SibylFS and ImpFS: formal specification and implementation for filesystems

Dr Tom Ridge

REMS workshop, Cambridge, 2016-05-27

What have we done? SibylFS

- SibylFS, a filesystem specification and test oracle (POSIX, Mac, Linux and BSD filesystems)
- Spec formal, mechanized, written in Lem (translated to HOL, Isabelle, OCaml)
- Test oracle can analyse traces of real-world filesystems
- Tested lots of combinations of libc/OS/filesystem

Amusing application

- We presented at SOSP'15
- Also at that venue was a paper on FSCQ, a filesystem verified in Coq

dsheets commented on 3 Oct 2015

On an empty file system built from `27c4078`, running

```
#include <unistd.h>
#include <fcntl.h>

int main() {
  close(creat("spaaaaaace"));
  truncate("spaaaaaace",10);

  return 0;
}
```

results in a file, `spaaaaaace`, which contains the text `spaaaaaace` instead of 10 null bytes. For applications which expect POSIX, OS X, Linux, or BSD behavior when using `truncate` to extend files, this unexpected data can result in incorrectly formatted files.

zeldovich commented on 3 Oct 2015 Owner

Thanks; I will take a look. Sounds like we might have gotten the spec wrong.

Structure of the specification

- This is relevant to what we are trying to do now...

```
Call to libc
|
V
-------
SibylFS spec
(3 or 4 layers)
-------
|
V
-------
Dir_heap     // this is essentially two maps
-------
```

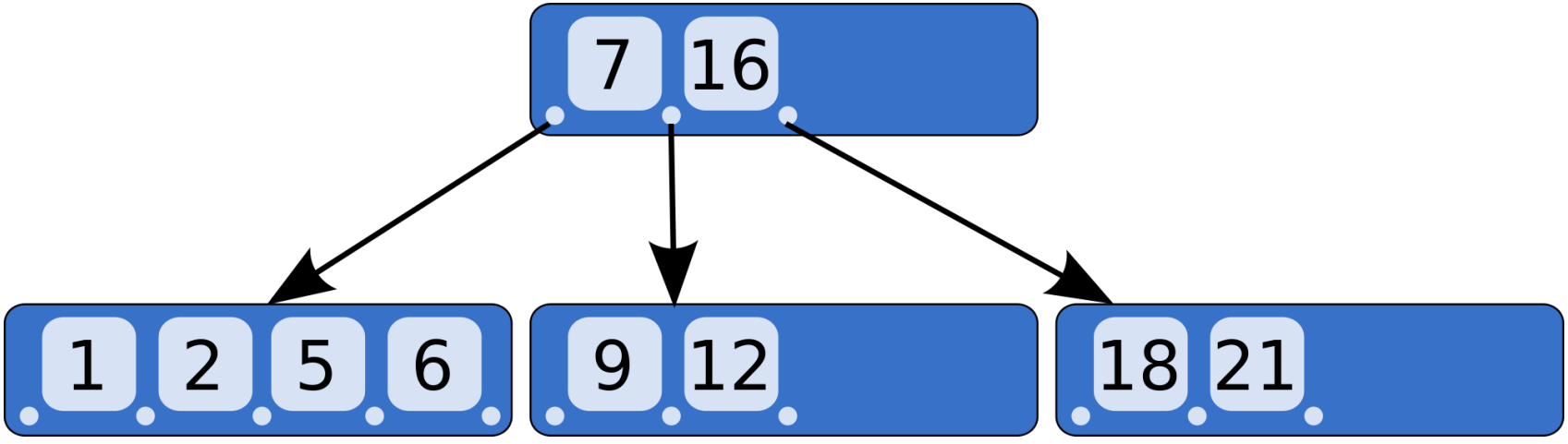
- `Dir_heap` is 2 maps, one from file ids to files, one from directory ids to directories

What are we doing now? ImpFS

- ImpFS, a verified filesystem implementation (from the start, this was what I really wanted to build)
- Based on B-trees

What are B-trees?

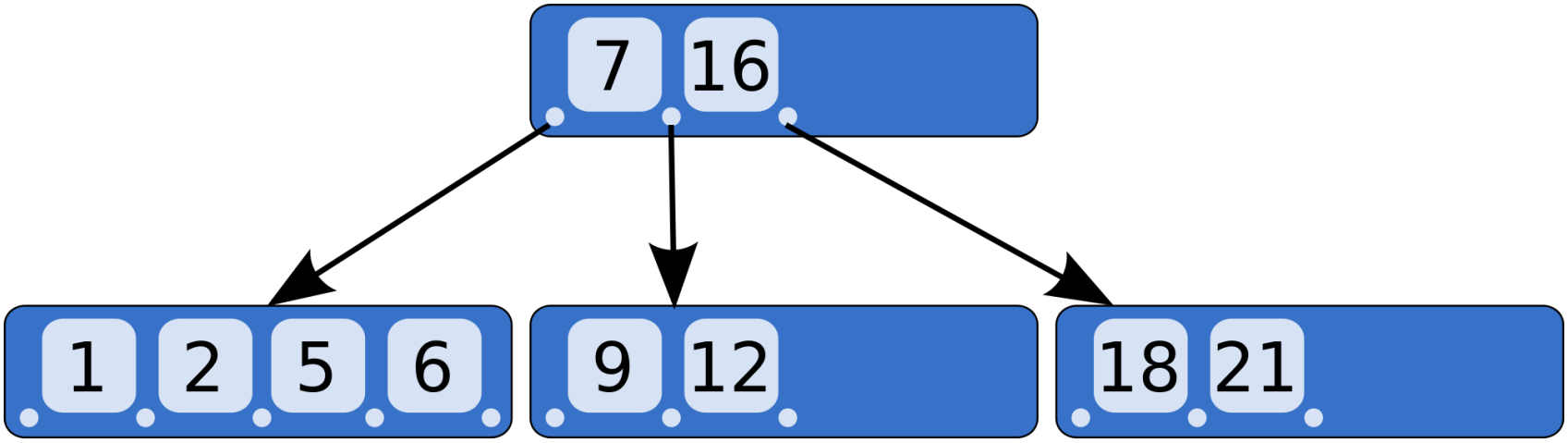
- A balanced tree-like datastructure, implementing a map:



- This shows the keys; in the leaves, we also have values associated with each key; the keys in the nodes allow you to navigate to a desired key in a leaf

Why are B-trees important in filesystems (and databases etc)?

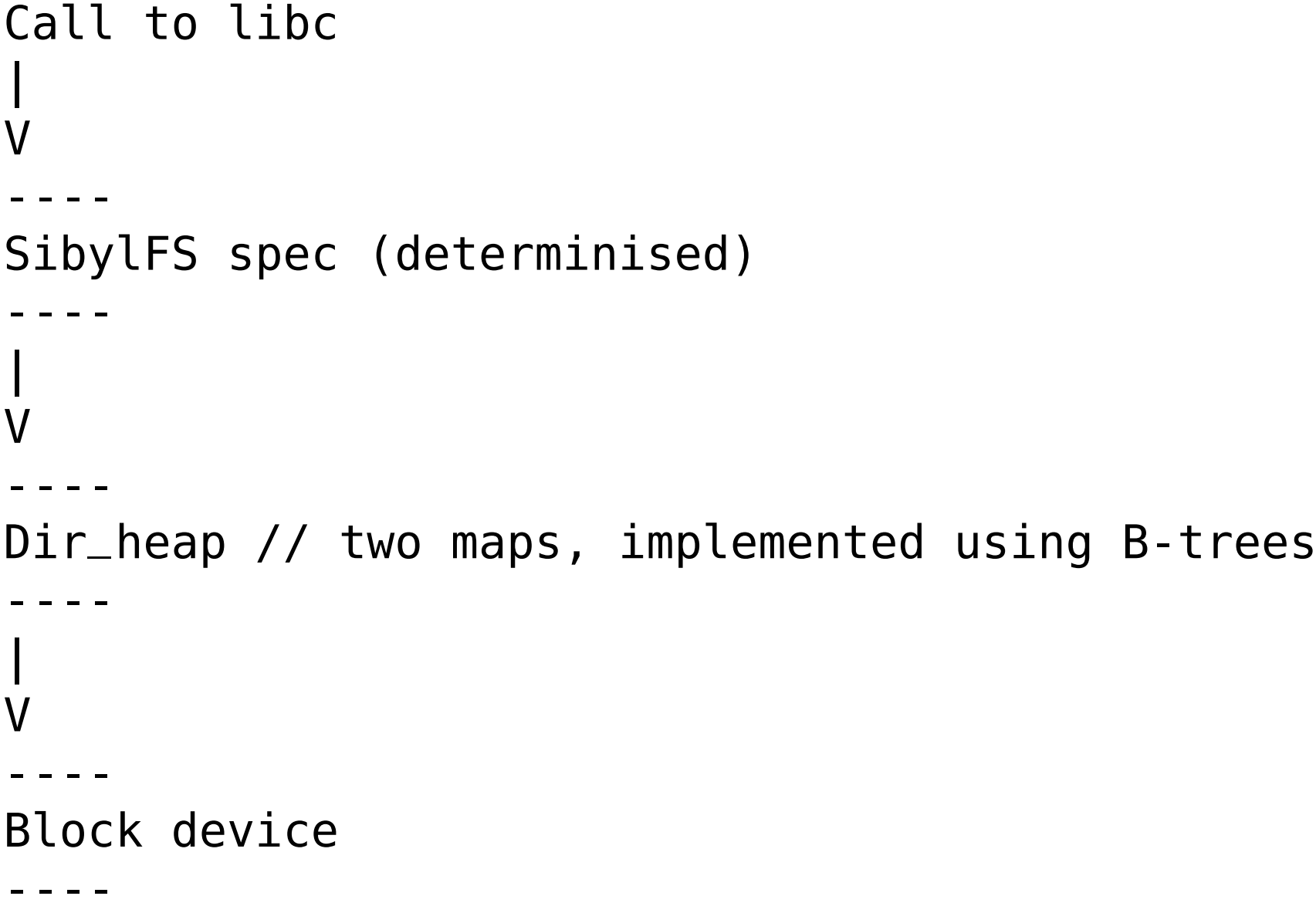
- Nodes are mapped onto blocks
- Block reads are slow compared to everything else
- Tree structure minimizes the number of blocks that are traversed (read from disk) to locate a value for a given key
- N.B. two datastructures - the tree on disk, and the partial tree in memory (e.g. the stack of blocks leading to a desired leaf)
- Moreover, if we update a map (add/remove a key) we can share most of the blocks with the previous version of the map



How to get a verified filesystem?

- Take the SibylFS specification
- Remove non-determinism, so it becomes an executable in-memory filesystem
- Implement the two maps at the bottom of the spec layers, and interface to a block device, to get a "real" filesystem using block storage; B-trees are used to implement maps on top of a block device

Structure of ImpFS



Of course, it is never that easy (challenges)

- Block layer (asynchronous; reorders writes; syncable; behaviour on crash; on-disk layout of data; pointers everywhere etc)
- Concurrency (we want an arbitrary number of B-trees concurrently using the block layer, ensuring all invariants are respected)
- **B-tree implementation and proof of correctness** taking account of block layer and concurrency requirements (I am most proud of this bit)
- Caching (want to avoid writes to block layer where possible)
- GC for unreachable blocks
- We want bounded memory usage (so have fixed size block caches etc)
- Performance
- "Semantic" optimizations
- Extra features (snapshotting etc)

ImpFS status

- Working on finishing the B-tree implementation and proof (should be an OCaml library and Isabelle theories in next few months)
- We hope for working filesystem prototype before October

Reuse

- B-tree proofs are very nice, and hopefully will form the basis for modern descriptions of this datastructure
- B-tree implementation (OCaml, Scala etc)
- ImpFS (in OCaml) could be used as a filesystem for Mirage, or for per-application private filesystems, or for filesystems on non-traditional block devices (network connections etc)
- B-tree can be used in many other applications (databases; per-application persistent kv maps etc)

End