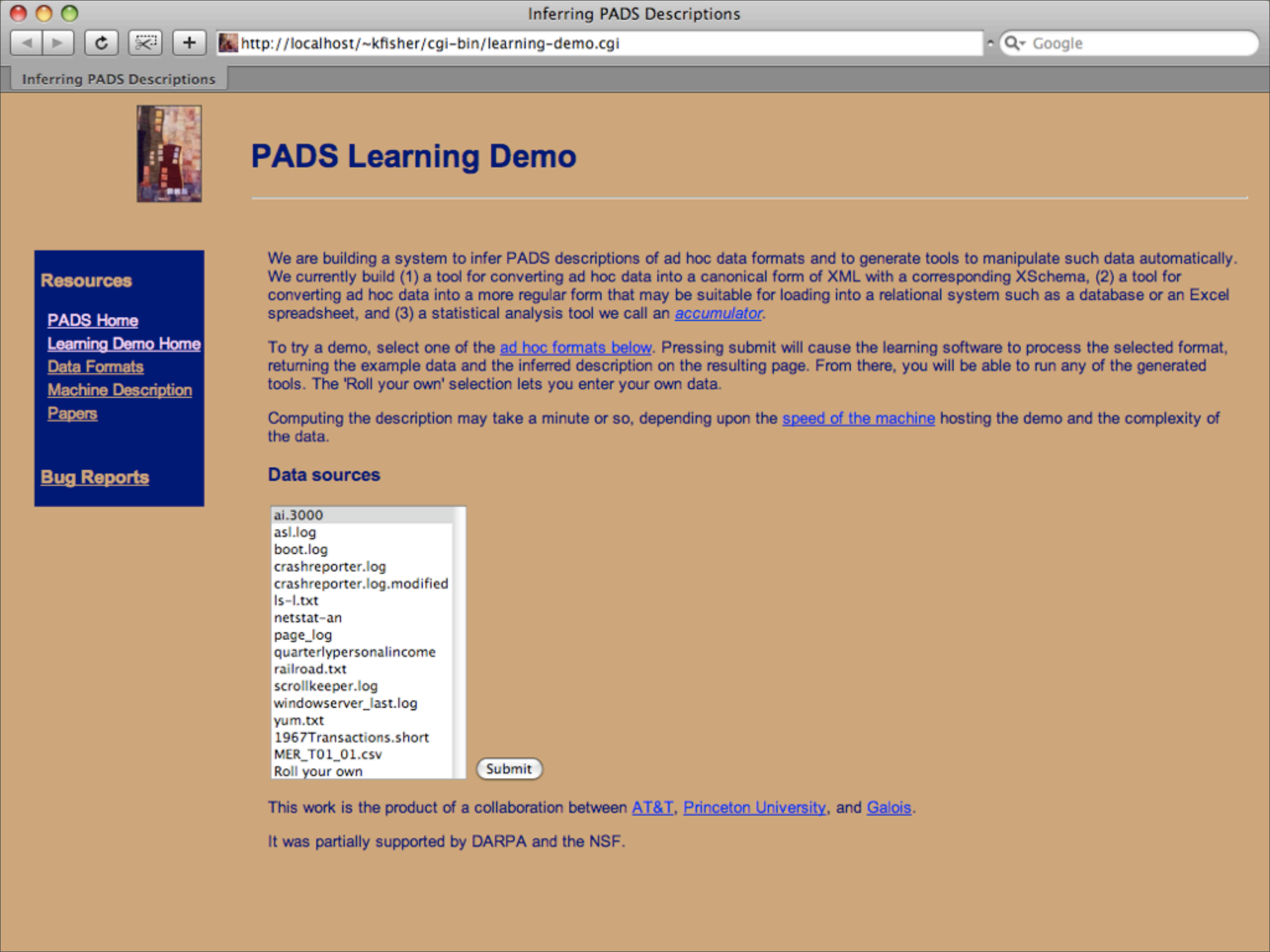
Typing Directories

Kathleen Fisher
AT&T Labs Research

Joint work with David Walker and Kenny Zhu


Inferring PADS Descriptions
http://localhost/~kfisher/cgi-bin/learning-demo.cgi
Google
Inferring PADS Descriptions

PADS Learning Demo

Resources
PADS Home
Learning Demo Home
Data Formats
Machine Description
Papers
Bug Reports

We are building a system to infer PADS descriptions of ad hoc data formats and to generate tools to manipulate such data automatically. We currently build (1) a tool for converting ad hoc data into a canonical form of XML with a corresponding XSchema, (2) a tool for converting ad hoc data into a more regular form that may be suitable for loading into a relational system such as a database or an Excel spreadsheet, and (3) a statistical analysis tool we call an *accumulator*.

To try a demo, select one of the ad hoc formats below. Pressing submit will cause the learning software to process the selected format, returning the example data and the inferred description on the resulting page. From there, you will be able to run any of the generated tools. The 'Roll your own' selection lets you enter your own data.

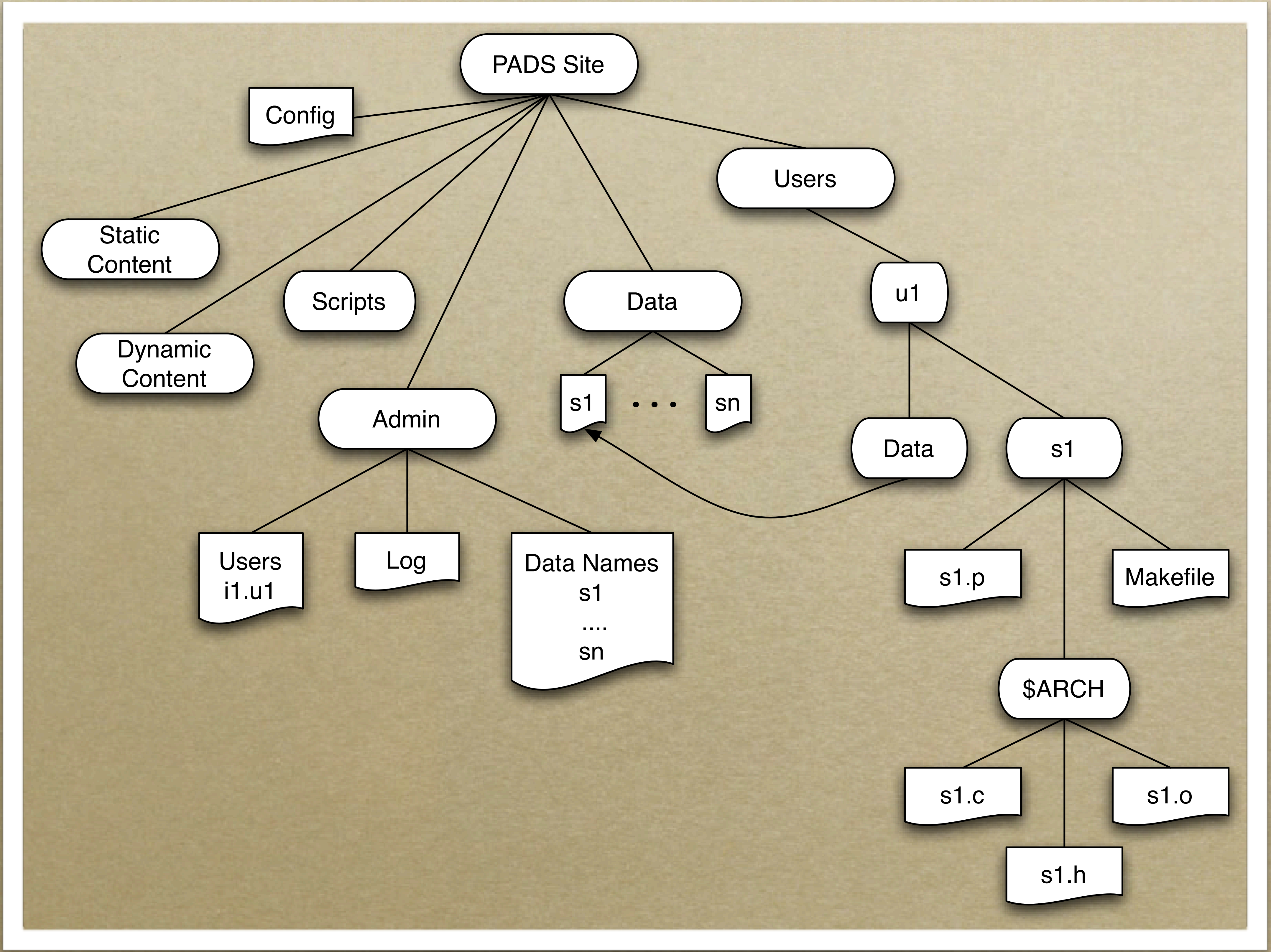
Computing the description may take a minute or so, depending upon the speed of the machine hosting the demo and the complexity of the data.

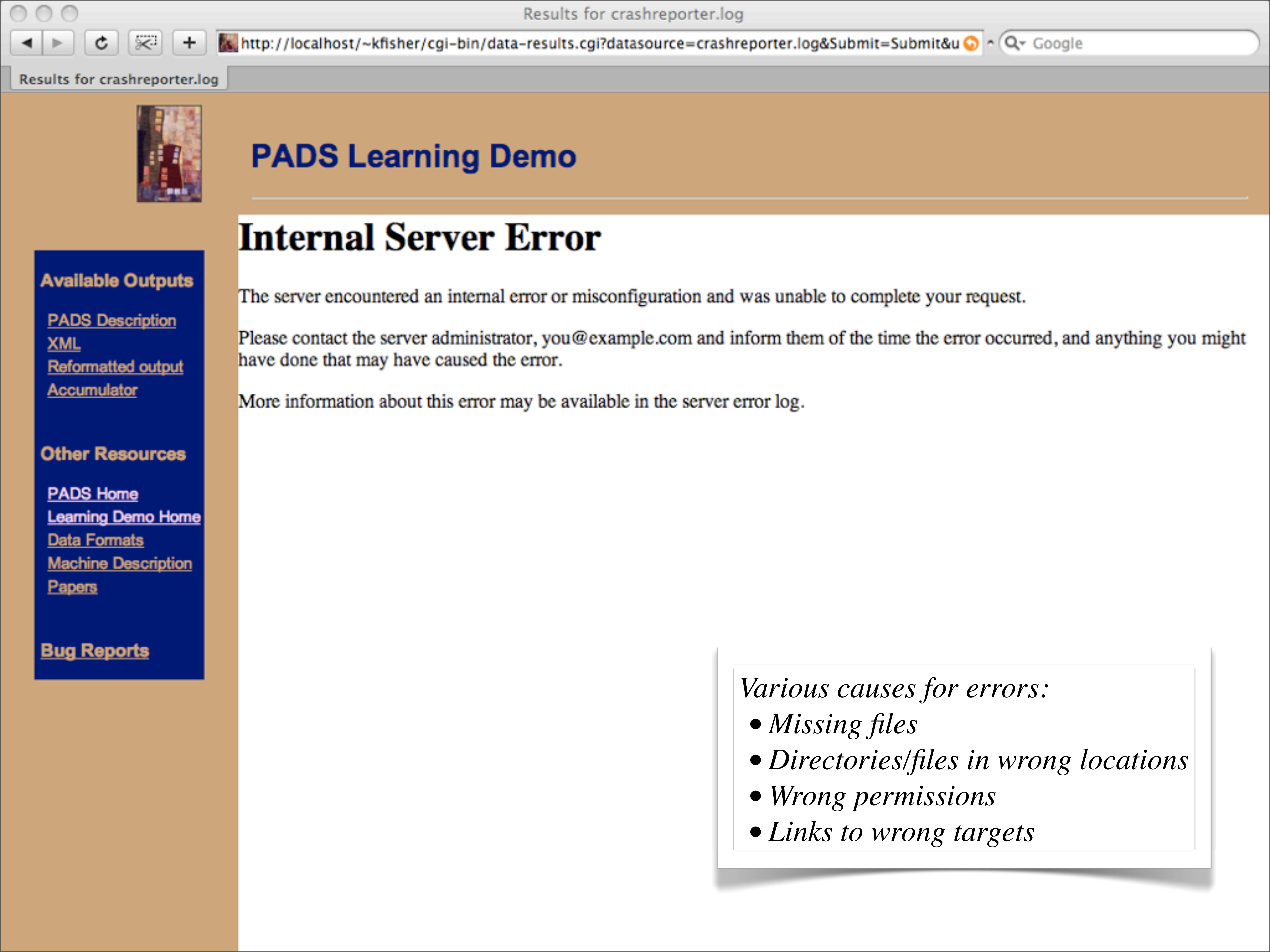
Data sources

This work is the product of a collaboration between AT&T, Princeton University, and Galois.

It was partially supported by DARPA and the NSF.

PADS Web Site





Internal Server Error

The server encountered an internal error or misconfiguration and was unable to complete your request.

Please contact the server administrator, you@example.com and inform them of the time the error occurred, and anything you might have done that may have caused the error.

More information about this error may be available in the server error log.

Various causes for errors:

- *Missing files*
- *Directories/files in wrong locations*
- *Wrong permissions*
- *Links to wrong targets*

If only I could...!!!

- *Describe required file and directory structure, including permissions, etc.*
- *Check that actual file system matches specification.*
- *Eliminate a whole class of errors!*

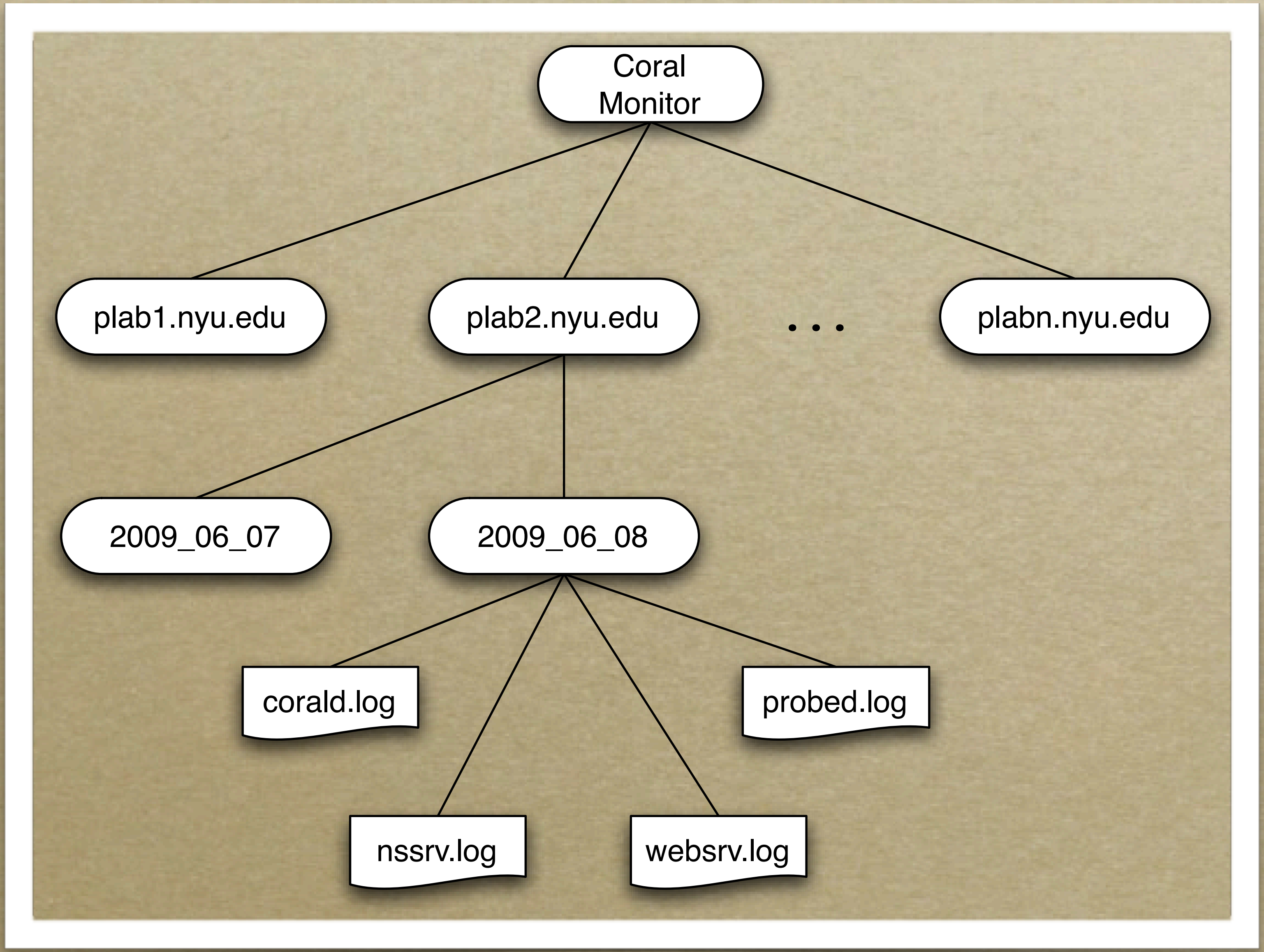
If only I could...!!!

- *Describe required file and directory structure, including permissions, etc.*
- *Check that actual file system matches specification.*
- *Eliminate a whole class of errors!*

Are there other examples where such a description might be useful?

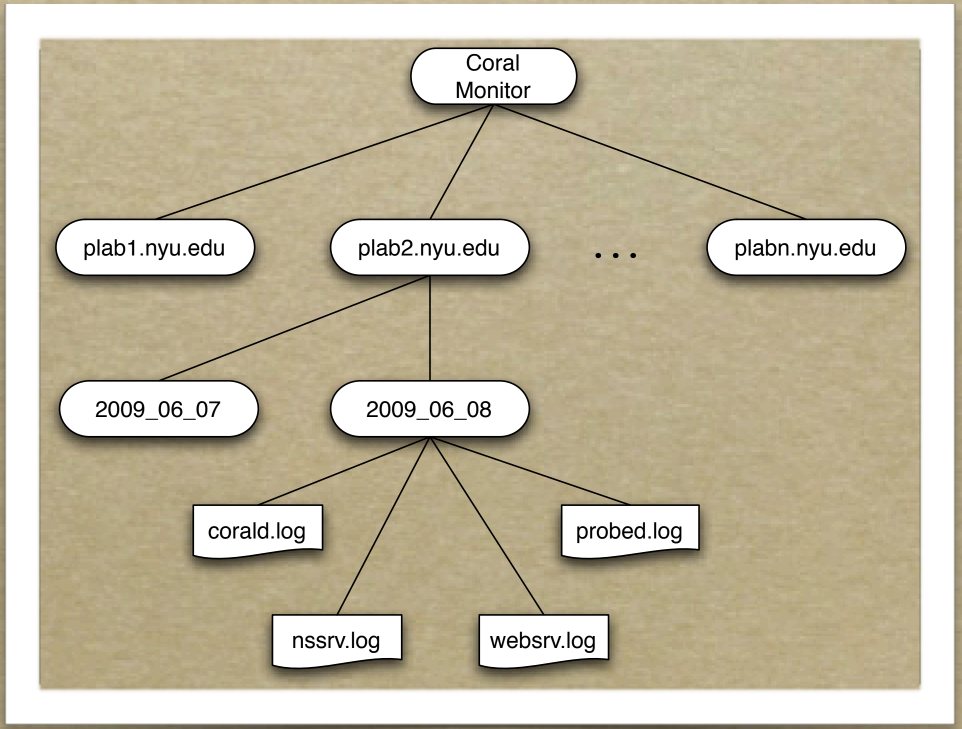
Coral Monitoring System

Monitoring system for an "Internet-scale, self-organizing, web-content distribution network" developed at Princeton.

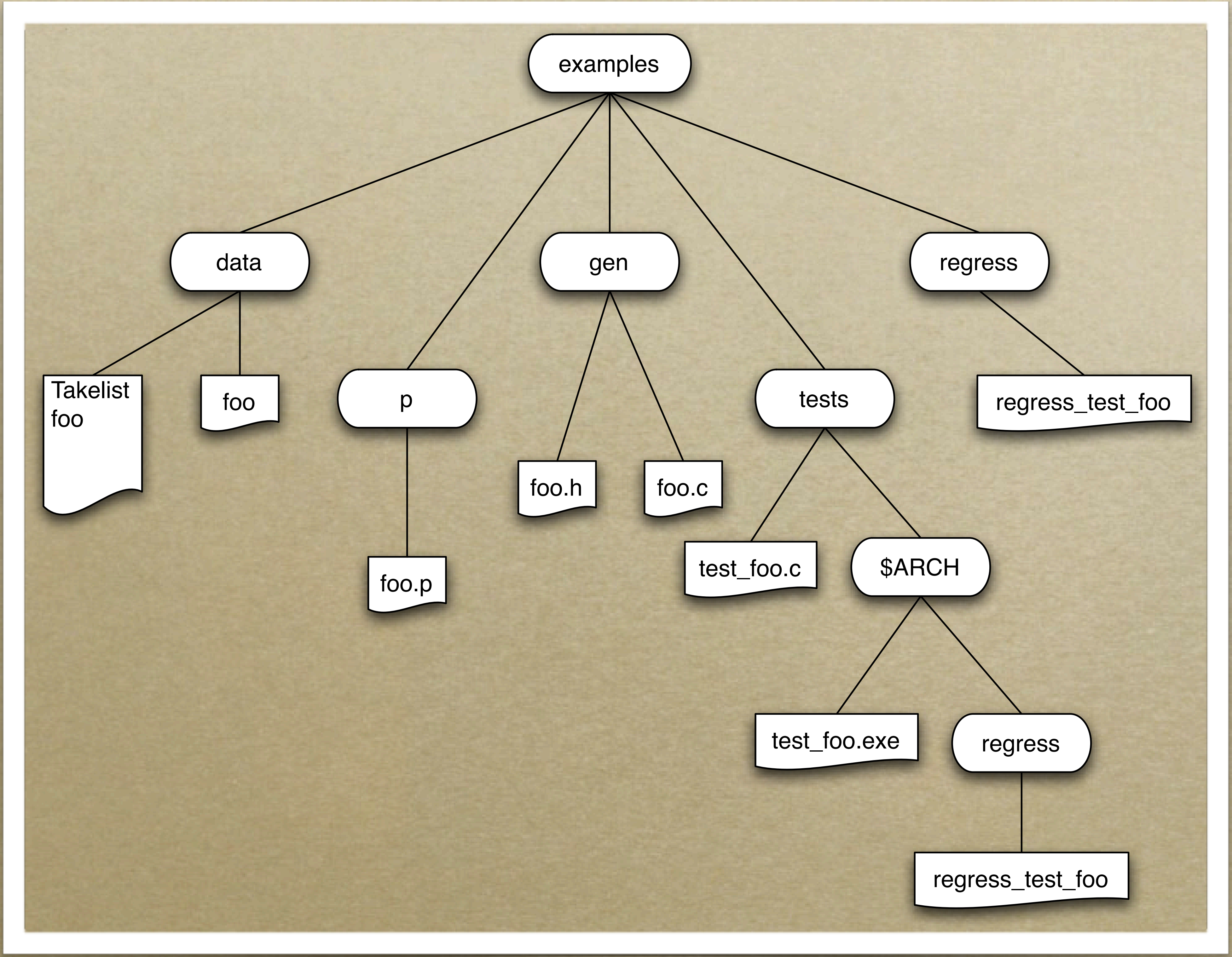


Observations on Monitoring

- *Coral is similar to other monitoring systems: PlanetLab and a multitude of systems at AT&T.*
- *Often a configuration file specifies which hosts to monitor, what data to collect, and how often.*
- *File and directory names encode meta-data.*
- *Want to guarantee no missing or corrupt files.*



PADS Regression Suite



Other Possible Examples

- *GHC SourceTree*
- *Cabal system for GHC libraries*
- *File Hierarchy Standard (FHS) for unix-like installations*
- *CVS, SVN, other source control systems*
- *Disk cache for browser history, IMAP mail*
- *Scientific data sets:*
  (e.g., "Huge Data but Small Programs")

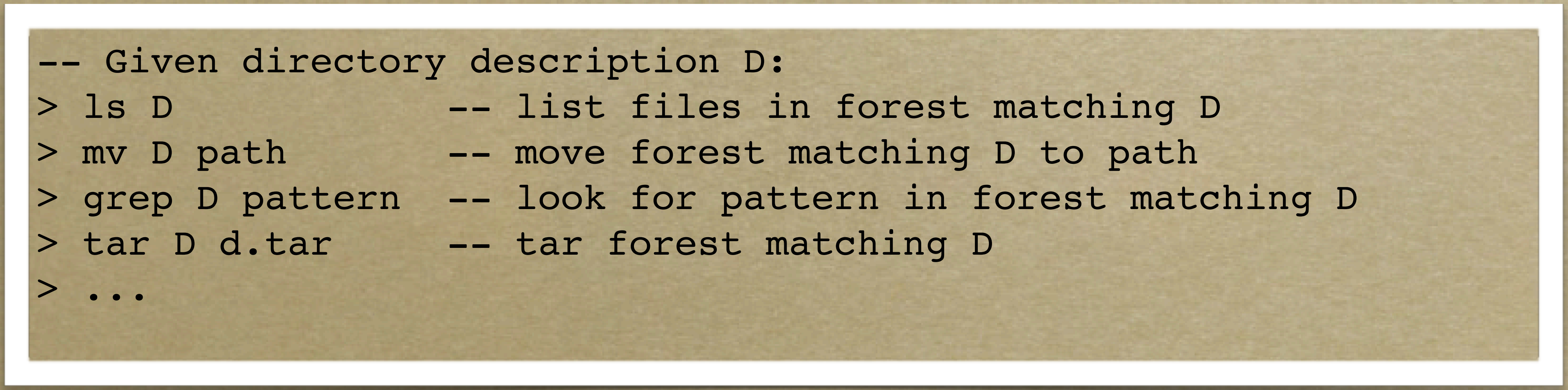
Other Possible Examples

- *GHC SourceTree*
- *Cabal system for GHC libraries*
- *File Hierarchy Standard (FHS) for unix-like installations*
- *CVS, SVN, other source control systems*
- *Disk cache for browser history, IMAP mail*
- *Scientific data sets:*
  (e.g., "Huge Data but Small Programs")

Question 1: Can you think of other examples?

Possible Uses

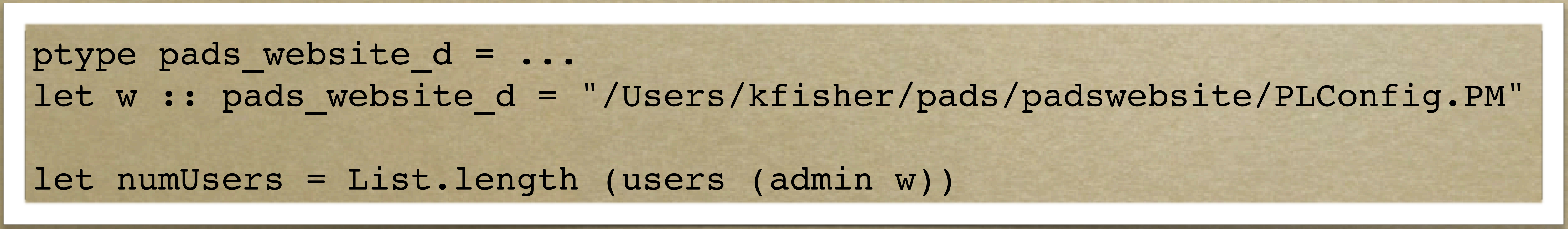
- *Document structure*
  - *Where do I find particular file? Where do I put a particular file? Output of system probe tools?*
- *Check/Fix current state*
  - *Missing or extra files, wrong permissions owners or groups, wrong link targets, stale data.*
- *Semantics-based shell tools*



```
-- Given directory description D:
> ls D               -- list files in forest matching D
> mv D path          -- move forest matching D to path
> grep D pattern     -- look for pattern in forest matching D
> tar D d.tar        -- tar forest matching D
> ...
```

Possible Uses, continued

- *Programmatic interface to directories and files that leverages PL idioms.*
  - *Connect program variable to path on disk with associated directory description*
  - *Lazily construct an in-memory representation*



```
ptype pads_website_d = ...
let w :: pads_website_d = "/Users/kfisher/pads/padswebsite/PLConfig.PM"

let numUsers = List.length (users (admin w))
```

Possible Uses, continued

- *Programmatic interface to directories and files that leverages PL idioms.*
    - *Connect program variable to path on disk with associated directory description*
    - *Lazily construct an in-memory representation*

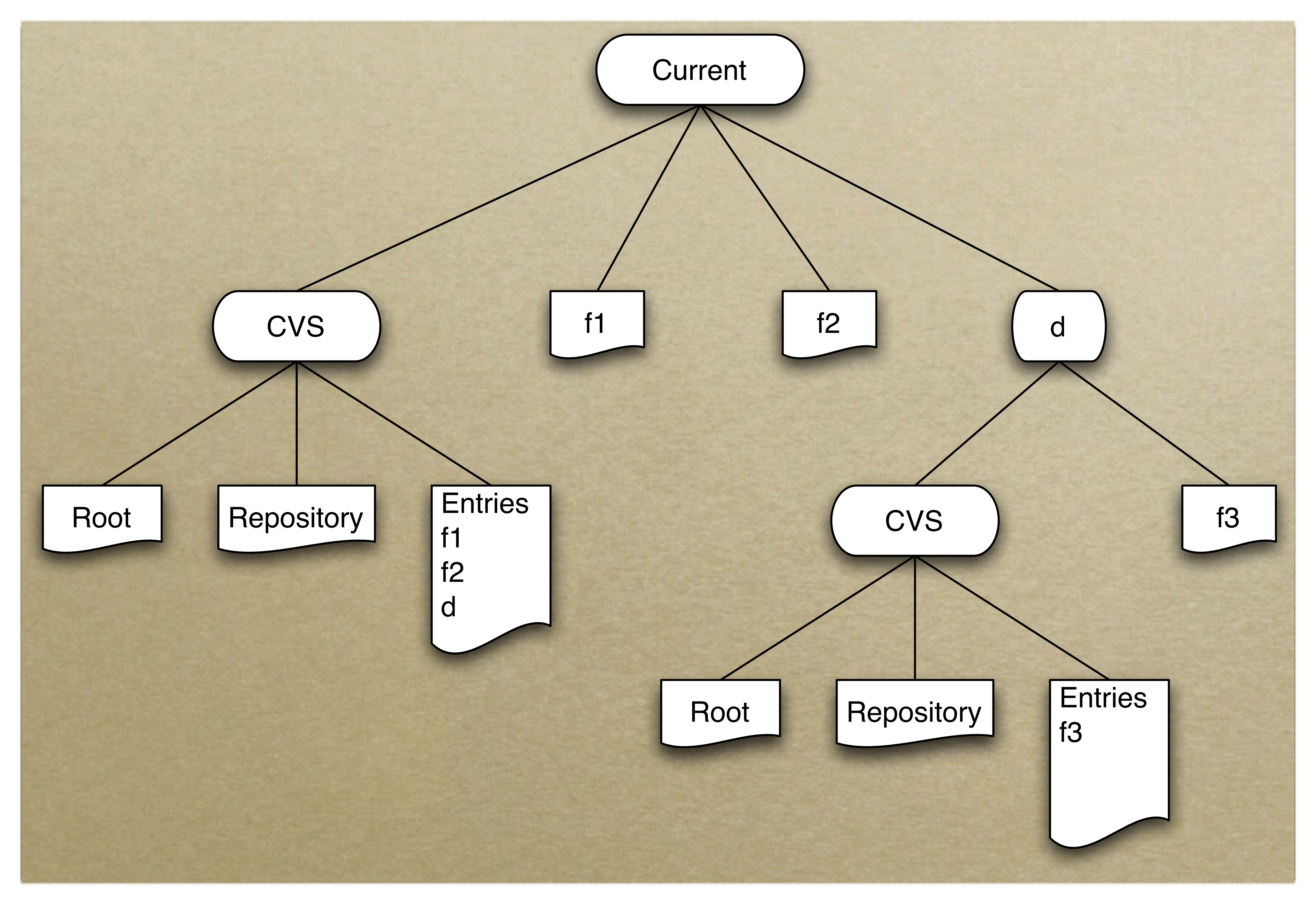
```
ptype pads_website_d = ...
let w :: pads_website_d = "/Users/kfisher/pads/padswebsite/PLConfig.PM"

let numUsers = List.length (users (admin w))
```

Question 2: Can you think of other uses?

Observations

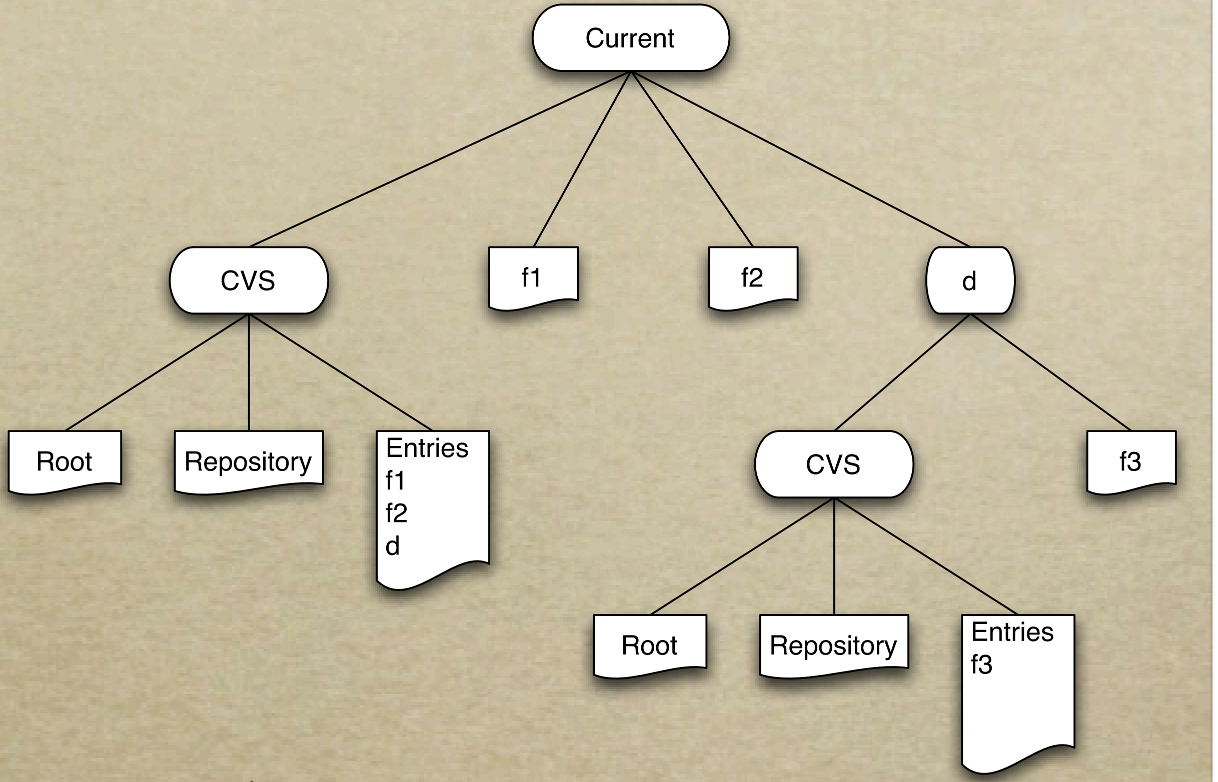
- ***File names*** *sometimes encode extra information.*
- ***Meta-data*** *is important: permissions, owners, groups, create time, modification time, sizes.*
- ***Symbolic links*** *are important.*
- *Files contain information about the* ***structure*** *of other parts of the system.*
- ***Presence*** *and* ***absence*** *information is important.*
- *Want to* ***transform*** *physical rep into logical.*

Example: CVS Directories

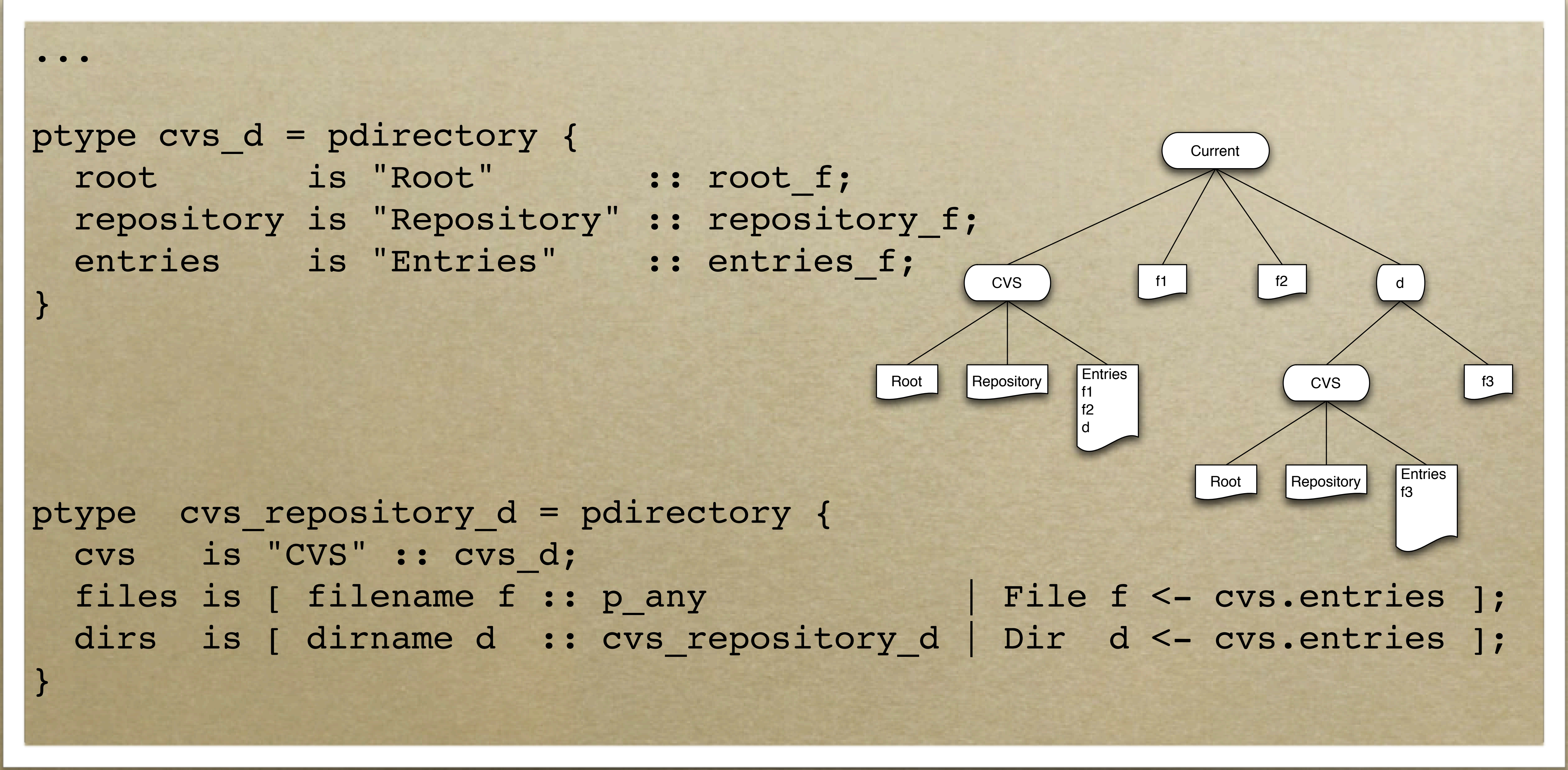


Example: CVS Directories

```
ptype root_f = ...
ptype repository_f = ...

ptype d_entry_t = precord {
  "D/" ;
  dirname :: pstring "/";
  "////";
}

ptype f_entry_t = precord {
  "/";
  filename :: pstring "/";        "/";
  version  :: pint * "." * pint;  "/";
  mod_time :: pdate "/";          "/";
  rest     :: pstring "/";        "/";
}

ptype entry_t   = Dir of d_entry_t | File of f_entry_t
ptype entries_f = psource (entry_t plist)
```



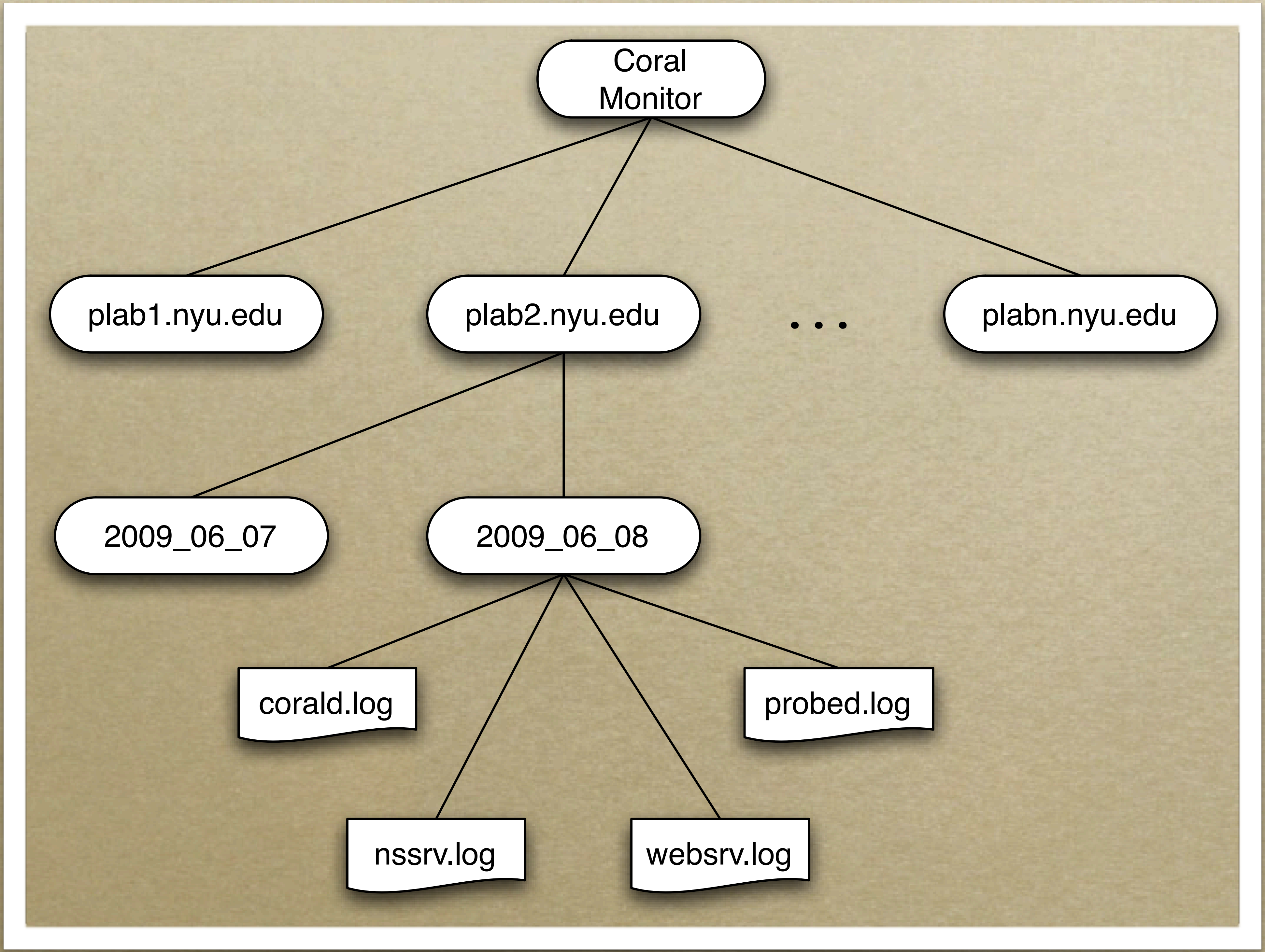
Example: CVS Directories



```
...

ptype cvs_d = pdirectory {
  root       is "Root"       :: root_f;
  repository is "Repository" :: repository_f;
  entries    is "Entries"    :: entries_f;
}
```

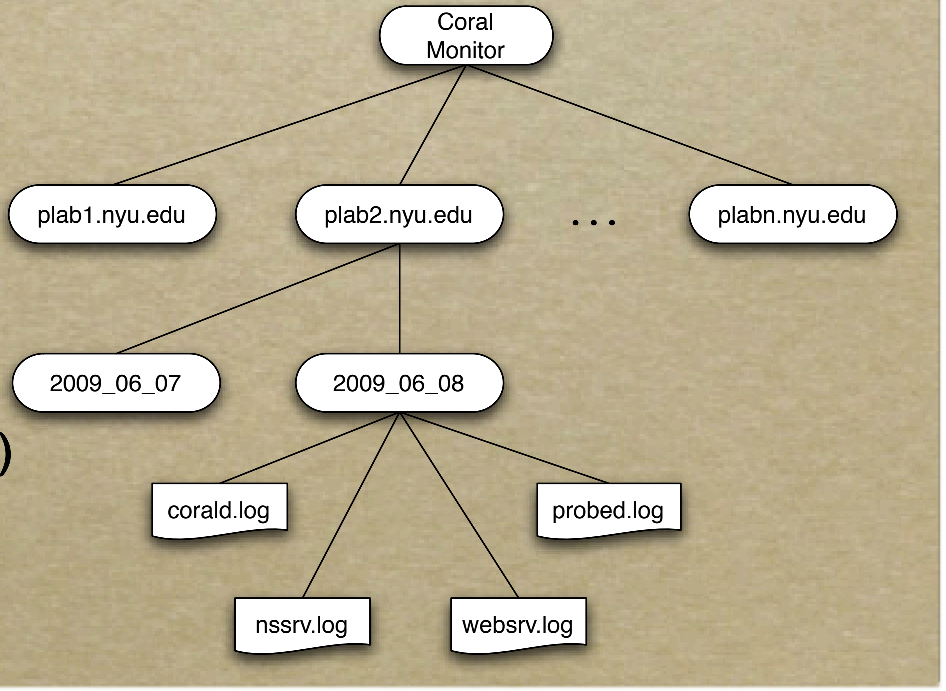
```
ptype  cvs_repository_d = pdirectory {
  cvs   is "CVS" :: cvs_d;
  files is [ filename f :: p_any            | File f <- cvs.entries ];
  dirs  is [ dirname d  :: cvs_repository_d | Dir  d <- cvs.entries ];
}
```

Coral Monitoring System

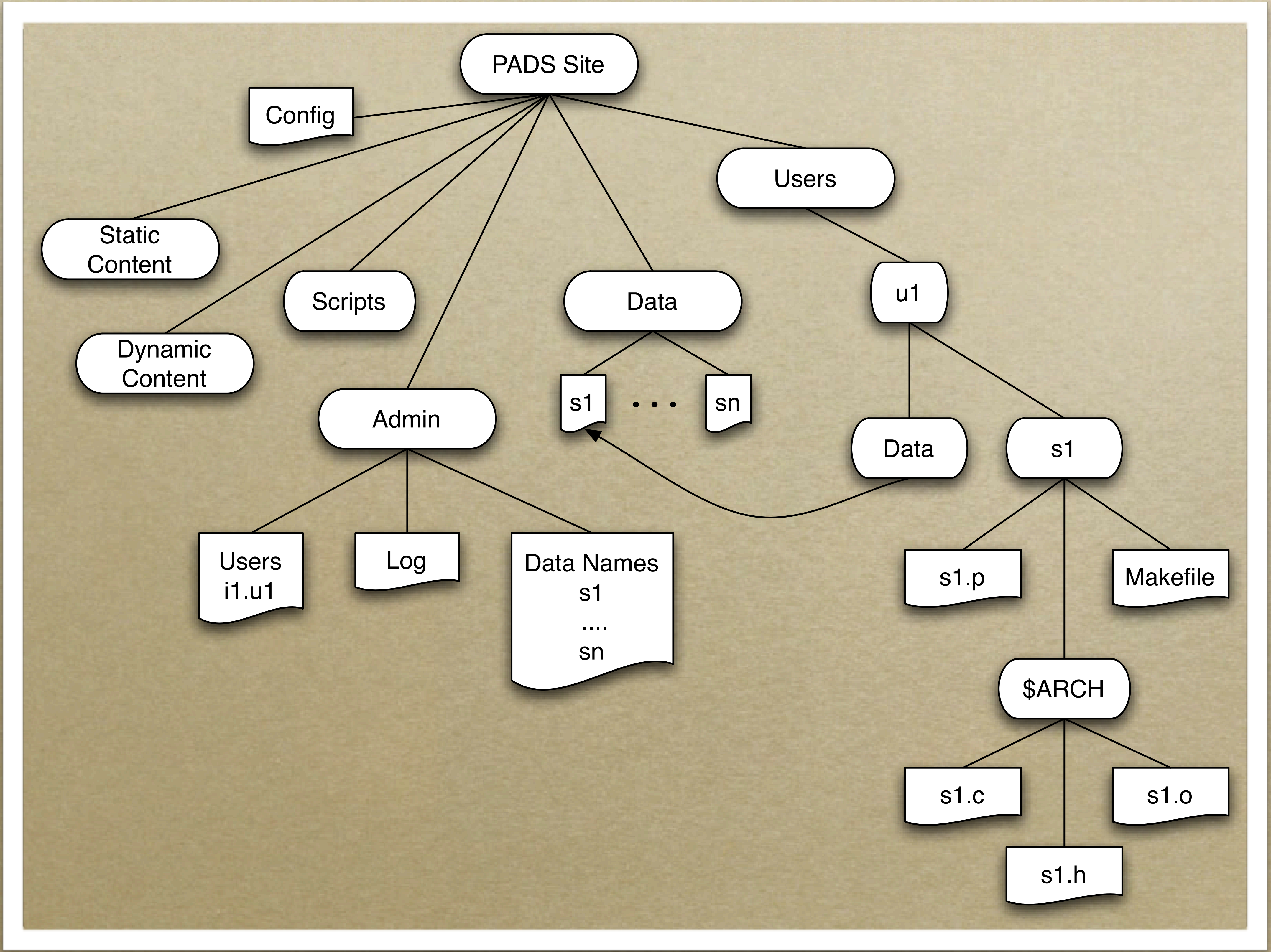


Example: Coral Monitoring

```
ptype corald_t = ...  {- pads description -}
ptype dns_t    = ...  {- pads description -}
ptype web_t    = ...  {- pads description -}
ptype probe_t  = ...  {- pads description -}


ptype host_d(h::phostname, t::pdate) =  pdirectory {
  corald   is "corald.log" :: corald_t  <| timestamp >= t |>;
  coraldns is "nssrv.log"  :: dns_t     <| timestamp >= t |>;
  coralweb is "websrv.log" :: web_t     <| timestamp >= t |>;
  probe    is "probed.log" :: probe_t   <| timestamp >= t |>;
  host :: phostname = h;
  time :: pdate     = t;
 }


ptype coral_d =  pdirectory {
   hosts is (host :: phostname)/(time :: pdate)
         :: host_d(host,time) list;
 }
```



PADS Web Site



Example: PADS Web Site

```
let check p = p == "rwxrwxr-x"

ptype website_d(config::ppath)  = pdirectory {
  c       is config             :: config_f  <| check perm |>;
  static  is static_path c      :: static_d  <| check perm |>;
  dynamic is cgi_path c         :: cgi_d     <| check perm |>;
  scripts is script_path c      :: scripts_d <| check perm |>;
  admin   is static_dst c       :: info_d    <| check perm |>;
  data    is (learn_home c) ++ "/examples/data"
                                :: dataSource_d(sources admin_info)
                                   <| check perm |>;
  users   is tmp_root c         :: users_d(admin_info, data_dir)
                                   <| check perm |>;
}
```

Implementation Plans

- *Intend to build Haskell-based implementation.*
  - *Leverage type-directed programming.*
  - *Leverage laziness in loading structures.*
  - *Possible challenges:*
    - *Scale to large data sets?*
    - *How to manage mutability?*
    - *Will systems programmers be able to cope with unfamiliar language?*

Design Constraints

- *Programmer specifies the file and directory structure in **one place**.*
  - *Precludes one declaration of desired type and another of directory structure.*
- *Generated in-memory type is **data-specific** and contains records and data types.*
  - *Implies that processing directory/file specifications will generate record and data type declarations.*

Questions for the Audience

Question 3: *Given these design constraints,*

- *Can I implement it as an embedded language?*
- *If not, is there an existing framework (Template Haskell?) expressive enough?*
- *If not, what is the best way to extend the language?*

Question Summary

- *Examples where a directory description might be useful.*
- *Possible uses for directory descriptions.*
- *Suggestions on implementation strategies.*