Building Automated Health Checks into the Grid

International Summer School on Grid Computing

July 22, 2003

Michael T. Feldmann

Center for Advanced Computing Research

California Institute of Technology

Goals for talk

- Answer/motivate a few questions
  - What is grid health monitoring?
  - Why is grid health monitoring important?
  - Do I need a grid health monitoring system?
- Motivate utility of health monitoring systems
- Introduce a particular implementation
  - Inca - TeraGrid

Outline

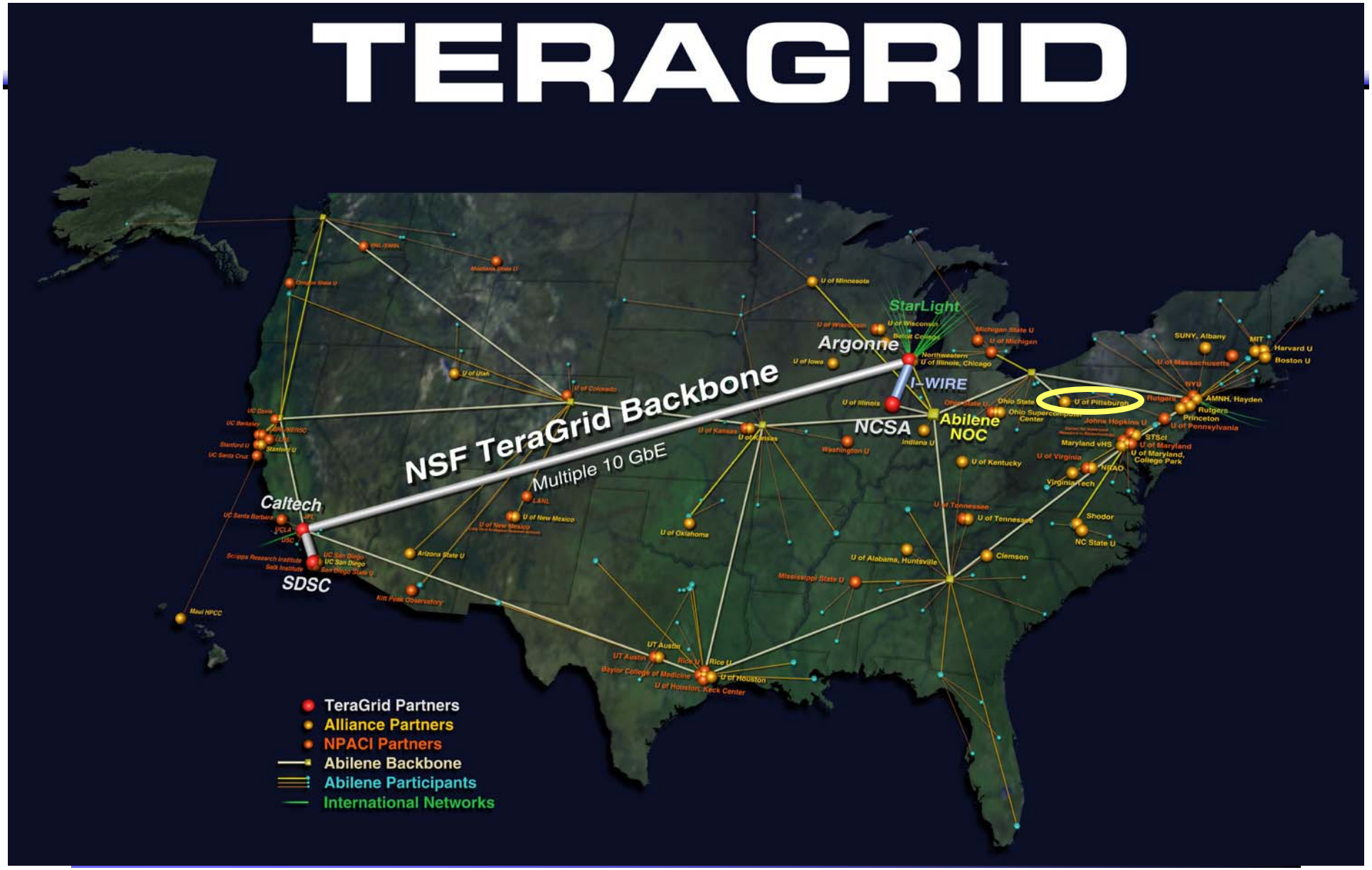
- Background
- Define health monitoring needs
- Design framework to meet these needs
- How to design framework that really works!
- Conclude

What is the "grid-computing"?

- The whole question of this summer school!
- Various definitions/characteristics
  - Shared resource environment
  - Distributed resources
  - Loosely-coupled resources
  - Potential shared interfaces in heterogeneous environment
  - political/social components
  - ...the list goes on ...

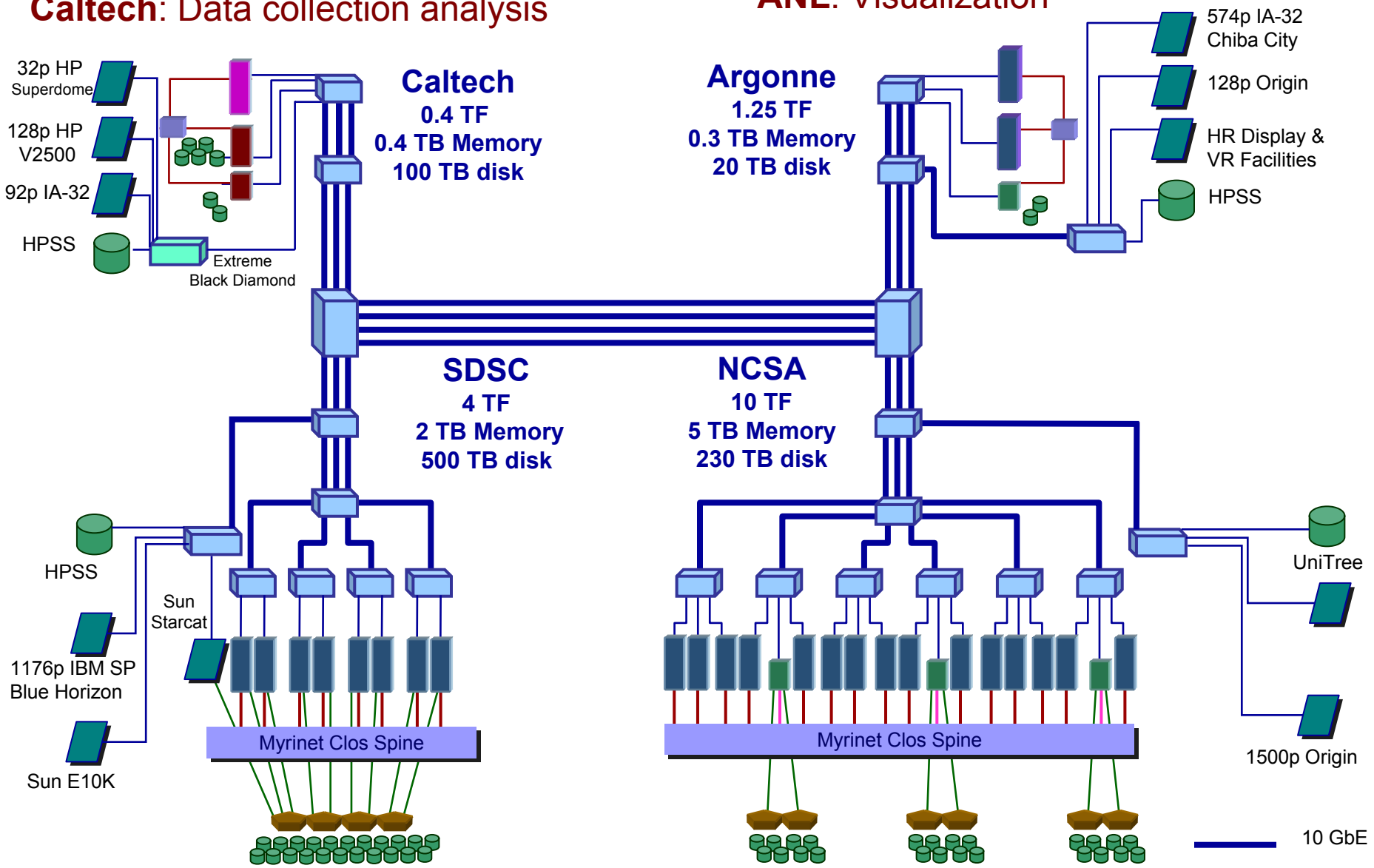
My experience

- Teragrid
  - applications consultant
  - interface with users and support staff
- Inca
  - develop python API for unit tests
- Scientist
  - quantum chemist
  - application development & user bias


TERAGRID
NSF TeraGrid Backbone
Multiple 10 GbE
Caltech
SDSC
Argonne
StarLight
I-WIRE
NCSA
Abilene
NOC
TeraGrid Partners
Alliance Partners
NPACI Partners
Abilene Backbone
Abilene Participants
International Networks

Caltech: Data collection analysis

ANL: Visualization



SDSC: Data-Intensive

NCSA: Compute-Intensive

What is the Teragrid?

- NSF grid computing collaboration
  - Members
    - CACR-Caltech
    - NCSA
    - SDSC
    - ANL
    - PSC
    - ... other new members ...
  - Resources
    - ~15 Tflop
    - ~40Gb/sec backbone
    - ~1PB fast disk cache
    - ~7TB RAM
    - visualization facilities
    - ... more ...

What else is the Teragrid?

- Learning experience
  - How do we get several sites to work together?
  - How do we define reasonable interfaces?
  - What tools can we provide to all users?
- The Teragrid is a great opportunity to explore the design of grid-computing environments.
- Goals of designers: Make everyone happy!

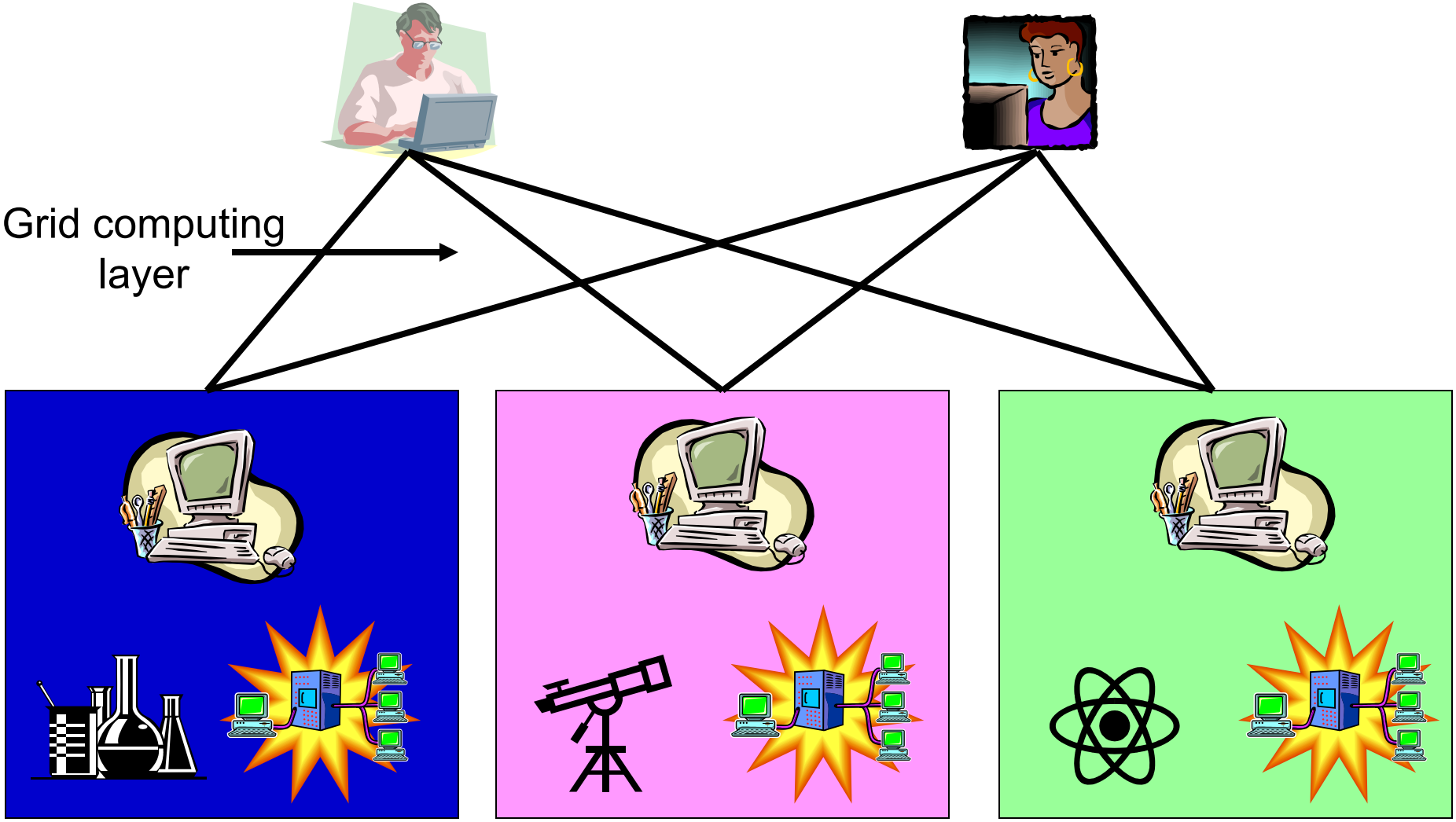
What does the support staff want?

- Easily maintained/robust environment
- Grid health monitoring
  - cluster hardware
  - software stack correctness
  - benchmark performance/correctness
- Simple mechanism to interact with user problems
- Ability to find problems before users do!
- Verify minimum level of functionality
- Enable "real science" to be done

What does a user want?

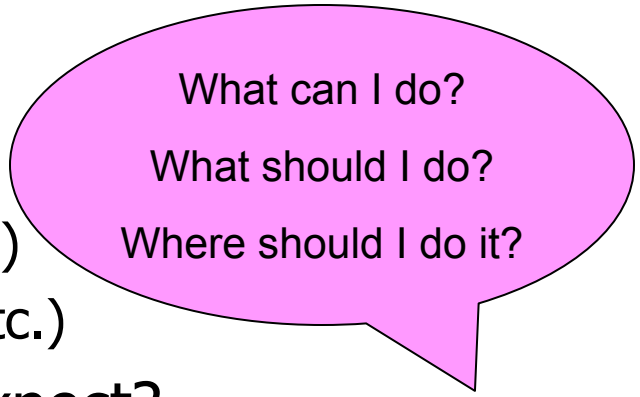
- Flexible yet easy to use environment
- Robust environment
- Fast response time to fix broken system
- Powerful systems
- Application performance/correctness
- Some minimum level of functionality
- Get their work done!

Simple view of grid computing



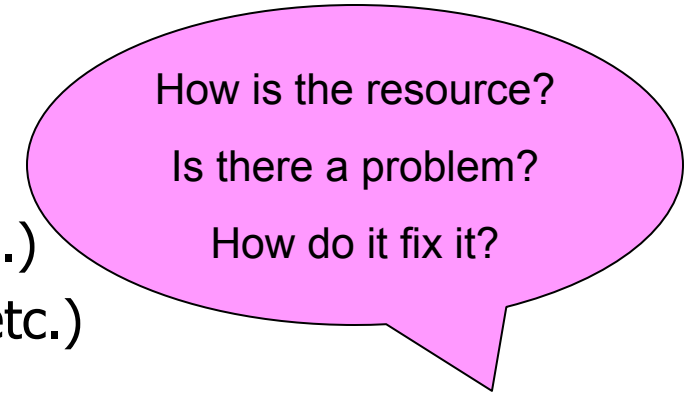
Possible user actions?

- Query grid health/history?
  - What resources exist?
    - hardware (compute, instruments, etc.)
    - software (math libraries, compilers, etc.)
  - What kind of performance can I expect?
    - compute, I/O, network, etc.
- Take actions based on health/history
  - Submit my data intensive task to site "A"
  - Store my data at site "B"
  - Run small development tasks on site "C"
  - Submit my compute intensive task to job manager "X"



Possible support staff actions?

- Query grid health/history?
  - Are the resources functioning?
    - hardware (compute, instruments, etc.)
    - software (math libraries, compilers, etc.)
  - Is performance out of the norm?
    - compute, I/O, network, etc.
- Take actions based on health/history
  - Find problem
  - Fix problem
  - Document and archive problem/solution



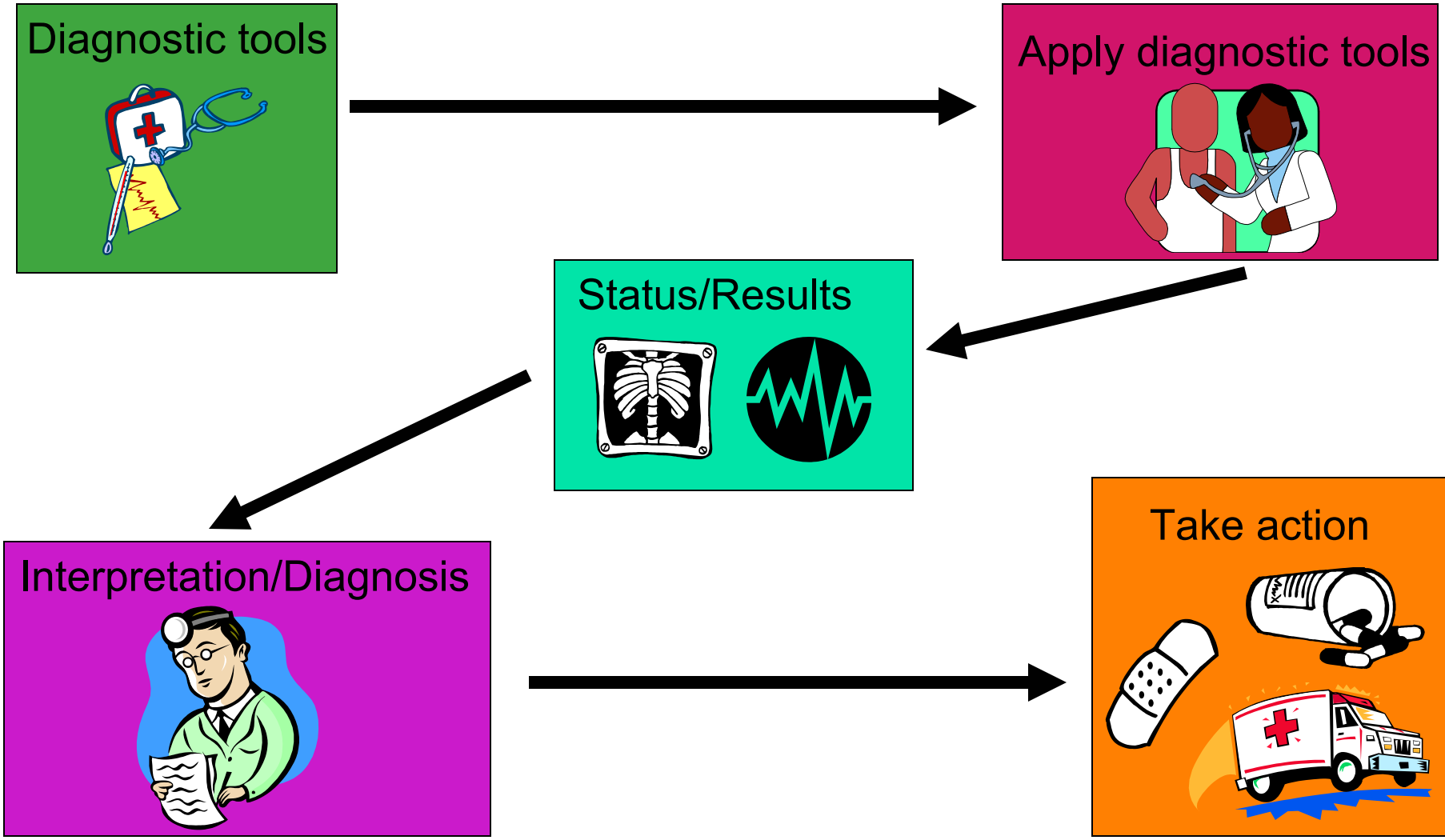
Similarities

- Developers, and support staff often ask the same questions about the system
- Can we build one tool to satisfy everyone's needs?!?
- Each group takes different actions based on the system status but a common tool may be possible

How to determine "resource health"?

- Content questions:
  - What information might be useful?
  - What might someone want to see?
  - How do we probe the system to get "health" information?
  - What do we archive?
- How invasive are these probes?
- How do we construct a health monitoring framework?

Resource health



Other related/important projects

- User docs
- Unit tests
  - Java unit test
  - python unit test
  - ...
- Grid health monitoring
  - EDG: R-GMA
  - NPACI "hotpage" type reporters
  - CACR-nodemon
  - ...

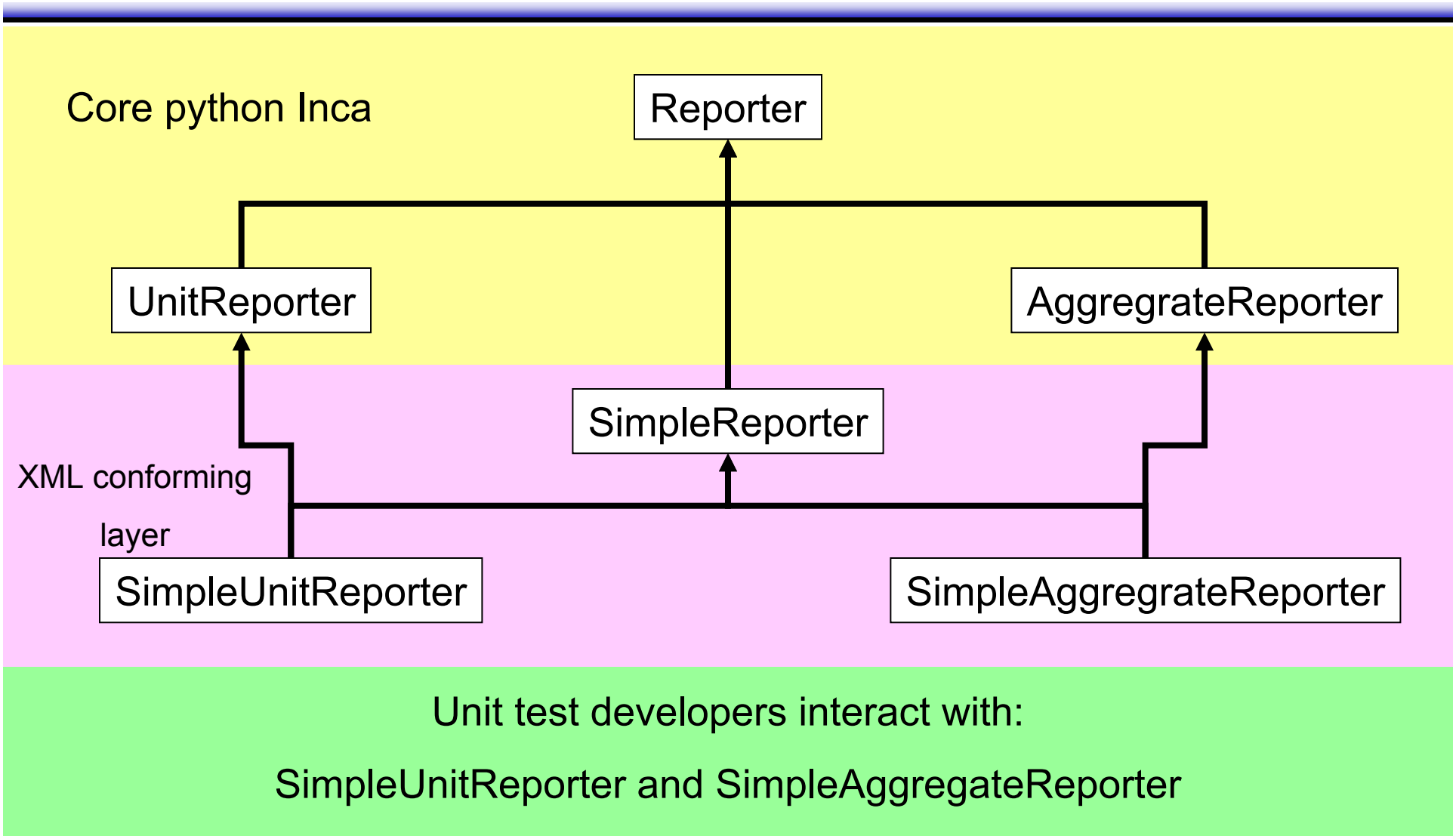
Diagnositic Tools

- Unit tests
  - A unit test is the smallest unit of testing code that can be checked against some resource
- Reporters
  - Various classes of "Reporter"
  - Provide a minimal set of data to be given to the archiving/publishing mechanism
  - Common use is to put unit tests in a simple Reporter or into some type of aggregate Reporter

Reporter structure

- A Reporter must report some minimal set of status output
- A Reporter can be nested within an AggregateReporter
- A Reporter must be self-contained (can be copied anywhere and run)
- Output must conform to established schema

Python API example (other APIs exist)



Unit test developers interact with:

SimpleUnitReporter and SimpleAggregateReporter

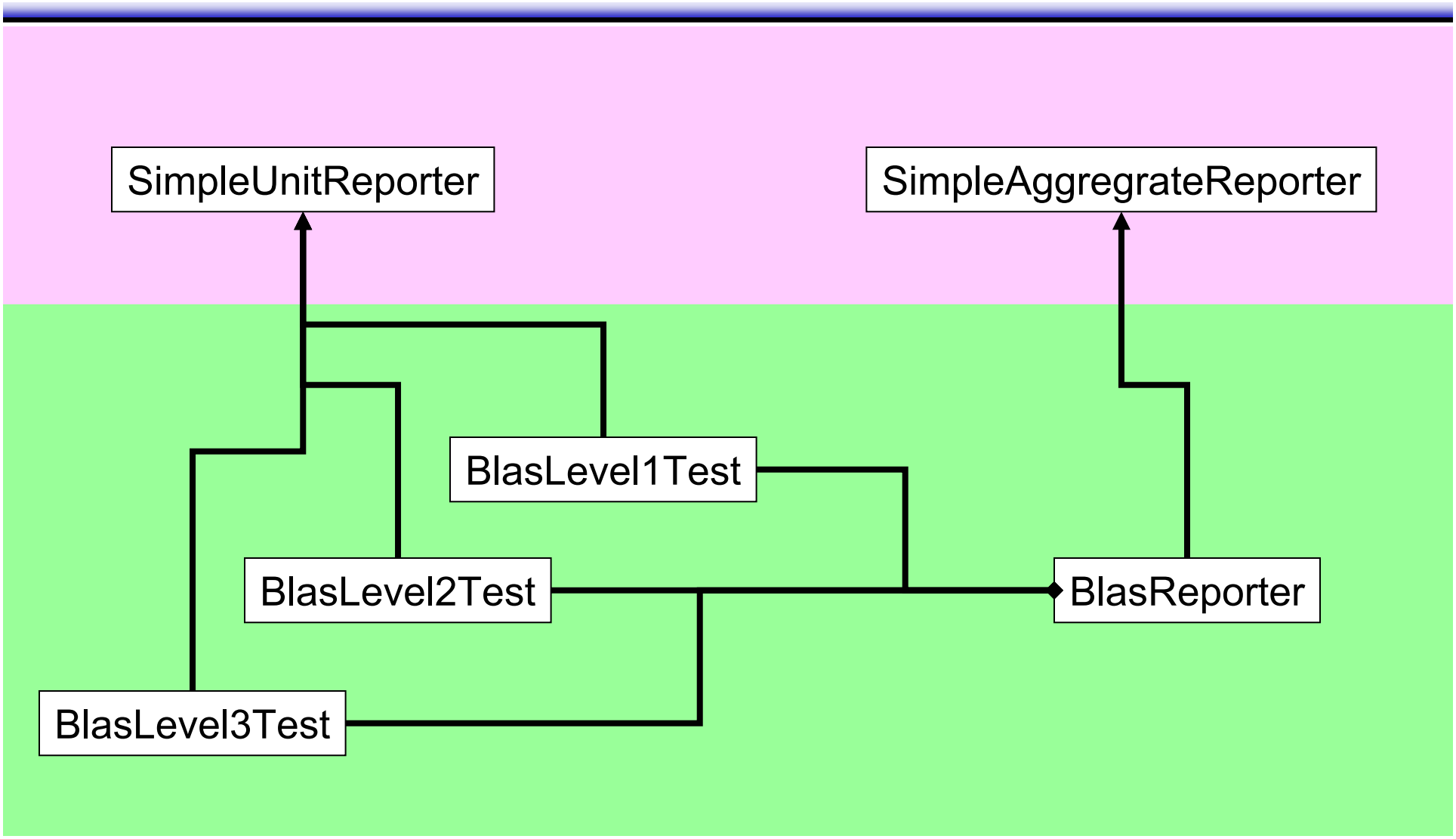
Example SimpleUnitReporter

- get machine information
- get user information
- get environment information
- build test
- run test
- analyze test output
- put test output into xml that conforms to the established schema

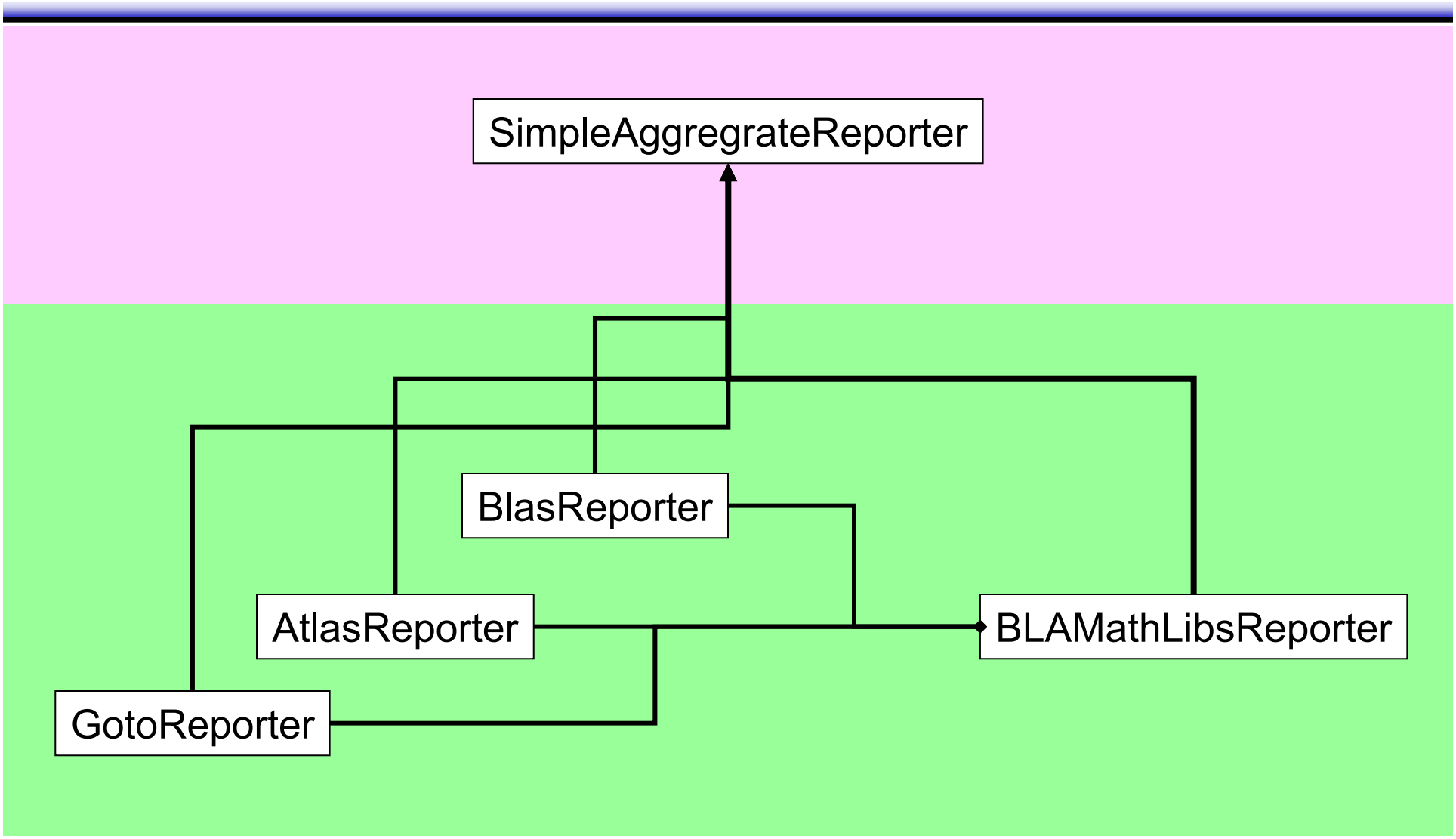
Example SimpleAggregateReporter

- get machine information
- get user information
- get environment information
- register some Reporters
- run each Reporter *(satisfy AR dependencies)*
- put test output into xml that conforms to the established schema

Python API example



cont. Python API example



Example SUR

```
#!/usr/bin/python

#This class will test what modules load into python without any trouble
import os,string,sys
from Inca.Test import *

class module_list_loader_Reporter(SimpleUnitReporter):
   def __init__(self):
      SimpleUnitReporter.__init__(self)
      self.name = "module_list_loader_Reporter"
      self._test_script = "module_load_tester.py"
      self.module_list_file = "module_list.python.2.2.1"
      self._module_list = []
      self._results = []
      self.platforms = ["universal"]#,"Unix"]
      self.description = "Attempt to load a list of python modules."

   def get_python_module_list(self):
      file = open(self.module_list_file,"r")
      lines = file.readlines()
      modules = []
      for line in lines:
         chunks = string.split(line)
         valid_module=0
         for platform in self.platforms:
            if(platform == chunks[1]):
               valid_module = 1
         if(valid_module):
            self._module_list.append(chunks[0])
      return 0
```

```
   #build a minimal script that tries to load the module
   def build_module_loader_tester(self,module):
      file = open(self._test_script,"w")
      lines = ""
      lines += "import "+module+"\n"
      file.write(lines)
      return 0

   def attempt_to_load_module(self,module):
      self.build_module_loader_tester(module)
      result = self.system_command("python "+self._test_script)
      return result

   def get_results(self):
      for module in self._module_list:
         tuple = (module,self.attempt_to_load_module(module))
         self._results.append(tuple)
         if(tuple[1]==0):
            success = "Success loading module:\t"+tuple[0]+os.linesep
            self.add_success(success)
      return 0

   def analyze_test(self):
      #failed
      unit = "failed_modules"
      value = len(self.results_dict["failures"])
      self.add_xml_to_body({"ID":"failures",unit:value})
      #successes
      unit = "loaded_modules"
      value = len(self.results_dict["successes"])
      self.add_xml_to_body({"ID":"successes",unit:value})
      return 0

   def run_test(self):
      self.get_python_module_list()
      self.get_results()
```

Example SAR

```
#!/usr/bin/python

import os,string,sys
from Inca.Test import *
from module_list_loader_Reporter import module_list_loader_Reporter

class PYTHON_Reporter(SimpleAggregateReporter):
   def __init__(self):
      SimpleAggregateReporter.__init__(self)
                self.setName("python_unit_test")
                self.setUrl("www.python.org")
                self.setDescription("Test your local version of python.")

   def extractPackageVersion(self):
                self.PackageVersion = string.replace(sys.version,os.linesep,"")

   def execute(self,execute_flag,args="trash"):
                self.setPackageVersion(self.extractPackageVersion())
                module_loader_tester = module_list_loader_Reporter()
                self.addReporter(module_loader_tester)
                if(args!="trash"):
                                return self.execute_AggregateReporter(execute_flag,args)
                else:
                                self.processArgs_auto()
                                return self.execute_AggregateReporter(execute_flag)

if __name__ == "__main__":
   PYTHONtester = PYTHON_Reporter()
   PYTHONtester.execute("FAIL_ON_FIRST")
   print PYTHONtester
```

Example XML output

```
<?xml version="1.0" ?>
<INCA_Reporter>
  <INCA_Version>1.3</INCA_Version>
  <localtime>Thu Jul 17 21:10:59 2003</localtime>
  <gmt>Thu Jul 17 21:10:59 2003</gmt>
  <ipaddr>131.215.148.2</ipaddr>
  <hostname>tg-log-h</hostname>
  <uname>Linux tg-log-h 2.4.19-SMP #4 SMP Wed May 14 07:34:24 UTC 2003 ia64 unknown</uname>
  <url>www.python.org</url>
  <name>python_unit_test</name>
  <description>Test your local version of python.</description>
  <version>0.1</version>
  <INCA_input>
    <help>false</help>
    <version>false</version>
    <verbose>0</verbose>
  </INCA_input>
  <results>
    <ID>results</ID>
    <ModuleLoadingTest>
      <ID>ModuleLoadingTest</ID>
      <LocalPythonVersion>python version: 2.2.3 (#1, Jun 2 2003, 19:59:06) [GCC 3.2]</LocalPythonVersion>
      <ModuleListVersion>2.2.1</ModuleListVersion>
      <failures>
        <ID>failures</ID>
        <number>3</number>
        <failed_module>audioop</failed_module>
        <failed_module>imageop</failed_module>
        <failed_module>rgbimg</failed_module>
      </failures>
      <successes>
        <ID>successes</ID>
        <number>186</number>
      </successes>
    </ModuleLoadingTest>
  </results>
  <exit_status>false<exit_message>ModuleLoadingTest returned too many failures: 3 failure(s)</exit_message></exit_status>
</INCA_Reporter>
```

Resource health

Diagnostic tools

SimpleUnitReporter

SimpleAggregateReporter

Applying Diagnositic Tools

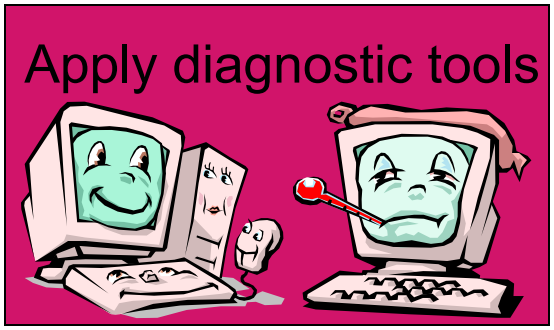
- Each resource needs to run the test harness
- How frequently should we run each test?
- The test harness employed must manage and schedule the application of the unit tests

Resource health

Diagnostic tools

SimpleUnitReporter

SimpleAggregateReporter



Resource Status

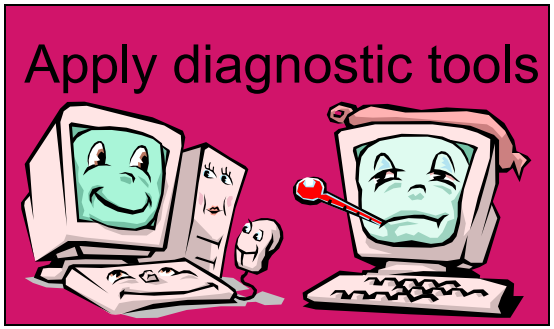
- The SimpleAggregateReporter provides xml that conforms to the Inca schema and is the primary interface with the Inca Harness
- Why is this "Inca schema" important?
  - XML schema provides a "standard form" for someone to fill in
  - Provides an interface for unit test writers and those writing the Inca publishing tools

Resource health

Diagnostic tools

SimpleUnitReporter

SimpleAggregateReporter



Status/Results

xml schema

Interpreting/diagnosis of result

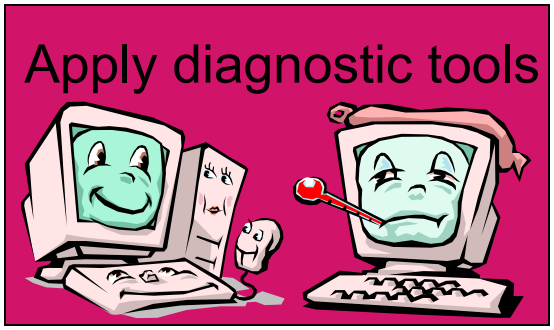
- Inca user interface
  - web interface
    - users can access current and past data
    - users can personalize their view of the Inca archive
      - performance evaluation person may want a lot of detail
      - sys admin might only want correctness information
  - other interfaces
    - applications making direct queries to Inca

Resource health

Diagnostic tools

SimpleUnitReporter

SimpleAggregateReporter



Status/Results

xml schema

Interpretation/Diagnosis

User can interact with

Inca Harness

Taking Action

- This is NOT part of the Inca framework
- Fixing possible problems is the work of a knowledgeable systems support staff member
- Inca is a tool to help easily identify problems and verify a minimal set of requirements have been met to belong to the Teragrid

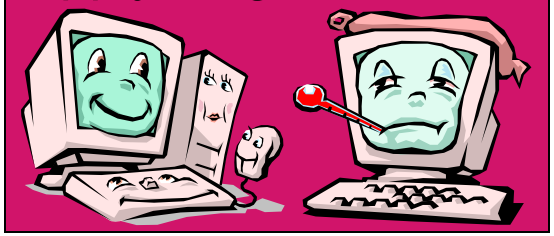
Resource health

Diagnostic tools

SimpleUnitReporter

SimpleAggregateReporter

Apply diagnostic tools



Status/Results

xml schema

Interpretation/Diagnosis

User can interact with

Inca Harness

Take action

human does this

How do we really make this work?!?

- Completely spanning set of unit tests
- "Easy to use" publishing mechanism for archive
- Low resource usage
- Low maintenance
- High robustness

Completely spanning set of unit tests

- We need unit tests that test all aspects of the resources
- Leveraging previous work
  - Many users have there own small set of tests
  - Many sources of tests exists (i.e. netlib.org, experienced sys admins, code self tests, etc.)
- We need unit tests that are correct!
  - It does us no good if a unit test is written which reports incorrect status

Publishing mechanism & ease of use

- The interaction a user or application makes with the archive is of critical importance
- We may have a very rich depot of information but if the user can not easily interact it loses value.
- Inca provides a web interface for human interaction
- Inca will also provide a command line driven interface for easy application interactions

Low resource usage

- Some tests take very few resources
  - version reporters
  - Answering "Does software XYZ exist?"
- Some tests take a lot
  - performance evaluation tests often take a lot
    - large hpl runs
- We must intelligently schedule tasks
  - Stay current enough to be useful
  - Not burden the system

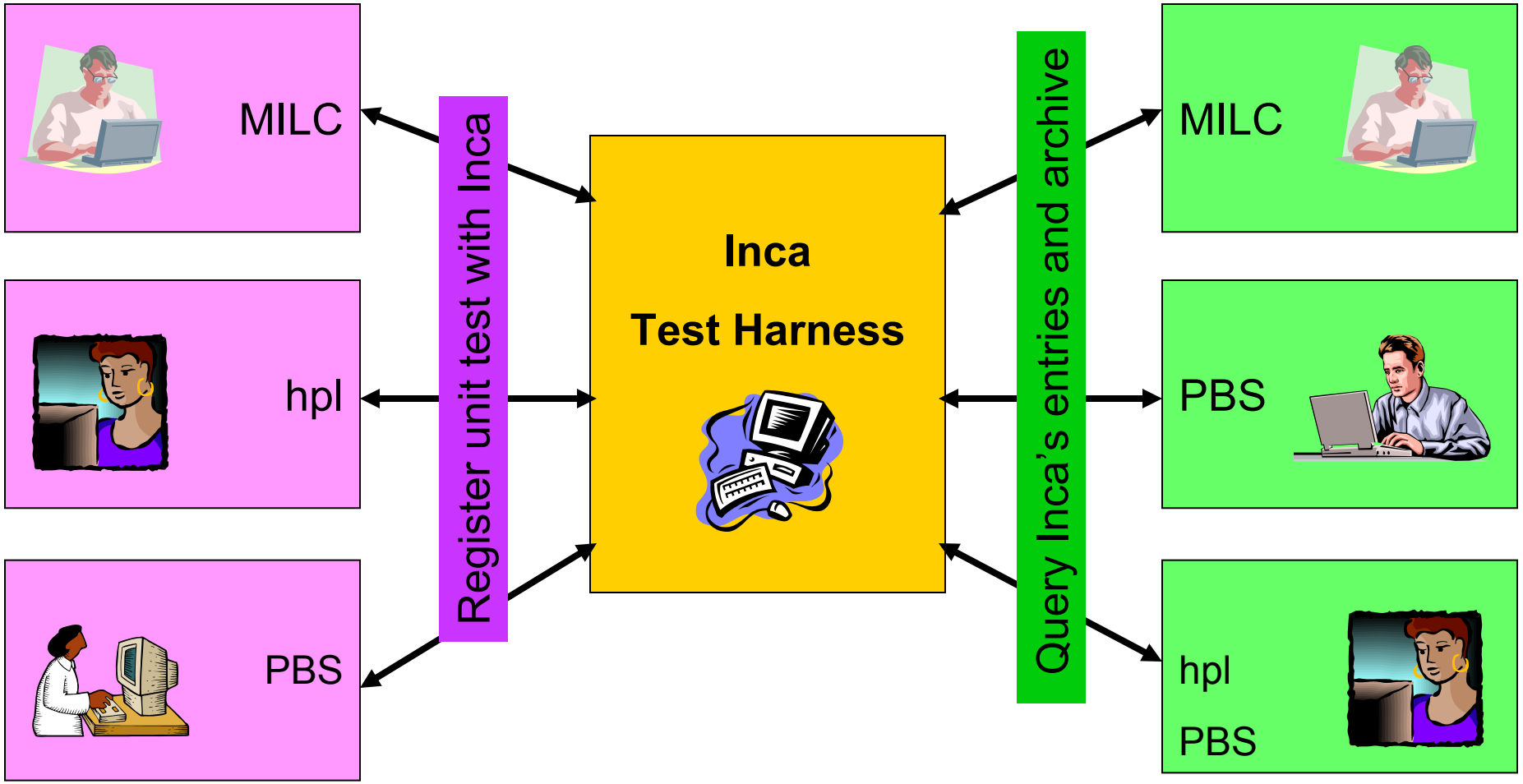
Low maintenance & high robustness

- Inca (-> time will tell)
  - Cons
    - very young code
      - not thoroughly tested over time
  - Pros
    - is still very young
      - small group of people very reactive to possible problems
      - flexibility still exists to recover from such issues
    - engineered with grid computing in mind

How do we leverage current unit tests?

- Unit tests construction
  - must be simple/intuitive to write
  - motivate others to write tests for us!
- Unit test schema
  - must have adequate richness in expression
  - must be easy/intuitive to manipulate with publishing tools

Big picture view of Inca framework



Recap of goals of unit test construction

- Large span of potential problems
  - detect all possible errors before users do
  - shorten debug time
- Leverage existing work
  - low barrier to learn to construct a unit test
  - wrap existing simple/not-so-simple tests

Inca in more detail

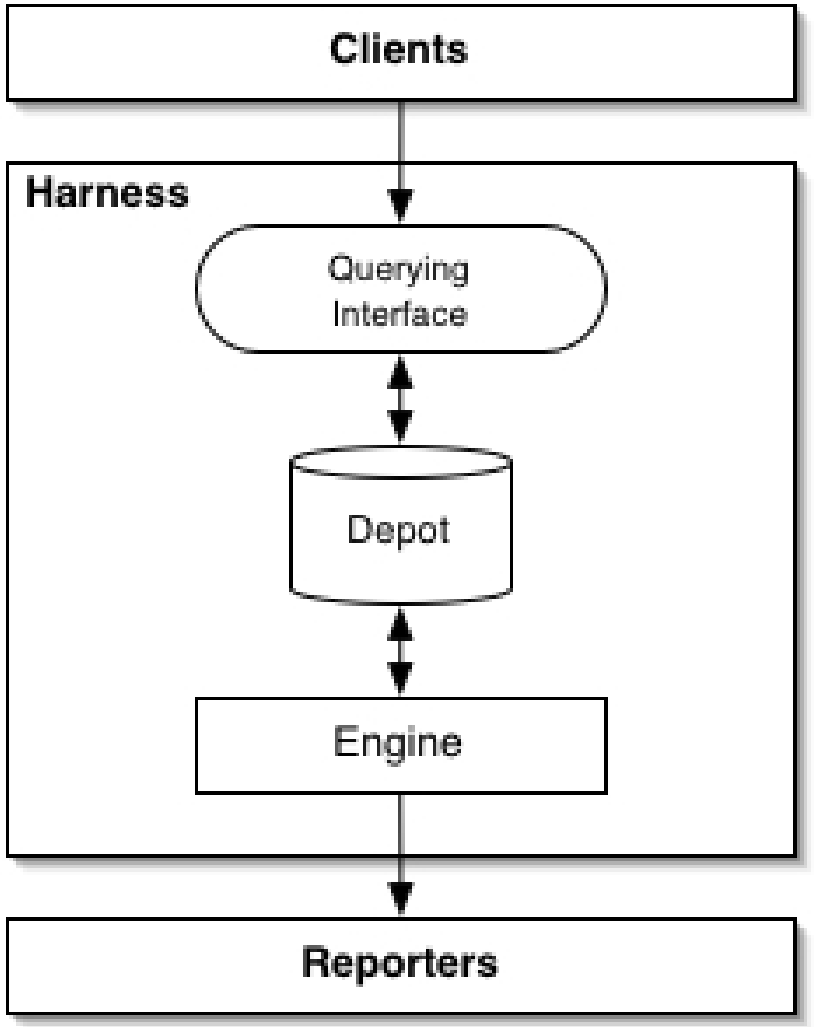
- We have examined the "big picture"
- Many ways to implement material presented thus far
- Inca is a work in progress
- Many encouraging early successes

More about reporters

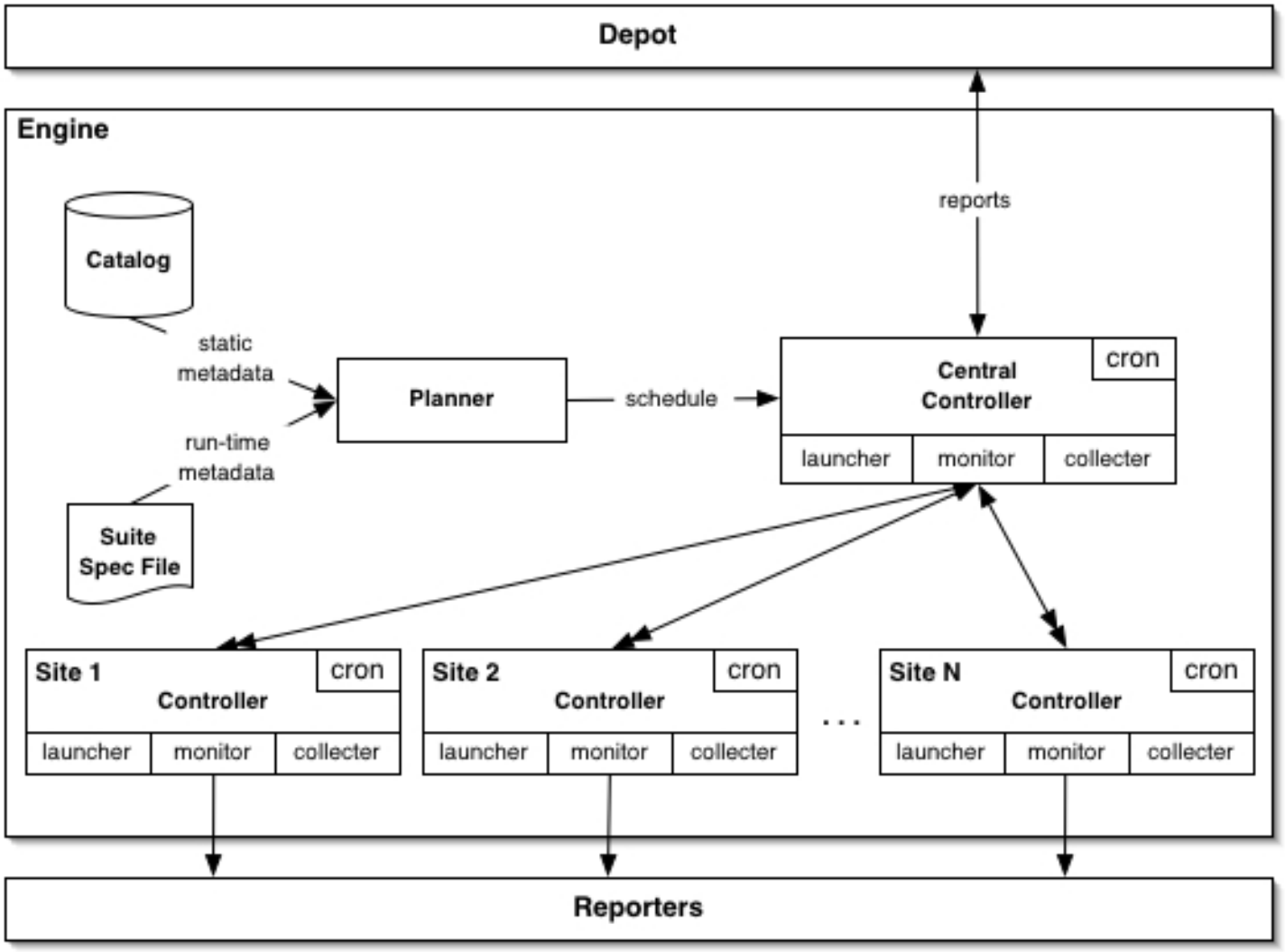
- A reporter suite is an ensemble of reporters plus a catalog reference and spec file
  - A catalog lists available reporters and static attributes (e.g., timeout)
  - A spec file is a run-time description of a reporter suite (mapping, inputs, frequency)
- An aggregate reporter executes a series of reporters and reports an aggregated result
- A reporter can have dependencies on other reporters (data and functional)

Harness

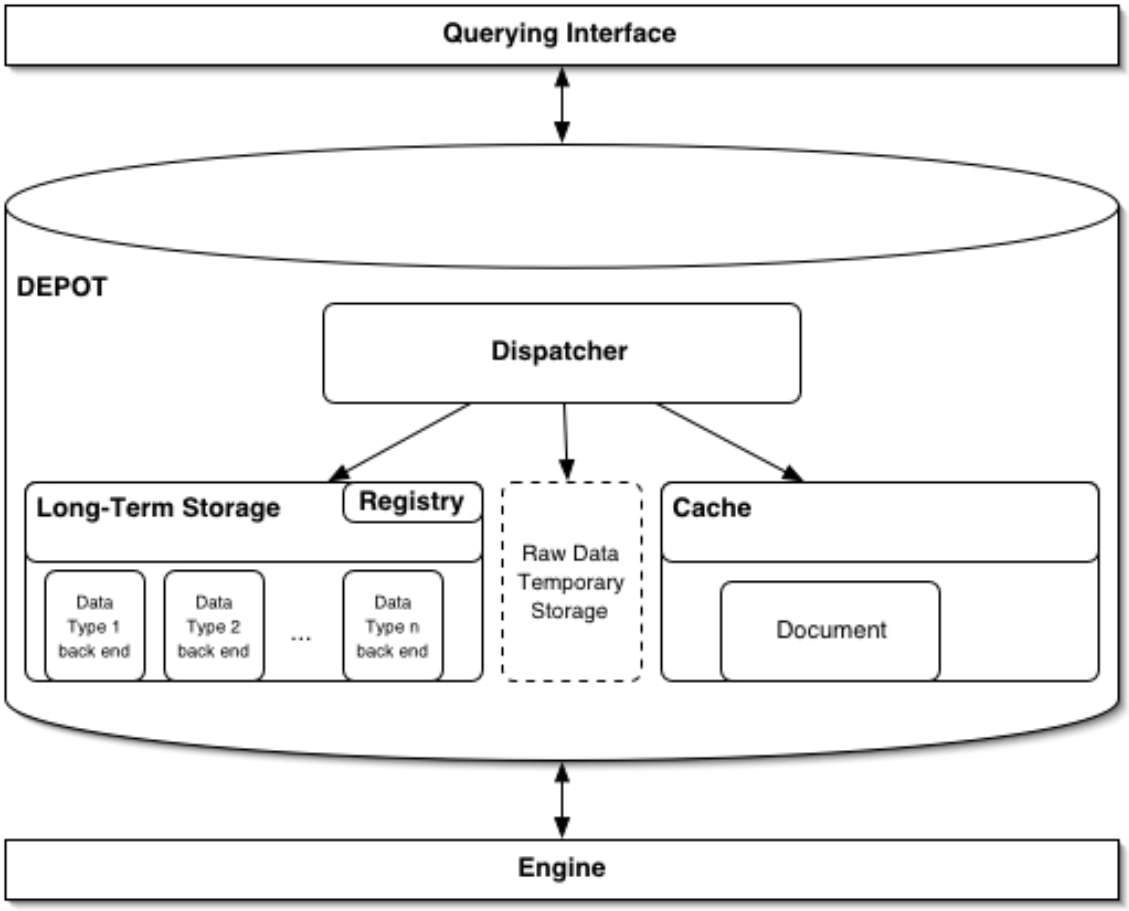
- Engine:
  - Planning and execution of reporter suites
  - Collects output to central location
- Depot: caches and archives output
- Querying Interface
- Modes of operation
  - One-shot
  - Monitoring



Harness Engine



Depot


Querying Interface
DEPOT
Dispatcher
Long-Term Storage
Registry
Data Type 1 back end
Data Type 2 back end
...
Data Type n back end
Raw Data Temporary Storage
Cache
Document
Engine

Depot - Dispatcher

- Implemented as Java Web Service
- Has 3 public functions
  1. Init - registers a branch id with a archiving policy
  2. uploadReport - updates the cache and the archive of the given data
  3. Query - accepts an xml query and sends it to appropriate place (cache or archive)

Depot - Cache

- Implemented as single XML Document
- Location of data determined by branch id
- Holds the last reported data for all reporters - with timestamp

Depot - Long Term Storage

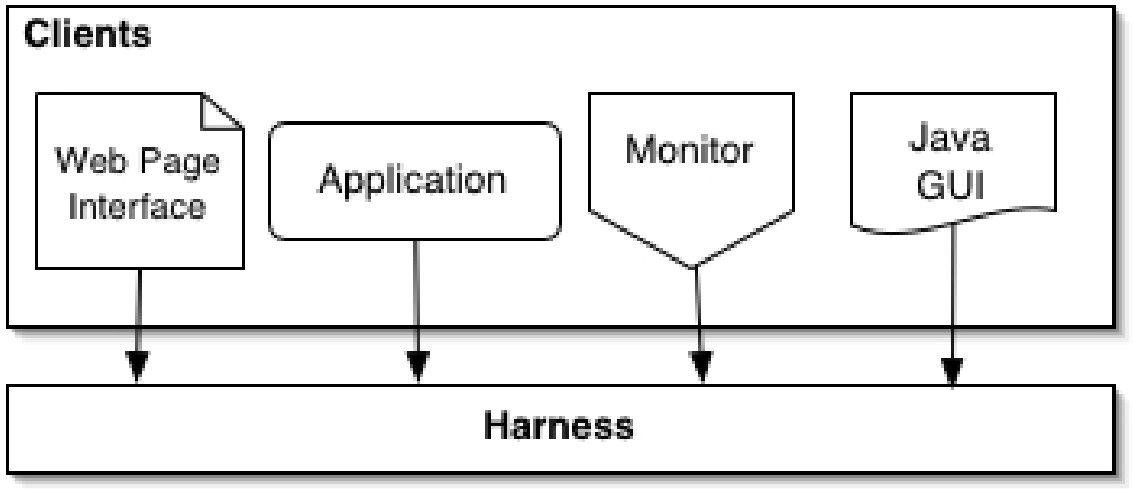
- Currently implemented with RRDTool
- Location of data determined by branch id
- Requires that the branch id be registered by running init

Querying Interface

- Cache
  - Implemented using MDS2
  - MDS2 is built on top of LDAP and so any LDAP API can be used to query the cache
  - xml2ldif converter acts as information provider to MDS2
- Archival
  - SOAP call

Clients

- Reporter
  - Web page interface to reporters
- Depot



Web interface to reporters

- A purpose of API: make web interface easier
  - Normal test output, XML, can be munged into web pages
  - Run test with -help -verbose=1,2 gives description of test in XML
    - Tests are self documenting
    - Can be used to generate web forms
- Built example dynamic forms page from help output
  - http://repo.teragrid.org/cgi-inca/cgi-bin/newdir.cgi

Web interface to reporters

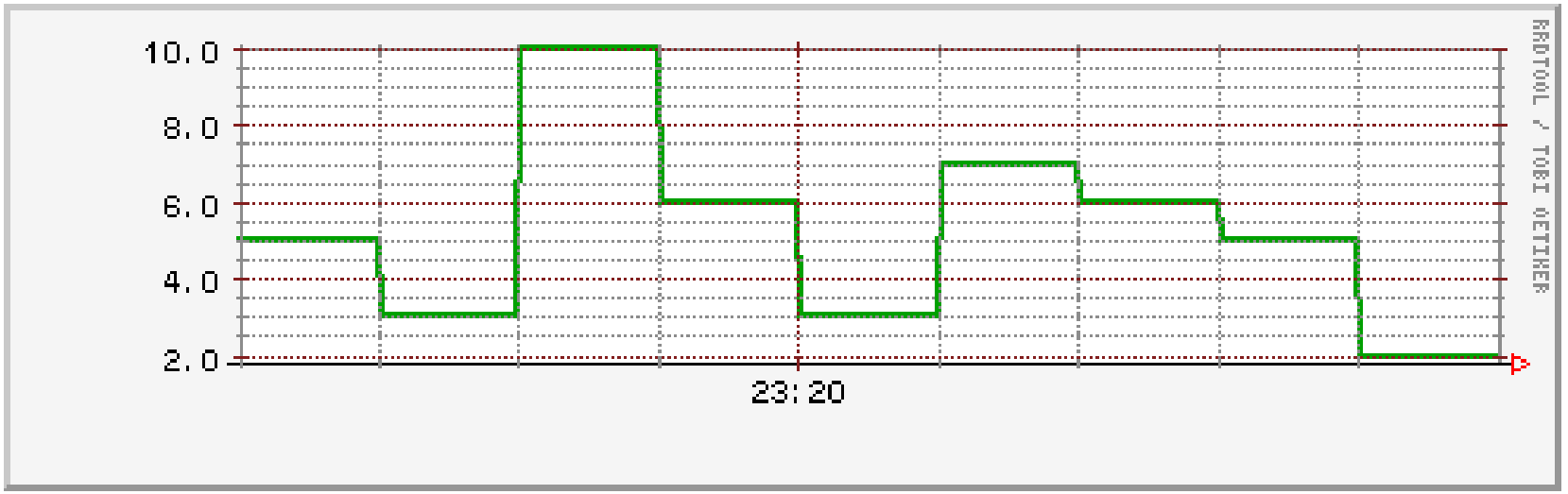
- Repo.teragrid.org/cgi-inca/cgi-bin/newdir.cgi
  - Top level looks like a directory listing of all tests
  - Serve as a repository for tests
  - Click on a "file" and a form is made from test XML
- Form can
  - Create test
  - Command line
  - Get help
- Form will
  - Run tests
  - Combine tests

Web interface to depot cache

- Display cached data in a user-friendly manner
  - Package version information
  - Package unit test results
- Demo

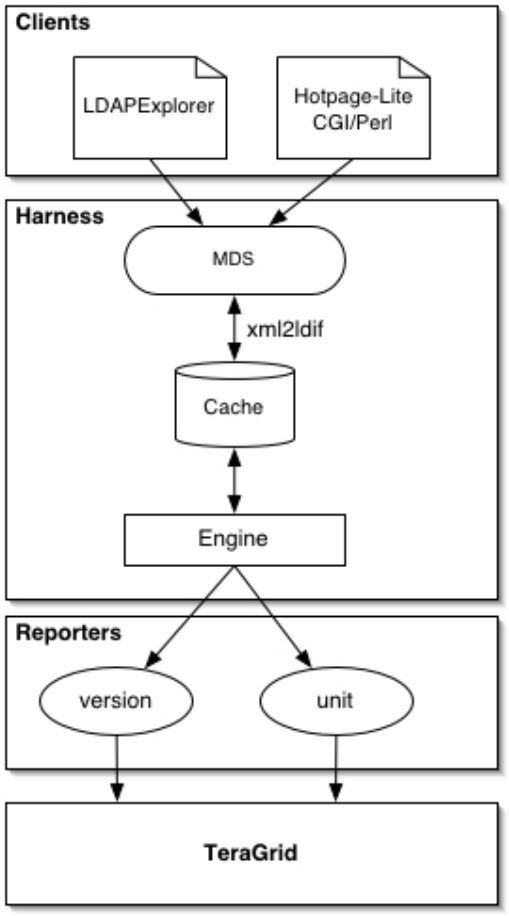
Web interface to depot archive

- Generate graphs from RRDTool on the fly



Inca Test Harness Status

- Reporter
  - Helper APIs in Perl (and soon Python)
  - Version and unit reporters from grid, cluster, and perf eval groups
- Harness
  - Running since March 17 at SDSC and NCSA
  - Scheduled execution of test suites
  - Data centrally collected, cached, and published into MDS
- Client
  - Perl driven "Hotpage" web interface
    - Displays unit and version data
  - LDAPBrowser for raw data



Summary

- Inca is new software built to create the TG Grid Hosting Environment
- Test Harness and Reporting Framework addresses testing, verification, and monitoring
  - Stack Certification, Deployment verification
    - Harness engine running in one-shot mode
  - Monitoring, Benchmarking
    - Harness engine running in monitoring mode
    - Web interface to view collected data and analysis
  - User-level verification
    - Web interface to reporters
- Also beneficial to other Grid efforts as well

Acknowledgements

- Funding
  - NSF Cooperative Agreement No. ACI-0122272 titled "Collaborative Research: The TeraGrid: Cyberinfrastructure for 21st Century Science and Engineering"
- Developers
  - Shava Smallen (SDSC)- project lead
  - Cathie Mills (SDSC)
  - Brian Finley (ANL)
  - Tim Kaiser (SDSC)
  - many others ...
- Caltech-CACR
  - Performance Evaluation – Sharon Burnett
  - Applications – Roy Williams
  - Clusters – Jan Lindheim