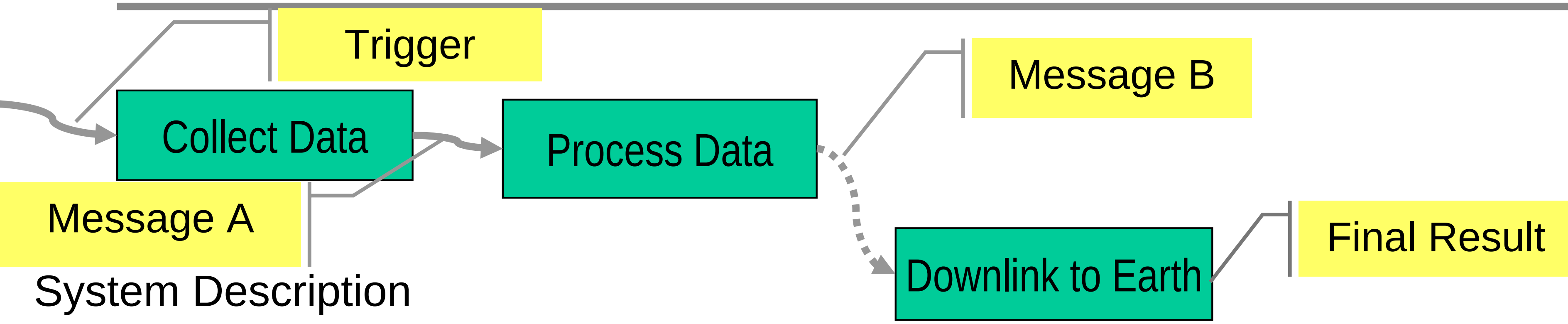
Decision-Theoretic Planning with (Re)Deployment of Components in Distributed Real-time & Embedded Systems

Douglas C. Schmidt
schmidt@dre.vanderbilt.edu

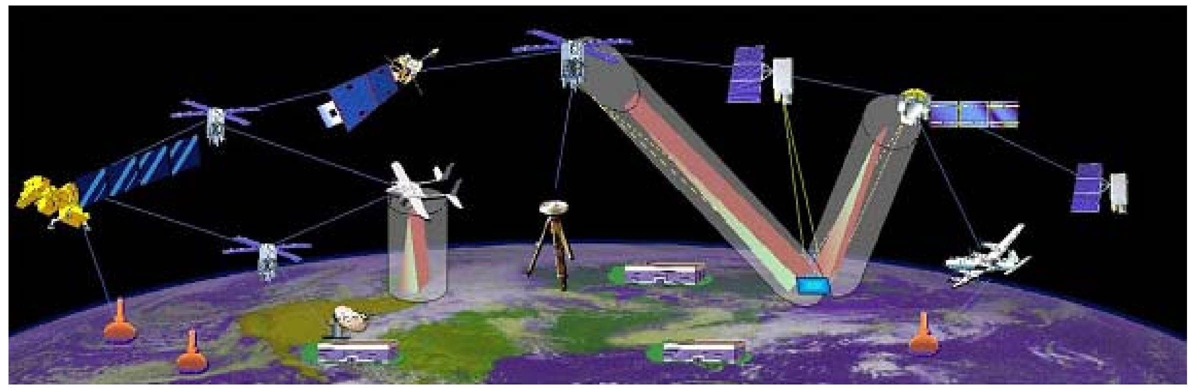
Nishanth Shankaran, John S. Kinnebrew,
Gautam Biswas, Dipa Suri, & Adam S. Howell

Motivating Application: Earth Science Enterprise Mission



System Description

- End-to-end systems-tasks/work-flows represented as *operational string* of components



- Classes of operational strings with respect to importance
  - Mission Critical, Mission Support, & Best Effort
- Operational strings simultaneously share resources
- Strings are dynamically added/removed from the system based on mission & mode

System requirements

1. Automatically & accurately adapt to dynamic changes in mission requirements & environmental conditions
2. Handle failures arising from system failures

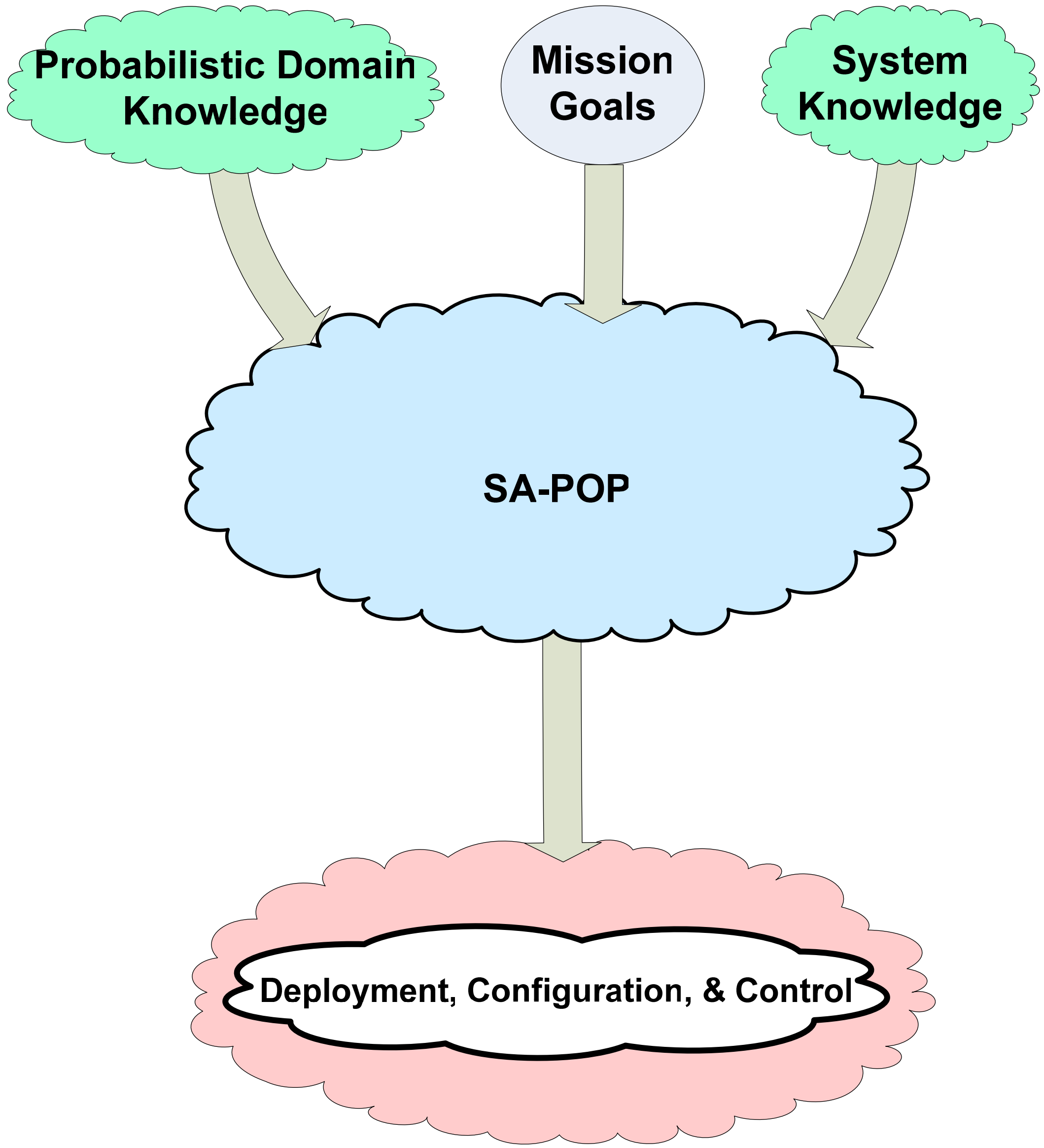
SA-POP Research and Development Challenges

Research Challenges

1. Efficiently handle uncertainty in planning
2. Incorporate resource-aware scheduling with planning

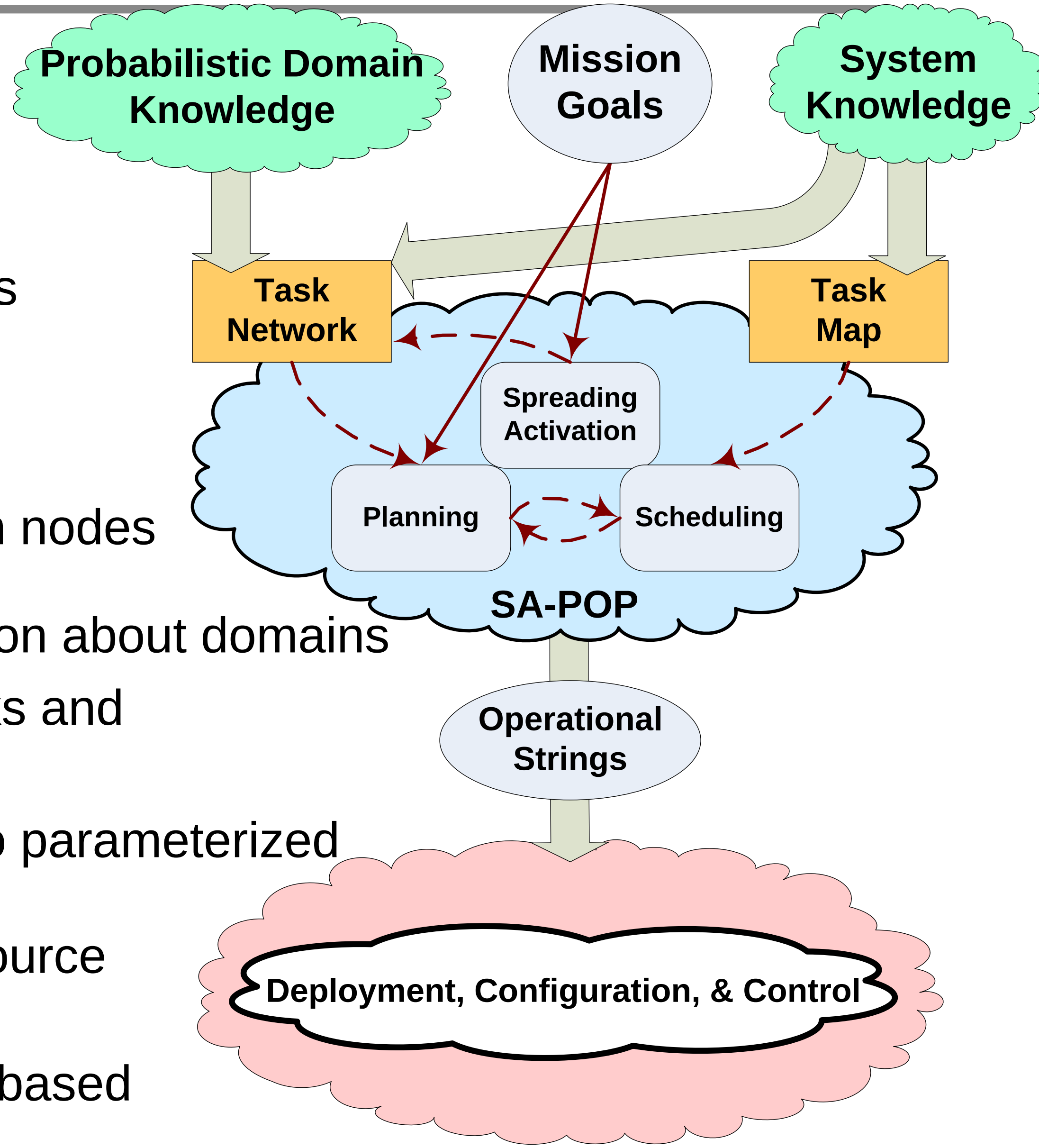
Development Challenges

1. Take advantage of functionally interchangeable components to efficiently meet resource constraints
2. Plan with multiple interacting goals, but produce distinct operational strings



SA-POP is available at:
http://www.dre.vanderbilt.edu/~jkinnebrew/SA-POP/index.html

SA-POP: Planning in DRE Systems with Components



Task is an abstraction of functionality

- Multiple (parameterized) components may have the same function but different resource usage

Task Network specifies probabilistic effects and requirements for tasks

- Condition nodes specify data flow and system/environmental conditions
- Task nodes have links to/from condition nodes specifying effects/preconditions
- Links incorporate probabilistic information about domains

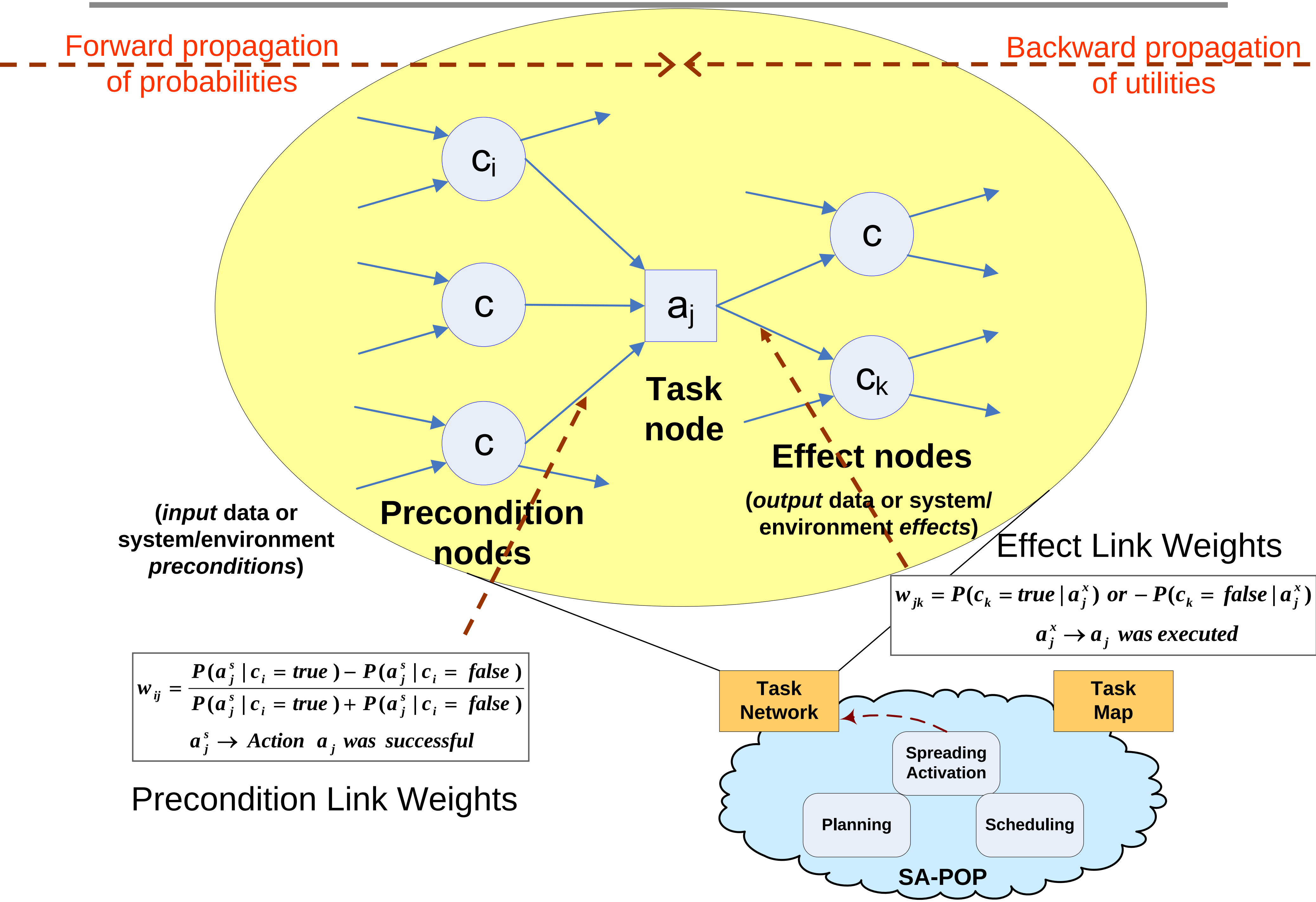
Task Map allows conversion between tasks and components

- Maps tasks (functionality abstraction) to parameterized components (implementation)
- Associates expected or worst case resource usage with each implementation

Operational String specifies a component-based application to achieve a goal

- Set of tasks along with ordering and timing constraints
- Data connections between tasks
- Implementation (parameterized component) suggested for each task

SA-POP: Expected Utility Calculation using Spreading Activation



SA-POP: Operational String Generation

Four hierarchical decision points in each interleaved planning+scheduling step:

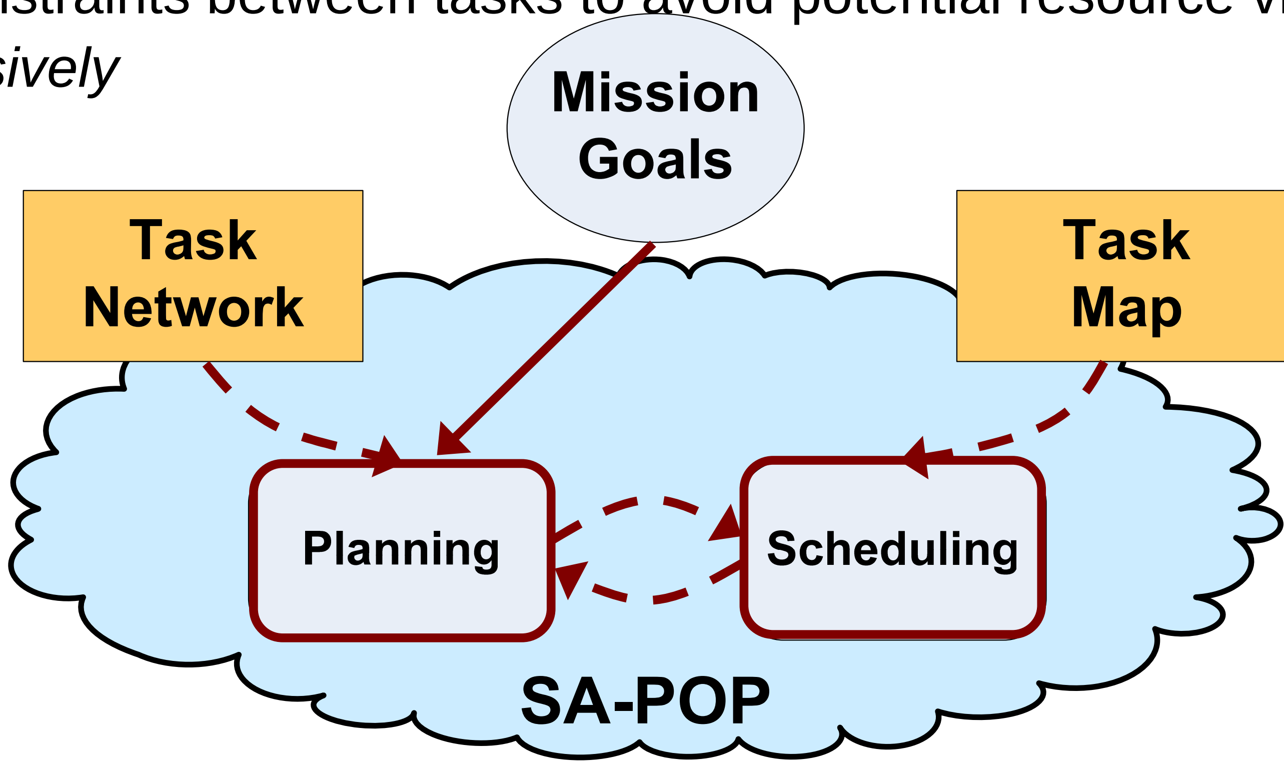
Partial Order Planning:

1. **Goal/subgoal choice**: choose an open condition, which is goal or subgoal unsatisfied in the current plan.
2. **Task choice**: choose a task that can achieve current open condition.

Resource Constrained Scheduling:

3. **Task instantiation**: choose an implementation for this task from the Task Map.
4. **Scheduling decision(s)**: adjust task start/end time windows and/or add ordering constraints between tasks to avoid potential resource violations.

Continue recursively



RACE Research & Development Challenges

Research Challenges

1. Efficiently allocate computing & network resources to application components
2. Avoid over-utilization of system resources – ensure system stability
3. Maintain QoS even in the presence of failure
4. Ensure end-to-end QoS requirements are met – even under high load conditions

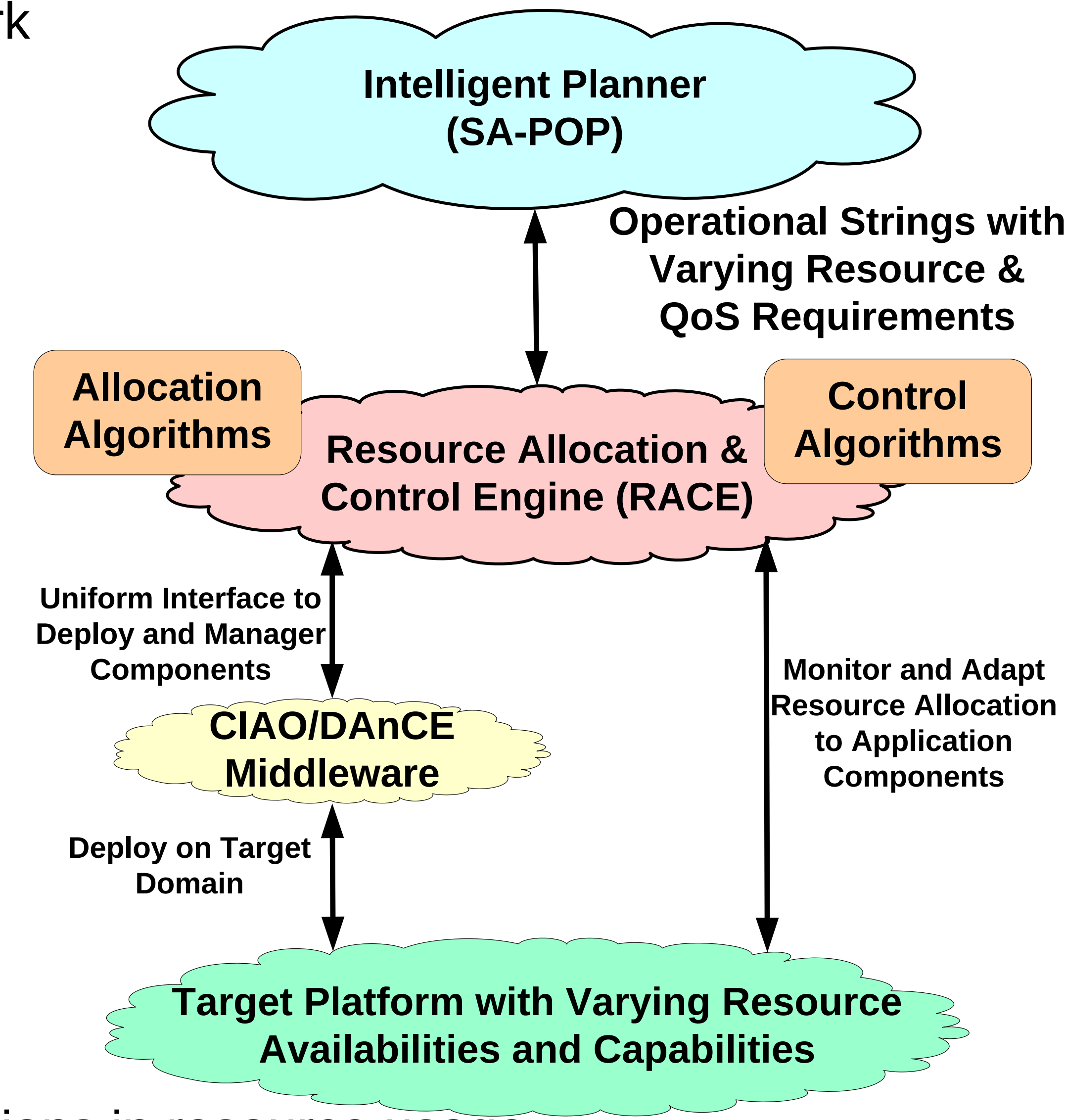
Development Challenges

1. Need multiple resource management algorithms depending on application characteristics & current system condition (resource availability)

2. Single resource management mechanism customized for a specific mission goal or set of mission goals might be effective for that specific scenario
3. However, can not be reused for other scenario ➔ Reinvent the wheel for every scenario

RACE Functional Architecture

- Dynamic resource management framework atop CORBA Component Model (CCM) middleware (CIAO/DAnCE)
- Allocates components to available resources
- Configure components to satisfy QoS requirements based on dynamic mission goals
- Perform run-time adaptation
  - Coarse-grained mechanisms
    - React to new missions, drastic changes in mission goals, or unexpected circumstances such as loss of resources
    - e.g., component re-allocation or migration
  - Fine-grained mechanisms
    - Compensate for drift & smaller variations in resource usage
    - e.g., adjustment of application parameters, such as QoS settings



RACE Software Component Architecture

Input Adapter

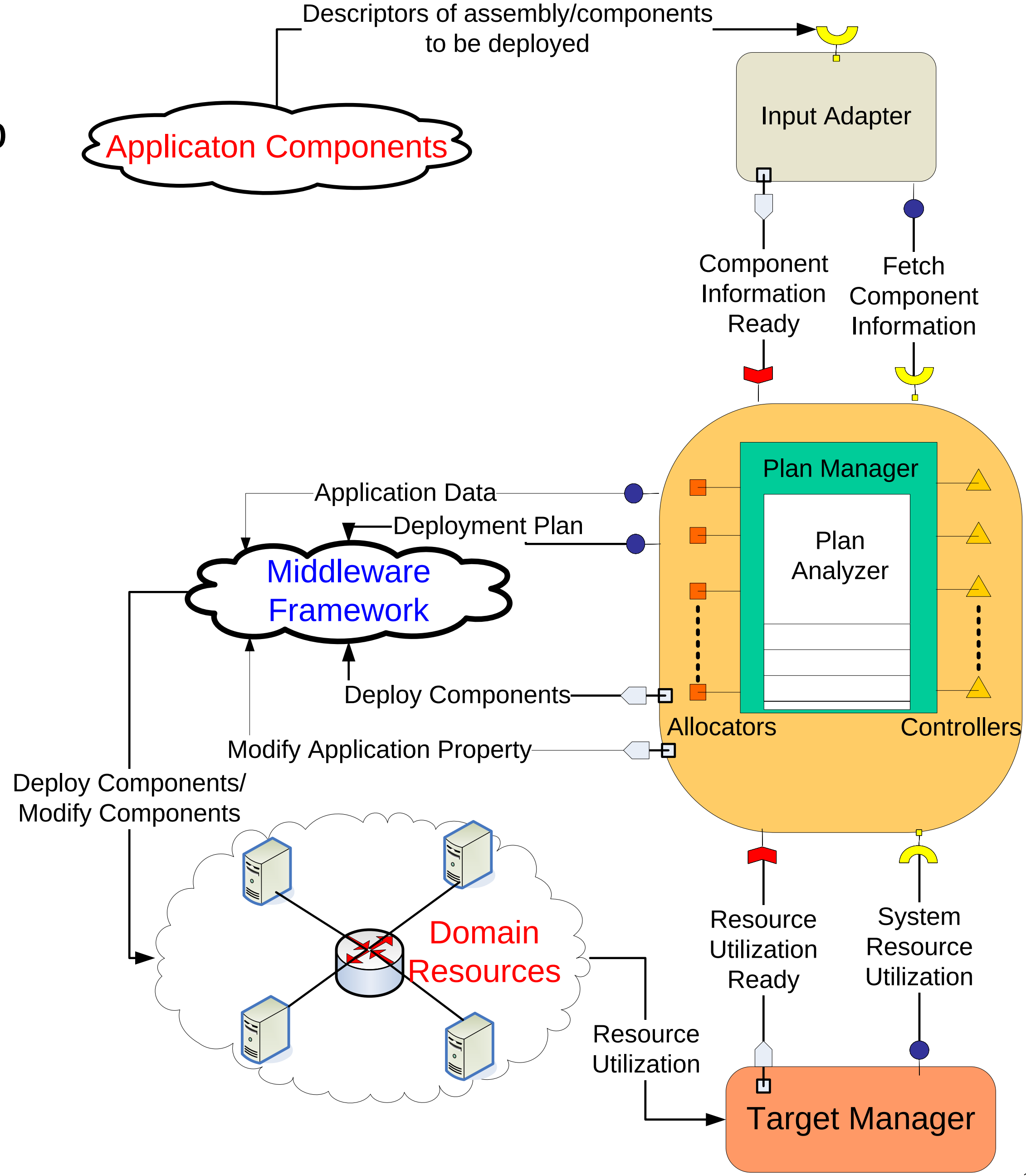
- A generic interface that translates application components descriptors to internal data structure

Plan Analyzer

- Examines input descriptors & select appropriate allocation algorithms based on application characteristics
- Add appropriate application QoS & resource monitors

Planner Manager

- Maintains a registry of installed planners (algorithm implementations) along with their over head & currently executing sequences of planners that are generated by the Plan Analyzer



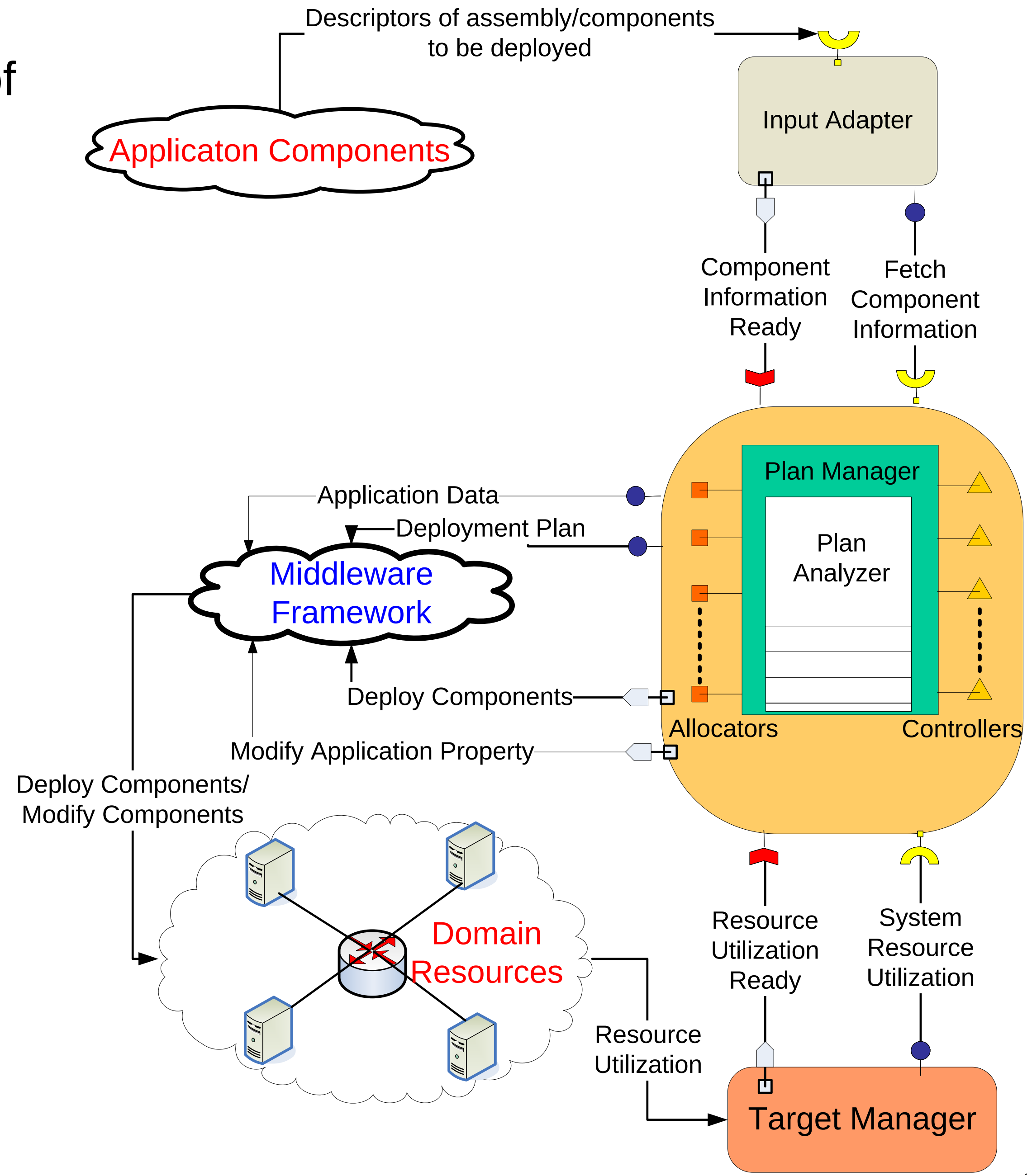
RACE Software Component Architecture

CIAO/DAnCE Middleware

- A CCM middleware framework atop of which components of the operational strings are deployed

Target Manager

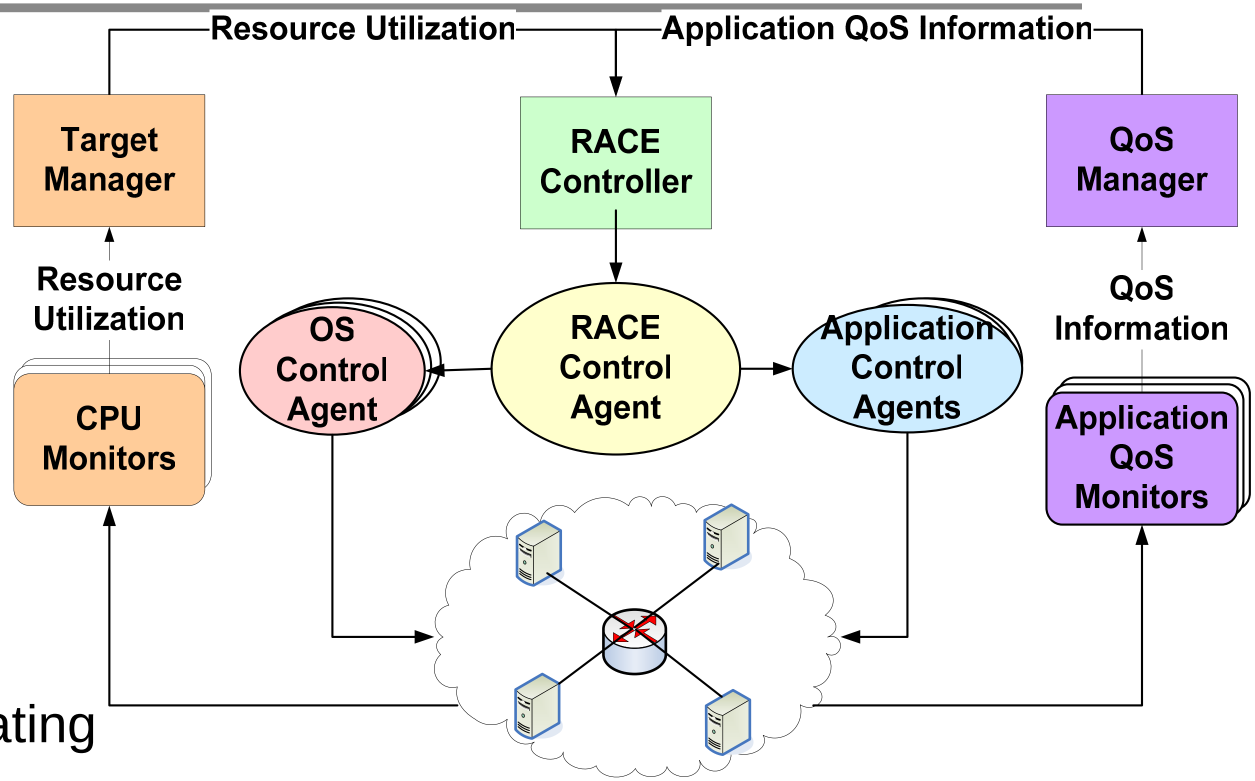
- Runtime resource utilization monitor that tracks the utilization of system resources, such as onboard CPU, memory, and network bandwidth utilization.



RACE: Addressing Dynamic Resource Management Challenges

Need for a control framework

- Allocation algorithms allocates resource to components based on current system condition & estimated resource requirements
- No accurate *apriori* knowledge of input workload & how the execution time depend on input workload



- Dynamic changes in system operating modes

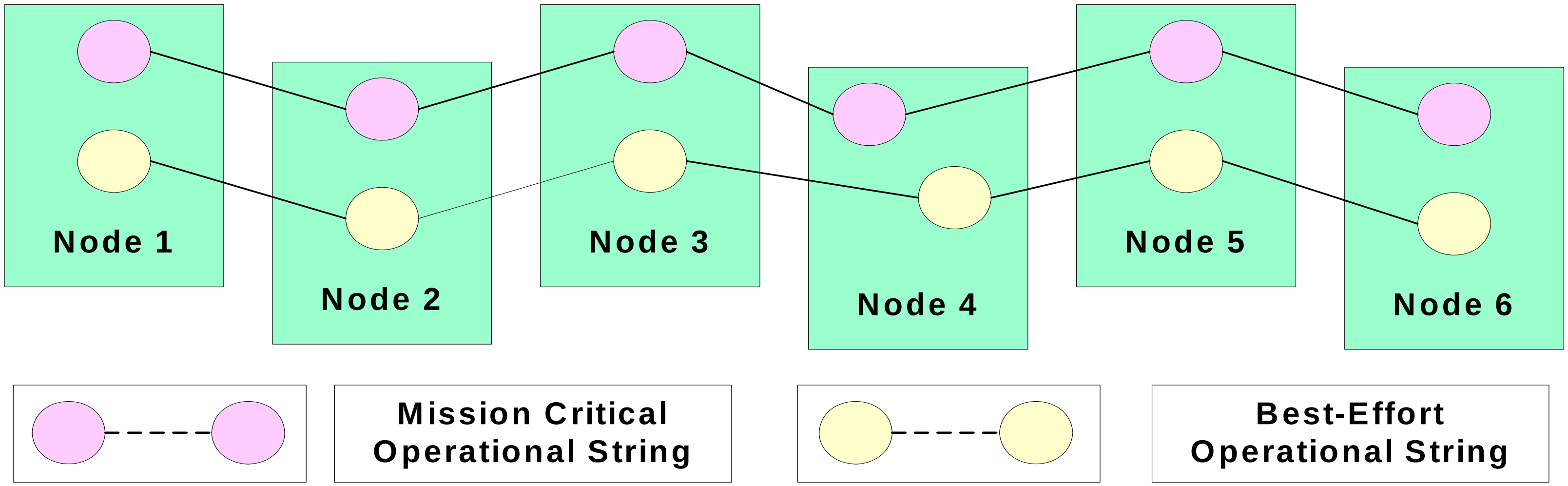
Control objectives:

- Ensure end-to-end QoS requirements are met at all times
- Ensure resource utilization is below the set-point – ensure system stability

- RACE Controller: Reallocates resources to meet control objectives
- RACE Control Agents: Maps resource reallocation to OS/application specific parameters

RACE is available with the *Component-Integrated ACE ORB* (CIAO) (http://deuce.doc.wustl.edu/Download.html)

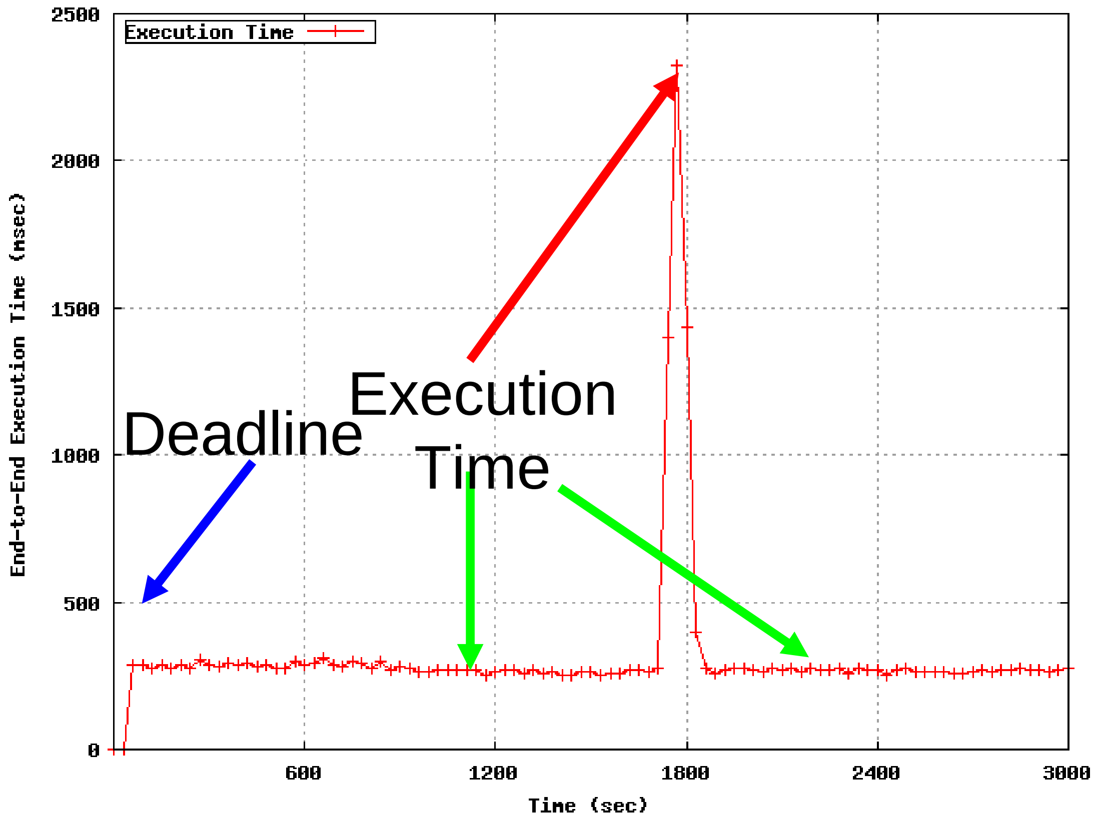
Experimentation Results – Hardware / Software Testbed



- Experiments were performed on the ISISLab testbed at Vanderbilt University (www.dre.vanderbilt.edu/ISISlab)
- Hardware Configuration
  - 6 nodes with 2.8 GHz Intel Xeon dual processor, 1 GB physical memory, 1Ghz Ethernet network interface, and 40 GB hard drive
- Software Configuration
  - Redhat Fedora Core Release 4 operating
  - ACE+TAO+CIAO Middleware
- Two operational strings - one mission-critical, one best effort -with 6 components each were deployed on 6 nodes

Experimentation Results – Performance Analysis

- An end-to-end deadline of 500 ms was specified for the mission-critical operational string
- Mission critical string was deployed at time T = 0s, and best-effort was deployed at time T = 1800 sec
- Until T = 1800 sec, end-to-end execution time of mission critical string is lower than its deadline
- At T = 1800 sec, end-to-end execution time of mission critical string is way above its deadline



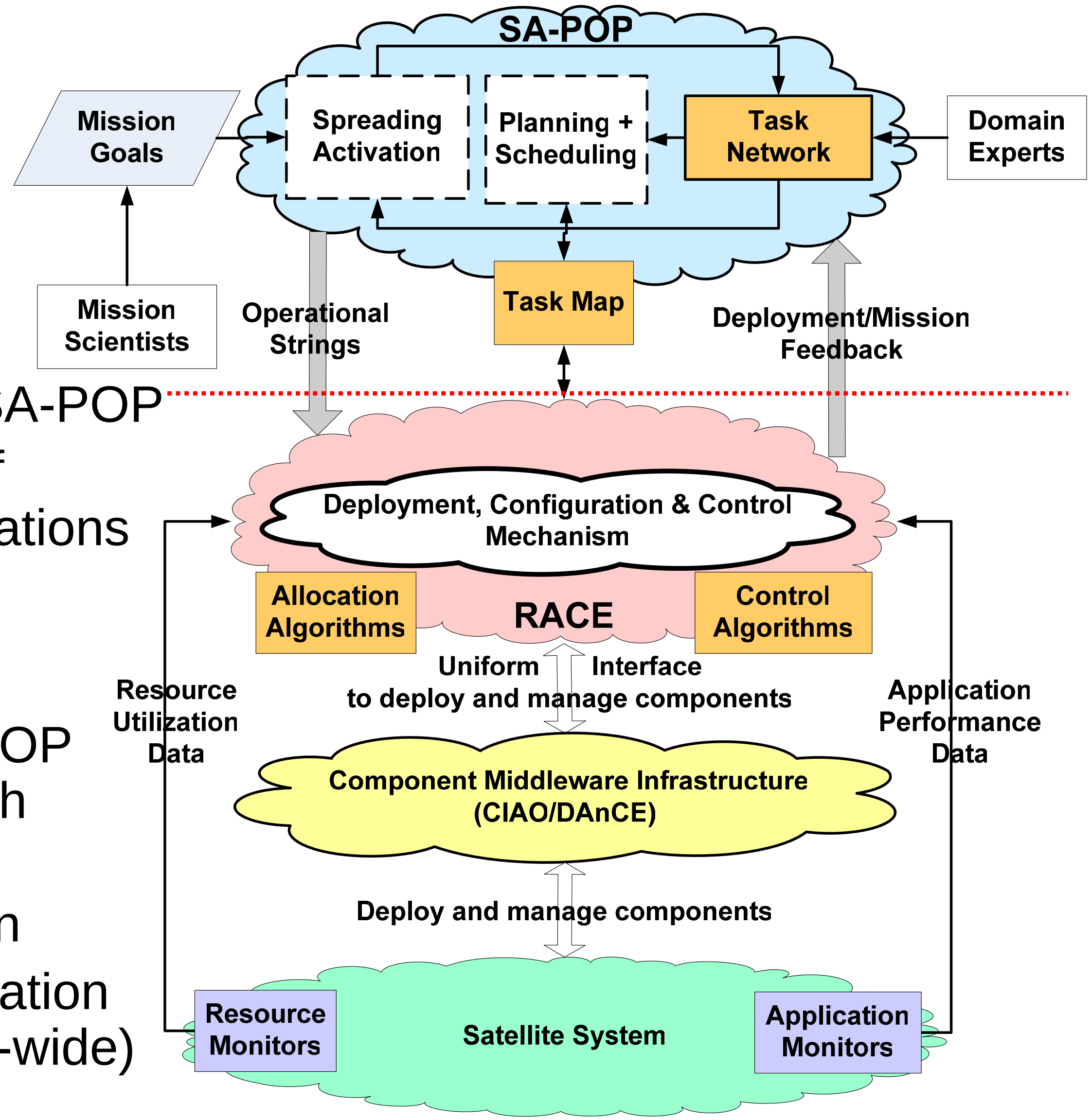
- This is due to excessive resources consumption by best-effort string
- RACE reacts to increase in execution time by perform adaptive system control modifications by modifying operating system priority, scheduler class and/or tearing down lower priority operational string(s)

RACE ensures end-to-end deadline of mission critical string is met even under fluctuations in resource availability/demand

Lessons Learned from SA-POP and RACE Integration

Flexible

- Pluggable resource allocation and control algorithms in RACE
- SA-POP task network and task map tailored to domain
- Unlimited combinations of goals, goal priorities, and timing requirements in SA-POP
- Shared task map allows substitution of functionally equivalent task implementations by RACE



Scalable

- Separation of concerns between SA-POP and RACE limits search spaces in each
- SA-POP handles cascading planning choices in operational string generation
- SA-POP only considers resource allocation *feasibility* with *course-grained* (system-wide) resource constraints
- RACE handles resource allocation *optimization* with *fine-grained* (individual processing node) resource constraints and dynamic control for fixed operational strings

Lessons Learned from SA-POP and RACE Integration

Dynamic

- SA-POP task network with spreading activation provides expected utility information for generating robust applications in uncertain environments
- SA-POP replanning achieved efficiently with incrementally updated task network and plan repair as necessary
- RACE control algorithms alleviate need for replanning in many cases
- RACE provides reallocation and redeployment of revised operational strings when replanning is necessary