Data Access and Analysis with Distributed, Federated Data Servers in climate*prediction*.net

Neil Massey[1]

neil.massey@comlab.ox.ac.uk

Tolu Aina[2], Myles Allen[2], Carl Christensen[1], David Frame[2], Daniel Goodman[1], Jamie Kettleborough[3], Andrew Martin[1], Stephen Pascoe[3], David Stainforth[2]

1: Computing Laboratory, Oxford University
2: Department of Physics, Oxford University
3: Rutherford Appleton Laboratory

EGU 2005, 24th to 29th April, Vienna.

climate*prediction*.net

Oxford University

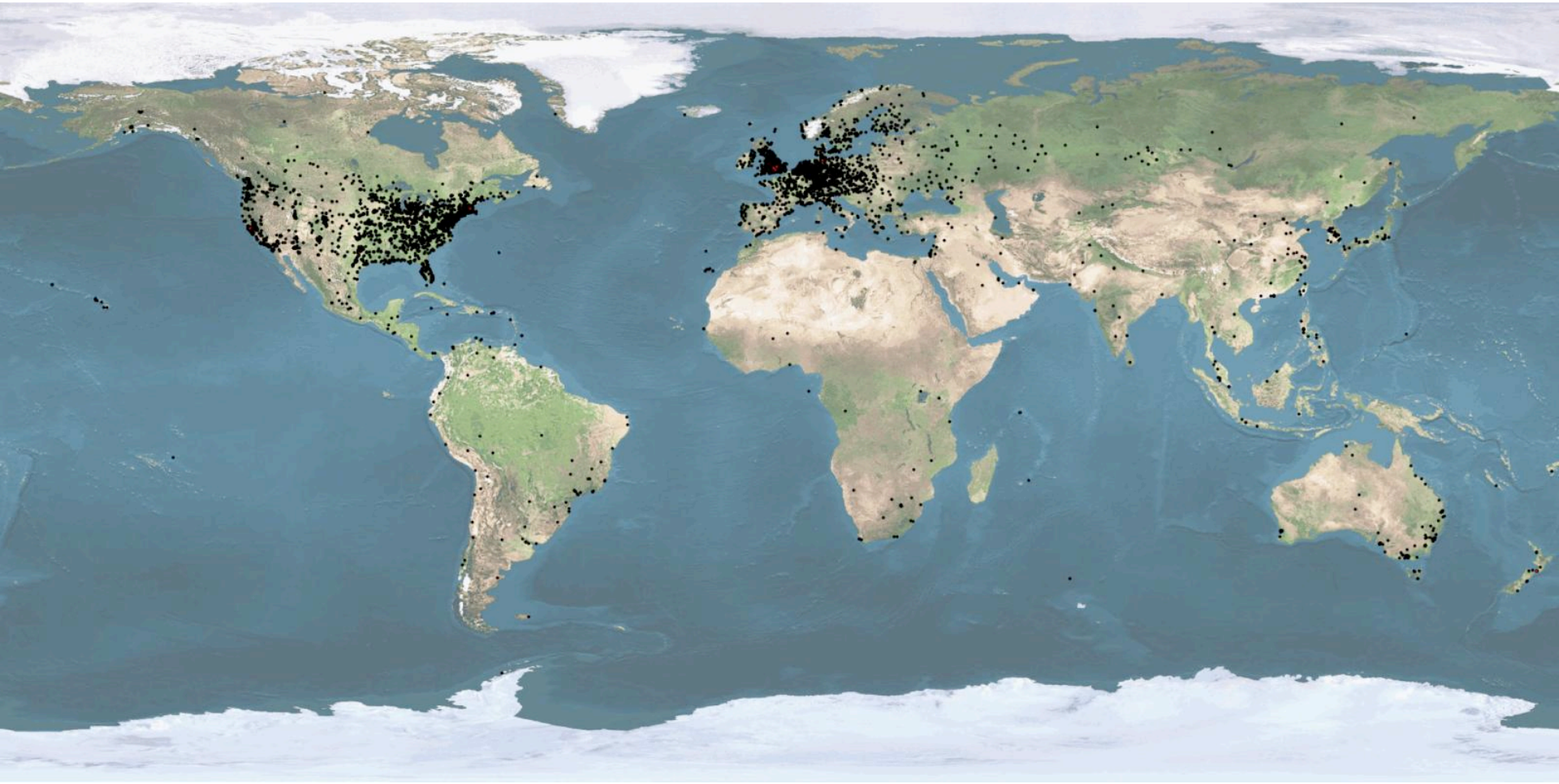
Contents

- Overview of project
- Analysing data over distributed data nodes
- Using web services to access and analyse data
- Notification of results in an asynchronous workflow
- Using web services to wrapper legacy code
- Presenting interfaces to data

climateprediction.net

- The Goals
  - Harness the power of idle PCs to help quantify uncertainty in predictions of the 21st century climate
  - To improve public understanding of the nature of uncertainty in climate prediction
- The Method
  - Invite the public to download a full resolution, 3D climate model and run it locally on each PC
  - Use each PC to run a single member of a massive, perturbed physics ensemble
  - Provide visualisation software and educational packages to maintain interest and facilitate school and undergraduate projects etc.

Distribution of client users

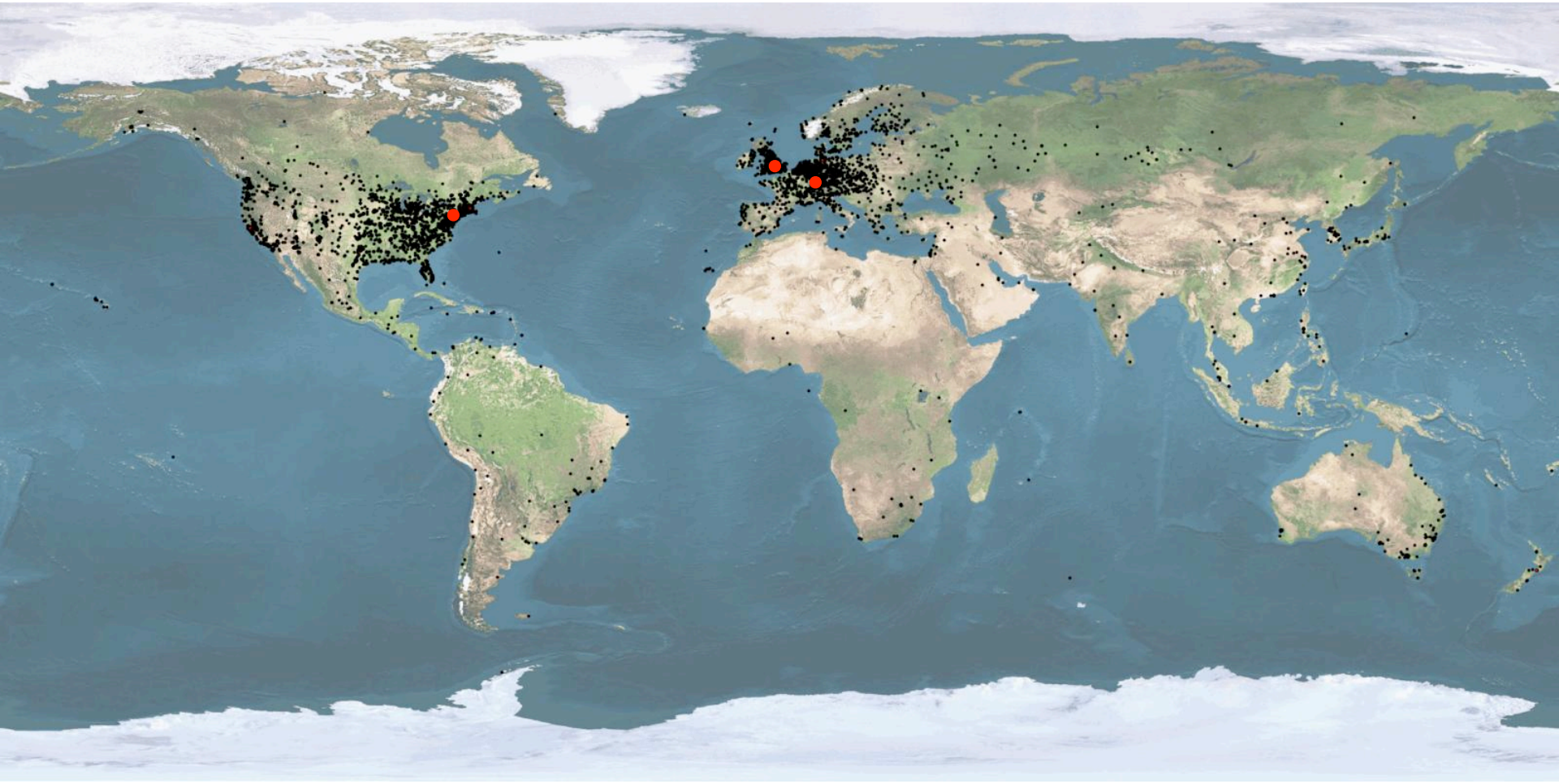


~100,000 volunteers, 130 countries

climateprediction.net

Oxford University

Distribution of data nodes



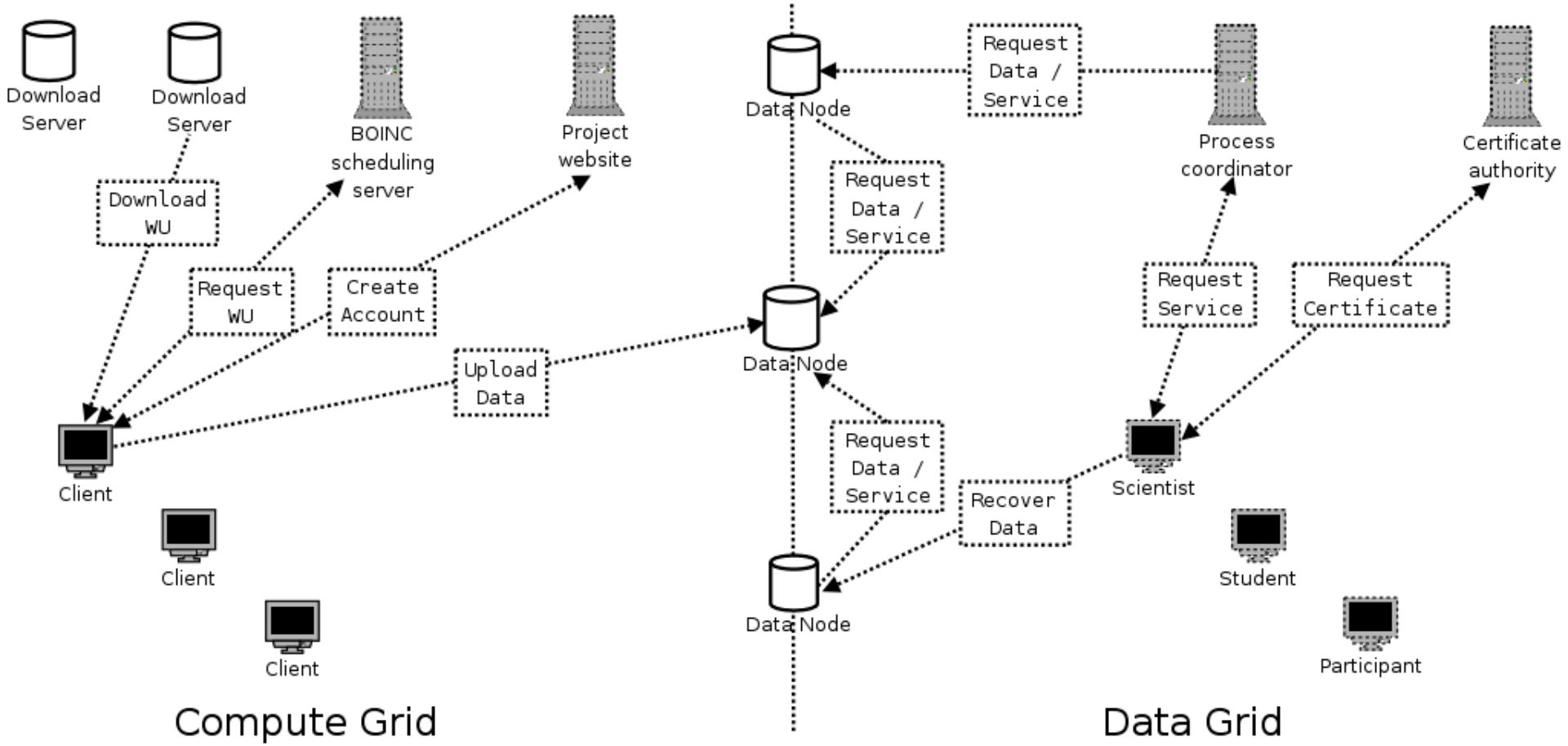
~100,000 volunteers, 130 countries

5 data nodes

climate*prediction*.net

Oxford University

Network



climateprediction.net

Oxford University

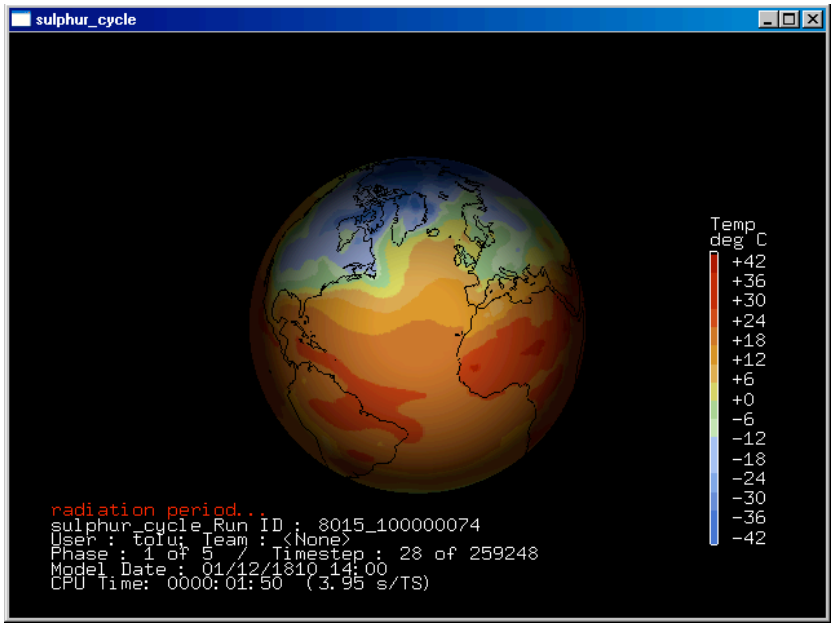
The client

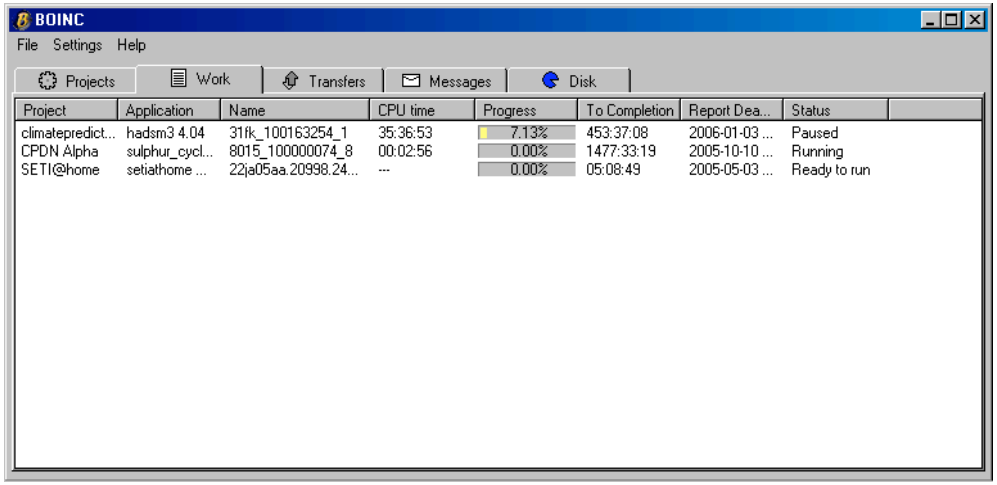
- Public donated computing time running under the BOINC infrastructure
- Clients download a full GCM, integrate for 45 years, ~2 to 6 weeks
- Clients can track progess with a visualisation program
- Produce ~600MB of data of which ~12MB is uploaded directly to a data node
- Also upload meta-data to a central database

climate*prediction*.net

Oxford University

The client





climate*prediction*.net

Oxford University

Data nodes

- Rely on donated server space
- Data is federated across the nodes
- Do not want end users to FTP the raw data
- Instead:
  - Provide a secure, robust, efficient, scalable environment for data discovery and analysis
  - Transparently provide access to the meta-data and data simultaneously
  - Allow multiple interfaces to access the data

Distributed analysis of data

- Design problems / goals:
  - Data set is federated across data nodes
  - Only one copy of the data set
  - Data set is large
  - Servers are donated, potentially no root access
  - Analysis are computationally expensive, may take several days
- Design decisions:
  - Copy data set instead of moving
  - Just in time data access
  - Distributed analysis
  - Service oriented architecture
  - Asynchronous service request, delivery
  - Web services for infrastructure

Web services to analyse data

- Why web services?
  - Lightweight way to build grid like infrastructure
  - Open, standardised protocols
  - XML RPC without firewall problems
  - Security capabilities present in the software stack
  - Lots of industry input (Microsoft, IBM, Sun, etc.)
  - Momentum in the UK academic community (WSRF, OMII)
- Technology
  - Java Virtual Machine
  - Apache HTTP server, Apache Tomcat, Apache Axis
  - ~30MB install, not including JVM

Composing analysis

- Decompose analysis into component parts
  - Example: computing normalised differences
    - » Collect data as URIs by querying database
    - » Compute the difference
    - » Normalise the difference
- Present an API to scientists to allow the composition
  - Popular components as “modules”
  - Method to chain them together
- Incorporate data and meta-data
- Allows analysis to be distributed
- Must move and copy data between the parts

Moving and copying data

- Data too large to move around as SOAP message
- Move data via URI reference to data
- Move data only when it is needed
- Analysis will produce intermediate results
- Store these in a transitory manner
- Copy persistent data but move intermediate data
- Present results to scientists in the same transitory store
  - Garbage collection
  - When to collect
  - How big a cache?

Notification of results

- Results could take a long time to generate
- Architecture is asynchronous
- Several options for notifying that computation is complete:
  - Email
  - Dynamically changing web site
  - RSS feed
- Results are then available via FTP web service

Wrappering legacy code in web services

- Many legacy applications that can be used for analysis
- Too much effort to re-implement them in WS
- Most applications can be wrappered
  - C++, FORTRAN, Python, Perl, even command line tools
  - SOAP libraries exist for many languages
  - However, full WS stack does not
  - This includes security
  - And WSDL generation
- Example: using a CDAT function as a web service

Wrappering legacy code: localhost web service

- If SOAP libraries exist for the legacy language:
  - Write a simple Java interface to the function
  - Write the actual function in the legacy language, wrappering it within SOAP function calls
  - The Java interface, calls the legacy function using SOAP, over localhost and using a designated port
  - Secure port using iptables
- Advantages:
  - Easy to implement
  - Fits web services model
- Disadvantages:
  - Have to dump the data to the file system
  - Have to write serializers in both languages
  - Cannot use with command line tools

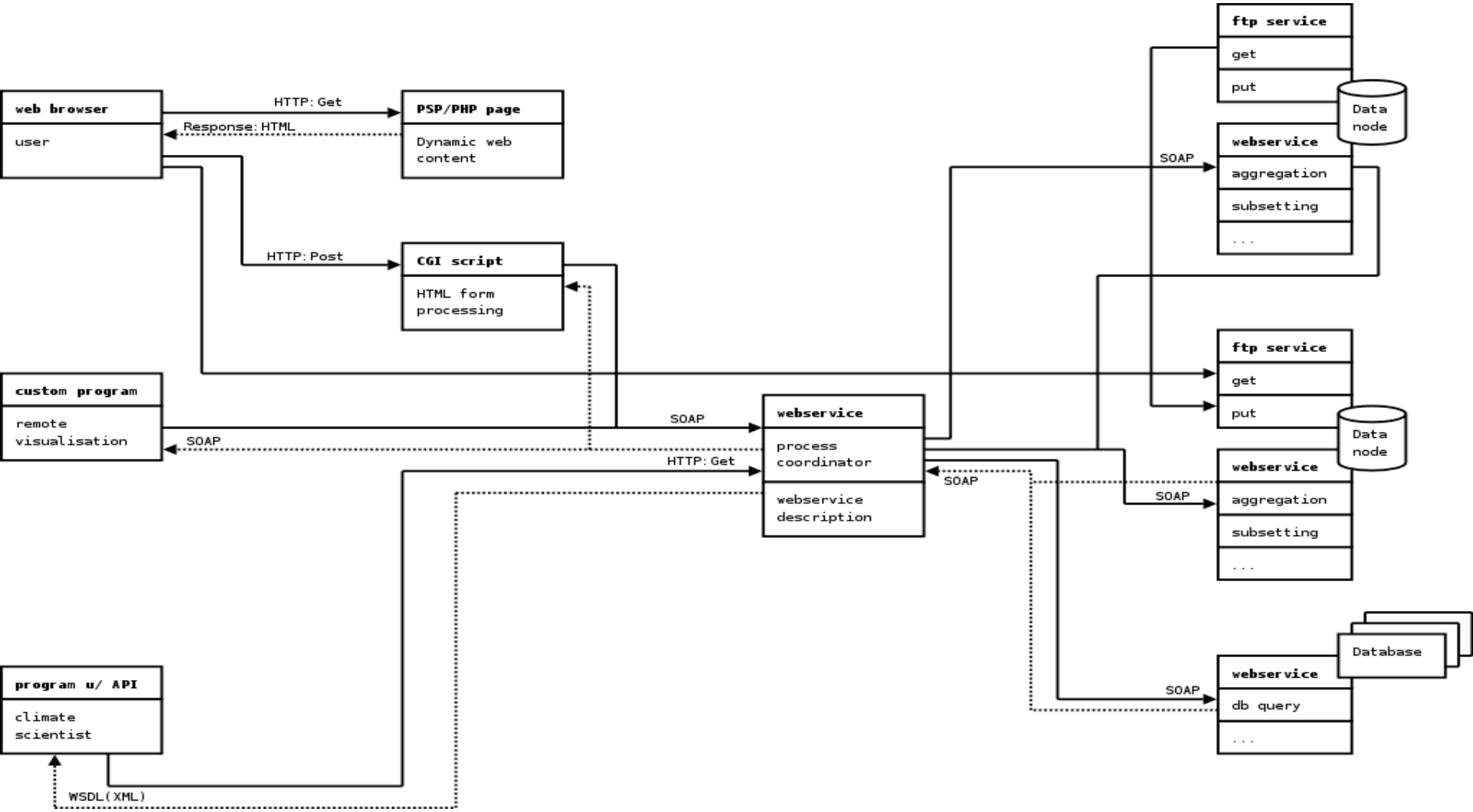
Wrappering legacy code: Java Native Interface

- JNI allows running of native code
  - To run a Python program:
    - » Embed the Python interpreter in a C program
    - » Import and run the Python from the embedded interpreter
    - » Write a Java wrapper
    - » Use JNI to generate the “glue”
- Advantages:
  - Don’t need to dump data: can just stream or pass a pointer!
  - Can run command line tool
  - Can use “externals”
- Disadvantages:
  - Tricky to get right
  - Have to ensure data is in compatible format for each language

Connecting Interfaces

- Different interfaces for different levels of users
  - Scientists
  - School children
  - Custom software
- Write the web services first:
  - Avoids replication
  - Almost anything can connect to them (Java, Python, C++, PHP 5, Perl, etc.)
  - Increases maintainability: one common API
  - Increases flexibility of web service API

Connecting interfaces



climateprediction.net

Oxford University

Connecting Interfaces – Example

- Simple web portal to query parameter combinations from the database
  <url masked for web presentation>
- Workflow:
  - User connects to portal with browser.

    Python server page generates dynamic content
  - User selects parameters, clicks “submit query”

    Form data is sent via HTTP POST to CGI Python CGI script
    CGI script processes POST data, creates webservice call
    Web service does processing and returns SOAP to the CGI script
    CGI script caches results and creates an ID for the cache
    CGI script calls another PSP page with returned data
    PSP script presents data to the user as HTML
  - User selects “List Runs”

    HTML generated by PSP calls another PSP script with cache ID
    PSP generates HTML to present to user

References

- **climateprediction.net:** *http://www.climateprediction.net*
- **Public Computing: Reconnecting People to Science:**
  Dr. David P. Anderson, *http://boinc.berkeley.edu/madrid.html*
- **BOINC: A System for Public-Resource Computing and Storage**
  Dr. David P. Anderson, *http://boinc.berkeley.edu/grid_paper_04.pdf*
- **OMII:** *http://www.omii.ac.uk/mission.htm*
- **Java Native Interface:** *http://java.sun.com/docs/books/tutorial/native1.1/*
- **Python** *http://www.python.org*

- neil.massey@comlab.ox.ac.uk