



# The Past, Present, and Future of my "Python Optional" Physics Experiment

David M Beazley  
<https://www.dabeaz.com>  
@dabeaz

CWI Lectures, November 21, 2019



# CO-CONSPIRATORS

Peter Lomdahl

Tim Germann

Niels Grønbech-Jensen

Brad Holian

Shujia Zhou

Pablo Tamayo

In 1996, I modified  
Python to run on a  
supercomputer

This one →

1992

## FASTEST CONNECTION MACHINE

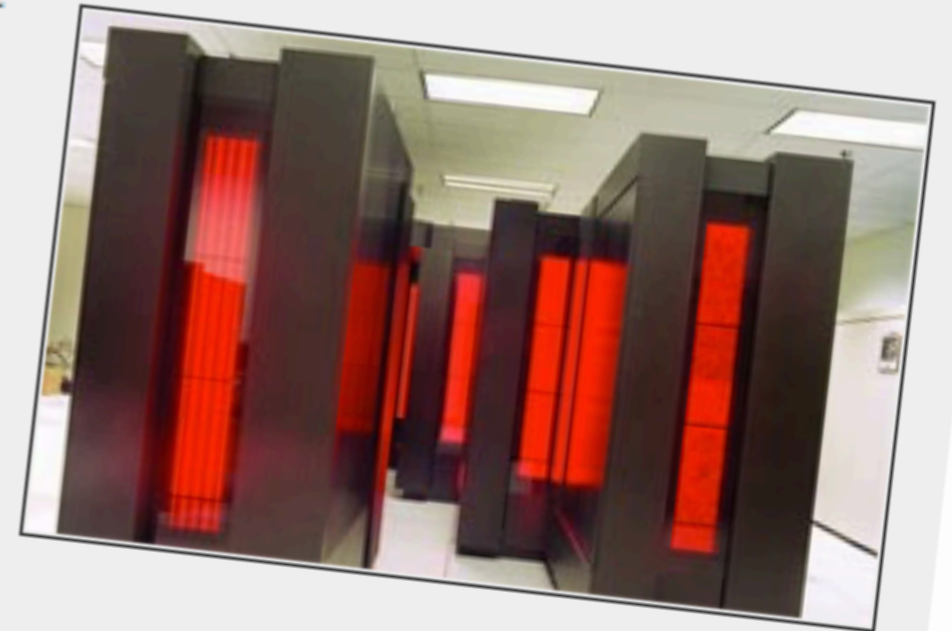
Installed at Los Alamos, CM-5 had a theoretical peak speed of 130 gigaflops, more than a factor of 1,000 over the Cray-1. The parallel supercomputer ran the most demanding algorithms.

### Speed is the thing

Featured in 1993's Jurassic Park, the CM-5 Connection Machine was a massively parallel supercomputer containing thousands of off-the-shelf microprocessors commonly used on high-end workstations.

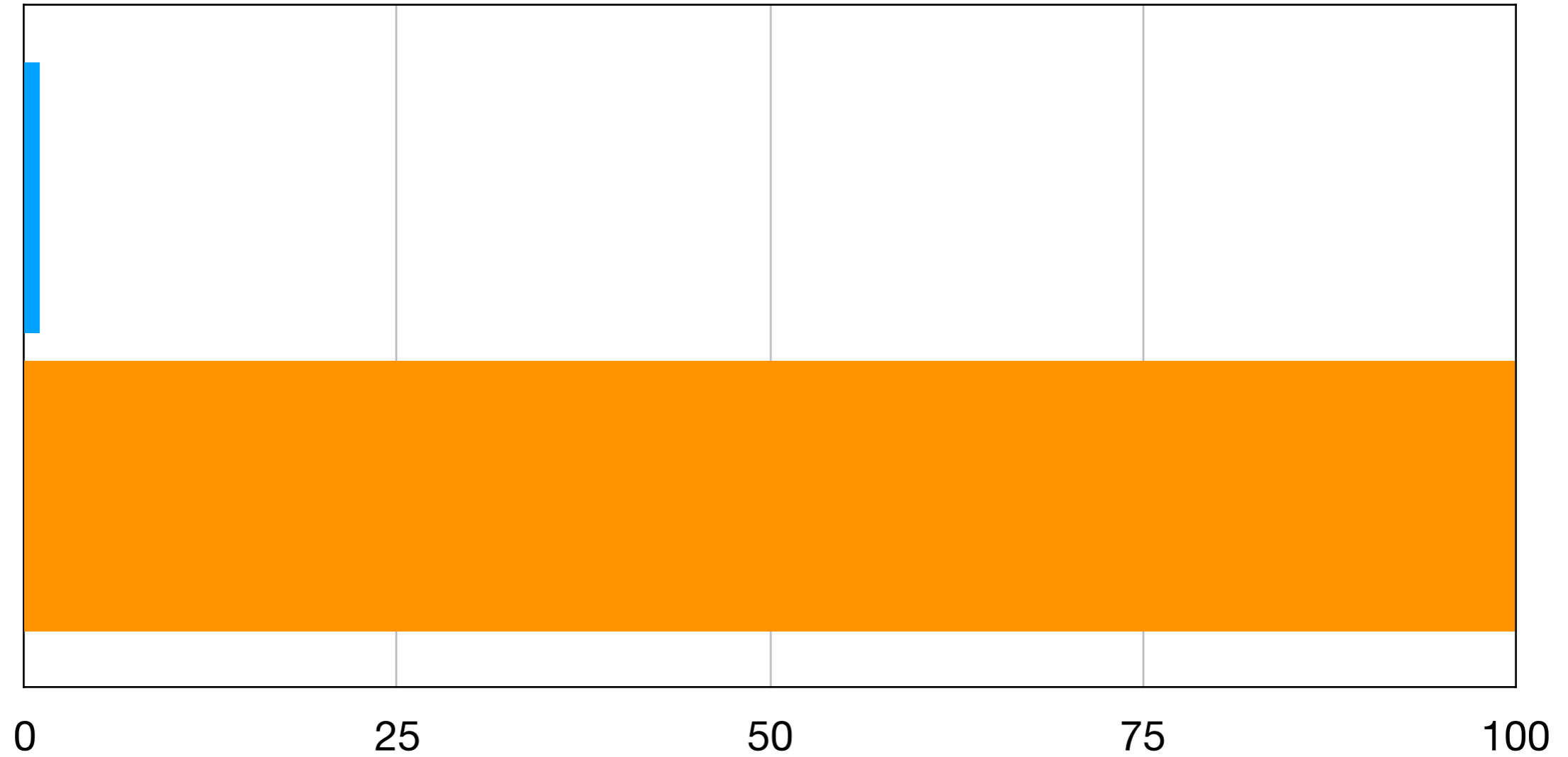
The processors were connected in parallel through various network architectures so that they could work simultaneously on different parts of a single, large problem.

Today, our supercomputers perform at speeds greater than a petaflop—that's one million billion operations per second.



WHY?!?!?

■ Python      ■ Quite Literally Everything Else [1]



**PERFORMANCE (FLOPS)**

**[1] Random Hacker Site Comment**

**"Surely not"**

**"Unfortunately, it seems that many of the efforts to develop tools and languages have sacrificed code performance in favor of portability or ease of use. [...] Compromising performance [...] seems unacceptable."**

**- Beazley & Lomdahl (1994)**

**Debugging and Performance Tuning for Parallel Computing Systems,  
IEEE Computer Society, (1996)**

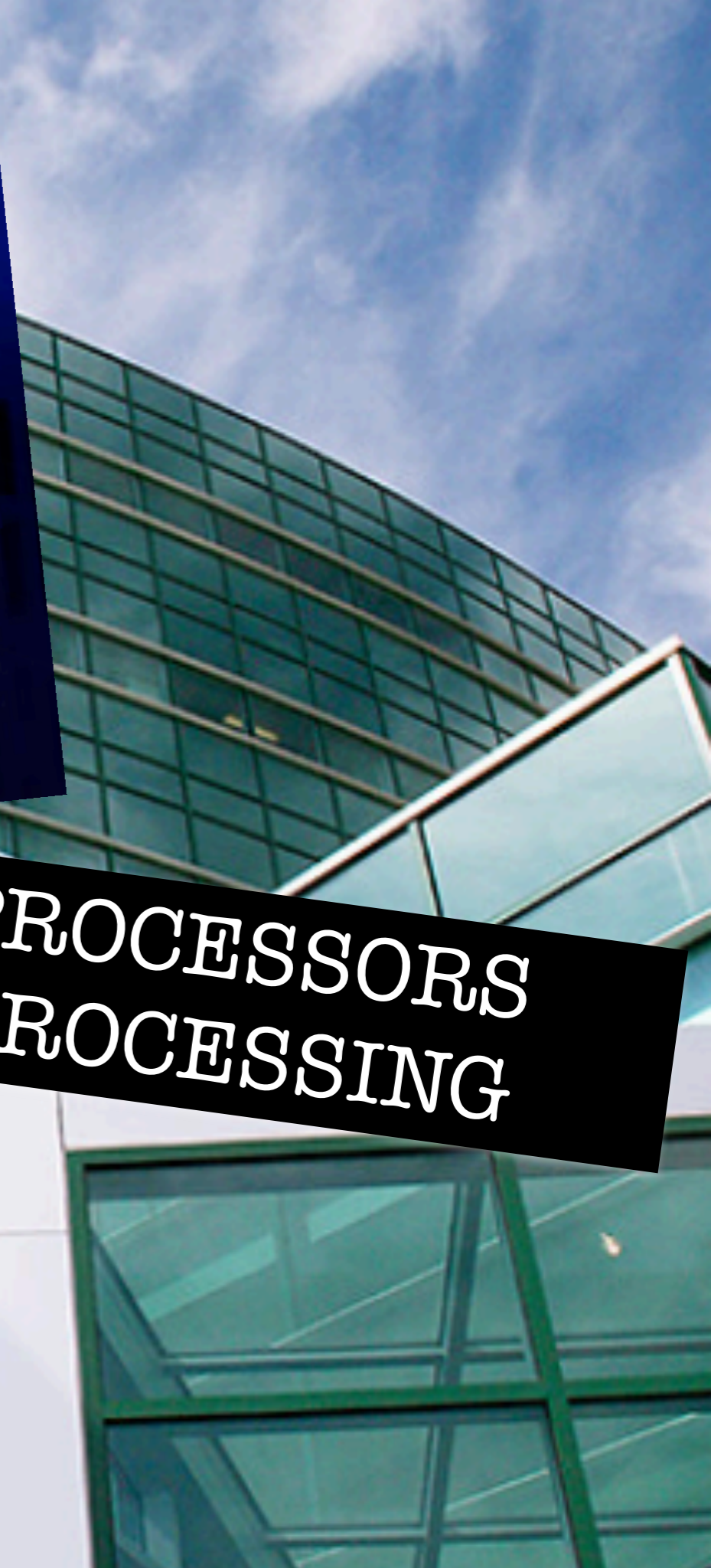


FIG. 10

It is seen at once from Fig. 10 that the motion is finite for  $E > 0$ .

SUPERCOMPUTER  
CENTERS,  
SHIMMERING





**PROCESSORS  
PROCESSING**



SERIOUS  
SCIENCE  
MISSIONS



# Science on a Grand Scale

*Through the Computing Grand Challenge Program, participants harness world-leading supercomputing and science to advance scientific discovery and Laboratory missions.*



Nevertheless... there it was.

Python running on a  
supercomputer.

```
$ python
```

```
Python 1.3 (Nov 21 1996) [GCC 4.8.4]
```

```
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
```

```
>>> print "Hello World"
```

```
Hello World
```

```
>>>
```



WHO LET THIS HAPPEN?!?!?





1951

## DAWN OF COMPUTING

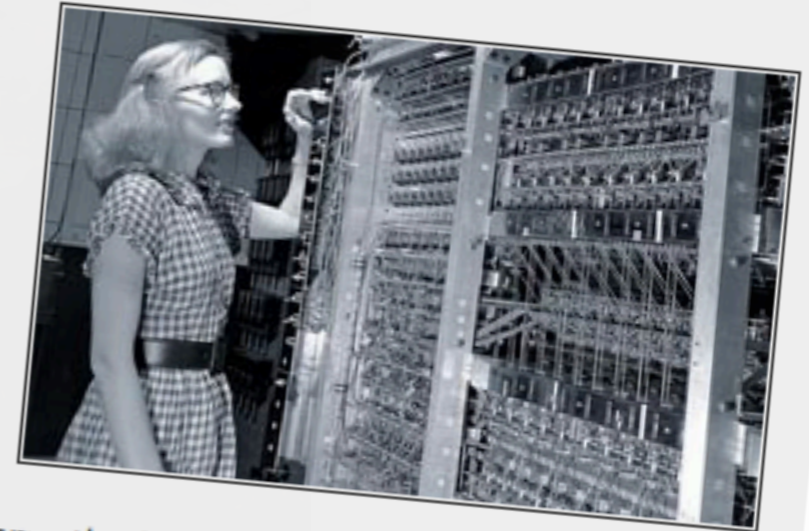
Computer development exploded during the war. One of the first electronic computers, MANIAC supported atomic energy research, solving hydrogen-bomb design problems.

### MANIAC

Nick Metropolis and John von Neumann built Los Alamos' MANIAC (mathematical analyzer, numerator, integrator, and computer), one of the first electronic digital computers.

Los Alamos played an important role developing major computing advances, such as parallel processing and cluster architecture.

In 2008, our Roadrunner computer became the first to break the petaflop barrier—one quadrillion floating-point operations per second—enabling scientists to model a complex phenomena including nuclear behavior, pandemics, supernovae and climate change.





**1951**

## **DAWN OF COMPUTING**

Computer development exploded during the war. One of the first electronic computers, MANIAC supported atomic energy research, solving hydrogen-bomb design problems.

**1974**

## **VECTOR SUPERCOMPUTER**

Los Alamos acquired its first vector supercomputer. Such supercomputers used vector processors that greatly improved performance on numerical simulations important to nuclear weapons research.

### **Supercharged simulations**

Vector Supercomputers use one-dimensional arrays of data called vectors that operate on single data items. Dominate through the 1990s, the vector led way to the microprocessor.

However, the vector design remains notable because it helped shape the computer industry to this day.

Application software ran on vector supercomputers included nuclear code, cryptanalytic code (cryptographic security systems), petroleum reservoir models, and climatological and meteorological models.





1951

## DAWN OF COMPUTING

Computer development exploded during the war. One of the first electronic computers, MANIAC supported atomic energy research, solving hydrogen-bomb design problems.

1974

## VECTOR SUPERCOMPUTER

Los Alamos acquired its first vector supercomputer. Such supercomputers used vector processors that greatly improved performance on numerical simulations important to nuclear weapons research.

1987

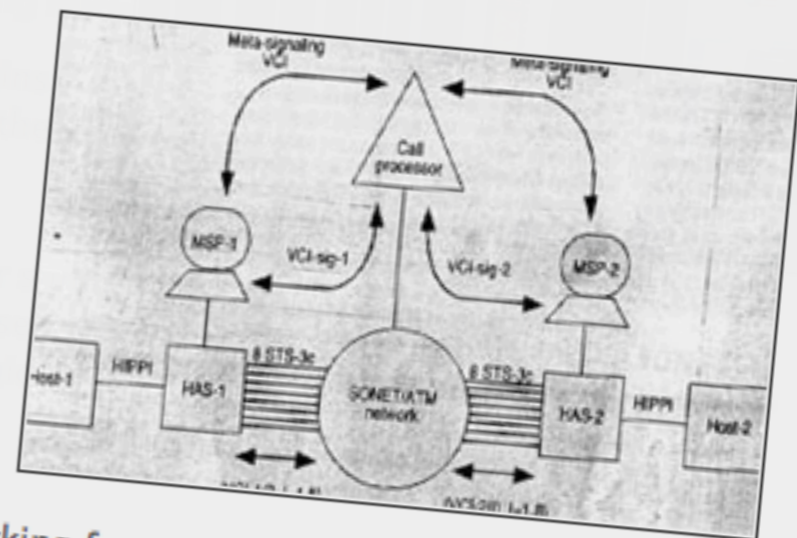
## CREATING SUPERCOMPUTERS

Los Alamos developed HIPPI, an interface that transmits large amounts of data and interconnects computers to perform as a supercomputer.

### High-performance parallel interface

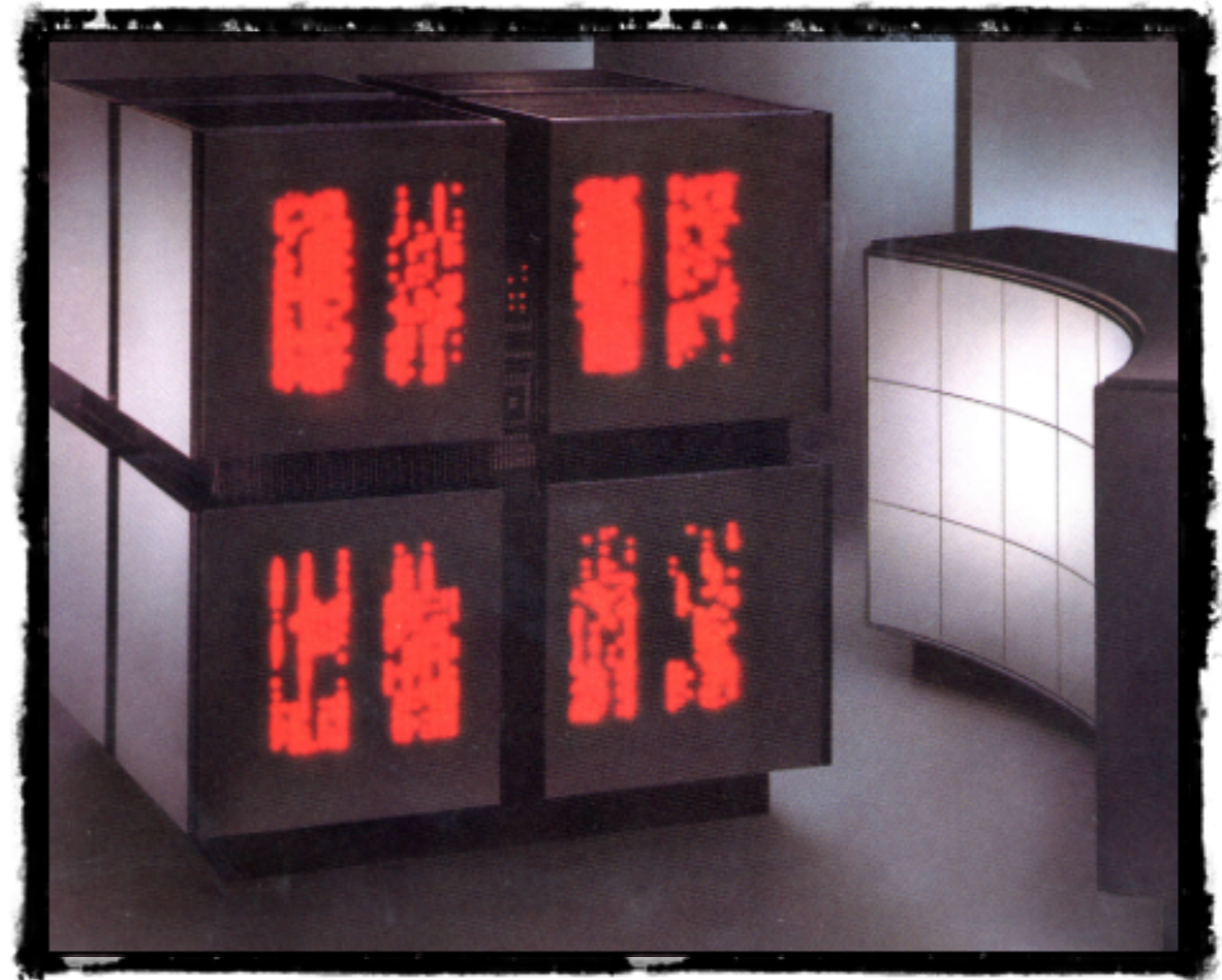
Our high-performance parallel interface (HIPPI) was the first "near-gigabit" standard for network data transmission.

By 1995, HIPPI was linked to a synchronous optical network (SONET) highway that transmitted computer data cross-country at 800 million bits per second, enabling research to perform combined computations. It provides high-speed, wide-area networking for commercial users who have computing sites at more than one location.



# Advanced Computing Laboratory (ACL)

*"In 1989, seeing the potential of new technology for addressing the "Grand Challenge" computational programs in science and engineering, Los Alamos set up the Advanced Computing Laboratory as a kind of **proving ground** for testing Massively Parallel Processors (MPPs) on real problems."*



Who uses a "proving ground?"

# THEORETICAL DIVISION

(CONDENSED MATTER PHYSICS GROUP)



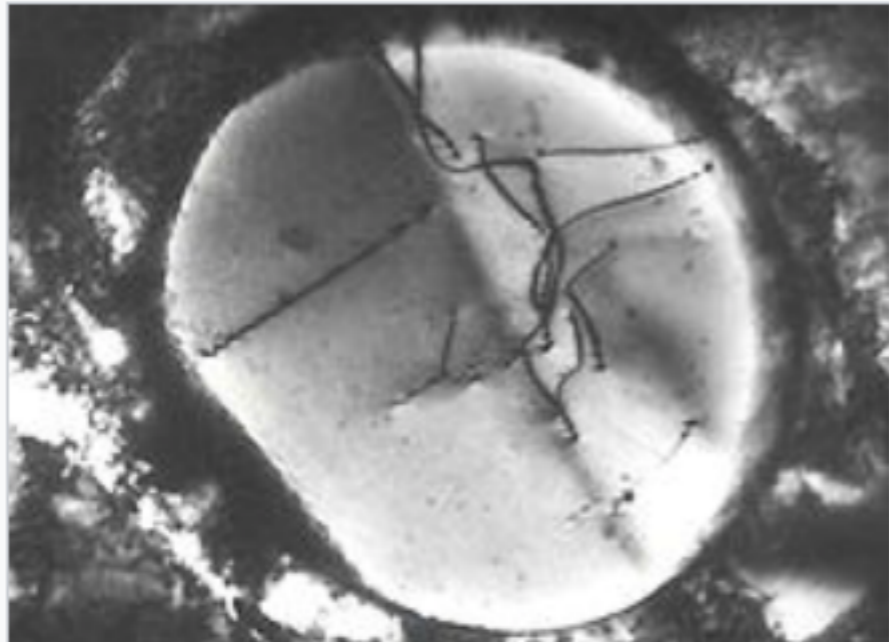
# MY DESK



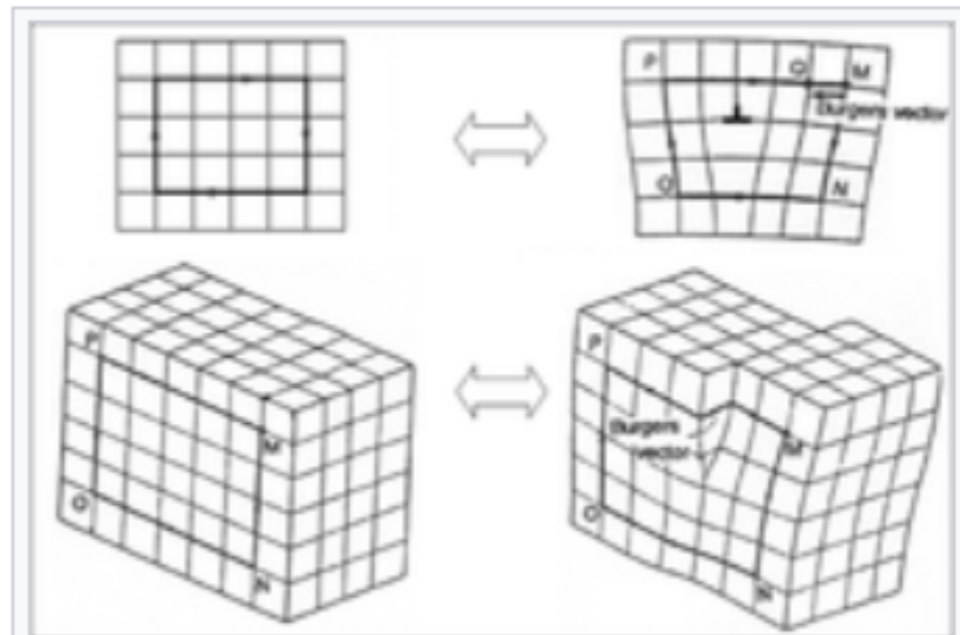


# THE CODE : SPaSM

- A "greenfield" project started in 1992
- Written in ANSI C
- Goal: Dislocation dynamics in materials



Transmission electron micrograph of dislocations



Top right: edge dislocation.  
Bottom right: screw dislocation.

- How? Short-range molecular dynamics

# THE PLAYERS

- Peter Lomdahl (Principal).

**Background: Electrical engineering, Mathematical Physics.  
Was also the Group Unix Systems Administrator**

- David Beazley (Ph.D Student).

**Background: Applied mathematics, numerical analysis  
Past job: Writing graphics device drivers in x86 for use in Modula-2**

- Brad Holian.

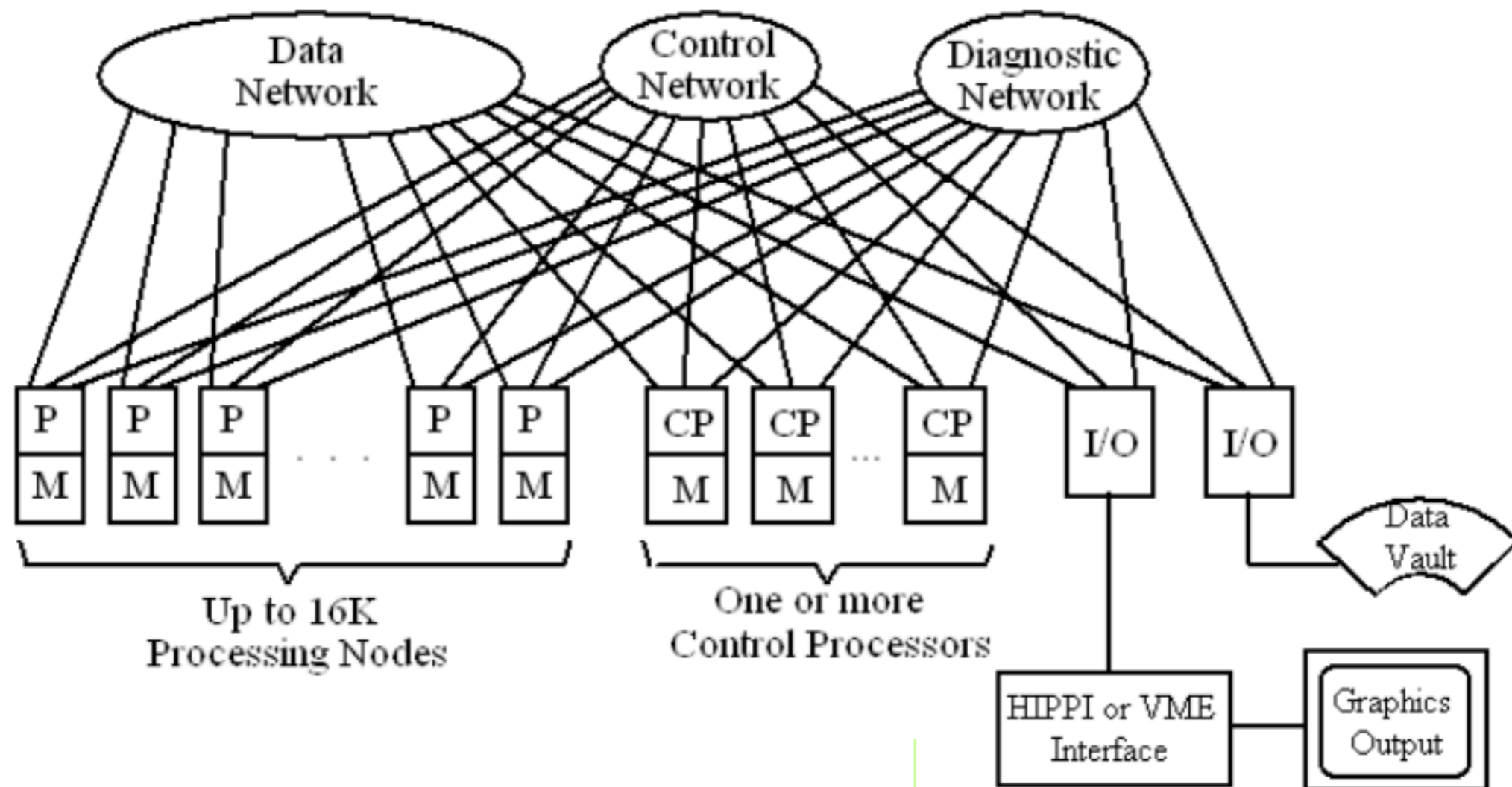
**Background: Computational Chemistry**

- Tim Germann

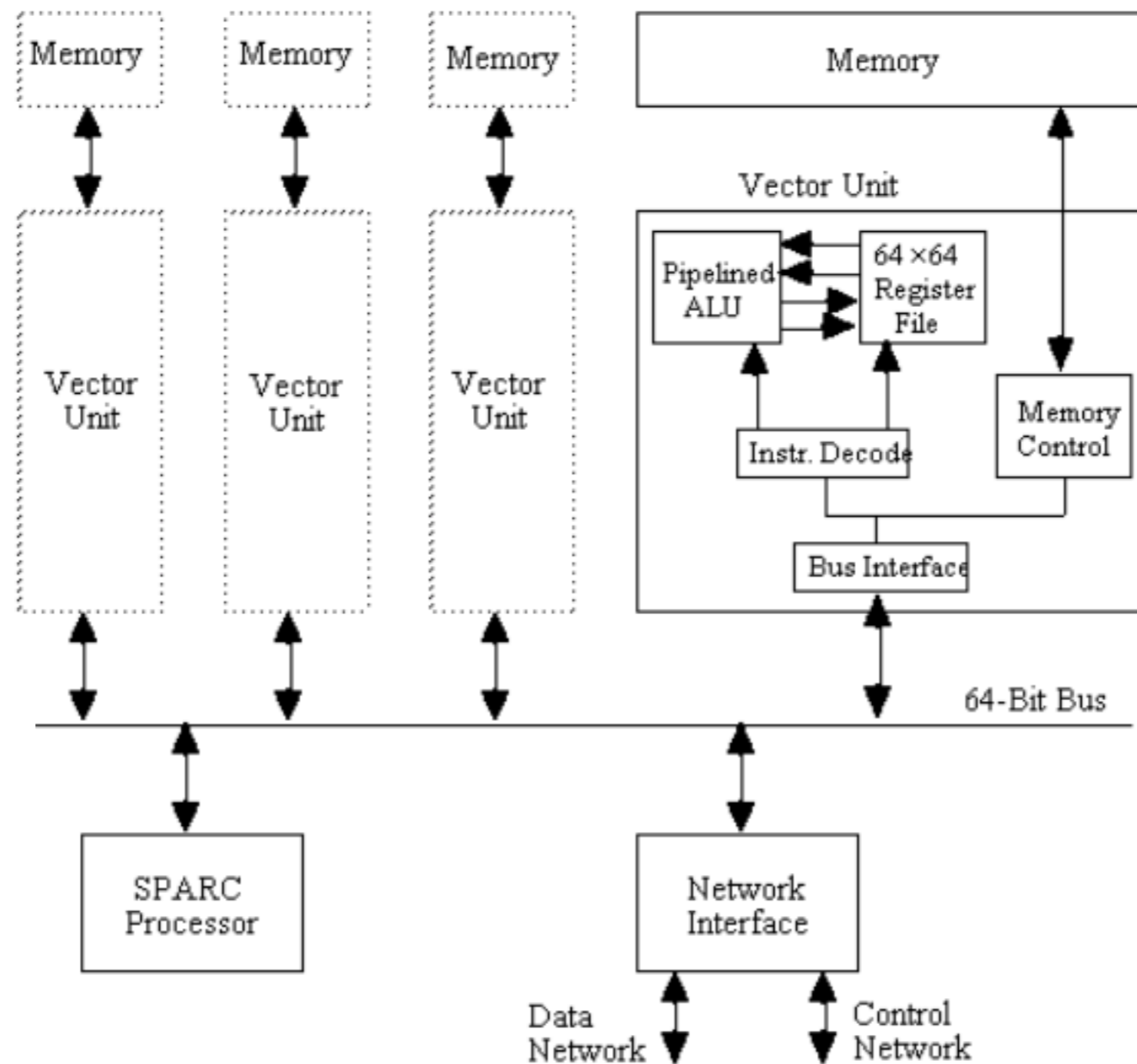
**Background: Computer Science, Computational Chemistry  
The current "owner" of SPaSM**

# THE MACHINE

## Fat-Tree-Based Connection Machine 5



# THE MACHINE



**Effects of pairwise versus many-body forces on high-stress plastic deformation**

139

B. L. Holian,\* A. F. Voter, N. J. Wagner,<sup>†</sup> R. J. Ravelo, and S. P. Chen  
*Theoretical Division, Los Alamos National Laboratory, Los Alamos, New Mexico 87545*

W. G. Hoover and C. G. Hoover  
*Lawrence Livermore National Laboratory, Livermore, California 94550*

J. E. Hammerberg  
*Applied Theoretical Physics Division, Los Alamos National Laboratory, Los Alamos, New Mexico 87545*

T. D. Dontje  
*Thinking Machines Corporation, Cambridge, Massachusetts 02142*  
(Received 29 October 1990)

We propose a model embedded-atom (many-body) potential and test it against an effective, density-independent, pairwise-additive potential in a variety of nonequilibrium molecular-dynamics simulations of plastic deformation under high stress. Even though both kinds of interactions have nearly the same equilibrium equation of state, the defect energies (i.e., vacancy formation and surface energies) are quite different. As a result, we observe significant qualitative differences in flow behavior between systems characterized by purely pairwise interactions versus higher-order many-body forces.

**INTRODUCTION**

Only in the case of noble gases can the interactions between atoms be described realistically by density-independent, pairwise-additive forces. The repulsive and attractive forces arise from spherical electron clouds that stick close to the nuclei. In metals, however, the electrons are not all localized about the nuclei, and, in fact,

If the embedding function is linear in the local embedding density, then obviously its contribution will be simply pairwise additive, but if  $\mathcal{F}$  is *not* linear in  $\rho_i$  then higher-order *many-body* contributions result.

One significant consequence of the nonlinearity of the embedding function is that the energy  $E_{\text{vac}}$  required to form a vacancy in a solid can be made to be much smaller than the bulk cohesive energy  $E_c$  per particle (the two

## MODEL EMBEDDED-ATOM POTENTIAL

Our proposed EAM is constructed from simple analytical functions for the pair-potential contribution  $\varphi$ , the localization function  $w$ , and the embedding function  $\mathcal{F}$ . One motivation for employing analytical functions is that, at this stage of their development, massively parallel computers, which are extremely useful for large-scale MD simulations, can have communication or memory limitations that make large tables of interactions unfeasible. Since the principal bottleneck in MD simulations is the calculation of forces on atoms due to their local environment, computation time is minimized when  $\varphi$  and  $w$  are made as short-ranged as possible. To this end, we employ a cubic spline at the point of maximum attractive force in  $\varphi$ , which makes  $\varphi$  go smoothly to zero for separations  $r > r_{\max}$ ; likewise,  $w$  is smoothly truncated at  $r_{\max}$ .

Within the nearest-neighbor approximation, we can rather easily specify the normal density  $\rho_0$  at zero pressure and temperature (or equivalently, the equilibrium nearest-neighbor separation  $r_0$ ), the cohesive energy  $E_{\text{coh}}$ , and the bulk modulus  $B_0$ . First, we insist that the minimum well depth  $\chi\epsilon$  of the pair potential occur at  $r_0$ , where  $\chi$  is the fractional pair-potential contribution to the total cohesion. Second, we require that, at normal density, the embedding function exhibit a minimum that contributes the remaining fraction  $1-\chi$  to the cohesive energy; thus, in the nearest-neighbor approximation,

$$E_{\text{coh}} = \frac{1}{2}d(d+1)\epsilon, \quad (3)$$

where  $d$  is the dimensionality of the system, i.e., 1, 2, or 3, and the factor  $d(d+1)$  is the number of nearest neighbors in the  $d$ -dimensional close-packed solid. If  $\chi=1$ , one obtains only the pair-potential part, without any many-body embedding. We choose the value of  $\chi$  to be approximately equal to the ratio of vacancy energy to cohesive energy,  $E_{\text{vac}}/E_{\text{coh}}$ , which for metals is typically between 0.2 and 0.4. Finally, the bulk modulus can be adjusted by varying the choice of pair potential (as is possible with a flexible form such as the Morse potential); in this work, we have chosen a short-range Lennard-Jones

$$a_2 = \frac{5r_{\text{spl}}^2 - r_{\text{max}}^2}{8r_{\text{spl}}^3(r_{\text{max}}^2 - r_{\text{spl}}^2)}\varphi', \quad (6b)$$

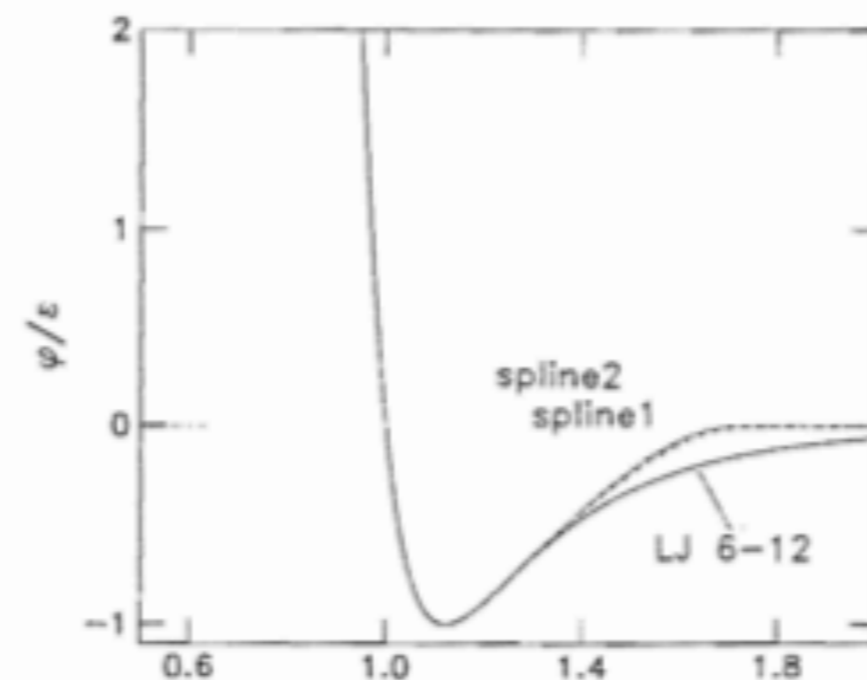
$$a_3 = \frac{3r_{\text{spl}}^2 - r_{\text{max}}^2}{12r_{\text{spl}}^3(r_{\text{max}}^2 - r_{\text{spl}}^2)^2}\varphi'. \quad (6c)$$

For the LJ-spl potential (in units of  $\sigma$  and  $\epsilon$ ),  $r_{\text{spl}}=1.244455$ ,  $r_{\text{max}}=1.711238$ ,  $a_2=0.5424494\chi$ , and  $a_3=0.09350527\chi$ . (This LJ-spl potential differs from the Holian-Evans potential<sup>5</sup> only in that the spline here is in  $r^2$  rather than  $r$ . As a result, no square root or division by  $r$  is required in the MD force calculation.) In Fig. 1, we show the LJ-spl potential.

For the embedding (many-body part of our EAM, the local weighting function  $w$  is given by

$$w(r) = \frac{1}{d(d+1)\epsilon} \left[ \frac{r_{\text{max}}^2 - r^2}{r_{\text{max}}^2 - r_0^2} \right]^2, \quad (7)$$

where  $e$  is the base of the natural logarithms. The weighting of neighbors looks qualitatively like a Gaussian; beyond  $r_{\text{max}}$  it is zero (see Fig. 2). At the normal bulk density ( $r=r_0$ ) the local embedding density, as given by the nearest-neighbor approximation, is  $\rho_l=1/e$ .



# User Interface

```
#!/bin/csh
#JSUB-notty -exact_proc
#JSUB-nproc 1024
#JSUB-dedicated
#JSUB-project mat1-p
#JSUB-cpu 30min
#JSUB-mem 16G
#JSUB-interruptable
```

$r_{max} = 2.50$

```
setenv CMMD_HEAP_SIZE 512000
```

```
SPaSM -il -B -m30000 -p8:8:16 -r0:0:0:80:80:80 -c4:4:2 -C0.25 < in >>& junk Log 1
SPaSM -il -B -m30000 -p8:8:16 -r0:0:0:80:80:160 -c4:4:4 -C0.25 < in >>& junk Log 2
SPaSM -il -B -m30000 -p8:8:16 -r0:0:0:80:80:320 -c4:4:8 -C0.25 < in >>& junk Log 3
SPaSM -il -B -m30000 -p8:8:16 -r0:0:0:160:160:160 -c8:8:4 -C0.25 < in >>& junk Log 4
SPaSM -il -B -m30000 -p8:8:16 -r0:0:0:80:80:640 -c4:4:16 -C0.25 < in >>& junk Log 5
SPaSM -il -B -m30000 -p8:8:16 -r0:0:0:160:160:320 -c8:8:8 -C0.25 < in >>& junk Log 6
SPaSM -il -B -m30000 -p8:8:16 -r0:0:0:160:80:640 -c8:4:16 -C0.25 < in >>& junk Log 7
SPaSM -il -B -m60000 -p8:8:16 -r0:0:0:160:160:640 -c8:8:16 -C0.25 < in >>& junk Log 8
SPaSM -il -B -m120000 -p8:8:16 -r0:0:0:320:320:320 -c16:16:8 -C0.25 < in >>& junk Log 20
SPaSM -il -B -m120000 -p8:8:16 -r0:0:0:320:160:640 -c16:8:16 -C0.25 < in >>& junk Log 21
SPaSM -il -B -m130000 -p8:8:16 -r0:0:0:320:320:640 -c16:16:16 -C0.20 < in >>& junk Log 22
SPaSM -il -B -m30000 -p16:16:4 -r0:0:0:80:80:80 -c2:2:8 -C0.25 < in >>& junk Log 23
SPaSM -il -B -m30000 -p16:16:4 -r0:0:0:80:80:160 -c2:2:16 -C0.25 < in >>& junk Log 24
SPaSM -il -B -m30000 -p16:16:4 -r0:0:0:80:80:320 -c2:2:32 -C0.25 < in >>& junk Log 25
SPaSM -il -B -m30000 -p16:16:4 -r0:0:0:160:160:160 -c4:4:16 -C0.25 < in >>& junk Log 26
SPaSM -il -B -m30000 -p16:16:4 -r0:0:0:80:80:640 -c2:2:64 -C0.25 < in >>& junk Log 27
SPaSM -il -B -m30000 -p16:16:4 -r0:0:0:160:160:320 -c4:4:32 -C0.25 < in >>& junk Log 28
SPaSM -il -B -m30000 -p16:16:4 -r0:0:0:160:80:640 -c4:2:64 -C0.25 < in >>& junk Log 29
SPaSM -il -B -m60000 -p16:16:4 -r0:0:0:160:160:640 -c4:4:64 -C0.25 < in >>& junk Log 30
SPaSM -il -B -m120000 -p16:16:4 -r0:0:0:320:320:320 -c8:8:32 -C0.25 < in >>& junk Log 31
SPaSM -il -B -m120000 -p16:16:4 -r0:0:0:320:160:640 -c8:4:64 -C0.25 < in >>& junk Log 32
SPaSM -il -B -m130000 -p16:16:4 -r0:0:0:320:320:640 -c8:8:64 -C0.20 < in >>& junk Log 33
```

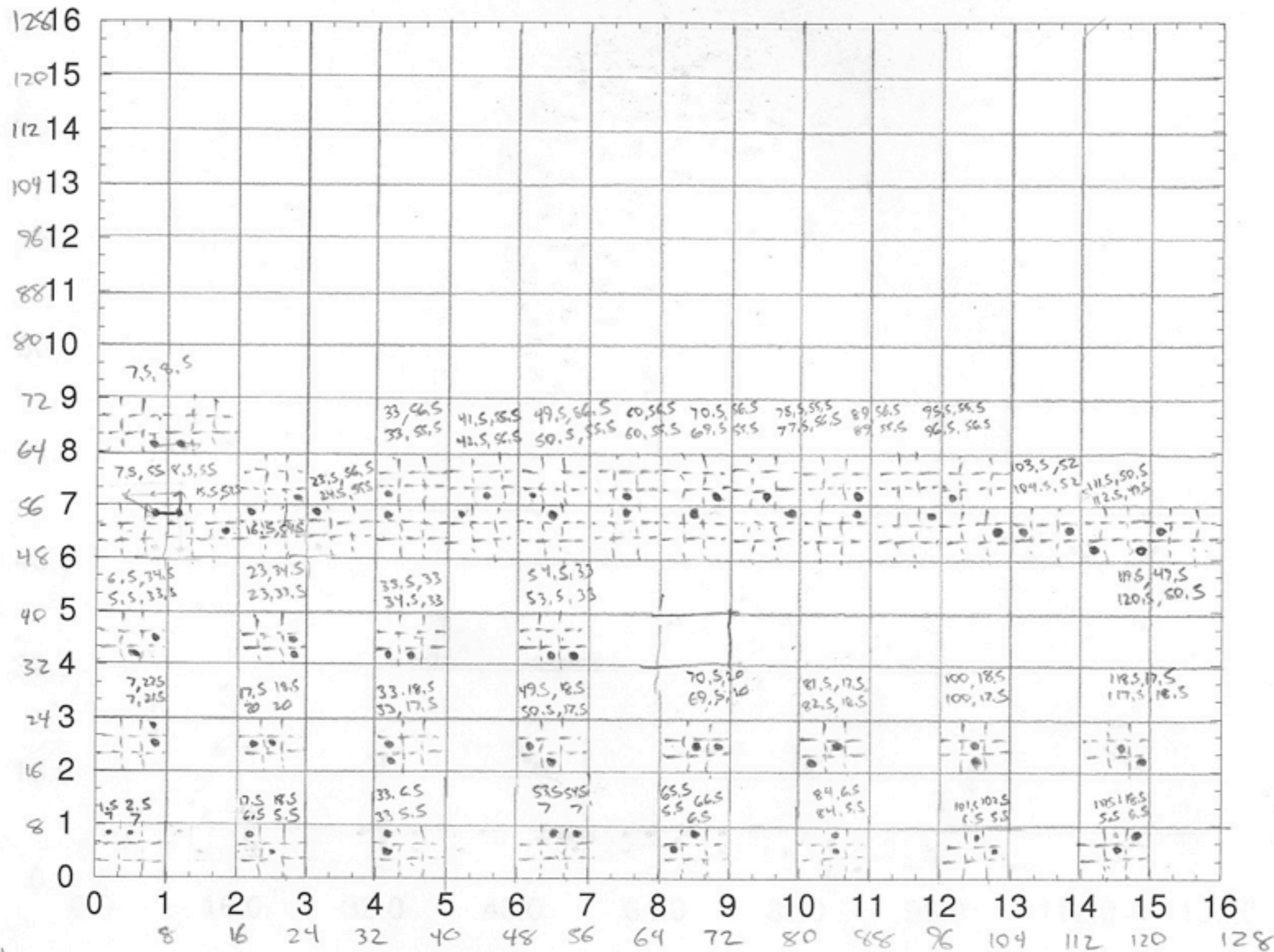
```
SPaSM -il -B -m30000 -p8:8:16 -r0:0:0:80:80:80 -c4:4:2 -C1.00 < in >>& junk Log 34
```

# Testing

$d=1.$

$r_m=1.711$

Force Test



$r_{rot}=2.$



```

pe0: [0][2]:Pos : 1.5 7 0 0 Force : -4.8 0
pe0: [1][2]:Pos : 2.5 7 0 0 Force : 4.8 0
pe2: [1][1]:Pos : 18.5 5.5 0 0 Force : -0.30616 0.30616
pe2: [0][2]:Pos : 17.5 6.5 0 0 Force : 0.30616 -0.30616
pe4: [0][1]:Pos : 33 5.5 0 0 Force : 0 -4.8
pe4: [0][2]:Pos : 33 6.5 0 0 Force : 0 4.8
pe6: [1][2]:Pos : 53.5 7 0 0 Force : -4.8 0
pe6: [2][2]:Pos : 54.5 7 0 0 Force : 4.8 0
pe8: [0][1]:Pos : 65.5 5.5 0 0 Force : 0.30616 0.30616
pe8: [1][2]:Pos : 66.5 6.5 0 0 Force : -0.30616 -0.30616
pe10: [1][1]:Pos : 84 5.5 0 0 Force : 0 -4.8
pe10: [1][2]:Pos : 84 6.5 0 0 Force : 0 4.8
pe12: [2][1]:Pos : 102.5 5.5 0 0 Force : -0.30616 0.30616
pe12: [1][2]:Pos : 101.5 6.5 0 0 Force : 0.30616 -0.30616
pe14: [1][1]:Pos : 117.5 5.5 0 0 Force : 0.30616 0.30616
pe14: [2][2]:Pos : 118.5 6.5 0 0 Force : -0.30616 -0.30616
pe32: [2][1]:Pos : 7 21.5 0 0 Force : 0 -4.8
pe32: [2][2]:Pos : 7 22.5 0 0 Force : 0 4.8
pe34: [0][1]:Pos : 17.5 20 0 0 Force : -4.8 0
pe34: [1][1]:Pos : 18.5 20 0 0 Force : 4.8 0
pe36: [0][0]:Pos : 33 17.5 0 0 Force : 0 -4.8
pe36: [0][1]:Pos : 33 18.5 0 0 Force : 0 4.8
pe38: [1][0]:Pos : 50.5 17.5 0 0 Force : -0.30616 0.30616
pe38: [0][1]:Pos : 49.5 18.5 0 0 Force : 0.30616 -0.30616
pe40: [1][1]:Pos : 69.5 20 0 0 Force : -4.8 0
pe40: [2][1]:Pos : 70.5 20 0 0 Force : 4.8 0
pe42: [0][0]:Pos : 81.5 17.5 0 0 Force : 0.30616 0.30616
pe42: [1][1]:Pos : 82.5 18.5 0 0 Force : -0.30616 -0.30616
pe44: [1][0]:Pos : 100 17.5 0 0 Force : 0 -4.8
pe44: [1][1]:Pos : 100 18.5 0 0 Force : 0 4.8
pe46: [2][0]:Pos : 118.5 17.5 0 0 Force : -0.30616 0.30616
pe46: [1][1]:Pos : 117.5 18.5 0 0 Force : 0.30616 -0.30616
pe64: [1][0]:Pos : 5.5 33.5 0 0 Force : 0.30616 0.30616
pe64: [2][1]:Pos : 6.5 34.5 0 0 Force : -0.30616 -0.30616
pe66: [2][0]:Pos : 23 33.5 0 0 Force : 0 -4.8
pe66: [2][1]:Pos : 23 34.5 0 0 Force : 0 4.8
pe68: [0][0]:Pos : 33.5 33 0 0 Force : -4.8 0
pe68: [1][0]:Pos : 34.5 33 0 0 Force : 4.8 0
pe70: [1][0]:Pos : 53.5 33 0 0 Force : -4.8 0
pe70: [2][0]:Pos : 54.5 33 0 0 Force : 4.8 0
pe96: [2][2]:Pos : 7.5 55 0 0 Force : -4.8 0
pe97: [2][1]:Pos : 15.5 53.5 0 0 Force : 0.30616 0.30616
pe97: [0][2]:Pos : 8.5 55 0 0 Force : 4.8 0
pe98: [0][2]:Pos : 16.5 54.5 0 0 Force : -0.30616 -0.30616
pe99: [0][2]:Pos : 24.5 55.5 0 0 Force : -0.30616 0.30616
pe100: [0][2]:Pos : 33 55.5 0 0 Force : 0 -4.8
pe101: [0][2]:Pos : 41.5 55.5 0 0 Force : 0.30616 0.30616
pe102: [1][2]:Pos : 50.5 55.5 0 0 Force : -0.30616 0.30616
pe103: [1][2]:Pos : 60 55.5 0 0 Force : 0 -4.8
pe104: [1][2]:Pos : 69.5 55.5 0 0 Force : 0.30616 0.30616
pe105: [2][2]:Pos : 78.5 55.5 0 0 Force : -0.30616 0.30616
pe107: [0][2]:Pos : 89 55.5 0 0 Force : 0 -4.8
pe107: [2][2]:Pos : 95.5 55.5 0 0 Force : 0.30616 0.30616
pe108: [2][1]:Pos : 103.5 52 0 0 Force : -4.8 0
pe109: [0][1]:Pos : 104.5 52 0 0 Force : 4.8 0
pe109: [2][1]:Pos : 111.5 50.5 0 0 Force : 0.30616 -0.30616
pe110: [0][0]:Pos : 112.5 49.5 0 0 Force : -0.30616 0.30616
pe110: [2][0]:Pos : 119.5 49.5 0 0 Force : 0.30616 0.30616
pe111: [0][1]:Pos : 120.5 50.5 0 0 Force : -0.30616 -0.30616
pe114: [2][0]:Pos : 23.5 56.5 0 0 Force : 0.30616 -0.30616
pe116: [0][0]:Pos : 33 56.5 0 0 Force : 0 4.8
pe117: [1][0]:Pos : 42.5 56.5 0 0 Force : -0.30616 -0.30616
pe118: [0][0]:Pos : 49.5 56.5 0 0 Force : 0.30616 -0.30616
pe119: [1][0]:Pos : 60 56.5 0 0 Force : 0 4.8

```

```

pe120: [2][0]:Pos : 70.5 56.5 0 0 Force : -0.30616 -0.30616
pe121: [1][0]:Pos : 77.5 56.5 0 0 Force : 0.30616 -0.30616
pe123: [0][0]:Pos : 89 56.5 0 0 Force : 0 4.8
pe124: [0][0]:Pos : 96.5 56.5 0 0 Force : -0.30616 -0.30616
pe128: [2][0]:Pos : 7.5 65 0 0 Force : -4.8 0
pe129: [0][0]:Pos : 8.5 65 0 0 Force : 4.8 0

```

Debugging

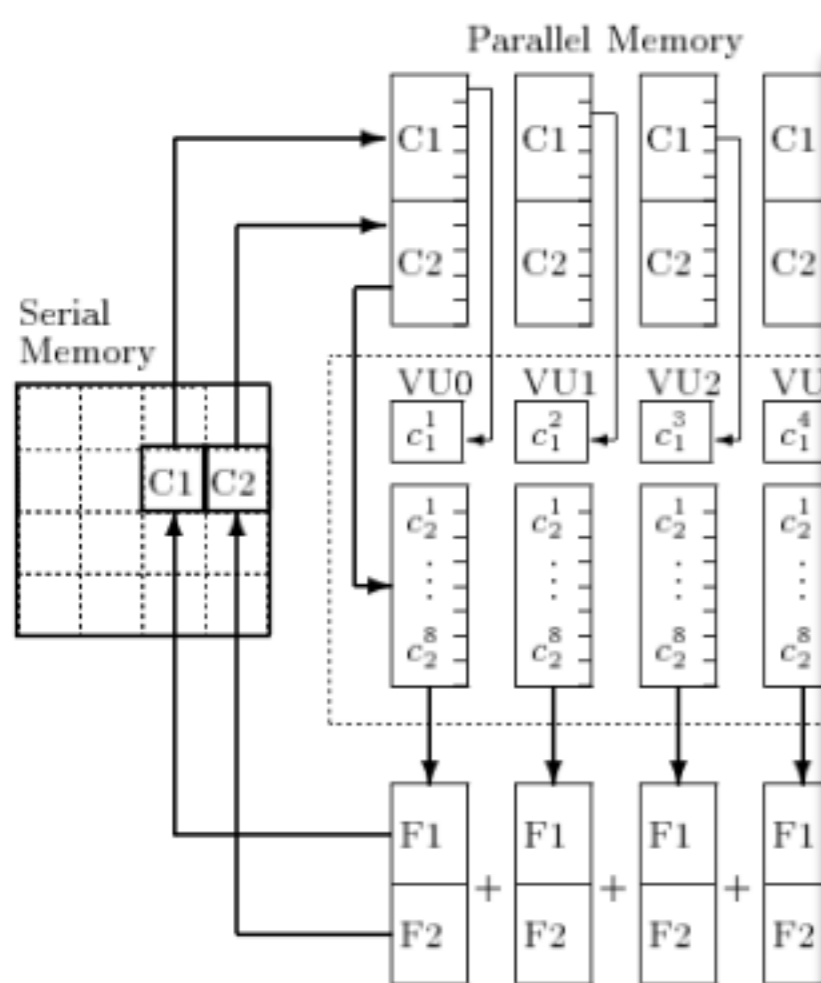


Fig. 3. Calculating forces on the VUs

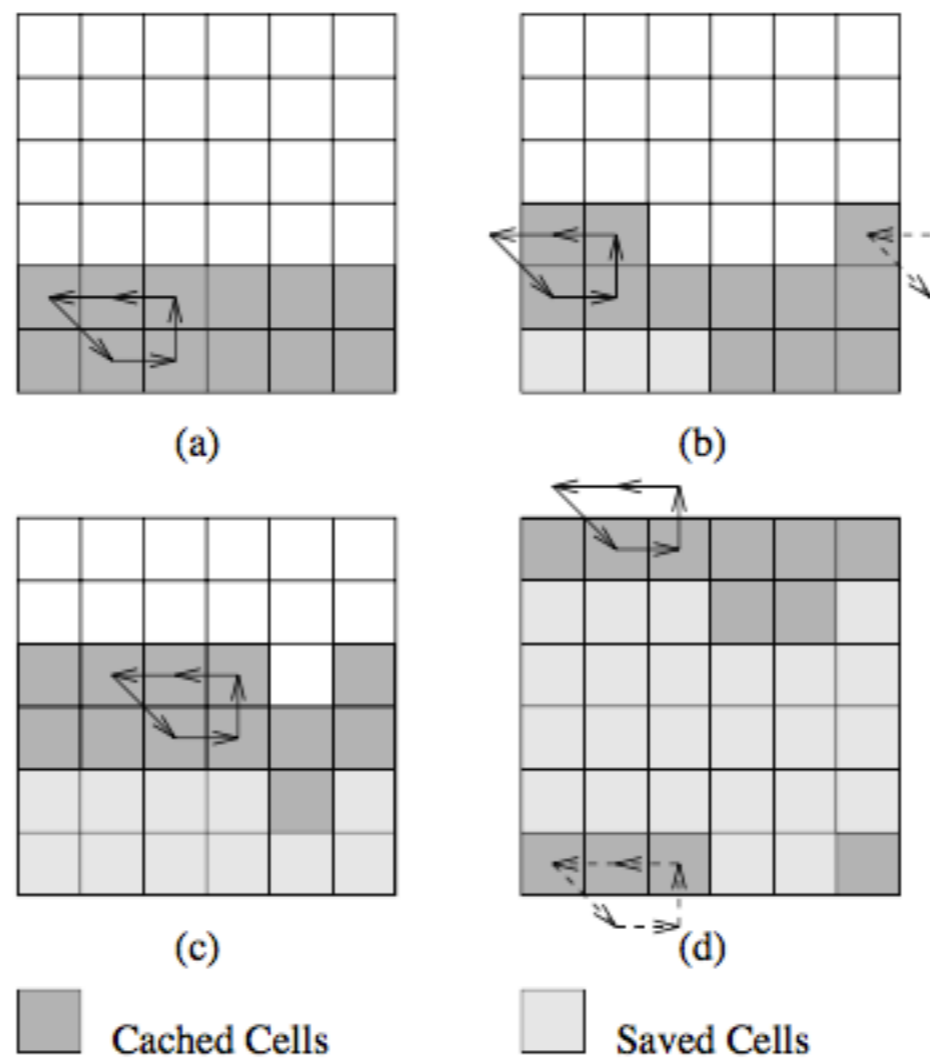
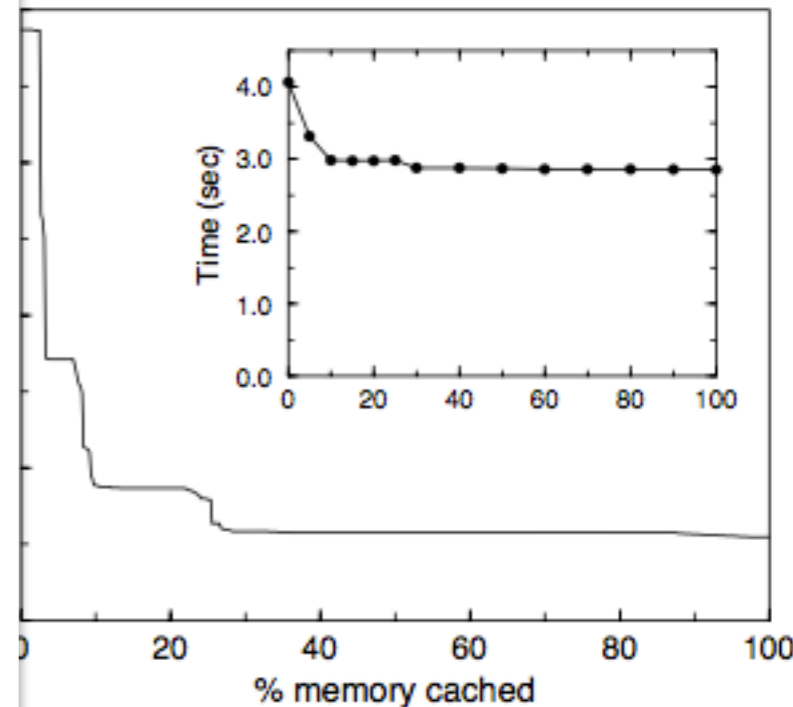


Fig. 5. Caching at different stages of force calculation



Number of cache loads and time per timestep particles on 1024 PNs. 512 cells per PN.

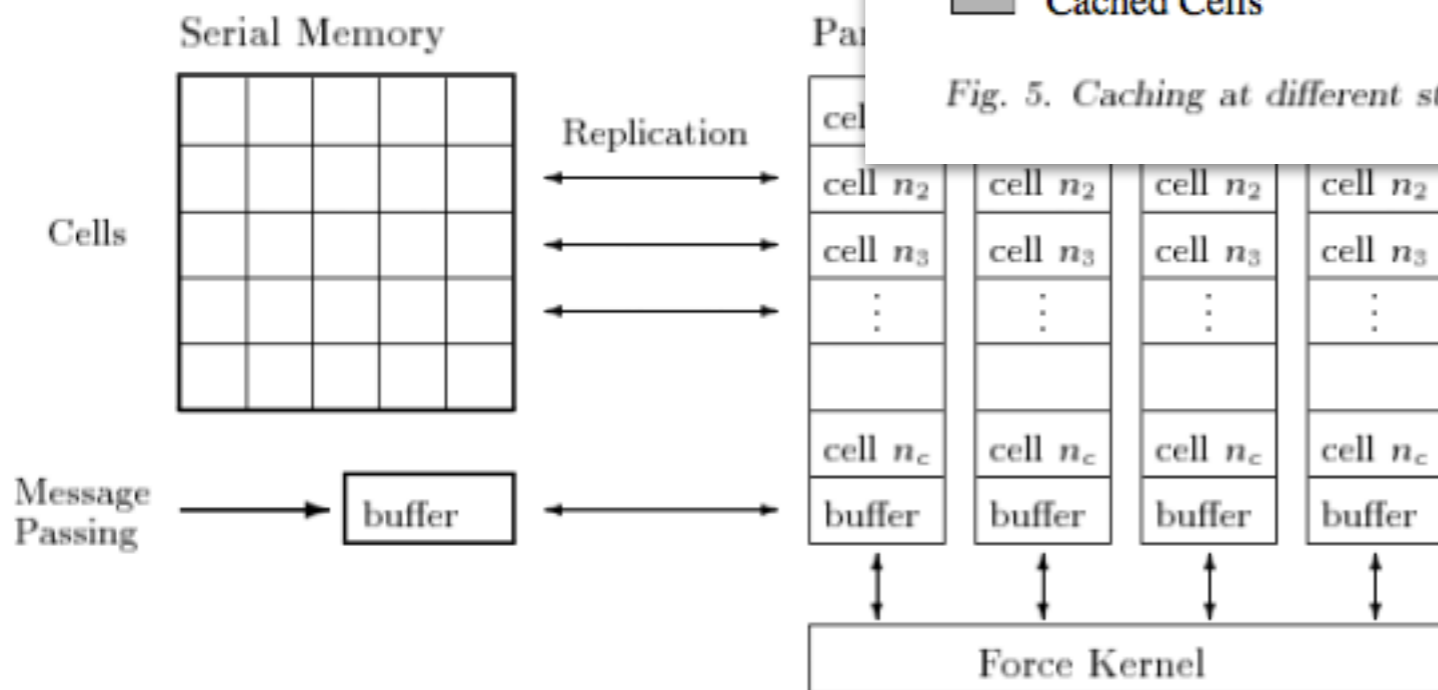
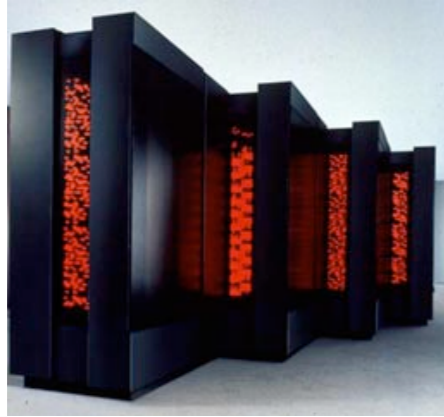


Fig. 4. Parallel Memory Caching Layout

Tuning

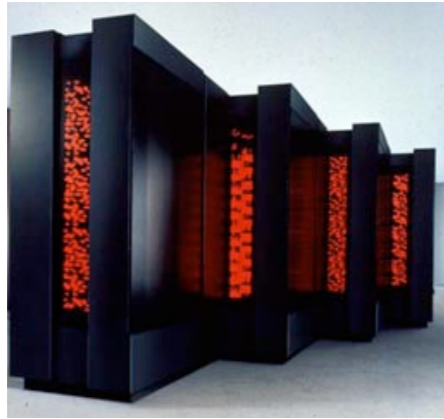
# Our Workflow



Batch job

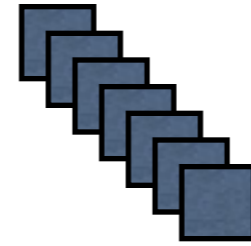
```
setenv CMMD_HEAP_SIZE 512000
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:80 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:160 -c4
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:320 -c4
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:160 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:640 -c4
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:320 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:80:640 -c
SPaSM -i1 -B -m60000 -p8:8:16 -r0:0:0:160:160:640 -
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:320:320
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:160:640
SPaSM -i1 -B -m130000 -p8:8:16 -r0:0:0:320:320:640
```

# Our Workflow



3-6 hours

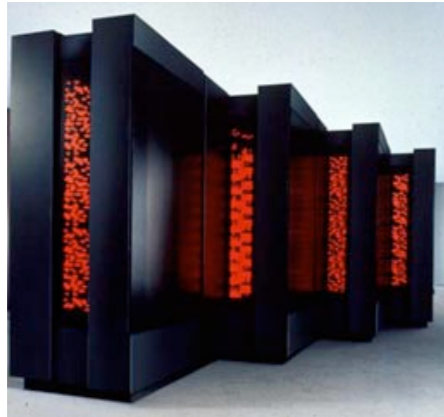
Data Files



Batch job

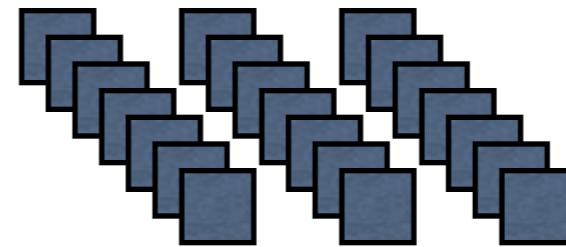
```
setenv CMMD_HEAP_SIZE 512000
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:80 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:160 -c4
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:320 -c4
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:160 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:640 -c4
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:320 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:80:640 -c
SPaSM -i1 -B -m60000 -p8:8:16 -r0:0:0:160:160:640 -
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:320:320
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:160:640
SPaSM -i1 -B -m130000 -p8:8:16 -r0:0:0:320:320:640
```

# Our Workflow



3-6 hours

Data Files



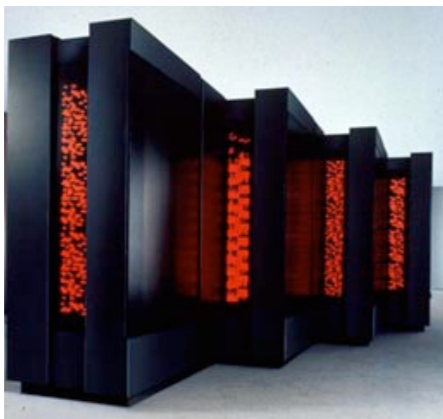
(10-20 GB)

repeat

Batch job

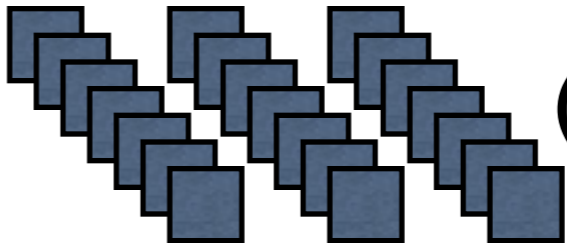
```
setenv CMMD_HEAP_SIZE 512000
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:80 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:160 -c4
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:320 -c4
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:160 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:640 -c4
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:320 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:80:640 -c
SPaSM -i1 -B -m60000 -p8:8:16 -r0:0:0:160:160:640 -
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:320:320
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:160:640
SPaSM -i1 -B -m130000 -p8:8:16 -r0:0:0:320:320:640
```

# Our Workflow



3-6 hours

Data Files



(10-20 GB)

repeat

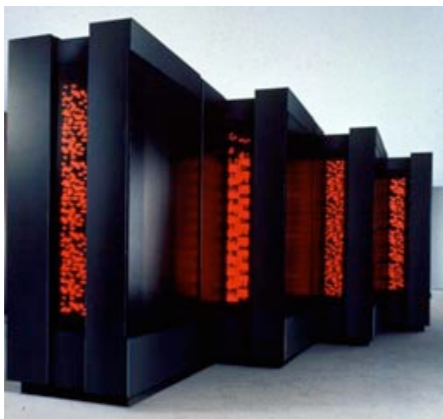
FTP (10 Mbit/s)

Batch job

```
setenv CMMD_HEAP_SIZE 512000
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:80 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:160 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:320 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:160 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:640 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:320 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:80:640 -c
SPaSM -i1 -B -m60000 -p8:8:16 -r0:0:0:160:160:640 -
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:320:320
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:160:640
SPaSM -i1 -B -m130000 -p8:8:16 -r0:0:0:320:320:640
```



# Our Workflow



3-6 hours

Data Files

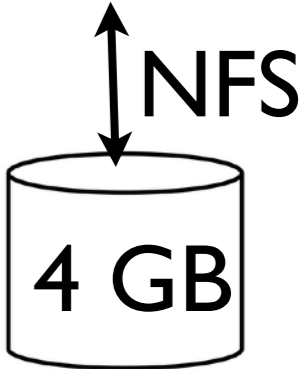
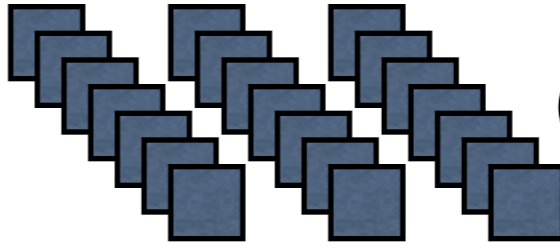
(10-20 GB)

repeat

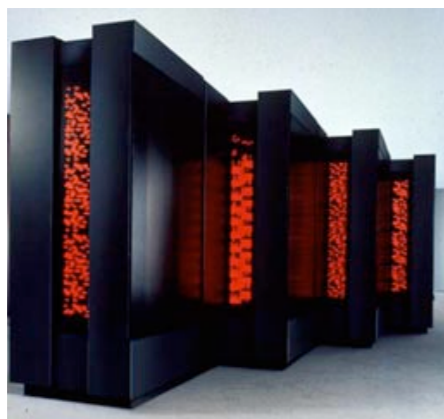
FTP (10 Mbit/s)

Batch job

```
setenv CMMD_HEAP_SIZE 512000
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:80 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:160 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:320 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:160 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:640 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:320 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:80:640 -c
SPaSM -i1 -B -m60000 -p8:8:16 -r0:0:0:160:160:640 -
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:320:320
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:160:640
SPaSM -i1 -B -m130000 -p8:8:16 -r0:0:0:320:320:640
```



# Our Workflow



3-6 hours

Data Files

(10-20 GB)

repeat

Batch job

```
setenv CMMD_HEAP_SIZE 512000
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:80 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:160 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:320 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:160 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:80:80:640 -c4:
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:160:320 -
SPaSM -i1 -B -m30000 -p8:8:16 -r0:0:0:160:80:640 -c
SPaSM -i1 -B -m60000 -p8:8:16 -r0:0:0:160:160:640 -
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:320:320
SPaSM -i1 -B -m120000 -p8:8:16 -r0:0:0:320:160:640
SPaSM -i1 -B -m130000 -p8:8:16 -r0:0:0:320:320:640
```

FTP (10 Mbit/s)



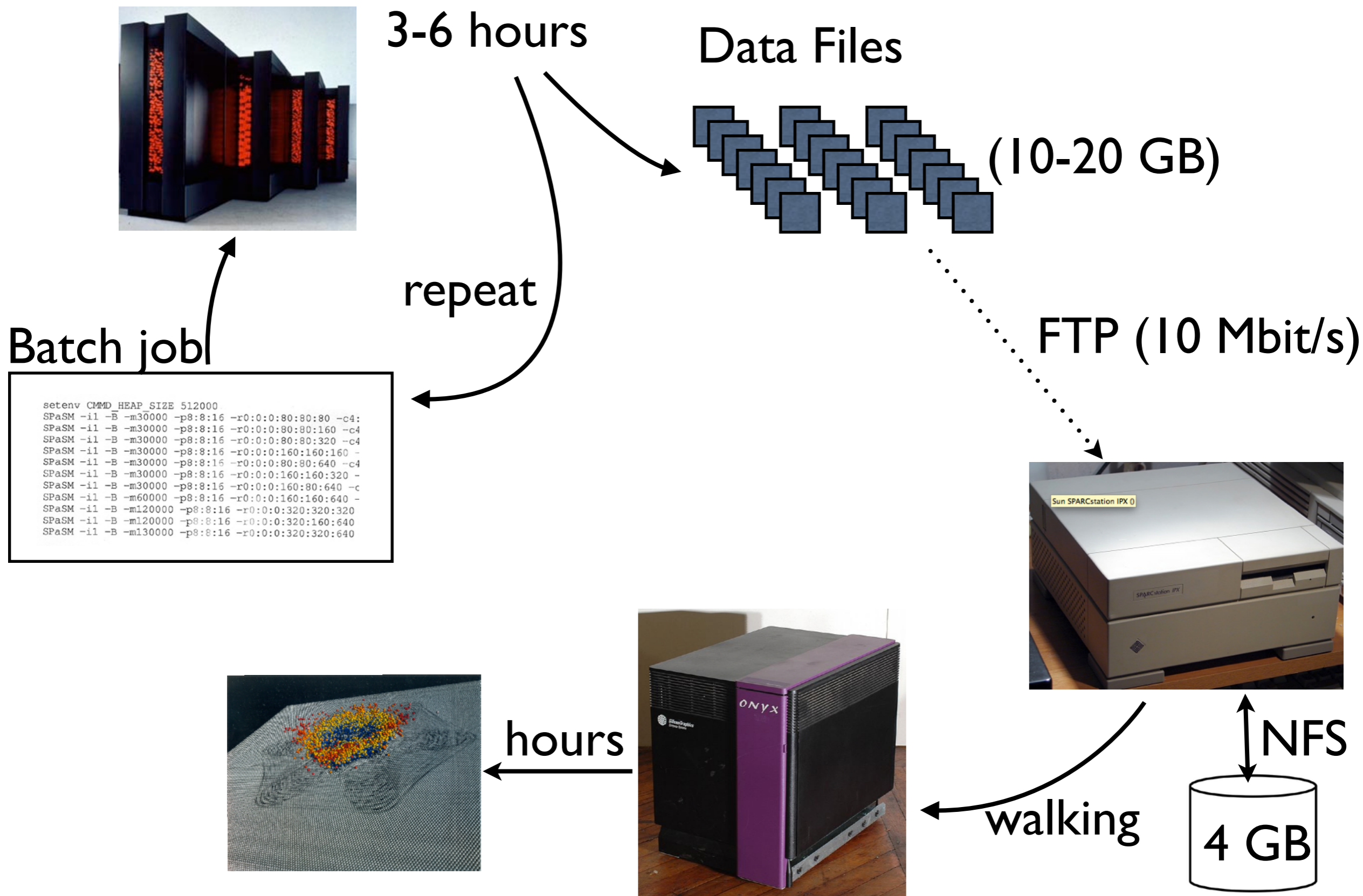
walking

NFS

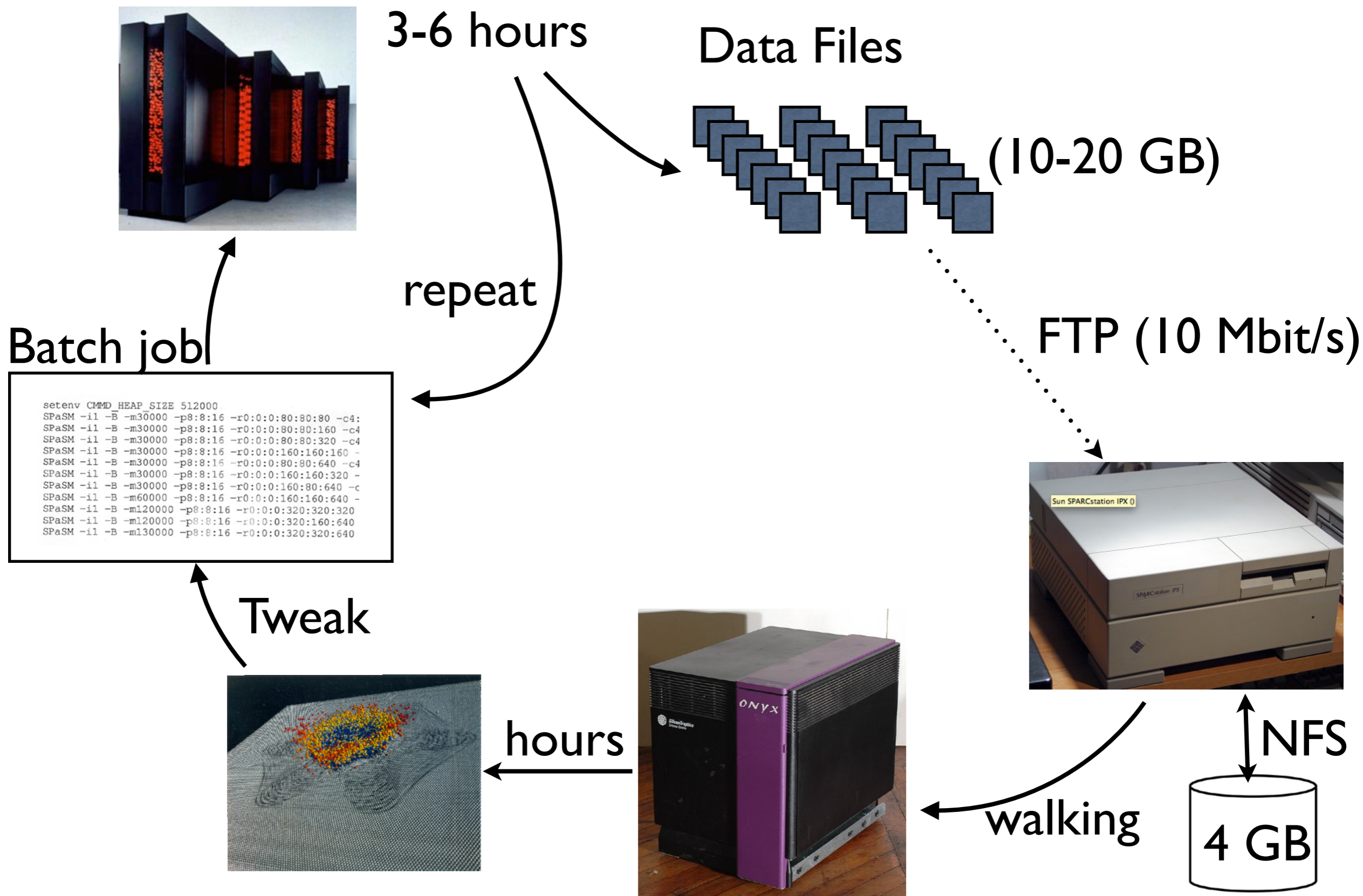
4 GB



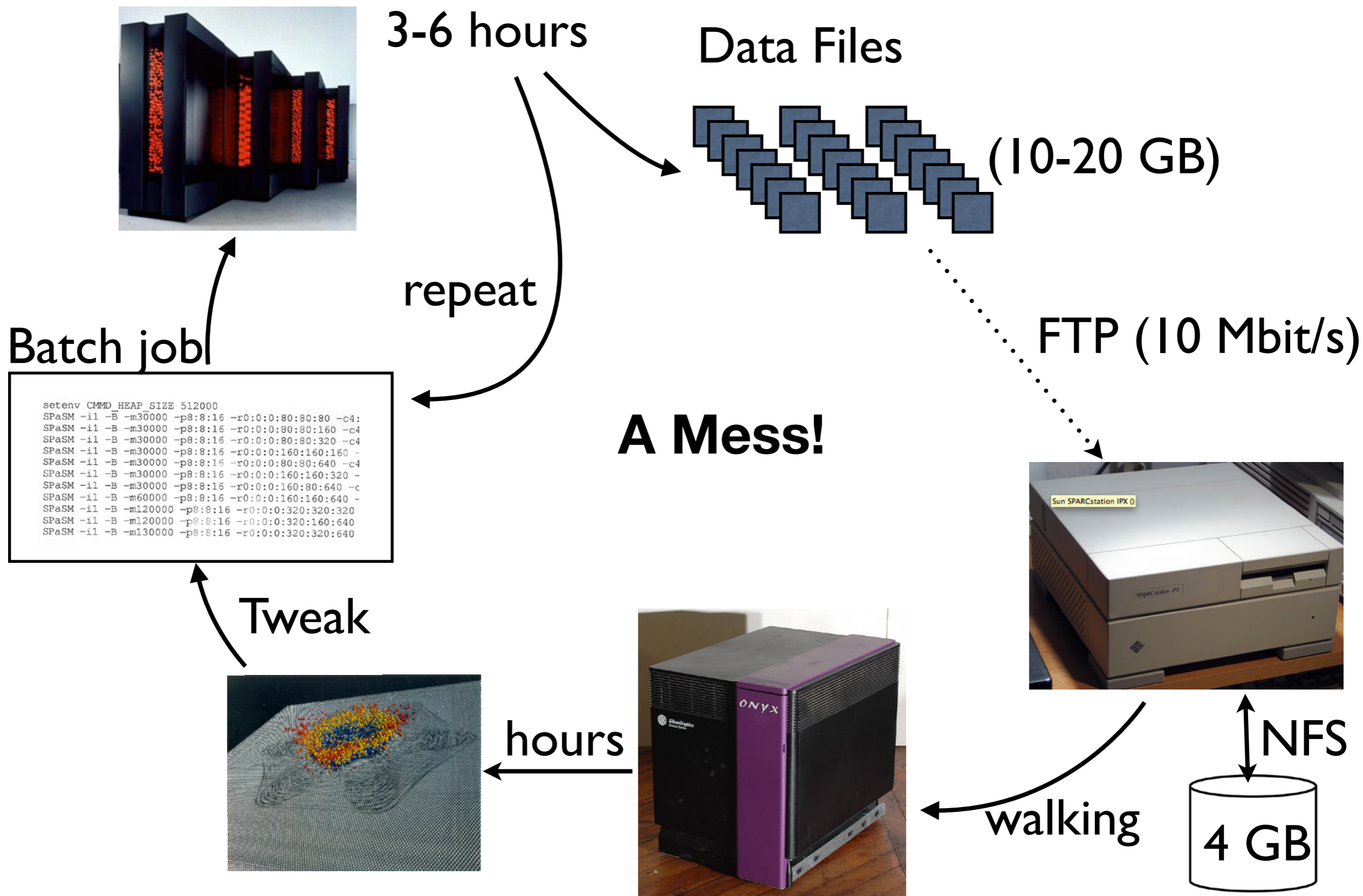
# Our Workflow



# Our Workflow



# Our Workflow



# A Problem: Tools

"Closing the gap between potential and measured performance of computing systems is a continuing problem of fundamental importance. The difficulty of the problem is exacerbated by the increasing complexity of systems as they become more parallel, more heterogeneous, and more distributed."

**Debugging and Performance Tuning for Parallel Computing Systems,  
IEEE Computer Society, (1996)**

# A Problem: Tools

"Closing the gap between potential and **measured performance** of computing systems

is a continuing problem of fundamental

"After investing substantial effort in developing a program, its execution may yield only a small fraction of peak system performance."

**Debugging and Performance Tuning for Parallel Computing Systems,  
IEEE Computer Society, (1996)**

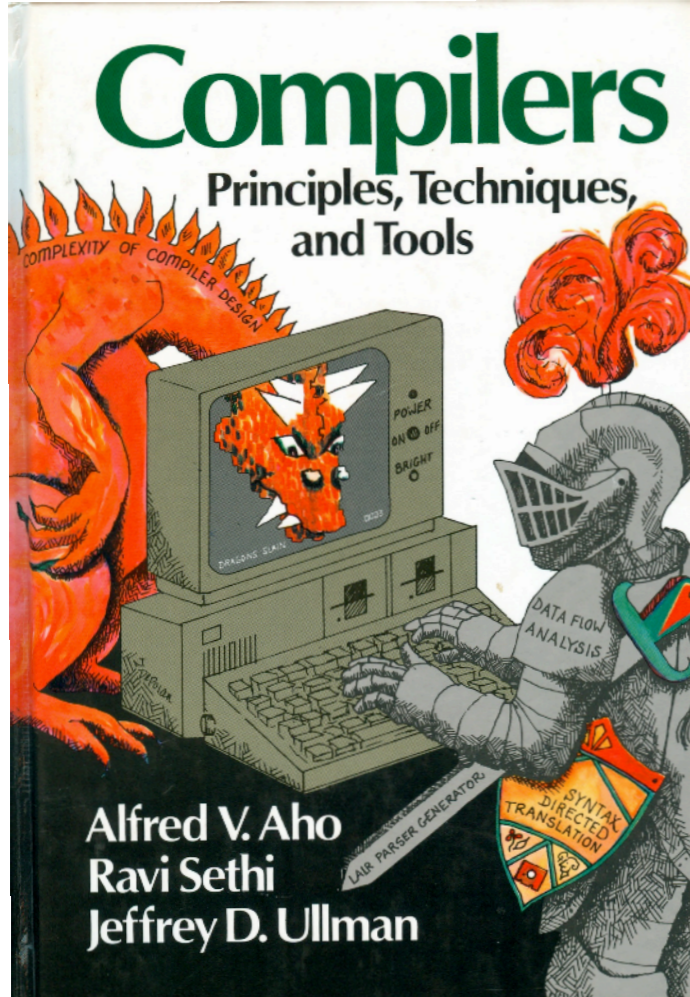
1. Make it work fast.
2. Make it work.

Is PERFORMANCE the  
actual problem?

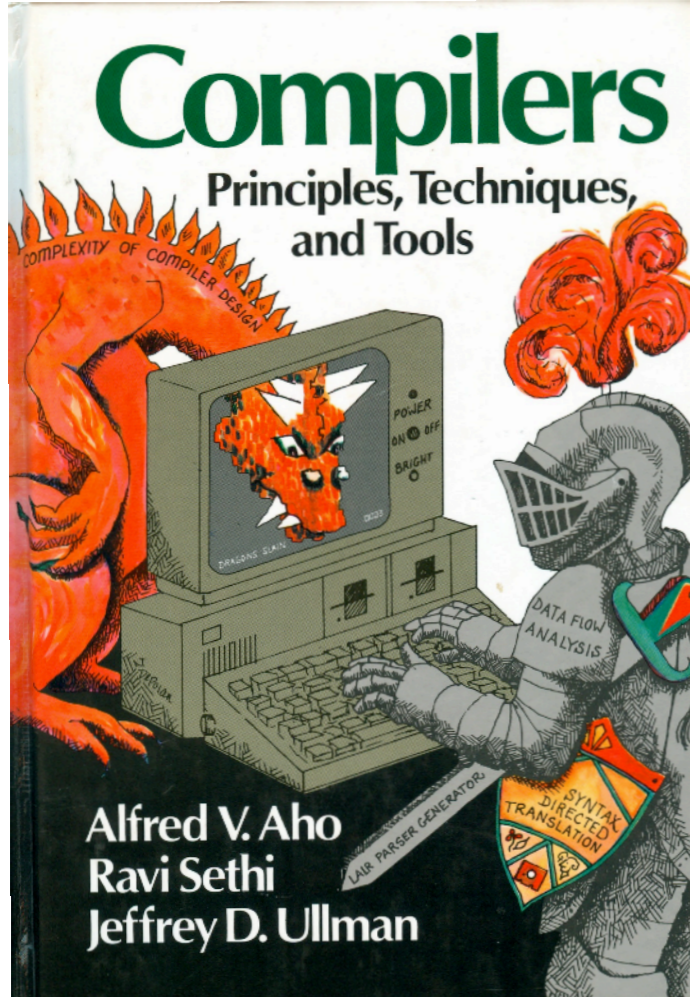
1995



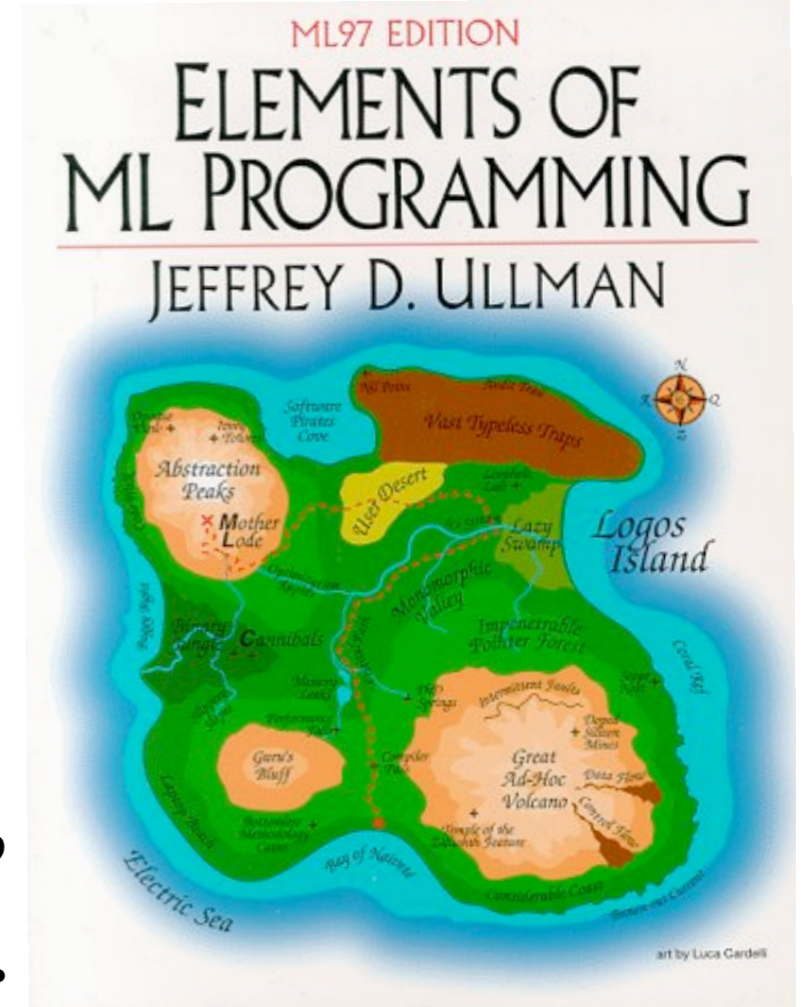
(I had finished the first year of a computer science Ph.D.)

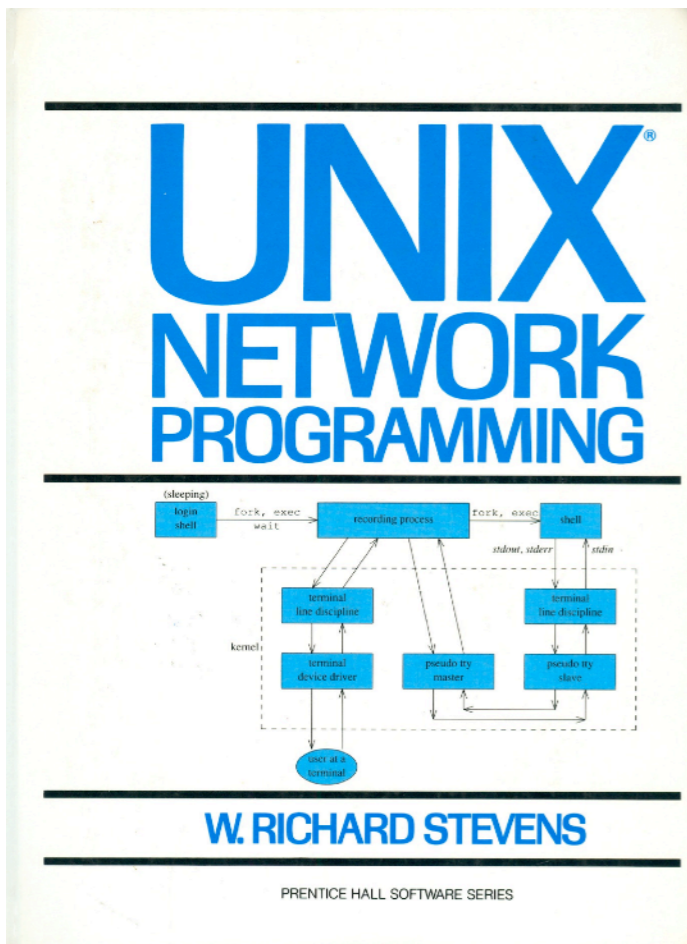
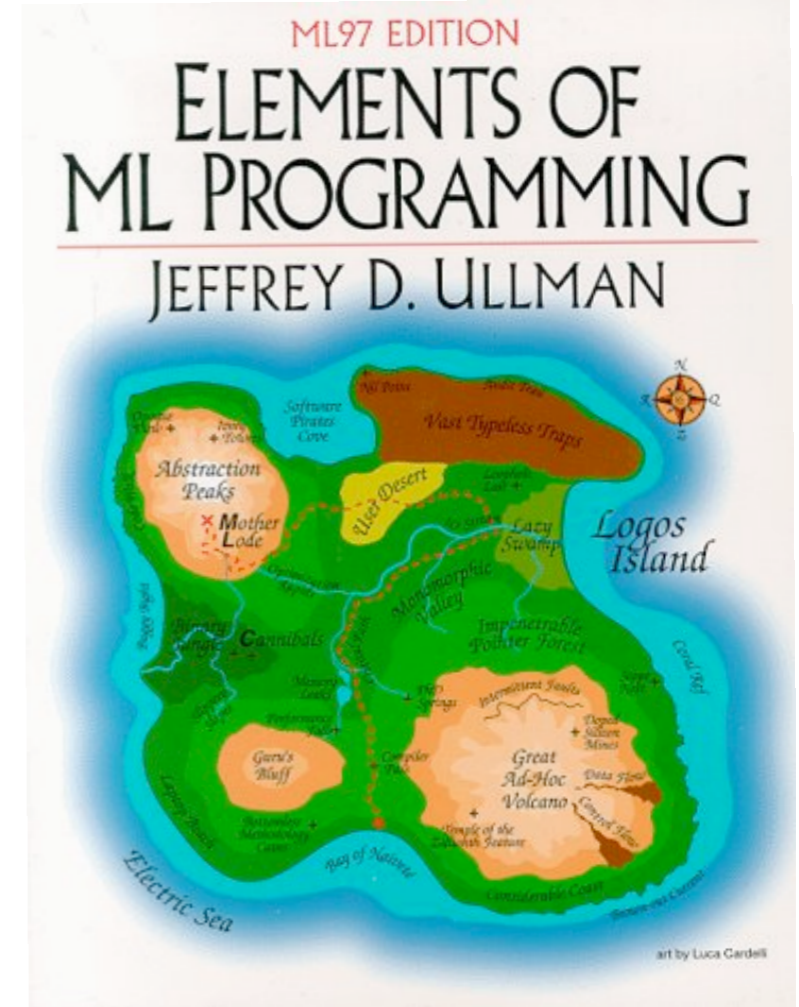
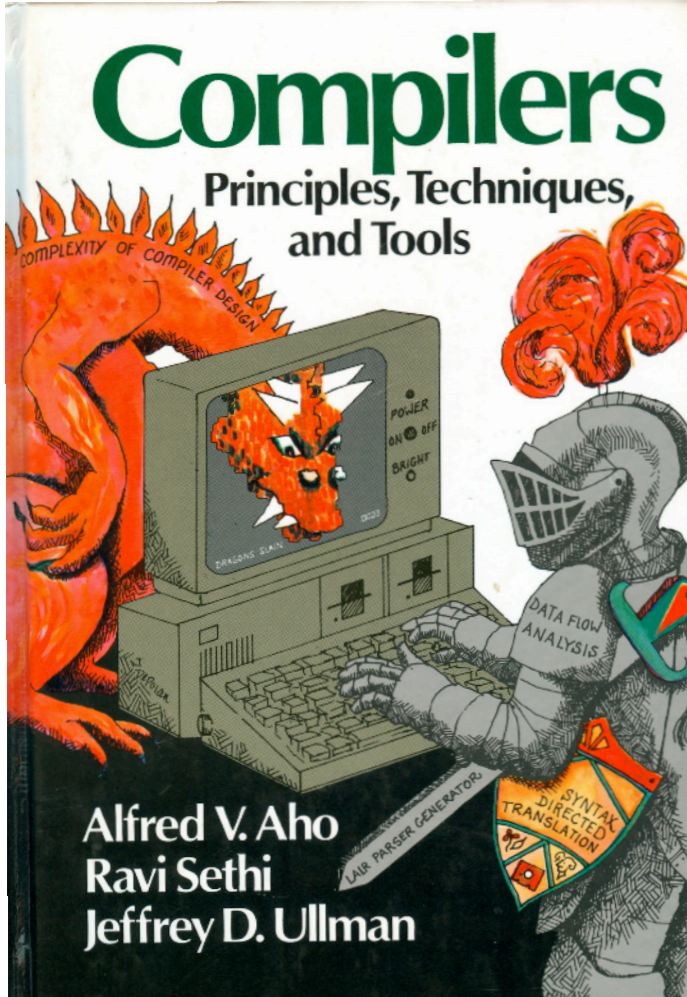


Compilers!

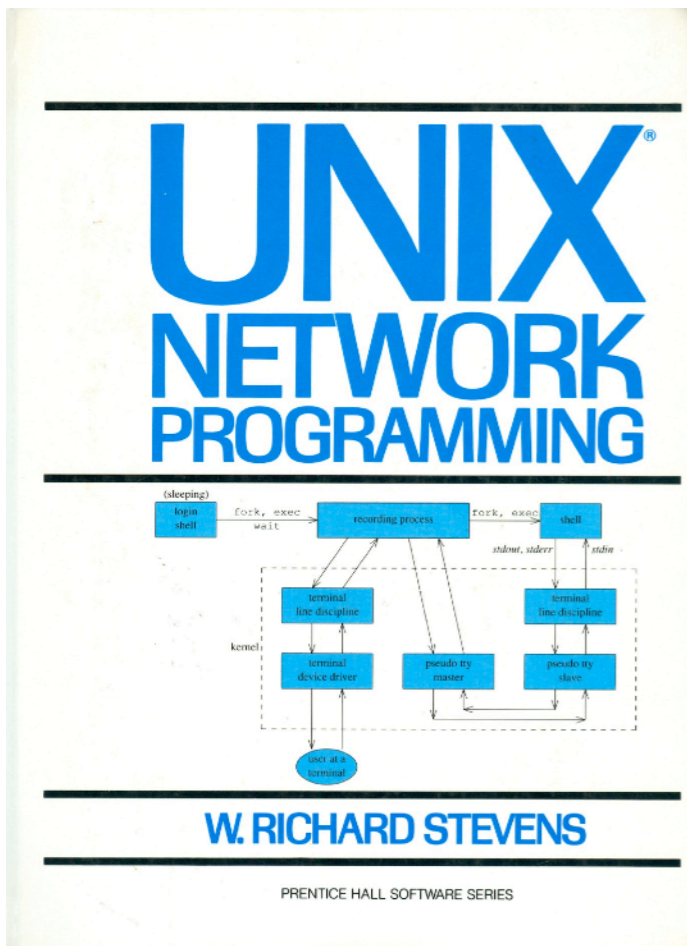
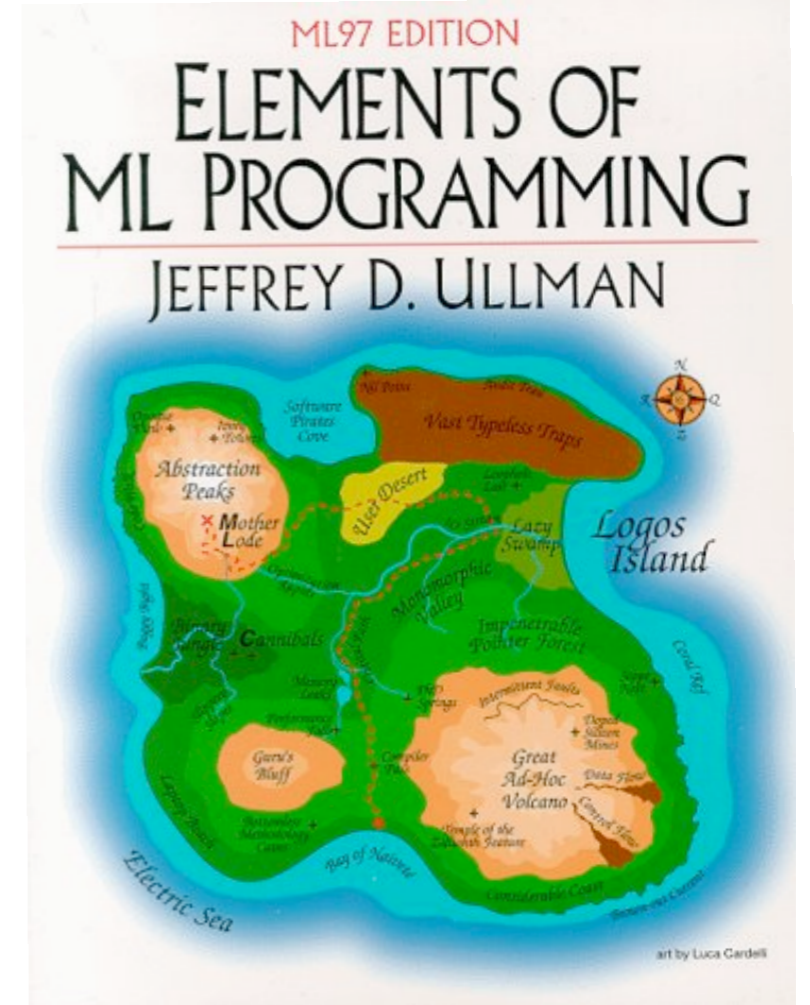
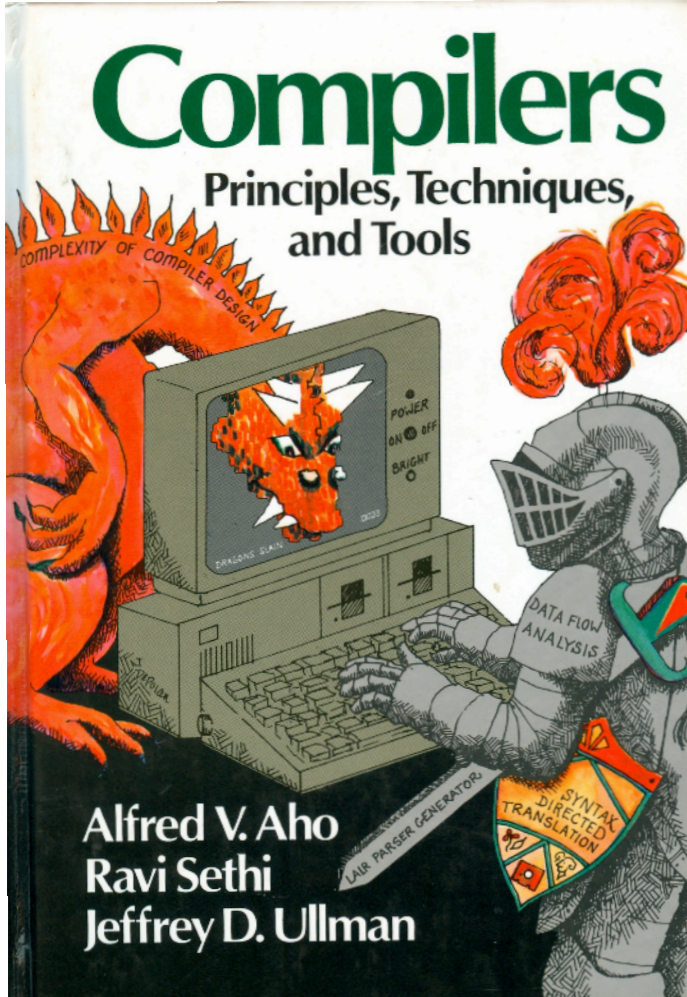


Functional  
Programming,  
Type systems.



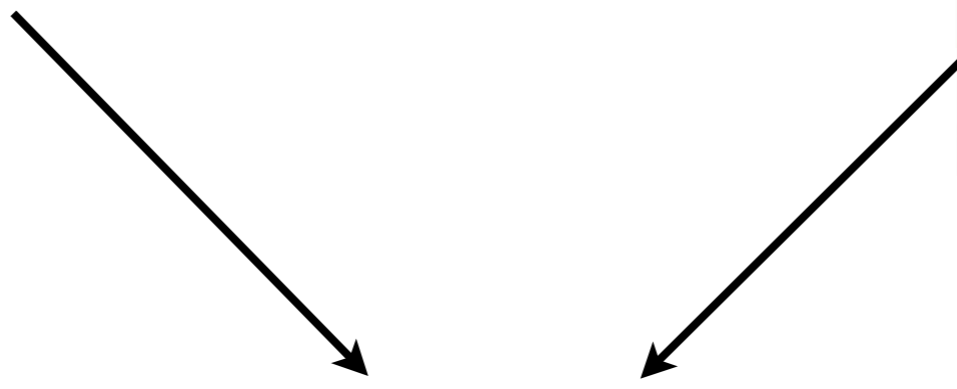
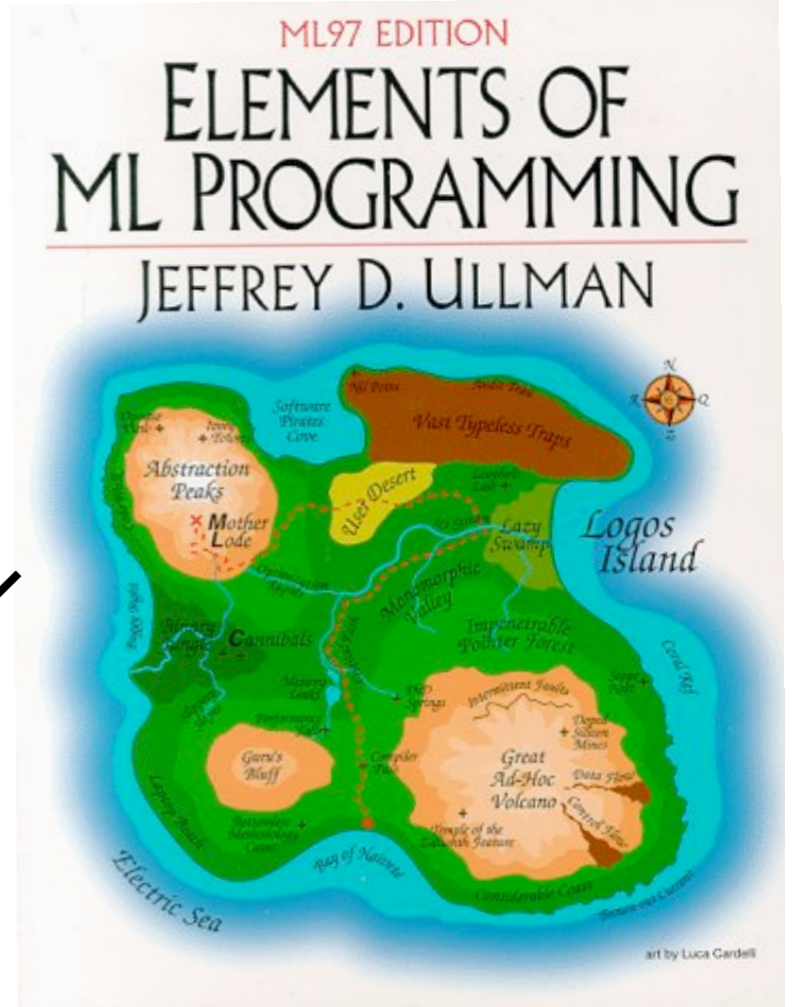
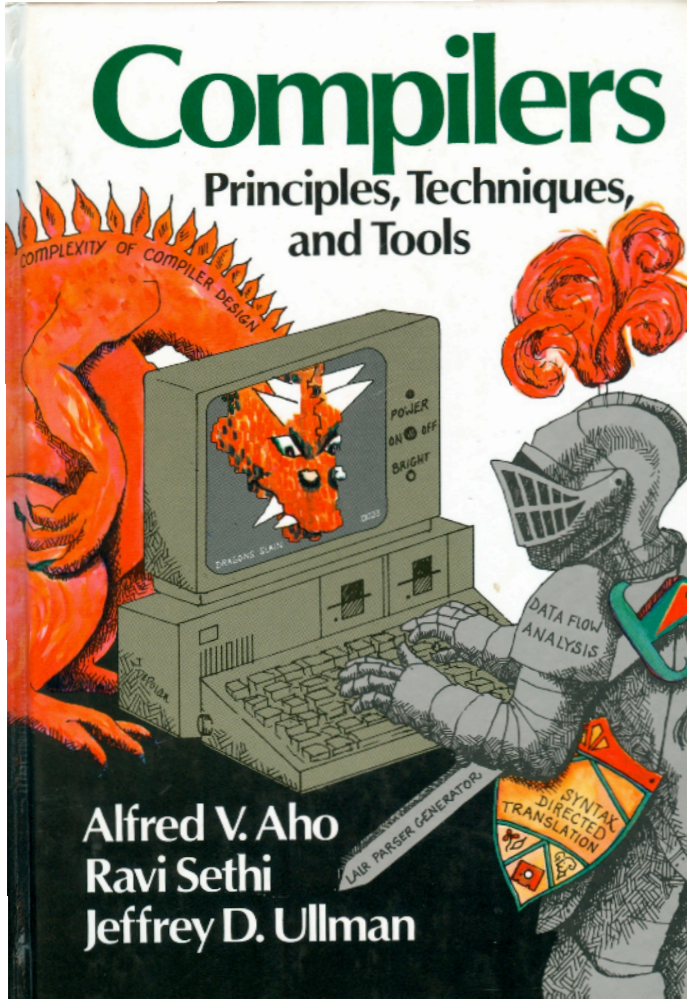


# Systems

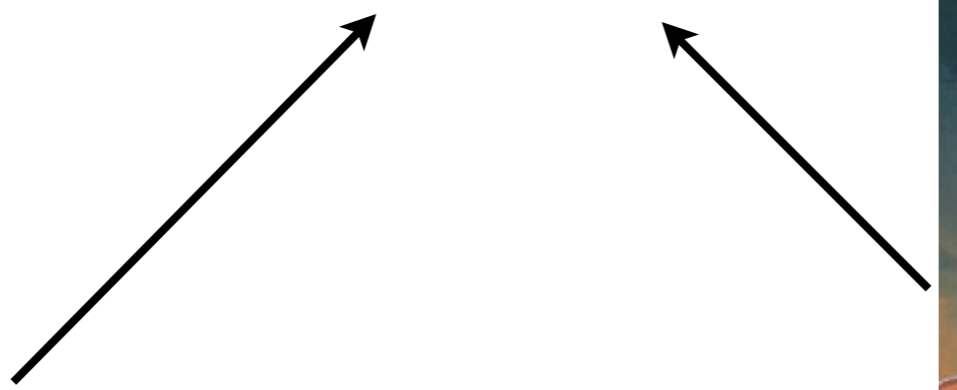
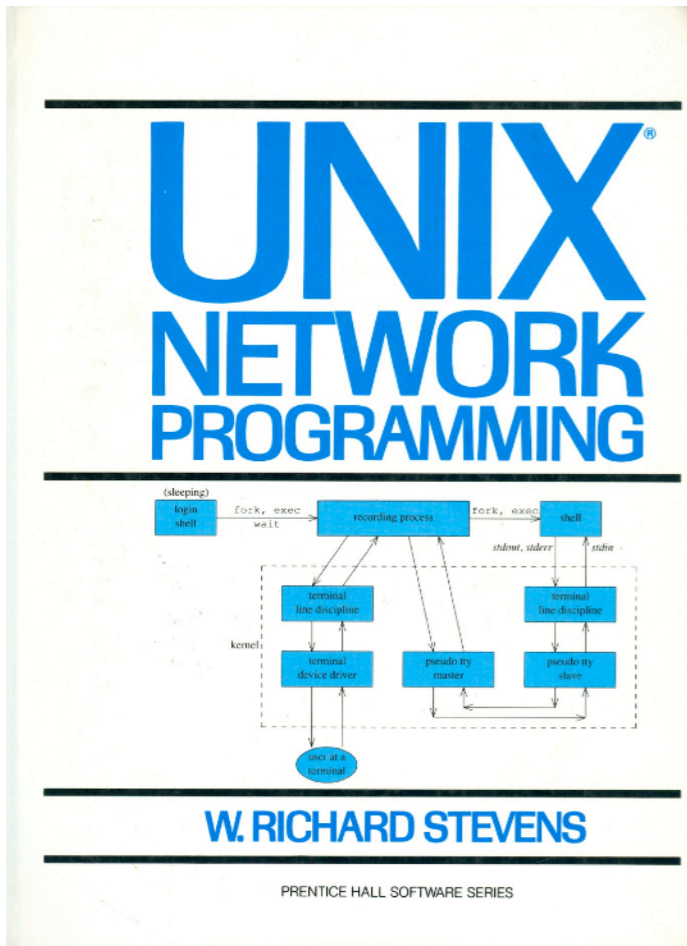


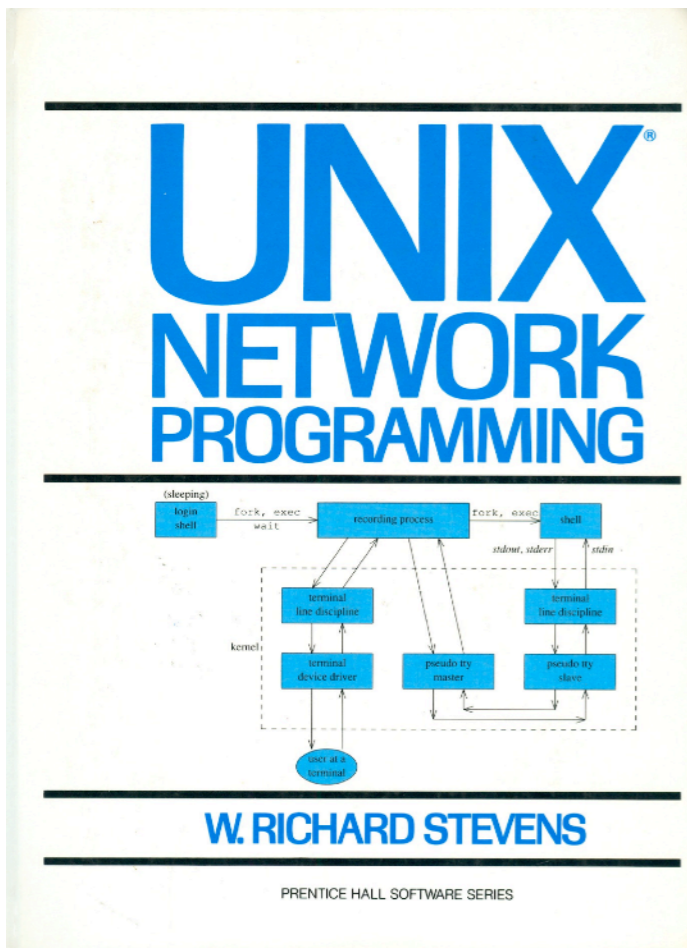
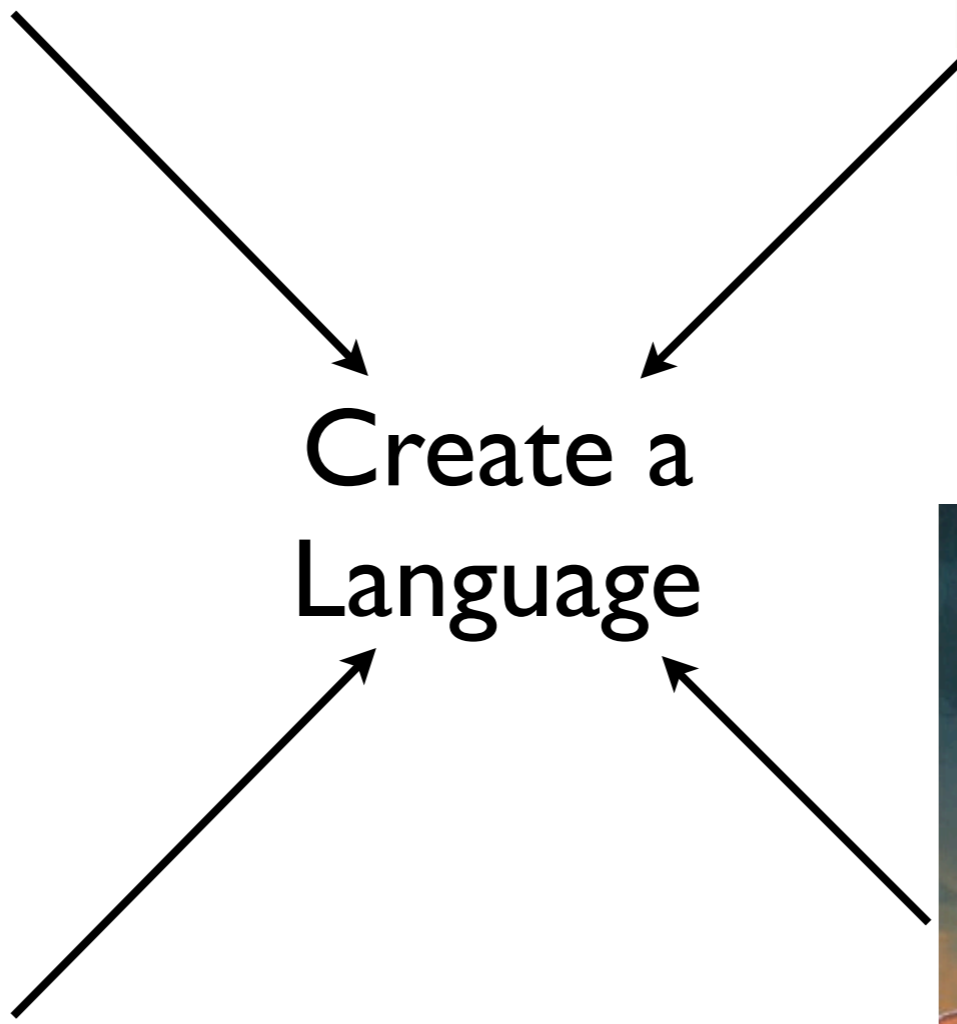
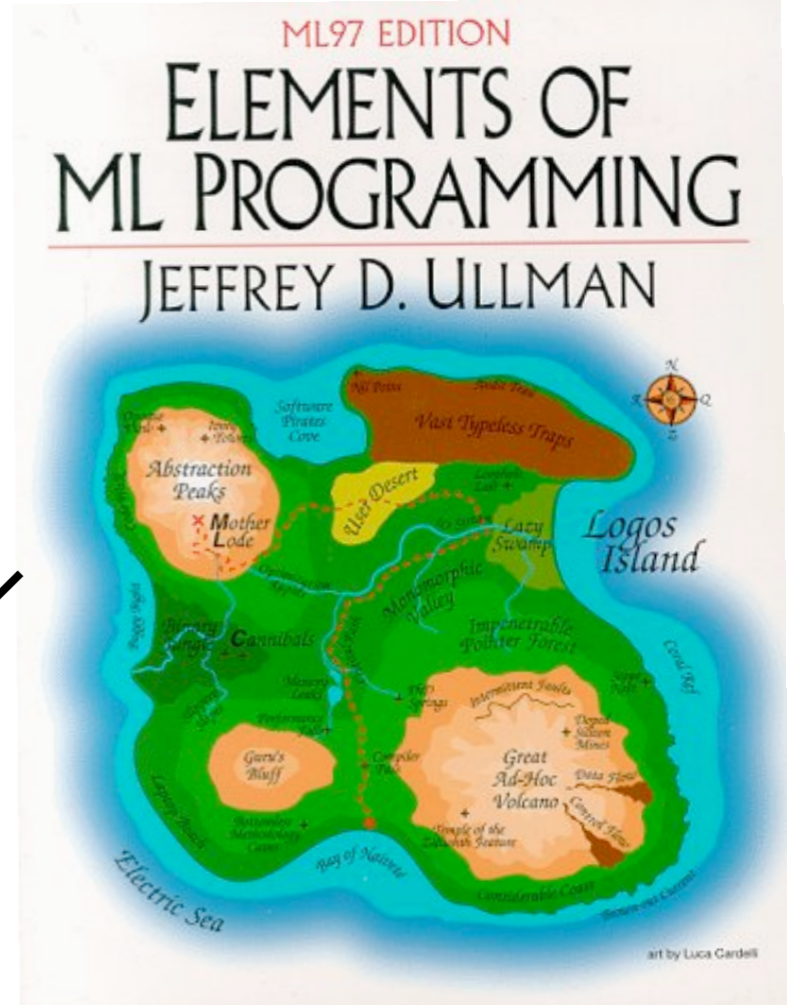
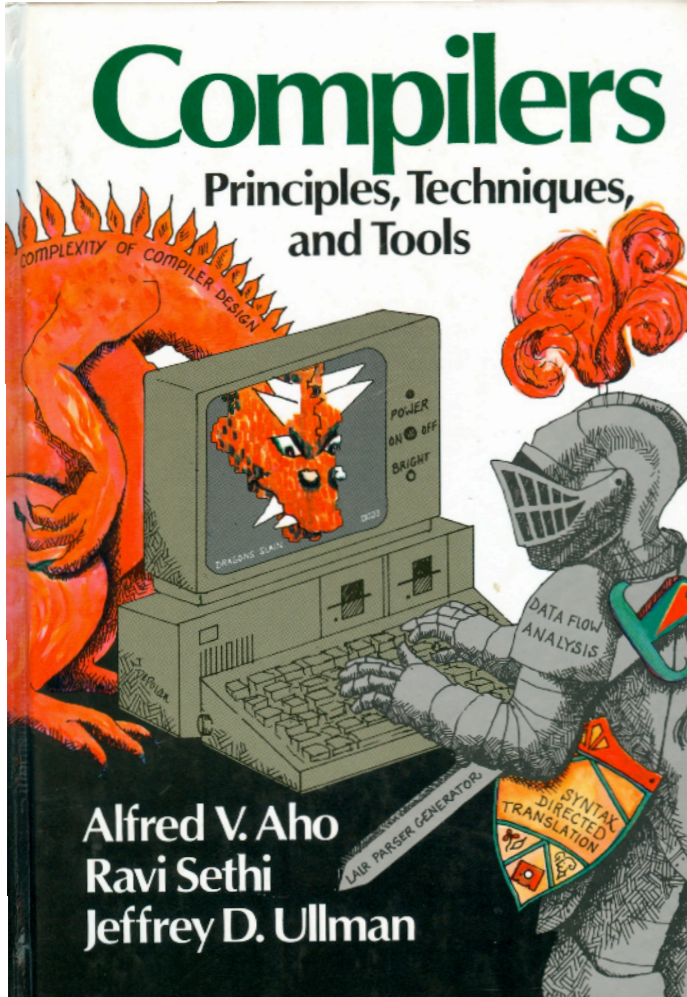
A distracted  
supervisor!





?????????





## 4.2 Arithmetic Expressions

Almost any C-syntax arithmetic expression can be used within the SPaSM interface. In addition to the standard arithmetic operations, the `<math.h>` library is also available.

```
SPaSM [10] > i = 2+(3*4);  
SPaSM [10] > f = sqrt(x*x + y*y + z*z)/3.5;  
SPaSM [10] > r = sin(cos((3+(4+(5+7.3)))));
```

← **A REPL is born!**

## 4.3 Function Evaluation

Functions can be called as in C. Functions may take parameters which may be constants, built-in variables, user-defined variables, or arithmetic expressions.

```
SPaSM [10] > init_lj(1.0,1.0,2.5);  
SPaSM [10] > timesteps(nsteps, energy_n, output_n, checkp_n);  
SPaSM [10] > x = sqrt(y);  
SPaSM [10] > total_particles = count_particles();
```

Functions may return ints, reals, or no value at all. The return values of functions can be used in arithmetic expressions.

## 4.4 Displaying and printing values



## 4.8 Scripts

The following shows a sample scriptfile :

```
! A sample SPaSM input script
! Dave Beazley
! July 31, 1995

totaltime = 10.0;           ! How long to run this simulation (in time)
energy_interval = 0.1;     ! How often to calculate energy (in time)
output_interval = 0.5;    ! How often to dump data (in time)
checkp_interval = 1.0;    ! How often to checkpoint (in time)
Benchmark=1;              ! Report timings
MovieMode=1;              ! Output frame numbers for visualization
FilePath="/sda/sda2/beazley/test";

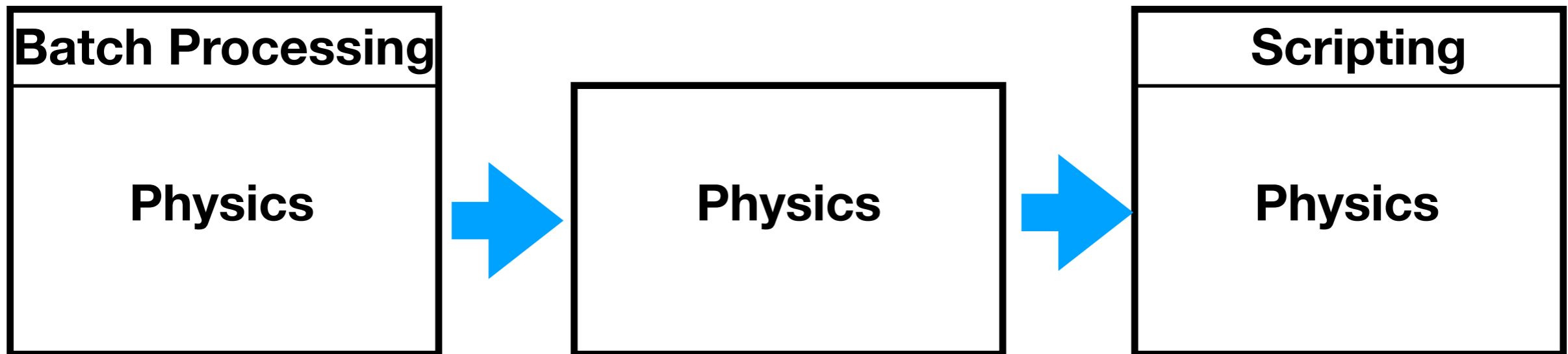
! Initial condition parameters

exdot = 0.0;
eydot = 0.001;
ezdot = 0.0;
lx    = 80;
ly    = 40;
lz    = 10;
lc    = 20;
gapx  = 5.0;
gapy  = 25.0;
gapz  = 5.0;
alpha = 7;
cutoff= 1.7;

! Only set up initial conditions if not restarting.

if (Restart == 0)
    ic_crack(lx,ly,lz,lc, gapx, gapy, gapz, alpha, cutoff);
    set_initial_strain(0,0.017,0);
endif;
```

Don't worry, all of this is "optional."



- It's just a new "main". Move along...

- It fact, it was better than "optional."
- It was all automagically created!

## 6.4 Interface files

The functions available to the interpreter are defined in special interface files (which have a `.i` suffix). An example interface file is shown below :

```
/*
*****
* SPaSM.i
*
* Main SPaSM user interface file.
*
* This file describes the SPaSM variables and functions
* that should be available on all versions of SPaSM.
*
*****/
```

```
%init spasm_interface_init;
%{
```

### 6.4.8 The translator

The files `trans_cm5` and `trans_t3d` are the translator functions for converting `.i` files to `.c` files (the translator is easily compiled for any machine however). The syntax for using these translators is as shown :

```
% trans_cm5 spasm <SPaSM.i
```

This will create a file called `spasm_wrap.c`. The interface file should be directed into `stdin`. See the Makefile for more details.

## 6.4 Interface files

The functions available to the interpreter are defined in special interface files (which have a `.i` suffix). An example interface file is shown below :

```
/******  
* SPaSM.i  
*  
* Main SPaSM user interface file.  
*  
* This file describes the SPaSM variables and functions  
* that should be available on all versions of SPaSM.  
*  
*****/
```

```
%init spasm_interface_init;  
%{
```

### 6.4.8 The translator

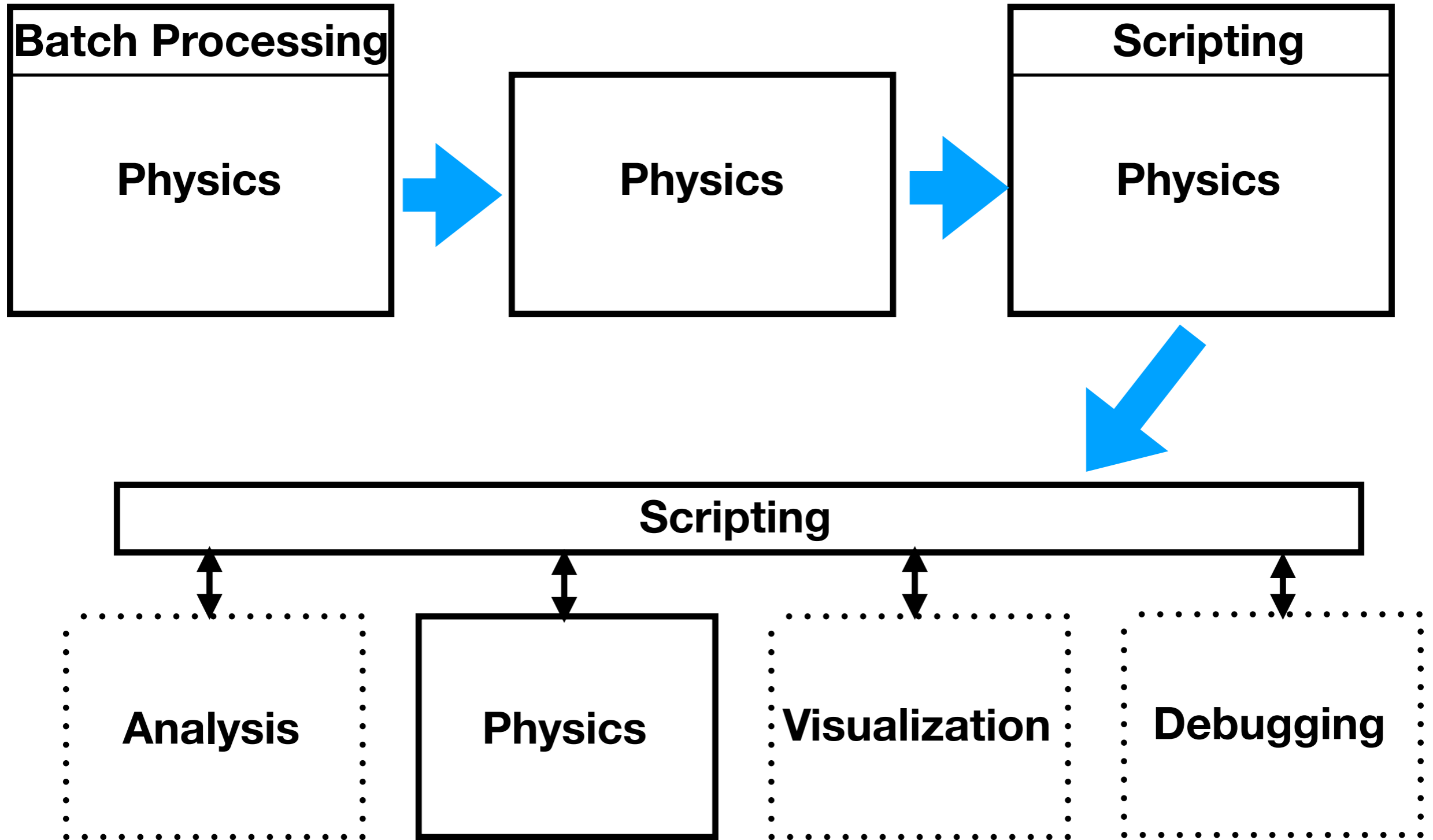
The files `trans_cm5` and `trans_t3d` are the translator functions for converting `.i` files to `.c` files (the translator is easily compiled for any machine however). The syntax for using these translators is as shown :

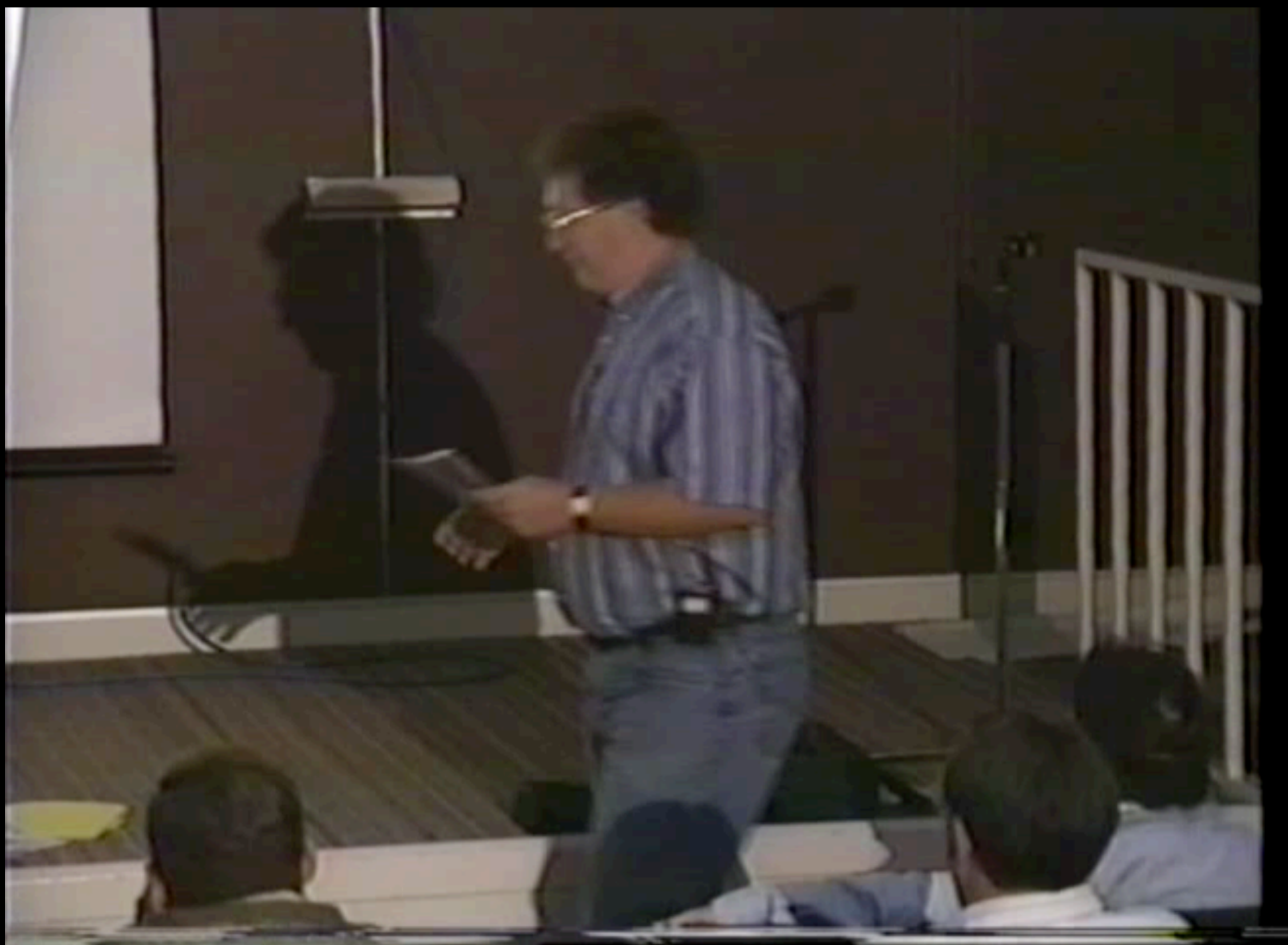
```
% trans_cm5 spasm <SPaSM.i
```

**Automatic Code Generation** ←

This will create a file called `spasm_wrap.c`. The interface file should be directed into `stdin`. See the Makefile for more details.

# The Vision



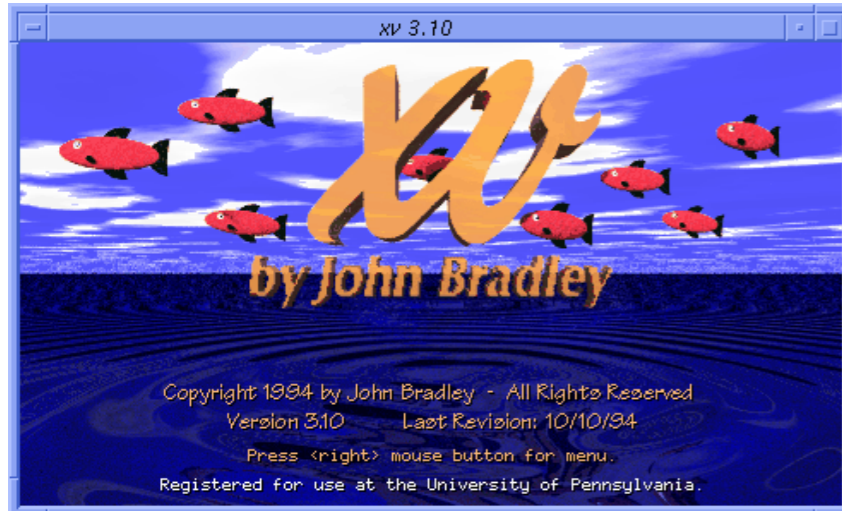


<https://www.youtube.com/watch?v=d5f9-Y3Wtbc>

(Dave describing the system at LLNL in 1996)

# The "Visualization" System

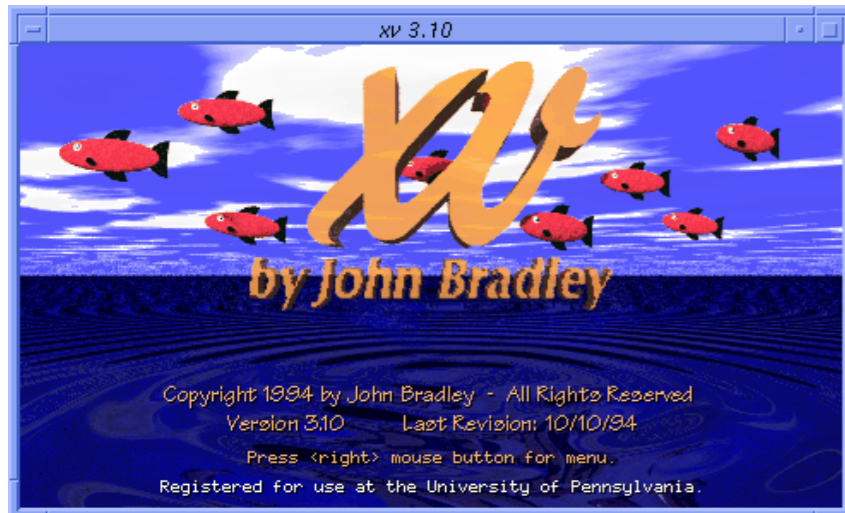
## Workstation



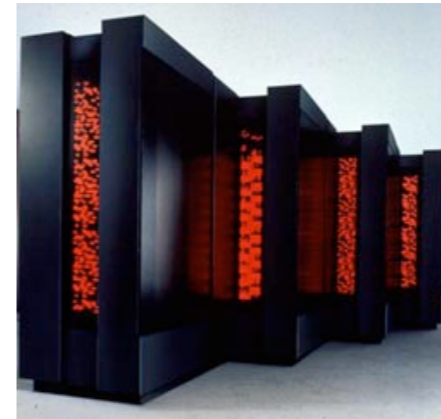


# The "Visualization" System

## Workstation



## Supercomputer

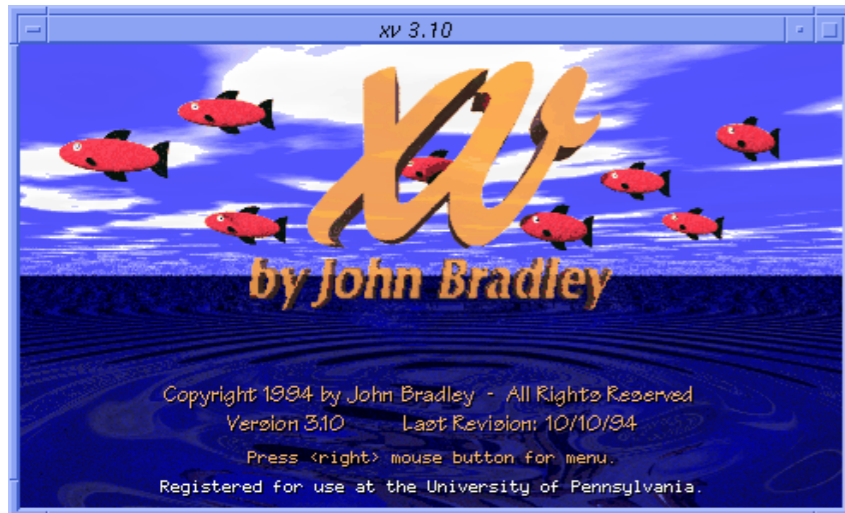


Copy/Paste

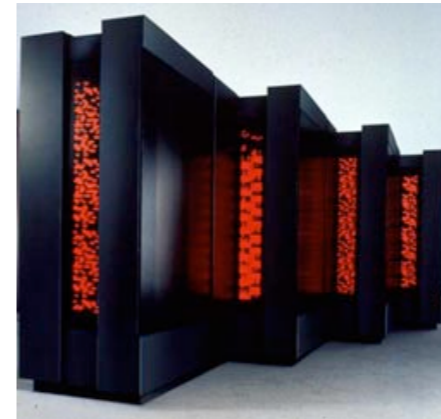
**GIF Encoder**

# The "Visualization" System

## Workstation



## Supercomputer



Copy/Paste

**GIF Encoder**

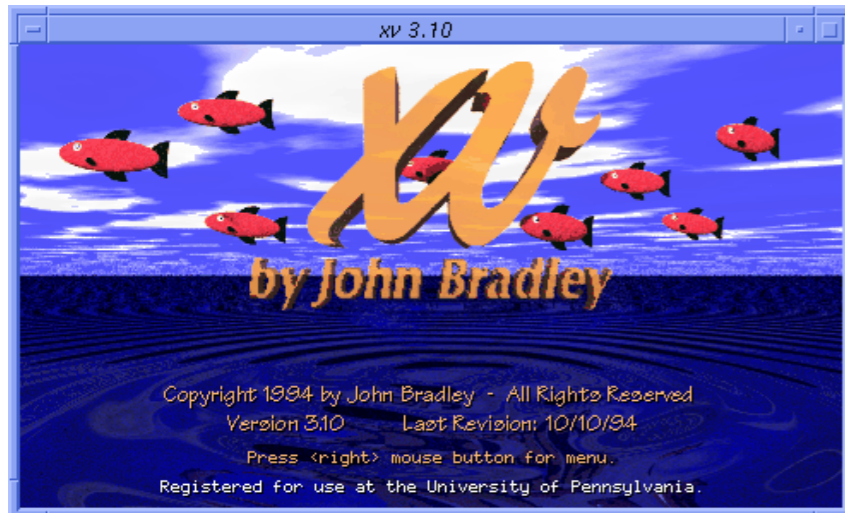
**Graphics**

Adapt &  
Extend

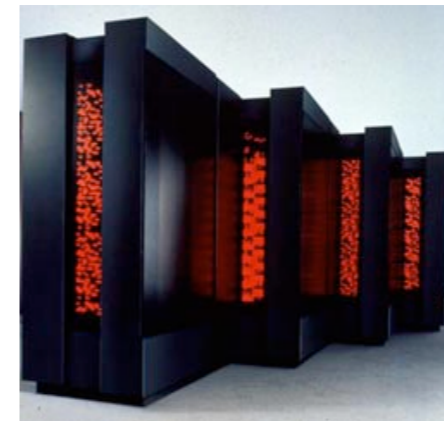
**x86 device  
driver project  
(from 1990)**

# The "Visualization" System

## Workstation



## Supercomputer



Copy/Paste

polling

GIF Encoder

Adapt &  
Extend

Graphics

x86 device  
driver project  
(from 1990)

Image "Server"

Internet

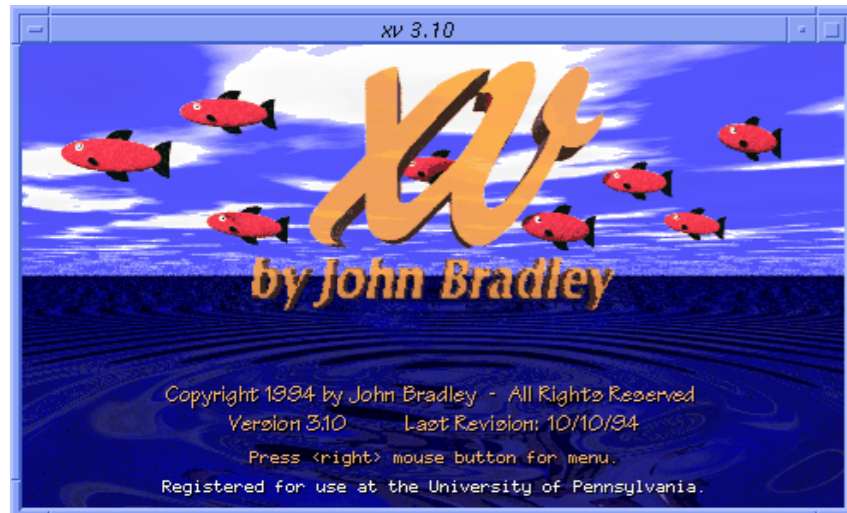
Sockets

GIF Images

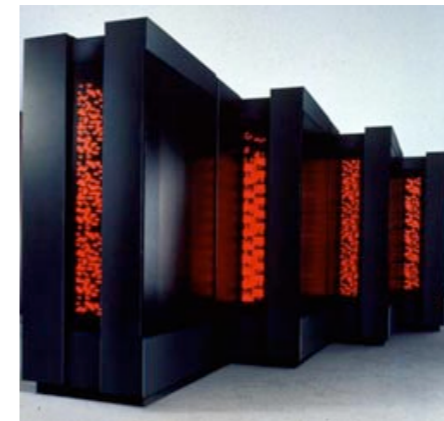


# The "Visualization" System

## Workstation



## Supercomputer



Copy/Paste

polling

Image "Server"

Internet

GIF Images

GIF Encoder

Adapt &  
Extend

Graphics

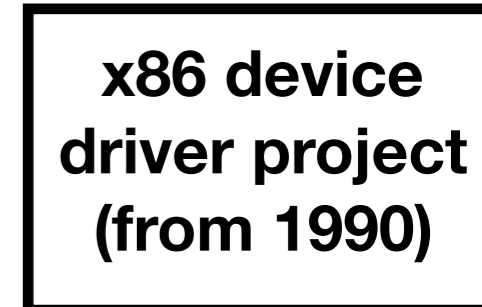
x86 device  
driver project  
(from 1990)

Sockets

Hack

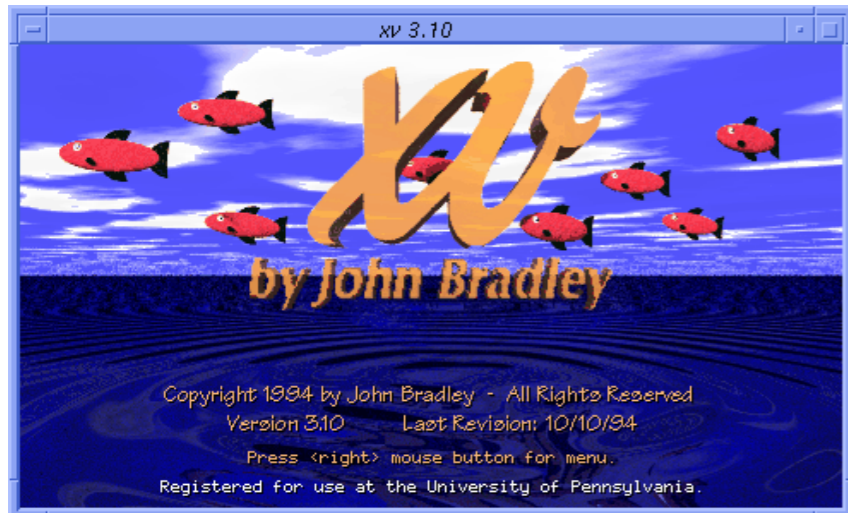
Visualize

"Trackball"  
(OpenGL)

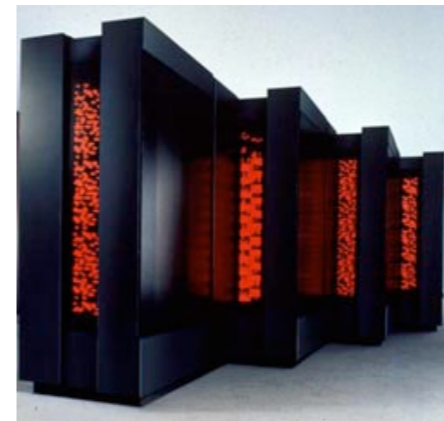


# The "Visualization" System

## Workstation



## Supercomputer



Copy/Paste

polling

Image "Server"

Internet

GIF Images

GIF Encoder

Adapt &  
Extend

Graphics

x86 device  
driver project  
(from 1990)

Sockets

Hack

Visualize

"Trackball"  
(OpenGL)

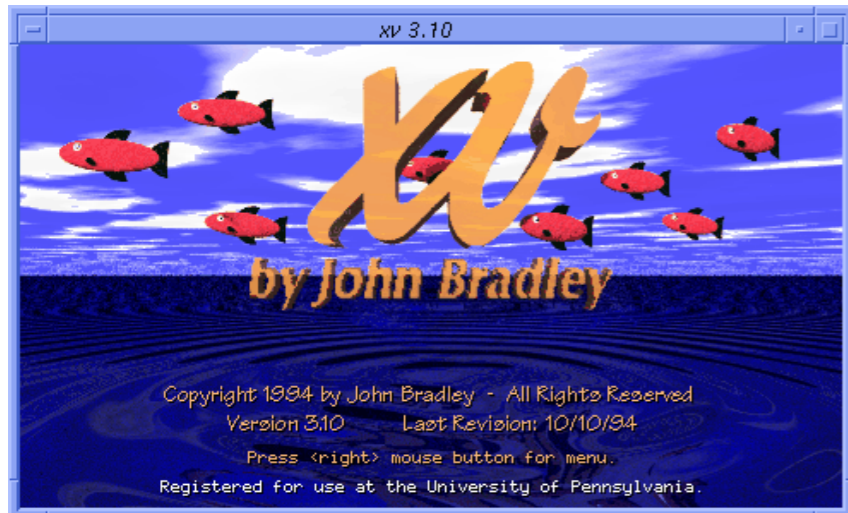
Scripting



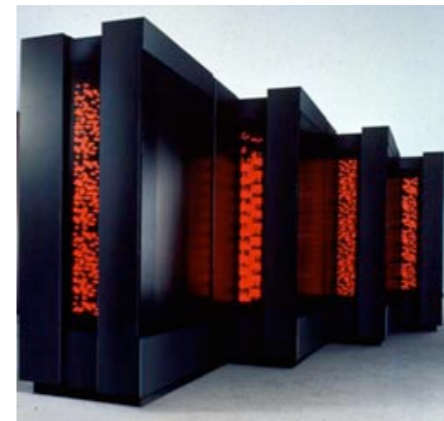
Users ("us")

# The "Visualization" System

## Workstation



## Supercomputer



Copy/Paste

polling

Image "Server"

Internet

GIF Images

GIF Encoder

Adapt &  
Extend

Graphics

x86 device  
driver project  
(from 1990)

Sockets

Hack

Visualize

"Trackball"  
(OpenGL)

Scripting

It was  
awesome!

Users ("us")



# MAKING APPLICATIONS PROGRAMMABLE

**Paul F. Dubois**

**Department Editor:**  
**Paul F. Dubois**  
*dubois1@llnl.gov*

Imagine a physics program written in Fortran, whose user interface is a programming language (not Fortran, of course, but imagine something similar). In this programming language you can create variables, assign values, loop, if-test, define functions, and so on. Additionally, the major variables and functions in the Fortran modules are known to this interface language, so that through the user interface you can assign values to variables, execute Fortran modules, print out the values of variables, plot them, and so on. The Fortran variables and functions are accessible in the user interface language, as if they were built in to it. I call this a "programmable application." This article will explain the general principles behind programmable applications and the benefits of using this approach to scientific programming.

Over the last nine years my team and I have developed and used The Basis System, a collection of software for creating programmable applications. It has been used in over 100 applications and is currently used in some of Lawrence Livermore National Laboratory's most important programs. This experience is the basis for this article, but this article is not about Basis per se.

That is, after initializing, input is obtained that describes a problem from various sources. When the input is complete, the input values are used to calculate the initial value of all the remaining state variables. Then a calculational phase ensues, often the evolution of a time-dependent calculation or an iteration to steady state. Finally, some criterion is satisfied, and the calculation is over. Calculations and other activities may ensue to assist the user in drawing some conclusion about the problem.

At the end of each of these stages there can be various activities such as edits or postprocessing dumps. Sometimes, if the calculation is simple enough, there is a loop around most of the program so that the user can do parameter studies.

When a one- or two-person team develops such a program, it may have an input section that is done with "namelist" input (see below), and the main program often directly reflects the model, possibly with a user-dialog thrown in to control the iteration and examine things as they go along. A namelist input might be used to set flags called revealing names such as `iedit` or `iplot` to select a level of verbosity or types of plots to be done. Often, little output

# MAKING APPLICATIONS PROGRAMMABLE

Paul F. Dubois

Department Editor:  
Paul F. Dubois  
*dubois1@llnl.gov*

Imagine a physics program whose user interface is a programming language you can use to define functions, variables, and functions. In this interface language you can assign values, print out the values. The Fortran variable user interface language is this a "programmable" the general principle and the benefits of programming.

Over the last nine years and used The Basis for creating programs

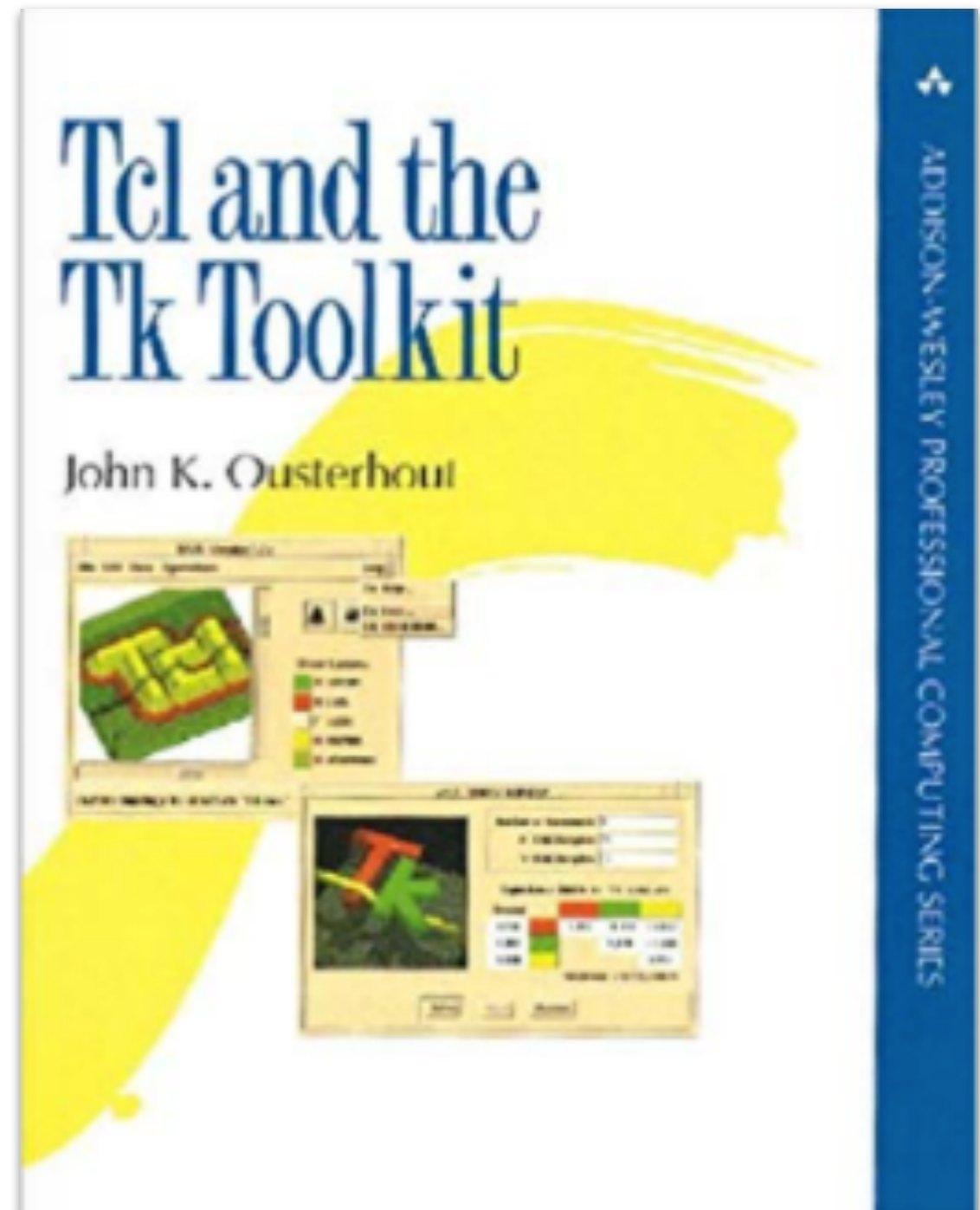
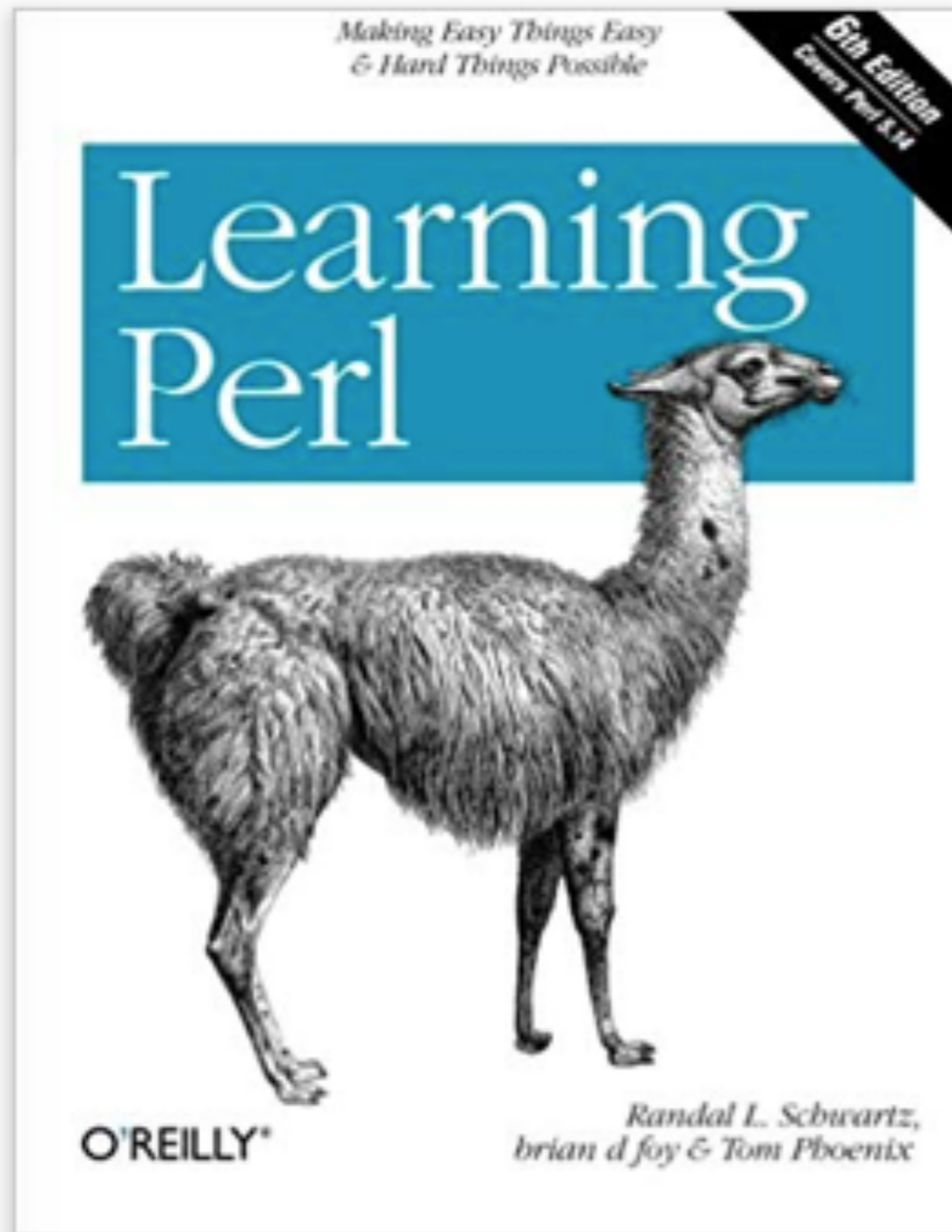
over 100 applications and is currently used in some of Lawrence Livermore National Laboratory's most important programs. This experience is the basis for this article, but this article is not about Basis per se.

"Imagine a physics program written in Fortran, whose user interface is a programming language (not Fortran, of course, but imagine something similar). In this programming language you can create variables, assign values, loop, if-test, define functions, and so on."

thrown in to control the iteration and examine things as they go along. A namelist input might be used to set flags called revealing names such as `iedit` or `iplot` to select a level of verbosity or types of plots to be done. Often, little output



# Other scripting



1996

The stage is set

# 4th International Python Conference

This is a gathering of the Python community, users, developers, and language designers, to discuss the Python programming language's present and future. Welcome!

Included in your registration packet you will find:

- Your payment receipt
- If you prepaid: a badge for the conference - to be worn with the LLNL badge.
- The Agenda
- A List of Attendees
- Information about Accomodations and Social Activities
- Directions and Maps to the conference and accomodations
- Directions to the Social Activities
- The proceedings

Credit is due our gracious hosts, Lawrence Livermore National Laboratory, and the Corporation for National Research Initiatives for helping to sponsor and organize the conference and the Python Software Activity.

For an on-line version of this document, see:  
<<http://www.python.org/workshops/1996-06/packet.html>>

# 4th Inter Python Co

This is a gathering of the Python community, users, and Python programming language's present and future.

Included in your registration packet you will find:

- Your payment receipt
- If you prepaid: a badge for the conference - to
- The Agenda
- A List of Attendees
- Information about Accommodations and Social
- Directions and Maps to the conference and ac
- Directions to the Social Activities
- The proceedings

Credit is due our gracious hosts, Lawrence Livermore National Research Initiatives for helping to sponsor Software Activity.

For an on-line version  
<<http://www.python.org/wo>

## WEDNESDAY, JUNE 5TH CONFERENCE PROGRAM CONTINUED

### SYSTEMS & INTEGRATIONS

- 9:00 - 9:30 Using Emacs OO-Browser with Python  
Harri Pasanen
- 9:30 - 10:00 Python Interface with Narcisse Graphics  
Zane Motteler
- 10:00 - 10:30 Gist: A Scientific Graphics Package for Python  
Lee Busby
- 10:30 - 11:00 Building a Programmable Interface for Physics Codes Using  
Numeric Python  
Tser-Yuan (Brian) Yang, P. F. Dubois, Zane Motteler
- 11:00 - 11:30 Building Ariel - OpenGL GUI and Python to C  
Jim Hugunin
- 12:00 - 2:00 *Luncheon at Retzlaff Winery*

---

### STATE OF THE ART

- 2:00 - 2:30 Extensibility in Python  
Manus Hand
- 2:30 - 3:00 Using SWIG to Control, Prototype, and Debug C Programs  
with Python  
David M. Beazley

4th Intern

# Gist: A Scientific Graphics Package for Python

Lee E. Busby

## Abstract

"Gist" is a scientific graphics library written by David H. Munro of Lawrence Livermore National Laboratory (LLNL). It features support for three common graphics output devices: X Windows, (Color) PostScript, and ANSI/ISO Standard Computer Graphics Metafiles (CGM). The library is small (written directly to Xlib), portable, efficient, and full-featured. It produces X versus Y plots with "good" tick marks and tick labels, 2-dimensional quadrilateral mesh plots with contours, vector fields, or pseudo color maps on such meshes, with 3-dimensional plots on the way.

The Python Gist module utilizes the new "Numeric" module due to J. Hugunin and others. It is therefore fast and able to handle large datasets. The Gist module includes an X Windows event dispatcher which can be dynamically added (e.g., via importing a dynamically loaded module) to the Python interpreter after a simple two-line modification to the Python core. This makes fast mouse-controlled zoom, pan, and other graphic operations available to the researcher while maintaining the usual Python command-line interface.

Munro's Gist library is available at <http://www.llnl.gov>

- The proceedings

Credit is due our gracious hosts, Lawrence Livermore National Research Initiatives for helping to sponsor Software Activity.

For an on-line version  
<<http://www.python.org/wo>

## STATE OF THE ART

2:00 - 2:30 Extensibility in Python

Manus Hand

2:30 - 3:00 Using SWIG to Control, Prototype, and Debug C Programs with Python

David M. Beazley

5TH  
GRAM

Python

graphics

for Python

for Physics Codes Using

dis, Zane Motteler

Python to C

Python

Phone: 707-438-2300

Fax: 707-438-4813

10000 Research

Post Office

Lawrence Livermore Lab

Phone: 415-438-3420

Fax: 415-438-4800

http://www.llnl.gov

Software Communications

Phone: 415-438-4750 x212

Fax: 415-438-4800

http://www.llnl.gov

Software Communications

Phone: 415-438-4750 x212

Fax: 415-438-4800

http://www.llnl.gov

Software Communications

4th Intern

# Package for Python

THE 5TH PROGRAM

## Gist: A Sci

### Abstract

"Gist" is a scientific Laboratory (LLNL) PostScript, and AN directly to Xlib), p marks and tick lab color maps on suc

The Python Gist fast and able to can be dynamic after a simple t and other graphic operations command-line interface.

Munro's Gist library is a Proxy connections to the Social Activities

- The proceedings

Credit is due our gracious hosts, Lawrence Liverm National Research Initiatives for helping to sponsor Software Activity.

For an on-line version  
<<http://www.python.org/wo>

# A Python Interface with Narcisse Graphics

Zane C. Motteler

Narcisse	Curve, Surface and Mesh Objects	Graph Objects	Plotter Class	narcissemodule	Examples
----------	---------------------------------	---------------	---------------	----------------	----------

## Narcisse

Narcisse is a graphics package developed by our French colleagues at Centre d'Etudes de Limeil-Valenton of the Commissariat d'Energie Atomique. Narcisse is quite comprehensive; it can do two-, three-, and four-dimensional plots (the latter meaning that the surface is colored according to the values of an arbitrary function). One can open and send plots to a Narcisse window on a distant machine (handy if you want to show the boss a nice plot but you don't want to run over to her building with a hard copy).

### STATE OF THE ART

- 2:00 - 2:30 Extensibility in Python  
Manus Hand
- 2:30 - 3:00 Using SWIG to Control, Prototype, and Debug C Programs with Python  
David M. Beazley

4th Intern

Package for Python

# *Building a Programmable Interface for Physics Codes Using Numeric Python*

T.-Y. B. Yang, P. F. Dubois, and Z. C. Motteler

Lawrence Livermore National Laboratory

## 1.0 Introduction

Our goal is to create a "plug and play" programmable interface that gives the users flexibility to run the applications in the way appropriate for their physics problems, and also allows the code developers to query and to change, from the Python interpreter, variables buried in the physics modules, which for speed reasons are implemented in C, C++, and Fortran, and to execute compiled modules. Some of the philosophy behind such programmable applications was presented elsewhere.<sup>1</sup> The programmable applications with Python interface, from the bottom up as shown in Fig. 1, consists of the following four levels:

ics

Examples

can do  
to the  
machine  
th a

grams

Credit is  
Nationa  
Softwar

# NUMERICAL PYTHON

**Paul F. Dubois, Konrad Hinsien,  
and James Hugunin**

**Department Editor:**  
**Paul F. Dubois**  
*dubois1@llnl.gov*

**P**ython is a small and easy-to-learn language with surprising capabilities. It is an interpreted object-oriented scripting language and has a full range of sophisticated features such as first-class functions, garbage collection, and exception handling. Python has properties that make it especially appealing for scientific programming:

- Python is quite simple and easy to learn, but it is a full and complete language.
- It is simple to extend Python with your own compiled objects and functions.
- Python is portable, from Unix to Windows 95 to Linux to Macintosh.
- Python is free, with no license required even if you make a commercial product out of it.
- Python has a large user-contributed library of “modules.” These modules cover a wide variety of needs, such as audio

benchmarks for the basic language and the numerical extension are available on the Python Web site (<http://www.python.org>).

The numerical extension is still in beta test and may therefore change slightly from this description. In particular, the beta-test period is needed to sort out some controversies in naming and coercion rules. But with this tutorial as a start and the latest readme file for the numerical extension, you should be able to start using it. Note that you will need to add the extension to the Python source; every effort is made to keep a “minimal Python” as small as possible, as Python is being used for applications where a small size is important.

Python is extremely well suited to the development of programmable applications, as has been advocated on these pages (CIP 8:1, 1994, p. 70). It has a scripting language as the user interface and compiled code for the compute-intensive portions.

## Introducing Python

Python is an interpreter. You can either enter commands directly into the interpreter or, more commonly, create a file containing a script. On Unix, you can invoke Python with the script as the first argument, or you can use the usual trick of starting the script with a comment like this:

```
#!/usr/local/bin/python
```

Then you give execute permission to the script file. When you execute it, the Python interpreter is invoked on the script file itself. Since the above line is a comment as far as Python is concerned, it is then ignored.




# SWIG: Convert C declarations to Python wrappers.

## SWIG Interface file

```
%module SPaSM
%{
#include "SPaSM.h"
%}

void memory(int natoms);
void geometry(double ...
void processors(int ...
void set_boundary_period
...
```

SWIG  


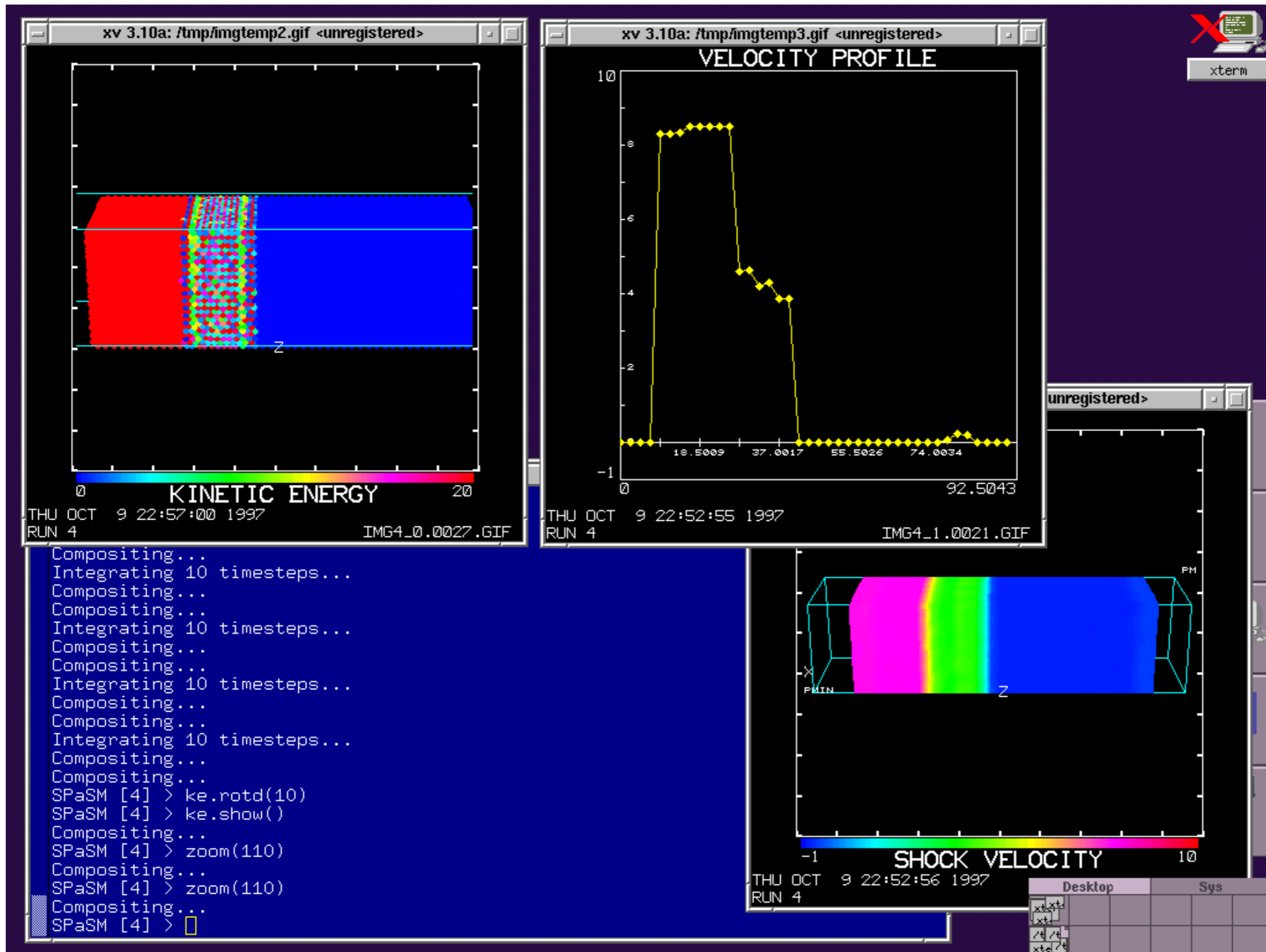
## Python

```
>>> from SPaSM import *
>>> memory(20000)
>>> geometry(0,0,0,80,80,
...          160,2.0)
...

```

Released as open-source in February, 1996

# SPaSM: Now Running under Python (1996)



Why

Python?

# Syntax

```
# Shock wave problem
```

```
nx          = 15
ny          = 15
nz          = 50
shock_velocity = 8.5
temp        = 0.01
width       = 0.3333      # Width is percent of total z length
r0          = 1.0901733   # Lattice spacing
gap         = 0.10       # Gap (% of z length)
cutoff      = 2.0        # Interaction cutoff
cvar.Dt     = 0.0025     # Timestep
cvar.Benchmark = 1
```

```
ic_shock(nx,ny,nz,shock_velocity,width,gap,temp,r0,cutoff)
```

```
init_lj(1,1,cutoff)
```

```
set_boundary_periodic()
```

```
set_path("./Data")
```

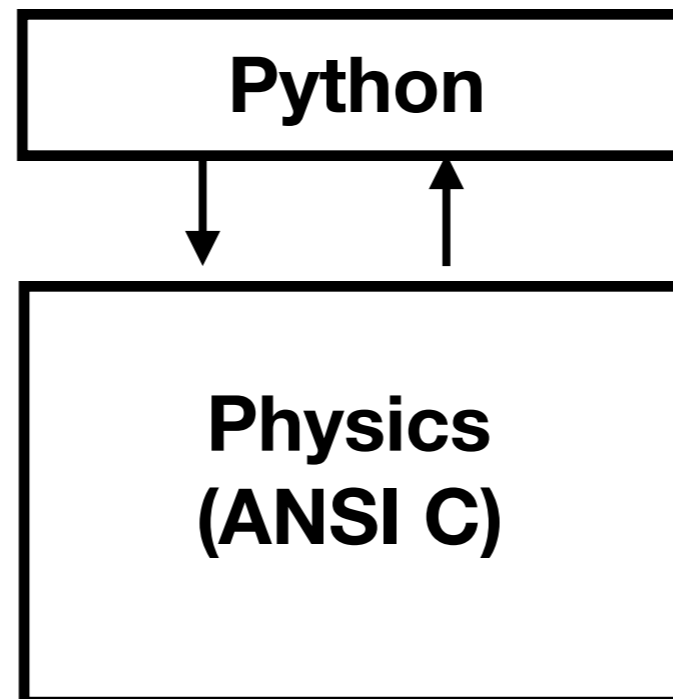
```
timesteps(10000,25,25,500)
```

# The REPL

```
Python 3.8.0b4 (default, Sep  2 2019, 12:54:09)
[Clang 9.0.0 (clang-900.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> print("Hello World")
Hello World
>>> 37 + 42
79
>>>
```

**Interactive Experimentation was Critical**

# C Extensions



Python could talk to existing C code (via Swig, etc.)

# Abstractions

Python was flexible, yet pragmatic

- Imperative
- Functional
- Object-oriented

You could adapt Python to your problem  
(as opposed to the other way around).

# Loose Coupling

```
class Bike:  
    def __turn__(self):  
        ...
```





# Implementation

- Python was written in ANSI C
- Few dependencies
- Avoided advanced implementation "tricks"
- You could actually understand it

*Python was easily "hackable", which also meant "readable", "fixable" and "adaptable." We were often running on esoteric hardware with basically no support. If you wanted something to work, you were on your own.*

# Performance

- Parallel machines had 1000s of processing nodes, each with limited memory
- Each one had a copy of Python
- Python was relatively small
- Little impact on existing code

*This point was critical to selling people on the idea.*

# It Suited Us

*"A scientific program is usually the product of one or two people, who write it initially to solve a class of problems faced by themselves and perhaps a few friends. It is much rarer for a decision to be made early to write a large program; rather, the programs that prove to be useful are added to, and evolve into, large programs over time."*

**-- Paul Dubois**

# The Present

# A wide range of applications have been studied with SPaSM: 1993-2012 covers



# SPaSM is still in use (2017)

Acta Materialia 126 (2017) 313–328



Contents lists available at ScienceDirect

Acta Materialia

journal homepage: [www.elsevier.com/locate/actamat](http://www.elsevier.com/locate/actamat)



Full length article

## On the ultimate tensile strength of tantalum

Eric N. Hahn <sup>a, b, \*</sup>, Timothy C. Germann <sup>b</sup>, Ramon Ravelo <sup>c, d</sup>, James E. Hammerberg <sup>c</sup>,  
Marc A. Meyers <sup>a</sup>



<sup>a</sup> Materials Science and Engineering Program, University of California San Diego, La Jolla, CA, 92093, USA

<sup>b</sup> Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM, 94550, USA

<sup>c</sup> X-Computational Physics Division, Los Alamos National Laboratory, Los Alamos, NM, 94550, USA

<sup>d</sup> Physics Department and Materials Research Institute, University of Texas, El Paso, TX, 79968, USA

### ARTICLE INFO

#### Article history:

Received 14 September 2016

Received in revised form

9 December 2016

Accepted 14 December 2016

Available online 10 January 2017

#### Keywords:

Tensile strength

Spall

Non-equilibrium molecular dynamics

Tantalum

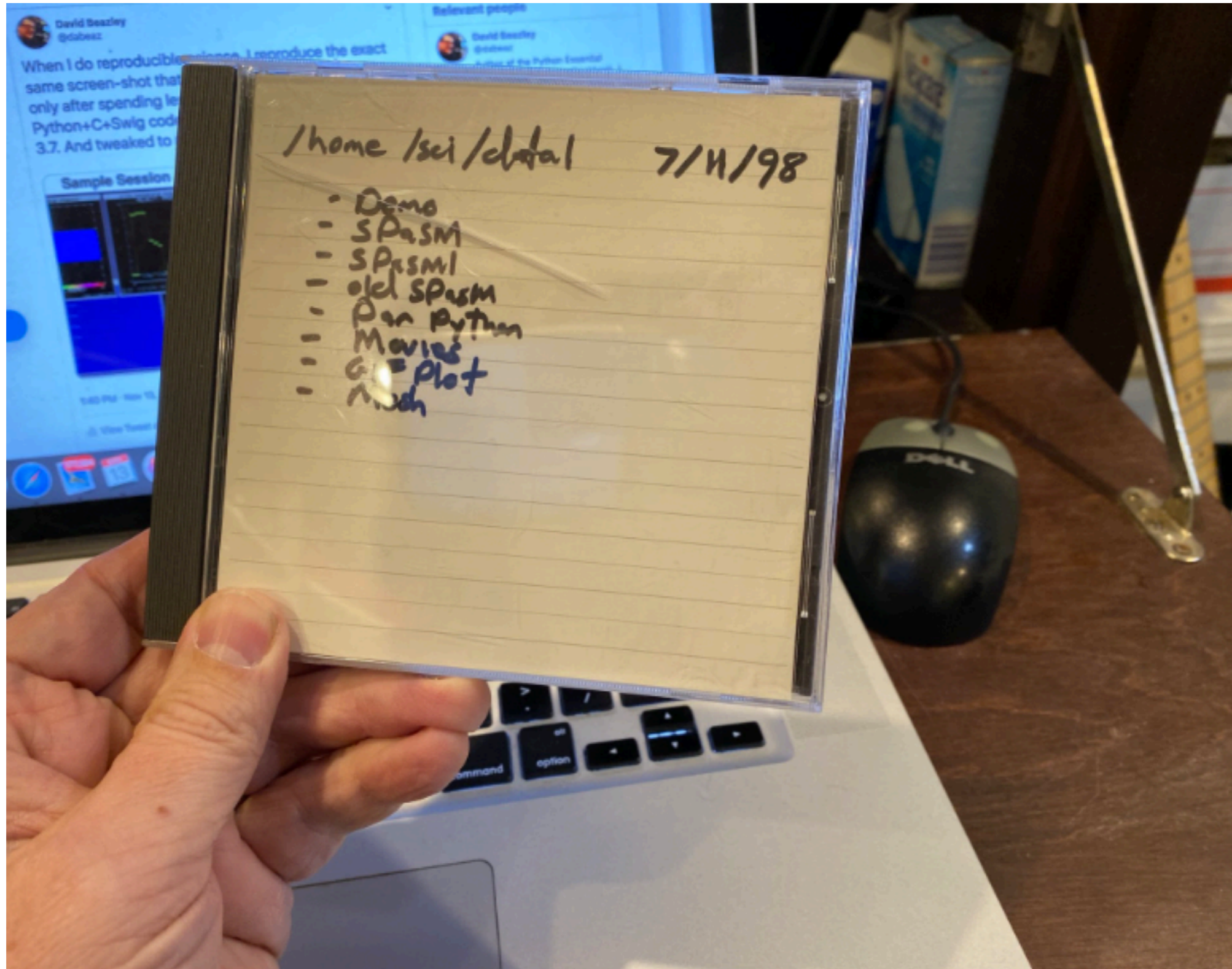
### ABSTRACT

Strain rate, temperature, and microstructure play a significant role in the mechanical response of materials. Using non-equilibrium molecular dynamics simulations, we characterize the ductile tensile failure of a model body-centered cubic metal, tantalum, over six orders of magnitude in strain rate. Molecular dynamics calculations combined with reported experimental measurements show power-law kinetic relationships that vary as a function of dominant defect mechanism and grain size. The maximum sustained tensile stress, or spall strength, increases with increasing strain rate, before ultimately saturating at ultra-high strain rates, i.e. those approaching or exceeding the Debye frequency. The upper limit of tensile strength can be well estimated by the cohesive energy, or the energy required to separate atoms from one another. At strain rates below the Debye frequency, the spall strength of nanocrystalline Ta is less than single crystalline tantalum. This occurs in part due to the decreased flow stress of the grain boundaries; stress concentrations at grain boundaries that arise due to compatibility requirements; and the growing fraction of grain-boundary atoms as grain size is decreased into the nanocrystalline regime. In the present cases, voids nucleate at defect structures present in the microstructure. The exact makeup and distribution of defects is controlled by the initial microstructure and the plastic deformation during

# Alas...

- It does not always use its Python interface
- Supercomputing is a strange world
- Machines often change
- At one point, the CPUs got smaller and the OS eliminated support for dynamic linking
- Created a build/deployment problem
- Also: Challenges in porting to Python versions

# Can I make it work?





# Demo

*I ported SPaSM from 1998 (Python 1.4) to 2019  
(Python 3.8) just for this talk.*

*It runs on my laptop.*

*It took about a day.*

**How was this even  
possible?**

# Modular DIY

*The Python environment made us think more about program structure. Modularity, relationships between components, portability, and other software matters. But not in a heavy-handed manner.*

*Our goal was never to create a "software product."*

*SPaSM was not developed as software product*

# Habitability

*Think of a farmhouse. It starts as a small home with a barn out back. As the family grows and the needs of the farm grow, a back room is added to the house, then a canning room, then a room for grandma; stables are added to the barn, then a wing for milking more cows. Finally the house and barn are connected because it is too difficult to get from the house to the barn in a blizzard. The result is rambling, but each part is well-suited to its needs, each part fits well with the others, and the result is beautiful because it is a living structure with living people inside. The inhabitants are able to modify their environment because each part is built according to familiar patterns of design, use, and construction and because those patterns contain the seeds for piecemeal growth.*

**-- Richard Gabriel (Patterns of Software)**

The  
Future?

*"Much of scientific programming is exploratory in nature, and for that sort of programming the use of compiled languages will cease. Interpreters will simply be fast enough for most such calculations. More computationally intensive programs will be written as extensions of interpreted environments."*

- Paul Dubois (Computers in Physics, Mar. 1997)

*Is Python's current ecosystem of "scientific" tools the realization of that future?*

*Is Python's current ecosystem of "scientific" tools the realization of that future?*

*(honest answer, I don't know)*



*Is Python's current ecosystem of "scientific" tools the realization of that future?*

*(honest answer, I don't know)*

*(a different talk perhaps...)*

# Thanks!

## Acknowledgments

- Guido van Rossum
- Peter Lomdahl (LANL)
- Tim Germann (LANL)

## Me:

- <http://www.dabeaz.com>
- @dabeaz (Twitter)