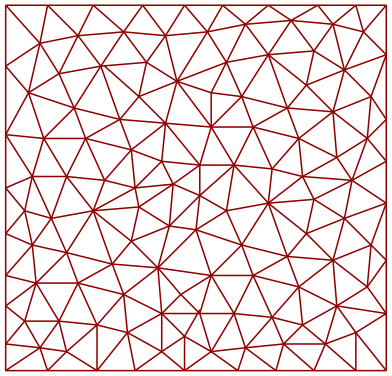
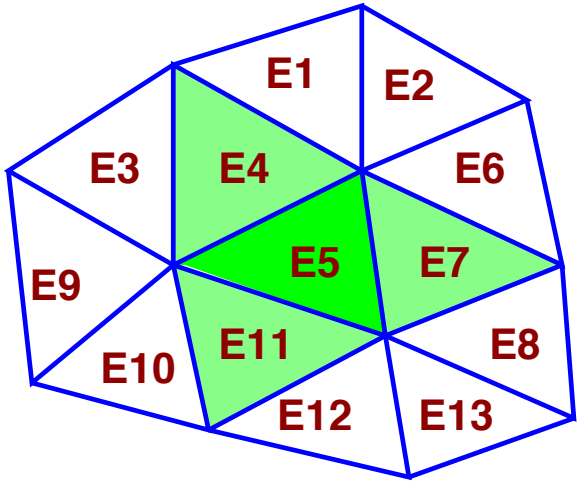
Finite Element Methods

- Simulate physical phenomena governed by partial differential equations
- Most real-world problems do not have an analytical solution
  - must be computed numerically
- Compute approximate solutions within rigid, provable error bounds
- Discretize the domain into "elements"
- Elements form the "mesh"
- Solve on each element, "paste together" to obtain solution



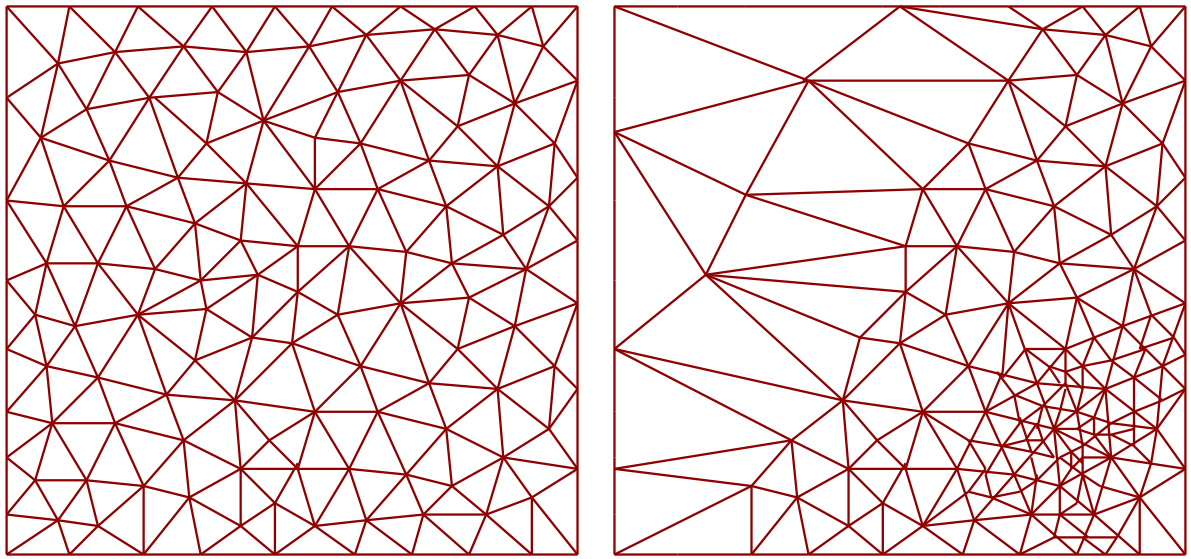
- Solution at each step on an element typically depends on the previous values at that element and neighboring elements



For example, at time $t+1$, value at `E5` may depend on the time t value of `E5` and the time t values of its immediate neighbors `E4`, `E7`, and `E11`.

Adaptive Methods

- More elements $\Longrightarrow$ better accuracy, but higher cost
- Adaptivity concentrates computational effort where it is needed
- Guided by error estimates or error indicators
- h-adaptivity: mesh enrichment



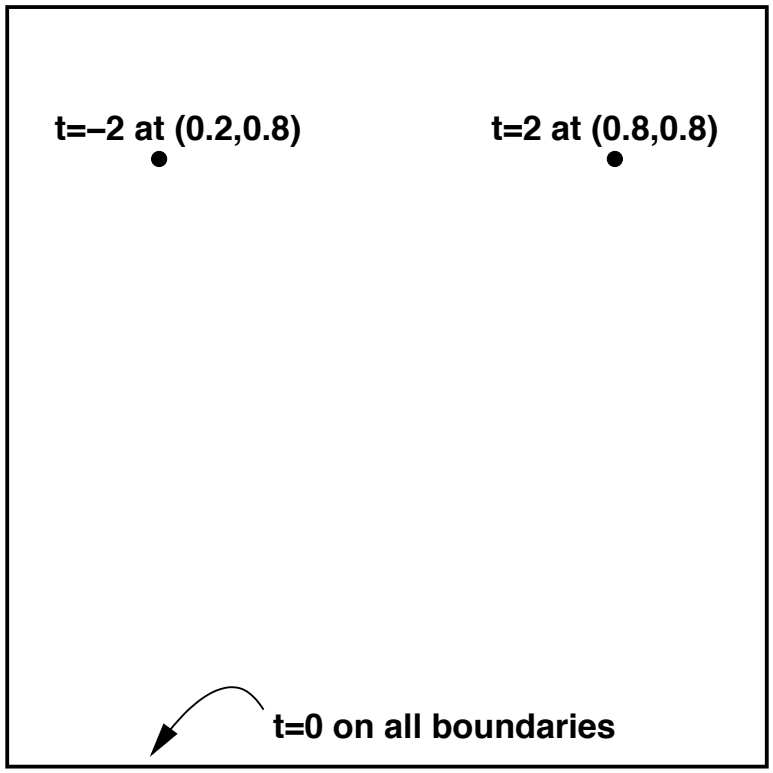
Uniform mesh Adapted mesh

- p-adaptivity: method order variation; r-adaptivity: mesh motion
- Local refinement method: time step adaptivity
- Adaptivity is essential

A Simple Adaptive Computation

Straightforward heat equation solution using Jacobi iteration

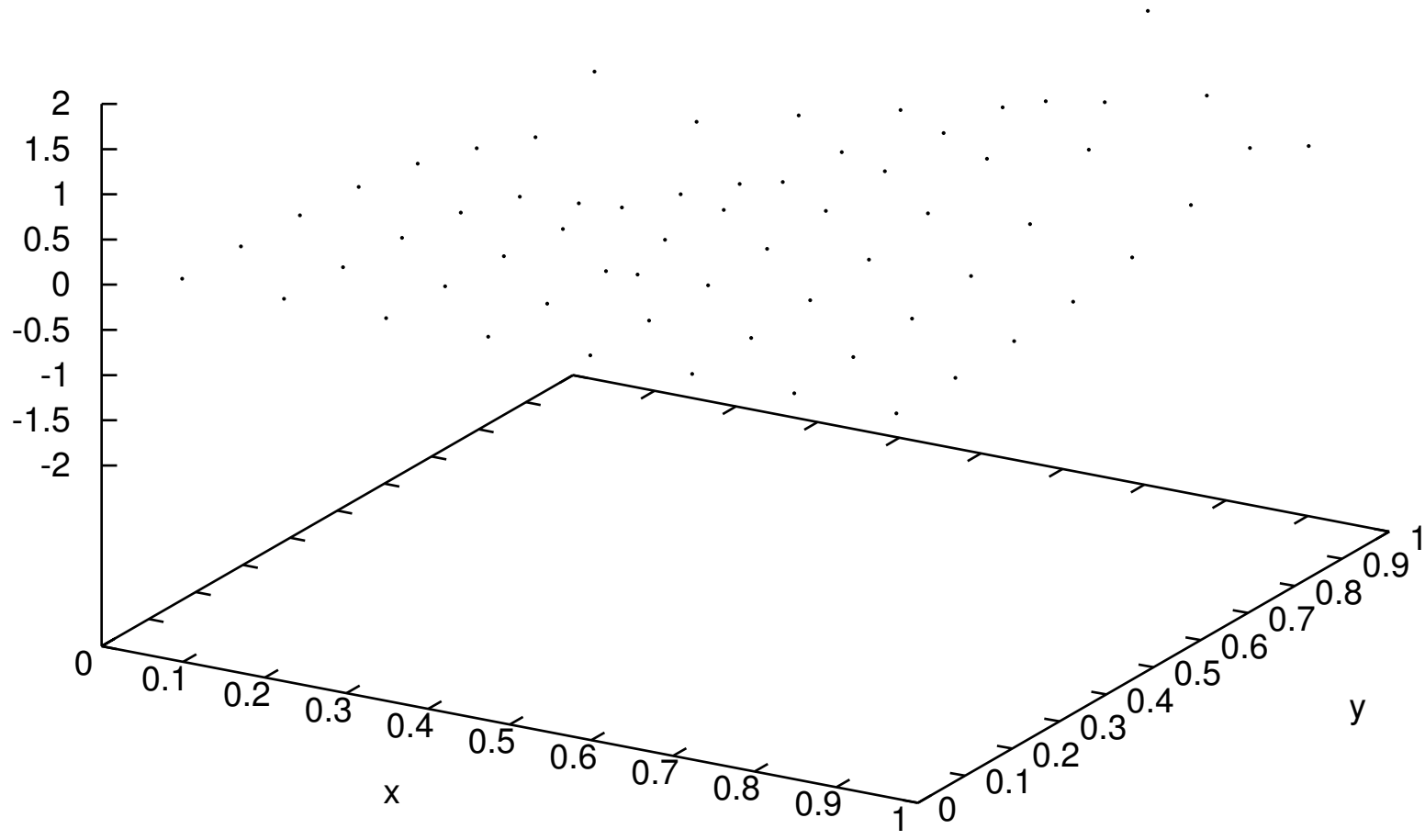
- Problem setup – unit square, all boundaries have temperature 0, fixed heat source with temperature 2 at $(\frac{4}{5}, \frac{4}{5})$ and temperature -2 at $(\frac{1}{5}, \frac{4}{5})$



- Approach: at each iteration, the new solution value is the average of neighbors' solution values
- By no means is this the most efficient technique, but it is simple enough to understand and demonstrates the important ideas

A Simple Adaptive Computation

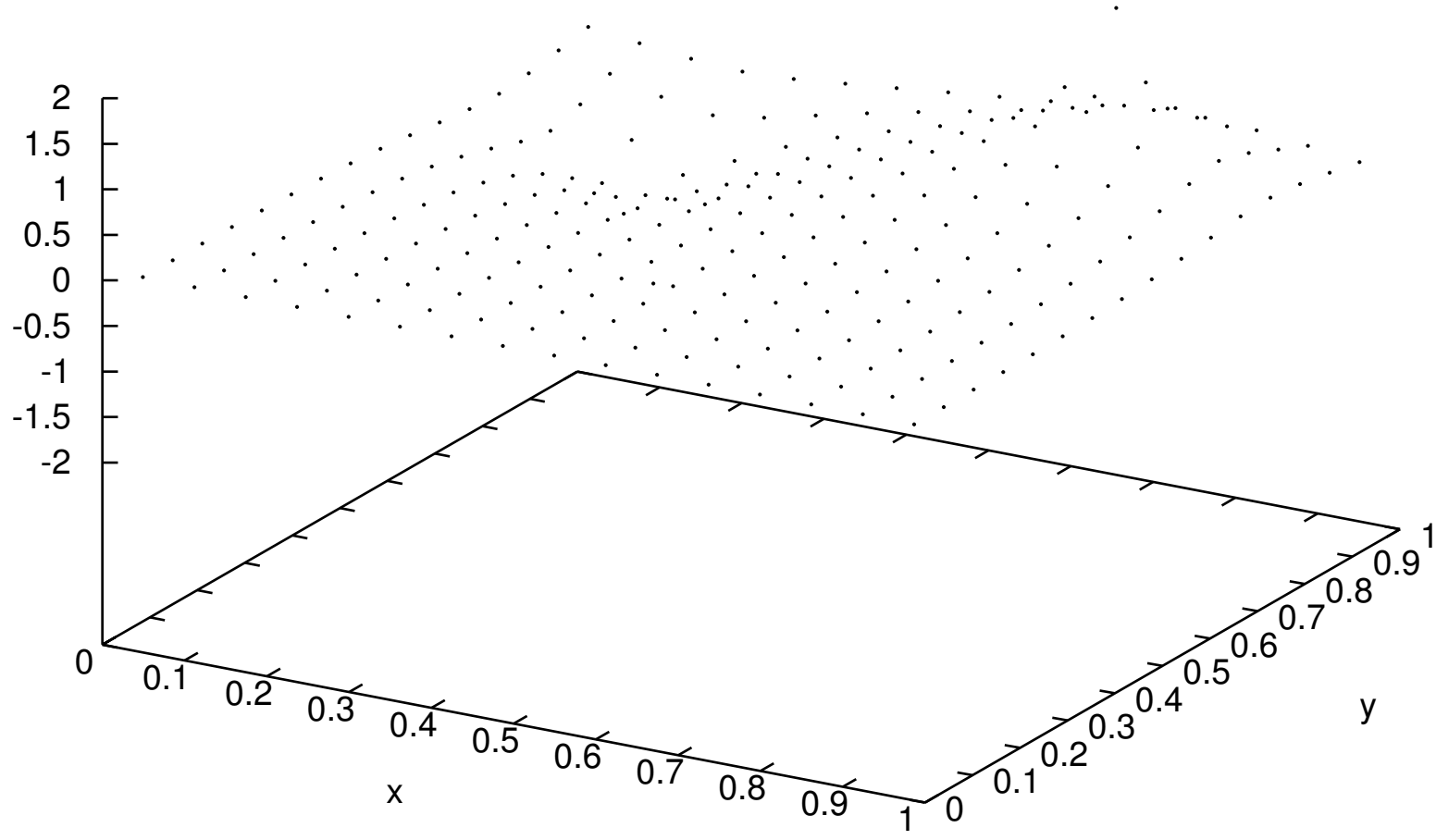
The number of solution points determines how accurate the solution can be.



Uniform distribution of 64 points, 74 iterations, total 4736 steps

A Simple Adaptive Computation

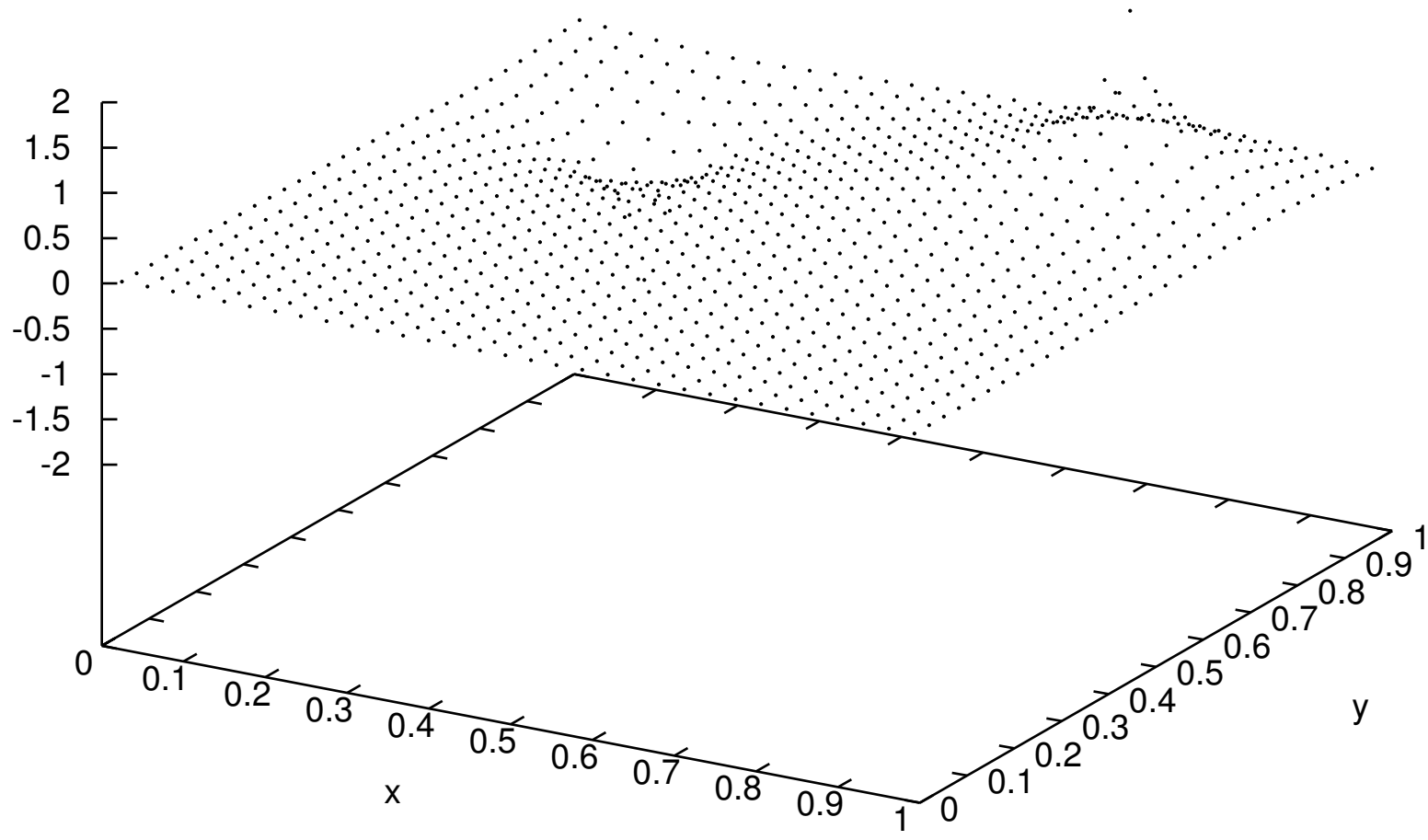
The number of solution points determines how accurate the solution can be.



Uniform distribution of 256 points, 168 iterations, total 43,008 steps

A Simple Adaptive Computation

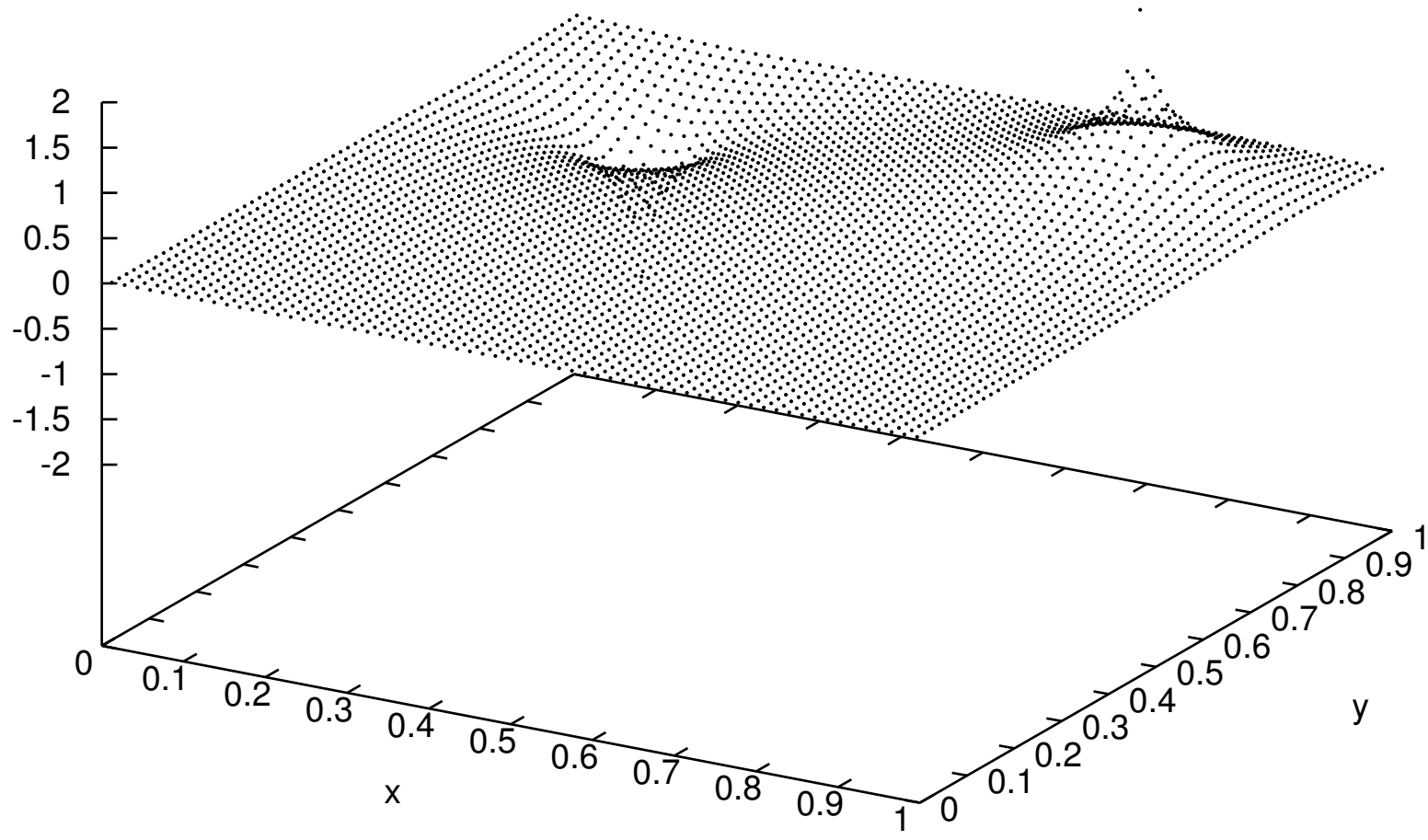
The number of solution points determines how accurate the solution can be.



Uniform distribution of 1024 points, 338 iterations, total 346,112 steps

A Simple Adaptive Computation

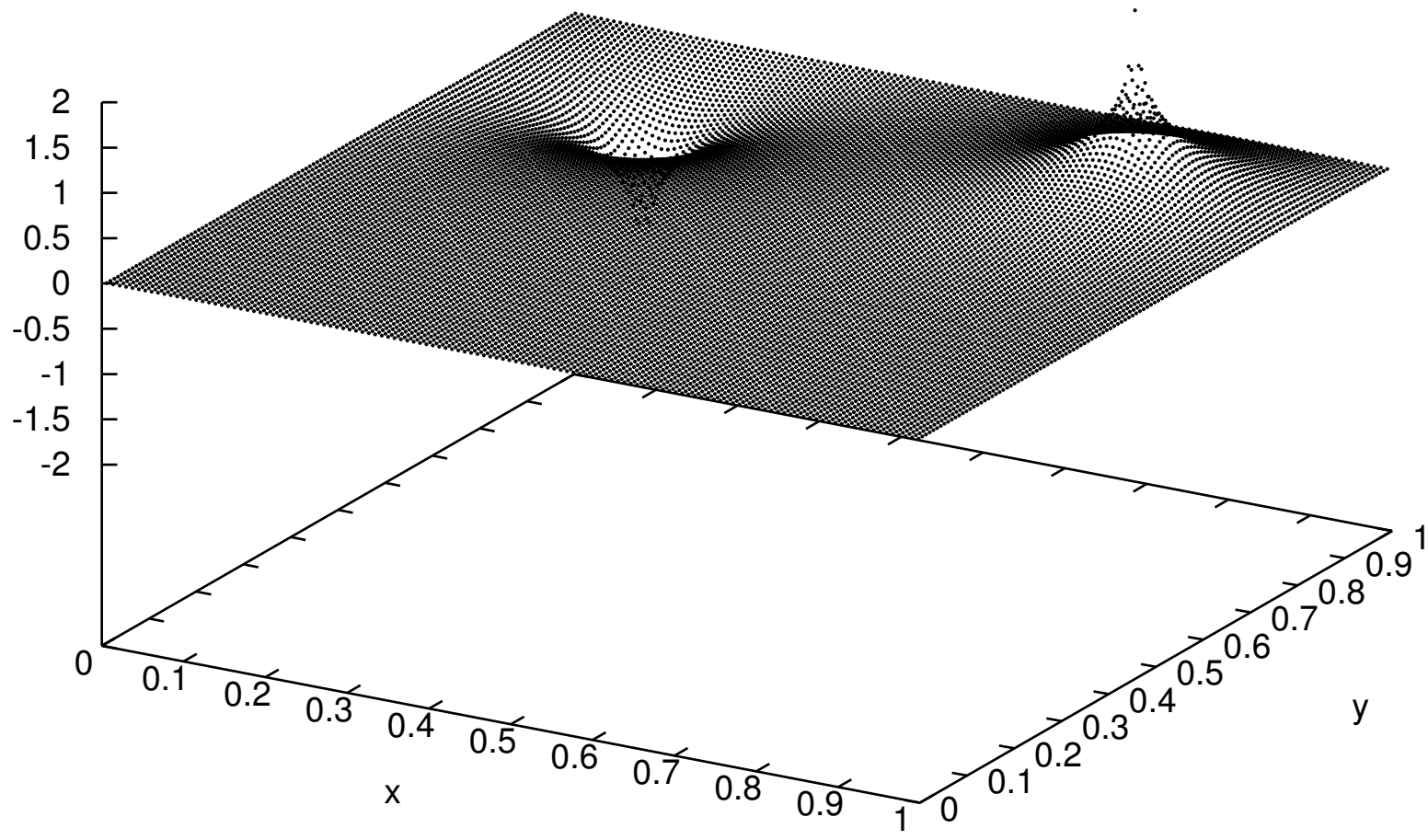
The number of solution points determines how accurate the solution can be.



Uniform distribution of 4096 points, 718 iterations, total 2,940,928 steps

A Simple Adaptive Computation

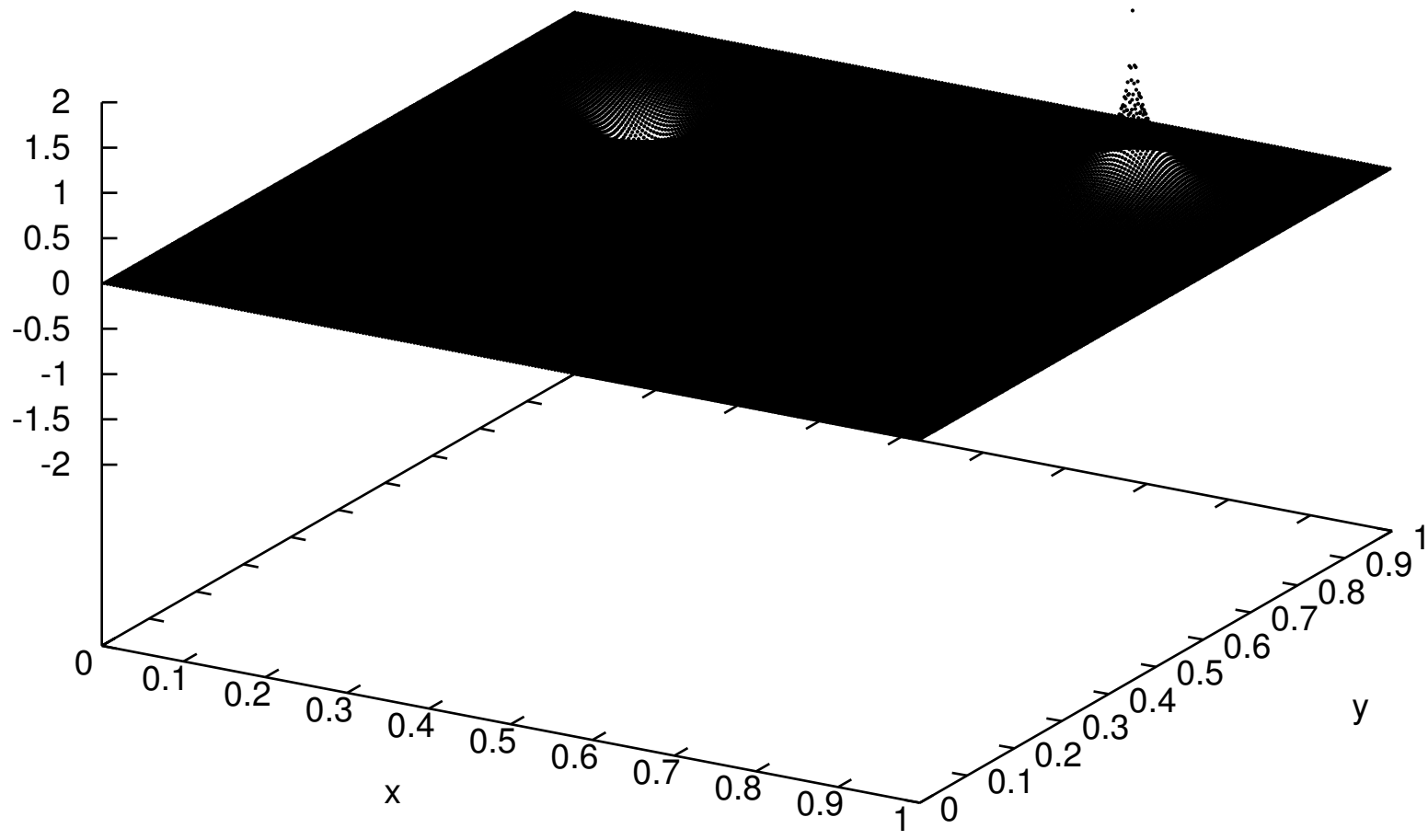
The number of solution points determines how accurate the solution can be.



Uniform distribution of 16,384 points, 1302 iterations, total 21,331,968 steps

A Simple Adaptive Computation

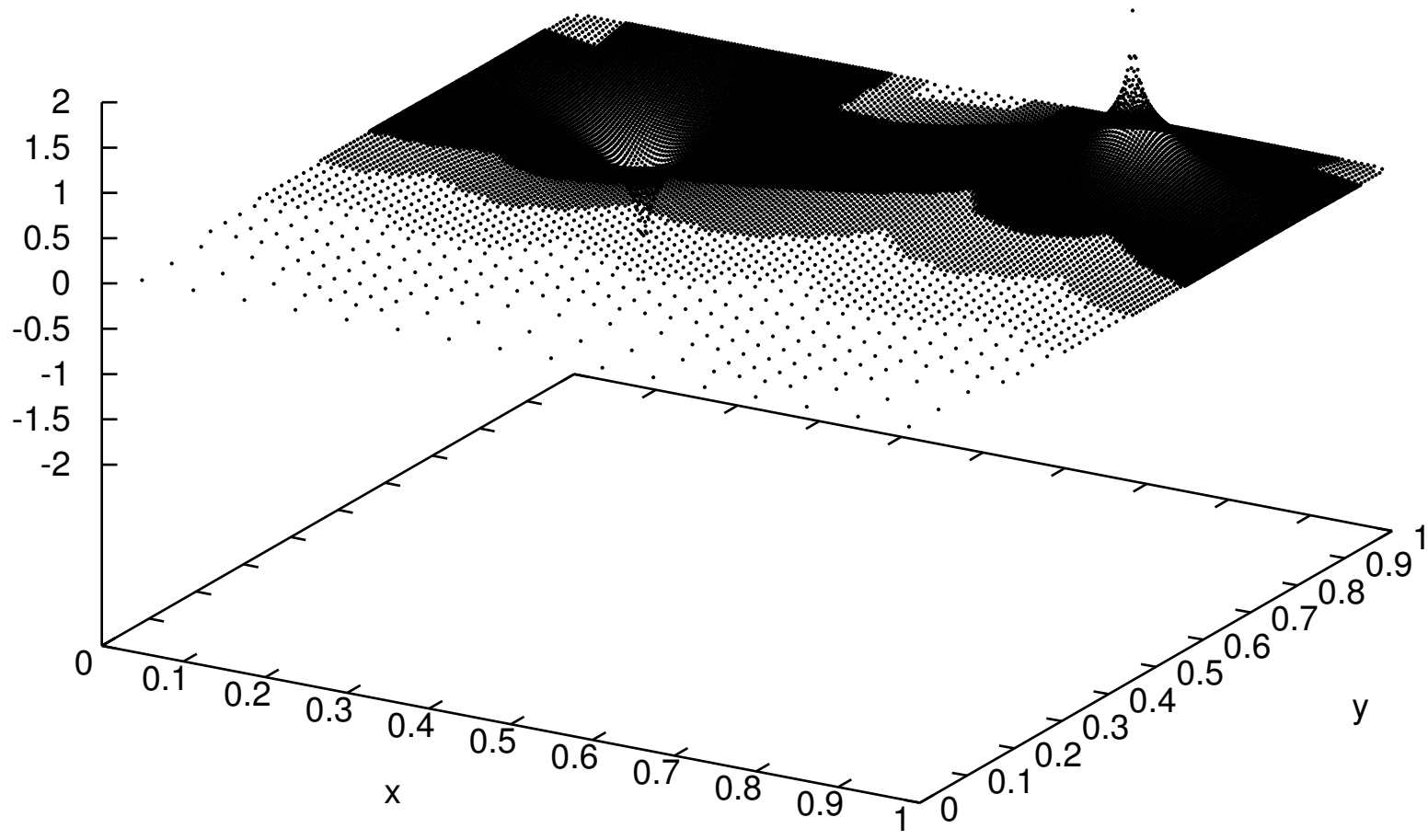
The number of solution points determines how accurate the solution can be.



Uniform distribution of 65,536 points, 1508 iterations, total 98,828,288 steps

A Simple Adaptive Computation

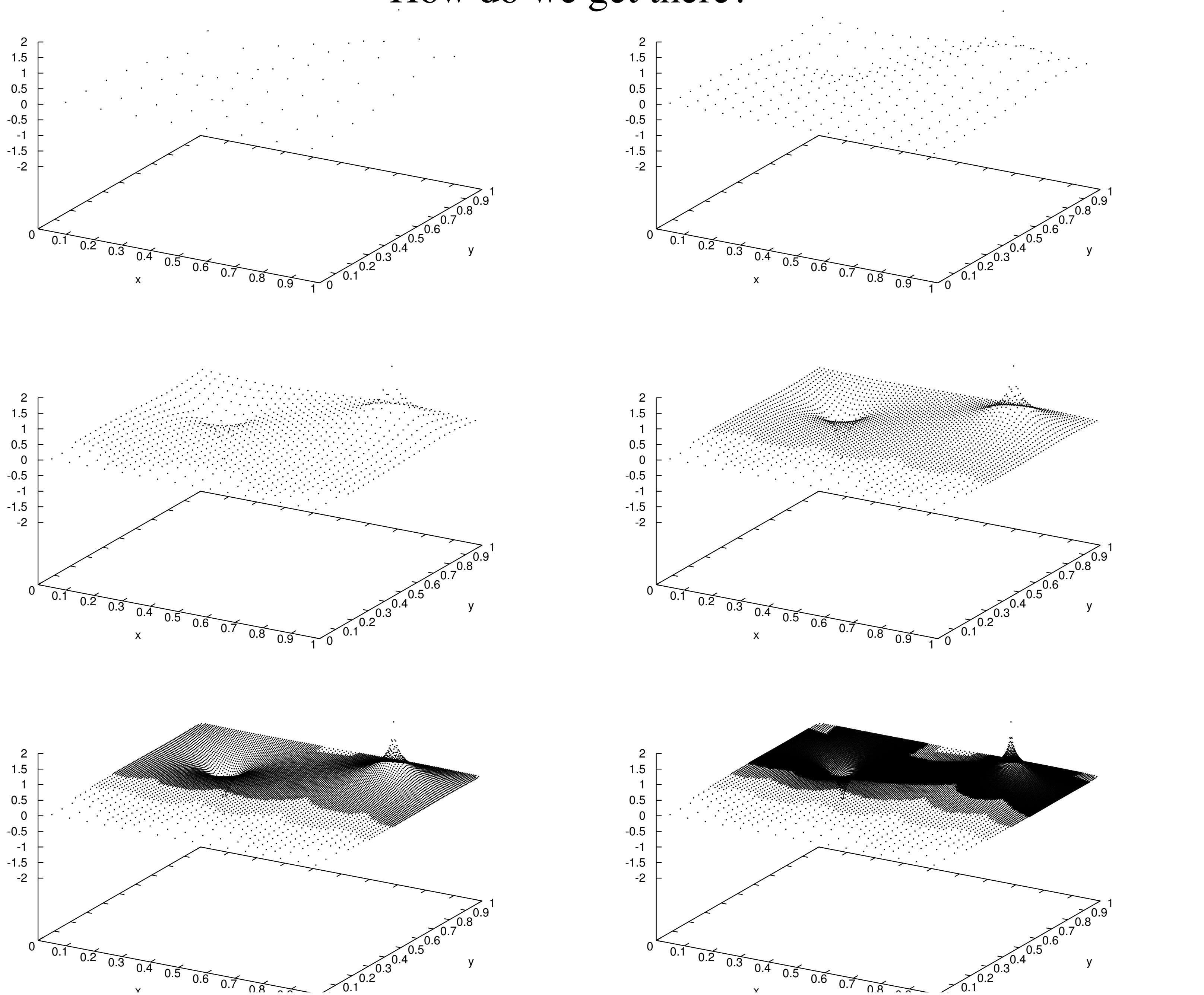
But there's no need for all those points away from the heat source!



Same accuracy as most refined grid, but only 26,812 points, 28,453,576 steps

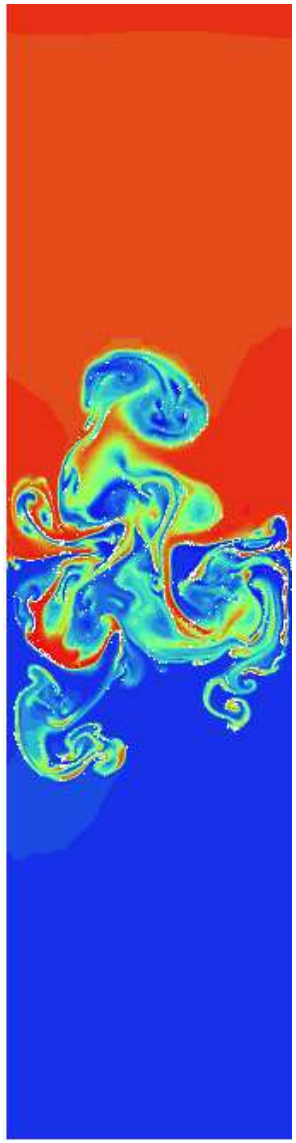
A Simple Adaptive Computation

How do we get there?



Large-scale Rayleigh-Taylor Instability

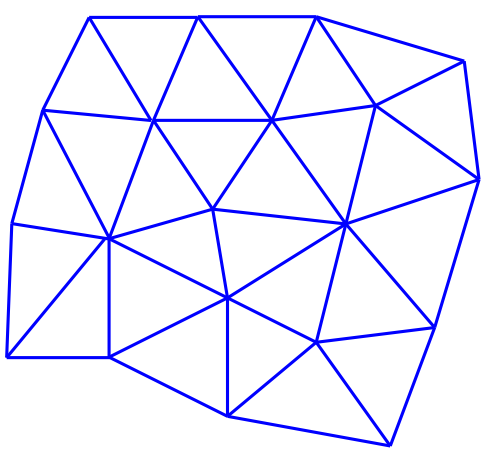
- Department of Energy/ASCI project
  - University of Chicago
  - Argonne National Laboratory
  - Rensselaer Polytechnic Institute
- Goal: model a deflagration leaving the surface of a compact star
  - Long term: simulate thermonuclear flashes in astrophysical bodies (neutron stars, white dwarves)
  - Study Rayleigh-Taylor instabilities
  - Ideal gases: $\rho = 2$ on top, $\rho = 1$ on bottom
  - Sinusoidal velocity perturbation with magnitude 0.05
- Parallel adaptive solution
  - Discontinuous Galerkin solution of the Euler equations
  - Woodward and Colella flux
  - Similar accuracy without adaptivity would require orders of magnitude more work

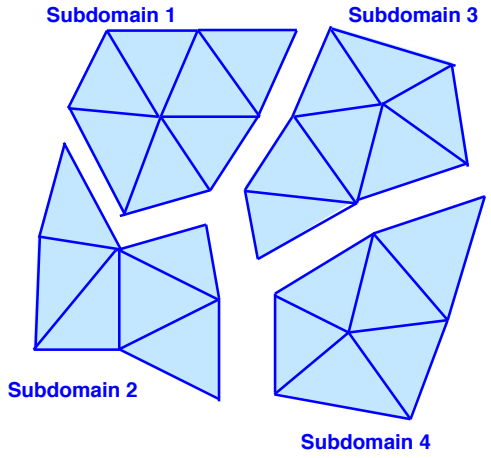


[Animations]

Parallel Strategy

- Dominant paradigm: Single Program Multiple Data (SPMD)
  - distributed memory; communication via message passing
- Can run the same software on shared and distributed memory systems
- Adaptive methods necessitate linked structures
  - automatic parallelization is difficult
- Explicitly distribute the computation via a domain decomposition

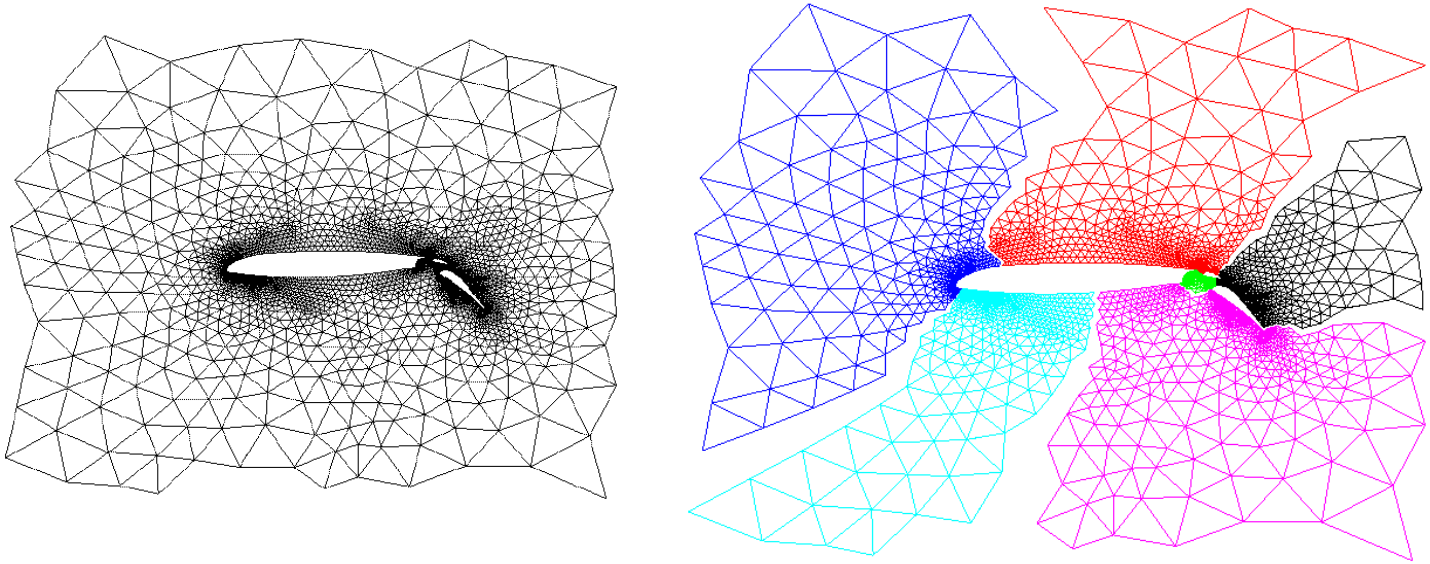




- Distributed structures complicate matters
  - interprocess links, boundary structures, migration support
  - very interesting issues, but not today's focus

Mesh Partitioning

- Determine and achieve the domain decomposition

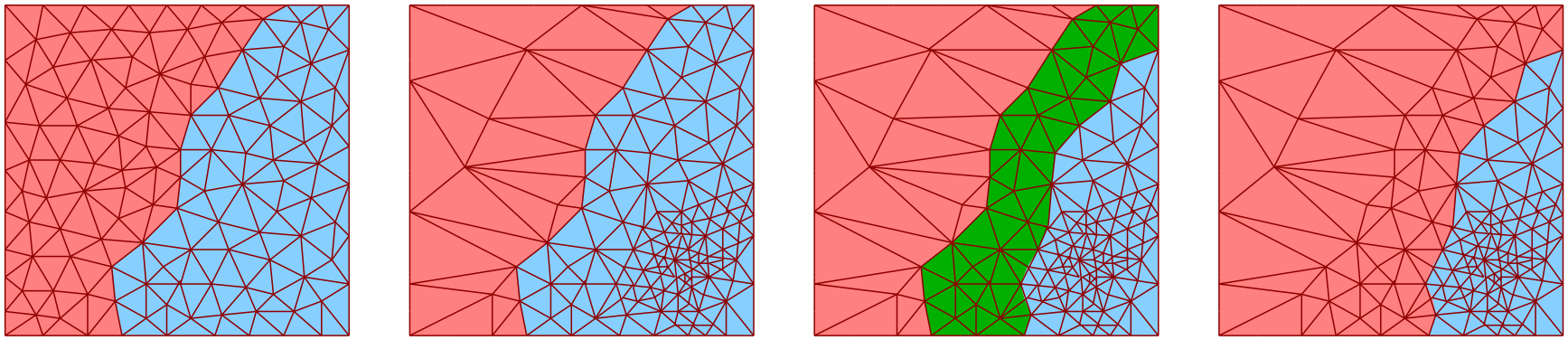


- "Partition quality" is important to solution efficiency
  - evenly distribute mesh elements (computational work)
  - minimize elements on partition boundaries (communication volume)
  - minimize number of "adjacent" processes (number of messages)
- But.. this is essentially graph partitioning: *"Optimal" solution intractable!*

Why dynamic load balancing?

Need a rebalancing capability in the presence of:

- Unpredictable computational costs
  - Multiphysics
  - Adaptive methods



Initial balanced partition | Adaptivity introduces imbalance | Migrate as needed | Rebalanced partition

- Non-dedicated computational resources
- Heterogeneous computational resources of unknown relative powers

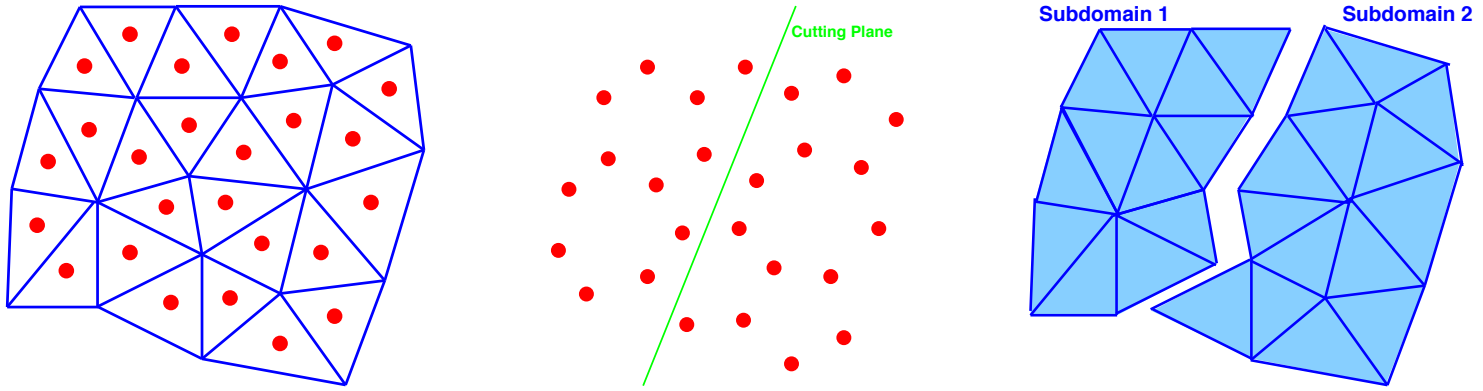
Load Balancing Considerations

- Like a partitioner, a load balancer seeks
  - computational balance
  - minimization of communication and number of messages
- But also must consider
  - cost of computing the new partition
    - may tolerate imbalance to avoid a repartition step
  - cost of moving the data to realize it
    - may prefer incrementality over resulting quality
- Must be able to operate in parallel on distributed input
  - scalability
- It is *not* just graph partitioning – no single algorithm is best for all situations
- Several approaches have been used successfully

Geometric Mesh Partitioning/Load Balancing

Use only coordinate information

- Most commonly use "cutting planes" to divide the mesh



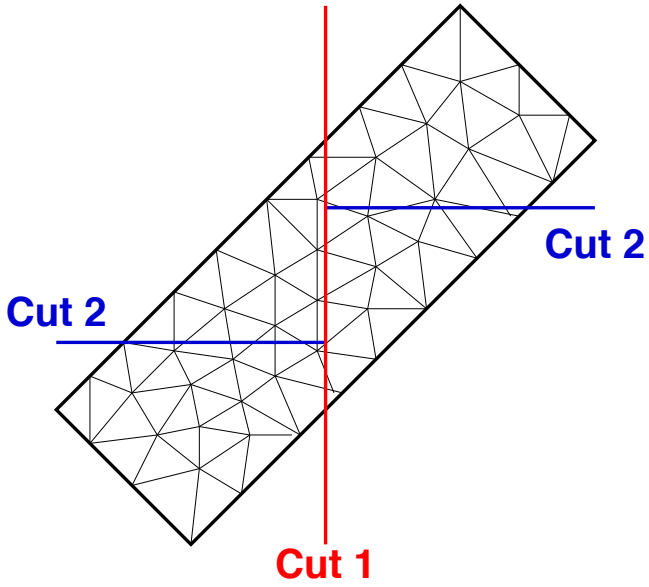
- Tend to be fast, and can achieve strict load balance
- "Unfortunate" cuts may lead to larger partition boundaries
  - cut through a highly refined region
- May be the only option when only coordinates are available
- May be especially beneficial when spatial searches are needed
  - contact problems in crash simulations

Recursive Bisection Mesh Partitioning/Load Balancing

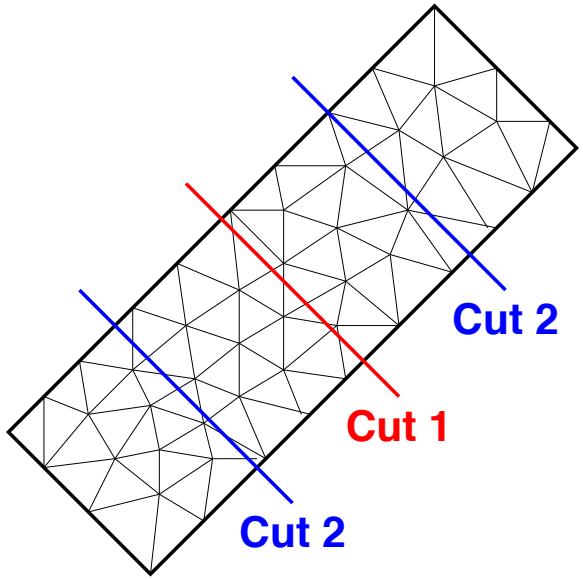
Simple geometric methods

- Recursive methods, recursive cuts determined by

Coordinate Bisection (RCB)



Inertial Bisection (RIB)

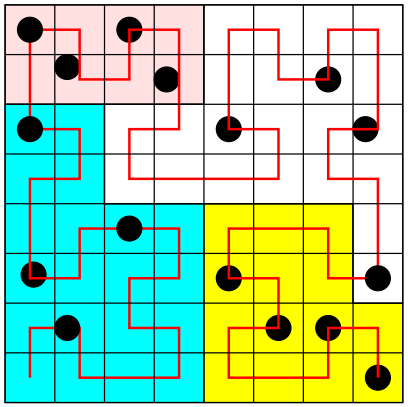


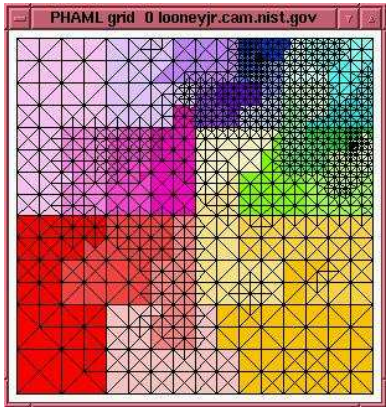
- Simple and fast
- RCB is incremental
- Partition quality may be poor
- Boundary size may be reduced by a post-processing “smoothing” step

SFC Mesh Partitioning/Load Balancing

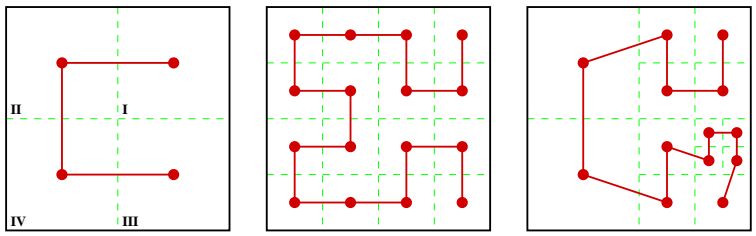
Another geometric method

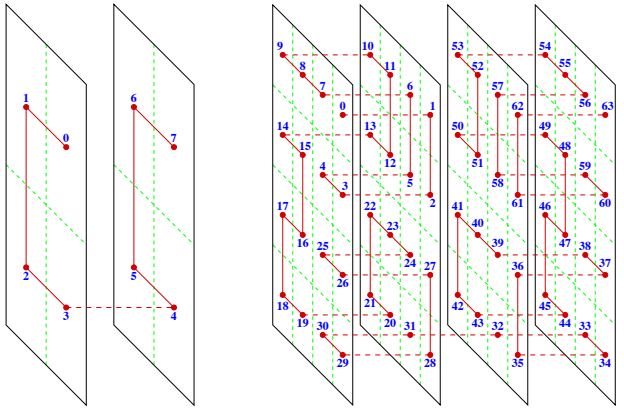
- Use the locality-preserving properties of space-filling curves (SFCs)
- Each element is assigned a coordinate along an SFC
  - a linearization of the objects in two- or three-dimensional space





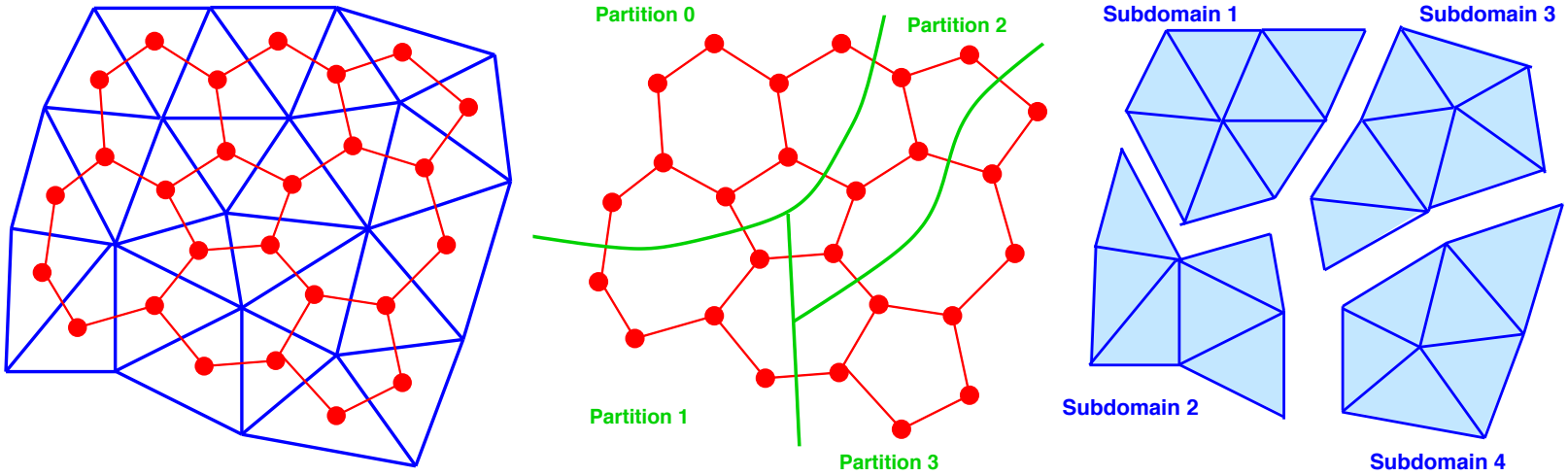
- Hilbert SFC is most effective





Graph-Based Mesh Partitioning/Load Balancing

Use connectivity information



- Spectral methods (Chaco)
  - prohibitively expensive and difficult to parallelize
  - produces excellent partitions
- Multilevel partitioning (Parmetis, Jostle)
  - much faster than spectral, but still more expensive than geometric
  - quality of partitions approaches that of spectral methods
- May introduce some load imbalance to improve boundary sizes

Load Balancing Algorithm Implementations

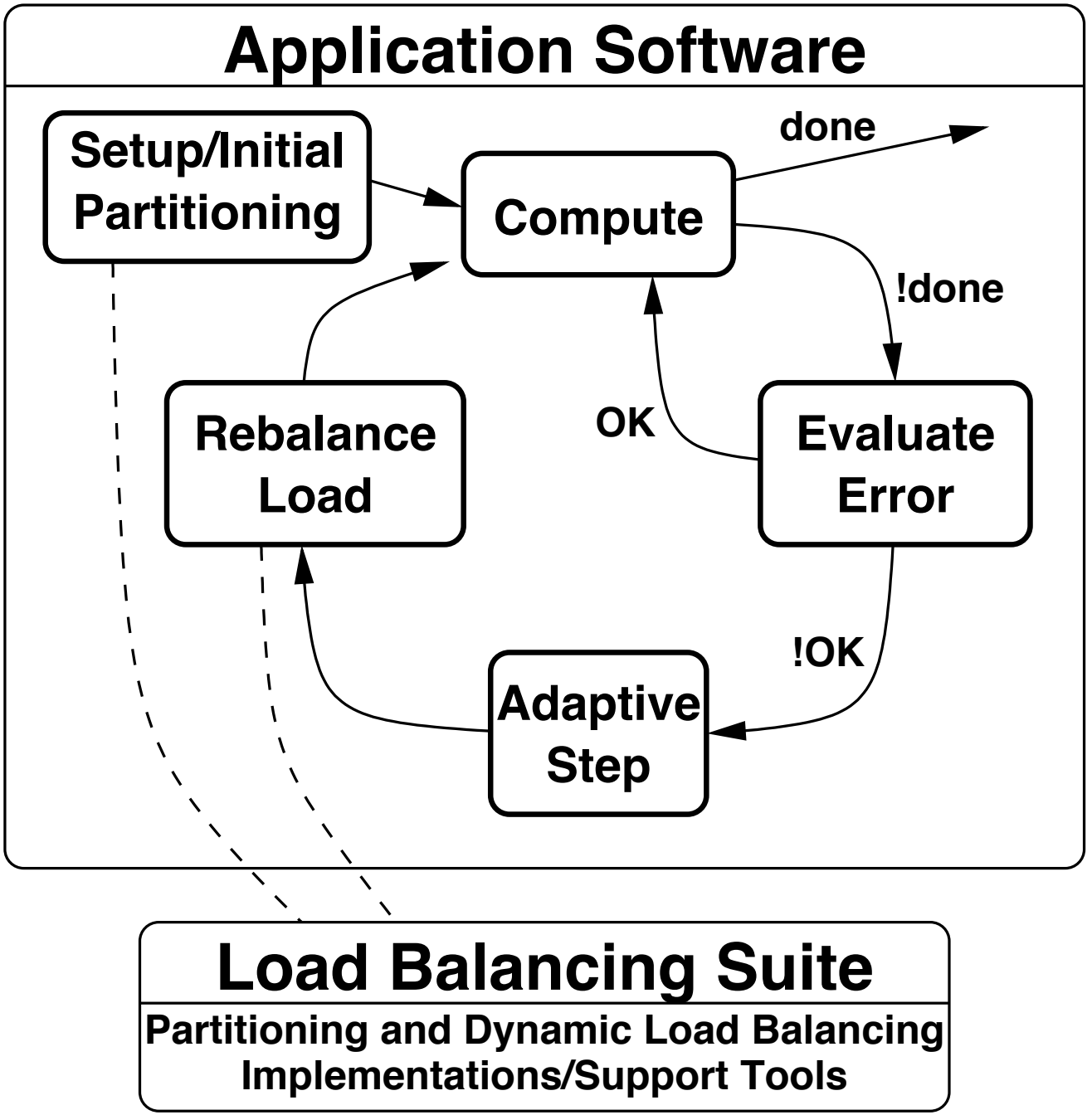
- Again, no single algorithm is best in all situations
- Some are difficult to implement
- Bad: implementation within an application or framework
  - likely usable only by a single application
  - at best, usable by a few applications that share common data structures
  - unlikely that an expert in load balancing is the developer
- Better: implementation within reusable libraries
  - load balancing experts can develop and optimize implementations
  - application programmers can make use without worrying about details
  - but...how to deal with the variety of applications and data structures?
    - require specific input and output structures
      - applications must construct them
    - data-structure neutral design
      - applications only need to provide a small set of callback functions

Zoltan Toolkit

Includes suite of partitioning algorithms, developed at

- General interface to a variety of partitioners and load balancers
- Application programmer can avoid the details of load balancing
- Interact with application through callback functions and migration arrays
  - "data structure neutral" design
- Switch among load balancers easily; experiment to find what works best
- Provides high quality implementations of:
  - Coordinate bisection, Inertial bisection
  - Octree/SFC partitioning (with Loy, Gervasio, Campbell – RPI)
  - Hilbert SFC partitioning
  - Refinement tree balancing (Mitchell – NIST)
- Provides easier-to-use interfaces for:
  - Metis/Parmetis (Karypis, Kumar, Schloegel – Minnesota)
  - Jostle (Walshaw – Greenwich)
- Freely available: `http://www.cs.sandia.gov/Zoltan/`

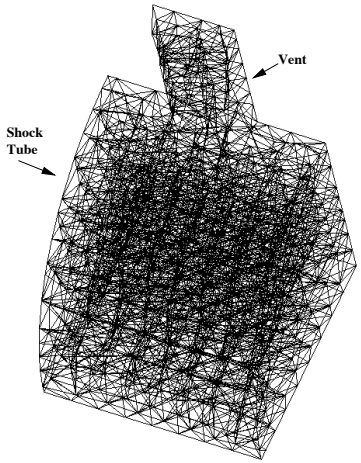
Typical Computation Flow

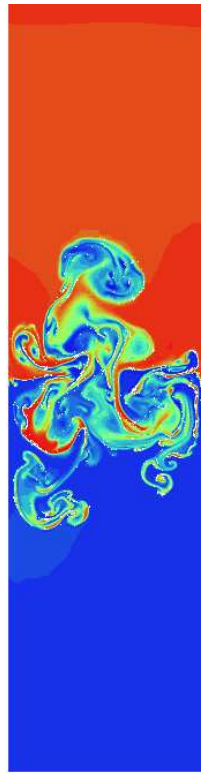


Example Parallel Adaptive Software

We wish to run several applications.

- Rensselaer's "LOCO"
  - parallel adaptive discontinuous Galerkin solution of compressible Euler equations in C.
  - using Parallel Mesh Database
  - "perforated shock tube" problem
- Rensselaer's "DG"
  - also discontinuous Galerkin methods, but in C++
  - using Algorithm-Oriented Mesh Database
  - Rayleigh-Taylor flow instabilities and others
- Mitchell's PHAML
  - Fortran 90, adaptive solutions of various PDEs
- Simmetrix, Inc. MeshSim-based applications
- Real interest for parallel computing is in 3D transient problems





Target Computational Environments

- FreeBSD Lab, Williams CS: 12 dual 2.4 GHz Intel Xeon processor systems
- *Bullpen Cluster*, Williams CS: 13 node Sun cluster, total of 4 300 MHz and 21 450 MHz UltraSparc II processors
- *Dhanni Cluster*, Williams CS: 8 nodes, each with 2 dual-core 2.8 GHz Intel Xeon processors
- *Medusa Cluster*, RPI: 32 dual 2.0GHz Intel Xeon processor systems
- ASCI-class supercomputers: large clusters of SMPs
- *System X*, Virginia Tech: 1100 dual 2.3 GHz Apple G5 nodes, 12.25 TeraOps
- TeraGrid – NSF's Internet "Grid Computer": nodes at NCSA, San Diego Supercomputing Center, Argonne National Laboratory, Caltech, and the Pittsburgh Supercomputer Center, peak projected performance 40 TeraOps

This is just a small sample of the wide variety of systems in use.

- long-term "resource-aware computation" goal: software that can run efficiently on any of them
- work described here is one step toward this goal, current focus on systems found at places like Williams

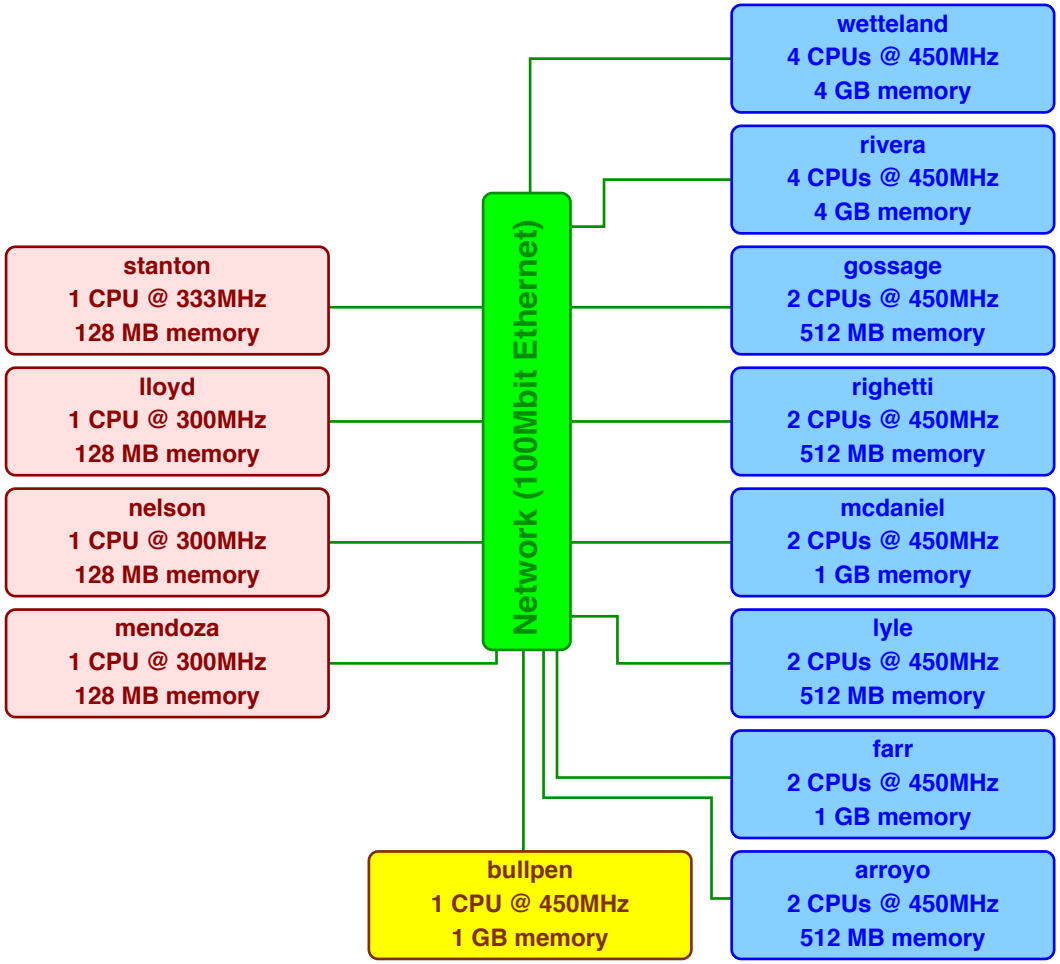
Resource-Aware Computing Motivations

- Heterogeneous processor speeds
  - seem straightforward to deal with
  - does it matter?
- Distributed *vs*. shared memory
  - some algorithms may be a more appropriate choice than others
- Non-dedicated computational resources
  - can be highly dynamic, transient
  - will the situation change by the time we can react?
- Heterogeneous or non-dedicated networks
- Hierarchical network structures
  - message cost depends on the path it must take
- Relative speeds of processors/memory/networks
  - important even when targeting different homogeneous clusters

What Can Be Adjusted?

- Choice of solution methods and algorithms
  - different approaches for multithreading *vs*. distributed memory
- Parallelization paradigm
  - threads *vs*. message passing *vs*. actor/theater model *vs*. hybrid approaches
  - "bag-of-tasks" master/slave *vs*. domain decomposition
- Ordering of computation and/or communication
- Replication of data or computation
- Communication patterns (*e.g.*, message packing)
- Optimal number of processors, processes, or threads
  - not necessarily one process/thread per processor
- Our focus: partitioning and dynamic load balancing
  - tradeoffs for imbalance *vs*. communication volume
  - variable-sized partitions
  - avoid communication across slowest interfaces

Bullpen Cluster at Williams College



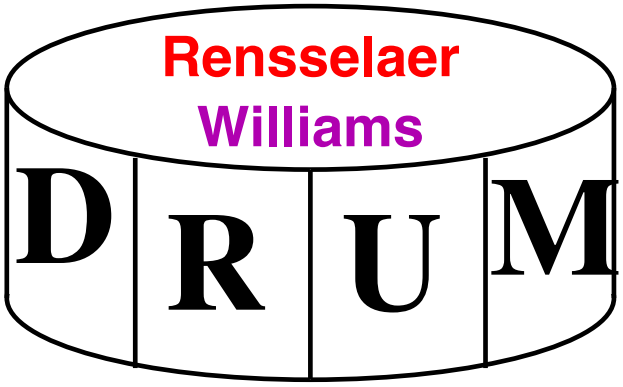
All nodes contain (aging) Sun UltraSparc II processors

`http://bullpen.cs.williams.edu/`

Resource-Aware Load Balancing

- Goal: account for environment characteristics in load balancing
- Idea: build a model of the computing environment and use it to guide load balancing
  - represent heterogeneity and hierarchy
    - processor heterogeneity, SMP
    - network capabilities, load, hierarchy
  - static capability and dynamic monitoring feedback
- Use existing load balancing procedures to produce, as appropriate
  - variable size partitions
  - "hierarchical" partitions
- Longer-term: tailor other parts of the computation to the environment
- Alternate approach: process-level or system-level load balancing

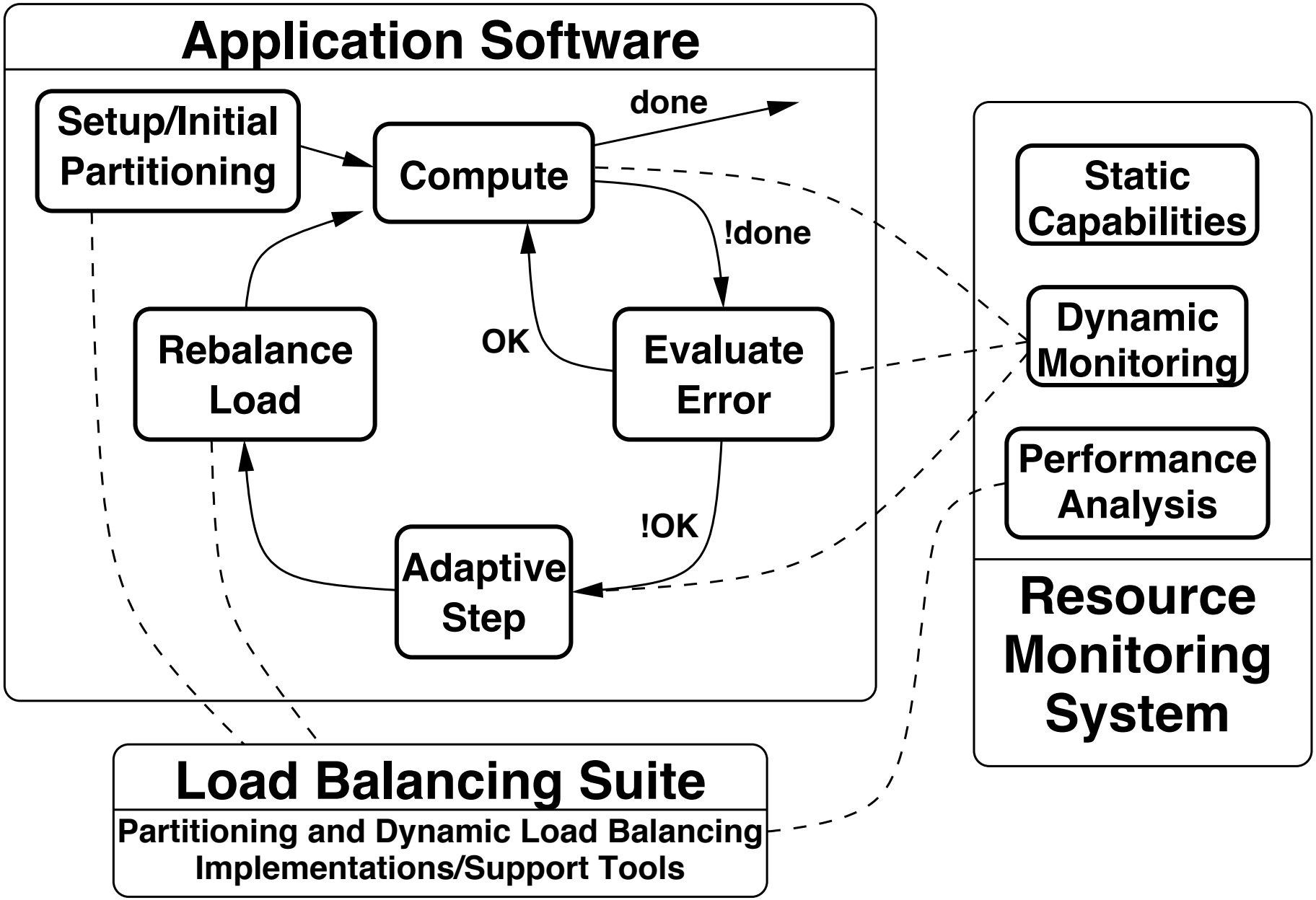
DRUM: Dynamic Resource Utilization Model



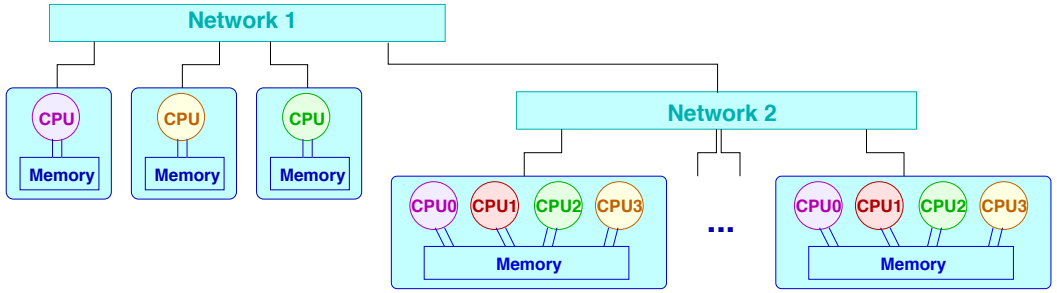
- Run-time model encapsulates the details of the execution environment
- Supports dynamic load balancing for environments with
  - heterogeneous processing capabilities
  - heterogeneous network speeds
  - hierarchical network topology
  - non-dedicated resources
- Not dependent on any specific application, data structure, or partitioner

`http://www.cs.williams.edu/drum/`

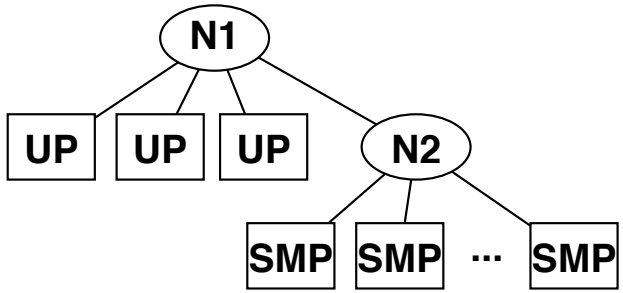
Computation Flow with DRUM Monitoring



DRUM: Dynamic Resource Utilization Model



Computing Environment



Machine Model

- Tree structure based on network hierarchy
- Computation nodes, assigned "processing power"
  - UP – uniprocessor node
  - SMP – symmetric multiprocessing node
- Communication nodes
  - network characteristics (bandwidth, latency)
  - assigned a processing power as a function of children

DRUM: Dynamic Resource Utilization Model

- Static capabilities
  - gathered by benchmarks or specified manually *once* per system
  - processor speeds, network capabilities and topology
- Dynamic performance monitoring
  - gathered by "agent" threads managed through a simple API
  - communication interface (NIC) monitors
    - monitor incoming and outgoing packets and/or available bandwidth
  - CPU/Memory monitors
    - monitors CPU and memory usage and availability
- Combine static capability information and dynamic monitoring feedback
- Straightforward to use powers to create weighted partitions with existing procedures
- Optional Zoltan interface allows use by applications with no modifications
- More details and computational results in recent papers:
  Applied Numerical Mathematics, 52(2-3), pp. 133-152, 2005
  Computing in Science & Engineering, 7(2), pp. 40-50, 2005