PERCIPIENCE: ASSOCIATIVE FILE SYSTEMS FOR UNSTRUCTURED DATA RELATIONSHIPS

Presented by: Tony Mason

PHD STUDENT

SUPERVISORS: NORM HUTCHINSON, ALEXANDRA FEDOROVA, ANDREW WARFIELD

DO WE NEED FILE SYSTEMS ANYMORE?

App-based environments, with silos

Proposal: eliminate file systems entirely

- Applications are assigned raw disk space
- Choose best storage model
- Data sharing via *copying*

Observation: common functionality across applications that requires trust are the proper domain of the operating system.

FILE SYSTEM NAMESPACES HAVE NOT EVOLVED

Storage *has* evolved

Two namespace consumers: applications and users

- Applications don’t need hierarchical names
- Applications can embed data schema in their hierarchical names
- Users use names to embed semantic information

Key-Value store

Enhanced data location

MODELS OF FILE SYSTEM NAME SPACES

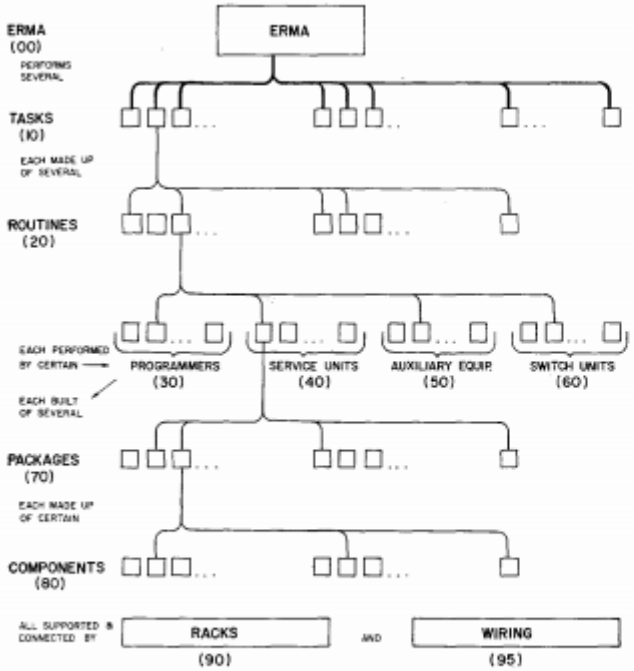


Fig. 2. The ERMA Mark I hierarchical structure

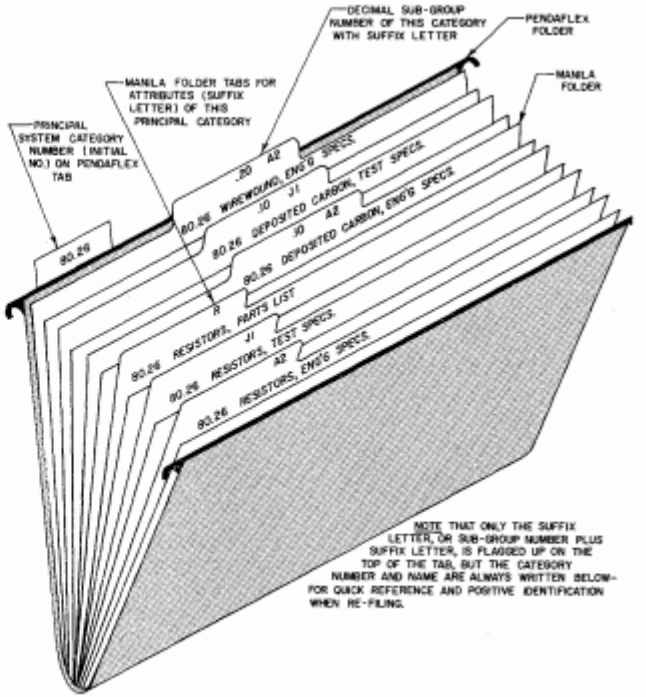
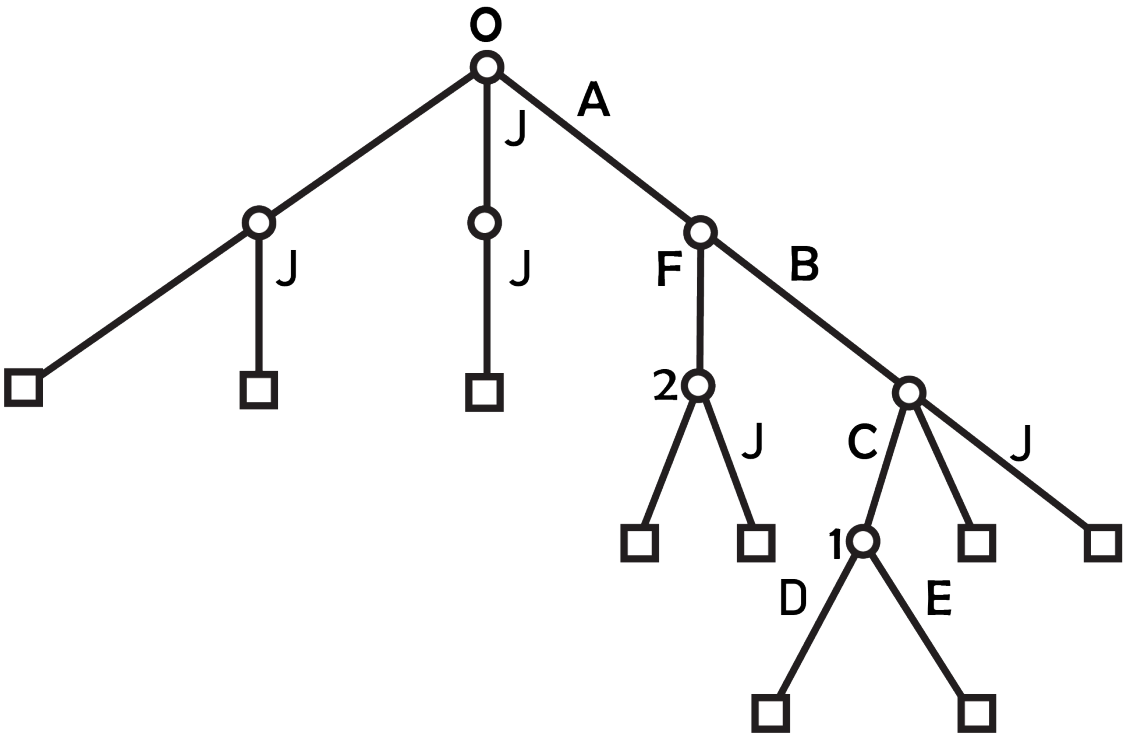
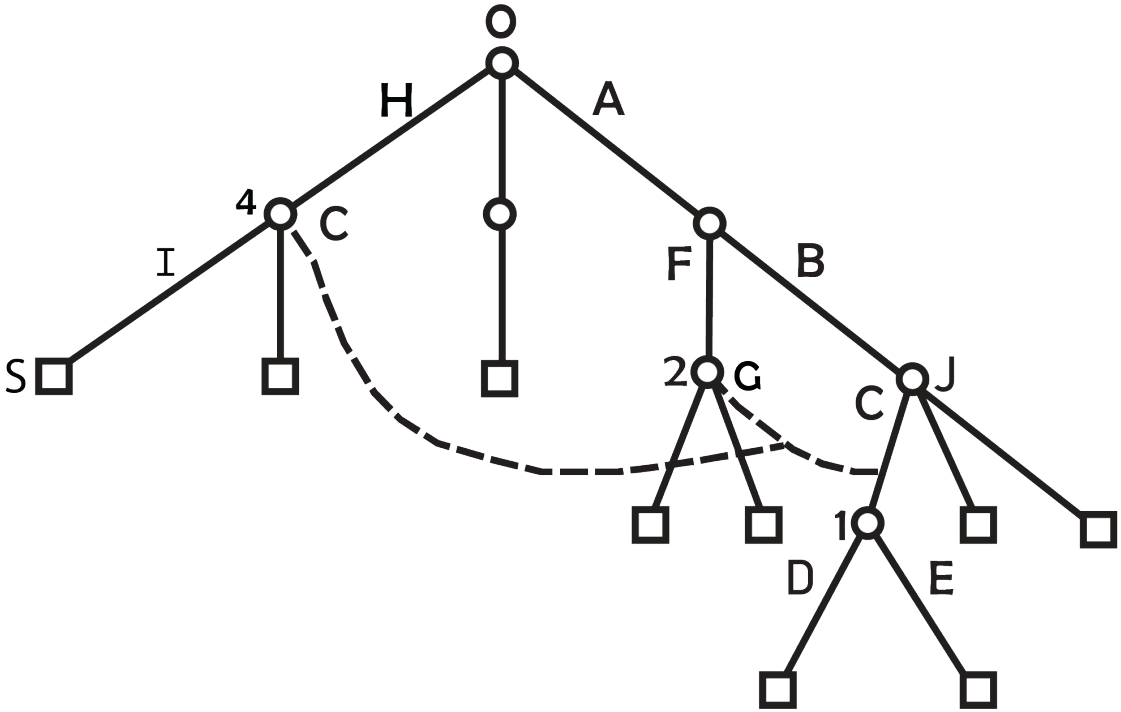


Fig. 3. Records storage and arrangement

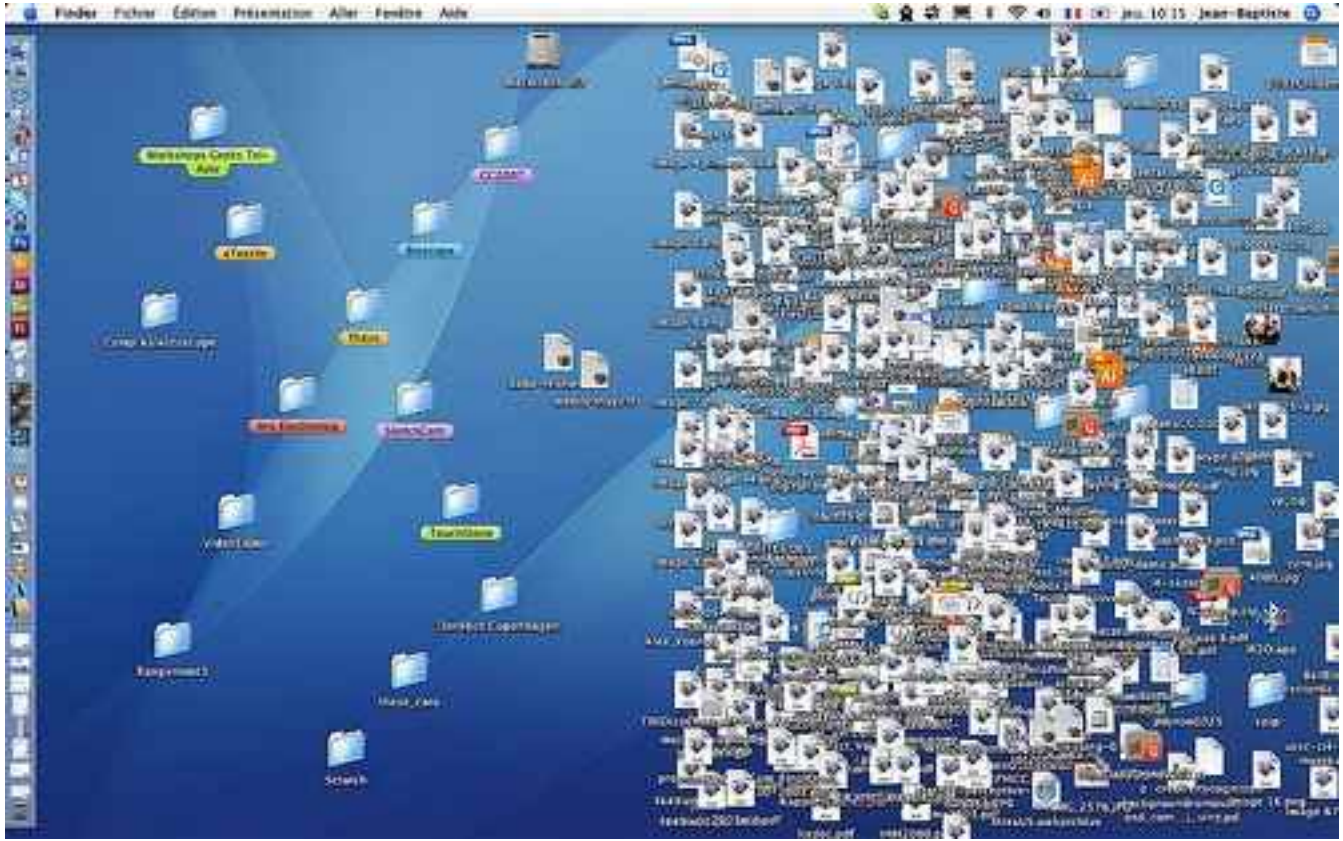
MULTICS: FORMALIZE HIERARCHY



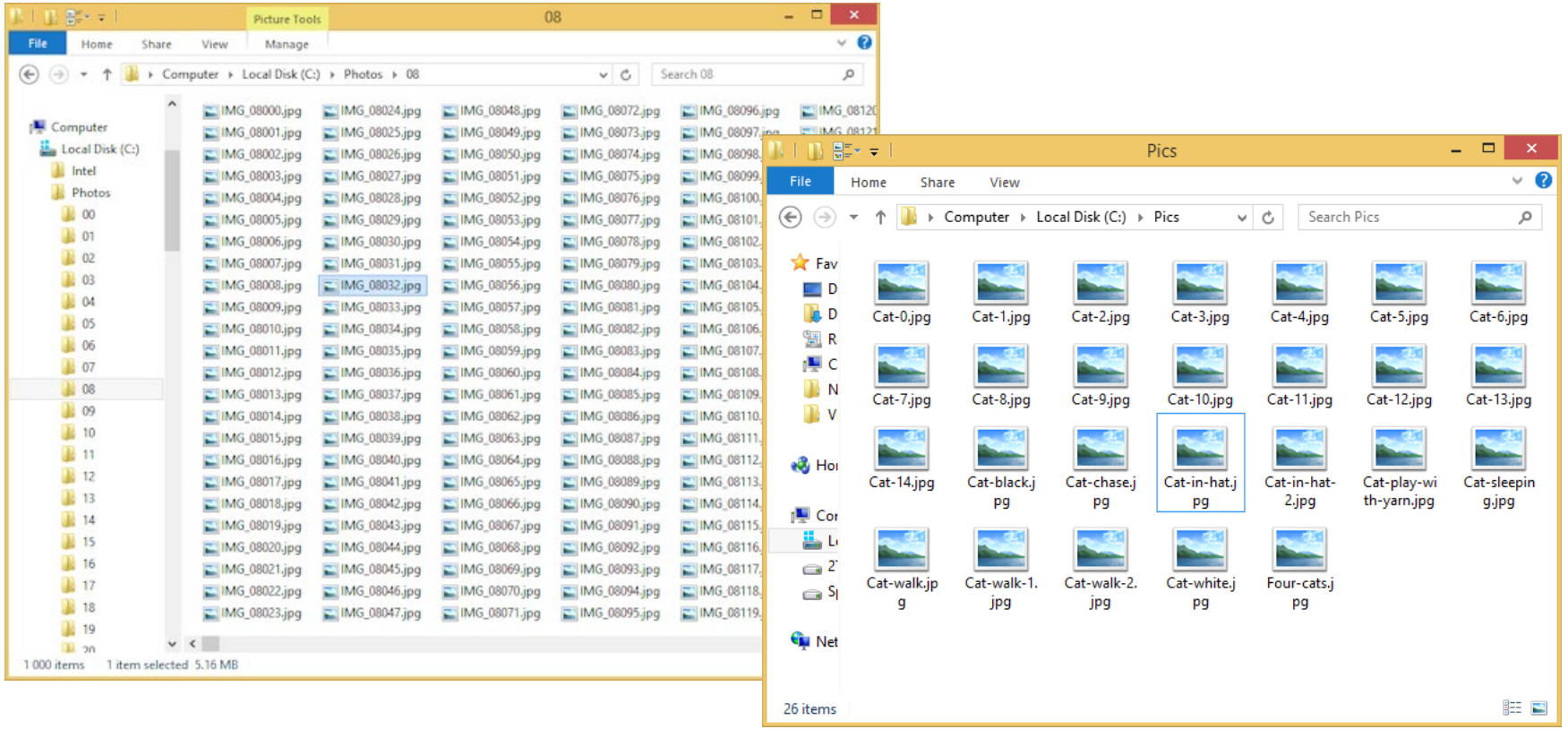
MULTICS – ADD LINKS



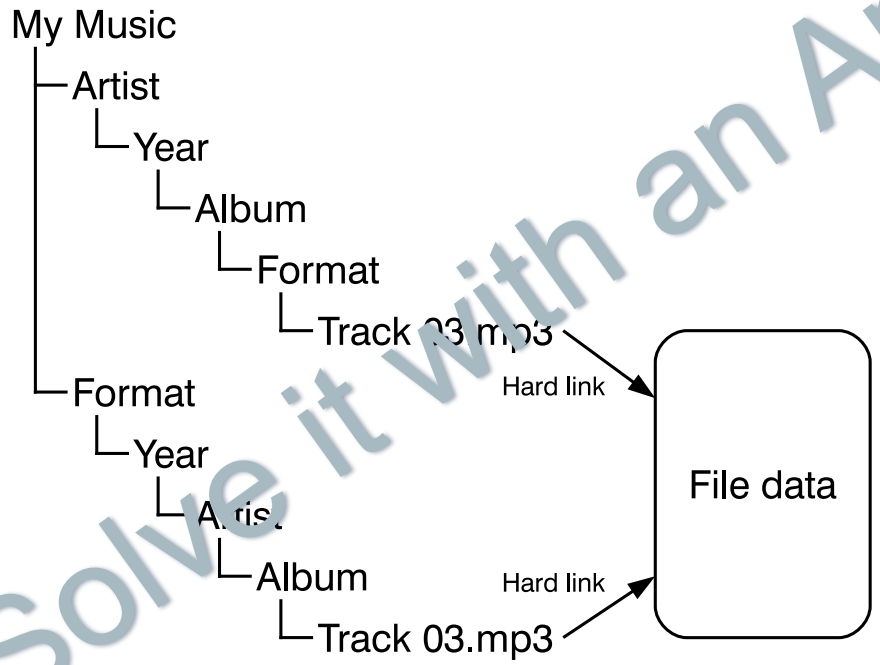
USERS STRUGGLE TO FIND THINGS



FILES WITHOUT USEFUL INFORMATION IN THEIR NAMES



INDEXING INFORMATION WITH MULTIPLE PRIMARY ATTRIBUTES



APPS: STRINGS ATTACHED

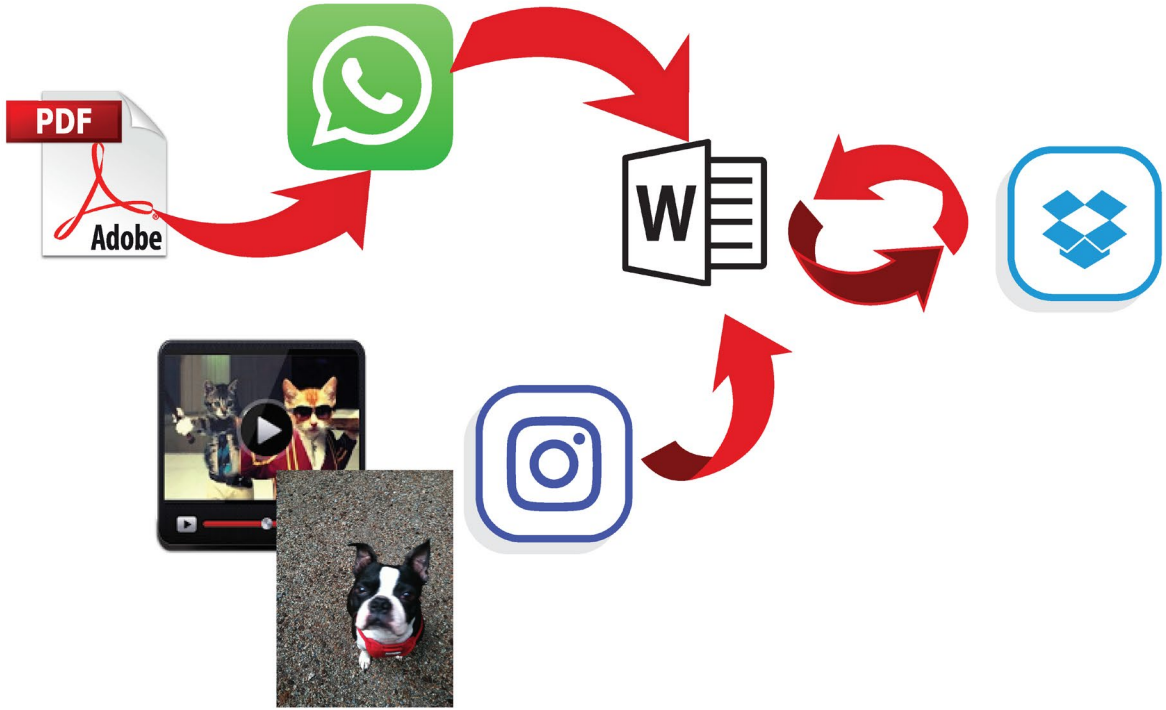
Lock-in

DRM/Rights/Commercial Interests

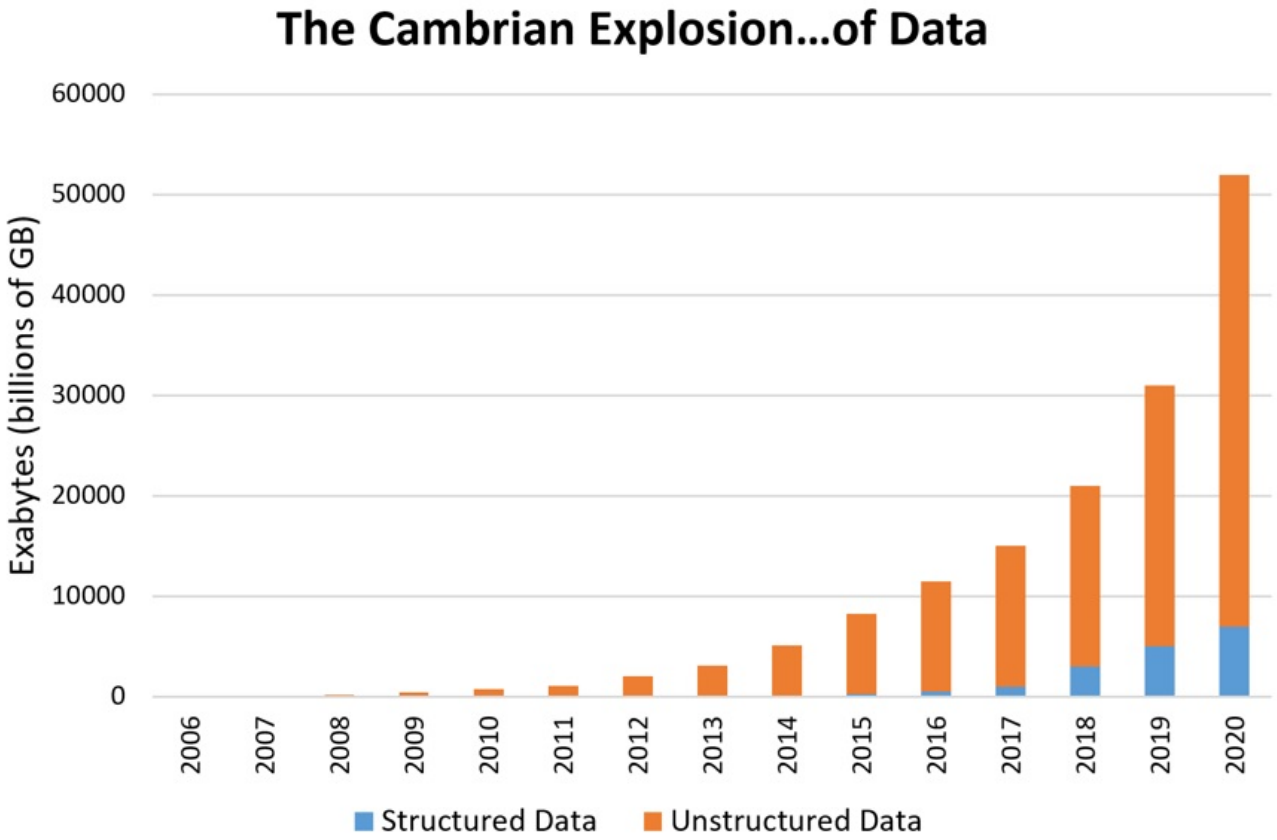
Cloud-driven

Limited cross-app support

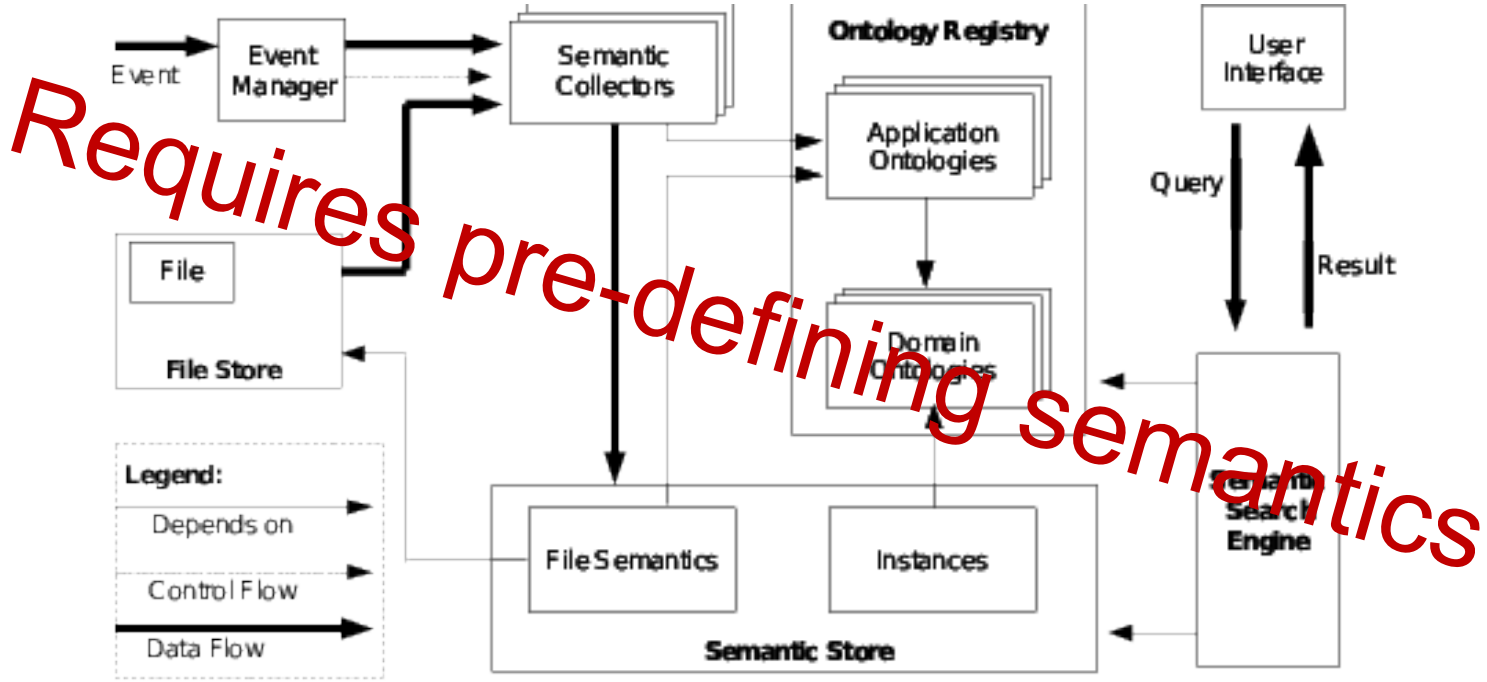
WHICH APP IS RESPONSIBLE?



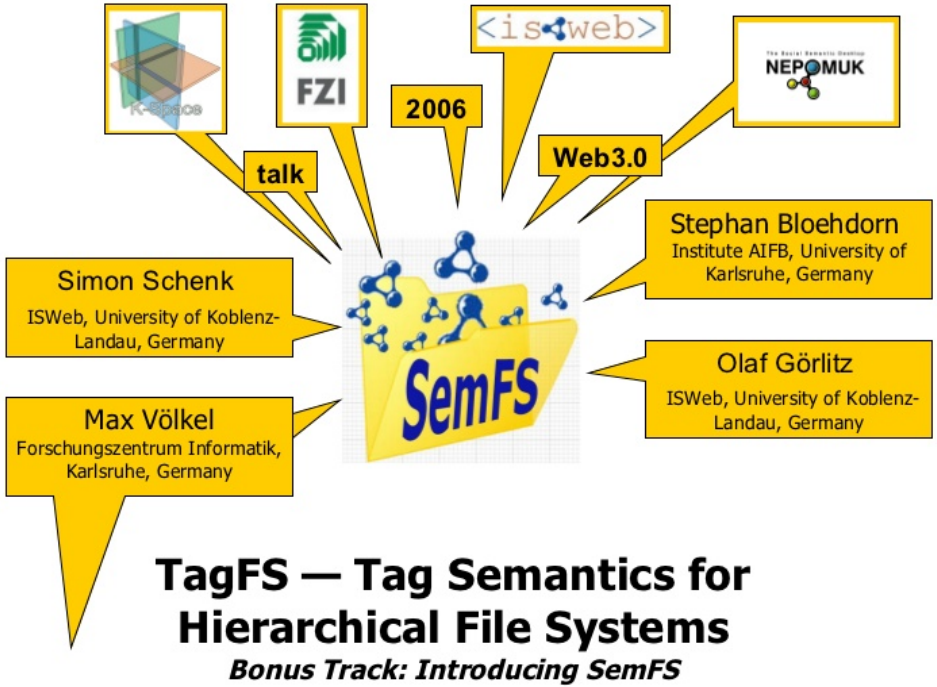
PROBLEM IS INTENSIFYING



POSSIBLE SOLUTION: SEMANTIC FILE SYSTEMS



POSSIBLE SOLUTION: SEMANTIC FS + TAGGING



POSSIBLE SOLUTION: PROVENANCE

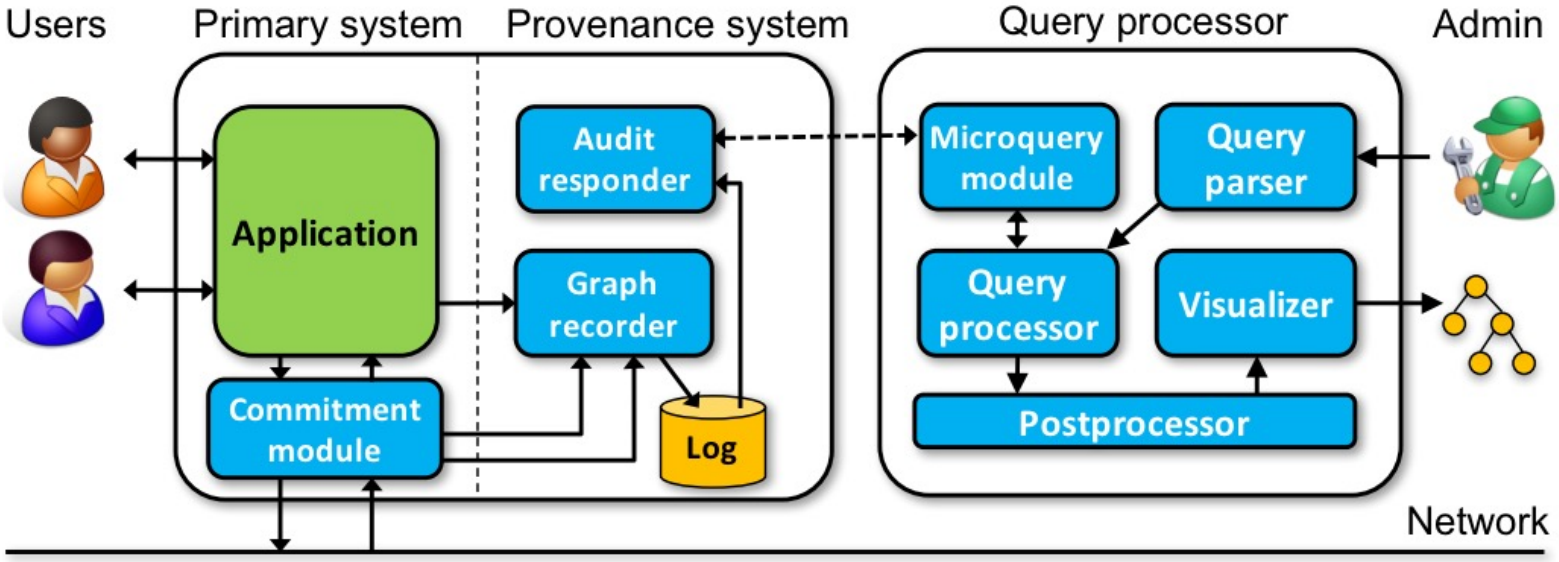
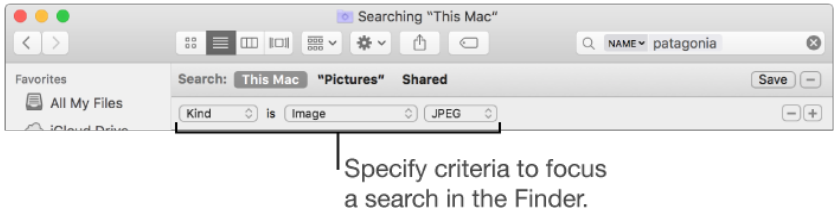
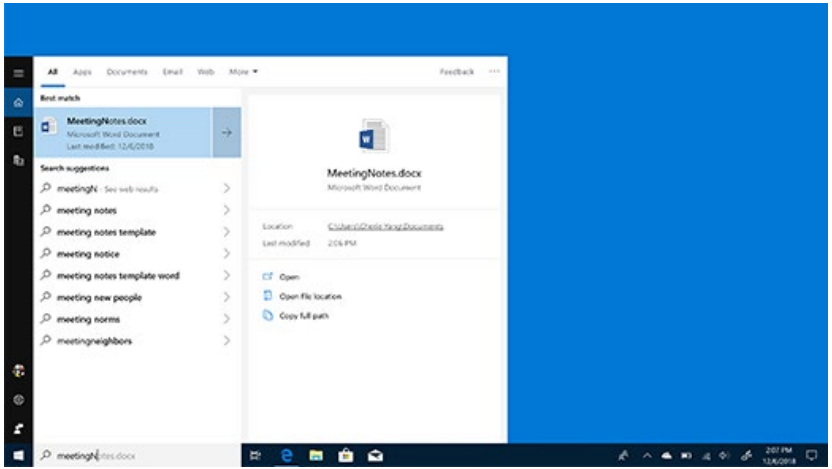


Figure 5: System architecture.

SIMPLISTIC USER SEARCH TOOLS





SOPHISTICATED SYSTEMS SEARCH TOOLS

GREP(1) User Commands GREP(1)

NAME top

grep, egrep, fgrep - print lines that match patterns

SYNOPSIS top

grep [*OPTION*...] *PATTERNS* [*FILE*...]
grep [*OPTION*...] **-e** *PATTERNS* ... [*FILE*...]
grep [*OPTION*...] **-f** *PATTERN_FILE* ... [*FILE*...]

DESCRIPTION top

grep searches for *PATTERNS* in each *FILE*. *PATTERNS* is one or patterns separated by newline characters, and **grep** prints each line that matches a pattern.

A *FILE* of "-" stands for standard input. If no *FILE* is given, recursive searches examine the working directory, and nonrecursive searches read standard input.

In addition, the variant programs **egrep** and **fgrep** are the same as **grep -E** and **grep -F**, respectively. These variants are deprecated, but are provided for backward compatibility.

OPTIONS top

Generic Program Information

--help Output a usage message and exit.

-V, --version
Output the version number of **grep** and exit.

Matcher Selection

-E, --extended-regexp
Interpret *PATTERNS* as extended regular expressions (EREs, see below).

-F, --fixed-strings
Interpret *PATTERNS* as fixed strings, not regular expressions.

FIND(1) FreeBSD General Commands Manual FIND(1)

NAME

find -- walk a file hierarchy

SYNOPSIS

find [**-H** | **-L** | **-P**] [**-EXdsx**] [**-f** *path*] *path* ... [*expression*]
find [**-H** | **-L** | **-P**] [**-EXdsx**] **-f** *path* [*path* ...] [*expression*]

DESCRIPTION

The **find** utility recursively descends the directory tree for each *path* listed, evaluating an *expression* (composed of the ``primaries'' and ``operands'' listed below) in terms of each file in the tree.

The options are as follows:

-E Interpret regular expressions followed by **-regex** and **-iregex** primaries as extended (modern) regular expressions rather than basic regular expressions (BRE's). The re format(7) manual page fully describes both formats.

-H Cause the file information and file type (see stat(2)) returned for each symbolic link specified on the command line to be those of the file referenced by the link, not the link itself. If the referenced file does not exist, the file information and type will be for the link itself. File information of all symbolic links not on the command line is that of the link itself.

-L Cause the file information and file type (see stat(2)) returned for each symbolic link to be those of the file referenced by the link, not the link itself. If the referenced file does not exist, the file information and type will be for the link itself.

This option is equivalent to the deprecated **-follow** primary.

-P Cause the file information and file type (see stat(2)) returned for each symbolic link to be those of the link itself. This is the default.

-X Permit **find** to be safely used in conjunction with xargs(1). If a file name contains any of the delimiting characters used by xargs(1), a diagnostic message is displayed on standard error, and the file is skipped. The delimiting characters include single (`` ' '') and double (`` " '') quotes, backslash (``\''), space, tab and newline characters.

However, you may wish to consider the **-print0** primary in conjunction with ``**xargs -0**'' as an effective alternative.

-d Cause **find** to perform a depth-first traversal.

This option is a BSD-specific equivalent of the **-depth** primary specified by IEEE Std 1003.1-2001 (``POSIX.1''). Refer to its description under *PRIMARIES* for more information.

SEARCH LIMITATIONS

“I’m looking for that document I write last summer after I came back from holiday in Burkina Faco”

“I know this PDF came from an e-mail, show me that e-mail.”

“I moved the linked content for this poster… how do I find it?”

“Show me the files that I access most.”

“Is that document on Dropbox, my laptop, my desktop, Google Drive, or somewhere else?”

“Show me the other documents I accessed while doing my taxes in 2016.”

PRIMITIVE SEARCH: A CURATED INDEXED LIST

The WWW Virtual Library : en · es · fr · zh

The WWW Virtual Library

Quick search:

Agriculture
Irrigation, Livestock, Poultry Science, ...

The Arts
Art History, Classical Music, Theatre and Drama, ...

Business and Economics
Finance, Marketing, Transportation, ...

Communications and Media
Broadcasters, Publishers, Telecommunications, ...

Computing and Computer Science
Artificial Intelligence, Cryptography, Logic Programming, ...

Education
Primary, Secondary, Tertiary, ...

Engineering
Architecture, Electrical, Mechanical, ...

Humanities and Humanistic Studies
History, Languages and Linguistics, Museums, ...

Information and Libraries
Information Quality, Knowledge Management, Libraries, ...

International Affairs
International Relations and Security, Sustainable Development, ...

Law
Arbitration, Forensic Toxicology, Legal History, ...

Natural Sciences and Mathematics
Biosciences, Earth Science, Medicine and Health, Physics, ...

Recreation
Gardening, Recreation and Games, Sport, ...

Regional Studies
African, Asian, Latin American, European, ...

Social and Behavioural Sciences
Anthropology, Archaeology, Population and Development Studies, ...

Society
Peoples, Religion, Gender Studies, ...

About | Contact | Donors | Topics

PAGERANK: AUTOMATE WEB INDEXING

Searching the web: **return an answer**

Searching our own data: **return the** answer

Subtly different

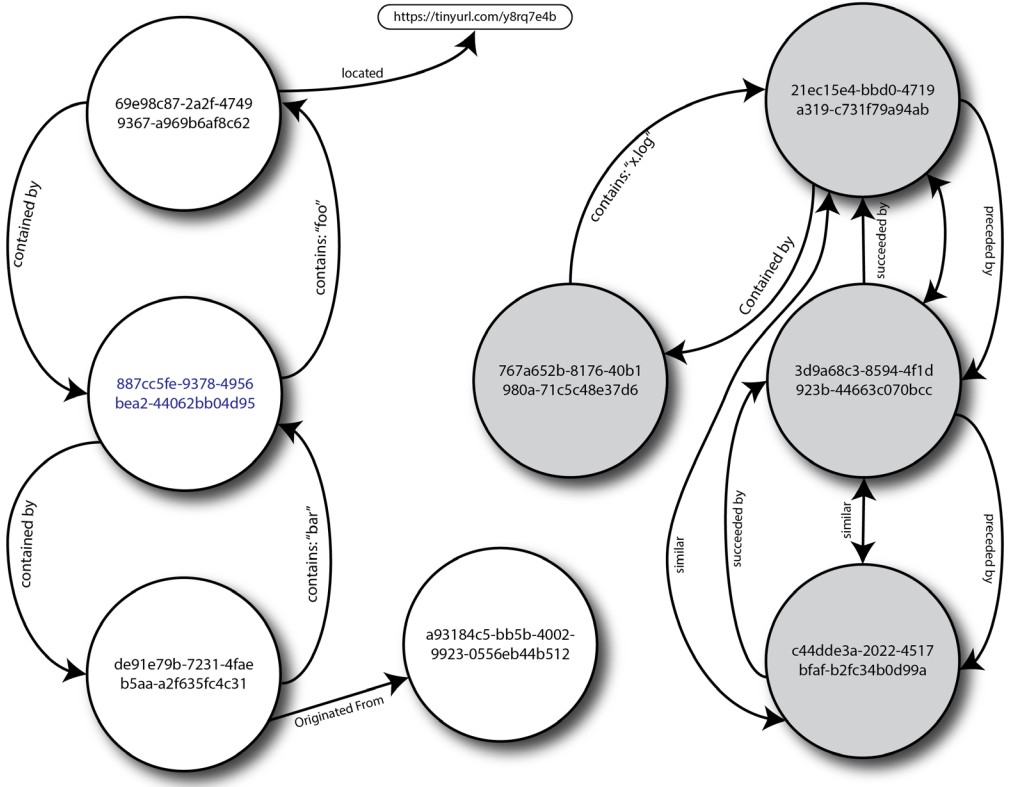
NEW IDEA: FILE RELATIONSHIP GRAPH

Immutable Names (Identifiers)

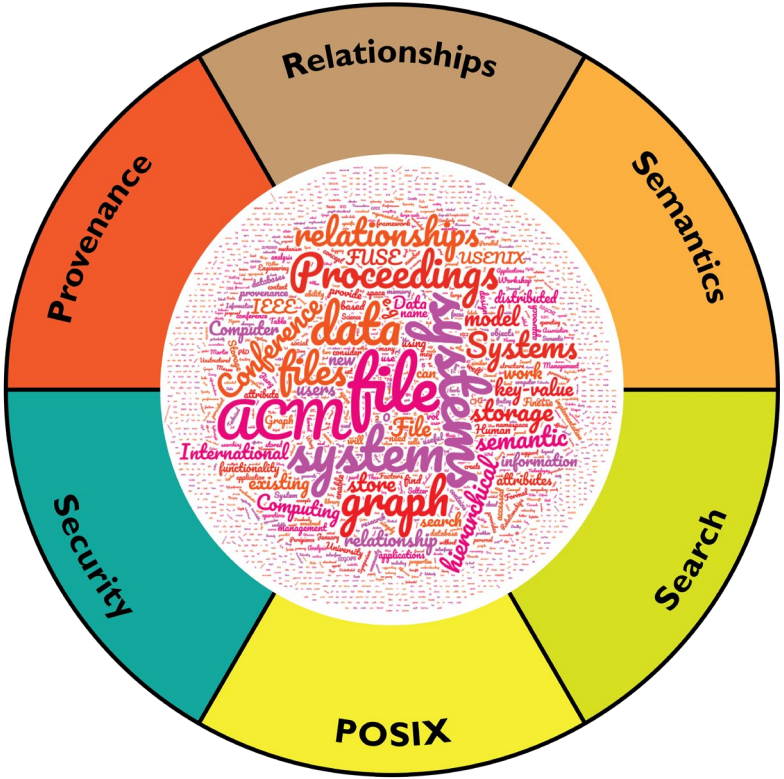
Files are vertices

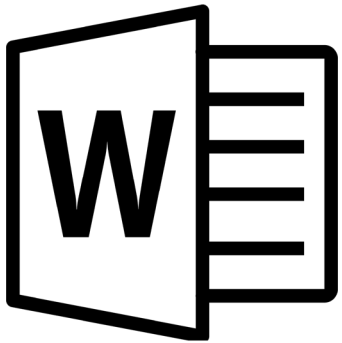
Edges are relationships
- System Defined
- Application Defined

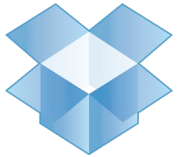
Graph Queries



GOAL: ADDRESS MULTIPLE NEEDS USING GRAPHS







Dropbox

INTERNET SEARCH

Internet Search

- Recent index of content
- Focused on structured documents (HTTP, XML, etc.)
- Use link counts as a proxy for relevance

Goal: to return **a useful answer**

LOCAL SEARCH

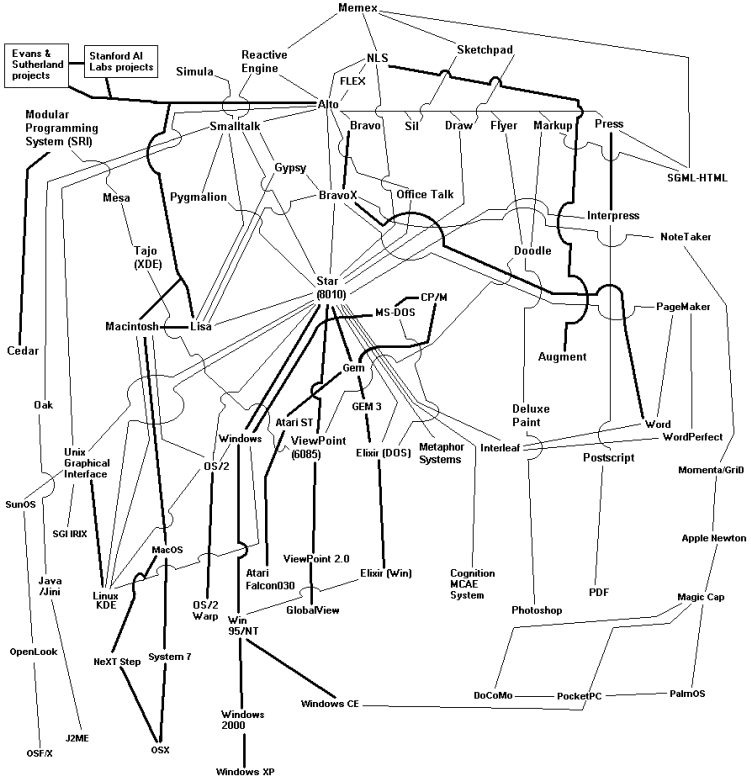
Current:

- Emphasis is on what something *is* (e.g., attribute search)
- Links for “multiple relationships”

Challenges:

- Users do not always remember specific searchable attributes
- Work flows obscure movement
- Detritus is exposed in the file system (“try to hide”)
- Names are mutable, leads to broken links

GRAPHICAL HISTORY OF FILE SYSTEMS



POSIX FILE SYSTEMS

POSIX (IEEE 1003.1) is a formal specification of behavior for “portable operating systems”

- Codified existing UNIX functionality (1988)
- Has evolved somewhat in the intervening years
- Include file system semantics (open, close, read, write)

Benefits:

- Allows good portability of applications across POSIX systems
- Provides a range of useful services for dealing with files and directories
  - Symbolic links
  - Hard links
  - Asynchronous I/O (aio_* - added to POSIX)

POSIX FILE SYSTEMS (2)

Disadvantages

- Expensive behaviors (e.g., path based access validation)
- Difficult to enhance
  - Key/Value storage
  - Semantic search
  - Change journaling
  - HPC extensions: abandoned due to inability to find consensus
- Esoteric features
  - Shared file descriptor semantics (file pointer)

PRIOR WORK

Network File System – “we’re on a network, so we’ll try as best we can”

Andrew File System – “we’re on a network, we’re going to make you think we’re local”

Tiger (and others) – “we’re focused on media, we don’t care about POSIX”

UFO – “let’s add extensions at user mode for HTTP and FTP support”

File System Toolkit (and others) – “let’s extend things to overcome POSIX limitations”

Google File System – “we can’t work with POSIX so we’ll just use a library”

Sedar – “let’s add semantic information with deep archiving, POSIX access via an NFS file server”

KBDBFS – “Berkeley DB inside the kernel, with a POSIX shim on top of it”

CRUISE – “how to get 1PB/s in an MPI (HPC) environment – don’t use POSIX”

FILE SYSTEMS ARE CHALLENGING

Kernel level file systems are notoriously difficult

- Non-uniform VFS layer
  - Original motivation: NFS
  - Different UNIX systems have different VFS (or no VFS layer at all)
  - Non-UNIX/Linux systems do not have VFS (or something else)
- Challenges
  - Highly parallel
  - Asynchronous I/O models
  - Complex edge conditions
  - Multi-year efforts for experienced teams

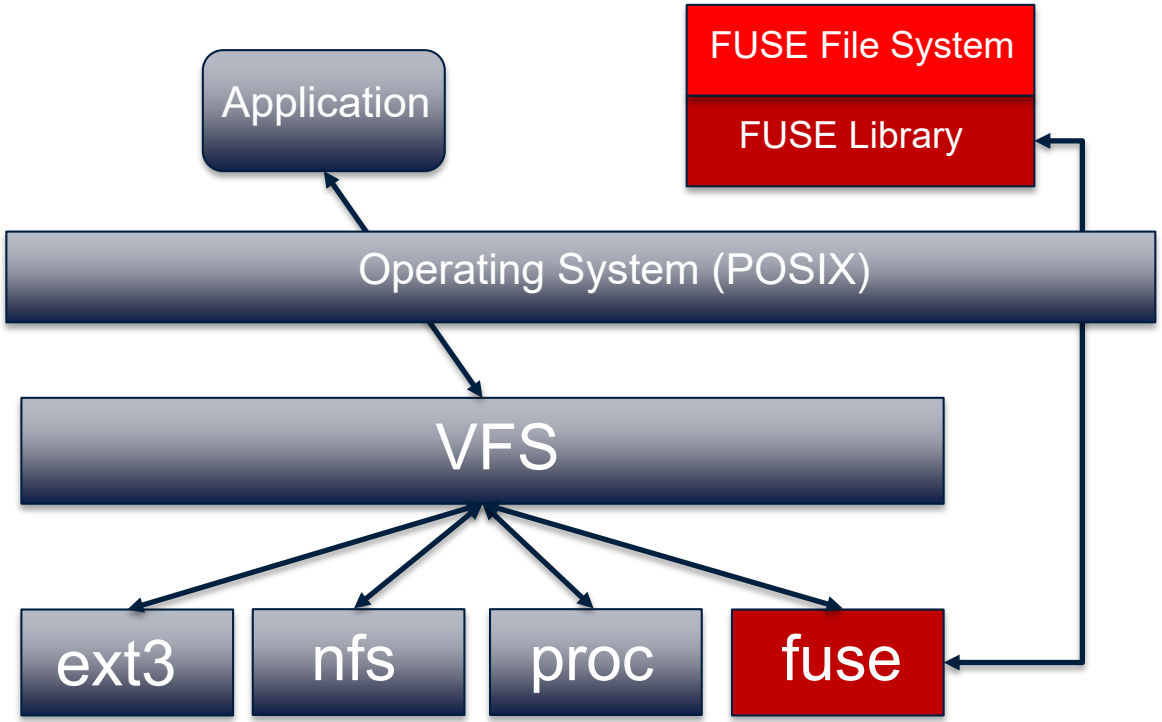
FILE SYSTEMS IN USER SPACE

FUSE is a kernel level reflector

- /dev/fuse

FUSE library interacts with kernel

FUSE file system implements VFS-like interface (defined by FUSE library)



POSIX + EXTENSIBILITY

POSIX yields strong benefits for application compatibility

Extensibility is important to surface enhanced functionality

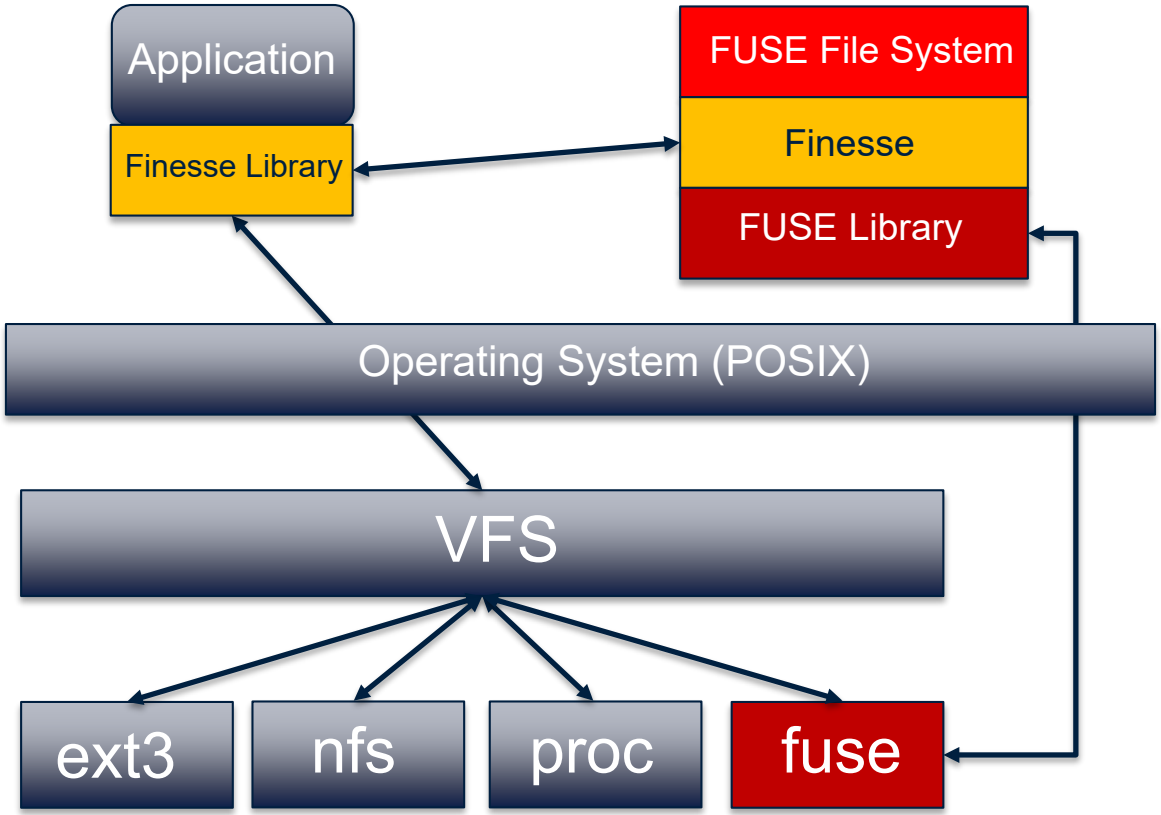
- Provides evolutionary path
- Permit enhancing file system semantics

FUSE FILE SYSTEMS

FUSE is a popular model for constructing file systems

- Advantages:
  - Easier to implement
  - Many languages to support FUSE
  - Portable (Linux, UNIX, Mac OS X, Windows)
- Disadvantages:
  - Performance
  - Focuses on (weak) POSIX semantics (“good enough”)

FINESSE



Finesse:

- Augment FUSE
- Permits extension without sacrificing compatibility

Application:

- Direct linked
- Shared library “override”
- Enhanced libc

FINESSE BENEFITS

FUSE file sytems work without alteration

- No changes are required for some benefits (perf)
- Changes are *permissible* without kernel level (VFS) changes

FUSE itself can be made faster

- Permits kernel bypass for some operations
- Allows enhanced data sharing (e.g., directories)

EVALUATION

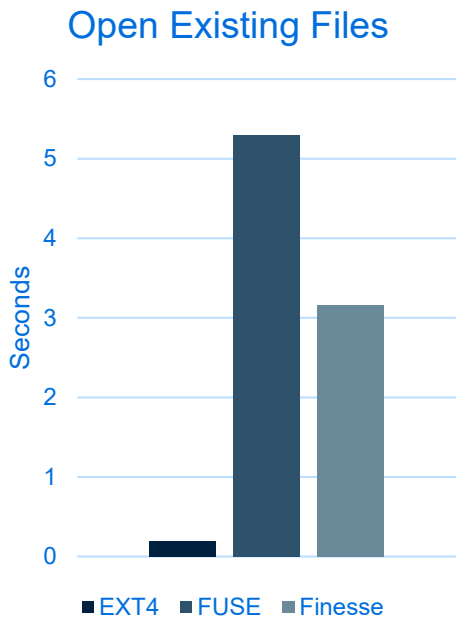
Evaluate impact on existing calls

- Note that Finesse imposes an additional call for open
  - Send map name message (to obtain handle)
  - Issue open call
  - Despite this, 31% *faster*
  - 2% slower opening non-existent files

Evaluate potential optimized interface

- Path search: frequently done, often with an array of options
- Single call permits execution in the server
- 80-88% less time to execute than FUSE
- 1.8x more time than native EXT4

MICRO BENCHMARKS



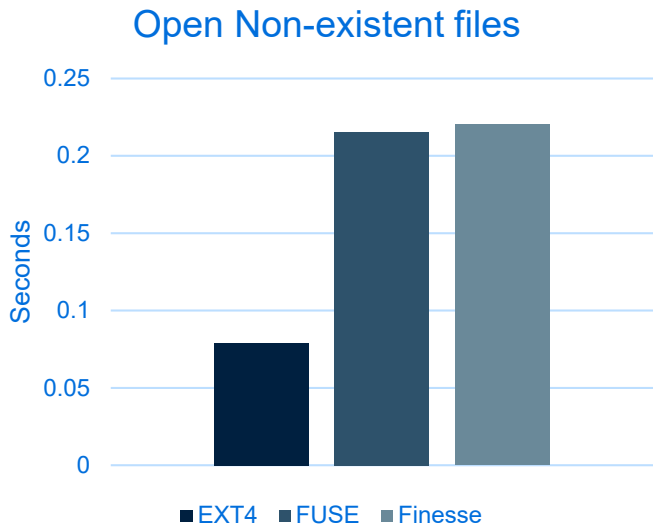
Create 1,000 random files in a directory

Open, then close the files

Delete the files

MICROBENCHMARKS

Attempt to open 1,000 non-existent files in a single directory.

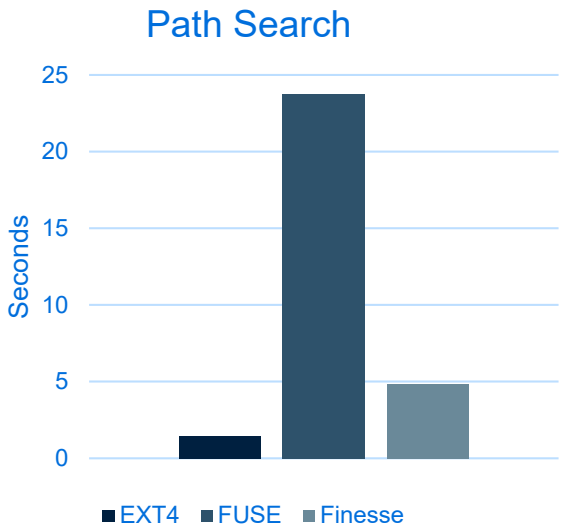


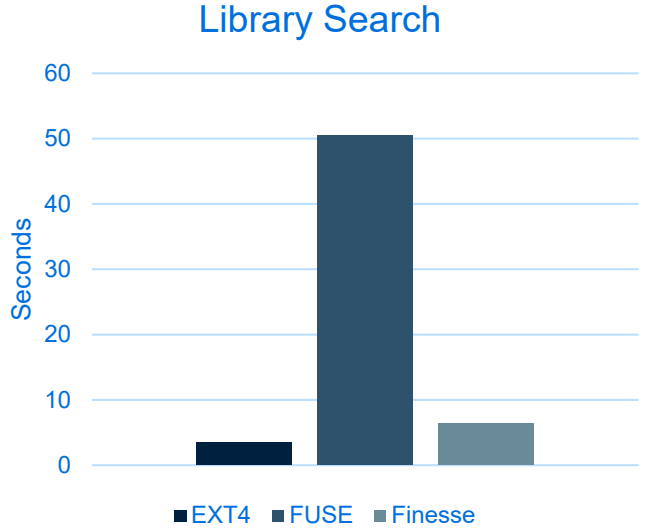
MICROBENCHMARKS

Lookup every executable in the search path (path search)

Lookup every library in the library search path

(Each run = 100 iterations)





FUTURE WORK

Extend functionality beyond passthrough (read-only) support

Additional benchmarks (not just micro-benchmarks)

Code is unoptimized

- POSIX message queues (shared memory) – consider other IPC (e.g., ffwd work?)
- Performance is not CPU saturated – find bottlenecks and remove

Pick Further interface enhancements

- Directory mapped into shared memory
- Evaluate other common sequences and batch

Is existing FUSE the right model?

OBSERVATIONS

Several insightful observations during this work

- Shared kernel state (file handle, position pointer) – complicates this work
- Kernel path is almost all CPU time in these tests
- FUSE and Finesse elapsed is 2-3x CPU time
  - Suggests plenty of room for improvement
- Finesse pre-lookup speeds up overall open time. Cache pre-seeding?

Relatively easy to add functionality

- How to generalize this – enable innovation via Finesse