Operating Systems: ENGR 3950U

Course Project Presentation

Simulated Unix File System

Faculty of Engineering and Applied Science
University of Ontario Institute of Technology

November 30, 2012

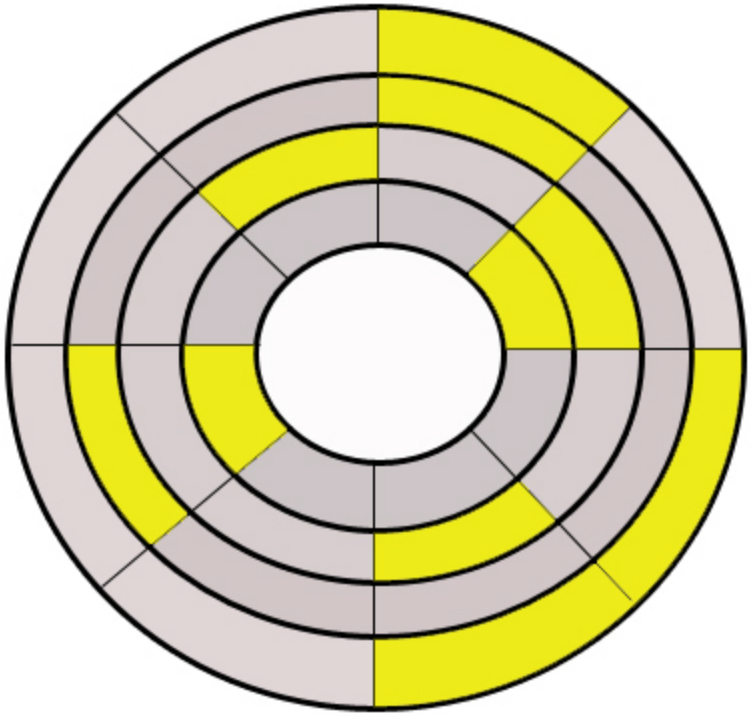
***Jonathan Gillett***
***Joseph Heron***
***Daniel Smullen***

# SneakyFS

cs-club.ca/sneakyfs/

# What is a File System?

- Abstraction from disk sectors to blocks
- Abstraction from blocks to files
- Files are accessed by OS



# Comparing Different File Systems

## WINDOWS

### FAT32

- File Allocation Table
- Simplistic, carryover from FAT16
- Address drives with letters
- Address directory tokens separated by \
- File extensions:
- *.exe *.doc

### NTFS

- File ID
- Journaling

## UNIX/LINUX

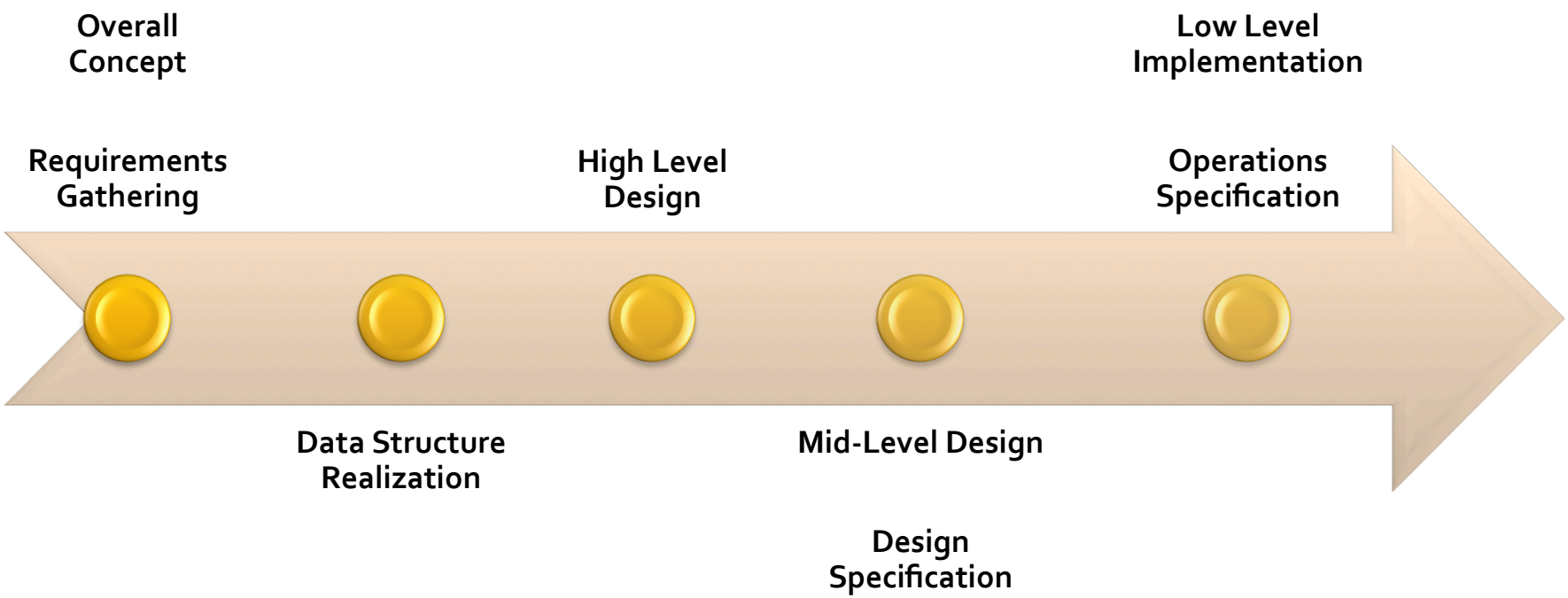
### Ext2, Ext3, Ext4

- Different features per iteration
- Address everything based on root: /
- Directory and file tokens
- chmod
- POSIX

It all comes down to METADATA and DATA STRUCTURES.

*More on this later…*

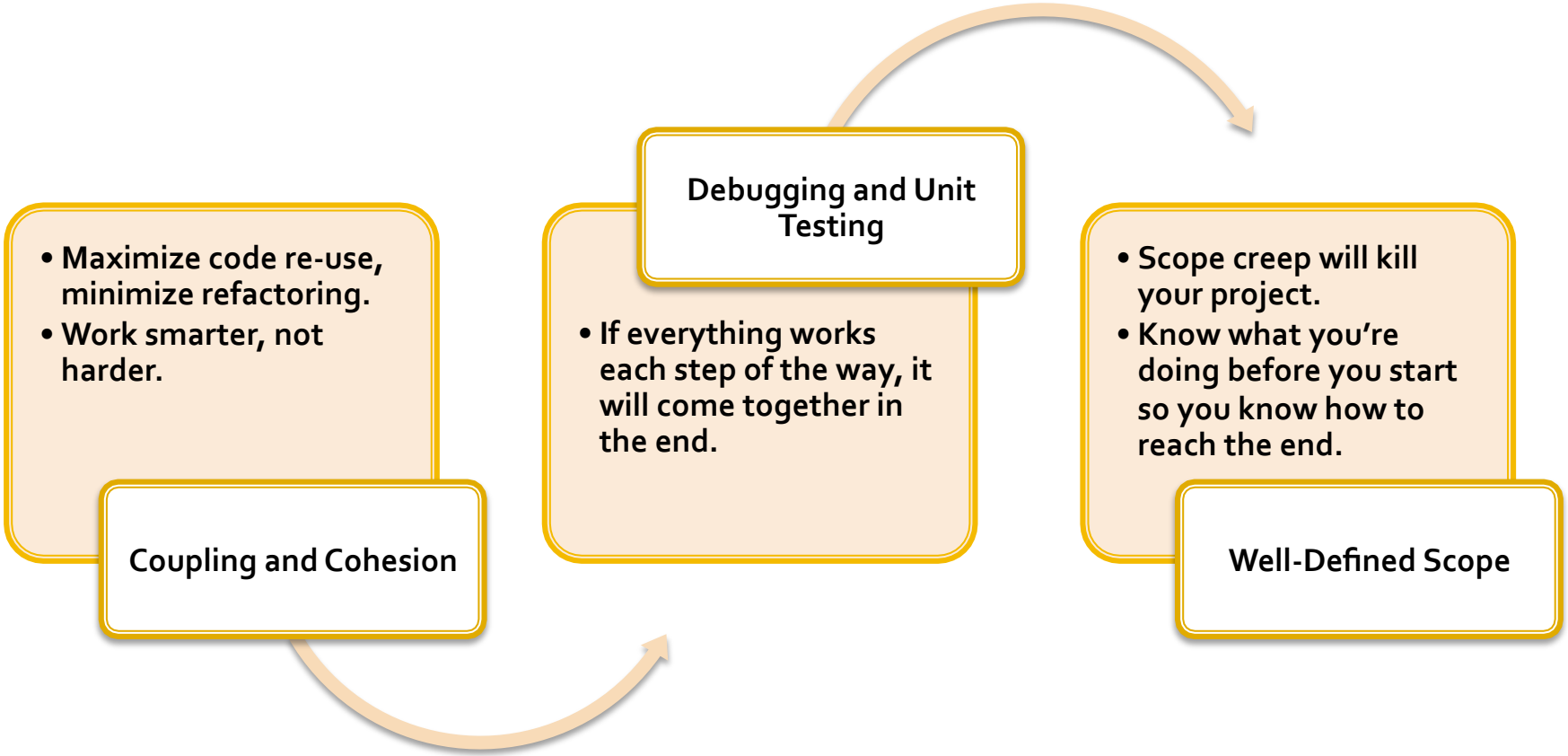
# Designing a File System



# Plan to Succeed


**Great Success!**

# Design Phase vs. Rapid Application Development

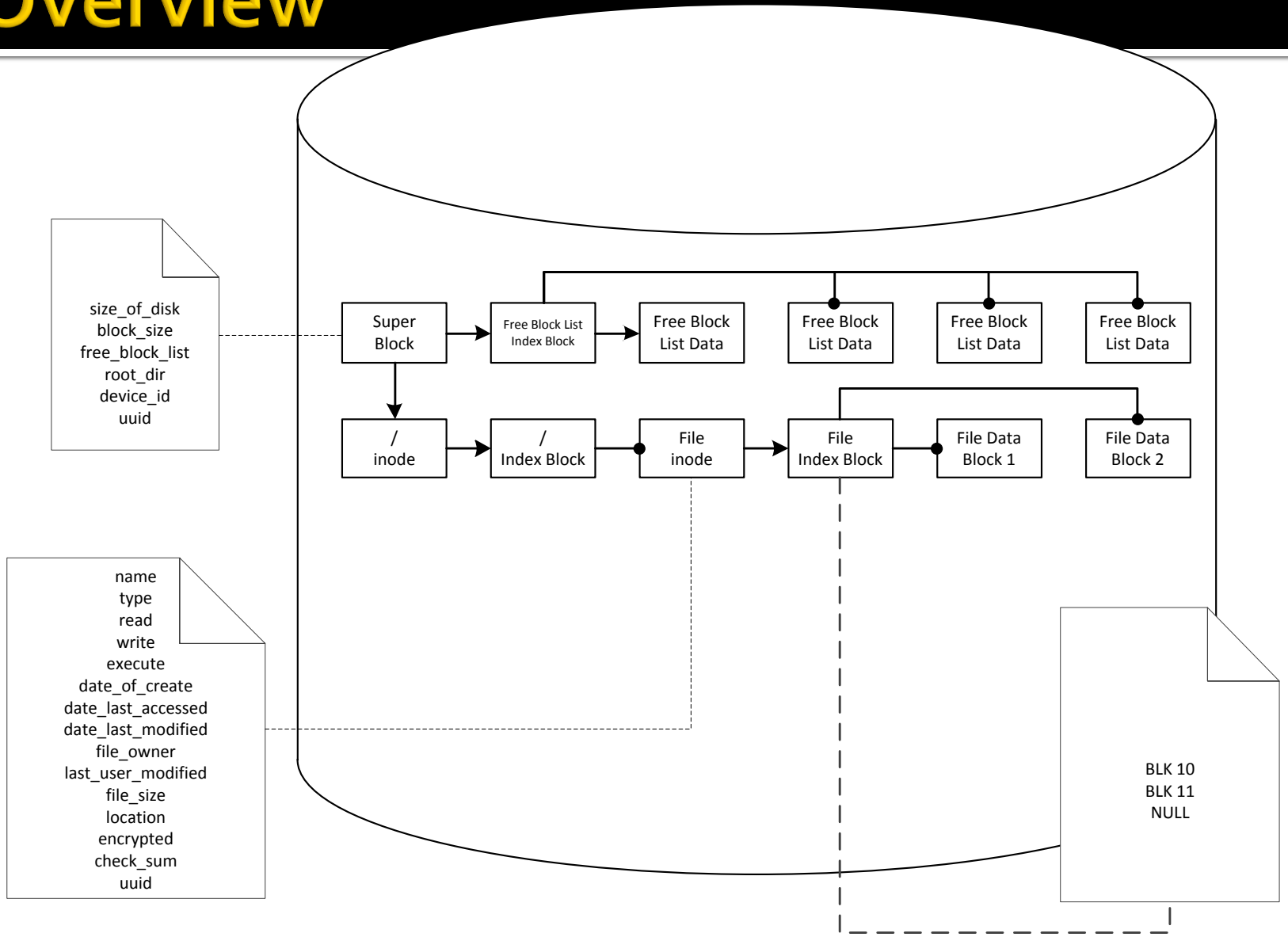


# SneakyFS
# High Level Design Introspective

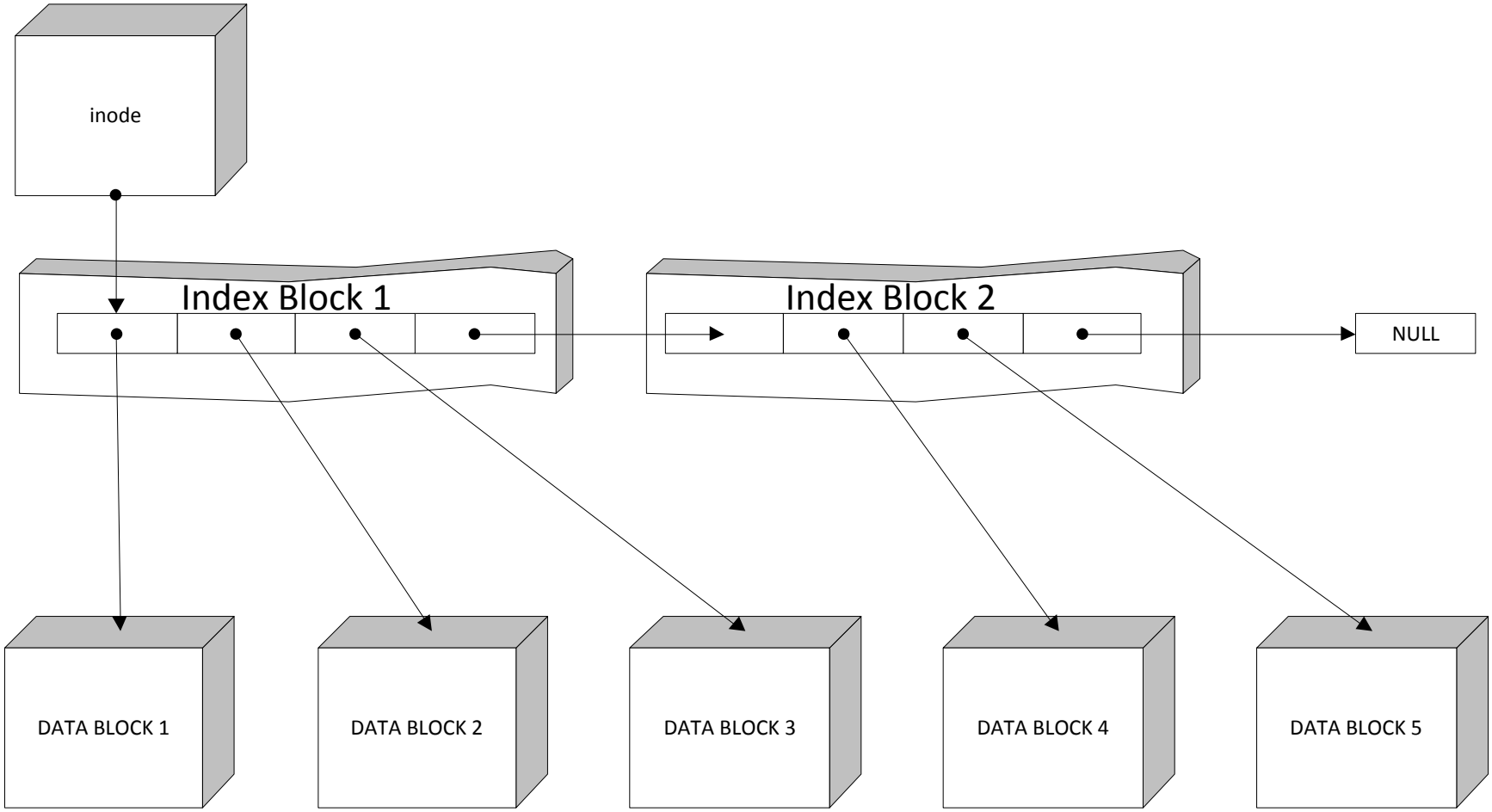
# Key Features

- Unique Allocation Structure
- Journaling Using Copy-On-Write
- Supports Transparent File System Encryption
- Object Oriented Design Principles Implemented Using C
- Universal Unique Identifiers

# Hybrid Allocation Structure Overview



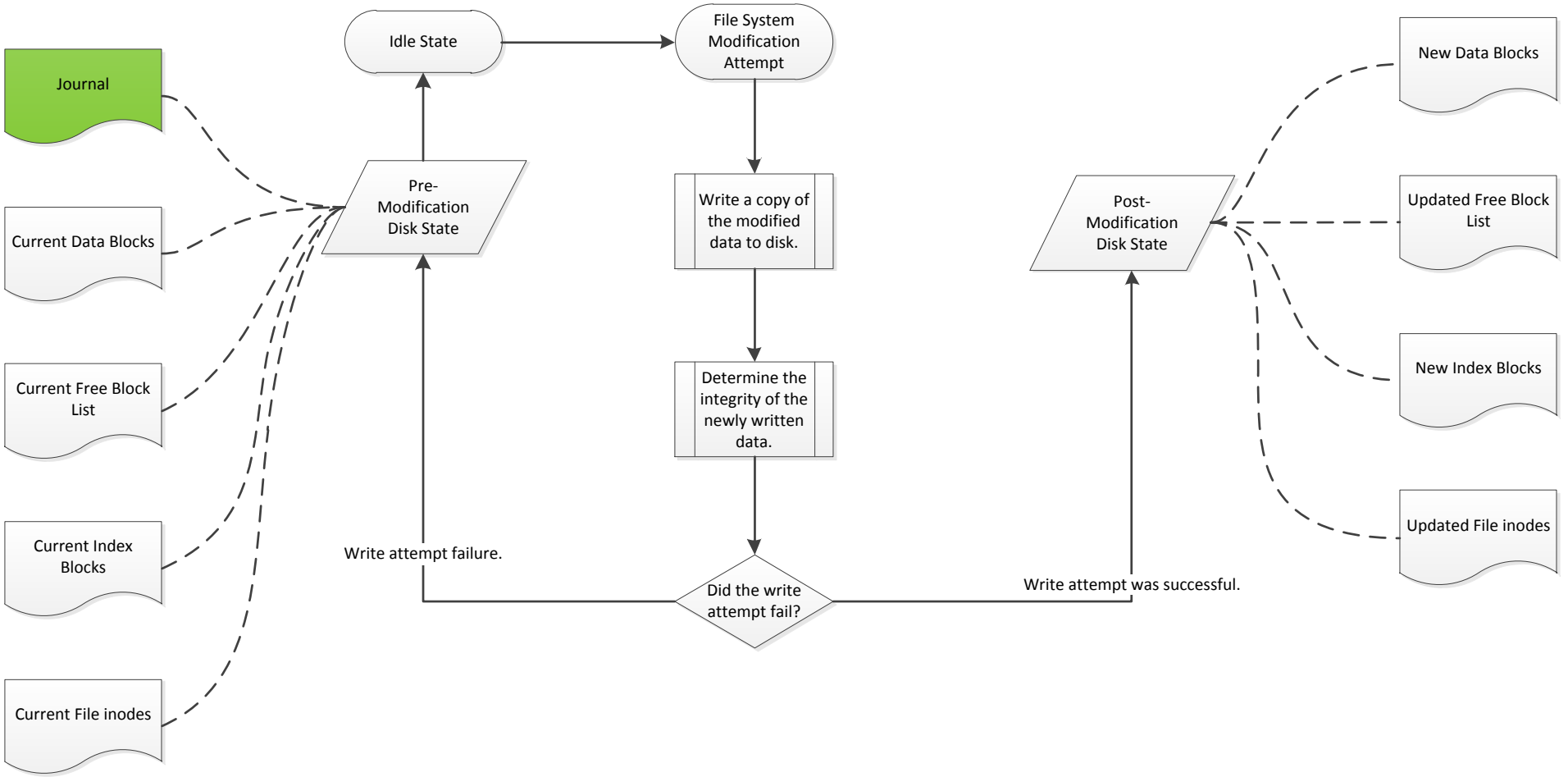
# Hybrid Allocation Index Data Structure



# Journaling – What is it?

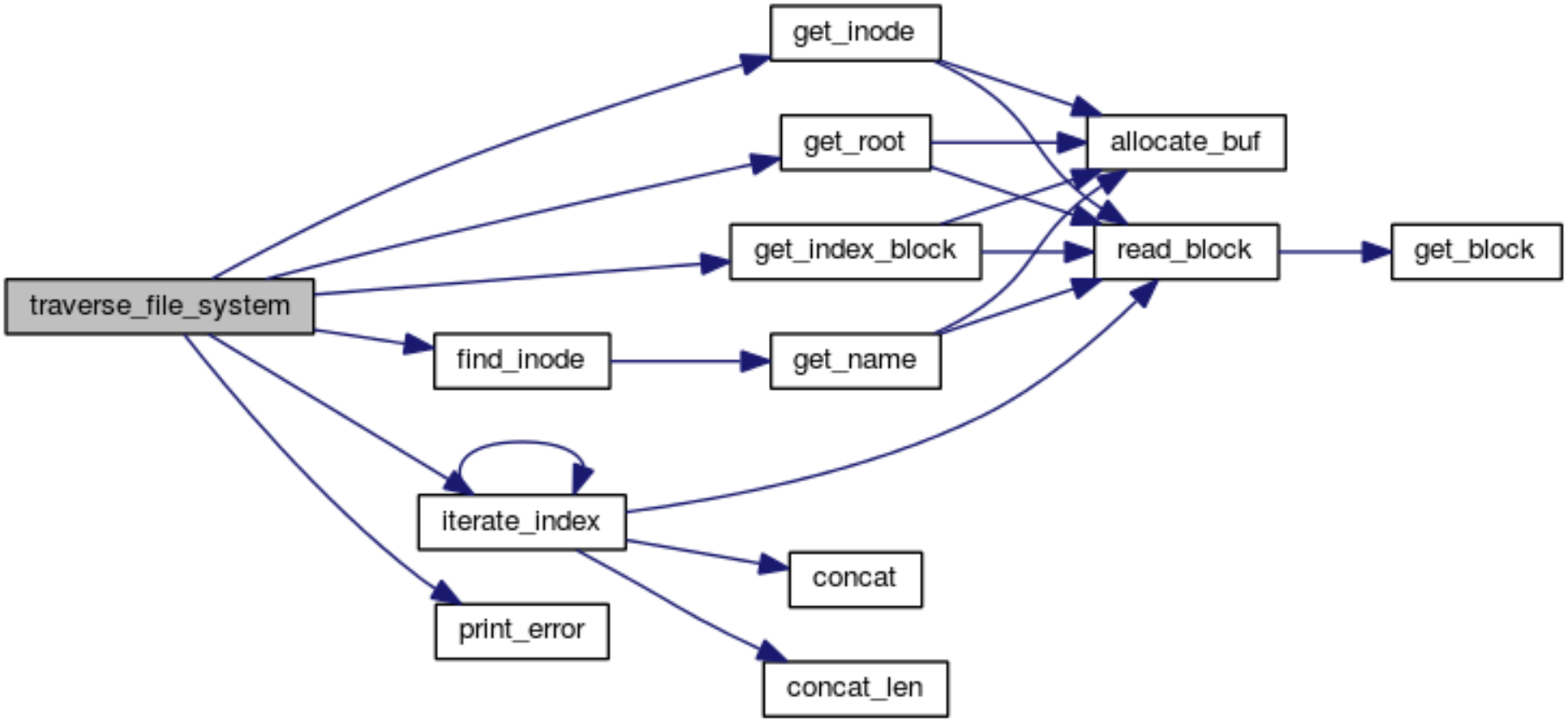
- Prevents data loss.
- Maintains safe state.
- Various methodologies:
  - Physical Journals
  - Logical Journals
  - Write Cache
- Alternative Methodologies…

# Journaling Using Copy-On-Write



# SneakyFS
# Low Level Design Introspective

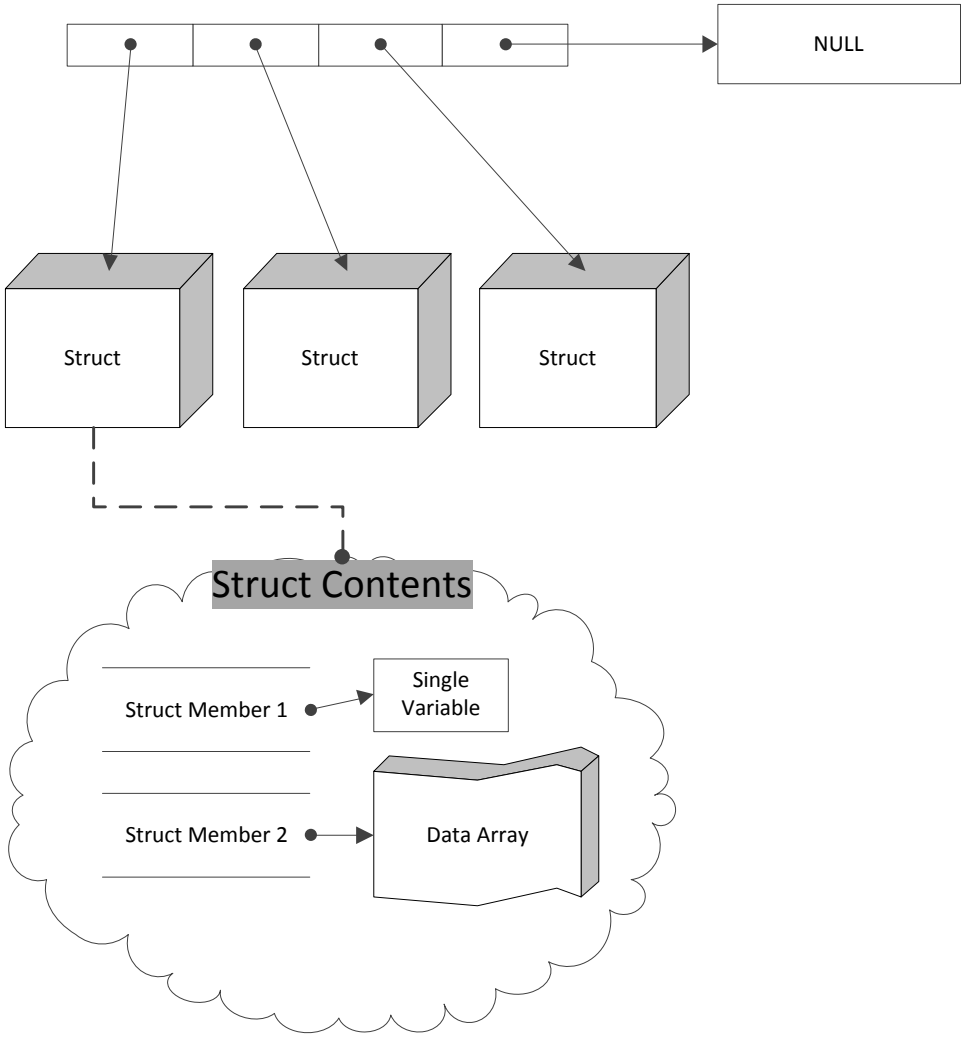
# Functional Design for C Call Graphs



# "Object Oriented" Design Using C

- C doesn't support classes.
- C doesn't support array return types.
- C does not support information hiding.
  - So what does C support?

# Object Encapsulation Using C Data Structures



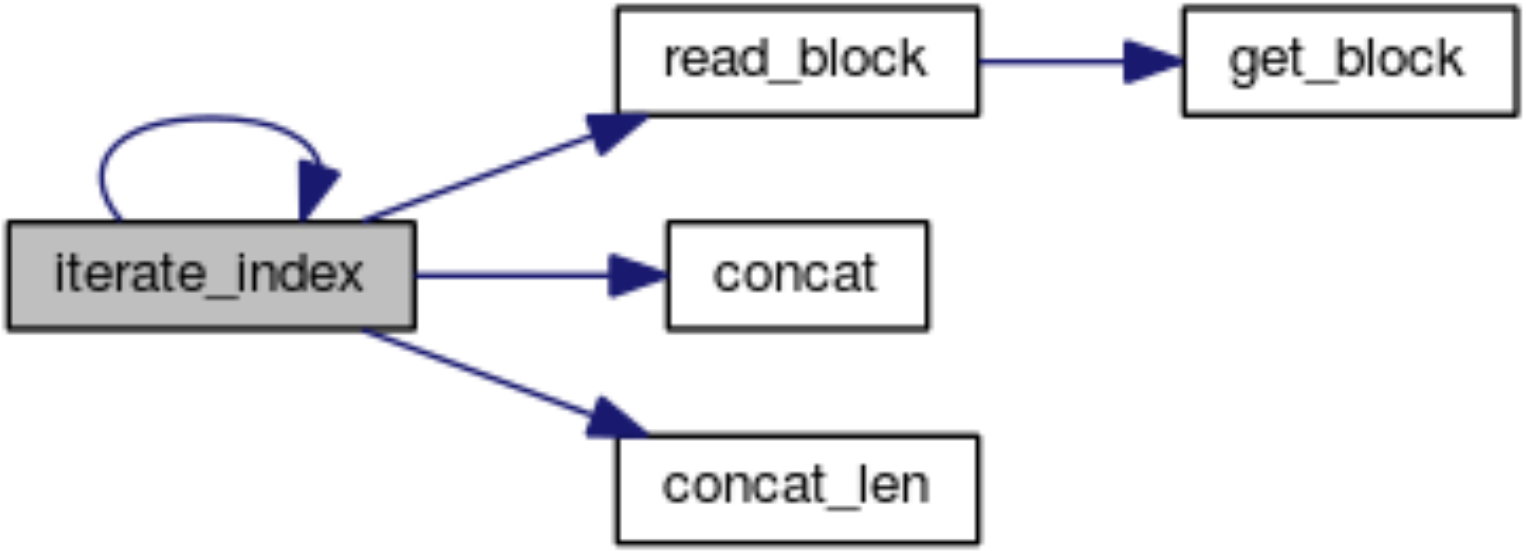

*This technique was used in the design of SneakyFS with respect to the free block list data structure, the inode data structure, the system wide open file table, and the super block data structure in the source code.*

# Function Pointers and Visibility in C

- No equivalent entity to classes.
- Interface objects are impossible.
- Information hiding limited to module scope - .c files and .h files
- Logical basis for visibility.

*This technique was used within the unit testing suite in the source code.*

# Using Recursion




- Used to traverse index block data structures, and load the contents into memory.
- Uses a second argument to carry data through the recursion tree.

# Generic Programming I

- C has only static types:
  - Every variable must be assigned a type.
- Void pointers:
  - Will take any data type!

*This technique was used as part of the library of global internal functions within glob_func.c*

# Generic Programming II

```
void* concat(void* src_1, void* src_2, uint32_t size)

<SNIP>
/* Get the number of items in each array using the size as an index multiplier.
           * In order to check for NULL have to verify that all bytes given the size of data
           * contain NULL, for example the NULL character for uint32_t is 00 00 00 00
           */
          if (_src_1 != NULL)
          {
                    while (true)
                    {
                              num_null = 0;

                              for (uint32_t k = 0; k < size; ++k)
                              {
                                        if (_src_1[(i * size) + k] == NULL)
                                        {
                                                  ++num_null;
                                        }
                              }

                              /* Reached the NULL character for data type */
                              if (num_null == size)
                              {
                                        break;
                              }

                              /* Increment counter for number of items in array */
                              ++i;
                    }
          }
<SNIP>
```

# Universal Unique Identifiers

- Distinguish data structures by some means other than name and temporal attributes.

*This technique was used within the super block data structure, and the inode data structures.*

# Design Statistics

- Maximum Disk Size Supported

$$2^{32} \times BLKSIZE = 4294967296 \times 128\ blocks$$
$$= 549755813888\ bytes$$
$$\approx 500\ Gbytes$$

(Default Block Size 128 bytes)

- Maximum Disk Size Supported

$$2^{32} = 4294967296\ \frac{bytes}{block}$$
$$4294967296\ blocks \times 4294967296\ \frac{bytes}{block}$$
$$= 18446744073709551616\ bytes$$
$$\approx 17\ exabytes$$

(Maximum Block Size 4294967296 bytes)

# Maximum File Size

The number of index blocks required to address a file's data blocks:

$$= \frac{FILESIZE}{BLKSIZE} - 1$$

Free block list data structure overhead:

$$= \left\lceil \frac{NUMBLKS}{BLKSIZE} \right\rceil + 1\ blocks$$

Using the default file system values of 512 and 128 for the number of blocks and block size, respectively, this yields a minimum overhead on disk of:

$$= \left\lceil \frac{512}{128} \right\rceil + 2\ blocks$$
$$= 1\ block_{[Free\ Block\ List\ Index]} + 4\ blocks_{[Free\ Block\ List]} + 1_{[Super\ Block]}$$
$$= 6\ blocks$$

Using the example of the default file system values of 512 and 128 for the number of blocks and block size respectively, this yields a maximum file size of:

$$512\ blocks$$
$$-6\ blocks_{[File\ System\ Overhead]}$$
$$-3\ blocks_{[Index\ Structure\ Overhead]}$$
$$-1\ block_{[inode]}$$
$$= 502\ blocks$$

SneakyFS

# Demonstration

# Retrospection I

- Application of knowledge from all previous course materials
- Extensive working knowledge of underlying computer processes and data structures
- Realistic project goals, and how to achieve them
- Team strengths, weaknesses, dynamics, leadership, making it work under pressure

# Retrospection II

- Design realization, practice, implementation
- Requirements specification
- Pair programming, extreme programming
- Collaborative development using Github, meld
- **Documentation**
- **LaTeX**