Center for Hybrid and Embedded Software Systems

College of Engineering, University of California at Berkeley
Presented by: Edward A. Lee, EECS, UC Berkeley

Citris Founding Corporate Members Meeting, Feb. 27, 2003
Davis, California

Board of Directors
Tom Henzinger, tah@eecs.berkeley.edu
Edward A. Lee, eal@eecs.berkeley.edu
Alberto Sangiovanni-Vincentelli, alberto@eecs.berkeley.edu
Shankar Sastry, sastry@eecs.berkeley.edu

Other key faculty
Alex Aiken, aiken@eecs.berkeley.edu
Dave Auslander, dma@me.berkeley.edu
Ruzena Bajcsy, ruzena@eecs.berkeley.edu
Karl Hedrick, khedrick@me.berkeley.edu
Kurt Keutzer, keutzer@eecs.berkeley.edu
George Necula, necula@eecs.berkeley.edu
Masayoshi Tomizuka, tomizuka@me.berkeley.edu
Pravin Varaiya, varaiya@eecs.berkeley.edu

Chess

Hybrid & Embedded Software Systems

- Computational systems
  - but not first-and-foremost a computer
- Integral with physical processes
  - sensors, actuators
- Reactive
  - at the speed of the environment
- Heterogeneous
  - hardware/software, mixed architectures
- Networked
  - adaptive software, shared data, resource discovery

cellular phones

Mission of Chess

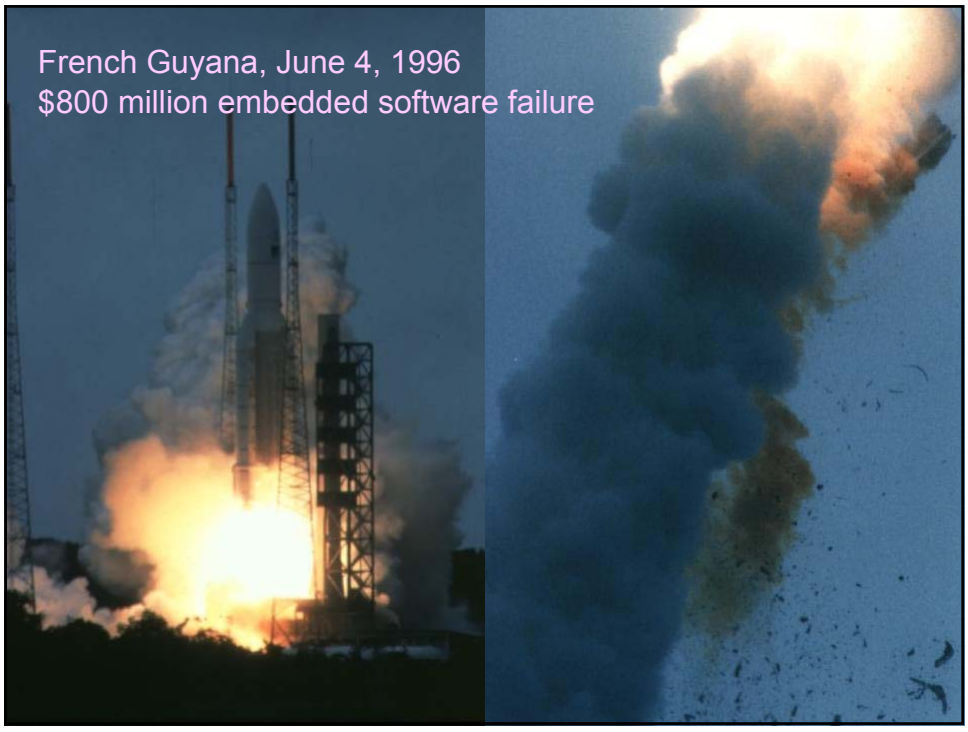
To provide an environment for graduate research on the design issues necessary for supporting next-generation embedded software systems.

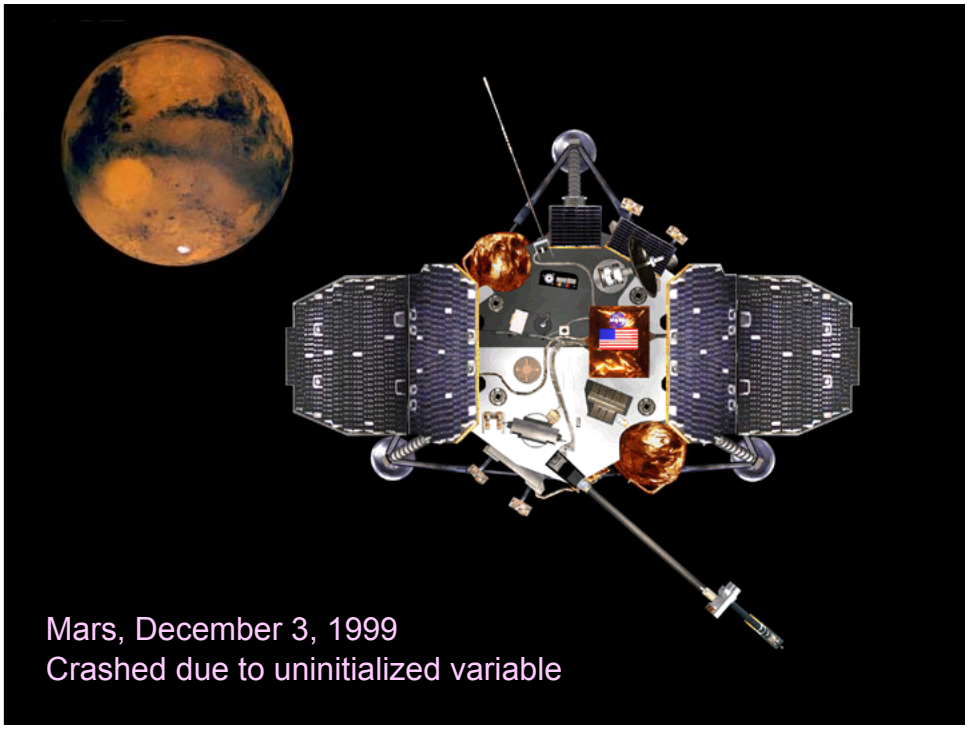
- Model-based design
- Tool-supported methodologies

For

- Real-time
- Fault-tolerant
- Robust
- Secure
- Heterogeneous
- Distributed

Software

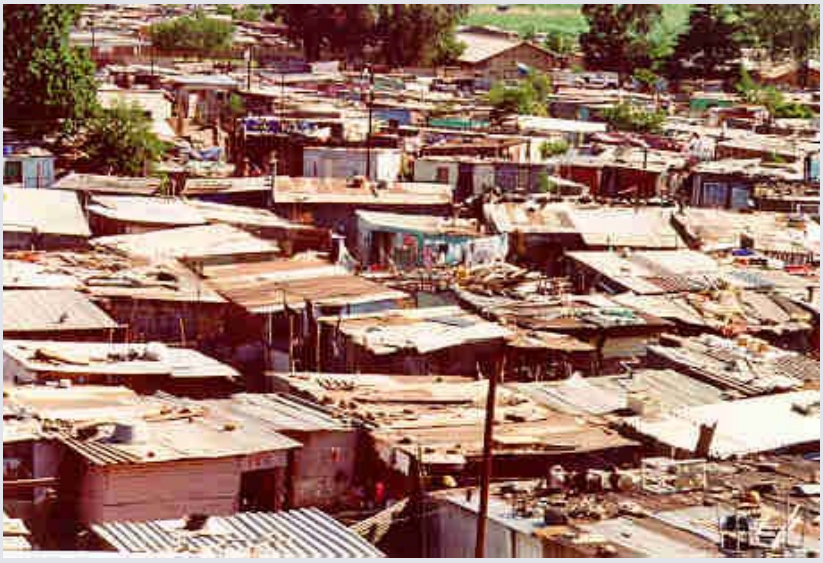




Mars, December 3, 1999
Crashed due to uninitialized variable

$4 billion development effort
40-50% system integration & validation cost

Embedded Software Architecture Today



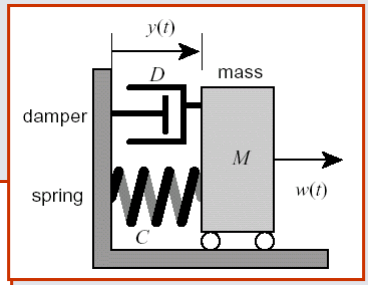
Embedded Software Architecture Tomorrow

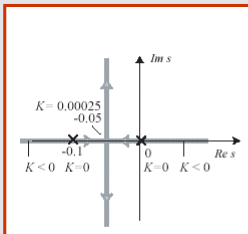
The Goal

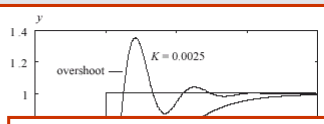
- To create a modern computational systems science and systems design practice with
  - Concurrency
  - Composability
  - Time
  - Hierarchy
  - Heterogeneity
  - Resource constraints
  - Verifiability
  - Understandability

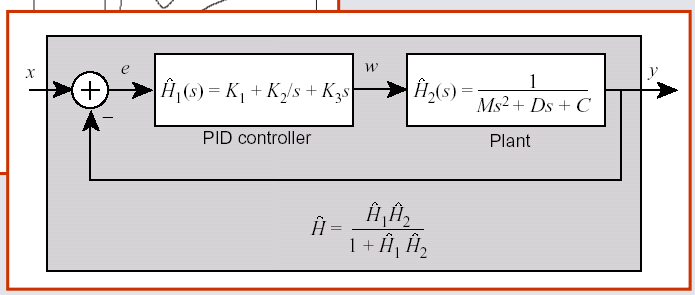
A Traditional Systems Science – Feedback Control Systems

- Models of continuous-time dynamics
- Sophisticated stability analysis
- But not accurate for software controllers



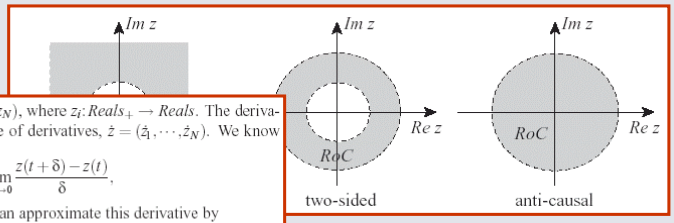






Discretized Model – A Step Towards Software

- Numerical integration techniques provided sophisticated ways to get from the continuous idealizations to computable algorithms.
- Discrete-time signal processing techniques offer the same sophisticated stability analysis as continuous-time methods.
- But it's *still* not accurate for software controllers



In general, z is an N-tuple, $z = (z_1, \cdots, z_N)$, where z_i: $Reals_+ \rightarrow Reals$. The derivative of an N-tuple is simply the N-tuple of derivatives, $\dot{z} = (\dot{z}_1, \cdots, \dot{z}_N)$. We know from calculus that

$$\dot{z}(t) = \frac{dz}{dt} = \lim_{\delta \to 0} \frac{z(t+\delta) - z(t)}{\delta},$$

and so, if $\delta > 0$ is a small number, we can approximate this derivative by

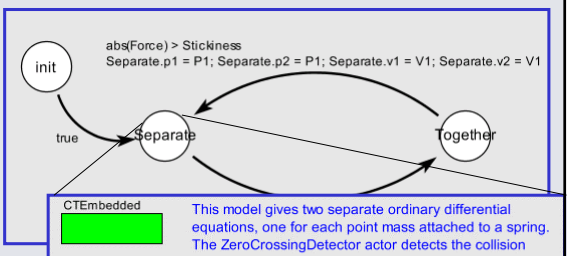
$$\dot{z}(t) \approx \frac{z(t+\delta) - z(t)}{\delta}.$$

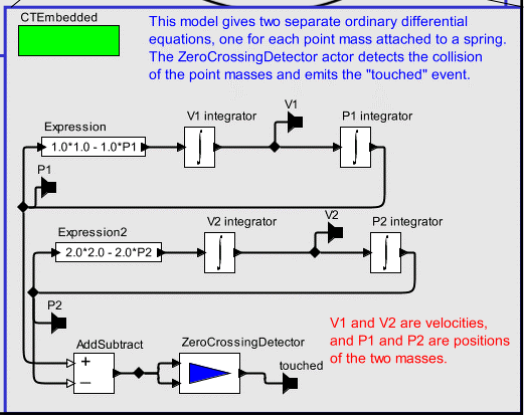
Using this for the derivative in the left-hand side of (5.50) we get

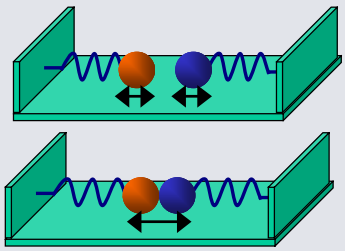
$$z(t+\delta) - z(t) = \delta g(z(t), v(t)). \qquad (5.51)$$

Hybrid Systems – Reconciliation of Continuous & Discrete

- UCB researchers have contributed hugely to the theory and practice of blended discrete & continuous models.
- But it's *still* not accurate for software controllers



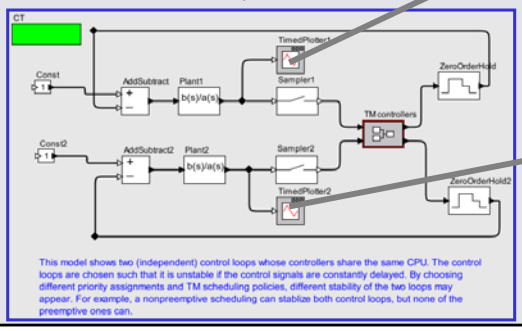


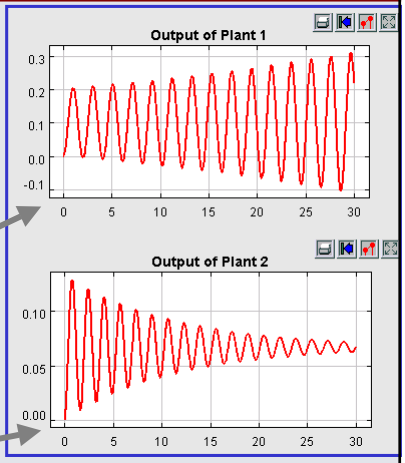


Timing in Software is More Complex Than What the Theory Deals With

An example, due to Jie Liu, models two controllers sharing a CPU under an RTOS. Under preemptive multitasking, only one can be made stable (depending on the relative priorities). Under non-preemptive multitasking, both can be made stable.

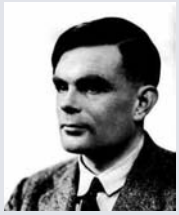
Where is the theory for this?





How Safe is Our Real-Time Software?

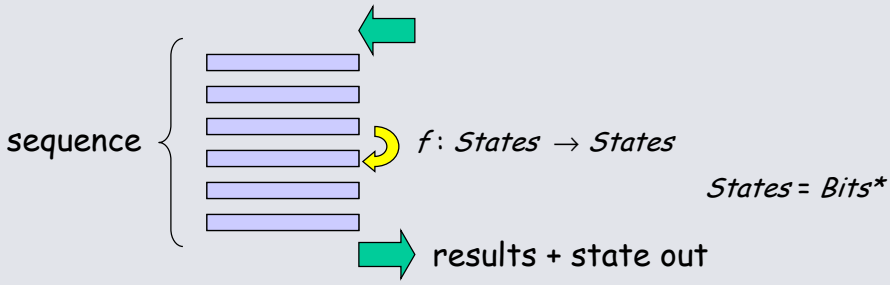
Another Traditional Systems Science - Computation, Languages, and Semantics



Alan Turing

Everything "computable" can be given by a terminating sequential program.

- Functions on bit patterns
- Time is irrelevant
- Non-terminating programs are defective



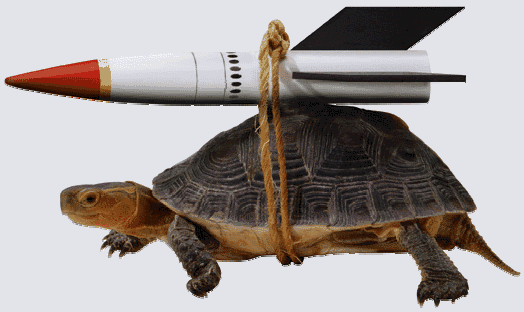
Current fashion – Pay Attention to "Non-functional properties"

- Time
- Security
- Fault tolerance
- Power consumption
- Memory management

But the formulation of the question is very telling:

How is it that *when* a braking system applies the brakes is any less a *function* of the braking system than *how much* braking it applies?

What about "real time"?



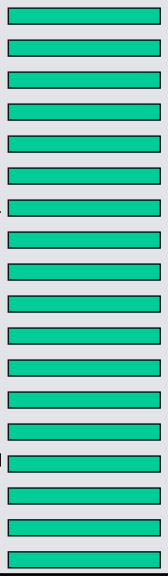
Make it faster!

Processes and Process Calculi

Infinite sequences of state transformations are called "processes" or "threads"

incoming message ⇨

outgoing message ⇦



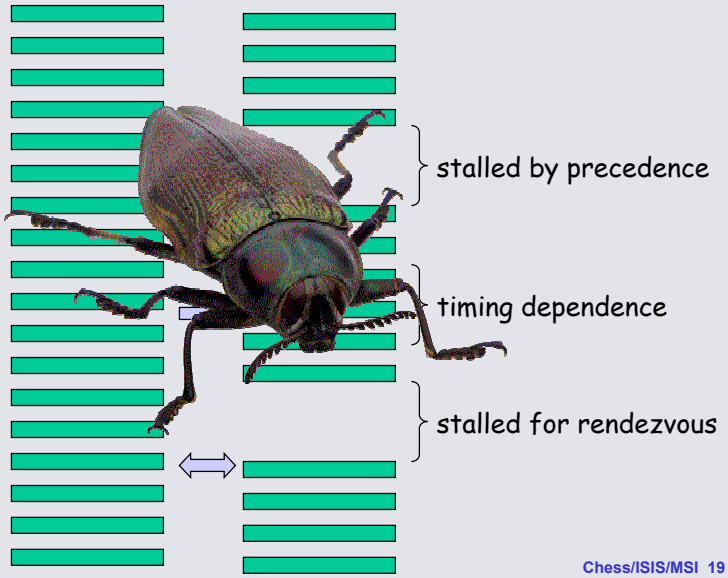
Various messaging protocols lead to various formalisms.

In prevailing software practice, processes are sequences of external interactions (total orders).

And messaging protocols are combined in ad hoc ways.

Prevailing Practice in Embedded Software – Interacting Processes

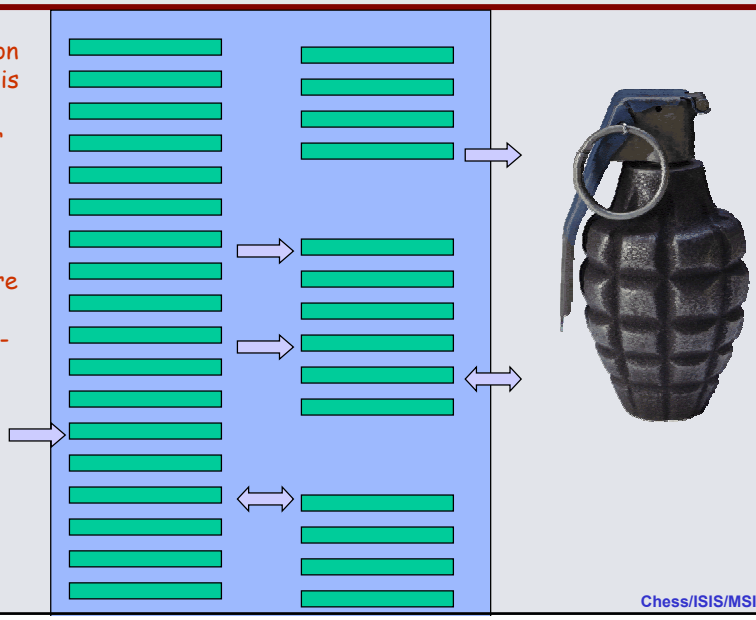
Software realizing these interactions is written at a very low level (semaphores and mutexes). *Very* hard to get it right.



Interacting Processes – Not Compositional

An aggregation of processes is not a process (a total order of external interactions). What is it?

Many software failures are due to this ill-defined composition.



Compositionality

Non-compositional formalisms lead to very awkward architectures.

Real-Time Multitasking?

Prioritize and Pray!

Promising Alternatives

- Synchronous languages (e.g. Esterel)
- Time-driven languages (e.g. Giotto)
- Hybrid systems
- Timed process networks
- Discrete-event formalisms
- Timed CSP

We are working on interface theories and meta models that express dynamic properties of components, including timing.

Current Research Focus Areas

- Interfaces theories for component-based design
- Meta-modeling (models of modeling strategies)
- Principles of actor-oriented design
- Software architectures for actor-oriented design
- Automotive systems design
- Avionics systems design
- Virtual machines for embedded software
- Semantic models for time and concurrency
- Design transformation technology (code generation)
- Visual syntaxes for design
- Application-specific processors

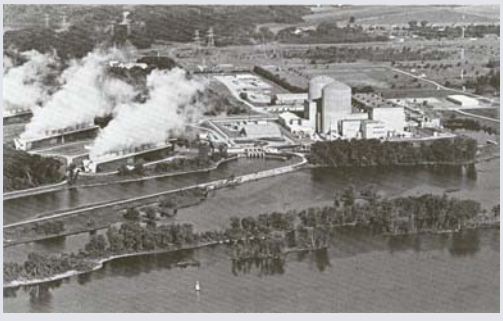
• Mobies	• Ptolemy
• SEC	• Mescal
• ISIS	• Metropolis
• Giotto	• Bear

Application Inspired by 9/11

Drawing by a 5 year old made on 9/11/01

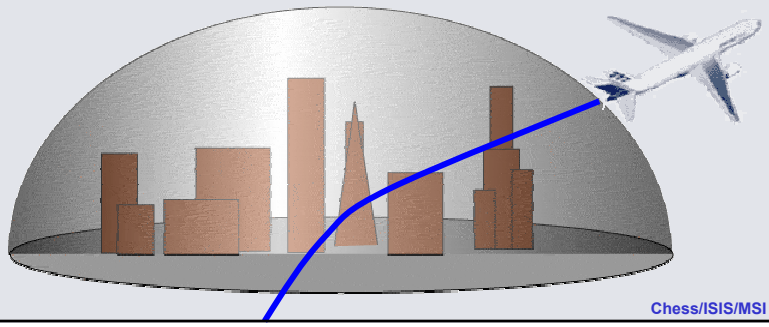
Need to Shield

- Major cities
- Government centers
- Chemical and nuclear plants
- Military installations
- Critical infrastructure



Softwalls Project

- Carry on-board a 3-D database with "no-fly-zones"
- Enforce no-fly zones using on-board, non-networked avionics
- This is a hybrid system with extreme safety requirements
- Rigidity/brittleness of existing software is a major impediment



Impact on Education – Intellectual Groupings in EECS

Multimedia
Robotics, Vision
Discrete-event systems
Simulation
Real-time systems
Concurrent software
Networks

Communications
Information theory
Queueing theory
Signal processing

EIS

Linear systems
Control
Nonlinear systems

CS

Languages
Complexity
Automata
Software engineering
Compilers
Operating systems
Algorithms
Graphics
User interfaces
Databases
Artificial Intelligence

EE

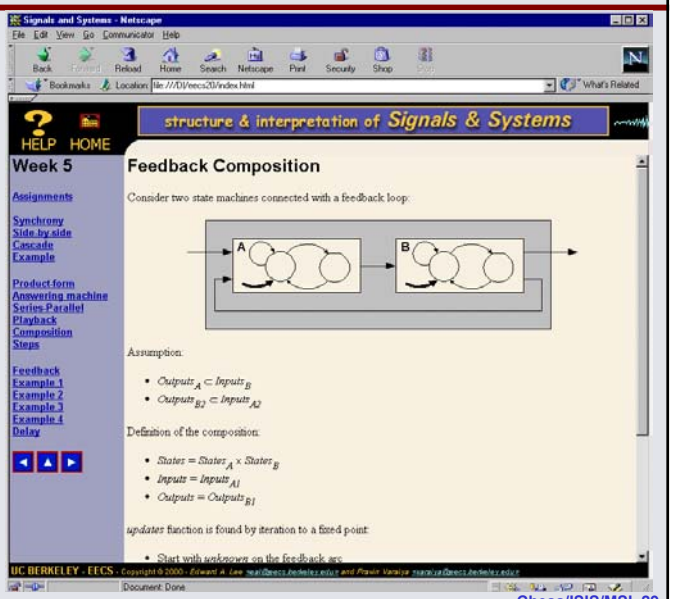
Circuits
Electronics
Devices
Process technology
E & M
Power systems
Plasmas
Quantum & Optical

Architecture
CAD for VLSI
Configurable systems

Education Changes – The Starting Point

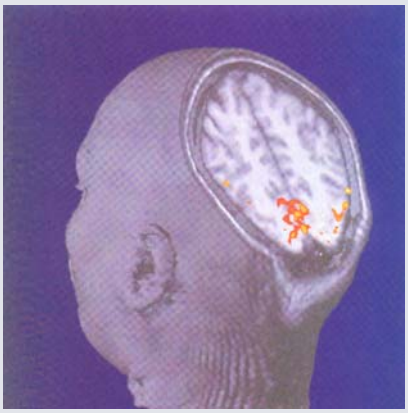
Berkeley has a required sophomore course that addresses mathematical modeling of signals and systems from a computational perspective.

The web page at the right illustrates a broad view of feedback, where the behavior is a fixed point solution to a set of equations. This view covers both traditional continuous feedback and discrete-event systems.



Themes of the Course

- The connection between *imperative* and *declarative* descriptions of signals and systems.
- The use of *sets and functions* as a universal language for declarative descriptions of signals and systems.
- State machines and frequency domain analysis as complementary tools for designing and analyzing signals and systems.
- Early and often discussion of applications.



Brain response when seeing a discrete Fourier series.

Conclusion

We are on the line to build a *new system science* that is at once physical and computational.

It will form the foundation for our understanding of computational systems that engage the physical world.

And it will change how we teach, research and engineer systems.