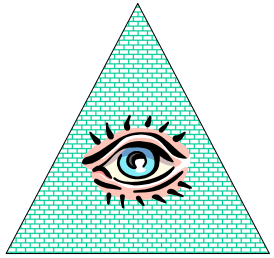
# Introspective Computing



John Kubiatowicz
UC Berkeley
ISAT Study 6/18, Stanford

# Today: building materials prevalent

- Originally: worried about squeezing the last ounce of performance from limited resources
- Today: worried about an abundance (embarrassment) of riches?
  - Billions of transistors on a chip (17nm Yeah!)
  - Microprocessor Report articles wondering if all the lessons of RISC are now irrelevant
- Moore's laws: exponential growth of everything
  - Transistors, Performance, Disk Space, Memory Size
- So, what matters any more????



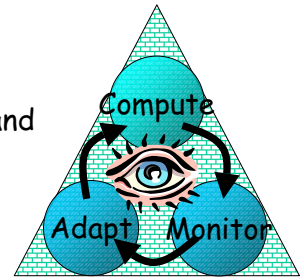
# Simple answer: Performance is the wrong metric

- Complexity:
  - more than 50% of design teams now for verification
- Power
  - Processor designs hampered in performance to keep from melting
  - Why 3 or 4 orders of magnitude difference in power consumption between custom hardware and general Von Neuman architectures?
- Energy
  - Portable devices
- Scalability, Reliability, Maintainability
  - How to keep services up 24x7?



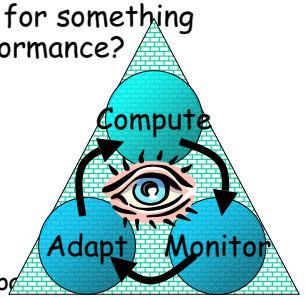
# The Biological Inspiration

- Biological Systems are built from (extremely) faulty components, yet:
  - They operate with a variety of component failures ⇒ Redundancy of function and representation
  - They have stable behavior ⇒ Negative feedback
  - They are self-tuning ⇒ Optimization of common case
- Introspective Computing:
  - Components for computing
  - Components for monitoring and model building
  - Components for continuous adaptation



# The New Architectural Creed

- Question: Can we use Moore's law gains for something other than just raw computational performance?
- Examples:
  - Online algorithmic validation
  - Model building for data rearrangement
    - Availability
    - Better prefetching
  - Extreme Durability (1000-year time scale?)
    - Use of erasure coding and continuous repa
  - Stability through Statistics
    - Use of redundancy to gain more predictable behavior
    - *Systems version of Thermodynamics!*
  - Continuous Dynamic Optimization of other sorts






# Why a dynamic loop?

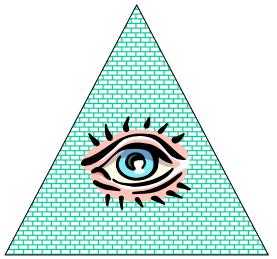
- The world is unpredictable:
  - Manufacturing imperfections not uniform
  - Demand for resources varies widely
  - Environmental conditions change without warning
- Computation is too complex to predict:
  - Modern systems are chaotic!
    - Extreme sensitivity to initial conditions
    - Timing, Humans, Data-dependent behavior
    - Computation spread over globe – initial conditions never same!
  - Even writers of software surprised by behavior
- Ideal place to compute: at chaotic boundary
  - Lower voltages until errors in mid range of tolerance
  - Keep data and algorithmic redundancy at a point that is "sufficient but not excessive"
  - Communication resources tuned to surpass noise
  - Keep heat generation within tolerances



# Side comment: The network is the computer

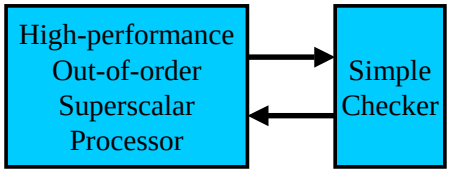
- Modern uses of computation are distributed:
  - Data produced at one site used many places in world
  - Global-scale sources of information
  - Global-scale computation (SETI@home, Supercomputer centers, etc)
- On-chip environment is reasonably stable, but… Subject to single points of failure:
  - Mechanical faults destroy all components on chip
  - Power outages/natural disasters affect all components in area
  - Denial of service and compromises tend to take out all components in same administrative domain
- Advantages of full geographic distribution:
  - Data may be encoded and spread widely
  - Hot spares available even with massive failure in one location
  - Permanent history of computation may be kept for later use! (OceanStore, etc)





# Validation

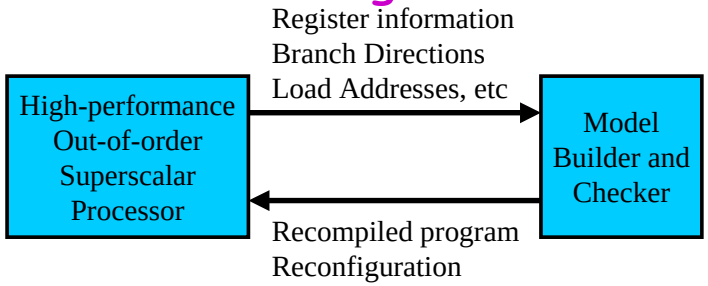
# Online Hardware Verification: DIVA (Todd Austin)




- OOO Superscalar coupled with checker
- Checker receives:
  - Ordered instructions + results
    - Check results of each instruction independently
    - Check data flow through registers quickly, in order
  - Checker is simpler and hence easier to design correctly
- Catches many types of errors in Superscaler
  - Transient/Permanent hardware failures
  - Design flaws



# Online Validation: Coarser grained




- Continuous checking of invariants
  - User supplied? Hopefully not
  - Compiler generated? (Consider proof carrying code)
- Building Markov models of correct behavior
  - Basic block frequencies
  - Result distribution
  - Previous models kept in "permanent" storage?



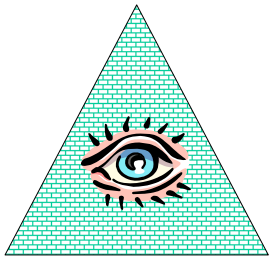
# Quick Aside: Proof Carrying Codes (Necula)

- Technique which annotates execution binaries with "Proofs":
  - These prove that certain invariants must be true
    - Memory accesses never exceed bounds
    - Loop indices have certain properties
    - Other loop invariants remain true
  - Cool property: Independent of optimization!
    - Proof on topology of computation, not specific instance
  - Original use: prove that code is safe to remote server
    - Intended for internet environment
- Introspective use?
  - Watch execution trace
    - compress with grammar extractor (e.g. SEQUITUR, Larus)
  - Check that invariants of code still hold



# Online Dynamic Compilation: e.g. Transmeta

- Used for backward compatibility with x86
  - Advantages: hardware architecture can change
- Consider as introspective system:
  - Quick compilation to "get it right"
  - Continuous on-line optimization
    - Analysis of execution to extract common paths
    - Extraction of runtime constants
    - Recompilation to incorporate changes
  - Note: OOO architectures doing very primitive compiler transformations (dataflow analysis, reordering)
- Correctness:
  - Why not use techniques like proof-carrying codes to allow advanced optimization to be a bit "fast and loose"?
  - Optimizations could be "not quite correct" or even generated via genetic algorithms!



# Performance



# Prefetching Linked Structures

```
loop: ld   r1, 4(r2)
      add  r7,r1,r0
      ld   r2, 0(r2)
      bnz  r2, loop
```

Register Assoc Table

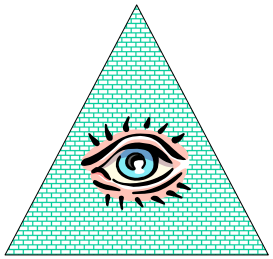
r1: loop
r2: loop+4
r3: …..

Adder Assoc (Hash Table)

loop: Addr1
loop+4: Addr2

- Building Model of complex linked data structures:
  - A "Link" is (base-load inst, offset)
    - For our example: AAAAA
    - For depth-first tree search: AAABBABBAABBABB
  - Build grammar to describe this
  - Recognize reuse of grammars
- Can be used for prefetching:
  - Once pattern recognized, separate processor can use grammar to run ahead of processor
  - 30% hit rate for prefetcher for some SPEC95
    - Still in preliminary stages (Not yet performance)
- Other possibilities:
  - Better branch prediction/etc



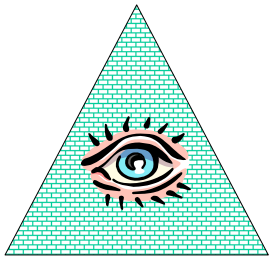


# Power/Energy



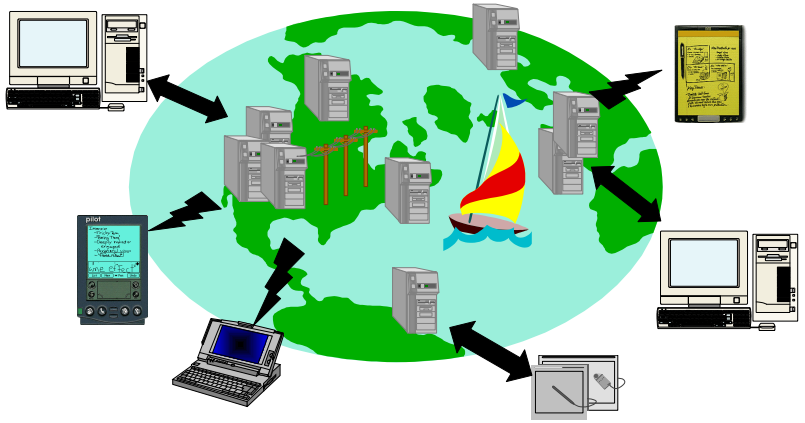
# Introspection to reduce Energy?

- Sure
  - Use introspection to adjust operating point (voltage)
  - Use models of behavior to shut off circuits
- More interesting: Re-coding of computation
  - 3 orders of magnitude between specialized hardware solution and low-power processor implementation
  - Von-Neuman energy penalty?
- Or: encoding to reduce transitions on buses?
  - Trying to reduce total *number* of transitions
  - Do so in a way that has net Energy savings
    - Use transition coding
    - Assign low-weight code to frequent values/transitions
  - Can reduce transitions by 70% on Spec



# Extreme Durability





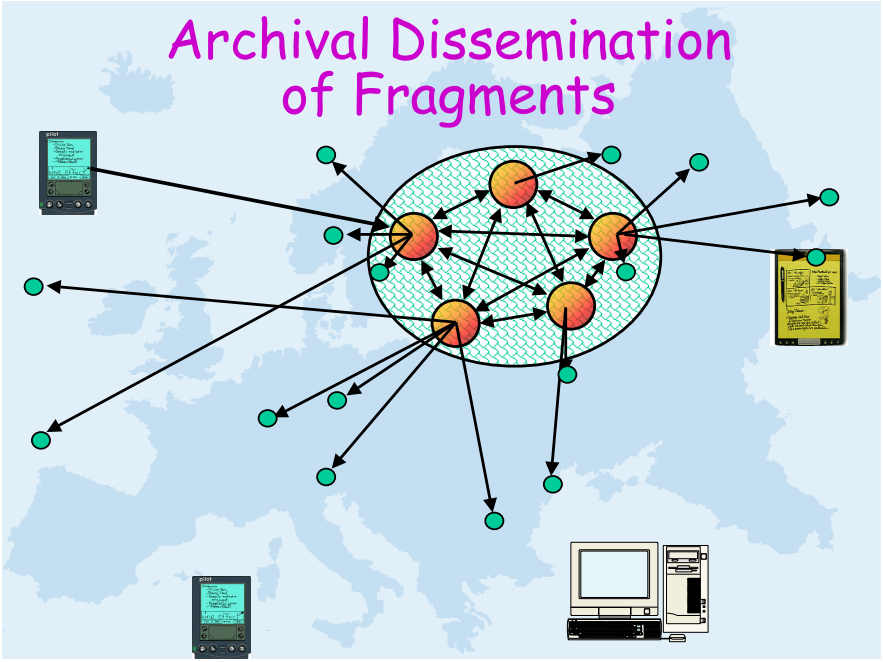
# OceanStore

## Global-Scale Persistent Storage

# Deep Archival Storage

- Version based storage
  - Every update creates a new version
  - Keep every version for all time!
  - Copy-on-write methodology for efficiency
- "Time-travel" possible
  - Can always back up to previous version
  - Basic component of maintainable system: undo mistakes
- Achieve 1000 year hedge against decay?
  - Wide-spread replication for fault tolerance
    - m-of-n coding (Erasure codes)
  - Continuous repair to combat entropy



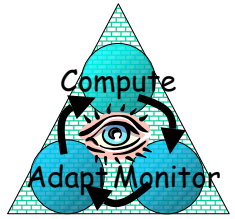


# Automatic Maintenance

- Erasure coding extremely efficient at preservation/reconstruction ("Holographic")
- OceanStore Routing Infrastructure (Tapestry) knows every location of fragment
  - Permits infrastructure to notice:
    - Servers going away for a while
    - Or, going away forever!
- Continuous sweep through data
  - Simple calculation of 1 mole repair
    - Rate ¼ , 10 billion servers, 64 fragments,
    - 1 Mbit/sec bandwidth



# Conclusion:




- Introspective Computing
  - Devote resource for continuous:
    - Monitoring, model building, adaptation, repair
  - Goals?
    - Availability and Fault Tolerance
    - Stability of performance
    - Operating at Chaotic boundary
    - Tolerance to Byzantine faults and denial of service
  - Architecture?
    - Special instruction sets for monitoring/analysis
    - Transactions at all levels
    - CMP + very efficient FIFO Communication paths