



# Choose at Most Two: Performance, Portability, and Development Cost

**Dan Reed**

**NCSA and Computer Science**

**University of Illinois**

**reed@ncsa.uiuc.edu**



# A Morality Play In Two Parts

- **Sin and its evils**

- skimping on software investment
- failing to leverage/recognize economics
- failing to stay the course

**Moonshine Production**



- **Redemption**

- embrace scale and complexity
- recognize and exploit dynamic behavior
- leverage community activities
- exploit the market and government

# The High-Performance Conundrum

- **The conundrum**

- large NRE costs
  - particularly software
- modest size markets
  - TMC, KSR, ...
- NCSA's previous vendors
  - almost all RIP

How do we make a small fortune in HPC?



- **Implications**

- 1980s-1990s market shakeout
- limited base to amortize software costs

- **Rescue (Ben Franklin)?**

- open source software
- commodity components

# The Eternal Tension



DILBERT © United Feature Syndicate, Inc.  
Redistribution in whole or in part prohibited.

**For a successful technology, reality must take precedence over public relations, for nature cannot be fooled.**

*Richard Feynman*

# It's The Software, Duh!

- **We have this fascination with hardware**
  - it's tangible and obvious
  - it's necessary but not sufficient
    - like a car without a road
  - look at our experiences
    - CDC 6600, Cray 1, CM-5, KSR, ...
- **Software is almost always overlooked**
  - both application and infrastructure
  - application embodies the science
  - tools enable/hinder productivity
  - it's not cheap; must keep supporting it!
- ***Our leverage on the market***
  - *is asymptotically decreasing*



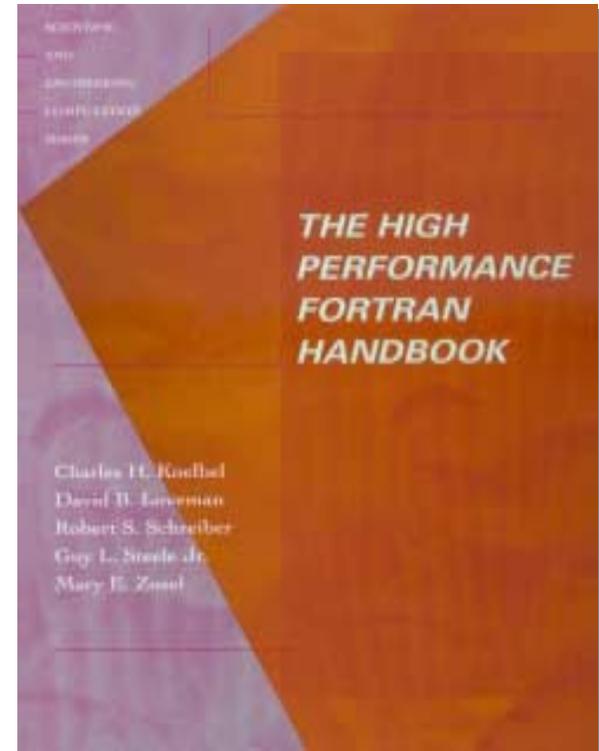
# HPF: I Feel Your Pain

- **Lessons**

- irregular data structures
  - language support needed
- data distributions
  - best not part of the language
- compilation and tuning
  - major research challenges
  - inverse mappings for tuning

- **Observations**

- HPF locality model is semi-implicit
- we expected too much too soon
  - 5-10 years needed for mature compilers and idiom identification
  - languages and idioms must evolve



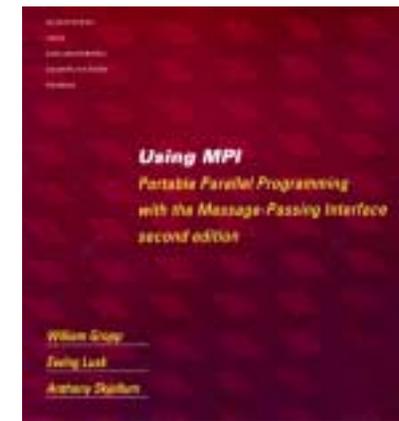
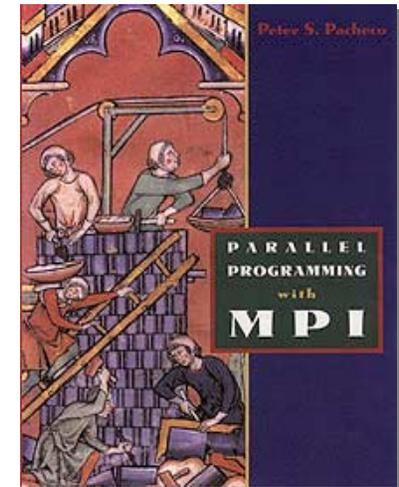
# MPI: It Hurts So Good

- **Observations**

- “assembly language” of parallel computing
- lowest common denominator
  - portable across architectures and systems
- upfront effort repaid by
  - system portability
  - explicit locality management
- remember what Churchill said about democracy
  - it applies to MPI as well (sorry Rusty *et al*)

- **Costs and implications**

- human productivity
  - low-level programming model
- software innovation
  - limited development of alternatives



# Performance Optimization

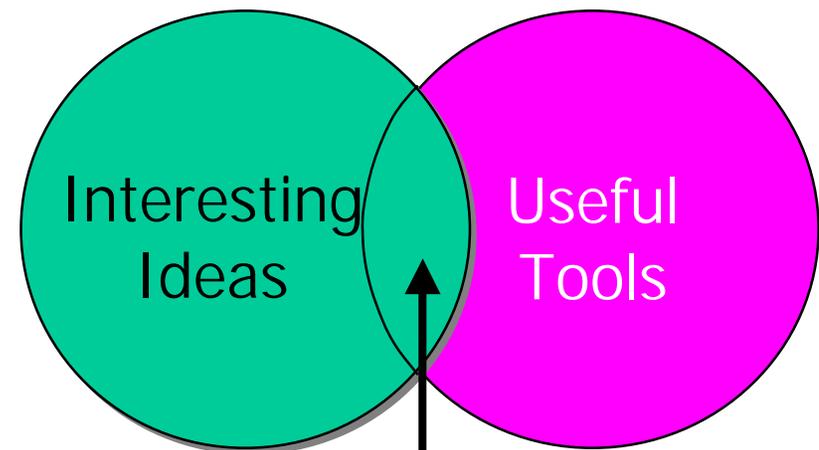
- **Experience and idioms**

- effective tuning exploits common usage idioms
- new systems are often unstable and evolving
  - applications and tools depend on many things
- experience takes time



- **Experience means**

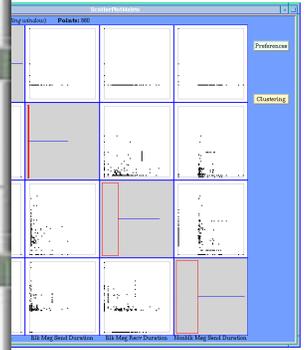
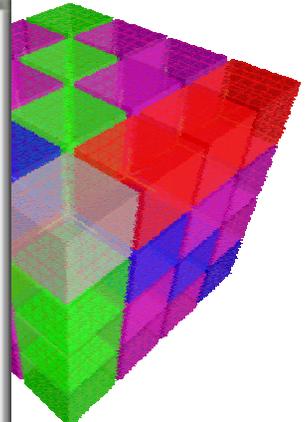
- *multiple system generations*
- *similar programming models*
- *user interaction and insights*



*Software Tool Research*

# Performance Tool Experiences

The screenshot shows the svPablo performance tool interface. The main window displays project information for 'MLC on A-32', source files, and performance contexts. A 'Performance Metric Select' dialog is open, showing options for Call Statistics, Loop Statistics, and HW Statistics by Line. A 'Specific Metric' dialog is also open, showing hardware statistics for 'Clock Cycles' and 'Floating Point Instructions'.



Instrumentation Points  
Parser Compiler  
Trans  
Instrumented Code

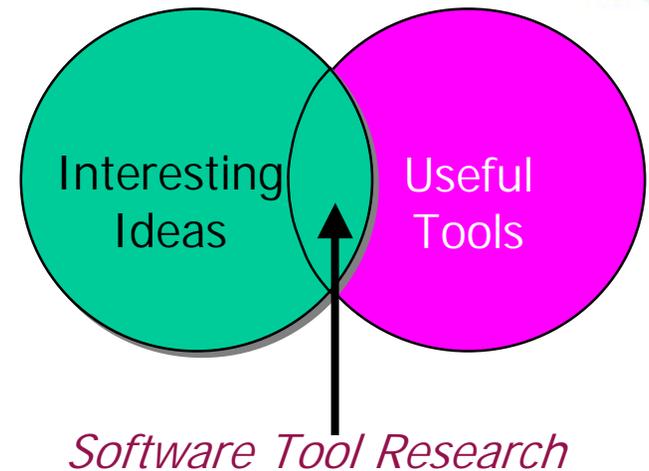
Correlation  
Drilldown

$$a(i,j) = 0.175 * (a(i-1,j) + a(i+1, j) + a(i,j-1) + a(i, j+1))$$



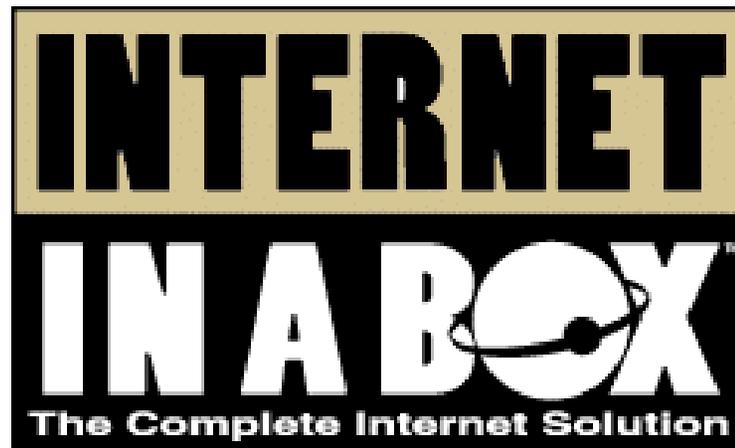
# Software: We Need A New Approach

- **It's time for a change**
  - complexity is rising dramatically
    - highly parallel and distributed systems
    - multidisciplinary applications
  - we need to get ahead of the curve
    - think about coming technologies
- **Deus ex machina software model**
  - it's not working effectively
    - it will scale even less effectively to petaflops
  - it may be time to adopt decentralized control
    - Internet folks realized this long ago
    - profound implications for software design
- **Embrace dynamic equilibrium**
  - dynamic adaptation and lossy behavior
  - failures are just really, really bad performance 😊



# A Modest/Crazy Proposal

- What is a 100,000 processor system?

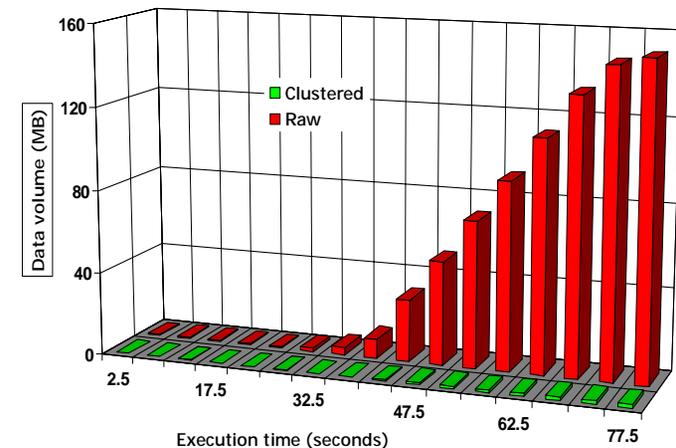


- **Statistics is your friend!**
- **Embrace failure, complexity, and scale**
  - a mind set change

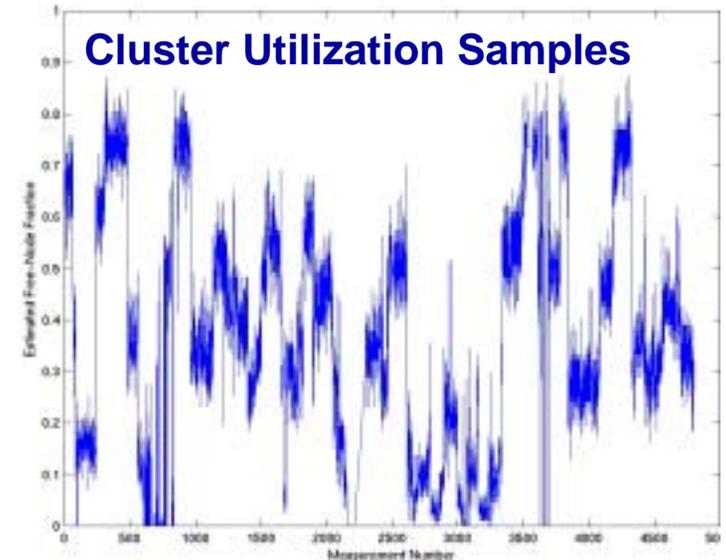
# The Large System Problem

- **Detailed measurements enable**
  - flexible post-mortem analysis
  - spatio-temporal correlations
- **But, they produce large data volumes**
  - exacerbated by trans-terascale systems
    - 10K-100K processors
- **Possible solutions**
  - statistical clustering (activity identification)
  - projection pursuit (metric identification)
  - population sampling (behavioral identification)
- **Population sampling (Linux clusters)**
  - 8% error bound and 90 percent confidence
  - 87 node minimum sample size
    - for 1024 processors; does not rise
  - 94 percent of cases within 8 percent

## Statistical Clustering



## Cluster Utilization Samples



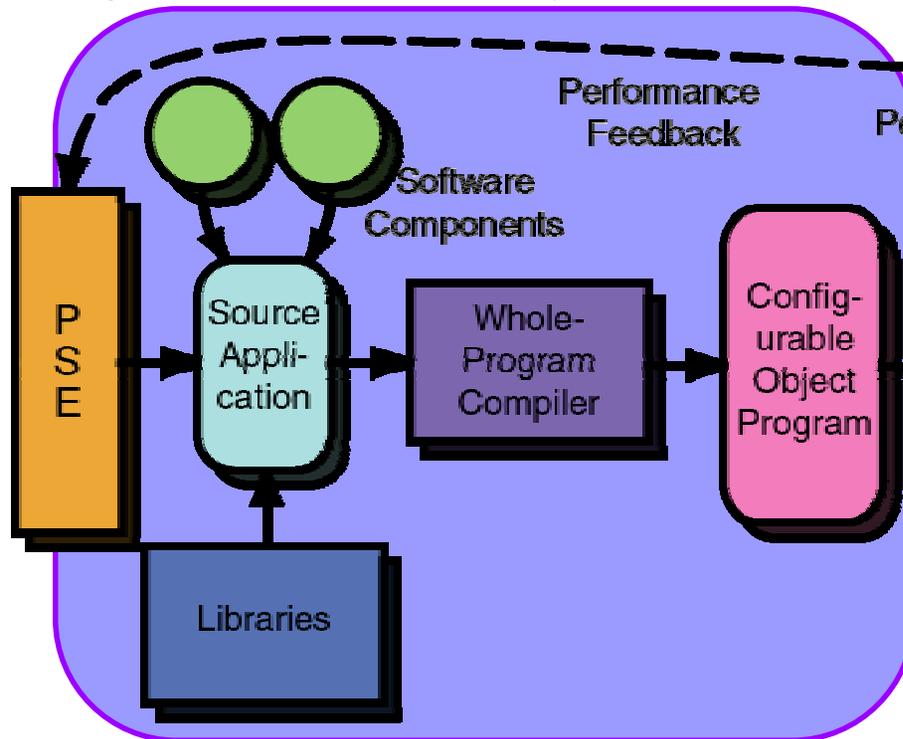
# Intelligent Software: An Analogy

- **50 MPH is a legal stricture with no ambiguity**
  - 51 MPH is a violation and you could be cited and fined
    - rarely are violators ticketed for such small violations
  - context determines actual behavior
    - city rush hour traffic rarely obeys speed limits
    - hazardous conditions change the effective speed limit
- **What really happens**
  - police use contextual discretion
    - “small” violations for “reasonable intervals” are tolerated
      - rush hour, weather, and special events
  - obeying the spirit of the law is usually the correct thing
    - perturbations about the limits are expected and accepted
  - if something happens, you want justice (or mercy), not legalisms
- **Intelligent, adaptive software is similar**
  - application needs and available resources determine behavior
  - *performance contracts must be flexible, with contextual discretion*

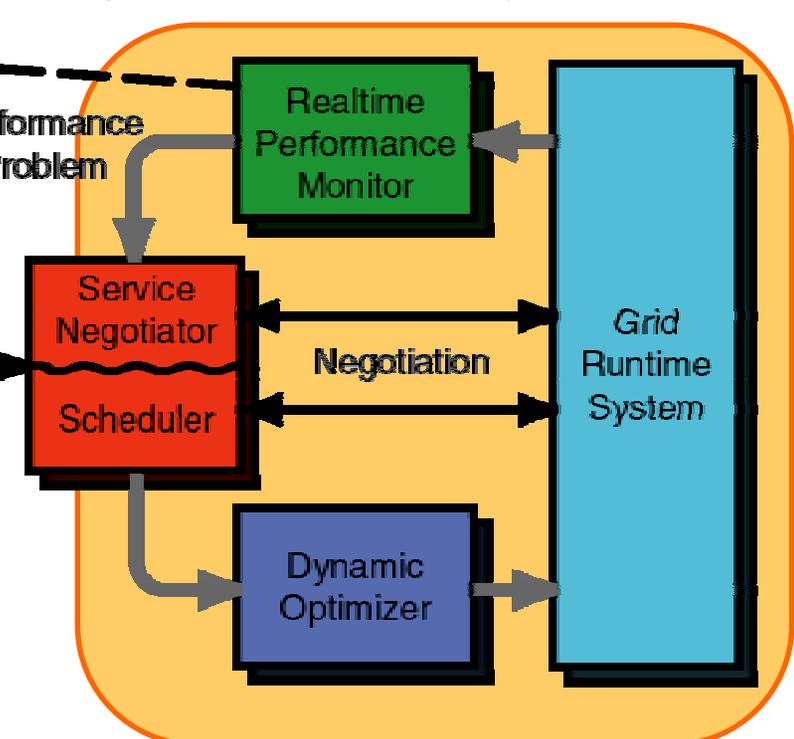


# GrADS: The “Big Picture”

Program Preparation System (PPS)



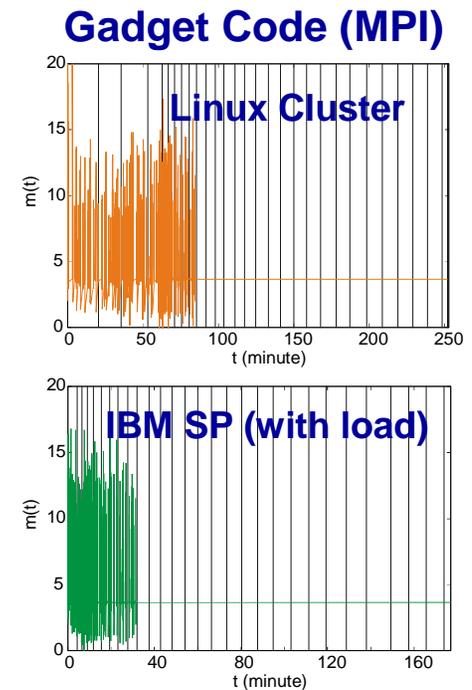
Program Execution System (PES)



GrADS participants: Andrew Chien, Fran Berman, Jack Dongarra, Ian Foster, Dennis Gannon, Ken Kennedy, Carl Kesselman, Lennart Johnsson, and Dan Reed

# Signatures and Contracts

- **Application intrinsic metrics (signatures)**
  - compact description of temporal application demands
  - values independent of execution platform
- **Execution space metrics (signatures)**
  - reflect application demands and resource response
- **A contract specifies that given**  
a set of **resources** (compute, network, I/O, ...)  
with certain **capabilities** (FLOP rate, latency, ...)  
for given **application parameters** (matrix size, ...)  
**the application will**  
**exhibit a specified, measurable, and desirable performance**
  - sustain F FLOPS/second, render R frames/second, ...
- **Performance contracts specify a convolution of**
  - application intrinsic behavior and system resource responses



# Software Signatures

- **Separate two aspects**

- application stimuli
- system resource responses

- **Polyline fitting**

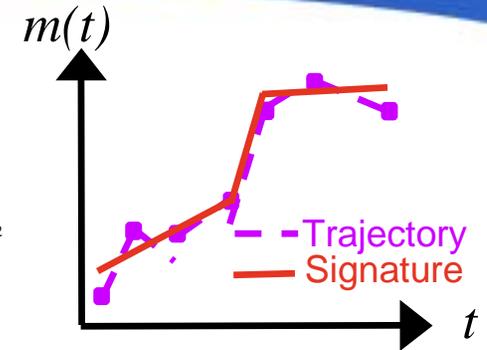
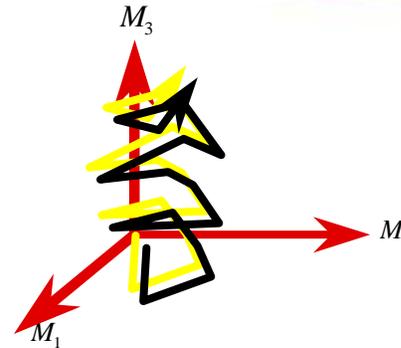
- based on least squares linear curve fitting
  - fit as many points as possible via one segment
    - continue until error exceeds some criteria
  - begin a new segment and repeat as needed
- curves are computed in real-time

- **Signature comparison**

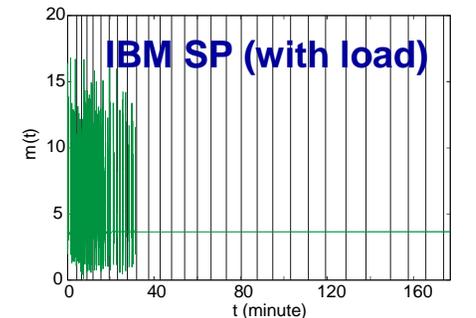
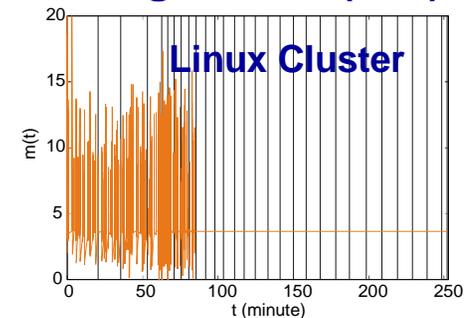
- degree of similarity (DoS) of  $q$  with respect to  $p$  is

$$\max\left(1 - \frac{\int |p(t) - q(t)| dt}{\int p(t) dt}, 0\right)$$

- higher DoS equivalent to greater similarity

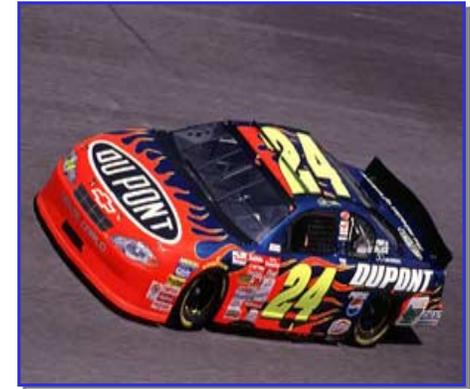


## Gadget Code (MPI)



# Choose At Most Two

- **High performance**
  - exploitation of system specific features
    - cache footprint, latency/bandwidth ratios, ...
  - *mitigates against portable code*
- **Portability**
  - targeting the lowest common denominator
    - standard hardware and software
      - *mitigates against high performance code*
- **Development cost**
  - cost shifting to hide human investment
    - people are the really expensive part
  - specialization to problem solution
  - *mitigates against portable, high-performance code*



Performance



Portability

**One Can Choose None!**

# How To Choose Wisely

- **Performance**

- runtime adaptation
- dynamic code generation

- **Portability**

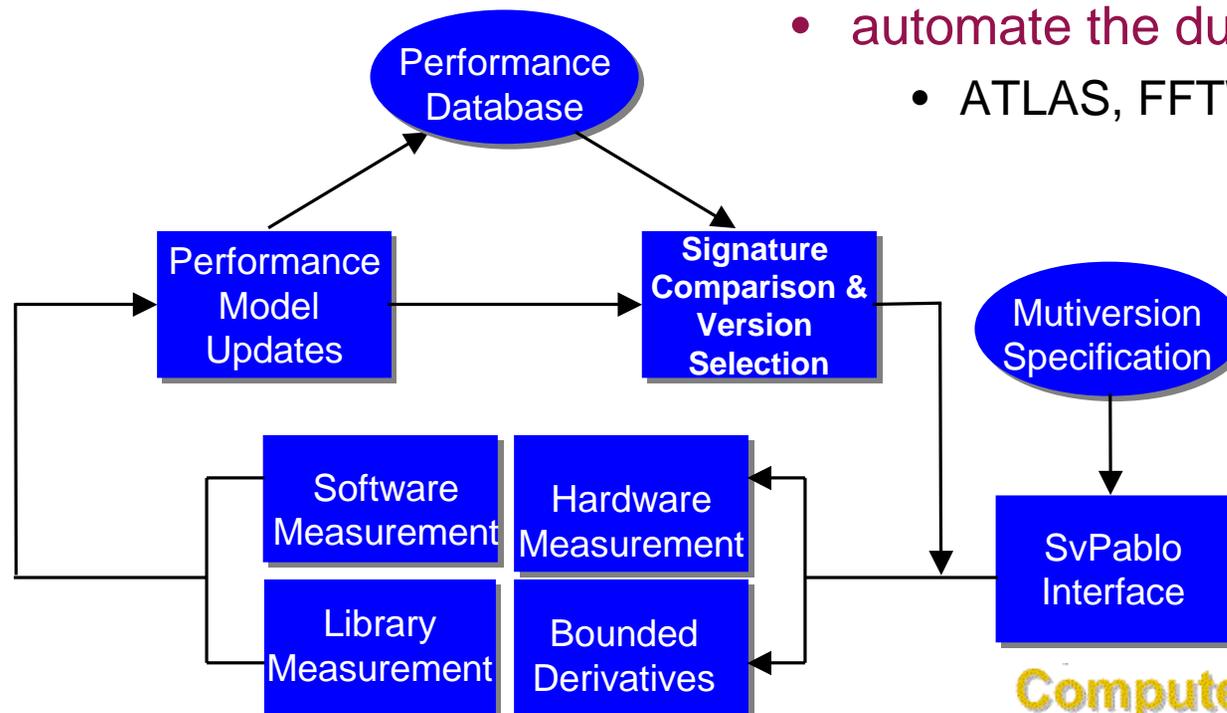
- automatic specialization

- **Development cost**

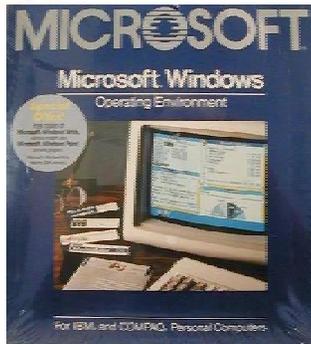
- quantitative cost-benefit ratios

- **The moral of the story**

- capture insights/experience
  - do what humans do well
- automate the dull stuff
  - ATLAS, FFTW, ...

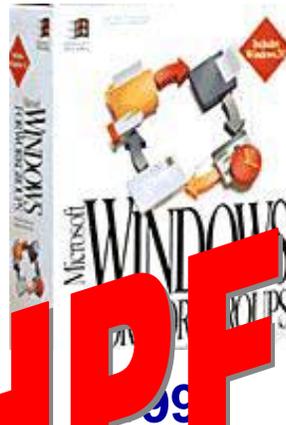


# Microsoft Lesson: Stay The Course

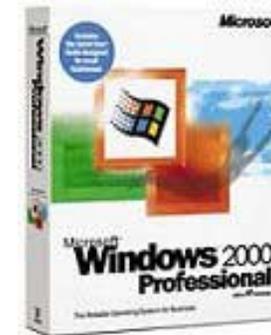


1985

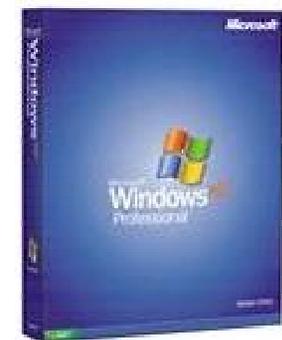
**HPF** **V5.0?**



1998



2000



2001

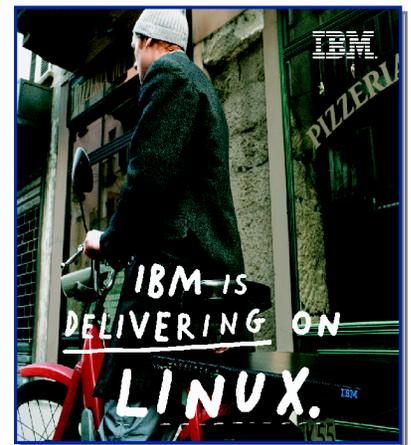
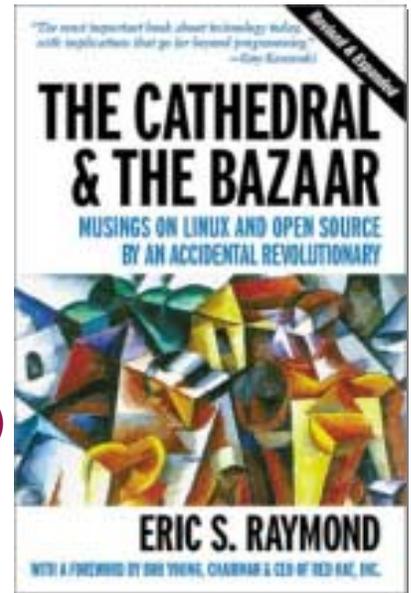
**Continue investment  
and innovation until it works!**

# HPC Open Source Software

- **Open source development**
  - not all software has an economic market
    - market cannot sustain development costs
    - HPC software is the canonical example
- **Economic lack does not imply lack of value**
  - promote software creation in “non-economic” markets
  - precompetitive collaboration among companies (e.g., OSDL)
- **U.S. PITAC subcommittee report on open source**
  - <http://www.ccic.gov/ac/pres-oss-11sep00.pdf>
  - urges U.S. federal investment in open source projects
- **NCSA open source license**
  - <http://www.opensource.org/licenses/UoI-NCSA.html>

**“The Linux issue is whether this is a fundamentally disruptive technology, like the microprocessor and the Internet. We’re betting that it is.”**

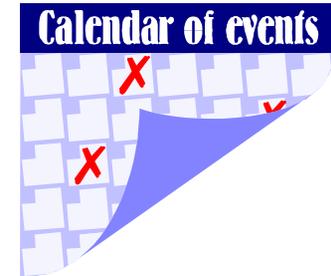
***Irving-Wladawsky-Berger, IBM***



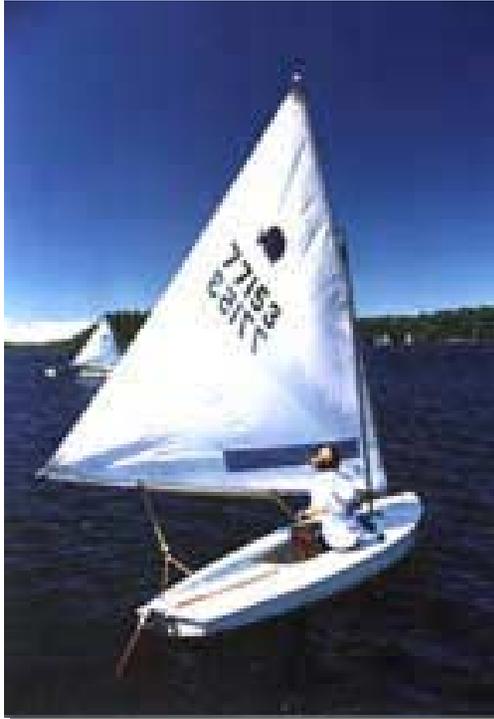


# HPC: Taking The Long View

- **Go long**
  - sustained, long-term exploration
  - long-term, strategic plans and support
  - twenty years is about the right time scale
    - think about major defense procurements
- **Go deep**
  - multidisciplinary examination of problems
    - understand applications and needs
  - resource scale matched to problem scale
  - \$500M-\$1B/year is about the right monetary scale
    - for each substantive project
- **Implications**
  - ask strategic questions
  - articulate strategic visions
  - make long-term commitments
  - *do not be afraid to fail*



# A Tool and Market for Every Task



- **Different development models**
  - standard market economics
  - long-term government investment
- **Each targets different applications**
  - understand application needs