

# Disruptive Programming Language Technologies

Todd A. Proebsting  
Microsoft Research

November 9, 2002

# Richard Hamming's Snare

---

- Richard Hamming's three questions for new hires at Bell Labs:
  1. "What are you working on?"
  2. "What's the most important open problem in your area?"
  3. "Why aren't they the same?" (Ouch!)

"You and Your Research" --- Richard Hamming (1986)

# The Least Important Open Problem in Programming Languages\*

---

Increasing program performance via compiler optimization

- Moore's Law suffices
- Algorithms and design make the big difference
- **Challenge:** Name a single significant software product that relied on compiler optimization for viability.

\* The opinions expressed here are mine and mine alone. Microsoft disavows any connection to them...

# The Most Important Open Problem In Programming Languages\*

---

## Increasing Programmer Productivity

- ◆ Write programs correctly
- ◆ Write programs quickly
- ◆ Write programs easily

### ■ Why?

- ◆ Decreases support cost
- ◆ Decreases development cost
- ◆ Decreases time to market
- ◆ Increases satisfaction

\*Standard disclaimer.

# Language Choice Affects Productivity

---

- The center of the programmer's universe!
  - ◆ Core abstractions, mechanisms, services, guarantees
  - ◆ Affect how programmers approach a task (*C vs. LISP*)
  - ◆ Assumptions, expectations, patterns
    - types
    - events
    - immutable data
    - garbage collection
    - regular expressions
    - first-class functions, closures
    - ...

# Language Design: C vs. LISP

---

- What's the difference between a C programmer and a LISP programmer?
  - ◆ A LISP programmer knows the value of everything and the cost of nothing.
  - ◆ A C programmer knows the cost of everything and the value of nothing.  
E.g., garbage collection, first-class functions, safety...
- The languages encourage this thinking:  
`(map fn L) vs. while (*d++ = *s++);`
- Some "value investors" are reaping strong returns nowadays. ([www.paulgraham.com](http://www.paulgraham.com))

# Programming Language Technologies: Recent Research vs. Progress(!)

---

- Recent (perpetual?) academic research:
  - ◆ Type theory
  - ◆ Functional programming
  - ◆ Object-oriented programming
  - ◆ Parallel programming
  - ◆ Static analysis
  - ◆ Compiler optimization
- Recent adoption: Perl, Python, Visual Basic, Java
  - ◆ Almost void of innovation on type theory, functional programming, OO programming, optimization, etc!
  - ◆ Perversely hopeful development for new language design efforts.

# *The Innovator's Dilemma (C. Christensen)*

---

## languages

- "... why ~~companies~~ that did everything right---were in tune with their competition, listened to their customers, and invested aggressively in new technologies---still lost their market leadership when confronted with disruptive changes in technology..."

--- the book's back cover

- Why is C/C++ losing steam? 😊
  - ◆ Can we use the book's lessons to help future language efforts? (Not the book's intent...)

# The Innovator's Dilemma: Cable-Actuated Excavators



- A “disruptive” technology
    - ◆ Disadvantage in primary market
    - ◆ Advantage in secondary market
    - ◆ Sold in small, low-margin market
  - Established companies concentrate and innovate on primary market; ignore secondary
  - Timely improvements lessen disruptive technology's liabilities, increasing markets, market share, margins, etc.
- hydraulic mechanisms  
small, unreliable  
safe, attaches to tractor  
independent contractors
- capacity (for excavation)



# The Innovator's Dilemma: C

---

- A "disruptive" language
  - ◆ Disadvantage safe, GC'ed interpreters  
SLOW
  - ◆ Advantage Rapid Application Develop
  - ◆ Sold in small, low-margin market web developers, ISV's  
(established competitor ignored market)
- Established companies concentrate on primary differentiator SPEED
- Timely improvements lessen disruptive technology's liabilities, increasing markets, market share, margins, etc.  
Moore's Law (for free!)  
RAD enhancements

# Distinguishing/Disruptive Technologies: Alleviating Real Problems

---

- Perl
  - ◆ Scripting with data structures (“duct tape”)
  - ◆ Regular expressions
- Visual Basic
  - ◆ Drag-and-drop environment (Windows for the masses)
  - ◆ Component-friendly
- Java
  - ◆ Browser applets

Languages yield pervasive patterns and abstractions

# An Opportunity!

---

- Languages (or language technologies) that solve real problems can succeed
  - ◆ Even if slow
  - ◆ Even with simple types
  - ◆ Even without academic significance
  - ◆ Even without rocket science
  - ◆ **If useful**
- Researchers need not despair
  - ◆ Golden opportunity to use disruptive technology as a Trojan Horse for disseminating research ideas

# Future Disruptive Language Technologies (My Recurring Wish List)

---

- My criteria: technology must
  - ◆ Have disadvantages
  - ◆ Be mostly ignored by recent PLDI and POPL conferences
  - ◆ Alleviate real problems...  
"What does it do?"
- For each candidate technology: 2 slides
  - ◆ Opportunity                      what's the issue?
  - ◆ Current solutions                what's done now
  - ◆ Proposal                            sketch of language solution
  - ◆ Disadvantages                    why some (many?) will scoff
  - ◆ Unmet needs                      benefits to adopters



# Candidate: Flight Data Recorders

---

- Opportunity: How do you debug a program that misbehaved after the error occurred?
  - ◆ Microsoft “Watson” experience
    - 50% of crashes caused by 1% of bugs.
- Current solutions
  - ◆ Ad hoc attempts to reproduce error condition
  - ◆ Examine stack trace, program state (“core dump”)

# Disruptive Flight Data Recorders

---

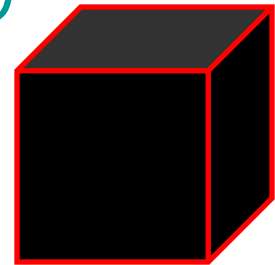
Add persistent, automatic “tracing” of function calls, events, I/O, etc. to the language run time.  
(E.g., AMOK/IDAL from IDA on CRAY-1)

- Important disadvantages

- ◆ Will slow every program down
- ◆ Will require storage

- Unmet needs

- ◆ Diagnostic data available to programmer --- 1/50 rule
- ◆ “Introspective” data available to program



# Candidate: Checkpoints/Undo

---

- Opportunity: Programs provide checkpoint or “undo” facilities in haphazard, unreliable ways. (E.g., MS Outlook, TurboTax, almost all tiny apps.)
- Current solutions:
  - ◆ Checkpoint by saving document to a file
    - Doesn't scale well to unbounded undo
  - ◆ Programmatic checkpoint by saving *select* data to file
    - Subject to judgment (and error)
  - ◆ Undo by saving operations and their inverse data
    - Tedious
    - Error-prone

# Disruptive Checkpoints/Undo

---

Make checkpointing and undo (i.e., restore to checkpoint) primitives in the programming language. Transactions.

- Important disadvantages
  - ◆ External side-effects pose limitations (e.g., I/O)
  - ◆ Slower than hand-crafted solution
- Unmet needs
  - ◆ Simplicity
  - ◆ Automation

```
checkpoint X;  
<random code>  
restore/commit X;
```

# Candidate: Parsing

---

- Opportunity: Parsing is common and difficult in general.
- Current solutions:
  - ◆ Parser generators for subsets of CFLs
  - ◆ Regular expressions ala Perl
  - ◆ Roll your own parser (and cross your fingers that nobody ever needs to maintain it)

# Disruptive Parsing

---

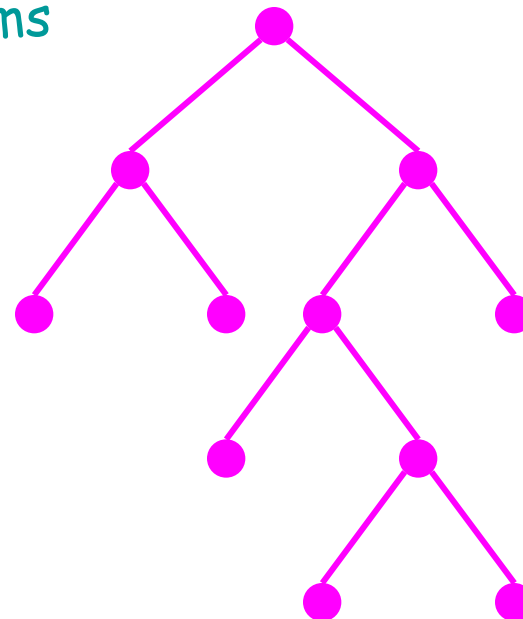
“Scannerless Generalized LR Parsing” (or Earley parsing) could be integrated into a language

- Important disadvantages
  - ◆ Slow
  - ◆ Ambiguity presents its own problems
- Unmet needs
  - ◆ Handle arbitrary CFL grammar
  - ◆ Spec-driven systems adapt smoothly to change
  - ◆ Confidence that parser meets spec
    - XML grammar has 80+ productions...

# Candidate: Constraint Solvers

---

- Opportunity: Many applications have a subproblem that involves solving (or optimizing) a system subject to constraints
  - ◆ Natural fit for visual layout problems (e.g., render tree structures, resize windows, summarize maps)
  - ◆ Natural fit for optimization problems
- Current solutions
  - ◆ Hand-rolled algorithms
  - ◆ Library routines
  - ◆ Third-party solvers
  - ◆ Give up

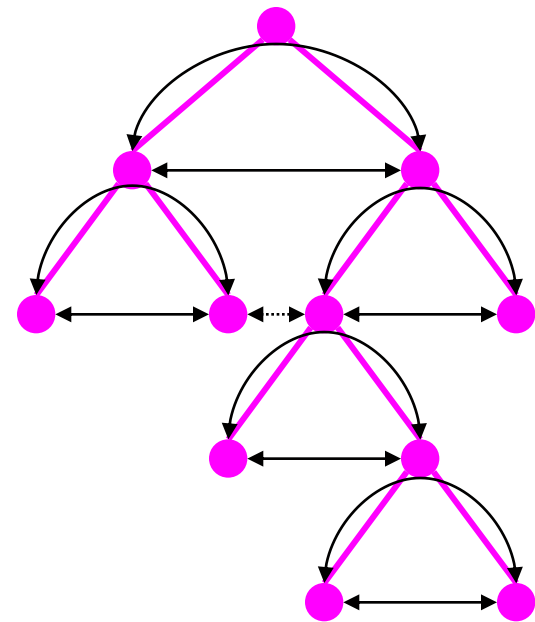


# Disruptive Constraint Solvers

---

Integrate linear programming constraint solver (or, better, integer programming) into a programming language

- Important disadvantages
  - ◆ Slower than tailored algorithmic solutions
- Unmet needs
  - ◆ Quick and dirty solutions
    - Visual layout (Interviews-Tk?)



# Candidate: Concurrent Programming

---

- Opportunity: Many applications are explicitly or implicitly concurrent or distributed
  - ◆ Concurrency models many applications better than "objects," yet the world is mired in OO religion.
- Common solutions
  - ◆ OS threads, shared data, P(), V()
  - ◆ Language threads, shared data, P(), V()
  - ◆ Remote procedure calls

# Disruptive Concurrent Programming

---

- Concurrent functional programming language (Erlang™?)
  - ◆ Lightweight processes (10,000's)
  - ◆ Message passing  
(non-blocking send, blocking receive with timeouts)
  - ◆ Higher-order functions w/ pattern-matching dispatch
  - ◆ Immutable data (except message queues)

- Important disadvantages
  - ◆ Immutable data can be slower to manipulate
  - ◆ Doesn't look like C++, not OO

- Unmet needs
  - ◆ Concurrency-Oriented Programming
    - Processes+Messages+Immutable data, which can be reasoned about

## Notable Omissions:

- Monads
- Continuations
- Lazy evaluation
- Complex type system

# A Final Prediction

---

- The next big programming language will be slower than what it replaces
- Why?
  - ◆ The incumbent language will have been optimized relentlessly
  - ◆ To replace it, the new language must offer something new that will be valuable even if slow.

# Shameless Self-Interest

---

- I manage the Programming Language Systems group in Microsoft Research
  - ◆ We work on programming language design and implementation
  - ◆ We appreciate small, simple solutions
  - ◆ We're a small group: Chris Fraser, Dave Hanson and me
  - ◆ We're recruiting! (Full-time researchers and interns)
- Email: [toddpro@microsoft.com](mailto:toddpro@microsoft.com)

# The End

---