FINAL: Flexible and Scalable Composition of File System Name Spaces

Michael J. Brim, Barton P. Miller
University of Wisconsin

Vic Zandy
IDA Center for Computing Sciences

ROSS 2011
May 31, 2011

Background: Single System Image (SSI)

Unified view of distributed system resources

- allow applications to access resources as if local
- simplifies development of applications, tools, and middleware

Examples:

- unified process space: BProc, Clusterproc
- unified file space: Unix United
- distributed operating systems: LOCUS, Sprite, Amoeba, MOSIX, GENESIS, OpenSSI, Kerrighed

TBON-FS: SSI for Group File Operations

TBON-FS client views unified file name space

- constructed from independent file servers
- target: SSI for 10k – 100k servers

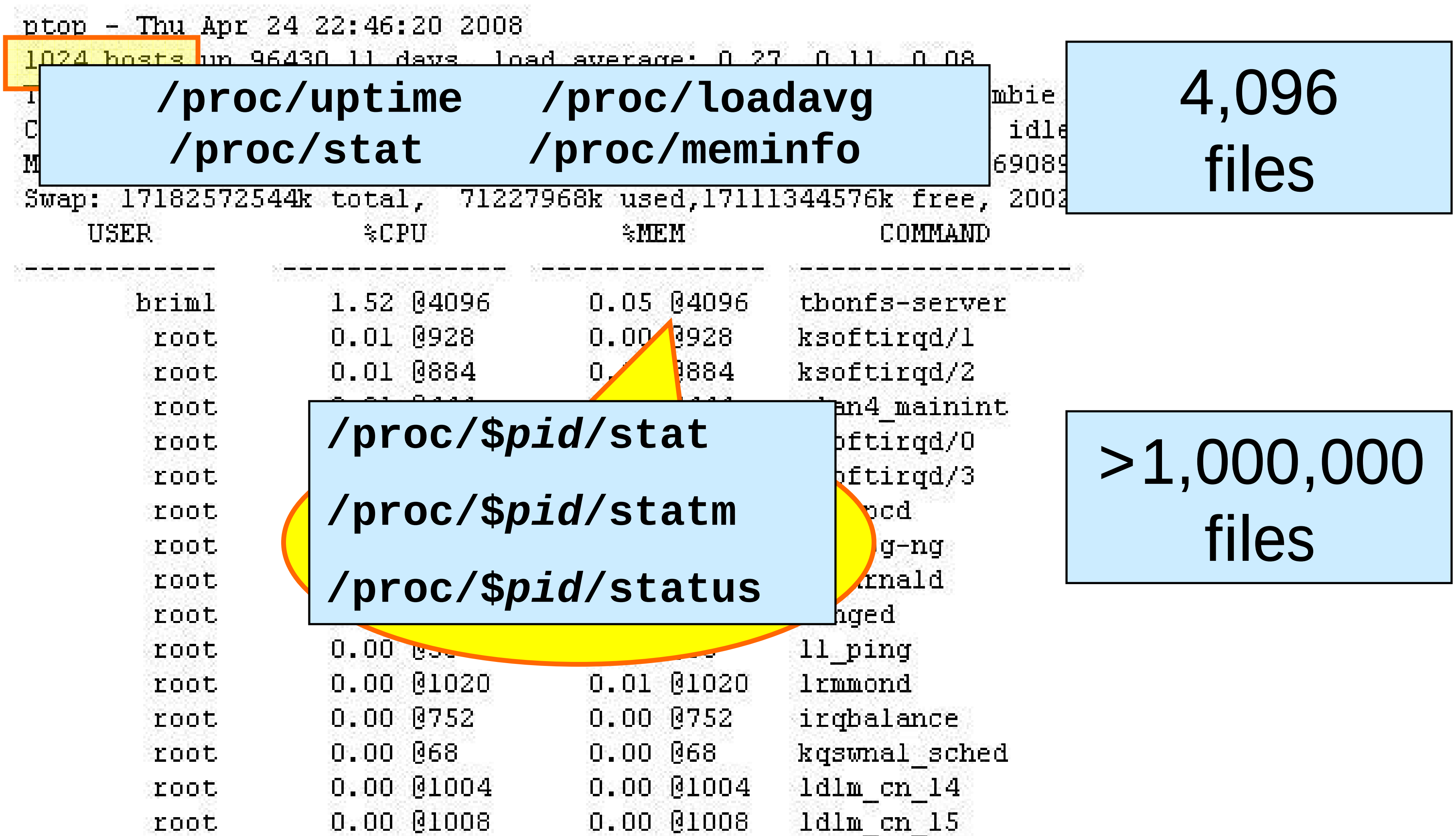
Group file operation idiom: **gopen( )**

- Open files in directory as a group ⇒ **gfd**
- Apply file operations on `gfd` to entire group

TBON-FS employs *Tree-Based Overlay Network*

- provides scalable group file operations via TBON multicast communication and data aggregation

Scalable Distributed Monitoring: ptop



TBON-FS: Problematic Scenario

Prototype used server isolation

- `/tbonfs/$server/…`
- leads to non-scalable group creation

```
mkdir group_dir
foreach member ( /tbonfs/*/path/to/file ) {
    server = …
    symlink $member group_dir/file.$server
}
```

We can do better!!

Custom `ptop` Name Space

Automatic groups:

- host files (4)
- process files (3)

Strategy:

- Create group directories containing files from all hosts/processes

```
/ptop/
     /hosts/
           /loadavg/
                   /host1
                   /…
                   /hostn
           /meminfo/…
           /stat/…
           /uptime/…
     /procs/
           /stat/
                /hostpid1
                /…
                /hostpidn
           /statm/…
           /status/…
```

Goal: Scalable SSI Name Spaces

Let clients specify name space

- name space suited for client needs
- *automatic creation* of natural groups
- easy creation of custom groups

Efficient, distributed name space composition

- avoid traditional SSI scalability barriers of centralization or consensus

Name Space Composition @ Scale

Lots of prior work in name space composition

- mounts and union mounts
- private name spaces for custom views & security
- global name spaces that aggregate resources

Ill-suited to composing 10k – 100k spaces

- inefficient composition
  - pair-wise operations (e.g., mount)
  - fine-grained directory entry manipulation
- inflexible structure and semantics

Desired Composition Properties

Flexibility: describe a wide range of compositions

Clarity: simple, intuitive semantics

Efficiency & Scalability:

- avoid centralized, pair-wise composition
- use TBON for distributed composition

File Name space Aggregation Language

Two primary abstractions

1. Tree: a file name space
2. File Service: access to local/remote file system(s)

A set of tree composition operations

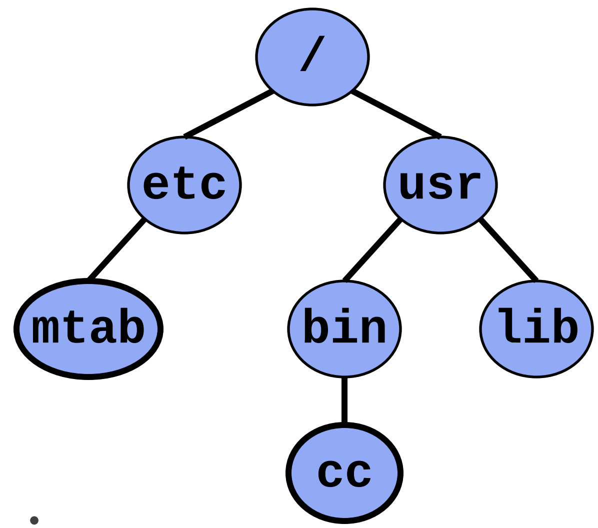
- get or prune a sub-tree
- path extend a tree
- combine two or more trees

FINAL Abstractions: Tree

Assume name spaces are traditional directory trees

Name Space Abstraction

- rooted tree of named vertices
- edges for parent dir, children



Tree is essentially a name space *view*

- independent of underlying file service name spaces
  - each vertex associated with `(service, path)`
- views are immutable

FINAL Abstractions: File Service

File service provides:

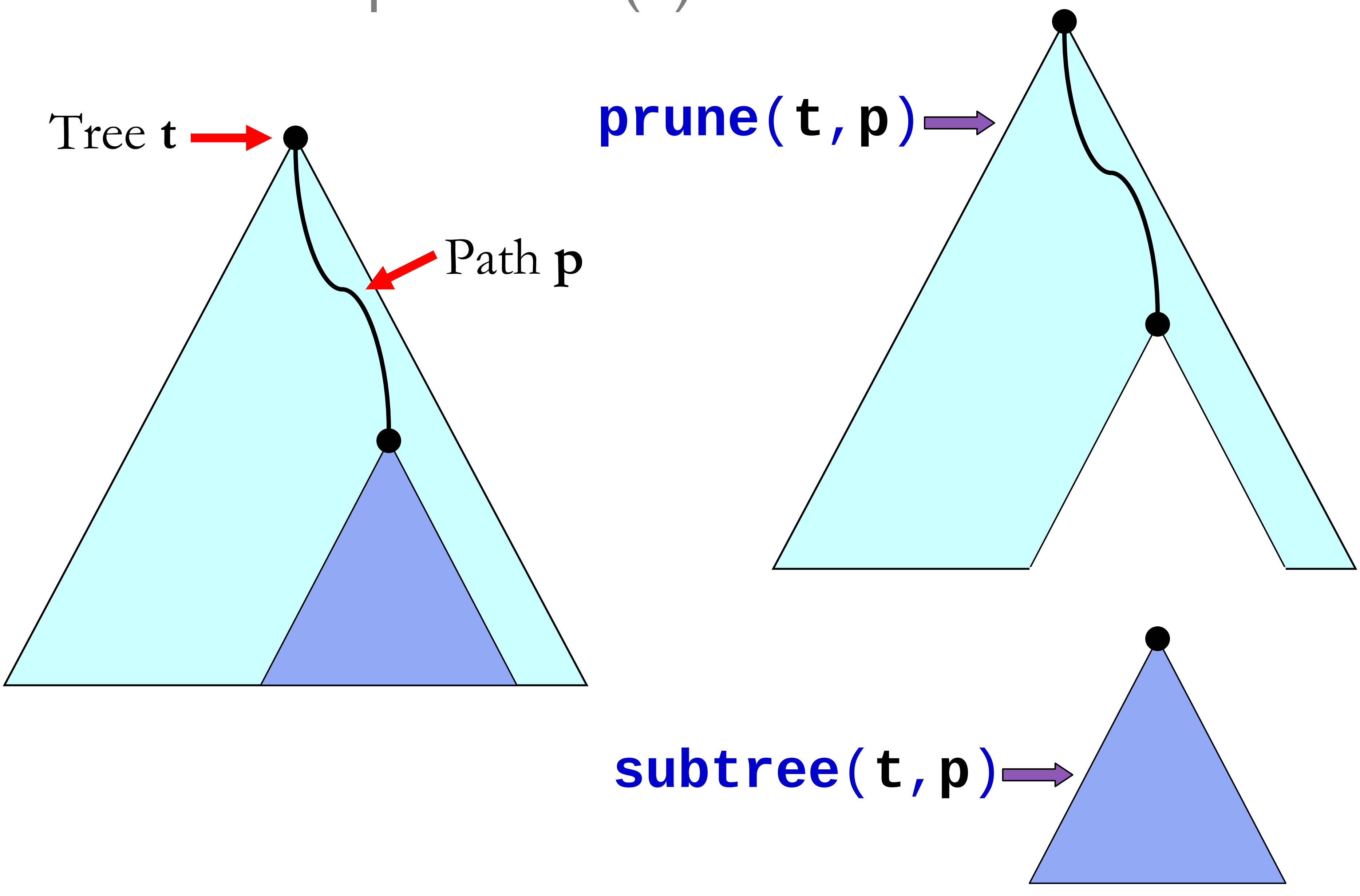
- access to a physical name space
- operations on files in that name space
    - e.g., `stat()`, `open()`, `read()`, `write()`, `lseek()`

Define service instance by name, returns snapshot view

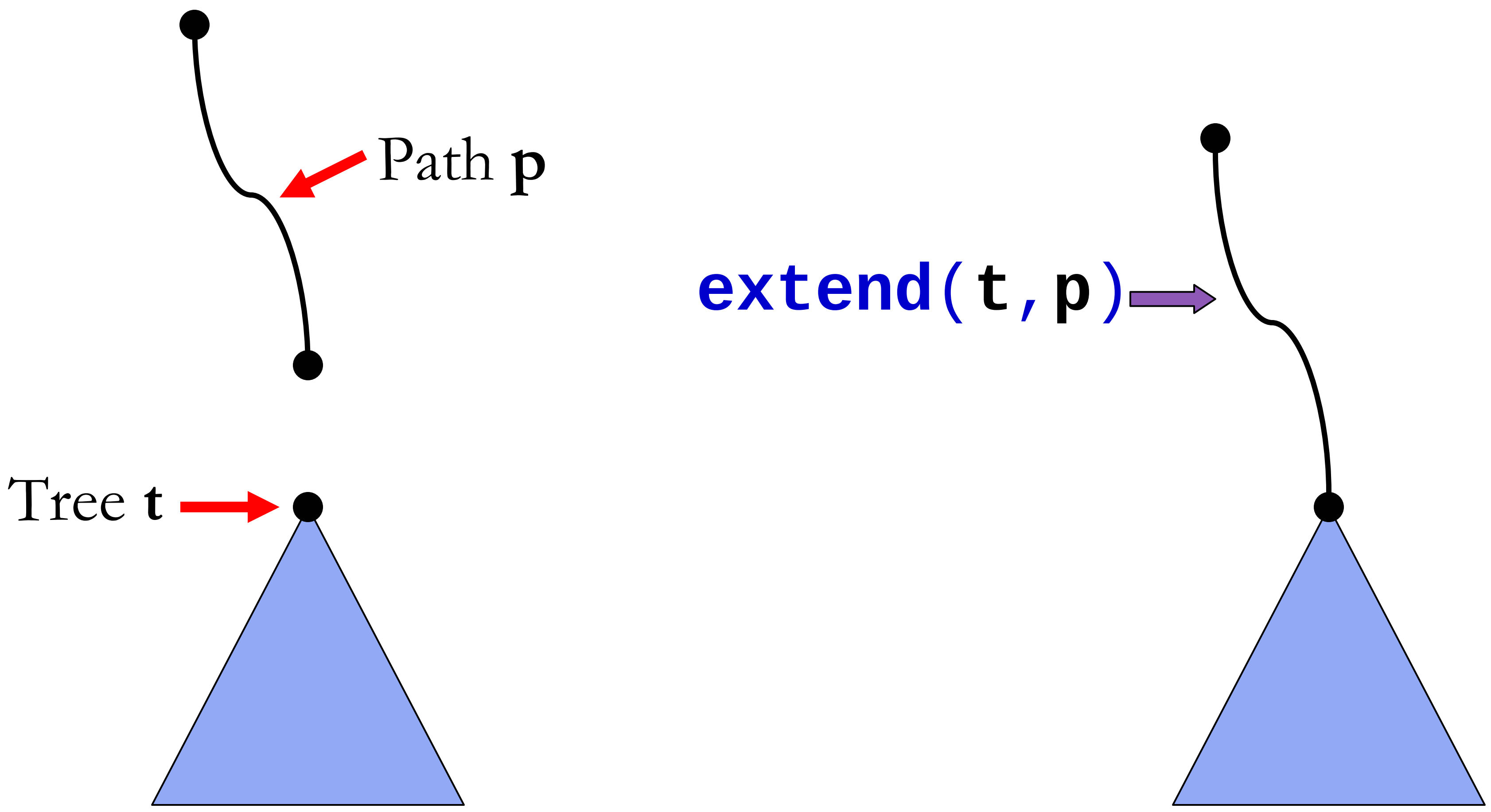
- key-value pairs for service options
- Examples:

```
local()
nfs( host=server, mount=path )
9P( srv=file, mount=path )
```

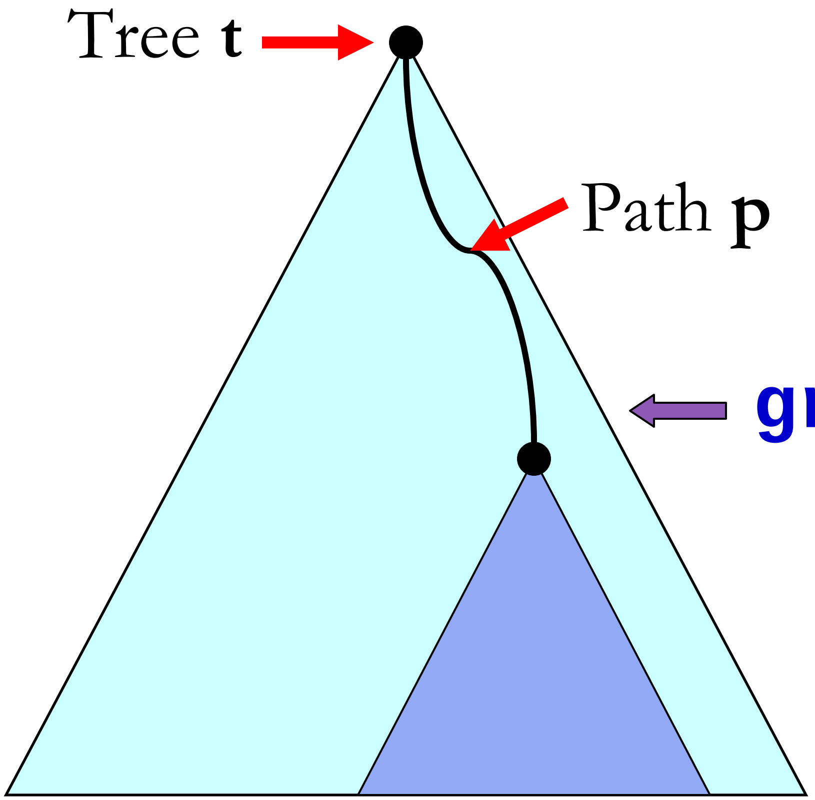
FINAL Path Operations (1)



FINAL Path Operations (2)



FINAL Composition Operations (1)

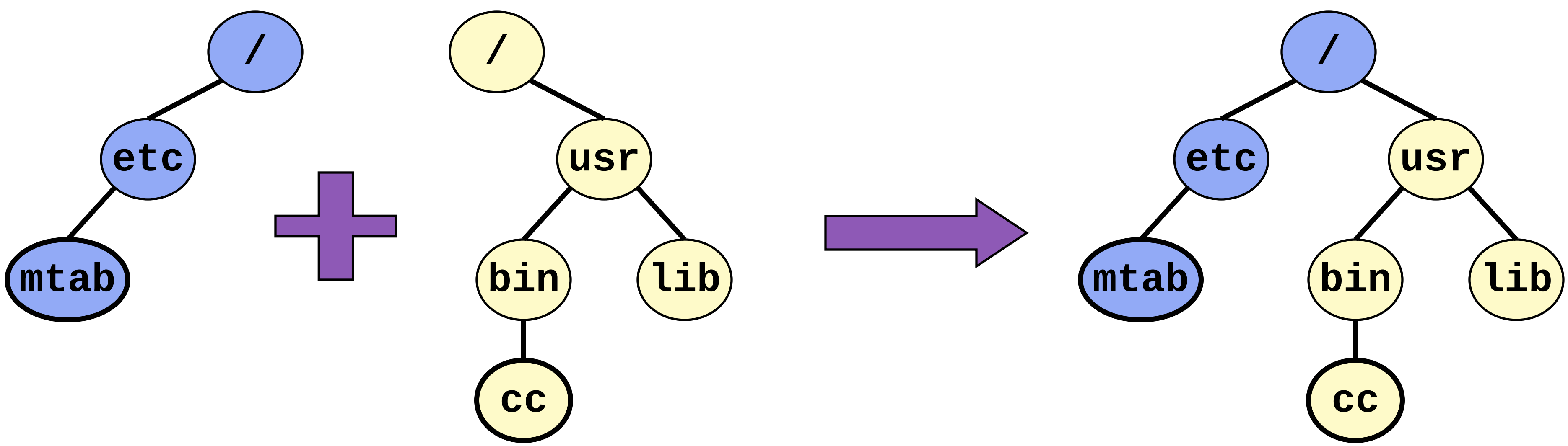



```
graft( prune(t,p),
       subtree(t,p),
       p )
```

FINAL Composition Operations (2)

`merge( {Tree_k}, conflict_fn )`

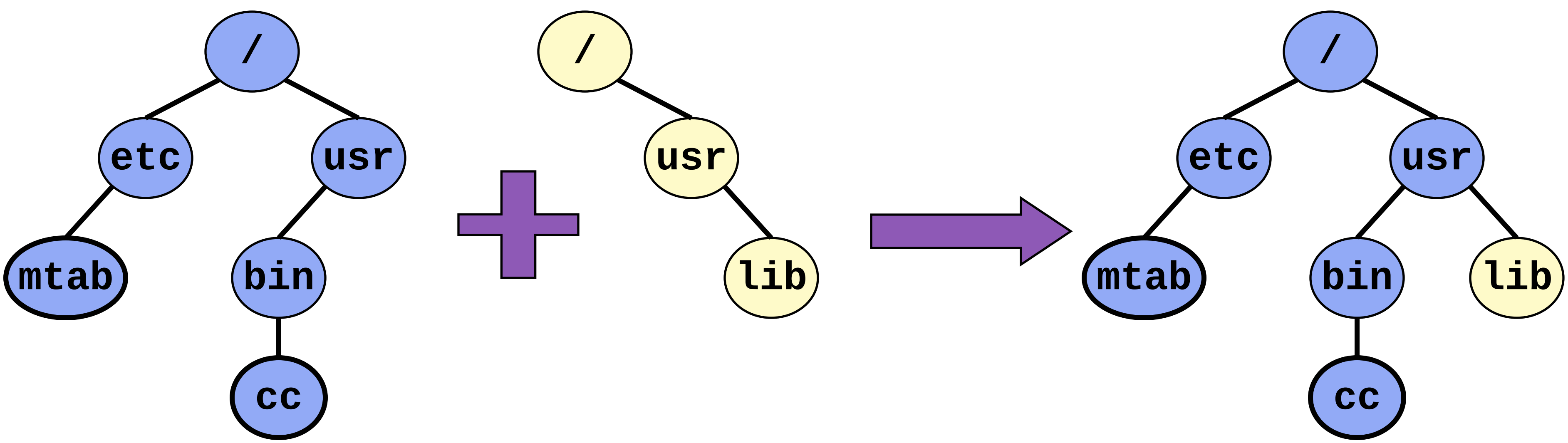
- Deep merge of all trees in input set
- Conflict function called with vertices sharing same path, returns vertices to add to result tree



FINAL Composition Operations (3)

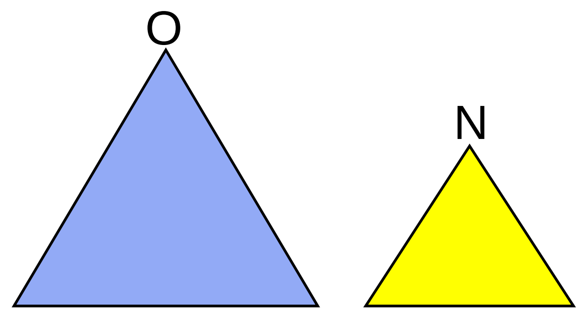
$$\texttt{merge(\{Tree}_k\texttt{\}, overlay)}$$

- Precedence to first tree containing shared path



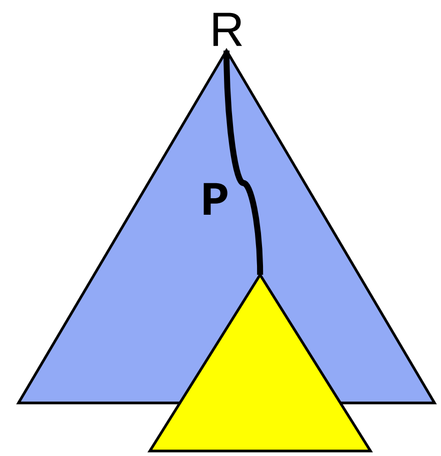
Composition Examples: OS mounts

O : original name space
N : new file system name space
R : result name space



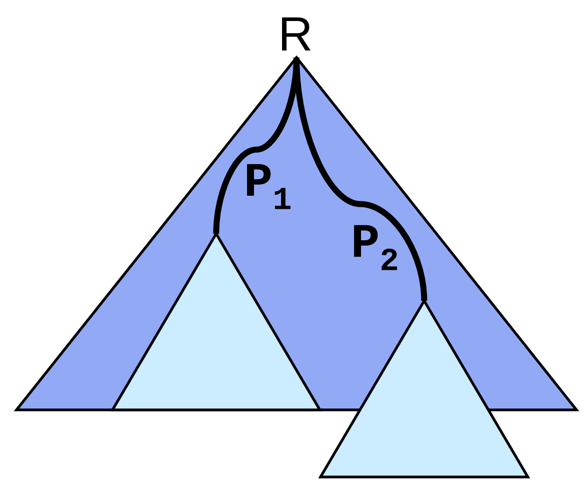
- Standard mount
  - replace sub-tree at path P

```
R = graft( prune(O,P), N, P )
```



- Bind mount
  - make sub-tree at path P_1 also visible at P_2

$$R = \mathtt{graft}(\ \mathtt{prune}(O,P_2),\ \mathtt{subtree}(O,P_1),\ P_2\ )$$

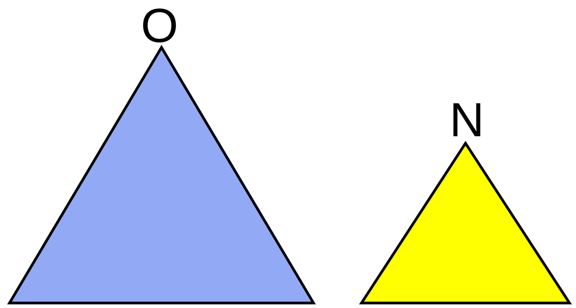


Composition Examples: OS mounts

O : original name space

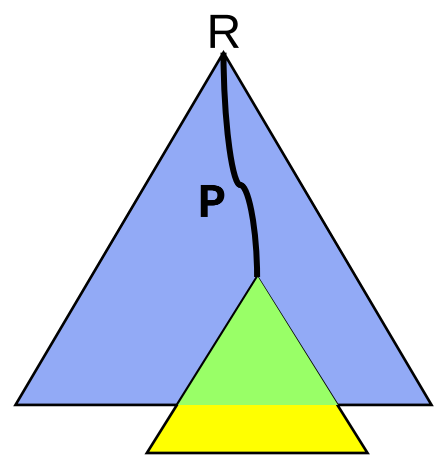
N : new file system name space

R : result name space



- Union mount
  - lay N over sub-tree at path P

```
R = graft( prune(O,P),
           merge({subtree(O,P),N},
                 overlay),
           P )
```



TBON-FS + FINAL

Client mounts views of TBON-FS service

```
graft( local(), tbonfs_svc(final_spec), mountpt )
```

TBON-FS service

- `merge()` all server name spaces
  - conflict function currently hard-coded
- each server name space constructed from FINAL specification given by client
  - specs can depend on local context
  - results in similar name spaces across servers

Example: Automatic File Groups

Client FINAL

```
T = tbonfs_svc(hosts,
               srv_final)
root = graft(local(), T,
             "/tbonfs/config")
```

Server FINAL

```
E = subtree(local(),"/etc")
G = subtree(E,"/group")
P = subtree(E,"/passwd")
GP = merge({G,P},overlay)
root = GP
```

```
/tbonfs/
       /config/
              /group/
                    /host1
                    /…
                    /hostn
              /passwd/
                     /host1
                     /…
                     /hostn
```

Example: Server-local Context

- Handle heterogeneity across servers by hiding name space differences
- Ex: Batch Job System
  - temporary file staging area

Server FINAL

```
T = subtree(local(), "/tmp")
if( T == NULL )
  T = subtree(local(),
              "/scratch")
if( T == NULL )
  T = subtree(local(),
              getenv(HOME))
root = extend(T,"/tmp")
```

```
/tbonfs/
        /tmp/...
```

Example: Cloud Management

- Group distributed hosts by resources provided
  - OS version and CPU type
  - Resource amounts
    - Disk, Memory, # CPUs

<u>Server FINAL</u>

```
L = local()
os = getenv(OSTYPE)
arch = getenv(MACHTYPE)
OA = extend(L, “/$os/$arch”)
root = OA
```

```
/cloud/
      /Linux/
            /x86/
                /path/
                     /host_i
                     /…
                     /host_k
            /x86_64/…
            /ppc32/…
            /ppc64/…
      /WinXP/$arch/…
      /Win7/$arch/…
```

Performance Considerations

Improving efficiency of FINAL operations

- immutable view semantics imply tree copies
  - views implemented as versioned trees
- deep merges can be costly
  - lazy evaluation of specifications as new paths are accessed

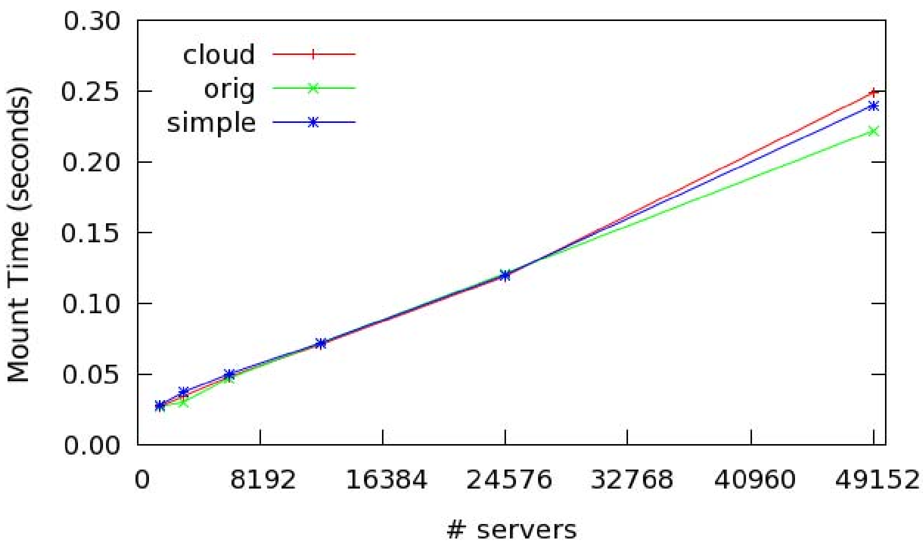
TBON-FS name space caching

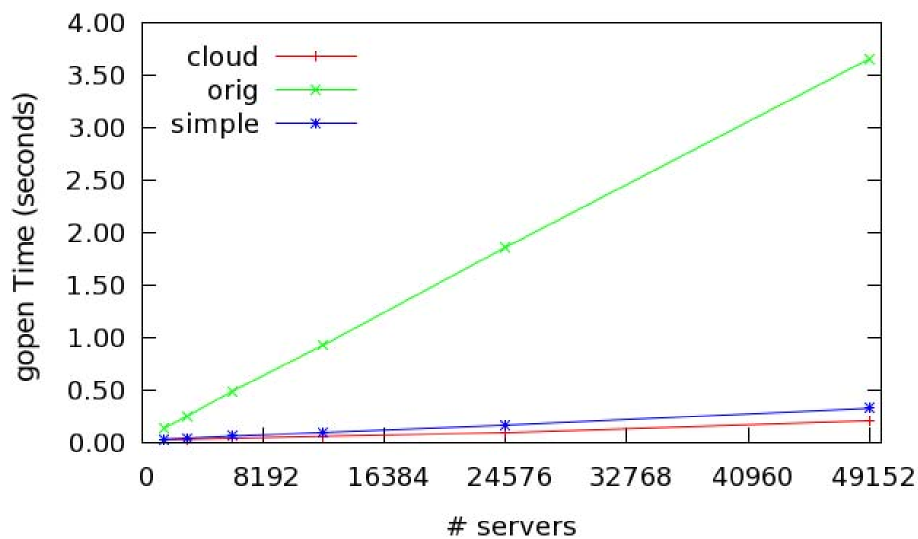
- client only has mount paths
- servers cache accessed portion of name space
- potential for improved lookup latency through caching of merged name space within TBON

Performance Evaluation

Measured:

1. Time to construct name space @ mount
2. Time to gopen( )
3. Effect on group file ops → none, as expected





Conclusion

TBON-FS targets SSI for 10k – 100k servers

FINAL provides *flexibility* to customize name space

- helps improve efficiency of file group definition

FINAL compositions are *scalable*

- use trees to compose trees
- server name spaces constructed in parallel