# Logic File System

## the filesystem of the 21st century

or hierarchical filesystems considered harmful

Yoann Padioleau

supervised by Olivier Ridoux

IRISA / University of Rennes 1

# Plan

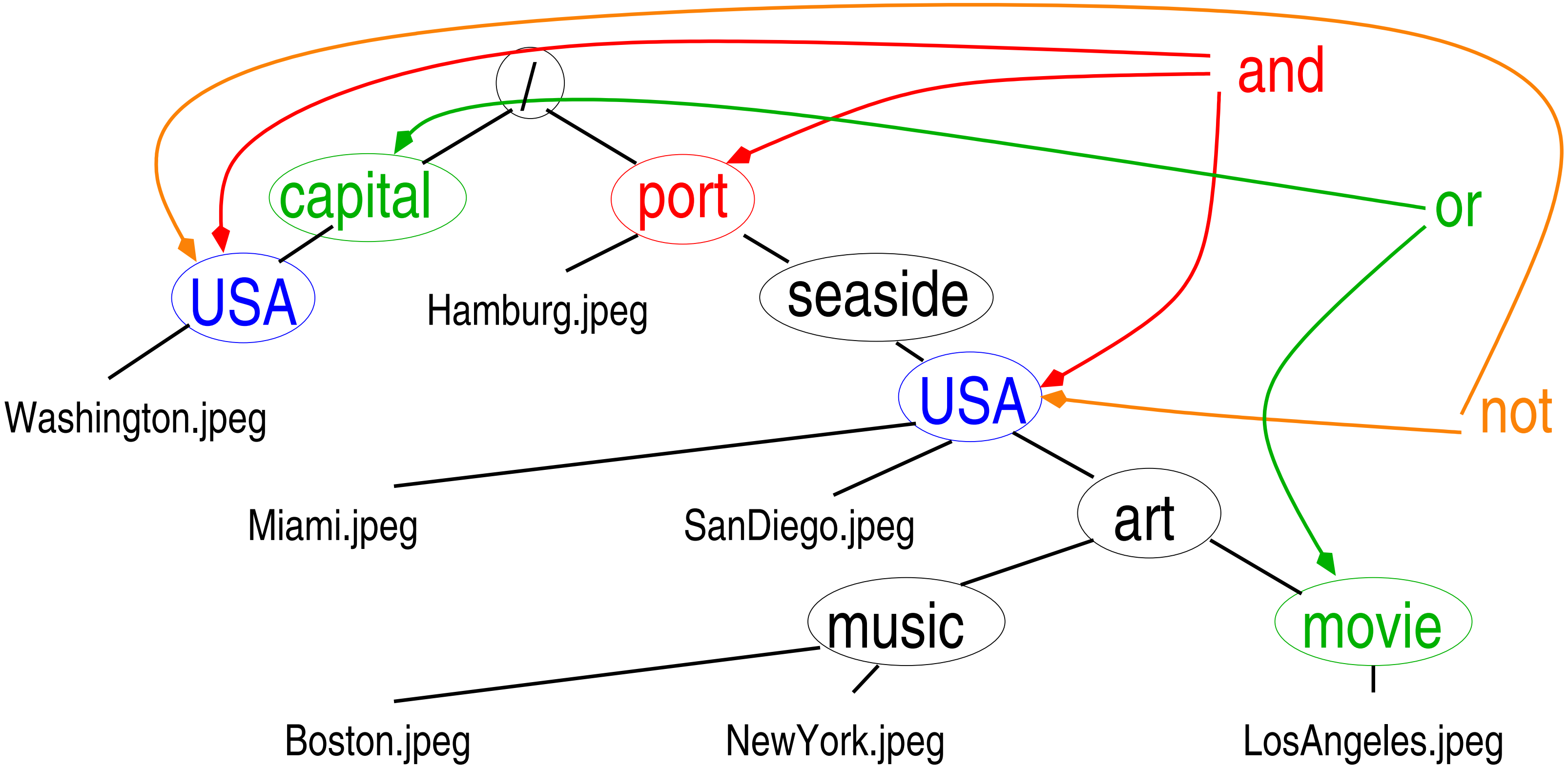
- Motivations
  (combine navigation and query in a filesystem: advanced dirs)
  (several points of view on file contents: advanced files)
- Demo (managing photos, music, code, bibtex, . . . )
- Specification (cd = ?, ls = ?, mv = ?, cat = ?,. . . )
- Security (read-write-execute = ?, user-group-other = ?)
- Implementation and evaluation (fast enough ?)
- Related work (advanced filesystems, search tools, and IDE)
- Conclusion and further works (take over the world)

# A toy example

to represent a collection of city maps

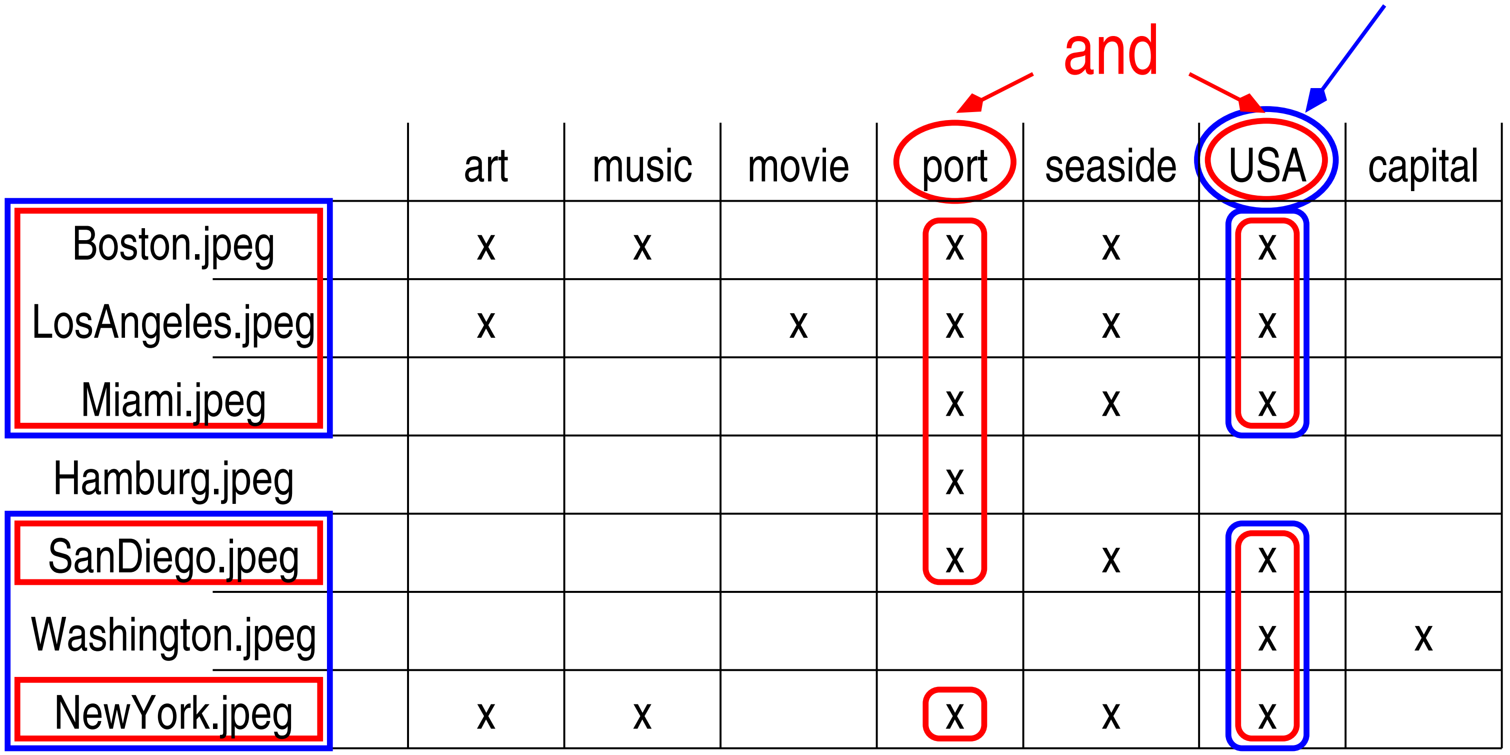
- a collection of cities — Boston, Hamburg, Los Angeles, Miami, New York, San Diego, Washington
- a collection of descriptive attributes
  - to be a port, on the seaside,
  - to be in the USA,
  - to be a capital,
  - to be an art city, for music, or movie

# A hierarchical filesystem



a meaning for cd /USA, cd not USA, cd capital or movie, and cd port and USA?

# Boolean organization (Google)



| | art | music | movie | port | seaside | USA | capital |
|---|---|---|---|---|---|---|---|
| Boston.jpeg | x | x | | x | x | x | |
| LosAngeles.jpeg | x | | x | x | x | x | |
| Miami.jpeg | | | | x | x | x | |
| Hamburg.jpeg | | | | x | | | |
| SanDiego.jpeg | | | | x | x | x | |
| Washington.jpeg | | | | | | x | x |
| NewYork.jpeg | x | x | | x | x | x | |

good for cd not USA and cd capital or movie

but, not incremental enough for cd USA and cd port and USA

# Motivation

→ merge navigation (as in hierarchical organizations)

and querying (as in boolean organizations)

in a file system

(every tool benefits of it: from shells to multimedia players)

advanced (virtual) directories

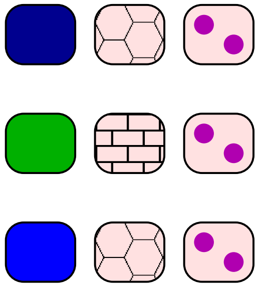
# LFS organization

- assign multiple properties to files

file1.txt =

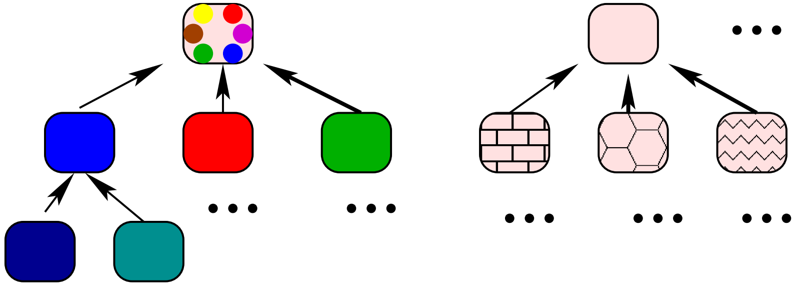
file2.jpg =

file7.jpg =



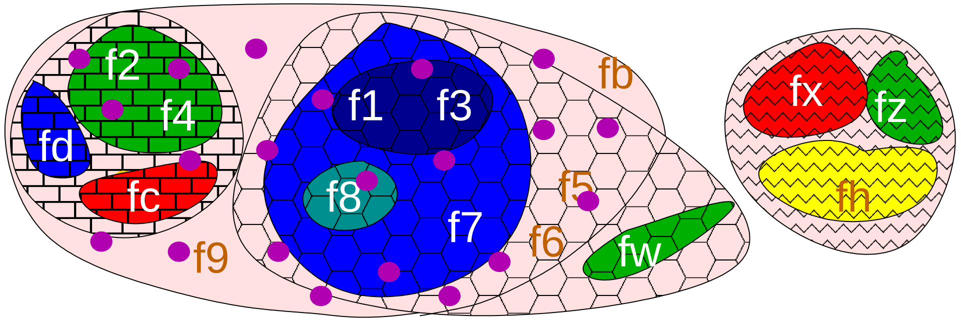
⋮

- order properties



an example:

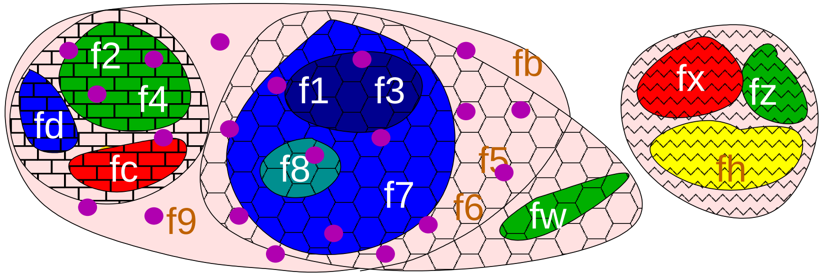
Venn diagram representation

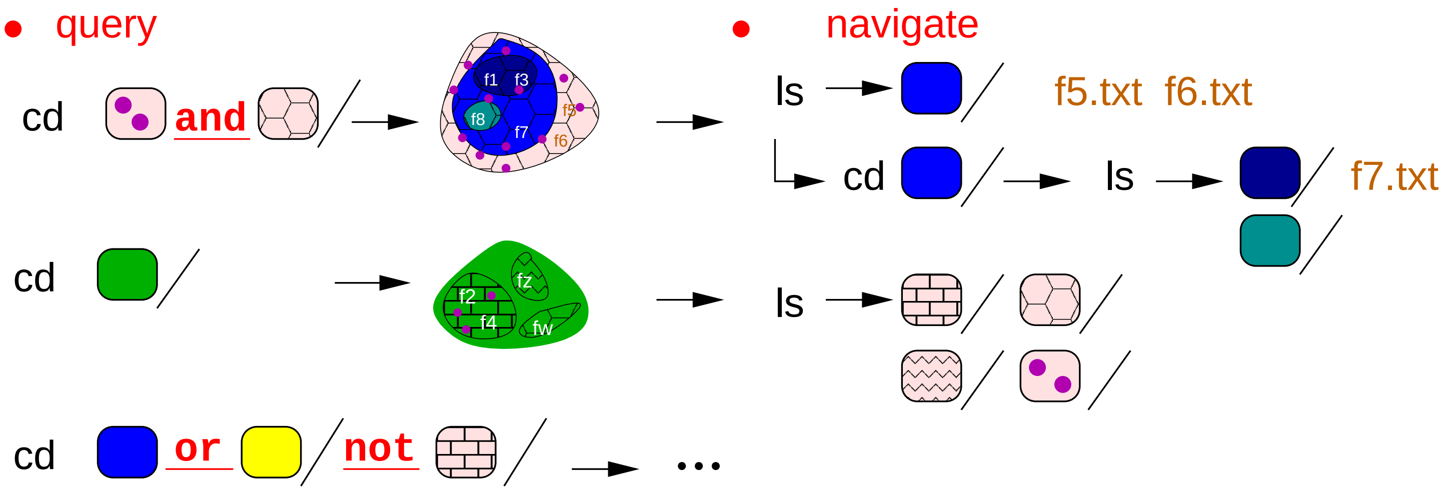


# Advanced directories

an example:

Venn diagram representation





- query

cd ● and ⬡ / → →

cd 🟩 / → →

cd 🟦 or 🟨 / not ▦ / → …

- navigate

ls → 🟦 / f5.txt f6.txt

cd 🟦 / → ls → 🟦 / f7.txt

🟦 /

ls → ▦ / ⬡ / ▨ / ● /

# From advanced directories to advanced files

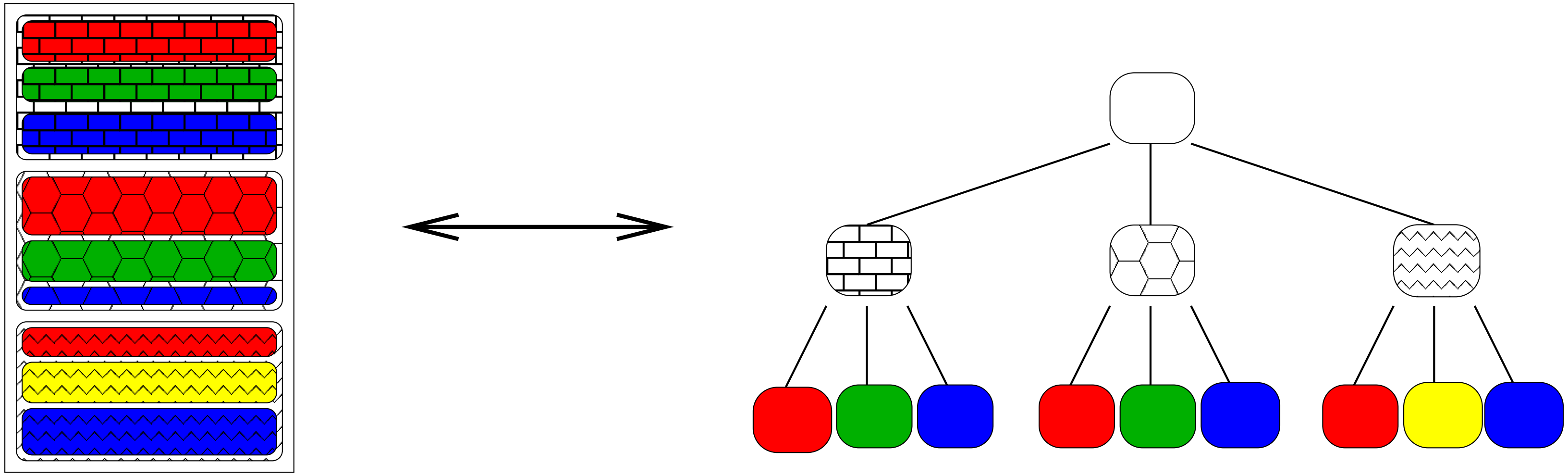
- in hierarchical filesystems, files have the same problems as directories: rigid contents (file contents vs directory contents)

- we have proposed a substitute to directory trees
  navigation and querying among files: advanced dirs

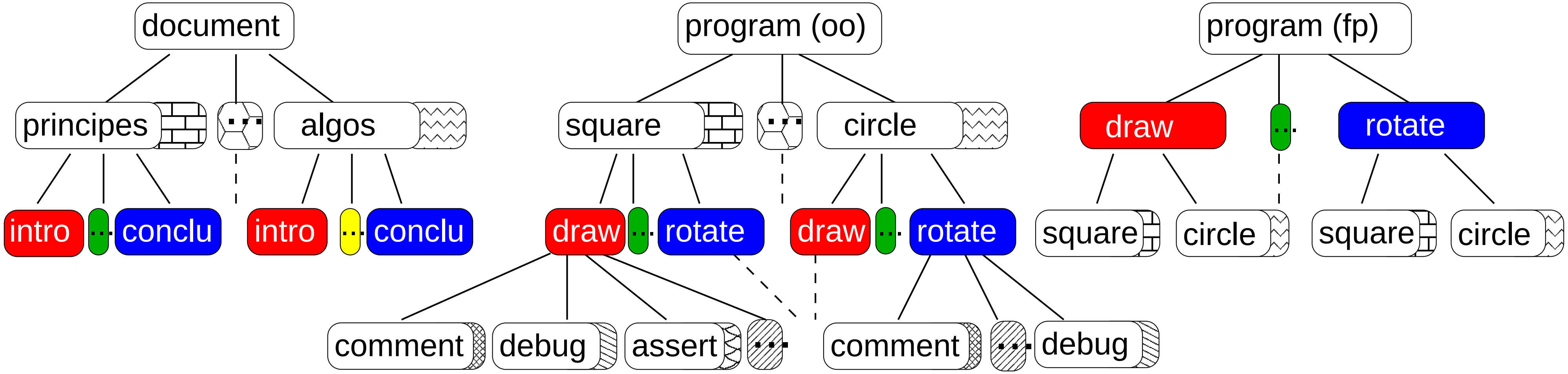
- the missing link:
  navigation and querying inside files: advanced files

# Hierachies, and their problems, are everywhere

- a file contents (symbolic)
- file structure (symbolic)



- a latex file
- a c++ file
- a caml file



# Motivation

⟶ query, better understanding

navigation, easy focus on one task

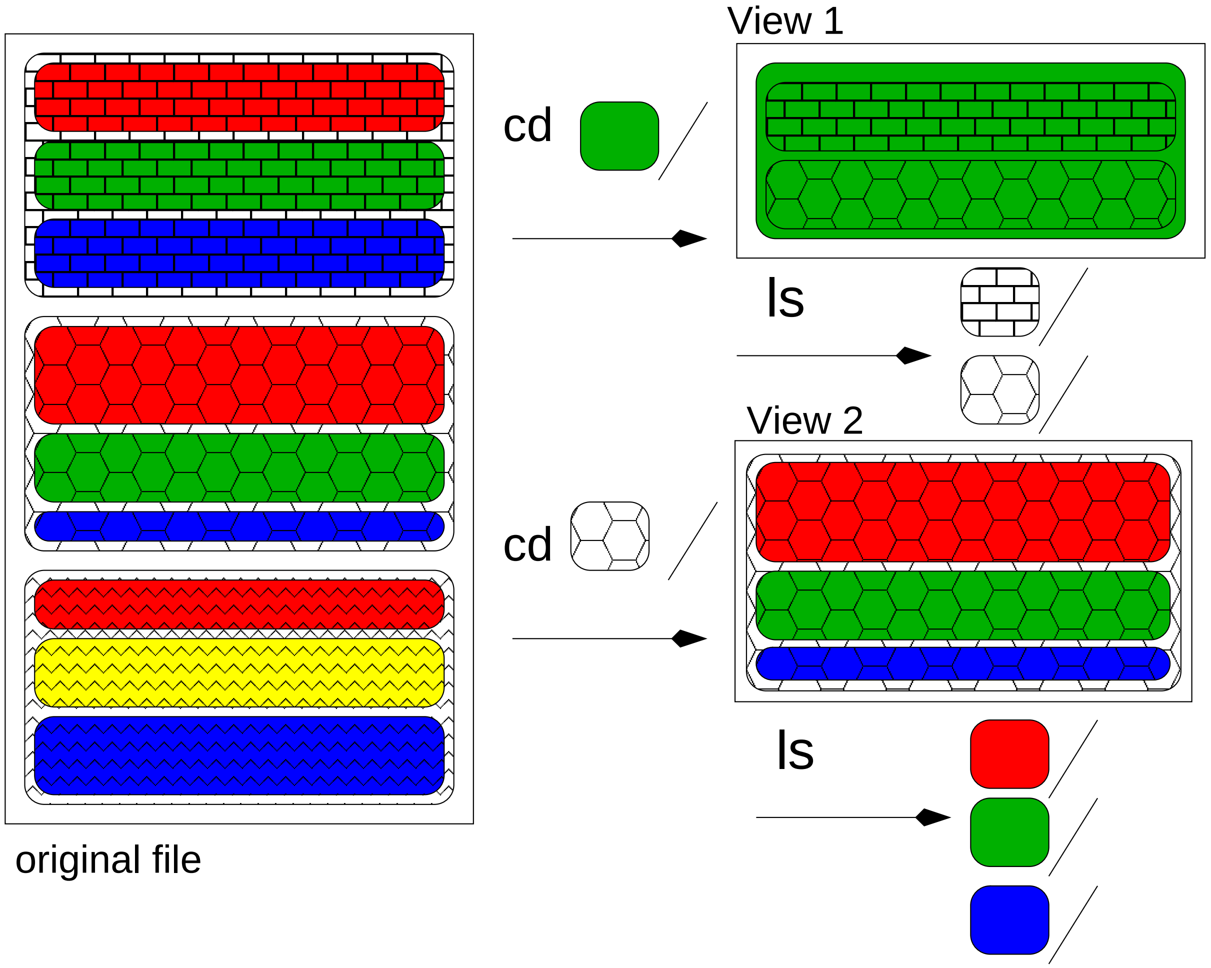
editing of points of view coherent change

on file contents

advanced (virtual) files

# Advanced files

- assign multiple properties to parts of file
- query, navigate, view, update



# Demo

Party Time!

# Specification

based on a previous work on Logic Information Systems (LIS) [Ferré&Ridoux, IP&M'2004]

(LFS is to LIS what O'Caml is to $\lambda$calcul)

# LFS content

- a logic — $A \models B$
  - core deduction rules ($a \wedge b \models a$)
  - axioms (*music* $\models$ *art*)
  - additional logic engines (*size*:$4$*Mo*$45$*Ko* $\models$ *size*:$>3$*Mo*)

- information — an attachment $d$ (description) of a logic formula to every object $o$ of a collection $\mathcal{O}$ (files or parts of file)

  $d($*SanDiego.jpg*$) =$ *port* $\wedge$ *seaside* $\wedge$ *USA* $\wedge$ *size*:$4$*Mo*$45$*Ko*

  $d($`y = x;`$) =$ *var:x* $\wedge$ *var:y* $\wedge$ *function:f*

  - user gives properties (interesting, tested)
  - LFS gives system properties (size:3Mo, owner:pad)
  - additional transducers give properties (res:640x480, var:x)

# Querying LFS

- paths are formulas
- extension — given a path $pwd$, the set of all objects that satisfy this path

$$\textsf{ext}(pwd) = \{o \in \mathcal{O} \mid d(o) \models pwd\}$$

$$\textsf{ext}(pwd) \quad \approx \quad \textsf{ls} \ \ \textsf{-R} \ \ pwd$$

$$\textsf{LosAngeles.jpg} \in \textsf{ext}(\textsf{art}) \text{ because}$$

$$d(\textsf{LosAngeles.jpg}) = \textsf{movie} \wedge \textsf{port} \wedge \textsf{USA} \models \textsf{movie} \models \textsf{art}$$

- paths denote directories that denote extensions

(working directory $\equiv$ working query)

# Navigating LFS

- to be a subdirectory of a directory — given a directory (denoting) $O$, every directory (denoting) $O'$ such that $O' \subset O$

$$\mathit{Dirs}(pwd) = max_{\models}\{p \in \mathcal{P} \mid \emptyset \subset \mathit{ext}(pwd \wedge p) \subset \mathit{ext}(pwd)\}$$

$p$ refines $pwd$

only «largest» subdirectories are relevant to navigation

(most relevant hints)

- to be a file of a directory — given a path $pwd$, to be in $\mathit{ext}(pwd)$, and in the extension of no subdirectory

$$\mathit{Files}(pwd) = \mathit{ext}(pwd) - \bigcup_{p \in \mathit{Dirs}(pwd)} \mathit{ext}(p)$$

$$\mathit{Files}(p) \bigcup \mathit{Dirs}(p) \quad \approx \quad \text{ls} \quad p$$

# Security

- security properties $=$ logic properties (secu:alice=rw)
- security rules $=$ logic rules (rw $\models$ r, group $\models$ user)

  $\longrightarrow$ security in LFS (almost) for free (à la ACL and more)

- examples:
  - `mv foo.c secu:alice=rw/secu:bob=r/`
  - `mv foo.txt secu:not(ridoux)=rw/`
  - `cd owner:pad/secu:group(student)=r/`

  (only the owner of the file can change the properties of the file)

# Implementation

# Basic principles

to implement the specification at a reasonable cost

- to call $\models$ is costly

  avoid it by representing on disk a directed acyclic graph (DAG) of the properties: logic cache

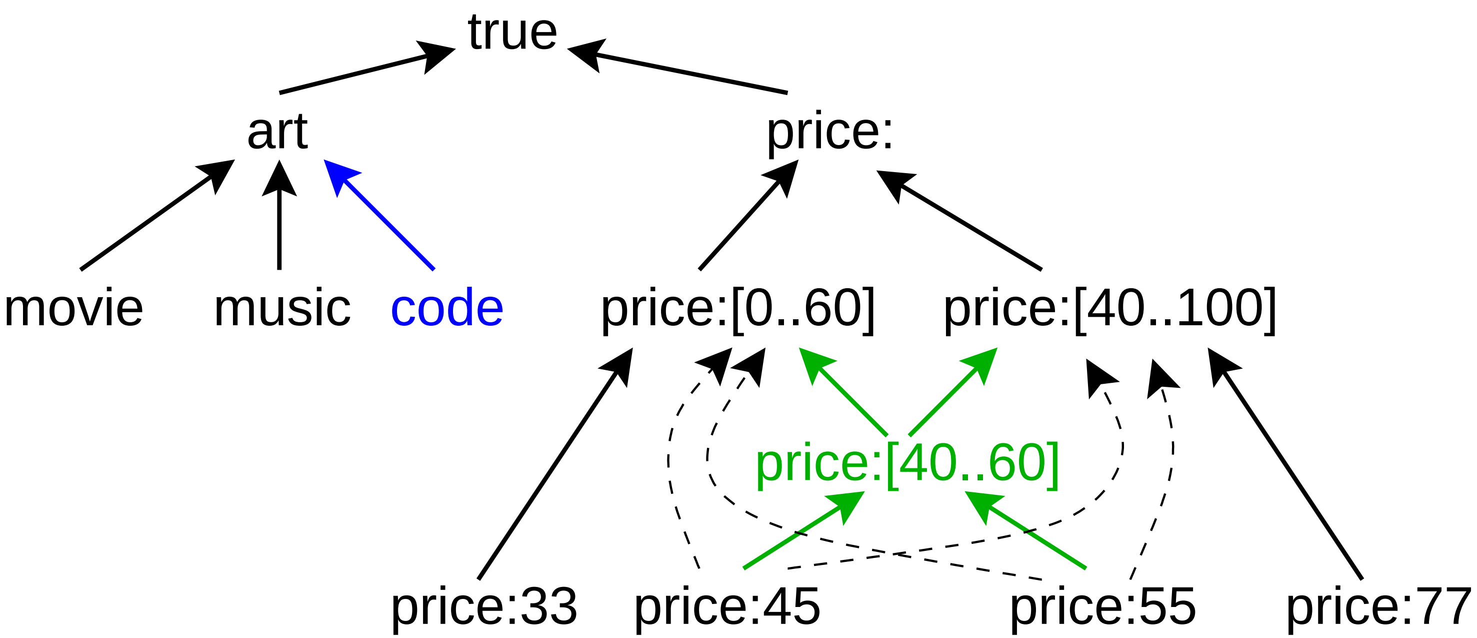
- to go through all objects ($\{o \in \mathcal{O} \mid \ldots\}$) and all properties ($\{p \in \mathcal{P} \mid \ldots\}$) is costly

  avoid it by representing relation $d$ as a table and inverted table on disk: indexing.

- to call part transducers is costly

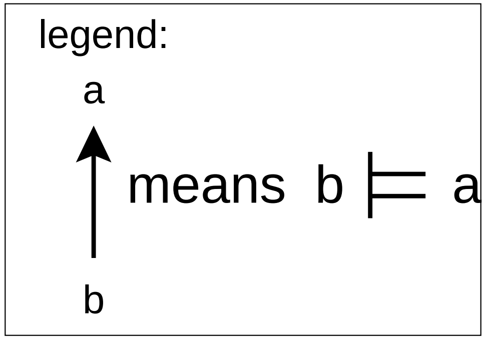
  define synchronization points for re-indexing

# Logic Cache

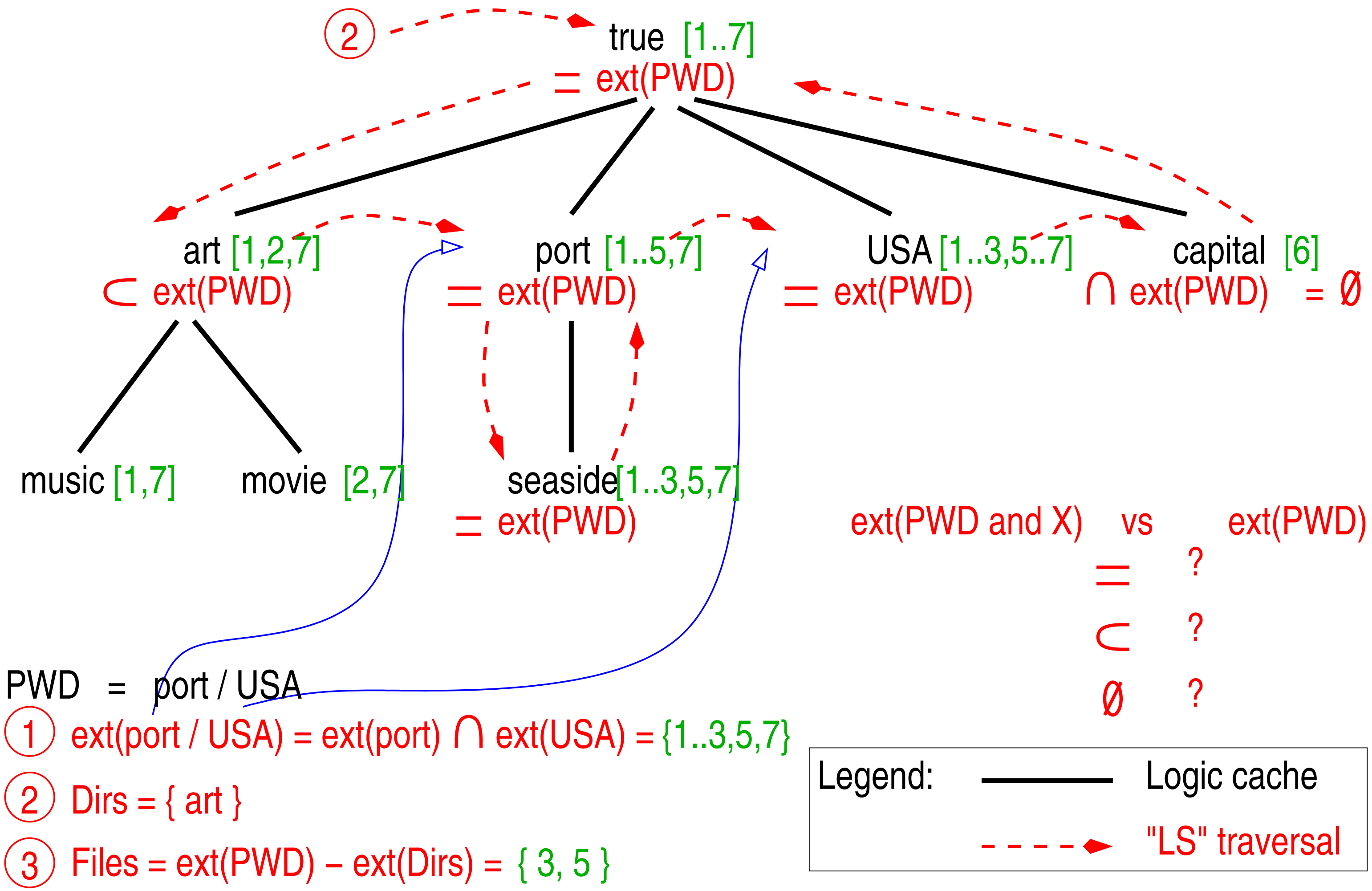


- cd art/; mkdir code
- cd price:[40..60]/

note that this is not the navigation structure (sub−property != sub−directory)



# Computing *Files* and *Dirs*

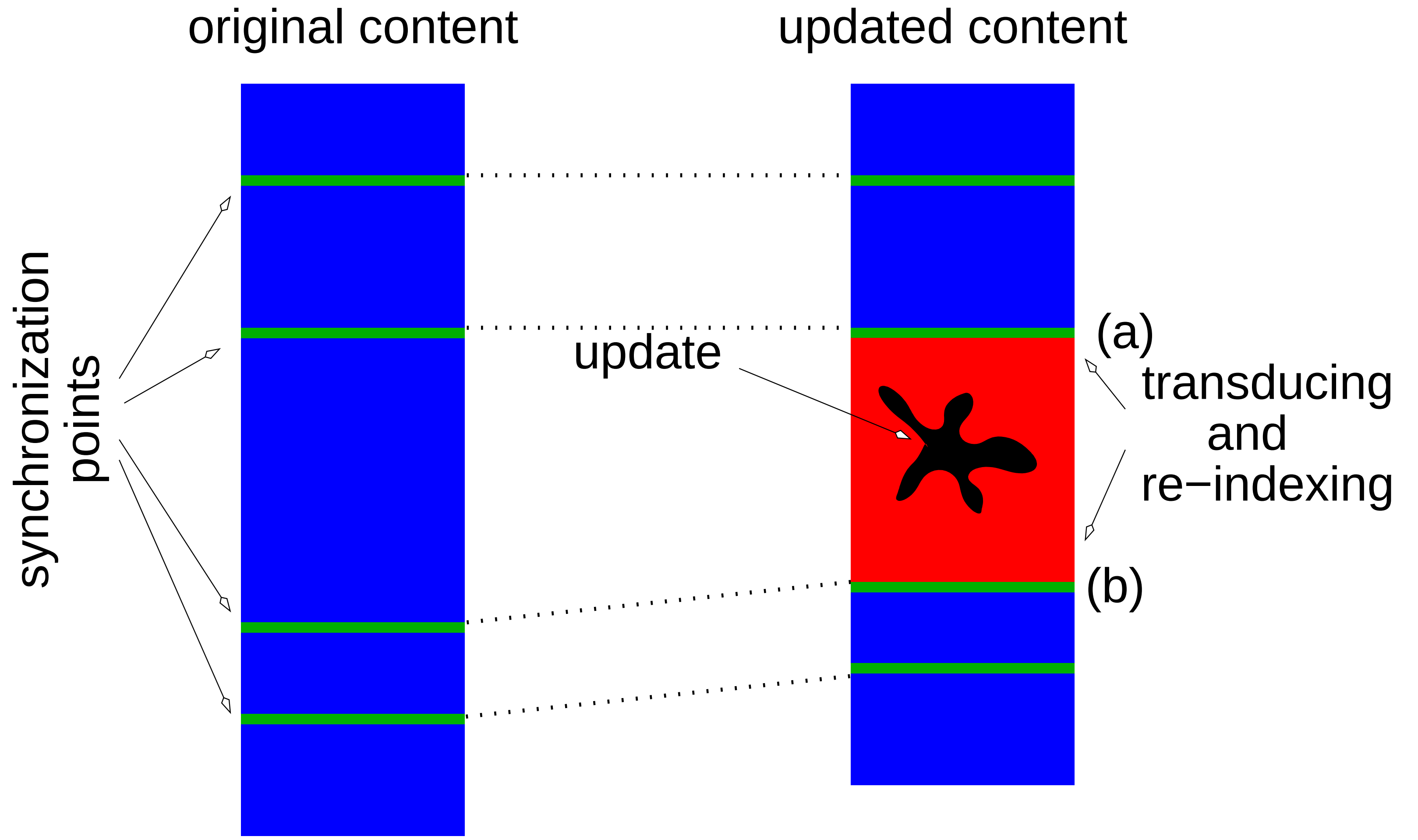


PWD = port / USA

1. ext(port / USA) = ext(port) ∩ ext(USA) = {1..3,5,7}
2. Dirs = { art }
3. Files = ext(PWD) – ext(Dirs) = { 3, 5 }

# Synchronization points

places in a file where a part transducer goes back to its initial state



# Evaluation

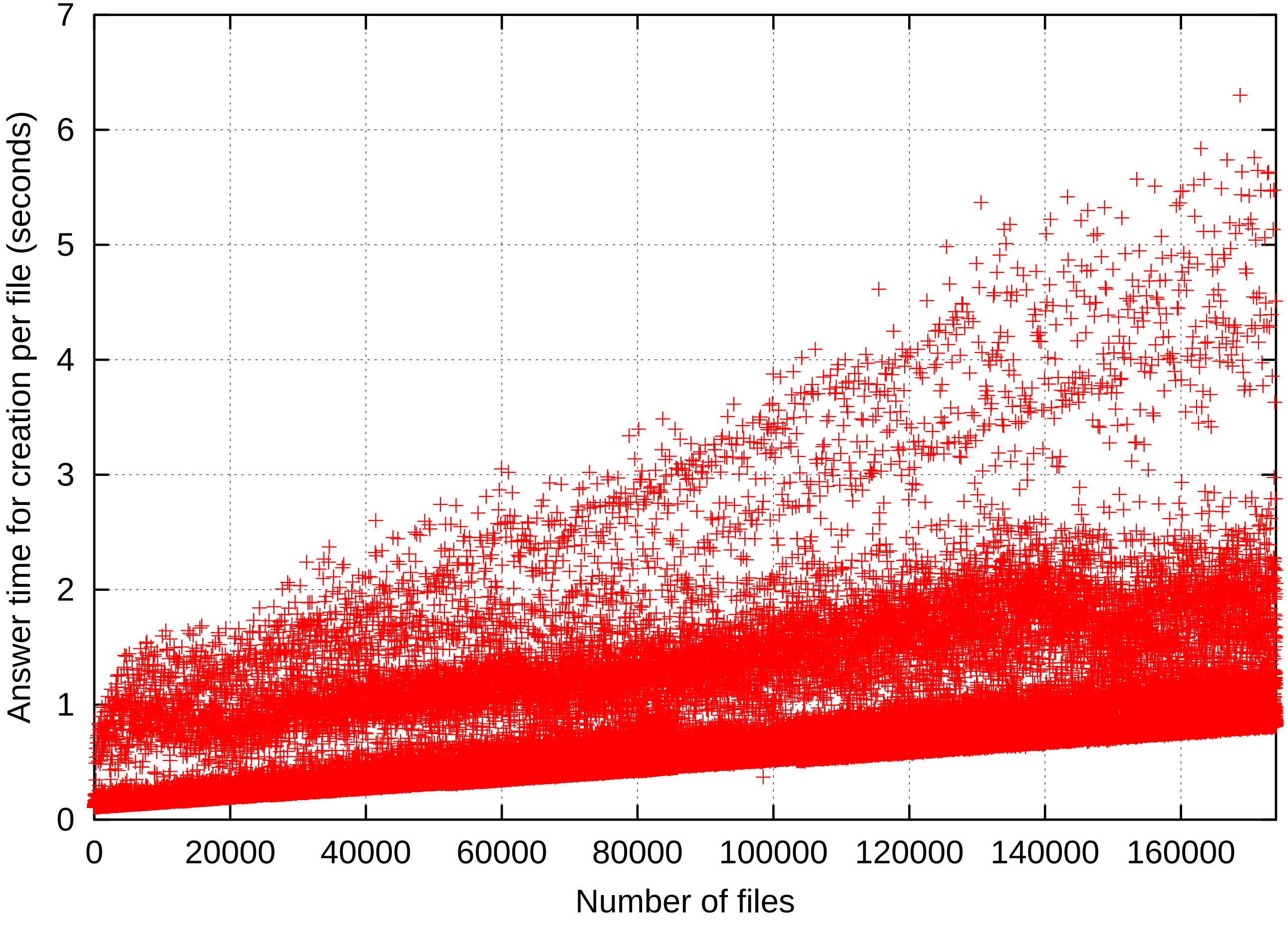
- software — a filesystem coded mostly in O'Caml
- platform — Linux kernel 2.4, with a 2Ghz Pentium 4, 750Mb RAM, and a 40 Gb IDE disk.

# Benchmark

- data — music, emails, man pages, latex, bibtex, ocaml sources
- context —
  - up to 100 000 files (≈ 10 prop per files)
  - up to 100 000 lines (≈ 10 prop per parts)
- time —
  - `ls` ≈ 1 sec (compute *Dirs* and *Files*)
  - `cd` ≈ 1 sec (may have to insert property in cache)
  - `cp` ≈ 1 sec (file transducer calls and indexing)
  - `cat` ≈ 1 sec (compute view content with *ext*)
  - `edit` ≈ 1 sec (part transducer calls and reindexing)
  - `cd parts` from 0.1 sec to 3 min (pure indexing)

⟶ compatible with an interactive usage

# E-mails from linux kernel mailing list



# Related work

- advanced file systems

  (SFS [Gifford et al.91], CATFS, BeFS [Giampaolo.93], Nebula [Bowman et al.94], HAC [Gopal & Manber.99])

  ⟶ no navigation after arbitrary query

- information retrieval tools

  (SuperBook [Remde et al.87], Scatter/Gather [Cutting et al.92], concept analysis [Lindig.95], Flamenco [Yee et al.03])

  ⟶ navigation after query but poor query

- programming language IDEs

  (Emacs [Stallman.81], Smalltalk [Goldberg et al.84], Omega [Linton.84], CIA [Chen et al.90], TuringTool [Cordy et al.90], Hyperspace [Tarr et al.99])

  ⟶ limited query / navigation / view / updating inside files

# Contributions

- a running alternative to hierarchical file systems
- integration of query and navigation, with manual and automatic assignation and ordering of properties, for files and parts of files. (LFS is to LIS what O'Caml is to $\lambda$calcul)
- an architecture for generic services

  (many kind of files: music, email, ocaml, perl, latex, bibtex, . . . )

  (many kind of logics: boolean, intervals, regexp, type, . . . )
- a security model
- encouraging performances

$\longrightarrow$ USENIX[2003] (advanced dirs)

$\longrightarrow$ USENIX[2005] (advanced files)

# Further work

- more general kinds of parts: tokens, video segments, . . .

  [1] cd scene:kung-fu|amour

  [2] play movie.mpeg

  [3] ls ⟶ Tom Cruise/ Monica Belluci/ JCVD/ . . .

- explore LFS usage
  - for software engineering (UML, AOP, slicing, versioning, . . . )
  - for personal digital assistant (PDA)
- distribution
  - of computation (LFS algorithms are highly parallelizable)
  - of data (Semantic Web, peer-to-peer)
- relations/links between objects (ln foo.o <compiled> foo.c )
- improve performance (especially file creation)

"That's all Folks!"

# A LFS scenario (1/2)

mounting

```
[1] % mount /dev/hda1 /lfs/ ; cd /lfs/
```

taxonomy

```
[2] % mkdir art
[3] % cd art; mkdir music; mkdir movie; ...
```

(adds *music* $\models$ *art*, $\ldots$)

context

```
[4] % cp /x/washington.jpg   USA/capital/
[5] % cp /x/Boston.jpg seaside/USA/
```

($d$(*Boston.jpg*) $=$ *seaside* $\wedge$ *USA*)

updating

```
[6] % cd USA/seaside/
[7] % mv Boston.jpg music/
```

($d$(*Boston.jpg*) $=$ *music* $\wedge$ *seaside* $\wedge$ *USA*)

navigating and querying

```
[8] % ls port/USA                ⟶ art/ Miami.jpg SDiego.jpg
[9] % ls USA                     ⟶ art/ port/ capital/
[10] % ls !USA                   ⟶ Hamburg.jpg
[11] % cd port|art/!(USA&FR)/    ⟶ ...
```

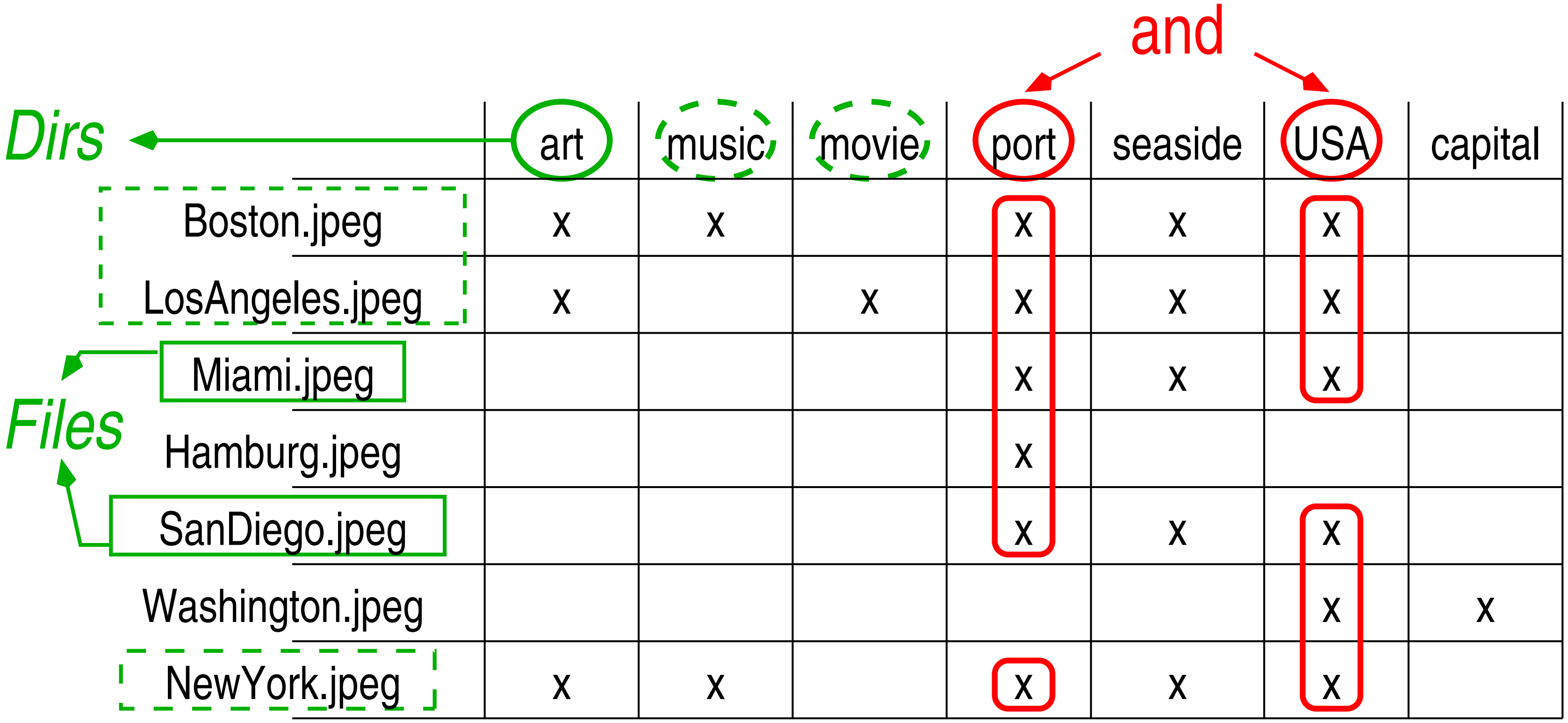
# A LFS scenario (2/2)

```
[1] % cp /x/foo.c
/lfs/project:foobar/
[2] % cp /x/trans_C.exe
/lfs/parts_transducer:c/
[3] % cd project:foobar/
[4] % cat foo.c
int f(int x) {
   int y;
   assert(x > 1);
   y = x;
   printf("x = %d", x);
   return y * 2
}
int f2(int z) {
   return z * 4
}
```

```
[5] % cd parts; ls
function:f/ function:f2/
var:x/ var:y/ var:z/
debug/ assert/
foo.c
[6] % cd function:f/-
!(debug|assert)/
[7] % cat foo.c
int f(int x) {
   int y;
   .........:1
   y = x;
   .........:2
   return y * 2
}
.........:3
```

# LFS organization

$\mathit{movie} \models \mathit{art}$ $\mathit{music} \models \mathit{art}$ $\ldots$



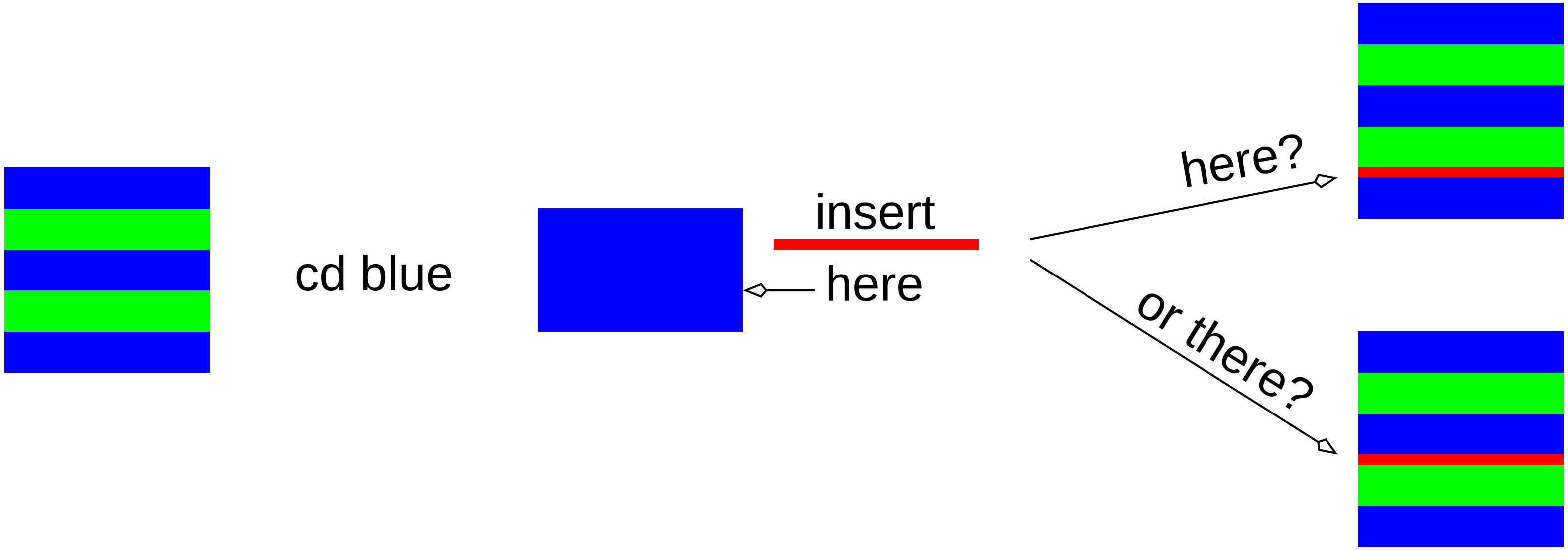
| | art | music | movie | port | seaside | USA | capital |
|---|---|---|---|---|---|---|---|
| Boston.jpeg | x | x | | x | x | x | |
| LosAngeles.jpeg | x | | x | x | x | x | |
| Miami.jpeg | | | | x | x | x | |
| Hamburg.jpeg | | | | x | | | |
| SanDiego.jpeg | | | | x | x | x | |
| Washington.jpeg | | | | | | x | x |
| NewYork.jpeg | x | x | | x | x | x | |

## indexing

| | function:f | function:f2 | var:x | var:y | var:z | debugging | specification |
|---|---|---|---|---|---|---|---|
| 1 | X | | X | | | | |
| 2 | X | | | X | | | |
| 3 | X | | X | | | | X |
| 4 | X | | X | X | | | |
| 5 | X | | X | | | X | |
| 6 | X | | | X | | | |
| 7 | X | | | | | | |
| 8 | | X | | | X | | |
| 9 | | X | | | X | | |
| 10 | | X | | | | | |

# updating LFS

- re-indexing — updating a view ⟶ updating the object×property matrix,
- marks of missing parts — . . . . . . . . . . . . : X

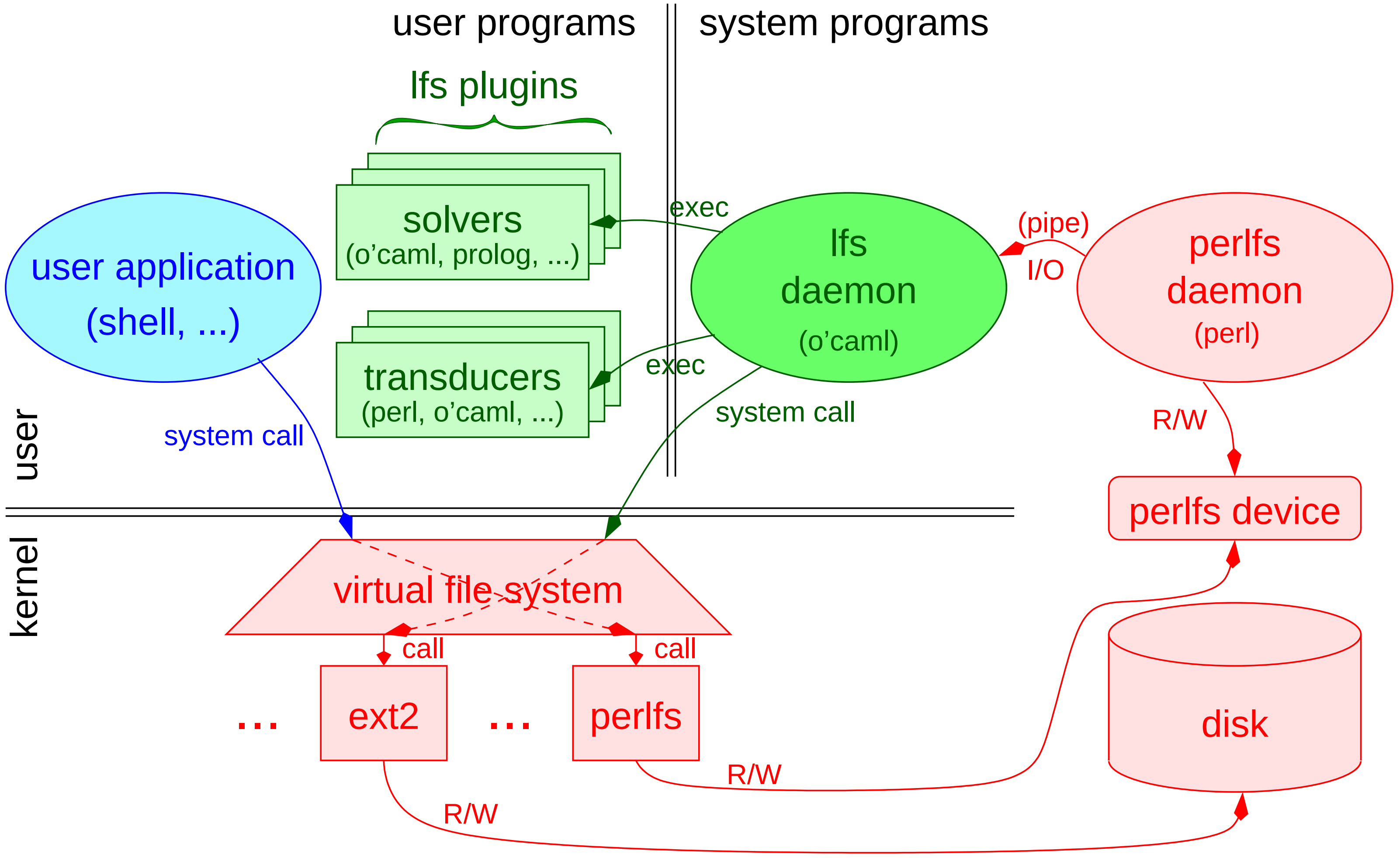


to know where to insert new parts

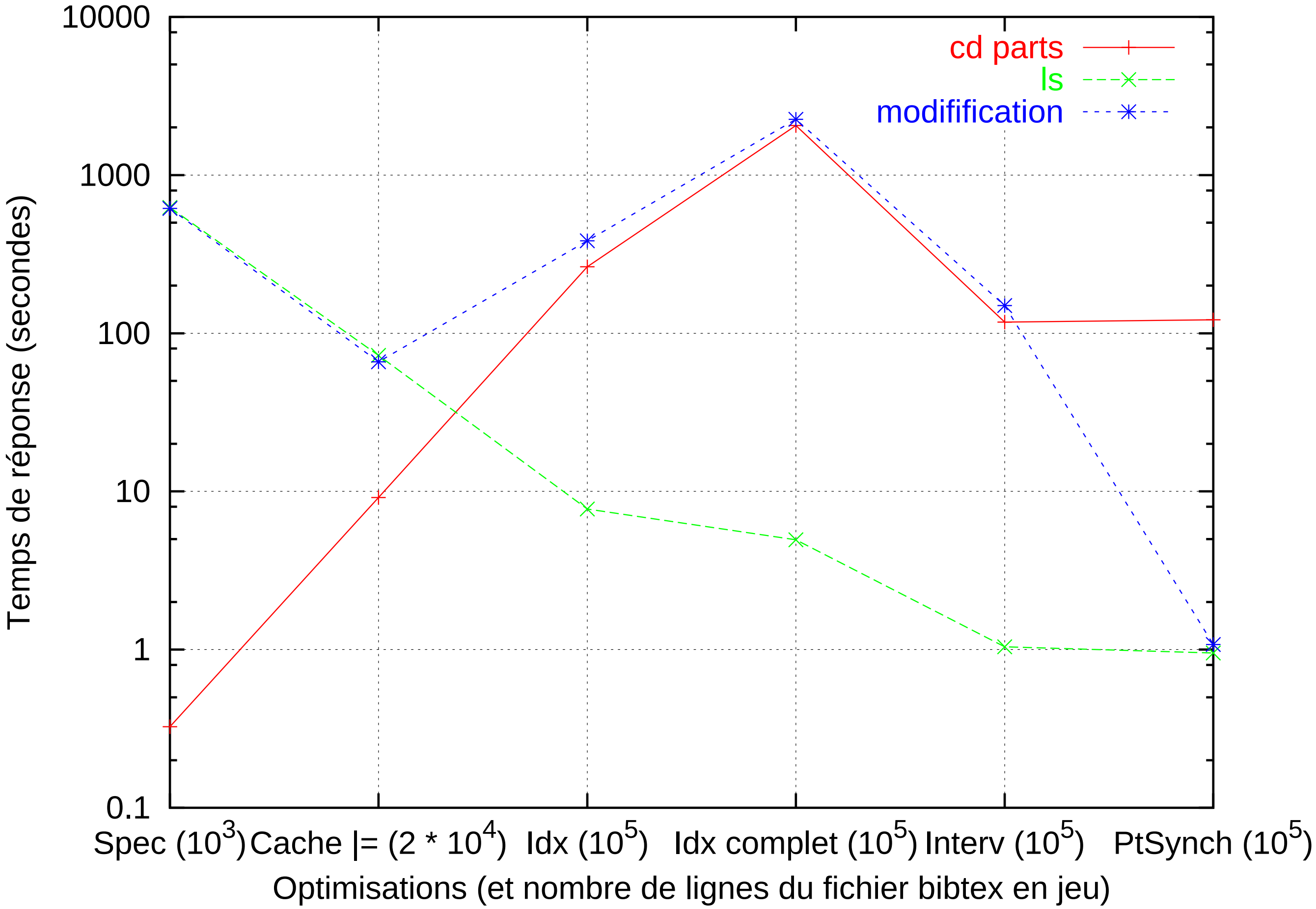
# Semantics of LFS operations

- readdir(path) — lists $\mathit{Files}(\mathit{path}) \bigcup \mathit{Dirs}(\mathit{path})$ (ls path)
- lookup(name,path) — checks $\mathit{name} \in \mathit{Files}(\mathit{path}) \bigcup \mathit{Dirs}(\mathit{path})$
- create(name,path) — adds $d(\mathit{name}) = \mathit{path}$ (touch path/name)
- mkdir(name,path) — adds $\mathit{name} \models \mathit{path}$ (mkdir path/name)
- file operations — as usual (open, read, write, . . . )
- . . .

# LFS software architecture



# Do optimisations optimise ?


10000
1000
100
10
1
0.1
Temps de réponse (secondes)
cd parts
ls
modifification
Spec ($10^3$)
Cache |= (2 * $10^4$)
Idx ($10^5$)
Idx complet ($10^5$)
Interv ($10^5$)
PtSynch ($10^5$)
Optimisations (et nombre de lignes du fichier bibtex en jeu)