Gold and Fool’s Gold: Successes, Failures, and Futures in Computer Systems Research

Butler Lampson
Microsoft

Usenix Annual Meeting

June 2, 2006

Context: Moore's Law and Friends

	months for 2 x	10 *years*	6/2006 *best*	6/2006 *cost*
Processing	18	100 x	2x4 GIPS	$20/GIPS
Storage (disk)	12	1,000 x	750 GB	$0.35/GB
LAN BW	18	100 x	1 GB/s	$1/MB/s
WAN BW	12	1,000 x	4 GB/s	$1000/MB/s/mo
Display pixels	360	10 x	4 M	$100/M

Implication: spend hardware to simplify software.

Huge components work (operating system, database, browser)

Better hardware enables new applications.

Complexity goes into software.

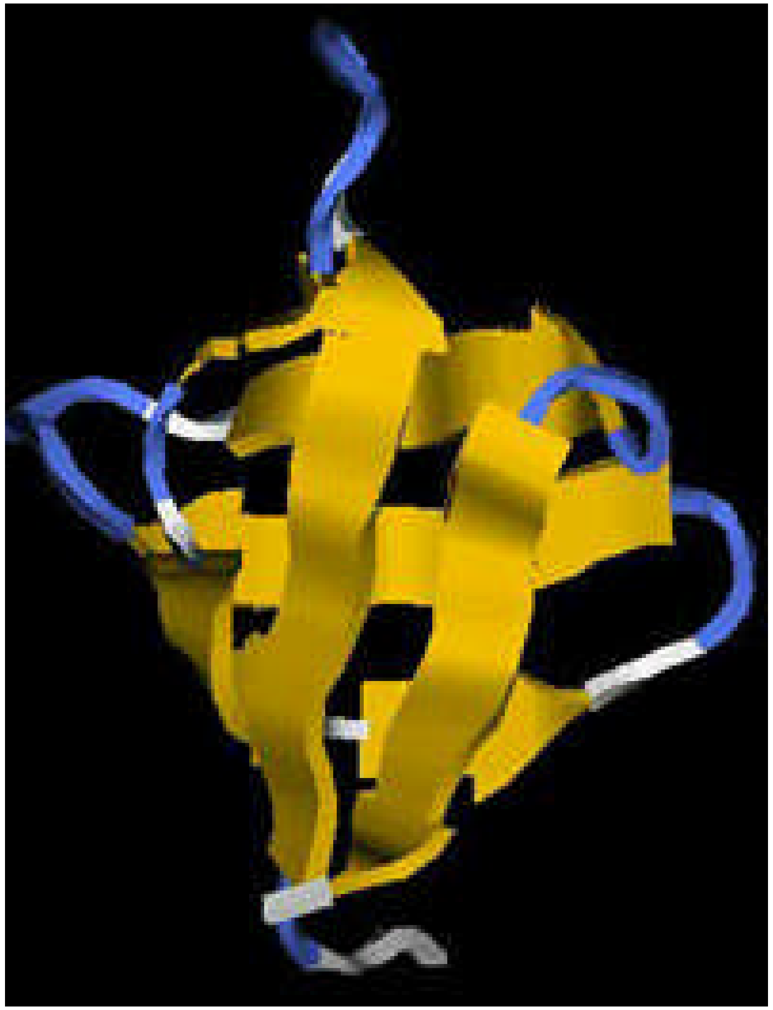
What is computing good for?

Simulation	1950	nuclear weapons, protein folding, payroll, games, virtual reality
Communication (storage)	1980	email, airline tickets, books, movies, Google, Terraserver
Embodiment (physical world)	2010	factories, cars, robots, smart dust

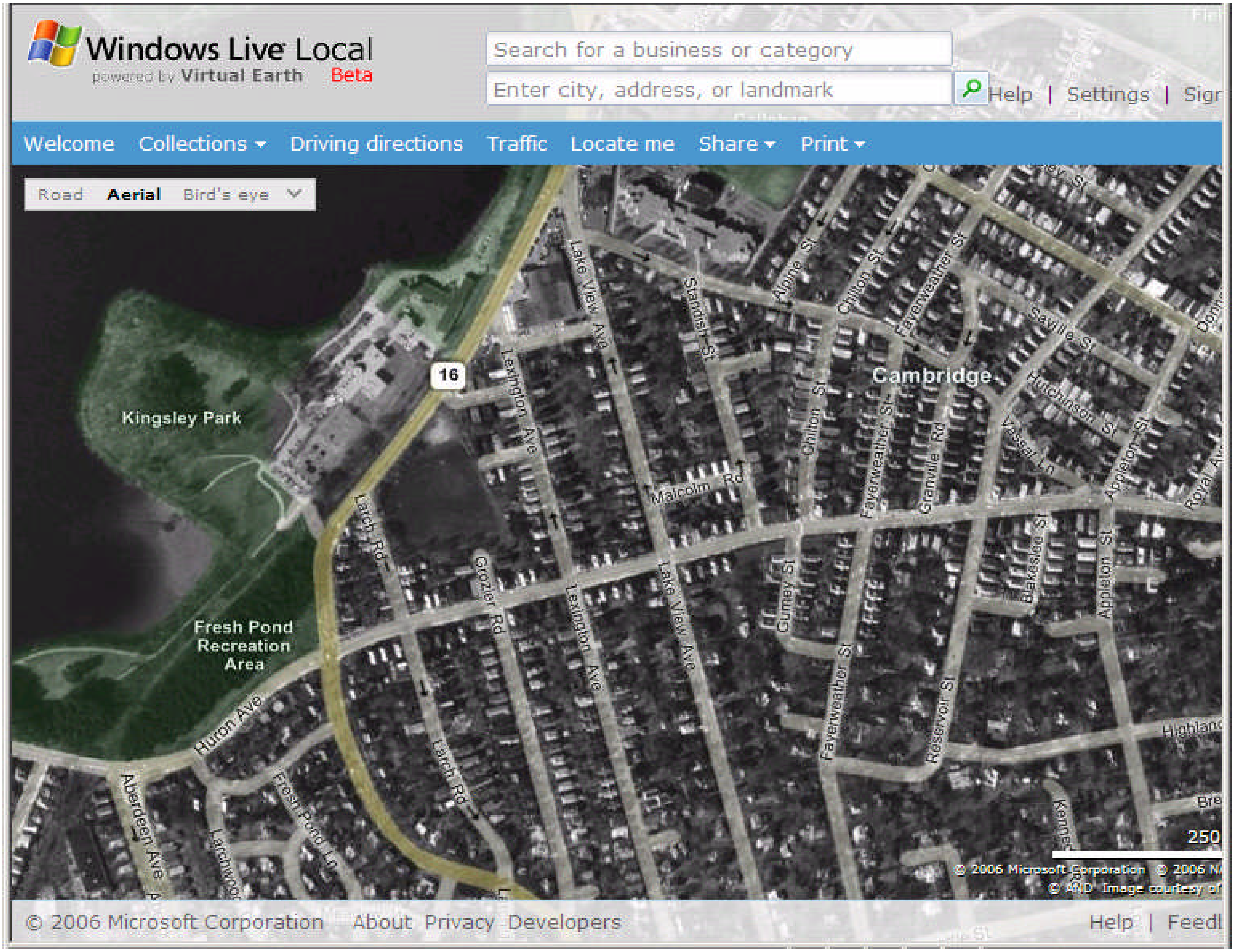
Simulation: Protein Folding

UNFOLDING OF THE DNA BINDING DOMAIN OF HIV INTEGRASE

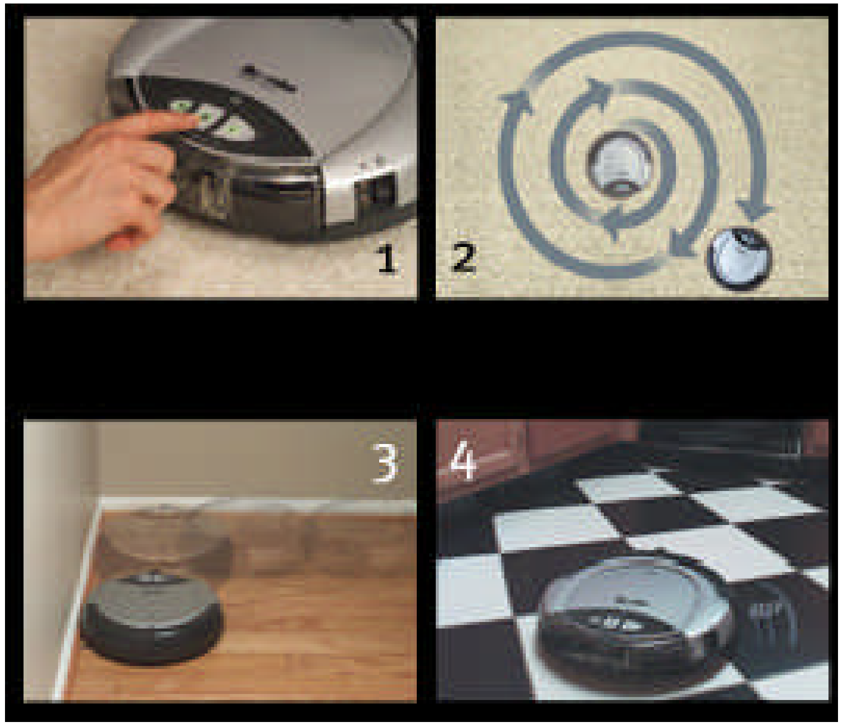
HIV uses proteins to insert its genetic code into our DNA. The DNA binding domain of HIV integrase (below) is the protein which HIV uses to grab onto our DNA such that it can then connect its genetic code into ours.



Communication: Maps and Pictures



Embodiment: Roomba Vacuum



256 *bytes* of RAM, $199

History: What Worked?

YES

Virtual memory
*Address spaces
*Packet nets
Objects / subtypes
RDB and SQL
*Transactions
*Bitmaps and GUIs
Web
Algorithms

NO (Not Yet?)

*Capabilities
*Fancy type systems
Functional programming
*Formal methods
Software engineering
*RPC (except for Web)
*Distributed computing
Persistent objects
*Security
RISC

History: What Worked?

MAYBE

Parallelism (but now we *really* need it)

Garbage collection

Interfaces and specifications

Reuse / components

Works for	Unix filters
	Platforms
	Big things (OS, DB, browser)
Flaky for	Ole/COM/Web services

The Failure of Systems Research

- We didn’t invent the Web
- Why not? Too simple
  - Old idea
    - But never tried
  - Wasteful
    - But it’s fast enough
  - Flaky
    - But it doesn’t have to work
- Denial: It doesn’t scale
  - Only from 100 to 100,000,000

The Future: Motherhood Challenges

- Correctness
- Scaling
- Parallelism
- Reuse
- Trustworthiness
- Ease of use

Jim Gray’s challenges

1. The Turing test: win the impersonation game 30% of the time.
   - Read and understand as well as a human.
   - Think and write as well as a human.
2. Hear and speak as well as a person: speech↔text.
3. See and recognize as well as a person.
4. Remember what is seen and heard; quickly return it on request.
5. Answer questions about a text corpus as well as a human expert. Then add sounds, images.
6. Be somewhere else: observe (tele-past), interact (tele-present).
7. Devise an architecture that scales up by 10^6.
8. Programming: Given a specification, build a system that implements the spec. Do it better than a team of programmers.
9. Build a system used by millions, administered by ½ person.
   - Prove it only services authorized users.
   - Prove it is almost always available: (out < 1 second / 100 years)

A Grand Challenge:

Reduce highway traffic deaths to zero

- A pure computer science problem
- Needs
  - Computer vision
  - World models for roads and vehicles
  - *Dealing with uncertainty* about sensor inputs, vehicle performance, changing environment
  - *Dependability*

What is dependability?

- Formally, the system meets its spec
  - We have the theory needed to show this formally
  - But doing it doesn’t scale
  - And worse, we can’t get the formal spec right
    - Though we can get partial specs right
    - “Sorry, can’t find any more bugs.”
- Informally, users aren’t surprised
  - Depends on user expectations
    - Compare 1980 AT&T with cellphones
    - How well does the market work for dependability?

How much dependability?

- How much do we have? It varies
  - As much as the market demands
    - Is there evidence of market failure?
  - Almost any amount is possible
    - If you restrict the aspirations
    - In other words, there’s a tradeoff
- How much do we need? It varies
  - But safety-critical apps are growing fast
  - What’s the value of a life? Wild inconsistency
    - Look at British railways
- Dependable vs. secure

Measuring dependability

- Probability of failure
  - From external events
  - From internal malfunction
    - complexity (LOC ☺) × good experience (testing etc.)
- Cost of failure
  - Injury or death
  - External damage
    - Business interruption
    - Breakage
    - Bad PR
  - TCO
- What’s the budget? Who gets fired?

Dependability through redundancy?

- Good in its place
- But need independent failures
  - Can’t usually get it for software
    - Example: Ariane 5
  - Even harder for specs
    - *The unavoidable price of reliability is simplicity*—Hoare
- And a way to combine the results

Dependable ⇒ No catastrophes

- A realistic way to reduce aspirations
  - Focus on what’s *really* important
- What’s a catastrophe?
  - It has to be *very* serious
  - Must have some numeric measure
    - Dollars, lives? Say $100B, 1000 for terrorism
    - Less controversial: Bound it by size of CCB
- Must have a “threat model”: what can go wrong
  - Probabilities must enter
  - But how?

Examples of catastrophes

- USS Yorktown
- Terac 25 and other medical equipment
- Loss of crypto keys
- Destruction of big power transformers

- Are there any computer-only catastrophes?

Misleading examples of catastrophes

- Avionics, nuclear reactors
  - Most attention has gone here
  - But they are atypical
    - Lots of stuff has to work
    - Shutdown is impossible or very complex
- Impossible goals
  - Never lose a life.
    - Maybe OK for radiation
    - No good for driving
  - No terrorist incidents
  - No downtime

Catastrophe prevention that hasn't worked

- Trusted computing base for security
- Electric power grid
- Air traffic control
  - The spec said 3 seconds down/year/workstation

Architecture — Catastrophe Mode

- Normal operation vs. catastrophe mode
  - Catastrophe mode $\Rightarrow$ high assurance CCB
- Catastrophe mode requires
  - Clear, limited goals = limited functionality
    - Hence easier than security
  - Strict bounds on complexity
    - Less than 50k lines of code?
- Catastrophe mode is not a retrofit

Catastrophe mode

- What it does
  - Hard stop (radiation therapy)
    - Might still require significant computing
  - Soft stop (driving a car)
    - Might require a lot of the full functionality, but the design center is very different
  - Drastically reduced function (ship engines)
- How it does it
  - Take control, by reboot or hot standby
  - Censor (no radiation if limits exceeded)
  - Shed functions

Techniques

- Reboot—discard corrupted state
- Shed load
- Shed functions
- Isolate CCB, with minimal configuration

- Transactions with acceptance test
  - Approval pages for financial transactions
- Undo and rollback
- Well-tested components
  - Unfortunately, successful components are very big

Learning from security

- Perfection is not for this world
  - The best is the enemy of the good
  - Set reasonable goals
- Dependability is not free
  - Customers can understand tradeoffs
  - Though perhaps they undervalue TCO
- Dependability is holistic
- Dependability is fractal

Dealing with Uncertainty

- Unavoidable in dealing with the physical world
  - Need good models of what is possible
  - Need boundaries for the models
- Unavoidable for “natural” user interfaces: speech, writing, language
  - The machine must guess; what if it guesses wrong?
- Goal: see, hear, speak, move as well as a person. Better?
- Teach as well as a person?

Example: Speech “Understanding”

- Acoustic input: waveform (speech + noise)
- “Features”: compression
- Phonemes
- Words: dictionary
- Phrases: Language model
- Meaning: Domain model

Uncertainty at each stage.

Example: Robots

- Where am I?
- What is going on?
- What am I trying to do?
- What should I do next?
- What happened?

Paradigm?: Probability Distributions

- Could we have distributions as a standard data type?
  - Must be parameterized over the domain (like lists)
- What are the operations?

- Basic problem (?): Given distribution of *x*, compute distribution of *f*(*x*).
  - Hard when *x* appears twice in *f* – independence

Conclusions for Engineers

- Understand Moore’s law
- Aim for mass markets
  - Computers are *everywhere*
- Learn how to deal with uncertainty
- Learn how to avoid catastrophe