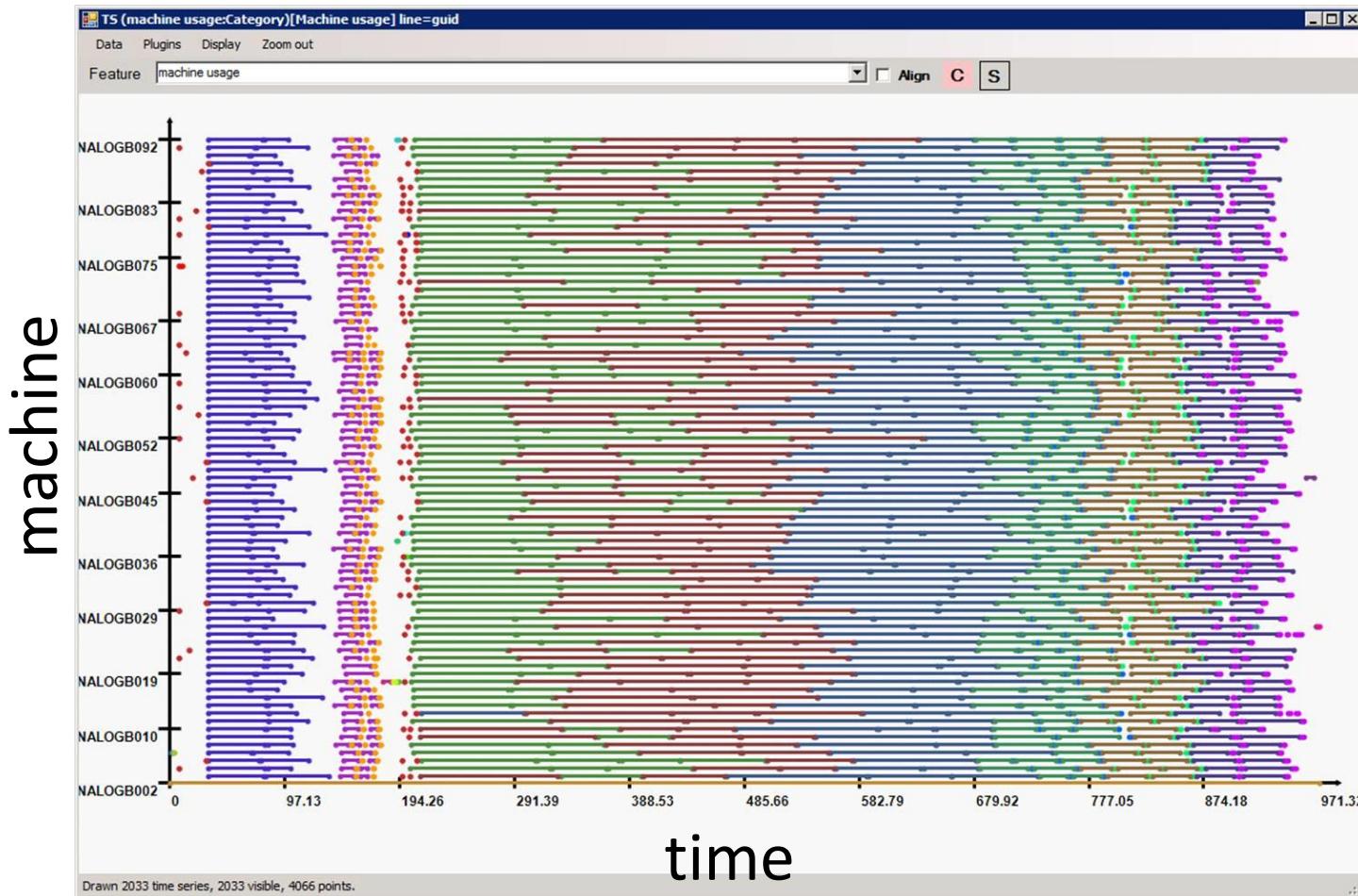


24 hours/8 core machine

Learn from Many Examples



Highly efficient parallelization



THE END

BASIC

The '60s

Spacewars



PDP/8

A Manual for BASIC, the elementary algebraic language designed for use with the Dartmouth Time Sharing System.

John G. Kemeny
Thomas E. Kurtz

This third edition prepared
the assistance of William C.
Gall Hannigan, Prof. Willie
Steinick.

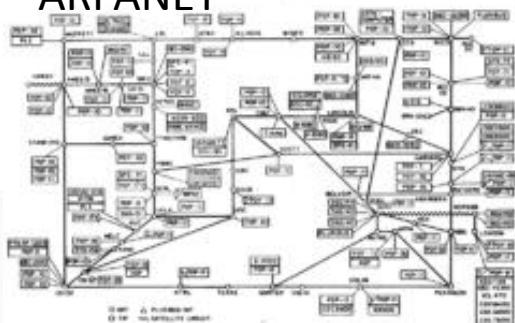
1 January 1966

(c)

Copyright 1966 by the Trustees
of Dartmouth College. Reproduction
non-commercial use is permitted
due credit is given to Dartmouth



ARPANET LOGICAL MAP, MARCH 1972



Artificial Intelligence Group

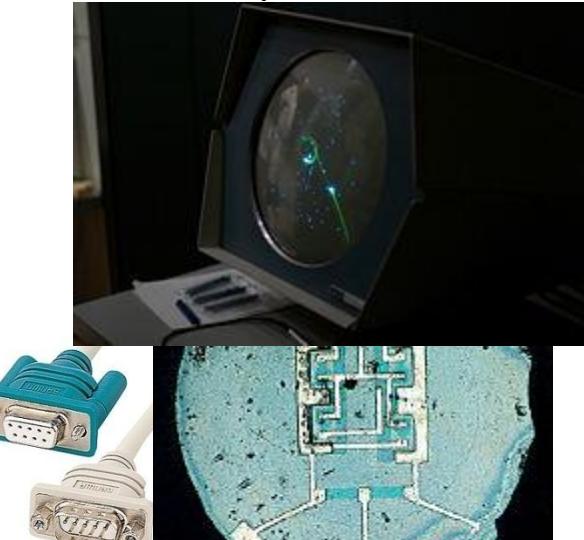
J. McCarthy
B. Brayton
D. Edwards
P. Fox
L. Hodas
D. Luckham
K. Malling
D. Park
S. Russell

```
(defun factorial (n)
  (if (<= n 1) 1
    (* n (factorial (- n 1)))))
```

COMPUTATION CENTER
RESEARCH LABORATORY OF ELECTRONICS
Massachusetts Institute of Technology
Cambridge, Massachusetts



OS/360

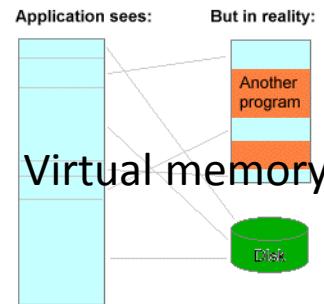


Multics

Time-sharing

A	1	0	0	0	0	0	1
B	1	0	0	0	0	1	0
C	1	0	0	0	1	1	0
D	1	0	0	1	0	0	0
E	1	0	0	0	1	0	1
F	1	0	0	1	0	1	0
G	1	0	0	1	1	0	0
H	1	0	0	1	0	0	0
I	1	0	0	1	0	0	1
J	1	0	0	1	0	1	0
K	1	0	0	1	0	1	1
L	1	0	0	1	1	0	0
M	1	0	0	1	1	0	1
N	1	0	0	1	1	1	0
O	1	0	0	1	1	1	1
P	1	0	1	0	0	0	0
Q	1	0	1	0	0	0	1
R	1	0	1	0	0	1	0
S	1	0	1	0	1	0	1
T	1	0	1	0	1	0	0
U	1	0	1	0	1	0	1
V	1	0	1	0	1	1	0
W	1	0	1	0	1	1	1
X	1	0	1	1	0	0	0
Y	1	0	1	1	0	0	1
Z	1	0	1	1	0	1	0

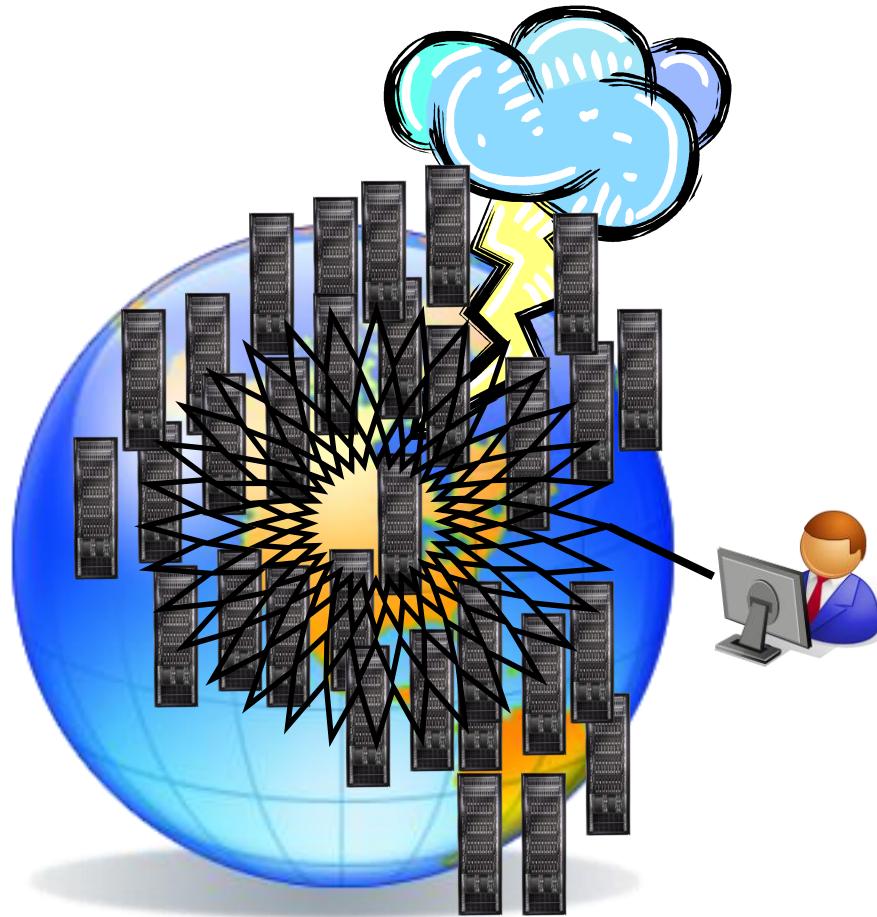
Application sees:



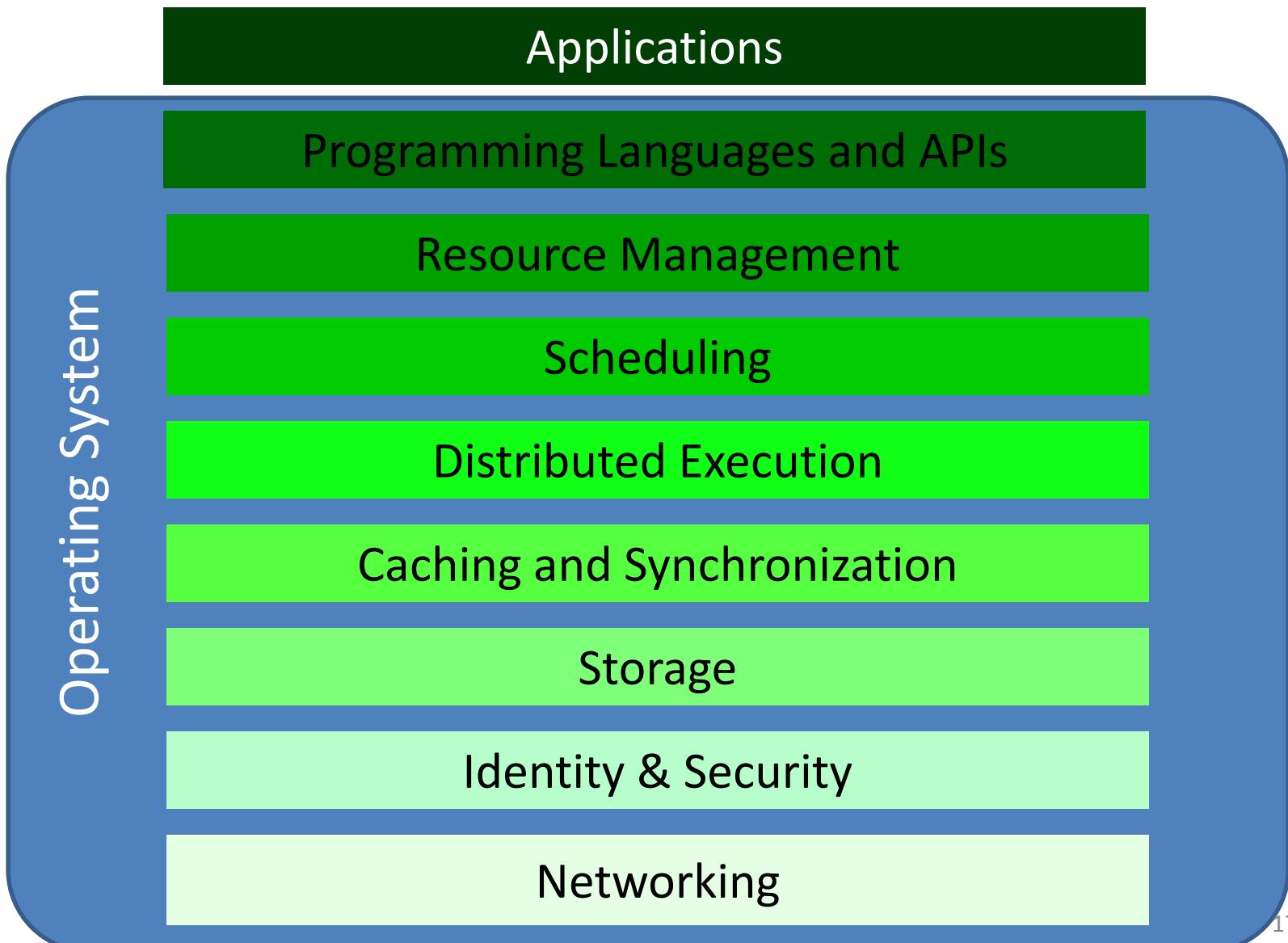
But in reality:



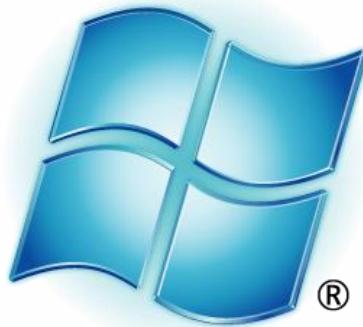
What about the 2010's?



Layers



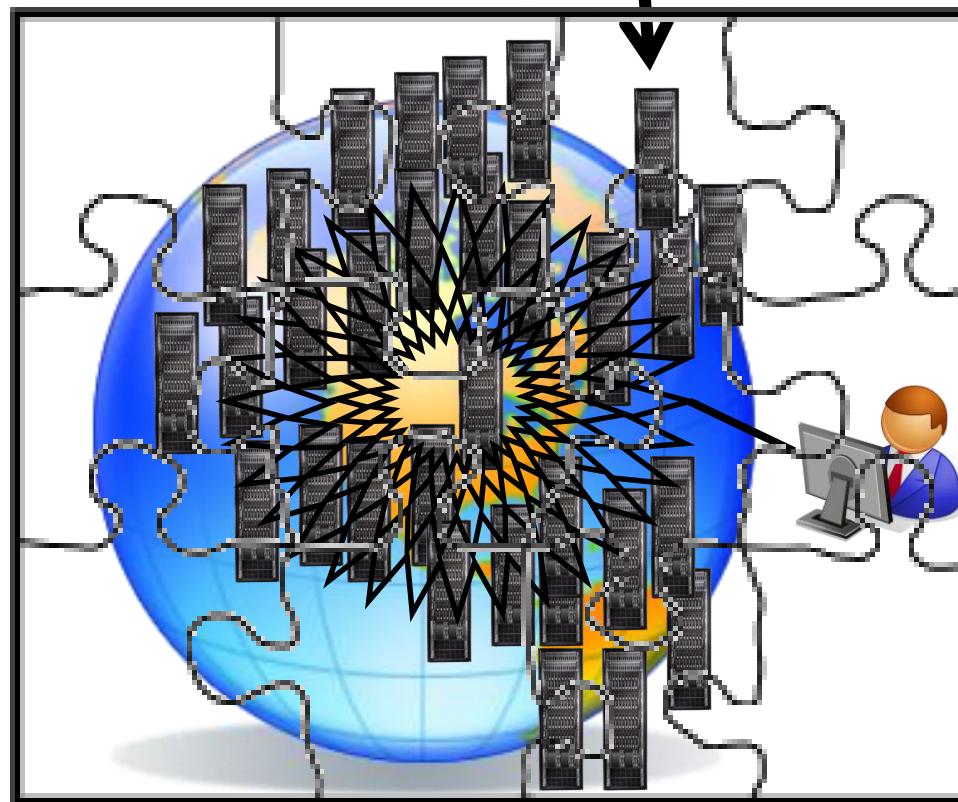
Pieces of the Global Computer



Яндекс

And many, many more...

This lecture



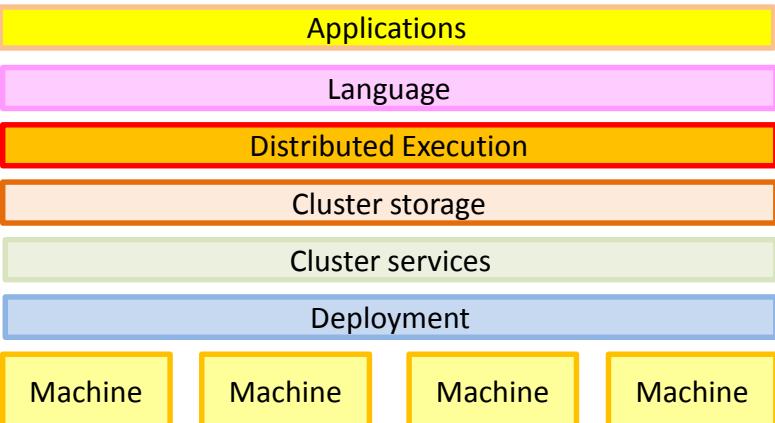
The cloud

- “The cloud” is evolving rapidly
- New frontiers are being conquered
- The face of computing will change forever
- There is still a lot to be done

You might do it!



Conclusion



Bibliography (1)

Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks

Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly

European Conference on Computer Systems (EuroSys), Lisbon, Portugal, March 21-23, 2007

DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language

Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey

Symposium on Operating System Design and Implementation (OSDI), San Diego, CA, December 8-10, 2008

Hunting for problems with Artemis

Gabriela F. Crețu-Ciocârlie, Mihai Budiu, and Moises Goldszmidt

USENIX Workshop on the Analysis of System Logs (WASL), San Diego, CA, December 7, 2008

DryadInc: Reusing work in large-scale computations

Lucian Popa, Mihai Budiu, Yuan Yu, and Michael Isard

Workshop on Hot Topics in Cloud Computing (HotCloud), San Diego, CA, June 15, 2009

Distributed Aggregation for Data-Parallel Computing: Interfaces and Implementations,

Yuan Yu, Pradeep Kumar Gunda, and Michael Isard,

ACM Symposium on Operating Systems Principles (SOSP), October 2009

Quincy: Fair Scheduling for Distributed Computing Clusters

Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg

ACM Symposium on Operating Systems Principles (SOSP), October 2009

Bibliography (2)

[**Autopilot: Automatic Data Center Management**](#), Michael Isard, in *Operating Systems Review*, vol. 41, no. 2, pp. 60-67, April 2007

[**Distributed Data-Parallel Computing Using a High-Level Programming Language**](#), Michael Isard and Yuan Yu, in *International Conference on Management of Data (SIGMOD)*, July 2009

[**SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets**](#)

Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou, Very Large Databases Conference (VLDB), Auckland, New Zealand, August 23-28 2008

[**Incorporating Partitioning and Parallel Plans into the SCOPE Optimizer**](#), Jingren Zhou, Per-Åke Larson, and Ronnie Chaiken, in Proc. of the 2010 ICDE Conference (ICDE'10).

[**Nectar: Automatic Management of Data and Computation in Datacenters**](#), Pradeep Kumar Gunda, Lenin Ravindranath, Chandramohan A. Thekkath, Yuan Yu, and Li Zhuang, in *Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI)*, October 2010

[**Optimus: A Dynamic Rewriting Framework for Execution Plans of Data-Parallel Computations**](#), Qifa Ke, Michael Isard, Yuan Yu, Proceedings of EuroSys 2013, April 2013