Walnut Kernel

A Data Object Store for the Walnut Kernel

Supervisors: Ron Pose and Carlo Kopp

Stewart Smith
sesmith@csse.monash.edu.au

To be covered

- Background
  - Existing Data Storage
  - Walnut Data Storage
  - Consistency
- Aims
  - Approaches to implementation
- Status

Skip quickly over, not really very interesting

Background

Data Storage

- File Systems
  - FFS, LFS, UFS, FAT, NTFS, ext, ext2, ext3, reiserfs, xfs, jfs, HFS, HFS+, BeFS, WinFS
- Databases
  - Relational, Object

File Systems: file & directory concept. Users familiar with. implemented in many different ways.
BeFS and WinFS stand out as being a little different: BeFS has indexes of attributes on FS objects.

Databases have imposed strict structure on data, not too useful as generic data storage systems.

What Is Walnut?

- A persistent, password-capability Operating System
  - Memory is a cache for on-disk objects
- Developed at Monash
- Runs on PCs
- Messy code, cool ideas

Persistence: EVERYTHING is preserved across reboots
i.e. “Save” is no longer needed, it’s automatic. But with versioning support, you can backtrack :)

Walnut Data Storage

- Object = File, referenced by globally unique ID, not a name
- Everything on Walnut is an object
  - Processes
  - Data
- concept of filenames/directories
  - userspace nameserver (**not** kernel)

object: referenced like inode on unix
nameserver similar to DNS system. give it a name, it gives you an ID.

Walnut Data Storage

- Objects are mapped into a processes address space
- there are **no** read() or write() syscalls
- all IO is done on objects in memory, via CPU instructions
- Periodically, data is flushed to disk

File IO vs LOAD/STORE

Comparison

UNIX

```
FILE *a;
char buffer[100];
a = fopen(”blogs”,”r”);
fgets(a,buffer,100);
```

Walnut

```
char *buffer;
buffer = load_cap(cap);
```


really a great big lie of a mock up :)

What does Walnut Store?

- Object contents and kernel internal data
  - Attribute => Data
- One large attribute, rest are meta-data
- I won't worry about details
  - assume an object is several attributes

Consistency Example: A simple Program

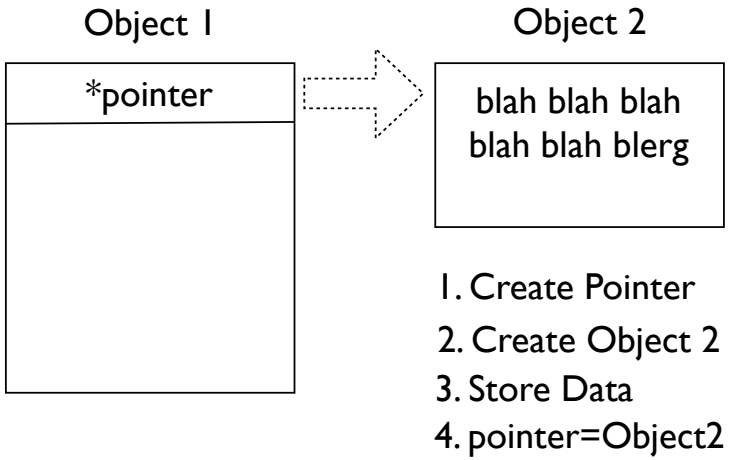
1. Create Pointer
2. Create Object 2
3. Store Data
4. pointer=Object2

This is purely a conceptual model, it is not a reflection on how Walnut processes should (or can) do things.

Scenario

- Program running
- Power Failure
- System restarted
- Program resumes where it left off
  - ‘cause we’re persistent!

Inter-object Relationships



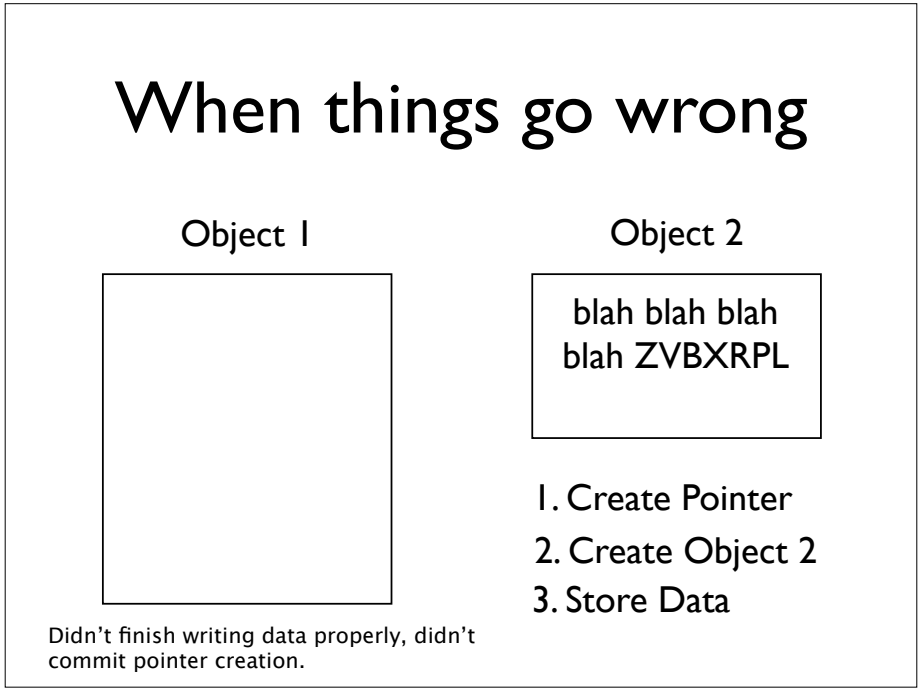
1. Create Pointer
2. Create Object 2
3. Store Data
4. pointer=Object2

this will appear step by step

What could go wrong when restoring?

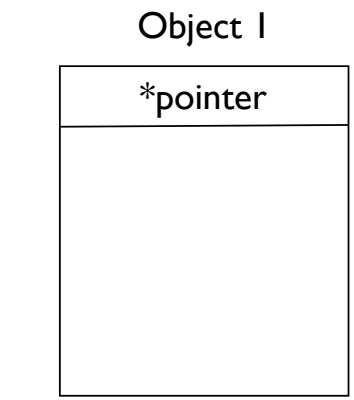
- Operations committed out-of-order
  - Image on disk won't match an active state

i.e. what's on disk never happened.



didn't finish writing data properly, didn't commit pointer creation.

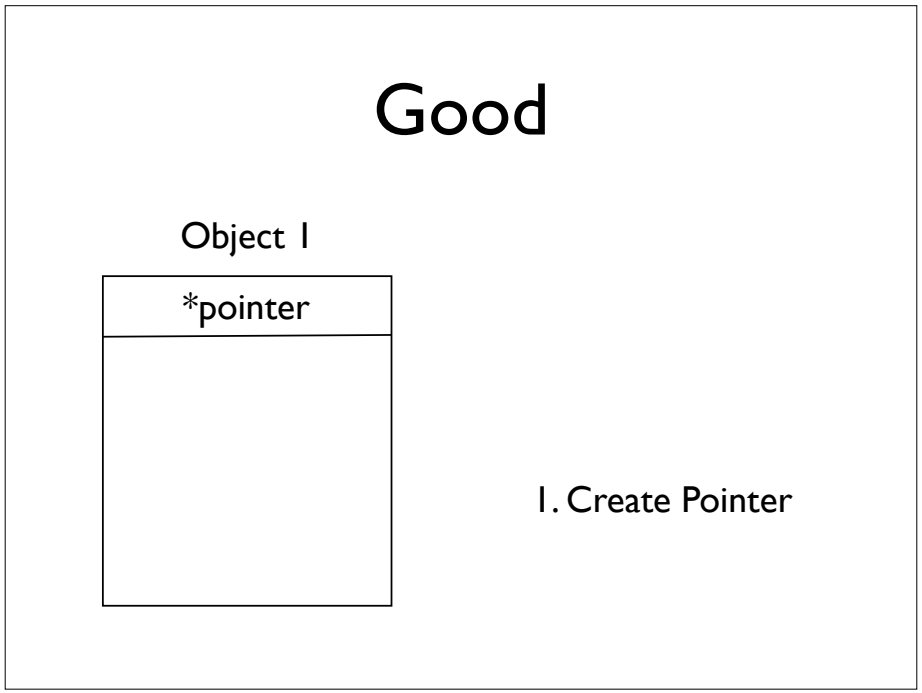
Worse



1. Create Pointer
2. Create Object 2
3. Store Data

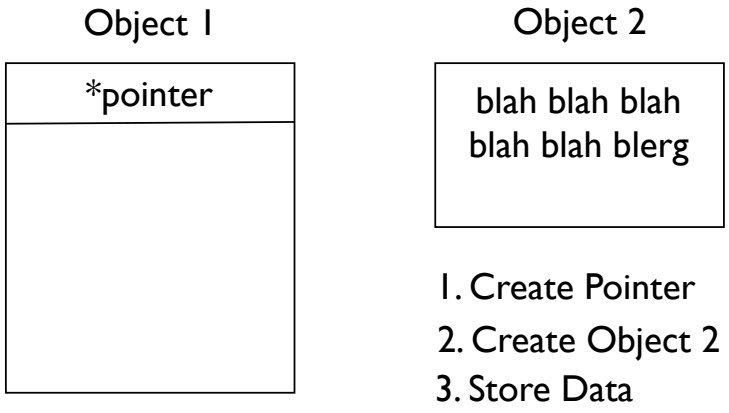
didn't commit create object. didn't commit store data

didn't commit create object. didn't commit store data



We can at least continue to work, even though we've lost a bit of work, we can recover it.

Even Better



1. Create Pointer
2. Create Object 2
3. Store Data

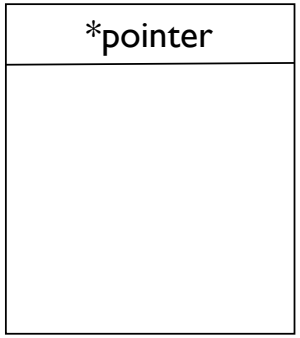
We've got more work saved, less we have to redo.

Consistency

- Order of objects being flushed matters!
- Inconsistency hard to detect, easy to notice
  - i.e. it screws up

How would you detect this? (hint: you can't)

Object 1



1. Create Pointer
2. Create Object 2
3. Store Data

a process would have no hope of working out that this happened **between instructions.**

a process would have no hope of working out that this happened **between instructions.**

Walnut Data Store Requirements

- **MUST** be able to be a primary data store
- **MUST** be data consistent after crash
- **MUST** deal with inter-object dependencies

- This is what I'm aiming to design and implement

Aims

To implement an Object Store for the Walnut Kernel, which ensures consistency of objects and allows for revision tracking.

- Design down to on-disk format
  - How things appear on disk
  - block-by-block
- Include "policy" on use of on-disk format
  - how to manipulate on-disk structures

Testing

- Simulate Kernel environment in user space
  - file as block device
  - Easily movable to kernel
- Move to raw block device & kernel
  - hopefully with Linux FS interface (for testing)

Linux FS Interface

- Purely for testing. i.e. a hack.
  - use a walnut object to contain directory listing
  - much like traditional UNIX FS and how a Walnut nameserver may work.

Directory Object (id=100)

Name	Object ID
.	100
..	57
buffy	105
willow	1

Example Directory Object (this would be a directory file in a traditional unix system.)

Approaches

Bad Solutions

- Update disk after every **CPU** instruction
  - slow
- Halt system while dirty objects are flushed
  - **everything** stops for a few seconds
- Buy a UPS and shutdown cleanly

Possibility: Transactions

- Databases use them to ensure consistency
- simple concept (begin + rollback || commit)
- Either completes successfully or not at all
- used internally in modern filesystems
- On main memory?

how do we define a programmers interface to transactions on main memory? Do we force explicit calls?

Could look at it as the “flush to disk” is a transaction.

BSD FFS Soft-Updates

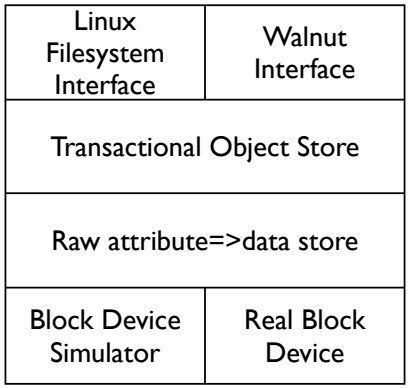
- Tracks meta-data dependencies
  - alternative to journalling
- Carefully orders commit to disk
  - ensure consistency
- Theory possibly quite useful for Walnut

Revision Tracking

- RCS/CVS style! (or a simplification)
- Possible by-product of transactions
- Useful when dirty objects exceed phsyical RAM
  - CONSISTENT revision tags
- Could have security issues in Walnut

Especially useful when we have more dirty objects than RAM, we have the last consistent version tagged as CONSISTENT and our current ‘dirty’ version tagged as non-consistent so that in the event of a crash, the consistent one is restored.

Status



Green: mostly implemented
yellow: designing.
Walnut interface: not going to worry about (yet)
real block device: they exist

Status

- Started design document of disk format
  - More notes that require formalizing.
- Have working simulator of Linux buffer cache (block device interface)

Answers?

(I refuse to have a “Questions” slide :)