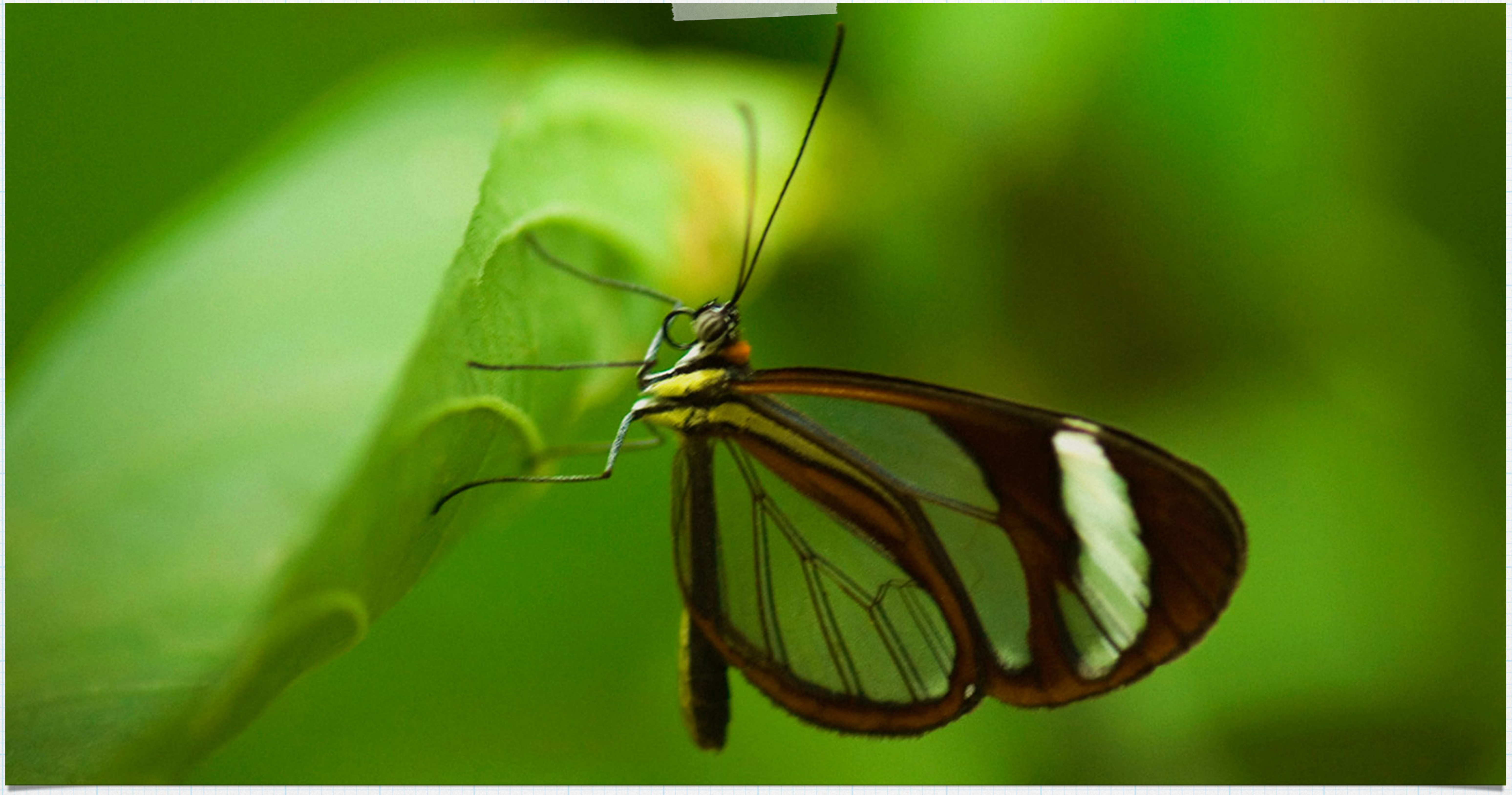

# Making Kernel Objects Movable

A History and a way Forward

Christoph Lameter, cl@linux.com, linux.conf.au 2017 Hobart

# Memory Management Basics

* Linux Memory Reclaim is fundamental to Linux
* The ideal of 4K size (a page) allocation / freeing and reuse
* No fragmentation because we aways allocate and free the same sized memory.
* But then there are allocations not 4k to deal with.

## Slabs: The need for smaller objects

* First a K&R allocator (SLOB)
* Then the Solaris SLAB approach
* Then SLUB page fraction based approach

* Acquire memory in pages from page allocator
* Internal fragmentation. Sparsely allocated blocks
* Inode/dentry reclaim and its issues

## And the need for larger objects

* 4k size is tiny if i want to send 4GB of data.... 1 mio 4k blocks have to be managed to lets say write file. No way that this could be fast.
* Hardware needs to track 4k fragments. Massive Scatter Gather lists. Device drivers manage state of 4k chunks.
* TLB coverage issues.
* Pointer issues
* Subsystems work faster when large sizes are available.
* Some hardware requires large contiguous memory (nics, graphics, embedded devices etc)
* File sizes are increasing, I/O devices performance are increasing but the page size stay the same. Result massive amounts of mangement overhead in th kernel.

## Multiple sizes required!

No way around the fragmentation problem.
Memory gets garbled.
Gaps wasted that are not fit for further allocation.

# Remedial Techniques

- Fragmentation Avoidance
  - SLAB
  - Page allocator
- Huge Pages / Giant Pages
- Preallocation of memory on bootup
- Page Based Defragmentation

## Defragmentation requires

* Ability to rearrange objects in memory in such a way that large contiguous sections of memory become available.
* Rearrangement requires the ability to MOVE objects if they are in the way of creating contiguous memory.
* But we can only move user space pages these days.

# Existing Defragmentation Logic

* Works based on page migration which was introduced to move user space memory between NUMA nodes
* Works only on user space pages where the kernel can track all current users of the page.
* Pages may be pinned by kernel or device driver (f.e RDMA subsystem is doing this in a large scale)
* Kernel pages (slab memory) are not migratable and are mixing freely with user space pages.

## Existing APIs to move objects

* System calls for user space pages
* Address space operations have callbacks
* Main problem is kernel memory in particular Inode / Dentry caches for Filesystem metadata

# Challenges in Making Kernel object movable

* Arbitrary references may exist to the object. Difficult to track these references in particular if only a ref count exists.

* Complex allocation / freeing techniques in use

* Seems like a daunting task (But then page migration was considered to be impossible for a long time ...)

# Solutions

* References: Do an optimistic attempt to track down all references. Do not migrate if all or not accounted for (that is what page migration does)

* Callbacks for the Slabs in the same way that the address space operations provide? Tracking down of references would then be subsystem specific and needs to be provided when new kernel objects types are created.

* Garbage collection: Require all pointers to be specially marked and provide generic methods to recover memory.
* Problem with garbage collection times: But then we already have these reclaim passes that cause long delays.
* Relates to many other techniques in the kernel.

## Step by Step (like page migration?)

* First allow the removal of an object from memory and the restoration from storage (possible for inodes and dentries the biggest problems). Requires only the ability to remove references to an kernel object.
* Then improve the process by performing direct copies without going through a medium.
* Enhance performance by exploiting the fact that we do not hit storage. Move pointers instead of invalidating them etc.

## What now?

## Questions and Opinions?

# Callbacks

- We want both reclaim as well as object migration.
- First step: Exempt the object from freeing
- Page migration uses isolate_lru():
  `void *isolate_object(struct kmem_cache *, void **objects, int node)`
- Reclaim/Move object:
  `void migrate_objects(kmem_cache *, void **objects, int nr, int node, void *private)`
- object state must always be defined. Thus a ctor is required to bring objects into a defined state upon creation. requirements are like for RCU slabs.