

# Data, Problems, and Languages

## ApacheCon 2010



## Abstract

*Much research work over the next decade will be driven by those seeking to solve complex problems employing the cloud, multicore processors, distributed data, business analytics, and mobile computing.*

*In this talk I'll discuss some past approaches but also look at work being done in the labs on languages like X10 that extend the value of Java through parallelism, technologies that drive cross-stack interoperability, and approaches to handling and analyzing both structured and unstructured data.*

The content of this presentation is my own and does not necessarily represent my employer's positions, strategies or opinions.

## Rules of conversation

- For any programming language I mention that does something, there will be 10 that do something similar that I won't mention, including your favorites.
- For any feature in any programming language, someone else has done it before, probably.
- No syntax wars today, nor arguments about weakly vs. strongly typed languages, braces, indentation, or tabs.
- IDEs and tools like debuggers are critical to the success of languages and thus to solving problems, but I don't address them here. (Someone has probably done something in Emacs or Eclipse, anyway.)

## Some personal history

- Started programming in 1973 at the age of 15 using BASIC on a time-sharing system.
- Problem solved with this was education.
- Next moved on to APL, an interactive, non-compiled language optimized for manipulating arrays with an unusual and interesting notation going back to 1957.
- I “misused” this to create a text processor for our high school newspaper.
- That is, I used a language for processing data outside of the usual application areas, though I didn’t have a lot of choice.



*Photo courtesy of Wikipedia*

## IBM Research, 1980s-90s: Axiom / A# / Aldor

- Core research was with AXIOM, a computer algebra system.
- These systems manipulate equations and formulas symbolically, not just numerically.
- Example computations involve integration, expression simplification, differential equations, matrices, and so on.
- Much of the research around them is in algorithms and the software needed to implement them efficiently.
- The better known systems today include Maple and Mathematica though some open source systems like Sage are available.

## Categories and parametric polymorphism

- Category- and domain-producing functions use the same language as first-order functions.

```
-- A function returning an integer.
factorial(n: Integer): Integer == {
    if n = 0 then 1 else n*factorial(n-1)
}

-- Functions returning a category and a domain.
Module(R: Ring): Category == Ring with {
    *: (R, %) -> %
}

Complex(R: Ring): Module(R) with {
    complex: (% , %) -> R;
    real: % -> R;
    imag: % -> R;
    conjugate: % -> %; ...
} == add {
    Rep == Record(real: R, imag: R);
    real(z:%): R == rep(z).real;
    (w: %) + (z: %): % == ...
}
```

## Conditional types

- Type producing expressions may be conditional.

```
UnivariatePolynomial(R: Ring): Module(R) with {  
  coeff: (% , Integer) -> R;  
  monomial: (R, Integer) -> %;  
  if R has Field then EuclideanDomain;  
  ...  
} == add {  
  ...  
}
```

- <http://www.algor.org/>
- A major problem was how to make this strongly typed library usable in a weakly typed interface that allowed easy calculations.
- <http://axiom-developer.org/axiom-website/bookvol0.pdf>

## Why special programming language work for math?

- By looking at math you have
  - Rich relationships among non-trivial concepts, with complex interfaces
  - A well-defined domain
  - Many programming language problems that had early use here: algebraic expressions, arrays, big integers, garbage collection, pattern matching, parametric polymorphism, ...
  - Large libraries requiring efficient code
- Simple programming interfaces quickly become insufficient.
- In the case of Axiom, the library was built on ideas borrowed from mathematical category theory, and so things like **AbelianMonoid**, **Ring**, **Field**, and **Module(R)** become part of the implementation.

## Problem areas driving programming evolution today

- Big Data, including streams
- Multicore and concurrent processing
- Cloud computing
- Simplification
- Cost and efficiency
- All the above, at the same time

# Data is getting big ... and wide, and tall, and deep



1.3B RFID tags in 2005  
30B RFID tags by 2010



WW Information volume  
doubling every two years



Capital market data volumes  
grew 1,750% 2003-06



2 Billion internet users  
by 2011



3.3B mobile phone users  
in 2007, 4.6B  
subscriptions in 2009

## Elements of a programming model

- Programming language
  - Safety, portability, productivity
  
- Tools
  - Compiler, debugger, test and performance, IDE
  
- Runtime
  - Performance, security, reliability, monitoring, management
  
- Ecosystem
  - Skills, support, documentation, libraries, ISVs

## Elements of a **parallel** programming model

- Programming language
  - Safety, portability, productivity
  - **Concurrency, synchronization, memory model**
- Tools
  - Compiler, debugger, test and performance, IDE
  - **Parallelization, multi-threaded analysis**
- Runtime
  - Performance, security, reliability, monitoring, management
  - **Scalability, OS interaction, races, deadlocks**
- Ecosystem
  - Skills, support, documentation, libraries, ISVs
  - **Parallel algorithms**

# Emerging parallel programming models

## Task-oriented

Richer, safer  
multi-thread programming

*Java fork-join, Intel TBB,  
Microsoft TPL, Cilk*

## Actor-oriented

Message passing between  
processes or active objects

*Erlang, Scala Actors,  
ActorFoundry, Kilim,  
Microsoft Axum*

## Hybrid data parallel

Data parallelism across  
multiple processor types  
(esp. GPU)

*Nvidia CUDA, OpenCL,  
Microsoft DirectCompute,  
Intel Ct/RapidMind*

## Cluster data parallel

Massive parallel processing  
of (mostly unstructured) data

*Hadoop (+ Pig/Hive/JAQL),  
StreamSQL, Streams SPL*

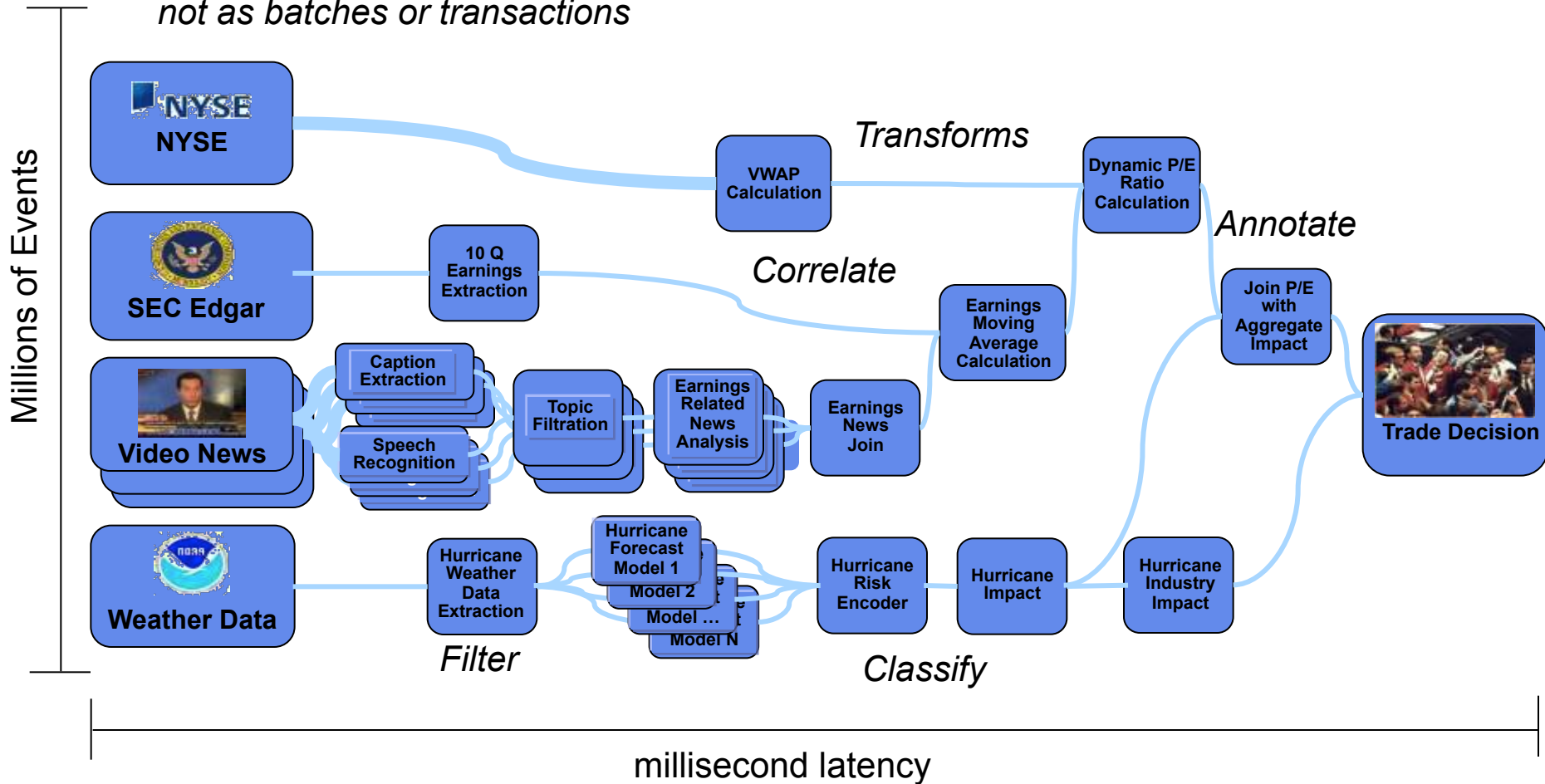
## Partitioned Global Address Space (PGAS)

Shared memory model  
for large scale clusters

*Unified Parallel C,  
Co-Array Fortran,  
X10*

# System S and streaming computing

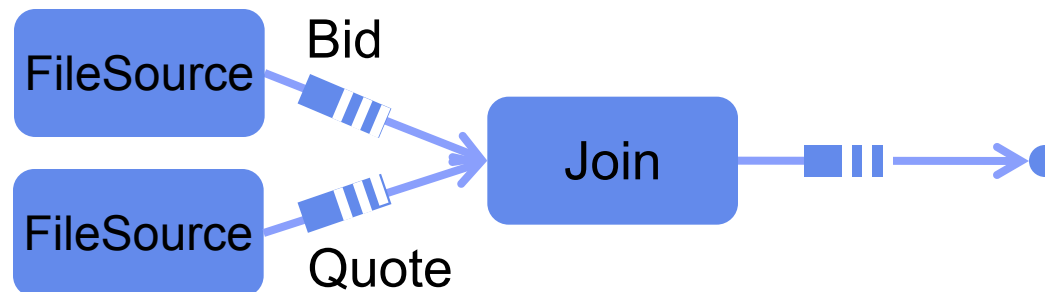
*Data processed as it arrives,  
not as batches or transactions*



[http://www-01.ibm.com/software/sw-library/en\\_US/detail/R924335M43279V91.html](http://www-01.ibm.com/software/sw-library/en_US/detail/R924335M43279V91.html)

## Example of SPL

```
stream<string8 buyer, string8 item, decimal64 price> Bid = FileSource() {
  param  fileName : "BidSource.dat";
}
stream<string8 seller, string8 item, decimal64 price> Quote = FileSource() {
  param  fileName : "SaleSource.dat";
}
stream<string8 buyer, string8 seller, string8 item> Sale = Join(Bid; Quote) {
  window Bid      : sliding, time(30);
        Quote     : sliding, count(50);
  param  match    : Bid.item == Quote.item && Bid.price >= Quote.price;
  output Sale     : item = Bid.item;
}
```



## Apache technologies for Big Data include ...

### ■ Hadoop MapReduce

- “Hadoop MapReduce is a programming model and software framework for writing applications that rapidly process vast amounts of data in parallel on large clusters of compute nodes.”
- <http://hadoop.apache.org/mapreduce/>

### ■ Pig

- “**Ease of programming.** It is trivial to achieve parallel execution of simple, ‘embarrassingly parallel’ data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain.”
- “Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist”
- <http://pig.apache.org/>



# X10

- Productivity goal
  - More convenient than Java: type inference, closures, syntactic slickness
  - More accurate than Java: constrained types, better generics
- Performance goal
  - Concurrency is in the language
  - Detailed model of multicore, accelerators, etc.
  - For sequential programming, structs and fewer required runtime checks
- Target areas of application are scientific computing and business analytics
- Two back ends
  - Java
  - C++: faster and required for many multi-core processors

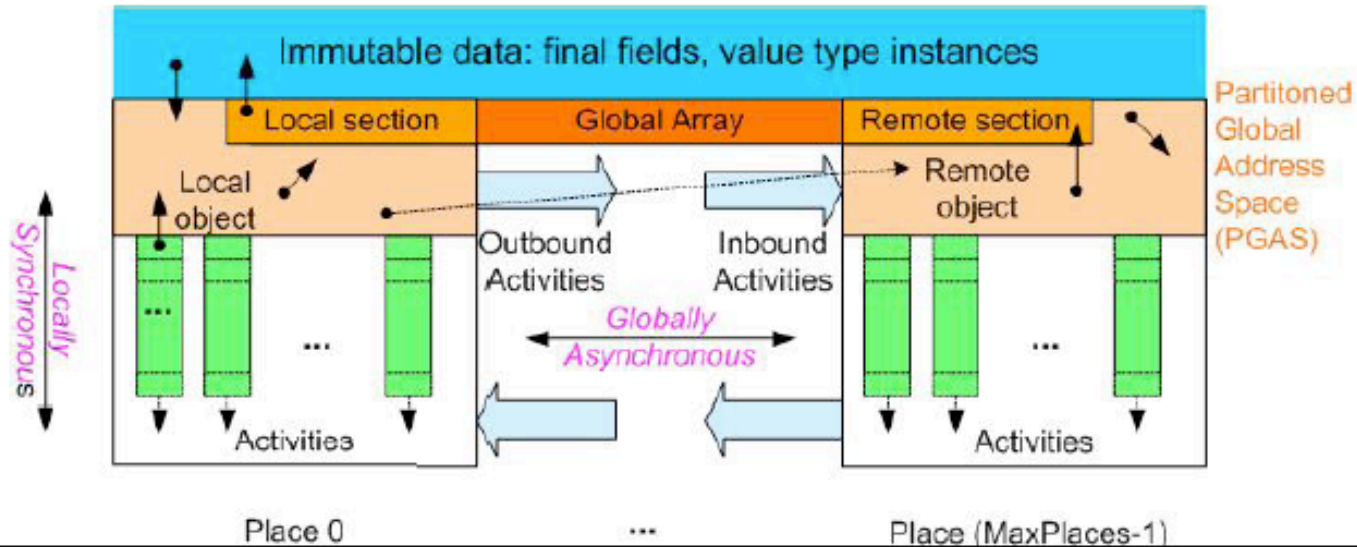
## X10 and concurrency

- Concurrency is hard
- X10 does not try to hide concurrency or the multiprocessor, or pretend that concurrency is just a minor extension.
- From the developers:

*“If you are not used to concurrency,  
X10 looks horrible and complicated.  
If you are used to concurrency,  
X10 looks pretty slick.”*

- <http://x10.codehaus.org/>

# Asynchronous Partitioned Global Address Space



<p>Fine grained concurrency</p> <ul style="list-style-type: none"> <li>• <i>async S</i></li> </ul>	<p>Atomicity</p> <ul style="list-style-type: none"> <li>• <i>atomic S</i></li> <li>• <i>when (c) S</i></li> </ul>	<p>Global data-structures</p> <ul style="list-style-type: none"> <li>• <i>points, regions, distributions, arrays</i></li> </ul>
<p>Place-shifting operations</p> <ul style="list-style-type: none"> <li>• <i>at (P) S</i></li> </ul>	<p>Ordering</p> <ul style="list-style-type: none"> <li>• <i>finish S</i></li> <li>• <i>clocks</i></li> </ul>	

## Simplification

- At this point the discussion often jumps right to scripting and dynamic languages.
- Simplicity is relative to other approaches in the same application domain.
- Simple on client frontend isn't the same as simple on the server backend.
- We need more information like “it's probably best not to try to solve that kind of problem in this language.”
- Things to avoid when simplifying:
  - Lack of interoperability
  - Bad scalability
  - Universal but ugly client-side user interfaces
  - Write-only syntax

## Scripting

- “Scripting is like fashion, it varies by the season and personal taste.”
- Scripting languages like JavaScript, PHP, Python, and Ruby are not going away anytime soon, in that order (?).
- Many of the people to whom I’ve spoken predict great things for the future of JavaScript, beyond its near ubiquity and despite its “bad parts.”
- Google’s V8 JavaScript engine should lead to the language getting embedded in more programming languages.
- Would WoW use JavaScript today if Blizzard started over?



## Pay by the bite (byte?)

### Sample cloud computing costs

Instance	Linux Usage
Small	\$0.085 per hour
Large	\$0.34 per hour

### Future computing costs?

Instance	Linux Usage
Object creation	\$? per instance
Loop (small)	\$? per instance
List reduction	\$? per instance
Hash table lookup	\$? per instance

## Customers will want to measure other things

- Customers will want to pay by the task accomplished, not by the programming features or operations used.
- So programming languages and execution environments, both local and distributed, will get measured by how efficiently they use processor, memory, and disk resources as well as how quickly and correctly they produce their results.
- Time-to-execution will also be critical: a programming language that allows me to code something in one line in one minute might be favored over the one that requires ten lines done in one hour.
- Languages that are unnecessarily resource hogs will get marginalized.
- How do you answer: how much energy is used in computing that result?

## Conclusions and summary

- Add features to solve problems, not just add features.
- *Programming Language Design: A Problem Solving Approach*
- One language does not fit all and never will.
- People are terrified of being stuck with huge libraries of source code written in now abandoned languages.
- Java is not going away and will remain critical for enterprises for many decades to come.
- “It’s the JVM, stupid.”
- Programming languages will be divided between languages that compile down to Java bytecodes and those that don’t.
- We’ll see a lot more languages in the future.

## Thanks to ...

- Colleagues who have shared their time and wisdom with me on these topics.
- They include John Duimovich, Sam Ruby, Brent Hailpern, David Boloker, Bob Blainey, Stephen Watt, Vijay Saraswat, David Ungar, Tessa Lau, Rodric Rabbah, John Field, Martin Hirzel, and members of the IBM Research staff.
- I thank them for their conversations and sharing material with me, much of which I have liberally borrowed.