Software Technology for Problem Solving on Computational Grids

An Overview of the GrADS Project

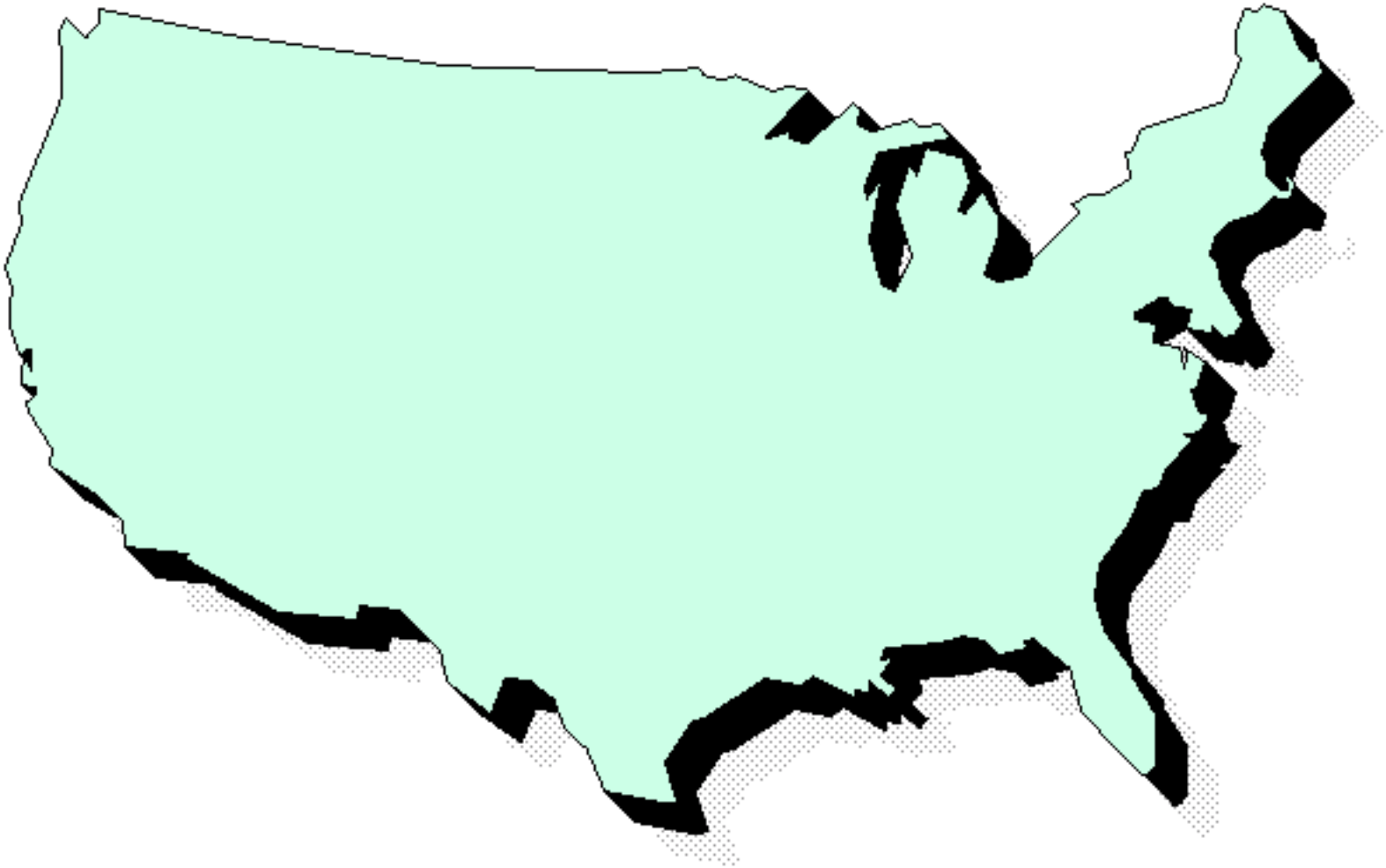
Ken Kennedy

Center for High Performance Software

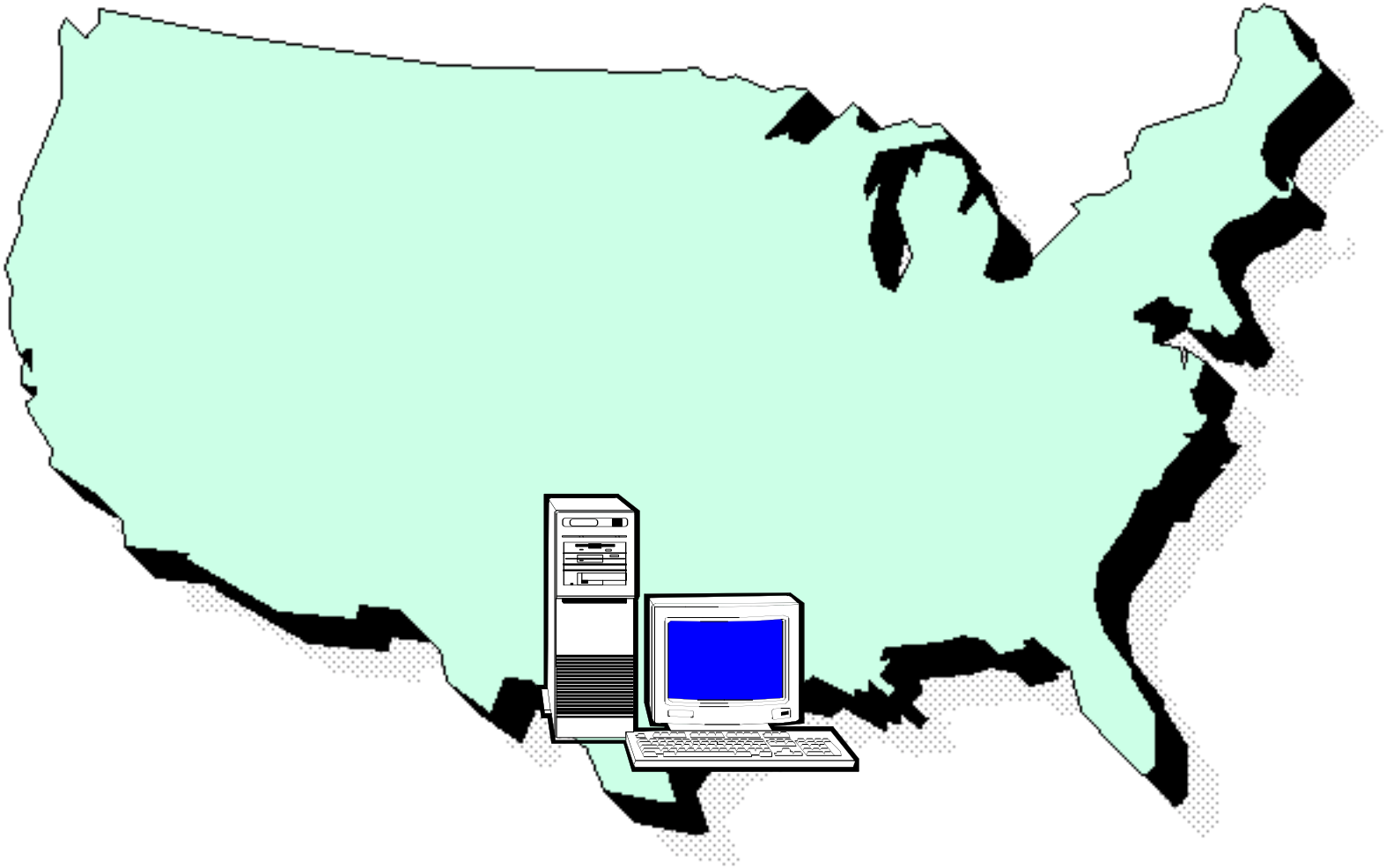
Rice University

http://www.cs.rice.edu/~ken/Presentations/GrADSOverview.pdf

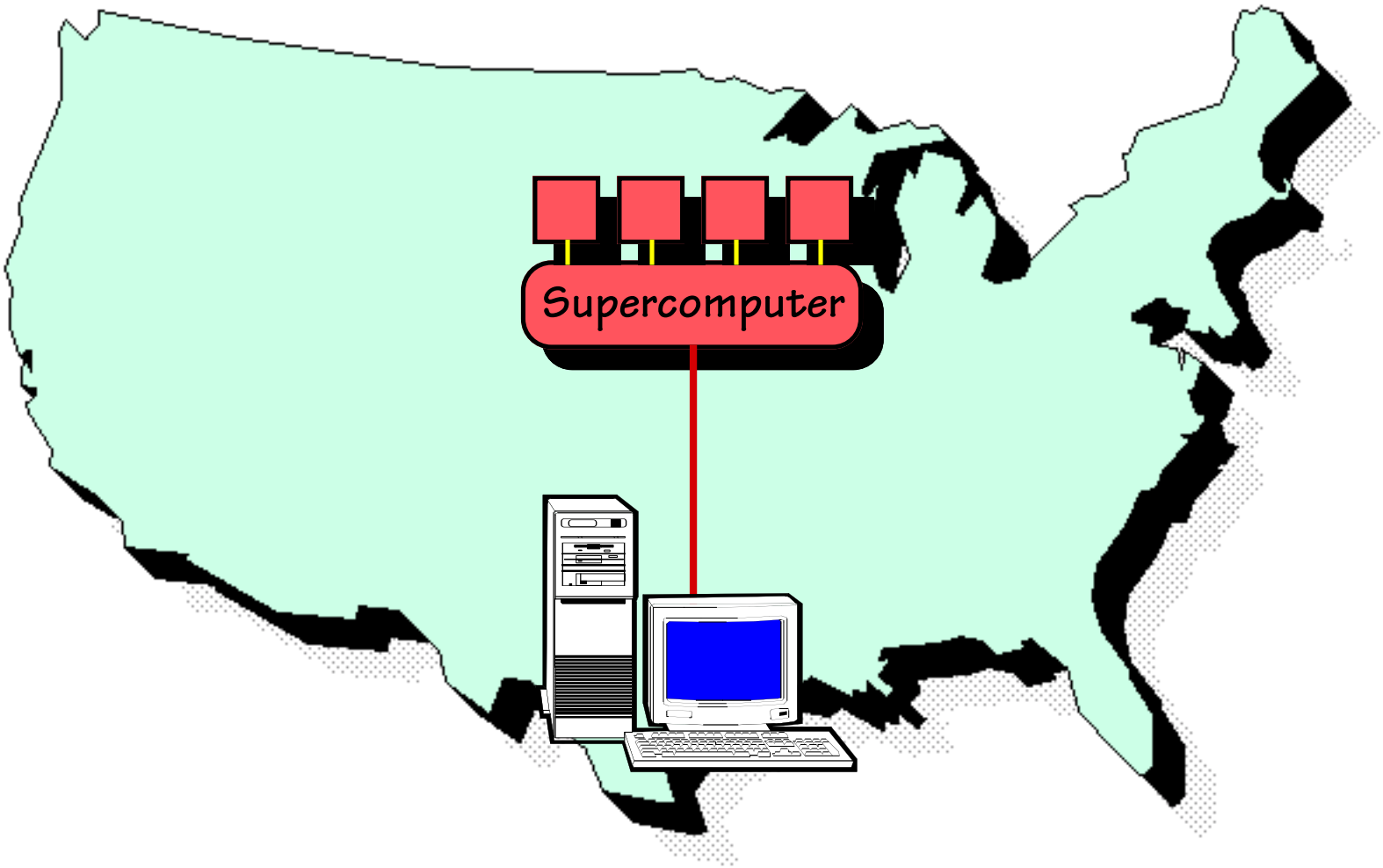
National Distributed Computing



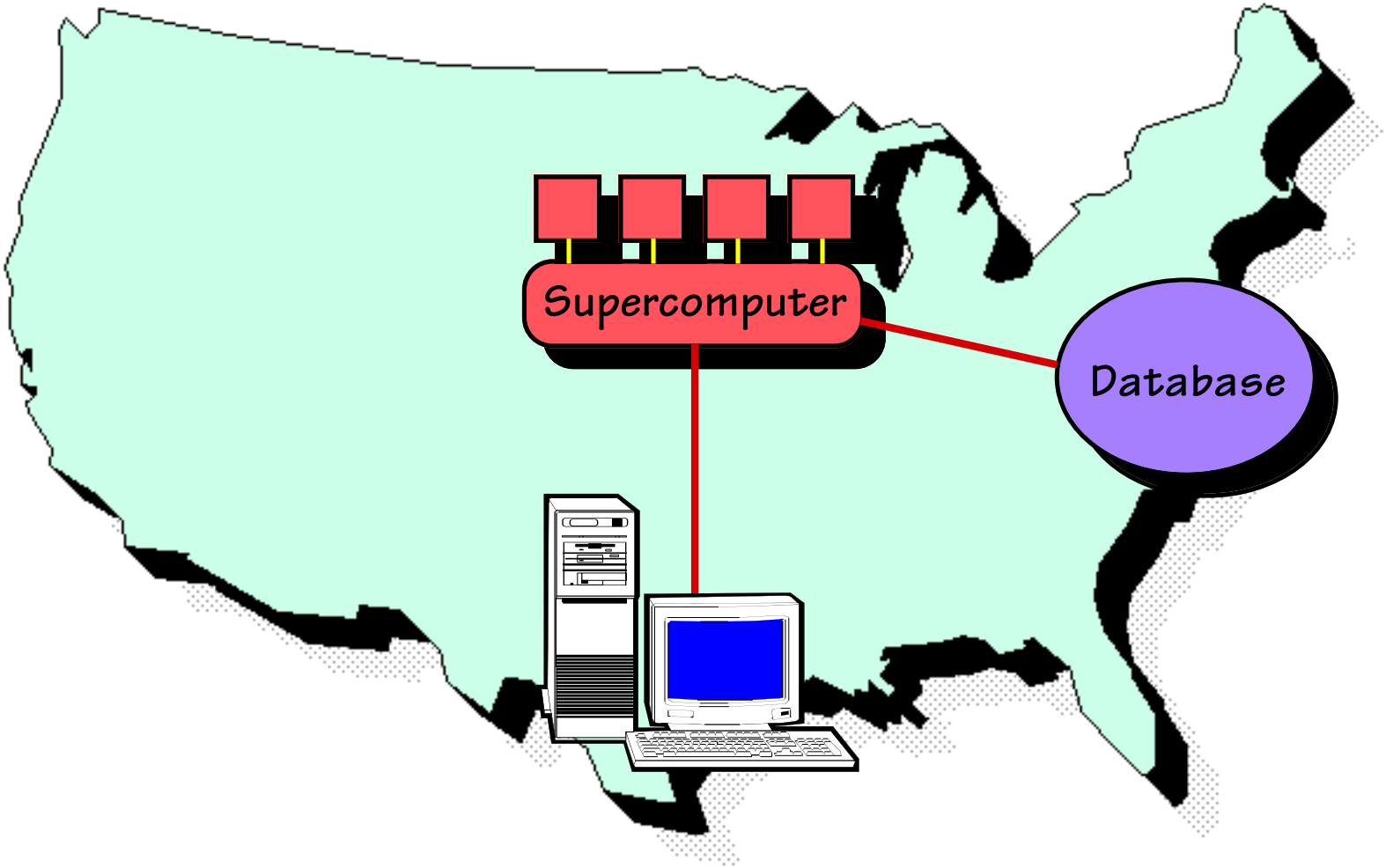
National Distributed Computing



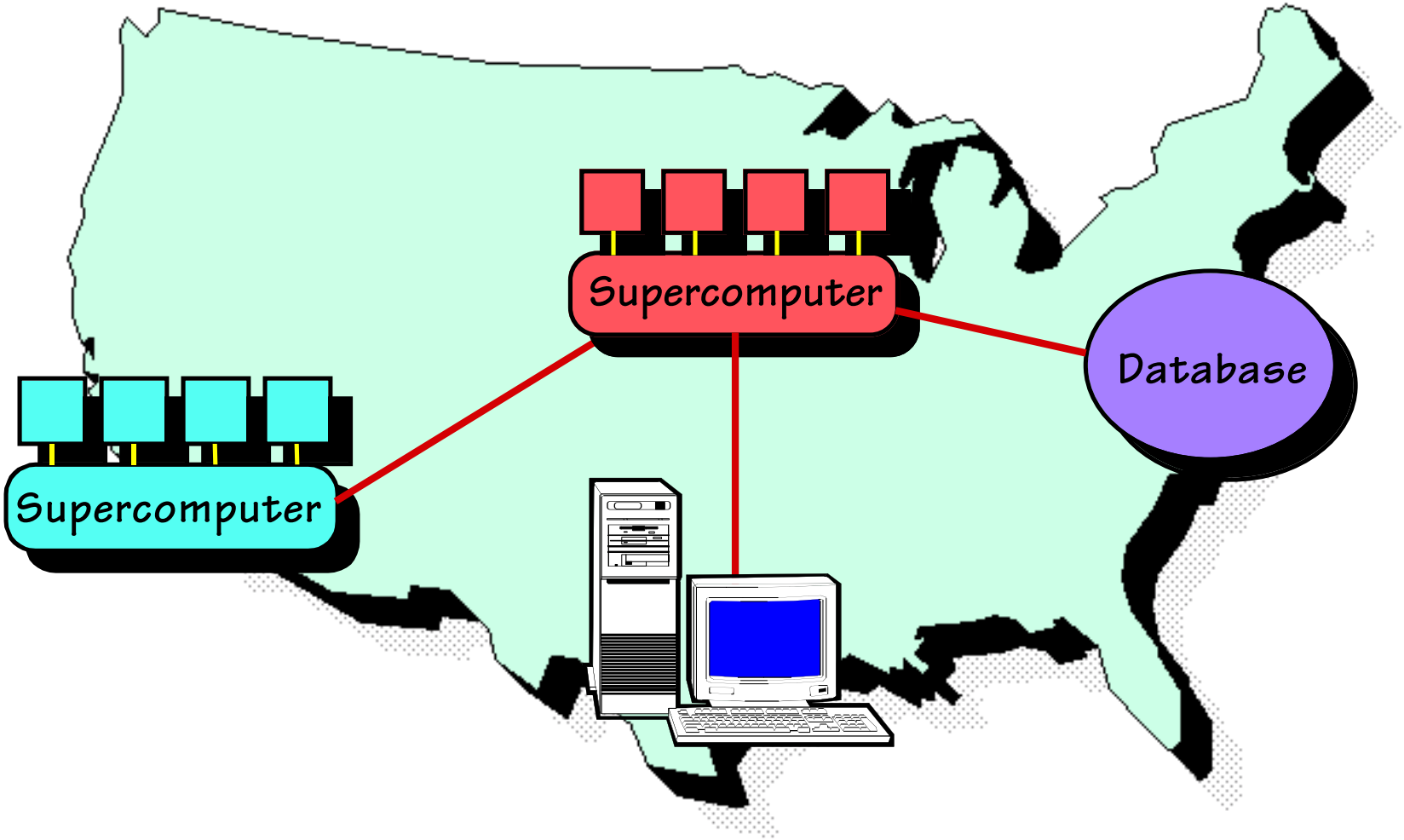
National Distributed Computing



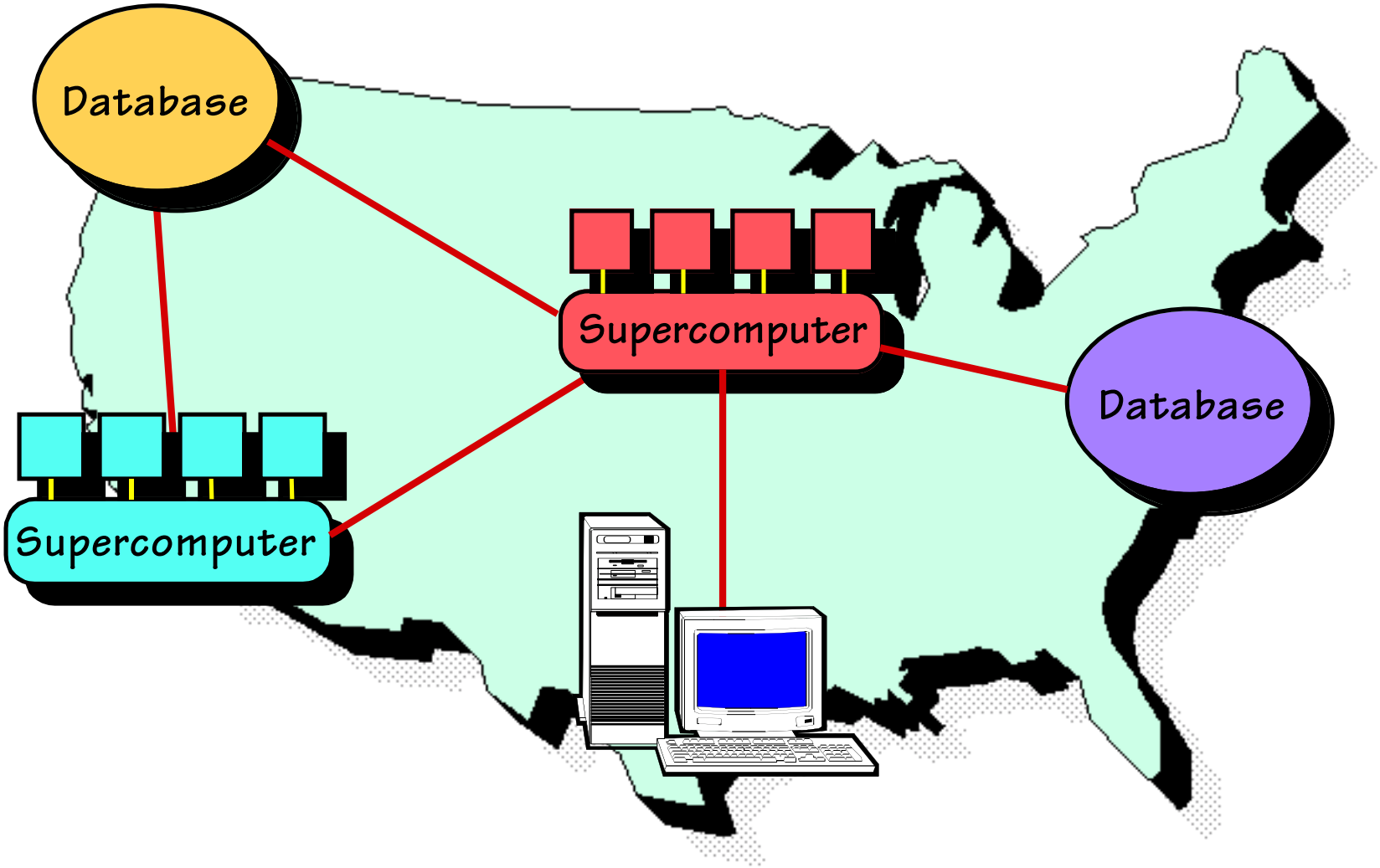
National Distributed Computing



National Distributed Computing



National Distributed Computing



What Is a Grid?

- Collection of computing resources
  - Varying in power or architecture
  - Potentially dynamically varying in load
    - Unreliable?
  - No hardware shared memory
- Interconnected by network
  - Links may vary in bandwidth
  - Load may vary dynamically
- Distribution
  - Across room, campus, state, nation, globe
- Inclusiveness
  - Distributed-memory parallel computer is a degenerate case

A Software Grand Challenge

- Application Development and Performance Management for Grids
  - Goal: Reliable performance on heterogeneous platforms, under varying load

    on computation nodes and on communications links
- Challenges:
  - Presenting a high-level application development interface
    - If programming is hard, its useless
  - Designing qnd constructing applications for adaptability
  - Mapping applications to dynamically changing configurations
  - Determining when to interrupt execution and remap
    - Application monitors
    - Performance estimators

Globus

- Developed by Ian Foster and Carl Kesselman
  - Originally to support the I-Way (SC-96)
- Basic Services for distributed computing
  - Accounting
  - Resource directory
  - User authentication
  - Job initiation
  - Communication services (Nexus and MPI)
- Applications are programmed by hand
  - User responsible for resource mapping and all communication
  - Many applications, most developed with Globus team
    - Globus developers acknowledge how hard this is

What is Needed

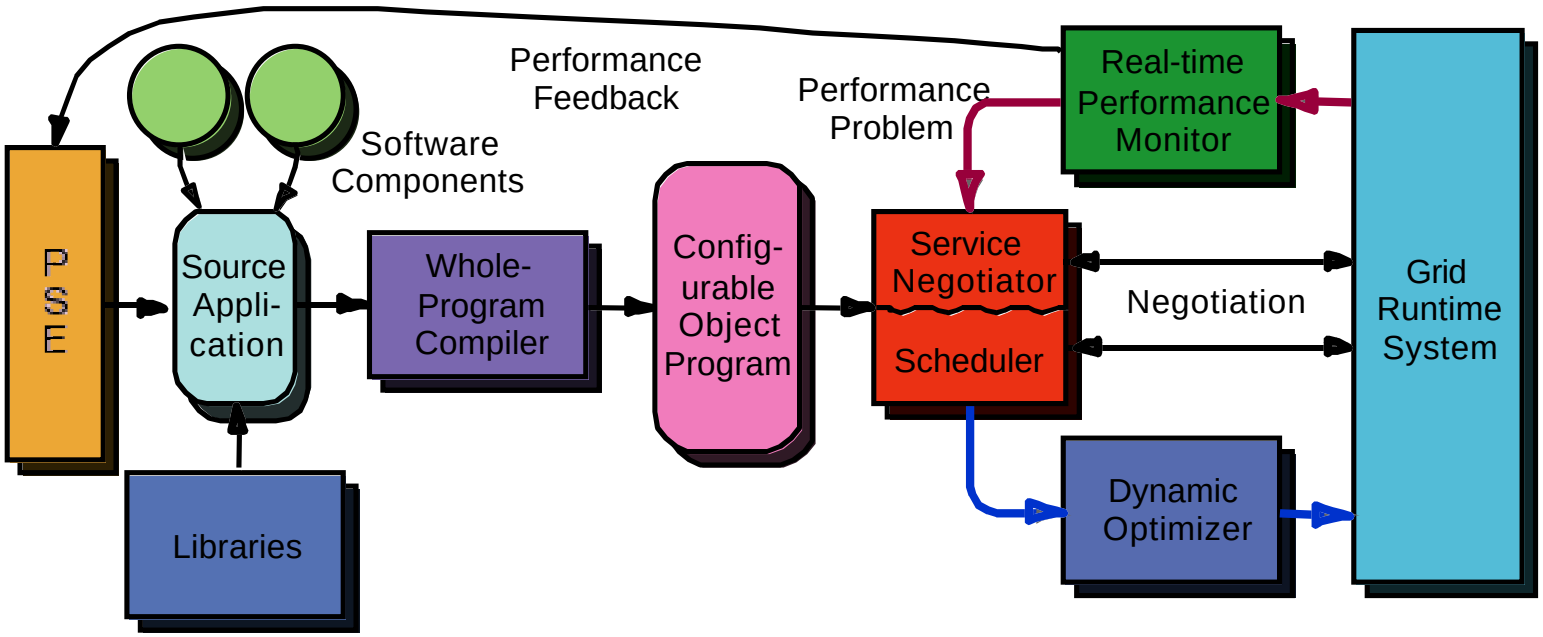
- Execution infrastructure for reliable performance
  - Automatic resource location and execution initiation
  - dynamic configuration to available resources
- Performance monitoring and control strategies
  - deep integration across compilers, tools, and runtime systems
  - performance contracts and dynamic reconfiguration
- Abstract Grid programming models and easy-to-use programming interfaces
  - design of an implementation strategy for those models
  - problem-solving environments
- Robust reliable numerical and data-structure libraries
  - predictability and robustness of accuracy and performance
  - reproducibility, fault tolerance, and auditability

Programming Models

- Distributed Collection of Objects (for serious experts)
  - message passing for communiction
- Distribution of Shared-Memory Programs (for experts)
  - language-based decomposition specification from programmer
  - parametrizable for reconfiguration
    - example: reconfigurable distributed arrays (DAGH)
  - implemented as distributed object collection
  - implicit or explicit communications
- Problem-Solving Environment (for non-experts)
  - packaged components
  - graphical or scripting language for glue

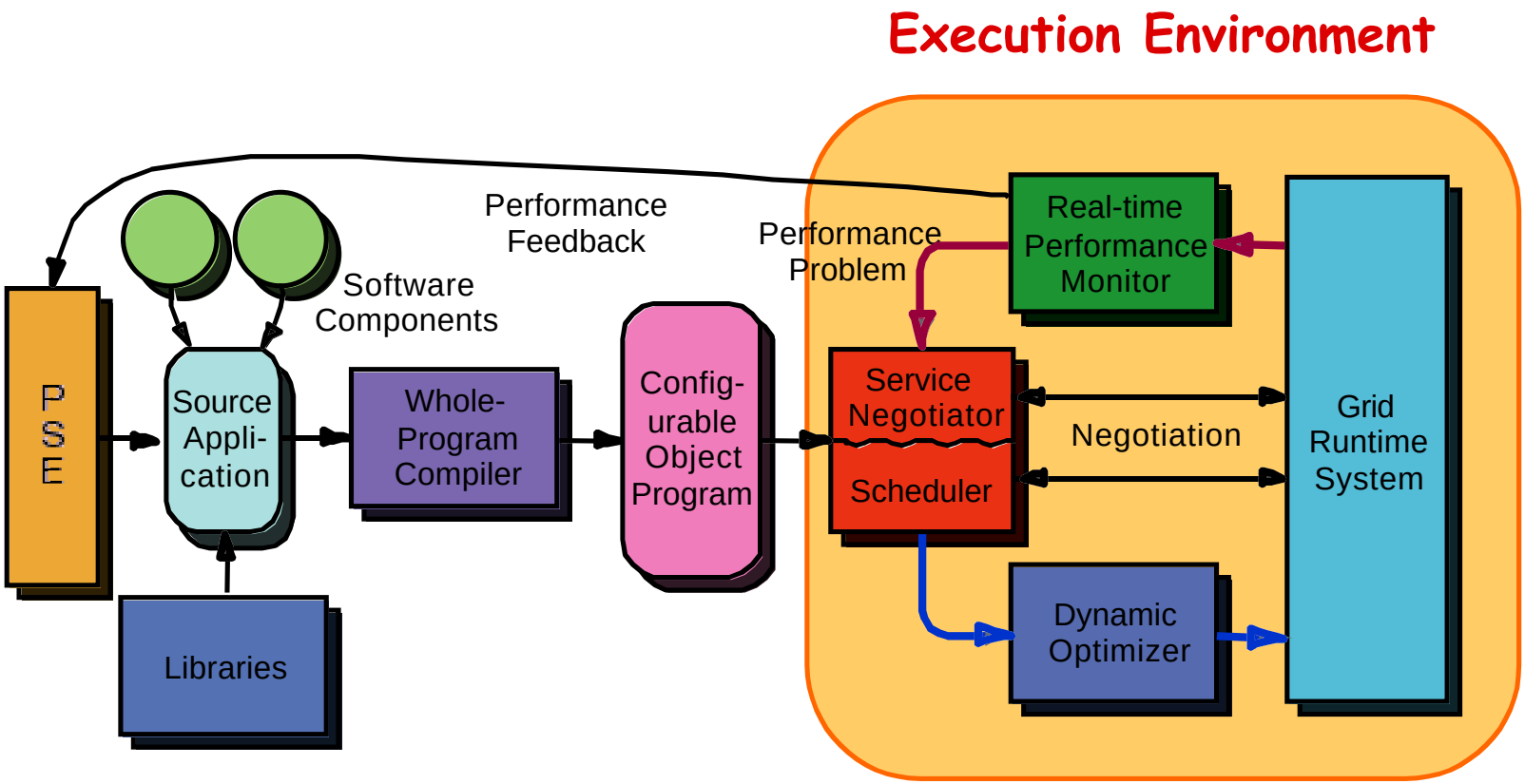
GrADSoft Architecture

- Goal: reliable performance under varying load



GrADS Project (NSF NGS): Berman, Chien, Cooper, Dongarra, Foster, Gannon
Johnsson, Kennedy, Kesselman, Mellor-Crummey, Reed, Torczon, Wolski

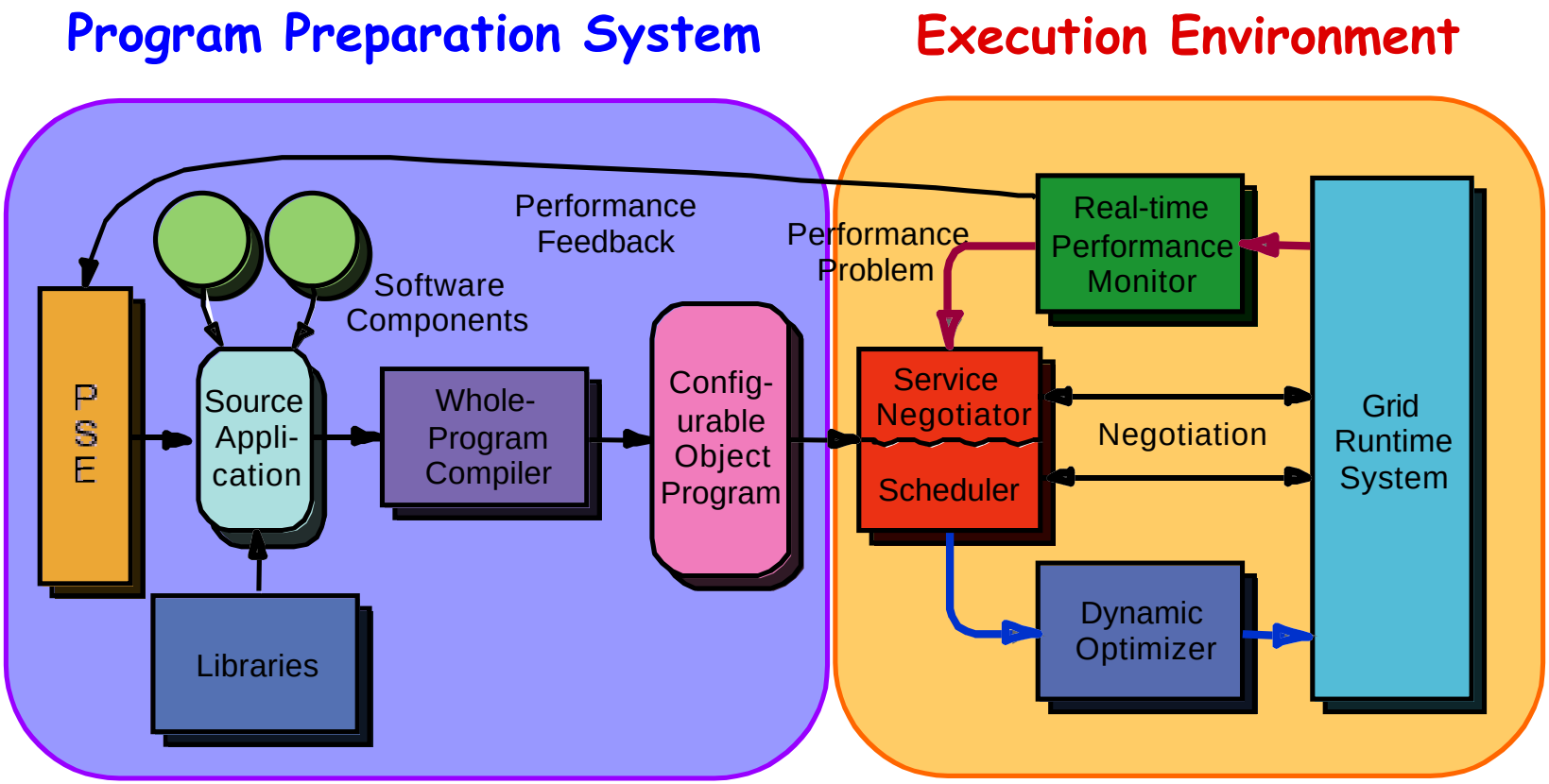
GrADSoft Architecture



Performance Contracts

- At the Heart of the GrADS Model
  - Fundamental mechanism for managing mapping and execution
- What are they?
  - Mappings from resources to performance
  - Mechanisms for determining when to interrupt and reschedule
- Abstract Definition
  - Random Variable: $\rho(A,I,C,t_0)$ with a probability distribution
    - A = app, I = input, C = configuration, t_0 = time of initiation
    - Important statistics: lower and upper bounds (95% confidence)
- Challenge
  - When should a contract be violated?
    - Strict adherence balanced against cost of reconfiguration

GrADSoft Architecture



Configurable Object Program

- Representation of the Application
  - Supporting dynamic reconfiguration and optimization for distributed targets, may include
    - Program intermediate code
    - Annotations from the compiler

      mapping strategy and performance model
    - Historical information (run profile to now)
- Mapping strategies
  - Aggregation of data regions (submeshes) or tasks
  - Definition of parameters for algorithm selection
- Challenge: synthesis of performance models
  - User input, especially associated with libraries
  - Synthesize different components, scale models to problem size

Execution Cycle

- Configurable Object Program is presented
  - Space of feasible resources must be defined
  - Mapping strategy and performance model provided
- Service Negotiator solicits acceptable resource collections
  - Performance model is used to evaluate each
  - Best match is selected and contracted for
- Execution begins
  - Dynamic optimizer tailors program to resources
    - Selects mapping strategy
    - Inserts sensors
- Contract monitoring is conducted during execution
  - Soft violation detection based on fuzzy logic

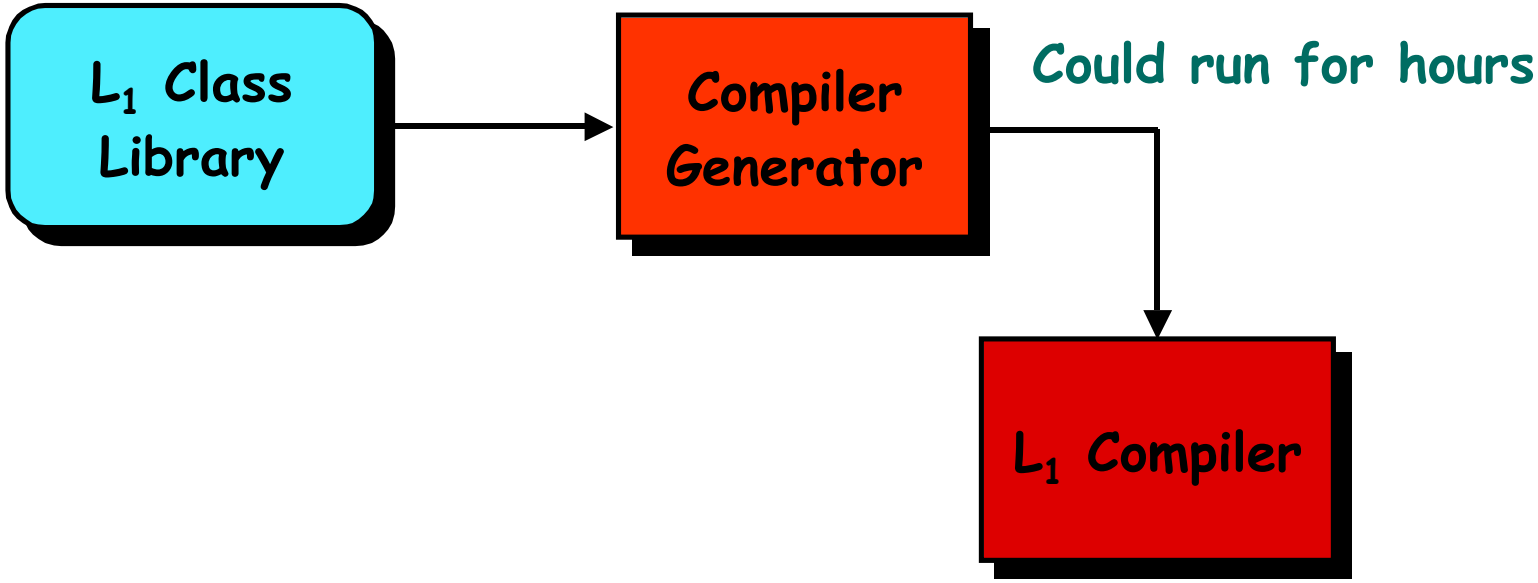
Contract Monitoring

- Input:
  - Performance model
    - Integrated from a variety of sources: user, compiler, application experience, application signatures
  - Resources contracted for
- Trigger
  - Registration information from sensors installed in applications
    - Inserted by dynamic optimizer or user
- Output
  - Rule based contract monitor that decides when contract violation is serious enough to merit reconfiguration
    - Based on information from sensors

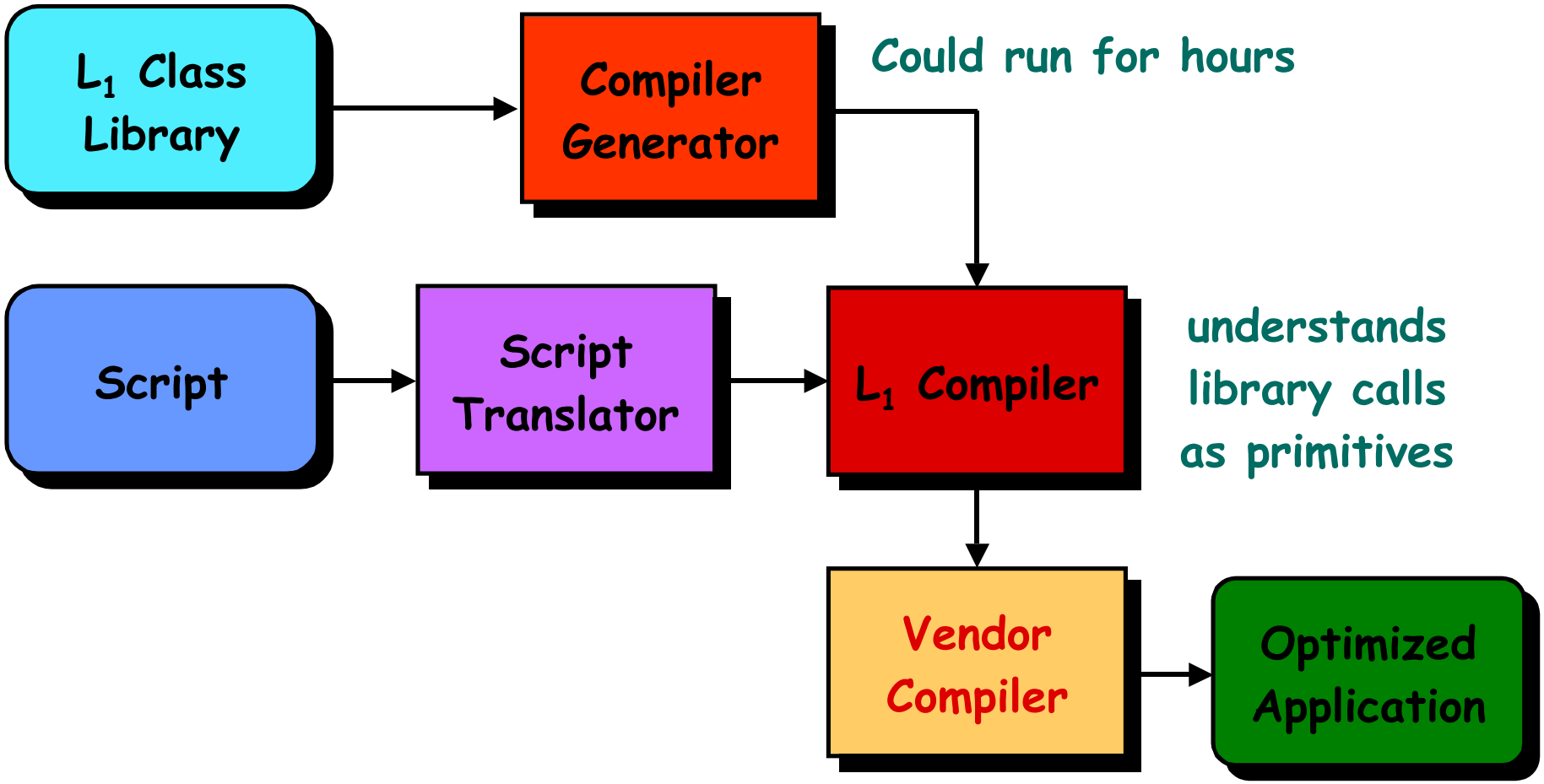
High Level Programming System

- Built on Libraries Coded by Professionals
  - Included mapping strategies and cost models
  - High level knowledge of integration strategies
- Integration Produces Configurable Object Program
  - Integrated mapping strategy and cost model
  - Performance enhanced through context-dependent variants
  - Context includes potential execution platform
- Dynamic Optimizer Performs Final Binding
  - Implements mapping strategy
  - Chooses machine-specific variants
  - Inserts monitoring probes
- Strategy: move compilation overhead to PSE-generation time

Telescoping Languages



Telescoping Languages



Testbeds

- MacroGrid (Carl Kesselman)
  - Collection of processors running Globus
    - Consistent software environment
  - At all 8 GrADS sites
    - Availability listed on web page
  - Permits experimentation with real applications
    - Cactus (Ed Seidel, Ian Foster)
- MicroGrid (Andrew Chien)
  - Cluster of processors (currently Compaq Alphas)
  - Runs standard Grid software (Globus, Nexus)
  - Permits simulation of varying loads and configurations
    - Network and processor
  - Extensive performance modeling

Research Strategy

- Begin Modestly
  - prototype a series of applications using components of envisioned execution system
    - ScaLAPACK Demo, Cactus Demo, XCAT PSE
  - evolve the definition of performance contracts
  - prototype reconfiguration system
    - with performance monitoring, without dynamic optimization
- Move from Hand Development to Automated System
  - identify key steps
  - use experience to expand design of software integration and execution systems
- Experiment
  - use an artificial testbed to test performance under varying conditions
  - move to real applications on the MacroGrid testbed

Management

- Integrated Project
  - Goals include building two major infrastructures
    - Execution environment and Program Preparation System
  - Diverse backgrounds required
    - Many sites involved in research
  - Extensive planning required
- Workshops
  - Three research workshops per year
    - Summary of progress, discussion of technical direction
  - Workshops include planning meeting by Executive Committee
- Documentation Series
  - Numbered design and planning documents
    - in planning stage

Site Visit Schedule

- Introduction
  - Overview/Original GrADS Design [Ken Kennedy]
  - Evolution of the GrADS Software Architecture an Lessons Learned [Fran Berman]
- Experiments
  - The ScaLAPACK Experiment [Jack Dongarra]
  - Experiments with Cactus [Ian Foster]
- Performance Modeling and Contracts [Ruth Aydt]
- Testbeds
  - MacroGrid [Carl Kesselman]
  - MicroGrid [Andrew Chien]
- Futures
  - Program Preparation System [Keith Cooper]
  - G-Commerce [Rich Wolski]

Milestones

- Year 1
  - MicroGrid
  - Implementation of Applications on MacroGrid and MicroGrid
  - System architecture design
  - Information services design and mechanisms
- Year 2
  - Prototype GrADSoft components and validate on testbeds
    - performance models, adaptive app-level scheduling, performance measurement, early dynamic optimizer, new library interfaces with performance models
  - Component interoperation interface development
- Year 3
  - Emergence of GrADSoft Toolkit
- Discussion in individual talks

Summary

- Goal:
  - Build programming systems for the Grid that enable ordinary users to develop and run applications in this complex environment
- Execution of applications must be adaptive
  - In response to changing loads and resources
- Software support is challenging
  - Must manage execution to reliable completion
  - Must prepare a program for execution
  - Should support high-level domain-specific programming
    - For breadth of user community
- Research strategy involves developing a series of prototypes
  - Evolve the infrastructure based on lessons learned