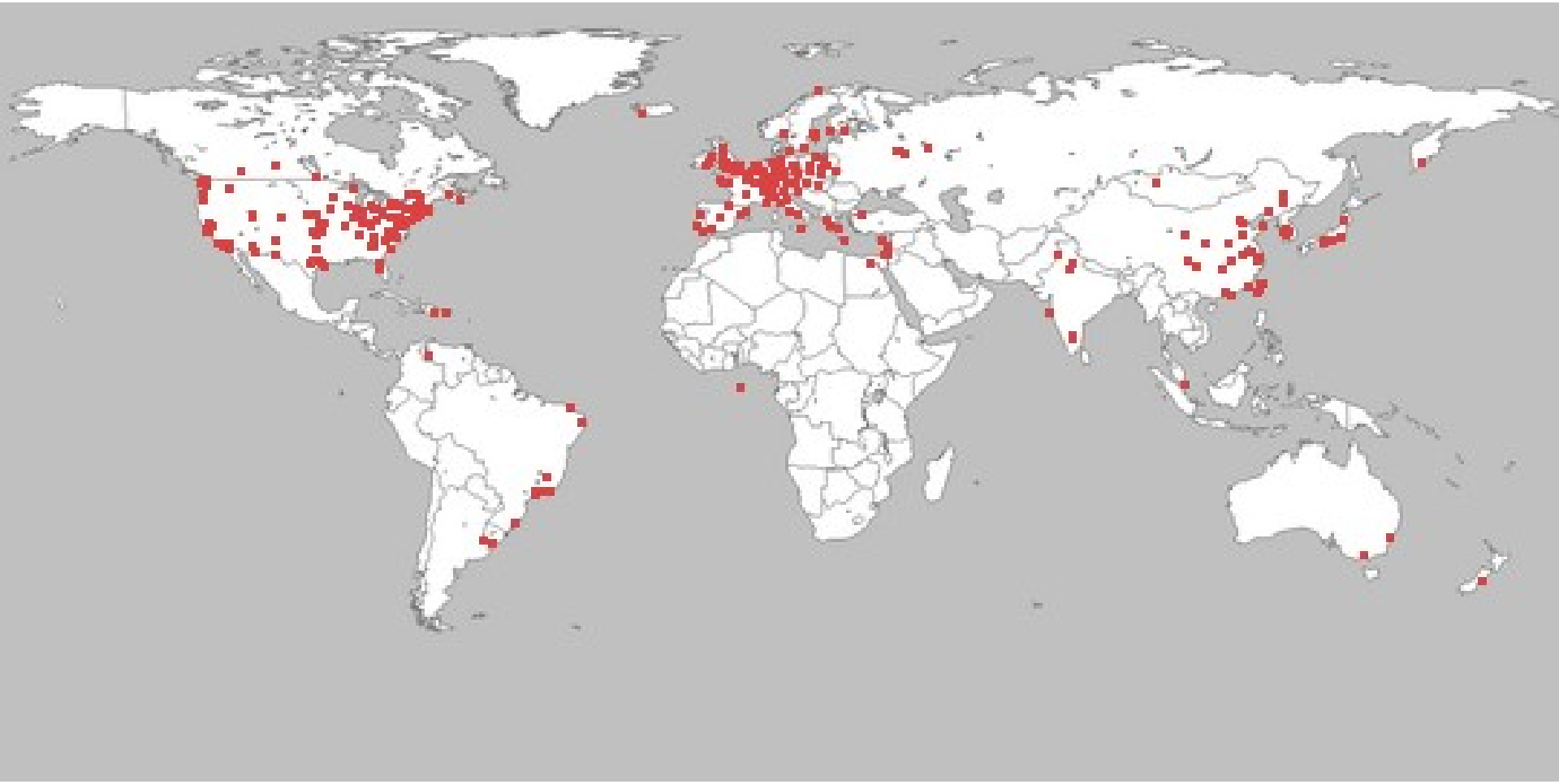
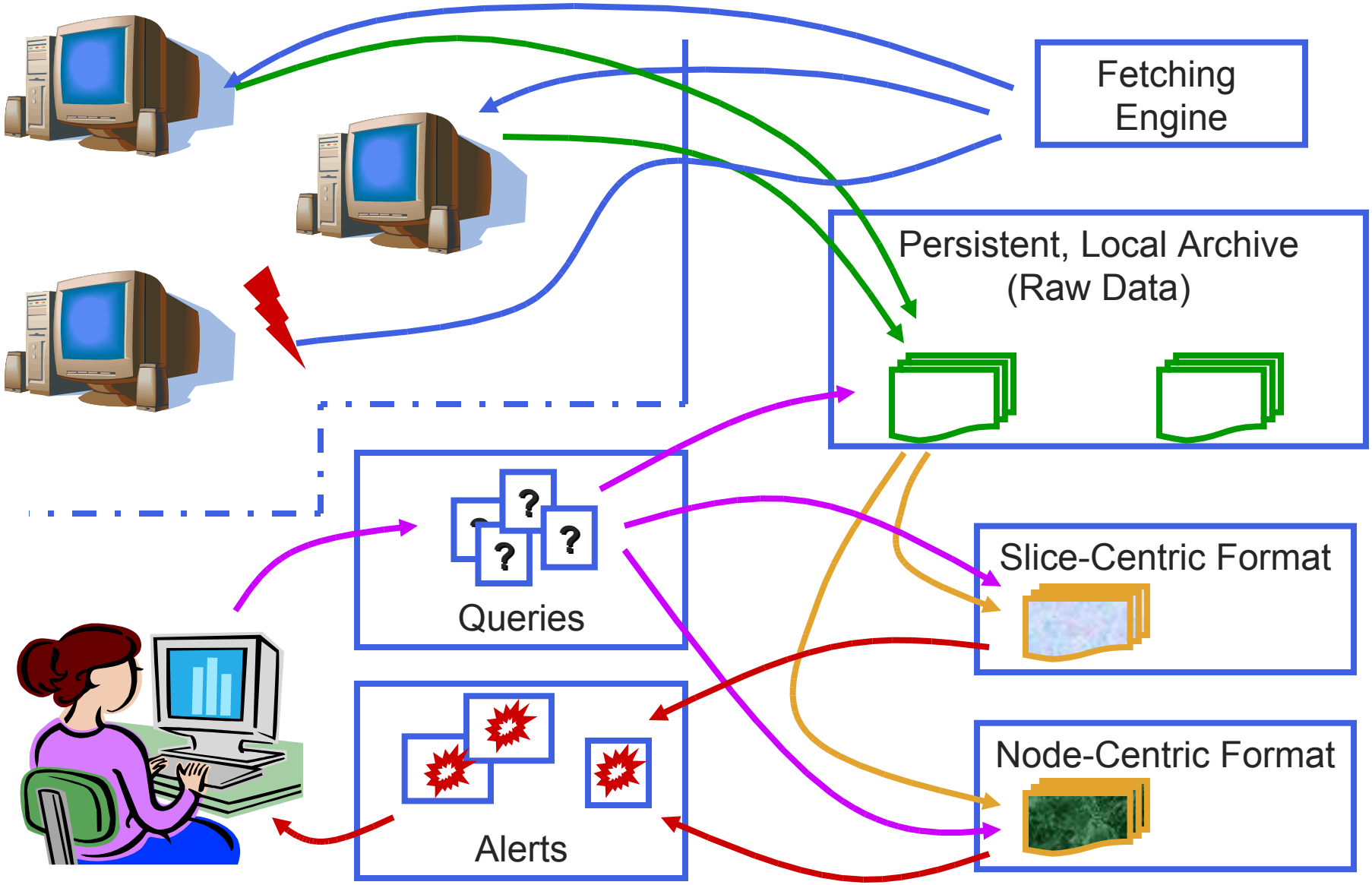


Monitoring PlanetLab

- Keeping PlanetLab up and running 24-7 is a major challenge
- Users (mostly researchers) need to know which nodes are up, have disk space, are lightly loaded, responding promptly, etc.
- CoMon [Pai & Park] is one of the major tools used to monitor the health, performance and security of the system

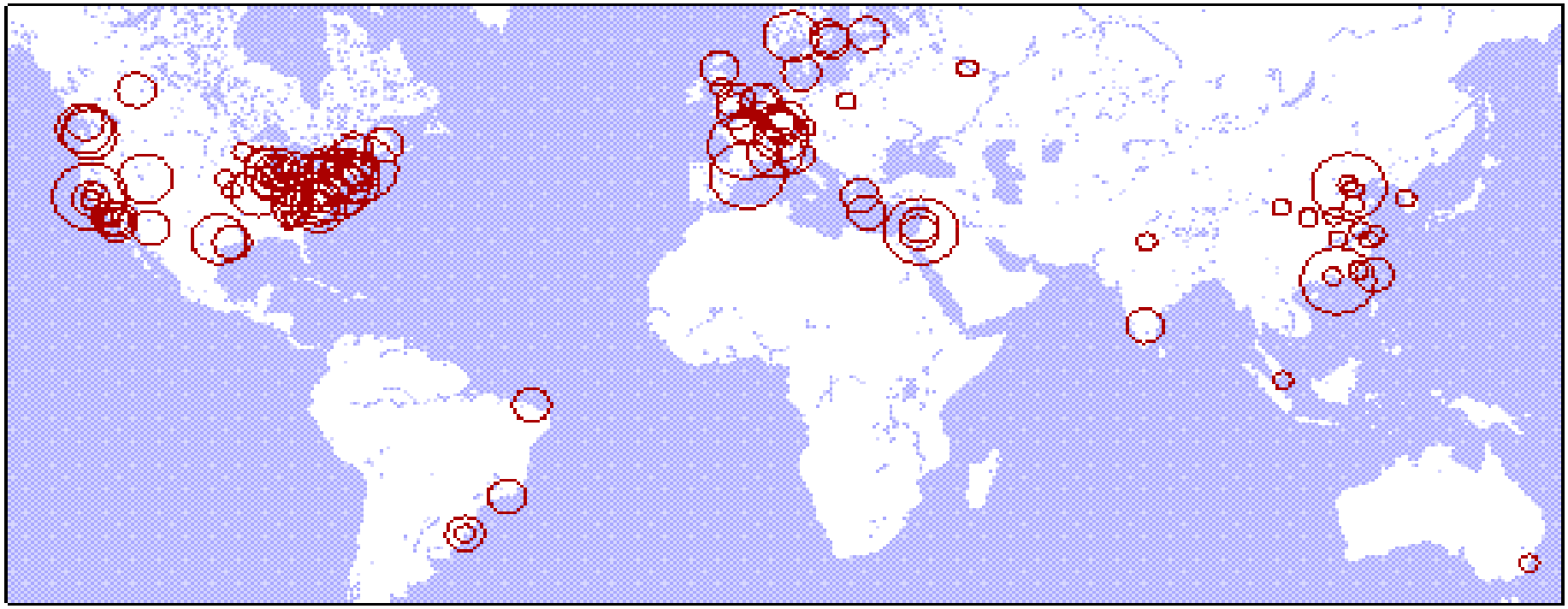
CoMon System Structure



Related Systems - AT&T Web Hosting

- An order of magnitude more complex than CoMon
- Many machines monitoring many AT&T servers
  - programs executed on remote machines to extract information
  - centralized archives, reports and alerts
- Extremely complex architecture
  - scripts and C programs and information passed through undocumented environment variables
  - you'd better hope the wrong guy doesn't get hit by a bus!

Related Systems - Coral CDN [Freedman]

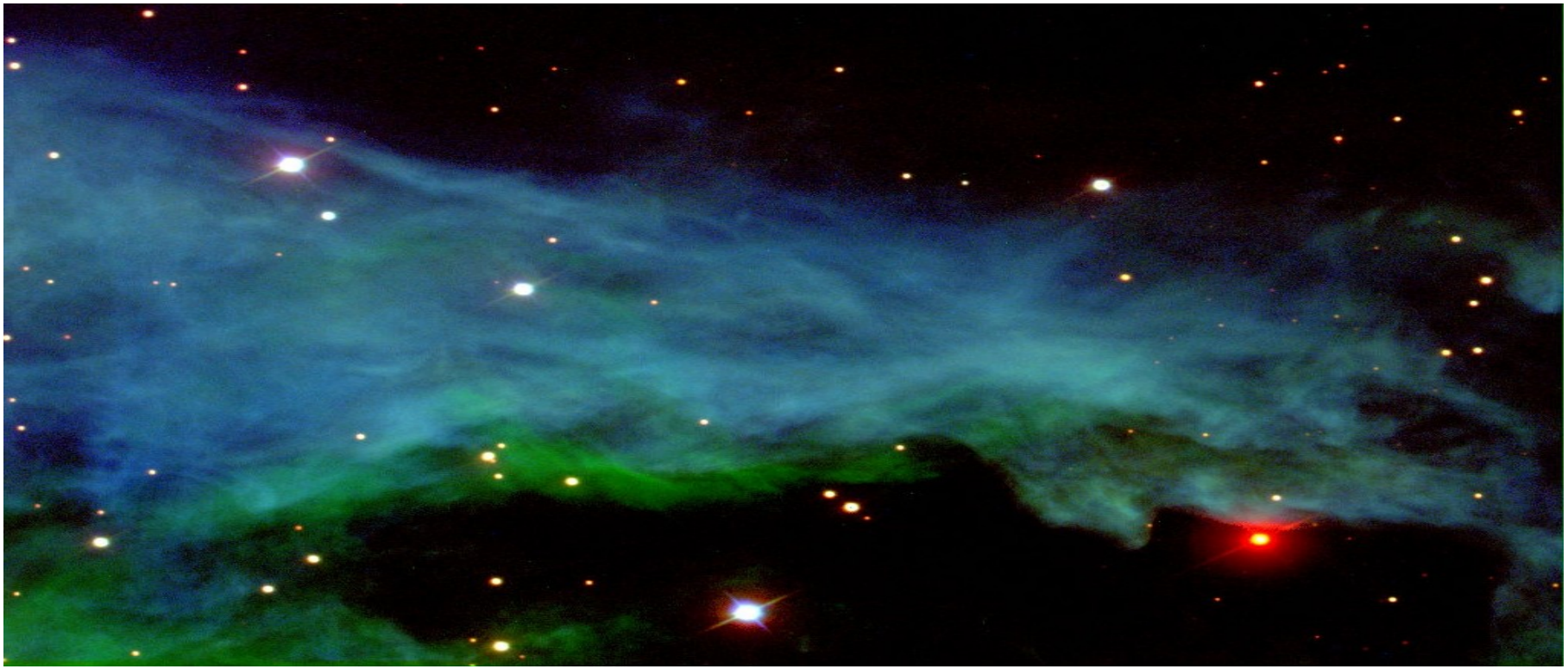


- 260 nodes worldwide
- periodic archiving for health, performance and research via scripts, perl and C
- data volume causes many annoyances:
  - too many files to use standard unix utilities

Related Systems - bioPixie [Troyanskaya et al.]

- An online service that pulls together information from a variety of other genomics information repositories to discover gene-gene interactions
- Sources include:
  - micro-array data, gene expression data, transcription binding sites
  - curated online data bases
  - source characteristics range from: infrequent but large new data dumps to modestly sized, regular (ie: monthly) dumps
- Most of the data acquisition is only partly automated

Related Systems – Cosmological Data



- Sloan Digital Sky Survey: mapping the entire visible universe
- Data available: Images, spectra, “redshifts,” object lists, photometric calibrations ... and other stuff I know even less about

Research Goals

To make acquiring, archiving, querying, transforming and programming with distributed ad hoc data so easy a caveman can do it.

Research Goals

To support three levels of abstraction/user communities:

- the **computational scientist**:
  - wants to study biology, physics; does not want to "program"
  - uses off-the-shelf tools to collect data & take care of errors, load a database, edit and convert to conventional formats like XML and RSS

- the **functional programmer**:
  - likes to map, fold, and filter (don't we all?)
  - wants programming with distributed data to be just about as easy as declaring and programming with ordinary data structures

- the **tool developers:**
  - enjoys reading functional pearls about the ease of developing apps using HOAS and tricked-out, type-directed combinators
  - develop new generic tools for user communities

Language Support for Distributed Ad Hoc Data

David Walker
Princeton University

In Collaboration With:

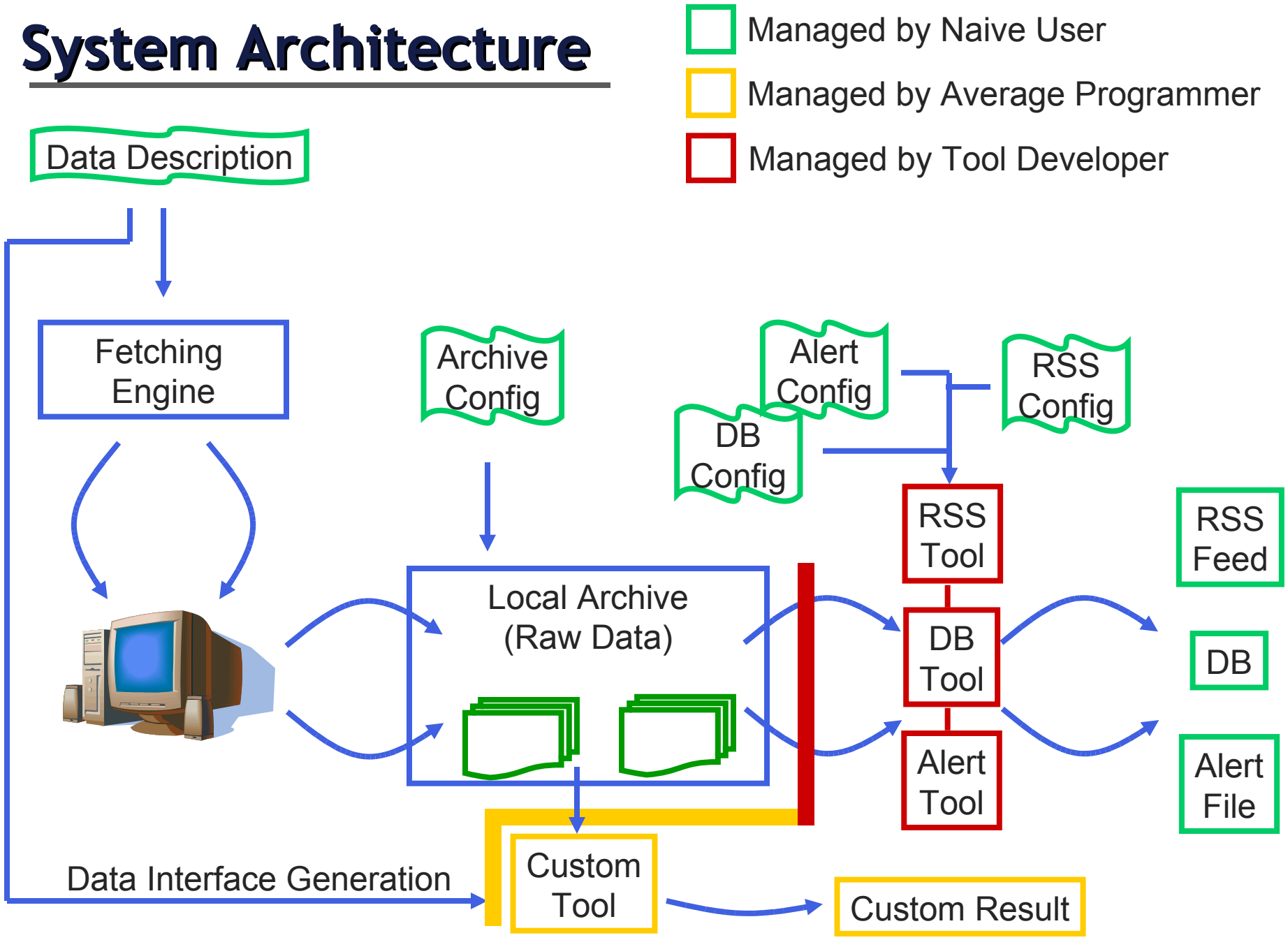
Daniel S. Dantas, Kathleen Fisher, Limin Jia, Yitzhak Mandelbaum, Vivek Pai, Kenny Q. Zhu

Approach

- Provide a domain-specific language extension for specifying properties of distributed data sources including:
  - **Location** or access function or data generation procedure
  - **Availability** (schedule of information availability)
  - **Format** (uses PADS/ML as a sublanguage)
  - **Proprocessing** information (decompression/decryption)
  - **Failure modes**

- From these specifications, generate “feeds” with nice interfaces for functional programmers and tool developers
  - streams of meta-data * data pairs
  - meta data includes schedule time, arrival time, location, network and data error codes

System Architecture



Back to CoMon …

Every node delivers this data every 5 minutes

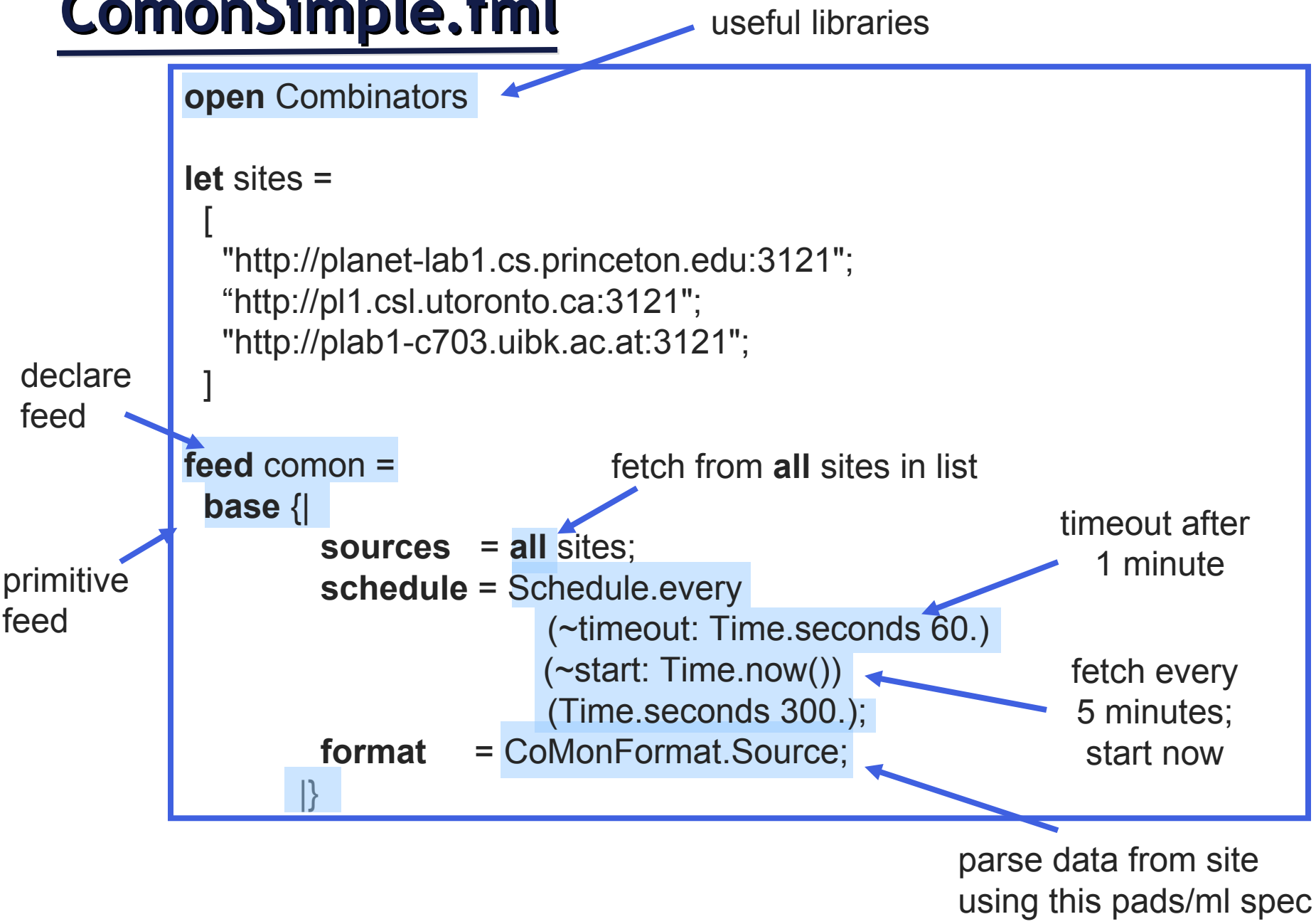
```
Date: 1202486984.709880
VMStat: 10 14 64 22320 24424 409284 0 0 4891 796 1971 2399 61 59 0 17
CPUUse: 60 100
DNSFail: 0.0 -1.0 0.0 -1.0
RWFS: 221
...
```

```
open Built_ins

ptype ‘a entry(name) = ...
ptype ‘a entry_list(name) = ...
ptype source = {
    date            : pfloat64 entry("Date");
    vm_stat         : pint entry_list("VMStat");
    cpu_use         : pint entry_list("CPUUse");
    dns_fail        : pfloat32 entry_list("DNSFail");
    rwfs            : pint entry("RWFS");
    ...
}
```

CoMonFormat.pml
[see Mandelbaum’s thesis]

ComonSimple.fml



Tool Configs

parameters

tool name

```
Tool archive
{
 arch_dir          = “temp/”;
 log_file_name     = “comon”;
 max_file_count    = 1;
 compress_files    = true;
}
```

```
Tool accum
{
 minalert    = false;
 maxalert    = false;
 lesssig     = Some 3;
 moresig     = Some 3;
 useralert   = fn x -> x;
 slicesize   = Some 1000;
 slicefile   = Some “accumslice.xml”;
 totalfile   = Some “accum.xml”;
}
```

```
Tool rss
{
  title      = “PlanetLab Disk Usage”;
  link       = “http://comon.cs.princeton.edu”;
  desc       = “This rss feed provides PlanetLab Disk usage info”;
  schedule = Some (Time.seconds 300.);
  path       = comon.source.entries.diskusage ;
  rssfile    = Some “rssdir/comon.rss”;
}
```

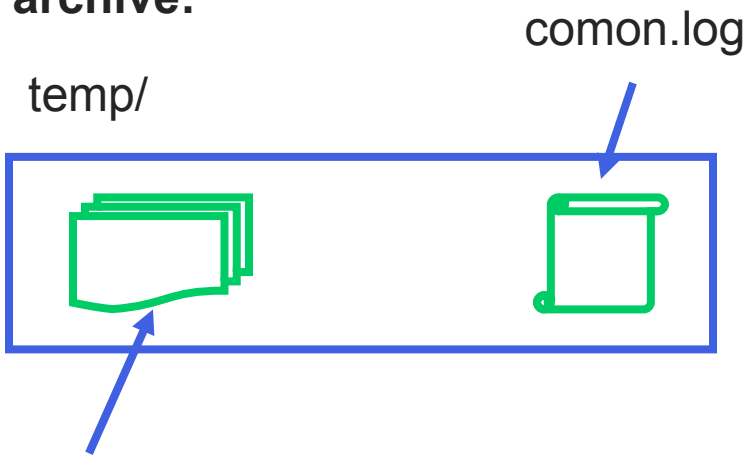
```
Tool rrd
{ ... }
```

```
Tool print
{ ... }
```

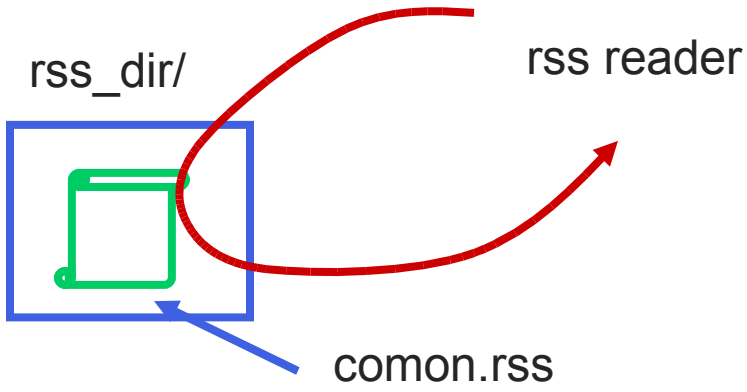
```
Tool select
{ ... }
```

Tool Results

archive:



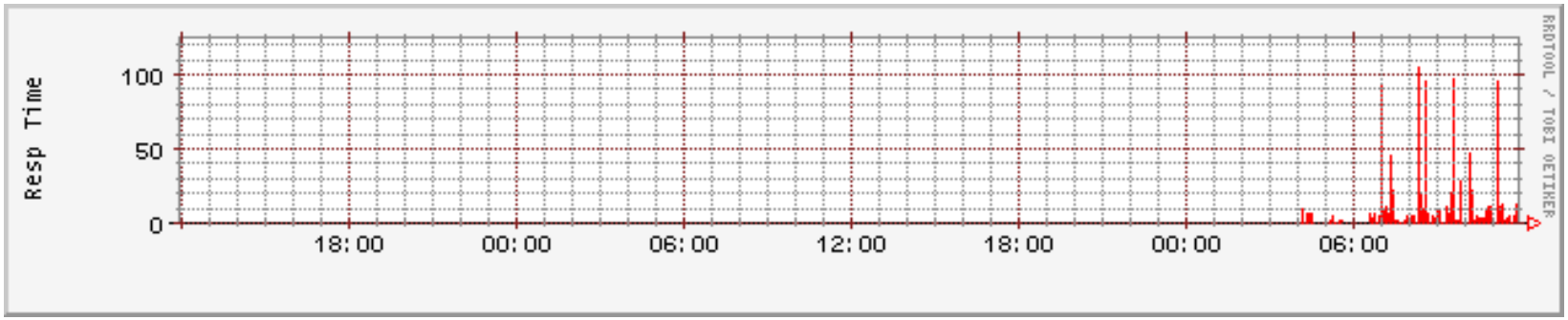
rssfeed:



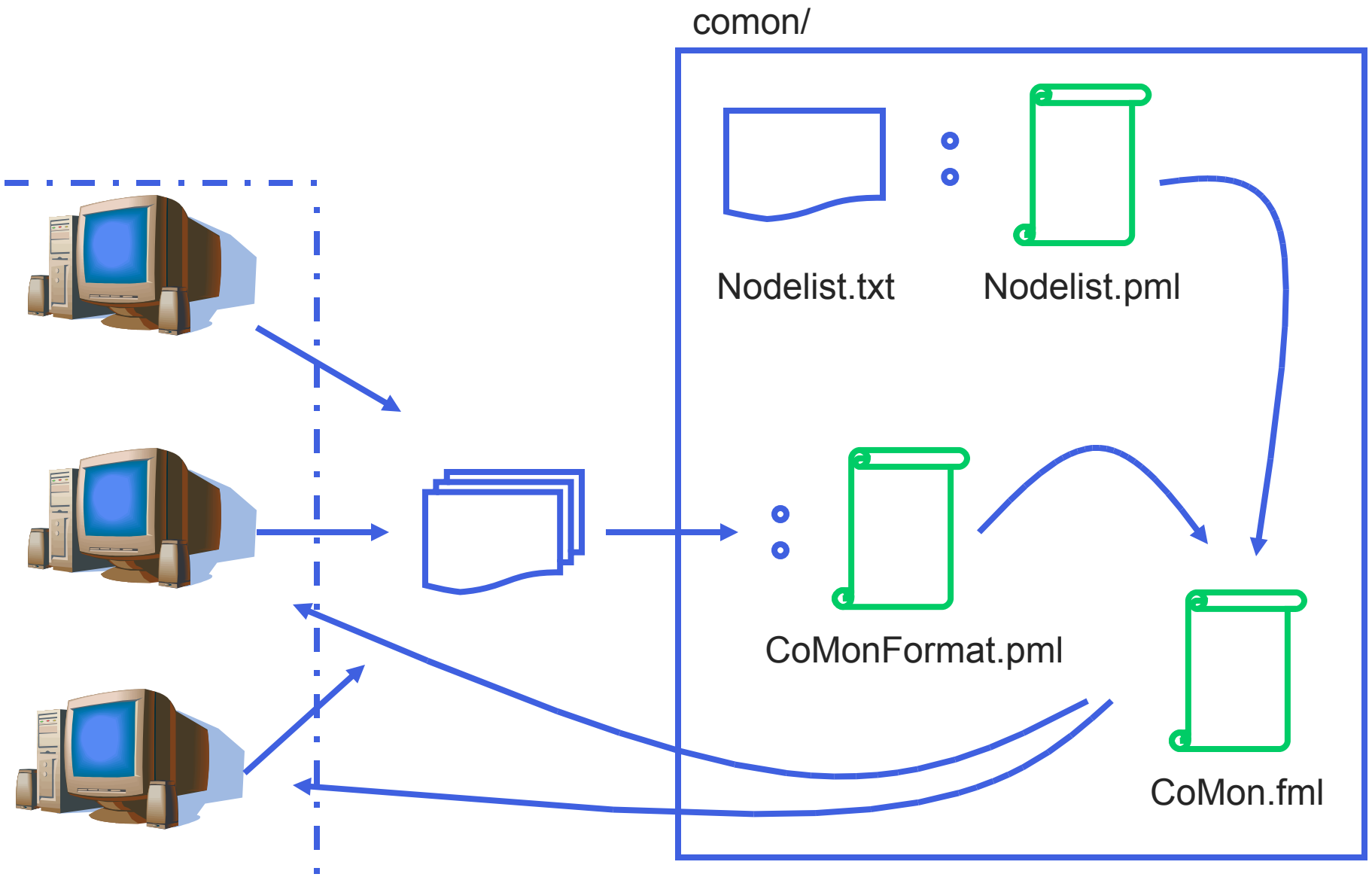
accum:

```
<feed_accumulator>
  <net_errors>
    <error>
      <errcode>1</errcode>
      <errmsg>Misc HTTP error</errmsg>
...
```

rrd:



A More Advanced Example: CoMon.fml



Format Descriptions

Nodelist.txt:

```
plab1-c703.uibk.ac.at
plab2-c703.uibk.ac.at
#planck227.test.ibbt.be
#pl1.csl.utoronto.ca
#pl2.csl.utoronto.ca
#plnode01.cs.mu.oz.au
#plnode02.cs.mu.oz.au...
```

CoMonFormat.pml (as before):

```
open Built_ins

ptype ‘a entry(name) = ...
ptype ‘a entry_list(name) = ...
ptype source = {
    date        : pfloat64 entry("Date");
    vm_stat     : pint entry_list("VMStat");
...
}
```

Nodelist.pml:

```
open Built_ins

ptype nodeitem =
  Comment of '#' * pstring_SE(peor)
| Data of pstring_SE(peor)

ptype source = nodeitem precord plist (No_sep, No_term)
```

CoMon.fml:

```
let isNode item = match item with Hosts.Data s -> true | _ -> false

let makeURL (Nodelist.Data s) = "http://" ^ s ^ ":3121"

feed nodelists =  base {|
   sources      = all ["file:///" ^ Sys.getcwd ()  ^ "/nodelist"];
   schedule     = Schedule.every (Time.hours 24.);
   format       = Nodelist.Source;
|}

feed comon =
  foreach nodelist in nodelists create
    base {|
          sources   = all (List.map makeURL (List.filter isNode nodelist));
          schedule = Schedule.every (~start:Time.now())
                                    (~duration:Time.hours 24.)
                                    (Time.minutes 5.);
          format    = CoMonFormat.Source;
    |}
```

find local nodelist

grab it every day

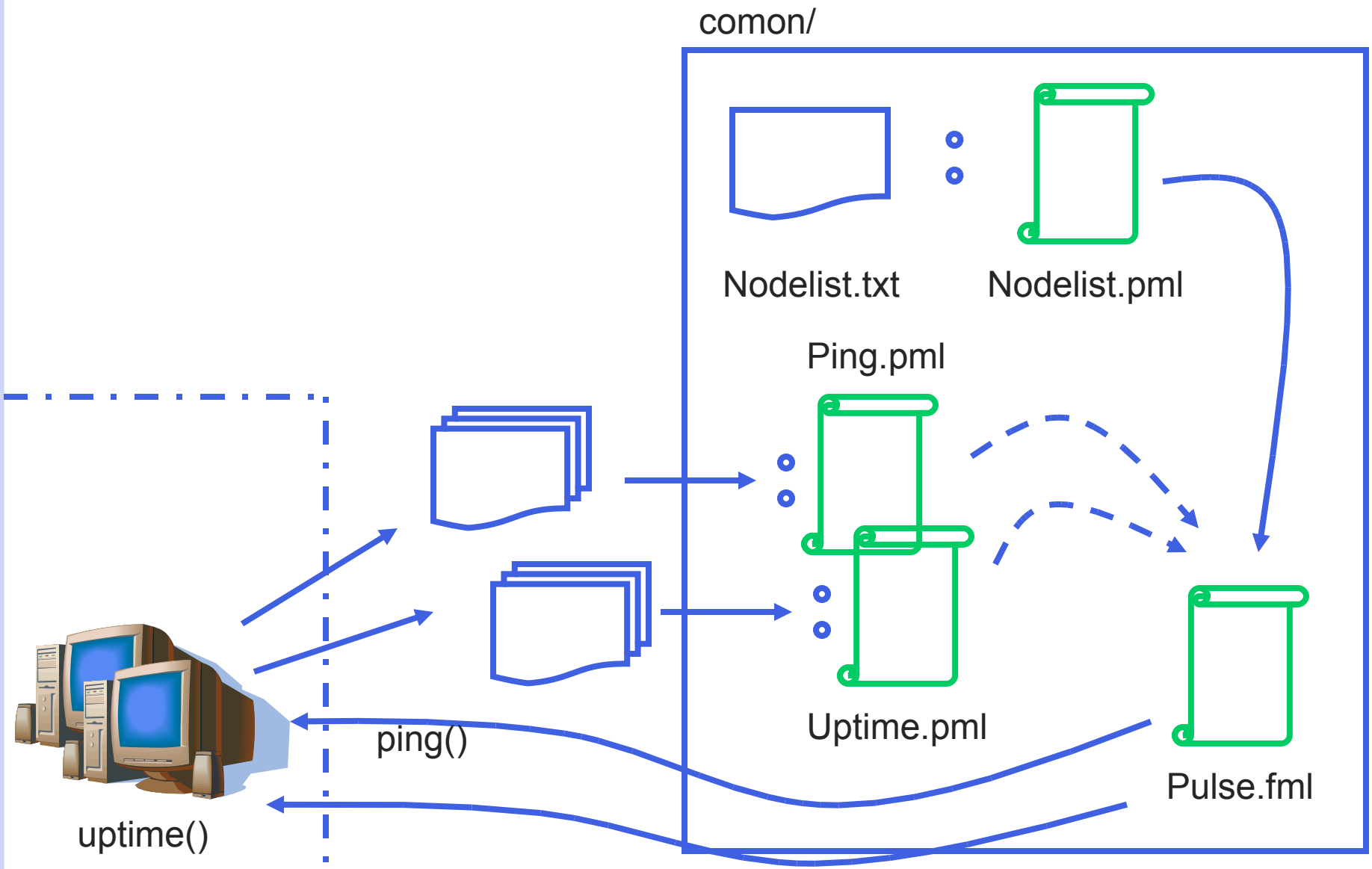
construct URL syntax

filter out comment lines

fetch every 5 min all day long

repeatedly get current nodelist

AT&T Web Hosting



Pulse.fml:

```
let isNode item = match item with Hosts.Data s -> true | _ -> false
let mk_host (Hosts.Data h) = h

feed hostList =  base {|
   sources   = all ["file:///" ^ Sys.getcwd ()  ^ "/machine_list"];
   schedule = Schedule.every (~start:(Time.now())) (Time.hours 24.);
   format     = Hosts.Source;
|}

feed hosts = {| mk_host n | n <- (flatten hostList), isNode n |}

feed stats =
  foreach h in hosts create
  let s = Schedule.once (~timeout: Time.seconds 60.) () in
  ( base {| sources   = proc ("ping -c 2 " ^ h);
            format     = Ping.Source;
            schedule = s; |},

    base {| sources   = proc ("ssh " ^ h ^ " uptime");
            format     = Uptime.Lines;
            schedule  = s; |}
  )
```

get hostlists

create intermediate feed of hosts

execute ping

format Ping.Source

execute uptime

pair results in feed

Formal Semantics

Feed Typing Rules: G |- F : t feed

Denotational Semantics:

[[F]] : universe -> environment -> (meta * value) set

where

type universe = location * time -> value * time

type environment = variable -> value

type meta = time * ...

Questions I have

- What are the *essential* language constructs/combinators?
- What are the *essential* tools we need to provide to our naive users?
- What are the *canonical* interfaces we should be providing?

- How would I implement this in Haskell or Clean or F#?

Conclusion

- PADS/D is (will be!) a high-level, declarative language designed to make it easy to specify:
  - where your data is located
  - how your data is generated
  - when your data is available
  - what preprocessing needs to be done
  - how to handle failure conditions
- And generate useful processing tools:
  - archiver, rss feeds, database, error profiler, debugging printer, ...
- And facilitate functional programming with distributed data

Example program

```
open Feedmain
open ComonSimple

let myspec = comon

let emptyT () = Hashtbl.create 800
let addT t idata =
  let (meta, data) = (IData.get_meta idata, IData.get_contents idata) in ...
let printT t = ...
let getload idata = match (IData.get_contents i) with
  None -> None | Some d -> List.hd (d.loads.2)

(* every 600 seconds output the 10 locations with the least load *)
let rec findnodes f =
  let (slice, rest) = sliceuntil (later_than (Time.now() +. 600.)) f in
  let loads = mapi getload slice in
  let loadT = foldi addT emptyT loads in
  let _ = printT loadT in
  findnodes rest

findnodes (to_feed myspec)
```

Formal Typing

Feed Typing Rules:

G |- F : t feed

Example Rules:

G |- F1 : t1 feed G |- F2 : t2 feed

G |- (F1,F2) : t1 * t2 feed

G |- F1 : t1 feed G,x:t1 |- F2 : t2 feed

G |- foreach x in F1 create F2 : t2 feed