

Manageability & Supportability Research Group

STRIDER:
**A Computer Genomics Approach
to Systems Management
and Support**

Yi-Min Wang

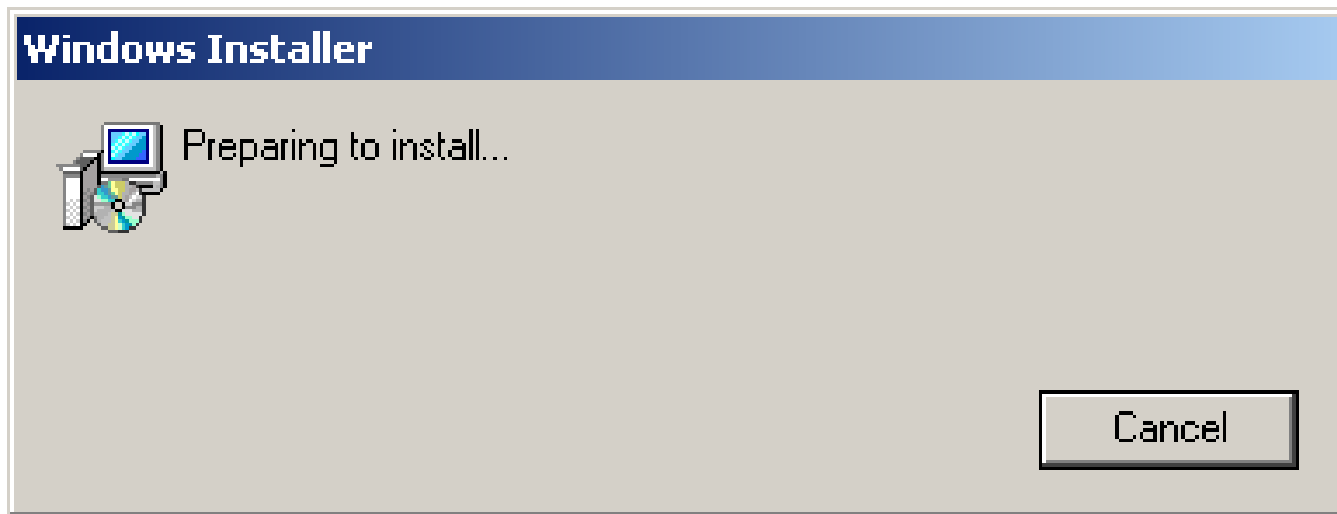
Microsoft Research, Redmond

*John Dunagan, Dan Simon, Chad Verbowski, & Helen J.
Wang, MSR Redmond*

Yu Chen, Yuan Chun, & Zheng Zhang, MSR-Asia

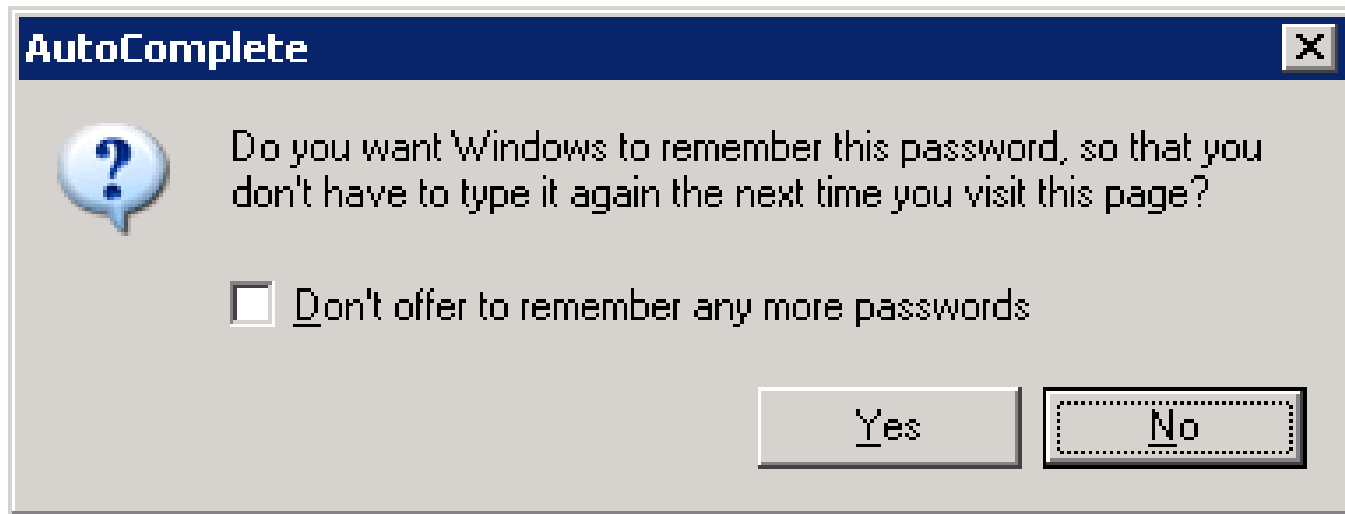
Rich's Registry Problem

- *“I don't need anything new to be installed!”*



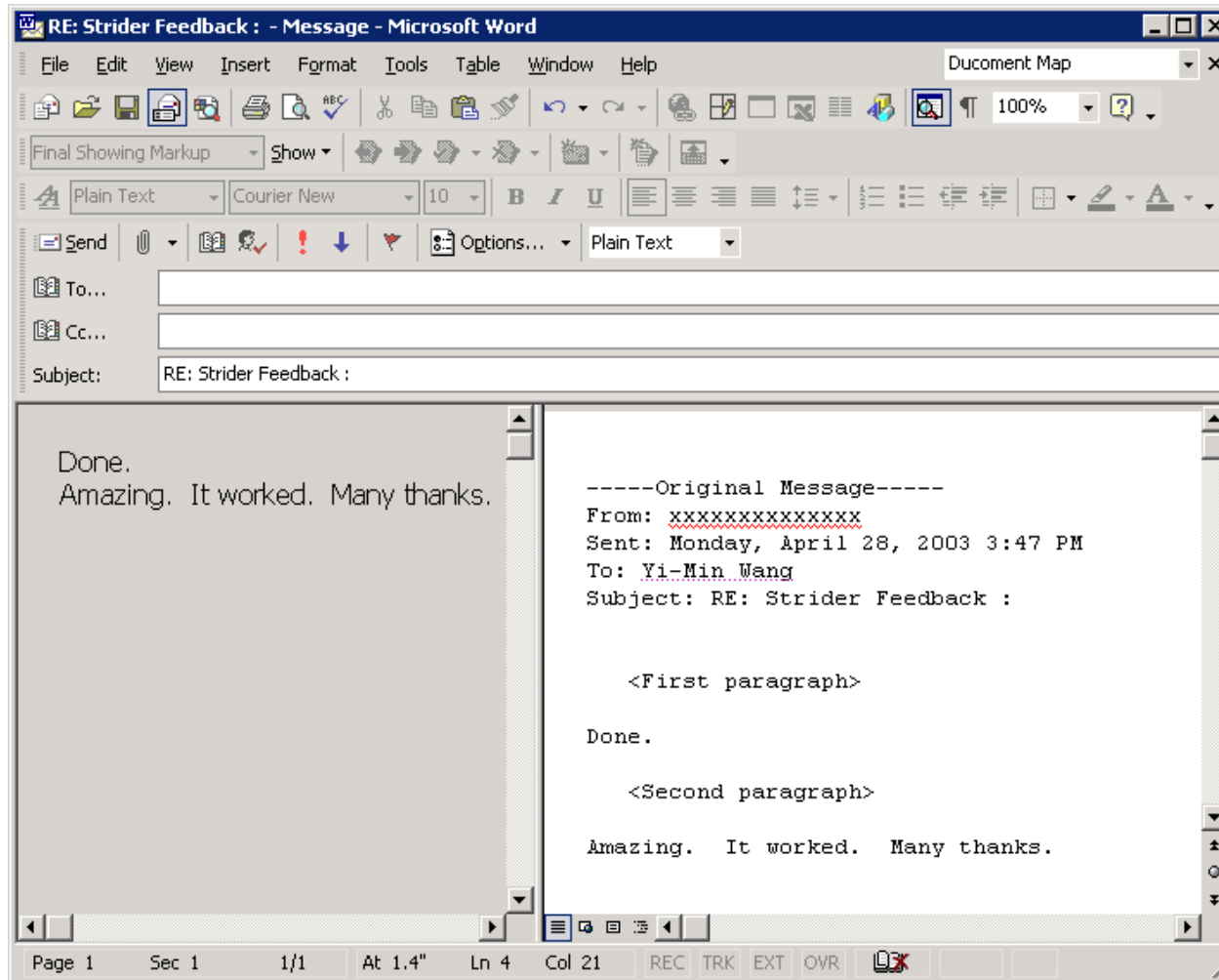
Joe's Registry Problem

- “How do I get this option back, once I said Don't?”



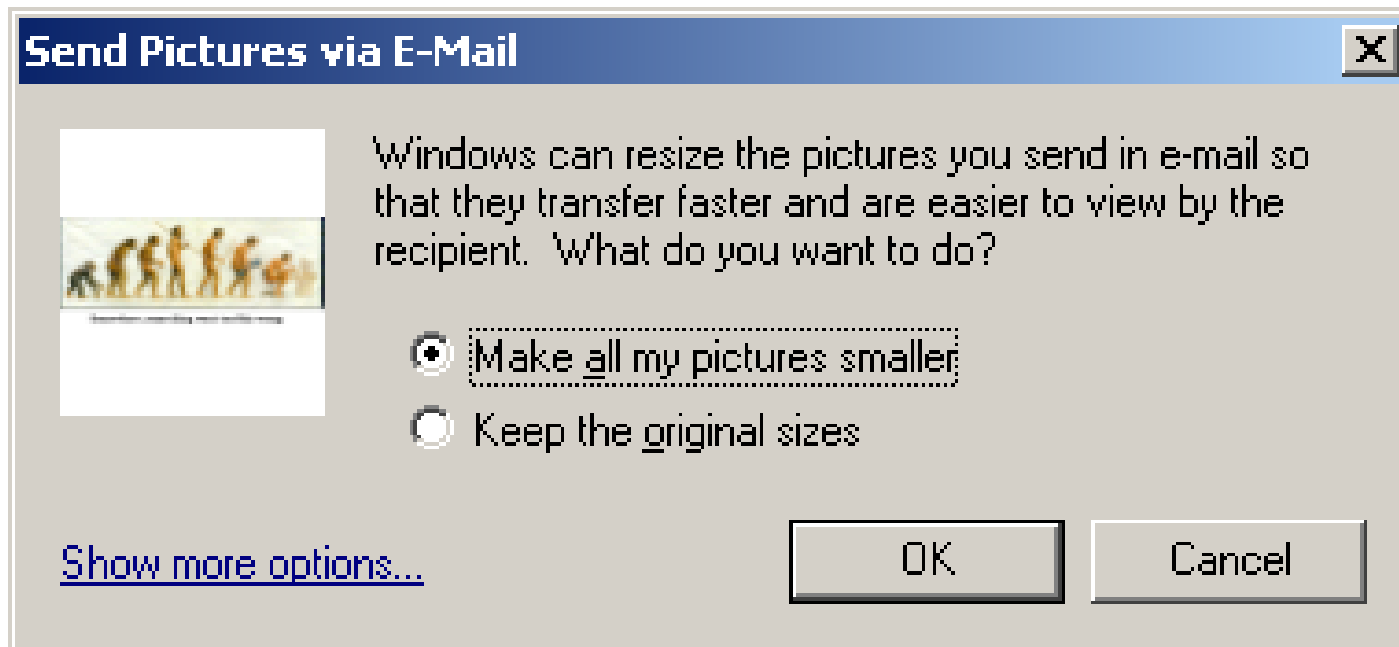
Leslie's Office-Word Problem

- “How do I turn this option off?!”



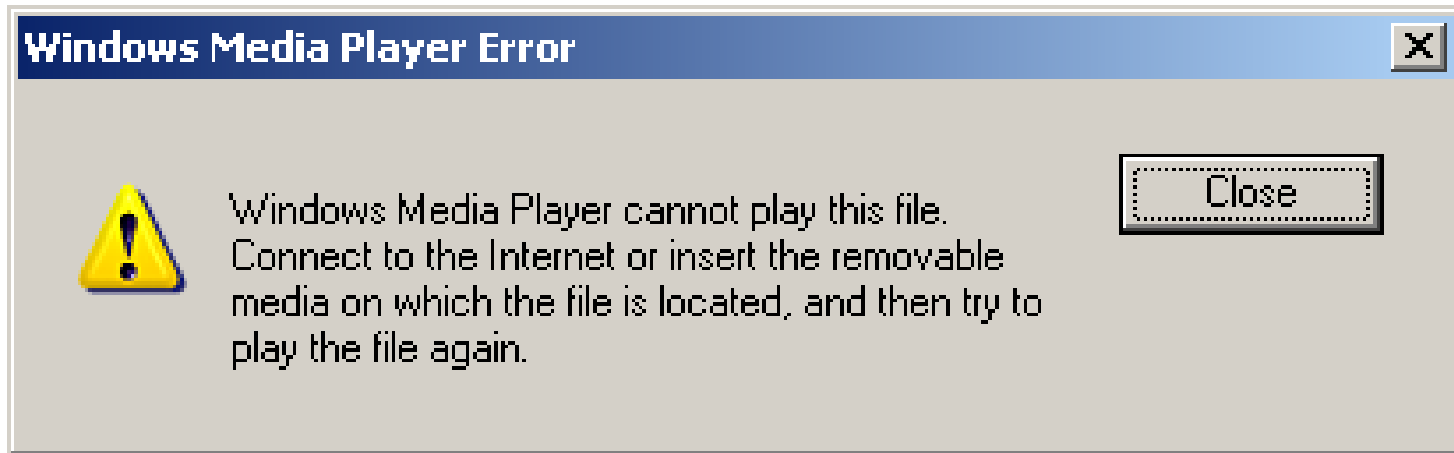
Bertrand's Registry Problem

- “What happened to my JPG resize option?”

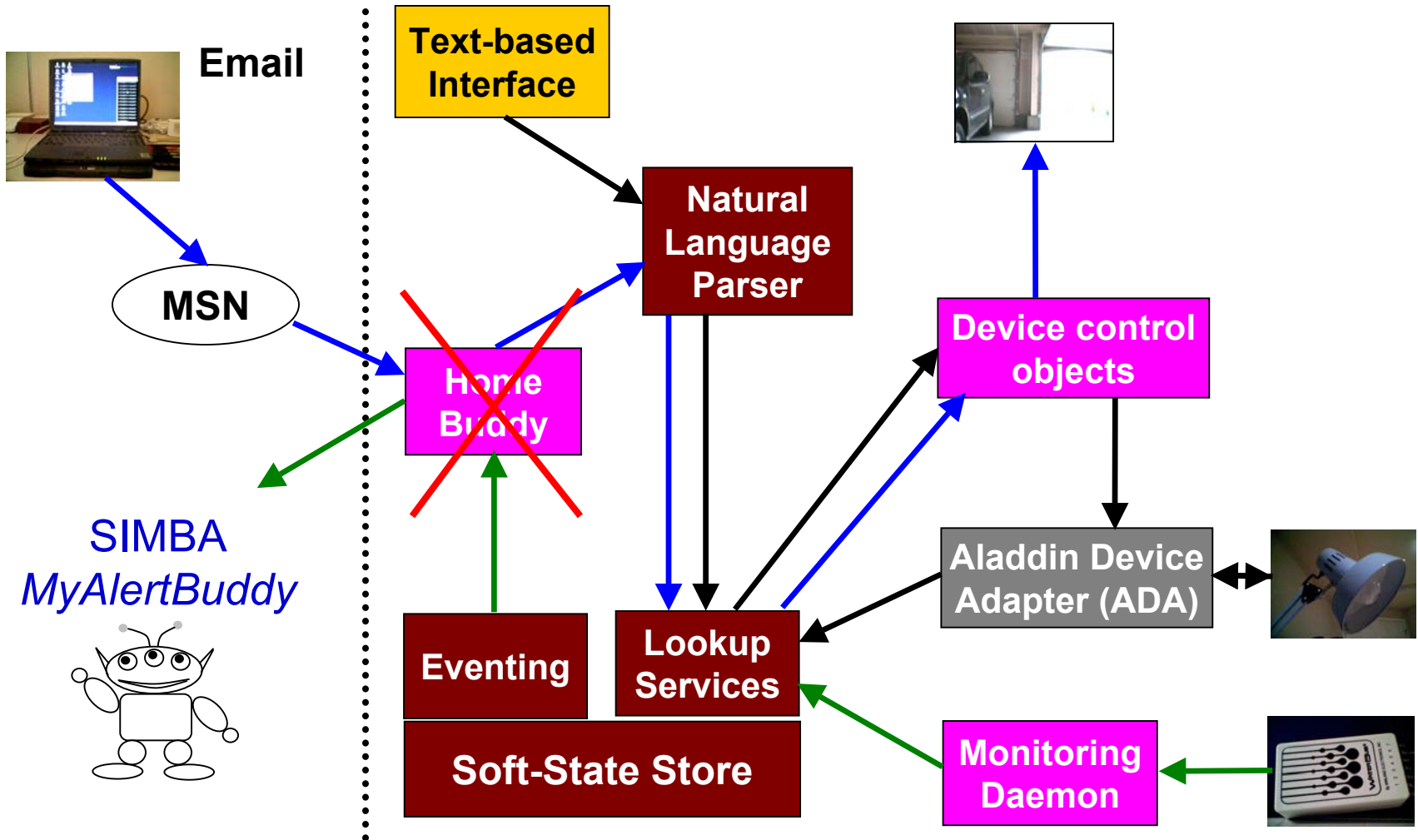


Chad's Registry Problem

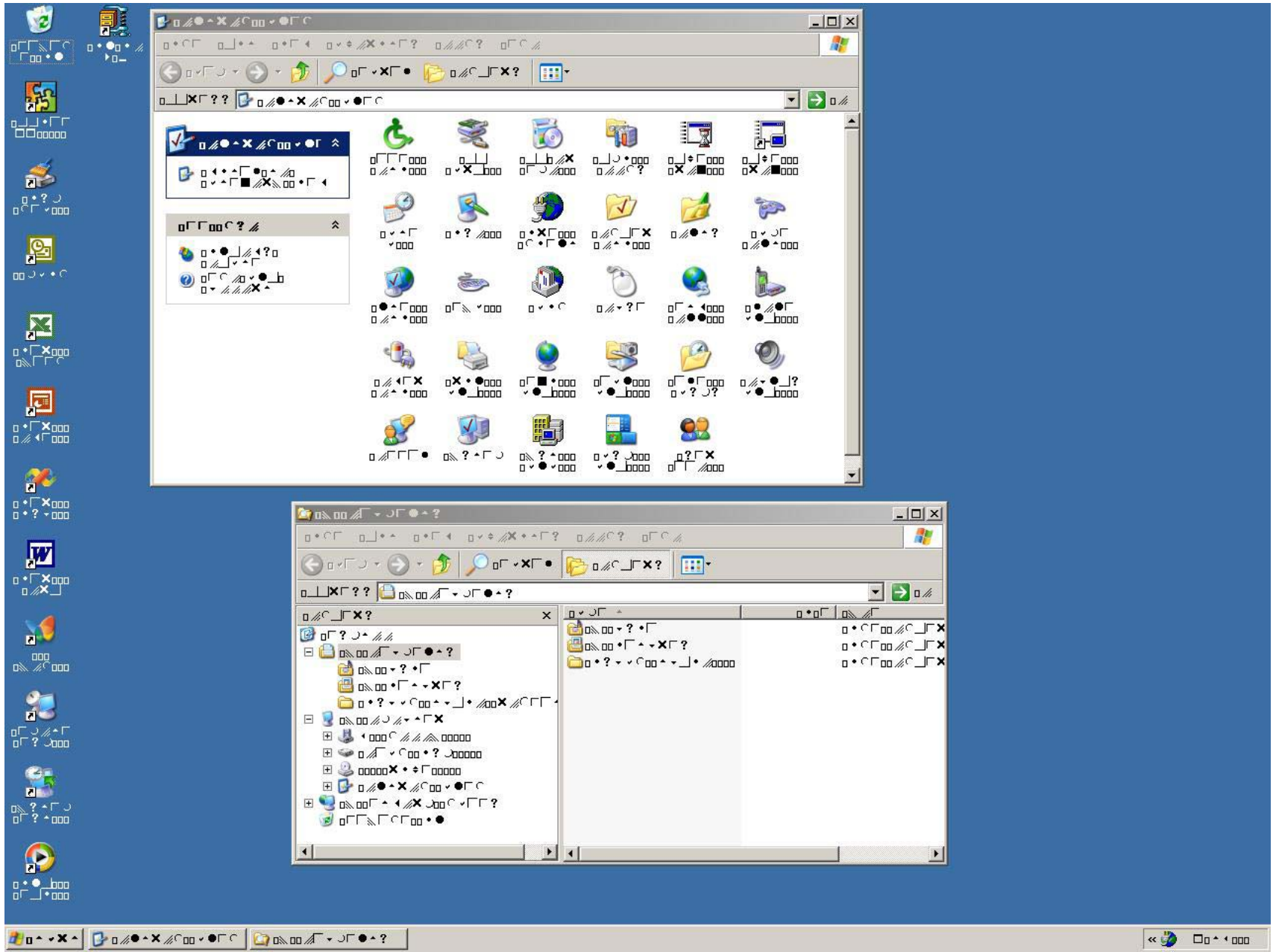
- *“My Internet connection is perfectly fine!”*



Aladdin's Registry Problem



Scott and Susi's Registry Problem



OUTLINE

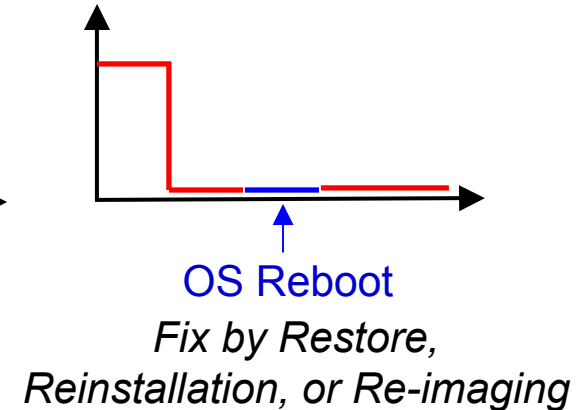
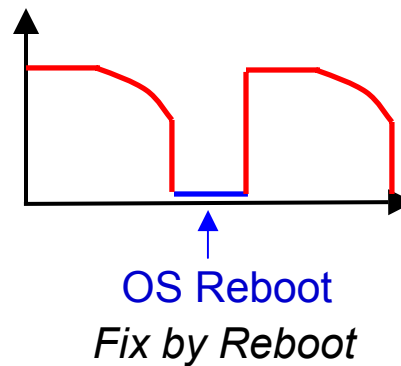
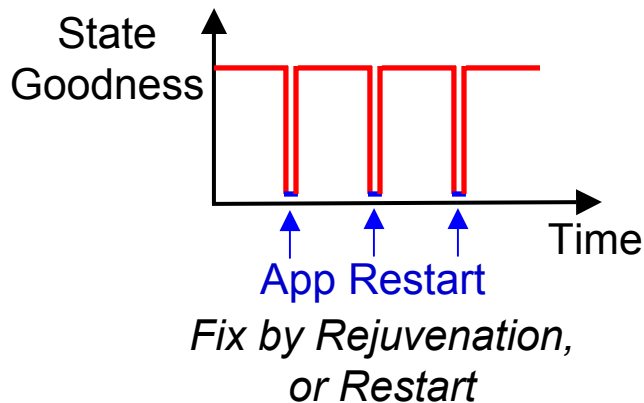
- The Problem: Computer Fragility
- The Solution: Strider Troubleshooter
 - Strider Principles: problem formulation & decomposition
 - Strider Architecture
 - Preliminary Experimental Results
 - Limitations & Challenges
- Beyond Troubleshooting
 - *A Comprehensive Solution to Computer Fragility*
 - Strider On-Line Analyzer
- Summary

The Problem: Computer Fragility

- ***“It worked yesterday, but not today.”***
- ***“It works on that machine, but not this one.”***
- Latencies between “fault” & “error”, and between “error” & “failure”
- Initial Focus
 - Config problems in Windows Registry
 - Problems that persist across reboots → Higher Total Cost of Ownership (TCO)

Current Focus

Crash			<i>Today's Strider</i>
Hang			<i>Today's Strider</i>
Unexpected Behavior			<i>Today's Strider</i>



- User-centric view

- “Troubles” broadly defined as *“not delivering user-expected services”*
- It doesn’t matter what the spec says
- It doesn’t matter if it can be “easily” fixed by a hard-to-find option
- It doesn’t matter if it’s caused by user mistakes or not well-understood side effect

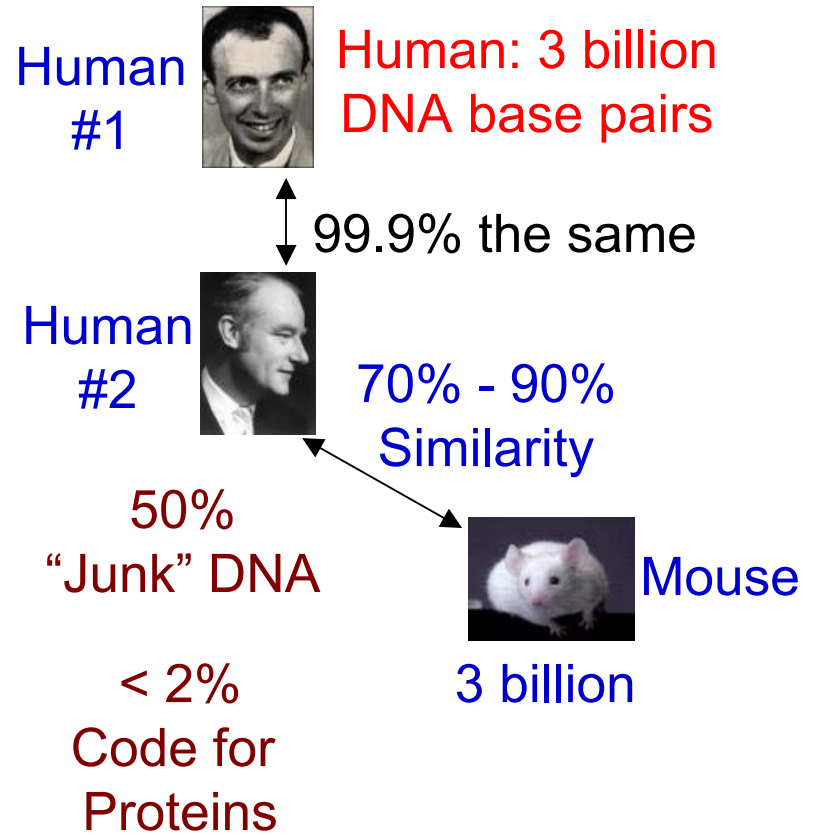
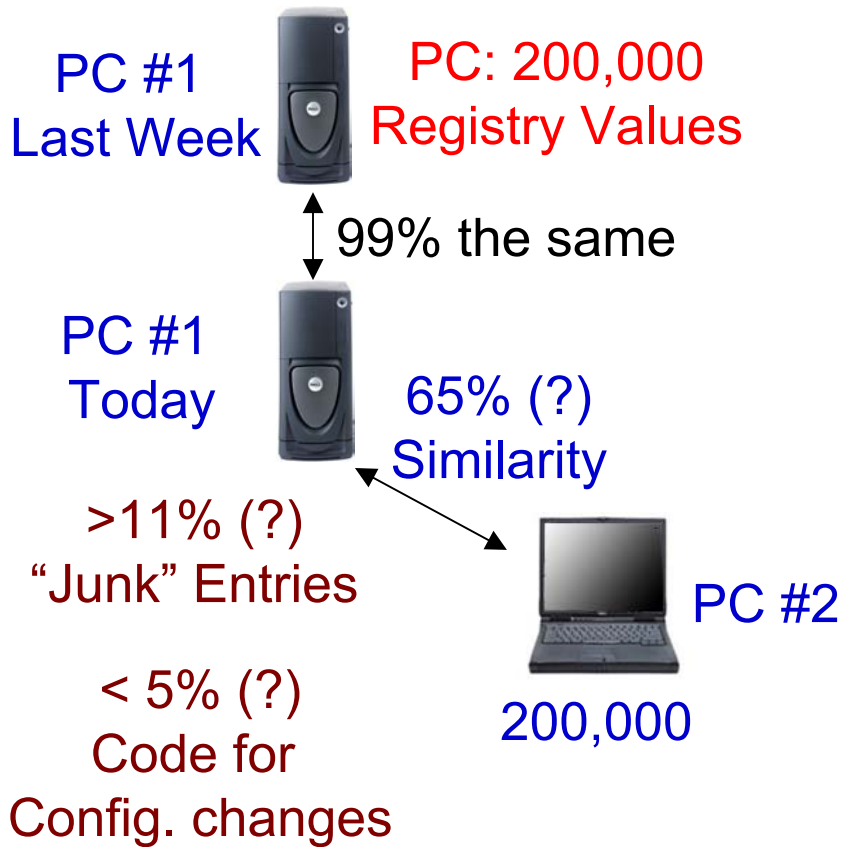
The Solution: Strider Troubleshooter

- The approach: state-based troubleshooting
 - Treat all states as a high-dimensional state vector
- The goal: point to the root cause entry to suggest a localized undo
 - As opposed to global undo followed by selective roll-forward
- Key idea: use system checkpoints for troubleshooting
 - As opposed to using them for global rollback (whole-system rollback vs. per-app-execution rollback)

Why not just use global rollback?

- In practice, many things should not be rolled back
 - User passwords, documents, pictures, etc.
- Consistency / dependency information not available / complete
 - Apps & system components may require an (unspecified) set of states to be consistent
- Selective rollback may break consistency
 - Especially for long-latency problems
- Hard to distinguish among user data, app data, and app programs
 - Roll-forward is hard as well

Inspired by *the Human Genome Project*



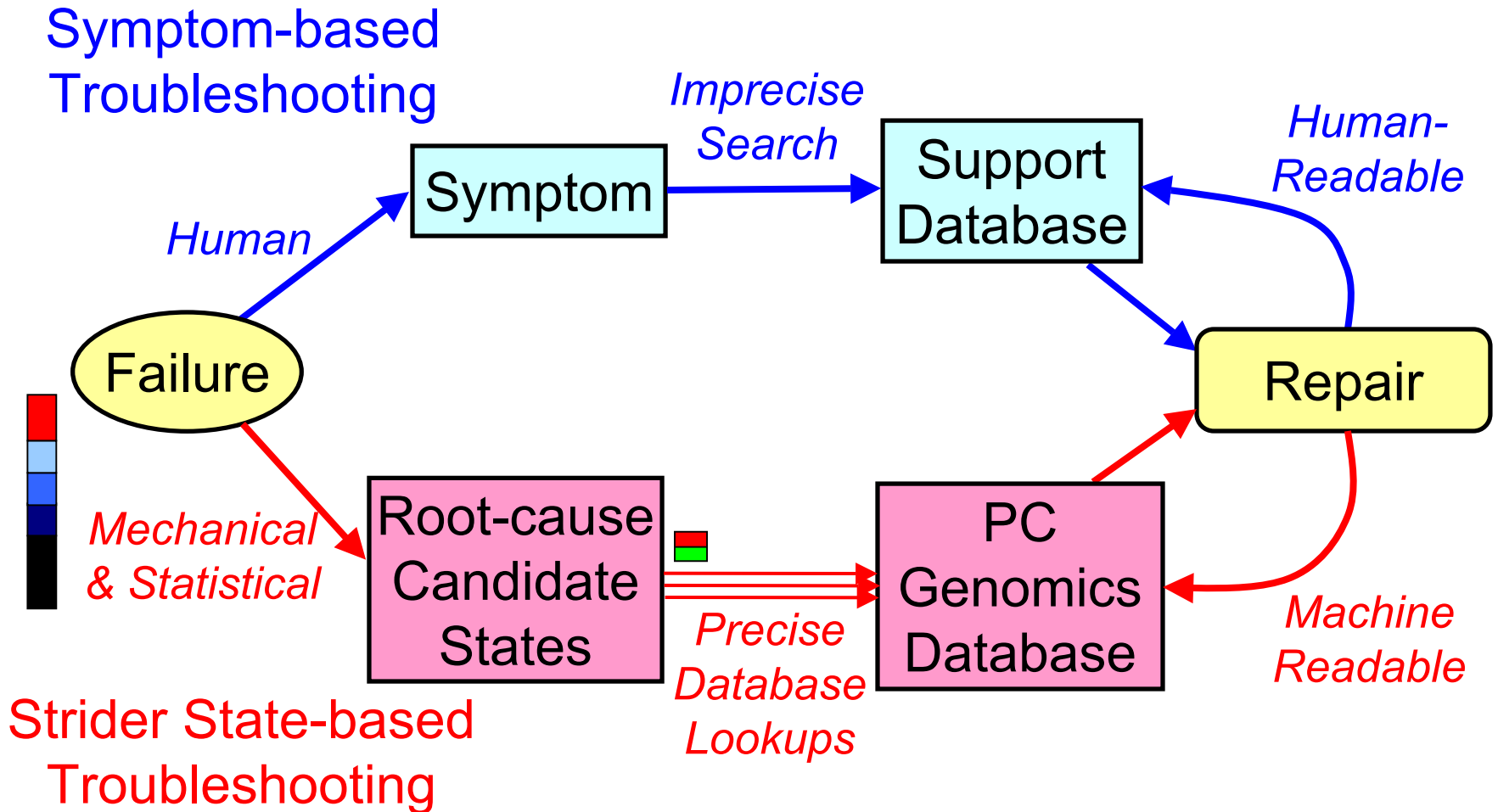
Registry Entries for
"No Internet Connection" disease
Found at the *EnableAutodial* entry under
`HKCU\Software\Microsoft\Windows\
CurrentVersion\Internet Settings`

Gene for
Huntington's disease
Found at the tip of the short arm of
`Chromosome 4`

- **The Fundamentals: Strider Principles**
 - #1: *Diagnostics Through Genomics*
 - #2: *Attack The Mess With The Mass*
 - #3: *Seek Constancy In An Inconstant World*
 - #4: *Think Outside The White-box*
- **The Strider Architecture**
 - Decomposition into 5 components
 - The Strider Toolkit of command-line tools
- **Beta Release**
 - UI troubleshooter with subset of capabilities available internally

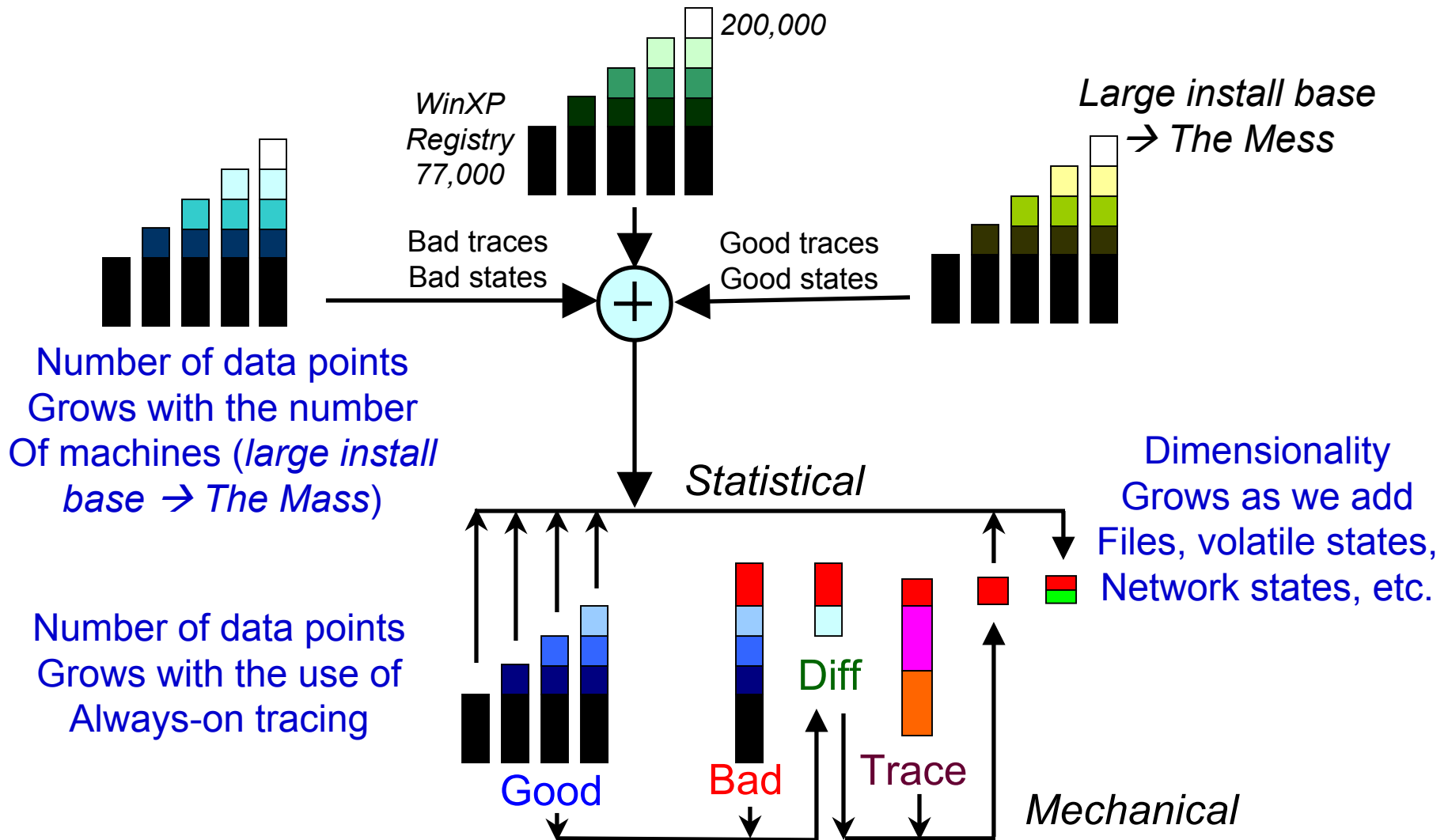
Diagnostics through Genomics

First-level decomposition: Mechanical, Statistical, & Database



Attack The Mess With The Mass

Second-level decomposition: Diff, Trace, & Intersection



Seek Constancy In An Inconstant World

Self-filtering of complexity as noise

- Most of the differences are not significant for systems management and troubleshooting
 - Registry entries that are constantly changing are less important; they are simply “operational states”
 - *Inverse Change Frequency (ICF) ranking*
 - Registry entries that are always different on different machines constitute natural diversity among Windows machines
- Start with deterministic bad state, end with deterministic bad behavior
 - Nondeterministic activities in-between are often less important
 - *Order ranking*

Think Outside The White-box

Handling legacy with behavior monitoring & modeling

- The good state from the past or on another machine → Black-box manifest for “*the golden state relevant to the current failure*”
- Registry behavior monitoring → Black-box manifest for *which entries are operational states and which are configuration settings*
- State-to-UI mapping → Black-box manifest for *which UI owns which set of states*
 - E.g., comprehensive tracing of all UI's on WinXP

Registry Behavior Monitoring & Modeling

- Preliminary results for corporate desktops
- Registry contains about 200,000 values
- In a 3-month period with 90 checkpoints
 - 84% never changed:
 - Static installation or configuration settings
 - 16% (31,000) changed at least once
 - 21,500+ (11%) from “*Operation Registry*”
 - Windows\ShellNoRoam, Cache, MRU, Counts, etc.
 - 4,000+ (2%) from “*Installation Registry*”
 - Software\Classes, Installer\, Uninstall\, etc.
 - 5,500- (3%) from “*Configuration Registry*”
 - SMS (1,000), Print (1,000), Jet (300), Telephony (230), Office Language (200), etc.

Strider Architecture with 5 Components

- **Mechanical**

1. **State Diff**: what have changed since “last known working state”
2. **State Tracing**: what actually get used by the failing program execution
3. **Diff-Trace Intersection**: which changed states actually matter

- **Statistical**

4. **State Ranking**:

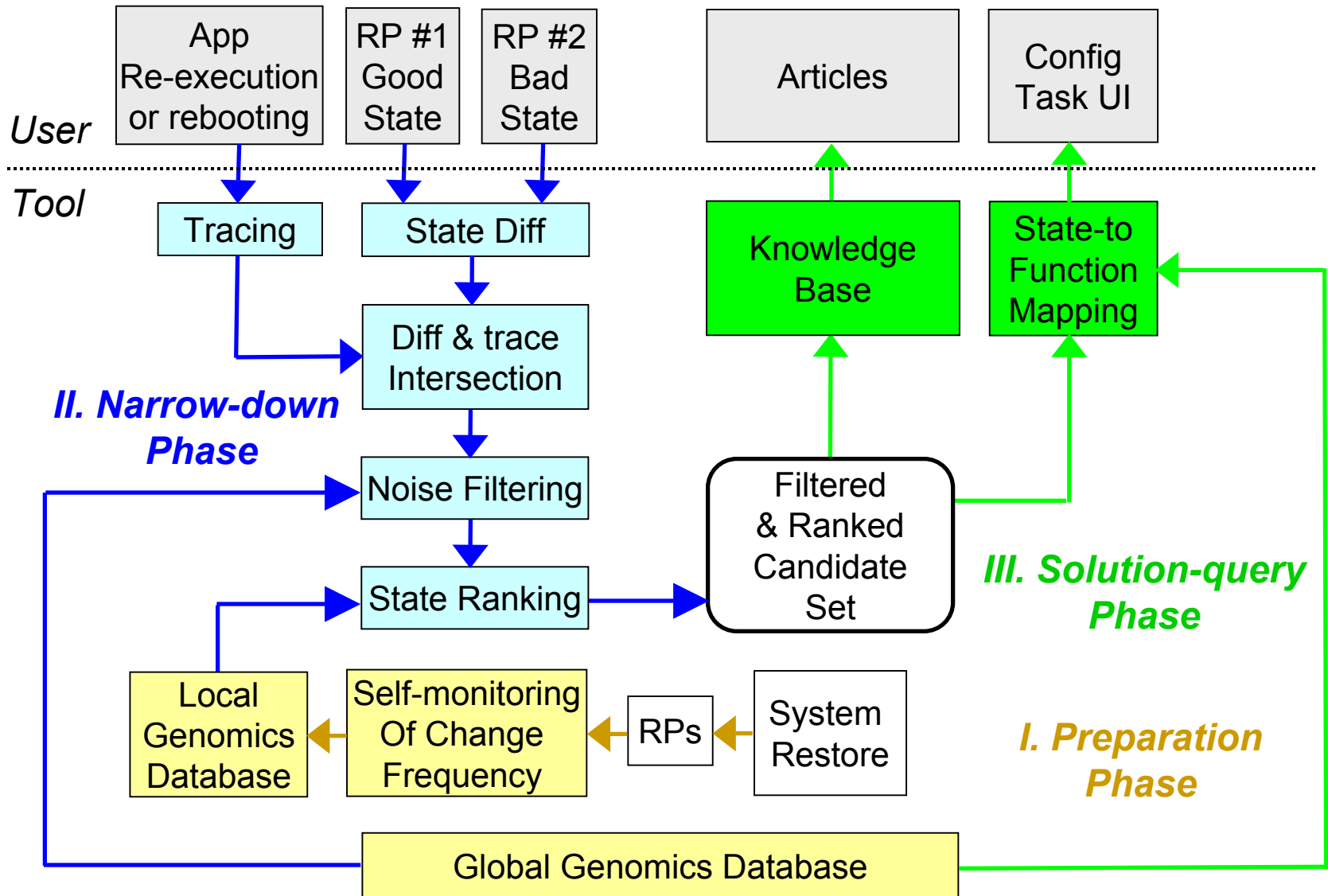
- *Inverse Change Frequency (ICF) ranking*: states with high change frequencies are less likely to be root causes
- *Order ranking*: states accessed earlier in the trace are more likely to be root causes

- **Database**

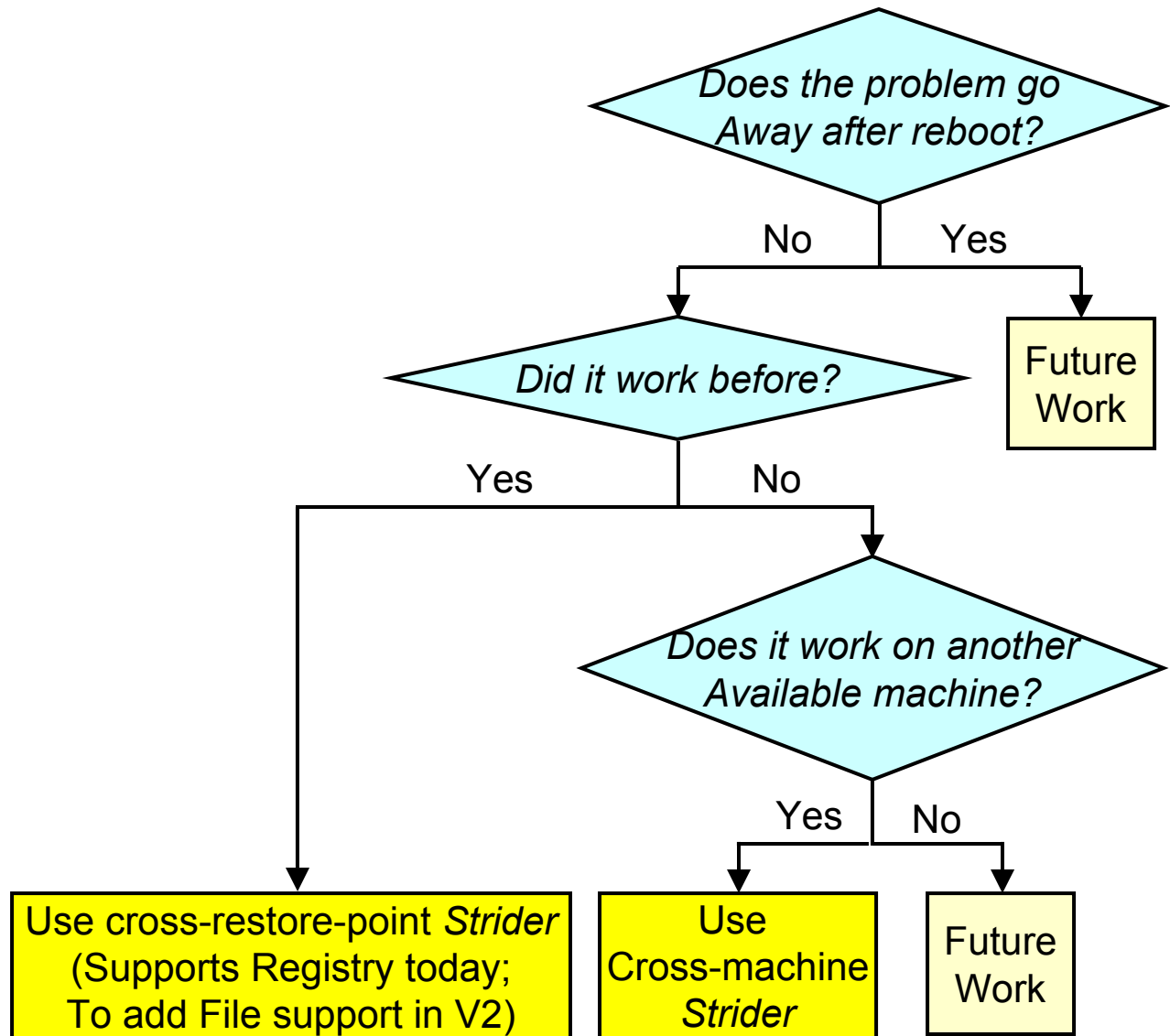
5. **Computer Genomics Database**: state functional & failure info

- Noise filtering: *“Is this a junk entry?”*
- State-to-function mapping: *“Who owns this entry?”*
- Knowledge Base: *“Are there known problems with this entry?”*
- **Local database – based on state ranking threshold**

Current Strider Troubleshooter Architecture



When to invoke Strider

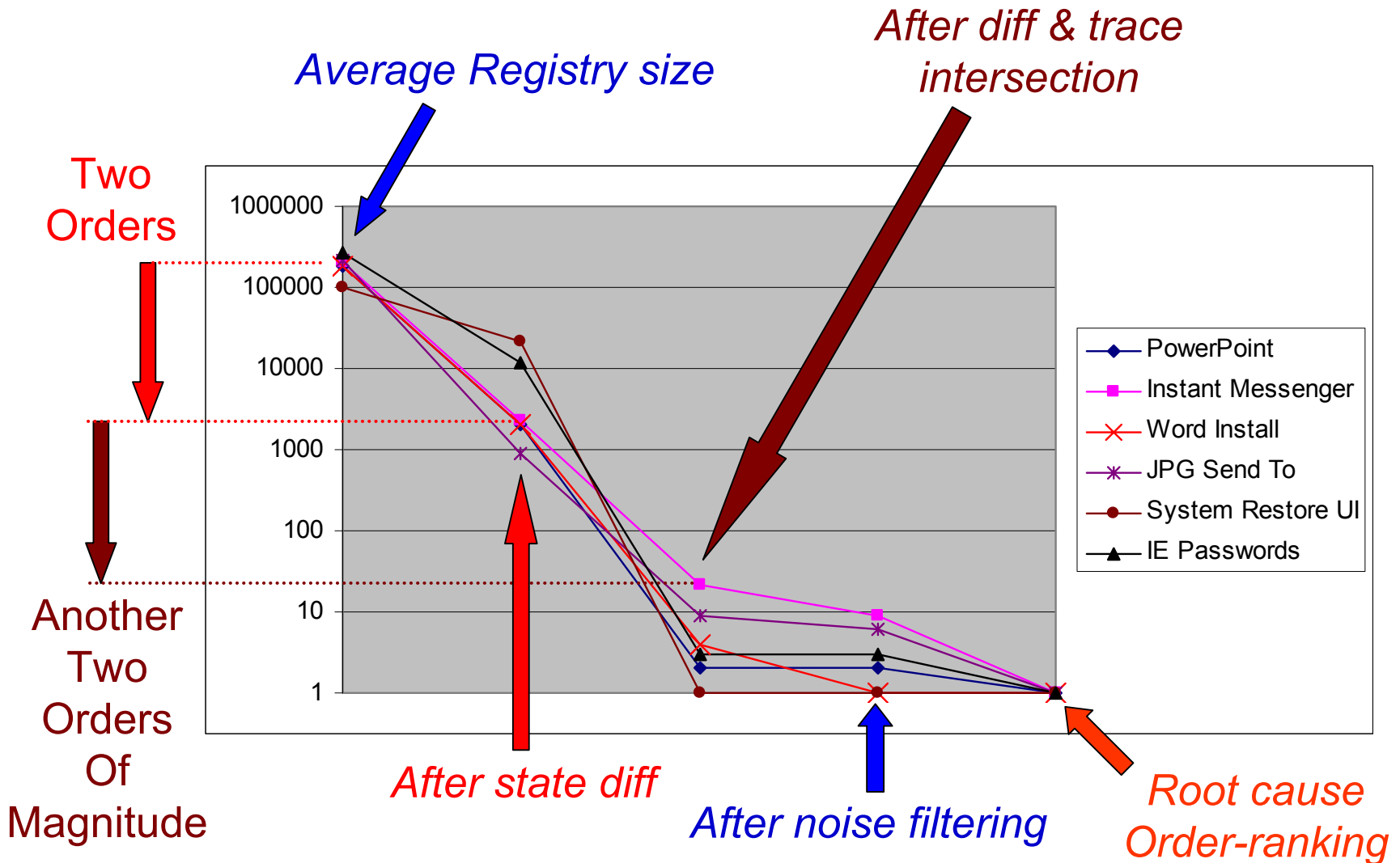


Case Studies & Performance Evaluation

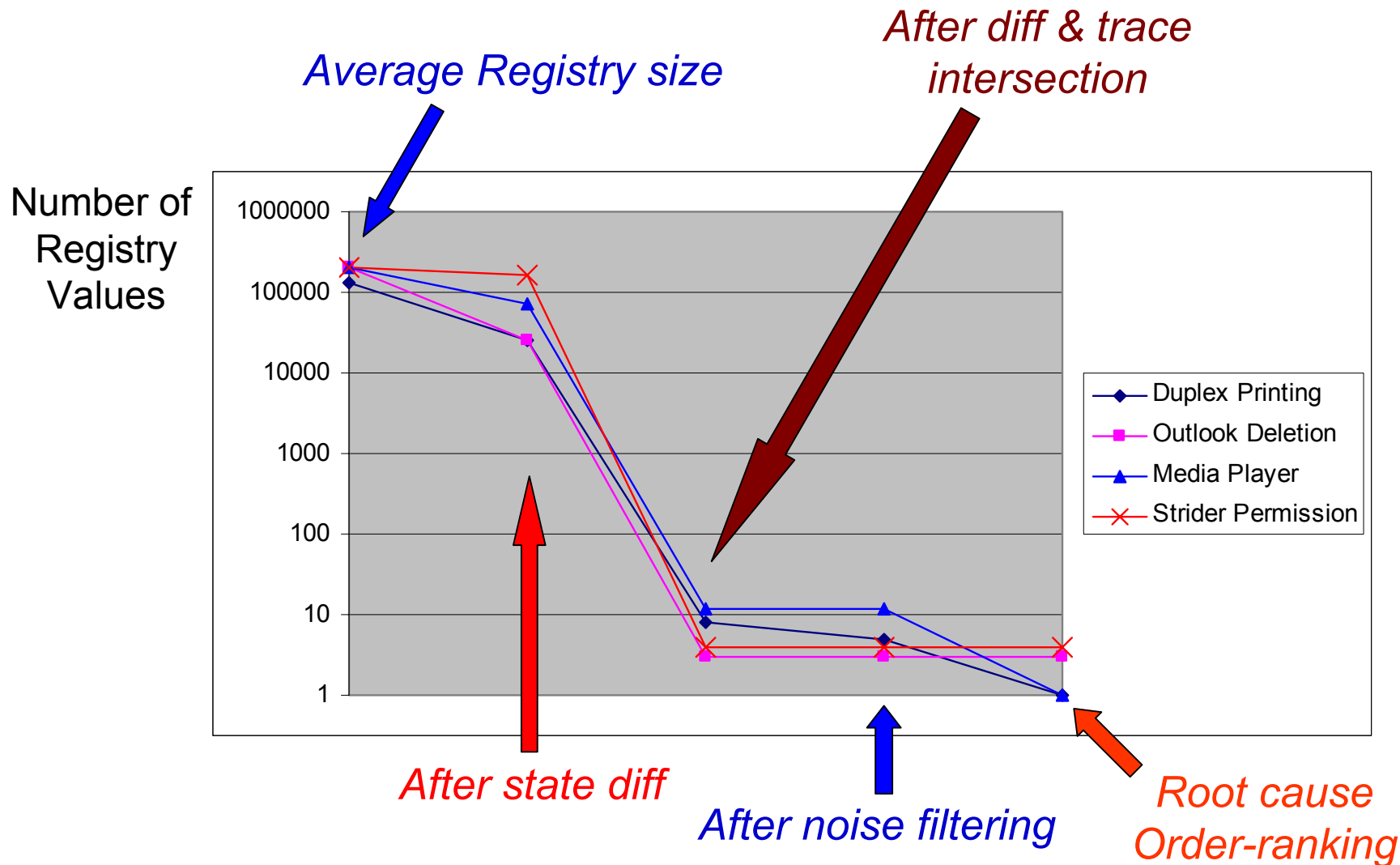
(Subset from <http://SysMan/Strider/Fellowship/>)

- **Media Player *OpenURL* problem**
 - *EnableAutodial* set to 1
 - Root cause: *HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\EnableAutodial*
- **JPG Send To → Mail Recipient no-resize-option problem**
 - *.jpg\PerceivedType* deleted
 - Root cause: *HKCR\.jpg\PerceivedType*
- **System Restore failed to display restore points (pre-SP1)**
 - *.htc\Content Type* deleted
 - Root cause: *HKCR\.htc\Content Type*
- **PowerPoint *SlideShow* problem**
 - Dual-monitor mode turned on
 - Root cause: “*Registry\Machine\System\CurrentControlSet\Control\Video\{97136B31-AD36-422F-96FB-4821C6187AD7}\0001*”
- **Instant Messenger performance problem**
 - Firewall client software disabled
 - Root cause: *HKLM\Software\Microsoft\Firewall Client\Disable*
- **Word “*Prepare to install*” problem**
 - Japanese language turned on
 - Root cause: *HKCU\Software\Microsoft\Office\10.0\Common\LanguageResources\1041*

Cross-restore-point Strider Effectiveness



Cross-machine Strider Effectiveness



Current Research Effort

- Formulating troubleshooting cost for performance evaluation
 - **Minimize** $\sum_p \mathbf{Cost}(\mathbf{Strider}(p))$
 - $p = (State_{good}, State_{bad}, Trace_{bad})$
 - $Strider() = Ranking(Filtering (Diff \& Intersection))$
 - $Cost() = \sum_e Cost(e)$ where e ranks no lower than root cause in $Strider(p)$, or cost for a Strider failure
- **StriderX: cross-machine Strider**
 - Algorithms for finding $State_{good}$
- **Registry statistical analysis**
 - Algorithms for *Ranking* and *Filtering*

Limitations & Challenges

- Rely on the user to specify **correct** “good state” & “bad state”
 - User may remember the “good state” incorrectly
 - Latent errors/failures
 - User may specify a “bad state” that does not capture the root cause
 - App may persist config setting only upon exiting
- Can we do it by just analyzing all the states?

- Rely on the user to gather *sufficient* trace
 - Start with per-process, per-action trace
 - Synchronous, direct dependency
 - Next with all-process, per-action trace
 - Synchronous, indirect dependency
 - Need to improve with process dependency tracking
 - Next with trace including app launch
 - Asynchronous dependency; error-to-failure latency
 - Next with trace including OS booting
 - Need to improve with OS-component dependency tracking
 - Sometimes it's not obvious what to trace
 - E.g., mouse stopped working
 - Sometimes it requires creative thinking to get “better” traces
 - Per-action trace in place of boot trace

- **Limitations of current state-ranking techniques**
 - Batch read of Registry entries at boot time or app launch sometimes breaks order ranking
 - Big binary blobs often breaks the Inverse Change Frequency (ICF) ranking
 - Need statistical techniques to de-emphasize ICF ranking for such data
 - Abnormal data content not captured by ICF
 - Static ICF dictionary instead of customized one

Beyond Troubleshooting:

A Comprehensive Solution to Computer Fragility

1. Troubleshooting

- Strider Troubleshooter

2. Rollback

- System Restore (with Rollback Consistency Checkers)

3. Self-monitoring

- Always-on Strider On-Line Analyzer (change audits & behavior monitoring)

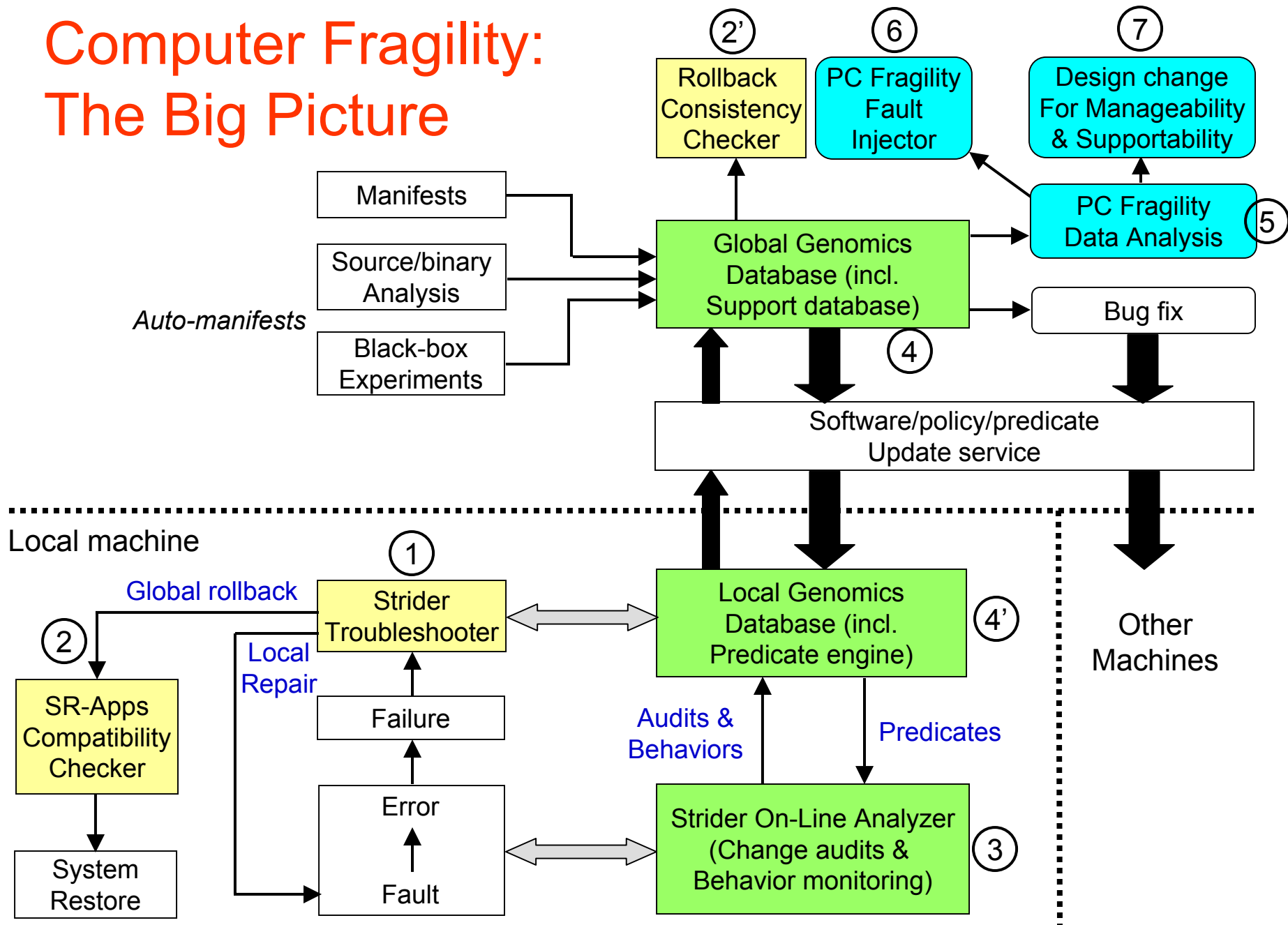
4. PC Genomics Database

5. PC Fragility Data Analysis

6. PC Fragility Fault Injector

7. Design for Manageability & Supportability

Computer Fragility: The Big Picture



Strider On-Line Analyzer:

TiVo for Computers

- **Change audits**
 - To catch which user/app/system component is breaking other apps or the system
 - Be able to answer the following question at any time: *“What have changed on my machine since last week?”*
- **App/system good-behavior baselining**
 - App behavior monitoring & modeling
 - Fine-grained state behavior modeling
- **Challenges**
 - Efficient, always-on Registry/File access tracer
 - Effective noise filtering to reduce log size
 - User-friendly presentation of change-audit summary

Summary

- Real world may not be as messy as the worst case
 - Individual's experience, opinion, and imagination can be discouraging
- Systems management is a huge research opportunity
 - Problem formulation & decomposition: most important first step to deal with *complexity of the whole system*
 - Then we can do technical research in individual components
- Scalable, scientific evaluation methodology remains a research challenge
- Techniques that allow smooth transition from legacy to brave new world are key to success
 - Common perception: systems management = engineering for legacy; systems research focuses on brave new world