Automatic Tool Generation for Ad Hoc Data Processing

David Walker (PI)

Kathleen Fisher

Yitzhak Mandelbaum, Vivek Pai (Co-PI)

Princeton University & AT&T Research

Data, Data Everywhere

- Software systems of all kinds generate vast amounts of data -- often in *ad hoc* formats.
- Ad hoc data is mostly structured:
  - Not free text.
  - But not held in standard relational database.
  - Not necessary standardized (ie, not XML, CSV)
    - More non-standard legacy data formats than anything else?

Ad Hoc Data from Web Server Logs (CLF)

207.136.97.49 - - [15/Oct/1997:18:46:51 -0700] "GET /tk/p.txt HTTP/1.0" 200 30

199.202.84.61 - - [16/Oct/1997:14:32:22 -0700] "POST /scpt/conf HTTP/1.0" 200 941

234.200.68.71 - - [15/Oct/1997:18:53:33 -0700] "GET /tr/gift.gif HTTP/1.0” 200 409

240.142.174.15 - - [15/Oct/1997:18:39:25 -0700] "GET /tr/wool.gif HTTP/1.0" 404 178

188.168.121.58 - - [16/Oct/1997:12:59:35 -0700] "GET / HTTP/1.0" 200 3082

214.201.210.19 ekf - [17/Oct/1997:10:08:23 -0700] "GET /new.gif HTTP/1.0" 304 -

Ad Hoc Data: DNS packets

```
00000000: 9192 d8fb 8480 0001 05d8 0000 0000 0872  ...............r
00000010: 6573 6561 7263 6803 6174 7403 636f 6d00  esearch.att.com.
00000020: 00fc 0001 c00c 0006 0001 0000 0e10 0027  ...............'
00000030: 036e 7331 c00c 0a68 6f73 746d 6173 7465  .ns1...hostmaste
00000040: 72c0 0c77 64e5 4900 000e 1000 0003 8400  r..wd.I.........
00000050: 36ee 8000 000e 10c0 0c00 0f00 0100 000e  6...............
00000060: 1000 0a00 0a05 6c69 6e75 78c0 0cc0 0c00  ......linux.....
00000070: 0f00 0100 000e 1000 0c00 0a07 6d61 696c  ............mail
00000080: 6d61 6ec0 0cc0 0c00 0100 0100 000e 1000  man.............
00000090: 0487 cf1a 16c0 0c00 0200 0100 000e 1000  ................
000000a0: 0603 6e73 30c0 0cc0 0c00 0200 0100 000e  ..ns0...........
000000b0: 1000 02c0 2e03 5f67 63c0 0c00 2100 0100  ......_gc...!...
000000c0: 0002 5800 1d00 0000 640c c404 7068 7973  ..X.....d...phys
000000d0: 0872 6573 6561 7263 6803 6174 7403 636f  .research.att.co
```

Ad Hoc Data in Biology

```
format-version: 1.0
date: 11:11:2005 14:24
auto-generated-by: DAG-Edit 1.419 rev 3
default-namespace: gene_ontology
subsetdef: goslim_goa "GOA and proteome slim"

[Term]
id: GO:0000001
name: mitochondrion inheritance
namespace: biological_process
def: "The distribution of mitochondria\, including the mitochondrial genome\, into
daughter cells after mitosis or meiosis\, mediated by interactions between
mitochondria and the cytoskeleton." [PMID:10873824,PMID:11389764, SGD:mcc]
is_a: GO:0048308 ! organelle inheritance
is_a: GO:0048311 ! mitochondrion distribution
```

www.geneontology.org

Ad Hoc Data in Finance

```
HA00000000START OF TEST CYCLE
aA00000001BXYZ U1AB00000400000100B0000004200
HL00000002START OF OPEN INTEREST
d 00000003FZYX G1AB000003000300000
HM00000004END OF OPEN INTEREST
HE00000005START OF SUMMARY
f 00000006NYZX
B1QB000520001200000700000B000050000000520000
  00490000005100+00000100B000000053000000525000000535000
HF00000007END OF SUMMARY
```

www.opradata.com

Properties of Ad hoc Data

- Data arrives "as is" -- you don't choose the format
- Documentation is often out-of-date or nonexistent.
- Data is buggy.
  - Missing data, "extra" data, …
  - Human error, malfunctioning machines, software bugs …
  - Errors are sometimes the *most* interesting portion of the data.
- Data sources often have high volume.
- Data can be created by malicious sources attempting to exploit software vulnerabilities
  - c.f. Ethereal network monitoring system

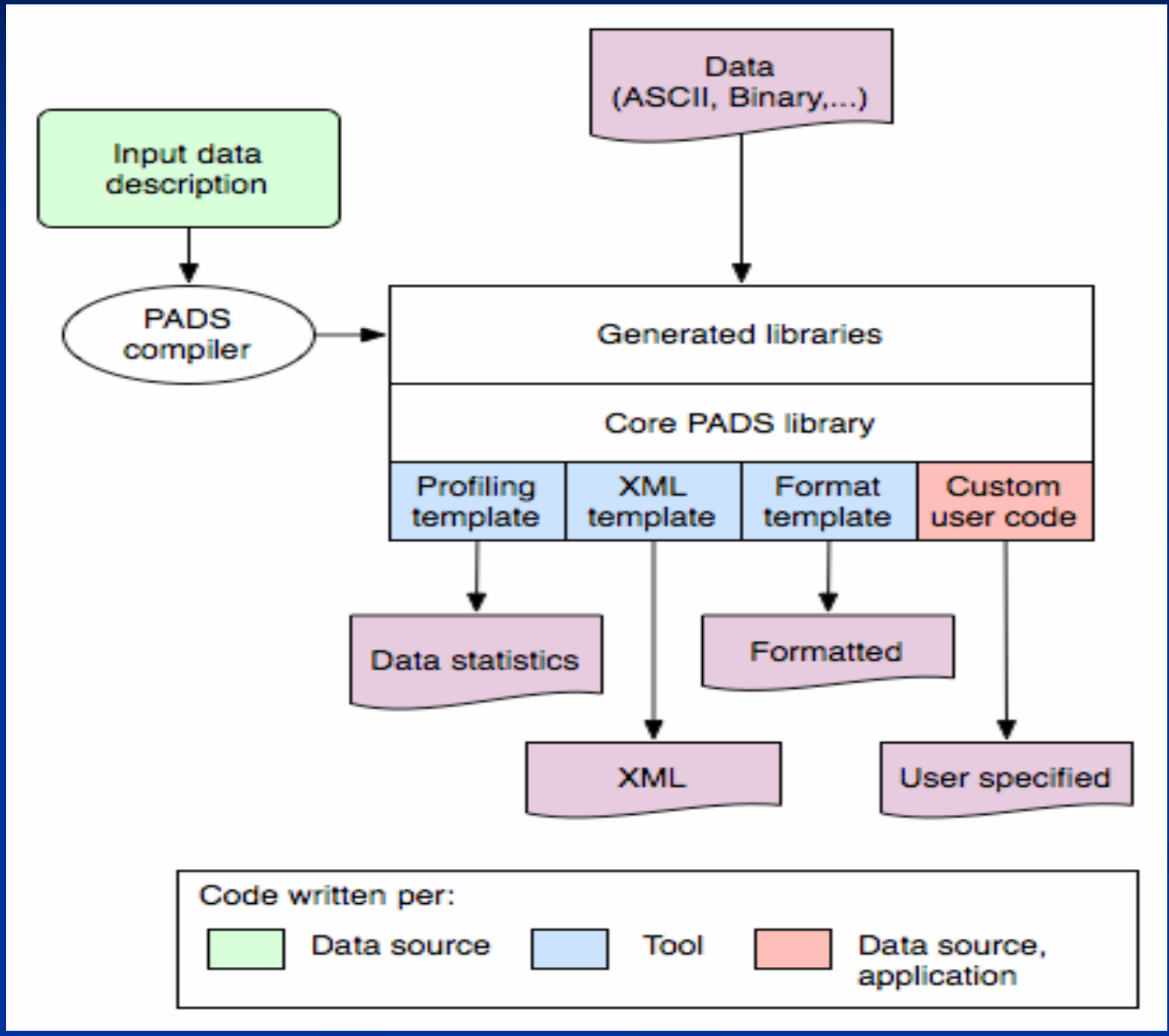
Project Goals

- Automated generation of tools
  - for continuous, real-time monitoring and historical access to widely disparate kinds of data
- Widely applicable
  - Operate over data in any legacy format
- Simple adoption and ease-of-use
  - Minimize programming effort
- Robust in the face of errors
  - And provide capabilities/programming model for diagnosing and processing errors

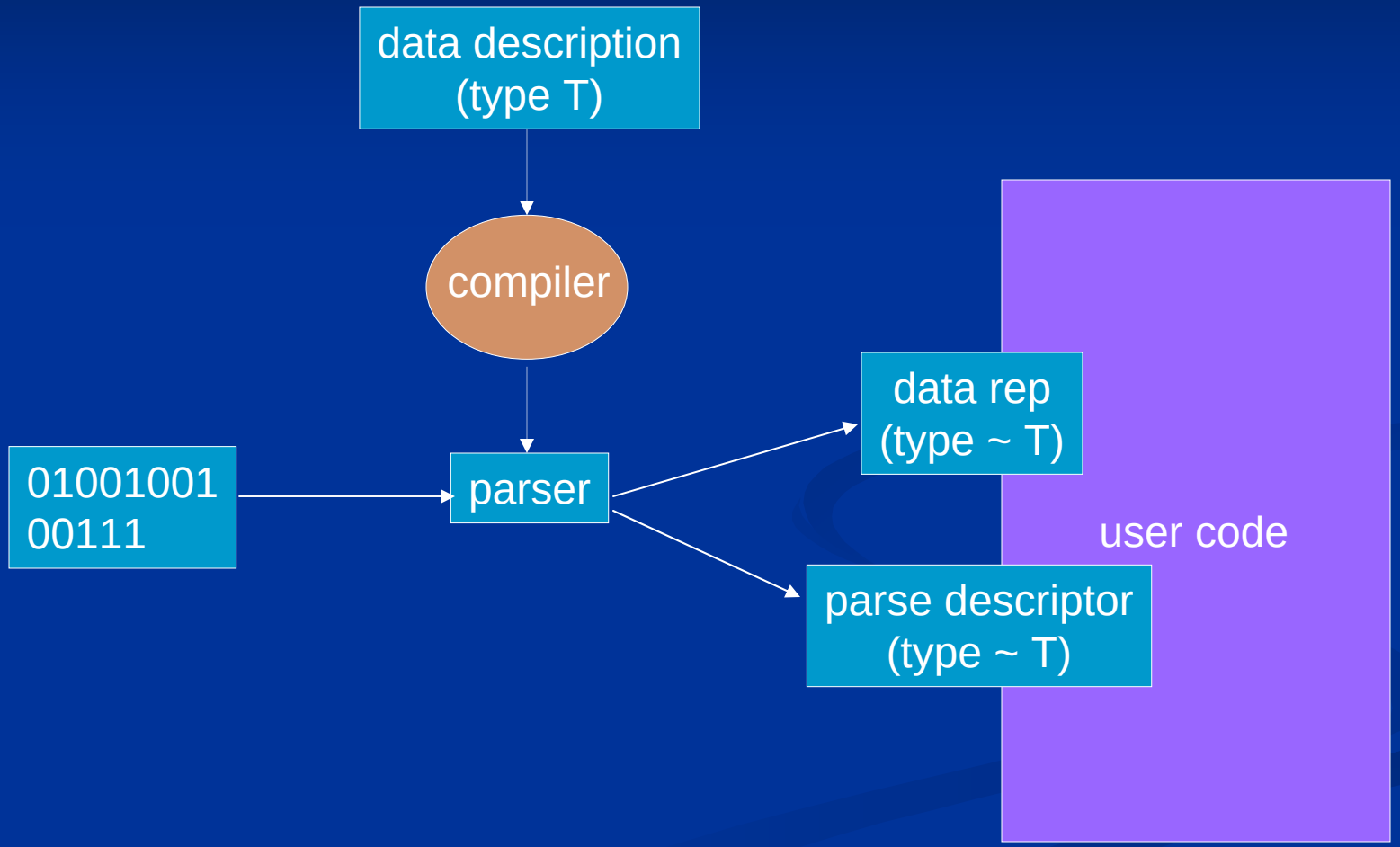
Approach: A Domain-Specific Programming Language

- PADS/ML Language & Compiler System
  - Declarative "data description language"
  - Supports ASCII, binary & mixed ad hoc data formats
  - Automatically generates a variety of programming libraries & stand-alone tools
  - Formal denotational semantics for reasoning about specification properties

PADS Compiler System



Pads Parsing



PADS Language

Data Descriptions look (largely) like a series of type definitions in your favourite programming language

- Rich and extensible set of base types:
  - `Pint8`, `Pint32`, …
  - `Pstring('|'), PstringFW(x)`
  - `Pdate, Ptime, Pip, …`
- Dependent type constructors:
  - `(Dependent) records`
  - `(Dependent) ML-style datatypes`
    - `(ie: data-dependent recursive unions of records)`
  - `Lists`
  - `Parameterization:  By description & value`
  - `Arbitrary (Data Dependent) Constraints`

An Example Type Definition

```
ptype weblog = {
        host client;                    /- Client requesting service
  ' ';  auth_id remoteID;               /- Remote identity
  ' ';  auth_id auth;                   /- Name of authenticated user
  " [";  Pdate(:']':) date;             /- Timestamp of request
  "] ";  http_request request;          /- Request
  ' ';  Puint16_FW(:3:) response;       /- 3-digit response code
  ' ';  Puint32 contentLength;          /- Bytes in response
};
```

207.136.97.50 - - [15/Oct/1997] "GET /turkey.gif HTTP/1.0" 200 3013

PADS/ML Regulus Format:

```
ptype Timestamp =
  Ptimestamp_explicit_FW(8, "%H:%M:%S", gmt)

ptype Pip =
  Puint8 * '.' * Puint8 * '.' * Puint8 * '.' * Puint8

ptype (Alpha) Pnvp(p : string -> bool) =
  { name : [name : Pstring('=') | p name]; '=';
    value : Alpha }

ptype (Alpha) Nvp(name:string) =
  Alpha Pnvp(fun s -> s = name)

ptype SVString = Pstring_SE("/;|\\|/")

ptype Nvp_a = SVString Pnvp(fun _ -> true)

ptype Details = {
    source      : Pip Nvp("src_addr");
';'; dest       : Pip Nvp("dest_addr");
';'; start_time : Timestamp Nvp("start_time");
';'; end_time   : Timestamp Nvp("end_time");
';'; cycle_time: Puint32 Nvp("cycle_time")
}
```

```
ptype Semicolon = Pcharlit(';')
ptype Vbar = Pcharlit('|')

pdatatype Info(alarm_code : int) =
  Pmatch alarm_code with
    5074 -> Details of Details
  | _    -> Generic of (Nvp_a,Semicolon,Vbar) Plist

pdatatype Service =
    Dom of "DOMESTIC"
  | Int of "INTERNATIONAL"
  | Spec of "SPECIAL"

ptype Raw_alarm = {
    alarm       : [ i : Puint32 | i = 2 or i = 3];
':'; start      : Timestamp Popt;
'|'; clear      : Timestamp Popt;
'|'; code       : Puint32;
'|'; src_dns    : SVString Nvp("dns1");
';'; dest_dns   : SVString Nvp("dns2");
'|'; info       : Info(code);
'|'; service    : Service
}

let checkCorr ra = ...

ptype Alarm = [x:Raw_alarm | checkCorr x]
ptype Source = (Alarm,Peor,Peof) Plist
```

Sample Regulus Data:

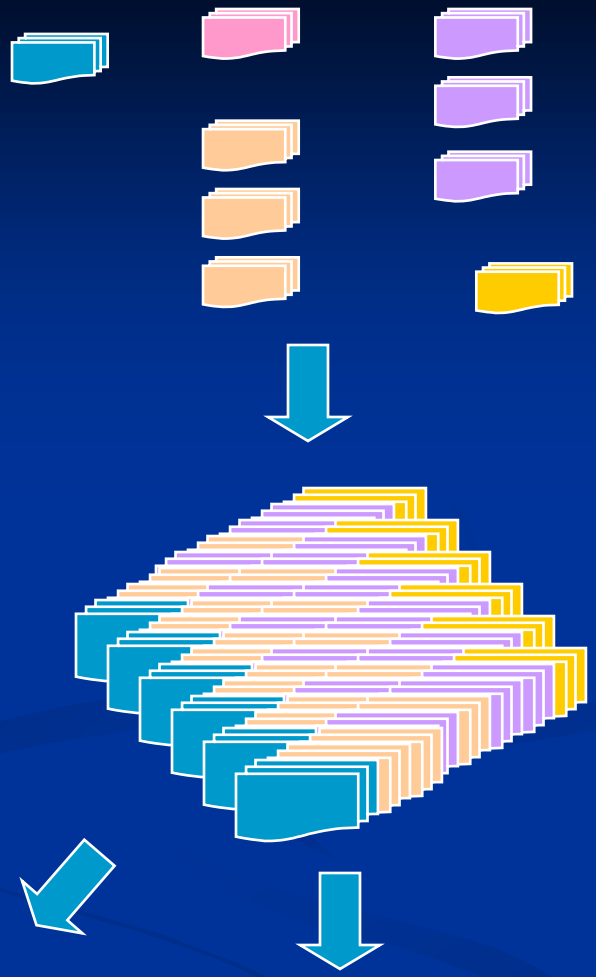
```
2:3004092508||5001|dns1=abc.com;dns2=xyz.com|c=slow link;w=lost packets|INTERNATIONAL
3:|3004097201|5074|dns1=bob.com;dns2=alice.com|src_addr=192.168.0.10;
dst_addr=192.168.23.10;start_time=1234567890;end_time=1234568000;cycle_time=17412|SPECIAL
```

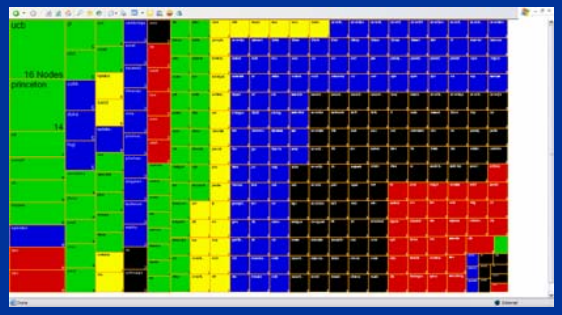
Advantages Over "Ad Hoc" Methods

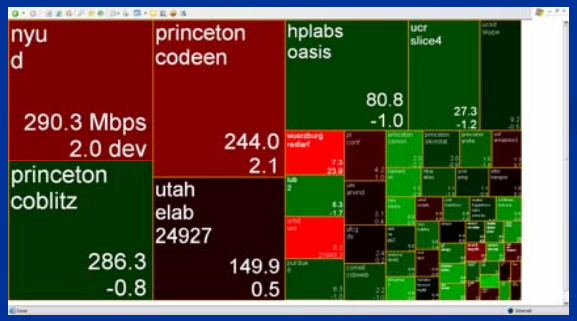
- Big bang for user buck:
  - 1 description ==> many tools
  - 1 format-independent tool ==> many data sources
- Descriptions document data sources
  - the documentation IS the tool generator
- Descriptions are easy to write, easy to understand.
- Tools are robust
  - Error handling code generated automatically; doesn't clutter documentation.
- Descriptions have formal semantics

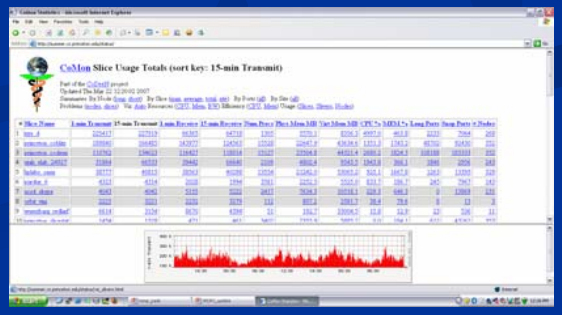
Up Next

- New sorts of specifications for data location, access mode & temporal availability
  - Rearchitect Vivek Pai's CoMon infrastructure to generate monitoring and visualization tools from specs
- Learn descriptions from example data automatically









Progress Summary

[NSF support since summer 06]

- PADS [http://www.padsproj.org]
  - Defined tool generation architecture & formal semantics for PADS/ML [POPL 07, TFP 07]
  - First release of PADS/ML: Jan 07
- CoMon [http://comon.cs.princeton.edu]
  - New visualization tools & backend optimizations
  - Modularization of software infrastructure in preparation for using PADS-generated software components
  - Usage: 700+ ips; 30+ users accessing monitoring and visualization tools at any time

End

Existing Approaches

- Lex/Yacc
  - Over & Underkill.
- Perl/C
  - Code brittle with respect to changes in input format.
  - Analysis often ends up interwoven with parsing.
  - Error code, if written, swamps main-line computation. If not written, errors can corrupt "good" data.
  - Everything has to be coded by hand.
- Packet description languages
  - eg: PacketTypes, Datascript
  - Binary data
  - Focus on correct data.

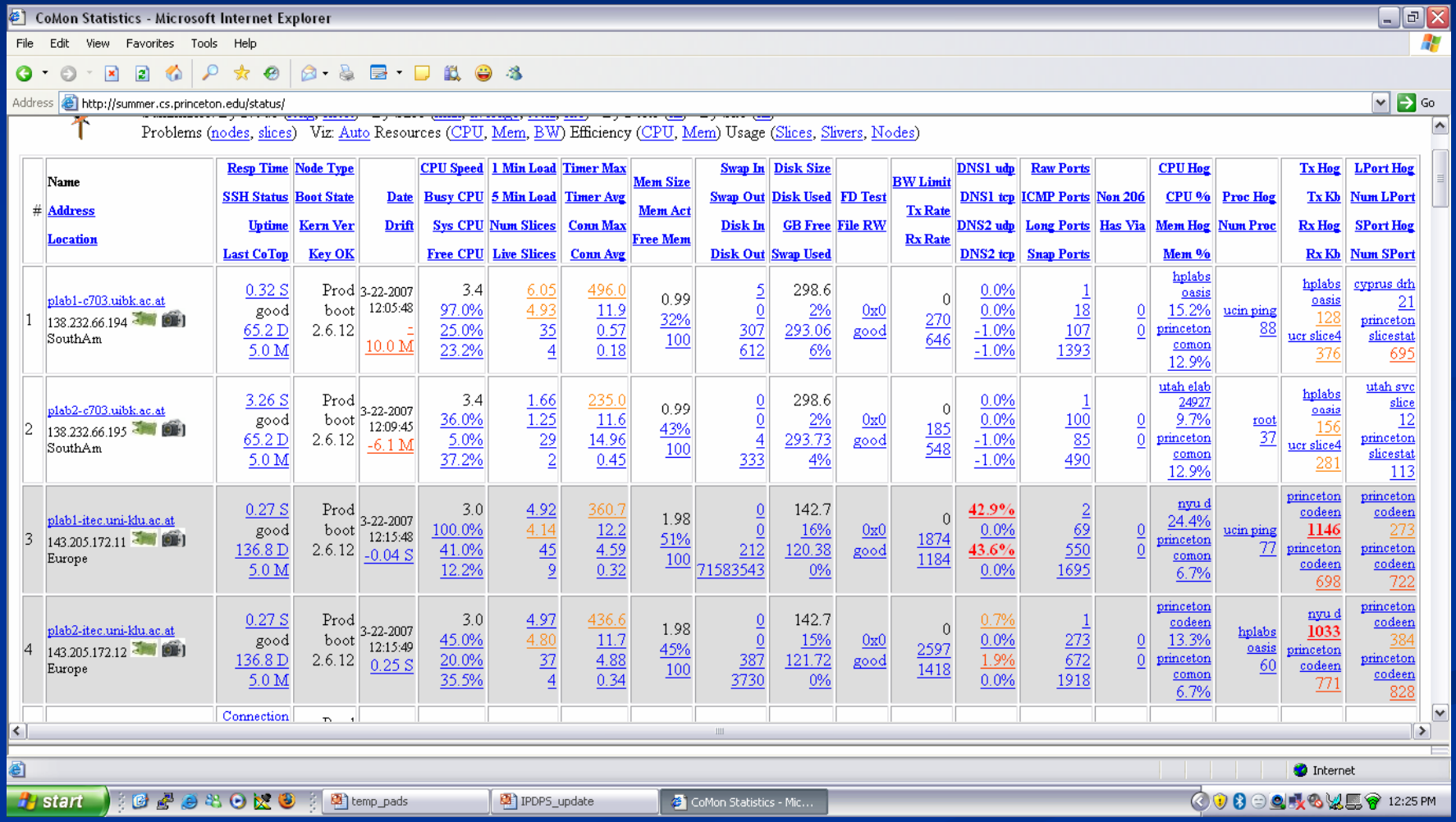
CoMon Monitoring Framework

- End Goal: To generate custom monitoring components from high-level specifications
  - Another whole domain-specific collection of tools
- Current CoMon Status
  - Data Volume
    - 700 nodes, 300 experiments, 10 data points per experiment instance
  - Interactivity
    - 5 minute refresh interval, History stored over two days
    - Continuous & on-demand visualization mechanisms
  - Usage
    - 700 IPs; 30 users accessing visualization tools at any time

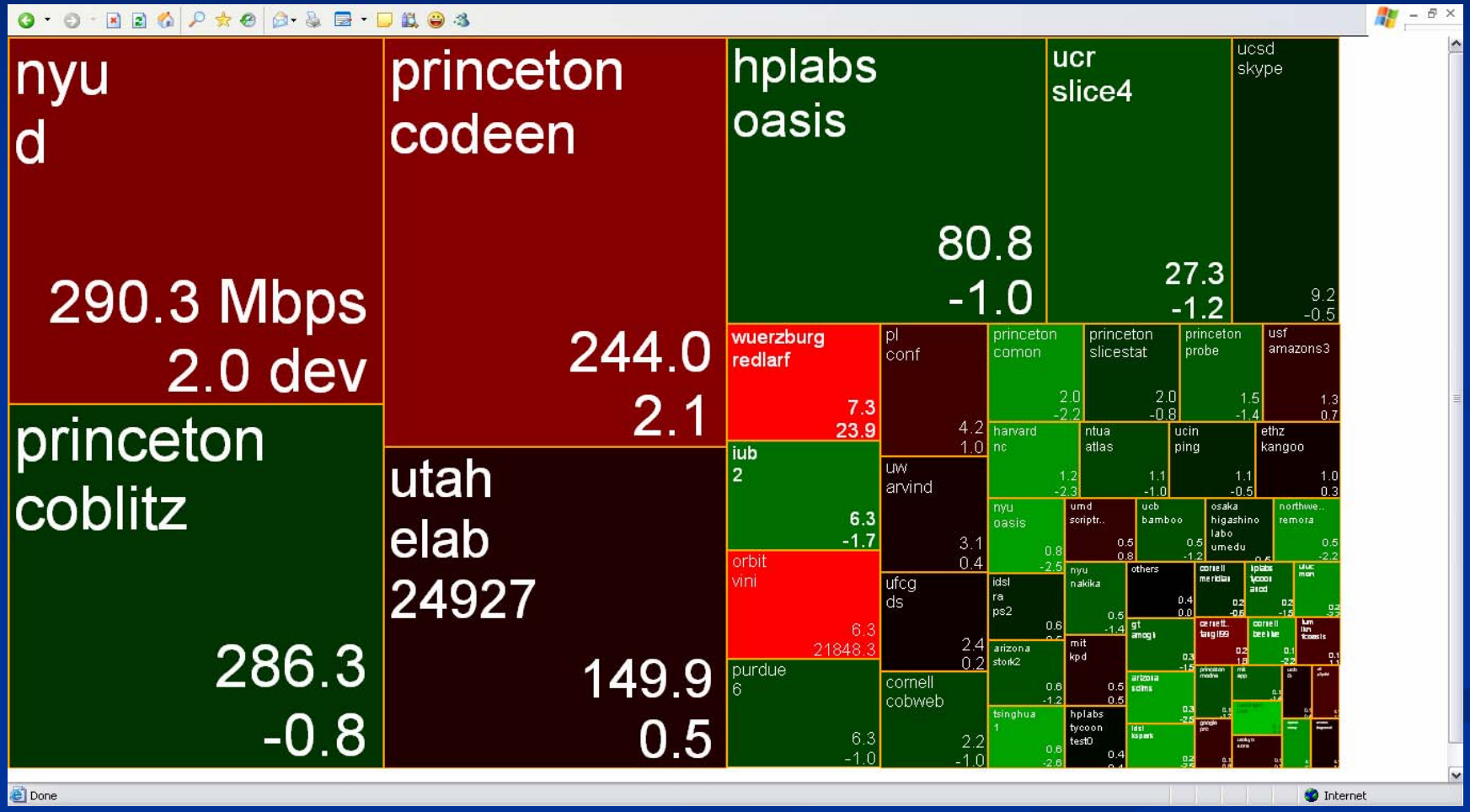
Approach & Technology

- PADS/ML Language & Compiler System
  - Declarative "data description language"
  - Supports ASCII, binary & mixed ad hoc data formats
  - Automatically generates a variety of programming libraries & stand-alone tools
- CoMon systems monitoring infrastructure
  - Current data volume: 700 nodes, 300 experiments, 10 data points/experiment
  - Interactivity: 5 minute refresh interval; history stored over two days
  - Infrastructure: data acquisition tools; simple query infrastructure; anomaly detection; multiple visualization modes

CoMon Node-Centric



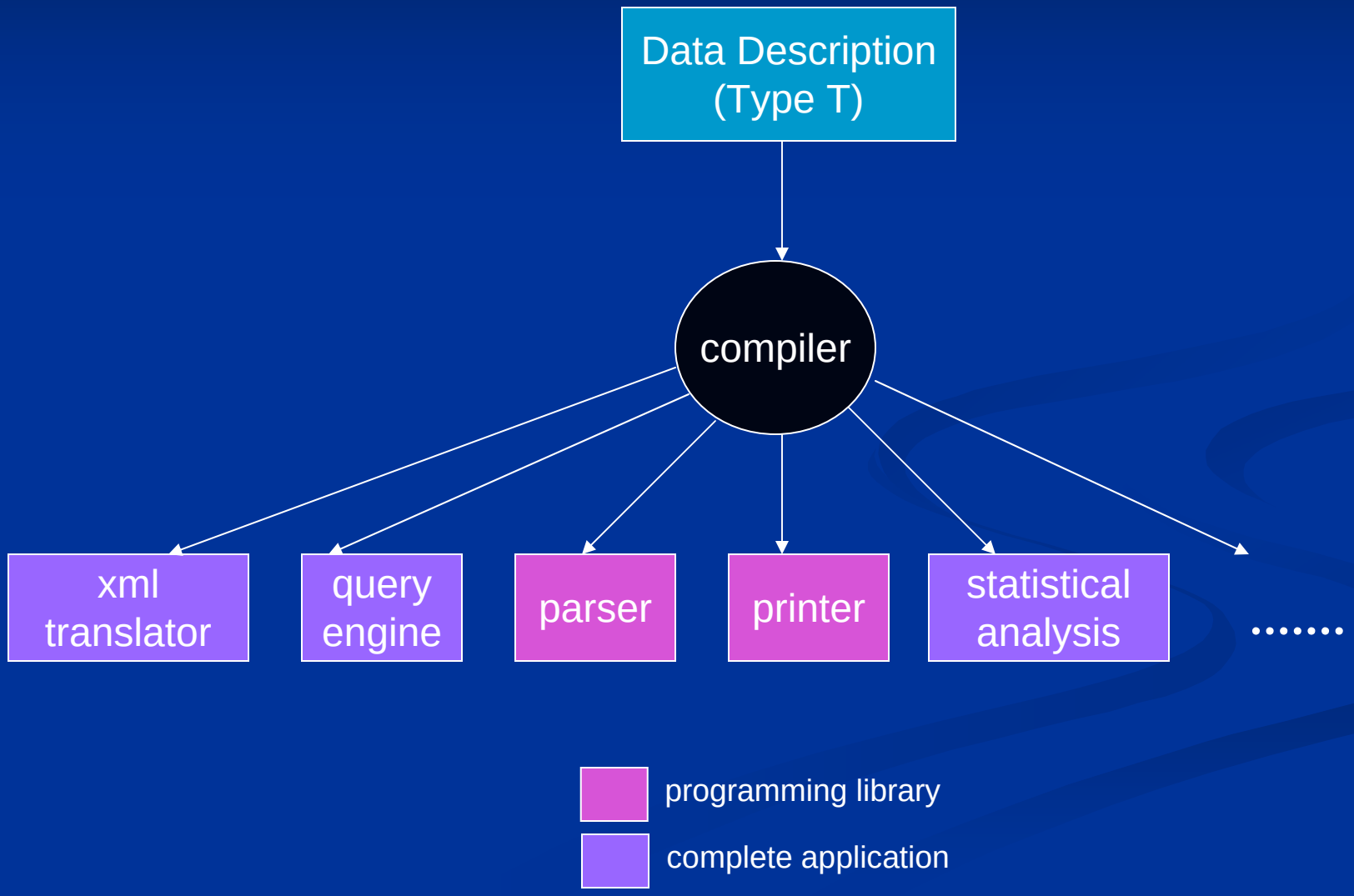
CoMon TreeMap Support



Progress Summary [Since Summer 06]

- PADS [www.padsproj.org]
  - Defined tool generation architecture & formal semantics [POPL 07, TFP 07]
  - First release of PADS/ML: Jan 07
  - Up Next: Support for distributed data sources
    - New abstractions for remote locations, access modes and schedule of data availability
  - Up Next: Automated inference of data descriptions
- CoMon
  - New visualization support & backend optimizations
  - Modularization of software infrastructure
  - Up Next: Automatic generation of data gathering

PADS:
One Description, Many Tools



CoMon Experiment-Centric

CoMon Statistics - Microsoft Internet Explorer

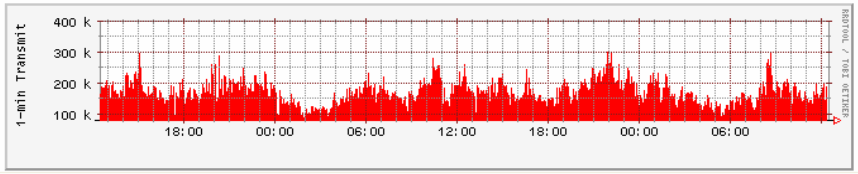
File Edit View Favorites Tools Help

Address http://summer.cs.princeton.edu/status/

CoMon Slice Usage Totals (sort key: 15-min Transmit)

Part of the CoDeeN project
Updated Thu Mar 22 12:20:02 2007
Summaries: By Node (long, short) By Slice (max, average, total, site) By Ports (all) By Site (all)
Problems (nodes, slices) Viz: Auto Resources (CPU, Mem, BW) Efficiency (CPU, Mem) Usage (Slices, Slivers, Nodes)

#	Slice Name	1-min Transmit	15-min Transmit	1-min Receive	15-min Receive	Num Procs	Phys Mem MB	Virt Mem MB	CPU %	MEM %	Long Ports	Snap Ports	# Nodes
1	nyu_d	225417	227919	66365	64718	1305	5570.1	8356.3	4997.0	463.8	2233	7064	268
2	princeton_coblitz	189840	166485	143977	124563	15528	22647.9	43634.6	1351.3	1743.2	48702	92430	352
3	princeton_codeen	110762	134623	116427	118014	15127	23504.8	44321.4	2680.2	1824.3	108188	183333	352
4	utah_elab_24927	31864	66533	39442	66640	2109	4802.4	9543.5	1943.8	366.1	1846	2956	243
5	hplabs_oasis	38777	40815	38563	40298	13554	21242.0	53065.2	925.1	1667.8	1263	13395	329
6	purdue_6	4323	4314	2028	1994	3581	2252.5	5525.0	833.7	186.7	245	7967	143
7	ucsd_skype	4043	4042	5155	5222	2417	7634.3	10518.1	229.3	646.3	0	13869	231
8	orbit_vini	2225	3221	2232	3279	112	807.2	2581.7	38.4	79.6	8	13	3
9	wuerzburg_redlarf	6614	3154	8670	4394	51	192.7	33004.5	15.8	12.9	23	536	11
10	princeton_slicestat	1454	1528	471	461	3402	2355.9	5885.2	0.0	184.1	612	43262	353

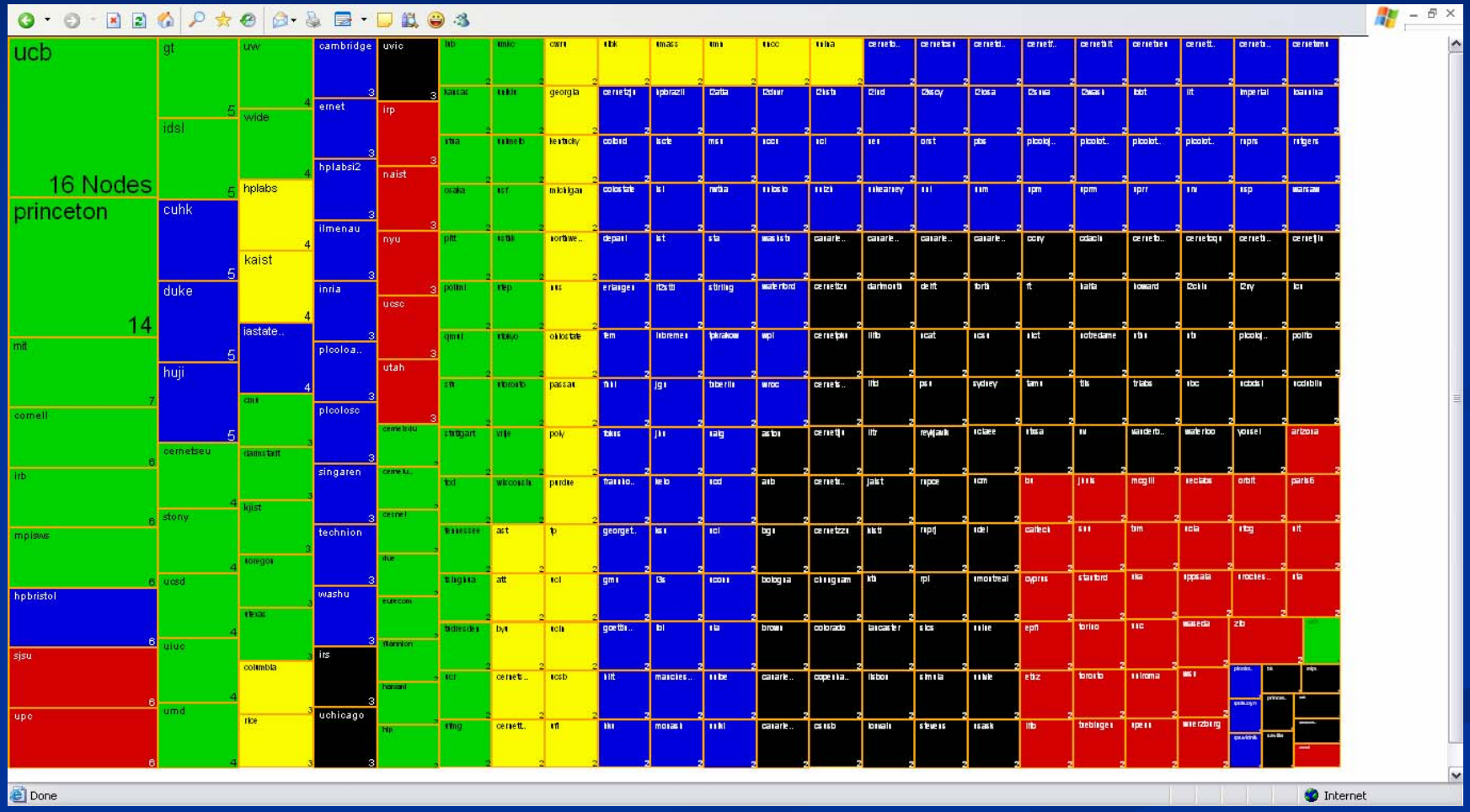


http://summer.cs.princeton.edu/status/viz_slivers.html

Internet

start temp_pads IPDPS_update CoMon Statistics - Mic... 12:26 PM

More TreeMap Support



Options

- Commercial systems (e.g., OpenView)
  - Good if you can afford it
  - Possibly steep SNMP support curve
- Grid systems (e.g., Ganglia)
  - Easy to deploy for Grid monitoring
  - Extensible, but framework different for extensions
- Ops monitoring (e.g., Nagios)
  - Event monitoring, versus quantity monitoring
  - Good for failure notifications, etc