

File Systems (Part 6)

Desired Properties of Directories

- **No restrictions on names**
- **Fast**
- **Space-efficient**

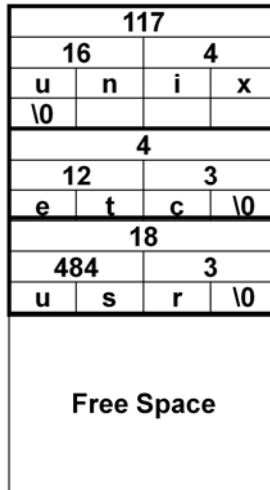
S5FS Directories

Component Name	Inode Number
directory entry	

.	1
..	1
unix	117
etc	4
home	18
pro	36
dev	93

An S5FS directory consists of an array of pairs of *component name* and *inode number*, where the latter identifies the target file's *inode* to the operating system (an inode is data structure maintained by the operating system that represents a file). Note that every directory contains two special entries, "." and "..". The former refers to the directory itself, the latter to the directory's parent (in the case of the slide, the directory is the root directory and has no parent, thus its ".." entry is a special case that refers to the directory itself).

FFS Directory Format

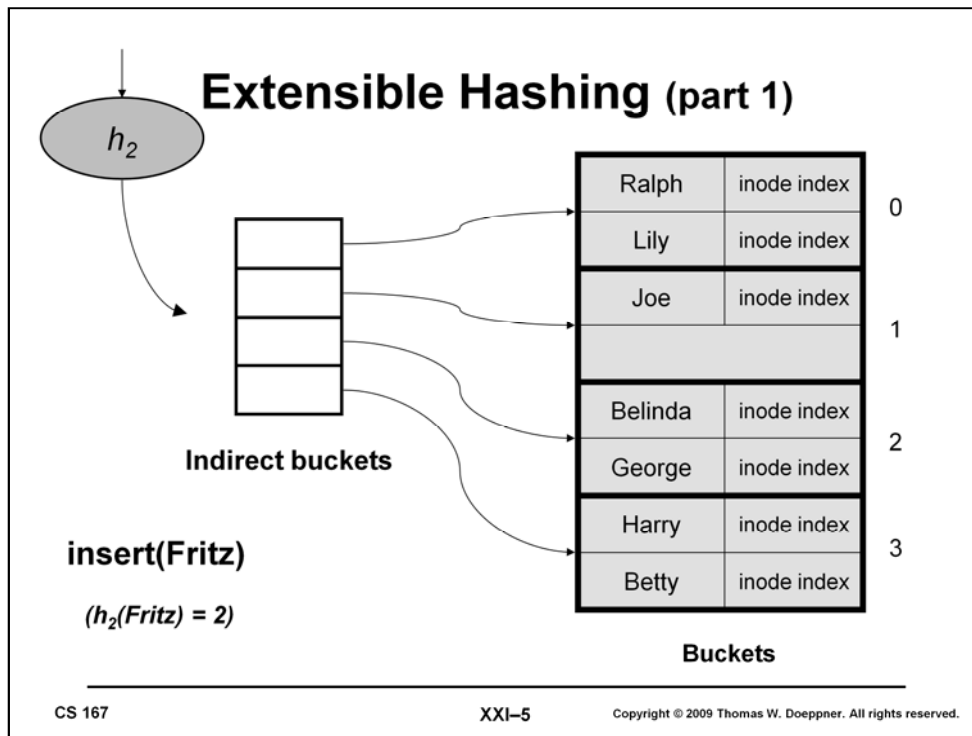


Directory Block

FFS allows component names of directory path names to be up to 255 characters long, thereby necessitating a variable-length field for components. Directories are composed of 512-byte blocks and entries must not cross block boundaries. This design adds a degree of atomicity to directory updates. It should take exactly one disk write to update a directory entry (512 bytes was chosen as the smallest conceivable disk sector size). If two disk writes are necessary to modify a directory entry, then clearly the disk will crash between the two!

Like the S5FS directory entry, the FFS directory entry contains the inode number and the component name. Since the component name is of variable length, there is also a string length field (the component name includes a null byte at the end; the string length does not include the null byte). In addition to the string length, there is also a record length, which is the length of the entire entry (and must be a multiple of four to ensure that each entry starts on a four-byte boundary). The purpose of the record length field is to represent free space within a directory block. Any free space is considered a part of the entry that precedes it, and thus a record length longer than necessary indicates that free space follows. If a directory entry is free, then its record length is added to that of the preceding entry. However, if the first entry in a directory block is free, then this free space is represented by setting the inode number to zero and leaving the record length as is.

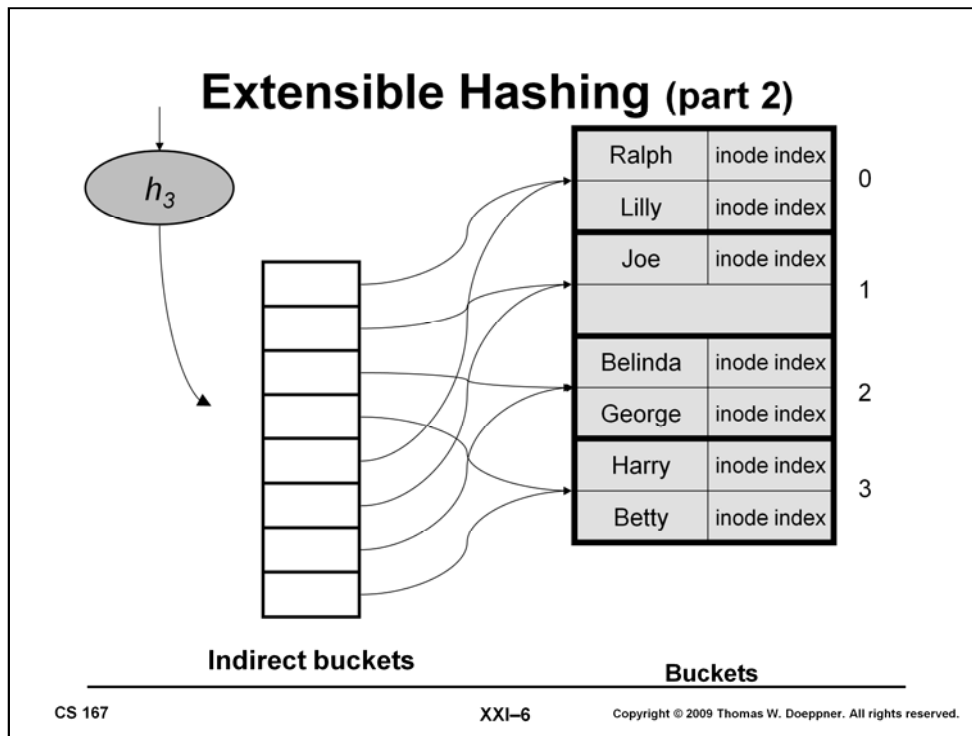
Compressing directories is considered to be too difficult. Free space within a directory is made available for representing new entries, but is not returned to the file system. However, if there is free space at the end of the directory, the directory may be truncated to a directory block boundary.



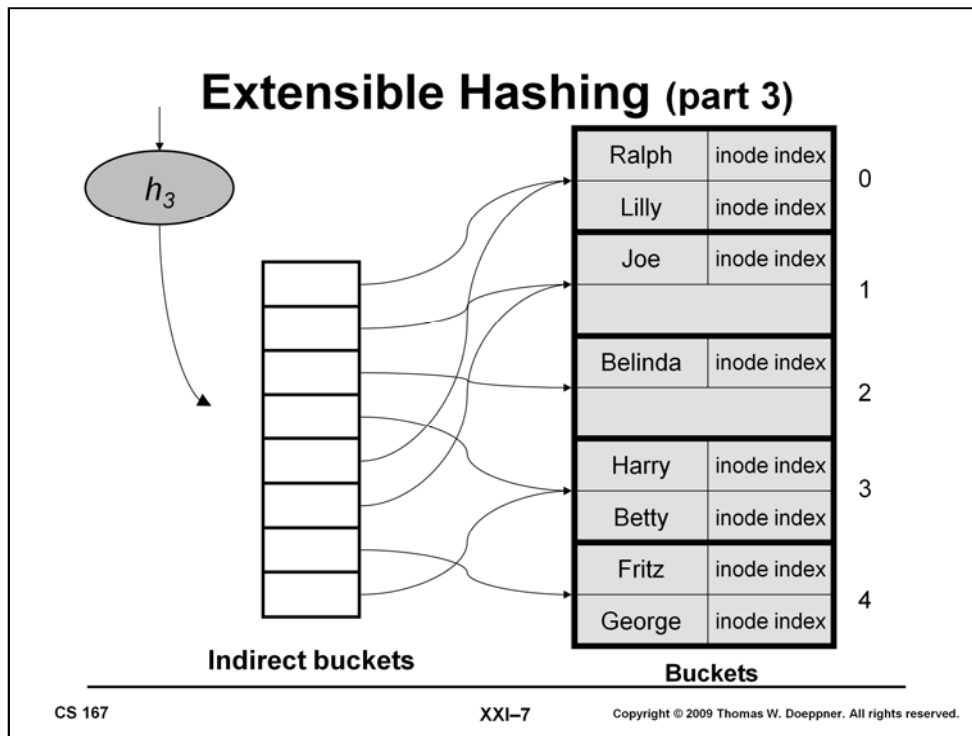
See the text, Section 7.3.1.1, which describes the use of extensible hashing for directories. h_1 is a hash function that maps its arguments into 2^i buckets. In this example, each bucket is of size 2.

Here we have four buckets and hence are using h_2 , which actually computes indexes into the array of indirect buckets, which, in turn, lead to the appropriate bucket. Each of our buckets holds two items. We are about to add an entry for Fritz. However, $h_2(\text{Fritz})$ is 2 and the bucket it leads to is already full.

Sun's ZFS file system uses extensible hashing.

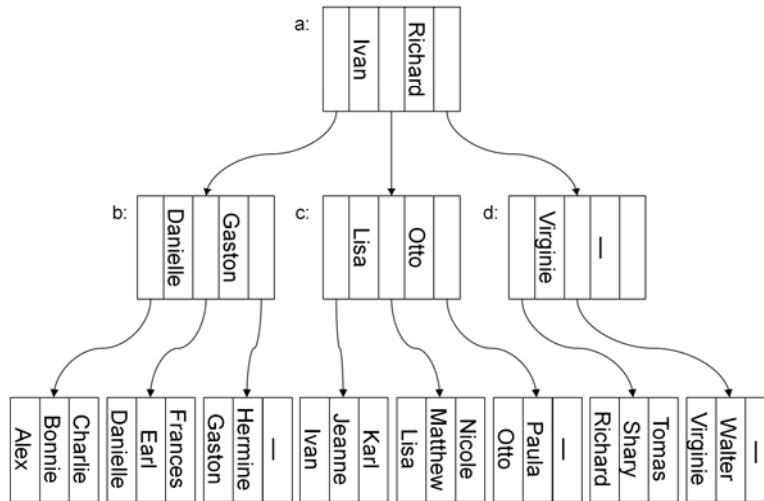


Here we've switched to h_3 , which maps names into eight buckets. However, rather than double the number of buckets and rehash the contents of all the old buckets, we take advantage of the array of indirect buckets. We double their number, but, initially, the new ones point to the same buckets as the old ones do: indirect buckets 0 and 4 point to bucket 0, indirect buckets 1 and 5 point to bucket 1, and so forth.



For the sake of Fritz we add a new (direct) bucket which we label 4. We rehash Fritz and the prior contents of bucket 2 under h_3 , with the result that Fritz and George end up in the bucket referred to by indirect bucket 6, while Belinda stays in the bucket referred to by indirect bucket 2. Thus, we set indirect bucket 6 to refer to the new bucket 4. If, for example, we add another name that would go into bucket 0, we'd have to add another bucket to hold it and rehash the current contents of bucket 0.

B+ Trees (part 1)

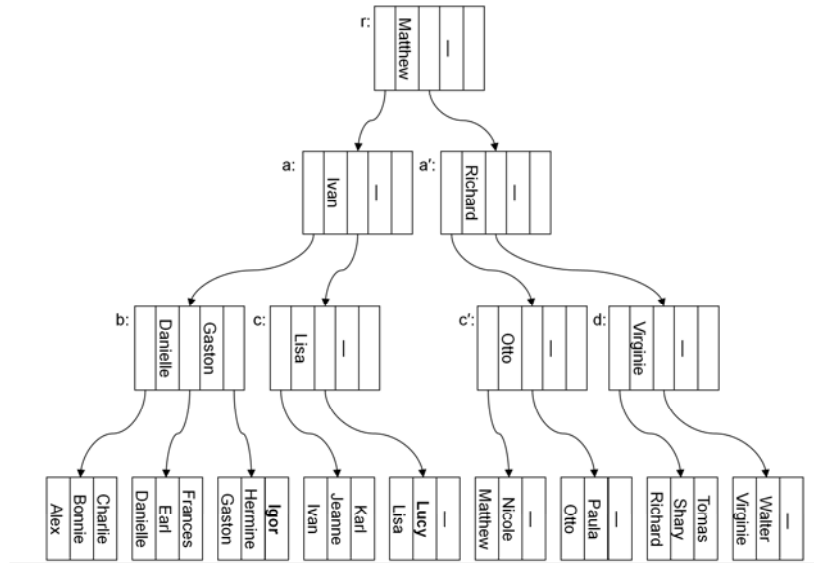


A B+ tree representing a directory. To keep the figure relatively simple, all entries occupy the same amount of space.

See the discussion in the text starting at page 7-54.

Microsoft's NTFS uses B+ trees.

B+ Trees (part 2)



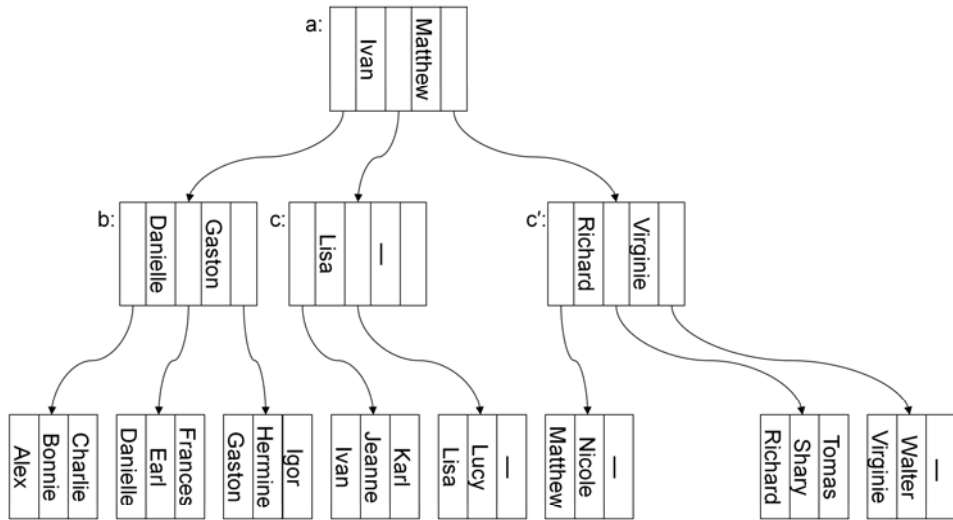
CS 167

XXI-9

Copyright © 2009 Thomas W. Doepfner. All rights reserved.

The result of adding entries for Igor and Lucy to the example of the previous slide.

B+ Trees (part 3)



The directory after removing Paula and Otto.

Summing Up File Systems

- **What's important**
 - space
 - speed
 - reliability (with respect to hardware/software failures)
- **Examples**
 - S5FS
 - FFS (\approx ext2)
 - ext3
 - NTFS
 - WAFL
 - ZFS

Databases vs. File Systems

- **Why not implement file system on top of a database?**
- **Why not implement a database on top of a file system?**